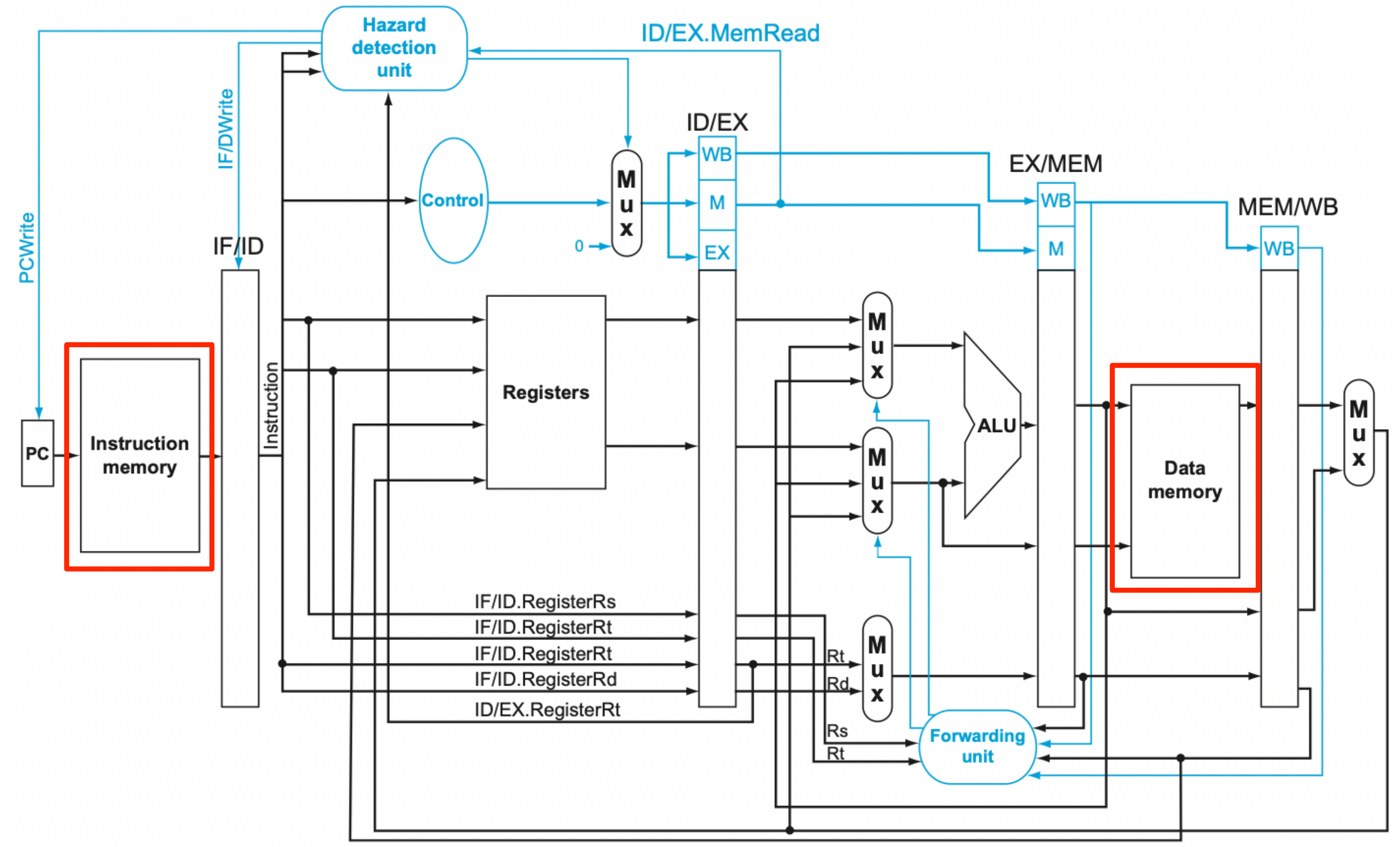
Computer Architecture
(ENE1004)

