CMSC 313 COMPUTER ORGANIZATION & ASSEMBLY LANGUAGE PROGRAMMING
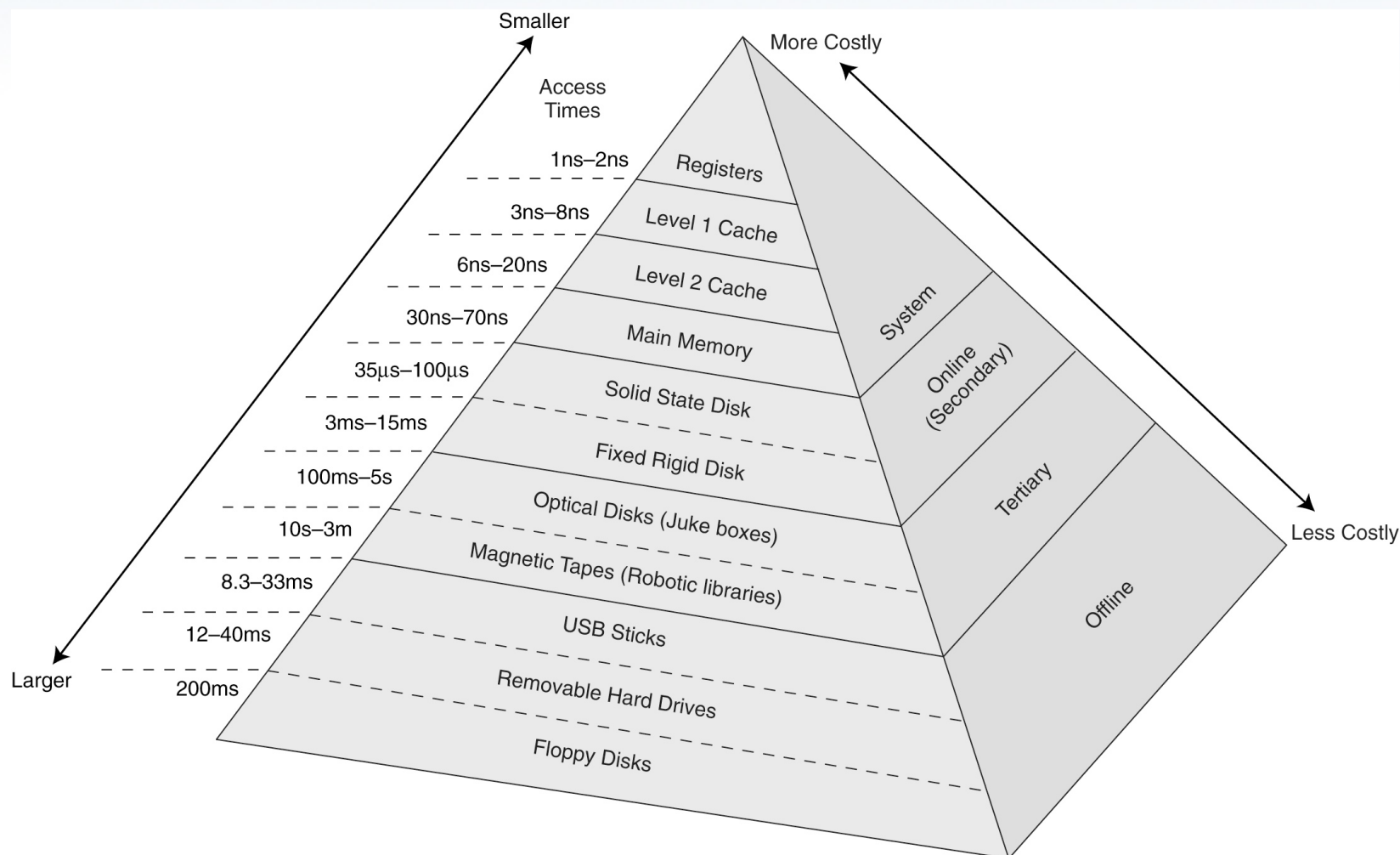
Lecture 26, Fall 2014

TOPICS TODAY

- **Memory Hierarchy**

- **Storage Technologies (RAM & Disk)**

- **Caching**

MEMORY HIERARCHY

6.3 The Memory Hierarchy

- This storage organization can be thought of as a pyramid:

Smaller

More Costly

Access Times

Access Times	Level
1ns–2ns	Registers
3ns–8ns	Level 1 Cache
6ns–20ns	Level 2 Cache
30ns–70ns	Main Memory
35μs–100μs	Solid State Disk
3ms–15ms	Fixed Rigid Disk
100ms–5s	Optical Disks (Juke boxes)
10s–3m	Magnetic Tapes (Robotic libraries)
8.3–33ms	USB Sticks
12–40ms	Removable Hard Drives
200ms	Floppy Disks

System

Online (Secondary)

Tertiary

Offline

Less Costly

Larger

STORAGE TECHNOLOGIES
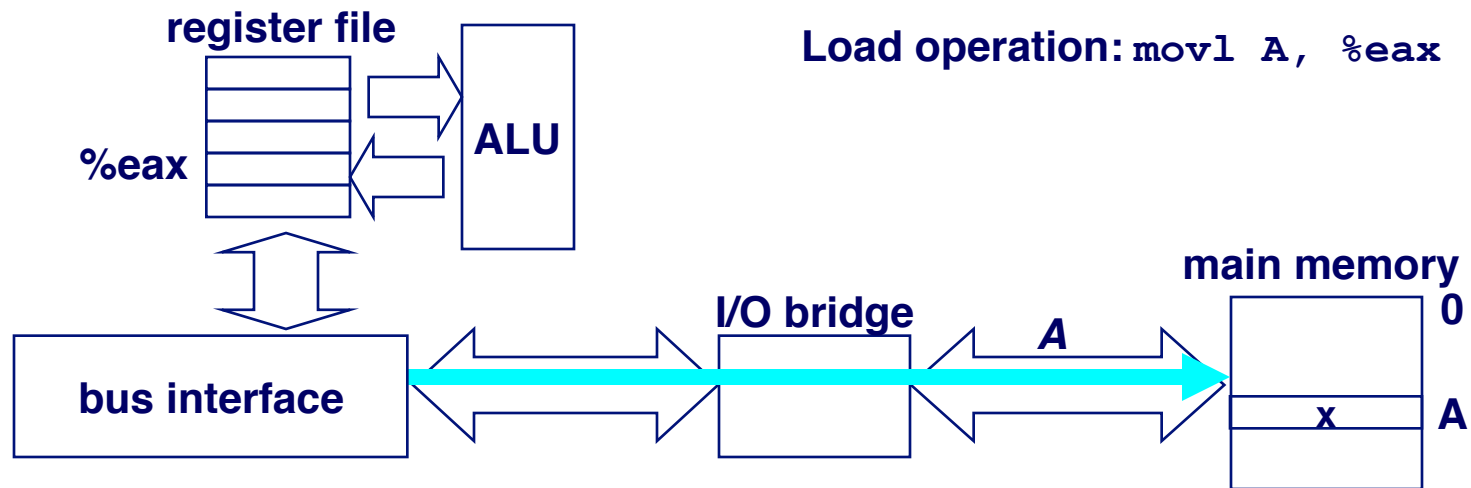
Typical Bus Structure Connecting CPU and Memory

