Memory

COE 308

Computer Architecture

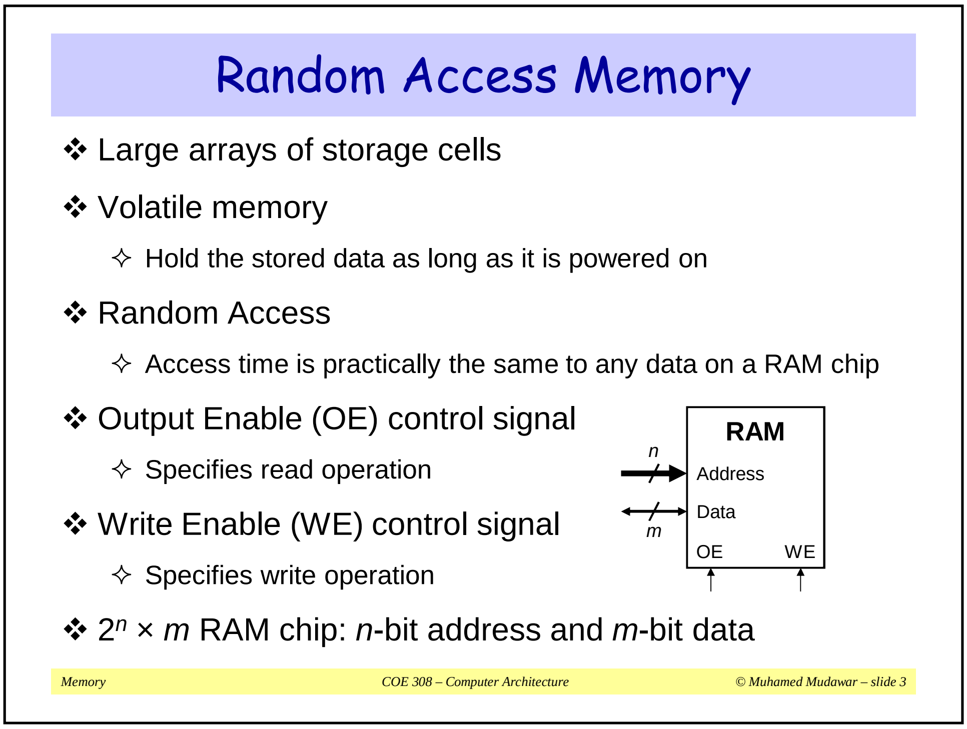
Prof. Muhamed Mudawar

Computer Engineering Department

King Fahd University of Petroleum and Minerals

Presentation Outline

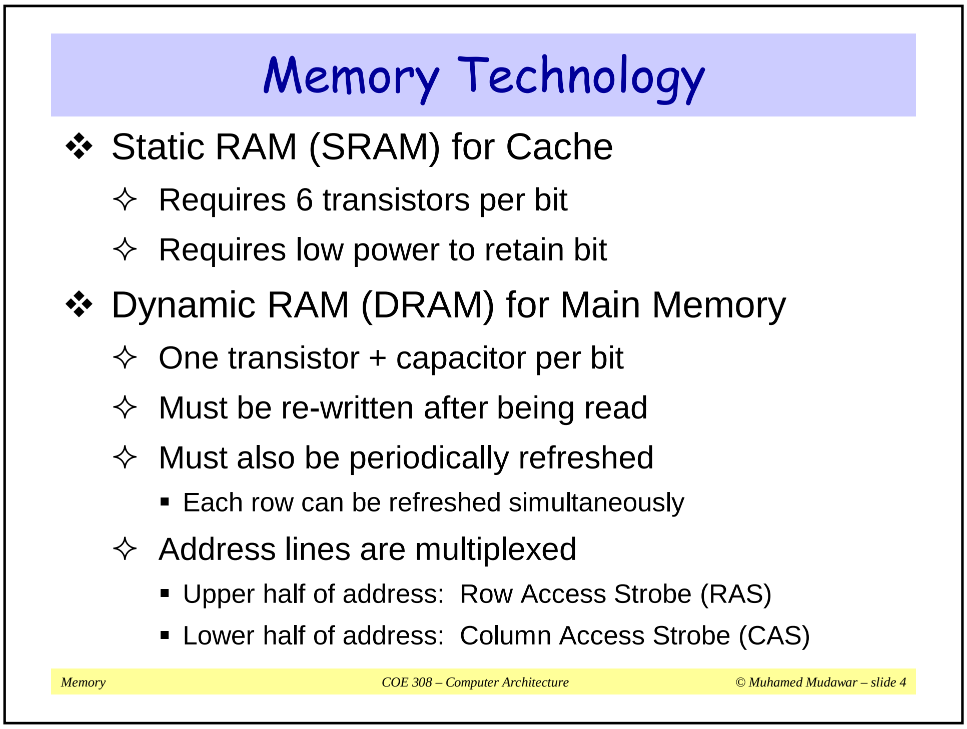
- ❖ Random Access Memory and its Structure
- ❖ Memory Hierarchy and the need for Cache Memory
- ❖ The Basics of Caches
- ❖ Cache Performance and Memory Stall Cycles
- ❖ Improving Cache Performance
- ❖ Multilevel Caches



Random Access Memory

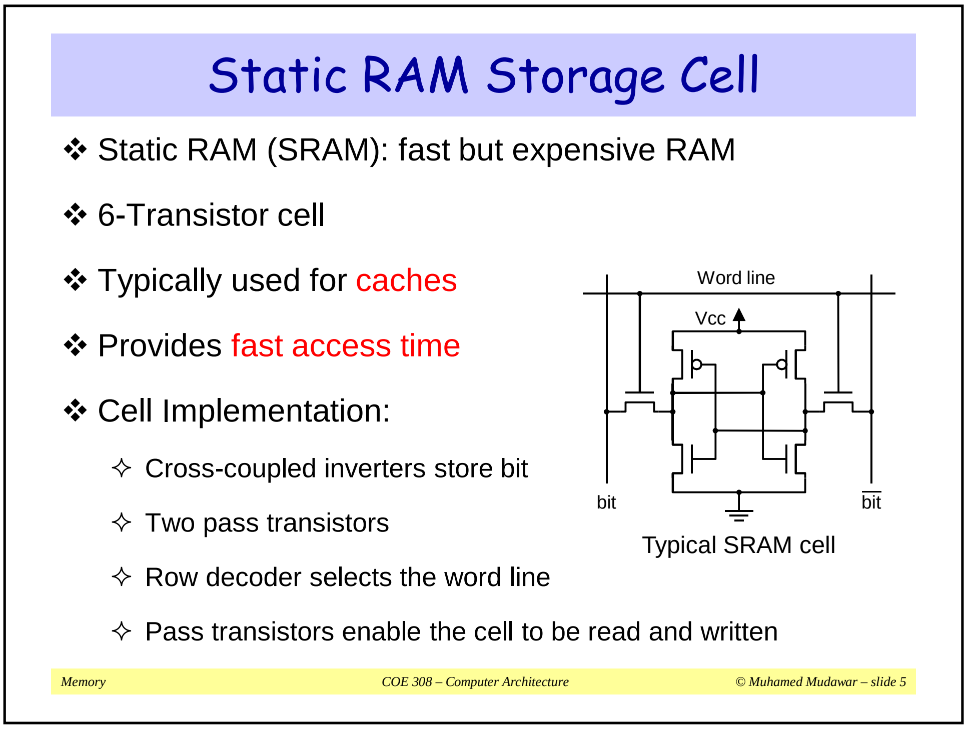
- ❖ Large arrays of storage cells
- ❖ Volatile memory
  - ✧ Hold the stored data as long as it is powered on
- ❖ Random Access
  - ✧ Access time is practically the same to any data on a RAM chip
- ❖ Output Enable (OE) control signal
  - ✧ Specifies read operation
- ❖ Write Enable (WE) control signal
  - ✧ Specifies write operation
- ❖ $2^n \times m$ RAM chip: n-bit address and m-bit data

RAM
n
Address
Data
m
OE
WE



Memory Technology

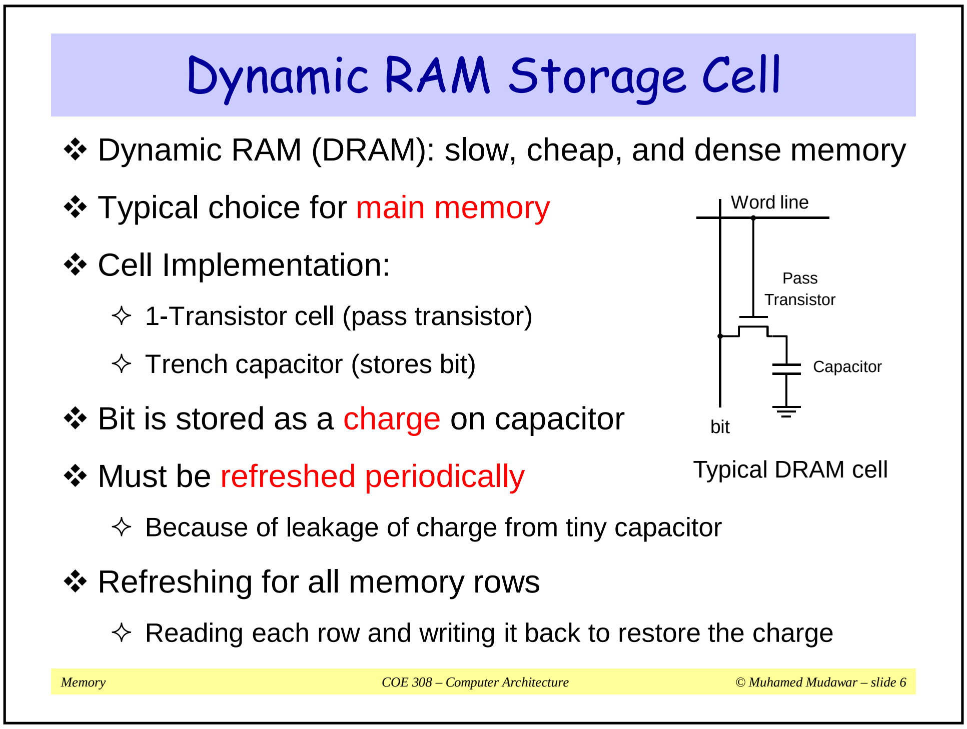
- ❖ Static RAM (SRAM) for Cache
  - ✧ Requires 6 transistors per bit
  - ✧ Requires low power to retain bit
- ❖ Dynamic RAM (DRAM) for Main Memory
  - ✧ One transistor + capacitor per bit
  - ✧ Must be re-written after being read
  - ✧ Must also be periodically refreshed
    - ▪ Each row can be refreshed simultaneously
  - ✧ Address lines are multiplexed
    - ▪ Upper half of address: Row Access Strobe (RAS)
    - ▪ Lower half of address: Column Access Strobe (CAS)



Static RAM Storage Cell

- ❖ Static RAM (SRAM): fast but expensive RAM
- ❖ 6-Transistor cell
- ❖ Typically used for caches
- ❖ Provides fast access time
- ❖ Cell Implementation:
  - ✧ Cross-coupled inverters store bit
  - ✧ Two pass transistors
  - ✧ Row decoder selects the word line
  - ✧ Pass transistors enable the cell to be read and written

Typical SRAM cell



Dynamic RAM Storage Cell

- ❖ Dynamic RAM (DRAM): slow, cheap, and dense memory
- ❖ Typical choice for main memory
- ❖ Cell Implementation:
  - ✧ 1-Transistor cell (pass transistor)
  - ✧ Trench capacitor (stores bit)
- ❖ Bit is stored as a charge on capacitor
- ❖ Must be refreshed periodically
  - ✧ Because of leakage of charge from tiny capacitor
- ❖ Refreshing for all memory rows
  - ✧ Reading each row and writing it back to restore the charge

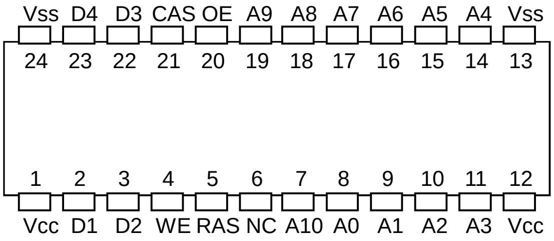
Typical DRAM cell

Typical DRAM Packaging

- 24-pin dual in-line package for 16Mbit = $2^{22} \times 4$ memory
- 22-bit address is divided into
  - 11-bit row address
  - 11-bit column address
  - Interleaved on same address lines

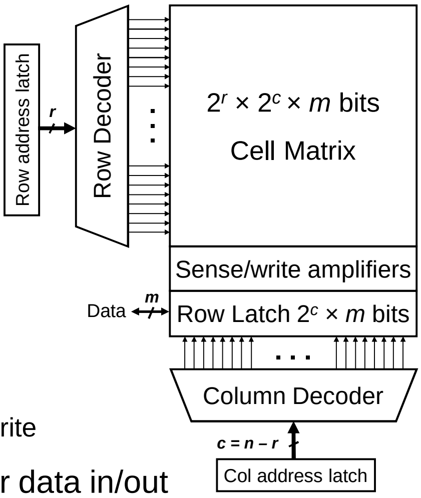
Legend

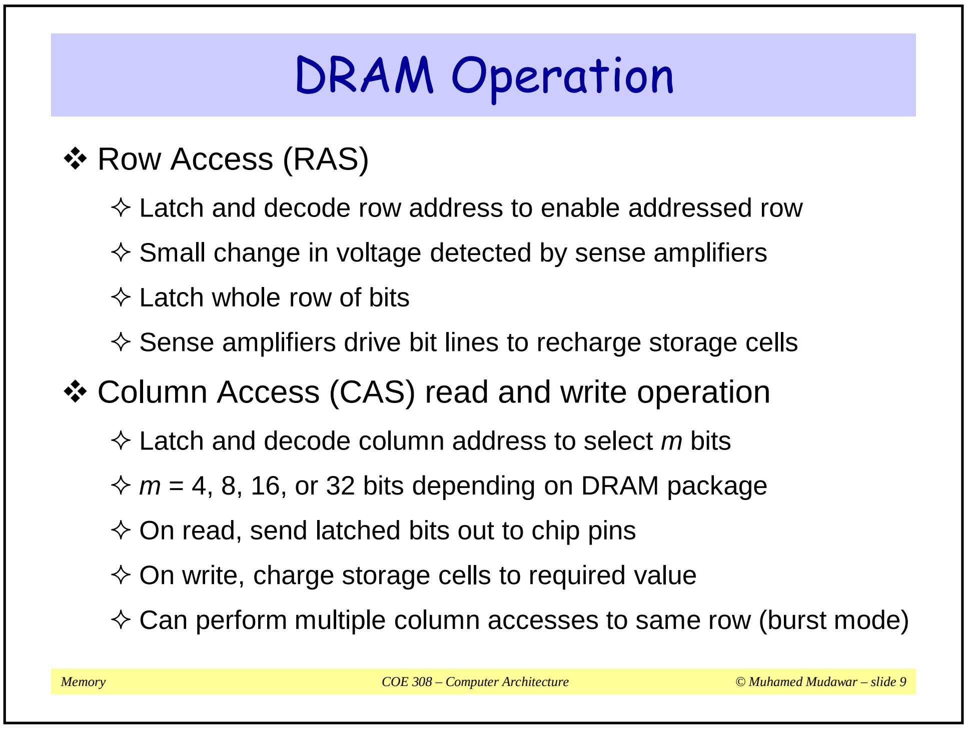
A*i*	Address bit *i*
CAS	Column address strobe
D*j*	Data bit *j*
NC	No connection
OE	Output enable
RAS	Row address strobe
WE	Write enable


Vss D4 D3 CAS OE A9 A8 A7 A6 A5 A4 Vss
24 23 22 21 20 19 18 17 16 15 14 13
1 2 3 4 5 6 7 8 9 10 11 12
Vcc D1 D2 WE RAS NC A10 A0 A1 A2 A3 Vcc

