Cache Optimization

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

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 Policies (1)

- **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 x 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 Policies (2)

- **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

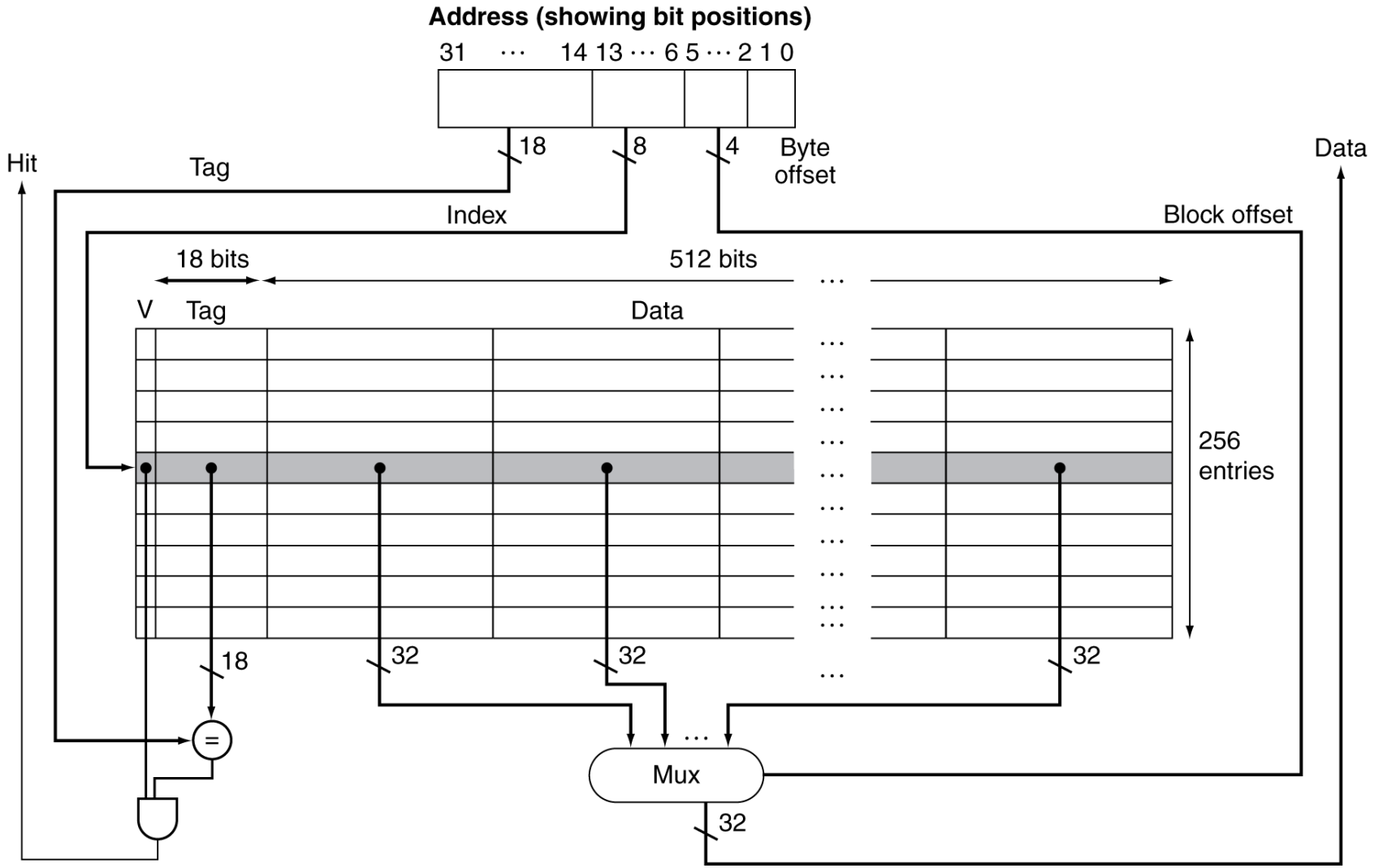
Write Policies (3)

- **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

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 x 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



Cache Performance (1)

- **Measuring cache performance**
  - Components of CPU time
    - Program execution cycles include cache hit time
    - Memory stall cycles: mainly from cache misses
  - With simplifying assumptions:

$$Memory\ stall\ cycles$$

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

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

Cache Performance (2)

- **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 x 100 = 2
  - D-cache: 0.36 x 0.04 x 100 = 1.44
- **Actual CPU = 2 + 2 + 1.44 = 5.44**
  - Ideal CPU is 5.44/2 – 2.72 times faster

Cache Performance (3)

- **Average access time**
  - Hit time is also important for performance
  - Average memory access time (AMAT)

$$AMAT = Hit\ time + Miss\ rate \times Miss\ penalty$$

  - Example
    - CPU with 1ns clock
    - Hit time = 1 cycle
    - Miss penalty = 20 cycles
    - I-cache miss rate = 5%
    - AMAT = 1 + 0.05 x 20 = 2ns
      - » 2 cycles per instruction

