RUTGERS

THE STATE UNIVERSITY
OF NEW JERSEY

Computer Architecture (CS-211) Recitation-9

Song Wen

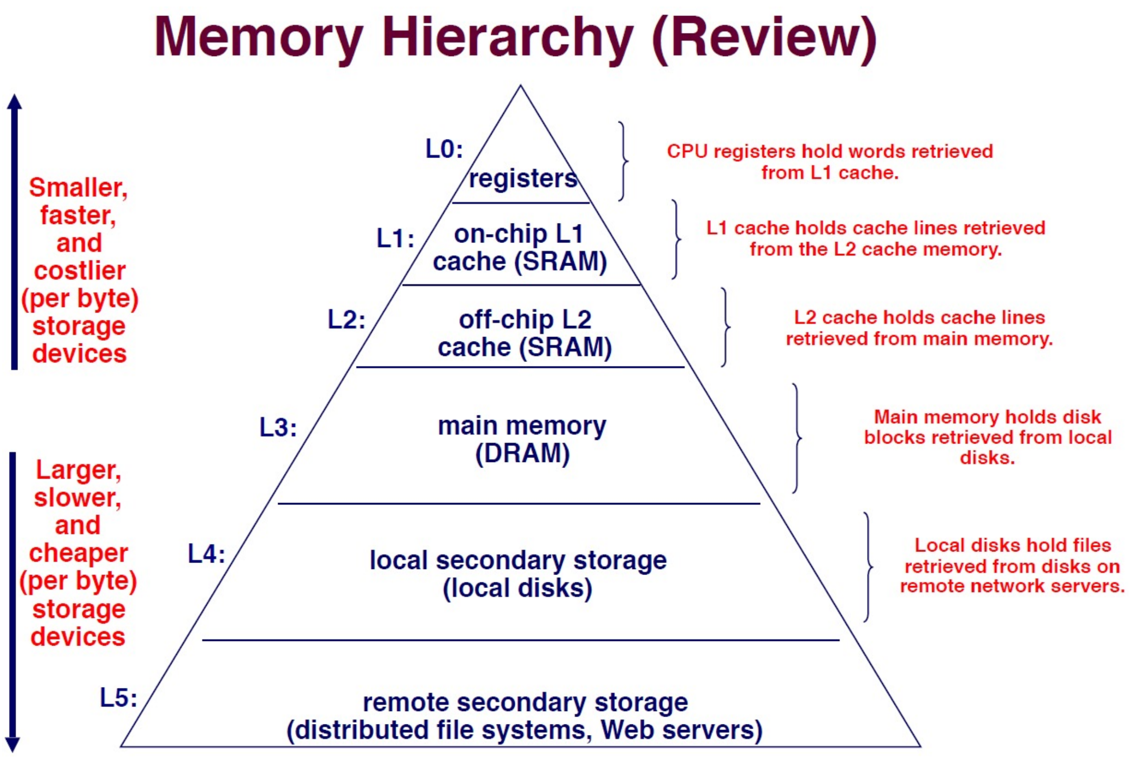
Topics

- Cache
- PA4

* Some materials are collected and compiled from previous year's CS 211 lectures and TAs

