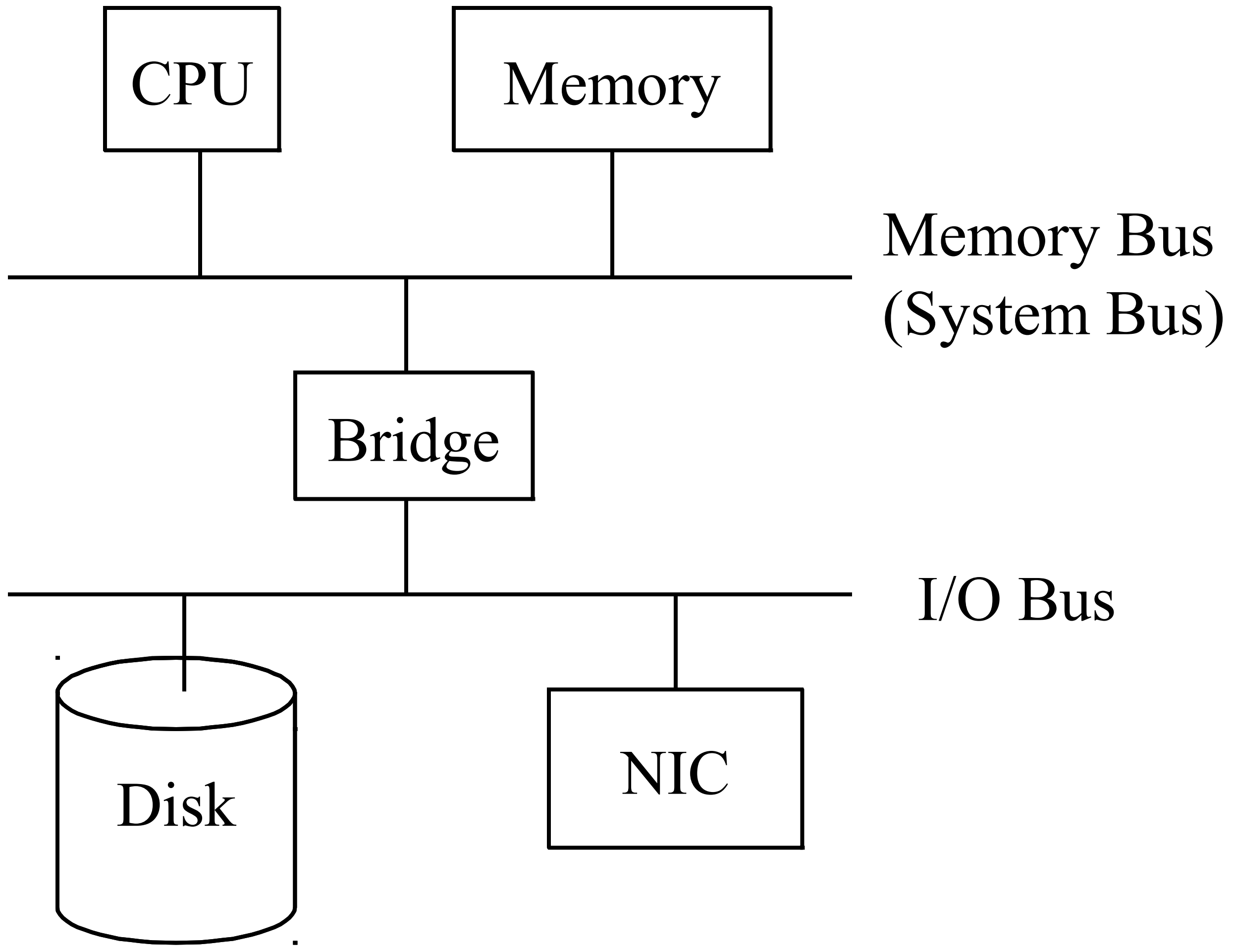
Disks

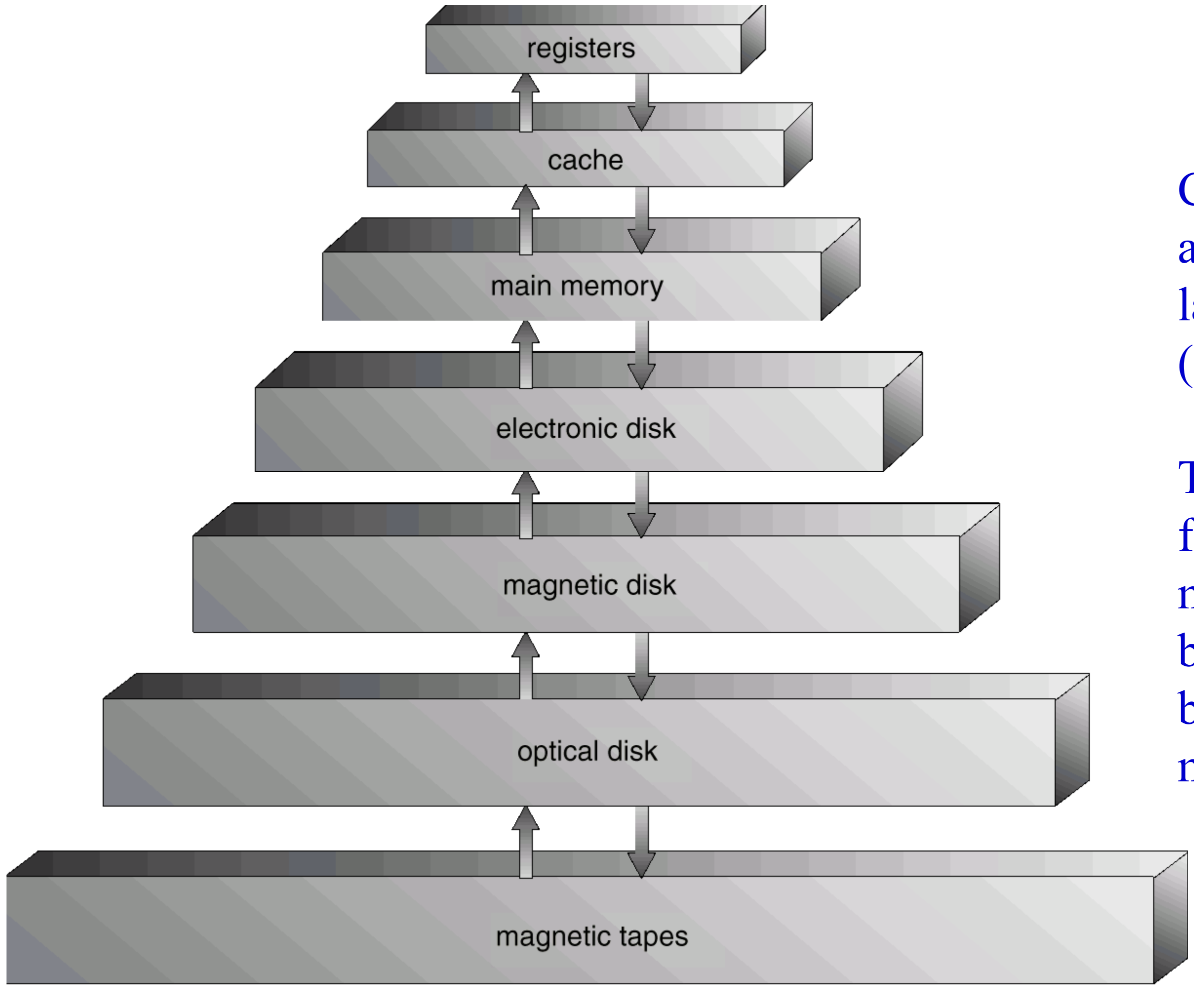
Operating Systems

Department of Computer Science
Rutgers University

File System: Abstraction for Secondary Storage



Storage-Device Hierarchy



Current magnetic disks are actually substantially larger than optical disks (e.g., DVD, BlueRay)

