Cache Memory

Rabi Mahapatra & Hank Walker

Adapted from lectures notes of Dr. Patterson and Dr. Kubiatowicz of UC Berkeley

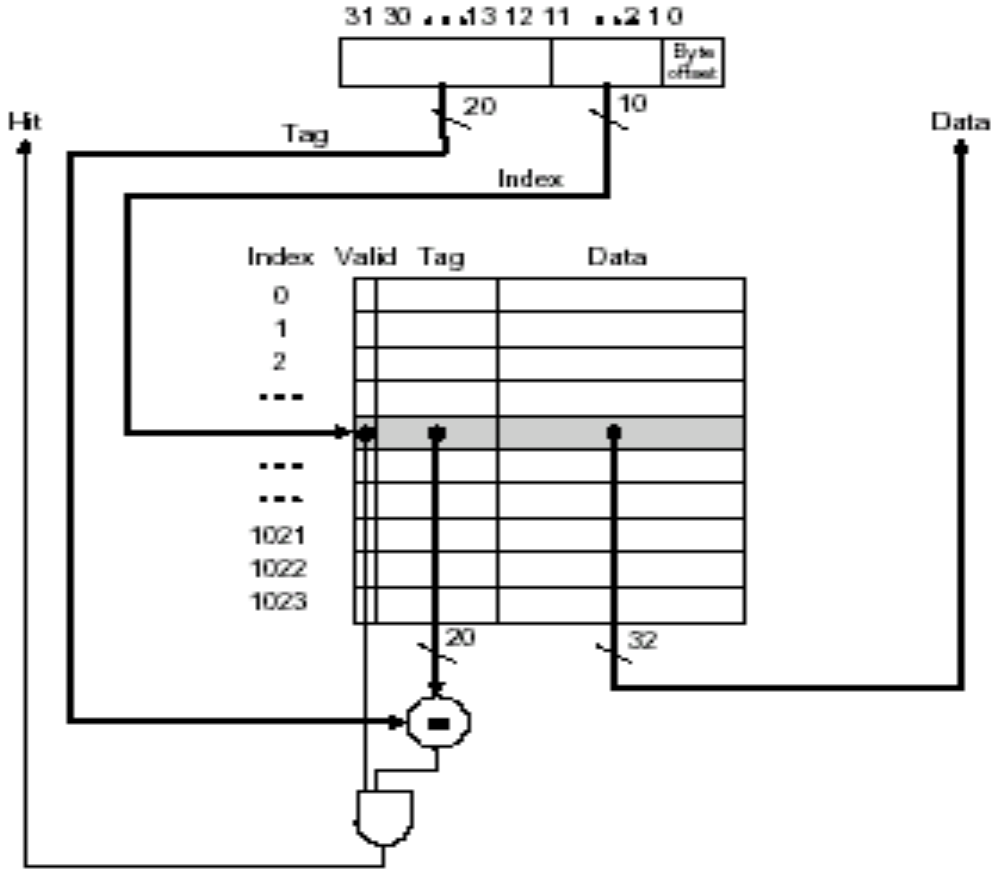
Revisiting Memory Hierarchy

- Facts
  - Big is slow
  - Fast is small
- Increase performance by having “hierarchy” of memory subsystems
- “Temporal Locality” and “Spatial Locality” are big ideas

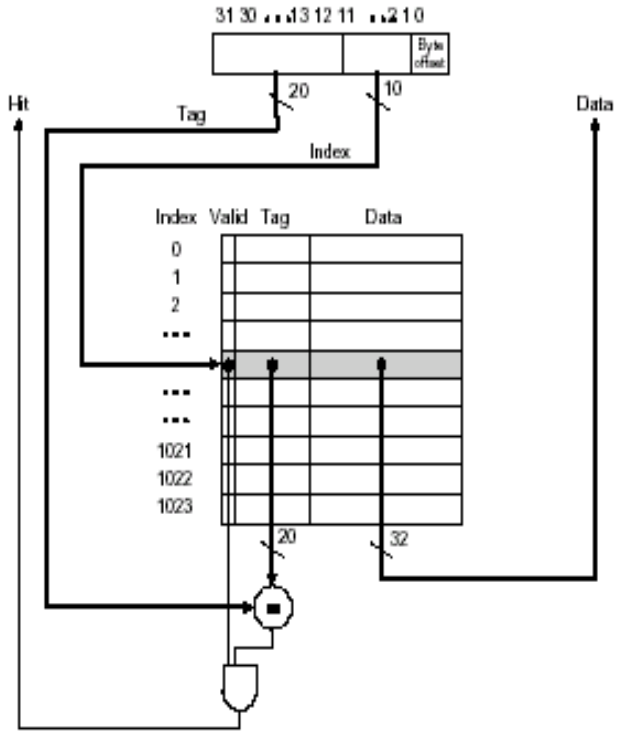
Revisiting Memory Hierarchy

- Terms
  - Cache Miss
  - Cache Hit
  - Hit Rate
  - Miss Rate
  - Index, Offset and Tag

Direct Mapped Cache



Direct Mapped Cache [contd…]