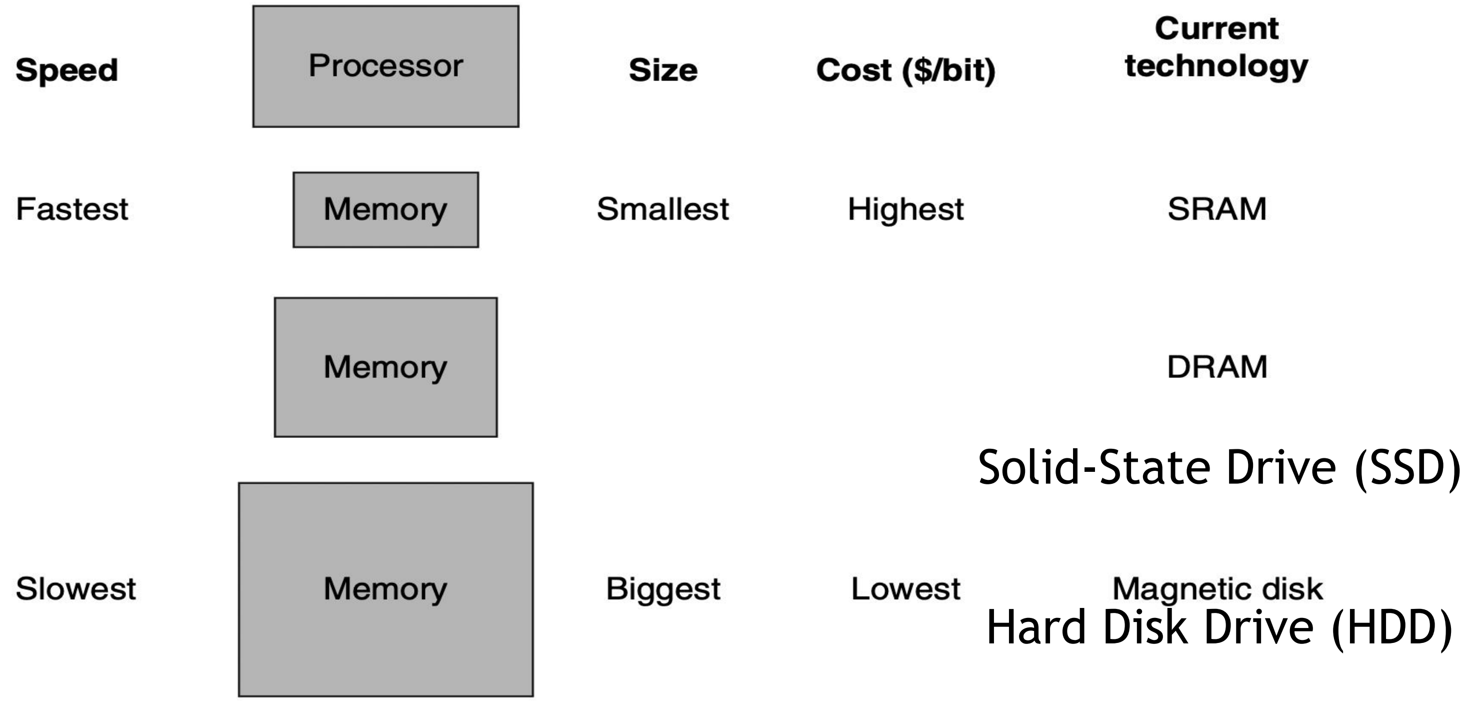
Lec – 19: Large and Fast: Exploiting Memory Hierarchy (Chapter 5) – 1

