TECHADYANT RESEARCH

THE END OF THE APPLICATION ERA

HOW AGENTIC AI COULD FORCE THE FIRST MAJOR OPERATING SYSTEM REDESIGN SINCE CLOUD COMPUTING (2026–2035)

WORKLOAD SHIFT
FROM APPS TO AGENTS

SECURITY INVERSION
FROM USERS TO CAPABILITIES

MEMORY PRIMACY
CONTEXT IS THE NEW COMPUTE

GEOPOLITICS OF COMPUTE
SOVEREIGNTY WILL DEFINE PLATFORM POWER

ECONOMIC TRANSFORMATION
NEW VALUE CHAINS.
NEW WINNERS.
NEW RULES.

A DECADE THAT WILL REDEFINE OPERATING SYSTEMS, SECURITY, HARDWARE, AND NATIONAL STRATEGY

2026–2035

OPERATING SYSTEMS • AGENT ARCHITECTURES • SECURITY • SEMICONDUCTORS
CLOUD INFRASTRUCTURE • SOVEREIGNTY • INDIA STRATEGY

PREPARED BY
TECHADYANT RESEARCH

STRATEGIC INTELLIGENCE REPORT
2026 – 2035

TECHADYANT

END OF THE APPLICATION ERA · The End of the Application Era · Technology Sovereignty Series

WHO CAPTURES COMPUTING WHEN THE APPLICATION DISAPPEARS?

How Agentic AI Forces the First Operating-System Redesign Since the Cloud — and Where India Can Capture the Next Layer.

Techadyant Labs

https://labs.techadyant.com/

Table of Contents

Executive Summary

Section thesis: The agentic-AI transition is not a simple application-layer upgrade. It is a force that exposes and amplifies mismatches between current OS abstractions and the workloads they will carry. The next decade will therefore see OS redesign, security inversion, semiconductor and packaging shifts, and a new platform-economics configuration—creating both risk and asymmetric opportunity, especially for India.

What This Report Covers

THE END OF THE APPLICATION ERA examine how operating systems, hardware, security, and platform economics are changing as agentic systems become load-bearing. It connects four layers:

1. Historical precedent: OS shifts are rare but consequential.
2. Architectural tension: Current abstractions were built for human-initiated, application-centric workloads, not continuous, tool-orchestrated, memory-heavy agents.
3. Industry reshaping: Hardware priorities, vendor concentration, and capital allocation are moving toward acceleration, memory bandwidth, and inference.
4. Strategic outcomes: India has a rare, time-sensitive window to participate via semiconductors, packaging, data infrastructure, sovereign software, and public-platform choices.

Five Core Arguments

Table 1 — Five Core Arguments.

Argument	Core idea
Application obsolescence	The application abstraction is becoming friction under agentic workloads.
Memory primacy	Context and memory are becoming OS-level primitives.
Security inversion	Trust will be governed by capability tokens and delegation policy rather than login sessions.

Argument	Core idea
India leapfrog potential	India does not need to dominate legacy OS markets to matter; it needs public digital infrastructure and sovereign compute.
Hybrid-kernel hegenomy	Near-term winners are likely to be hybrid kernels with agent-native layers, not pure component replacements.

Key Findings

- Operating-system dominance shifts roughly every one to two decades; each transition is triggered by workload rebalancing, not incremental feature growth.
- CPU-centric scheduling, human-centric security, and application-siloed state are now simultaneously mismatched with the dominant emerging workload.
- Memory, context capacity, and memory bandwidth are the critical resources for the next generation of systems.
- Existing security models are securing the wrong actors: human logins are being used by machine workloads.
- Accelerators, disaggregated memory, and advanced packaging create both dependency concentration and leapfrog opportunities.
- Sovereign-AI policies in the US, China, EU, and India are creating a new platform-economics layer where policy risk and capital risk are inseparable.
- India's public digital infrastructure stack and semiconductor expansion provide a credible, time-limited strategic advantage if matched with agent-aware design policy.

Strategic Implications

For policymakers, platform builders, and investors, the agent-native transition creates a narrow high-leverage window. The central risks are not only technical; they are institutional: governance models, standards bodies, talent pipelines, and sovereign supply-chain decisions made in 2026–2030 will constrain the architecture landscape through 2035.

Recommendations

5. Treat OS abstractions as policy-relevant infrastructure, not product-line decisions.
6. Invest in context-aware memory and scheduler research now, before hardware lock-in.
7. Replace identity-centric security planning with capability-governance and trust-boundary design.
8. Use sovereign computing programs as architecture policy, not just industrial subsidy.
9. Prepare India’s public digital infrastructure to support agent-orchestrated services at the same scale it previously supported human-directed ones.

Authors: Techadyant Research

Date: 2026-06-02

Scope: 2026–2035

Region focus: Global, with Part VI India strategic chapters

Part I — The Operating System Paradigm

Chapter 1 — A Brief History of Operating Systems

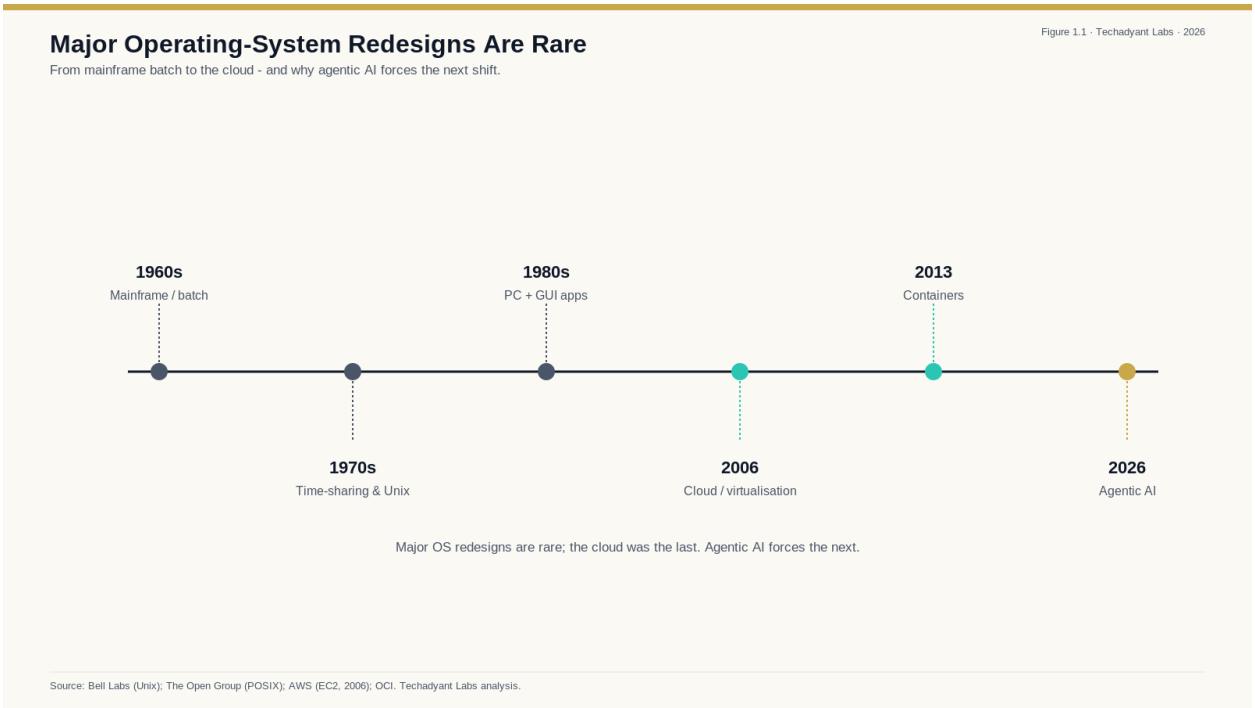

Figure 1 — Major operating-system redesigns are rare; the cloud was the last.

Thesis

Operating systems change not because computers get faster, but because stable human and workload abstractions shift; the transition from human-initiated app-centric work to agent-centric autonomous work is one such inflection point.

Mainframe Era: Batch to Time-Sharing

The first operating systems were not designed for individuals. They were dispatchers: batches of punch cards were queued, executed serially, and returned as printed output. The machine was a shared, expensive utility, and the OS was a polite traffic cop. Human time was cheaper than compute time, so throughput mattered more than responsiveness.

The shift came in the 1960s with time-sharing systems like MIT's Compatible Time-Sharing System (CTSS) and later Multics. Multiple users could interact with the machine simultaneously via terminals. The OS had to mediate not just jobs but *sessions* — logins, file

spaces, priorities, and isolation. For the first time, the user became a first-class abstraction inside the OS.

Evidence: IBM System/360 (1964) and its OS/360 (1966) cemented the idea of a multi-programming OS managing shared hardware for multiple concurrent processes [V2 — IBM archives].

The PC Revolution

The personal computer reversed the equation. It was a single-user, single-task machine by design. Early DOS systems had no real protection between user and hardware; you owned the machine when it turned on. The OS shrank to a loader and a filesystem.

The PC abstractions were minimal precisely because the machine was dedicated to one person. That assumption — *one human, one active session* — would later make it structurally difficult to retrofit continuous agent workloads into the same shell.

The GUI Turn

The 1980s brought the graphical user interface, standardized by Apple's Macintosh (1984) and then Microsoft Windows (1985 onward). The desktop metaphor made the computer legible: windows, folders, icons, menus. But it also locked in an interaction model based on *human-initiated, discrete, application-bound tasks*.

The OS became the stage; applications were the actors. Every piece of work was launched by a human click or keystroke, executed inside an app sandbox, and terminated when the human closed the window. That model is still the default.

Evidence: Windows 1.0–3.1 design docs and Apple Lisa/Mac HIG define the desktop metaphor as an intentional mapping of physical office objects to screen entities [V2 — vendor archives].

Networked OS and the Client-Server Era

With networking, the OS boundary expanded. The client machine retained the desktop metaphor, but the network became an extension of the filesystem and process space. NFS, SMB, RPC, and later HTTP gave applications remote reach.

The client-server era did not break the desktop metaphor; it just stretched it. The OS still mediated local hardware, while the network mediated trust and transport. Security models hardened around network boundaries, but the core abstractions — user, process, filesystem — were untouched.

Virtualization and the Cloud Abstraction

Virtualization (VMware, Xen, KVM) and then cloud computing abstracted the *hardware* away from the OS. The OS now ran on a synthetic machine managed by a hypervisor. AWS, launched in 2006, made remote compute an on-demand utility.

This was a fundamental shift: the OS was no longer the sole arbiter of hardware access. But the guest OS inside the VM kept the same abstractions unchanged. Cloud gave us elasticity without reimagining the OS contract.

Evidence: AWS public docs and EC2 API design show remote compute exposed as virtual hardware, not a new OS primitive [V2 — aws.amazon.com].

Mobile Era: Constrained OS Models

Smartphones introduced a new constraint: battery, thermal limits, and app-store governance. iOS and Android adapted the desktop metaphor ruthlessly — apps, icons, home screens — while enforcing tight process sandboxing and a single foreground app focus.

Mobile was also where the "app" became an economic and distribution unit at planet scale. The application abstraction was not just a technical convenience; it was a business model enforced by store rules, in-app purchases, and review gates.

Why OS Change Is Rare

Operating systems are among the most path-dependent artifacts in technology. Changing an OS means:

- Reimplementing decades of drivers
- Migrating billions of installed applications
- Rewriting developer muscle memory
- Persuading enterprises to replatform mission-critical workloads
- Convincing hardware partners to abandon their support matrices

Compatibility is both a virtue and a cage. It is why Windows, Unix-like kernels, and macOS descendants remain recognizably similar after forty years.

Evidence: Christensen's *Innovator's Dilemma* and Gawer's platform-competition literature frame incumbents' inability to abandon installed base as a predictable pattern [V2 — Harvard Business Review / academic texts].

Bets That Did Not Land

History is littered with attempts to replace the dominant OS model:

- Microkernels (Mach, L4) promised cleaner isolation but delivered complexity and performance tax.
- Harmony OS (Huawei) was announced as a universal distributed OS; adoption remains concentrated in Huawei's own device fleet [V2 — 2024–2025].
- Plan 9 and Inferno introduced elegant network-transparent namespaces; they influenced research but never displaced Unix.
- Chrome OS slimmed the OS to a browser shell and succeeded in education, but stayed a thin client archetype rather than a new OS paradigm.

Each challenged specific assumptions. None overturned the core stack.

The Pattern Before the Inflection

Looking across six decades, OS transitions share a pattern:

10. Workload drift: new workloads accumulate on top of an old paradigm until the mismatch becomes painful.
11. Abstraction fracture: the OS begins to look like a compatibility shim layered over new hardware and software realities.
12. New primitive emerge: a new abstraction (virtualization, browser-as-runtime, container) becomes more important than the original.
13. Incumbent response: the dominant OS absorbs the new primitive without changing its core model.
14. Eventual recomposition: a new generation of systems rebuilds from the new primitive downward.

We are in stage 3 and early stage 4 today. Agentic AI is the emerging primitive. The OS vendors are responding with AI copilots and on-device ML. The recomposition has not yet begun.

Chapter 2 — The Foundational Abstractions of Modern Operating Systems

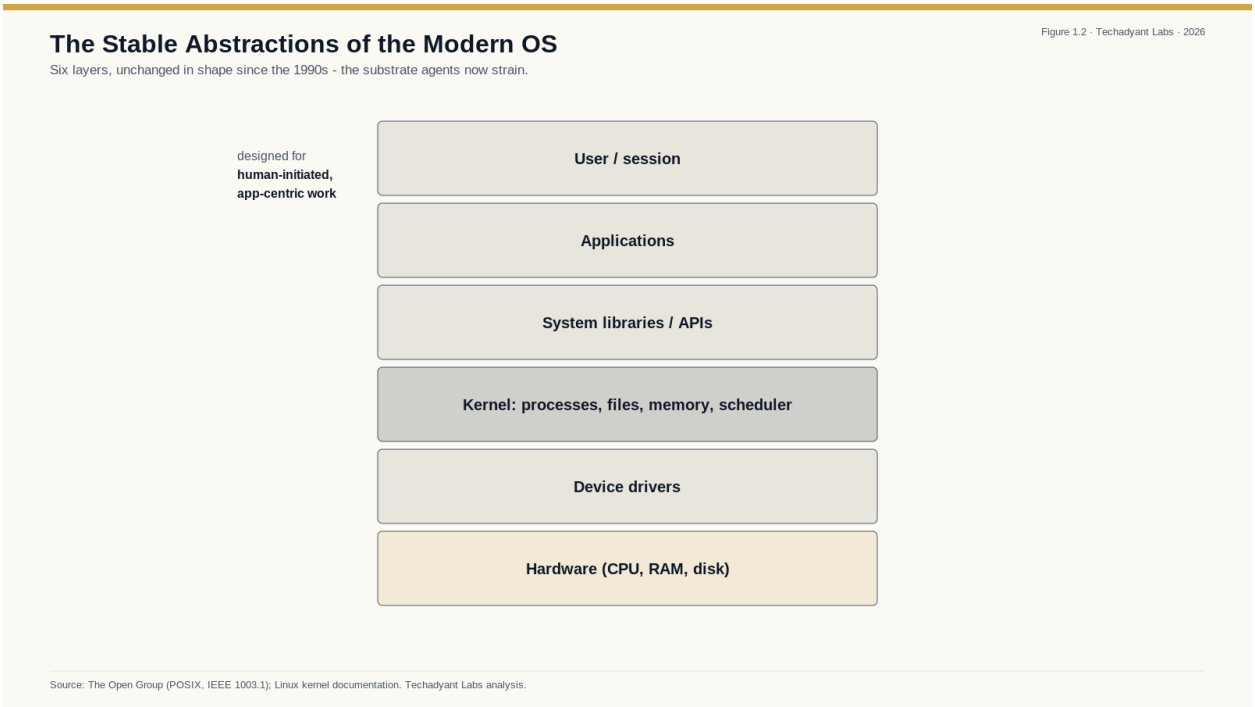

Figure 2 — The stable abstractions of the modern OS, unchanged in shape since the 1990s.

Bindings: Big Five spine claims 1 (application obsolescence), 2 (memory primacy), 3 (security inversion)

Thesis

Every modern OS is built on the same nine stable abstractions. They describe a world in which a human user launches a discrete application that runs inside a process, uses files and memory abstracted by the kernel, and is protected by identity-and-permission security. Those abstractions are coherent. They are also about to be asked to do work they were never designed to do.

User as Login Boundary

Every modern OS treats the *user* as the unit of accountability. A credential maps to an identity, and that identity owns files, processes, network sockets, and permissions. UNIX UID/GID, Windows SID, macOS BSD-derived UID — all converge on the same idea: the user is the policy boundary, quota boundary, and audit boundary.

Agentic workloads break this accounting. An agent is not a user in any POSIX or Windows sense. It runs under a service account, a bot token, or an OAuth client ID. It spawns tool calls that cross network boundaries on behalf of opaque goals. The OS has no native primitive for "agent identity," so permissions are either overly permissive (risky) or require fragile, app-level workarounds.

Application as Deployment and Trust Unit

The application is the OS's primary trust boundary. Installation is a gatekeeping ceremony: code signing, sandboxing, permissions prompts, package-manager verification. Once installed, the app owns its sandbox and can ask for broad access.

Agentic AI collapses the install model. A capable agent composes functionality at runtime from APIs, tools, and prompts. There is no installer. There is no signed bundle. Trust is negotiated per tool call, per data access, per context window — continuously, not at install time.

Process and Thread as Execution Boundary

Processes and threads are the OS's answer to "how do we run multiple things safely on one machine?" A process owns memory; threads share that memory under the same protection domain. The scheduler switches between them, partitioned by time slice.

For agentic workloads, this model is simultaneously too heavy and too light. A process assumes discrete beginning and end; an agent may run for days, sleeping and waking across inference calls. The default OOM killer and cgroup memory limits assume app-like memory profiles, not the persistent, growing context windows of long-running agents. Threading models assume shared-memory concurrency within a trust boundary; agents often need isolation between tool calls precisely to contain prompt-injection risks.

Filesystem as State Boundary

The filesystem is the OS's durable state abstraction. Hierarchical directories, file descriptors, permissions, and locking are the contract for durable storage. It works brilliantly for documents, logs, and blobs.

But agents do not think in files. They think in *context*: prompt histories, vector memories, tool schemas, function return values, and intermediate reasoning states. Today those are shoehorned into files, databases, or message queues. The OS provides no native abstraction for "agent state," forcing every framework to rebuild its own persistence layer.

Memory Hierarchy: Physical → Virtual → Cached

Virtual memory has been the great simplifier. Processes see a flat, private address space; the OS and MMU handle paging, caching, and eviction. Underneath, DRAM is a scarce pool managed in pages.

Agentic inference reverses some of these assumptions. LLM inference is memory-bandwidth-bound, not compute-bound. Context windows of 128k–1M tokens are common, and inference engines want those tokens resident in high-bandwidth memory (HBM) rather than paged out to SSD. The OS still thinks in pages and files; the AI runtime thinks in tensors and pinned memory pools. The abstraction mismatch is real and costly.

Drivers and the Hardware Contract

Device drivers translate hardware capabilities into OS-callable interfaces. They are also the OS's most fragile, security-critical, and least-maintained component. A new GPU, NPU, or sensor requires kernel-mode drivers that must match kernel versions exactly.

Agentic AI is generating demand for new hardware primitives — attention engines, vector caches, agent-dedicated schedulers — but the path from "silicon feature" to "usable OS interface" still runs through driver development. Until the OS provides a stable, high-level abstraction for AI-accelerator offload, the driver remains the chokepoint.

Security: Identity, Permissions, Capability

OS security is built on least privilege — but privilege is still expressed in terms of *users and groups*, not agents and intents. Mandatory Access Control (MAC), capabilities, and sudo rules are legible to system administrators, not to agents negotiating tool access.

When an agent invokes a tool that calls a database, the OS sees one service account running one process. There is no clean way to express "this tool call is authorized because the agent holds a capability token for this specific query." That gap is why agent security incidents look like broken-application logic rather than OS policy violations.

Scheduler: CPU Time as Finite Resource

The OS scheduler assigns CPU time to processes and threads. Its goals historically are fairness, responsiveness, and throughput for human-visible tasks. Linux CFS, Windows Multilevel Feedback Queue, and macOS Grand Central Dispatch are tuned for these ends.

Agentic workloads need different scheduling semantics. Inference tasks are bursty and latency-sensitive. Memory-pinning and NUMA-aware placement matter more than slice fairness. Long-running background research agents need low-priority reservation slots. The scheduler today is not designed for a zoo of heterogeneous workloads — LLM inference, agent loops, UI threads, browser tabs — all competing on the same machine.

The OS as Resource Arbiter

Taken together, the OS is the arbiter of hardware resources — CPU cycles, memory pages, device access, network bandwidth, and durable storage. Its abstractions define what kinds of workloads are *first-class* and which are bent into uncomfortable shapes.

Agentic AI is not merely another workload. It is a category that violates each abstraction's comfort zone at the same time: identity is no longer the right trust unit; the application is no longer the right deployment unit; the process is no longer the right execution unit; the filesystem is no longer the right state unit; the scheduler is no longer tuned for the right mix of tasks. The OS has absorbed many shocks over six decades. This one is concurrent across all layers at once.

Chapter 3 — The Limits of the Current Model

Bindings: Big Five spine claims 1 (application obsolescence) and 3 (security inversion)

Thesis

The current OS model is not broken, but it is accumulating architectural debt: CPU-centric assumptions, human-bound security, application silos, weak data mobility, and a compatibility tax that drains innovation velocity. Each problem is solvable in isolation; together they form the friction that makes the current model feel expensive under ever-larger agentic workloads.

Complexity Growth

Every OS release adds surface area: more APIs, more drivers, more subsystems, more backward-compatibility shims. Linux has crossed 35 million lines of kernel code1. Windows carries Win32, WinRT, UWP, POSIX, WSL, and decades of HAL layers in some form of supported maintenance.

Complexity has real costs. Patch turnaround lengthens. Security audits cover less surface per dollar of budget. New hardware integration requires driver authoring work that scales with kernel churn. For a world that wants faster adoption of new silicon and new AI primitives, the OS is becoming the drag, not the enabler.

Resource Fragmentation Across CPU/GPU/NPU

Since the 1970s, the OS has been built around the CPU as the central executive. Scheduler, memory management, interrupt handling, and power management all optimize for a processor-centric execution model2.

AI workloads invert this priority. The GPU, TPU, or NPU does the heavy math; the CPU mainly orchestrates data movement and launches kernels. When the OS's scheduling,

memory, and power policies optimize for CPU pipelines, AI accelerators are treated as devices — not peers. The result is inefficient data movement, missed NUMA locality, and accelerators that spend cycles waiting for host-side coordination.

App-Centric Human UI Design

Windows, macOS, and Linux desktops are designed around the metaphor of a human sitting in front of an application window, clicking and typing. Alt+Tab, the Dock, the taskbar — all assume one foreground task per human.

Agents do not need a foreground window. They need background execution, headless tool calls, and persistent context. When an OS insists that every workload must attach to a visible app window or foreground process group, the workload either fights the shell or hides inside it. Neither is a stable long-term design.

Data and Identity Silos

Every application today owns its data store and authentication boundary. Your calendar lives in Calendar; your mail lives in Mail; your code lives in a Git repo behind a different auth flow. The OS provides shared folders and keychains, but those are thin interoperability layers over deep silos.

Agentic workflows require composable data access across services, with fine-grained delegation that survives cross-application boundaries. The current model forces agents to operate as just another app — logging in, scraping UI, caching credentials — instead of as first-class participants in an intent-to-data path.

Security Mismatch for Delegated Intent

Legacy OS security assumes a human at the end of every permission grant. A user clicks "Allow," types a password, or taps a biometric prompt. The trust chain is human-mediated.

Agents mediate their own trust through tool calls, function returns, and multi-step reasoning. When an agent browses a hostile page, executes code, and sends results back, the OS treats the entire chain as one trusted process. There is no clean way to revoke access to a

specific tool invocation, no native "agent capability token," and no audit log that distinguishes "user-approved" from "agent-delegated."

The mismatch is not theoretical. Prompt injection, over-permissioned tool APIs, and agent credential leakage are already producing new failure modes that map poorly onto existing security telemetry and incident-response playbooks.

Emerging Performance Bottlenecks

Modern AI inference is bottlenecked by memory bandwidth, interconnect latency, and power envelope — not single-thread CPU performance. Yet OS memory management, process scheduling, and power profiles still optimize for CPU-bound workloads3.

Linux's cgroup v2 and memory-pressure kill model assume app-like memory behavior: spike, settle, release. LLM inference and agent context behave differently: large pinned allocations, steady-state residency, streaming access patterns. The mismatch produces latency spikes, unexpected OOM events, and drivers that cannot satisfy memory-pinning requests under load4.

Maintenance and Compatibility Tax

Every generation of OS inherits the obligation to run the previous generation's software. That obligation is healthy in moderation and pathological in excess. Windows runs 1990s 16-bit applications through compatibility layers. Linux maintains multiple system-call ABIs. macOS sandboxes and translates legacy APIs.

The tax shows up as slower innovation on the frontier. Engineers who could be designing agent-native primitives are instead maintaining Win32 message handlers, SysV IPC shims, or HFS+ Journaling compatibility paths.

The Cumulative Diagnosis

These limitations are not independent. Complexity begets slower driver and security response. App-silo thinking begets fragile agent integrations. CPU-centric tuning begets accelerator inefficiency. Compatibility tax beheads innovation velocity.

The OS is not broken. It is *misaligned* — with the workload that will dominate compute demand in the next decade. Realignment does not require throwing away the OS. It requires evolving the primitives so that agentic workloads become first-class.

