Memory Hierarchy, and Cache Memory

Ver. 3

Professor: Yang Peng

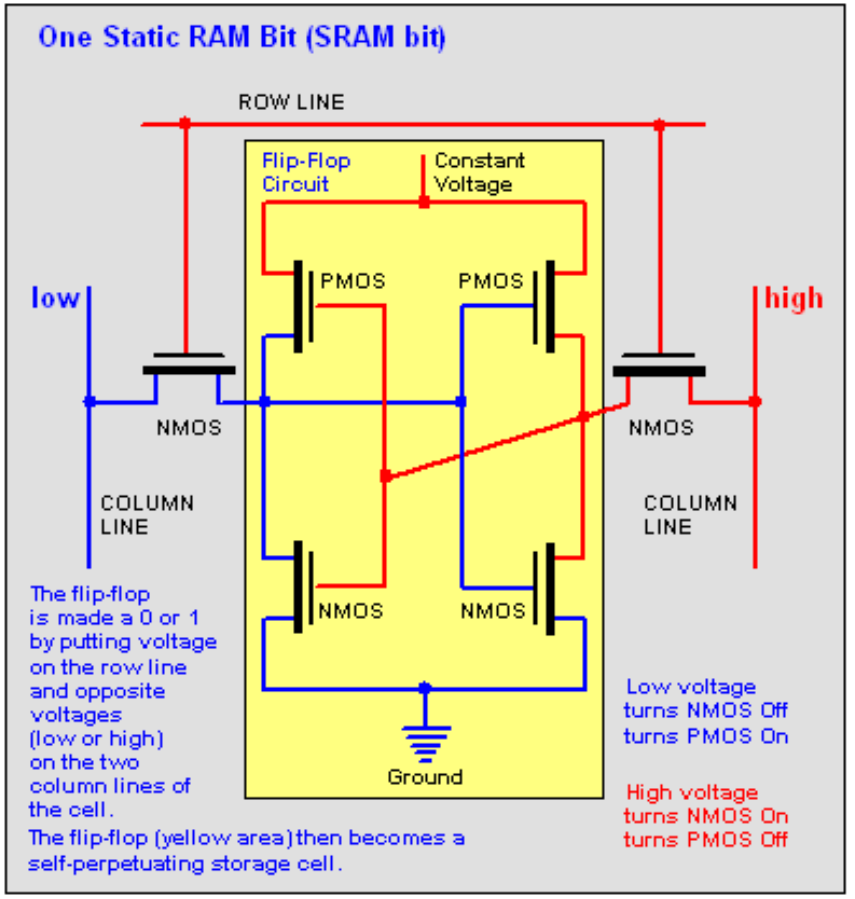
Topics

- Locality and Memory Hierarchy
  - Memory types and hierarchy
  - Spatial/temporal access locality
- Cache Memory
  - Basic algorithm
  - Performance metrics
  - Direct mapping

Memories: SRAM

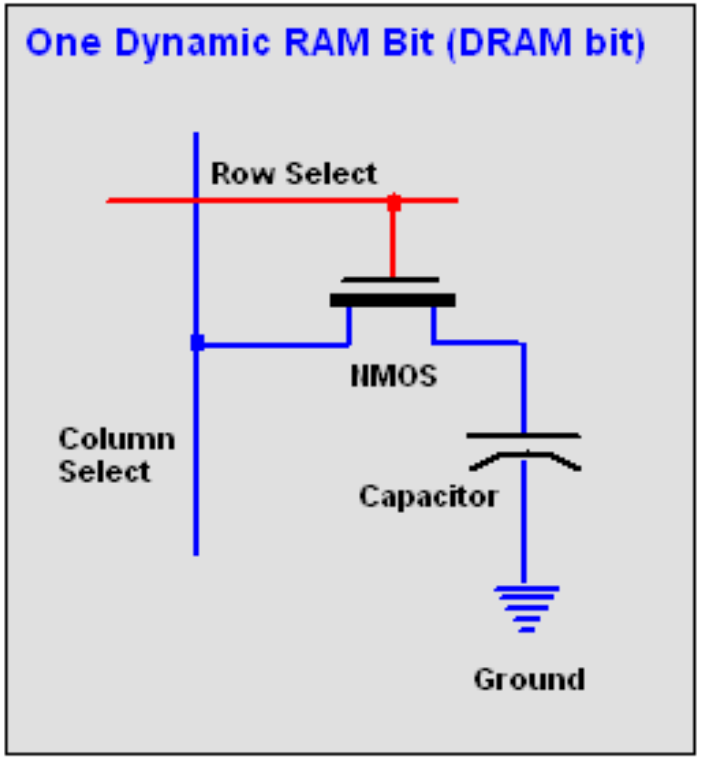
From Computer Desktop Encyclopedia

- Static Random-Access Memory
  - Value is stored using a pair of inverting gates
  - Fast, but takes up more space (4 to 6 transistors) than DRAM
  - Expensive



