This week and next

- **Tomorrow:– Quiz #4**

- **Next week:**
  - No class Monday — Patriot’s Day holiday
  - Tuesday — Regular class but *NO QUIZ*
    - Virtual Thursday!
  - No class Thursday — Project Presentation Day
  - Regular class Friday

The Memory Hierarchy

Professor Hugh C. Lauer

CS-2011, Machine Organization and Assembly Language

(Slides include copyright materials from *Computer Systems: A Programmer's Perspective*, by Bryant and O'Hallaron, and from *The C Programming Language*, by Kernighan and Ritchie)

Today

- Storage technologies and trends
- Locality of reference
- Caching in the memory hierarchy

Reading Assignment: §6.1 – §6.5

Random-Access Memory (RAM)

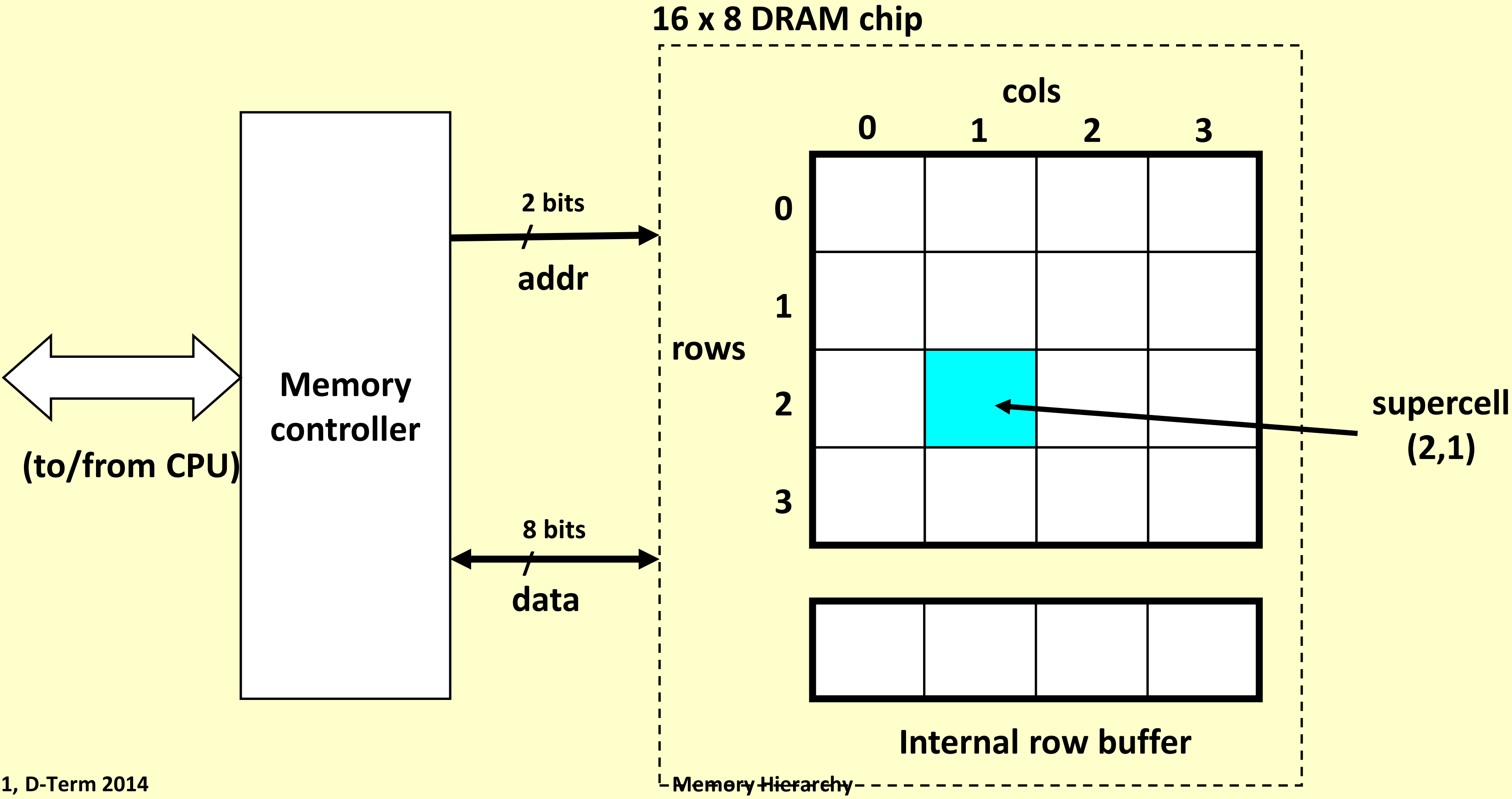
- **Key features**
  - RAM is traditionally packaged as a chip.
  - Basic storage unit is normally a cell (one bit per cell).
  - Multiple RAM chips form a memory.
- **Static RAM (SRAM)**
  - Each cell stores a bit with a four or six-transistor circuit.
  - Retains value indefinitely, as long as it is kept powered.
  - Relatively insensitive to electrical noise (EMI), radiation, etc.
  - Faster and more expensive than DRAM.
- **Dynamic RAM (DRAM)**
  - Each cell stores bit with a capacitor. One transistor is used for access
  - Value must be refreshed every 10-100 ms.
  - More sensitive to disturbances (EMI, radiation,…) than SRAM.
  - Slower and cheaper than SRAM.

SRAM vs DRAM Summary

	Trans. per bit	Access time	Needs refresh?	Needs EDC?	Cost	Applications
SRAM	4 or 6	1X	No	Maybe	100x	Cache memories
DRAM	1	10X	Yes	Yes	1X	Main memories, frame buffers

Conventional DRAM Organization

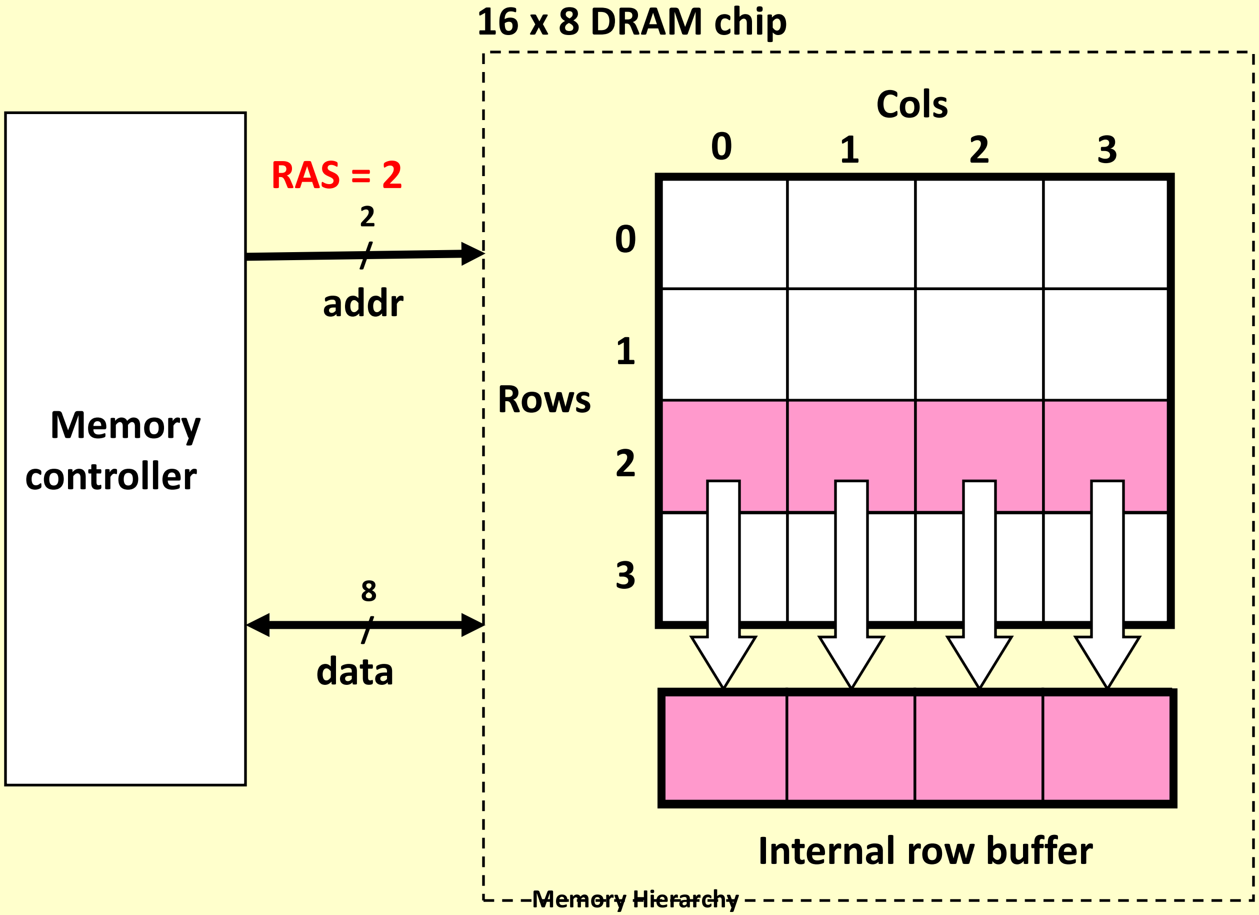
- **d x w DRAM:**
  - dw total bits organized as d supercells of size w bits



Reading DRAM Supercell (2,1)

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

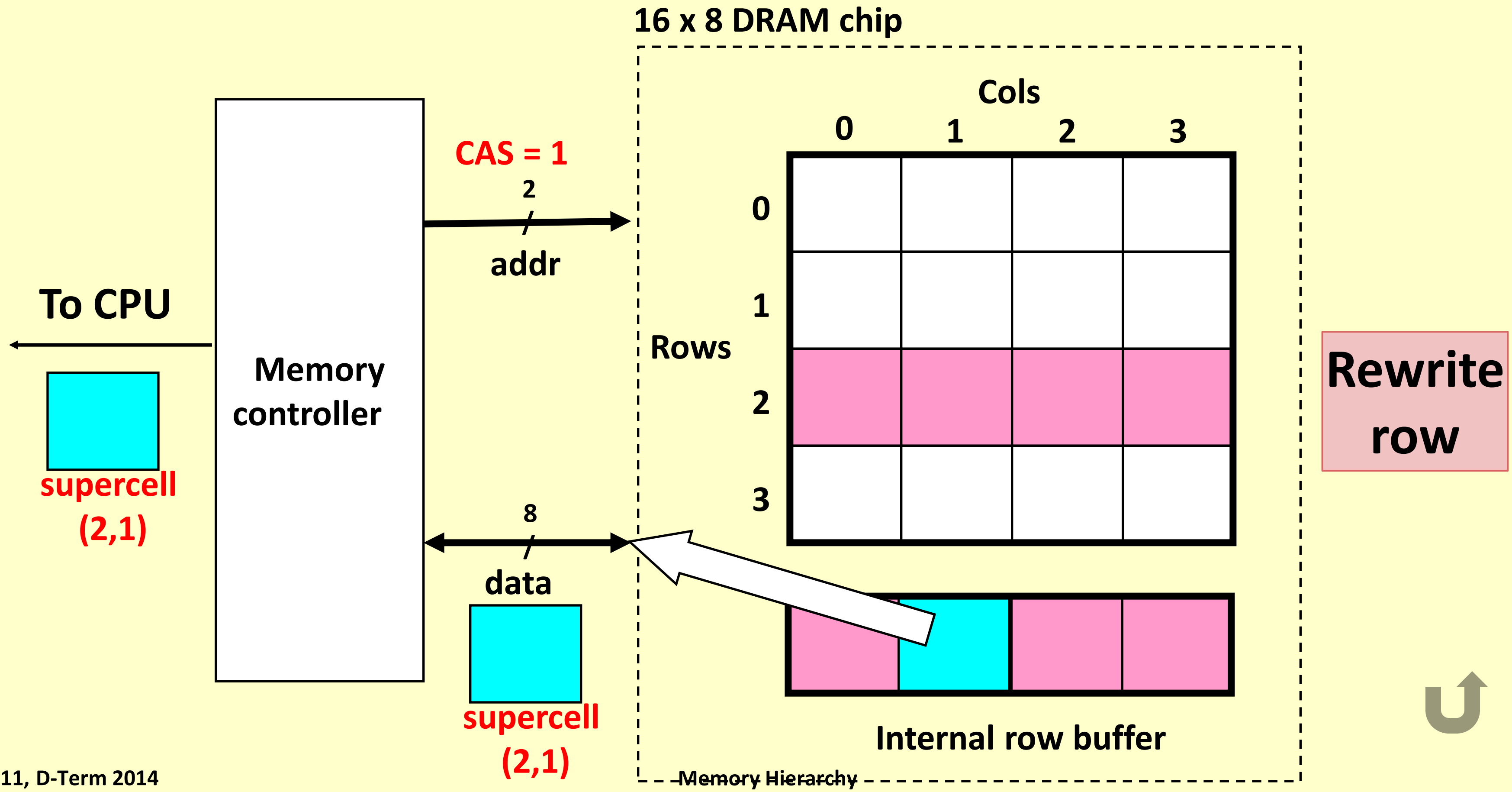
Step 1(b): Row 2 copied from DRAM array to row buffer.



