COMP201

Computer Systems & Programming

Lecture #29 – Locality, The Memory Hierarchy
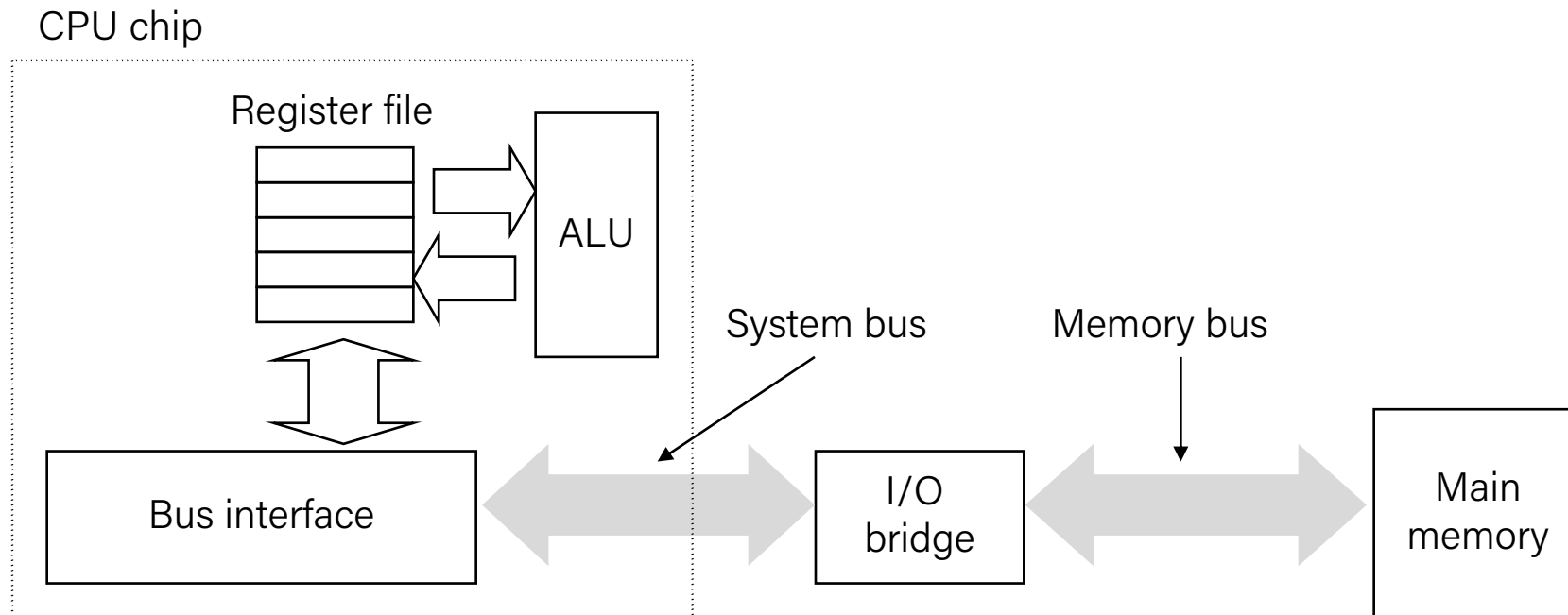
Aykut Erdem // Koç University // Fall 2020

