Server Long Term RAM

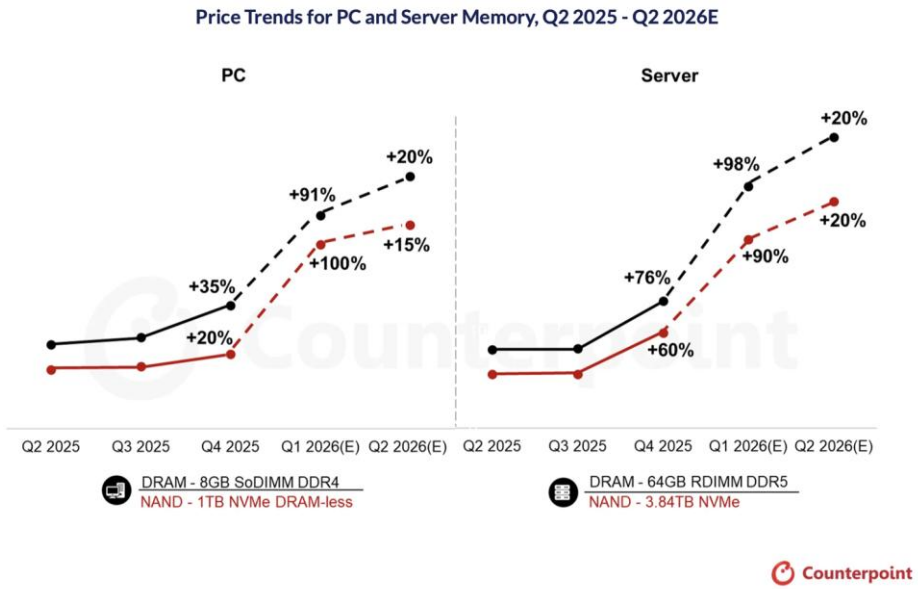
Hui Sub (David) Shim, Katherine Mohr, Philip Levis

MemoryDAX/DAM Summer Retreat 2026

22 June, 2026

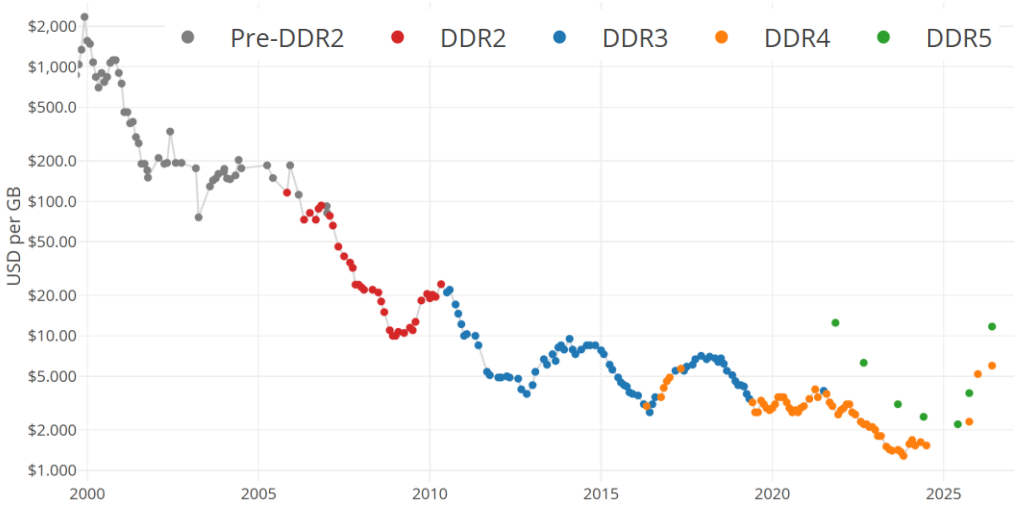
We’ve hit the Memory Wall

- DRAM dominates server cost
  - Over half total server cost (Azure, Meta)
- DRAM prices on the rise
  - ~3x increase from 2025 Q4 to 2026 Q1
  - DRAM cost scaling is slowing
- Only worsens as
  - CPUs get faster, processing more memory
  - HBM demand increases due to AI



We've hit the Memory Wall

- DRAM dominates server cost
  - Over half total server cost (Azure, Meta)
- DRAM prices on the rise
  - ~3x increase from 2025 Q4 to 2026 Q1
  - DRAM cost scaling is slowing
- Only worsens as
  - CPUs get faster, processing more memory
  - HBM demand increases due to AI



https://dam.stanford.edu/memory-prices.html

As DRAM cost scaling is slowing, we must investigate other forms of RAM.

A Class of Emerging Memories: LtRAM

+ **Denser, cheaper** per bit
+ **Read latencies** are competitive with DRAM

- **Write latencies** are orders of magnitude slower
- Hard **endurance limits**
- Do not naturally read/write at **cache-line granularity**

