CSCI 36000/73000

Fall 2024

Course personnel information

- Instructor:
  - Sourya Saha
    - Email: ssaha2@gradcenter.cuny.edu
    - Class hours: Wednesday 5:30 – 8:15 PM, HN C107
    - Office hours: Prior slot reservation is advised
    - Course resources: Blackboard

Course structure

- **Lectures:** The lectures will involve both slides and blackboard discussions. Attending all the lectures will be very important for the students to develop the concepts and skills, and also to be able to perform well in the exams, assignments, and quizzes.

- **Homework (20%):** Written homework will be assigned throughout the semester. Assignments will be due at the beginning of the lecture on the dates announced. It is your responsibility to keep track of assignments and their due dates.

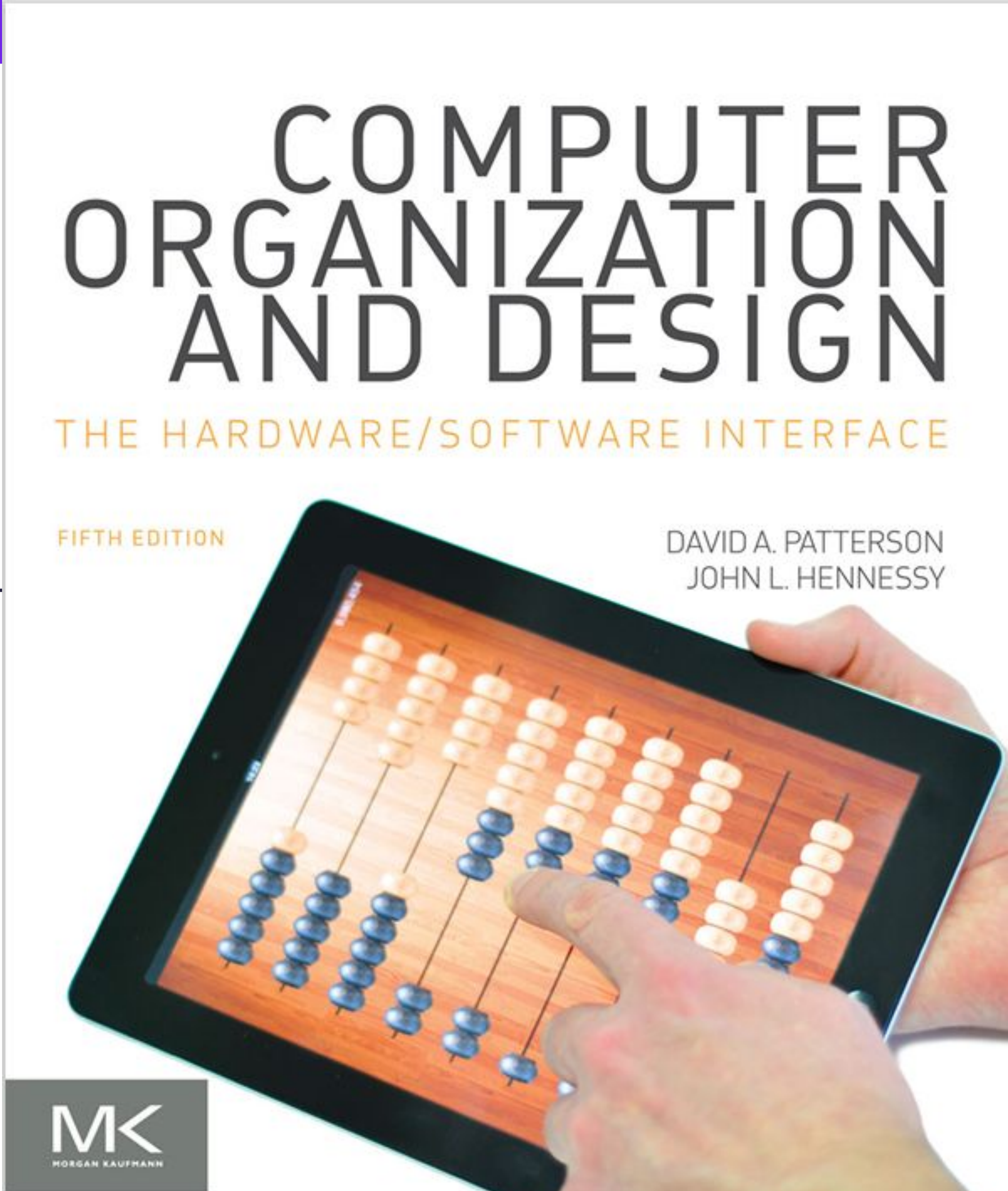
- **Quizzes (30%):** There will also be 4 - 5 unannounced pop quizzes during class hours containing 10-15 multiple choice questions. There will be no retake of the quizzes.

