

Memory Hierarchy Design

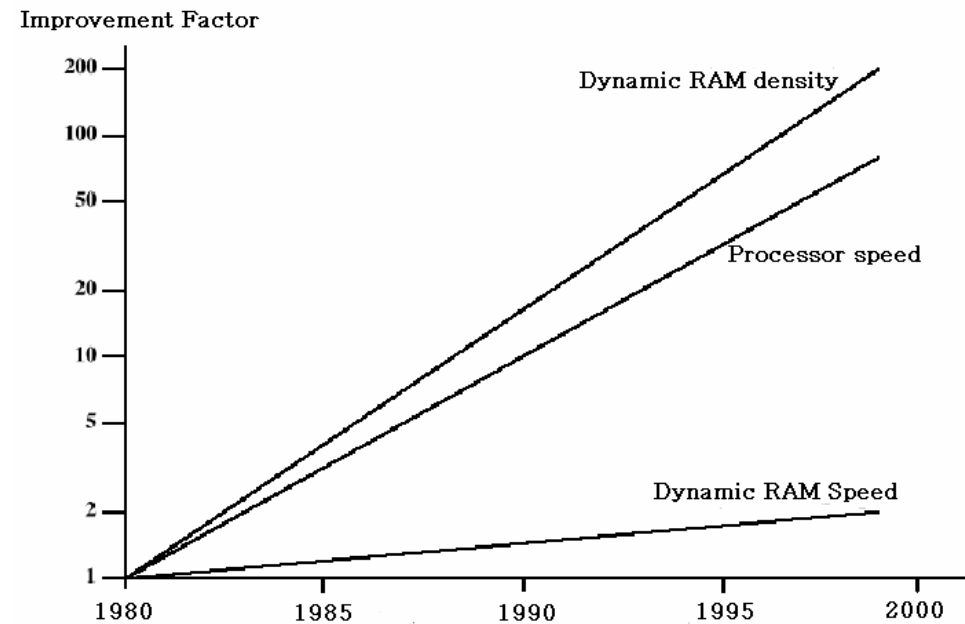
Chapter 5

Outline

- Review of the ABCs of Caches (5.2)
- Cache Performance
- Reducing Cache Miss Penalty

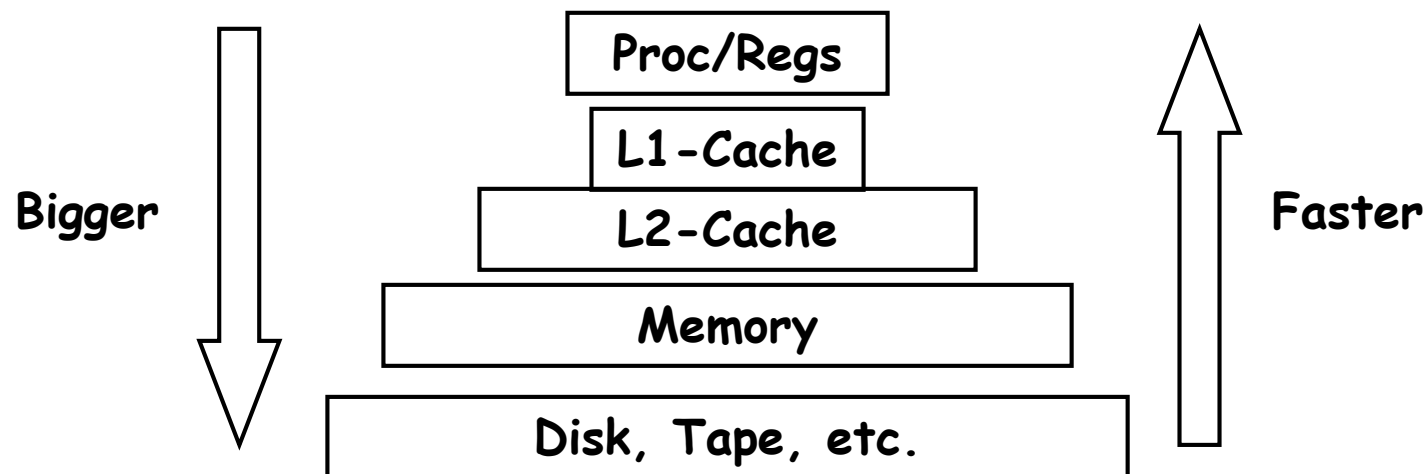
Overview

- Problem
 - CPU vs Memory performance imbalance
- Solution
 - Driven by temporal and spatial locality
 - Memory hierarchies
 - Fast L1, L2, L3 caches
 - Larger but slower memories
 - Even larger but even slower secondary storage
 - Keep most of the action in the higher levels



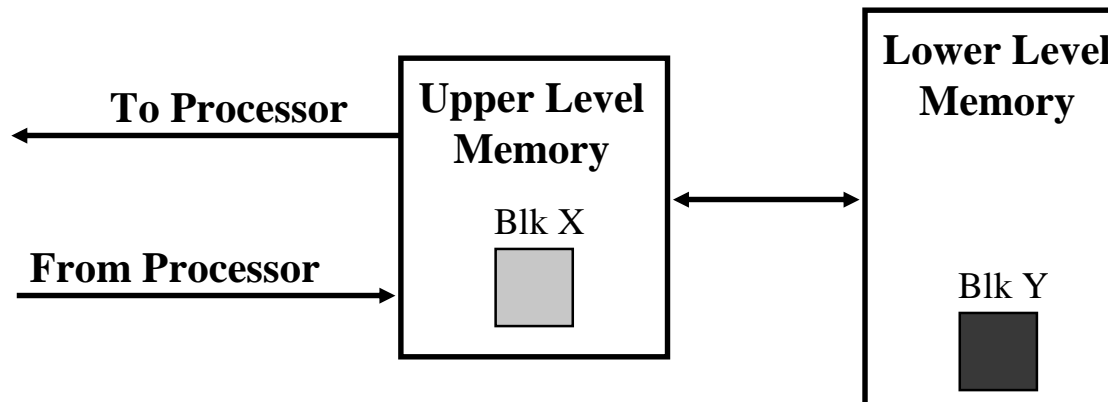
Review: What is a cache?

- Small, fast storage used to improve average access time to slow memory.
- Exploits spatial 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?



Review: Terminology

- Hit: data appears in some block in the upper level (example: Block X)
 - Hit Rate: the fraction of memory access found in the upper level
 - Hit Time: Time to access the upper level which consists of
RAM access time + Time to determine hit/miss
- Miss: data needs to be retrieve from a block in the lower level (Block Y)
 - Miss Rate = $1 - (\text{Hit Rate})$
 - Miss Penalty: Time to replace a block in the upper level +
Time to deliver the block the processor
- Hit Time \ll Miss Penalty (500 instructions on 21264!)

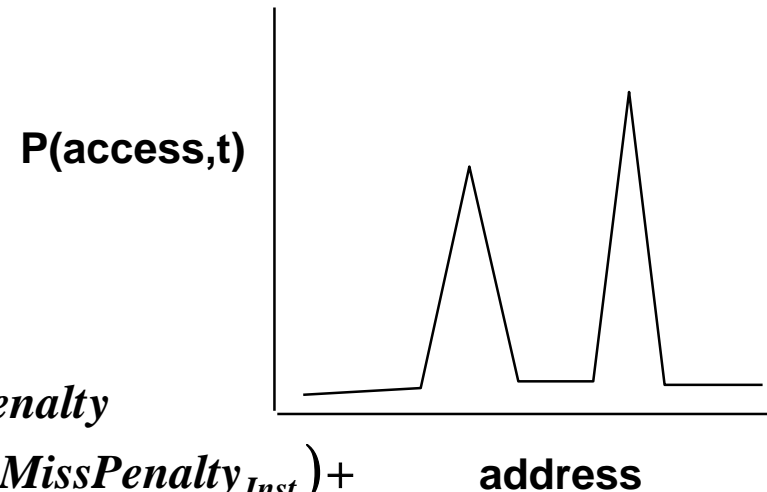


Why it works

- Exploit the statistical properties of programs
- Locality of reference
 - Temporal
 - Spatial

Average Memory Access Time

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



- Simple hardware structure that observes program behavior and reacts to improve future performance
- Is the cache visible in the ISA?

Locality of Reference

- Temporal and Spatial
- Sequential access to memory
- Unit-stride loop (cache lines = 256 bits)

```
for (i = 1; i < 100000; i++)  
    sum = sum + a[i];
```

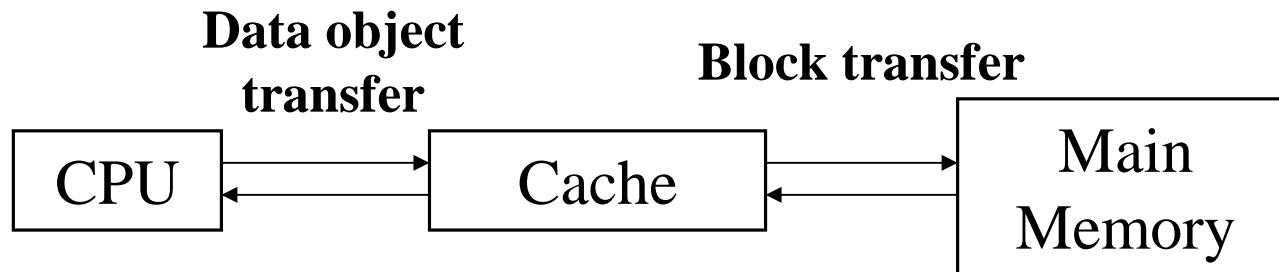
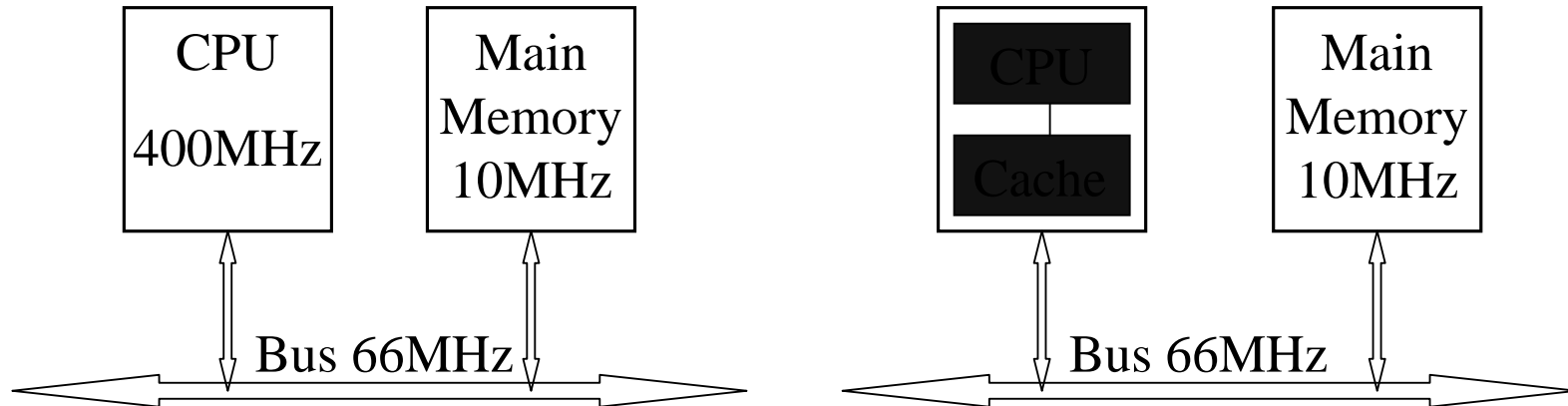
- Non-unit stride loop (cache lines = 256 bits)

```
for (i = 0; i <= 100000; i = i+8)  
    sum = sum + a[i];
```

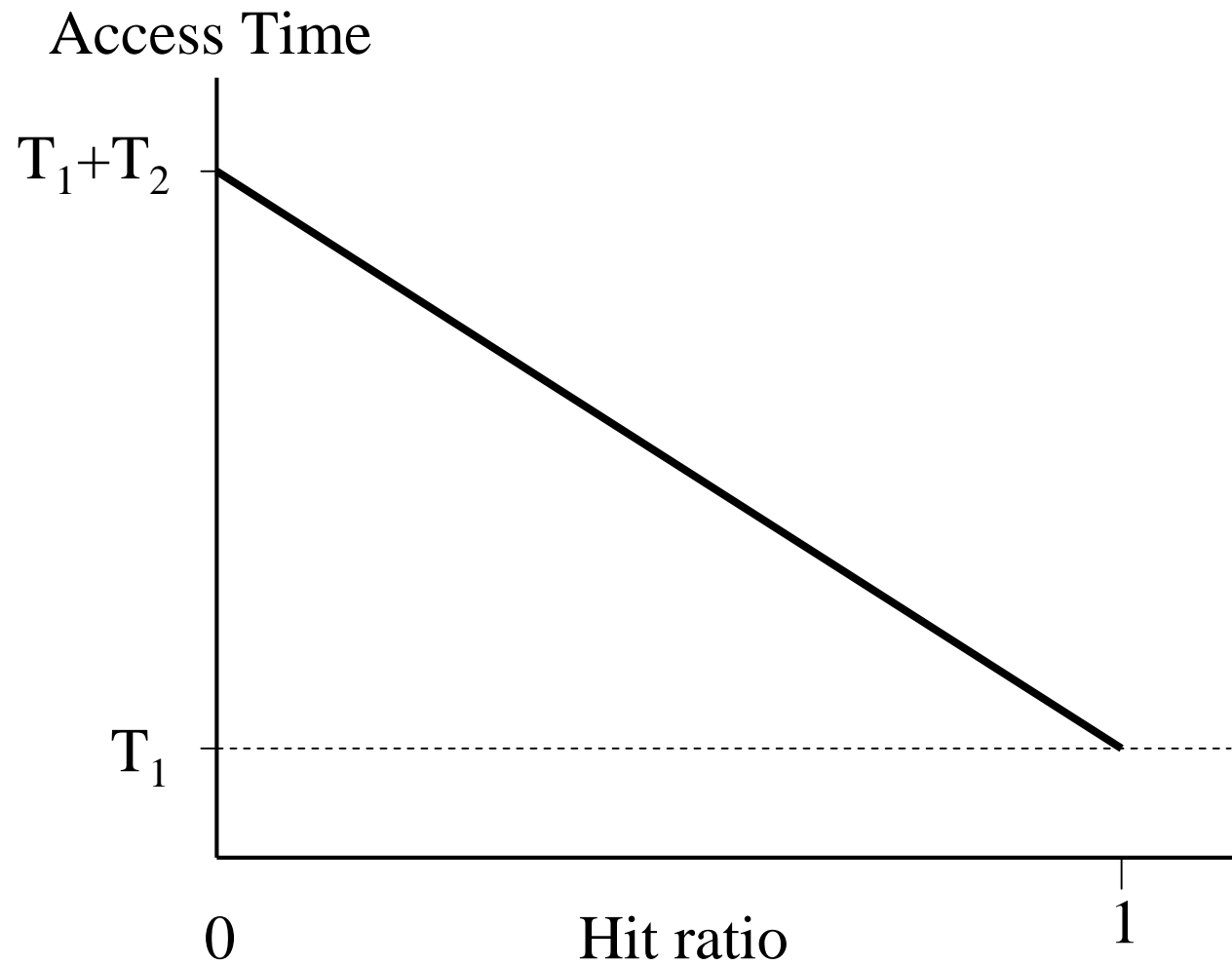
Locality

- Temporal locality
 - We are likely to need this word again in the near future.
- Spatial locality
 - There is a high probability that the other data in the block will be needed soon.

Cache Systems



Example: Two-level Hierarchy



Basic Cache Read Operation

- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU
- Cache includes tags to identify which block of main memory is in each cache slot

Elements of Cache Design

- Cache size
- Line (block) size
- Number of caches
- Block placement
- Block identification
- Replacement algorithm
- Write strategy

Cache Size

- Cache size \ll main memory size
- Small enough
 - Minimize cost
 - Speed up access (less gates to address the cache)
 - Keep cache on chip
- Large enough
 - Minimize average access time
- Optimum size depends on the workload
- Practical size?

