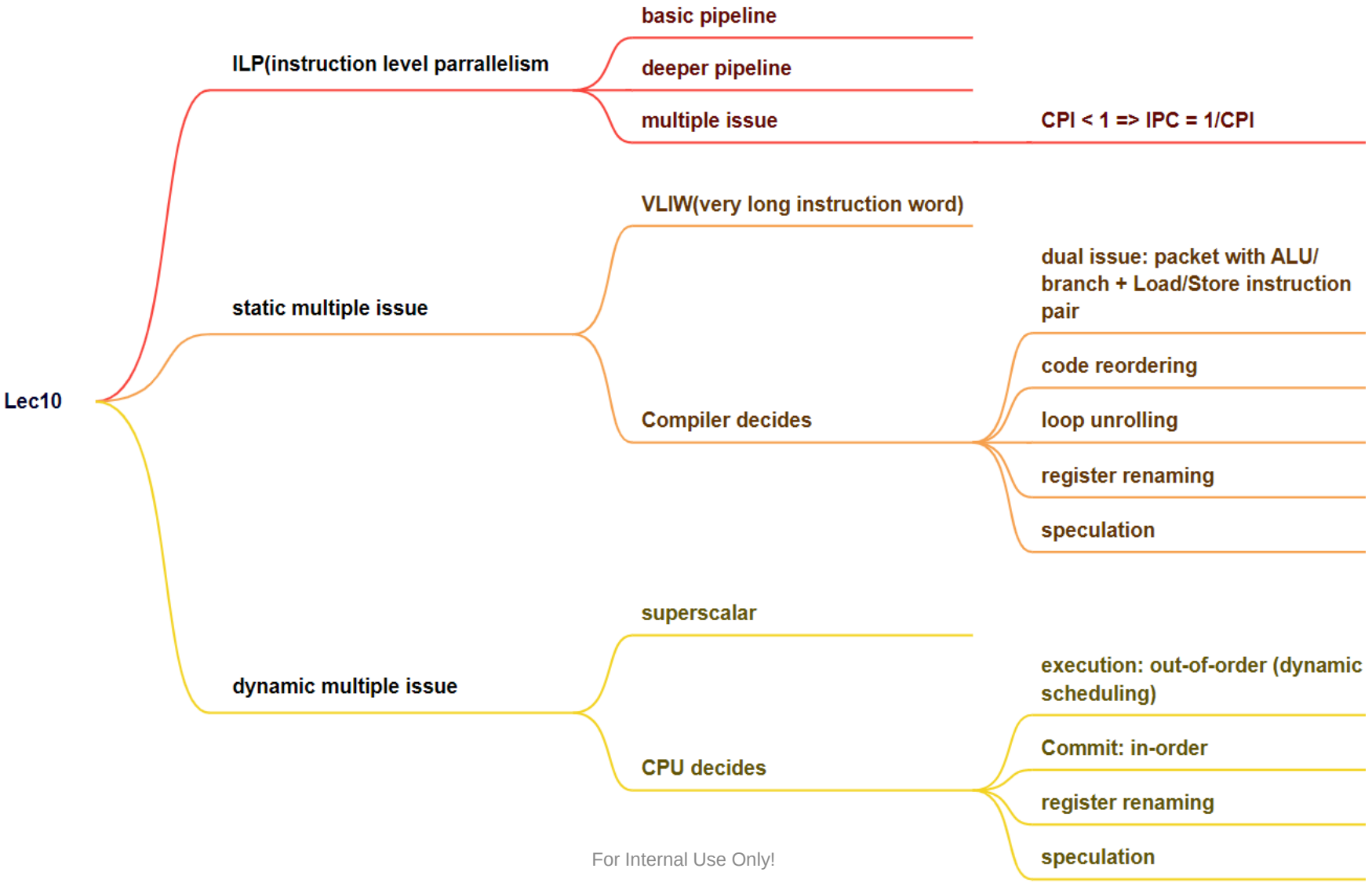
COMPUTER ORGANIZATION

Lecture 11
Memory Hierarchy (1)

2024 Spring

Recap



Chapter 5 Overview

- Storage Technology
  - Memory hierarchy
  - Principle of locality
- Cache(/ˈkæʃ/)
  - The basics of caches
  - Measuring cache performance
  - Improving cache performance
- Virtual memory
- Dependability
  - Hamming code

