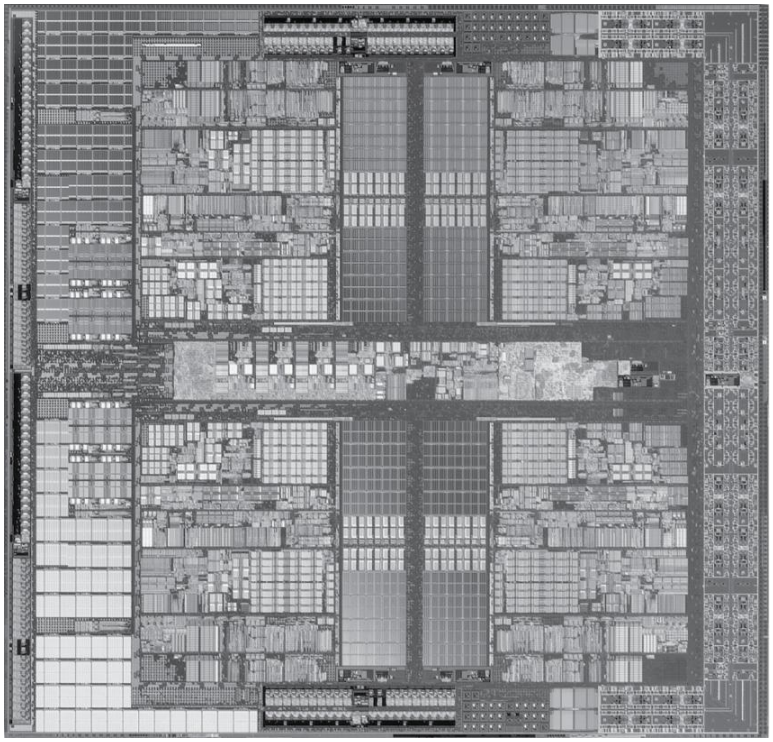
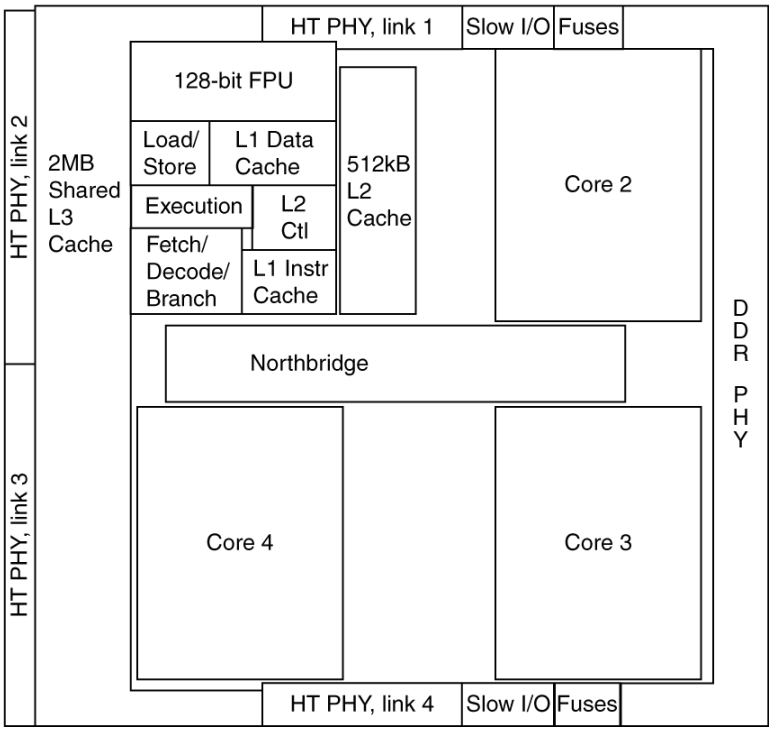
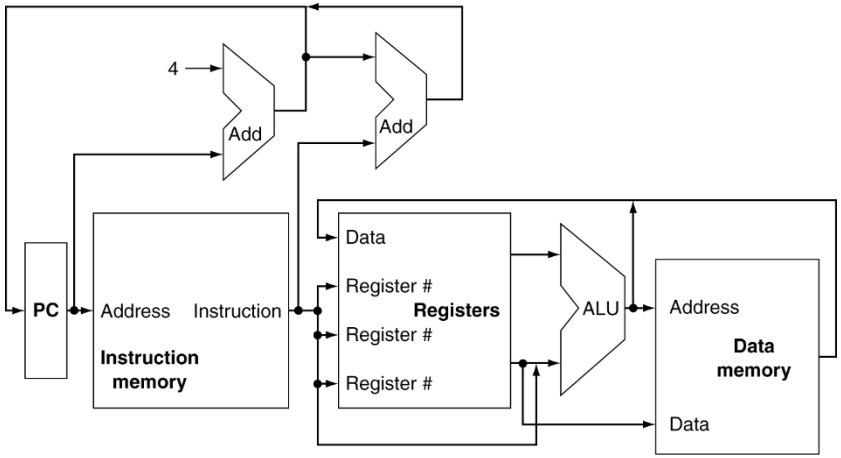
Large and Fast: Exploiting Memory Hierarchy

Jo, Heeseung

Inside the Processor (CPU)







Memory Technology

Static RAM (SRAM)

- 0.5ns – 2.5ns, $2000 – $5000 per GB

Dynamic RAM (DRAM)

- 50ns – 70ns, $20 – $75 per GB

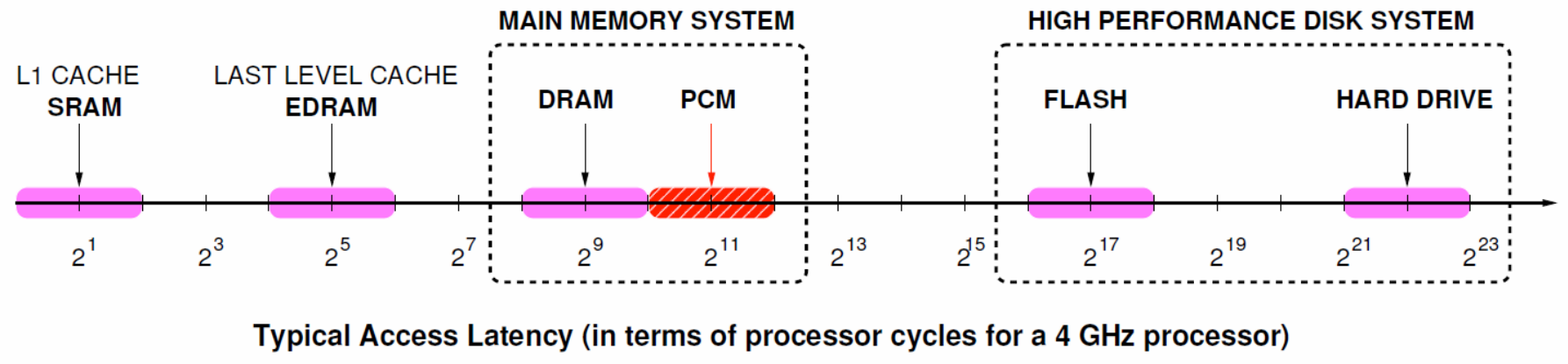
Magnetic disk

- 5ms – 20ms, $0.20 – $2 per GB

Ideal memory

- Access time of SRAM
- Capacity and cost/GB of disk

Registers vs. Memory



Typical Access Latency (in terms of processor cycles for a 4 GHz processor)