Typical Memory Structure

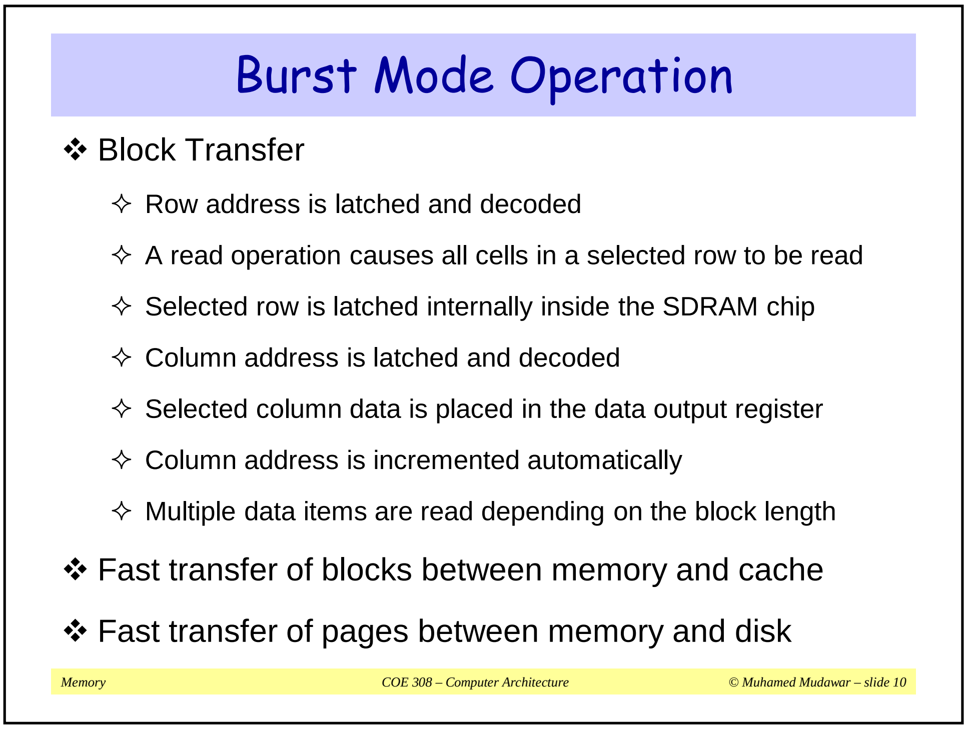
- Row decoder
  - Select row to read/write
- Column decoder
  - Select column to read/write
- Cell Matrix
  - 2D array of tiny memory cells
- Sense/Write amplifiers
  - Sense & amplify data on read
  - Drive bit line with data in on write
- Same data lines are used for data in/out





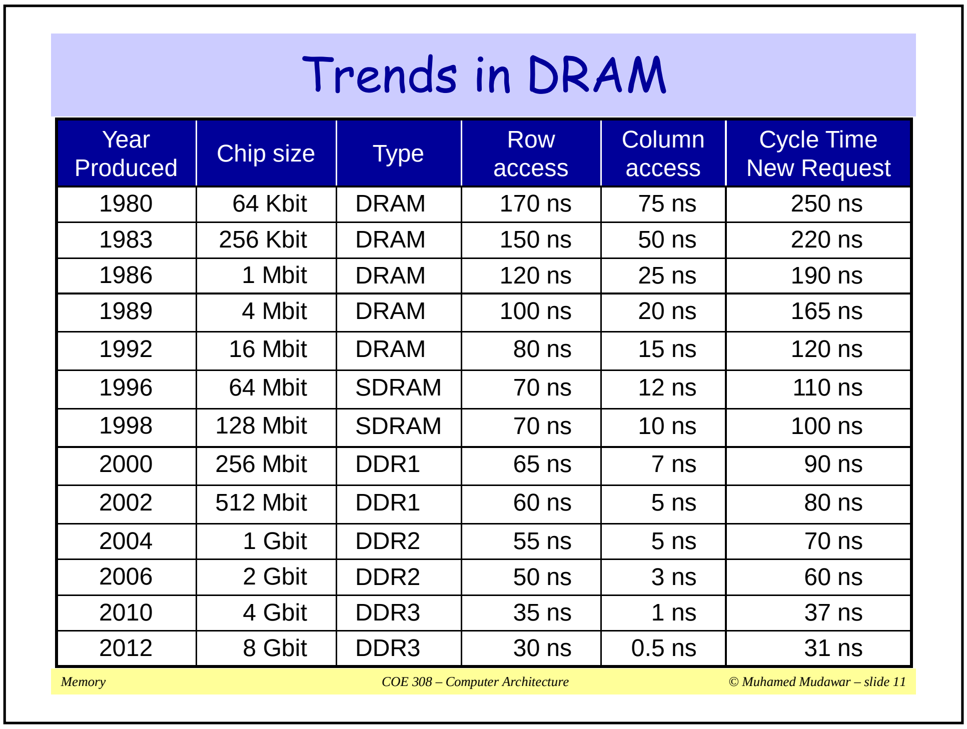
DRAM Operation

- ❖ Row Access (RAS)
  - ✧ Latch and decode row address to enable addressed row
  - ✧ Small change in voltage detected by sense amplifiers
  - ✧ Latch whole row of bits
  - ✧ Sense amplifiers drive bit lines to recharge storage cells
- ❖ Column Access (CAS) read and write operation
  - ✧ Latch and decode column address to select *m* bits
  - ✧ *m* = 4, 8, 16, or 32 bits depending on DRAM package
  - ✧ On read, send latched bits out to chip pins
  - ✧ On write, charge storage cells to required value
  - ✧ Can perform multiple column accesses to same row (burst mode)



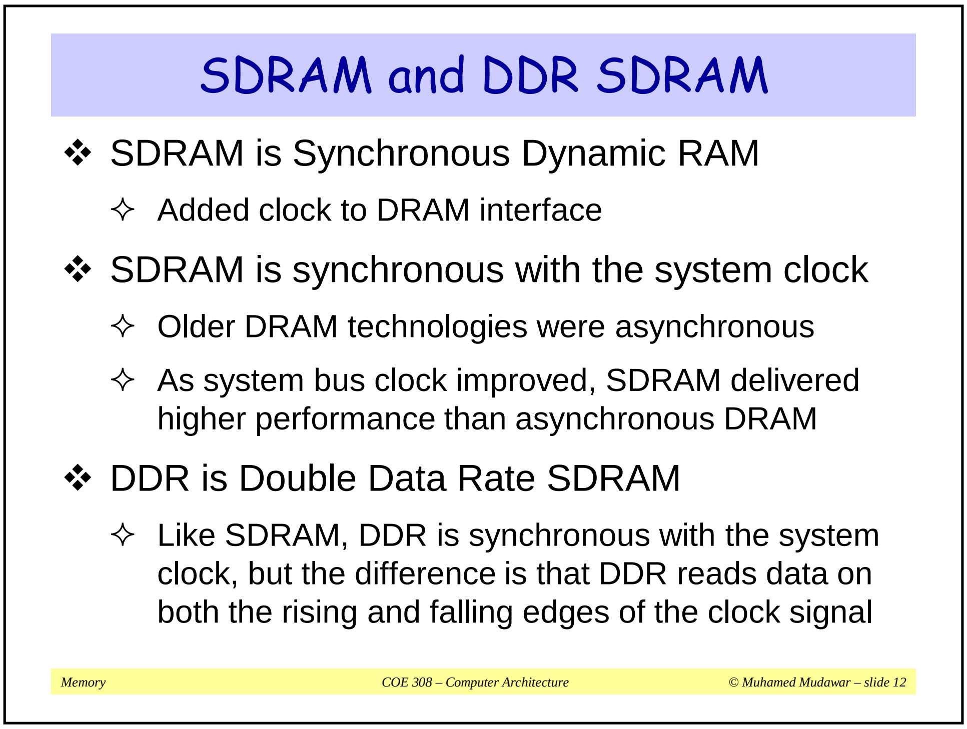
Burst Mode Operation

- ❖ Block Transfer
  - ✧ Row address is latched and decoded
  - ✧ A read operation causes all cells in a selected row to be read
  - ✧ Selected row is latched internally inside the SDRAM chip
  - ✧ Column address is latched and decoded
  - ✧ Selected column data is placed in the data output register
  - ✧ Column address is incremented automatically
  - ✧ Multiple data items are read depending on the block length
- ❖ Fast transfer of blocks between memory and cache
- ❖ Fast transfer of pages between memory and disk



Trends in DRAM

Year Produced	Chip size	Type	Row access	Column access	Cycle Time New Request
1980	64 Kbit	DRAM	170 ns	75 ns	250 ns
1983	256 Kbit	DRAM	150 ns	50 ns	220 ns
1986	1 Mbit	DRAM	120 ns	25 ns	190 ns
1989	4 Mbit	DRAM	100 ns	20 ns	165 ns
1992	16 Mbit	DRAM	80 ns	15 ns	120 ns
1996	64 Mbit	SDRAM	70 ns	12 ns	110 ns
1998	128 Mbit	SDRAM	70 ns	10 ns	100 ns
2000	256 Mbit	DDR1	65 ns	7 ns	90 ns
2002	512 Mbit	DDR1	60 ns	5 ns	80 ns
2004	1 Gbit	DDR2	55 ns	5 ns	70 ns
2006	2 Gbit	DDR2	50 ns	3 ns	60 ns
2010	4 Gbit	DDR3	35 ns	1 ns	37 ns
2012	8 Gbit	DDR3	30 ns	0.5 ns	31 ns



SDRAM and DDR SDRAM

- SDRAM is Synchronous Dynamic RAM
  - Added clock to DRAM interface
- SDRAM is synchronous with the system clock
  - Older DRAM technologies were asynchronous
  - As system bus clock improved, SDRAM delivered higher performance than asynchronous DRAM
- DDR is Double Data Rate SDRAM
  - Like SDRAM, DDR is synchronous with the system clock, but the difference is that DDR reads data on both the rising and falling edges of the clock signal

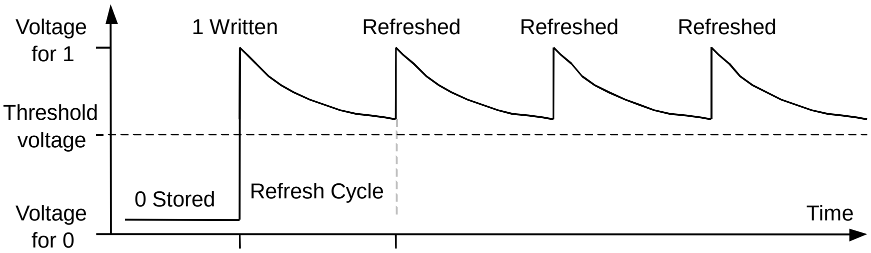
Transfer Rates & Peak Bandwidth

Standard Name	Memory Bus Clock	Millions Transfers per second	Module Name	Peak Bandwidth
DDR-200	100 MHz	200 MT/s	PC-1600	1600 MB/s
DDR-333	167 MHz	333 MT/s	PC-2700	2667 MB/s
DDR-400	200 MHz	400 MT/s	PC-3200	3200 MB/s
DDR2-667	333 MHz	667 MT/s	PC-5300	5333 MB/s
DDR2-800	400 MHz	800 MT/s	PC-6400	6400 MB/s
DDR2-1066	533 MHz	1066 MT/s	PC-8500	8533 MB/s
DDR3-1066	533 MHz	1066 MT/s	PC-8500	8533 MB/s
DDR3-1333	667 MHz	1333 MT/s	PC-10600	10667 MB/s
DDR3-1600	800 MHz	1600 MT/s	PC-12800	12800 MB/s
DDR4-3200	1600 MHz	3200 MT/s	PC-25600	25600 MB/s

- 1 Transfer = 64 bits = 8 bytes of data

DRAM Refresh Cycles

- Refresh cycle is about tens of milliseconds
- Refreshing is done for the entire memory
- Each row is read and written back to restore the charge
- Some of the memory bandwidth is lost to refresh cycles

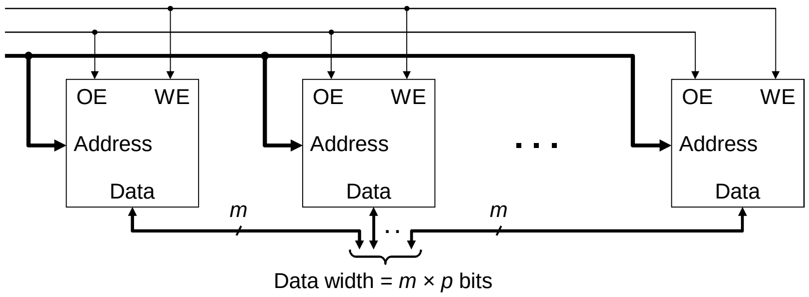


Loss of Bandwidth to Refresh Cycles

- Example:
  - A 256 Mb DRAM chip
  - Organized internally as a 16K × 16K cell matrix
  - Rows must be refreshed at least once every 50 ms
  - Refreshing a row takes 100 ns
  - What fraction of the memory bandwidth is lost to refresh cycles?
- Solution:
  - Refreshing all 16K rows takes: 16 × 1024 × 100 ns = 1.64 ms
  - Loss of 1.64 ms every 50 ms
  - Fraction of lost memory bandwidth = 1.64 / 50 = 3.3%

Expanding the Data Bus Width

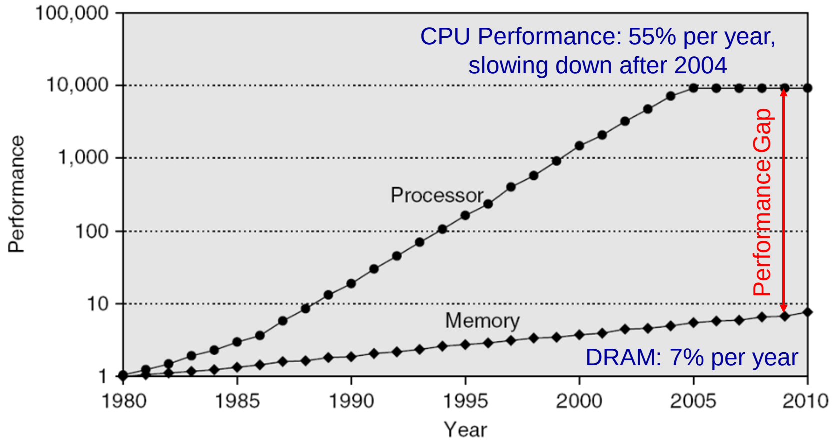
- Memory chips typically have a narrow data bus
- We can expand the data bus width by a factor of *p*
  - Use *p* RAM chips and feed the same address to all chips
  - Use the same Output Enable and Write Enable control signals



Next . . .

