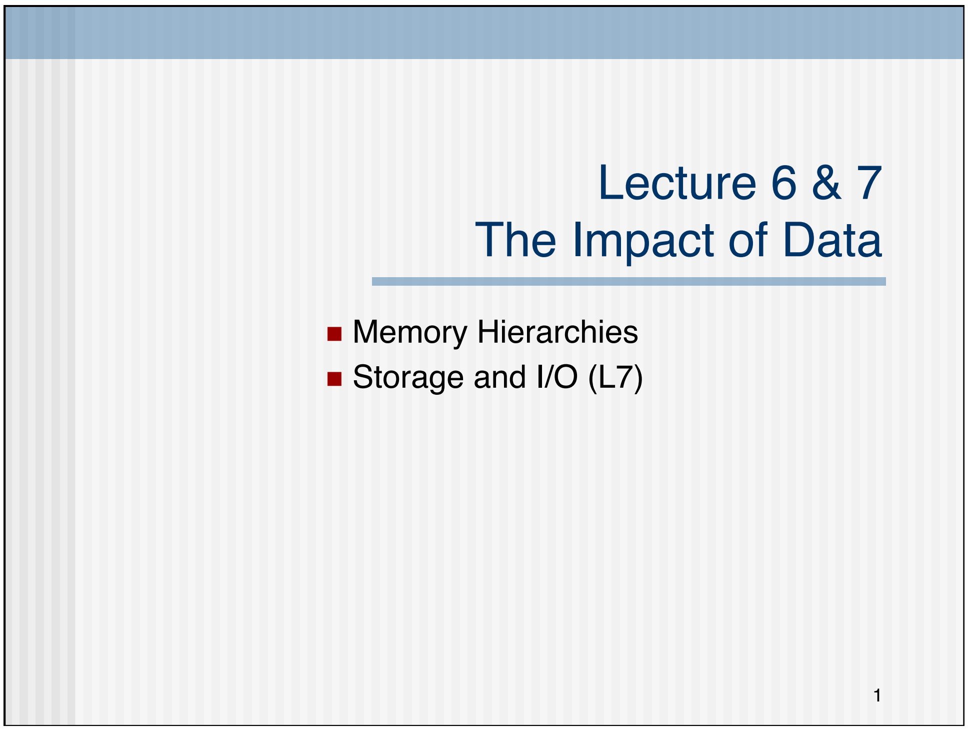
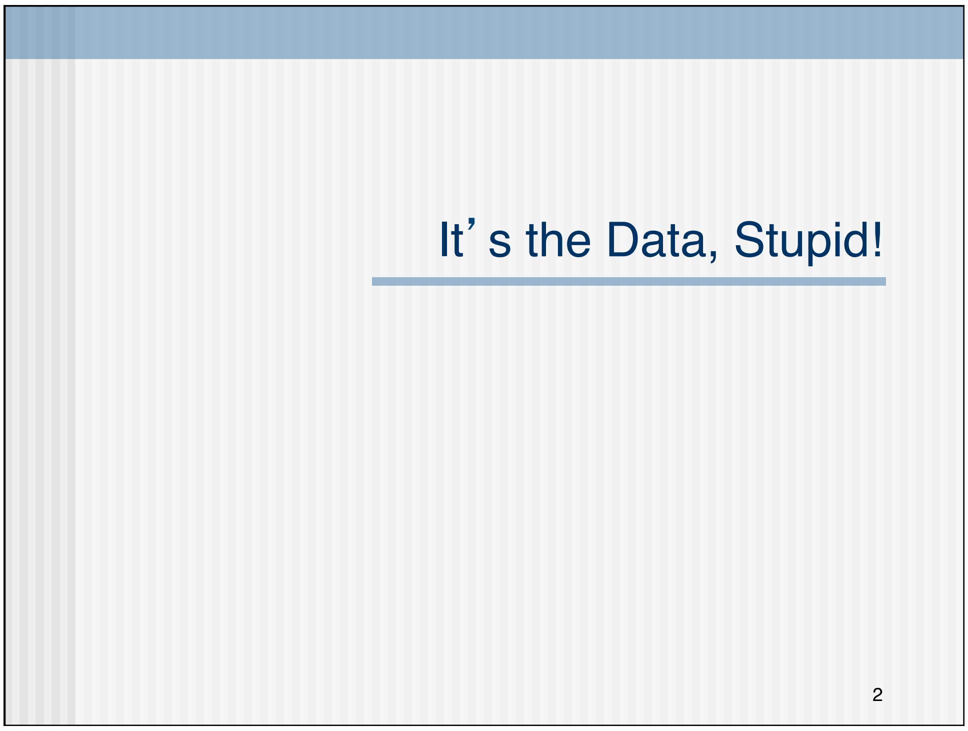


Lecture 6 & 7
The Impact of Data

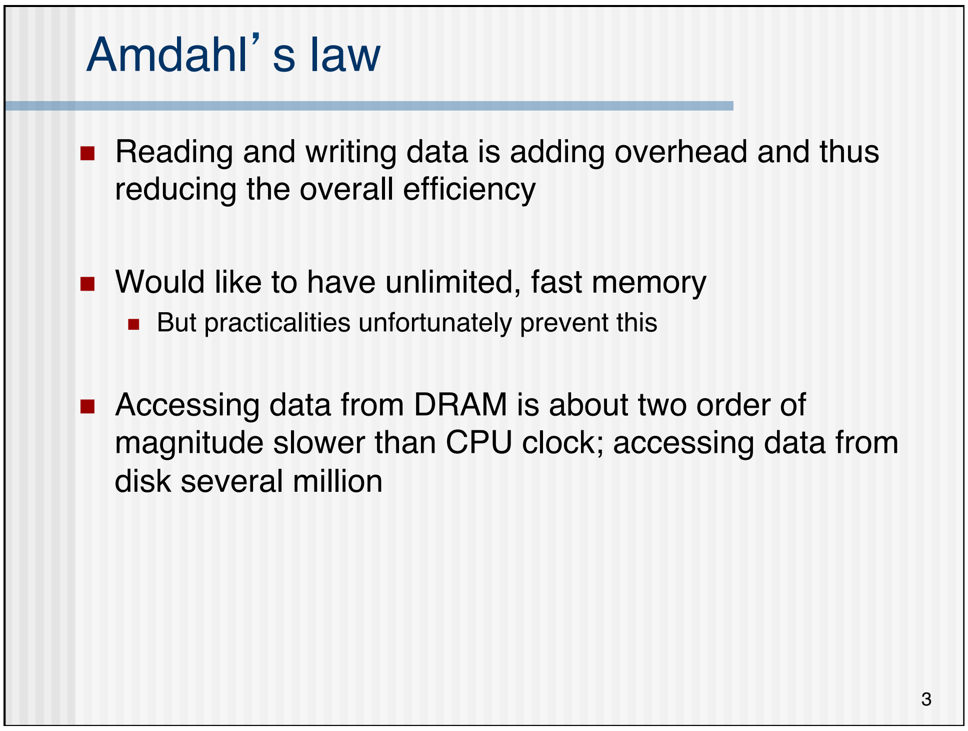
- Memory Hierarchies
- Storage and I/O (L7)

1



It’s the Data, Stupid!

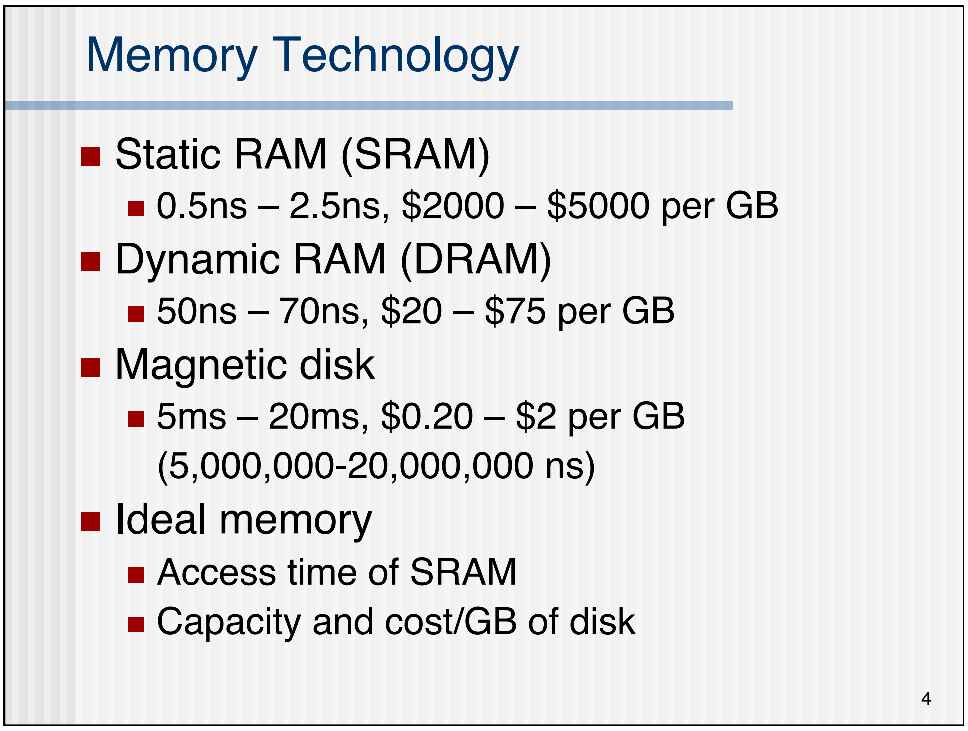
2



Amdahl’ s law

- Reading and writing data is adding overhead and thus reducing the overall efficiency
- Would like to have unlimited, fast memory
  - But practicalities unfortunately prevent this
- Accessing data from DRAM is about two order of magnitude slower than CPU clock; accessing data from disk several million

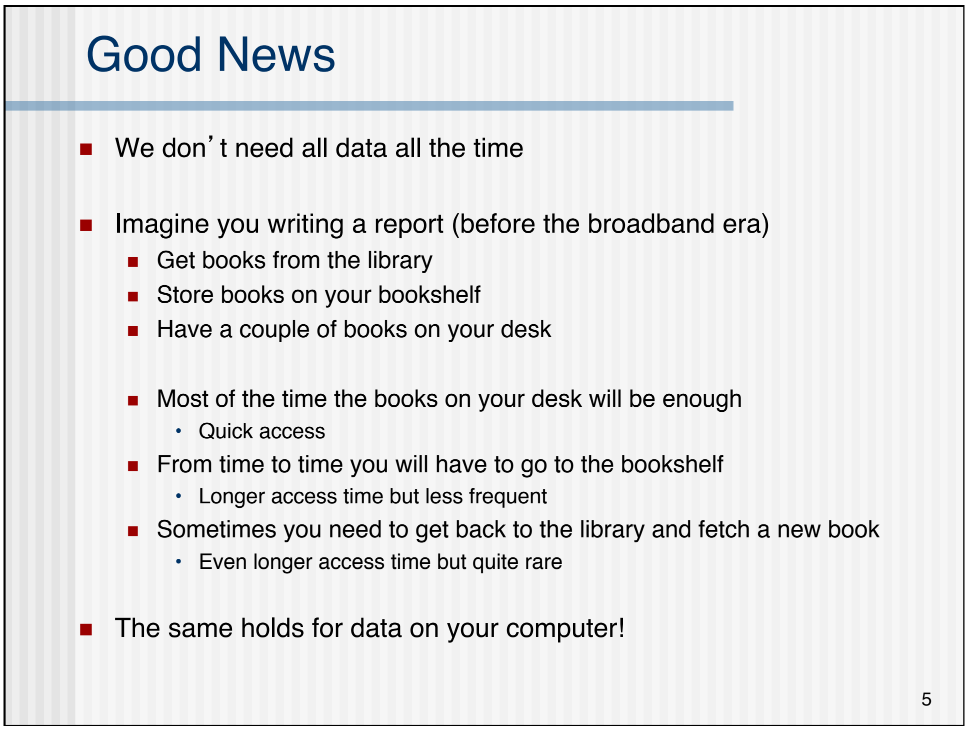
3



Memory Technology

- Static RAM (SRAM)
  - 0.5ns – 2.5ns, \$2000 – \$5000 per GB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, \$20 – \$75 per GB
- Magnetic disk
  - 5ms – 20ms, \$0.20 – \$2 per GB (5,000,000-20,000,000 ns)
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

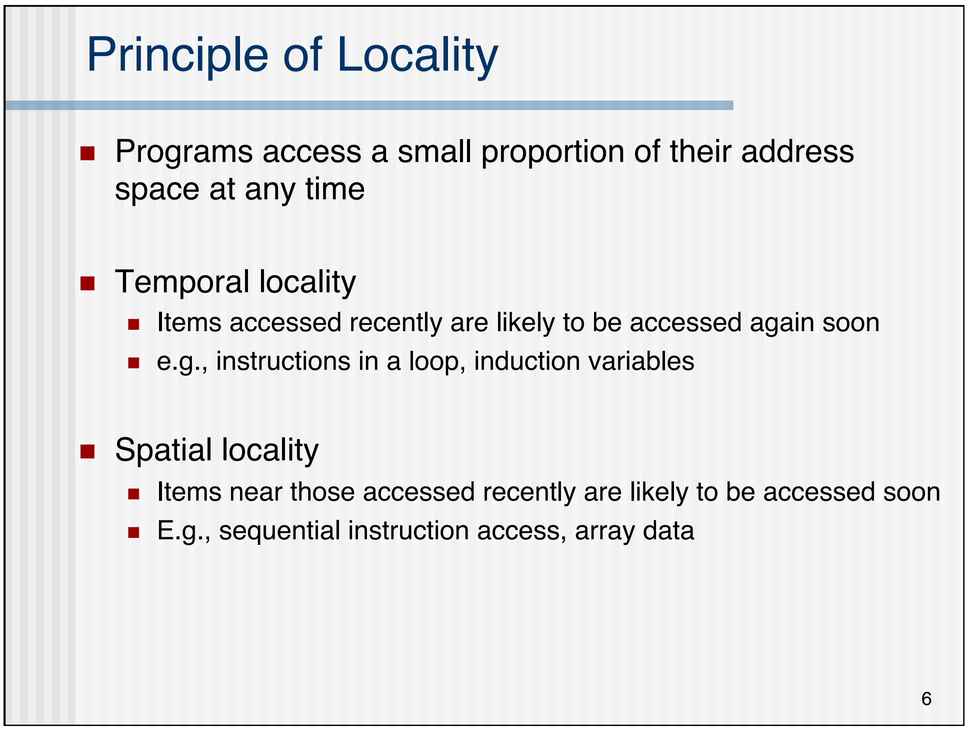
4



Good News

- We don’t need all data all the time
- Imagine you writing a report (before the broadband era)
  - Get books from the library
  - Store books on your bookshelf
  - Have a couple of books on your desk
  - Most of the time the books on your desk will be enough
    - Quick access
  - From time to time you will have to go to the bookshelf
    - Longer access time but less frequent
  - Sometimes you need to get back to the library and fetch a new book
    - Even longer access time but quite rare
- The same holds for data on your computer!