Cache Performance (4)

- **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

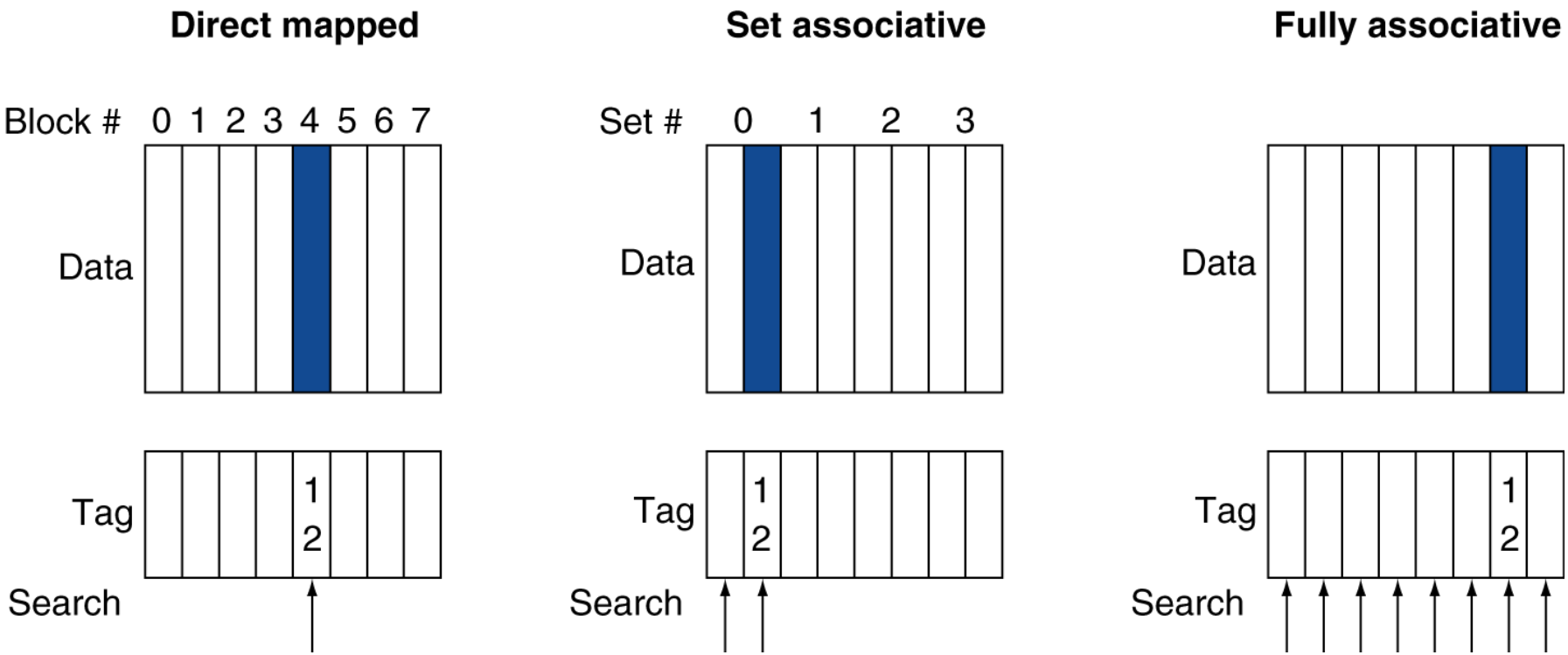
Associative Caches (1)

- **Fully associative**
  - Allows 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 Caches (2)

- **Example**



Associative Caches (3)

- **Spectrum of associativity: for an 8-entry cache**

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

Associative Caches (4)

- **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]	

Associative Caches (5)