Number of Caches

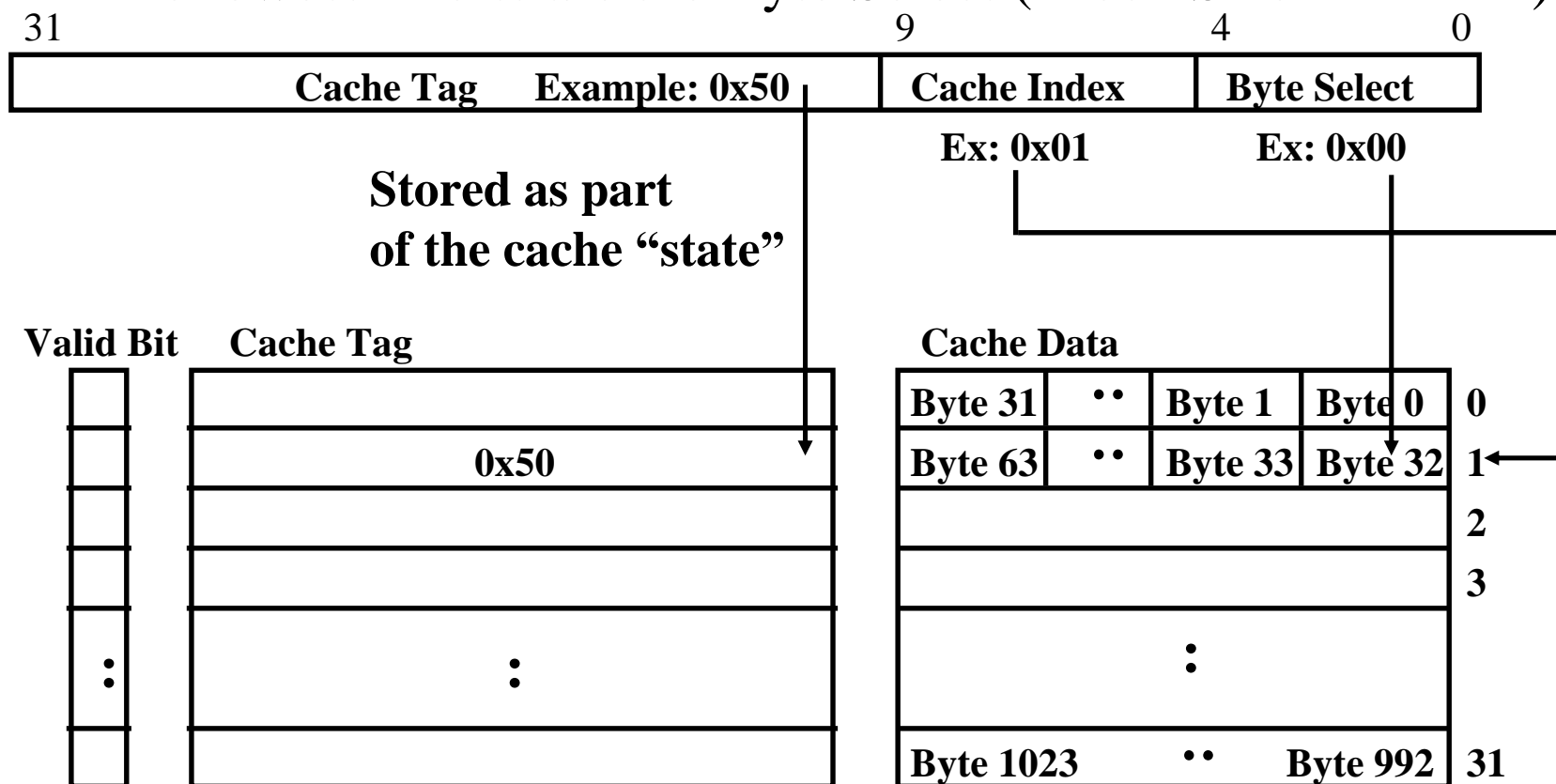
- Increased logic density => on-chip cache
 - Internal cache: level 1 (L1)
 - External cache: level 2 (L2)
- Unified cache
 - Balances the load between instruction and data fetches
 - Only one cache needs to be designed / implemented
- Split caches (data and instruction)
 - Pipelined, parallel architectures

Block Placement

- Q1: Where can a block be placed in the upper level?
 - Fully Associative,
 - Set Associative,
 - Direct Mapped

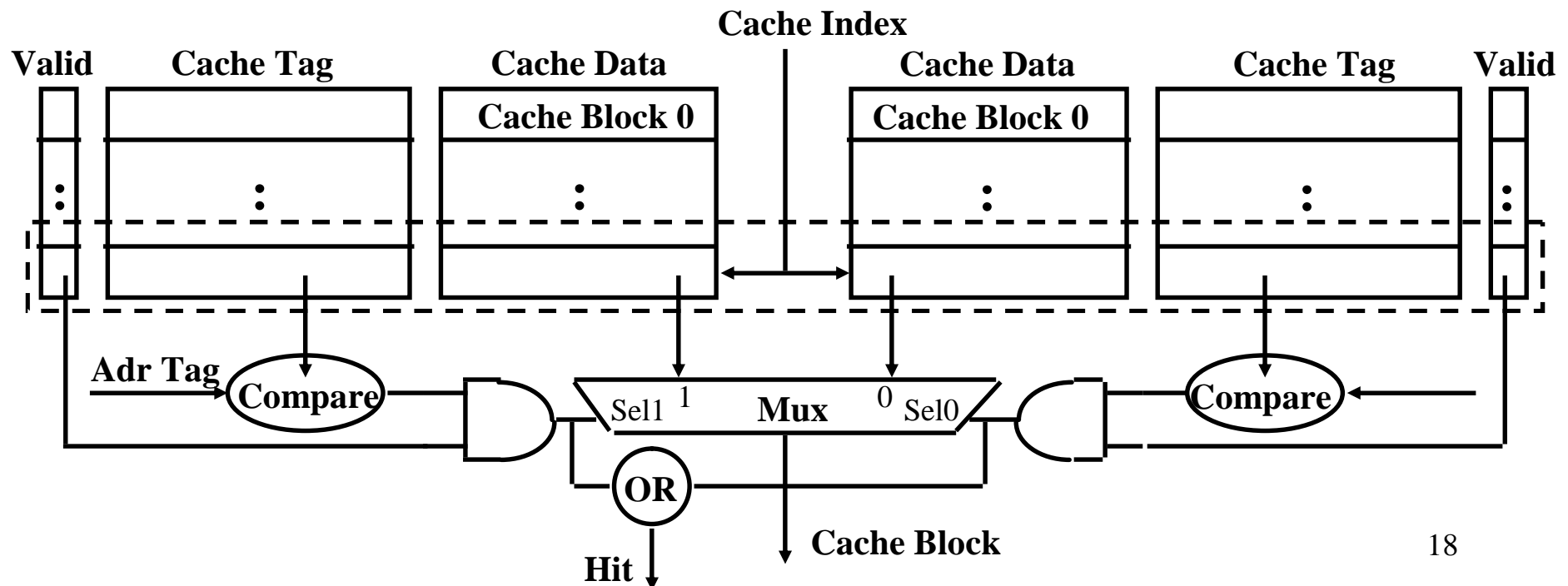
1 KB Direct Mapped Cache, 32B blocks

- 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 = $2^{**} M$)



Review: Set Associative Cache

- N-way set associative: N entries for each Cache Index
 - N direct mapped caches operates in parallel
 - How big is the tag?
- 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

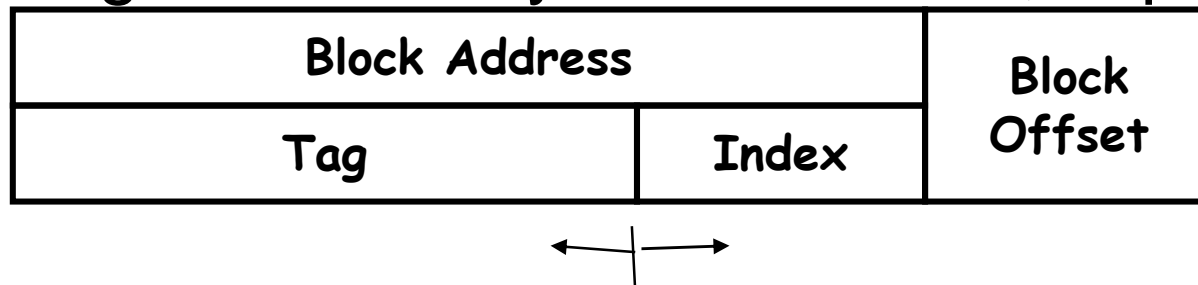


n-way Set Associative

- There are n blocks in a set.
- Direct mapped is simply one-way set associative.
- A fully associative cache with m blocks could be called “ m -way set associative”.
- Direct mapped can be thought of as having m sets, and fully associative as having one set.

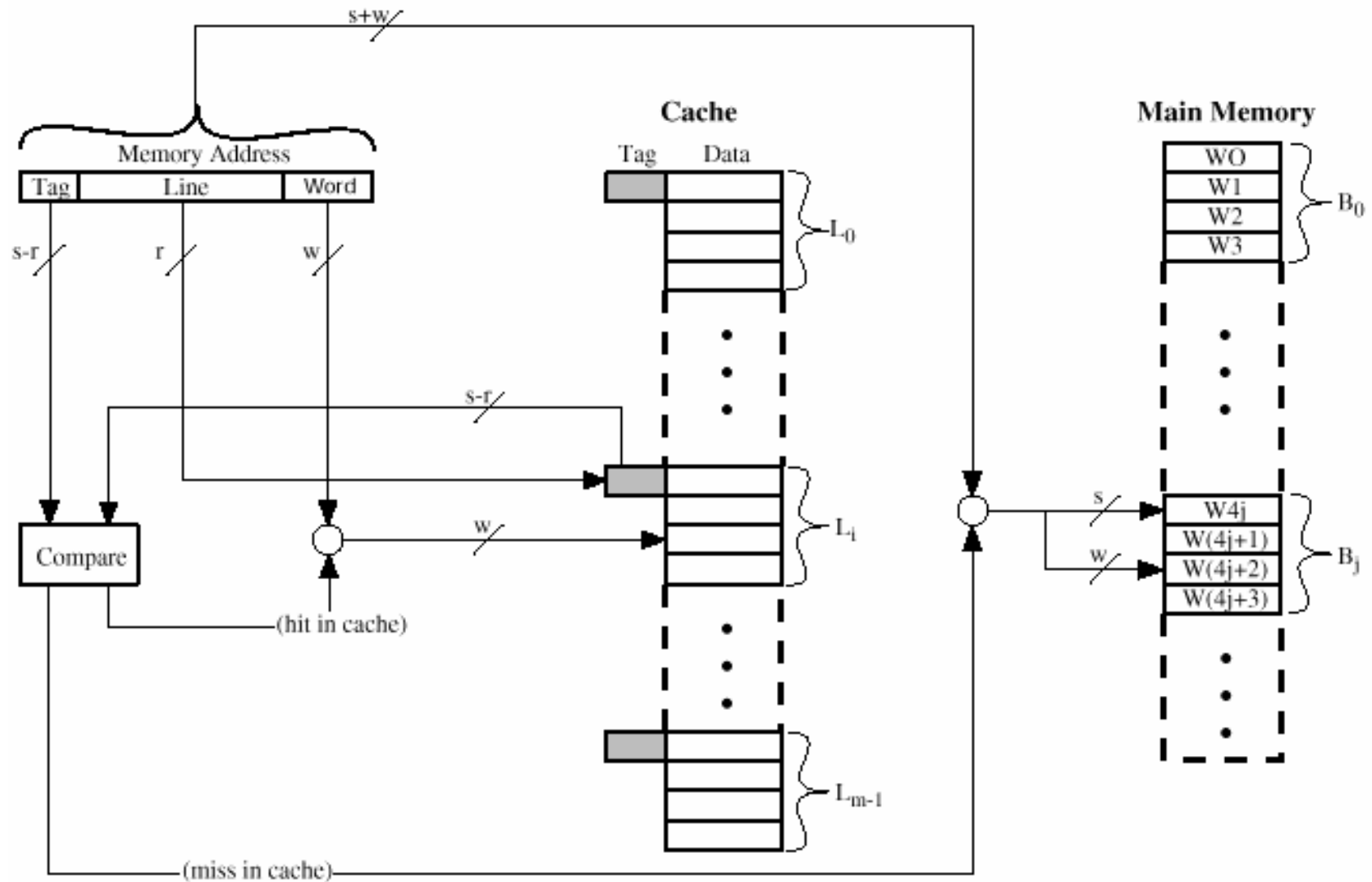
Q2: How is a block found if it is in the upper level?

- Index identifies set of possibilities
- Tag on each block
 - No need to check index or block offset
- Increasing associativity shrinks index, expands tag

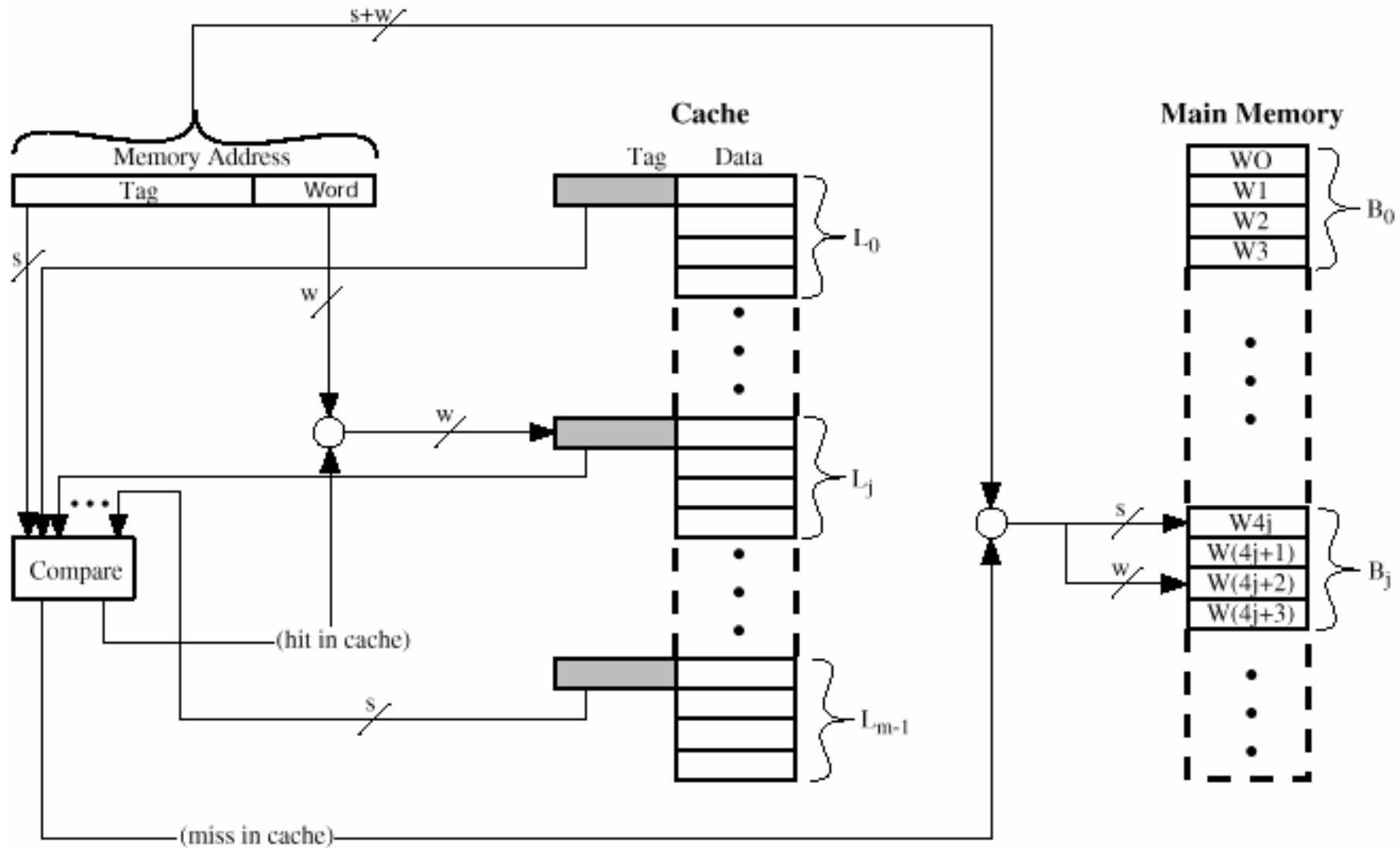


$$\text{Cache size} = \text{Associativity} * 2^{\text{index_size}} * 2^{\text{offset_size}}$$