Tapes are still heavily used for backup and archival of massive amounts of data, but have progressively been losing ground against magnetic disks

Secondary Storage

Secondary storage typically:

- Is storage outside of memory
- Does not permit direct execution of instructions or data retrieval via load/store instructions

Characteristics:

- It’s large: 0.1 TB – 4 TB (as of April 2013)
- It’s cheap: 1 TB 7200rpm SATA disks on order of $100
- It’s persistent: data is maintained across process execution and power down (or loss)
- It’s slow: milliseconds to access

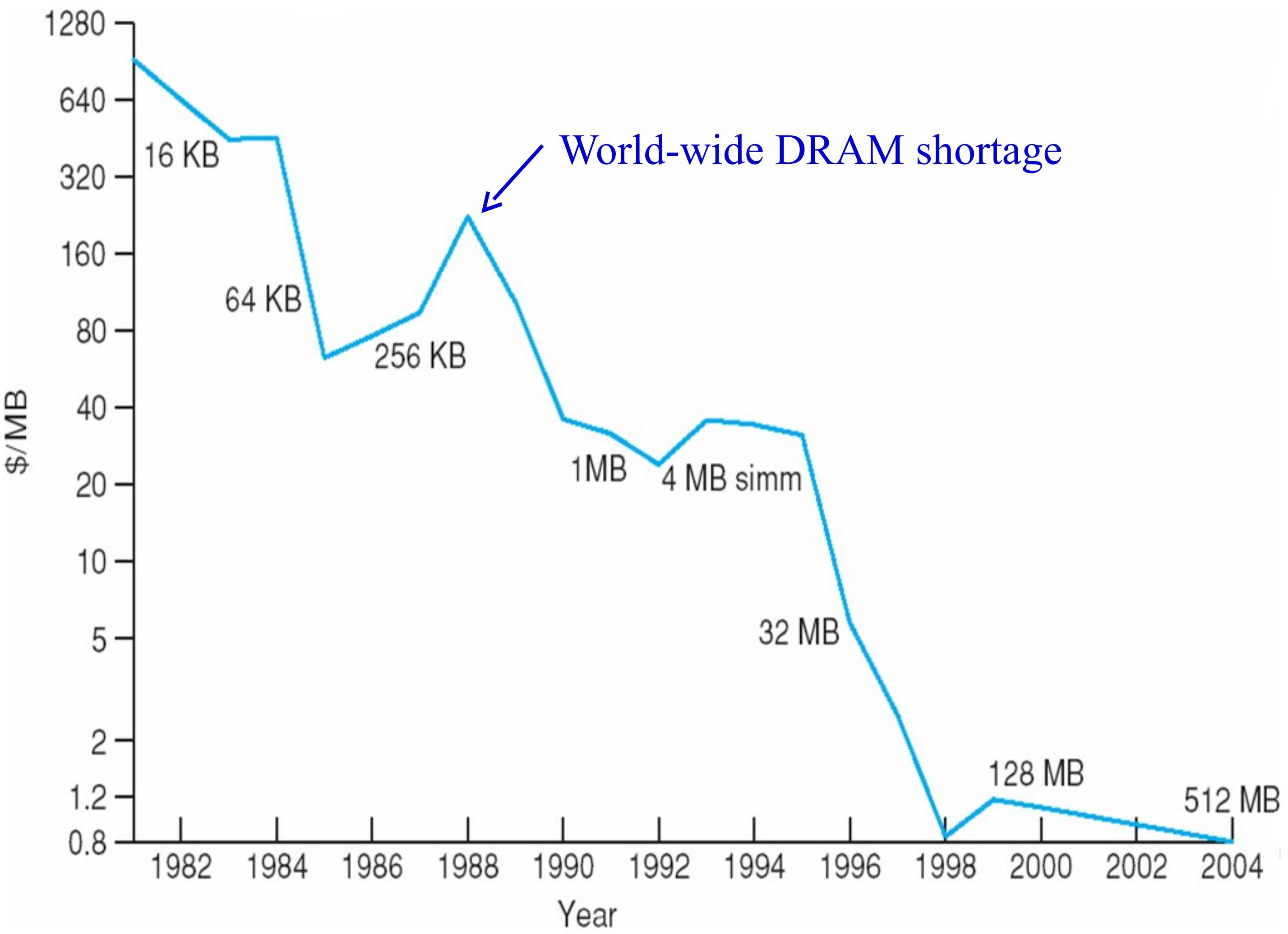
Costs

Main memory is much more expensive than disk storage

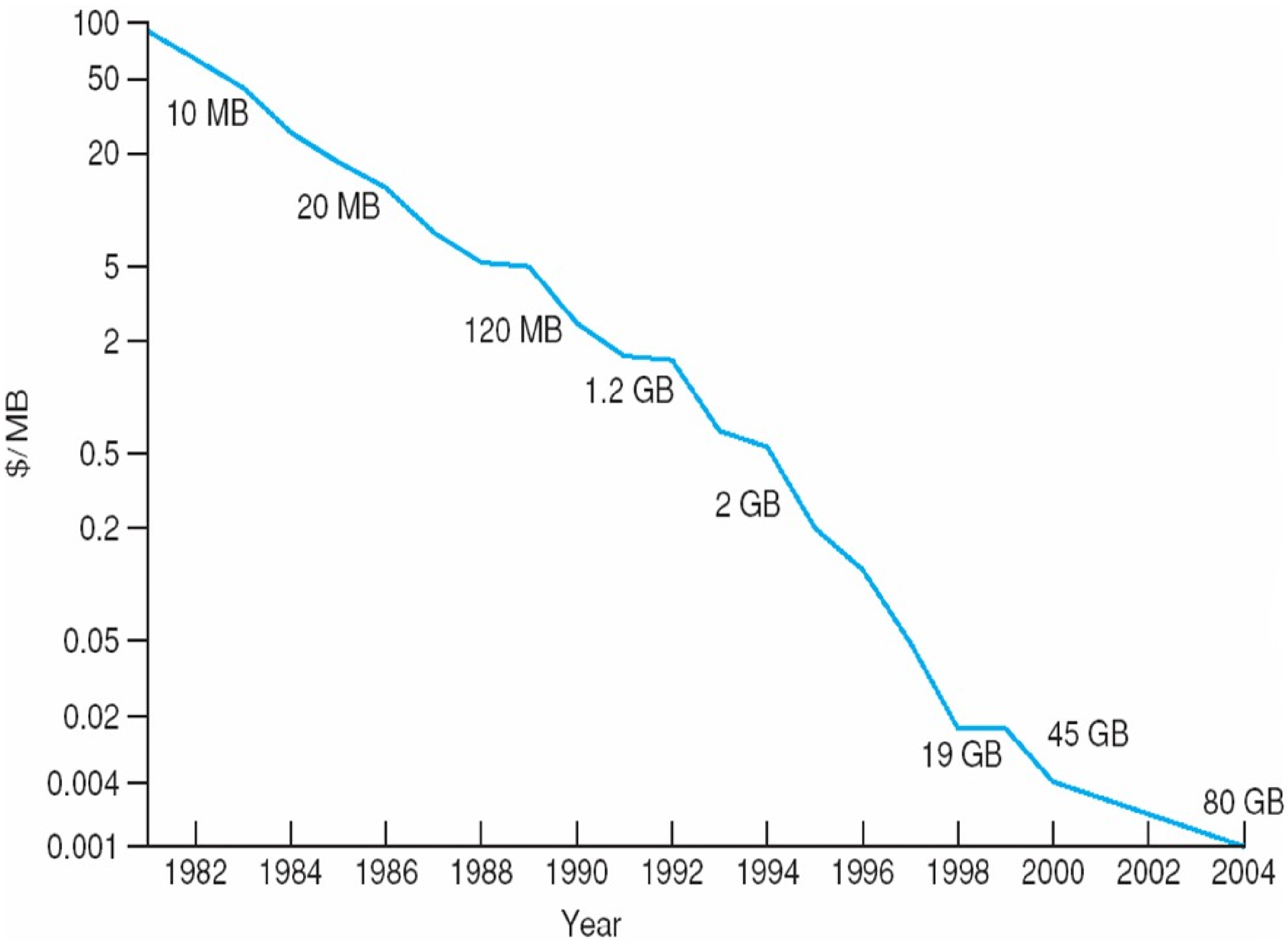
The cost/MB of hard disk storage is competitive with magnetic tape if only one tape is used per drive