- **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**

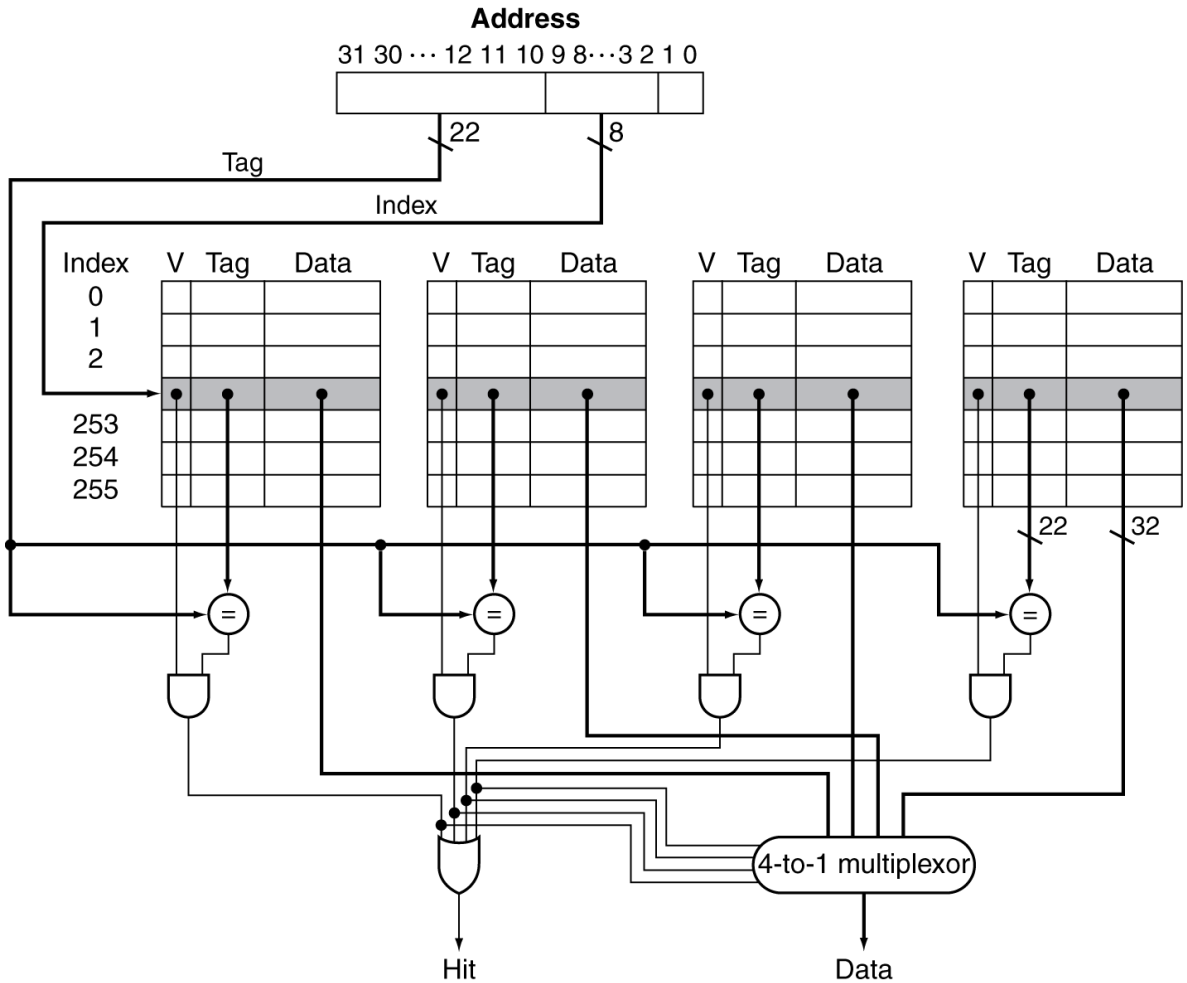
Block address		Hit/miss	Cache content after access			
0		miss	**Mem[0]**			
8		miss	Mem[0]	**Mem[8]**		
0		hit	**Mem[0]**	Mem[8]		
6		miss	Mem[0]	Mem[8]	**Mem[6]**	
8		hit	Mem[0]	**Mem[8]**	Mem[6]	

Associative Caches (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%

Associative Caches (7)

- **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 (1)

- **Multilevel caches**
  - Primary cache attached to CPU
    - Small, but fast
  - Level-2 (L2) cache services misses from primary cache
    - Larger, slower, but still faster than main memory
  - Main memory services L2 cache misses
  - Some high-end systems include L3 cache

Multilevel Caches (2)

- **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 x 400 = 9

Multilevel Caches (3)

- **Now add L2 cache**
  - Assumptions
    - Access time = 5ns
    - Global miss rate to main memory = 0.5%
  - Primary miss with L2 hit
    - Penalty = 5ns/0.25ns = 20 cycles
  - Primary miss with L2 miss
    - Extra penalty = 400 cycles
  - CPI = 1 + 0.02 x 20 + 0.005 x 400 = 3.4
  - Performance ratio = 9 / 3.4 = 2.6

Multilevel Caches (4)

- **Considerations**
  - Primary cache
    - Focus on minimal hit time
  - L2 cache
    - Focus on low miss rate to avoid main memory access
    - Hit time has less overall impact
  - Results
    - L1 cache usually smaller than a single-level cache
    - L1 block size may be smaller than L2 block size
    - L2 cache is much larger than in a single-level cache
    - L2 cache uses higher associativity for reducing miss rates

Interactions w/ 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

Interactions w/ Software

- **Misses depend on memory access patterns**
  - Algorithm behavior
  - Compiler optimization for memory access

- **Standard algorithmic analysis often ignores the impact of the memory hierarchy**