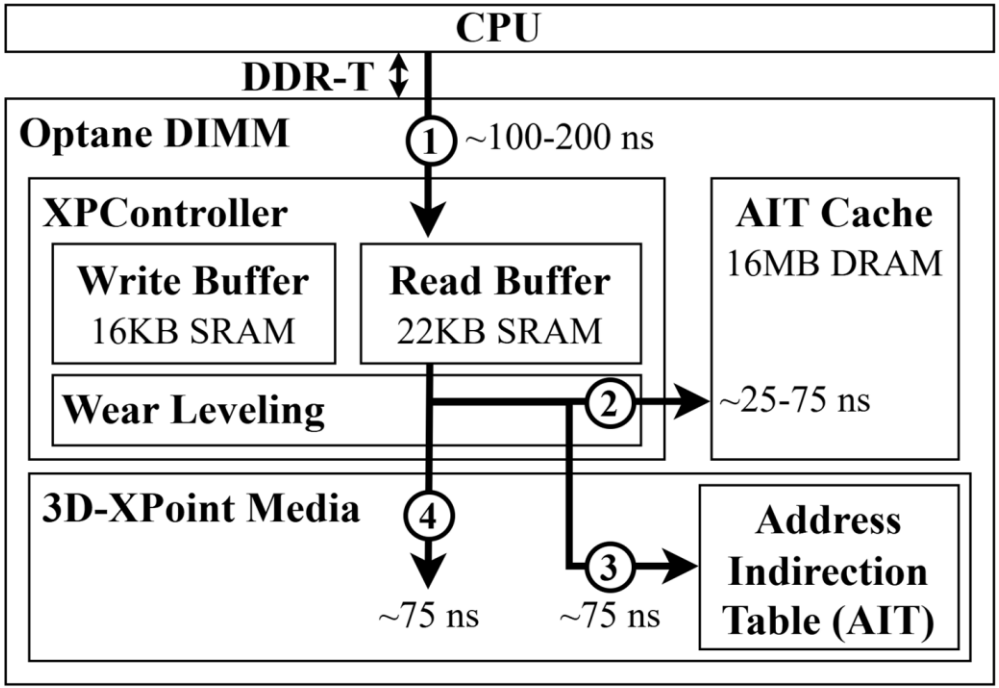
Device	Read lat.	Write lat.	Endur-ance	Write gran.	Cost per bit
DRAM	80 ns	80 ns	∞	cache line	high
3D V-RRAM	100 ns	1 μs	10^6	page	low
3D FeFET	200 ns	10 μs	10^5	page	low
PCM	300 ns	1 μs	10^8	256 B	mid
STT-MRAM	20 ns	20 ns	10^{15}	word	high

Table 1. Representative LtRAM memories and their characteristic shape, with DRAM as reference. Numbers should be read for shape rather than precision, as each row can shift by orders of magnitude with material and circuit choices [15]

Overview

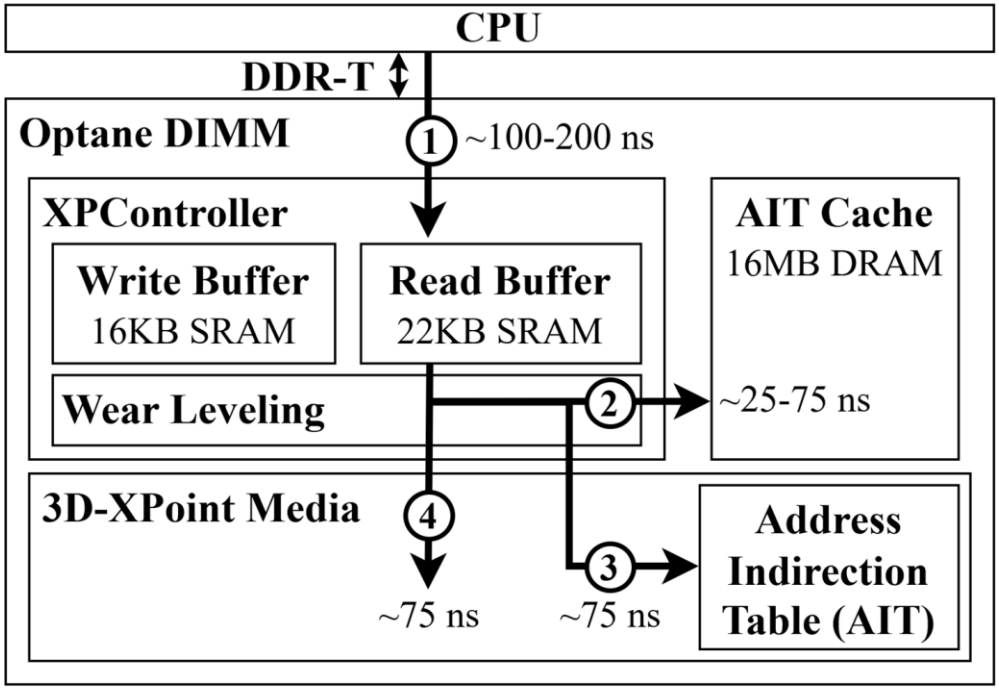
- **Optane deep dive:** the cost of making asymmetric media look like a DIMM
- **Proposed interface:** a HW/SW contract enabling Application Read-Only Memory
- **Prototype:** Enzian + NOR flash as a first LtRAM prototype
- **Latency projections:** what future LtRAM devices could look like
- **Open questions:** placement and wear management policies