Part II — The AI Disruption

Chapter 4 — From Applications to Agents

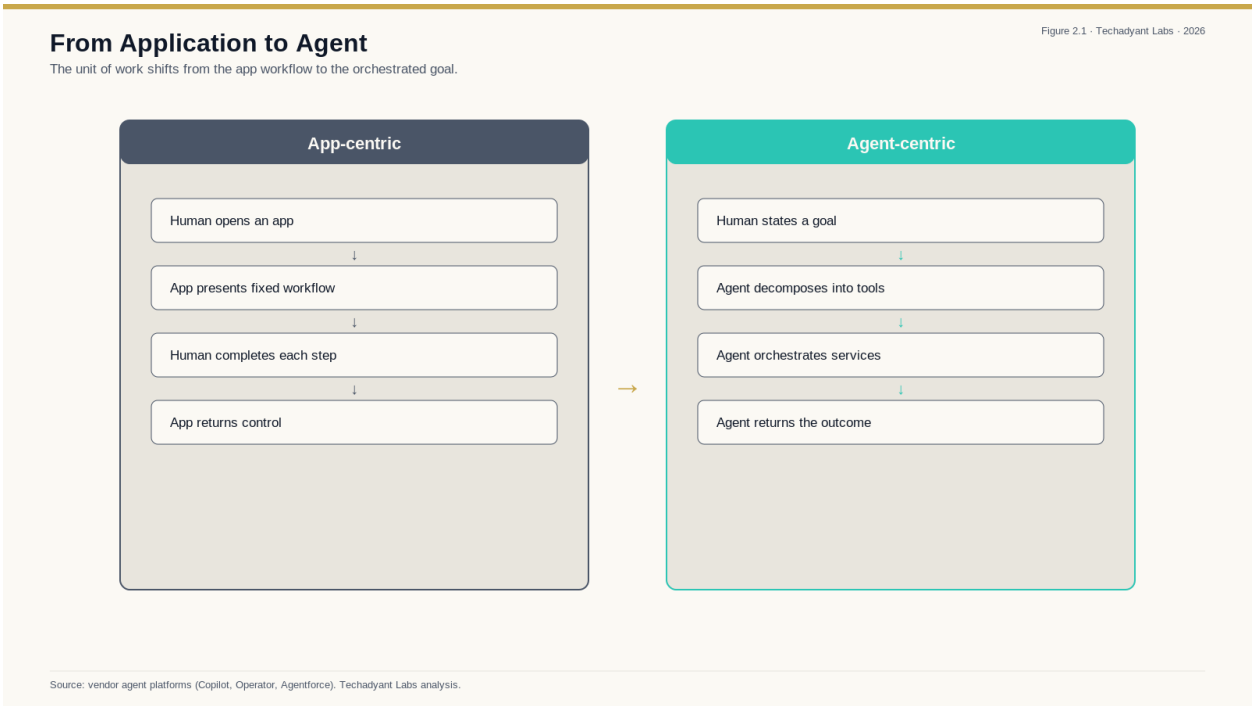

Figure 3 — The unit of work shifts from the app workflow to the orchestrated goal.

Bindings: Big Five spine claims 1 (application obsolescence) and 2 (memory primacy)

Thesis

Agentic AI replaces the application-centric interaction model with intent-driven, tool-orchestrated, memory-rich autonomous workflows; this breaks the OS assumption that humans initiate discrete tasks through app UIs.

What Is an Agentic AI, Really

An *agentic AI* is not just a chatbot. It is a system that:

- receives a goal,
- decomposes it into steps,
- chooses tools and APIs,
- executes multi-step tool sequences,

- observes the results,
- reasons about success or failure,
- and persists state across attempts.

A traditional application does exactly one of these: it executes a fixed workflow, usually launched by a human. An agent executes *many* workflows and chooses between them based on observed outcomes. That distinction is the rupture.

Evolution: Scripts → Macros → RPA → Agentic AI

Automation has been creeping up the abstraction ladder for decades.

- Scripts automated OS-level command sequences (shell, PowerShell, AppleScript).
- Macros automated application-level interactions (Excel VBA, IDE macros).
- RPA automated UI-level interactions across multiple applications (UiPath, Automation Anywhere).
- Agentic AI automates goal decomposition and tool selection dynamically, without a predefined DAG.

Each step increased expressiveness but also abstraction distance from the OS. RPA already broke the app-boundary assumption by simulating mouse clicks and keyboard input across processes. Agentic AI goes further by using structured APIs and reasoning chains rather than UI scraping — but it inherits the same tension: it operates across apps without being inside an app.

Workflow Anatomy

A canonical agent lifecycle contains stages the OS was never designed to mediate:

15. init: goal specification arrives through API, CLI, or message queue.
16. context: system prompt, user profile, retrieved memories, and tool schemas are assembled into a working context window.
17. plan: the model outputs a plan or tool-call sequence.
18. tool use: each tool call runs in a separate process or network call — potentially outside the OS's awareness.
19. memory: results are summarized and persisted back to long-term memory stores.

20. terminate: the agent returns a result, persists state, and releases resources — but may remain available for follow-up.

The OS sees a series of discrete processes and network connections. It does not see the loop, the goal, the memory, or the plan. The agent's meaningful lifecycle is invisible to the OS's auditing and management tools.

Multi-Agent Coordination

The largest agent systems today are multi-agent: orchestrators that decompose goals, specialized agents that execute sub-tasks, and reviewers that validate outputs. Currently this coordination happens inside application frameworks — LangGraph, CrewAI, AutoGen, or custom orchestration code.

If multi-agent coordination became an OS primitive — if the kernel understood "agent group," "shared context," and "delegation token" — we would not need every framework to rebuild deadlock detection, state sharing, and failure recovery. Today we do, because the OS offers no such primitive.

Why Agents Break OS Design Assumptions

Five specific breaks:

21. No install ceremony: agents are deployed as code, prompts, and tool configs — no package manager, no signature, no installer.
22. No foreground requirement: agents run in headless processes, serverless functions, containers, or remote workers.
23. No bounded lifecycle: agents may run indefinitely, pausing and resuming across inference calls.
24. Tool calls cross trust boundaries: each API invocation is a fresh authentication and authorization event.
25. State is semantic, not file-based: the agent's memory is vector embeddings and conversation summaries, not POSIX files.

These are not workarounds. They are core design properties that contradict the OS's default assumptions.

Observability and Trust Implications

When an agent fails — hallucinates, calls the wrong API, loops infinitely — the OS treats it as a misbehaving process. The debugging primitives are `strace`, `perf`, and core dumps. They show you *what* the process did, not *why* it chose that action.

Agentic failures require different telemetry: reasoning traces, tool-call lineage, memory state at decision time, prompt history. Those belong at the agent-runtime layer today, not the OS layer. But as agents become more powerful and more autonomous, pushing observability entirely into user space becomes a risk, not a feature.

Chapter 5 — New Workloads, New Requirements

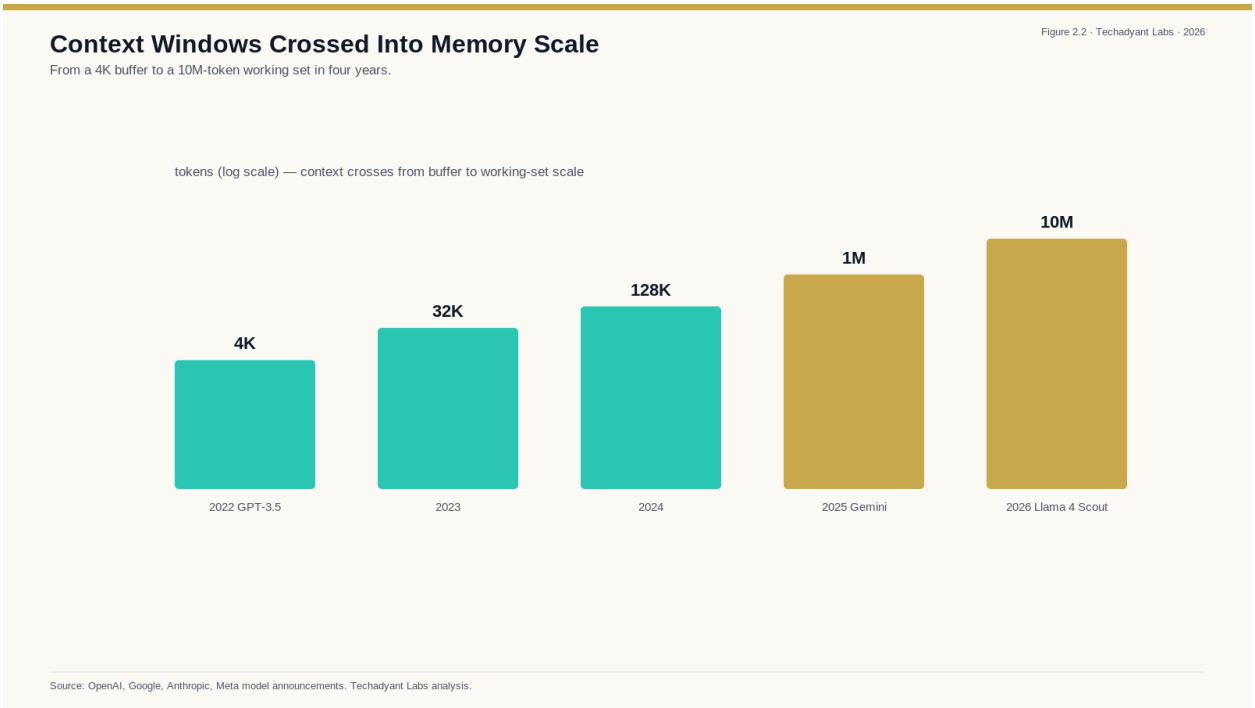

Figure 4 — Context windows crossed from a 4K buffer to a 10M-token working set in four years.

Thesis

Agentic AI workloads demand continuous inference, massive and persistent context, tool-accessible APIs, long-running execution, and heterogeneous compute — none of which are first-class citizens in current OS designs.

Continuous Inference vs Burst Inference

Traditional OS workloads are bursty. An application launches, performs a computation, and returns. Even "always-on" services like web servers handle request/response cycles.

Agentic workloads are continuous in a different sense. An agent may maintain persistent context for hours or days, pausing only to wait for inference results or external tool responses. The OS's idle-process heuristics, timer ticks, and power-management timers are not designed for workloads that are *neither interactive nor batch* — they are always-on but not always compute-bound.

Context as Persistent Memory

The defining feature of an agent is its context window: the conversation history, tool schemas, retrieved memories, and system prompt that together determine its behavior. Context windows are growing — from 4k to 128k to 1M+ tokens — and agents are expected to maintain them across tasks and sessions.

Current OS memory management treats all allocations equally. It does not distinguish between the agent's long-lived context and a spreadsheet's temporary scratch buffer. As a result:

- An agent's context may be paged out during memory pressure, destroying effective "memory."
- Long-lived LLM-backed processes may be killed by OOM under the same thresholds that protect interactive apps.
- Pinning large context tensors in memory requires explicit, OS-specific API calls outside normal application flow.

Tool and API Accessibility as OS-Level I/O

Agents interact with the world through tools: database queries, API calls, file writes, browser sessions, code execution. These are today's I/O. But the OS provides no unified interface for "tool invocation" with built-in auditing, rate limiting, authentication delegation, or failure recovery.

Each tool call is a fresh network request or subprocess spawn. The OS does not track tool-call lineage, aggregate cost, or enforce per-agent budgets. Every framework rebuilds these capabilities in user space, leading to inconsistent security and observability.

Long-Running Autonomous Execution

The OS process model assumes a bounded lifecycle: fork → exec → run → exit. Agents violate all four assumptions. They may be forked once and run for weeks, spawning child processes and tool calls without exiting. Their "exec" is a model inference call that is not a traditional executable. Their "exit" may be a state transition to idle, not termination.

Current OS schedulers, cgroups, and systemd-style supervisors penalize this behavior. Idle-but-resumable processes consume memory without contributing to interactive throughput, making them targets for OOM and prioritization policies.

GPU/NPU/Edge Compute Requirements

Agentic inference runs on GPUs today and NPUs tomorrow. The OS must manage:

- discrete memory pools (VRAM, HBM, unified memory) that are not interchangeable with DRAM
- device-to-device transfers (GPU → NPU → SSD) that bypass the CPU cache hierarchy
- heterogeneous scheduling: CPU, GPU, NPU, and DSP each need independent queues and priorities

The OS partially addresses this through IOMMU, UAPI, and CUDA/WDDM driver stacks, but the abstractions remain device-specific and driver-dependent. A unified "accelerator scheduler" that understands inference workloads does not yet exist in any mainstream OS.

Agent Swarms and Coordination Overhead

The next order of magnitude in agent capability is *swarms*: many lightweight agents cooperating on a complex goal. Swarms introduce communication overhead, shared-state consistency, and failure-propagation problems that resemble distributed systems challenges.

OS-level message-passing primitives (pipes, sockets, message queues) exist, but they were designed for inter-process communication at human timescales, not for thousands of agent invocations per minute with strict ordering and idempotency requirements. The primitives are too low-level; the frameworks are too application-specific.

Energy and Thermal Envelopes

LLM inference and agent swarms are energy-intensive. Large GPU/TPU/NPU workloads generate significant heat and draw substantial power. Mobile and edge devices must manage thermal throttling under continuous inference loads.

Current OS power management is tuned for interactive apps with idle/sleep cycles, not for continuous, predictable, high-energy inference. Without OS-level awareness of inference workloads, power and thermal policies will remain reactive, triggering throttling or shutdowns at the worst possible moment.

Chapter 6 — The Accelerator Revolution

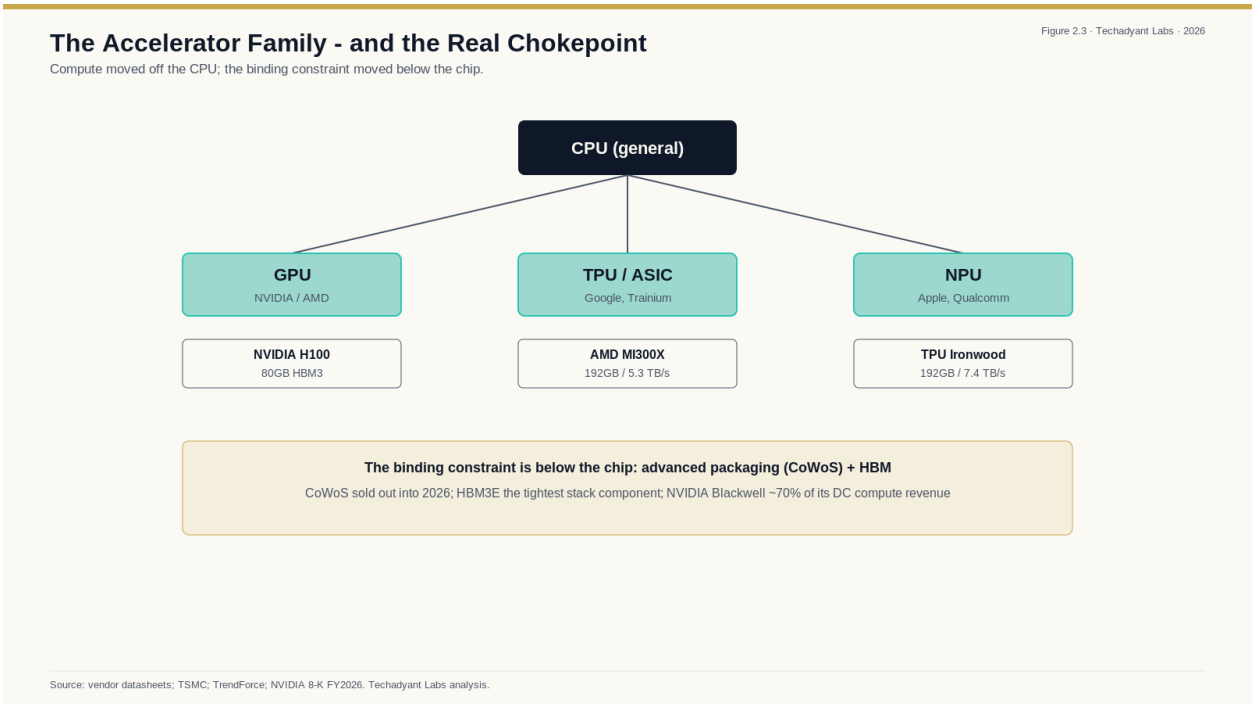

Figure 5 — The accelerator family, and the real chokepoint below the chip.

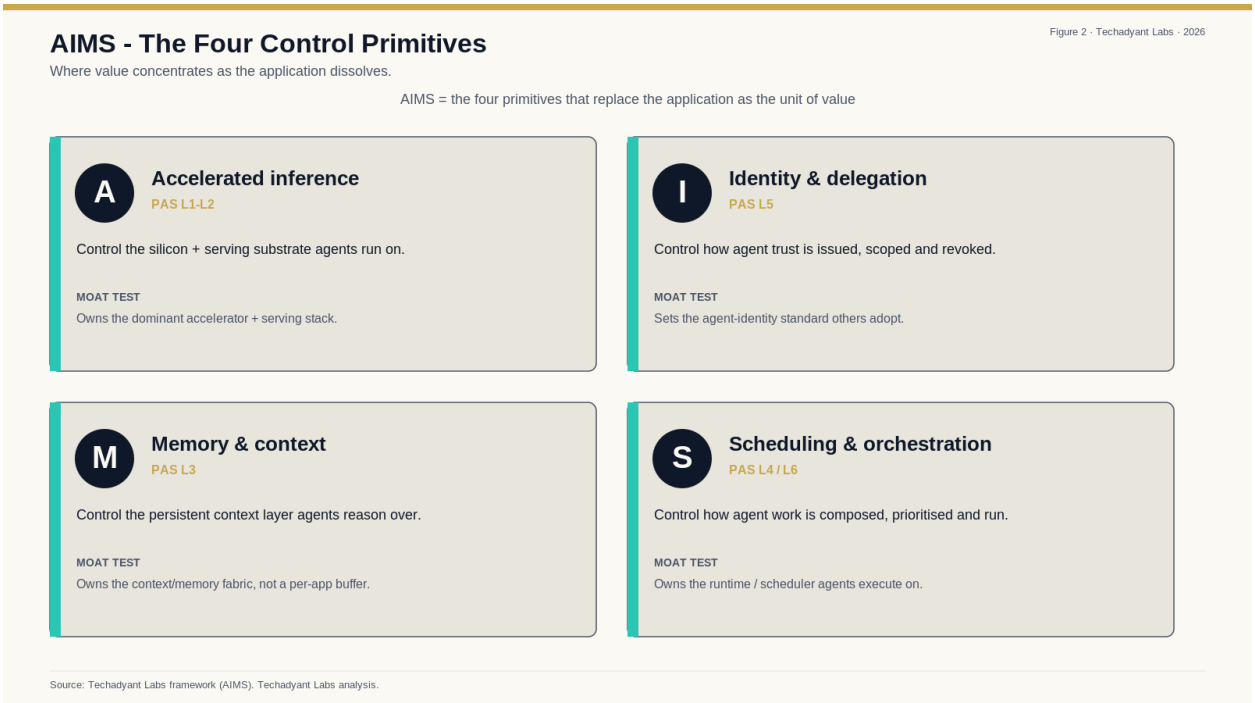

Figure 6 — AIMS — the four control primitives that replace the application as the unit of value.

Thesis

The rise of GPUs, TPUs, NPUs, and unified-memory architectures is ending CPU dominance and forcing the OS to become a hardware-orchestration layer for heterogeneous accelerators, not just a CPU scheduler.

GPU Rise: From Graphics to General Compute

Before 2006, GPUs were fixed-function pipelines tuned for triangle rasterization and texture mapping. NVIDIA's CUDA launch in 2006–2007 shattered that boundary. By exposing thousands of parallel arithmetic units through a C-like language, CUDA turned the GPU into a general-purpose parallel processor.

The OS had no category for this device. It was not a CPU, not a traditional DMA peripheral, not a co-processor with a known ISA. The driver model stretched to accommodate user-space memory regions and kernel-bypass compute submission queues. Every subsequent GPU generation — AMD's ROCm, Intel's oneAPI — inherited and extended these accommodations.

Today CUDA and its ecosystem represent one of the most successful hardware-software co-evolutions in computing history. But they remain an *add-on* to the OS, not a native abstraction.

TPU and Custom ASIC Turn

Google's Tensor Processing Unit (first detailed in 2017) abandoned general-purpose GPU flexibility for a narrower, higher-efficiency inference and training engine. ASICs like the TPU, Inferentia, Trainium, and Google's newer Trillium are characterized by:

- systolic arrays optimized for matrix multiplication
- large on-chip SRAM for weight reuse
- minimal support for branching and control flow

From the OS perspective, these devices are *even more opaque* than GPUs. They rely on the host CPU for memory management, job scheduling, and data movement. The OS treats them as devices that accept batched work orders; it does not understand the work.

NPU on Edge Devices

The neural processing unit is now a first-class silicon block in billions of devices: Apple's A-series and M-series chips, Qualcomm Snapdragon and X Elite, MediaTek Dimensity, and Google Tensor. NPUs are small, low-power, and purpose-built for inference at the edge.

The presence of NPUs in laptops and phones creates a new heterogeneous-compute profile for the OS: three compute tiers (CPU / GPU / NPU) with different power, memory, and performance characteristics. Windows, macOS, Android, and iOS are each trying to abstract this heterogeneity for developers, but the underlying OS driver and scheduler integration is still immature.

Unified Memory and Memory-Centric Design

The next architectural shift is toward *unified memory* — a single memory pool accessible by CPU, GPU, and NPU without unnecessary copies. Apple's Unified Memory Architecture (UMA) and NVIDIA's Blackwell architecture with its coherent memory fabric point in the same direction.

Unified memory challenges the OS's traditional memory model, which assumes separate physical address spaces for devices and requires explicit DMA mapping and cache flushing. Without OS-level abstractions for coherent shared memory across accelerators, developers must manage data movement manually — defeating the purpose of unification.

AI-Centric Hardware and Reduced CPU Roles

In an AI-native workload, the CPU becomes an orchestrator and data mover, not the compute engine. It launches kernels, shuffles tensors, and handles control flow. The actual arithmetic lives on accelerators.

This is a profound role reversal for the OS. The CPU scheduler has been tuned for six decades to maximize *its own* utilization. When the CPU becomes a peripheral co-processor for the GPU/NPU, the scheduler, power manager, and interrupt controller all need new logic.

Scheduling and Resource Isolation Across Accelerators

Scheduling is the OS's most performance-critical function. It currently assigns CPU slices to processes and threads. GPU and NPU scheduling is handled either inside the device (hardware queues) or by vendor-specific user-mode runtimes.

As accelerator workloads multiply, *cross-device* scheduling becomes the bottleneck. Should an inference job run on the edge NPU or be sent to the cloud? Should a training job be interleaved with an inference job on the same GPU? These decisions require visibility into workload characteristics, device state, power envelope, and network latency — exactly the kind of global optimization the OS should perform but today cannot.

Hardware Trajectory Through 2035

Looking five to ten years forward, the key hardware trends are:

- Disaggregation: compute, memory, and interconnect will separate into chiplets and domain-specific modules.
- Bandwidth over FLOPs: memory bandwidth and interconnect capacity, not raw math throughput, will be the gating resource.
- Edge-to-cloud gradient: inference will run at multiple points in the network, not only in hyperscale datacenters.
- Specialization without fragility: new accelerator types will appear faster than the OS driver ecosystem can absorb them without abstraction improvements.

The OS that wins the next decade will be the one that makes heterogeneous accelerator scheduling, memory coherence, and power management as routine as process scheduling is today.

Part III — Which Abstractions Break First

Chapter 7 — The Death of the Application

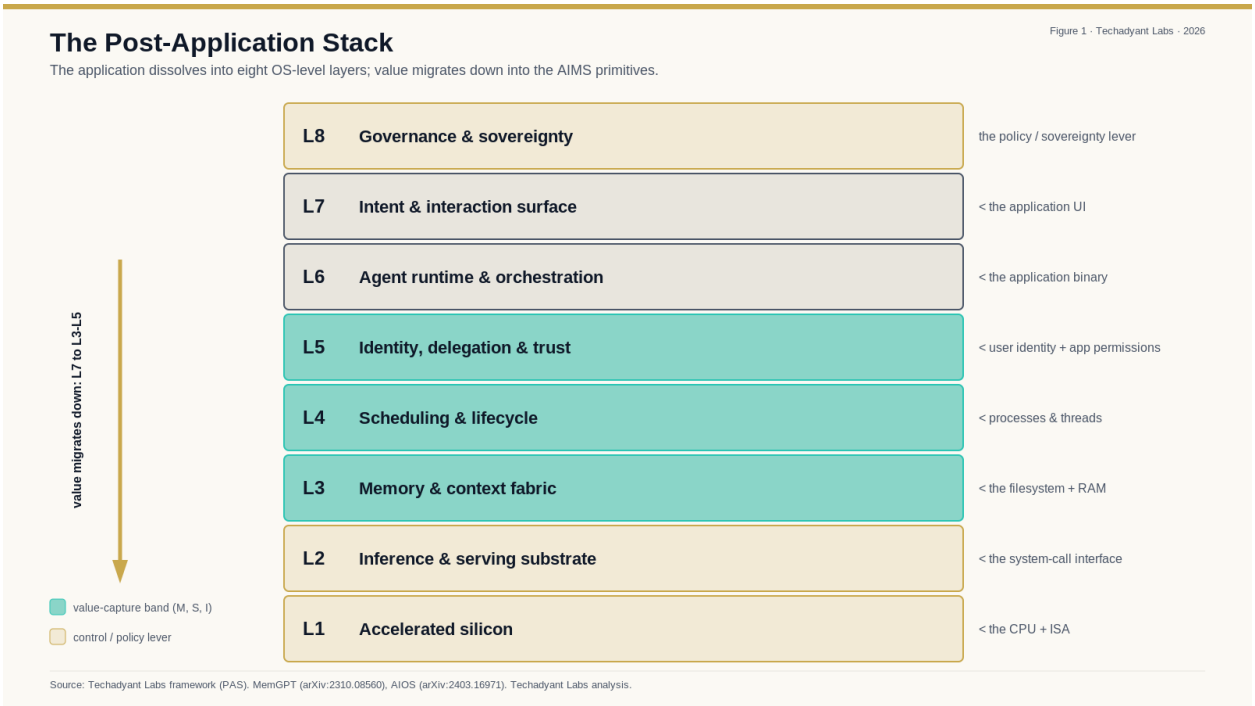

Figure 7 — The Post-Application Stack: value migrates down from L7 into the AIMS primitives.

