

Cache

cache (kash) *n.* : hidden storage for
valuables

Computer Cache

- Fast storage for valuable data/instructions that are likely to be accessed in the future
- Speeds up access for these data/instructions
- Cache is essentially a hash table: stores a key (address), value pair



Addr	Data

Cache
0.5ns

Data	Data	Data

Memory
50ns

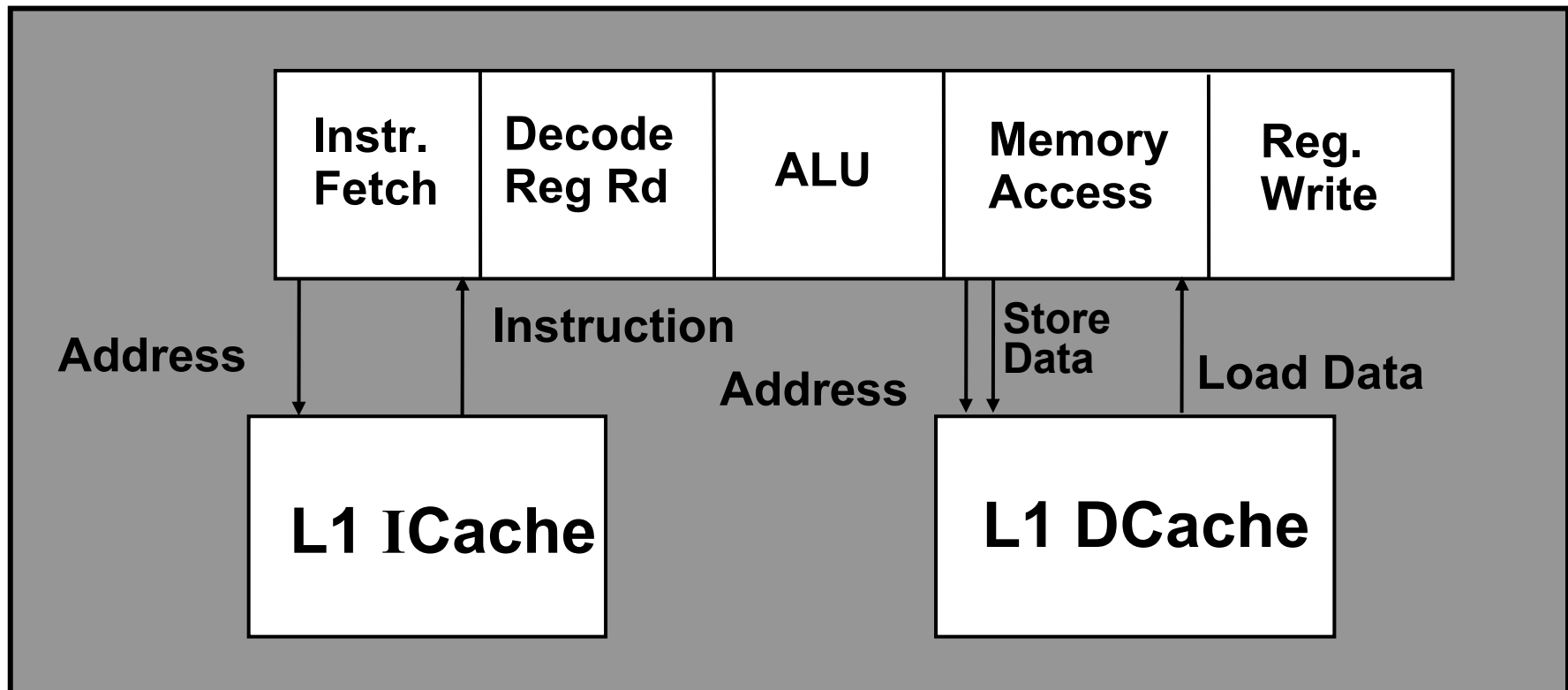
Memory Technology

- Static RAM (SRAM)
 - 0.5ns – 2.5ns, \$1000 – \$4000 per GB
- Dynamic RAM (DRAM)
 - 20ns – 60ns, \$5 – \$50 per GB
- Magnetic disk
 - 5ms – 20ms, \$0.05 – \$1 per GB
- Ideal memory
 - Access time of SRAM
 - Capacity and cost/GB of disk

On-Chip Level 1 Caches

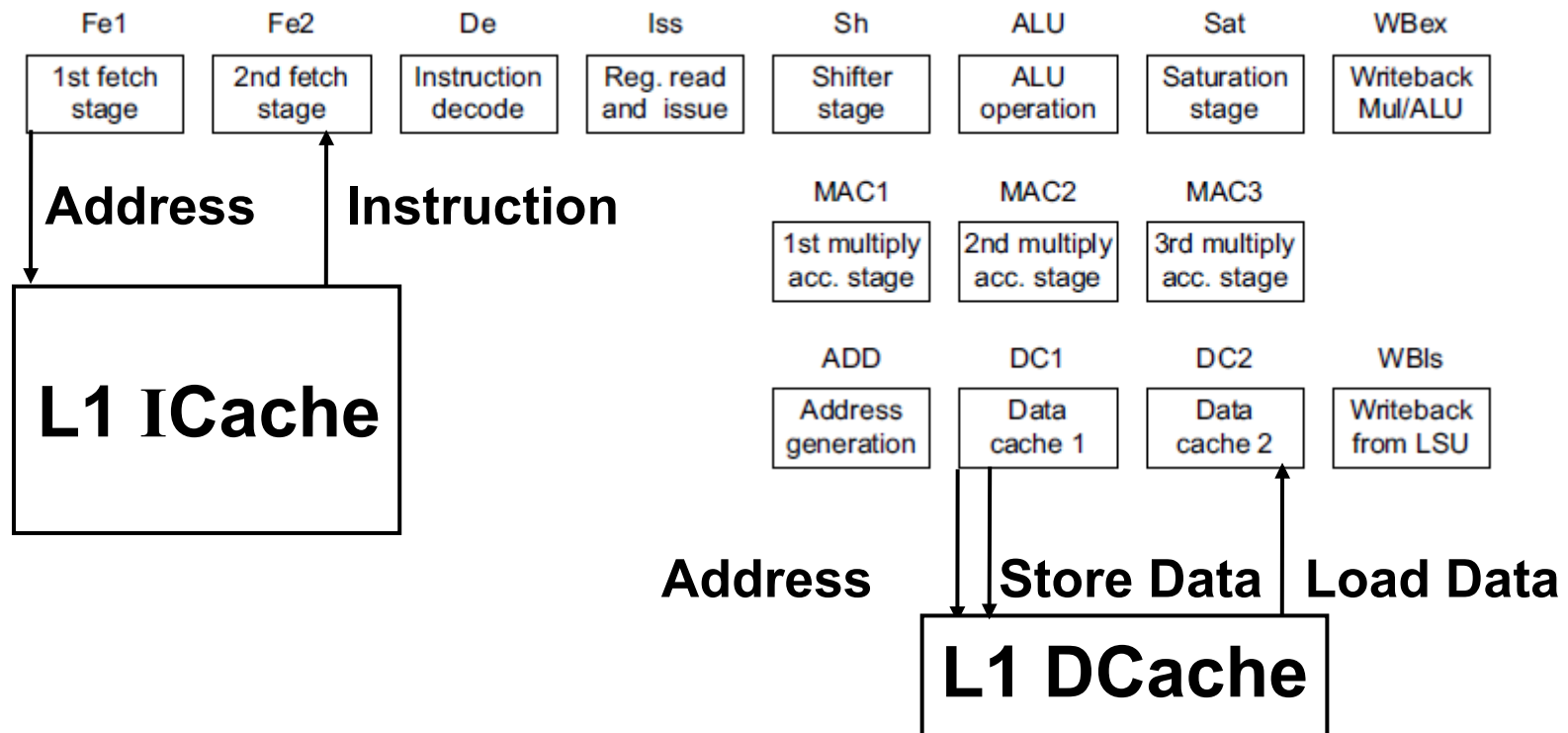
- Two small Level 1 (L1) caches are used to feed the pipeline
 - One for instructions: ICache
 - One for data: DCache

Processor



Level 1 Caches

- L1 caches must be fast (small) to keep up with pipeline
 - 0.5ns access time for 2 Ghz processor
- Most current processors, including the Pi's ARM, use two or more pipeline stages



Level 1 Caches: Real Stuff

- Intel 2.0 GHz Xeon E5-2650 (Sandy Bridge) has 32Kbyte L1 ICache and DCache
 - 4 clock cycles to access L1 cache = 2.0ns
- ARM 700 MHz 1176JZFS has 16Kbyte L1 ICache and DCache
 - 2 clock cycles to access L1 cache = 2.86ns
- Intel Xeon has faster clock rate (lower CCT): 2.0 GHz vs. 0.7 GHz for ARM
- Which has better CPI?

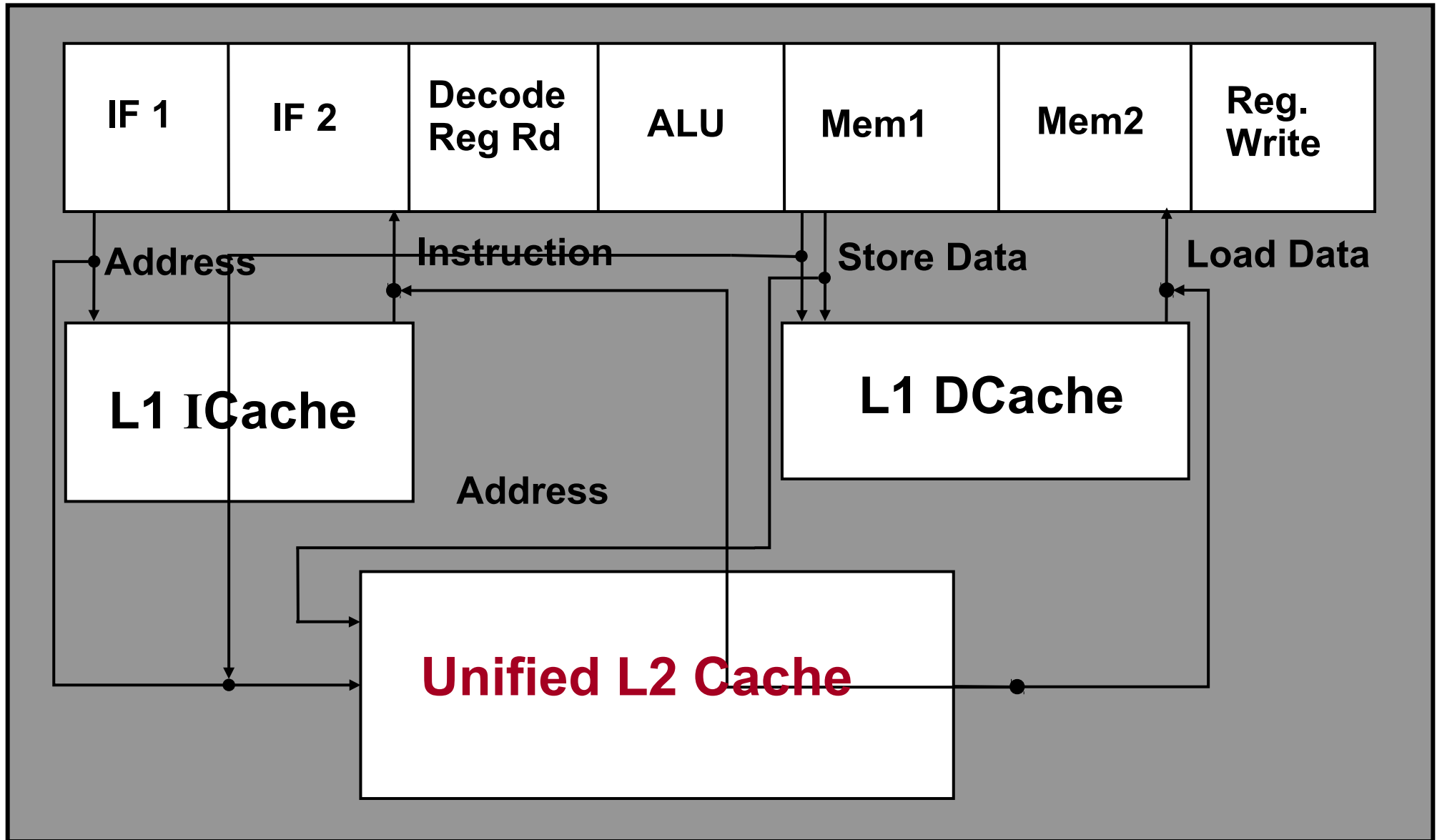
Off-Chip Access Penalty

- Accessing off-chip memory very slow because:
 - Large capacitance from macroscopic pins on CPU package
 - Large capacitance from circuit board wiring connecting CPU and off-chip memory
- Implies we want more on-chip memory, does not have to keep up with pipeline

Two-Level Cache

- Second type of on-chip memory that is larger and slower than L1 cache
- Typically L2 cache is **unified**, holds both instruction and data
- If instruction or data is not found in L1 cache, hopefully it will be in L2

