Georgia Tech Computer Science

CS2200
Systems and Networks
Spring 2022

Lecture 18: Memory Hierarchy

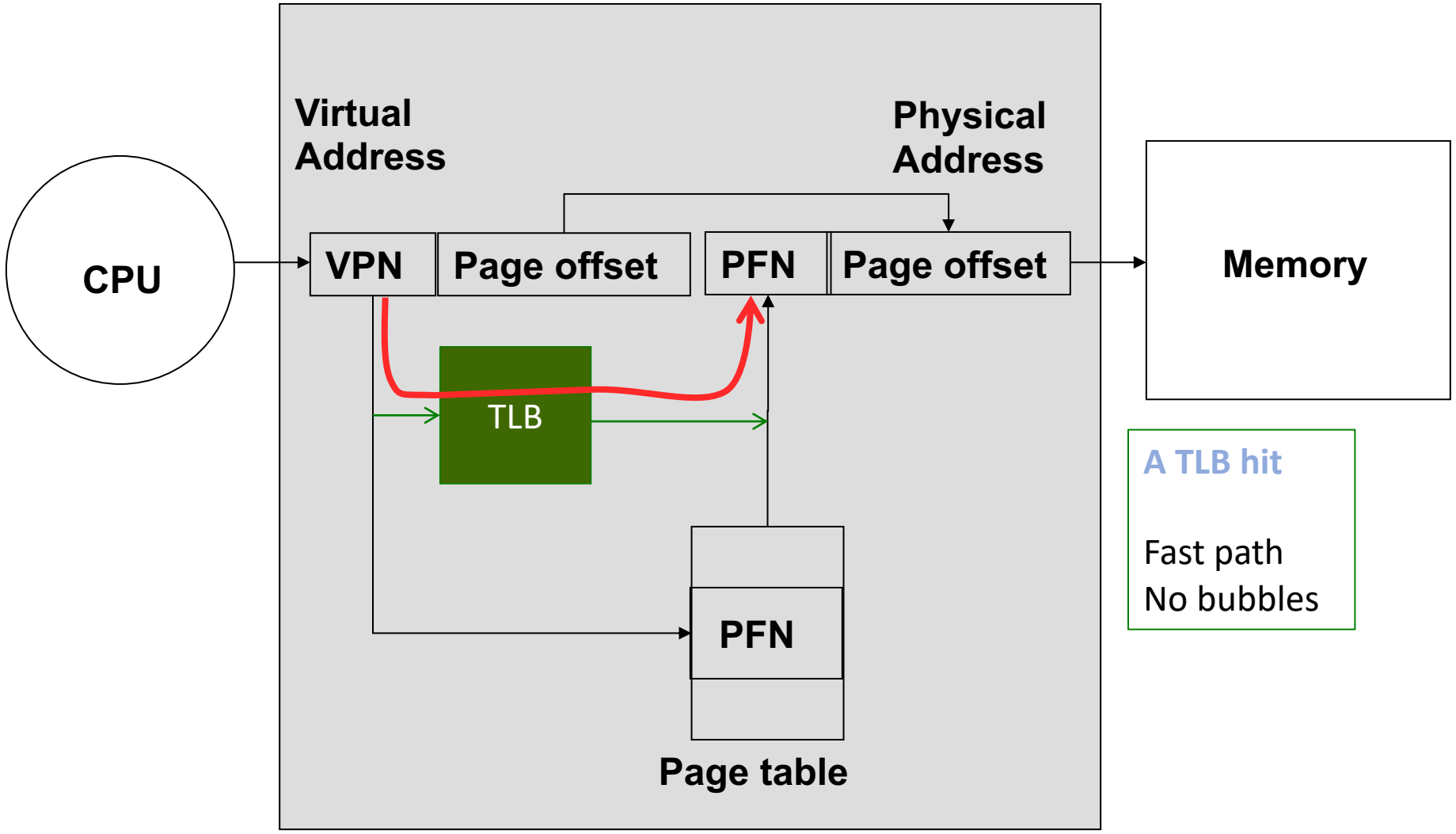
Alexandros (Alex) Daglis
School of Computer Science
Georgia Institute of Technology
adaglis@gatech.edu

Lecture slides adapted from Bill Leahy and Charles Lively of Georgia Tech

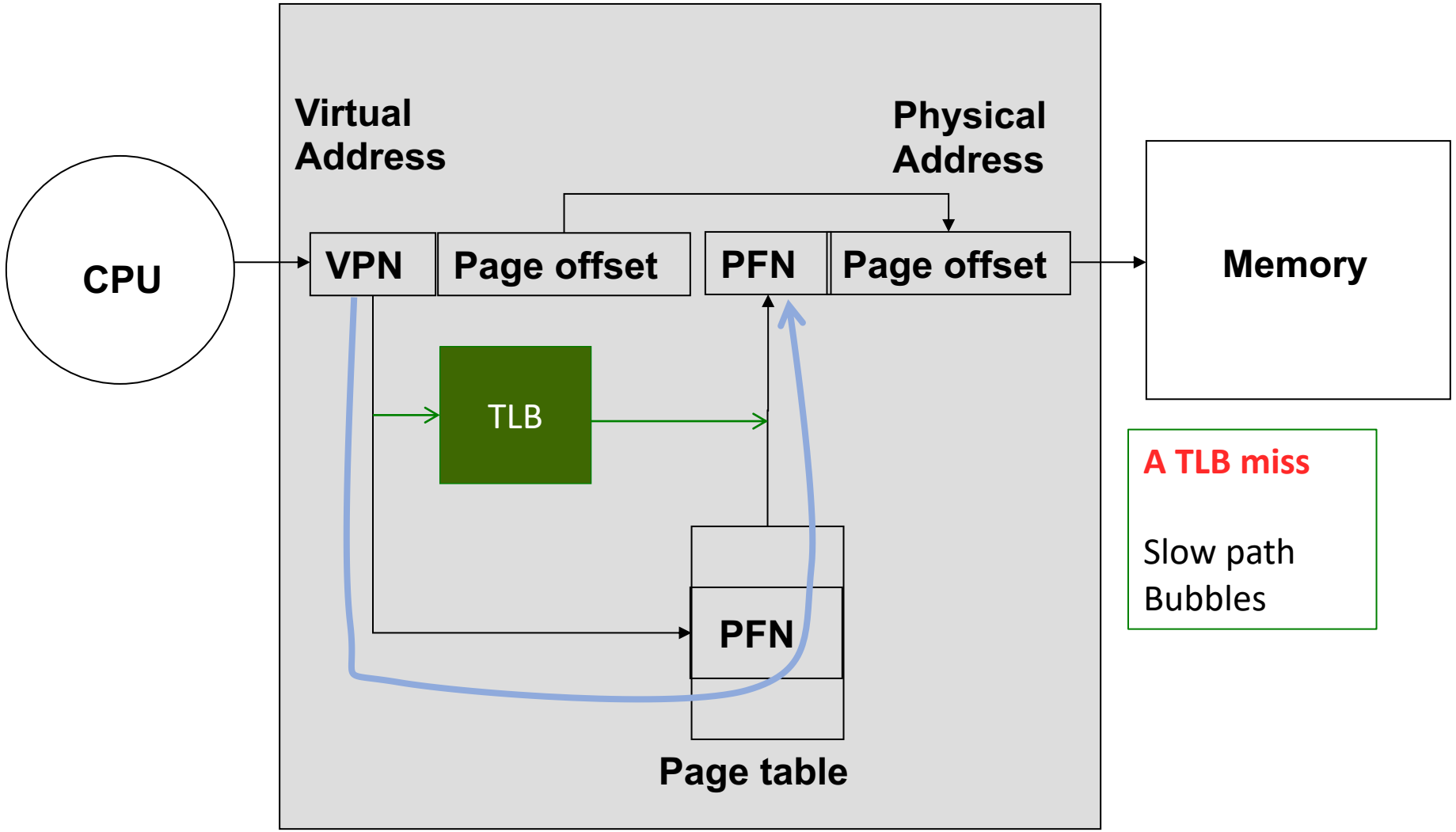
Agenda

- Finishing Chapters 7 & 8 – Memory Management
- Starting Chapter 9 – Memory Hierarchy

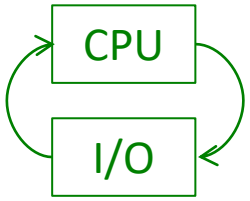
Speeding up address translation



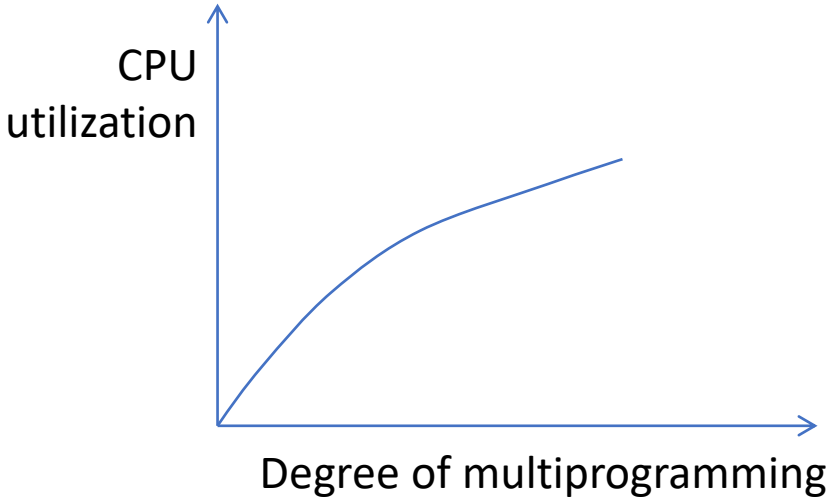
Speeding up address translation



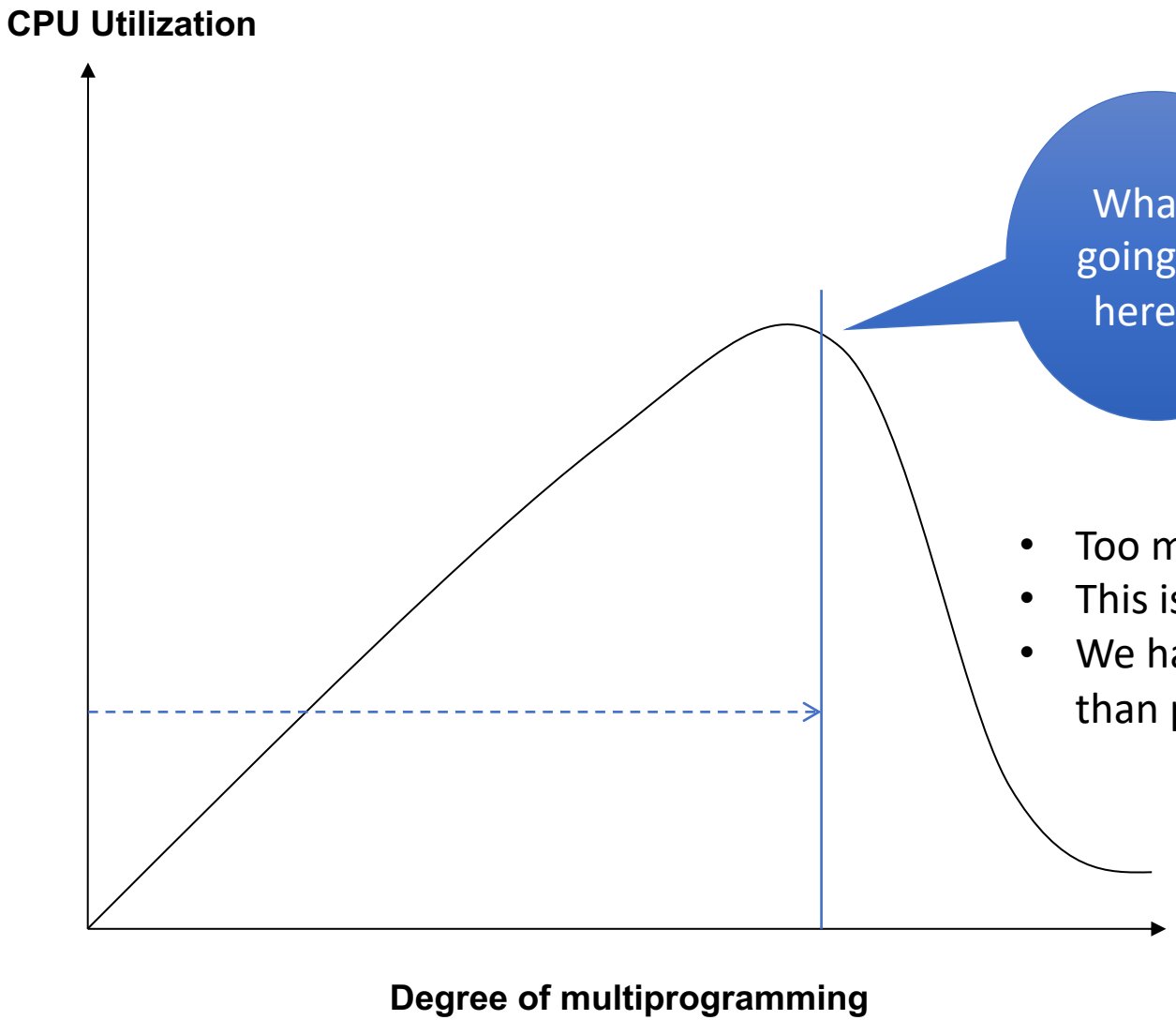
Given the nature of a process



- We want to increase multiprogramming to keep the CPU busy doing useful work
- This is what we want to see:



Extending the utilization curve



Thrashing

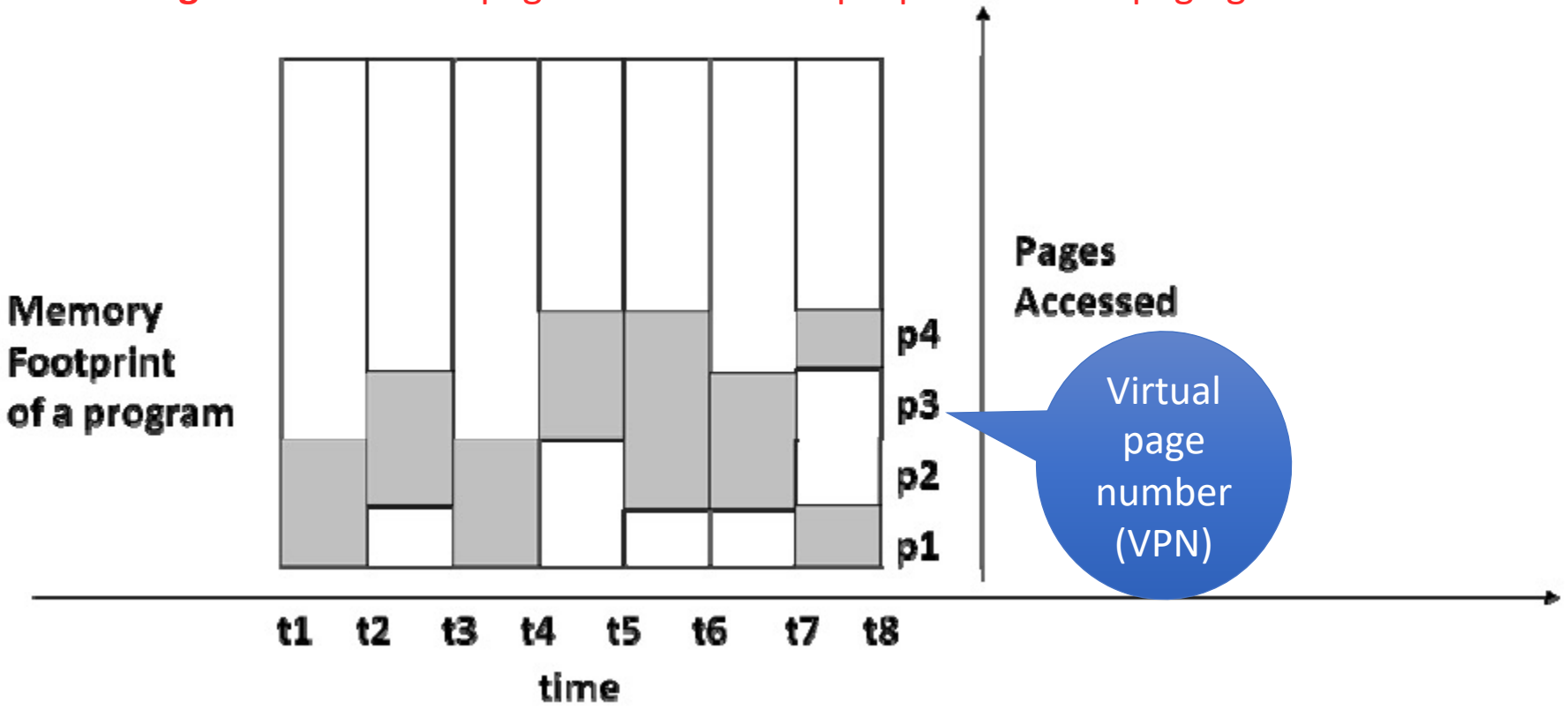
Nobody is getting any useful work done

- Too many processes asking for memory
- This is overcommitment of memory
- We have a bigger aggregate working set than physical memory can hold

Paging is implicit I/O on behalf of a process
→ System became I/O bound
→ Throughput drops precipitously

Working set of a program

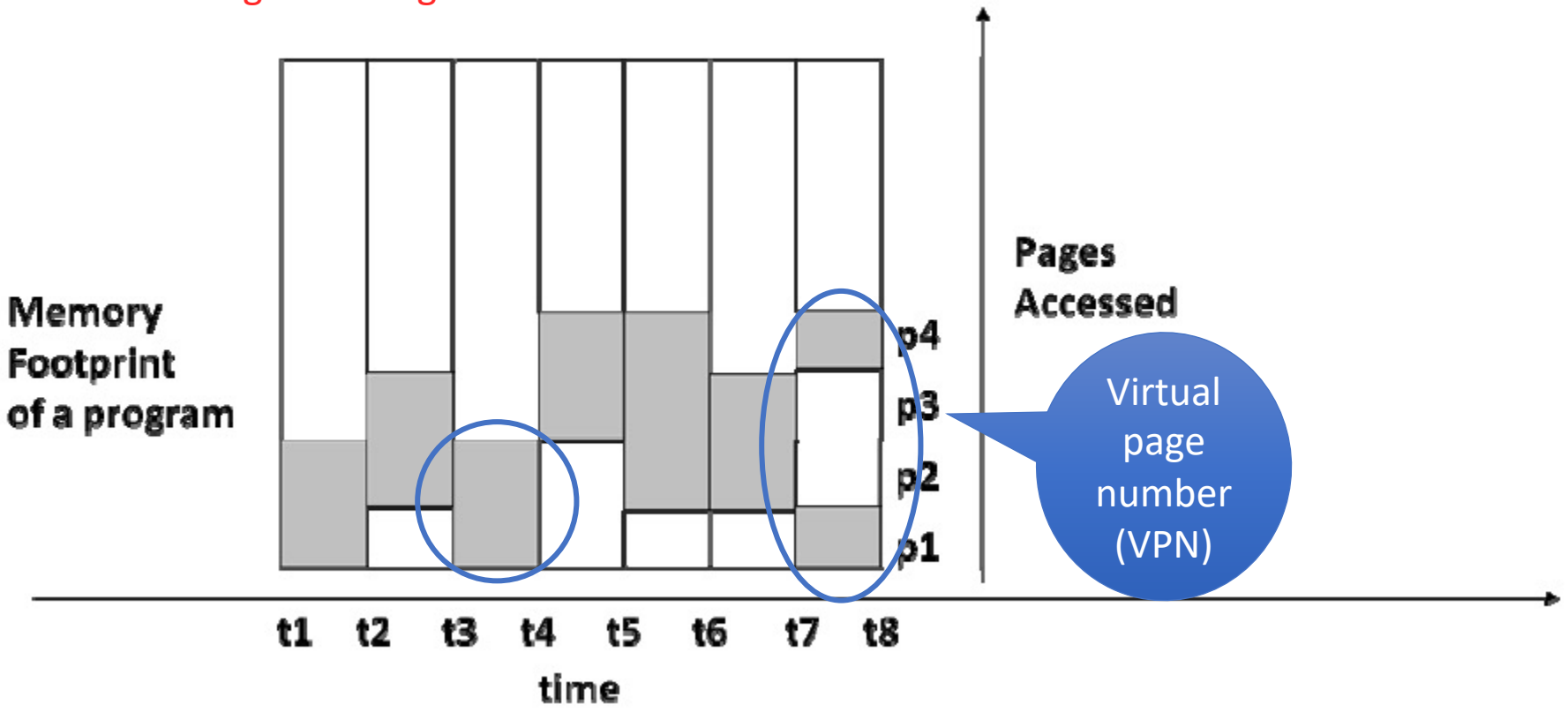
Working set is the set of pages needed to keep a process from paging



Working set size: number of page frames needed to hold working set

Working set of a program

The working set changes over time



$WS_{t3\text{-}t4}$ = { p1, p2 }

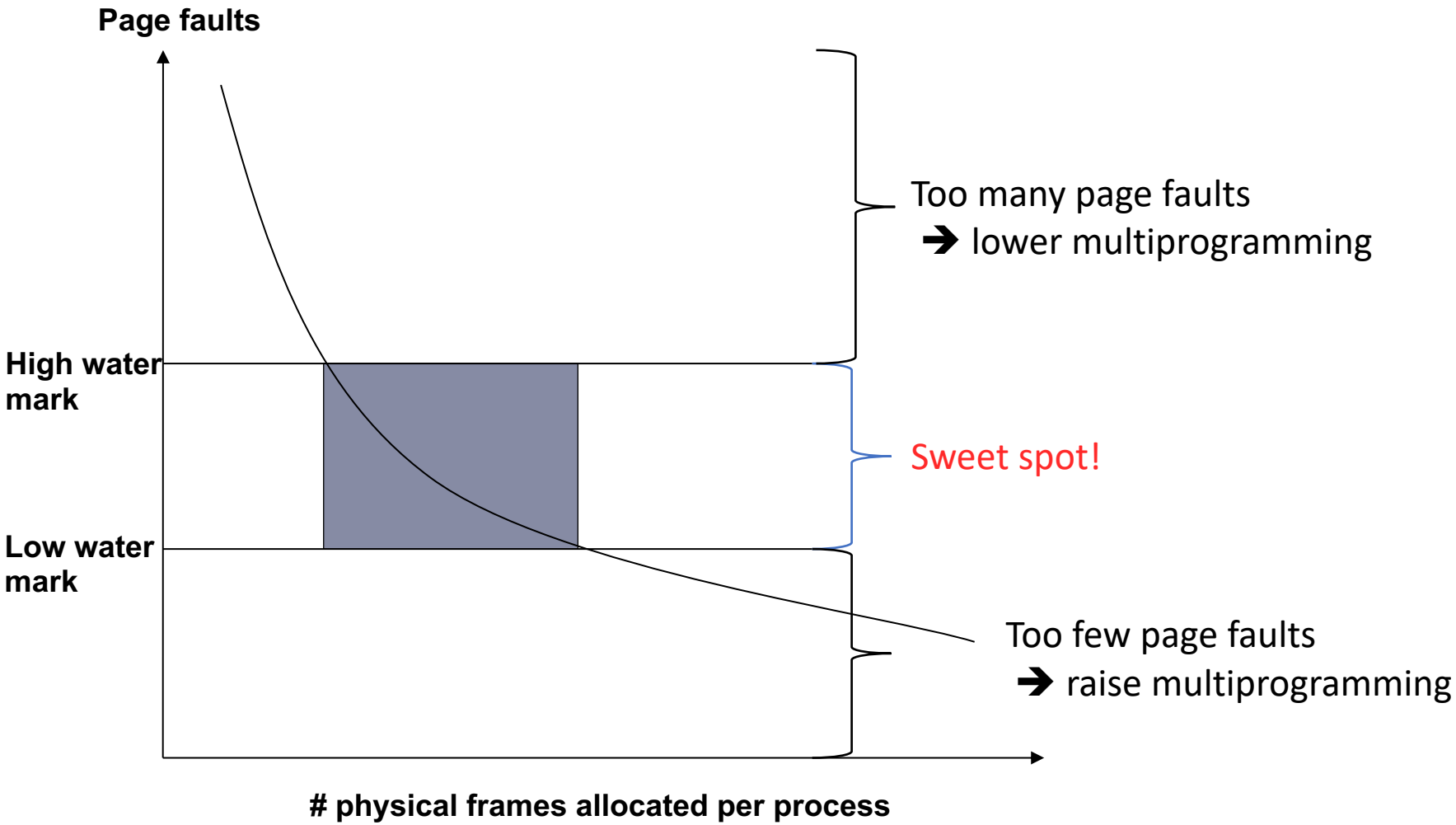
$WS_{t7\text{-}t8}$= { p4, p1 }

Memory pressure

$$memory.pressure = \sum_{i=1}^{n} WSS_i$$

- $P_1, P_2, P_3, \ldots$ are processes in memory each with a working set WSS_i
- The count of active processes *n* signifies the degree of multiprogramming
- How do we control the degree of multiprogramming?
  - **∑WSS > total physical memory**
    - ➔ swap out some processes
  - **∑WSS < total physical memory**
    - ➔ increase degree of multiprogramming

Controlling thrashing



Page faults are disruptive…

- … from a process point of view
  - ➔ implicit I/O
- … from a CPU-utilization perspective
  - ➔ overhead that doesn't contribute to work

- We need to limit impact of page faults to improve system performance

Session ID: dag2200

We can tell a system is thrashing if

20% A. I just want the participation credit

27% B. It has too few page faults per second

27% C. It has too many page faults per second

20% D. The ratio of I/O operations to CPU operations is not optimal

7% E. The combined working set of all processes is greater than the number of available page frames

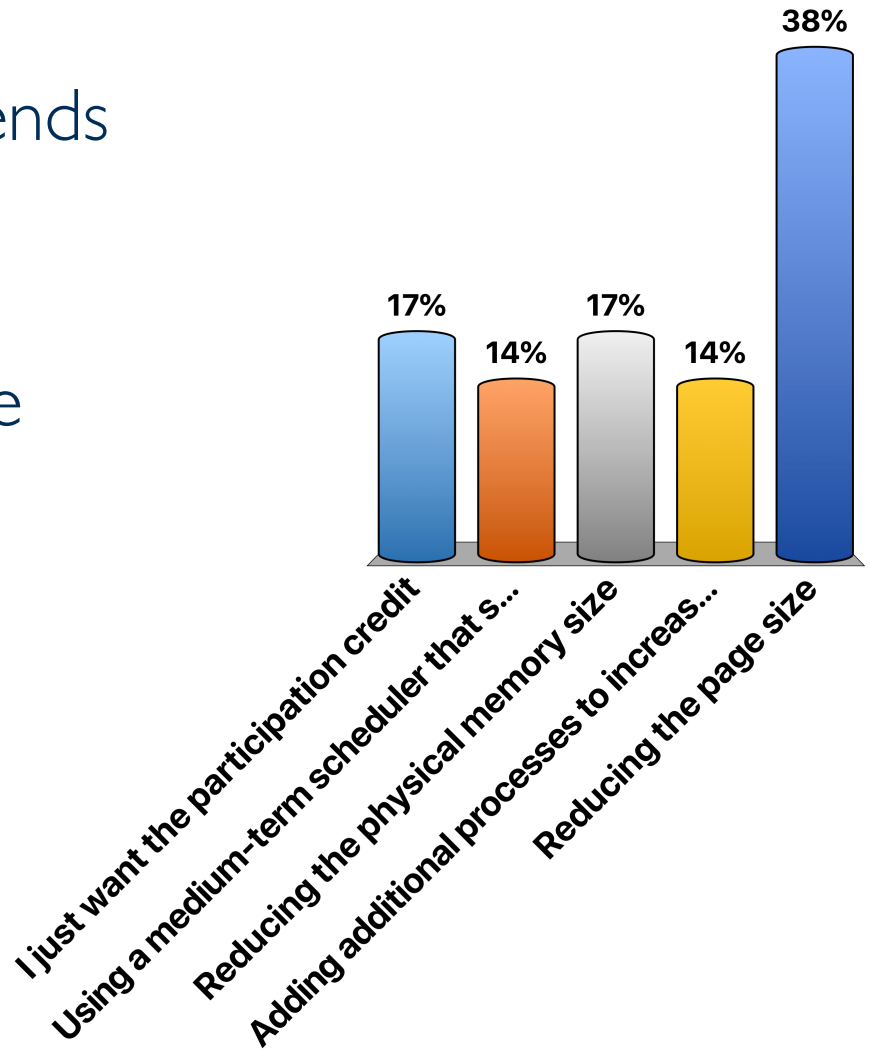
If only it were as easy as B! Thrashing implies too many page faults, but too many page faults don't always imply thrashing! An application can be constantly changing its working set without changing its working set *size*, for instance.

