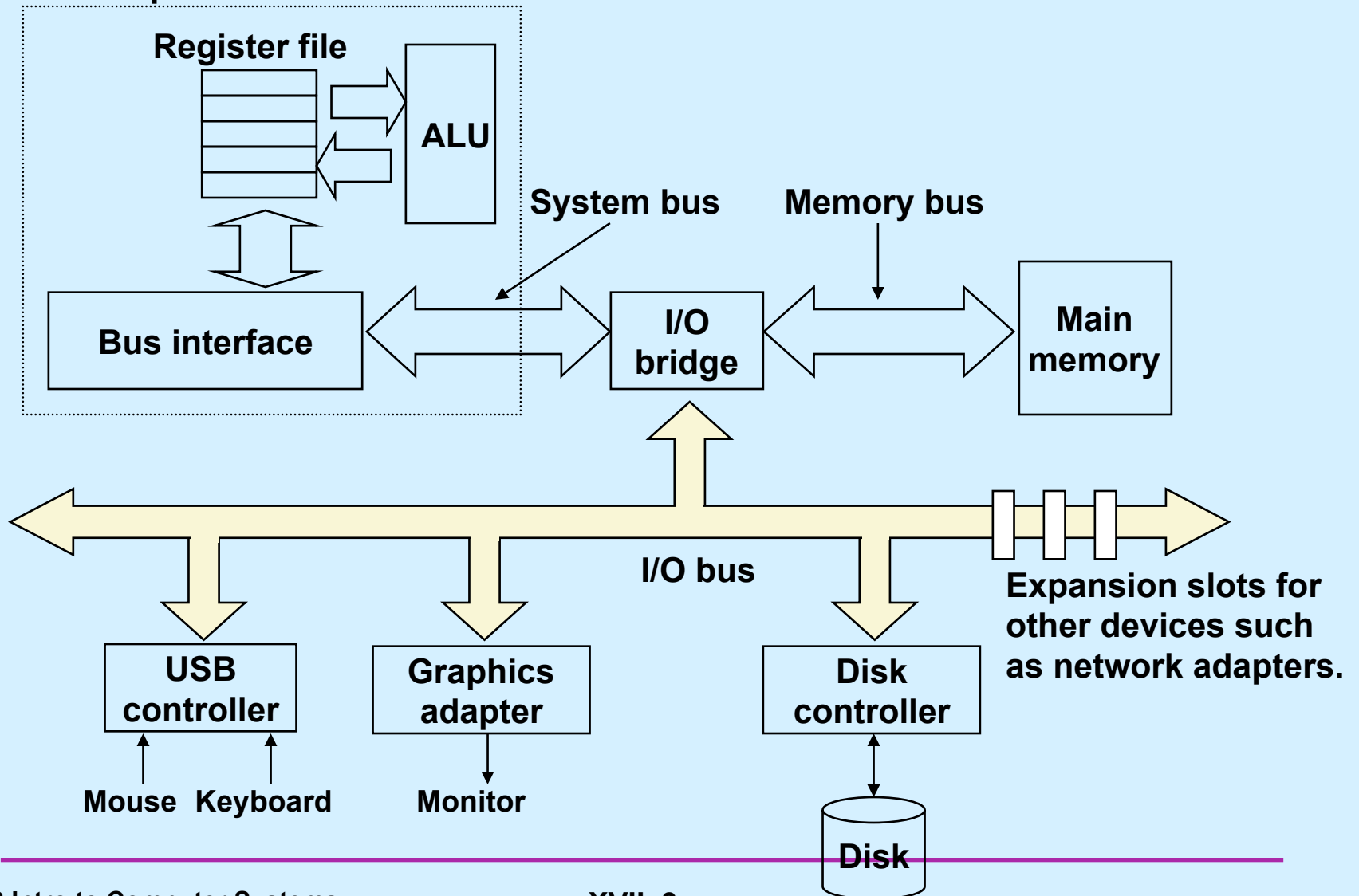# CS 33

## Memory Hierarchy II

# Logical Disk Blocks

- **Modern disks present a simpler abstract view of the complex sector geometry:**
  - the set of available sectors is modeled as a sequence of b-sized **logical blocks** (0, 1, 2, ...)

- **Mapping between logical blocks and actual (physical) sectors**
  - maintained by hardware/firmware device called disk controller
  - converts requests for logical blocks into (surface, track, sector) triples