L2 Cache

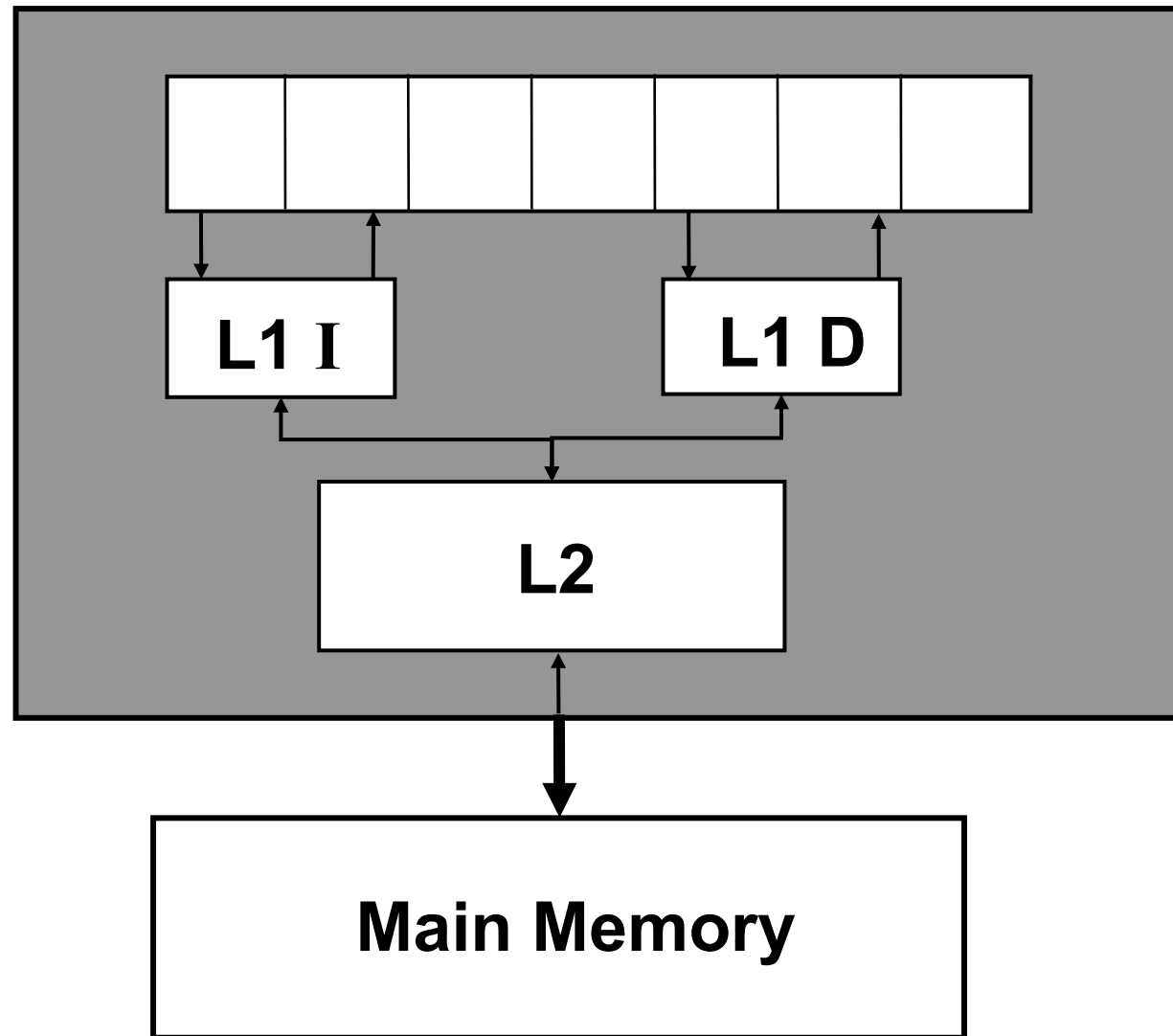


Level 2 Cache: Real Stuff

- ARM 1176JZFS doesn't really have L2 (it does, but it belongs to the GPU, not the CPU).
- Intel 2.0GHz E5-2650 has 256 Kbyte L2 cache
 - Also has 20 MB L3 cache

Main Memory

- Memory access requires a slow off-chip access



Principle of Locality

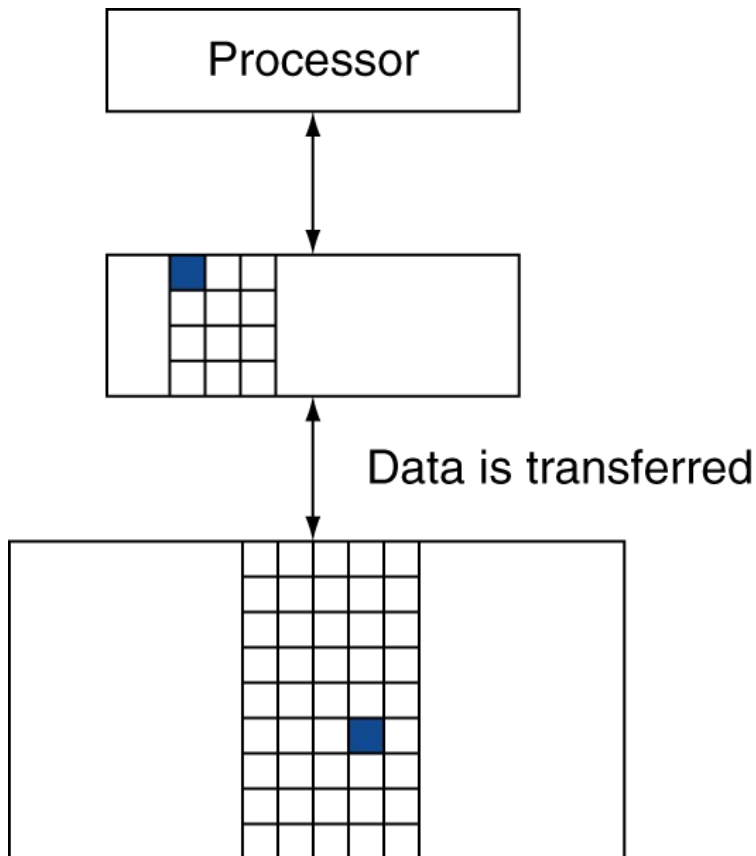
- Programs access a small proportion of their address space at any time
- Temporal locality
 - Items accessed recently are likely to be accessed again soon
 - e.g., instructions in a loop, induction variables
- Spatial locality
 - Items near those accessed recently are likely to be accessed soon
 - E.g., 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
 $= 1 - \text{hit ratio}$
 - Then accessed data supplied from upper level



Cache Organizations

- Direct Mapped
- Set Associative
- Fully Associative

Cache Protocol: Direct Mapped

- Where in cache should data be placed so it can be quickly located?
- Direct Mapped (Simplest):
 - Cache size is selected to be 2^i , e.g., 2^{16}
 - The entire 32-bit address is used to locate an instruction/data in main memory
 - The lower i bits (e.g., 16) are used to (quickly) find an item in the cache
 - Implies that each instruction/data from main memory is copied to (mapped to) a specific cache location

Four Questions for Caches and Memory Hierarchy

Q1: Where can a block be placed in the upper level?

(Block placement)

Q2: How is a block found if it is in the upper level?

(Block identification)

Q3: Which block should be replaced on a miss?

(Block replacement)

Q4: What happens on a write?

(Write strategy)

Direct Mapped Questions

Q1: Where can a block be placed in the upper level?

(Block placement)

- Answer: Single location
- Bookshelf example: One slot for books whose authors begin with P (“Patterson, David”)

Q2: How is a block found if it is in the upper level?

(Block identification)

Q3: Which block should be replaced on a miss?

(Block replacement)

Direct Mapped Questions

Q1: Where can a block be placed in the upper level?

(Block placement)

Q2: How is a block found if it is in the upper level?

(Block identification)

- Look in “P” slot. Is that enough?
- No. Must compare “atterson, David”. Called a “tag”.

Q3: Which block should be replaced on a miss?

(Block replacement)

Direct Mapped Questions

Q1: Where can a block be placed in the upper level?

(Block placement)

Q2: How is a block found if it is in the upper level?

(Block identification)

Q3: Which block should be replaced on a miss?

(Block replacement)

- When we go to the library and get Patterson's book, what should we replace?
- Whatever's in the "P" slot.

Direct Mapped Organization

Data has **only one** place to go, determined by address (**$\text{addr mod num_blocks}$**)

Possible parameters: cache size, block size (num blocks = cache size / block size)

It makes the most sense to make a “block” a series of contiguous memory locations

Example: 1 KB cache with 32B blocks

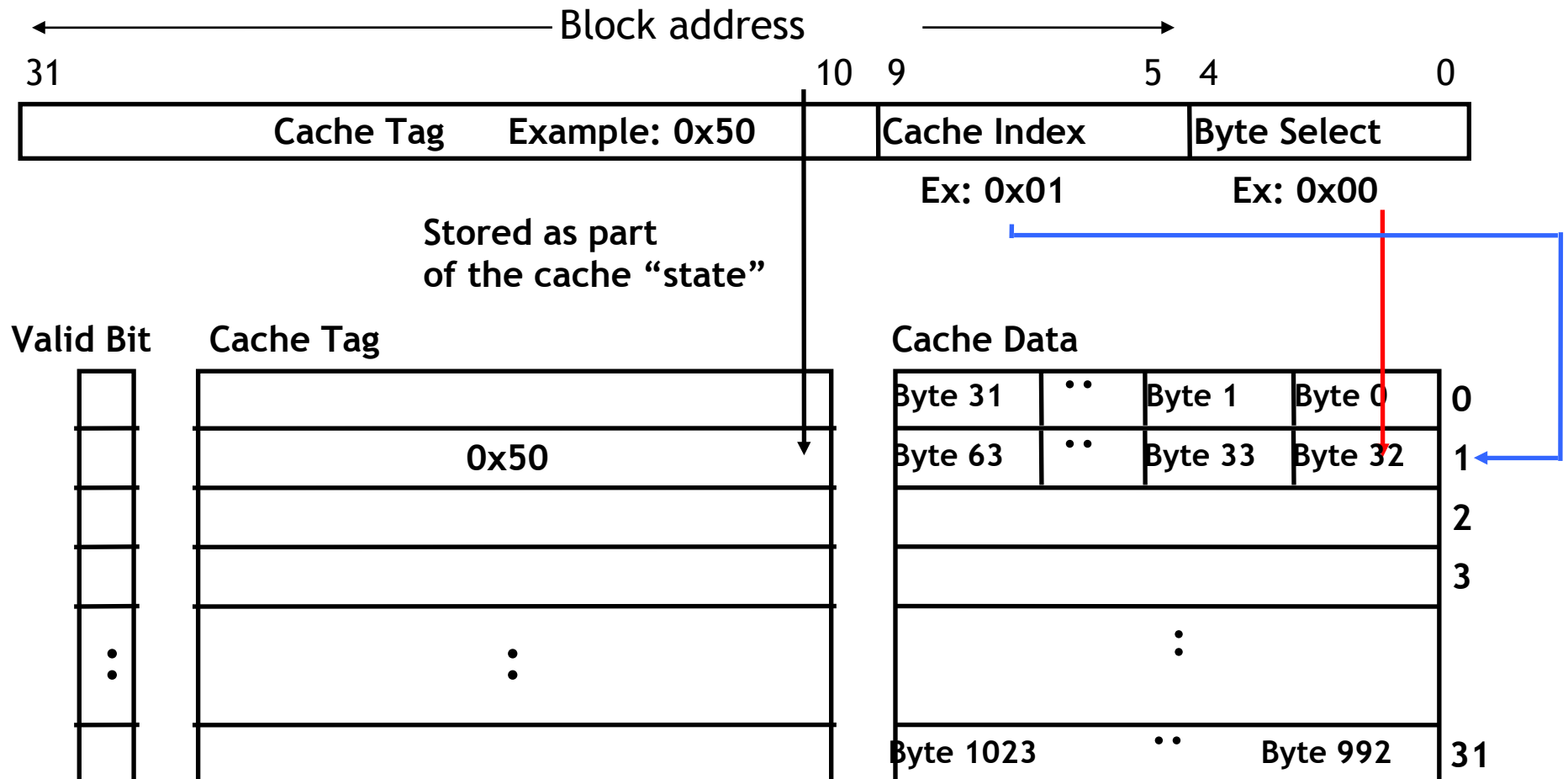
- How many bits specify a byte within a block?
- How many blocks?
- How many bits specify which block?
- The rest of the bits are the tag

When would big blocks be good?
When would small blocks be good?