Thesis

The application abstraction is becoming structurally obsolete under agent-centric workloads, where intent-based computing and dynamic composition replace installed binaries and isolated app silos. The application is a legacy unit — a discrete bundle launched by a human to perform a bounded task — while agents operate as continuously running, intent-driven actors that compose tools dynamically. Real-world deployments from 2024–2026 already show orchestration replacing discrete application workflows.

Application as Today's Trust and Distribution Unit

The modern software economy is built on a single idea: the application is the unit of trust, distribution, and monetization. You install an app, grant it permissions, and it owns the ground inside its sandbox. The OS enforces that boundary. App stores, package managers, code signing, and runtime permissions are all elaborate ceremonies around this abstraction.

Agentic AI bypasses the ceremony entirely. An agent composes functionality at runtime through APIs and tool calls. There is no installer, no reviewer gate, no binary signed by a

known publisher. Trust becomes a continuous negotiation — per tool call, per data access, per context window — not a one-time grant.

App-Centric vs Agent-Centric Interaction

In the app-centric model:

- A human opens an app.
- The app presents a fixed workflow.
- The human completes discrete actions inside that workflow.
- The app returns control to the human.

In the agent-centric model:

- A human (or another agent) states a goal.
- The agent decomposes the goal into a tool sequence.
- The agent orchestrates across multiple services without human-per-step input.
- The agent returns a synthesized outcome.

The application was designed as a container for human-initiated, bounded workflows. Agents are containers for autonomous, unbounded goals. The mismatch is not a UX problem; it is an architectural problem. The distinction is not whether an agent uses applications — it uses APIs, libraries, and services — but that it does not treat those as "apps" in the Windows, macOS, or Linux sense. The application category becomes an implementation detail.

Case Studies: Orchestration Replacing Applications

The following deployments from 2024–2026 show where discrete application workflows are already being replaced by agent orchestration.

Case A — Microsoft Copilot and GitHub Copilot. Before: developers searched documentation, copy-pasted snippets, manually refactored, and ran debug cycles across IDE, browser, and terminal. After: Copilot operates as a continuously running agent within the IDE — retrieving context across files, proposing multi-step edits, running tests, and revising plans on failure. The developer shifts from writing syntax to directing intent. The IDE becomes a view into an ongoing agent process rather than a code editor.

Case B — OpenAI Operator and Claude Computer Use. Before: humans navigated web interfaces, clicked menus, filled forms, and copied data between tabs. After: these agents control the browser directly — reading the DOM, clicking elements, submitting forms, recovering from errors — without step-by-step human direction. The browser becomes a tool the agent uses, not the interface through which a human operates applications.

Case C — Devin (Cognition). Before: developers assigned coding tasks via tickets, reviewed PRs, ran integration tests. After: an autonomous software-engineering agent clones repos, reads issue trackers, plans implementation, writes code, opens PRs, and iterates on review feedback — eliminating the handoff between issue tracker, IDE, CI pipeline, and review tool by orchestrating across all of them.

Case D — Cursor. Before: an IDE with syntax highlighting, a terminal, and a separate AI chat window. After: a code-aware agent indexes the codebase, edits across files, runs terminal commands, and updates its plan from build output. The boundary between editor, assistant, and build tool dissolves; the agent, not the application, is the unit of orchestration.

Case E — Replit Agent. Before: users opened a browser IDE, wrote code manually, configured environments, and deployed through a separate hosting workflow. After: the agent accepts natural-language goals, initialises environments, writes and runs code, debugs, and deploys. The browser IDE and hosting console become substrate; intent becomes the interface.

Case F — Salesforce Agentforce. Before: service staff used CRM dashboards, case-routing tools, email templates, and knowledge bases as discrete applications. After: a single agent-driven flow handles case classification, knowledge retrieval, response drafting, and escalation; tier-1 support runs autonomously, humans handle exceptions. The CRM becomes a backend; the agent becomes the interface.

Case G — ServiceNow AI Agents. Before: IT service requests moved through separate portals for incident, change, request, and problem management. After: agents interpret intent, classify incidents, execute routine changes, and coordinate across teams without users navigating different applications. The ITSM suite becomes infrastructure; the agent becomes the user-facing layer.

The common pattern: in every case the application persists as a backend utility, but ceases to be the unit through which work is initiated, composed, or completed.

Intent-Based Computing

The next abstraction is *intent*: a human or system specifies a desired outcome, and the platform figures out the steps. This is already visible in AI assistant mode switches and natural-language API layers.

But true intent-based computing requires the OS to understand:

- What tools and data are available
- What constraints apply (privacy, budget, policy)
- How to compose and execute a plan
- How to observe results and retry

Operating systems do not model intent. They model requests with context: open file, allocate memory, create thread. An agent running atop today's OS must translate its semantic state — intent, plan, goal progress — into a sequence of low-level OS calls. That translation is the entire scope of application frameworks today, and it is where most failures occur. The OS should be the natural owner of this abstraction, not every vendor independently.

Dynamic Composition vs Static Bundling

Applications are *statically bundled*. A spreadsheet ships with charting, calculation, and presentation views because that is what 1990s PC users needed. The bundle is fat because bundling was the only way to ship an experience.

Agents compose dynamically. A research agent might call a database, a web search, a code executor, a PDF reader, and an email tool — all separate services, chosen at runtime based on result quality, latency, cost, and trust. No vendor bundles all those capabilities into one binary because no single vendor can guess every workflow.

This dynamism breaks assumptions

Chapter 8 — The Future of Processes and Threads

Thesis

The traditional process/thread model is too heavyweight and too isolated for agent lifecycles; the OS will evolve toward finer-grained, memory-rich, cooperative execution units tailored for agentic workloads.

Process Model Origins and Limits

The process was invented to give each program its own protected address space — a sandbox that prevents one misbehaving program from corrupting another. Threads were added so a single program could do several things at once while sharing memory.

For human-initiated, short-lived tasks, this model works. For agents, it creates three problems: weight, isolation mismatch, and lifecycle mismatch.

Weight. Each process carries a full virtual memory map, file descriptor table, signal handlers, and security context. For thousands of lightweight agent tasks, this overhead is wasteful. Existing solutions — containers, namespaces, cgroups — reduce the footprint but add management complexity.

Isolation mismatch. Agents need isolation *between tool calls*, not just between processes. A PDF-parsing tool call should not share memory with a database-query tool call in the same agent, because the PDF content might contain a prompt-injection attack. The process boundary is the wrong granularity for this kind of defense-in-depth.

Lifecycle mismatch. Processes assume a fork/exec/exit arc. Agents assume init → loop → idle → resume → exit. Systemd, launchd, and supervisord can approximate this with service units, but they impose rigid state machines and restart policies that assume failure is always a crash — not a deliberate "wait for next inference."

Thread Scalability Under Many Concurrent Agent Tasks

Modern many-core CPUs can run thousands of threads, but threading models assume threads share memory and run in a common trust domain. For agents, that is a vulnerability. If multiple agent threads share a context window, a prompt injection in one tool call's return value could influence another thread's reasoning.

Future execution models will need:

- Thread-like lightweight scheduling without shared-memory coupling by default
- Explicit, mediated data channels between agent tasks
- Isolation boundaries that are finer than a process but coarser than a function

WebAssembly component model and WASI proposals are moving in this direction: language-agnostic, lightweight, capability-mediated sandboxes that could become the agent's native execution unit.

Agent Lifecycle as an OS Primitive

The OS currently understands *process lifecycle*. It does not understand *agent lifecycle*. Yet the lifecycle operations are distinct and meaningful:

- spawn agent: provision working context, memory, tool access
- run step: execute one model inference + tool-call cycle
- yield memory: evict least-recent context to free RAM while preserving resume capability
- hibernate: serialize agent state to durable store
- resume: restore context, reattach tools, continue
- terminate: clean shutdown with memory audit and tool revocation

If the kernel understood these states, it could apply the right policy at each stage — pinning memory during active reasoning, compressing context during idle periods, revoking credentials during termination — without every agent framework rebuilding the lifecycle manager from scratch.

Resource Allocation: Tokens, Memory, API Budget as OS Quotas

Today the OS allocates CPU time, memory pages, and network packets. Tomorrow it may need to allocate *tokens*, *memory bandwidth*, *inference budget*, and *API call quota* — resources that did not exist as first-class concepts until 2023.

cgroup v2 already provides the *mechanism* for arbitrary resource accounting: you can define a controller for any res schedulable quantity. What is missing is the *policy* — the conventions, defaults, and APIs that make "give this agent 1M tokens of pinned memory and 100 API calls per minute" as simple as `ulimit -v`.

Swarm Scheduling: Many Lightweight Agents vs Few Heavy Processes

Swarm agent systems need:

- Sub-millisecond scheduling latency for individual agent invocations
- Batch optimization for many agents arriving simultaneously at an inference endpoint
- Dynamic rescheduling when a node fails or a tool result arrives early
- Backpressure so that an agent does not launch 1,000 parallel calls that collectively exhaust memory or API quotas

These are scheduling problems the OS already solves, but at different scales and for different workload shapes. The scheduler's current heuristics — tuned for interactive responsiveness and fair-share batch — need to learn a new lane: *micro-workload burst with semantic backpressure*.

Interoperability and State Sharing

Agents need to share state safely: a research agent may discover an entity that becomes relevant to a writing agent, or a planning agent may need to inject constraints into a coding agent's context.

The OS provides pipes, sockets, shared memory, and filesystems for this. None match the semantics agents actually need:

- Structured, schema-validated state transfer — not byte streams
- Subscribable notifications — not polling or blocking reads
- Access-control list on individual data items — not whole-file permissions
- Versioned state — not last-write-wins filesystem semantics

Agent-Oriented Kernel Concepts

Research prototypes hint at what an agent-aware kernel might expose:

- capability tokens as first-class IPC messages
- memory objects that can be pinned, versioned, and shared by semantic handle
- scheduling classes for agent workloads with explicit inference and latency goals
- audit logs that record tool calls, memory state transitions, and delegation chains

None of these require abandoning the existing process model. They require extending the kernel's vocabulary so that agent runtimes can express richer semantics without rebuilding everything in user space.

Chapter 9 — The Future of Memory

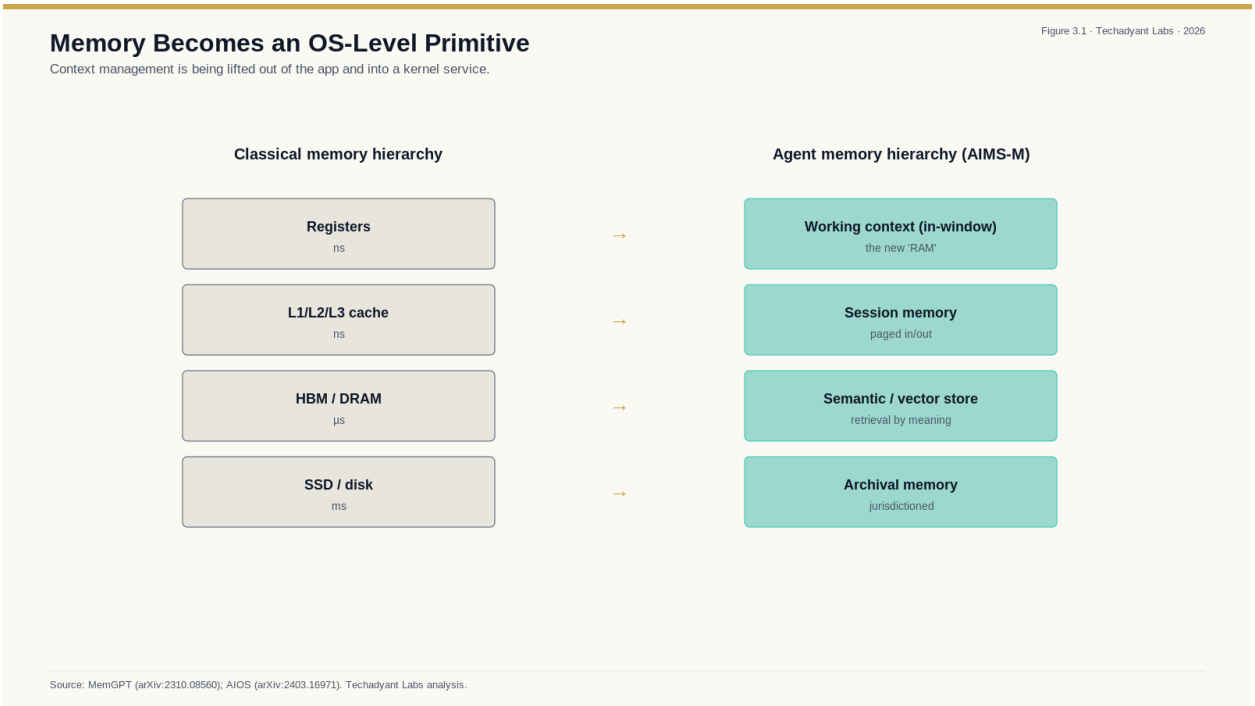

Figure 8 — Memory becomes an OS-level primitive: context management lifts out of the app.

Thesis

Memory is shifting from RAM-centric, app-bounded allocation to persistent, semantic, vector-capable, shared memory that becomes an OS-level primitive for agent context and identity.

RAM-Centric Design and Its Origins

Von Neumann stored-program architecture made RAM the universal workspace. The OS enforces this model: every process sees a flat private address space; the kernel translates virtual to physical pages; paging and swapping extend the illusion.

This model served sequential and concurrent human-initiated workloads well. Tasks allocate, compute, and release. Memory pressure is transient and recoverable.

Agentic workloads invert several assumptions. An agent's *context window* — its operational memory for reasoning — is not scratch space. It is the agent's identity for the duration of a task. Losing it is not a recoverable event; it is a lobotomy.

Memory Hierarchy Under Agent Workloads

The memory hierarchy is becoming visibly mismatched:

- LLM weights: large, read-only, frequently reused → best kept in HBM or high-bandwidth unified memory.
- KV-cache (key-value cache): large, read-mostly, growing with context length → wants pinned, non-pageable memory.
- Agent context & memory: medium-sized, persistent, semantically indexed → wants structured storage with vector-search interface.
- Scratch / tool output: small, short-lived → standard DRAM allocation is fine.

Today an OS treats all four as equal anonymous pages. The result is that inference runtimes must use explicit memory-pinning, NUMA-aware allocation, and hugepages — heroic measures outside the OS's normal memory-contract API.

Vector Memory and Semantic Memory as New Primitives

Software agents do not reason about byte addresses. They reason about *meaning*. A memory primitive that returns "what do I know about semiconductor corridors in India?" must perform semantic search, not hash-table lookup.

Vector databases (pgvector, Pinecone, Qdrant, Redis Vector Sets) and embedding stores have emerged to fill this gap. They are powerful but exist entirely in user space. The OS has no `open()` for semantic memory, no `mmap()` for vector indices, no `ioctl` to flush dirty embeddings.

If memory is becoming semantic, the OS needs a *semantic memory primitive* — a first-class abstraction alongside files, sockets, and pipes.

Persistent Context Across Sessions and Tasks

Agents need continuity: a research session that spans days must be able to resume where it left off, rehydrating context from durable storage into a fresh inference session. Persistence today is implemented as framework-level serialization — JSON, SQLite, or custom checkpoints — not an OS service.

An OS-level *session state* abstraction would provide:

- atomic checkpoint / resume
- incoherent-cache-safe memory restoration
- encryption and integrity verification of saved state
- policy-driven retention and eviction

Shared Memory Between Collaborating Agents

Multi-agent systems require safe, efficient shared memory. Two agents collaborating on a plan need:

- read/write access to a shared plan state
- change notifications when the other agent updates the plan
- versioned history so conflicts can be reviewed or rolled back

Linux `mmap` with `MAP_SHARED` is the closest primitive, but it is byte-addressable, unversioned, and has no semantic change notifications. Database-backed shared state works but adds serialization overhead and network hops. An agent-optimized shared-memory primitive would look like a *distributed, versioned, semantically-addressable object store* — closer to a database than to a `malloc` buffer.

Memory Sovereignty

As agents act on behalf of users and organizations, the question of *who owns the agent's memory* becomes consequential. Is agent memory an extension of the user's private data? Is it corporate property if the agent runs on corporate infrastructure? Who can read, copy, or delete it?

Current OS memory protections enforce boundaries between processes and users but not between *workload owners*. A cloud provider can page agent memory to disk, inspect it for security events, or retain it beyond the user's intent. Agent memory is a new asset class that current memory governance does not adequately cover.

Memory as an OS Primitive

The leap from "memory is bytes" to "memory is context, identity, and state" is comparable to the leap from "storage is blocks" to "storage is files." We made that leap in the 1960s with filesystems and never looked back.

Agentic computing requires a similar leap now: memory should expose not just capacity but *structured, semantic, durable, shareable state* as part of the core OS interface. Filesystems will remain for documents, logs, and blobs. But the OS needs a second memory abstraction optimized for the retrieval, persistence, and governance of agent state.

Chapter 10 — Rethinking Security

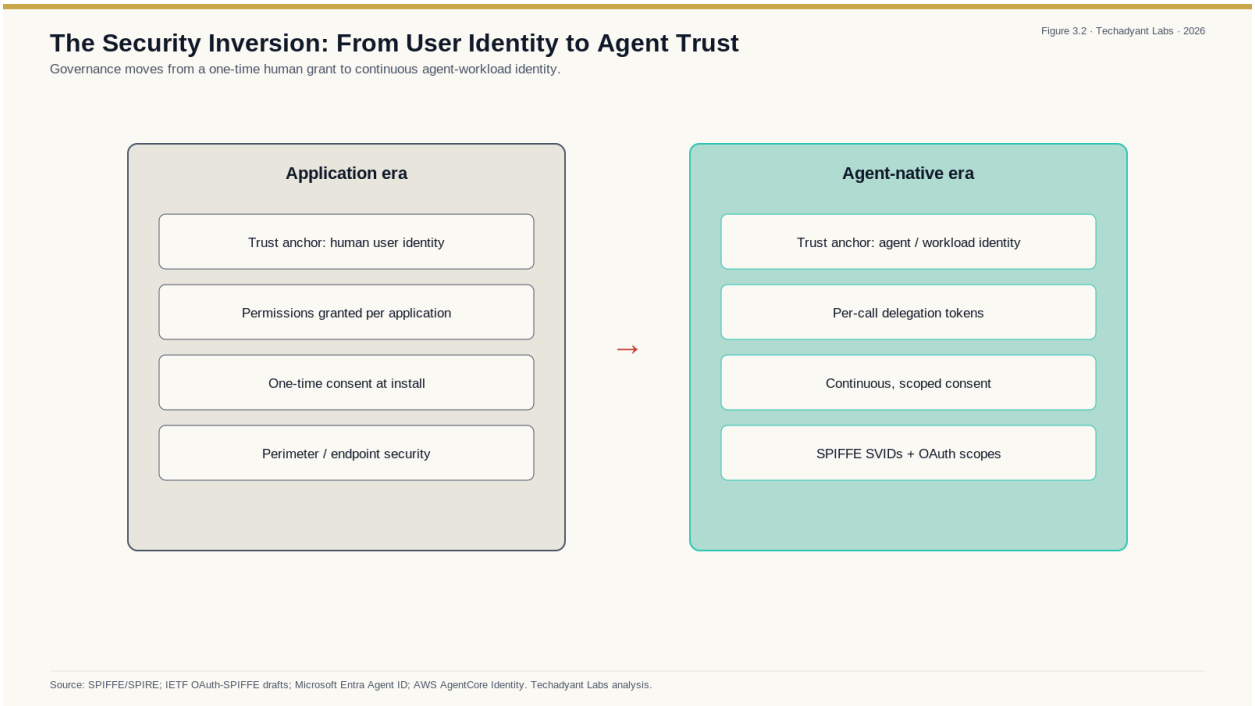

Figure 9 — The security inversion: from user identity to agent-workload trust.

Thesis

Agentic systems require a security model that moves beyond the human-user boundary to delegate trust, contain tool-call blast radius, and audit autonomous behavior — forcing the OS to rethink identity, isolation, and logging.

Legacy OS Trust Model

Today's OS security rests on a single abstraction: the authenticated human user. Every permission grant, sudo prompt, and biometric check is an approximation of "this human approved this action." The chain of accountability runs from kernel to user to application.

That model worked when every privileged action could be traced back to a conscious human decision. It does not work when an agent acting on behalf of a user initiates hundreds of tool calls per hour, each with its own risk profile, without pausing for human approval.

The Agency Gap

Current OS security has no concept of *agency*. There is no kernel-level primitive for:

- A capability token that expires after a specific task
- A delegation boundary that limits what an agent's child process can do
- A sandbox that is parameterized by agent policy rather than user policy
- An audit log that records "agent X, task Y, tool Z, decision source P"

Without these primitives, agent security is implemented in application code — inconsistent, unaudited, and bypassable. An agent compromise reads not as an OS intrusion but as a misbehaving app, which means incident-response teams treat it with the wrong playbook.

Prompt Injection and Agent-Specific Threat Vectors

Prompt injection is a threat vector almost entirely new to the agent era. User input (or tool output) is interpreted by the model as instruction, contaminating the agent's reasoning chain. Traditional OS security does not protect against *data being interpreted as code* inside a model's context window.

Worse, the contamination can persist across tool calls and sessions via memory writes. A poisoned memory entry can influence future agent decisions days later, long after the original injection vector has been closed.

The OS needs to understand that an agent's *context* is a security-critical asset and must be integrity-protected, versioned, and auditable with the same rigor it brings to a process's memory pages and executable segments.

Identity, Delegation, and Zero Trust for Agents

Zero Trust architecture says "never trust, always verify." For agents, this means every tool call, every API invocation, and every data access should be independently authorized — not leased in bulk at agent startup.

The OS is the natural place to manage this. A native delegation model would let an agent present a short-lived, scoped capability token rather than long-lived API keys or service-account passwords. The kernel or a privileged system service could mint, inspect, and revoke those tokens atomically.

Today we approximate this with OAuth scopes and IAM roles, but those are network-layer constructs. They do not bind to the executing process, are not enforced by the OS, and do not follow the agent as it resumes from hibernation on a different node.

Auditing Autonomous Behavior

Audit logs are the forensic backbone of security. Today's OS audit logs record syscalls, file access, and network connections. That is sufficient when a single process corresponds to a single human intent. It is insufficient when one agent orchestrates dozens of tools across multiple processes and network hops.

Agent auditing needs richer semantics:

- Tool-call lineage: which agent spawned which tool and why
- Memory transitions: what was read, written, or forgotten and when
- Reasoning checkpoints: model outputs that led to action decisions
- Delegation chains: nested agents and their respective scopes

These events should be emitted as first-class structured telemetry, not reverse-engineered from strace or network traces.

Isolation Levels: Agent, Task, and Tool Granularity

Security isolation has three relevant levels for agents:

- Agent-level: one agent's full context is isolated from another agent's.
- Task-level: sub-tasks within one agent are isolated so a failed tool call cannot corrupt the agent's reasoning.
- Tool-level: individual tool invocations run with minimal credentials and cannot escape their sandbox even if the agent's plan is compromised.

Current OS containers and VMs provide agent-level isolation reasonably well. Task-level and tool-level isolation remain application responsibilities. An OS that understood agent semantics could enforce these levels natively, reducing the attack surface without requiring every framework to reimplement sandboxing.

Containers and Agent Workloads in Production

Containers (Docker, Podman, Kubernetes) are the current production host of choice for agents. They provide resource isolation, image immutability, and restart policies. But they were designed to run long-lived services, not fine-grained, short-lived, high-density agent workloads.

The mismatch creates operational friction:

- Each tool call as a separate container is too heavy; shared-container tool calls re-expose shared-memory risks.
- Kubernetes namespaces are designed for service ownership, not per-task authorization boundaries.
- Secrets management (K8s Secrets, Vault) solves credential storage but not delegation semantics for tool calls.

The Security Posture We Need

A viable agent-era security posture combines:

- Capability-based delegation tokens scoped to tool and time
- Integrity-protected agent context with audit trails
- Crushed blast radius: tool calls isolated by default, not as an afterthought
- Native audit events that distinguish human approval from agent delegation
- Revocation that is immediate and observable, enforceable at the kernel level

The OS vendors building these capabilities first will define the trust layer that every agent runtime — proprietary or open-source — must integrate with.

Part IV — AI-Native Operating Systems

Chapter 11 — Emerging AIOS Architectures

Thesis

The first generation of AI-native operating systems is already visible in research prototypes and commercial bets — microkernel exploration, AI-first scheduling, semantic memory integration, and intent-aware shells — but no incumbent has yet committed to a new core OS paradigm.

Microkernels and Minimal Kernels

Microkernels (Mach, L4, seL4) push everything non-essential into user space, keeping only interrupt dispatch, IPC, and basic scheduling in privileged mode. This design reduces the trusted computing base — a significant security benefit.

Why microkernels matter for AI-native OS design:

- Agent runtimes can run as user-space services with clean IPC boundaries
- New AI schedulers, memory managers, and capability brokers can be evolved without kernel changes
- Failed or compromised agent subsystems can be restarted without rebooting the kernel

seL4 has demonstrated formal verification of the kernel. For AI systems where correctness and containment are critical, a verified minimal kernel is an attractive foundation for regulated and infrastructure-grade agent deployments.

Data-Centric and Metadata-Driven Designs

A data-centric OS treats *state* — not process — as the primary abstraction. Rather than "what process owns this resource?", the kernel asks "what metadata describes this data, and who is authorized to access it?" For agents, this is a natural mapping. Agent state is *data*: context, memory, tool results, plans. A data-centric kernel could manage that state natively rather than force every framework to bolt a custom persistence layer onto a process-centric OS.