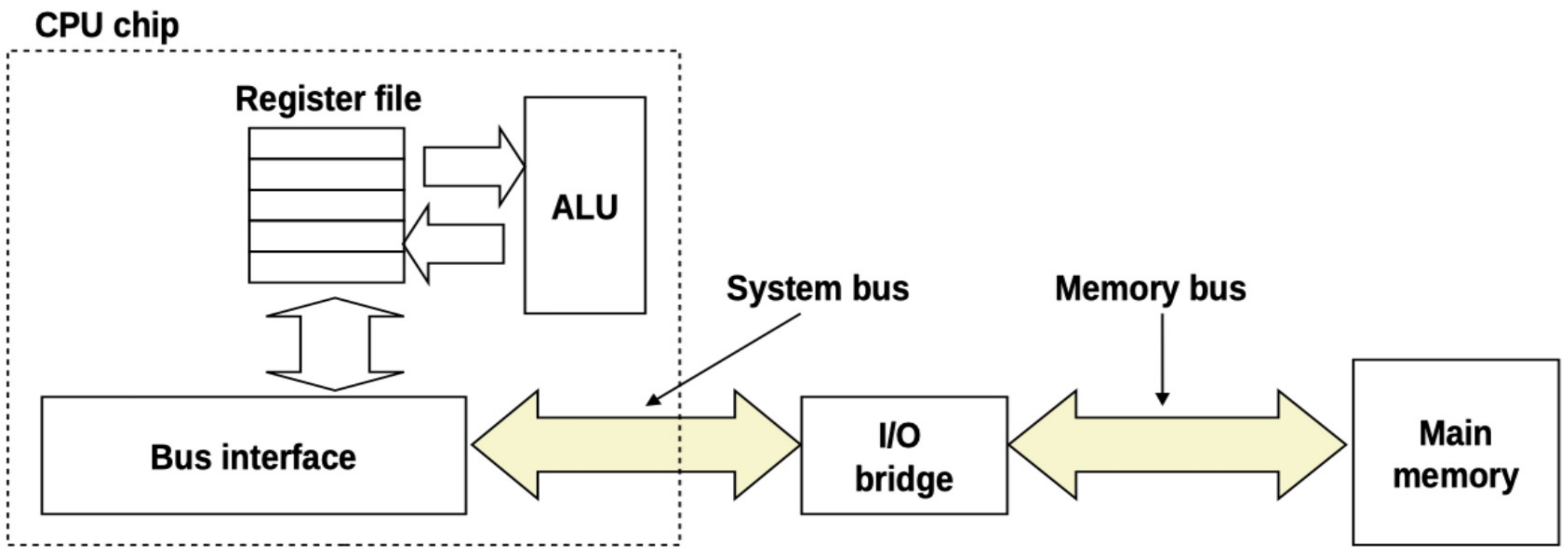
Memory Hierarchy

- A cache is a **smaller, faster memory**, closer to a processor core, which stores copies of the data from frequently used main memory locations.


Memory Hierarchy (Review)
Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0:
registers
L1:
on-chip L1 cache (SRAM)
L2:
off-chip L2 cache (SRAM)
L3:
main memory (DRAM)
L4:
local secondary storage (local disks)
L5:
remote secondary storage (distributed file systems, Web servers)
CPU registers hold words retrieved from L1 cache.
L1 cache holds cache lines retrieved from the L2 cache memory.
L2 cache holds cache lines retrieved from main memory.
Main memory holds disk blocks retrieved from local disks.
Local disks hold files retrieved from disks on remote network servers.

Memory Hierarchy

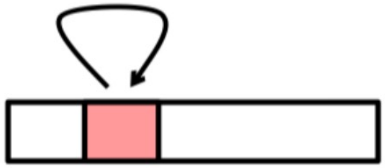
- A bus is a collection of parallel wires that carry address, data, and control signals.
- Buses are typically shared by multiple devices.



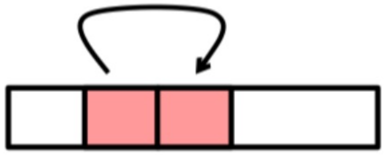
Locality

- **Principle of Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently**

- **Temporal locality:**
  - Recently referenced items are likely to be referenced again in the near future



