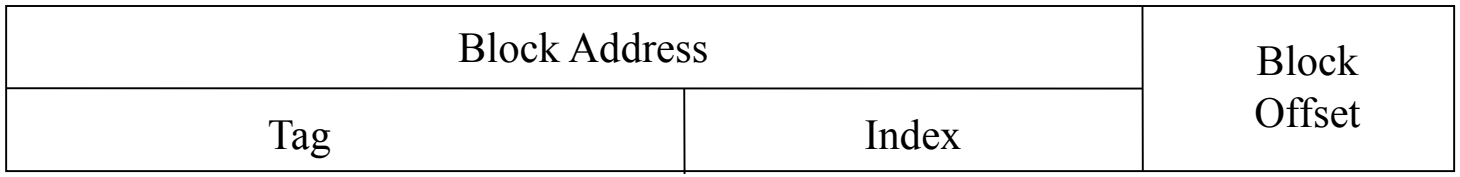
Memory Hierarchy: Cache Performance

Dr. Tao Xie

These slides are adapted from notes by Dr. David Patterson (UCB)

Q2: Block Identification

- Tag on each block
  - No need to check index or block offset
- Increasing associativity shrinks index, expands tag

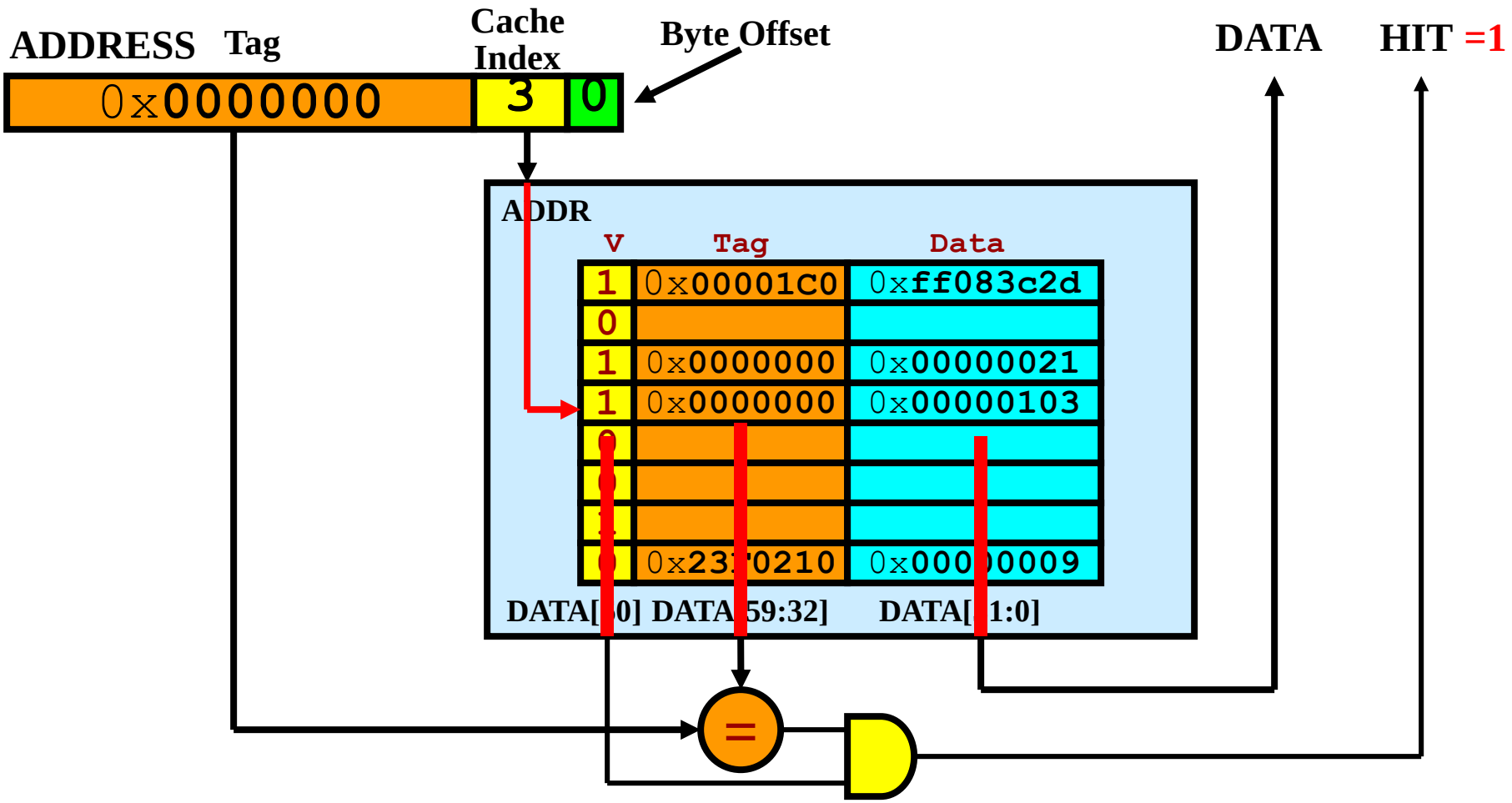


Fully Associative: No index

Direct Mapped: Large index

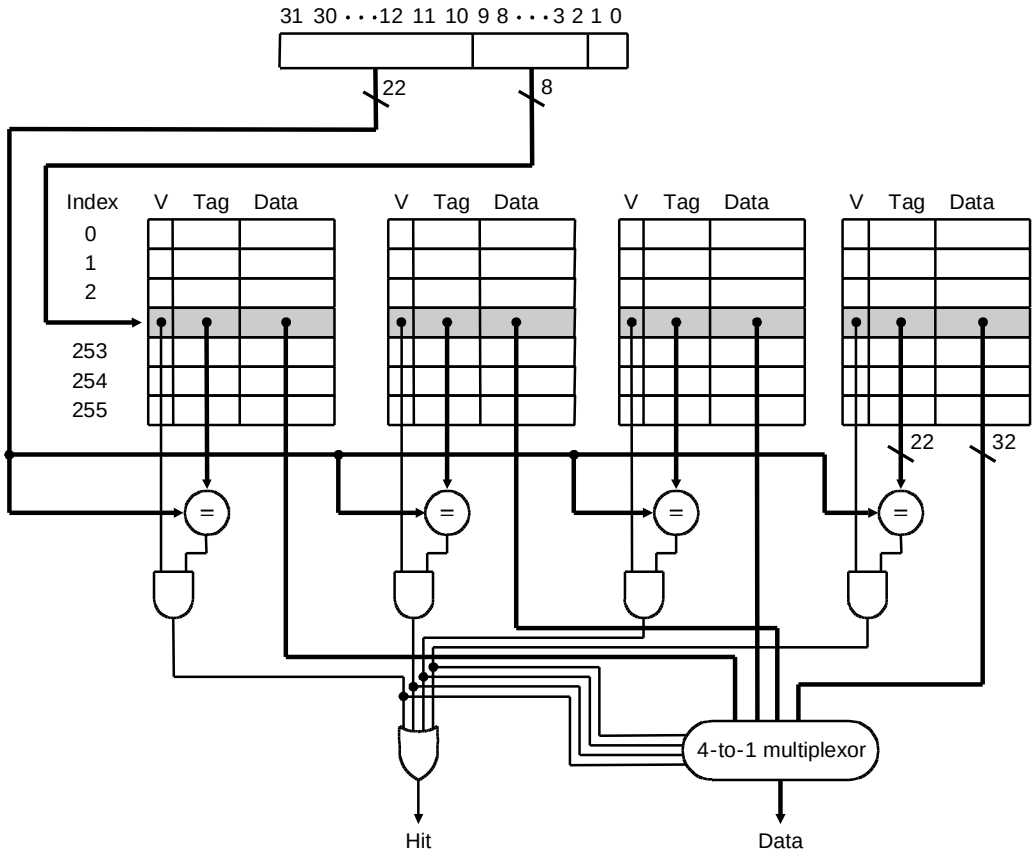
An address is divided into two parts. The block address can be further divided into the tag field and the index field. The block offset field selects the desired data from the block, the index field selects the set, and the tag field is compared against it for a hit.[2]

Direct-Mapped Cache Design



Set Associative Cache Design

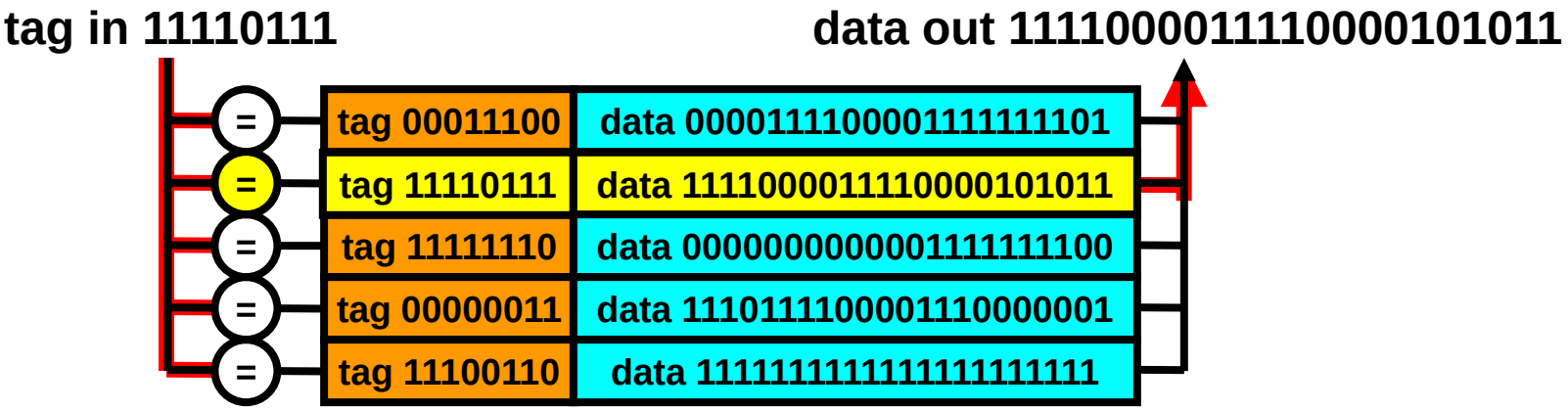
- Key idea:
  - Divide cache into sets
  - Allow block anywhere in a set
- Advantages:
  - Better hit rate
- Disadvantage:
  - More tag bits
  - More hardware
  - Higher access time



A Four-Way Set-Associative Cache

Fully Associative Cache Design

- Key idea: m-way set associative
  - 1 comparator required for each block
  - No address decoding
  - Practical only for small caches due to hardware demands



Calculating Bits in Cache

- How many total bits are needed for a direct- mapped cache with 64 KBytes of data and one word blocks, assuming a 32-bit address and one word equal to 4 bytes?

