Improving 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 accesses x Miss rate x Miss penalty

- Above assumes 1-cycle to hit in cache
  - Hard to achieve in current-day processors (faster clocks, larger caches)
  - More reasonable to also include hit time in the performance equation

Average memory access time = Hit Time + Miss rate x Miss Penalty

D
- Small/simple caches
- Avoiding address translation
- Pipelined cache access
- Trace caches

B
- Larger block size
- Larger cache size
- Higher associativity
- Way prediction
- Compiler optimizations

A
- Multilevel caches
- Critical word first
- Read miss before write miss
- Merging write buffers
- Victim caches

C
- Nonblocking caches
- Hardware prefetching
- Compiler prefetching

Reducing Miss Penalty via Multilevel Caches

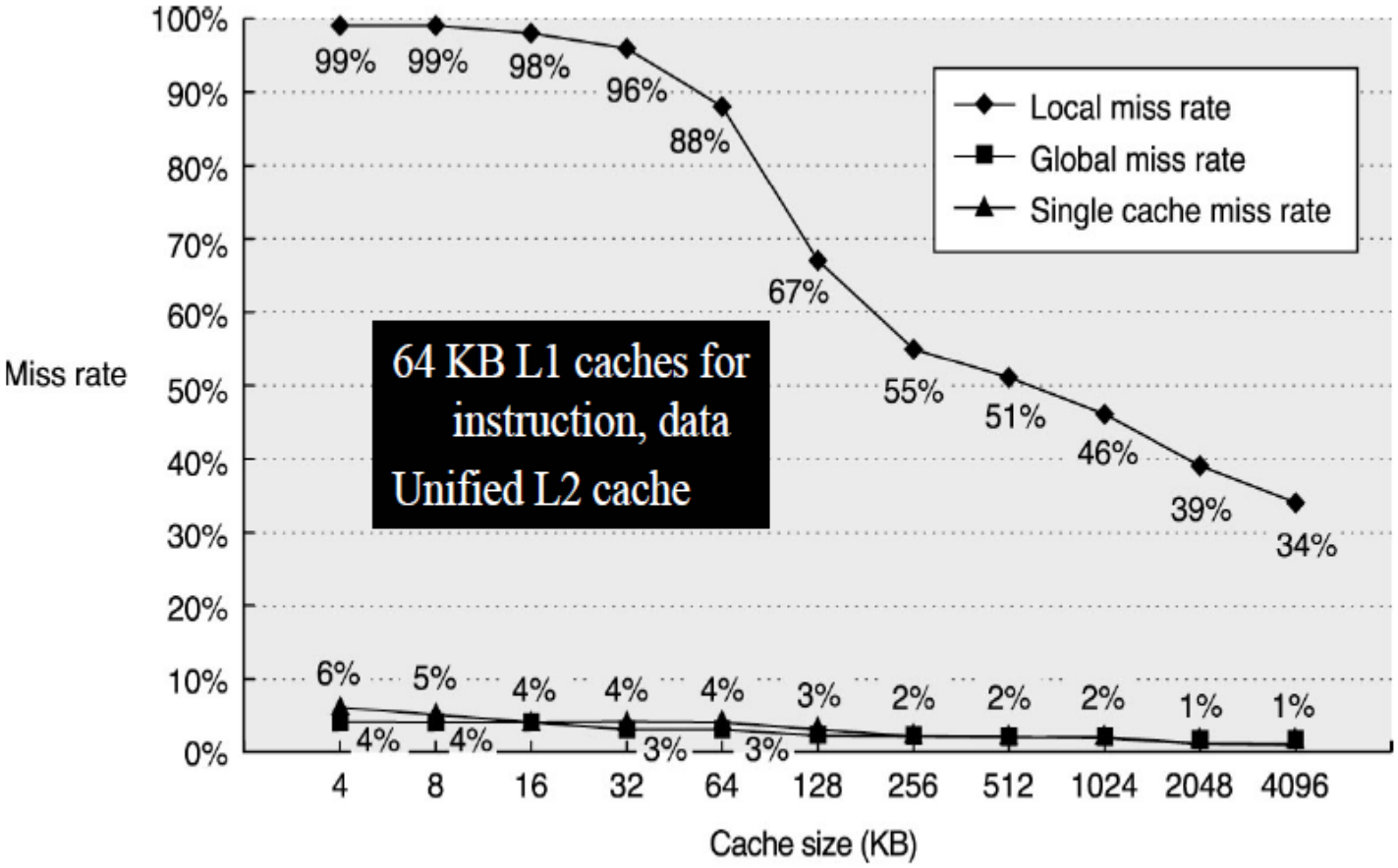
- *Idea*: Have multiple levels of caches
  - Tradeoff between size (cache effectiveness) and cost (access time)
- For a 2-level cache

  Average memory access time = Hit time (L1) + Miss rate (L1) x Miss penalty (L1)

  Miss penalty (L1) = Hit time (L2) + Miss rate (L2) x Miss penalty (L2)

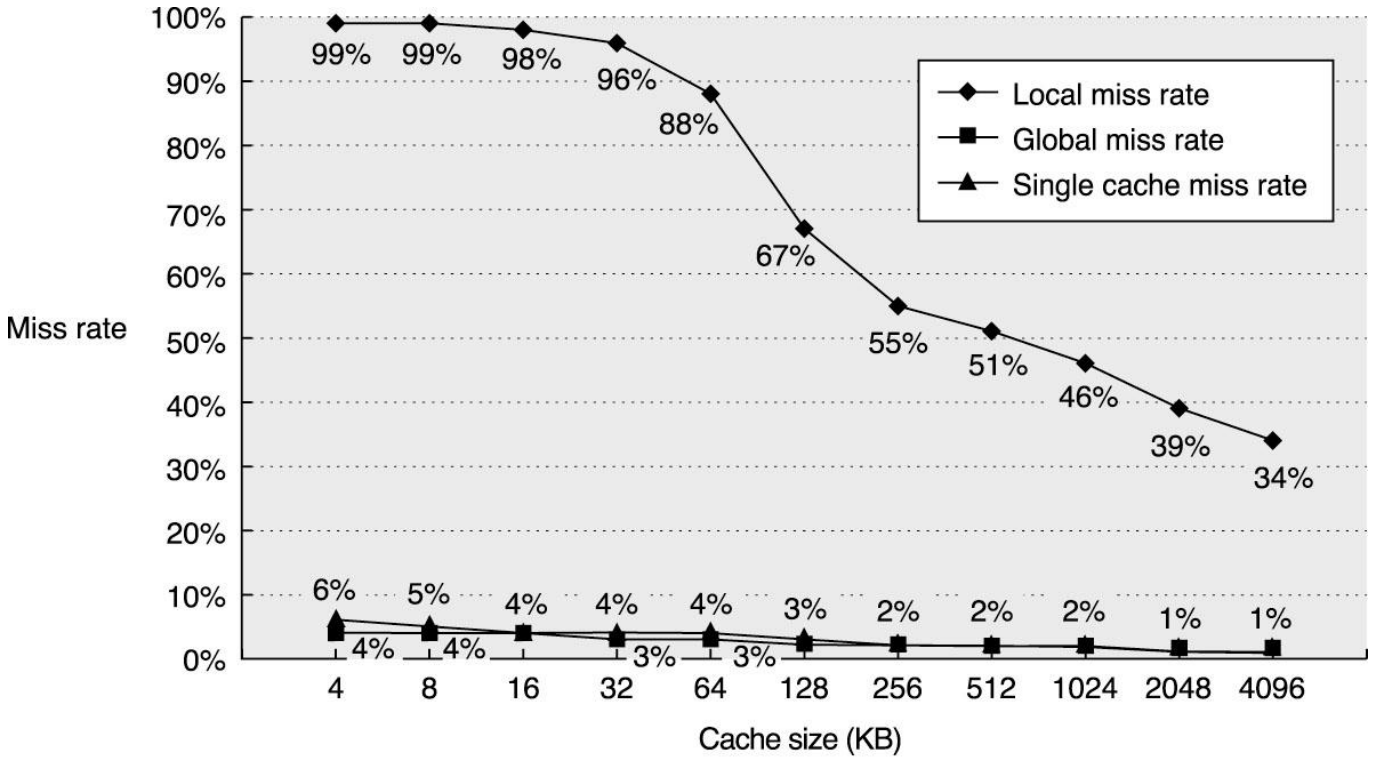
- Distinguish between two kinds of miss rates
  - Local miss rate = Miss rate (L1) or Miss rate (L2)
  - Global miss rate = Number of misses/total number of memory accesses
    = Miss rate (L1), but Miss rate (L1) x Miss rate(L2)
- Example: 1000 references, 40 misses in L1 cache and 20 in L2
  - **Local miss rates: 4% (L1), 50% (L2) = 20/40**
  - Global miss rates: 4% (L1), 2% (L2)
  - Avg. memory access time = 1 + **4%** x (10 + **50%** x 100) = 3.4 cycles

