

Advanced AI Computing and Systems

Lecture 2: Memory Evolution — Memory Hierarchy and SOTA

Prof. Kai Chen

Key Question

How to design memory systems to fully utilize modern CPUs/GPUs, especially in the AI era?

Today's Path

[Review] Von Neumann Architecture.

[Core] Memory wall: performance is limited by data supply.

[Core] CPU vs GPU memory subsystem.

[Concept] Memory Technology Evolution.

Readings

Wulf & McKee. [Hitting the Memory Wall: Implications of the Obvious](#). SIGARCH Computer Architecture News, 1995.

Goto & van de Geijn. [Anatomy of High-Performance Matrix Multiplication](#). ACM TOMS, 2008.

Dylan Patel et al. [Scaling the Memory Wall: The Rise and Roadmap of HBM](#). SemiAnalysis, August 12, 2025.

Rodrigo Huerta et al. [Dissecting and Modeling the Architecture of Modern GPU Cores](#). MICRO, 2025.

Fetch-and-Execute Workflow

Fetch instructions and data from

memory

Execute instructions on ALU

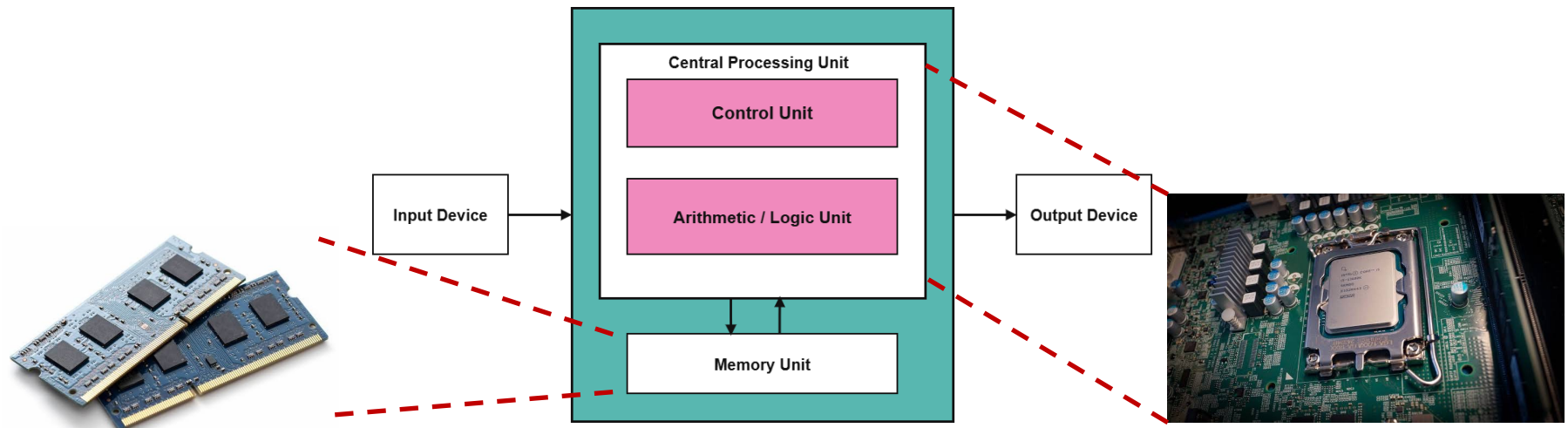
Write back computation results to

memory

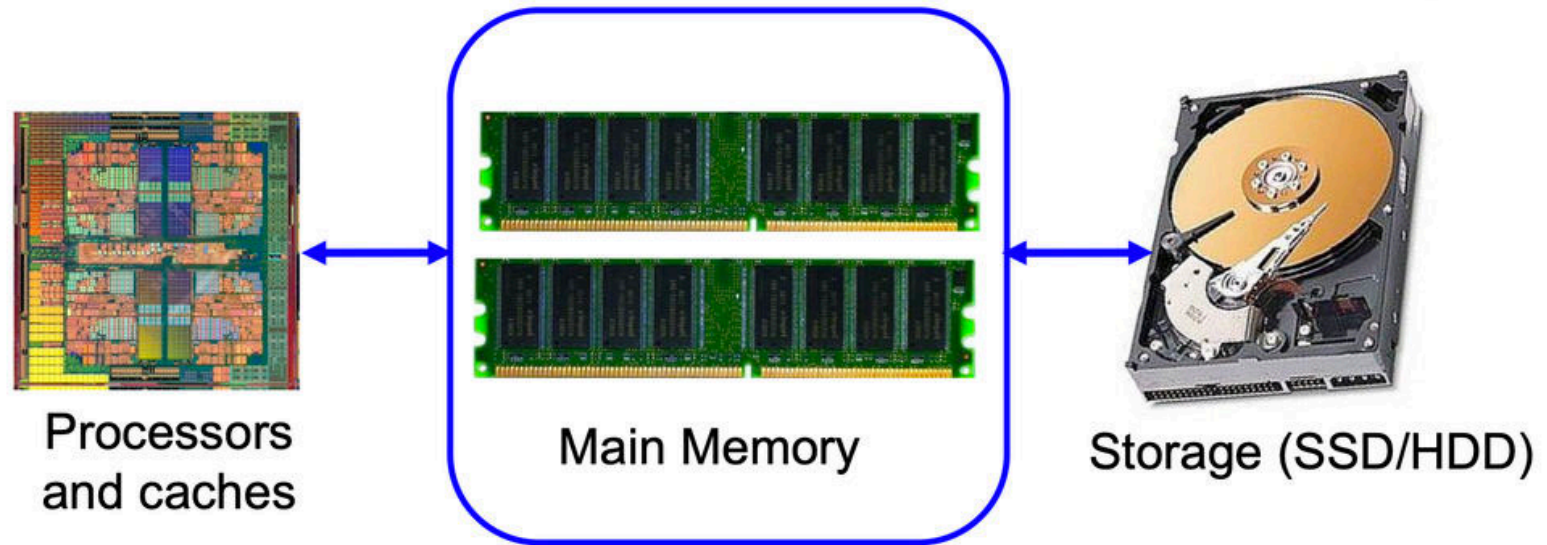
Separation of memory and PU

Bottleneck: Memory Wall

PU execute instructions orders of magnitude faster than memory access!



The Main Memory System

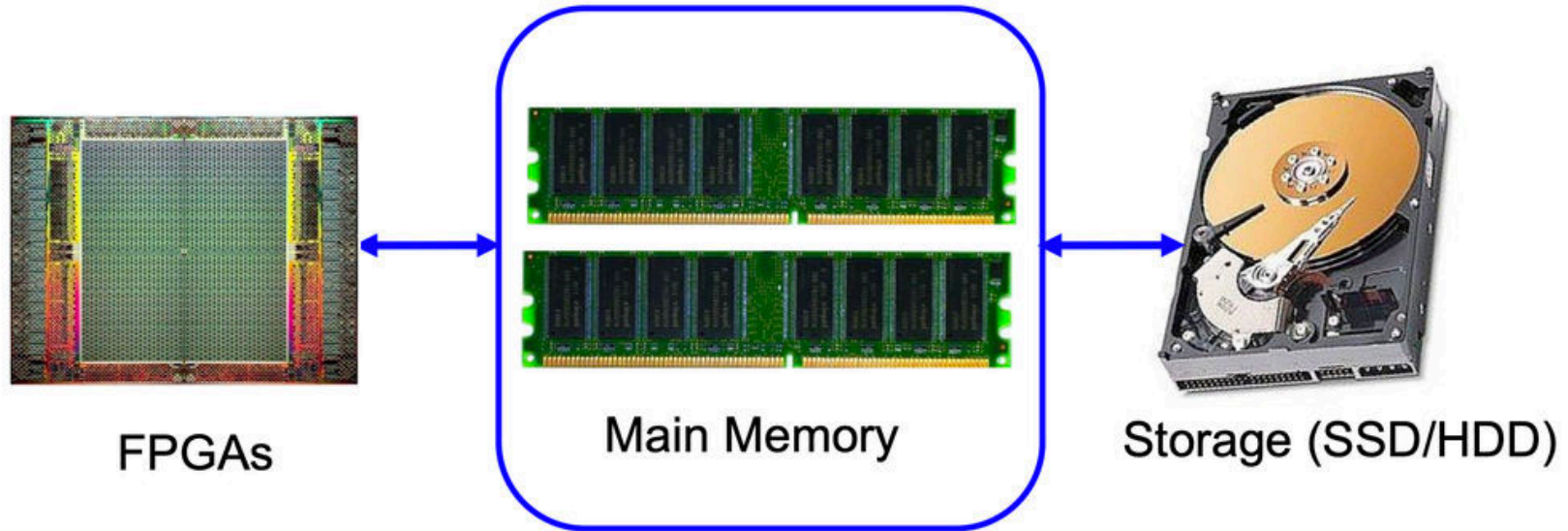


Main memory is a critical component of all computing systems

Server, mobile, embedded, desktop, sensor

- Storage retains programs and data across runs. Main memory holds the portions available for active execution.
- Caches keep copies close to the execution units, reducing transfers from main memory.

The Main Memory System

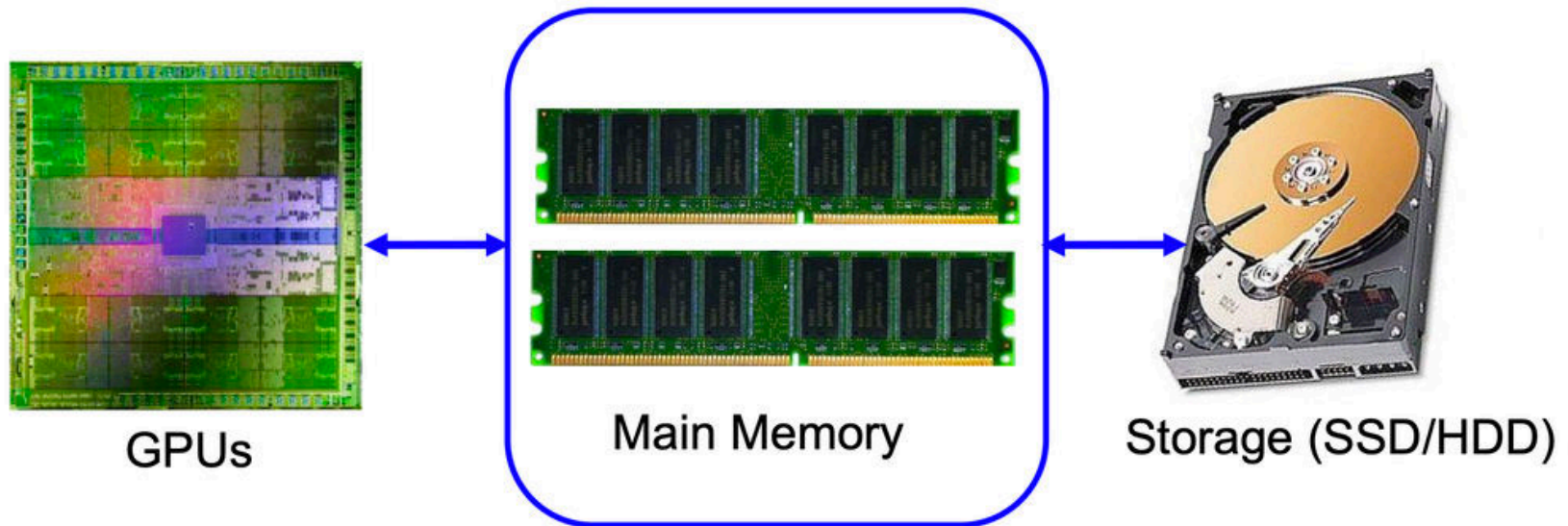


Main memory is a critical component of all computing systems

Server, mobile, embedded, desktop, sensor

- An FPGA configures logic blocks and their connections into a circuit for a particular task.
- On-chip SRAM, such as block RAM, buffers inputs and intermediate results near the processing logic.

The Main Memory System

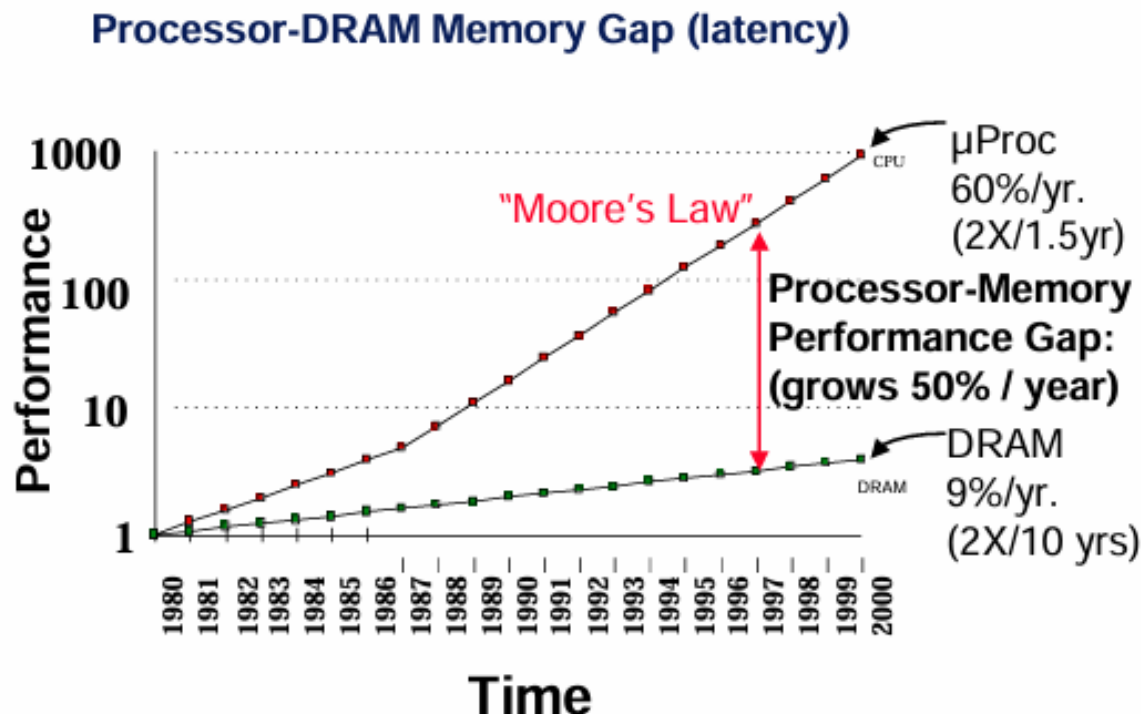


Main memory is a critical component of all computing systems

Server, mobile, embedded, desktop, sensor

- A discrete GPU uses its own device DRAM: GDDR chips or HBM stacks.
- Results can stay in device memory for the next computation, or return to CPU memory when needed.

Processor-DRAM Performance Gap grows 50% per Year



Adapted from: J. Kubiawicz, "Memory and Caches," UC Berkeley CS152, 1999.

- The 50% gap describes the historical period in this graph. Faster processors alone do not guarantee faster applications.

Scaling Laws Meet Memory Walls

Scaling Laws of LLM Workloads

Increased amount of parameters and data.

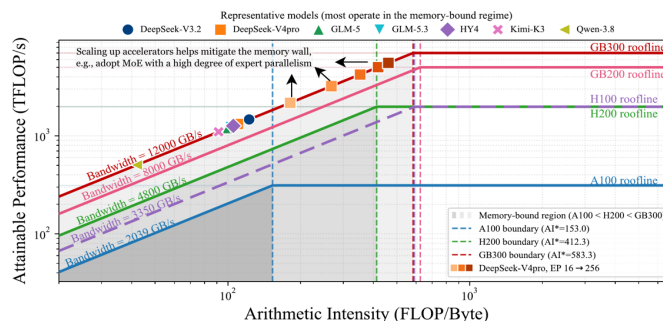
Increased chain-of-thought length and search depth.

Larger **Model Size** and **Context State**

Post-Dennard Scaling of AI Accelerators

Tensor Core throughput has been growing much faster than the capacity of HBM.

Widening the **Memory Wall** across Generations



The arithmetic intensity shifts towards memory-bound as model size and context length increase.

Memory-bound region keeps growing across GPU generations.

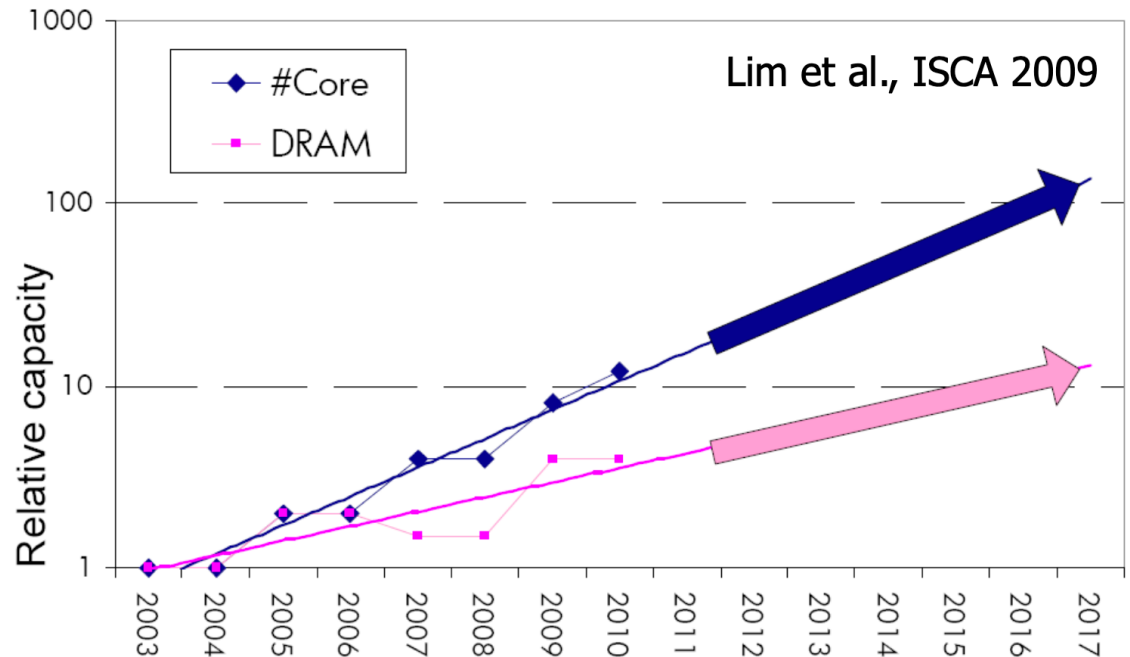
Capacity, Bandwidth, and Latency

PU core number
doubling every ~2 years.

DRAM DIMM capacity,
however, doubling
every ~3 years.