- How many total bits would be needed for a 4-way set associative cache to store the same amount of data

- How many total bits are needed for a direct- mapped cache with 64 KBytes of data and 8 word blocks, assuming a 32-bit address?

Calculating Bits in Cache

- How many total bits are needed for a direct- mapped cache with 64 KBytes of data and one word blocks, assuming a 32-bit address?
  - 64 Kbytes = 16 K words = 2^14 words = 2^14 blocks
  - block size = 4 bytes => offset size = 2 bits,
  - #sets = #blocks = 2^14 => index size = 14 bits
  - tag size = address size - index size - offset size = 32 - 14 - 2 = 16 bits
  - bits/block = data bits + tag bits + valid bit = 32 + 16 + 1 = 49
  - bits in cache = #blocks x bits/block = 2^14 x 49 = 98 Kbytes
- How many total bits would be needed for a 4-way set associative cache to store the same amount of data
  - block size and #blocks does not change
  - #sets = #blocks/4 = (2^14)/4 = 2^12 => index size = 12 bits
  - tag size = address size - index size - offset = 32 - 12 - 2 = 18 bits
  - bits/block = data bits + tag bits + valid bit = 32 + 18 + 1 = 51
  - bits in cache = #blocks x bits/block = 2^14 x 51 = 102 Kbytes
- Increase associativity => increase bits in cache

Calculating Bits in Cache

- How many total bits are needed for a direct- mapped cache with 64 KBytes of data and 8-word blocks, assuming a 32-bit address (one word=4bytes)?
- Increase block size => decrease bits in cache