Ideal Memory System



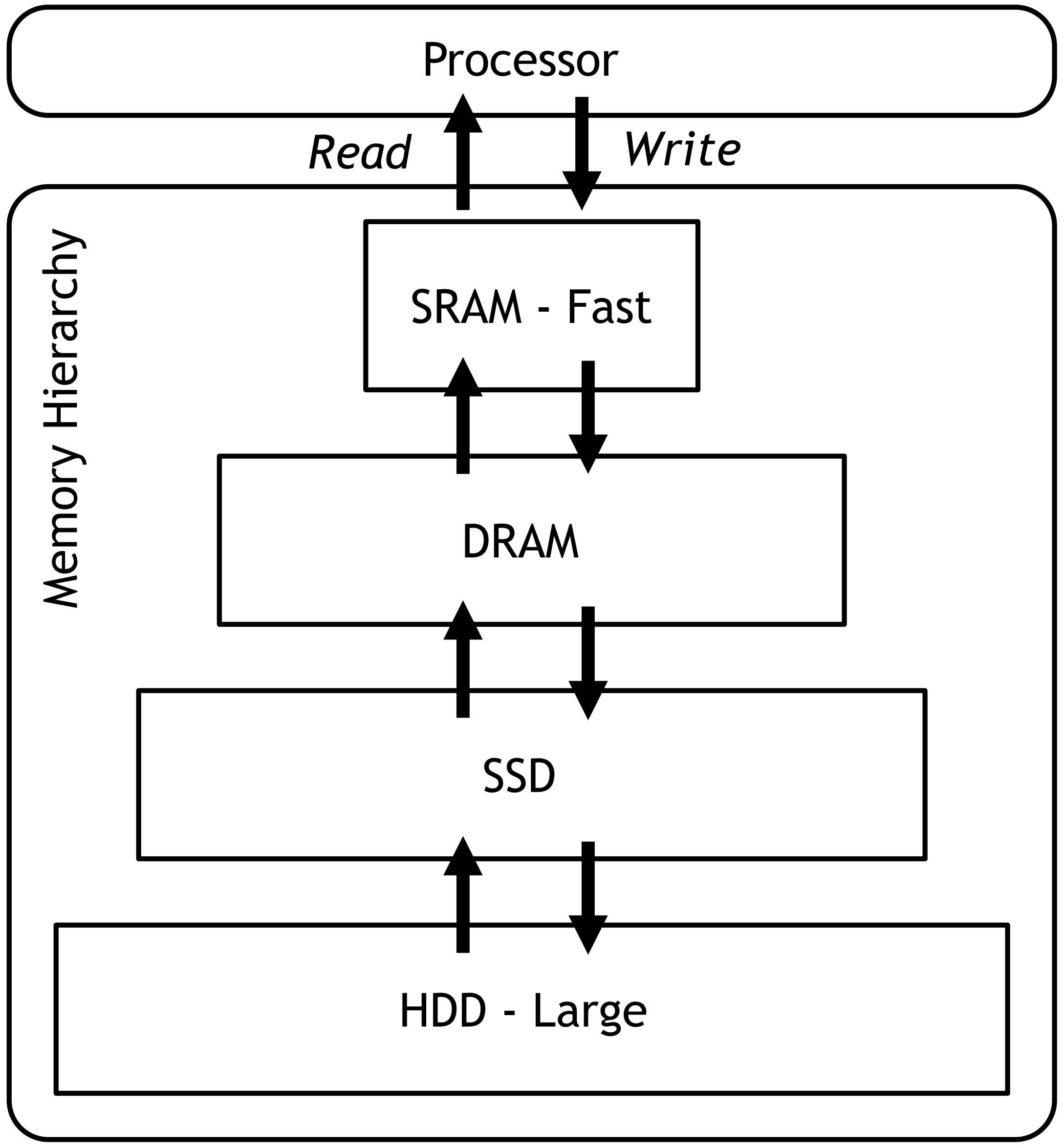
- All the data (program, data) are kept in the memory
  - Load and store instructions access (read from and write into) the memory
- From the earliest days, programmers have wanted “unlimited amounts of fast memory”
  - Fast memory (performance angle)
  - Large memory (capacity angle)

Current Memory Technologies



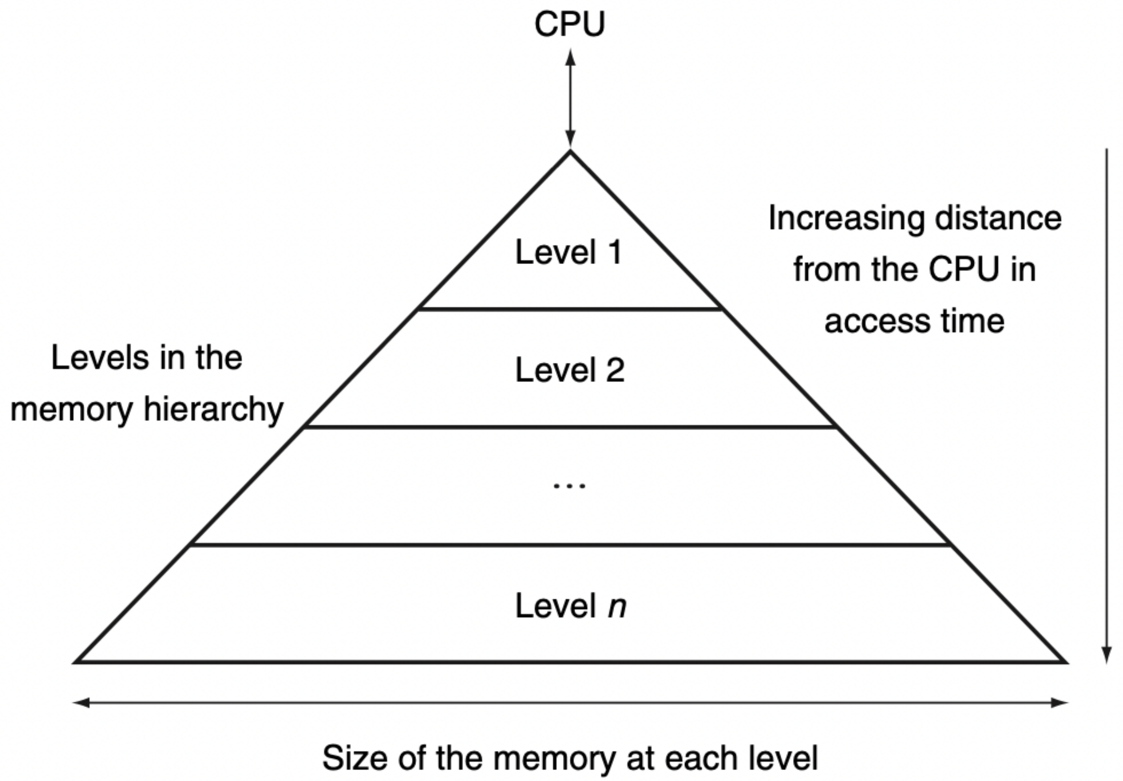
- Comparison: SRAM vs DRAM vs Solid-State Drive (SSD) vs Hard Disk Drive (HDD)
  - Cost ($/bit) comparison: SRAM > DRAM > SSD > HDD
  - Performance (read/write speed) comparison: SRAM > DRAM > SSD > HDD
  - Capacity (size) comparison: SRAM < DRAM < SSD < HDD
- There is NO single memory technology that is both large and fast
  - However, modern computers provide an “illusion” of the large and fast memory