Storage Technology

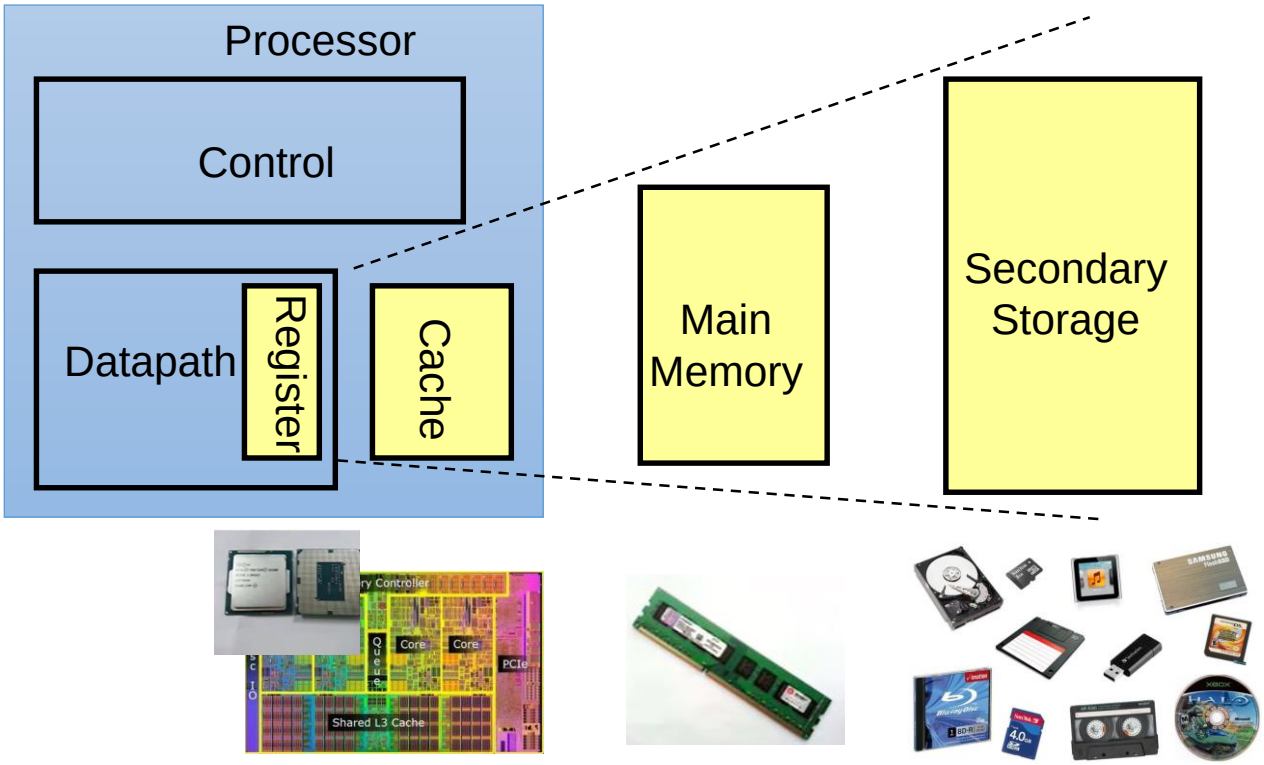
- Random access memory: Access time same for all locations
  - SRAM: Static Random Access Memory (caches)
    - Low density, high power, expensive, fast
    - Static: content will last (forever until lose power)
  - DRAM: Dynamic Random Access Memory (main memory)
    - High density, low power, cheap, slow
    - Dynamic: need to be refreshed regularly
    - Double data rate (DDR) DRAM (Transfer on rising/falling edges)
- Flash
- Magnetic disk

Ideal memory
- Access time of SRAM
- Capacity and cost/GB of disk

Memory technology	Typical access time	$ per GiB in 2012
SRAM semiconductor memory	0.5–2.5 ns	$500–$1000
DRAM semiconductor memory	50–70 ns	$10–$20
Flash semiconductor memory	5,000–50,000 ns	$0.75–$1.00
Magnetic disk	5,000,000–20,000,000 ns	$0.05–$0.10

Memory Hierarchy

- An Illusion of a large, fast, cheap memory
  - Fact: Large memories slow, fast memories small
  - How to achieve: hierarchy, parallelism
- An expanded view of memory system:



Why Hierarchy Works?

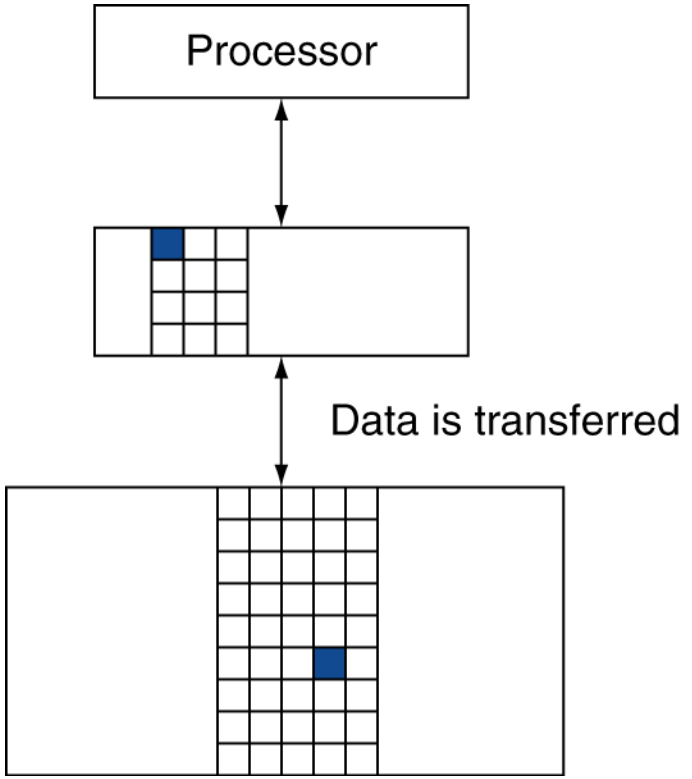
- Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time
    - Loops
    - Arrays
    - Sequential instructions
  - 90/10 rule: 10% of code executed 90% of time
- Two types of locality:
  - Temporal locality: if an item is referenced, it will tend to be referenced again soon
  - Spatial locality: if an item is referenced, items whose addresses are close by tend to be referenced soon

A widely held rule of thumb is that a program spends 90% of its execution time in only 10% of the code.