5



Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data

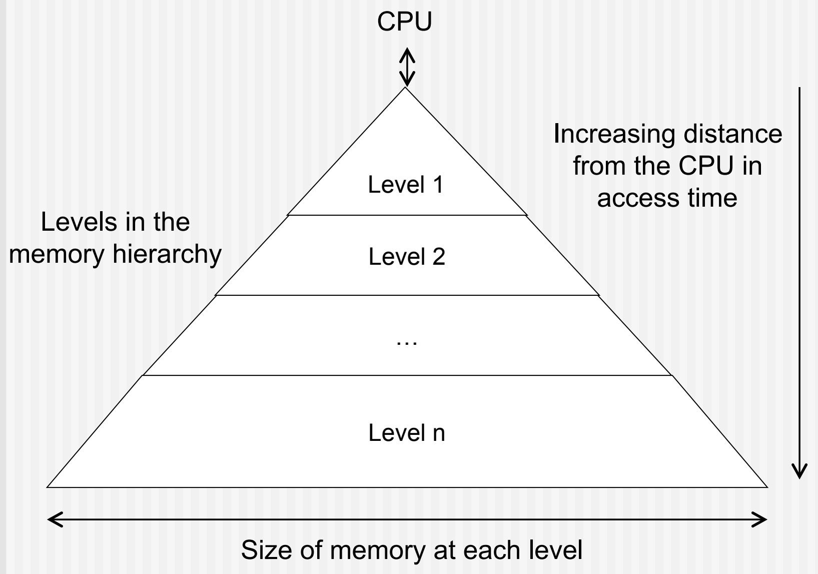
6

Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU

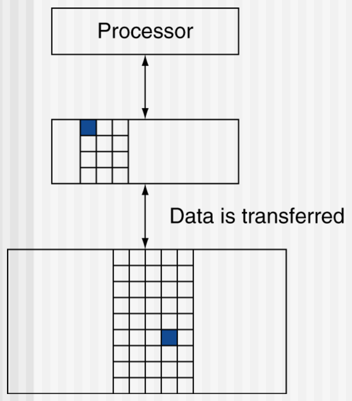
7

Memory Hierarchy Levels


CPU
Levels in the memory hierarchy
Level 1
Level 2
…
Level n
Increasing distance from the CPU in access time
Size of memory at each level

8

Memory Hierarchy Levels



- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses
      = 1 – hit ratio
  - Then accessed data supplied from upper level

9

Cache

Cache: a safe place for hiding or storing things.

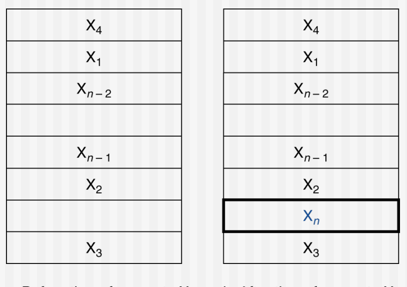
Webster's New World Dictionary of the American Language

- In the report example the desk acted as cache
  - And the bookshelf as main memory
  - And the library as disk storage

10

Cache Memory

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses $X_1, \ldots, X_{n-1}, X_n$



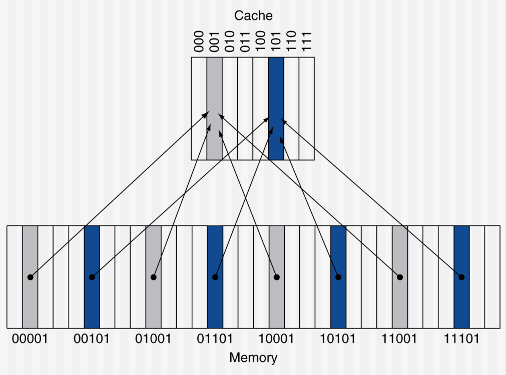
a. Before the reference to X_n b. After the reference to X_n

- How do we know if the data is present?
- Where do we look?

11

Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits

12

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

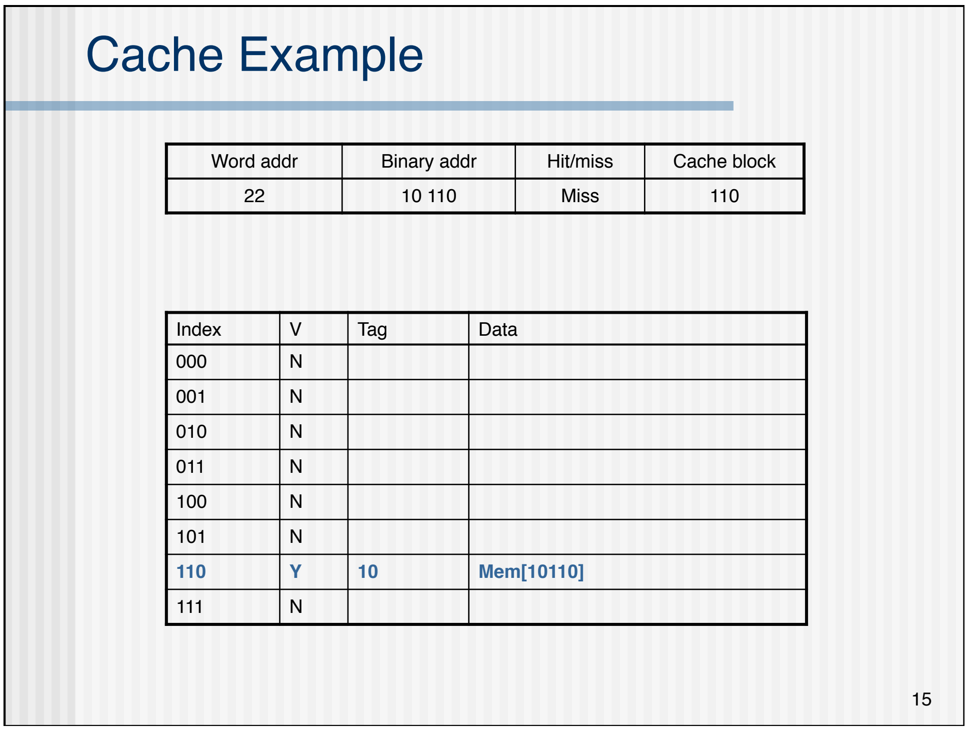
13

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

14

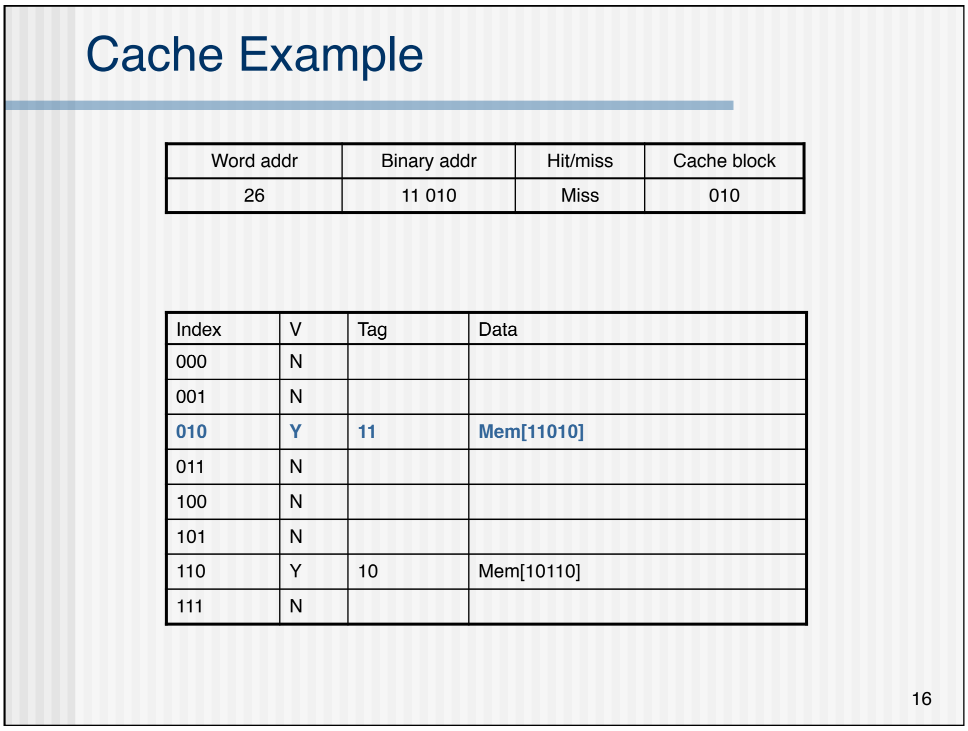


Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

15

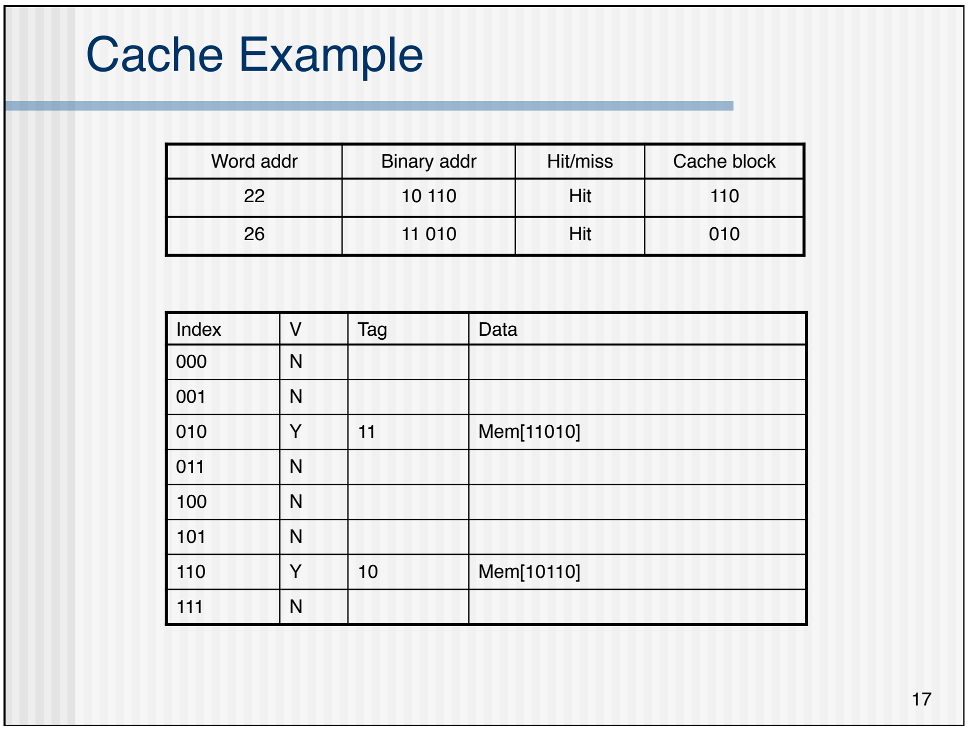


Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

16

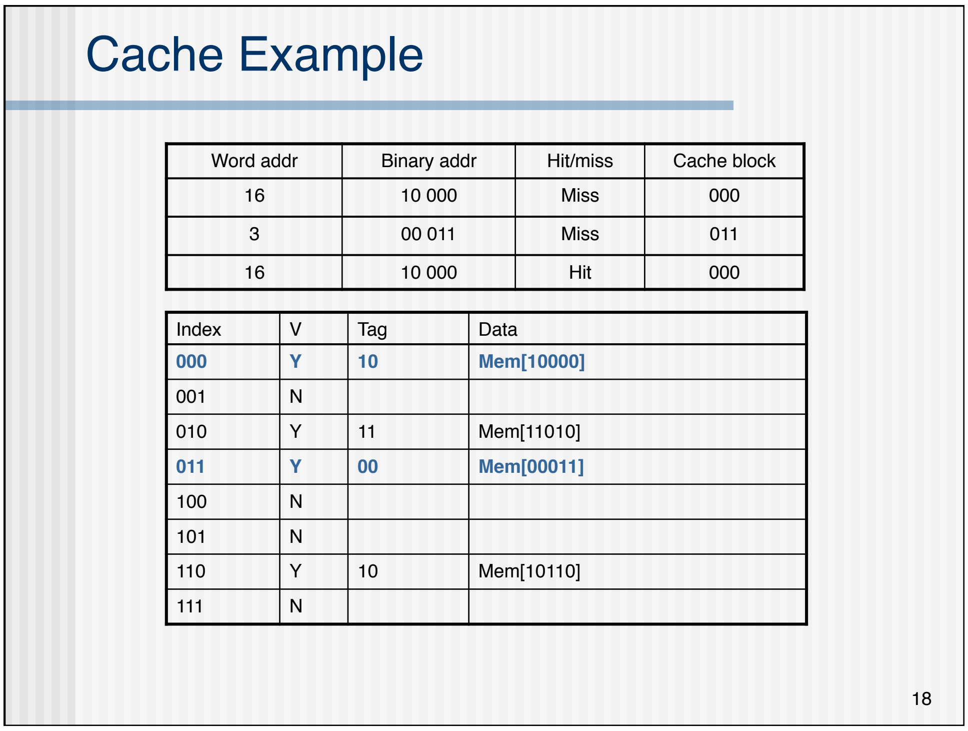


Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

17

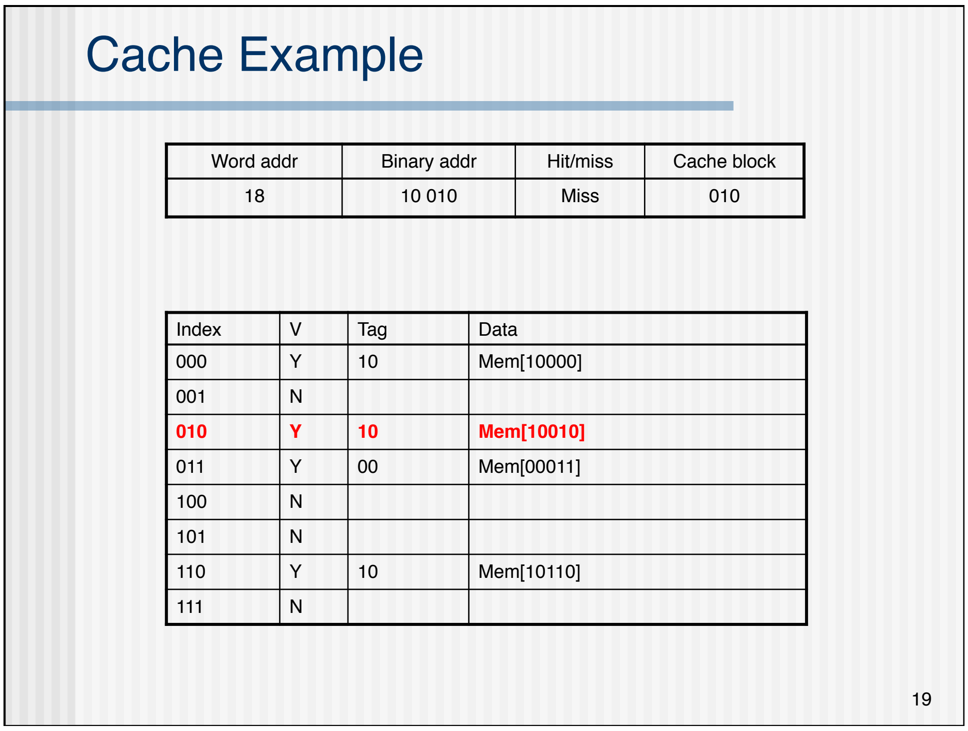


Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

18

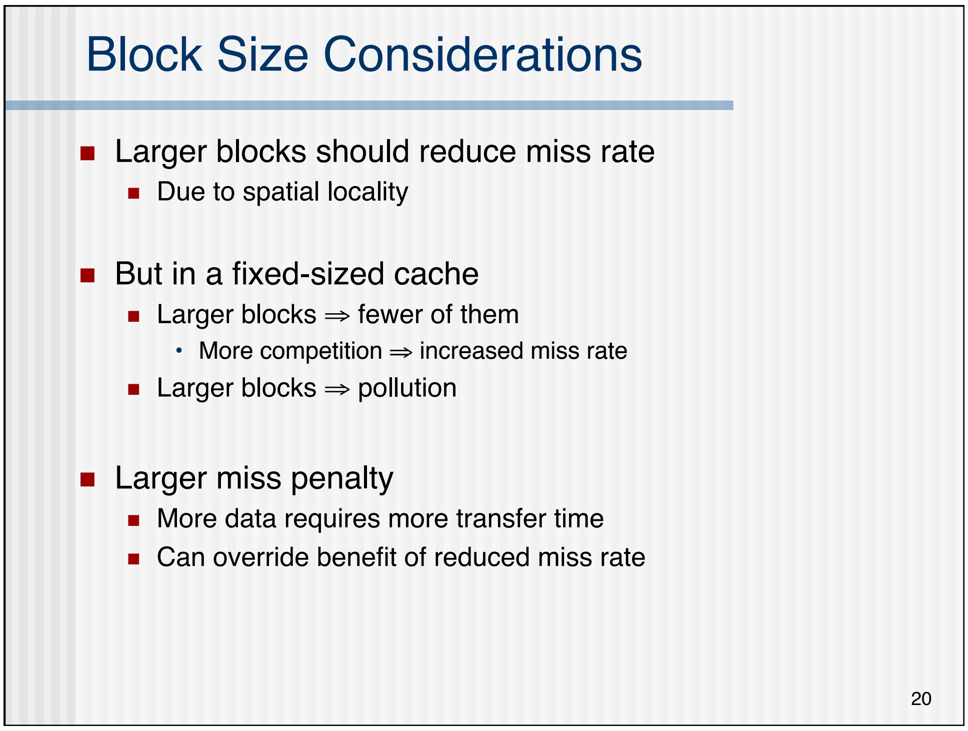


Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

19

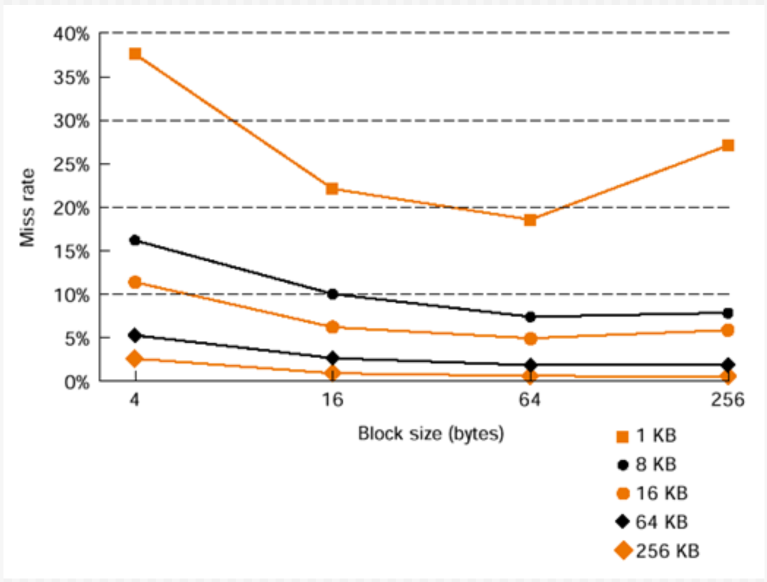


Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks ⇒ fewer of them
    - More competition ⇒ increased miss rate
  - Larger blocks ⇒ pollution
- Larger miss penalty
  - More data requires more transfer time
  - Can override benefit of reduced miss rate

