Modern OS Memory Management

Advanced Operating Systems — Week 6

Hunter College, CUNY

Faye
czhang2@gradcenter.cuny.edu

March 2026

References:
Silberschatz, Galvin, Gagne — *Operating System Concepts*, 10th Ed., Chapters 9–10
https://os-book.com/OS10/slide-dir/index.html
https://www.cs.hunter.cuny.edu/~sweiss

Contents

1 The Original Memory Crisis

The Memory Wall

The "memory wall" describes the growing disparity between CPU speed and memory access latency. Processors can execute instructions in sub-nanosecond cycles, yet DRAM access still takes ∼50–100 ns. As workloads scale — especially AI training and inference — this gap becomes the dominant performance bottleneck. The classical cache → DRAM → disk hierarchy can no longer keep up.

1.1 Physical Limits

The traditional hierarchy is straining under modern workloads:

- **Cache**: fast but tiny (tens of MB). Transistor density limits further growth.
- **DRAM**: capacity is growing slowly; DDR5 bandwidth improvements are incremental.
- **Disk/SSD**: orders of magnitude slower; even NVMe SSDs add microseconds of latency.
- **AI workloads**: large-language models require hundreds of GB of parameter memory, far exceeding single-node DRAM capacity.

1.2 Software Flaws in the Classical Model

- **4 KB page granularity**: designed decades ago. Modern workloads with huge datasets suffer from massive page-table overhead and TLB pressure.
- **1:1 thread-to-kernel-thread mapping**: each OS thread costs a full stack (∼8 MB on Linux) and a kernel scheduling entity. Applications needing millions of concurrent tasks (web servers, microservices) are throttled.

2 Breaking the Memory Wall: HBM & Compute-in-Memory

2.1 High Bandwidth Memory (HBM)

HBM stacks multiple DRAM dies vertically using **through-silicon vias (TSVs)** and connects them to the processor via a **silicon interposer** (a wide, short bus on the package).

- **Bandwidth**: HBM3 delivers up to **819 GB/s** per stack — roughly 5× DDR5.
- Used in NVIDIA H100/A100, AMD MI300X, and other AI accelerators.

OS-level challenges:

- **Limited capacity**: HBM offers only 16–32 GB per package (vs. TB-scale DDR). The OS must manage it as a *fast tier*, migrating hot pages in and cold pages out.
- **Thermal walls**: 3D stacking creates heat dissipation problems. The OS may need to throttle access patterns or redistribute work.
- **Silent data corruption**: at extreme densities, single-bit errors are more likely. HBM uses **SEC-DED** (Single Error Correction, Double Error Detection) ECC, but the OS must handle uncorrectable errors gracefully.

2.2 Compute-in-Memory (CIM)

CIM performs **multiply-accumulate (MAC)** operations *inside* the memory array itself, eliminating the need to move data to the CPU for simple operations.

- Especially useful for **matrix–vector multiplication** in neural network inference.
- The OS role shifts: instead of just managing addresses, it must **map computation to memory**, deciding which operations execute in-place vs. on the CPU/GPU.
- Requires new abstractions in the memory subsystem and potentially new system calls.

3 Tiered Memory via the CXL Protocol

Compute Express Link (CXL) is an open interconnect standard (built on PCIe 5/6 physical layer) that enables cache-coherent access to external memory pools.

Tier	Latency	Advantage	OS Strategy
DRAM (local)	∼80 ns	Fastest	Default allocation
CXL-attached DRAM	∼170–250 ns	Expandable capacity	Hot/cold page migration
CXL-attached NVM	∼300–500 ns	Persistent, dense	Cold data, checkpointing
NVMe SSD	∼10–100 μs	Cheap, high capacity	Swap / memory-mapped files

3.1 Hardware 2LM vs. OS-Managed zNUMA

- **Hardware 2LM** (Two-Level Memory): the memory controller transparently caches CXL memory in local DRAM. The OS sees a single flat address space. Simple but inflexible.
- **OS-managed zNUMA**: the OS treats CXL memory as a separate NUMA node with its own zone. The kernel makes explicit migration decisions based on access frequency. More complex but allows fine-grained policy.

3.2 Meta's Transparent Page Placement (TPP)

Meta's production system uses a proactive page-management policy:

- **Proactive demotion**: cold pages are moved from local DRAM to CXL memory *before* memory pressure forces it.
- Uses a **64-bit bitmap** per page to track access history over time windows.
- Result: **17% throughput improvement** compared to reactive migration in production web workloads.

4 Memory Disaggregation

The Waste Problem

Studies show that **∼50% of VMs waste their allocated memory** — they are provisioned for peak demand but run far below it most of the time (memory stranding).

Memory disaggregation decouples memory from compute, allowing memory to be pooled and shared across machines.

4.1 Azure Pond

Microsoft's Azure Pond uses **CXL-based shared memory pools**:

- A rack-level memory pool is shared among multiple compute blades.
- VMs can dynamically expand or shrink their memory allocation without migration.
- Reduces overall memory provisioning by 20–30%.

4.2 AWS Nitro

Amazon's **Nitro** system offloads the hypervisor to a custom **ASIC**:

- Network, storage, and security processing are handled by dedicated Nitro cards (hardware), not by software running on the host CPU.
- Nearly **all host CPU and memory** are available to customer VMs.

- **Security bonus**: the hardware-based isolation reduces the attack surface. The Nitro hypervisor has a minimal software footprint, reducing the chance of memory-related vulnerabilities.

