Georgia Tech Computer Science

CS2200
Systems and Networks
Spring 2022

Lecture 19:
Memory Hierarchy pt 2

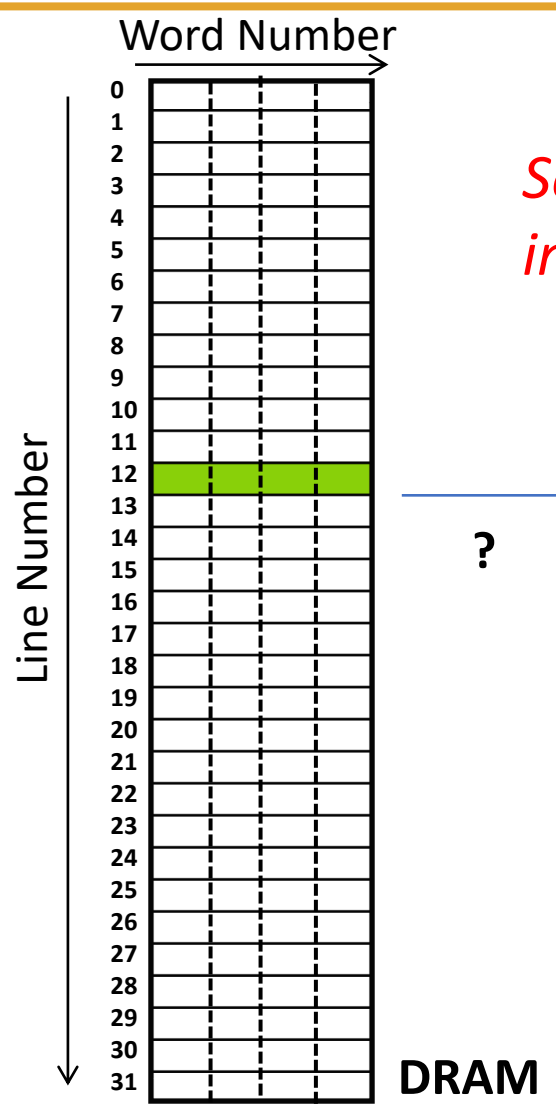
Alexandros (Alex) Daglis
School of Computer Science
Georgia Institute of Technology
adaglis@gatech.edu

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

Announcements

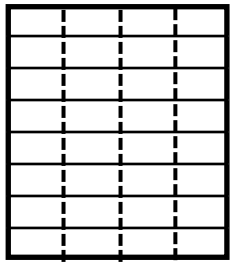
- Exam 2 will take place tomorrow at 6:30pm
  - Be there 5 minutes in advance
  - 70min
  - Note the update in Question 6 (announced on Canvas)
  - Mega-thread of clarification questions on Piazza

Cache Placement

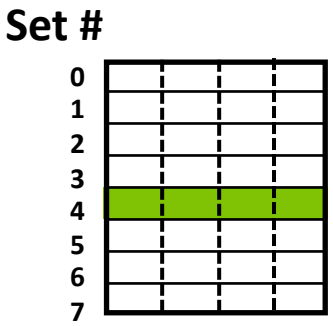


Searching is easy but inefficient use of space

?

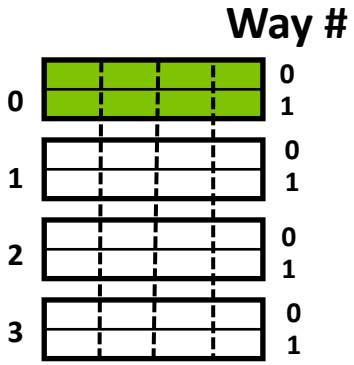


Cache



Direct Mapped

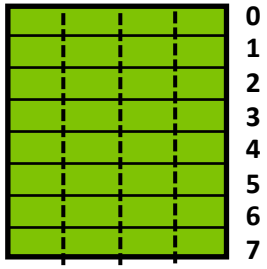
one location



(2-way) Set Associative

2 locations

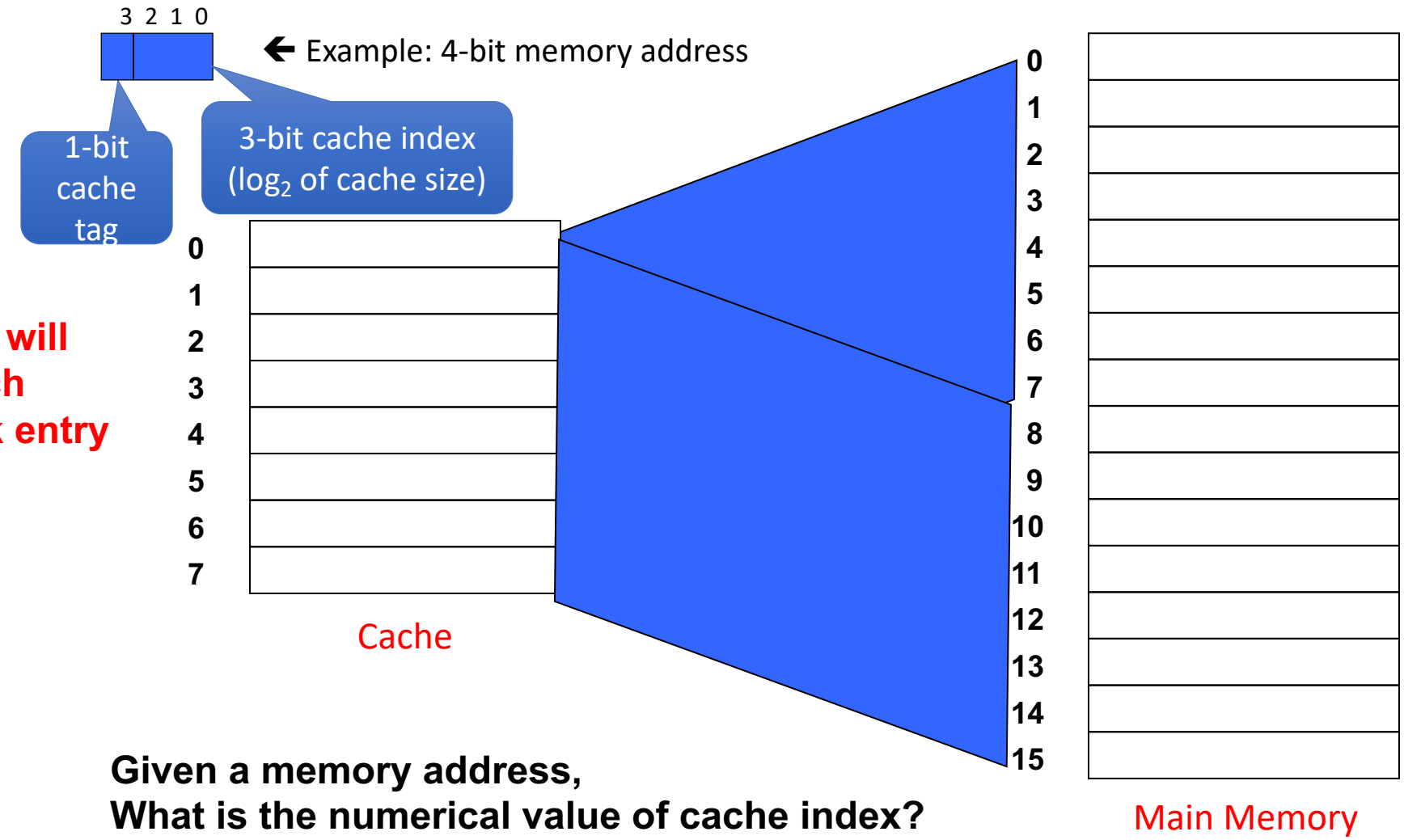
Searching is more expensive



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.