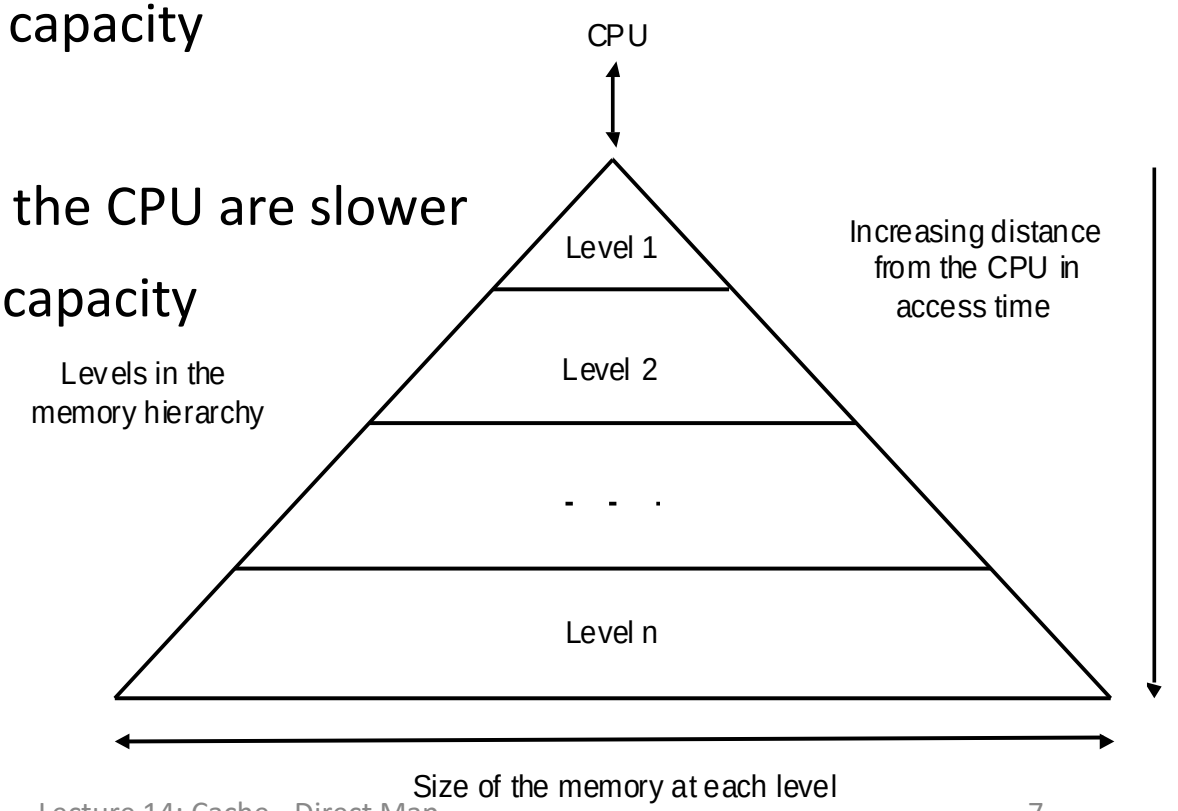
Memories: DRAM

- Dynamic Random-Access Memory
  - Value is stored as a charge on capacitor (must be refreshed)
  - Very small but slower than SRAM (factor of 5 to 10)
  - Cheap



