Introduction to Computer Architecture

Chapter 5

Large and Fast:
Exploiting Memory Hierarchy - 2

Hyungmin Cho

Department of Computer Science and Engineering
Sungkyunkwan University

Improving Cache Performance

- Associativity
  - Drawback of a direct-mapped cache:
    - If frequently used addresses are mapped to the same cache index, many cache misses will happen!

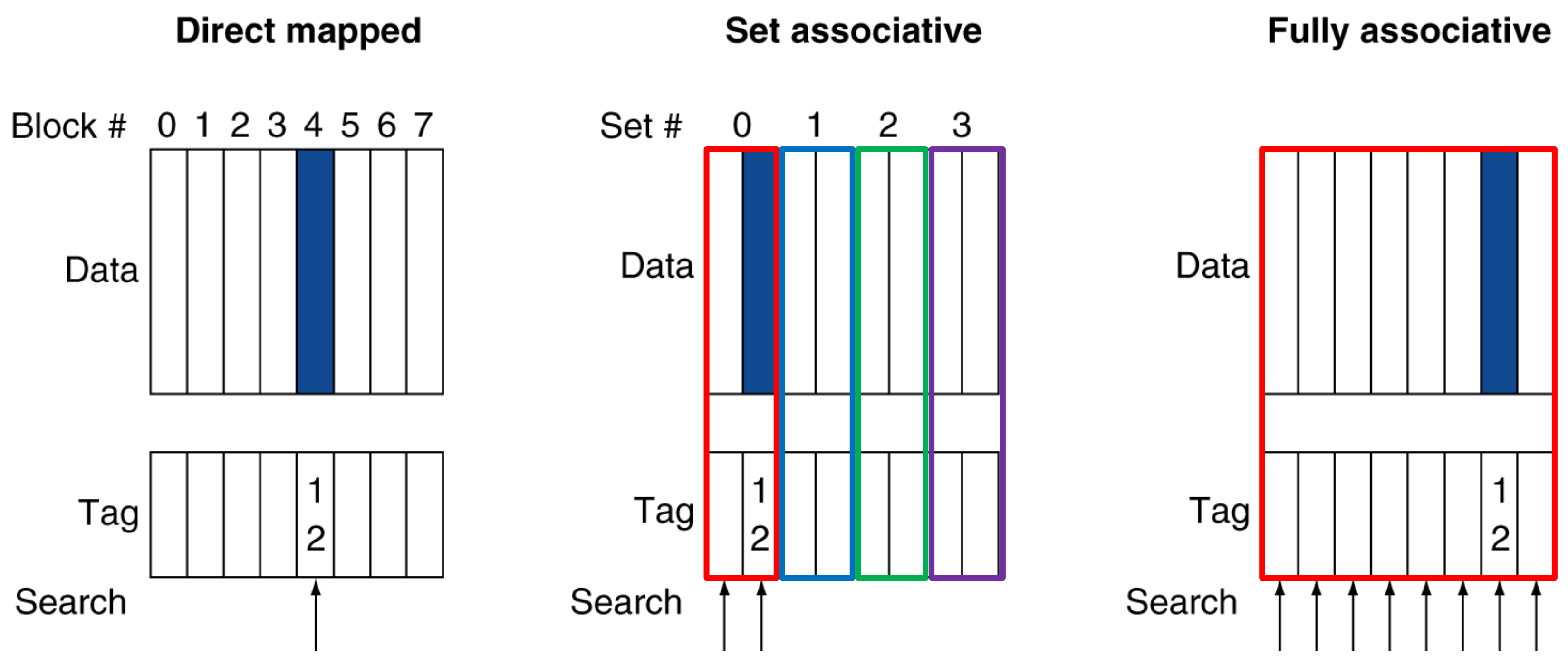
  → Solution: Multiple slots per index!

- Multi-level Cache
  - Small cache has high miss rate
  - Large cache has slow hit time

  → Solution: Use multiple levels of caches!

- Software Optimization
  - Solution: Strategically design data access pattern

Associative Cache Example



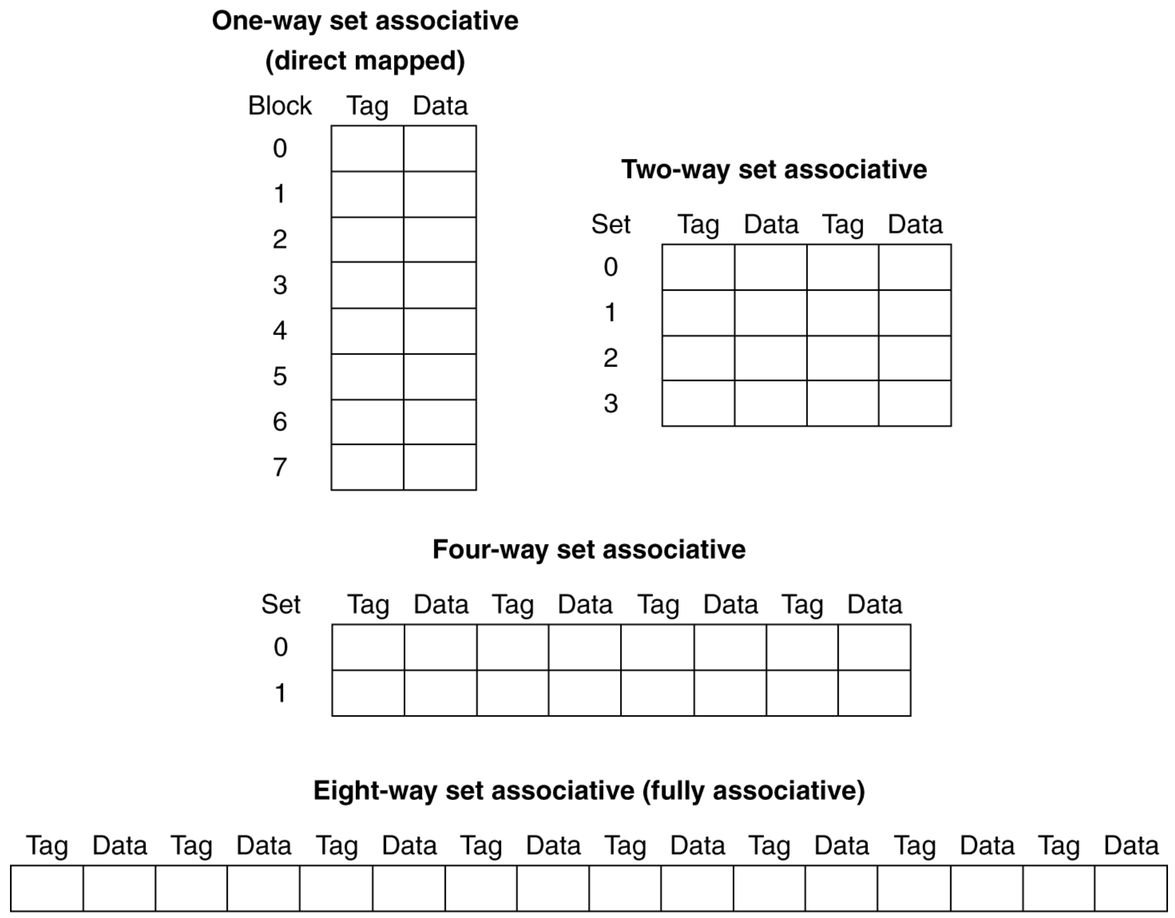
Associative Caches

- Fully associative
  - Allow a given block to be placed anywhere in the cache memory.
  - Requires all locations to be searched at once (Otherwise, very slow..)
  - A comparator for each entry (expensive)

- ***n***-way set associative
  - Each **set** contains ***n*** locations (***n*** blocks)
  - Block number determines the set index
    - `(Block number) modulo (#Sets in cache)`
  - Search all entries **in a given set** at once
  - ***n*** comparators (less expensive)
  - ***n*** may not be a power-of-two

Spectrum of Associativity

- For a cache with 8 block locations



Associativity Example

- Compare 4-block caches
  - Direct mapped / 2-way set associative / fully associative
  - Block address access sequence: `0, 8, 0, 6, 8`

