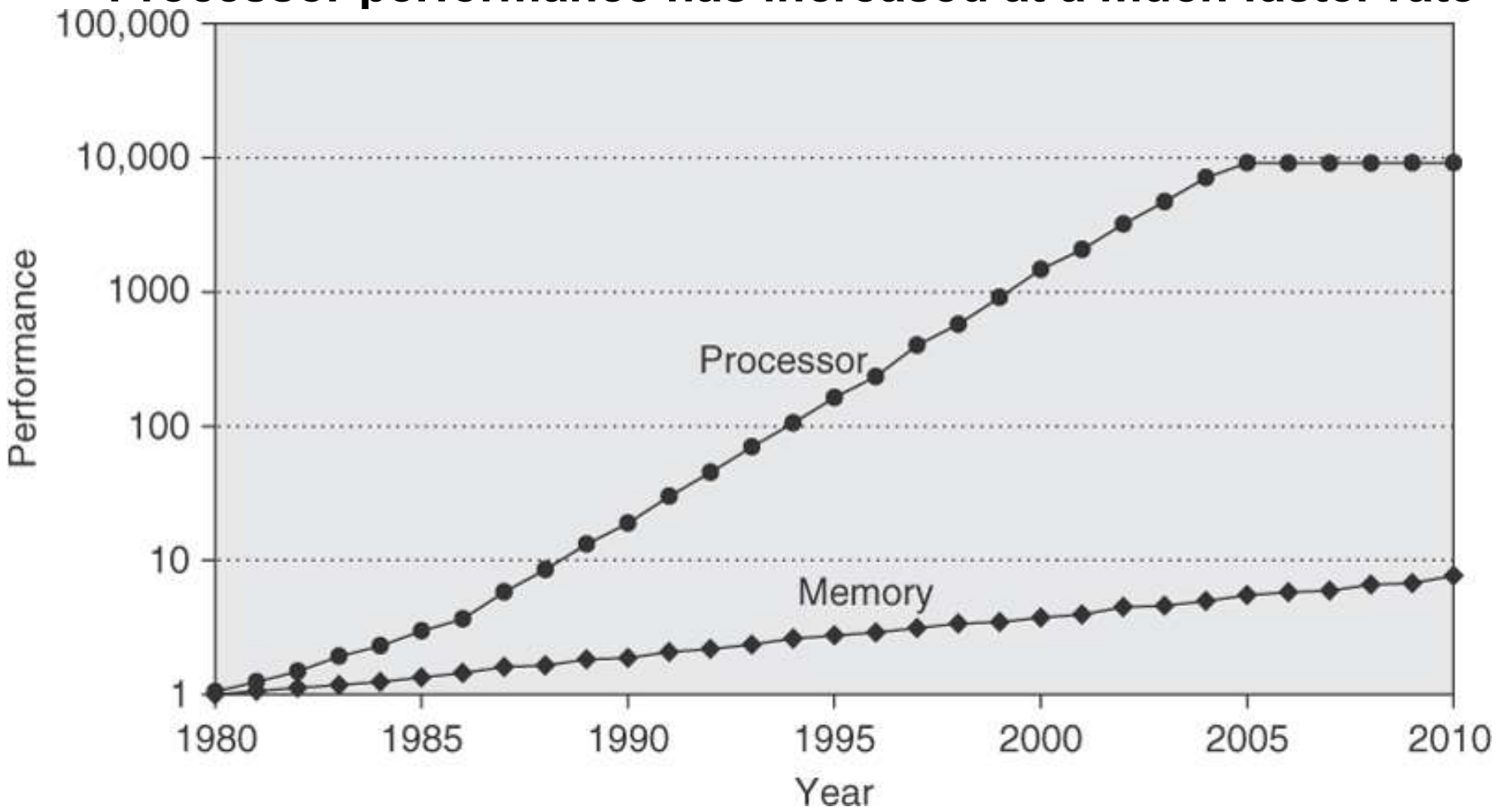
Memory Hierarchy and Cache Design

Why is memory important?

- **Processor performance has increased at a much faster rate**

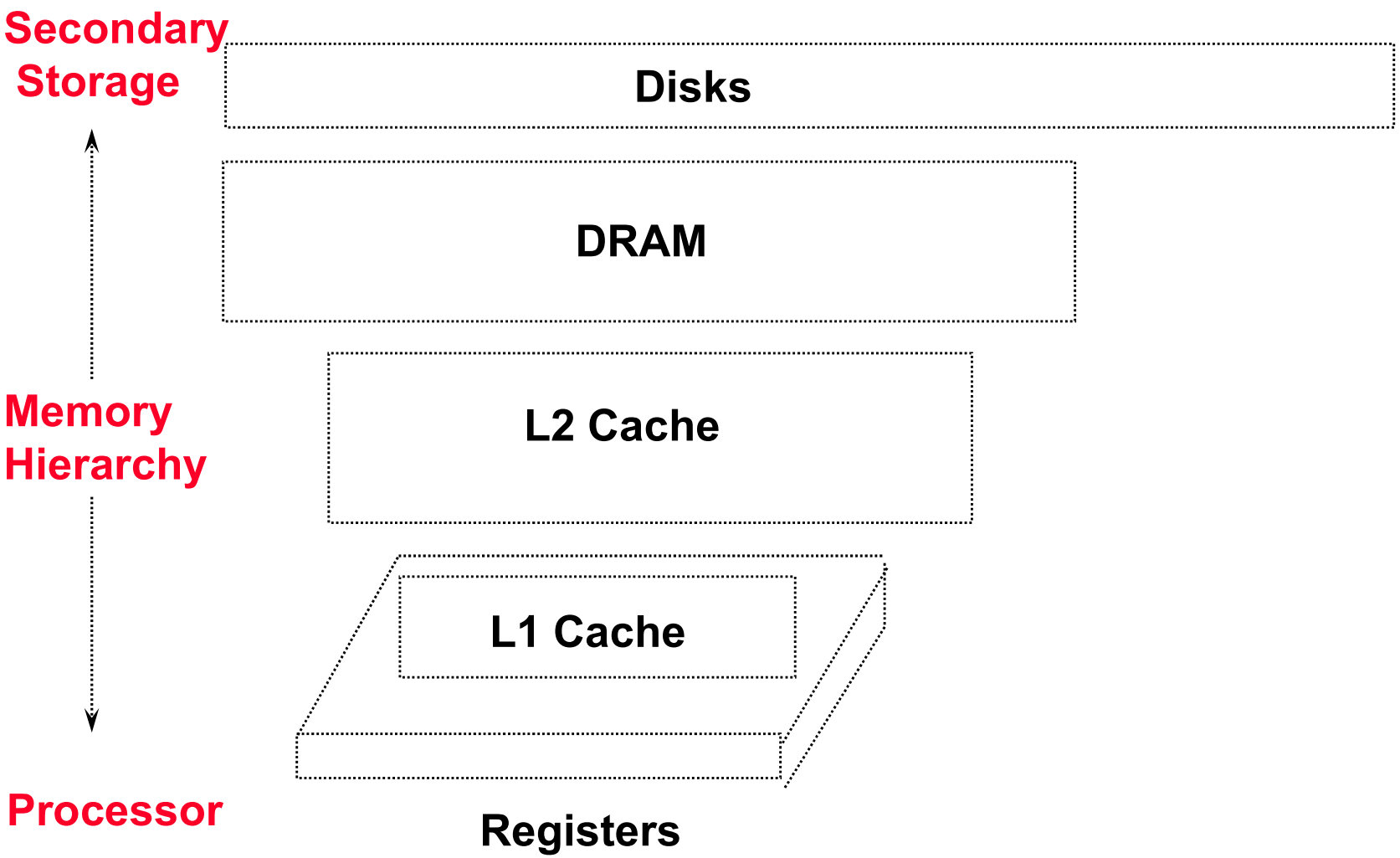


- **1980: no cache in mproc;**
  1995 2-level cache, 60% trans. on Alpha 21164 mproc

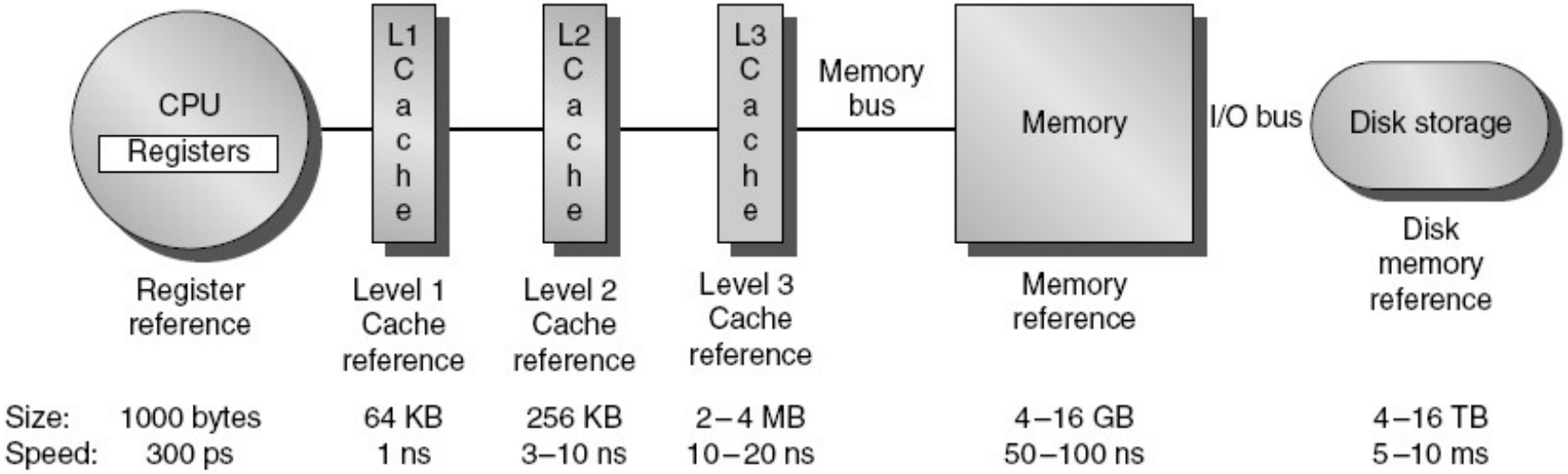
How do architects address this gap?

- **Programmers want unlimited amounts of memory with low latency**
- **Fast memory technology is more expensive per bit than slower memory**
- **Solution: organize memory system into a hierarchy**
  - Entire addressable memory space available in largest, slowest memory
  - Incrementally smaller and faster memories, each containing a subset of the memory below it, proceed in steps up toward the processor
- **Temporal and spatial locality insures that nearly all references can be found in smaller memories**
  - Gives the allusion of a large, fast memory being presented to the processor

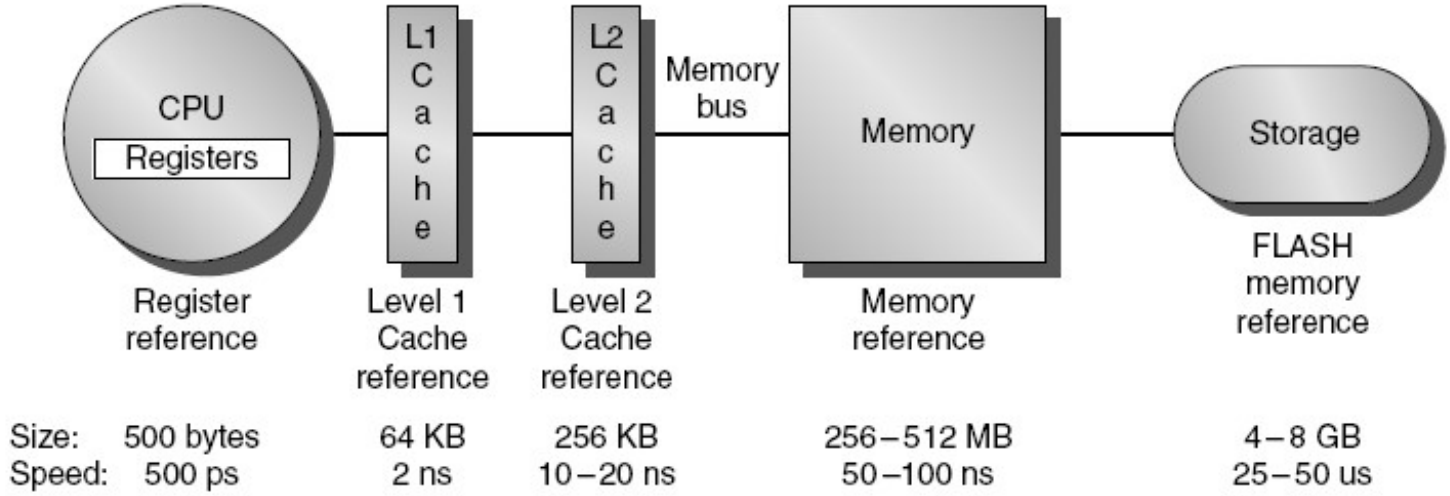
Memory Hierarchy



Memory Hierarchy



(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device

Generations of Microprocessors

- **Time of a full cache miss in instructions executed:**

1st Alpha:	**340 ns/5.0 ns = 68 clks x 2 or**	**136**
2nd Alpha:	**266 ns/3.3 ns = 80 clks x 4 or**	**320**
3rd Alpha:	**180 ns/1.7 ns =108 clks x 6 or**	**648**

access time/cycle time X instructions per cycle.

newer processors waste more instructions because of cache misses.

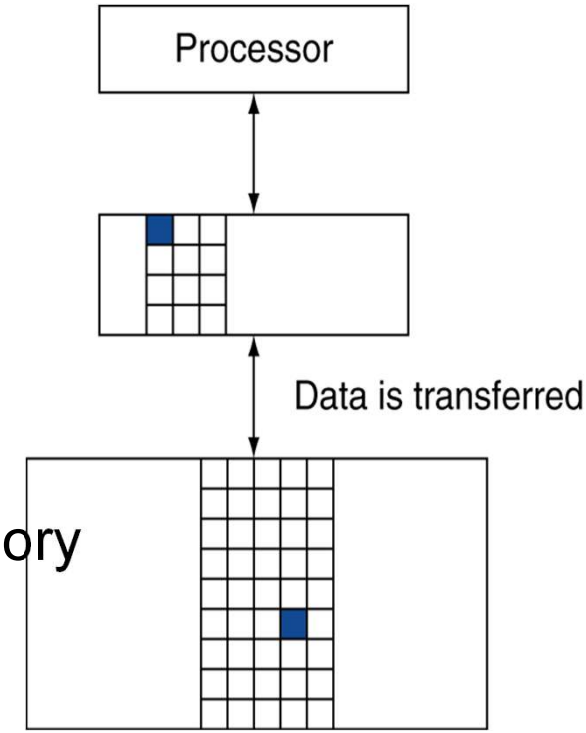
General Principles of Memory

- **Locality**
  - ***Temporal Locality*: referenced memory is likely to be referenced again soon (e.g. code within a loop)**
  - ***Spatial Locality*: memory close to referenced memory is likely to be referenced soon (e.g., data in a sequentially access array)**
- **Locality + smaller HW is faster = memory hierarchy**
  - ***Levels*: each smaller, faster, more expensive/byte than level below**
  - ***Inclusive*: data found in top also found in the bottom**
- **Definitions**
  - ***Upper*: closer to processor**
  - ***Block*: minimum unit that present or not in upper level**
  - **Block address: location of block in memory**
  - ***Hit time*: time to access upper level, including hit determination**

Cache: A definition

- **Webster's Dictionary :**
  - Cache (kash), a safe place for hiding or storing things
- **In CS (originally)**
  - The first level of memory hierarchy encounter after the CPU
- **In CS today (typically)**
  - A term applied to any level of buffering employed to reuse commonly accessed items

Cache Measures



- ***Hit rate*: fraction found in that level**
  - **So high that usually talk about *Miss rate***
  - **Miss rate fallacy:** as MIPS to CPU performance, miss rate to average memory access time in memory

- **Avg. memory−access time = Hit rate x Hit access time + Miss rate x Miss penalty**
- ***Miss penalty*: time to replace a block from lower level, including time to replace in CPU**
  - *access time*: time to lower level = (lower level latency)
  - *transfer time*: time to transfer block = (BW upper & lower, block size)

Speculative and multithreaded processors may execute other instructions during a miss (reduce performance impact of misses)

Cache

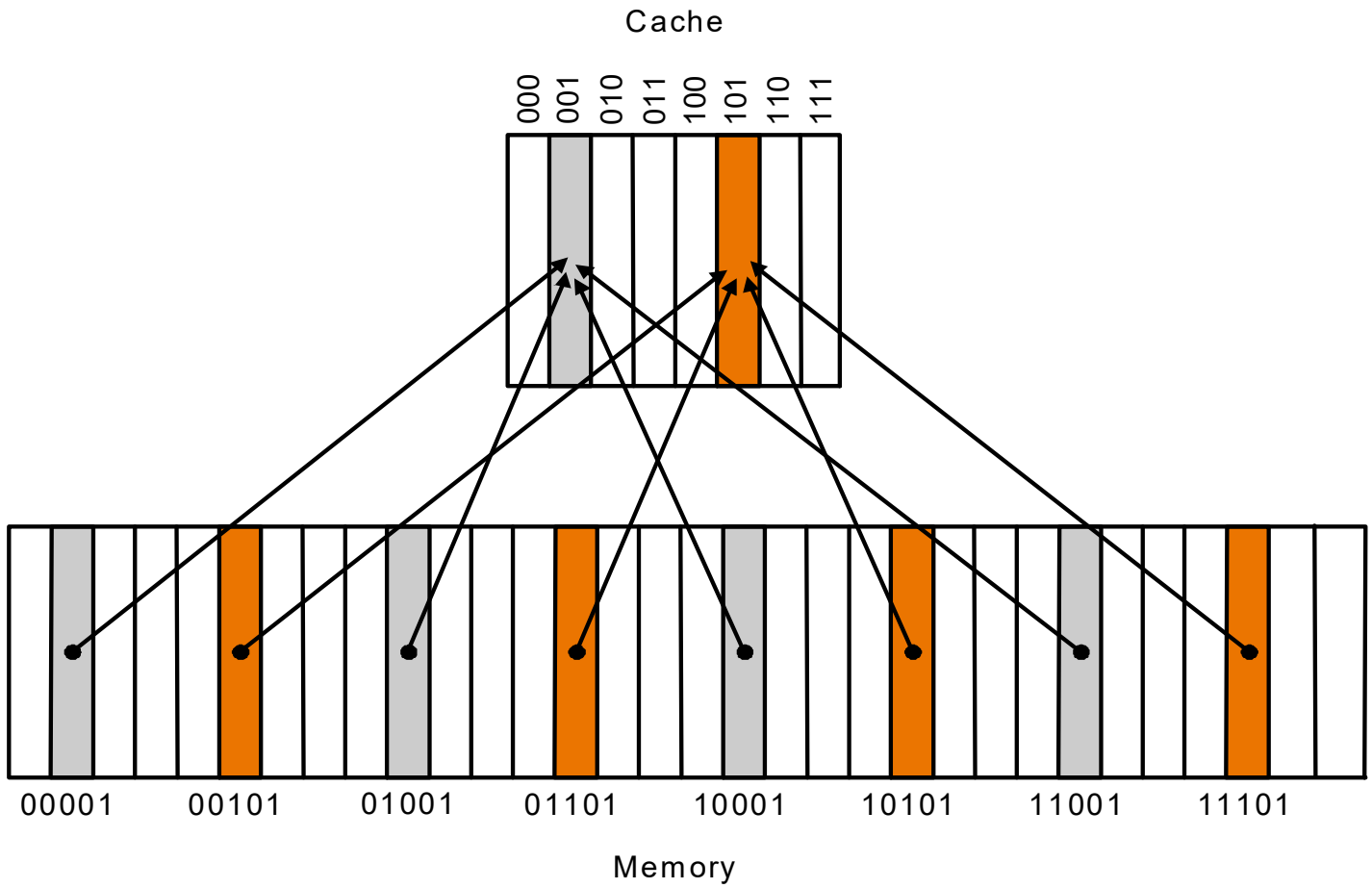
- **Two issues**
  - **How do we know if a data item is in the cache?**
  - **If it is, how do we find it?**
- **Our first example**
  - **Block size is one word of data**
  - **”Direct mapped"**

For each item of data at the lower level, there is exactly one location in the cache where it might be.

