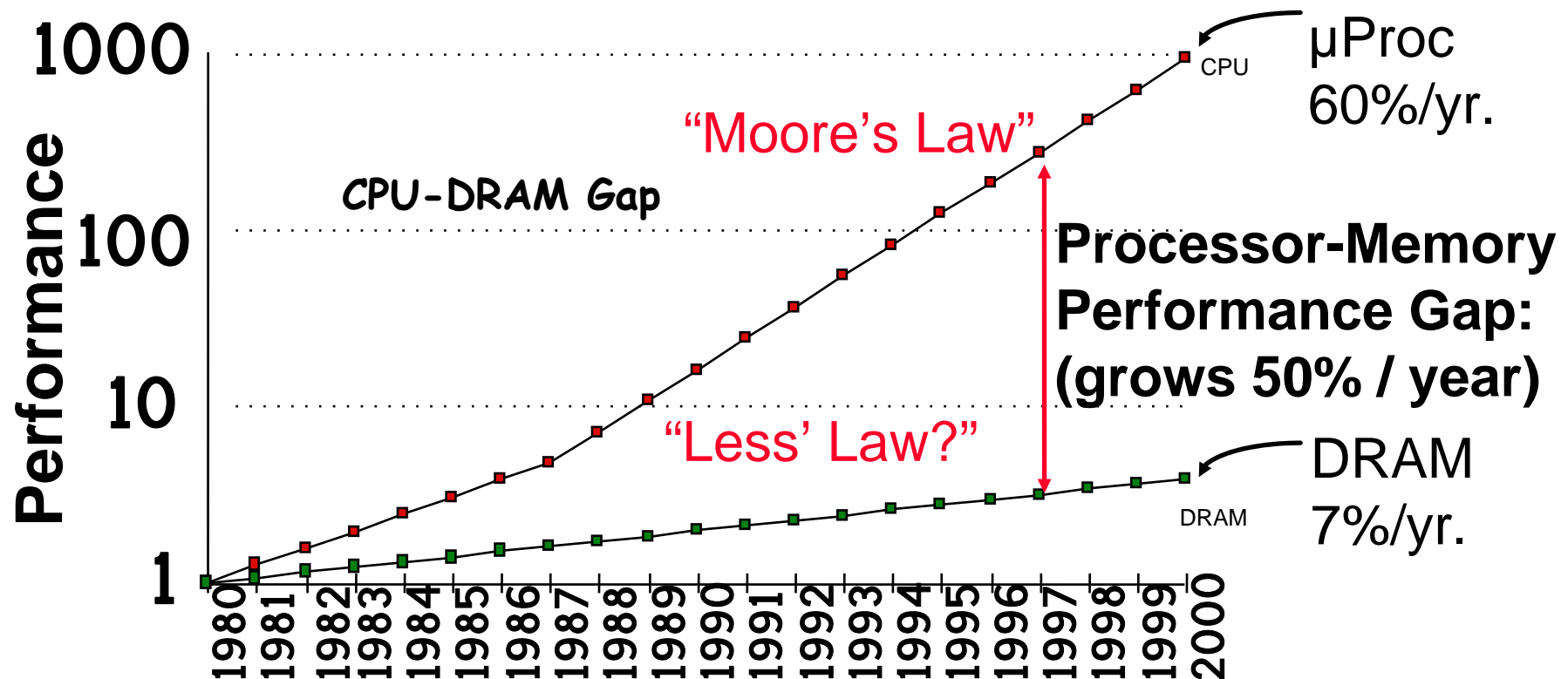


CS252
Graduate Computer Architecture
Lecture 3

Caches and Memory Systems I

January 24, 2001
Prof. John Kubiatowicz

Question: Who Cares About the Memory Hierarchy?



- 1980: no cache in μ proc; 1995 2-level cache on chip (1989 first Intel μ proc with a cache on chip)

Generations of Microprocessors

- Time of a full cache miss in instructions executed:

1st Alpha: $340 \text{ ns} / 5.0 \text{ ns} = 68 \text{ clks} \times 2 \text{ or } 136$

2nd Alpha: $266 \text{ ns} / 3.3 \text{ ns} = 80 \text{ clks} \times 4 \text{ or } 320$

3rd Alpha: $180 \text{ ns} / 1.7 \text{ ns} = 108 \text{ clks} \times 6 \text{ or } 648$

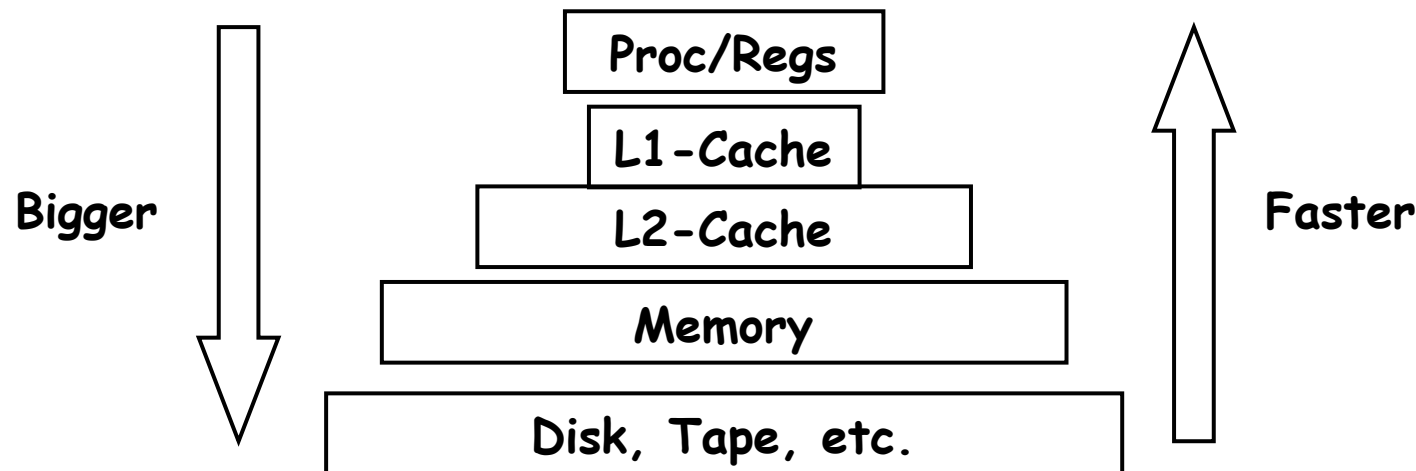
- $1/2X$ latency $\times 3X$ clock rate $\times 3X$ Instr/clock $\Rightarrow -5X$

Processor-Memory Performance Gap "Tax"

| Processor | % Area (-cost) | %Transistors (-power) |
|---|-------------------|--------------------------|
| • Alpha 21164 | 37% | 77% |
| • StrongArm SA110 | 61% | 94% |
| • Pentium Pro | 64% | 88% |
| - 2 dies per package: Proc/I\$/D\$ + L2\$ | | |
| • Caches have no inherent value, only try to close performance gap | | |

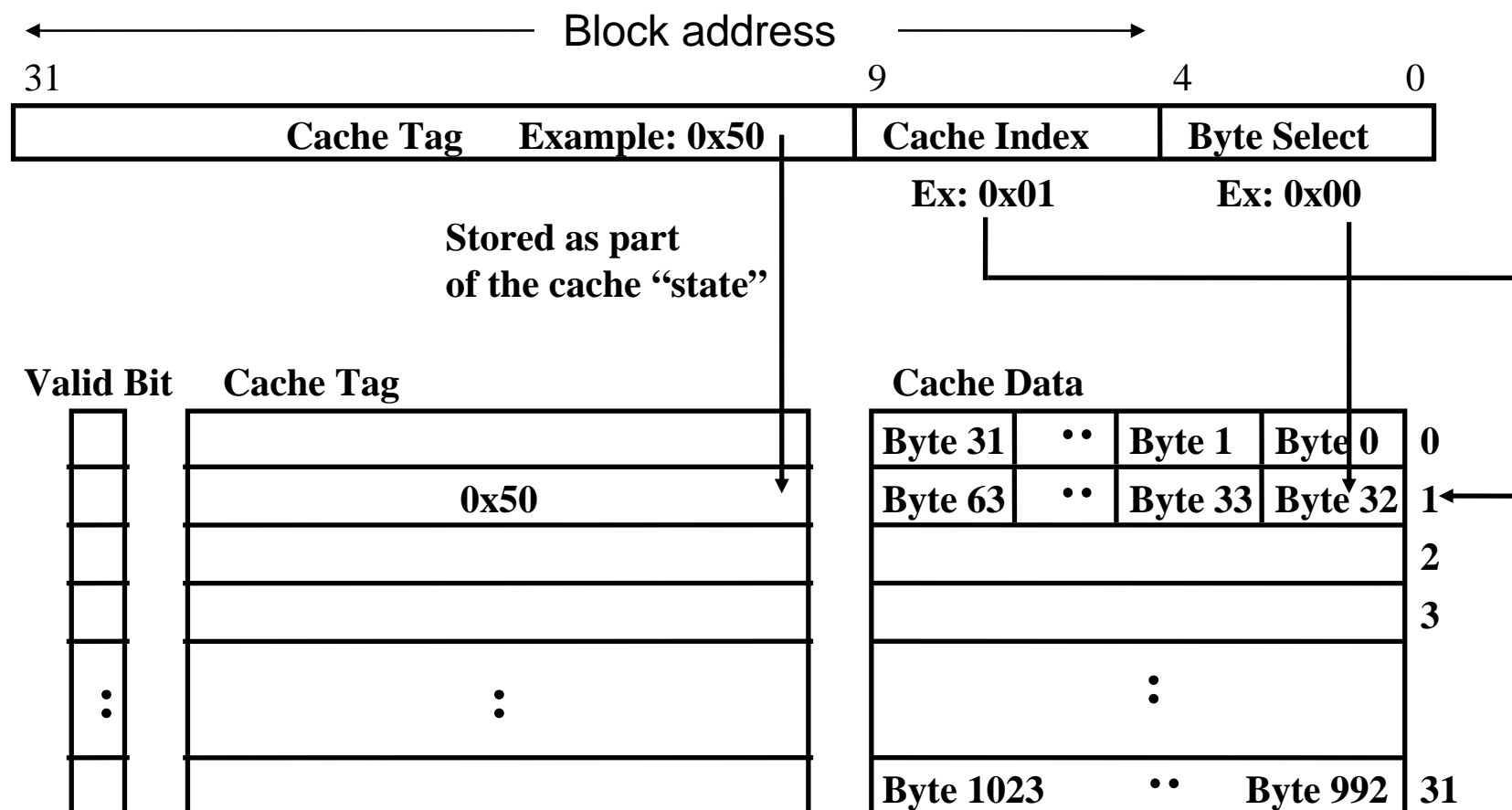
What is a cache?