20

Miss Rate vs Block Size



Cache Misses

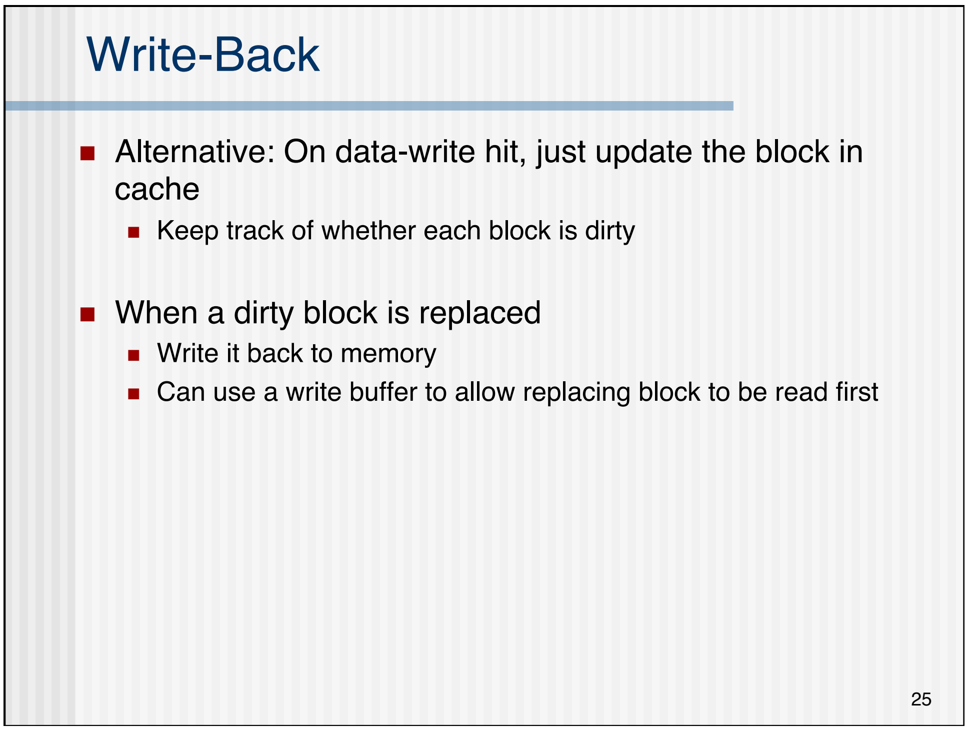
- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

Writing to Cache

- Reading from Cache is relatively simple
  - Reading alone doesn’t introduce real dependencies
    - Input dependency
- Writing to Cache is more tricky
  - Cache and main memory are no longer in sync
  - Must ensure the correct value is read after a write

Write-Through

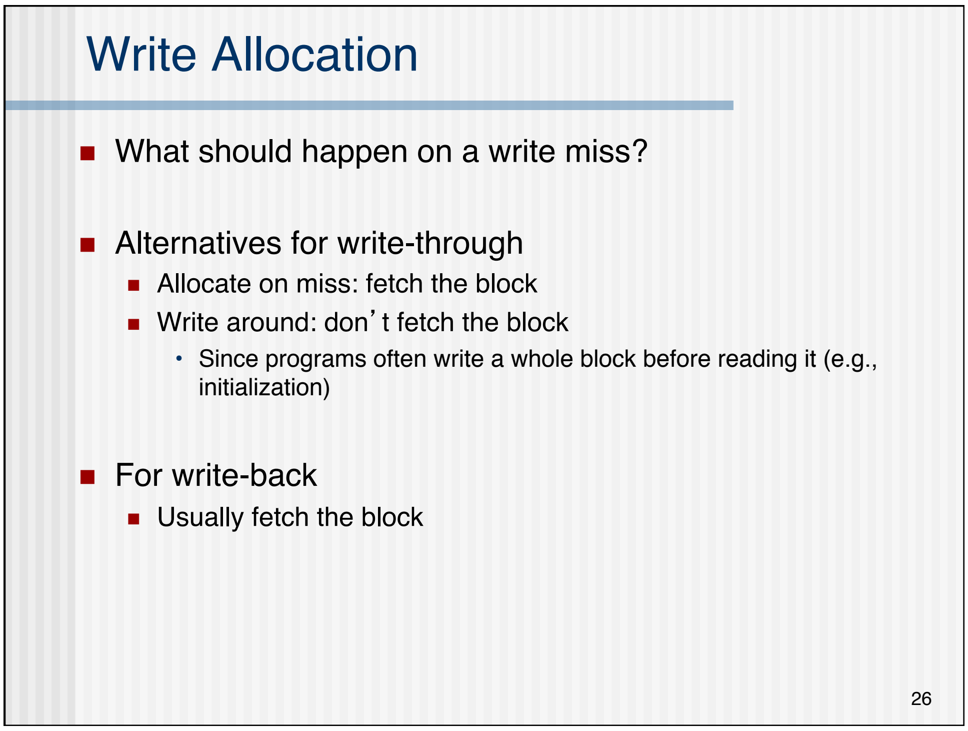
- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = $1 + 0.1\times100 = 11$
- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full



Write-Back

- Alternative: On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first

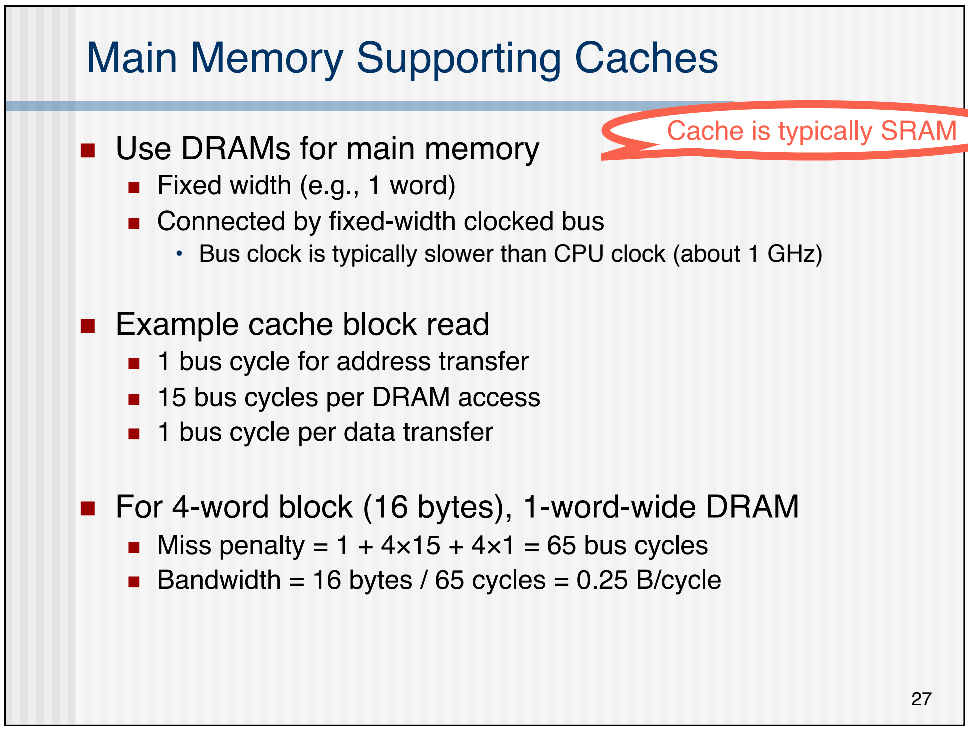
25



Write Allocation

- What should happen on a write miss?
- Alternatives for write-through
  - Allocate on miss: fetch the block
  - Write around: don’t fetch the block
    - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
  - Usually fetch the block

26

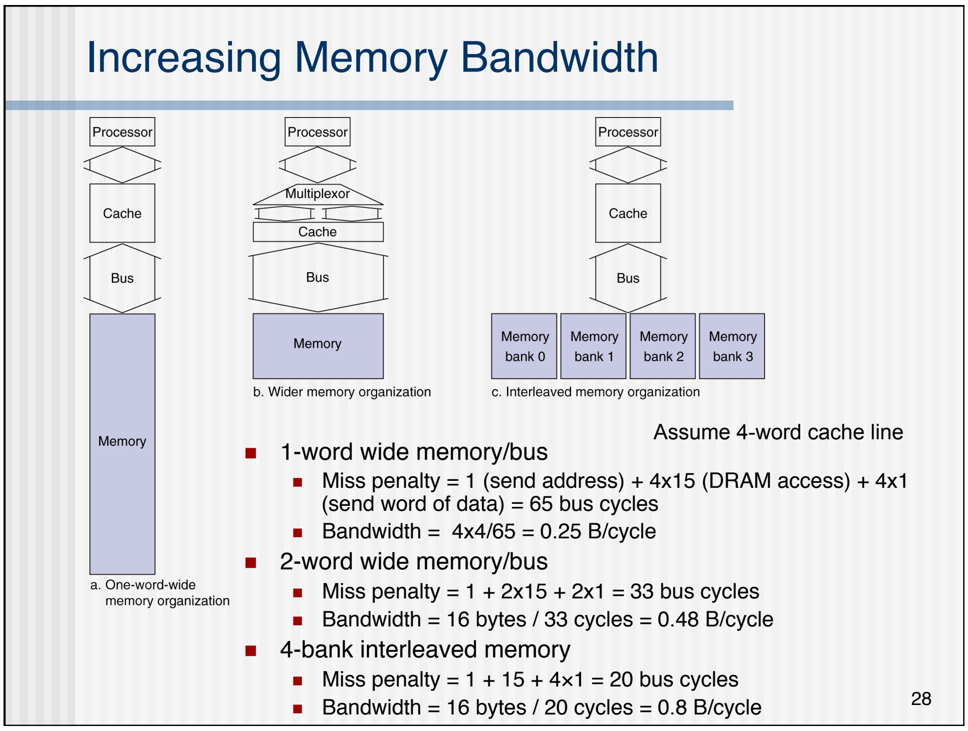


Main Memory Supporting Caches

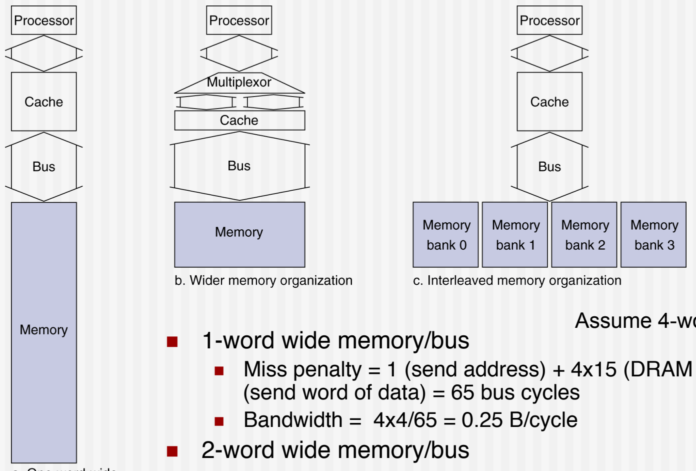
Cache is typically SRAM

- Use DRAMs for main memory
  - Fixed width (e.g., 1 word)
  - Connected by fixed-width clocked bus
    - Bus clock is typically slower than CPU clock (about 1 GHz)
- Example cache block read
  - 1 bus cycle for address transfer
  - 15 bus cycles per DRAM access
  - 1 bus cycle per data transfer
- For 4-word block (16 bytes), 1-word-wide DRAM
  - Miss penalty = 1 + 4×15 + 4×1 = 65 bus cycles
  - Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle

27



Increasing Memory Bandwidth



a. One-word-wide memory organization

b. Wider memory organization

c. Interleaved memory organization

Assume 4-word cache line

- 1-word wide memory/bus
  - Miss penalty = 1 (send address) + 4x15 (DRAM access) + 4x1 (send word of data) = 65 bus cycles
  - Bandwidth = 4x4/65 = 0.25 B/cycle
- 2-word wide memory/bus
  - Miss penalty = 1 + 2x15 + 2x1 = 33 bus cycles
  - Bandwidth = 16 bytes / 33 cycles = 0.48 B/cycle
- 4-bank interleaved memory
  - Miss penalty = 1 + 15 + 4x1 = 20 bus cycles
  - Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle

28

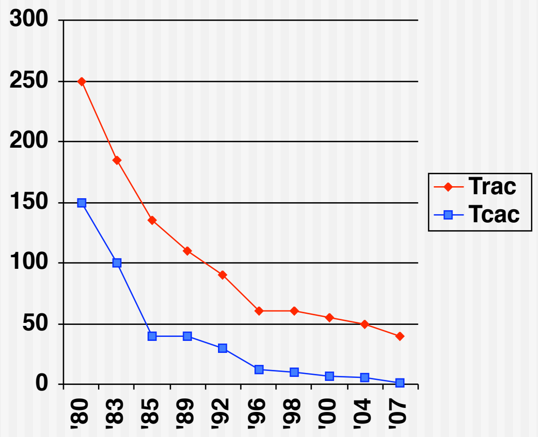
Advanced DRAM Organization

- Bits in a DRAM are organized as a rectangular array
  - DRAM accesses an entire row
  - Burst mode: supply successive words from a row with reduced latency
- Double data rate (DDR) DRAM
  - Transfer on rising and falling clock edges
- Quad data rate (QDR) DRAM
  - Separate DDR inputs and outputs