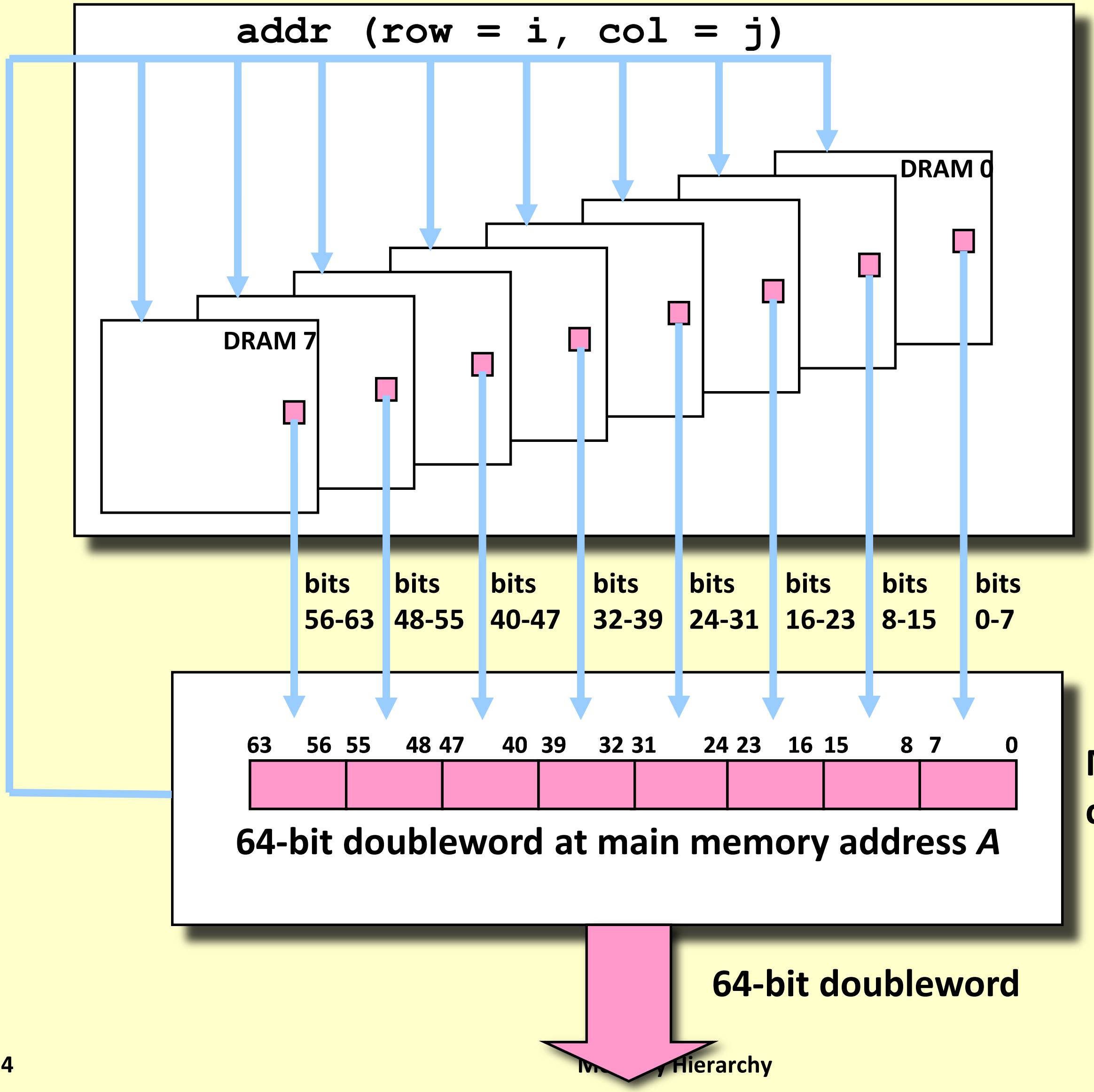
Reading DRAM Supercell (2,1)

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.



Memory Modules



🟪 : supercell (i,j)

64 MB
memory module
consisting of
eight 8Mx8 DRAMs

Memory
controller

64-bit doubleword

Enhanced DRAMs

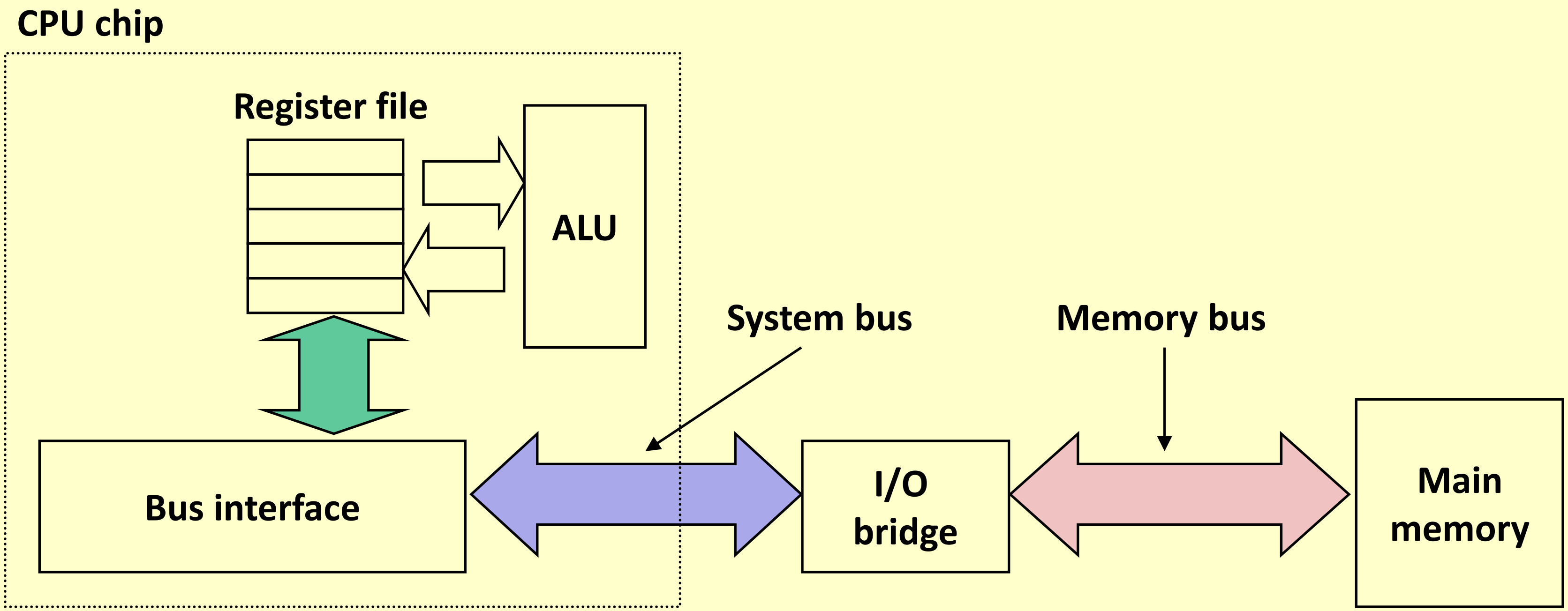
- **Basic DRAM cell has not changed since its invention in 1966.**
  - Commercialized by Intel in 1970.
- **DRAM cores with better interface logic and faster I/O :**
  - Synchronous DRAM (SDRAM)
    - Uses a conventional clock signal instead of asynchronous control
    - Allows reuse of the row addresses (e.g., RAS, CAS, CAS, CAS)
  - Double data-rate synchronous DRAM (DDR SDRAM)
    - Double edge clocking sends two bits per cycle per pin
    - Different types distinguished by size of small prefetch buffer:
      - DDR (2 bits), DDR2 (4 bits), DDR4 (8 bits)
    - By 2010, standard for most server and desktop systems
    - Intel Core i7 supports only DDR3 SDRAM

Nonvolatile Memories

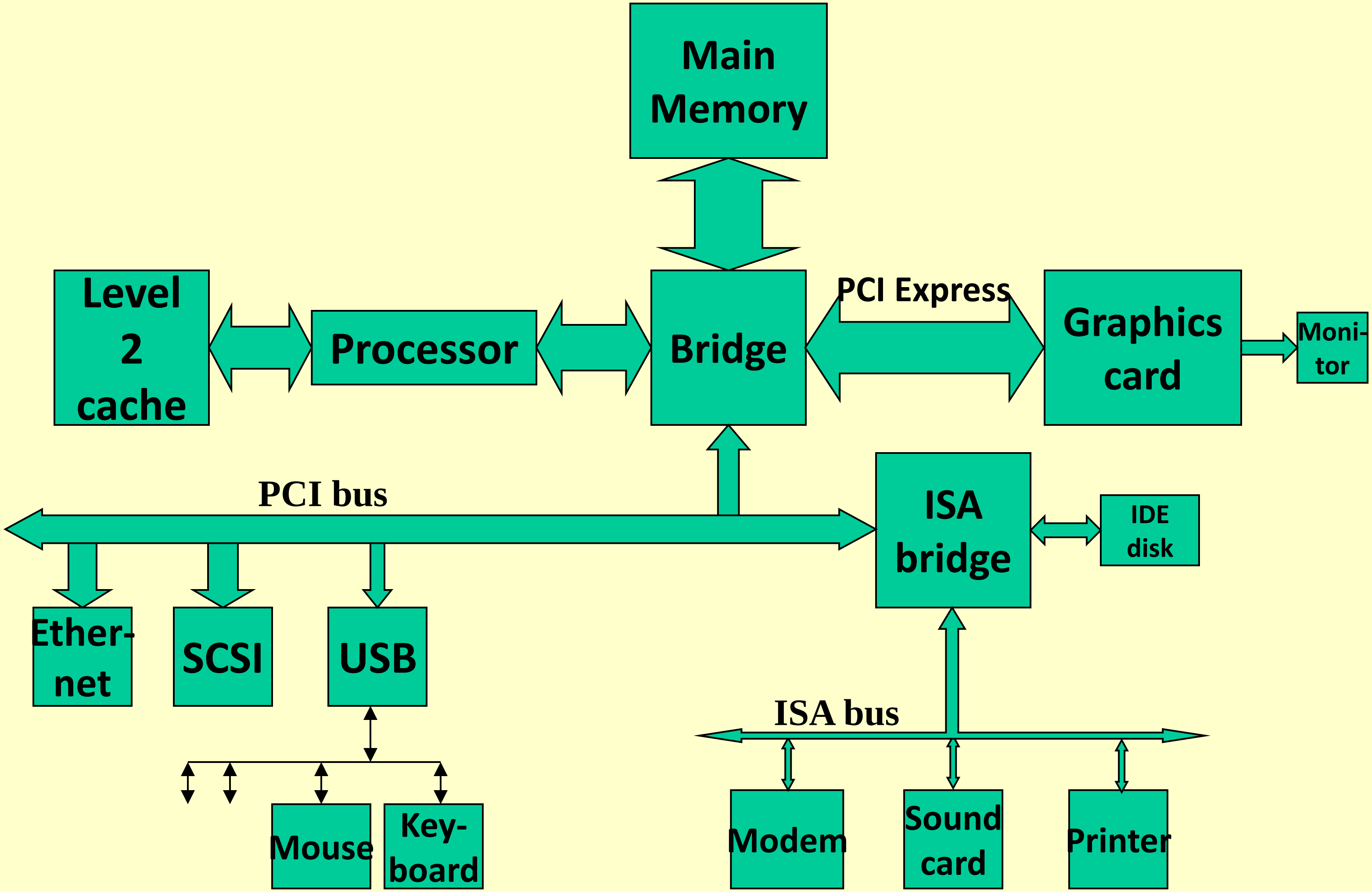
- **DRAM and SRAM are volatile memories**
  - Lose information if powered off.
- **Nonvolatile memories retain value even if powered off**
  - Read-only memory (ROM): programmed during production
  - Programmable ROM (PROM): can be programmed once
  - Eraseable PROM (EPROM): can be bulk erased (UV, X-Ray)
  - Electrically eraseable PROM (EEPROM): electronic erase capability
  - Flash memory: EEPROMs with partial (sector) 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

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.

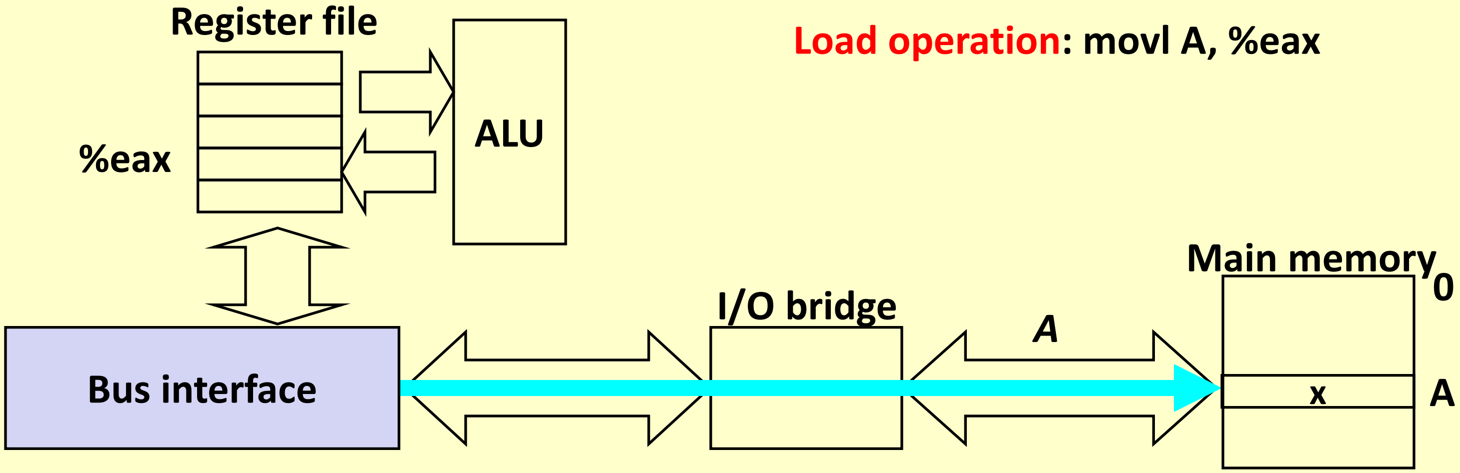


Hardware Organization — 2005 era Pentium



Memory Read Transaction (1)