- **Allows controller to set aside spare cylinders for each zone**
  - accounts for the difference in "formatted capacity" and "maximum capacity"
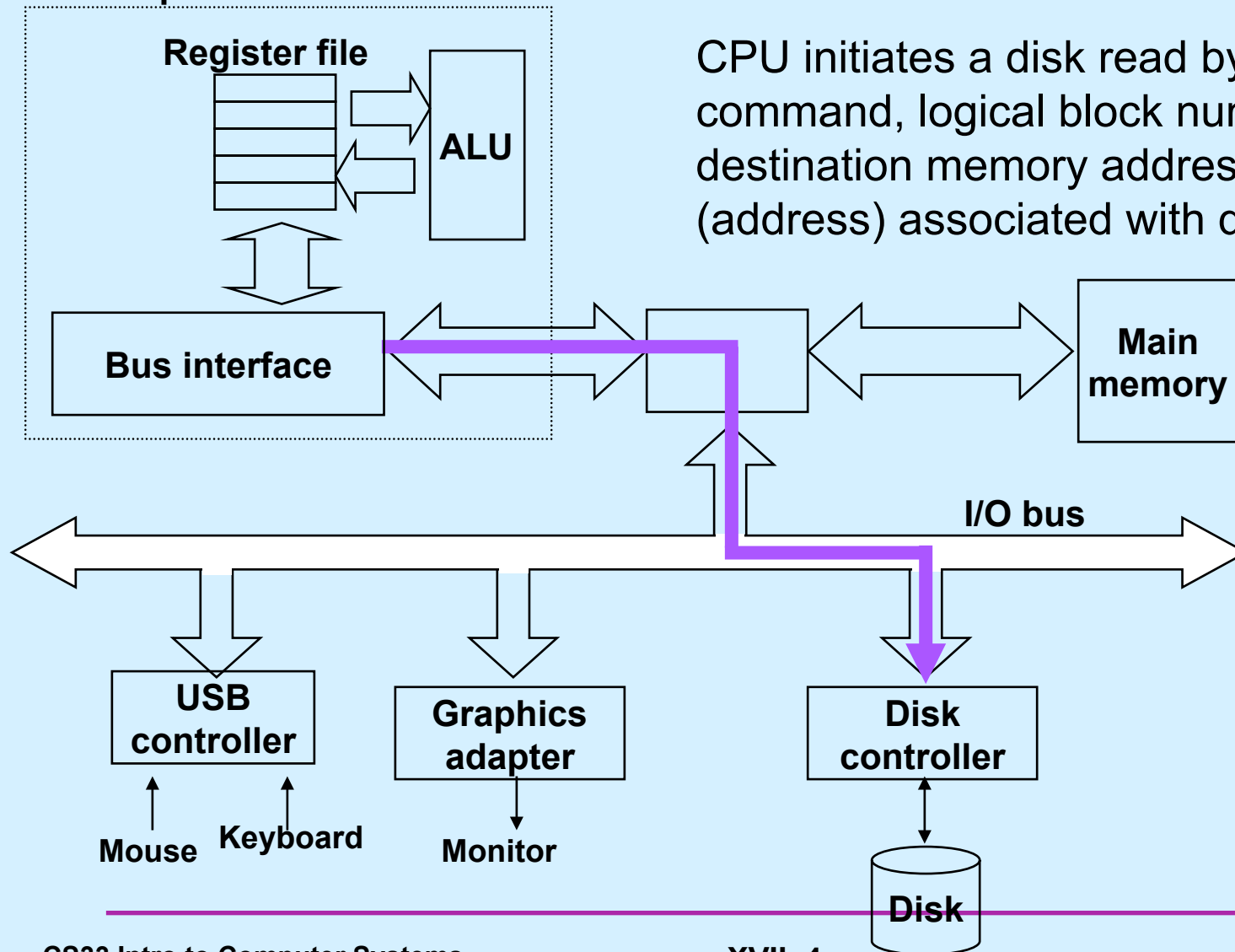
# I/O Bus

**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**
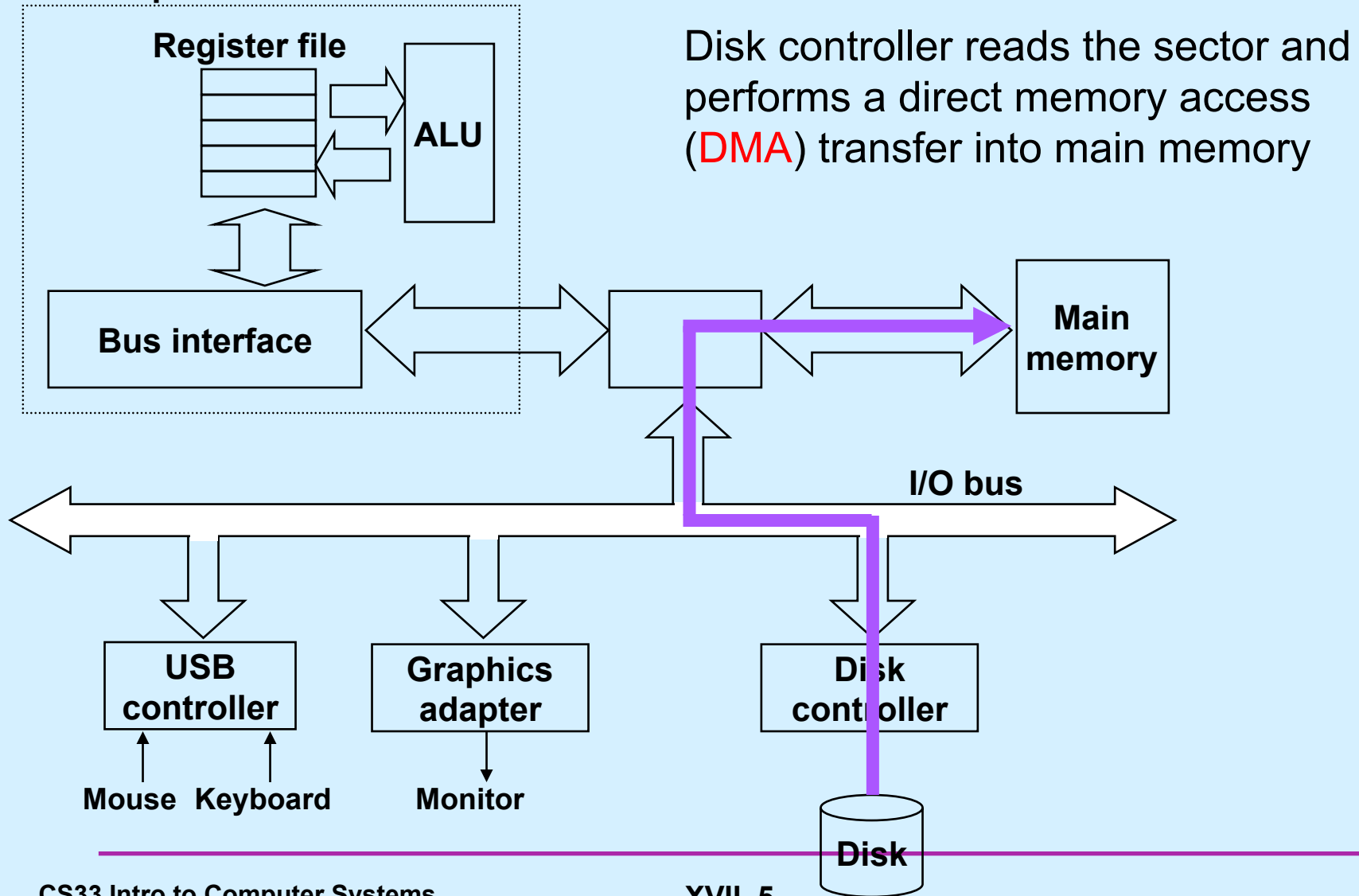
# Reading a Disk Sector (1)