29

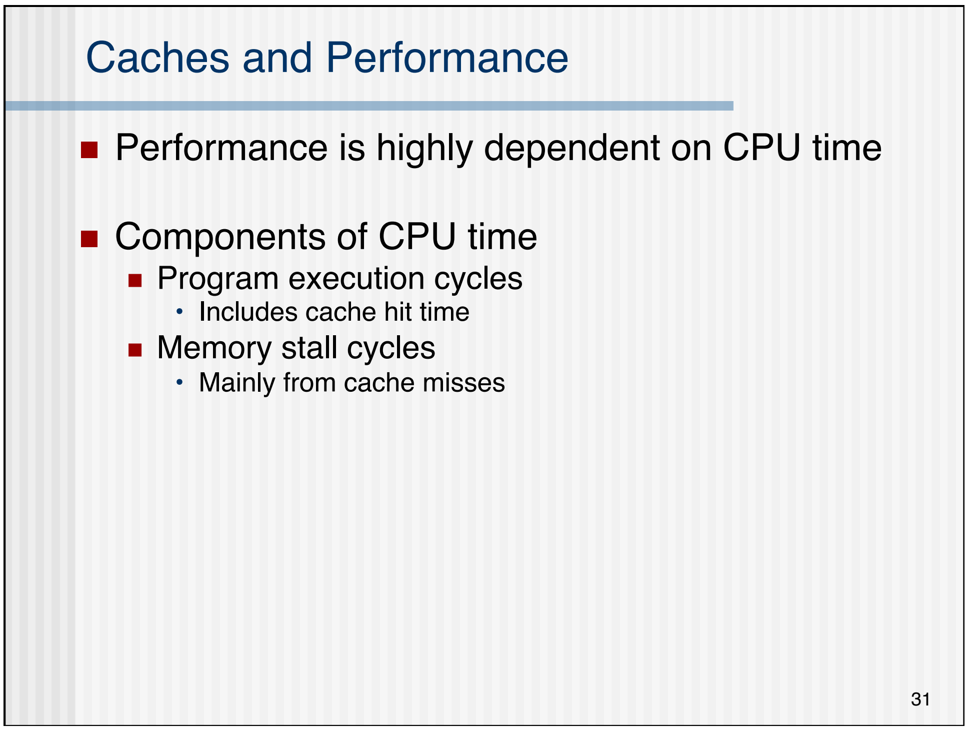
DRAM Generations

Year	Capacity	$/GB
1980	64Kbit	$1500000
1983	256Kbit	$500000
1985	1Mbit	$200000
1989	4Mbit	$50000
1992	16Mbit	$15000
1996	64Mbit	$10000
1998	128Mbit	$4000
2000	256Mbit	$1000
2004	512Mbit	$250
2007	1Gbit	$50



Factor 6 improvement compared to two orders of magnitude in clock rate

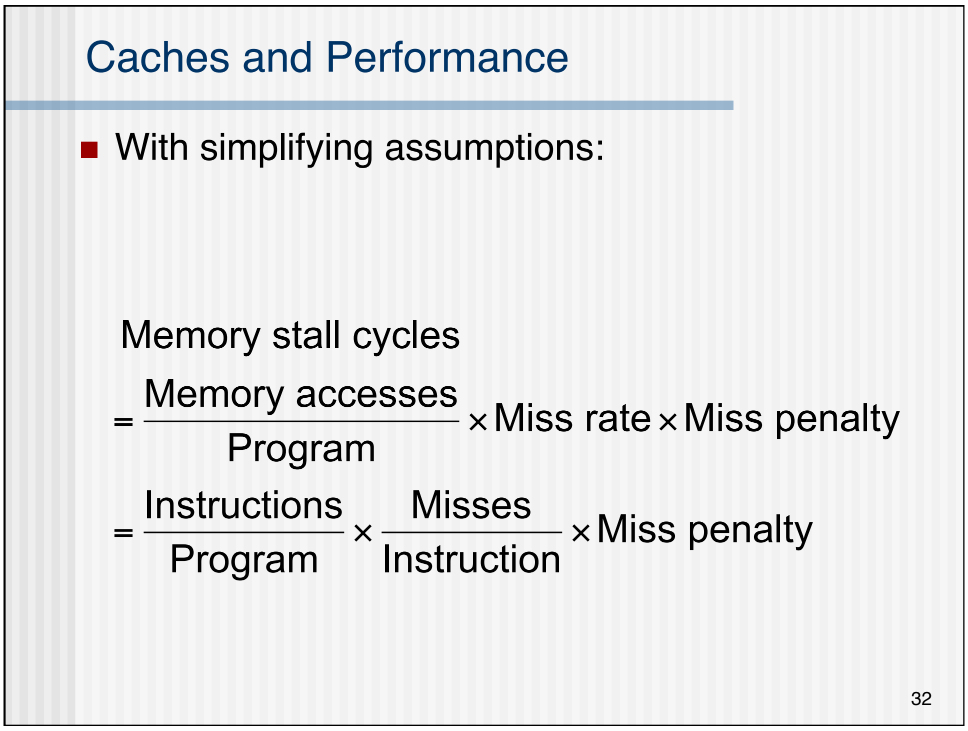
30



Caches and Performance

- Performance is highly dependent on CPU time
- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses

31

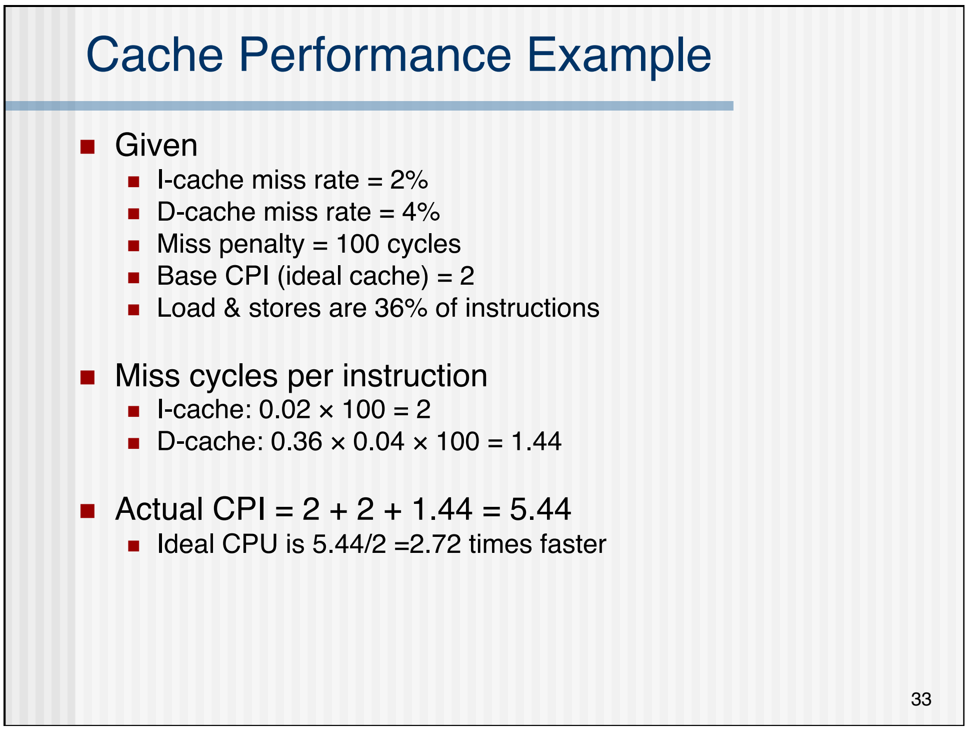


Caches and Performance

- With simplifying assumptions:

$$\begin{aligned}&\text{Memory stall cycles}\\&=\frac{\text{Memory accesses}}{\text{Program}}\times\text{Miss rate}\times\text{Miss penalty}\\&=\frac{\text{Instructions}}{\text{Program}}\times\frac{\text{Misses}}{\text{Instruction}}\times\text{Miss penalty}\end{aligned}$$

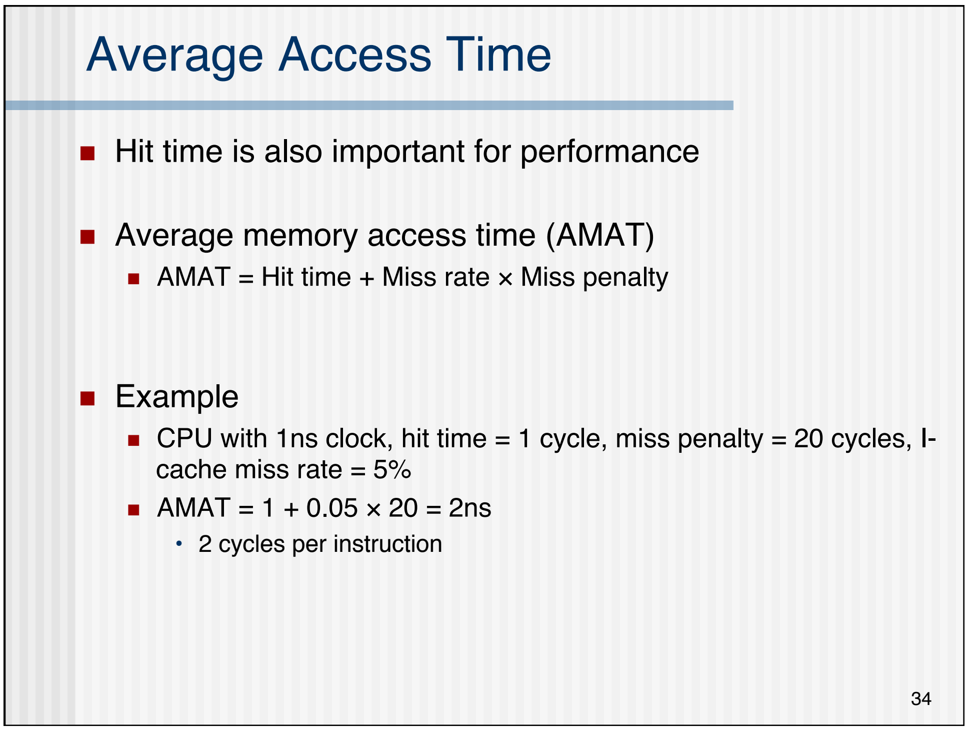
32



Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- Miss cycles per instruction
  - I-cache: 0.02 × 100 = 2
  - D-cache: 0.36 × 0.04 × 100 = 1.44
- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPU is 5.44/2 =2.72 times faster

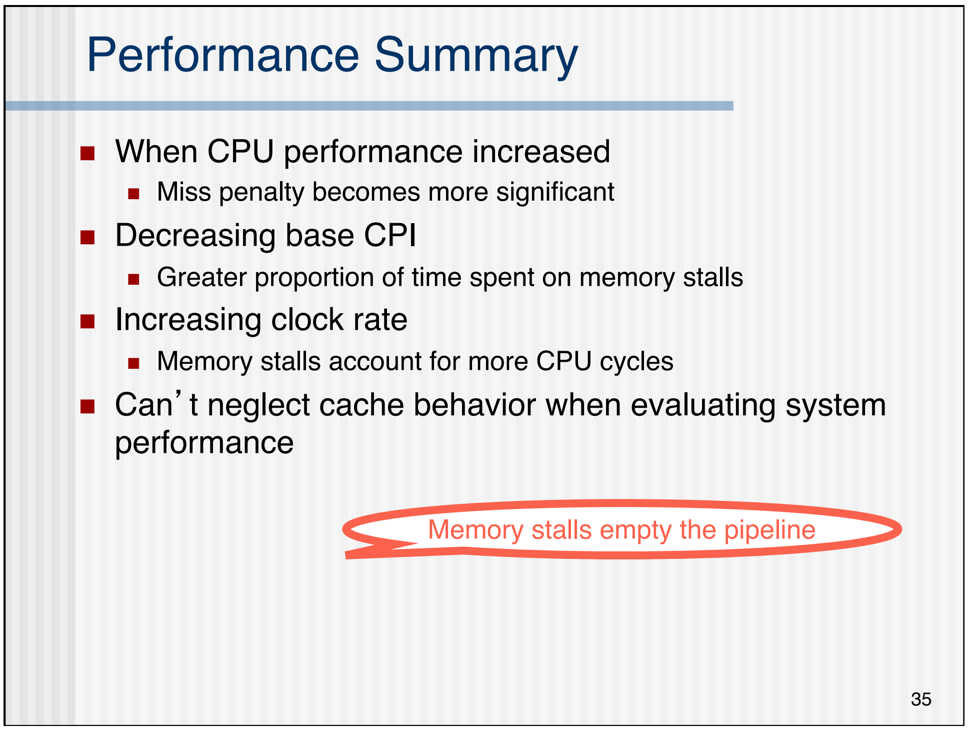
33



Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction

34

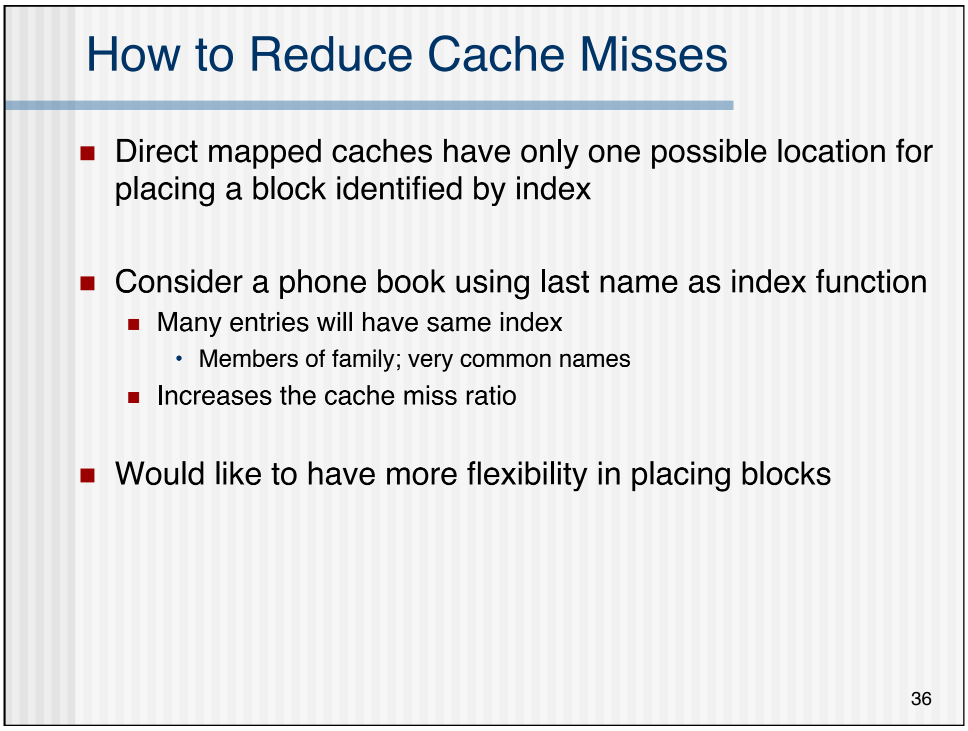


Performance Summary

- When CPU performance increased
  - Miss penalty becomes more significant
- Decreasing base CPI
  - Greater proportion of time spent on memory stalls
- Increasing clock rate
  - Memory stalls account for more CPU cycles
- Can’t neglect cache behavior when evaluating system performance

Memory stalls empty the pipeline

35



How to Reduce Cache Misses

- Direct mapped caches have only one possible location for placing a block identified by index
- Consider a phone book using last name as index function
  - Many entries will have same index
    - Members of family; very common names
  - Increases the cache miss ratio
- Would like to have more flexibility in placing blocks

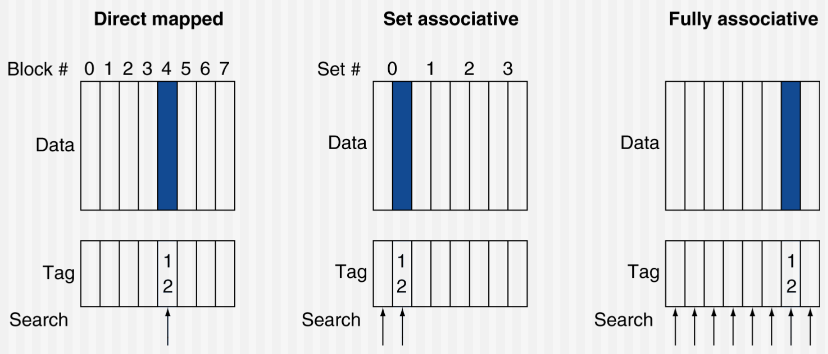
36

Associative Caches

- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

37

Associative Cache Example

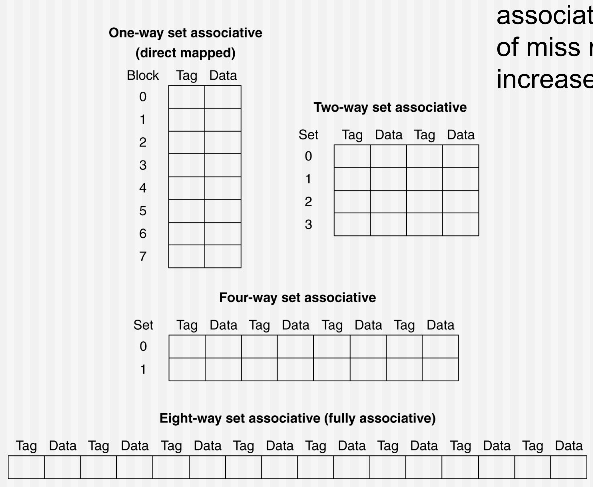


38

Spectrum of Associativity

- For a cache with 8 entries

With increasing associativity decrease of miss rate but increase of hit time


One-way set associative
(direct mapped)
Block Tag Data
0
1
2
3
4
5
6
7
Two-way set associative
Set Tag Data Tag Data
0
1
2
3
Four-way set associative
Set Tag Data Tag Data Tag Data Tag Data
0
1
Eight-way set associative (fully associative)
Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data