- Small, fast storage used to improve average access time to slow memory.
- Exploits spacial and temporal locality
- In computer architecture, almost everything is a cache!
 - Registers a cache on variables
 - First-level cache a cache on second-level cache
 - Second-level cache a cache on memory
 - Memory a cache on disk (virtual memory)
 - TLB a cache on page table
 - Branch-prediction a cache on prediction information?



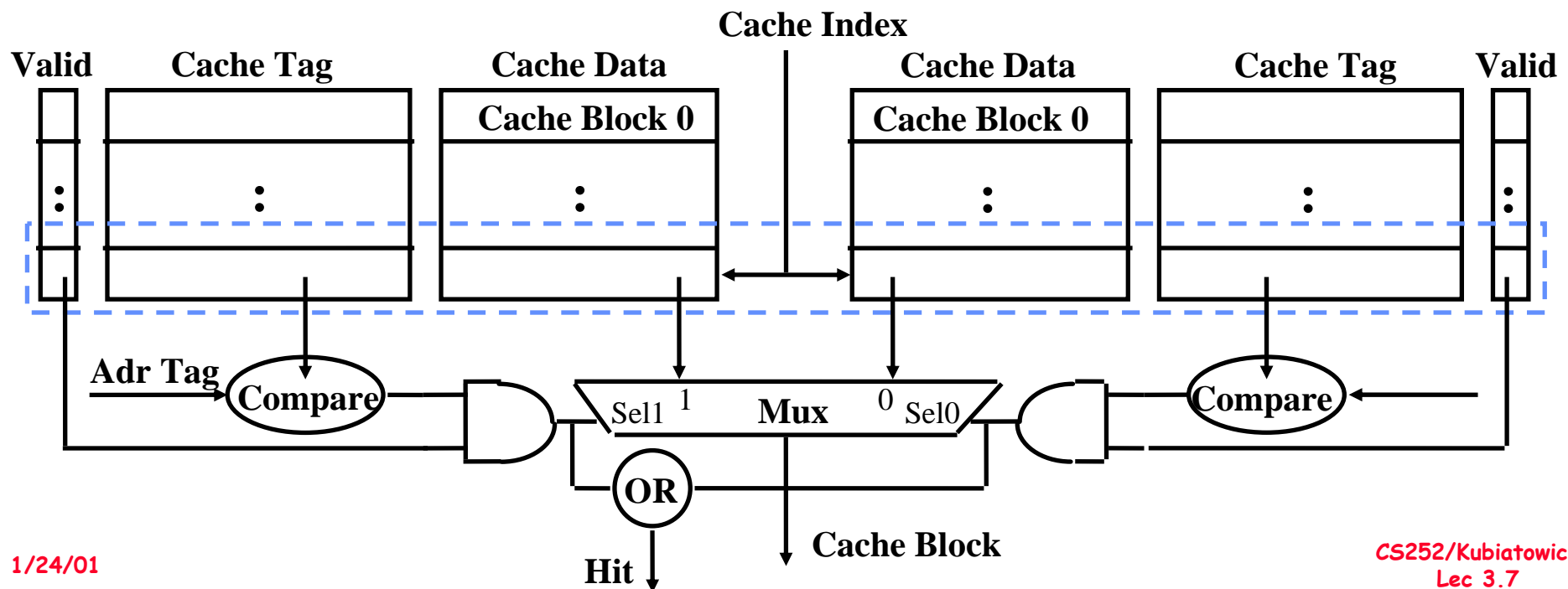
Example: 1 KB Direct Mapped Cache

- 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)



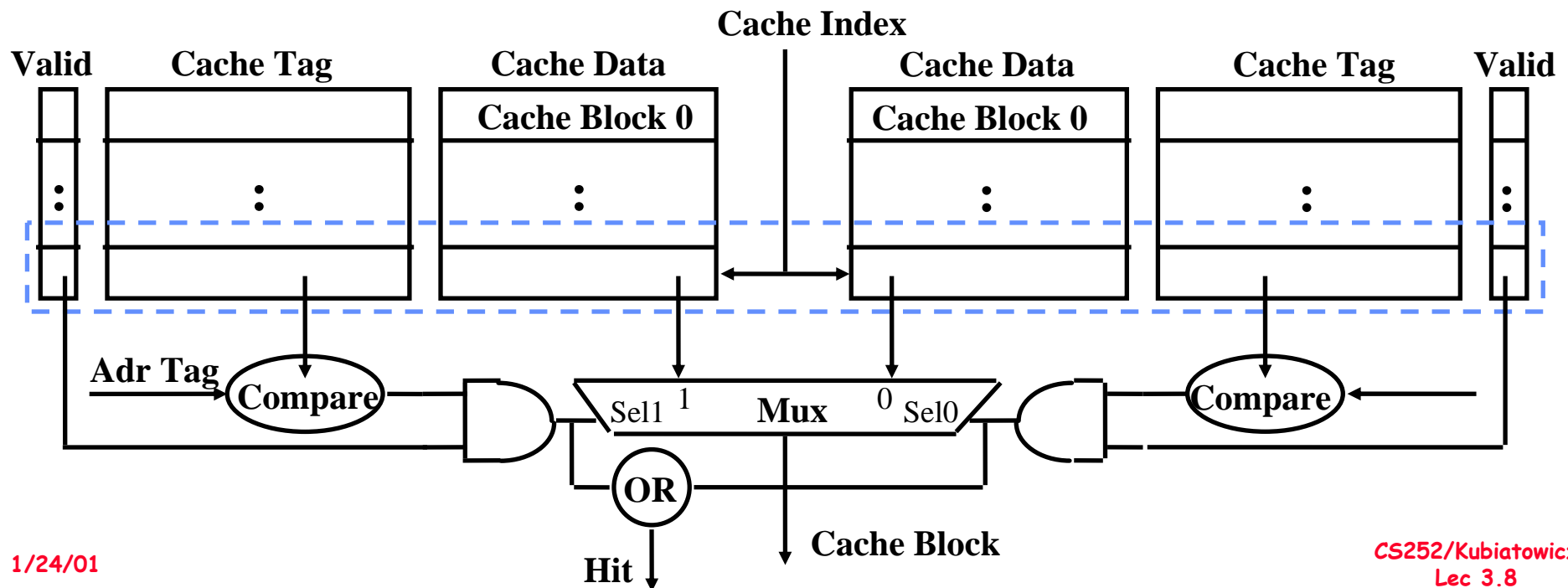
Set Associative Cache

- **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



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.



Review: 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

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

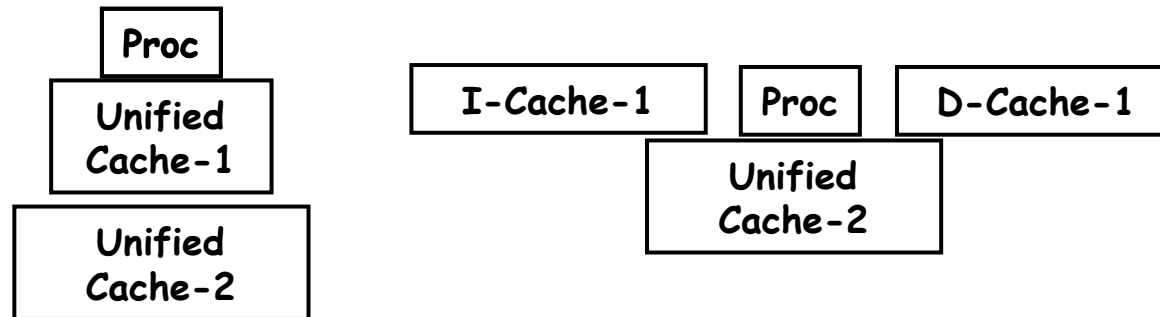
$$\begin{aligned} AMAT &= HitTime + MissRate \times MissPenalty \\ &= (HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst}) + \\ &\quad (HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data}) \end{aligned}$$

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
- $CPI = \text{ideal CPI} + \text{average stalls per instruction}$
$$1.1(\text{cycles/ins}) +$$
$$[0.30 (\text{DataMops/ins})$$
$$\quad \times 0.10 (\text{miss/DataMop}) \times 50 (\text{cycle/miss})] +$$
$$[1 (\text{InstMop/ins})$$
$$\quad \times 0.01 (\text{miss/InstMop}) \times 50 (\text{cycle/miss})]$$
$$= (1.1 + 1.5 + .5) \text{ cycle/ins} = 3.1$$
- 58% of the time the proc is stalled waiting for memory!
- $AMAT = (1/1.3) \times [1 + 0.01 \times 50] + (0.3/1.3) \times [1 + 0.1 \times 50] = 2.54$

Example: Harvard Architecture

- Unified vs Separate I&D (Harvard)



- Table on page 384:
 - 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 \Rightarrow 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)

$$AMAT_{\text{Harvard}} = 75\% \times (1 + 0.64\% \times 50) + 25\% \times (1 + 6.47\% \times 50) = 2.05$$

$$AMAT_{\text{Unified}} = 75\% \times (1 + 1.99\% \times 50) + 25\% \times (1 + 1 + 1.99\% \times 50) = 2.24$$