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

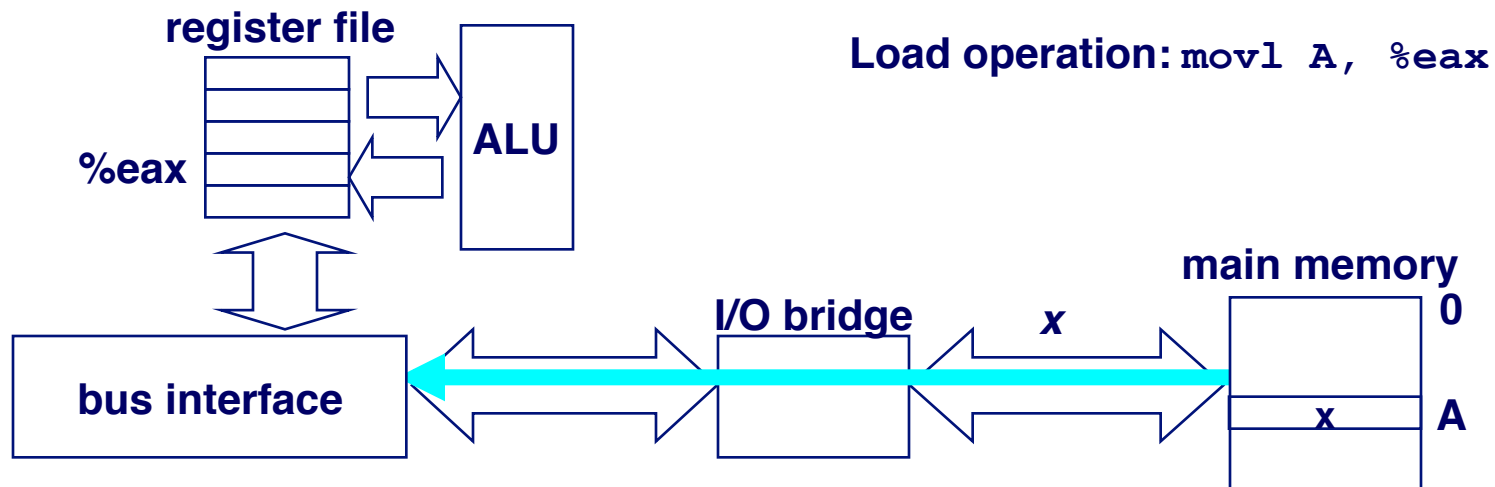**Buses are typically shared by multiple devices.**
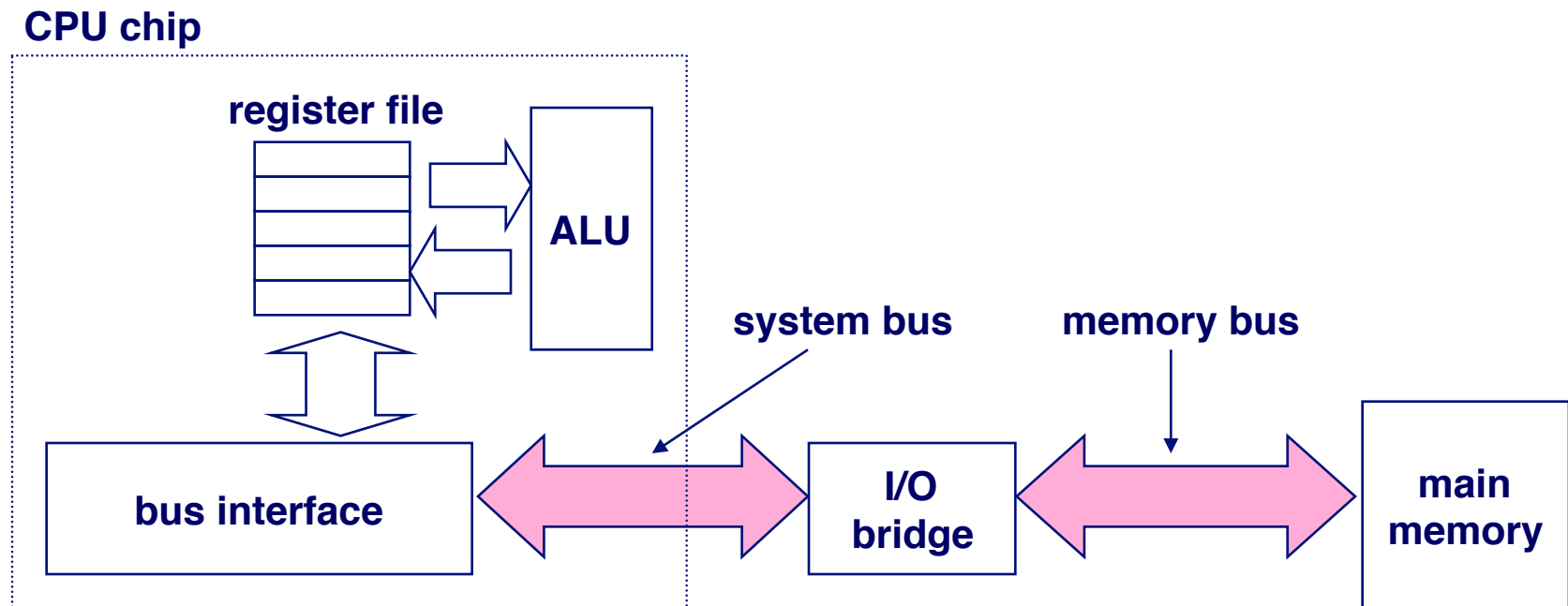
Memory Read Transaction (1)

CPU places address A on the memory bus.

Load operation: `movl A, %eax`
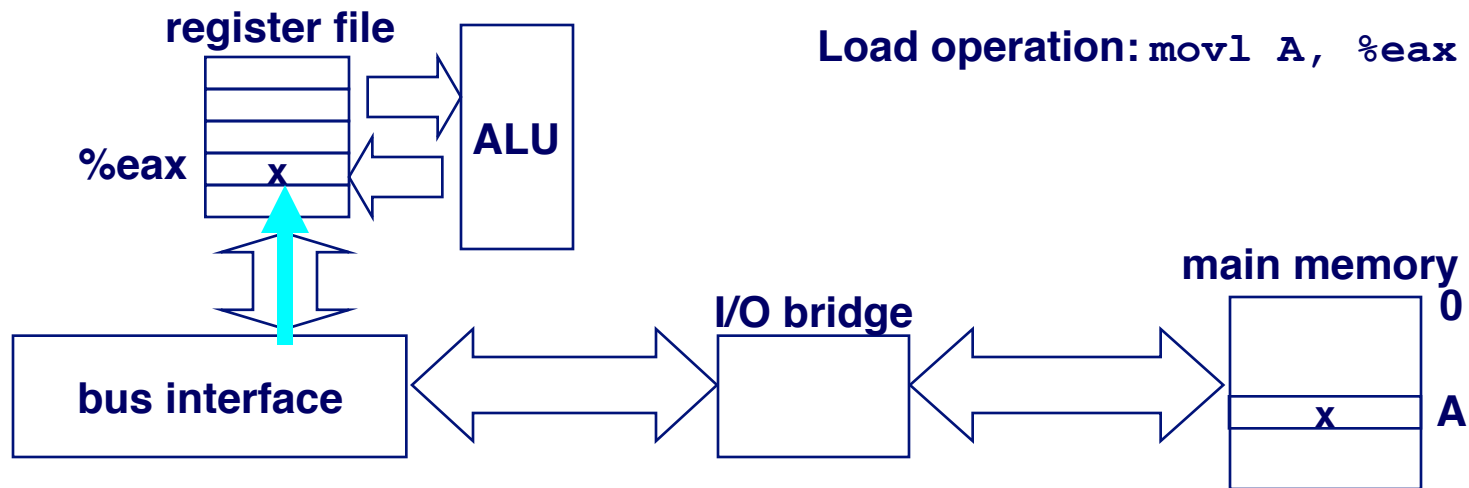
Memory Read Transaction (2)

Main memory reads A from the memory bus, retrieves word x, and places it on the bus.

register file

%eax

ALU

Load operation: `movl A, %eax`

bus interface

I/O bridge

x

main memory

0

x

A

Memory Read Transaction (3)

CPU reads word x from the bus and copies it into register %eax.

Load operation: `movl A, %eax`

register file

ALU
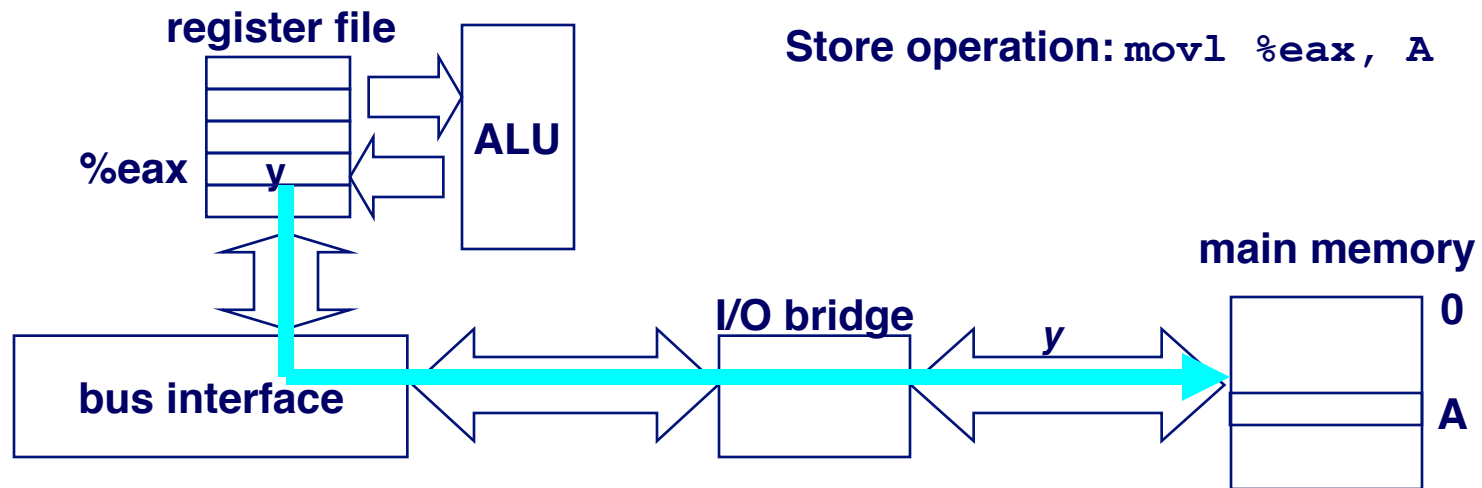
%eax   x

bus interface

I/O bridge

main memory

0

x   A

Memory Write Transaction (1)

CPU places address A on bus. Main memory reads it and waits for the corresponding data word to arrive.

