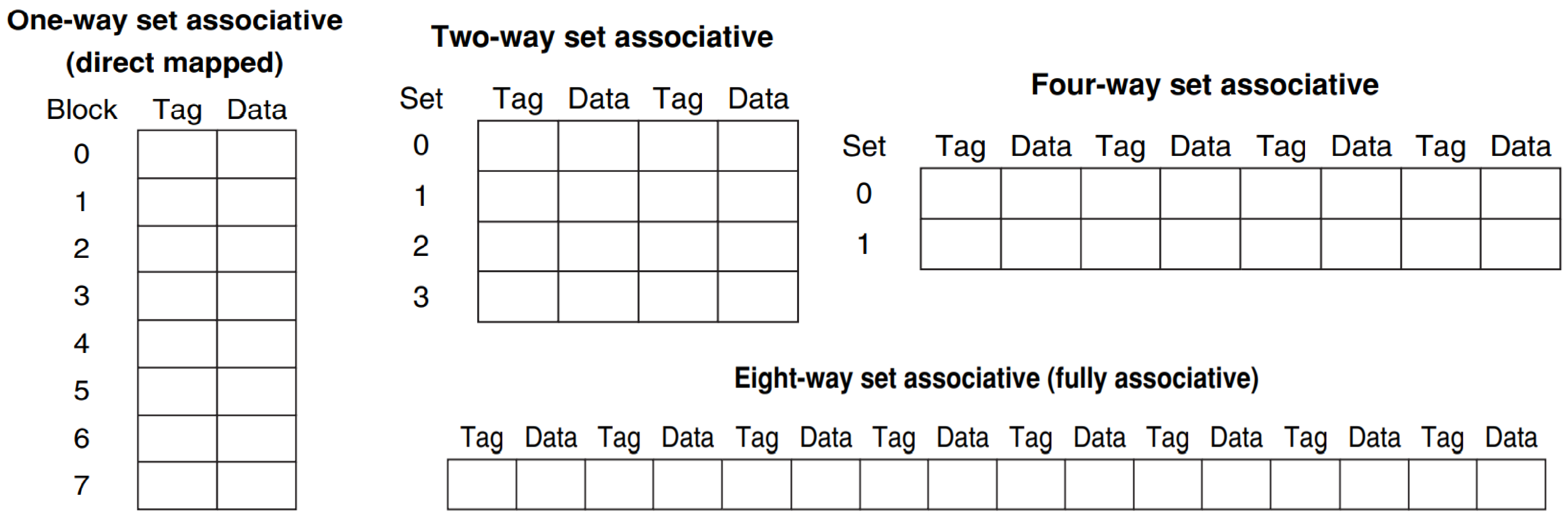
Computer Architecture
(ENE1004)

Lec – 23: Large and Fast: Exploiting Memory Hierarchy (Chapter 5) – 5

Schedule

- Final exam: Jun. 19, Monday
  - (24334) 1:25~2:25pm
  - (23978) 2:30~3:30pm
  - Sample questions will be provided by Jun. 17 (Saturday)
- Assignment #2: Jun. 20, Tue. by midnight
- Remaining class days
  - 12 (Mon) - today
  - 15 (Thur) – short, no attendance check

Set Associative Cache



- Using the same amount of cache blocks (8 blocks), there are different schemes
  - Direct mapped cache can be considered as 1-way set associative cache (1 block in each of 8 sets)
  - 2-way set associative cache (2 blocks in each of 4 sets)
  - 4-way set associative cache (4 blocks in each of 2 sets)
  - Fully associative cache is 8-way set associative cache (8 blocks in a single set)

Misses and Associativity in Caches (1)

- Assume there are three small caches
  - Direct mapped cache vs 2-way set associative cache vs fully associative cache
  - Each consists of four blocks (a total of 4 blocks in each cache)
  - Each block is a single byte (so, there is no byte offset)
- Assume five cache requests with the following addresses are given
  - 0, 8, 0, 6, and 8

(direct mapped)

Block	Tag	Data
0		
1		
2		
3		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				

(fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data

Misses and Associativity in Caches (2) – Direct Mapped

- Assume five cache requests with the following addresses are given
  - 0 (0000) – index/block#: 0 module 4 = 0 (00); tag: 00
  - 8 (1000) – index/block#: 8 module 4 = 0 (00); tag: 10
  - 0 (0000) – index/block#: 8 module 4 = 0 (00); tag: 00
  - 6 (0110) – index/block#: 6 module 4 = 2 (10); tag: 01
  - 8 (1000) – index/block#: 8 module 4 = 0 (00); tag: 10

(direct mapped)

Block	Tag	Data
0		
1		
2		
3		

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Index/block#			
		0	1	2	3
0	miss	Memory[0]			
8	miss	Memory[8]			
0	miss	Memory[0]			
6	miss	Memory[0]		Memory[6]	
8	miss	Memory[8]		Memory[6]	

- All the five requests are misses! (hit ratio = 0; miss ratio = 1)

Misses and Associativity in Caches (3) – 2-way Set Asso.

- Assume five cache requests with the following addresses are given
  - 0 (0000) – Set#: 0 module 2 = 0 (0); tag: 000
  - 8 (1000) – Set#: 8 module 2 = 0 (0); tag: 100
  - 0 (0000) – Set#: 0 module 2 = 0 (0); tag: 000
  - 6 (0110) – Set#: 6 module 2 = 0 (0); tag: 011
  - 8 (1000) – Set#: 8 module 2 = 0 (0); tag: 100

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				

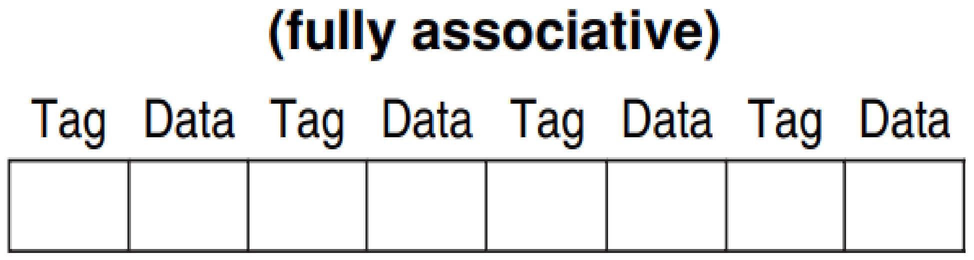
Set#

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Set 0	Set 0	Set 1	Set 1
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[6]		
8	miss	Memory[8]	Memory[6]		

- A total of 4 requests are misses! (hit ratio = 0.2; miss ratio = 0.8)

Misses and Associativity in Caches (4) – Fully Asso.

- Assume five cache requests with the following addresses are given
  - 0 (0000) – Tag: 0000
  - 8 (1000) – Tag: 1000
  - 0 (0000) – Tag: 0000
  - 6 (0110) – Tag: 0110
  - 8 (1000) – Tag: 1000