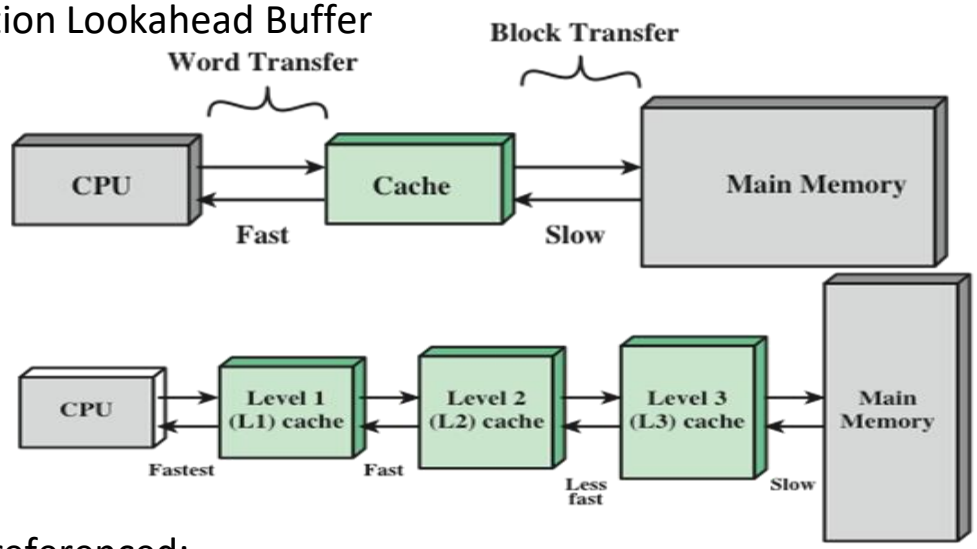
Memory Hierarchy

- *Memory Hierarchy*
  - Levels closer to the CPU are faster
    - Relatively small capacity
    - SRAM
  - Levels further from the CPU are slower
    - Relatively large capacity
    - DRAM



