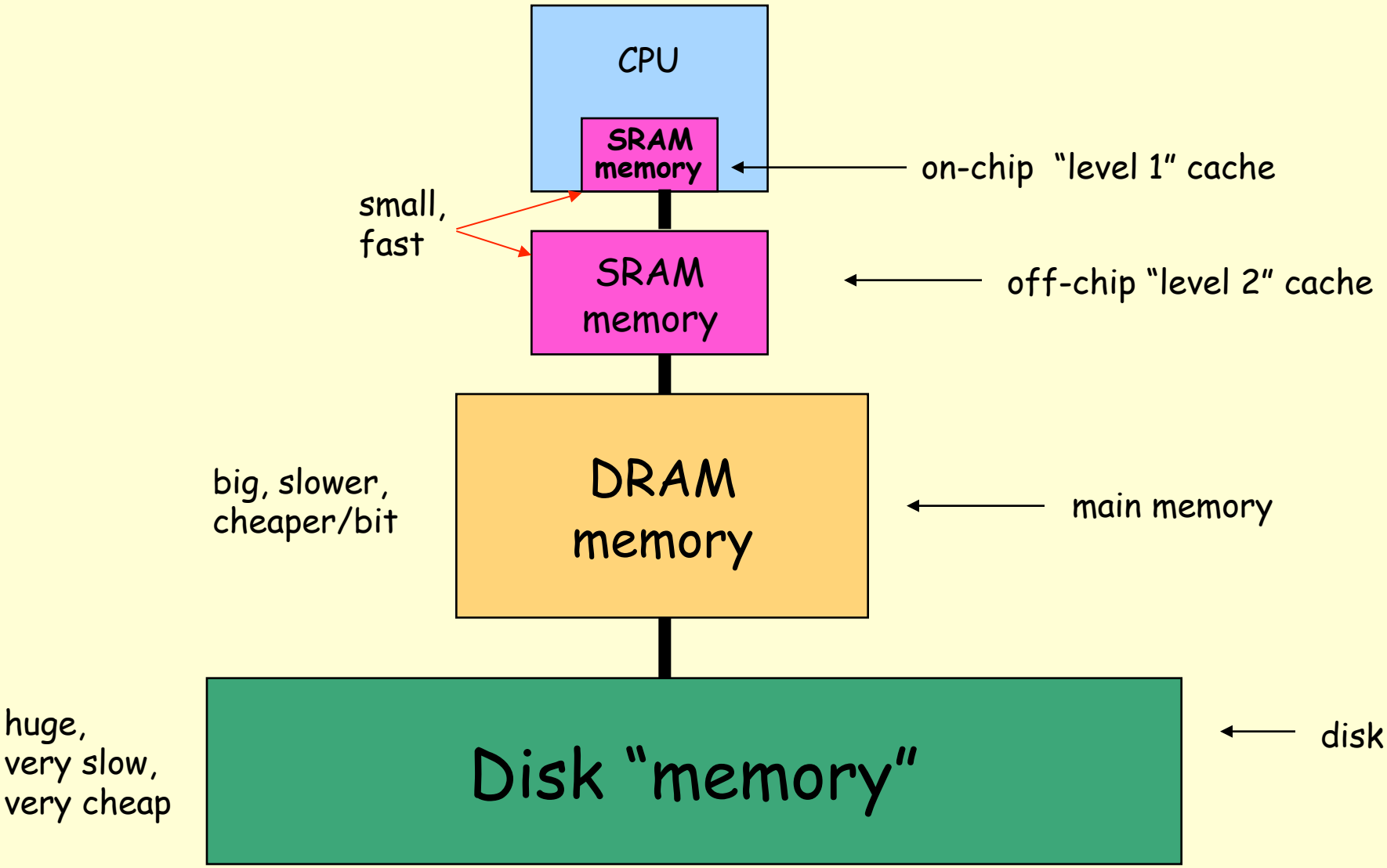
Caching

A typical memory hierarchy



Mental Simulation: Assumptions

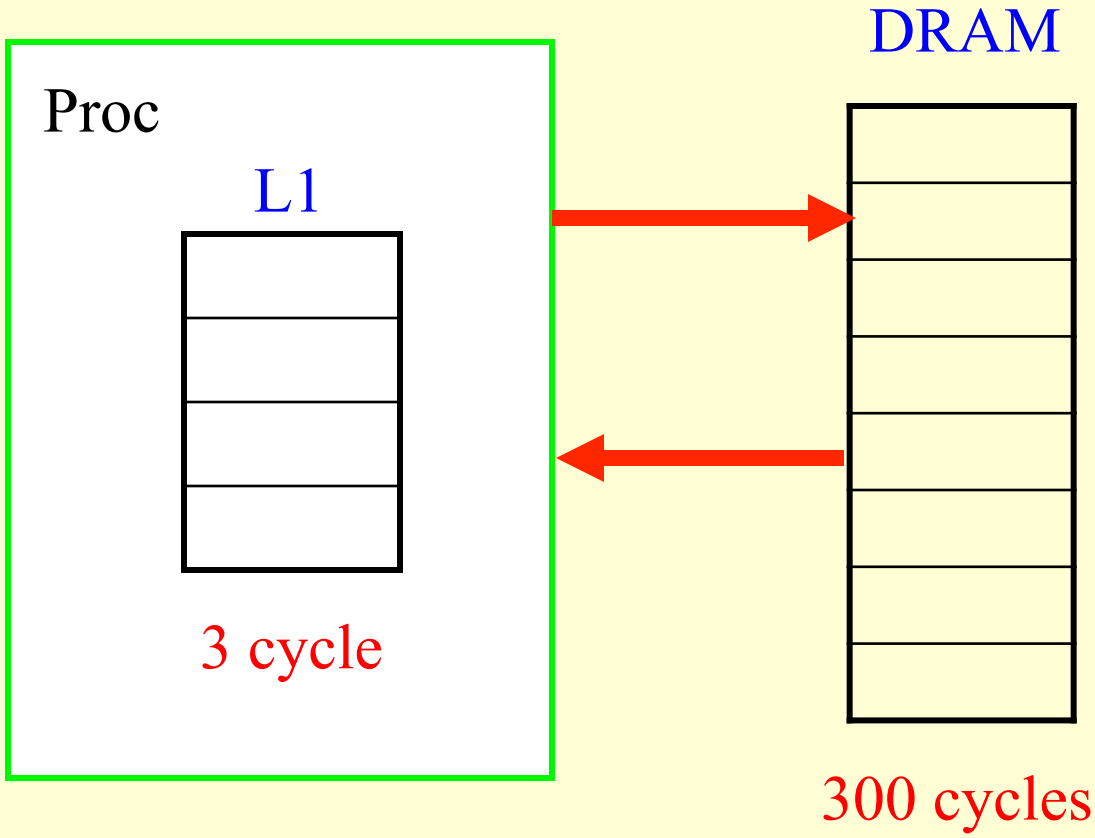
- Our main memory can hold 8 words
  - 1 word is one data element (an integer)
    - 32 bits (4 bytes) in one word
    - Each data element starts on an address that is a multiple of 4
  - Our data will be at addresses: 0, 4, 8, 12, 16, 20, 24, 28
  - 300 Cycles to access main memory
- Cache which can hold 4 words (data elements)
  - 3 cycles to access cache
  - We'll look at a few different "designs" of cache