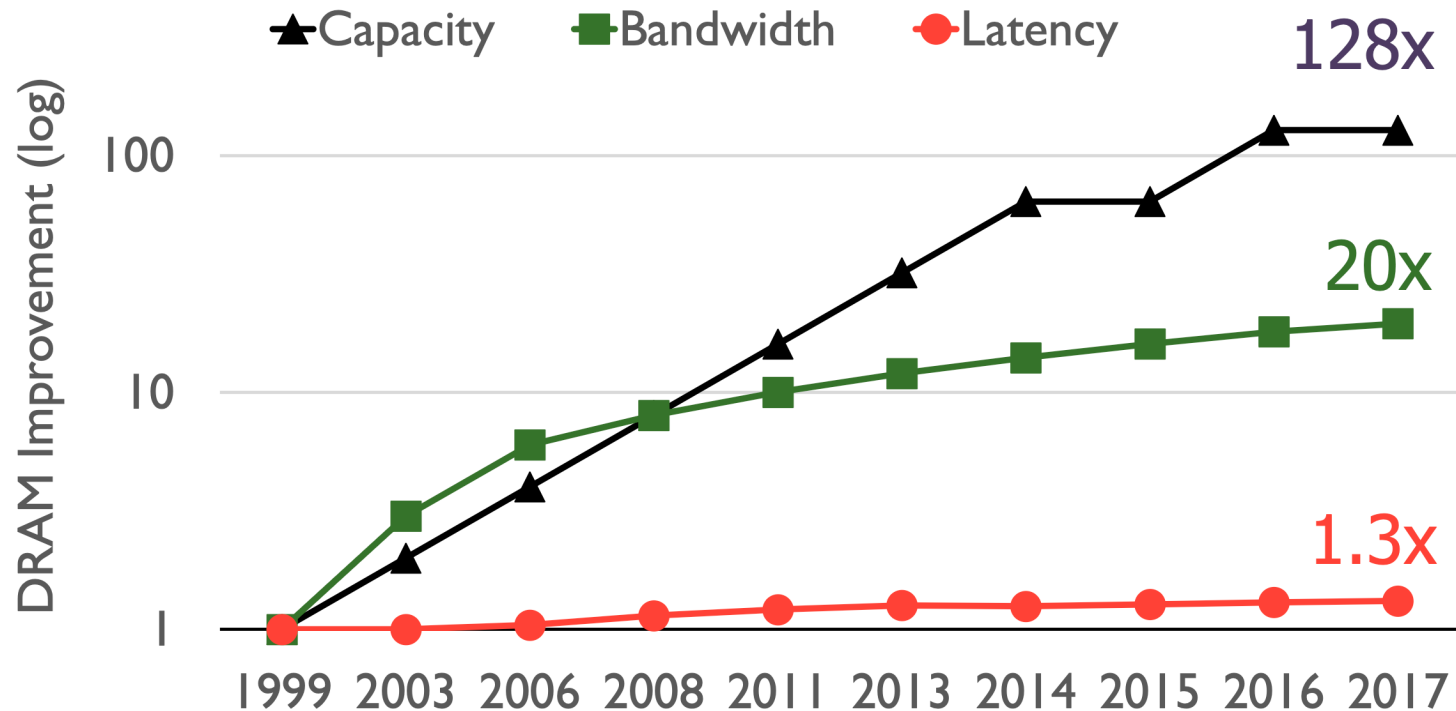
Memory capacity per
core expected to **drop**
by 30% every two years

Much data may load
from low-speed disk!



Adapted from: K. T. Lim et al., “Disaggregated Memory for Expansion and Sharing in Blade Servers,” ISCA, 2009, Fig. 1(a). 2011–2017 projected.

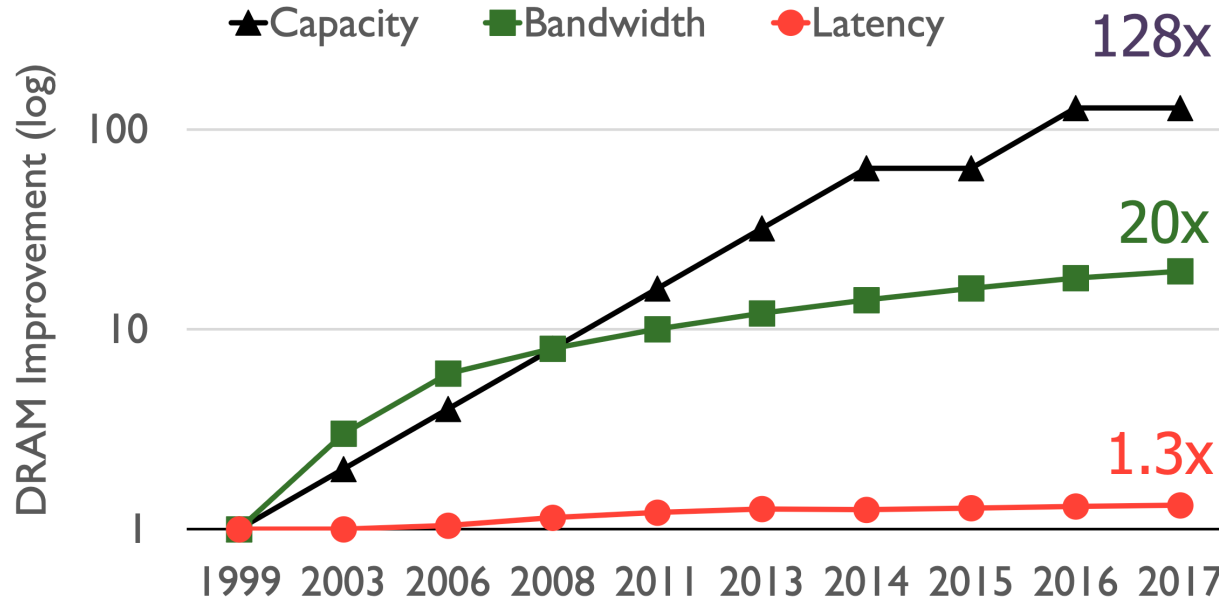
Capacity, Bandwidth, and Latency



Adapted from: K. K. Chang, "Understanding and Improving DRAM Latency," Ph.D. dissertation, CMU, 2017, Fig. 1.1.

Trends of bandwidth and latency are worsen.

Capacity, Bandwidth, and Latency



Adapted from: K. K. Chang, "Understanding and Improving DRAM Latency," Ph.D. dissertation, CMU, 2017, Fig. 1.1.

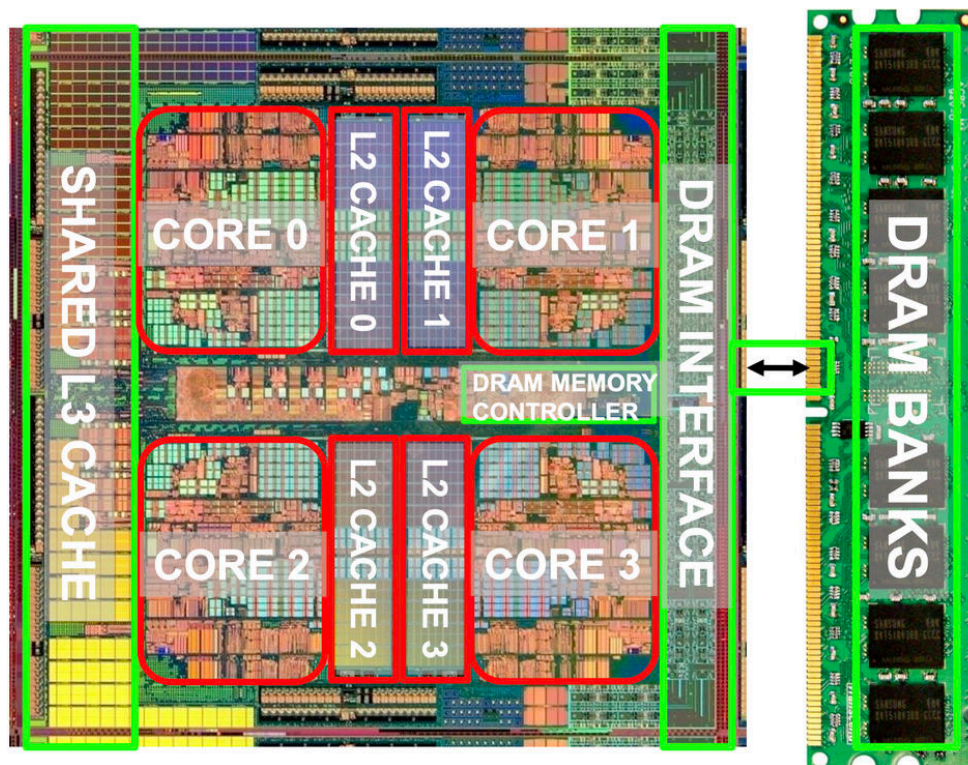
Trends of bandwidth and latency are worsen.

In modern computing systems,
memory performance typically the bottleneck of entire systems.

Deeper and Larger Cache Hierarchy

AMD Barcelona, 2007

Source: O. Mutlu, “Digital Circuits and Computer Architecture,” ETH Zürich, 2017; the image: AMD.

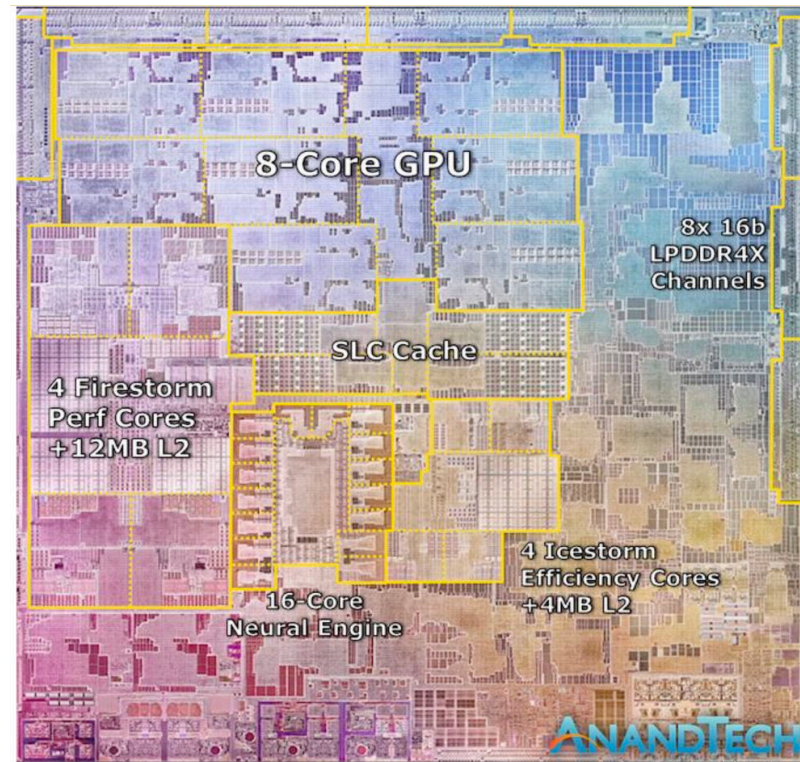


- Private caches keep frequently used data near each core. A shared cache supports reuse across the processor.

Deeper and Larger Cache Hierarchy

Apple M1, 2020

Source: A. Frumusanu,
“The 2020 Mac Mini
Unleashed,”
AnandTech, 2020.



- The die dedicates substantial area to on-chip storage, keeping data near the CPU and GPU execution resources.

Deeper and Larger Cache Hierarchy

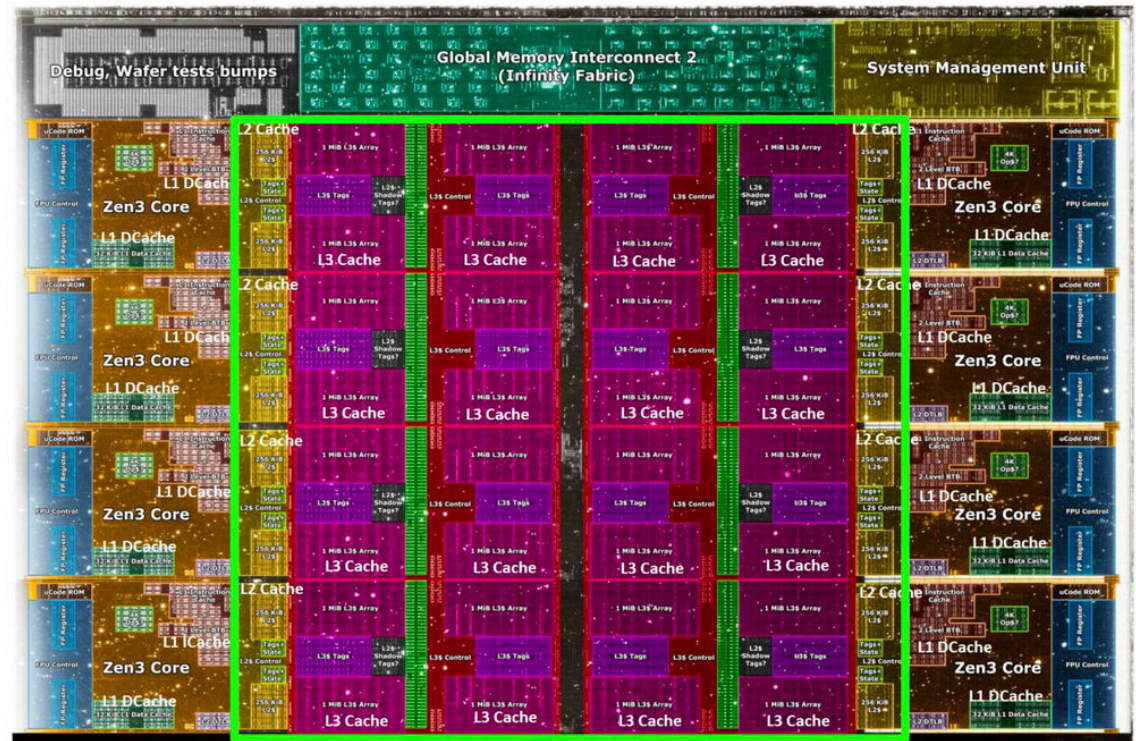
AMD Ryzen 5000, 2020

8 cores

L1: 32KB per core

L2: 512KB per core

L3: 32MB shared



- A shared last-level cache can retain data used by several cores before a request reaches DRAM.

Deeper and Larger Cache Hierarchy

IBM POWER10, 2020

- 15-16 cores
- L2: 2MB per core
- L3: 120MB shared



- Cache capacity helps reduce traffic to main memory, while access time still depends on the location of the data.

Deeper and Larger Cache Hierarchy

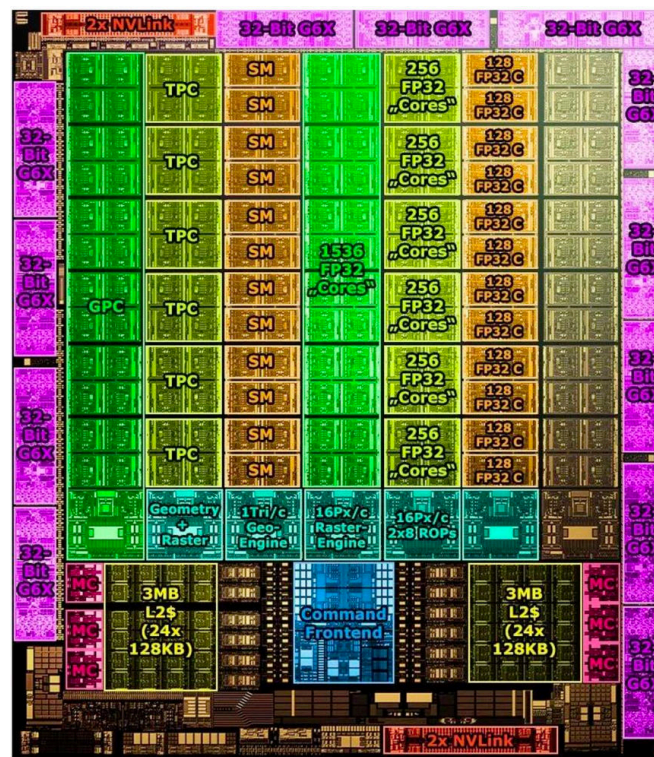
Nvidia Ampere (GA102), 2020

L1: 128KB per SM

L2: 6MB shared

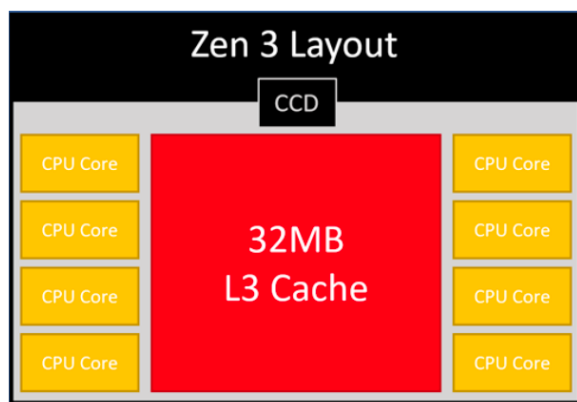
NVIDIA A100

Accelerator shares the same architecture, but includes more SMs and L2 caches.



- Many SMs share the device-wide L2 cache. A larger accelerator needs more capacity and data supply.

Deeper and Larger Cache Hierarchy



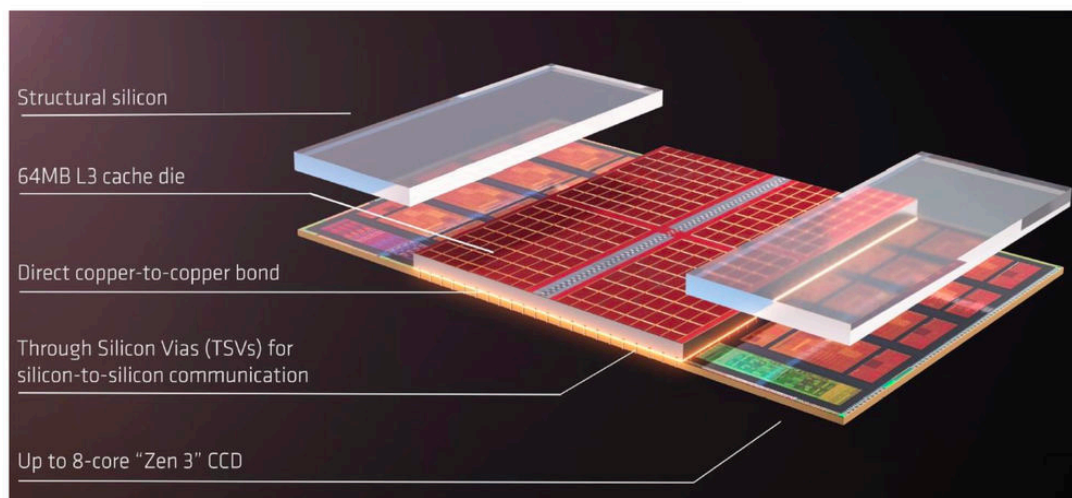
<https://community.microcenter.com/discussion/5134/comparing-zen-3-to-zen-2>

Additional 64 MB L3 cache die

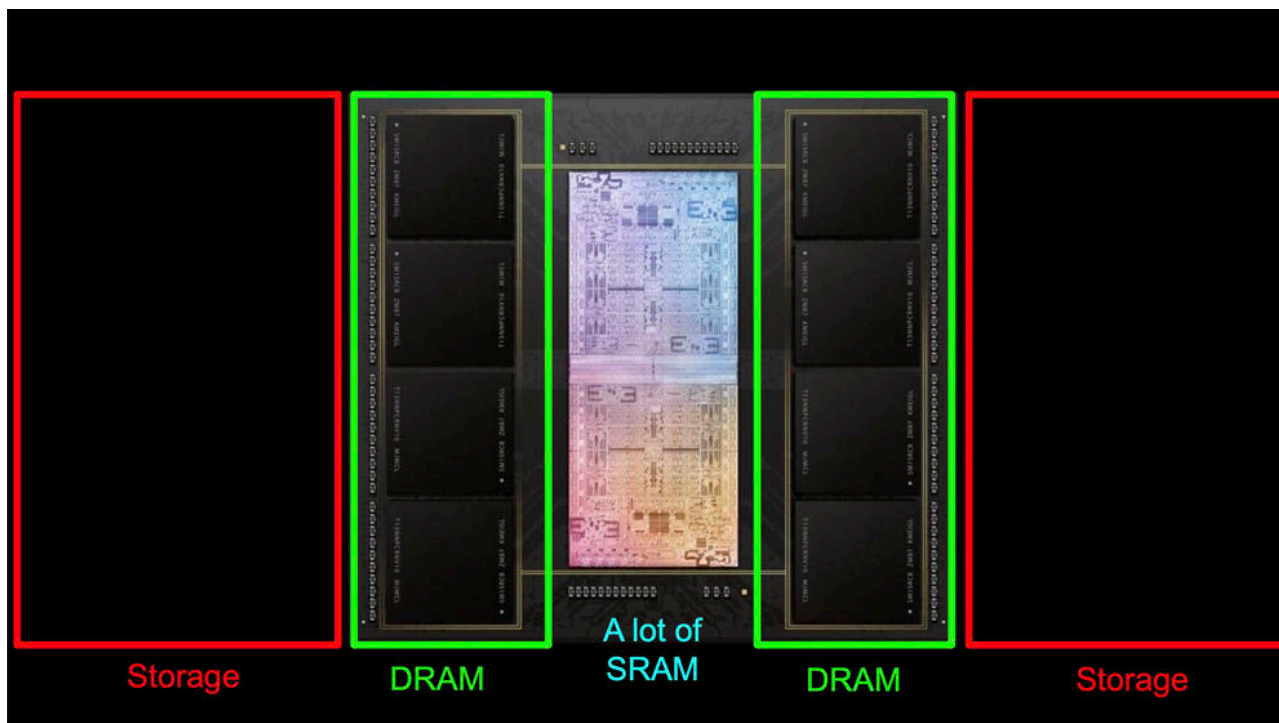
Stacked on top of the processor die

Total of 96 MB L3 cache

AMD increases the L3 size of their 8-core Zen 3 processors from 32 MB to 96 MB



Deeper and Larger Cache Hierarchy



Apple M1 Ultra System, 2022

- The system places multiple storage levels around its compute resources, balancing capacity, access time, and reuse.

