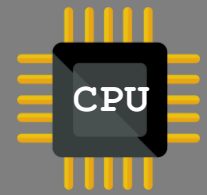


Topic 5

Memories III

Xinfei Guo
xinfei.guo@sjtu.edu.cn

June 20th, 2022

T5 learning goals

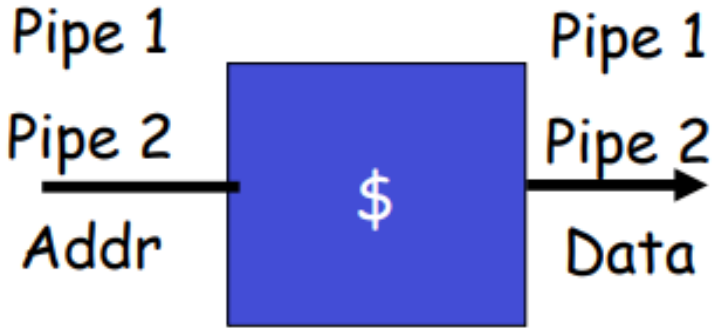
- Memories
  - Memories I: Memory Overview
  - Memories II: Cache (Advanced Cache Optimizations)
  - Memories III: More Advanced Cache Optimizations & Memory Management

Recall Superscalar

- Increasing issue width => wider caches
- Parallel cache access is harder than parallel FUs
  - fundamental difference: caches have state, FUs don't
  - one port affects future for other ports
- Several approaches used
  - true multi-porting
  - muliple cache copies
  - virtual multi-porting
  - Non-Uniform Cache Architecture (NUCA)

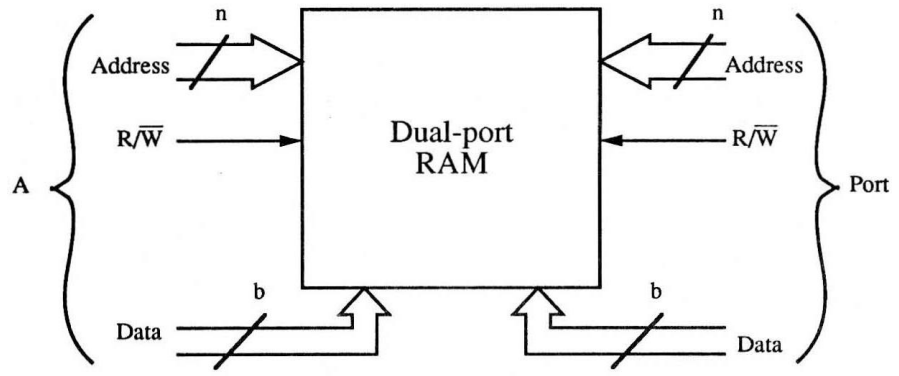
Virtual Multi-Porting

- Used in Power2 and Alpha 21264
  - 21264 uses wave pipelining
- Time-share a single port
  - may require access to be faster than clock
  - Requires very careful array design to guarantee balanced paths
  - probably not scalable beyond 2



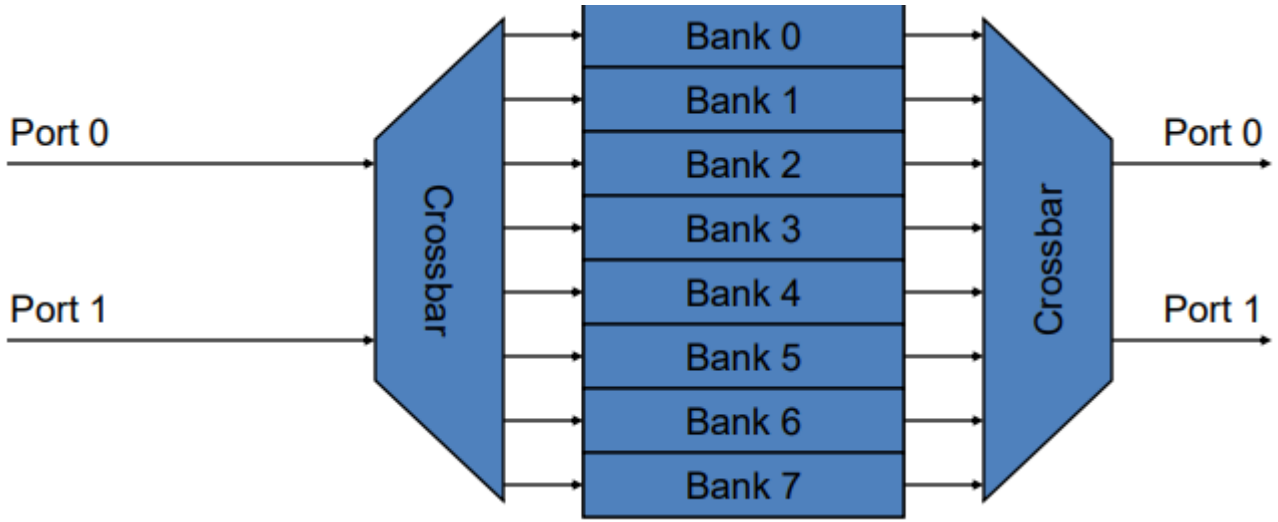
Multi-Port Caches: True Multiporting

- Superscalar processors requires multiple data references per cycle
- Time-muliplex a single port (double pump)
  - need cache access to be faster than datapath clock
  - not scalable
- Truly muliported SRAMs are available, but
  - more area
  - slower access (undesirable for L1 D$
  - SRAM access difficult to pipeline



Multiporting Multi-banking Caches

- Used in Intel Pentium (8 banks)
- Need routing network
- Must deal with bank conflicts
  - Many simultaneous requests to same bank
  - Bank conflicts not known till address generated

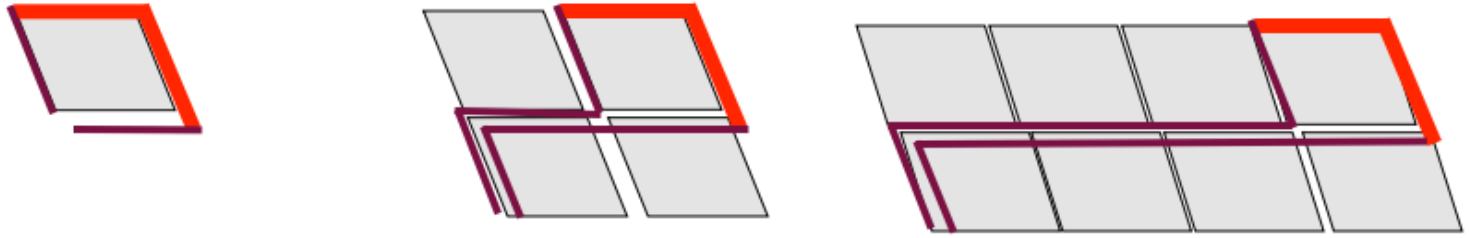


More Combined Schemes

- Virtual multiporting of multiple banks
- Multiple ports and virtual multiporting
- Multiple banks with multiply virtually multiported ports
- No good solution known at this time
  - Current generation superscalars get by with 1-3 ports

Non-Uniform Cache Architecture (NUCA)

- ASPLOS 2002 proposed by UT-Austin (Kim, Burger, Keckler)
- Facts
  - Large shared on-die L2
  - Wire-delay dominating on-die cache

