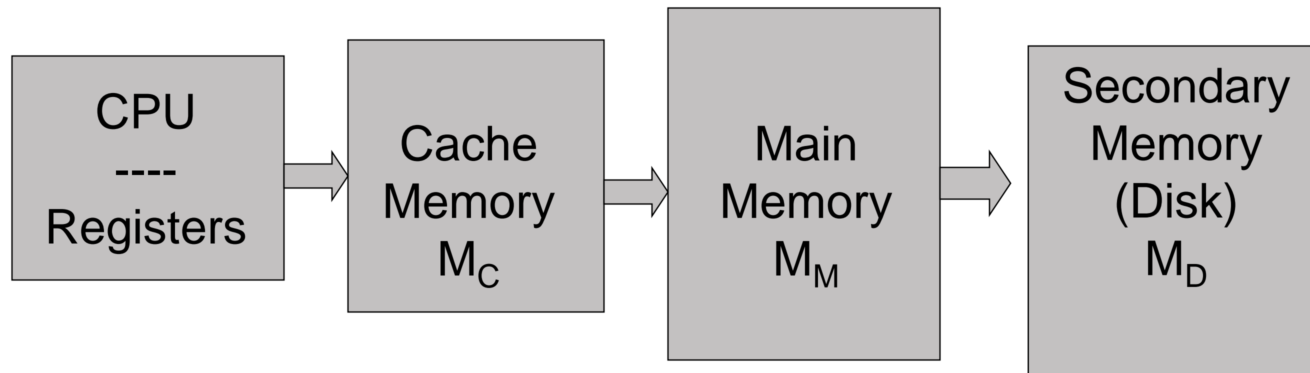
Cache Memory

Patterson & Hennessey

Chapter 5

Memory Hierarchy



Memory Content: $M_C \subseteq M_M \subseteq M_D$

<u>Memory Parameters:</u>

- Access Time: increase with distance from CPU
- Cost/Bit: decrease with distance from CPU
- Capacity: increase with distance from CPU

Cache Memory

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses $X_1, \ldots, X_{n-1}, X_n$

X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_3

a. Before the reference to X_n

X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_n
X_3

b. After the reference to X_n

- How do we know if the data is present in cache?
- Where do we look in the cache?
- Where do we put new data in the cache?

Hits vs. Misses

- Read hits
  - this is what we want!
- Read misses
  - stall the CPU, fetch block from memory, deliver to cache, restart
- Write hits:
  - can replace data in cache and memory (write-through)
  - write the data only into the cache (write-back the cache later)
- Write misses:
  - read the entire block into the cache, then write the word

Average Memory Access Time

- *Look-through cache*: main accessed after cache miss detected:

  T_C, T_M = cache and main memory access times

  H_C = cache hit ratio

  $$T_{A_{avg}} = T_C * H_C + (1-H_C)(T_C + T_{M_{avg}})$$

  $$= T_C + \underbrace{(1-H_C)(T_{M_{avg}})}_{\text{miss penalty}}$$

- *Look-aside cache*: main accessed concurrent with cache access
  - abort main access on cache hit
  - main access already in progress on cache miss
  - Wasted main bus cycles on cache hit (problem if memory shared)

  $$T_{Aavg} = T_C * H_C + (1-H_C)(T_M)$$

Average Memory Access Time

- Extending to 3rd level (disk):

$$T_{A_{avg}} = T_C + (1\text{-}H_C)(T_M) + (1\text{-}H_C)(1\text{-}H_M)(T_{D_{avg}})$$

Note that $T_M << T_{D_{avg}}$

Fully Associative Cache

- Associative memory – access by "content" rather than address

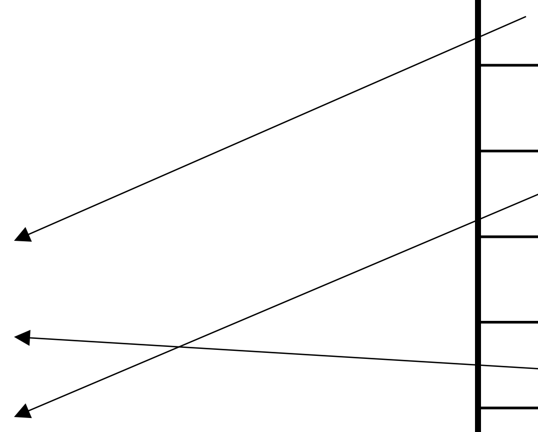
Concurrently compare CPU address
to all cache address fields

Valid entry

Address from CPU

V	Address	Data
1	3 =	D3
1	7 =	D7
1	5 =	D5
0	? =	?