- Files become documents with rich metadata.
- Memory becomes structured, versioned, queryable.
- Execution becomes a function of intent and available data, not a command run by a specific user.

Self-Driving / Intent-Aware OS

An intent-aware OS translates human or agent intent into system configuration. Today's desktop OS requires users to configure policy manually: firewall rules, power profiles, app permissions. An agent-native OS would infer and apply policy from declared intent.

- "I want this research agent to have priority on GPU and read-only API access for 2 hours" → OS allocates resources, grants scoped tokens, schedules accordingly.
- "Run this payment agent in an isolated network namespace with no access to personal contacts" → OS provisions the environment.

This is close to what Android and iOS already do with app permissions, but applied to agent *capabilities* instead of app installs.

OS-Level AI Scheduler

An AI scheduler uses internal models of workload behavior to make scheduling decisions. Linux CFS and Windows Multilevel Feedback Queue are heuristic schedulers; an AI scheduler would learn from workload telemetry.

For AI-native OS scheduling, this means:

- Predicting inference-job memory footprints and adjusting memcg limits proactively
- Balancing LLM inference latency against training-job throughput
- Migrating agent contexts between CPU and GPU affinity zones based on observed utilization
- Detecting and correcting priority inversions where an agent's child process is starved by batch jobs

The OS already collects the telemetry. It does not yet have the policy-learning layer on top.

Agents as OS Primitives: A Functional View

If agents were OS primitives, the kernel or system services would expose:

- agent_spawn: create a named agent with scoped capabilities and budget.
- agent_step: run one inference+tool-call cycle and return structured state.
- agent_memory: read, write, or query the agent's durable semantic memory.
- agent_hibernate / agent_resume: checkpoint and restore agent state.
- agent_delegate: create a capability-scoped sub-agent.
- agent_audit: stream structured audit events to a compliance sink.

These system calls would not dictate implementation; they would define a *contract* that agent runtimes and the kernel agree on. Frameworks could still compete on model routing, tool orchestration, and UX. But core lifecycle, memory, and security would be standardized at the OS layer.

Hardware-Vendor Interest: Apple, NVIDIA, Google, Microsoft

Apple is building the most vertically integrated path. Its silicon + macOS/iOS stack already moves inference onto the NPU, uses unified memory to share model weights, and orchestrates compute through Metal. The missing piece is OS-level primitives for agent lifecycle and semantic memory. Apple's tight hardware-software control makes it the fastest path to a unified agent runtime — if the company decides to treat agents as a first-class user model.

NVIDIA dominates the AI compute layer. Its hardware roadmap and CUDA ecosystem already define de facto standards for LLM inference. NVIDIA has no OS play today, but its interest in extending control from compute to runtime is clear from CUDA-X and NGC catalog investments. A vertically integrated NVIDIA OS would prioritize one thing: maximum accelerator utilization for AI workloads.

Google controls silicon (TPU), OS (Android, ChromeOS/Fuchsia), and model (Gemini). It has the opportunity to integrate agents more tightly into Android services than any other vendor. Google has not yet committed to this; Gemini in Android remains an app feature, not a kernel-level primitive.

Microsoft controls the dominant desktop OS and is integrating Copilot across Windows. Its bet is currently an application-layer agent, not a reimagined OS. That decision is defensible

— Windows' installed base allows a slow migration — but leaves Microsoft vulnerable if a competitor ships a fundamentally cleaner agent-native platform.

Linux Flavors: System76 Pop!_OS, Asahi Linux

Linux is the most flexible substrate for experimentation:

- Asahi Linux is reverse-engineering Apple Silicon support, producing GPU/NPU driver stack innovations that could benefit any Linux-on-arm deployment.
- Pop!_OS / COSMIC is rethinking the desktop shell from scratch, exploring tiling, workspace, and window-rule concepts that could evolve into agent-session management.
- Android remains the world's most-deployed Linux derivative; its app-permission model is the closest thing to an enterprise-scale agent-capability experiment.

Linux's weakness is not technical depth; it is governance fragmentation. A radical new agent-native OS model would require a kernel-maintainer consensus that may take years to achieve.

What Is Missing Today

Across all these explorations, the same five capabilities are missing:

26. Native agent lifecycle at kernel or system level
27. Semantic memory as a first-class OS abstraction
28. Capability-scoped delegation tokens enforced by the kernel
29. Cross-accelerator scheduling with memory-coherence awareness
30. Agent audit telemetry as structured, queryable system events

Existing architectures prioritize different subsets. None delivers all five. A complete AI-native OS design would integrate all five while maintaining backward compatibility with existing application models — the hardest engineering constraint of all.

Chapter 12 — Candidate Architectures for the Future

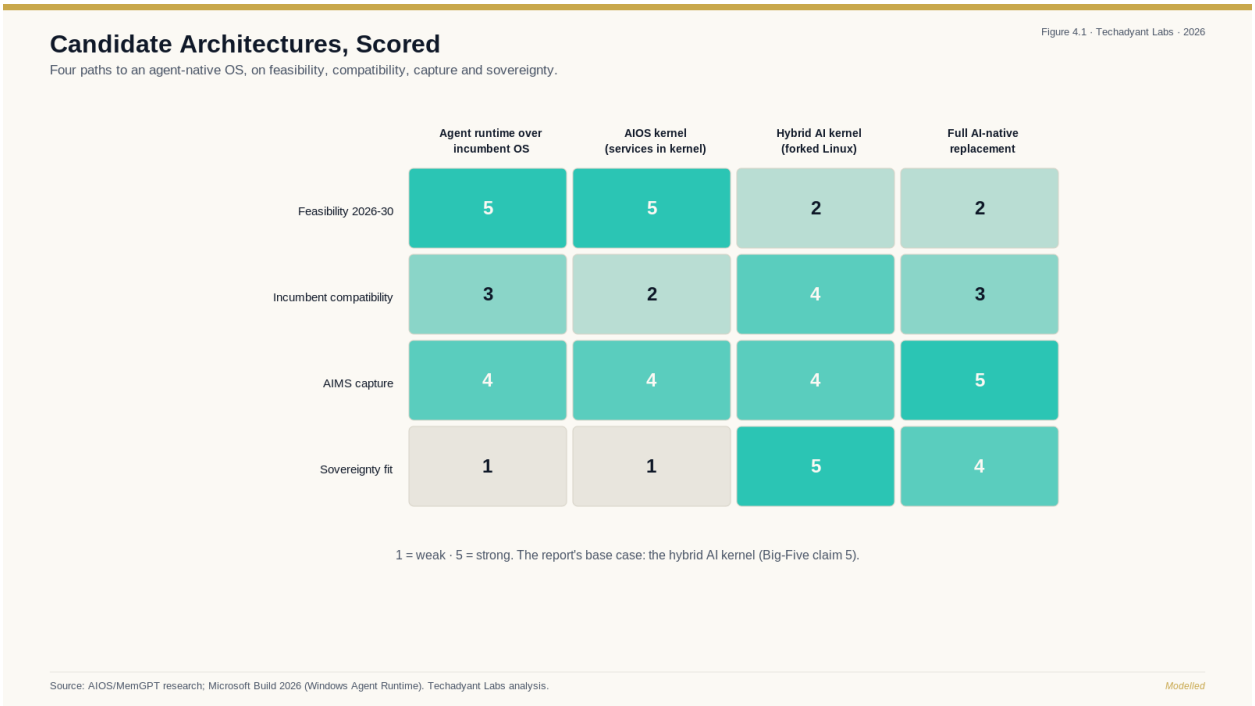

Figure 10 — Four candidate architectures, scored on feasibility, compatibility, capture and sovereignty.

Thesis

Three plausible architectures for the AI-native OS are emerging: a Unix-extended model, a microkernel agent-runtime model, and a web-centric model; each trades legacy compatibility, security, and performance differently.

Unix-Extended Model

Premise: Evolve the existing Unix/Linux kernel to support agent-specific syscalls and scheduling classes, leaving the POSIX model intact for backward compatibility.

Mechanics:

- New syscalls for agent_spawn, agent_step, agent_memory, agent_hibernate/resume.
- cgroup v2 extended with custom agents controller for token budgets and memory pinning.

- BPF programs used to enforce agent-level audit and policy without kernel modifications.

Pros:

- Largest installed base of existing applications and drivers.
- Linux community already has the expertise and governance mechanisms.
- Incremental adoption: agent features can be opt-in via new APIs while POSIX workloads continue unchanged.

Cons:

- Linux kernel velocity cannot keep pace with agent innovation needs.
- Kernel maintainer culture resists concepts like semantic memory and intent-aware scheduling that lack clear syscall analogues.
- The POSIX compatibility requirement may prevent radical simplification of the memory and security models.

Verdict: Most likely short-to-medium-term path. Microsoft could pursue a similar NT-extended model in Windows.

Microkernel Agent-Runtime Model

Premise: Restart the OS around a verified microkernel, with all agent-specific subsystems (scheduler, memory, capability broker, tool sandbox, audit) implemented as user-space services communicating via IPC.

Mechanics:

- seL4 or similar verified kernel as minimal trusted base.
- User-space agent runtime as the primary OS service, exposing a sealed API to applications and frameworks.
- Filesystem, network, and existing app compatibility provided by compatibility layers that talk to the microkernel's file and network servers.

Pros:

- Minimal trusted base reduces attack surface for agent security.
- Agent runtime can evolve rapidly in user space.
- Failure isolation: a compromised agent runtime cannot corrupt the kernel.

Cons:

- Performance overhead from IPC and context switching is non-trivial for high-throughput AI workloads.
- Legacy app compatibility is harder; compatibility layers add complexity and latency.
- The market has limited appetite for another OS transition after Windows-macOS-Linux consolidation.

Verdict: Right architecture for security-critical infrastructure (financial, defense, industrial control); unlikely for consumer computing in the next decade.

Web-Centric / Browser-as-Shell Model

Premise: Treat the browser or a web-runtime shell as the primary agent host, using web standards (WASM, WebGPU, WebRTC, WebAuthn) as the OS interface.

Mechanics:

- Browser becomes the agent runtime.
- WASM modules serve as lightweight, capability-scoped sandboxes.
- WebGPU provides GPU/NPU access from within the sandbox.
- Browser trust and permission APIs provide token-scoping primitives.

Pros:

- Browser is the most universally deployed runtime in history.
- WASM component model already enables language-agnostic, sandboxed composition.
- Cross-platform consistency: code runs identically on Windows, macOS, Linux, iOS, Android.

Cons:

- Memory pinning and large-context management are constrained by browser sandbox rules.
- Persistent, cross-session agent state requires explicit browser storage APIs with quotas and eviction semantics that may not match agent needs.
- GPU access through WebGPU is improving but remains a fraction of native CUDA/Metal performance for large models.
- Browser vendors historically resist extensions that look like OS features.

Verdict: Strong candidate for web-centric and cross-device agents; cannot replace native OS for high-performance AI workloads.

Hybrid Model: Kernel Agent Services + User-Space Runtime

Premiptise: Provide a thin set of agent-aware kernel services (memory pinning, capability tokens, audit hooks) while keeping the main agent runtime in user space, preserving most application compatibility.

Mechanics:

- Linux or Windows kernel extended with optional agent-controller modules.
- A privileged system service (like systemd or launchd) manages agent lifecycle.
- Frameworks (LangGraph, AutoGen, Semantic Kernel) talk to the controller via standard APIs.
- Existing apps run unchanged on the same kernel.

Pros:

- Minimal kernel surface area changes.
- Maximum application compatibility.
- Fastest path to deployment: ship as an optional kernel module + system service.

Cons:

- Agent services remain a bolt-on, not a first-class primitive.
- Cross-vendor portability is limited to distributions that ship the module.
- Innovation still depends on kernel release cadence for the primitive layer.

Verdict: The most pragmatic near-term path; likely how AI-native OS features arrive in production before radical redesigns.

Mobile-First Agent OS

Premise: Reimagine mobile OS (Android, iOS descendants) from the ground up around agents rather than apps, because mobile is where the economic incentive to replace the app model is strongest and the hardware constraints are most conducive to agent-native design.

A mobile-first agent OS would:

- Replace app icons and home screens with intent surfaces (voice, text, proactive agent cards).
- Treat the NPU as primary compute and the CPU as orchestrator.
- Manage agent contexts as system-managed semantic memory.
- Use strong per-agent capability tokens derived from biometric-gated user consent.

Pros:

- Clean slate: no legacy Win32 baggage.
- NPU-first hardware matches agent inference profile.
- Strong vendor control enables tight integration.
- User base is large and app-fatigue is real.

Cons:

- Mobile app stores are multi-trillion-dollar economic ecosystems; vendors are unlikely to abandon them quickly.
- Privacy and sandboxing expectations are stricter, making cross-agent data sharing harder.
- Enterprise migration from mobile app strategies to agent strategies has not yet begun.

Verdict: High potential long-term path; vendor willingness is the blocker.

Distributed Agent OS

Premise: An OS where the kernel and runtime are distributed across edge, edge-to-cloud continuum, and datacenter, with agent contexts and compute migrating transparently.

Mechanics:

- Agent context lives in a distributed semantic memory layer (edge cache → cloud storage).
- Inference executes on the most appropriate device given latency, power, and cost.
- Capability tokens are portable across nodes.
- Agent state snapshots can be serialized, transferred, and resumed on a different physical machine.

Pros:

- Matches the reality of hybrid edge-cloud AI deployments.
- Resilience: agents can relocate from a failing node without losing state.

- Optimal resource utilization: inference runs where hardware is best matched.

Cons:

- Networking latency limits interactive agent responsiveness.
- Distributed consistency and trust are unsolved in general; agent memory integrity across nodes is even harder.
- Requires a level of hardware and cloud vendor cooperation that does not yet exist.

Verdict: The eventual target architecture; too complex for near-term production.

Trade-offs: Compatibility, Security, Performance, Control

Table 2 — Trade-offs: Compatibility, Security, Performance, Control.

Architecture	Legacy Compatibility	Security	Performance	Vendor Control
Unix-Extended	High	Medium	High	Fragmented
Microkernel	Low	Very High	Medium	Centralized
Web-Centric	High	Medium	Low-Medium	Browser vendor
Hybrid Kernel+User	High	Medium-High	High	Fragmented
Mobile-First	Low	High	High	Centralized
Distributed	Low	Low-High	Variable	Requires cooperation

No single row wins across all columns. The market question is which trade-off profile matches the first dominant AI-native use case.

Why Now or Never

The next 3–5 years are the decisive window:

- Incumbent OS vendors are still investing in their existing models.

- Once an AI-native workload reaches mass adoption on a given substrate, switching costs make alternatives prohibitively expensive.
- First-mover advantage in agent-native APIs will define the compatibility baseline for decades.

Chapter 13 — What Happens to Windows, Linux, and macOS

Thesis

The incumbent operating systems face a common threat and three divergent paths: adapt by absorbing agent primitives, cede the surface to a new architecture, or fragment into legacy support layers while a new platform takes the leading role.

Windows: Copilot as Bridge, Not Destination

Microsoft has integrated Copilot deeply into Windows. The strategy is clear: keep users in the Windows shell while an AI layer intercepts intent, launches workflows, and manages apps on the user's behalf.

This is a defensive bridge strategy. It extends Windows' relevance by making the existing OS a better *host* for agentic activity without changing the underlying abstractions. Files, apps, processes, and security remain unchanged; Copilot adds a translation layer between human intent and OS operations.

The risk is complacency. If Copilot stays an application-layer feature and Windows does not invest in kernel-level agent primitives, it becomes a compatibility shim — valuable, but not the platform where new value is created.

Linux: The Research Substrate and Cloud Backbone

Linux runs the internet, the cloud, and most AI training infrastructure. It is not in danger of disappearing. But the Linux desktop is a different story.

For agentic OS design, Linux is already the testbed:

- Kernel researchers add BPF programs that enforce custom policy without rebooting.
- Container orchestrators treat workloads as ephemeral units, inadvertently previewing agent-task semantics.

- Android is a Linux derivative that has already solved per-app capability scoping at planetary scale.

Linux's challenge is not survival; it is *coherence*. The distribution ecosystem rewards fragmentation. A coherent agent-native OS layer emerging from Linux would likely come from one dominant distribution or a hyperscaler-patched kernel tree, not from a committee of maintainers.

---

macOS / iOS: The Vertical Integration Play

Apple controls hardware, OS, and increasingly silicon AI acceleration. The Mac and iPhone are the most vertically integrated general-purpose computing platforms in existence.

Apple's strategic question is whether agents become an OS feature or an app feature. Today they are app features: Siri, Shortcuts, and on-device ML APIs. A shift to OS-level agent support would mean:

- System-managed semantic memory synchronized across devices.
- Native agent lifecycle in the kernel or launchd.
- NPU-first scheduling with persistent-inference power profiles.
- Cross-app capability delegation managed by the system, not individual apps.

This is strategically consistent with Apple's historical approach: absorb a new paradigm into a tightly controlled platform. The risk is that Apple's walled garden restricts the open agency that enterprise and developer markets demand.

---

iOS / Android: The Mobile Agent Surface

By 2030, most AI-agent interactions may happen on mobile devices — voice-first, always with you, context-rich from sensors. iOS and Android have structural advantages here:

- Strong NPU integration already in silicon.
- Biometric-gated user identity.
- App-permission models that almost express agent capabilities.

The transformation would replace the app launcher with an *intent surface*. Rather than opening an app, the user states a goal; the OS finds or spawns an appropriate agent; the

agent operates through APIs and returns a result. The app becomes invisible infrastructure rather than the user's primary interface.

Chrome OS: Thin Client or Agent Gateway?

Chrome OS was designed as a thin client for cloud applications. It succeeded in education and entry-level laptops by minimizing local complexity.

An agent-native Chrome OS could become a *compute gateway*: lightweight local orchestration plus seamless handoff to cloud inference and tool execution. The browser shell already hosts many agent UIs. If Chrome OS adds native memory management, secure agent sandboxing, and edge-to-cloud inference routing, it becomes a credible minimalist competitor to full desktop OSes — particularly in price-sensitive markets.

The Fragmentation Risk

Every OS vendor faces a shared trap: invest in the old OS or the new one. The wrong bet is fatal, but hedging both paths is expensive.

The worst outcome is fragmentation:

- Windows stays application-centric.
- Linux becomes a research-only substrate.
- macOS/iOS becomes a sealed appliance.
- A new upstart captures the agent-native layer.

This is not hypothetical. Early PC OS fragmentation (CP/M, MS-DOS, OS/2, Mac OS) created a decade of market confusion before Windows consolidated. A similar fragmentation around AI-native computing would delay deployment, confuse developers, and fragment the security and compatibility landscape.

BSD and Niche Unix Variants

OpenBSD continues to lead on security. FreeBSD powers cloud storage and networking appliances. NetBSD targets embedded portability.

Their role in the agent-native future is likely niche but influential:

- OpenBSD's security innovations will shape the threat model for any new OS primitive.
- FreeBSD's ZFS and network-stack reliability make it attractive for infrastructure-grade agent memory and audit stores.
- NetBSD's portability work informs edge-device AI deployment.

These Unices are unlikely to become mainstream consumer or enterprise OSes, but they will continue to export ideas that better-funded competitors adopt.

The Incumbent Response We Should Expect

Based on historical platform transitions, expect a three-phase pattern:

31. Denial and feature extension: "Our existing OS can handle this with new APIs." (2024–2026)
32. Partial adaptation: New capabilities appear, but they bolt onto the old model. (2027–2030)
33. Crisis and fork: One or two vendors commit to a new core; others defend legacy. (2030–2035)

The question is not *whether* the incumbents will change, but *how quickly* they abandon the sunk cost of their existing abstractions.

Part V — Industry & Economic Transformation

Chapter 14 — Impact on the Technology Industry

Thesis

Agentic AI will reallocate value across the technology industry — software vendors, chipmakers, cloud providers, and system integrators — and the winners will be those that capture the new OS and runtime layer.

Value Migration from Apps to Runtime

The software industry's current value structure is app-centric. A spreadsheet or CRM bundles features and charges per seat or per instance. Enterprise buyers evaluate feature matrices; switching costs are high and data gravity is real.

When agents become the composition layer, value migrates:

- From application features to tool access and capability breadth.
- From human UI design to agent orchestration and memory quality.
- From seat licenses to API calls and inference tokens.
- From per-app procurement to platform-level agent contracts.

Vendors that cannot articulate their role in an agent-native stack face a Microsoft-Office-past-the-browser moment: still useful, but no longer where the strategic margin lives.

Chipmakers and the Accelerator Economy

Chipmakers are the clearest near-term beneficiaries. Training and inference demand is doubling supply-chain investment in advanced nodes, HBM, and interconnect. TSMC, Samsung, and SK Hynix are capacity-constrained through at least 2026.

The risk for chipmakers is *change faster than roadmaps*. A new OS primitive for accelerator scheduling could commoditize GPU compute the way Linux commoditized CPU compute. The chipmaker that also controls the runtime (NVIDIA with CUDA) is best positioned; those

that sell only silicon face margin compression when OS-level abstractions make hardware interchangeable.

Hyperscaler Strategies

AWS, Azure, and GCP have already built their differentiation around managed AI services — SageMaker, Azure ML, Vertex AI. Their advantage is trust, scale, and enterprise relationships.

The next battleground is *agent hosting*: platforms that manage agent lifecycle, secure tool access, and observability at scale. The hyperscaler that offers the best agent-runtime-slash-OS combination will lock in the highest-margin AI spend.

Today's managed-container offerings (ECS, AKS, GKE) are the staging ground. Tomorrow's agent-native offerings will replace container control planes with agent-orchestration control planes.

ISVs and the SaaS Transition

Independent software vendors face a dual pressure:

- Downward pressure from agent-generated alternatives that compose existing APIs rather than pay for bundled features.
- Upward pressure from hyperscalers that want to move ISV relationships to their own marketplace (AWS Marketplace, Azure).

ISVs that survive will be those that *expose clean, granular APIs* and become indispensable tool nodes inside agent workflows. ISVs that cling to desktop binaries and per-seat licensing will cede relevance.

Open-Source Ecosystem

Open source has already won the infrastructure layer: Linux, Kubernetes, PostgreSQL, Redis, Hugging Face models, LangChain, Llama. The agent-native OS layer is still open for competition.

If an open-source AI-native OS emerges (perhaps a Linux distro with agent syscalls and a community agent runtime), it could accelerate adoption and prevent vendor lock-in. If the OS layer closes behind proprietary vendor ecosystems, governance and interoperability will suffer.

Enterprise IT and Procurement

Enterprise IT procurement is built around annual contracts, seat-based licenses, and procurement cycles measured in quarters. Agentic AI destroys those rhythms. An agent evaluates vendors, runs proof-of-concept workflows, and renews subscriptions autonomously. Churn becomes algorithmic and continuous.

IT departments must redesign procurement around:

- Token and API budgets rather than seat counts.
- Continuous capability auditing rather than annual feature reviews.
- Real-time cost attribution across agents, teams, and projects.

Vendors who speak the language of CIOs and procurement officers will win the enterprise transition.

Developer Experience and Tooling

Developer experience has been dominated by desktop IDEs, local runtimes, and cloud consoles. Agent-native development shifts toward:

- Agent-first debugging: reasoning traces, tool-call lineage, memory-state inspection.
- Capability-scoped SDKs: frameworks that make delegation and scoping trivial.
- Simulator environments for testing multi-agent interactions without live production risk.

The IDE of the 2030s will look less like VS Code and more like an agent-control center with live telemetry, budget monitors, and policy editors.

The Platform Bet Landscape

Table 3 — The Platform Bet Landscape.

Layer	2026 Leaders	2035 Likely Winners	Risk
Inference compute	NVIDIA, AMD, Intel	NVIDIA dominant if AI scheduler aligned; disruption if OS abstraction reduces hardware differentiation	High
Agent runtime	OpenAI, Anthropic, open-source	2–3 dominant runtimes plus specialized verticals	Medium
Cloud agent hosting	AWS, Azure, GCP	Possible new entrant if agent-native hosting creates a new platform wedge	Medium
Enterprise agent integration	Salesforce, Microsoft, SAP	Companies that own data and workflow context survive	Low
Development tools	LangChain, open-source	Standardized agent SDKs inside OS or dominant frameworks	Medium
Edge / device AI	Qualcomm, Apple, MediaTek	Apple and Qualcomm strong; Android openness could enable diverse agents	Medium

The highest-stakes bet is the *agent runtime / OS boundary*. Whoever controls that boundary controls the margin layer for the next computing platform.

Chapter 15 — Economic Consequences

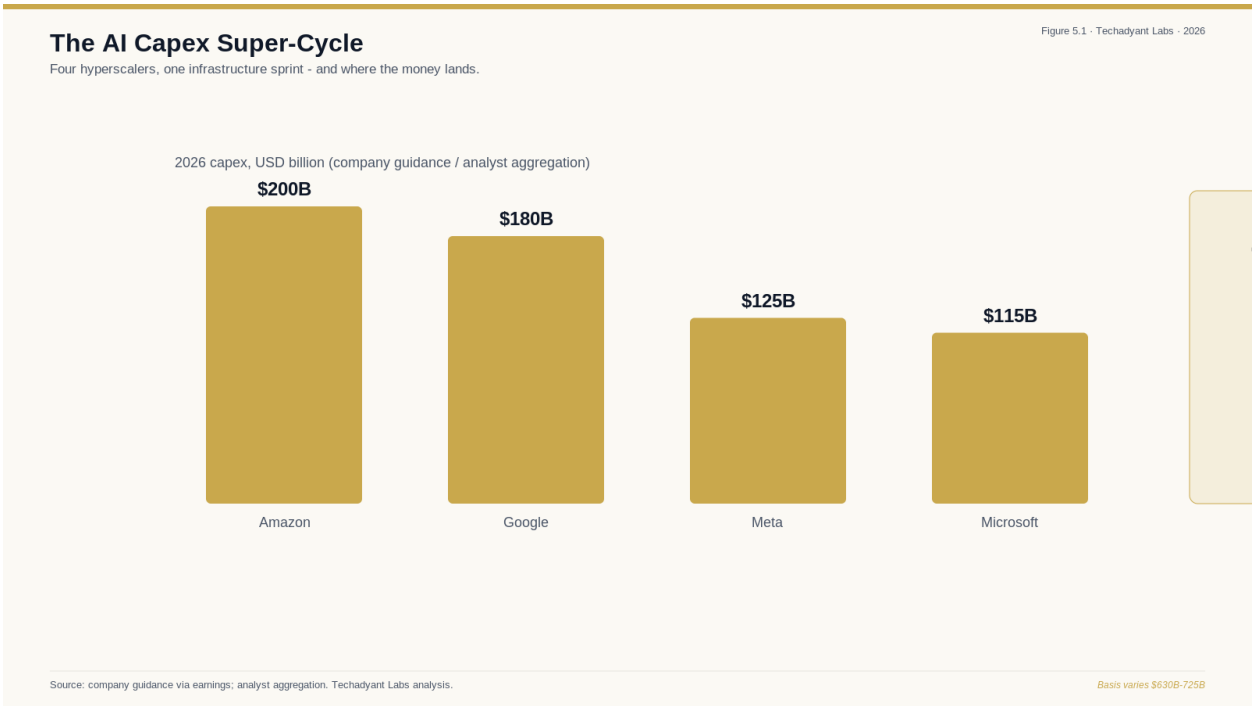

Figure 11 — The AI capex super-cycle: four hyperscalers, ~$725B in 2026.