**CPU chip**

**Register file**

**ALU**

CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller
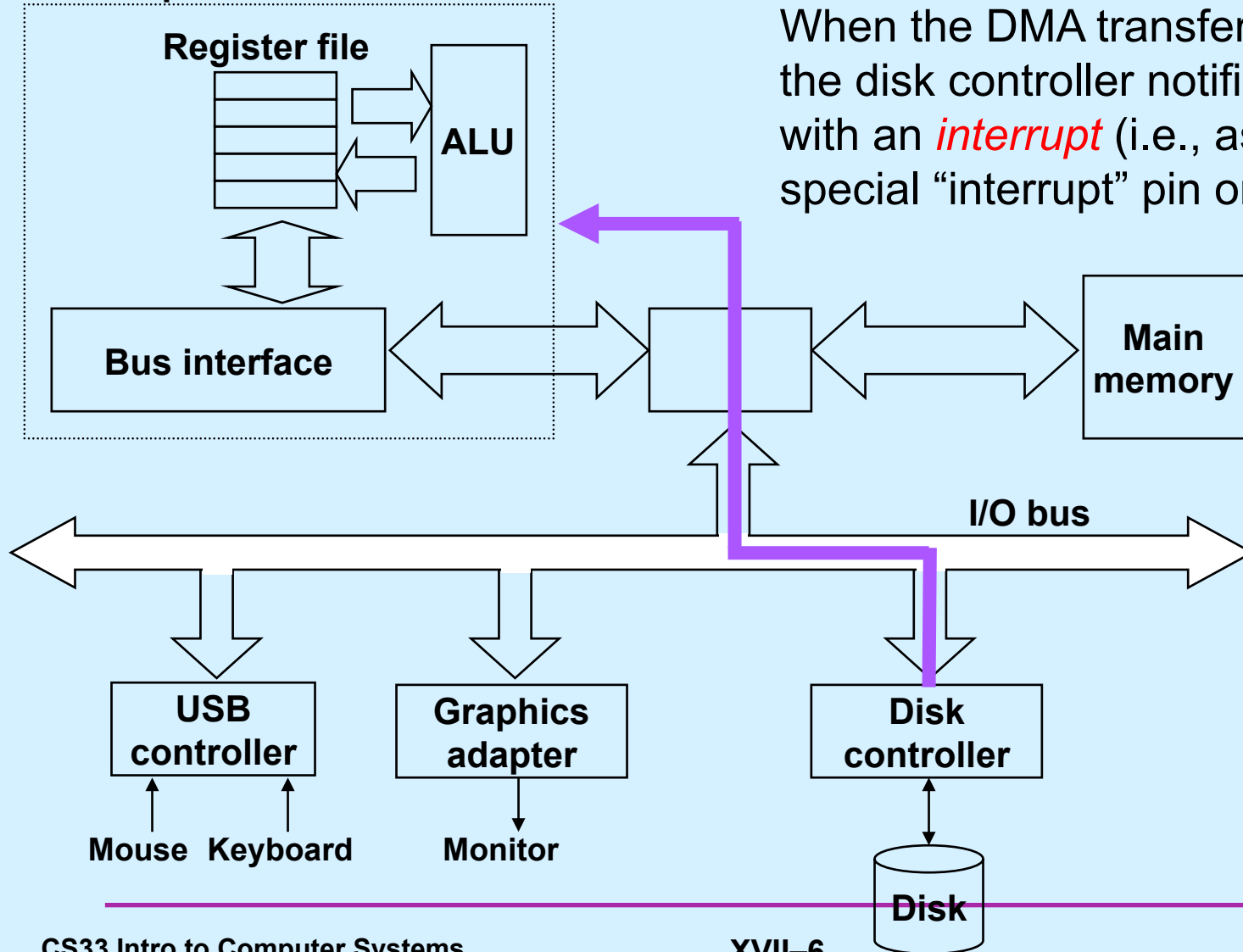
**Bus interface**

