Cache Memory and Performance

Many of the following slides are taken with permission from

Complete Powerpoint Lecture Notes for
Computer Systems: A Programmer's Perspective (CS:APP)

Randal E. Bryant *and* *David R. O'Hallaron*

http://csapp.cs.cmu.edu/public/lectures.html

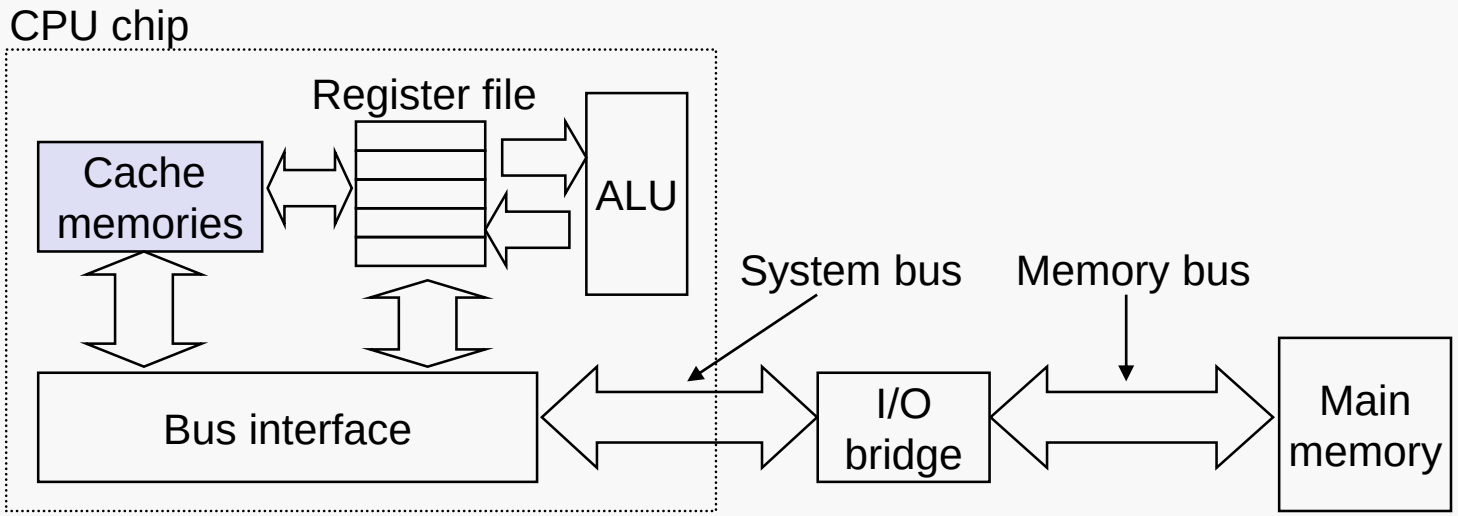
The book is used explicitly in CS 2505 and CS 3214 and as a reference in CS 2506.

Cache Memories

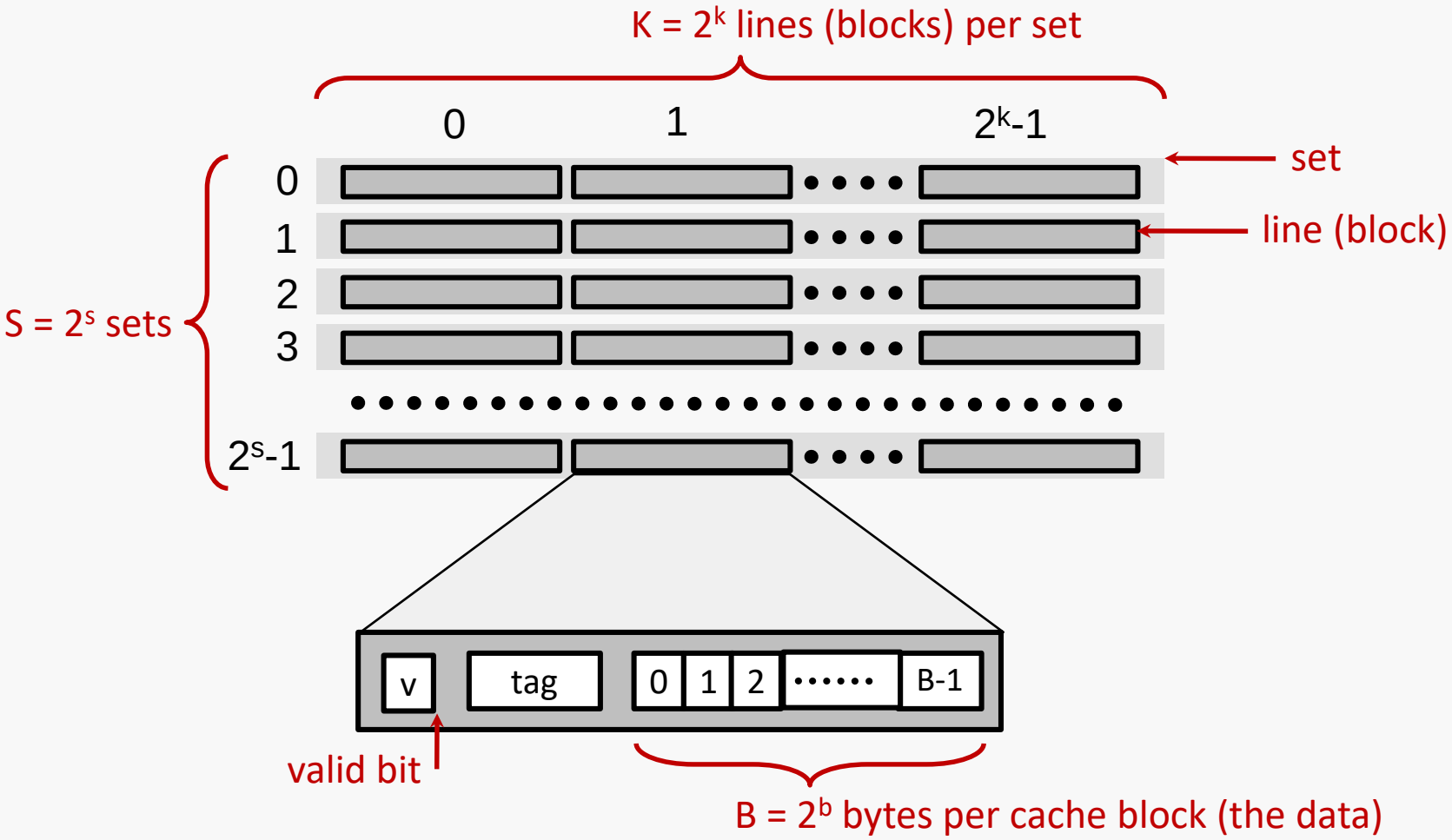
Cache memories are small, fast SRAM-based memories managed automatically in hardware.

CPU looks first for data in caches (e.g., L1, L2, and L3), then in main memory.

Typical system structure:



General Cache Organization (S, K, B)



Cache size:

C = S x K x B data bytes

Cache Defines View of DRAM

The "geometry" of the cache is defined by:

$S = 2^s$ the number of sets in the cache
$K = 2^k$ the number of lines (blocks) in a set
$B = 2^b$ the number of bytes in a line (block)

These values define a related way to think about the organization of DRAM:

DRAM consists of a sequence of blocks of B bytes.
The bytes in a block (line) can be indexed by using b bits.