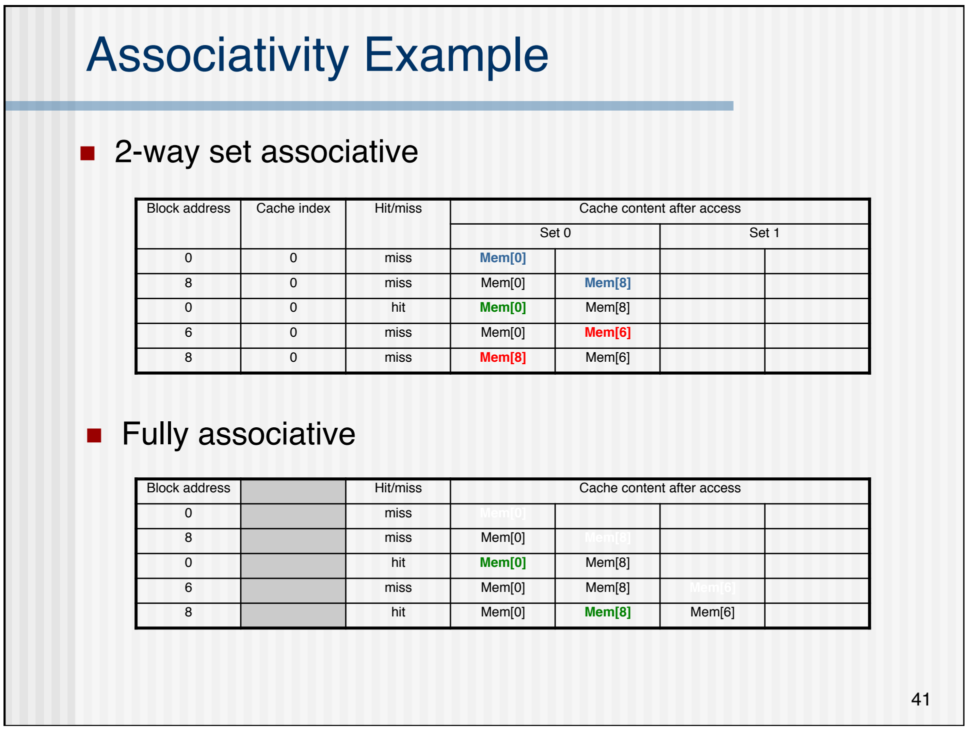
39

Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8
- Direct mapped

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	**Mem[0]**			
8	0	miss	**Mem[8]**			
0	0	miss	**Mem[0]**			
6	2	miss	Mem[0]		**Mem[6]**	
8	0	miss	**Mem[8]**		Mem[6]	

40



Associativity Example

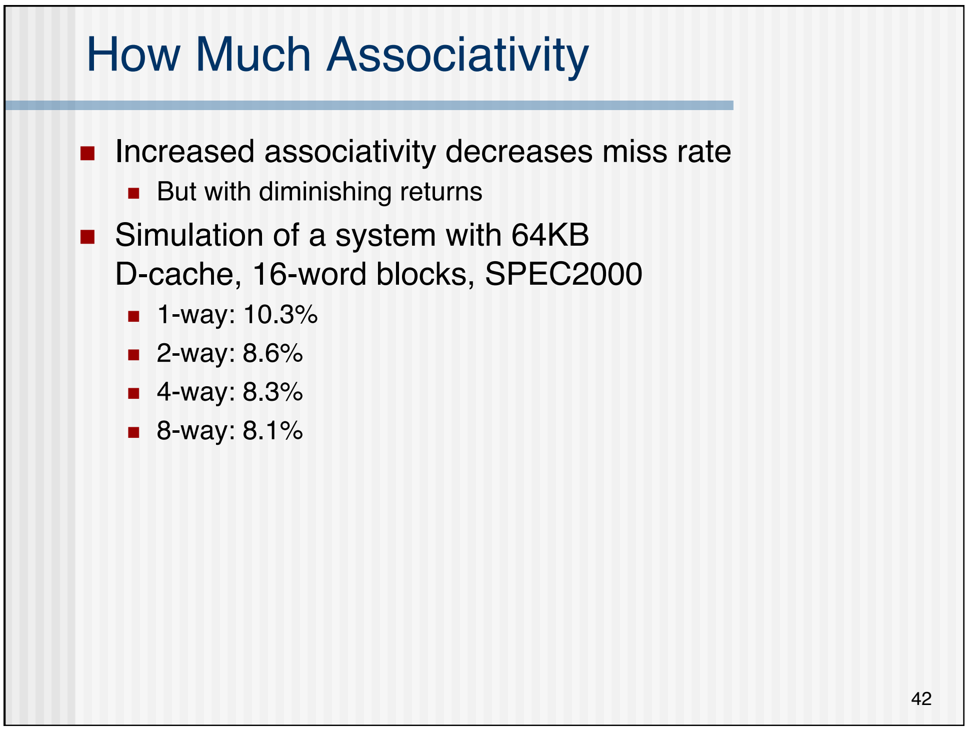
- 2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	**Mem[0]**			
8	0	miss	Mem[0]	**Mem[8]**		
0	0	hit	**Mem[0]**	Mem[8]		
6	0	miss	Mem[0]	**Mem[6]**		
8	0	miss	**Mem[8]**	Mem[6]		

- Fully associative

Block address		Hit/miss	Cache content after access			
0		miss	**Mem[0]**			
8		miss	Mem[0]	**Mem[8]**		
0		hit	**Mem[0]**	Mem[8]		
6		miss	Mem[0]	Mem[8]	**Mem[6]**	
8		hit	Mem[0]	**Mem[8]**	Mem[6]	

41



How Much Associativity

- Increased associativity decreases miss rate
  - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

42

Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity

43

How to reduce cache misses

- Chose access patterns that exploit spatial locality
- Simple example (Fortran):

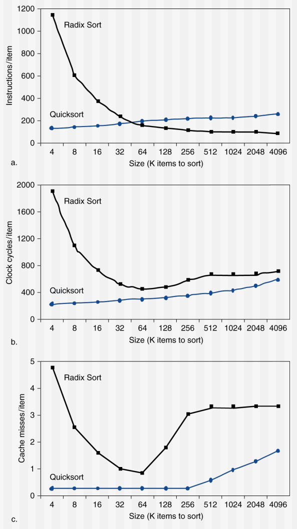
```
do i=1,n                    do j=1,m
  do j=1,m                    do i=1,n
    A(i,j)=A(i,j)+B(i,j)        A(i,j)=A(i,j)+B(i,j)
  end do                      end do
end do                      end do
```

- Remember Fortran is using column major memory layout (as opposed to row major in e.g. C)

44

Example: Sorting

- Radix Sort has an algorithmic advantage over quicksort
  - Lower number of instructions for large arrays
- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access

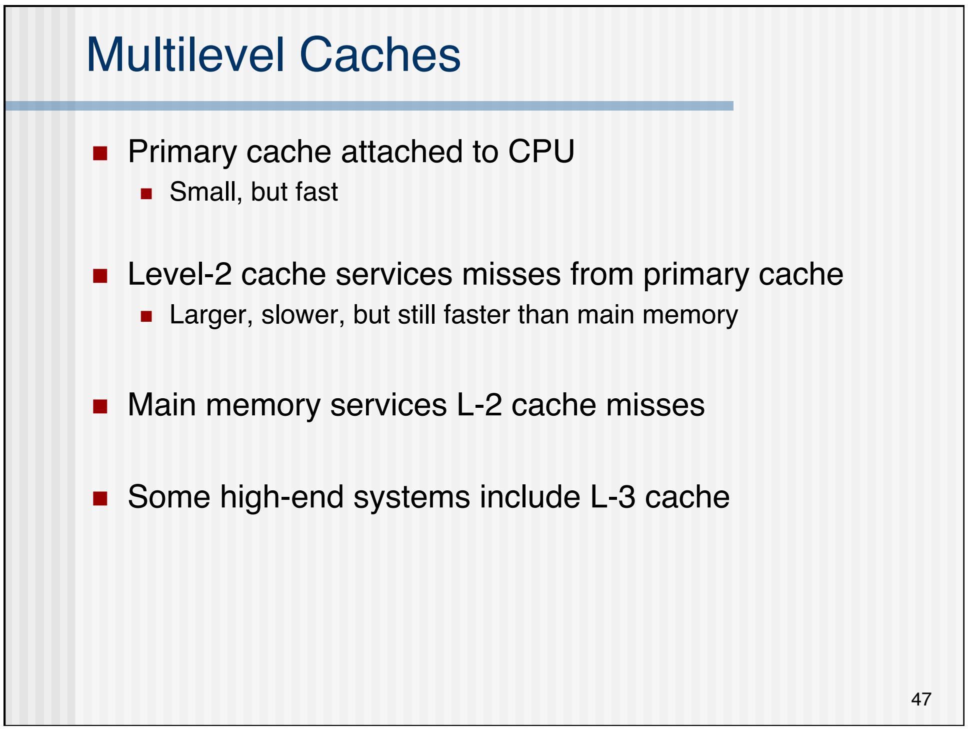


45

How to Improve Cache Performance

- DRAM performance has not kept pace with CPU performance
- Use additional caches (fast SRAM) close to CPU to improve performance
- Multilevel Caches - Cache Hierarchies

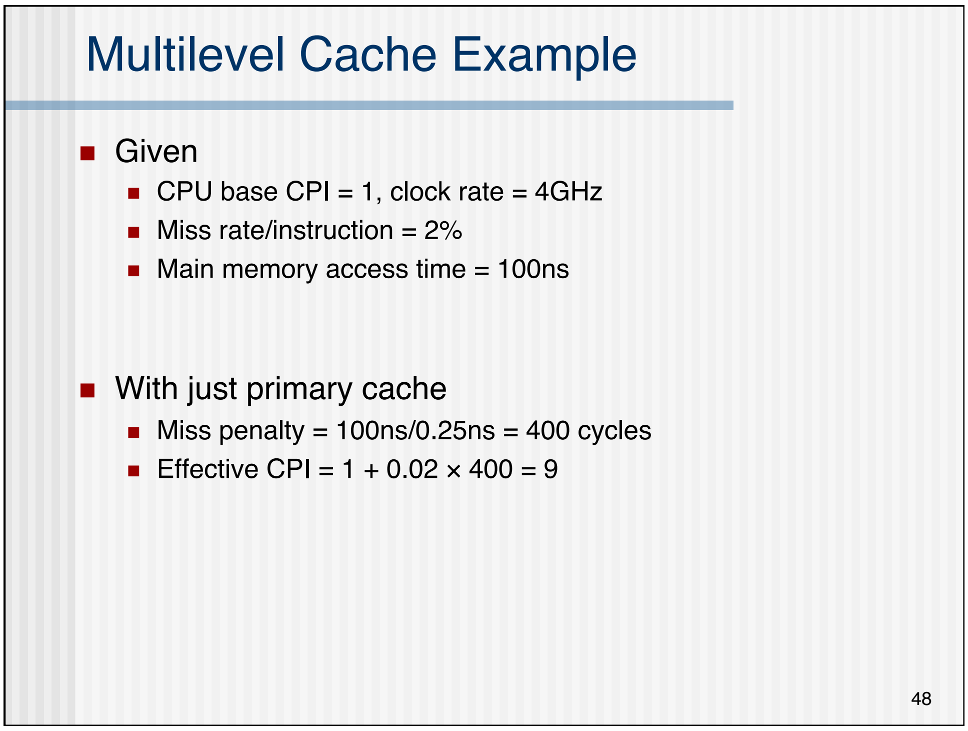
46



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

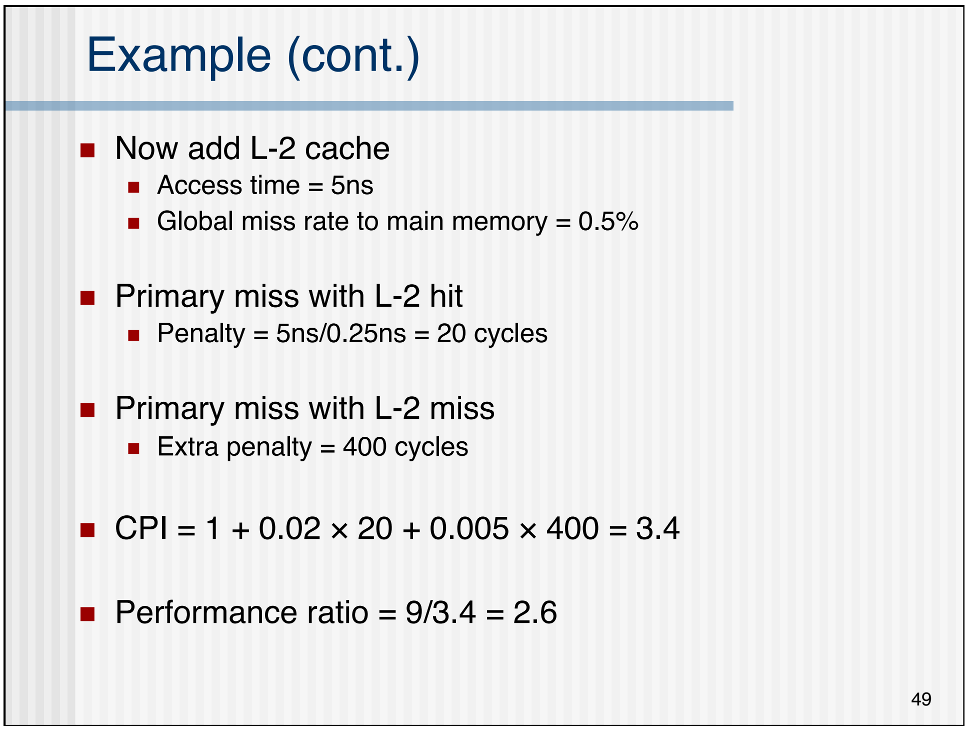
47



Multilevel Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9

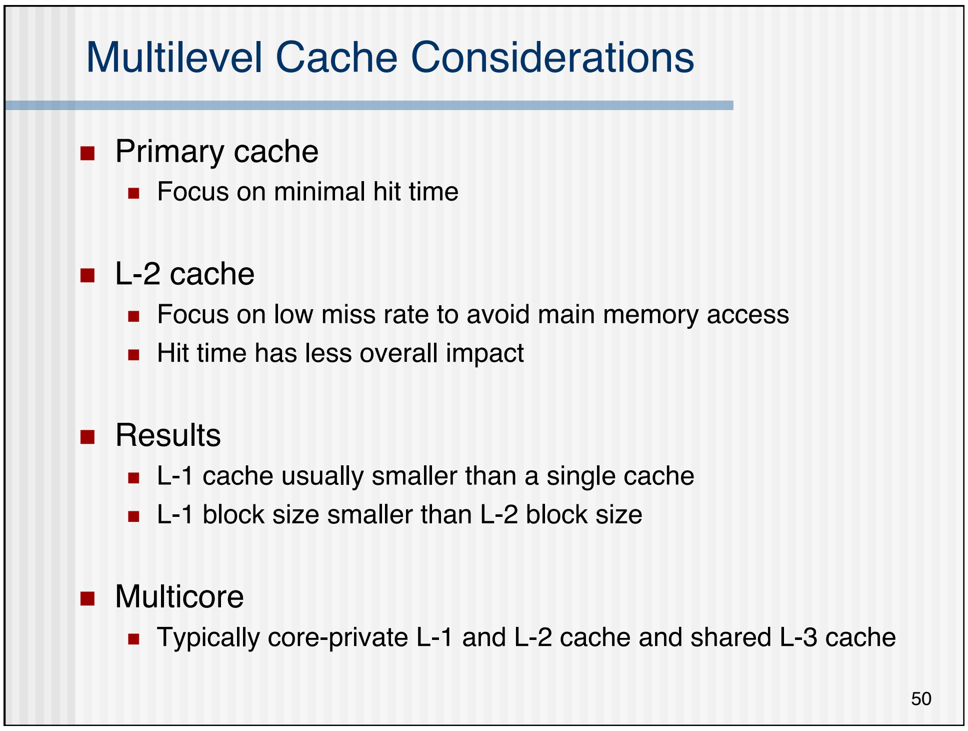
48



Example (cont.)