Review: Four Questions for Memory Hierarchy Designers

- Q1: Where can a block be placed in the upper level?
(Block placement)
 - Fully Associative, Set Associative, Direct Mapped
- Q2: How is a block found if it is in the upper level?
(Block identification)
 - Tag/Block
- Q3: Which block should be replaced on a miss?
(Block replacement)
 - Random, LRU
- Q4: What happens on a write?
(Write strategy)
 - Write Back or Write Through (with Write Buffer)

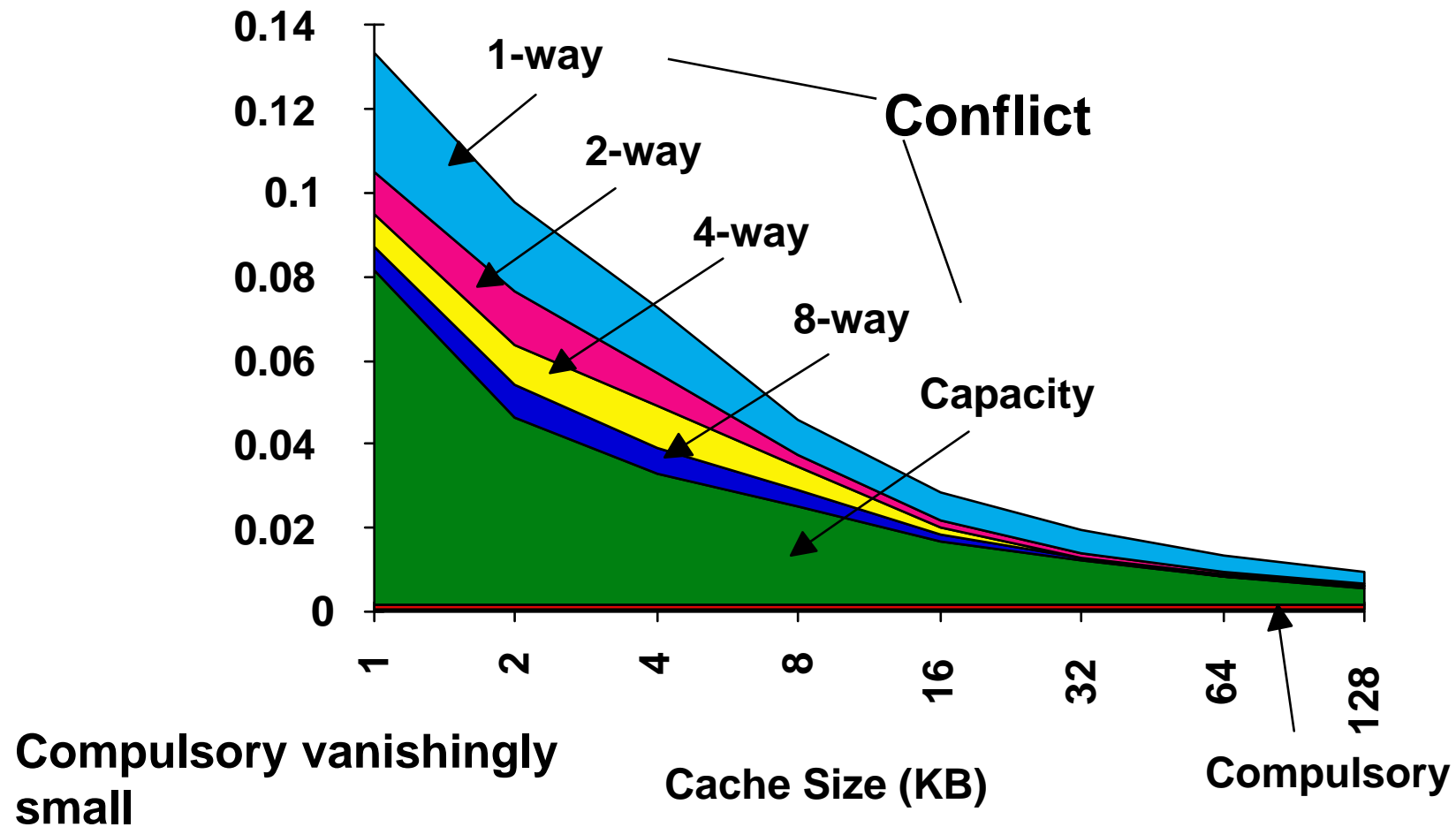
Review: Improving Cache Performance

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

Reducing Misses

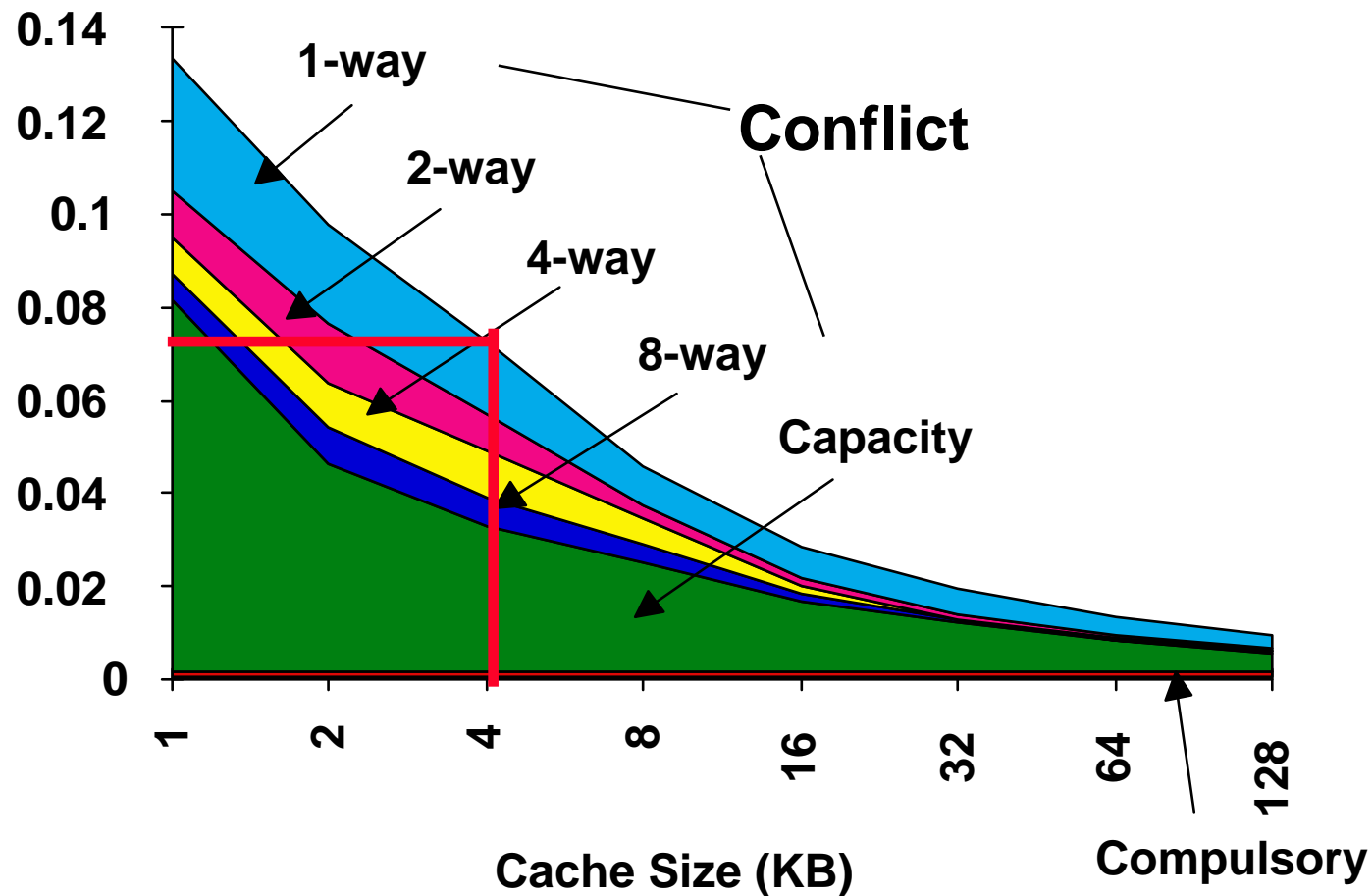
- Classifying Misses: 3 Cs
 - **Compulsory**—The first access to a block is not in the cache, so the block must be brought into the cache. Also called **cold start misses** or **first reference misses**.
(Misses in even an Infinite Cache)
 - **Capacity**—If the cache cannot contain all the blocks needed 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)
- More recent, 4th “C”:
 - **Coherence** - Misses caused by cache coherence.