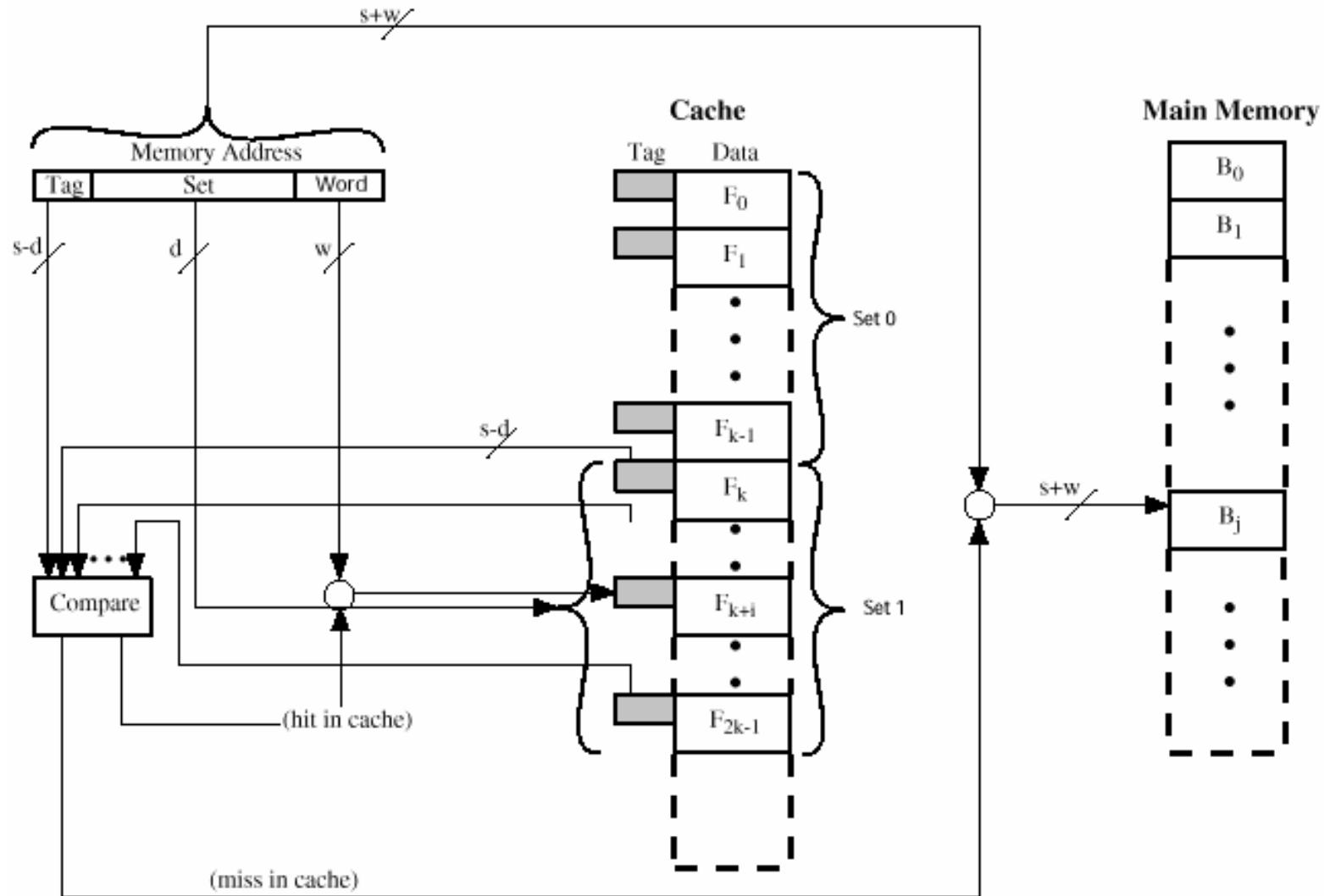
Direct Mapping



Associative Mapping



K-Way Set Associative Mapping



Replacement Algorithm

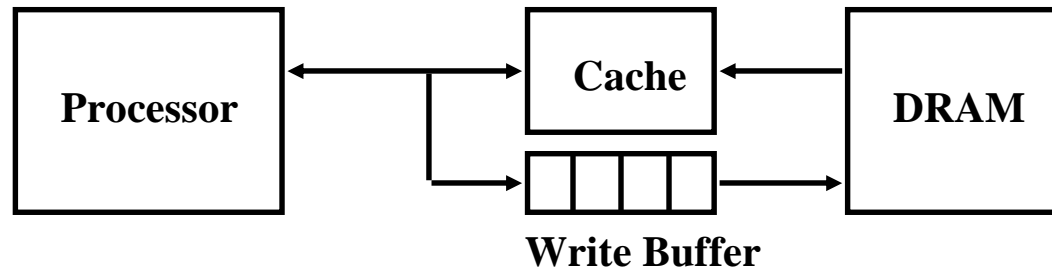
- Simple for direct-mapped: no choice
- Random
 - Simple to build in hardware
- LRU

Size	Associativity					
	Two-way		Four-way		Eight-way	
	LRU	Random	LRU	Random	LRU	Random
16KB	5.18%	5.69%	4.67%	5.29%	4.39%	4.96%
64KB	1.88%	2.01%	1.54%	1.66%	1.39%	1.53%
256KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

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?
- Pros and Cons of each?
 - WT: read misses cannot result in writes
 - WB: no repeated writes to same location
- WT always combined with write buffers so that don't wait for lower level memory
- What about on a miss?
 - Write_no_allocate vs write_allocate

Write Buffer for Write Through

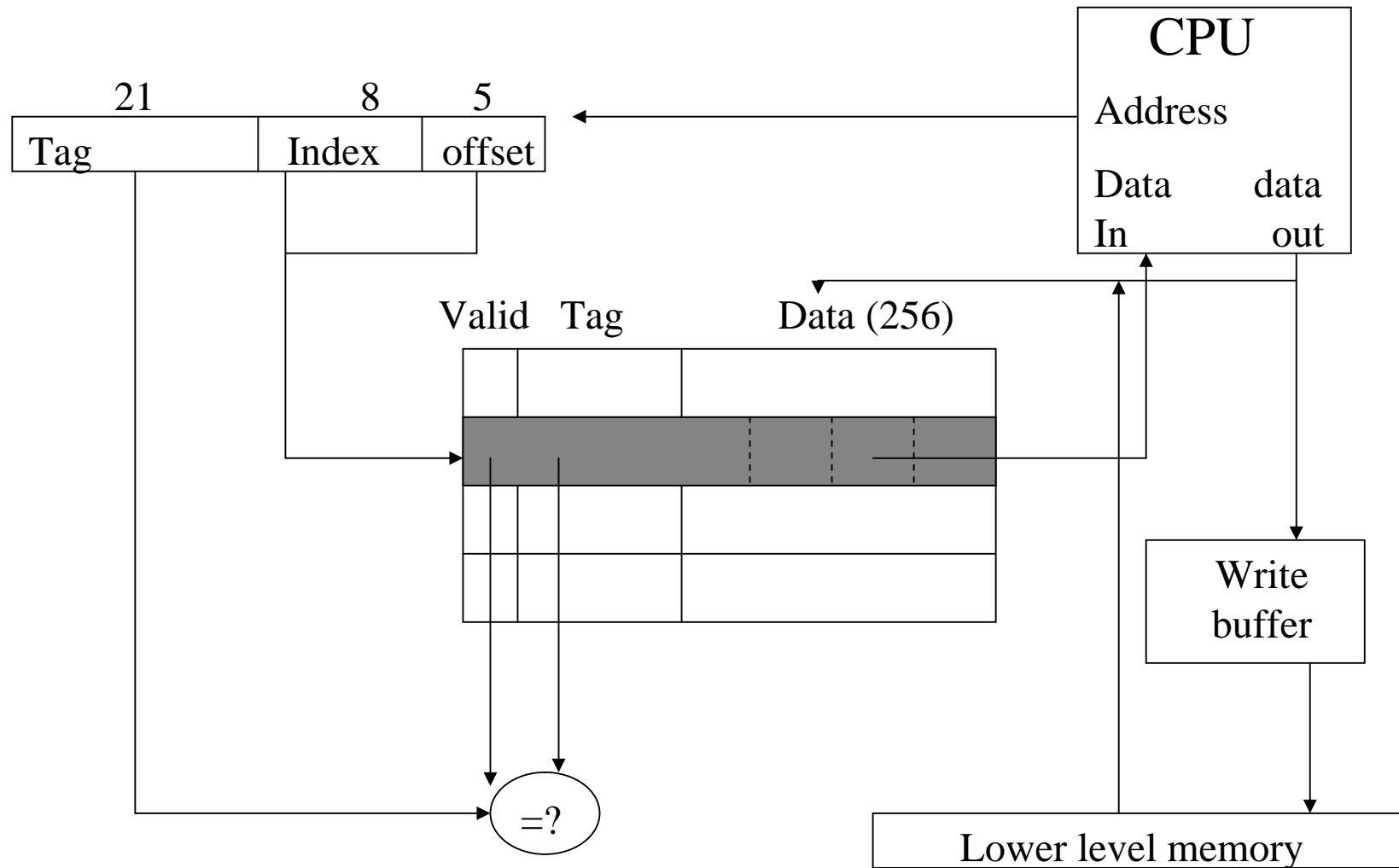


- A Write Buffer is needed between the Cache and Memory
 - Processor: writes data into the cache and the write buffer
 - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO:
 - Typical number of entries: 4
 - Works fine if: Store frequency (w.r.t. time) $\ll 1 / \text{DRAM write cycle}$

Write Policy

- Write is more complex than read
 - Write and tag comparison can not proceed simultaneously
 - Only a portion of the line has to be updated
- Write policies
 - Write through – write to the cache and memory
 - Write back – write only to the cache (dirty bit)
- Write miss:
 - Write allocate – load block on a write miss
 - No-write allocate – update directly in memory

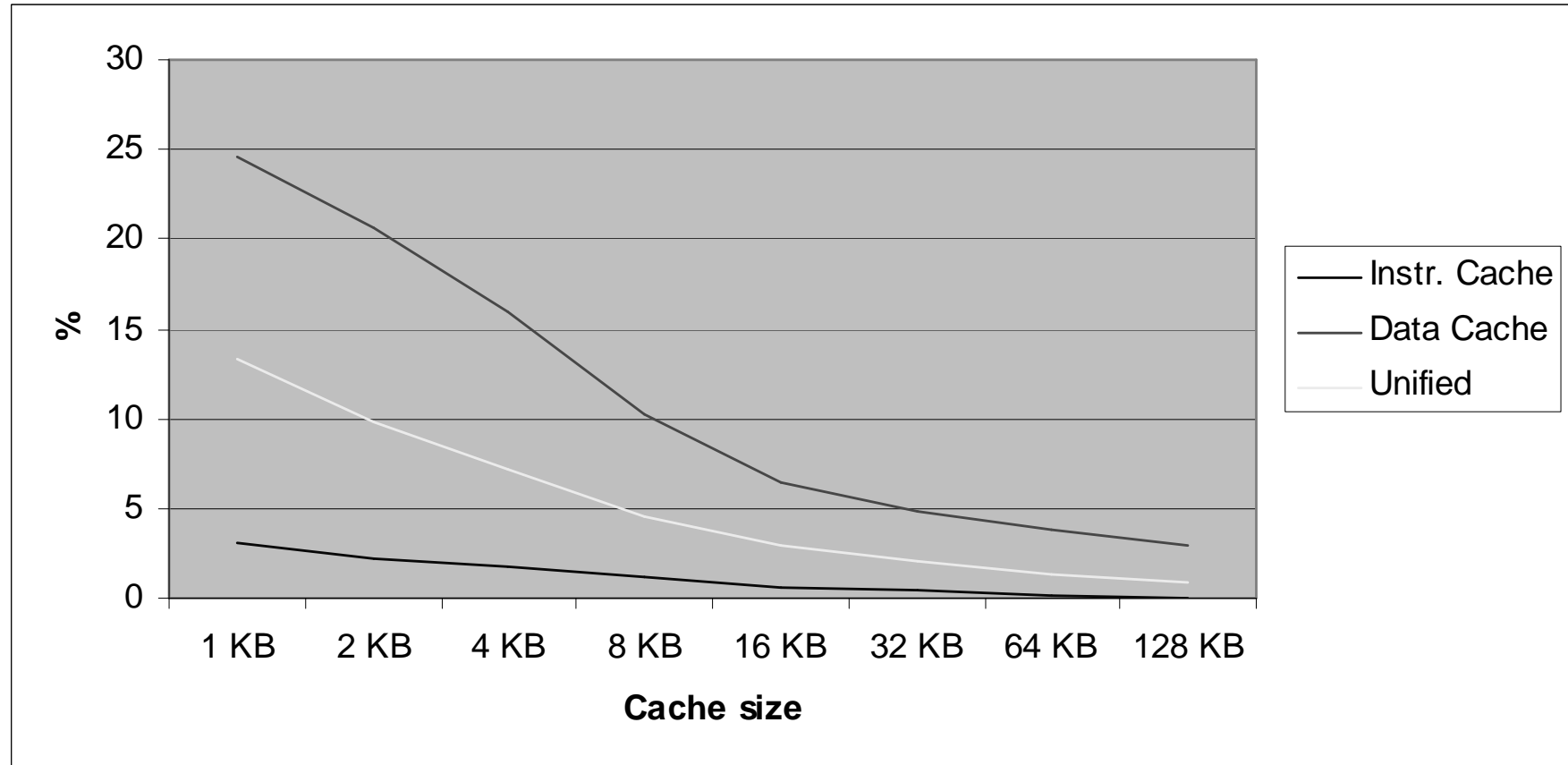
Alpha AXP 21064 Cache



Outline

- Review of the ABCs of Caches
- Cache Performance (5.3)
- Reducing Cache Miss Penalty

DECstation 5000 Miss Rates



Direct-mapped cache with 32-byte blocks

Percentage of instruction references is 75%

Cache Performance Measures

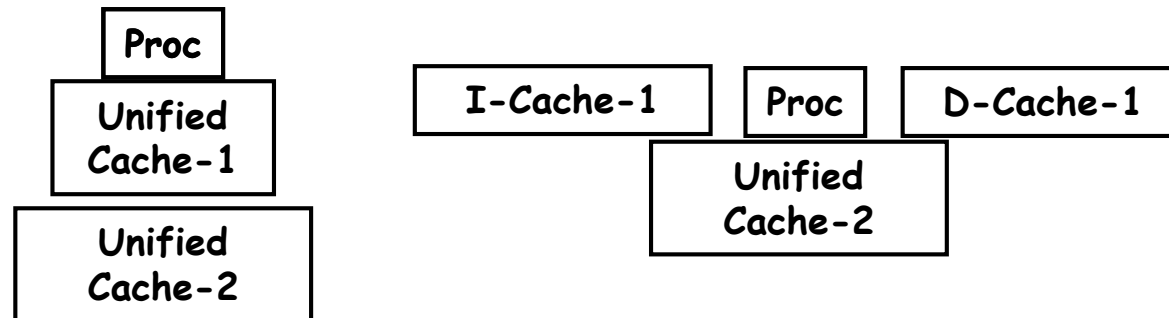
- *Hit rate*: fraction found in that level
 - So high that usually talk about *Miss rate*
 - Miss rate fallacy: as MIPS to CPU performance,
- Average memory-access time
 - = Hit time + Miss rate x Miss penalty (ns)
- *Miss penalty*: time to replace a block from lower level, including time to replace in CPU
 - *access time* to lower level = f(latency to lower level)
 - *transfer time*: time to transfer block =f(bandwidth)

Cache Performance Improvements

- Average memory-access time
= Hit time + Miss rate x Miss penalty
- Cache optimizations
 - Reducing the miss rate
 - Reducing the miss penalty
 - Reducing the hit time

Example: Harvard Architecture

- Unified vs Separate I&D (Harvard)



- Statistics (given in H&P):
 - 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)

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

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

Cache Performance Equations

$$\text{CPU}_{\text{time}} = (\text{CPU execution cycles} + \text{Mem stall cycles}) * \text{Cycle time}$$

$$\text{Mem stall cycles} = \text{Mem accesses} * \text{Miss rate} * \text{Miss penalty}$$

$$\text{CPU}_{\text{time}} = \text{IC} * (\text{CPI}_{\text{execution}} + \text{Mem accesses per instr} * \text{Miss rate} * \text{Miss penalty}) * \text{Cycle time}$$