- Direct mapped

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	**Mem[0]**			

Associativity Example

- Fully associative

Block address		Hit/miss	Cache content after access			
0		miss	**Mem[0]**			

Associativity Example

- 2-way set associative

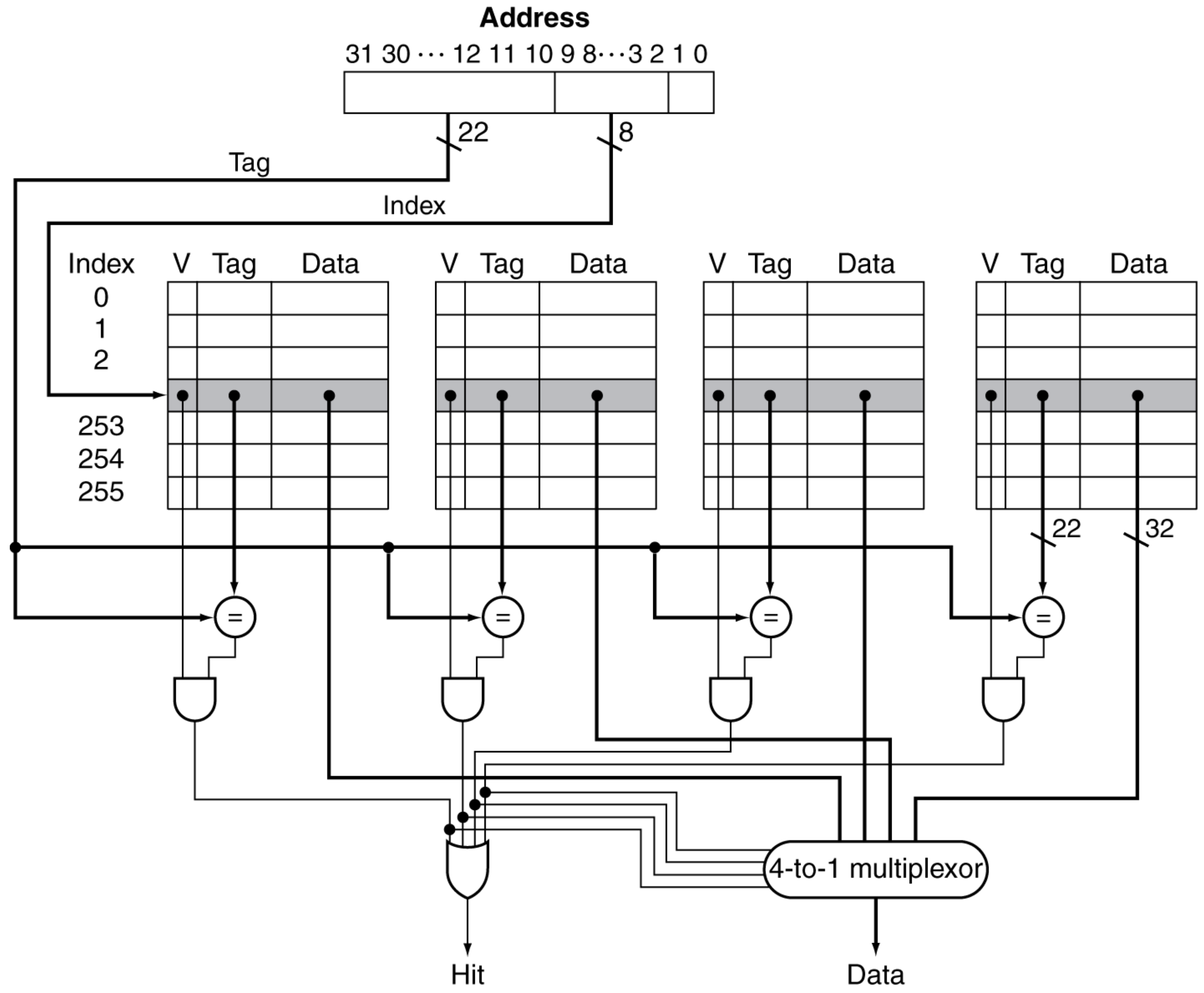
Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	**Mem[0]**			

How Much Associativity

- Increased associativity *generally* decreases miss rate
  - But with diminishing returns

- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

Set Associative Cache Organization



Replacement Policy

- Direct mapped: no choice

- Set associative
  - Choose a non-valid (empty) location, if there is any.
  - Otherwise, need to choose one of the blocks in the set as the **victim**

- The best choice? The block that will be used in the most distant future.
  - Theoretically true, but impossible to know

Replacement Policies for Set Associative 1

- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that

- Pseudo LRU (for 4+ ways)
  - Use an approximation
  - e.g., A tree-like structure that has an LRU bit per 2 entries.

Replacement Policies for Set Associative 2

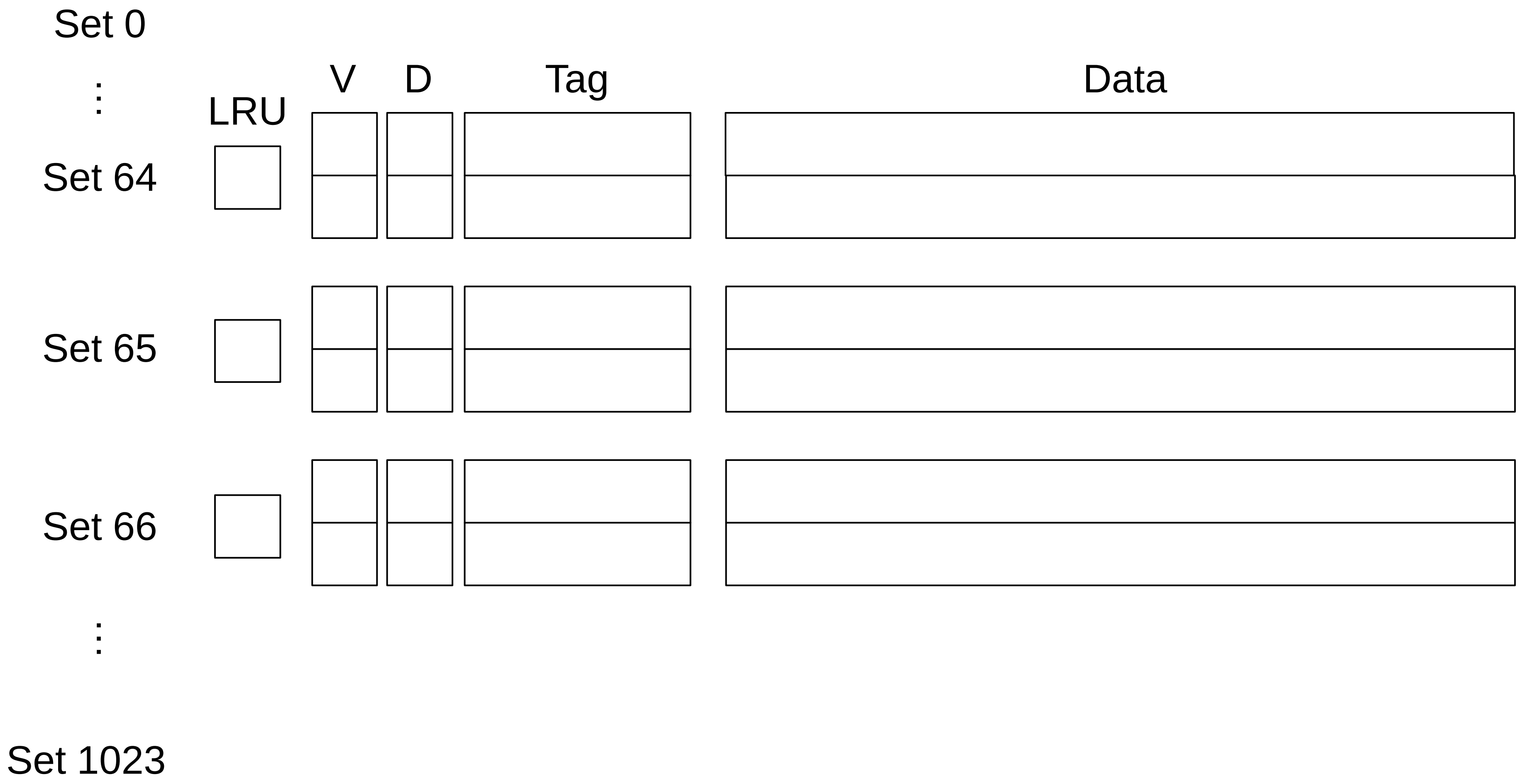
- Sequential (Round Robin)
  - Select the replacement victim in the circular order (i.e., 0→1 →2 →3 →0 →1…)
- Random
  - Similar performance as LRU if the associativity is high enough.