—— Hennessy & Patterson. *Computer Architecture, A quantitative Approch*

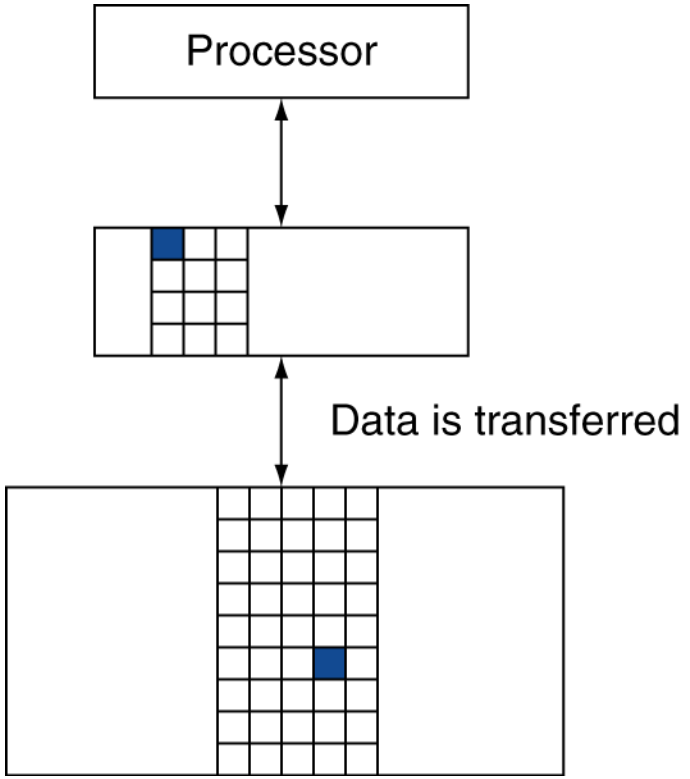
Memory Hierarchy: Principle

- At any given time, data is copied between only two adjacent levels:
  - Upper level: the one closer to the processor
    - Smaller, faster, uses more expensive technology
  - Lower level: the one away from the processor
    - Bigger, slower, uses less expensive technology
- Block: basic unit of information transfer
  - Minimum unit of information that can either be present or not present in a level of the hierarchy

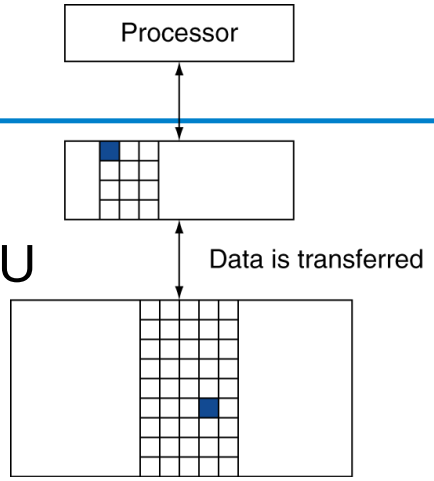


Memory Hierarchy: Terminology

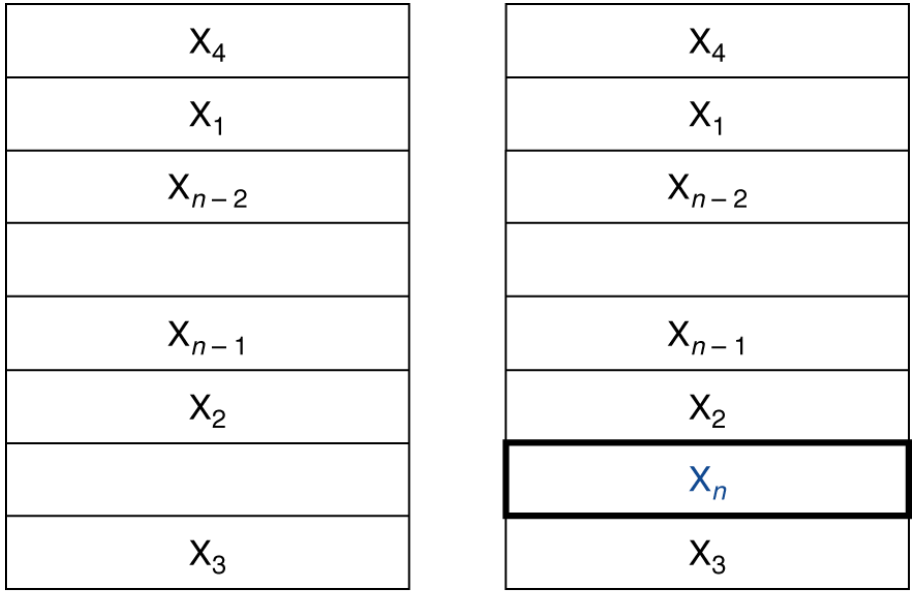
- Block (or line): unit of copying
  - May be multiple words
- Hit: data appears in upper level
  - Hit rate(ratio): hits/accesses
  - Hit time: time to access the upper level
- Miss: data needs to be retrieved from a block in the lower level
  - Miss rate(ratio) = misses/accesses = 1 - hit rate
  - Miss Penalty: time to fetch a block into a level of the memory hierarchy from the lower level
- Hit Time << Miss Penalty



Cache Memory



- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses X_1, …, X_{n-1}
  - and how about X_n