Example: 1 KB Direct Mapped Cache with 32 B Blocks

For a 2^N byte cache:

- The uppermost $(32 - N)$ bits are always the Cache Tag
- The lowest M bits are the Byte Select (Block Size = 2^M)
- On cache miss, pull in complete “Cache Block” (or “Cache Line”)



Example: Direct Mapped Cache

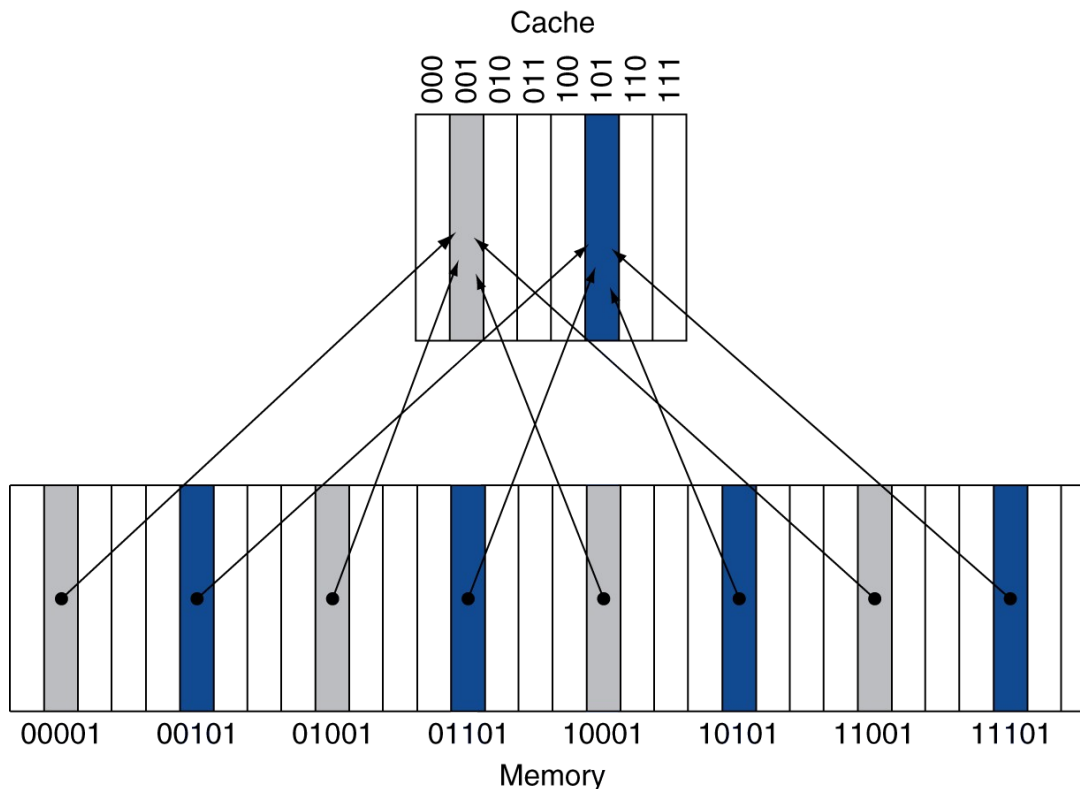
Direct-mapped cache has 4 1-word (4-byte) blocks

Behavior w/ memory addr sequence 0,8,0,6,8

Structure of Tag|Index|Byte Select?

Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
 - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits

Direct Mapped Problem

Direct-mapped cache has 4 1-word blocks

Let's say our sequence is 0, 4, 0, 4 ...

“Conflict” miss - these two addresses conflict

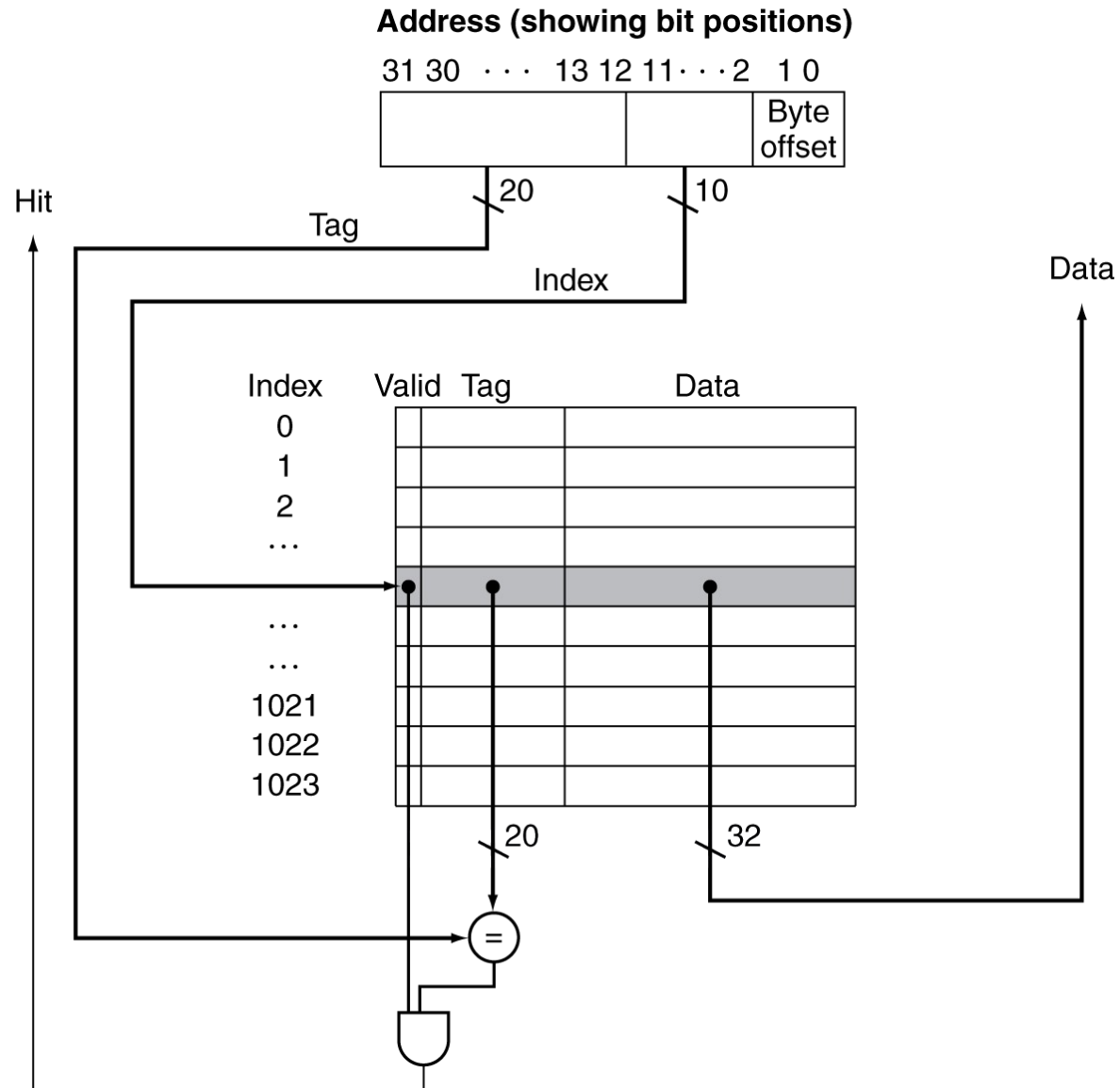
- But other blocks are empty!

Why? Only one place for each block to go.

What if we had TWO places for each block to go, and it could pick either one?

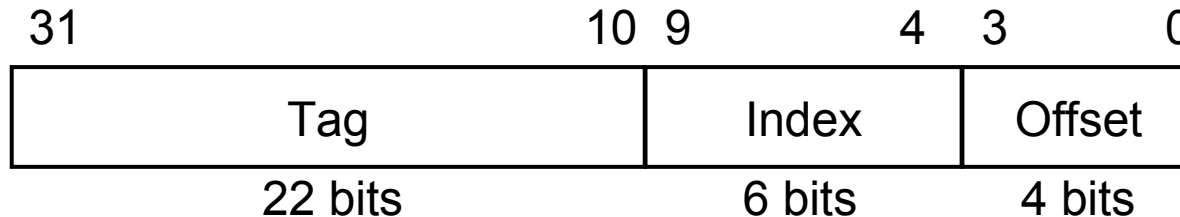
- Bookshelf example: 2 slots for OP

Address Subdivision



Example: Larger Block Size

- 64 blocks, 16 bytes/block
 - To what block number does address 1200 map?
- Block address = $\lfloor 1200/16 \rfloor = 75$
- Block number = $75 \text{ modulo } 64 = 11$



Block Size Considerations

- Larger blocks should reduce miss rate
 - Due to spatial locality
- 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

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

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	10	Mem[10010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

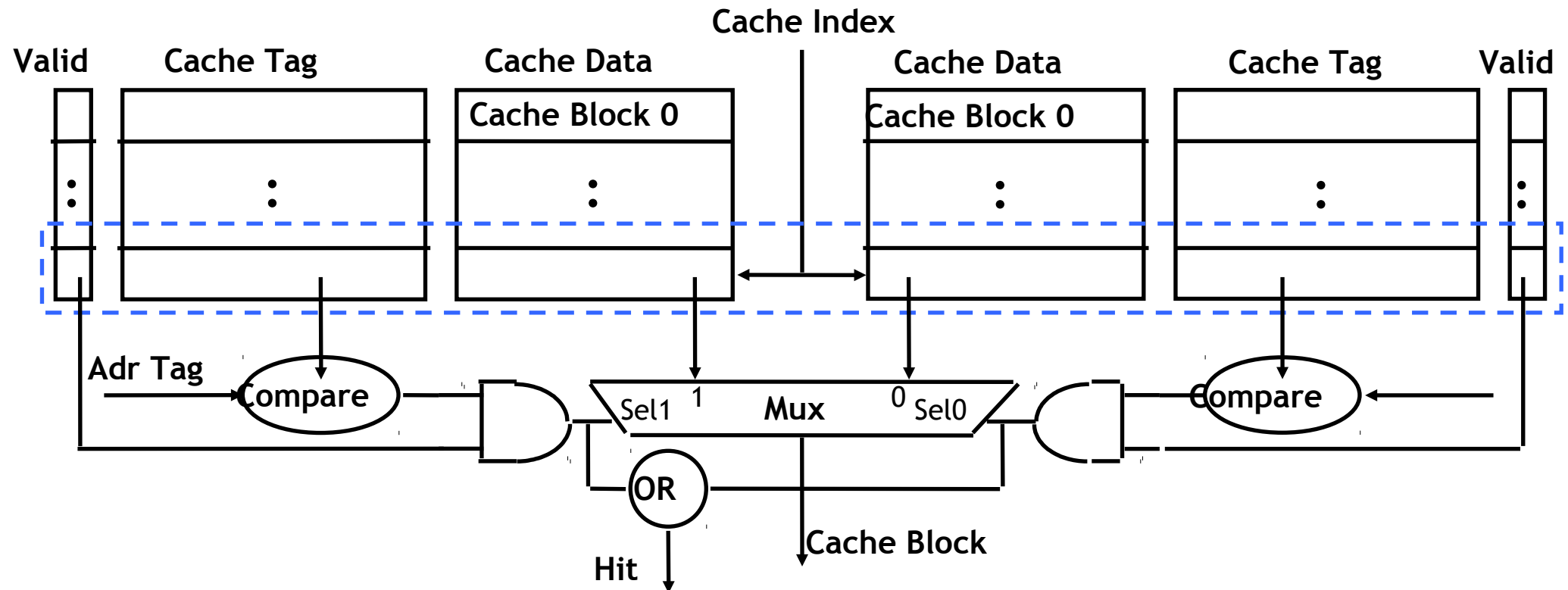
Example: Set Associative Cache

N-way set associative: N entries for each Cache Index

- N direct mapped caches operates in parallel

Example: Two-way set associative cache

- Cache Index selects a “set” from the cache
- The two tags in the set are compared to the input in parallel
- Data is selected based on the tag result



2-Way Set Associative Problem

2-way set associative cache has 4 1-word blocks

- 2 sets x 2 entries
- 1/2 as many entries as direct mapped

Let's say our sequence is 0, 4, 0, 4 ...

2-Way Set Associative Problem

2-Way Set Associative cache has 4 1-word blocks

Behavior w/ memory addr sequence 0,8,0,6,8

Structure of Tag|Index|Byte Select?

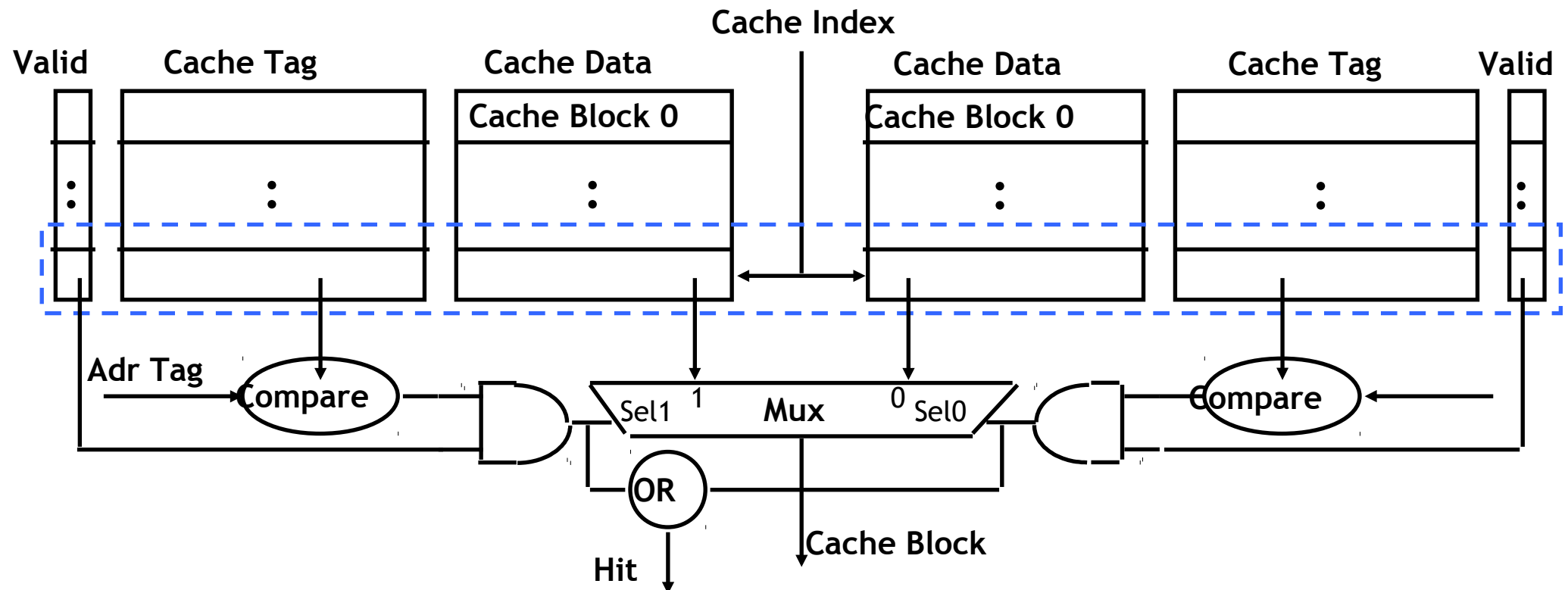
Disadvantage of Set Associative Cache

N-way Set Associative Cache versus Direct Mapped Cache:

- N comparators vs. 1
- Extra MUX delay for the data
- Data comes AFTER Hit/Miss decision and set selection

In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:

- Possible to assume a hit and continue. Recover later if miss.