Cache Memory

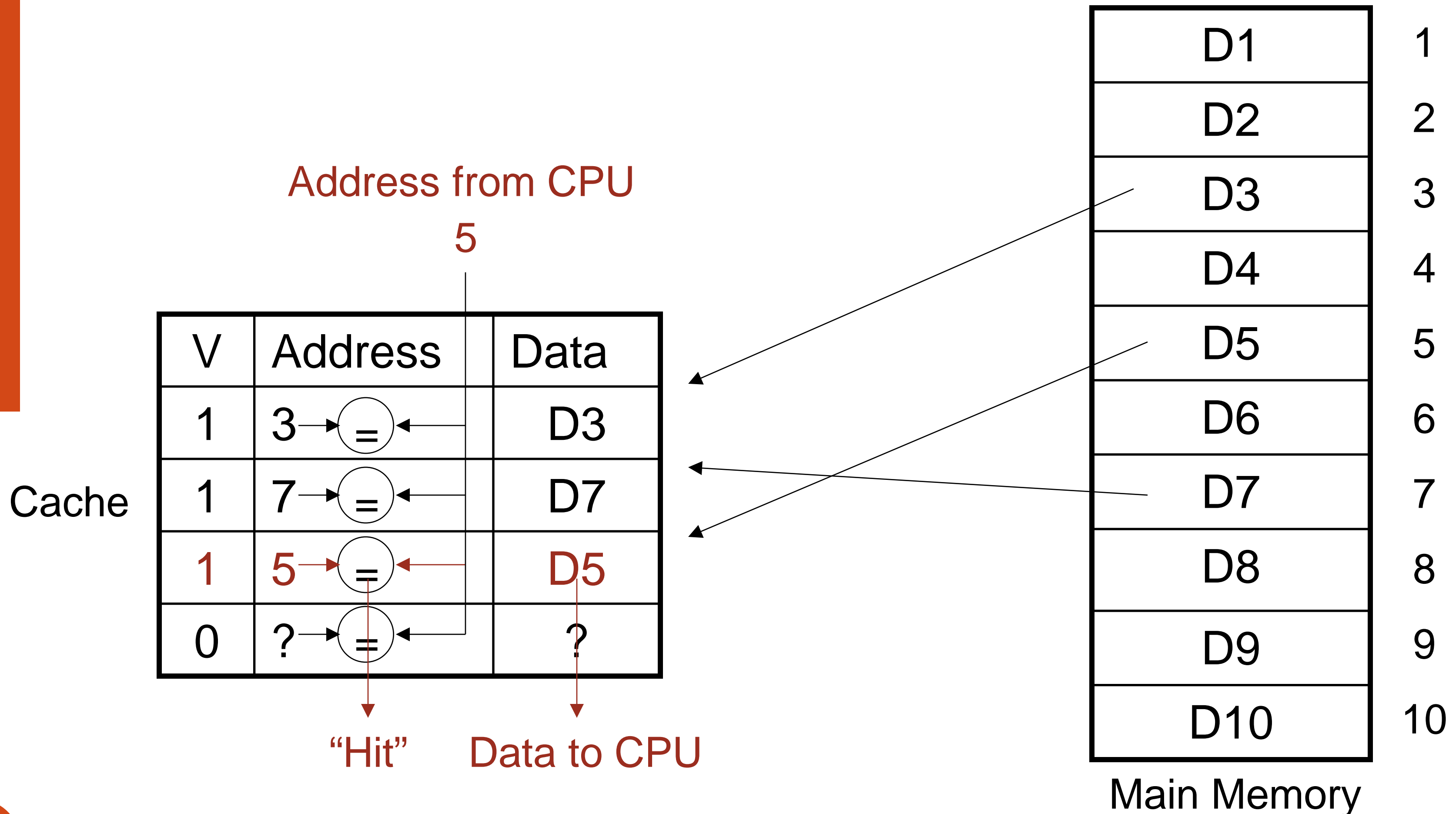


D1	1
D2	2
D3	3
D4	4
D5	5
D6	6
D7	7
D8	8
D9	9
D10	10

Main Memory

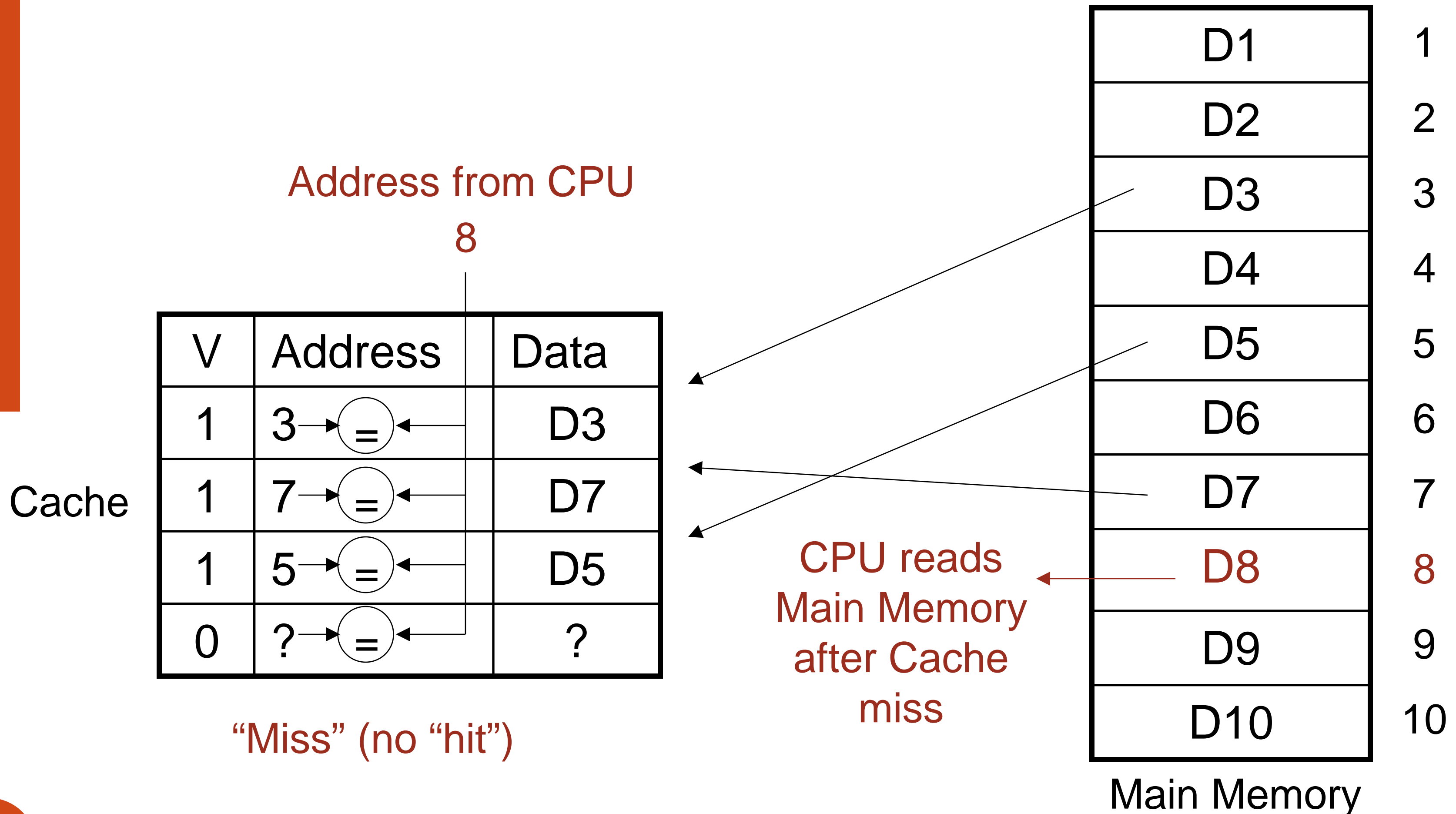
Fully Associative Cache

- CPU memory read from address 5



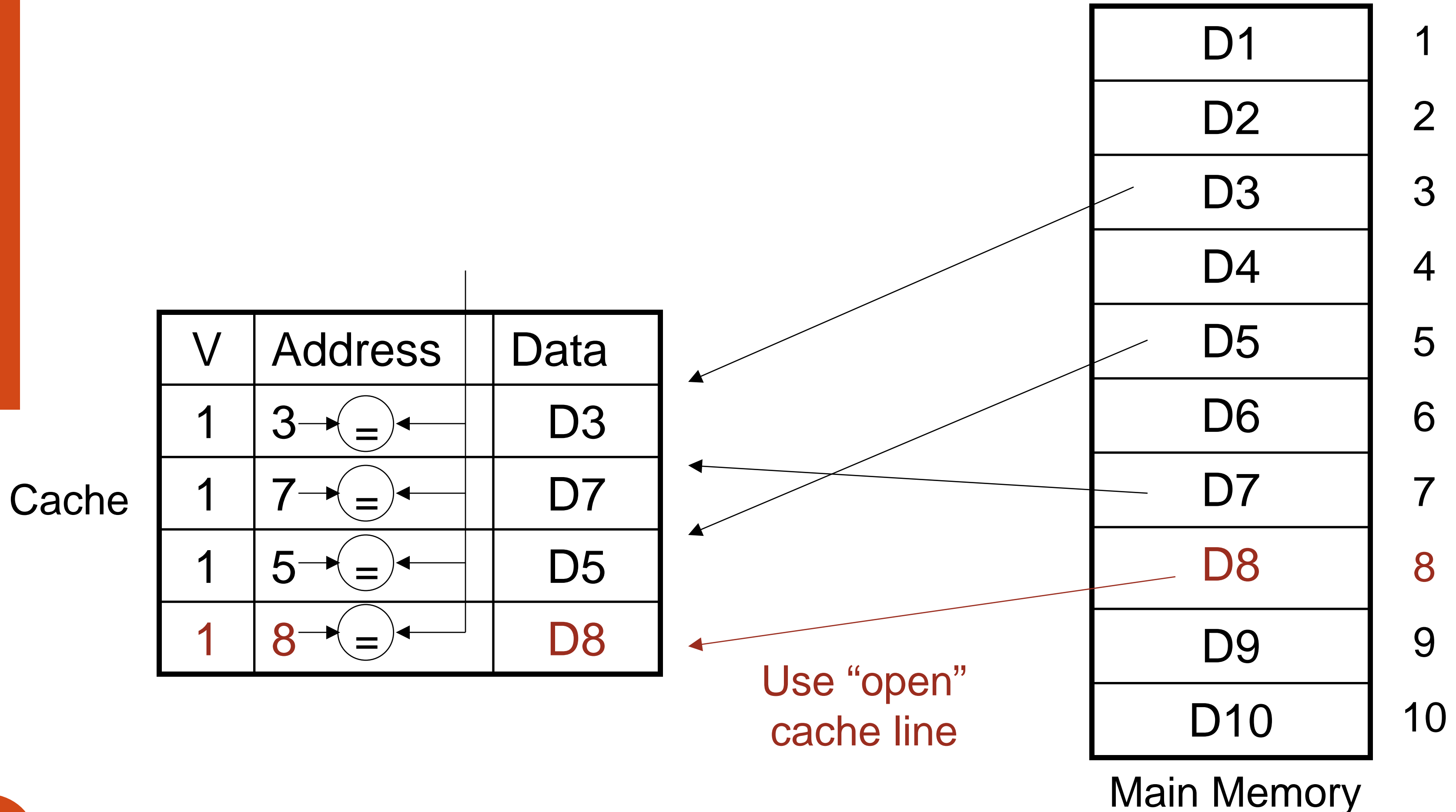