**Main memory**

**I/O bus**

**USB controller**

**Graphics adapter**

**Disk controller**

Mouse    Keyboard    Monitor

**Disk**

# Reading a Disk Sector (2)

**CPU chip**

**Register file**

**ALU**

Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory

**Bus interface**

**Main memory**

**I/O bus**

**USB controller**

**Graphics adapter**

**Disk controller**

**Mouse** **Keyboard**

**Monitor**

**Disk**

# Reading a Disk Sector (3)

**CPU chip**

**Register file**

**ALU**

**Bus interface**

**Main memory**

**I/O bus**

**USB controller**

Mouse  Keyboard

**Graphics adapter**

Monitor

**Disk controller**

**Disk**

When the DMA transfer completes, the disk controller notifies the CPU with an *interrupt* (i.e., asserts a special "interrupt" pin on the CPU)

# Solid-State Disks (SSDs)

I/O bus

Requests to read and write logical disk blocks

Solid State Disk (SSD)

Flash translation layer

Flash memory

Block 0

| Page 0 | Page 1 | · · · | Page P-1 |

· · ·

Block  B-1

| Page 0 | Page 1 | · · · | Page P-1 |

- **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 100,000 repeated writes**

# SSD Performance Characteristics

| | | | |
|---|---|---|---|
| Sequential read tput | 250 MB/s | Sequential write tput | 170 MB/s |
| Random read tput | 140 MB/s | Random write tput | 14 MB/s |
| Random read access | 30 us | Random write access | 300 us |

- ## Why are random writes so slow?
  - erasing a block is slow (around 1 ms)
  - modifying a page triggers a copy of all useful pages in the block
    - » find a used block (new block) and erase it
    - » write the page into the new block
    - » copy other pages from old block to the new block

# 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 X25 guarantees 1 petabyte ($10^{15}$ bytes) of random writes before they wear out
  - in 2010, about 100 times more expensive per byte
- **Applications**
  - MP3 players, smart phones, laptops
  - beginning to appear in desktops and servers

# Reading a File on a Rotating Disk

- **Suppose the data of a file is stored on consecutive disk sectors on one track**
  - **this is the best possible scenario for reading data quickly**
    - » **single seek required**
    - » **single rotational delay**
    - » **all sectors read in a single scan**
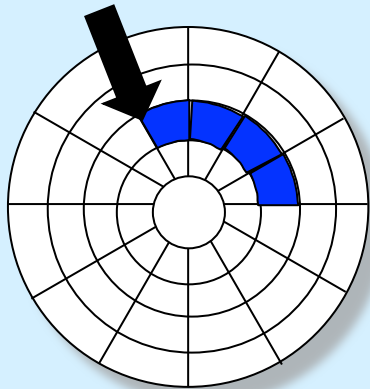
# Quiz 1

We have two files on the same (rotating) disk. The first file's data resides in consecutive sectors on one track, the second in consecutive sectors on another track. It takes a total of *t* seconds to read all of the first file then all of the second file.

Now suppose the files are read concurrently, perhaps a sector of the first, then a sector of the second, then the first, then the second, etc. Compared to reading them sequentially, this will take

- a) less time
- b) about the same amount of time
- c) more time
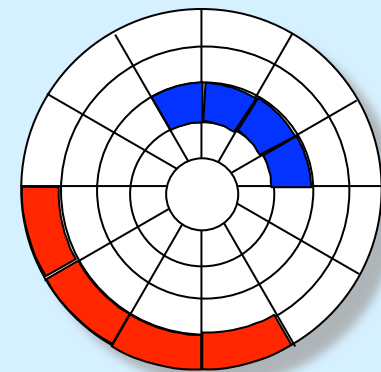
# Quiz 2

**You've replaced the rotating disk on your computer with a solid-state disk. The data of the two files are again in consecutive locations. Suppose it still takes a total of *t* seconds to read the first file then the second. Suppose it took *u* seconds to read the two files concurrently on the rotating disk. It takes *v* seconds to read them concurrently on the SSD.**

   *a) v < u* **(faster on the SSD)**

   *b) v ≈ u* **(about the same)**

   *c) v > u* **(slower on the SSD)**

# Storage Trends

**SRAM**

| Metric | 1980 | 1985 | 1990 | 1995 | 2000 | 2005 | 2010 | *2010:1980* |
|---|---|---|---|---|---|---|---|---|
| $/MB | 19,200 | 2,900 | 320 | 256 | 100 | 75 | 60 | *320* |
| access (ns) | 300 | 150 | 35 | 15 | 3 | 2 | 1.5 | *200* |