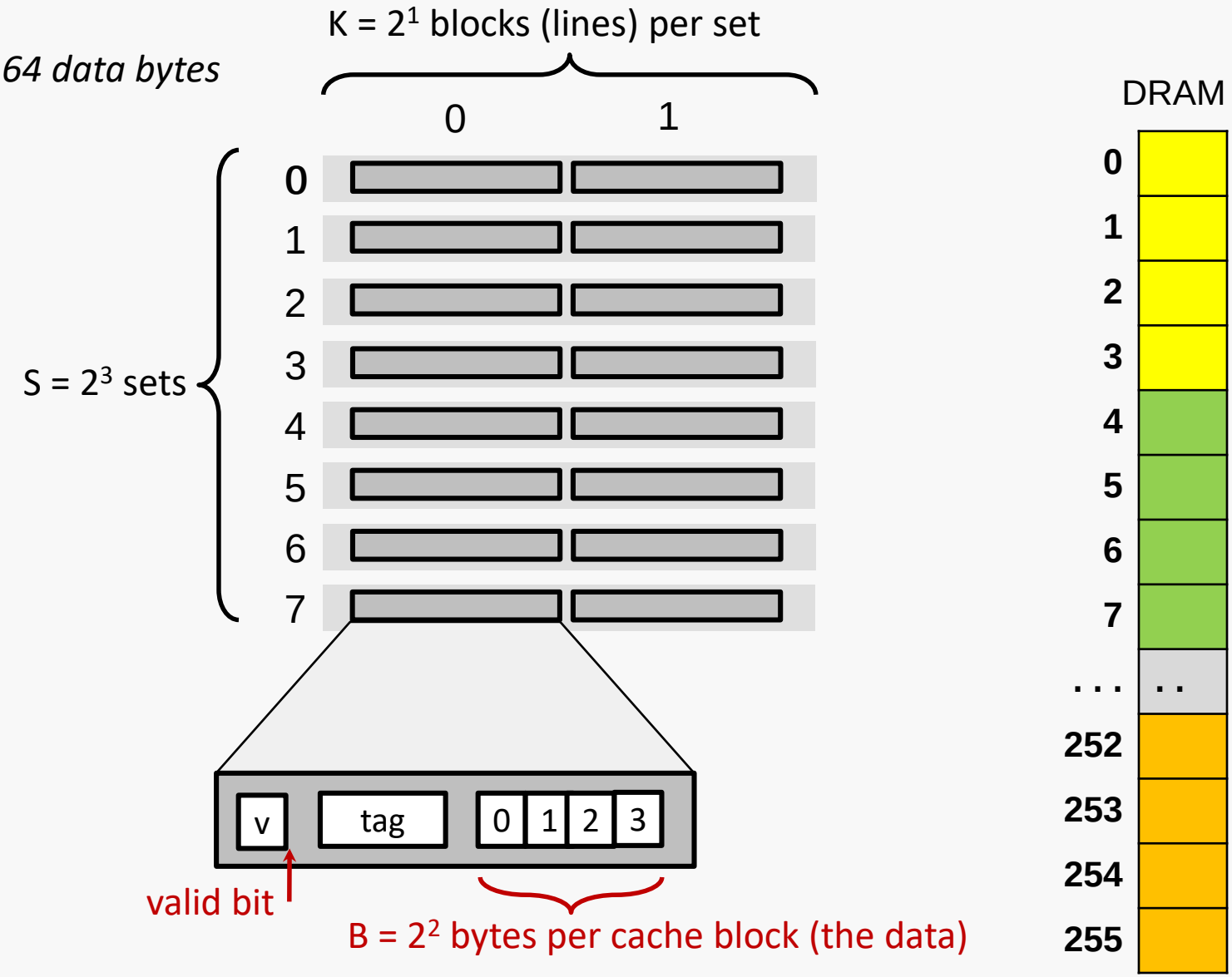
DRAM consists of a sequence of groups of S blocks (lines).
The blocks (lines) in a group can be indexed by using s bits.

Each group contains SxB bytes, which can be indexed by using s + b bits.

Cache (8, 2, 4) and 256-Byte DRAM

Cache size:

C = S x K x B = 64 data bytes

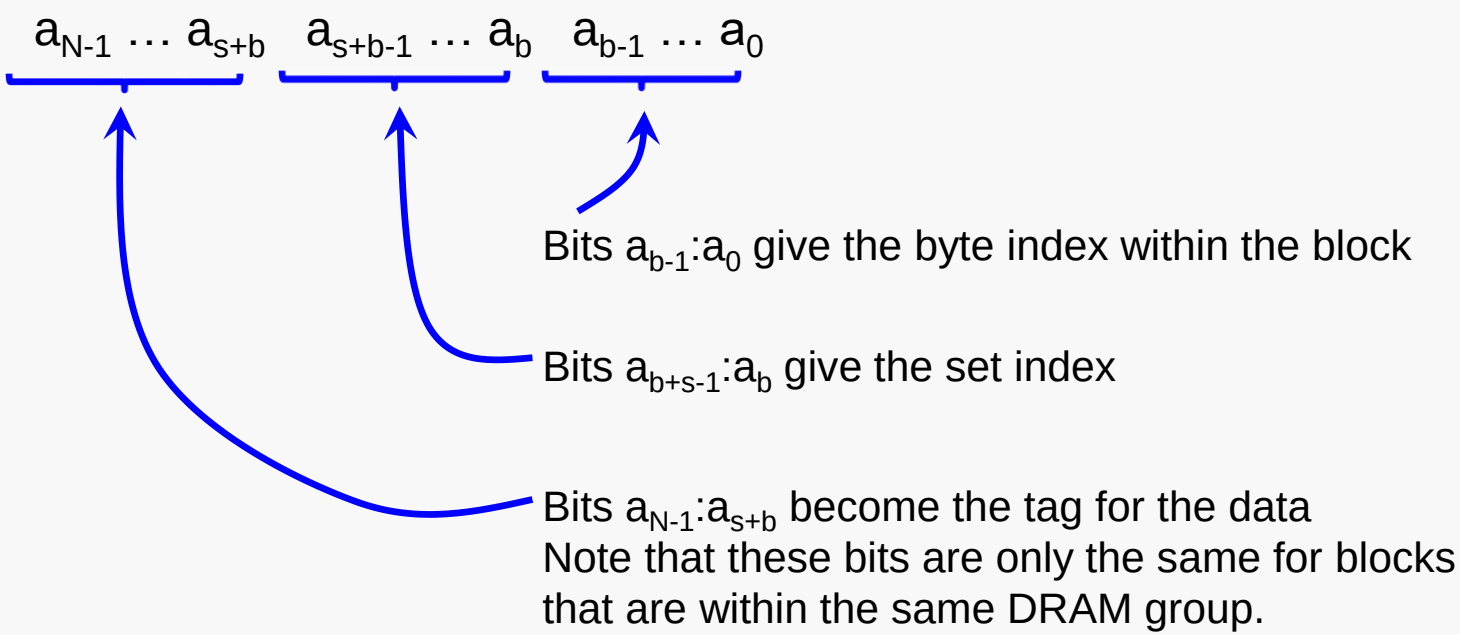


Cache View of DRAM

So, to generalize, suppose a cache has:

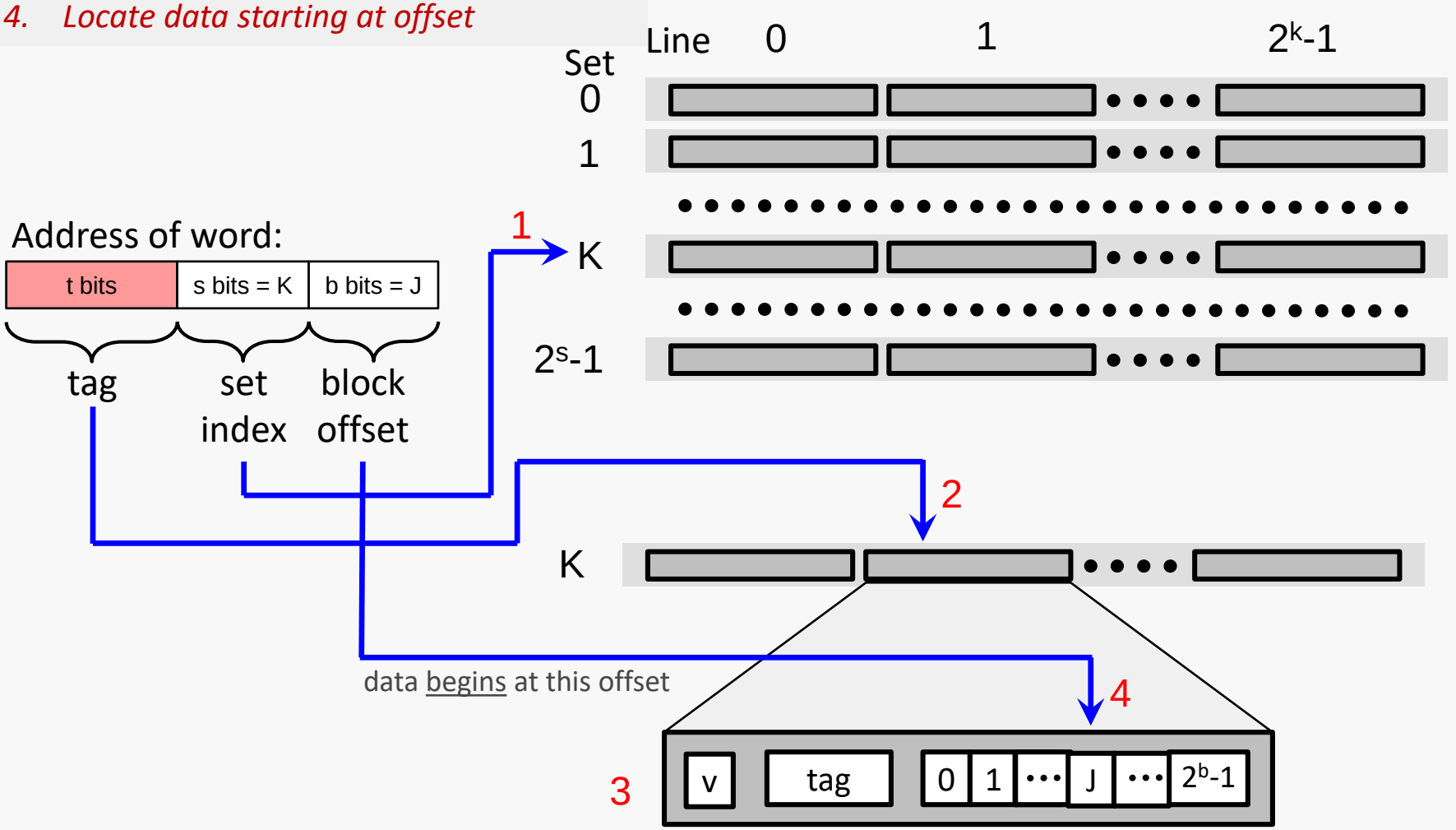
- $S = 2^s$ sets
- $K = 2^k$ blocks (lines) per set
- $B = 2^b$ bytes per block (line)

And, suppose that DRAM uses N-bit addresses. then for any address:



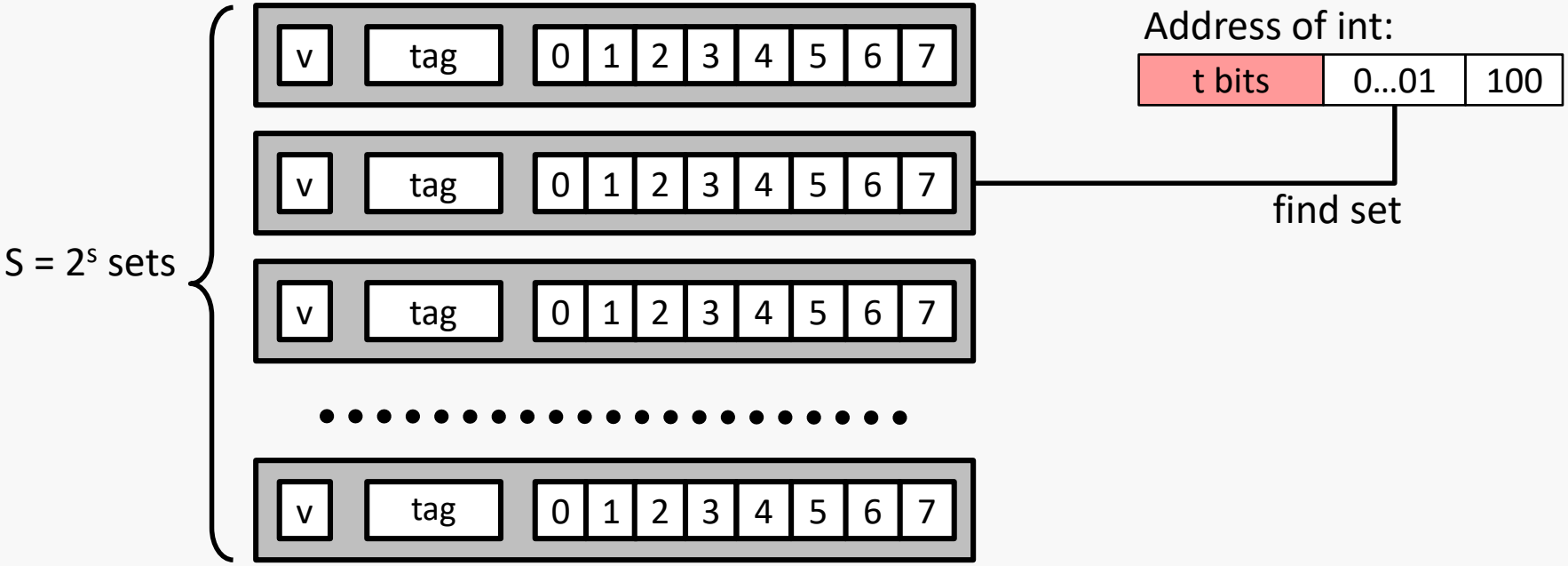
Cache Read

1. *Locate set*
2. *Check if any line in set has matching tag*
3. *Yes + line valid: hit*
4. *Locate data starting at offset*



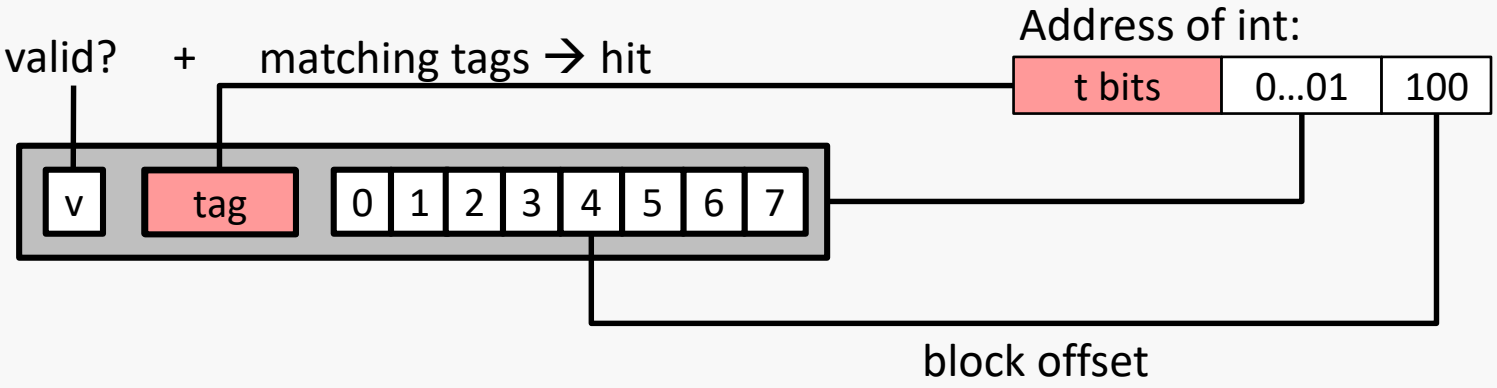
Example: Direct Mapped Cache (K = 1)