0	mem loc ~~0~~
1	mem loc 1
2	mem loc 2
3	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

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)

0	mem loc ~~0~~ 8
1	mem loc 1
2	mem loc 2
3	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

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

0	mem loc ~~0~~ ~~8~~ 0
1	mem loc 1
2	mem loc 2
3	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

Example

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

misses hits miss

3 2 1 0

9

← 4-bit memory address

9 = 1 0 0 1

3-bit index

9 MOD 8 is 1

0	mem loc ~~0~~ ~~8~~ 0
1	mem loc 1
2	mem loc 2
3	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	

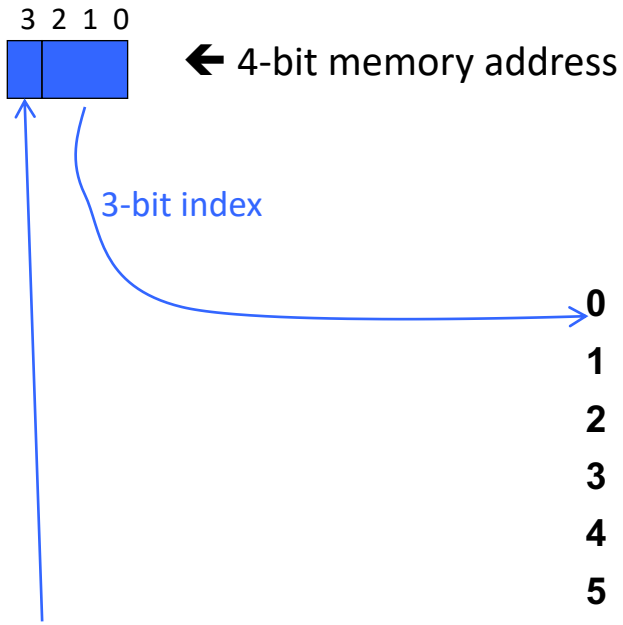
Memory

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



This remaining bit determines if cache data is for memory address 0 or 8

	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

Valid	Tag	Data

Metadata

Real data

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).

19% A. I just want the participation credit

16% B. 1 bit valid flag, 6 bit tag, 10 bit data

41% C. 1 bit valid flag, 10 bit tag, 16 bit data

25% D. 1 bit valid flag, 10 bit tag, 6 bit data

0% E. 2 bit valid flag, 12 bit tag, 16 bit data

Interpreting memory addresses

Memory address

Cache Tag	Cache Index

- index = memory addr mod cache size (for direct-mapped cache)
- In previous example with 8-entry cache:
  - 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

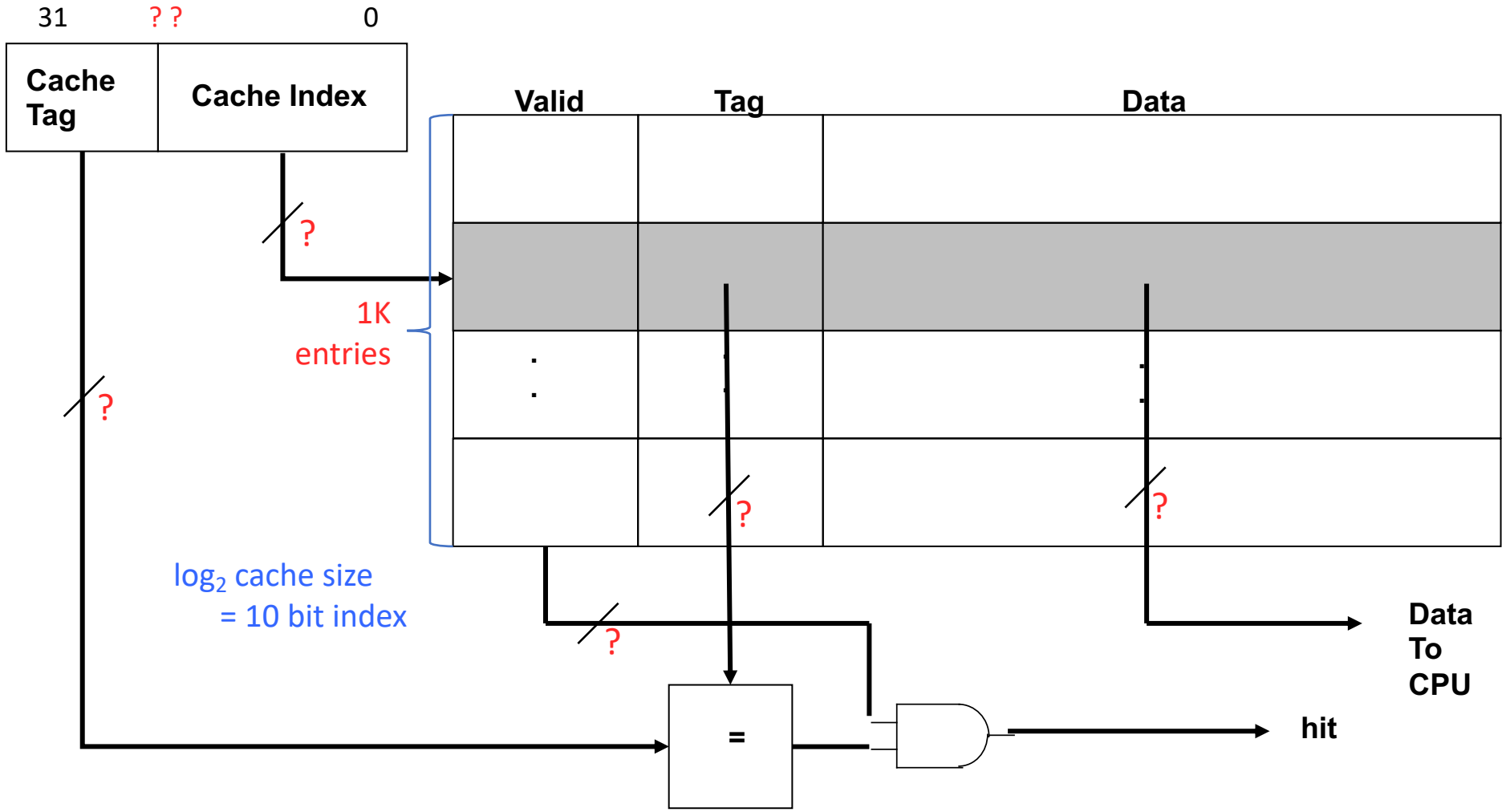
	valid	tag	data
0	1	1	loc ~~0~~ 1
1	0	1	loc ~~2~~ 3
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access location 3

