

The Memory Hierarchy

15-213 / 18-213: Introduction to Computer Systems
10th Lecture, Sep. 27, 2012

Instructors:

Dave O'Hallaron, Greg Ganger, and Greg Kesden

1

Today

- DRAM as building block for main memory
- Locality of reference
- Caching in the memory hierarchy
- Storage technologies and trends

2

Byte-Oriented Memory Organization



- **Programs refer to data by address**
 - Conceptually, envision it as a very large array of bytes
 - In reality, it's not, but can think of it that way
 - An address is like an index into that array
 - and, a pointer variable stores an address
- **Note: system provides private address spaces to each "process"**
 - Think of a process as a program being executed
 - So, a program can clobber its own data, but not that of others

From 2nd lecture 3

Simple Memory Addressing Modes

- **Normal (R) Mem[Reg[R]]**
 - Register R specifies memory address
 - Aha! Pointer dereferencing in C

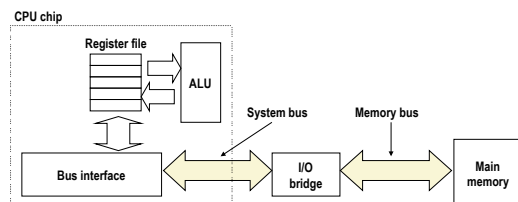
```
movl (%ecx), %eax
```
- **Displacement D(R) Mem[Reg[R]+D]**
 - Register R specifies start of memory region
 - Constant displacement D specifies offset

```
movl 8(%ebp), %edx
```

From 5th lecture 4

Traditional Bus Structure Connecting CPU and Memory

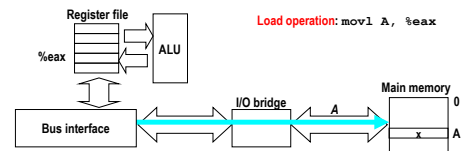
- A **bus** is a collection of parallel wires that carry address, data, and control signals.
- Buses are typically shared by multiple devices.



5

Memory Read Transaction (1)

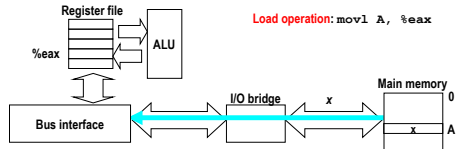
- CPU places address A on the memory bus.



6

Memory Read Transaction (2)

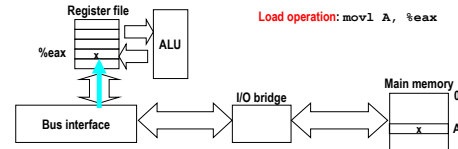
- Main memory reads A from the memory bus, retrieves word x, and places it on the bus.



7

Memory Read Transaction (3)

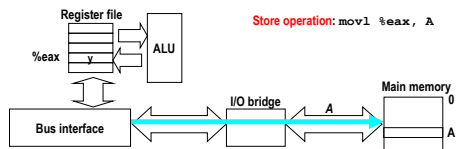
- CPU read word x from the bus and copies it into register %eax.



8

Memory Write Transaction (1)

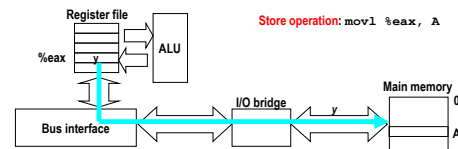
- CPU places address A on bus. Main memory reads it and waits for the corresponding data word to arrive.



9

Memory Write Transaction (2)

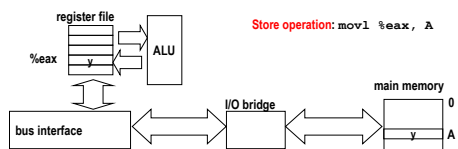
- CPU places data word y on the bus.