Multilevel Caches (cont'd)



- Doesn't make much sense to have L2 caches smaller than L1 caches
- L2 needs to be significantly bigger to have reasonable miss rates
  - Cost of big L2 is smaller than big L1

Second Level Cache



- Global cache miss is similar to single cache miss rate of second level cache provided L2 cache is much bigger than L1.
- Local cache rate is NOT good measure of secondary caches as it is function of L1 cache. Global cache miss rate should be used.

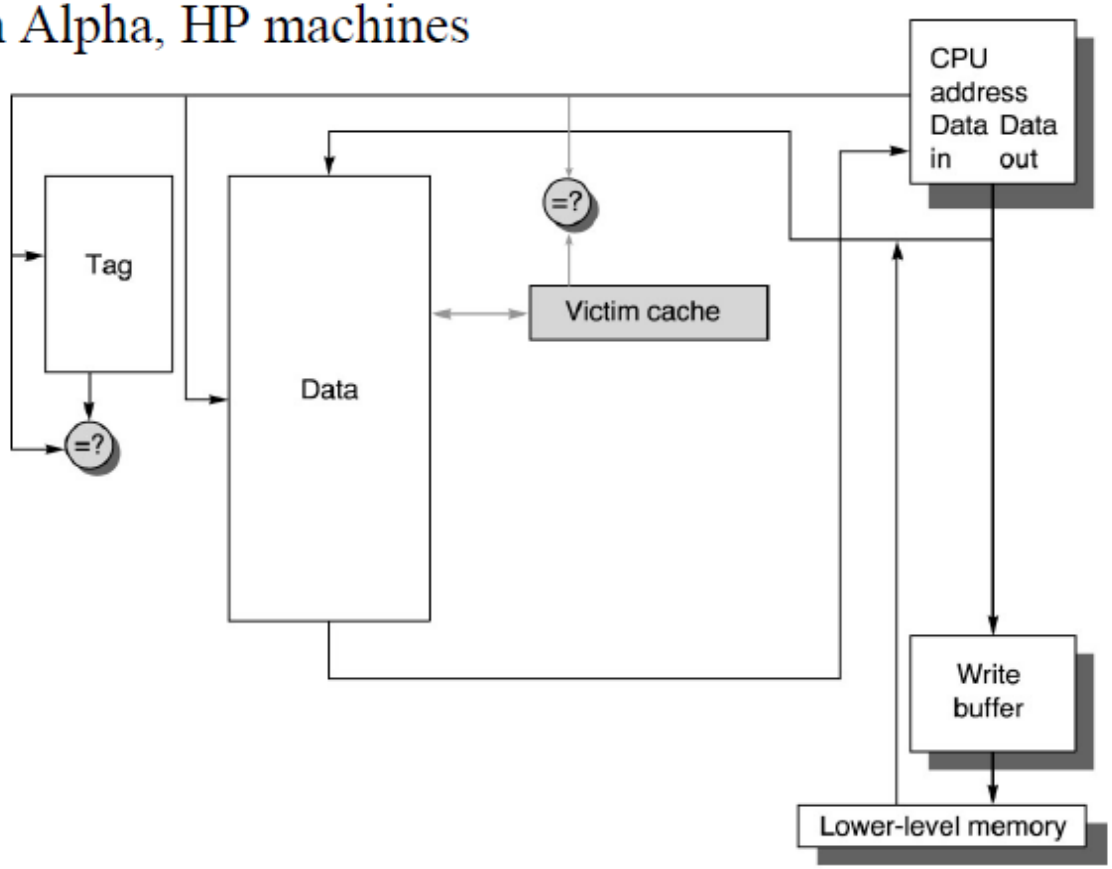
Reduce Miss Penalty via Critical Word First and Early Restart

- *Idea*: Don't wait for full block to be loaded before restarting CPU
  - Early restart: request the words in a block in order. 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

- Drawbacks
  - Generally useful only in large blocks
  - Programs exhibiting spatial locality a problem; tend to want next sequential word, so limited benefit by early restart

Reducing Miss Penalty via a "Victim Cache"

- How to combine the fast hit time of direct-mapped caches, yet still avoid conflict misses?
- Remember what was recently discarded, just in case it is needed again
  - Jouppi [1990]: 4-entry victim cache reduced conflict misses by 20% - 95% for a 4 KB direct mapped data cache
  - Used in Alpha, HP machines



B. Reducing Cache Misses

Classifying Misses: 3 Cs

- Compulsory (Also called cold start or first reference misses)
  - The first access to a block is not in the cache, so the block must be brought into the cache.
    (Misses in even an Infinite Cache)
- Capacity
  - The cache may not contain all blocks needed during program execution, so misses will occur due to blocks being discarded and later retrieved
    (Misses in Fully Associative Size X Cache)
- Conflict (Also called collision or interference misses)
  - Additional misses that occur because another block is occupying cache (the rest of the cache might be unused)
    (Misses in N-way Associative, Size X Cache)

