The Memory Hierarchy and Intro to Caches

CS 105: Computer Systems
Lecture 20

Melissa O'Neill

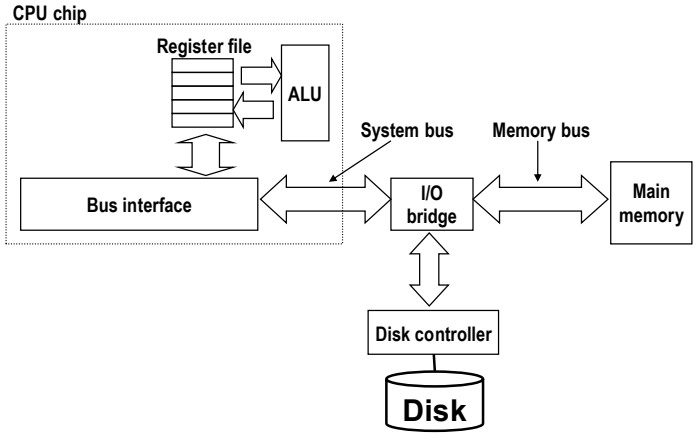
April 8, 2026

Learning Goals

- **Summarize the trade-offs of different types of computer memory**
- **Explain the principle of locality in programs.**
- **Understand how caches are organized and accessed**
  - How to find the data in the cache?

Recall: system architecture

- **Storage (so far): registers, main memory, disk**



Volatile vs. Nonvolatile Memories

- **Main memory is made of *volatile* memory**
  - Lose information if powered off
  - E.g., Dynamic RAM (DRAM) and Static RAM (SRAM)

DRAM used for main memory

- **Nonvolatile memories**
  - Retain value even if powered off
  - E.g., hard disk drive, solid state disks (SSDs)

Img src: https://www.differencebetween.com/difference-between-synchronous-and-asynchronous-dram/

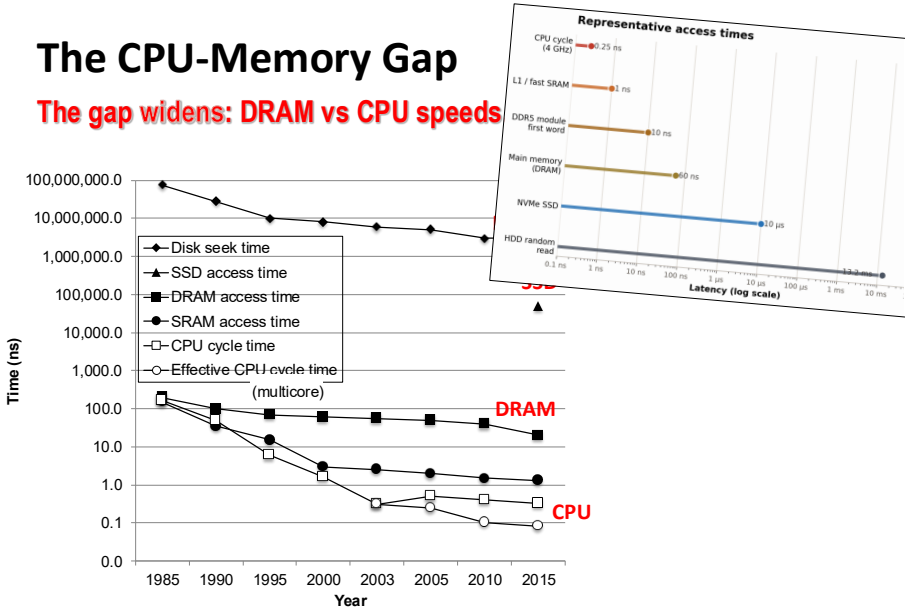
Random-Access Memory (RAM)

- **Key features**
  - RAM is traditionally packaged as a chip
  - One bit of storage per cell
  - Multiple RAM chips form a memory

- **RAM comes in two varieties:**
  - SRAM (Static RAM)
    - Holds what you write as long as you have power
    - Requires more hardware circuitry than DRAM to store a bit → $$$
    - Typically used for cache memories
  - DRAM (Dynamic RAM)
    - Can forget! Stores data in capacitor, periodically re-writes it (many times per second) → slower than SRAM
    - Typically used for main memory

The CPU-Memory Gap

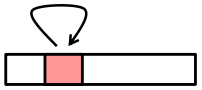
The gap widens: DRAM vs CPU speeds



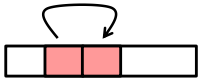
Locality

- **Principle of Locality: Programs tend to use data and instructions with memory 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