Qureshi (IBM Research) et al., Scalable High Performance Main Memory System Using Phase-Change Memory Technology, *ISCA* 2009.

Principle of Locality

Programs access a small proportion of their address space at any time

Temporal locality

- Items accessed recently are likely to be accessed again soon
- e.g., instructions in a loop, induction variables

Spatial locality

- Items near those accessed recently are likely to be accessed soon
- E.g., sequential instruction access, array data

Taking Advantage of Locality

Memory hierarchy

Store everything on disk

Copy recently accessed (and nearby) items from disk to smaller DRAM memory

- Main memory

Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory

- Cache memory attached to CPU

Memory Hierarchy Levels

Block (aka line): unit of copying

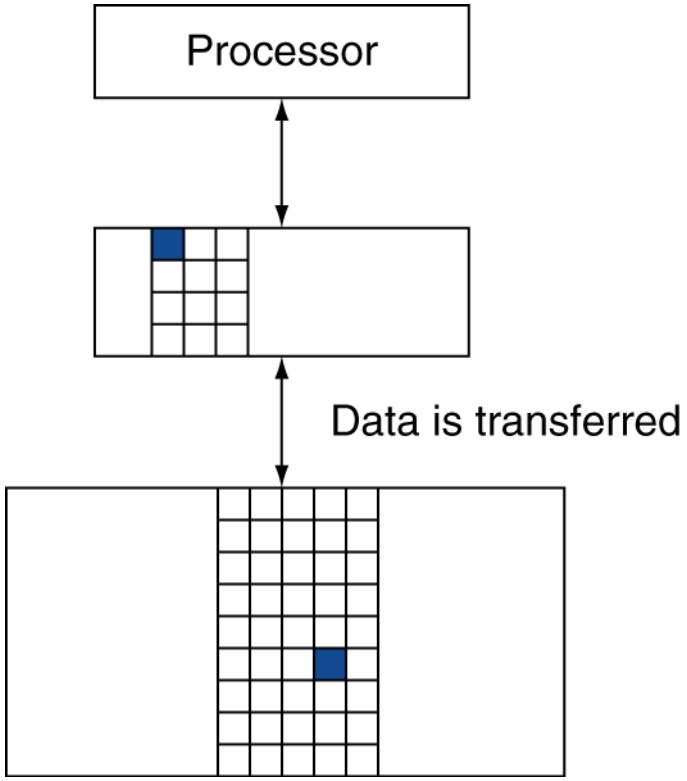
- May be multiple words

If accessed data is present in upper level

- Hit: access satisfied by upper level
  - Hit ratio: hits/accesses

If accessed data is absent

- Miss: block copied from lower level
  - Time taken: miss penalty
  - Miss ratio: misses/accesses
    = 1 – hit ratio
- Then accessed data supplied from upper level

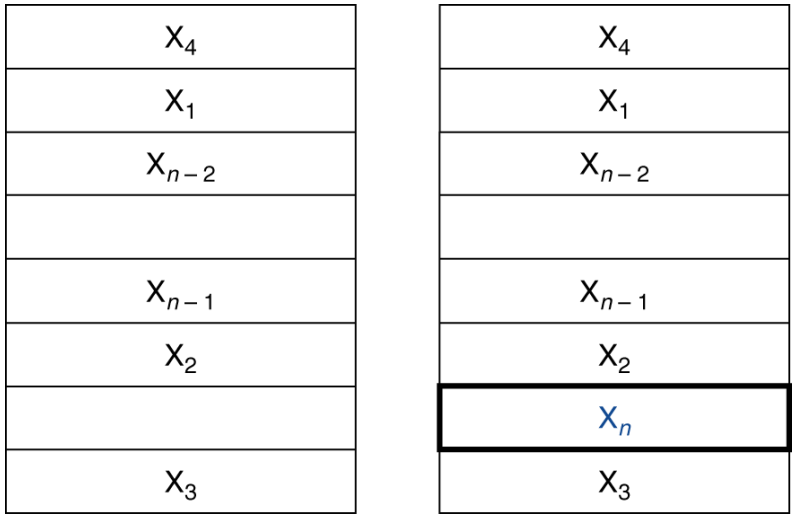


Cache Memory

Cache memory

- The level of the memory hierarchy closest to the CPU

Given accesses X_1, …, X_{n-1}, X_n



a. Before the reference to X_n b. After the reference to X_n

How do we know if the data is present?