3Cs Absolute Miss Rate (SPEC92)

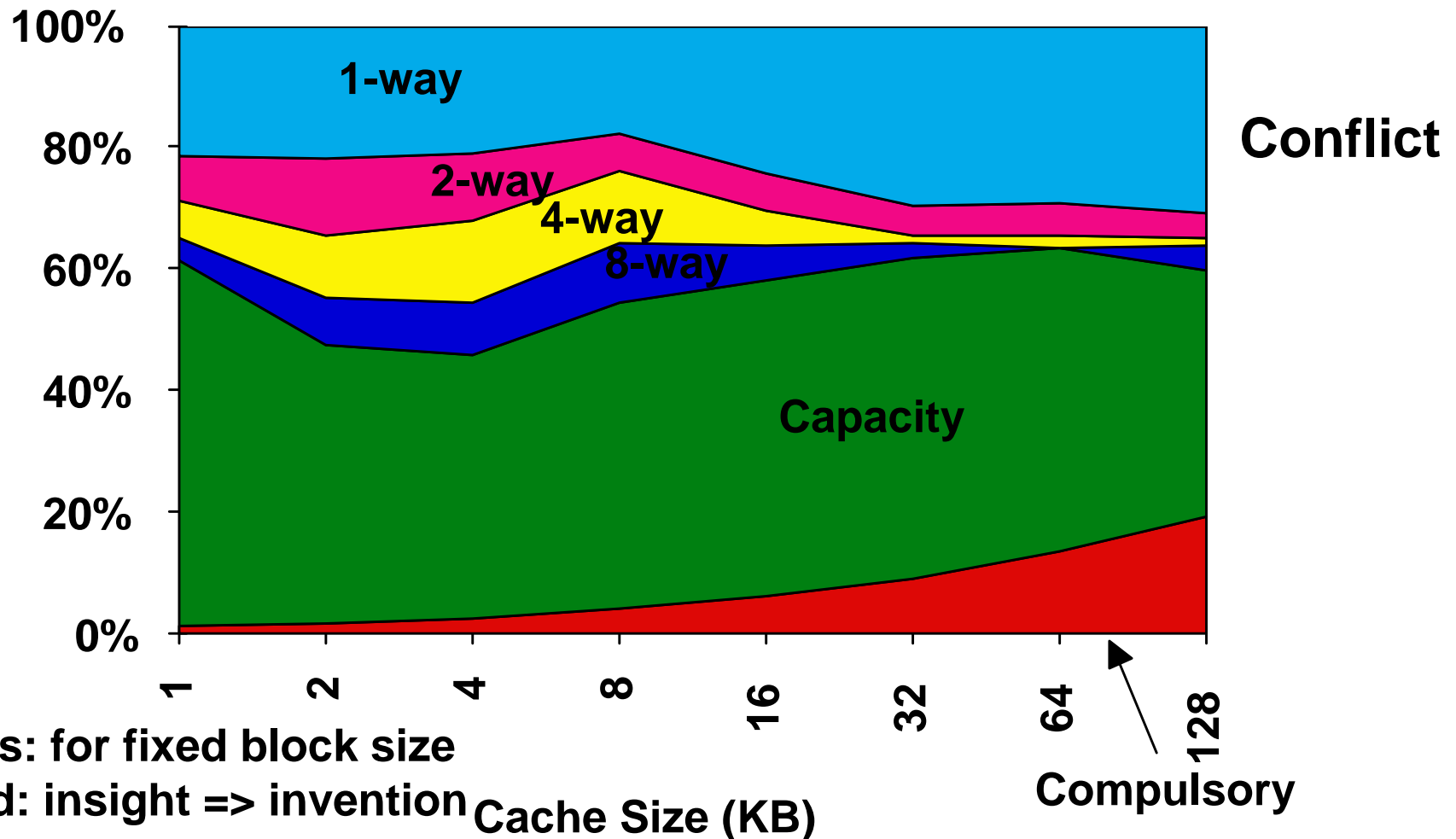


2:1 Cache Rule

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



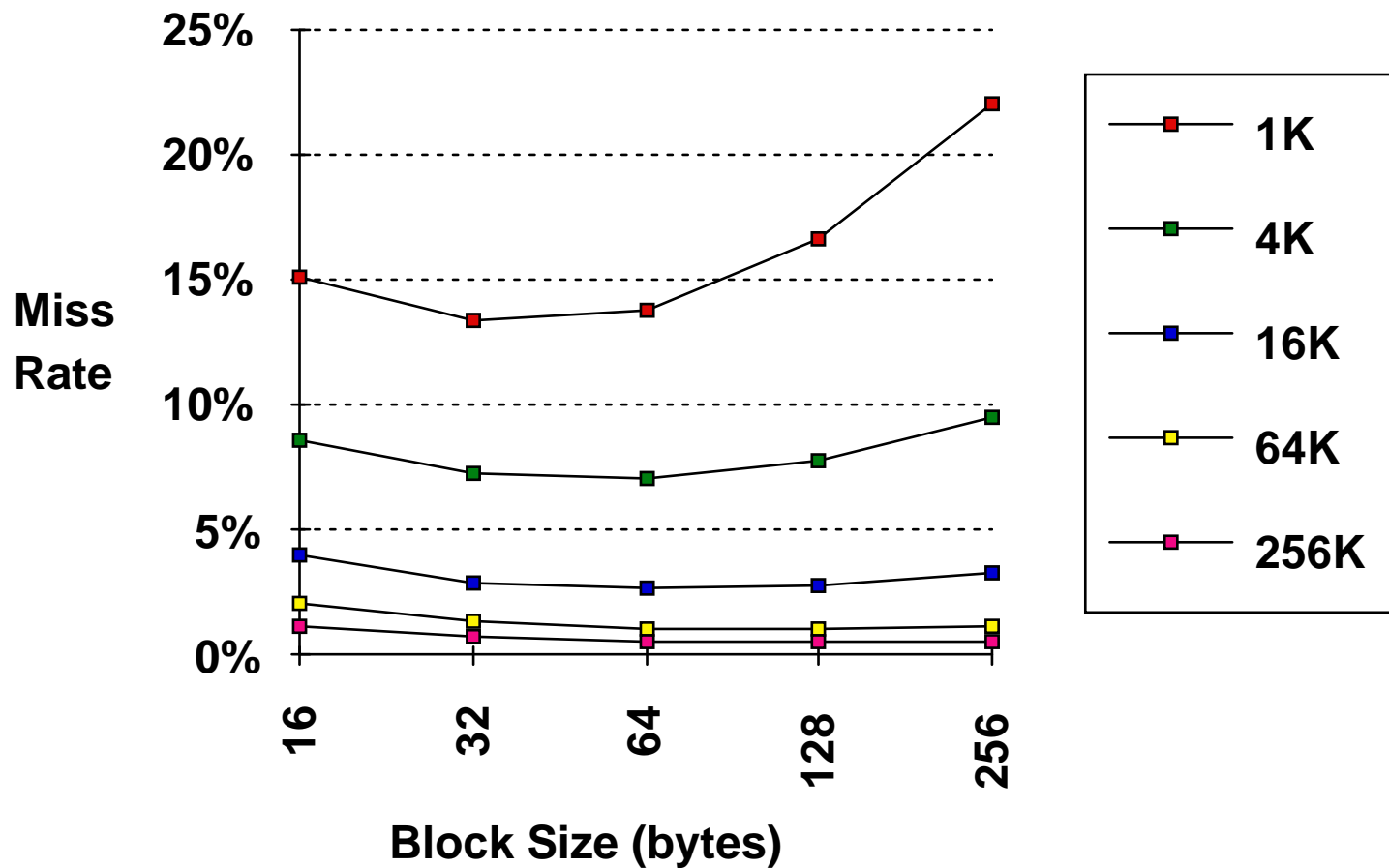
3Cs Relative Miss Rate



How Can Reduce Misses?

- 3 Cs: **Compulsory, Capacity, Conflict**
- In all cases, assume total cache size not changed:
- What happens if:
 - 1) Change Block Size:
Which of 3Cs is obviously affected?
 - 2) Change Associativity:
Which of 3Cs is obviously affected?
 - 3) Change Compiler:
Which of 3Cs is obviously affected?

1. Reduce Misses via Larger Block Size



2. Reduce Misses via Higher Associativity

- **2:1 Cache Rule:**
 - Miss Rate DM cache size N - Miss Rate 2-way cache size $N/2$
- **Beware: Execution time is only final measure!**
 - Will Clock Cycle time increase?
 - Hill [1988] suggested hit time for 2-way vs. 1-way external cache +10%, internal + 2%

Example: Avg. Memory Access Time vs. Miss Rate

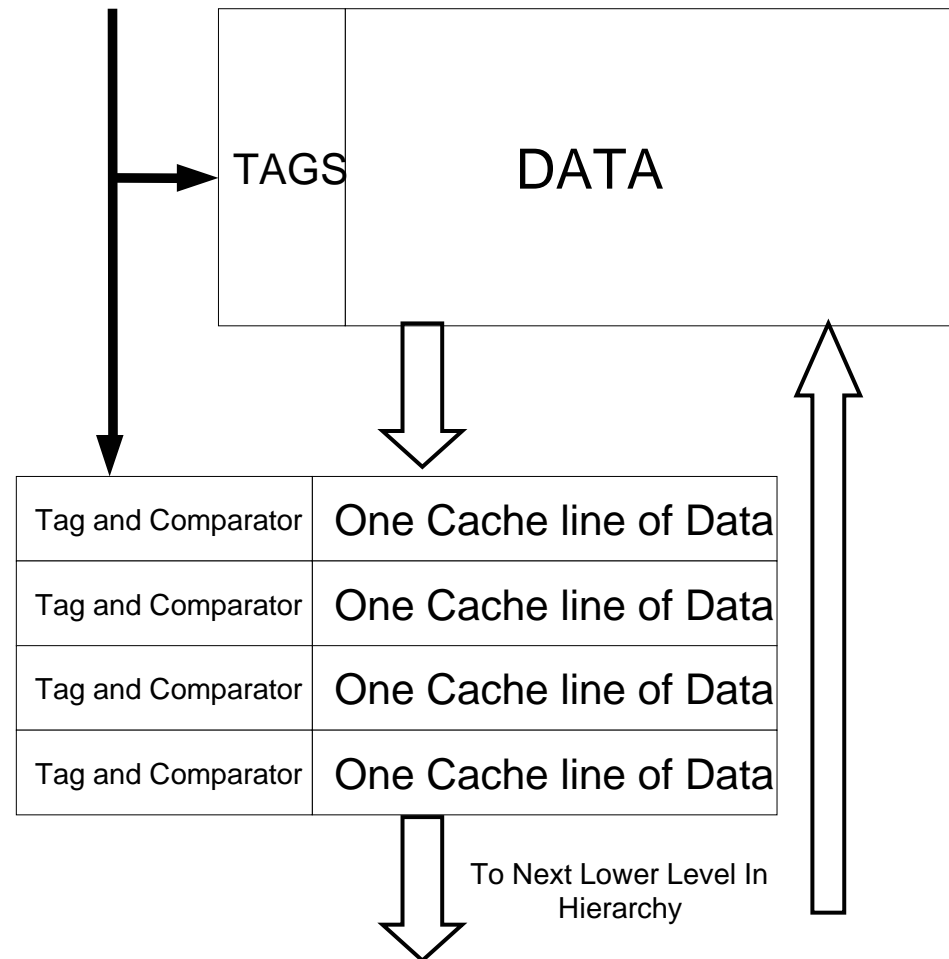
- Example: assume CCT = 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. CCT direct mapped

| Cache Size (KB) | Associativity | | | |
|--------------------|---------------|-------|-------|-------|
| | 1-way | 2-way | 4-way | 8-way |
| 1 | 2.33 | 2.15 | 2.07 | 2.01 |
| 2 | 1.98 | 1.86 | 1.76 | 1.68 |
| 4 | 1.72 | 1.67 | 1.61 | 1.53 |
| 8 | 1.46 | 1.48 | 1.47 | 1.43 |
| 16 | 1.29 | 1.32 | 1.32 | 1.32 |
| 32 | 1.20 | 1.24 | 1.25 | 1.27 |
| 64 | 1.14 | 1.20 | 1.21 | 1.23 |
| 128 | 1.10 | 1.17 | 1.18 | 1.20 |