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

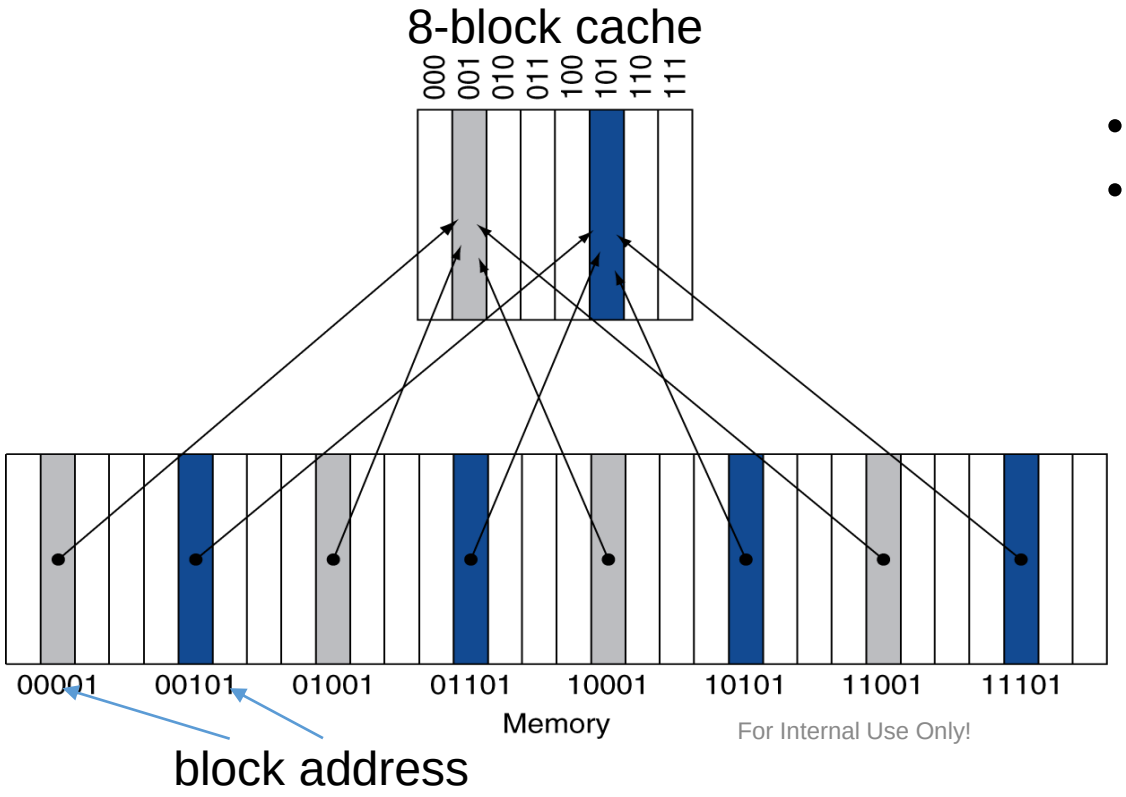
cache content example

Questions for Hierarchy Design

Q1: Where can a block be placed in the upper level?
=> *block placement*
Q2: How is a block found if it is in the upper level?
=> *block finding*
Q3: Which block should be replaced on a miss?
=> *block replacement*
Q4: What happens on a write?
=> *write policy*

Direct Mapped Cache

- Block Placement: Location determined by address
- One data in memory is mapped to only one location in cache
  - location = (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Modulo is computed using low-order address bits

e.g.: block address 29_{ten} (11101_{two}) in the momory is mapped to 5_{ten}(101_{two}) in the cache
29 mod 8 → 5
11101 mod 8 → 101

Tags and Valid Bits

- Block Finding: 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

Cache Example

- 8-blocks, 1 byte/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		

Cache Example

Decimal 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		

Cache Example

Decimal 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		

Cache Example

Decimal 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		

Cache Example

Decimal 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		

Cache Example

Decimal addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010
16	10 000	Hit	000

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	~~11~~ **10**	~~Mem[11010]~~ **Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