**DRAM**

| Metric | 1980 | 1985 | 1990 | 1995 | 2000 | 2005 | 2010 | *2010:1980* |
|---|---|---|---|---|---|---|---|---|
| $/MB | 8,000 | 880 | 100 | 30 | 1 | 0.1 | 0.06 | *130,000* |
| access (ns) | 375 | 200 | 100 | 70 | 60 | 50 | 40 | *9* |
| typical size (MB) | 0.064 | 0.256 | 4 | 16 | 64 | 2,000 | 8,000 | *125,000* |

**Disk**

| Metric | 1980 | 1985 | 1990 | 1995 | 2000 | 2005 | 2010 | *2010:1980* |
|---|---|---|---|---|---|---|---|---|
| $/MB | 500 | 100 | 8 | 0.30 | 0.01 | 0.005 | 0.0003 | *1,600,000* |
| access (ms) | 87 | 75 | 28 | 10 | 8 | 4 | 3 | *29* |
| typical size (MB) | 1 | 10 | 160 | 1,000 | 20,000 | 160,000 | 1,500,000 | *1,500,000* |

# CPU Clock Rates

Inflection point in computer history when designers hit the "Power Wall"

| | 1980 | 1990 | 1995 | 2000 | 2003 | 2005 | 2010 | *2010:1980* |
|---|---|---|---|---|---|---|---|---|
| CPU | 8080 | 386 | Pentium | P-III | P-4 | Core 2 | Core i7 | --- |
| Clock rate (MHz) | 1 | 20 | 150 | 600 | 3300 | 2000 | 2500 | 2500 |
| Cycle time (ns) | 1000 | 50 | 6 | 1.6 | 0.3 | 0.50 | 0.4 | 2500 |
| Cores | 1 | 1 | 1 | 1 | 1 | 2 | 4 | 4 |
| Effective cycle time (ns) | 1000 | 50 | 6 | 1.6 | 0.3 | 0.25 | 0.1 | 10,000 |

# The CPU-Memory Gap

**The gap widens between DRAM, disk, and CPU speeds**



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

Labels on chart: Disk, SSD, DRAM, CPU

**Light travels ~11 inches in a nanosecond!**

Y-axis: ns

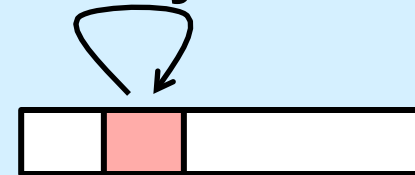X-axis: Year (1980, 1985, 1990, 1995, 2000, 2003, 2005, 2010)
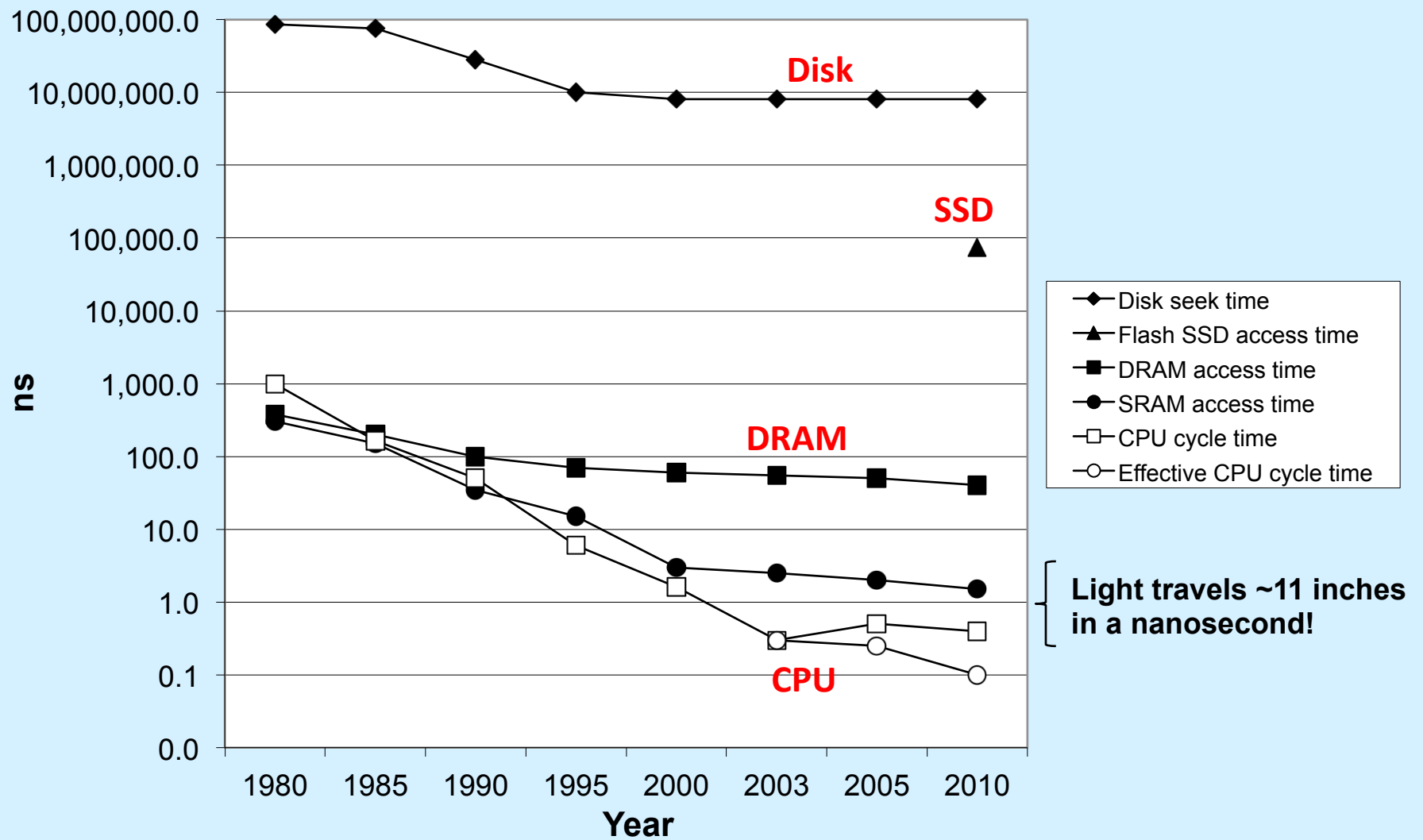
# Locality to the Rescue!