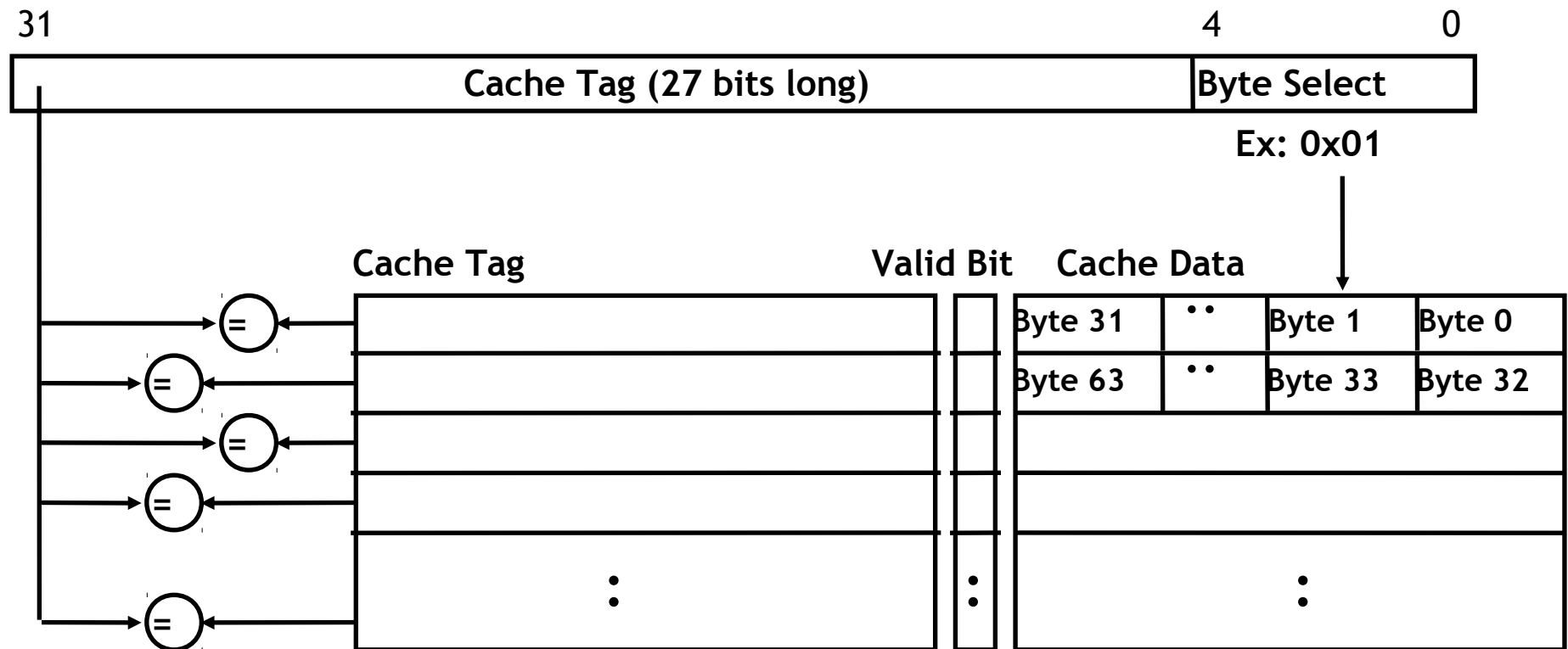


Example: Fully Associative

Fully Associative Cache

- Forget about the Cache Index
- Compare the Cache Tags of all cache entries in parallel
- Example: Block Size = 32 B blocks, we need N 27-bit comparators

By definition: Conflict Miss = 0 for a fully associative cache



Fully Associative Problem

Fully associative cache has 4 1-word blocks

Behavior w/ memory addr sequence 0,8,0,6,8

Structure of Tag|Index|Byte Select?

A Summary on Sources of Cache Misses

Compulsory (cold start or process migration, first reference): first access to a block

- “Cold” is a fact of life: not a whole lot you can do about it
- Note: If you are going to run “billions” of instructions, Compulsory Misses are insignificant

Capacity:

- Cache cannot contain all blocks accessed by the program
- Solution: increase cache size

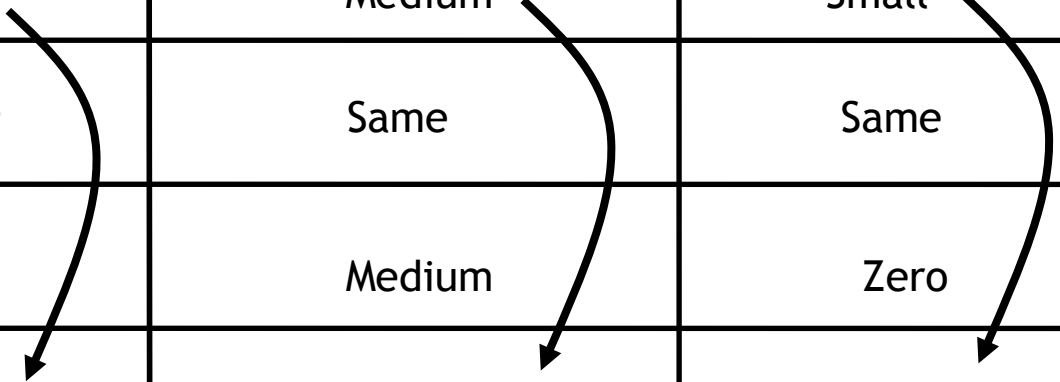
Conflict (collision):

- Multiple memory locations mapped to the same cache location
- Solution 1: increase cache size
- Solution 2: increase associativity

Coherence (Invalidation): other process (e.g., I/O) updates memory

Design options at constant cost (#bits)

	Direct Mapped	N-way Set Associative	Fully Associative
Cache Size	Big	Medium	Small
Compulsory Miss	Same	Same	Same
Conflict Miss	High	Medium	Zero
Capacity Miss	Low	Medium	High
Coherence Miss	Same	Same	Same



Note:

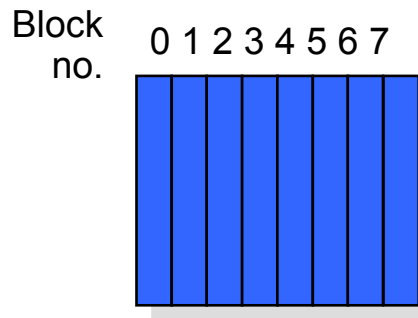
If you are going to run “billions” of instruction, Compulsory Misses are insignificant (except for streaming media types of programs).

Q1: Where can a block be placed in the upper level?

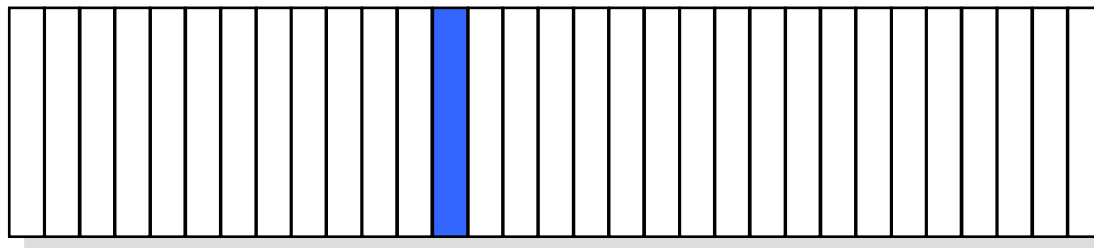
Block 12 placed in 8 block cache:

- Fully associative, direct mapped, 2-way set associative
- S.A. Mapping = Block Number Modulo Number Sets

Fully associative:
block 12 can go
anywhere



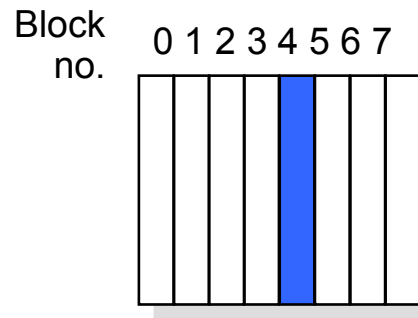
Block-frame address



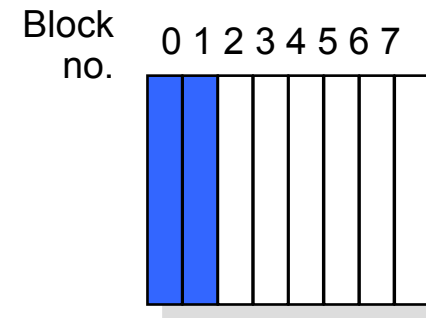
Block no.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Direct mapped:
block 12 can go only
into block 4 (12 mod 8)

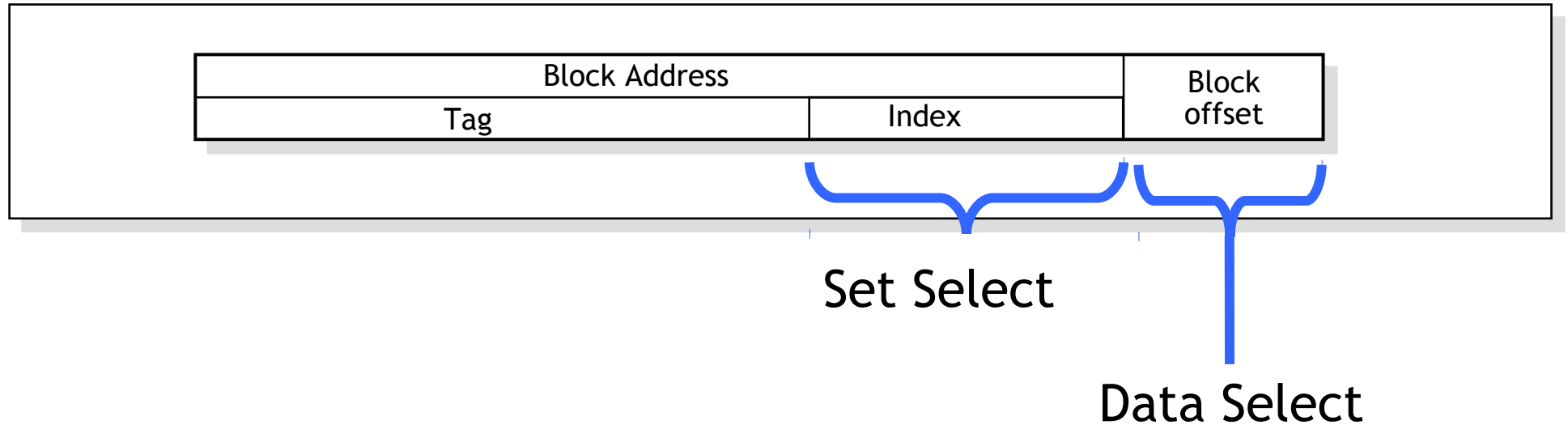


Set associative:
block 12 can go
anywhere in set 0
(12 mod 4)



Set Set Set Set
0 1 2 3

Q2: How is a block found if it is in the upper level?



Direct indexing (using index and block offset), tag compares, or combination

Increasing associativity shrinks index, expands tag

- Fewer sets, more possibilities per set

Q3: Which block should be replaced on a miss?

What do we really want?

Easy for Direct Mapped

Set Associative or Fully Associative:

- Random
- LRU (Least Recently Used)

Associativity	2 way	2 way	4 way	4 way	8 way	8 way
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Q4: What happens on a write?

What happens on a write miss?

- Don't have to read data or check tags - just write!

Writes are more complex than reads:

- Desirable: Cache is always a subset of memory (“consistent”)
- So when we write to the cache, we also should write to the memory as well (“write through”)
 - If we don't do this: “write back”
- Write-through could lead to performance problems - writing to memory could mean performance is proportional to memory time instead of cache time
- Solution: “write buffer”

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 = $1 + 0.1 \times 100 = 11$
- Solution: write buffer
 - Holds data waiting to be written to memory
 - CPU continues immediately
 - Only stalls on write if write buffer is already full

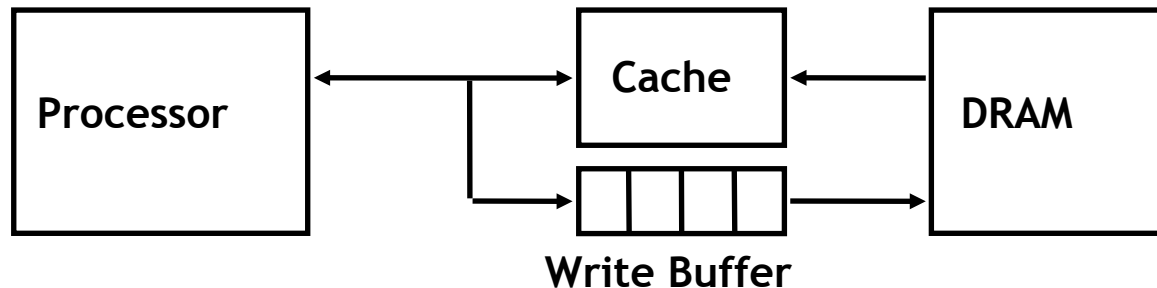
Write-Back

- 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

Write Buffer for Write Through



A Write Buffer is needed between the Cache and Memory

- Processor: writes data into the cache and the write buffer
- Memory controller: write contents of the buffer to memory

Write buffer is just a FIFO:

- Typical number of entries: 4
- **Must handle bursts of writes**
- Works fine if: Store frequency (w.r.t. time) $\ll 1 / \text{DRAM write cycle}$

Q4: What happens on a write?