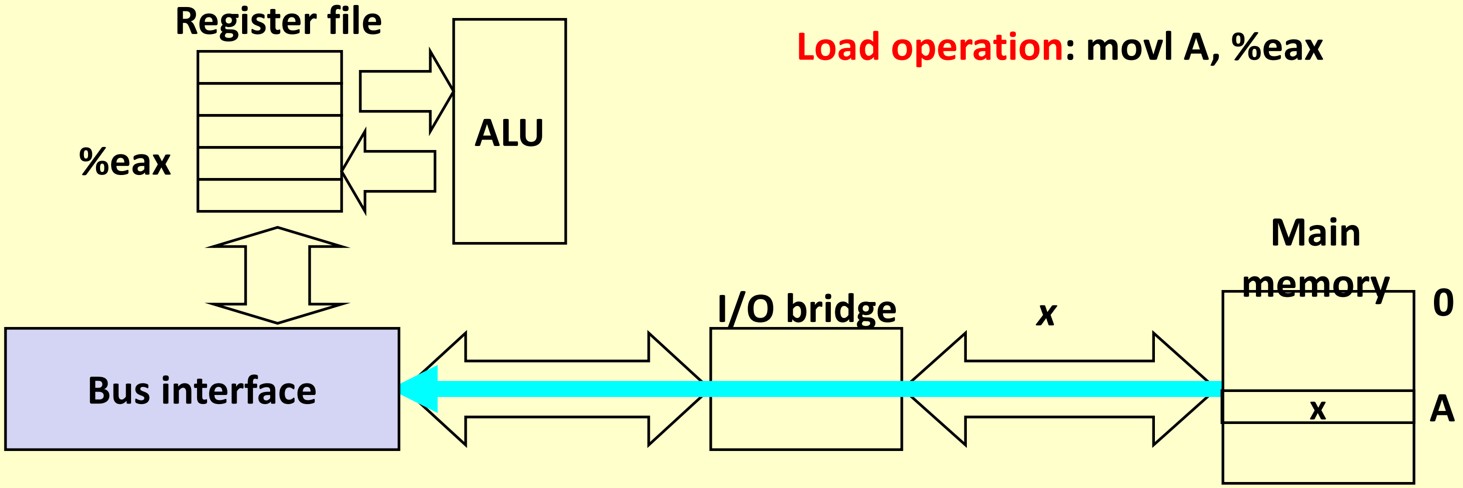
- CPU places address A on the memory bus.



Load operation: movl A, %eax

Memory Read Transaction (2)

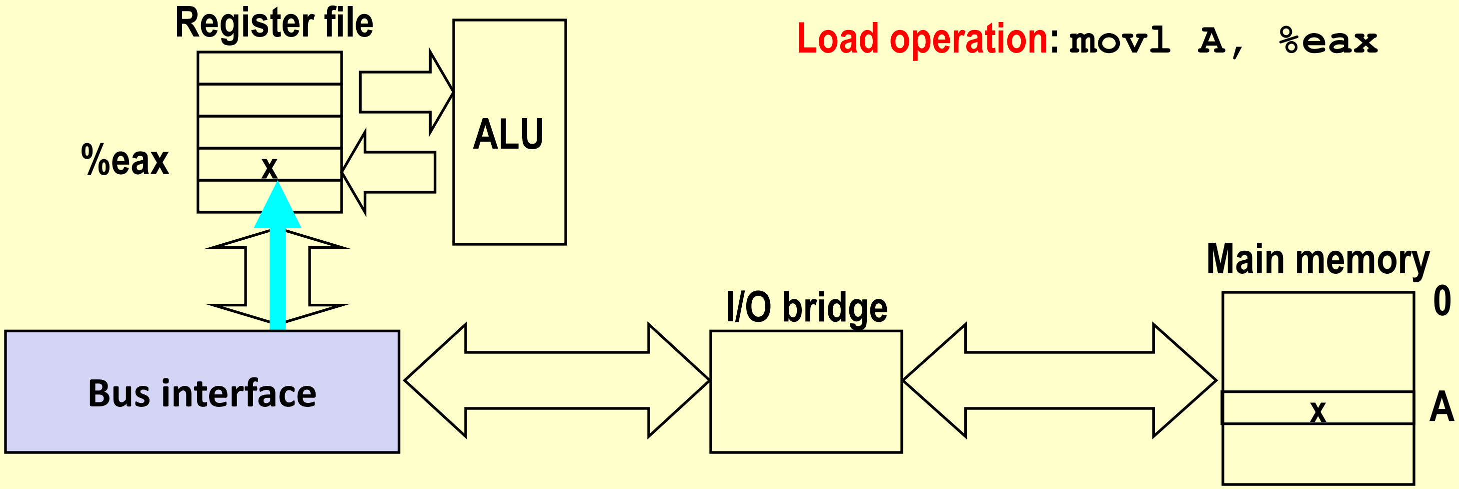
- Main memory reads A from the memory bus, retrieves word x, and places it on the bus.



Load operation: movl A, %eax

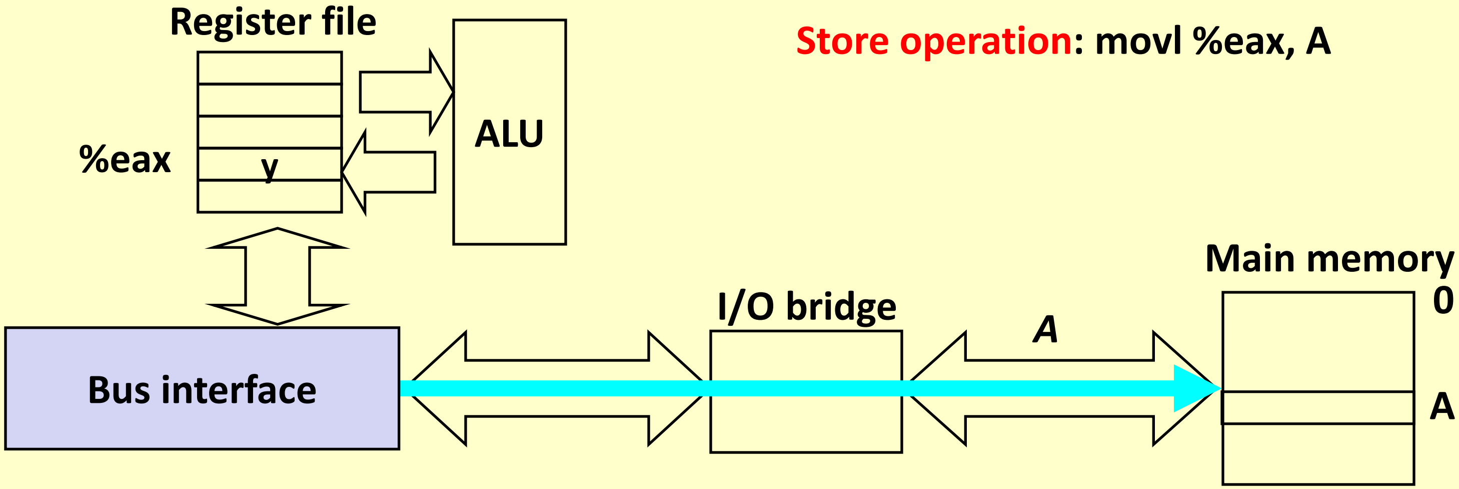
Memory Read Transaction (3)

- **CPU read word x from the bus and copies it into register %eax.**



Memory Write Transaction (1)

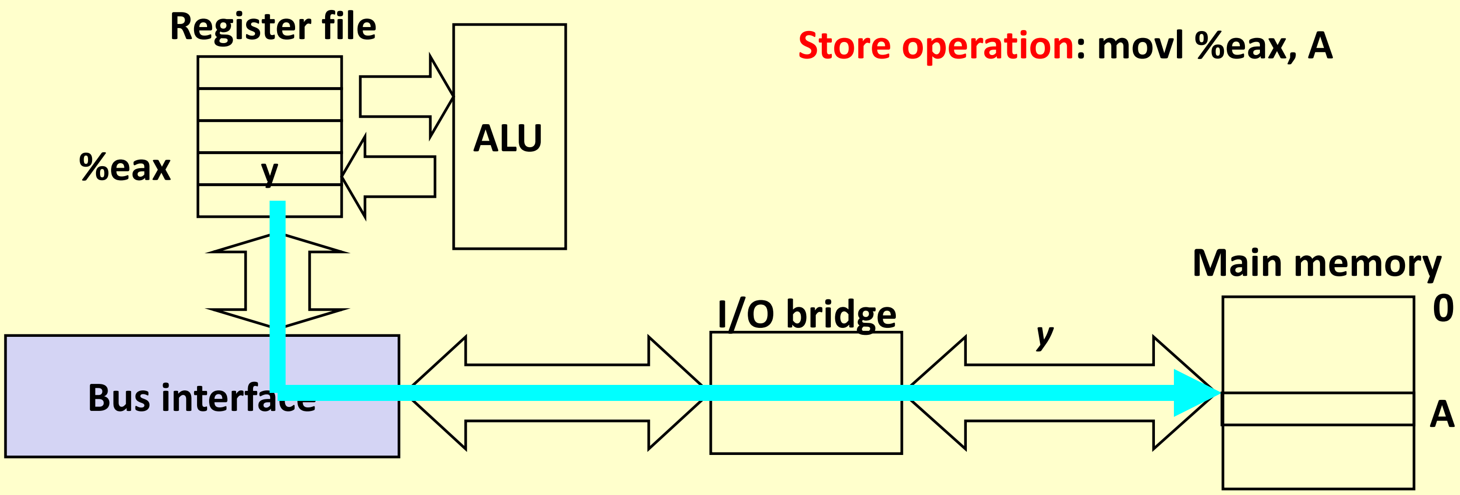
- CPU places address A on bus. Main memory reads it and waits for the corresponding data word to arrive.



Store operation: movl %eax, A

Memory Write Transaction (2)

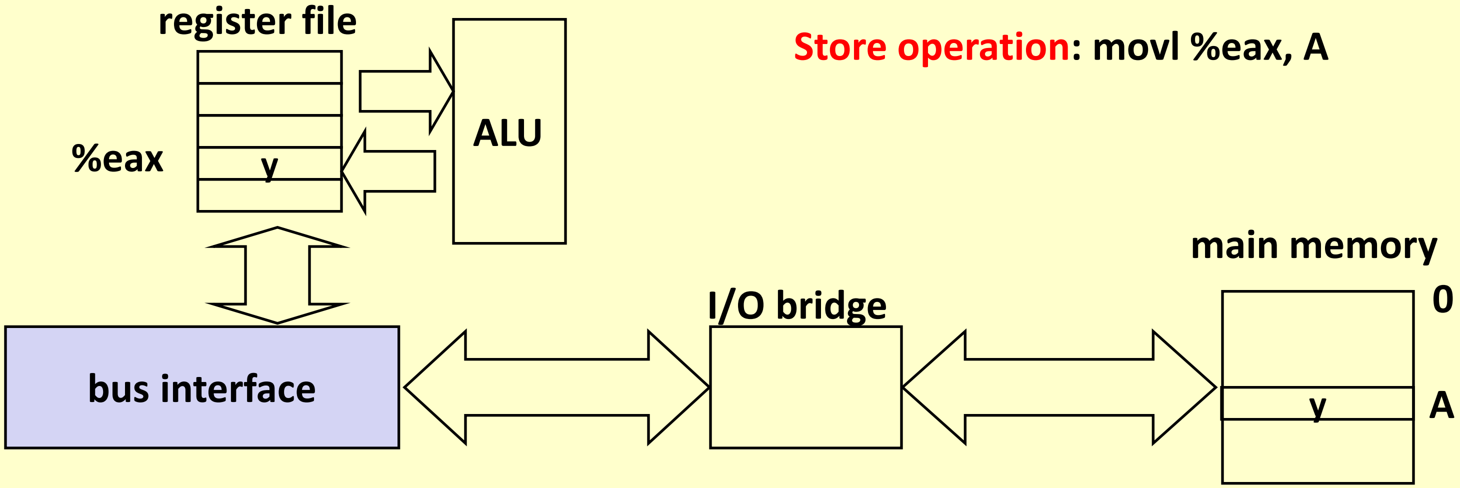
- CPU places data word y on the bus.



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

Questions?

What's Inside A Disk Drive?