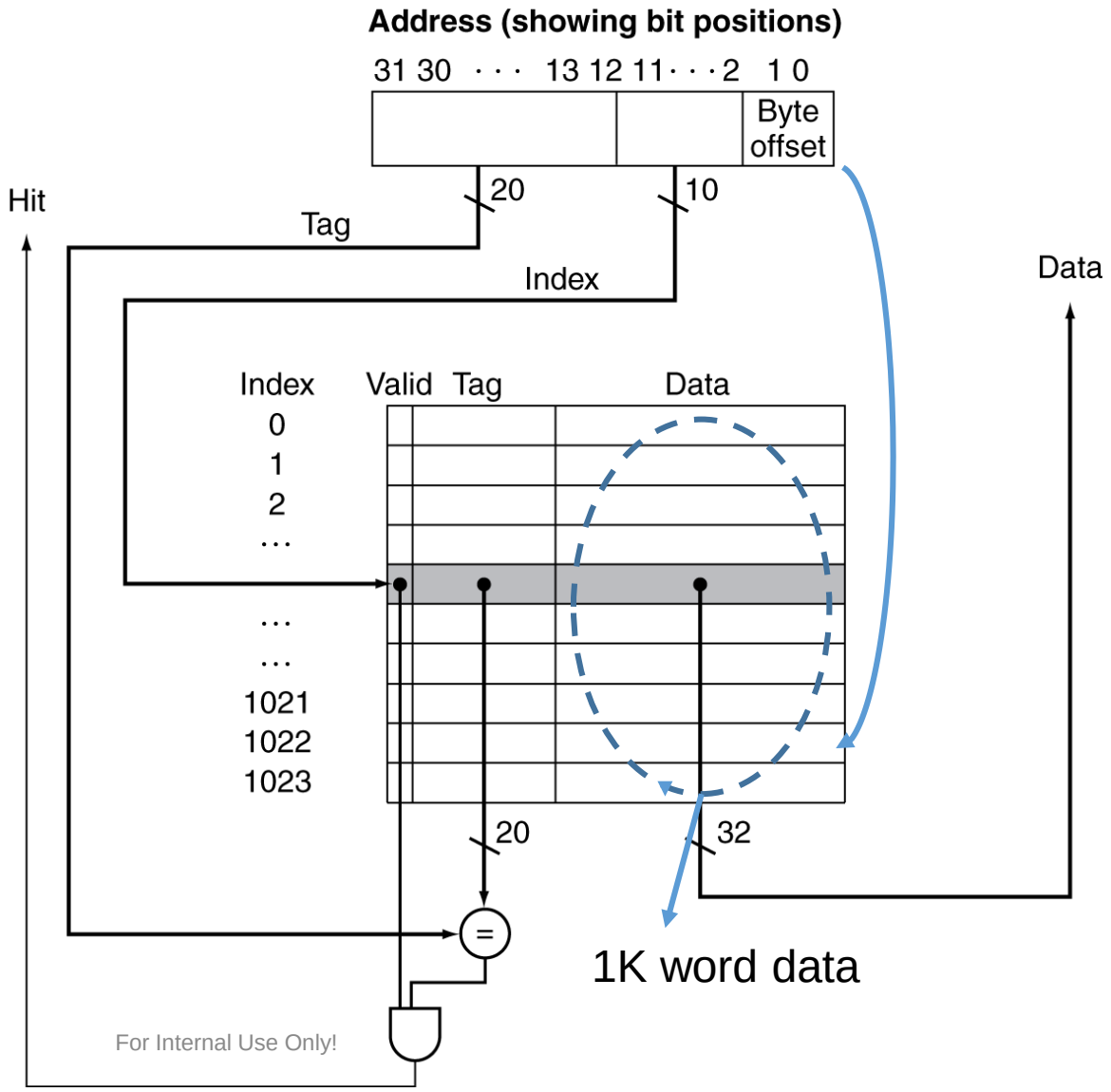
Cache Example

- 8-blocks, 1 byte/block, direct mapped
- Sequence of 9 memory references to an empty 8-block cache

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss	$(10010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$

Address Subdivision

- 1K blocks,
  1-word block:
  - Bytes offset within a block: 2 bits
  - Cache index: $\log_2(1K) = 10$ bits
  - Cache tag: 32-10-2 = 20 bits
  - Valid bit (When start up, valid is 0)

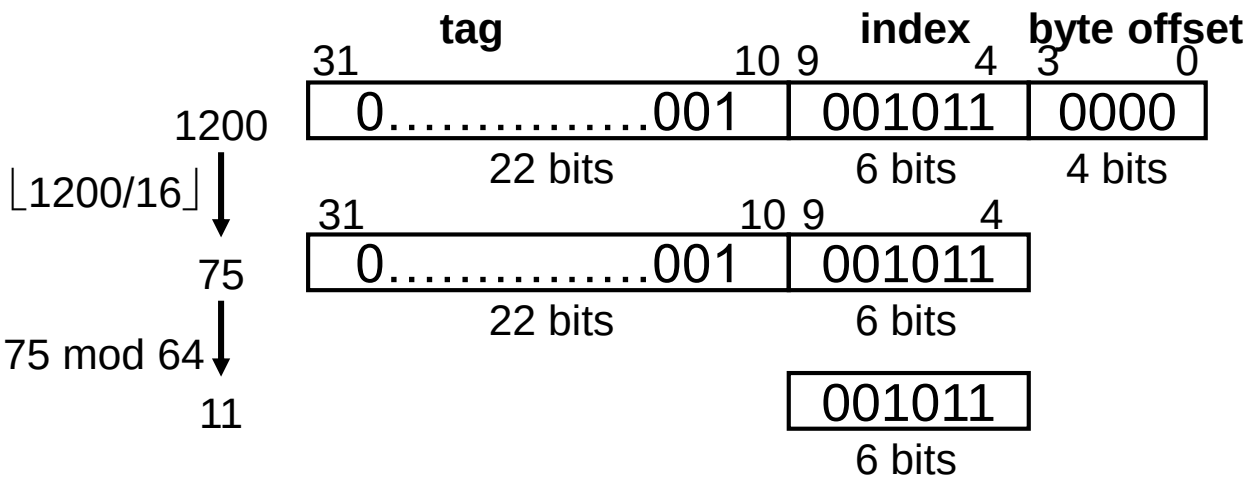


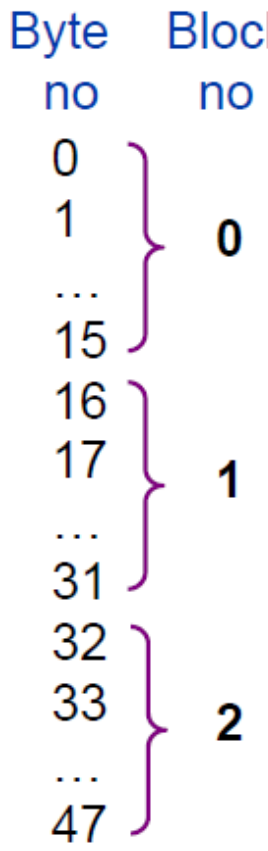
Example: Larger Block Size

- 64 blocks, 16 bytes/block
  - To what block number does address 1200 map?
- 1200_{ten}=0…1001011**0000**$_{two}$

tag **index** byte offset

32-6-4 = 22 64 = 2^6 16 = 2^4





- Or calculate using decimal:
  - Block address = ⌊1200/16⌋ = 75
  - Block number(index) = 75 modulo 64 = 11