10

Memory Write Transaction (3)

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

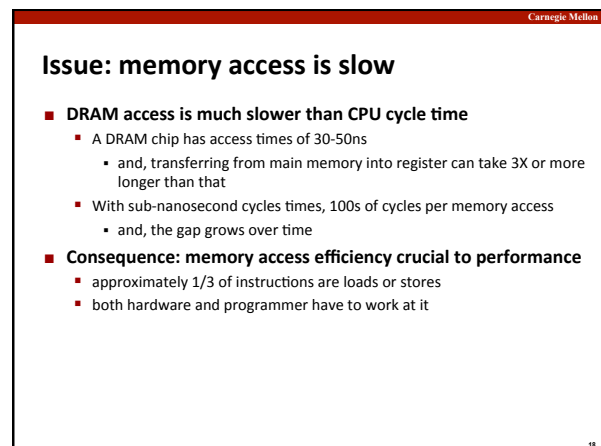
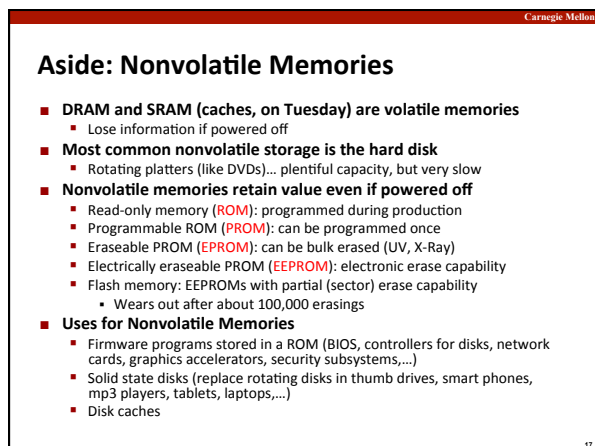
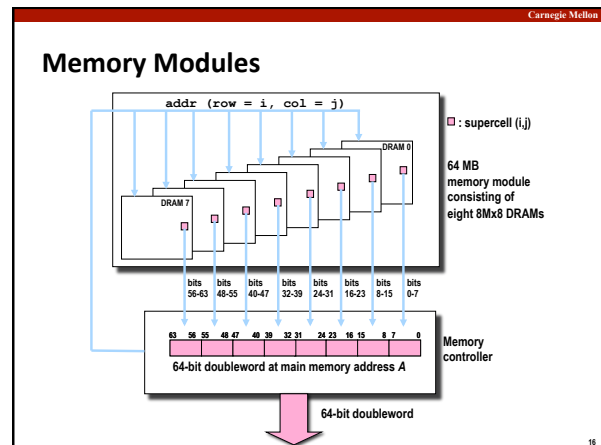
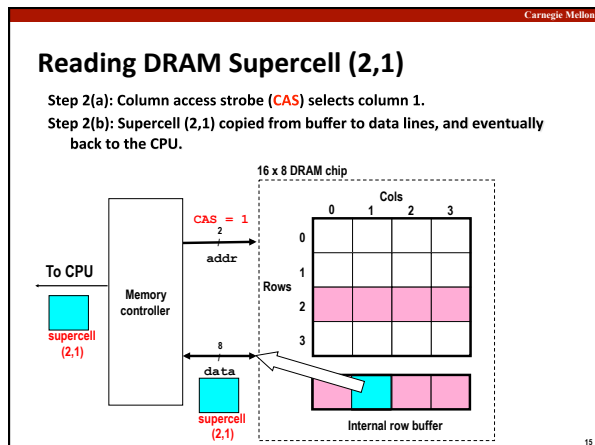
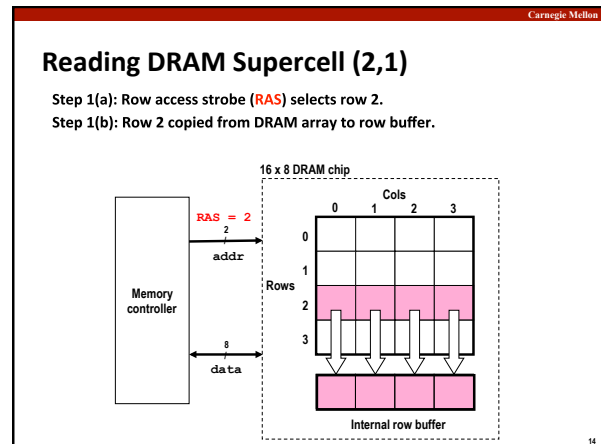
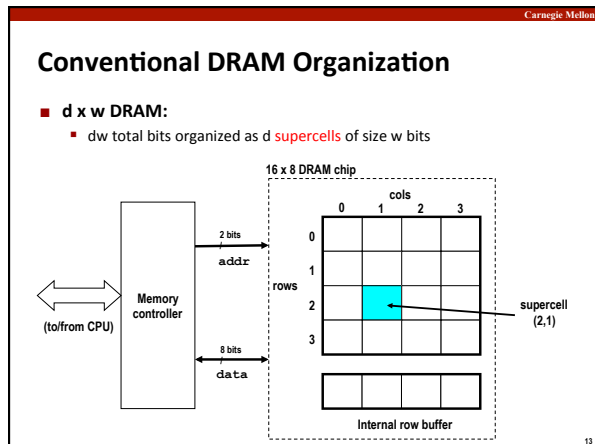