Fully Associative Cache

- CPU memory read from address 8



Cache Memory Structures

- Update cache with data from address 8

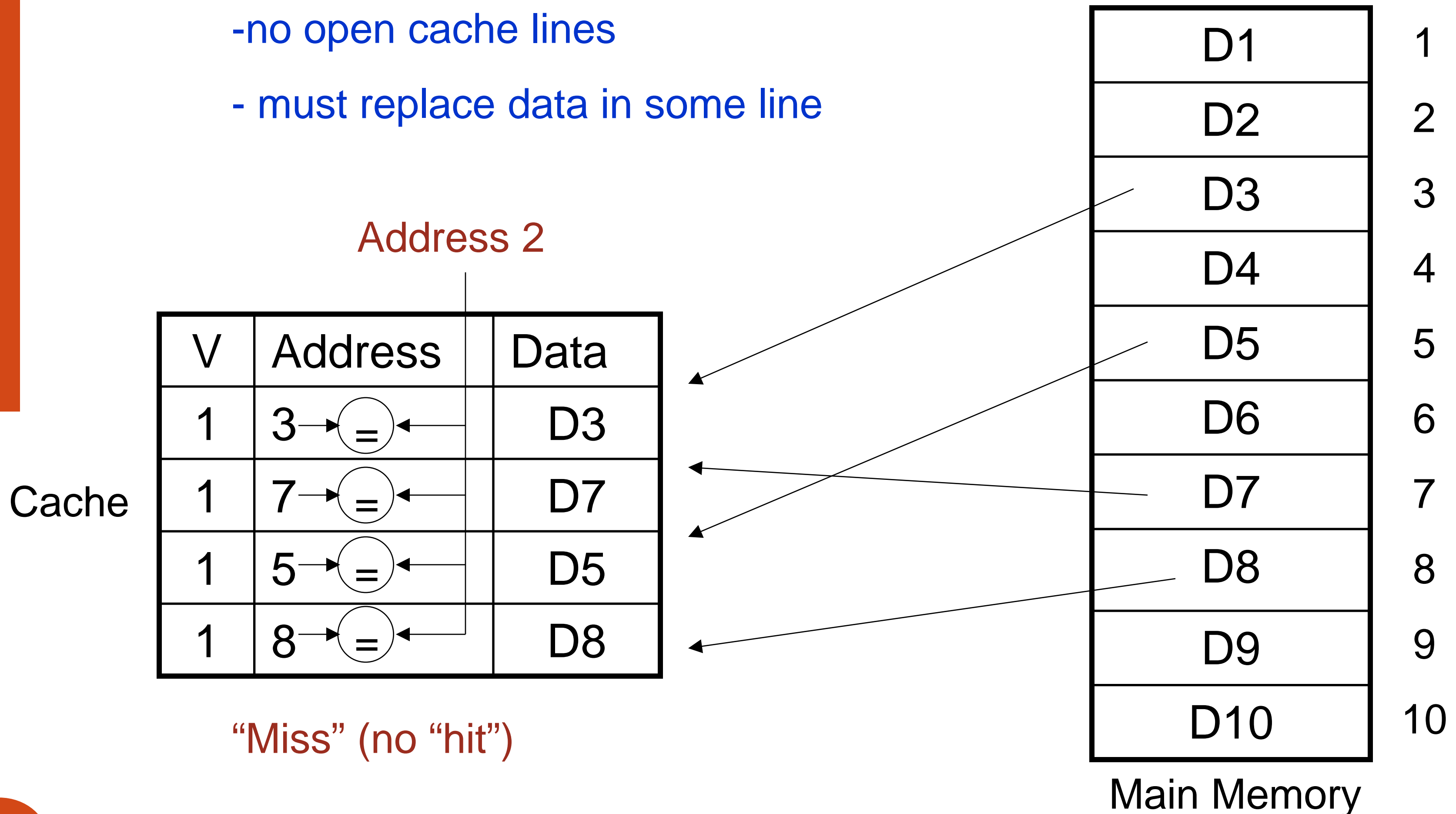


Cache Memory Structures

- Where should the CPU place data from address 2??