Exercise: Locality Example

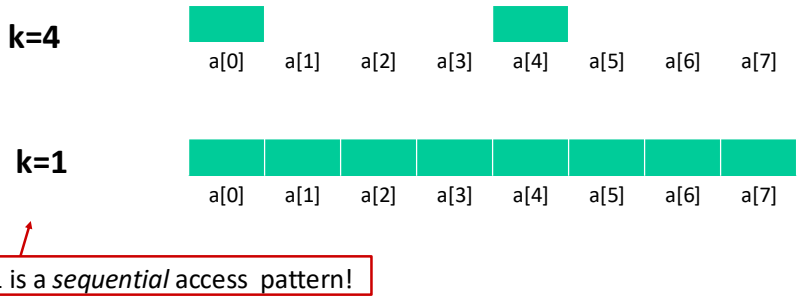
```
…
sum = 0;
for (int i = 0; i < n; ++i)
   sum += a[i];
return sum;
```

- **What spatial and temporal locality do you see?**

Strides

- **Stride-*k* reference pattern: visiting every *k*th element of a contiguous vector**

```
for (i = 0; i < n; i += k)
   sum += a[i];
```



Qualitative Estimates of Locality

- **Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer!**
- **Does this function have good locality with respect to array `a`?**

```
int sum_rows(int a[M][N])
{
  int i, j, sum = 0;

  for (i = 0; i < M; i++)
    for (j = 0; j < N; j++)
      sum += a[i][j];
  return sum;
}
```


Exercise: Locality and stride

- **Does this function have good locality with respect to array `a`? What is the stride?**

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```


Memory Hierarchies

- **Some fundamental properties of hardware and software:**
  - Well-written programs tend to exhibit good locality.
  - The gap between CPU and main memory speed is widening.
  - Fast storage technologies cost more per byte, have less capacity, and require more power (heat!).
- **Observation: these properties *complement* each other!**
- **…. suggests an approach for organizing memory and storage systems known as a memory hierarchy**

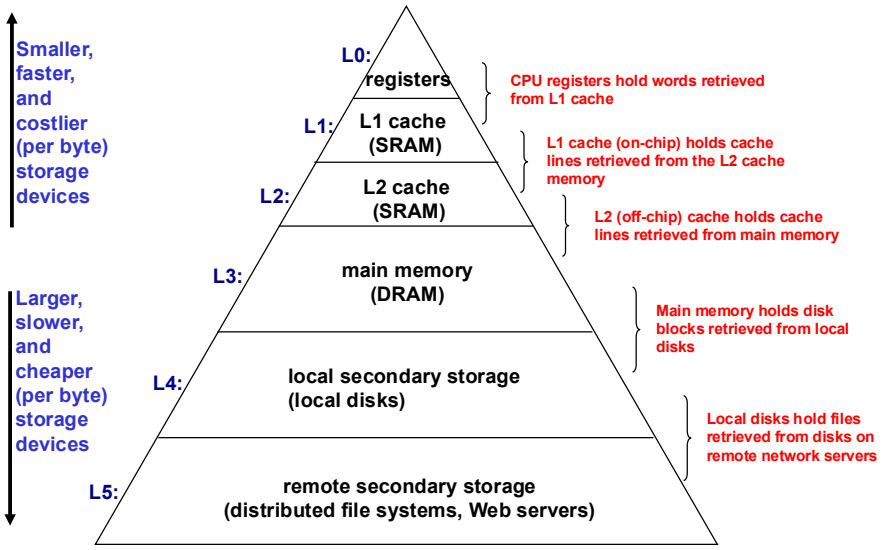
Caches

- ***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.

Notice how "cache" is both a concept and a physical device

An Example Memory Hierarchy



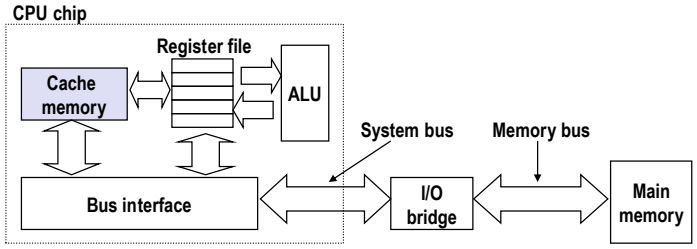
Memory Hierarchy Reflection

- **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.

Cache Memories

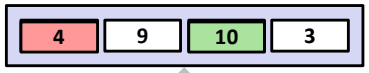
- **Cache memories are small, fast SRAM-based memories managed automatically in hardware**
  - Hold frequently accessed blocks of main memory
- **CPU looks first for data in cache**
- **Typical system structure:**



General Cache Concepts

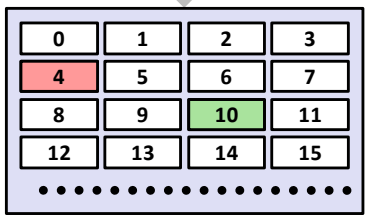
Cache (level *k*)