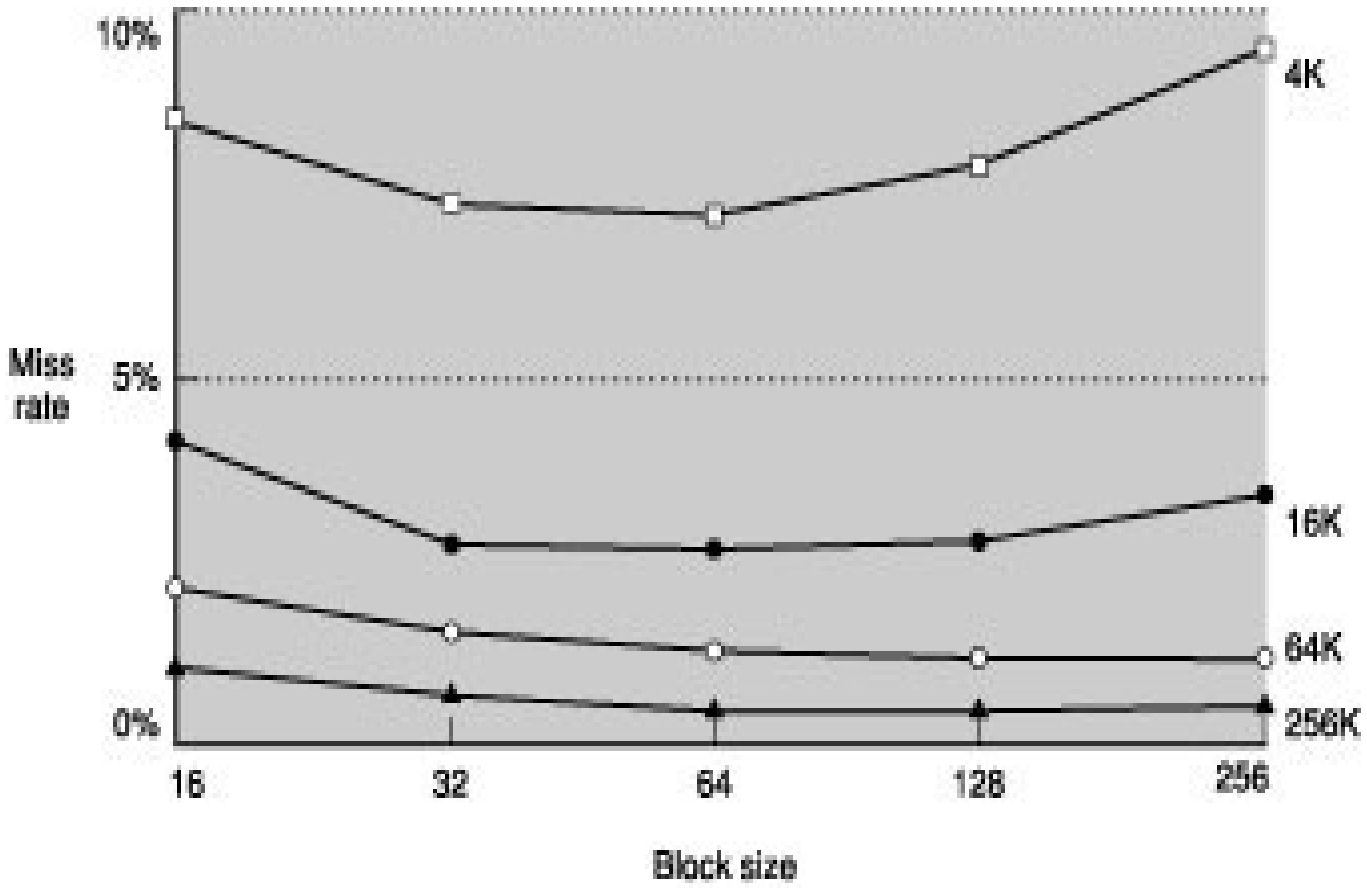
e.g., lots of items at the lower level share locations in the upper level

Direct mapped cache

- **Mapping**
  - **Cache address is Memory address modulo the number of blocks in the cache**
  - **(Block address) modulo (#Blocks in cache)**

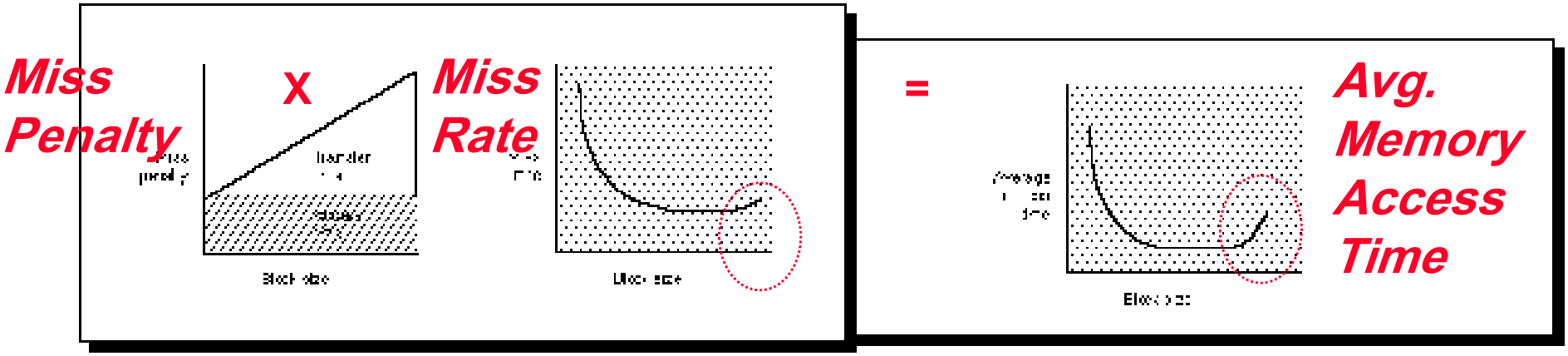


Block Size vs. Performance



Block Size vs. Cache Measures

- **Increasing Block Size generally increases Miss Penalty and decreases Miss Rate**



Block Size **Block Size** **Block Size**

Avg. memory-access time = Hit rate x Hit access time
+ Miss rate x Miss penalty

Implications For CPU

- Fast hit check since every memory access
  - Hit is the common case
- Unpredictable memory access time
  - 10s of clock cycles: wait
  - 1000s of clock cycles:
    - Interrupt & switch & do something else
    - New style: multithreaded execution

Four Questions for Memory Hierarchy Designers

- **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)***

Q1: Where can a block be placed in the upper level?

- **Direct Mapped:** Each block has only one place that it can appear in the cache**.**
- **Fully associative:** Each block can be placed anywhere in the cache**.**
- **Set associative:** Each block can be placed in a restricted set of places in the cache.
  - **If there are n blocks in a set, the cache placement is called n-way set associative**
- **What is the associativity of a direct mapped cache?**

Associativity Examples

(Figure 5.2, pg. 376)

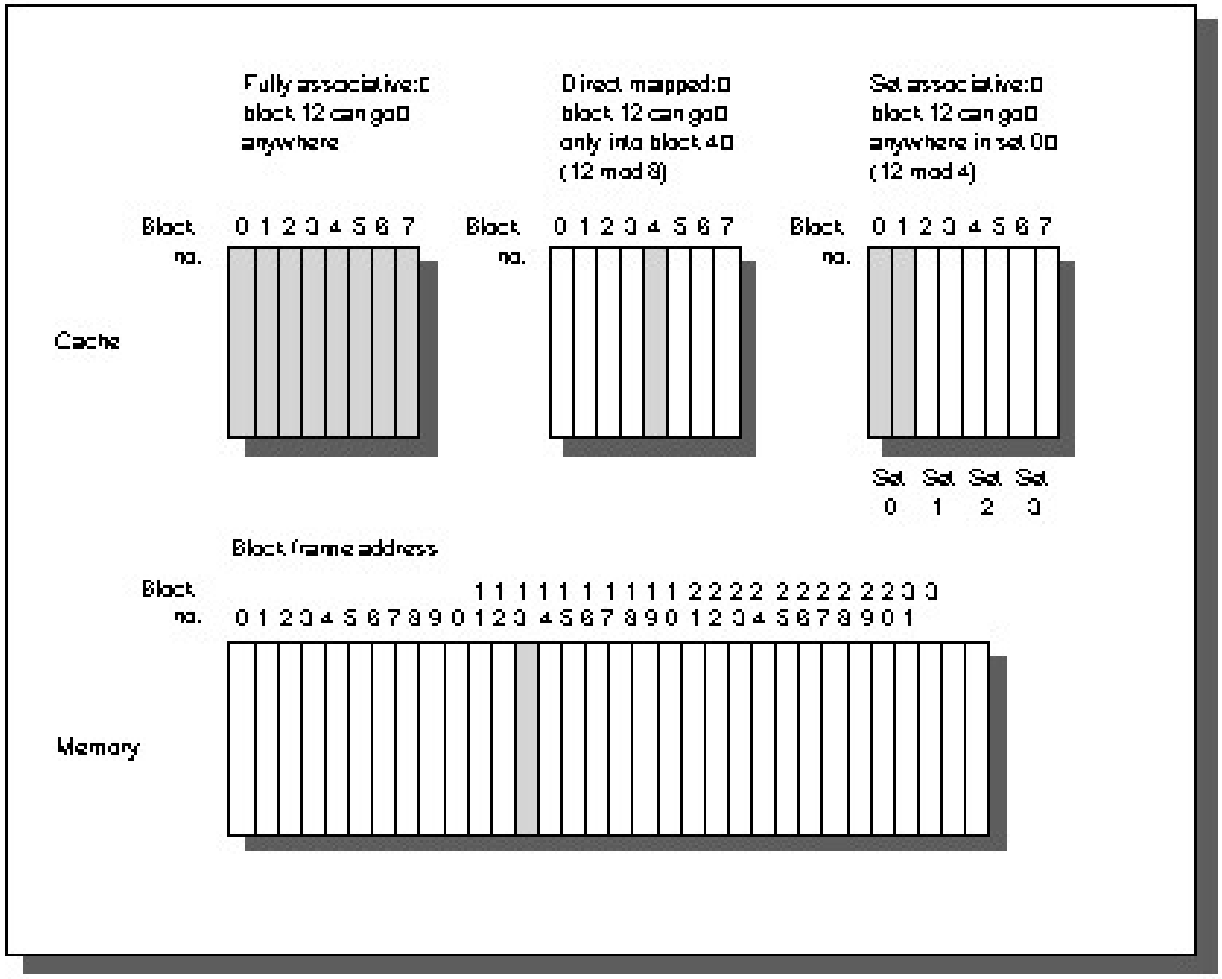


FIGURE 5.2 This example cache has eight block frames and memory has 32 blocks.

Fully associative:

Block 12 can go anywhere

Direct mapped:

Block no. = (Block address) mod (No. of blocks in cache)

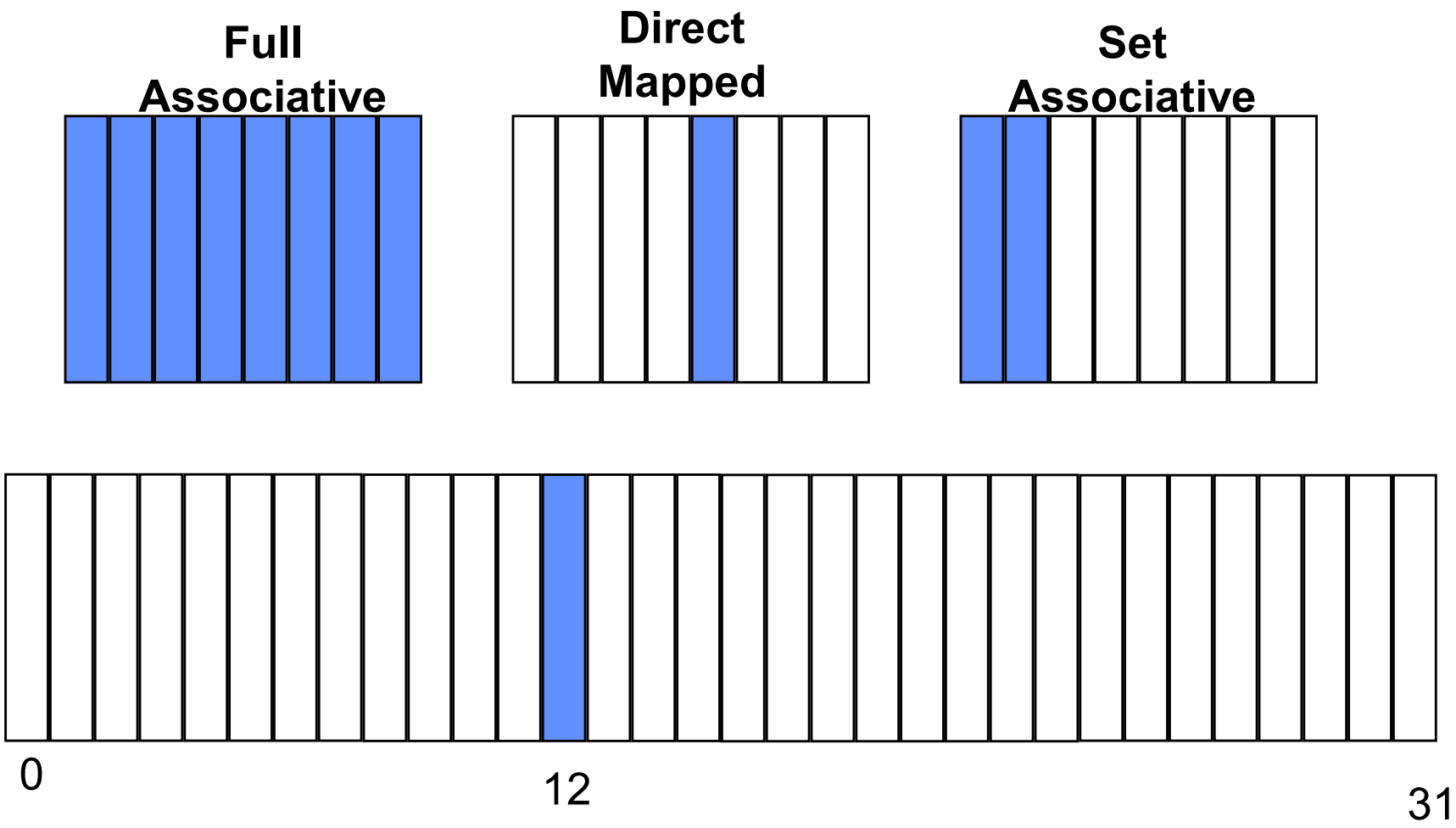
Block 12 can go only into block 4 (12 mod 8)

Set associative:

Set no. = (Block address) mod (No. of sets in cache)

Block 12 can go anywhere in set 0 (12 mod 4)

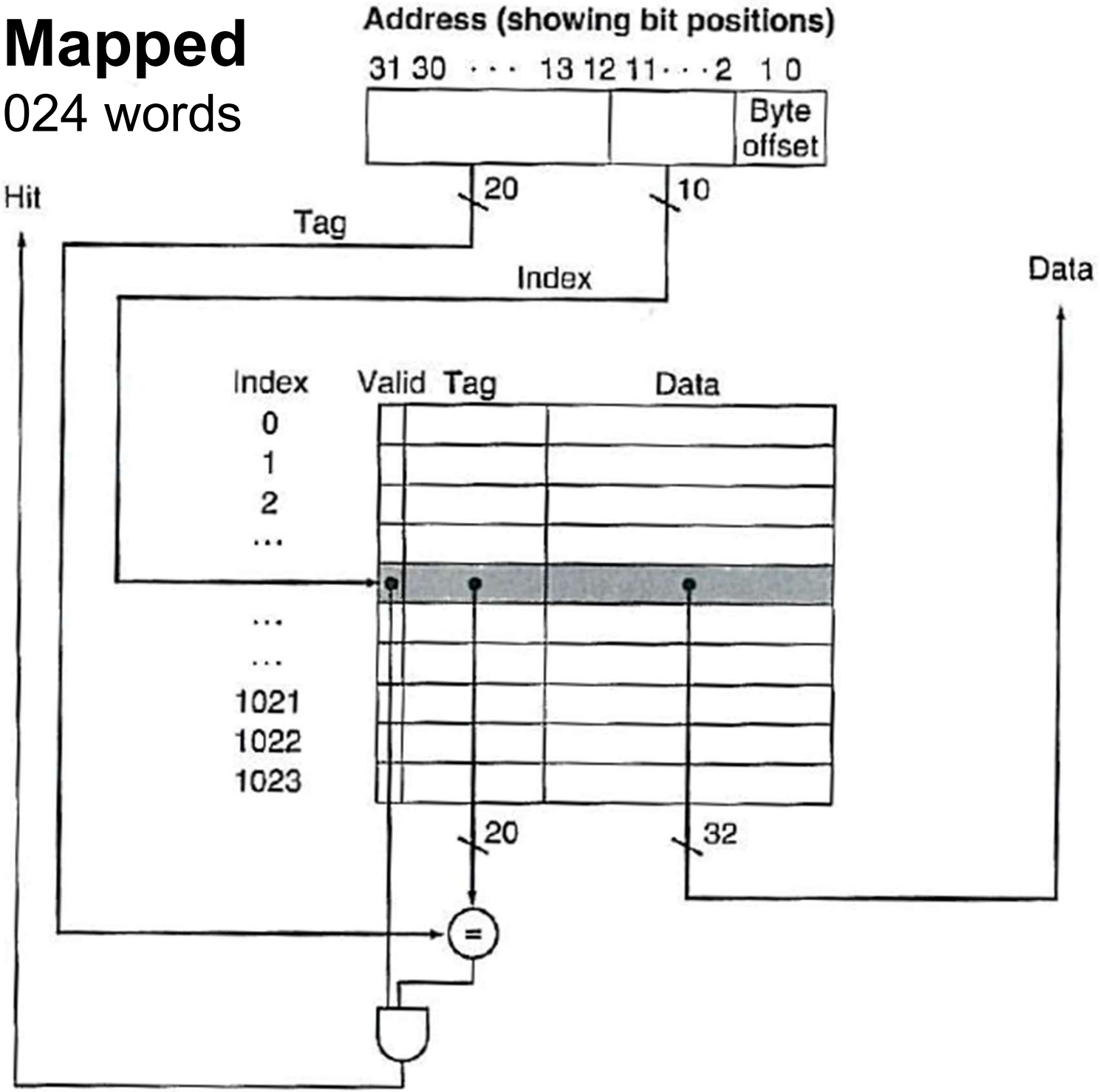
Q1: Where can a block be placed in the upper level?