Direct mapped: One line per set
Assume: cache block size 8 bytes



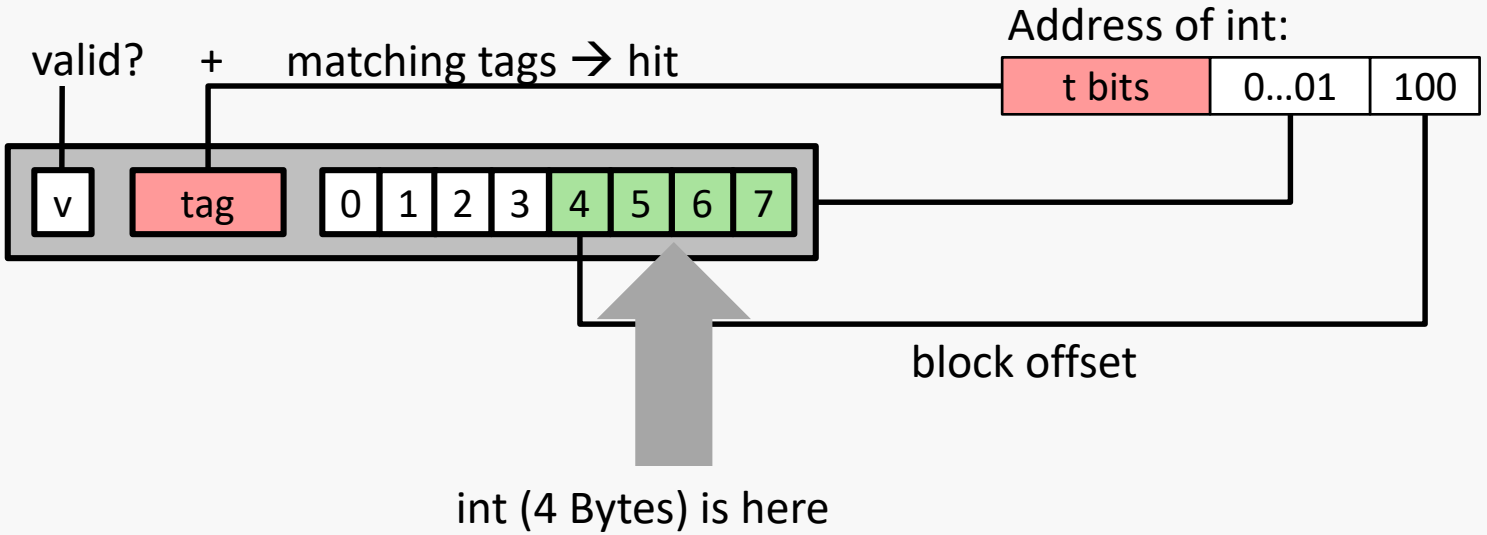
Example: Direct Mapped Cache (K = 1)

Direct mapped: One line per set
Assume: cache block size 8 bytes



Example: Direct Mapped Cache (K = 1)

Direct mapped: One line per set
Assume: cache block size 8 bytes

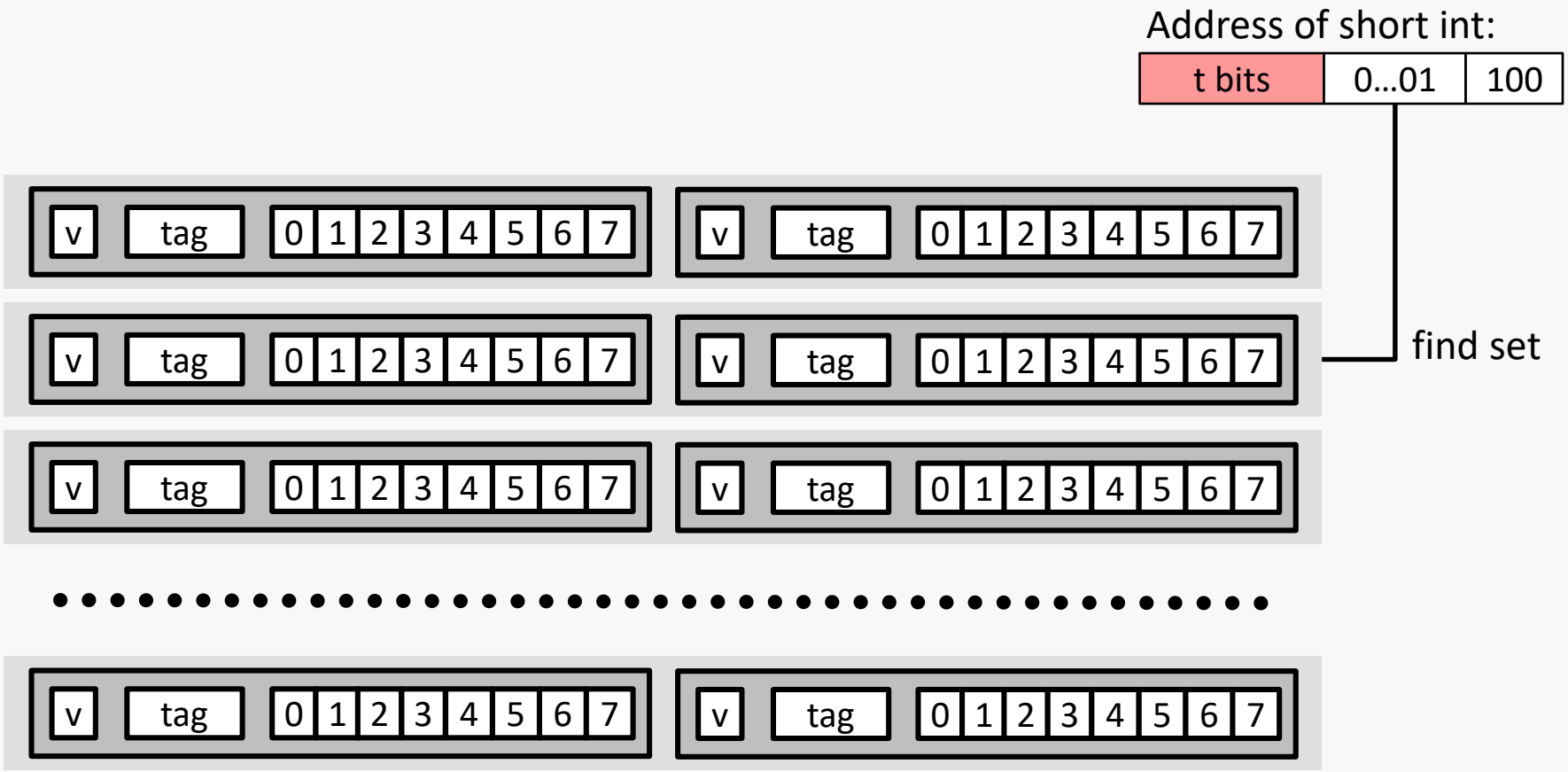


No match: old line (block) is evicted and replaced by requested block from DRAM

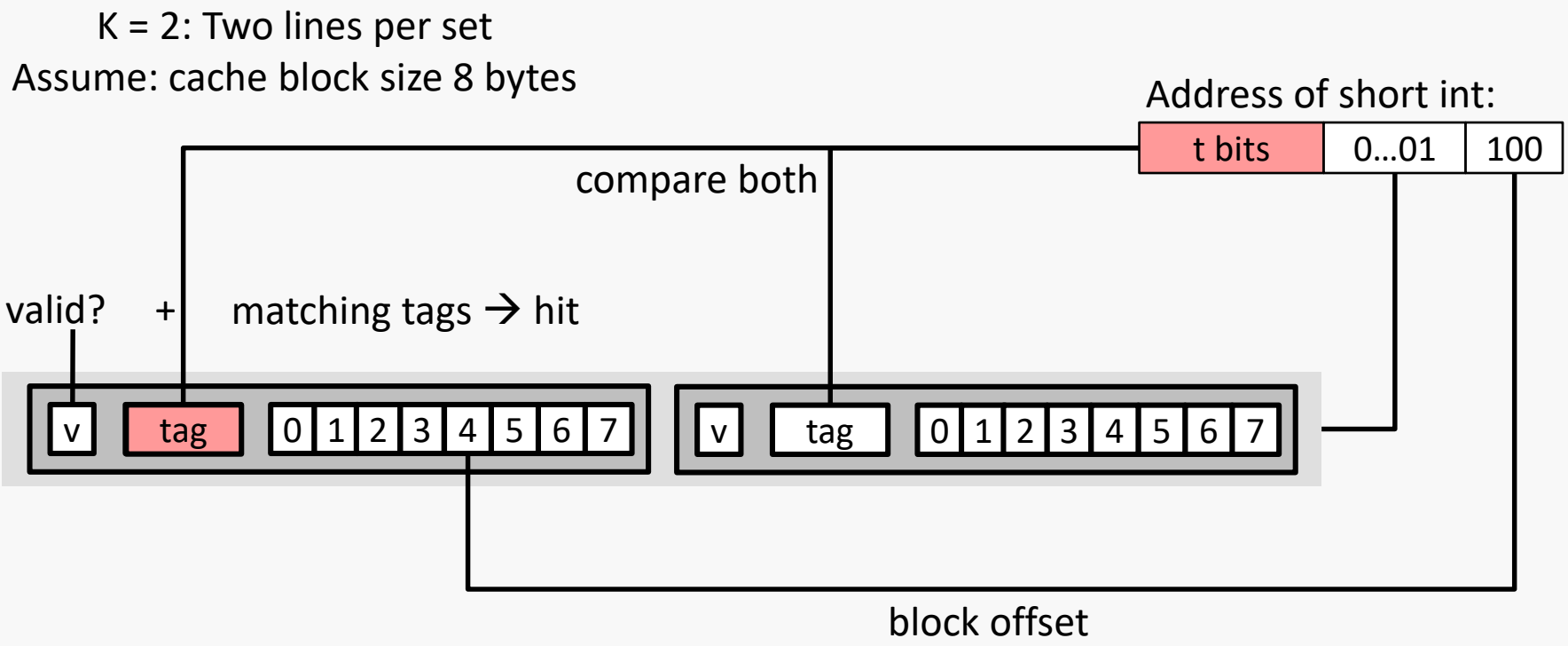
K-way Set Associative Cache (Here: K = 2)