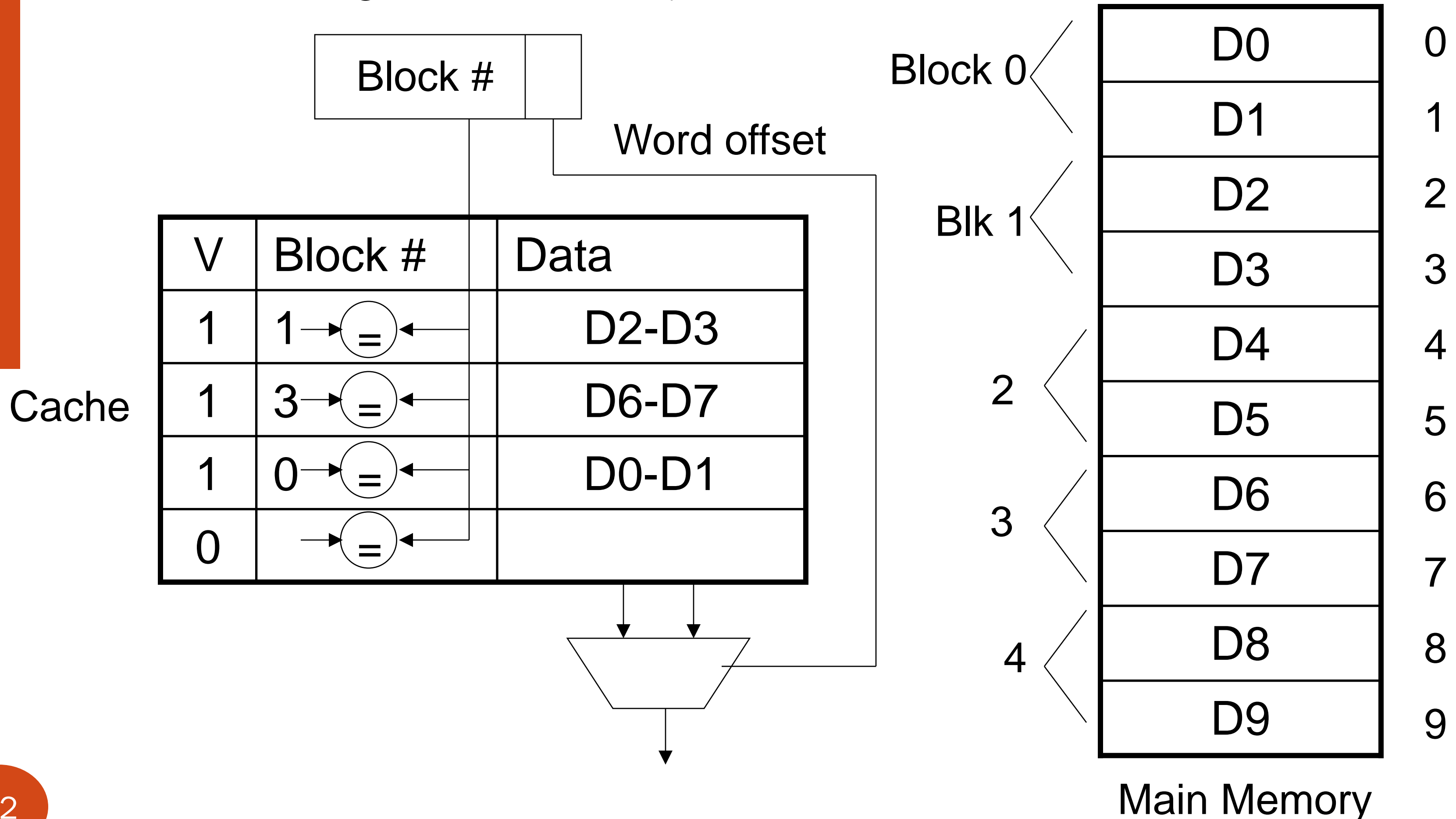
    -no open cache lines

    - must replace data in some line



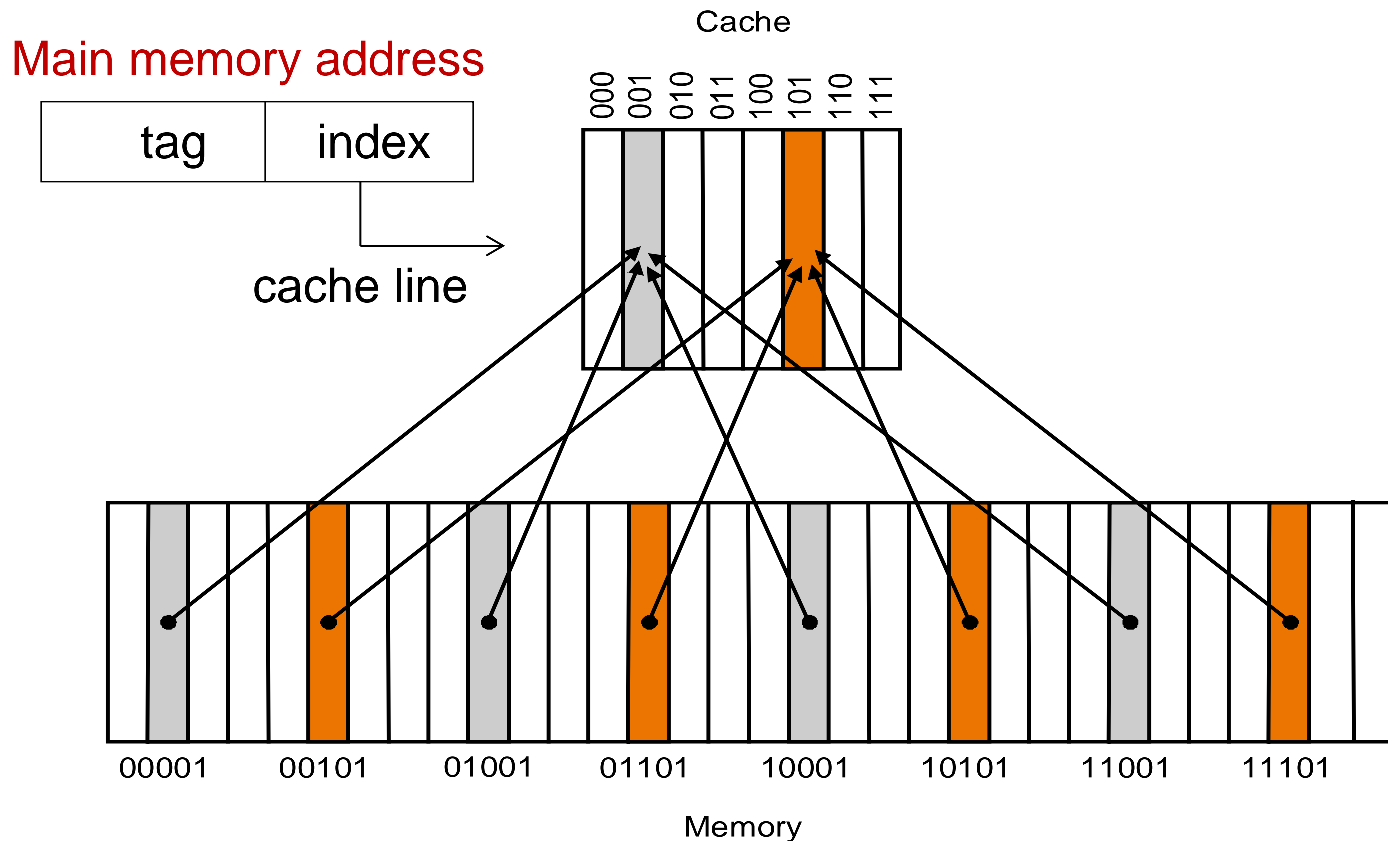
Multi-Word Blocks

- Cache line holds a multi-word block from main memory
- Take advantage of spatial locality of reference



Direct Mapped Cache

- Location is determined by the main memory address
- Mapping: address is modulo the #blocks in the cache



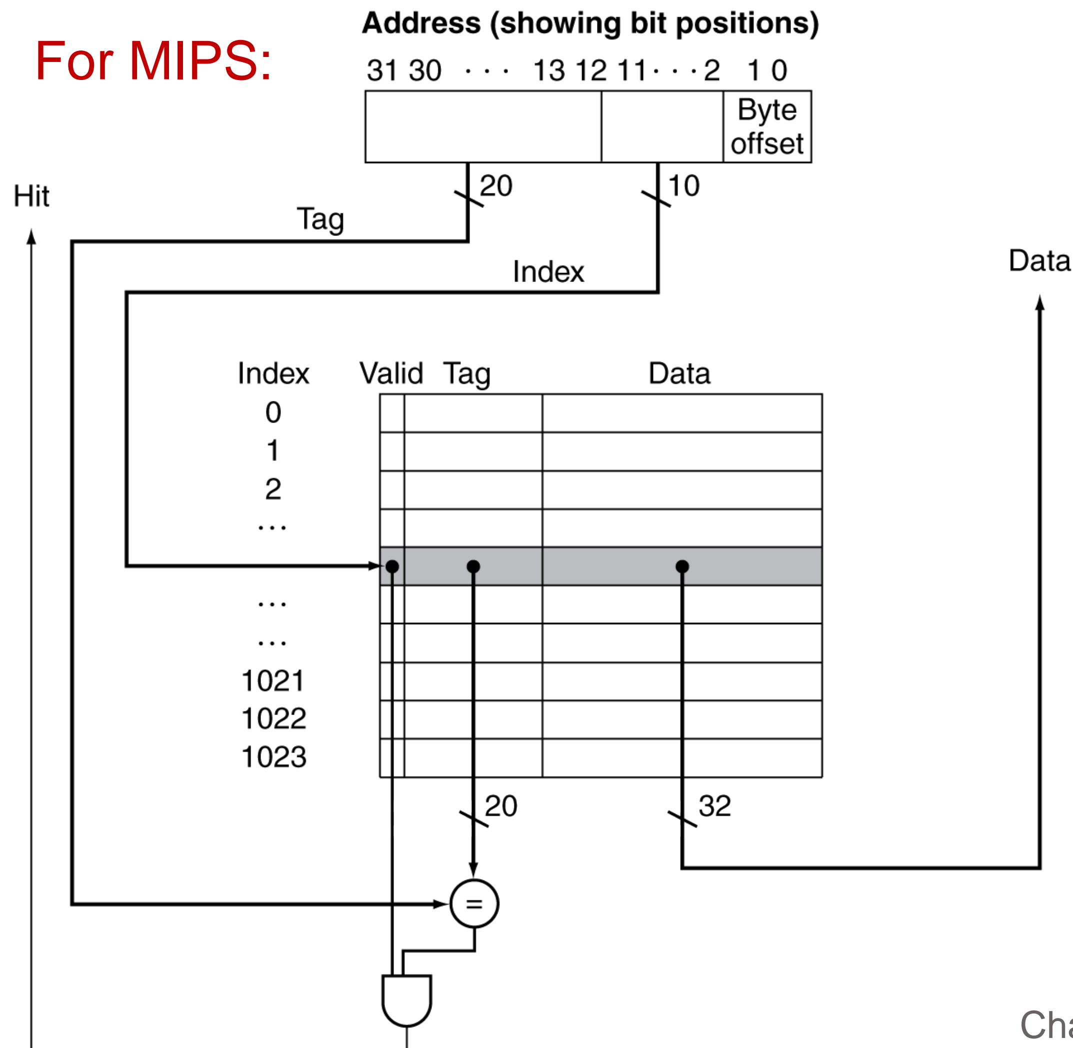
Tags and Valid Bits

- 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
  - Reset all valid bits to 0 if main memory changed ("invalidate the cache")

Direct Mapped Cache

Address Subdivision

For MIPS:



What kind of locality are we taking advantage of?

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		

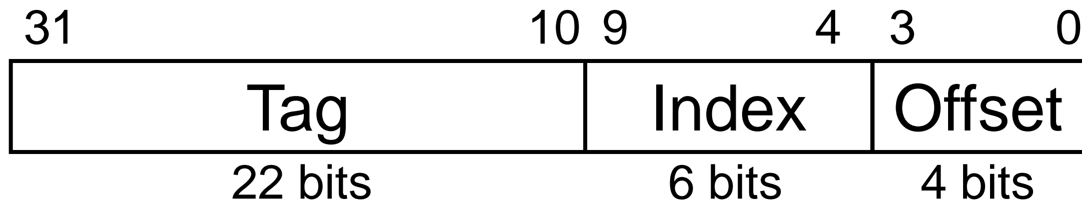
Replaces Mem[110010

Block Size Considerations

- Larger blocks can 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
- Larger miss penalty to load larger block
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help

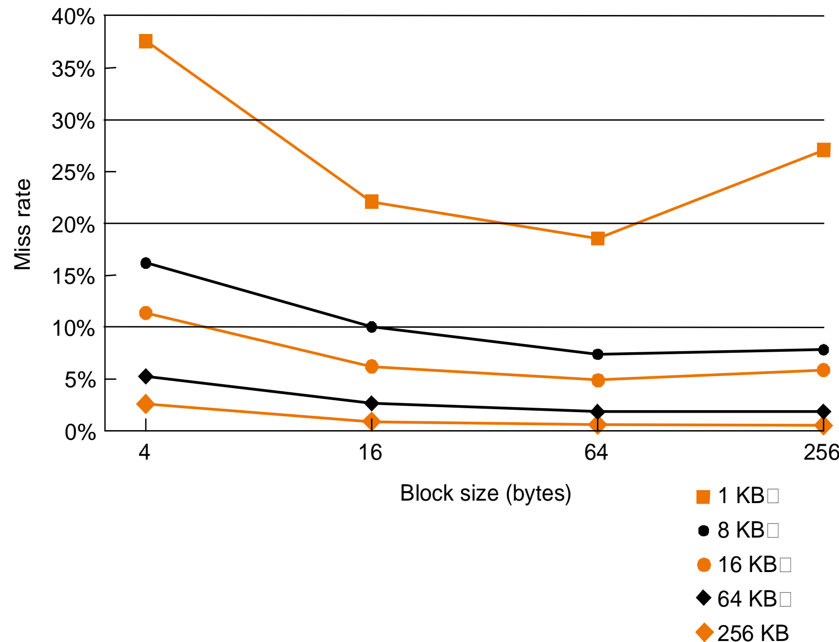
Example: Larger Block Size

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



Performance

- Increasing the block size tends to decrease miss rate:



- Use split caches because there is more spatial locality in code:

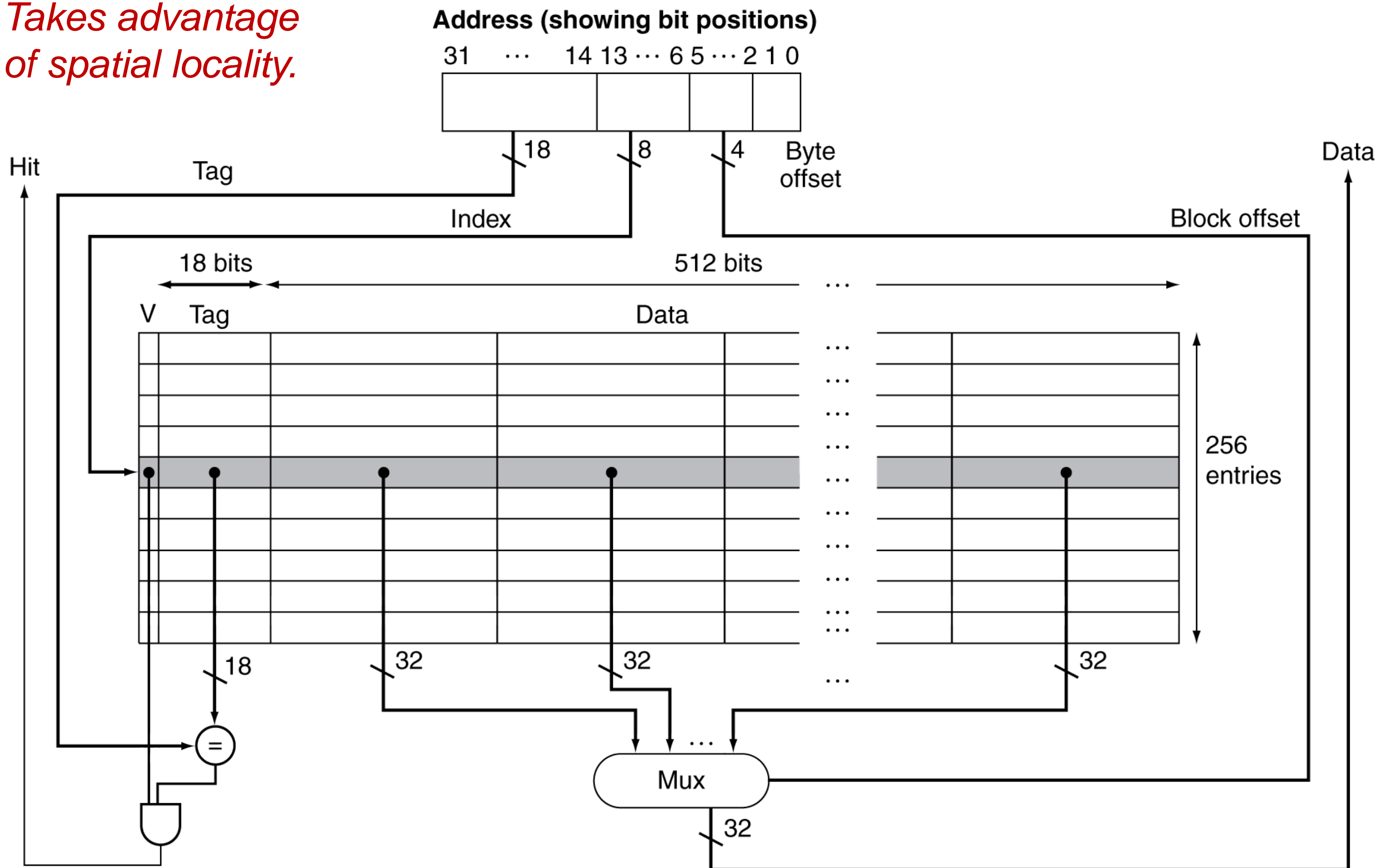
Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

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/block
  - D-cache: write-through or write-back
- SPEC2000 miss rates
  - I-cache: 0.4%
  - D-cache: 11.4%
  - Weighted average: 3.2%

Example: Intrinsity FastMATH

Takes advantage of spatial locality.



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

Cache Misses

- Compulsory – cache "empty" at startup
- Capacity – cache unable to hold entire working set
- Conflict – two memory loc's map to same cache line

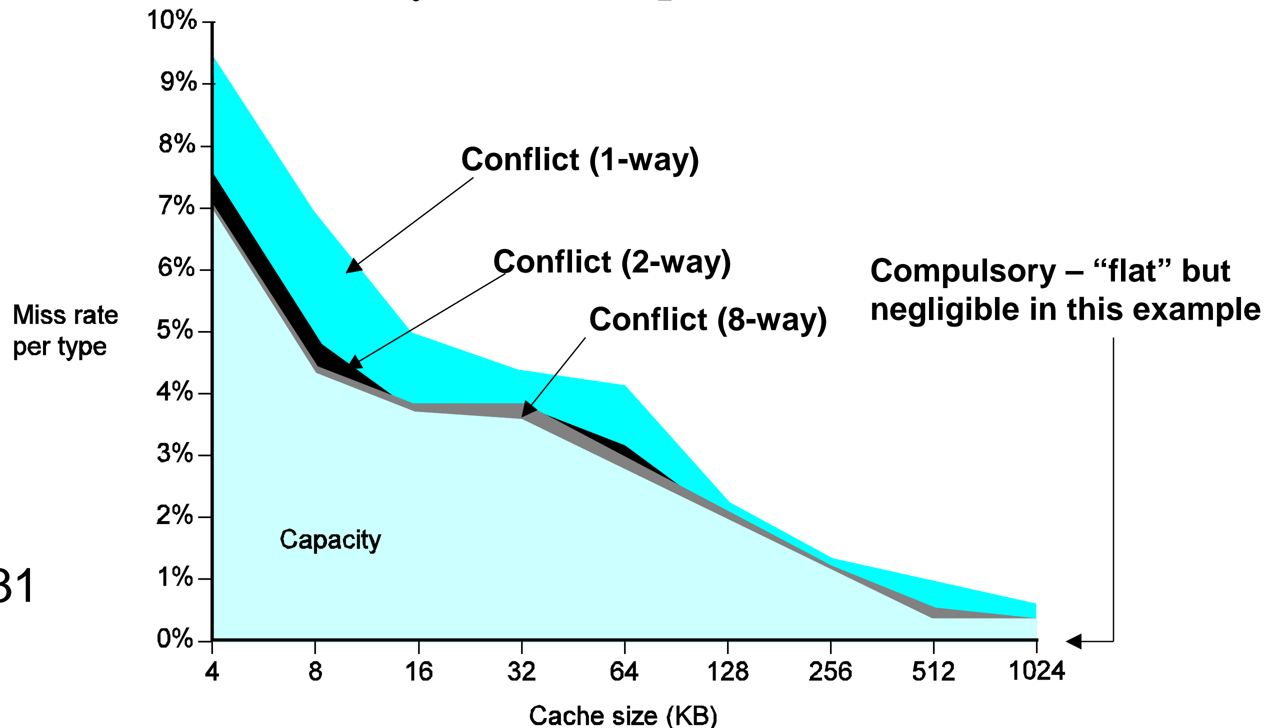


Fig. 7.31

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

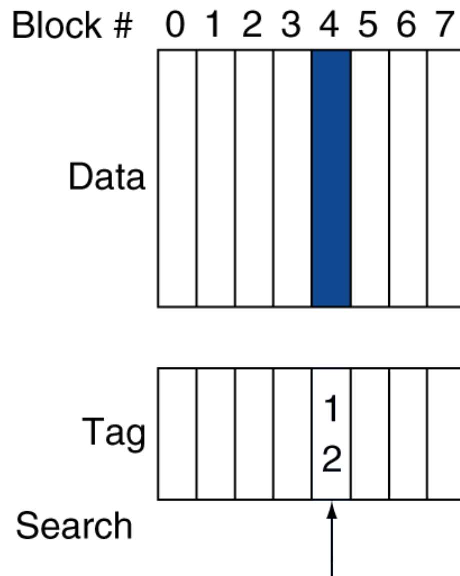
Replacement Policy

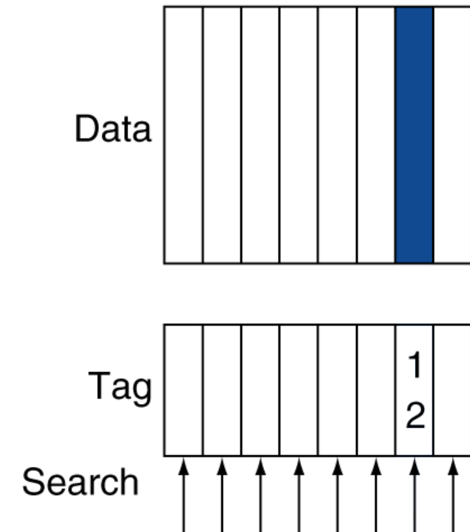
- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity

Decreasing miss ratio with associativity

- Problems:
  - Fully associative – high cost, but misses only when capacity reached since any item can go in any line
  - Direct-mapped: less expensive, but can hold only one item with a given index, leading to conflicts
- Compromise: Set Associative Cache
  - "N-way" set associative cache has N direct-mapped caches
  - One "set" = N lines with a particular index
  - Item with index K can be placed in line K of any of the N direct-mapped caches
  - Results in fewer misses due to conflicts

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, 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	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]	

Continued on next slide

Associativity Example

- 2-way set associative (sequence: 0, 8, 0, 6, 8)

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]		

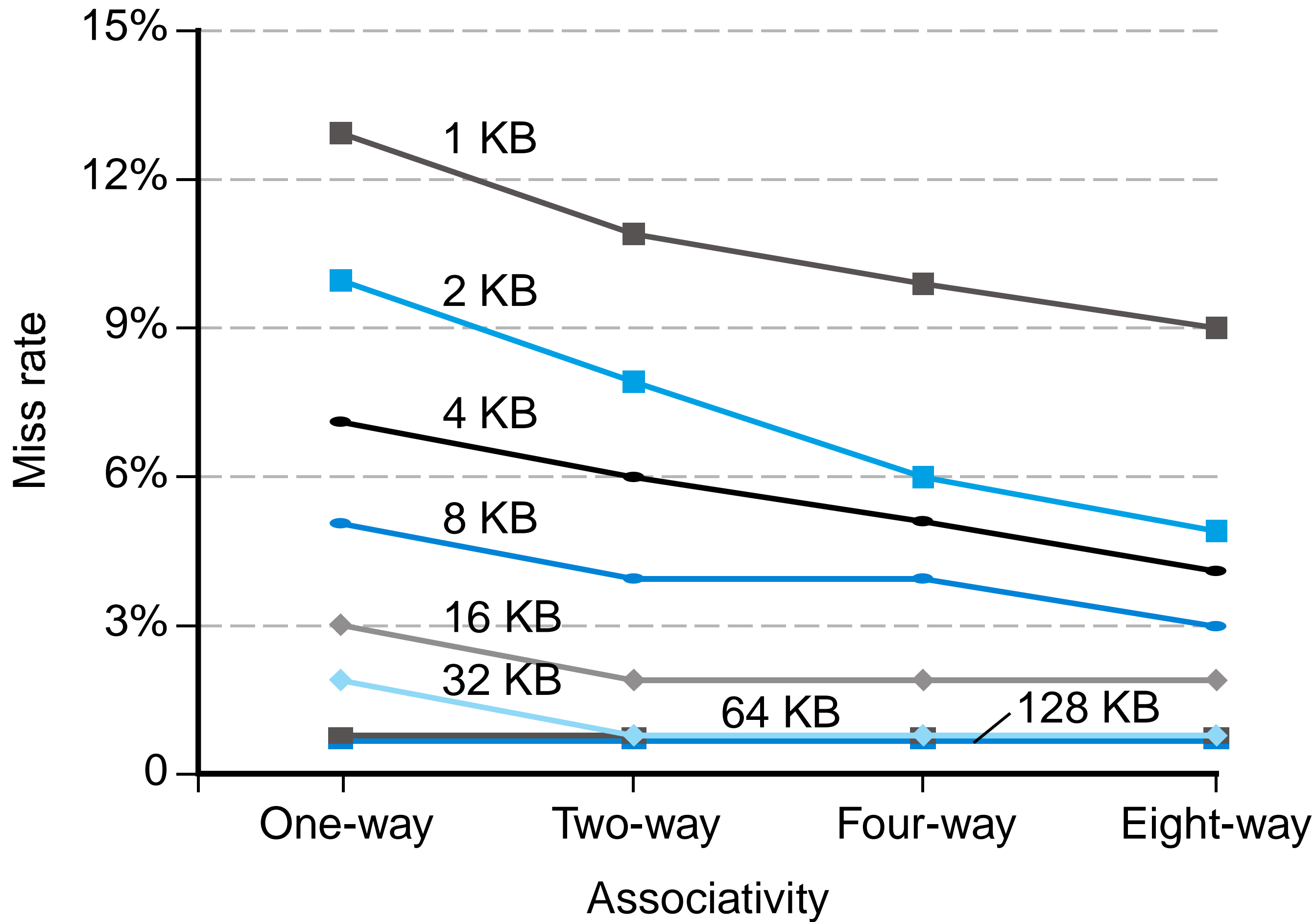
- Fully associative (sequence: 0, 8, 0, 6, 8)

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]	

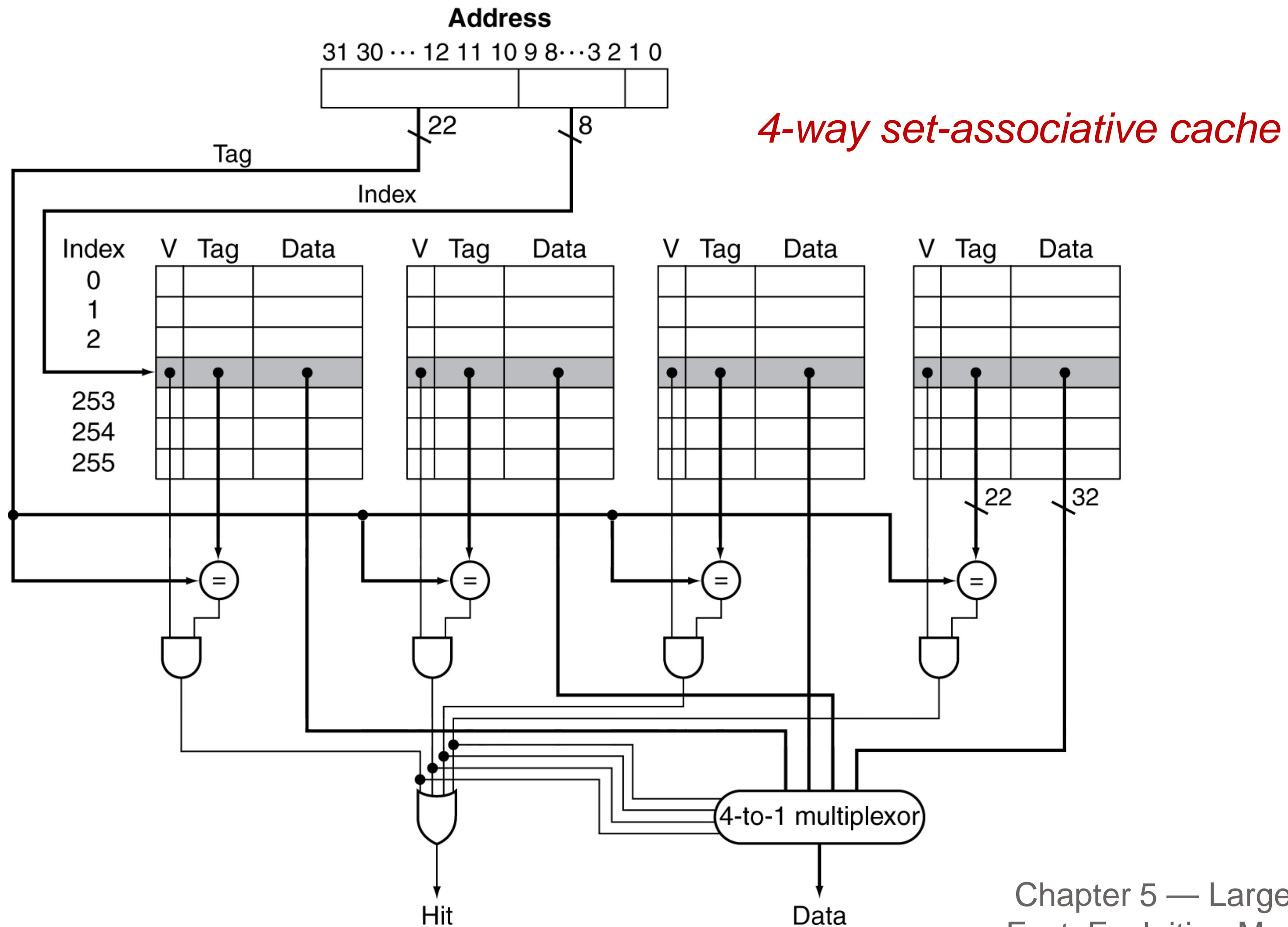
How Much Associativity

- Increased associativity decreases miss rate
  - But 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%