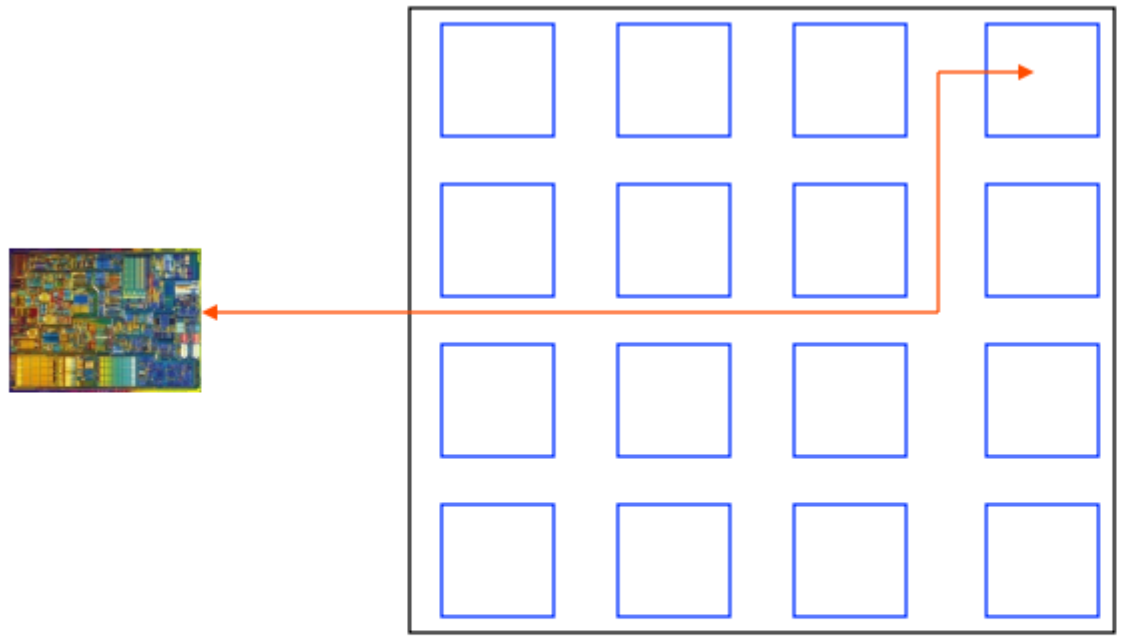


3 cycles
1MB
180nm, 1999

11 cycles
4MB
90nm, 2004

24 cycles
16MB
50nm, 2010

Multi-banked L2 cache

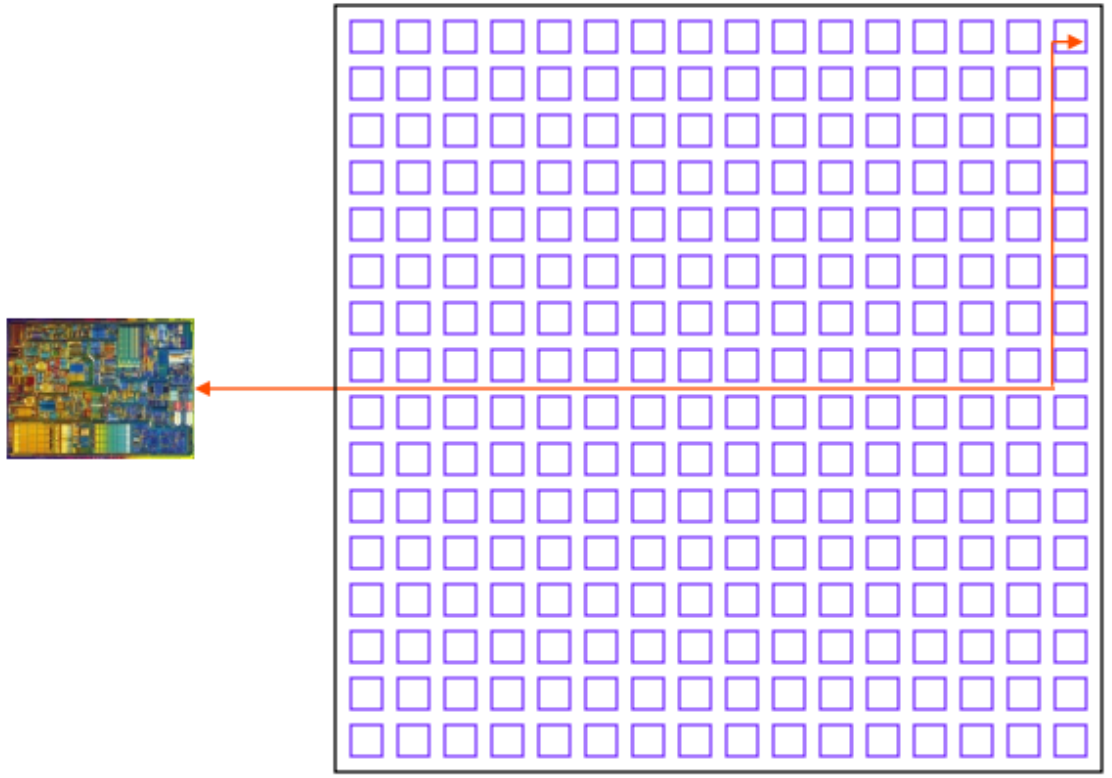


Bank=128KB
11 cycles

2MB @ 130nm

Bank Access time = 3 cycles
Interconnect delay = 8 cycles

Multi-banked L2 cache



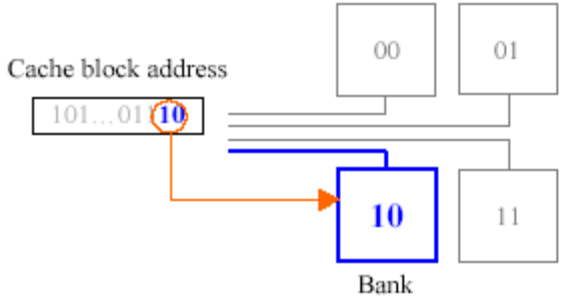
Bank=64KB
47 cycles

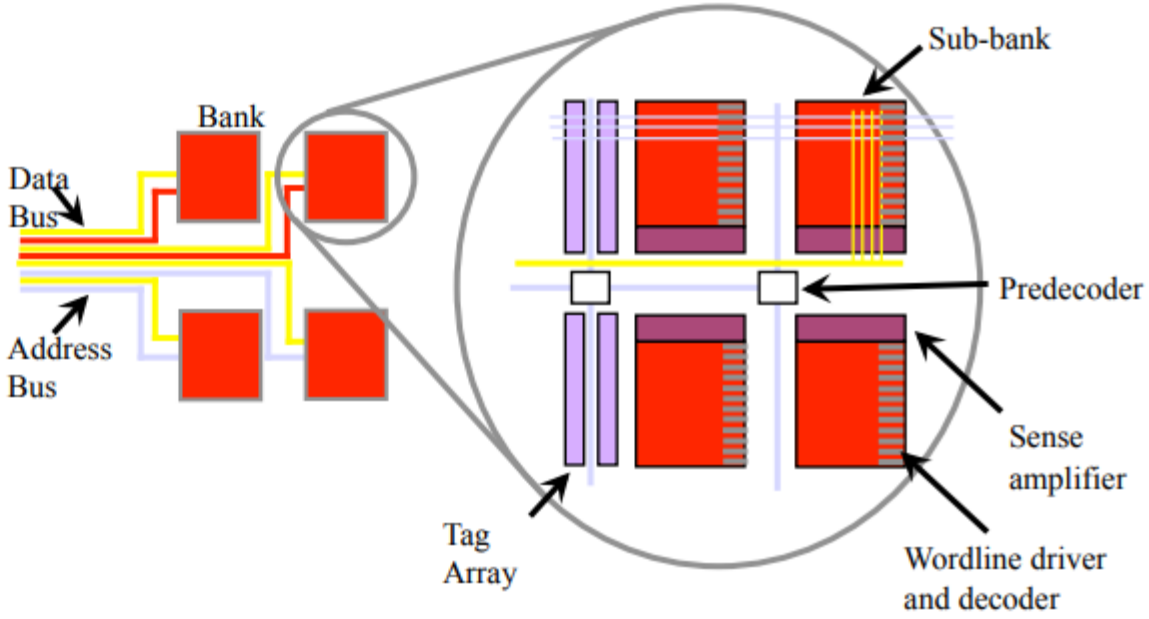
16MB @ 50nm

Bank Access time = 3 cycles
Interconnect delay = 44 cycles

Static NUCA-1

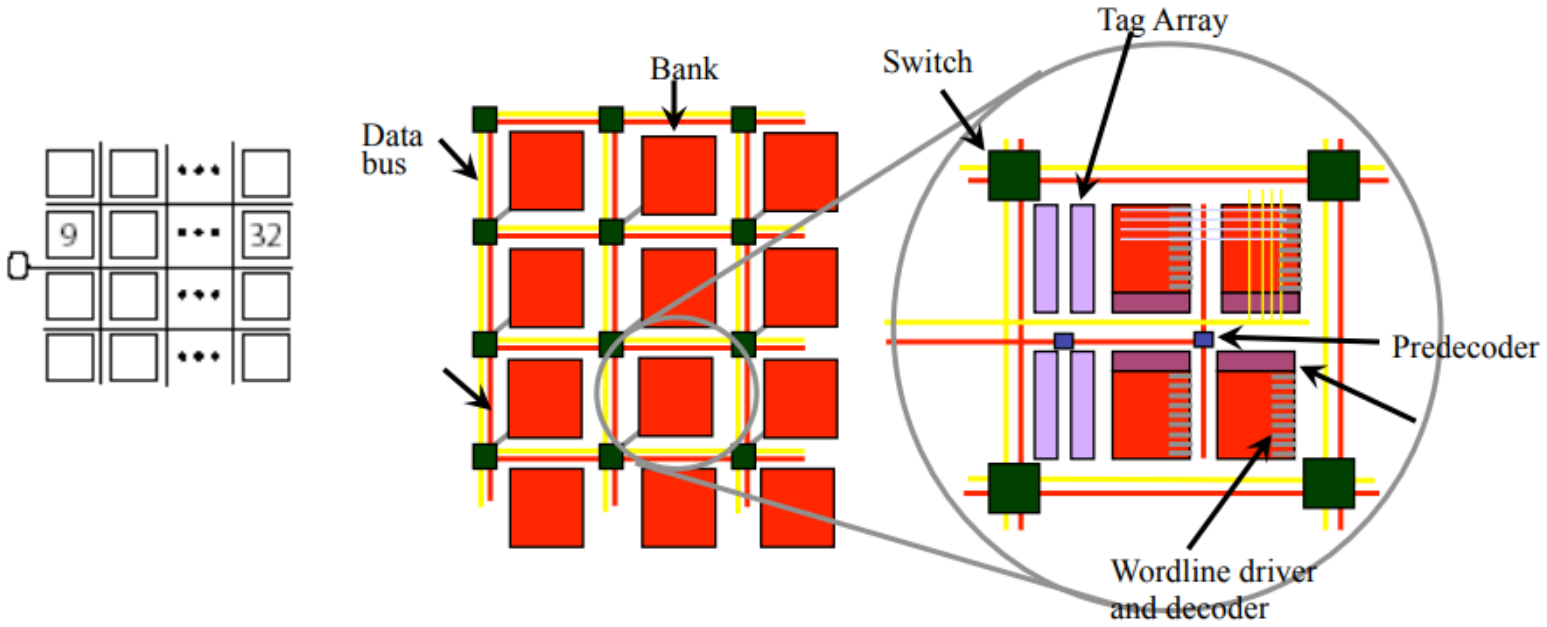
- Used in IBM Power4 level-2 cache
- Use private per-bank channel
- Each bank has its distinct access latency
- Statically decide data location for its given address
- Average access latency =34.2 cycles
- Wire overhead = 20.9% → an issue





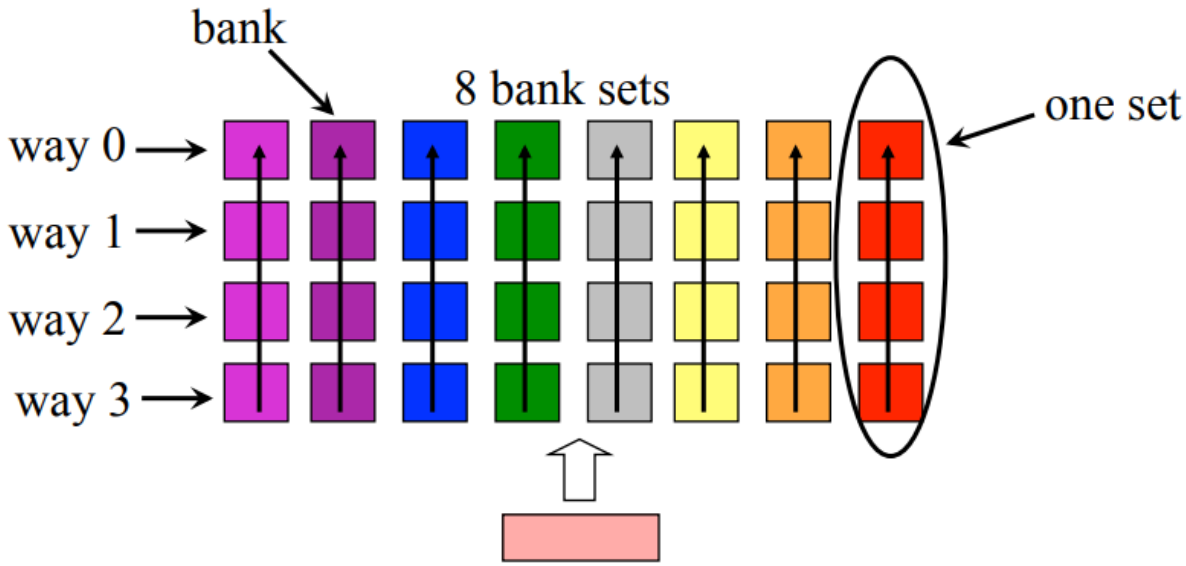
Static NUCA-2 (similar to NoC)

- Use a 2D switched network to alleviate wire area overhead
- Average access latency =24.2 cycles
- Wire overhead = 5.9%



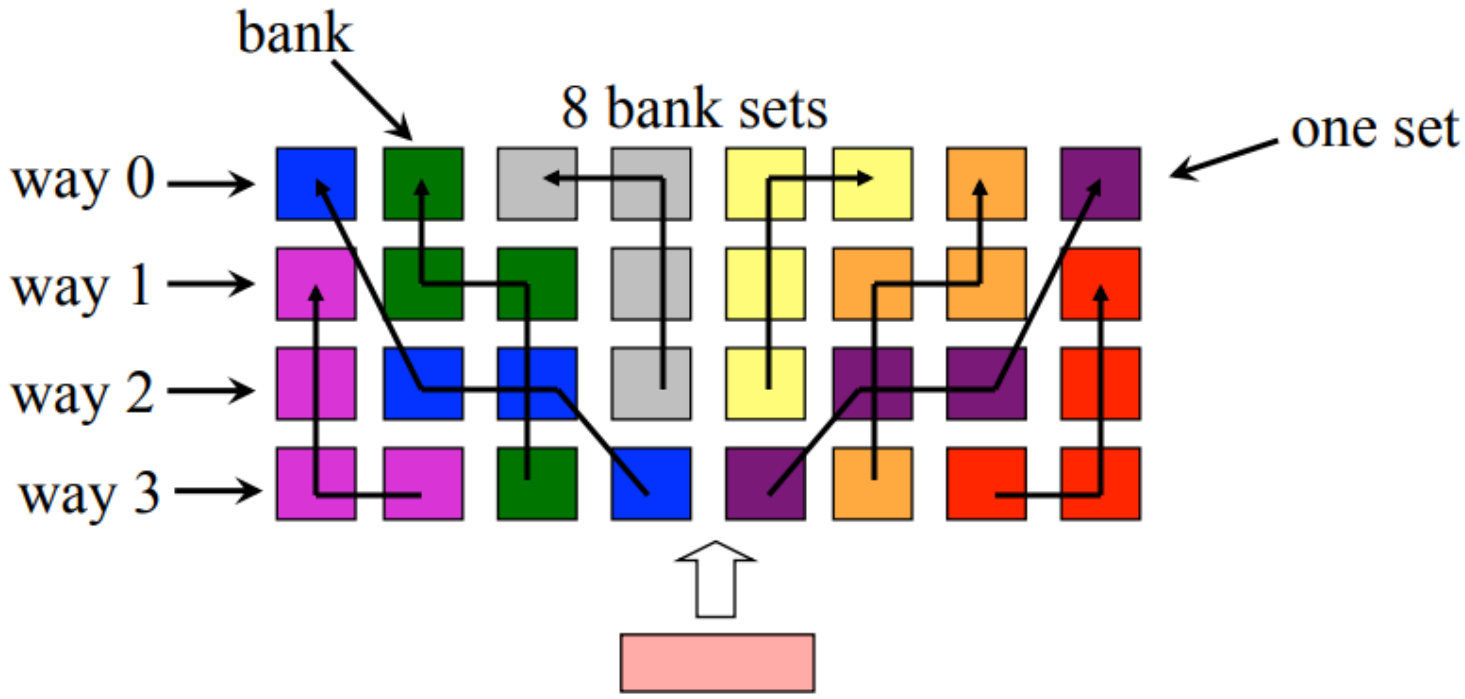
Dynamic NUCA

- Data can dynamically migrate
- Move frequently used cache lines closer to CPU
- Simple Mapping
  - All 4 ways of each bank set needs to be searched
  - Farther bank sets → longer access