K = 2: Two lines per set

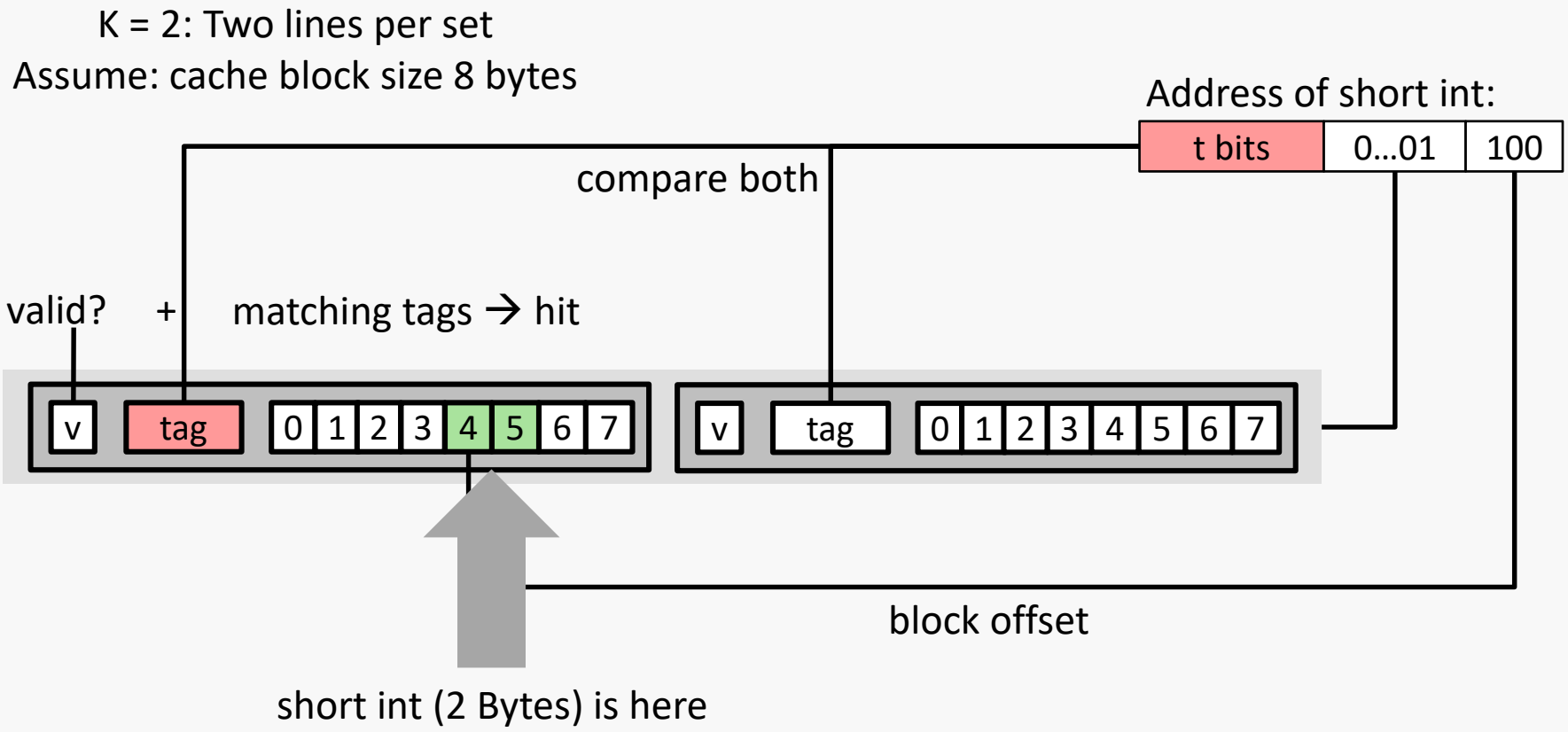
Assume: cache block size 8 bytes



K-way Set Associative Cache (Here: K = 2)



K-way Set Associative Cache (Here: K = 2)



No match:

- One line in set is selected for eviction and replacement
- Replacement policies: random, least recently used (LRU), …

Cache Organization Types

The "geometry" of the cache is defined by:

$S = 2^s$ the number of sets in the cache
$K = 2^k$ the number of lines (blocks) in a set
$B = 2^b$ the number of bytes in a line (block)

K = 1 (k = 0)	*direct-mapped cache* only one possible location in cache for each DRAM block
S > 1 K > 1	K-*way associative cache* K possible locations (in same cache set) for each DRAM block
S = 1 (only one set) K = # of cache blocks	*fully-associative cache* each DRAM block can be at any location in the cache

Cache Example

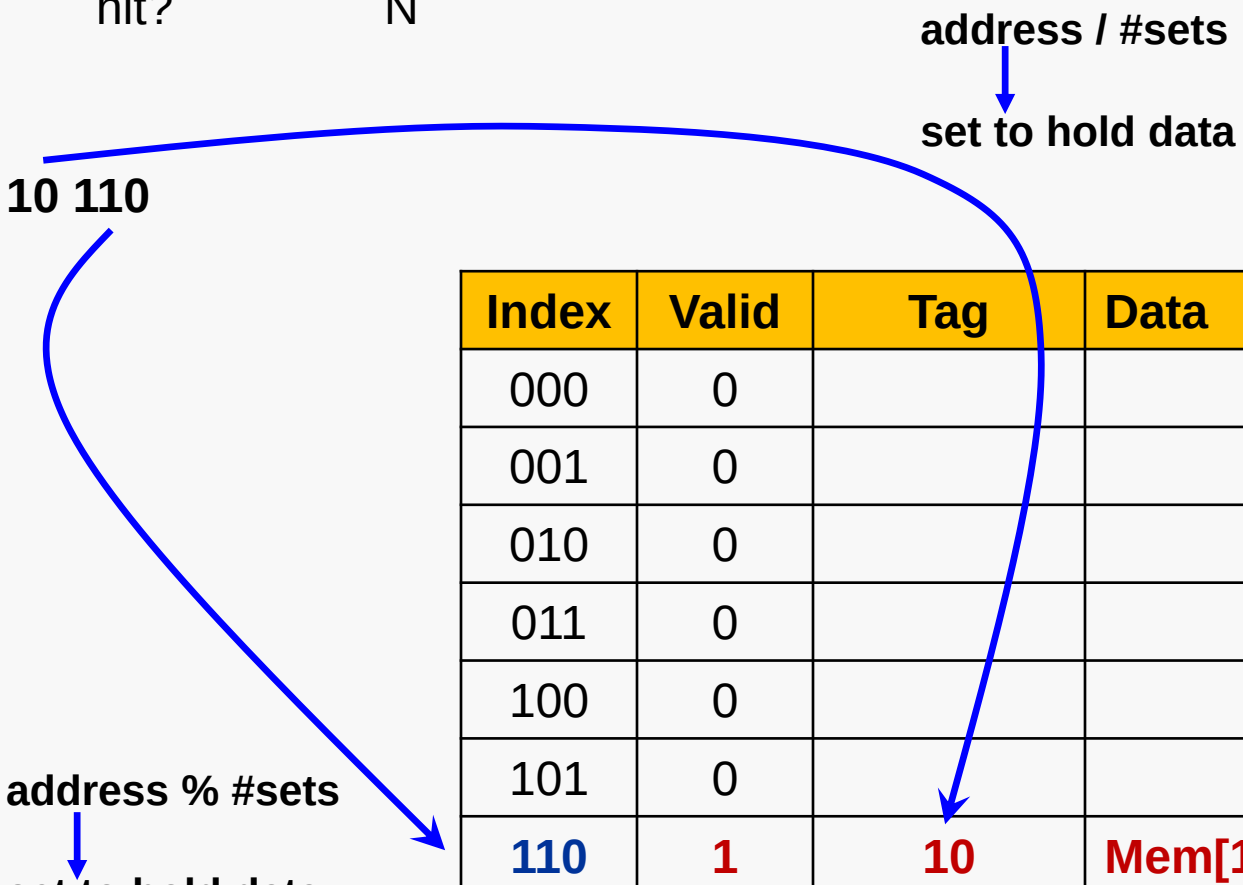
5-bit addresses (32-word DRAM)