Cache Example

- 128KB data capacity, 64 bytes per block, 32bit address
- 2-way set associative, LRU replacement
- Store policy: write-back, write-allocate.

```
        address
Load  0x20001000
Store 0x36801038
Load  0x20001080
Load  0x20001008
Load  0x40811024
Load  0x53841010
Store 0x53841020
Load  0x20001008
```

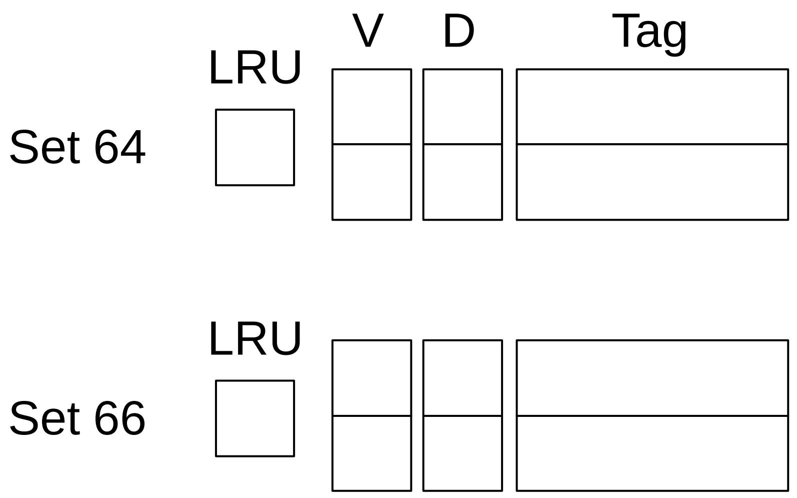
Cache Example



Cache Example

- 128KB data capacity, 64 bytes per block, 32bit address
- 2-way set associative, LRU replacement
- Store policy: write-back, write-allocate.

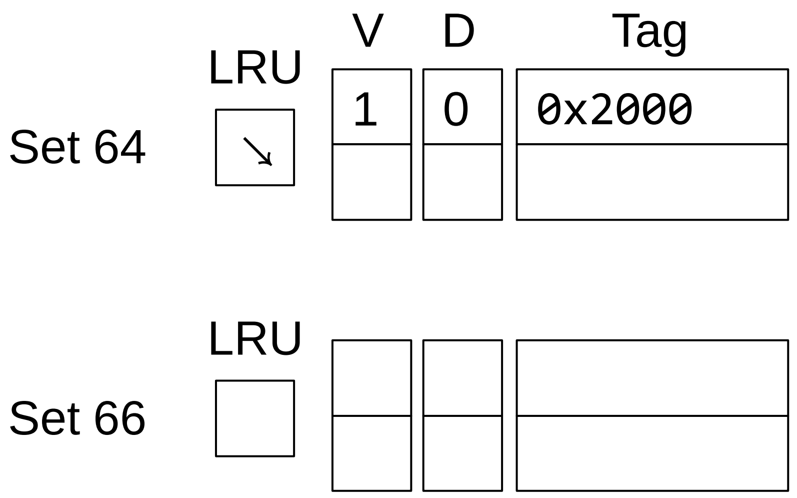
	address	tag	index	offset
Load	0x20001000	0x2000	$0001000000_2 \rightarrow 64_{10}$	000000_2



Cache Example

- 128KB data capacity, 64 bytes per block, 32bit address
- 2-way set associative, LRU replacement
- Store policy: write-back, write-allocate.

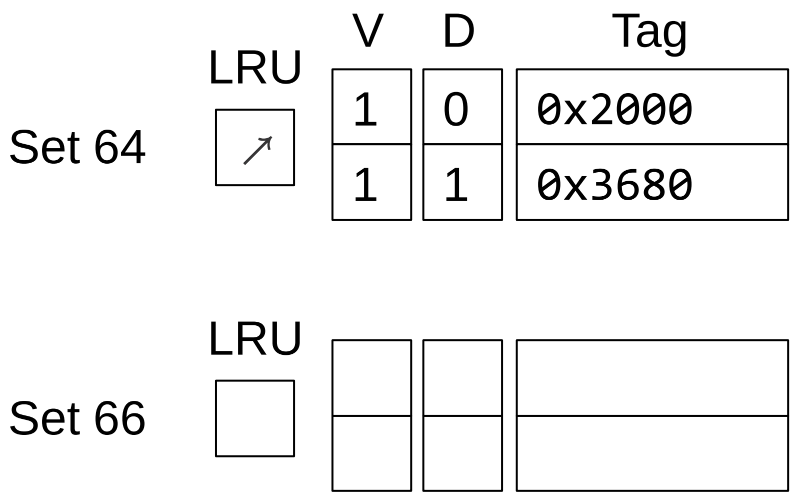
```
             address       tag          index          offset
■  Load   0x20001000    0x2000   0001000000₂→64₁₀    000000₂    miss
```



Cache Example

- 128KB data capacity, 64 bytes per block, 32bit address
- 2-way set associative, LRU replacement
- Store policy: write-back, write-allocate.

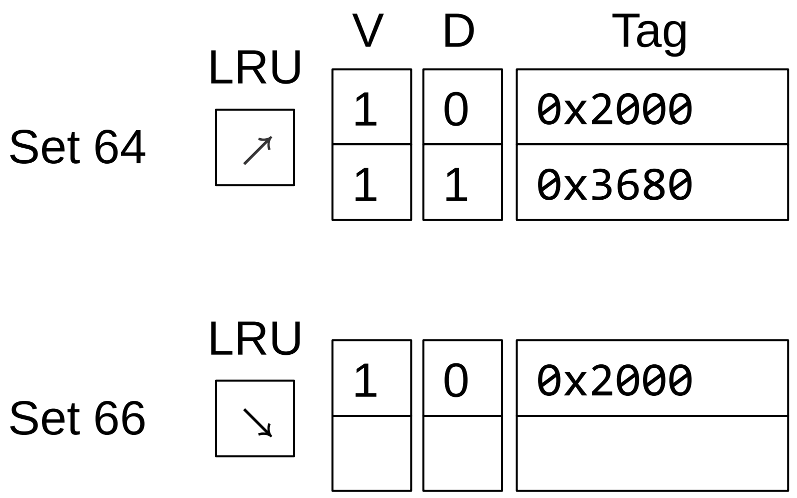
	address	tag	index	offset	
Load	0x20001000	0x2000	$0001000000_2 \rightarrow 64_{10}$	000000_2	miss
Store	0x36801038	0x3680	$0001000000_2 \rightarrow 64_{10}$	111000_2	miss



Cache Example

- 128KB data capacity, 64 bytes per block, 32bit address
- 2-way set associative, LRU replacement
- Store policy: write-back, write-allocate.

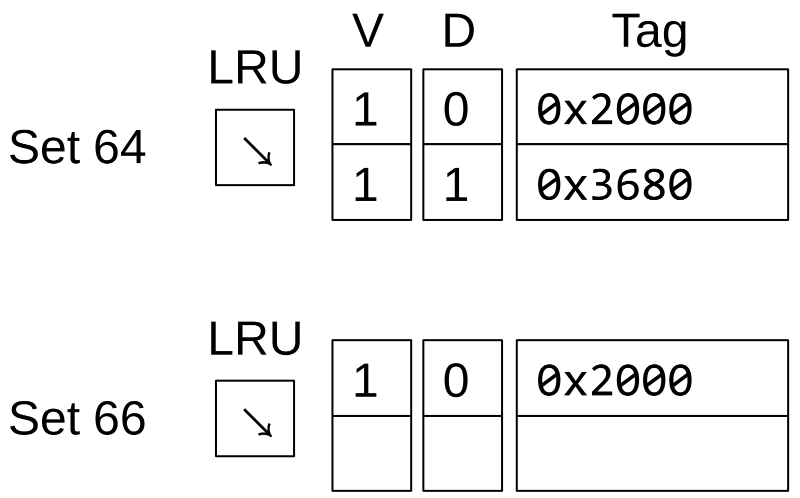
	address	tag	index	offset	
Load	0x20001000	0x2000	$0001000000_2 \rightarrow 64_{10}$	000000_2	miss
Store	0x36801038	0x3680	$0001000000_2 \rightarrow 64_{10}$	111000_2	miss
Load	0x20001080	0x2000	$0001000010_2 \rightarrow 66_{10}$	000000_2	miss



Cache Example

- 128KB data capacity, 64 bytes per block, 32bit address
- 2-way set associative, LRU replacement
- Store policy: write-back, write-allocate.