• What is the size of cache ?
4K

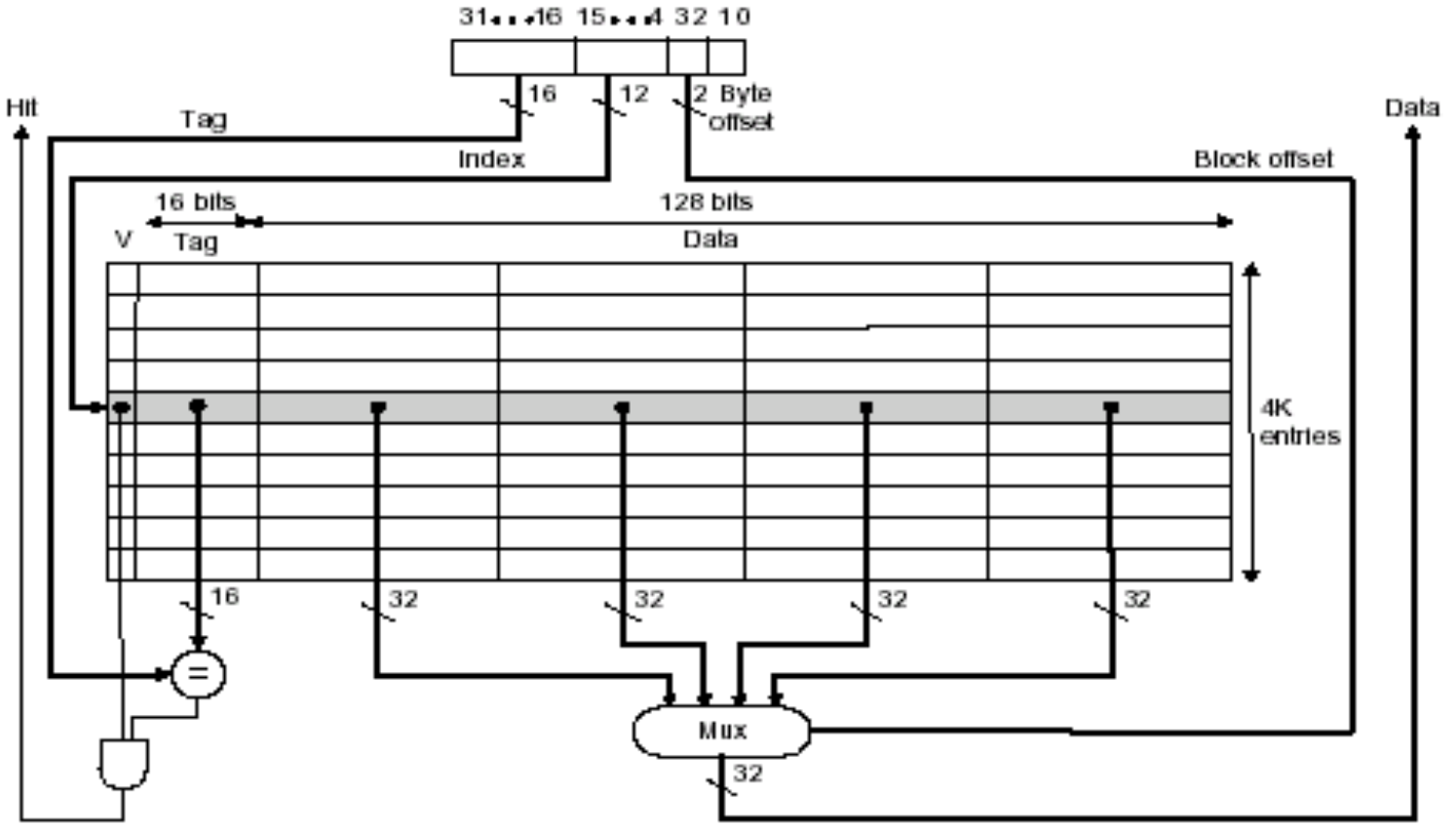
• If I read
0000 0000 0000 0000 0000 0000 1000 0001

• What is the index number checked ?
64

• If the number was found, what are the inputs to comparator ?

Direct Mapped Cache [contd…]

Taking advantage of spatial locality, we read 4 bytes at a time

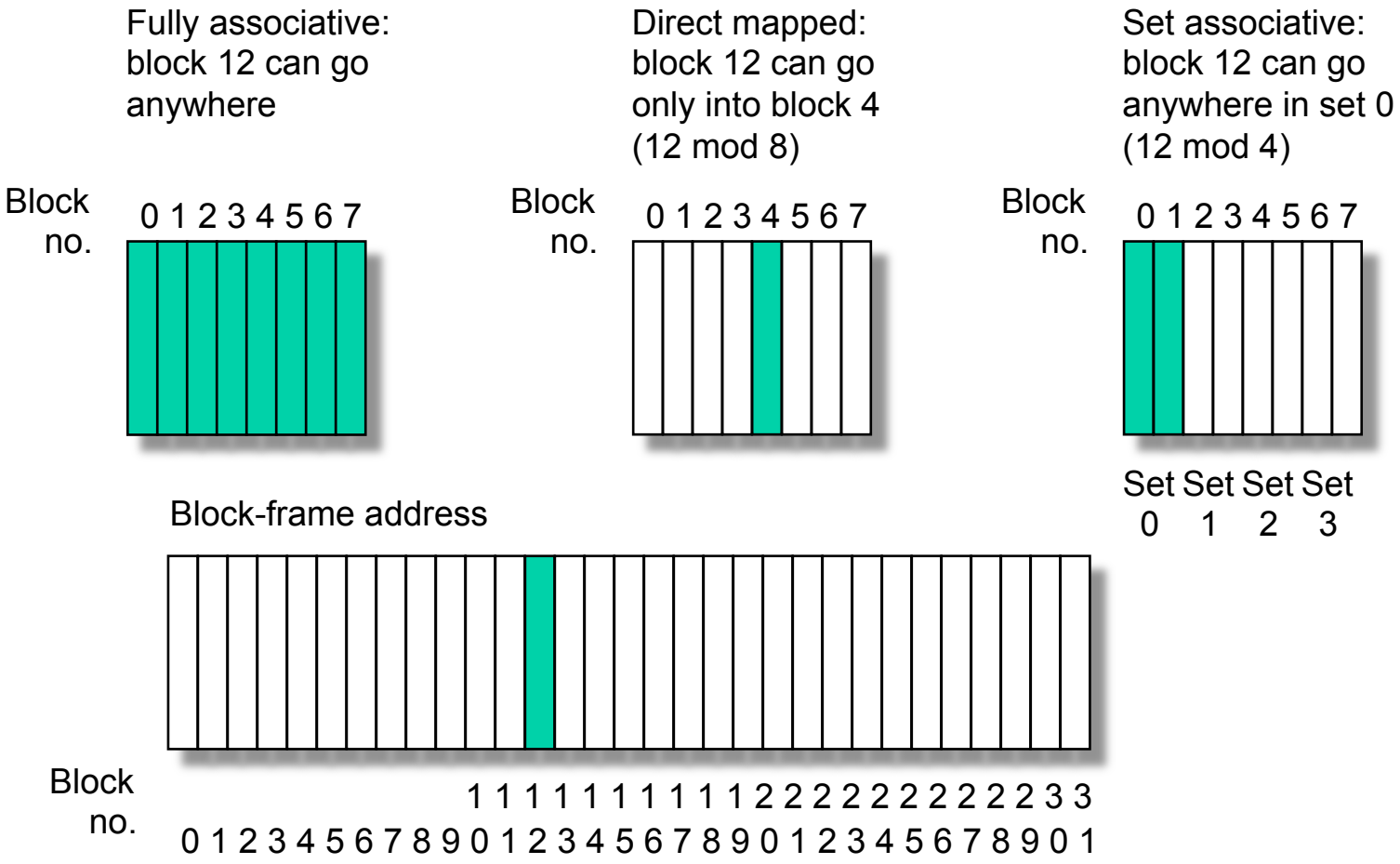


Direct Mapped Cache [contd…]

- Advantage
  - Simple
  - Fast
- Disadvantage
  - Mapping is fixed !!!