  –64 Kbytes = 2^14 words = (2^14)/8 = 2^11 blocks

  –block size = 32 bytes => offset size = 5 bits,

  –#sets = #blocks = 2^11 => index size = 11 bits

  –tag size = address size - index size - offset size = 32 - 11 - 5 = 16 bits

  –bits/block = data bits + tag bits + valid bit = 8x32 + 16 + 1 = 273 bits

  –bits in cache = #blocks x bits/block = 2^11 x 273 = 68.25 Kbytes

Q3: Block Replacement

- On a miss, data must be read from memory.
- So, where do we put the new data?
  - Direct-mapped cache: must place in fixed location
  - Set-associative, fully-associative - can pick within set

Replacement Algorithms

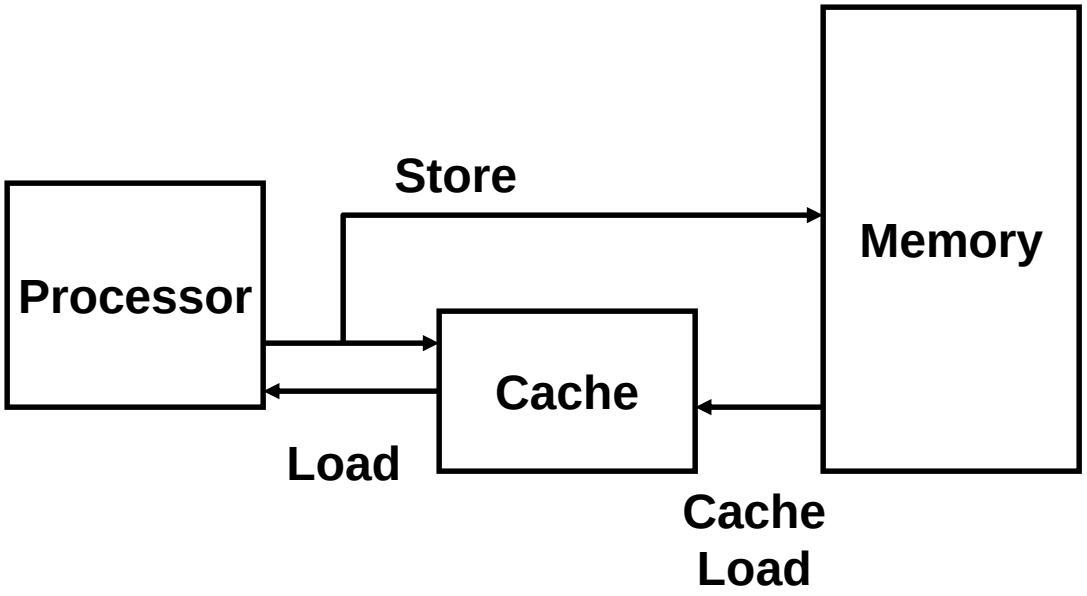
- *When a block is fetched, which block in the target set should be replaced?*
- Optimal algorithm:
    - replace the block that will not be used for the longest time (must know the future)
- Usage based algorithms:
  – Least recently used (LRU)
    - replace the block that has been referenced least recently
    - hard to implement
- Non-usage based algorithms:
  – First-in First-out (FIFO)
    - treat the set as a circular queue, replace head of queue.
    - easy to implement
  – Random (RAND)
    - replace a random block in the set
    - even easier to implement

Q4: Write Strategy

- What happens on a write?
  - Write through - write to memory, stall processor until done
  - Write buffer - place in buffer (allows pipeline to continue*)
  - Write back - delay write to memory until block is replaced in cache

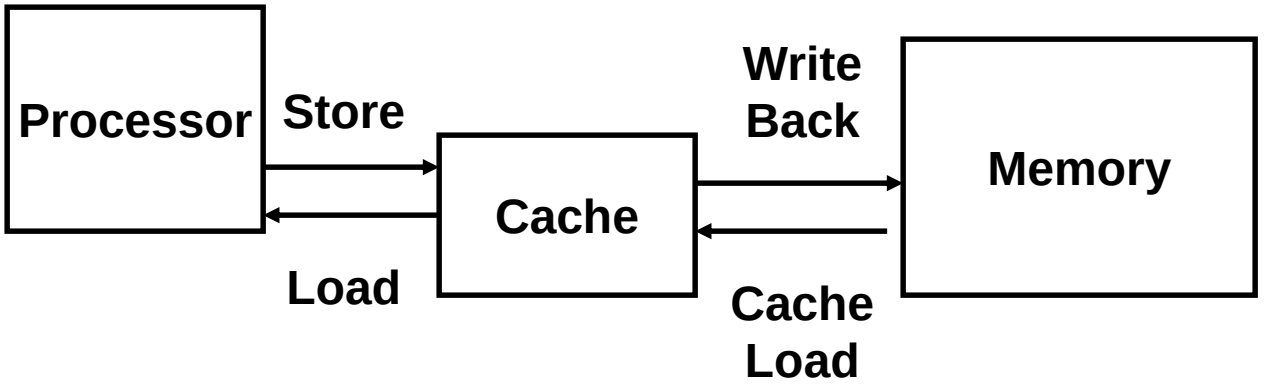
Write Through

- Store by processor updates cache *and* memory
- Memory always consistent with cache
- WT always combined with write buffers so that don't wait for lower level memory



Write Back

- Store by processor only updates cache line
- Modified line written to memory only when it is evicted
  - Requires "dirty bit" for each line
    - Set when line in cache is modified
    - Indicates that line in memory is stale
- Memory not always consistent with cache
- No writes of repeated writes



Cache Basics

- Cache: level of temporary memory storage between CPU and main memory. Improves overall memory speed by taking advantage of the principle of locality

- Cache is divided into sets; each set holds from a particular group of main memory locations