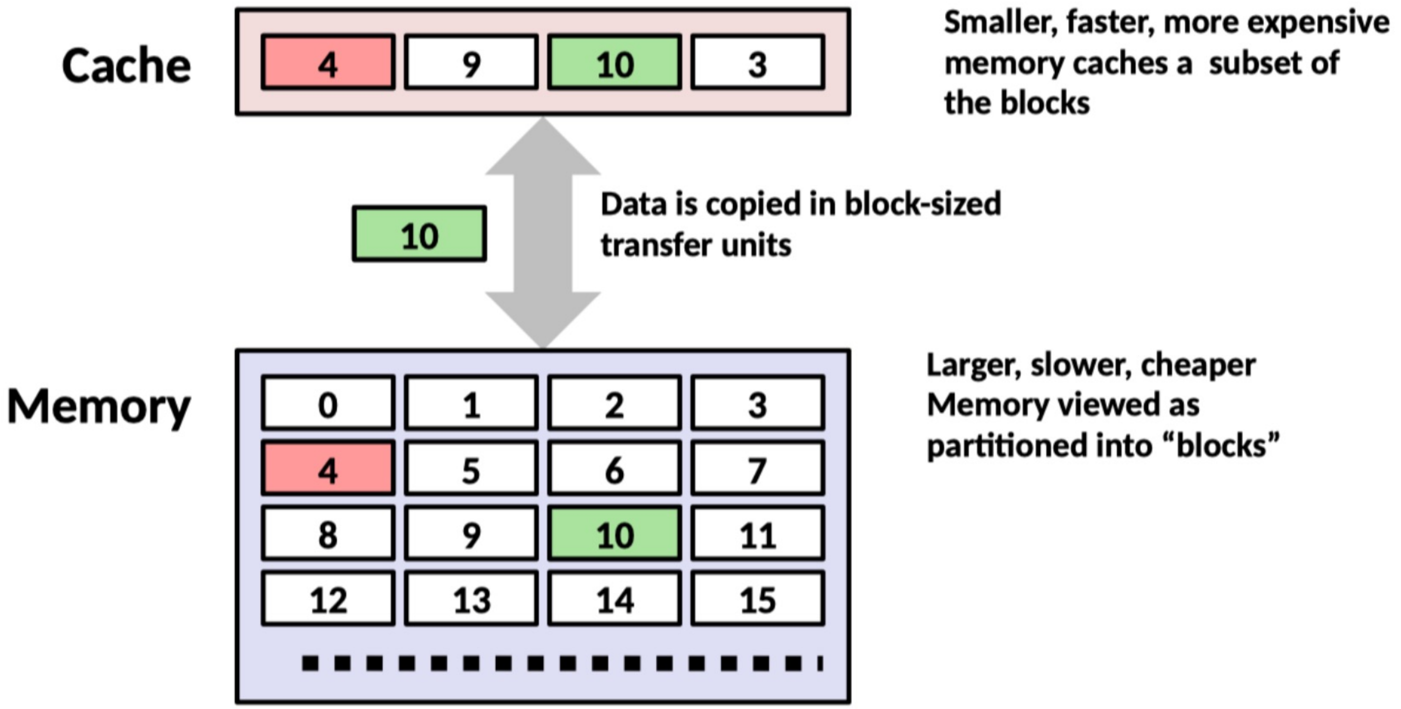
- **Spatial locality:**
  - Items with nearby addresses tend to be referenced close together in time