Recap

- Buffer Overflow Attacks and What to Do About Them

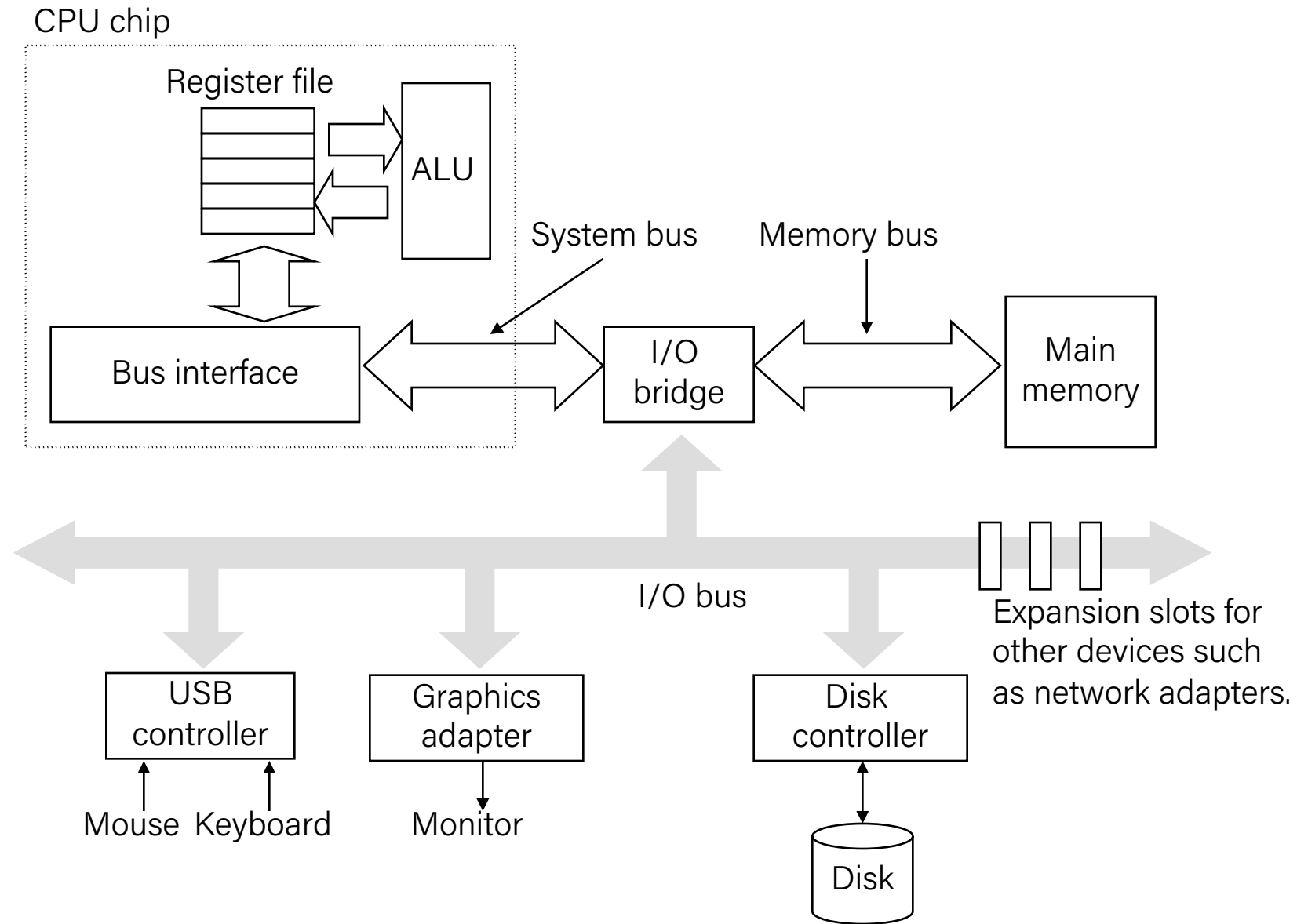- Storage technologies and trends

Recap: Nonvolatile Memories

- **DRAM and SRAM are volatile memories**
  - Lose information if powered off.
- **Nonvolatile memories retain value even if powered off**
  - Read-only memory (**ROM**): programmed during production
  - Programmable ROM (**PROM**): can be programmed once
  - Eraseable PROM (**EPROM**): can be bulk erased (UV, X-Ray)
  - Electrically eraseable PROM (**EEPROM**): electronic erase capability
  - Flash memory: EEPROMs. with partial (block-level) erase capability
    - Wears out after about 100,000 erasings
- **Uses for Nonvolatile Memories**
  - Firmware programs stored in a ROM (BIOS, controllers for disks, network cards, graphics accelerators, security subsystems,…)
  - Solid state disks (replace rotating disks in thumb drives, smart phones, mp3 players, tablets, laptops,…)
  - Disk caches

Recap: Traditional Bus Structure Connecting CPU and Memory

- A bus is a collection of parallel wires that carry address, data, and control signals.

- Buses are typically shared by multiple devices.

CPU chip

Register file

ALU

System bus

Memory bus

Bus interface

I/O bridge

Main memory

Recap: I/O Bus

CPU chip

Register file

ALU

System bus

Memory bus

Bus interface

I/O bridge

Main memory

I/O bus

USB controller

Graphics adapter

Disk controller

Expansion slots for other devices such as network adapters.