Block 12 placed in 8 block cache

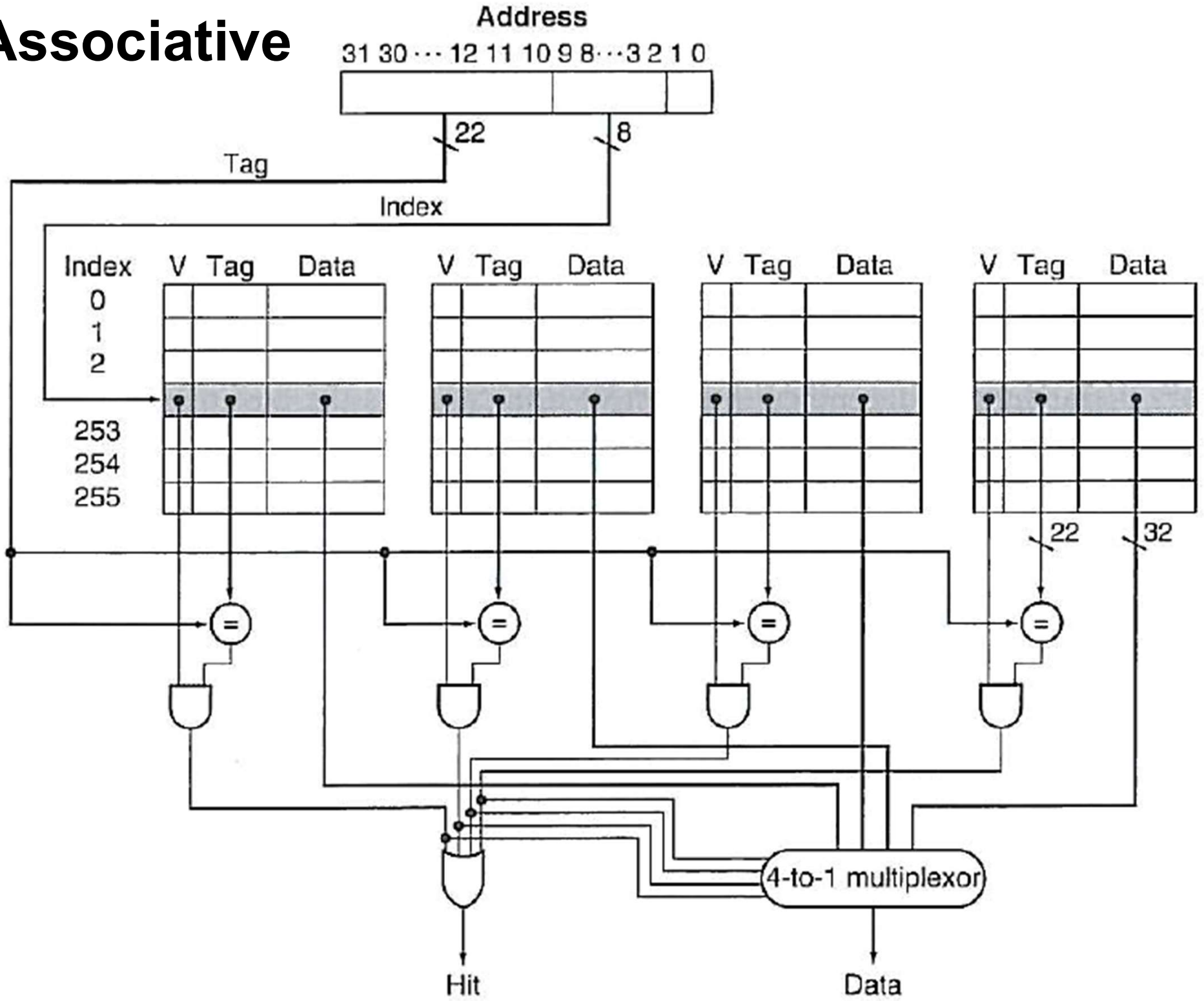
Direct Mapped

4KB or 1024 words



Set Associative

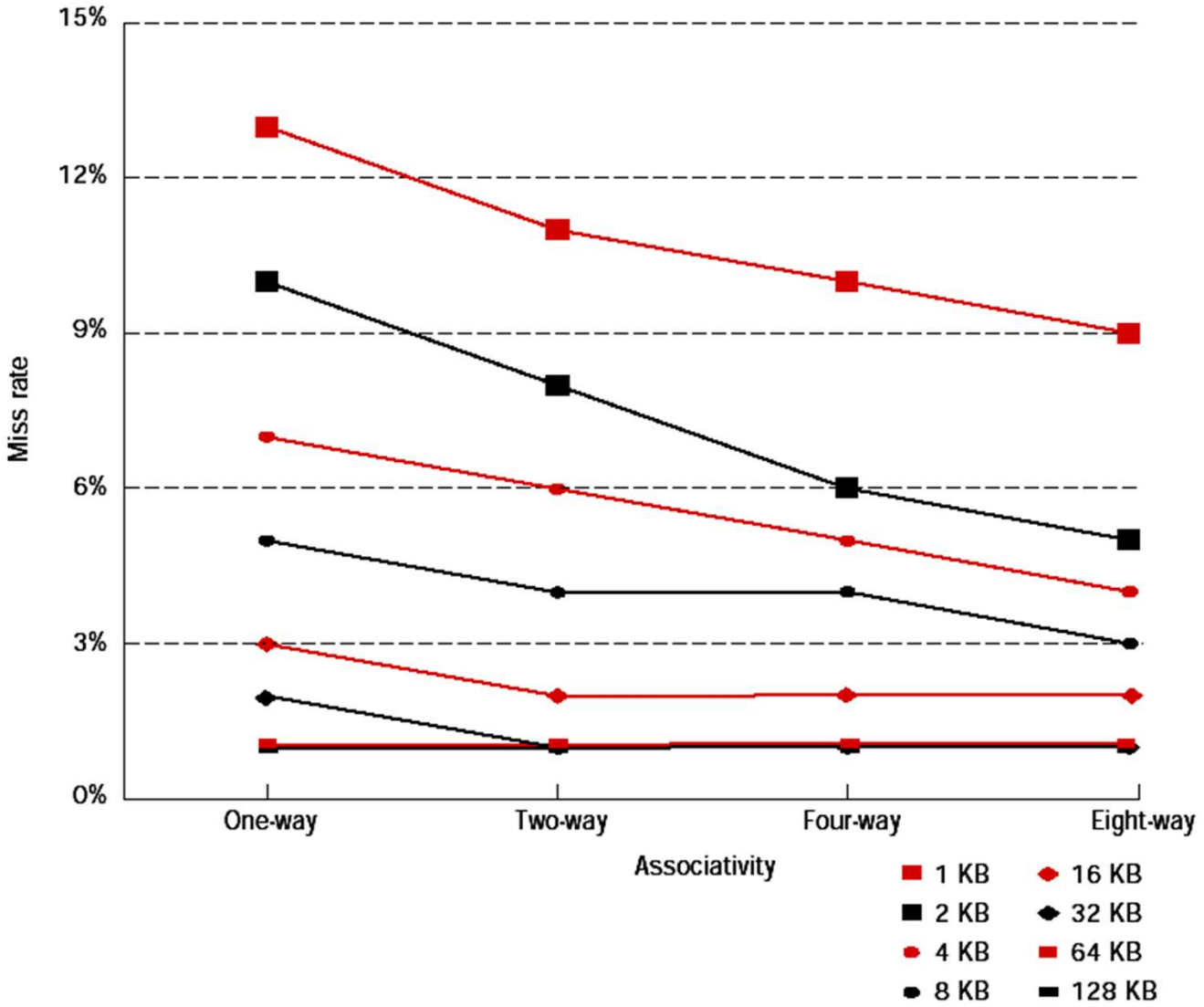
4-way



Fully Associative

- **Can you draw it?**
- **How many comparators will you need?**

Associativity Performance



Q2: How Is a Block Found if it is in the Upper Level?

- The address can be divided into two main parts
  - Block offset: selects the data from the block

    offset size = $\log_2$(block size)
  - Block address: tag + index
    - index: selects set in cache

      index size = $\log_2$(#blocks/associativity)
  - tag: compared to tag in cache to determine hit

    tag size = addreess size - index size - offset size

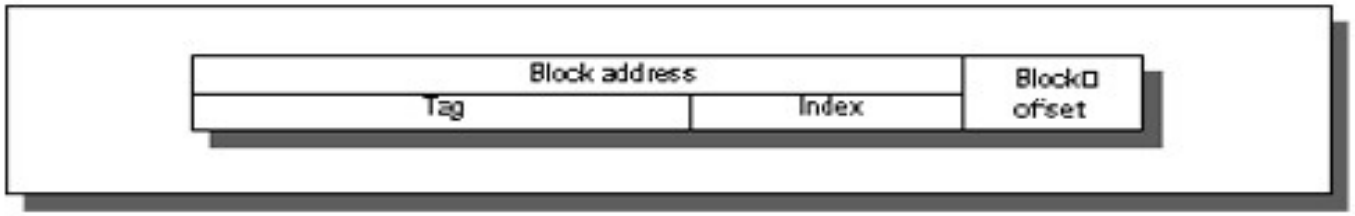
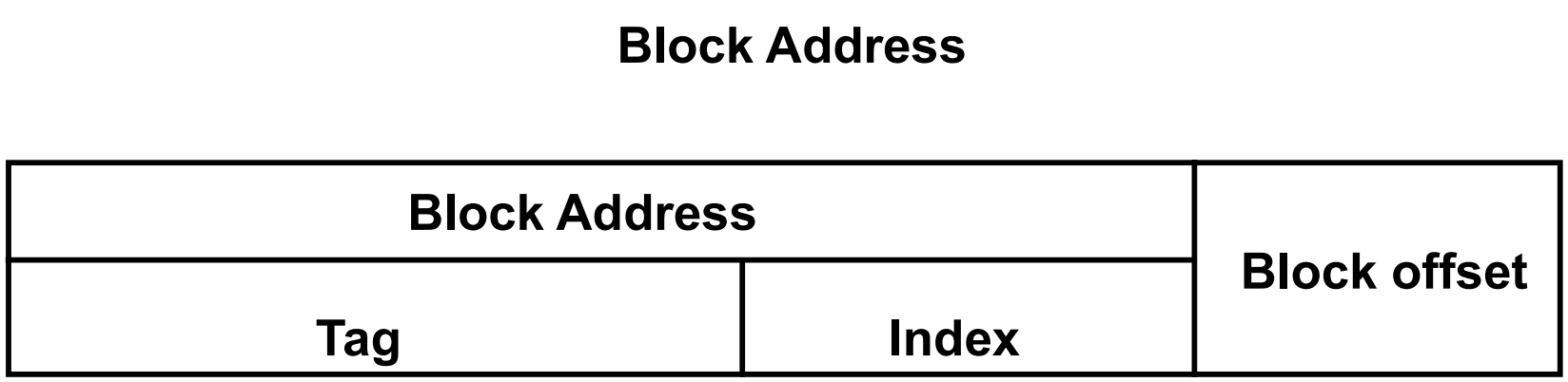


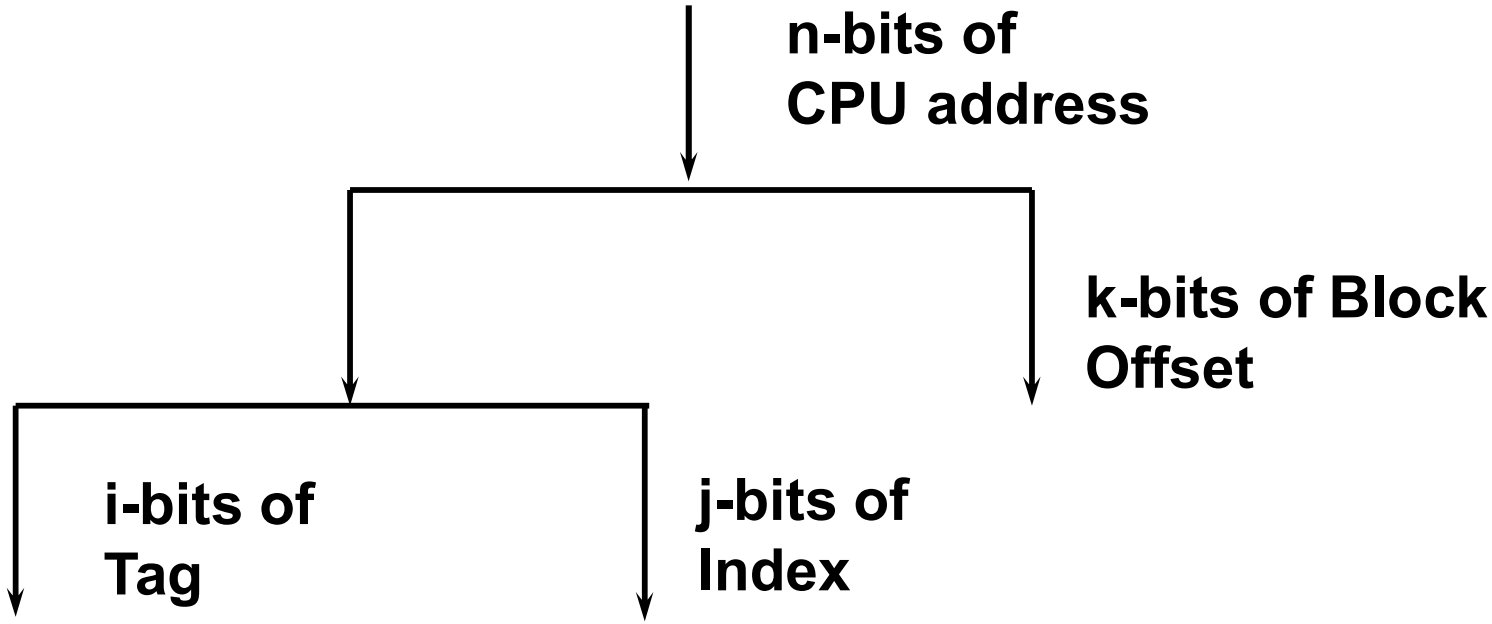
FIGURE 5.3 The threeportionsof an addressin a set-associativeor direct-mapped cache.

Q2: How Is a Block Found If It Is in the Upper Level?