Performance vs. associativity

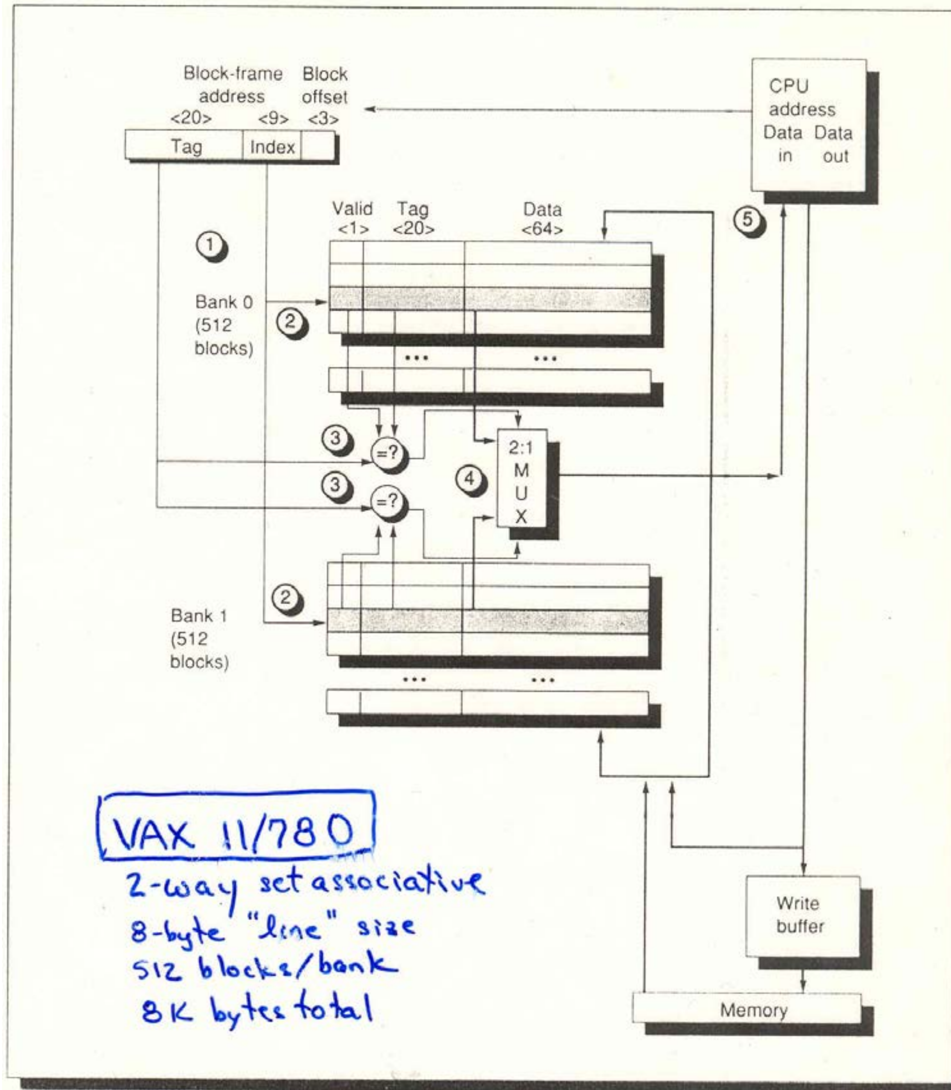


Set Associative Cache Organization



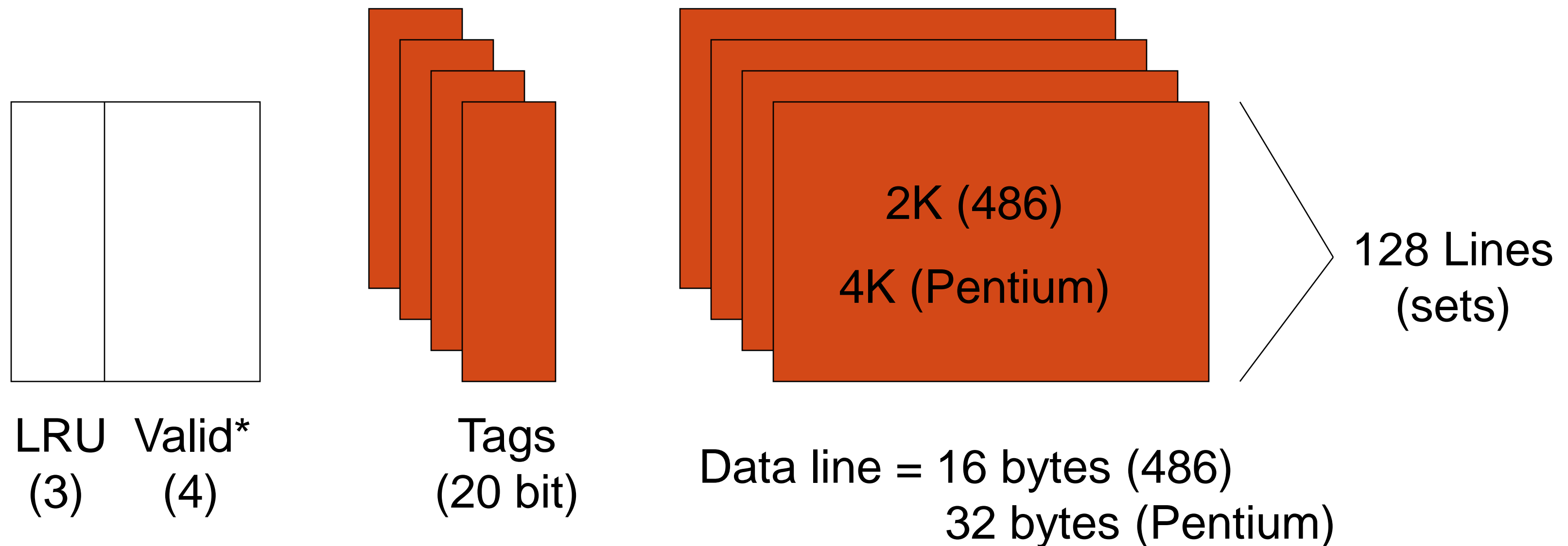
4-way set-associative cache

VAX 11/780 Cache



Intel 80486/Pentium L1 Cache

80486 4-way Set Associative (Pentium 2-way)



* Instruction cache: 1 valid bit per line

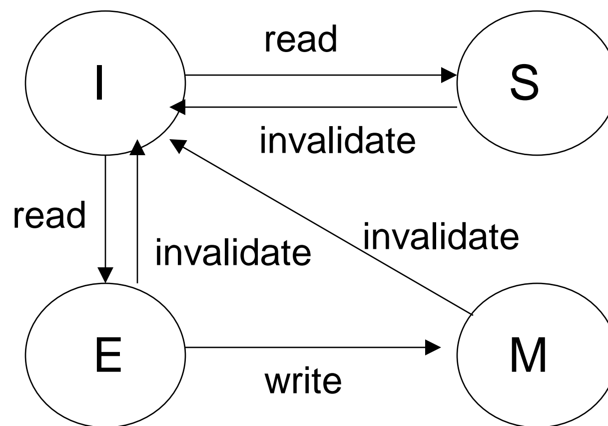
* Data cache: 2-bit MESI state per line

Write-back/Write-through programmable

80486 – 8K unified I/D
Pentium – 8K/8K I/D
Pentium II/III – 16K/16K I/D

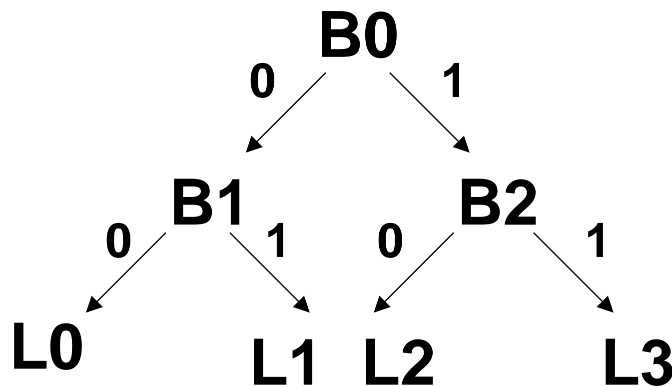
MESI Cache Line State

- MESI = 2-bit cache line "state" for "write-back" cache
  - I = Invalid (line does not contain valid data)
  - S = Line valid, with shared access – no writes allowed
  - E = Line valid, with exclusive access – data may be written
  - M = Line valid & modified since read from main (must be rewritten to main memory)