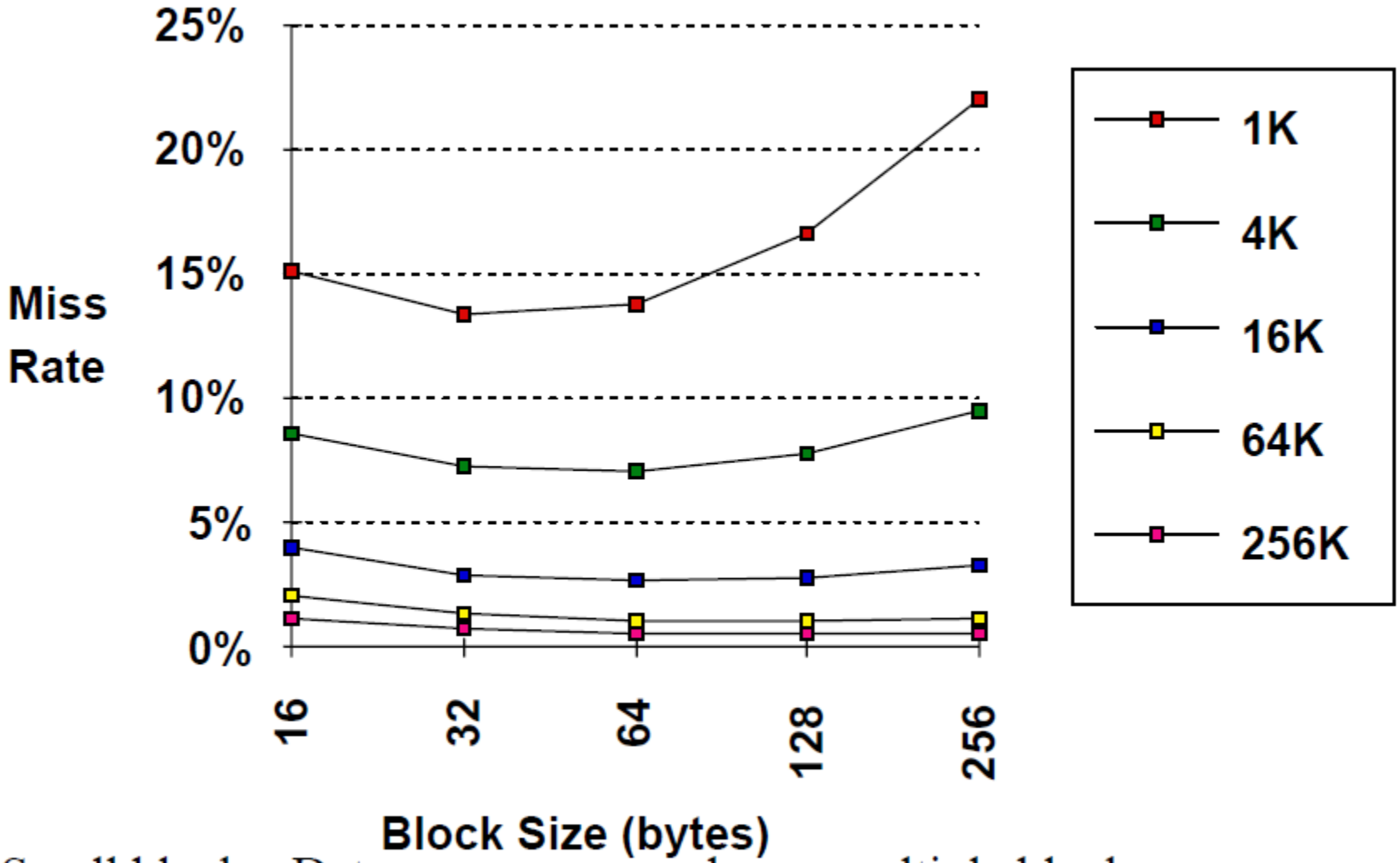
How Can We Reduce Misses?

- 3 Cs: Compulsory, Capacity, Conflict

- If we assume that total cache size is not changed, what happens if we

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?

Reducing Miss Rate via Larger Block Sizes



- Small blocks: Data accesses spread over multiple blocks
- Large blocks: Not all the data is useful, but displaces useful data
- Also note larger blocks mean higher miss penalty

Reducing Miss Rate via Higher Associativity

- 2:1 Cache Rule
  - Miss Rate of a direct-mapped cache size of size N ~ Miss Rate of a 2-way cache of size N/2
- Is this actually the case?
  - *Issue*: Increase in **clock cycle time (CCT)** may diminish benefits
- Higher associativity leads to higher hit time and can outweigh the benefit
- Average memory access time for SPEC92 vs. associativity

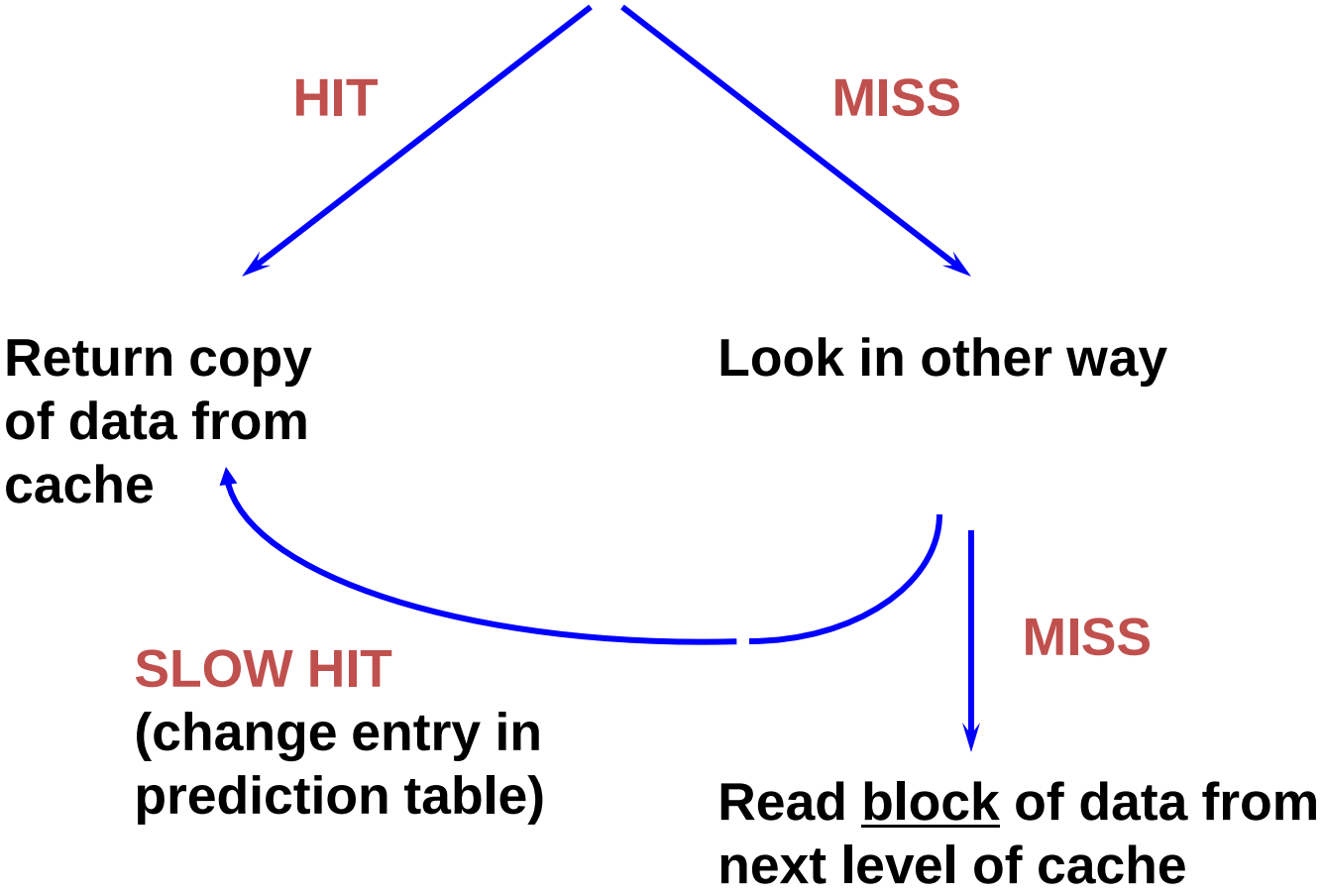
Size (KB)	*Associativity*			
	1-way	2-way	4-way	8-way
4	3.44	3.25	3.22	3.28
8	2.69	2.58	2.55	2.62
16	2.23	2.40	2.46	2.53
32	2.06	2.30	2.37	2.45
64	1.92	2.14	2.18	2.25
128	1.52	1.86	1.92	2.00
256	1.32	1.66	1.74	1.82
512	1.20	1.55	1.59	1.66