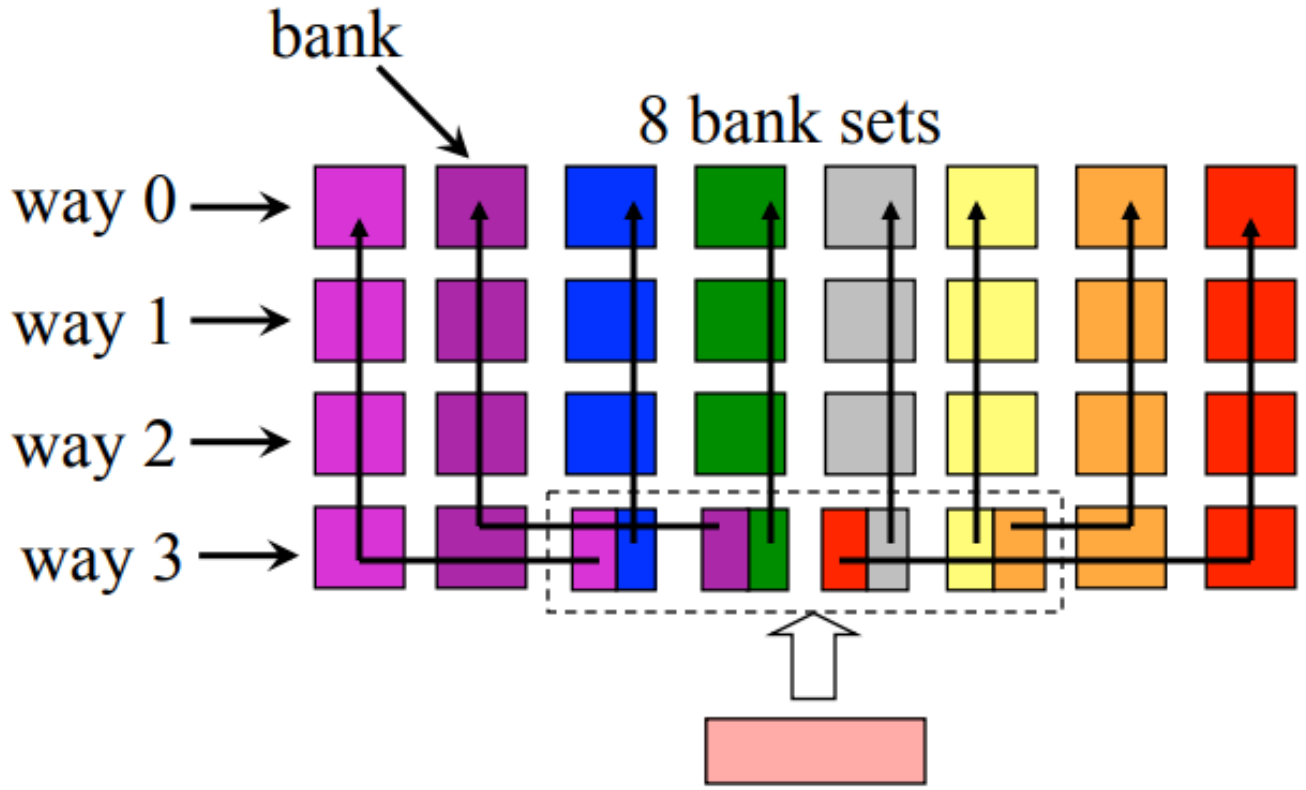
Dynamic NUCA

- Fair Mapping
  - Average access time across all bank sets are equal



Dynamic NUCA

- Shared Mapping
  - Sharing the closest banks for farther banks

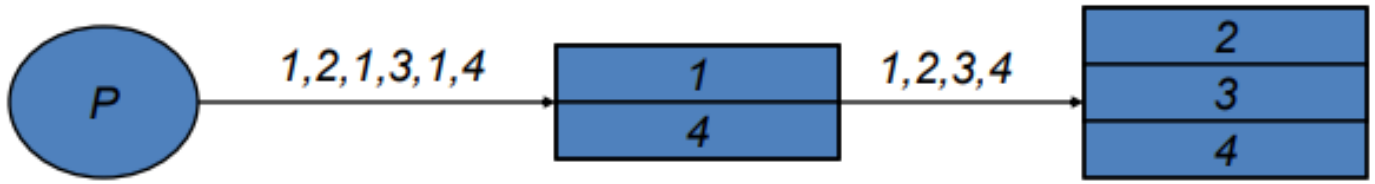


Latency vs. Bandwidth

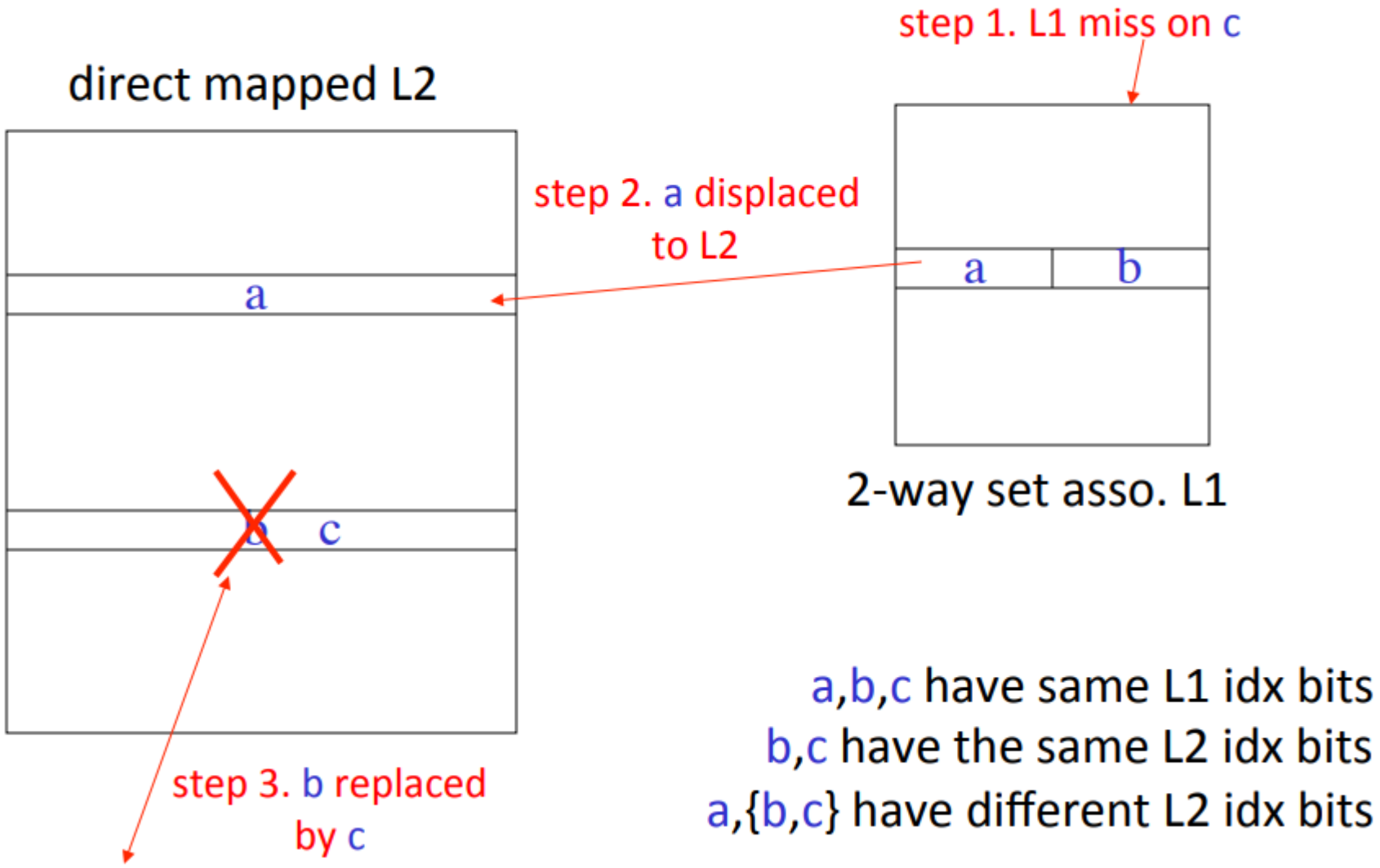
- Bandwidth can be handled by “spending” more (hardware cost)
  - Wider buses, interfaces
  - Banking/interleaving, multiporting
- Ignoring cost, a well-designed system should never be bandwidth-limited
  - Can’t ignore cost!
- Bandwidth improvement usually increases latency
  - No free lunch
- Hierarchies decrease bandwidth demand to lower levels
  - Serve as traffic filters: a hit in L1 is filtered from L2
- Parallelism puts more demand on bandwidth
- If average b/w demand is not met => infinite queues
  - Bursts are smoothed by queues
- If burst is much larger than average => long queue
  - Eventually increases delay to unacceptable levels

Inclusion in multilevel caches

- Multi-level inclusion holds if L2 cache is superset of L1
- Example: local LRU (Least recently used) not sufficient to guarantee inclusion
  - Assume L1 holds two and L2 holds three blocks
  - Both use local LRU
- Final state: L1 contains 1, L2 does not
  - Inclusion not maintained
- Different block sizes also complicate inclusion



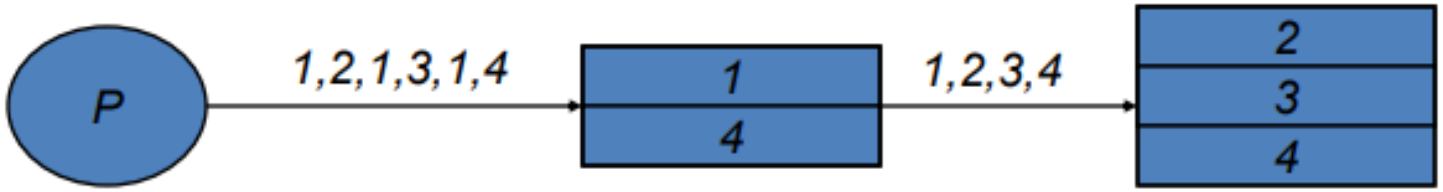
Possible Inclusion Violation



a,b,c have same L1 idx bits
b,c have the same L2 idx bits
a,{b,c} have different L2 idx bits

Multilevel Inclusion

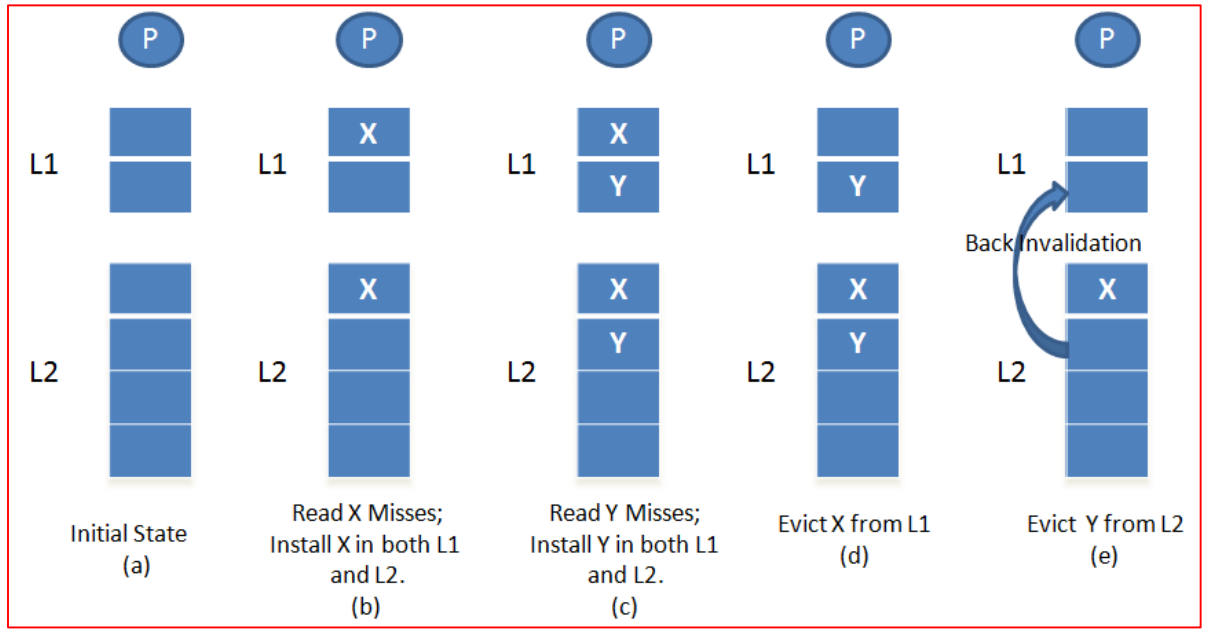
- Inclusion takes effort to maintain
  - Make L2 cache have bits or pointers giving L1 contents
  - Invalidate from L1 before replacing from L2
  - In example, removing 1 from L2 also removes it from L1
- Number of pointers per L2 block
  - L2 blocksize/L1 blocksize