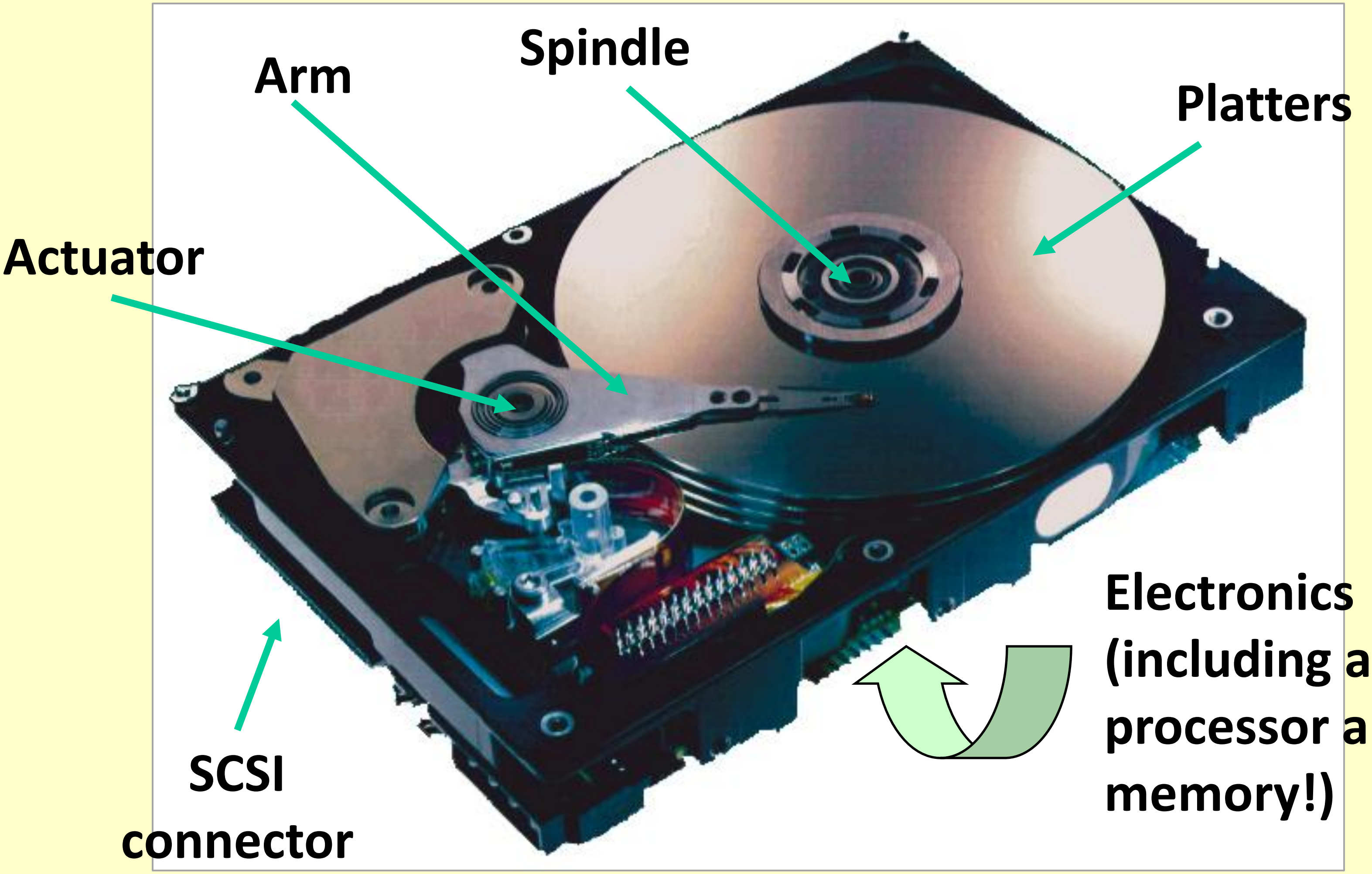
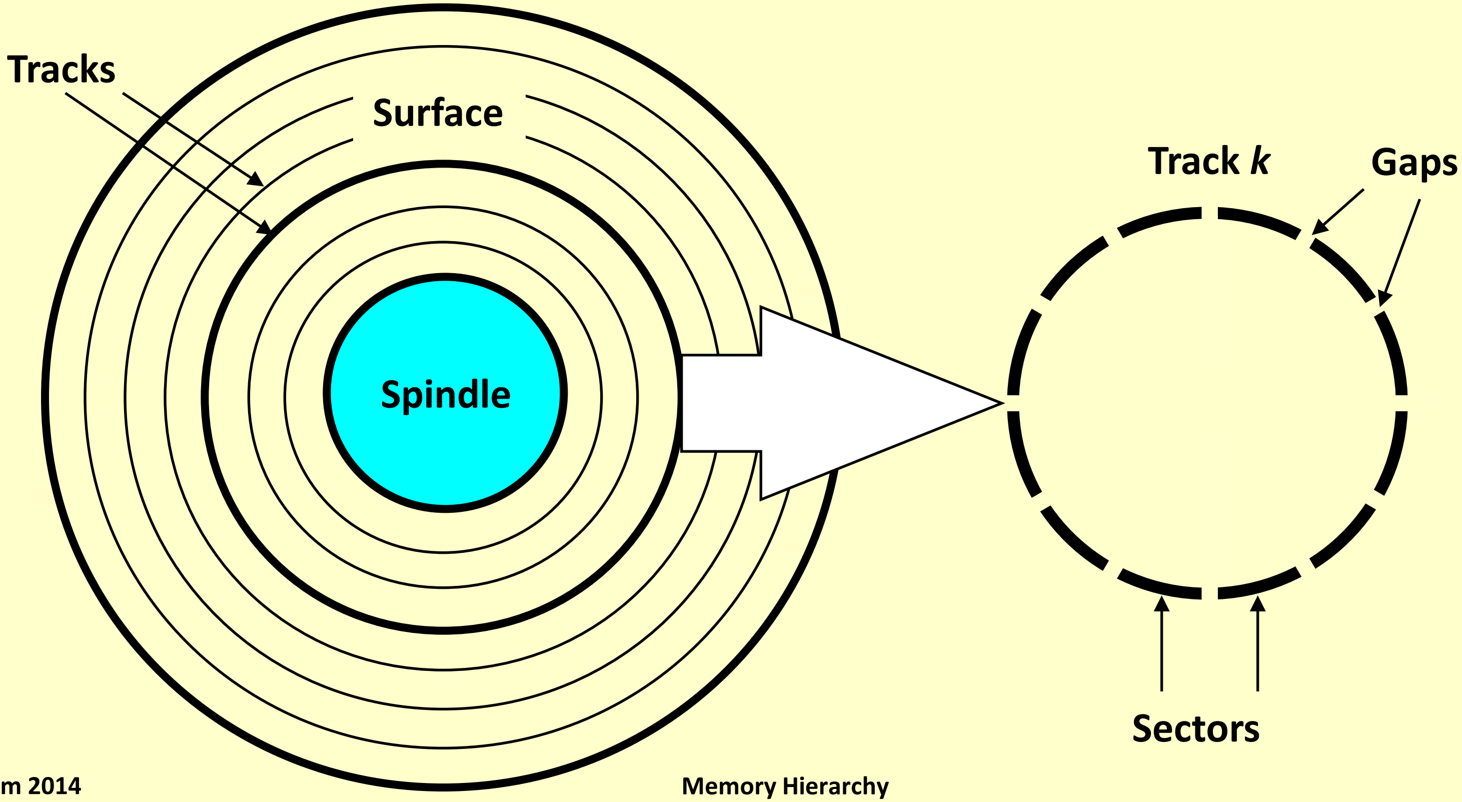


Image courtesy of Seagate Technology

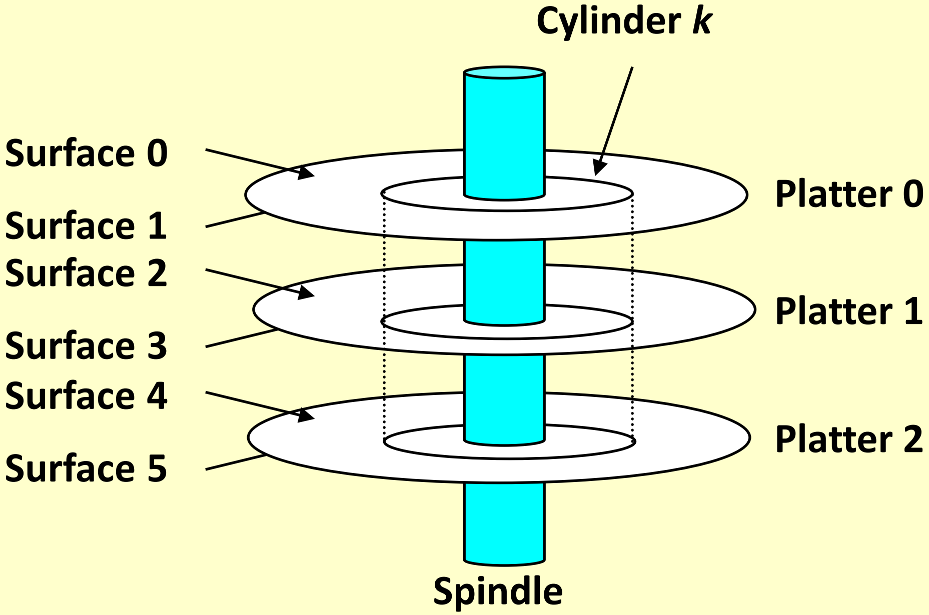
Disk Geometry

- 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 (Multiple-Platter View)

- **Aligned tracks form a cylinder.**



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 (Lawsuit pending! Claims deceptive advertising).
- **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)

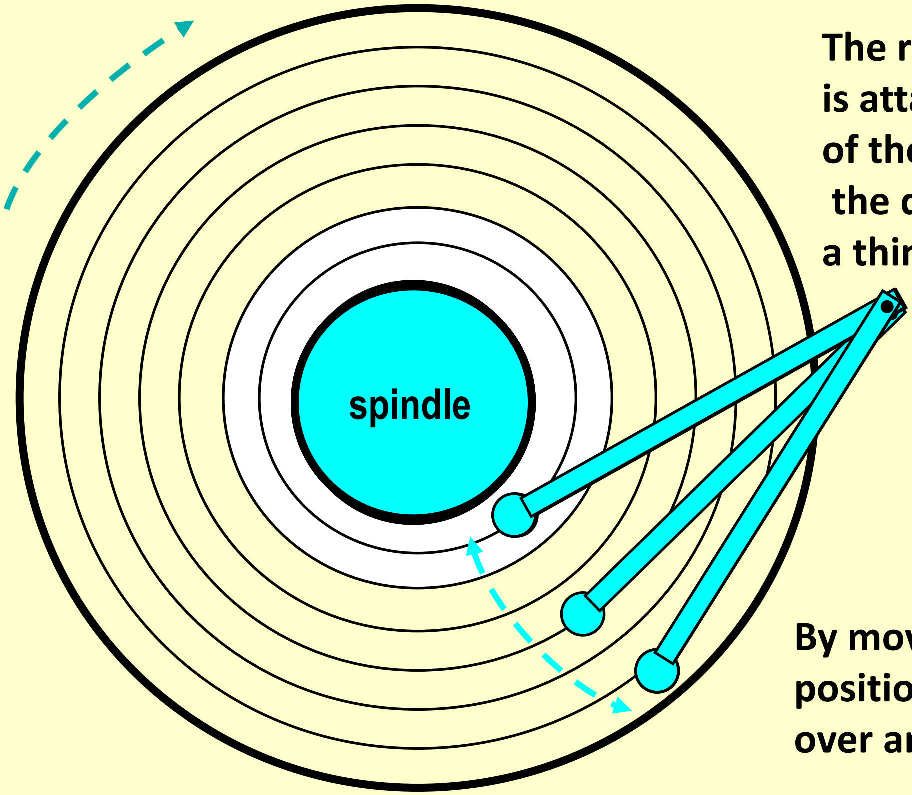
Example:

- 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)

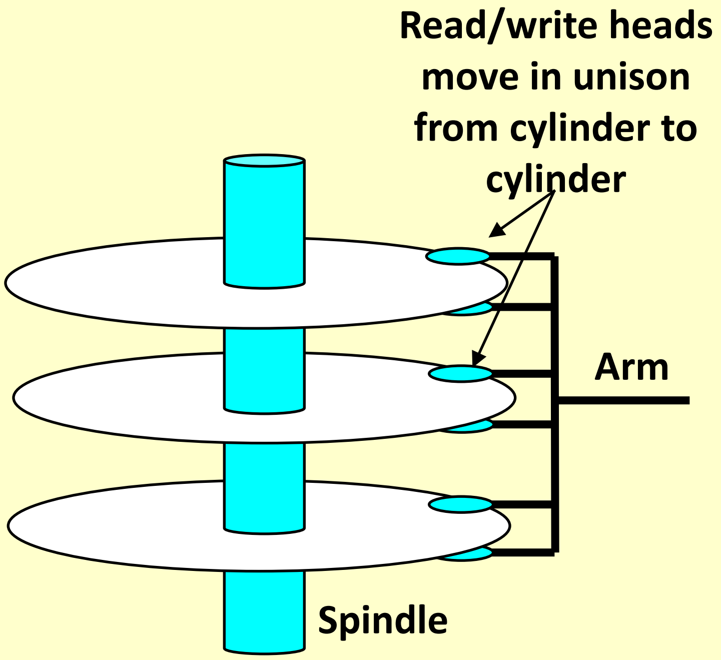
The disk surface spins at a fixed rotational rate



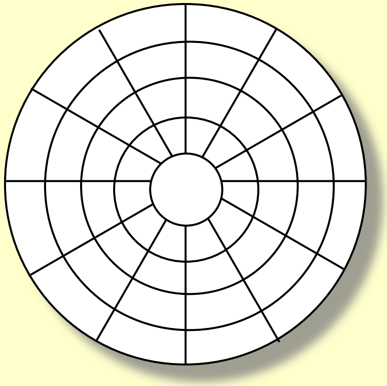
The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air.