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

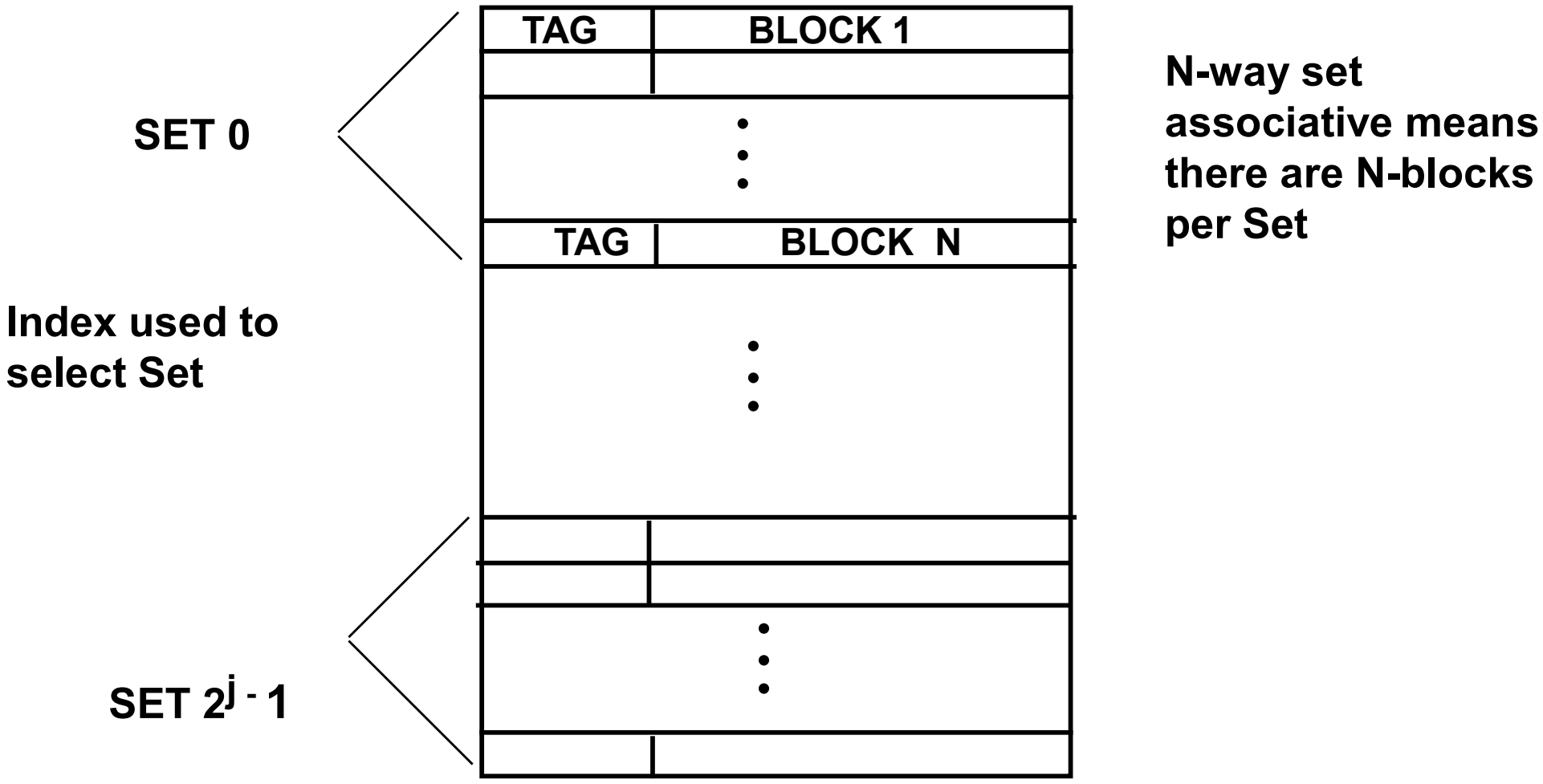


Address organization for Set Associative Cache



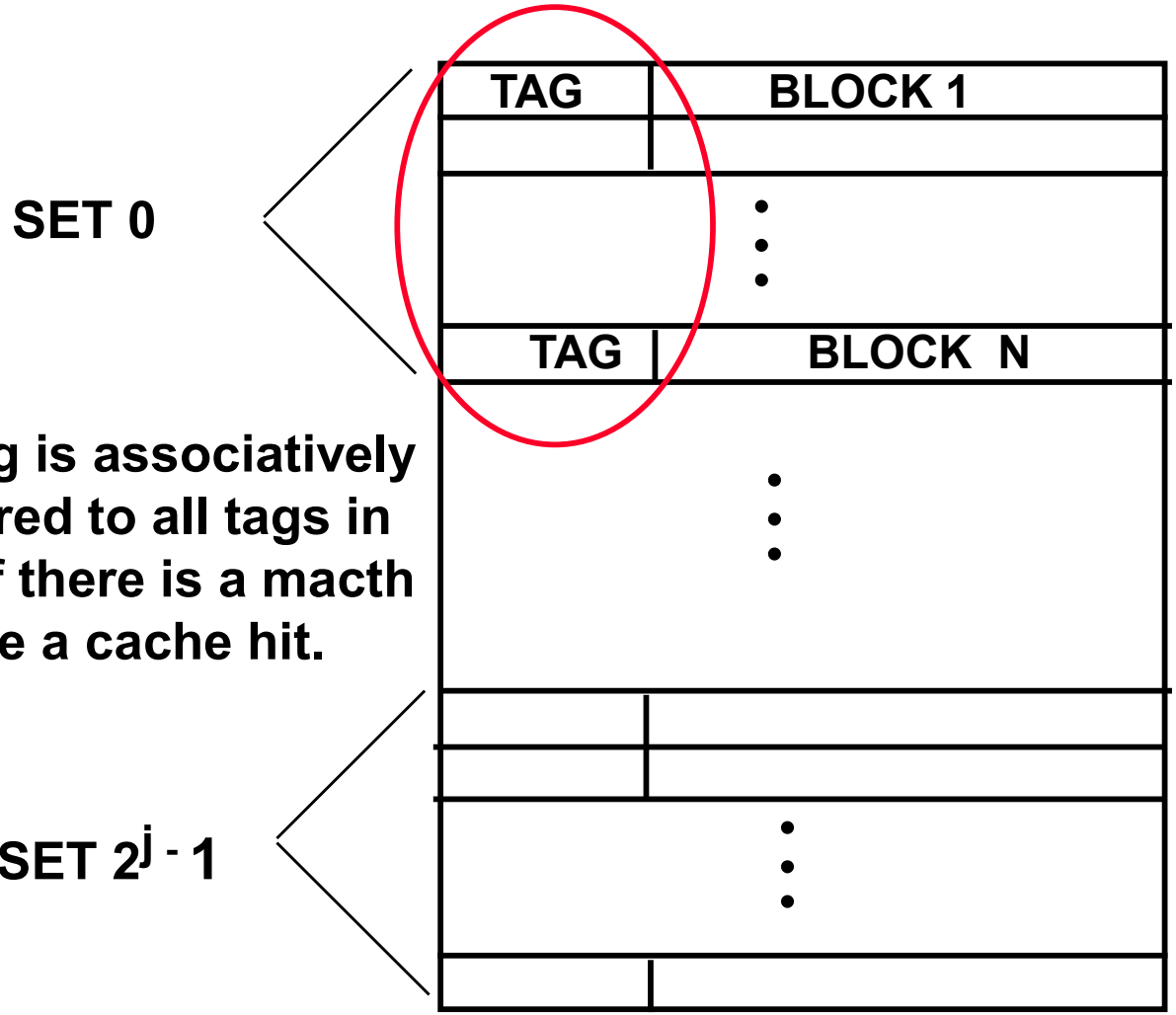
n= i + j + k

Cache organization



N-way set associative means there are N-blocks per Set

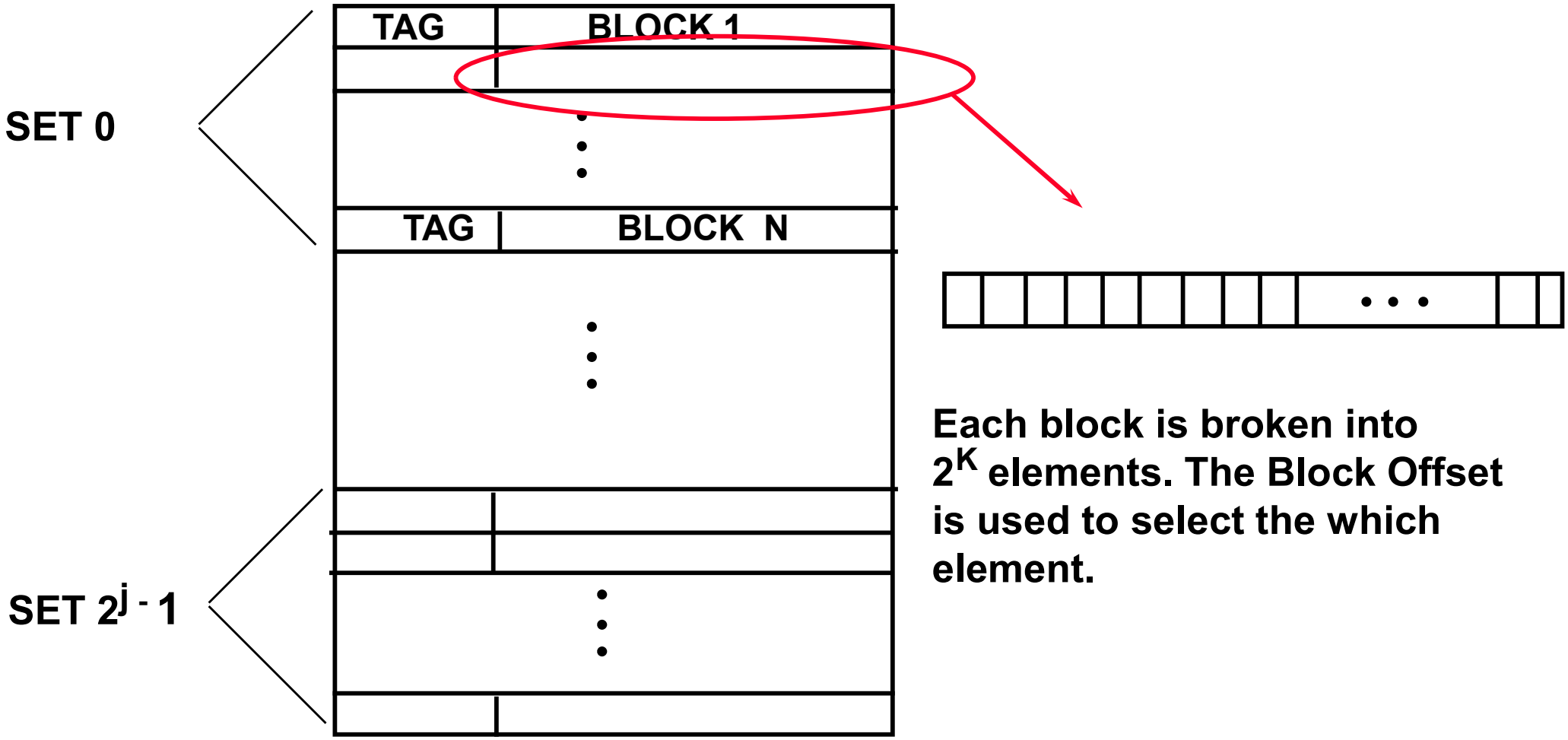
Cache organization



The Tag is associatively compared to all tags in a set. If there is a macth we have a cache hit.

There must also be a mechanism to indicate invalid block data. This is commonly done by attaching a valid bit to the Tag field (not shown).

Cache organization



Each block is broken into 2^K elements. The Block Offset is used to select the which element.

The Tag and Index identify the block & the Block Offset identifies the element.

Q3: Which Block Should be Replaced on a Miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random - easier to implement
  - Least Recently used - harder to implement - may approximate
- Miss rates for caches with different size, associativity and replacemnt algorithm.

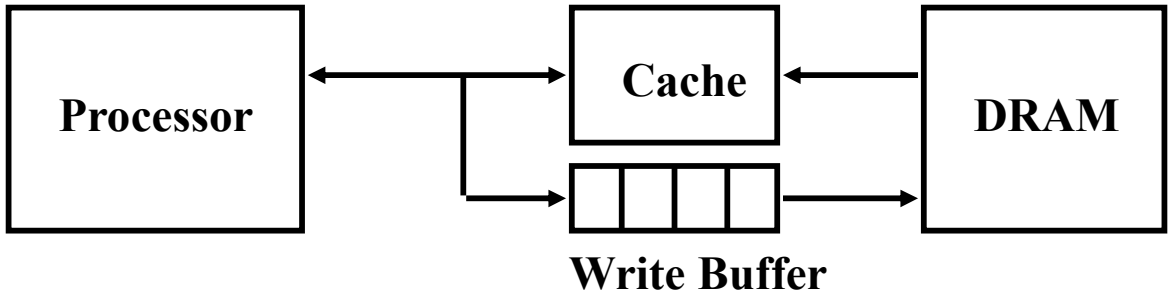
Associativity:	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.18%	5.69%	4.67%	5.29%	4.39%	4.96%
64 KB	1.88%	2.01%	1.54%	1.66%	1.39%	1.53%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

For caches with low miss rates, random is almost as good as LRU.

Q4: 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? (add a dirty bit to each block)**
- **Pros and Cons of each:**
  - **Write through**
    - **read misses cannot result in writes to memory,**
    - **easier to implement**
    - **Always combine with write buffers to avoid memory latency**
  - **Write back**
    - **Less memory traffic**
    - **Perform writes at the speed of the cache**

Write Buffer for Write Through



- 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:
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle
- Memory system designer's nightmare:
  - Store frequency (w.r.t. time) -> 1 / DRAM write cycle
  - Write buffer saturation

Q4: What Happens on a Write?

	Write-Through	Write-Back
Policy	Data written to cache block also written to lower-level memory	Write data only to the cache Update lower level when a block falls out of the cache
Debug	Easy	Hard
Do read misses produce writes?	No	Yes
Do repeated writes make it to lower level?	Yes	No

Hits & Misses (Read v.s. Write)

- **Read hits**
  - **This is what we want!**
- **Read misses**
  - **Stall the CPU, fetch block from memory, deliver to cache, restart**
- **Write hits**
  - **Can replace data in cache and memory (write-through)**
  - **Write the data only into the cache (write-back the cache later)**
- **Write misses**
  - **See next slide**

What happens on a Write Miss?

Write Allocate vs Non-Allocate

- 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!

Split vs. Unified Cache

- **Unified cache (mixed cache):** Data and instructions are stored together (von Neuman architecture)
- **Split cache:** Data and instructions are stored separately (Harvard architecture)
- **Why do instructions caches have a lower miss ratio?**

Size	Instruction Cache	Data Cache	Unified Cache
1 KB	3.06%	24.61%	13.34%
2 KB	2.26%	20.57%	9.78%
4 KB	1.78%	15.94%	7.24%
8 KB	1.10%	10.19%	4.57%
16 KB	0.64%	6.47%	2.87%
32 KB	0.39%	4.82%	1.99%
64 KB	0.15%	3.77%	1.35%
128 KB	0.02%	2.88%	0.95%

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 (later).**

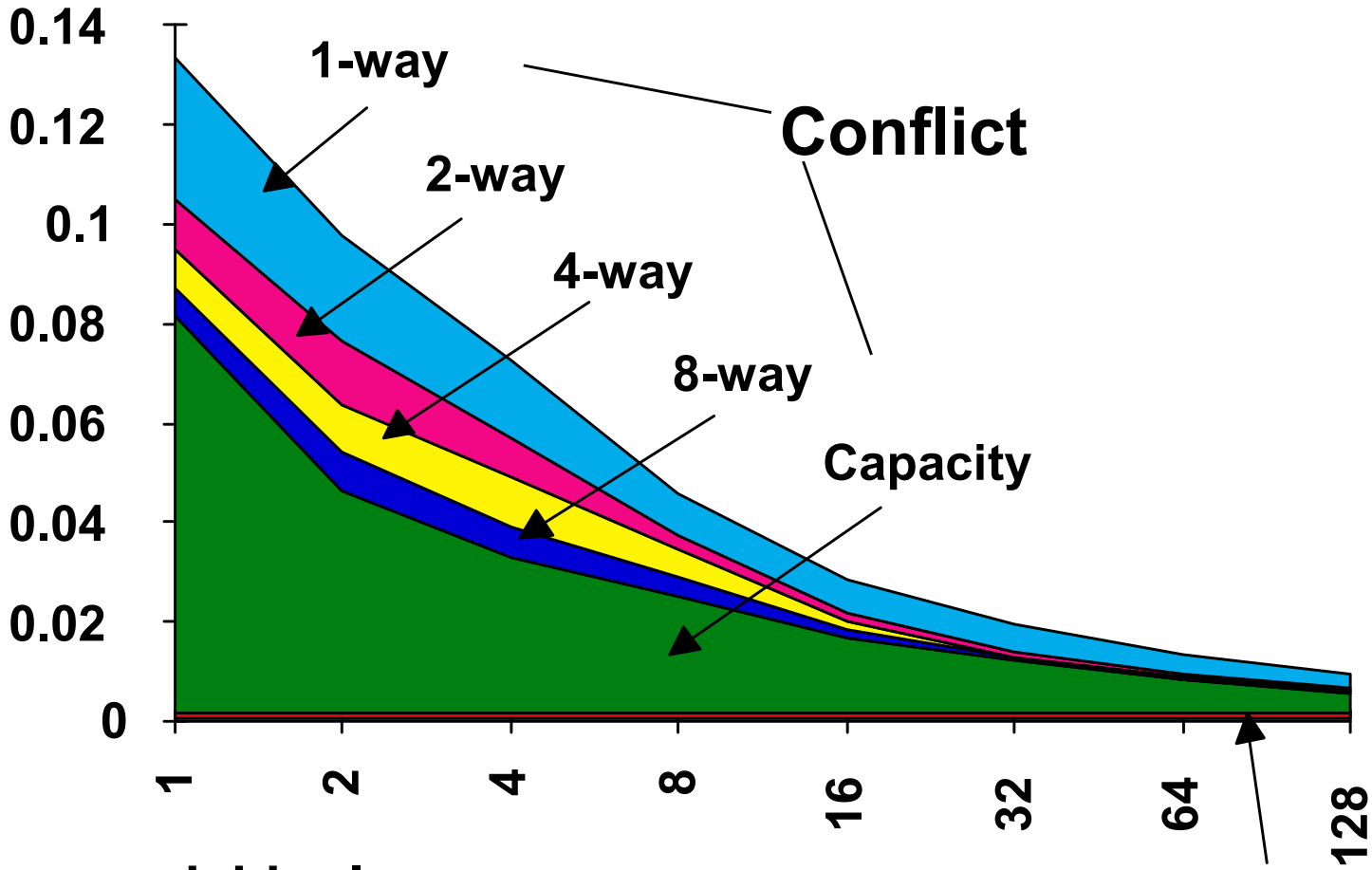
Types of misses

- **Compulsory**
  - Very first access to a block (cold-start miss)
- **Capacity**
  - Cache cannot contain all blocks needed
- **Conflict**
  - Too many blocks mapped onto the same set

How do you solve

- **Compulsory misses?**
  - Larger blocks with a side effect!
- **Capacity misses?**
  - Not much options: enlarge the cache otherwise face “thrashing!”, computer runs at a speed of the lower memory or slower!
- **Conflict misses?**
  - Full associative cache with a cost of hardware and may slow the processor!