Example: Bits required for a Cache

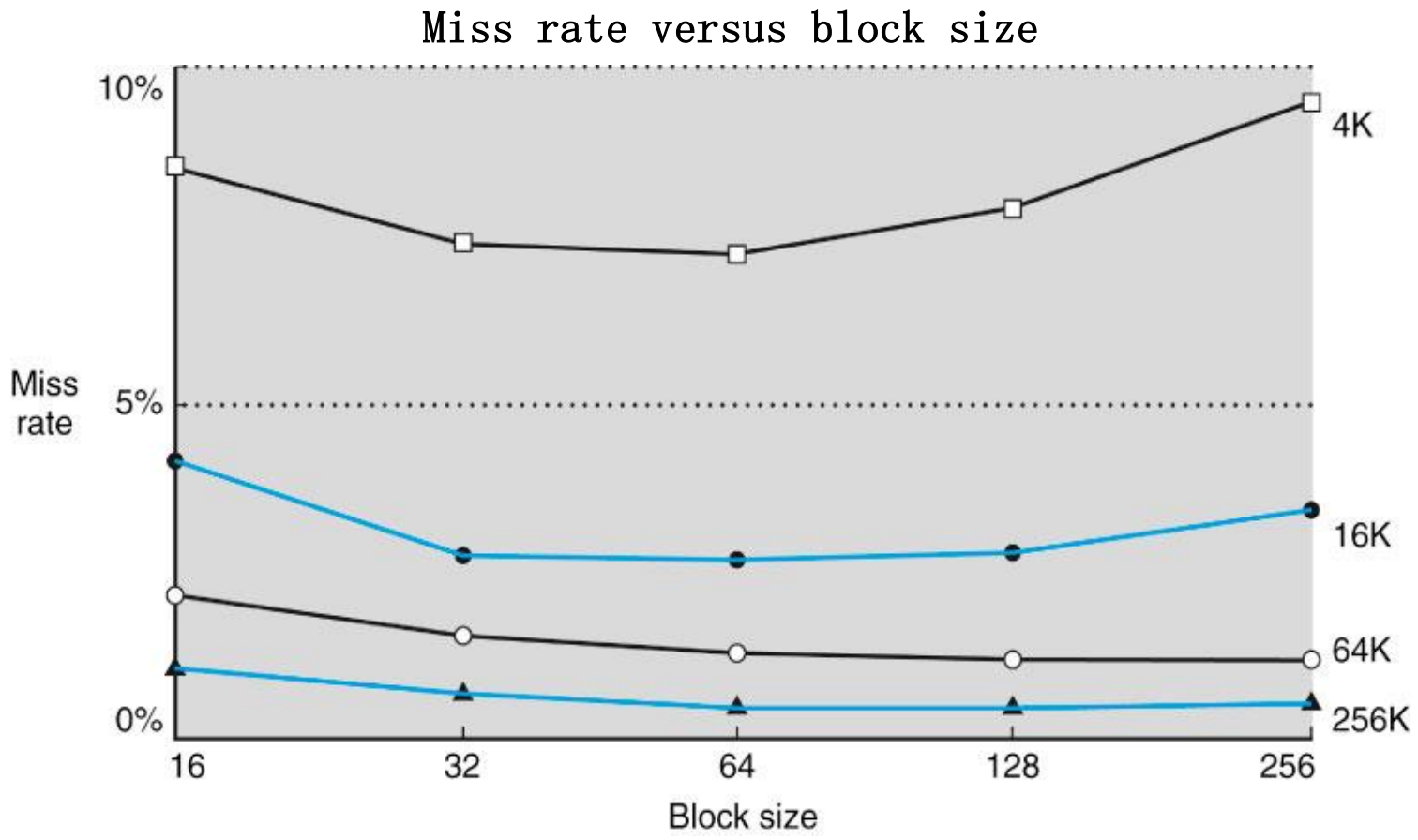
- 16 KB of data (cache capacity), 4-word blocks, assume a 32-bit address
  - Each block has 16Byte → Byte offset within a block: 4 bits
  - Total no of blocks = 16KB cache data size/16Byte→ = 1K → Index: 10 bits
  - Tag: 32-10-4 = 18 bits
  - Valid: 1 bit
  - Total cache size: 2^{10} x (4 x 32 + 18 + 1) = 2^{10} x 147 = 147 Kb = 18.375KB

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
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help

Block Size Considerations

- Increase block size tends to decrease miss rate



Cache Misses

- Miss categories
  - Compulsory misses (cold-start misses)
  - Capacity misses
  - Conflict misses
- Read Hit and Miss:
  - 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

Write Hit: 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 = Base CPI + Memory-stall cycles/instruction

      $=1 + 0.1 \times 100 = 11$
- 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 Hit: 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

Write Miss: 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 (no write allocate)
    - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
  - Allocate on miss: fetch the block (write allocate)

Write Policies Summary

- If the block is found in the cache
  - Update that block, Then…
  - Update immediately → write-through policy
  - or, wait until kicking block out → write-back policy
- If the block is not in the cache
  - Allocate a block in the cache → write allocate policy
  - or, Write directly to memory without allocation → no write allocate policy