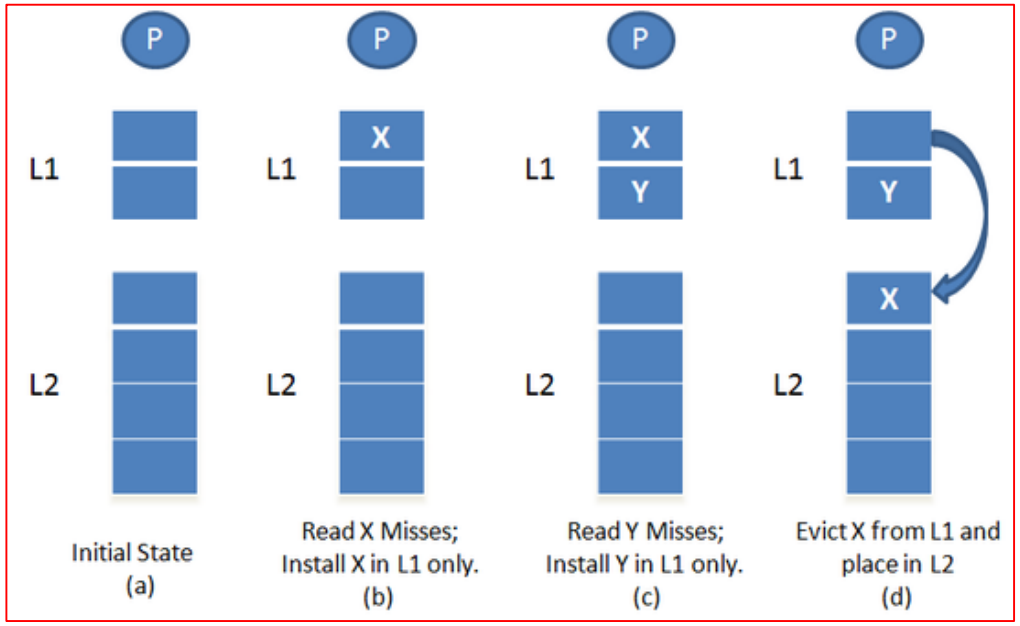
Inclusion versus Exclusion

- **Inclusion**
  - Bring block from memory into L2 then L1
    - A block in the L1 is always in the L2
  - If block evicted from L2, must also evict it from L1
    - Why? more on this when we talk about multicore
- **Exclusion**
  - Bring block from memory into L1 but not L2
    - Move block to L2 on L1 eviction
      - L2 becomes a large victim cache
    - Block is either in L1 or L2 (never both)
  - Good if L2 is small relative to L1
    - Example: AMD's Duron 64KB L1s, 64KB L2

Inclusion versus Exclusion



Inclusive Policy



Exclusive Policy

source: wiki

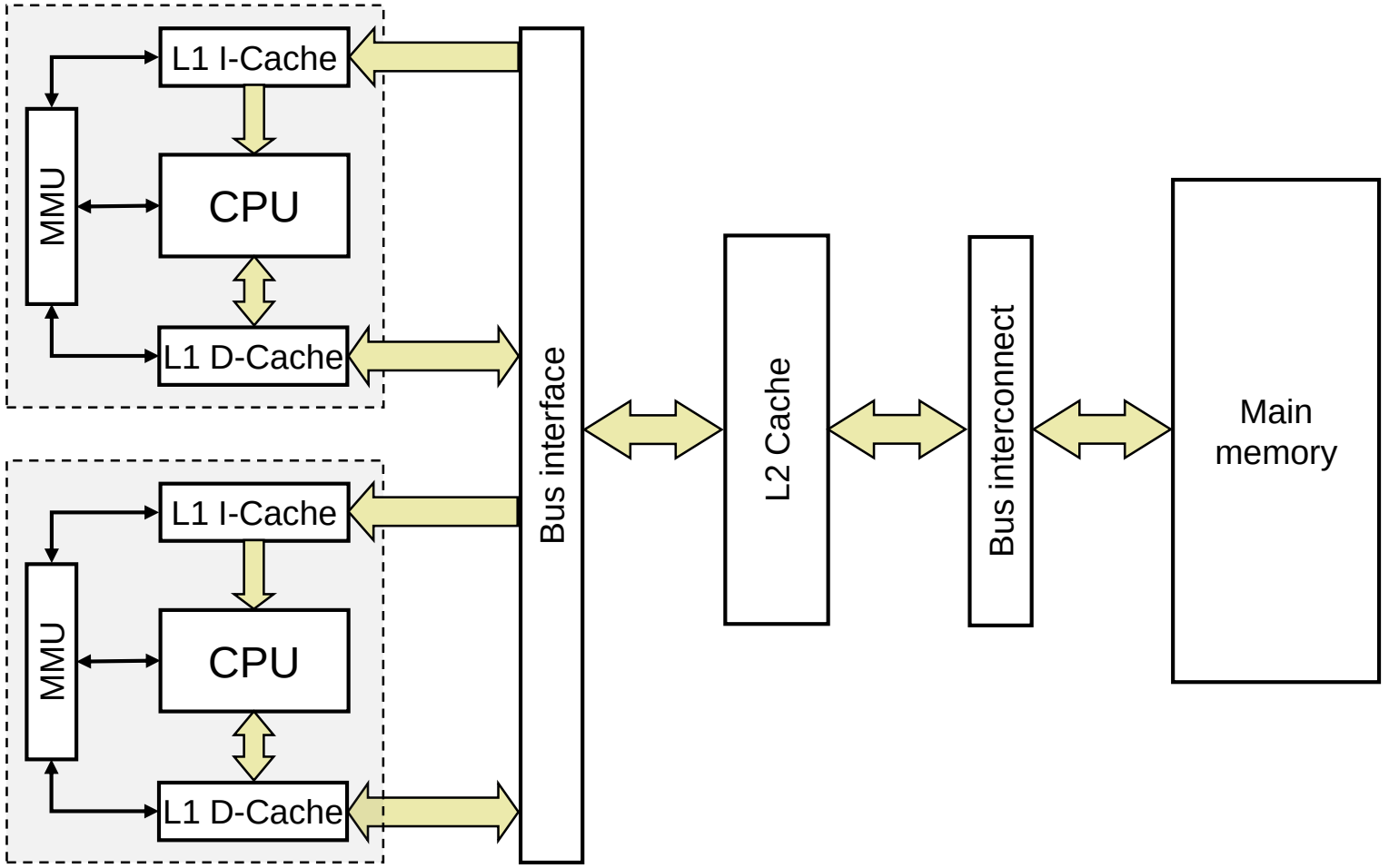
Split vs. Unified Caches

- **Split I$/D$**: insns and data in different caches
  - To minimize structural hazards and t_{access}
  - Larger unified I$/D$ would be slow, 2nd port even slower
  - Optimize I$ and D$ separately
    - Not writes for I$, smaller reads for D$
  - Why is 486 I/D$ unified?

- **Unified L2, L3**: insns and data together
  - To minimize $\%_{miss}$
  + Fewer capacity misses: unused insn capacity can be used for data
  – More conflict misses: insn/data conflicts
    - A much smaller effect in large caches
  - Insn/data structural hazards are rare: simultaneous I$/D$ miss
  - Go even further: unify L2, L3 of multiple cores in a multi-core

Cache Sharing

- Caches can also be shared by several processors in the system.
  - L1 caches are typically private.
  - Sharing often occurs at L2 or L3 caches.
  - If at L3, both L1 and L2 caches are private.



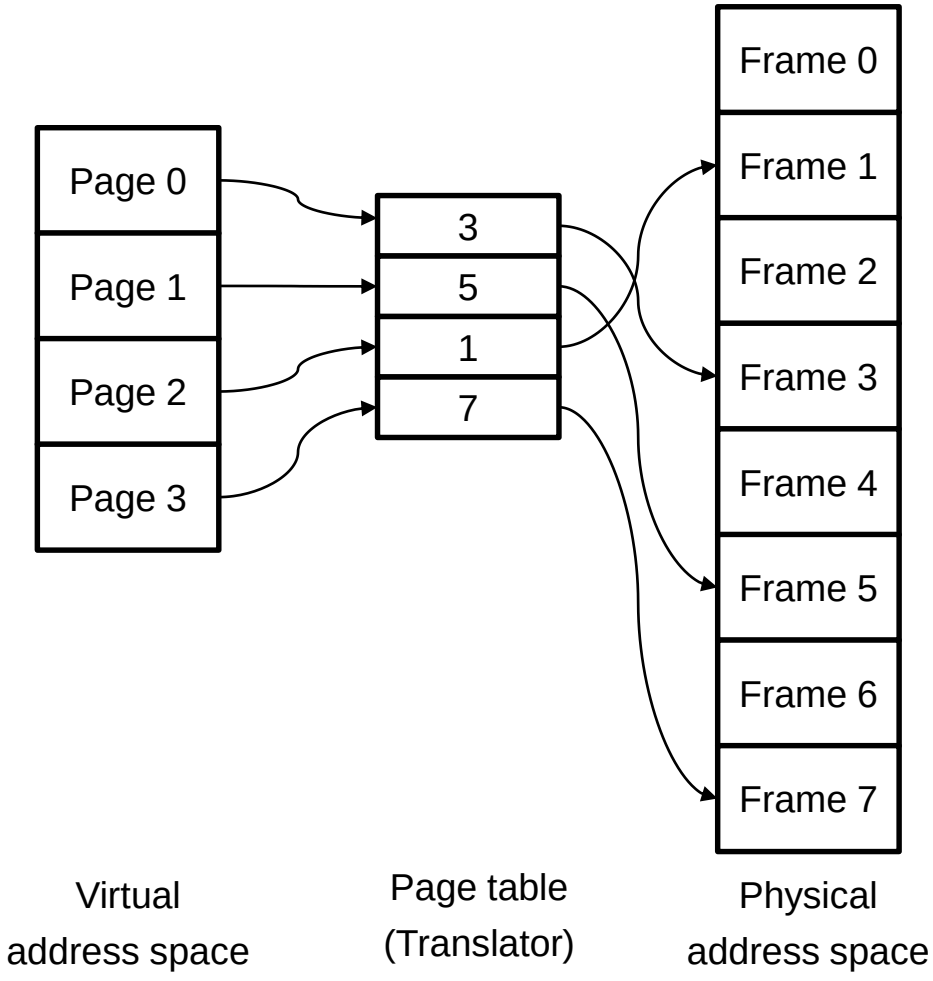
source: ARM

Memory Management: Why Manage Memory?

- Applications' instructions and data are stored in main memory.
  - And cached when required
- While this is fine for a single application, it poses a security risk with multiple programs.
  - An untrusted program could read sensitive data.
  - Or corrupt the state of another application, including taking control of it
- We provide an operating system to prevent this happening.
- What hardware support can we provide to improve the performance of the OS?