3Cs Absolute Miss Rate (SPEC92)

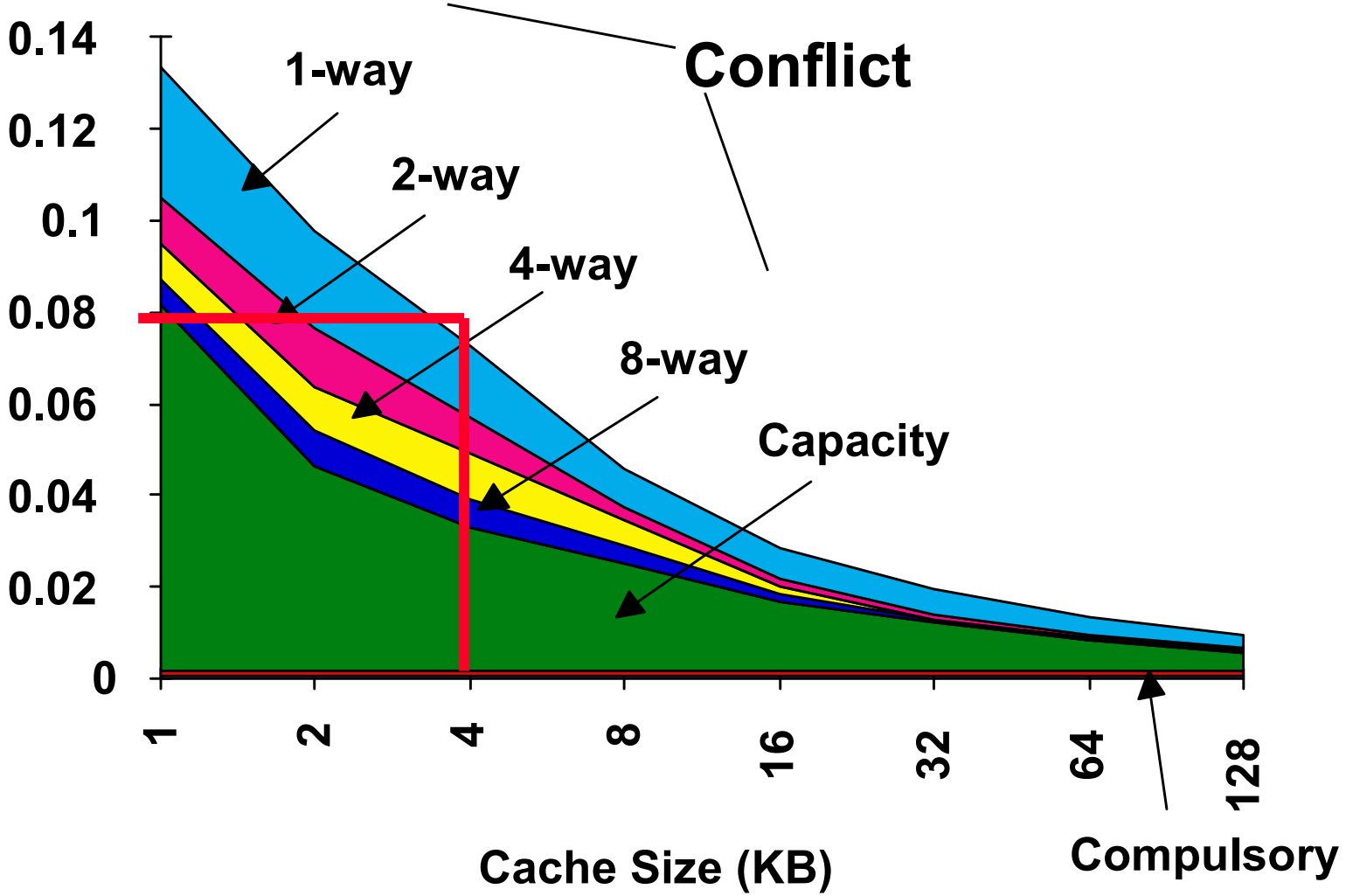


Compulsory vanishingly small

2:1 Cache Rule

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

Or an old rule of thumb: 2x size => 25% cut in 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?

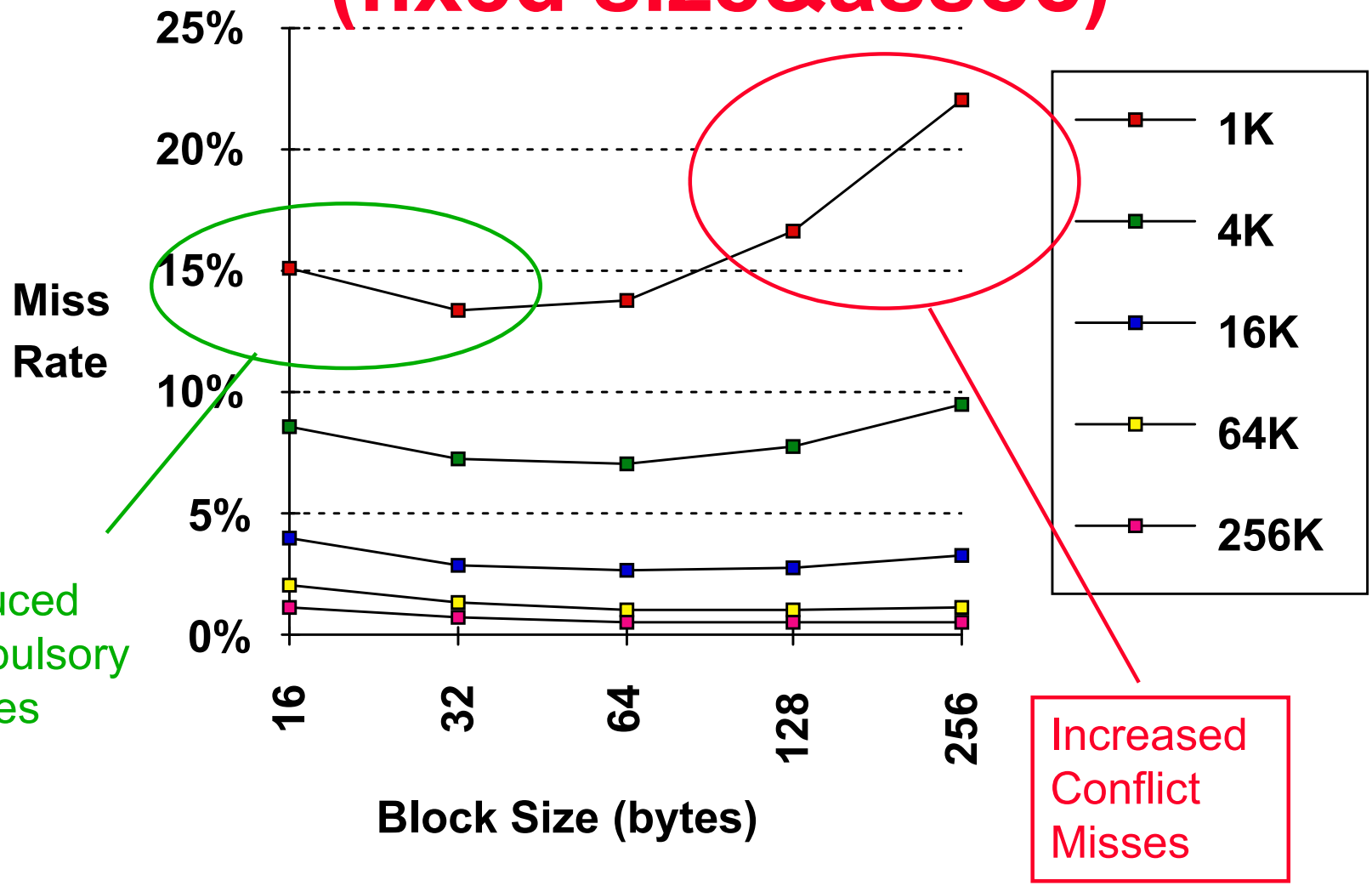
2) Change Associativity:
   Which of 3Cs is obviously affected?

3) Change Compiler:
   Which of 3Cs is obviously affected?

Basic cache optimizations:

- **Larger block size**
  - Reduces compulsory misses
  - Increases capacity and conflict misses, increases miss penalty
- **Larger total cache capacity to reduce miss rate**
  - Increases hit time, increases power consumption
- **Higher associativity**
  - Reduces conflict misses
  - Increases hit time, increases power consumption
- **Higher number of cache levels**
  - Reduces overall memory access time
- **Giving priority to read misses over writes**
  - Reduces miss penalty

Larger Block Size (fixed size&assoc)



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

- **CCT = Clock Cycle Time**
- **Example:**

Assume that

CCT(2-way) = 1.10 CCT(1-way)
CCT(4-way) = 1.12 CCT(1-way)
CCT(8-way) = 1.14 CCT(1-way)

Example: Avg. Memory Access Time vs. Miss Rate (cont.)

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

AMAT for each cache size/organization (Red means A.M.A.T. not improved by more associativity)

Other Optimizations: Victim Cache

- Add a small fully associative victim cache to place data discarded from regular cache
- When data not found in cache, check victim cache
- 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Get access time of direct mapped with reduced miss rate

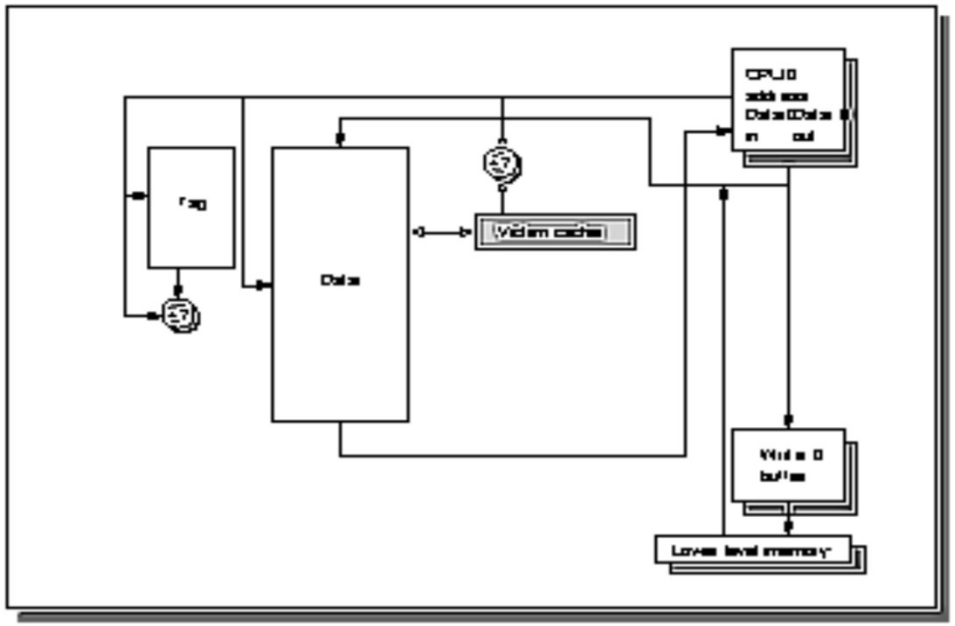
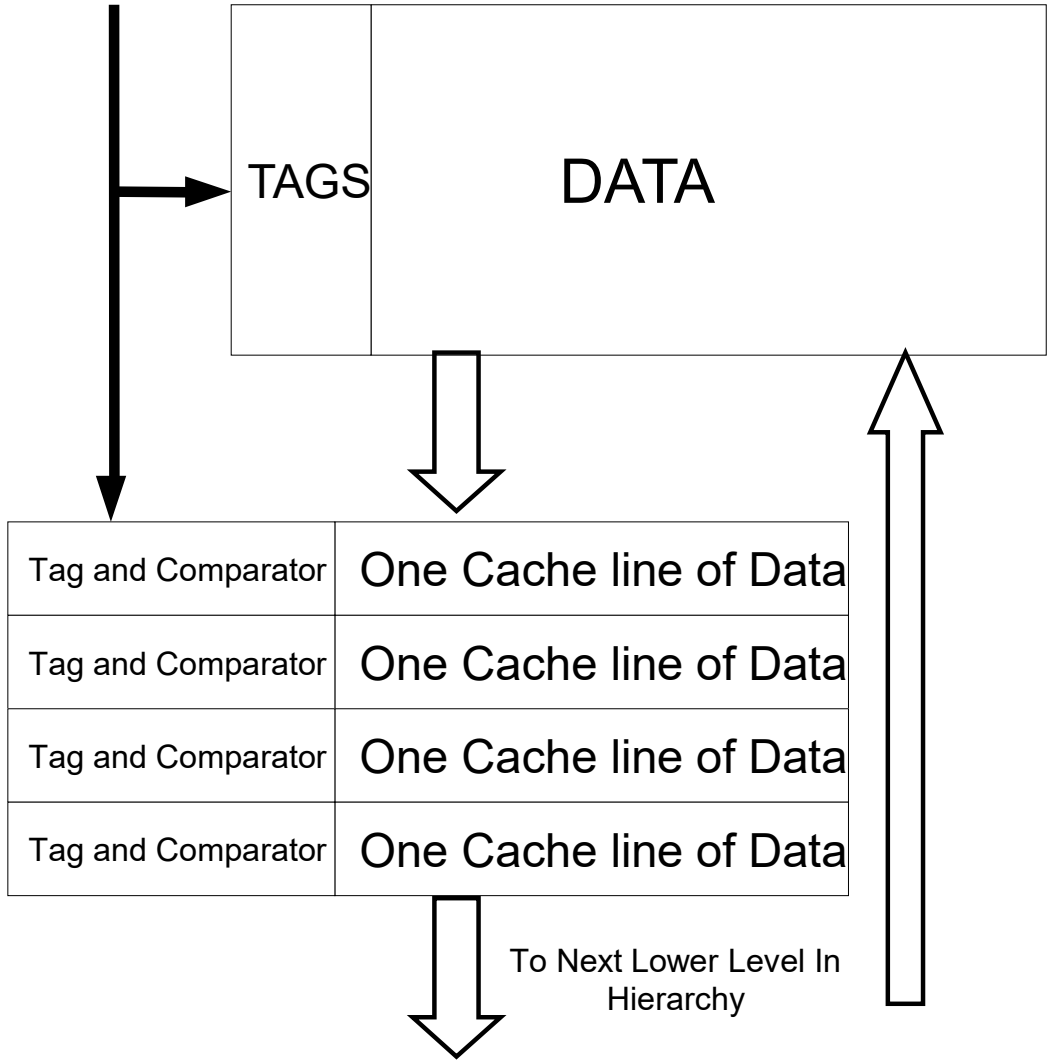


FIGURE 5.15 Placement of victim cache in the memory hierarchy.

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 associative cache?
- Two cahces H1 and H2 caches
- Place Block In H1 if Miss check H2 if hit swap if miss bring from memory to H1 and send block in H1 to H2.
- Divide cache: on a miss, check other half of cache to see if the data is there, if so have a pseudo-hit (slow hit)

- Drawback: Difficult to build a CPU pipeline if hit may take either 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 & Data

- E.g., Instruction Prefetching
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in "stream buffer"
  - On miss check stream buffer
  - IBM POWER4 has 8 data prefetch streams
- 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

- Compiler or hand inserted prefetch instruction
- Data Prefetch
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Nonfaulting prefetching instructions cannot cause faults. They are 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!

Non-binding is more common.

Software pre-fetching Example

```
for (int i=0; i<1024; i++)
{
    A[i] = A[i] + 100;
}
```

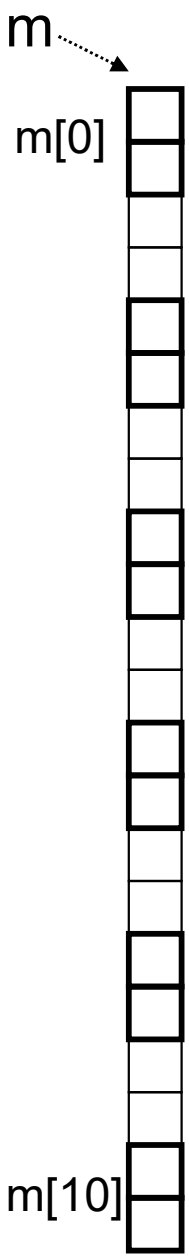
- Each iteration, the i^{th} element of the array is accessed.
- We can prefetch the elements that are going to be accessed in future iterations
  - inserting a "prefetch" instruction as shown below:

```
for (int i=0; i<1024; i=i+k)
{
     prefetch (A[i + k]);
     A[i]   = A[i]  + 100;
     A[i+1] = A[i+1] + 100;
     A[i+2] = A[i+2] + 100;
     A[i+3] = A[i+3] + 100;
}
```

7. Reducing Misses by Compiler Optimizations

- 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

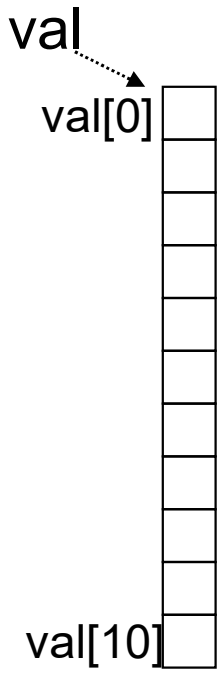
Merging Arrays Example

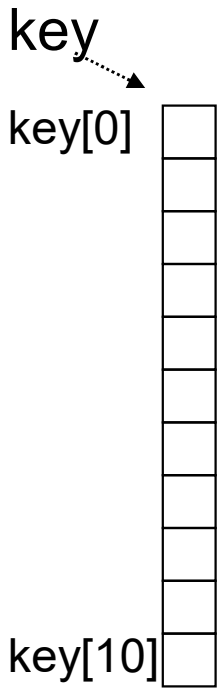



```
/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];

for (int i = 0; i < 11 ; i++)
     if ( key[i] %5 && val[i] >1000)
        printf("found key");

/* After: 1 array of structures */
struct merge {
  int val;
  int key;
};
struct merge m[SIZE];
for (int i = 0; i < 11 ; i++)
     if ( m[i].key %5 && m[i].val >1000)
        printf("found key");
```

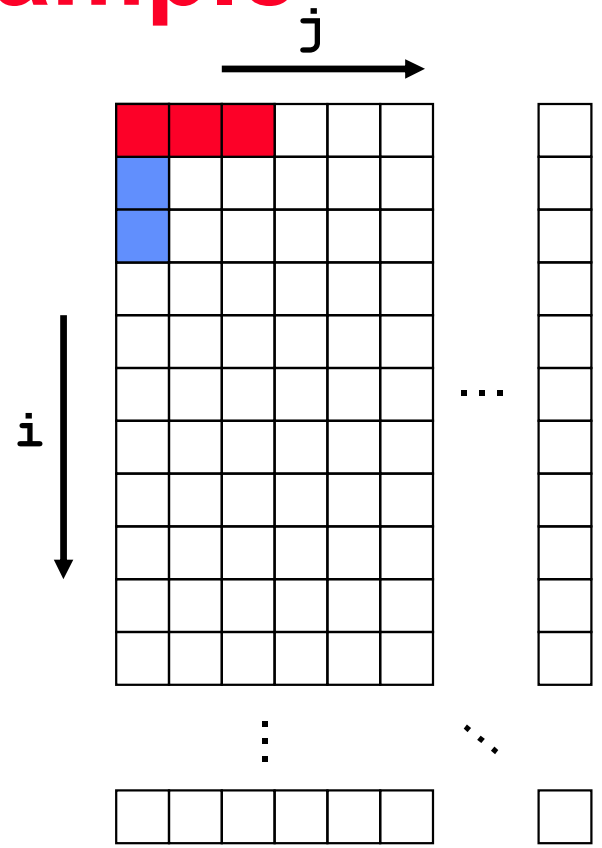




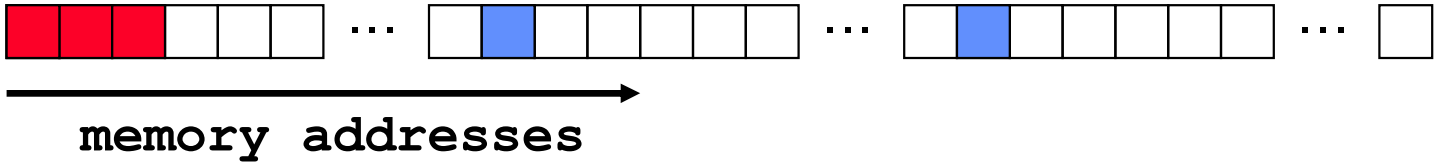
Reducing conflicts between val & key; improve spatial locality

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

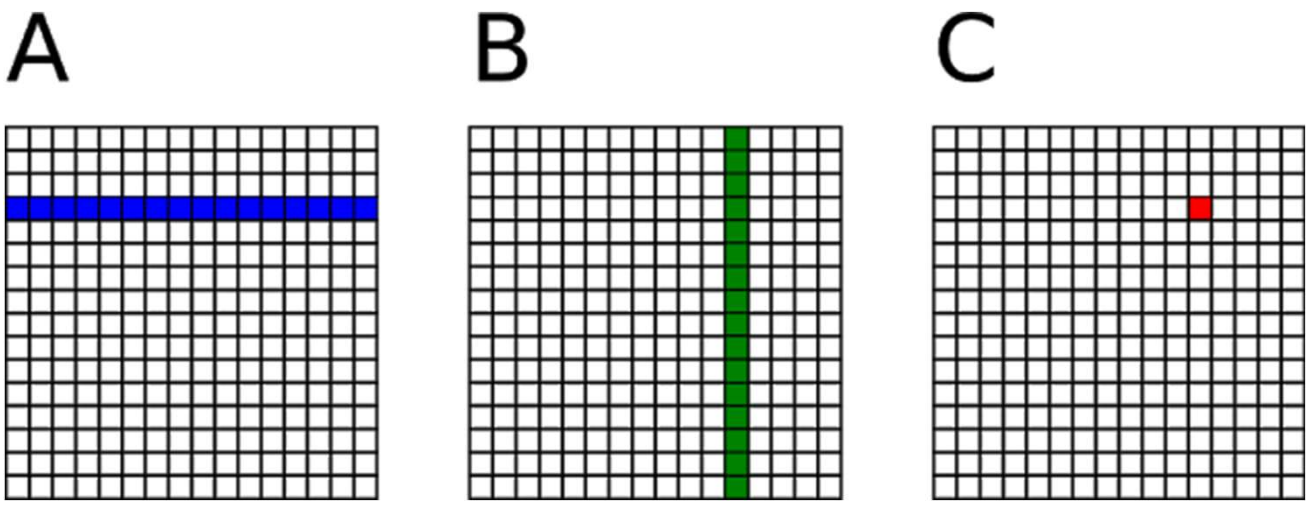
. Problem: When accessing multiple multi-dimensional arrays (e.g., for matrix multiplication), capacity misses occur if not all of the data can fit into the cache.