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

➔ loss of spatial locality

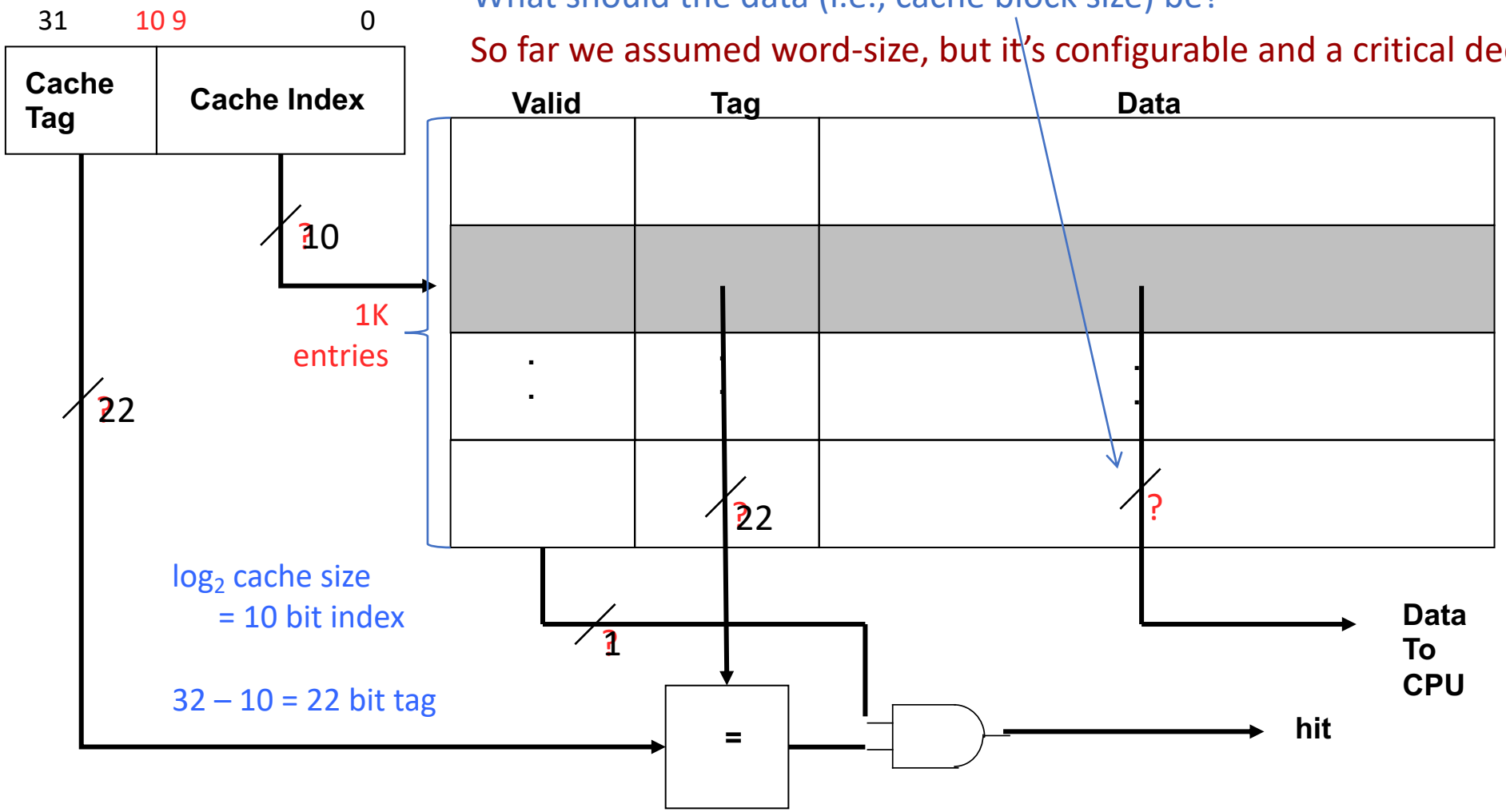
Hardware


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

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



$\log_2$ cache size = 10 bit index

32 – 10 = 22 bit tag

Example

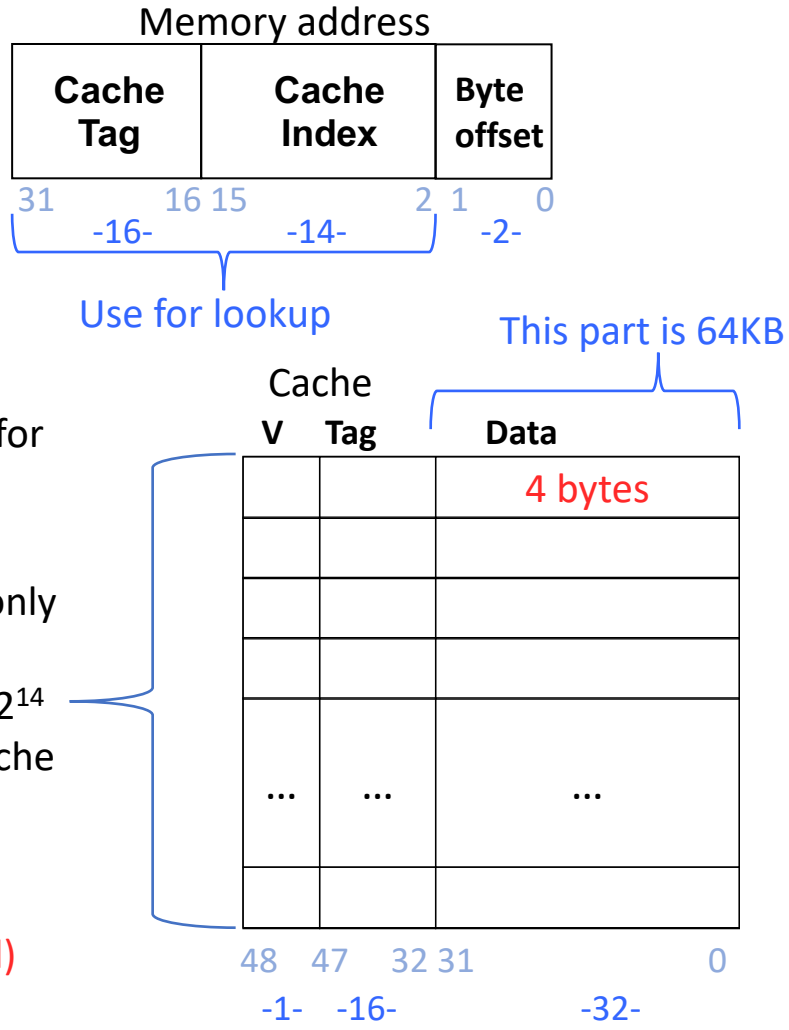
Let us consider the design of a direct-mapped cache for a realistic memory system.

- 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

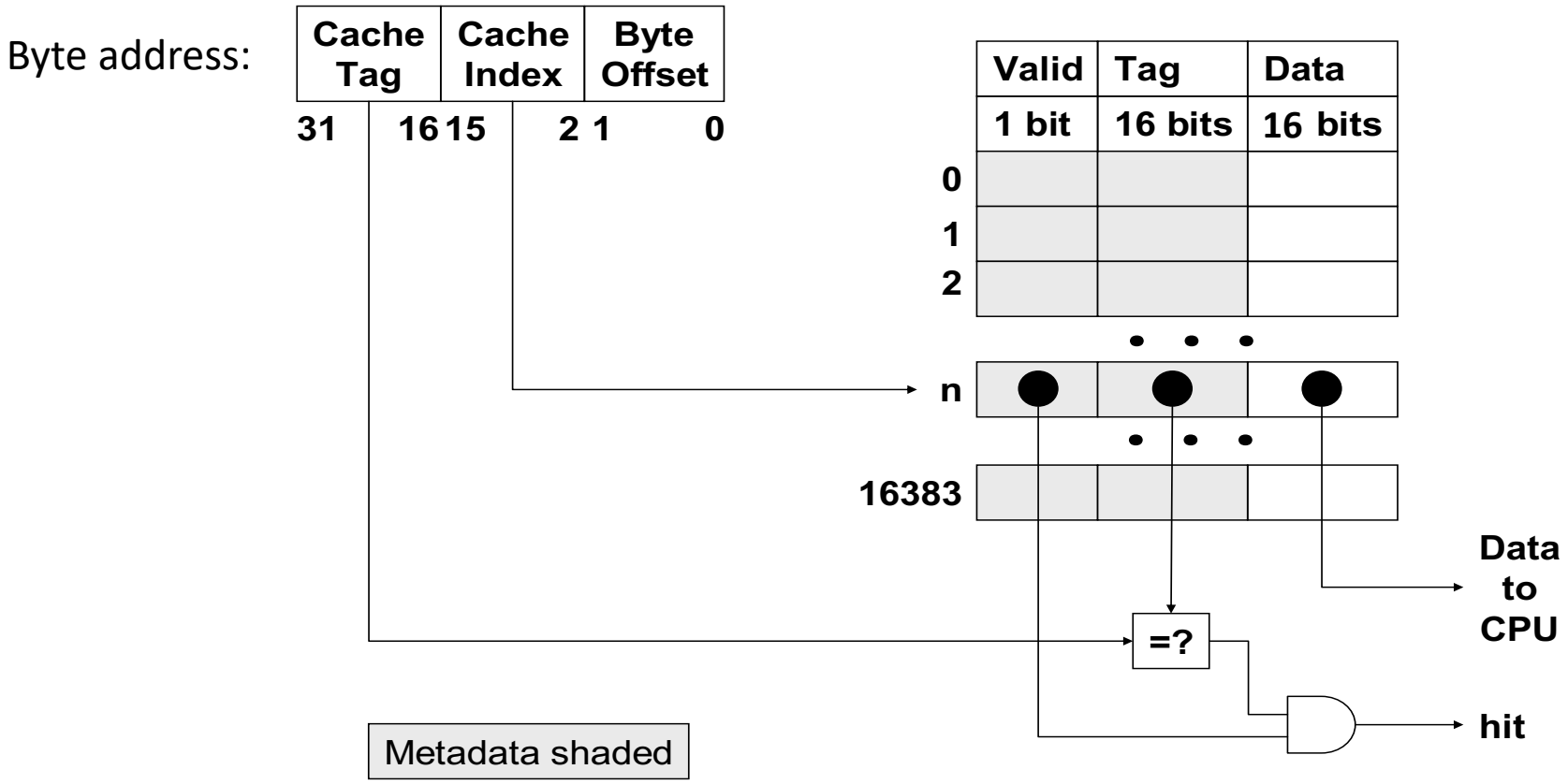
Consider only real data → $64k/4=2^{14}$ rows in cache

Metadata (overhead)