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

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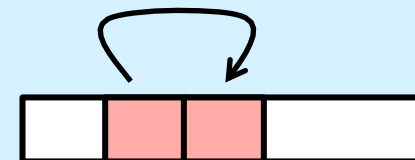
- **Spatial locality:**
  - items with nearby addresses tend to be referenced close together in time

# 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)**    <span style="color:red">**Spatial locality**</span>
  - **reference variable `sum` each iteration**    <span style="color:red">**Temporal locality**</span>
- **Instruction references**
  - **reference instructions in sequence.**    <span style="color:red">**Spatial locality**</span>
  - **cycle through loop repeatedly**    <span style="color:red">**Temporal locality**</span>

# 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;

}
```

# Quiz 3

Does this function have good locality with respect to array `a`?
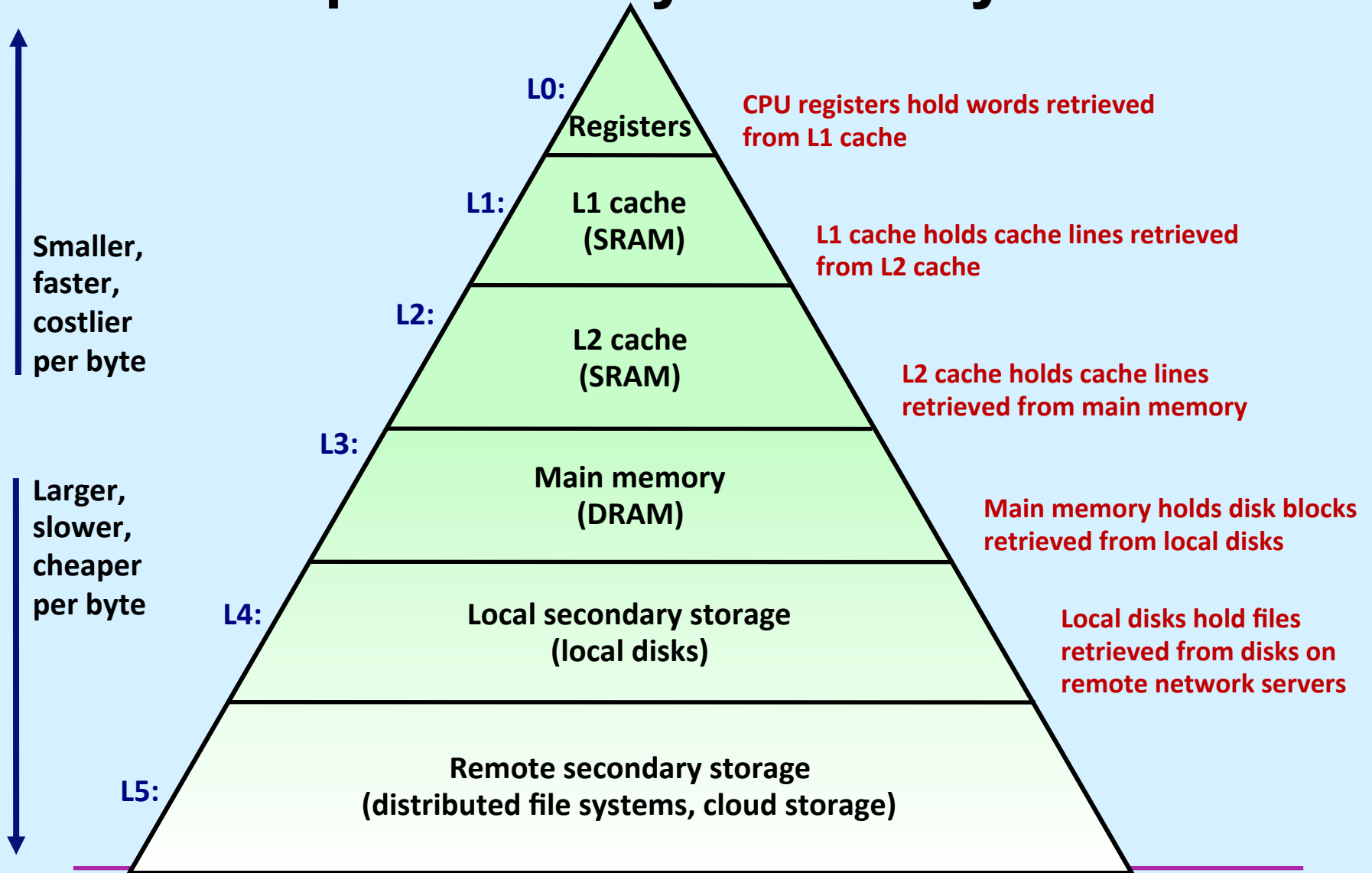
    a) yes

    b) no