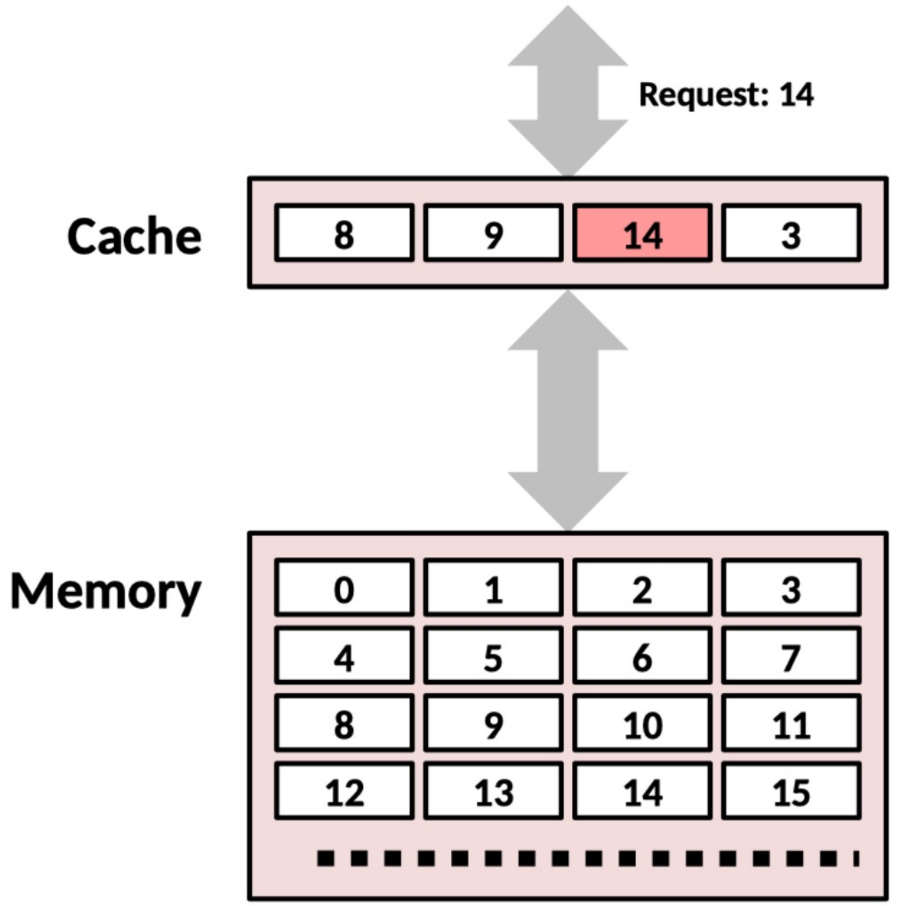
Caches

- ***Cache:*** **A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.**
- **Fundamental idea of a memory hierarchy:**
  - For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.
- **Why do memory hierarchies work?**
  - Because of locality, programs tend to access the data at level k more often than they access the data at level k+1.
  - Thus, the storage at level k+1 can be slower, and thus larger and cheaper per bit.
- ***Big Idea:*** **The memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.**

General Cache Concepts


Cache
4
9
10
3
Smaller, faster, more expensive memory caches a subset of the blocks
10
Data is copied in block-sized transfer units
Memory
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Larger, slower, cheaper Memory viewed as partitioned into "blocks"

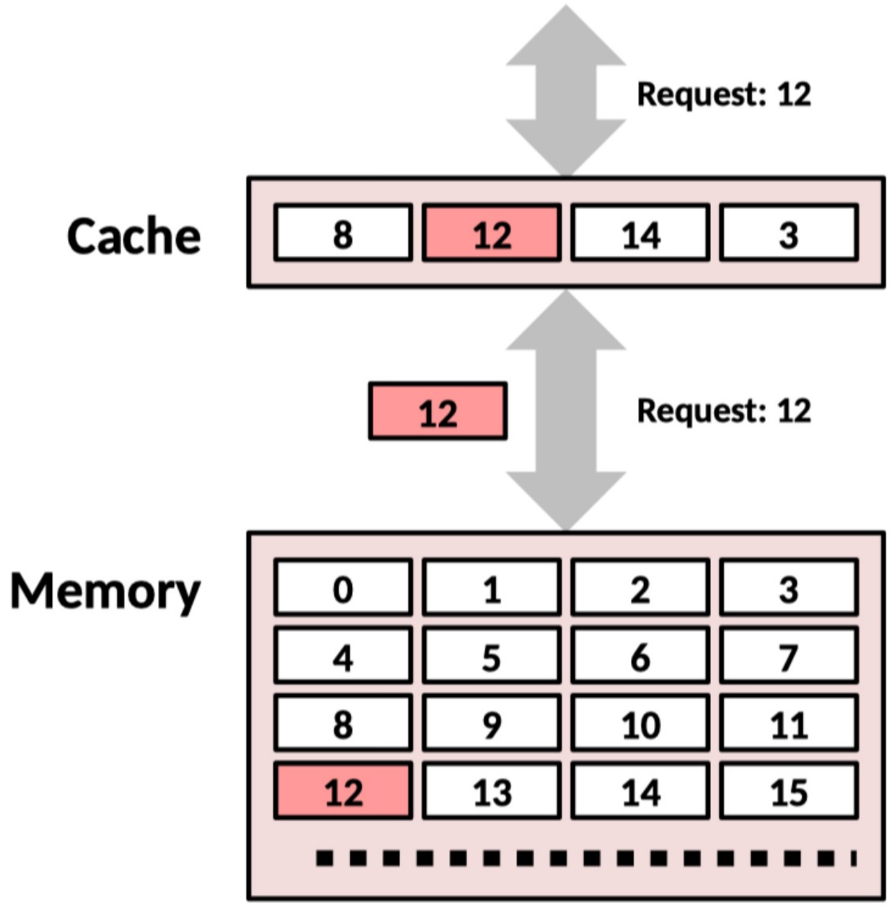
General Cache Concepts: Hit



Data in block b is needed

Block b is in cache:
Hit!