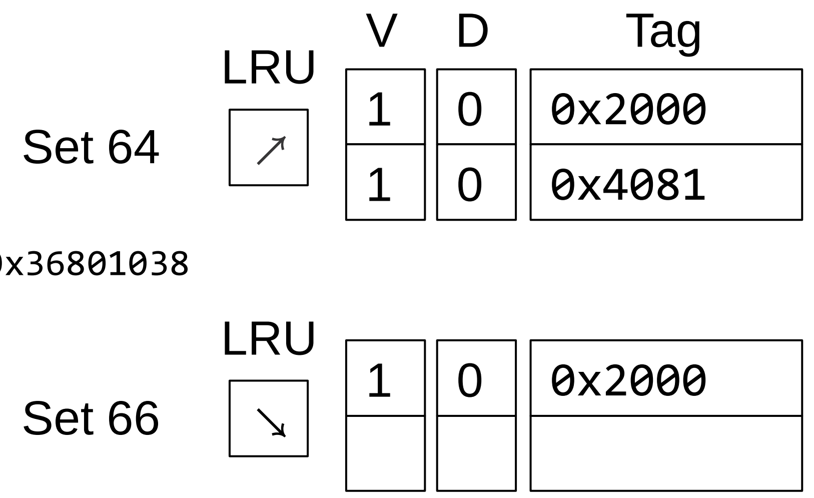
	address	tag	index	offset	
Load	0x20001000	0x2000	$0001000000_2 \rightarrow 64_{10}$	000000_2	miss
Store	0x36801038	0x3680	$0001000000_2 \rightarrow 64_{10}$	111000_2	miss
Load	0x20001080	0x2000	$0001000010_2 \rightarrow 66_{10}$	000000_2	miss
Load	0x20001008	0x2000	$0001000000_2 \rightarrow 64_{10}$	001000_2	hit



Cache Example

- 128KB data capacity, 64 bytes per block, 32bit address
- 2-way set associative, LRU replacement
- Store policy: write-back, write-allocate.

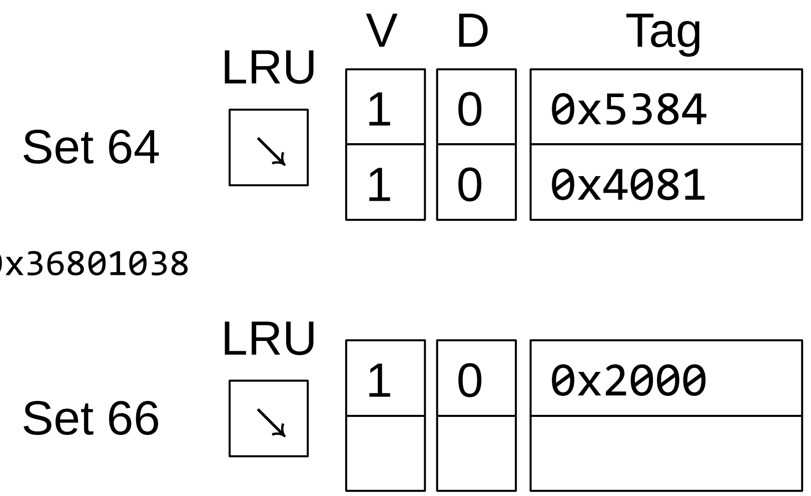
	address	tag	index	offset	
Load	0x20001000	0x2000	$0001000000_2 \rightarrow 64_{10}$	000000_2	miss
Store	0x36801038	0x3680	$0001000000_2 \rightarrow 64_{10}$	111000_2	miss
Load	0x20001080	0x2000	$0001000010_2 \rightarrow 66_{10}$	000000_2	miss
Load	0x20001008	0x2000	$0001000000_2 \rightarrow 64_{10}$	001000_2	hit
Load	0x40811024	0x4081	$0001000000_2 \rightarrow 64_{10}$	100100_2	miss, write back 0x36801038



Cache Example

- 128KB data capacity, 64 bytes per block, 32bit address
- 2-way set associative, LRU replacement
- Store policy: write-back, write-allocate.

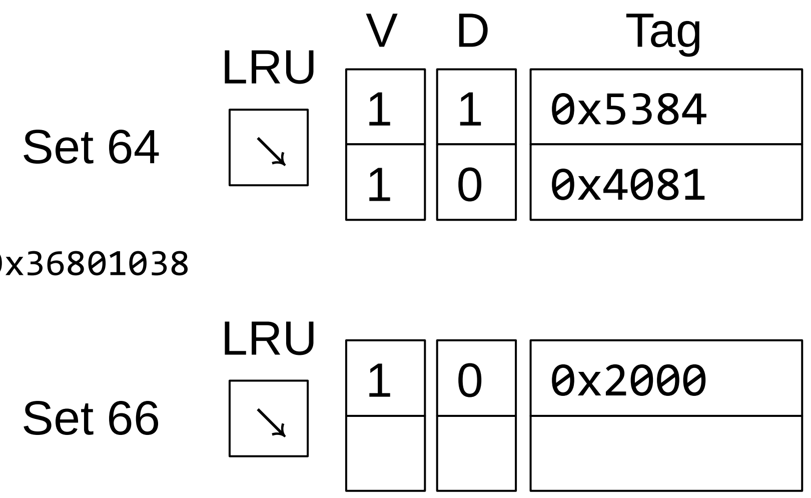
	address	tag	index	offset	
Load	0x20001000	0x2000	$0001000000_2 \rightarrow 64_{10}$	000000_2	miss
Store	0x36801038	0x3680	$0001000000_2 \rightarrow 64_{10}$	111000_2	miss
Load	0x20001080	0x2000	$0001000010_2 \rightarrow 66_{10}$	000000_2	miss
Load	0x20001008	0x2000	$0001000000_2 \rightarrow 64_{10}$	001000_2	hit
Load	0x40811024	0x4081	$0001000000_2 \rightarrow 64_{10}$	100100_2	miss, write back 0x36801038
Load	0x53841010	0x5384	$0001000000_2 \rightarrow 64_{10}$	010000_2	miss



Cache Example

- 128KB data capacity, 64 bytes per block, 32bit address
- 2-way set associative, LRU replacement
- Store policy: write-back, write-allocate.

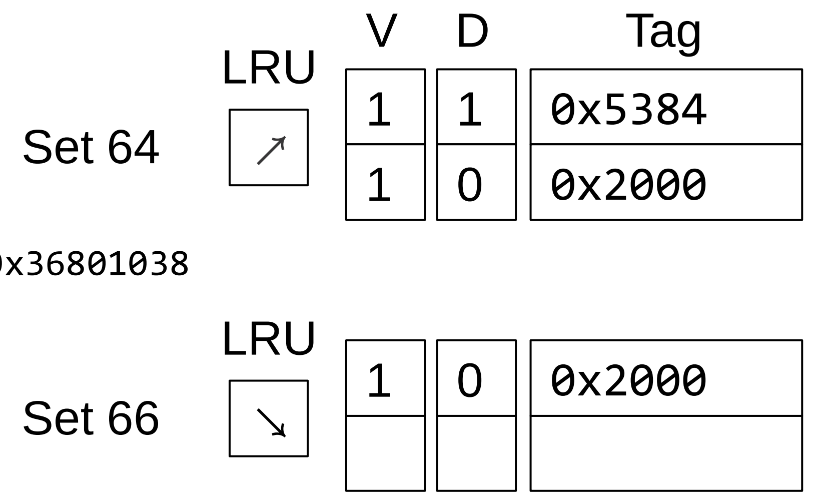
	address	tag	index	offset	
Load	0x20001000	0x2000	$0001000000_2 \rightarrow 64_{10}$	000000_2	miss
Store	0x36801038	0x3680	$0001000000_2 \rightarrow 64_{10}$	111000_2	miss
Load	0x20001080	0x2000	$0001000010_2 \rightarrow 66_{10}$	000000_2	miss
Load	0x20001008	0x2000	$0001000000_2 \rightarrow 64_{10}$	001000_2	hit
Load	0x40811024	0x4081	$0001000000_2 \rightarrow 64_{10}$	100100_2	miss, write back 0x36801038
Load	0x53841010	0x5384	$0001000000_2 \rightarrow 64_{10}$	010000_2	miss
Store	0x53841020	0x5384	$0001000000_2 \rightarrow 64_{10}$	100000_2	hit