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

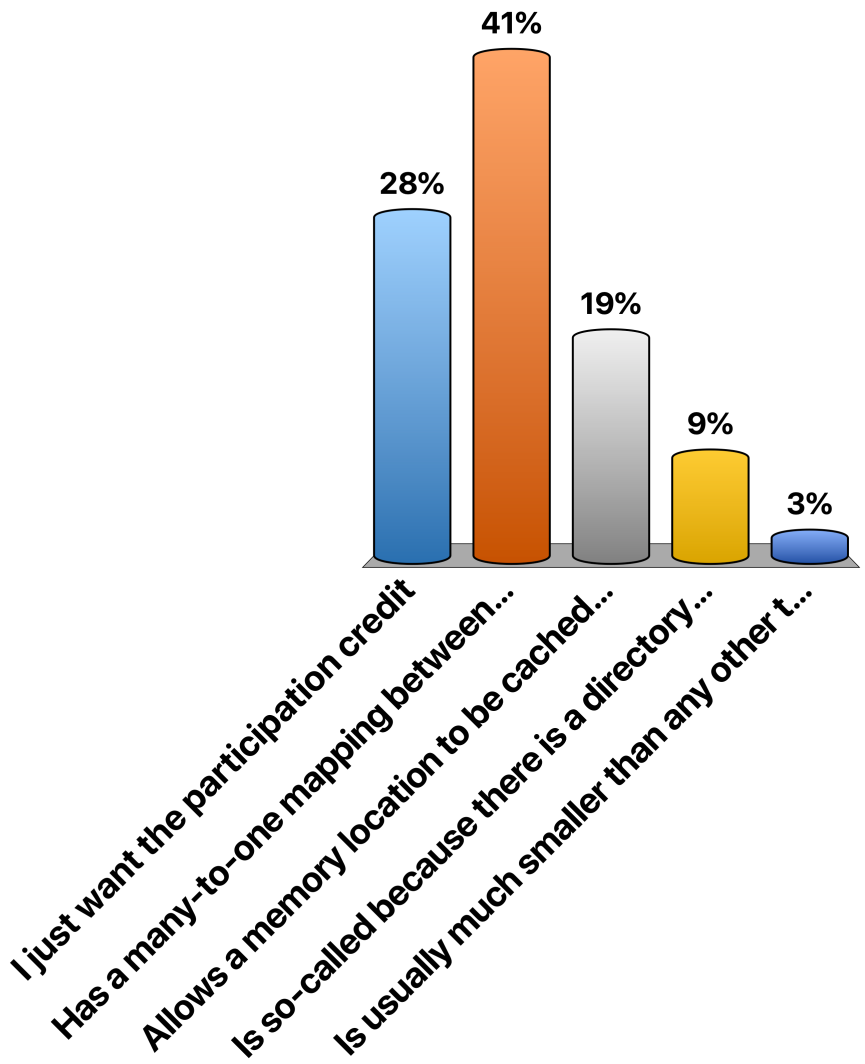
Memory address interpretation when single cache block contains multiple bytes



Session ID: dag2200

A direct-mapped cache

A. I just want the participation credit
B. Has a many-to-one mapping between memory locations 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

23% A. I just want the participation credit

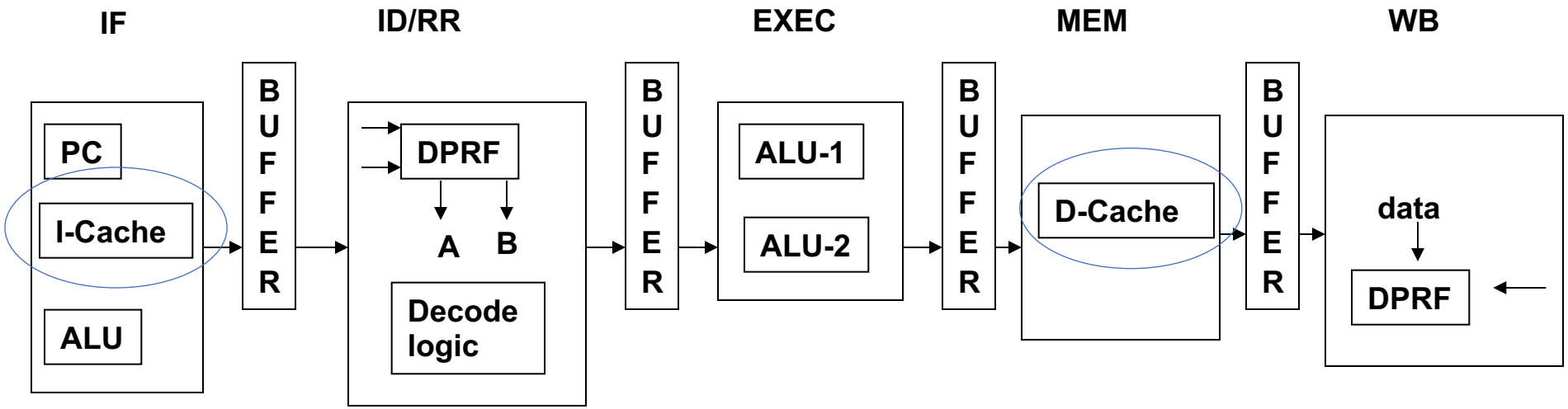
13% B. There is one 1-bit tag comparator for each cache line

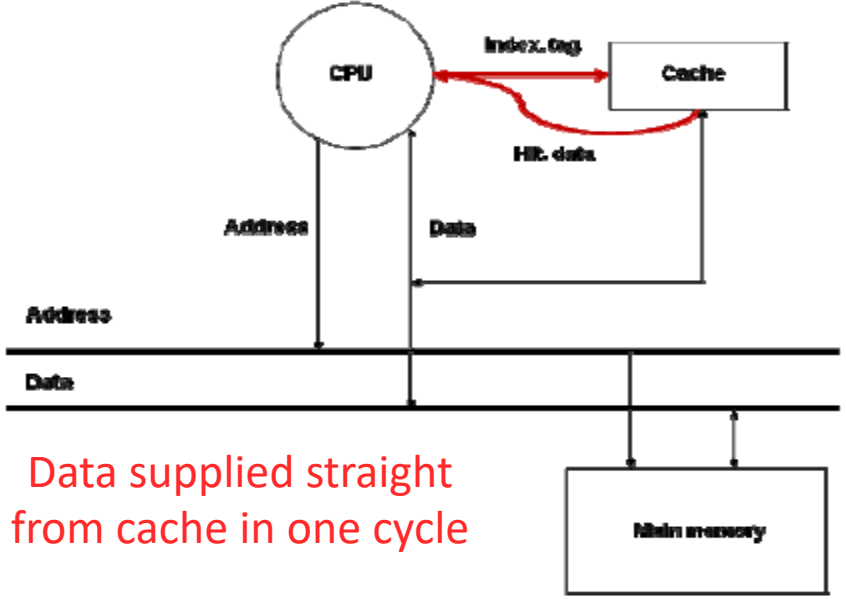
29% C. There is one t-bit tag comparator for each cache line

