Memory Hierarchy
3 Cs
and
6 Ways to Reduce Misses

Soner Onder

Michigan Technological University

Randy Katz & David A. Patterson

University of California, Berkeley

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)

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

Cache Performance

CPUtime = Instruction Count x ($CPI_{execution}$ + Mem accesses per instruction x Miss rate x Miss penalty) x Clock cycle time

Misses per instruction = Memory accesses per instruction x Miss rate

CPUtime = IC x ($CPI_{execution}$ + Misses per instruction x Miss penalty) x Clock cycle time

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*.
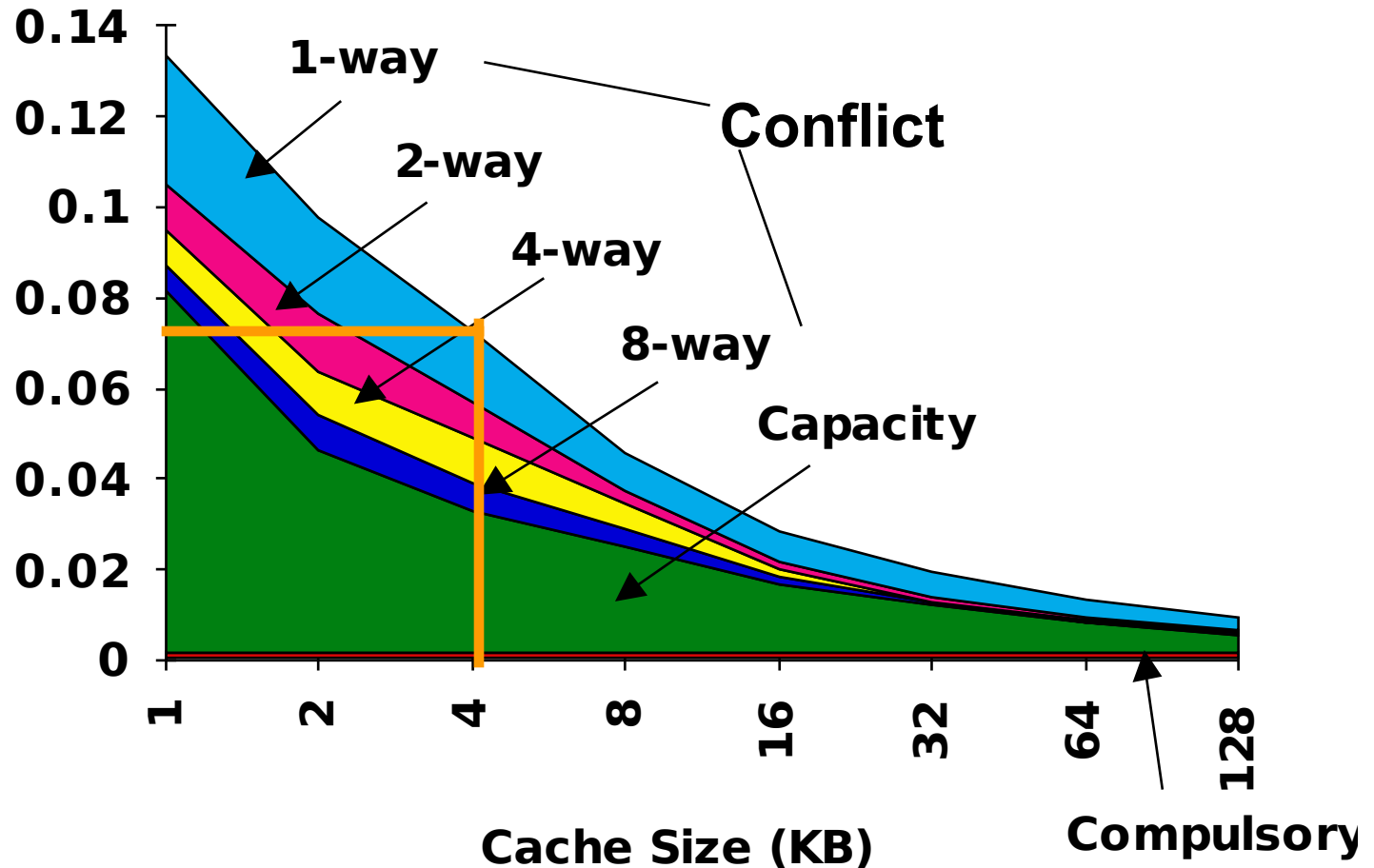*(Misses in N-way Associative, Size X Cache)*
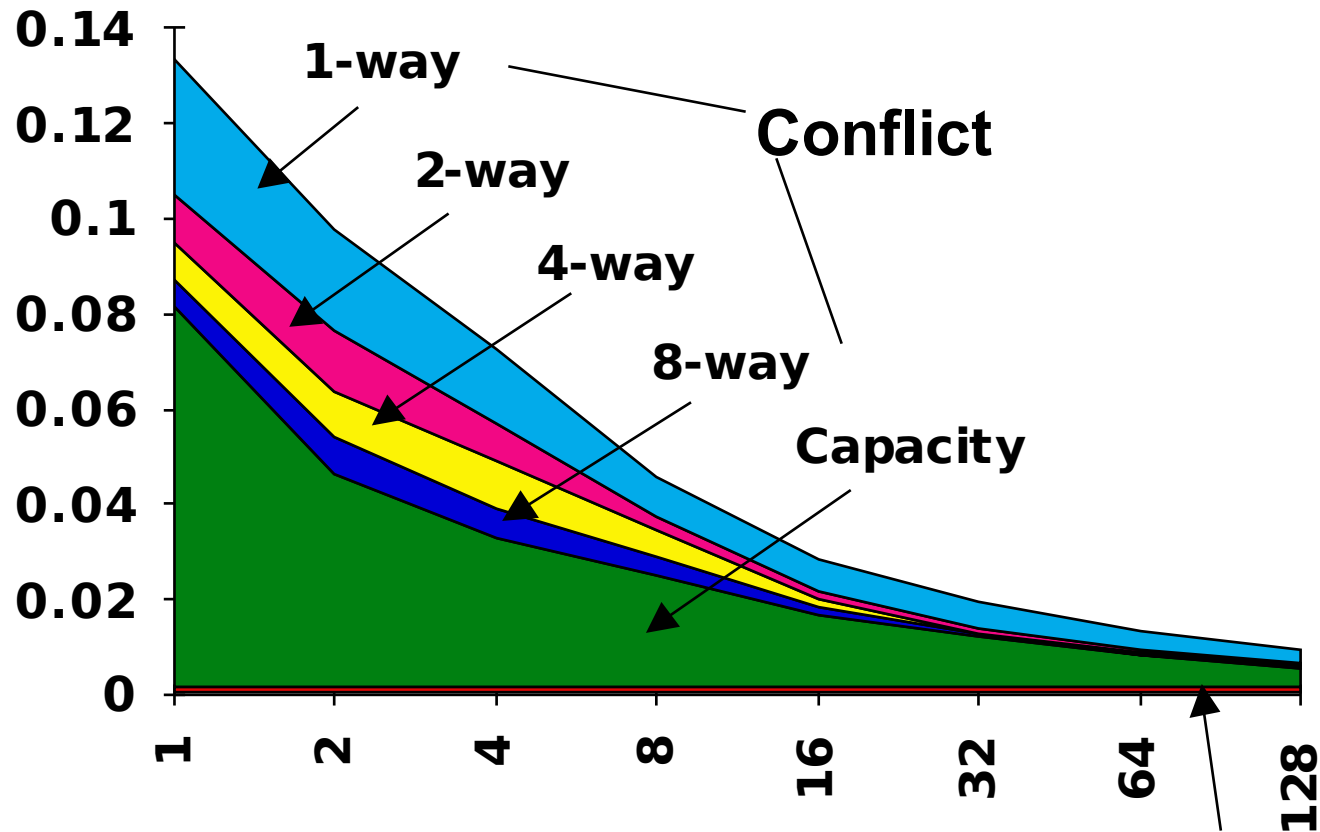
Cache Size (KB)