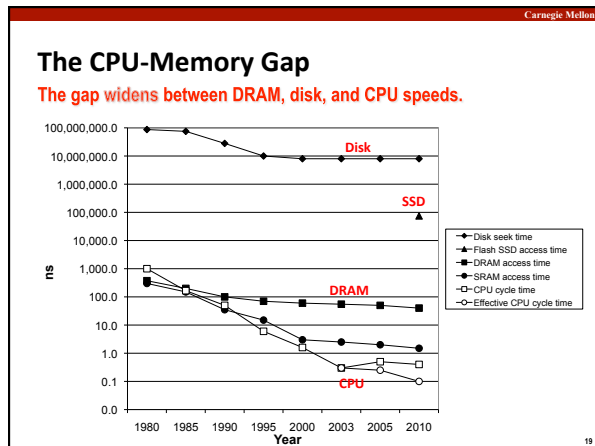
11

Dynamic Random-Access Memory (DRAM)

- **Key features**
 - DRAM is traditionally packaged as a chip
 - Basic storage unit is normally a cell (one bit per cell)
 - Multiple DRAM chips form main memory in most computers
- **Technical characteristics**
 - Organized in two dimensions (rows and columns)
 - To access (within a DRAM chip): select row then select column
 - Consequence: 2nd access to a row faster than different column/row
 - Each cell stores bit with a capacitor; one transistor is used for access
 - Value must be refreshed every 10-100 ms
 - Done within the hardware

12





Locality to the Rescue!

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

Today

- DRAM as building block for main memory
- **Locality of reference**
- Caching in the memory hierarchy
- Storage technologies and trends