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 400 cycles
- CPI = $1 + 0.02 \times 20 + 0.005 \times 400 = 3.4$
- Performance ratio = 9/3.4 = 2.6

49



Multilevel Cache Considerations

- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size
- Multicore
  - Typically core-private L-1 and L-2 cache and shared L-3 cache

50

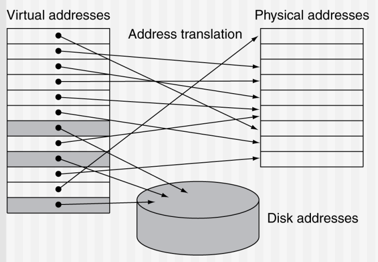
Virtual Memory

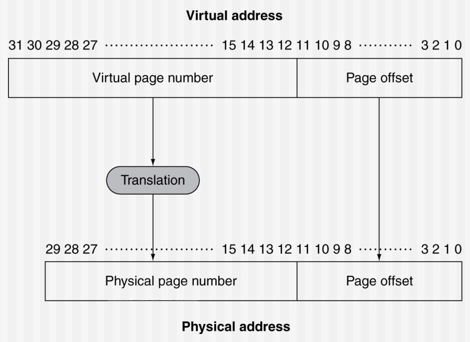
- Use main memory as a “cache” for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)
- Programs share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs
- CPU and OS translate virtual addresses to physical addresses
  - VM “block” is called a page
  - VM translation “miss” is called a page fault

51

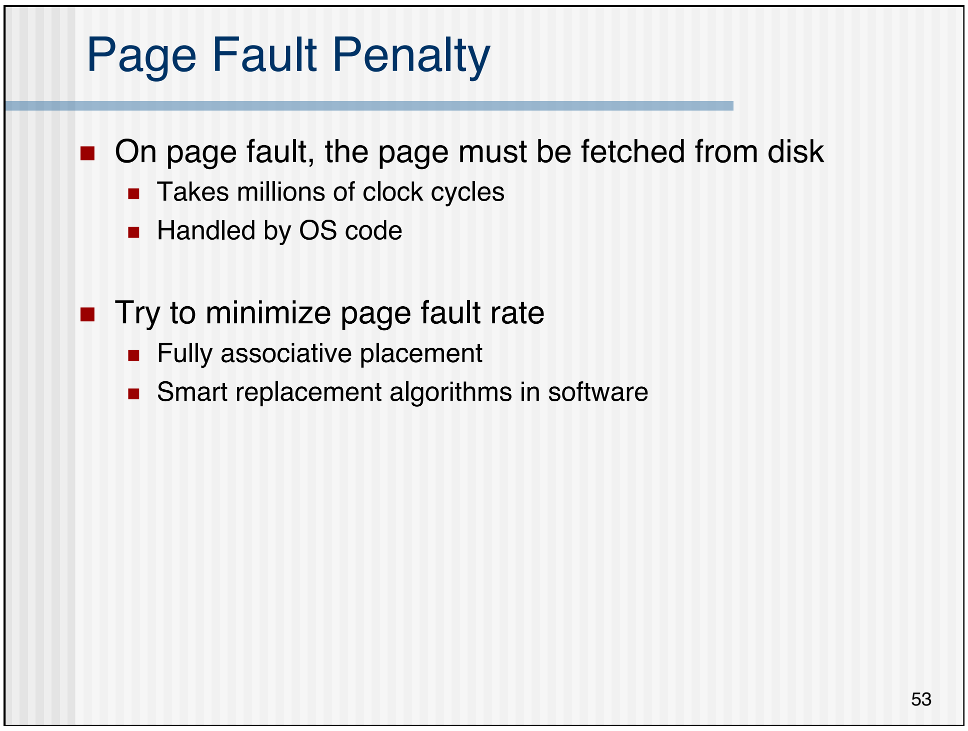
Address Translation

- Fixed-size pages (e.g., 4K)





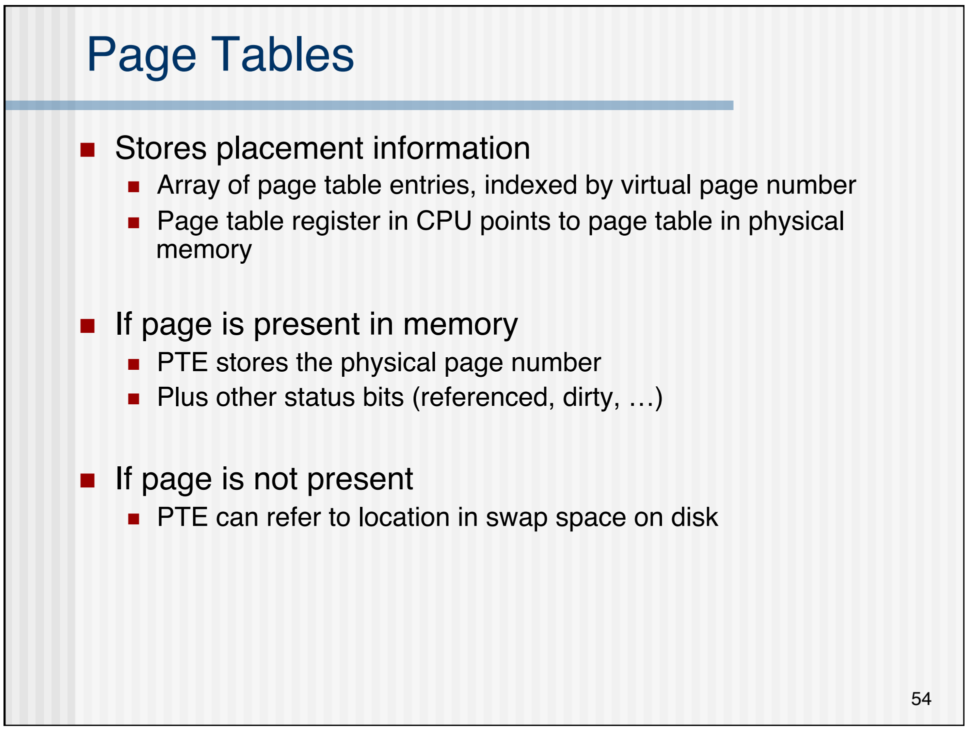
52



Page Fault Penalty

- On page fault, the page must be fetched from disk
  - Takes millions of clock cycles
  - Handled by OS code
- Try to minimize page fault rate
  - Fully associative placement
  - Smart replacement algorithms in software

53

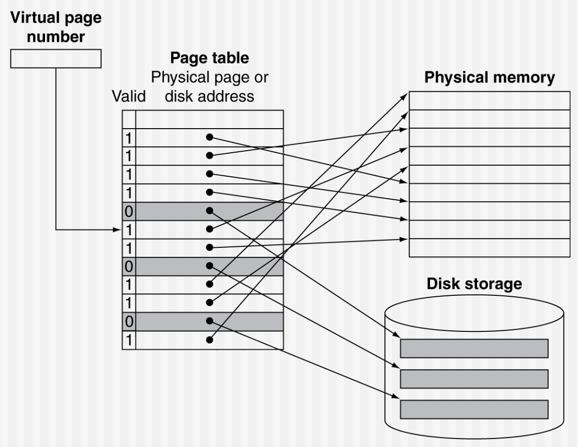


Page Tables

- Stores placement information
  - Array of page table entries, indexed by virtual page number
  - Page table register in CPU points to page table in physical memory
- If page is present in memory
  - PTE stores the physical page number
  - Plus other status bits (referenced, dirty, …)
- If page is not present
  - PTE can refer to location in swap space on disk

54

Mapping Pages to Storage



55

Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently
- Disk writes take millions of cycles
  - Block at once, not individual locations
  - Write through is impractical
  - Use write-back
  - Dirty bit in PTE set when page is written

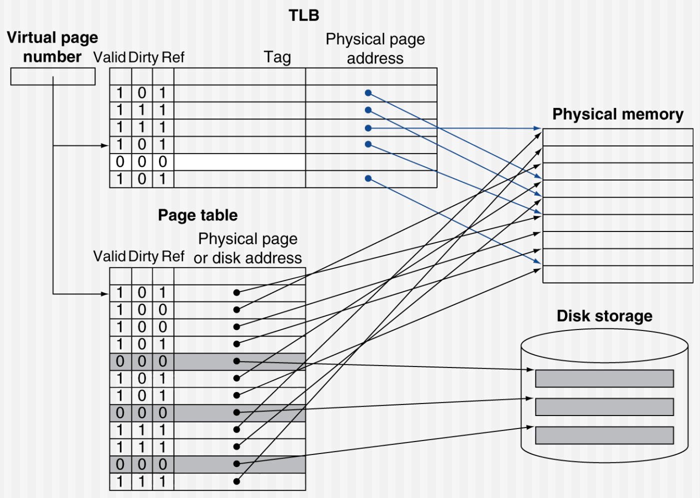
56

Fast Translation Using a TLB

- Address translation would appear to require extra memory references
  - One to access the PTE
  - Then the actual memory access
- But access to page tables has good locality
  - So use a fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB)
  - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
  - Misses could be handled by hardware or software

57

Fast Translation Using a TLB



58

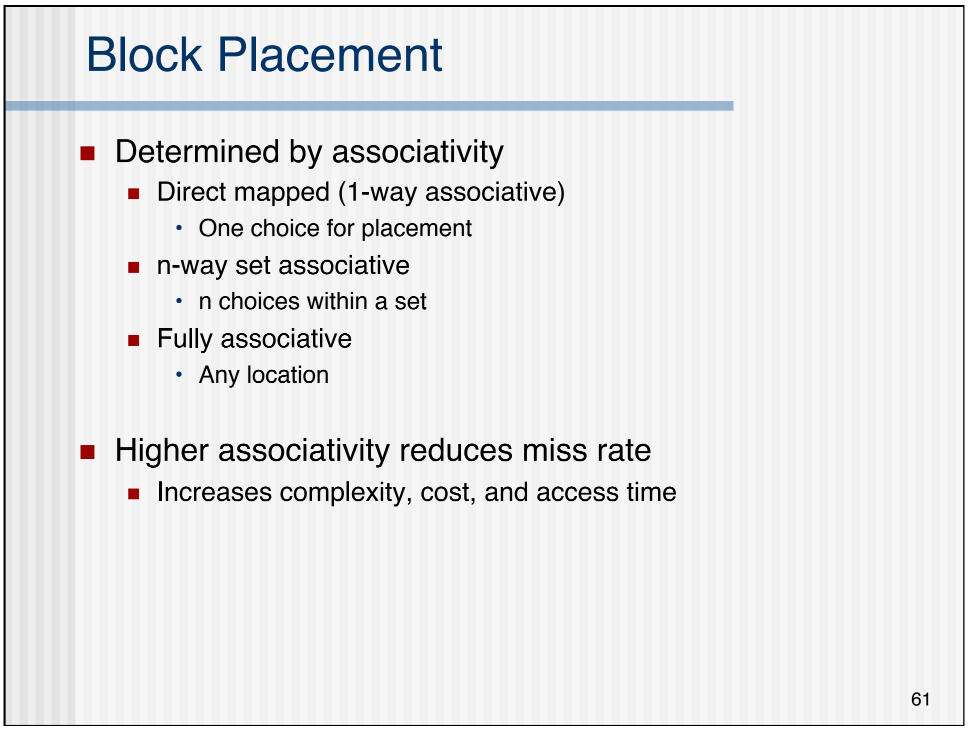
Memory Hierarchy Summary

59

The Memory Hierarchy

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of caching
- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy

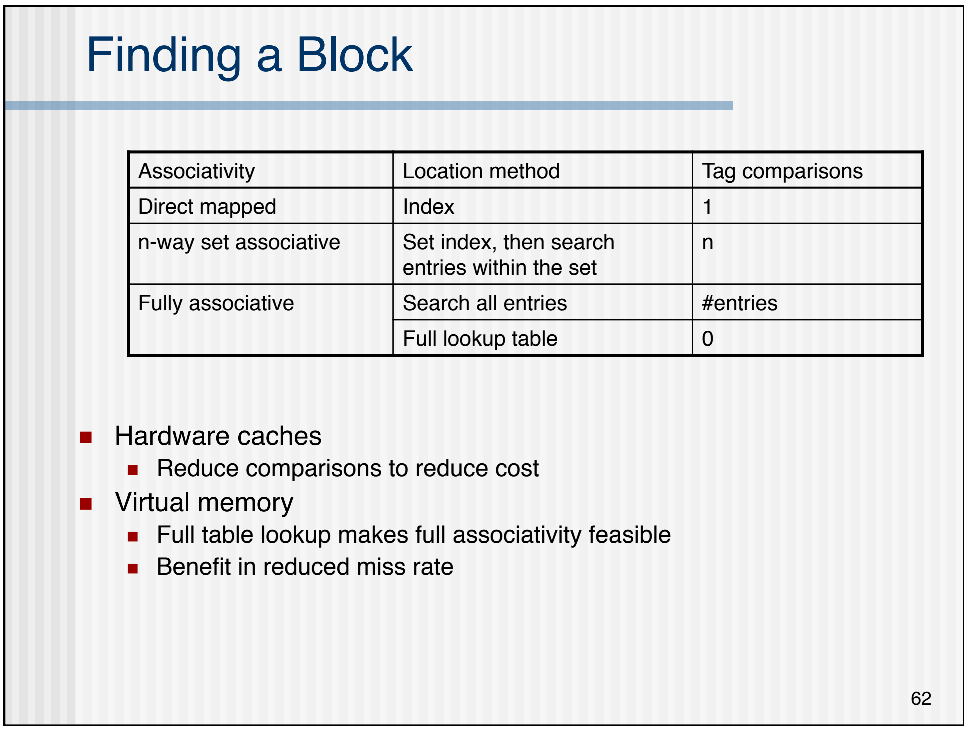
60



Block Placement

- Determined by associativity
  - Direct mapped (1-way associative)
    - One choice for placement
  - n-way set associative
    - n choices within a set
  - Fully associative
    - Any location
- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time