```
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 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** <span style="color:red">**memory hierarchy**</span>

# An Example Memory Hierarchy

**Smaller, faster, costlier per byte**

**Larger, slower, cheaper per byte**

**L0:** Registers

CPU registers hold words retrieved from L1 cache

**L1:** L1 cache (SRAM)

L1 cache holds cache lines retrieved from L2 cache

**L2:** L2 cache (SRAM)

L2 cache holds cache lines retrieved from main memory

**L3:** Main memory (DRAM)

Main memory holds disk blocks retrieved from local disks

**L4:** Local secondary storage (local disks)

Local disks hold files retrieved from disks on remote network servers

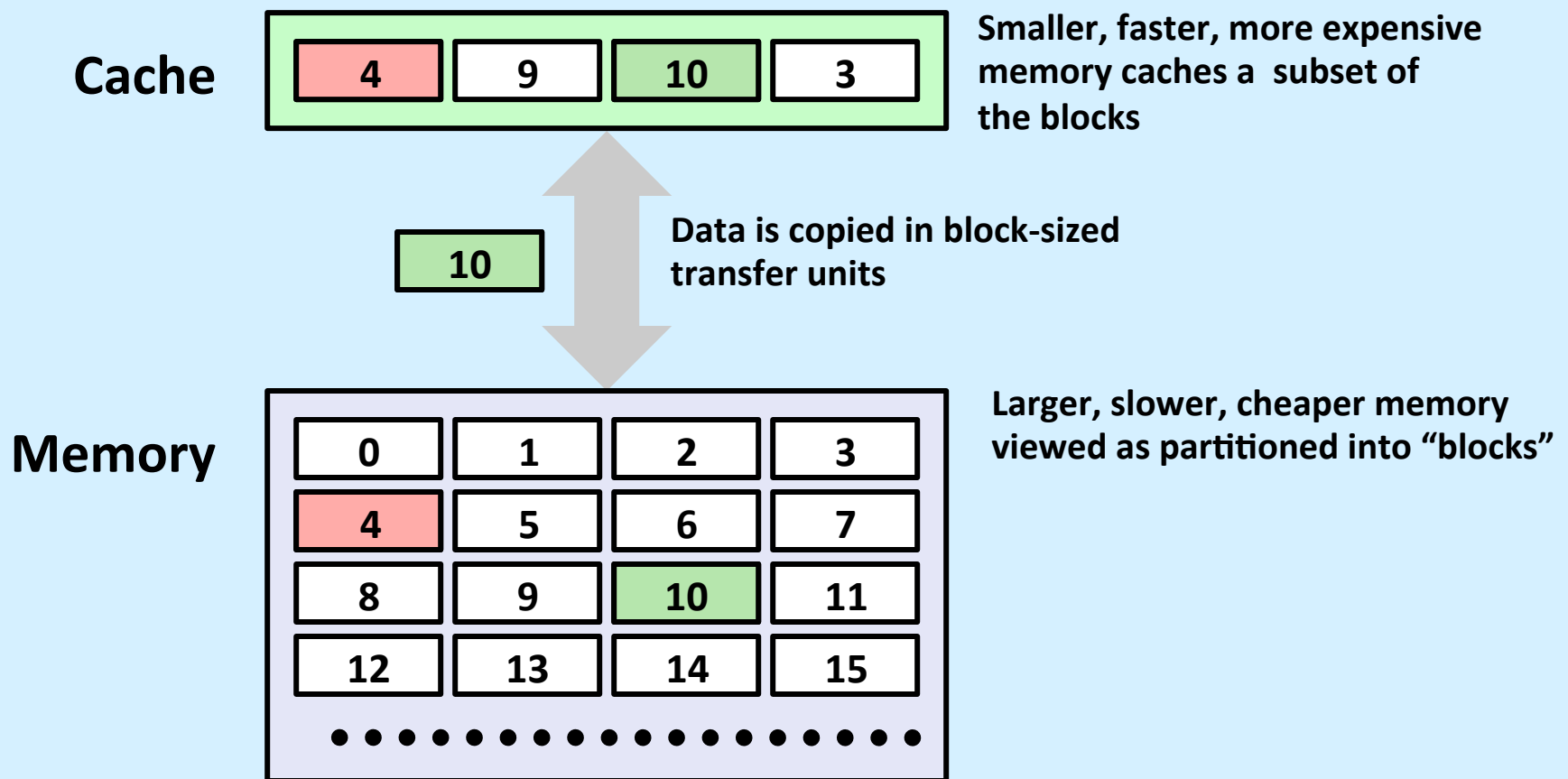**L5:** Remote secondary storage (distributed file systems, cloud storage)