Compulsory vanishingly small

Compulsory

2:1 Cache Rule

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

1-way **2-way** **4-way** **8-way**

Conflict

Capacity

Compulsory

Cache Size (KB)

3Cs Relative Miss Rate
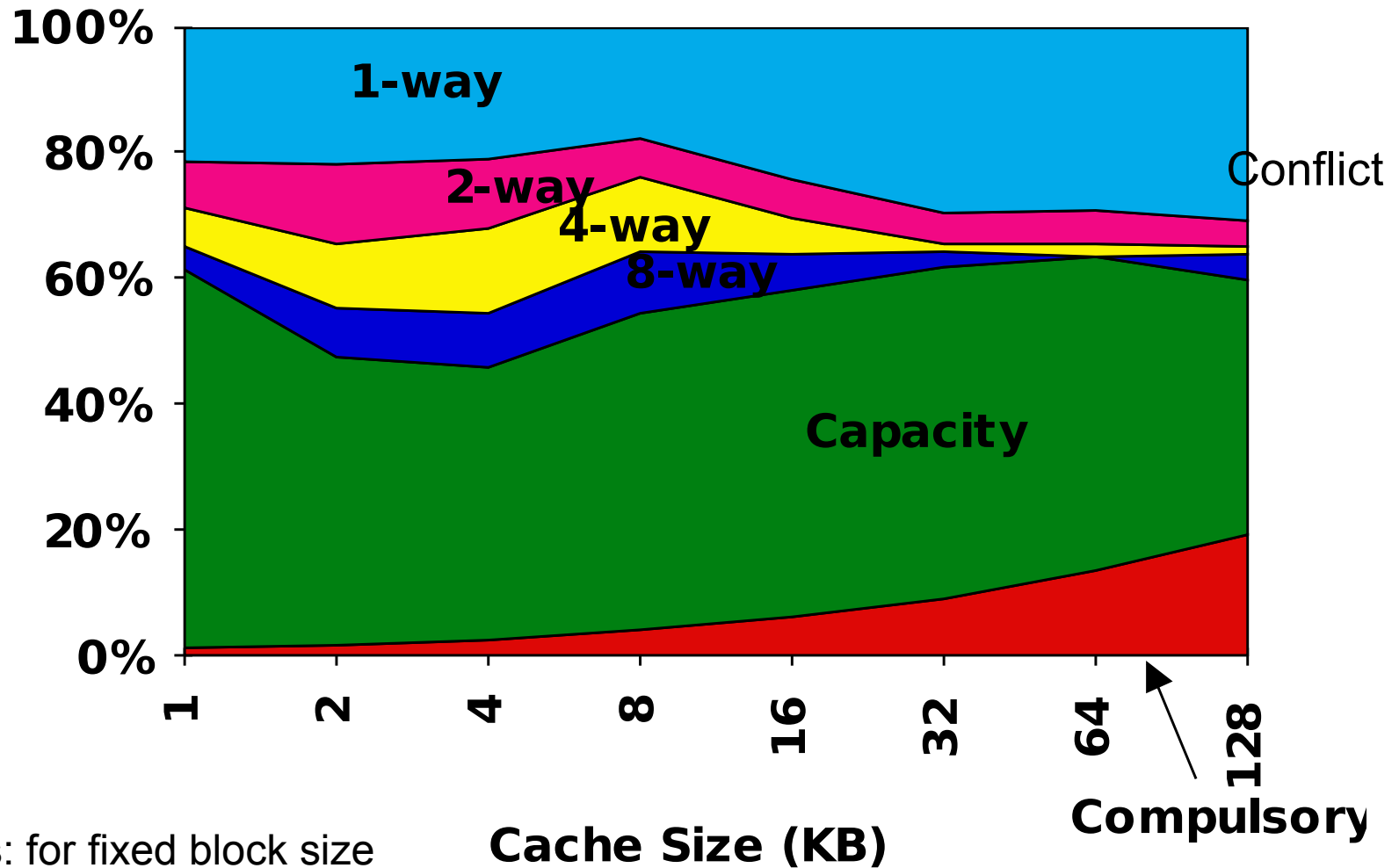
Flaws: for fixed block size
Good: insight => invention

How Can Reduce Misses?