Program1: avg 4 elems, print 4 elems

int data[8] = {1,2,3,4,5,6,7,8}

Memory Access Pattern:

data[0]
data[1]
data[2]
data[3]
data[0]
data[1]
data[2]
data[3]

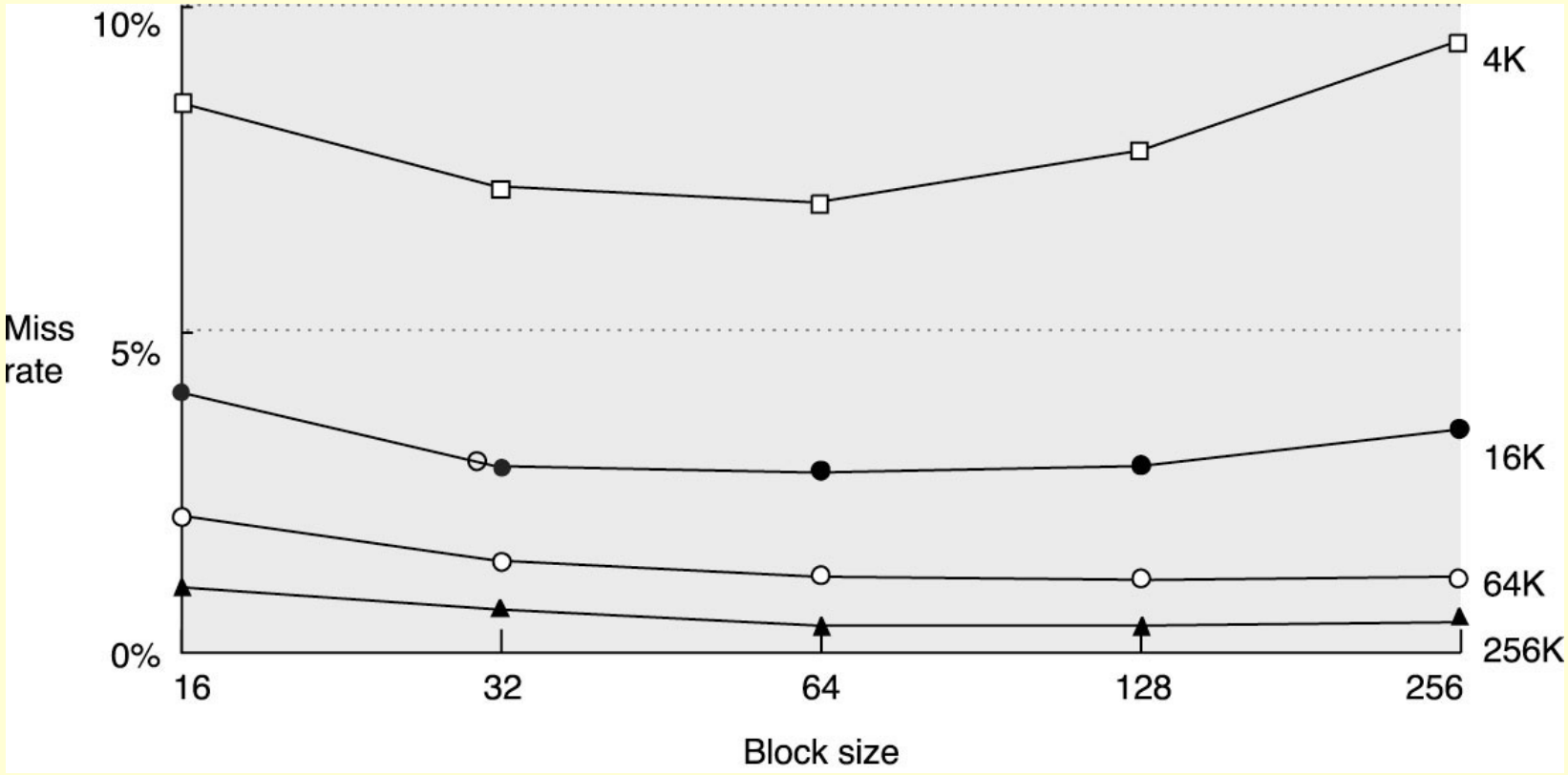


Effects of Cache Size

Size	I Cache	Data Cache	Unified Cache
1 KB	3.06%	24.61%	13.34%
2 KB	2.26%	20.57%	9.78%
4 KB	1.78%	15.94%	7.24%
8 KB	1.10%	10.19%	4.57%
16 KB	0.64%	6.47%	2.87%
32 KB	0.39%	4.82%	1.99%
64 KB	0.15%	3.77%	1.35%
128 KB	0.02%	2.88%	0.95%