register file

%eax | **y**

ALU

Store operation: `movl %eax, A`

bus interface

I/O bridge

A

main memory

0

A
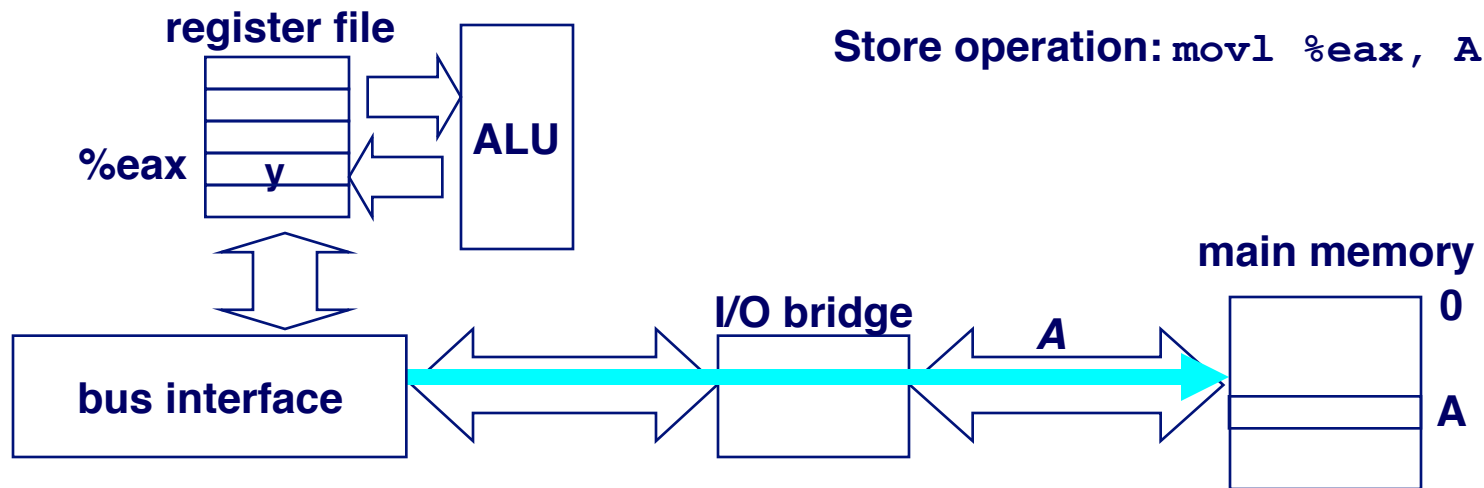
Memory Write Transaction (2)

CPU places data word y on the bus.

Store operation: `movl %eax, A`

Memory Write Transaction (3)

Main memory reads data word y from the bus and stores it at address A.

Store operation: `movl %eax, A`

register file

%eax | y

ALU

bus interface

I/O bridge

main memory

0

y | A
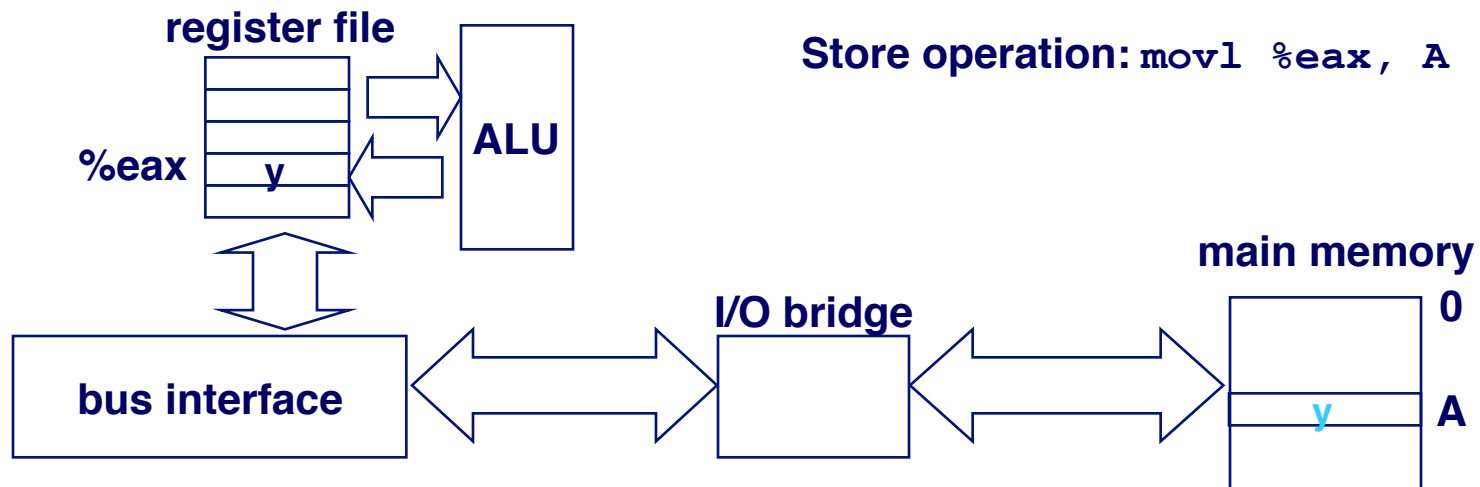
Random-Access Memory (RAM)

Key features

- **RAM is packaged as a chip.**
- **Basic storage unit is a cell (one bit per cell).**
- **Multiple RAM chips form a memory.**

Static RAM (SRAM)

- **Each cell stores bit with a six-transistor circuit.**
- **Retains value indefinitely, as long as it is kept powered.**
- **Relatively insensitive to disturbances such as electrical noise.**
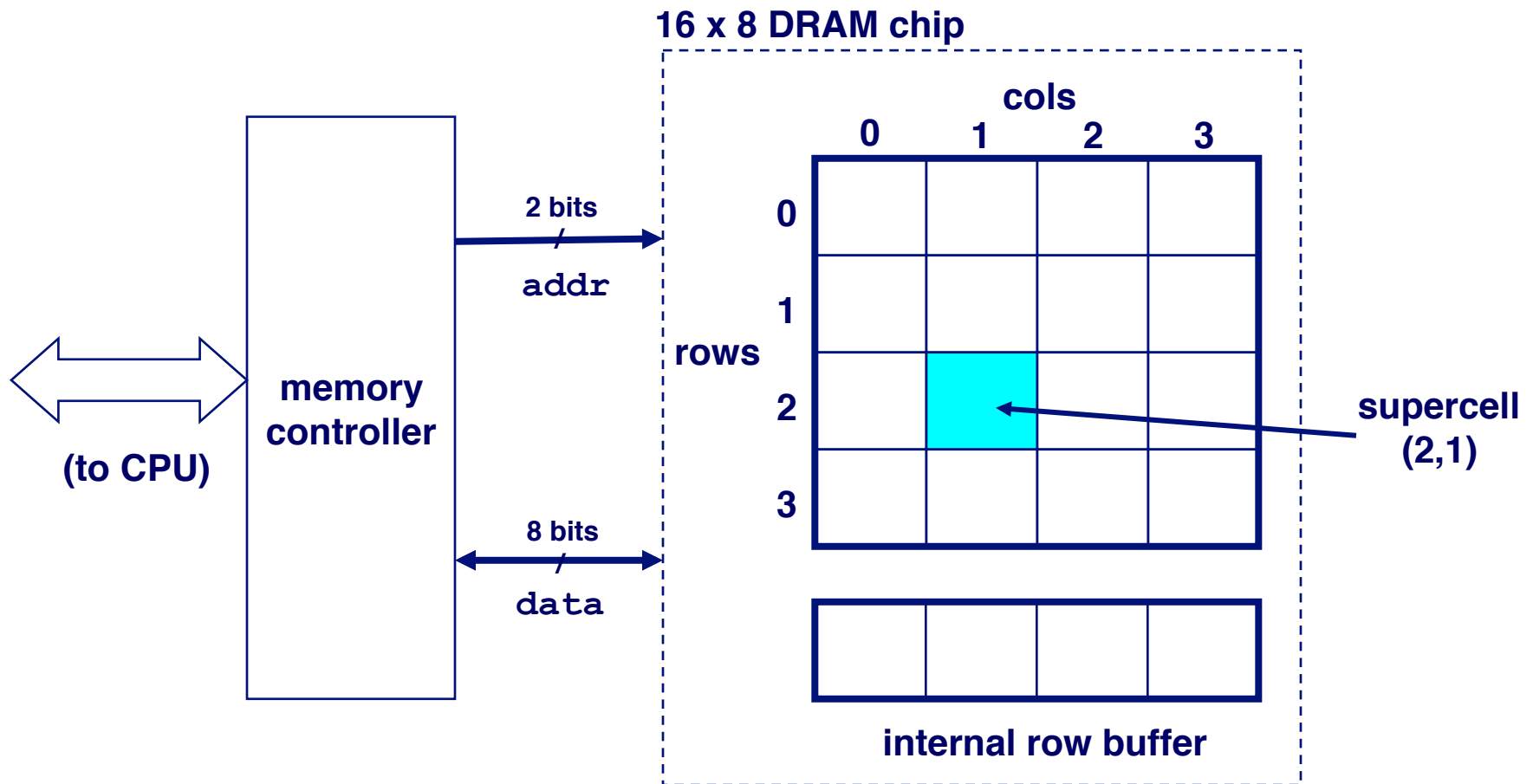- **Faster and more expensive than DRAM.**

Dynamic RAM (DRAM)

- **Each cell stores bit with a capacitor and transistor.**
- **Value must be refreshed every 10-100 ms.**
- **Sensitive to disturbances.**
- **Slower and cheaper than SRAM.**

Conventional DRAM Organization

d x w DRAM:

- dw total bits organized as d **supercells** of size w bits

16 x 8 DRAM chip

cols

supercell (2,1)

internal row buffer

(to CPU)

memory controller

2 bits / addr

8 bits / data

rows

Reading DRAM Supercell (2,1)

Step 1(a): Row access strobe (RAS) selects row 2.

Step 1(b): Row 2 copied from DRAM array to row buffer.