3 Cs: Compulsory, Capacity, Conflict

In all cases, assume total cache size not changed:

What happens if:
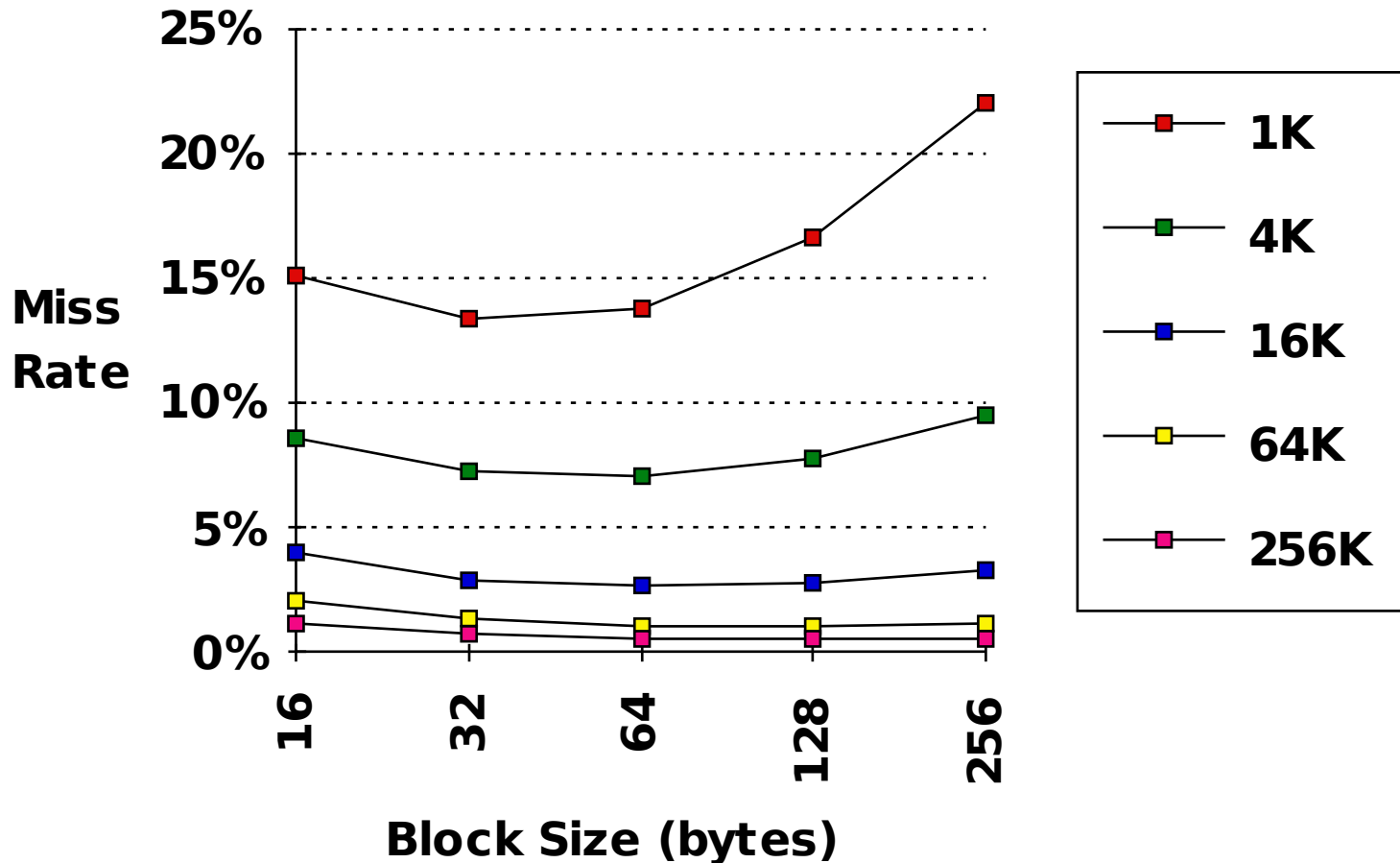
1) Change Block Size:
Which of 3Cs is obviously affected?

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

Effect of Block size on Average Memory Access time

Block Size	Miss Penalty	Cache Size			
		4K	16K	64K	256K
16	82	8.027	4.231	2.673	1.894
32	84	7.082	3.411	2.134	1.588
64	88	7.160	3.323	1.933	1.449
128	96	8.469	3.659	1.979	1.470
256	112	11.651	5.685	2.288	1.549

Block sizes 32 and 64 bytes dominate
Longer hit times?
Higher cost?

2. Make Caches Bigger

Bigger caches have lower miss rates.

Bigger caches cost more.

Bigger caches are slower to access.

It is the average memory access time and the cost of the cache that ultimately determines the cache size.

3. Reduce Misses via Higher Associativity

2:1 Cache Rule:

- **Miss Rate Direct Mapped 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 Associativity

Example: assume CCT = 1.36 for 2-way, 1.44 for 4-way, 1.52 for 8-way vs. CCT direct mapped. Miss penalty is 25 cycles.

AVG-Memory access time = hit time + miss rate x miss penalty.

| Cache Size	1-way	2-way	4-way	8-way
4	3.44	3.25	3.22	3.28
8	2.69	2.58	2.55	2.62
16	2.23	2.40	2.46	2.53
32	2.06	2.30	2.37	2.45
64	1.92	2.14	2.18	2.25
128	1.52	1.84	1.92	2.00
256	1.32	1.66	1.74	1.82
512	1.20	1.55	1.59	1.66

4. 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% conflicts for a 4 KB direct mapped data cache

Used in Alpha, HP machines
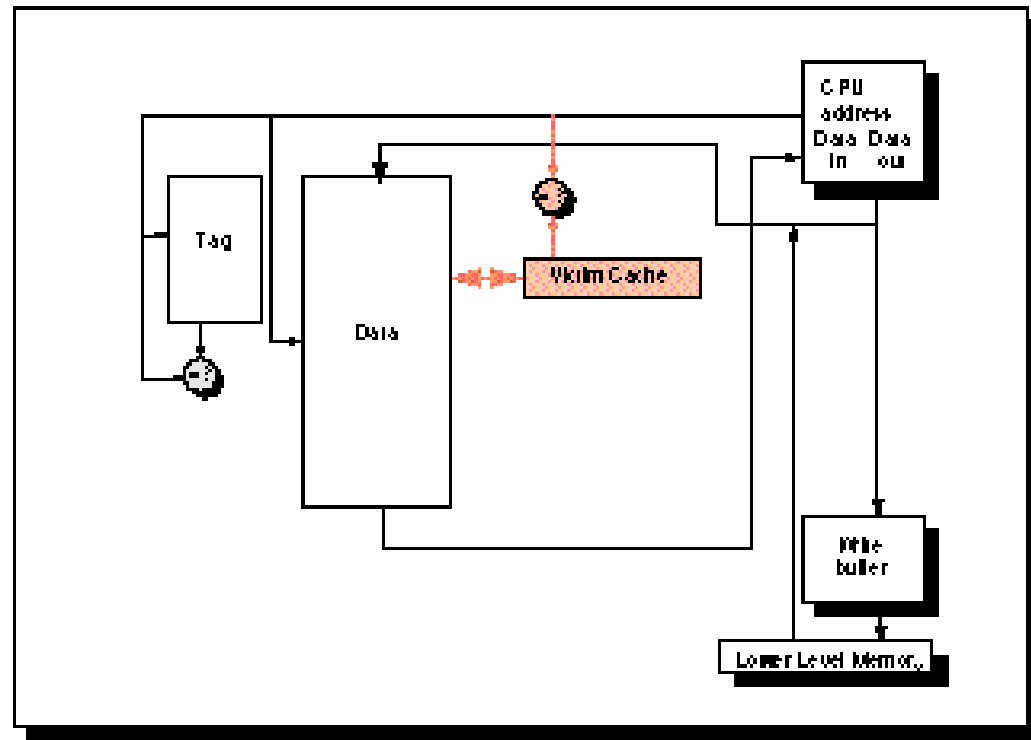
5. Reducing Misses via "Pseudo-Associativity"