Prior Attempt: Intel Optane



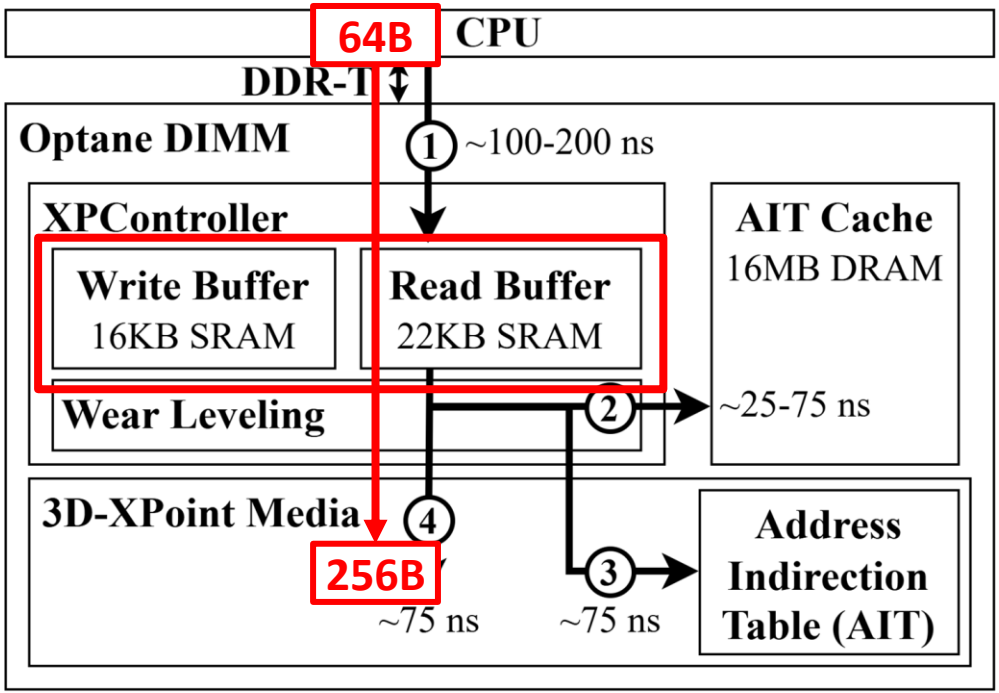
- Byte-addressable, non-volatile DIMM
- Exposes a **DRAM-compatible interface** for accessing 3D-XPoint media (Phase Change Memory)
- Translates DRAM-style accesses behind a **complex on-DIMM controller**

Optane's Controller Overhead



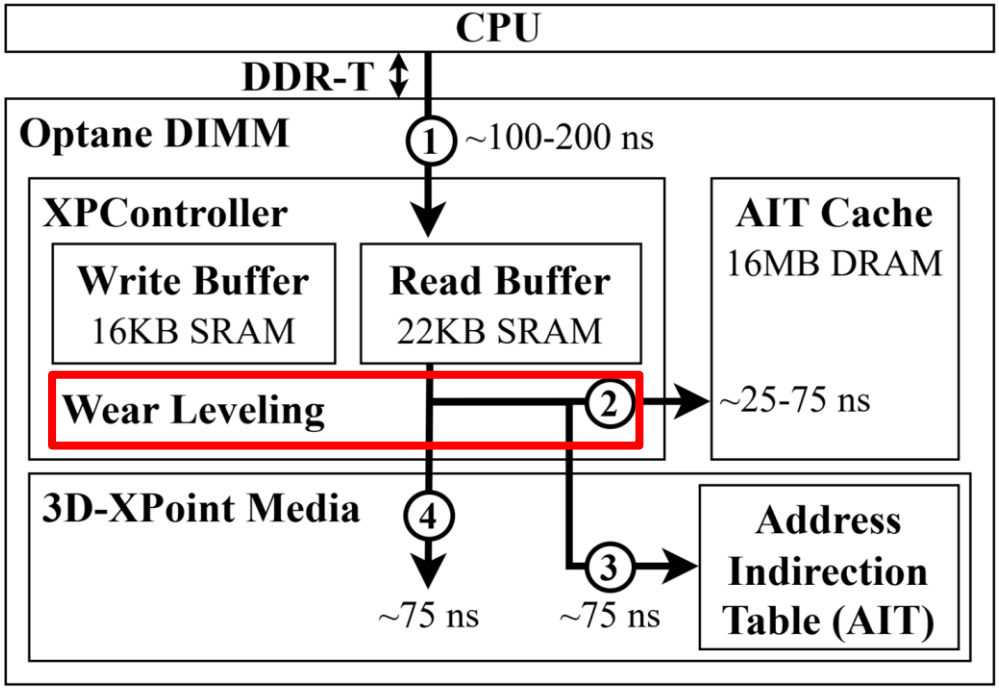
1. Granularity mismatch
2. Hardware wear-leveling
3. Address Indirection Table
4. Workload-Agnostic Design

Optane's Controller Overhead



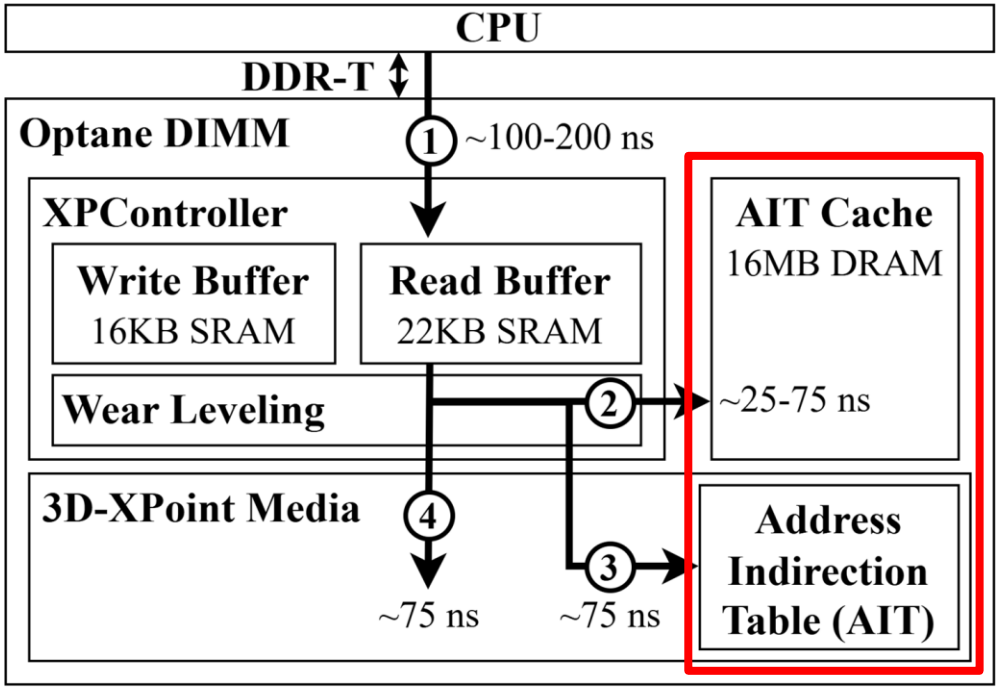
1. **Granularity mismatch**
   - 64B CPU cache line vs 256B XPLine
   - **~75%** write throughput drop
   - **2x** random-read latency
2. Hardware wear-leveling
3. Address Indirection Table
4. Workload-Agnostic Design