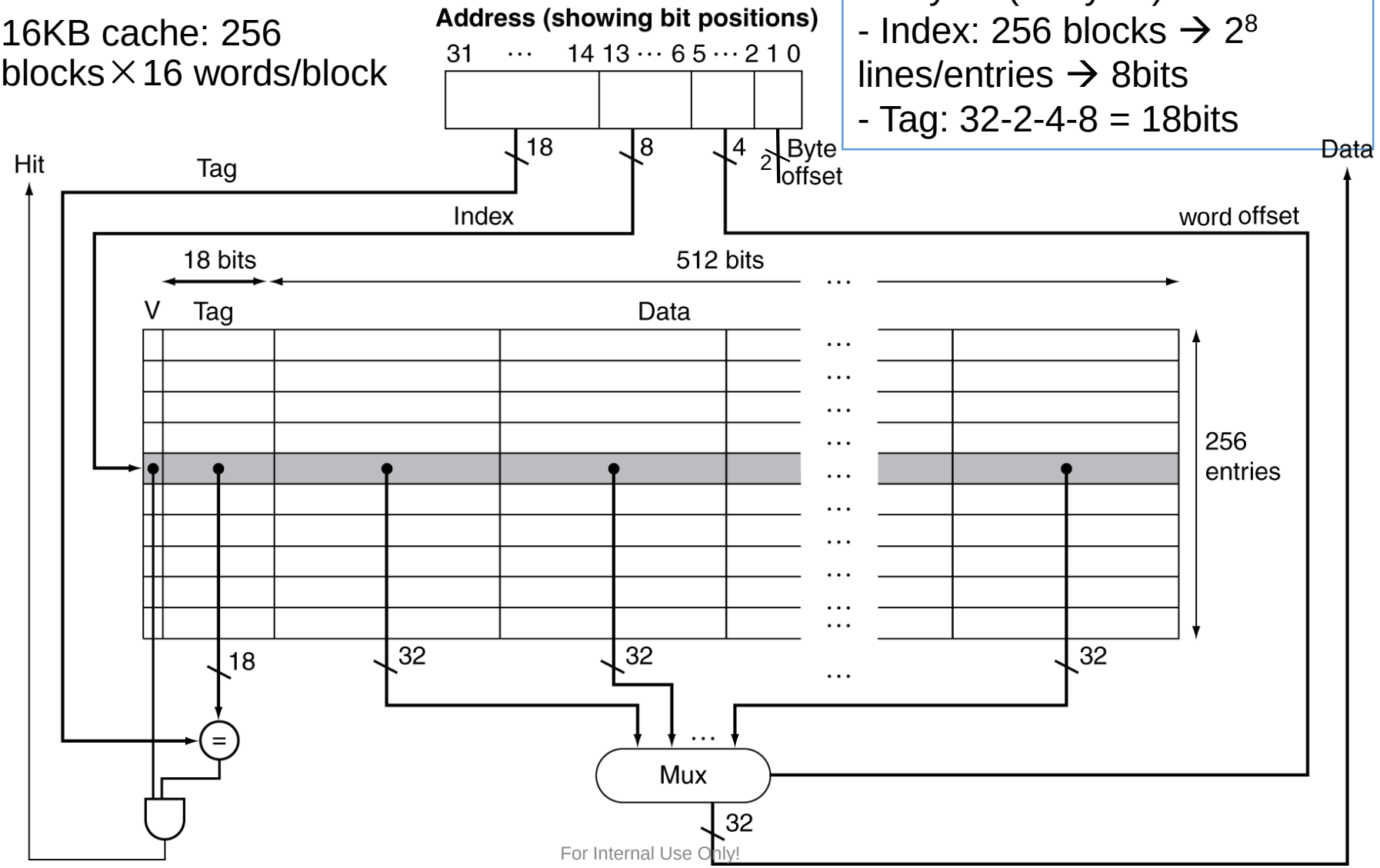
Intrinsity FastMATH

- Embedded RISC-V processor
  - 12-stage pipeline
  - Instruction and data access on each cycle
- Split cache: separate I-cache and D-cache
  - Each 16KB: 256 blocks × 16 words/block
  - D-cache: write-through or write-back
- SPEC2000 miss rates
  - I-cache: 0.4%
  - D-cache: 11.4%
  - Weighted average: 3.2%

Example: Intrinsity FastMATH Processor

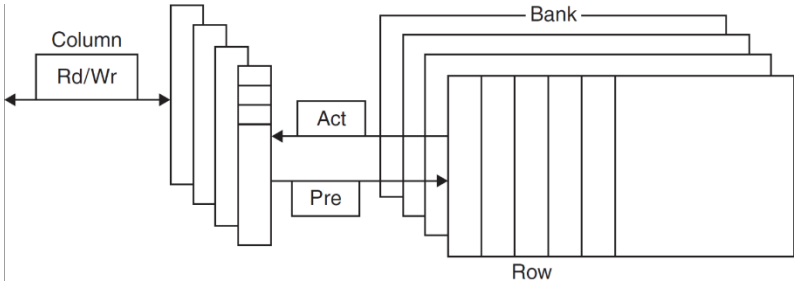
- each line: 16 words data → 64 bytes (2^6 bytes) → 512 bits
- Index: 256 blocks → 2^8 lines/entries → 8bits
- Tag: 32-2-4-8 = 18bits