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)

Hit Time

Pseudo Hit Time   **Miss Penalty**

Time

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

6. Reducing Misses by Compiler Optimizations

McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache, 4 byte blocks <u>in software</u>

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 the 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 stuctures */

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
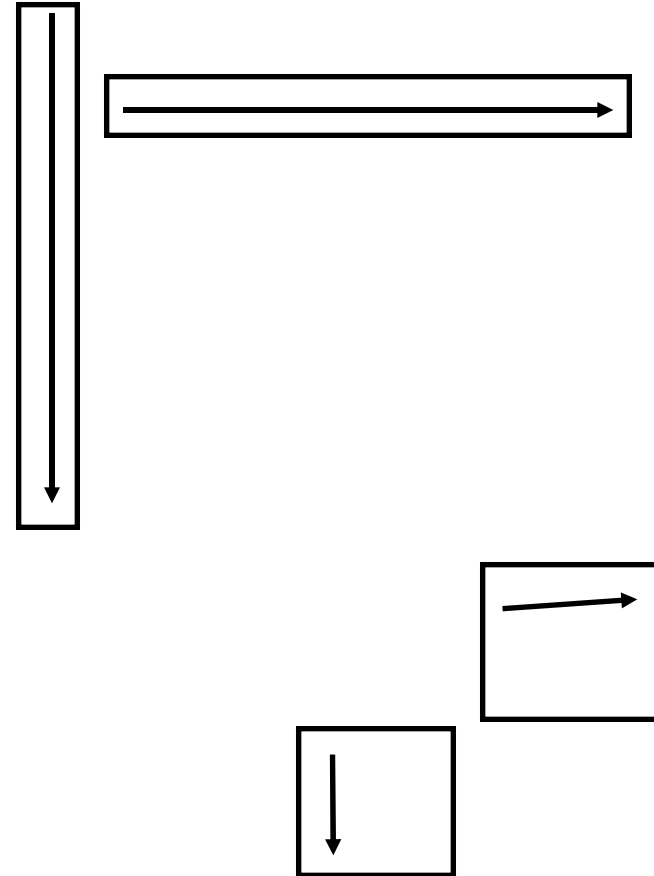
Loop Fusion Example

```
/* 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 NxN 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:

- 3 NxNx4 => no capacity misses; otherwise ...

Idea: compute on BxB 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;

      };
```
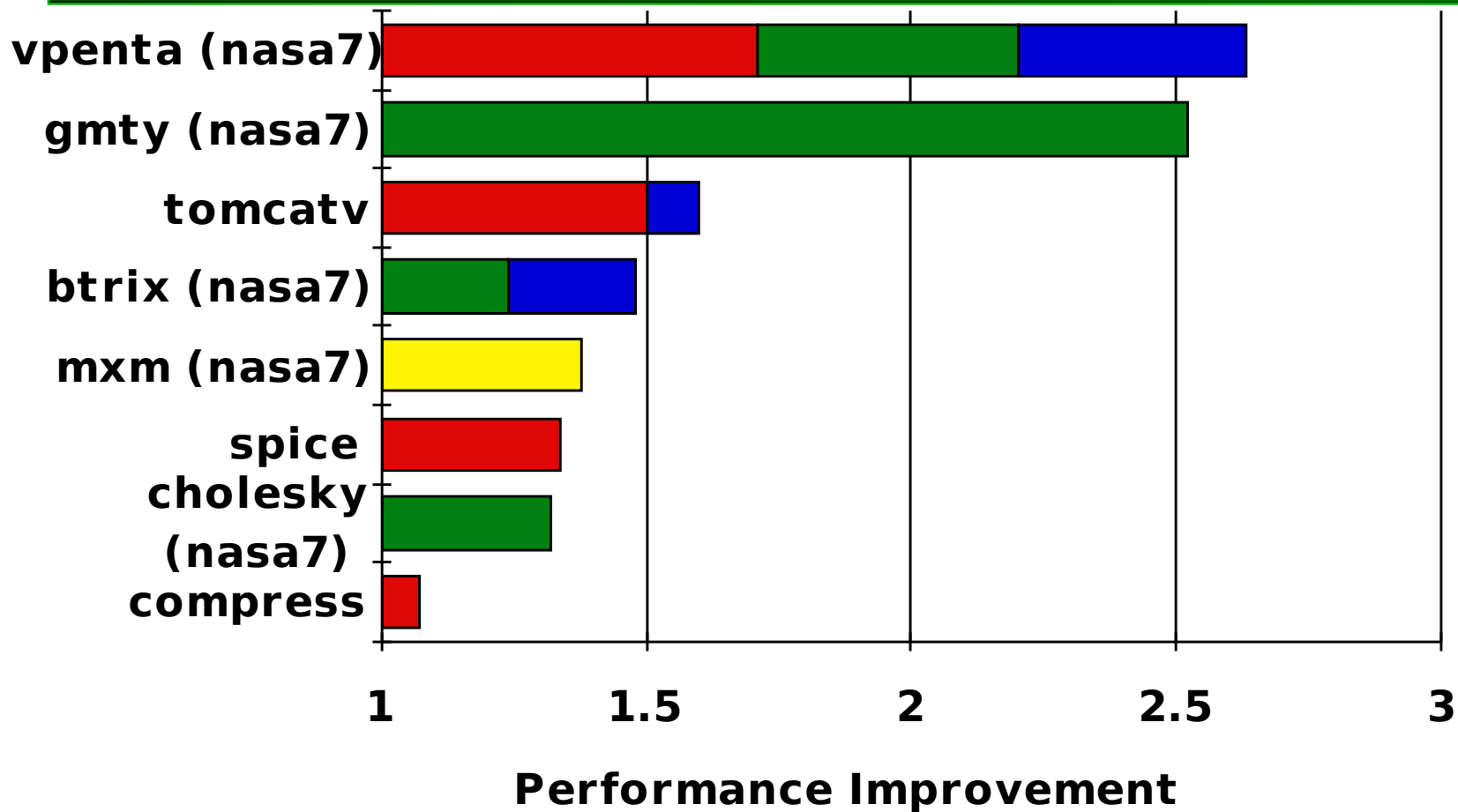
B called *Blocking Factor*

Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$

Conflict Misses Too?