Optane's Controller Overhead



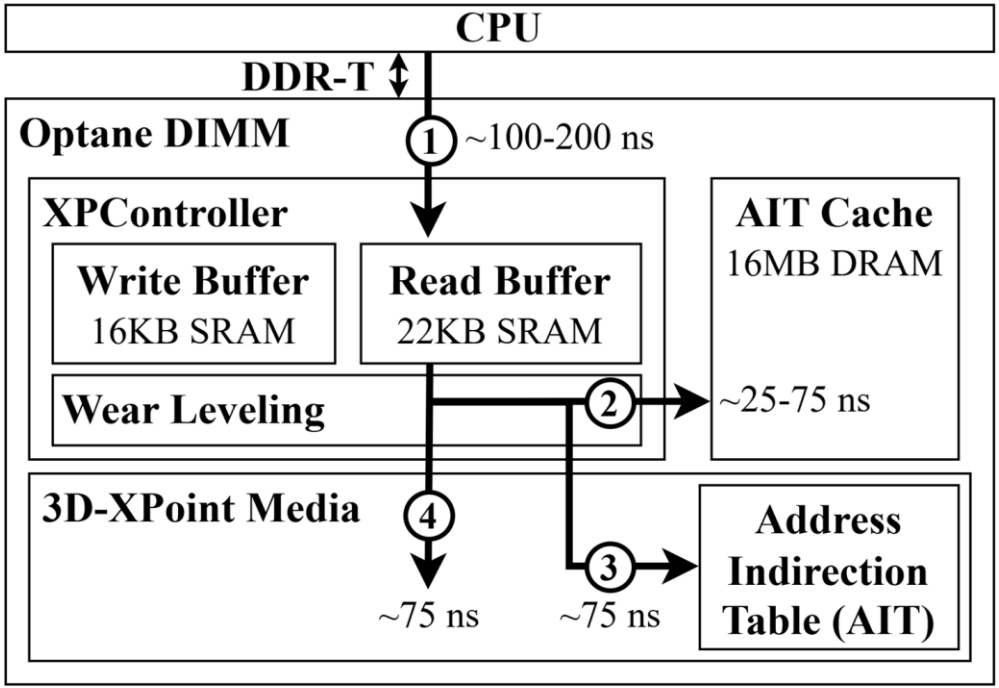
1. Granularity mismatch
2. **Hardware wear leveling**
   - Internal migration every ~3.4MB
   - **~60us** tail read latency
     (**~160x** typical random-read latency)
3. Address Indirection Table
4. Workload-Agnostic Design

Optane's Controller Overhead



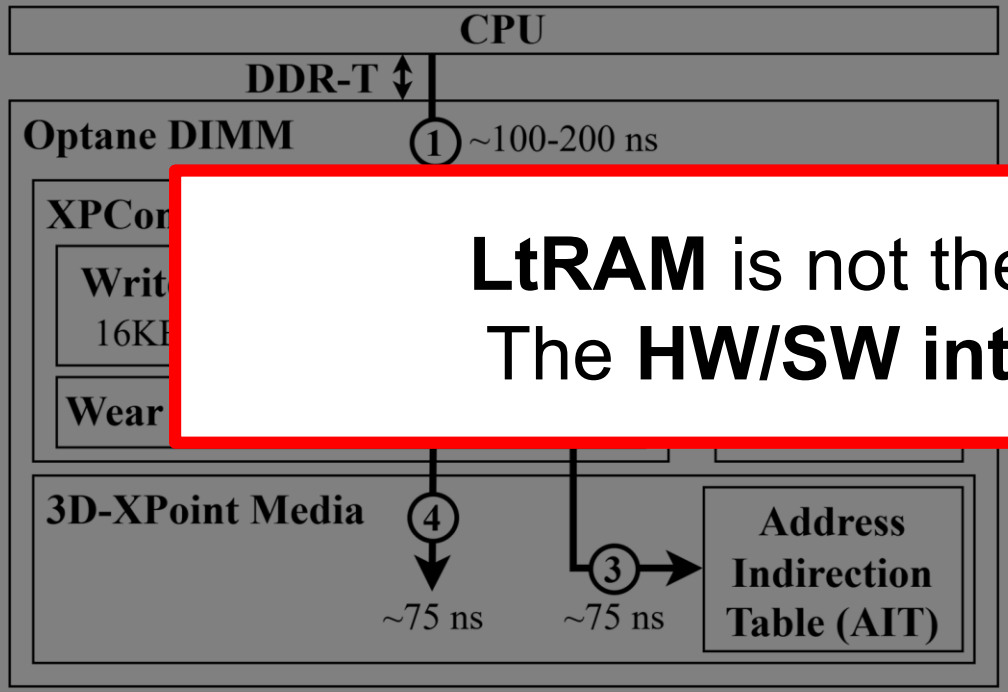
1. Granularity mismatch
2. Hardware wear leveling
3. **Address Indirection Table (AIT)**
   - Map from physical address to media
   - Necessary for wear leveling
   - **~76-200ns** latency added on every read
4. Workload-Agnostic Design