Cache Example

- 128KB data capacity, 64 bytes per block, 32bit address
- 2-way set associative, LRU replacement
- Store policy: write-back, write-allocate.

	address	tag	index	offset	
Load	0x20001000	0x2000	$0001000000_2 \rightarrow 64_{10}$	000000_2	miss
Store	0x36801038	0x3680	$0001000000_2 \rightarrow 64_{10}$	111000_2	miss
Load	0x20001080	0x2000	$0001000010_2 \rightarrow 66_{10}$	000000_2	miss
Load	0x20001008	0x2000	$0001000000_2 \rightarrow 64_{10}$	001000_2	hit
Load	0x40811024	0x4081	$0001000000_2 \rightarrow 64_{10}$	100100_2	miss, write back 0x36801038
Load	0x53841010	0x5384	$0001000000_2 \rightarrow 64_{10}$	010000_2	miss
Store	0x53841020	0x5384	$0001000000_2 \rightarrow 64_{10}$	100000_2	hit
Load	0x20001008	0x2000	$0001000000_2 \rightarrow 64_{10}$	001000_2	miss



Multi-level Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache handles misses from the L-1 cache
  - Larger, slower, but still faster than the main memory

- Main memory handles L-2 cache misses
- Some high-end systems include L-3 cache
  - Sometimes called the last-level cache or LLC.

Multi-level Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns

- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1×100% + 400×2% = 9

Example (cont.)

- Now add L-2 cache
  - Access time = 5ns
  - **Global** miss rate to main memory = 0.5%
    - **Local** L2 cache miss rate?

- Primary miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - **Extra** penalty = 400 cycles

- CPI = 1×100% + 20×2% + 400×0.5% = 3.4
- Performance ratio = 9/3.4 = 2.6× faster with L2

Multi-level Cache Considerations

- Primary (L-1) cache
  - Focus on minimal hit time

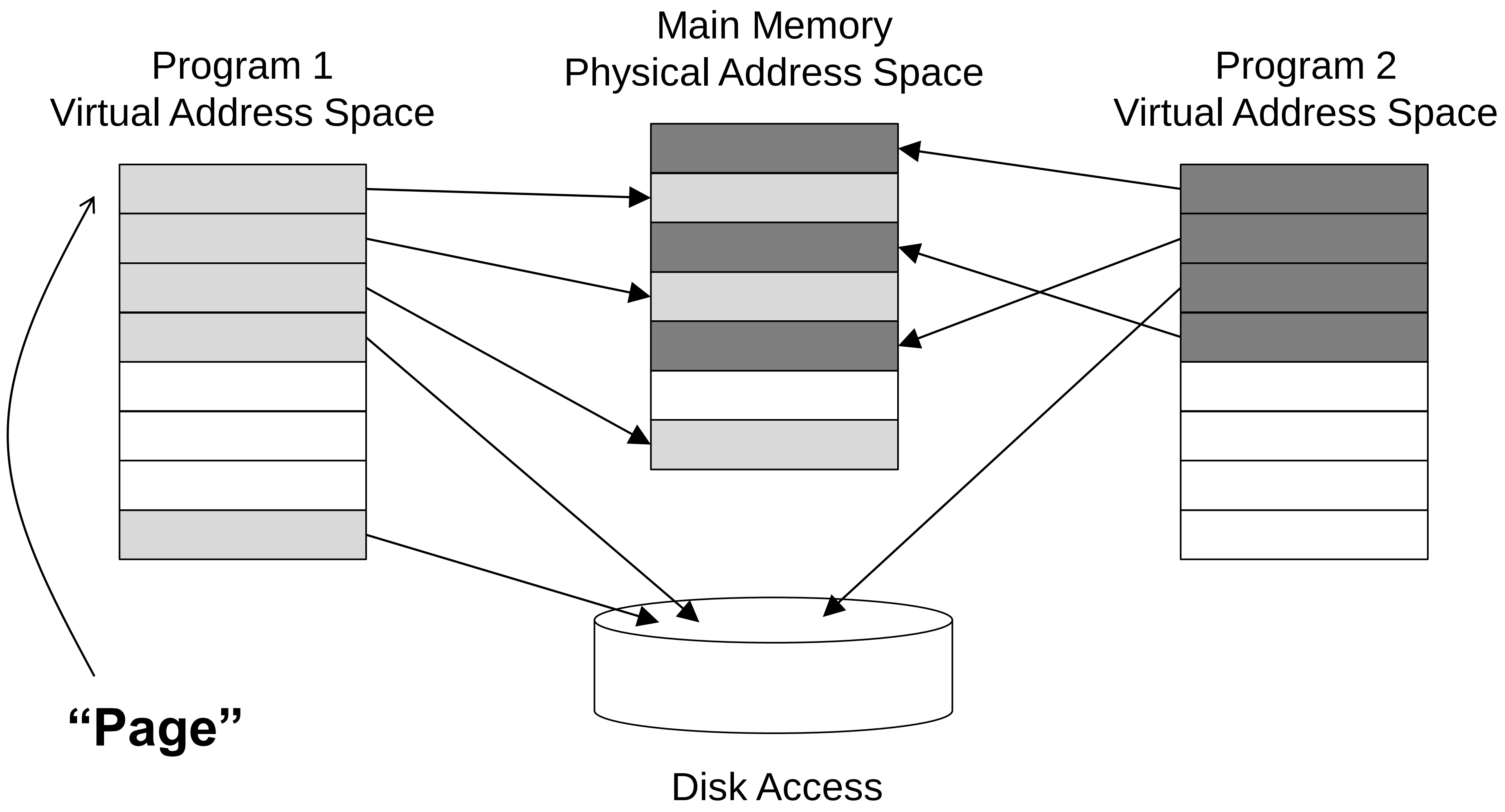
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact

Sources of Misses – The Three C's

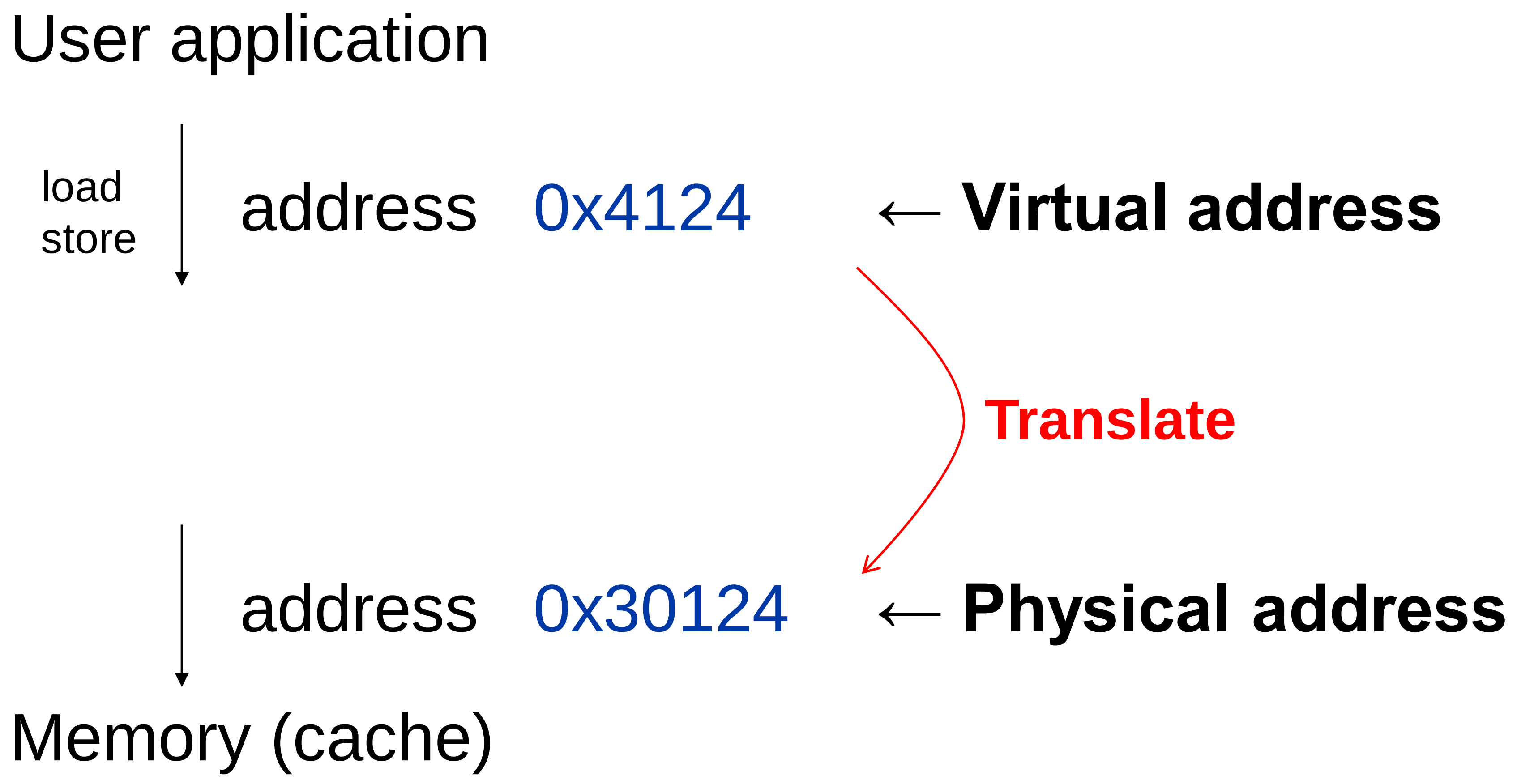
- **Compulsory** misses (a.k.a. cold start misses)
  - First access to a block
- **Capacity** misses
  - Due to finite cache size
  - A replaced block is later accessed again
- **Conflict** misses (a.k.a. collision misses)
  - In a non-fully associative cache
  - Due to competition for entries in a set
  - Do not occur in fully associative caches

VIRTUAL MEMORY

Virtual Memory



Virtual Address



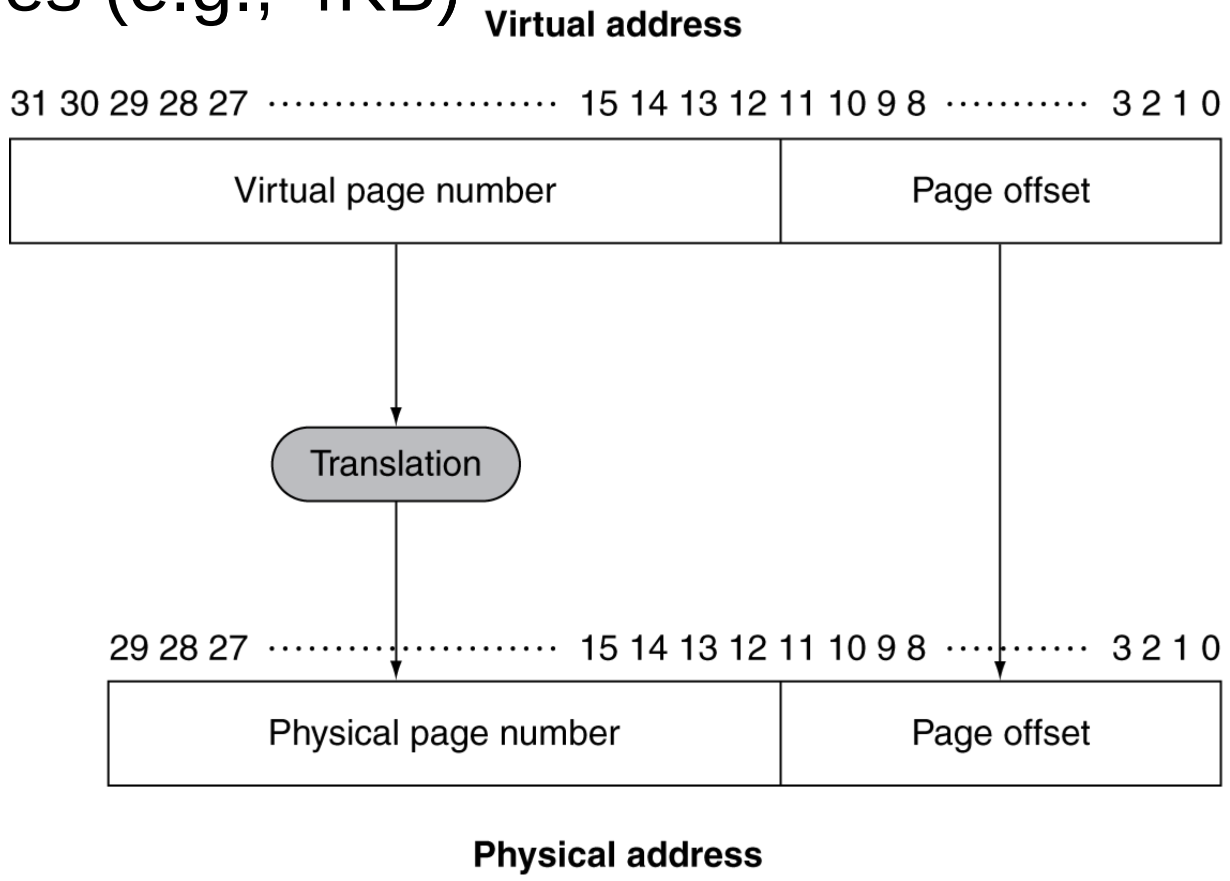
Why Virtual Memory?

- Each program gets a private virtual address space
  - Multiple programs share main memory
  - Do not need to be programmed for a specific memory size

- If there isn’t enough main memory, relocates some data to storage (disk).
  - Managed jointly by CPU hardware and the operating system (OS)

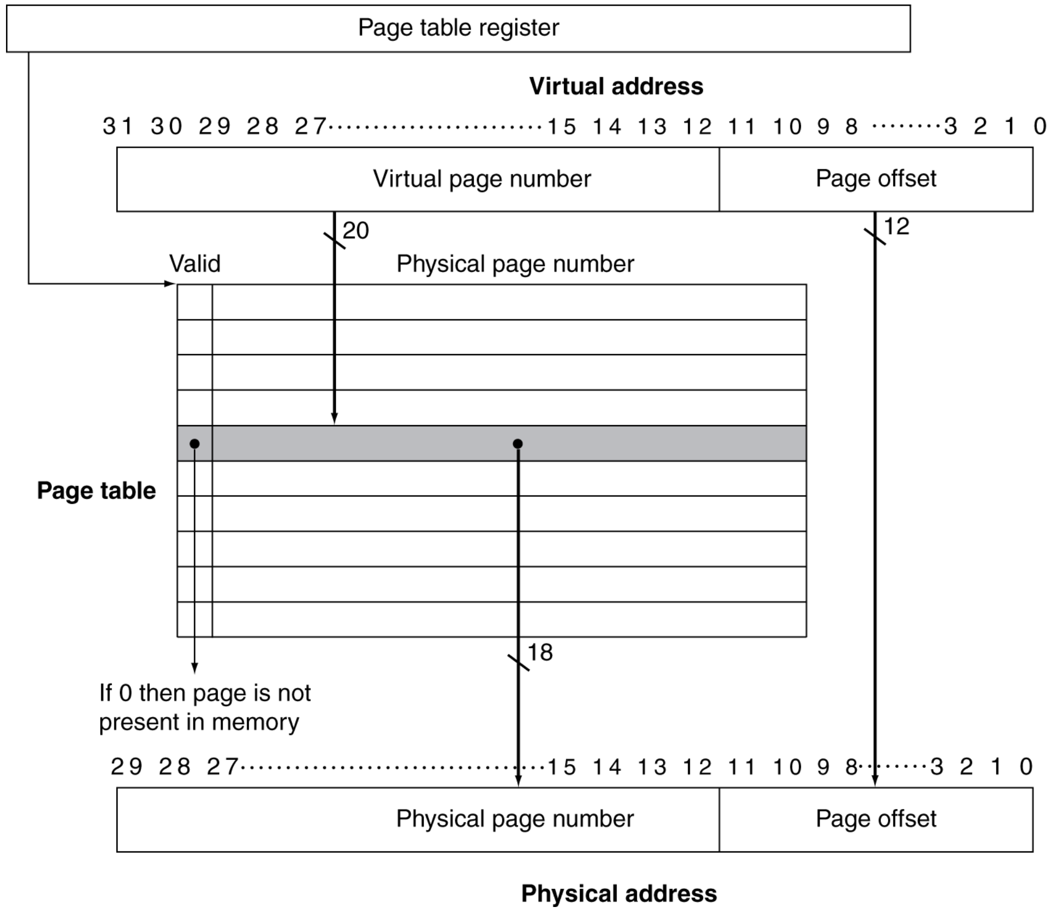
Address Translation

- Fixed-size pages (e.g., 4KB)