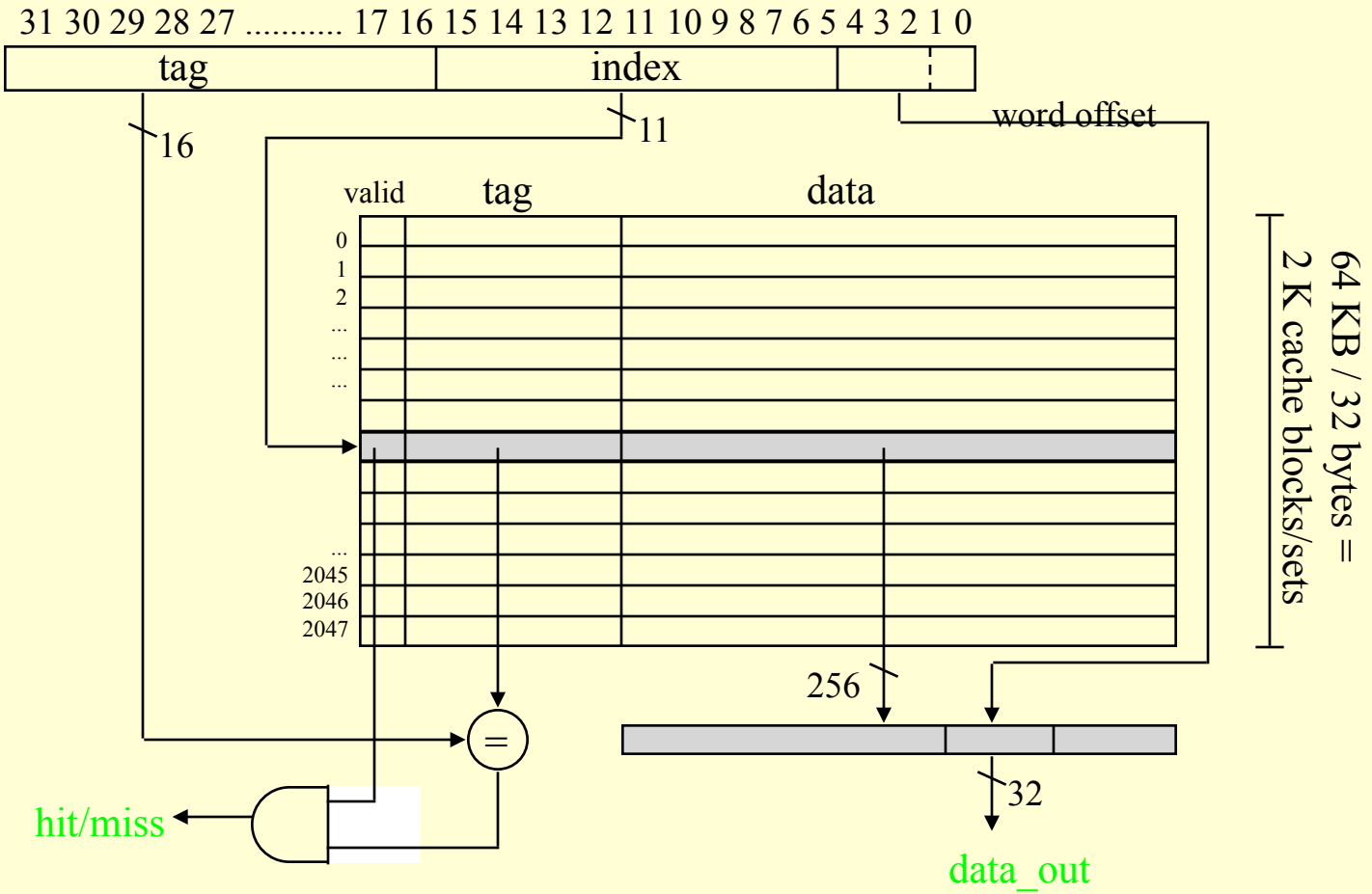
Miss Rates

Effects of Block Size



Accessing a Direct Mapped Cache

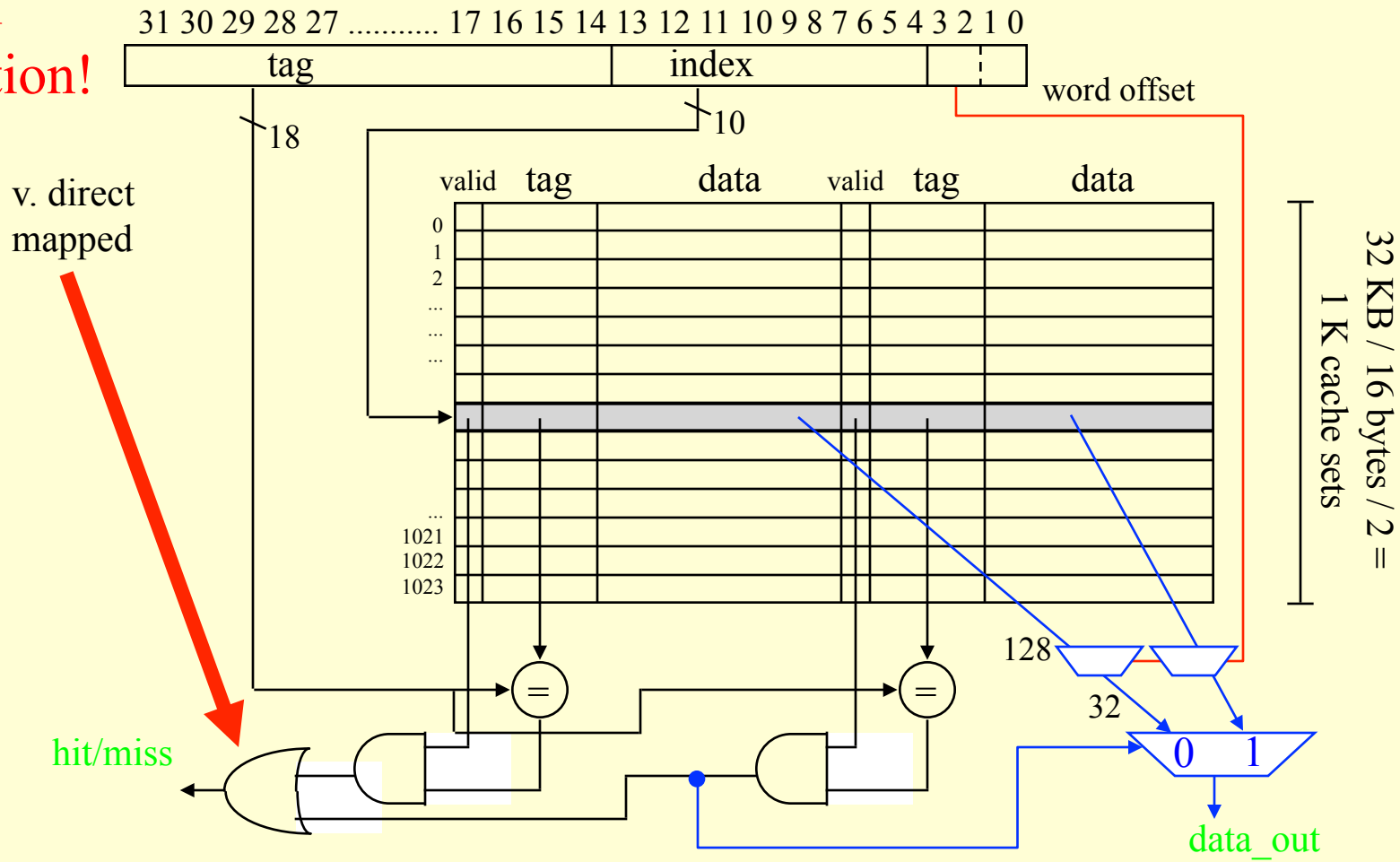
- 64 KB cache, direct-mapped, 32-byte cache block size