Reducing Miss Rate via Way Prediction and Pseudoassociativity

- How to combine fast hit time of direct-mapped caches with the lower conflict misses of set-associative caches?
  - Previously looked at Victim Caches
- Way prediction: Predict which block in a set is likely to be accessed by the next memory access hitting this set
  - Tag comparison only with this block (cheaper as opposed to with all)
    - Higher cost to check non-predicted blocks
  - Simplest prediction: remember the last word accessed
  - Used in Alpha 21264 (1-cycle if correct prediction (85%), 3-cycles o.w.)
- Pseudoassociative or Column associative
  - Access proceeds as in direct-mapped cache
  - On a miss, check another location ("pseudoset") before going to memory
    - Counts as a "slower hit"
    - If most hits become slow hits, degrading performance is possible
  - Used in MIPS R10000 L2 cache, similar in UltraSPARC

Way Predicting Caches

- Use processor address to index into way prediction table
- Look in predicted way at given index, then:



Reducing Miss Rate by Compiler Optimizations

- Compiler optimizations can help reduce both instruction and data cache misses (for a fixed cache organization)
- Instruction misses
  - Reorder procedures in memory so as to reduce conflict misses
    - Ensure that procedures used frequently do not map to same blocks/sets
    - Conflicts determined by profiling
- Data misses
  - Several optimizations that reorder data access patterns
  - Two examples
    - Loop interchange
    - Blocking

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];
```

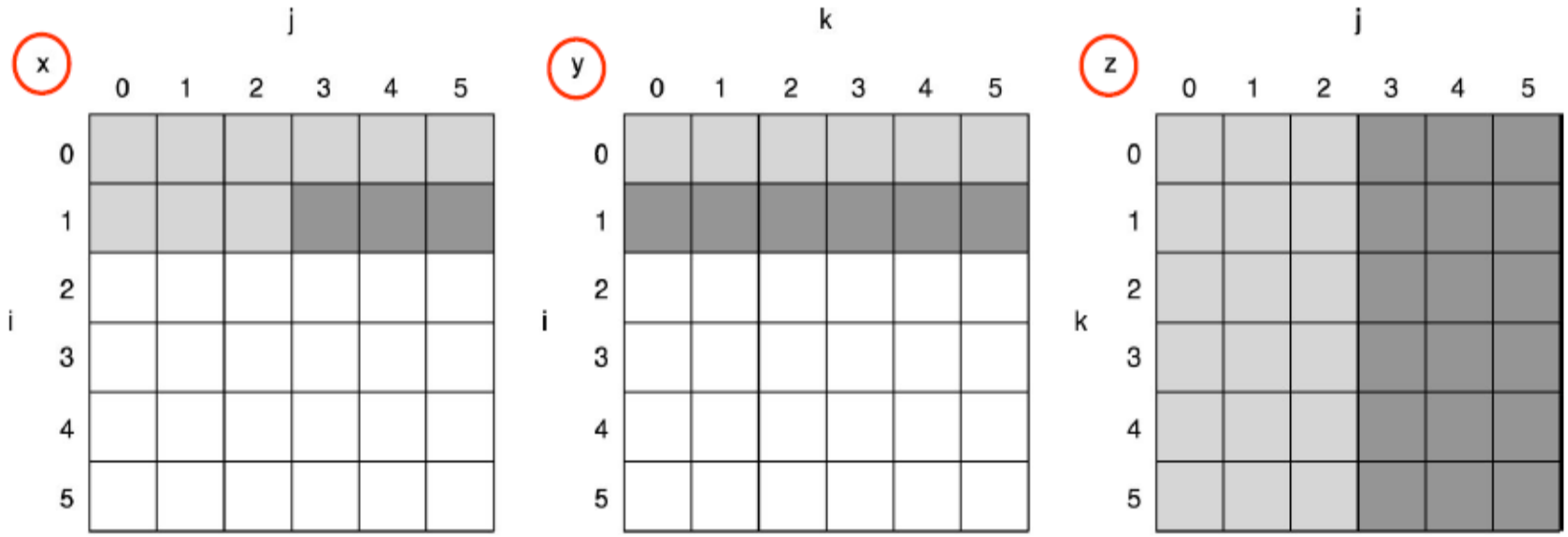
- "After" version accesses memory sequentially instead of in strides of 100 words
  - Improved spatial locality: use all of the words in fetched blocks

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;
       };
```

Capacity misses depend on N, cache size

- if all three matrices fit and there are no conflict misses, best performance
- if cache can hold one NxN matrix and one row of N elements, then y and z can be in the cache
- else, misses for both y and z
- worst case: $2N^3 + N^2$ misses



Blocking Example (cont'd)

```
/* 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;
      };
```

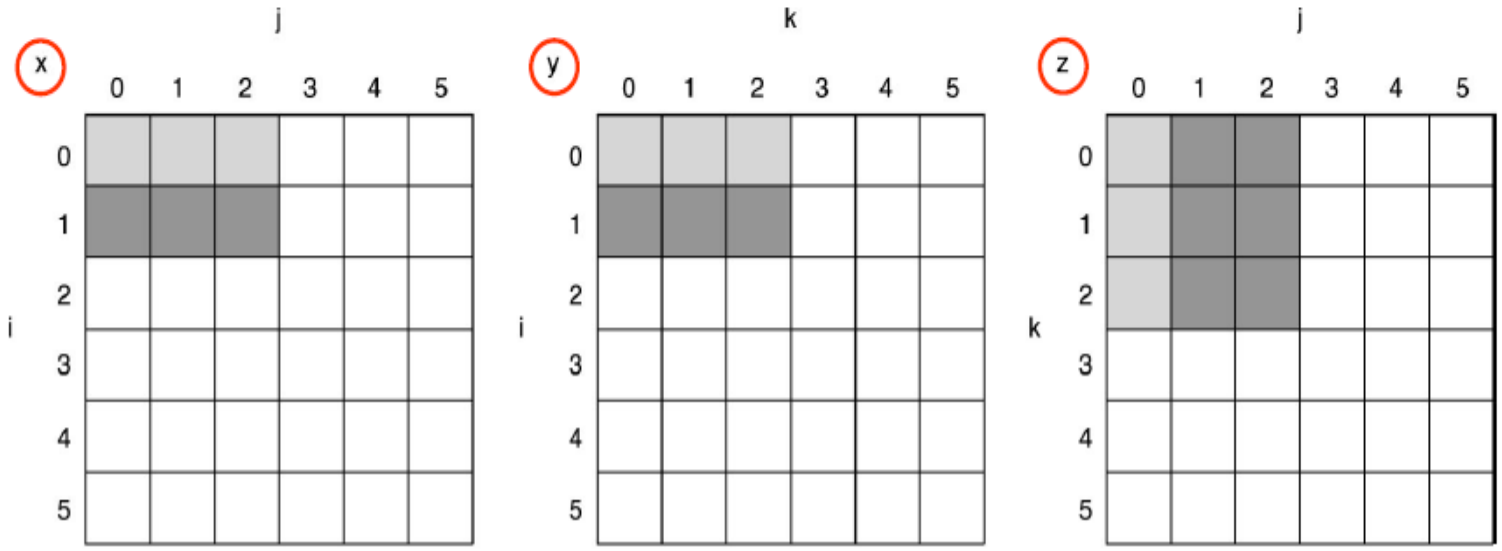
N^2/B^2

NB (x) + NB (y) + B^2 (z)