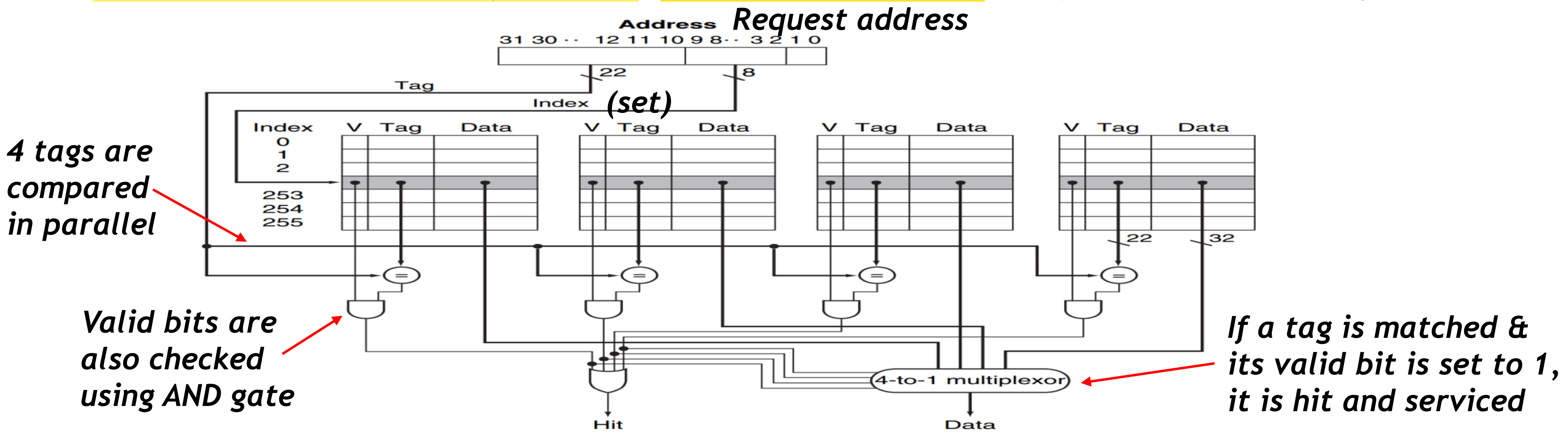


Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	Block 1	Block 2	Block 3
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[8]	Memory[6]	
8	hit	Memory[0]	Memory[8]	Memory[6]	

- A total of 3 requests are misses! (hit ratio = 0.4; miss ratio = 0.6)

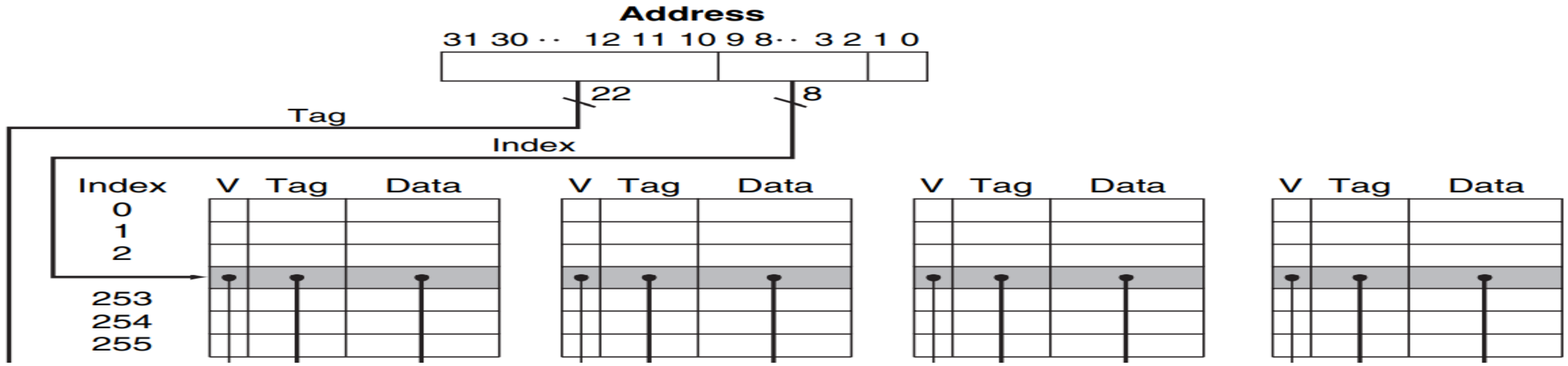
How to Locate a Block in the Cache?

- How can a processor find a data block in a set-associative cache?
  - In a direct-mapped cache, (i) find “index”, and (ii) check its “valid bit” & “tag”
  - In a set-associative cache, (i) find “set”, and
  - (ii) check “valid bit” & “tag” for all the blocks within the set
- In a *n*-way set-associative cache, *n* blocks in a set should be compared
- An example: a 4-way set-associative cache with 1,024 1-word blocks
  - 2 bits for 1-word block’s byte offset, 8 bits for 1024/4 sets, 22 (32-10-2) bits for tag