CPU Memory Hierarchy

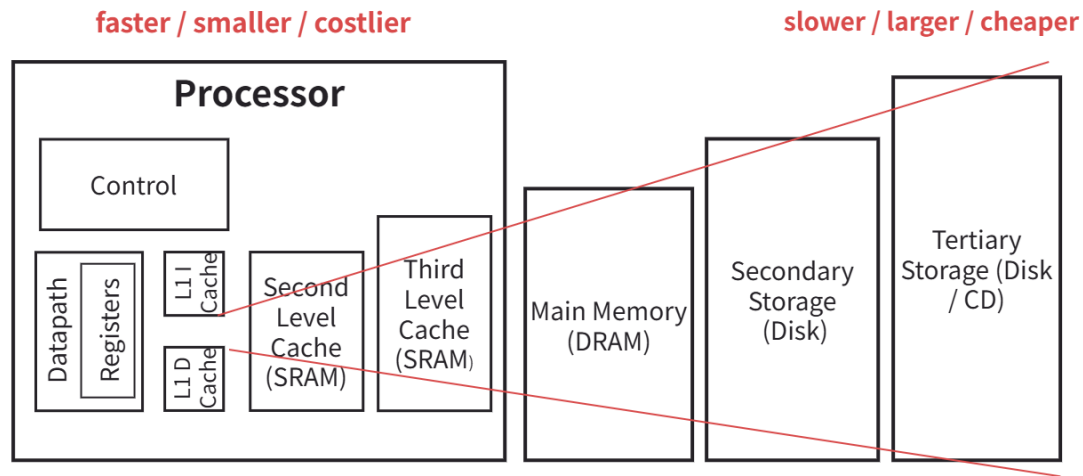
Lecture 2: Memory Evolution — Memory Hierarchy and SOTA

Locality and cache performance
DRAM organization and data delivery

Memory Hierarchy: Taking Advantage of Locality

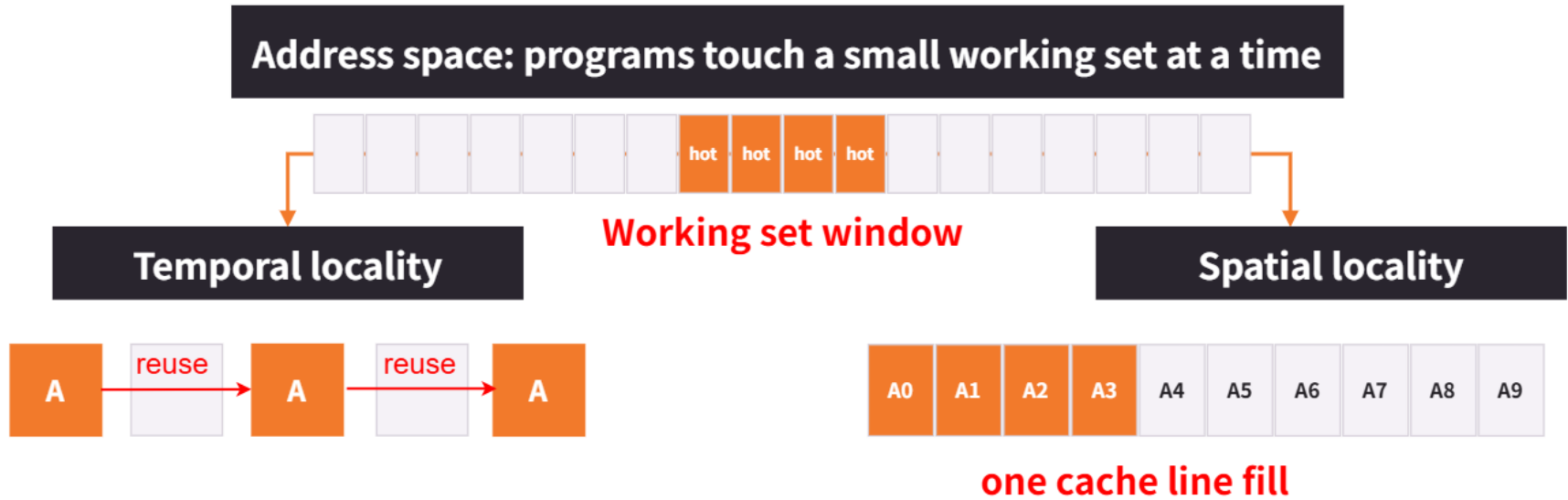
For frequent data: use fast but expensive and small memory (e.g., cache)

For not frequent data: use slow but cheap and large memory (e.g., main memory, disk)



- The working set changes over time. The hierarchy is useful when a small, active subset receives most accesses.

Locality: The Hot Feature Makes Caches Useful



Temporal Locality

A piece of data is quickly reused in the future.

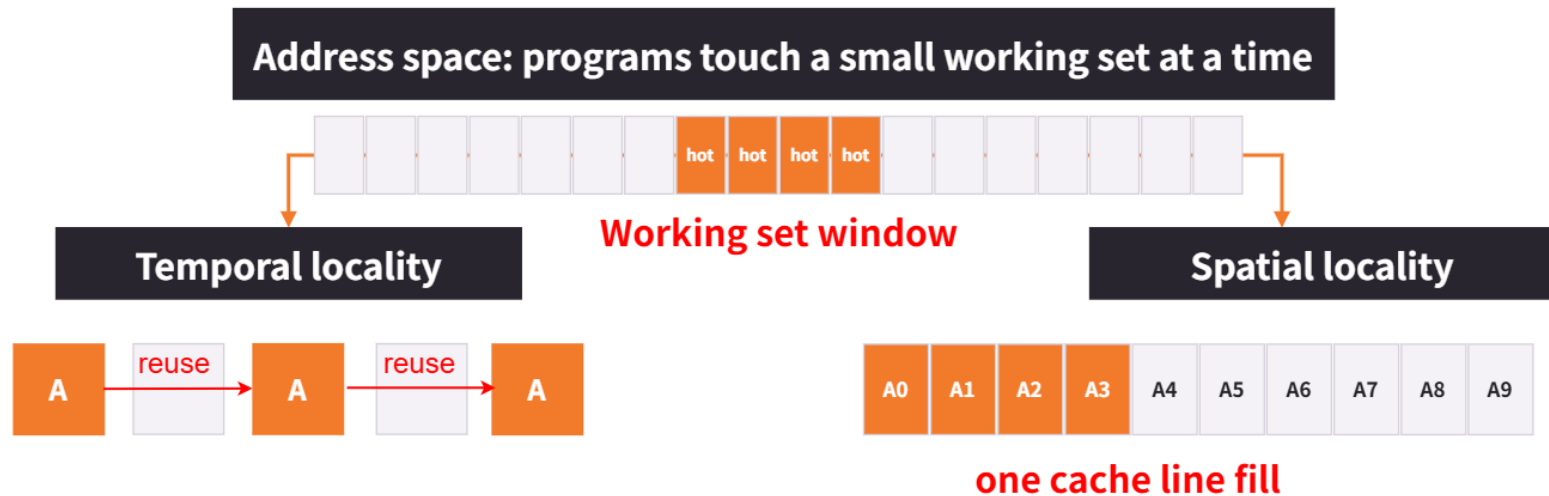
E.g., Loops

Spatial Locality

The neighbors of the current data access tend to be accessed in the future.

E.g., array

Locality: The Hot Feature Makes Caches Useful



Temporal Locality

A piece of data is quickly reused in the future.

E.g., Loops

Spatial Locality

The neighbors of the current data access tend to be accessed in the future.

E.g., array

Capture localities with high-speed caches to address memory wall.

Locality: The Hot Feature Makes Caches Useful

"D" reused every iteration

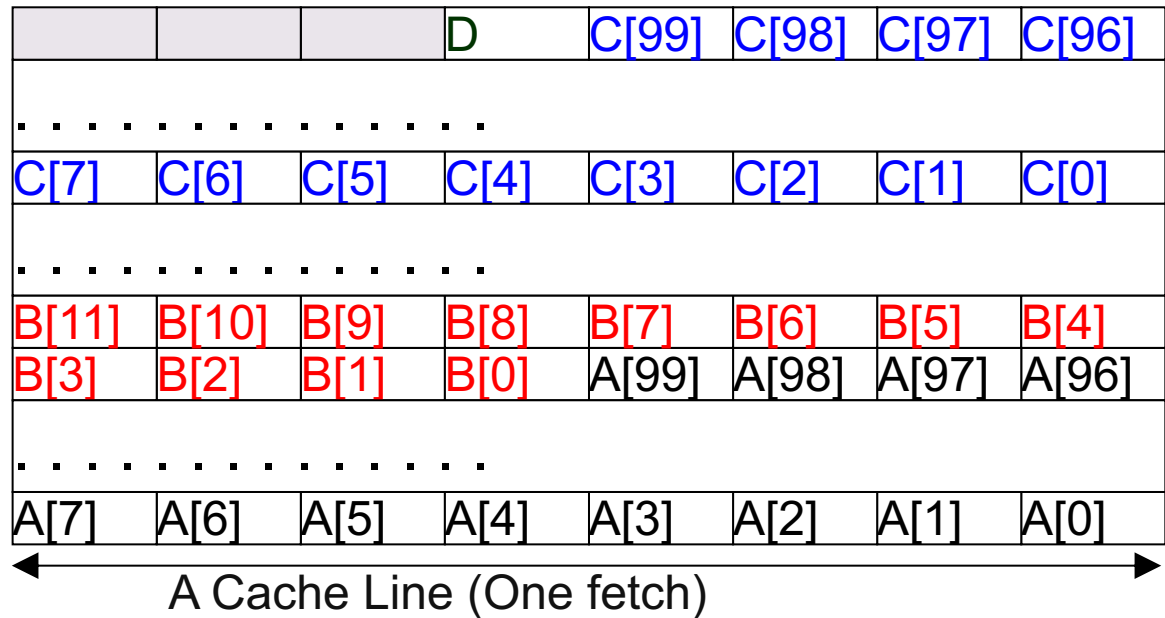
Temporal locality

"A", "B", "C" used in sequence

Spatial locality

Fetch once, cache a line

```
Int A[100], B[100], C[100], D;  
for (i = 0; i < 100; i++) {  
    C[i] = A[i] * B[i] + D  
}
```



Hit or Miss of Caches: Benefit vs. Penalty

Hit: The requested memory block is found in the current cache level, with a valid tag match.

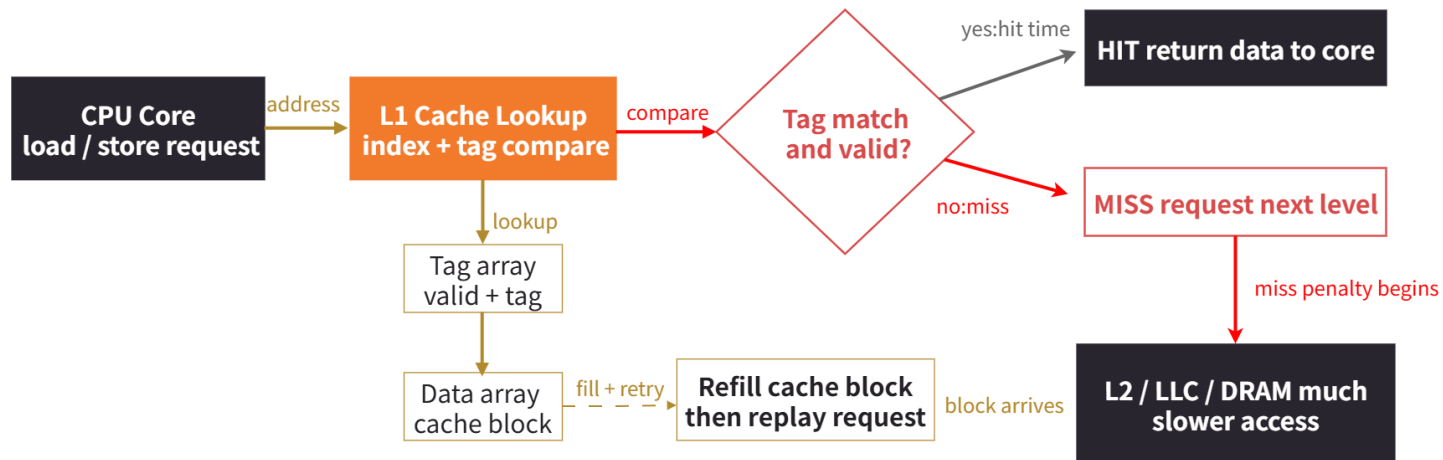
Hit Rate: Hits / Total Accesses

Cache Read/Write: Find the data on cache and then perform read/write

Miss: The requested memory block is not present in the current cache level, so it must be retrieved from the next lower level.

Miss Rate: 1 - Hit Rate

Miss Penalty: The additional time required to service a miss



Average Memory Access Time (AMAT)

$AMAT = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$

Miss Penalty: time to fetch a block from lower memory level

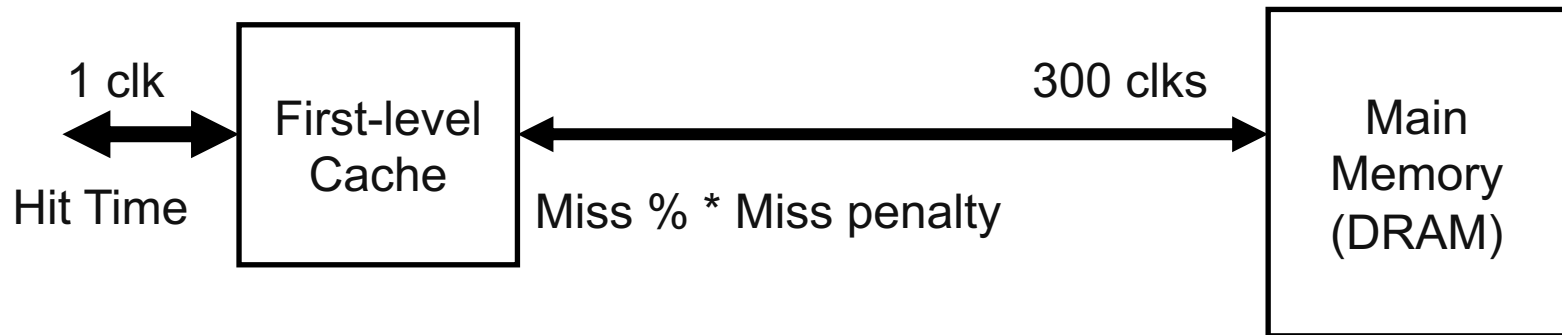
Access Time: Latency for the memory to handle one access

Transfer Time: Latency to transfer data block to higher level

$\text{Miss penalty} = \text{access time} + \text{data size} / \text{bandwidth}$

- A cache helps when common hits remain fast and sufficiently few accesses pay the miss penalty.

Memory Hierarchy Performance



Average Memory Access Time (AMAT)

$$= \text{Hit Time} + \text{Miss rate} * \text{Miss Penalty}$$

$$= T_{\text{hit}}(L1) + \text{Miss}\%(L1) * T(\text{memory})$$

Example:

Cache Hit = 1 cycle

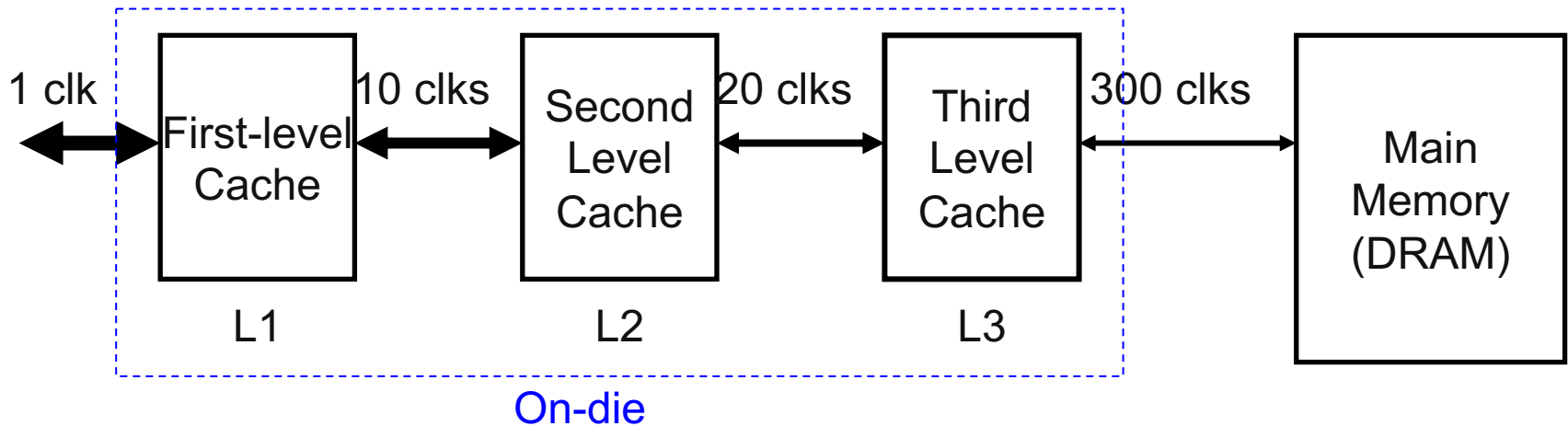
Miss rate = 10% = 0.1

Miss penalty = 300 cycles

$$\text{AMAT} = 1 + 0.1 * 300 = 31 \text{ cycles}$$

Can we improve it?

Reducing Penalty: Multi-Level Cache



Average Memory Access Time (AMAT)

$$\begin{aligned} &= T_{hit}(L1) + \text{Miss}\%(L1) * (T_{hit}(L2) \\ &+ \text{Miss}\%(L2) * (T_{hit}(L3) + \text{Miss}\%(L3) * T(\text{memory}))) \end{aligned}$$

AMAT = ?