Blocking factor: compute in blocks of BxB

B chosen such that 1 row of B and 1 BxB matrix can fit in the cache. This ensures that y and z blocks are resident

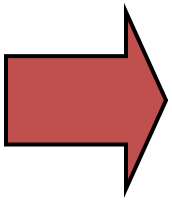
Capacity misses: $2N^3/B + N^2$



Merging Arrays

```
int val[SIZE];
int key[SIZE];

for (i=0; i<SIZE; i++){
  key[i] = newkey;
  val[i]++;
}
```



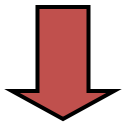
```
struct record{
   int val;
   int key;
};
struct record records[SIZE];

for (i=0; i<SIZE; i++){
   records[i].key = newkey;
   records[i].val++;
}
```

- Reduces conflicts between `val` & `key` and improves spatial locality

Loop Fusion

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

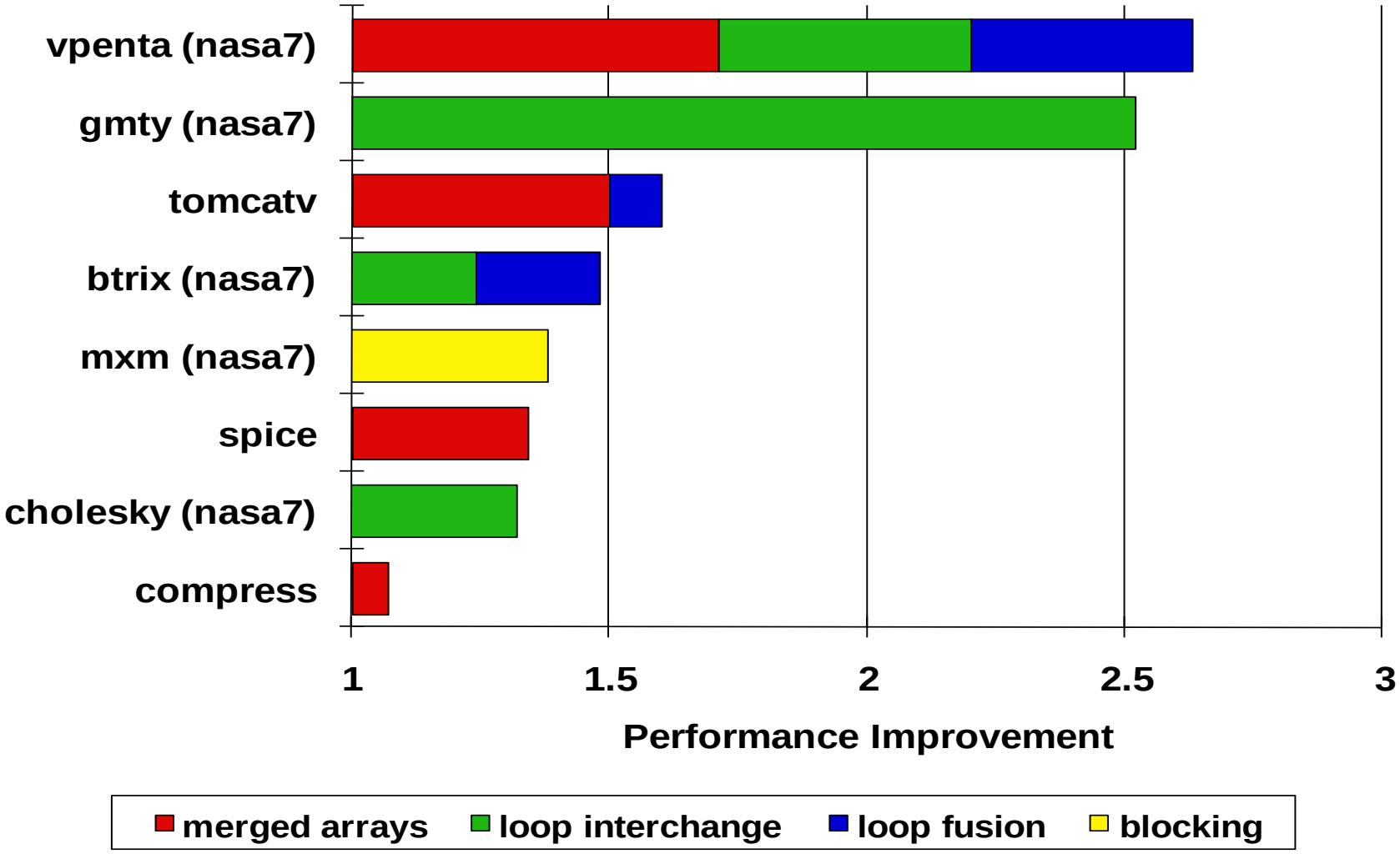


```
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++){
        a[i][j] = 1/b[i][j] * c[i][j];
        d[i][j] = a[i][j] + c[i][j];
}
```

Reference can be directly to register

Splitted loops: every access to a and c misses. Fused loops: only 1st access misses. Improves temporal locality

Summary of Compiler Optimizations to Reduce Cache Misses



C. Using Parallelism to Reduce Miss Penalty/Rate

- *Idea*: Permit multiple "outstanding" memory operations
  - Can overlap memory access latencies
  - Can benefit from activity done on behalf of other operations