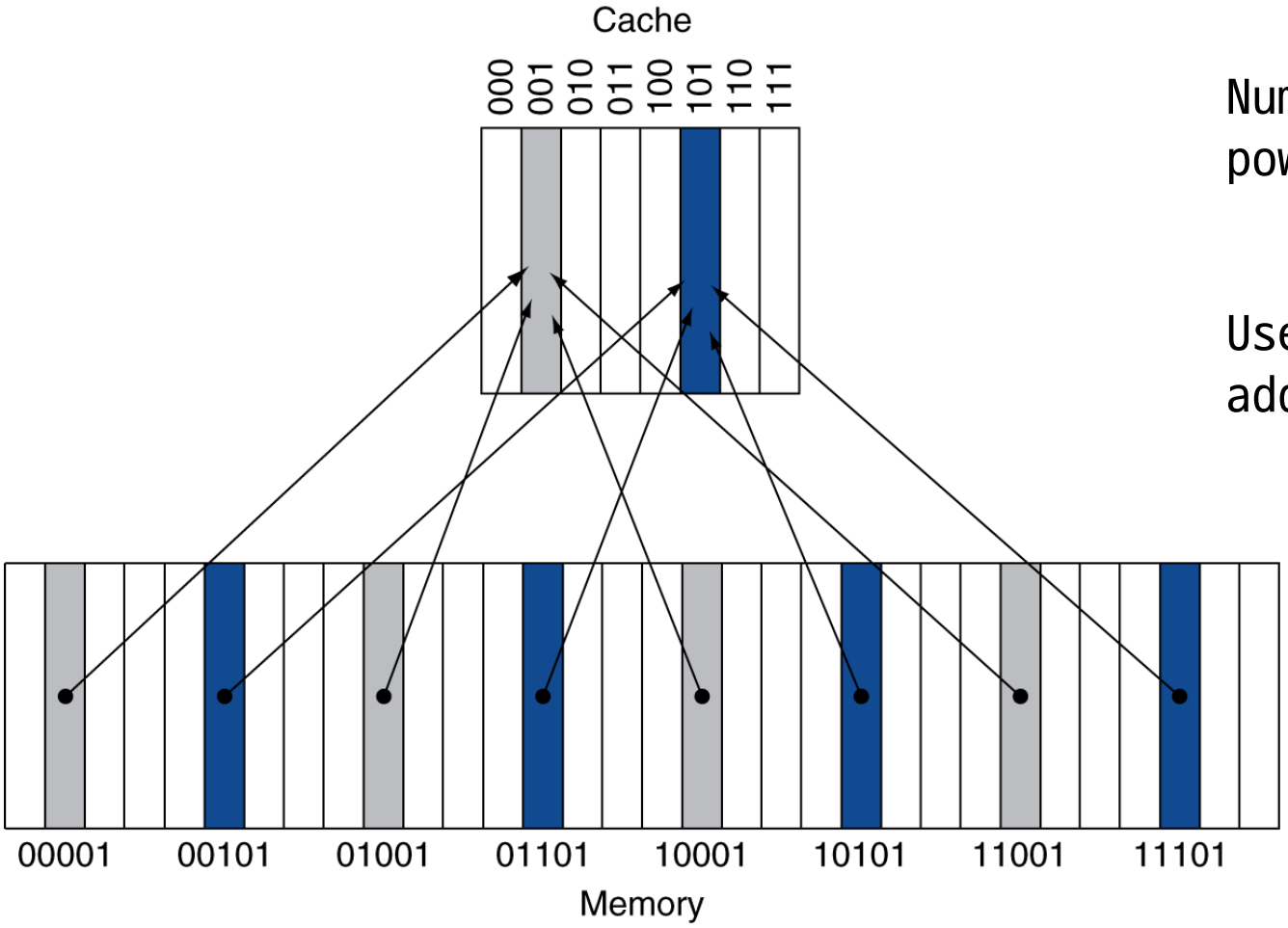
Where do we look?

Direct Mapped Cache

Location determined by address

Direct mapped: only one choice

- (Block address) modulo (#Blocks in cache)



Num. of Blocks is a power of 2

Use low-order address bits

Tags and Valid Bits

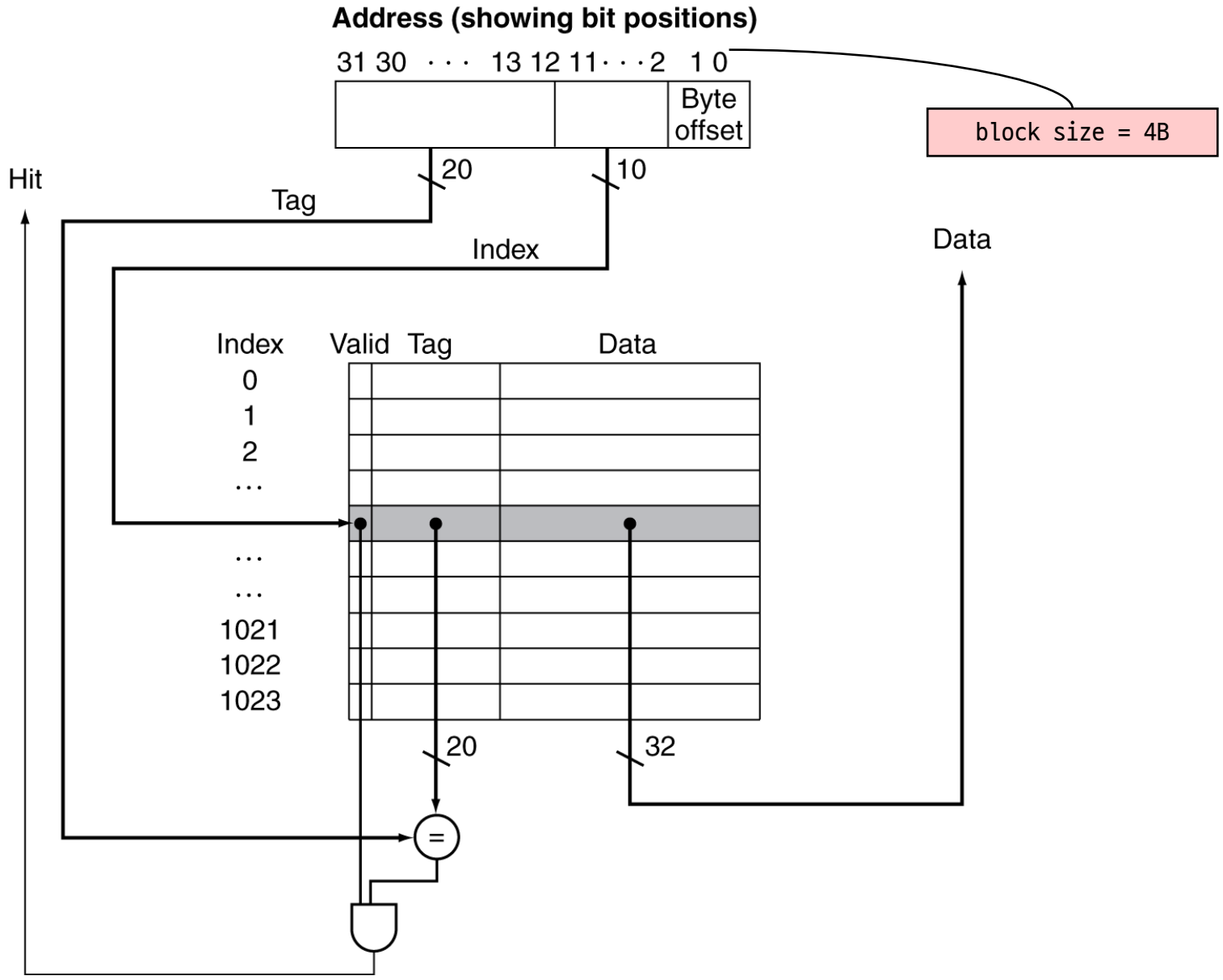
How do we know which particular block is stored in a cache location?