- **Tests (50%):** There will be two exams.
  - Exam 1 – To be decided
  - Exam 2 – Finals week.

TOPIC I: MEMORY ARCHITECTURE

Slides adapted from Patterson-Hennessy (Chapter 5)

Slides adapted from Patterson-Hennessy (Chapter 5)



Section goals

- Identify the memory technologies found in a computer and be aware of the way in which memory technology is changing.
- Appreciate that most data on the memory bus is cache refill traffic
- Describe the various ways of organizing cache memory and appreciate the cost-performance trade-offs for each arrangement.
- Appreciate the need for cache coherency in multiprocessor systems

Memory Technology

- Random Access:
  - "Random" is good: access time is the same for all locations
  - DRAM: Dynamic Random Access Memory
    - High density, low power, cheap, slow
    - Dynamic: need to be "refreshed" regularly
  - SRAM: Static Random Access Memory
    - Low density, high power, expensive, fast
    - Static: content will last "forever"(until lose power)
- "Non-so-random" Access Technology:
  - Access time varies from location to location and from time to time
  - Examples: Disk, CDROM
- Sequential Access Technology: access time linear in location (e.g.,Tape)

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

Main Memory Background

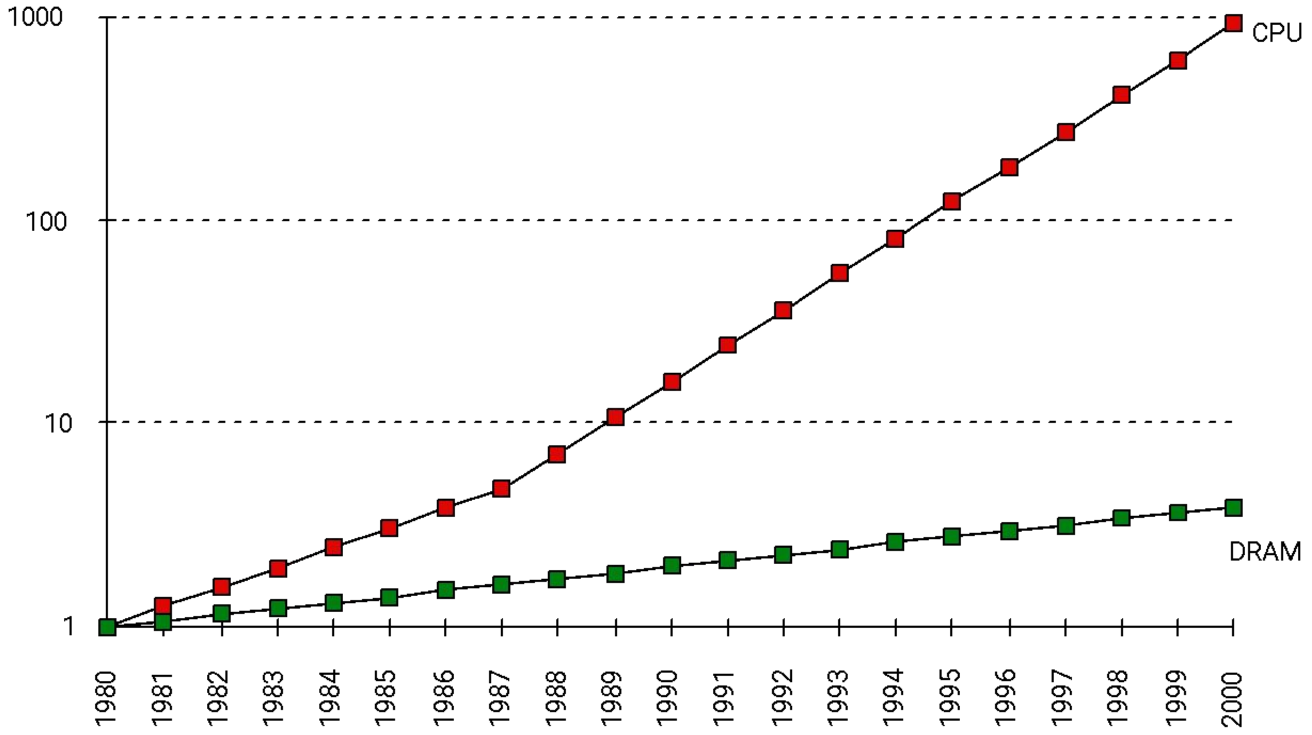
- Main Memory is *DRAM :* Dynamic Random Access Memory
  - Dynamic since needs to be refreshed periodically (8 ms)
  - Addresses divided into 2 halves (Memory as a 2D matrix):
    - *RAS* or *Row Access Strobe*
    - *CAS* or *Column Access Strobe*
- Cache uses *SRAM :* Static Random Access Memory
  - No refresh (6 transistors/bit vs. 1 transistor)