Three commonly-employed schemes

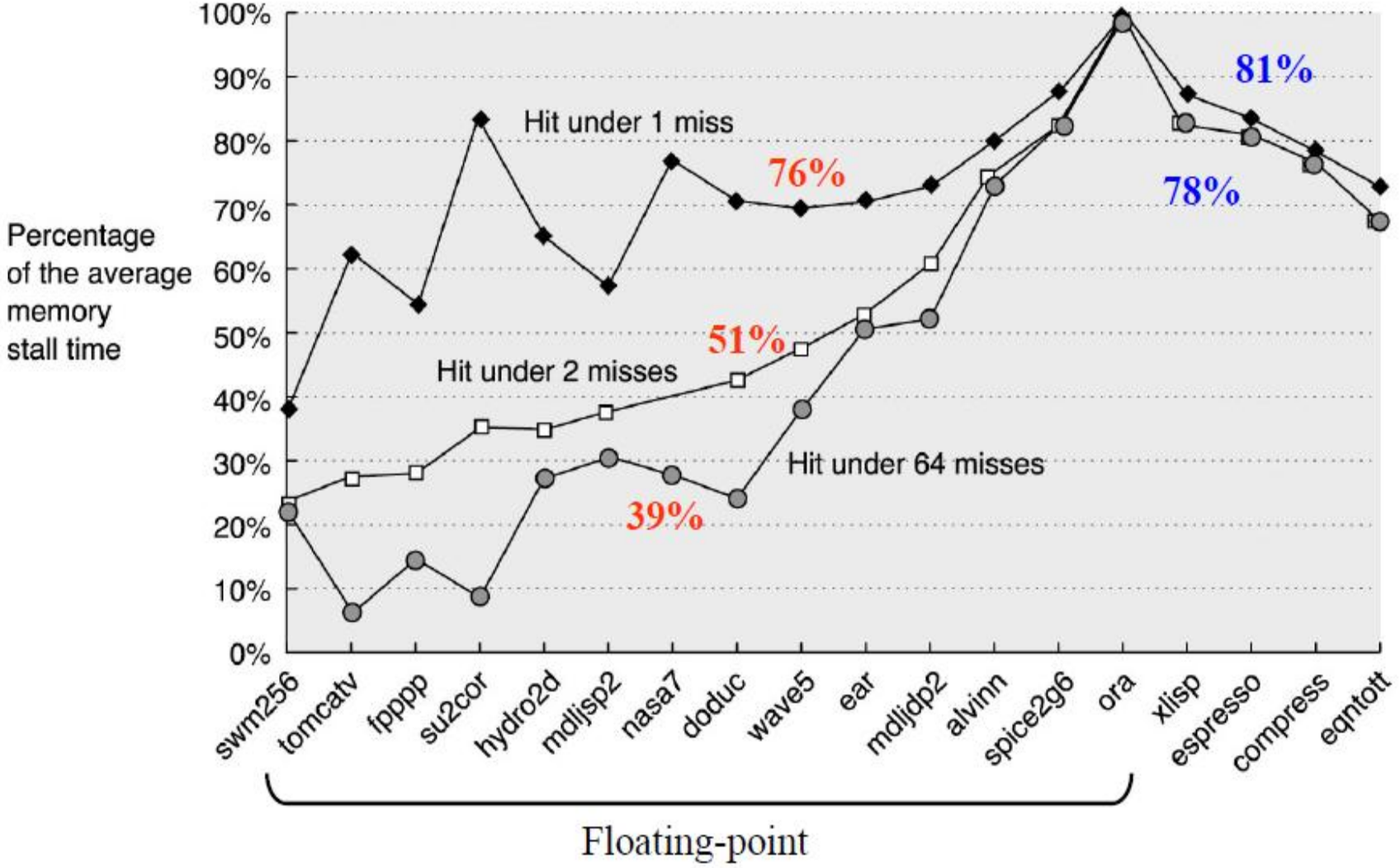
- Non-blocking caches
- Hardware prefetching
- Software prefetching

Non-blocking Caches to Reduce Stalls on Misses

- Decoupled instruction and data caches allow CPU to continue fetching instructions while waiting on a data cache miss
  - L1 cache misses can be tolerated by superscalar out-of-order machines
- Non-blocking or lockup-free caches 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
  - Typically also requires multiple memory banks
  - Pentium Pro allows 4 outstanding memory misses