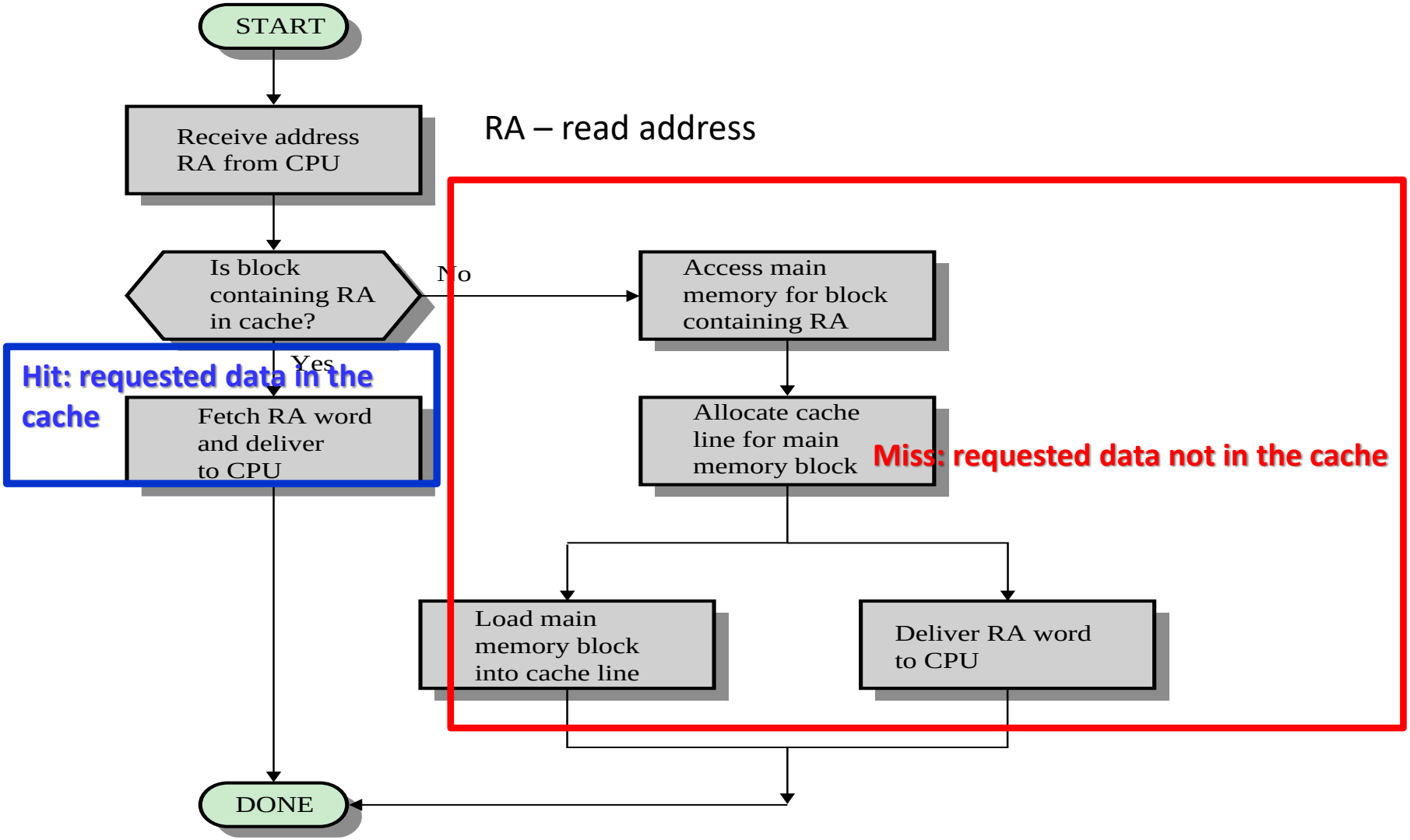
Locality

- Cache: maintain instructions, data, and address information that will be accessed soon
  - Instruction Cache
  - Data Cache
  - Translation Lookahead Buffer



- If an item is referenced:
  - ***Temporal locality:*** *Examples in programming?*
  - ***Spatial locality:*** *Examples in programming?*
  - ***Sequential Locality:*** *Instructions tend to be accessed sequentially.*

Cache Read Operation



Effective Execution Time

- **Block**: unit of data transfer (called *refill line* as well) (why?)
- **Hit rate**: percentage of accesses found in cache
- **Miss rate**: percentage of accesses not in cache (1 - hit rate)
- **Hit time**: time to access cache
- **Miss penalty**: time to replace block in cache with appropriate one (replace a block, not a data), tends to large. (100 times more than hit time)
- Effective execution time (EET):
  - ***hit rate * hit time + miss rate * miss penalty***
  - Goal is, as less EET as possible. (of course)
  - Decreasing miss rate (simultaneously increasing hit rate) is an important goal.

Class Exercise

1. A certain RISC processor has an on-chip cache with the following specifications (because its logic simple and thus has space for cache!):

- Cache hit rate for the L1 cache is 98%
- Instructions that are located in-cache execute in **1 clock cycle**.
- Instructions that are not found in the on-chip cache will cause the processor to stop program execution and refill a portion of the cache. This operation takes **100 clock cycles** to execute.

Q) What is the EET in nanoseconds for this RISC processor if the clock frequency is 100 MHz?

Answer:
Clock cycle time = 1/100M = 10ns
EET = hit rate X hit time + miss rate X miss penalty
= 0.98 X 1 (clock cycle) X 10ns + 0.02 X 100 (clock cycle) X 10 ns
= 9.8 ns + 20 ns = 29.8 ns

Cache Organization

- Cache memory consists of several cache entries.
  - LRU bits: used for replacing a victim cache line with a new entry (To be discussed in Lec. 15)
  - Valid bit: indicating if the cache line is valid.
  - Tag field: holding the memory address and determining if there is a match to a CPU request
  - Data field: Copies of the contents of main memory (instructions / data)

LRU bits	Valid	Tag	Data
Bit0: either set 0/1 or 2/3 accessed Bit1: either set 0 or 1 accessed Bit2: either set 2 or 3 accessed	0 (invalid) 1 (valid)	Which column of memory stored	An actual copy of memory block