Intel Approximated-LRU Replacement

- 3-bit number B2B1B0 assigned to each "set" of 4 lines
  - Access L0/L1 – set B0=1
  - Access L2/L3 – set B0=0
  - Access L0 – set B1=1 or L1 – set B1=0
  - Access L2 – set B2=1 or L1 – set B2=0



B0B1 = 00 – replace L0

01 – replace L1

B0B2 = 10 – replace L2

11 – replace L3

Measuring Cache Performance

- Components of CPU time:
  - Program execution cycles
    - Includes cache hit time
    - Execution time = (execution cycles + stall cycles) x cycle 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 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 = 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

Example – “gcc”compiler on MIPS

- Instruction count = IC
- Instruction cache miss rate = 5%
- Data cache miss rate = 10%
- Clocks/instruction (CPI) = 4 if no misses/stalls
- Miss penalty = 12 cycles (all misses)
- Instruction frequencies:
  - lw = 22% of instructions executed
  - sw = 11% of instructions executed

What is the effect of cache misses on CPU performance (ex, on CPI)?

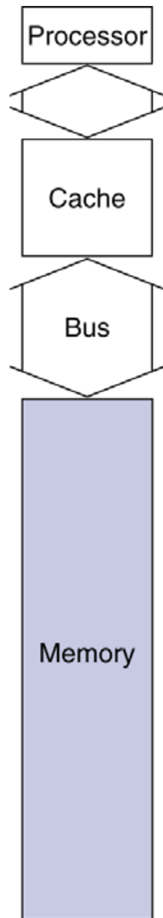
Performance Summary

- Two ways of improving performance:
  - decrease the miss ratio
  - decrease the miss penalty
- 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

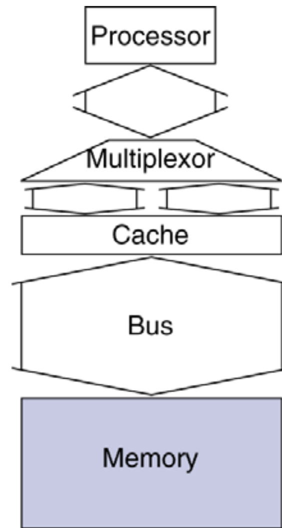
Main Memory Supporting Caches

- 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 bytes / 65 cycles = 0.25 B/cycle

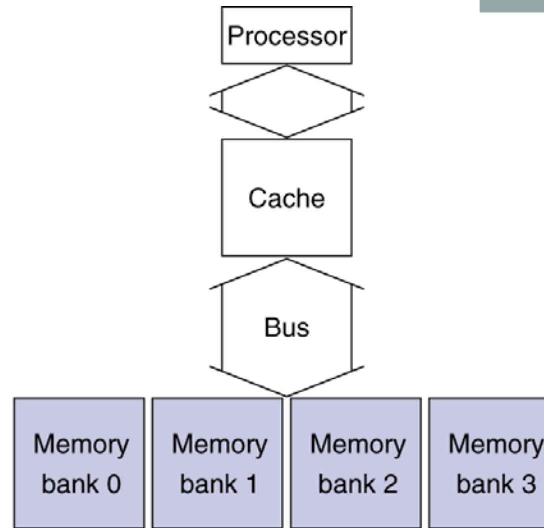
Increasing Memory Bandwidth



a. One-word-wide memory organization



b. Wider memory organization



c. Interleaved memory organization

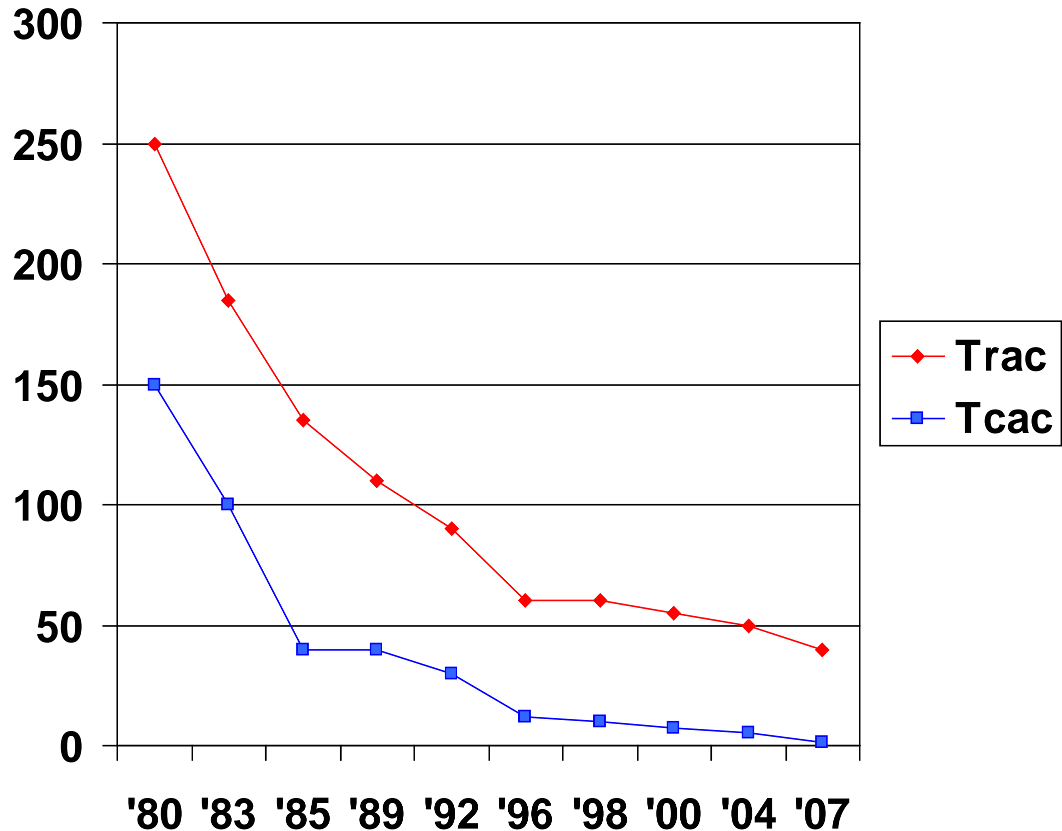
- 4-word wide memory
  - Miss penalty = 1 + 15 + 1 = 17 bus cycles
  - Bandwidth = 16 bytes / 17 cycles = 0.94 B/cycle
- 4-bank interleaved memory
  - Miss penalty = 1 + 15 + 4×1 = 20 bus cycles
  - Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle

Advanced DRAM Organization

- Bits in a DRAM are organized as a rectangular array
  - DRAM accesses an entire row
  - Burst mode: supply successive words from a row with reduced latency
- Double data rate (DDR) DRAM
  - Transfer on rising and falling clock edges
- Quad data rate (QDR) DRAM
  - Separate DDR inputs and outputs

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



Multilevel Caches

- Primary (Level-1) cache attached to CPU
  - Small, but fast
  - Try to optimize hit ratio
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
  - Optimize penalty for L-1 cache miss
- 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 = $1 + 0.02 \times 400 = 9$

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 = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 500 cycles
- 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

Intel Pentium P4 vs. AMD Opteron

Characteristic	Intel Pentium P4	AMD Opteron
L1 cache org.	Split instr/data cache	Split instr/data cache
L1 cache size	8KB data, 96KB trace cache (12K RISC instr)	64KB each I/D
L1 cache associativity	4-way set-associative	2-way set associative
L1 replacement	Approximated LRU	LRU
L1 block size	64 bytes	64 bytes
L1 write policy	Write-through	Write-back
L2 cache org.	Unified I/D	Unified I/D
L2 cache size	512KB	1024KB
L2 cache associativity	8-way set-associative	16-way set-associative
L2 replacement	Approximated LRU	Approximated LRU
L2 block size	128 bytes	64 bytes
L2 write policy	Write-back	Write-back

Figure 7.35

Interactions with Advanced CPUs

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