Mouse Keyboard

Monitor

Disk

Recap: Disk Access Time

- Average time to access some target sector approximated by:
  - Taccess = Tavg seek + Tavg rotation + Tavg transfer
- Seek time (Tavg seek)
  - Time to position heads over cylinder containing target sector.
  - Typical Tavg seek is 3—9 ms
- Rotational latency (Tavg rotation)
  - Time waiting for first bit of target sector to pass under r/w head.
  - Tavg rotation = 1/2 $\times$ 1/RPMs x 60 sec/1 min
  - Typical Tavg rotation = 7200 RPMs
- Transfer time (Tavg transfer)
  - Time to read the bits in the target sector.
  - Tavg transfer = 1/RPM $\times$ 1/(avg # sectors/track) $\times$ 60 secs/1 min.

Recap: Disk Access Time Example

- Given:
  - Rotational rate = 7,200 RPM
  - Average seek time = 9 ms.
  - Avg # sectors/track = 400.
- Derived:
  - Tavg rotation = 1/2 × (60 secs/7200 RPM) × 1000 ms/sec = 4 ms.
  - Tavg transfer = 60/7200 RPM × 1/400 secs/track × 1000 ms/sec = 0.02 ms
  - Taccess = 9 ms + 4 ms + 0.02 ms
- Important points:
  - Access time dominated by seek time and rotational latency.
  - First bit in a sector is the most expensive, the rest are free.
  - SRAM access time is about 4 ns/doubleword, DRAM about 60 ns
    - Disk is about 40,000 times slower than SRAM,
    - 2,500 times slower then DRAM.

Recap: Solid State Disks (SSDs)
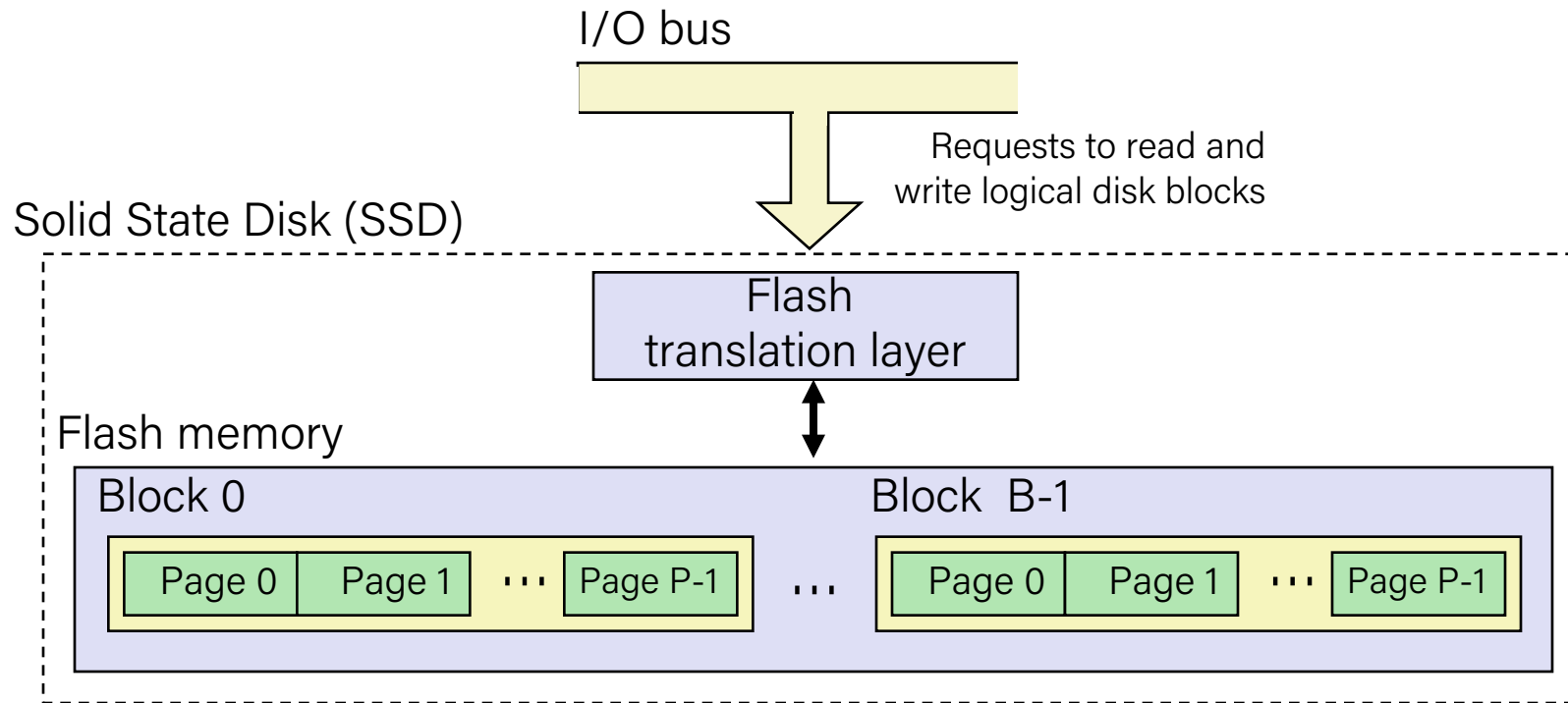
- Pages: 512KB to 4KB, Blocks: 32 to 128 pages