3% D. There is one 1-bit tag comparator for the entire cache

32% E. There is one t-bit tag comparator for the entire cache

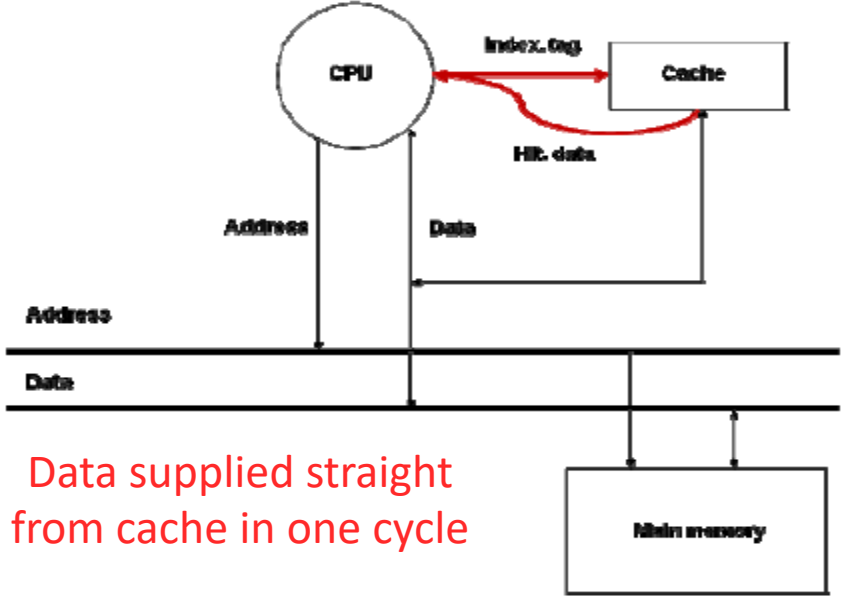
Pipelined processor with caches





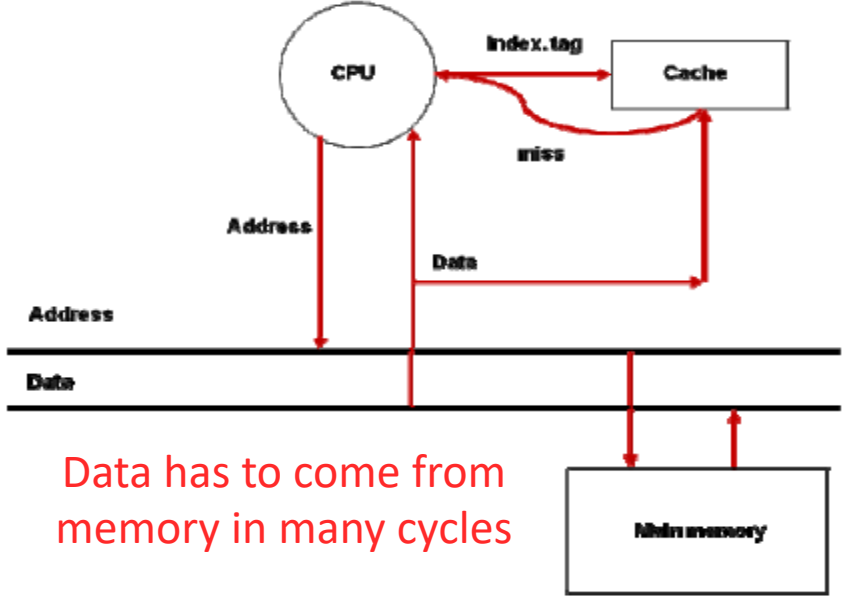
Data supplied straight from cache in one cycle

(a) Read Hit



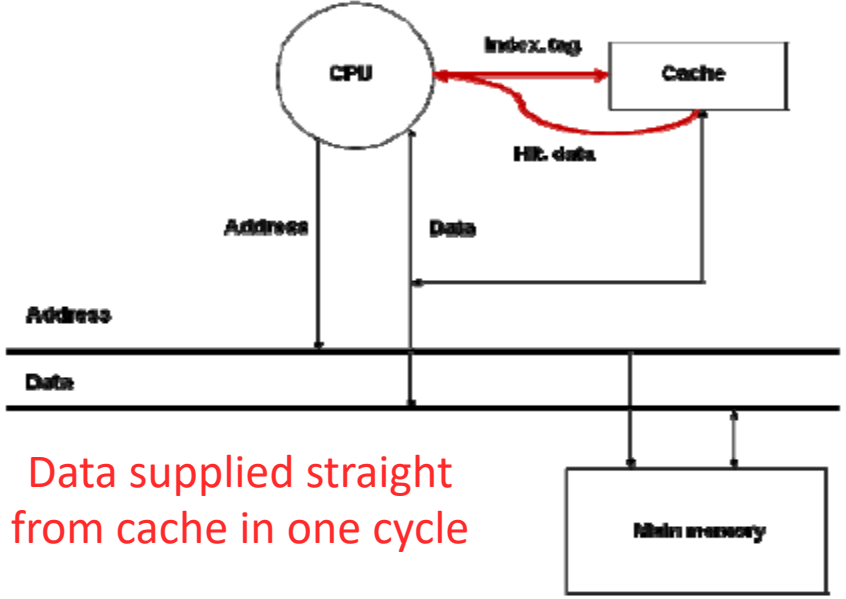
Data supplied straight from cache in one cycle

(a) Read Hit



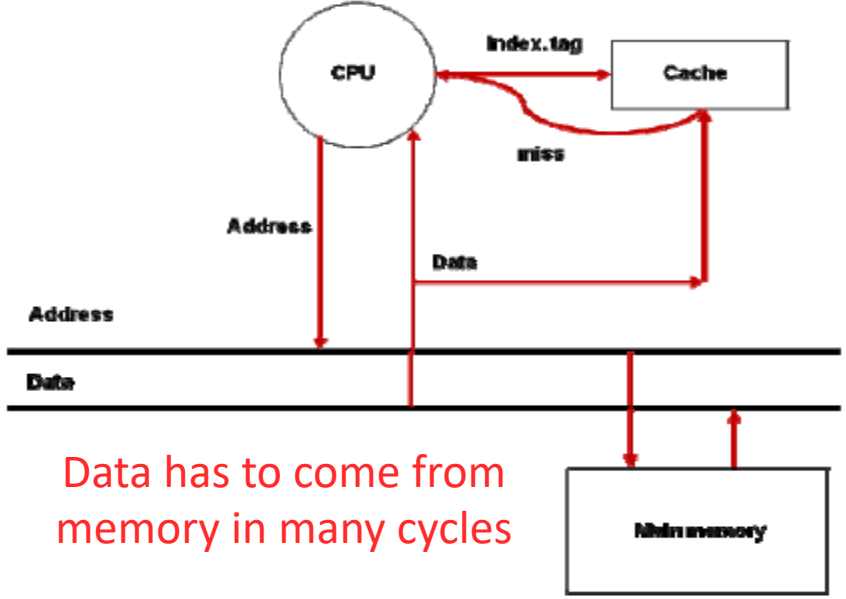
Data has to come from memory in many cycles

(b) Read miss



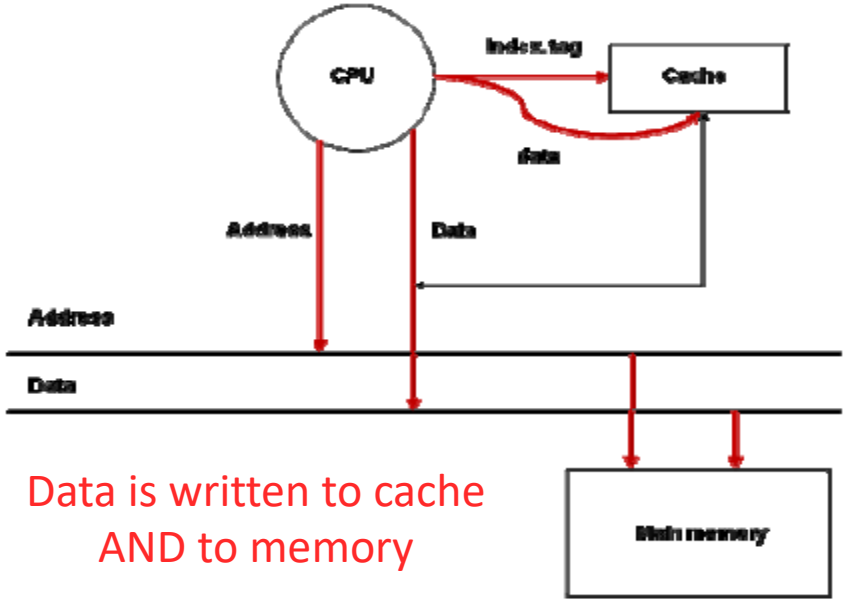
Data supplied straight from cache in one cycle