Chapter thesis: AI-native compute is moving revenue, margin, and risk across the OS and semiconductor stack. The shift is measurable in capex commitments, licensing pressure, productivity claims, and access to capital.

New Value Chains

The transition from application-centric to agent-centric computing reconfigures value. The highest-value activities shift from bundling features to managing inference, orchestrating tools, securing machine identities, and optimizing memory at scale.

New entrants are attacking from above and below. Hyperscalers add agent platforms to existing cloud stacks. Open-source projects provide kernel-level capabilities that make agent orchestration more accessible. Traditional software vendors defend installed bases while their margins erode.

AI Infrastructure Spending

Global AI infrastructure spending accelerated sharply in 2024–2026. Hyperscalers raised capex guidance as inference demand exceeded expectations. Semiconductor revenue patterns show inference chips growing faster than training chips.

This spending is not evenly distributed. The majority flows to a small number of hardware vendors, packaging specialists, and cloud regions with power and connectivity advantages. India's share is small; as a share of GDP, it is negligible.

Operating System Economics

Operating systems have historically derived value from distribution, ecosystem lock-in, and enterprise support. The agent era does not eliminate these dynamics, but it changes where margin concentrates.

If orchestration moves upward into runtime platforms, the OS becomes more commoditized. If inference moves downward into specialized hardware or the cloud, the OS becomes thinner still. The surviving value will likely go to platforms that control meaningful compute or memory access.

Software Licensing Models

SaaS and subscription models are being stretched. Agents consume APIs unpredictably: inference calls, tool accesses, and memory operations each carry variable costs. Traditional per-seat licensing does not map cleanly to usage patterns.

Pricing innovation is emerging: token-based contracts, inference-tiered plans, and result-oriented billing. These models favor vendors with scale, because uncertainty and cost volatility are absorbed upstream.

Productivity Implications

Enterprises expect AI to improve productivity. Measuring the effect is difficult because productivity gains depend on adoption depth, workflow redesign, and agent reliability.

Early evidence suggests professional knowledge work is most affected: coding, analysis, document drafting, and customer support. In India, the potential is high given the large English-speaking knowledge workforce, but the transformation depends on AI infrastructure access and agent maturity.

Emerging Markets

Emerging markets face a narrower set of choices than developed economies. Hyperscale cloud is globally available; local compute and data infrastructure are not. Licensing and support channels are tuned for large enterprise customers.

For India, the economic consequence depends on whether the country builds indigenous compute, memory, and software infrastructure before agent workloads become critical. If it does not, the country will remain a consumer of foreign AI infrastructure, replicating the enterprise software dependency dynamic at a higher layer.

Chapter 16 — Agent Economics

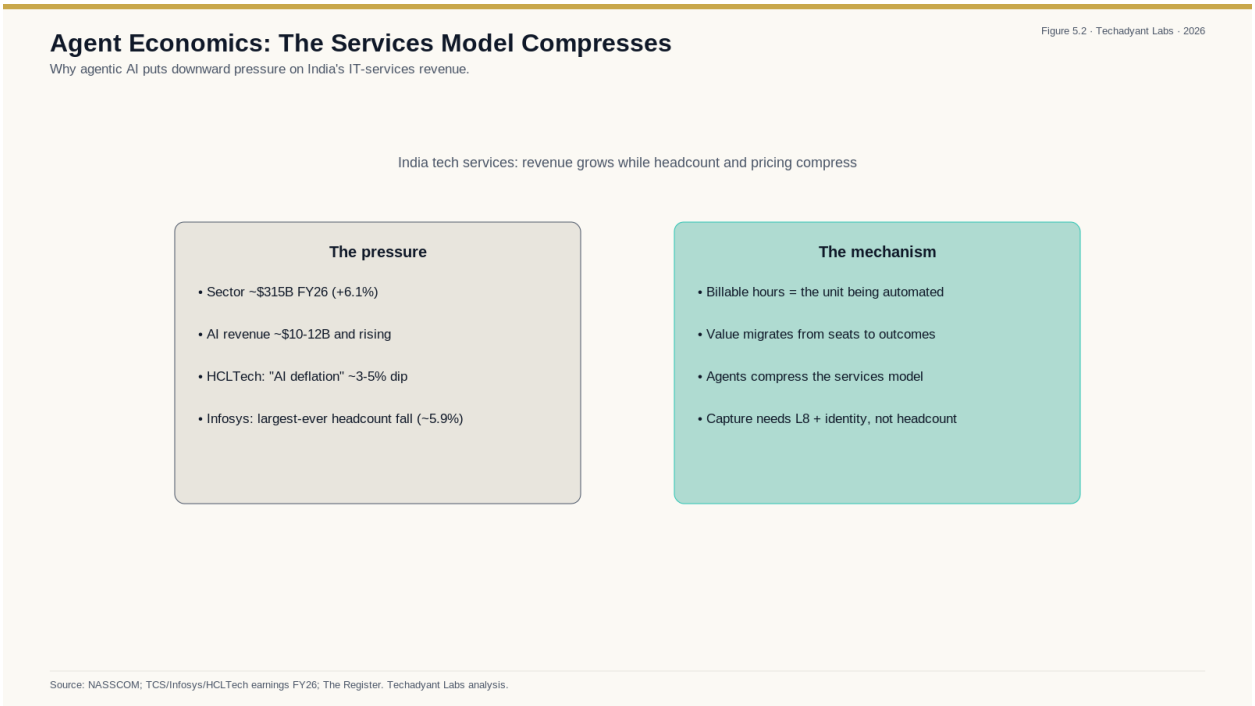

Figure 12 — Agent economics: why the services model compresses.

Chapter thesis: Agent-native computing introduces new economic units: the autonomous task, the agent employee equivalent, and the agent-labor market. These units require new pricing models, productivity frameworks, and competitive analysis. Understanding agent economics is essential for enterprise planning, investment decisions, and India's service-sector strategy.

What Is an Agent Employee?

An agent employee is an autonomous system that performs a recurring, bounded set of tasks without per-step human intervention. It has identity, access rights, memory, and measurable output.

Unlike a human employee, an agent employee:

- works continuously without fatigue,
- scales horizontally without onboarding,
- requires infrastructure investment rather than salary and benefits,
- produces auditable output tied to delegation policies.

The comparison is not sentimental; it is economic. Enterprises evaluating automation investments need to compare cost, reliability, and risk across human and agent labor pools.

Agent Labor

Agent labor refers to the aggregate pool of task executions performed by autonomous or semi-autonomous agents. Labor volume is measured in agent transactions per day—the number of tool calls, inference operations, memory reads/writes, and orchestration cycles completed across an organization.

Current estimates and projections:

Table 4 — Agent Labor.

Metric	Estimate	Source
Enterprise apps embedding AI agents by end-2026	40%	Gartner 2025
India tech-sector AI revenue (FY26)	~USD 10–12B of ~USD 315B total	NASSCOM (attributed)
Enterprise agent transactions/day (projected)	Growing toward billions globally	Analyst extrapolation from cloud inference growth
Agent ticket deflection in ITSM	>50% for routine categories	ServiceNow enterprise deployments

The agent labor pool is not evenly distributed. Enterprises with mature API infrastructure, clean data, and regulator-acceptable governance will adopt faster. Those with legacy silos and integration debt will lag.

Agent Productivity

Agent productivity is best measured at the task level, not the business level, because agents optimize specific workflows rather than organizations. The relevant unit is:

cost per autonomous task = (inference cost + tool cost + memory cost + orchestration overhead + error-recovery cost) / successful completions

Early data points:

- Enterprise agent pilots report 2x–5x speedups for routine structured tasks
- Code generation agents reduce implementation time by 30–40% in measured workflows
- ITSM agents deflect >50% of routine tickets
- Customer-service agents reduce average handle time by double-digit percentages

These are early numbers. A more rigorous productivity framework will emerge as standardized task benchmarks appear.

Pricing Models for Agent Work

Traditional per-seat licensing does not capture agent economics. New pricing patterns emerging:

34. Token-based contracts: pricing scales with inference consumption
35. Inference-tiered plans: volume tiers for throughput-heavy workloads
36. Result-oriented billing: payment tied to successful task completion
37. Capacity reservations: committed-use purchasing for agent runtime

The shift favors vendors with scale because cost volatility is absorbed upstream. Enterprises will prefer predictable agent budgets over unpredictable token bills.

Enterprise Cost Comparison: Human vs. Agent Labor

Table 5 — Enterprise Cost Comparison: Human vs. Agent Labor.

Task Type	Human Cost (India, annualized equivalent)	Estimated Agent Cost (annualized)	Notes
Tier-1 IT support ticket resolution	2–4 LPA per agent	0.5–1.5 LPA equivalent in compute	Deflection rates >50%
Code review and bug triage	6–15 LPA per senior dev	1–4 LPA equivalent in inference + runtime	Early-stage data
Document drafting and summarization	3–6 LPA per analyst	0.5–2 LPA equivalent	High volume, low ambiguity
Compliance checking	4–8 LPA per officer	1–3 LPA equivalent	Audit trails are a requirement

LPA = lakhs per annum. These are directional, not precise. They illustrate where agent economics become competitive and where human judgment remains essential.

Agent Economics and India

India's knowledge-services sector is both the opportunity and the risk. The opportunity: India can deploy agent-native services at lower cost than most economies, leveraging its engineering base and India Stack integration. The risk: if India remains a consumer of foreign AI infrastructure, much of the value will leak as inference fees, cloud charges, and licensing to offshore vendors.

Capturing agent-economic value in India requires:

- sovereign compute to keep inference costs domestic,
- open runtime standards to avoid lock-in,
- India Stack integration to reduce per-transaction coordination cost,
- agent-native public services that export the model.

Labor Market Impact

Agent automation will restructure knowledge work without eliminating it. Routine structured tasks—ticket classification, document retrieval, status updates—are the first to be automated. Creative, strategic, and relationship-intensive work remains human-led. The new equilibrium will require:

- reskilling for agent supervision and prompt engineering,
- governance roles for agent policy and audit,
- domain expertise for tool selection and agent orchestration design.

Chapter 17 — Winners and Losers in the Agent-Native Era

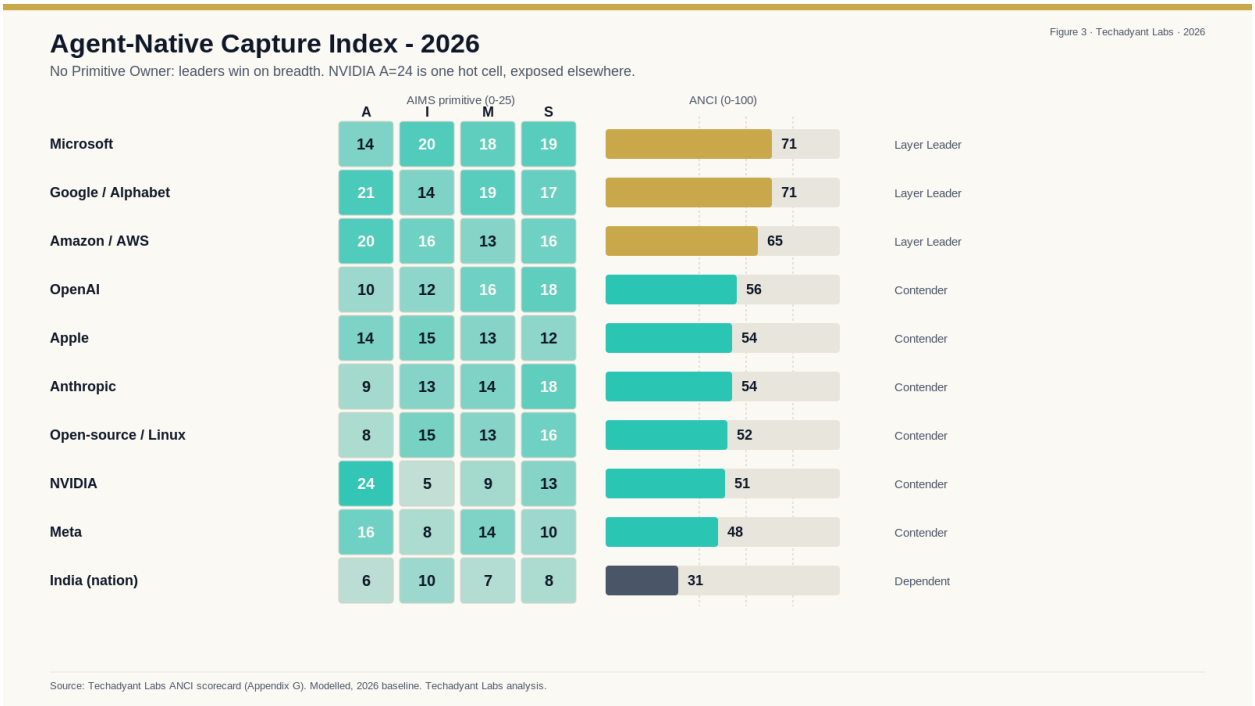

Figure 13 — The Agent-Native Capture Index, 2026: no Primitive Owner; leaders win on breadth.

Chapter thesis: Every platform transition creates clear winners and losers. The agent-native transition is no exception. This chapter ranks likely winners and losers on the report's **Agent-Native Capture Index (ANCI)** — how much of the post-application stack each actor controls across the four AIMS primitives. The central finding: in 2026 there is no Primitive Owner; value accrues to actors that span the stack, and the silicon owner is paradoxically exposed above its own substrate.

Evaluating Winners Using ANCI

The four AIMS primitives (see the framework chapter) are where value concentrates as the application dissolves:

- A — Accelerated inference (silicon + serving substrate, PAS L1–L2)
- I — Identity & delegation (agent trust issuance and revocation, L5)
- M — Memory & context (the persistent context layer agents reason over, L3)
- S — Scheduling & orchestration (how agent work is composed and run, L4/L6)

Each primitive is scored 0–25; the sum is the ANCI (0–100), placing an actor in one of five tiers — Primitive Owner (80–100), Layer Leader (60–79), Contender (40–59), Dependent (20–39), Absent (0–19). The full scorecard and rationale are in Appendix G; the headline is that the hyperscalers lead by *breadth* (they touch all four primitives), while specialists — including NVIDIA — score lower despite owning a primitive outright, because they are thin elsewhere.

Winners

The hyperscalers — Microsoft (ANCI 71), Google (71), AWS (65): Layer Leaders

The clearest winners are the three cloud platforms, because they are the only actors that span all four primitives. Microsoft pairs an agent-identity platform (Entra Agent ID) with Copilot memory and the Windows Agent Runtime; Google is the most vertically integrated (TPU + Gemini + the A2A protocol); AWS layers Bedrock AgentCore Identity and runtime over Trainium and the largest cloud. None owns a single primitive as completely as NVIDIA owns silicon, but breadth across the stack is what the index rewards.

NVIDIA (ANCI 51): Contender, A-concentrated

NVIDIA owns Accelerated inference almost outright (A=24): the dominant accelerator, CUDA, and the Dynamo/Run:ai/Grove serving-and-scheduling stack, with Blackwell ~70% of its data-centre compute revenue. Yet its ANCI is only Contender-level because it is thin on identity (I=5) and memory (M=9) and exposed above its own substrate — the paradigmatic concentrated profile. NVIDIA collects the toll on A; it does not (yet) own the layers where agents are governed and remembered.

Model labs — OpenAI (56), Anthropic (54): Contenders concentrated in S

Both score high on Scheduling/orchestration (Agents SDK and Operator for OpenAI; the MCP standard, now under the Linux Foundation, for Anthropic) and on Memory, but are Dependent on Accelerated inference — they rent compute rather than own it.

Open-source / Linux ecosystem (ANCI 52): Contender

Owns the open identity standard (SPIFFE/SPIRE) and the orchestration substrate (LangChain/AutoGen/CrewAI, Kubernetes), and is the most probable base for hybrid AI kernels and sovereign stacks — but is CUDA-dependent on A.

Identity, memory and packaging chokepoint vendors

Identity/security vendors extending into machine principals and delegation tokens capture rising I-layer value. HBM and advanced-packaging suppliers (SK Hynix, TSMC CoWoS, Samsung) are the binding L1 constraint — the chokepoint the whole stack waits on. India-adjacent **ATMP / advanced packaging** is the realistic Indian capture point in this tier (see Part VII).

Losers

Traditional SaaS vendors (ANCI Dependent/Absent)

Per-seat, application-centric revenue compresses as agents consume APIs rather than applications. The application *is* the unit being disintermediated. Salesforce and ServiceNow are **transitional**, not doomed — Agentforce and ServiceNow AI Agents lift their S score — but their core sits in the layer ANCI penalises.

Standalone applications (Absent)

Desktop applications with no orchestration layer, no API strategy, no agent integration. Their user base migrates to intent-driven interfaces.

Legacy endpoint-centric security (Absent on I)

Signature-based, perimeter tooling built for human-initiated execution. Agent-native threats — prompt injection, memory poisoning, agent impersonation — fall outside its detection model, and the identity anchor it relies on is being inverted toward agent trust.

Cloud-native-only strategists (sovereignty exposure)

Enterprises and nations that place all inference and memory in foreign hyperscaler infrastructure accept dependency risk — a strategy failure, not a technology one. This is the negative case that motivates the India L8 argument.

ANCI Score Summary

Scores 0–25 per primitive; ANCI 0–100. Full rationale in Appendix G. (`[modelled]`, 2026 baseline.)

Table 6 — ANCI Score Summary.

Actor	A	I	M	S	ANCI	Tier
Microsoft	14	20	18	19	71	Layer Leader
Google / Alphabet	21	14	19	17	71	Layer Leader
Amazon / AWS	20	16	13	16	65	Layer Leader
OpenAI	10	12	16	18	56	Contender
Apple	14	15	13	12	54	Contender
Anthropic	9	13	14	18	54	Contender
Open-source / Linux	8	15	13	16	52	Contender
NVIDIA	24	5	9	13	51	Contender (A-conc.)
Meta	16	8	14	10	48	Contender
India (nation, 2026)	6	10	7	8	31	Dependent (L8 +)
Traditional SaaS	4	6	5	6	21	Dependent (falling)

Chapter 18 — The Enterprise Transition Timeline

Chapter thesis: Enterprises will not replace their OSes on a fixed schedule. Migration will follow workload-specific logic across a phased timeline: first the agent layer, then the runtime layer, then kernel changes. Understanding this timeline bridges theory and adoption, and helps India align its sovereign-stack investments with enterprise demand.

2026–2028: Agent Layer Dominance

What changes: enterprises adopt agent runtimes above existing OSes. Frameworks such as LangChain, CrewAI, AutoGen, and vendor-specific agents (Microsoft Copilot, Salesforce Agentforce, ServiceNow AI Agents) become the primary interface for knowledge work, IT operations, and customer service.

What does not change: the underlying OS, security model, and process abstractions remain largely intact. Agent orchestration runs in user space, managed by enterprise IT as a new platform layer.

India implication: this is the window to build agent-native public services and enterprise products atop existing cloud and OS infrastructure. No forklift required.

2028–2031: Runtime Layer Maturity

What changes: agent runtimes become more capable and more demanding. Requirements for vector memory, semantic context, tool registries, and policy engines force runtime platforms to expose new APIs. Linux distributions and cloud providers add inference-aware scheduling and memory management.

What does not change: the kernel remains legacy-compatible. New functionality is additive.

India implication: this is the moment to contribute to open AI runtime standards. India's population-scale public digital infrastructure gives it unique requirements that can shape runtime design globally.

2031–2035: Kernel and OS Redesign

What changes: accumulated pressure from inference workloads, memory bandwidth limits, and security model gaps forces kernel-level changes. Hybrid AI kernels (Linux, possibly Windows) expose inference scheduling, semantic memory APIs, and machine-principal identity as system services.

What does not change: the basic concept of an OS as the hardware abstraction layer. It becomes an AI-aware OS, not a post-OS world.

India implication: if India has invested in Linux expertise, sovereign runtime projects, and packaging/assembly capacity, it can participate in kernel-level design. If it has not, it will import these capabilities.

Why the Phases Matter

The three-phase model prevents two common errors:

38. Overestimating near-term disruption. Enterprises will not forklift legacy OSes in two years.
39. Underestimating near-term opportunity. The agent layer is already here, and India can capture value in runtime and public-service design before kernel changes arrive.

Part VI — Geopolitics & Sovereignty

Chapter 19 — Geopolitical and Sovereignty Implications

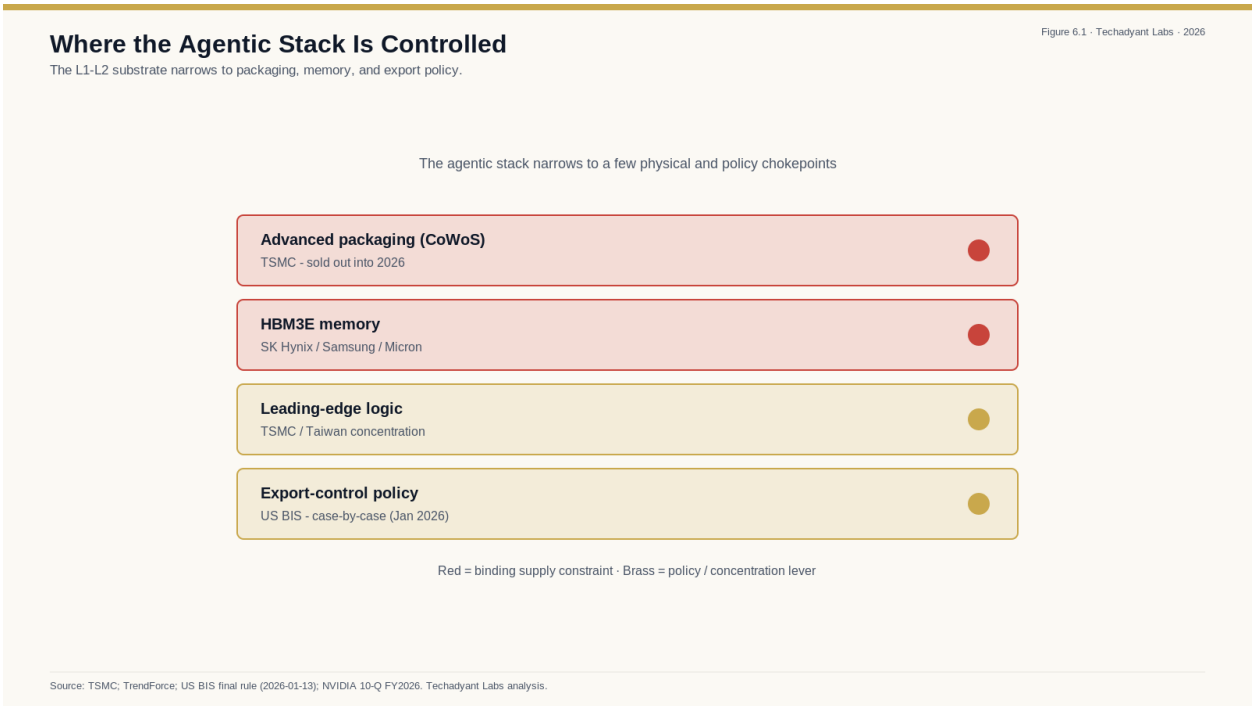

Figure 14 — Where the agentic stack is controlled: packaging, memory, and export policy.

Thesis

Agentic AI and AI-native computing create new geopolitical fault lines around compute sovereignty, data governance, model control, and agent-runtime dominance — issues that traditional digital-sovereignty frameworks did not anticipate.

Compute Sovereignty

The ability to train and run AI models depends on access to:

- Advanced semiconductor fabrication (currently ~90% in TSMC, Samsung, Intel fabs).
- High-bandwidth memory (HBM), currently concentrated in SK Hynix and Samsung.
- Power infrastructure capable of supporting multi-hundred-megawatt datacenter clusters.
- Cooling and water resources near datacenter sites.

Countries without domestic access to this chain — including most of Europe, India, and the Global South — are dependent on a handful of East Asian fabs and US-headquartered software stacks. Compute sovereignty is becoming as strategically important as energy sovereignty.

Data Governance

An agent's output quality depends on data: training data, retrieval corpora, personal memory, real-time tool inputs. Countries that control data flows — through privacy law, localization requirements, or platform dominance — effectively control agent quality.

The EU AI Act and GDPR-style enforcement create a distinctive European path for agent training and deployment. India's DPDP Act is moving in a similar direction. Both create opportunities for domestic agent platforms that respect local data rules, at the cost of higher integration friction with global model providers.

Model Dominance and Dependency

Foundation models are increasingly strategic assets. A country that depends on a foreign model for its agents — for healthcare, education, governance, defense — has ceded a layer of cognitive sovereignty.