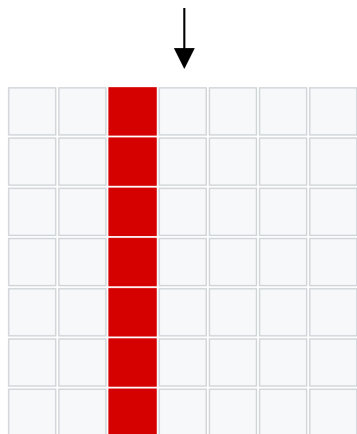
2.115 (compare to 31 with no multi-levels)

14.7x Speed Up

Reducing Penalty: Code Optimization

Poor locality: column-wise traversal

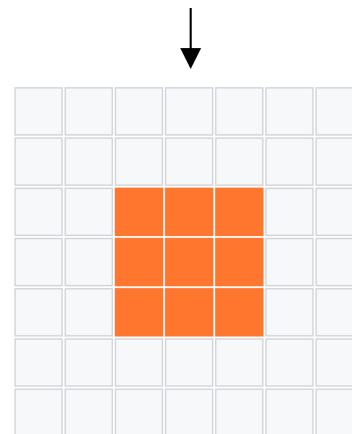
```
for (j = 0; j < N; j++)  
  for (i = 0; i < N; i++)  
    sum += A[i][j];
```



miss-heavy stream

Better locality: blocking / tiling

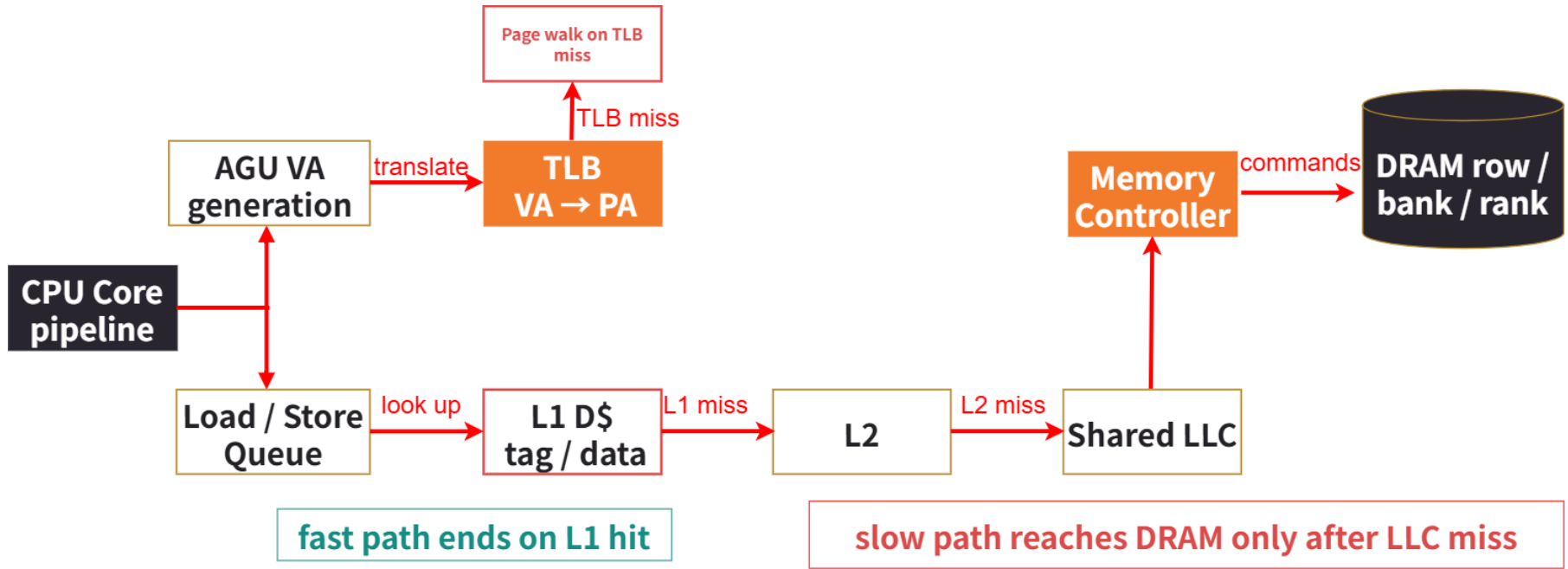
```
for (ii = 0; ii < N; ii += B)  
  for (jj = 0; jj < N; jj += B)  
    compute_tile(A[ii:ii+B][jj:jj+B])
```



cache-resident tile

transform

End-to-End Load/Store Path: Address Resolution – TLB – Cache Hierarchy – DRAM



- A TLB miss requires address translation work. A cache miss determines whether the data must come from a lower level.
- Only a miss through the cache hierarchy sends the data request to the memory controller and DRAM.

DRAM Behind Cache: SRAM vs. DRAM

Static Random-Access Memory (SRAM)

Write

Drive bit lines (bit=1,
bitbar=0)

Select row

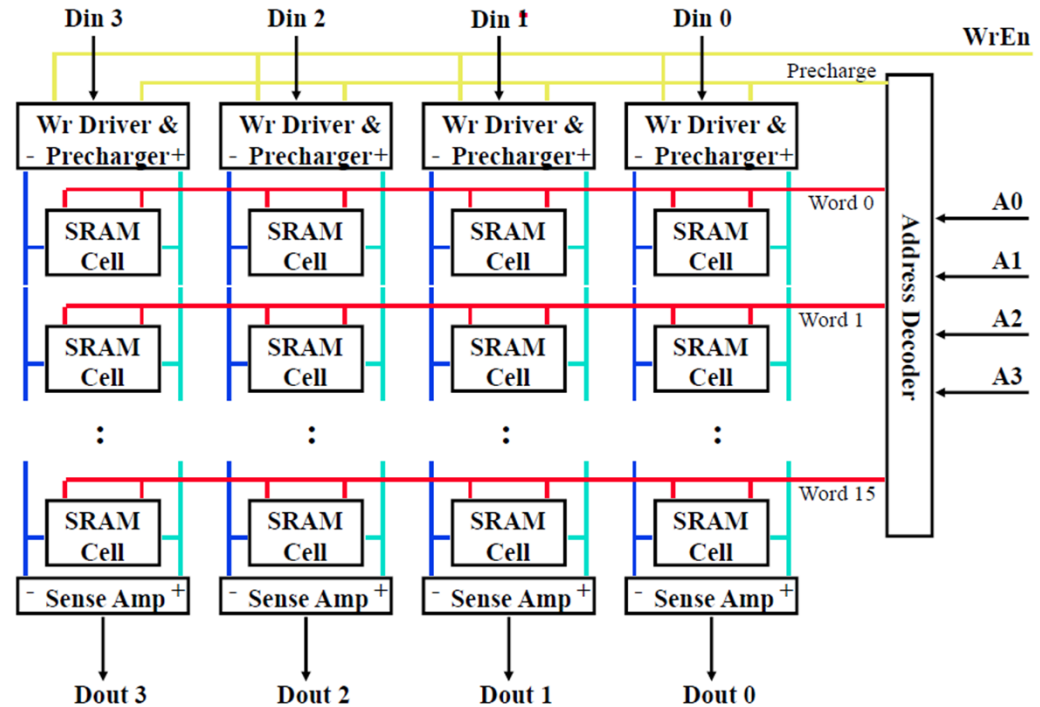
Read

Precharge bit and bitbar to
V_{dd}

Select row

Cell pulls one line row

Sense amp on column
detects difference between
bit and bitbar



Adapted from: Princeton COS375, "Memory,"
2015.

DRAM Behind Cache: SRAM vs. DRAM

SRAM --- Store data via stable transistor circuit

Requires **low power** to retain bit (423mV for 45nm)

Requires **6 transistors/bit** --- **expansive**

Usage: Caches, register files

DRAM --- Store data via tiny capacitors (bit is represented as charged and discharged)

One transistor/bit --- **cheap**

Must be **re-written after being read**

Must also be **periodically refreshed**

Each row can be refreshed simultaneously

Address lines are multiplexed:

Upper half of address: row access strobe (RAS)

Lower half of address: column access strobe (CAS)

Usage: DDR / GDDR / HBM

DRAM Behind Cache: SRAM vs. DRAM

Dynamic Random-Access

Memory (DRAM)

Write

Drive bit lines

Select row

Read

Precharge bit line to Vdd

Select row

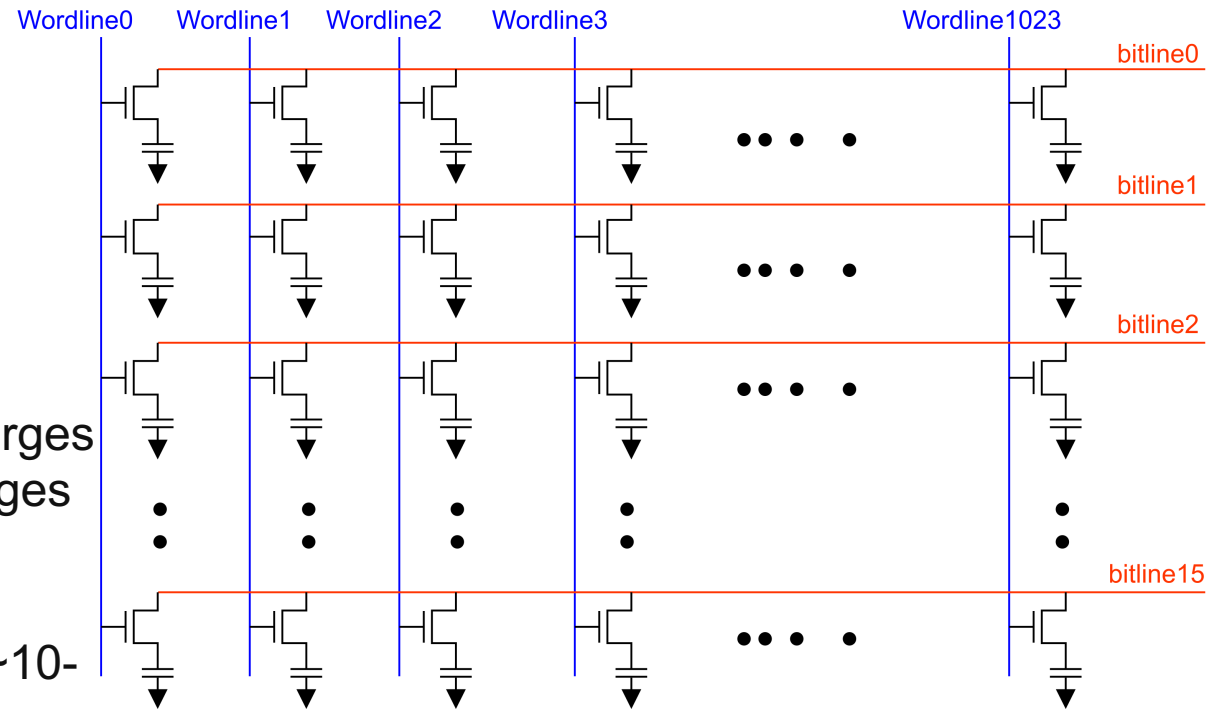
Cell and bit line share charges

Very small voltage changes
on the bit line

Sense (fancy sense amp)

Can detect changes of ~10-
100k electrons

Write: restore the value



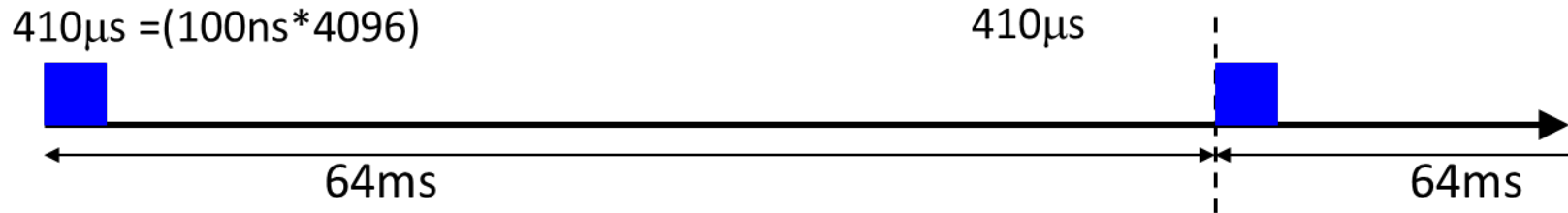
DRAM Behind Cache: DRAM Refresh

Leaky storage
Periodic refresh across DRAM rows
Un-accessible when refreshing
Read, and write the same data back

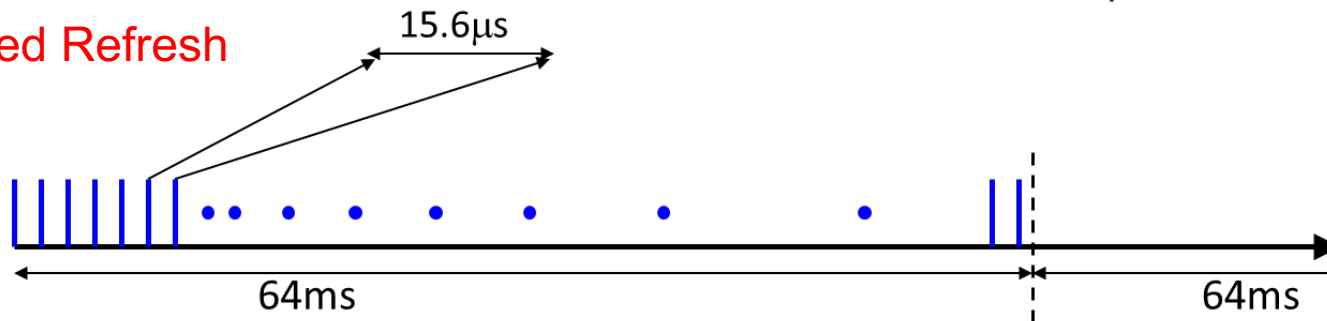
4K rows in a DRAM
100ns read cycle
Each row refreshed
in 64ms

$4096 * 100\text{ns} = 410\mu\text{s}$ to
refresh once
 $410\mu\text{s} / 64\text{ms} = 0.64\%$
unavailability

Bursty Refresh



Distributed Refresh



DRAM Behind Cache: Why DRAM is Slow?

Weak storage signal

DRAM stores bits as tiny capacitor charges, producing only a small voltage difference on long bitlines.

Sensing takes time

The sense amplifier must detect and amplify a very small voltage change into a full 0/1 value.

Destructive read

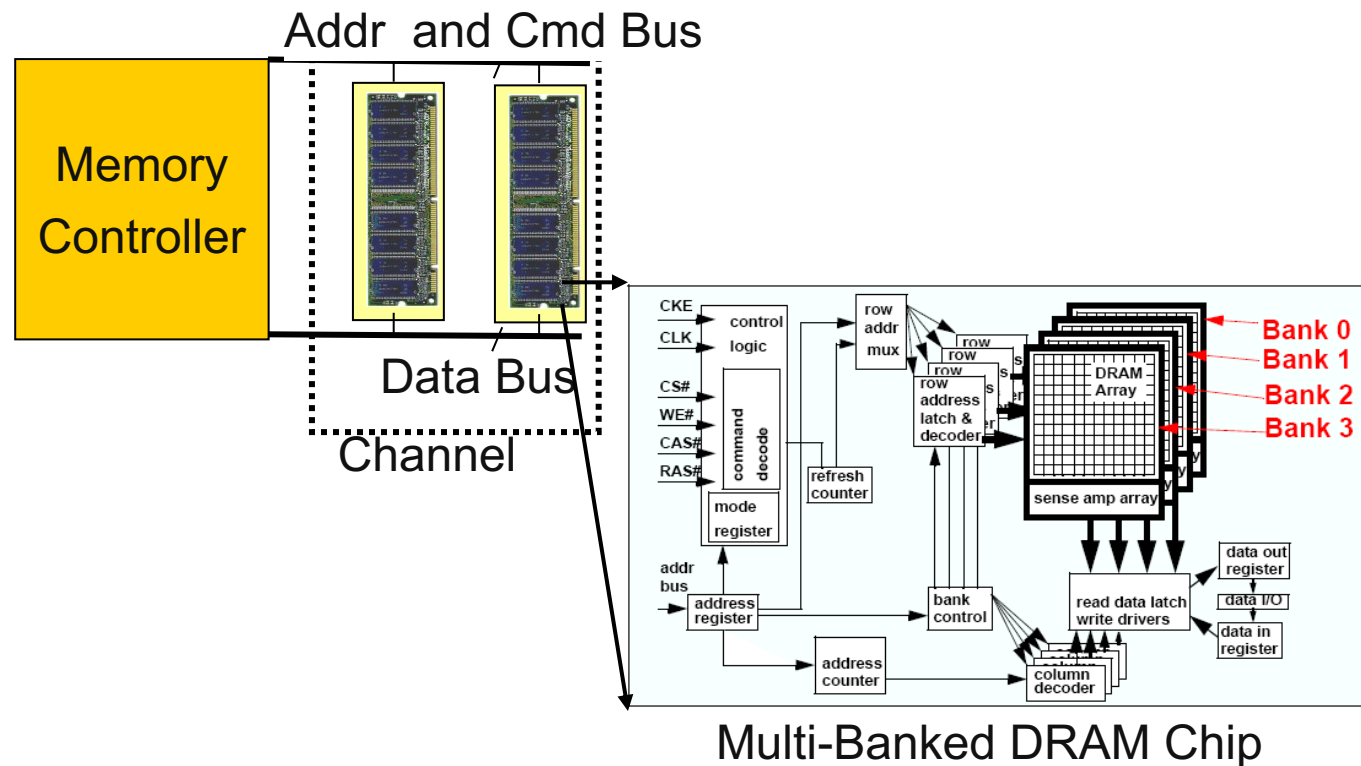
Reading a DRAM cell disturbs its stored charge, so the data must be restored after every row activation.

Large dense arrays

DRAM prioritizes capacity and cost per bit, so rows/bitlines are long and have higher RC delay.