Memory Hierarchy (1)



- Memory system includes different memory technologies in a *hierarchical* fashion
  - At the lowest level, the largest/slowest memory
  - At the first level, the fastest/smallest memory
  - As the distance from the processor increases, both the size and the access time increase
- The data is similarly hierarchical
  - All the data is stored at the lowest level
  - A level closer to the processor is generally a "subset" of any level further away
  - The subset can change over time by sending data upwards/downwards between any two level
  - The smallest subset is stored at the first level
  - Processor can directly access only the subset data at the first level

Memory Hierarchy (2)

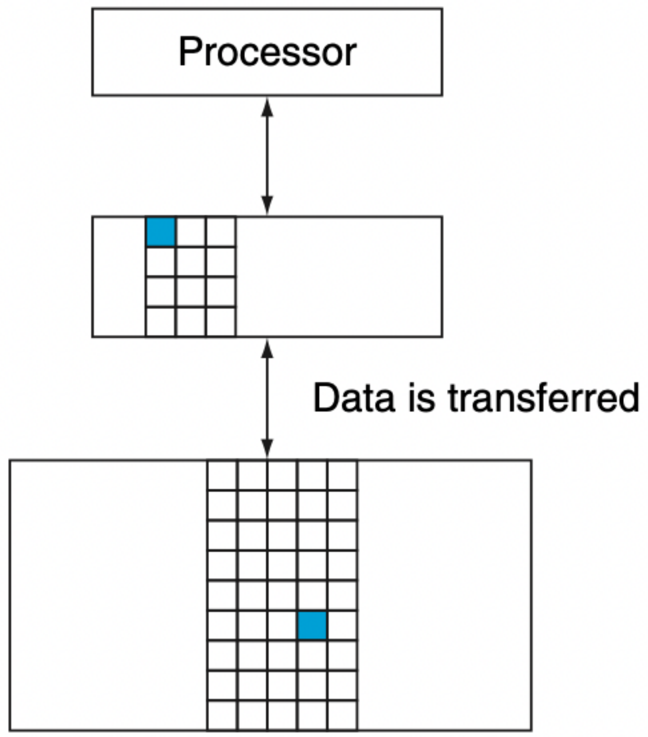

CPU
Level 1
Level 2
...
Level *n*
Levels in the memory hierarchy
Increasing distance from the CPU in access time
Size of the memory at each level

- Processor accesses only the level-1 memory
  - Processor *feels* like it works with the fastest memory
- Processor can access all the data set in the level-n memory
  - Any data can be sent upwards to the level-1 memory from level-n memory
  - Processor *feels* like it works with the largest memory

Memory Hierarchy (3)