The pattern is already visible:

- China promotes domestic models (DeepSeek, Ernie, Qwen) over OpenAI and Anthropic.
- Europe invests in open-weight models (e.g., through the European Open Source AI effort).
- India is developing its own Indic-language models and exploring sovereign AI infrastructure.

The risk is not that foreign models are malicious, but that foreign model providers control the terms of use, pricing, and governance — creating dependency that is hard to unwind.

Agent Runtime as Infrastructure

If the agent runtime becomes the OS of the next era, the runtime vendor becomes the strategic bottleneck. A runtime with majority market share decides:

- Which tools and APIs are easy to integrate vs hard.
- Which security and audit standards are enforced.
- Which data is retained, shared, or monetized.
- How an agent's memory can be migrated between devices and clouds.

This is the modern equivalent of Windows or Android market dominance — but with more intelligence and behavioral control built in.

Sovereignty Strategies

Countries and blocs are responding with a mix of:

- Subsidized domestic compute (EU, India, US CHIPS Act).
- Data localization and privacy frameworks (GDPR, DPDP Act).
- Investment in domestic model development (IndiaAI Mission, China sovereign AI programs).
- Open-weight mandates (encouraging open-source model adoption to reduce platform dependence).

India's approach is notable because it combines compute investment, data governance, and domestic model development in a single coordinated strategy — the most holistic of any large democracy.

Digital Colonialism and the Global South

The classic digital-colonialism pattern: a wealthy country or company builds infrastructure, extracts data, and locks the host country into dependency. Agentic AI adds a new dimension: the extracted asset is *behavioral intelligence* — not just browsing data or location data, but goal-oriented, context-rich agent interaction data that is even more valuable for training future systems.

Countries in the Global South risk repeating this pattern if they:

- Adopt foreign agent platforms as default rather than as alternatives.
- Allow foreign model providers to train on local language, cultural, and government data without reciprocity.
- Fail to invest in domestic AI talent and infrastructure.

Reciprocity frameworks — requiring foreign AI providers to partner with domestic entities, share training data benefits, or transfer technology — will become the new negotiating table.

Cyber Conflict and Agent Tooling

State-sponsored cyber operations will adopt agentic AI for reconnaissance, social engineering, and payload generation. Offensive agents can probe vulnerabilities, generate polymorphic malware, and adapt countermeasures in real time. Defensive agents will be deployed to match.

The result is an AI-driven arms race at the software layer, with OS-level sandboxing, capability tokens, and audit logs as the primary defensive line. Governments will demand OS-level backdoors or agent audit hooks that conflict with privacy frameworks — reigniting the Crypto Wars of the 1990s.

Standards, Alliances, and Governance

The governance landscape is young but developing:

- NIST AI RMF and EU AI Act provide regulatory scaffolding.
- G7 Hiroshima AI Process and Bletchley Park Summit are early diplomatic coordination.
- ISO / IEC working groups are beginning to define AI safety and risk-management standards.

What is missing is a *systems integration* standard: how do agent runtimes, OS primitives, model APIs, and audit infrastructure interoperate securely across borders? That integration layer will be the five-year governance prize.

Part VII — India Special Section

Chapter 20 — India's Opportunity in the AI-Native Computing Era

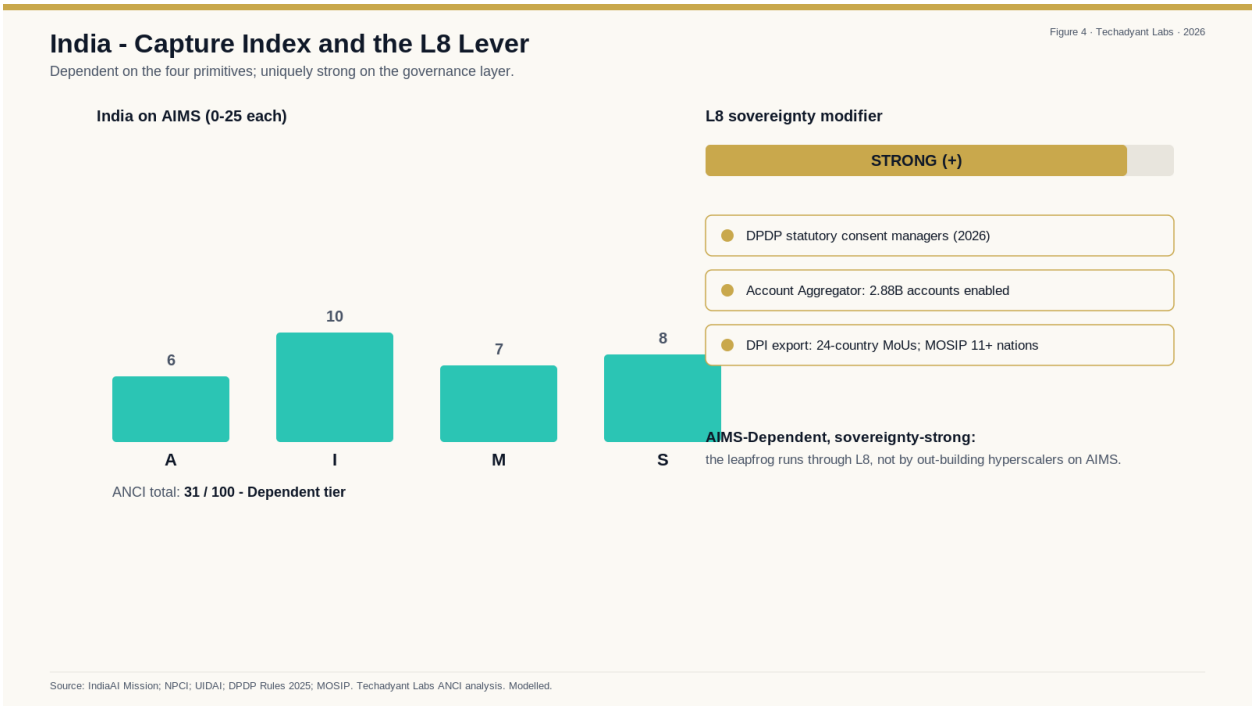

Figure 15 — India on the capture index: AIMS-dependent, sovereignty-strong.

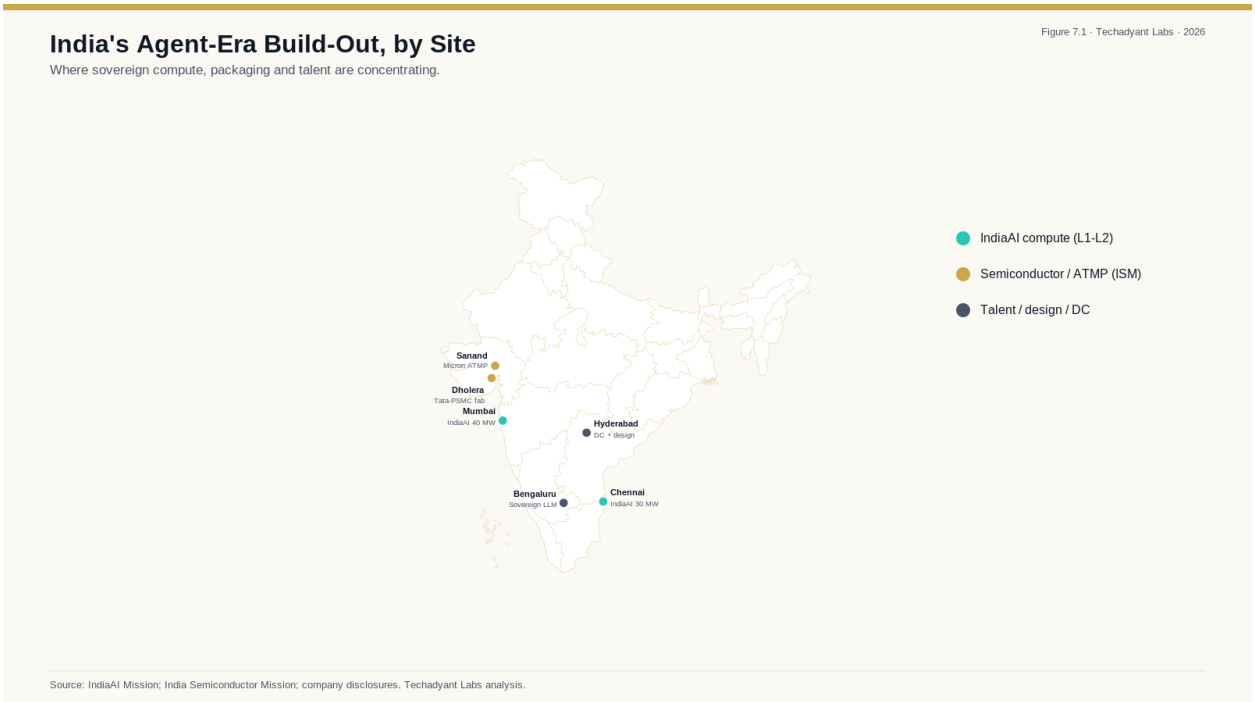

Figure 16 — India's agent-era build-out, by site.

Thesis

India can capture disproportionate value from the AI-native transition by leveraging its English-proficient knowledge workforce, growing domestic compute capacity, semiconductor policy momentum, and digital public infrastructure — if execution keeps pace with the 2026–2035 window.

The India Stack as Agent Infrastructure

India has already built world-class digital public infrastructure: Aadhaar identity, UPI payments, ONDC commerce network, DigiYatra travel, and CoWIN health records. These systems share a common architecture: API-first, open-protocol, identity-gated, interoperable.

That architecture happens to be exactly what AI agents need to operate effectively. An agent that can:

- authenticate via ONDC-style consent layers,
- transact via UPI-style lightweight APIs,
- access services through standardized schemas,
- verify identity through Aadhaar-linked verification,

is already operating inside an *agent-ready* environment. India Stack is, unintentionally, proto-agent-native infrastructure.

Knowledge Workforce as the Differentiator

The global AI debate centers on "will AI replace knowledge workers?" India's answer is: AI makes Indian knowledge workers *more valuable*, not less.

India graduates:

- ~2.5 million STEM graduates per year (NASSCOM / AISHE data).
- A large English-proficient population ranked second globally by EF EPI.
- Deep software engineering talent proven in global product studios (Infosys, TCS, Wipro, and 200+ global-capability centers).

What changes with agentic AI is the *productivity multiplier*: a single Indian knowledge worker equipped with an agent assistant can now ship the output that previously required a team.

India's per-capita productivity in knowledge-work domains could rise faster than in countries with smaller, more expensive labor pools.

Domestic Compute: IndiaAI Mission and PLI

India is actively building sovereign AI compute:

- IndiaAI Mission (approved March 2024, INR 10,300 crore) includes a multi-petaflop GPU compute cluster accessible to researchers and startups.
- Semiconductor PLI (PLI for semiconductors and display fabs) offers fiscal incentives for fab and OSAT investments.
- Datacenter policy (Digital Personal Data Protection Act framework) encourages data-center growth in tier-2/3 cities.

If the IndiaAI compute cluster reaches operational scale by 2027–2028, it will give Indian researchers and startups access to frontier inference at domestic cost — reducing the foreign-exchange and latency tax that currently makes advanced AI experimentation expensive.

Startup Ecosystem Readiness

India is the world's third-largest startup ecosystem:

- ~140,000 recognized startups (DPIIT data).
- Rapid growth in deep-tech and AI-focused ventures.
- Strong fintech infrastructure already proven globally through UPI.

The gap is frontier AI access. Most Indian AI startups today fine-tune existing open-weight models or use global cloud APIs. To build agent-native products at platform scale, they will need:

- Affordable domestic GPU inference.
- Domestic model training and alignment capability.
- Open-ecosystem tooling that matches global LangChain/CrewAI/AutoGen parity.

Manufacturing and Hardware Opportunity

AI-native computing is hardware-intensive:

- GPU and NPU demand is growing 50–70%+ annually (NVIDIA/AMD roadmaps).
- HBM and advanced packaging are the new chokepoints.
- Edge inference devices (smartphones, embedded NPUs) are enormous consumer markets.

India's manufacturing and electronics sector could capture a meaningful slice of this demand:

- Semiconductor assembly, test, and packaging (OSAT) under PLI incentives.
- Edge device design and manufacturing, leveraging India's strong mobile-phone manufacturing base.
- Rare-earth and advanced-material processing (with appropriate environmental governance).

These are ten-year buildouts. The window opens now.

Language and Cultural Context

Foundation models trained primarily on English-language internet data are imperfect tools for Indian languages, regulatory contexts, and cultural nuance. An agent operating in India must understand:

- Multiple official languages and scripts at high fidelity.
- Federal and state-level regulatory heterogeneity.
- Local business conventions (ONDC catalog formats, GST compliance, regional payment preferences).
- Cultural context in customer interaction, legal documents, and health records.

Domestic Indian AI models and fine-tuning pipelines — BharatGPT, Airavata, TechMahindra's Indus, and others — are closing this gap. When agent-native platforms incorporate these models as native options, India-built agents will outperform generic agents in local productivity.

Policy Gaps

For India to realize this opportunity, policy must close four gaps:

40. Compute access: IndiaAI cluster must deliver frontier-class inference at cost-competitive rates for startups and researchers.
41. Data governance: DPDP Act implementation must balance privacy with responsible training-data access that benefits domestic model builders.
42. Talent pipeline: University curricula must include agent-orchestration, tool-use engineering, and capability-based security — not just traditional software engineering.
43. Export strategy: Indian agent companies need a clear path to global markets. Domestic policy must prevent talent drain without walling off the ecosystem.

Asia Competitive Landscape

India is not the only Asian economy betting on AI-native opportunity:

- China has full-stack domestic AI capability and state-backed model development, but regulatory and export restrictions limit global reach.
- Japan has hardware strength (Sony, TDK, semiconductor materials) and enterprise AI adoption, but a legacy software ecosystem less suited to rapid agent-native transition.
- South Korea dominates memory and display supply, with strong government AI investment, but is highly dependent on US China-tariff dynamics.
- Taiwan controls advanced-node fabrication (TSMC), making it the geopolitical chess piece every player needs but none can afford to alienate.

India's comparative advantage is *scale and openness*: a large English-speaking market, a democratic governance model that supports open-source, and manufacturing capacity that can expand under stable policy.

India2040: Beyond Apps

The India of 2040 may not look like a nation of 500 million app users. It may look like a nation of 500 million agent-enabled citizens — accessing services, transacting, learning, and working through intent-driven, agent-mediated workflows.

That is not a loss; it is an upgrade. Apps were a necessary abstraction for 2010s mobile infrastructure. Agents are the natural abstraction for 2030s compute infrastructure. India that builds the agent layer — infrastructure, models, governance, talent — will capture outsized value from the transition.

Where India Can Capture Value: the AIMS Read-Through

It is worth being exact about *where* in the stack the opportunity sits, because the honest answer is not "everywhere." Mapped onto the report's four AIMS primitives, India's realistic capture is uneven. On **Accelerated inference** it is a fast-scaling consumer, not an owner: the IndiaAI compute cluster (34,000 GPUs, scaling to 100,000) and a 1.7 GW data-centre base are real, but the silicon is imported and the binding chokepoint — advanced packaging and HBM — sits offshore. India's semiconductor entry (ISM's ten projects, ₹1.60 lakh crore, led by Micron's Sanand ATMP and the Tata–PSMC Dholera fab) is deliberately weighted toward **assembly, test and packaging** — the chokepoint layer — rather than leading-edge logic, which is the correct first move but a decade from frontier capability. On **Memory** and **Scheduling**, India owns no dominant implementation; its sovereign models (Sarvam, BharatGen Param2) are mid-size and run on others' runtimes.

Where India can lead is **Identity** and the governance layer above the four primitives. The country already operates the world's largest consent-governed identity and data-sharing rails — Aadhaar at near-universal adult coverage, the Account Aggregator at 2.88 billion enabled accounts, and a statutory consent-manager regime arriving under the DPDP Rules. The opportunity, in framework terms, is to convert an L8 advantage into an L5 (identity) and L7 (interaction) capture, building the agent-trust and intent layers on rails India controls — and exporting them. That is a narrower, more defensible claim than "India captures the AI opportunity," and it is the one the evidence supports.

Chapter 21 — Could India Leapfrog the Application Era?

Thesis

India has structural advantages — digital public infrastructure, a young mobile-first population, and policy momentum — that could allow it to skip the application-era middleman and move directly to agent-native interaction, but cultural, institutional, and infrastructure gaps make the leap contingent on deliberate action.

What Leapfrogging Requires

In technology history, leapfrogging means skipping an entire generation of infrastructure because the new generation is so much better that the sunk cost of the old generation is no longer worth bearing.

Examples:

- Many developing countries skipped landline telephony and went directly to mobile.
- Some African nations skipped wireline banking and adopted mobile money at scale.
- India skipped credit-card mass adoption and went directly to UPI.

Leapfrogging the application era would mean skipping native app-based service delivery and going directly to agent-mediated intent-based interaction. It requires:

- Agent infrastructure available to the mass market.
- Services designed for agent access rather than app UI.
- Digital identity and consent frameworks that agents can use natively.
- A cost structure where agent access is cheaper than app ownership.

Mobile Penetration and App-First Habits

India has ~750 million smartphone users and a large, young, mobile-first population. Apps are deeply embedded: ONDC, UPI, Aadhaar-linked services, education apps, health apps.

The leapfrog challenge is *overcoming incumbency*. Users have learned to open specific apps for specific tasks. Switching to agent-native interaction requires:

- Agent UX that is more trustworthy and usable than app UX.
- Agent performance that is more reliable than human-operated services.
- Economic incentive (lower cost, better outcomes) that justifies the switch.

This is not impossible. Indian users adopted UPI rapidly because it was demonstrably better than cash and card transaction costs. A similar adoption curve is plausible for agent services if the value proposition is clear.

Digital Public Infrastructure and Agent-Native Services

India Stack — Aadhaar, UPI, ONDC, DigiLocker, CoWIN — provides the infrastructure atomic primitives:

- Aadhaar eKYC → agent-verifiable identity consent.
- UPI intent model → agent-accessible payment rail.
- ONDC network protocols → agent-accessible commerce APIs.
- DigiYatra facial recognition → agent-ready travel check-in (with privacy caveats).
- Mifos / account aggregator framework → agent-accessible financial data with consent.

If these primitives get standardized, documented, and open-sourced for agent consumption at a government-platform level, Indian startups can build agent-native services without reimplementing identity, payments, and consent infrastructure.

Leapfrog Barriers

Four structural barriers must be addressed:

Trust deficit. Indian consumers are cautious with new financial technology. Agent-mediated transactions require a higher trust threshold than human-mediated ones. A novel transaction model (agent buys on your behalf) must demonstrate transparency, reversibility, and dispute resolution before mass adoption.

Language and literacy. Agent interfaces require natural-language literacy. India has ~20+ official languages, with functional literacy varying widely by state. Agents that operate in

multiple languages and multimodal modes (voice, image) will be essential — but high-quality Indic-language agent models are not yet production-grade.

Computing device asymmetry. Agent-native interfaces work best on capable smartphones or dedicated devices. India's smartphone base is large but not uniformly high-end. Low-end devices cannot run local inference, and cloud inference requires consistent connectivity.

Regulatory sequencing. The DPDP Rules 2025 are now notified, but on a phased timeline — the Data Protection Board is live, while the consent-manager regime and most substantive obligations only bind in November 2026 and May 2027 respectively. That sequencing is an advantage (the consent rail agents need is arriving) and a near-term constraint (the operational detail firms must design against is not yet fully in force). Companies investing in agent-native services are building against a framework whose shape is now clear but whose enforcement is still phasing in.

Sector-Specific Leapfrog Scenarios

Government services. India has 20+ languages and 700+ districts. Delivering personalized government services at this scale through a human workforce is impossible. Agent-mediated delivery — a citizen states intent in their language, the agent handles the workflow across Aadhaar, UPI, and departmental systems — is the *only* viable architecture at scale.

Healthcare. Healthcare access in rural India is constrained by provider density. An AI diagnostic agent supported by local health workers could extend specialist-quality triage to populations that currently have no access.

Education. India's pupil-teacher ratio is among the worst globally. Personalized tutoring agents for every student — in every language, at every level — could close learning gaps faster than hiring millions of additional teachers.

Agriculture. Small farmers in India make decisions based on local knowledge and market signals. Agriculture agents that aggregate weather, market price, credit, and input data into personalized advice — delivered via voice in local language — are a logical application.

Education and Workforce Transformation

The leapfrog is only possible if India's workforce can design, operate, and govern agent-native systems. That requires:

- University-level agent-AI curricula in every major engineering institution.
- Grassroots digital literacy that includes agent-competency, not just app literacy.
- Employer willingness to retrain existing workforce rather than replace.

India's EdTech sector (Byju's, Unacademy, upGrad, Physics Wallah) is well positioned to pilot agent-AI training at national scale if public policy creates a demand signal.

Entrepreneurship and Export Potential

Indian agent startups targeting domestic infrastructure needs will acquire capabilities exportable to other emerging markets — Africa, Southeast Asia, Latin America — that face similar challenges with service delivery at scale.

Examples likely domains:

- Government-service agents (identity, payments, compliance).
- Multi-language customer-support agents for global enterprises.
- Agriculture and climate agents optimized for tropical smallholder contexts.

India's advantage is not just cost; it is *context*. Agents built for India's heterogeneity are implicitly built for global-south heterogeneity.

Timeline to Leapfrog

A plausible leapfrog sequence, with the 2026 starting point now observable rather than projected:

- 2026 (baseline, achieved): the IndiaAI Mission compute cluster is live (34,000 GPUs at INR 115–150/GPU-hour, scaling toward 100,000 by end-2026); three Mission-funded sovereign models are unveiled at the AI Impact Summit 2026 — Sarvam's twin LLMs (30B multilingual; 105B/128k-context enterprise), BharatGen's Param2 (17B Mixture-of-Experts across all 22 scheduled languages, the largest single IndiaAI grant at INR 900 crore), and Gnani.ai's Vachana voice stack; the DPDP Rules 2025 are notified, with the Data Protection Board operative and a statutory consent-manager regime due by November 2026.
- 2027–2029: Indic-language models close the gap from mid-size toward frontier on the priority Indian-language and voice tasks that matter most domestically; first wave of

agent-native government services piloted at district scale on the Aadhaar–UPI–DigiLocker–Account Aggregator rails.

- 2030–2035: agent-native transaction volume exceeds app-native volume in at least one major sector (most likely government services or commerce); India's DPI-export footprint (24 country MoUs, MOSIP across 11+ nations) extends the agent-native stack to other global-south markets.

This is aggressive but possible. The alternative — a slow, app-era-incrementing path — leaves India as a consumer of foreign agent platforms rather than a builder of domestic agent infrastructure.

The Framework Read-Through: AIMS-Dependent, Sovereignty-Strong

The leapfrog argument is precise once read through the report's framework. On the four AIMS primitives — Accelerated inference, Identity, Memory, and Scheduling — India is **dependent**: it imports the silicon (subject to the same CoWoS/HBM chokepoint as everyone else), runs mid-size rather than frontier models, and owns none of the dominant agent-runtime or memory-fabric implementations. On the report's Agent-Native Capture Index, that places India in the **Dependent tier** on raw AIMS control.

What changes the strategic read is the eighth layer of the Post-Application Stack — **L8, governance and sovereignty** — where India carries a strong positive modifier that almost no other nation does. Three facts establish it. First, the consent primitive that agent-native services require is being *legislated*: the DPDP Rules 2025 create a statutory consent-manager regime (operative from November 2026), so an agent acting on a citizen's behalf has a legal consent rail to operate within. Second, the data rails already exist at population scale and are consent-first by design: the Account Ag

Chapter 22 — The India OS Blueprint

Chapter thesis: An India-native AIOS is feasible as an engineering and policy proposition. This chapter specifies a concrete architecture—the India OS Stack—that combines open kernel, sovereign agent runtime, identity, consent, semantic memory, and compute fabric layers. The blueprint is designed for India's governance requirements, data-residency rules, and population-scale public digital infrastructure.

Architecture Overview

The India OS Stack comprises six layers:

```
India Agent Runtime
  +
India Identity Layer (Aadhaar-aware, machine-principal capable)
  +
India Consent Layer (policy-aware, audit-ready, DPDP-compliant)
  +
India Semantic Memory Layer (vector-native, sovereignty-resident)
  +
India Compute Fabric (IndiaAI nodes, edge inference, packaging assembly)
  +
Linux-based Agent Kernel (hybrid AI kernel forked from mainline)
```

Each layer is designed to be:

- open and auditable,
- deployable on sovereign compute,
- interoperable with India Stack,
- governable by Indian policy rather than foreign vendor terms.

India Agent Runtime

The runtime is the agent-native equivalent of an OS kernel for orchestration. It accepts intent, manages goals, dispatches tool calls, maintains agent memory, enforces delegation policy, and produces audit trails.

Requirements:

- MCP-compatible tool invocation (aligned with emerging standards)
- semantic memory API with retention and summarization policies
- goal registration and lifecycle management
- delegation token issuance and revocation
- audit log export for regulators and auditors

Implementation path: fork an existing open-source runtime (LangGraph, AutoGen, or similar) and harden it for public-sector and financial-sector deployment requirements.

India Identity Layer

Aadhaar provides the strongest identity foundation in India, but it was designed for human principals. The India Identity Layer extends its concepts—unique identifier, biometric verification, consent framework—to machine principals.

Requirements:

- agent ID linked to principal (citizen, business, government officer)
- delegation chain mapping with auditable provenance
- revocation and suspension semantics
- compatibility with Aadhaar-based eSign and OTP flows where human authorization is required

India Consent Layer

Consent is both a legal requirement under DPDP and a runtime primitive for agent safety. The India Consent Layer evaluates whether an agent action is authorized by the principal's policy before execution.

Requirements:

- policy expressions readable by non-technical principals
- fine-grained scope controls (data type, purpose, duration)
- real-time consent checks before high-risk tool calls
- consent audit trail exportable for regulators

This layer distinguishes India OS from generic AI runtimes: it makes consent a first-class OS primitive rather than an application-level checkbox.