Cache Design

- **Goal: Increase the performance of computer using Caches!**

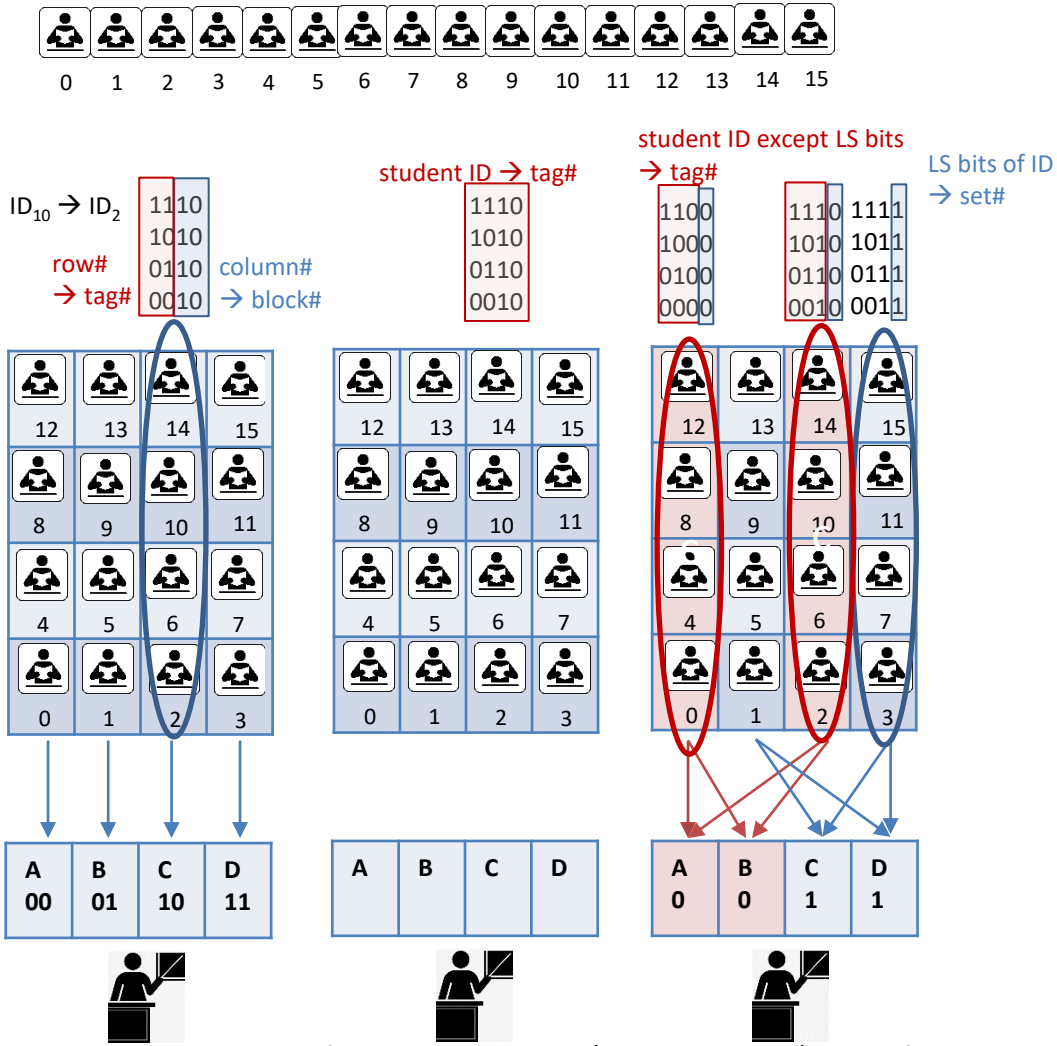
Design Issues	In Details	Examples
Cache Mapping Schemes	How to map the data in memory to the data in cache?	Direct cache mapping Associative cache mapping
Write Policies	When to write back dirty data from cache to main memory	Write through Write back
Replacement Algorithms	How to identify a victim cache line that should be written back to main memory	FIFO LRU: least recently use
Cache Address	A program is running in a logical (virtual address always starting address 0x00000000) but we only have one physical address.	Physical or virtual address
Cache Size	A long or short line? Which is efficient?	32, 64, or 128 bytes
# Caches	What is the optimal number of caches?	L1, L2, and L3

Mapping Function

- Processor can access the data/instruction by its address in main memory
- But data in cache DOES NOT have an address!
- How to map the address of memory to some data in cache?
- Mapping Schemes
  - Direct Mapping
  - Associative Mapping
  - Set Associative Mapping