So, why do we care?

- By it’s nature, DRAM isn’t built for speed
  - Response times dependent on capacitive circuit properties which get worse as density increases
- DRAM process isn’t easy to integrate into CMOS process
  - DRAM is off chip
  - Connectors, wires, etc introduce slowness
  - IRAM efforts looking to integrating the two
- The objective is to design Memory Architectures to minimize impact of DRAM latency
  - Low Level: Memory chips
  - High Level memory designs.
  - You will pay $$$$$$ and then some $$$ for a good memory system.

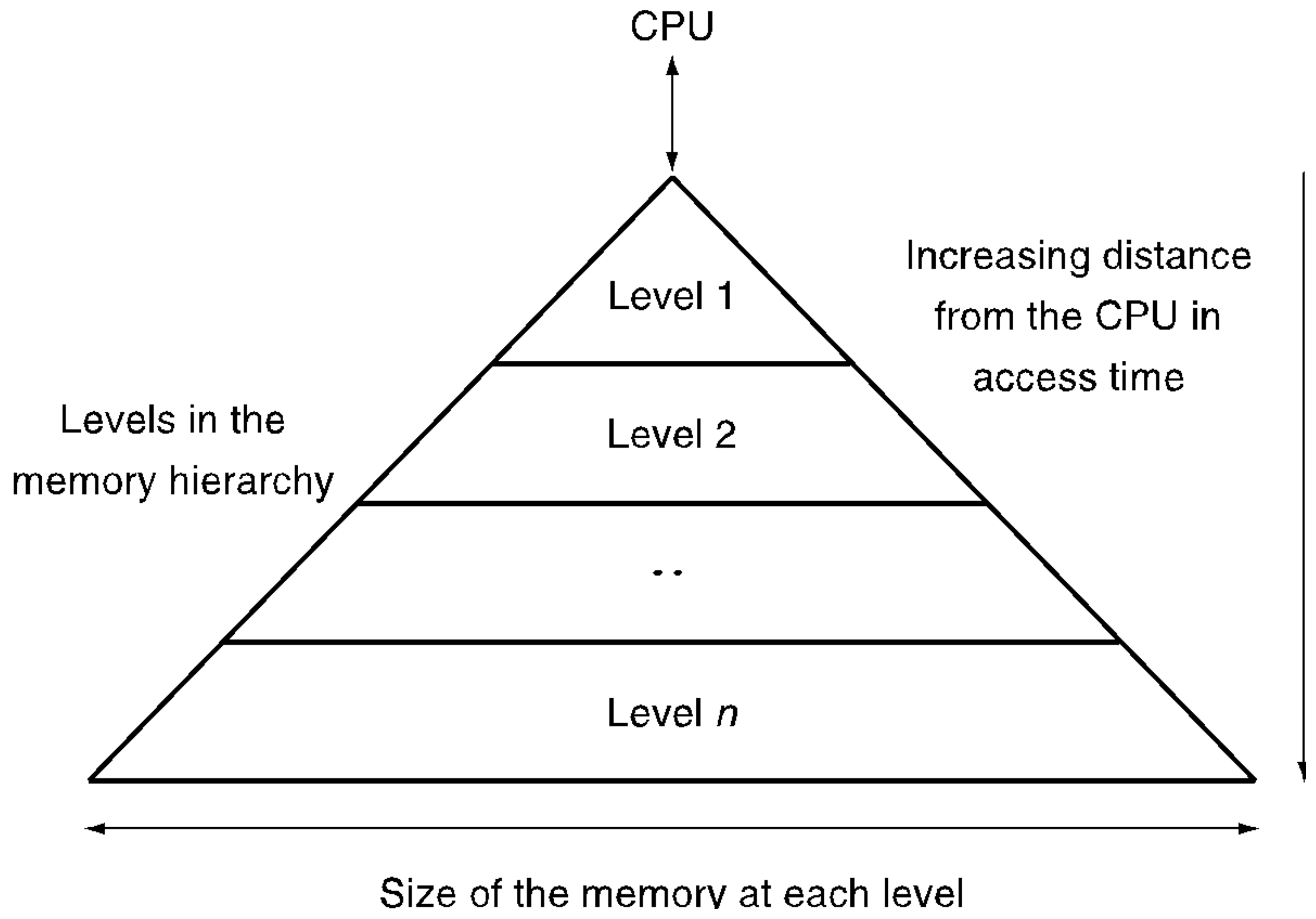
So, why do we care?