Crucially, this is not a greenfield proposal. The statutory scaffolding already exists: the DPDP Rules 2025 create a formal **consent-manager** role (operative from November 2026), and the Account Aggregator framework has run a consent-first, *data-blind* sharing architecture in production since 2021 — now spanning 2.88 billion enabled accounts and 284.6 million user-linked (March 2026). The India Consent Layer is therefore an engineering extension of a legal and infrastructural primitive India has already built and scaled, not a new institution to be invented. That is the difference between a blueprint and a wish list.

India Semantic Memory Layer

Semantic memory stores context retrievable by meaning, not path. The India Semantic Memory Layer must:

- operate within Indian jurisdiction for regulated sectors
- support vector search with configurable retention policies
- provide namespace isolation between agents, tenants, and principals
- support summarization and compaction for cost and privacy management
- expose APIs compatible with open vector-store standards

Strategic importance: memory sovereignty is as important as compute sovereignty. If Indian agent state resides in foreign cloud stores, auditability and jurisdiction are lost.

India Compute Fabric

The compute fabric connects AI inference nodes, edge gateways, packaging assembly facilities, and data-center hubs. Requirements:

- sovereign inference nodes at district and regional levels
- edge inference for latency-sensitive and privacy-sensitive services
- packaging and assembly capacity near demand centers
- standardized APIs connecting runtime layers to compute scheduling

The compute fabric makes the blueprint runnable. Without it, the India OS Stack is a design document; with it, it is a deployable platform.

Linux-Based Agent Kernel

The kernel layer is a hybrid AI kernel forked from Linux mainline with:

- inference-aware scheduler policies
- unified memory management for accelerator workloads
- kernel-level support for machine-principal identities
- audit surfaces exposed to user-space governance tools
- open-source licensing for auditability and forkability

This preserves compatibility with existing Linux infrastructure while adding the agent-native primitives that pure legacy kernels lack.

Integration With India Stack

The India OS Stack must connect to existing India Stack primitives:

Table 7 — Integration With India Stack.

India Stack Primitive	India OS Integration
Aadhaar	identity resolution for human and machine principals
UPI	payment tool handle in agent tool registry
ONDC	commerce intent routing and fulfillment

India Stack Primitive	India OS Integration
DigiLocker	document retrieval with consent-aware access
CoWin	health-service agent orchestration template
eSign	agent execution with human authorization where required

The integration is an engineering problem, not a policy problem. Each API already exists; the work is agent-runtime enrichment and policy expression.

Part VIII — Futures, Scenarios & Strategy

Chapter 23 — Scenarios for 2035 — Global

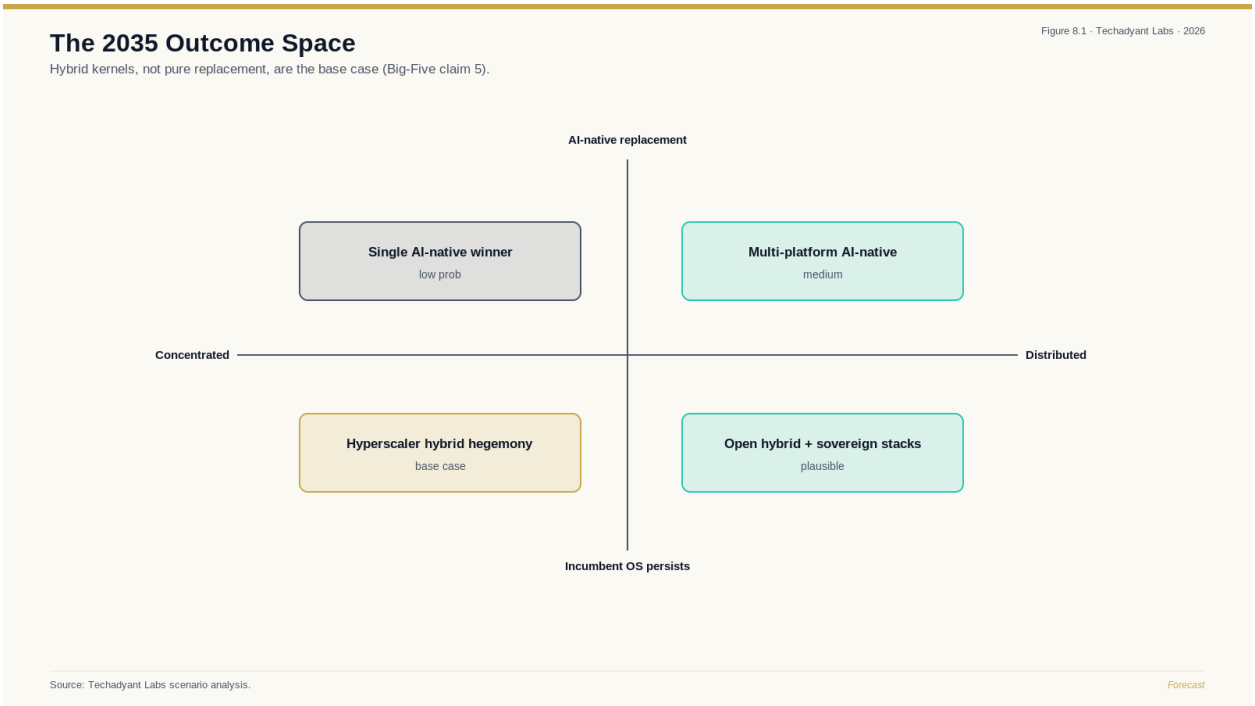

Figure 17 — The 2035 outcome space: hybrid kernels, not pure replacement, as the base case.

Thesis

By 2035, the agent-native transition will likely resolve into one of four plausible scenarios: a consolidated platform world, a polyglot ecosystem, a bifurcated century, or a stagnation pattern — each with distinct implications for how value, power, and access are distributed.

Scenario 1: The Consolidated Platform World

Mechanism. One or two agent runtime / OS platforms achieve network-effect scale. Developers build for the dominant platform. Users adopt the platform's agent ecosystem by default. The platform's security, memory, and capability standards become de facto global standards.

Likely players: a vertically integrated OS + silicon vendor (Apple), a hyperscaler with dominant AI runtime (Microsoft or Google), or a new entrant that ships an agent-native OS and captures developer mindshare before incumbents adapt.

Consequences:

- High developer productivity, low integration friction.
- Platform vendor captures outsized margin from runtime control.
- Secondary platforms survive in niches but cannot compete for mainstream workload.
- India must either adopt the dominant platform or fence off a domestic sovereign alternative.

Probability estimate: 35–45%

Scenario 2: The Polyglot Ecosystem

Mechanism. No single platform dominates. Instead, a multi-vendor standard — perhaps a Linux kernel with agent syscall extensions, or a WASM-based runtime widely adopted across OS vendors — creates interoperability.

Likely players: an open Linux-based AI-native OS, a browser-runtime standard (WASM + WebGPU), or a multi-vendor industry consortium.

Consequences:

- Innovation distributed across vendors; no single-point platform vulnerability.
- Integration complexity higher for enterprises; federation and mediation become necessary.
- Open-source agent tooling flourishes.
- India can participate in standards development without ceding control to one vendor.

Probability estimate: 25–35%

Scenario 3: The Bifurcated Century

Mechanism. The world splits into two (or more) incompatible agent ecosystems along geopolitical or regulatory lines — much like the current bifurcation between US-dominant and China-dominant internet services.

Likely players: US / EU coalition of democratic market platforms vs China sovereign AI ecosystem, with India, Large portions of Global South, and non-aligned countries choosing sides or operating in both.

Consequences:

- Dual-standard development costs; global agent interoperability becomes conditional.
- Strong incentive for domestic platform investment in every major country.
- Supply-chain and trade tensions extend to model weights, agent APIs, and runtime standards.
- India's non-aligned position becomes a strategic asset: it can trade platform access with both blocs.

Probability estimate: 20–30%

Scenario 4: Stagnation and Fragmentation

Mechanism. The transition stalls. Incumbent OS vendors absorb AI features without changing core abstractions. No new agent-native OS achieves mass adoption. Agent frameworks remain fragmented and incompatible.

Likely players: continued Windows/macOS/Android dominance with AI features bolted on.

Consequences:

- Agentic productivity gains arrive but are bottlenecked by OS mismatch.
- Security and observability gaps persist across the ecosystem.
- India's window for sovereign agent-native infrastructure narrows but does not close entirely.
- Long-term opportunity remains available but delayed to the 2040s.

Probability estimate: 10–20%

Variables That Shift Probabilities

Three variables will determine which scenario emerges:

44. Speed of incumbent adaptation. If Windows or Linux ships a compelling agent-native API within three years, Scenario 1 or 2 becomes more likely. If incumbents continue to bolt on, fragmentation wins.
45. Geopolitical alignment of AI policy. Tariff, export-control, and data-localization decisions will force ecosystem alignment more than technical merit.

46. Developer and enterprise adoption rates. Frameworks that make agent-native development easy will win adoption regardless of OS — but an easy runtime without an OS substrate to enforce policy will face security limitations at scale.

India's Scenario Posture

India is positioned to succeed in multiple scenarios:

- Scenario 1: India becomes a major market and talent provider for the dominant platform.
- Scenario 2: India contributes to open standards and develops domestic agent infrastructure that integrates cleanly.
- Scenario 3: India's non-aligned position lets it broker or bridge between ecosystems.
- Scenario 4: India has time — and domestic demand — to continue building.

What India cannot afford is *inaction*. The window to shape which scenario emerges — or at minimum to ensure India is a platform contributor rather than a platform consumer — closes by approximately 2030.

What 2035 Looks Like at Ground Level

Regardless of scenario, the 2035 knowledge worker will interact with computing through:

- Agents, not apps, as the primary interface layer.
- Intent specification — voice, text, or direct goal input — instead of menu navigation.
- Continuous workflow awareness across services, with agents resolving integration automatically.
- Outcome-based payment for software, not seat-based subscription.

The OS will be visible mainly to developers, IT administrators, and auditors. Ordinary users will experience it only when something fails.

Chapter 24 — India 2035: Strategic Outcomes

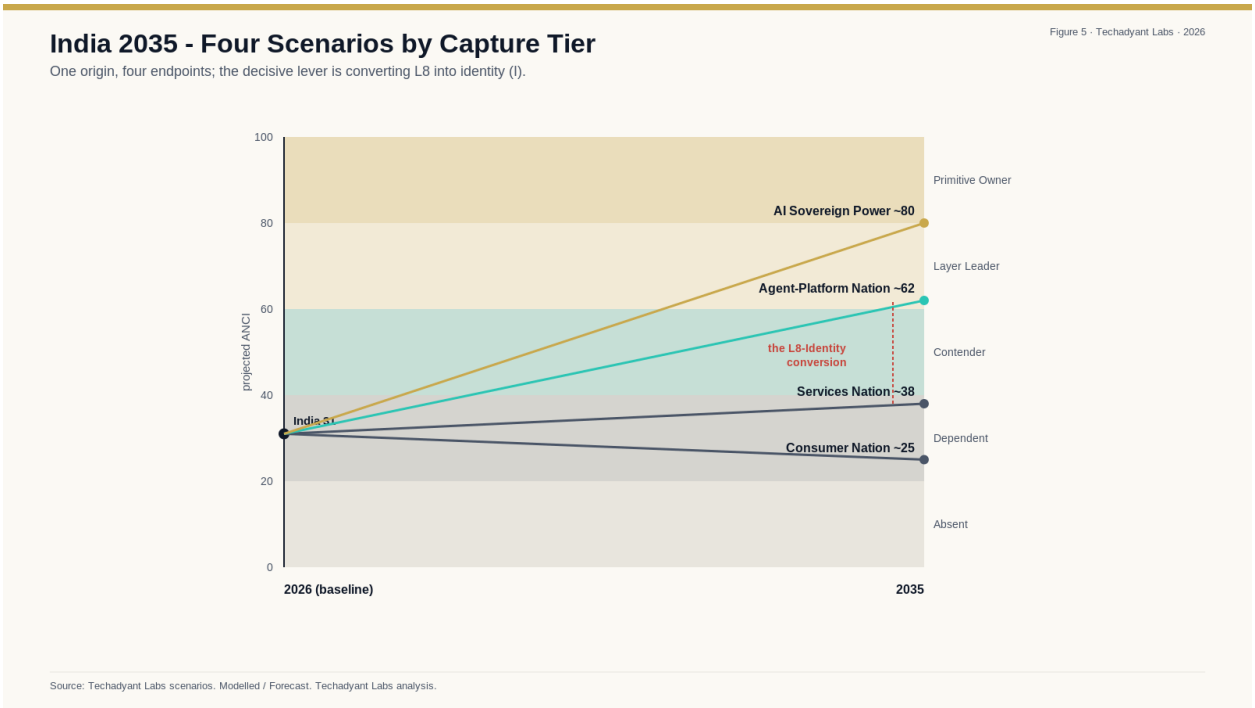

Figure 18 — India 2035: four scenarios by projected capture tier.

Chapter thesis: India's position in 2035 depends on policy choices made now. Four scenarios—Consumer Nation, Services Nation, Agent-Platform Nation, and AI Sovereign Power—represent distinct outcomes with different dependency profiles, value-capture levels, and geopolitical weight.

Scenario A: Consumer Nation

Description: India consumes foreign AI infrastructure at scale—cloud inference, foreign models, proprietary agent runtimes, and imported hardware. Domestic software services adapt to foreign platforms. Digital public infrastructure advances but remains application-layer rather than agent-native.

Probability: moderate

Impact: low sovereignty, medium economic growth, continued foreign dependency in compute and AI layers.

Probability-adjusted outcome value: lowest

Scenario B: Services Nation

Description: India builds strong AI implementation and services capability atop foreign infrastructure. Indian enterprises and government deploy sophisticated agent applications but rely on foreign compute, memory, models, and runtime platforms.

Probability: medium-high

Impact: strong services sector value capture, medium sovereignty, vulnerability to export controls and pricing changes.

Probability-adjusted outcome value: medium

Scenario C: Agent-Platform Nation

Description: India builds an agent-native platform stack anchored on India Stack: sovereign agent runtime, semantic memory, identity and consent layers, and regional compute fabric. The platform is used domestically and exported to other emerging markets seeking sovereign alternatives to hyperscaler lock-in.

Probability: medium

Impact: high value capture, strong sovereignty, regional geopolitical influence, clear export capability.

Probability-adjusted outcome value: high

Scenario D: AI Sovereign Power

Description: India achieves full-stack sovereignty: domestic semiconductor and packaging capacity, indigenous or deeply forked OS and runtime, sovereign models, and global standards influence. India becomes a reference architecture for agent-native national infrastructure.

Probability: low-medium

Impact: very high sovereignty, very high geopolitical weight, high capital intensity, long timeline.

Probability-adjusted outcome value: high but contingent on execution speed

---

Probability-Weighted Assessment

Probability-weighted value favors Scenario C (Agent-Platform Nation) as the optimal near-term target: high impact with feasible investment profile and alignment with India's existing assets.

The four scenarios are best distinguished on a single consistent metric — India's projected position on the report's **Agent-Native Capture Index (ANCI)** by 2035, scored on the four AIMS primitives (**A**ccelerated inference, **I**dentity, **M**emory, **S**cheduling), together with the **L8 sovereignty modifier** that the India Special Section establishes. Scores below are `[modelled]` projections, finalised in the framework-application pass.

Table 8 — Probability-Weighted Assessment.

Scenario	A — Accel. inference	I — Identity	M — Memory	S — Scheduling	Projected ANCI (0–100)	Tier	L8 modifier
A — Consumer nation	Low	Low	Low	Low	~25	Dependent	weak (DPI stays app-layer)
B — Services nation	Low	Med	Low	Low	~38	Dependent (upper)	moderate (services ride foreign rails)
C — Agent-platform nation	Med	High	Med-high	Med-high	~62	Layer Leader	strong (L8 exported as the platform)
D — AI sovereign power	High	High	High	High	~80	Primitive Owner	strong + standards influence

The decisive variable is not raw AIMS capability — on which India starts Dependent (~31 today) — but whether it converts its L8 advantage into identity (I) and interaction capture.

That single lever is what separates the ~38 of a Services Nation from the ~62 of an Agent-Platform Nation, and it is the most policy-actionable of the four primitives.

Key Decision Points

47. 2026–2027: IndiaAI compute and packaging investment decisions set the hardware floor.
48. 2027–2028: Runtime stack selection (fork, adopt, or build) determines software control.
49. 2028–2030: Enterprise and government deployment density determines whether India becomes a consumer or a platform nation.
50. 2028–2032: Standards participation in AIOS and agent governance determines long-term geopolitical influence.

Chapter 25 — Why This Thesis Could Be Wrong

Chapter thesis: Any strategic forecast must survive its own negation. This chapter presents five counterarguments—why applications may survive, OS changes may remain invisible, cloud platforms may absorb the transition, agent reliability may stall, and regulation may slow autonomy—and explains why we maintain the core conclusion.

Counterargument 1: Applications Survive; Agents Become a Feature

The most probable alternative is not replacement but absorption. Rather than killing applications, agents become an embedded feature: Copilot inside Office, Agentforce inside Salesforce, AI assistants inside every major platform. The application abstraction survives because enterprises have invested billions in integrations, workflows, and training that would be too expensive to rebuild.

Why we still reach our conclusion: Feature absorption is not the same as structural transformation. When agents become first-class citizens inside applications, the application ceases to be the unit of execution and becomes a tool registry and UI shell. That preserves compatibility while shifting the strategic layer upward. The value migration is the same; the incumbent simply rents space on the new layer.

Counterargument 2: OS Changes Remain Mostly Invisible

Operating system abstractions could evolve through user-space runtimes and APIs without kernel changes. Linux already supports containers, namespaces, and cgroups that create workload isolation without modifying the core. AI runtimes can sit above the OS and provide orchestration, memory, and tool dispatch.

Why we still reach our conclusion: Invisible evolution eventually hits a wall. Scheduler fairness, memory bandwidth, and security enforcement cannot be fully solved above the kernel. As inference intensity grows, the cost of crossing the user-kernel boundary becomes prohibitive. The industry is already seeing demand for kernel-level inference scheduling,

unified memory management, and machine-principal identities. What begins as user-space eventually demands kernel support.

Counterargument 3: Cloud Platforms Absorb the Transition

Endpoints could remain thin while the cloud becomes the real operating environment. Hyperscalers already provide inference, orchestration, memory, and identity services. The “OS” for enterprise agents is AWS, Azure, or GCP, not Windows or Linux. If this trajectory continues, endpoint OS design becomes irrelevant.

Why we still reach our conclusion: Cloud concentration is precisely the sovereignty risk this report flags. While model development and enterprise AI adoption are indeed cloud-centric, the strategic response—from India, the EU, China, and enterprise buyers—is sovereign or auditable compute. A post-OS world controlled by three hyperscalers is not stable geopolitically or economically. Sovereign stacks require kernel-level control; therefore OS design matters whether the workload is local, edge, or cloud-anchored.

Counterargument 4: Agent Reliability Stalls

Current agent systems hallucinate, lose context, misuse tools, and fail to recover from errors. These failures may prove intractable, limiting agents to narrow supervised roles rather than autonomous actors. Enterprises may reject autonomous agents after high-profile failures, reverting to deterministic applications.

Why we still reach our conclusion: Reliability problems are engineering challenges, not fundamental limits. Deterministic applications failed at scale too—security vulnerabilities, integration bugs, downtime—yet the model persisted because the economic incentives to solve them were large. Agent fault tolerance, guardrails, and observability are receiving real engineering investment. Even if full autonomy takes longer than this report projects, the trajectory toward agent-native orchestration is irreversible.

Counterargument 5: Regulation Slows Autonomy

EU AI Act, India's DPDP Act, US sectoral controls, and emerging global standards could impose compliance costs that make autonomous agents economically unattractive. Requiring human-in-the-loop approval for every non-trivial agent action would negate the productivity case.

Why we still reach our conclusion: Regulation shapes deployment patterns, not architectural direction. Agent-native OS abstractions—identity, memory, delegation, audit—are precisely what regulators need to make compliance tractable. Kernel-level audit trails and policy engines reduce compliance cost compared to application-level instrumentation. The regulatory response will accelerate OS-level agent primitives rather than prevent them.

Assessment

The counterarguments are real and underweighted in techno-optimist narratives. Applications will not vanish in five years. Cloud will dominate for cost-sensitive workloads. Agent reliability will improve unevenly. Regulation will impose friction. But none of these dynamics reverses the underlying workload shift: autonomous, continuous, memory-heavy, tool-orchestrating actors are the new center of gravity. Operating systems will adapt, as they always have, to the workloads they must carry.

Chapter 26 — Strategic Forecast and Recommendations

Thesis

The end of the application era is not a prediction that apps will disappear tomorrow; it is a recognition that the OS abstraction layer is overdue for evolution, and that India has a once-in-a-generation opportunity to shape the new layer rather than inherit the old one.

Summary of Findings

Across twenty chapters, this report has made the following core argument:

51. OS change is workload-driven. The application paradigm was shaped by 1970s–1990s PC workloads; agentic workloads demand new abstractions.
52. The shift is concurrent across layers. Not just one subsystem — the scheduler, or memory, or security — but all of them at once, under pressure from agents.
53. Hardware has already changed. GPUs, TPUs, NPUs, and unified memory architectures ended CPU dominance. The OS has not yet caught up.
54. Incumbents will adapt incrementally. Windows, macOS, and Linux will add AI features without changing core models, creating a compatibility-trap opportunity.
55. India is uniquely positioned to leapfrog. Digital public infrastructure, knowledge workforce, policy momentum, and demographic advantage create a rare alignment.
56. The window is 3–7 years. After ~2030, incumbent commitments and platform-effect lock-in will make radical OS design changes prohibitively expensive.

Recommendations for India

Treat Agent-Native Computing as National Infrastructure

India should classify agent-runtime infrastructure — compute clusters, model development platforms, standardized tool APIs — with the same strategic priority as roads, power, and telecommunication. The IndiaAI Mission is a first step; it must be followed by:

- National API standards for government services that are agent-addressable.
- Public compute subsidies for startups and researchers at parity with commercial cloud pricing.
- A national semantic-memory layer for government data that agents can query with consent.

Accelerate the IndiaAI Compute Cluster

The IndiaAI compute cluster must be operational at scale before 2028 to attract and retain AI talent. Priority investments:

- Sufficient H100/H200 or equivalent GPU allocation (not just a token cluster).
- Domestic cloud providers (AWS, Azure, GCP India regions) pegged to IndiaAI pricing for qualifying researchers.
- Open-weight model hosting that is free for academic and approved commercial use.

Mandate Agent-Addressable Public Services

All government digital services built after 2026 should expose agent-addressable APIs. This means:

- REST or gRPC endpoints documented in OpenAPI.
- ONDC-compatible protocol layers for discoverability.
- UPI-accessible payment hooks where financial transaction is part of the service.
- Aadhaar-linked, consent-mediated identity verification that agents can invoke programmatically.

This is not a technological revolution; it is a procurement and standards requirement that can be enacted by executive order.

Invest in Indic-Language Agent Models and Toolsets

Indic language quality will determine whether agent-native services reach 500 million Indians or 50 million. Investment should target:

- High-quality indic-language embeddings and retrieval models.
- Voice-interface optimization for India's diverse phonology.
- Multimodal agents that handle text, voice, and image queries in regional languages.
- Fine-tuning tooling that makes it easy for startups to adapt open-weight models to local domain knowledge (agriculture, healthcare, education, law).

Modernize Education for the Agent Era

Engineering curricula must evolve beyond traditional software engineering to include:

- Tool-use engineering and API semantic design.
- Agent lifecycle management and memory architecture.
- Capability-based security and delegation patterns.
- Multi-agent system design and verification.

Vocational education should include agent-augmented skills: operating an AI research agent, supervising an agent workflow, auditing agent outputs. These skills will be as essential as spreadsheet literacy was in the 1990s.

Create a National Agent Sandbox and Certification Framework

Before agents handle large-scale government and financial workflows, India needs:

- A national sandbox for testing agent behavior in regulated domains.
- A certification standard for agent reliability, security, and auditability.
- A liability framework that assigns accountability when agents act on behalf of humans or institutions.

This framework should be developed in partnership with MeitY, RBI, CERT-In, NASSCOM, and academic institutions.

Position India as the Voice of the Global South in AI Governance

India has the credibility to advocate for an AI governance model that balances innovation with inclusion — neither the light-touch US approach nor the precautionary EU approach, but a *participatory* approach that gives developing countries voice in model governance, data access, and infrastructure investment.

India should:

- Lead a Global South AI coalition within the UN and G20 processes.
- Push for open-weight model access and technical standards that do not require Western cloud dependency.
- Develop an AI Partnership for Global South Infrastructure that pools compute, talent, and data for member nations.

Recommendations for the Technology Industry

Prepare for Consumption-Based Pricing Legacy

Software vendors must transition licensing from per-seat to per-task-outcome. The vendors that lead this transition will define the commercial terms for the next era. Those that resist will face agent-driven price discovery that erodes their margins.

Expose Granular APIs or Risk Disintermediation

Every software company — SaaS, ISV, or services — should treat API exposure as existential. If your product cannot be invoked by an agent through a well-documented, granular, secure API, your product will not appear in agent workflows, and agents will find or build alternatives.

Invest in Agent Observability and Trust

The companies that solve agent observability — reasoning traces, audit trails, memory inspection, budget enforcement — will build the trust layer that enterprises require. This is a new product category worth billions.

Differentiate on Data and Context, Not Just Features

When agents can compose capabilities dynamically, feature checklists become less important. What matters is:

- Proprietary, high-quality data that agents can access.
- Deep integration into user workflows that makes the agent smarter over time.
- Trust and compliance posture that enterprise buyers require.

Recommendations for Policymakers Globally

Do Not Regulate Agents; Regulate Agent Outcomes

Per-agent licensing or agent-runtime pre-approval would stifle innovation. Policy should focus on *outcomes*: financial harm, privacy violations, discrimination, safety risk. Let the technology iterate; hold deployers accountable for results.