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

- **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;
}
```

25

Today

- DRAM as building block for main memory
- Locality of reference
- Caching in the memory hierarchy
- Storage technologies and trends

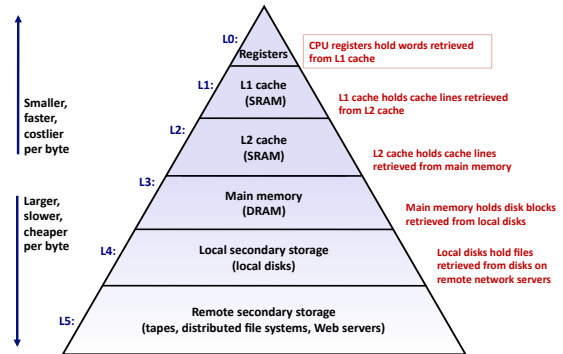
26

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**

27

An Example Memory Hierarchy



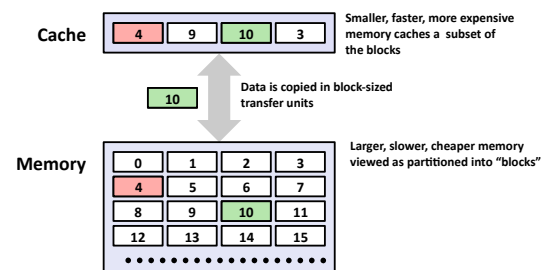
28

Caches

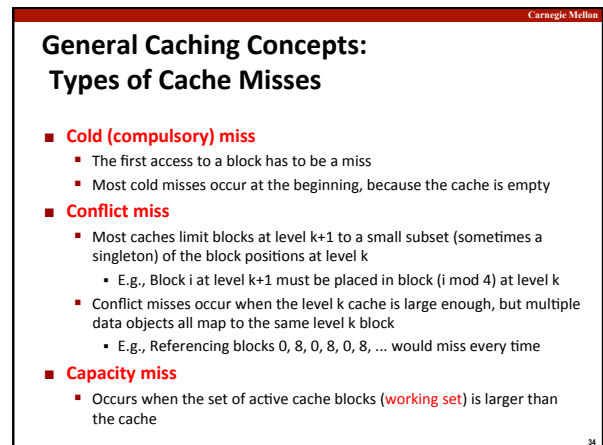
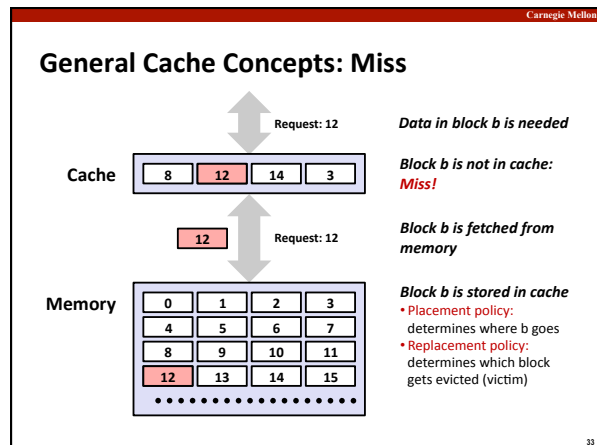
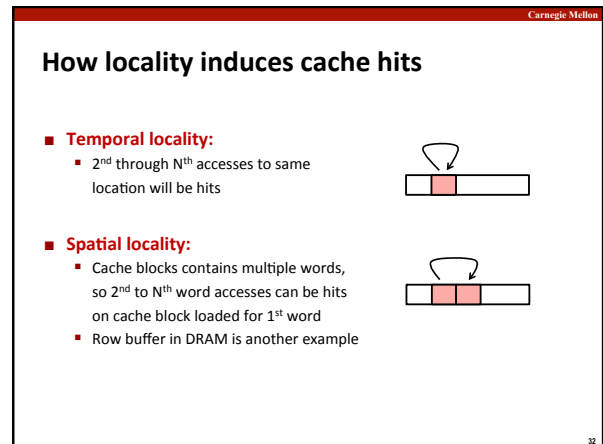
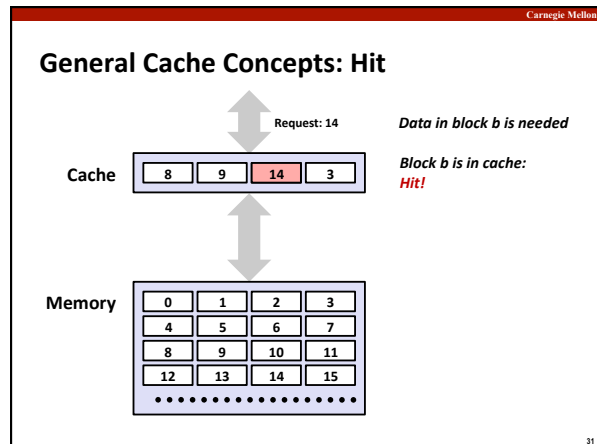
- **Cache:** A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.
- Fundamental idea of a memory hierarchy:
 - For each *k*, the faster, smaller device at level *k* serves as a cache for the larger, slower device at level *k*+1.
- Why do memory 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.

29

General Cache Concepts



30

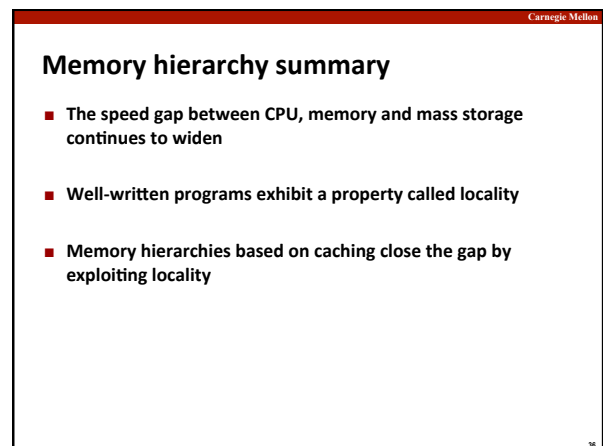