- ❖ Random Access Memory and its Structure
- ❖ Memory Hierarchy and the need for Cache Memory
- ❖ The Basics of Caches
- ❖ Cache Performance and Memory Stall Cycles
- ❖ Improving Cache Performance
- ❖ Multilevel Caches

Processor-Memory Performance Gap



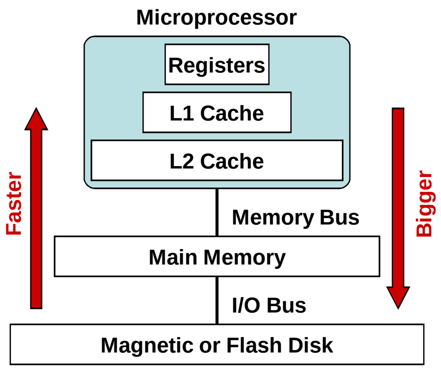
- ❖ 1980 – No cache in microprocessor
- ❖ 1995 – Two-level cache on microprocessor

The Need for Cache Memory

- Widening speed gap between CPU and main memory
  - Processor operation takes less than 1 ns
  - Main memory requires about 100 ns to access
- Each instruction involves at least one memory access
  - One memory access to fetch the instruction
  - A second memory access for load and store instructions
- Memory bandwidth limits the instruction execution rate
- Cache memory can help bridge the CPU-memory gap
- Cache memory is small in size but fast

Typical Memory Hierarchy

- Registers are at the top of the hierarchy
  - Typical size < 1 KB
  - Access time < 0.5 ns
- Level 1 Cache (8 – 64 KB)
  - Access time: 1 ns
- L2 Cache (512KB – 8MB)
  - Access time: 3 – 10 ns
- Main Memory (4 – 16 GB)
  - Access time: 50 – 100 ns
- Disk Storage (> 200 GB)
  - Access time: 5 – 10 ms



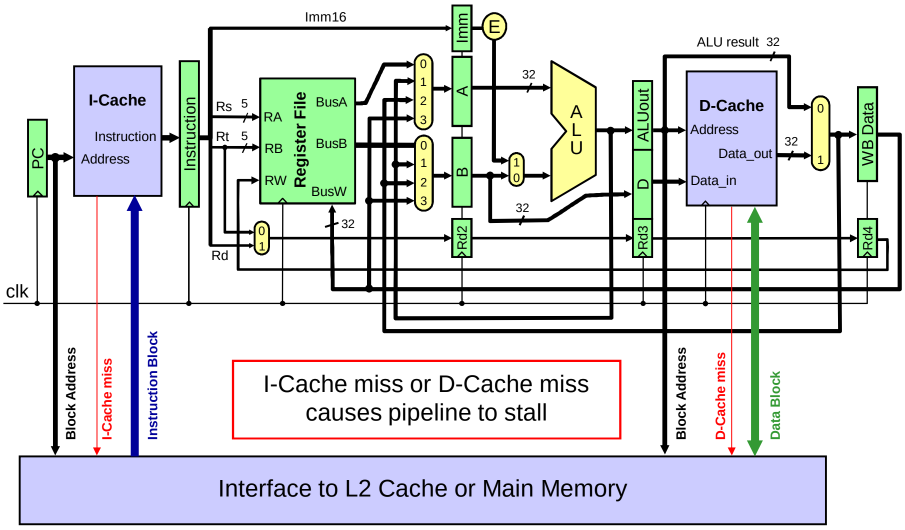
Principle of Locality of Reference

- ❖ Programs access small portion of their address space
  - ✧ At any time, only a small set of instructions & data is needed
- ❖ Temporal Locality (in time)
  - ✧ If an item is accessed, probably it will be accessed again soon
  - ✧ Same loop instructions are fetched each iteration
  - ✧ Same procedure may be called and executed many times
- ❖ Spatial Locality (in space)
  - ✧ Tendency to access contiguous instructions/data in memory
  - ✧ Sequential execution of Instructions
  - ✧ Traversing arrays element by element

What is a Cache Memory ?

- ❖ Small and fast (SRAM) memory technology
  - ✧ Stores the subset of instructions & data currently being accessed
- ❖ Used to reduce average access time to memory
- ❖ Caches exploit temporal locality by …
  - ✧ Keeping recently accessed data closer to the processor
- ❖ Caches exploit spatial locality by …
  - ✧ Moving blocks consisting of multiple contiguous words
- ❖ Goal is to achieve
  - ✧ Fast speed of cache memory access
  - ✧ Balance the cost of the memory system

Cache Memories in the Datapath

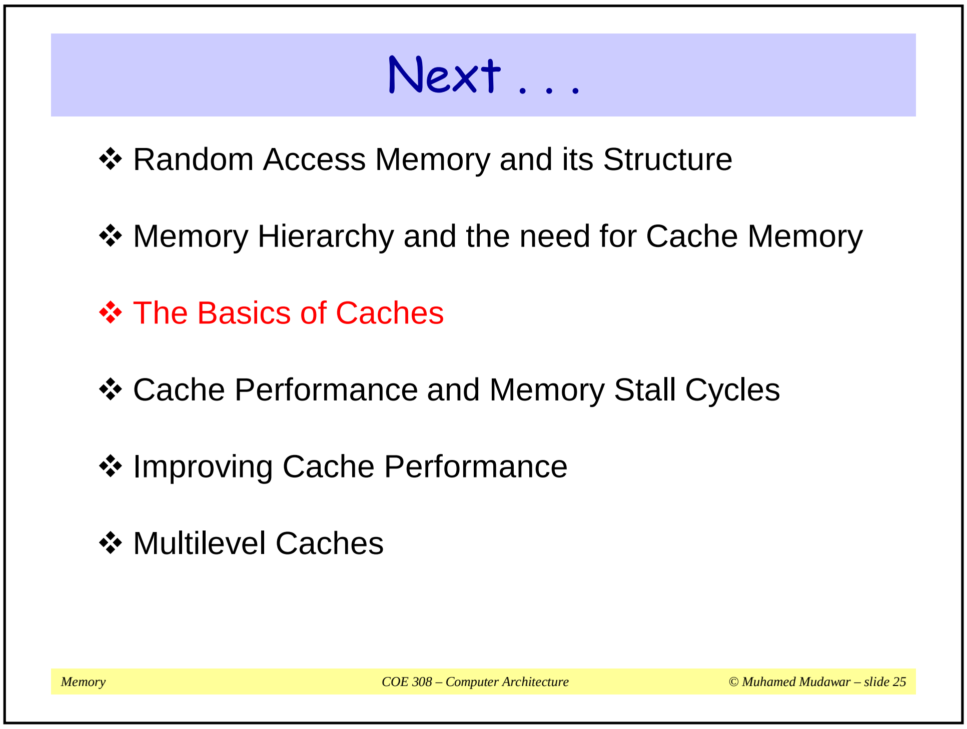


I-Cache miss or D-Cache miss causes pipeline to stall

Interface to L2 Cache or Main Memory

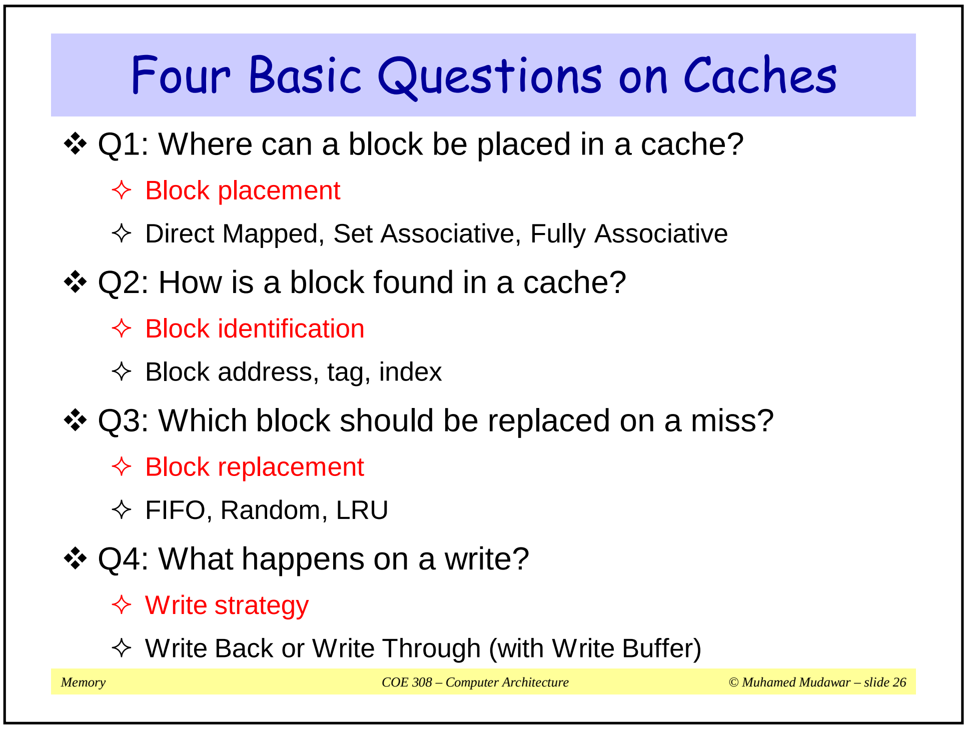
Almost Everything is a Cache !

- In computer architecture, almost everything is a cache!
- Registers: a cache on variables – software managed
- First-level cache: a cache on L2 cache or memory
- Second-level cache: a cache on memory
- Memory: a cache on hard disk
  - Stores recent programs and their data
  - Hard disk can be viewed as an extension to main memory
- Branch target and prediction buffer
  - Cache on branch target and prediction information



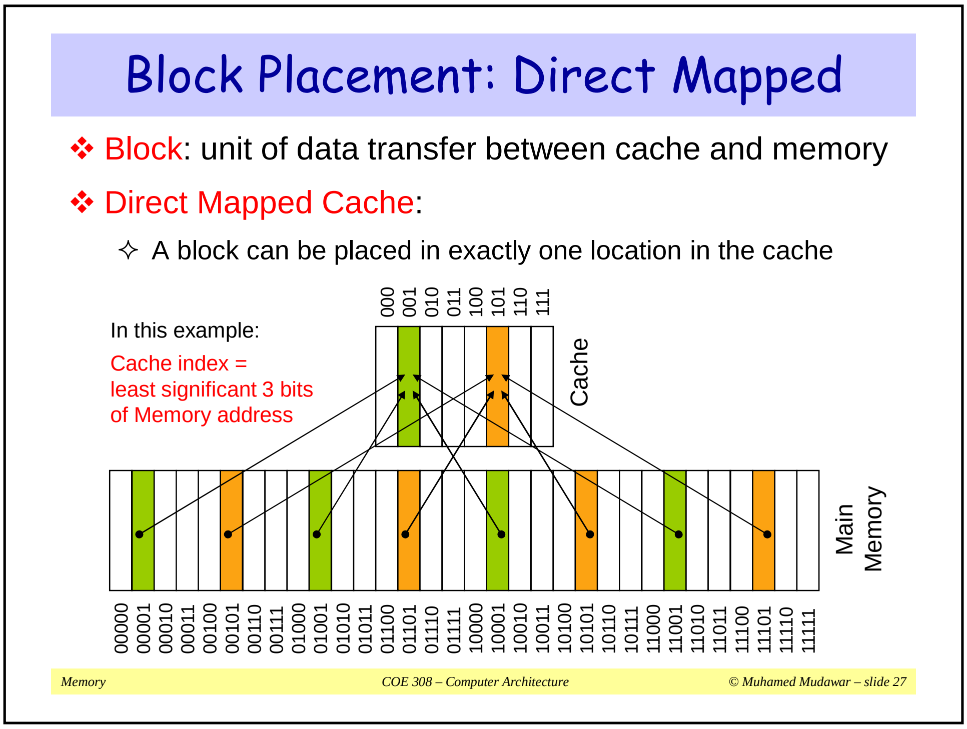
Next . . .

- Random Access Memory and its Structure
- Memory Hierarchy and the need for Cache Memory
- The Basics of Caches
- Cache Performance and Memory Stall Cycles
- Improving Cache Performance
- Multilevel Caches



Four Basic Questions on Caches

- Q1: Where can a block be placed in a cache?
  - Block placement
  - Direct Mapped, Set Associative, Fully Associative
- Q2: How is a block found in a cache?
  - Block identification
  - Block address, tag, index
- Q3: Which block should be replaced on a miss?
  - Block replacement
  - FIFO, Random, LRU
- Q4: What happens on a write?
  - Write strategy
  - Write Back or Write Through (with Write Buffer)

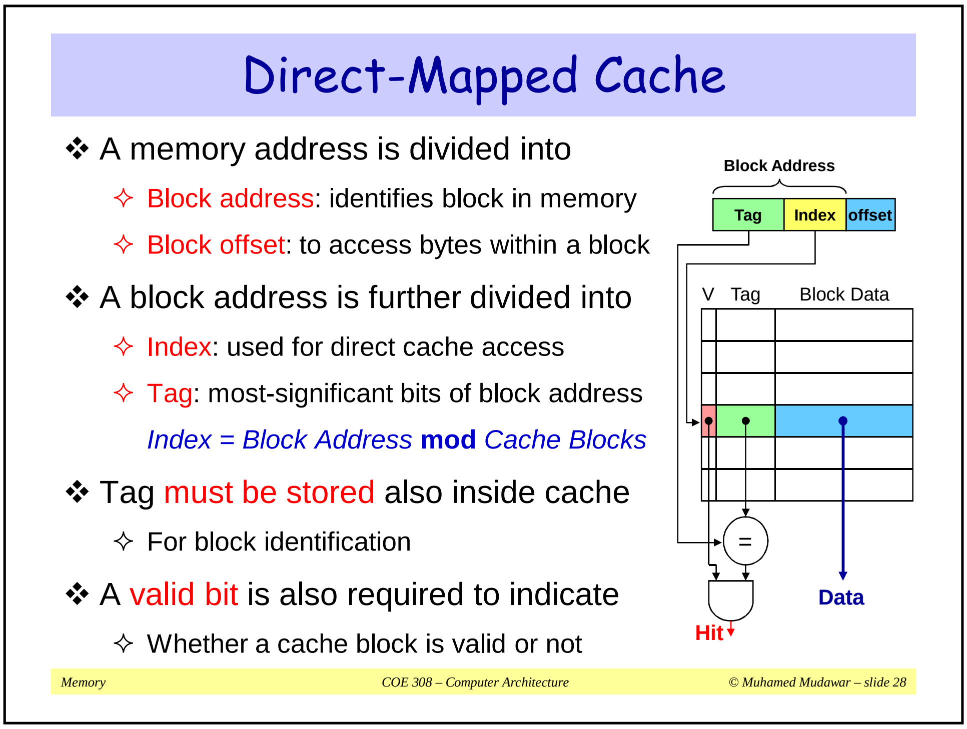


Block Placement: Direct Mapped

- Block: unit of data transfer between cache and memory
- Direct Mapped Cache:
  - A block can be placed in exactly one location in the cache

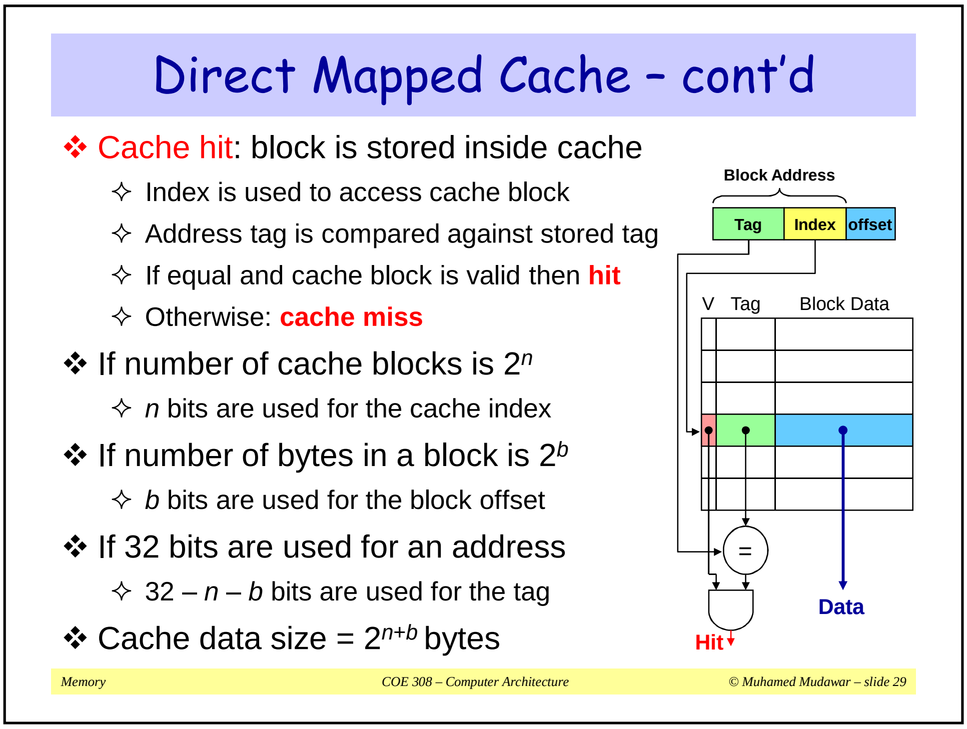
In this example:

Cache index = least significant 3 bits of Memory address



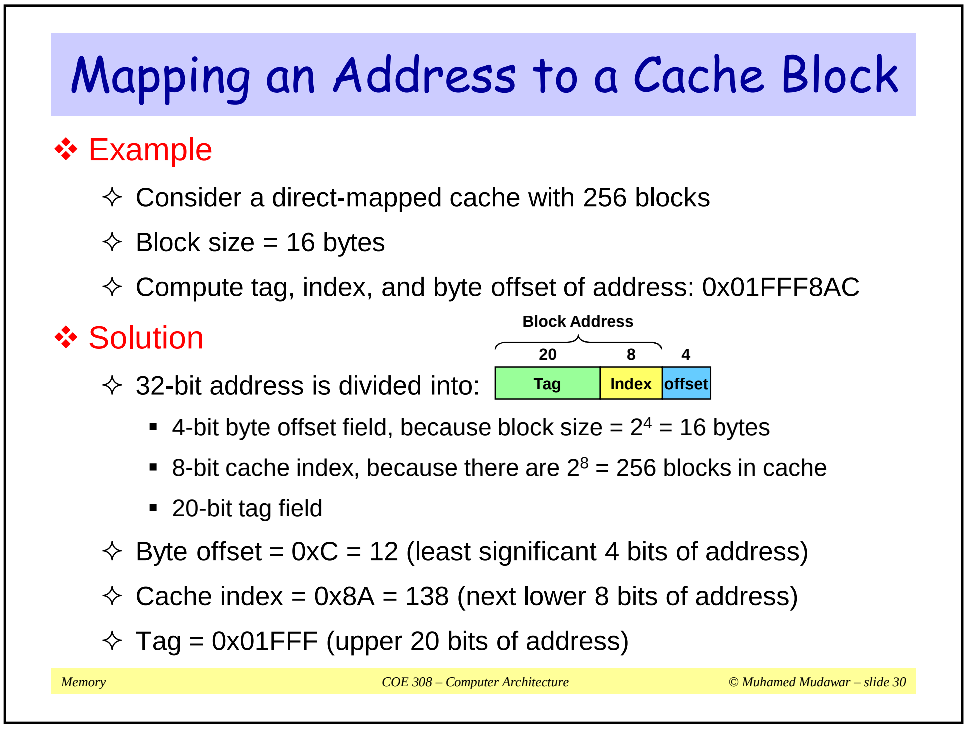
Direct-Mapped Cache

- A memory address is divided into
  - Block address: identifies block in memory
  - Block offset: to access bytes within a block
- A block address is further divided into
  - Index: used for direct cache access
  - Tag: most-significant bits of block address
  - *Index = Block Address* **mod** *Cache Blocks*
- Tag must be stored also inside cache
  - For block identification
- A valid bit is also required to indicate
  - Whether a cache block is valid or not



Direct Mapped Cache – cont'd

- ❖ Cache hit: block is stored inside cache
  - ✧ Index is used to access cache block
  - ✧ Address tag is compared against stored tag
  - ✧ If equal and cache block is valid then **hit**
  - ✧ Otherwise: **cache miss**
- ❖ If number of cache blocks is 2^n
  - ✧ n bits are used for the cache index
- ❖ If number of bytes in a block is 2^b
  - ✧ b bits are used for the block offset
- ❖ If 32 bits are used for an address
  - ✧ $32 - n - b$ bits are used for the tag
- ❖ Cache data size = 2^{n+b} bytes



Mapping an Address to a Cache Block

- ❖ Example
  - ✧ Consider a direct-mapped cache with 256 blocks
  - ✧ Block size = 16 bytes
  - ✧ Compute tag, index, and byte offset of address: 0x01FFF8AC
- ❖ Solution
  - ✧ 32-bit address is divided into:
    - 4-bit byte offset field, because block size = 2^4 = 16 bytes
    - 8-bit cache index, because there are 2^8 = 256 blocks in cache
    - 20-bit tag field
  - ✧ Byte offset = 0xC = 12 (least significant 4 bits of address)
  - ✧ Cache index = 0x8A = 138 (next lower 8 bits of address)
  - ✧ Tag = 0x01FFF (upper 20 bits of address)

Example on Cache Placement & Misses

- ❖ Consider a small direct-mapped cache with 32 blocks
  - ✧ Cache is initially empty, Block size = 16 bytes
  - ✧ The following memory addresses (in decimal) are referenced:

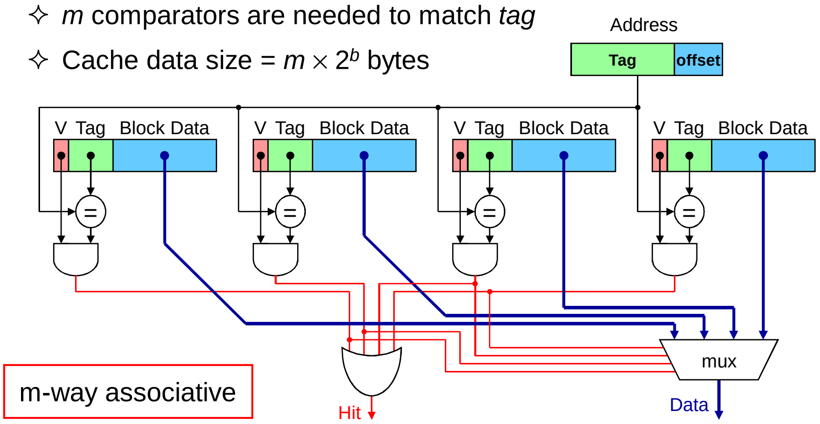
    1000, 1004, 1008, 2548, 2552, 2556.
  - ✧ Map addresses to cache blocks and indicate whether hit or miss
- ❖ Solution:

23	5	4
Tag	Index	offset

✧ 1000 = 0x3E8	cache index = 0x1E	Miss (first access)
✧ 1004 = 0x3EC	cache index = 0x1E	Hit
✧ 1008 = 0x3F0	cache index = 0x1F	Miss (first access)
✧ 2548 = 0x9F4	cache index = 0x1F	Miss (different tag)
✧ 2552 = 0x9F8	cache index = 0x1F	Hit
✧ 2556 = 0x9FC	cache index = 0x1F	Hit

Fully Associative Cache

- ❖ A block can be placed anywhere in cache ⇒ no indexing
- ❖ If *m* blocks exist then
  - ✧ *m* comparators are needed to match *tag*
  - ✧ Cache data size = $m \times 2^b$ bytes

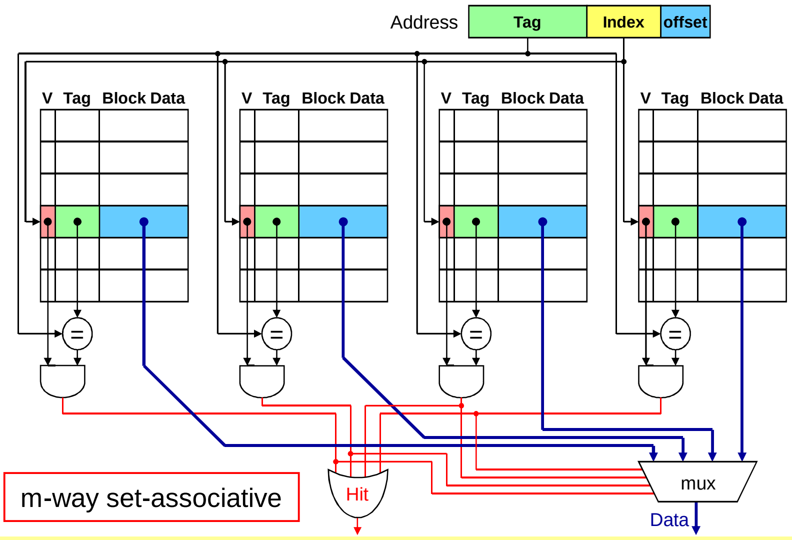


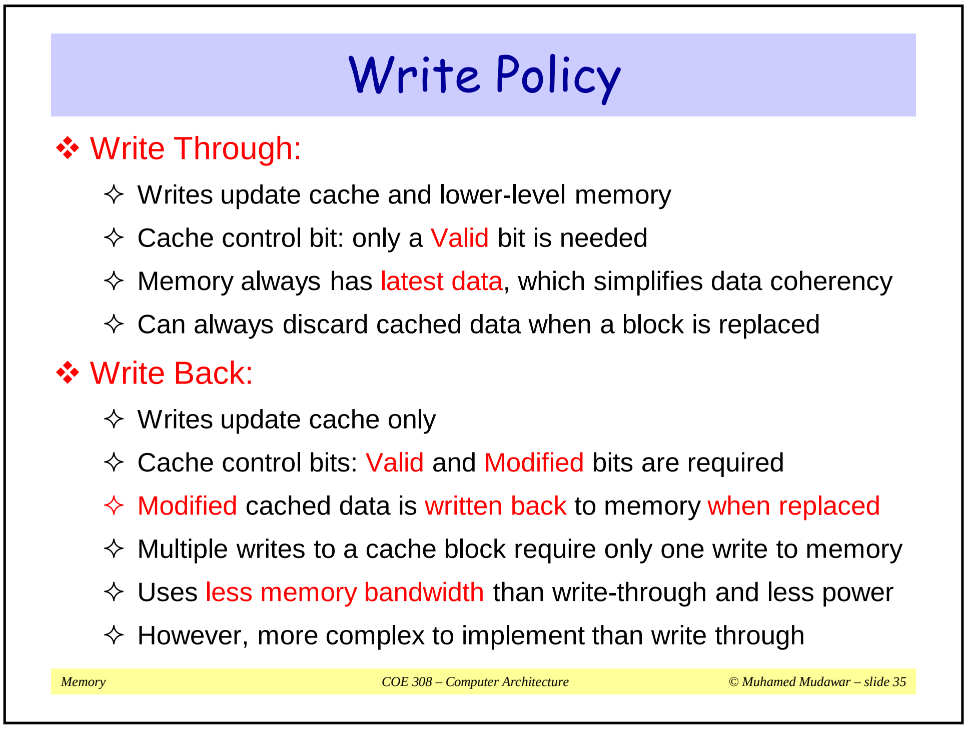
m-way associative

Set-Associative Cache

- A set is a group of blocks that can be indexed
- A block is first mapped onto a set
  - *Set index = Block address* **mod** *Number of sets in cache*
- If there are *m* blocks in a set (*m*-way set associative) then
  - *m* tags are checked in parallel using *m* comparators
- If 2^n sets exist then set index consists of *n* bits
- Cache data size = $m \times 2^{n+b}$ bytes (with 2^b bytes per block)
  - Without counting tags and valid bits
- A direct-mapped cache has one block per set (*m* = 1)
- A fully-associative cache has one set (2^n = 1 or *n* = 0)

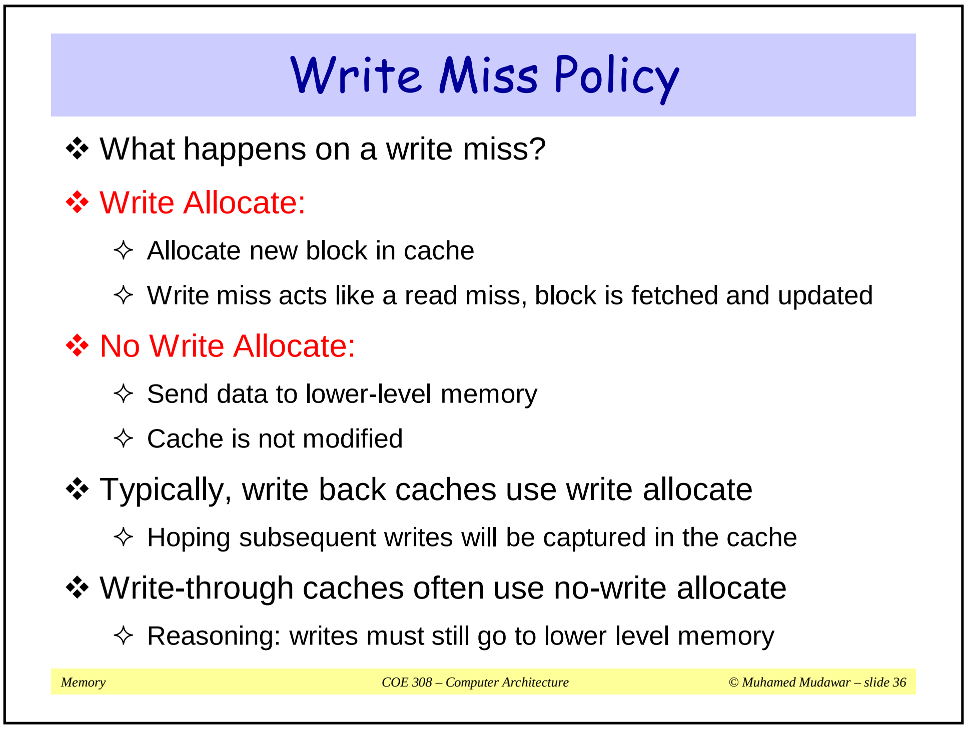
Set-Associative Cache Diagram





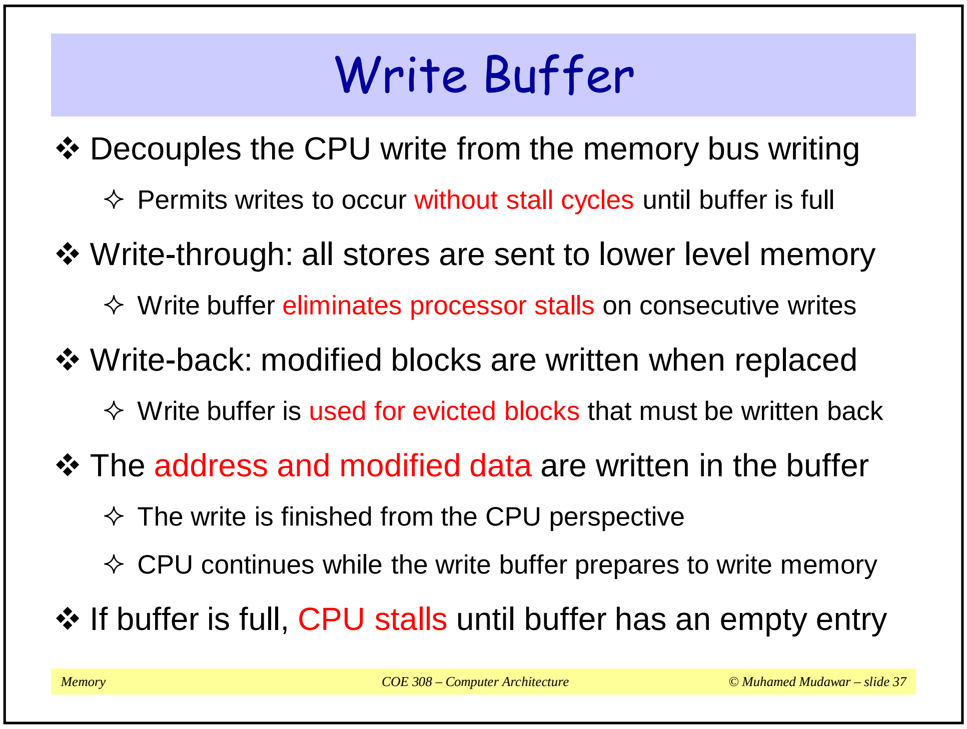
Write Policy

- ❖ Write Through:
  - ✧ Writes update cache and lower-level memory
  - ✧ Cache control bit: only a Valid bit is needed
  - ✧ Memory always has latest data, which simplifies data coherency
  - ✧ Can always discard cached data when a block is replaced
- ❖ Write Back:
  - ✧ Writes update cache only
  - ✧ Cache control bits: Valid and Modified bits are required
  - ✧ Modified cached data is written back to memory when replaced
  - ✧ Multiple writes to a cache block require only one write to memory
  - ✧ Uses less memory bandwidth than write-through and less power
  - ✧ However, more complex to implement than write through