- Cache parameters
  - Cache size, block size, associativity

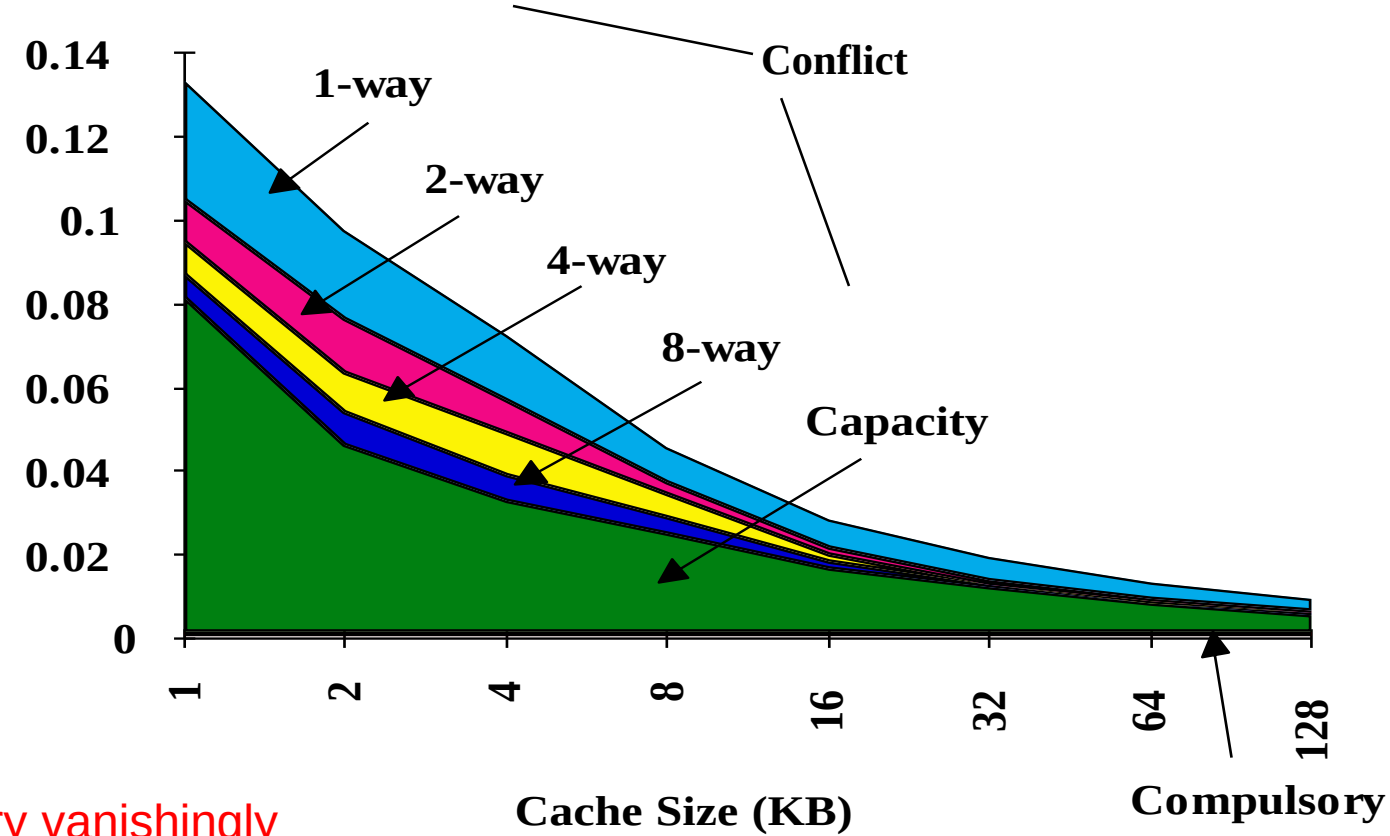
- 3 types of Cache (n total blocks):
  - Direct-mapped: n sets, each holds 1 block
  - Fully-associative: 1 set, holds n blocks
  - Set-associative: n/m sets, each holds m blocks

Classifying Misses: 3C

- *Compulsory*—The first access to a block is not in the cache, so the block must be brought into the cache. Also called *cold start misses* or *first reference misses*.
  (Misses in even an Infinite Cache)
- *Capacity*—If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved.
  (Misses in Fully Associative Size X Cache)
- *Conflict*—If block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory & capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. Also called *collision misses* or *interference misses*.
  (Misses in N-way Associative, Size X Cache)