(a) Read Hit



Data has to come from memory in many cycles

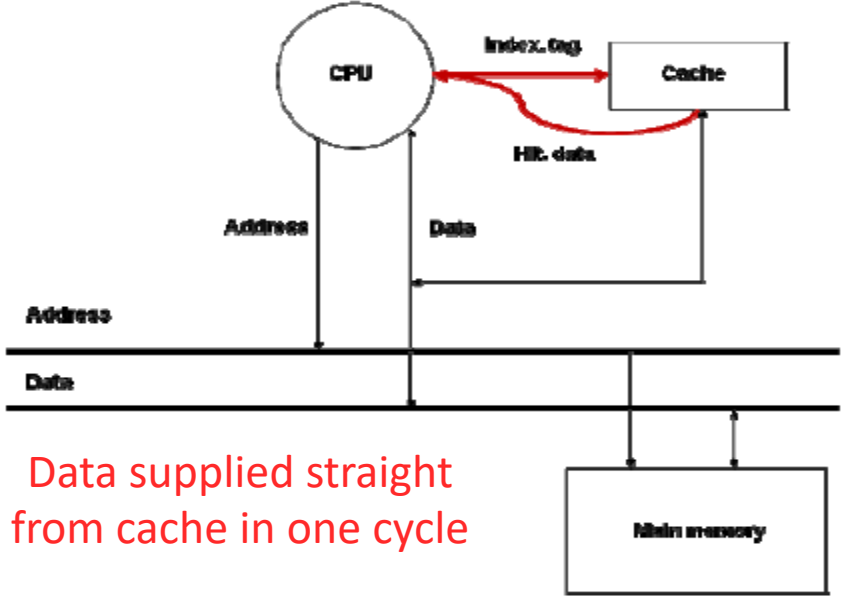
(b) Read miss



Data is written to cache AND to memory

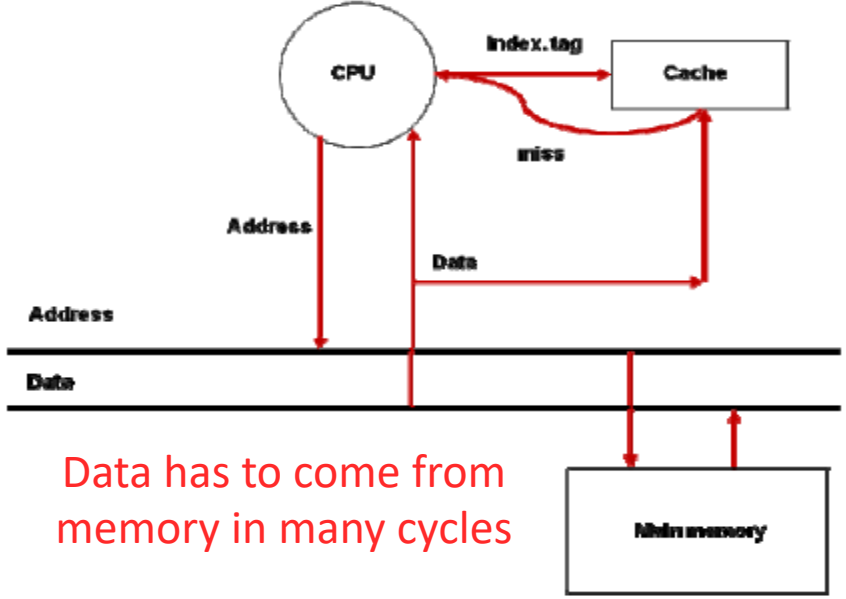
(c) Write-through

Different write policies



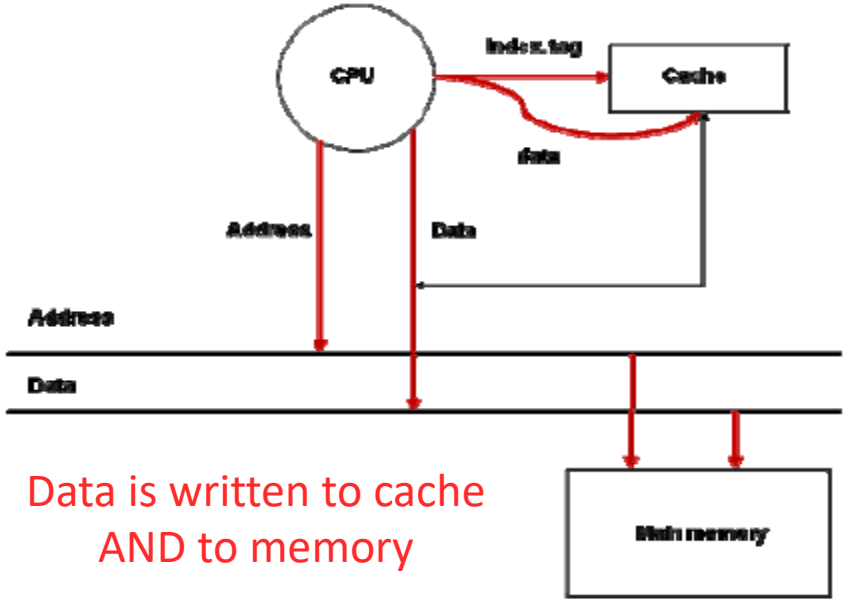
Data supplied straight from cache in one cycle

(a) Read Hit



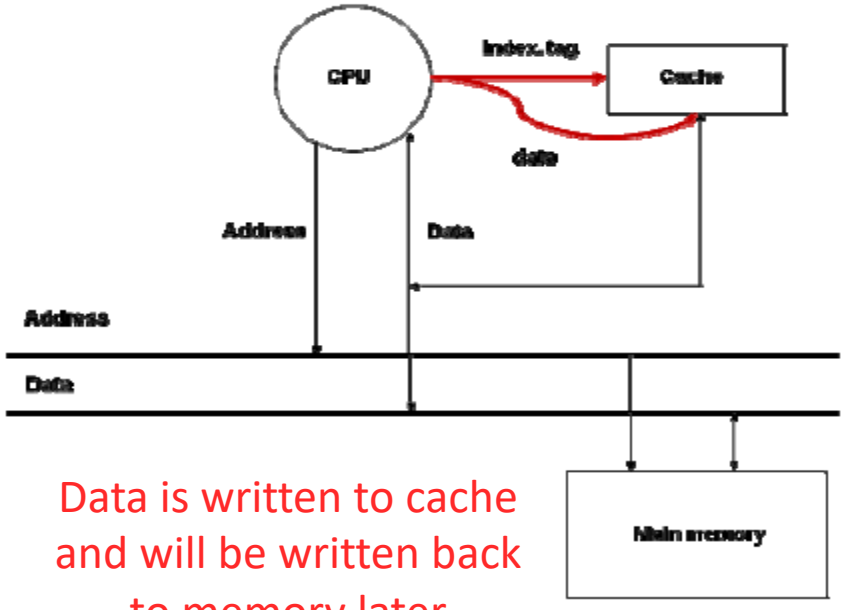
Data has to come from memory in many cycles