Carnegie Mellon

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

35



Today

- DRAM as building block for main memory
- Locality of reference
- Caching in the memory hierarchy
- Storage technologies and trends

37

What's Inside A Disk Drive?

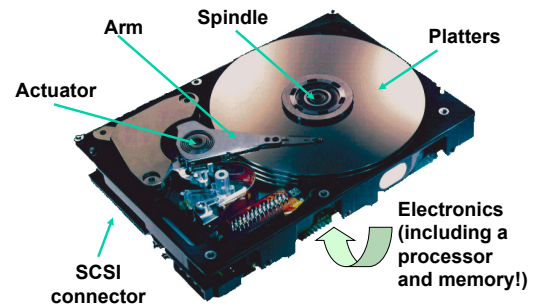
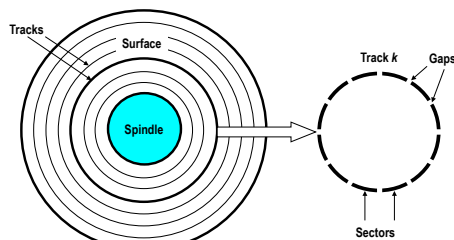


Image courtesy of Seagate Technology

38

Disk Geometry

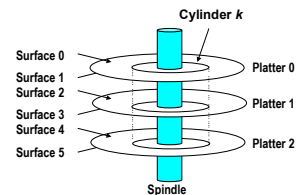
- Disks consist of **platters**, each with two **surfaces**.
- Each surface consists of concentric rings called **tracks**.
- Each track consists of **sectors** separated by **gaps**.



39

Disk Geometry (Multiple-Platter View)

- Aligned tracks form a cylinder.



40

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/in²): 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

41

Computing Disk Capacity

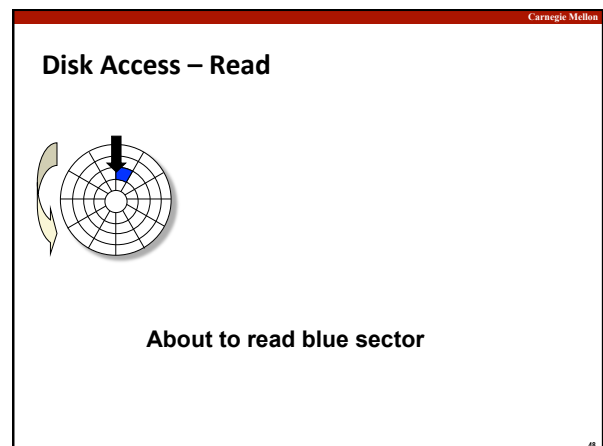
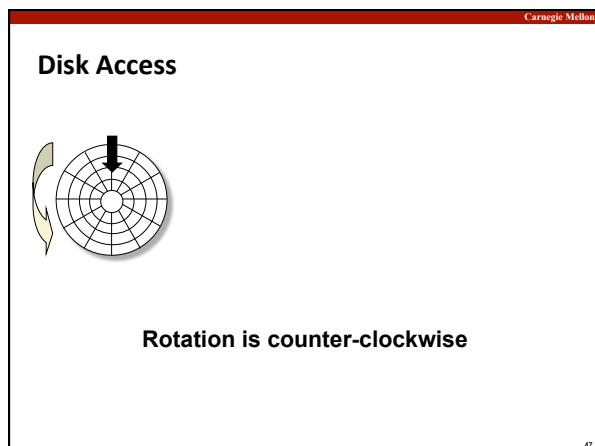
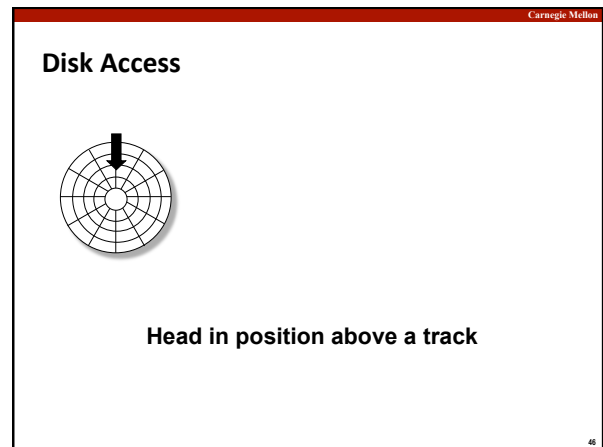
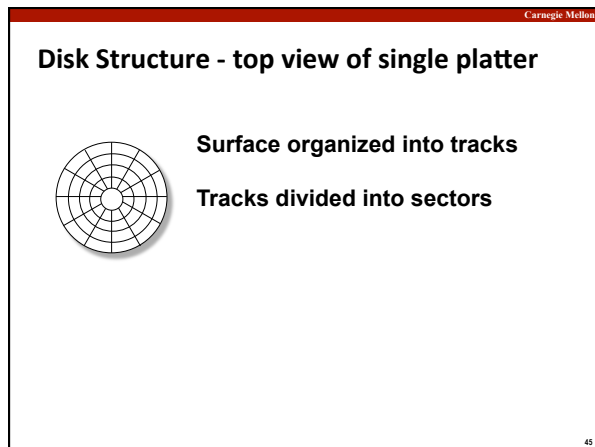