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

3. Reducing Misses via a "Victim Cache"

- How to combine fast hit time of direct mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Used in Alpha, HP machines



4. Reducing Misses via "Pseudo-Associativity"

- How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache?
- Divide cache: on a miss, check other half of cache to see if there, if so have a pseudo-hit (slow hit)



- Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles
 - Better for caches not tied directly to processor (L2)
 - Used in MIPS R1000 L2 cache, similar in UltraSPARC

5. Reducing Misses by Hardware Prefetching of Instructions & Datals

- E.g., Instruction Prefetching
 - Alpha 21064 fetches 2 blocks on a miss
 - Extra block placed in "stream buffer"
 - On miss check stream buffer
- Works with data blocks too:
 - Jouppi [1990] 1 data stream buffer got 25% misses from 4KB cache; 4 streams got 43%
 - Palacharla & Kessler [1994] for scientific programs for 8 streams got 50% to 70% of misses from 2 64KB, 4-way set associative caches
- Prefetching relies on having extra memory bandwidth that can be used without penalty

6. Reducing Misses by Software Prefetching Data

- Data Prefetch
 - Load data into register (HP PA-RISC loads)
 - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
 - Special prefetching instructions cannot cause faults; a form of speculative execution
- Prefetching comes in two flavors:
 - Binding prefetch: Requests load directly into register.
 - » Must be correct address and register!
 - Non-Binding prefetch: Load into cache.
 - » Can be incorrect. Frees HW/SW to guess!
- Issuing Prefetch Instructions takes time
 - Is cost of prefetch issues < savings in reduced misses?
 - Higher superscalar reduces difficulty of issue bandwidth

7. Reducing Misses by Compiler Optimizations

- McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache, 4 byte blocks in software
- Instructions
 - Reorder procedures in memory so as to reduce conflict misses
 - Profiling to look at conflicts(using tools they developed)
- Data
 - *Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays
 - *Loop Interchange*: change nesting of loops to access data in order stored in memory
 - *Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap
 - *Blocking*: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows


Merging Arrays Example

```
/* Before: 2 sequential arrays */  
int val[SIZE];  
int key[SIZE];  
  
/* After: 1 array of structures */  
struct merge {  
    int val;  
    int key;  
};  
struct merge merged_array[SIZE];
```

Reducing conflicts between val & key;
improve spatial locality

Loop Interchange Example

```
/* Before */  
for (k = 0; k < 100; k = k+1)  
    for (j = 0; j < 100; j = j+1)  
        for (i = 0; i < 5000; i = i+1)  
            x[i][j] = 2 * x[i][j];  
  
/* After */  
for (k = 0; k < 100; k = k+1)  
    for (i = 0; i < 5000; i = i+1)  
        for (j = 0; j < 100; j = j+1)  
            x[i][j] = 2 * x[i][j];
```



Sequential accesses instead of striding
through memory every 100 words; improved
spatial locality

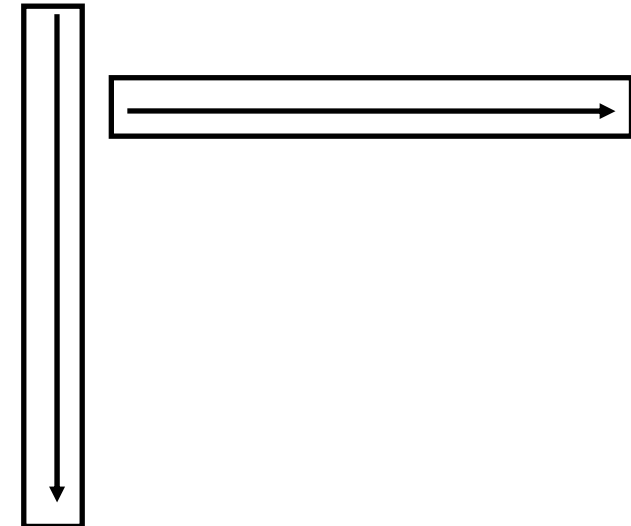
Loop Fusion Example

```
/* Before */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
    {
        a[i][j] = 1/b[i][j] * c[i][j];
        d[i][j] = a[i][j] + c[i][j];
    }
```

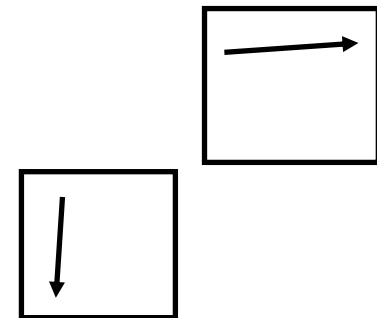
2 misses per access to a & c vs. one miss per access; improve spatial locality

Blocking Example

```
/* Before */  
for (i = 0; i < N; i = i+1)  
  for (j = 0; j < N; j = j+1)  
    {r = 0;  
      for (k = 0; k < N; k = k+1){  
        r = r + y[i][k]*z[k][j];  
      }  
      x[i][j] = r;  
    };
```



- Two Inner Loops:
 - Read all $N \times N$ elements of $z[]$
 - Read N elements of 1 row of $y[]$ repeatedly
 - Write N elements of 1 row of $x[]$
- Capacity Misses a function of N & Cache Size:
 - $2N^3 + N^2 \Rightarrow$ (assuming no conflict; otherwise ...)
- Idea: compute on $B \times B$ submatrix that fits

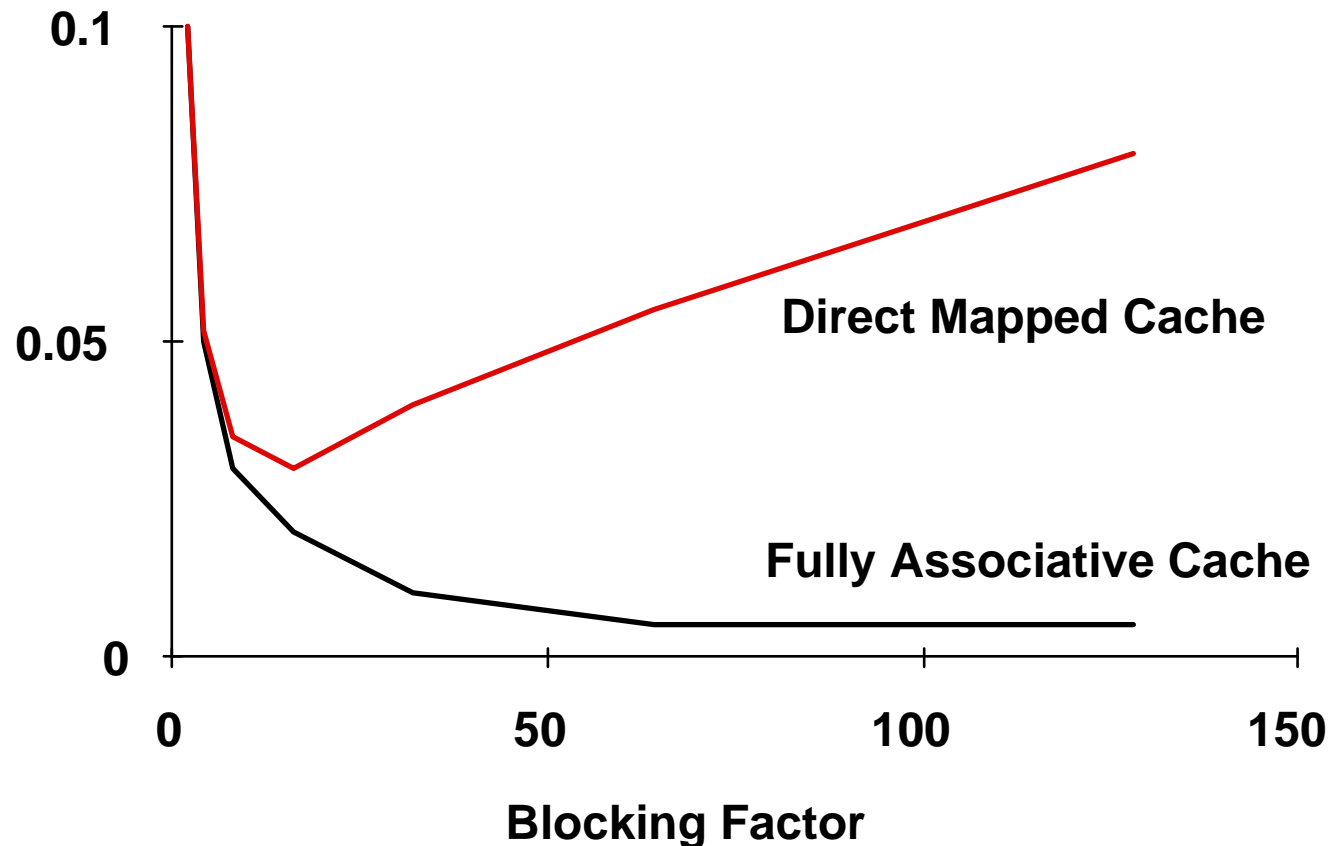


Blocking Example

```
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
    for (j = jj; j < min(jj+B-1,N); j = j+1)
        {r = 0;
         for (k = kk; k < min(kk+B-1,N); k = k+1) {
             r = r + y[i][k]*z[k][j];};
         x[i][j] = x[i][j] + r;
        };
```