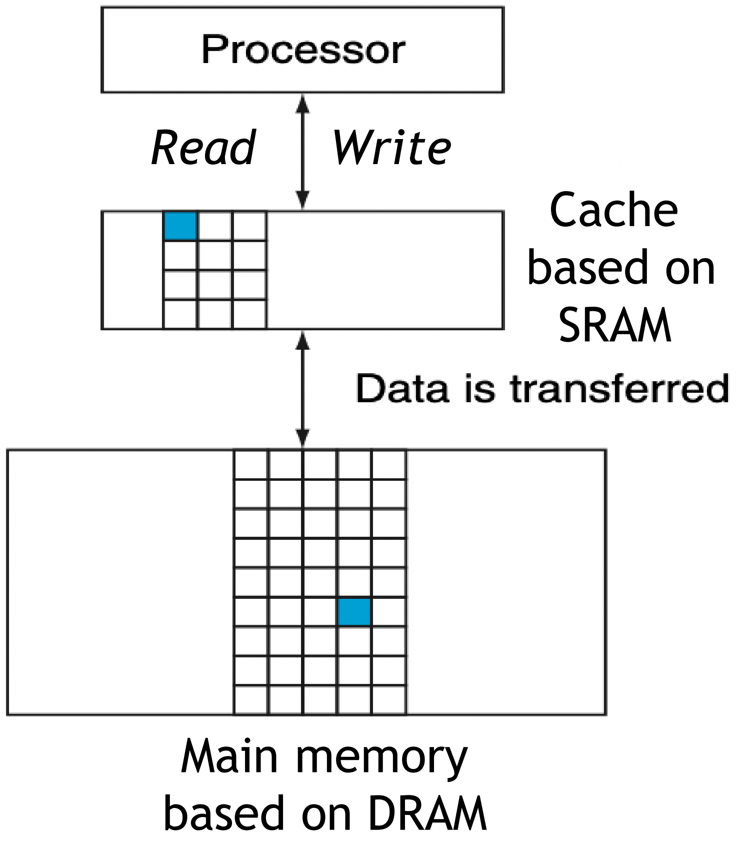
- A level closer to the processor is generally a “subset” of any level further away
- This concept is based on the principle of locality in the memory accesses
  - Two different types of locality are observed in real-world applications
- Temporal locality (locality in time)
  - If a data item is referenced (accessed), it will tend to be referenced again soon
  - So, if such a data item is stored in an upper level, we can reduce the access times to it
  - In a for loop, the variable for iterations is referenced again in a short period of time
- Spatial locality (locality in space)
  - If an item is referenced, items whose addresses are close by will tend to be referenced soon
  - So, if data items in the neighborhood are stored in an upper level, we can reduce the access times to them
  - In an array, neighboring elements are usually referenced together in a short period of time

In Every Pair of Levels



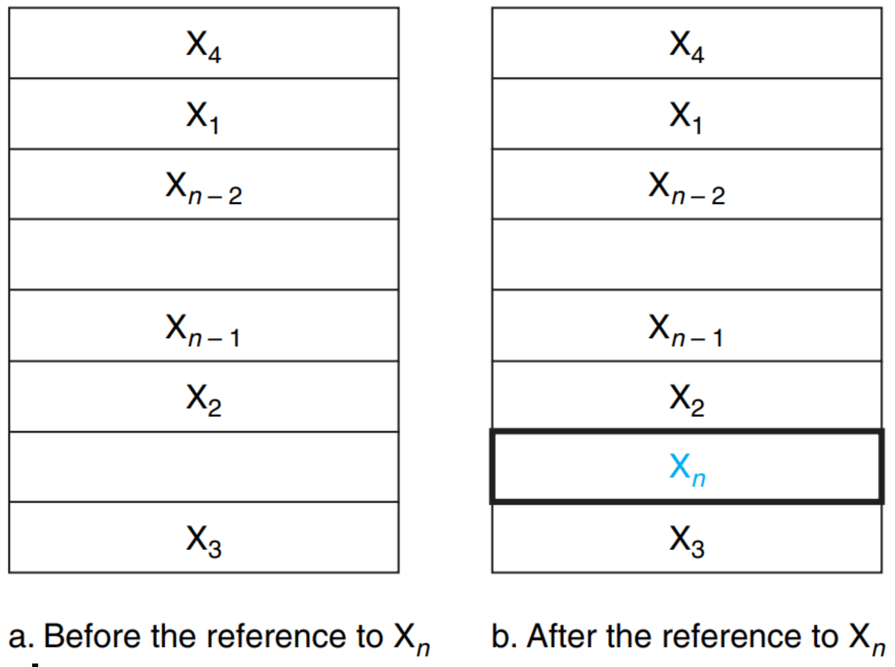
- Every pair of levels in the hierarchy can be thought of as having an upper and lower level
  - Data is copied between only two adjacent levels at a time
- The minimum unit of information that can be either present or not present in the two levels is called "block" or "line"
- If the data requested by the processor appears in some block in the upper level, the request is called a "hit"
  - It can be serviced quickly from the upper-level faster memory
  - The time to access the upper level memory is "hit time"
- If the data is not found in the upper level, it is called "miss"
  - The data is copied from the lower level to the upper level; then is read
  - The total time is "miss penalty"
- "Hit ratio" is the fraction of accesses found in the upper level
  - The higher the hit ratio, the higher the performance
  - "Miss ratio" = 1 - "hit ratio"