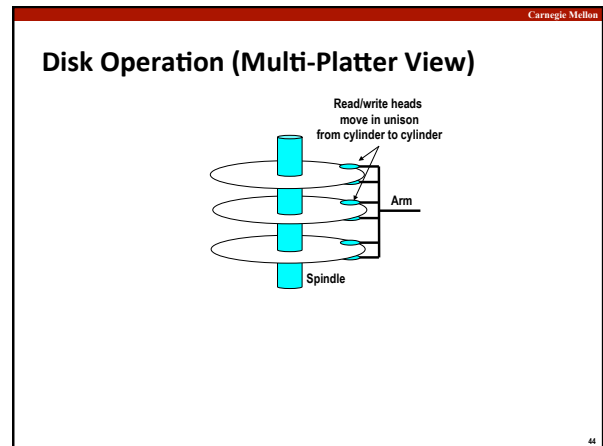
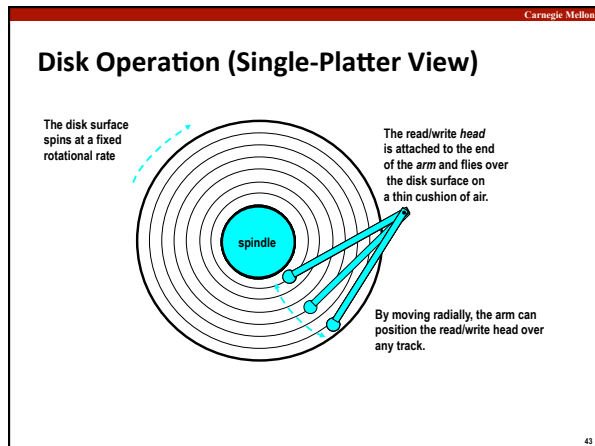
$$\text{Capacity} = (\# \text{ bytes/sector}) \times (\text{avg. } \# \text{ sectors/track}) \times (\# \text{ tracks/surface}) \times (\# \text{ surfaces/platter}) \times (\# \text{ platters/disk})$$

Example:

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

$$\begin{aligned} \text{Capacity} &= 512 \times 300 \times 20000 \times 2 \times 5 \\ &= 30,720,000,000 \\ &= 30.72 \text{ GB} \end{aligned}$$

42



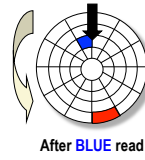
Disk Access – Read



After reading blue sector

49

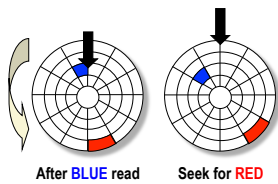
Disk Access – Read



Red request scheduled next

50

Disk Access – Seek

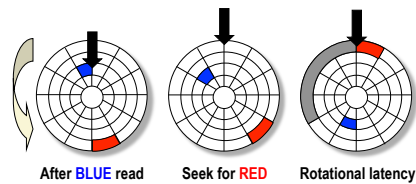


Seek for **RED**

Seek to red's track

51

Disk Access – Rotational Latency



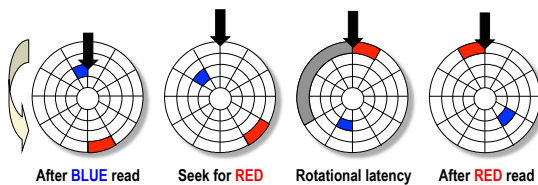
Seek for **RED**

Rotational latency

Wait for red sector to rotate around

52

Disk Access – Read



Seek for RED

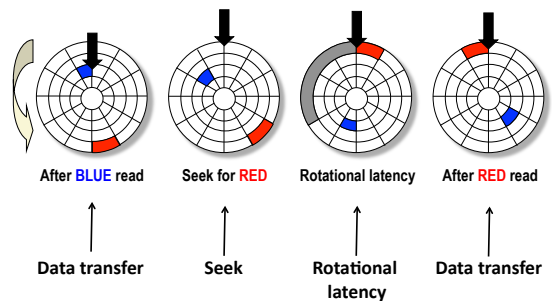