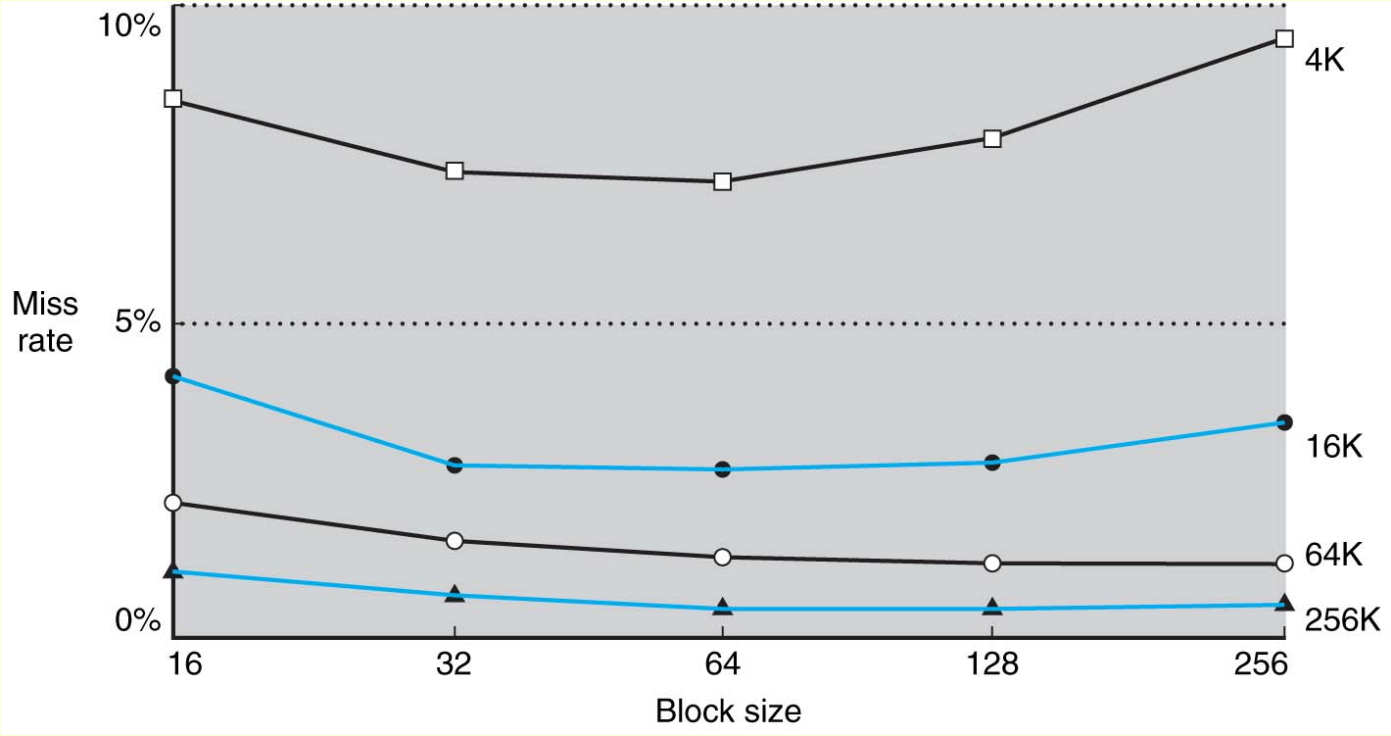
Accessing a 2-way Assoc Cache – Hit Logic

- 32 KB cache, 2-way set-associative, 16-byte block size

Exam Question!