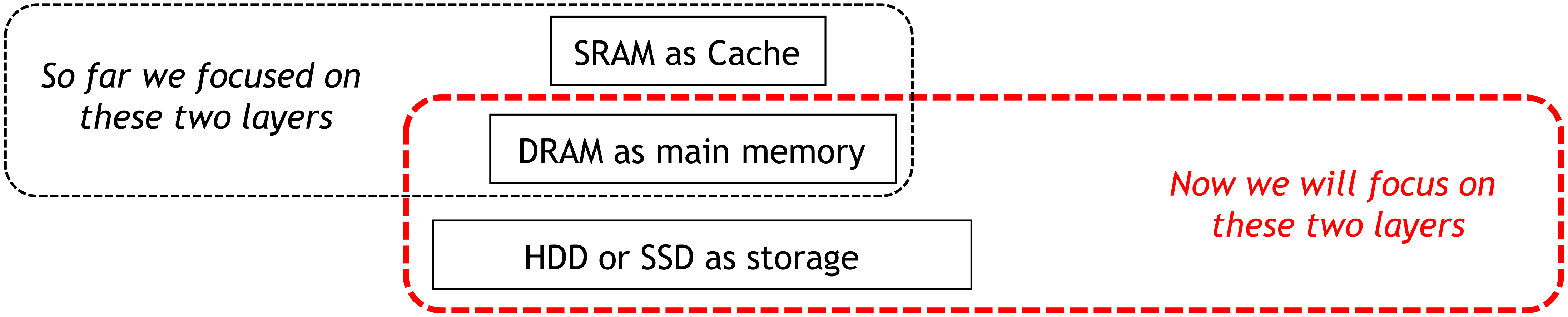
Address
Request address
31 30 · · 12 11 10 9 8 · · 3 2 1 0
22
8
Tag
Index
(set)
Index
0
1
2
253
254
255
V Tag Data
22
32
=
4-to-1 multiplexor
Hit
Data
4 tags are compared in parallel
Valid bits are also checked using AND gate
If a tag is matched & its valid bit is set to 1, it is hit and serviced

Choosing Which Block to Replace for a Miss?



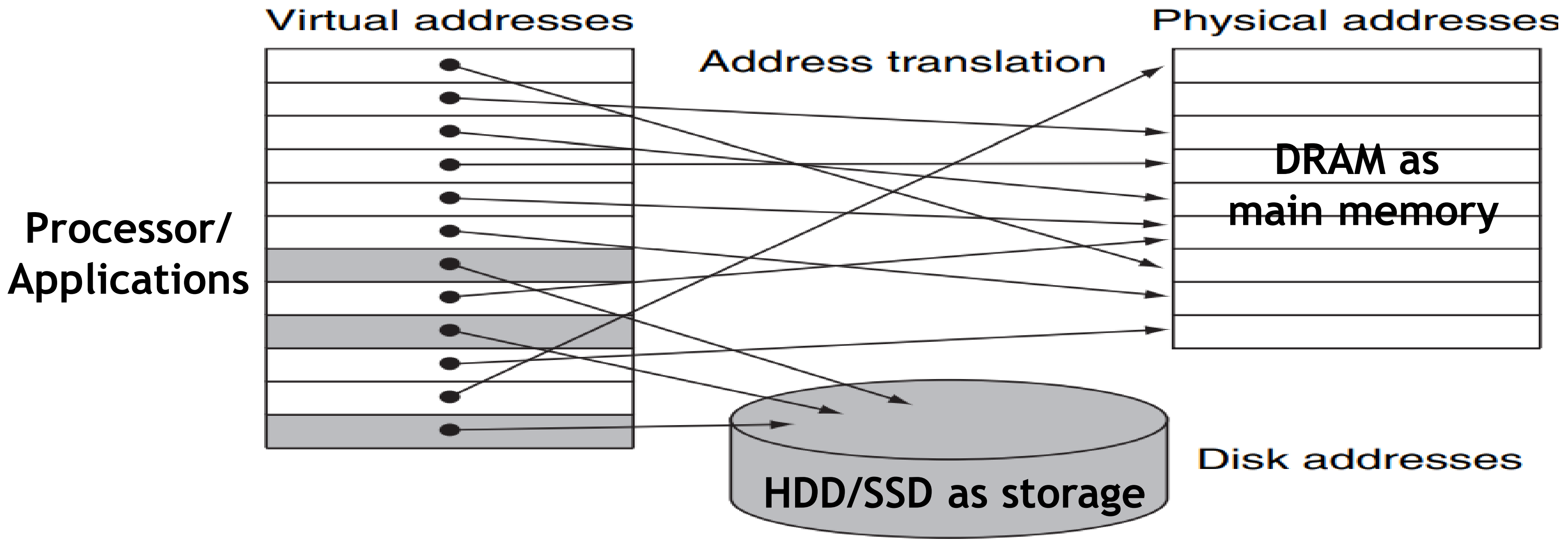
- Assume that, given a request, we found no tag match in the set
  - This is a miss; the requested data block does not exist in the cache
  - The request data block should be brought from the main memory and stored in the set
  - To store the requested block into the set, an existing block should be kicked out from the cache
  - Among the existing four blocks in the set, we must choose one block to replace
- The most commonly used scheme is to pick least-recently-used (LRU) one
  - The block replaced is the one that has been unused for the longest time
  - It does make sense as it may be able to improve the temporal locality
  - See the slide “Misses and Asso … (3) – 2-way Set Asso.”; Memory[8] is selected for Memory[6]