In reality, to diagnose thrashing, you'd look for a high paging rate, low CPU utilization, and several processes waiting on paging I/O for several seconds. Those metrics together are a good clue.

Session ID: dag2200

We can reduce thrashing by

A. I just want the participation credit
B. Using a medium-term scheduler that suspends processes until the condition improves
C. Reducing the physical memory size
D. Adding additional processes to increase the multiprogramming factor
E. Reducing the page size



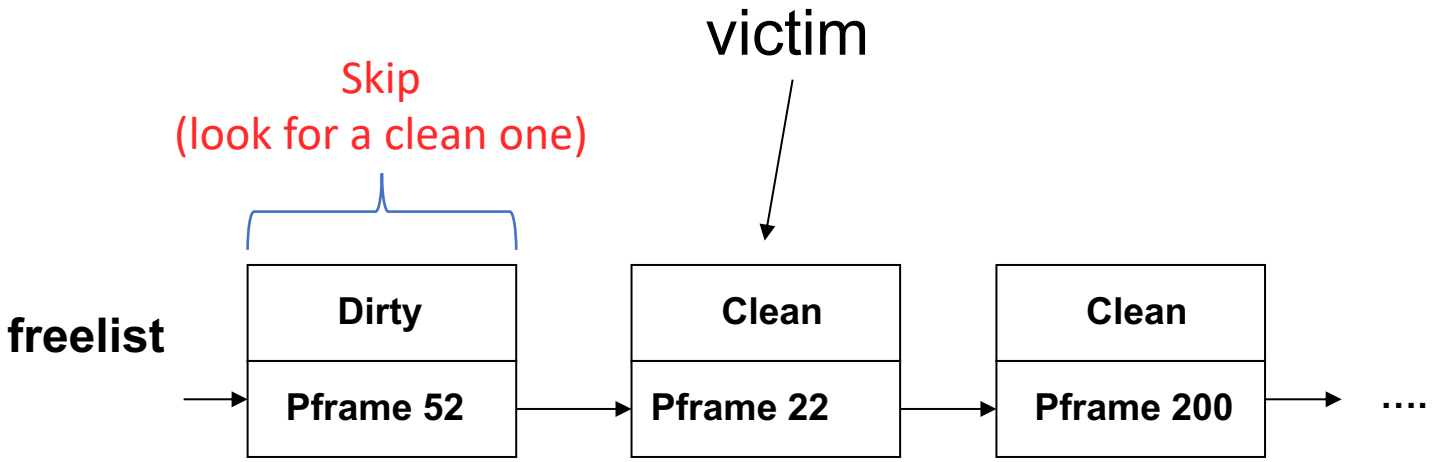
Of course this begs the question of how the medium-term scheduler is going to figure out that the system is thrashing…

Paging Optimizations

Accelerate page-in by removing functionality from critical path

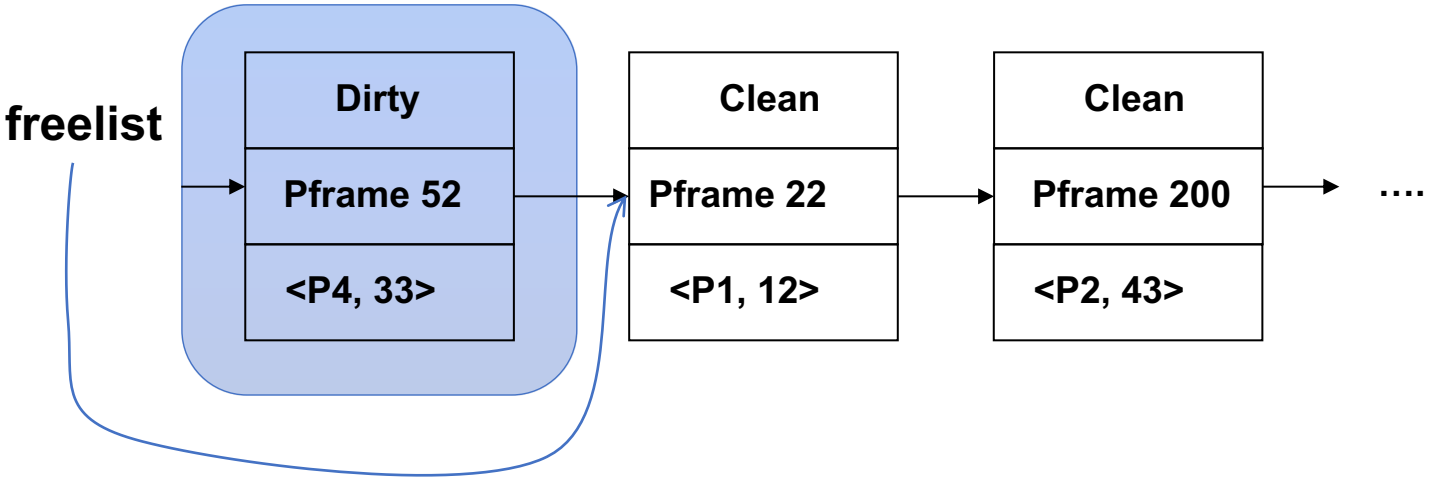
- Keep a (small) pool of free frames
  - Don't wait to start page replacement algorithm on a page fault

- Page replacement
  - Background activity of OS when CPU is not in use
  - If I/O is not busy, write out a "dirty" page which makes it "clean"



Reverse mapping to page table in free list

- Gives a "third" chance for reuse of a page before being kicked out
- P4 is running and page faults on VPN=33
- No need to go to disk!
- Just remap PFN=52 into PT for P4, VPN=33 and take it out of the freelist



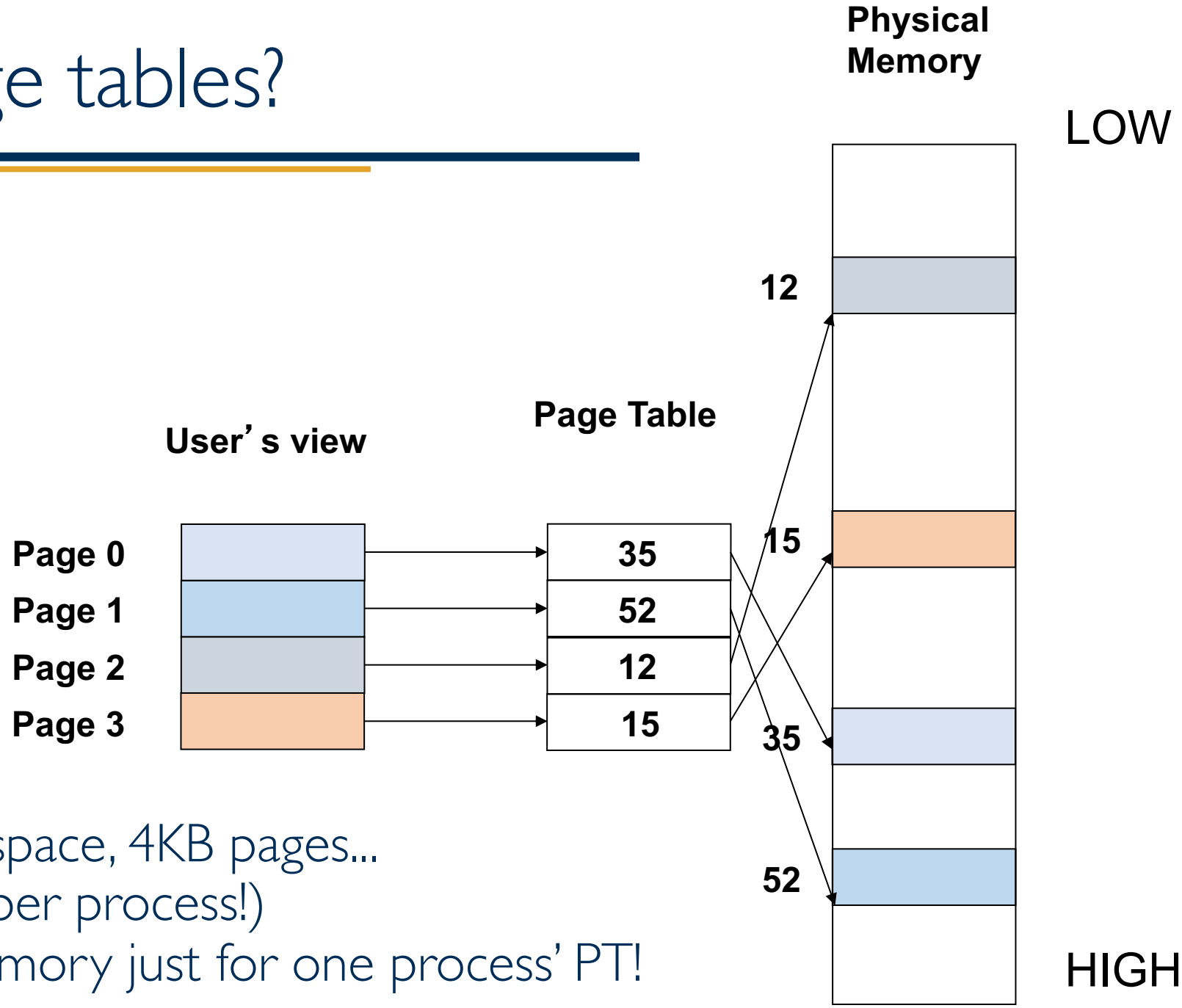
Linux VM and kswapd

```
$ free -h
             total       used       free     shared    buffers     cached
Mem:           15G       7.1G       8.5G       164K       703M       2.4G
…
-/+ buffers/cache:       4.0G        11G
Swap:         2.0G        26M       1.9G
```

- Kswapd
  - Paging daemon
  - Runs when "free" memory is low (about 2% of memory)
  - Uses a modified version of second-chance replacement
  - Links victim pages into the free list and sets their "invalid" PTE bits

- Page fault handler:
  - If the target page is still on the free list, it is reclaimed by removing it from the free list, marking its PTE bit "valid", and writing it out if it's dirty
  - Otherwise, the first frame in the Free List is removed, the target page is read into it, and the target page's PTE is modified to point to it and "valid" is set

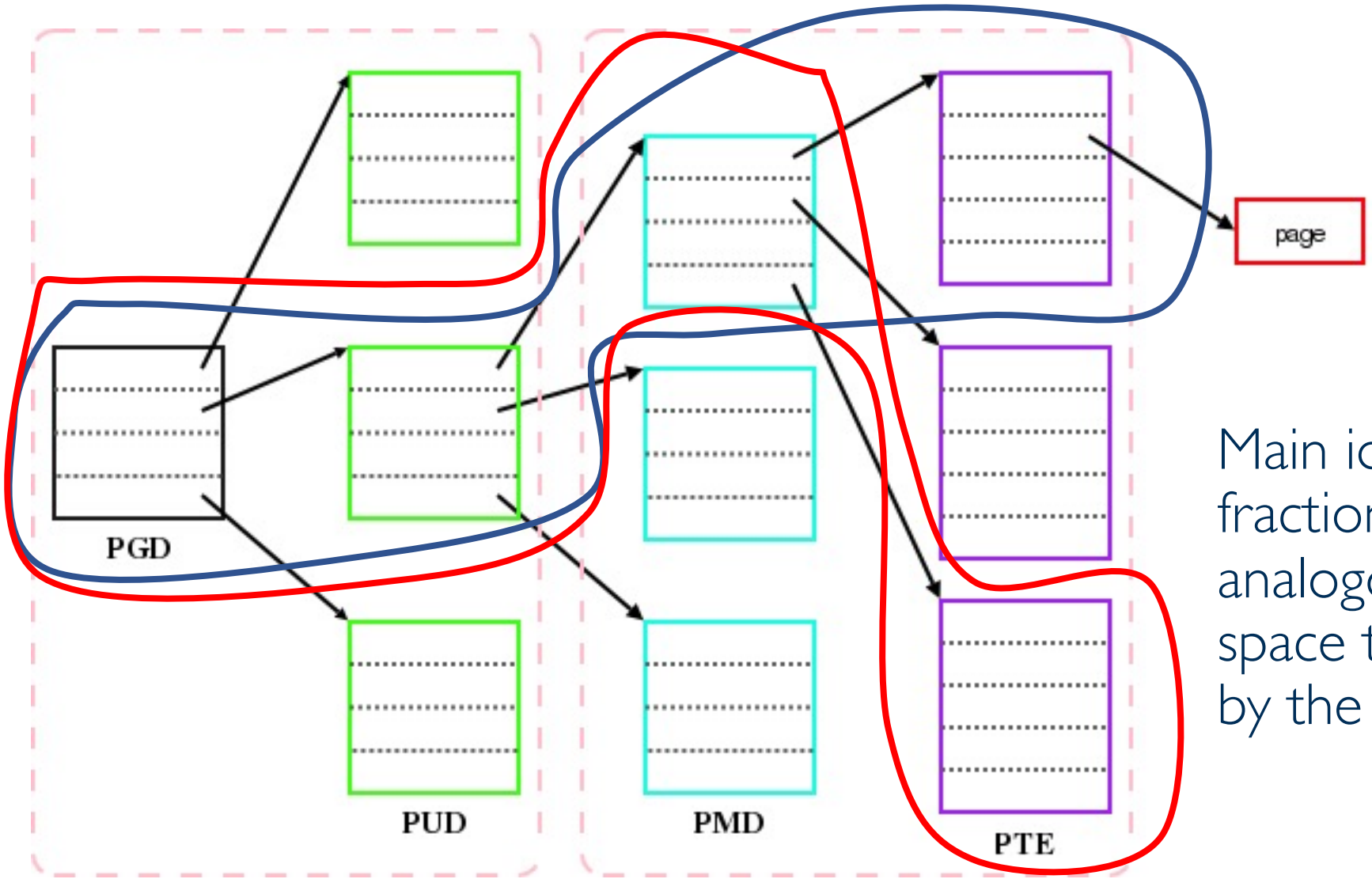
How big are page tables?



With 48-bit virtual address space, 4KB pages...