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

Disk Operation (Multi-Platter View)



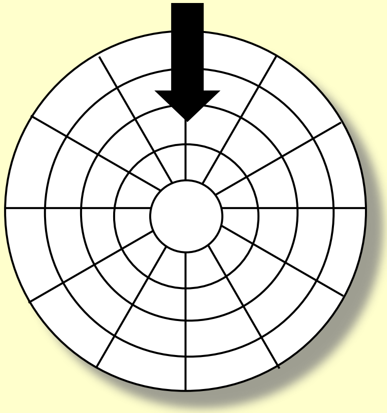
Disk Structure - top view of single platter



Surface organized into tracks

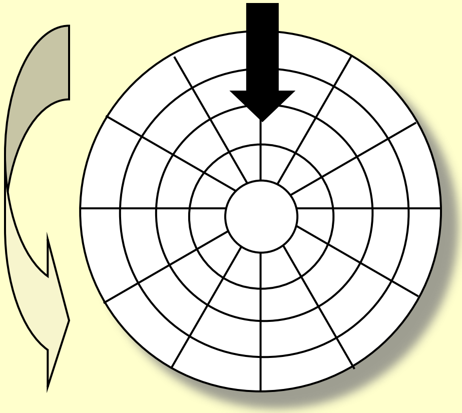
Tracks divided into sectors

Disk Access



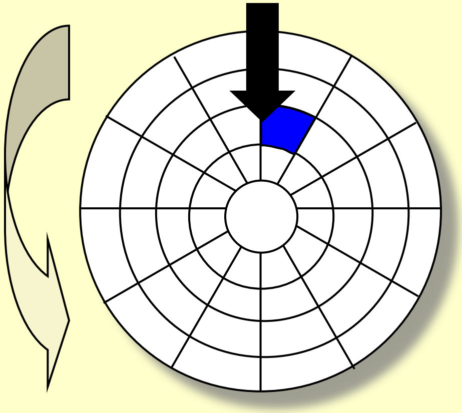
Head in position above a track

Disk Access



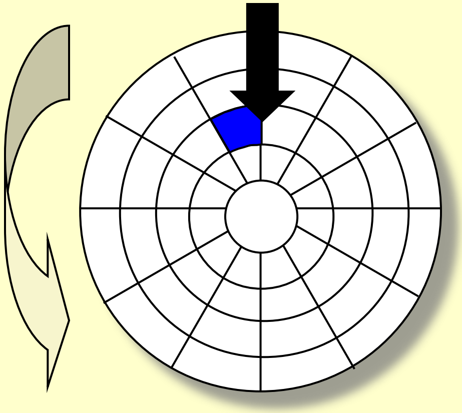
Rotation is counter-clockwise

Disk Access – Read



About to read blue sector

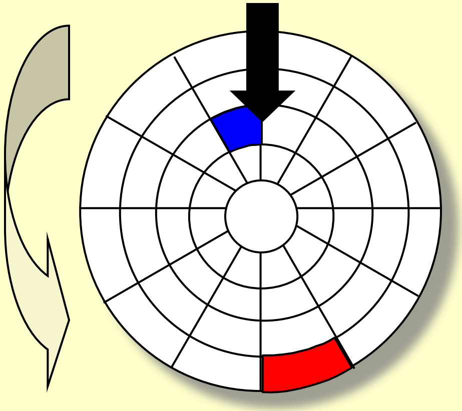
Disk Access – Read



After **BLUE** read

After reading blue sector

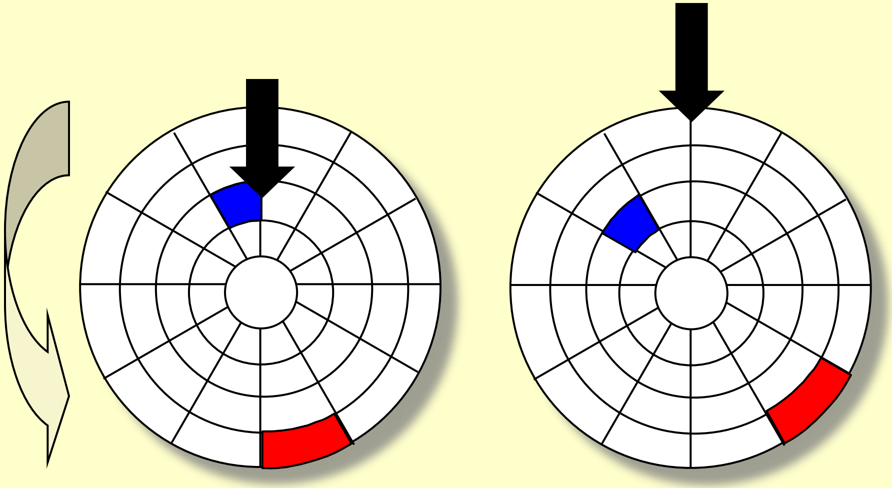
Disk Access – Read



After **BLUE** read

Red request scheduled next

Disk Access – Seek

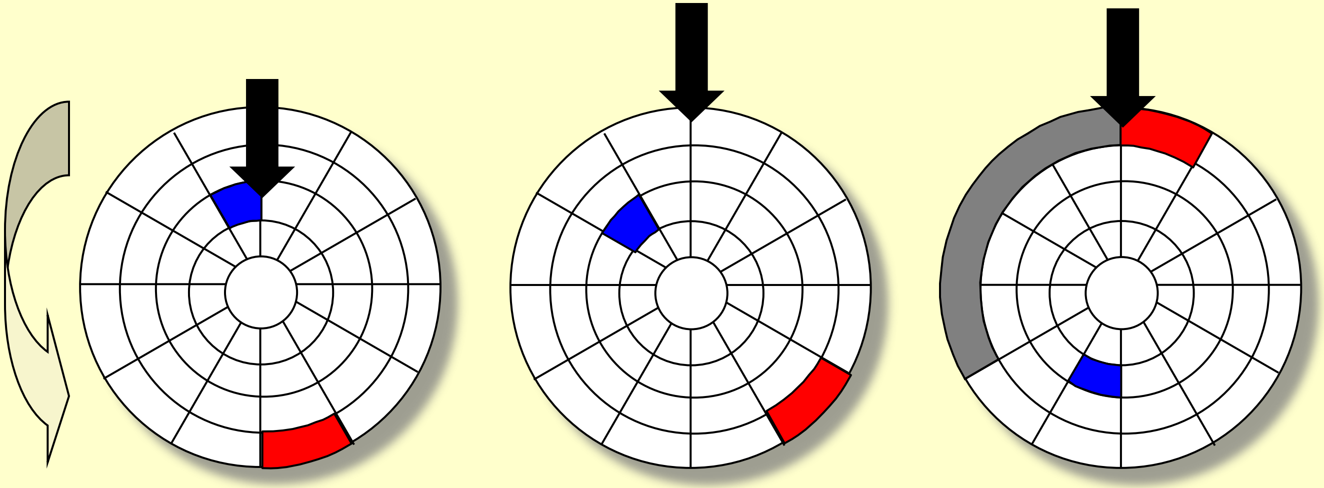


After **BLUE** read

Seek for **RED**

Seek to red's track

Disk Access – Rotational Latency



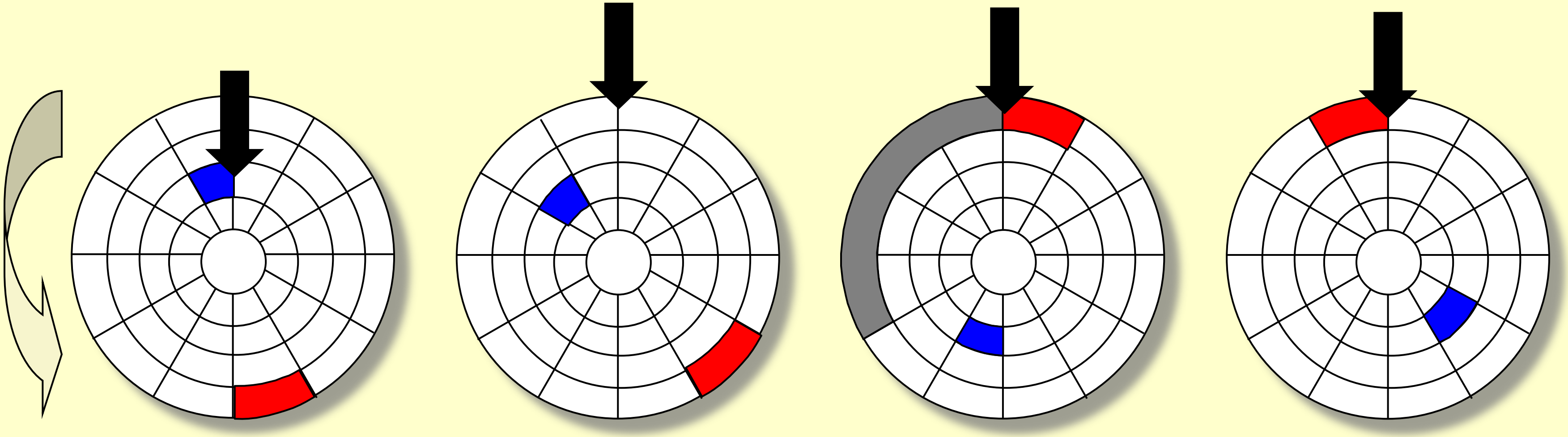
After **BLUE** read

Seek for **RED**

Rotational latency

Wait for red sector to rotate around

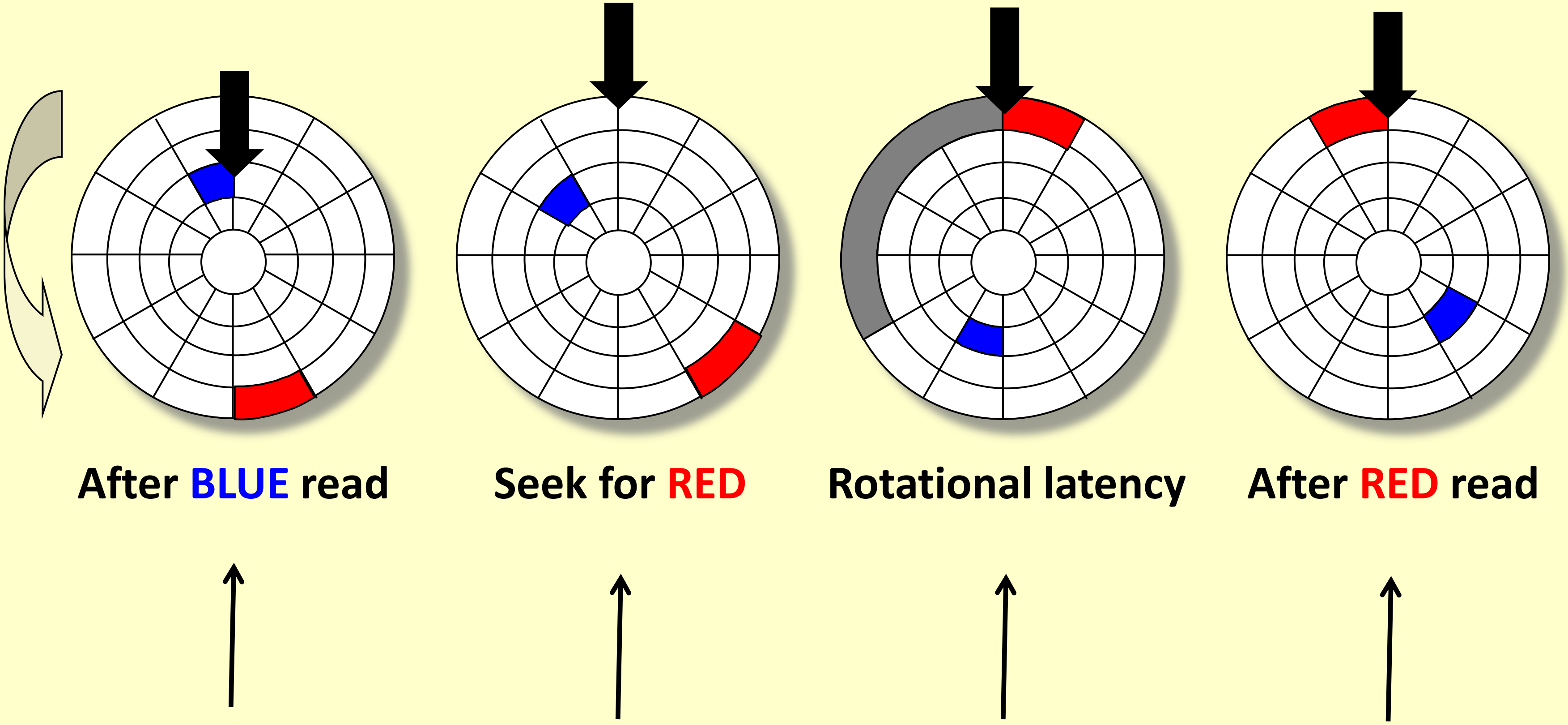
Disk Access – Read



After **BLUE** read   Seek for **RED**   Rotational latency   After **RED** read

Complete read of red

Disk Access – Service Time Components



Data transfer

Seek

Rotational latency

Data transfer

Disk Access Time

- **Average time to access some target sector approximated by :**
  - Taccess = 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 is 3—9 ms
- **Rotational latency (T_{avg} rotation)**
  - Time waiting for first bit of target sector to pass under r/w head.
  - T_{avg} rotation = 1/2 x 1/RPMs x 60 sec/1 min
  - Typical T_{avg} rotation = 7200 RPMs
- **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.