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

Performance Improvement

Legend:
- **merged arrays** (red)
- **loop interchange** (green)
- **loop fusion** (blue)
- **blocking** (yellow)

Benchmarks (top to bottom):
- vpenta (nasa7)
- gmty (nasa7)
- tomcatv
- btrix (nasa7)
- mxm (nasa7)
- spice
- cholesky (nasa7)
- compress

X-axis: 1, 1.5, 2, 2.5, 3

Summary

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

3 Cs: Compulsory, Capacity, Conflict

1. Reduce Misses via Larger Block Size

2. Make caches bigger

3. Reduce Misses via Higher Associativity

4. Reducing Misses via Victim Cache

5. Reducing Misses via Pseudo-Associativity

6. Reducing Misses by Compiler Optimizations

Remember danger of concentrating on just one parameter when evaluating performance
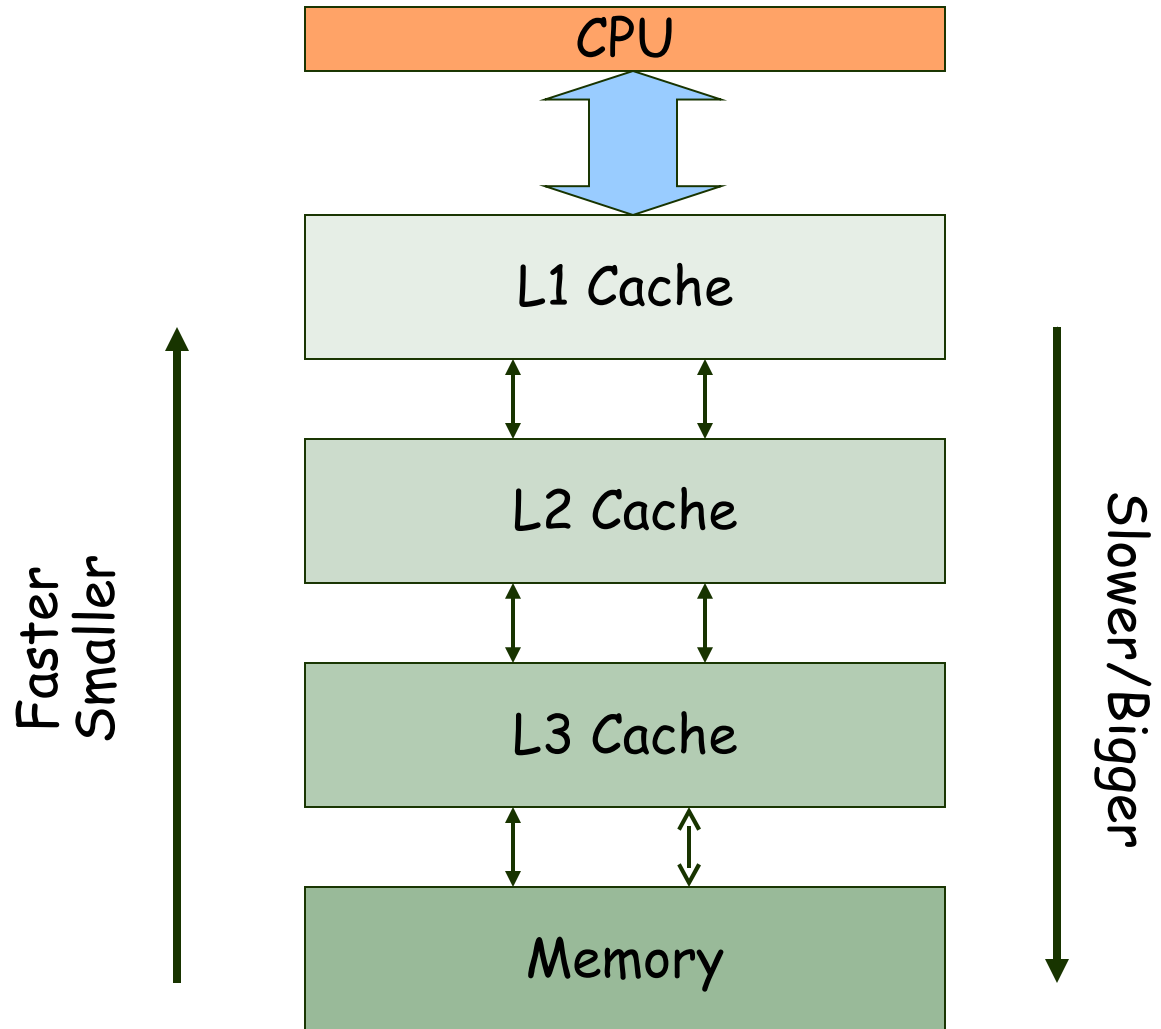
Review: Improving Cache Performance

1. Reduce the miss rate,

2. *Reduce the miss penalty,* or

3. Reduce the time to hit in the cache.

1. Reduce Miss Penalty with multi-level caches

A multi-level cache reduces the miss penalty :

Miss penalty for each level is smaller as we go up.