Direct-mapped, 8-blocks, 1 word/block

Index	Valid	Tag	Data
000	0		
001	0		
010	0		
011	0		
100	0		
101	0		
110	0		
111	0		

Cache Example

Memory access
address 10 110
hit? N



Index	Valid	Tag	Data
000	0		
001	0		
010	0		
011	0		
100	0		
101	0		
110	1	10	Mem[10110]
111	0		

Cache Example

Memory access
address 11 010
hit? N

address / #sets

set to hold data

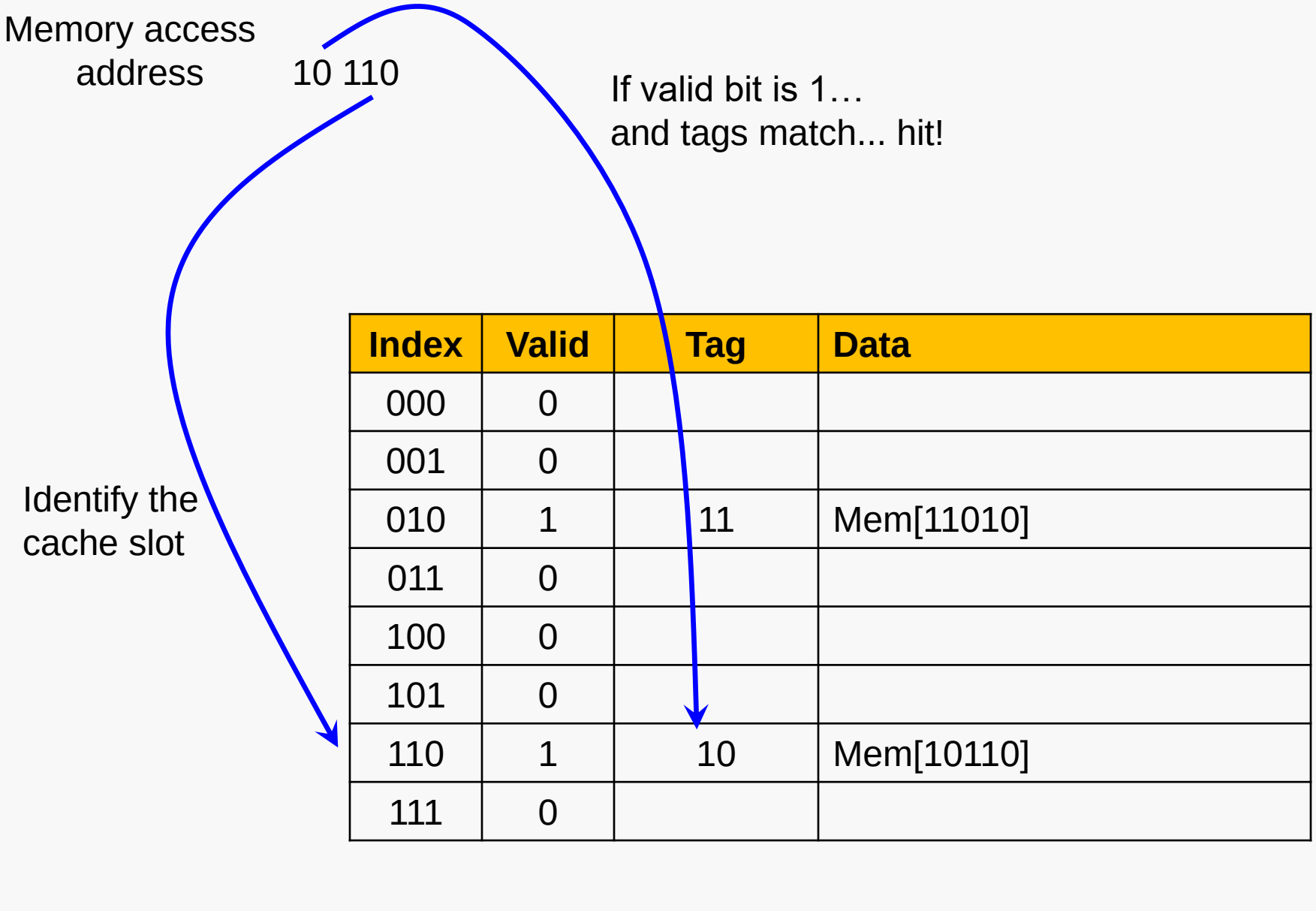
11 010

address % #sets

set to hold data

Index	Valid	Tag	Data
000	0		
001	0		
010	1	**11**	**Mem[11010]**
011	0		
100	0		
101	0		
110	1	10	Mem[10110]
111	0		

Cache Example



Index	Valid	Tag	Data
000	0		
001	0		
010	1	11	Mem[11010]
011	0		
100	0		
101	0		
110	1	10	Mem[10110]
111	0		

Cache Example

Memory accesses

address	10 000	miss
address	00 011	miss
address	10 000	hit

Index	Valid	Tag	Data
000	0	10	Mem[10000]
001	0		
010	1	11	Mem[11010]
011	0	00	Mem[00011]
100	0		
101	0		
110	1	10	Mem[10110]
111	0		

Cache Example

Memory access

address 10 010 miss (tags don't match)

Index	Valid	Tag	Data
. . .			
010	1	11	Mem[11010]
. . .			

replacement

Index	Valid	Tag	Data
. . .			
010	1	10	Mem[10010]
. . .			

Block Size Considerations

Larger blocks should reduce miss rate

- due to spatial locality (if present)

But in a fixed-sized cache

- larger blocks $\Rightarrow$ fewer of them
  more competition $\Rightarrow$ increased miss rate
- larger blocks $\Rightarrow$ pollution

Larger miss penalty

- can override benefit of reduced miss rate
- early restart and critical-word-first can help

Cache Misses

On cache hit, CPU proceeds normally

On cache miss:

- stall the CPU pipeline
- fetch block from next level of hierarchy
- instruction cache miss
  restart instruction fetch
- data cache miss
  complete data access

Write-Through

On data-write hit, could just update the block in cache

- but then cache and memory would be inconsistent

Write through: also update memory

But makes writes take longer

- e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
- effective CPI = 1 + 0.1×100 = 11

Solution: write buffer

- holds data waiting to be written to memory
  CPU continues immediately
- only stalls on write if write buffer is already full

Write-Back

Alternative: On data-write hit, just update the block in cache

- keep track of whether each block is dirty

When a dirty block is replaced

- write it back to memory
- can use a write buffer to allow replacing block to be read first

Write-Allocation

What should happen on a write miss?