. Solution: Divide the matrix into smaller submatrices (or blocks) that can fit within the cache

. The size of the block chosen depends on the size of the cache

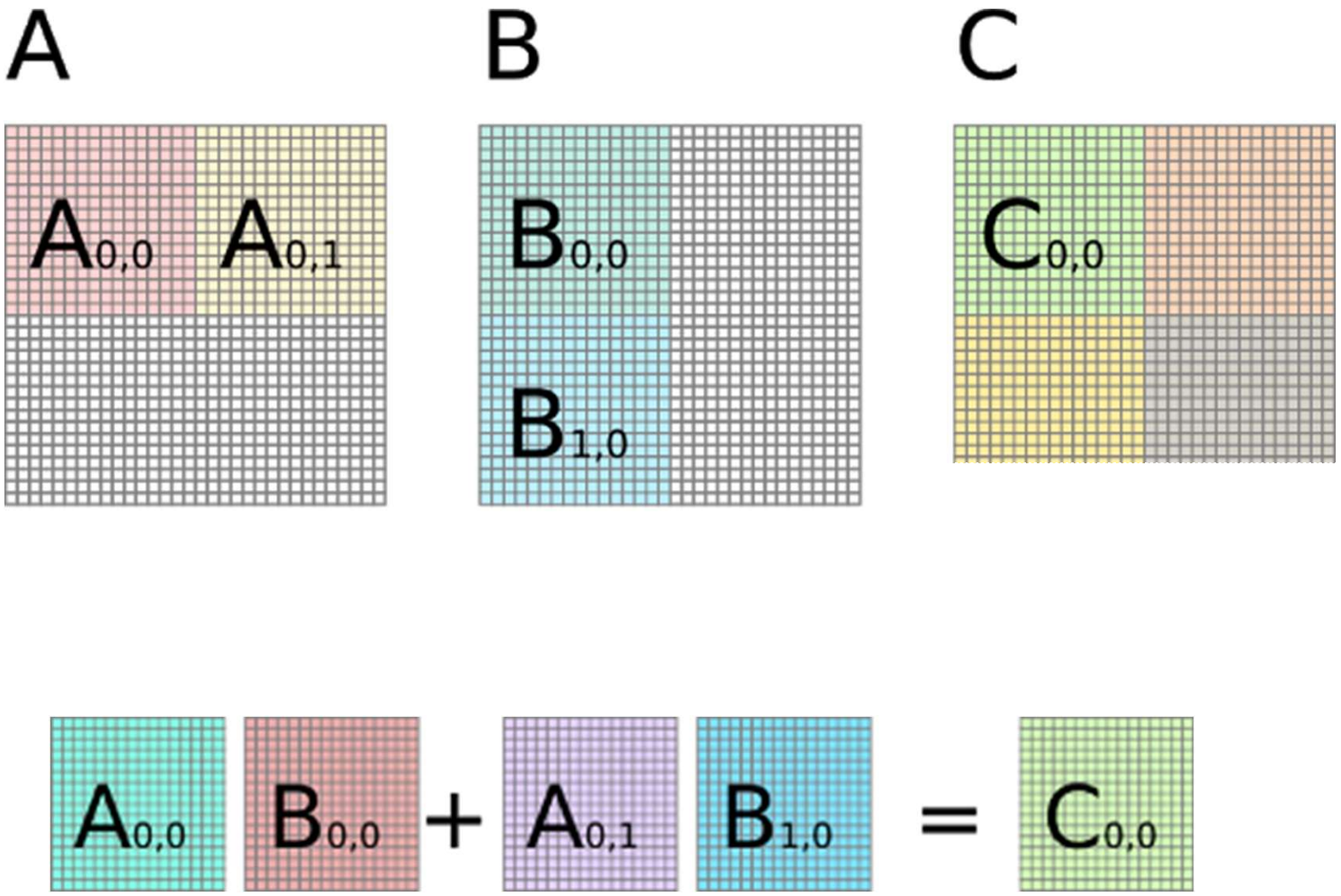
. Blocking can only be used for certain types of algorithms