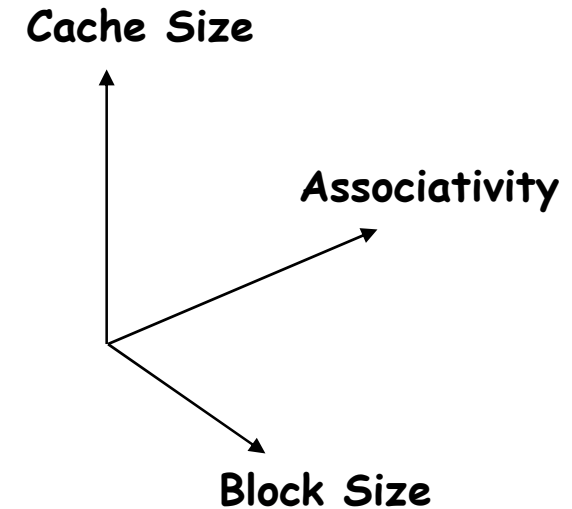
$$\text{Misses per instr} = \text{Mem accesses per instr} * \text{Miss rate}$$

$$\text{CPU}_{\text{time}} = \text{IC} * (\text{CPI}_{\text{execution}} + \text{Misses per instr} * \text{Miss penalty}) * \text{Cycle time}$$

The Cache Design Space

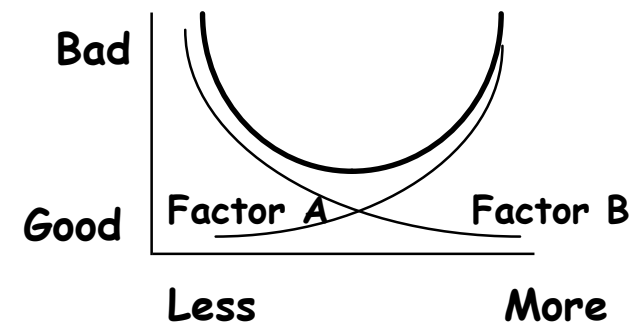
- Several interacting dimensions

- cache size
- block size
- associativity
- replacement policy
- write-through vs write-back



- The optimal choice is a compromise

- depends on access characteristics
 - workload
 - use (I-cache, D-cache, TLB)
- depends on technology / cost



- Simplicity often wins

Outline

- Review of the ABCs of Caches
- Cache Performance
- Reducing Cache Miss Penalty (5.4)
- Reducing Miss Rate (5.5)
- Reducing Miss Penalty and Miss Rate via Parallelism (5.6)

Reducing Miss Penalty

- Multi-level Caches
- Critical Word First and Early Restart
- Priority to Read Misses over Writes
- Merging Write Buffers
- Victim Caches
- Sub-block placement

Second-Level Caches

- L2 Equations

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

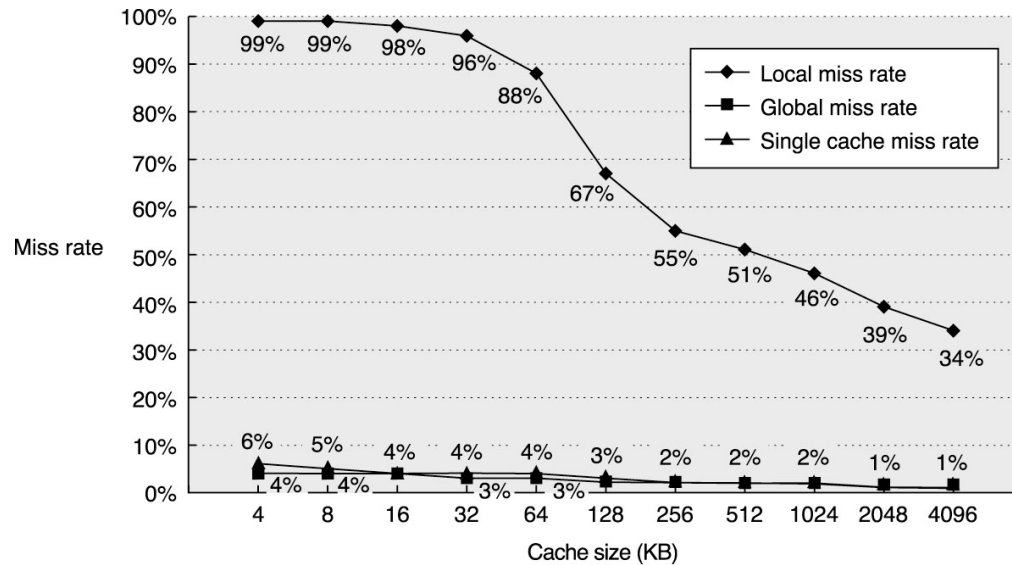
$$\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}$$

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

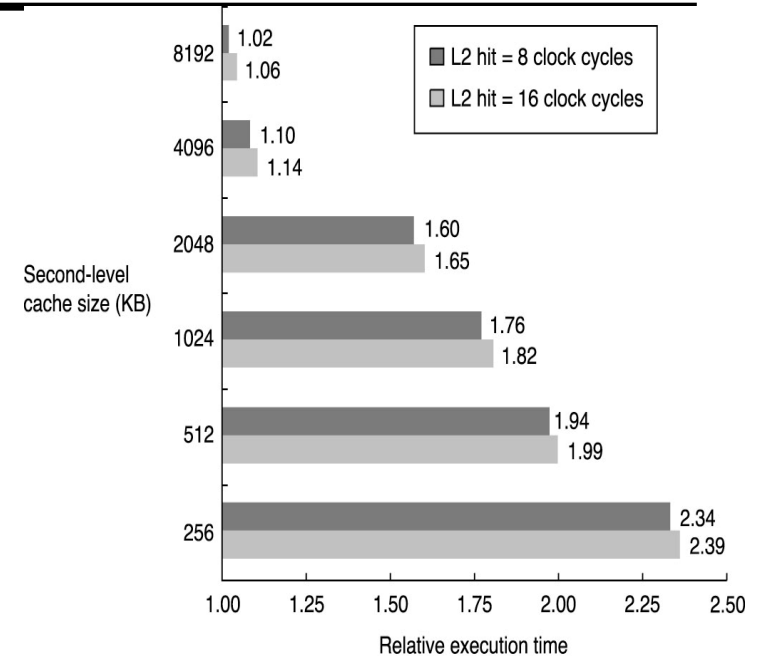
- Definitions:

- *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

Performance of Multi-Level Caches



© 2003 Elsevier Science (USA). All rights reserved.



© 2003 Elsevier Science (USA). All rights reserved.

- 32 KByte L1 cache;
- Global miss rate close to single level cache rate provided $L2 \gg L1$
- *local miss rate*
 - Do not use to measure impact
 - Use in equation!
- L2 not tied to clock cycle!
- Target miss reduction

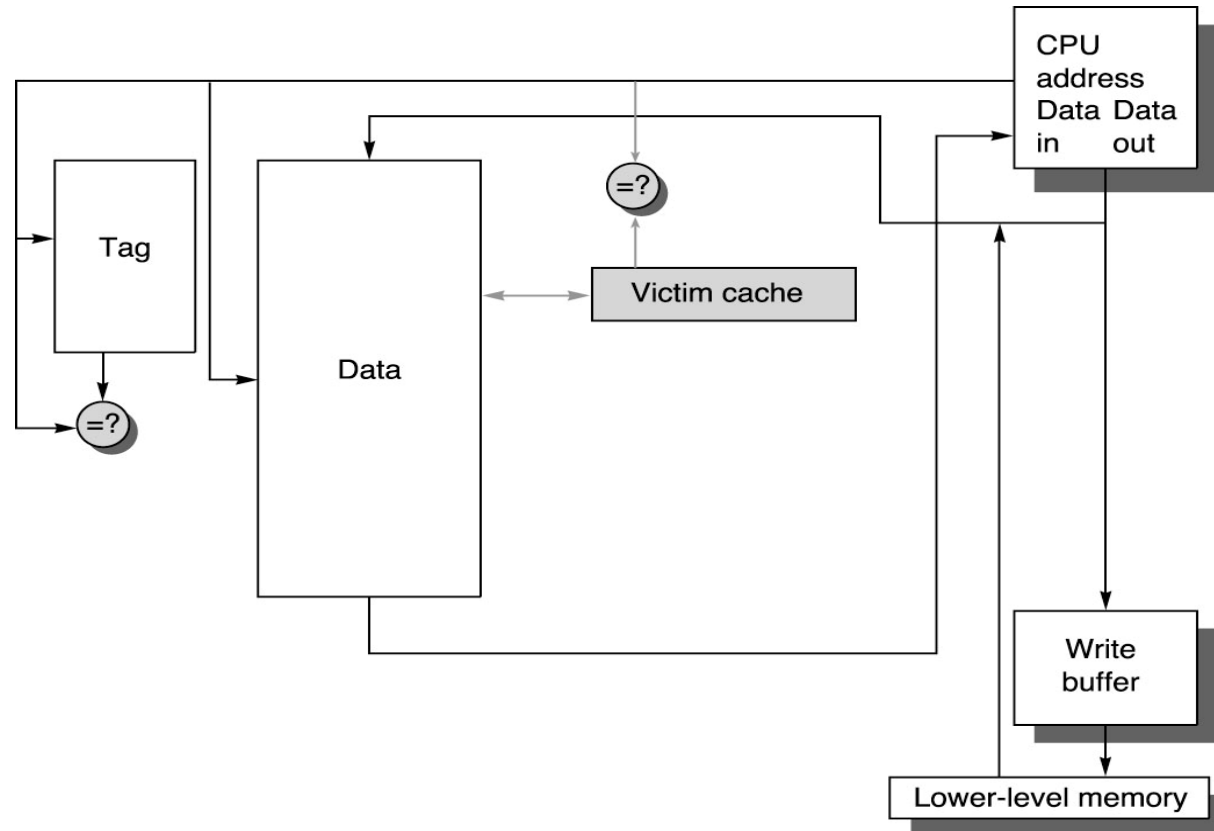
Early Restart and CWF

- Don't wait for full block to be loaded
 - *Early restart*—As soon as the requested word arrives, send it to the CPU and let the CPU continue execution
 - *Critical Word First*—Request the missed word first and send it to the CPU as soon as it arrives; then fill in the rest of the words in the block.
- Generally useful only in large blocks
- Extremely good spatial locality can reduce impact
 - Back to back reads on two halves of cache block does not save you much (see example in book)
 - Need to schedule instructions!

Giving Priority to Read Misses

- Write buffers complicate memory access
 - RAW hazard in main memory on cache misses
 - SW 512(R0), R3 (cache index 0)
 - LW R1, 1024(R0) (cache index 0)
 - LW R2, 512(R0) (cache index 0)
- Wait for write buffer to empty?
 - Might increase read miss penalty
- 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, then do the read
 - Optimized: copy dirty block to write buffer, then do the read
 - More optimization: write merging

Victim Caches



Write Merging

Write address

V V V V

100	1		0		0		0	
104	1		0		0		0	
108	1		0		0		0	
112	1		0		0		0	

Write address

V V V V

100	1		1		1		1	
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

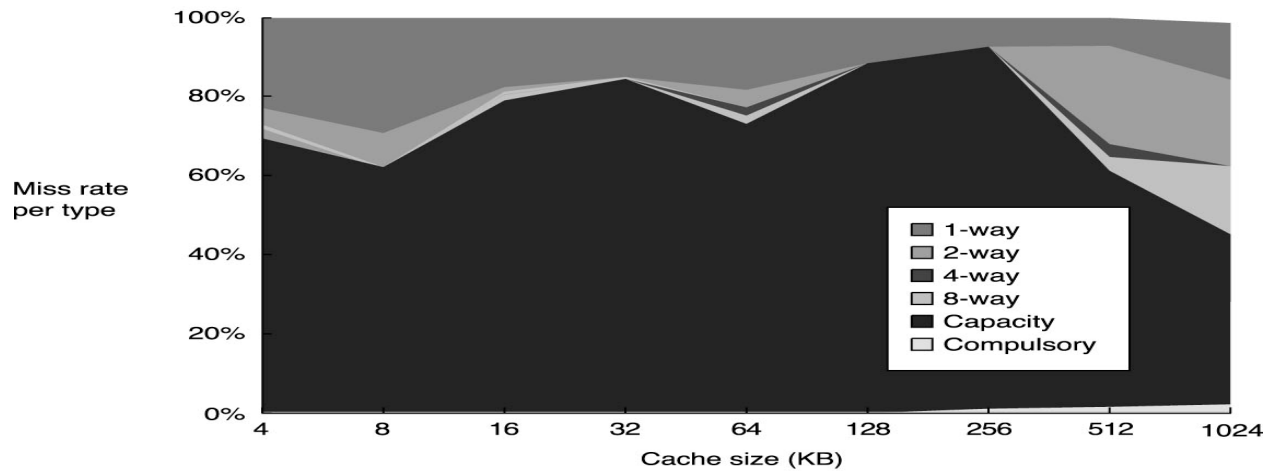
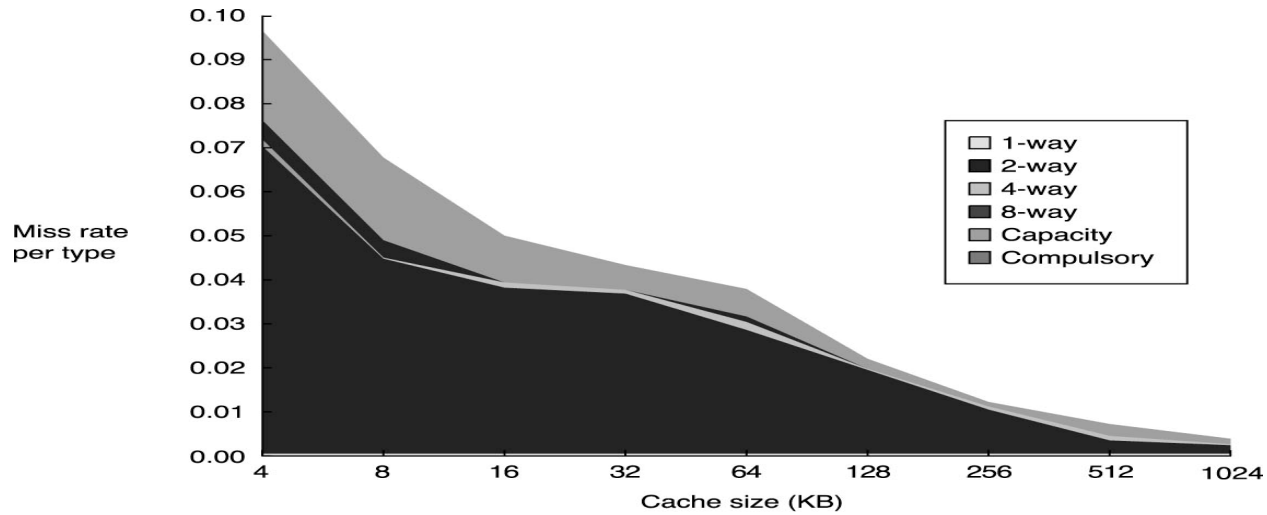
Outline

- Review of the ABCs of Caches
- Cache Performance
- Reducing Cache Miss Penalty (5.4)
- Reducing Miss Rate (5.5)
- Reducing Miss Penalty and Miss Rate via Parallelism (5.6)

Reducing Miss Rates: Types of Cache Misses

- **Compulsory**
 - First reference or cold start misses
- **Capacity**
 - Working set is too big for the cache
 - Fully associative caches
- **Conflict (collision)**
 - Many blocks map to the same block frame (line)
 - Affects
 - Set associative caches
 - Direct mapped caches