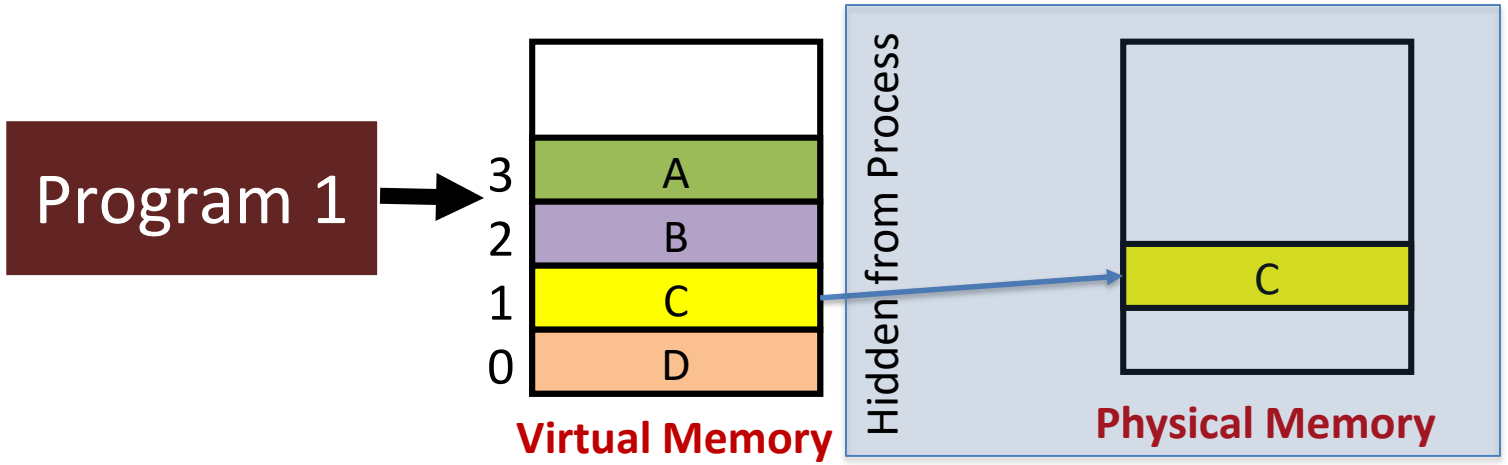
Paging

- The OS forbids direct access to memory.
- Instead, it introduces a layer of indirection.
  - Applications see memory as a range of virtual addresses.
  - The OS maps these to physical addresses.
- Paging is one method to achieve this.
  - Physical memory is split into fixed-sized frames.
  - Virtual memory is split into same-sized pages.
  - Each page is mapped to one frame.
    - Using the high-order bits from the address
    - This information is kept in the page table.
  - Virtual Memory allows the users to run more than one application at once.



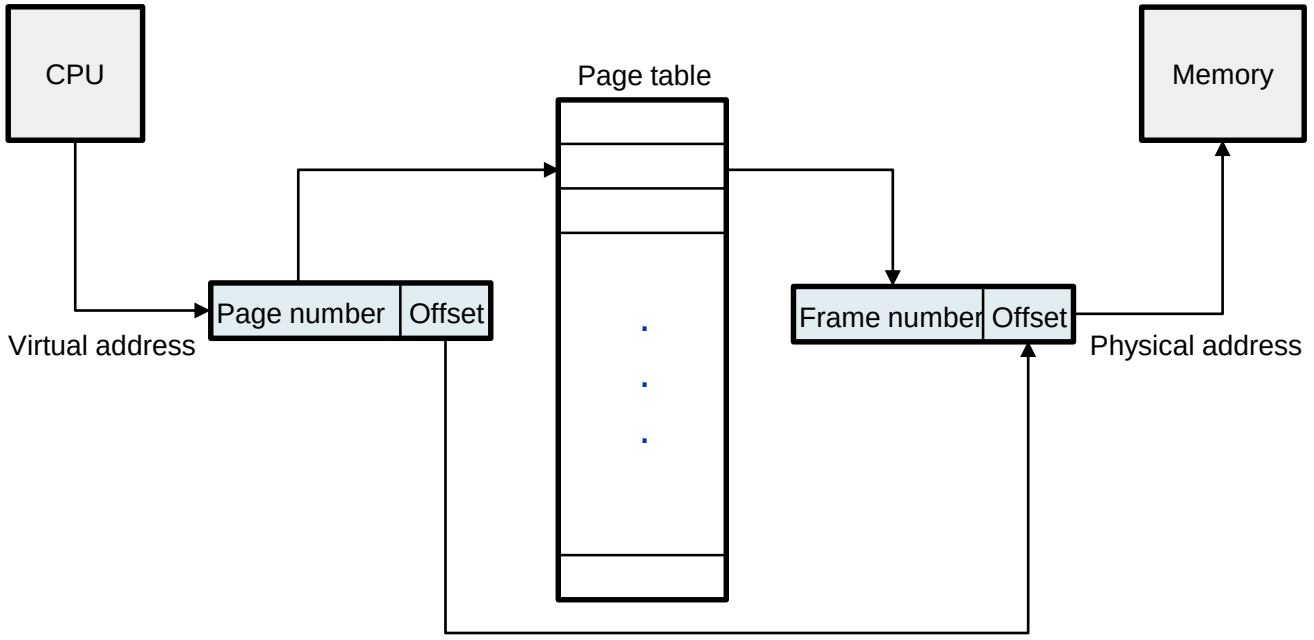
Review: Virtual Memory

- Give each program an illusion that it has exclusive access to entire main memory
- Big, Private, Cover



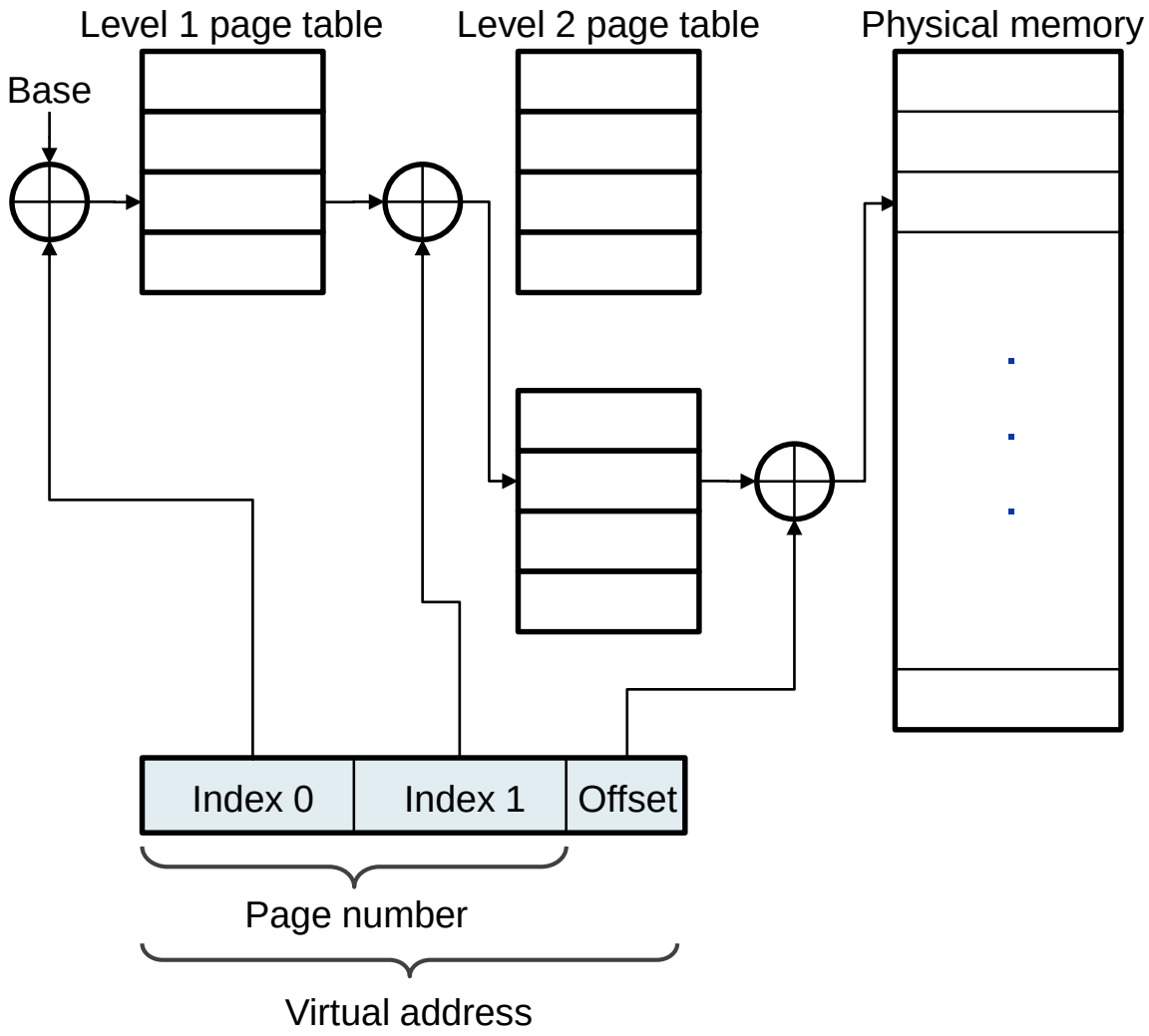
Page Address Translation

- The CPU works on virtual addresses.
  - To access physical memory, the virtual address has to be converted to a physical address by indexing into the page table.
- However, this makes the page table extremely large.



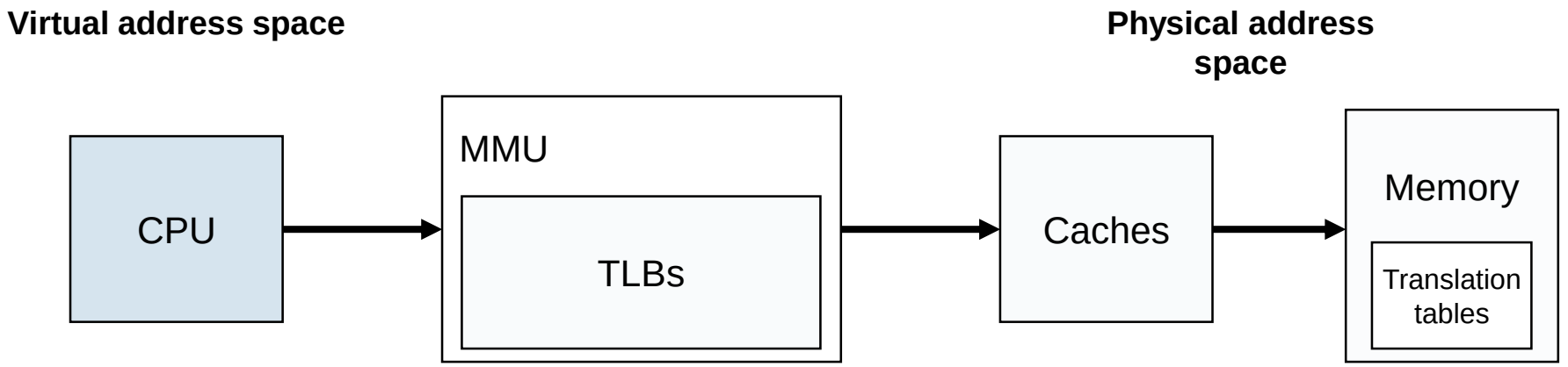
Multi-level Page Tables

- To prevent the need for a large page table, we can create a tree of page tables.
  - Paging the page table
  - First-level entries point to second-level tables.
  - Second-level entries point to third-level tables, etc.
  - Final-level entries point to memory frames.
- To access a frame, we must walk through the page tables using the virtual address.
  - This is extremely costly.
- Can we provide hardware to help?