- 1960-1985: Speed = *f*(no. operations)
- 1990
  - Pipelined Execution & Fast Clock Rate
  - Out-of-Order execution
  - Superscalar Instruction Issue



- 1998: Speed = *f*(non-cached memory accesses)
- What does this mean for
  - Compilers? Operating Systems? Algorithms? Data Structures?

Overall Perspective

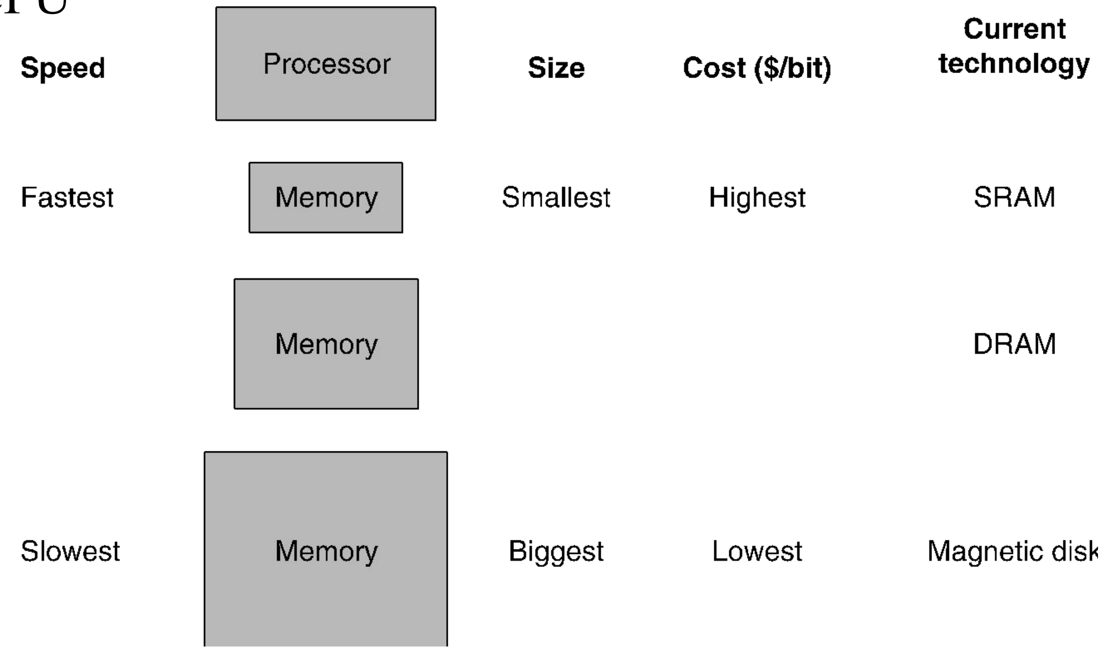

CPU
Level 1
Level 2
..
Level n
Levels in the memory hierarchy
Increasing distance from the CPU in access time
Size of the memory at each level