Alternatives for write-through

- allocate on miss: fetch the block
- write around: don't fetch the block

  since programs often write a whole block before reading it (e.g., initialization)

For write-back

- usually fetch the block

DRAM Supporting Caches

Use DRAMs for main memory

- fixed width (e.g., 1 word)
- connected by fixed-width clocked bus
- bus clock is typically slower than CPU clock

Example cache block read

- 1 bus cycle for address transfer
- 15 bus cycles per DRAM access
- 1 bus cycle per data transfer

For 4-word block, 1-word-wide DRAM

- miss penalty = 1 + 4×15 + 4×1 = 65 bus cycles
- bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle

Measuring Cache Performance

Components of CPU time

- program execution cycles
- includes cache hit time
- memory stall cycles
- mainly from cache misses

With simplifying assumptions:

$$\text{Memory stall cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Cache Performance Operation

Given

- I-cache miss rate = 2%
- D-cache miss rate = 4%
- miss penalty = 100 cycles
- base CPI (ideal cache) = 2

Miss cycles per instruction

- I-cache: $0.02 \times 100 = 2$
- D-cache: $0.36 \times 0.04 \times 100 = 1.44$

Actual CPI = 2 + 2 + 1.44 = 5.44

- ideal CPU is 5.44/2 =2.72 times faster

Average Access Time

Hit time is also important for performance

Average memory access time (AMAT)

- AMAT = Hit time + Miss rate × Miss penalty

Example

- CPU with 1ns clock,
  hit time = 1 cycle,
  miss penalty = 20 cycles,
  I-cache miss rate = 5%

- AMAT = 1 + 0.05 × 20 = 2ns (2 cycles per instruction)

Performance Summary

When CPU performance increased

- Miss penalty becomes more significant

Decreasing base CPI

- Greater proportion of time spent on memory stalls

Increasing clock rate

- Memory stalls account for more CPU cycles

Can’t neglect cache behavior when evaluating system performance

Associative Caches

Fully associative

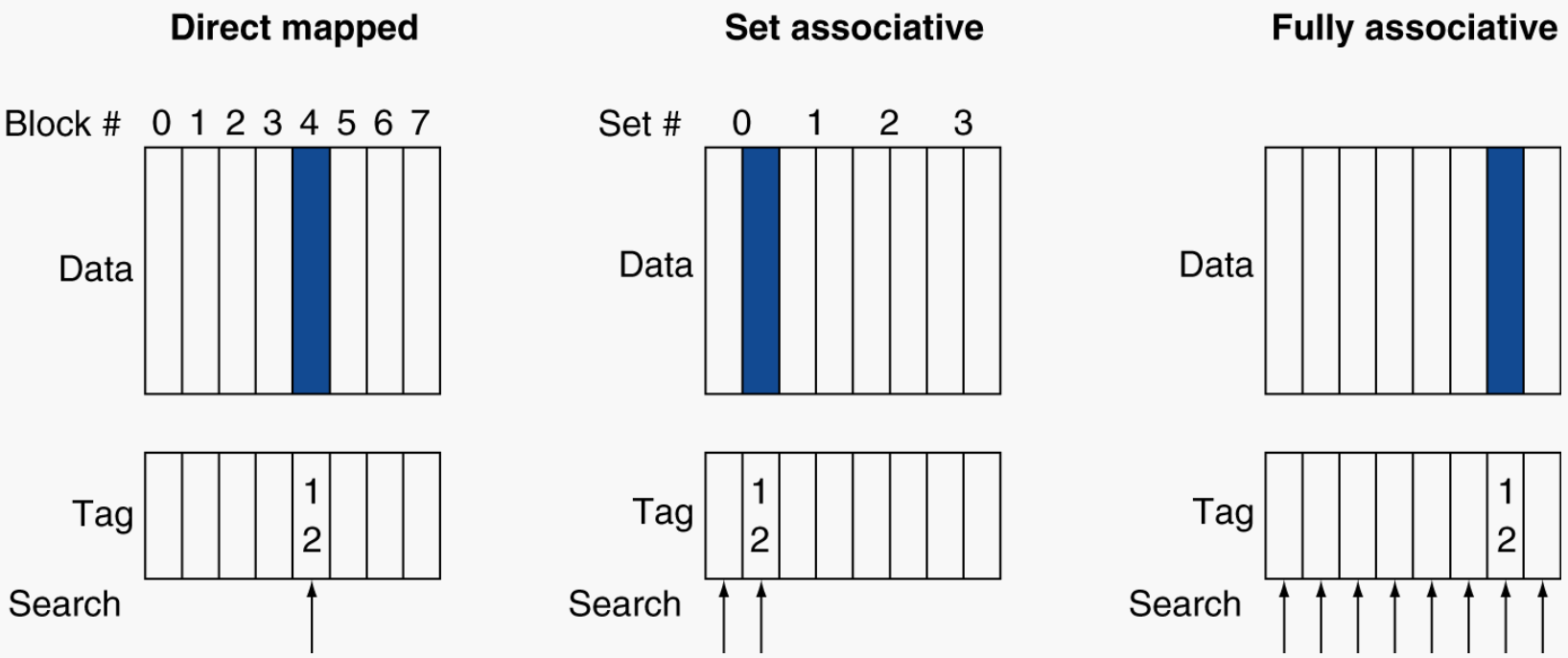
- allow a given block to go in any cache entry
- requires all entries to be searched at once
- comparator per entry (expensive)

K-way set associative

- cach set contains *K* entries
- block number determines which set

  (Block number) modulo (#Sets in cache)
- search all entries in a given set at once
- *K* comparators (less expensive)

Associative Cache Example



Spectrum of Associativity

Capacity: 8 blocks of user data

One-way set associative
(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Associativity Example

Compare 4-block caches

- direct mapped vs 2-way set associative vs fully associative
- block access sequence: 0, 8, 0, 6, 8

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	**Mem[0]**			
8	0	miss	**Mem[8]**			
0	0	miss	**Mem[0]**			
6	2	miss	Mem[0]		**Mem[6]**	
8	0	miss	**Mem[8]**		Mem[6]	

Direct mapped

Associativity Example

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	Mem[0]			
8	0	miss	Mem[0]	Mem[8]		
0	0	hit	**Mem[0]**	Mem[8]		
6	0	miss	Mem[0]	**Mem[6]**		
8	0	miss	**Mem[8]**	Mem[6]		

2-way set associative

Block address		Hit/miss	Cache content after access			
0		miss	Mem[0]			
8		miss	Mem[0]	Mem[8]		
0		hit	**Mem[0]**	Mem[8]		
6		miss	Mem[0]	Mem[8]	Mem[6]	
8		hit	Mem[0]	**Mem[8]**	Mem[6]	

Fully associative

How Much Associativity?