Metaphors of Mapping Functions

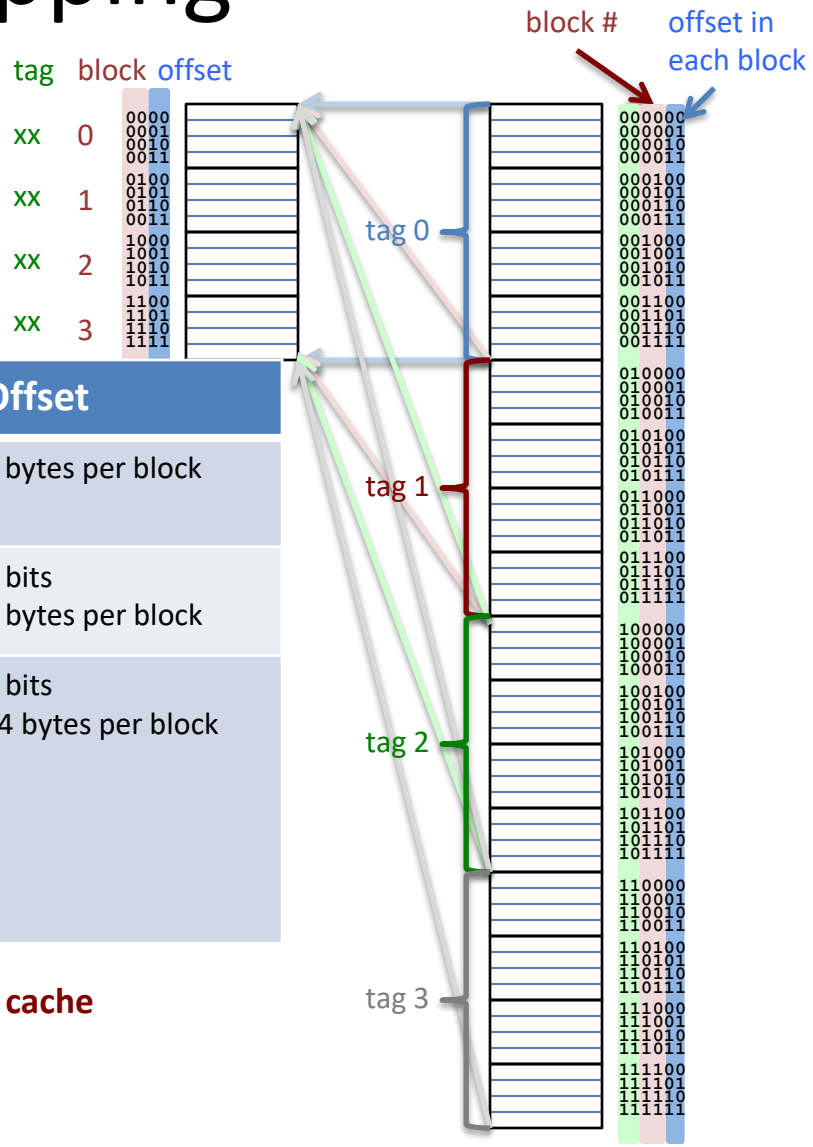
- 16 students are lined up to enter a tiered classroom. They are seated in the order of their student IDs.
- Today is a self-studying class. Whenever a student has a question, s/he must go down to the special desk with 4 chairs, right in front of the instructor.
- Each student has the same textbook and ask the instructor about a specific page# (i.e., offset)
- **Direct mapping:** students on the same column compete the same chair. E.g., students 0, 4, 8, and 12 need to share chair A, but one at a time
- **Associative mapping:** students can have any one of chairs A-D as far as it is unoccupied.
- **2-way set-associative mapping:** students on every other columns in the same color (red or blue) can choose one of the two same-colored chairs AB or CD.



The instructor can always identify each student with her/his tag on where s/he came from.

Direct Mapping

- Bits in main memory address is divided into three fields



	Tag	Block	Offset
General description	Identify which column in memory	# blocks or #cache lines in cache	# bytes per block
Example on the right side	2 bit 4 columns	2 bits 4 blocks	2 bits 4 bytes per block
More practical example - 512KB memory - 32KB cache - 64-byte / cache line	4 bits 512KB / 32KB = 16 columns	9 bits 32KB / 64B = 512 blocks	6 bits 64 bytes per block

How to find the correspondence between memory and cache
1) Check the block.
2) Then, compare the tag.

Direct Mapping

Divide the memory address into tag, block, and offset bits

Memory size	**M** bytes
Cache size	**C** bytes
Cache block size	**B** bytes

Tag bits: log(M/C)	Block bits: log(C/B)	Offset bits: log B

Memory size	64 bytes
Cache size	16 bytes
Cache block size	4 bytes

Tag (2)	Block (2)	Offset (2)

Compute address 47's tag, block and offset
47_{10} = 0x2F = 2_101111
10 11 11
Tag = 2, Block = 3, Offset = 3

Memory size	512KB
Cache size	32KB
Cache block size	64B

Tag (4)	Block (9)	Offset (6)