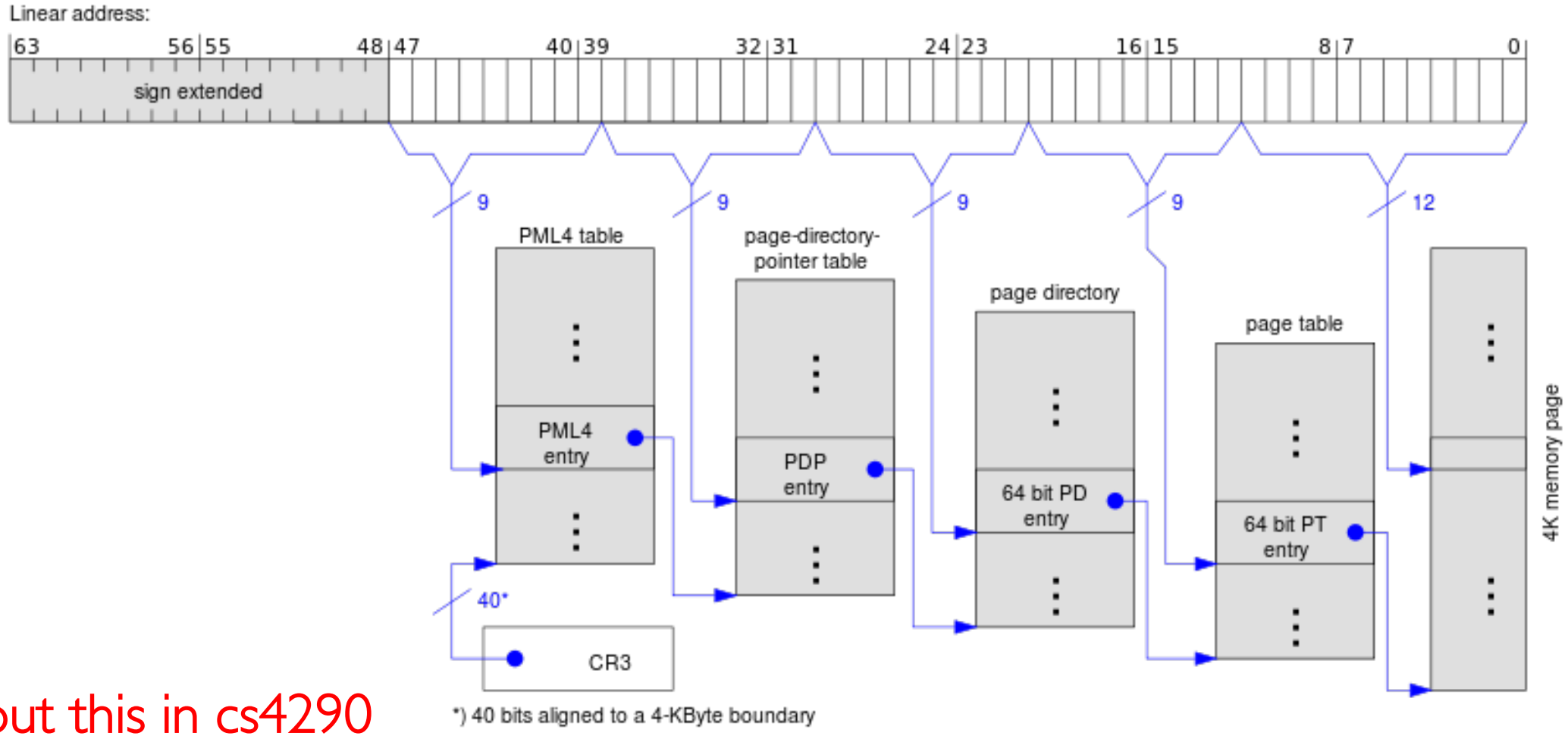
- Need 2^36 PT entries (per process!)
- That's several GBs of memory just for one process' PT!

Teaser: Hierarchical Page Table



Main idea: only populate a fraction of entire page table, analogous to the amount of VM space that is *actually* being used by the process

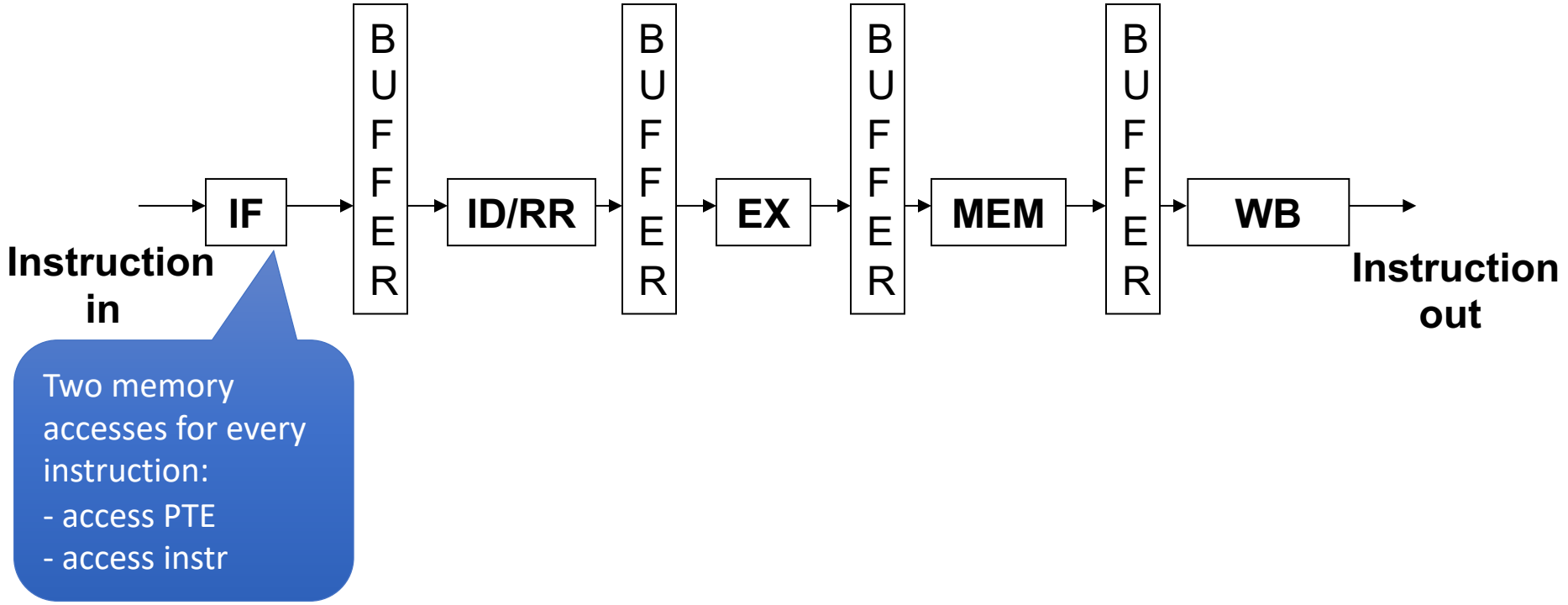
Intel's X86-64 4-level Page Tables



More about this in cs4290

Chapter 9:
Memory Hierarchy

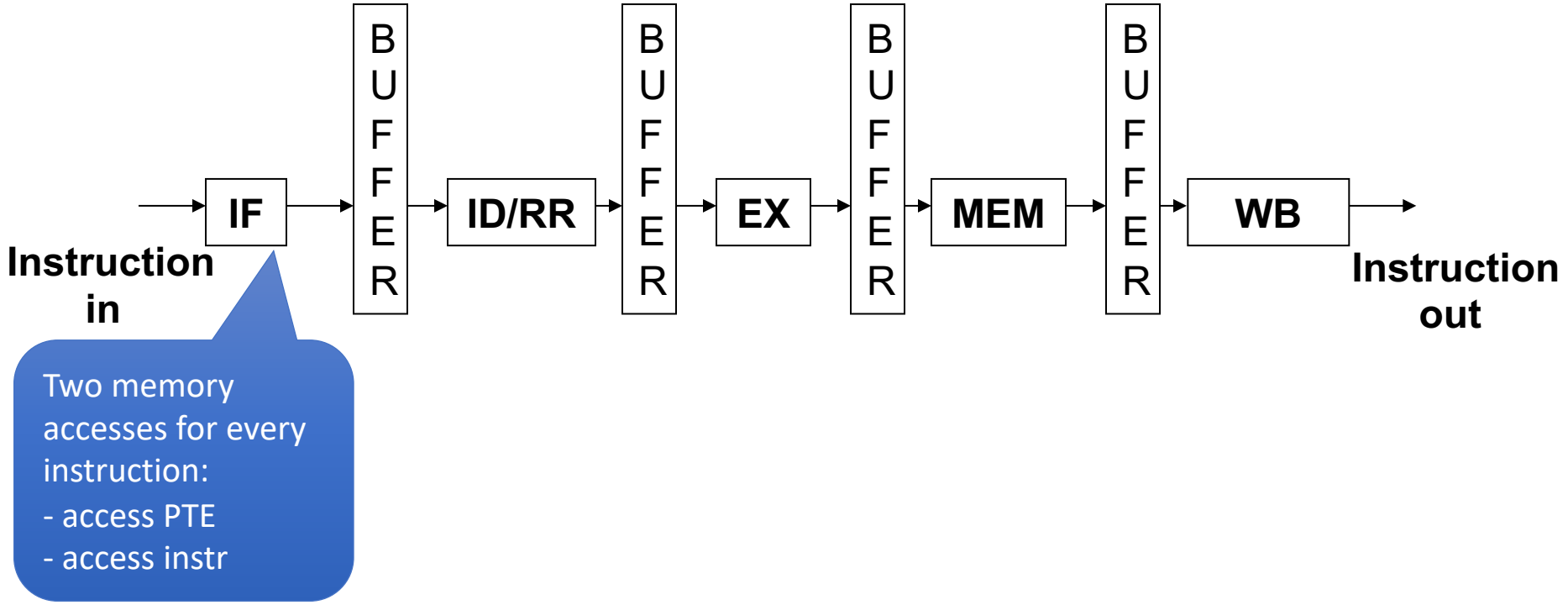
Recall: In a paged memory system



- What can we do?
  - Increase cycle time
  - Take 2 cycles in IF

Both bad!

In a paged memory system



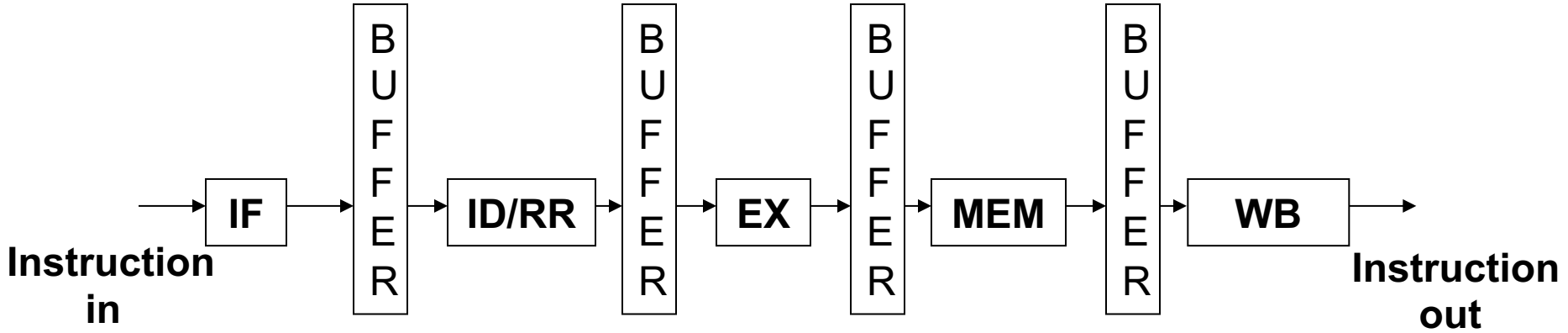
- TLB saves the day…
  - PC → TLB → memory is only one memory access
  - So we're back in business….

Are we really?

- What's the CPU clock cycle speed? ~1ns
- What's the memory access time? ~100ns

- It's roughly a 100:1 ratio!

What happens to the pipeline?



Not sustainable.

- With a [illegible] ratio [illegible] ppens in the IF stage on a memory fetch?
- 99 bub[illegible] On ever[illegible] uction!!

The register file and TLB…

- They're "kinda" memories, right?
- Why are they not a problem?
- TLB + Reg file ➔ Static RAM (SRAM) technology
- Memory ➔ Dynamic RAM (DRAM) technology
- SRAM ➔ 6 transistors per cell ➔ faster ➔ bulkier
- DRAM ➔ 1 transistor per cell ➔ slower ➔ denser

- You can have capacity or speed but not both!

What principle makes TLB work?

Locality

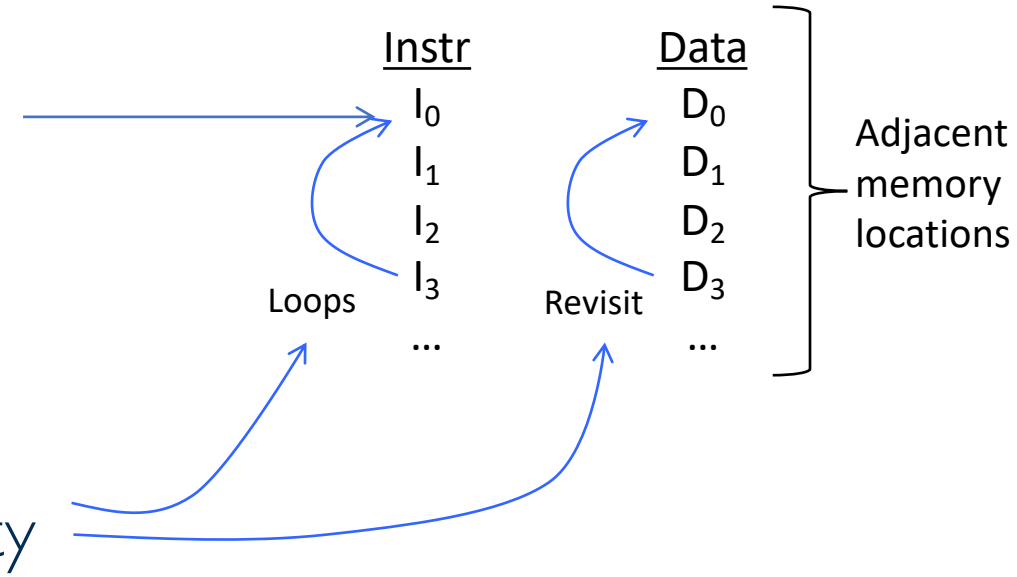
- Sequentiality of instructions in the program
- Sequentiality of data structures like arrays and structs

- "Recent" translations are stashed away from in-memory page tables into high-speed hardware
- Subsequent translations become faster ➔ no need to go to memory for most PTEs

- What if we apply the same principle to the data and instructions we need?