Virtual Memory


SRAM as Cache
DRAM as main memory
HDD or SSD as storage
So far we focused on these two layers
Now we will focus on these two layers

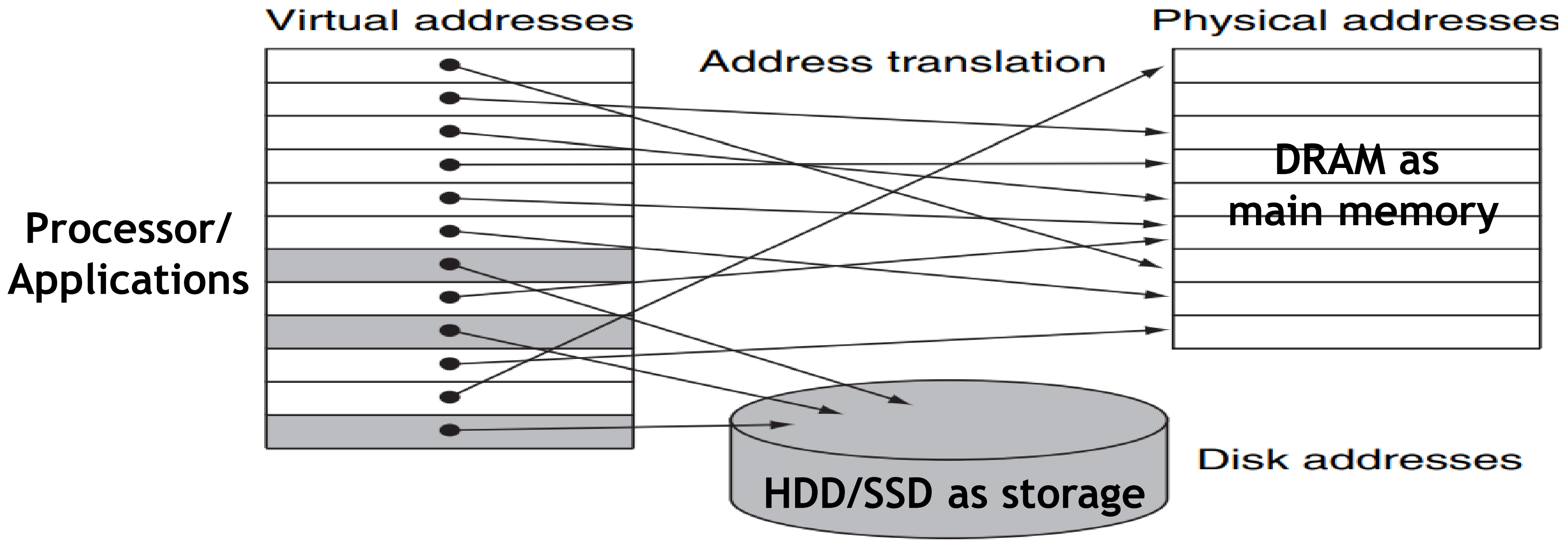
- Now, we will focus on another two-layers memory system: main memory + storage
  - All data sets of applications are stored in the storage
  - The main memory hold part (subset) of all the data sets in the storage
  - A CPU feels like the main memory holds all the data sets
  - So, particularly, we call this system “virtual memory”
- Page: The base unit of the virtual memory is called “page” (vs block/line in cache)
- Page fault: A virtual memory miss is called “page fault”