Rotational latency

After RED read

Complete read of red

53

Disk Access – Service Time Components



Seek for RED

Rotational latency

After **RED** read

Data transfer

Seek

Rotationa
latency

Data transfer

54

Disk Access Time

- **Average time to access some target sector approximated by :**
 - $T_{\text{access}} = T_{\text{avg seek}} + T_{\text{avg rotation}} + T_{\text{avg transfer}}$
- **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.
 - $T_{\text{avg rotation}} = 1/2 \times 1/\text{RPMs} \times 60 \text{ sec}/1 \text{ min}$
 - Typical Tavg rotation = 7200 RPMs
- **Transfer time (Tavg transfer)**
 - Time to read the bits in the target sector.
 - $T_{\text{avg transfer}} = 1/\text{RPM} \times 1/(\text{avg \# sectors/track}) \times 60 \text{ secs}/1 \text{ min}$.

55

Disk Access Time Example

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

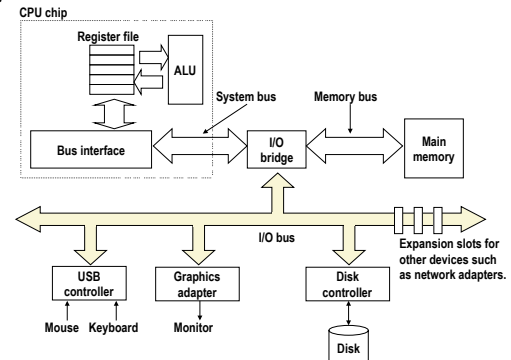
56

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”.