Basics of Caches (1)



- In this course, we'll focus on the most upper level two memory technologies
  - SRAM in upper level is called "cache"
  - DRAM in lower level is called "main memory"
  - Processor always works with the cache
  - Data set in the cache changes by transferring data between the cache and the main memory

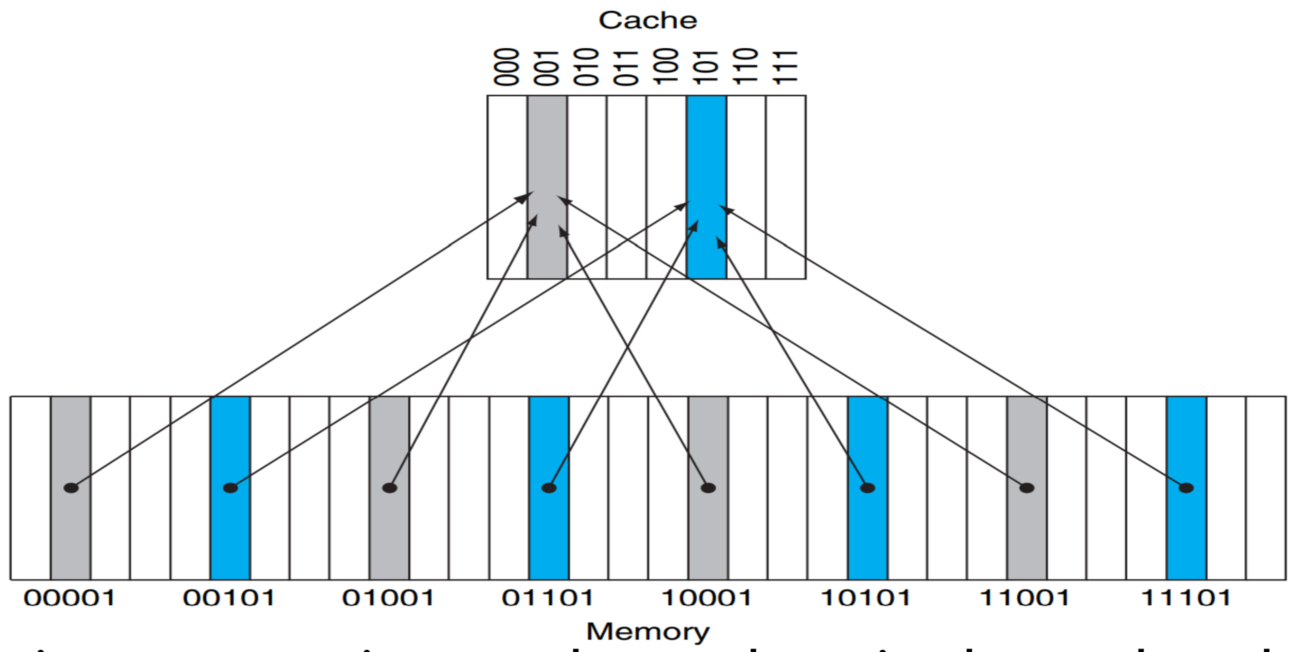
Basics of Caches (2)