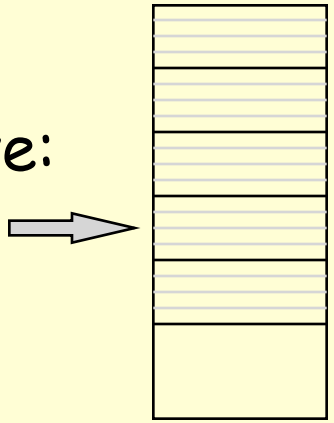
Effects of Cache Associativity



“Sweet Spot” of 32-64 bytes per block

Which Block Should be Replaced on a Miss?

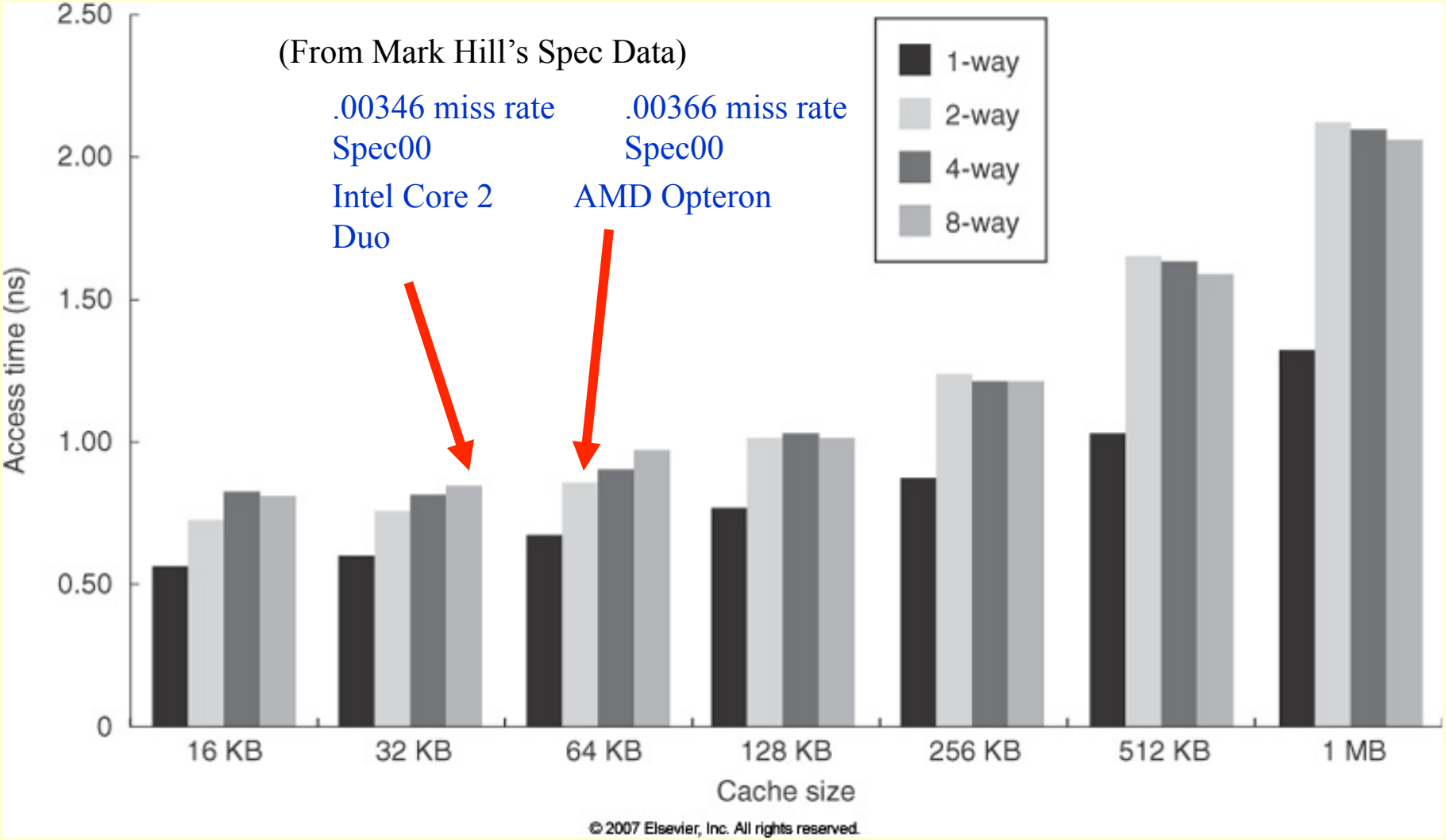
- Direct Mapped is Easy
- Set associative or fully associative:
  - "Random" (large associativities)
  - LRU (smaller associativities)
  - Pseudo Associative



Associativity:	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.18%	5.69%	4.67%	5.29%	4.39%	4.96%
64 KB	1.88%	2.01%	1.54%	1.66%	1.39%	1.53%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Numbers are averages across a set of benchmarks. Performance improvements vary greatly by individual benchmarks.