61

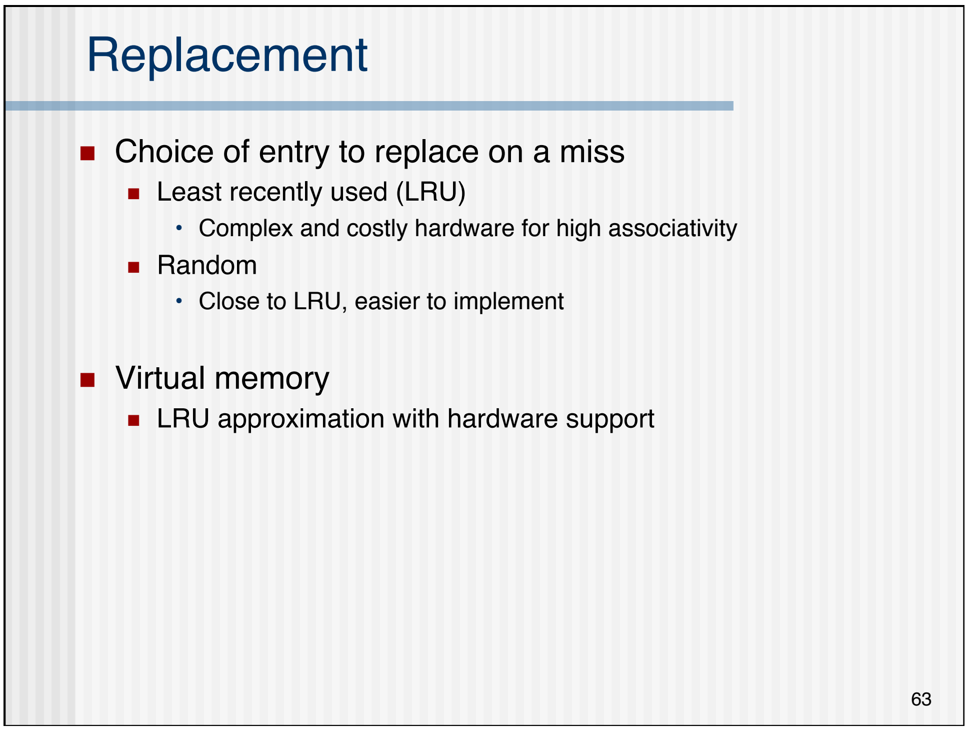


Finding a Block

Associativity	Location method	Tag comparisons
Direct mapped	Index	1
n-way set associative	Set index, then search entries within the set	n
Fully associative	Search all entries	#entries
	Full lookup table	0

- Hardware caches
  - Reduce comparisons to reduce cost
- Virtual memory
  - Full table lookup makes full associativity feasible
  - Benefit in reduced miss rate

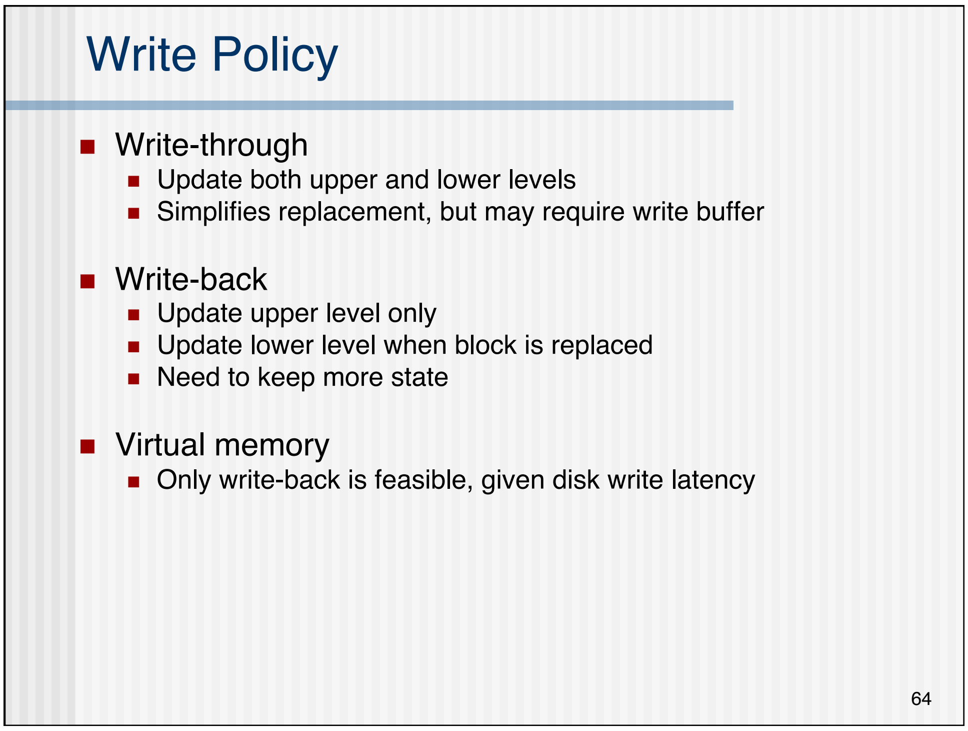
62



Replacement

- Choice of entry to replace on a miss
  - Least recently used (LRU)
    - Complex and costly hardware for high associativity
  - Random
    - Close to LRU, easier to implement
- Virtual memory
  - LRU approximation with hardware support

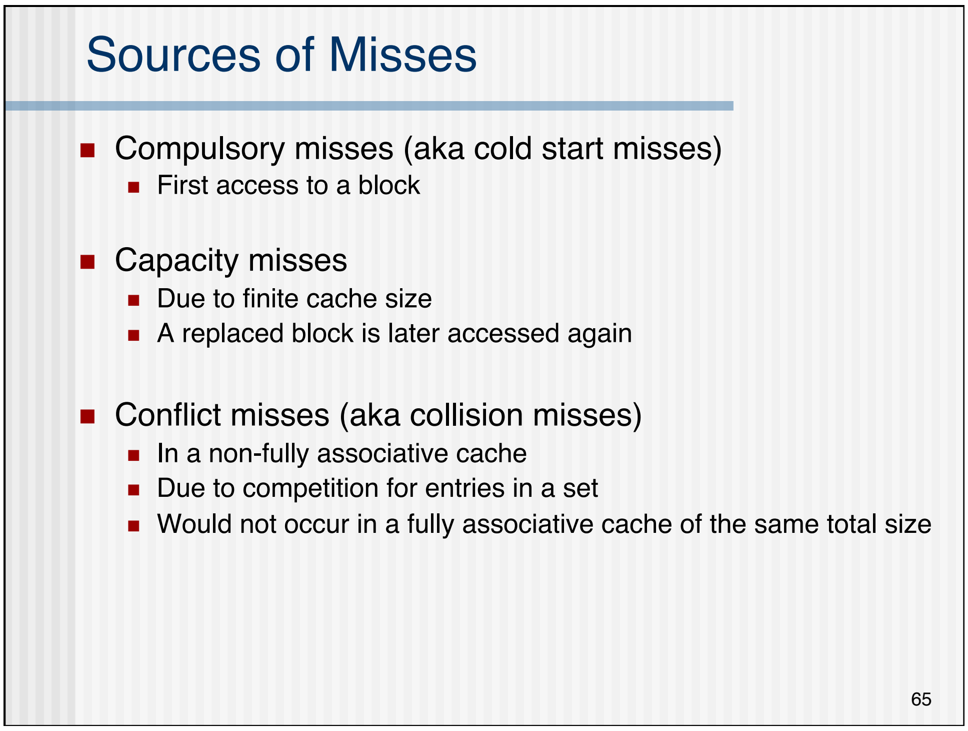
63



Write Policy

- Write-through
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer
- Write-back
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep more state
- Virtual memory
  - Only write-back is feasible, given disk write latency

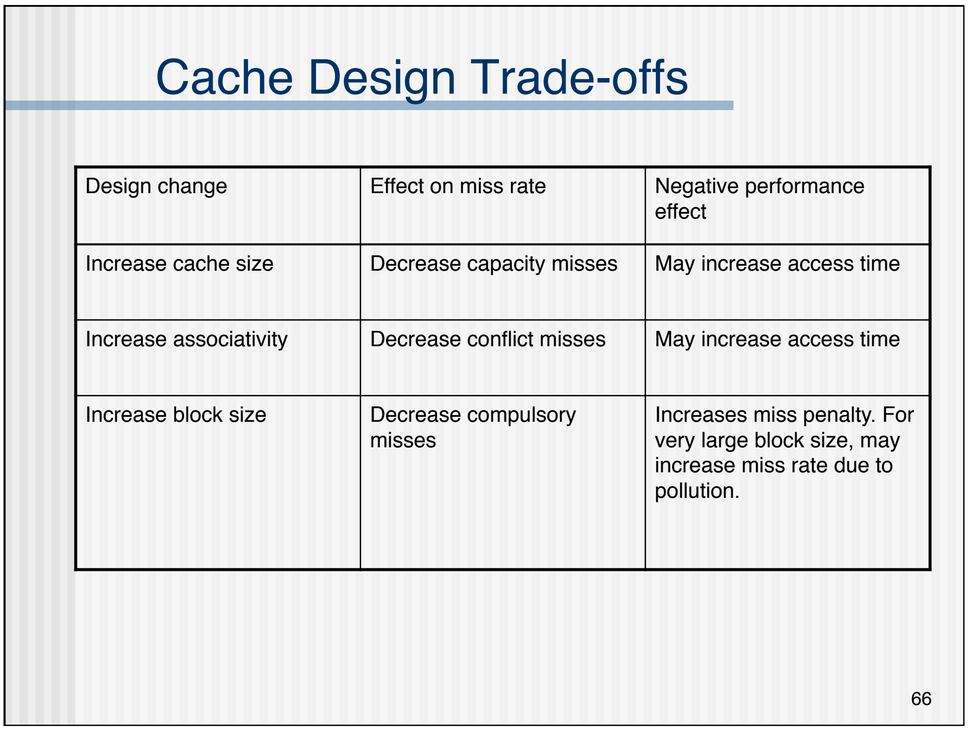
64



Sources of Misses

- Compulsory misses (aka cold start misses)
  - First access to a block
- Capacity misses
  - Due to finite cache size
  - A replaced block is later accessed again
- Conflict misses (aka collision misses)
  - In a non-fully associative cache
  - Due to competition for entries in a set
  - Would not occur in a fully associative cache of the same total size

65



Cache Design Trade-offs

Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.

66

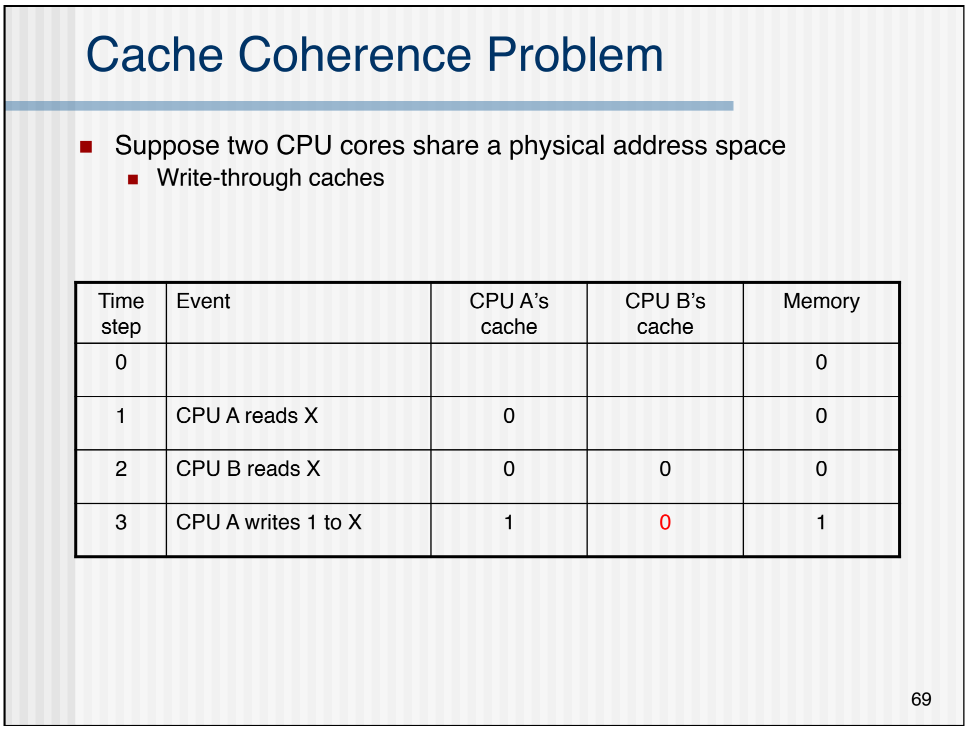
Caches in SMPs/Multicores

67

Key Issues

- Different CPUs/cores share same main memory (and often L3 cache) but have separate, local L1/L2 caches
- Depending on caching strategy and without extra precaution different CPUs might see different values
- Cache coherence tries to overcome this problem

68

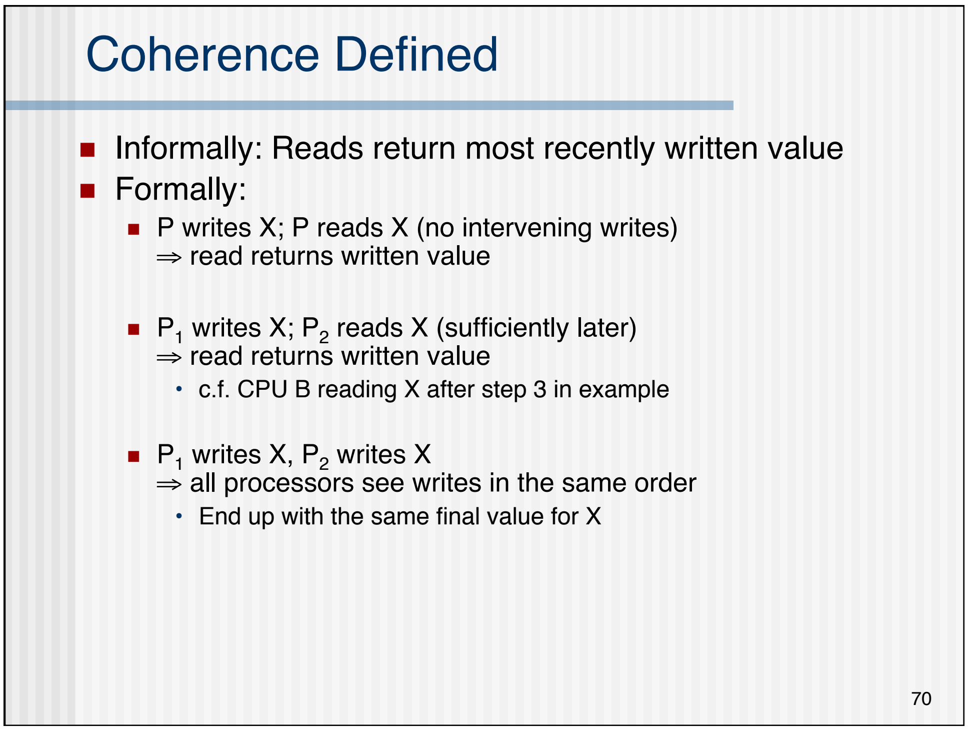


Cache Coherence Problem

- Suppose two CPU cores share a physical address space
  - Write-through caches

Time step	Event	CPU A's cache	CPU B's cache	Memory
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A writes 1 to X	1	0	1

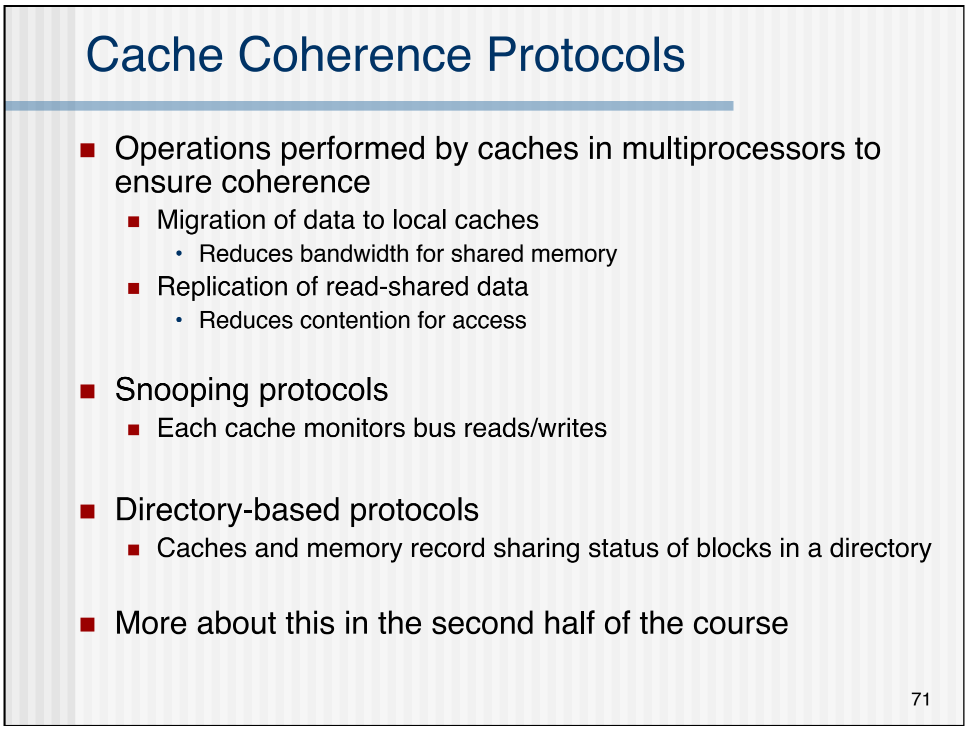
69



Coherence Defined

- Informally: Reads return most recently written value
- Formally:
  - P writes X; P reads X (no intervening writes)
    $\Rightarrow$ read returns written value
  - P_1 writes X; P_2 reads X (sufficiently later)
    $\Rightarrow$ read returns written value
    - c.f. CPU B reading X after step 3 in example
  - P_1 writes X, P_2 writes X
    $\Rightarrow$ all processors see writes in the same order
    - End up with the same final value for X