Write through—The information is written to both the block in the cache and to the block in the lower-level memory.

- On a write miss, do we allocate space in the cache? Typically no, since all writes have to go to main memory anyway. This is called *write no-allocate*.

Write back—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.

- On a write miss, do we allocate space in the cache? Typically yes, since we hope future writes might also be captured by the cache. This is called *write allocate*.

Q4: What happens on a write?

Write through—The information is written to both the block in the cache and to the block in the lower-level memory.

Write back—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.

- is block clean or dirty?

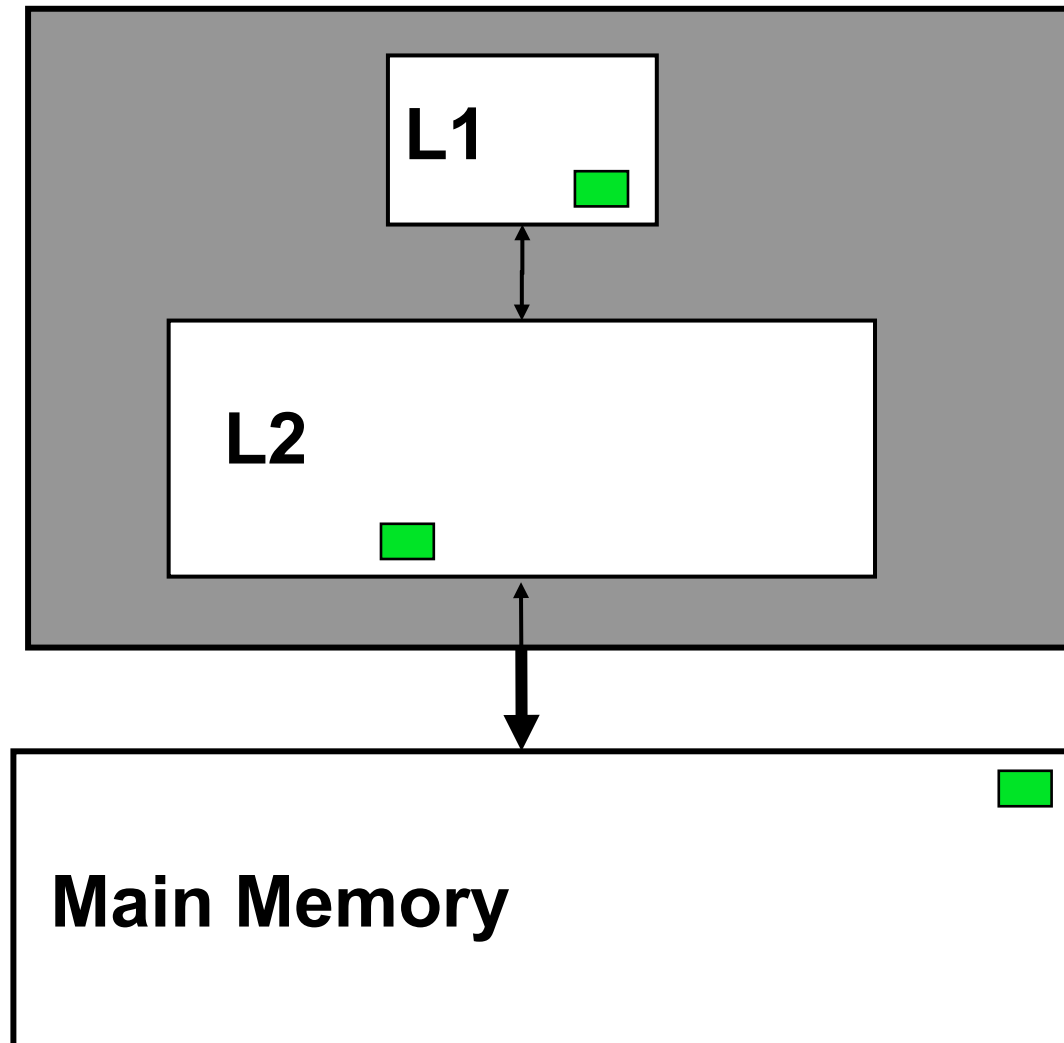
Pros and Cons of each?

- WT: read misses cannot result in writes, also simpler
- WB: no writes of repeated writes, more complex (dirty bit)

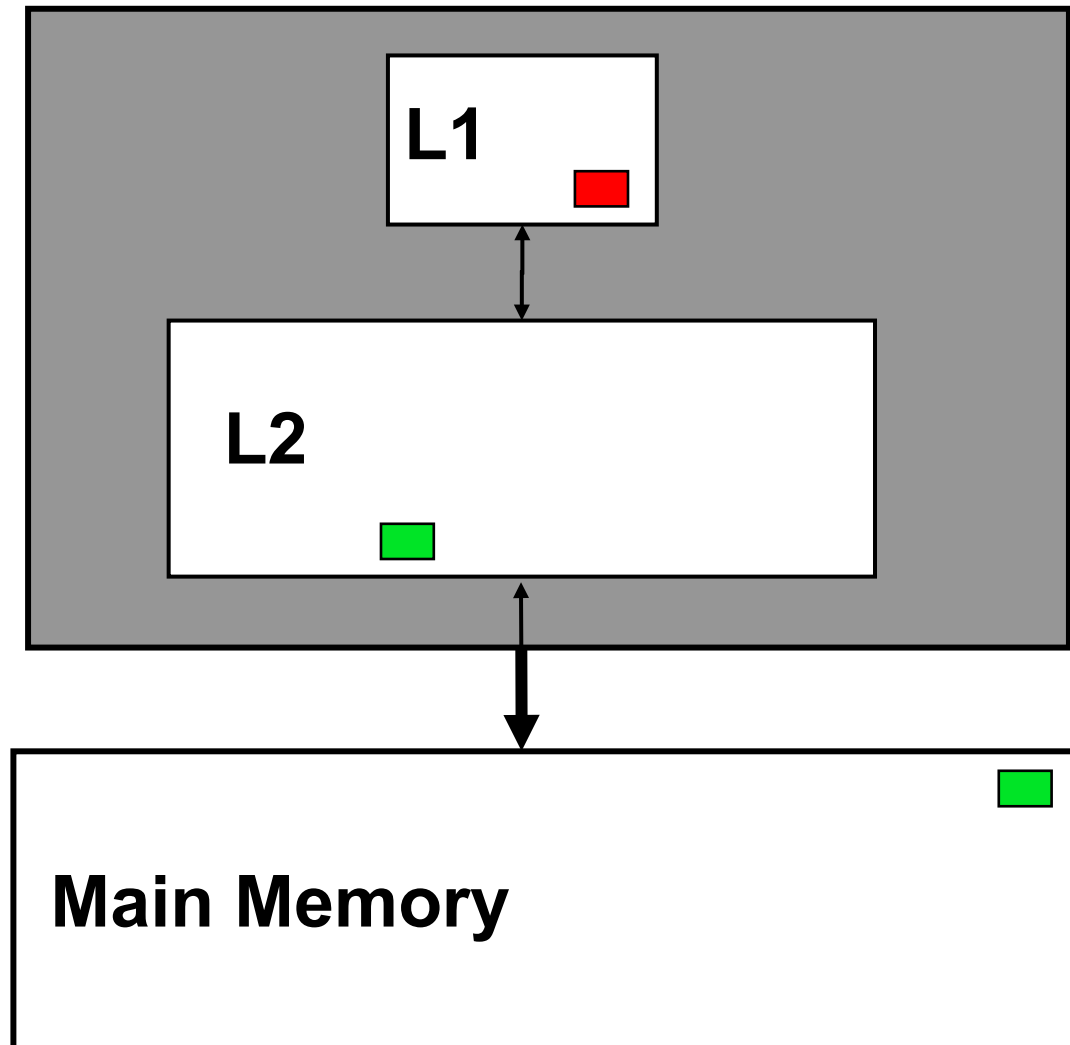
WT always combined with write buffers so that don't wait for lower level memory

Write Through

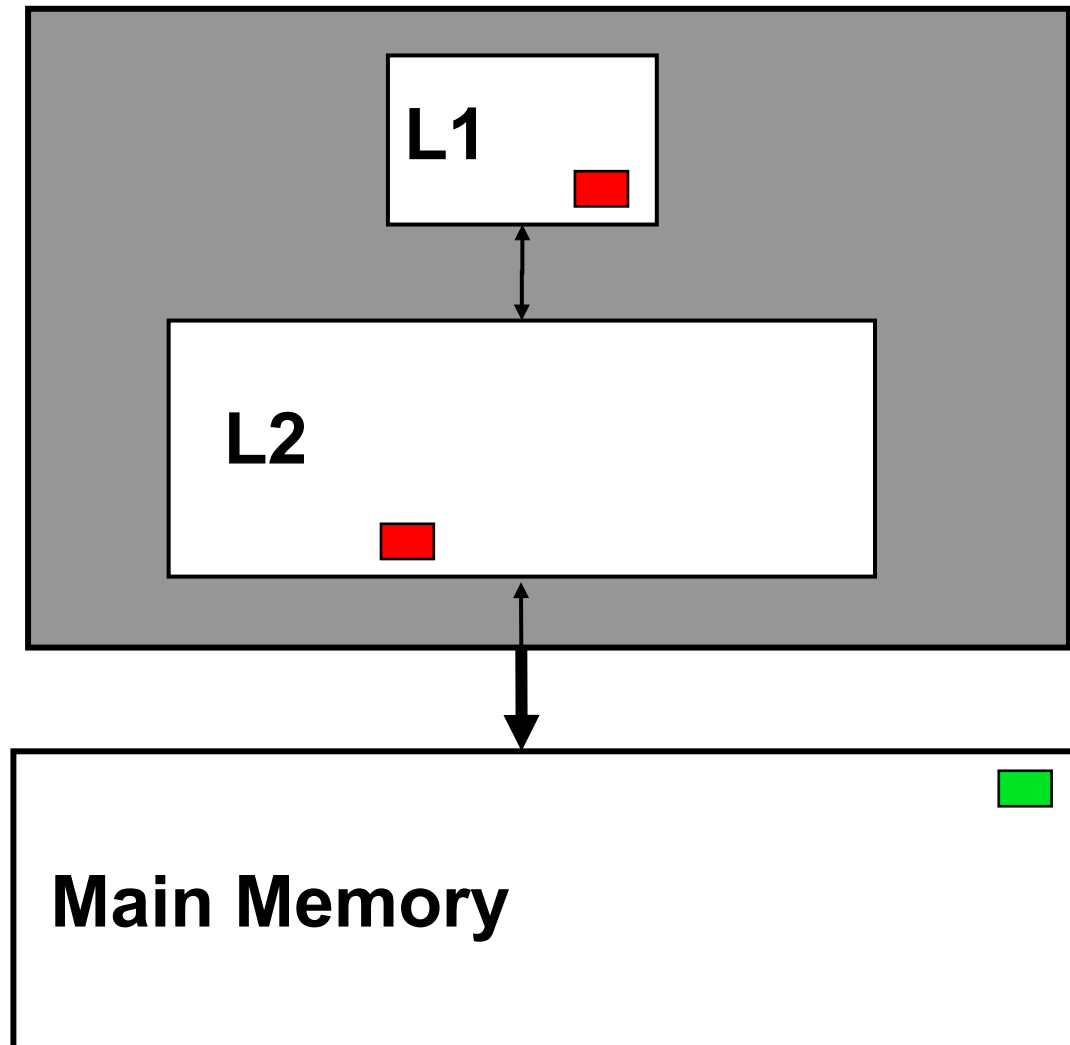
- When data changes in cache level i , change propagates immediately to level $i+1$