(b) Read miss



Data is written to cache AND to memory

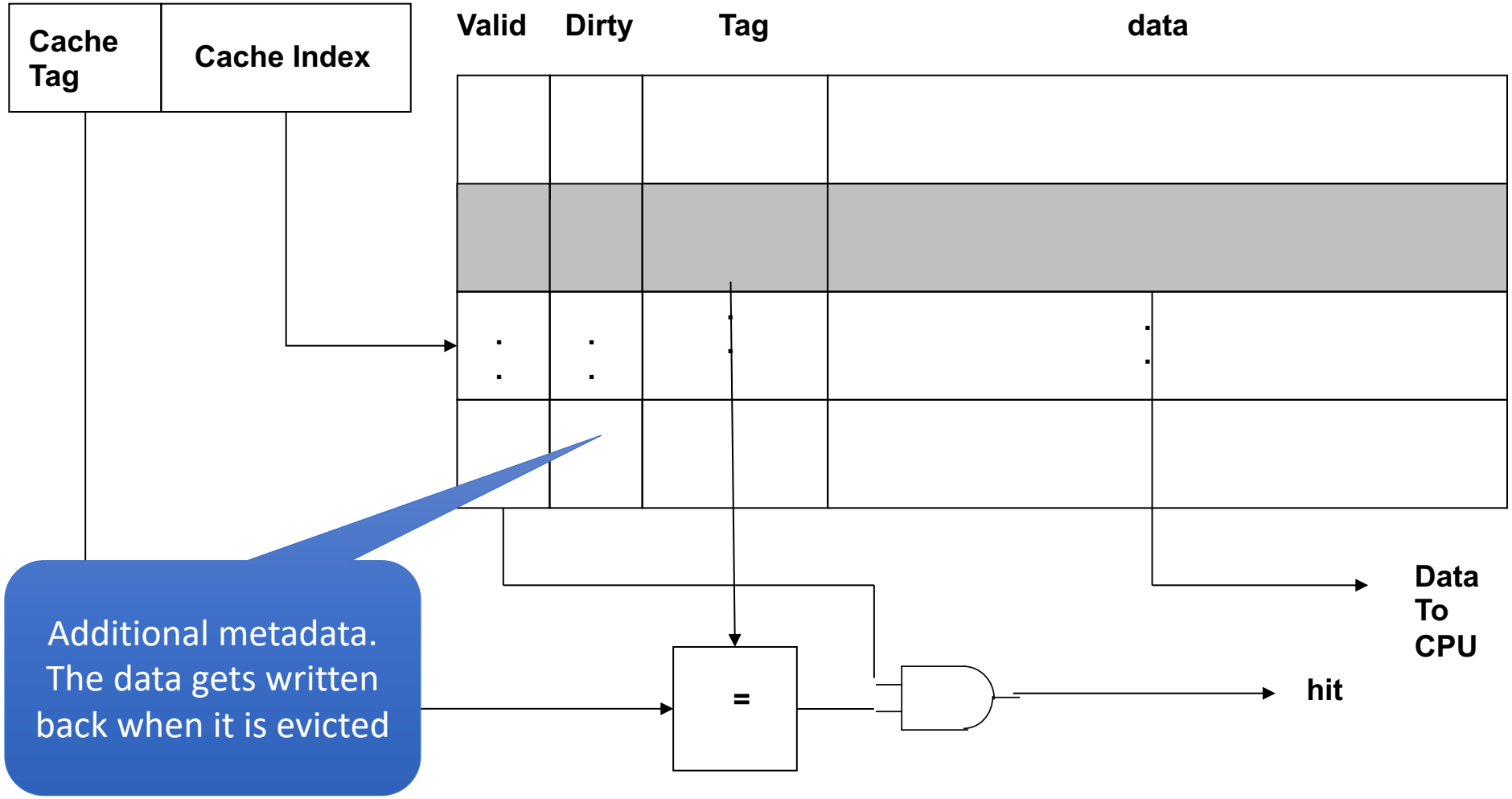
(c) Write-through



Data is written to cache and will be written back to memory later

(d) Write-back

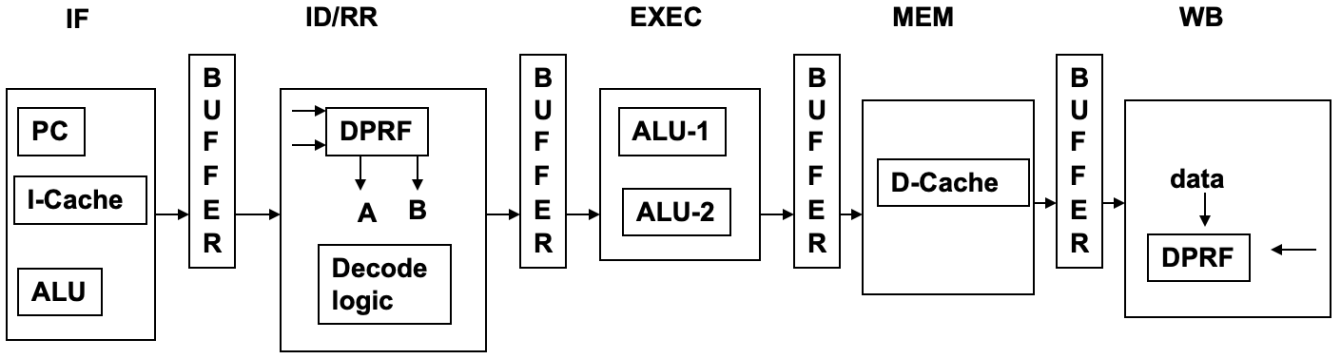
Write-back cache



Read miss stalls

```
I1: ld  r1,a        ;r1 <- memory at a
I2: add r3,r4,r5    ;r3 <- r4 + r5
I3: and r6,r7,r8    ;r6 <- r7 & r8
I4: add r2,r4,r5    ;r2 <- r4 + r5
I5: add r2,r1,r2    ;r2 <- r1 + r2
```

- We can treat a read-miss in MEM in a similar fashion as we did previously with registers and the busy bits
- MEM can reset the busy bit for r1 when it sees the read complete for I1 (instead of waiting for WB as we did before)



Execution time with caches

~~Execution time = N * CPI_{Avg} * cycle time~~

$CPI_{eff} = CPI_{Avg} + \text{Memory-stalls}_{Avg}$

Execution time = N * CPI_{eff} * cycle time

Execution time = N * (CPI_{Avg} + M-stalls_{Avg}) * cycle time

$\text{Memory-stalls}_{Avg}$ = $\text{misses per instruction}_{Avg}$ * $\text{miss-penalty}_{Avg}$

Total memory stalls = N * $\text{Memory-stalls}_{Avg}$

Session ID: dag2200

The effective CPI is…

Average CPI = 1.5

Average cache miss per instruction = 3%

Miss penalty = 20

21% A. I just want the participation credit

18% B. 1.8

50% C. 2.1

6% D. 21.5

6% E. 7.5

$CPI_{eff} = 1.5 + (3\% * 20) = 1.5 + 0.6 = 2.1$ CPI

Example

- Consider a pipelined processor that has an average CPI of 1.8 without accounting for memory stalls.
  - I-Cache has a hit ratio of 95%
  - D-Cache has a hit ratio of 98%.
- Assume that memory reference instructions account for 30% of all the instructions executed.
  - 80% are loads
  - 20% are stores
- On average
  - read-miss penalty is 20 cycles
  - write-miss penalty is 5 cycles.

Compute the effective CPI of the processor accounting for the memory stalls.

Solution

Cost of instruction misses:

= I-cache miss ratio * read miss penalty

= (1 – 0.95) * 20 = 1 cycle per instruction

Cost of data read misses:

= % memory reference instructions

* fraction that are loads * D-cache miss ratio * read miss penalty

= 0.3 * 0.8 * (1 – 0.98) * 20 = 0.096 cycles per instruction

Cost of data write misses:

= % memory reference instructions

* fraction that are stores * D-cache miss ratio * read miss penalty

= 0.3 * 0.2 * (1 – 0.98) * 5 = 0.006 cycles per instruction

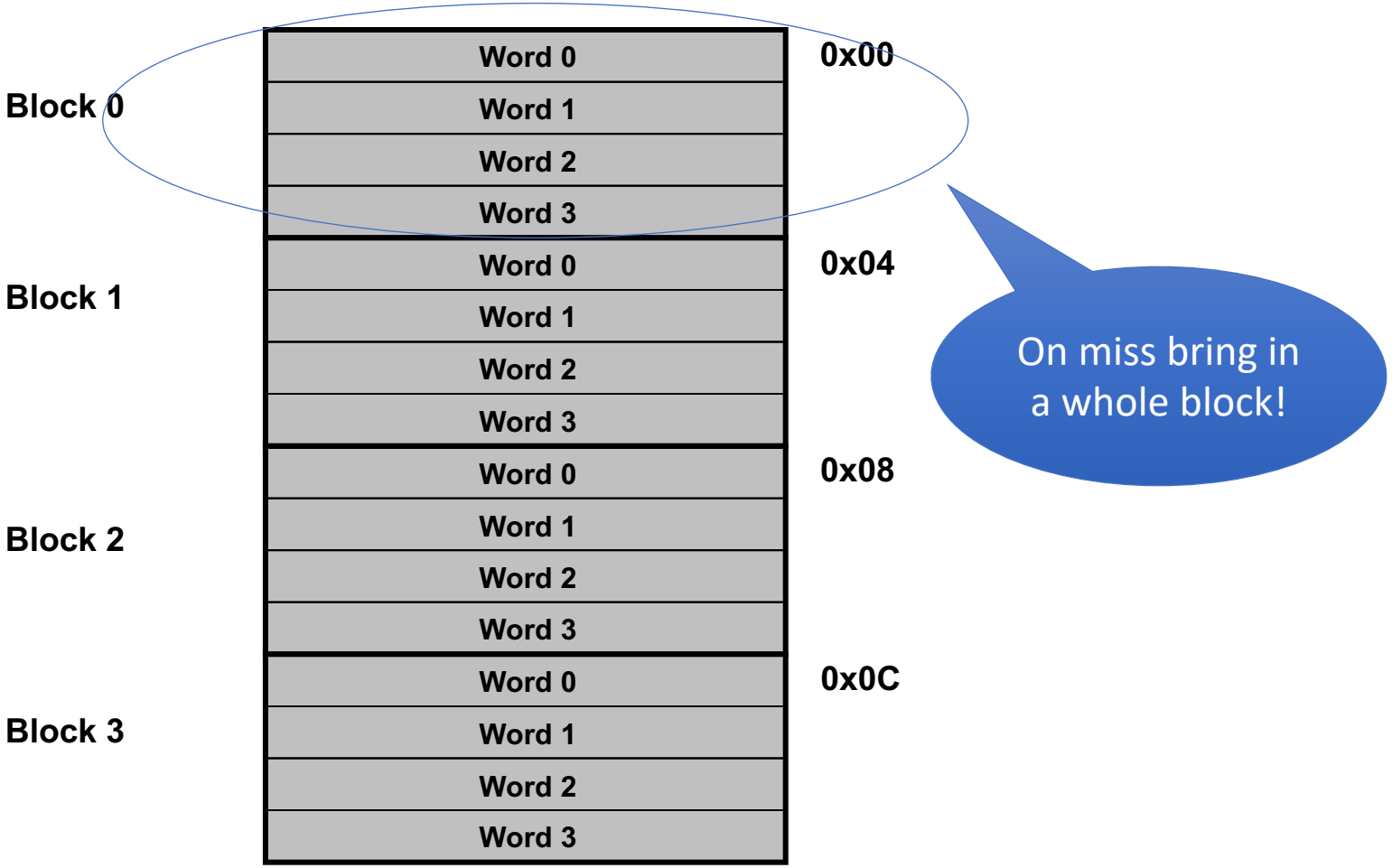
$CPI_{eff} = CPI_{avg}$ + cost of I-cache misses + cost of D-cache misses