Miss Rates: Absolute and Distribution



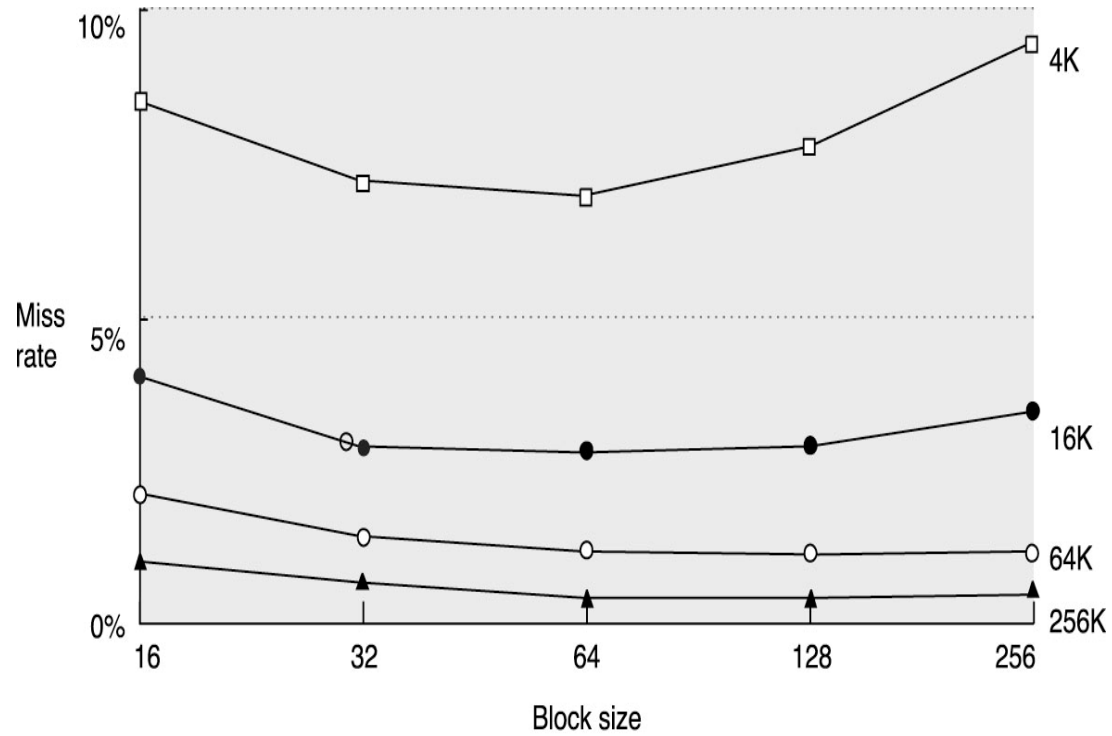
Reducing the Miss Rates

1. Larger block size
2. Larger Caches
3. Higher associativity
4. Pseudo-associative caches
5. Compiler optimizations

1. Larger Block Size

- Effects of larger block sizes
 - Reduction of compulsory misses
 - Spatial locality
 - Increase of miss penalty (transfer time)
 - Reduction of number of blocks
 - Potential increase of conflict misses
- Latency and bandwidth of lower-level memory
 - High latency and bandwidth => large block size
 - Small increase in miss penalty

Example



© 2003 Elsevier Science (USA). All rights reserved.

2. Larger Caches

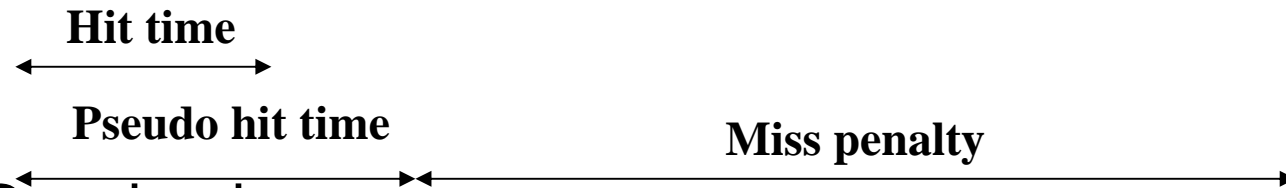
- More blocks
- Higher probability of getting the data
- Longer hit time and higher cost
- Primarily used in 2nd level caches

3. Higher Associativity

- Eight-way set associative is good enough
- 2:1 Cache Rule:
 - Miss Rate of direct mapped cache size N = Miss Rate
2-way cache size $N/2$
- Higher Associativity can increase
 - Clock cycle time
 - Hit time for 2-way vs. 1-way
external cache +10%,
internal + 2%

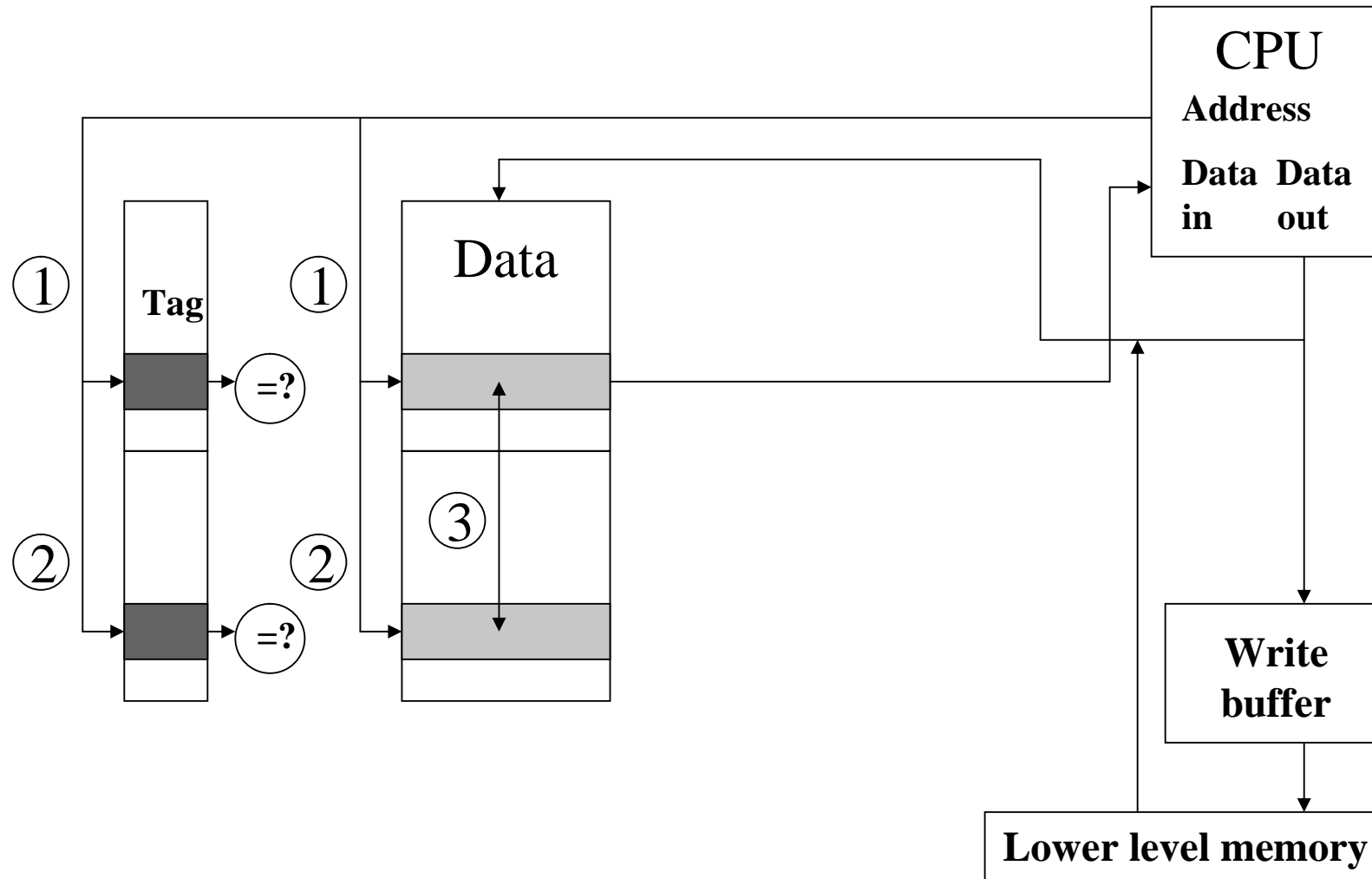
4. Pseudo-Associative Caches

- Fast hit time of *direct mapped* and lower conflict misses of *2-way set-associative* 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 design 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

Pseudo Associative Cache



5. Compiler Optimizations

- Avoid hardware changes
- Instructions
 - Profiling to look at conflicts between groups of instructions
- 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

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

Reducing conflicts between val & key; improved spatial locality

Loop Interchange

```
/* Before */
  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 (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
- Same number of executed instructions

Loop Fusion

```
/* 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 temporal locality

Blocking (1/2)

```
/* 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:**

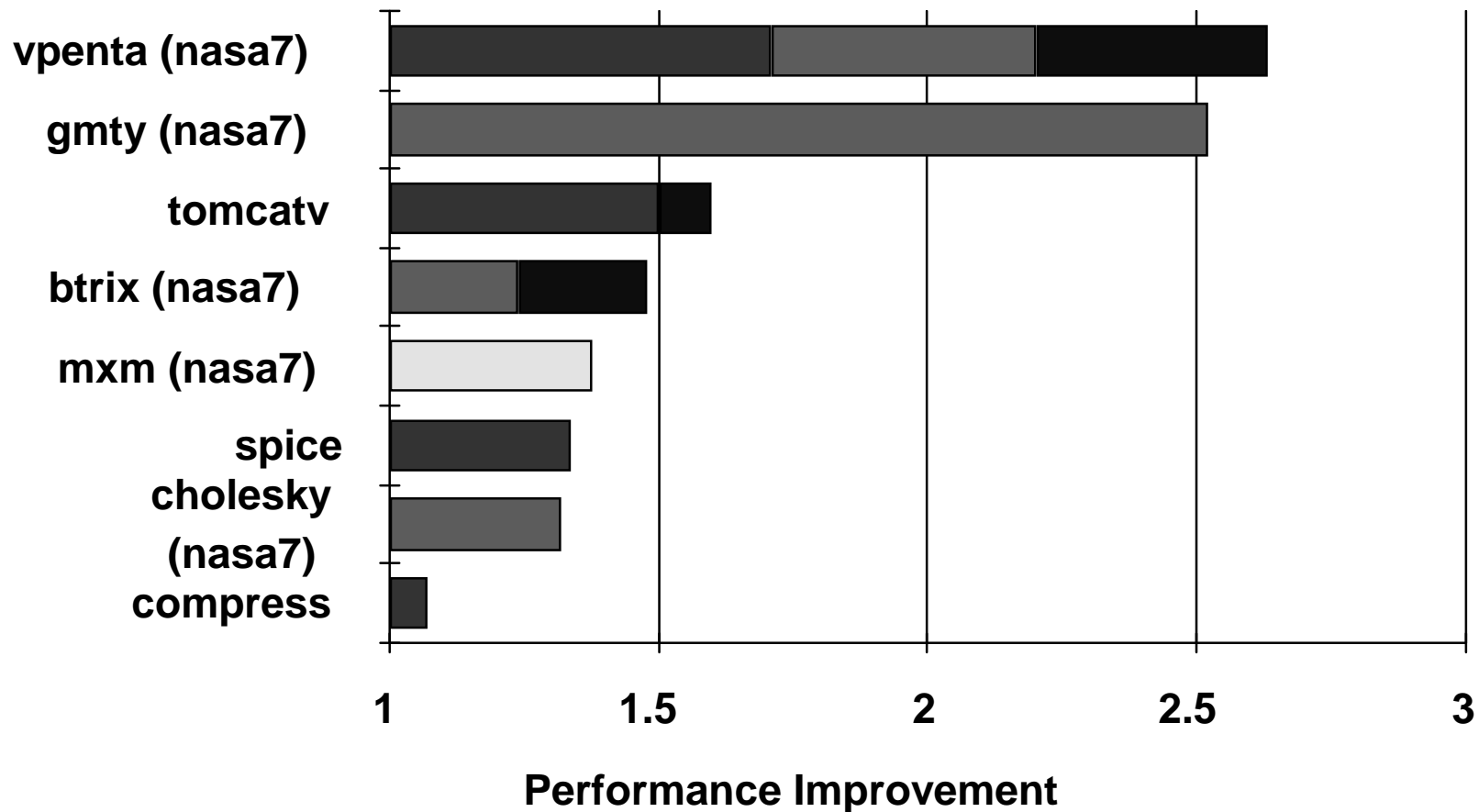
- 3 NxNx4 => no capacity misses
- Idea: compute on BxB submatrix that fits**

Blocking (2/2)