Smaller, faster, more expensive memory caches a subset of blocks

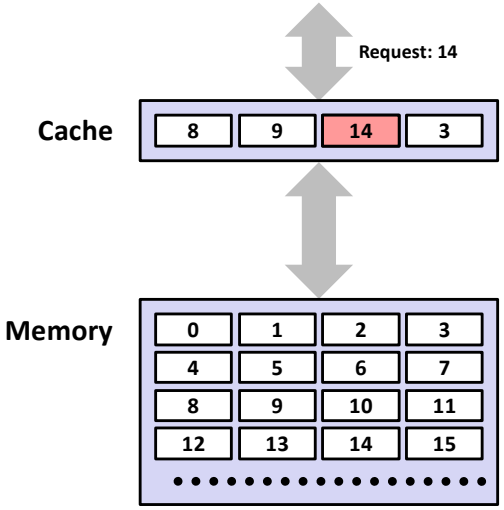


Memory (level *k*+1)

Larger, slower, cheaper memory viewed as partitioned into "blocks"



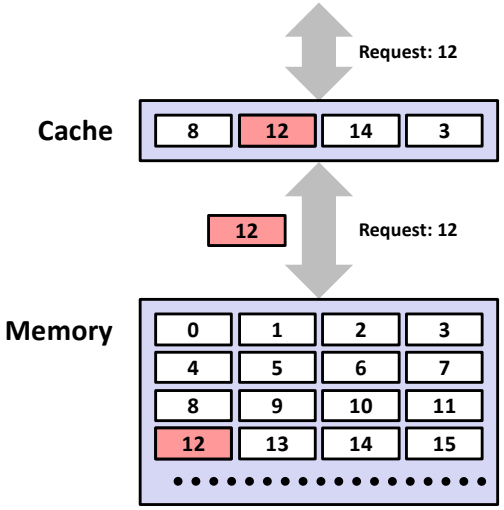
General Cache Concepts: Hit



Data in block 14 is needed

Block 14 is in cache:
Hit!

General Cache Concepts: Miss



Data in block 12 is needed

Block 12 is not in cache:
Miss!

Block 12 is fetched from memory

Block 12 is stored in cache

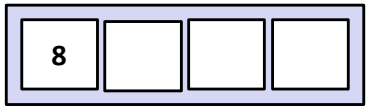
- **Placement policy:** determines where 12 *could* go
- **Replacement policy:** determines which block gets *evicted* from cache (victim) if necessary

Exercise: LRU replacement policy

- Show (on the next slide) the state of the cache after each of the following six requests for blocks:
  12, 14, 3, 10, 12, 5

- Assumptions
  - Replacement policy: Least Recently Used (LRU)
  - Placement policy: no restrictions (any of the four spots)

Cache (initial state)



Memory

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Exercise: LRU replacement policy (contd.)

After 1st request

After 2nd request

After 3rd request

After 4th request

After 5th request

After 6th request

Checking for data in the cache

Cache

8	9	14	3

Memory

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
• • • • • • • • • • • • • • • •			

Program wants data at address A

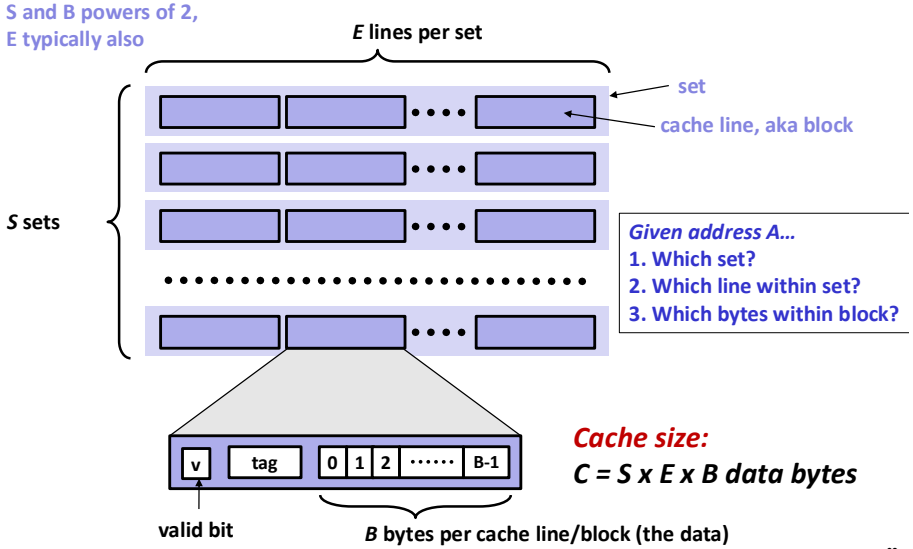
Is the cache storing that data?
How should we locate it? Guiding questions:

1. If data referenced by A was in the cache, in which spot(s) could it be?
2. Given the spot(s), is the data referenced by A actually stored in one of the spots?
3. Given that the data referenced by A is stored in the block at that spot, which part of the block is it?

Idea: use the address A *itself* to answer these questions!

General Cache Organization (S, E, B)

S and B powers of 2,
E typically also



Given address A…
1. Which set?
2. Which line within set?
3. Which bytes within block?

Cache size:
C = S x E x B data bytes

Cache Read

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