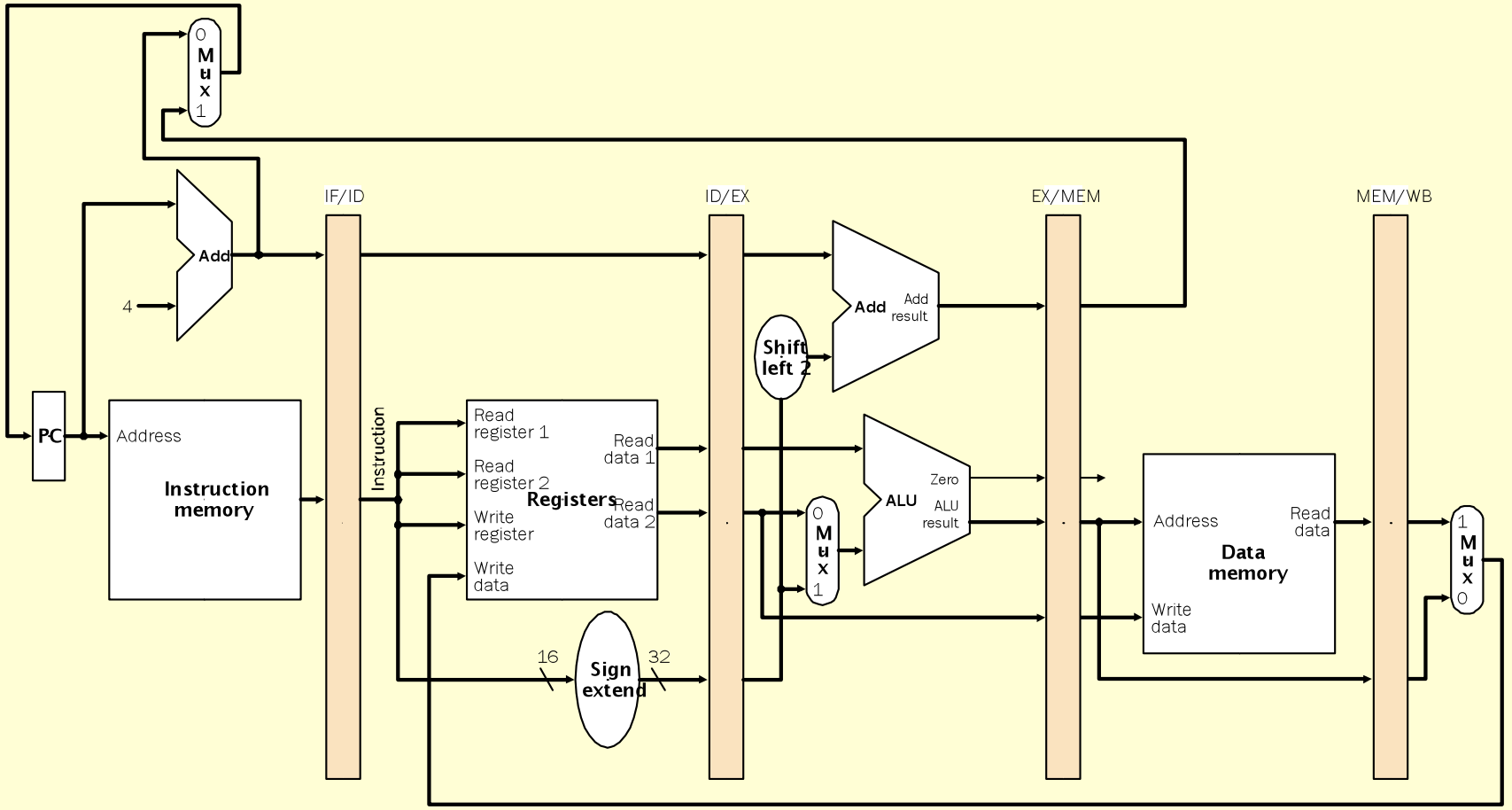
Cache Size and Associativity versus Access Time



90 nm, 64-byte clock, 1 bank

How does this affect the pipeline?

add $10, $1, $2 **sub $11, $8, $7** **lw $8, 50($3)** **add $3, $10, $11** Write Back



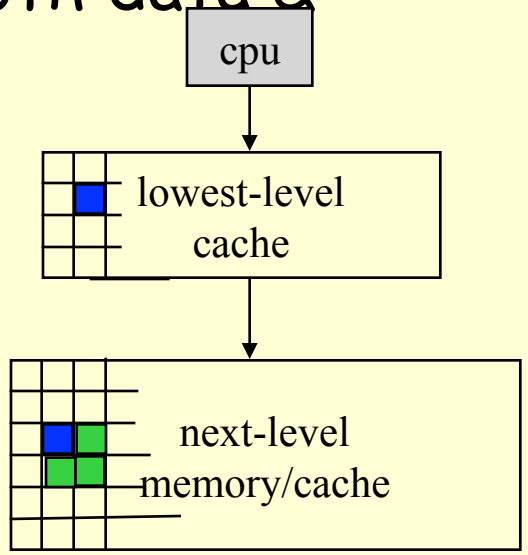
Cache Vocabulary

- cache hit: an access where data is already in cache
- cache miss: an access where data isn't in cache
- Hit time: time to access the cache
- miss penalty: time to move data from further level to closer, then to cpu
- hit rate: percentage of accesses that the data is found in the cache
- miss rate: (1 - hit rate)
- replacement policy
- write-back v. write-through

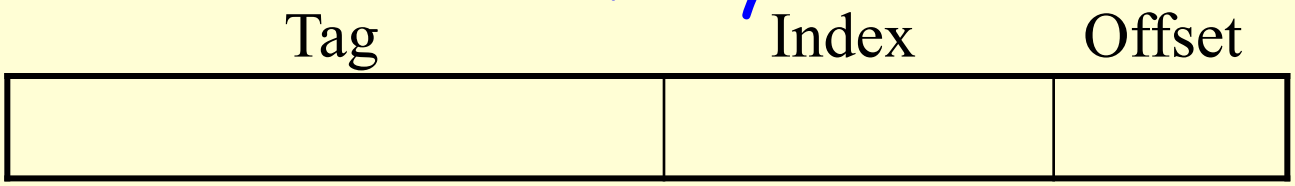
Cache Vocabulary

- cache block size or cache line size: the amount of data that gets transferred on a cache miss.
- instruction cache (I-cache): cache that can only hold instructions.
- data cache (D-cache): cache that can only hold data.
- unified cache: cache that holds both data & instructions.

A typical processor today has separate "Level 1" I- and D-caches on the same chip as the processor (and possibly a larger, unified "L2" on-chip cache), and larger L2 (or L3) unified cache on a separate chip.



Comparing anatomy of an address: How many bits?