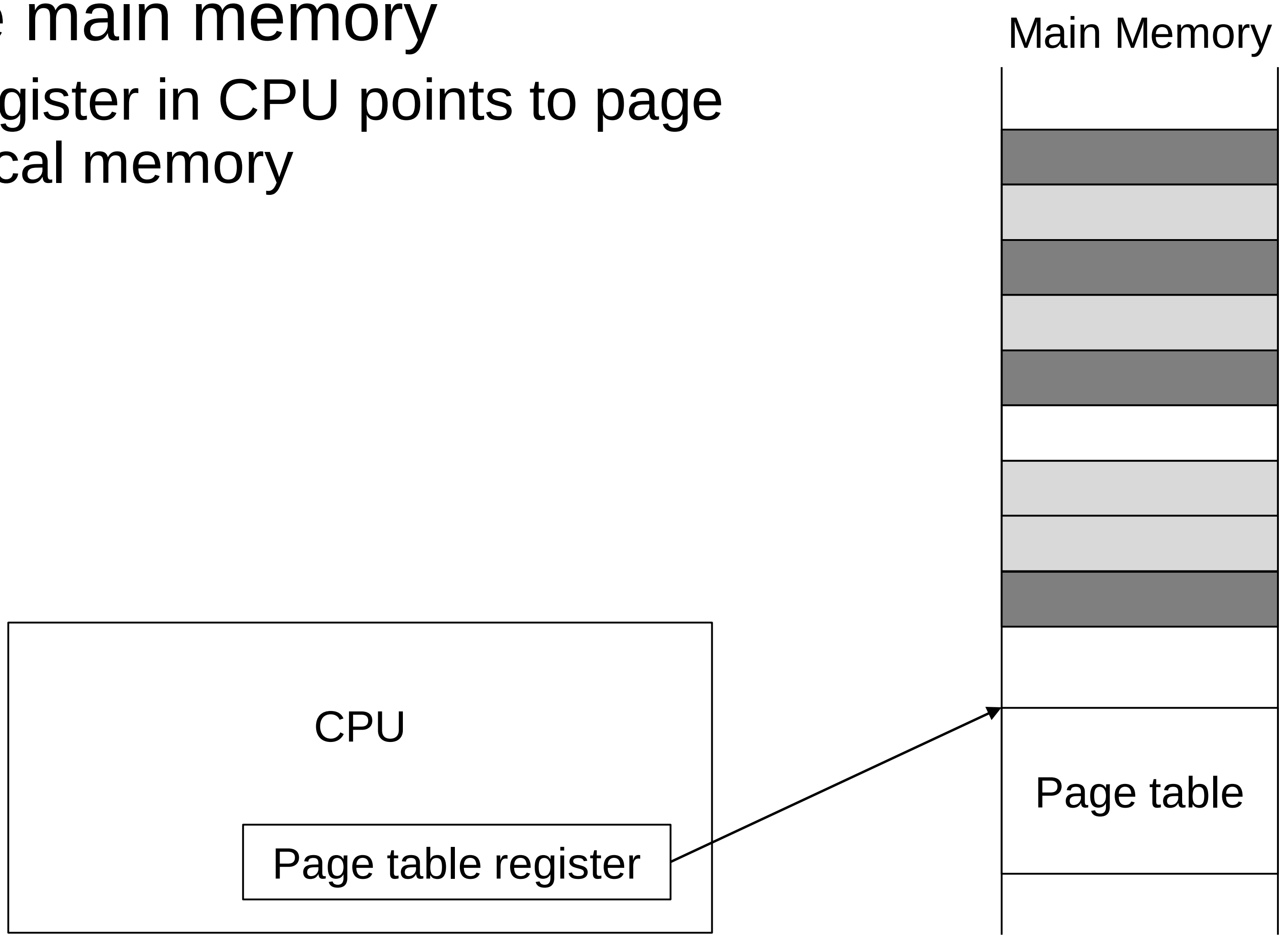
Translation Using a Page Table

- Stores page mapping information
  - Array of **page table entries**
  - Indexed by the virtual page number



Page Table in Memory

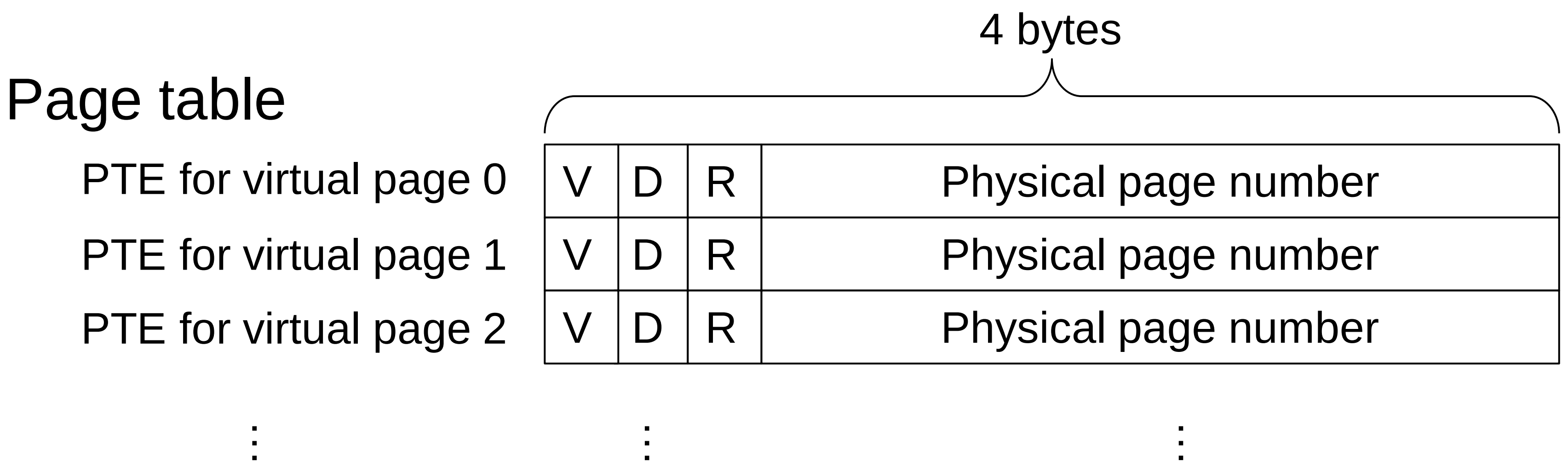
- Resides in the main memory
  - Page table register in CPU points to page table in physical memory



Page Table Entry (PTE)

- If page is present in memory
  - PTE stores the physical page number
  - Plus, other status bits (valid, dirty, referenced, …)

- If page is not present (i.e., valid bit is 0)
  - This means the page is in the disk or not mapped to memory .
  - Accessing the page → **Page fault**



Page Fault Handler

- **Page fault** is handled by OS
  - Locate page on disk
  - Choose a page to replace
    - If dirty, write to disk first
  - Read page into memory and update page table