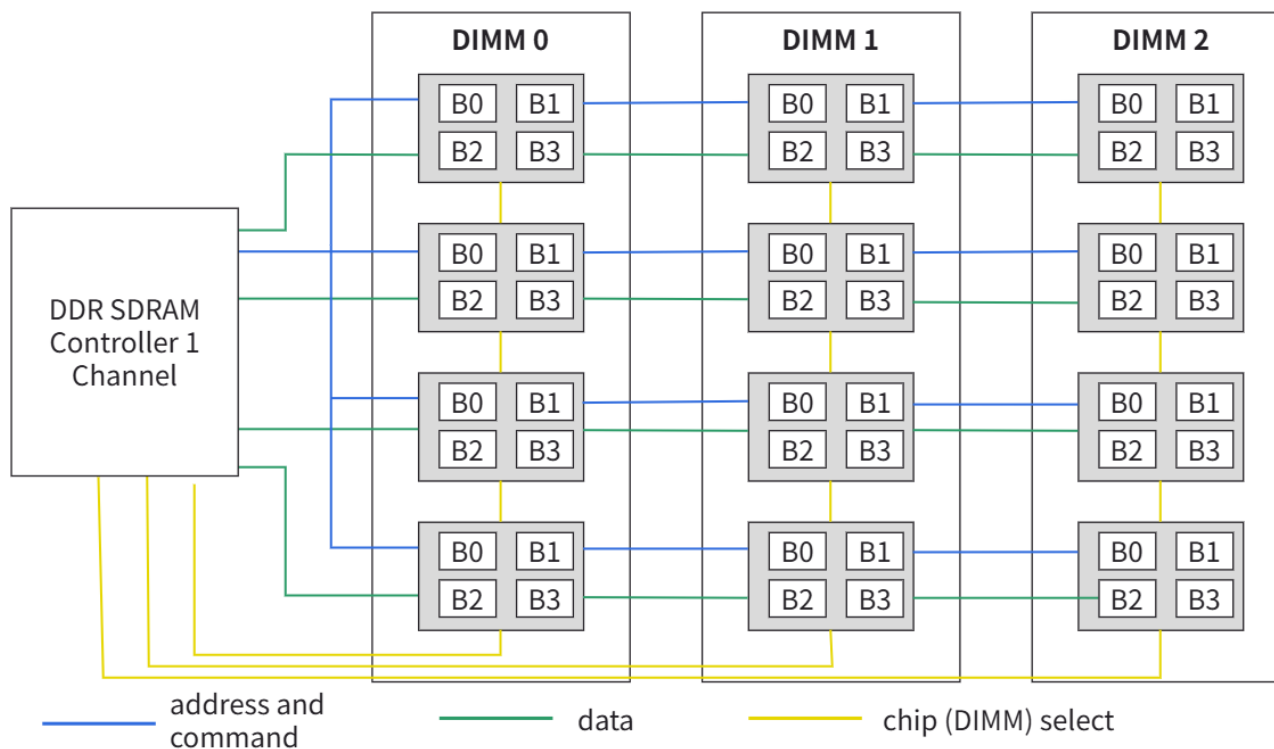
Solution: Multi-Bank, Multi-Rank DRAM

DRAM Behind Cache: Multi-Bank DRAM



- Banks provide independent storage arrays. Work in one bank can overlap preparation in another.
- The shared data bus still limits how much data the channel can return at once.

DRAM Behind Cache: Multi-Bank, Multi-Rank DRAM --- DIMM



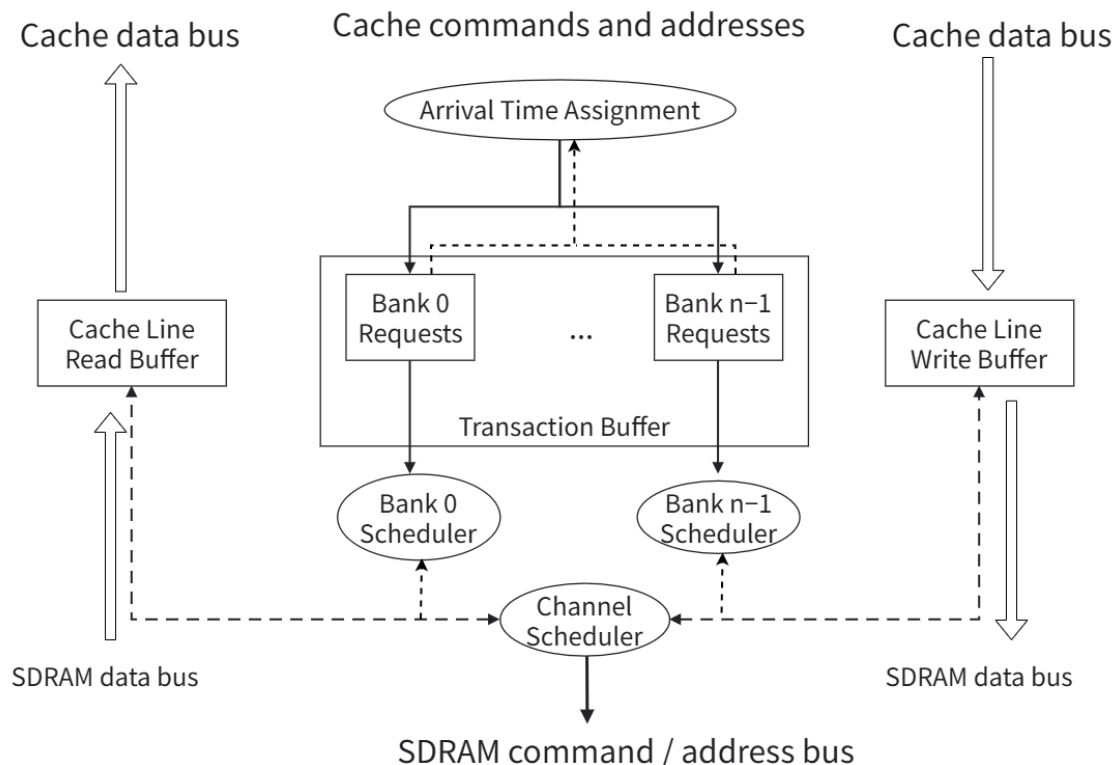
- A rank combines DRAM chips into the channel data width. Multiple ranks share the channel.
- The controller selects a rank and schedules requests across its banks.

Buffering cache line in both directions

Cache line write buffer

Scheduling requests in two-level

Bank and channel

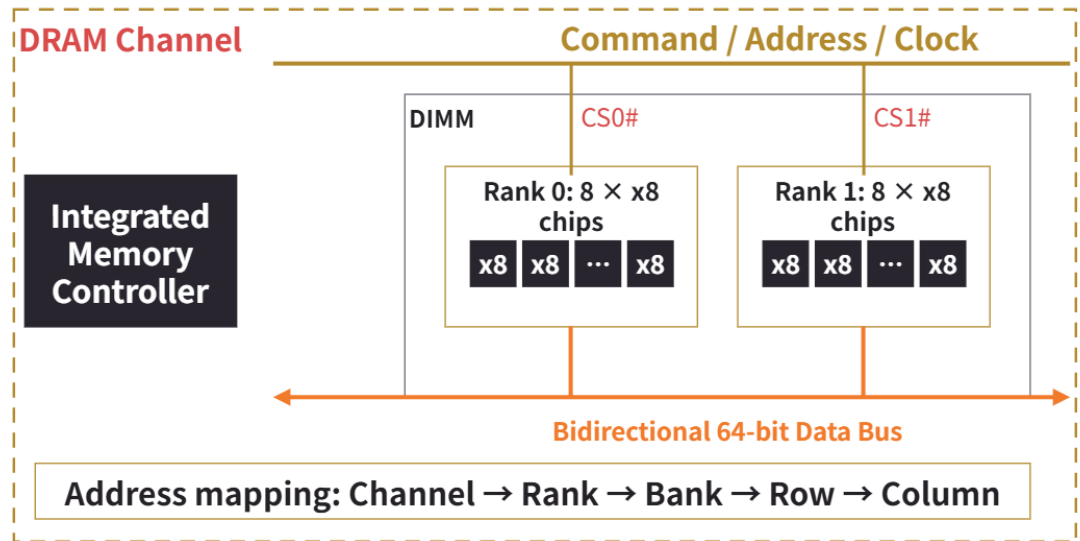


DRAM Behind Cache: Channel

One memory-controller port drives one channel

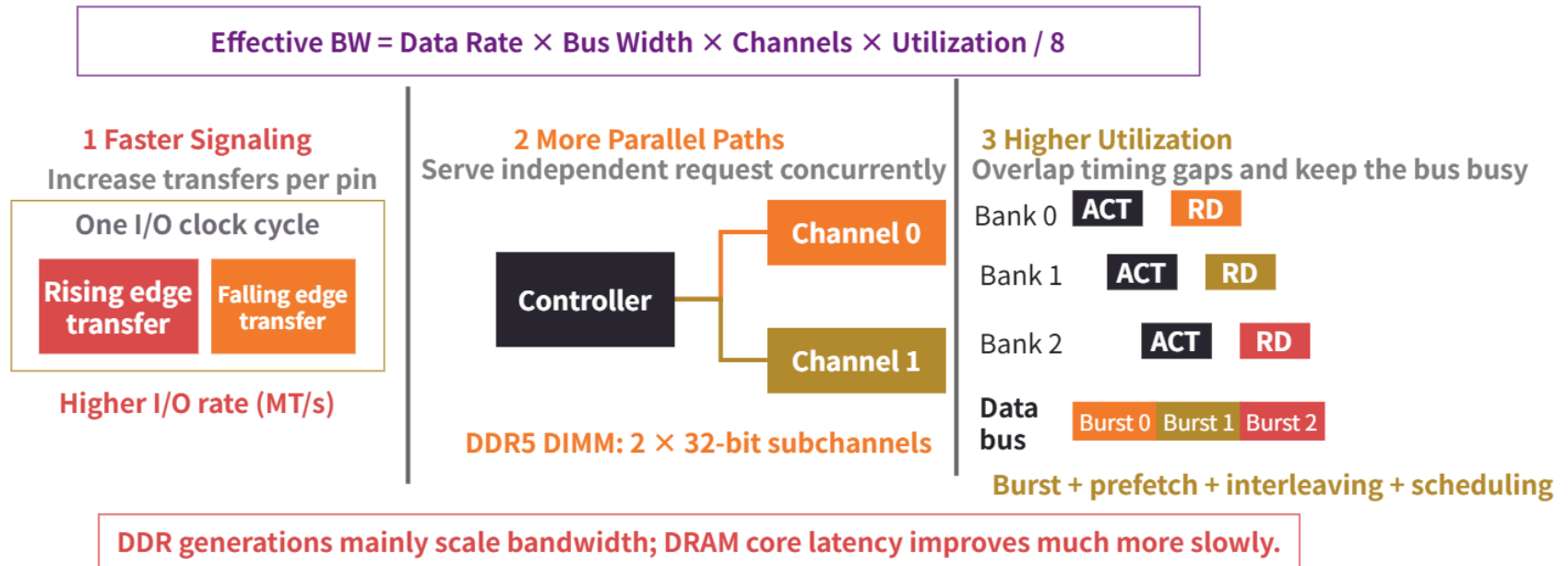
Command, address, and clock signals select the destination

All ranks share the data bus; only the selected rank may drive it



$$\text{Peak Bandwidth} = \text{Transfer Rate} \times \text{Bus Width} / 8$$

DRAM Behind Cache: Channel Bandwidth Grows Along Three Axes



- Independent channels add bandwidth when requests use them concurrently. Bank scheduling improves bus utilization.
- DDR5 splits 64 data bits into two 32-bit subchannels. This split alone does not double total data width.

DRAM Behind Cache: From DDR1 to DDR5

Generation	Data Rate	Prefetch / Burst	Bank Parallelism	DIMM Data Path	VDD	Peak BW (x64)
DDR	200-400 MT/s	2n / BL2-8	4 banks	1 x 64-bit	2.5 V	3.2 GB/s
DDR2	400-1066 MT/s	4n / BL4-8	4-8 banks	1 x 64-bit	1.8 V	8.5 GB/s
DDR3	800-2133 MT/s	8n / BL8	8 banks	1 x 64-bit	1.5 V	17.1 GB/s
DDR4	1600-3200 MT/s	8n / BL8	16 banks / 4 BG	1 x 64-bit	1.2 V	25.6 GB/s
DDR5	3200-6400 MT/s	16n / BL16	Up to 32 banks / 8 BG	2 x 32-bit	1.1 V	51.2 GB/s

Pin rate: 16x

Peak x64 BW: 16x

Voltage: 2.5 V -> 1.1 V

The interface evolves faster than the DRAM cell: bandwidth rises through faster I/O and more parallelism.

- At the selected upper rates, the same aggregate x64 width rises from 3.2 to 51.2 GB/s.
- Faster interfaces, longer bursts, and more bank parallelism raise throughput; cell access improves more slowly.

GPU Memory Hierarchy

Lecture 2: Memory Evolution — Memory Hierarchy and SOTA

CUDA memory spaces and access patterns
GDDR and HBM

Larger Capacity, Longer Access Time

Registers

Allocated per thread, hardware managed

Shared Memory + L1 Cache

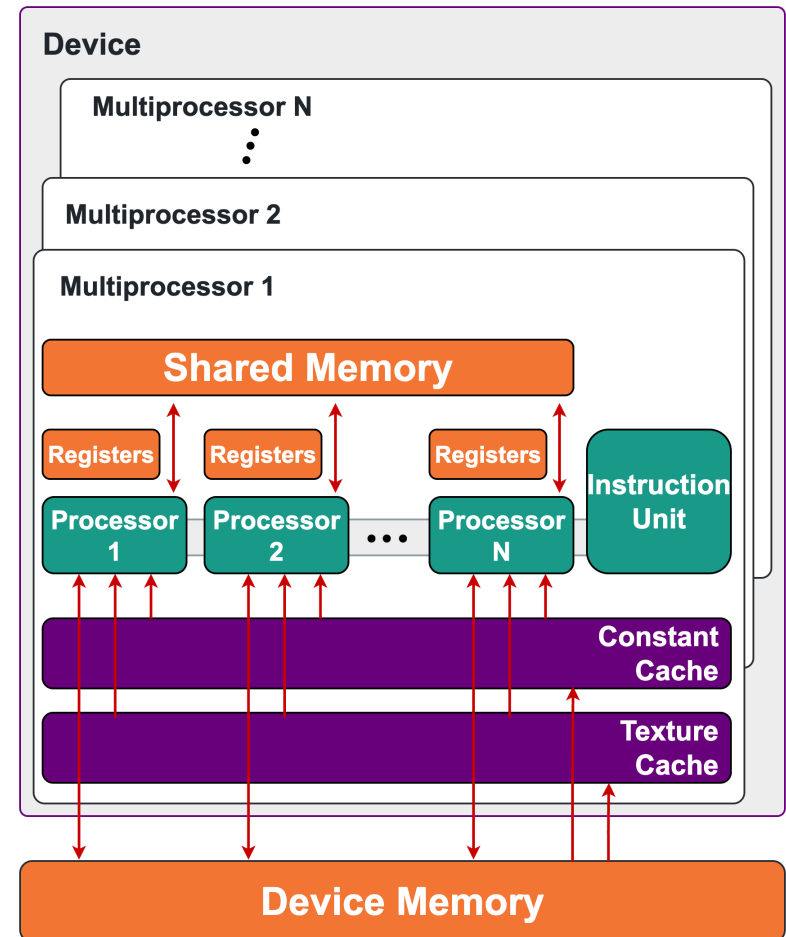
Per SM explicit scratchpad, hardware cache

L2 Cache

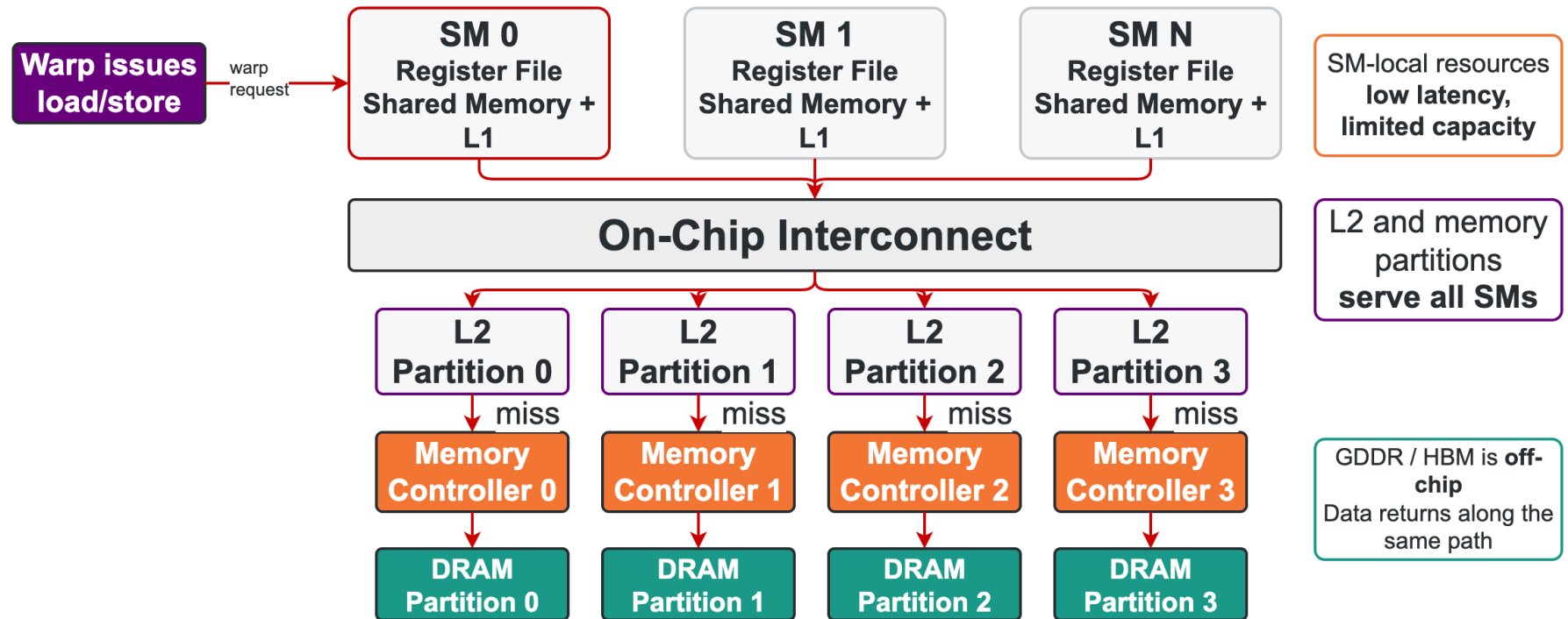
Device-wide, shared by all SMs

Device Memory: GDDR / HBM

Off-chip physical memory serving the global address space

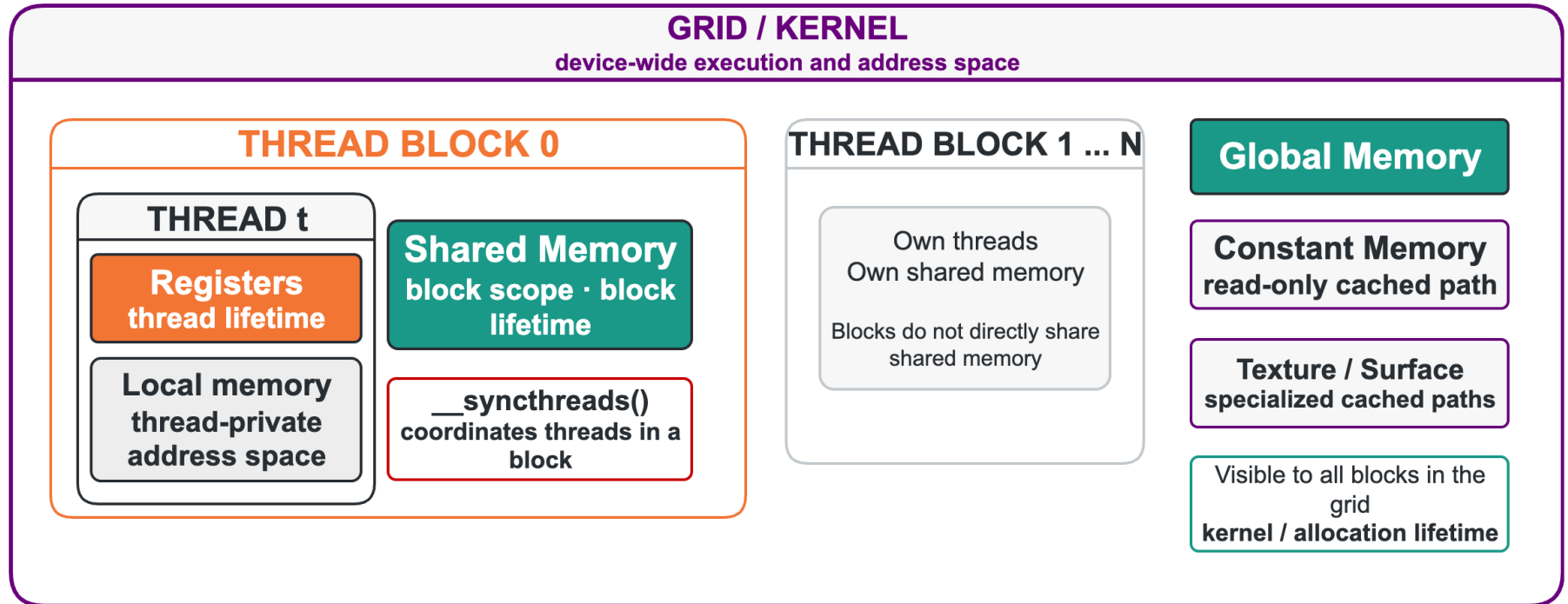


A Memory Access Traverses SM-Local Storage, L2, and Device DRAM



- A request reaches device-wide L2 through the on-chip interconnect. An L2 miss continues to a memory controller.
- All SMs share L2 and the DRAM partitions, implemented with GDDR devices or HBM channels.