Principle of Locality

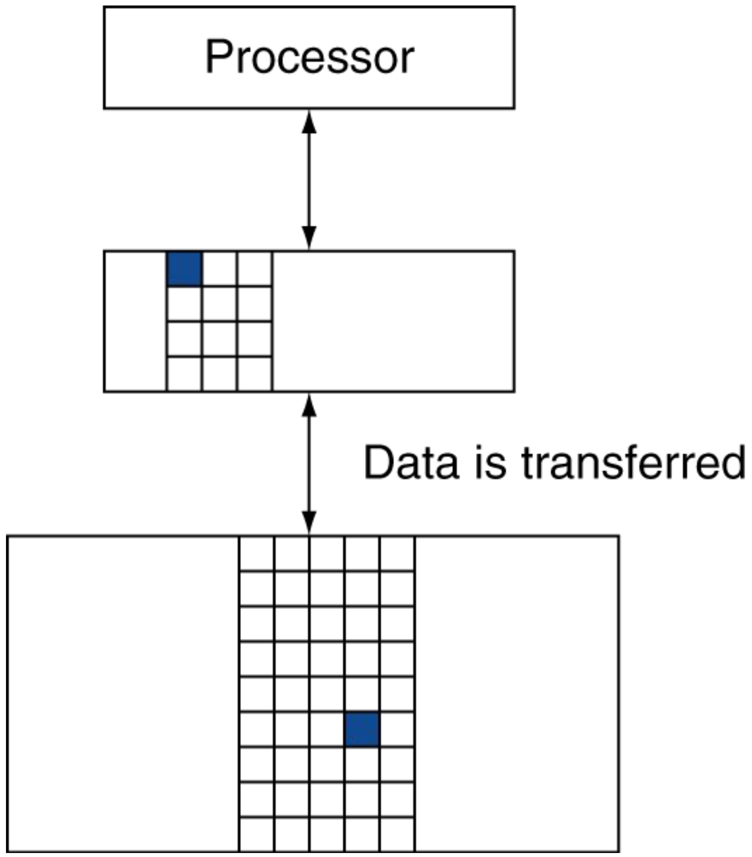
- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - Books read recently from library are going to be needed again
    - e.g., instructions in a loop, induction variables
- Spatial locality
  -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