16 x 8 DRAM chip

memory controller

RAS = 2

2 / addr

8 / data

cols

0   1   2   3

rows

0

1

2

3

internal row buffer
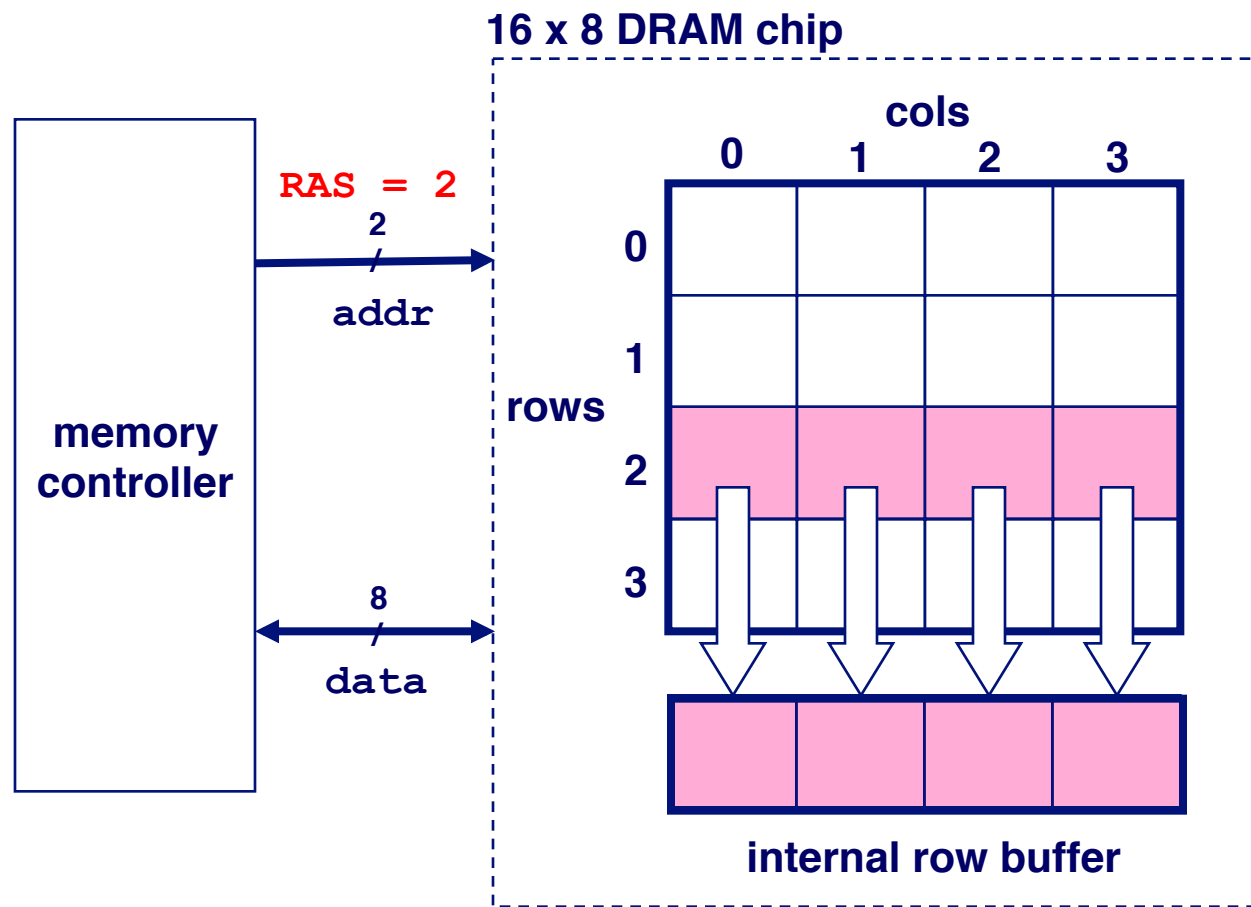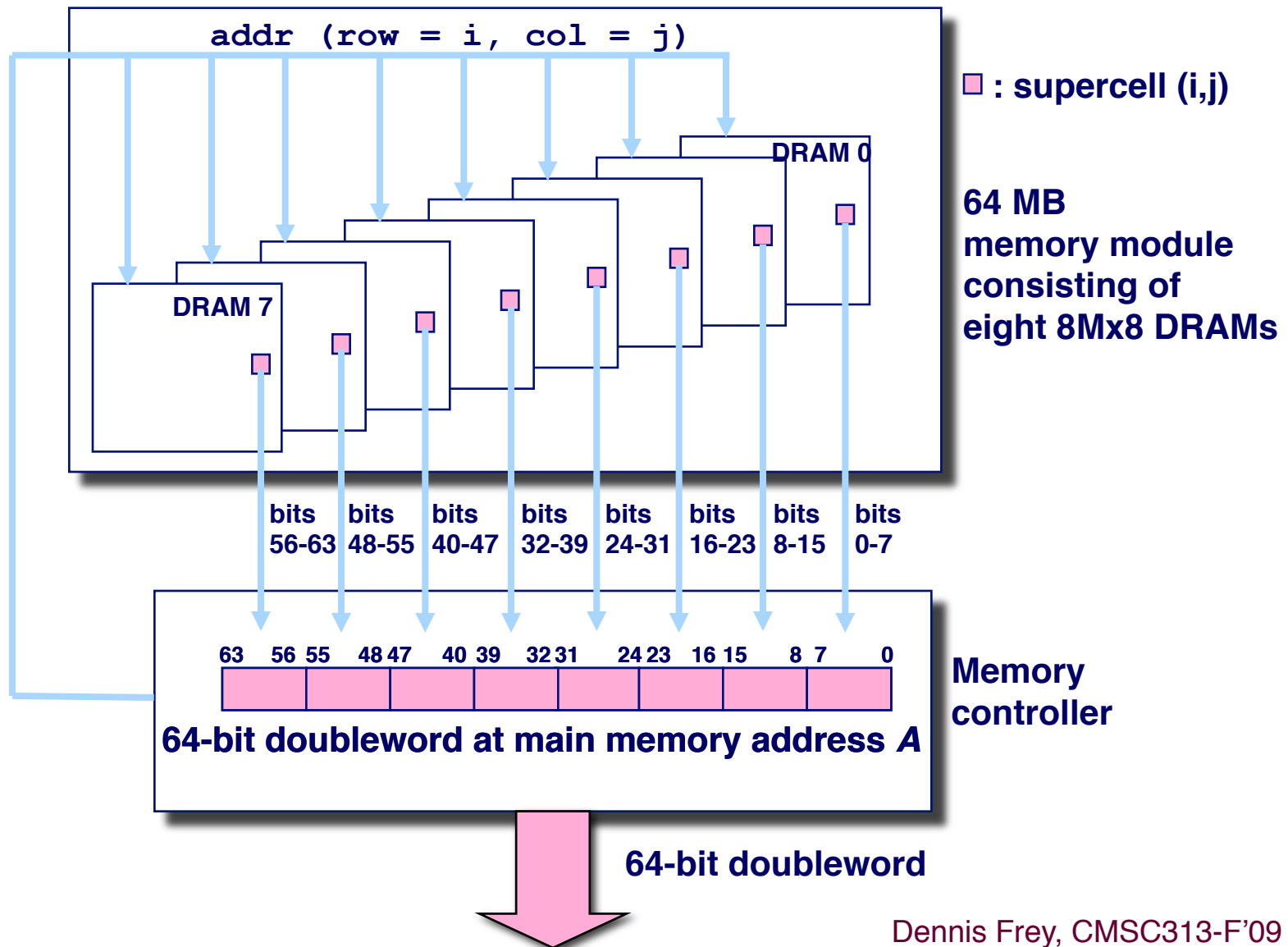
Reading DRAM Supercell (2,1)

Step 2(a): Column access strobe (CAS) selects column 1.

Step 2(b): Supercell (2,1) copied from buffer to data lines, and eventually back to the CPU.

16 x 8 DRAM chip

cols

0   1   2   3

CAS = 1

2

addr

To CPU

rows

0

1

2

3

memory controller

supercell (2,1)

8

data

supercell (2,1)

internal row buffer

Memory Modules

addr (row = i, col = j)

☐ : supercell (i,j)

DRAM 0

DRAM 7

64 MB
memory module
consisting of
eight 8Mx8 DRAMs

bits 56-63   bits 48-55   bits 40-47   bits 32-39   bits 24-31   bits 16-23   bits 8-15   bits 0-7

63 56 55 48 47 40 39 32 31 24 23 16 15 8 7 0

64-bit doubleword at main memory address *A*

Memory controller

64-bit doubleword

Dennis Frey, CMSC313-F'09

Nonvolatile Memories

DRAM and SRAM are volatile memories

- **Lose information if powered off.**

Nonvolatile memories retain value even if powered off.