```
/* 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*

Compiler Optimization Performance

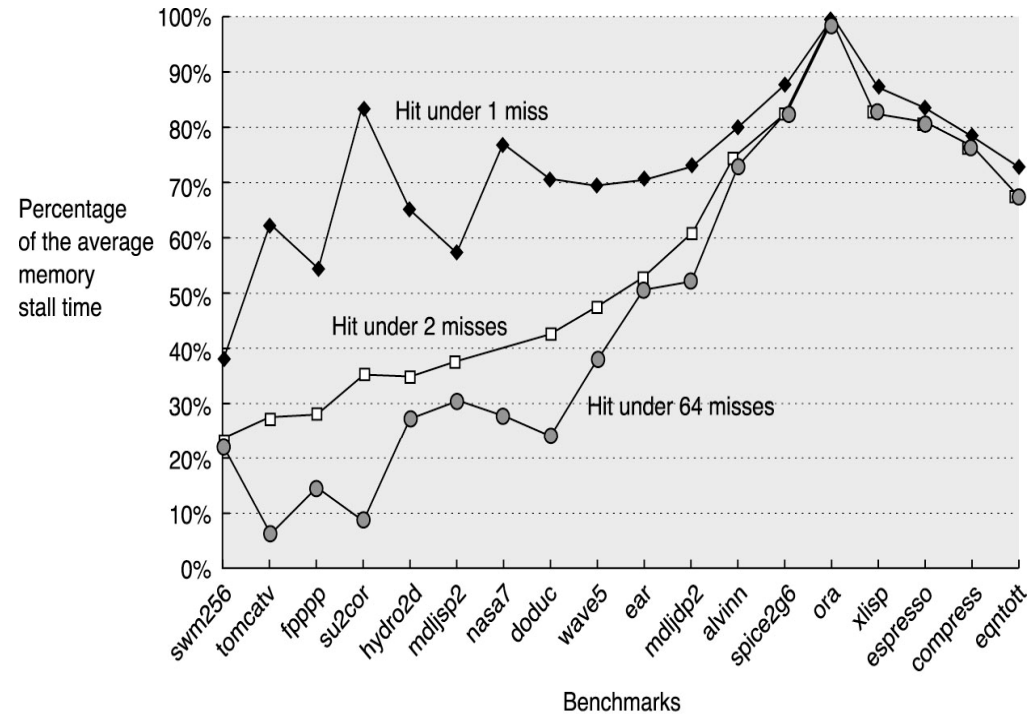


Reducing Cache Miss Penalty or Miss Rate via Parallelism

1. Nonblocking Caches
2. Hardware Prefetching
3. Compiler controlled Prefetching

1. Nonblocking Cache

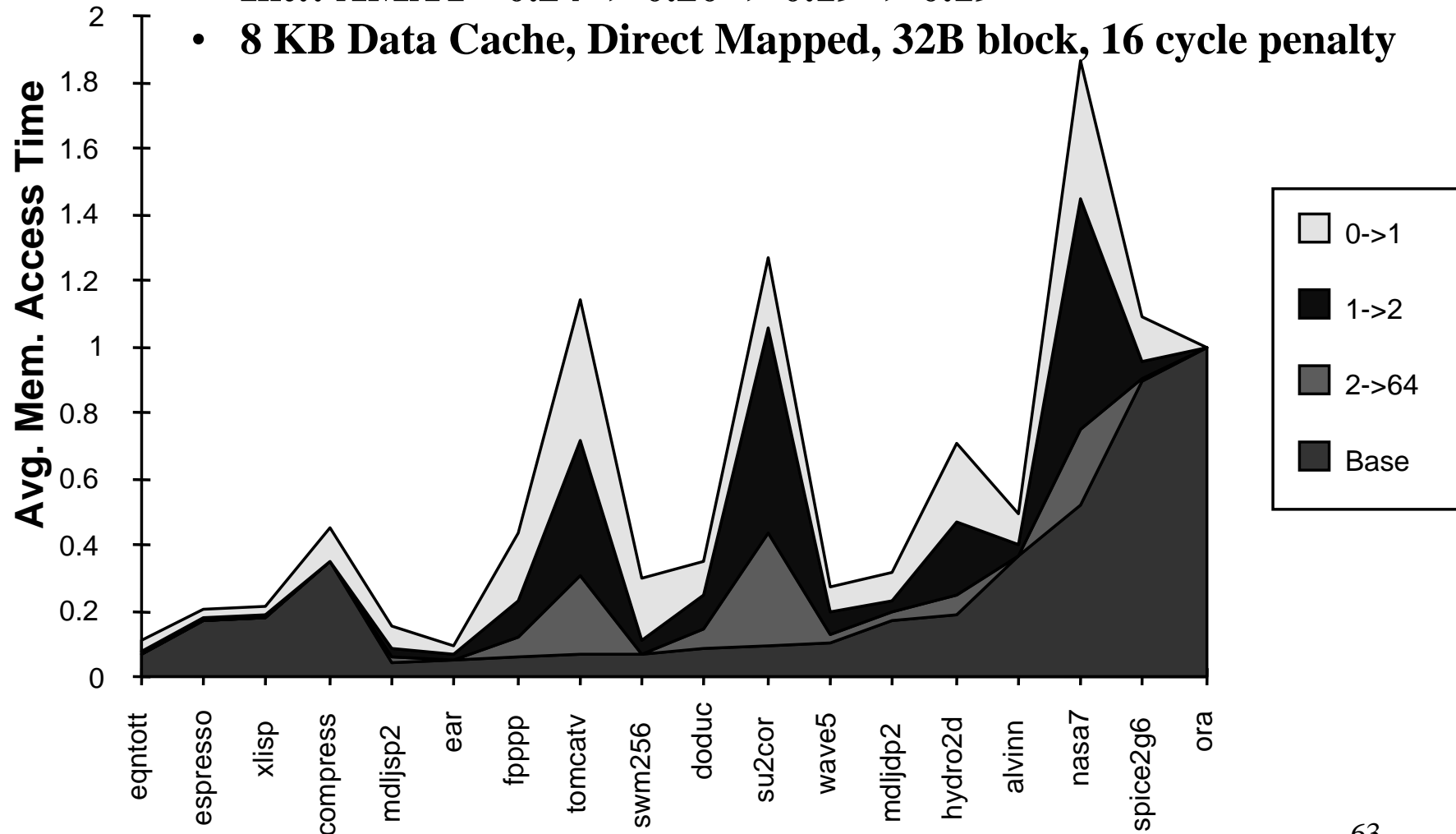
- Out-of-order execution
 - Proceeds with next fetches while waiting for data to come
- Non-blocking caches continue to supply cache hits during a miss
 - requires out-of-order execution CPU
- “hit under miss” reduces the effective miss penalty by working during miss vs. ignoring CPU requests
- “hit under multiple miss” may further lower the effective miss penalty by overlapping multiple misses
 - Significantly increases the complexity of the cache controller
 - Requires multiple memory banks (otherwise cannot support)
 - Pentium Pro allows 4 outstanding memory misses



© 2003 Elsevier Science (USA). All rights reserved.

Hit Under Miss

- **FP: AMAT= 0.68 -> 0.52 -> 0.34 -> 0.26**
- **Int:: AMAT= 0.24 -> 0.20 -> 0.19 -> 0.19**
- **8 KB Data Cache, Direct Mapped, 32B block, 16 cycle penalty**



2. Hardware Prefetching

- 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:
 - 1 data stream buffer gets 25% misses from 4KB DM cache; 4 streams get 43%
 - For scientific programs: 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

3. Compiler-Controlled Prefetching

- Compiler inserts data prefetch instructions
 - Load data into register (HP PA-RISC loads)
 - Cache Prefetch: load into cache (MIPS IV, PowerPC)
 - Special prefetching instructions cannot cause faults; a form of speculative execution
- Nonblocking cache: overlap execution with prefetch
- Issuing Prefetch Instructions takes time
 - Is cost of prefetch issues < savings in reduced misses?
 - Higher superscalar reduces difficulty of issue bandwidth

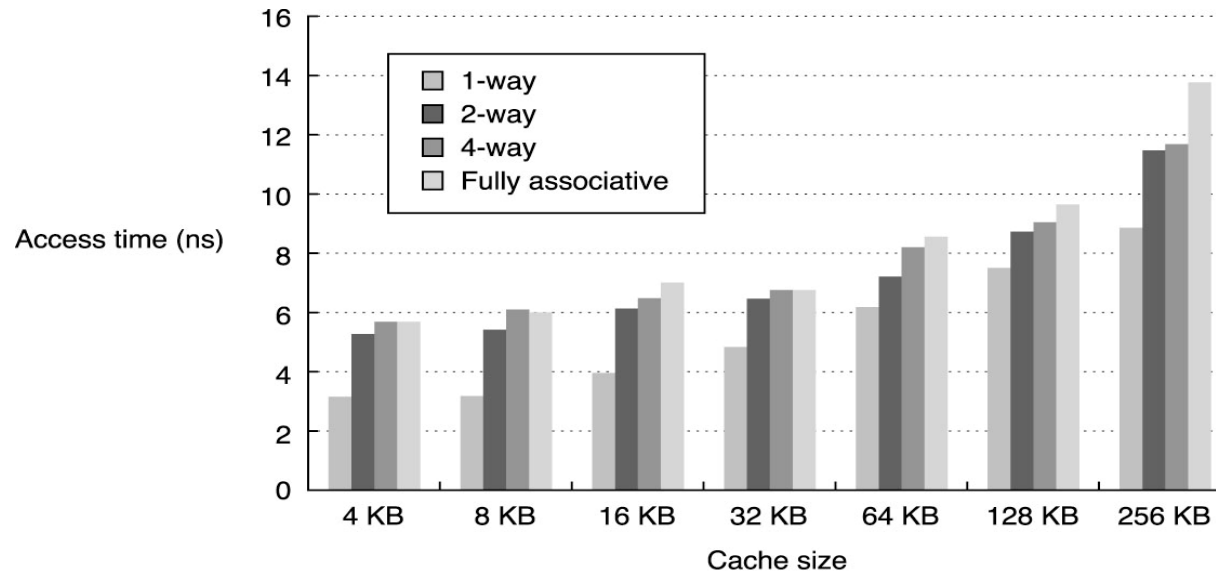
Reducing Hit Time

1. Small and Simple Caches
2. Avoiding address Translation during Indexing of the Cache

1. Small and Simple Caches

- Small hardware is faster
- Fits on the same chip as the processor
- Alpha 21164 has 8KB Instruction and 8KB data cache + 96KB second level cache?
 - Small data cache and fast clock rate
- Direct Mapped, on chip
 - Overlap tag check with data transmission
 - For L2 keep tag check on chip, data off chip → fast tag check, large capacity associated with separate memory chip

Small and Simple Caches

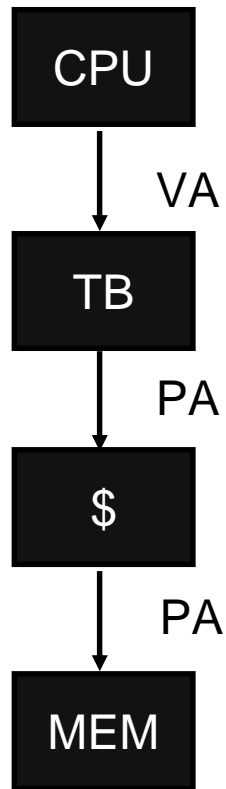


© 2003 Elsevier Science (USA). All rights reserved.

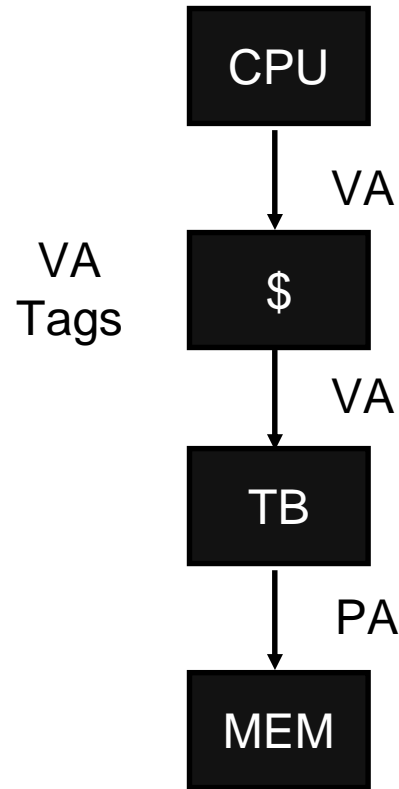
2. Avoiding Address Translation

- *Virtually Addressed Cache* (vs. *Physical Cache*)
 - Send virtual address to cache.
 - Every time process is switched must flush the cache;
 - Cost: time to flush + “compulsory” misses from empty cache
 - Dealing with *aliases* (two different virtual addresses map to same physical address)
 - I/O must interact with cache, so need virtual address
- **Solution to aliases**
 - HW guarantees that every cache block has unique PA
 - SW guarantee (*page coloring*): lower n bits must have same address; as long as covers index field & direct mapped, they must be unique;
- **Solution to cache flush**
 - *PID tag* that identifies process and address within process

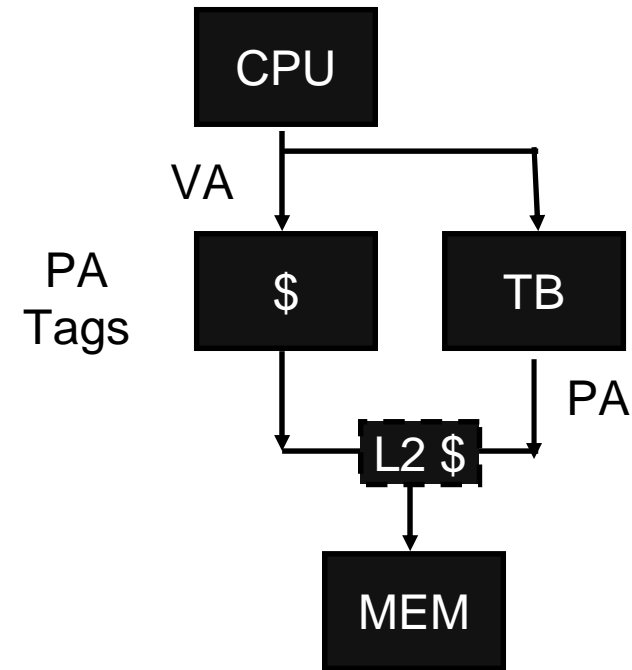
Virtual Addressed Caches



Conventional Organization

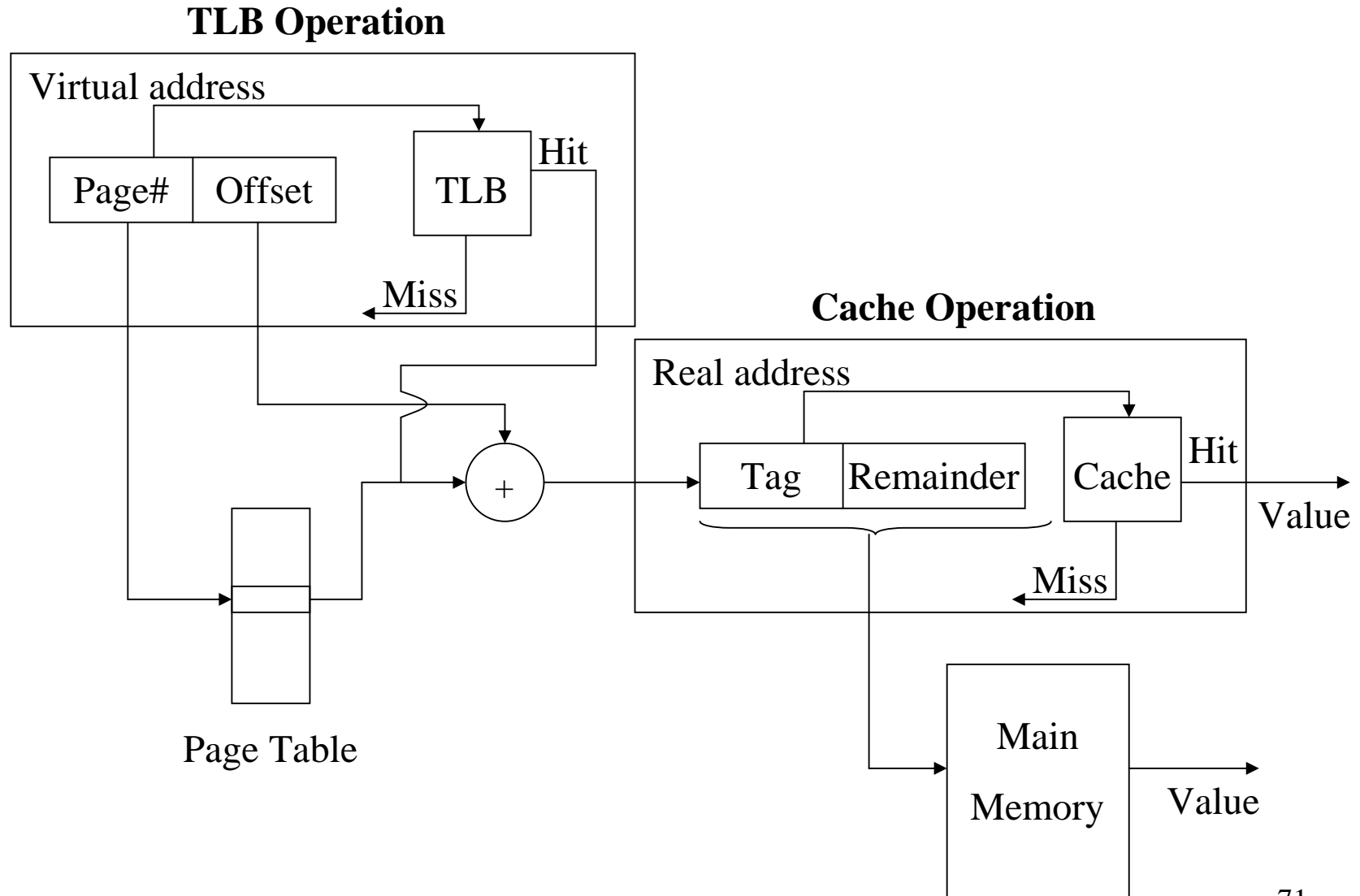


Virtually Addressed Cache
Translate only on miss
Synonym Problem

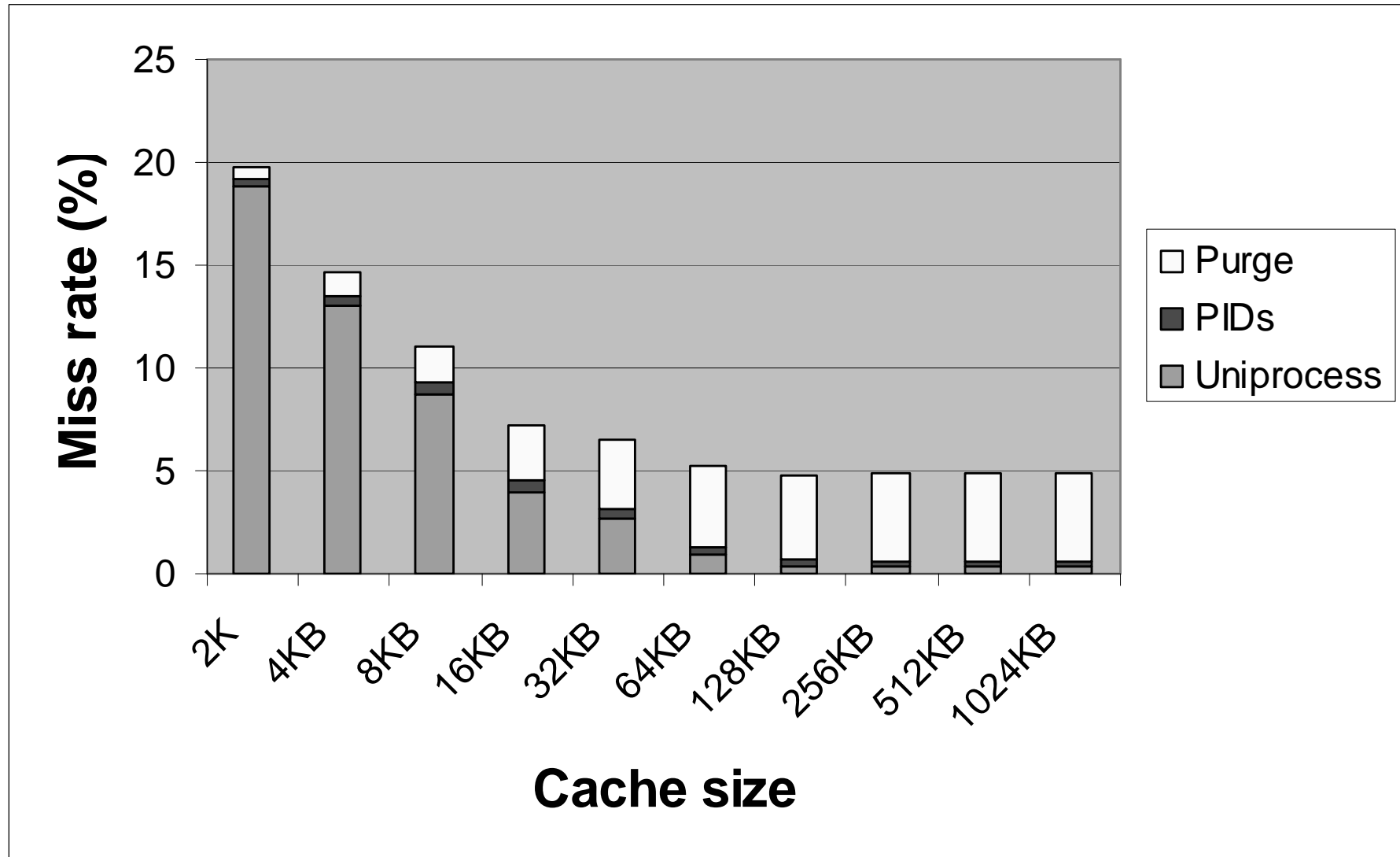


Overlap \$ access with VA translation: requires \$ index to remain invariant across translation