Direct Mapped:
Cache Line Size = 4 bytes
32 lines in cache

What would the "size" of the cache be? (in bytes)

A) 32
B) 32+(25/8)*32
C) 128
D) 128 + (25/8) * 32

You have a 2-way set associative cache which is LRU, has 32 byte lines and is 512 B. The word size is 4 bytes.

Assuming a cold start, what is the state of the cache after the following sequence of accesses?

0, 32, 64, 128, 512, 544, 768, 1024, ..

(see more complex problem as well)

****

Issues we touched on

- How data moves from memory to cache
  - Benefits: Temporal locality
- What to do when cache is full
  - Replacement policies
- Placement options for where data can "go" in cache
  - Direct-mapped, Set-associative, Fully-associative
- Moving "lines"/"blocks" into cache
  - Benefit: Spatial locality
- Writing values in a code
  - Cache/MM out of synch with registers
  - Write back policies in caches

Cache basics

- In running program, main memory is data's "home location".
  - Addresses refer to location in main memory.
  - "Virtual memory" allows disk to extend DRAM
    - Address more memory than you actually have (more later)
- When data is accessed, it is automatically moved up through levels of cache to processor
  - "lw" uses cache's copy
  - Data in main memory may (temporarily) get out-of-date
    - How?
    - But hardware must keep everything consistent.
  - Unlike registers, cache is not part of ISA
    - Different models can have totally different cache design

The principle of locality

Memory hierarchies take advantage of memory locality.

- The principle that future memory accesses are *near* past accesses.

Two types of locality:

- _____ locality - near in time: we will often access the *same* data again very soon
- _____ locality - near in space/distance: our next access is often very close to recent accesses.

This sequence of addresses has both types of locality

0, 4, 8, 0, 4, 8, 32, 32, 256, 36, 40, 32, 32…

How does HW decide what to cache?

Taking advantage of temporal locality:

bring data into cache whenever its referenced

kick out something that hasn't been used recently

Taking advantage of spatial locality:

bring in a block of contiguous data (cacheline), not just the requested data.

Some processors have instructions that let software influence cache:

Prefetch instruction ("bring location x into cache")

"Never cache x" or "keep x in cache" instructions

Cache behavior simulation

- Access globals frequently
- Sum Array
- Access the stack
- Compare Two Strings
- Search a linked list for the first time
- Repeatedly search a linked list
- Traverse a tree/graph
- Multiply a large matrix with power of 2 dims

…

Cache Issues

On a memory access -

- How does hardware know if it is a hit or miss?

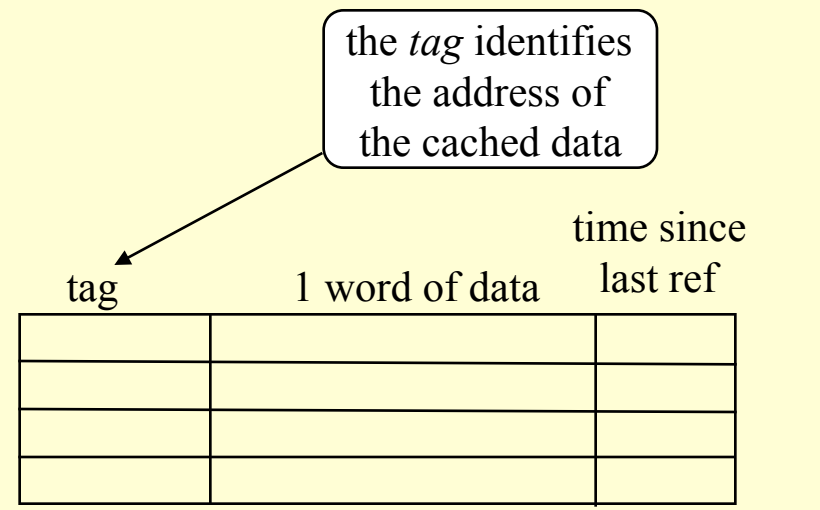
On a cache miss -

- where to put the new data?
- what data to throw out?
- how to remember what data is where?

A simple cache

address trace:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



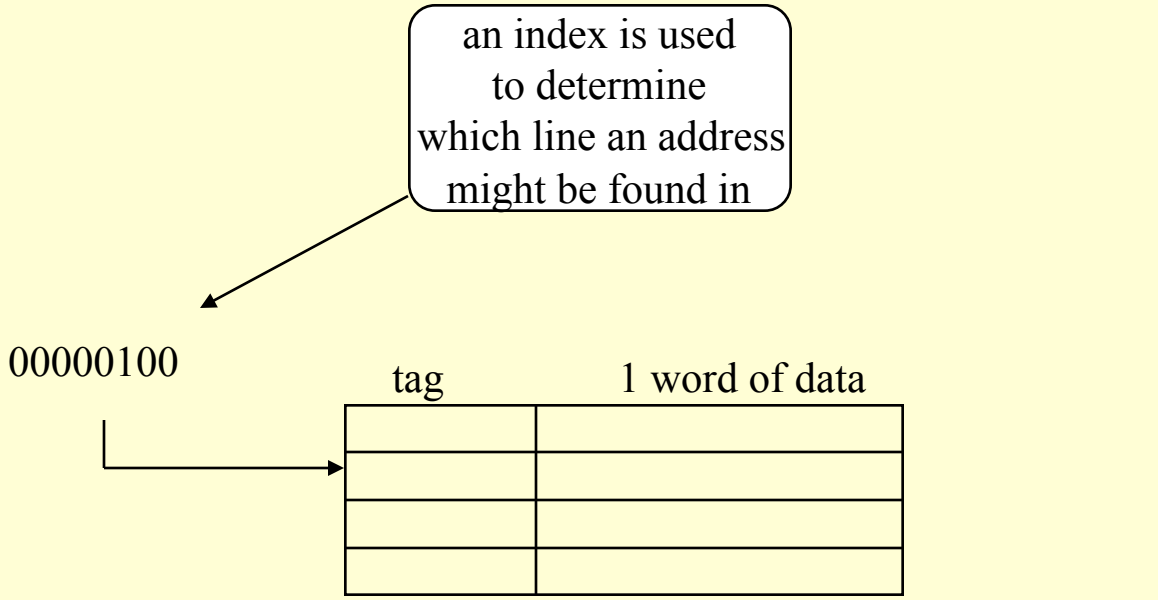
4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called
- The most popular replacement strategy is

A simpler cache

address trace:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



4 entries, each block holds one word, each word in memory maps to exactly one cache location.