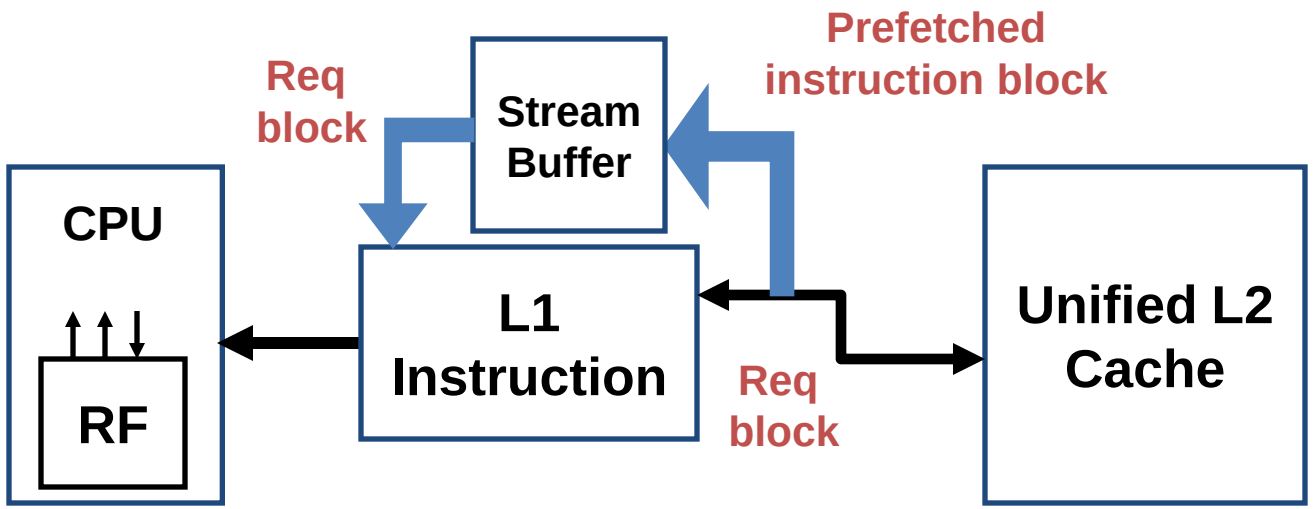
Value of Hit-Under-Miss for SPEC92

8KB direct-mapped cache, 32B blocks, 16-cycle penalty

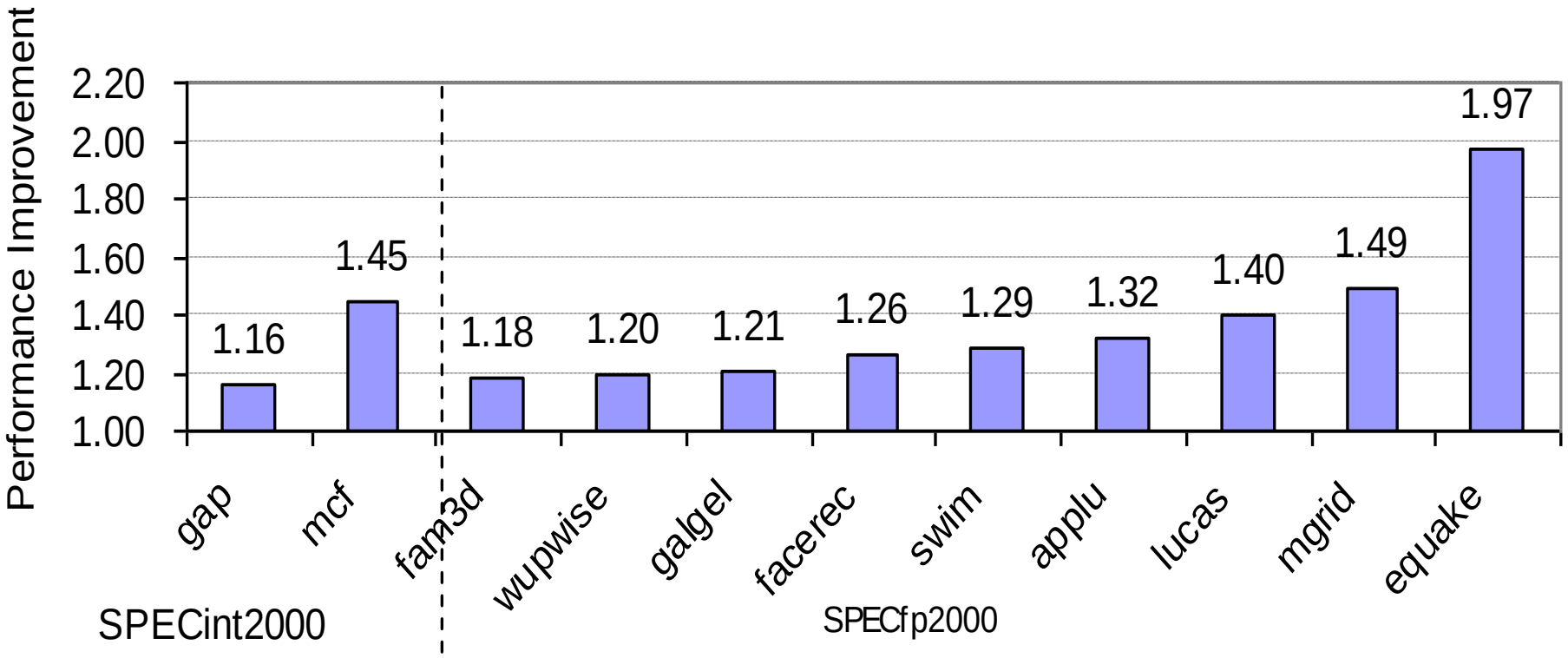


Hardware Instruction Prefetching

- Instruction prefetch in Alpha AXP 21064
  - Fetch two blocks on a miss; the requested block (i) and the next consecutive block (i+1)
  - Requested block placed in cache, and next block in instruction stream buffer
  - If miss in cache but hit in stream buffer, move stream buffer block into cache and prefetch next block (i+2)



Performance impact of prefetching



D. Reducing Cache Hit Time

- Obvious approach: Smaller and simpler (low associativity) caches
  - Notable that L1 cache sizes have not increased
    - Alpha 21264/21364; UltraSPARC II/III; AMD K6/Athlon

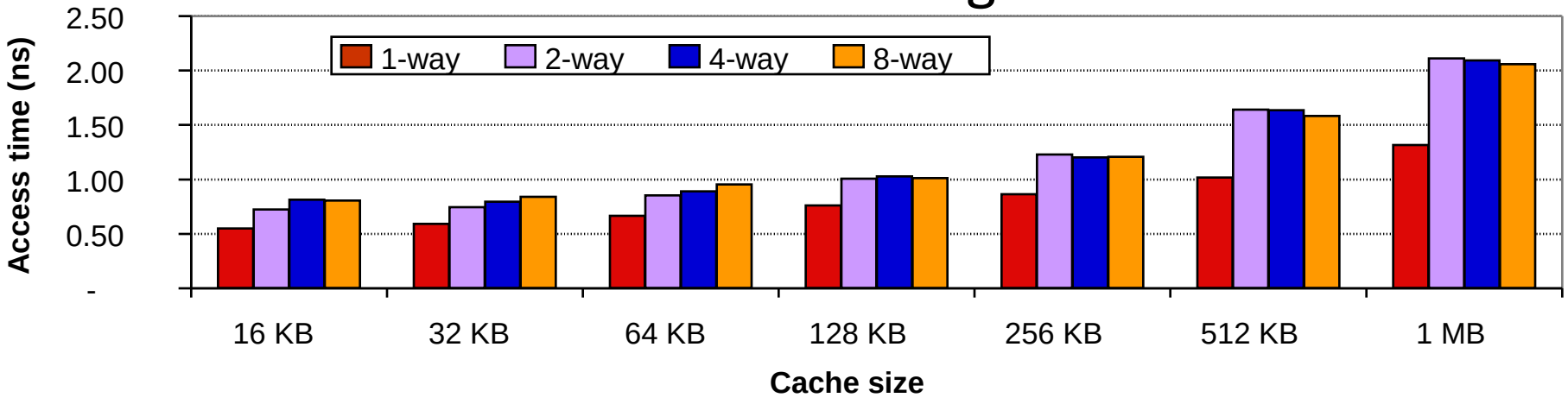
Other techniques

- Avoiding address translation during cache lookup
  - Alternative 1: Index caches using "virtual addresses"
    - Needs to cope with several problems
      - Protection (performed during address translation)
      - Reuse of virtual addresses across processes (flushing cache after context switch)
      - Aliasing/synonyms: Two processes refer to the same physical address (results in having multiple copies of the same data)
      - I/O (typically uses physical addresses)
  - Alternative 2: Use part of the page offset to index the cache
    does not change between virtual and physical addresses

Fast Hit via Small and Simple Caches

- Index tag memory and thereafter compare takes time
- ⇒ Small cache is faster
  - Also L2 cache small enough to fit on chip with the processor avoids time penalty of going off chip
- Simple ⇒ direct mapping
  - Can overlap tag check with data transmission since no choice

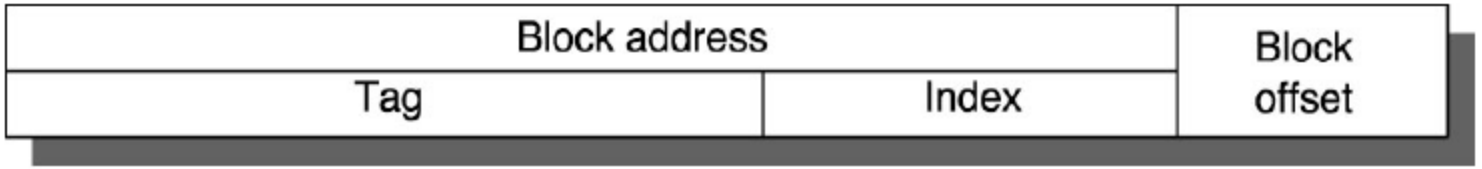
Access time estimate for 90 nm using CACTI model 4.0



D.1. Virtually Indexed, Physically Tagged Caches

- Overlap indexing of cache with translation of virtual addresses
  - Tag comparison done with physical addresses

Implications



- Direct-mapped caches can be no bigger than page size
- Set-associative caches
  - Page offset can be viewed as (Index + block offset) above
  - Cache size = $2^{\text{page offset}}$ x Set associativity
  - So, increased associativity allows larger cache sizes
    - Pentium III (8KB pages): 2-way set-associative 16 KB cache
    - IBM 3033 (4KB pages): 16-way set-associative 64 KB cache