- **Generic name is read-only memory (ROM).**
- **Misleading because some ROMs can be read and modified.**

Types of ROMs

- **Programmable ROM (PROM)**
- **Eraseable programmable ROM (EPROM)**
- **Electrically eraseable PROM (EEPROM)**
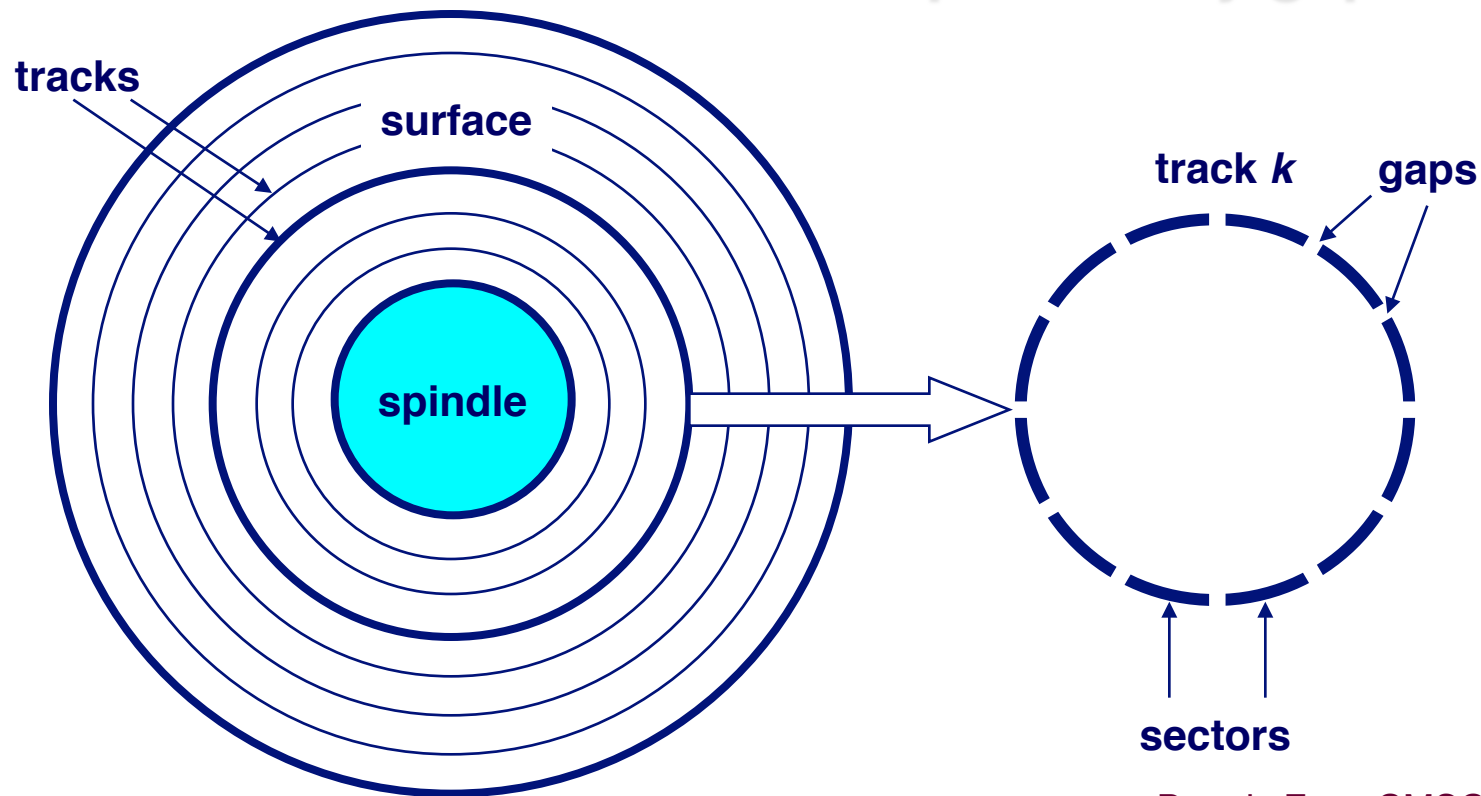- **Flash memory**

Firmware

- **Program stored in a ROM**
  - **Boot time code, BIOS (basic input/ouput system)**
  - **graphics cards, disk controllers.**

Disk Geometry

Disks consist of platters, each with two surfaces.

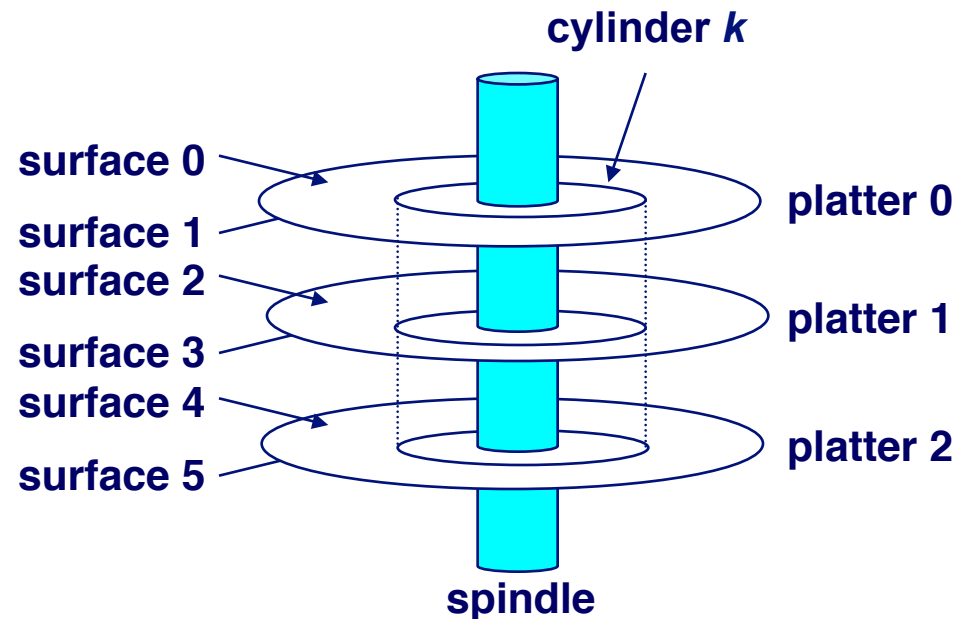**Each surface consists of concentric rings called tracks.**

Each track consists of sectors separated by gaps.

Disk Geometry (Muliple-Platter View)

Aligned tracks form a cylinder.

cylinder *k*

surface 0
surface 1
surface 2
surface 3
surface 4
surface 5

platter 0
platter 1
platter 2

spindle

Disk Capacity

Capacity: **maximum number of bits that can be stored.**

- Vendors express capacity in units of gigabytes (GB), where 1 GB = 10^9 bytes.

Capacity is determined by these technology factors:

- **Recording density** (bits/in): number of bits that can be squeezed into a 1 inch segment of a track.

- **Track density** (tracks/in): number of tracks that can be squeezed into a 1 inch radial segment.

- **Areal density** (bits/in^2): product of recording and track density.

Modern disks partition tracks into disjoint subsets called recording zones

- Each track in a zone has the same number of sectors, determined by the circumference of innermost track.

- Each zone has a different number of sectors/track

Computing Disk Capacity

Capacity = (# bytes/sector) x (avg. # sectors/track) x

(# tracks/surface) x (# surfaces/platter) x

(# platters/disk)

Example:

- 512 bytes/sector
- 300 sectors/track (on average)
- 20,000 tracks/surface
- 2 surfaces/platter
- 5 platters/disk

Capacity = 512 x 300 x 20000 x 2 x 5

= 30,720,000,000

= 30.72 GB

Disk Operation (Single-Platter View)

The disk surface spins at a fixed rotational rate
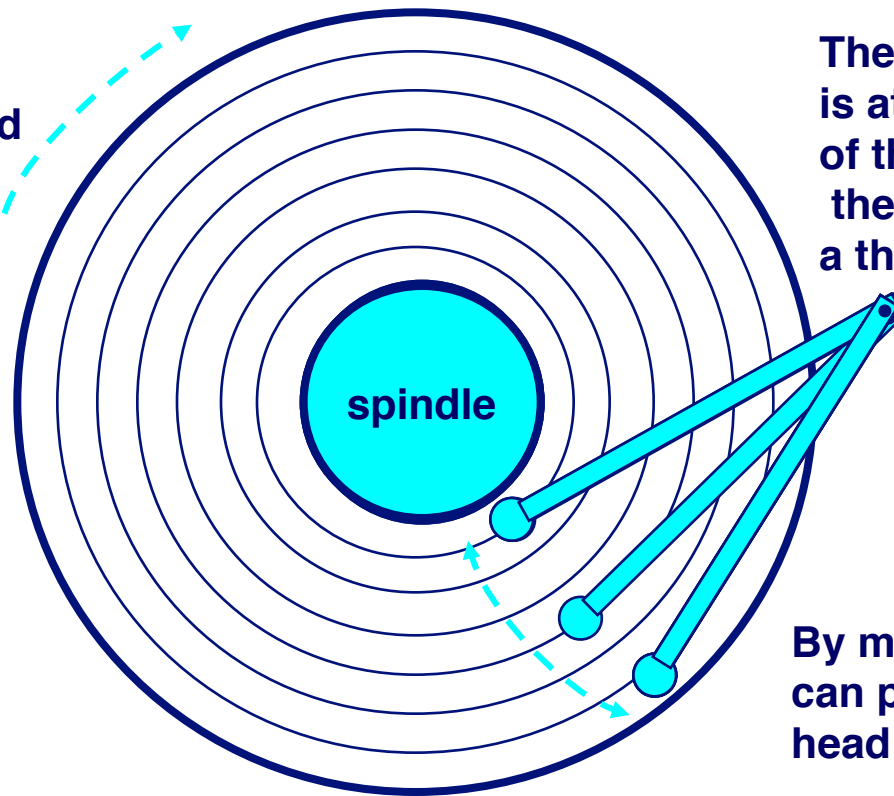
The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air.

spindle

By moving radially, the arm can position the read/write head over any track.