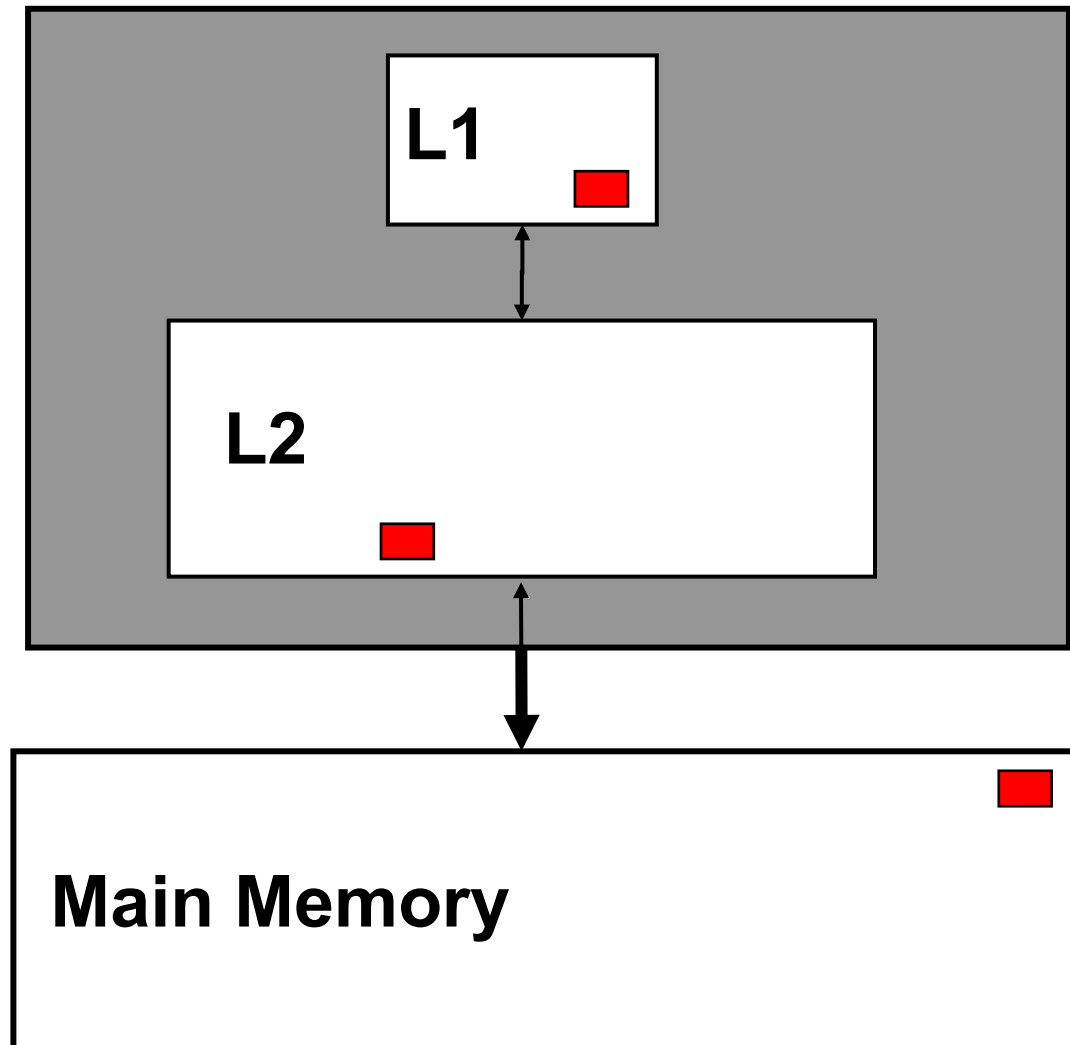
Write Through



Write Through



Write Through

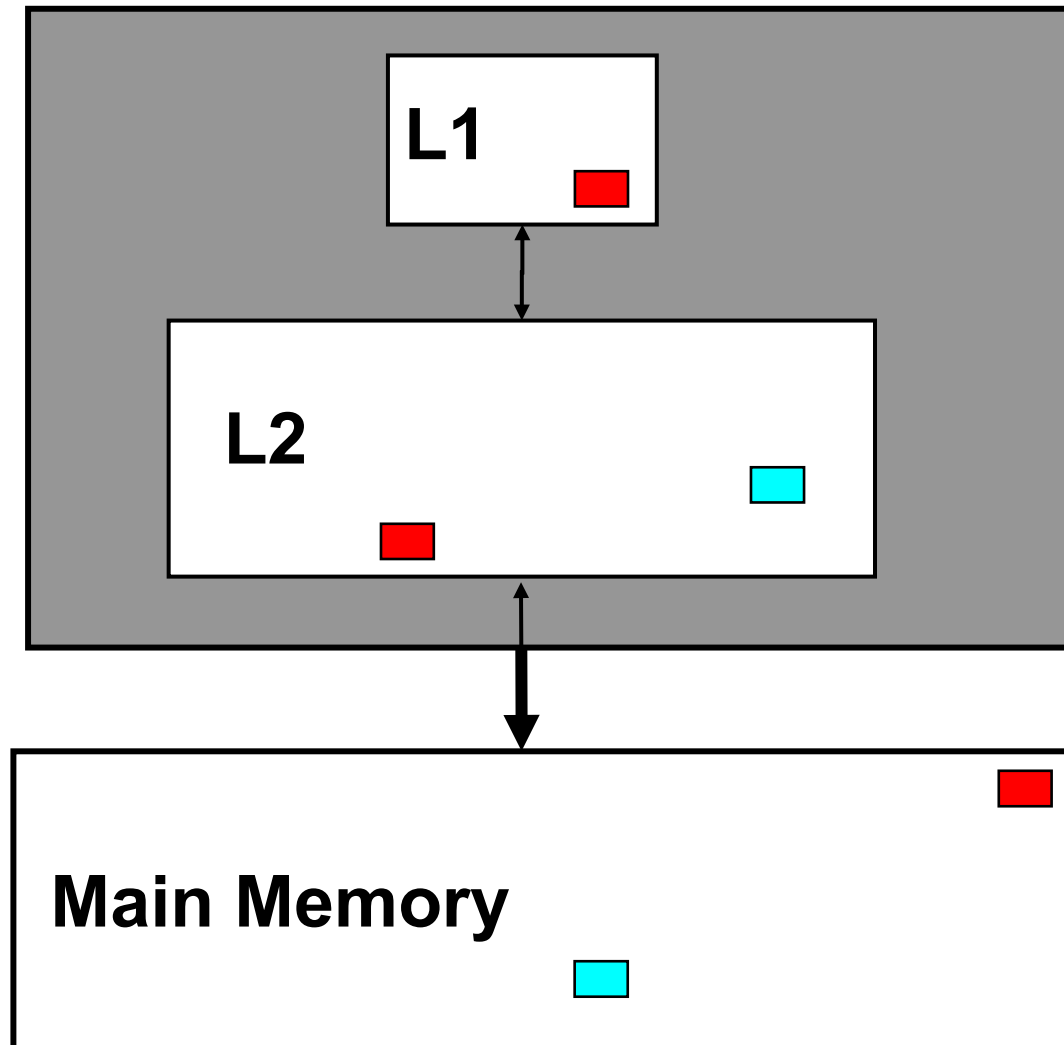


Write Through

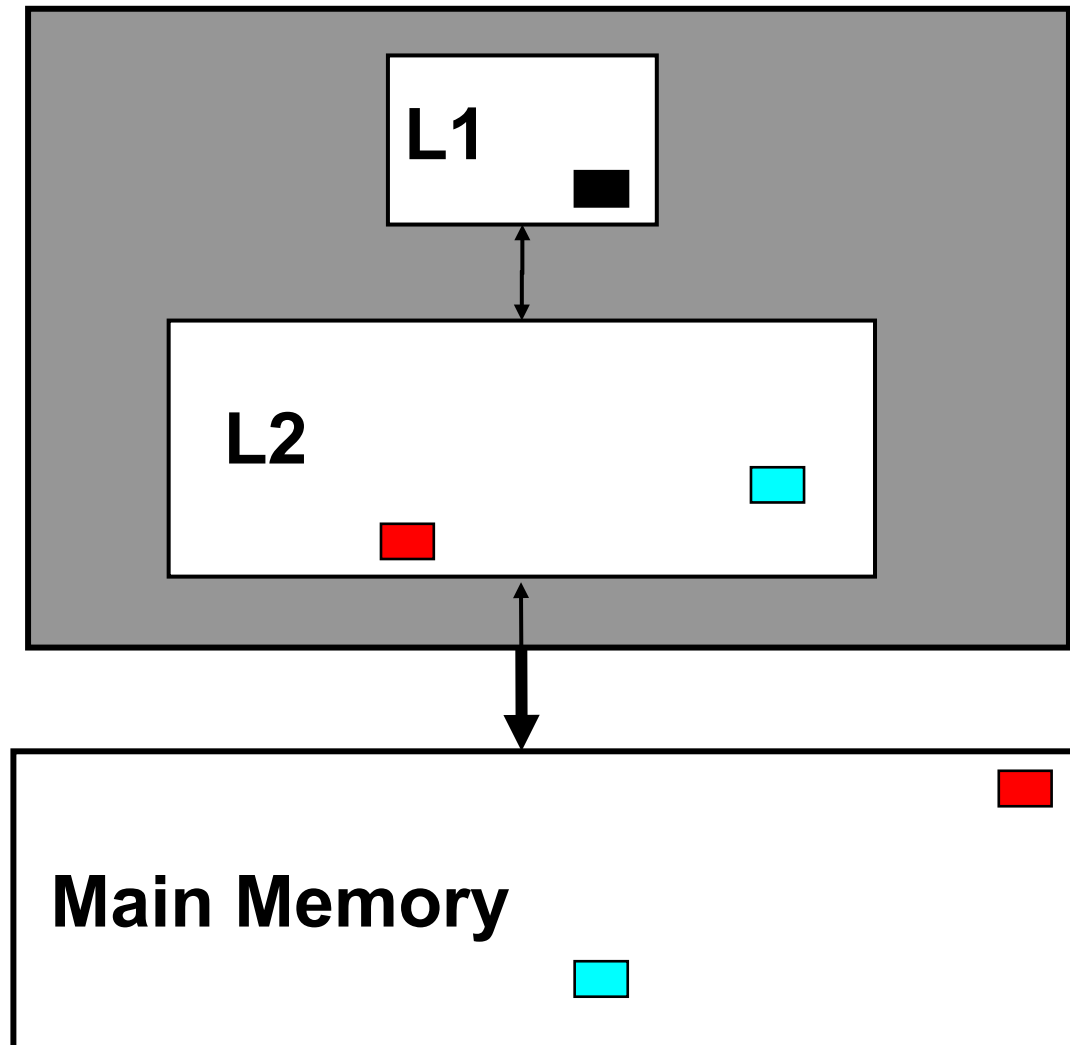
- Advantages
 - Block is now clean, can be overwritten on later replacement without delay
 - Block has a backup copy. If error is detected at level i , go to level $i+1$ to find a copy
- Disadvantages
 - Frequent writing uses memory bus, may slow concurrent I/O transfers or other processors sharing memory bus
 - Frequent writing consumes energy

Write Back

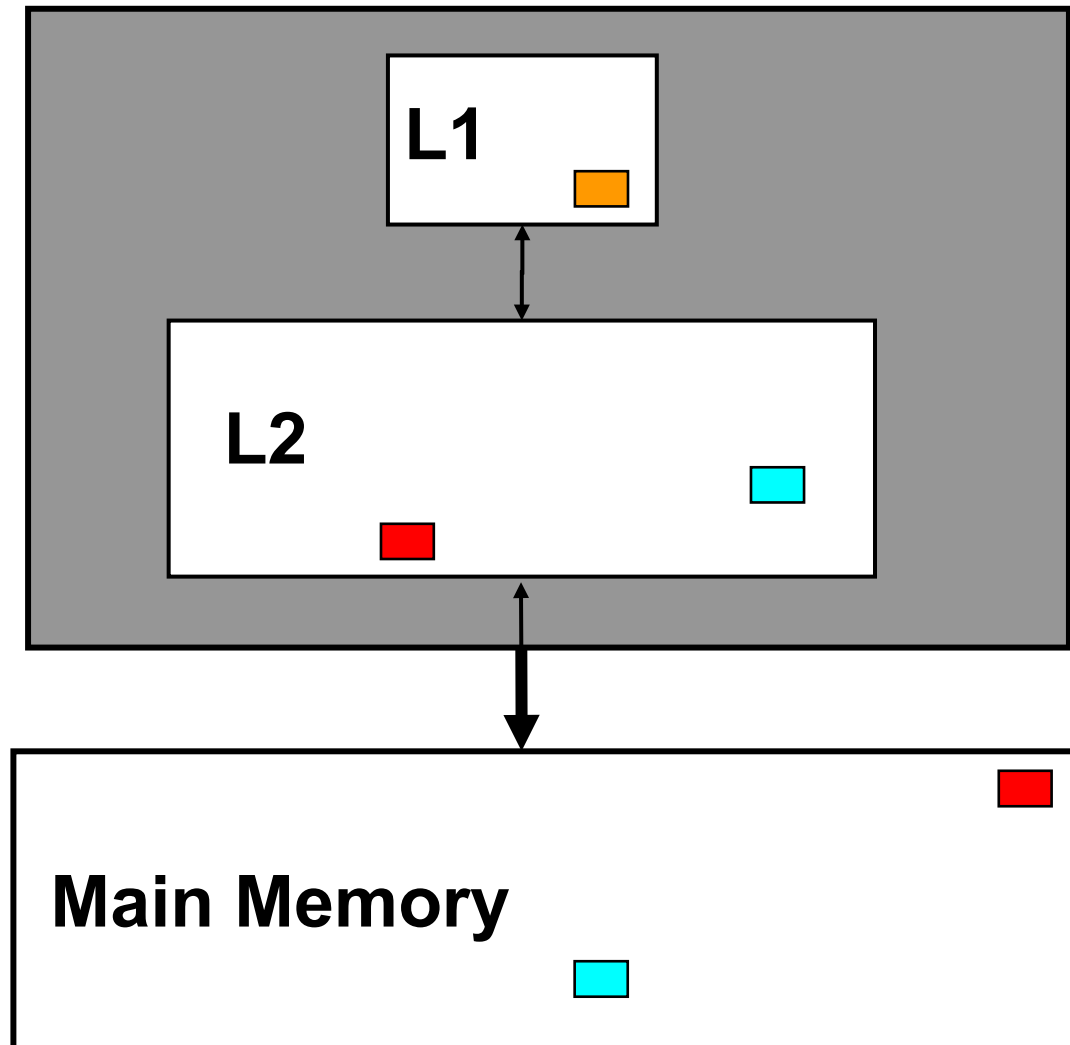
- Only write back dirty data when block is replaced