- Store block address as well as the data
- Actually, only need the high-order bits
- Called the tag

What if there is no data in a location?

- Valid bit: 1 = present, 0 = not present
- Initially 0

Address Subdivision



Cache Example

8-blocks, 1 word/block, direct mapped

Initial state

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	10	Mem[10010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

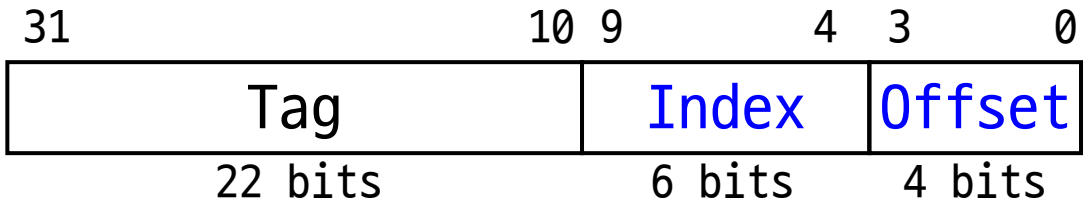
Example: Larger Block Size

32 bit, 64 blocks, 16 bytes/block

- To what block number does address 1200 map?

Block address = $\lfloor 1200/16 \rfloor = 75$

Block number = 75 modulo 64 = 11



Block Size Considerations

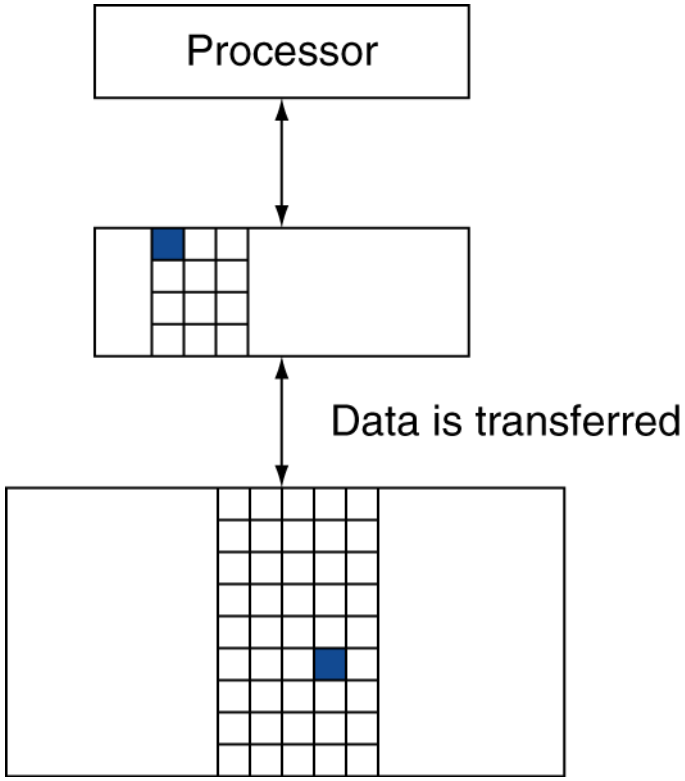
Larger blocks should reduce miss rate

- Due to spatial locality

But in a fixed-sized cache

- Larger blocks ⇒ fewer of them
  - More competition ⇒ increased miss rate
- Larger blocks ⇒ pollution

4 bytes/block vs. 16 bytes/block

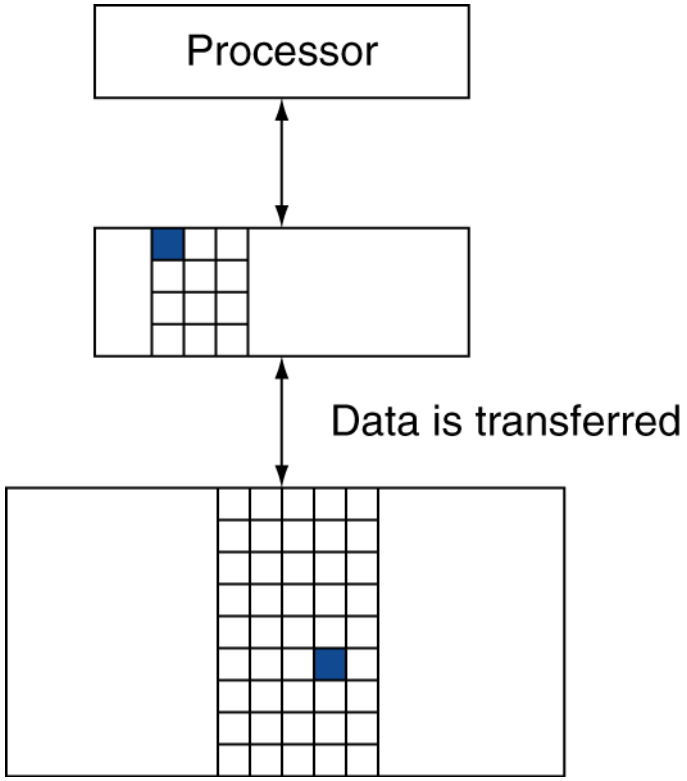


Cache Misses

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



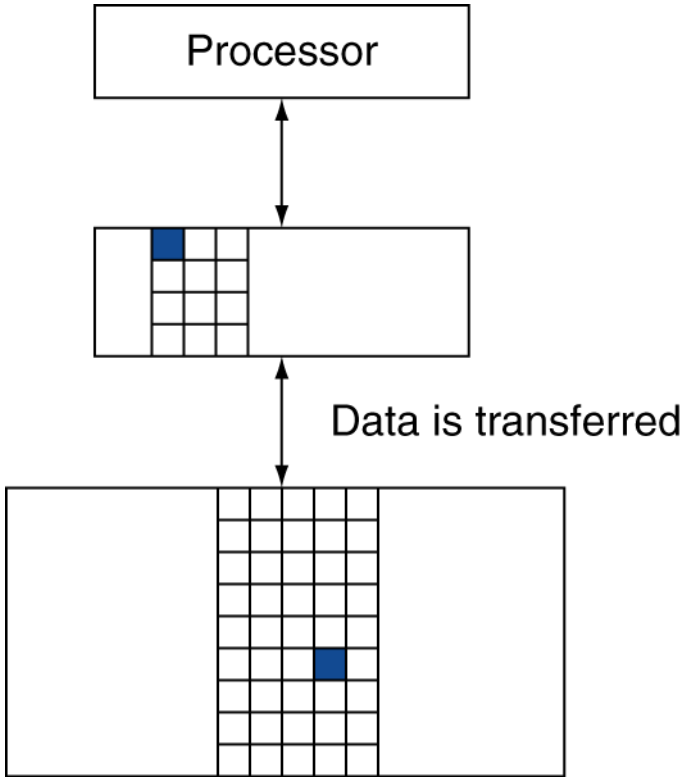
Write-Through

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

- But then cache and memory would be inconsistent

Write through: also update lower memory

- But makes writes take longer
- e.g., if base CPI = 1, 10% of instructions are stored, 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