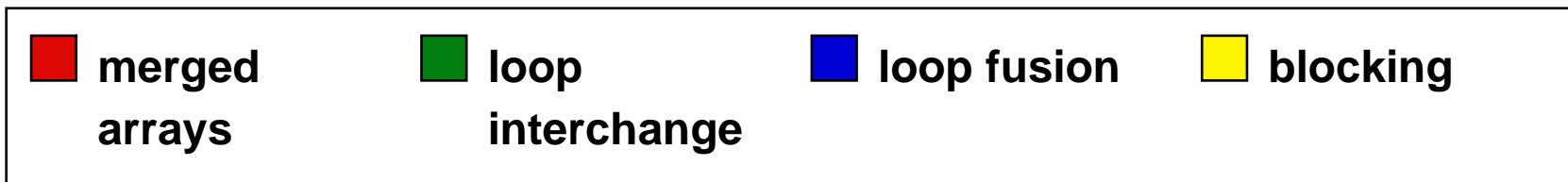
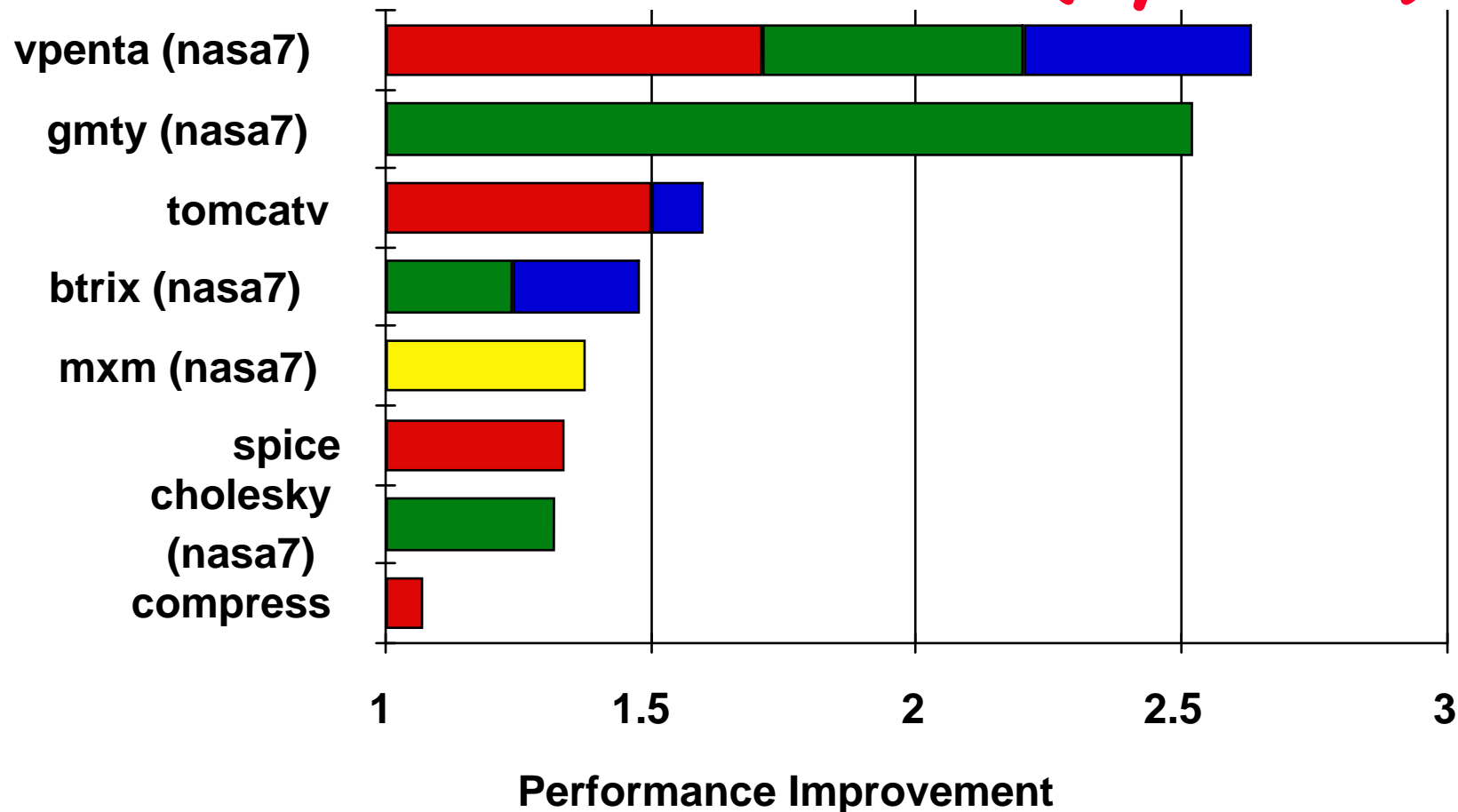
- B called *Blocking Factor*
- Capacity Misses from $2N^3 + N^2$ to $N^3/B + 2N^2$
- Conflict Misses Too?

Reducing Conflict Misses by Blocking



- Conflict misses in caches not FA vs. Blocking size
 - Lam et al [1991] a blocking factor of 24 had a fifth the misses vs. 48 despite both fit in cache

Summary of Compiler Optimizations to Reduce Cache Misses (by hand)



Summary: Miss Rate Reduction

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ accesses}{Instruction} \times \text{Miss rate} \times Miss\ penalty \right) \times Clock\ cycle\ time$$

- **3 Cs: Compulsory, Capacity, Conflict**
 1. Reduce Misses via Larger Block Size
 2. Reduce Misses via Higher Associativity
 3. Reducing Misses via Victim Cache
 4. Reducing Misses via Pseudo-Associativity
 5. Reducing Misses by HW Prefetching Instr, Data
 6. Reducing Misses by SW Prefetching Data
 7. Reducing Misses by Compiler Optimizations
- **Prefetching comes in two flavors:**
 - Binding prefetch: Requests load directly into register.
 - » Must be correct address and register!
 - Non-Binding prefetch: Load into cache.
 - » Can be incorrect. Frees HW/SW to guess!

Review: Improving Cache Performance

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

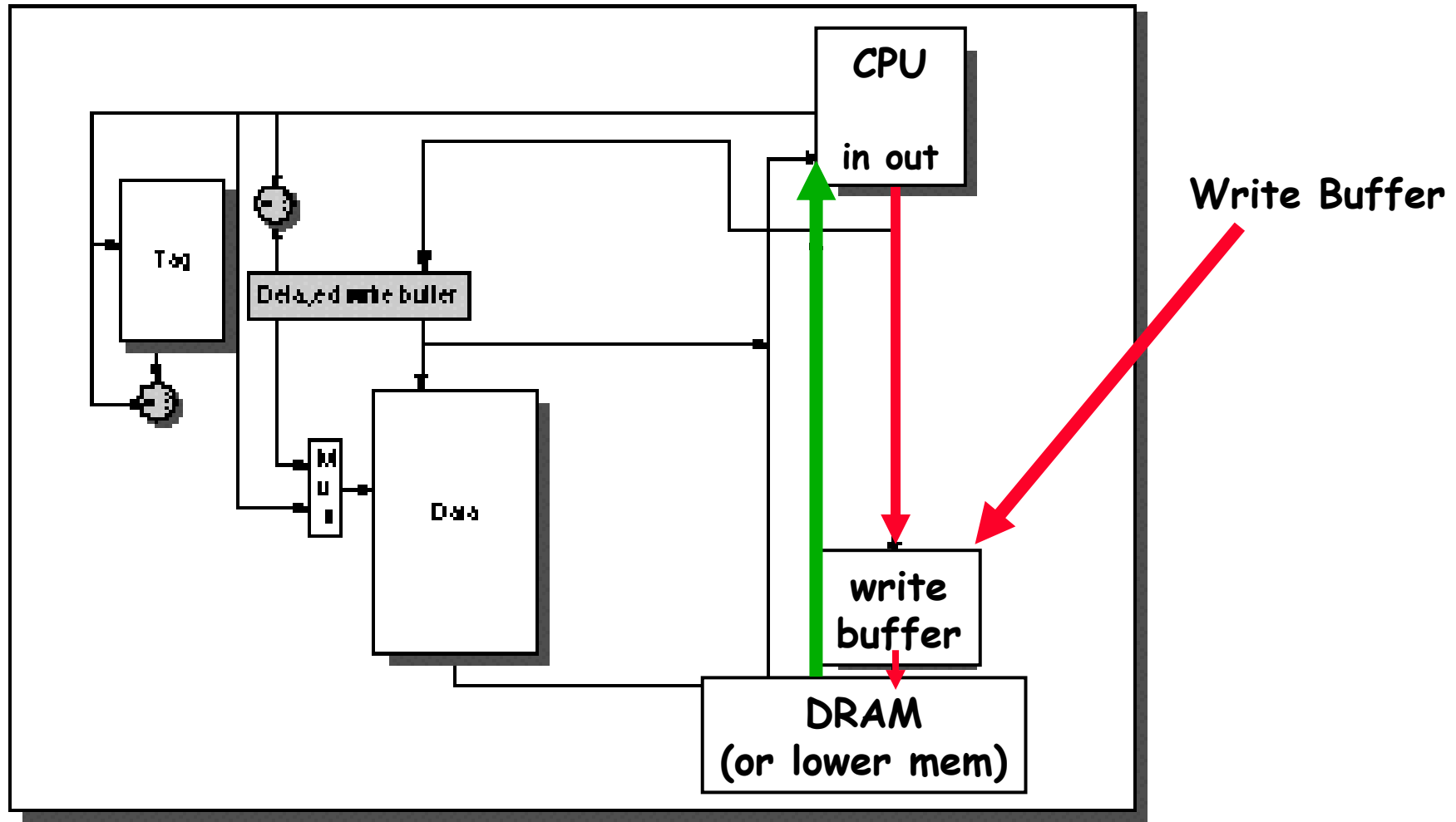
Write Policy: Write-Through vs Write-Back

- **Write-through:** all writes update cache and underlying memory/cache
 - Can always discard cached data - most up-to-date data is in memory
 - Cache control bit: only a *valid* bit
- **Write-back:** all writes simply update cache
 - Can't just discard cached data - may have to write it back to memory
 - Cache control bits: both *valid* and *dirty* bits
- **Other Advantages:**
 - **Write-through:**
 - » memory (or other processors) always have latest data
 - » Simpler management of cache
 - **Write-back:**
 - » much lower bandwidth, since data often overwritten multiple times
 - » Better tolerance to long-latency memory?