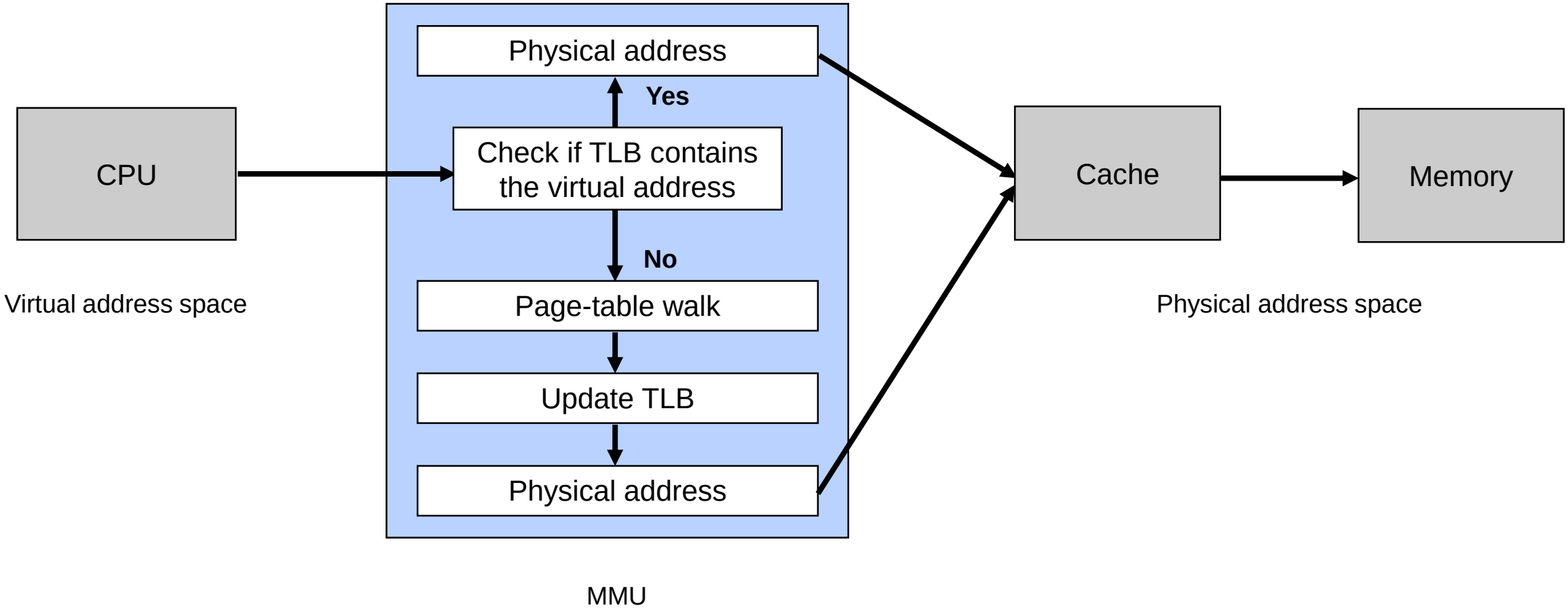


Memory-management Unit (MMU)

- Handles translation of virtual addresses to physical addresses
- The MMU provides hardware to read translation tables in memory.
- The translation lookaside buffers (TLBs) cache recent translations.

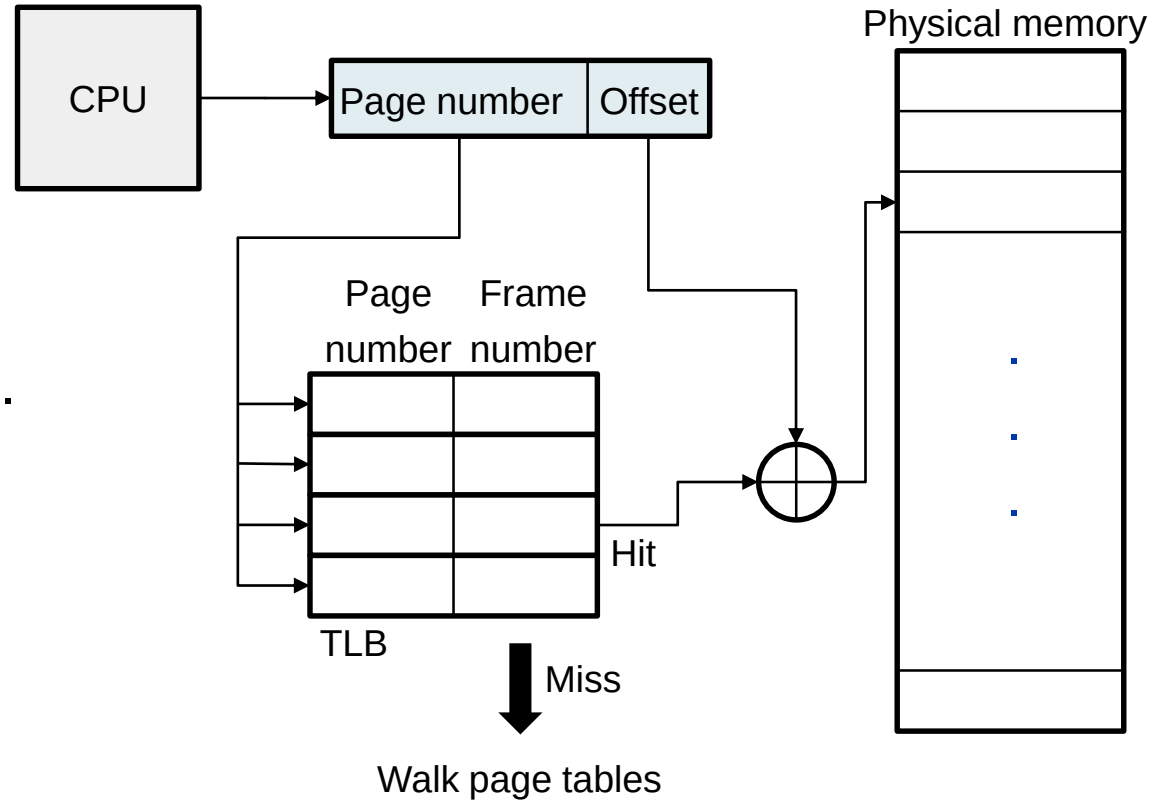


Overview of Memory Access Using an MMU



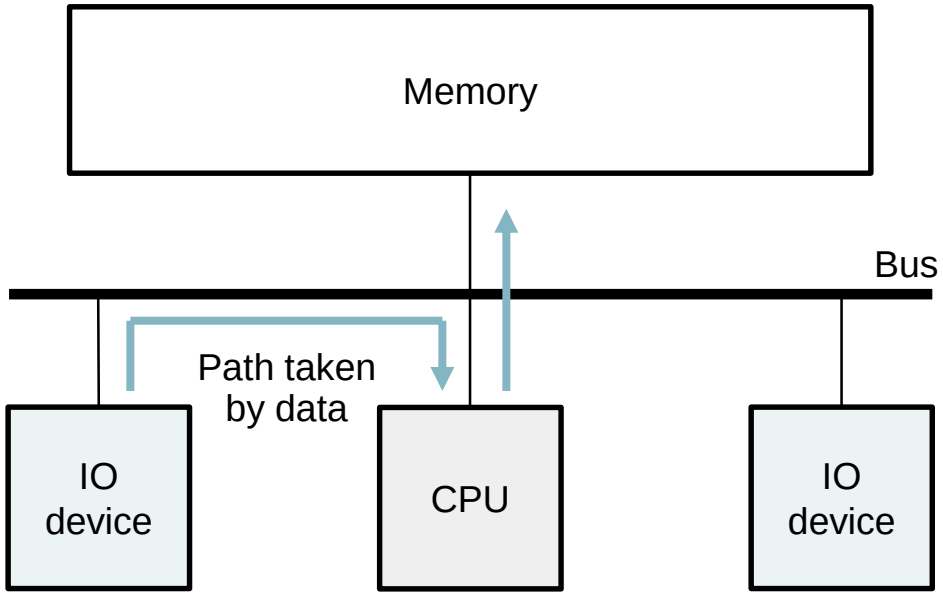
Fast Translation Using a TLB

- Translation Look-aside Buffer
  - Load part of the Page Table in cache
  - One translation per TLB entry
- The TLB is a cache of page translations.
  - Provides access to recent virtual to physical address mappings quickly
- Each block is one or two page-table entries.
- TLBs are usually fully associative.
- On a hit, forward the physical address to the L1 cache.
- On a miss, the MMU will walk the page tables to find the translation.
- Usually fully associative (why?)



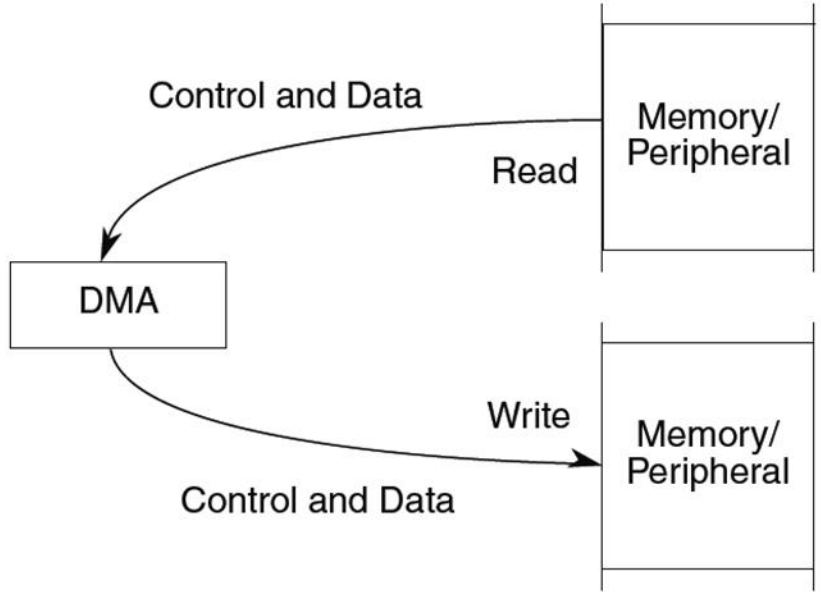
DMA: Direct memory access

- When an application wants to access IO, it does so through the OS.
  - Using a system call to read from or write to the device
- This requires the CPU to be involved in transferring data between the device and memory.
  - On a read, data are then available to the program in memory.
- This data transfer is costly and wasteful.
  - CPU can’t do any useful work while waiting.
  - Data travel further and take longer to transmit.

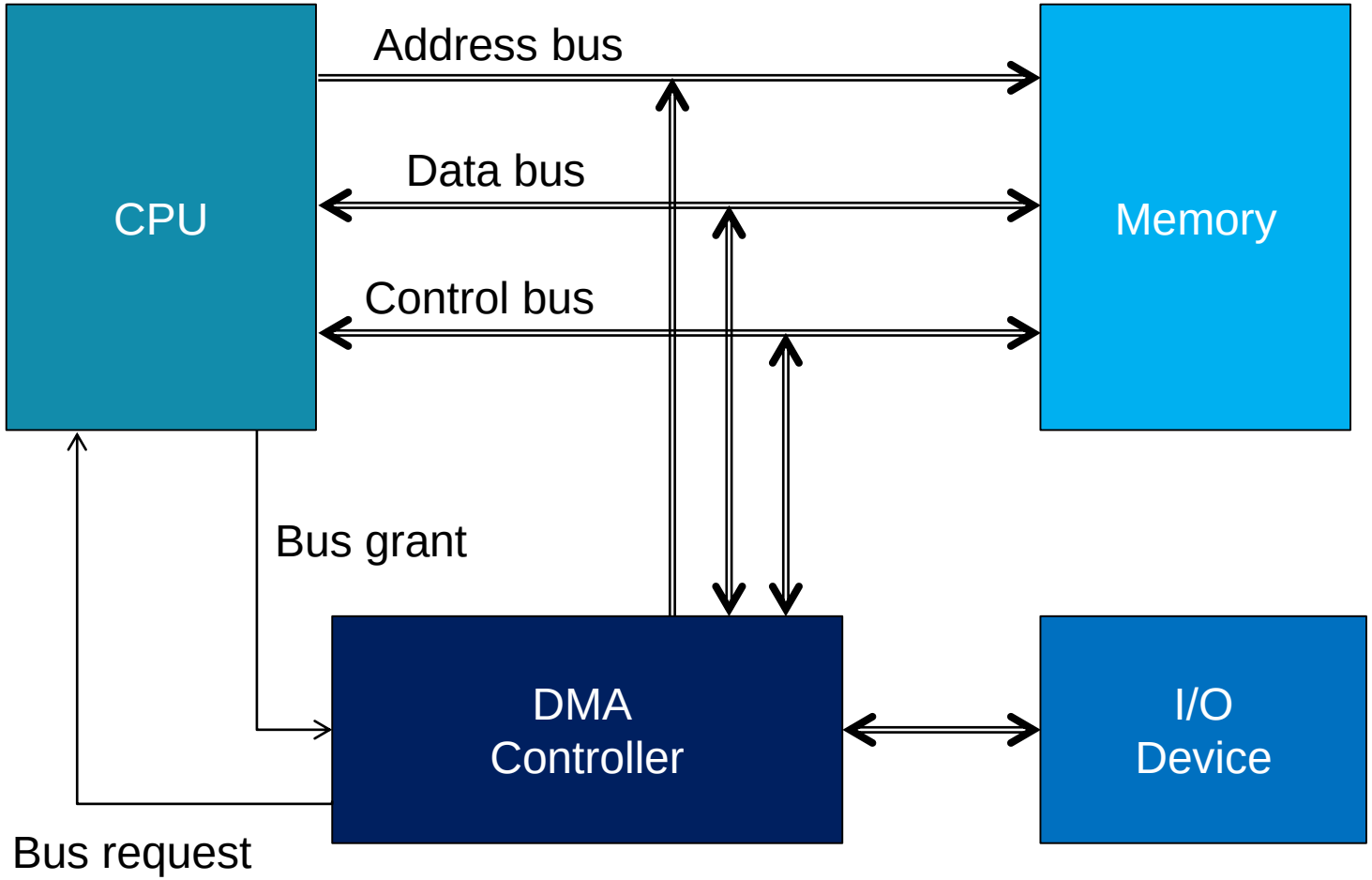


DMA

- Instead of involving the CPU, provide dedicated hardware to control the transfer.
  - A controller for direct memory access, or DMA.
  - CPU now just required to configure this DMA controller correctly.
- Typically supports multiple configurable options:
  - Number of data items to copy
  - Source and destination addresses
    - Fixed or changeable (e.g., increment and decrement)
  - Size of data item
  - Timing of transfer start
- A DMA controller can also work with interrupts, e.g., interrupt CPU at the end of a transfer.
- The main idea is to exempt the CPU from busy-waiting and frequent interruptions.