Increased associativity decreases miss rate

- but with diminishing returns

Simulation of a system with 64KB

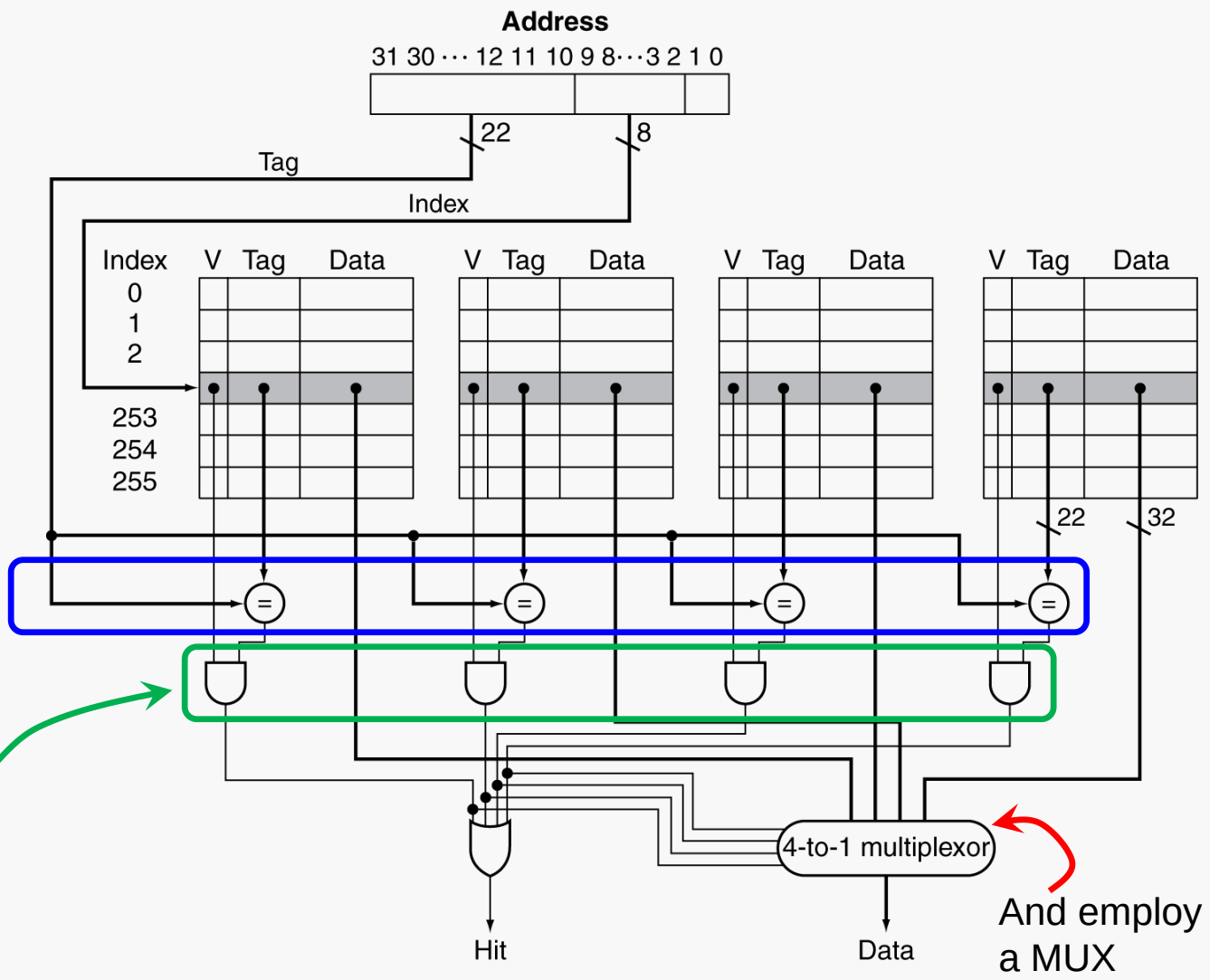
D-cache, 16-word blocks, SPEC2000

- 1-way: 10.3%
- 2-way: 8.6%
- 4-way: 8.3%
- 8-way: 8.1%

Set Associative Cache Organization

Compare the tag we're trying to match to all of the tags for blocks in the relevant set at the same time!

Then factor in the valid bits, also in parallel.



Replacement Policy

Direct mapped: only one choice

Set associative

- prefer non-valid entry, if there is one
- otherwise, choose among entries in the set

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

  Random
  - gives approximately the same performance as LRU for high associativity

Multilevel Caches

Primary (L-1) cache attached to CPU

- small, but fast

Level-2 cache services misses from primary cache

- larger, slower, but still faster than main memory

Main memory services L-2 cache misses

Some high-end systems include L-3 cache

Multilevel 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 + 0.02 × 400 = 9

Multilevel Cache Example

Now add L-2 cache

- access time = 5ns
- global miss rate to main memory = 0.5%

Primary miss with L-2 hit

- penalty = 5ns / 0.25ns = 20 cycles

Primary miss with L-2 miss

- extra penalty = 500 cycles

CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4

Performance ratio = 9/3.4 = 2.6

Multilevel Cache Considerations

Primary cache

- focus on minimal hit time

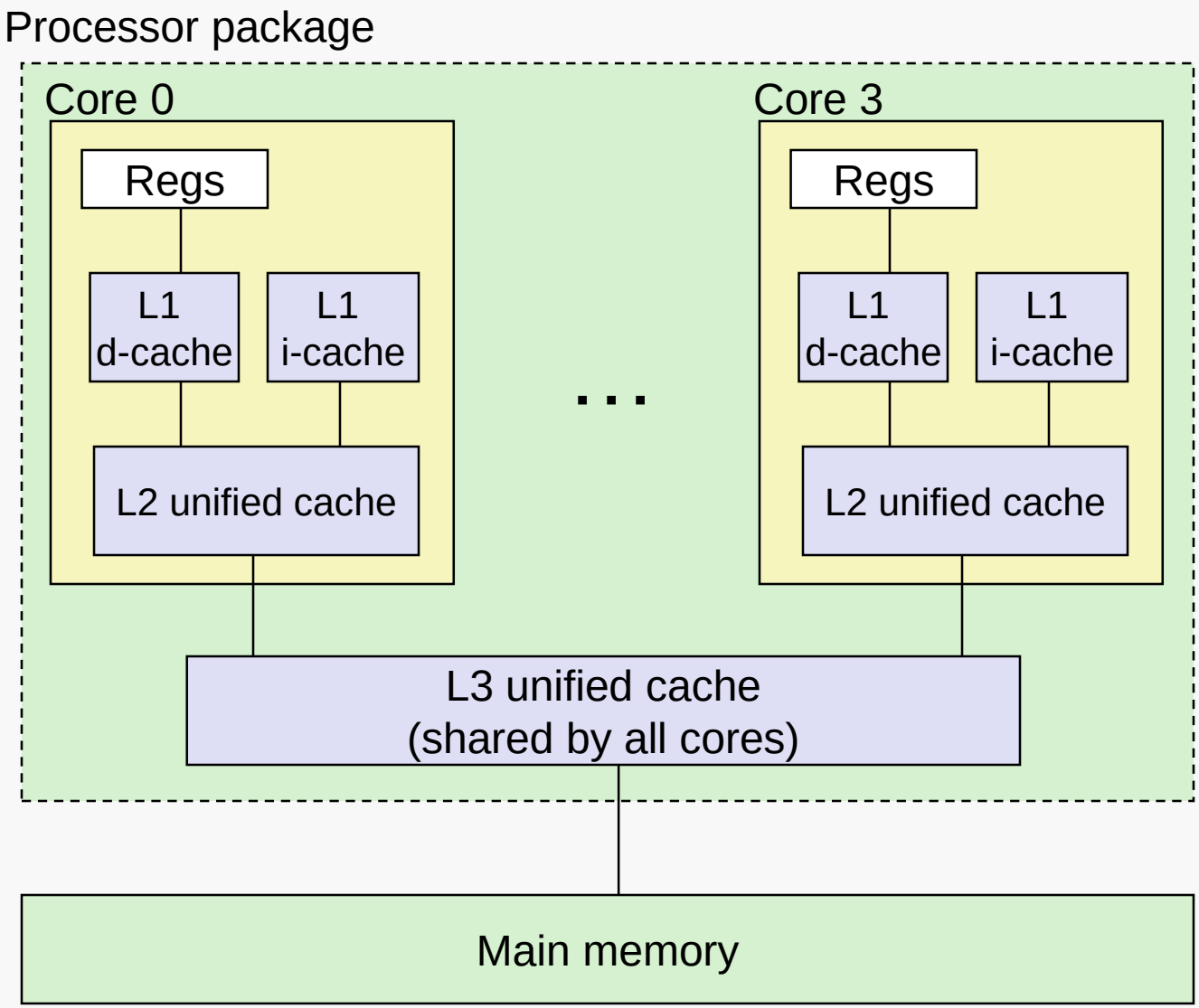
L-2 cache

- focus on low miss rate to avoid main memory access
- hit time has less overall impact

Results

- L-1 cache usually smaller than a single monolithic cache
- L-1 block size smaller than L-2 block size (then again…)

Example: Intel Core i7 Cache



2700 Series

L1 i-cache and d-cache:
32 KB, 8-way,
Access: 4 cycles

L2 unified cache:
256 KB, 8-way,
Access: 11 cycles

L3 unified cache:
8 MB, 16-way,
Access: 25-40 cycles

Block size: 64 bytes for all caches.