- Data read/written in units of pages.

- Page can be written only after its block has been erased

- A block wears out after about 100,000 repeated writes.

Recap: SSD Tradeoffs vs Rotating Disks

- **Advantages**
  - No moving parts → faster, less power, more rugged
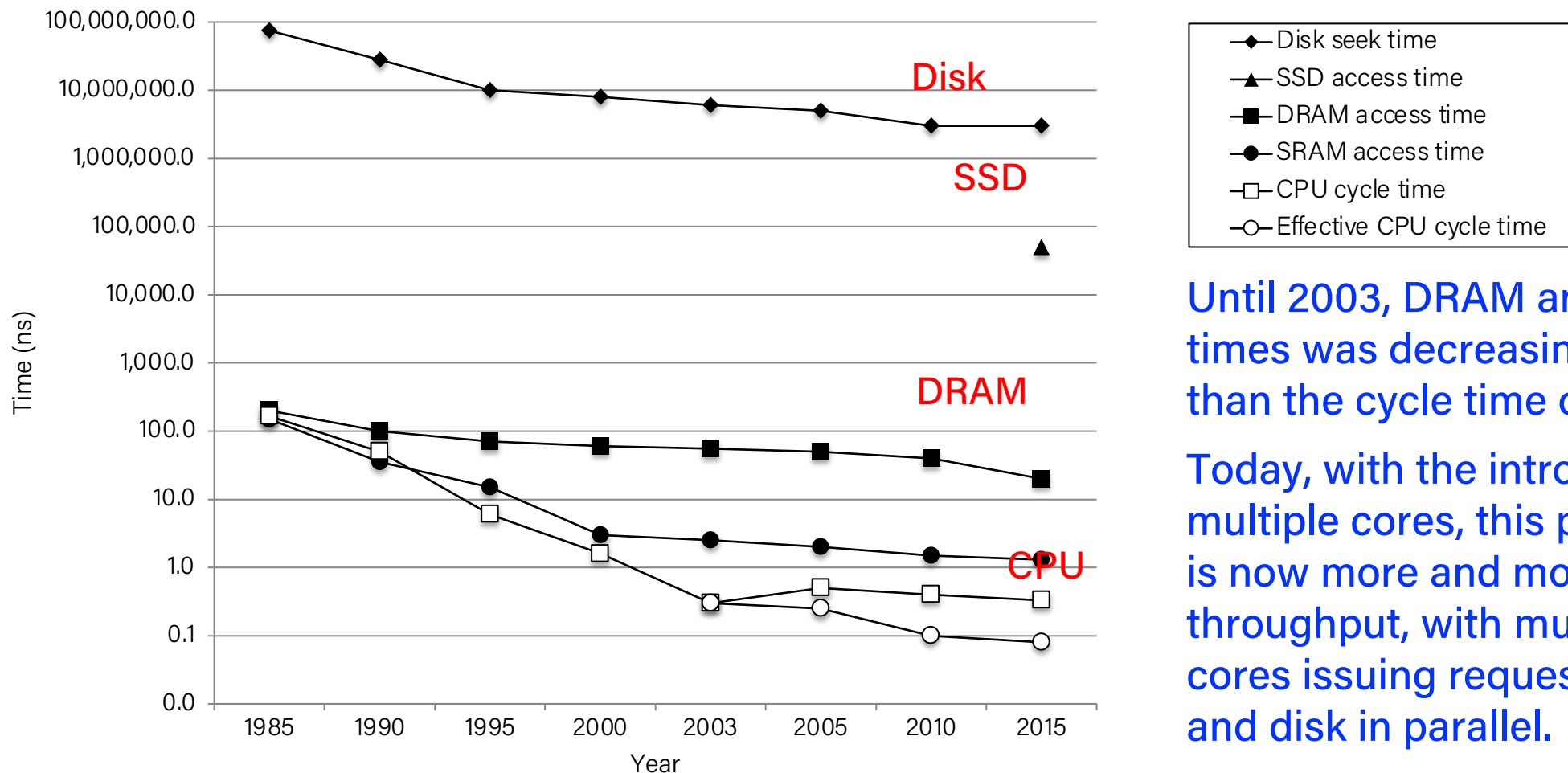
- **Disadvantages**
  - Have the potential to wear out
    - Mitigated by "wear leveling logic" in flash translation layer
    - E.g. Intel SSD 730 guarantees 128 petabyte (128 x 1015 bytes) of writes before they wear out
  - In 2015, about 30 times more expensive per byte