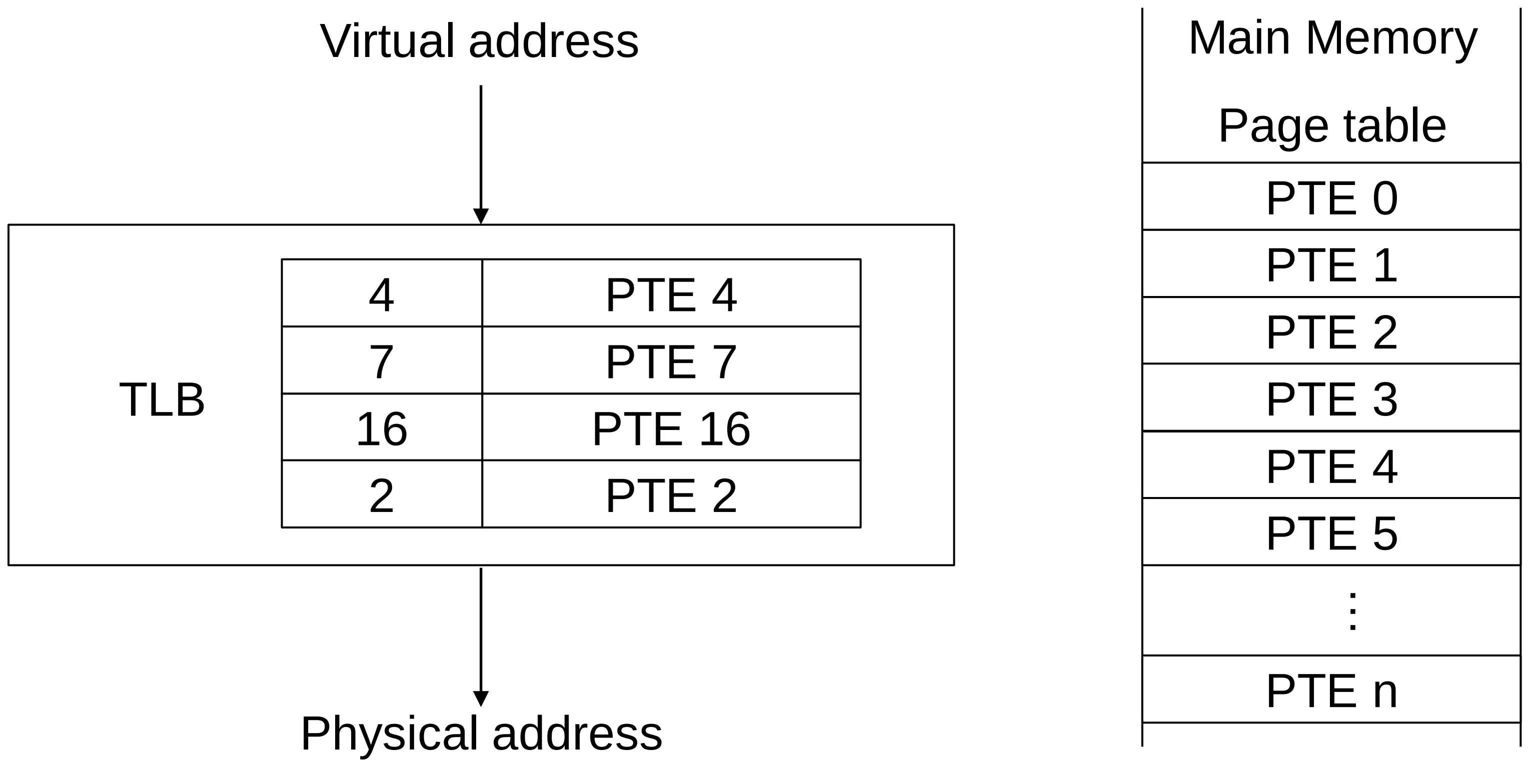
Fast Translation Using a TLB

- Address translation would appear to require extra memory references
  - One to access the PTE
  - Then the actual memory access

- Usually, accesses to page table has good locality pattens
  - Use a fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB)

TLB

- Store frequently accessed PTEs in a faster memory
- Typical: 16–512 PTEs
- Fully associative, n-way set associative, …

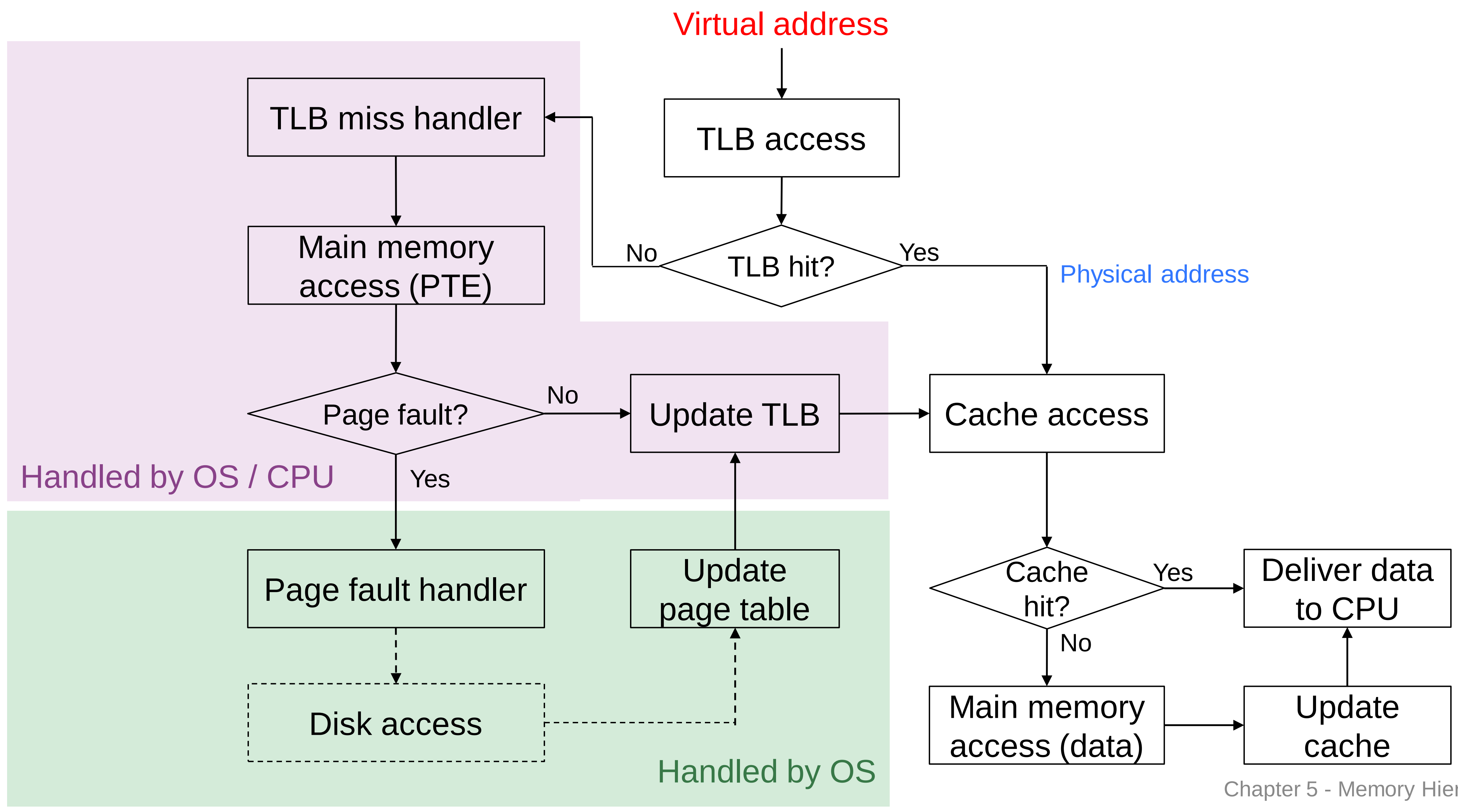


TLB Misses

- TLB miss: The page table entry (PTE) is not in TLB
  - This can be handled by the CPU (hardware TLB management) or by the OS (software TLB management)
    - x86: hardware TLB management
    - MIPS: software TLB management

- If PTE says…
  - The page is in memory
    - Load the PTE from memory to TLB and retry the memory access
  - The page is not in memory (page fault)
    - OS handles fetching the page and updating the page table

Memory Access Scenarios (Data load)


Virtual address
TLB access
TLB hit?
No
Yes
Physical address
TLB miss handler
Main memory access (PTE)
Page fault?
No
Yes
Update TLB
Cache access
Handled by OS / CPU
Page fault handler
Update page table
Disk access
Handled by OS
Cache hit?
Yes
No
Deliver data to CPU
Main memory access (data)
Update cache