The cheapest tape drives and the cheapest disk drives have had about the same storage capacity over the years

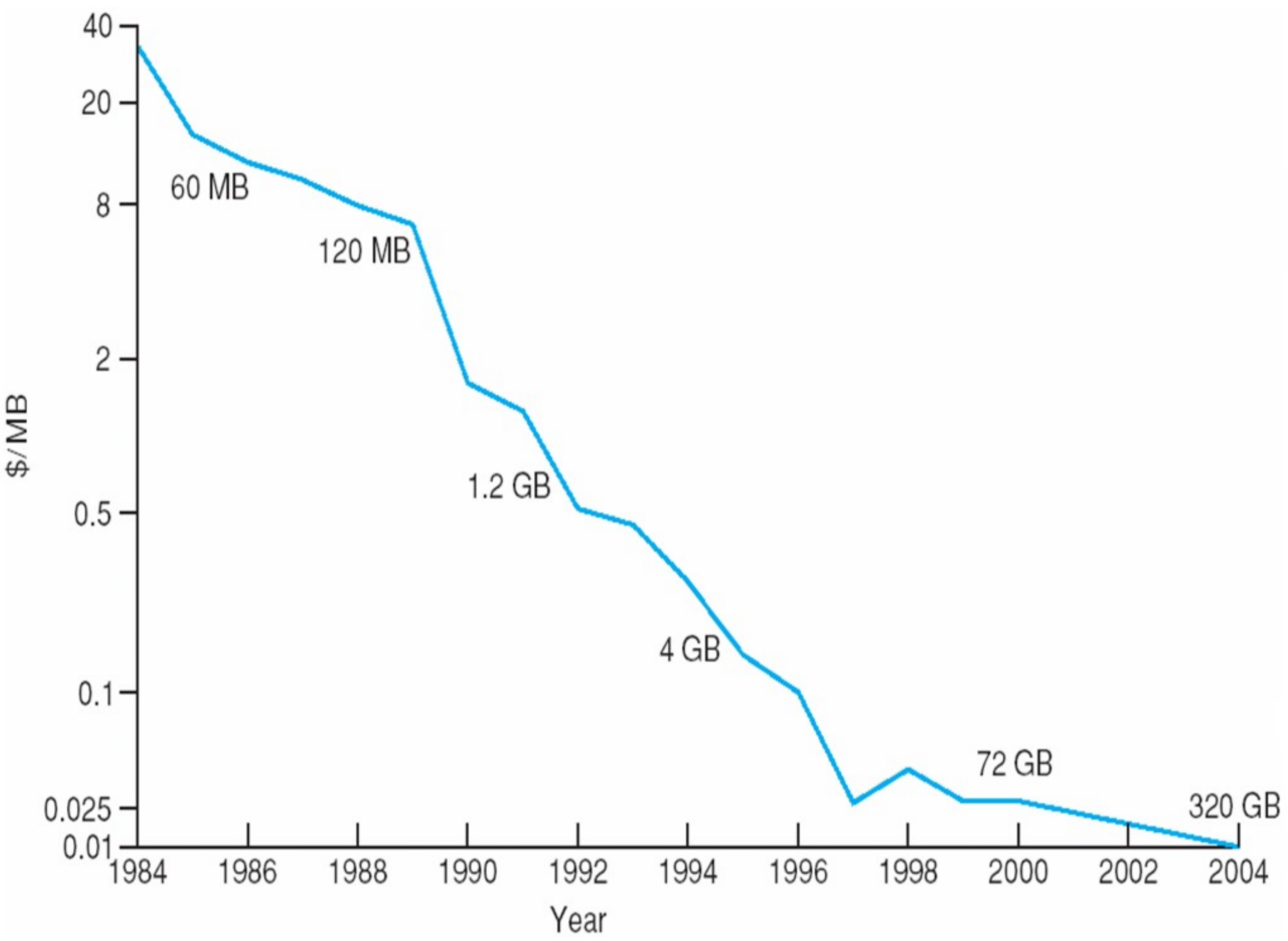
Cost of DRAM


World-wide DRAM shortage
$/MB
1280
640
320
160
80
40
20
10
5
2
1.2
0.8
16 KB
64 KB
256 KB
1MB
4 MB simm
32 MB
128 MB
512 MB
1982
1984
1986
1988
1990
1992
1994
1996
1998
2000
2002
2004
Year

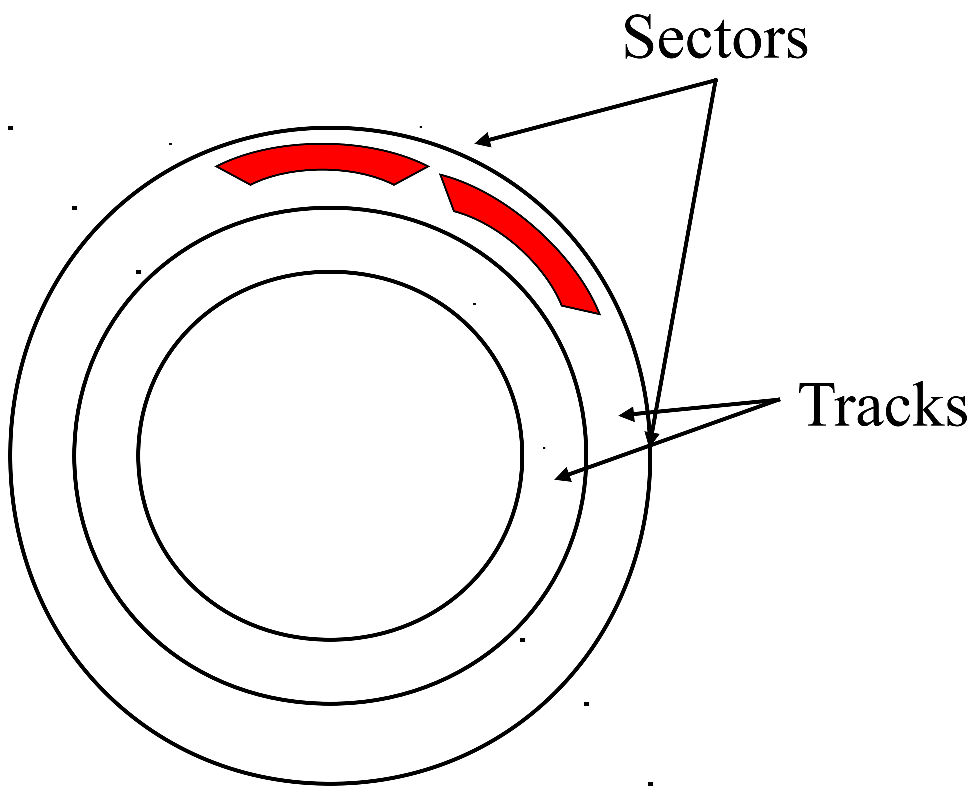
Cost of Disks



Cost of Tapes



Disks



Seek time: time to move the disk head to the desired track

Rotational delay: time to reach desired sector once head is over the desired track

Transfer rate: rate data read from/written to disk

Some typical parameters:

- Seek: ~2-10ms
- Rotational delay: ~3ms for 10000 rpm
- Transfer rate: 200 MB/s

Disk Scheduling

- Disks are at least 4 orders of magnitude slower than memory
  - The performance of disk I/O is vital for the performance of the computer system as a whole
  - Access time (seek time + rotational delay) >> transfer time for a sector
  - Therefore the order in which sectors are accessed (read, especially) matters a lot
- Disk scheduling
  - Usually based on the position of the requested sector rather than according to the process priority
  - Possibly reorder stream of read/write requests to improve performance

Disk Scheduling (Cont.)