Compute address 278407's tag, block and offset
278407_{10} = 0x43F87 = 2_100 0011
1111 1000 0111
1000 011111110 000111
Tag = 8, Block = 254, Offset = 7

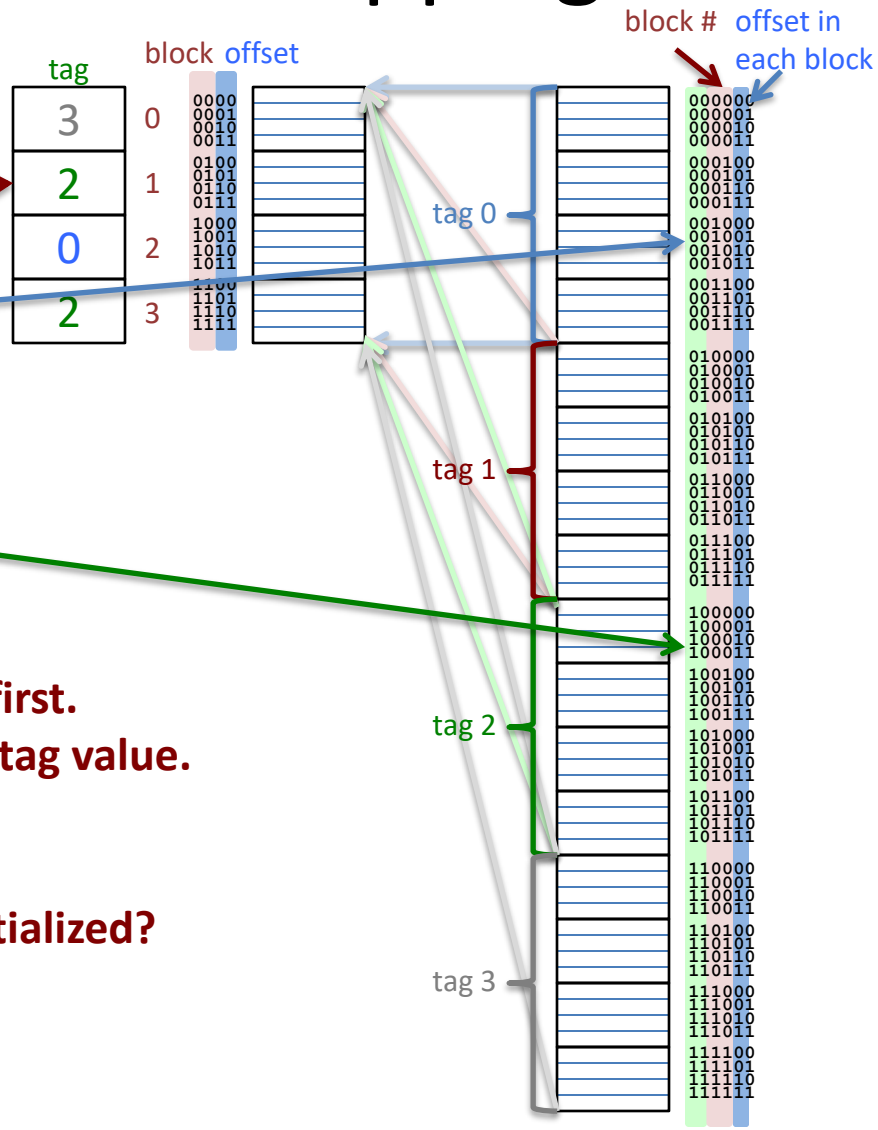
Memory size	4GB
Cache size	64KB
Cache block size	64B

Tag (16)	Block (10)	Offset (6)

#tags = 4G / 64KB = 2^{32}/ (2^6 * 2^{10}) = 2^{16}
#blocks = 64K / 64 = 2^{16} / 2^6 = 2^{10}
#offset = 64B = 2^6

Find Hit/Miss in Direct Mapping

- Let's assume that the cache tags are shown on the right.
- Does address **001001** make a hit or miss?
- Does address **100011** make a hit or miss?



How to find:

1) **Check the block at first.**
2) **Then, compare the tag value.**

Question:

How should tags be initialized?

The Valid bit!

Summary

- RISC:
  - More memory needed and more dependent on compiler techniques but faster.
- Memory Hierarchy:
  - Utilize locality of memory access.
- Cache memory mapping
  - Direct mapping
  - Associative mapping
  - Set associative mapping

} Study in the next lecture