# Putting Things Into Perspective ...

- **Reading from:**
  - **... the L1 cache is like grabbing a piece of paper from your desk (3 seconds)**
  - **... the L2 cache is picking up a book from a nearby shelf (14 seconds)**
  - **... main system memory is taking a 4-minute walk down the hall to talk to a friend**
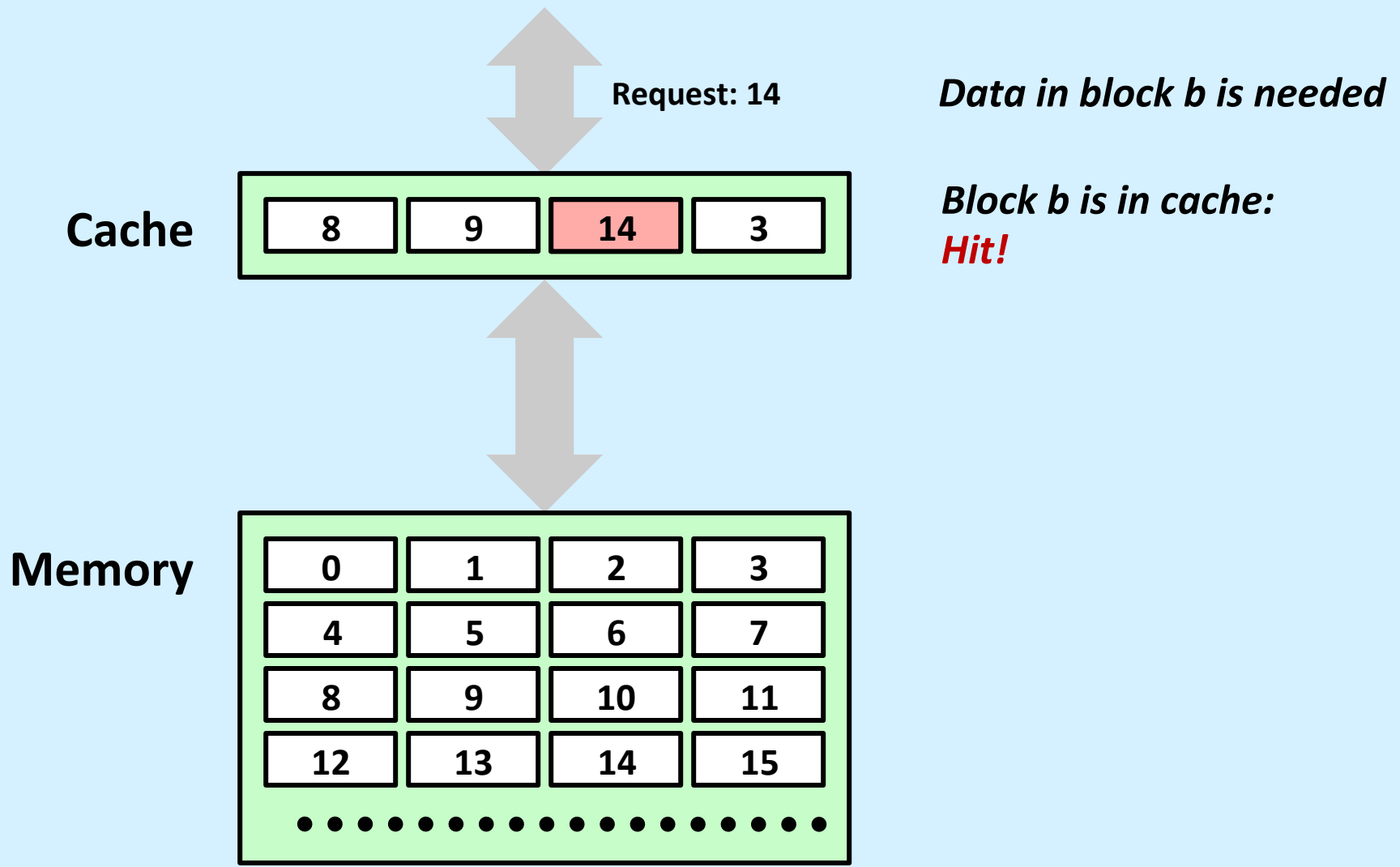  - **... a hard drive is like leaving the building to roam the earth for one year and three months**

# 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**

*Data in block b is needed*

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

**Cache**

| 8 | 9 | 14 | 3 |

**Memory**

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

# General Cache Concepts: Miss

**Request: 12**

**Cache**

| 8 | 12 | 14 | 3 |

*Data in block b is needed*

*Block b is not in cache: Miss!*

**12**

**Request: 12**

*Block b is fetched from memory*

**Memory**

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

*Block b is stored in cache*
- placement policy: determines where b goes
- replacement policy: determines which block gets evicted (victim)

# General Caching Concepts: 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 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 |
| L1 cache | 64-bytes block | On-Chip L1 | 1 | Hardware |
| L2 cache | 64-bytes block | On/Off-Chip L2 | 10 | Hardware |
| Virtual Memory | 4-KB page | 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 | CIFS/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**