Optane's Controller Overhead



1. Granularity mismatch
2. Hardware wear leveling
3. Address Indirection Table (AIT)
4. **Workload-Agnostic Design**
   - **70% read-bandwidth loss** under 50/50 R/W traffic unlike DRAM

Optane's Controller Overhead

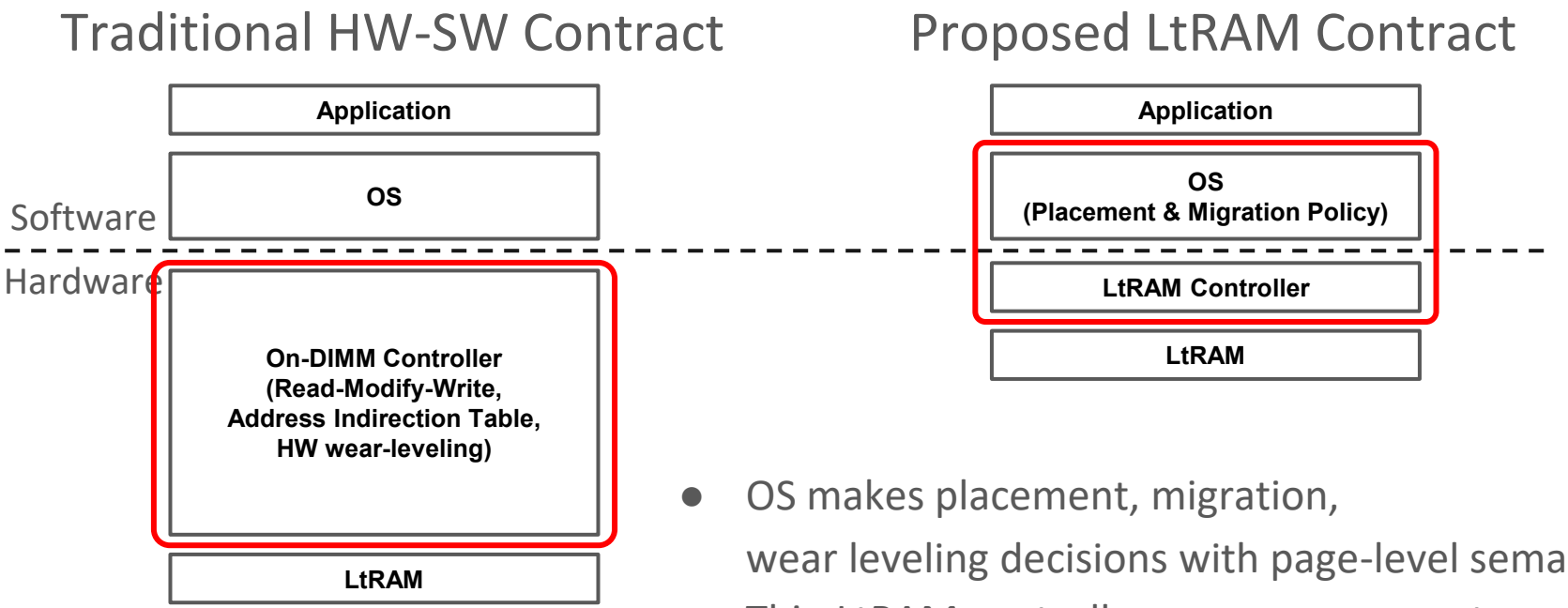


1. Granularity mismatch
2. Hardware wear leveling

AIT)

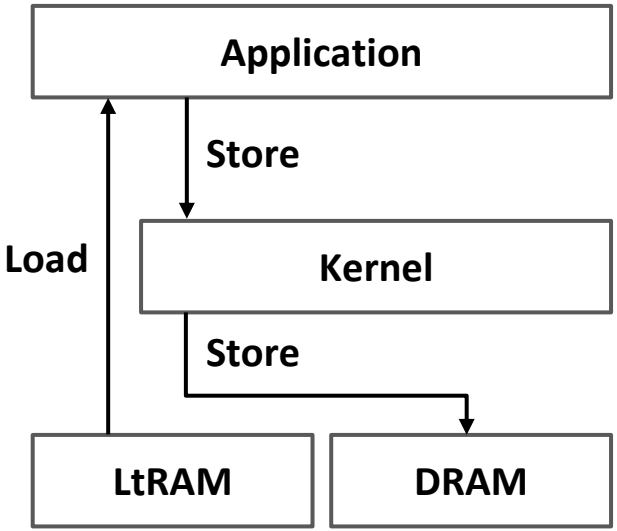
LtRAM is not the problem.
The **HW/SW interface** is.