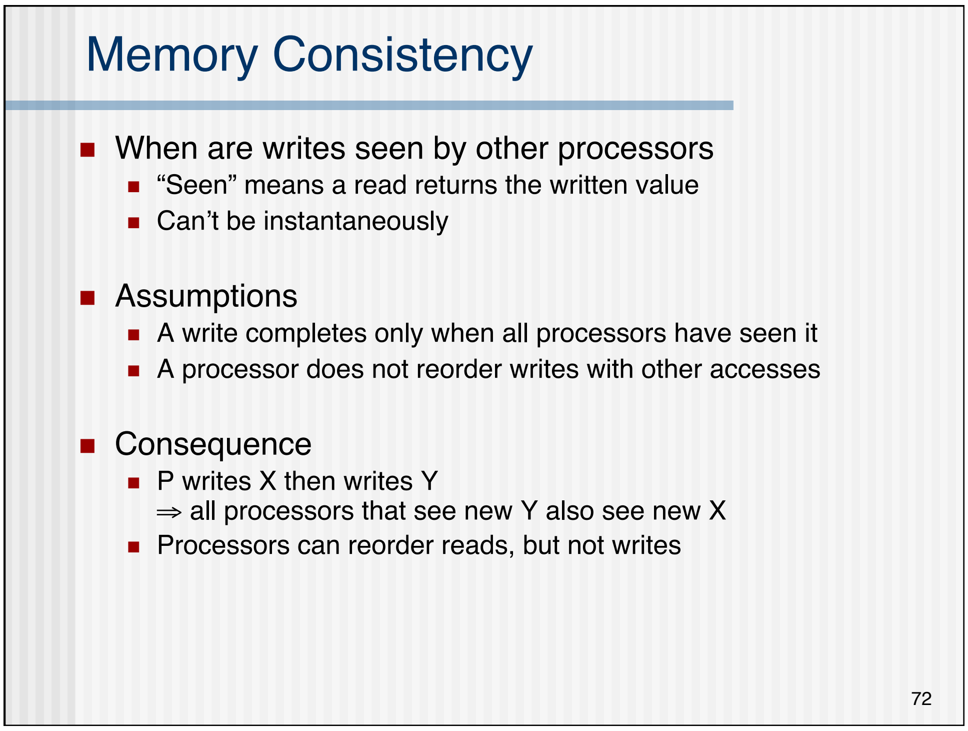
70



Cache Coherence Protocols

- Operations performed by caches in multiprocessors to ensure coherence
  - Migration of data to local caches
    - Reduces bandwidth for shared memory
  - Replication of read-shared data
    - Reduces contention for access
- Snooping protocols
  - Each cache monitors bus reads/writes
- Directory-based protocols
  - Caches and memory record sharing status of blocks in a directory
- More about this in the second half of the course

71



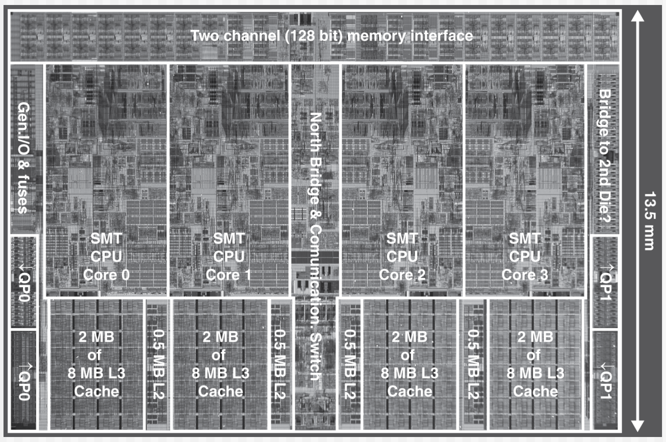
Memory Consistency

- When are writes seen by other processors
  - “Seen” means a read returns the written value
  - Can’t be instantaneously
- Assumptions
  - A write completes only when all processors have seen it
  - A processor does not reorder writes with other accesses
- Consequence
  - P writes X then writes Y
    $\Rightarrow$ all processors that see new Y also see new X
  - Processors can reorder reads, but not writes

72

Multilevel On-Chip Caches

Intel Nehalem 4-core processor



Per core: 32KB L1 I-cache, 32KB L1 D-cache, 512KB L2 cache
Shared: 8 MB L3 cache

73

3-Level Cache Organization

	Intel Nehalem	AMD Opteron X4
L1 caches (per core)	L1 I-cache: 32KB, 64-byte blocks, 4-way, approx LRU replacement, hit time n/a L1 D-cache: 32KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	L1 I-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, hit time 3 cycles L1 D-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, write-back/ allocate, hit time 9 cycles
L2 unified cache (per core)	256KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	512KB, 64-byte blocks, 16-way, approx LRU replacement, write-back/allocate, hit time n/a
L3 unified cache (shared)	8MB, 64-byte blocks, 16-way, replacement n/a, write-back/allocate, hit time n/a	2MB, 64-byte blocks, 32-way, replace block shared by fewest cores, write-back/allocate, hit time 32 cycles

n/a: data not available

74

Pitfalls

- Ignoring memory system effects when writing or generating code
  - Example: iterating over rows vs. columns of arrays
  - Large strides result in poor locality
- In multiprocessor with shared L2 or L3 cache
  - Less associativity than cores results in conflict misses
  - More cores ⇒ need to increase associativity

75

Summary and Outlook

- Fast memories are small, large memories are slow
  - We really want fast, large memories
  - Caching gives this illusion
- Principle of locality
  - Programs use a small part of their memory space frequently
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk
- Memory system design is critical for multiprocessors
  - Much more about this in 2nd half of course
- Next: What to do with all the data - Input/Output

76

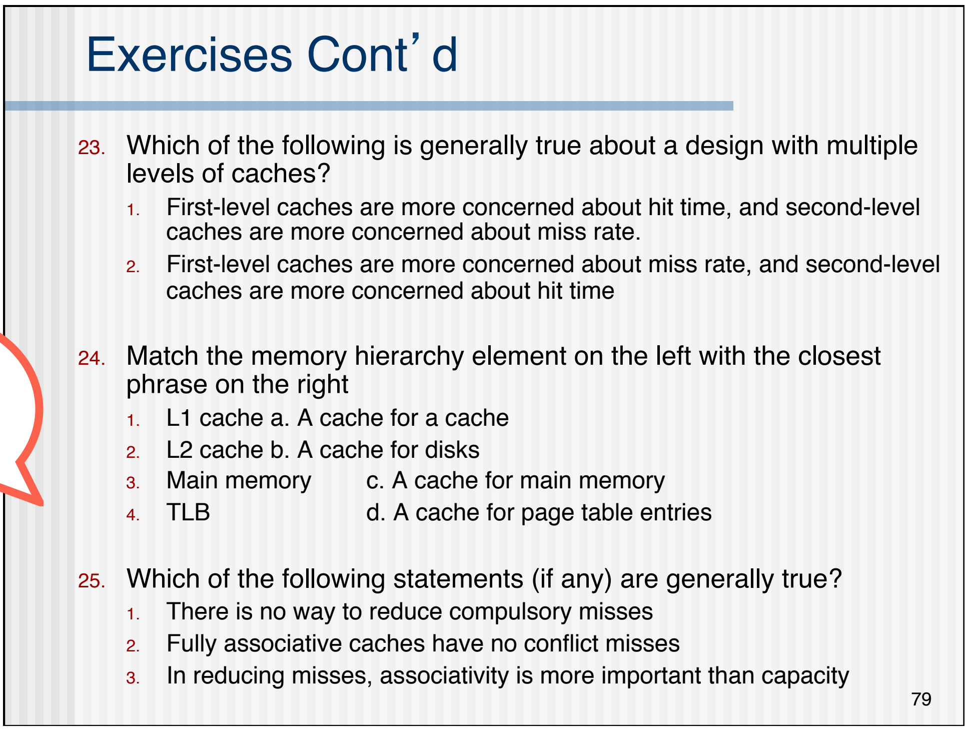
Further Readings

- **Computer Organization and Design - The Hardware/Software Interface,** 4th Edition, David A. Patterson and John L. Hennessy
  - Chapter 5

Exercises

21. Which of the following statements are generally true:
    1. Caches take advantage of temporal locality
    2. On a read, the value returned depends on which blocks are in the cache
    3. Most of the cost of the memory hierarchy is at the highest level
    4. Most of the capacity of the memory hierarchy is at the lowest level

22. Which of the following cache designer guidelines are generally valid? Justify
    1. The shorter the memory latency, the smaller the cache block
    2. The shorter the memory latency, the larger the cache block
    3. The higher the memory bandwidth, the smaller the cache block
    4. The higher the memory bandwidth, the larger the cache block



Exercises Cont' d

23. Which of the following is generally true about a design with multiple levels of caches?
    1. First-level caches are more concerned about hit time, and second-level caches are more concerned about miss rate.
    2. First-level caches are more concerned about miss rate, and second-level caches are more concerned about hit time

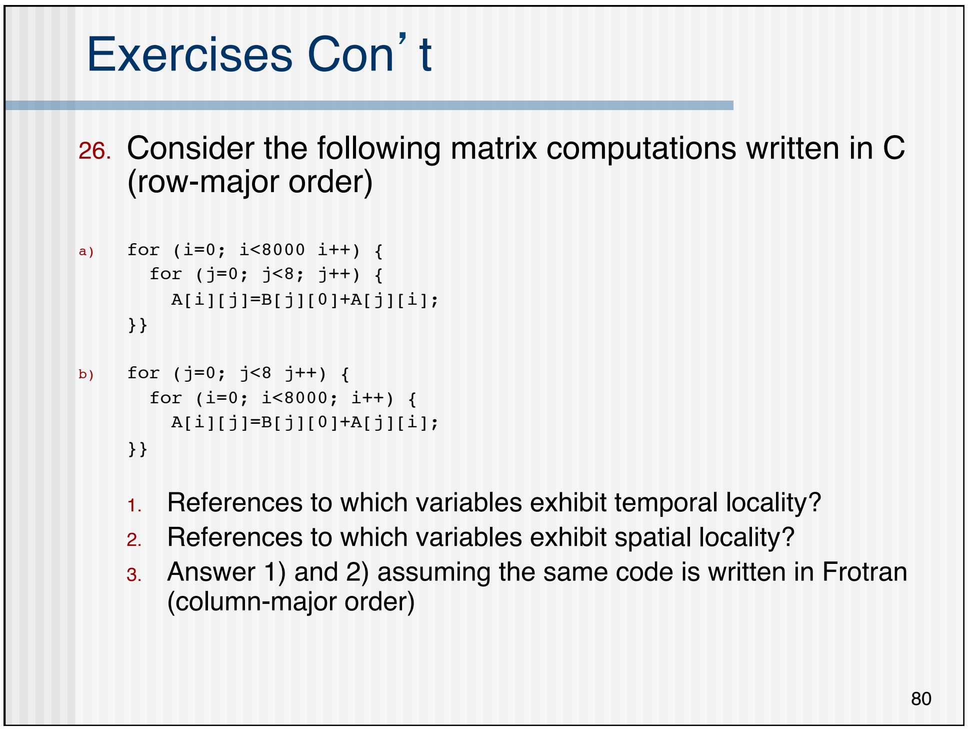
Translation Lookaside Buffer

24. Match the memory hierarchy element on the left with the closest phrase on the right

1.	L1 cache	a. A cache for a cache
2.	L2 cache	b. A cache for disks
3.	Main memory	c. A cache for main memory
4.	TLB	d. A cache for page table entries

25. Which of the following statements (if any) are generally true?
    1. There is no way to reduce compulsory misses
    2. Fully associative caches have no conflict misses
    3. In reducing misses, associativity is more important than capacity

79



Exercises Con' t

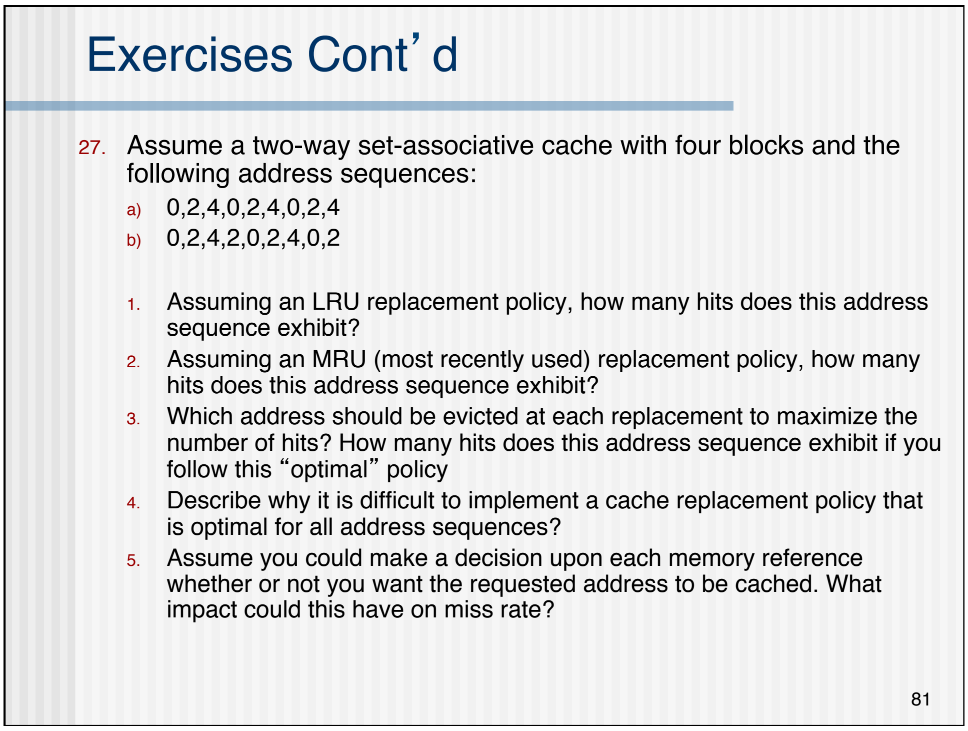
26. Consider the following matrix computations written in C (row-major order)

a)
```
for (i=0; i<8000 i++) {
  for (j=0; j<8; j++) {
    A[i][j]=B[j][0]+A[j][i];
}}
```

b)
```
for (j=0; j<8 j++) {
  for (i=0; i<8000; i++) {
    A[i][j]=B[j][0]+A[j][i];
}}
```

1. References to which variables exhibit temporal locality?
2. References to which variables exhibit spatial locality?
3. Answer 1) and 2) assuming the same code is written in Frotran (column-major order)

80



Exercises Cont’d

27. Assume a two-way set-associative cache with four blocks and the following address sequences:
    a) 0,2,4,0,2,4,0,2,4
    b) 0,2,4,2,0,2,4,0,2

    1. Assuming an LRU replacement policy, how many hits does this address sequence exhibit?
    2. Assuming an MRU (most recently used) replacement policy, how many hits does this address sequence exhibit?
    3. Which address should be evicted at each replacement to maximize the number of hits? How many hits does this address sequence exhibit if you follow this “optimal” policy
    4. Describe why it is difficult to implement a cache replacement policy that is optimal for all address sequences?
    5. Assume you could make a decision upon each memory reference whether or not you want the requested address to be cached. What impact could this have on miss rate?

81