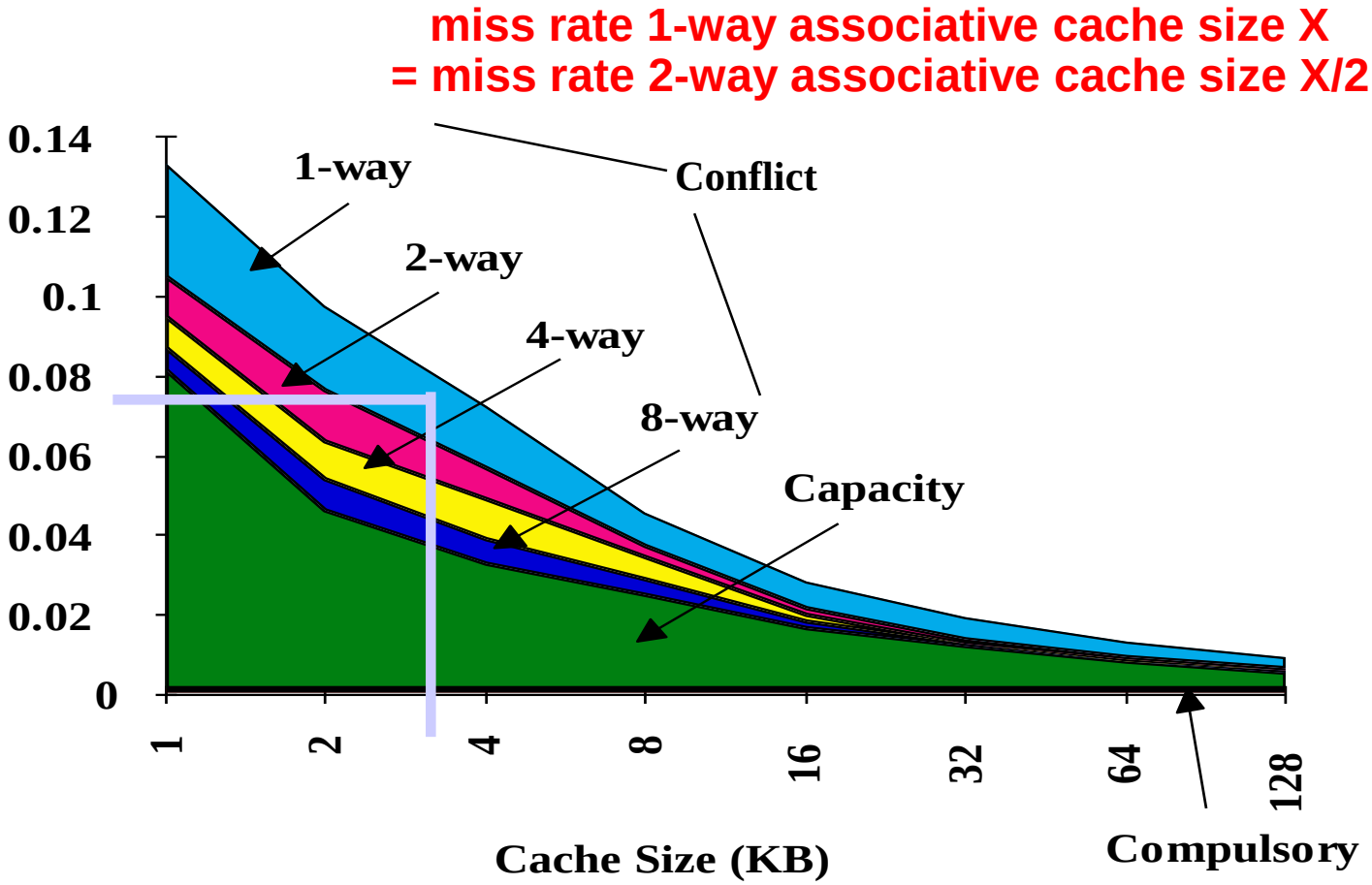
Classifying Misses: 3C

3Cs Absolute Miss Rate (SPEC92)

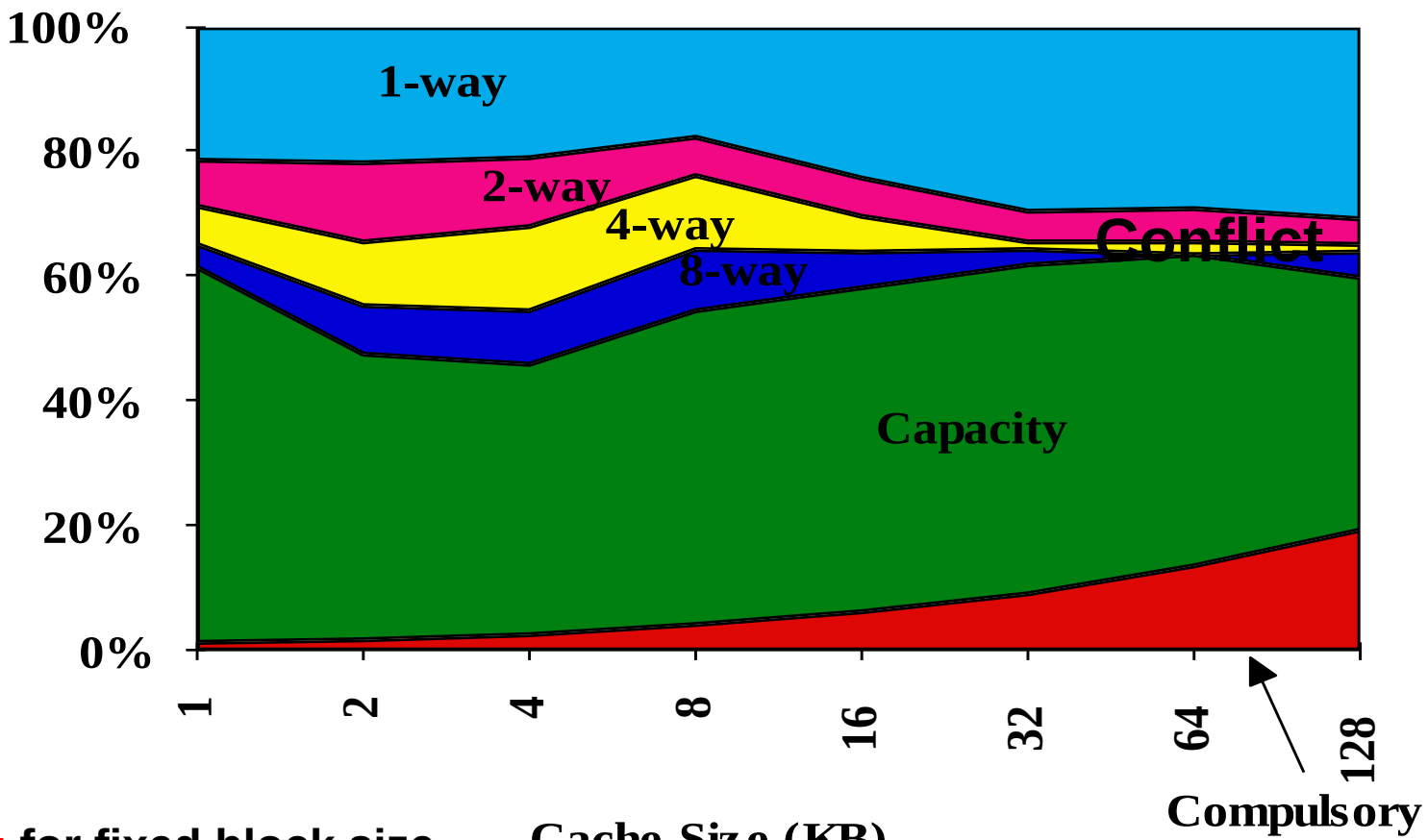


Compulsory vanishingly small

2:1 Cache Rule



3C Relative Miss Rate



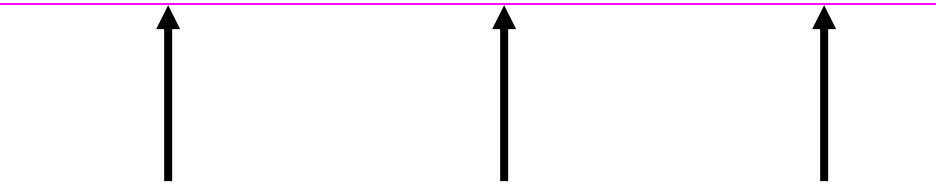
Flaws: for fixed block size

Good: insight => invention

Improve Cache Performance

improve cache and memory access times: (Page 290)

Average Memory Access Time = Hit Time + Miss Rate * Miss Penalty



$$CPUtime = IC * (CPI_{Execution} + \frac{MemoryAccess}{Instruction} * MissRate * MissPenalty * ClockCycleTime)$$