- **Applications**
  - MP3 players, smart phones, laptops
  - Beginning to appear in desktops and servers

Recap: The CPU-Memory Gap

- The gap widens between DRAM, disk, and CPU speeds.

Legend:
- Disk seek time
- SSD access time
- DRAM access time
- SRAM access time
- CPU cycle time
- Effective CPU cycle time

Until 2003, DRAM and disk access times was decreasing more slowly than the cycle time of a processor.

Today, with the introduction of multiple cores, this performance gap is now more and more a function of throughput, with multiple processor cores issuing requests to the DRAM and disk in parallel.

Locality to the Rescue!

- The key to bridging this CPU-Memory gap is a fundamental property of computer programs known as locality

Lecture Plan

- Locality of reference
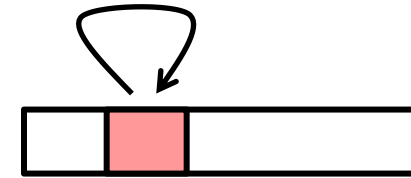- Caching in the memory hierarchy

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
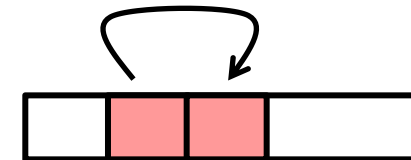
Locality Example

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

- Data references
  - Reference array elements in succession (stride-1 reference pattern). Spatial locality
  - Reference variable sum each iteration. Temporal locality
- Instruction references
  - Reference instructions in sequence. Spatial locality
  - Cycle through loop repeatedly. Temporal locality

Qualitative Estimates of Locality

- **Claim:** Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer.

- **Question:** Does this function have good locality with respect to array a?

```
int sum_array_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;
}
```

Locality Example 1

- Does this function have good locality with respect to array a?

```
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;
}
```