= 1.8 + 1 + (0.096 + 0.006) = 2.902

How to improve cache efficiency

- Exploit spatial locality
  - Bring more from memory into cache at a time
- Better organization
  - Exploit working set concept

Spatial Locality



(Re)Interpreting memory address

S = Size of cache; B = Block size; L = lines in cache

Cache Tag	Cache Index	Block Offset
t	n	b

$b = log_2 B$

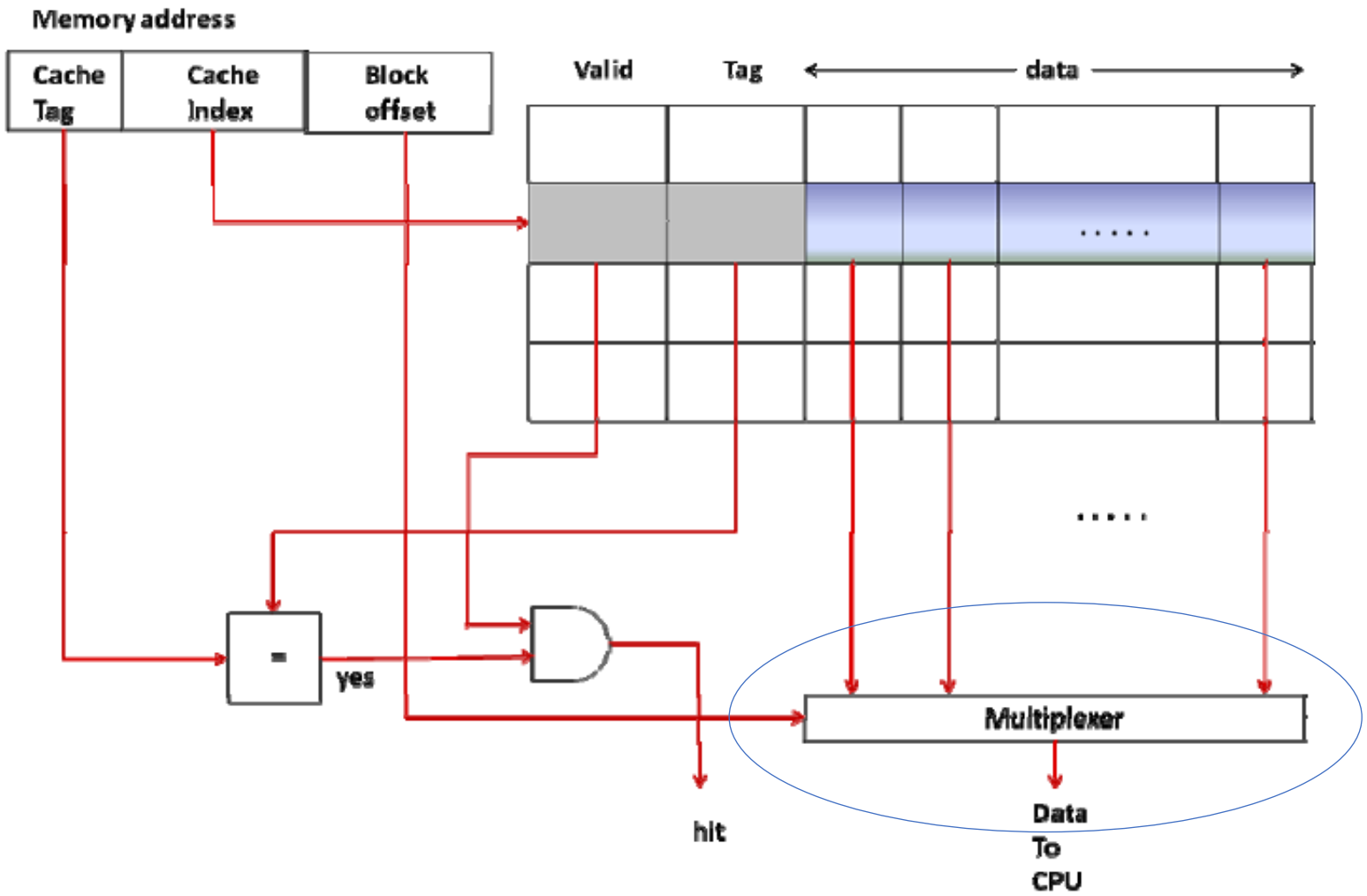
$L = S/B$

$n = log_2 L$

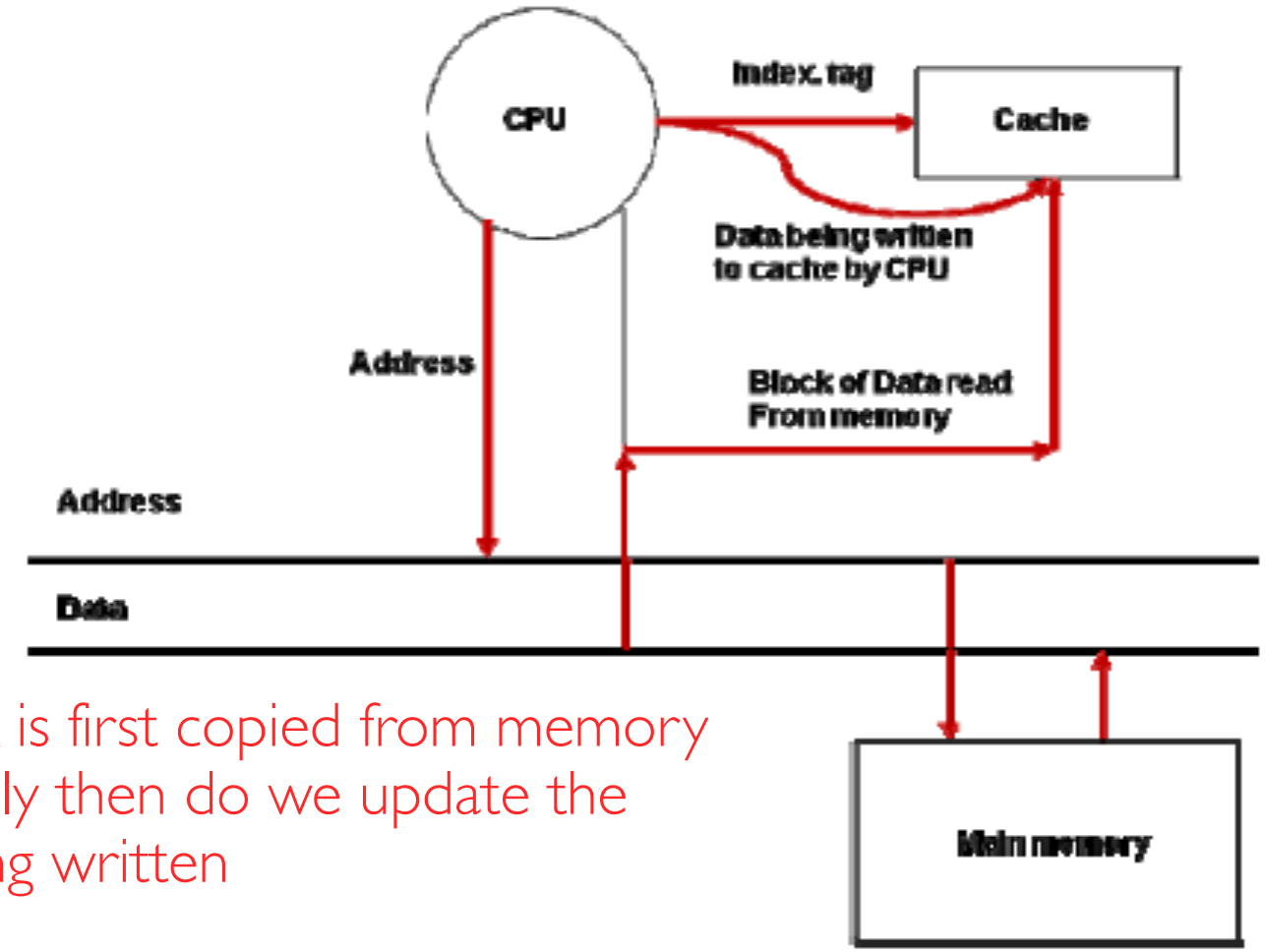
$t = a - (b+n)$

This is really just the word (or byte) number within the block

Multi-word cache organization



Write miss with multi-word cache block



The missing block is first copied from memory into the cache; only then do we update the specific word being written

Multi-word cache block example

Direct-mapped cache

- 32-bit byte-addressable memory address
- Each memory word contains 4 bytes
- Block size = 4 words (16 bytes)
- A memory access brings in a block
- 64K byte write-back cache