Disk Operation (Multi-Platter View)

read/write heads
move in unison
from cylinder to cylinder

arm

spindle

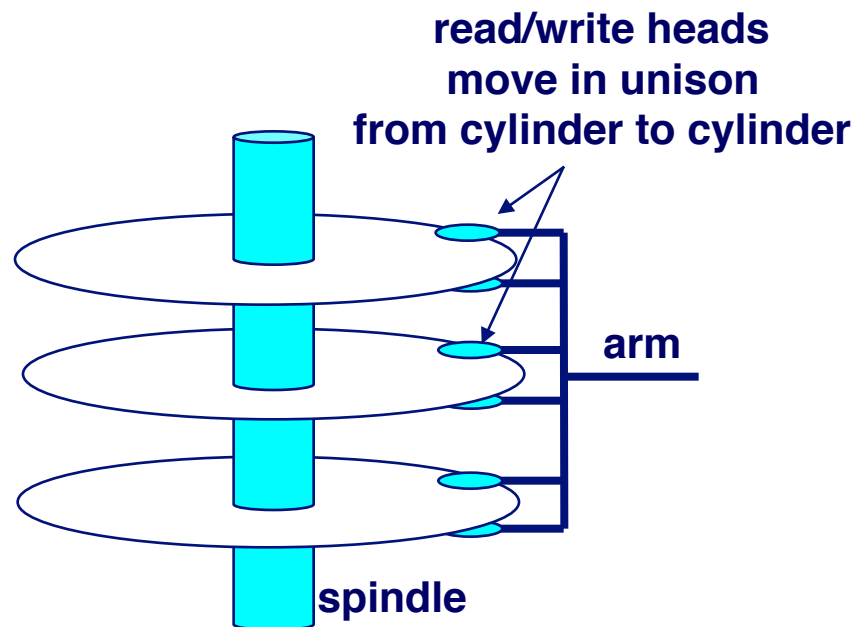# Disk Access Time

Average time to access some target sector approximated by :

- $T_{access} = T_{avg\ seek} + T_{avg\ rotation} + T_{avg\ transfer}$

Seek time ($T_{avg\ seek}$)

- Time to position heads over cylinder containing target sector.
- Typical $T_{avg\ seek}$ = 9 ms

Rotational latency ($T_{avg\ rotation}$)

- Time waiting for first bit of target sector to pass under read/write head.
- $T_{avg\ rotation}$ = 1/2 x 1/RPMs x 60 sec/1 min

Transfer time ($T_{avg\ transfer}$)

- Time to read the bits in the target sector.
- $T_{avg\ transfer}$ = 1/RPM x 1/(avg # sectors/track) x 60 secs/1 min.

Disk Access Time Example

Given:

- **Rotational rate = 7,200 RPM**
- **Average seek time = 9 ms.**
- **Avg # sectors/track = 400.**

Derived:

- **Tavg rotation = 1/2 x (60 secs/7200 RPM) x 1000 ms/sec = 4 ms.**
- **Tavg transfer = 60/7200 RPM x 1/400 secs/track x 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.**

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

bus interface

system bus

memory bus

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

Dennis Frey, CMSC313-F'09
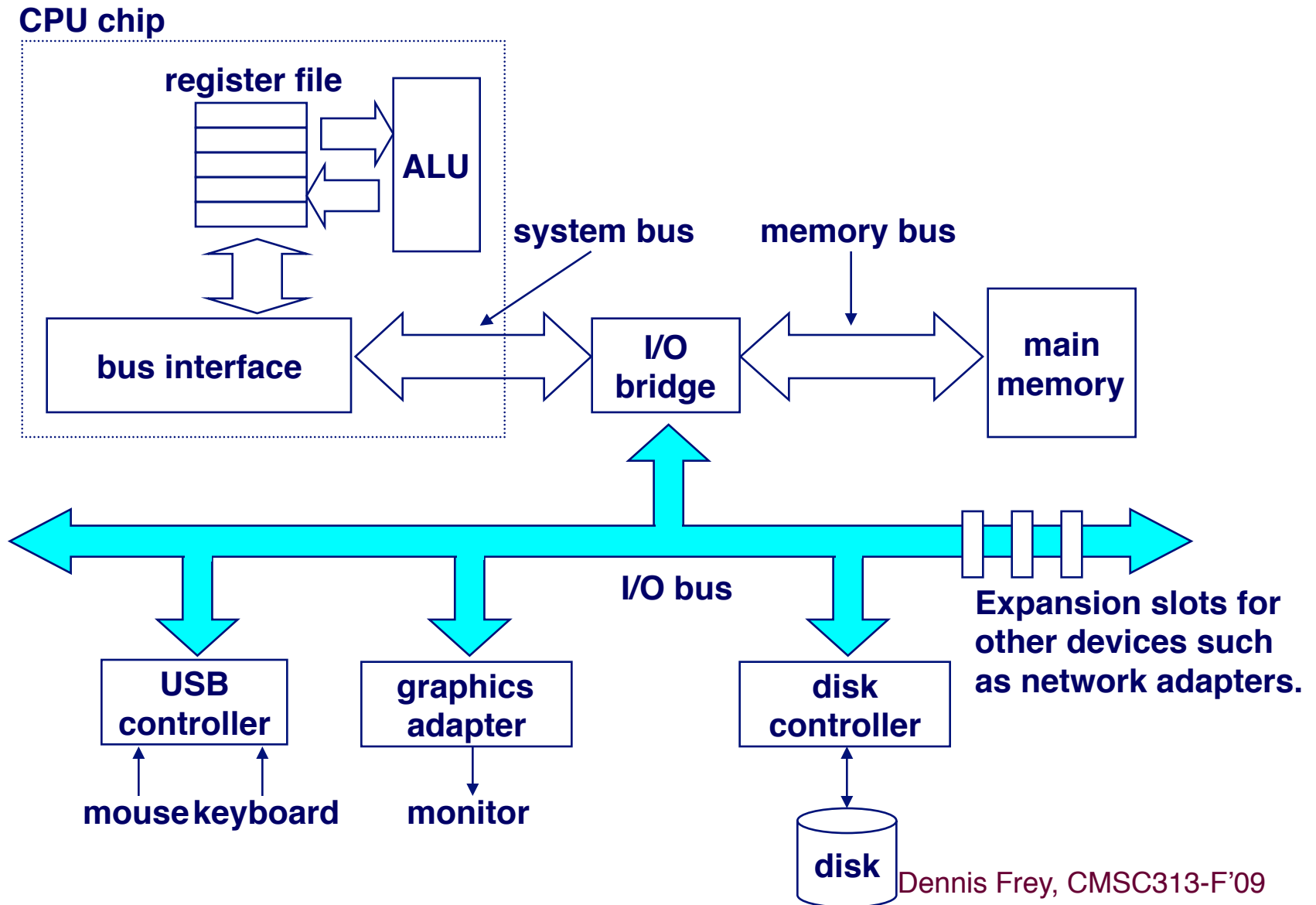
Reading a Disk Sector (1)

CPU chip

register file

ALU

bus interface

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

main memory

I/O bus

USB controller

graphics adapter

disk controller

mouse keyboard

monitor

disk

Dennis Frey, CMSC313-F'09
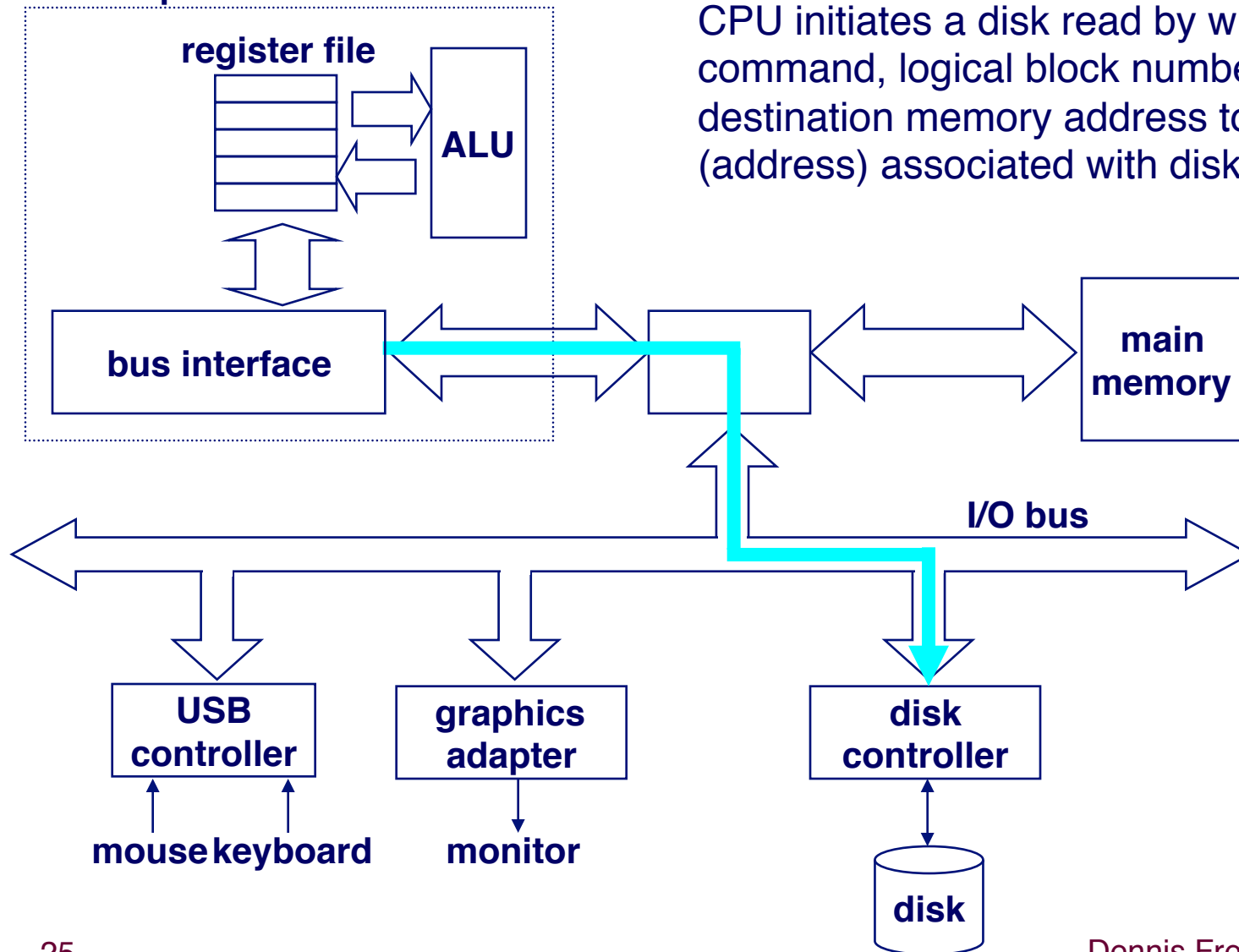
Reading a Disk Sector (2)