Matrix Multiplication

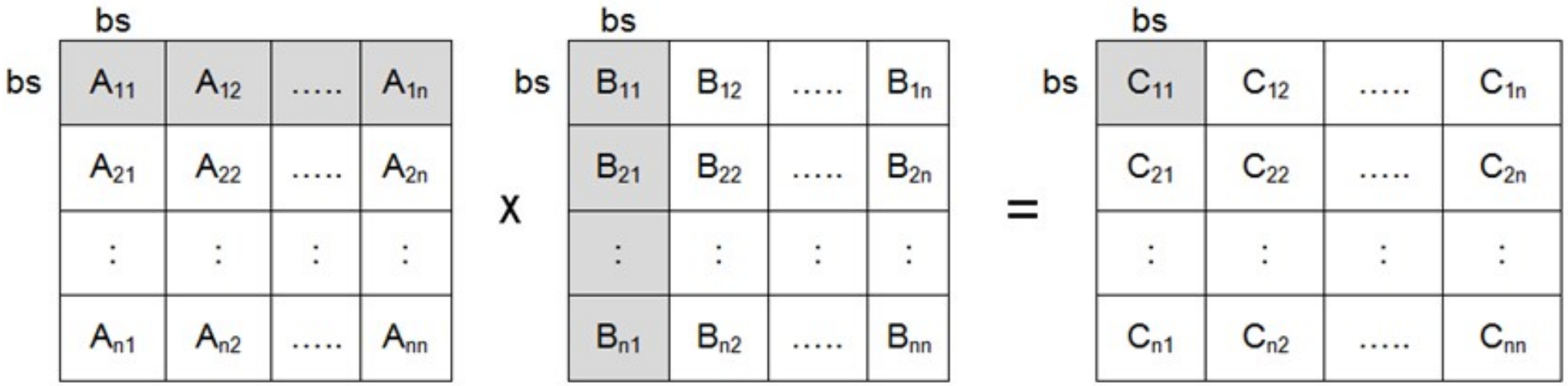



```
for (int i = 0; i < N; i++)
  for (int j = 0 ; j < N ; j++)
     for (int k = 0 ; k <N; k++)
        C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

Blocked Matrix Multiplication

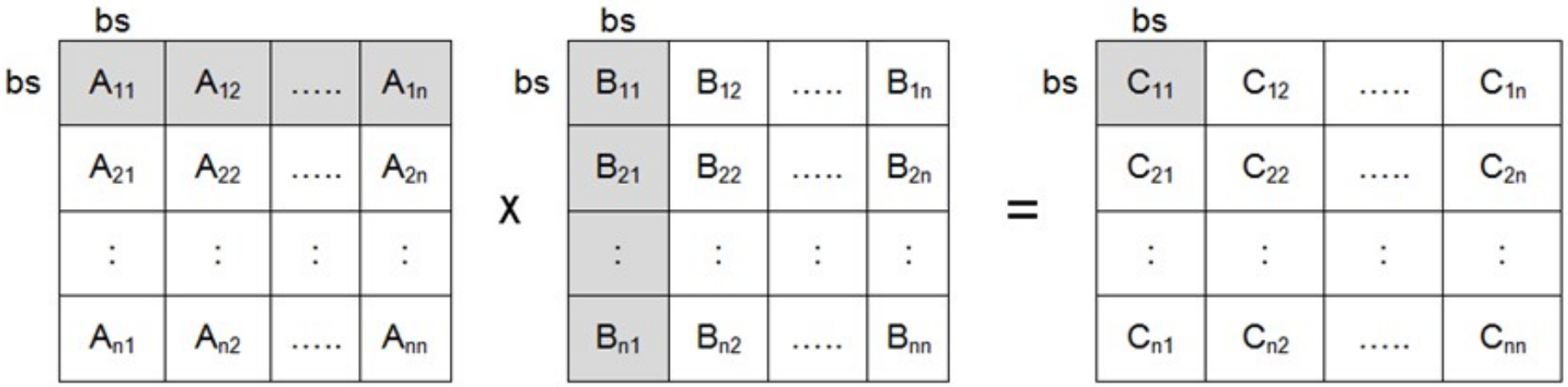


Blocked Matrix Multiplication



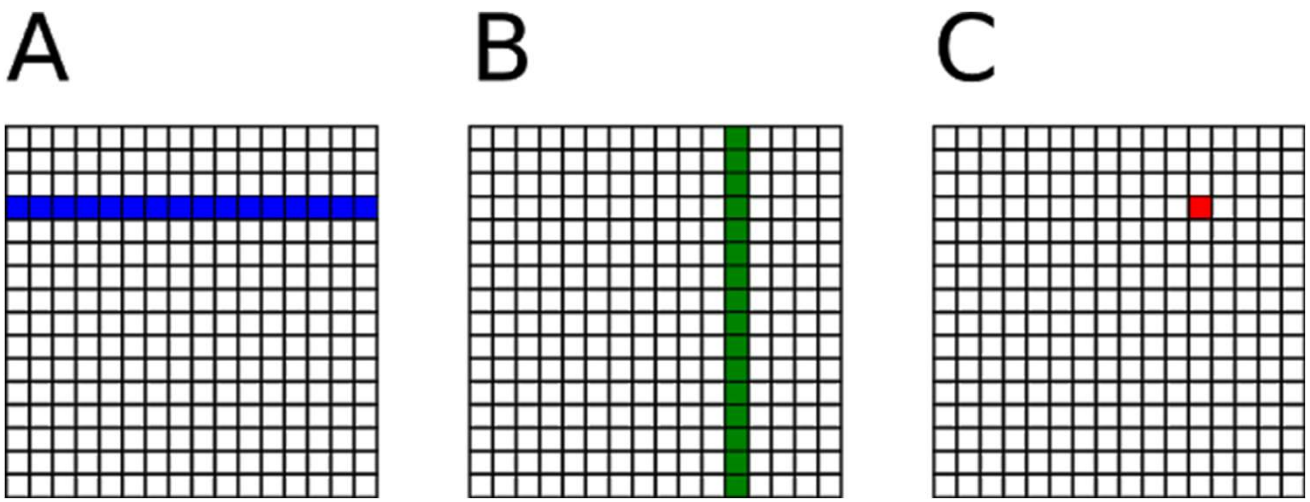
Blocked vs. Conventional

4096x4096 Array

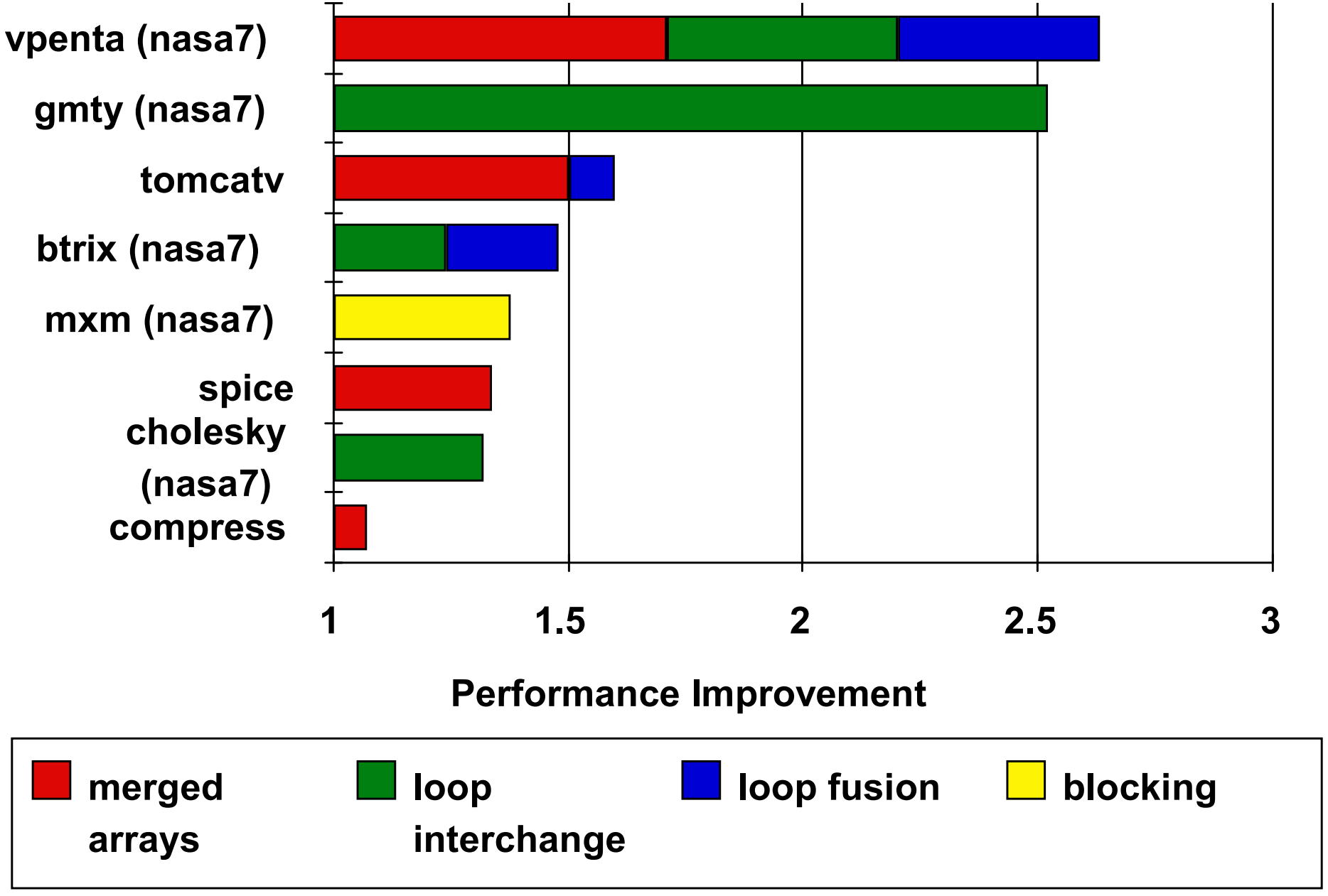


Block size: 32x32

vs.



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



Summary: Miss Rate Reduction

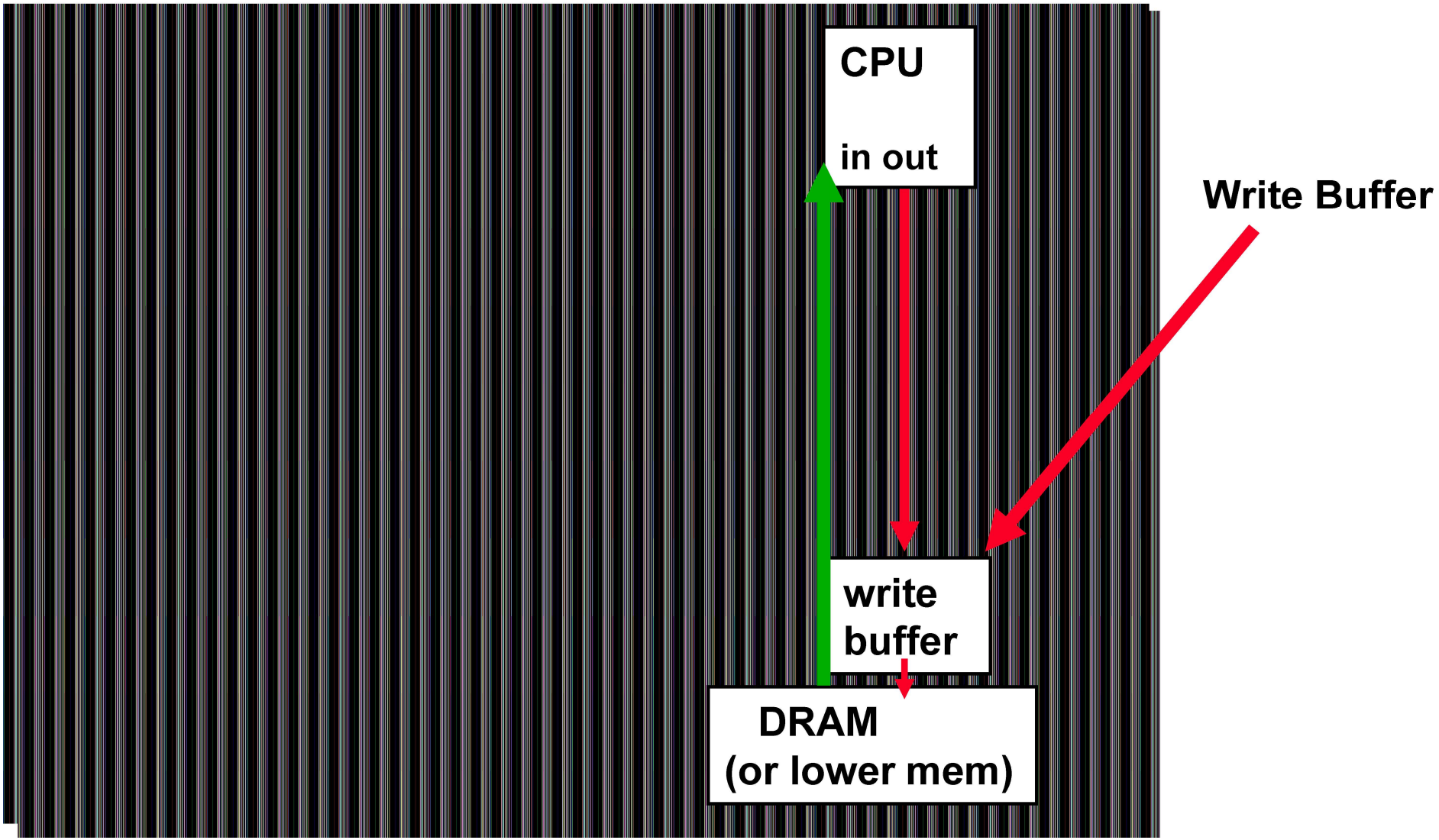
$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ \ accesses}{Instruction} \times \textbf{Miss rate} \times Miss\ \ penalty \right) \times Clock\ \ cycle\ \ time$$

- 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.

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 causes 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. Generally useful only in large blocks,
- In Spatial locality we want the words in order, so not clear if benefit critical word first. It improves performance if we want none sequential access of words in a block.

block

3.Reducing Miss Penalty: Multi-level Caches

- **Add a second level cache:**
  - **Often primary cache is on the same chip as the processor**
  - **Use SRAMs to add another cache above primary memory (DRAM)**
  - **Miss penalty goes down if data is in 2nd level cache**

- **Using multilevel caches:**
  - **Try and optimize the hit time on the 1st level cache**
  - **Try and optimize the miss rate on the 2nd level cache**

Multilevel Caches

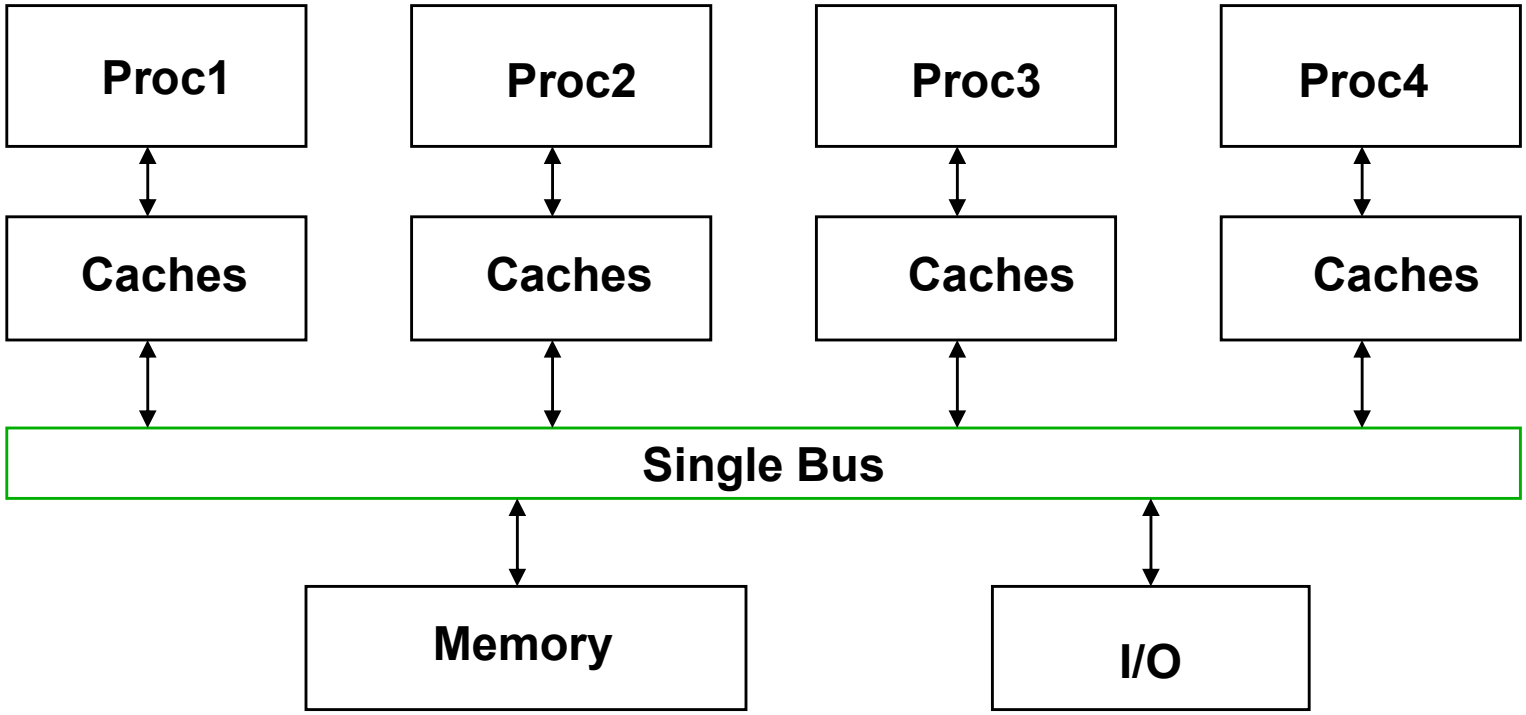
- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache

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
	Compiler Reduce Misses	+			0
miss penalty	Priority to Read Misses		+		1
	Early Restart & Critical Word 1st		+		3
	Second Level Caches		+		2

Cache Coherence

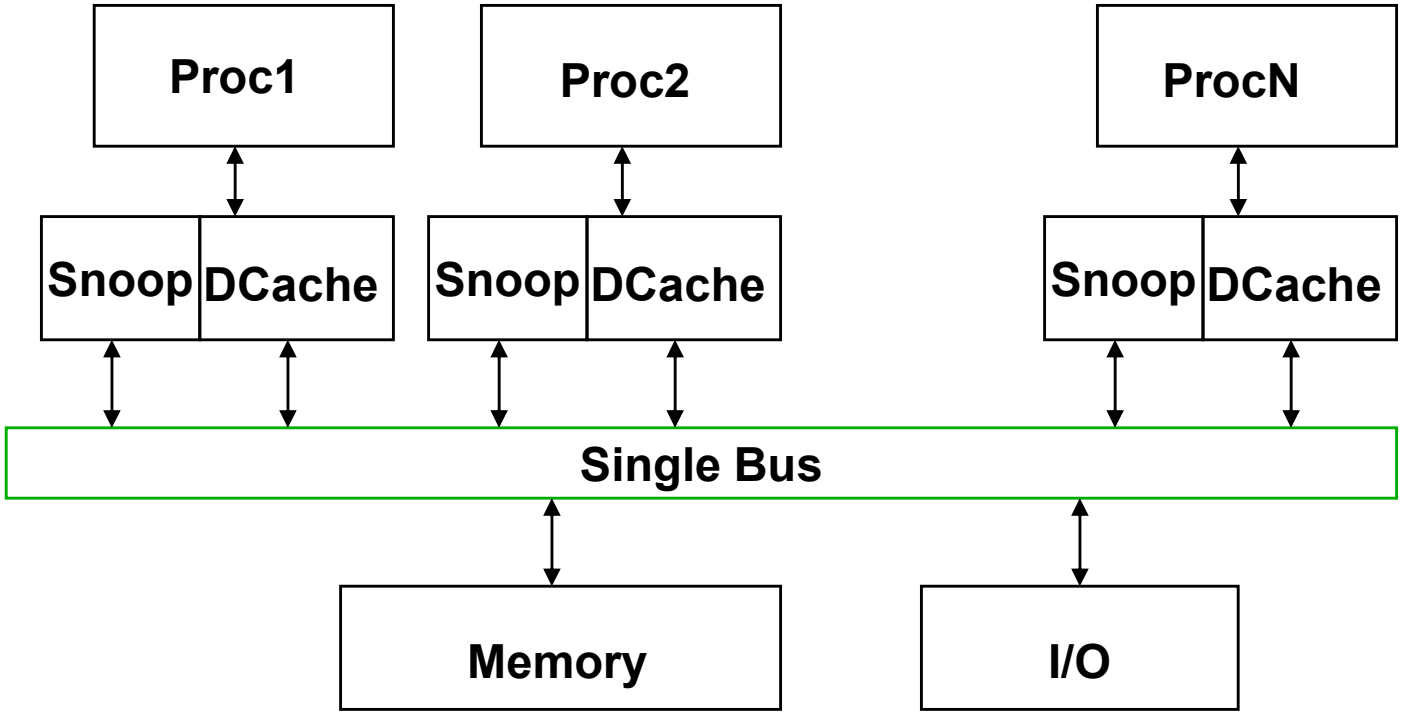
Single Bus (Shared Address UMA) Multi's



- Caches are used to reduce latency and to lower bus traffic
  - Write-back caches used to keep bus traffic at a minimum
- Must provide hardware to ensure that caches and memory are consistent (cache coherency)
- Must provide a hardware mechanism to support process synchronization

Multiprocessor Cache Coherency

- **Cache coherency protocols**
  - **Directory-based" The state of a block of memory is kept in one location (centralized) (centralized)**
  - **Bus snooping – cache controllers monitor shared bus traffic with duplicate address tag hardware (so they don't interfere with processor's access to the cache)**



Bus Snooping Protocols

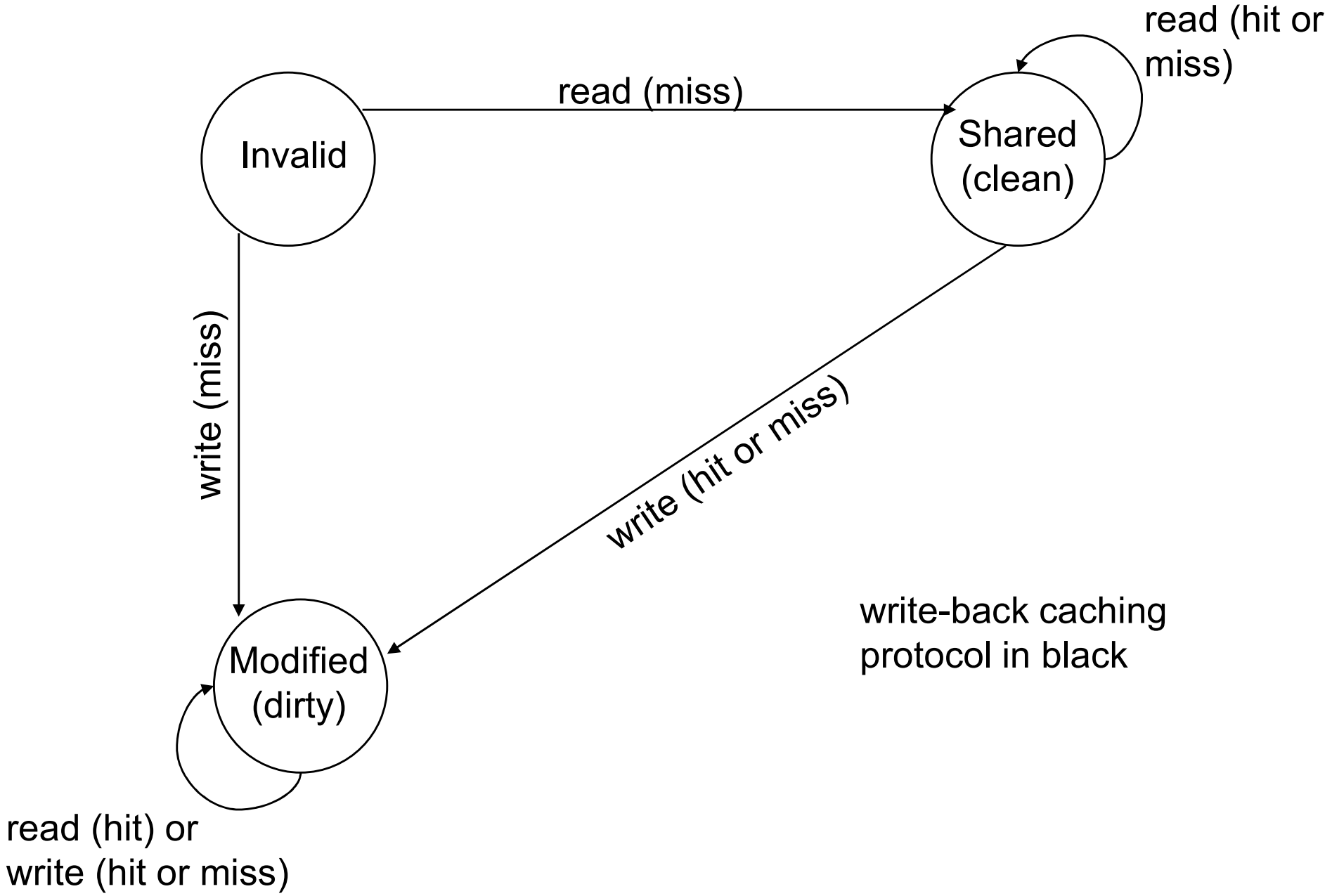
- Multiple copies are not a problem when reading
- Processor must have exclusive access to write a word
  - if two processors try to write to the same shared data word in the same clock cycle?
  - The bus arbiter decides which processor gets the bus first.
  - Then the second processor will get exclusive access.
  - Thus, bus arbitration forces sequential behavior.
    - This sequential consistency is the most conservative of the memory consistency models.
- All other processors sharing that data must be informed of writes

Handling Writes