CPU

L1 Cache

L2 Cache

L3 Cache

Memory

Faster Smaller

Slower/Bigger

Multi-level caches - Equations

L2 Equations

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

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

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

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* (Miss Rate$_{L1}$ × Miss Rate$_{L2}$)

- Global Miss Rate is what matters

Comparing Local and Global Miss Rates

32 KByte 1st level cache; Increasing 2nd level cache

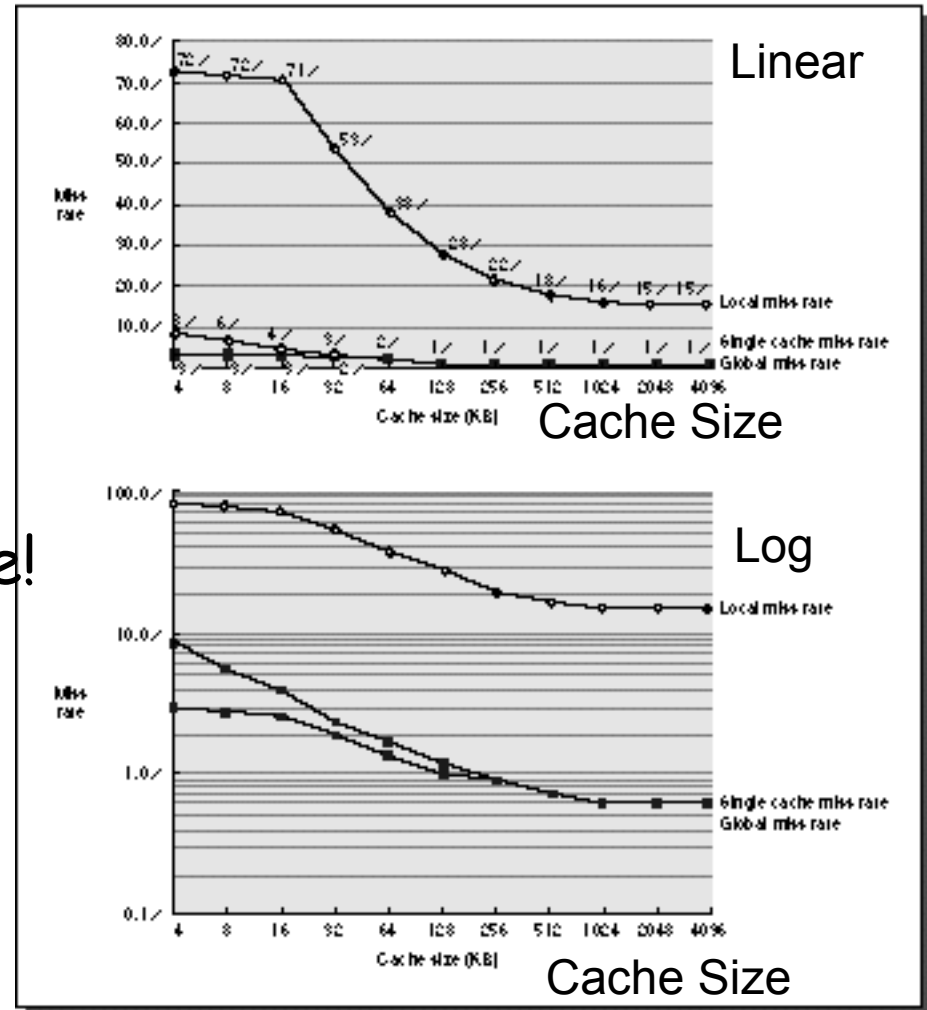Global miss rate close to single level cache rate provided L2 >> L1

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

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. Reducing Miss Penalty: Read Priority over Write on Miss

Write through with write buffers offer RAW conflicts with main memory reads on cache misses:

Write buffers may hold the updated value that is needed on cache miss.

SW r3,512(R0)   (Cache index 0)

LW r1,1024(R0)   (Cache index 0)

LW r2,512(R0)   (Cache index 0)

Is r2 = r3 ?

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

If we simply wait for write buffer to empty, we may 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?

- 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

4. Reduce Miss Penalty: Subblock Placement

Don't have to load full block on a miss

Have valid bits per subblock to indicate valid

(Originally invented to reduce tag storage)

Valid Bits

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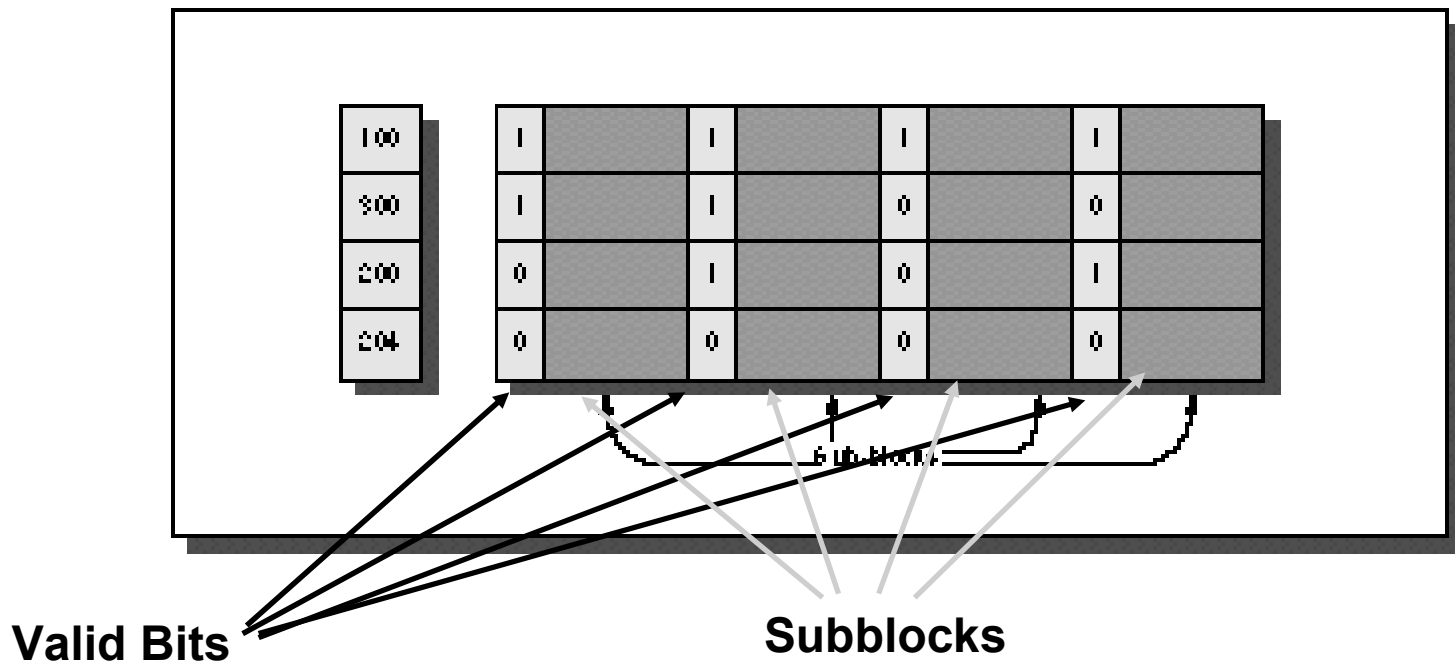
5. Reduce Miss Penalty: Non-blocking Caches to reduce stalls on misses