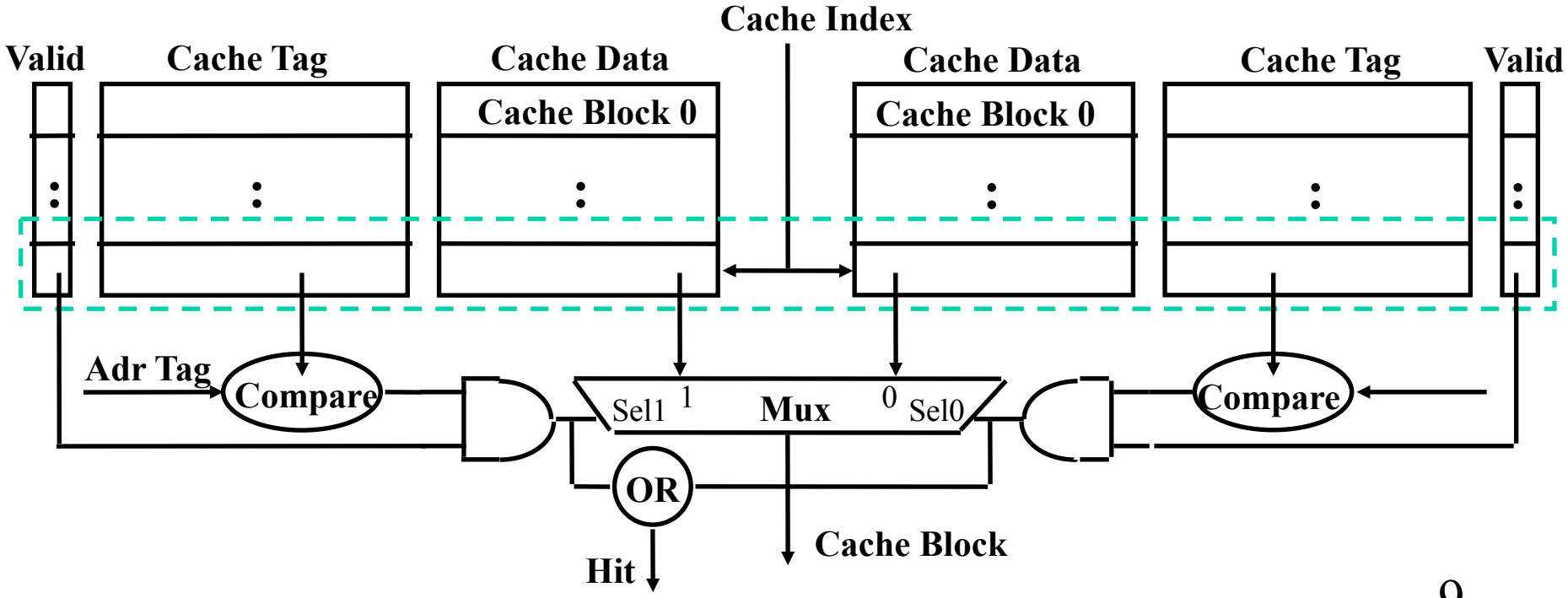
Associative Caches

- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets

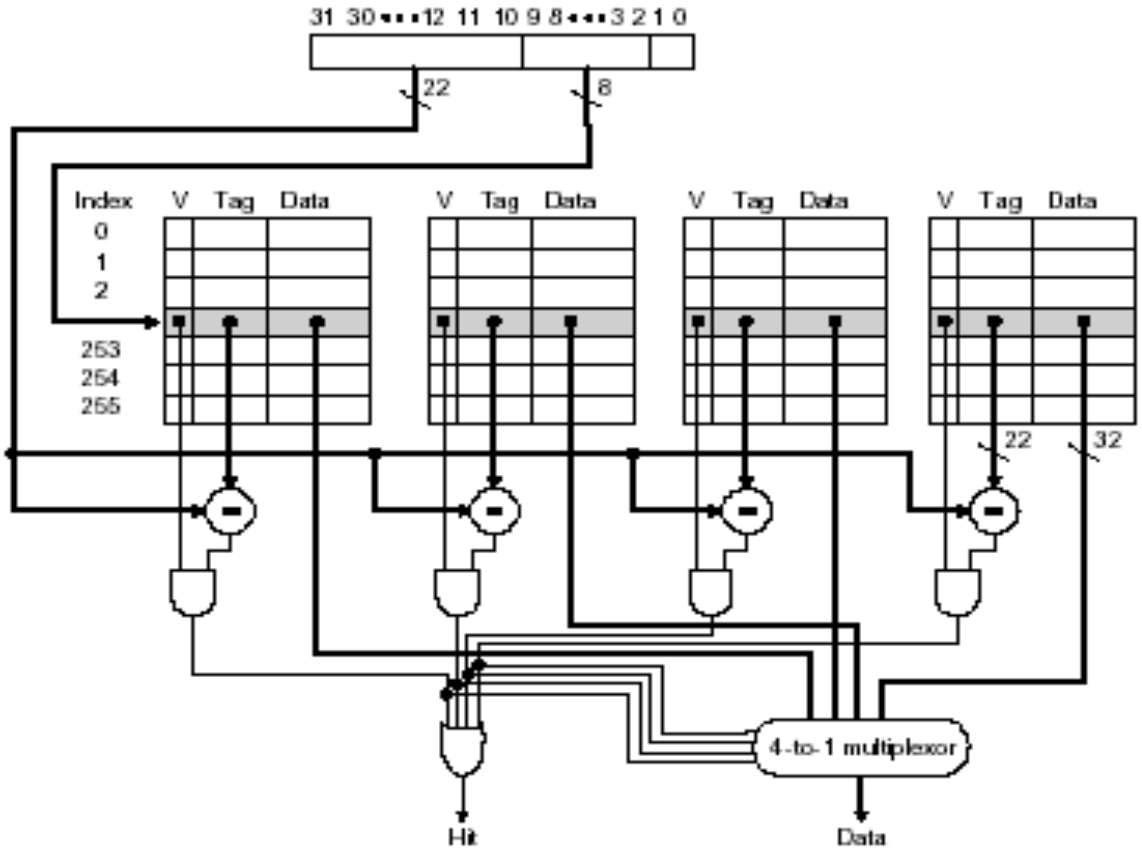


Set Associative Cache

- 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