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

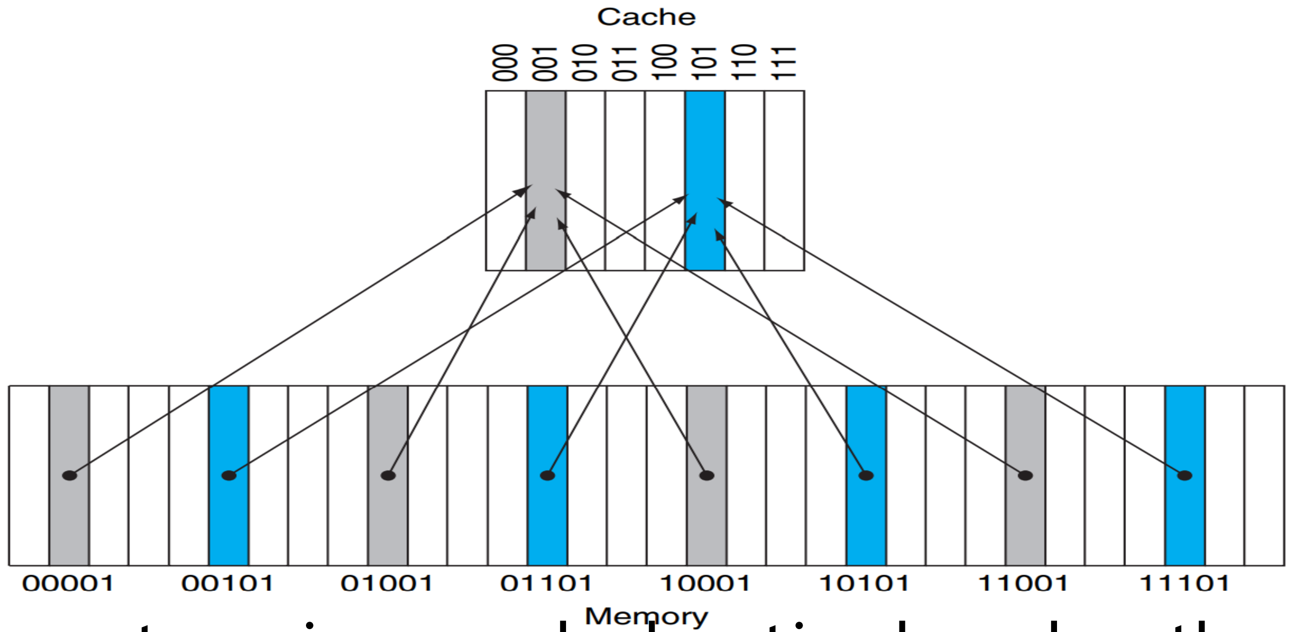
- An example of simple cache
  - The cache can hold up to 8 data items; currently, it holds 6 data items and two slots are empty
  - Given a new access to one of the six items, the data is provided from the cache to the processor
  - Given a new item X_n, it is brought from the main memory and accommodated in the cache
- Here, one may have the following important questions
  - (1) How do we know if a data item is in the cache?
  - (2) If it is, how do we find it?

Direct Mapped Cache (1)



- Answer: If each data item can go in exactly on place in the cache, then it is straightforward to find the data item if it is in the cache
- Direct-mapped cache: each memory location is mapped directly to one location in the cache
    - It assigns a location in the cache for each data item
    - The assignment of a cache location based on the address of the data item in memory
- Here, a cache location is assigned to multiple data items
    - 8 cache blocks should be assigned to 32 data items
    - 4 different data items can be placed in a single cache block

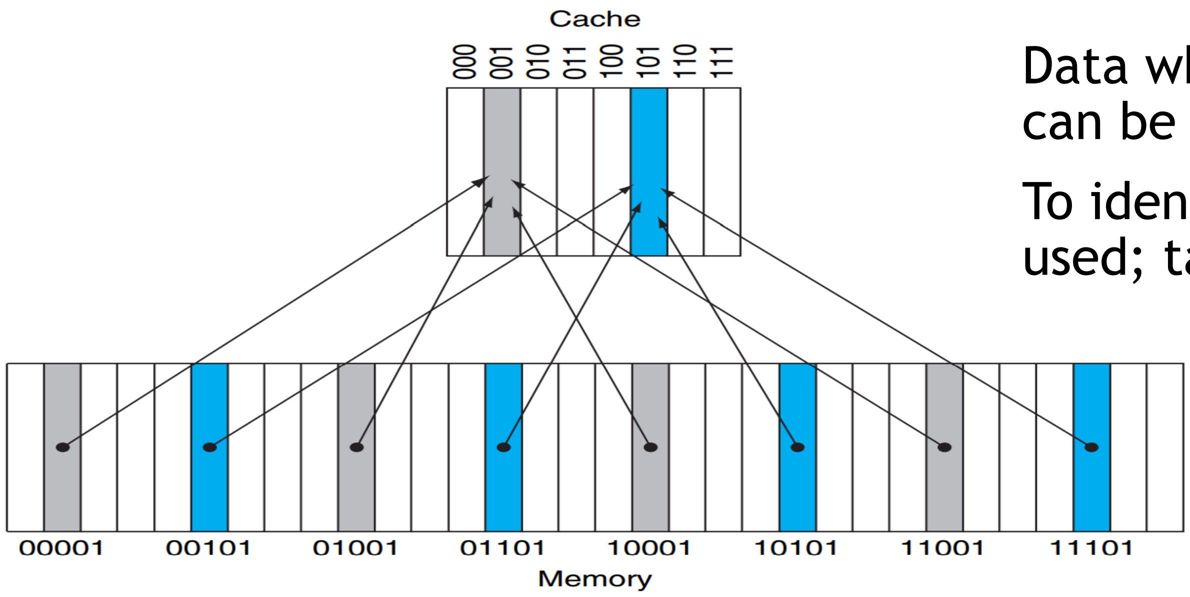
Direct Mapped Cache (2) – Cache Location