General Cache Concepts: Miss



Data in block b is needed

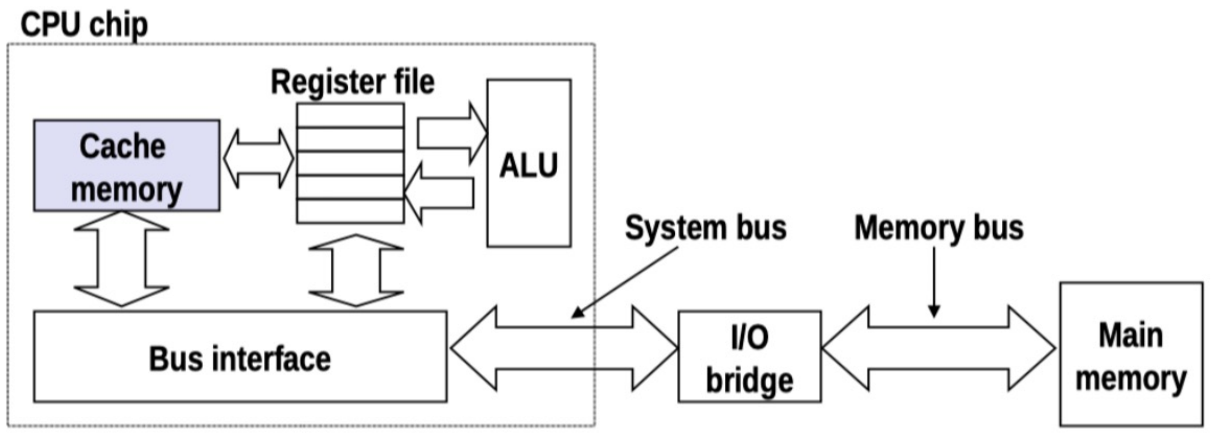
Block b is not in cache:
Miss!

Block b is fetched from memory

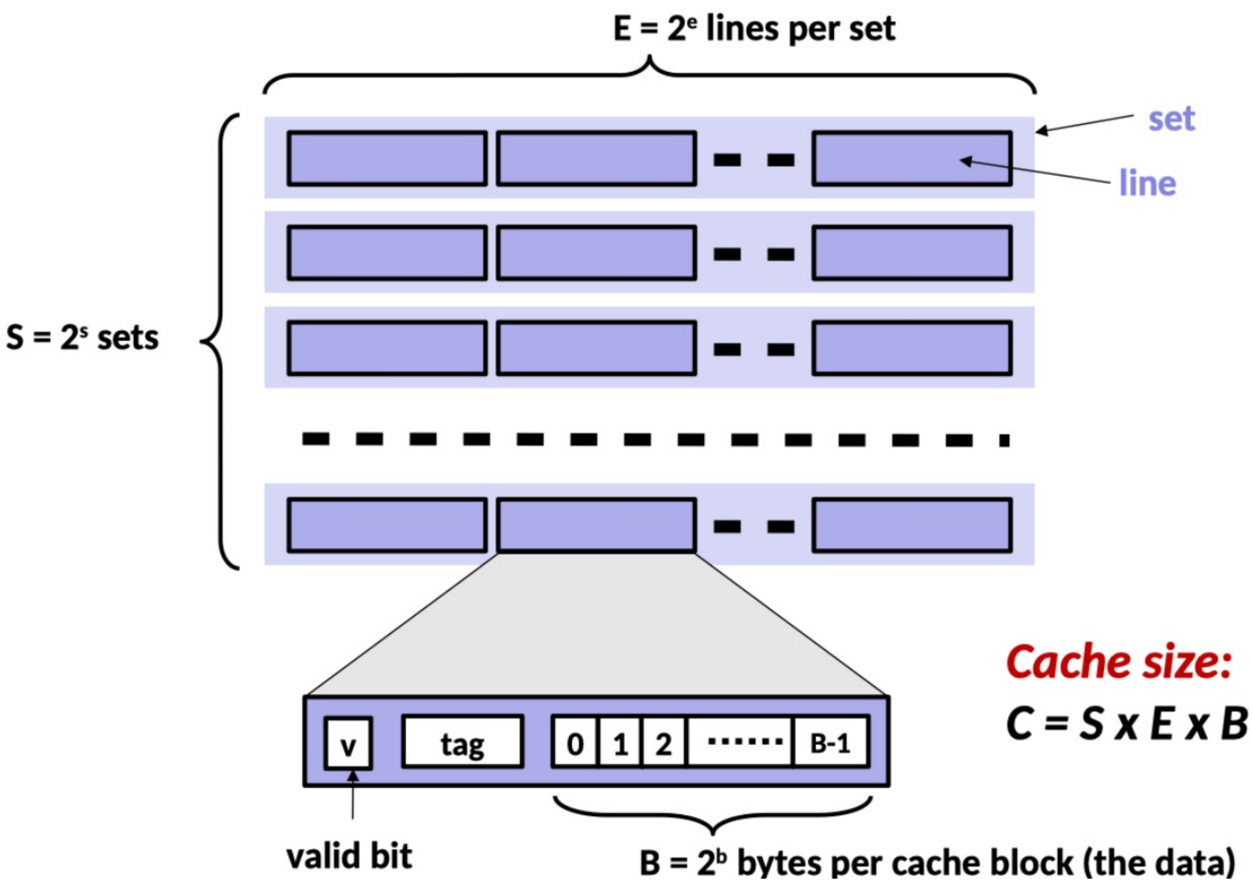
Block b is stored in cache
- Placement policy: determines where b goes
- Replacement policy: determines which block gets evicted (victim)