The principle is locality

- Spatial locality
- Temporal locality

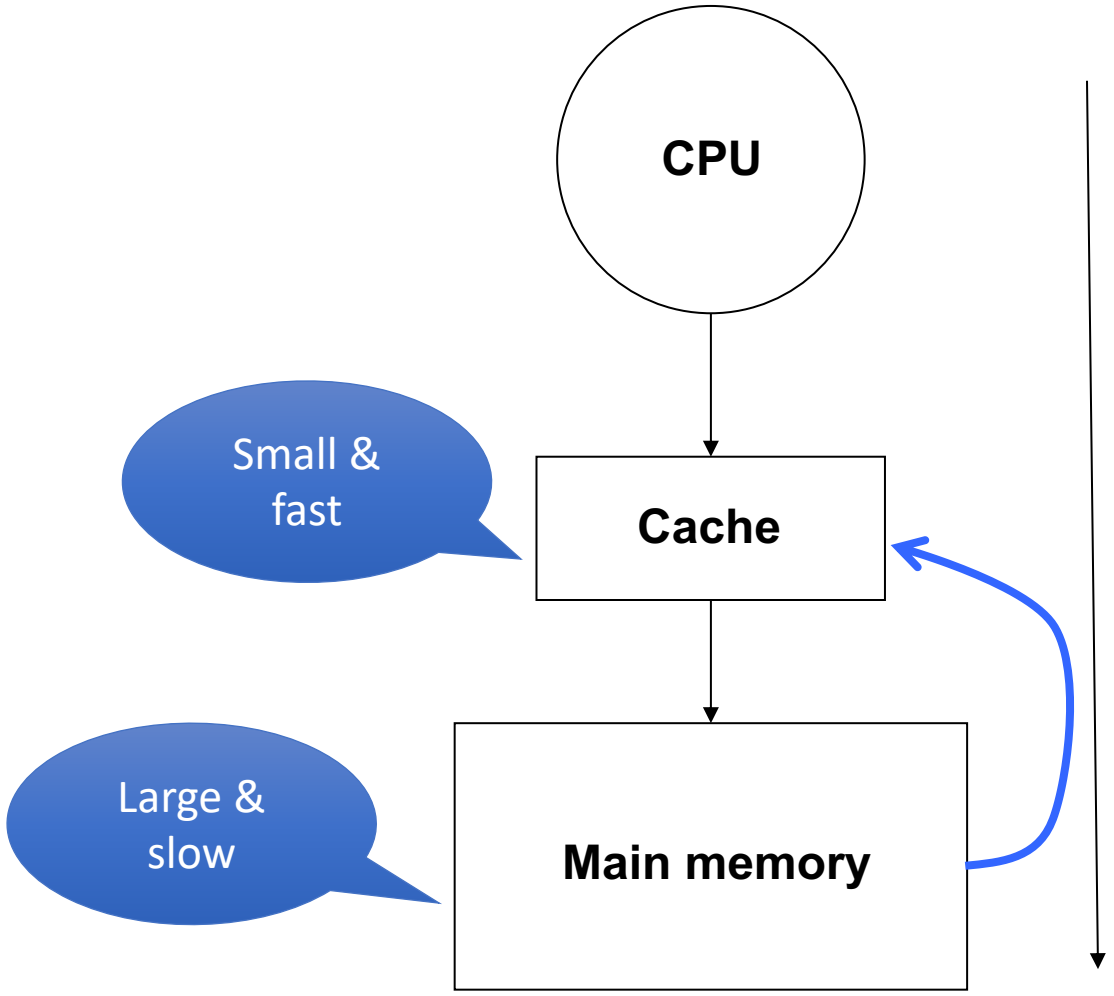


How do we put these two ideas together?

- Locality
- Small & fast or large & slow storage

- The concept of a "cache"
- TLB is a special instance of caching "addresses"

The memory hierarchy model



How do you use

- Spatial locality?
- Temporal locality?

Terminology

Hit	Hit ratio h	h + m = 1
Miss	Miss ratio m	
Cycle time	T_c cache access time	
	T_m memory access time	
Miss penalty	T_m	
Average memory access time (AMAT) *	AMAT = T_c + T_m * m	

* The book calls this *Effective Memory Access Time (EMAT)*

Modern memory hierarchy

$AMAT_i = T_i + M_i * AMAT_{i+1}$

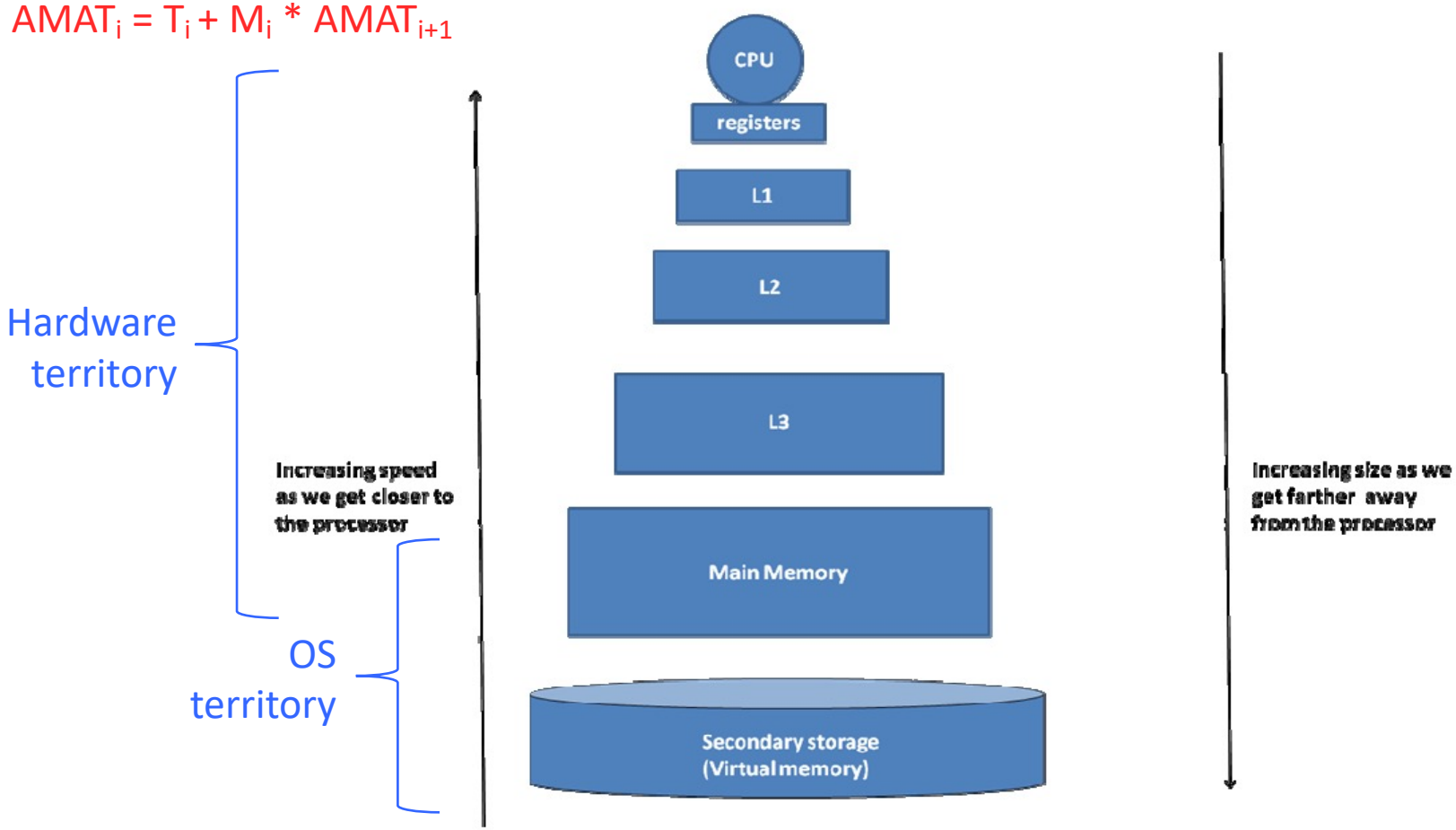


Figure 9.2: The entire memory hierarchy stretching from processor registers to the virtual memory.

Calculating AMAT

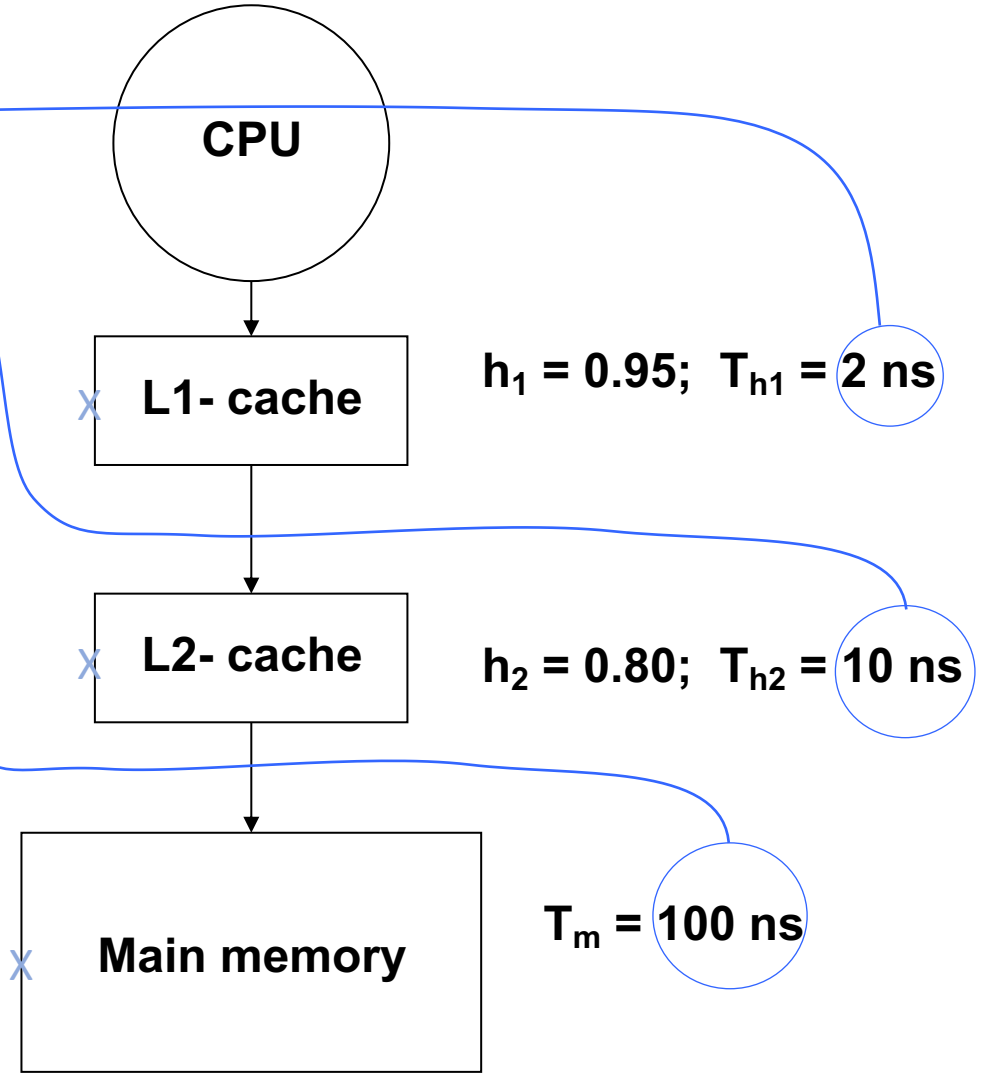
$AMAT_i = T_i + M_i * AMAT_{i+1}$

$AMAT_1 = 2ns + 0.05 * AMAT_2$

$AMAT_2 = 10ns * 0.20 * AMAT_3$

$AMAT_3 = 100ns$

$AMAT_1 = 3.5ns$



Session ID: dag2200

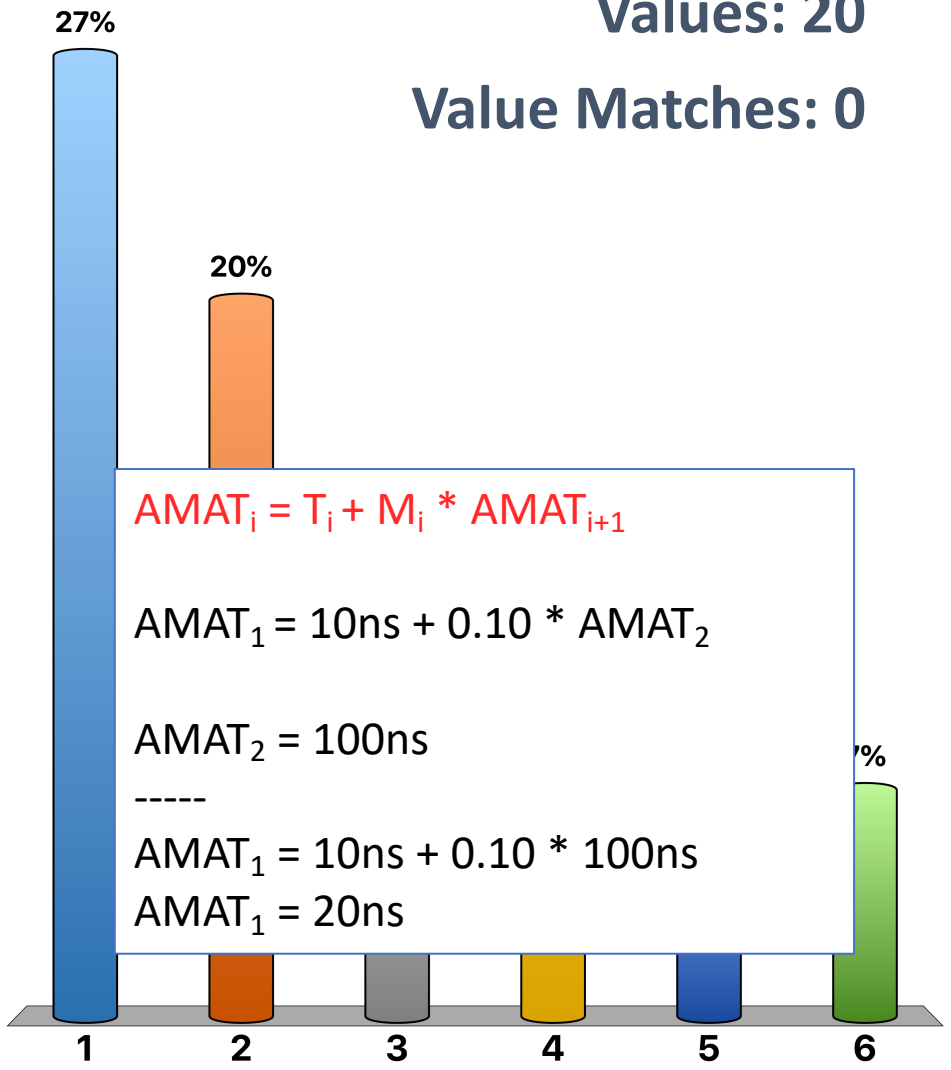
Calculate the AMAT...