Translation Lookaside Buffer

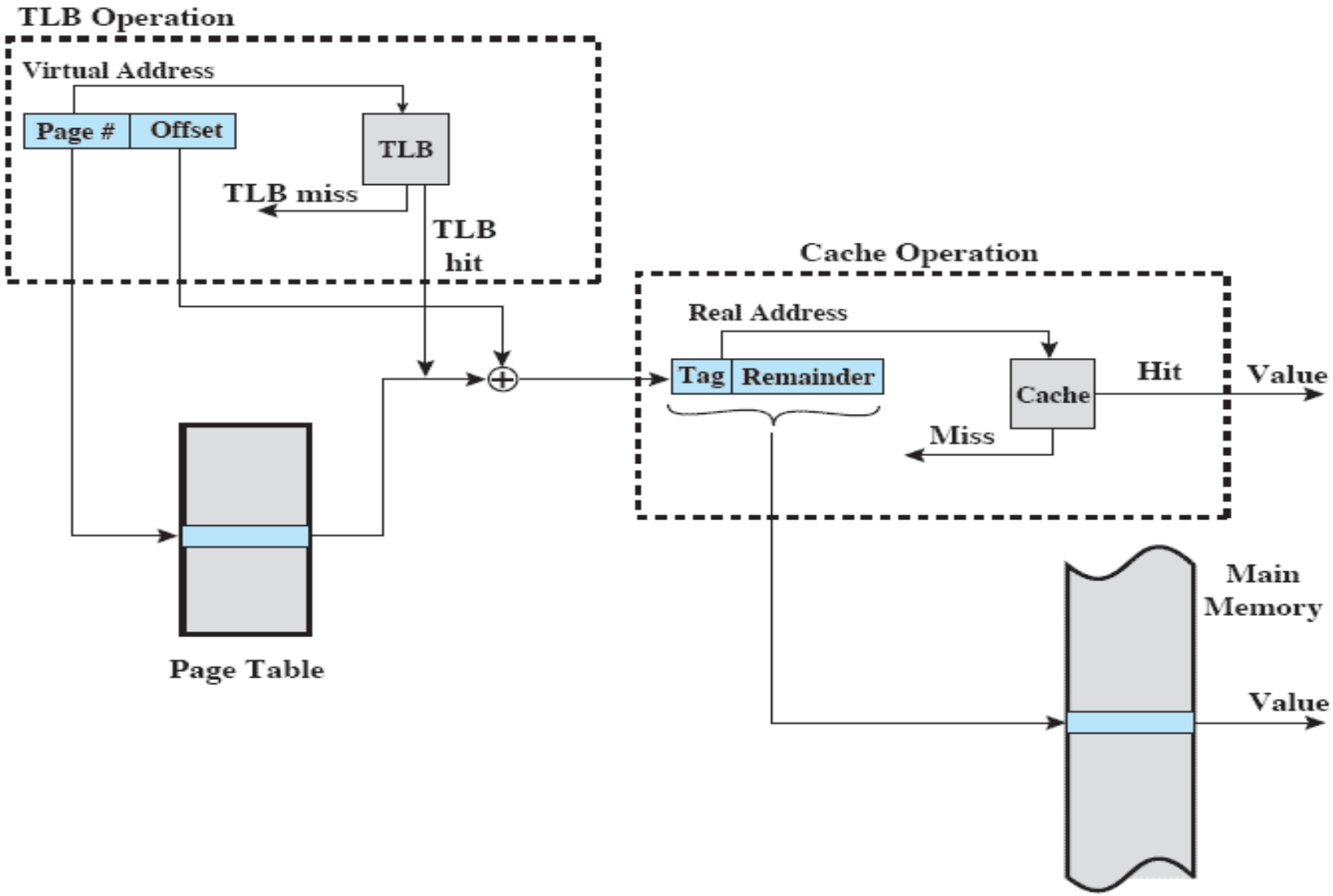
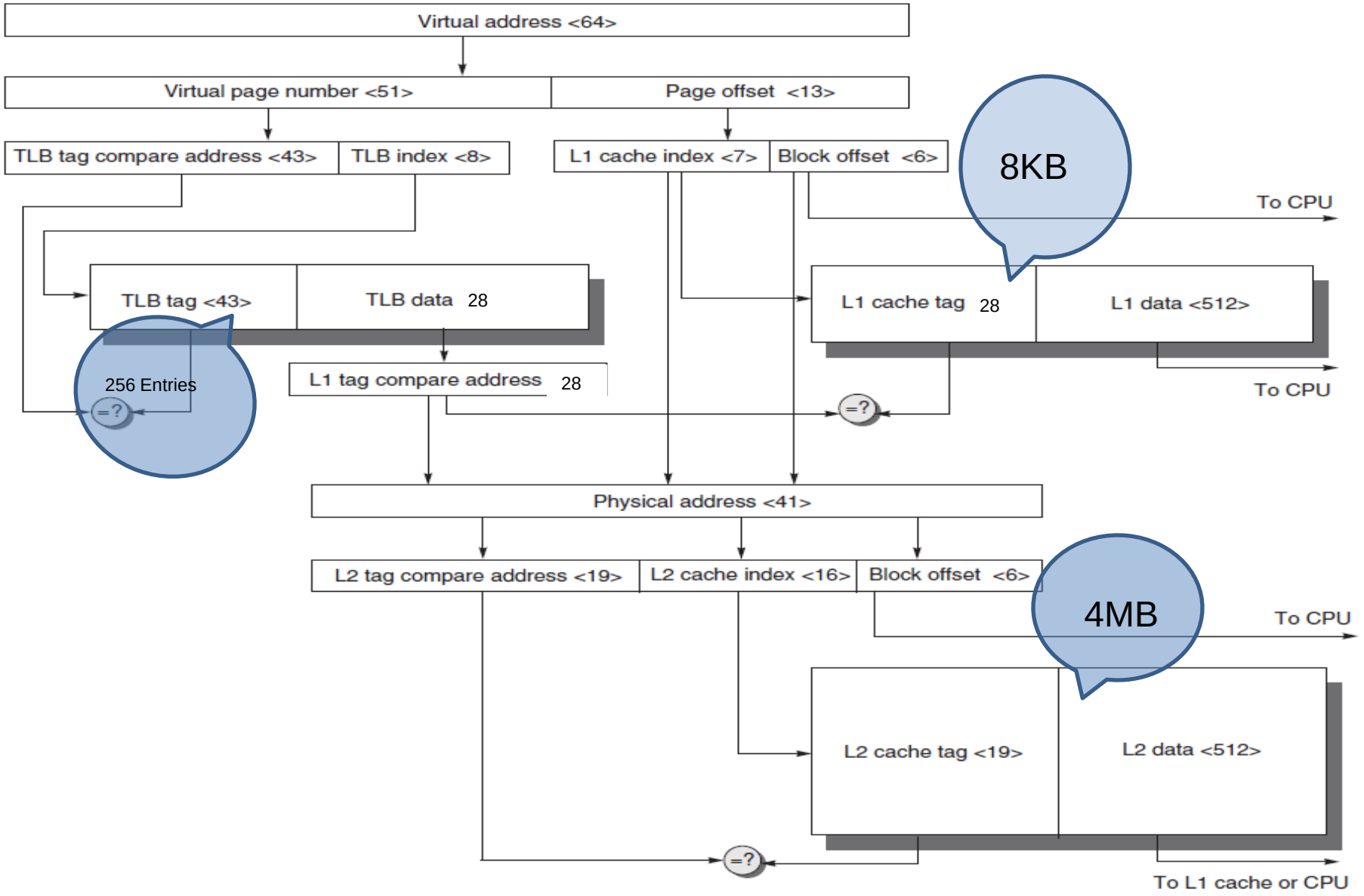


Figure 8.10 Translation Lookaside Buffer and Cache Operation



ure 5.3 The overall picture of a hypothetical memory hierarchy going from virtual address to L2 cache ess. The page size is 8 KB. The TLB is direct mapped with 256 entries. The L1 cache is a direct-mapped 8 KB, and L2 cache is a direct-mapped 4 MB. Both use 64-byte blocks. The virtual address is 64 bits and the physical address 0 bits. The primary difference between this figure and a real memory hierarchy, as in Figure 5.18 on page 327, is her associativity for caches and TLBs and a smaller virtual address than 64 bits.

Cache Optimization Summary

Technique	*MP*	*MR*	*HT*	*Complexity*
Multilevel caches	+			2
Early Restart & Critical Word 1st	+			2
Priority to Read Misses	+			1
Merging write buffer	+			1
Victim Caches	+	+		2
Larger Block Size	–	+		0
Higher Associativity		+	–	1
Pseudo-Associative Caches		+		2
Compiler Reduce Misses		+		0
Non-Blocking Caches	+			3
HW Prefetching of Instr/Data	+	+		2/3
Compiler Controlled Prefetching	+	+		3
Avoiding Address Translation			+	2
Trace Cache			+	3