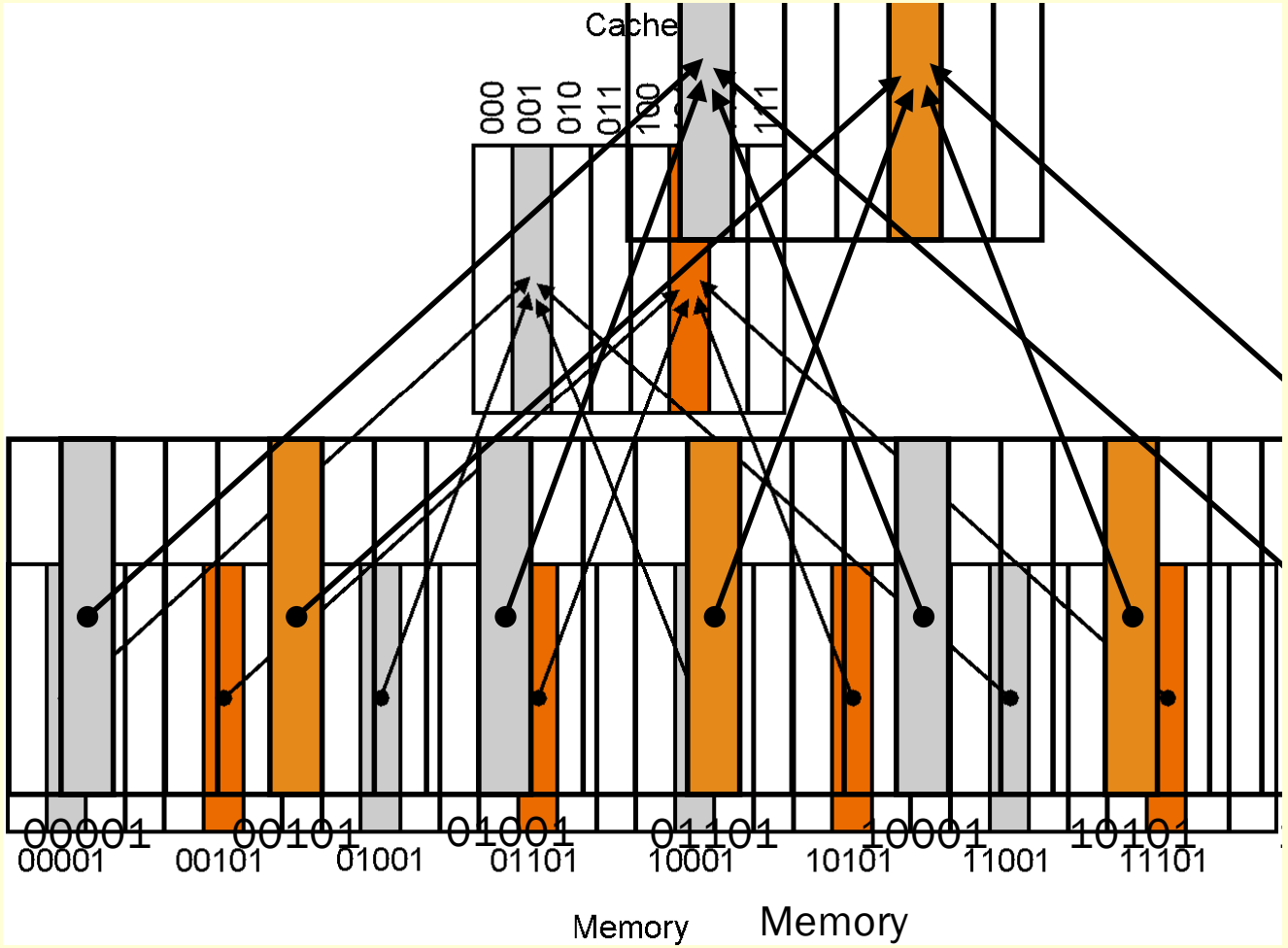
- A cache that can put a line of data in exactly one place is called
- What's the tag in this case?

Direct-mapped cache

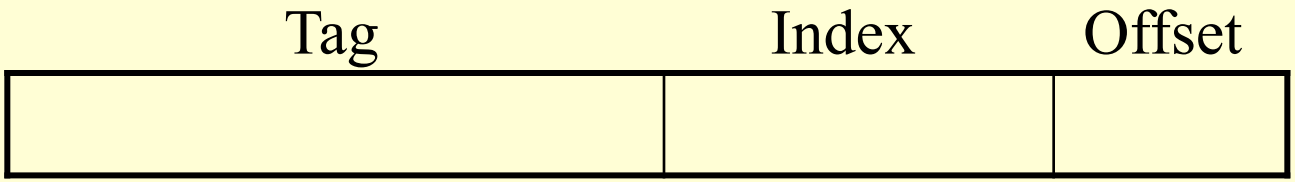
- Keeping track of when cache entries were last used (for LRU replacement) in big cache needs lots of hardware and can be slow.
- In a direct mapped cache, each memory location is assigned a single location in cache.
  - Usually* done by using a few bits of the address

** Some machines use a pseudo-random hash of the address*
But it is DETERMINISTIC!

Or another way to look at it: an 8 entry cache



Can you do?



Direct Mapped:
Cache Line Size = 16 bytes
8 lines in cache

What would the "size" of the cache be? (in bytes)