You have one level of cache which has a hit rate of 90% and an access time of 10ns; your main memory has an access time of 100 ns. What is the AMAT in ns?

Rank	Responses
1	1
2	8
3	5
4	7
5	4
6	6

Values: 20

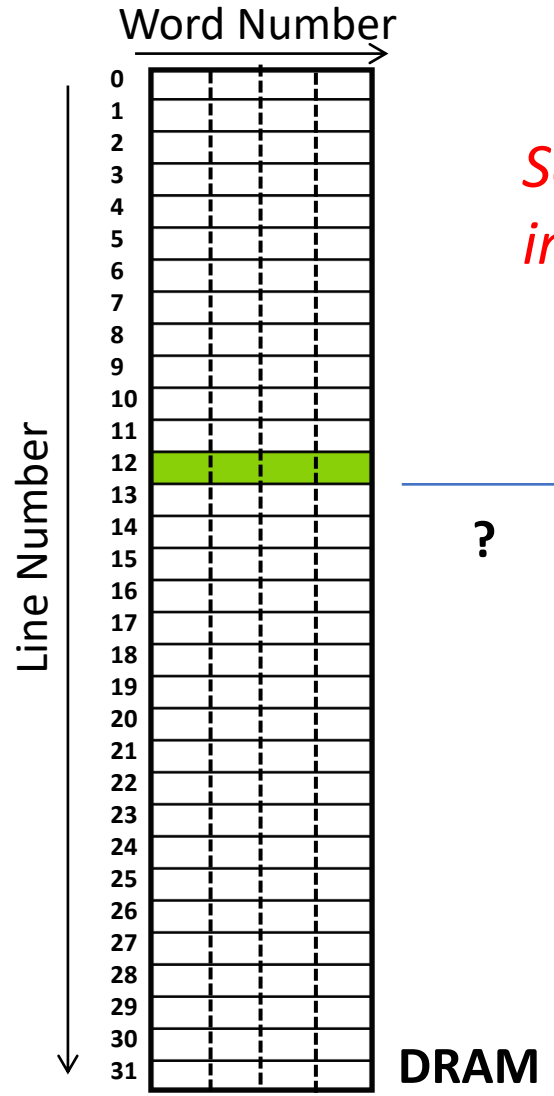
Value Matches: 0



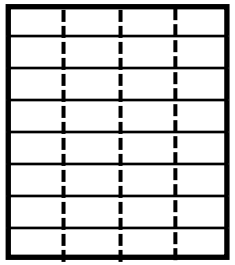
Cache Organizations

- Direct mapped
- Fully associative
- Set associative

Cache Placement

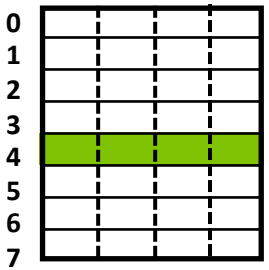


?

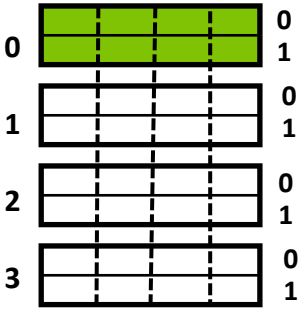


Cache

Searching is easy but inefficient use of space

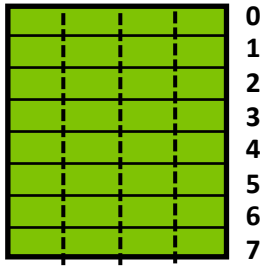


Direct Mapped
one location



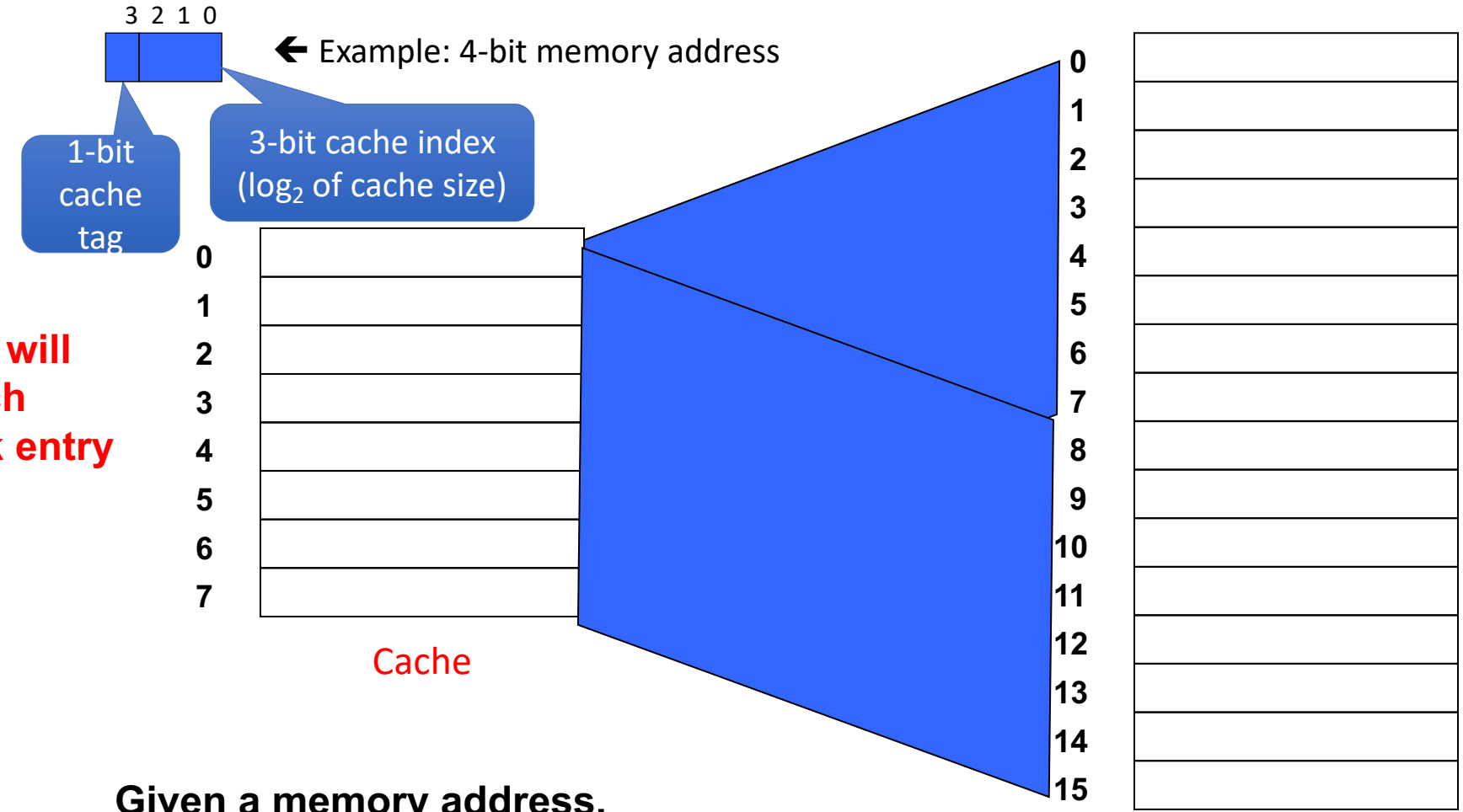
(2-way) Set Associative
2 locations

Searching is more expensive



Fully Associative
anywhere

Direct mapped cache



For now we will assume each cache block entry is a word

Given a memory address,
What is the numerical value of cache index?

Types of misses

- Compulsory (first time an address is requested)
- Capacity (cache is full)
- Conflict (vying for space in a full set in the cache)

Known as the 3 C’s!

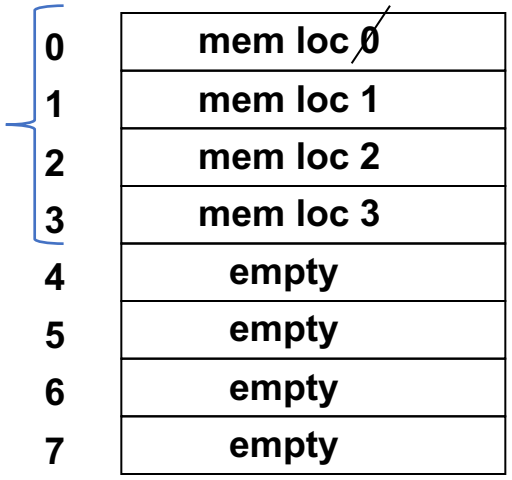
Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

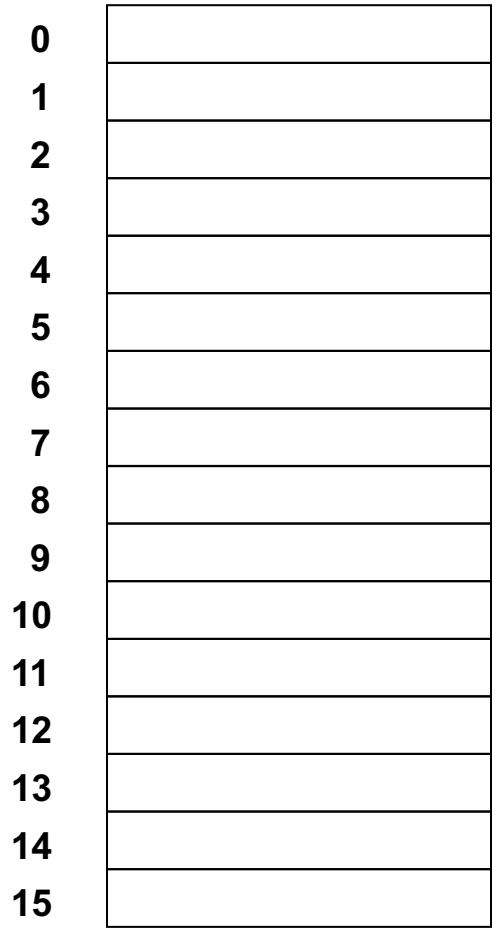
misses hits miss

These were compulsory misses.

The data were referenced for the first time.



Cache



Memory

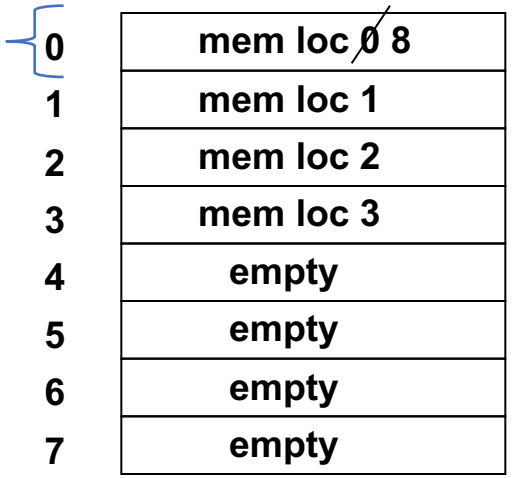
Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

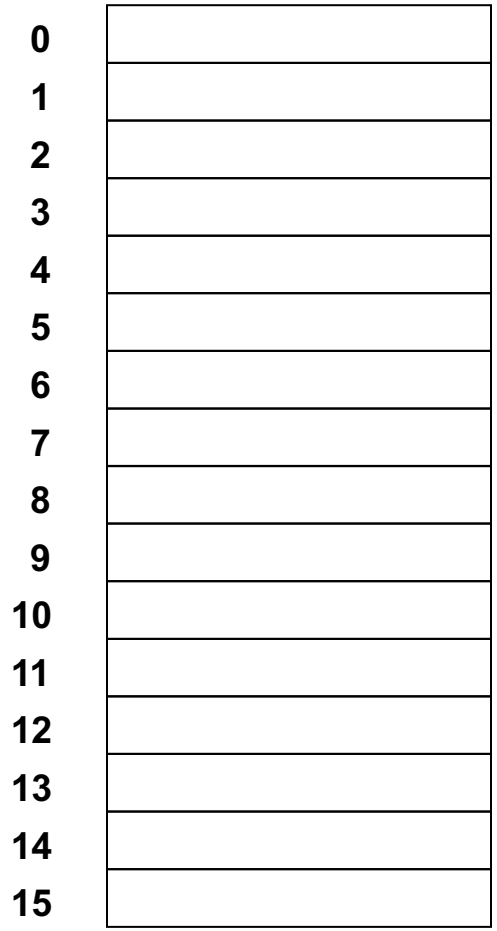
misses hits miss

This is also a compulsory miss.