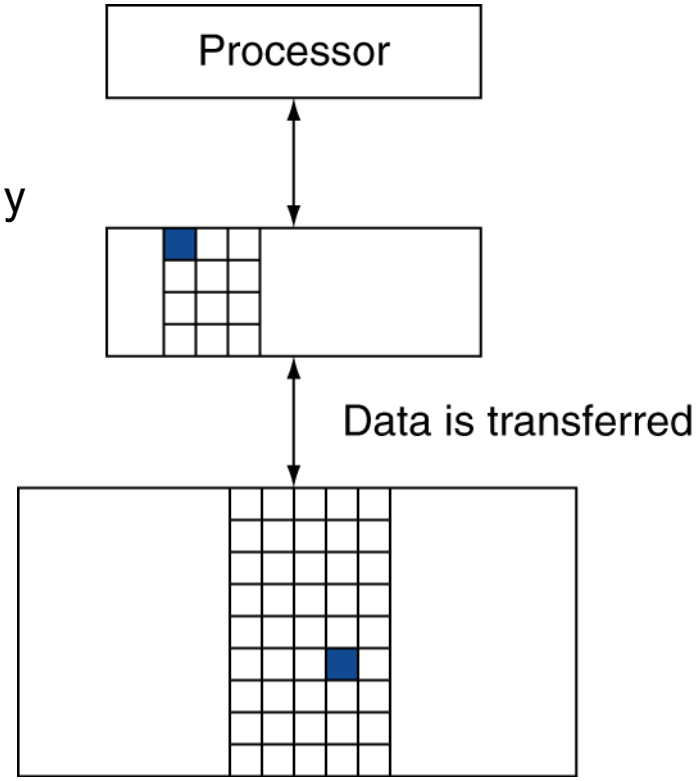
Write-Back

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 lower memory
- Can use a write buffer to allow replacing block to be read first



Write Allocation

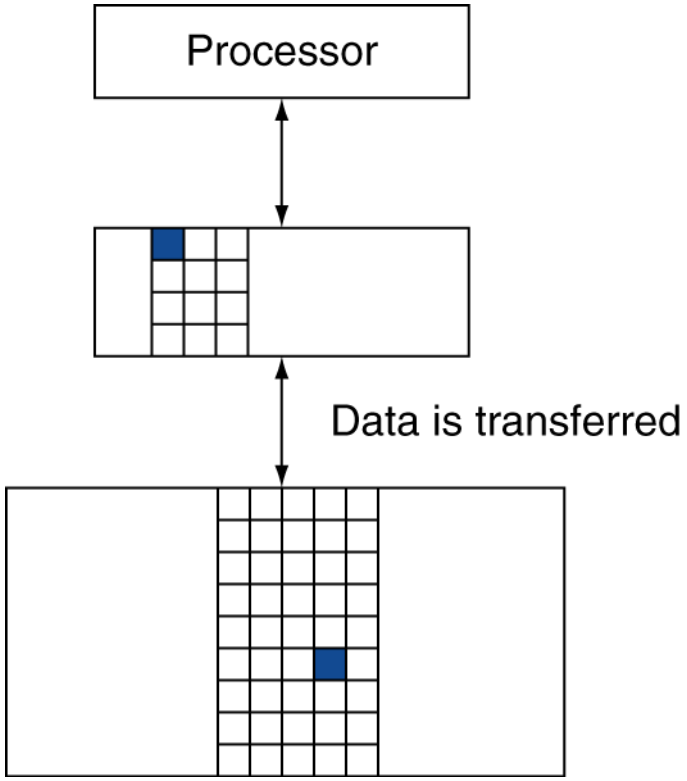
What should happen on a write miss?

Alternatives for write-through

- Allocate on miss: fetch the block (write allocation)
- Write around: don't fetch the block (no write allocation)
  - Pass cache