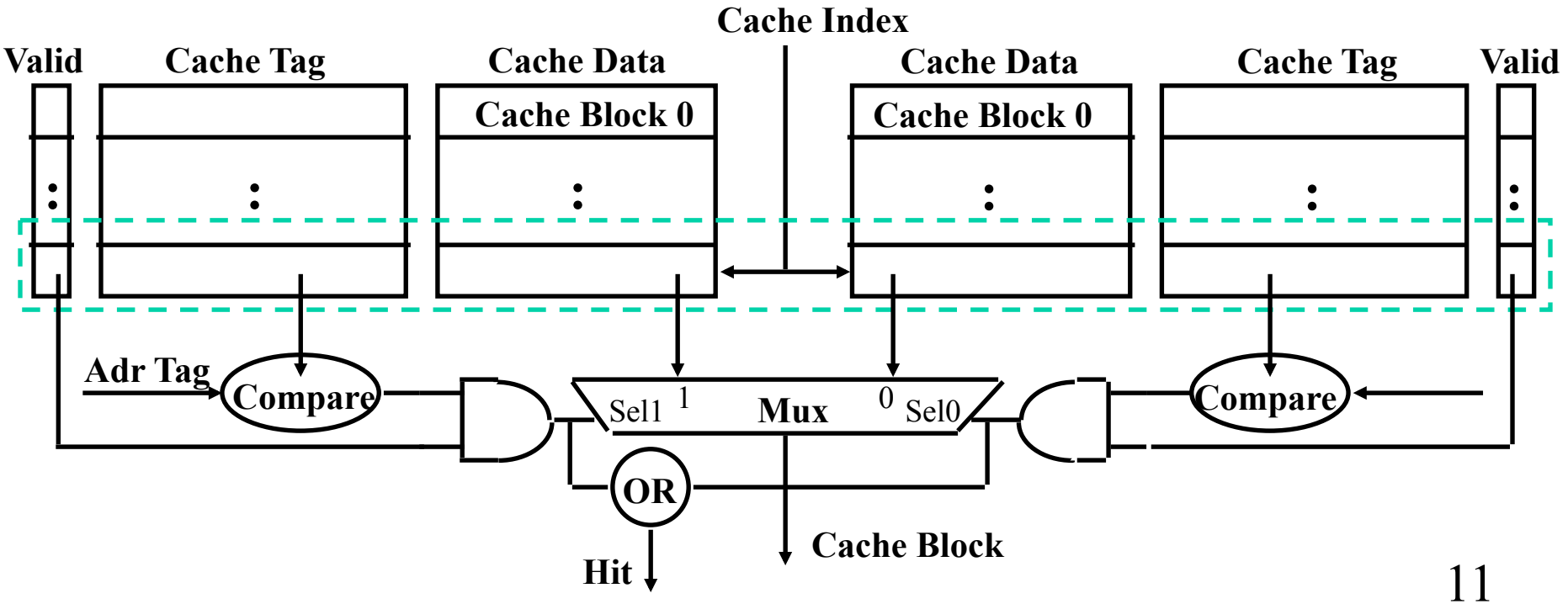
Example: 4-way set associative Cache



What is the cache size in this case ?

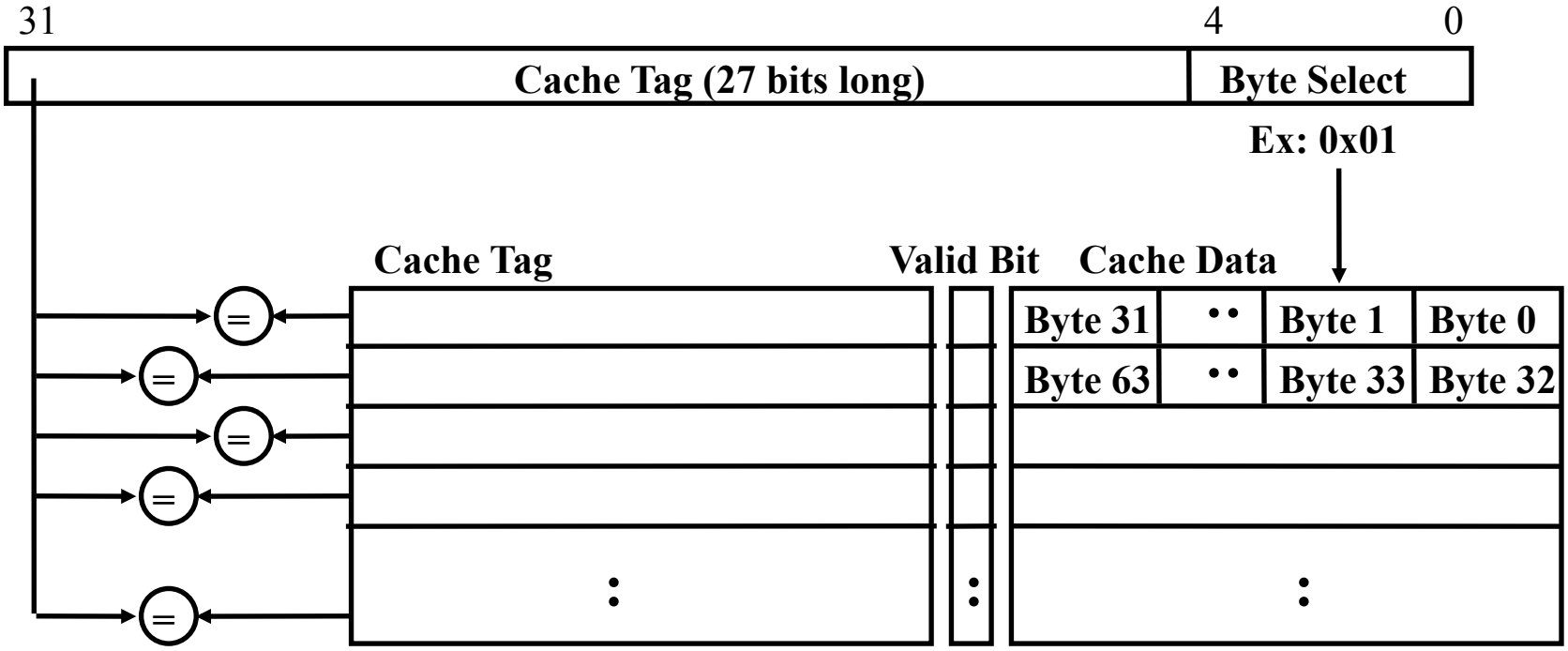
Disadvantages of Set Associative Cache

- 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:



Fully Associative Cache

- 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



Cache Misses

- Compulsory (cold start or process migration, first reference): first access to a block
  - "Cold" fact of life: not a whole lot you can do about it
  - Note: If you are going to run "billions" of instruction, Compulsory Misses are insignificant
- Capacity:
  - Cache cannot contain all blocks access 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

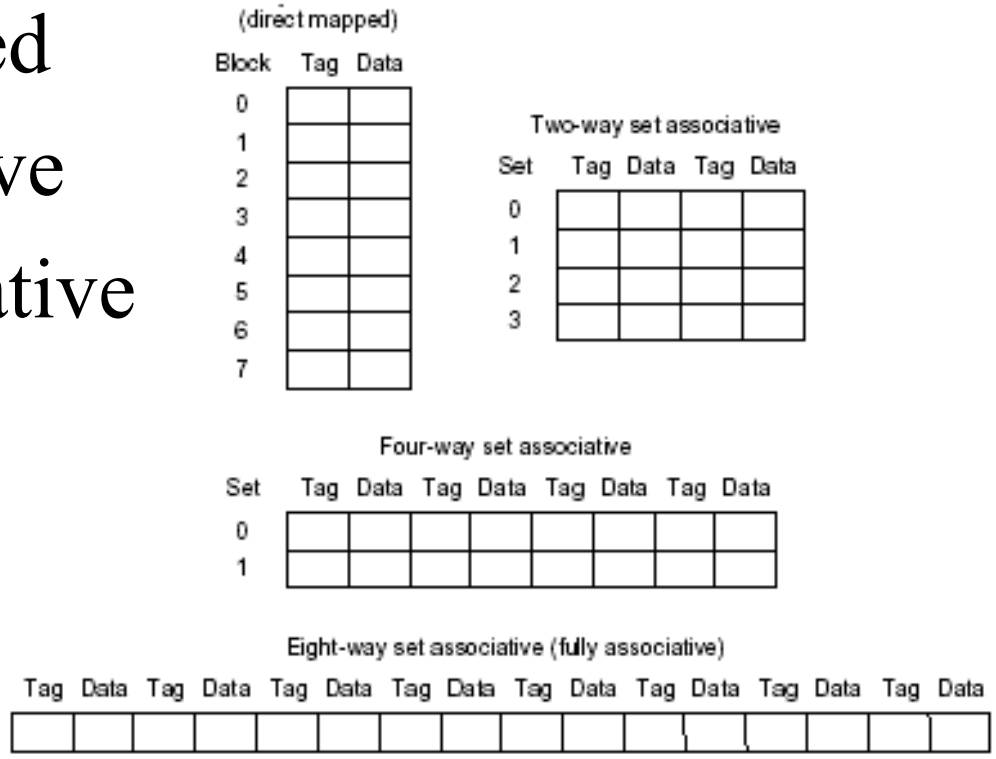
	Direct Mapped	N-way Set Associative	Fully Associative
Cache Size	Big	Medium	Small
Compulsory Miss	Same	Same	Same
Conflict Miss	High	Medium	Zero
Capacity Miss	Low	Medium	High
Coherence Miss	Same	Same	Same

Four Questions for Cache Design

- 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)

Where can a block be placed in the upper level?

- Direct Mapped
- Set Associative
- Fully Associative



Which block should be replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

Associativity:

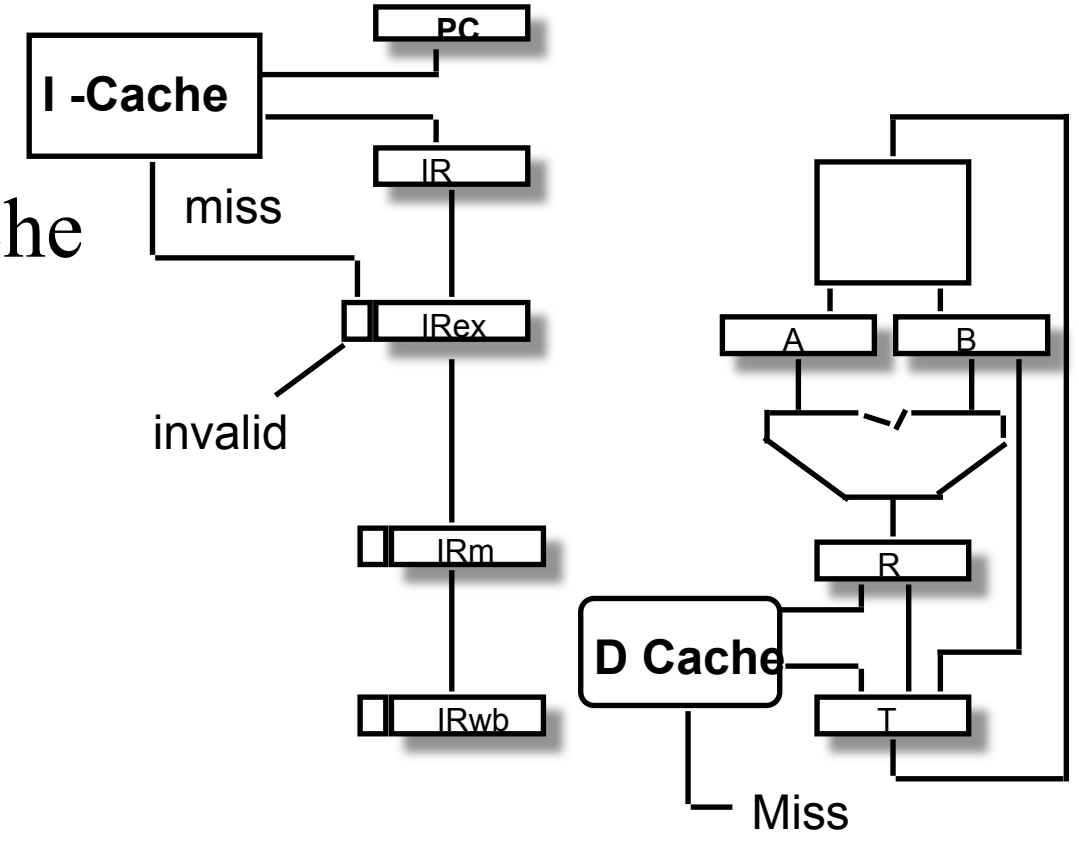
	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