Programmer's View of CUDA Memories



- Scope identifies which threads can access data. Lifetime identifies how long the data exists.
- `__syncthreads()` coordinates threads within one block; it does not synchronize different blocks.

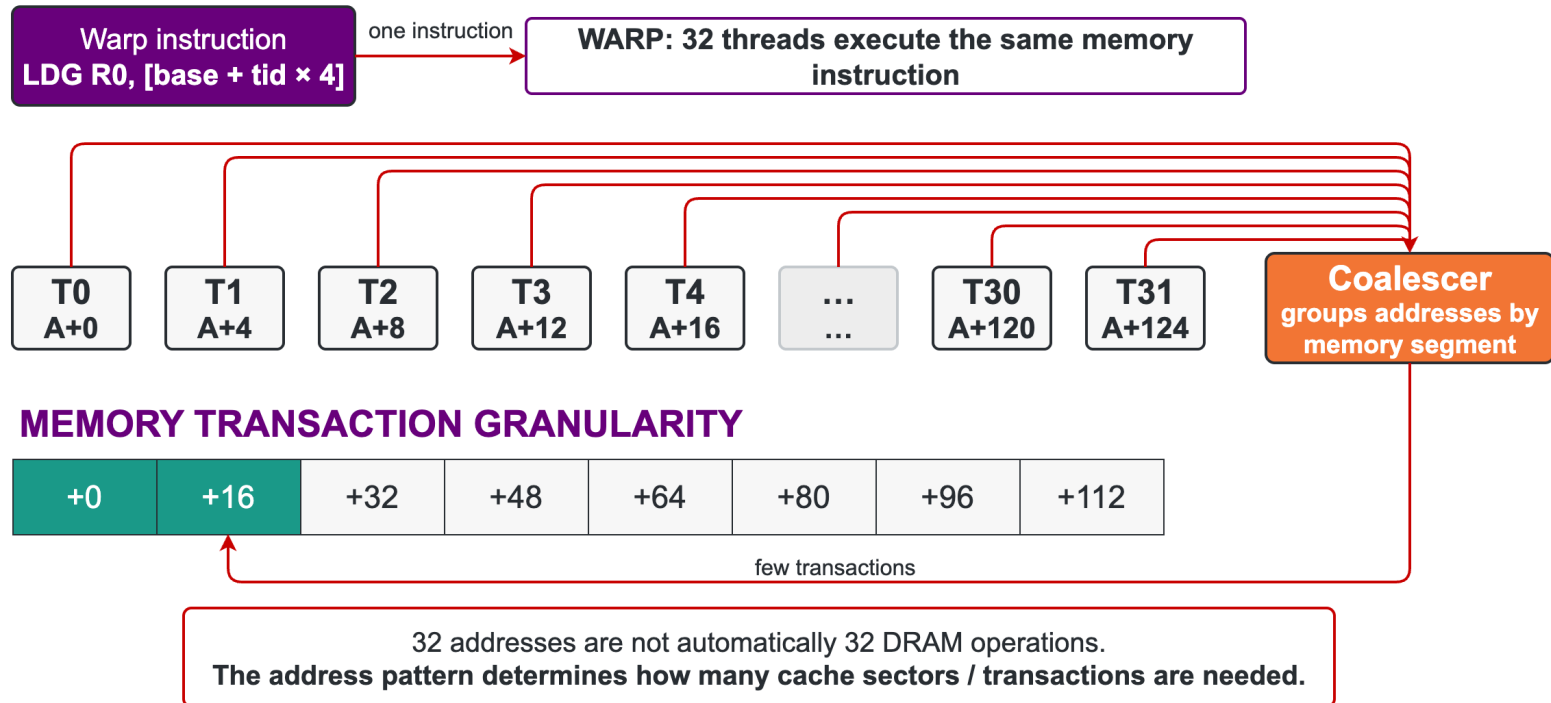
Every CUDA Memory Space Maps to a Physical Resource

Memory Space	Physical Location	Cached?	Scope	Lifetime	Access	Relative Latency	Performance Note
Registers	On-chip register file	—	Thread	Thread	R/W	Lowest	Usage can limit occupancy
Local	Device DRAM	L1 / L2	Thread	Thread	R/W	High	Spills and large private arrays
Shared	On-chip per SM	—	Block	Block	R/W	Low	Capacity can limit occupancy
Global	Device DRAM	L1 / L2	Grid / host	Allocation	R/W	High	Coalescing determines traffic
Constant	Device DRAM	Constant cache	Grid	Allocation	Read-only	Low on hit	Best for warp-uniform reads
Texture	Device DRAM	Texture cache	Grid	Allocation	Read / surface write	Low on hit	Optimized for spatial locality

Key distinction: **local** is private in scope, but usually **off-chip** in physical location.

- Local memory is thread-private in scope but usually resides in device DRAM and passes through L1/L2.
- Register and shared-memory allocations limit how many blocks and warps can reside on an SM.

Access Memory: One Warp Memory Instruction Generates Up to 32 Addresses

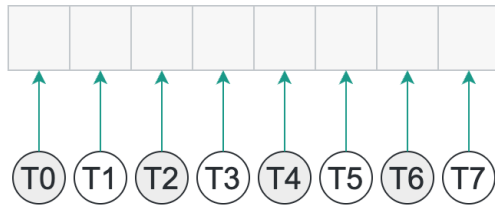


- The coalescer groups addresses by cache-sector or transaction granularity before DRAM access.
- The transaction count depends on alignment, address spacing, access width, and GPU architecture.

Access Memory: Coalescing Packs Nearby Addresses into Fewer Transactions

ALIGNED + CONTIGUOUS

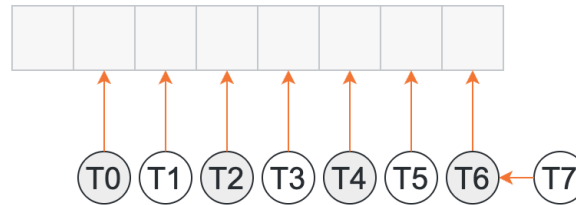
Minimum transactions



Transactions: **1–few**

MISALIGNED

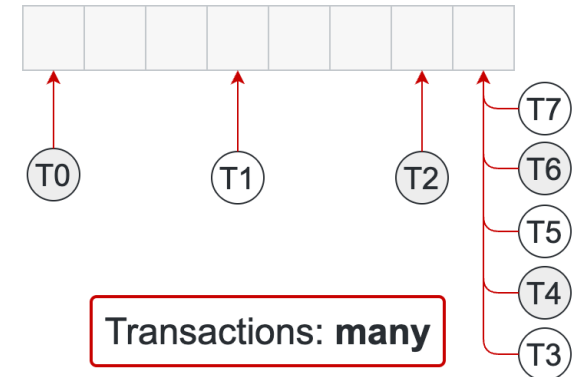
Crosses an extra boundary



Transactions: **few + 1**

STRIDED

Touches many segments



Transactions: **many**

Coalescing is evaluated per warp instruction and depends on access width, alignment, cache-sector size, and architecture.

- Coalescing is evaluated for one warp memory instruction. Aligned, contiguous addresses touch fewer segments.
- Misalignment can cross extra boundaries. Strided accesses can touch many more segments.

Access Memory: Data Layout Determines Useful Memory Bandwidth

ROW-MAJOR MATRIX

A[0][0]	A[0][1]	A[0][2]	A[0][3]	A[0][4]	A[0][5]	A[0][6]	A[0][7]
A[1][0]	A[1][1]	A[1][2]	A[1][3]	A[1][4]	A[1][5]	A[1][6]	A[1][7]
A[2][0]	A[2][1]	A[2][2]	A[2][3]	A[2][4]	A[2][5]	A[2][6]	A[2][7]
A[3][0]	A[3][1]	A[3][2]	A[3][3]	A[3][4]	A[3][5]	A[3][6]	A[3][7]
A[4][0]	A[4][1]	A[4][2]	A[4][3]	A[4][4]	A[4][5]	A[4][6]	A[4][7]
A[5][0]	A[5][1]	A[5][2]	A[5][3]	A[5][4]	A[5][5]	A[5][6]	A[5][7]
A[6][0]	A[6][1]	A[6][2]	A[6][3]	A[6][4]	A[6][5]	A[6][6]	A[6][7]
A[7][0]	A[7][1]	A[7][2]	A[7][3]	A[7][4]	A[7][5]	A[7][6]	A[7][7]

Warp walks across a row
contiguous addresses

Warp walks down a column
strided addresses

Same arithmetic, different layout → different
memory traffic.

$$\text{Useful bandwidth} = \text{Physical bandwidth} \times \frac{\text{Requested bytes}}{\text{Transferred bytes}}$$

COALESCED

Requested bytes

Transferred bytes

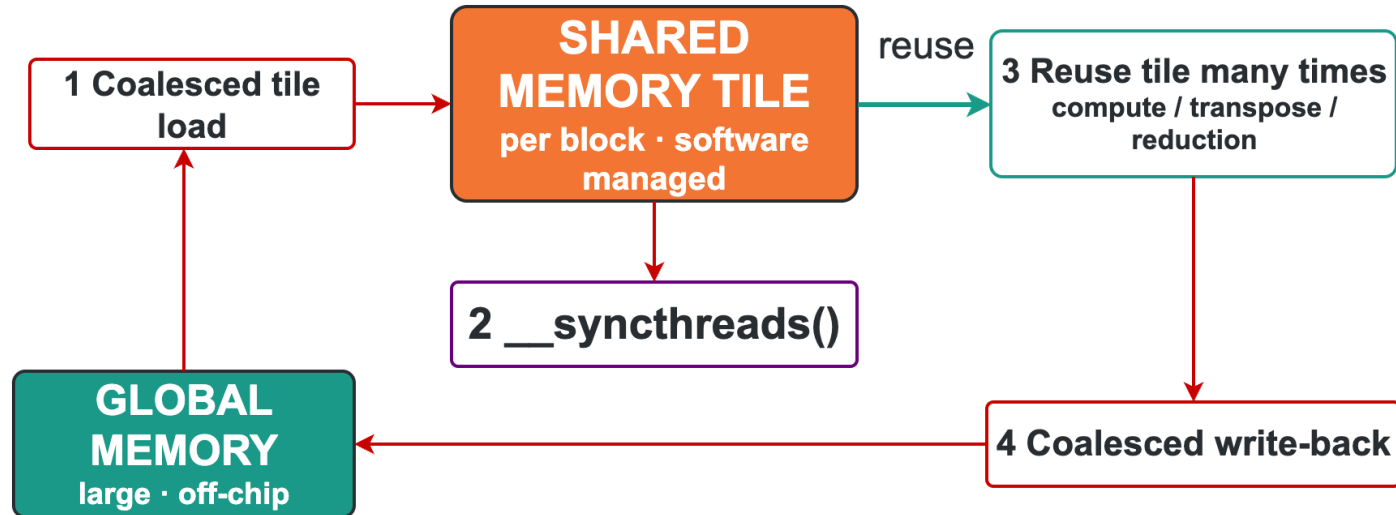
STRIDED

Requested bytes

Transferred bytes

- Row-wise and column-wise traversal can perform the same arithmetic while transferring different byte counts.
- Useful bandwidth depends on requested bytes relative to the bytes actually transferred.

Access Memory: Shared Memory Creates an Explicit On-Chip Data Stage



Shared memory is an explicitly addressed scratchpad.
It does not automatically cache arbitrary global-memory accesses.

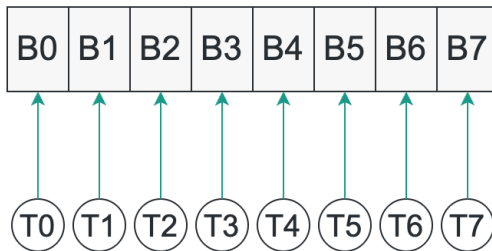
- Software explicitly chooses the data to stage in shared memory and when threads must synchronize.
- One off-chip load can support repeated on-chip reuse, reducing DRAM traffic and increasing arithmetic intensity.

Access Memory: Bank Conflicts Serialize Otherwise Parallel Accesses

Typical mapping for 32-bit words: $\text{bank} = (\text{address} / 4) \bmod 32$

CONFLICT-FREE

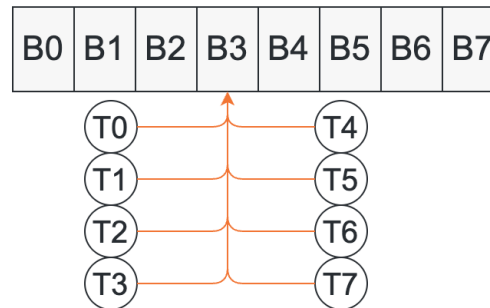
Different banks → parallel



1 bank access each

BROADCAST

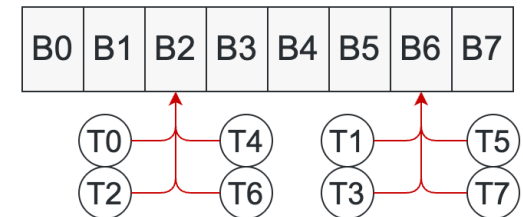
Same address → one value broadcast



1 address, broadcast

BANK CONFLICT

Same bank, different addresses → serialized



multiple serialized rounds

The real GPU usually exposes 32 banks; eight are shown here for readability.

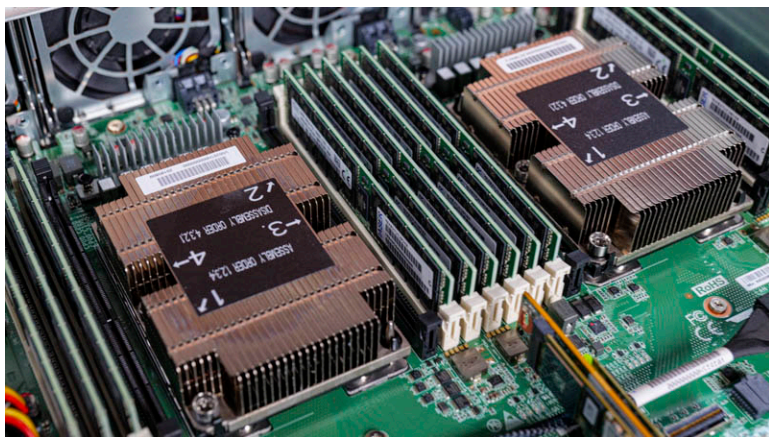
- Typical GPUs use 32 banks; the figure shows 8 for readability. Same-address reads can use broadcast.
- Different addresses in the same bank require multiple service rounds. This differs from global-memory coalescing.

From DDR DIMMs to GDDR

Socketed DIMM

Multiple ranks + replaceable modules

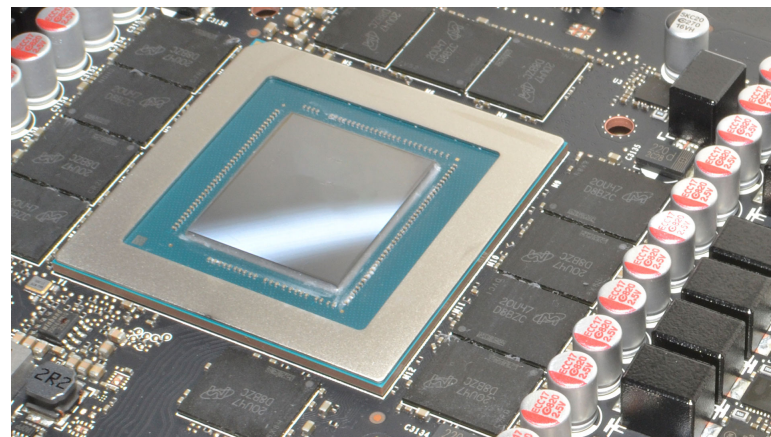
Longer routed channel



GDDR

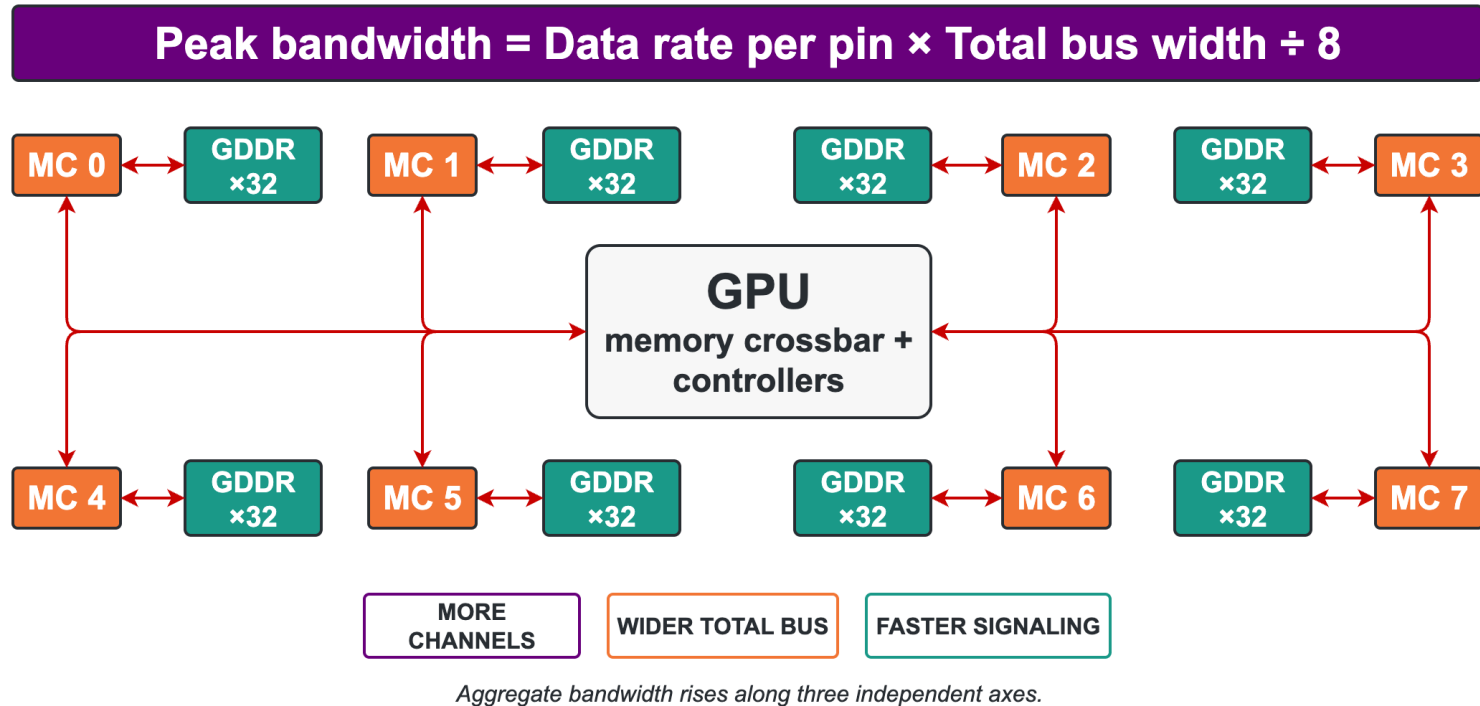
Soldered devices

Short point-to-point links



- Short, fixed point-to-point routes permit higher per-pin data rates, with less expandability than socketed DIMMs.