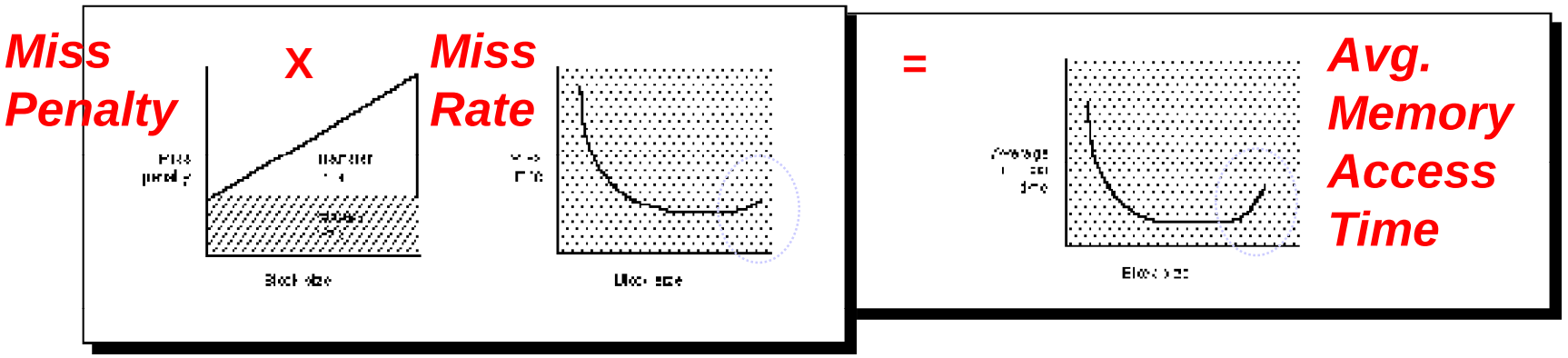
- Improve performance by:
    1. Reduce the miss rate,
    2. Reduce the miss penalty, or
    3. Reduce the time to hit in the cache.

Increasing Block Size

- One way to reduce the miss rate is to increase the block size
  - Take advantage of spatial locality
  - Decreases compulsory misses
- However, larger blocks have disadvantages
  - May increase the miss penalty (need to get more data)
  - May increase hit time (need to read more data from cache and larger mux)
  - May increase miss rate, since conflict misses
- Increasing the block size can help, but don't overdo it.

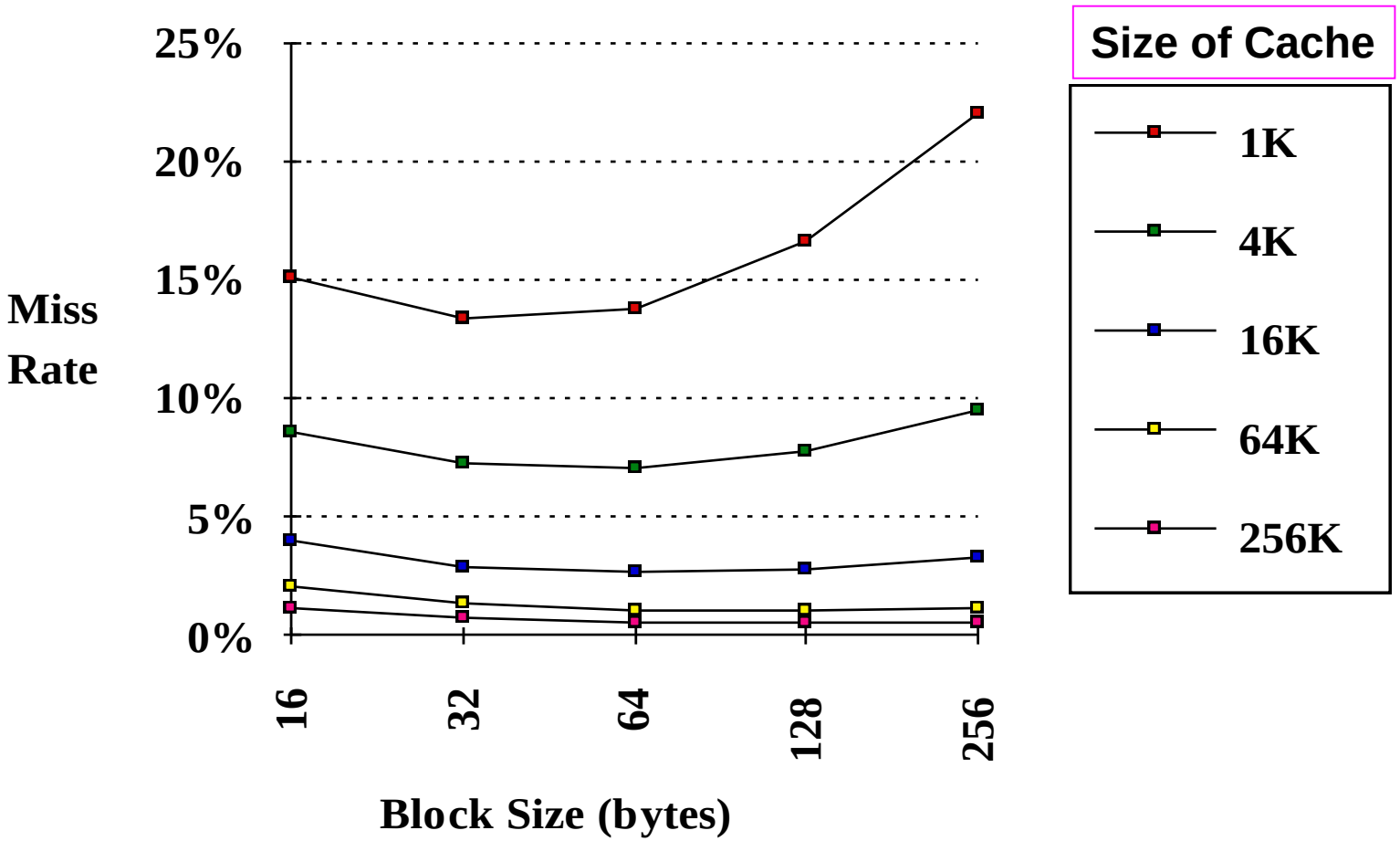
Block Size vs. Cache Measures

- Increasing Block Size generally increases Miss Penalty and decreases Miss Rate
- As the block size increases the AMAT starts to decrease, but eventually increases



Reducing Cache Misses: 1. Larger Block Size

Using the principle of locality. The larger the block, the greater the chance parts of it will be used again.