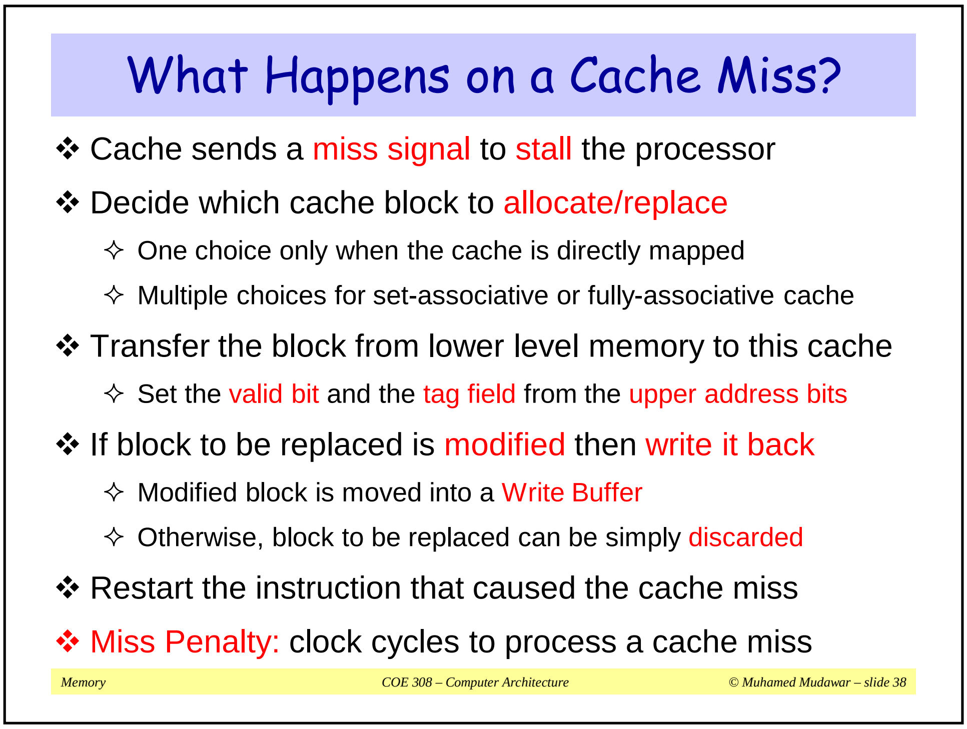
Write Miss Policy

- ❖ What happens on a write miss?
- ❖ Write Allocate:
  - ✧ Allocate new block in cache
  - ✧ Write miss acts like a read miss, block is fetched and updated
- ❖ No Write Allocate:
  - ✧ Send data to lower-level memory
  - ✧ Cache is not modified
- ❖ Typically, write back caches use write allocate
  - ✧ Hoping subsequent writes will be captured in the cache
- ❖ Write-through caches often use no-write allocate
  - ✧ Reasoning: writes must still go to lower level memory



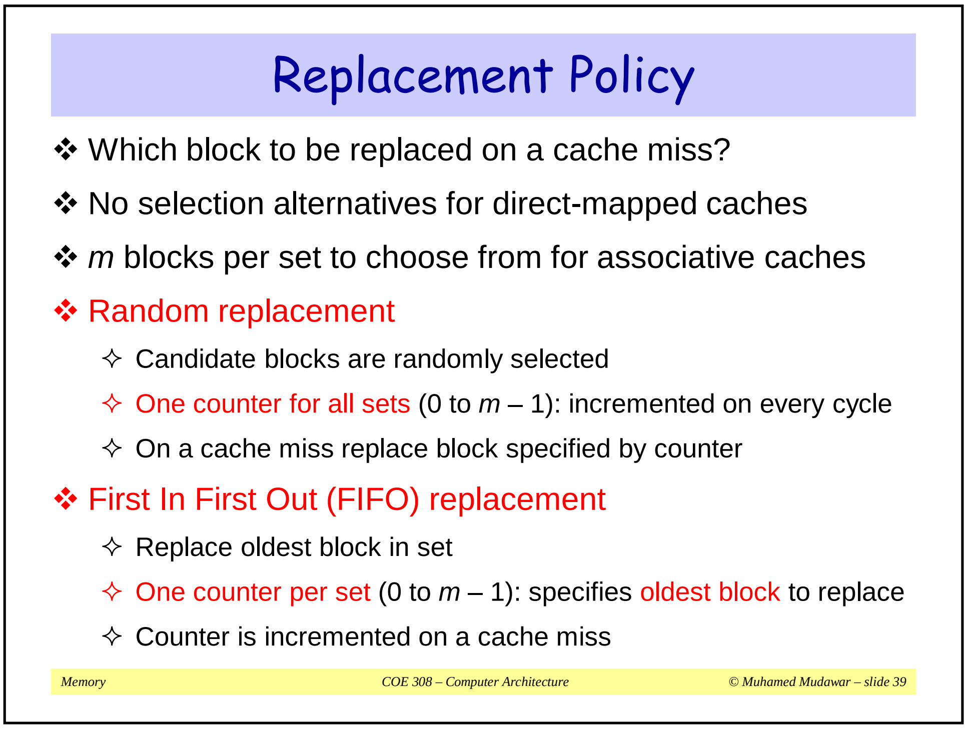
Write Buffer

- ❖ Decouples the CPU write from the memory bus writing
  - ✧ Permits writes to occur without stall cycles until buffer is full
- ❖ Write-through: all stores are sent to lower level memory
  - ✧ Write buffer eliminates processor stalls on consecutive writes
- ❖ Write-back: modified blocks are written when replaced
  - ✧ Write buffer is used for evicted blocks that must be written back
- ❖ The address and modified data are written in the buffer
  - ✧ The write is finished from the CPU perspective
  - ✧ CPU continues while the write buffer prepares to write memory
- ❖ If buffer is full, CPU stalls until buffer has an empty entry



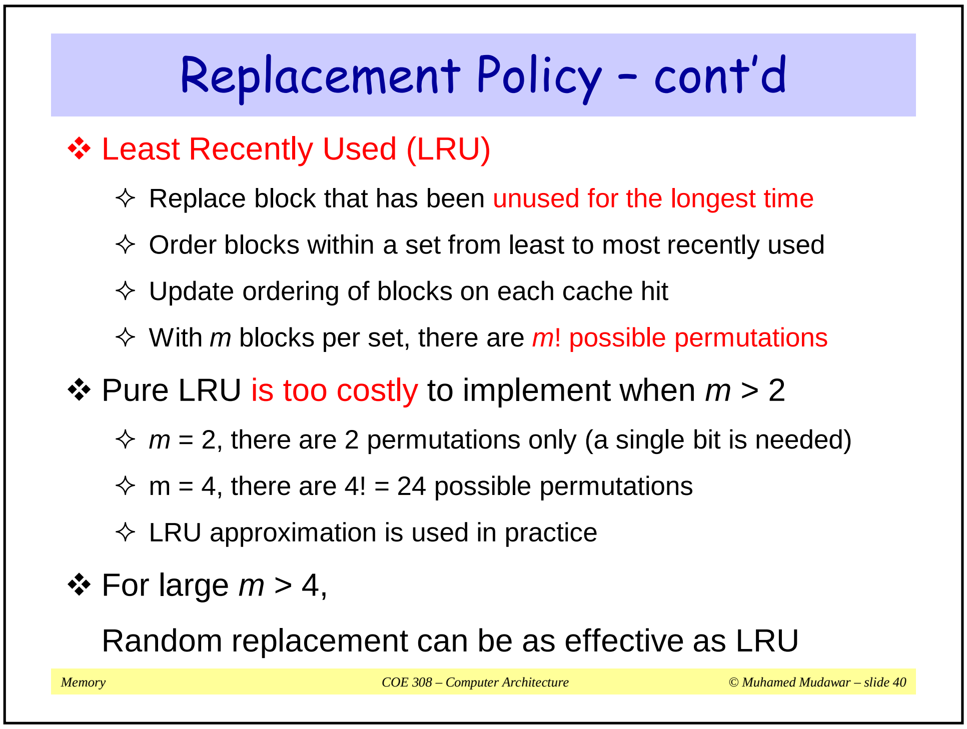
What Happens on a Cache Miss?

- ❖ Cache sends a miss signal to stall the processor
- ❖ Decide which cache block to allocate/replace
  - ✧ One choice only when the cache is directly mapped
  - ✧ Multiple choices for set-associative or fully-associative cache
- ❖ Transfer the block from lower level memory to this cache
  - ✧ Set the valid bit and the tag field from the upper address bits
- ❖ If block to be replaced is modified then write it back
  - ✧ Modified block is moved into a Write Buffer
  - ✧ Otherwise, block to be replaced can be simply discarded
- ❖ Restart the instruction that caused the cache miss
- ❖ Miss Penalty: clock cycles to process a cache miss



Replacement Policy

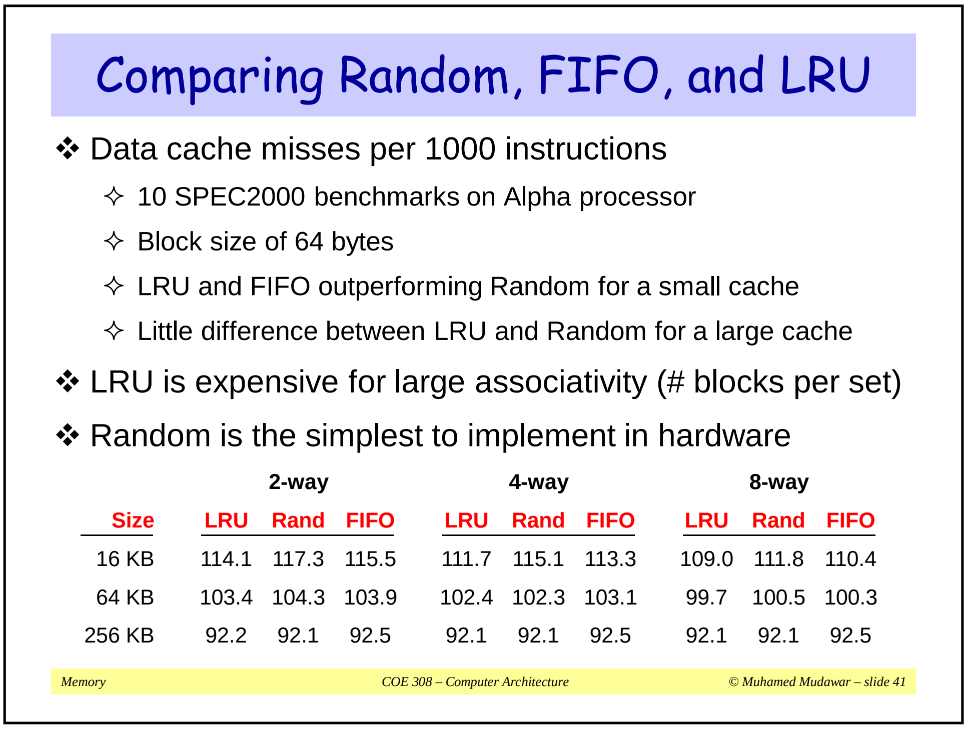
- ❖ Which block to be replaced on a cache miss?
- ❖ No selection alternatives for direct-mapped caches
- ❖ *m* blocks per set to choose from for associative caches
- ❖ Random replacement
  - ✧ Candidate blocks are randomly selected
  - ✧ One counter for all sets (0 to *m* – 1): incremented on every cycle
  - ✧ On a cache miss replace block specified by counter
- ❖ First In First Out (FIFO) replacement
  - ✧ Replace oldest block in set
  - ✧ One counter per set (0 to *m* – 1): specifies oldest block to replace
  - ✧ Counter is incremented on a cache miss



Replacement Policy - cont'd

- ❖ Least Recently Used (LRU)
  - ✧ Replace block that has been unused for the longest time
  - ✧ Order blocks within a set from least to most recently used
  - ✧ Update ordering of blocks on each cache hit
  - ✧ With *m* blocks per set, there are *m*! possible permutations
- ❖ Pure LRU is too costly to implement when *m* > 2
  - ✧ *m* = 2, there are 2 permutations only (a single bit is needed)
  - ✧ m = 4, there are 4! = 24 possible permutations
  - ✧ LRU approximation is used in practice
- ❖ For large *m* > 4,

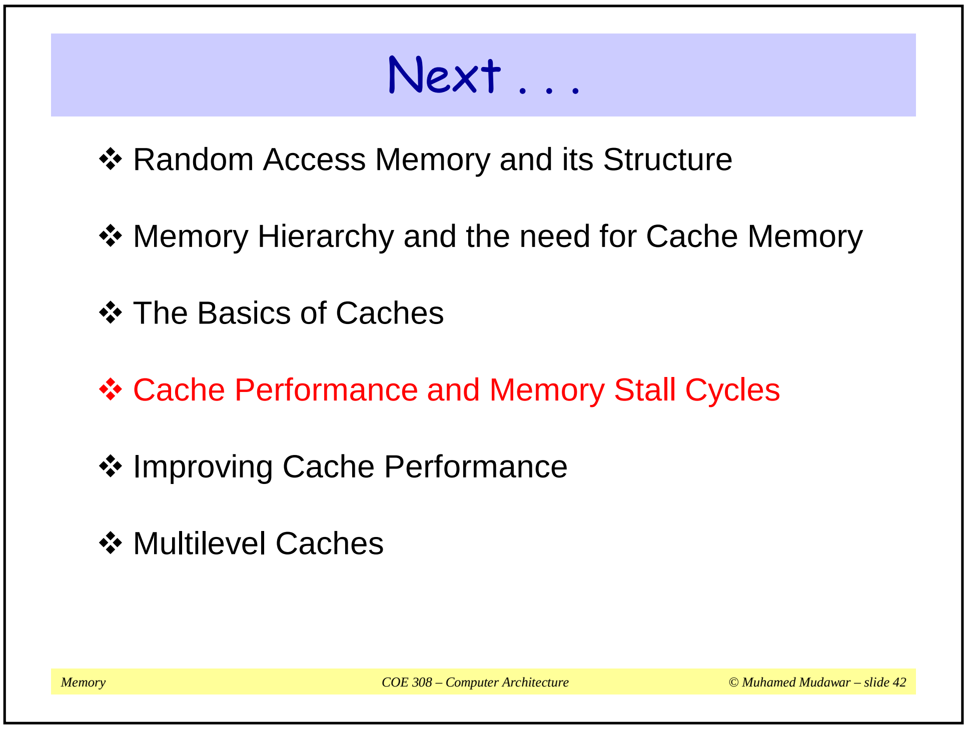
  Random replacement can be as effective as LRU



Comparing Random, FIFO, and LRU

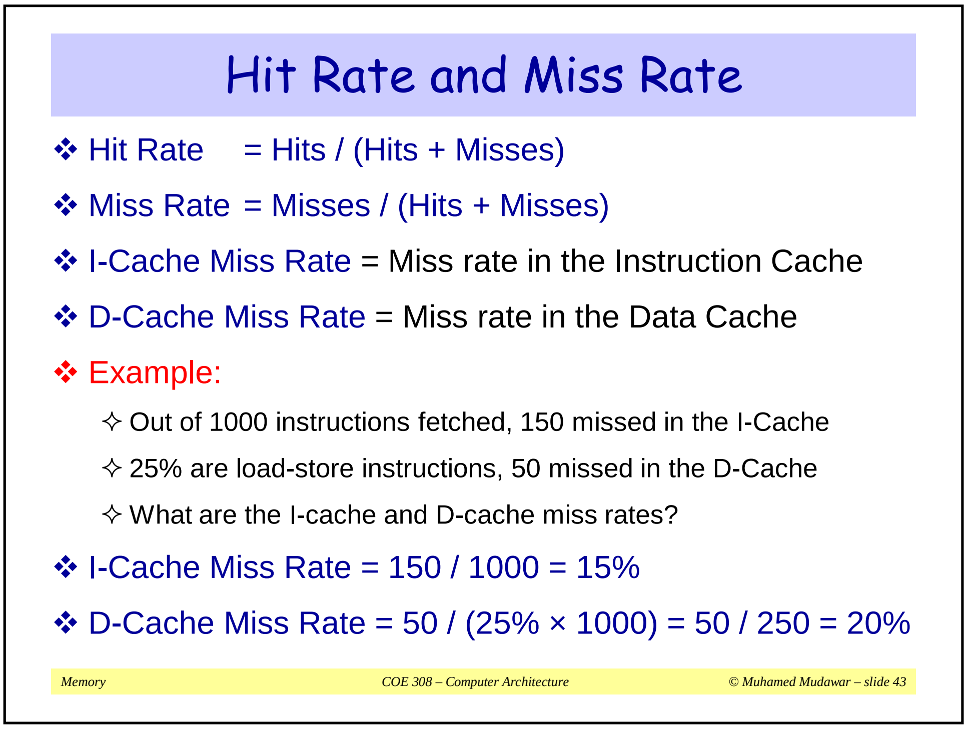
- ❖ Data cache misses per 1000 instructions
  - ✧ 10 SPEC2000 benchmarks on Alpha processor
  - ✧ Block size of 64 bytes
  - ✧ LRU and FIFO outperforming Random for a small cache
  - ✧ Little difference between LRU and Random for a large cache
- ❖ LRU is expensive for large associativity (# blocks per set)
- ❖ Random is the simplest to implement in hardware