From DDR DIMMs to GDDR: More Channels and Faster Signaling



- Aggregate bandwidth is the sum across independent memory-controller channels.
- More channels, wider buses, and faster signaling also raise pin count, routing complexity, and I/O power.

From DDR DIMMs to GDDR: GDDR5 to GDDR7

Increasing pin rate and signaling complexity

GDDR5	GDDR5X	GDDR6	GDDR6X	GDDR7
Signaling NRZ	Signaling NRZ	Signaling NRZ	Signaling PAM4	Signaling PAM3
Representative pin rate up to ~8 Gb/s	Representative pin rate ~10–14 Gb/s	Representative pin rate 14–20 Gb/s	Representative pin rate vendor-specific	Representative pin rate up to 32 Gb/s
Single ×32 interface Baseline high-speed graphics DRAM	Higher prefetch / clocking Extends the same signaling approach	Two ×16 channels More parallelism per package	Micron / NVIDIA extension More bits per symbol; tighter margins	Four channels per ×32 device Higher rate with better margin than PAM4

GDDR6X is a vendor extension, not the next JEDEC generation. Representative rates vary by product and speed bin.

- GDDR6X uses PAM4 as a vendor extension. GDDR7 uses PAM3 and further divides the channels.
- The rates shown are representative. Actual rates depend on the product and speed bin.

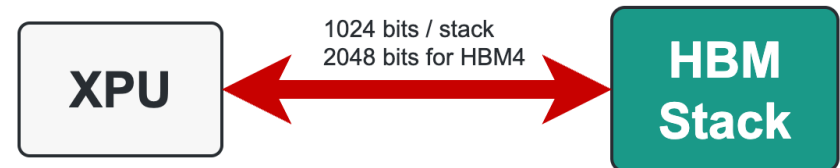
From GDDR to HBM: Thousands of Wires for HBMs

$$\text{Bandwidth} = \text{Per-pin data rate} \times \text{Interface width}$$

GDDR: NARROWER + FASTER



HBM: ULTRA-WIDE + MODERATE-RATE



20–32 Gb/s per pin
PCB point-to-point links
Signal integrity and I/O power dominate

Thousands of short package wires
Lower signaling rate per wire
Bandwidth/package and bandwidth/watt dominate

- HBM1–HBM3 use a 1024-bit interface per stack. HBM4 expands the interface to 2048 bits.
- Dense, short wiring improves bandwidth and energy per bit, while requiring advanced packaging.

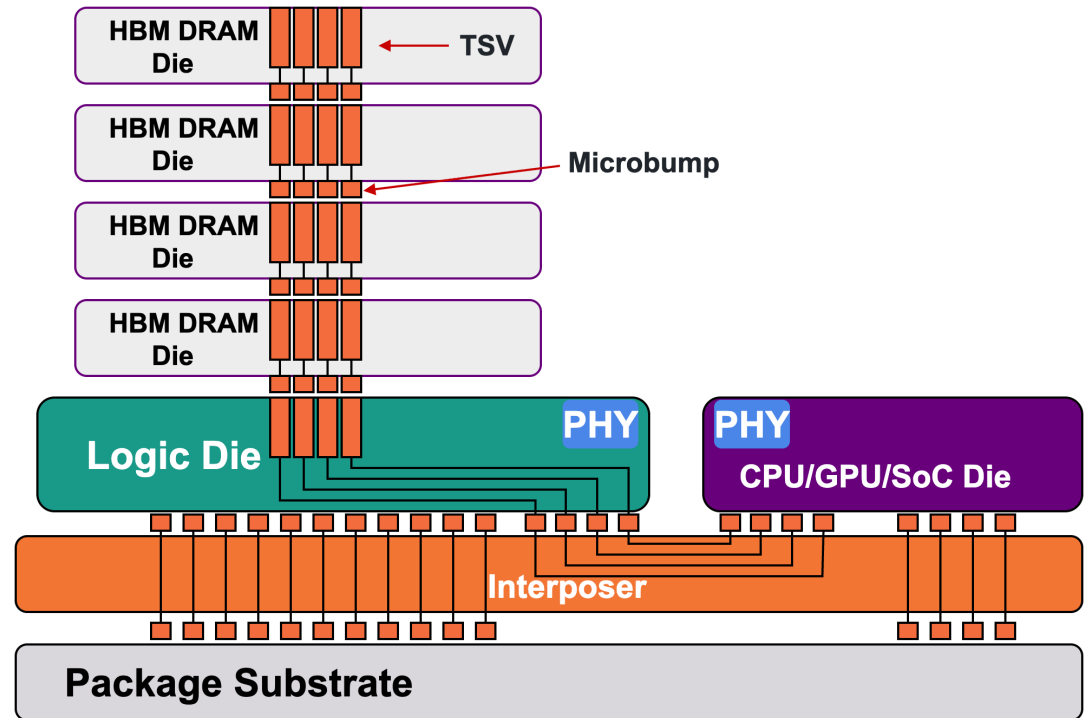
From GDDR to HBM: HBM Overview

Memory Stack in One Package

Micro-bumps connect adjacent dies

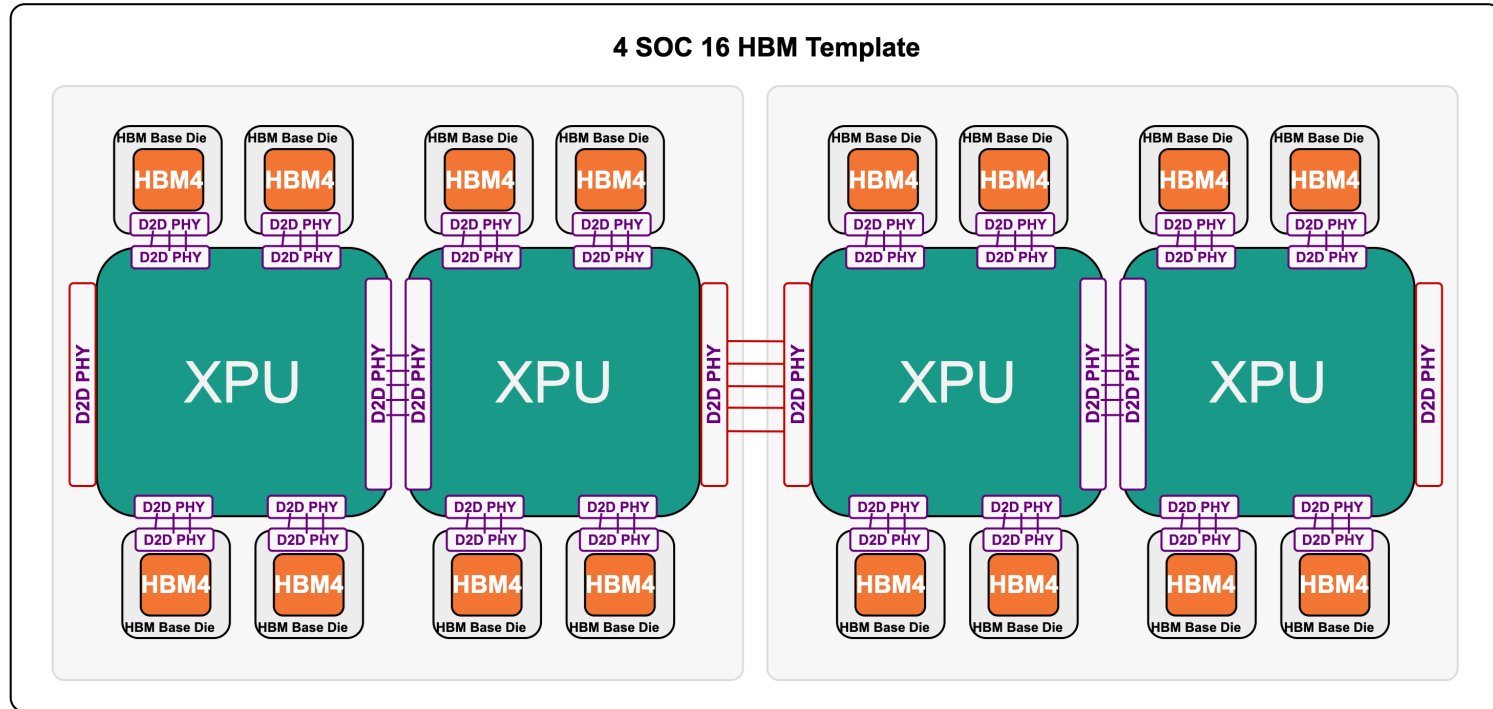
More stacked dies increase capacity

Wide channel organization increases bandwidth



- TSVs carry signals and power through the stack. Micro-bumps connect adjacent dies.

From GDDR to HBM: 2.5D Packaging



- HBM commonly sits beside the XPU on an interposer, forming a 2.5D package.
- Dense connections improve bandwidth and energy per bit, with costs in assembly, yield, cooling, and power delivery.

From GDDR to HBM: HBM1 to HBM4

Per-stack bandwidth grows through faster signaling, taller stacks, and finally a wider interface.

HBM	HBM2	HBM2E	HBM3	HBM3E	HBM4
I/O width 1024-bit	I/O width 1024-bit	I/O width 1024-bit	I/O width 1024-bit	I/O width 1024-bit	I/O width 2048-bit
Stack height 4-high	Stack height 4 / 8-high	Stack height 8-high	Stack height 8 / 12-high	Stack height 8 / 12-high	Stack height 12 / 16-high
Representative BW ~128 GB/s	Representative BW ~256 GB/s	Representative BW ~410 GB/s	Representative BW ~819 GB/s	Representative BW ~1.2 TB/s	Representative BW >2 TB/s
First wide-I/O stack	Higher capacity + speed	Faster HBM2	More channels + rate	Higher speed bins	2× I/O + advanced base die

Representative per-stack bandwidth; JEDEC limits and shipping products vary by vendor and speed bin.

- HBM scales through faster signaling, greater stack height, and the wider HBM4 interface.
- Per-stack figures are representative. Package space, thermal limits, and yield constrain stack count and height.

DDR, GDDR, and HBM Optimize Different Constraints

Design Dimension	DDR	GDDR	HBM
Primary target	CPU main memory	Graphics / edge AI	Datacenter AI / HPC
Topology	Shared channel + DIMMs	Point-to-point soldered devices	Stacked DRAM beside XPU
Interface strategy	Moderate width and rate	High per-pin rate	Ultra-wide interface
Packaging	Socketed module + PCB	BGA devices on PCB	TSV stack + interposer
Capacity scaling	More / denser DIMMs	More or denser devices	More stacks and taller stacks
Bandwidth / energy	Moderate / moderate	High / higher I/O power	Highest / best bandwidth per watt
Cost	Lowest system cost per GB	Middle	Highest packaging cost
Typical products	Servers and PCs	Gaming GPUs, edge accelerators	H100/B200-class GPUs, AI ASICs

DDR optimizes **capacity and expandability** · GDDR optimizes **pin bandwidth and cost** · HBM optimizes **bandwidth/package and bandwidth/watt**

- DDR favors capacity, expandability, and cost per GB. GDDR balances pin bandwidth and board cost.
- HBM favors bandwidth per package and per watt. The choice depends on the workload and system constraints.

SOTA Memory Technologies

Lecture 2: Memory Evolution — Memory Hierarchy and SOTA

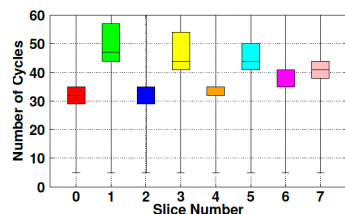
Shared caches and coherence
CXL, LPDDR, and memory placement

Multicore Technologies: Shared LLC

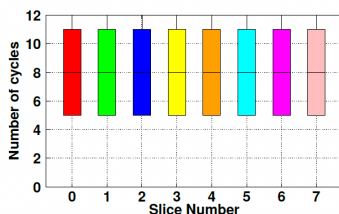
**A banked LLC is
logically shared but
physically distributed**

Physical-address hash
selects a home LLC slice

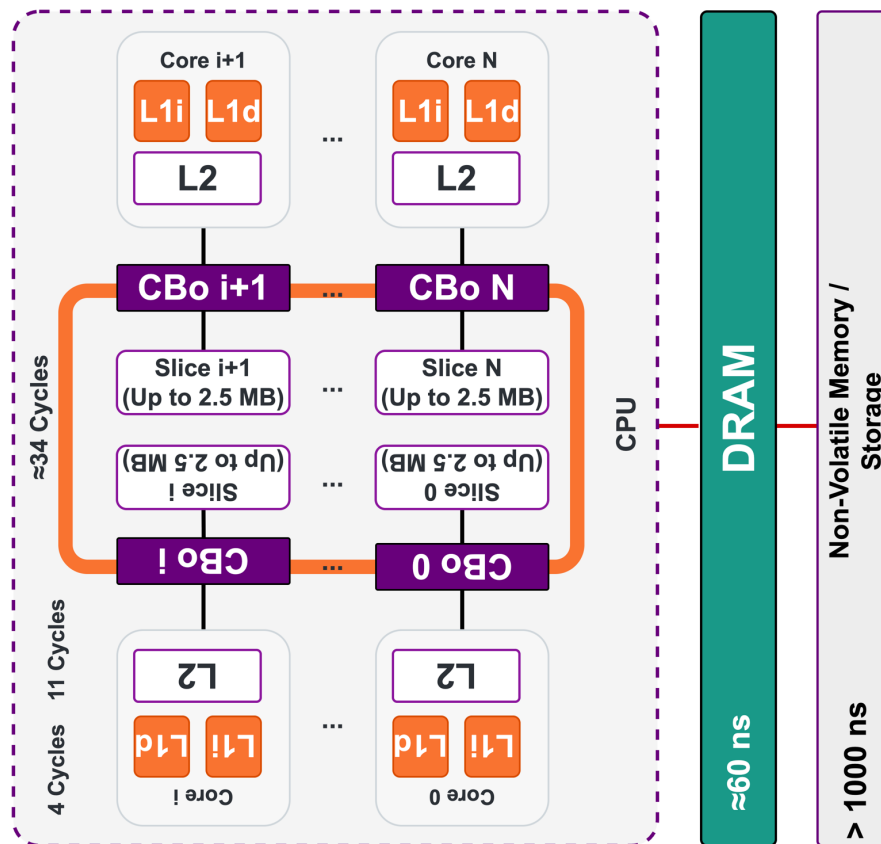
Read/Write to remote slices
would slow down memory
access speed



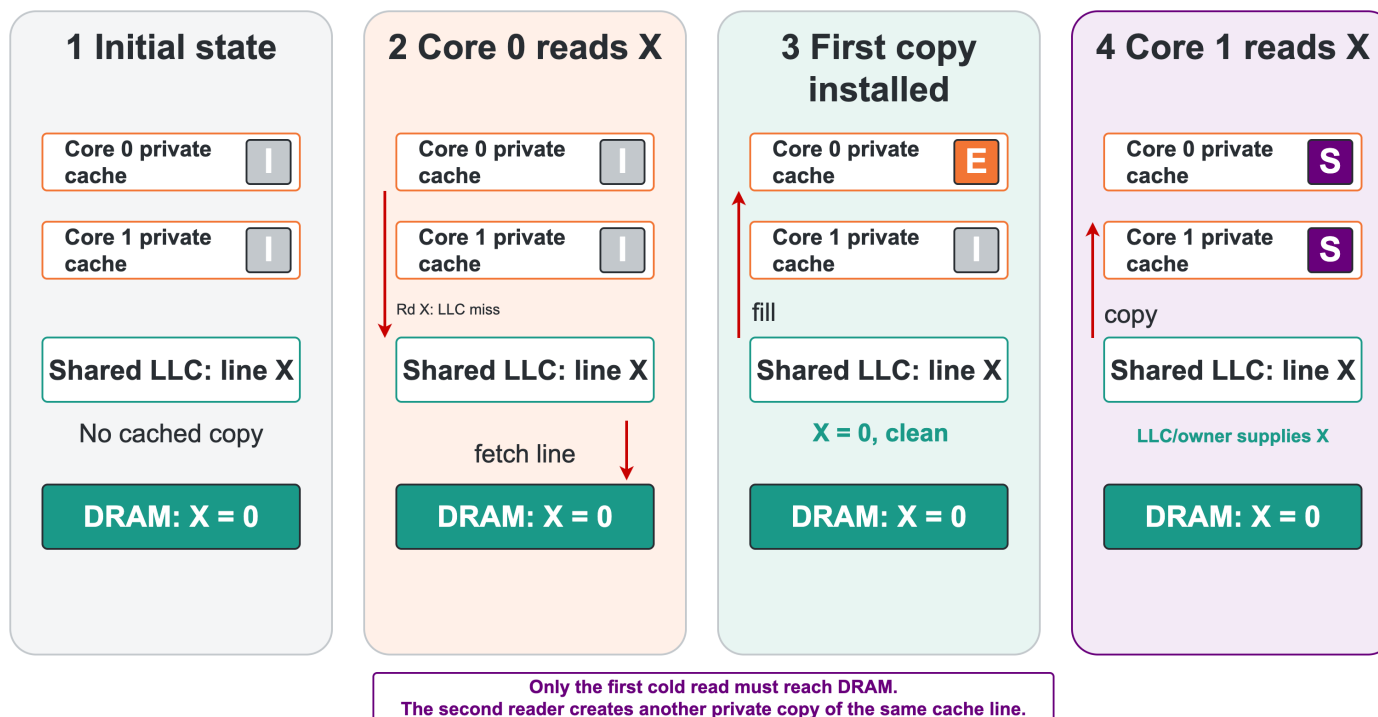
(a) Read.



(b) Write.