For write-back

- Usually fetch the block



Example: Intrinsity FastMATH

Embedded MIPS processor

- 12-stage pipeline
- Instruction and data access on each cycle

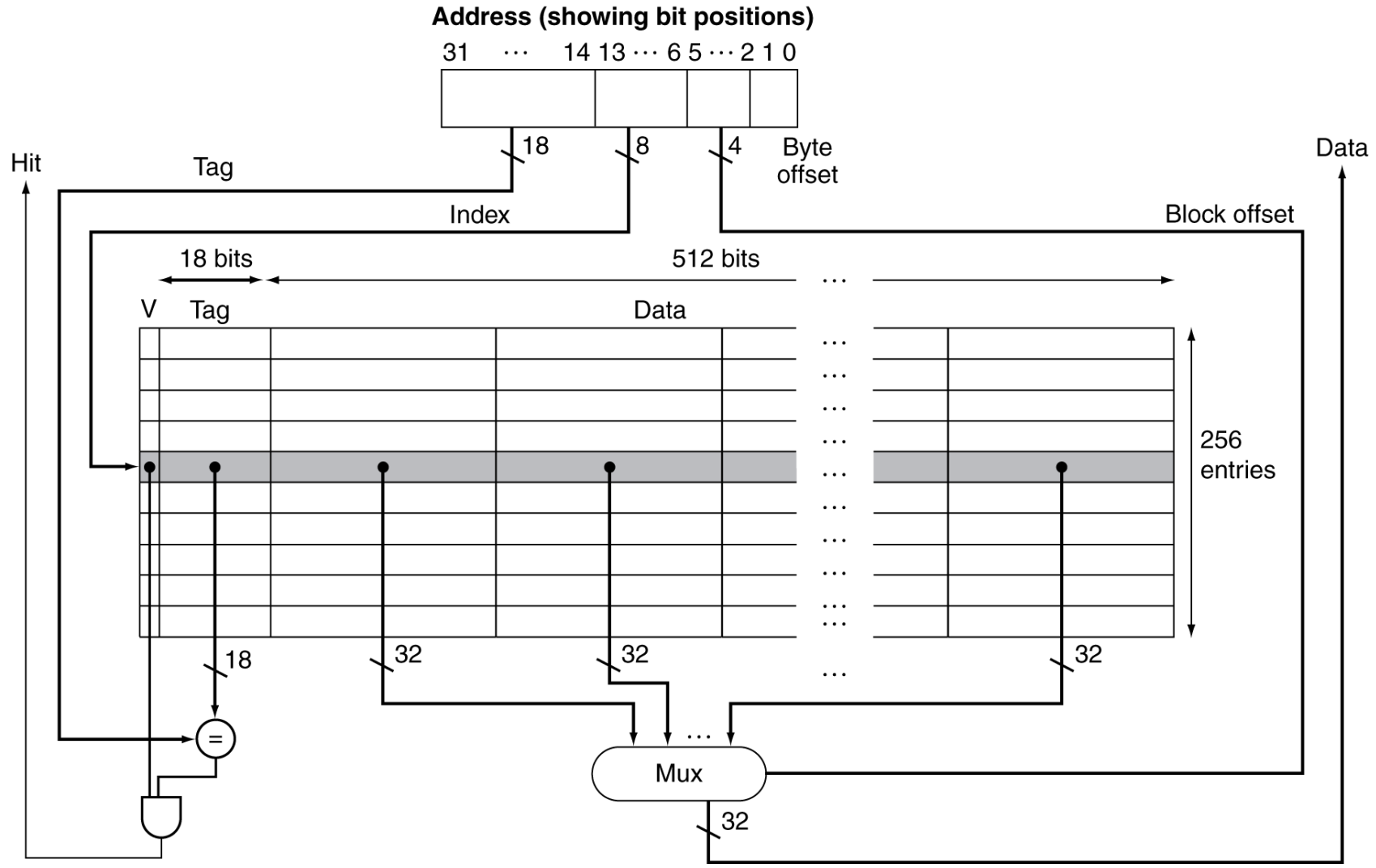
Split cache: separate I-cache and D-cache

- Each 16KB: 256 blocks × 16 words(64B) / block
- I-cache: read and no write
- D-cache: write-through or write-back

SPEC2000 miss rates

- I-cache: 0.4%
- D-cache: 11.4%

Example: Intrinsity FastMATH



Measuring Cache Performance

Components of CPU time

- Program execution cycles
  - Includes cache hit time
- Memory stall cycles
  - Mainly from cache misses

With simplifying assumptions:

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

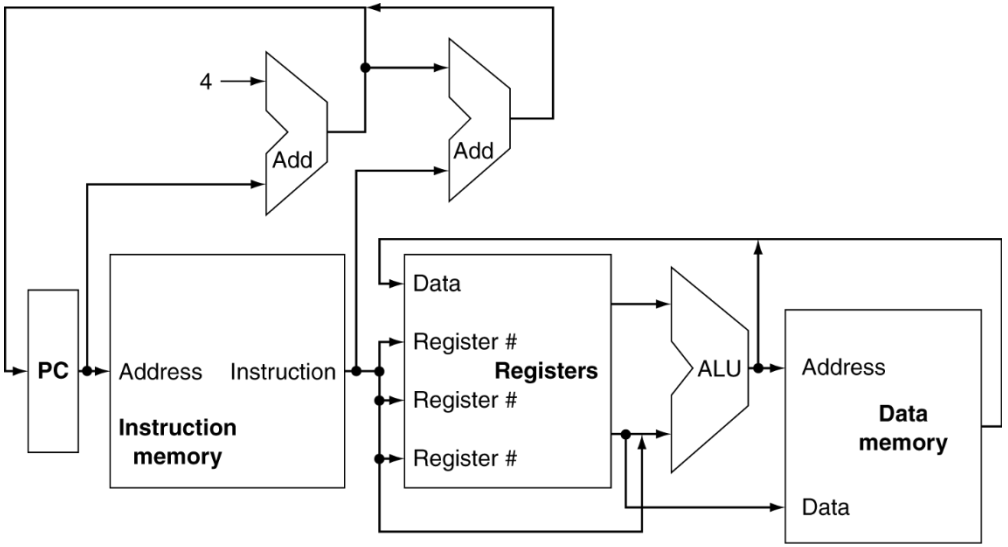
Cache Performance Example

Given

- I-cache miss rate = 2%
- D-cache miss rate = 4%
- Miss penalty = 100 cycles
- Load & stores are 36% of instructions