Non-blocking cache or *lockup-free cache* allow data cache to continue to supply cache hits during a miss

- requires out-of-order execution CPU

"*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
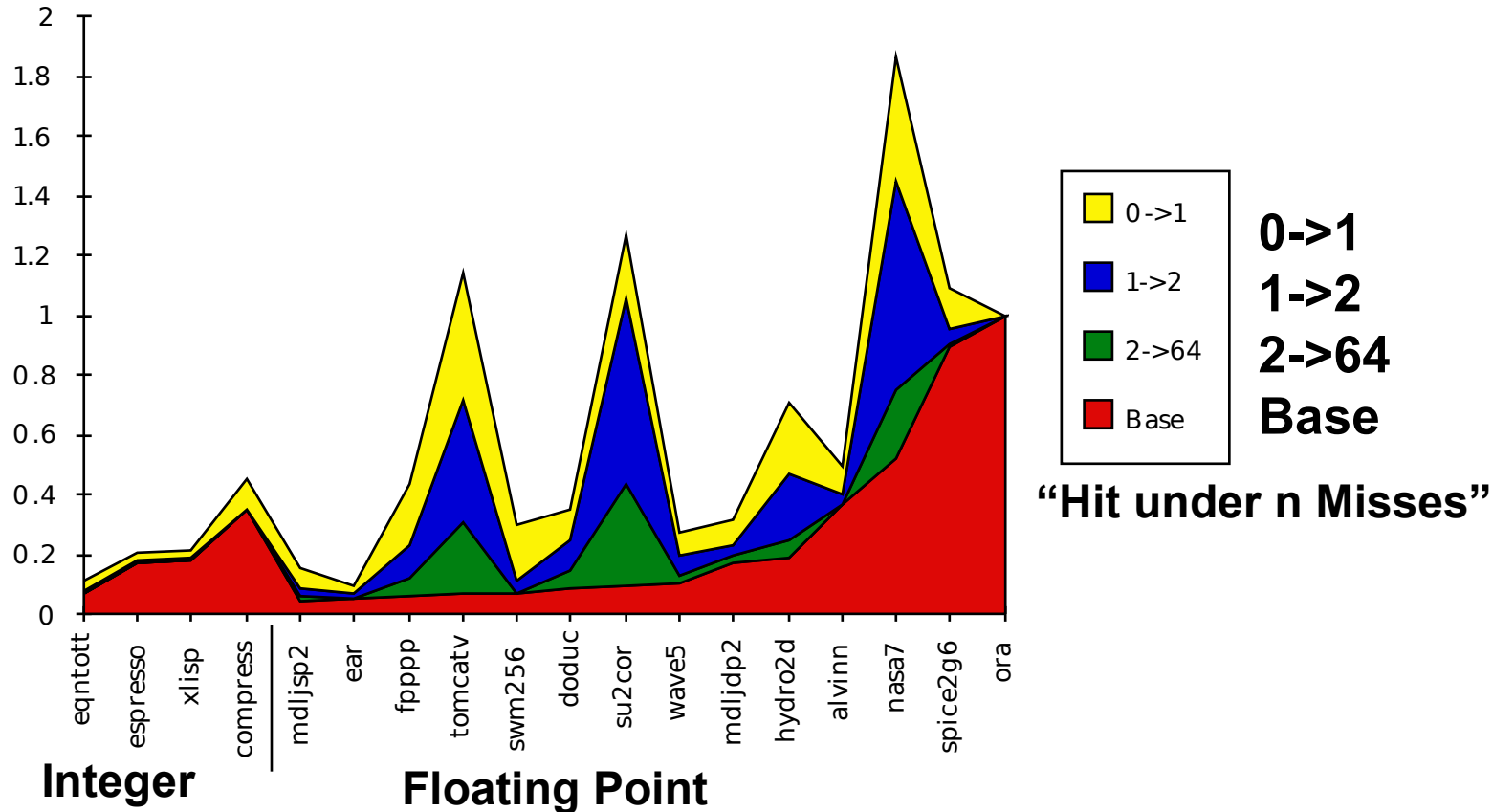
FP programs on average: AMAT= 0.68 -> 0.52 -> 0.34 -> 0.26

Int programs on average: AMAT= 0.24 -> 0.20 -> 0.19 -> 0.19

8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss

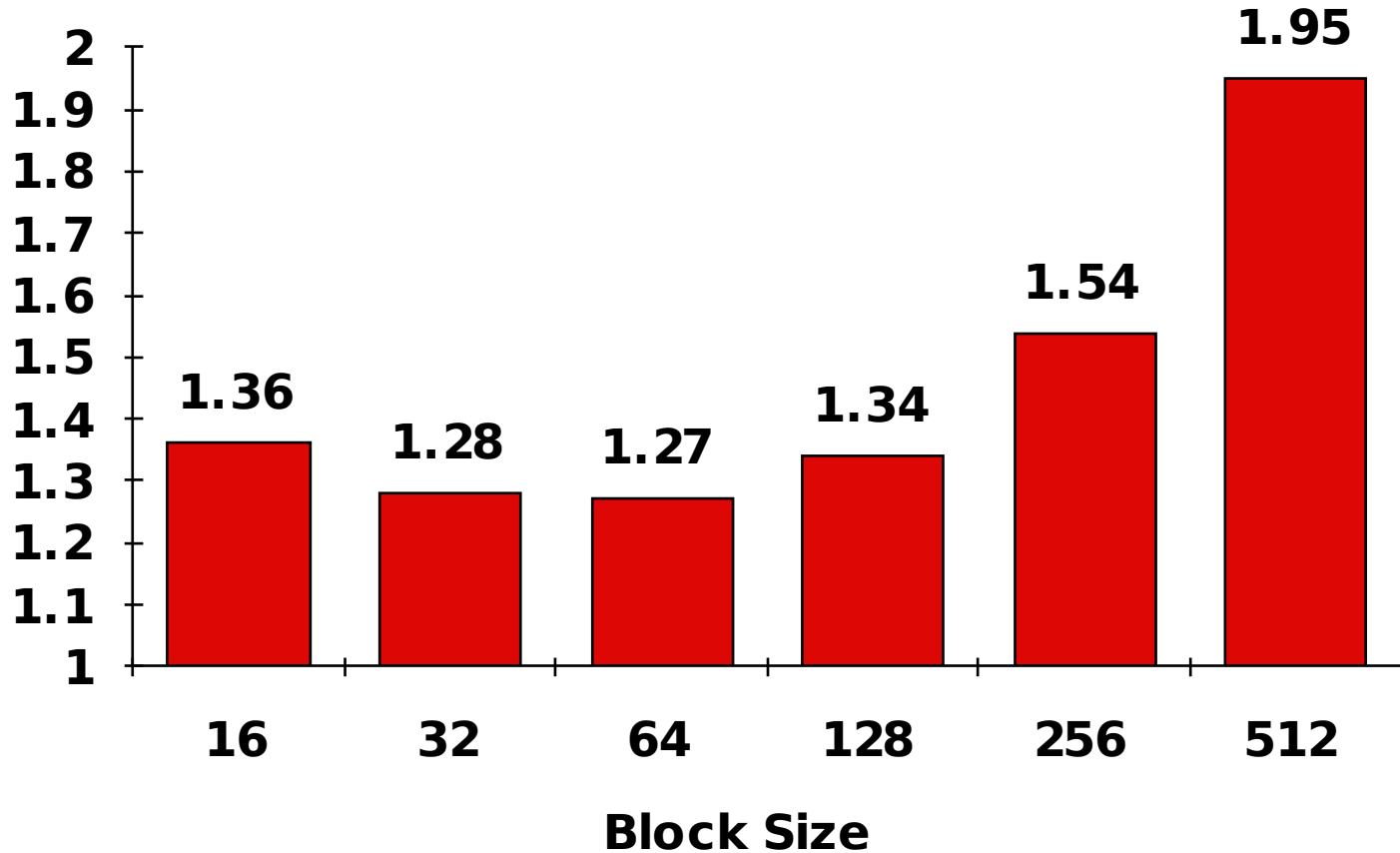# Reducing Misses:
Which apply to L2 Cache?