Cache Memories

- **Cache memories are small, fast SRAM-based memories managed automatically in hardware**
  - Hold frequently accessed blocks of main memory
- **CPU looks first for data in cache**
- **Typical system structure:**



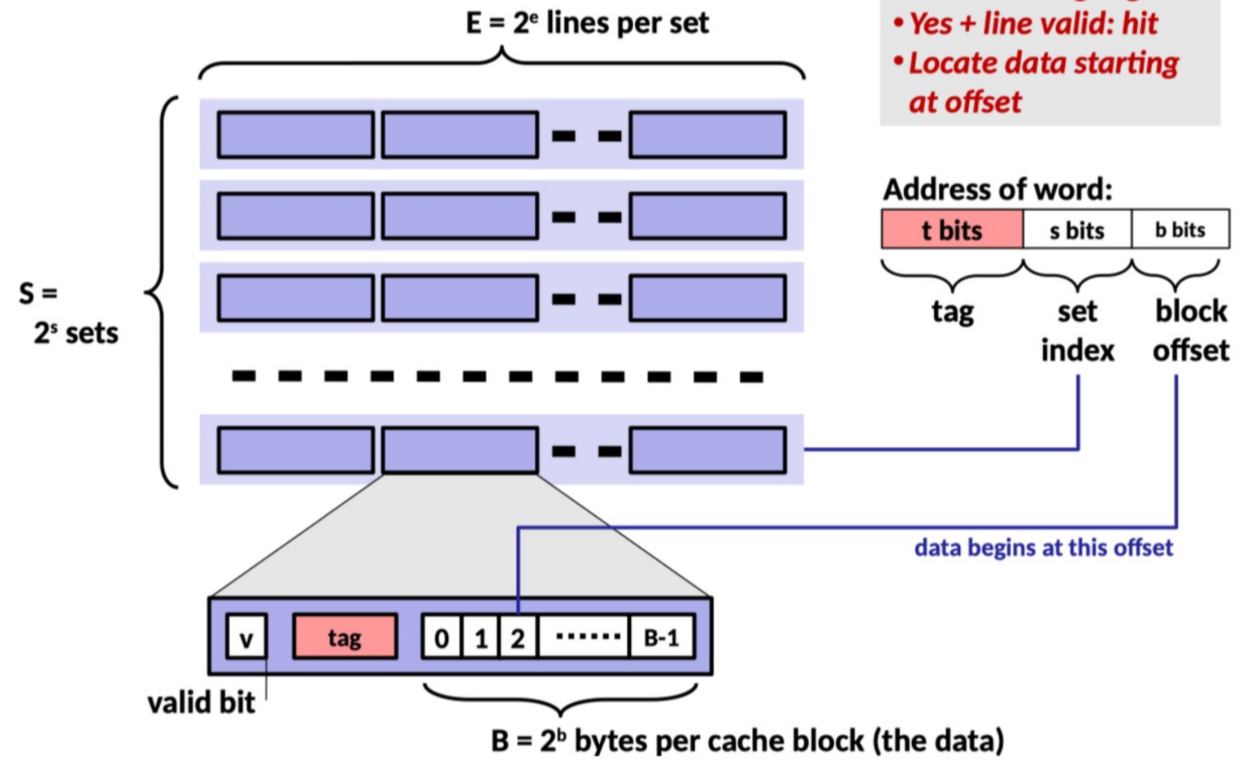
General Cache Organization(S, E, B)



Cache size:

$C = S \times E \times B$ bytes

Cache Read

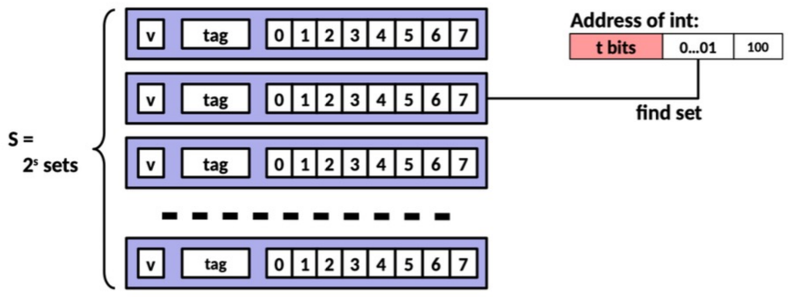


- *Locate set*
- *Check if any line in set has matching tag*
- *Yes + line valid: hit*
- *Locate data starting at offset*

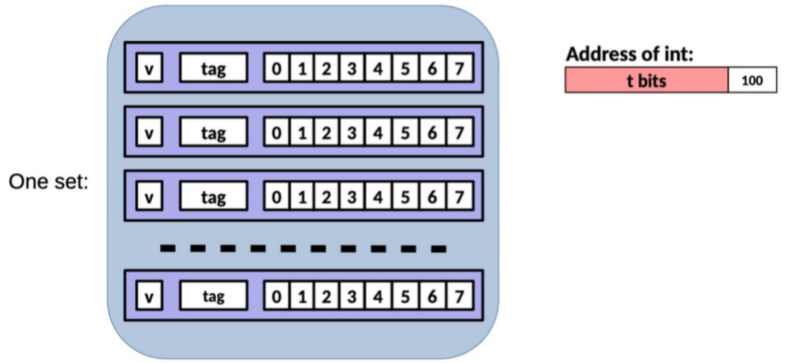
Cache Associativity

- **Direct mapped**
  - E = 1 (1 line per set)
- **n-way associative**
  - n lines per set
- **Fully associative**
  - S = 1 (1 set containing all lines)

Direct mapped: One line per set (E=1)



One set containing all lines (S=1)



Cache Size

- **Suppose we're given:**
  - 32 KB cache
  - 64 B line
  - 4-way set associative
  - 32-bit architecture
- **How many tag, set, block index bits?**
  - 64 B per block = 6 bits
  - 32 KB cache / 64 B per line = 2^9 lines
  - 2^9 lines / 4 lines per set = 2^7 sets
  - So 6 block index bits, 7 set index bits
  - 32 – 6 – 7 = 19 tag bits

Cache Size

- **Suppose we're given:**
  - Read 0x2b74ce2d
- **What are the tag, set, block index?**
  - (19 tag, 7 set, 6 block)
  - 0x2b74ce2d =

    0010 1011 0111 0100 1100 1110 0010 1101
  - Tag = 0010101101110100110
  - Set index = 0111000 (56)
  - Block index = 101101 (45)

Cache Size

- **Suppose we're given:**
  - Read 0x2b74ce2d
- **What are the tag, set, block index?**
  - (19 tag, 7 set, 6 block)
  - 0x2b74ce2d =

    0010 1011 0111 0100 1100 1110 0010 1101
  - Tag = 0010101101110100110
  - Set index = 0111000 (56)
  - Block index = 101101 (45)

PA4

Input: circuit description

```
INPUT 3 a b c
OUTPUT 1 d
AND a b   x
AND c x   d
```

Output: truth table

```
0 0 0 | 0
0 0 1 | 0
0 1 0 | 0
0 1 1 | 0
1 0 0 | 0
1 0 1 | 0
1 1 0 | 0
1 1 1 | 1
```

PA4

Input: circuit description

```
INPUT 3 a b c
OUTPUT 1 d
AND a b   x
AND c x   d
```

- INPUT and OUTPUT: number + variables
- AND, OR, NAND, NOR, XOR, NOT, PASS, DECODER, MULTIPLEXER : input variables + output variables

PA4

Output: truth table

```
0 0 0 | 0
0 0 1 | 0
0 1 0 | 0
0 1 1 | 0
1 0 0 | 0
1 0 1 | 0
1 1 0 | 0
1 1 1 | 1
```

all possible Inputs | outputs

- Generate and print the table one row at one time
- Represent any of the gates.
- Assign values to inputs and get the output of each gate.

Q&A

Thanks!