Memory Hierarchy for Locality

- Create a 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



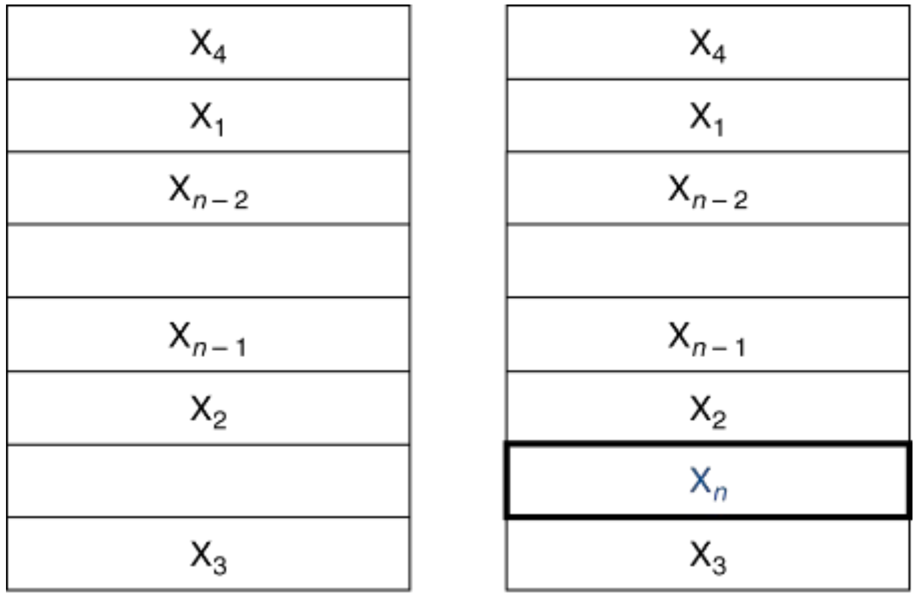
Memory Hierarchy Levels



- 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

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

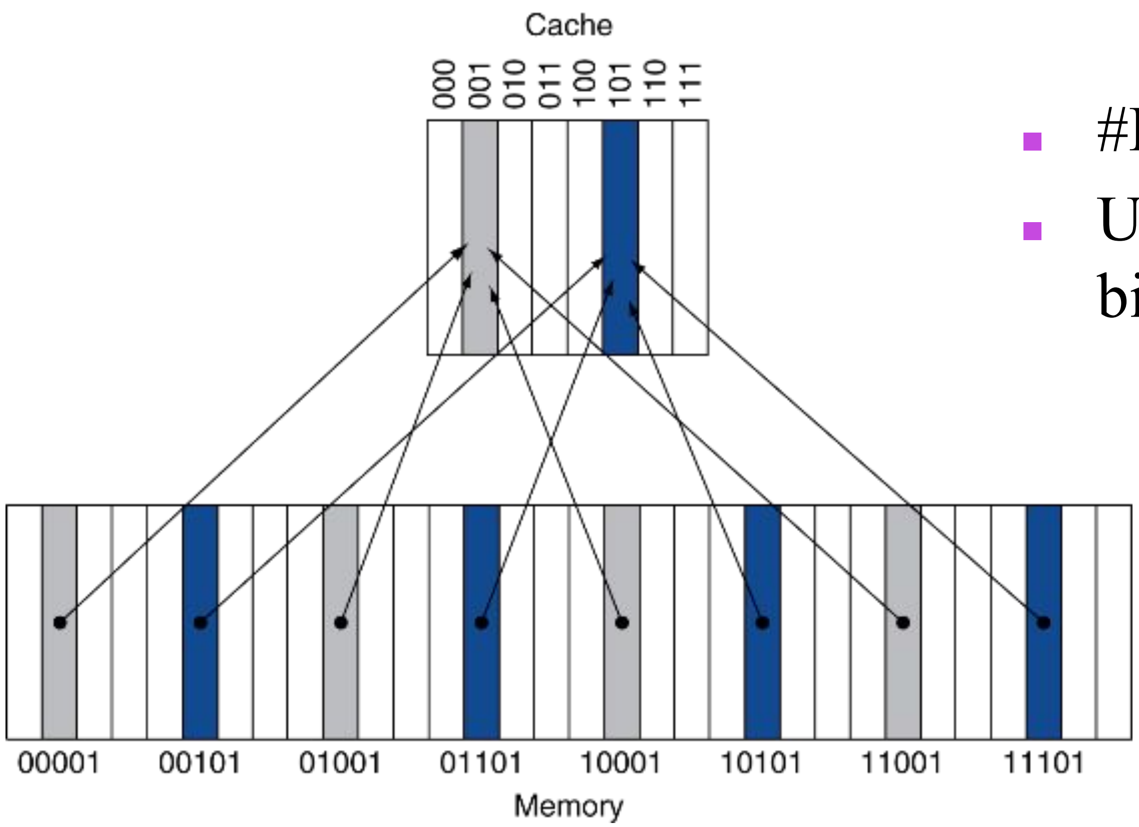
- How do we know if the data is present?
- Where do we look?

Cache design considerations

- Cache address
- Cache size
- Cache mapping functions
    - Direct
    - Associative
    - Set-associative
- Replacement algorithm
    - Least frequently used (LFU)
    - First in first out (FIFO)
    - Least recently used (LRU)
    - Random
- Write policy
    - Write through
    - Write back
    - Write once

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

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/garbage data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

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		

Cache Example

Word addr	Binary addr	Cache block
22	10 110	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

Word addr	Binary addr	Cache block
26	11 010	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

Word addr	Binary addr	Cache block
22	10 110	110
26	11 010	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

Word addr	Binary addr	Cache block
16	10 000	000
3	00 011	011
16	10 000	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

Word addr	Binary addr	Cache block
18	10 010	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		