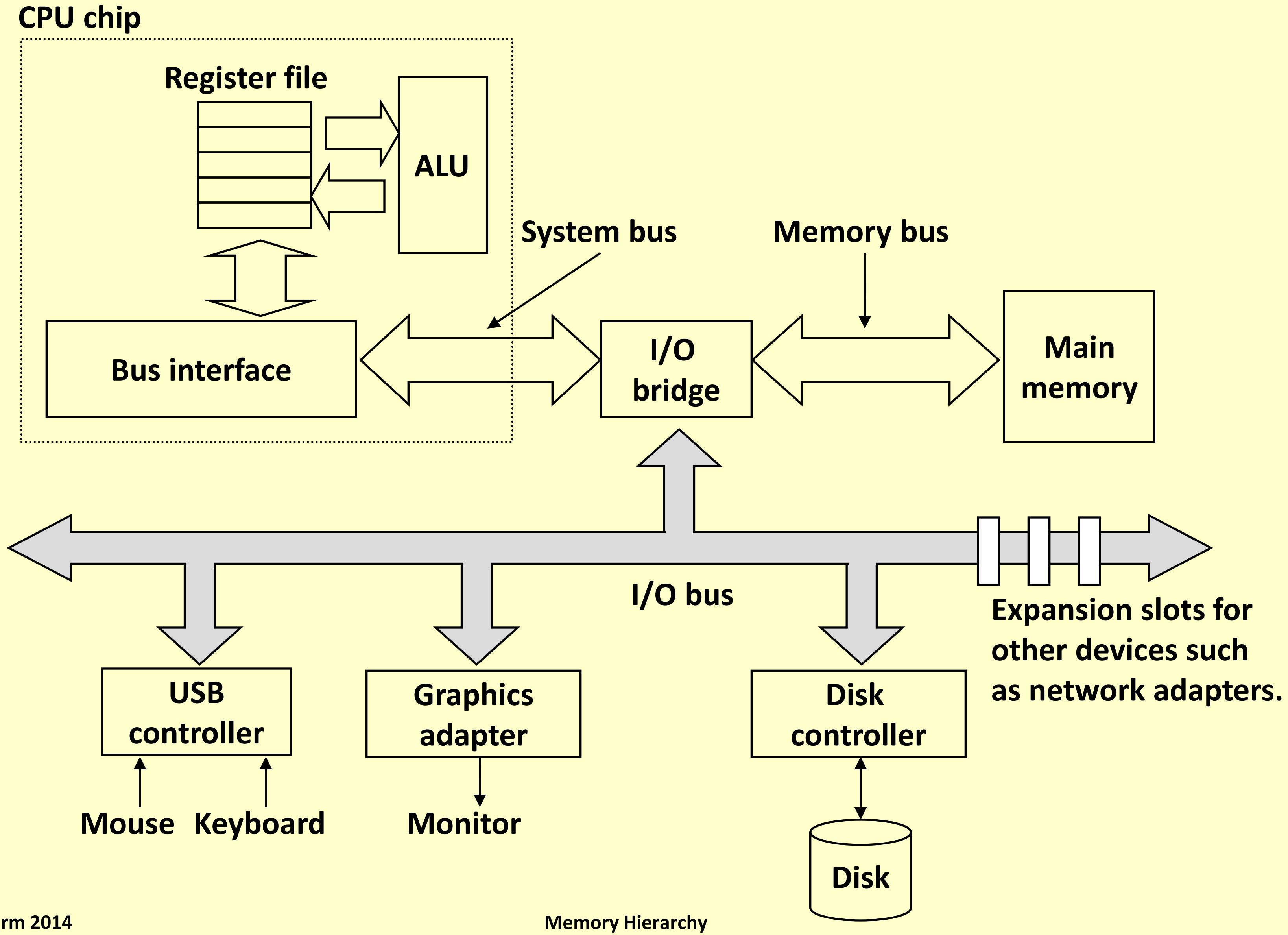
Disk Access Time Example

- **Given:**
  - Rotational rate = 7,200 RPM
  - Average seek time = 9 ms.
  - Avg # sectors/track = 400.
- **Derived:**
  - T_{avg} rotation = 1/2 x (60 secs/7200 RPM) x 1000 ms/sec = 4 ms.
  - T_{avg} transfer = 60/7200 RPM x 1/400 secs/track x 1000 ms/sec = 0.02 ms
  - T_{access} = 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.

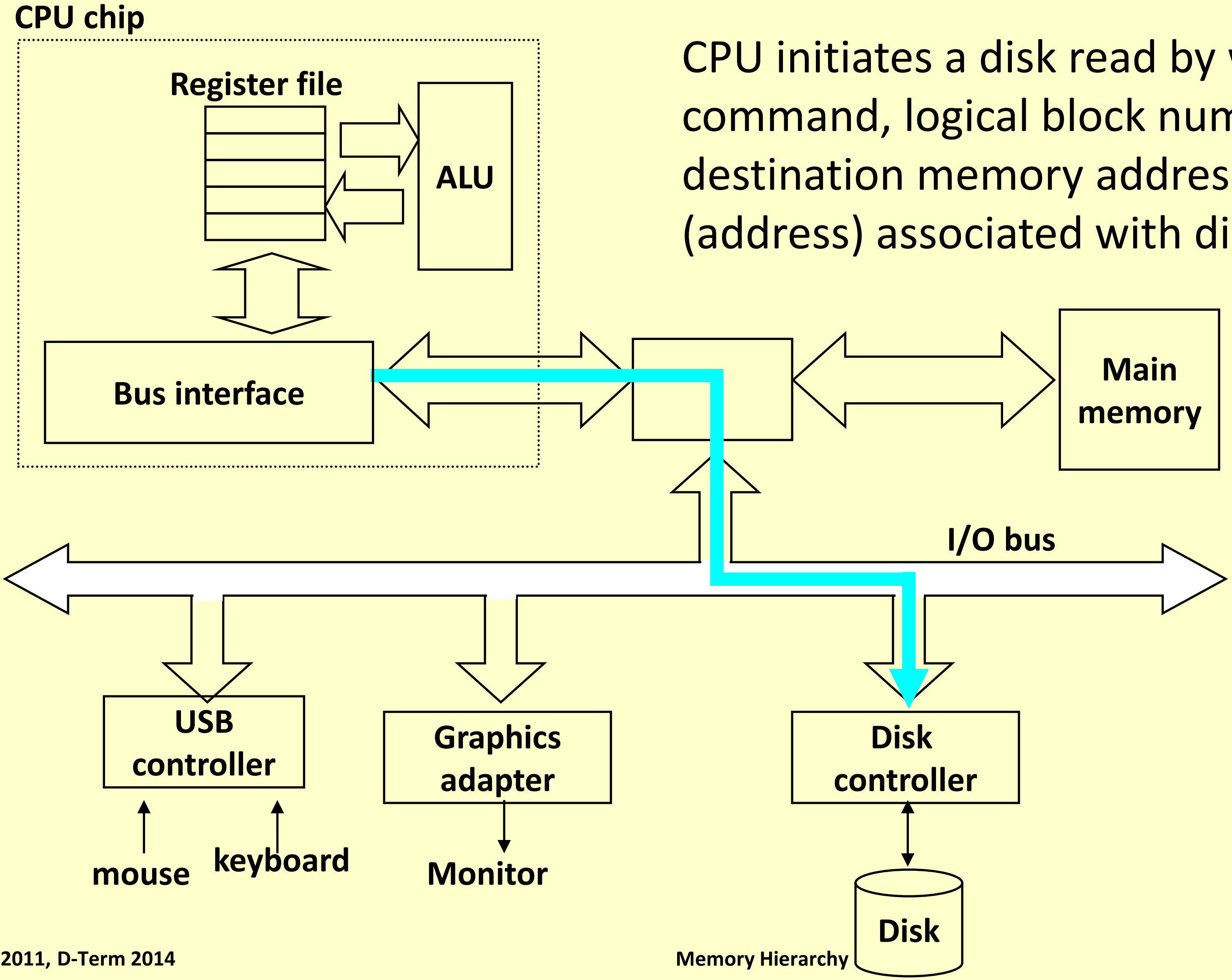
Logical Disk Blocks

- **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

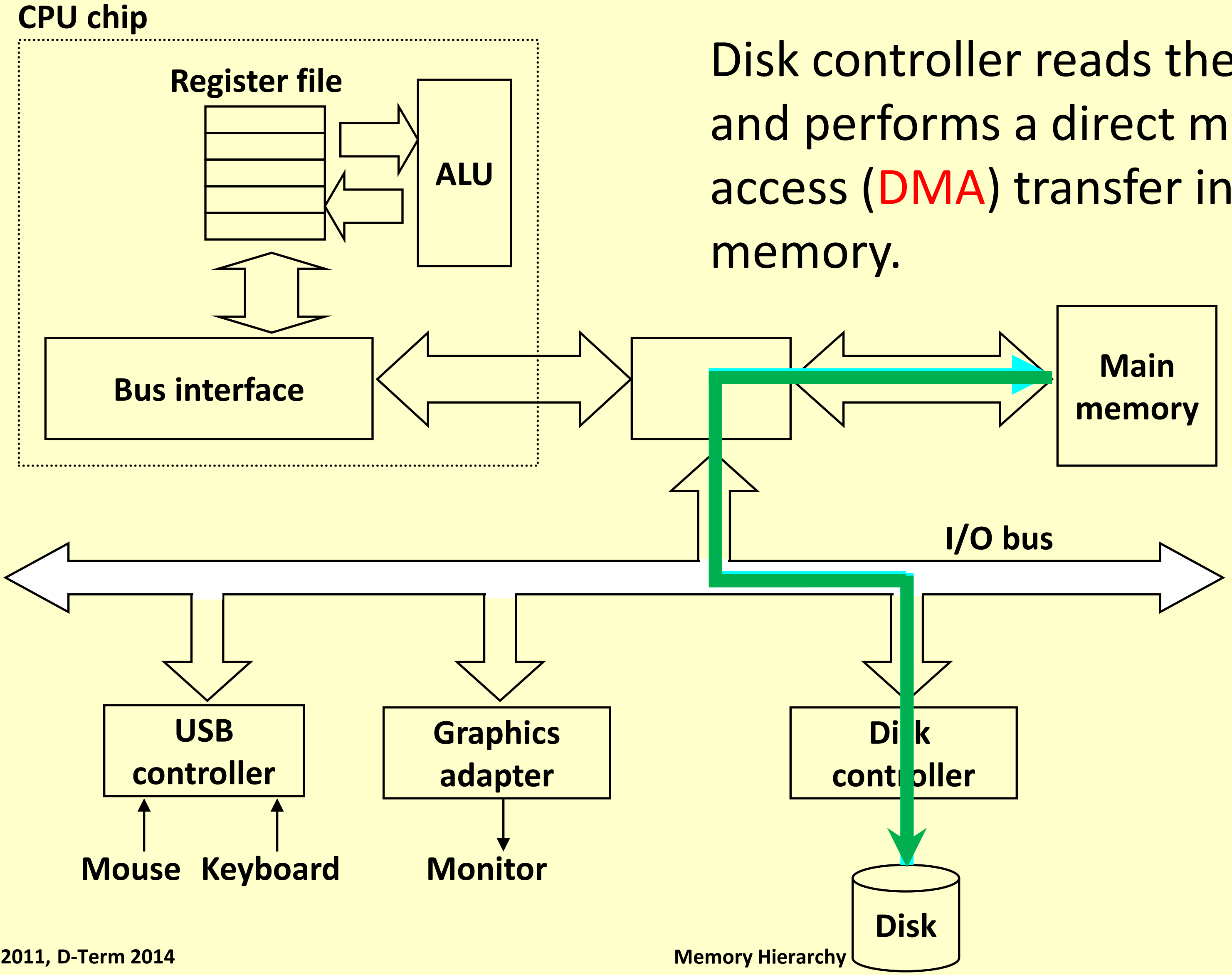

CPU chip
Register file
ALU
System bus
Memory bus
Bus interface
I/O bridge
Main memory
I/O bus
Expansion slots for other devices such as network adapters.
USB controller
Graphics adapter
Disk controller
Mouse
Keyboard
Monitor
Disk

Reading a Disk Sector (1)



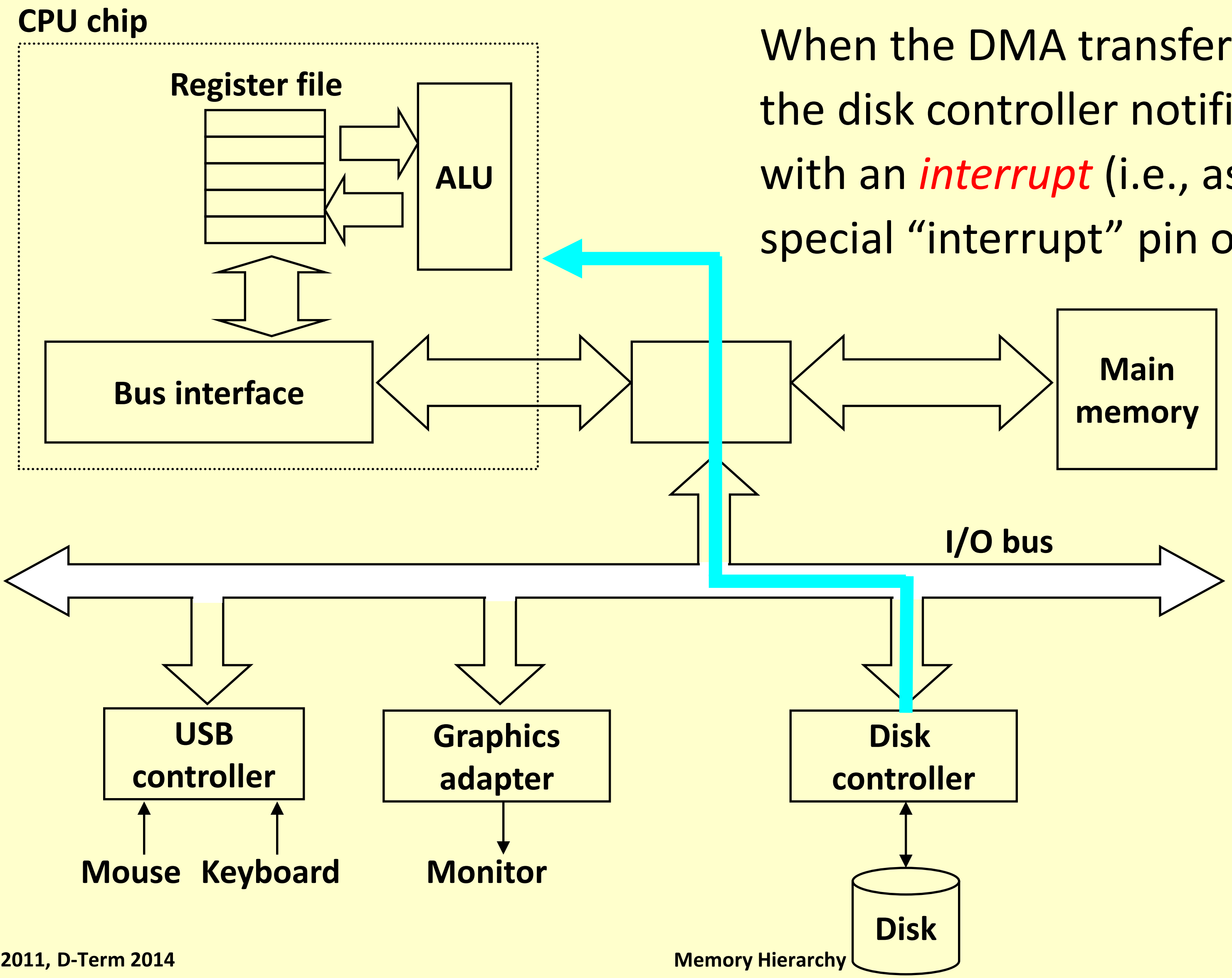
CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.