	2-way			4-way			8-way		
Size	**LRU**	**Rand**	**FIFO**	**LRU**	**Rand**	**FIFO**	**LRU**	**Rand**	**FIFO**
16 KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64 KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256 KB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5



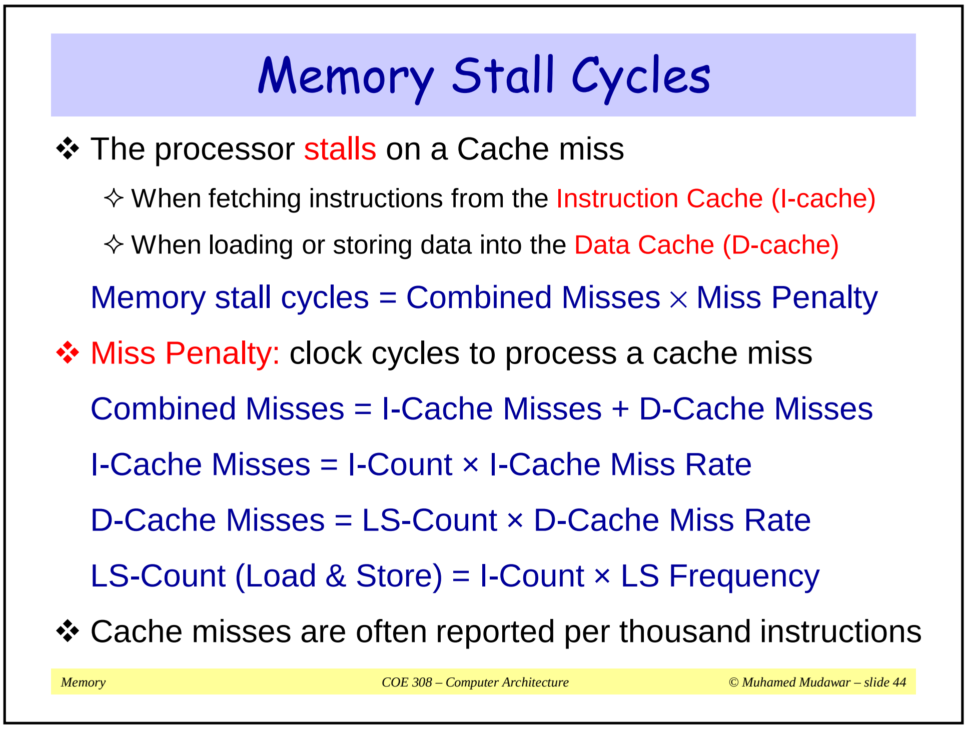
Next . . .

- ❖ Random Access Memory and its Structure
- ❖ Memory Hierarchy and the need for Cache Memory
- ❖ The Basics of Caches
- ❖ Cache Performance and Memory Stall Cycles
- ❖ Improving Cache Performance
- ❖ Multilevel Caches



Hit Rate and Miss Rate

- Hit Rate = Hits / (Hits + Misses)
- Miss Rate = Misses / (Hits + Misses)
- I-Cache Miss Rate = Miss rate in the Instruction Cache
- D-Cache Miss Rate = Miss rate in the Data Cache
- Example:
  - Out of 1000 instructions fetched, 150 missed in the I-Cache
  - 25% are load-store instructions, 50 missed in the D-Cache
  - What are the I-cache and D-cache miss rates?
- I-Cache Miss Rate = 150 / 1000 = 15%
- D-Cache Miss Rate = 50 / (25% × 1000) = 50 / 250 = 20%



Memory Stall Cycles

- The processor stalls on a Cache miss
  - When fetching instructions from the Instruction Cache (I-cache)
  - When loading or storing data into the Data Cache (D-cache)

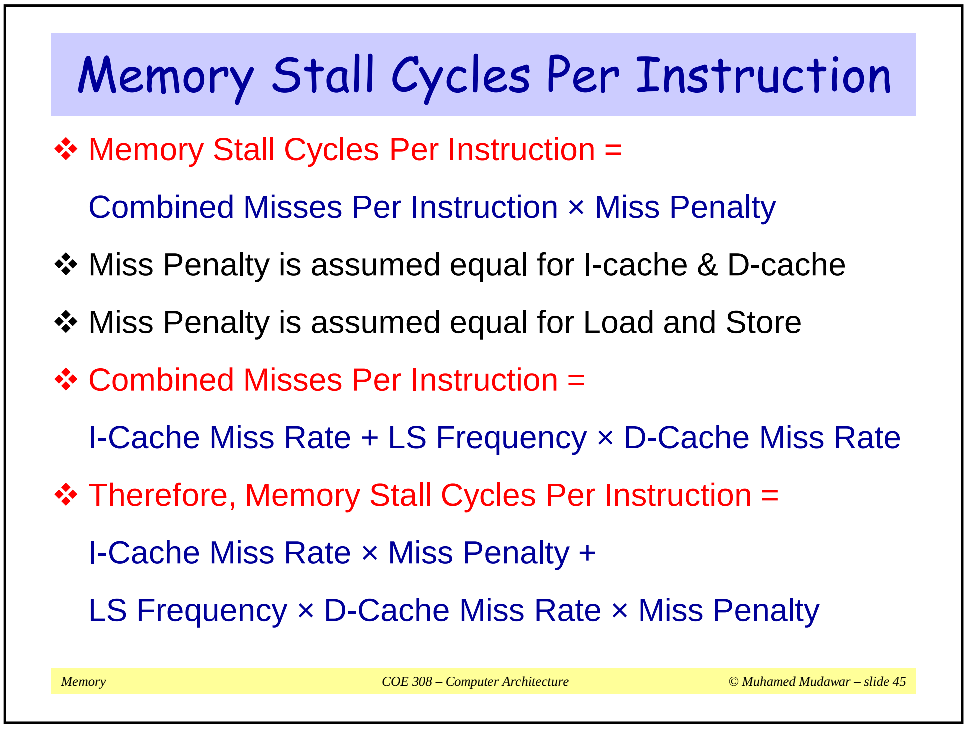
  Memory stall cycles = Combined Misses × Miss Penalty
- Miss Penalty: clock cycles to process a cache miss

  Combined Misses = I-Cache Misses + D-Cache Misses

  I-Cache Misses = I-Count × I-Cache Miss Rate

  D-Cache Misses = LS-Count × D-Cache Miss Rate

  LS-Count (Load & Store) = I-Count × LS Frequency
- Cache misses are often reported per thousand instructions



Memory Stall Cycles Per Instruction

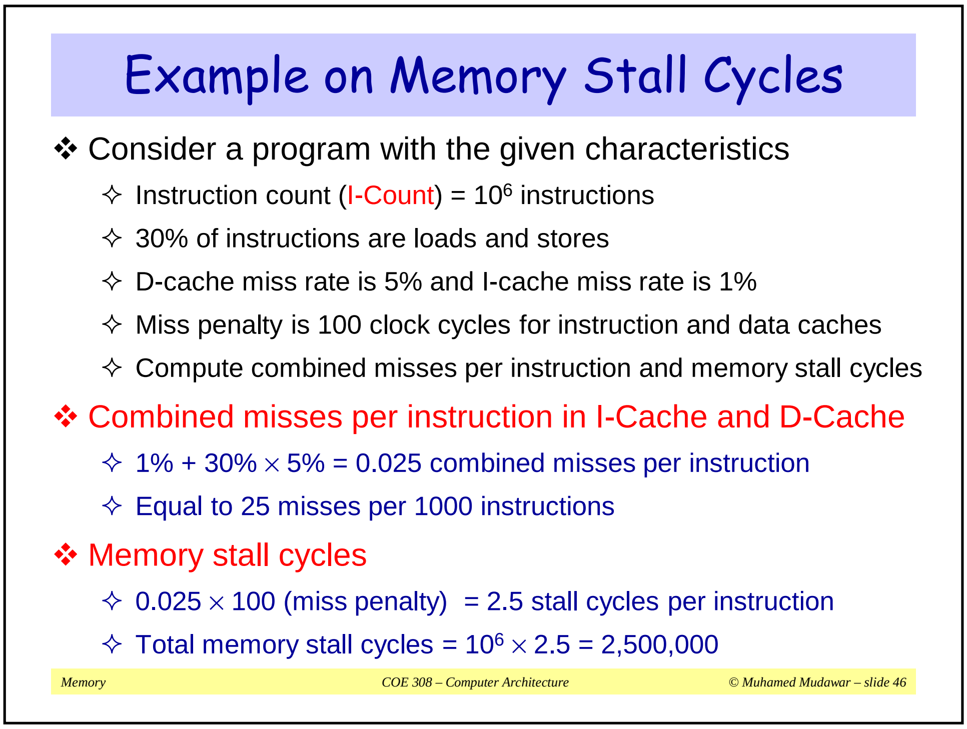
- Memory Stall Cycles Per Instruction =

  Combined Misses Per Instruction × Miss Penalty
- Miss Penalty is assumed equal for I-cache & D-cache
- Miss Penalty is assumed equal for Load and Store
- Combined Misses Per Instruction =

  I-Cache Miss Rate + LS Frequency × D-Cache Miss Rate
- Therefore, Memory Stall Cycles Per Instruction =

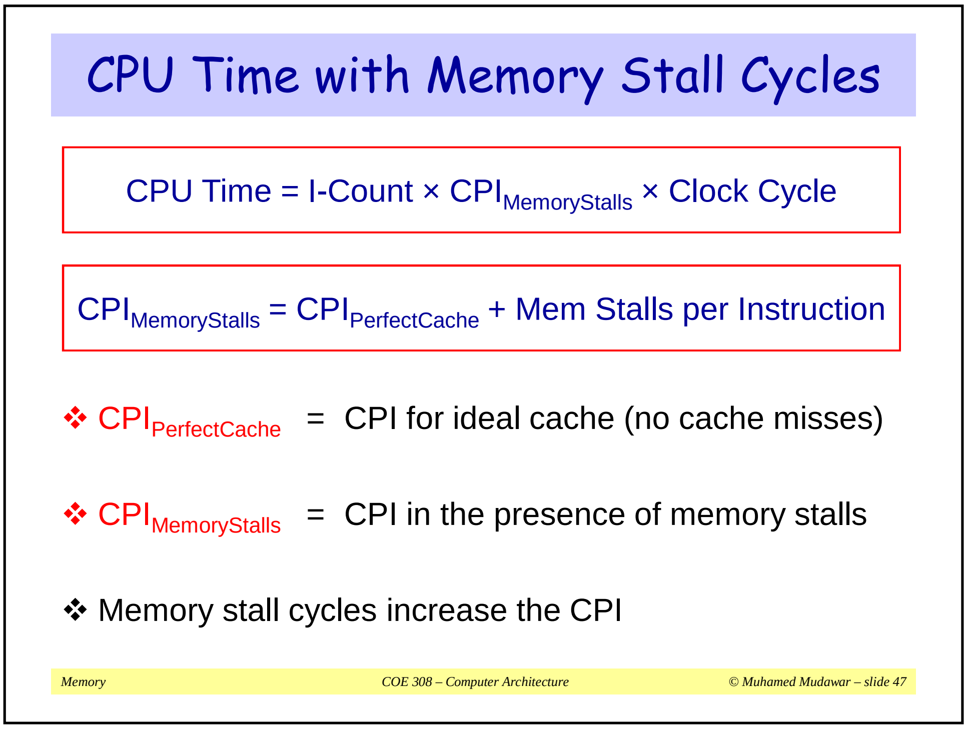
  I-Cache Miss Rate × Miss Penalty +

  LS Frequency × D-Cache Miss Rate × Miss Penalty



Example on Memory Stall Cycles

- Consider a program with the given characteristics
  - Instruction count (I-Count) = 10^6 instructions
  - 30% of instructions are loads and stores
  - D-cache miss rate is 5% and I-cache miss rate is 1%
  - Miss penalty is 100 clock cycles for instruction and data caches
  - Compute combined misses per instruction and memory stall cycles
- Combined misses per instruction in I-Cache and D-Cache
  - 1% + 30% × 5% = 0.025 combined misses per instruction
  - Equal to 25 misses per 1000 instructions
- Memory stall cycles
  - 0.025 × 100 (miss penalty) = 2.5 stall cycles per instruction
  - Total memory stall cycles = $10^6 \times 2.5$ = 2,500,000

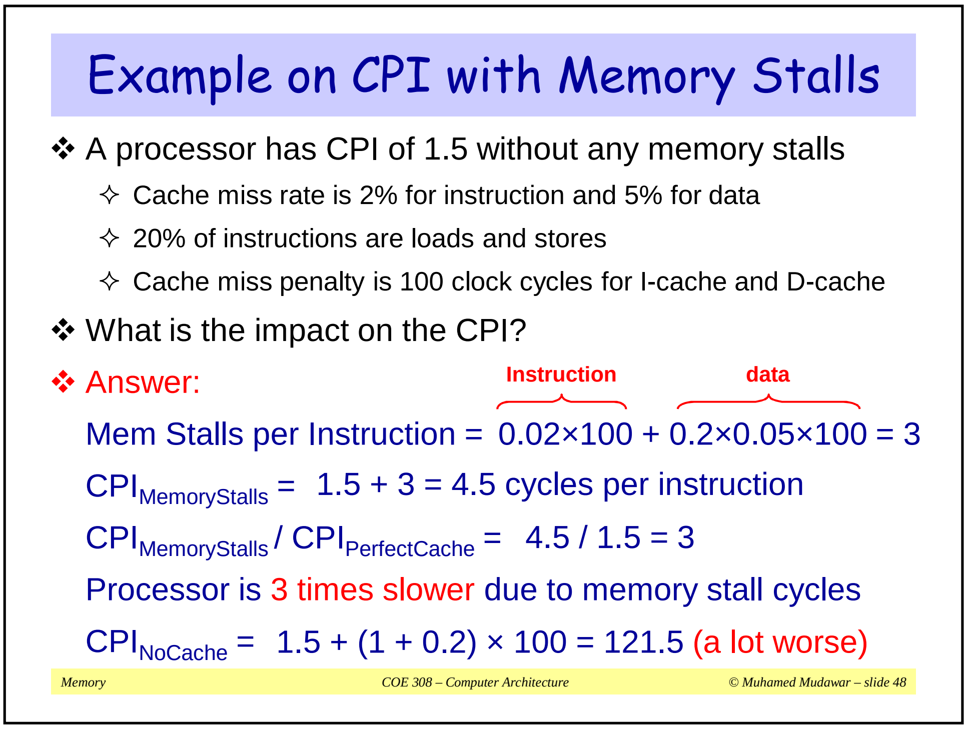


CPU Time with Memory Stall Cycles

$$\boxed{\text{CPU Time} = \text{I-Count} \times \text{CPI}_{\text{MemoryStalls}} \times \text{Clock Cycle}}$$

$$\boxed{\text{CPI}_{\text{MemoryStalls}} = \text{CPI}_{\text{PerfectCache}} + \text{Mem Stalls per Instruction}}$$

- $\text{CPI}_{\text{PerfectCache}}$ = CPI for ideal cache (no cache misses)
- $\text{CPI}_{\text{MemoryStalls}}$ = CPI in the presence of memory stalls
- Memory stall cycles increase the CPI



Example on CPI with Memory Stalls

- A processor has CPI of 1.5 without any memory stalls
  - Cache miss rate is 2% for instruction and 5% for data
  - 20% of instructions are loads and stores
  - Cache miss penalty is 100 clock cycles for I-cache and D-cache
- What is the impact on the CPI?
- Answer:

$$\text{Mem Stalls per Instruction} = \overbrace{0.02\times100}^{\text{Instruction}} + \overbrace{0.2\times0.05\times100}^{\text{data}} = 3$$