5 Non-Volatile Memory (NVM)

Non-volatile memory technologies (e.g., Intel Optane / 3D XPoint, Samsung CXL-NVM) provide:

- **Byte-addressable** access (like DRAM) — no need to read/write in block-sized chunks.
- **Persistence** — data survives power loss (like disk).
- Latency: ~300 ns (slower than DRAM, vastly faster than SSD).

5.1 App Direct Mode

In **App Direct Mode**, NVM is exposed as a separate address range (not cached by DRAM):

- Applications can `mmap()` the NVM region and access it directly.
- **Bypasses the page cache** entirely — reads and writes go straight to the persistent medium.
- Requires careful use of `clflush`/`clwb` and memory fences to ensure write ordering for crash consistency.
- Use cases: in-memory databases, persistent key-value stores, file systems (e.g., NOVA, ext4-DAX).

6 Rebuilding Kernel Security

The C/C++ Memory Safety Problem

Approximately **70% of CVEs** in major software projects (Microsoft, Google Chrome, Android) stem from memory-safety bugs in C/C++: buffer overflows, use-after-free, double-free, null pointer dereferences.

6.1 Rust in the Kernel

The **Rust programming language** offers memory safety through its ownership and borrow-checking system, enforced at *compile time*:

Rust Ownership & Borrowing Rules

1. Each value has exactly **one owner** at a time.
2. When the owner goes out of scope, the value is **dropped** (freed).
3. You can have *either* one mutable reference **or** any number of immutable references — never both simultaneously.
4. References must always be **valid** (no dangling pointers).

These rules are enforced by the **borrow checker** at compile time, with **zero runtime overhead**.

- Linux 6.1+ supports Rust for kernel modules.
- Android, Windows, and the Linux kernel are progressively adopting Rust for new drivers and subsystems.
- Eliminates entire classes of vulnerabilities without garbage collection or runtime checks.

7 ARM Memory Tagging Extension (MTE)

ARM's **Memory Tagging Extension** (available in ARMv8.5-A and later) provides hardware-assisted memory safety at sub-page granularity.

- Memory is divided into **16-byte granules**.
- Each granule is assigned a **4-bit color tag** (stored in dedicated tag memory).
- Pointers also carry a 4-bit tag in their top bits.
- On every memory access, the hardware checks that the **pointer tag matches the memory tag**. A mismatch triggers a fault.

What MTE catches:

- **Use-after-free (UAF)**: after `free()`, the allocator re-tags the memory. The stale pointer's tag no longer matches ⇒ fault.
- **Buffer overflow**: adjacent allocations have different tags. Writing past the boundary hits a different tag ⇒ fault.
- Operates at **sub-page** (16-byte) granularity, far finer than page-level protections.

With 4 bits, there are 16 possible tags. The probability of a tag collision (missing a bug) is $1/16 \approx 6\%$, which is acceptable for probabilistic detection in production.

8 Virtual Threads & Concurrent GC

8.1 M:N Multiplexing (Virtual Threads)

Traditional threading models use **1:1 mapping**: one user thread = one kernel thread. This limits scalability to thousands of threads.

Modern runtimes (Java 21 Virtual Threads, Go goroutines, Kotlin coroutines) use **M:N multiplexing**:

- **M** lightweight (virtual) threads are multiplexed onto **N** OS (carrier/platform) threads, where $M \gg N$.
- Each virtual thread has a tiny stack (∼hundreds of bytes to a few KB), grown on demand.
- Enables **millions of concurrent tasks** with minimal memory overhead.
- When a virtual thread blocks (e.g., on I/O), the runtime unmounts it from the carrier thread and mounts another — no kernel context switch needed.

8.2 Concurrent Garbage Collection

Modern GCs (ZGC, Shenandoah, G1) achieve **sub-millisecond pause times**:

- Use **colored pointers** or **load barriers** to perform most GC work *concurrently* with application threads.
- ZGC (Java): max pause < 1 ms regardless of heap size (up to 16 TB).
- Trade-off: slight throughput reduction ($\sim 5-10\%$) $for dramatically lower latency$.
- Implication for OS memory: GC-managed heaps interact with the VM subsystem; large heaps benefit from huge pages and NUMA-aware allocation.

9 Formal Verification of Memory-Critical Software

"Code is not merely tested; it is a mathematical theorem with a machine-checked proof."

Formal verification provides the **strongest** guarantee of correctness: a mathematical proof, checked by a computer, that the code satisfies its specification for *all* possible inputs — not just tested inputs.

9.1 seL4 Microkernel

- The **seL4** microkernel is the world's first OS kernel with a complete, formal, machine-checked proof of functional correctness.
- The proof guarantees: no buffer overflows, no null-pointer dereferences, no memory leaks, correct access control enforcement.
- Also verified: the compiled binary matches the C source (translation validation).
- Used in high-assurance systems: military, aerospace, autonomous vehicles.

9.2 StarMalloc

- **StarMalloc** is a formally verified memory allocator written in F* / Low*.
- Provides verified guarantees of memory safety, functional correctness, and absence of undefined behavior in the allocator itself.
- Demonstrates that formal verification is practical even for performance-critical systems code.

10 Important Dates

Item	Date	Notes
Project Idea Submission	March 9	Brief description of proposed topic
Project Proposal	March 18	Full proposal with scope and milestones
Final Exam Request	March 9	Submit request if accommodations needed