- Modulo operation
  - 7 module 3 = 1
  - 5 module 3 = 2
  - 10 module 3 = 1

- One simple way to assign a cache location based on the address of the data item
  - (Block address) modulo (Number of blocks in the cache)
- In the example, the number of blocks in the cache is 8 (2^3)
  - Memory address 1_{ten} (00001_{two}) is mapped to cache block 1_{ten} (001_{two}); 1 module 8 = 1
  - Memory addresses 9_{ten} (01001_{two}), 17_{ten} (10001_{two}), and 25_{ten} (11001_{two}) are also mapped to cache block 1_{ten} (001_{two}); 9 module 8 = 1, 17 module 8 = 1, and 25 module 8 =1
  - Memory addresses 5_{ten} (00101_{two}), 13_{ten} (01101_{two}), 21_{ten} (10101_{two}), and 29_{ten} (11101_{two}) are all mapped to cache block 5_{ten} (101_{two})
- Simply, check the last 3 bits of the memory address; the size of cache (2^3)!

Direct Mapped Cache (3) - Tag

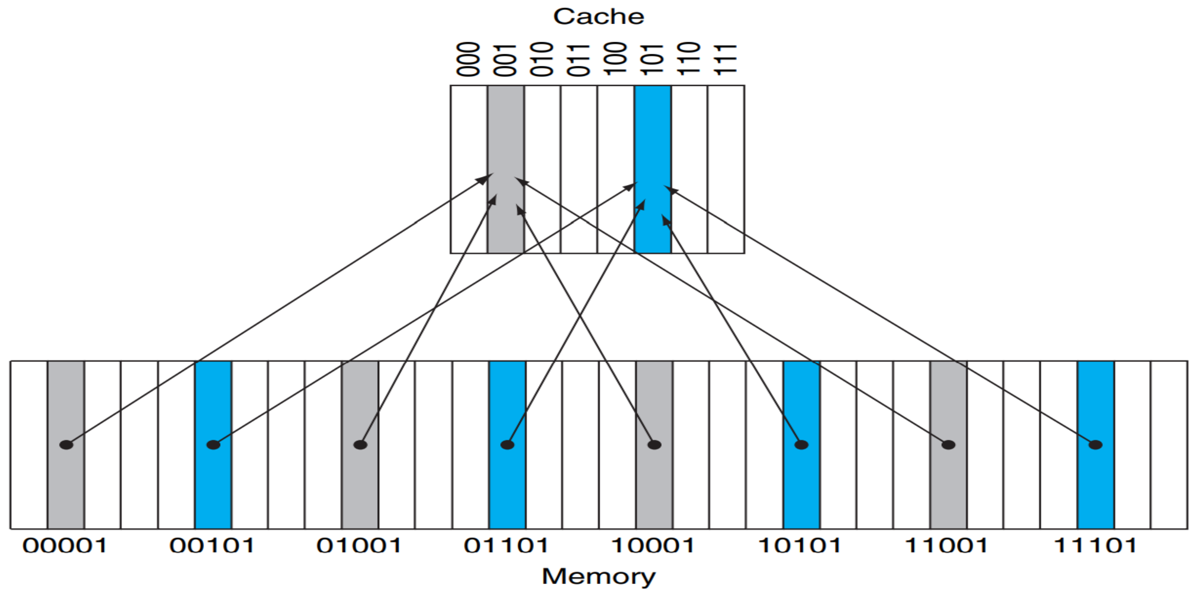


Data whose addresses are 00001, 01001, 10001, and 11001 can be placed in cache @ 001; indices are the same!

To identify one of the four, tags (00, 01, 10, 11) can be used; tags are all different!

- A cache location can contain the data items of different memory locations
  - Question: How do we know whether the requested data corresponds to the data in the cache?
  - We can answer the question by adding "tag" to each cache location
- Memory address is divided and used for two different purposes
  - The lower bits are used to identify the cache location (called "index")
  - The upper bits are used to identify a specific data (called "tag")
  - (b4 b3 b2 b1 b0) is divided; the lower 3 bits are "index", which is used for a cache location; the upper 2 bits are "tag", which is used for a specific data

Direct Mapped Cache (4) – Valid Bit



- A cache block does not always hold a valid data
  - The cache block can be empty, if none of the four data exist in the cache
  - When a processor starts up, the cache does not have any data
- We can address the above by adding a "valid bit" to each cache location
  - If it is set, the cache location has a valid data; then, refer to the tag if it is the data what you want
  - If it is not set, you do not have to check the tag, since any information in the cache location is not valid