Write Back

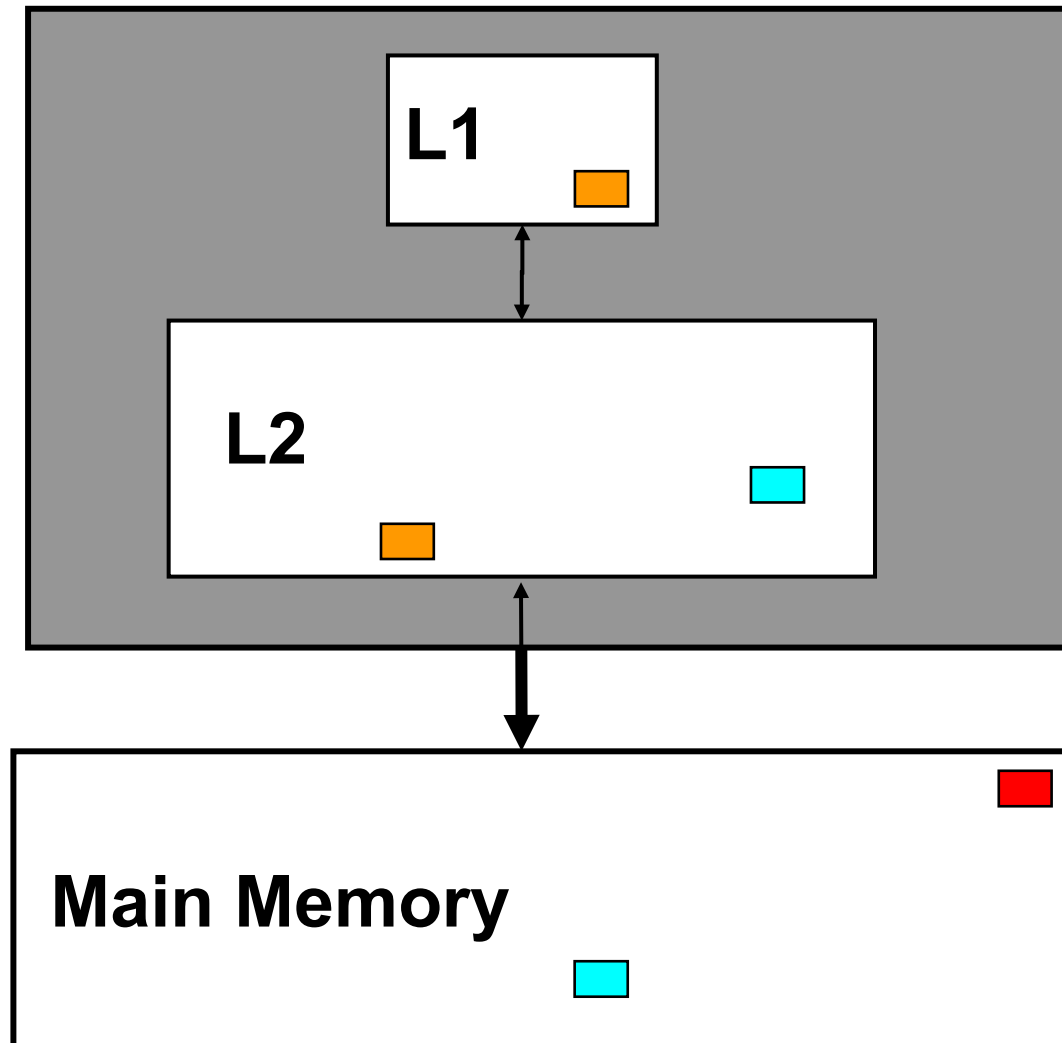


Write Back



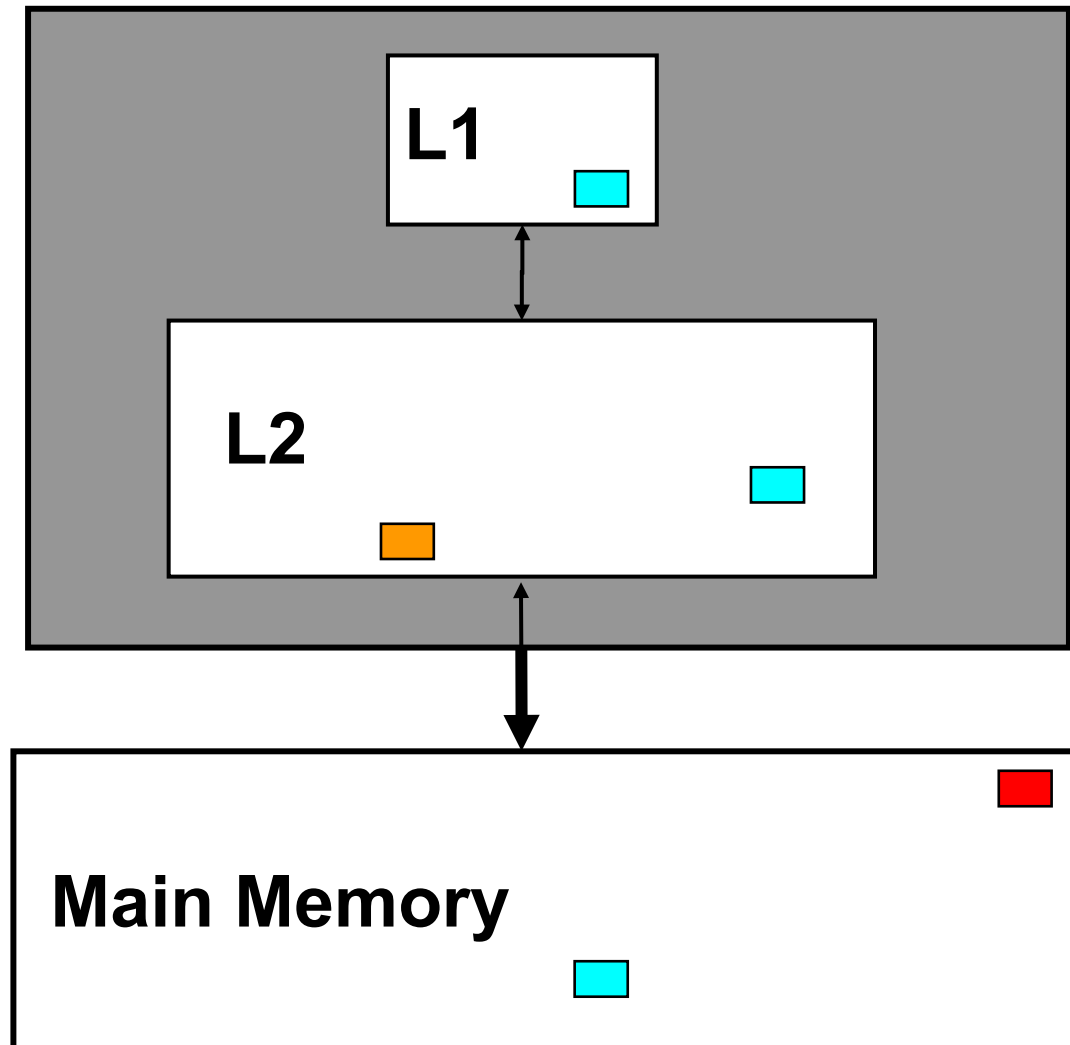
Write Back

- Need to replace L1 block. Write back to L2



Write Back

- Replace block in L1 with new block from L2



Write Back

- Disadvantages
 - Block is dirty, there's a delay when it's replaced
 - No backup copy if cache has an error
- Advantages
 - Less frequent writing reduces memory bus traffic
 - Less frequent writing consumes less energy

Typical Write Policy

- Write through from L1 to L2
 - The L1/L2 bus is private, no performance loss from contention
- Write back from L2 to memory
 - Memory bus is shared with I/O devices and possibly other processors, write back reduces bus contention

Cache Performance Example

- Given
 - I-cache miss rate = 2%
 - D-cache miss rate = 4%
 - Miss penalty = 100 cycles
 - Base CPI (ideal cache) = 2
 - Load & stores are 36% of instructions
- 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)
 - $AMAT = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$
- Example
 - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, l-cache miss rate = 5%
 - $AMAT = 1 + 0.05 \times 20 = 2\text{ns}$
 - 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

Improving Cache Performance: 3 general options

Time = IC x CT x (ideal CPI + memory stalls)

Average Memory Access time =
Hit Time + (Miss Rate x Miss Penalty) =

(Hit Rate x Hit Time) + (Miss Rate x Miss Time)

[note this means Miss Penalty + Hit Time = Miss Time]

Options to reduce AMAT:

1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.

Modern Cache Organizations

ARM 1176JZFS

- 16 KB 4-way L1 Icache
- 16 KB 4-way L1 write-back Dcache
- L1 has 2 cycle latency
- No L2

Intel Xeon

- 32 KB 8-way L1 ICache
- 32 KB 8-way L1 write-back Dcache (64B/line)
- L1 has 4 cycle latency
- 256 KB 8-way unified L2 (shared across cores)
- 20 MB 20-way unified L3 (shared across cores)

Average Memory Access Time

- Average memory access time (AMAT) is a function of the hit time, the miss rate, and the miss penalty (miss time)

Average Access time = hit_time x (1-miss_rate) + miss_time x miss_rate

- Access time decreases with if hit time, miss time, or miss rate decreases
- For two-level cache:

miss time L1 = hit_time_L2 x (1-miss_rate_L2) + miss_time_L2 x miss_rate_L2

AMAT Example

Example: 99% of questions I ask can be found in your textbook. It takes 1 minute to look up something in your textbook and 1 additional hour to go to campus to get a library book. What's AMAT (or ABAT, Average Book Access Time)?

Performance Impact

Suppose a processor executes at

Clock Rate = 2 GHz (.5 ns per cycle)

Base CPI = 1.1 (assuming 1-cycle cache hits)

50% arith/logic, 30% ld/st, 20% control

Suppose that 10% of data memory operations (lw, sw) get 50 cycle miss penalty

Suppose that 1% of instructions get same miss penalty

Performance Impact

Suppose a processor executes at

Clock Rate = 2 GHz (.5 ns per cycle)

Base CPI = 1.1 (assuming 1-cycle cache hits)

50% arith/logic, 30% ld/st, 20% control

Suppose that 10% of data memory operations (lw, sw) get 50 cycle miss penalty

Suppose that 1% of instructions get same miss penalty

What is AMAT if we only consider instructions?

$$\begin{aligned} \text{AMAT} &= \text{hit} + (\text{miss\%} \times \text{miss penalty}) \\ &= 1 + (1\% \times 50 \text{ cycles}) \\ &= 1.5 \end{aligned}$$

Performance Impact

Suppose a processor executes at

Clock Rate = 2 GHz (.5 ns per cycle)

Base CPI = 1.1 (assuming 1-cycle cache hits)

50% arith/logic, 30% ld/st, 20% control

Suppose that 10% of data memory operations (lw, sw) get 50 cycle miss penalty

Suppose that 1% of instructions get same miss penalty

What is AMAT if we only consider data?

$$\begin{aligned} \text{AMAT} &= \text{hit} + (\text{miss}\% \times \text{miss penalty}) \\ &= 1 + (10\% \times 50 \text{ cycles}) \\ &= 6 \end{aligned}$$

Performance Impact

Suppose a processor executes at
Clock Rate = 2 GHz (.5 ns per cycle)

Base CPI = 1.1 (assuming 1-cycle cache hits)

50% arith/logic, 30% ld/st, 20% control

Suppose that 10% of data memory operations (lw, sw) get 50 cycle miss penalty

Suppose that 1% of instructions get same miss penalty

What is AMAT if we consider both?

Weighted sum! Weight instrs by 1.0/1.3, data by .3/1.3

$$\text{AMAT} = (1.0/1.3) \text{ InstrAMAT} + (.3/1.3) \text{ DataAMAT}$$
$$= 2.54$$

Improving Cache Performance

- Assuming cache is fast enough to keep up with the pipeline (hit time is fast enough), there are two ways to increase AMAT:
 1. Reduce miss rate
 2. Reduce miss time

Know This!

Calculate runtime given cache statistics (miss rate, miss penalty, etc.)

Calculate Average Memory Access Time (AMAT)

Understand direct mapped, set-associative, fully associative

- Comparison between them
- Sources of cache misses
- Architecture
- Addressing into them (tag, index, byte)
- Cache behavior with them

Block Size Real Stuff

- Intel Xeon has 512-bit (64-byte, 16-word) cache blocks
- Pi ARM has 256-bit (32-byte, 8-word) cache blocks

Calculating Cache Performance

Processor:

- $CPI = 2$
- Icache miss rate = 2%, miss penalty = 100 cycles
- Dcache miss rate = 4%, miss penalty = 100 cycles

Using SPECint2000 load/store percentage of 36%:

- Speedup from this processor to one that never missed?
- What if $CPI = 1$?
- What if we doubled clock rate of computer without changing memory speed ($CPI = 2$)?

Calculating Cache Performance

Speedup from this processor to one that never missed?

- Perfect memory system

What if $CPI = 1$? (penalties don't change though)

- Improve pipeline performance and memory (hits) both, but penalty of a miss doesn't change

What if we doubled clock rate of computer without changing memory speed ($CPI = 2$)?