There are two memory addresses mapping to the same cache entry. Must replace the old one (0)



Cache



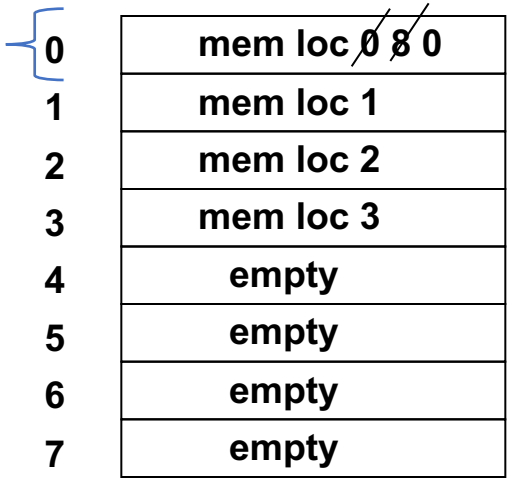
Memory

Example

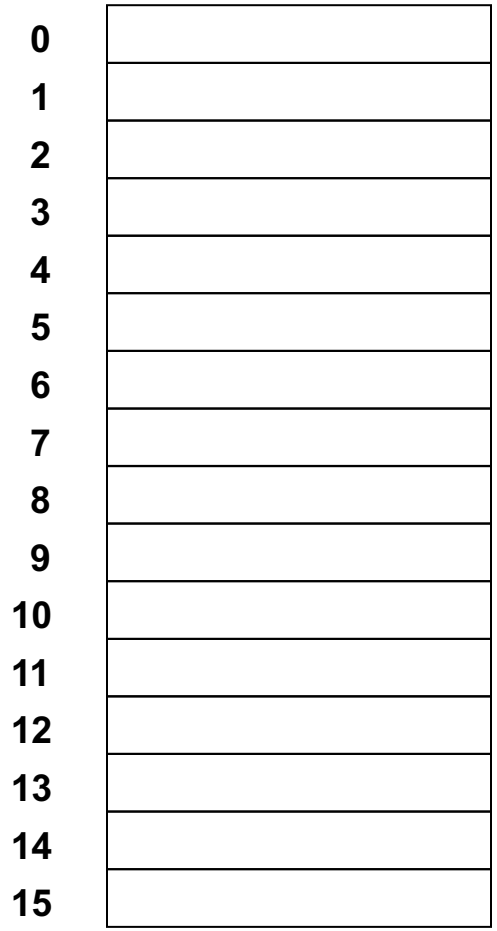
Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

misses hits miss

This is now a conflict miss.
Cache block 0 used to be in the cache, but was replaced because the previous access to block 8 maps to the same entry

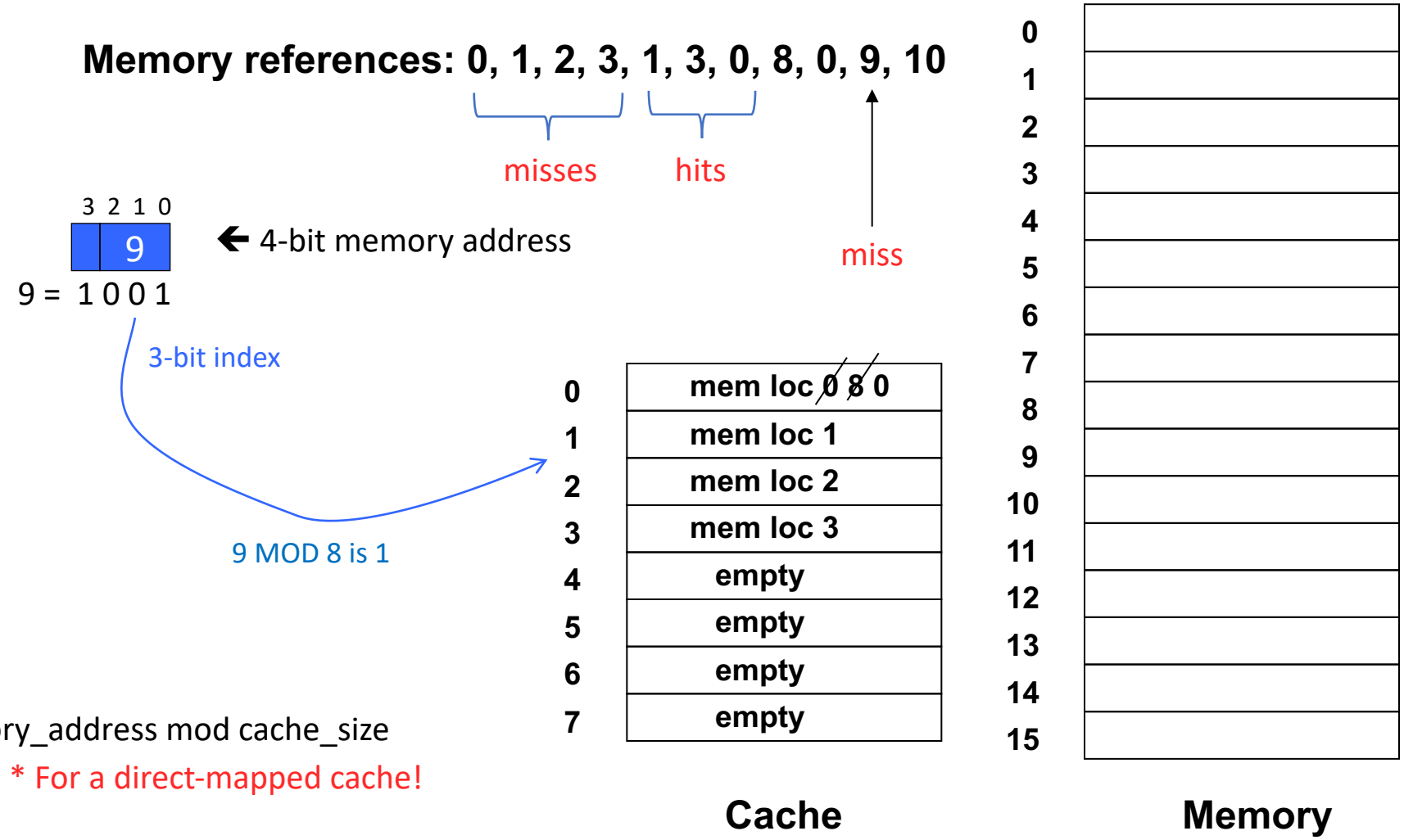


Cache



Memory

Example

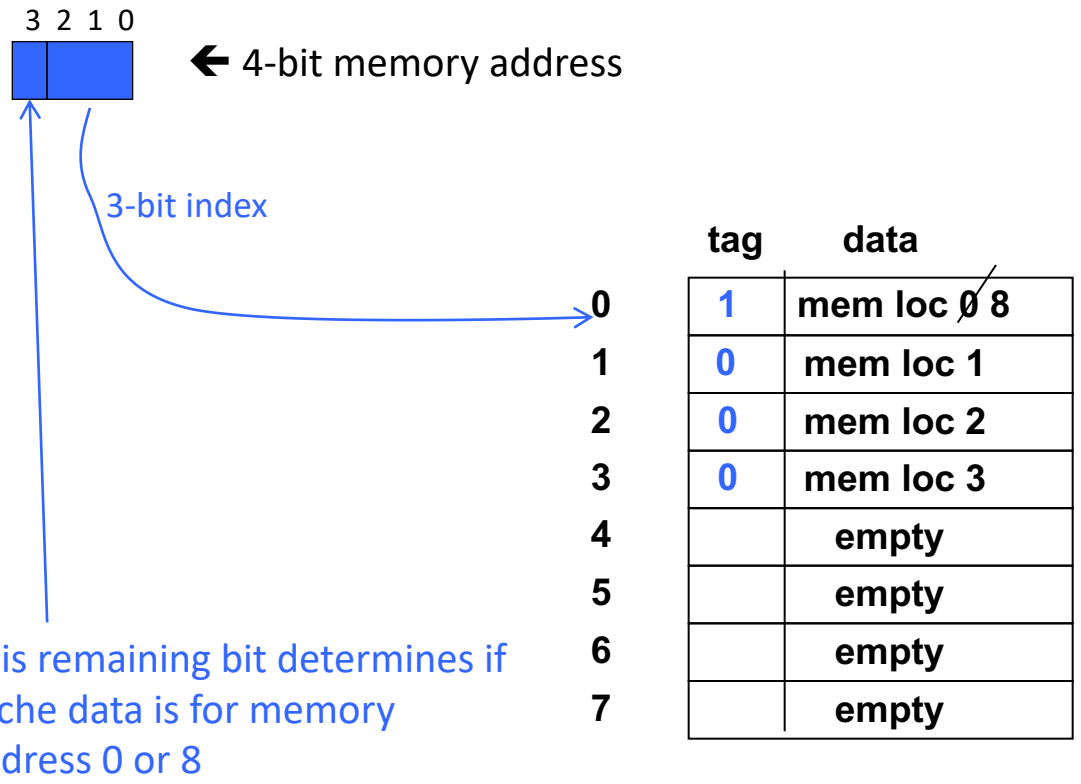


index = memory_address mod cache_size

* For a direct-mapped cache!

How do we disambiguate data?

We use the part of the memory address not used as cache index as the tag to label the data in the cache



	tag	data
0	1	mem loc ~~0~~ 8
1	0	mem loc 1
2	0	mem loc 2
3	0	mem loc 3
4		empty
5		empty
6		empty
7		empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

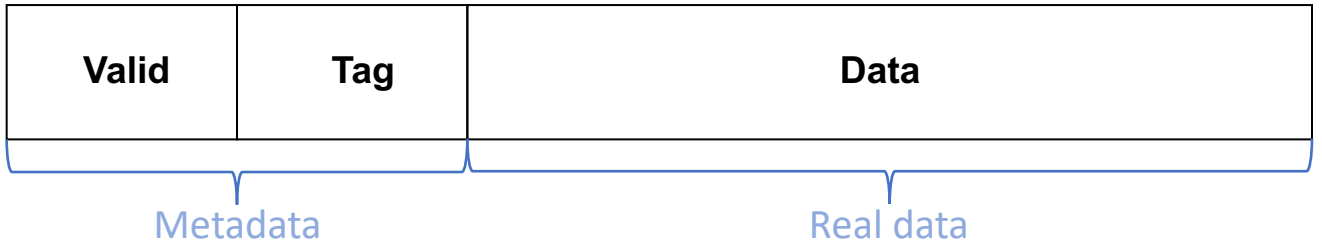
Memory

How do we know data is valid?

- At power-up, a cache may contain garbage!
- Tags help disambiguate, not validate

	valid	tag	data
0	1	1	loc 8
1	1	0	loc 1
2	1	0	loc 2
3	1	0	loc 3
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Fields in a Direct Mapped Cache



Session ID: dag2200

What does the cache entry contain?

You have a 64 entry direct-mapped cache for a word-addressable memory with 16-bit addresses and 16-bit words (like LC-3).

30% A. I just want the participation credit

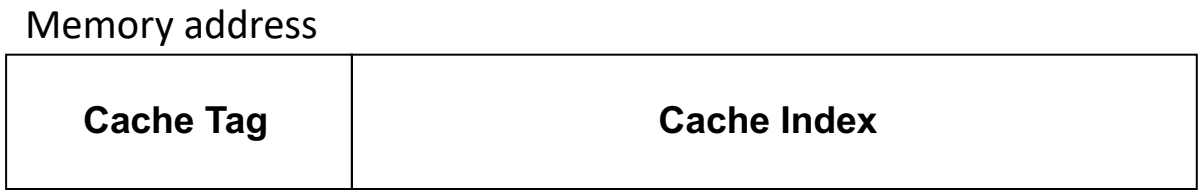
13% B. 1 bit valid flag, 6 bit tag, 10 bit data

20% C. 1 bit valid flag, 10 bit tag, 16 bit data

27% D. 1 bit valid flag, 10 bit tag, 6 bit data

10% E. 2 bit valid flag, 12 bit tag, 16 bit data

Interpreting memory addresses



- index = memory addr mod cache size
  - MemAddr = 8 ➔ index = 0
  - MemAddr = 0 ➔ index = 0
- number of tag bits = memory addr size - cache index size

Why not the other way around?
➔ use the low bits for cache tag?

Index first, tag last?

Address:

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1

index tag

	valid	tag	data
0	1	0	loc 0
1	0	X	empty
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access location 0

	valid	tag	data
0	1	1	loc 0 1
1	0	X	empty
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access location 1

	valid	tag	data
0	1	1	loc 0 1
1	1	0	loc 2
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access location 2

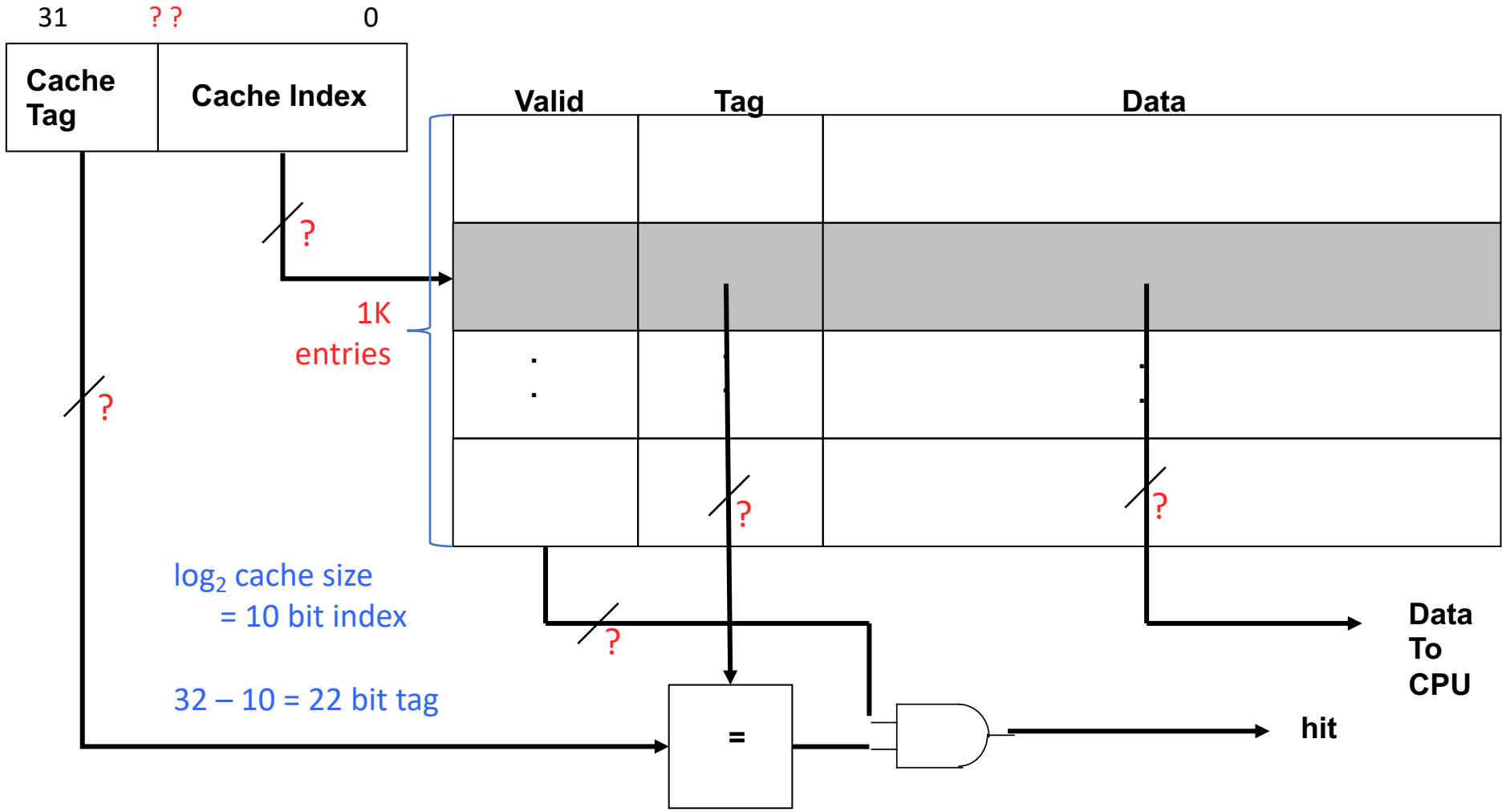
	valid	tag	data
0	1	1	loc 0 1
1	0	1	loc 2 3
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access location 3