Reading a Disk Sector (2)



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

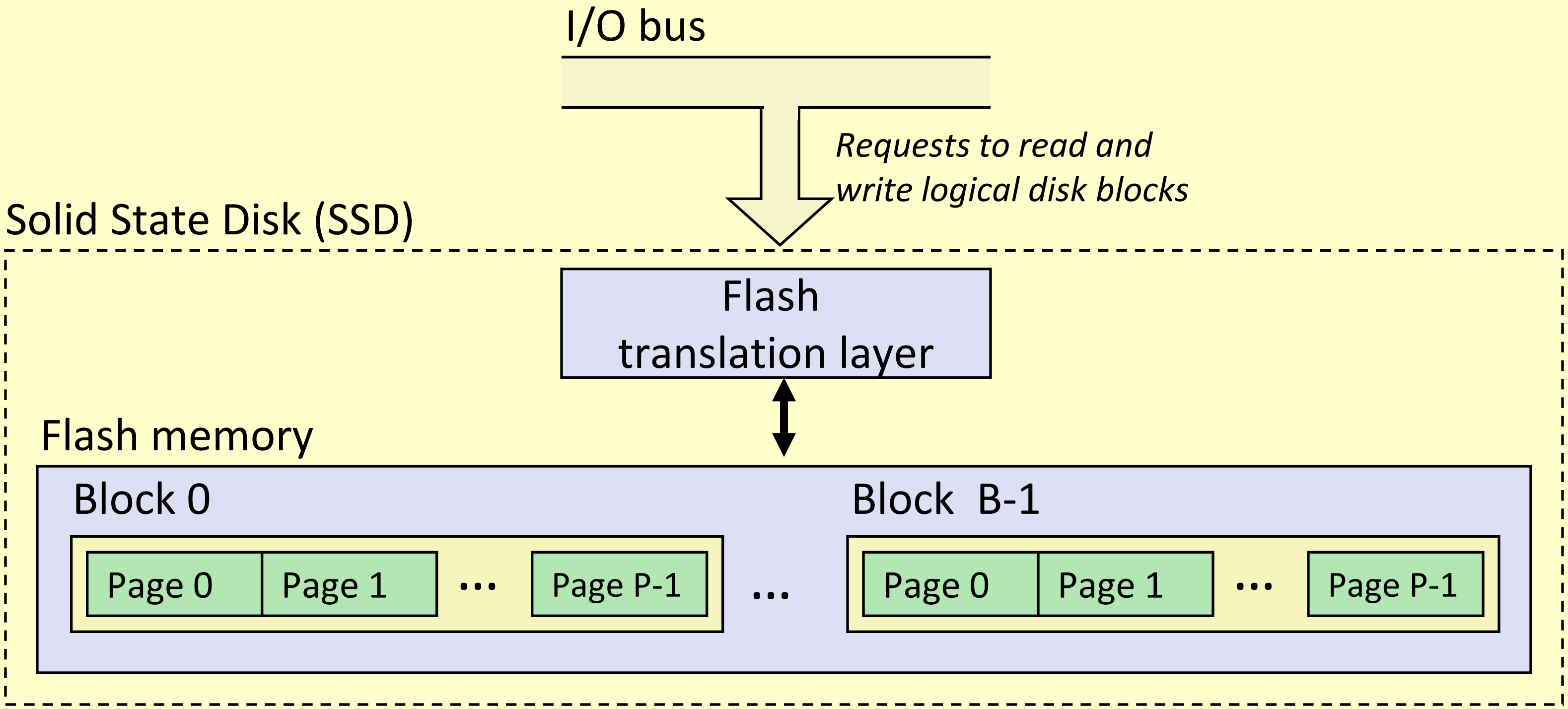
Reading a Disk Sector (3)



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

Questions?

Solid State Disks (SSDs)



- Pages: 512-byte to 4-kilobyte, 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 thru-put	250 MB/s	Sequential write thru-put	170 MB/s
Random read thru-put	140 MB/s	Random write thru-put	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)
  - Write to a page triggers a copy of all useful pages in the block
    - Find an unused 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
  - Some desktops and servers

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*

Questions?

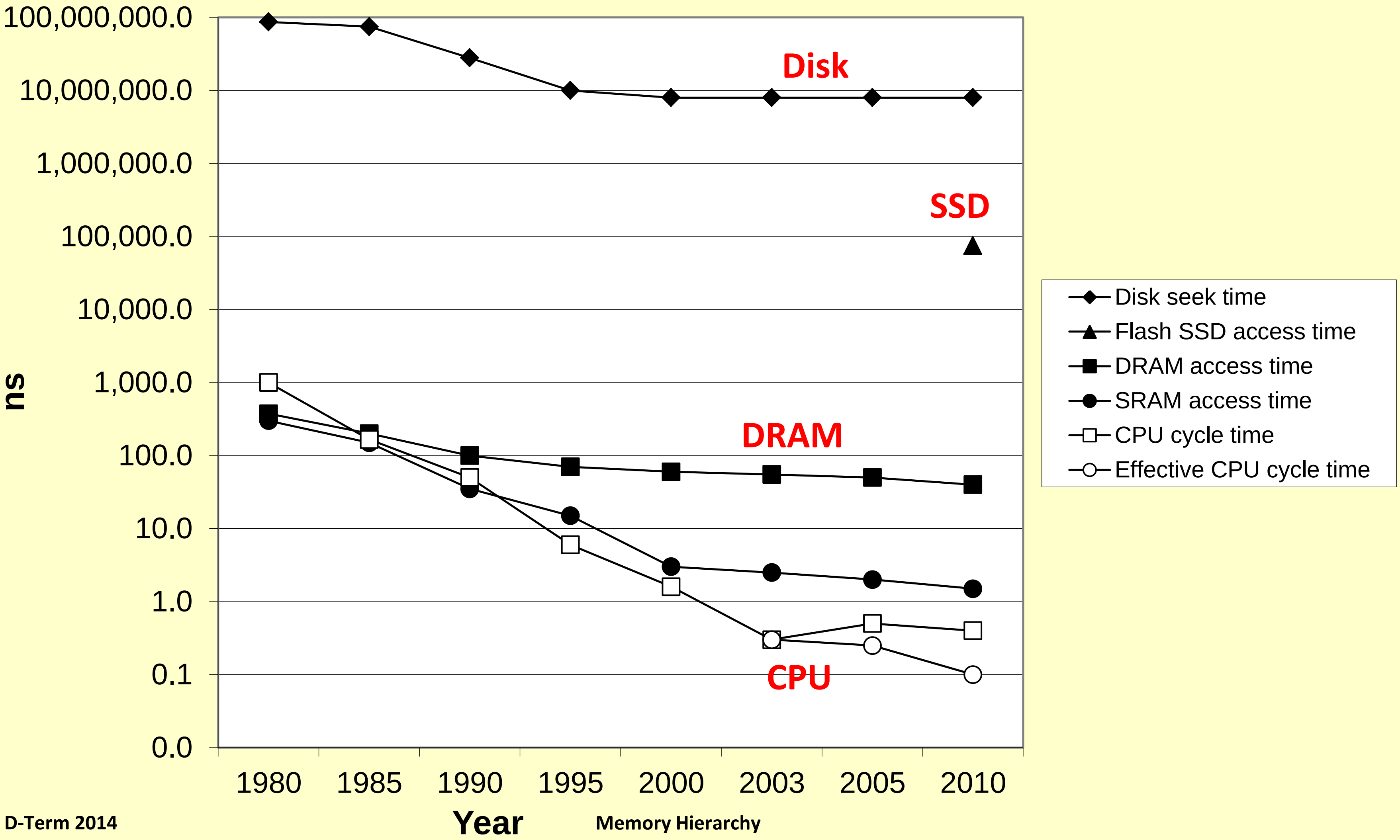
CPU Clock Rates

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.



Locality to the Rescue!

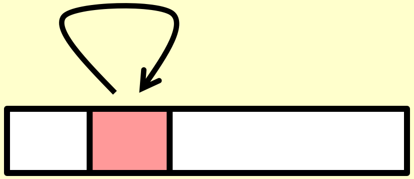
The key to bridging this CPU-Memory gap is a fundamental property of computer programs known as locality

Today

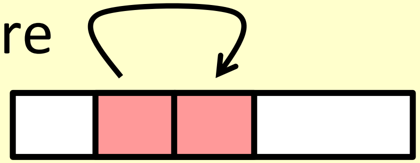
- Storage technologies and trends
- **Locality of reference**
- Caching in the memory hierarchy