Miss cycles per instruction

- I-cache: 1.0 × 0.02 × 100 = 2
- D-cache: 0.36 × 0.04 × 100 = 1.44



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

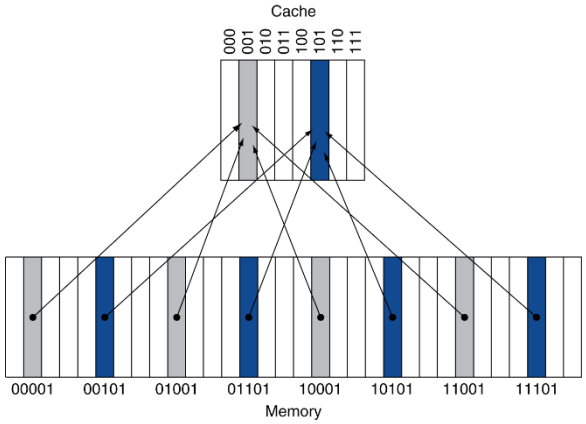
Direct mapped

Fully associative

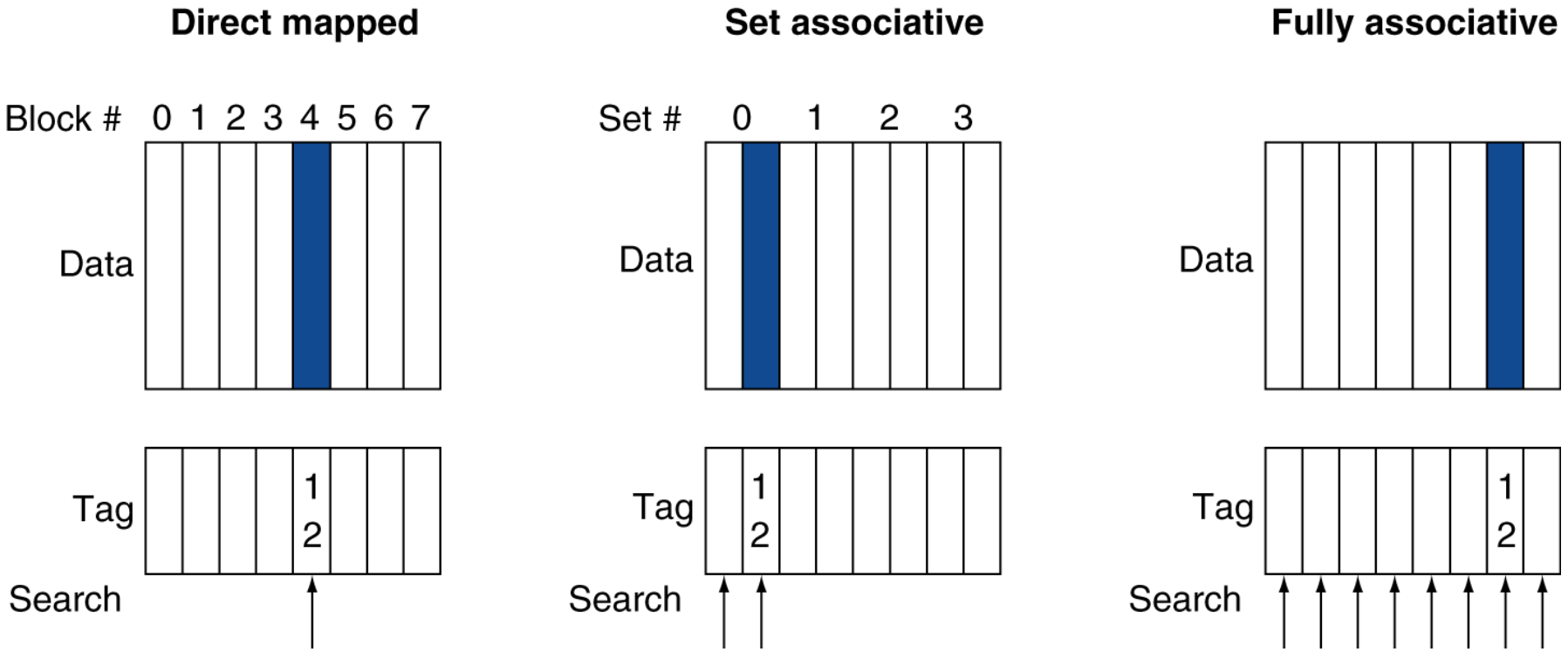
- Allow a given block to go in any cache entry
- Requires all entries to be searched at once
- Comparator per entry (expensive)

n-way set associative

- Each set contains *n* entries
- Block number determines which set
  - (Block number) modulo (#Sets in cache)
- Search all entries in a given set at once
- *n* comparators (less expensive)



Associative Cache Example



Spectrum of Associativity

For a cache with 8 entries

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

Direct mapped

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]	

Associativity Example

2-way set associative

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]		

Fully 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]	

How Much Associativity

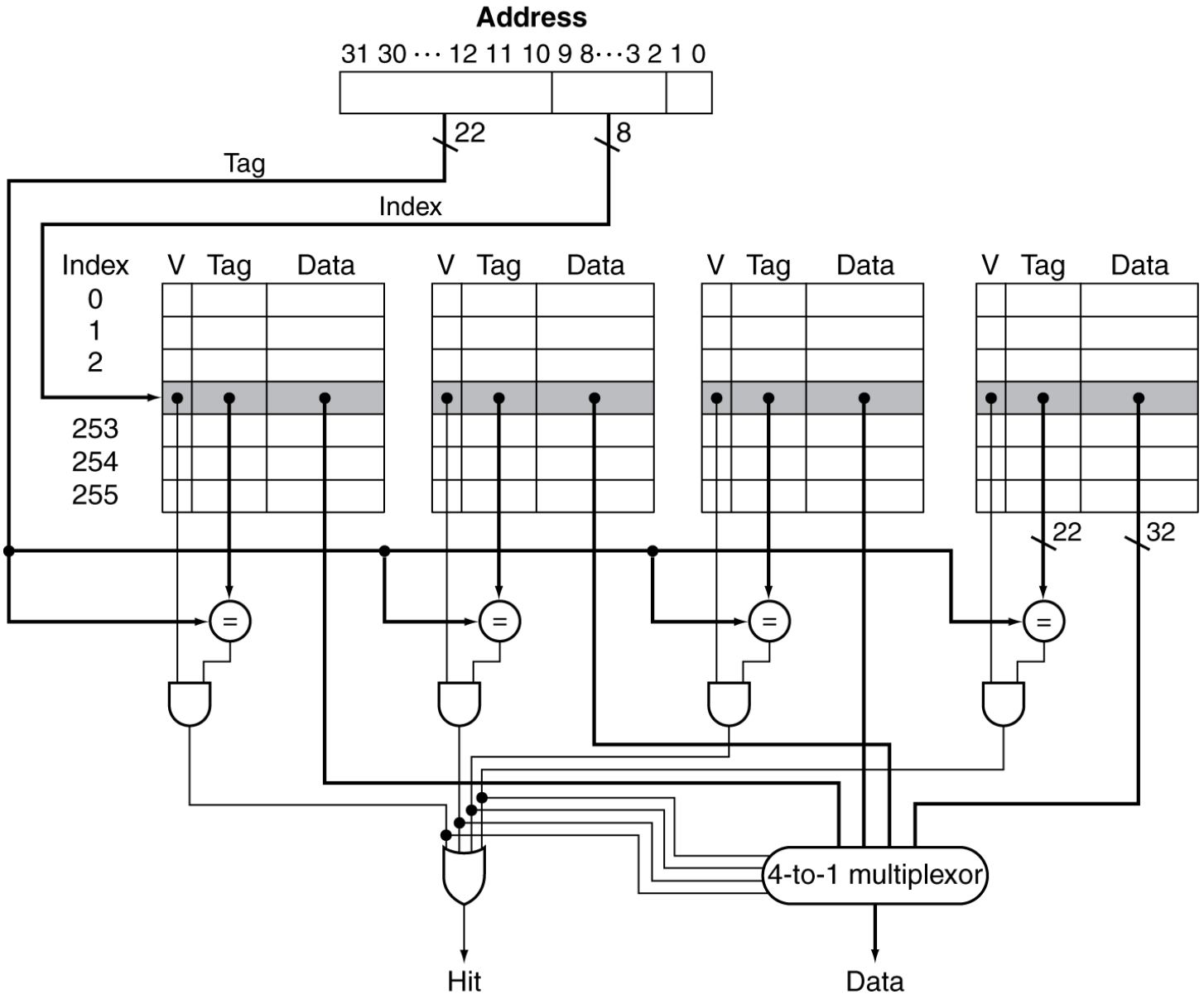
Increased associativity decreases miss rate

- But expensive

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%

4-way Set Associative Cache Organization



Replacement Policy

Which one should be evicted?