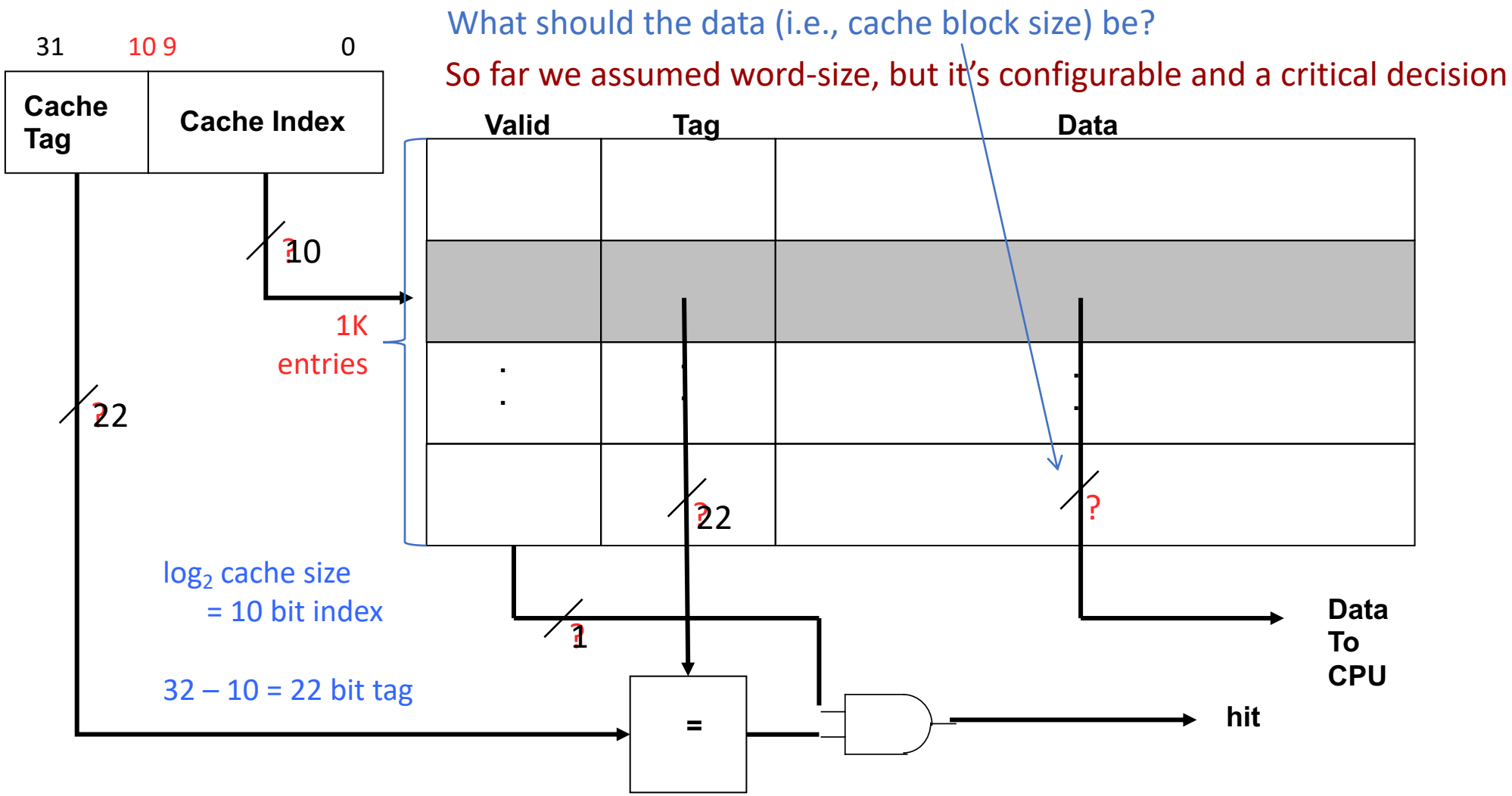
Cache occupancy if we switch index and tag is BAD!!

➔ loss of spatial locality

Hardware


31
? ?
0
Cache Tag
Cache Index
Valid
Tag
Data
?
1K entries
?
?
?
log2 cache size = 10 bit index
32 – 10 = 22 bit tag
=
hit
Data To CPU

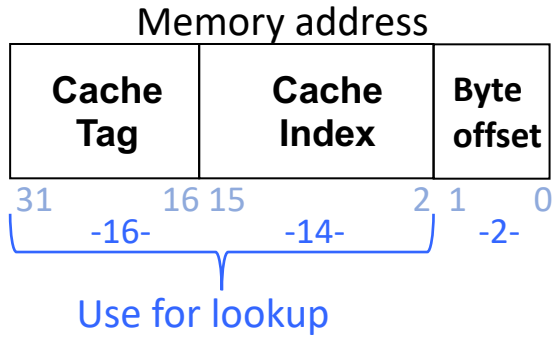
Hardware


What should the data (i.e., cache block size) be?
So far we assumed word-size, but it's configurable and a critical decision
31
10 9
0
Cache Tag
Cache Index
Valid
Tag
Data
10
1K entries
22
22
?
1
log2 cache size = 10 bit index
32 – 10 = 22 bit tag
=
Data To CPU
hit

Example

Let us consider the design of a direct-mapped cache for a realistic memory system.

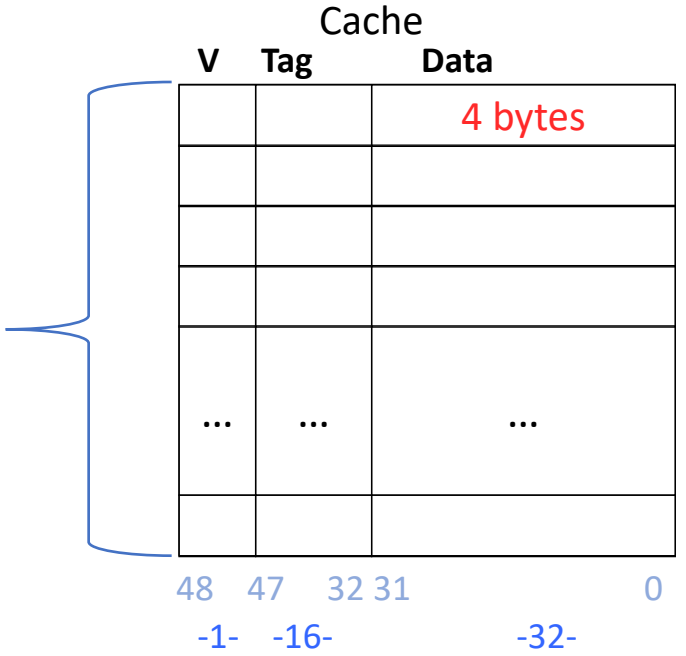
- Assume that the CPU generates a 32-bit byte-addressable memory address.
- Each memory word contains 4 bytes.
- A memory access brings a full word into the cache.
- The direct-mapped cache is 64K bytes in size (this is the amount of data that can be stored in the cache), with each cache entry containing one word of data.
- Compute the additional storage space needed for the valid bits and the tag fields of the cache.



Consider only word address for lookup

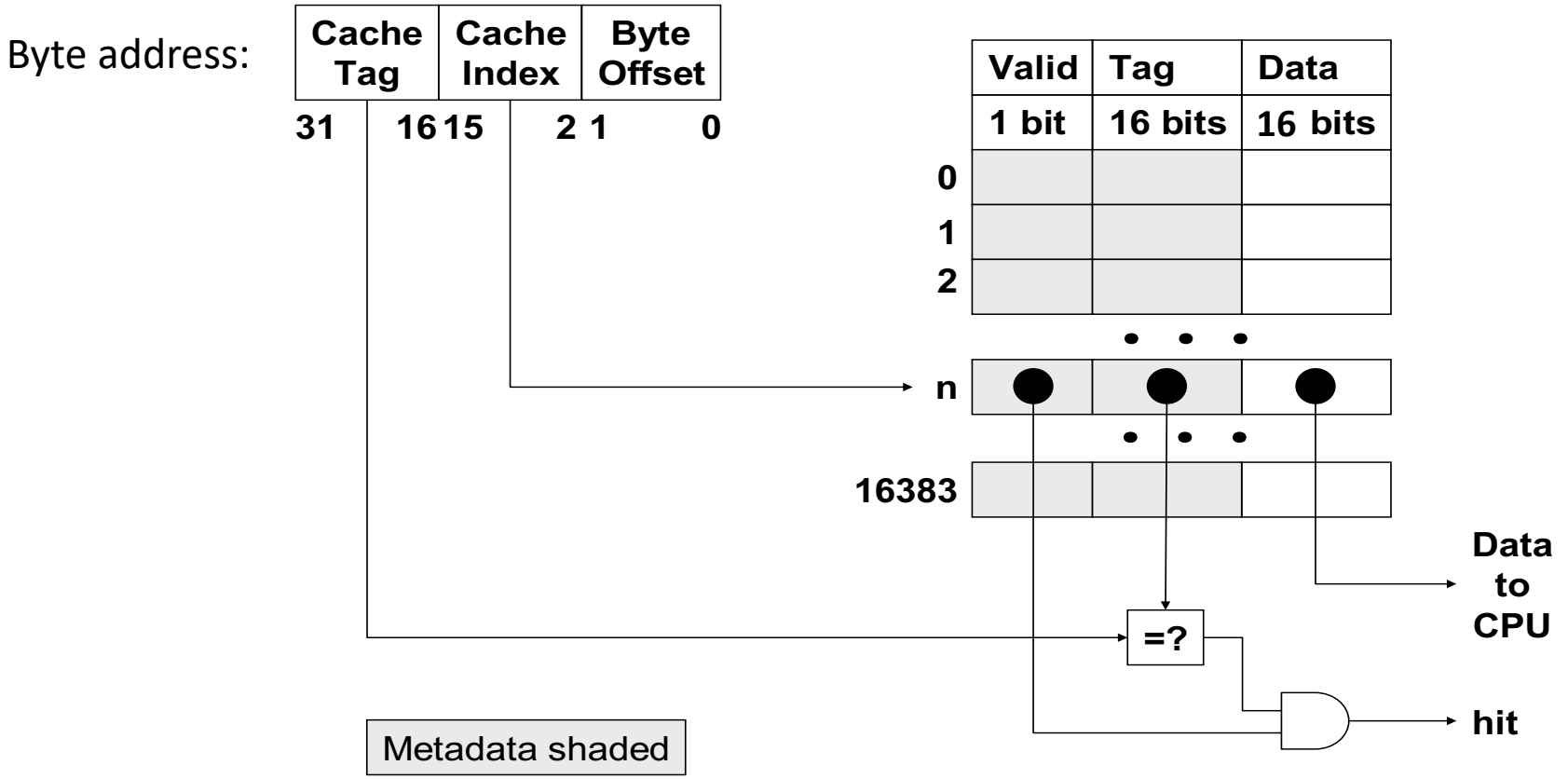
Consider only real data → 64k/4=2^{14} rows in cache

Metadata (overhead)



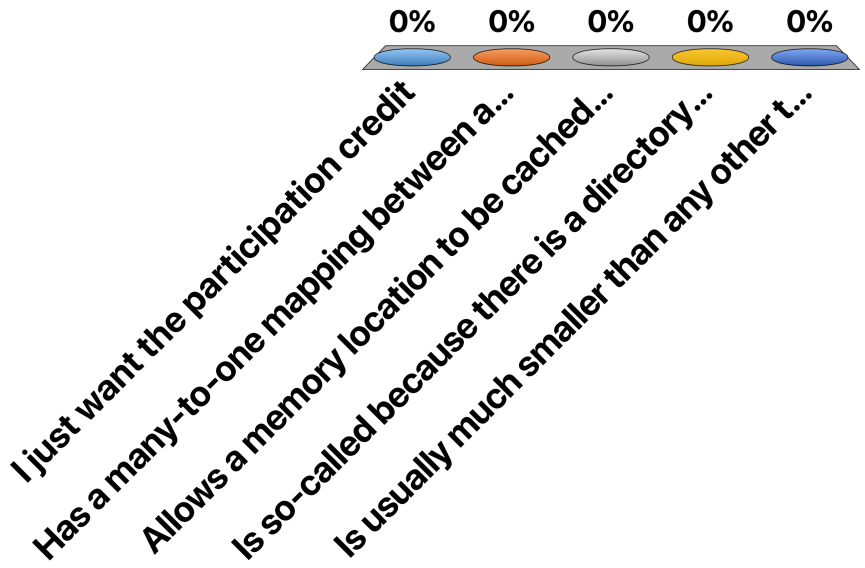
2^{14} * (1 + 16) additional bits for metadata

Memory address interpretation when single cache block contains multiple bytes



A direct-mapped cache

A. I just want the participation credit
B. Has a many-to-one mapping between a memory location and a cache location
C. Allows a memory location to be cached wherever there is space in the cache
D. Is so-called because there is a directory associated with the contents of the cache
E. Is usually much smaller than any other type of cache organization



Session ID: dag2200

In a direct-mapped cache with a t-bit tag

A. I just want the participation credit
B. There is one 1-bit tag comparator for each cache line
C. There is one t-bit tag comparator for each cache line
D. There is one 1-bit tag comparator for the entire cache
E. There is one t-bit tag comparator for the entire cache