- Just improve pipeline performance, not memory
- Ideally we'd have a 2x speedup, because we can run instructions twice as fast

Calculating Cache Performance, CPI=2

$$\frac{\text{CPU time with stalls}}{\text{CPU time, no stalls}} = \frac{I * \text{CPI}_{\text{stall}} * \text{clkcycle}}{I * \text{CPI}_{\text{perf}} * \text{clkcycle}}$$

$$\text{I miss CPI} = 2\% * 100\% * 100 = 2.00$$

$$\text{D miss CPI} = 4\% * 36\% * 100 = 1.44$$

$$\text{So memory stalls/instr} = 2.00 + 1.44 = 3.44$$

$$\text{CPI}_{\text{stall}} = 2 + 3.44; \text{CPI}_{\text{perf}} = 2$$

$$\text{CPI}_{\text{stall}} / \text{CPI}_{\text{perf}} = 5.44 / 2 = 2.72$$

Calculating Cache Performance, CPI=1

$$\frac{\text{CPU time with stalls}}{\text{CPU time, no stalls}} = \frac{I * \text{CPI}_{\text{stall}} * \text{clkcycle}}{I * \text{CPI}_{\text{perf}} * \text{clkcycle}}$$

$$\text{I miss CPI} = 2\% * 100 = 2.00$$

$$\text{D miss CPI} = 4\% * 36\% * 100 = 1.44$$

$$\text{So memory stalls/instr} = 2.00 + 1.44 = 3.44$$

$$\text{CPI}_{\text{stall}} = 1 + 3.44; \text{CPI}_{\text{perf}} = 1$$

$$\text{CPI}_{\text{stall}} / \text{CPI}_{\text{perf}} = 4.44 / 1 = 4.44$$

Calculating Cache Performance, 2x

$$\frac{\text{CPU time (slow clk)}}{\text{CPU time (fast clk)}} = \frac{I * \text{CPI}_{\text{slow}} * \text{clkcycle}}{I * \text{CPI}_{\text{fast}} * \text{clkcycle}/2}$$

$$I \text{ miss CPI} = 2\% * 200 = 4.00$$

$$D \text{ miss CPI} = 4\% * 36\% * 200 = 2.88$$

$$\text{So memory stalls/instr} = 4.00 + 2.88 = 6.88$$

$$\text{CPI}_{\text{fast}} = 2 + 6.88; \text{CPI}_{\text{slow}} = 5.44$$

$$\text{Speedup} = 5.44 / (8.88 * 0.5) = 1.23$$

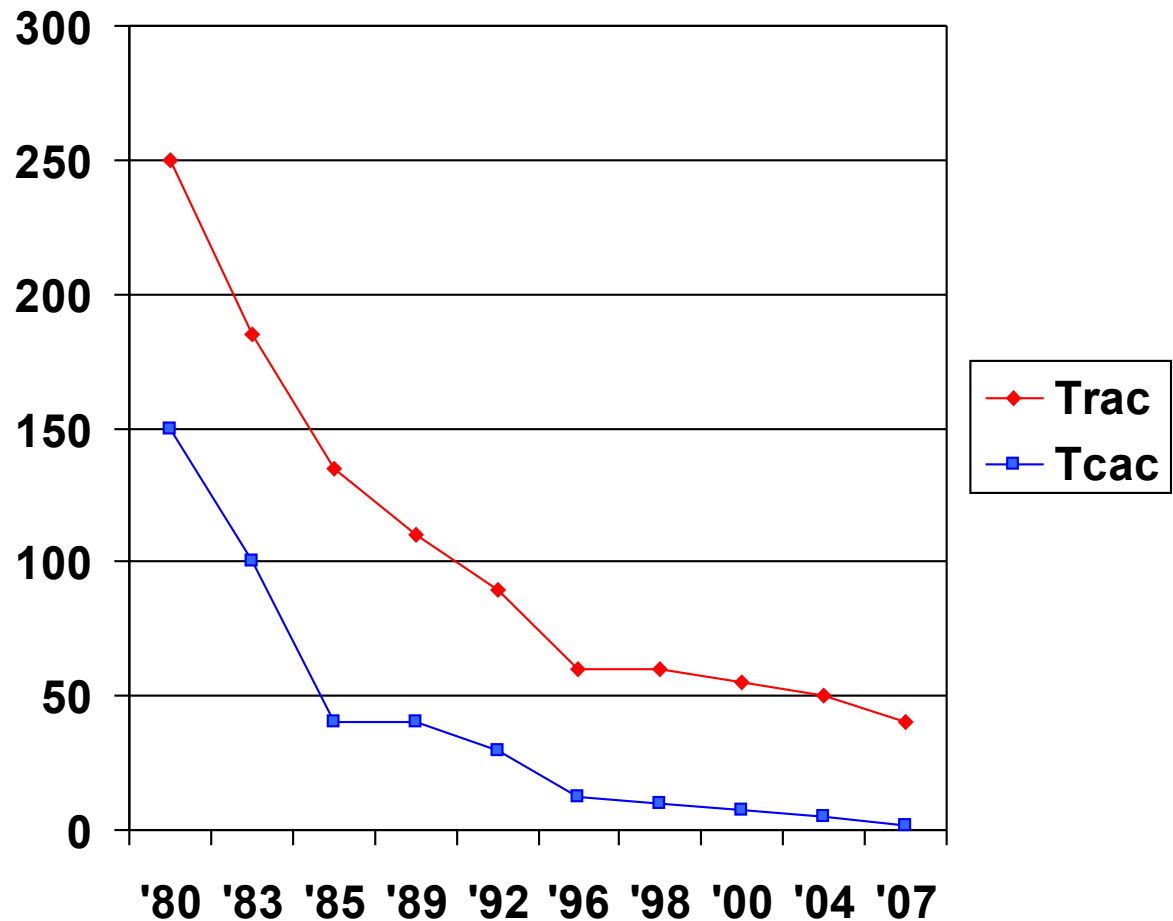
Ideal machine is 2x faster

Main Memory 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 = $1 + 4 \times 15 + 4 \times 1 = 65$ bus cycles
 - Bandwidth = $16 \text{ bytes} / 65 \text{ cycles} = 0.25 \text{ B/cycle}$

DRAM Generations

Year	Capacity	\$/GB
1980	64Kbit	\$1500000
1983	256Kbit	\$500000
1985	1Mbit	\$200000
1989	4Mbit	\$50000
1992	16Mbit	\$15000
1996	64Mbit	\$10000
1998	128Mbit	\$4000
2000	256Mbit	\$1000
2004	512Mbit	\$250
2007	1Gbit	\$50

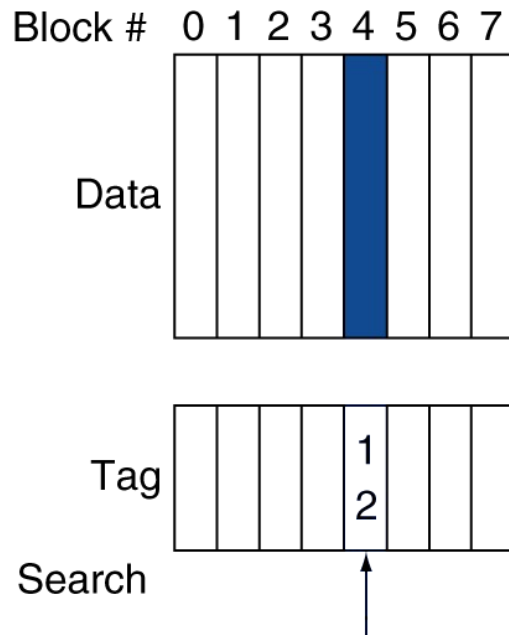


Associative Caches

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

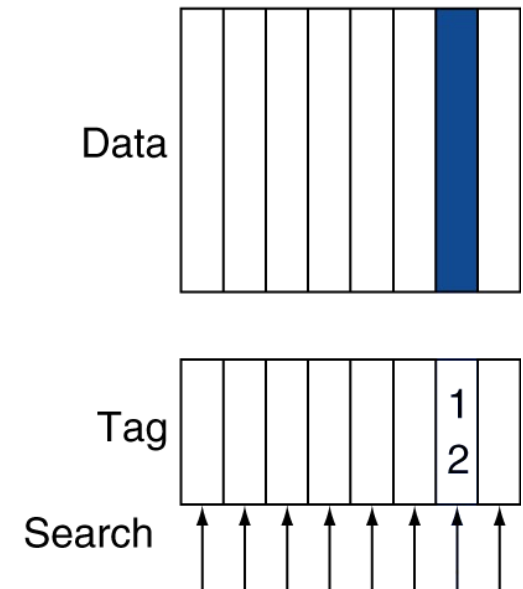
Direct mapped



Set associative



Fully associative



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, 2-way set associative, fully associative
 - Block access sequence: 0, 8, 0, 6, 8
- Direct mapped

Block address	Cache index	Hit/miss	Cache content after access	
			0	123
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]	91 Mem[6]

Associativity Example

- 2-way set associative

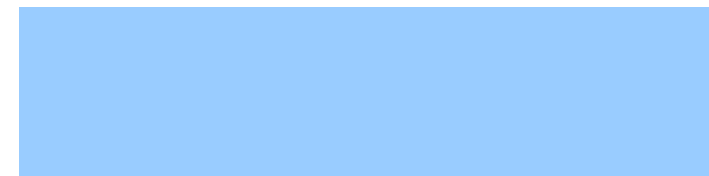
Block address	Cache index	Hit/miss	Cache content after access
			Set 0
0	0	miss	Mem[0]
8	0	miss	Mem[0]
0	0	hit	Mem[0]
6	0	miss	Mem[0]
8	0	miss	Mem[8]



Mem[8]
Mem[8]
Mem[8]
Mem[6]
Mem[6]

Fully associative

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



Mem[8]
Mem[8]
Mem[8]
Mem[8]
Mem[8]

Mem[6]

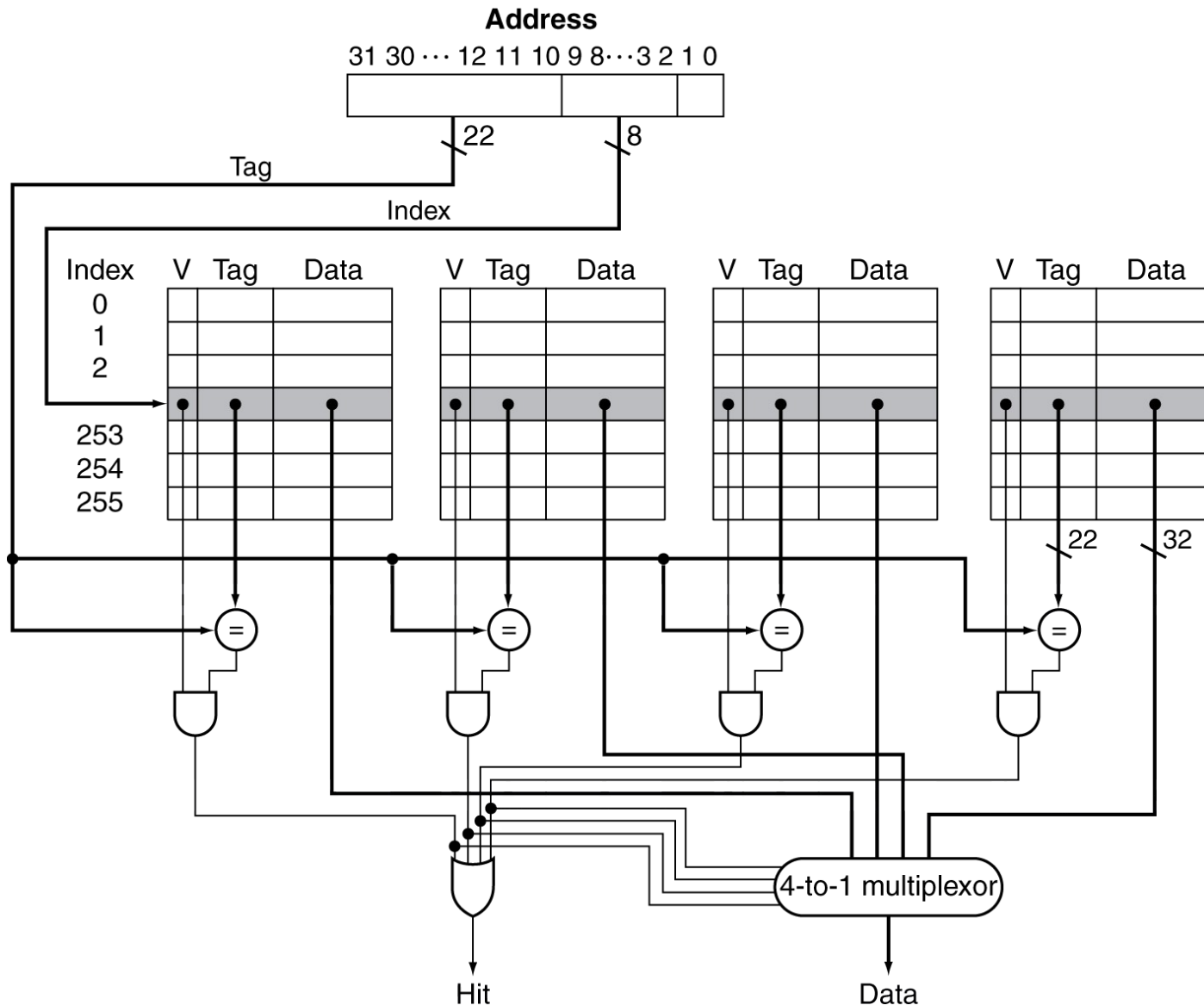
Mem[6]

92

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



Replacement Policy

- 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
- 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 = $100\text{ns} / 0.25\text{ns} = 400$ cycles
 - Effective CPI = $1 + 0.02 \times 400 = 9$

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 = $5\text{ns}/0.25\text{ns} = 20$ cycles
- Primary miss with L-2 miss
 - Extra penalty = 500 cycles
- $\text{CPI} = 1 + 0.02 \times 20 + 0.005 \times 400 = 3.4$
- 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 cache
 - L-1 block size smaller than L-2 block size