TLB and Cache Operation

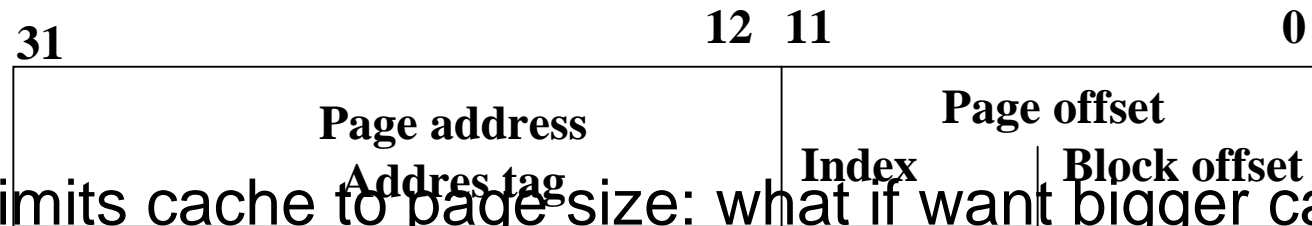


Process ID Impact



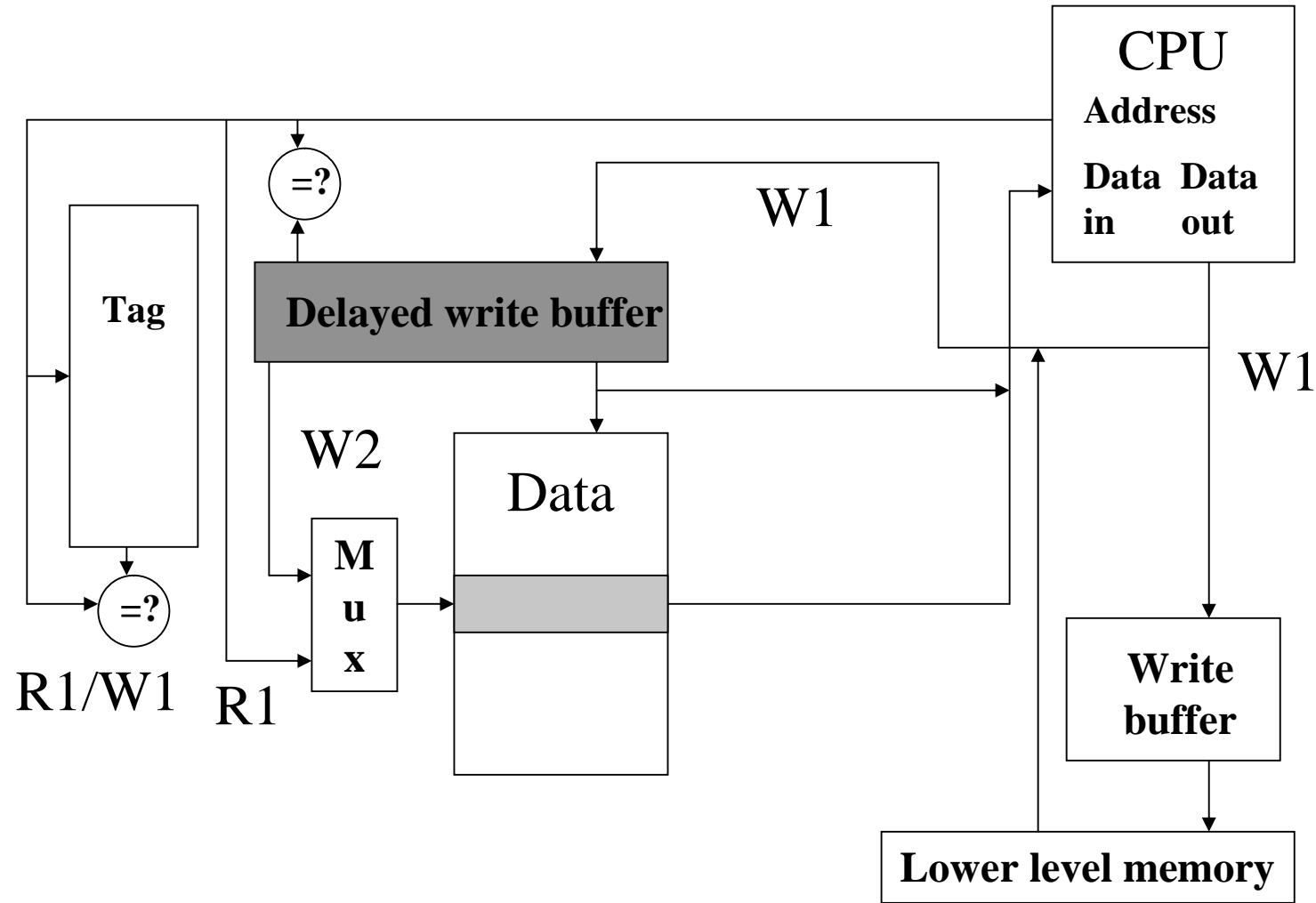
Index with Physical Portion of Address

- If index is physical part of address, can start tag access in parallel with translation so that can compare to physical tag



- Limits cache to page-size: what if want bigger caches and uses same trick?
 - Larger page sizes
 - Higher associativity
 - $\text{Index} = \log(\text{Cache Size} / [\text{block size} * \text{associativity}])$
 - Page coloring

3. Pipelined Writes



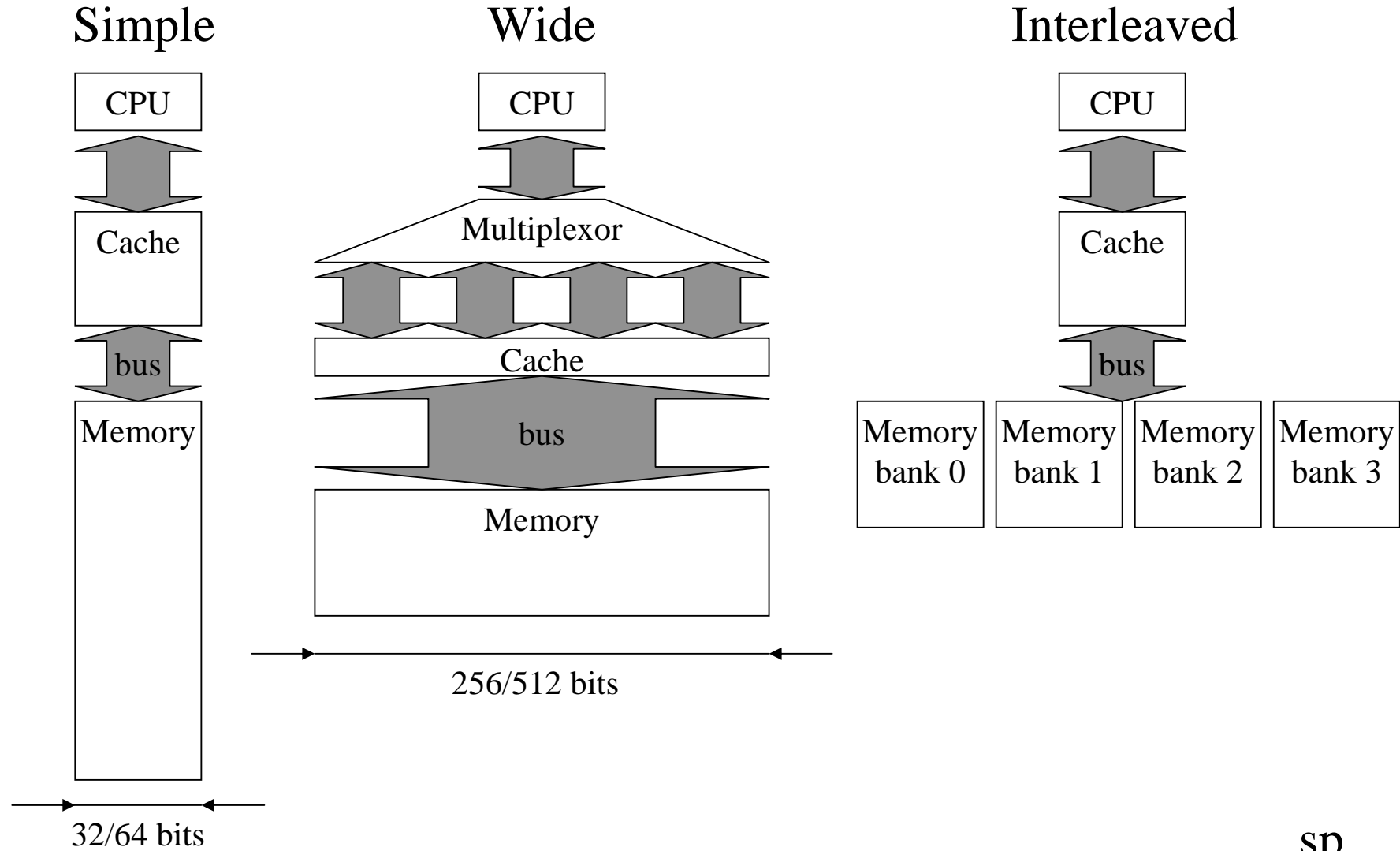
Cache Performance Summary

- Important Summary Table (Fig. 5.26)
- Understand the underlying tradeoffs
 - E.g. victim caches benefit both miss penalty and miss rates.
 - E.g. small caches improve hit rate but increase miss rate

Main Memory Background

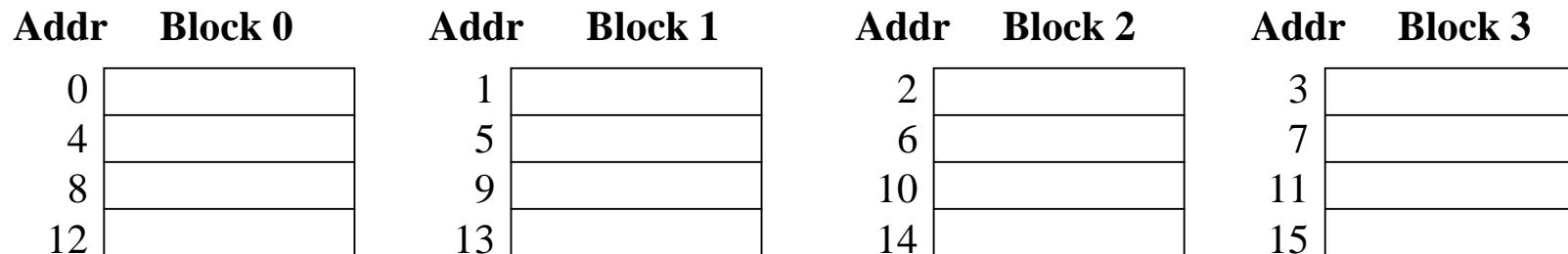
- Performance of Main Memory:
 - **Latency: Cache Miss Penalty**
 - *Access Time*: time between request and word arrives
 - *Cycle Time*: time between requests
 - **Bandwidth: I/O & Large Block Miss Penalty (L2)**
- Main Memory is *DRAM*: Dynamic Random Access Memory
 - **Dynamic since needs to be refreshed periodically**
 - **Addresses divided into 2 halves (Memory as a 2D matrix):**
 - *RAS* or *Row Access Strobe*
 - *CAS* or *Column Access Strobe*
- Cache uses *SRAM*: Static Random Access Memory
 - **No refresh (6 transistors/bit vs. 1 transistor /bit, area is 10X)**
 - **Address not divided: Full address**
- *Size*: DRAM/SRAM - 4-8
Cost & Cycle time: SRAM/DRAM - 8-16

Main Memory Organizations



Performance

- Timing model (word size is 32 bits)
 - 1 to send address,
 - 6 access time, 1 to send data
 - Cache Block is 4 words
- *Simple M.P.* $= 4 \times (1+6+1) = 32$
- *Wide M.P.* $= 1 + 6 + 1 = 8$
- *Interleaved M.P.* $= 1 + 6 + 4 \times 1 = 11$



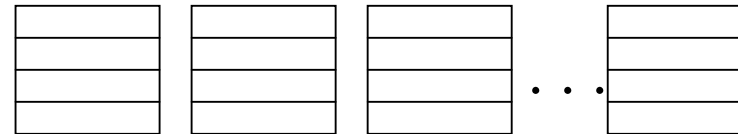
Four-way interleaved memory

Independent Memory Banks

- **Memory banks for independent accesses**
 - Multiprocessor
 - I/O
 - CPU with Hit under n Misses, Non-blocking Cache
- **Superbank: all memory active on one block transfer (or Bank)**



- **Bank: portion within a superbank that is word interleaved (or Subbank)**



Superbank number	Superbank offset	
	Bank number	Bank offset

Number of banks

- How many banks?

number banks \geq number clocks to access word in bank

- For sequential accesses, otherwise will return to original bank before it has next word ready
- (like in vector case)

- Increasing DRAM \Rightarrow fewer chips \Rightarrow harder to have banks

- 64MB main memory
 - 512 memory chips of 1-Mx1 (16 banks of 32 chips)
 - 8 64-Mx1-bit chips (maximum: one bank)
 - Wider paths (16 Mx4bits or 8Mx8bits)

Avoiding Bank Conflicts

- Lots of banks

```
int x[256][512];
for (j = 0; j < 512; j = j+1)
    for (i = 0; i < 256; i = i+1)
        x[i][j] = 2 * x[i][j];
```

- Even with 128 banks ($512 \bmod 128=0$), conflict on word accesses
- SW: loop interchange or array not power of 2 (“array padding”)
- HW: Prime number of banks
 - **bank number = address mod number of banks**
 - **address within bank = address / number of banks**
 - **modulo & divide per memory access with prime no. banks?**
 - **Let number of banks be = prime number = $2^K - 1$**
 - **address within bank = address mod number words in bank**
 - **easy if 2^N words per bank \rightarrow from chinese remainder theorem**

Fast Bank Number

- Chinese Remainder Theorem

As long as two sets of integers a_i and b_i follow these rules

$$b_i = x \bmod a_i, 0 \leq b_i < a_i, 0 \leq x < a_0 \cdot a_1 \cdot a_2 \cdot \dots$$

and a_i and a_j are co-prime if $i \neq j$, then the integer x has only one solution (unambiguous mapping):