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
  - WB: no writes of repeated writes
- WT always combined with write buffers so that don't wait for lower level memory

Two types of Cache

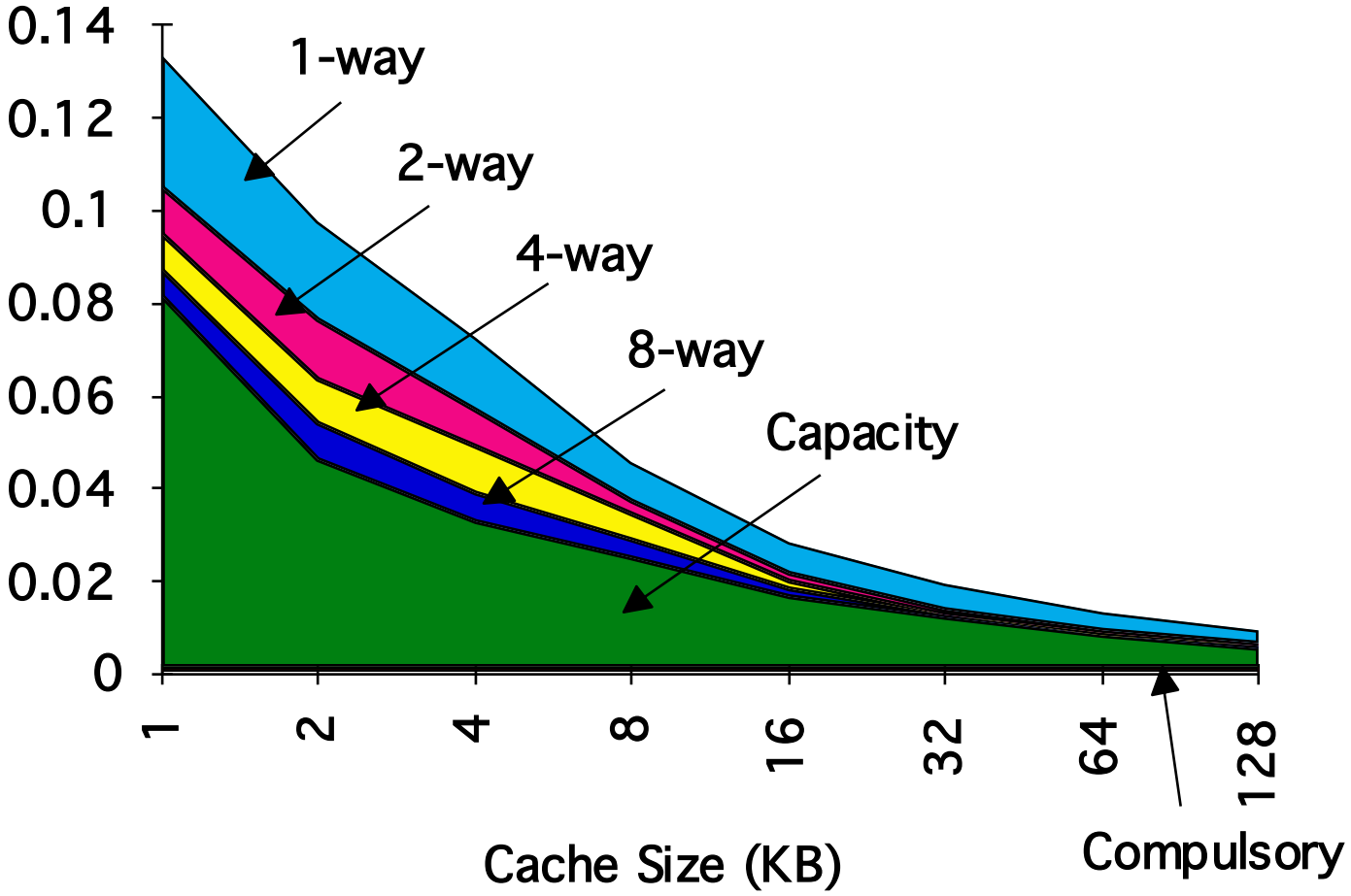
- Instruction Cache
- Data Cache



Improving Cache Performance

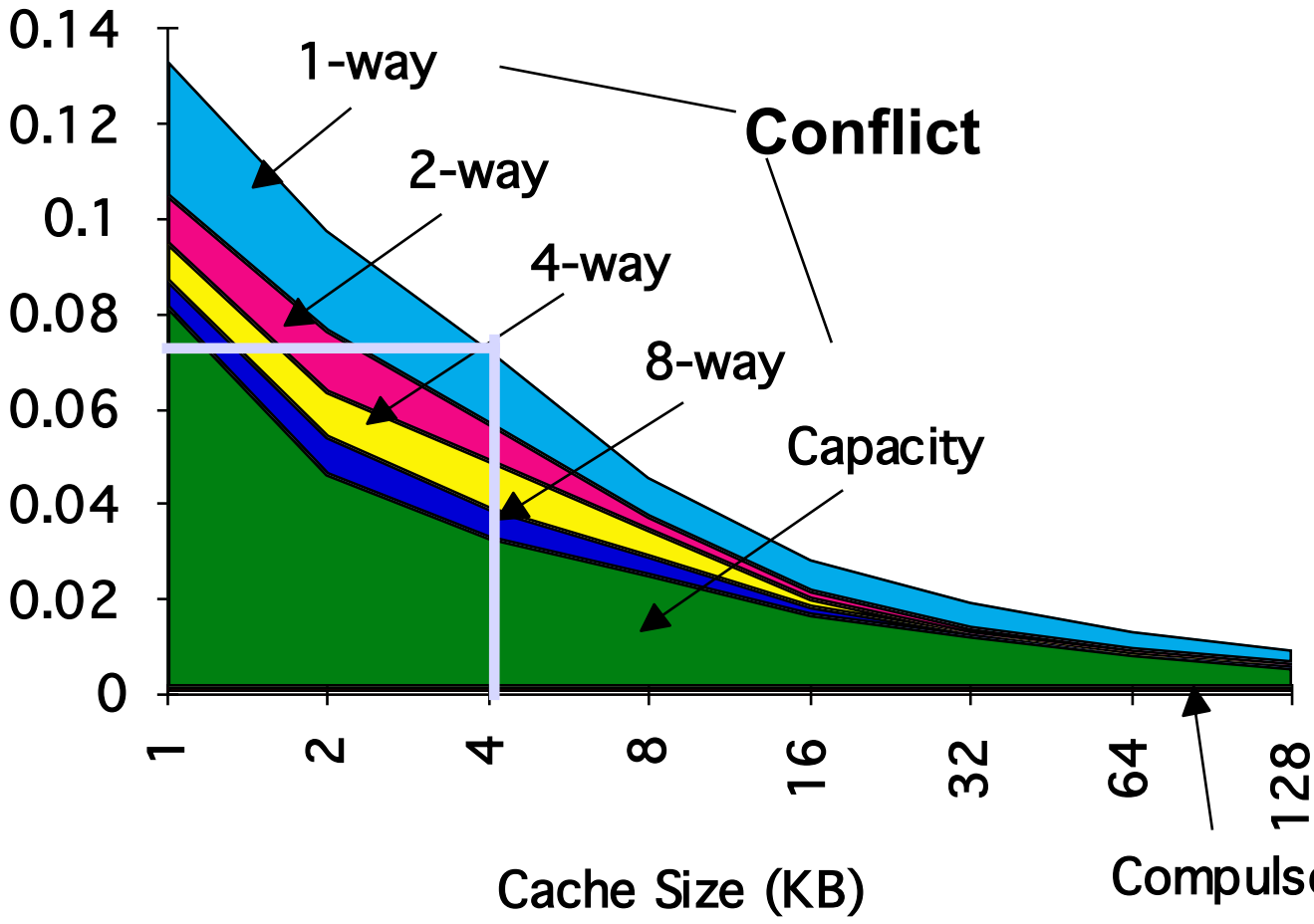