Write Policy 2: Write Allocate vs Non-Allocate (What happens on write-miss)

- Write allocate: allocate new cache line in cache
 - Usually means that you have to do a “read miss” to fill in rest of the cache-line!
 - Alternative: per/word valid bits
- Write non-allocate (or “write-around”):
 - Simply send write data through to underlying memory/cache - don't allocate new cache line!

1. Reducing Miss Penalty: Read Priority over Write on Miss



1. Reducing Miss Penalty: Read Priority over Write on Miss

- Write-through with write buffers offer RAW conflicts with main memory reads on cache misses
 - If simply wait for write buffer to empty, might increase read miss penalty (old MIPS 1000 by 50%)
 - Check write buffer contents before read; if no conflicts, let the memory access continue
- Write-back also want buffer to hold misplaced blocks
 - Read miss replacing dirty block
 - Normal: Write dirty block to memory, and then do the read
 - Instead copy the dirty block to a write buffer, then do the read, and then do the write
 - CPU stall less since restarts as soon as do read

2. Reduce Miss Penalty: Early Restart and Critical Word First

- Don't wait for full block to be loaded before restarting CPU
 - Early restart—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
 - Critical Word First—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called *wrapped fetch* and *requested word first*
- Generally useful only in large blocks,
- Spatial locality a problem; tend to want next sequential word, so not clear if benefit by early restart



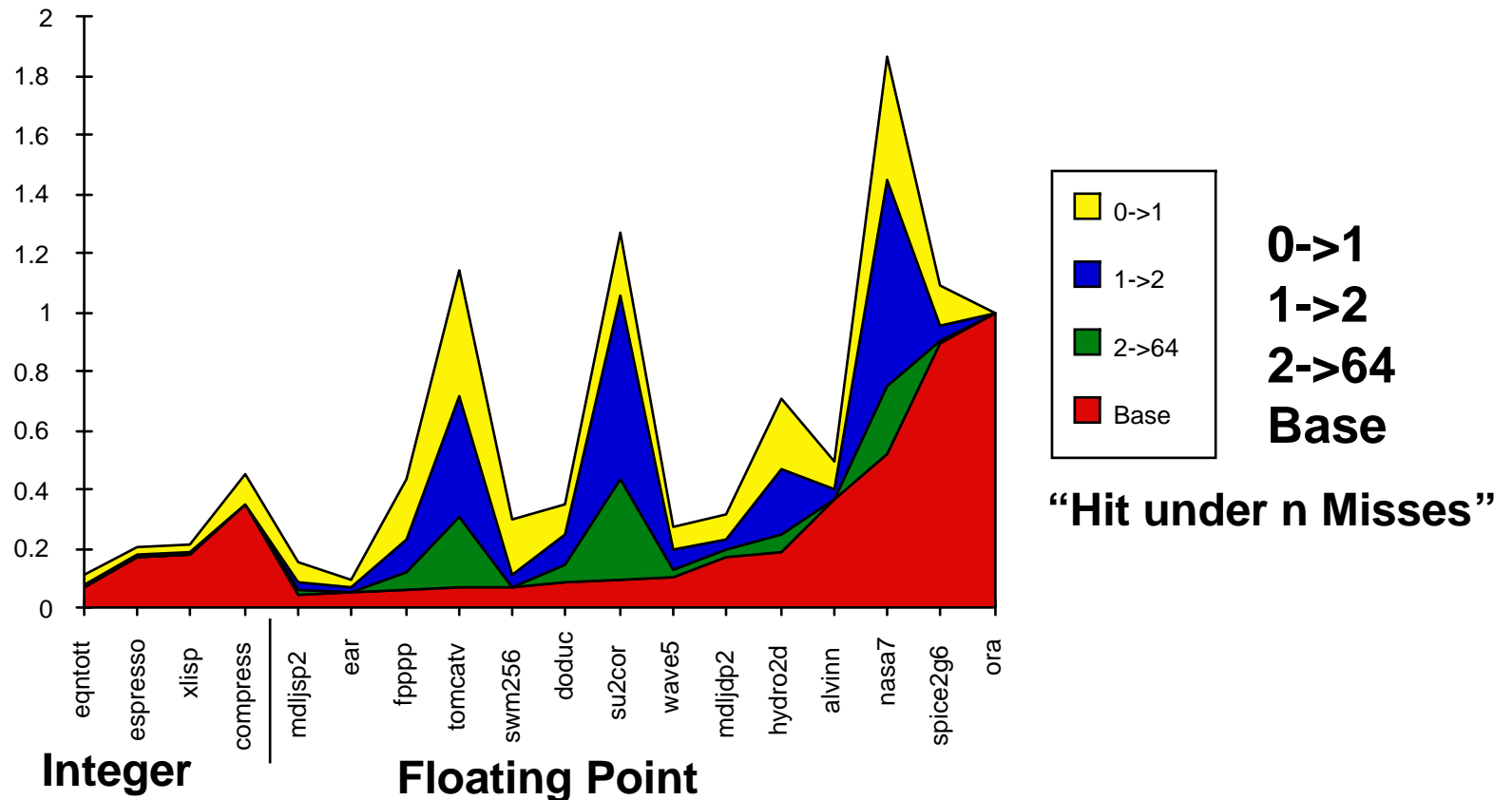
block

3. Reduce Miss Penalty: Non-blocking Caches to reduce stalls on misses

- Non-blocking cache or lockup-free cache allow data cache to continue to supply cache hits during a miss
 - requires F/E bits on registers or out-of-order execution
 - requires multi-bank memories
- “hit under miss” reduces the effective miss penalty by working during miss vs. ignoring CPU requests
- “hit under multiple miss” or “miss under miss” may further lower the effective miss penalty by overlapping multiple misses
 - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
 - Requires multiple memory banks (otherwise cannot support)
 - Pentium Pro allows 4 outstanding memory misses

Value of Hit Under Miss for SPEC

Hit Under i Misses



- FP programs on average: $AMAT = 0.68 \rightarrow 0.52 \rightarrow 0.34 \rightarrow 0.26$
- Int programs on average: $AMAT = 0.24 \rightarrow 0.20 \rightarrow 0.19 \rightarrow 0.19$
- 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss

4: Add a second-level cache

- L2 Equations

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

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

$$AMAT = \text{Hit Time}_{L1} +$$

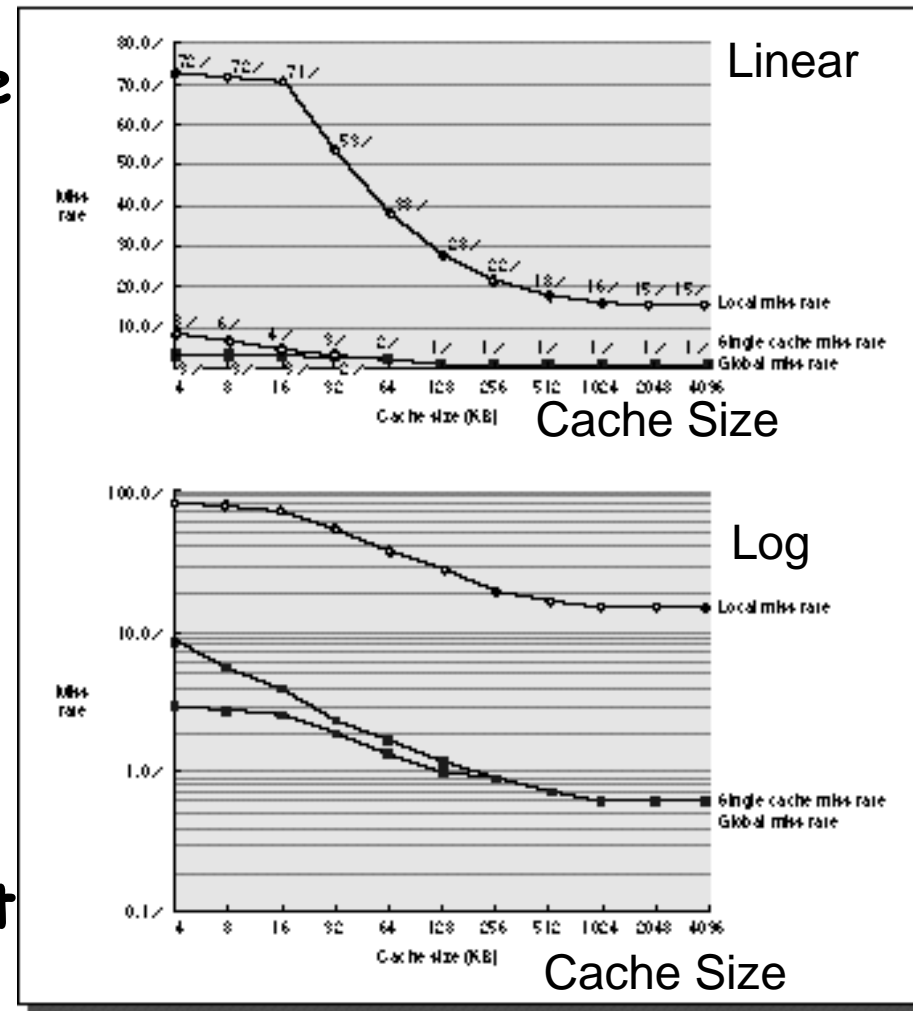
$$\text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})$$

- Definitions:

- *Local miss rate*— misses in this cache divided by the total number of memory accesses *to this cache* (Miss rate_{L2})
- *Global miss rate*— misses in this cache divided by the total number of memory accesses *generated by the CPU* ($\text{Miss Rate}_{L1} \times \text{Miss Rate}_{L2}$)
- Global Miss Rate is what matters

Comparing Local and Global Miss Rates

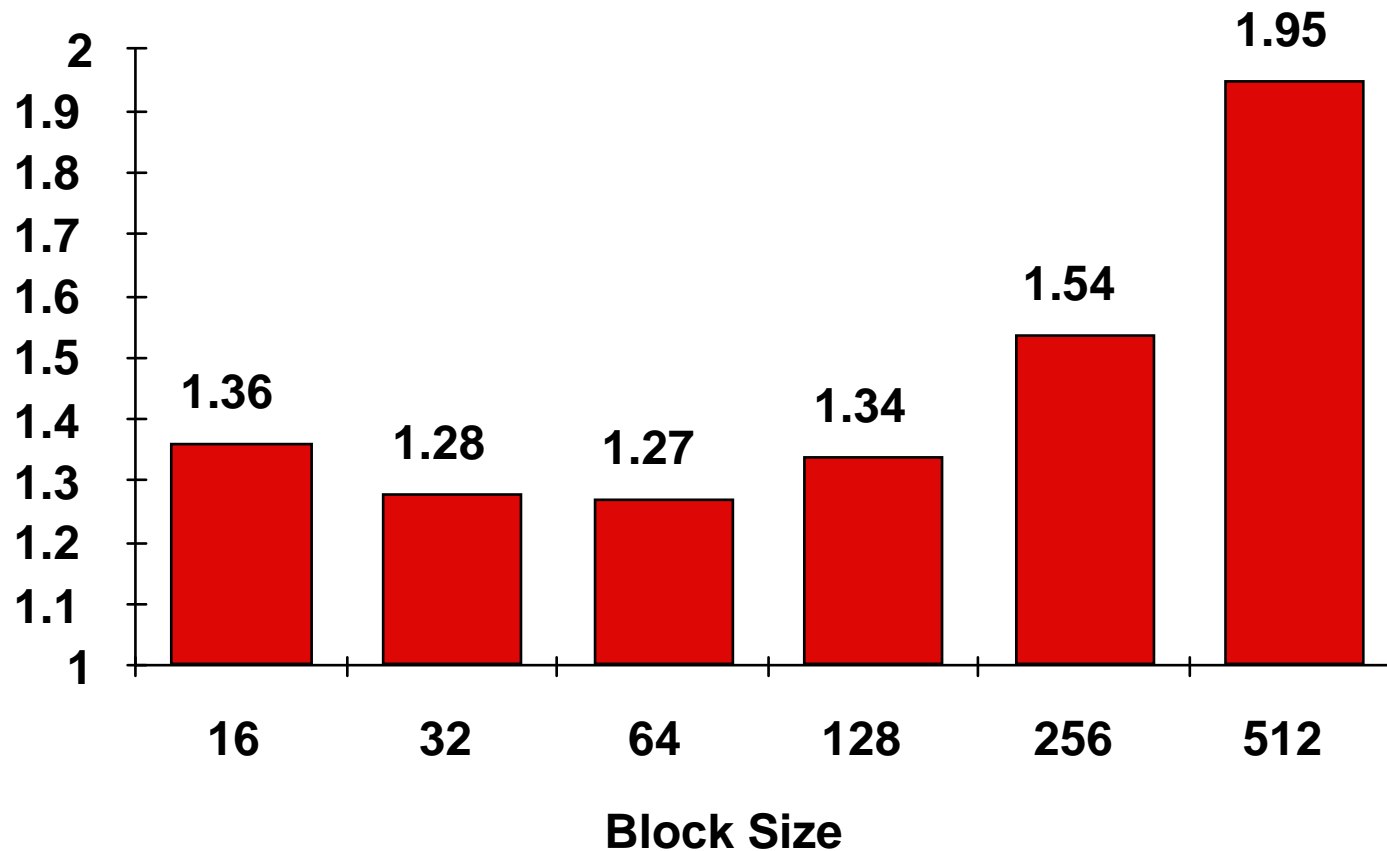
- 32 KByte 1st level cache; Increasing 2nd level cache
- Global miss rate close to single level cache rate provided $L2 \gg L1$
- Don't use local miss rate
- L2 not tied to CPU clock cycle!
- Cost & A.M.A.T.
- Generally Fast Hit Times and fewer misses
- Since hits are few, target miss reduction



Reducing Misses: Which apply to L2 Cache?

- Reducing Miss Rate
 1. Reduce Misses via Larger Block Size
 2. Reduce Conflict Misses via Higher Associativity
 3. Reducing Conflict Misses via Victim Cache
 4. Reducing Conflict Misses via Pseudo-Associativity
 5. Reducing Misses by HW Prefetching Instr, Data
 6. Reducing Misses by SW Prefetching Data
 7. Reducing Capacity/Conf. Misses by Compiler Optimizations

L2 cache block size & A.M.A.T. Relative CPU Time



- 32KB L1, 8 byte path to memory

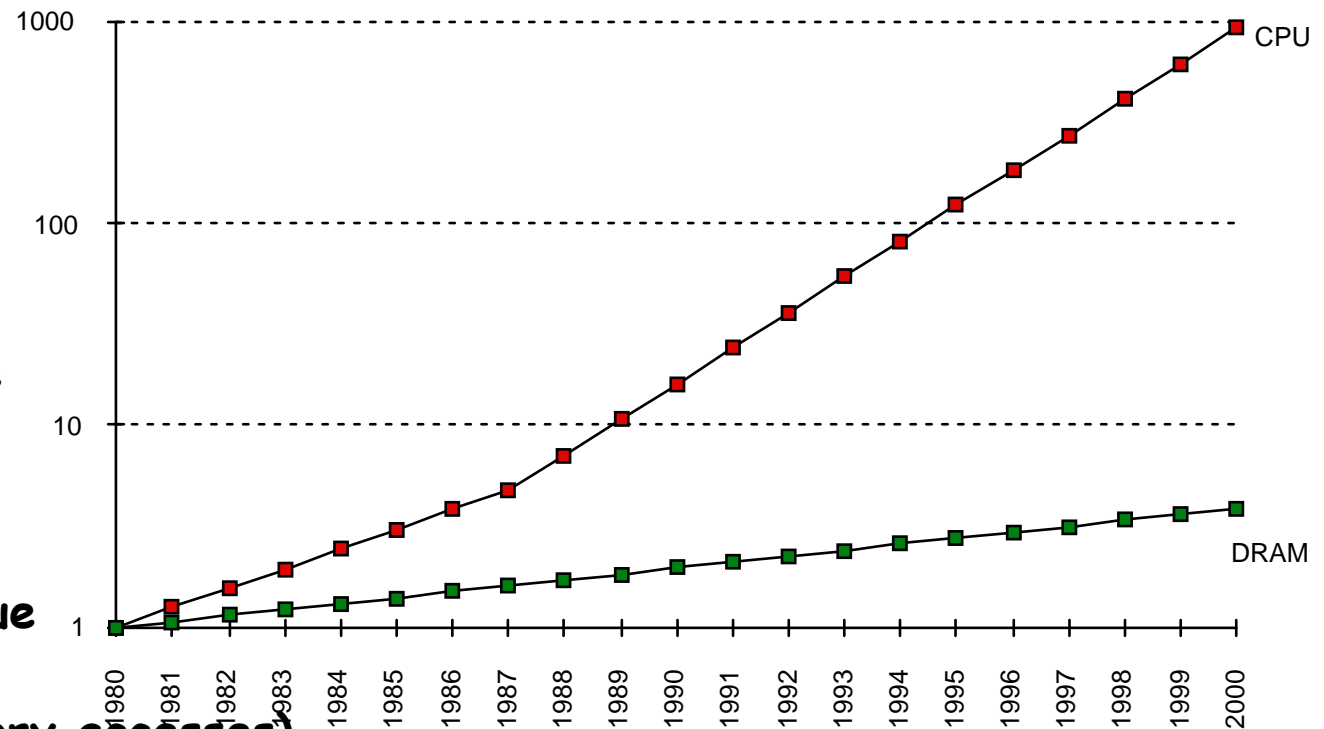
Reducing Miss Penalty Summary

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ accesses}{Instruction} \times Miss\ rate \times Miss\ penalty \right) \times Clock\ cycle\ time$$

- **Four techniques**
 - Read priority over write on miss
 - Early Restart and Critical Word First on miss
 - Non-blocking Caches (Hit under Miss, Miss under Miss)
 - Second Level Cache
- **Can be applied recursively to Multilevel Caches**
 - Danger is that time to DRAM will grow with multiple levels in between
 - First attempts at L2 caches can make things worse, since increased worst case is worse

What is the Impact of What You've Learned About Caches?

- 1960-1985: Speed = $f(\text{no. operations})$
- 1990
 - Pipelined Execution & Fast Clock Rate
 - Out-of-Order execution
 - Superscalar Instruction Issue



- 1998: Speed = $f(\text{non-cached memory accesses})$
- Superscalar, Out-of-Order machines hide L1 data cache miss (-5 clocks) but not L2 cache miss (-50 clocks)?

Cache Optimization Summary

| | <i>Technique</i> | <i>MR</i> | <i>MP</i> | <i>HT</i> | <i>Complexity</i> |
|--------------|-----------------------------------|-----------|-----------|-----------|-------------------|
| miss rate | Larger Block Size | + | - | | 0 |
| | Higher Associativity | + | | - | 1 |
| | Victim Caches | + | | | 2 |
| | Pseudo-Associative Caches | + | | | 2 |
| | HW Prefetching of Instr/Data | + | | | 2 |
| | Compiler Controlled Prefetching | + | | | 3 |
| | Compiler Reduce Misses | + | | | 0 |
| miss penalty | Priority to Read Misses | | + | | 1 |
| | Early Restart & Critical Word 1st | | + | | 2 |
| | Non-Blocking Caches | | + | | 3 |
| | Second Level Caches | | + | | 2 |