New Contract: Move Policy from Controller to OS



- OS makes placement, migration, wear leveling decisions with page-level semantics
- Thin LtRAM controller exposes asymmetry

LtRAM as Application Read-Only Memory (AROM)



- Application loads access LtRAM directly
- Stores to LtRAM pages never update LtRAM in place.
- Kernel writes LtRAM only during controlled migration

Application view:

- Ordinary load/store interface
- No explicit DRAM/LtRAM placement management

Controller Responsibilities

1. Serve cache-line reads
    - Preserve direct load access from applications
2. Accept 4KB page-granular writes
    - Aligned to LtRAM’s write boundary
    - Eliminates sub-page Read-Modify-Write
3. Expose cell health metadata
    - OS makes placement/wear-leveling decisions with device-state visibility

OS Responsibilities

1. Allow LtRAM writes only through the kernel migration path at 4KB granularity
2. Leverage Copy-on-Write to prevent applications from writing directly to LtRAM
    - A store to an LtRAM page faults, migrates the page to DRAM, then applies the write to DRAM
3. Admit only read-mostly pages to LtRAM
    - Placement policy keeps write-active data in DRAM and avoids mixed R/W traffic
4. Balance wear in software
    - OS-managed placement and token budgeting replace opaque hardware wear-leveling

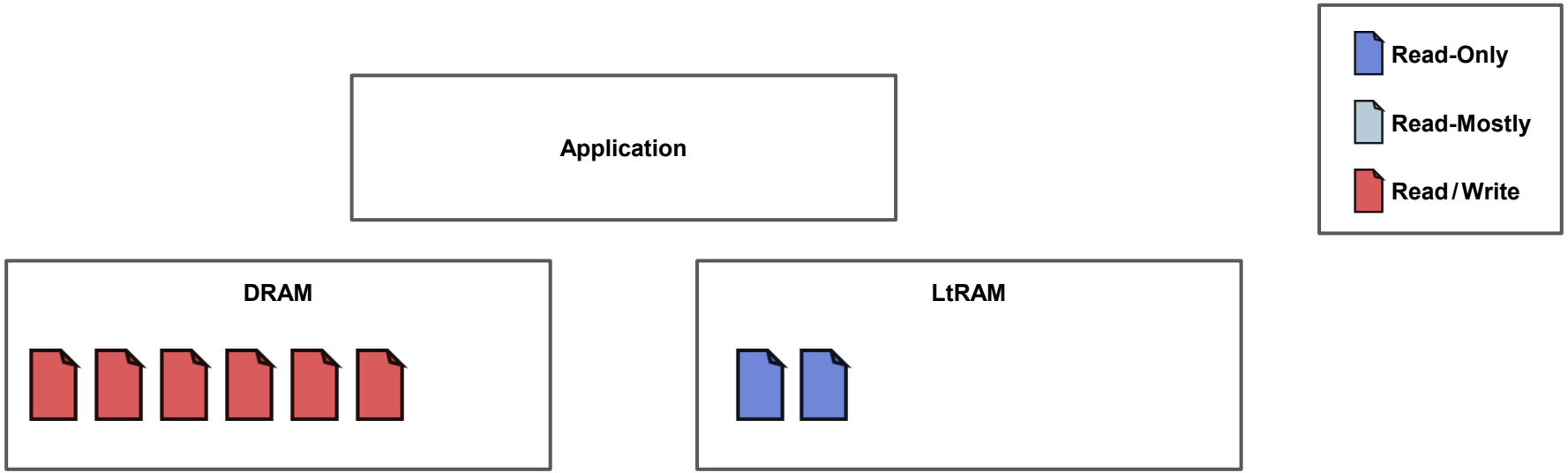
Page Lifecycle under the LtRAM Contract

1. Allocation
2. Migration to LtRAM
3. Migration to DRAM

Page Lifecycle under the LtRAM Contract

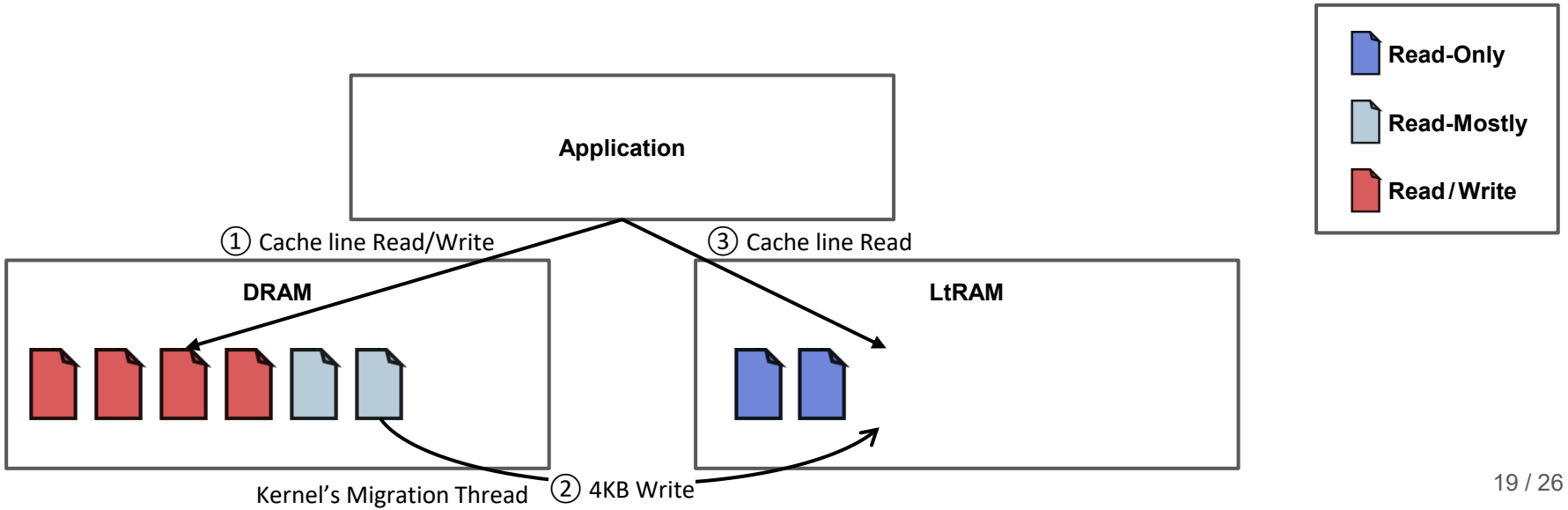
1. Allocation

- Pages default to DRAM unless explicitly marked as read-only (executable code, shared libraries, read-only memory-mapped files)



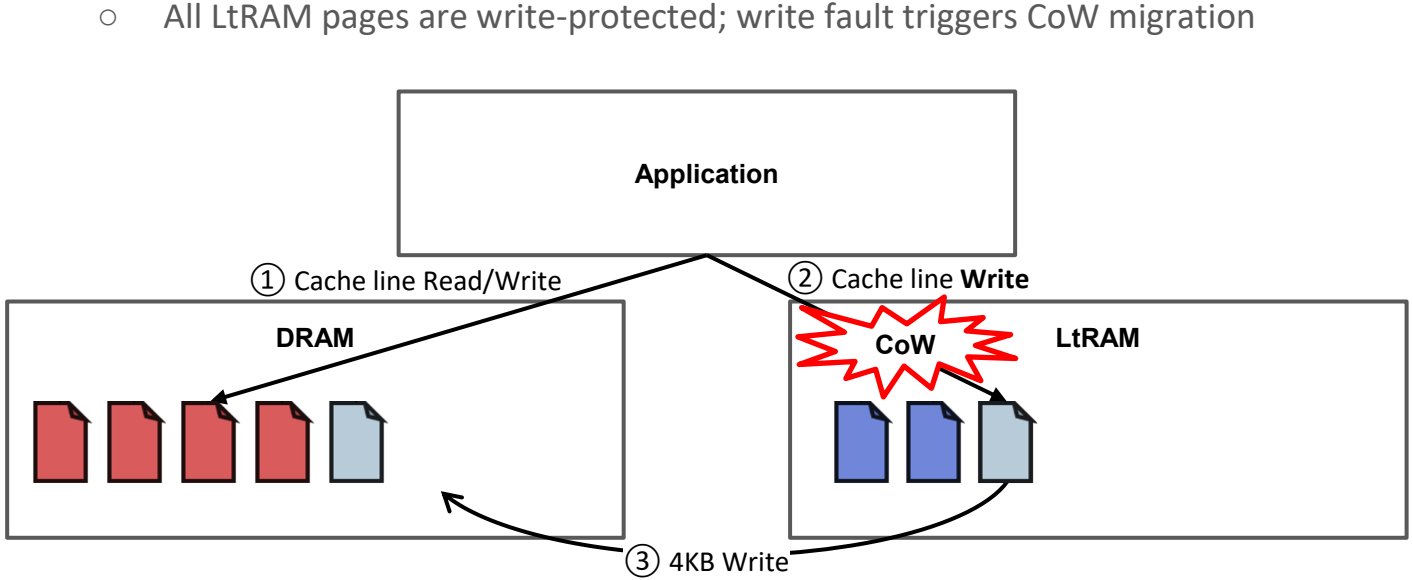
Page Lifecycle under the LtRAM Contract

1. Allocation
2. **Migration to LtRAM**
   - Migrate pages predicted to stay read-mostly



Page Lifecycle under the LtRAM Contract

1. Allocation
2. Migration to LtRAM
3. **Migration to DRAM**
    - All LtRAM pages are write-protected; write fault triggers CoW migration



Placement Policy

Problem

- When should a DRAM page move into LtRAM?
- **Bad placement**: pages fault back to DRAM, wasting **migration bandwidth and endurance**
- **Too conservative**: LtRAM **underutilization** and continued **DRAM pressure**

Candidate Policy Mechanisms

- **Dirty-bit classifier:** cheap baseline using existing kernel state
- **Online predictor:** tracks recent write behavior with small metadata
- **Memory Fit:** hardware-assisted per-page fit score for DRAM vs. LtRAM