- Reduce Miss Rate
  - Associativity
  - Victim Cache
  - Compiler Optimizations
- Reduce Miss Penalty
  - Faster DRAM
  - Write Buffers
- Reduce Hit Time

Reduce Miss Rate

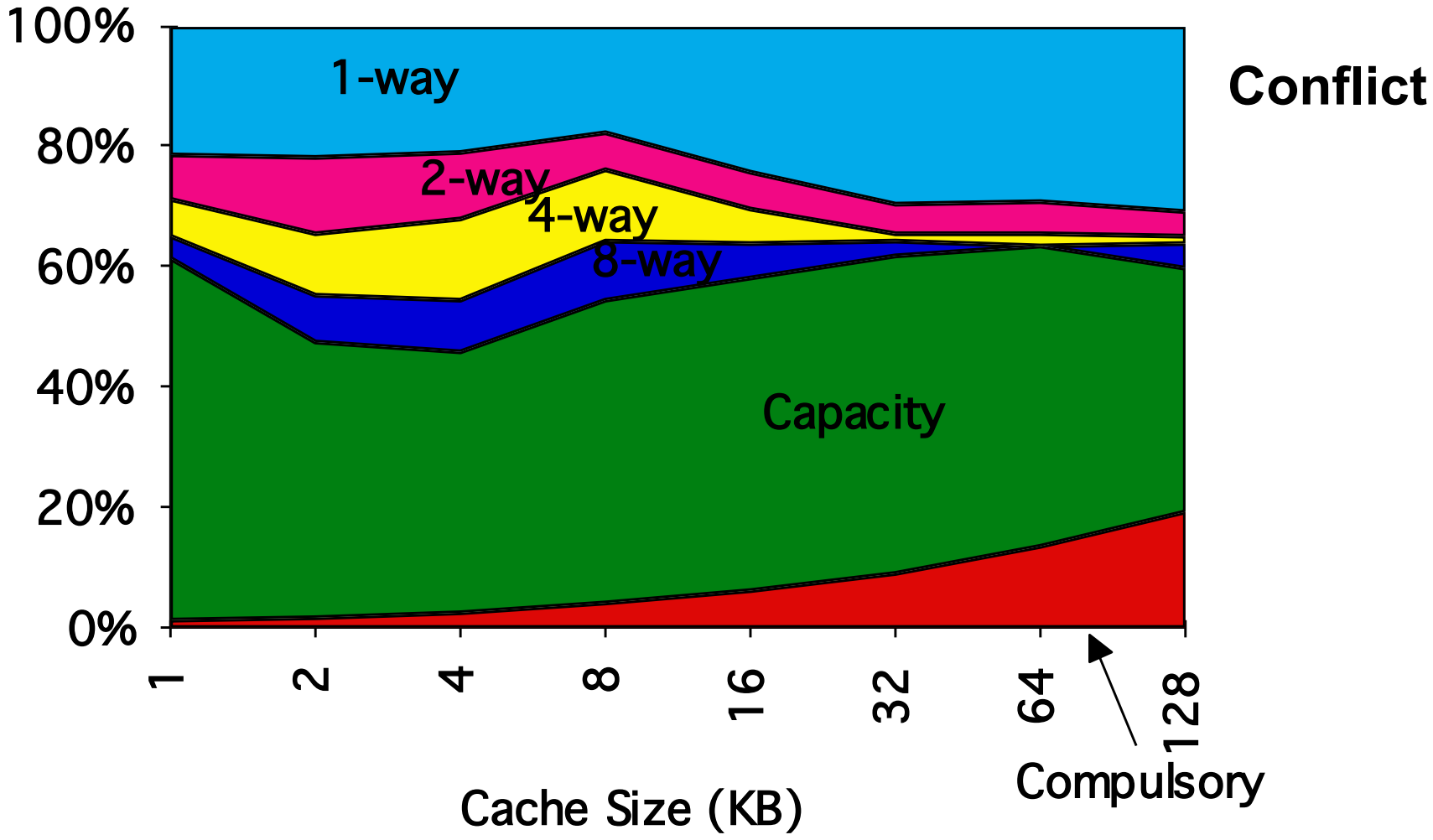


Reduce Miss Rate [contd…]

miss rate 1-way associative cache size X
= miss rate 2-way associative cache size X/2