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

Dennis Frey, CMSC313-F'09
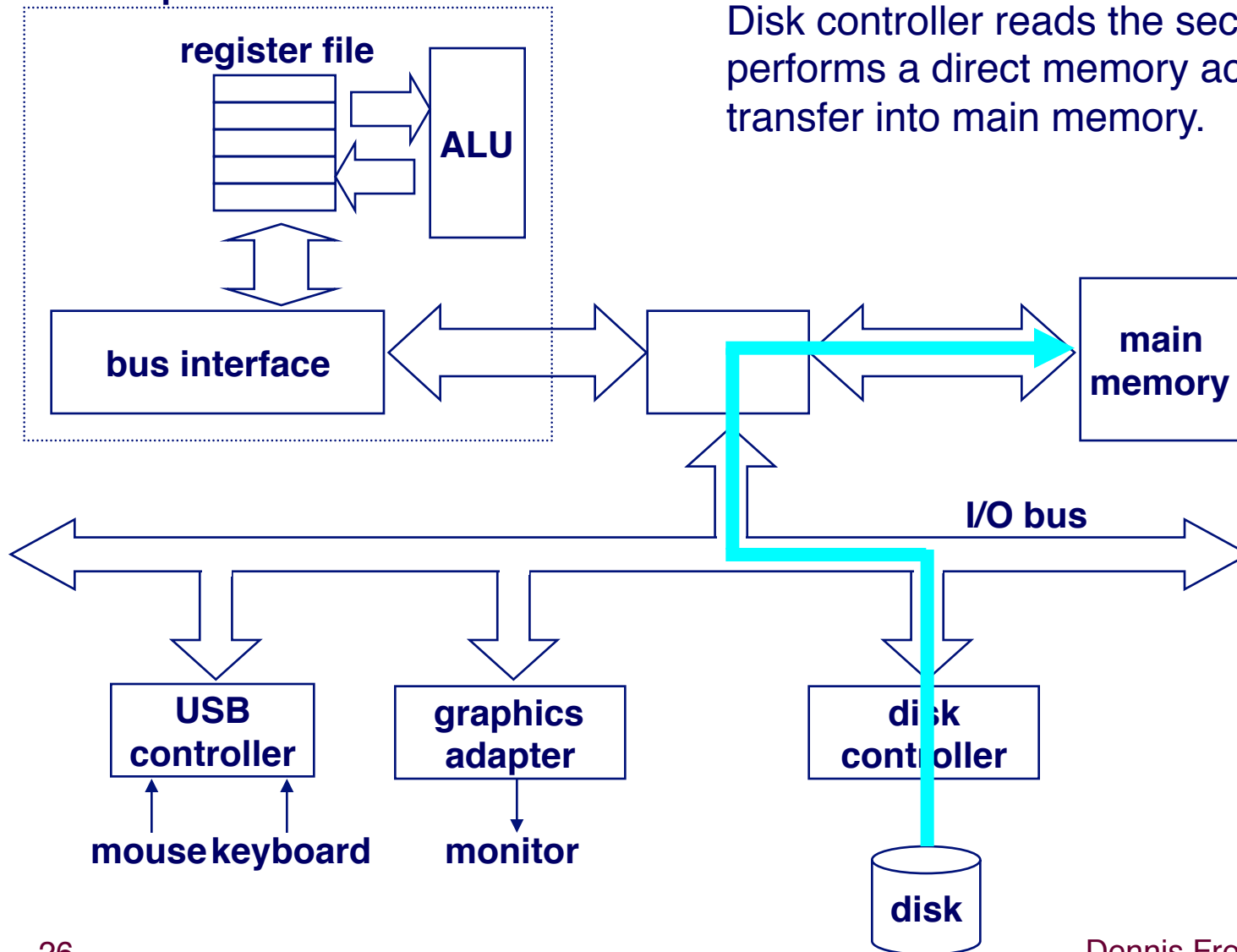
Reading a Disk Sector (3)

CPU chip

register file

ALU

bus interface

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

main memory

I/O bus

USB controller

graphics adapter

disk controller

mouse keyboard

monitor

disk
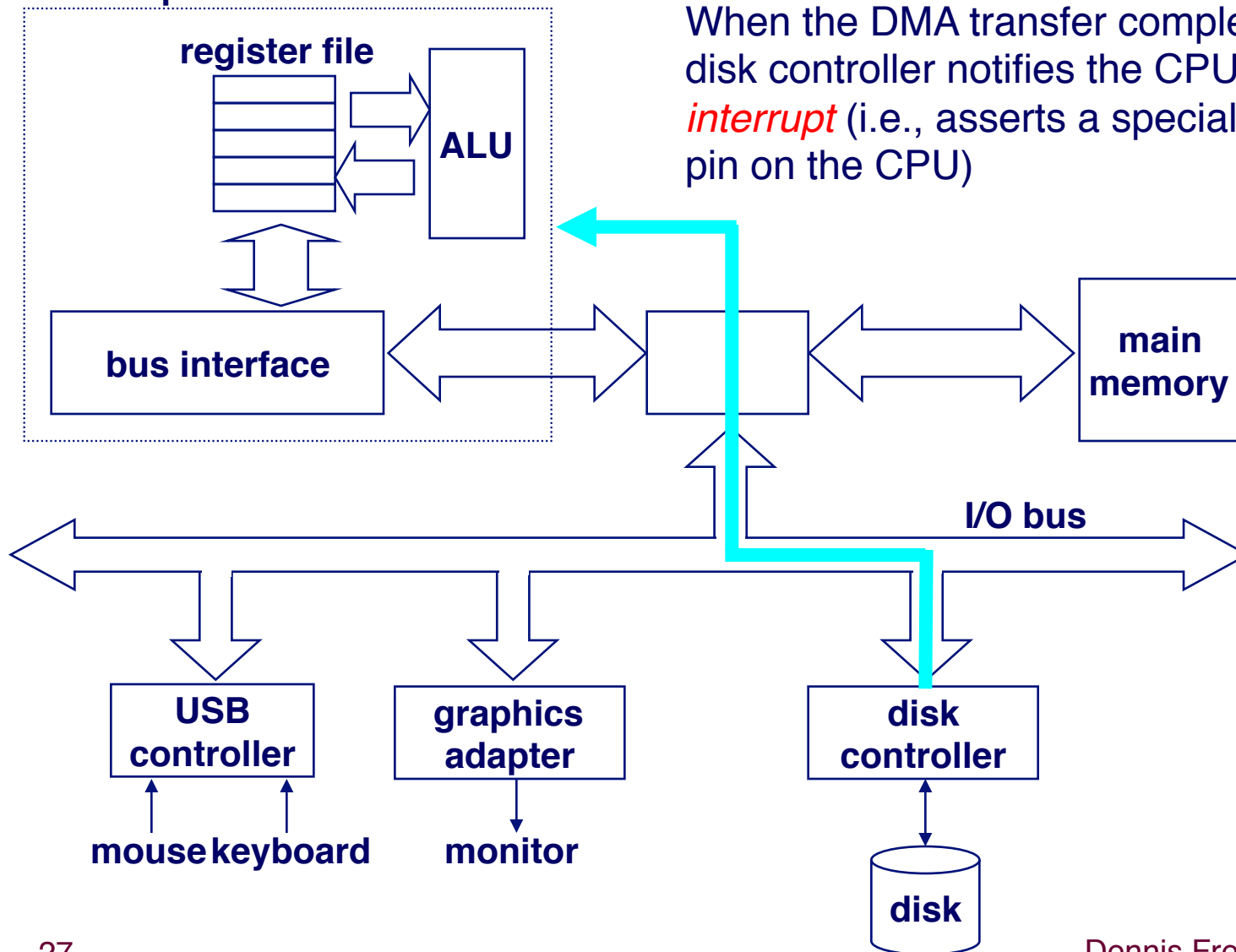
CACHING

Locality

Principle of Locality:

- **Programs tend to reuse data and instructions near those they have used recently, or that were recently referenced themselves.**

- **Temporal locality: Recently referenced items are likely to be referenced 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**
  - Reference array elements in succession (stride-1 reference pattern): **Spatial locality**
  - Reference `sum` each iteration: **Temporal locality**

- **Instructions**
  - Reference instructions in sequence: **Spatial locality**
  - Cycle through loop repeatedly: **Temporal locality**

Locality Example

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?

```c
int sumarrayrows(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

Question: Does this function have good locality?