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 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

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

Five techniques

- Read priority over write on miss
- Subblock placement
- 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

Prefetching

Can be done by the hardware, software, or both.

It may reduce the miss rate and miss penalty.

Anticipation of the future needs of the cache is essential:

Early determination.

Enough bandwidth.

1. Reducing Misses by <u>Hardware</u> Prefetching of Instructions & Data

E.g., Instruction Prefetching

- Alpha 21064 fetches 2 blocks on a miss
- Extra block placed in "<u>stream buffer</u>"
- 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

2. 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

Issuing Prefetch Instructions takes time

- Is cost of prefetch issues < savings in reduced misses?

- Higher superscalar reduces difficulty of issue bandwidth

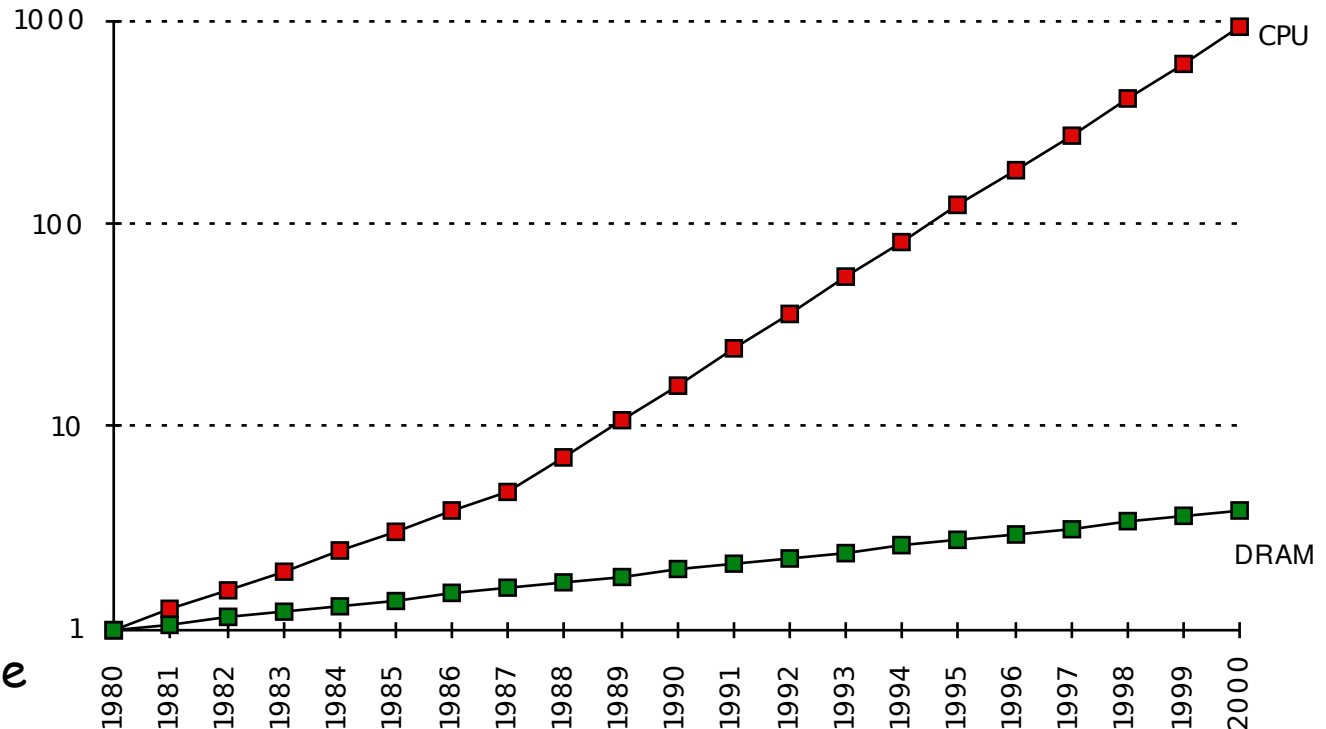# What is the Impact of What You've Learned About Caches?

1960-1985: Speed
= f(no. operations)

1990

- Pipelined
  Execution &
  Fast Clock Rate

- Out-of-Order
  execution

- Superscalar
  Instruction Issue

1998: Speed =
f(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

	Technique	MR	MP	HT	Complexity
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
	<u>Compiler Reduce Misses</u>	+			0
miss penalty	Priority to Read Misses		+		1
	Subblock Placement		+	+	1
	Early Restart & Critical Word 1st		+		2
	Non-Blocking Caches		+		3
	Second Level Caches		+		2