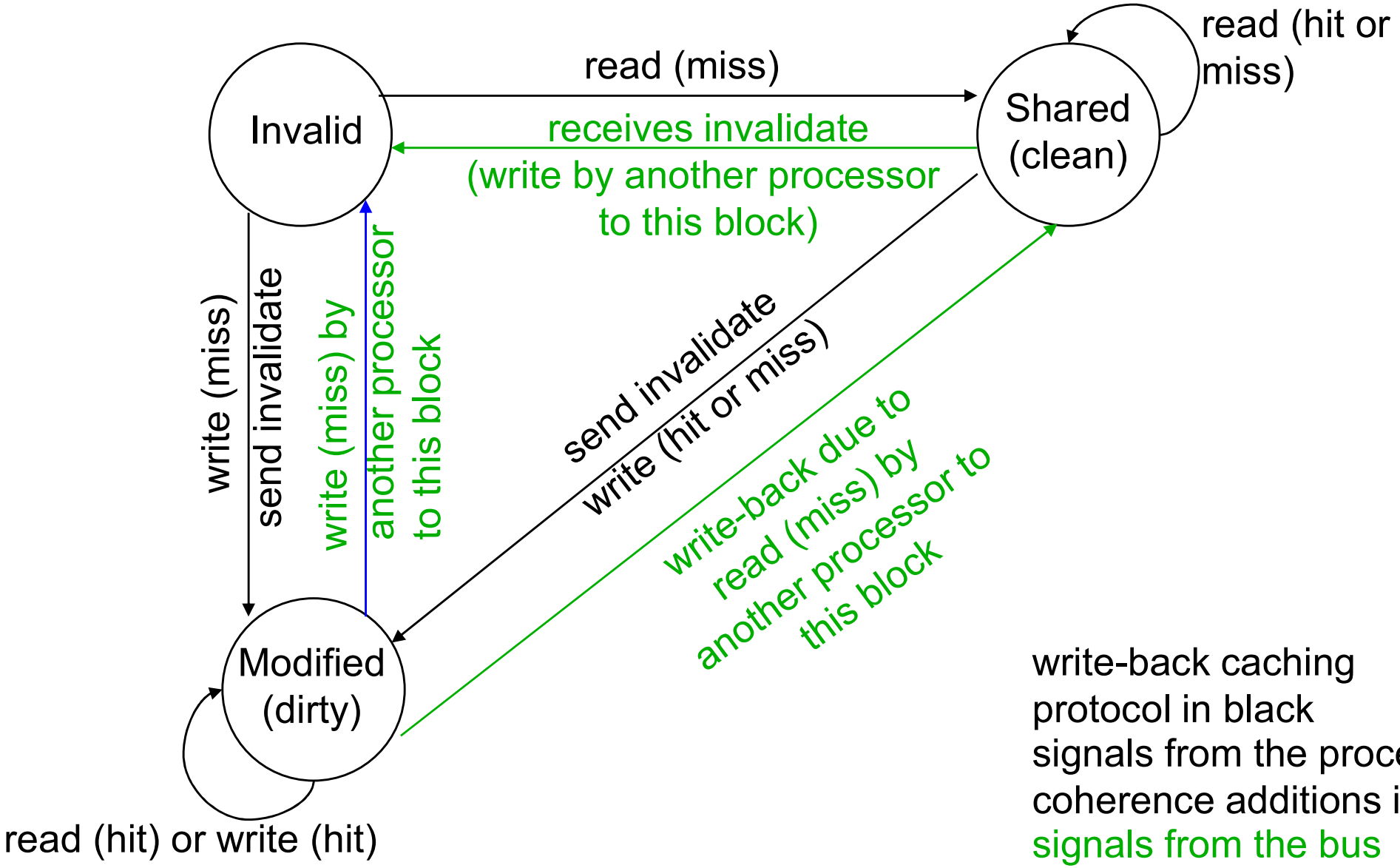
Ensuring that all other processors sharing data are informed of writes can be handled two ways:

1. **Write-update (write-broadcast) – writing processor broadcasts new data over the bus, all copies are updated**
   - **All writes go to the bus → higher bus traffic**
   - **Since new values appear in caches sooner, can reduce latency**
2. **Write-invalidate – writing processor issues invalidation signal on bus, cache snoops check to see if they have a copy of the data, if so they invalidate their cache block containing the word (this allows multiple readers but only one writer)**
   - **Uses the bus only on the first write → lower bus traffic, so better use of bus bandwidth**

A Write-Invalidate CC Protocol



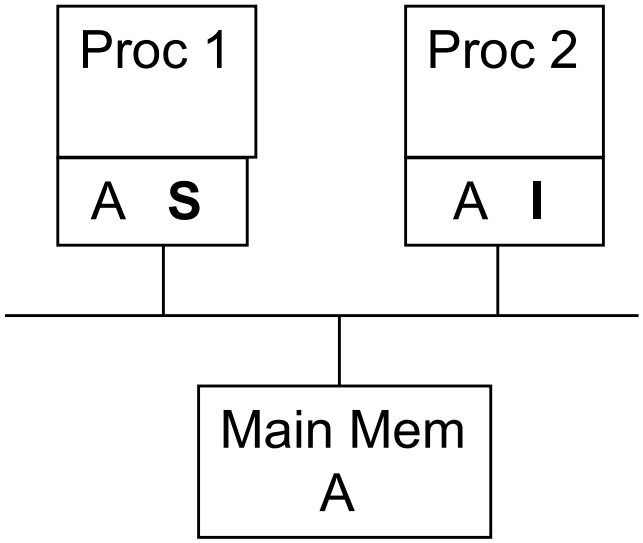
A Write-Invalidate CC Protocol

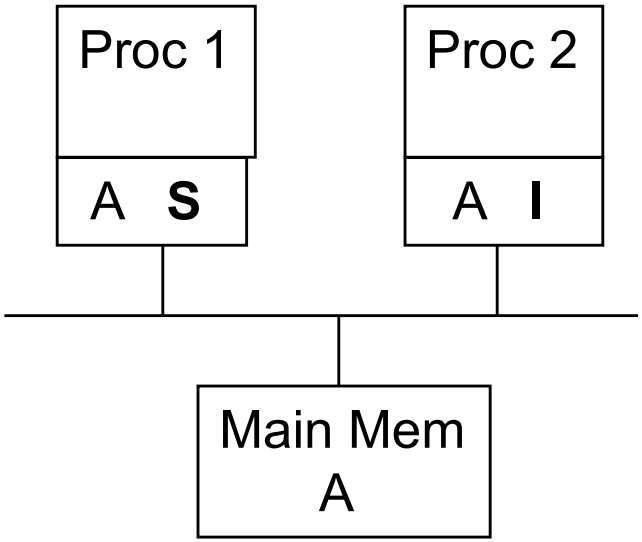


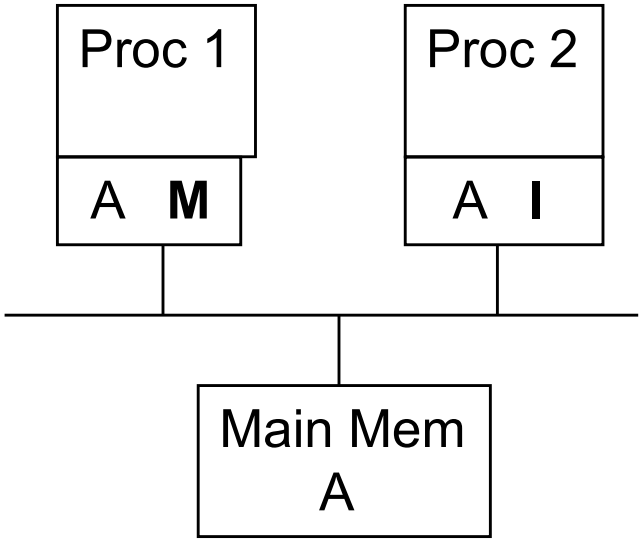
write-back caching protocol in black
signals from the processor
coherence additions in red
signals from the bus
coherence additions in blue

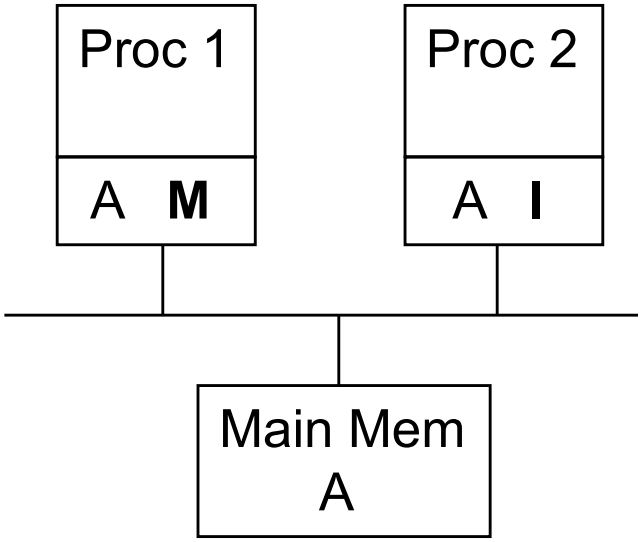
Write-Invalidate CC Examples

- **I = invalid (many), S = shared (many), M = modified (only one)**



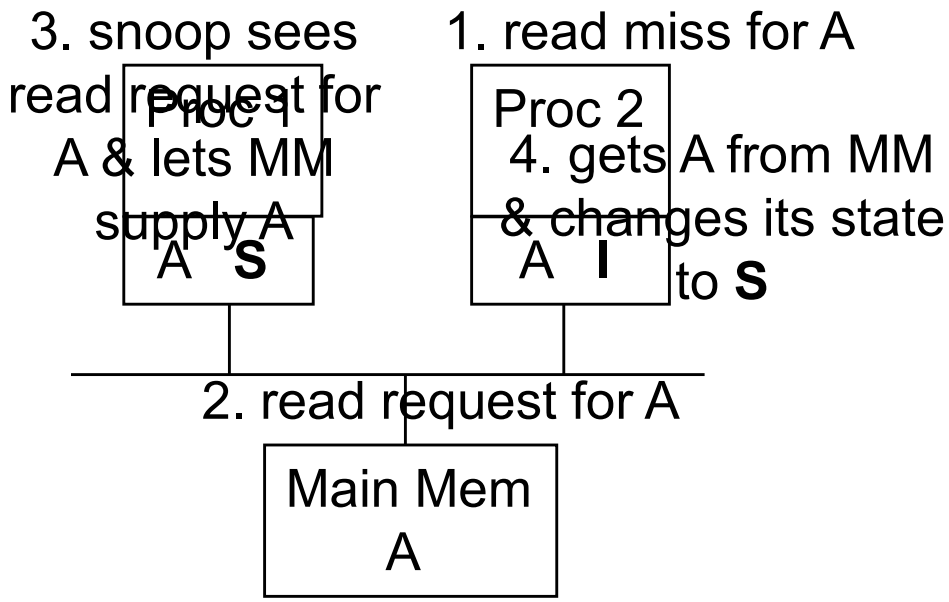


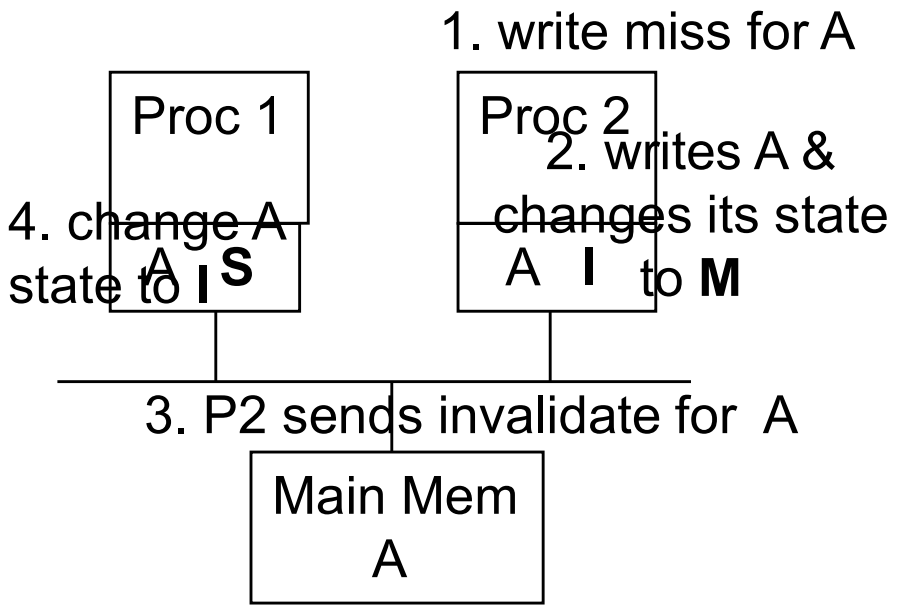


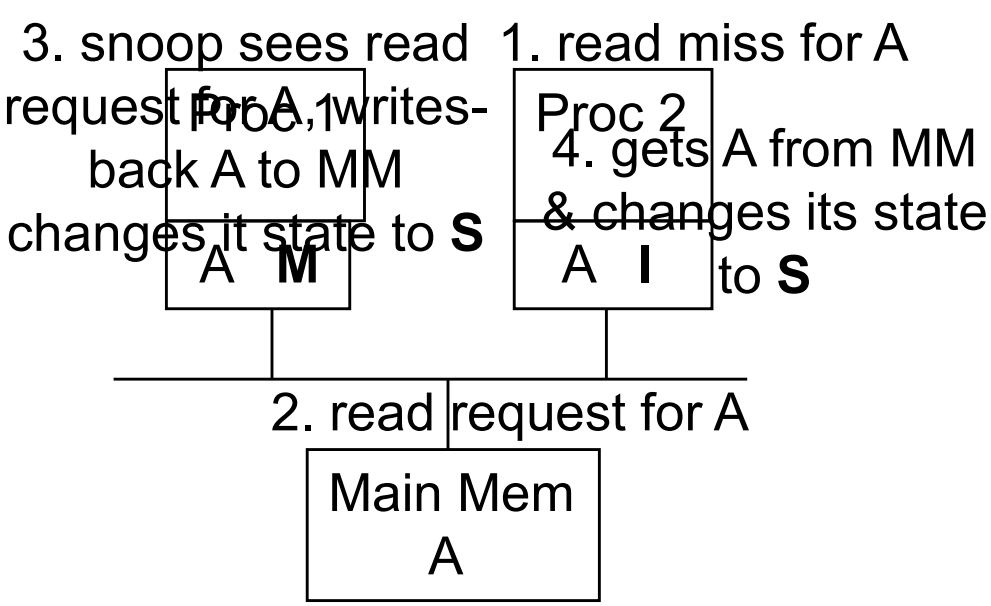


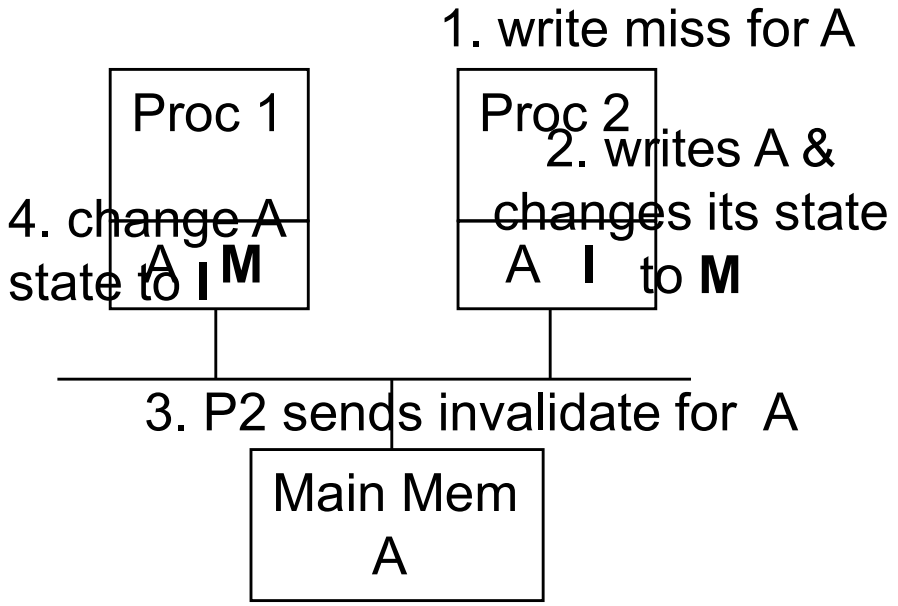
Write-Invalidate CC Examples

- **I = invalid (many), S = shared (many), M = modified (only one)**









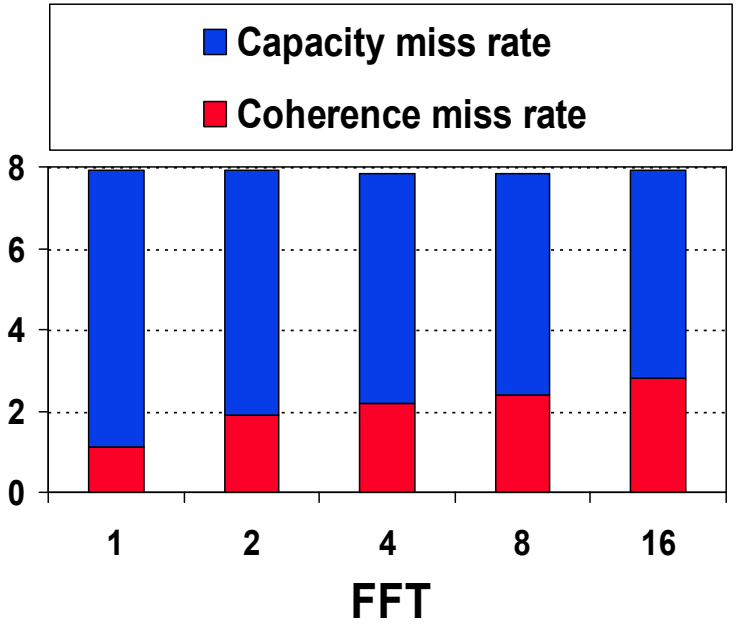
Note: Assume cache block is one word only and we are using write-allocate cache policy.

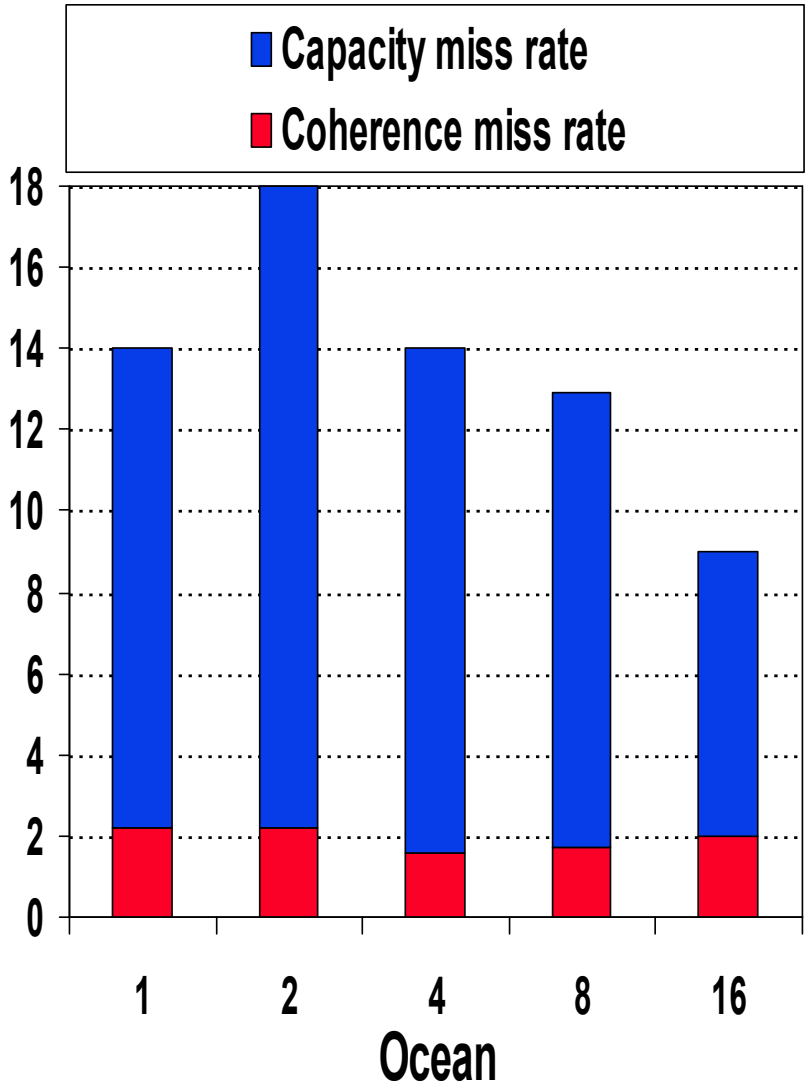
SMP Data Miss Rates

- **Shared data has lower spatial and temporal locality**
  - **Share data misses often dominate cache behavior even though they may only be 10% to 40% of the data accesses**

64KB 2-way set associative data cache with 32B blocks

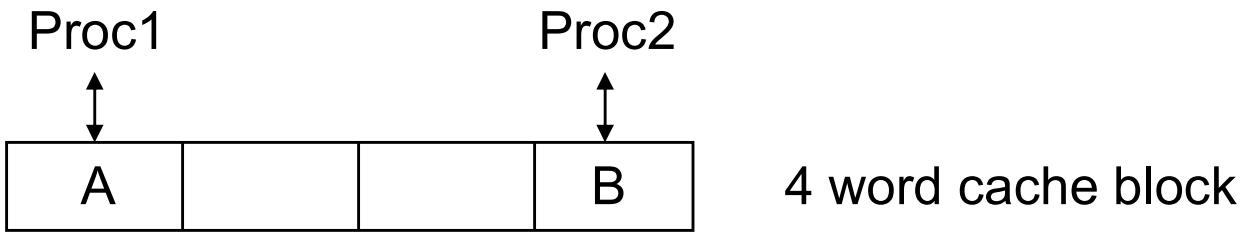
Hennessy & Patterson, *Computer Architecture: A Quantitative Approach*





Block Size Effects

- **Writes to one word in a multi-word block mean**
  - **either the full block is invalidated (write-invalidate)**
  - **or the full block is exchanged between processors (write-update)**
    - **Alternatively, could broadcast *only* the written word**
- **Multi-word blocks can also result in false sharing: when two processors are writing to two different variables in the same cache block**
  - **With write-invalidate false sharing increases cache miss rates**



- Compilers can help reduce false sharing by allocating highly correlated data to the same cache block

Other Coherence Protocols

- There are many variations on cache coherence protocols

- Another write-invalidate protocol used in the Pentium 4 (and many other micro's) is MESI with four states:
  - Modified – (same) only modified cache copy is up-to-date; memory copy and all other cache copies are out-of-date
  - Exclusive – only one copy of the shared data is allowed to be cached; memory has an up-to-date copy
    - Since there is only one copy of the block, write hits don't need to send invalidate signal
  - Shared – multiple copies of the shared data may be cached (i.e., data permitted to be cached with more than one processor); memory has an up-to-date copy
  - Invalid – same