3Cs Comparision



Compiler Optimizations

- Compiler Optimizations to reduce miss rate
  - Loop Fusion
  - Loop Interchange
  - Merge Arrays

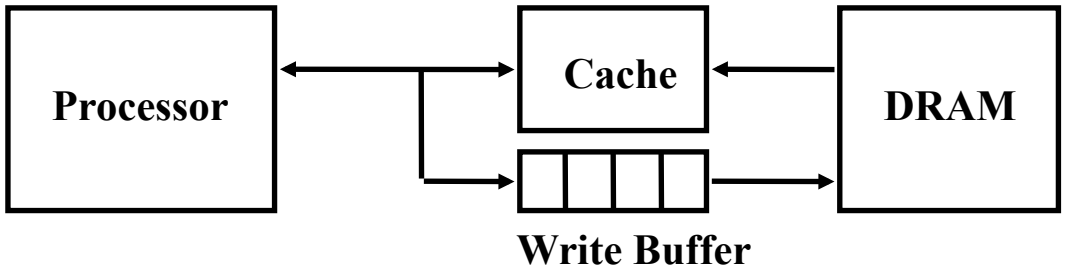
Reducing Miss Penalty

- Faster RAM memories
  - Driven by technology and cost !!!
  - Eg: CRAY uses only SRAM

Reduce Hit Time

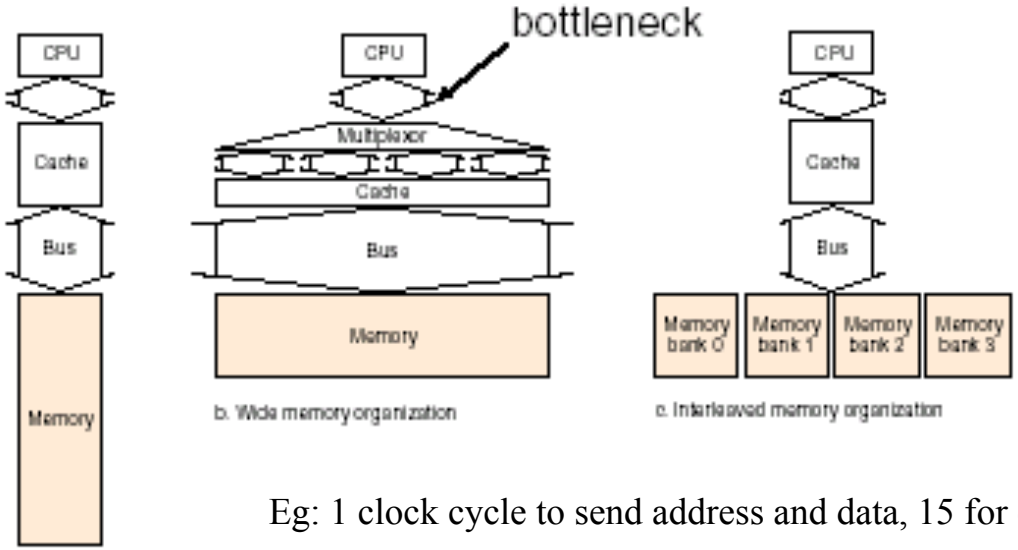
- Lower Associativity
  - Add L1 and L2 caches
  - L1 cache is small => Hit Time is critical
  - L2 cache has large => Miss Penalty is critical

Reducing Miss Penalty [contd…]



- 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:

Designing the Memory System to Support Caches



Eg: 1 clock cycle to send address and data, 15 for each DRAM access

Case 1: (Main memory of one word)
1 + 4 x 15 + 4 x 1 = 65 clk cycles, bytes/clock = 16/65 = 0.25

Case 2: (Main memory of two words)
1 + 2 x 15 + 2 x 1 = 33 clk cycles, bytes/clock = 16/33 = 0.48

Case 3: (Interleaved memory)
1 + 1 x 15 + 4 x 1 = 20 clk cycles, bytes/clock = 16/20 = 0.80

The possible R/W cases

- Read Hit
  - Good for CPU !!!
- Read Miss
  - Stall CPU, fetch, Resume
- Write Hit
  - Write Through
  - Write Back
- Write Miss
  - Write entire block into memory, Read into cache

Performance

- CPU time = (CPU execution clock cycles +
  Memory stall clock cycles) x clock cycle time

- Memory stall clock cycles =
  (Reads x Read miss rate x Read miss penalty +
  Writes x Write miss rate x Write miss penalty)

- Memory stall clock cycles =
  Memory accesses x Miss rate x Miss penalty

- Different measure: AMAT

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

- Note: *memory hit time is included in execution cycles.*

Performance [contd…]

- Suppose a processor executes at
  - Clock Rate = 200 MHz (5 ns per cycle)
  - Base CPI = 1.1
  - 50% arith/logic, 30% ld/st, 20% control
- Suppose that 10% of memory operations get 50 cycle miss penalty
- Suppose that 1% of instructions get same miss penalty
- CPI = Base CPI + average stalls per instruction

```
= 1.1(cycles/ins)  +
  [ 0.30 (DataMops/ins) x 0.10 (miss/DataMop) x 50 (cycle/miss)] +
  [ 1 (InstMop/ins) x 0.01 (miss/InstMop) x 50 (cycle/miss)]
= (1.1 +  1.5 + .5) cycle/ins
= 3.1
```

- 58% of the time the proc is stalled waiting for memory!
- AMAT=(1/1.3)x[1+0.01x50]+(0.3/1.3)x[1+0.1x50]=2.54

Summary

- The Principle of Locality:
  - Program likely to access a relatively small portion of the address space at any instant of time.
    - Temporal Locality: Locality in Time
    - Spatial Locality: Locality in Space
- Three (+1) Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Conflict Misses: increase cache size and/or associativity.
    Nightmare Scenario: ping pong effect!
  - Capacity Misses: increase cache size
  - Coherence Misses: Caused by external processors or I/O devices
- Cache Design Space
  - total size, block size, associativity
  - replacement policy
  - write-hit policy (write-through, write-back)
  - write-miss policy

Summary [contd…]

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation
- The optimal choice is a compromise
  - depends on access characteristics
    - workload
    - use (I-cache, D-cache, TLB)
  - depends on technology / cost
- Simplicity often wins