Reducing Cache Misses: 2. Higher Associativity

- Increasing associativity helps reduce conflict misses
- 2:1 Cache Rule:
  - The miss rate of a direct mapped cache of size N is about equal to the miss rate of a 2-way set associative cache of size N/2
  - For example, the miss rate of a 32 Kbyte direct mapped cache is about equal to the miss rate of a 16 Kbyte 2-way set associative cache
- Disadvantages of higher associativity
  - Need to do large number of comparisons
  - Need n-to-1 multiplexor for n-way set associative
  - Could increase hit time

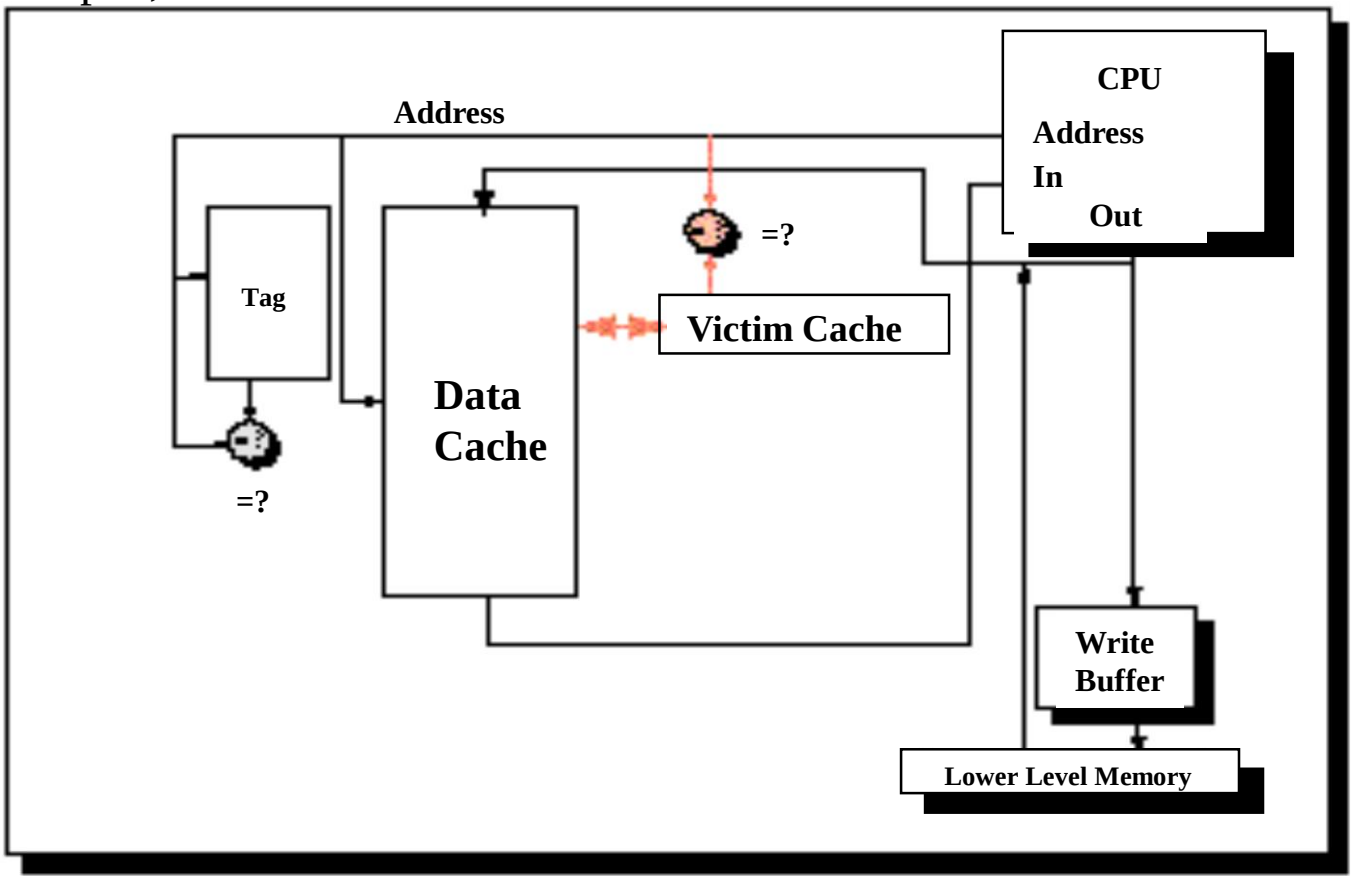
AMAT vs. Associativity

Cache Size	Associativity			
(KB)	1-way	2-way	4-way	8-way
1	7.65	6.60	6.22	5.44
2	5.90	4.90	4.62	4.09
4	4.60	3.95	3.57	3.19
8	3.30	3.00	2.87	2.59
16	2.45	2.20	2.12	2.04
32	2.00	1.80	1.77	1.79
64	1.70	1.60	1.57	1.59
128	1.50	1.45	1.42	1.44

Red means A.M.A.T. not improved by more associativity

Reducing Cache Misses: 3. Victim Cache

- Data discarded from cache is placed in an added small buffer (victim cache).
- On a cache miss check victim cache for data before going to main memory
- Jouppi [1990]: A 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Used in Alpha, HP PA-RISC CPUs.



Cache Performance Measures

- *Hit rate*: fraction found in the cache
  - So high that we usually talk about *Miss rate = 1 - Hit Rate*
- *Hit time*: time to access the cache
- *Miss penalty*: time to replace a block from lower level, including time to replace in CPU
  - *access time*: time to access lower level
  - *transfer time*: time to transfer block
- Average memory-access time (AMAT)

  = Hit time + Miss rate x Miss penalty (ns or clocks)

Cache Performance

- Miss-oriented Approach to Memory Access:

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemAccess}{Inst} \times MissRate \times MissPenalty \right) \times CycleTime$$

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemMisses}{Inst} \times MissPenalty \right) \times CycleTime$$

  - $CPI_{Execution}$ includes ALU and Memory instructions

- Separating out Memory component entirely
  - AMAT = Average Memory Access Time
  - CPI_{ALUOps} does not include memory instructions

$$CPUtime = IC \times \left(\frac{AluOps}{Inst} \times CPI_{AluOps} + \frac{MemAccess}{Inst} \times AMAT \right) \times CycleTime$$

$$\begin{aligned} AMAT &= HitTime + MissRate \times MissPenalty \\ &= \left(HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst}\right) + \\ &\quad \left(HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data}\right) \end{aligned}$$

Calculating AMAT

- If a direct mapped cache has a hit rate of 95%, a hit time of 4 ns, and a miss penalty of 100 ns, what is the AMAT?