Locality Example 2

- Can you permute the loops so that the function scans the 3-d array a with a stride-1 reference pattern (and thus has good spatial locality)?

```
int sum_array_3d(int a[N][N][N])
{
    int i, j, k, sum = 0;

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

Memory Hierarchies

- Some fundamental and enduring properties of hardware and software:
  - Fast storage technologies cost more per byte, have less capacity, and require more power (heat!).
  - The gap between CPU and main memory speed is widening.
  - Well-written programs tend to exhibit good locality.

- These fundamental properties complement each other beautifully.

- They suggest an approach for organizing memory and storage systems known as a **memory hierarchy**.

Lecture Plan

• Locality of reference

• Caching in the memory hierarchy

Example Memory Hierarchy

Smaller, faster, and costlier (per byte) storage devices

Larger, slower, and cheaper (per byte) storage devices

L0: Regs

L1: L1 cache (SRAM)

L2: L2 cache (SRAM)

L3: L3 cache (SRAM)

L4: Main memory (DRAM)

L5: Local secondary storage (local disks)

L6: Remote secondary storage (e.g., Web servers)

CPU registers hold words retrieved from the L1 cache.

L1 cache holds cache lines retrieved from the L2 cache.

L2 cache holds cache lines retrieved from L3 cache

L3 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 servers
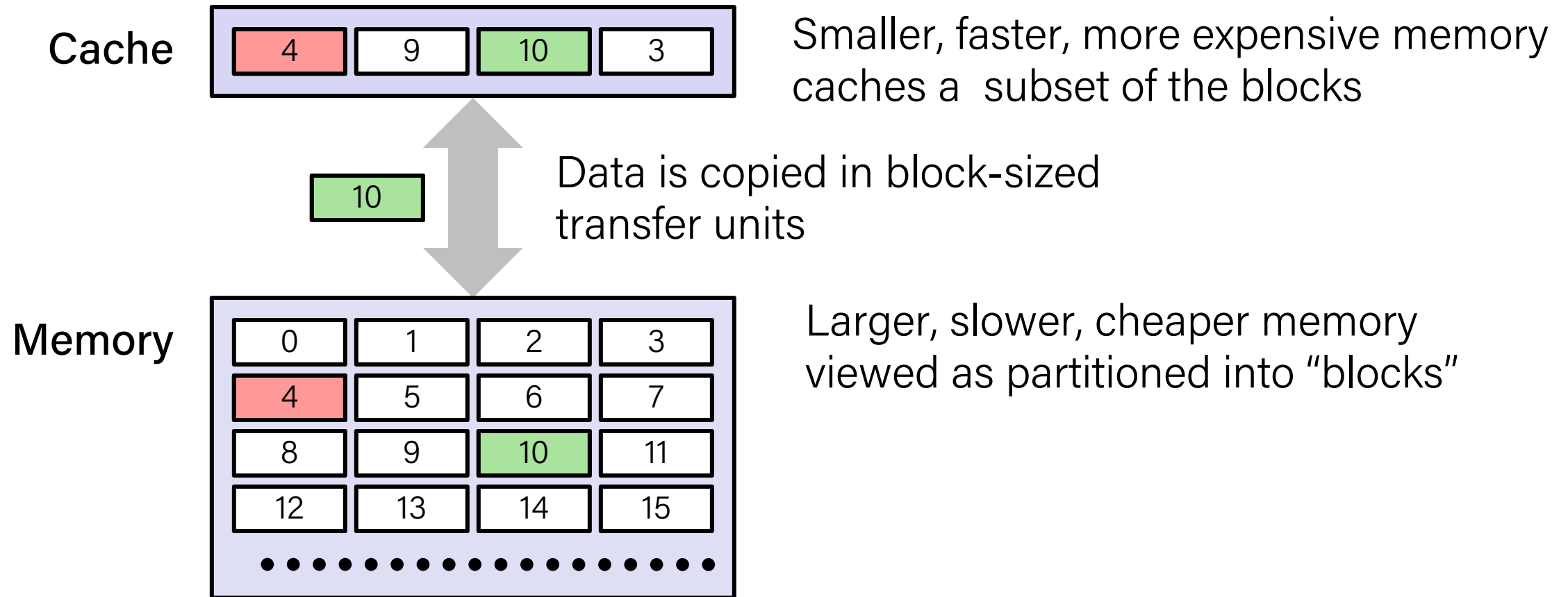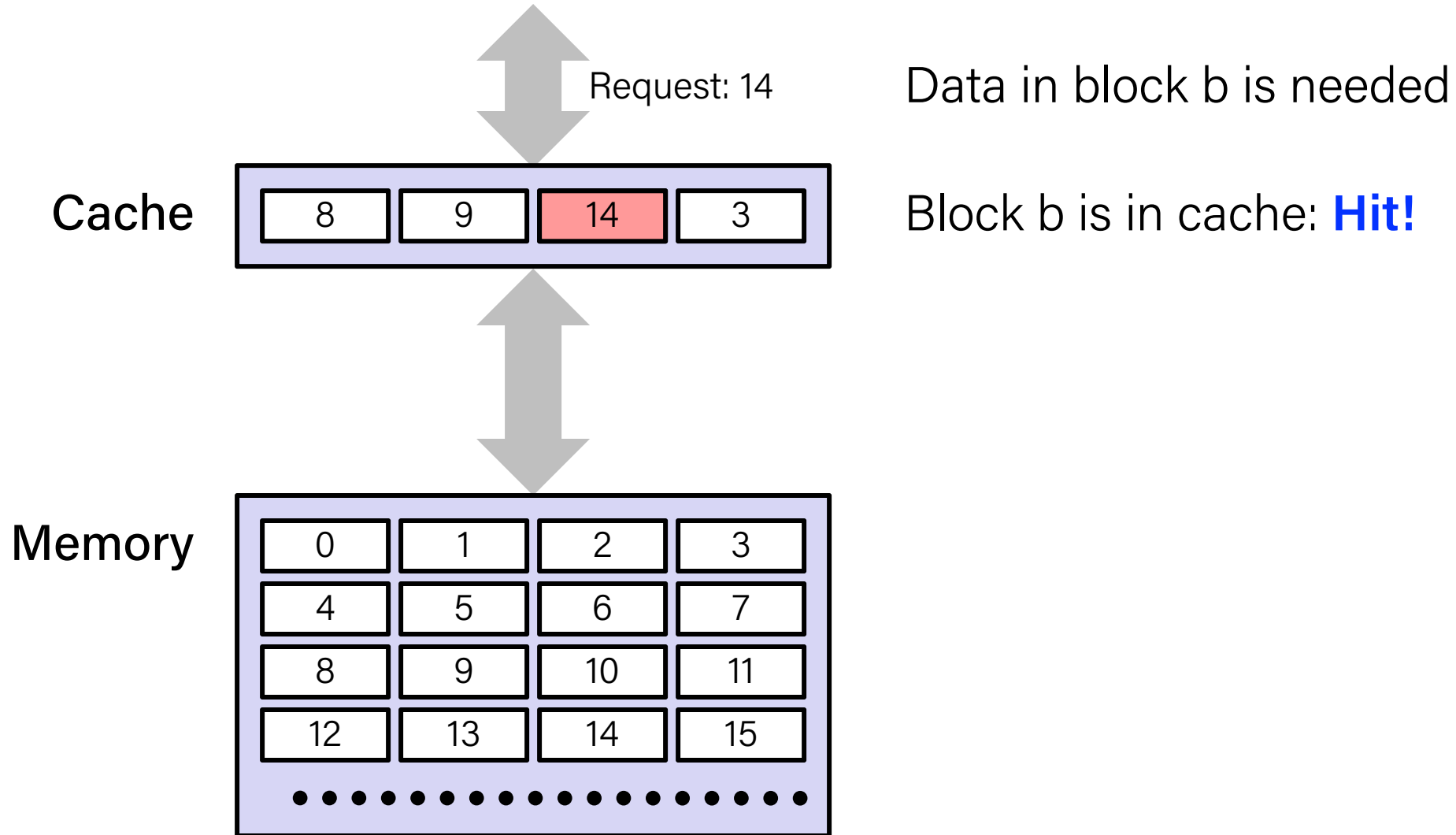
20

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

Request: 14

Cache

8	9	14	3

Memory

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

Data in block b is needed

Block b is in cache: **Hit!**

General Cache Concepts: Miss

Request: 12

| 8 | 12 | 14 | 3 |

Cache

12

Request: 12

Memory

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

• • • • • • • • • • • • • • • • • • • •
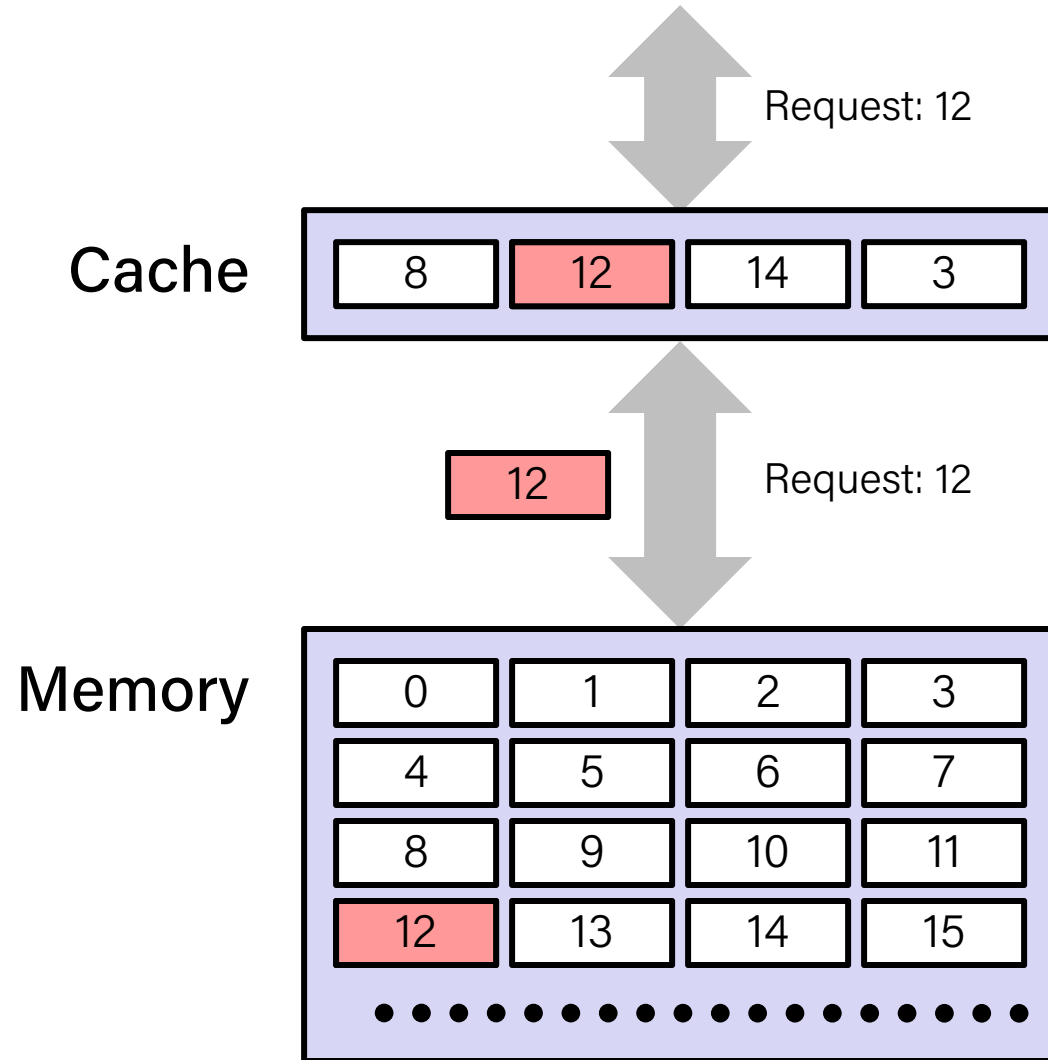
Data in block b is needed

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)

Types of Cache Misses

- **Cold (compulsory) miss**
  - Cold misses occur because the cache is empty.
- **Conflict miss**
  - Most caches limit blocks at level k+1 to a small subset (sometimes a singleton) of the block positions at level k.
    - E.g. Block i at level k+1 must be placed in block (i mod 4) at level k.
  - Conflict misses occur when the level k cache is large enough, but multiple data objects all map to the same level k block.
    - E.g. Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time.
- **Capacity miss**
  - Occurs when the set of active cache blocks (working set) is larger than the cache.

Examples of Caching in the Mem. Hierarchy

Cache Type	What is Cached?	Where is it Cached?	Latency (cycles)	Managed By
Registers	4-8 bytes words	CPU core	0	Compiler
TLB	Address translations	On-Chip TLB	0	Hardware MMU
L1 cache	64-byte blocks	On-Chip L1	4	Hardware
L2 cache	64-byte blocks	On-Chip L2	10	Hardware
Virtual Memory	4-KB pages	Main memory	100	Hardware + OS
Buffer cache	Parts of files	Main memory	100	OS
Disk cache	Disk sectors	Disk controller	100,000	Disk firmware
Network buffer cache	Parts of files	Local disk	10,000,000	NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

Summary

- The speed gap between CPU, memory and mass storage continues to widen.

- Well-written programs exhibit a property called locality.

- Memory hierarchies based on caching close the gap by exploiting locality.

Recap

- Locality of reference
- Caching in the memory hierarchy

Next: Cache memories