Wear Leveling Policy

Problem

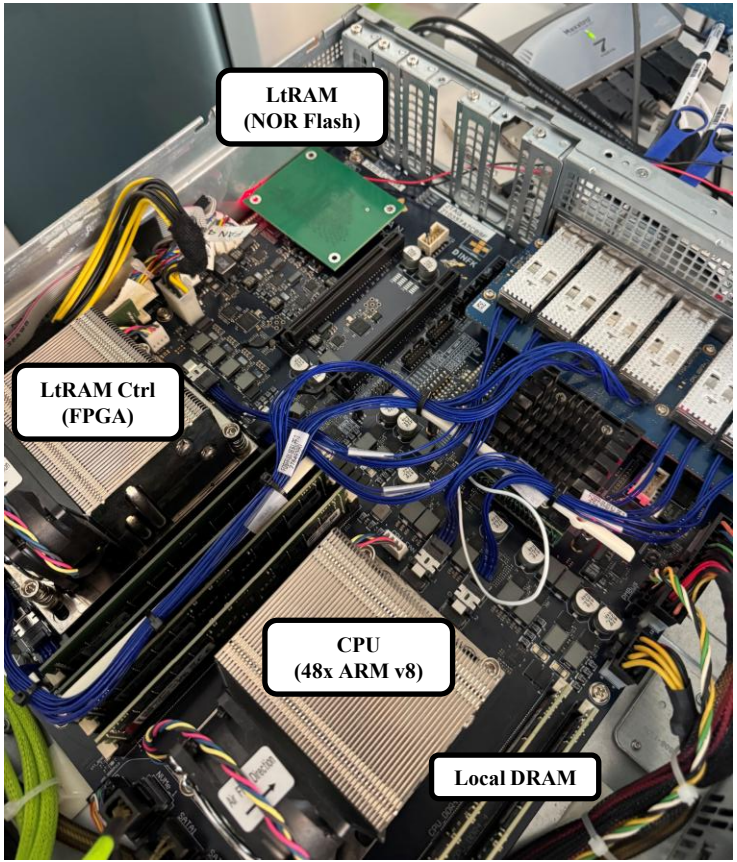
- How can the OS **preserve LtRAM lifetime** over the target deployment period?
- Pinned read-only data: force new writes onto a **smaller active pool**

- Too many migrations: LtRAM **tail latency** spike, **LtRAM endurance**

Candidate Policy Mechanisms

- **Endurance tokens:** limit total DRAM → LtRAM writes over time
- **Wear-ordered free list:** allocate least-worn free pages first
- **Read-only rebalancing:** relocate read-only data to recycle low-wear capacity
- **Start-Gap:** periodically remap pages to spread wear with minimal metadata

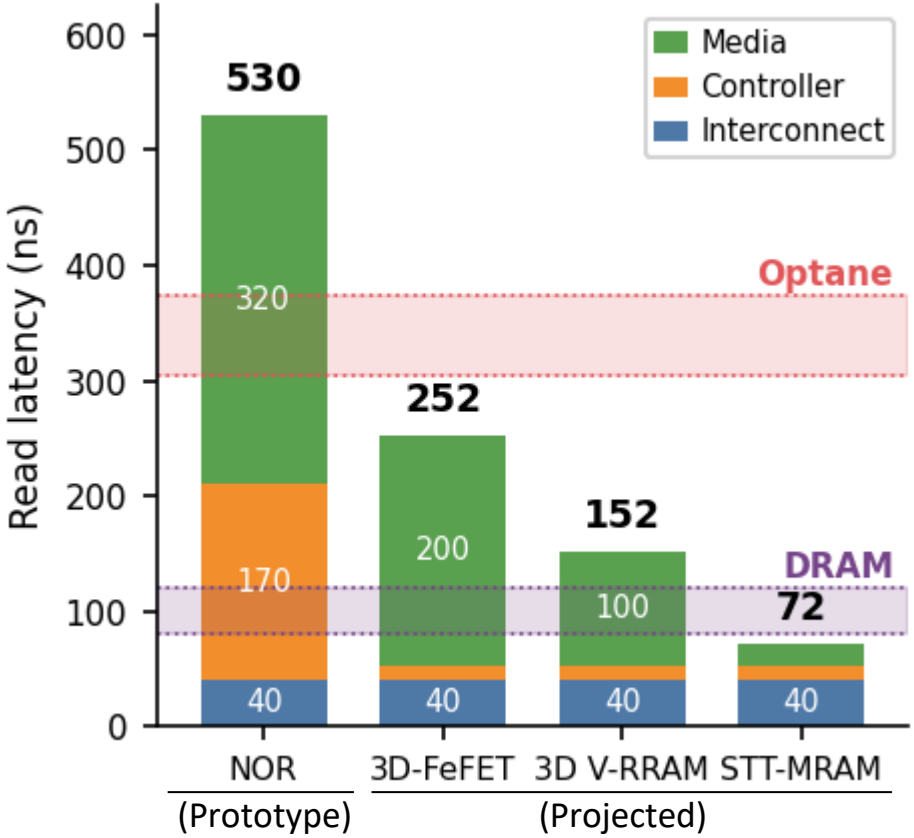
Prototype: Enzian + NOR flash



- **Enzian** research platform: ARM CPU + coherent FPGA fabric
- **CPU runs Linux 6.8** with LtRAM extensions
- FPGA acts as the **LtRAM controller**
- 256 MB Micron NOR flash used as the LtRAM

- Why NOR?
  - Asymmetry of LtRAM, Availability, Cost
  - Read (128 B): 490 ns
  - Write (256 B): 16 µs
  - Erase (4 KB): 18 ms

Projected Read Latency



- Projection Method
  - Hold interconnect constant at 40 ns
  - Reclock prototype controller: 170 → 12 ns
  - Add media latency from Table 1
- Reference points
  - 26–79% faster than Optane (300-400ns)
  - MRAM 10% faster than DRAM (80-120ns)
  - RRAM and FeFET are 1.9x-3.2x DRAM latency

Conclusion

- **DRAM cost pressure** motivates cheaper main-memory capacity
- LtRAM is **promising**, but not as a direct DRAM replacement
  - Optane shows the cost of hiding asymmetry in hardware
- LtRAM needs a **different HW/SW contract** to function as AROM
  - The controller provides cache-line reads, page-granular writes, and device health visibility
  - The OS manages placement policy, migration policy, CoW enforcement, and wear management

- Keep the controller thin, manage policy in software, and reduce DRAM footprint to lower server cost.

Thank you

hsshim@stanford.edu, kmohr@cs.stanford.edu