$\text{CPI}_{\text{MemoryStalls}}$ = 1.5 + 3 = 4.5 cycles per instruction

$\text{CPI}_{\text{MemoryStalls}} / \text{CPI}_{\text{PerfectCache}}$ = 4.5 / 1.5 = 3

Processor is 3 times slower due to memory stall cycles

$\text{CPI}_{\text{NoCache}}$ = 1.5 + (1 + 0.2) × 100 = 121.5 (a lot worse)

Average Memory Access Time

- ❖ Average Memory Access Time (AMAT)

  AMAT = Hit time + Miss rate × Miss penalty

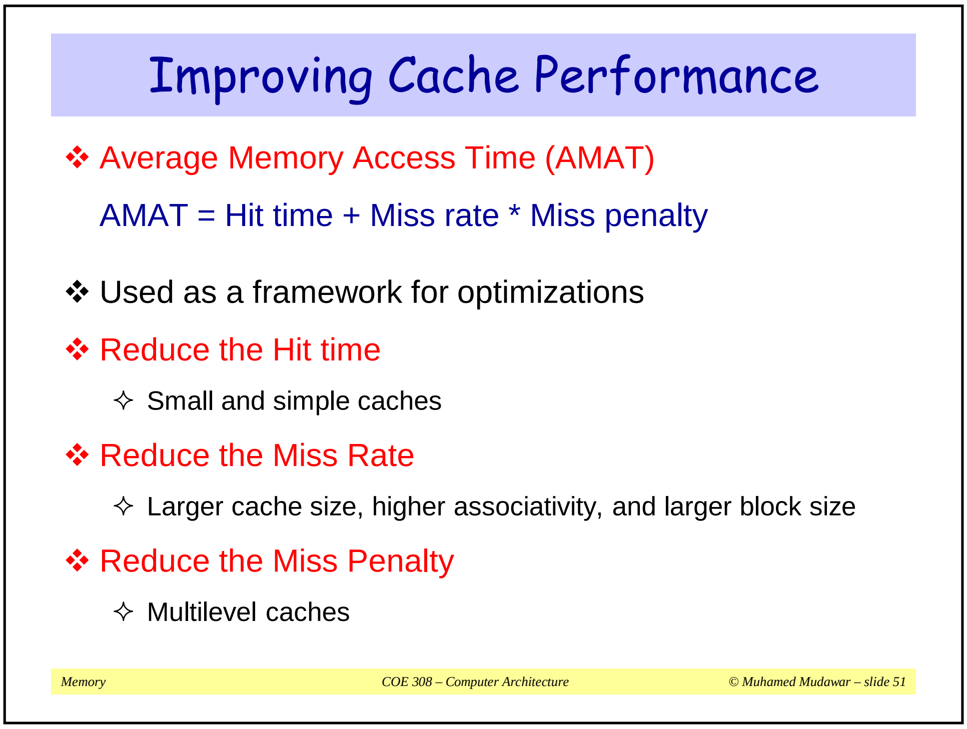
- ❖ Time to access a cache for both hits and misses
- ❖ Example: Find the AMAT for a cache with
  - ✧ Cache access time (Hit time) of 1 cycle = 2 ns
  - ✧ Miss penalty of 20 clock cycles
  - ✧ Miss rate of 0.05 per access
- ❖ Solution:

  AMAT = 1 + 0.05 × 20 = 2 cycles = 4 ns

  Without the cache, AMAT will be equal to Miss penalty = 20 cycles

Next . . .

- ❖ Random Access Memory and its Structure
- ❖ Memory Hierarchy and the need for Cache Memory
- ❖ The Basics of Caches
- ❖ Cache Performance and Memory Stall Cycles
- ❖ Improving Cache Performance
- ❖ Multilevel Caches

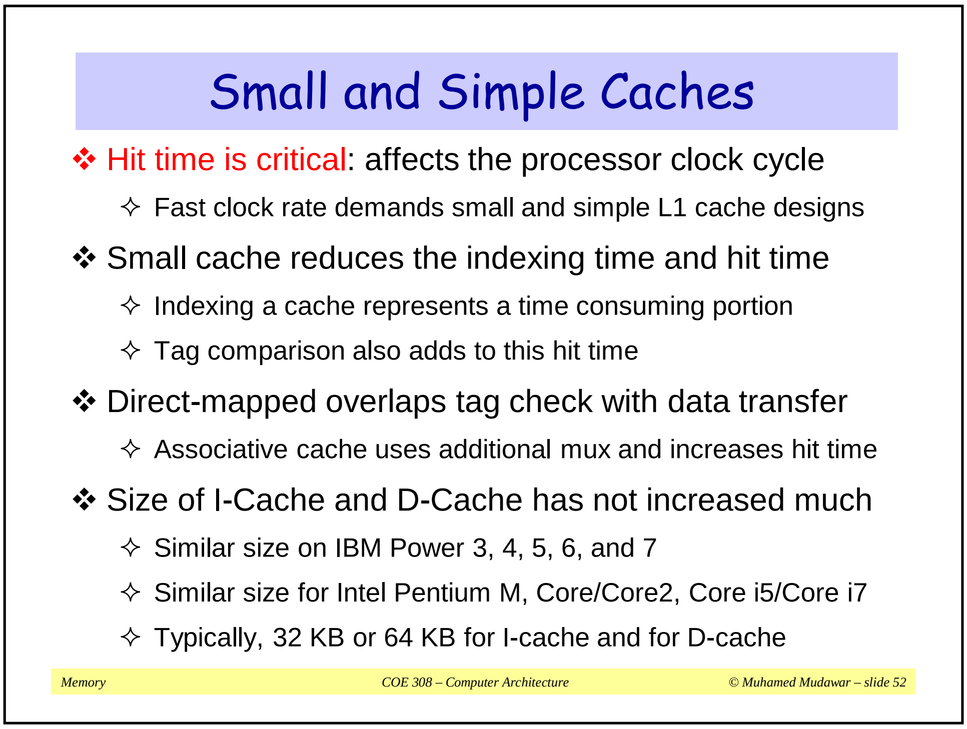


Improving Cache Performance

- Average Memory Access Time (AMAT)

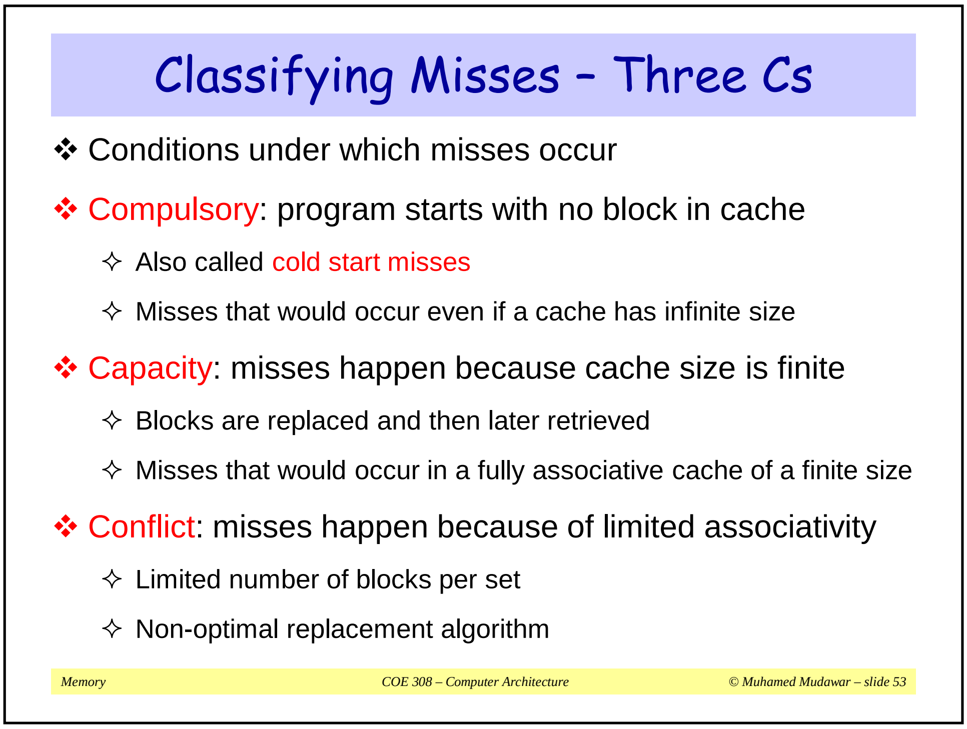
  AMAT = Hit time + Miss rate * Miss penalty

- Used as a framework for optimizations
- Reduce the Hit time
  - Small and simple caches
- Reduce the Miss Rate
  - Larger cache size, higher associativity, and larger block size
- Reduce the Miss Penalty
  - Multilevel caches



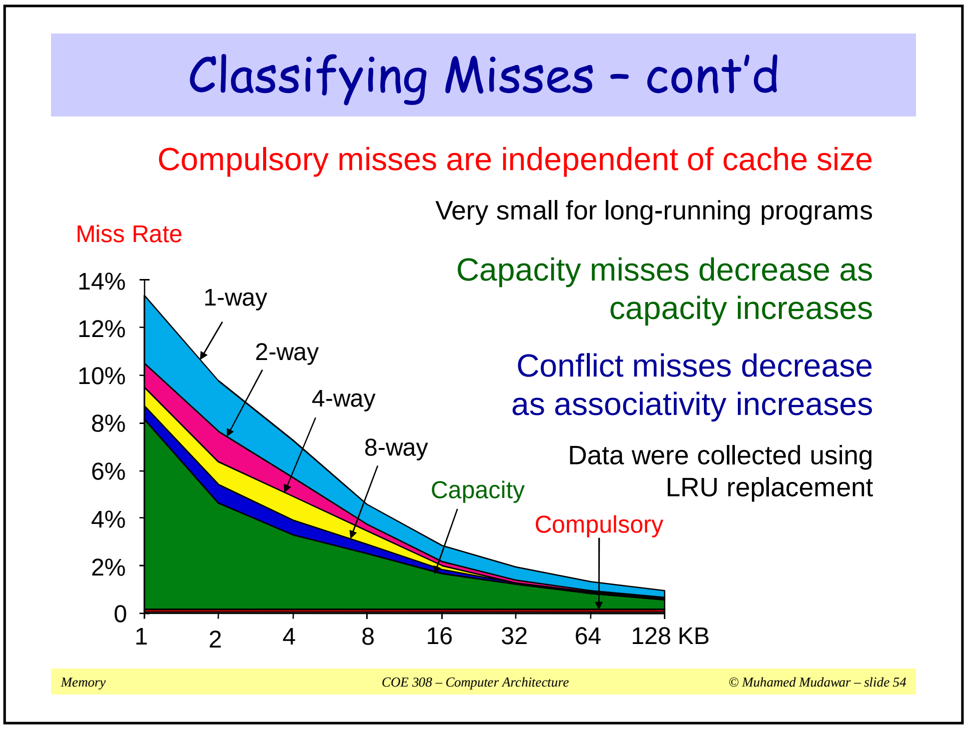
Small and Simple Caches

- Hit time is critical: affects the processor clock cycle
  - Fast clock rate demands small and simple L1 cache designs
- Small cache reduces the indexing time and hit time
  - Indexing a cache represents a time consuming portion
  - Tag comparison also adds to this hit time
- Direct-mapped overlaps tag check with data transfer
  - Associative cache uses additional mux and increases hit time
- Size of I-Cache and D-Cache has not increased much
  - Similar size on IBM Power 3, 4, 5, 6, and 7
  - Similar size for Intel Pentium M, Core/Core2, Core i5/Core i7
  - Typically, 32 KB or 64 KB for I-cache and for D-cache



Classifying Misses – Three Cs

- Conditions under which misses occur
- Compulsory: program starts with no block in cache
  - Also called cold start misses
  - Misses that would occur even if a cache has infinite size
- Capacity: misses happen because cache size is finite
  - Blocks are replaced and then later retrieved
  - Misses that would occur in a fully associative cache of a finite size
- Conflict: misses happen because of limited associativity
  - Limited number of blocks per set
  - Non-optimal replacement algorithm



Classifying Misses – cont'd

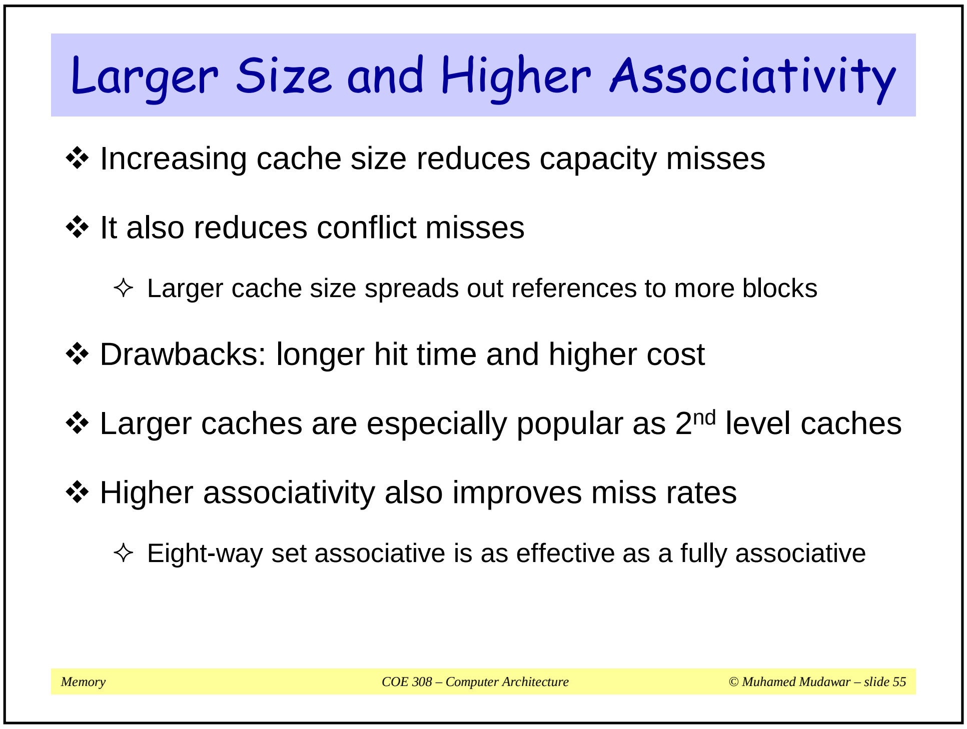
Compulsory misses are independent of cache size

Very small for long-running programs

Capacity misses decrease as capacity increases

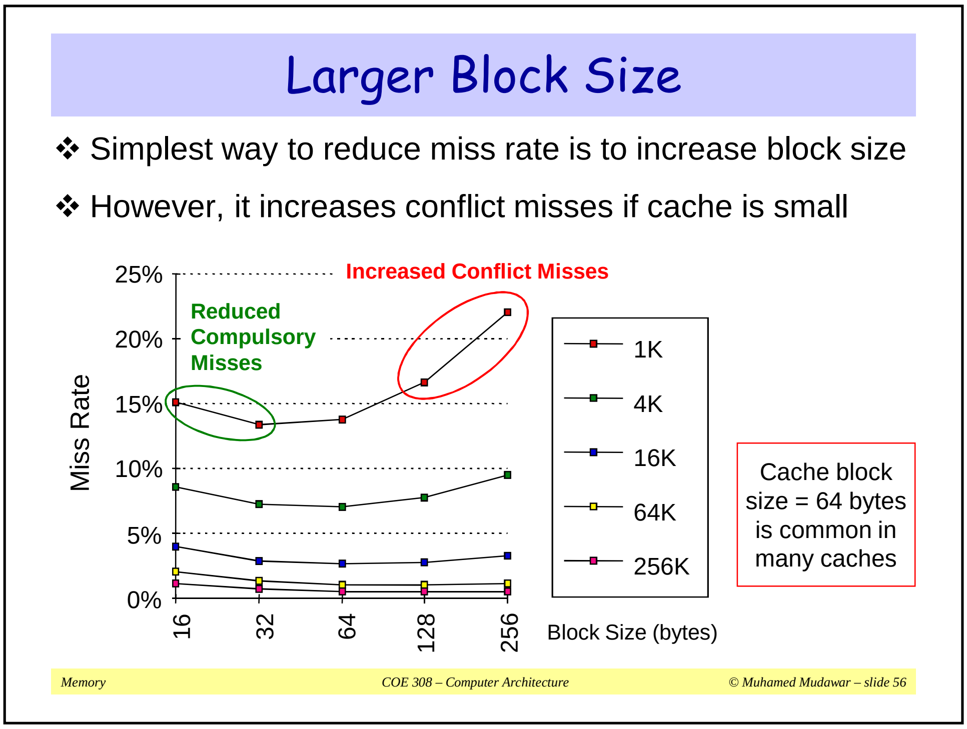
Conflict misses decrease as associativity increases

Data were collected using LRU replacement



Larger Size and Higher Associativity

- ❖ Increasing cache size reduces capacity misses
- ❖ It also reduces conflict misses
  - ✧ Larger cache size spreads out references to more blocks
- ❖ Drawbacks: longer hit time and higher cost
- ❖ Larger caches are especially popular as 2nd level caches
- ❖ Higher associativity also improves miss rates
  - ✧ Eight-way set associative is as effective as a fully associative



Larger Block Size

- ❖ Simplest way to reduce miss rate is to increase block size
- ❖ However, it increases conflict misses if cache is small

Increased Conflict Misses

Reduced Compulsory Misses

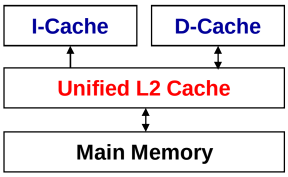
Cache block size = 64 bytes is common in many caches

Next . . .