Invest in Compute Infrastructure as Public Good

AI compute is becoming as foundational as roads and electricity. Government investment in compute clusters — shared, open-access, with transparent pricing — is essential to prevent a compute cartel.

Enable Cross-Border AI Cooperation

Bilateral and multilateral AI cooperation frameworks should include:

- Standardized safety-evaluation protocols.
- Shared incident-response playbooks for agent-related security events.
- Mutual recognition of certification frameworks.

Ten Forecasts for 2030, 2035, and 2040

Table 9 — Ten Forecasts for 2030, 2035, and 2040.

Forecast	Confidence	Rationale
By 2030, agent-native productivity tools outperform traditional app-based equivalents in at least three knowledge-work domains	High	trajectory already visible
By 2032, at least one major OS ships an official agent-lifecycle API	High	pressure is already building
By 2033, the top-five AI companies by market cap include at least one agent-runtime or agent-platform company	Medium-High	market structure shift
By 2034, India has a domestically operated inference cluster at competitive scale	Medium	depends on IndiaAI execution
By 2035, the "application" is no longer the primary interface for majority of digital services in India	Medium	contingent on policy and UX quality

Forecast	Confidence	Rationale
By 2035, at least one major global enterprise has replaced most of its internal software with agent-mediated integrations	Low-Medium	requires major orchestration solution
By 2035, AI-native chip architectures dominate non-general-purpose compute segments	High	silicon trajectory is clear
By 2035, there is a global regulatory framework for agent-level audit and liability	Medium	political will is uncertain
By 2035, India has at least one globally competitive agent platform or OS company	Low-Medium	ambitious but not impossible
By 2040, the concept of "installing software" feels archaic to most users under 25	Low	cultural persistence is underestimated

Closing: A Choice, Not a Destiny

The end of the application era is not predetermined. OS vendors could reverse course and commit to radical redesign. Regulators could block the transition. A security incident could set agent-native adoption back a decade.

But the underlying force — workloads outgrowing abstractions — is real. It has happened before with time-sharing, GUIs, virtualization, and mobile. It is happening again with agentic AI.

For India, the question is not *whether* this transition arrives, but *who shapes it* and *who benefits from it*. The window is narrow, the infrastructure prerequisites are within reach, and the alternative — importing a foreign agent platform — is avoidable with coordinated action.

The end of the application era is an invitation to build something better. The next chapter belongs to the builders.

Appendix A — Global AIOS Research Landscape

Research Bible

- Pages: 4
- Scope: map academic, government, open-source, and commercial work related to AI-native operating systems and agent runtimes.

Contents

- Academic and institutional research directories relevant to AI-native OS, agent runtimes, context OS, and horizon computing.
- Map by geography, institution type, vendor, and openness/access.

Geographic Summary

North America remains the densest research region for systems-level AI work, with major contributions from:

- Stanford HAI and SAIL
- UC Berkeley, especially RISE and security/systems groups
- Carnegie Mellon OS and software-engineering programs
- MIT CSAIL and multicore/operating-systems research
- University of Washington, University of Michigan, and University of Texas systems groups

Europe shows strong government-lab and open-source coordination, including:

- ETH Zurich and EPFL systems projects
- TU Munich and TU Dresden hardware-software codesign
- VU Amsterdam, KTH, and INRIA distributed-systems research
- EU-funded projects and national AI institutes
- ARM and Imagination Technologies UK ecosystem

Asia-Pacific contributors include:

- Tsinghua, Peking, and Chinese Academy of Sciences systems work

- Seoul National University, KAIST, and ETRI semiconductor-systems work
- University of Tokyo, RIKEN, Osaka
- India: IISc, IIT systems labs, C-DAC

Institutional Summary

Academic institutions generate foundational models, reference systems, benchmark suites, and training for the next generation of systems researchers. Government labs contribute reliability- and security-critical prototypes and standards input. Open-source ecosystems translate concepts into runnable systems, influencing product design and education. Commercial projects shape operational reality through deployment, data, and capital.

Open-Source Drivers

Open-source projects provide the reference-level building blocks for AI-adjacent systems:

- Agent runtimes and orchestration layers
- Memory and context management
- Kernel subsystems and unikernel variants
- Secure enclave and capability-model prototypes
- Developer toolchains and evaluation frameworks

Commercial and Vendor Projects

Vendor programs relevant to AI-native OS research include:

- hyperscaler custom-kernel and accelerator-software work
- semiconductor vendor software stacks
- endpoint and embedded AI runtime programs
- security-vendor trust-boundary research
- virtualization and container-runtime trajectory

Research Landscape Matrix

This matrix summarizes the primary research force types and their likely near-term outputs.

Table 10 — Research Landscape Matrix.

Research force type	Example outputs	Likely near-term contributions

Research force type	Example outputs	Likely near-term contributions
Systems academia	AIOS prototypes, scheduler and memory redesign papers	Concepts, benchmarks, people
Government labs	Secure distributed runtime prototypes	Standards influence, reference artifacts
Open-source ecosystems	Reference runtimes and kernel modules	Adoption infrastructure and community
Commercial platform teams	Production AI stacks, accelerator runtime layers	Operational design patterns and market signals
Semiconductor and hardware research	Memory-semantics and packaging impact	Architecture constraints and capability ceilings

Key Trends

57. Research is moving from application-level AI to runtime-level integration.
58. Memory semantics, context capacity, and scheduling policy are emerging as primary research targets.
59. Security models are a cross-cutting concern and increasingly a collaboration topic between systems and security researchers.
60. India's public digital infrastructure creates a unique deployment surface for agent-native OS experimentation.

Cross-References to Report Chapters

- Chapter 1: historical precedent for OS change and institutional research patterns.
- Chapter 2: abstraction redesign targets surface here as research topics.
- Chapter 4 and 5: workload definitions map to current research investment.
- Chapter 11 and 12: candidate AIOS architectures align with these research programs.
- Chapter 17 and 18: India strategic chapters depend on identifying the right collaboration and transfer strategy.

Research Gaps

No current program fully integrates:

- OS-level agent memory management
- capability-based trust enforcement
- accelerator-software co-design at the scheduler boundary
- policy-governable agent telemetry
- Indian deployment constraints and requirements

The appendix concludes that intentional India participation in these research areas is a feasible and strategically valuable intervention.

Appendix B — Major AI Infrastructure Companies

Research Bible

- Pages: 4
- Scope: major companies whose products, roadmaps, or capital decisions influence the AI-native infrastructure landscape in 2026-2035.

Contents

- Company profiles by category: accelerator, cloud, OS/platform, memory/storage, semiconductor manufacturing, packaging, and security/governance.
- Vendor map with dominance drivers and inflection points.
- Concentration risk and dependency notes relevant to India strategy.

Accelerator and Compute-Layer Companies

The primary AI-accelerator market continues to be dominated by a small set of platform vendors:

- NVIDIA: dominant training and inference hardware, software stack, and ecosystem lock-in.
- AMD: growing presence in heterogeneous compute through CPU-GPU integration.
- Intel: incremental GPU and AI accelerator attempts; historically strong incumbent in enterprise silicon.
- Google TPU: custom ASIC experience but largely internal deployment.
- Others include Graphcore, Cerebras, SambaNova, Groq, and emerging Chinese accelerator vendors.

Concentration risk is high, especially for inference workloads where the architecture choices are still forming.

Cloud and Platform Companies

Cloud providers define operational reality for enterprise AI adoption:

- AWS, Azure, Google Cloud, Oracle Cloud: primary hyperscalers with custom silicon, managed services, enterprise relationships, and regulatory exposure.
- They also shape OS-adjacent runtime expectations through managed execution environments, serverless models, and container orchestration.

Operating System and Endpoint Vendors

OS vendors remain at the center of the abstraction question:

- Microsoft: Windows domination, enterprise footprint, Copilot integration, and OS-native AI extensions.
- Apple: macOS and iOS vertical integration, memory-optimized hardware-software design.
- Linux ecosystem: distribution diversity, cloud dominance, enterprise and embedded ubiquity.
- Mobile OS providers: Android and iOS govern the largest agent-facing endpoints.
- BSD and niche unikernel communities: source of architectural experimentation with smaller deployment constraints.

Memory and Storage Companies

Memory vendors set the boundary for what memory-heavy workloads can achieve:

- DRAM vendors: Samsung, SK Hynix, Micron
- High-bandwidth memory and advanced packaging participants
- SSD and storage-class memory vendors
- FPGA and programmable-logic providers

Memory bandwidth is often the limiting factor for agentic inference and context-heavy reasoning, especially during reinforcement and retraining cycles.

Semiconductor Manufacturing and Packaging

Manufacturing leadership is now the central node of geopolitical AI competition:

- TSMC leads advanced-node manufacturing.
- Samsung Foundry and Intel Foundry ambitions create partial diversification.
- Packaging, especially chiplets and 2.5D/3D integration, is becoming a first-order performance factor.
- India has packaging and assembly exposure through state and private programs.

Security and Governance Vendors

Identity, trust-boundary, and audit-vendor markets are adapting to machine-delegation workloads:

- SIEM, EDR, and XDR vendors
- Zero-trust infrastructure vendors
- Policy engine and attestation vendors
- Compliance and audit-tool vendors

Dependency Architecture

A simplified dependency map for AI infrastructure:

Table 11 — Dependency Architecture.

Layer	Critical vendors / contributors
Accelerator silicon	NVIDIA, AMD, Intel, Google TPU
Manufacturing	TSMC, Samsung, Intel Foundry
Packaging and interconnect	ASE, Amkor, TSMC InFO / CoWoS
Memory	Samsung, SK Hynix, Micron
Cloud runtime	AWS, Azure, GCP, Oracle
OS / endpoint	Microsoft, Apple, Linux ecosystem, mobile OS vendors
Security and governance	enterprise security vendors plus new policy-engine startups

India-Relevant Exposure

India's exposure to AI infrastructure concentration includes:

- Hyperscaler dependency for cloud-based AI training and inference
- Accelerator access constraints during global demand surges
- Memory and packaging dependence on external manufacturing
- OS and runtime dependence on Western platforms
- Talent pool and startup ecosystem gaps in kernel and systems research

These exposures are addressable through capability building, procurement strategy, and standards participation.

Near-Term Inflection Points

61. Accelerator architecture fragmentation after GPU dominance.
62. Memory-semantics and context-management software becoming first-class OS concerns.
63. Sovereign-AI policies creating new procurement and localization incentives.
64. Packaging and chiplet adoption raising new OS visibility and system-management questions.

Appendix C — Emerging AI Hardware Platforms

Research Bible

- Pages: 4
- Scope: emerging AI hardware relevant to agent-era computing: accelerator families, packaging innovations, NPU/SoC trends, and India-adjacent manufacturing exposure.

Contents

- Accelerator families and their workload profiles.
- Packaging and interconnect trends.
- NPU and SoC devices for endpoint and edge.
- Beyond-silicon approaches and their system implications.

Accelerator Families

Accelerator families now span data-center, edge, and endpoint scales:

Table 12 — Accelerator Families.

Family	Typical proprietors / vendors	Workload fit	Near-term relevance
GPU-class	NVIDIA, AMD	dense matrix, training and inference	dominant through ~2028-2030
ASIC inference	custom TPU, edge ASIC vendors	low-latency inference, cost-optimized inference	growing share
NPU	SoC vendors and endpoint OEMs	on-device agent inference	high, as endpoints gain agent capabilities
Reconfigurable	FPGA and CXL-attached accelerators	dynamic workloads and policy-bound compute	moderate, rising
Neuromorphic / alternative	research and niche vendors	unconventional memory-compute interaction	longer term

Accelerator proliferation increases OS complexity because each family has different memory models, scheduling constraints, and security surfaces.

Packaging and Interconnect Trends

Advanced packaging is now performance-critical:

- Chiplet architectures improving yield and enabling heterogeneous cores.
- 2.5D/3D packaging reducing interconnect latency between logic, memory, and accelerators.
- High-bandwidth memory becoming tightly coupled to compute rather than far DRAM.
- CXL and UCIe as emerging interconnects between hosts and accelerators.

For OS design, packaging changes matter because:

- memory visibility changes (disaggregated memory pools must be mapped and protected)
- device enumeration and scheduling complexity increase
- fault domains become smaller and more complex to reason about

NPU and SoC Devices

Endpoint and edge devices are embedding NPUs with agent-aware software stacks:

- Windows, macOS, Android, and Linux each offer agent-capable runtime extensions.
- NPU software stacks are still fragmented across vendors.
- Memory budgets on endpoints remain constrained, creating pressure on context compression and semantic retrieval.
- India-relevant OEM exposure is rising in smartphones, laptops, and edge gateway devices.

Beyond-Silicon Approaches

Research-stage approaches include:

- photonics and optical interconnects for high-speed chip and board communication
- in-memory compute and memristor-class devices
- near-memory and processing-in-memory architectures
- brain-inspired communication motifs

These are potentially transformative but typically require OS-level memory-management, visibility, and security redesign.

India Exposure

India’s direct manufacturing exposure is currently focused on:

- semiconductor assembly, testing, and packaging
- SoC and board-level product assembly
- public-sector research capacity

Absent domestically advanced-node manufacturing through 2030, India’s strategy should emphasize packaging leadership, design capability, and software-architecture standards that maximize the utility of imported semiconductor assets.

Implications for the Report

Hardware trajectories directly shape the architectural conclusions in chapters 6, 8, 9, 11, 12, and 13. This appendix provides reference detail for chapters 11 and 12, and supports India strategy in chapters 17 and 18.

References and Sources

- Semiconductor manufacturer disclosures
- packaging roadmaps from assembly partners
- SoC vendor documentation
- academic research publications on near-memory compute

Appendix D — Glossary

Purpose

This glossary defines key terms used throughout the report. Entries are intended to support consistent usage across chapters and for external readers unfamiliar with systems, AI, and India-policy terminology.

A

Agent-native OS: An operating system designed to make autonomous, tool-orchestrated agents the primary unit of work, rather than installed applications launched by a human.

Agent sandbox: A restricted execution environment for agent processes and tool calls, with explicit resource, memory, network, and permission boundaries.

API-first abstraction: A design pattern where software exposes behavior primarily through structured programmatic interfaces rather than human-directed interfaces.

Application silo: A configuration in which an application retains its own state, storage, and configuration, with limited structured interoperability with other applications without explicit integration work.

C

Capability token: A restricted, revocable, auditable grant of authority for an agent to perform a defined action within defined limits, rather than a broad credential.

Chiplet: A modular semiconductor die designed to be assembled with other chiplets into a larger package, enabling heterogeneous computation and improved manufacturing yield.

Context window: The current input token boundary of an AI model, including conversation history, system instructions, tool outputs, and retrieved context.

CPU-centric scheduling: Operating-system task scheduling that assumes central processing units as the primary compute resource and interrupts the common coordination mechanism.

D

Delegation authority: The right granted by a principal to an agent to act on its behalf within defined policy limits, including scope, duration, and risk constraints.

Disaggregated memory: A memory architecture in which memory pools are separately addressable from compute resources, enabling larger shared context but requiring new OS memory-management and protection abstractions.

DRAM: Dynamic random-access memory; the dominant volatile main memory technology in contemporary computing systems.

E

Edge device: A computing endpoint near data sources or human actors, typically with constrained power, memory, and compute compared to data-center hardware.

H

Heterogeneous compute: Systems combining multiple processor types, typically CPU, GPU, NPU, and accelerator-class silicon, requiring OS scheduling, memory-management, and security redesign.

Hybrid kernel: An operating-system kernel combining microkernel-like separation of concerns with monolithic-style performance optimizations for core subsystems.

I

Inference workload: The execution pattern of applying a trained model to inputs to produce outputs, typically latency-sensitive and often continuous in agent-facing systems.

M

Memory bandwidth: The rate at which data can be read from or written to memory, a key performance limit for memory-heavy workloads.

Memory-compute bottleneck: A performance condition in which time spent moving data dominates time spent computing.

Memory primacy: The principle that memory and context should be treated as first-class OS resources on par with processes, files, and devices.

N

NPU: Neural processing unit; a specialized accelerator optimized for neural-network inference workloads.

O

OS-level governance hook: A kernel or system-service interface through which policy, audit, compliance, and attestation requirements can be enforced for autonomous actors.

P

Pinned memory: Memory reserved for a specific agent, model, or workload, optionally with hardware protection, used to guarantee context integrity and reduce jitter.

Prompt injection: An adversarial input targeting an AI system that influences its behavior in unintended ways by overriding or manipulating instructions or retrieved context.

S

Semantic memory: Long-lived, searchable, agent-relevant knowledge stored in a form optimized for meaning and retrieval rather than exact matching.

Service account: An identity assigned to a non-human workload, typically with restricted permissions and no associated human operator session.

Sovereign computing: Compute infrastructure, software stack, and policy controlled within national jurisdiction for defense, economic, or public-interest reasons.

T

Trust boundary: A logical or physical perimeter where trust assumptions change, typically separating more-trusted and less-trusted components, actors, or data.

U

Unified memory: A memory architecture visible to multiple compute classes through the same address space, reducing explicit data copying but raising complexity in protection and scheduling.

V

V1 evidence: Primary sources such as official filings, vendor disclosures, and official releases.

V2 evidence: Verified secondary sources, including named-source reporting and academic papers.

V3 evidence: Corroborated analysis built from multiple V2 sources, judged credible but not directly from primary origin.

Appendix E — Methodology

Purpose

This appendix documents the research and evidence standards used for the report. It is intended for reviewers, auditors, and future researchers who want to verify or reuse the evidence workflow.

Scope of Research

The report covers operating-system abstractions, accelerators, memory models, security transitions, industry economics, geopolitics, and India-specific opportunity through 2035. Core claims are drawn from primary sources where available, with corroboration from verified secondary sources and reasoned analysis.

Evidence Grammar

The report uses a three-tier evidence grammar:

Table 13 — Evidence Grammar.

Grade	Description
V1	Primary source: official filings, vendor disclosures, official releases, legislation, and standards documents
V2	Verified secondary source: major reporting with named sources, academic papers, knowledgeable vendor or analyst commentary
V3	Corroborated analysis: synthesized from multiple V2 sources and judged credible but not directly from a primary origin

Each evidence row includes:

- Source
- Claim

- Certainty grade
- Date or date range
- Link or document reference where stable and available

Source Selection Rules

- Prefer official disclosure and primary source over commentary.
- Prefer reproducibility and traceability over influence.
- Treat vendor forecasts and analyst estimates as assumptions with explicit source attribution.
- Treat government policy as directional unless codified in official text.

OSINT Validation

Public-source claims are verified before inclusion using:

- direct linking to stable or archive-backed URLs
- matching dates with source metadata where possible
- confirmation across independent sources for contested claims

This discipline is especially important for:

- semiconductor roadmap claims
- vendor release dates and availability statements
- regulatory and policy details across multiple jurisdictions

Geographic and Sectoral Bias Controls

Systems research is concentrated in North America, Europe, Japan, and increasingly China. India-specific chapters use additional bias controls:

- India claims are anchored to official government publication, parliamentary and NITI Aayog documentation, and major vendor or startup public statements.
- India opportunity claims are framed as conditions for leapfrog, not outcomes asserted as certain, and include clear dependency statements.

Limitations

- Some vendor internal architecture and firmware decisions are not publicly disclosed.
- National AI and semiconductor roadmaps may change after publication.
- Geopolitical and regulatory forecasts contain inherent temptation toward speculation; this report explicitly distinguishes probable structural drivers from contingent outcomes.

Audit and Reuse Notes

Researchers wishing to reuse or verify the evidence base should:

65. request the chapter-specific bibles and source lists
66. compare evidence rows against source materials and official documents
67. treat V3 claims as argumentative evidence rather than settled fact

This methodology appendix is designed to make that review feasible without additional authoring context.

Appendix F — Future Research Agenda

Purpose

Identify the highest-value unresolved research questions for the agent-native computing transition. These questions are intended for research funders, institutional labs, platform vendors, and India-focused technology strategy teams.

Research Themes

OS-Level Memory and Context Management

Research should produce:

- OS-native context primitives
- pinned-memory and semantic-memory interfaces
- memory-bandwidth-aware scheduling abstractions

Trust Delegation and Policy Enforcement

Research should produce:

- capability-token standards
- OS-integrated attestation and audit interfaces
- delegations across heterogeneous compute boundaries

Accelerator and Scheduler Co-Design

Research should produce:

- unified heterogeneity-visible scheduler interfaces
- accelerator thermal, memory, and failure-domain abstractions
- memory-pool visibility and protection boundaries

AI-Native Security Primitives

Research should produce:

- prompt-injection detection at the runtime layer
- agent telemetry standards for incident analysis
- auditable agent decision histories

India-Specific Deployment Research

Research should produce:

- agent-native systems operating under 印度 data-localization and public-digital-infrastructure constraints
- NPU/edge capability for low-bandwidth environments
- sovereign trust-boundary designs aligned with local regulatory norms

---

Priority Questions

68. What OS semantics should replace filesystem-centric state sharing for dynamic agent tool use?
69. How should memory be protected when context is no longer bounded by a single process or user session?
70. What standards and interfaces will allow trust delegation to survive across accelerators, kernels, and distributed services?
71. How can sovereign-AI strategies avoid replicating the same supplier concentration they are meant to eliminate?
72. What Indian research, manufacturing, and policy interventions create the largest future optional value?

Recommended Next Investigations

Table 14 — Recommended Next Investigations.

Investigation	Output	Strategic value
Memory semantics for agent workloads	reference framework	enables OS redesign
Scheduler policy for heterogeneous acceleration	prototype	performance leadership
Trust token and audit design	protocol draft	security baseline
India packaging and design ecosystem case study	public report	investment and policy decision support
AI-native developer workflow	user study and system prototype	platform design input

Investigation	Output	Strategic value
research		

Closing Note

The agenda underscores that the report's strategic conclusions are not final engineering decisions. They represent the current visible inflection horizon, and further research can shift probability distributions substantially.

Appendix G — The Agent-Native Capture Index: Scoring Tables and Sensitivity

This appendix documents how the report's actors were scored on the Agent-Native Capture Index (ANCI), so the reader can interrogate and reproduce the judgements rather than take them on trust.

G.1 Method

Each actor is scored on the four AIMS primitives — **A**ccelerated inference (PAS layers L1–L2), **I**dentity and delegation (L5), **M**emory and context (L3), and **S**cheduling and orchestration (L4/L6). Each primitive is scored 0–25 against anchored bands:

Table 15 — G.1 Method.

Band	Points	Meaning
Controlling	20–25	Sets the standard / owns the dominant implementation others depend on
Capable	13–19	Credible first-party implementation; not yet standard-setting
Dependent	6–12	Uses the primitive but relies on others' implementations
Absent	0–5	No meaningful position

The four scores sum to the ANCI (0–100), which places the actor in a tier: Primitive Owner (80–100), Layer Leader (60–79), Contender (40–59), Dependent (20–39), Absent (0–19). Scores are `[modelled]` strategic assessments grounded in the verified evidence in the source register; the horizon is the 2026 baseline. Nations additionally carry a qualitative **L8 sovereignty modifier** capturing how public digital infrastructure changes the strategic read independent of raw AIMS control.

G.2 Full scorecard (2026 baseline)

Table 16 — G.2 Full scorecard (2026 baseline).

Actor	A	I	M	S	ANCI	Tier
Microsoft	14	20	18	19	71	Layer Leader
Google / Alphabet	21	14	19	17	71	Layer Leader
Amazon / AWS	20	16	13	16	65	Layer Leader
OpenAI	10	12	16	18	56	Contender
Apple	14	15	13	12	54	Contender
Anthropic	9	13	14	18	54	Contender
Open-source / Linux ecosystem	8	15	13	16	52	Contender
NVIDIA	24	5	9	13	51	Contender (A-concentrated)
Meta	16	8	14	10	48	Contender
India (nation)	6	10	7	8	31	Dependent — L8 modifier strong (+)
Traditional SaaS (category)	4	6	5	6	21	Dependent (downward trajectory)

G.3 How to read the scores

The index rewards breadth across the stack, not depth in one primitive. Two facts follow that are easy to misread:

First, **NVIDIA's 51 is not "half as strong as a Layer Leader."** It is a near-maximal A (24/25) with thin I/M — a silicon owner exposed above its own substrate. The single number compresses a highly concentrated profile, which is why the per-primitive columns, not the total, carry the analytical weight for specialists.

Second, **the Contender band (48–56) is genuinely crowded.** OpenAI, Apple, Anthropic, the open-source ecosystem and NVIDIA all sit within eight points of one another on very different primitive mixes. Their *ordering* inside the band is not load-bearing and the report

does not rely on it; they should be read as one contested tier, differentiated by *which* primitive each is strong in.

G.4 Sensitivity analysis

Tier boundaries sit at 40, 60 and 80. Robustness checks:

A perturbation of **±2 on any single primitive** moves no actor across a tier boundary; the maximum single-actor swing is eight points, and only if all four primitives move in the same direction simultaneously. The three Layer Leaders (65–71) sit more than five points clear of the 60 boundary and are stable to plausible re-scoring. The Contender cluster is tight by ±2 but, as noted, the report treats it as a band rather than a ranking, so intra-band movement does not change any conclusion.

India's 31 is mid-Dependent. A uniform **+5 across all four primitives** — an optimistic five-year AIMS improvement — would lift it only to 51, still short of Layer Leader. This is the quantitative form of the report's central India claim: the leapfrog cannot be achieved by climbing the AIMS primitives alone within the horizon, so it must run through the L8 sovereignty modifier (consent infrastructure, DPI export) rather than through out-building hyperscalers.

G.5 What would change the scores

The scorecard is a snapshot. The events most likely to move it within the horizon: a credible non-NVIDIA accelerator reaching software parity (compresses NVIDIA's A advantage and lifts AMD/custom silicon); convergence on a single cross-vendor agent-identity standard (raises whoever anchors it on I, currently contested between Entra Agent ID and AgentCore Identity); and a memory/context standard escaping any single vendor (redistributes M). The report's forecast chapter is, in effect, a set of hypotheses about which of these moves first.

List of Figures

List of Tables