DMA Architecture



DMA Transfer Modes

- Burst
  - An entire block of data is transferred in one contiguous sequence.
  - The CPU remains inactive for relatively long periods of time (until the whole transfer is completed).
- Cycle stealing
  - DMA transfers one byte of data and then releases the bus returning control to the CPU.
  - Continually issues requests, transferring one byte per request, until it has transferred the entire block of data
  - It takes longer to transfer data/the CPU is blocked for less time.
- Transparent
  - DMA transfers data when the CPU is performing operations that do not use the system buses.

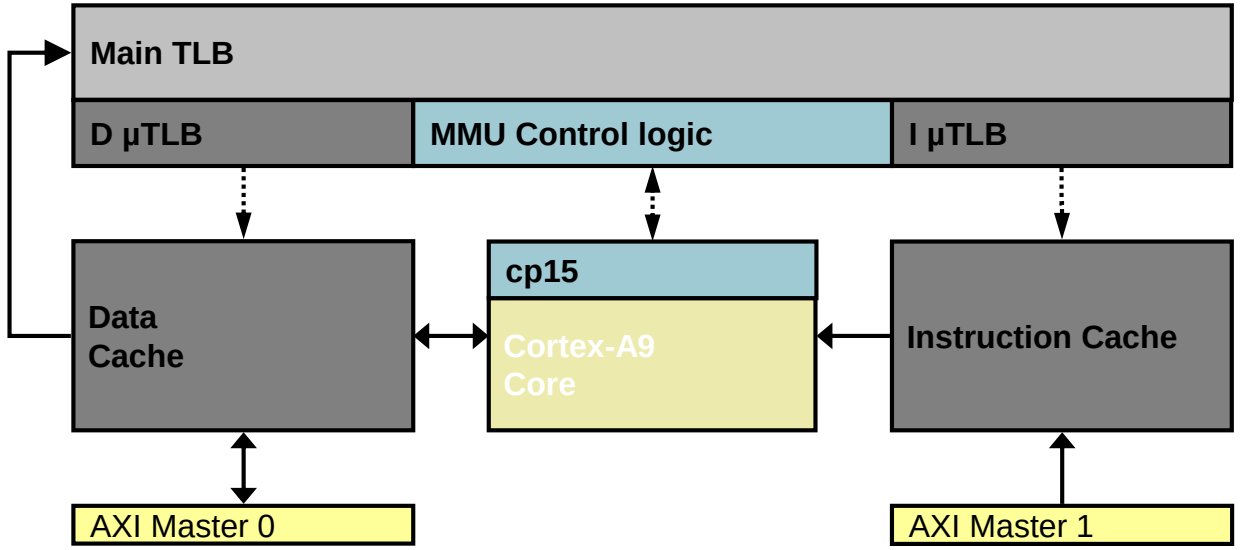
Case Studies

Case Study: ARM Cortex-A9 MMU

- The MMU in the Cortex-A9
  - Works with the L1 and L2 caches for virtual-to-physical address translation
  - Controls access to and from external memory
  - Is based on the Virtual Memory System Architecture from the Armv7-A architecture
  - Checks access permissions and memory attributes
  - Checks the virtual address (VA) and address space identifier (ASID)

Case Study: Cortex-A9 MMU

- Main TLB along with separate micro-TLBs for instructions and data for quick access
- Page-table walks can be configured to go through the L1 data cache.
  - Allows page tables to be cached

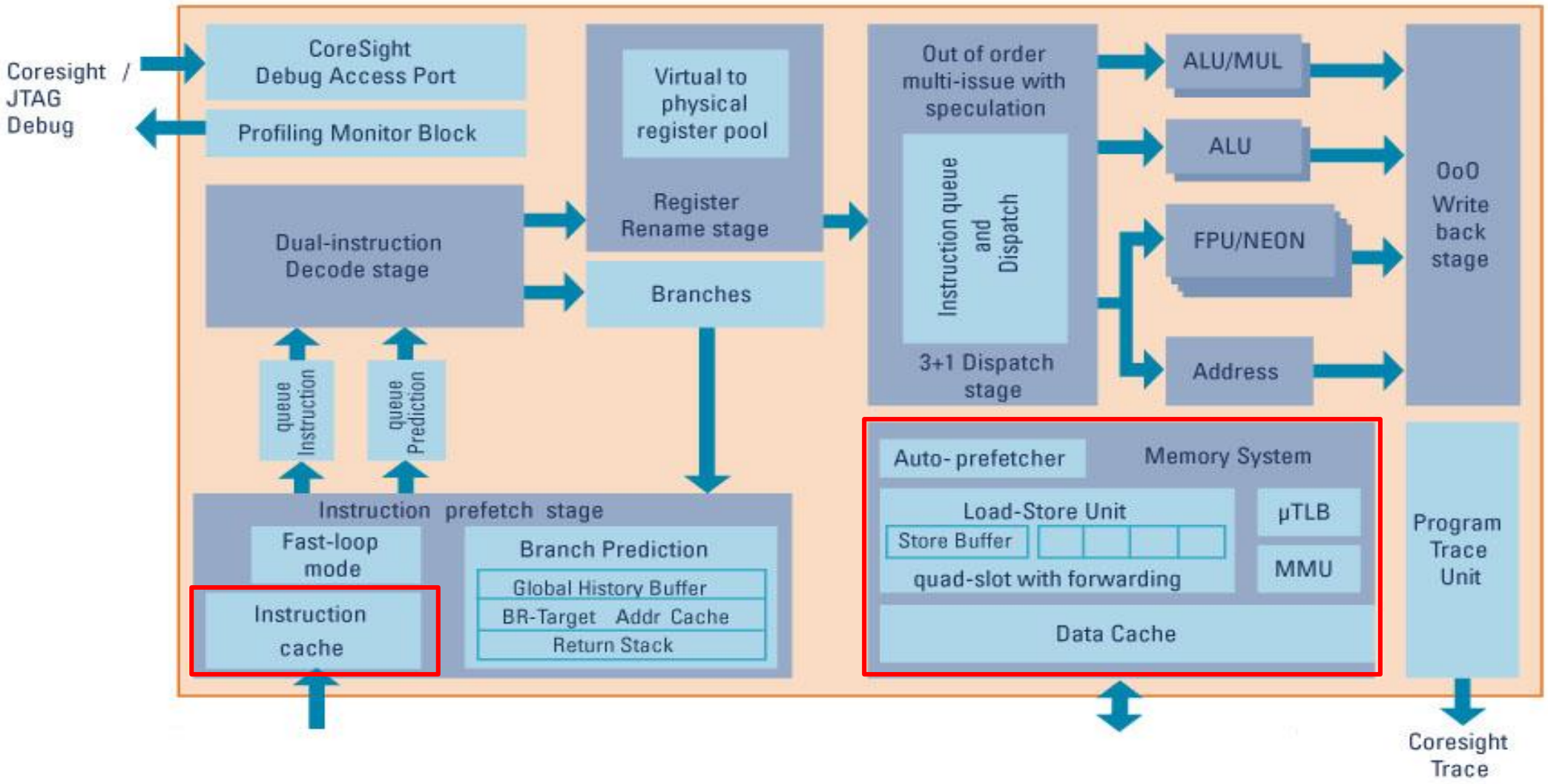


Case Study: Cortex-A9 MMU

- Memory access sequence
  1. MMU performs a lookup for the requested VA and current ASID in the relevant micro-TLB.
  2. If there is a miss in the micro-TLB, look in the main TLB.
  3. If there is a miss in the main TLB, the MMU performs a hardware page-table walk.

- If the MMU finds a matching TLB entry, the MMU
  1. Does permission checks; if these fail, the MMU signals a memory abort
  2. Determines if the access is secure/non-secure, shared/non-shareable, and memory attributes
  3. Performs translation for the memory access