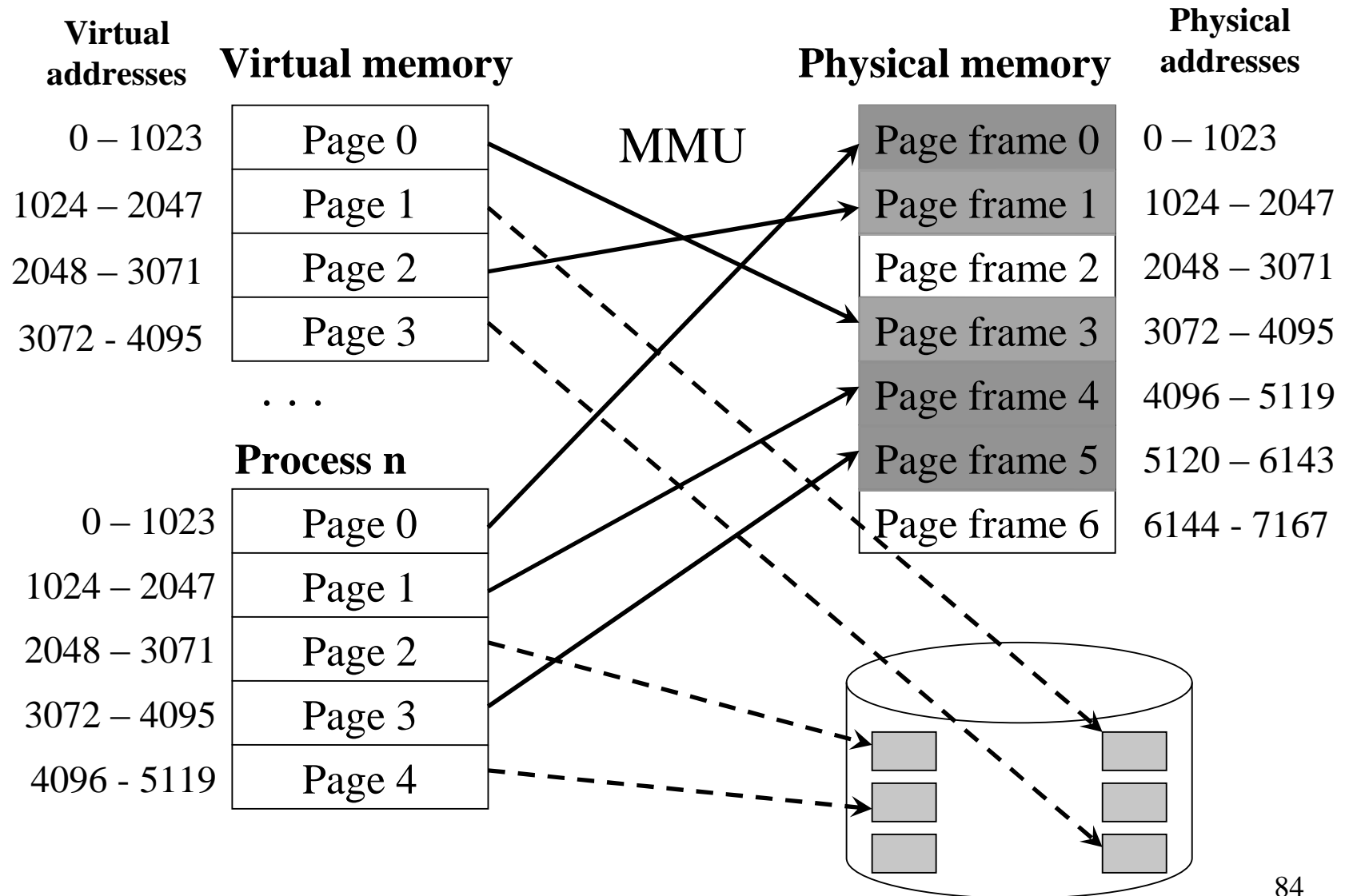
- **bank number = b_0 , number of banks = a_0 (3 in example)**
- **address within bank = b_1 , # of words in bank = a_1 (8 in ex)**
- **N word address 0 to N-1, prime no. banks, words power of 2**

		Seq. Interleaved			Modulo Interleaved		
Bank Number:		0	1	2	0	1	2
Address							
within Bank:	0	0	1	2	0	16	8
	1	3	4	5	9	1	17
	2	6	7	8	18	10	2
	3	9	10	11	3	19	11
	4	12	13	14	12	4	20
	5	15	16	17	21	13	5
	6	18	19	20	6	22	14
	7	21	22	23	15	7	23

Virtual Memory

- Overcoming main memory size limitation
- Sharing of main memory among processes
- Virtual memory model
 - Decoupling of
 - Addresses used by the program (virtual)
 - Memory addresses (physical)
 - Physical memory allocation
 - Pages
 - Segments
- Process relocation
- Demand paging

Virtual/Physical Memory Mapping



Caches vs. Virtual Memory

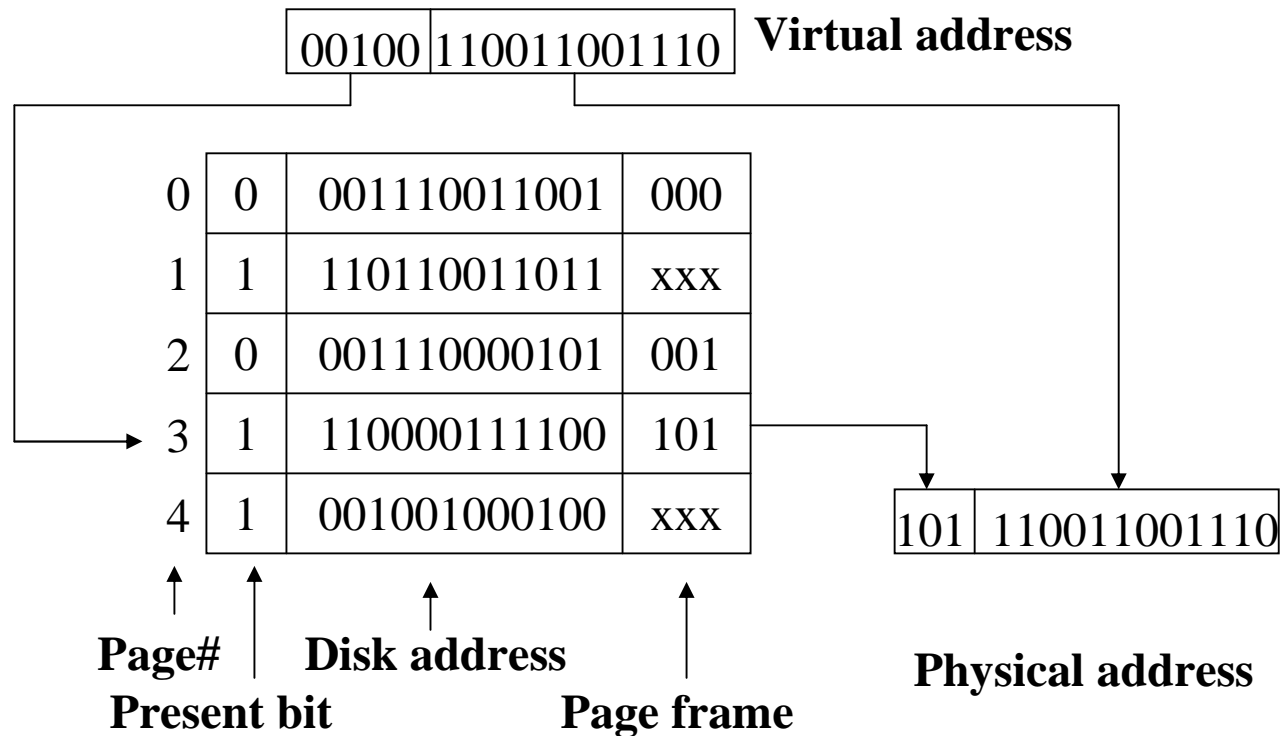
- Quantitative differences
 - Block (page) size
 - Hit time
 - Miss (page fault) penalty
 - Miss (page fault) rate
 - Size
- Replacement control
 - Cache: hardware
 - Virtual memory: OS
- Size of virtual address space = $f(\text{address size})$
- Disks are also used for the file system

Design Elements

- **Minimize page faults**
- Block size
- Block placement
 - Fully associative
- Block identification
 - Page table
- Replacement Algorithm
 - LRU
- Write Policy
 - Write back

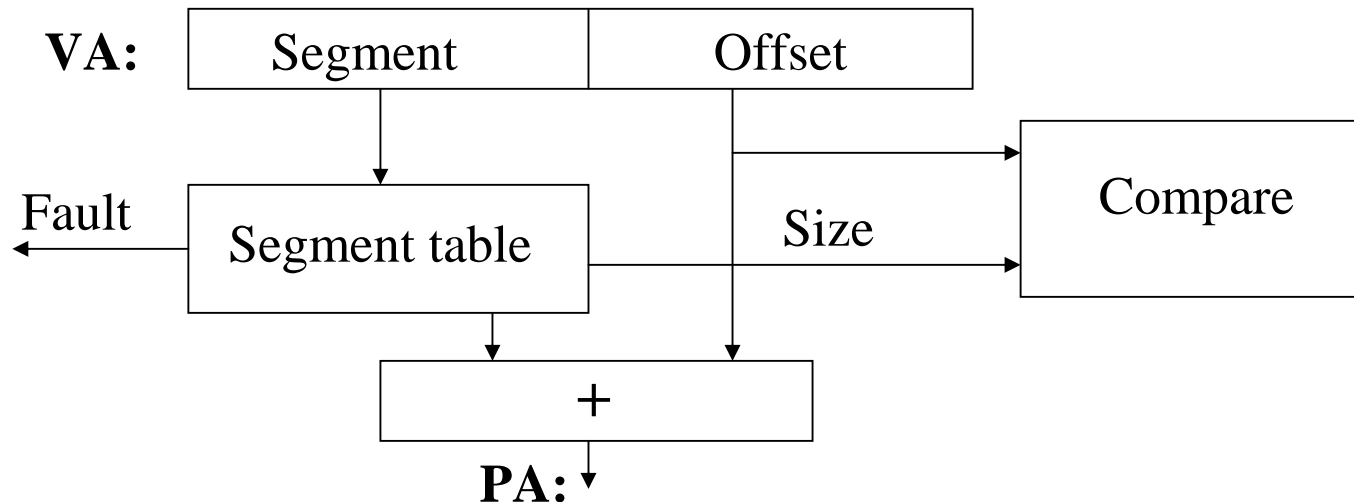
Page Tables

- Each process has one or more page tables
- Size of Page table (31-bit address, 4KB pages => 2MB)
 - Two-level approach: 2 virtual-to-physical translations
 - Inverted page tables



Segmentation

- Visible to the programmer
- Multiple address spaces of variable size
 - Segment table: start address and size
 - Segment registers (x86)
- Advantages
 - Simplifies handling of growing data structures
 - Independent code segments



Paging vs. Segmentation

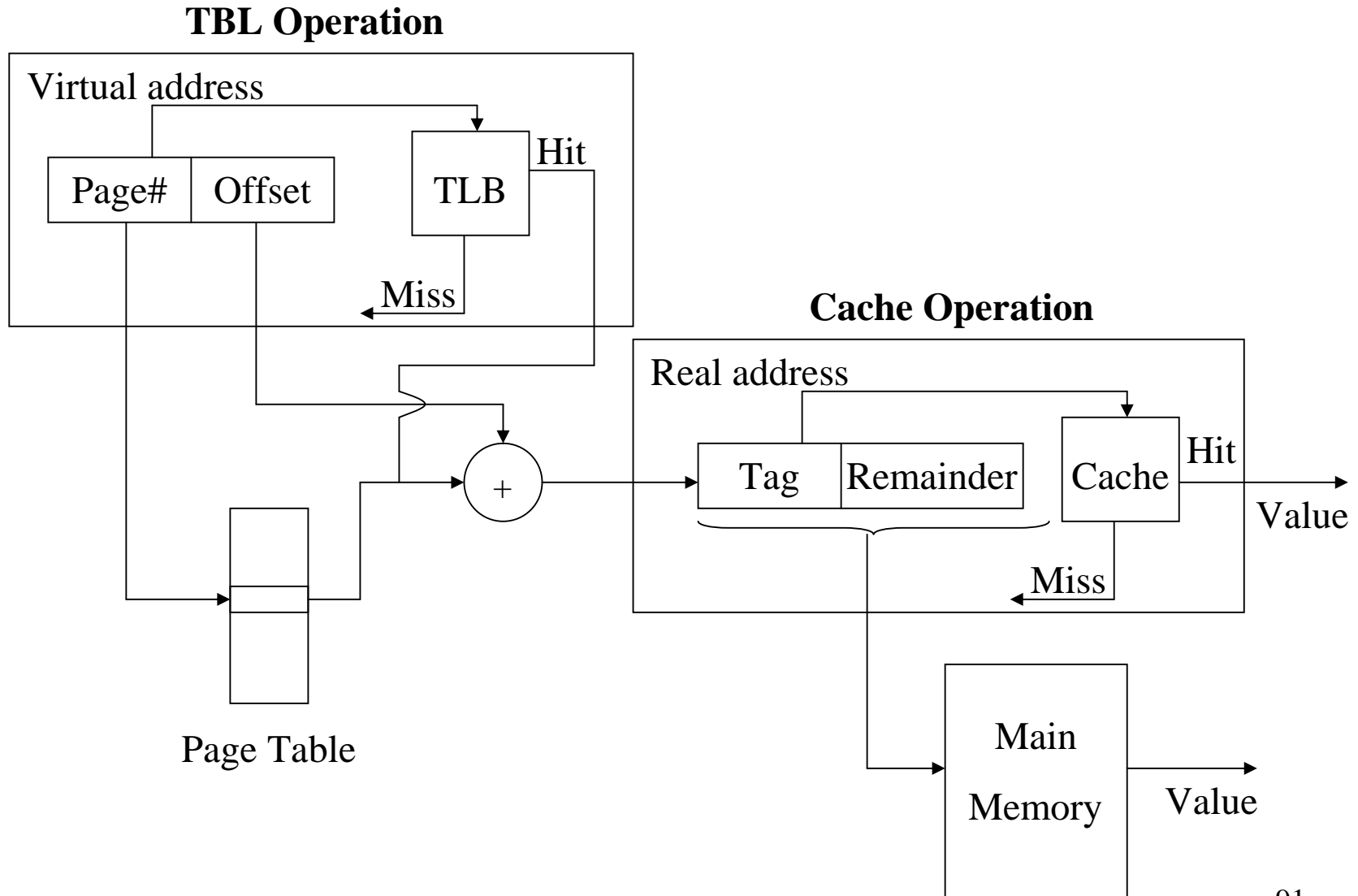
	Page	Segment
Address	One word	Two words
Programmer visible?	No	Maybe
Block replacement	Trivial	Hard
Fragmentation	Internal	external
Disk traffic	Efficient	Not efficient

Hybrids: Paged segments
Multiple page sizes

Translation Buffer

- Fast address translation
- Principle of locality
- Cache for the page table
 - Tag: portion of the virtual address
 - Data: page frame number, protection field, valid, use, and dirty bit
- Virtual cache index and physical tags
- Address translation on the critical path
 - Small TLB
 - Pipelined TLB
- TLB misses

TLB and Cache Operation



Page Size

- Large size
 - Smaller page tables
 - Faster cache hit times
 - Efficient page transfer
 - Less TLB misses
- Small size
 - Less internal fragmentation
 - Process start-up time

Memory Protection

- Multiprogramming
 - Protection and sharing \Leftrightarrow Virtual memory
 - Context switching
- Base and bound registers
 - $(\text{Base} + \text{Address}) \leq \text{Bound}$
- Hardware support
 - Two execution modes: user and kernel
 - Protect CPU state: base/bound registers, user/kernel mode bits, and the exception enable/disable bits
 - System call mechanism

Protection and Virtual Memory

- During the virtual to physical mapping
 - Check for errors or protection
 - Add permission flags to each page/segment
 - Read/write protection
 - User/kernel protection
- Protection models
 - Two-level model: user/kernel
 - Protection rings
 - Capabilities

Memory Hierarchy Design Issues

- Superscalar CPU and number of ports to the cache
 - Multiple issue processors
 - Non-blocking caches
- Speculative execution and conditional instructions
 - Can generate invalid addresses (exceptions) and cache misses
 - Memory system must identify speculative instructions and suppress the exceptions and cache stalls on a miss
- Compilers: ILP versus reducing cache misses

```
for (i = 0; i < 512; i = i + 1)
    for (j = 0; j < 512; j = j + 1)
        x[i][j] = 2 * x[i][j-1];
```
- I/O and cache coherency

Coherency