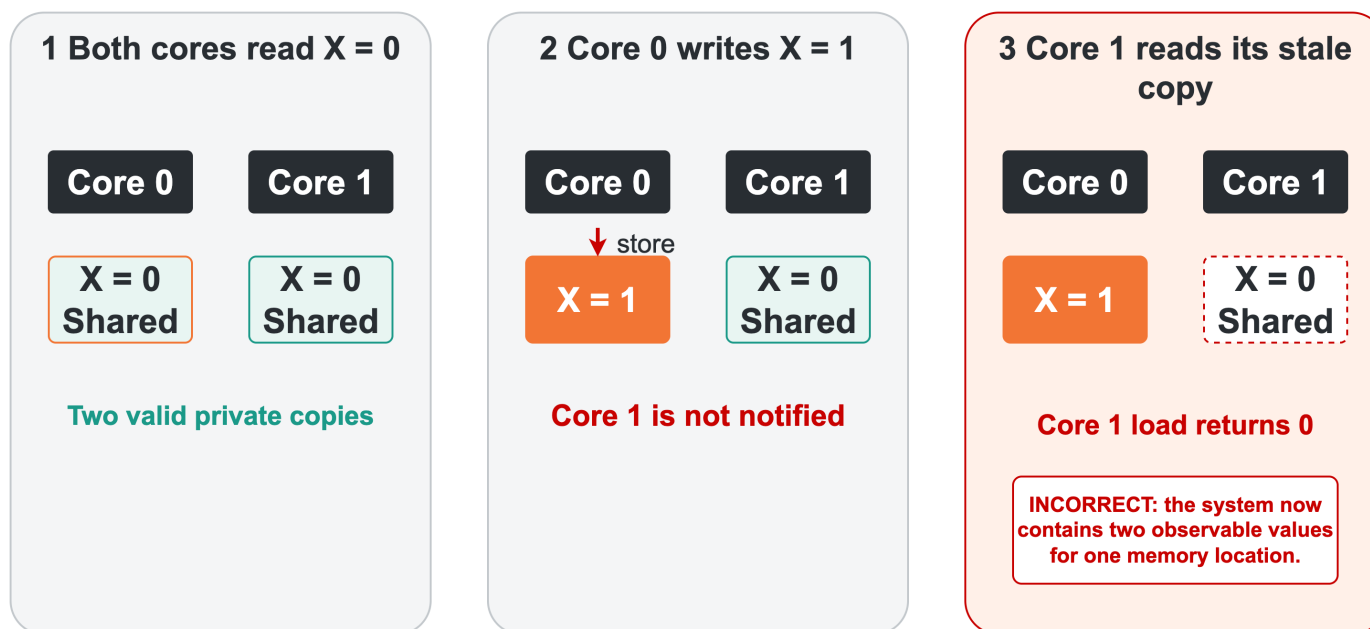
Multicore Technologies: Shared LLC



- Only the first cold read must reach DRAM. A second reader can obtain the line from the LLC or current owner.
- This is a simplified inclusive teaching model; modern LLCs may also be non-inclusive.

Multicore Technologies: Coherence

A write can make another core's copy stale

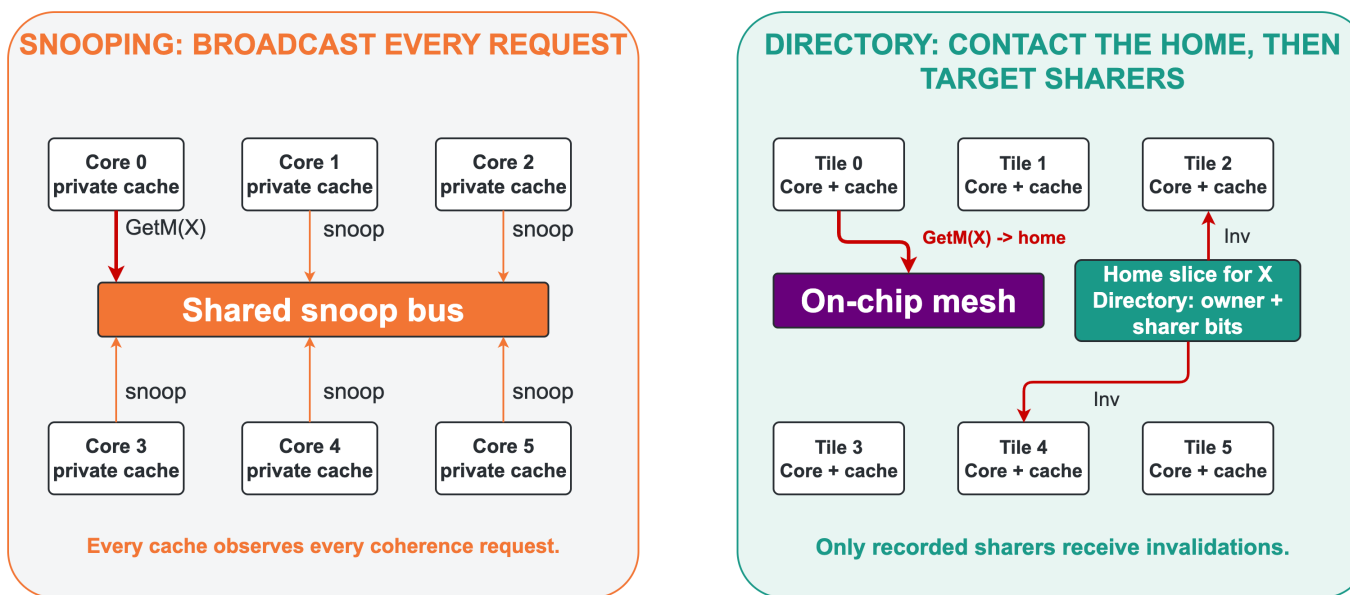


Write Propagation: Other processors must eventually observe the write

Write Serialization: All processors must agree on one order for writes to the same location

Multicore Technologies: Coherence

Coherence Protocols: Snooping and Directory



MESI = permission/state machine | Snooping or directory = how coherence messages are transported

- Snooping broadcasts coherence requests. A directory tracks owners and sharers to target the messages.
- MESI describes permissions. Snooping and directory designs determine how coherence messages reach the caches.

Capacity Wall of LLM Inference

A Case Study of DeepSeek

Approximate Session's KV-Cache Size

Context Length	DeepSeek v3.2	DeepSeek v4 (w CSA/HCA/SWA)
128K	~5.72 GiB	~0.62 GiB
256K	~11.44GiB	~1.21GiB
1M	~45.75 GiB	~4.83 GiB
2M	~91.50 GiB	~9.65 GiB

Given GB200 GPU Specification

Available HBM capacity for KV-Cache: ~150GB

At 1M context length, the maximum sessions can a GPU holds are:

DeepSeek v3.2: ~4

DeepSeek v4pro: ~32

GPU is much underutilized!

Capacity Wall: HBM can only hold limited KV-Caches, resulting in low batch size and computation utilization.

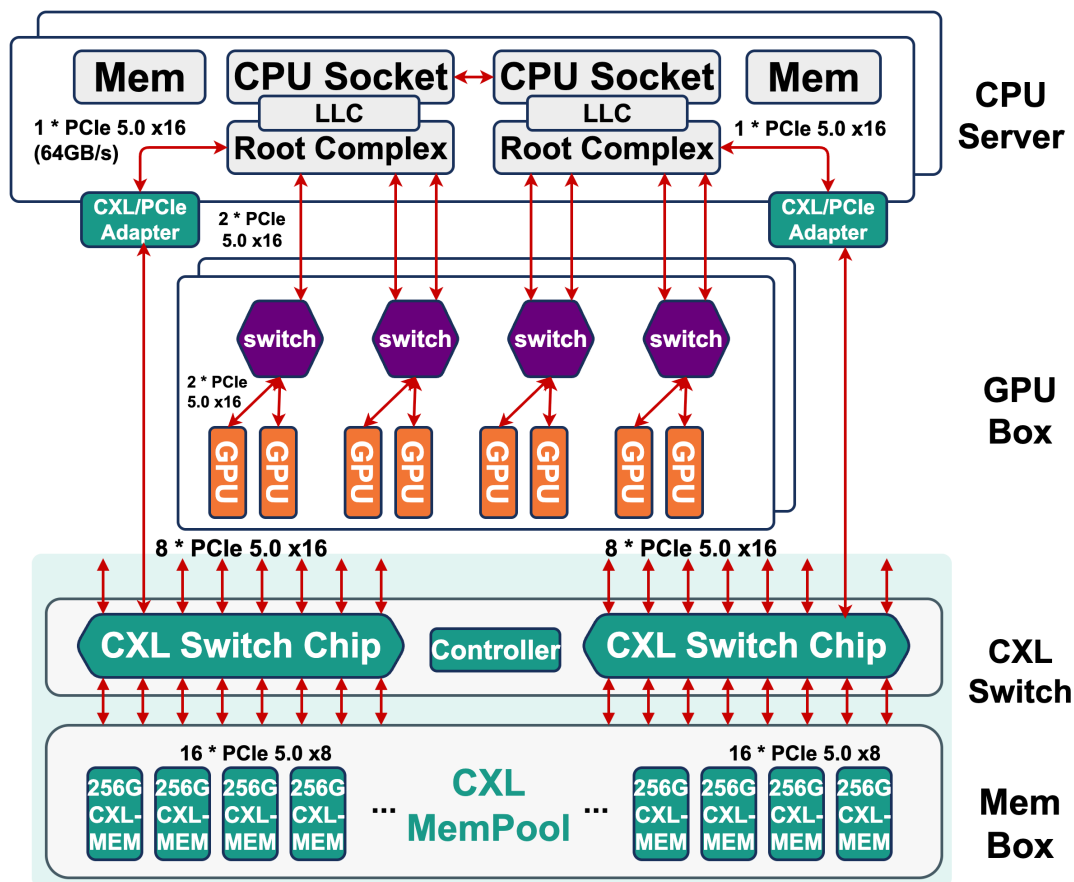
Memory Extension: CXL

Host address map

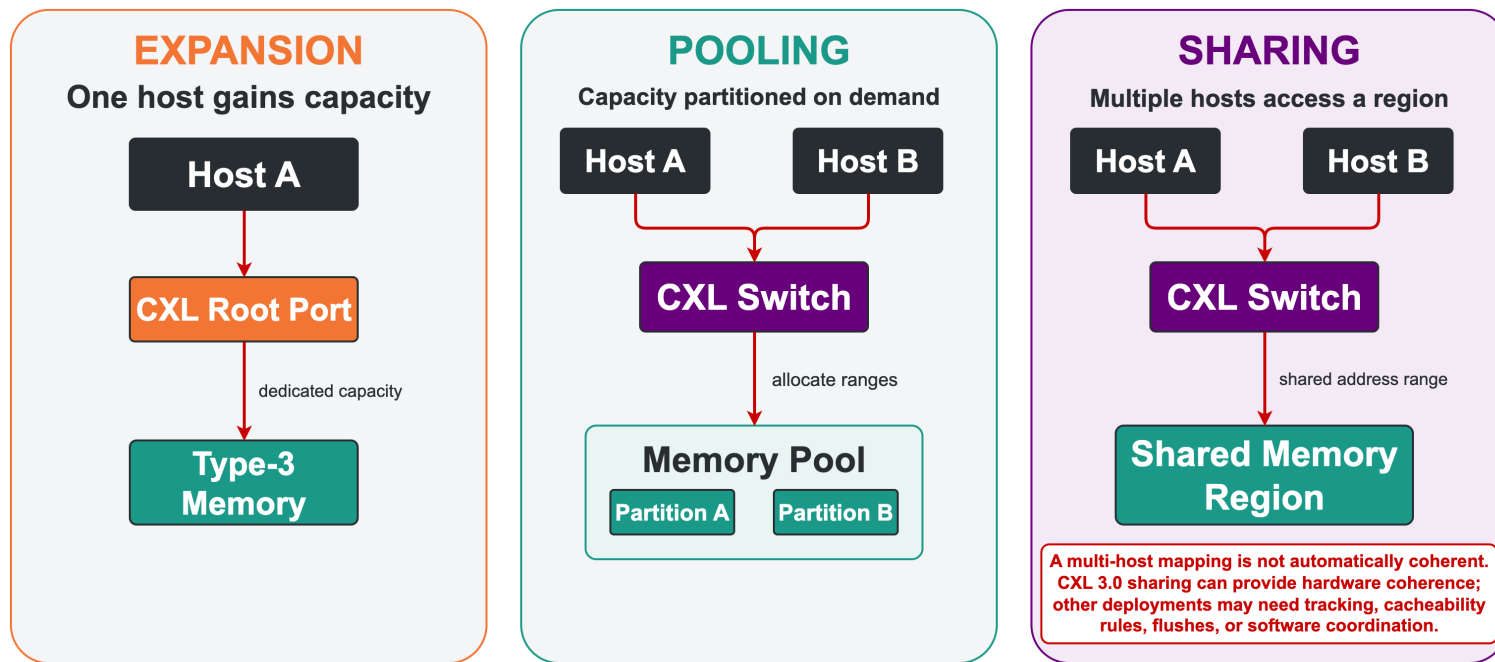
Unified memory across
host DRAM/HBM and CXL
HDM

Software sees host-
memory ranges

Hardware translates
load/store instructions into
CXL remote memory
accesses



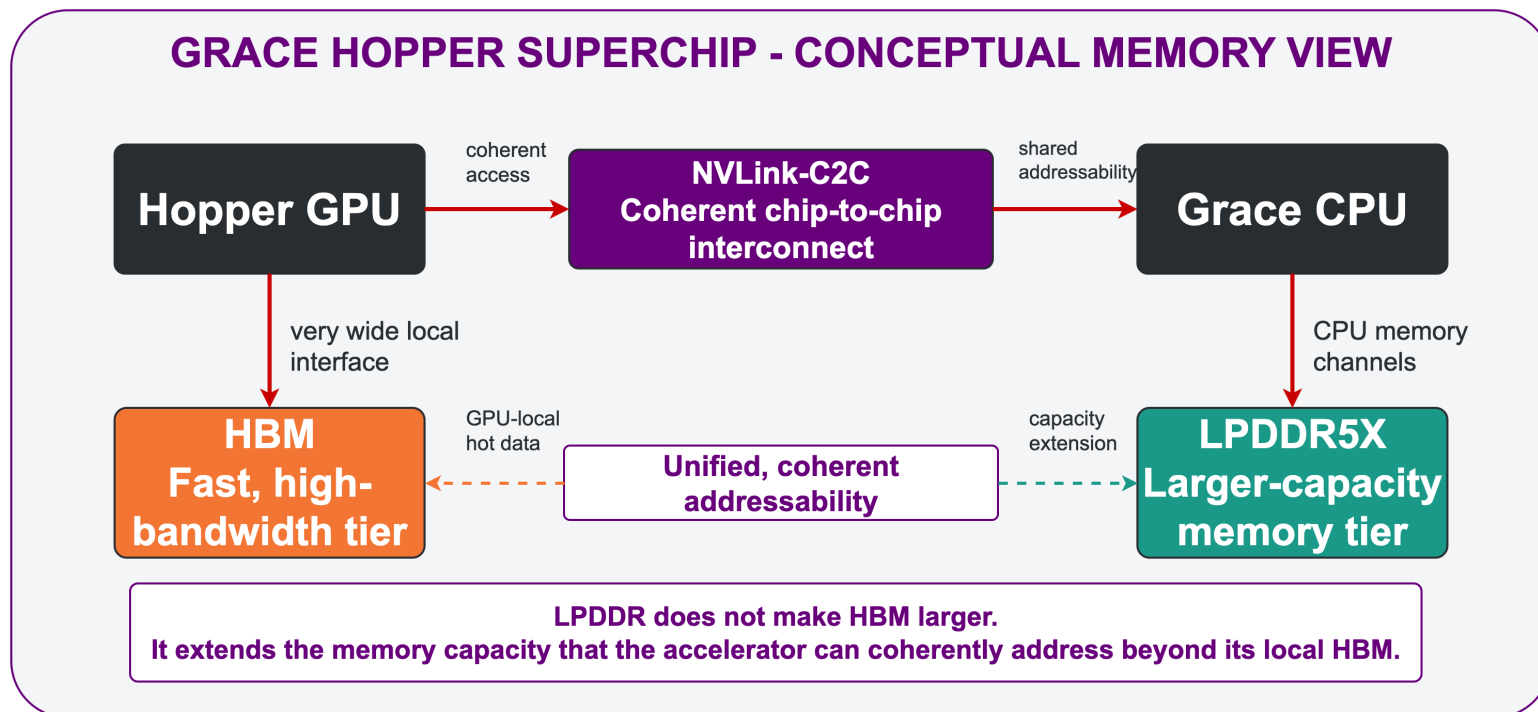
CXL Switches Enable Expansion, Pooling, and Sharing



CXL changes capacity from a fixed motherboard resource into a composable system resource.

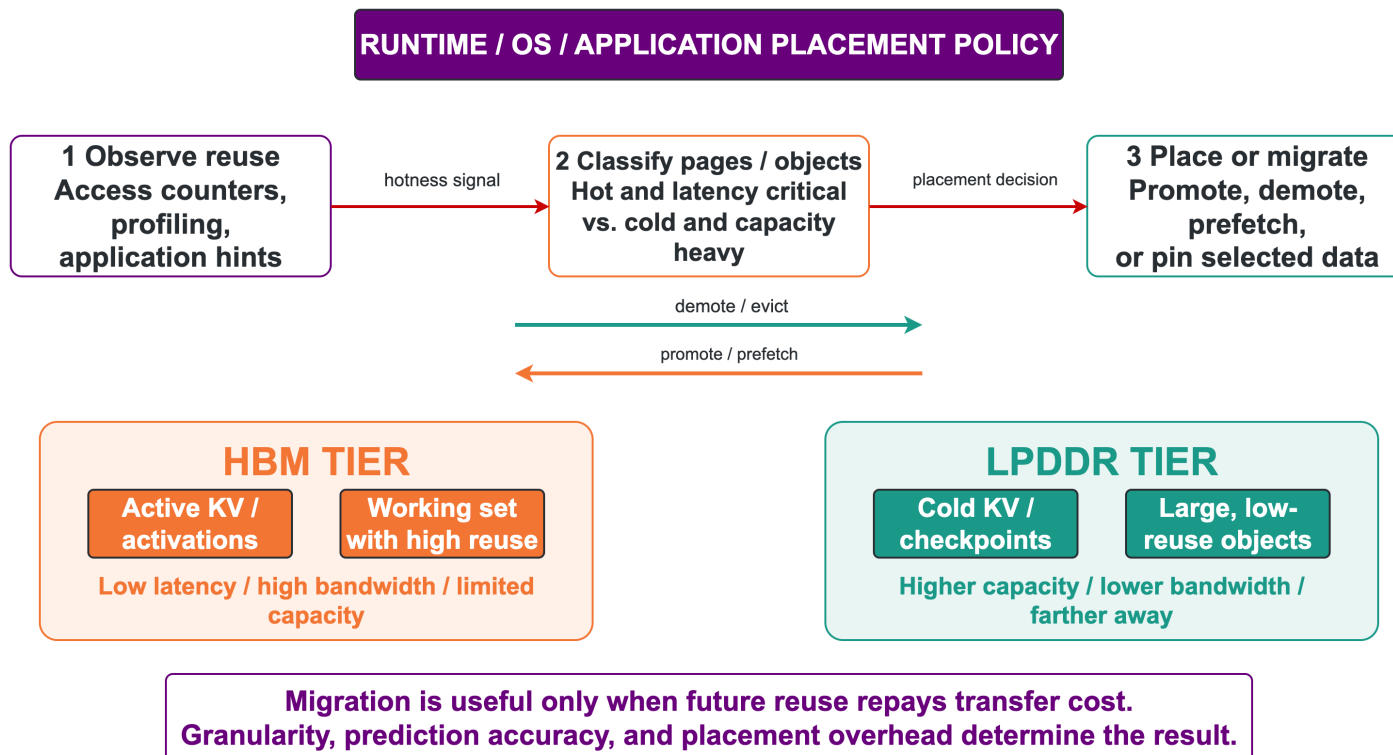
- Expansion adds capacity to one host. Pooling assigns portions of memory devices to different hosts.
- CXL 3.0 coherent sharing differs from merely allowing several hosts to map the same memory region.

Memory Extension: LPDDR



- Hopper retains local HBM. Grace connects LPDDR5X through the coherent NVLink-C2C CPU–GPU link.
- LPDDR extends the capacity the accelerator can address; it does not enlarge HBM itself.

Tiered Memory Management Decides What Stays in HBM



- Keep hot, latency-critical, frequently reused data in HBM. Place large, cold objects in the LPDDR capacity tier.
- Promotion helps only when future reuse saves more than the transfer and bookkeeping cost.