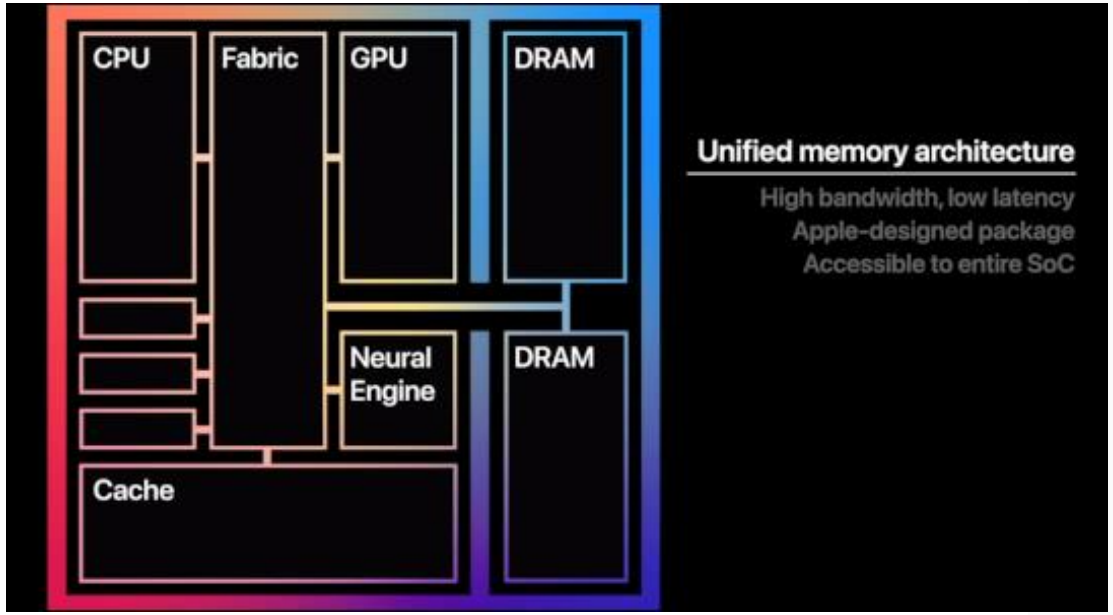
Cortex-A9 Memory System

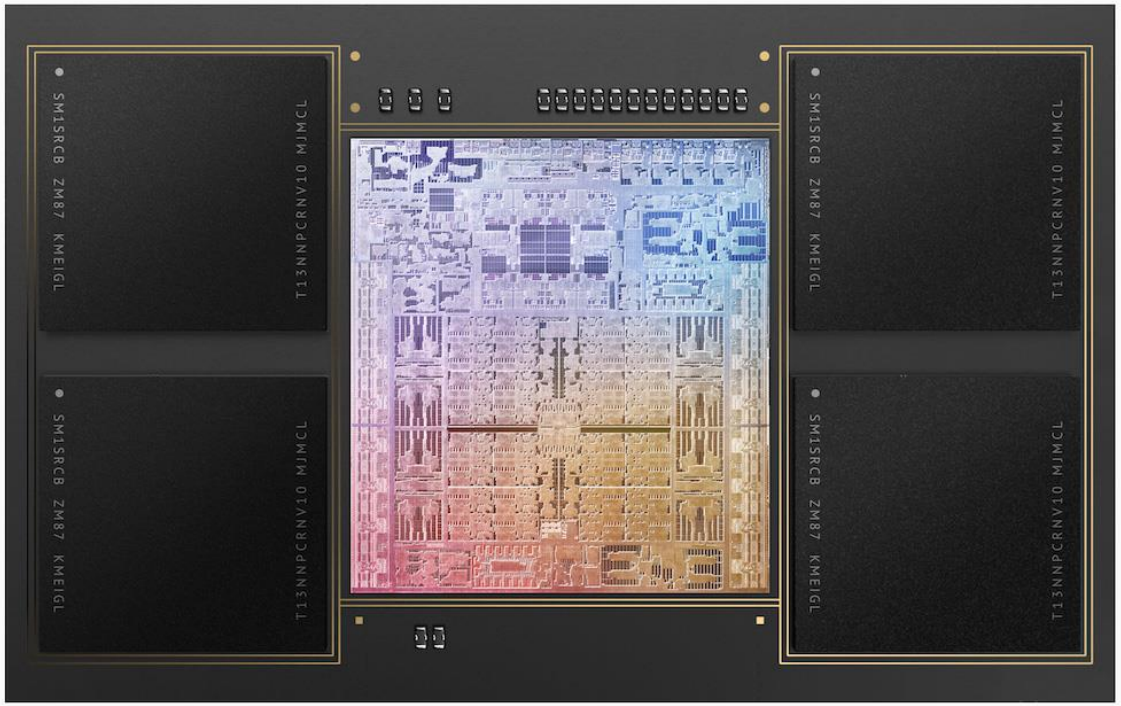


high-performance, low-power, Arm processor

Case Study: Apple M1 Unified Memory

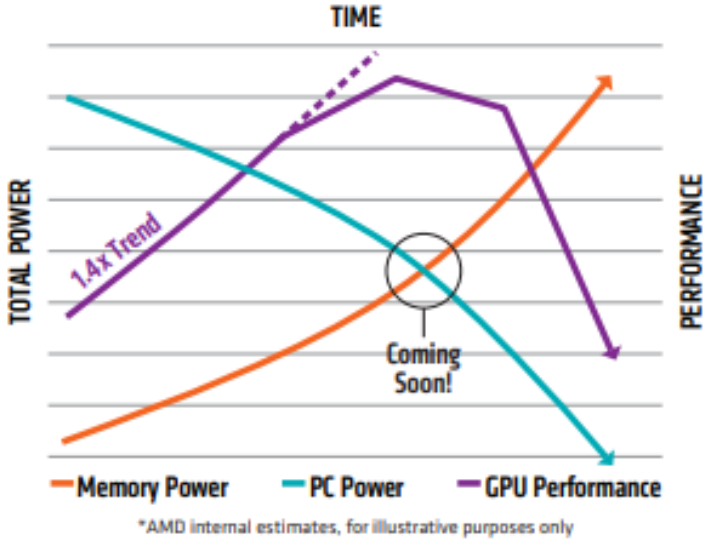
- In Apple's implementation of unified memory, the available RAM is on the M1 system-on-a-chip (SoC). While it's not located within the processor itself, it's still part of the same silicon seated to the side of the other fundamental components.
- UMA, CPU and graphics processor share a common pool of memory. Since the GPU and CPU are using that common pool of memory, the CPU doesn't have to send data anywhere.

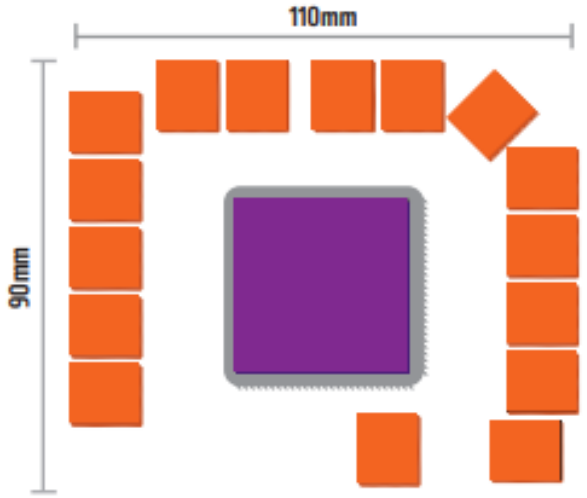


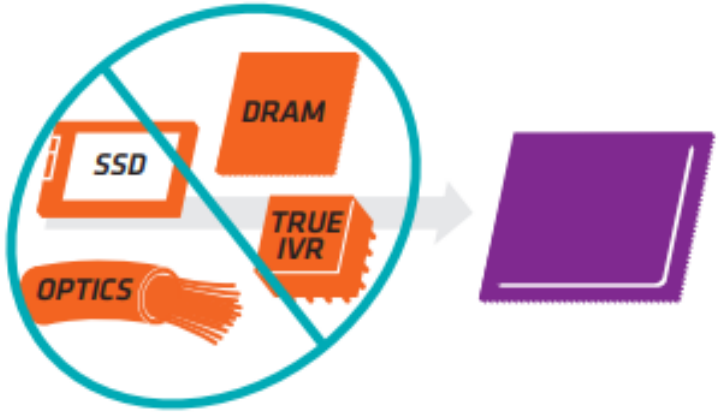


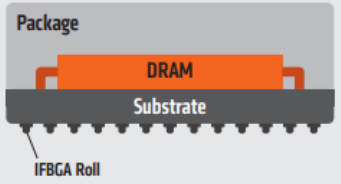
source: Apple

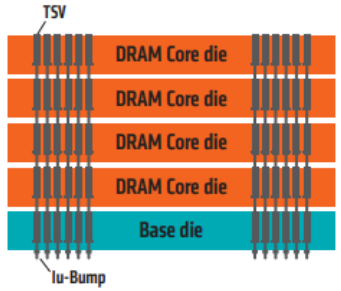
Case Study: AMD HBM











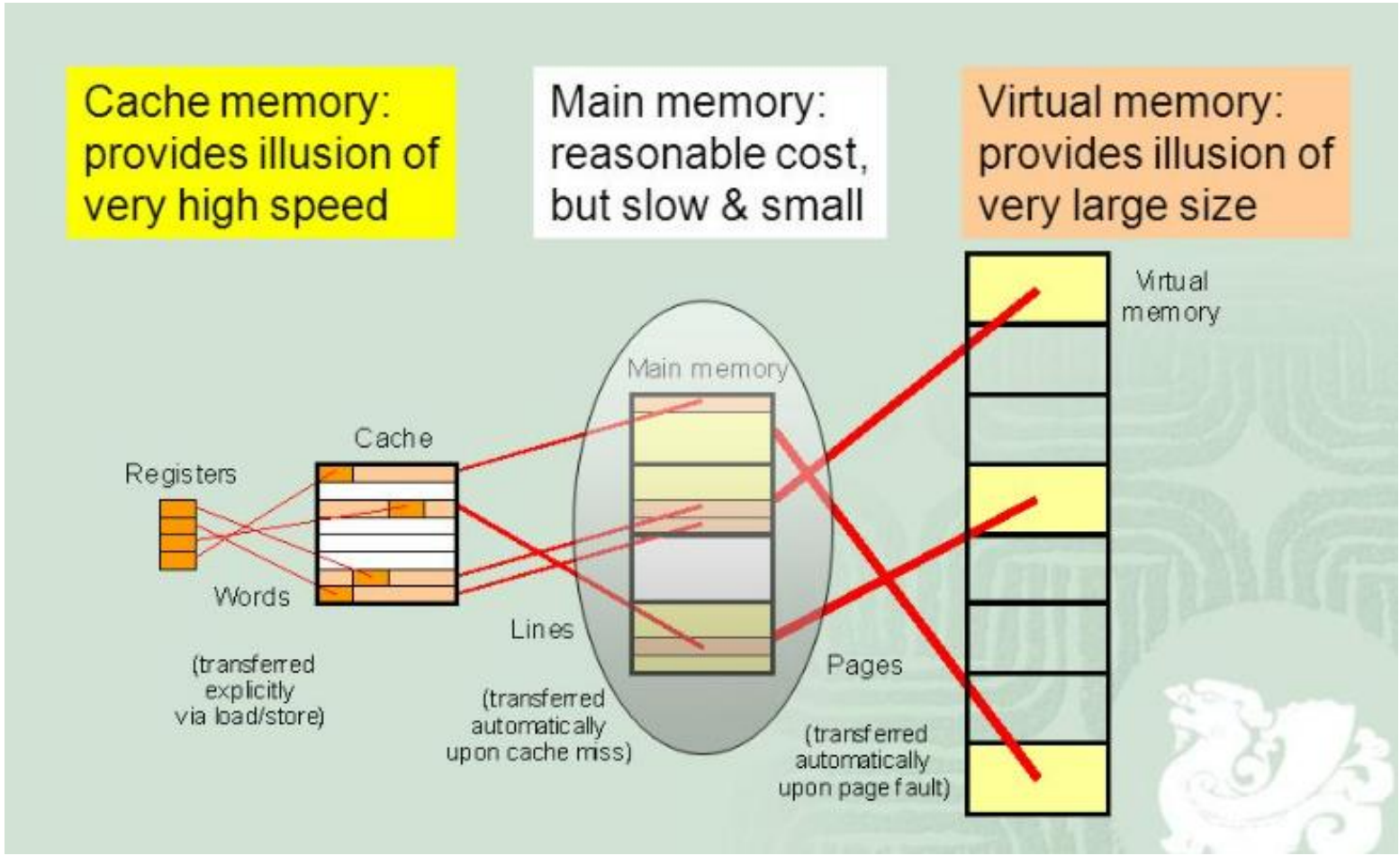
GDDR5	Per Package	HBM
32-bit	Bus Width	1024-bit
Up to 1750MHz (7GBps)	Clock Speed	Up to 500MHz (1GBps)
Up to 28GB/s per chip	Bandwidth	>100GB/s per stack
1.5V	Voltage	1.3V

https://www.amd.com/system/files/documents/high-bandwidth-memory-hbm.pdf

Memory Design

- Computer Architecture is about both processor and memories
- New architectures
  - Memory centric design
  - Process in Memory (Compute in Memory)
  - Near-memory Computing
  - Memory for

Memory Hierarchy Summary

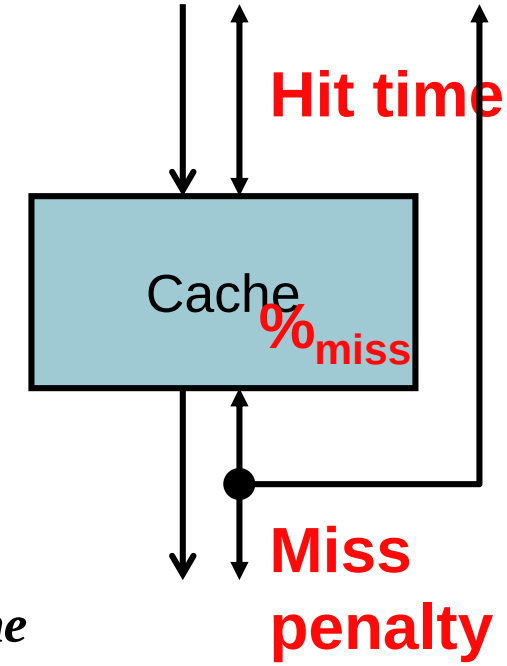


Data Movement in Cache

Cache performance

- CPI_{ALUOps} does not include memory instructions
- AMAT = Average Memory Access Time

$$AMAT = HitTime + MissRate \times MissPenalty$$
$$= (HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst}) +$$
$$(HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data})$$



$$CPUtime = IC \times \left(\frac{AluOps}{Inst} \times CPI_{AluOps} + \frac{MemAccess}{Inst} \times AMAT \right) \times CycleTime$$

Where are we Heading?

- T6: SIMD

Acknowledgement

Slides in this topic are inspired in part by material developed and copyright by:

- ARM Courseware
- Prof. Yonghong Yan @ University of South Carolina
- Prof. Mikko H. Lipasti @ U of Wisconsin Madison
- Prof. Joe Devietti @ Upenn, CIS 571
- Prof. Ron Dreslinski @ UMich, EECS 470
- Prof. Hakim Weatherspoon @ Cornell, CS 3410
- Prof. Krste Asanovic @ UCB, CS252
- Xinfei Guo @ JI, VE370 2021 SU

Action Items

- HW #4 is upcoming
- Lab #4 is out
- Reading Materials
  - Ch. 2.4
  - Ch. Appendix B.4
  - NUCA paper (on canvas, ieee_micro04_nuca.pdf)