Direct mapped: no choice

Set associative

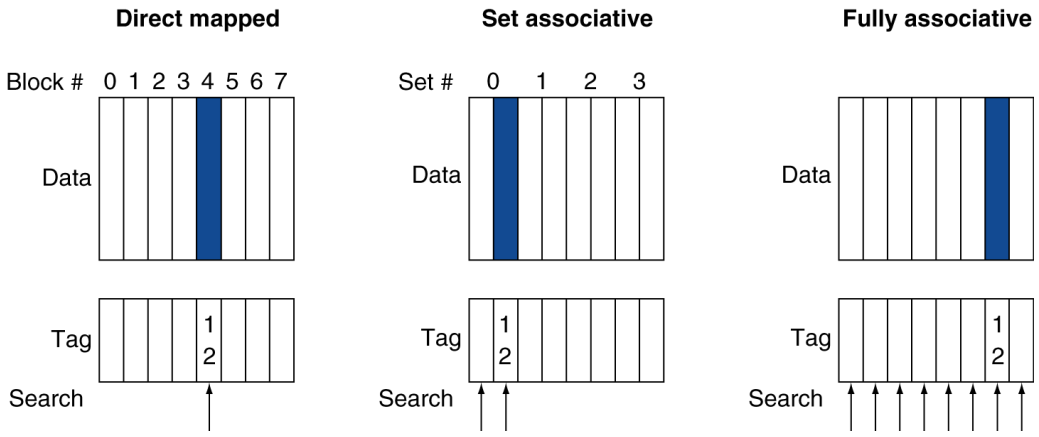
- Prefer non-valid entry, if there is one
- Otherwise, choose among entries in the set

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 cache attached to CPU

- Small, but fast

Level-2 cache services misses from primary cache

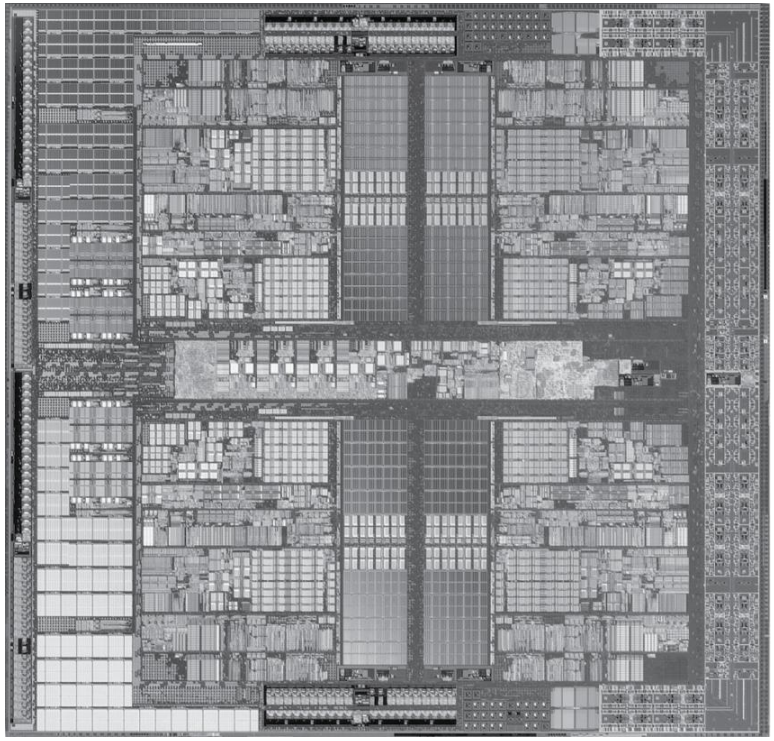
- Larger, slower, but still faster than main memory

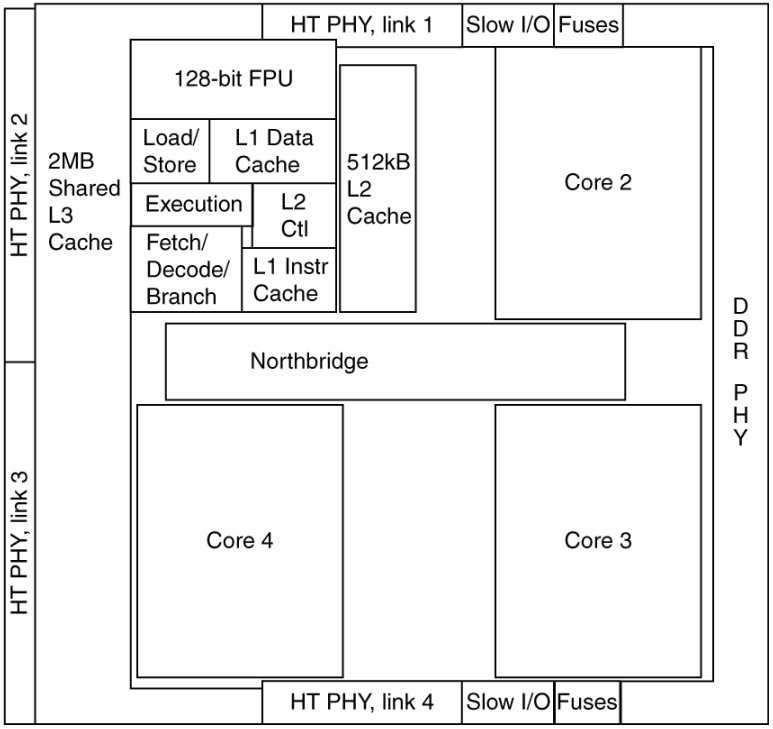
Main memory services L-2 cache misses

Current systems include L-3 cache

Inside the Processor (CPU)

AMD Barcelona: 4 processor cores





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

Example (cont.)

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 = 400 cycles

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

Performance ratio = 9/3.4 = 2.6

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 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 L-2 cache
- L-1 block size smaller than L-2 block size

Interactions with Advanced CPUs

Out-of-order CPUs can execute instructions during cache miss

- Pending load/store stays in load/store unit
  - Dependent instructions wait in reservation stations
- Independent instructions continue

Effect of miss depends on program data flow

- Much harder to analyze
- Use system simulation

Interactions with Software

Misses depend on memory access patterns

- Algorithm behavior
- Compiler optimization for memory access