```c
int sumarraycols(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

Question: 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)?

```c
int sumarray3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < M; 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 and have less capacity.**

- **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.

An Example Memory Hierarchy

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.

Dennis Frey, CMSC313-F'09

Caches

Cache: **A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.**

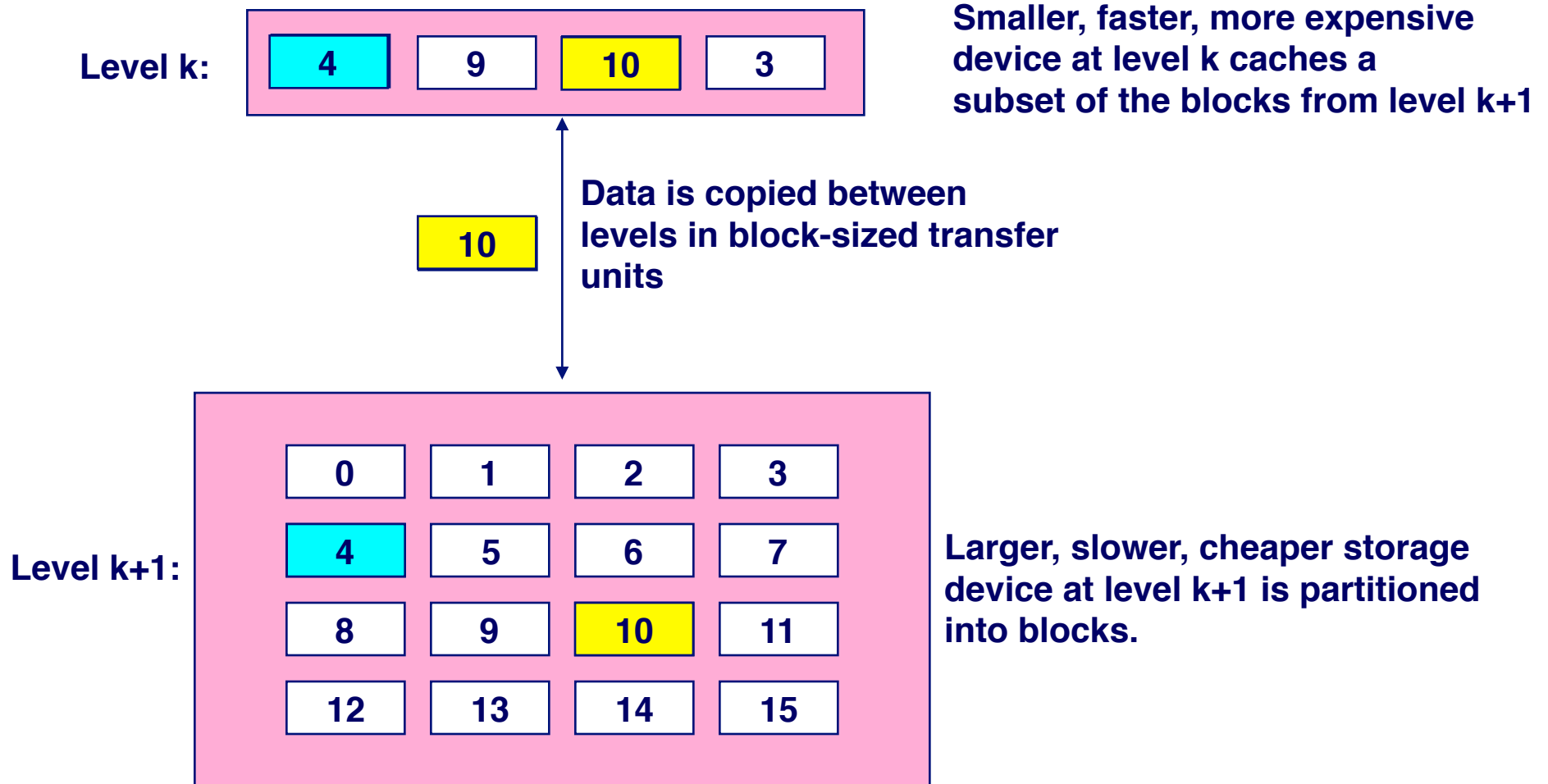**Fundamental idea of a memory hierarchy:**

- **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?

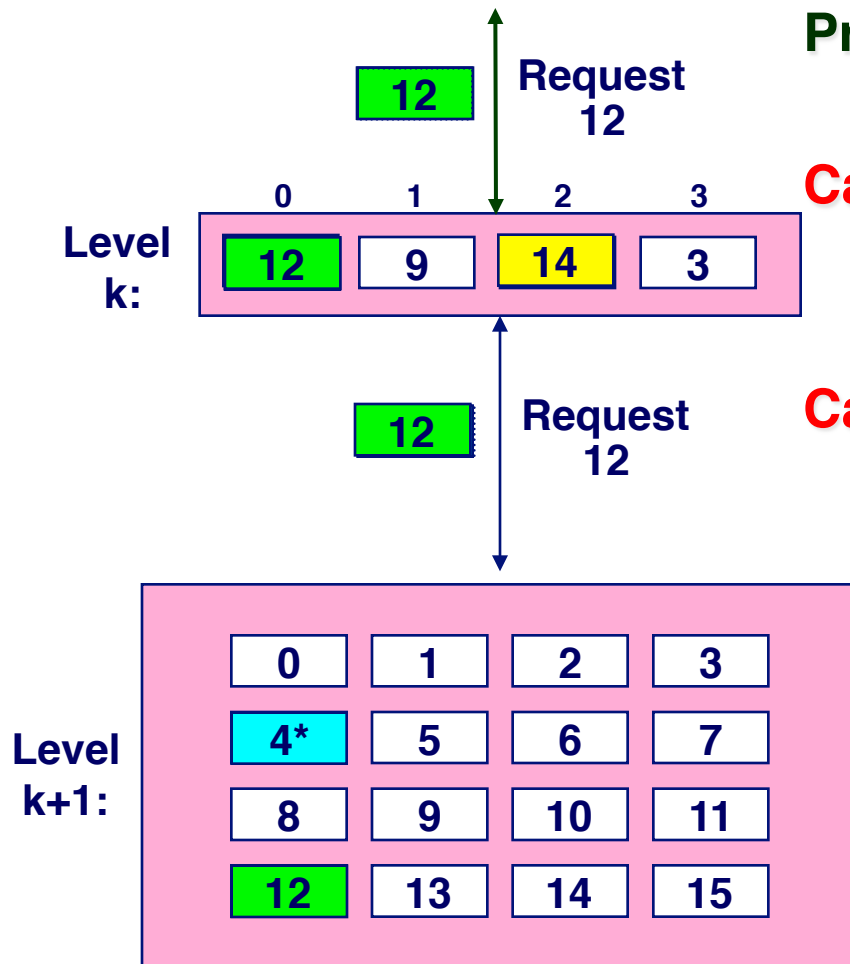- **Programs tend to access the data at level k more often than they access the data at level k+1.**
- **Thus, the storage at level k+1 can be slower, and thus larger and cheaper per bit.**
- **Net effect: A large pool of memory that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.**

Caching in a Memory Hierarchy

Level k:

| 4 | 9 | 10 | 3 |

Smaller, faster, more expensive device at level k caches a subset of the blocks from level k+1

10

Data is copied between levels in block-sized transfer units

Level k+1:

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

Larger, slower, cheaper storage device at level k+1 is partitioned into blocks.

General Caching Concepts

Request 12

0	1	2	3

Level k:

12	9	14	3

Request 12

Level k+1:

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

Program needs object d, which is stored in some block b.

Cache hit

- **Program finds b in the cache at level k. E.g., block 14.**

Cache miss

- **b is not at level k, so level k cache must fetch it from level k+1. E.g., block 12.**

- **If level k cache is full, then some current block must be replaced (evicted). Which one is the "victim"?**
  - **Placement policy: where can the new block go? E.g., b mod 4**
  - **Replacement policy: which block should be evicted? E.g., LRU**

General Caching Concepts

Types of cache misses:

- **Cold (compulsary) 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+1.
  - 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.

Dennis Frey, CMSC313-F'09

Examples of Caching in the Hierarchy

Cache Type	What Cached	Where Cached	Latency (cycles)	Managed By
Registers	4-byte word	CPU registers	0	Compiler
TLB	Address translations	On-Chip TLB	0	Hardware
L1 cache	32-byte block	On-Chip L1	1	Hardware
L2 cache	32-byte block	Off-Chip L2	10	Hardware
Virtual Memory	4-KB page	Main memory	100	Hardware +OS
Buffer cache	Parts of files	Main memory	100	OS
Network buffer cache	Parts of files	Local disk	10,000,000	AFS/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

NEXT TIME

- **Cache Mapping Schemes**

- **Virtual Memory**