- If replacing the cache with a 2-way set associative increases the hit rate to 97%, but increases the hit time to 5 ns, what is the new AMAT?

Calculating AMAT

- If a direct mapped cache has a hit rate of 95%, a hit time of 4 ns, and a miss penalty of 100 ns, what is the AMAT?

  AMAT = Hit time + Miss rate x Miss penalty = 4 + 0.05 x 100 = 9 ns

- If replacing the cache with a 2-way set associative increases the hit rate to 97%, but increases the hit time to 5 ns, what is the new AMAT?

  AMAT = Hit time + Miss rate x Miss penalty = 5 + 0.03 x 100 = 8 ns

Impact on Performance

- Suppose a processor executes at
  - –Clock Rate = 200 MHz (5 ns per cycle), Ideal (no misses) 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
- Calculate AMAT?

Hint: (1) First calculate CPI = ideal CPI + average stalls per instruction

(2) Next calculate AMAT

Impact on Performance

- Suppose a processor executes at
  - Clock Rate = 200 MHz (5 ns per cycle), Ideal (no misses) 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

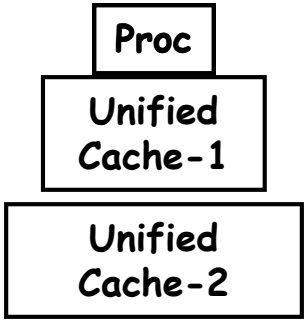
- CPI = ideal CPI + average stalls per instruction
  =1.1(cycles/ins) +
  [0.30 (DataMops/ins) x 0.10 (miss/DataMop) x 50 (cycle/miss)] +
  [1 (InstMop/ins) x 0.01 (miss/InstMop) x 50 (cycle/miss)]
  = (1.1 + 1.5 + .5) cycle/ins = 3.1

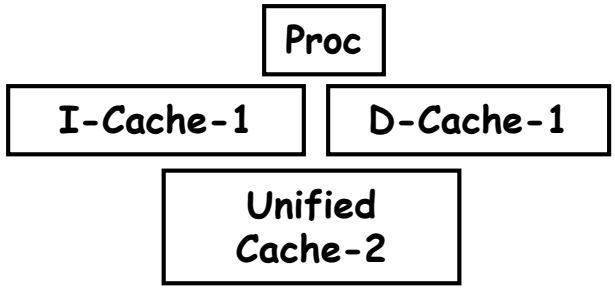
- AMAT=(1/1.3)x[1+0.01x50]+(0.3/1.3)x[1+0.1x50]=2.54

$$AMAT = HitTime + MissRate \times MissPenalty$$
$$= (HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst}) +$$
$$(HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data})$$

Unified vs Split Caches

- Unified vs Separate I&D

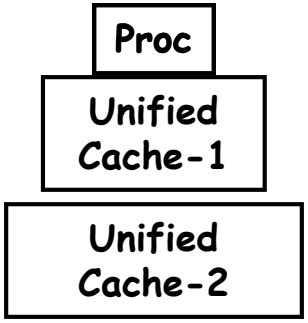


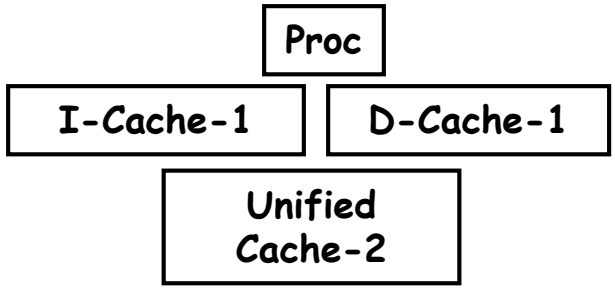


- Example:
  - 16KB I&D: Inst miss rate=0.64%, Data miss rate=6.47%
  - 32KB unified: Aggregate miss rate=1.99%
- Which is better (ignore L2 cache)?
  - Assume 33% data ops ⇒ 75% accesses from instructions (1.0/1.33)
  - hit time=1, miss time=50
  - Note that *data* hit has 1 stall for unified cache (only one port)

Unified vs Split Caches

- Unified vs Separate I&D





- Example:
  - 16KB I&D: Inst miss rate=0.64%, Data miss rate=6.47%
  - 32KB unified: Aggregate miss rate=1.99%
- Which is better (ignore L2 cache)?
  - Assume 33% data ops ⇒ 75% accesses from instructions (1.0/1.33)
  - hit time=1, miss time=50
  - Note that *data* hit has 1 stall for unified cache (only one port)

$AMAT_{Harvard}$=75%x(1+0.64%x50)+25%x(1+6.47%x50) = 2.05

$AMAT_{Unified}$=75%x(1+1.99%x50)+25%x(1+1+1.99%x50)= 2.24

Using a 2nd Level Cache

- A second level (L2) cache reduces the miss penalty by providing a large cache between the first level (L1) cahe and main memory
- L2 Equations

$$\text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1}$$

$$\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}$$

$$\text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})$$

Adding an L2 Cache

- If a direct mapped cache has a hit rate of 95%, a hit time of 4 ns, and a miss penalty of 100 ns, what is the AMAT?

- If an L2 cache is added with a hit time of 20 ns and a hit rate of 50%, what is the new AMAT?

Adding an L2 Cache

- If a direct mapped cache has a hit rate of 95%, a hit time of 4 ns, and a miss penalty of 100 ns, what is the AMAT?

  AMAT = Hit time + Miss rate x Miss penalty = 4 + 0.05 x 100 = 9 ns

- If an L2 cache is added with a hit time of 20 ns and a hit rate of 50%, what is the new AMAT?

  $AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times (Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2})$

  $= 4 + 0.05 \times (20 + 0.5 \times 100) = 7.5$ ns

Cache Performance Summary

- AMAT = Hit time + Miss rate x Miss penalty
- Split vs. Unified Cache
- 3C's of misses
  - compulsory
  - capacity
  - conflict
- Methods for improving performance
  - increase (change) cache size
  - increase (change) block size
  - increase (change) associativity
  - add a 2nd level cache