Virtual Address and Physical Address



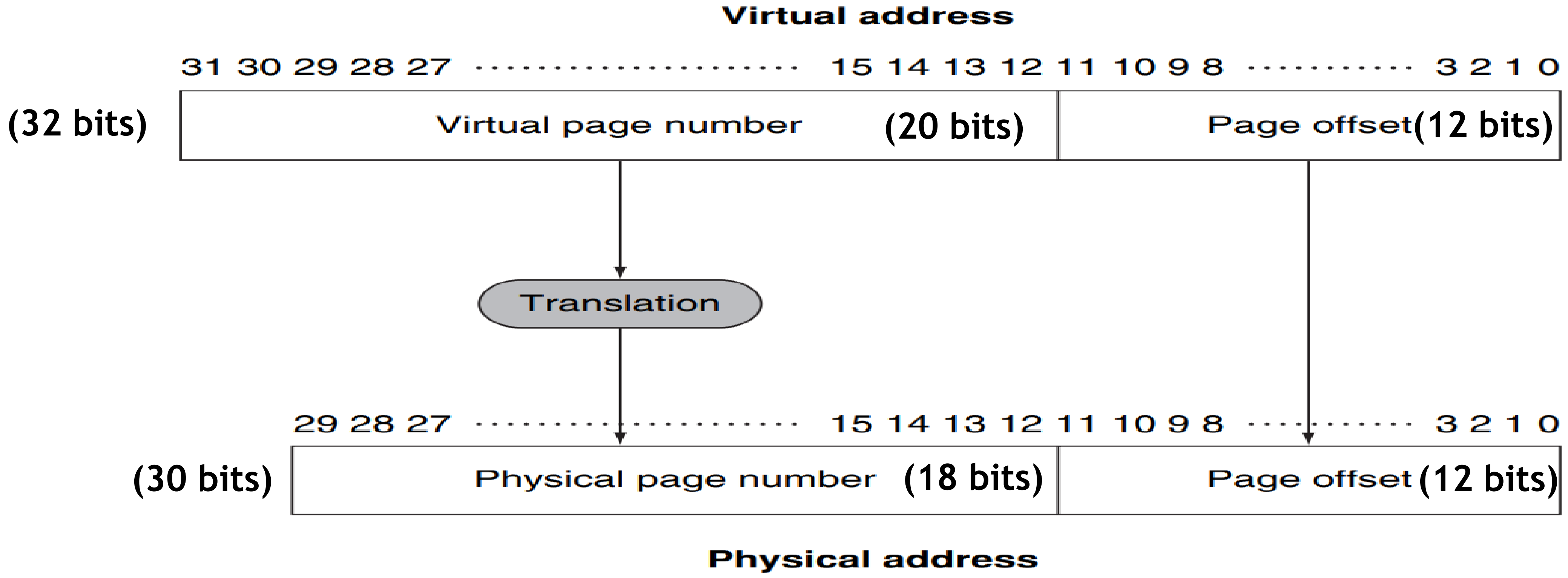
- DRAM can be accessed using its device addresses; we call them "physical addresses"
- However, a CPU does NOT use these physical addresses
  - Instead, the processor uses "virtual addresses"
  - Memory addresses generated in load/store, branch, and jump instructions are virtual addresses
  - This is because all the data of applications cannot be accommodated by limited capacity of DRAM
- Actually, an application running on the CPU has <u>part of its data in the main memory</u> while <u>the remaining data in the storage</u>; but, the CPU does NOT care about it and just uses the virtual addresses of the application

Address Translation: Virtual → Physical Address (1)