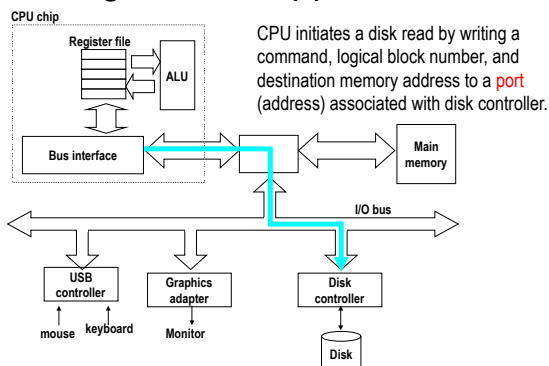
57

I/O Bus



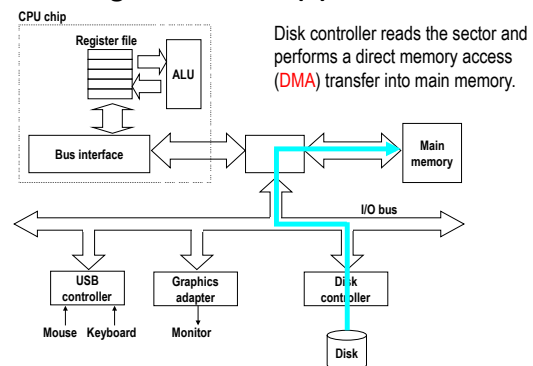
58

Reading a Disk Sector (1)

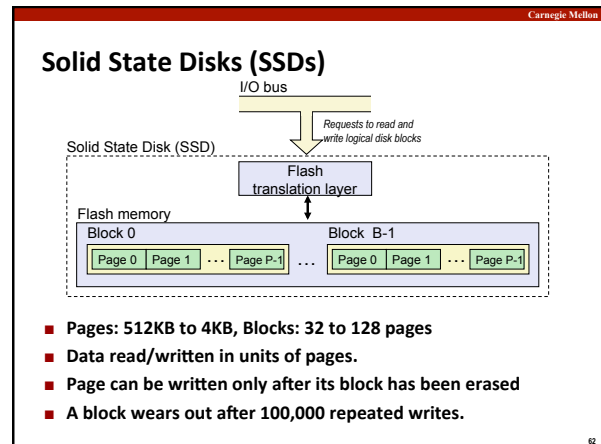
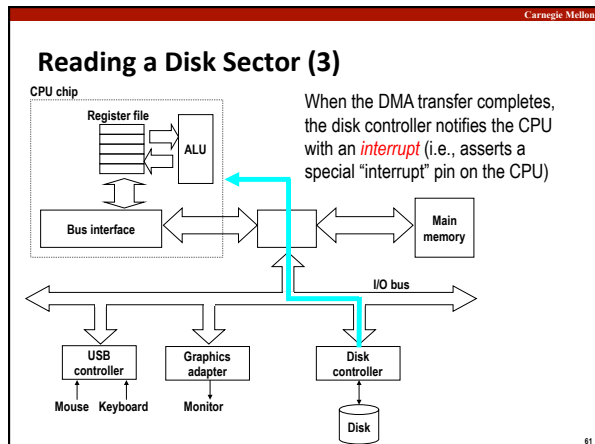


59

Reading a Disk Sector (2)



60



Carnegie Mellon

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)
 - Write to a page triggers a copy of all useful pages in the block
 - Find an used block (new block) and erase it
 - Write the page into the new block
 - Copy other pages from old block to the new block

63

Carnegie Mellon

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

64

Carnegie Mellon

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

65

Carnegie Mellon

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

66

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.

67

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

68

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

69

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[k][i][j];

    return sum;
}
```

70