16KB cache: 256 blocks × 16 words/block



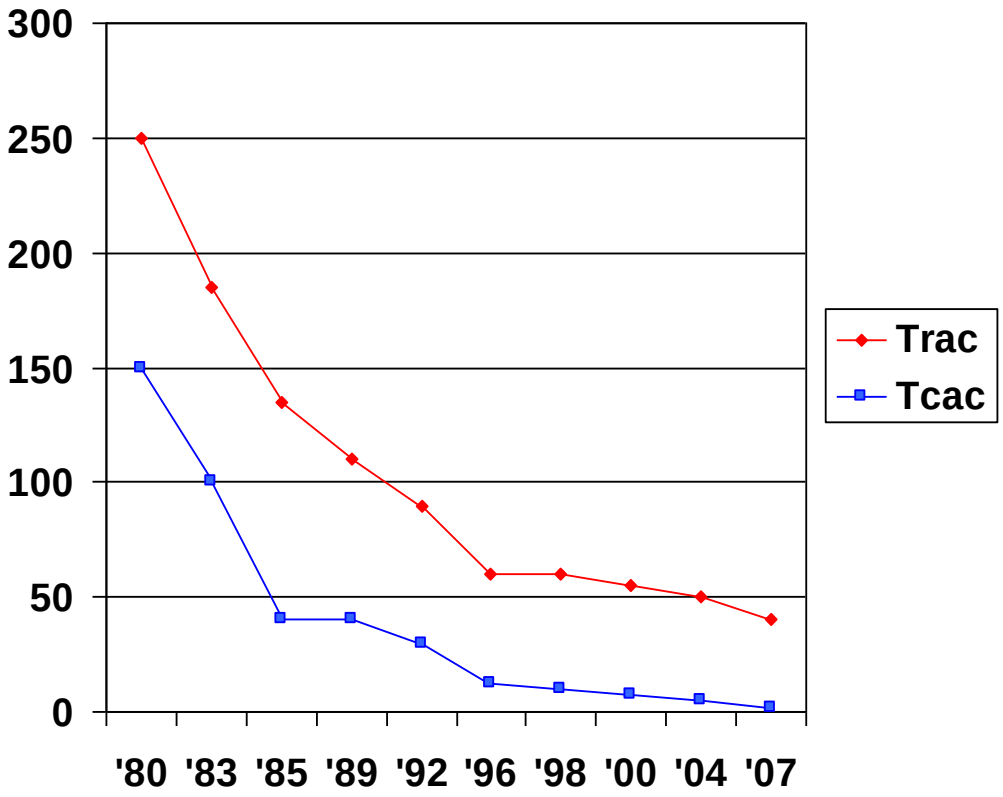
DRAM Technology/Organization

- Data stored as a charge in a capacitor
  - Single transistor used to access the charge
  - Must periodically be refreshed
    - Read contents and write back
    - Performed on a DRAM “row”
- 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



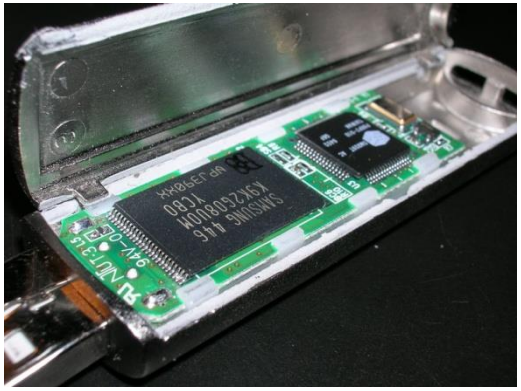
DRAM Generations

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



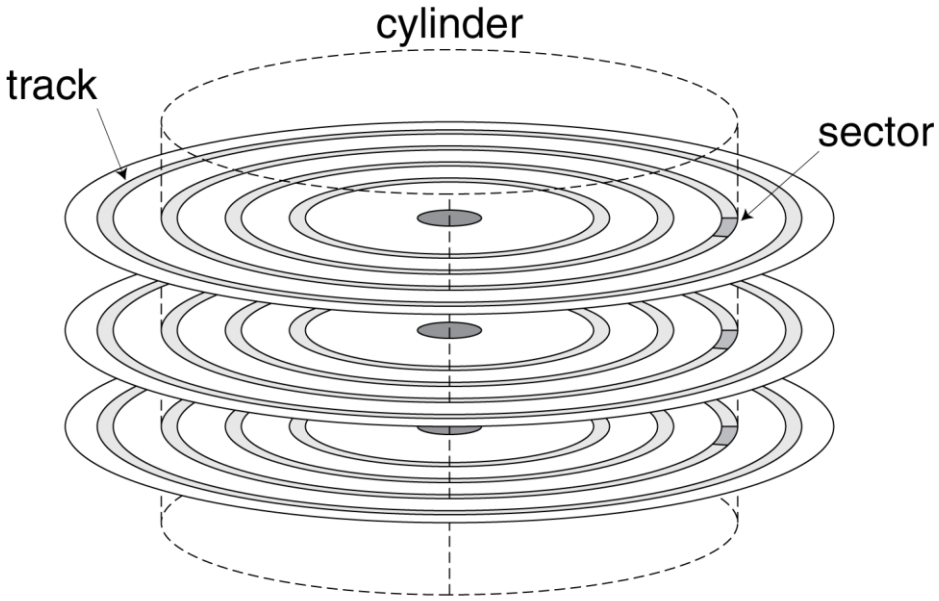
Flash Storage

- Nonvolatile semiconductor storage
  - 100× – 1000× faster than disk
  - Smaller, lower power, more robust
  - But more $/GB (between disk and DRAM)
- Flash bits wears out after 1000's of accesses
  - Not suitable for direct RAM or disk replacement
  - Wear leveling: remap data to less used blocks



Disk Storage

- Nonvolatile, rotating magnetic storage



Disk Sectors and Access

- Each sector records
  - Sector ID
  - Data (512 bytes, 4096 bytes proposed)
  - Error correcting code (ECC)
    - Used to hide defects and recording errors
  - Synchronization fields and gaps
- Access to a sector involves
  - Queuing delay if other accesses are pending
  - Seek: move the heads
  - Rotational latency
  - Data transfer
  - Controller overhead