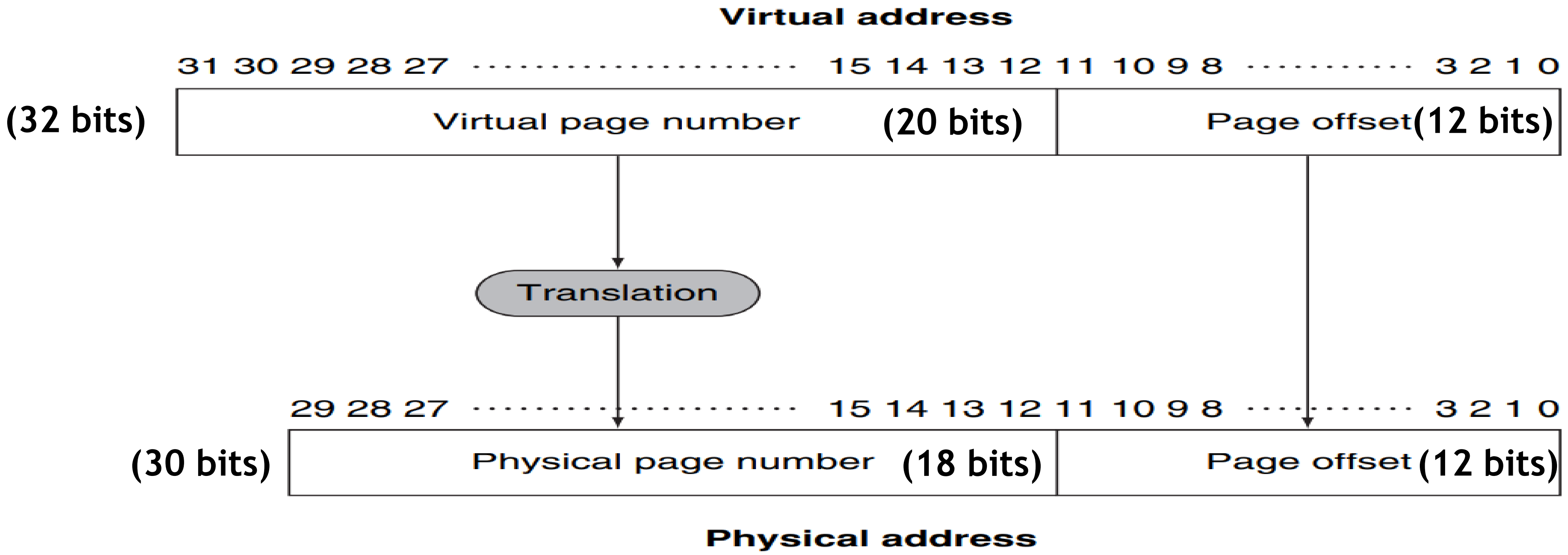
- To request a data in the main memory, an address translation is required
  - Processor requests a data item with its virtual address
  - DRAM can be accessed using its physical address
- Note that a data of an application can be in either the main memory or the storage
  - If the data is in the main memory, the virtual address → the physical address
  - If the data is in the storage, the virtual address → the disk address

Address Translation: Virtual → Physical Address (2)



- Recall that the base unit in the virtual memory is “page”
  - The virtual address for a page can be divided into “virtual page number” + “page offset”
  - The page offset indicates a specific byte within the page (recall “byte offset” in cache)
  - So, we can infer the size of a page from the number of bits for the page offset
  - Here, the lower 12 bits of a 32-bit address is used as page offset; the page size is 2^{12} bytes (4 KB)
- The upper portion of the virtual address indicates “virtual page number”

Address Translation: Virtual → Physical Address (3)



- Address translation: virtual page number → physical page number
  - The page offset does NOT change; it is for specifying bytes within a page
  - In general, # of virtual pages is much larger than # of physical pages (sizes of apps >>> DRAM)
- Here, we can infer the DRAM size from the number of bits in the physical address
  - The physical address is 30-bit (2^{30} bytes = 1 GB)
  - There are 2^{18} pages, each of which is 2^{12} bytes, in this 1 GB DRAM