- Random Access Memory and its Structure
- Memory Hierarchy and the need for Cache Memory
- The Basics of Caches
- Cache Performance and Memory Stall Cycles
- Improving Cache Performance
- Multilevel Caches

Multilevel Caches

- Top level cache should be kept small to
  - Keep pace with processor speed
- Adding another cache level
  - Can reduce the memory gap
  - Can reduce memory bus loading



- Local miss rate
  - Number of misses in a cache / Memory accesses to this cache
  - Miss Rate_{L1} for L1 cache, and Miss Rate_{L2} for L2 cache
- Global miss rate

  Number of misses in a cache / Memory accesses generated by CPU

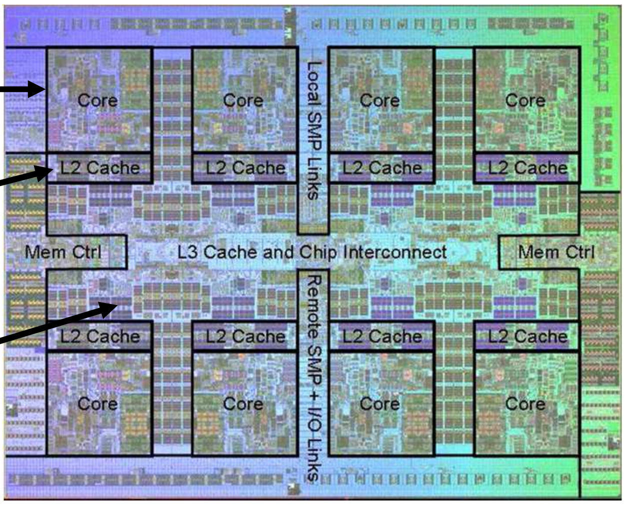
  Miss Rate_{L1} for L1 cache, and Miss Rate_{L1} × Miss Rate_{L2} for L2 cache

Power 7 On-Chip Caches [IBM 2010]

32KB I-Cache/core
32KB D-Cache/core
3-cycle latency

256KB Unified
L2 Cache/core
8-cycle latency

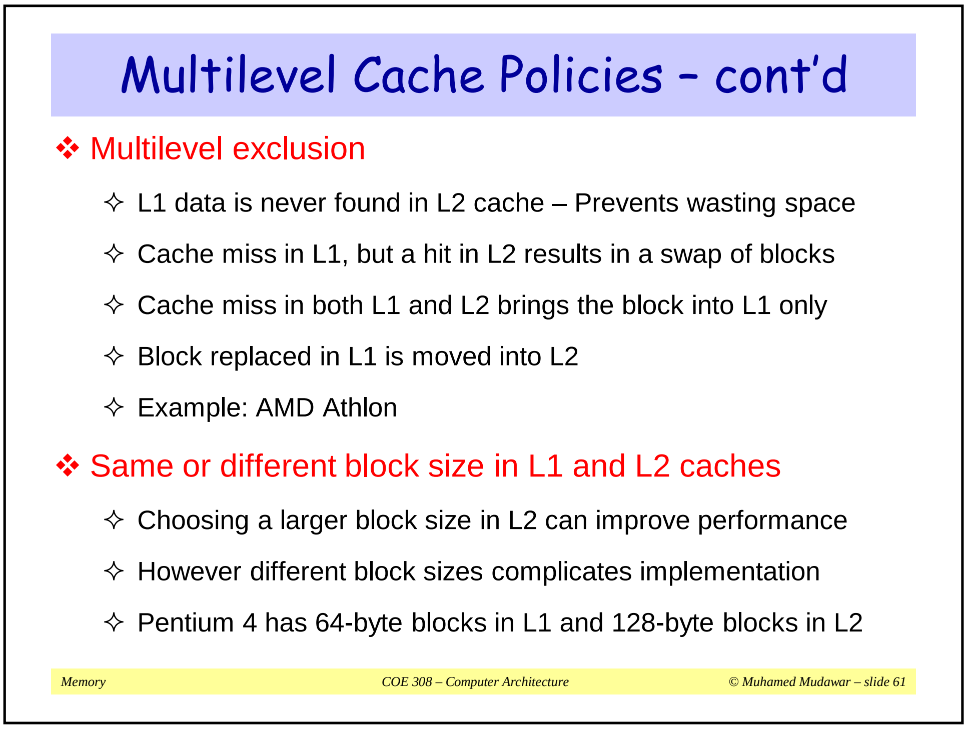
32MB Unified
Shared L3 Cache
Embedded DRAM
25-cycle latency
to local slice



Multilevel Cache Policies

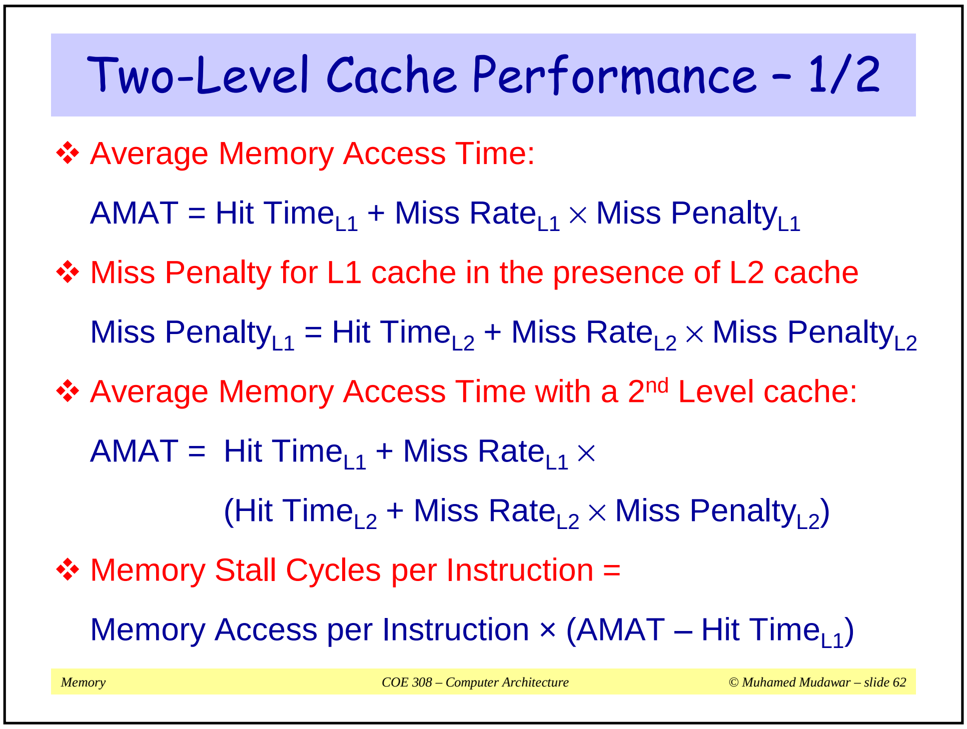
❖ Multilevel Inclusion

- L1 cache data is always present in L2 cache
- A miss in L1, but a hit in L2 copies block from L2 to L1
- A miss in L1 and L2 brings a block into L1 and L2
- A write in L1 causes data to be written in L1 and L2
- Typically, write-through policy is used from L1 to L2
- Typically, write-back policy is used from L2 to main memory
  - To reduce traffic on the memory bus
- A replacement or invalidation in L2 must be propagated to L1



Multilevel Cache Policies – cont'd

- Multilevel exclusion
  - L1 data is never found in L2 cache – Prevents wasting space
  - Cache miss in L1, but a hit in L2 results in a swap of blocks
  - Cache miss in both L1 and L2 brings the block into L1 only
  - Block replaced in L1 is moved into L2
  - Example: AMD Athlon
- Same or different block size in L1 and L2 caches
  - Choosing a larger block size in L2 can improve performance
  - However different block sizes complicates implementation
  - Pentium 4 has 64-byte blocks in L1 and 128-byte blocks in L2



Two-Level Cache Performance – 1/2

- Average Memory Access Time:

  AMAT = Hit Time_{L1} + Miss Rate_{L1} × Miss Penalty_{L1}

- Miss Penalty for L1 cache in the presence of L2 cache

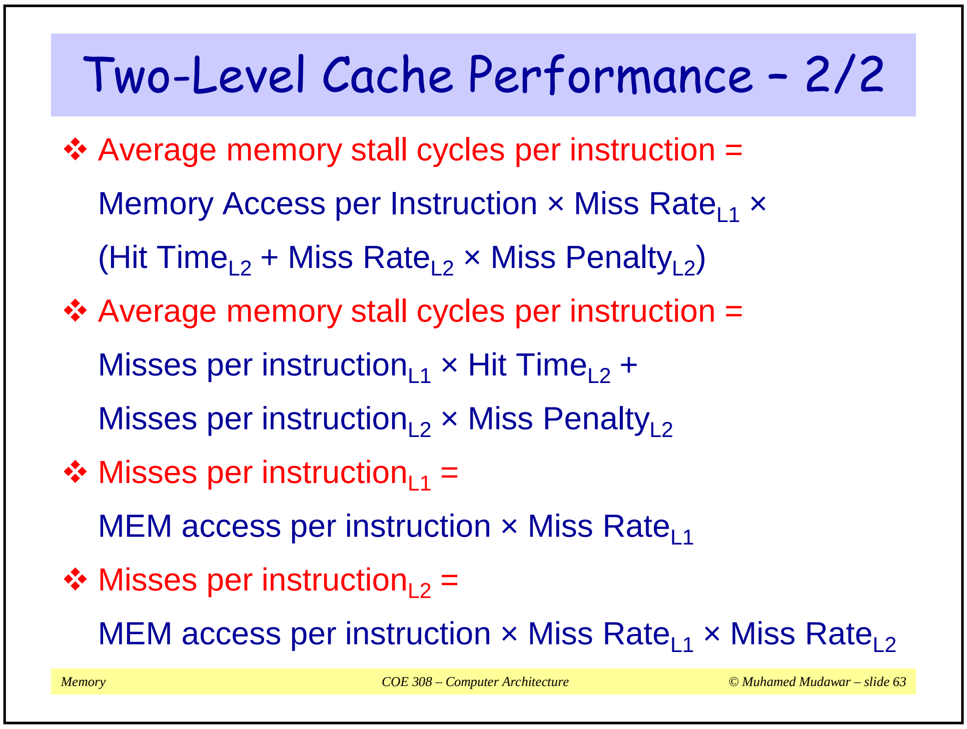
  Miss Penalty_{L1} = Hit Time_{L2} + Miss Rate_{L2} × Miss Penalty_{L2}

- Average Memory Access Time with a 2nd Level cache:

  AMAT = Hit Time_{L1} + Miss Rate_{L1} ×
  (Hit Time_{L2} + Miss Rate_{L2} × Miss Penalty_{L2})

- Memory Stall Cycles per Instruction =

  Memory Access per Instruction × (AMAT – Hit Time_{L1})



Two-Level Cache Performance – 2/2

- ❖ Average memory stall cycles per instruction =

  Memory Access per Instruction × Miss Rate$_{L1}$ ×

  (Hit Time$_{L2}$ + Miss Rate$_{L2}$ × Miss Penalty$_{L2}$)

- ❖ Average memory stall cycles per instruction =

  Misses per instruction$_{L1}$ × Hit Time$_{L2}$ +

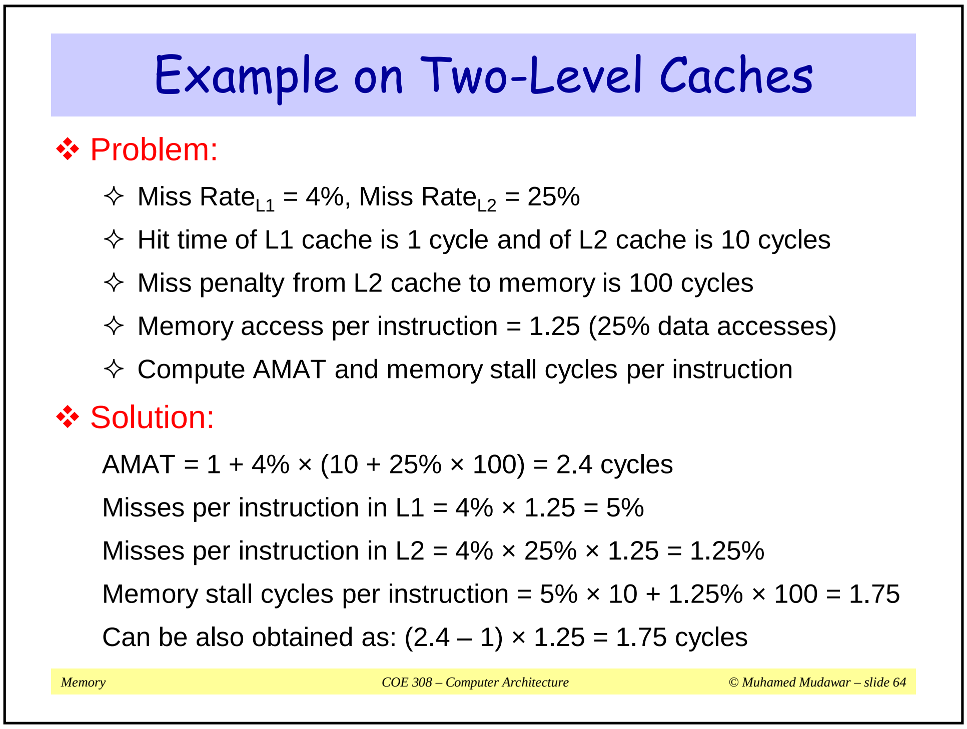
  Misses per instruction$_{L2}$ × Miss Penalty$_{L2}$

- ❖ Misses per instruction$_{L1}$ =

  MEM access per instruction × Miss Rate$_{L1}$

- ❖ Misses per instruction$_{L2}$ =

  MEM access per instruction × Miss Rate$_{L1}$ × Miss Rate$_{L2}$



Example on Two-Level Caches

- ❖ Problem:
  - ✧ Miss Rate$_{L1}$ = 4%, Miss Rate$_{L2}$ = 25%
  - ✧ Hit time of L1 cache is 1 cycle and of L2 cache is 10 cycles
  - ✧ Miss penalty from L2 cache to memory is 100 cycles
  - ✧ Memory access per instruction = 1.25 (25% data accesses)
  - ✧ Compute AMAT and memory stall cycles per instruction
- ❖ Solution:

  AMAT = 1 + 4% × (10 + 25% × 100) = 2.4 cycles

  Misses per instruction in L1 = 4% × 1.25 = 5%

  Misses per instruction in L2 = 4% × 25% × 1.25 = 1.25%

  Memory stall cycles per instruction = 5% × 10 + 1.25% × 100 = 1.75

  Can be also obtained as: (2.4 – 1) × 1.25 = 1.75 cycles