Locality

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

- **Question:** 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;
}
```

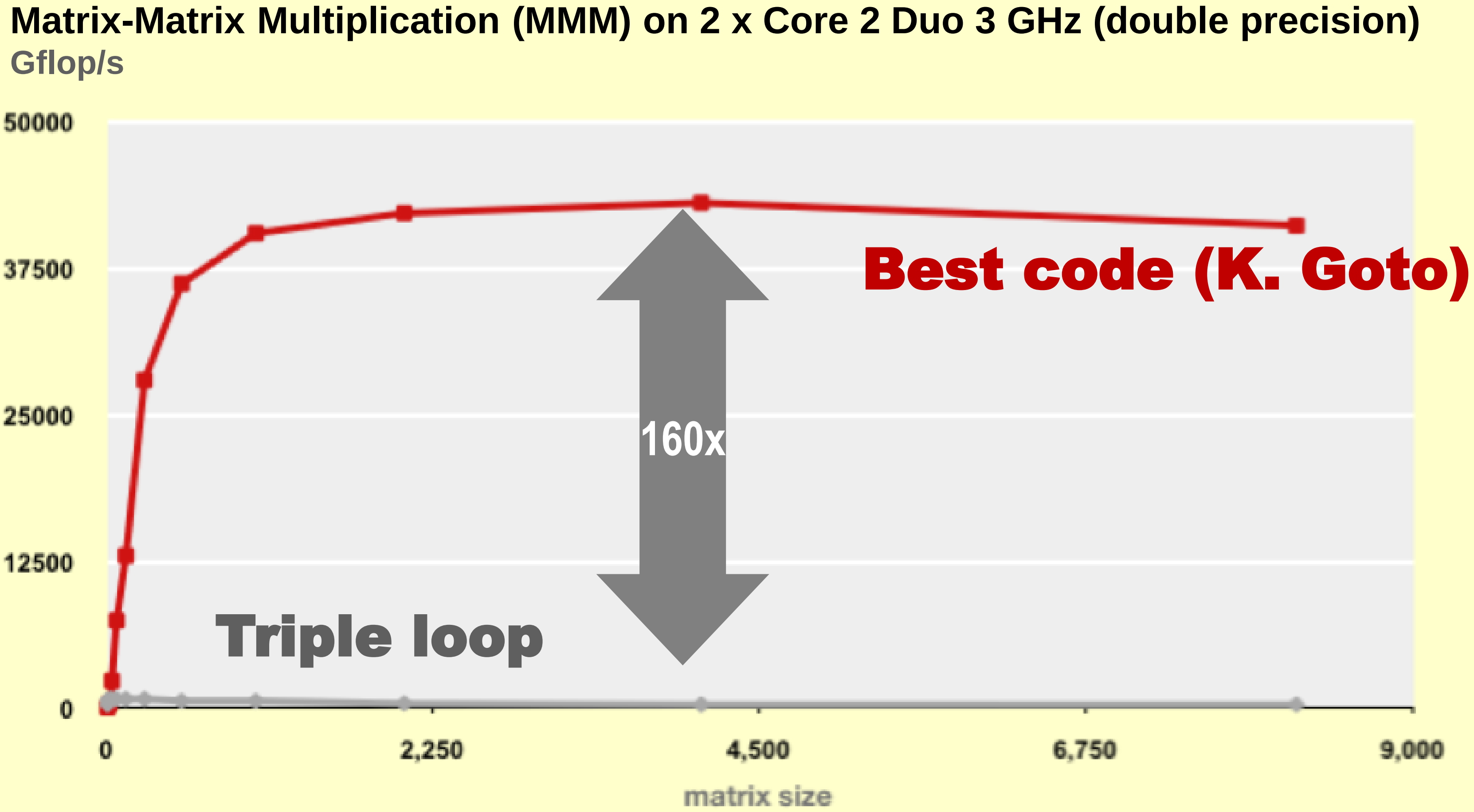
Locality Example

- **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)?**

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

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

Example Matrix Multiplication



- Standard desktop computer, vendor compiler, using optimization flags
- Both implementations have exactly the same operations count ($2n^3$)
- What is going on?

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)?

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

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < N; k++)
                sum += a[i][j][k];
    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.**

Today

- Storage technologies and trends
- Locality of reference
- Caching in the memory hierarchy

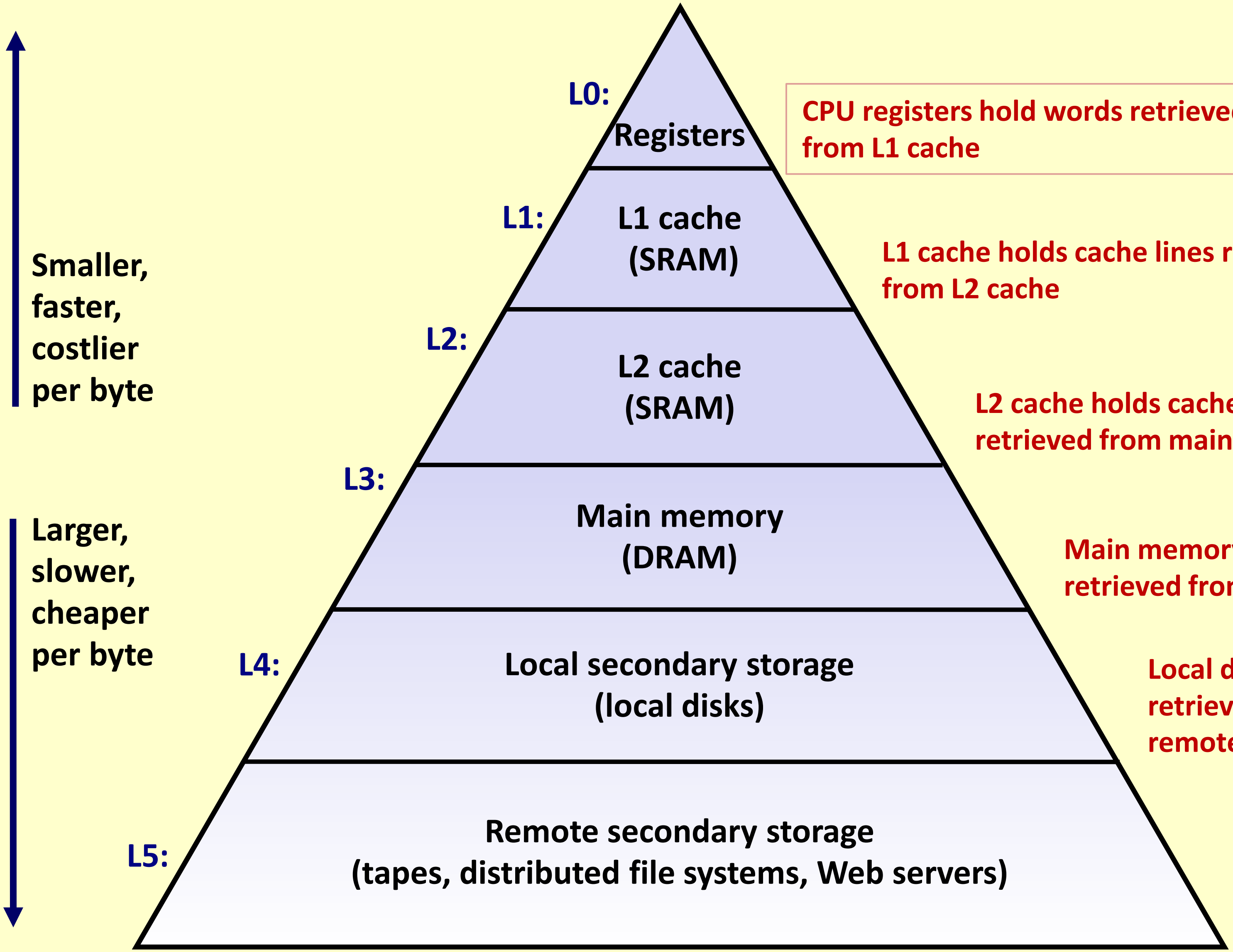
Definition:– Cache

- A small fast memory that holds a (frequently accessed) subset of items from a much larger, slower memory

- Reason:–

  To approximate the performance of the fast memory while retaining the size of the larger memory

An Example Memory Hierarchy



CPU registers hold words retrieved from L1 cache

L1 cache holds cache lines retrieved from L2 cache

L2 cache holds cache lines retrieved from main memory

Main memory holds disk blocks retrieved from local disks

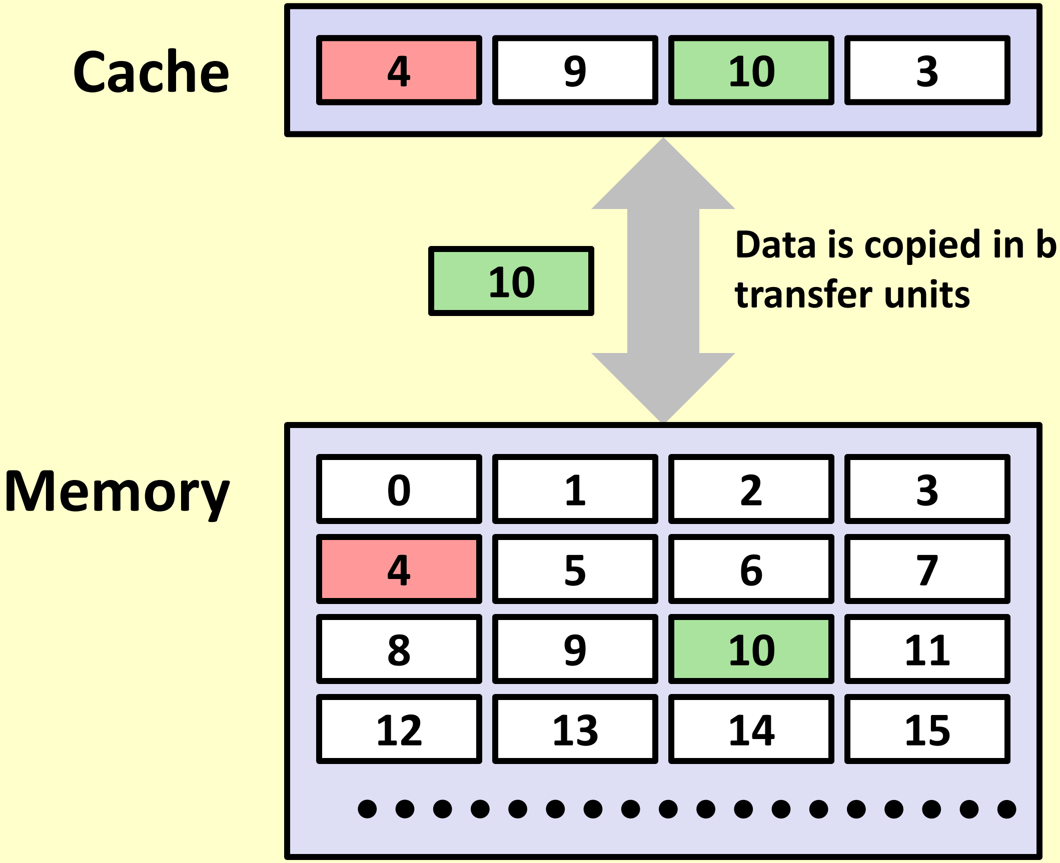
Local disks hold files retrieved from disks on remote network servers

Fig. 6.23

Caches and Memory Hierarchies

- **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 illusion of a large pool of storage …**
  - **… as big and as cheap as the bottom layer**
  - **… as fast as the top layer**

Caches in Microprocessors



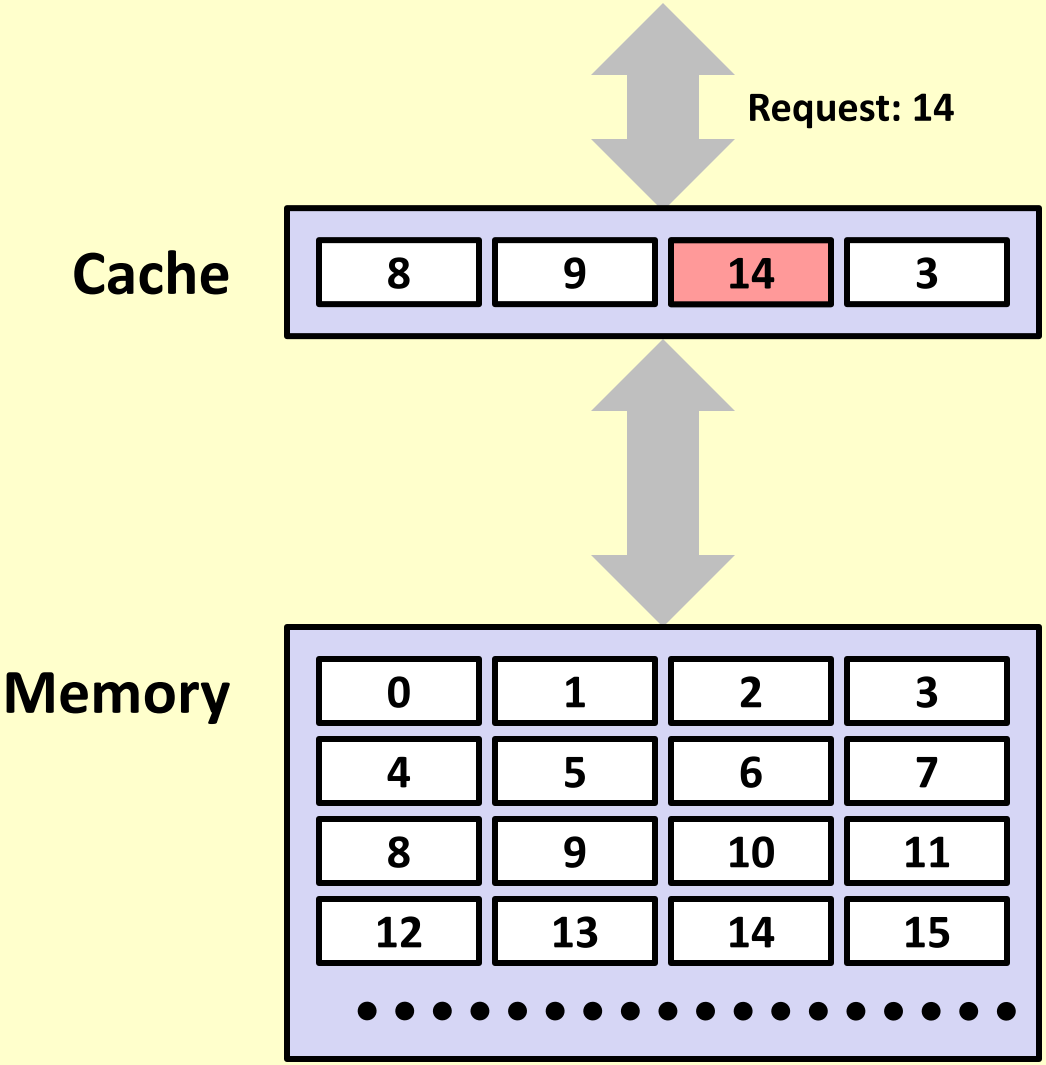
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

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

Data is copied in block-sized transfer units

Larger, slower, cheaper memory viewed as partitioned into “blocks”

General Cache Concepts: Hit



Request: 14

Cache

8	9	14	3

Memory

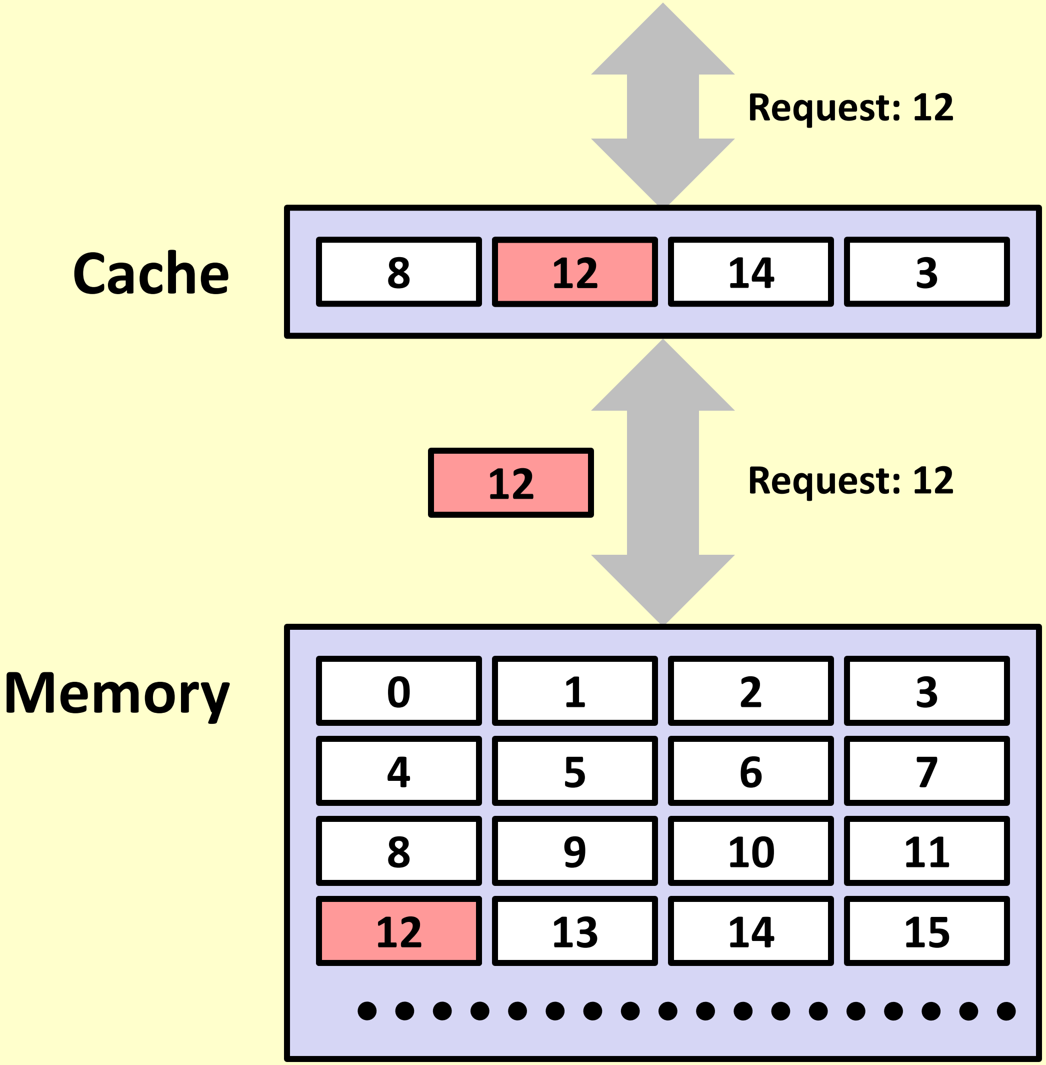
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ●

Data in block b is needed

Block b is in cache:
Hit!

General Cache Concepts: Miss



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 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	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

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.

Questions?