Several algorithms exist to schedule the servicing of disk I/O requests.

We illustrate them with a request queue (tracks 0-199).

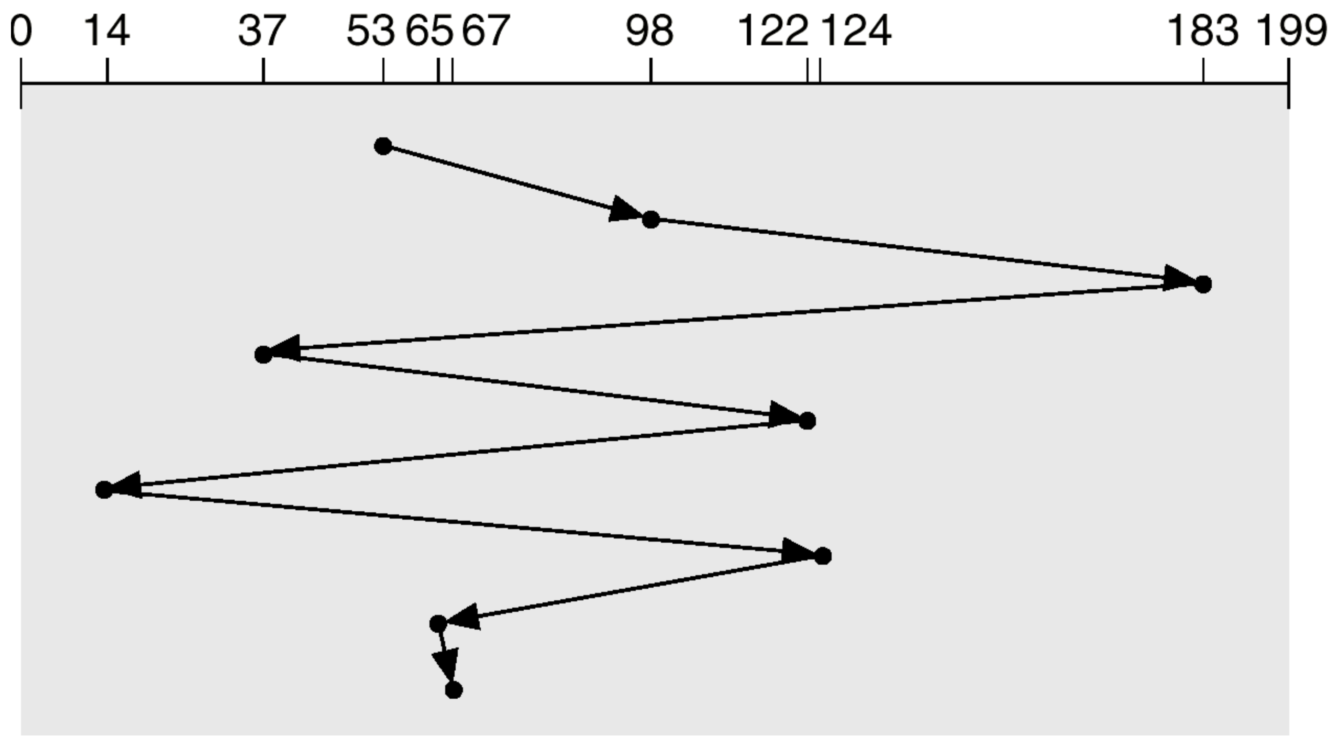
98, 183, 37, 122, 14, 124, 65, 67

Head pointer 53

FCFS

Illustration shows total head movement of 640 cylinders.

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



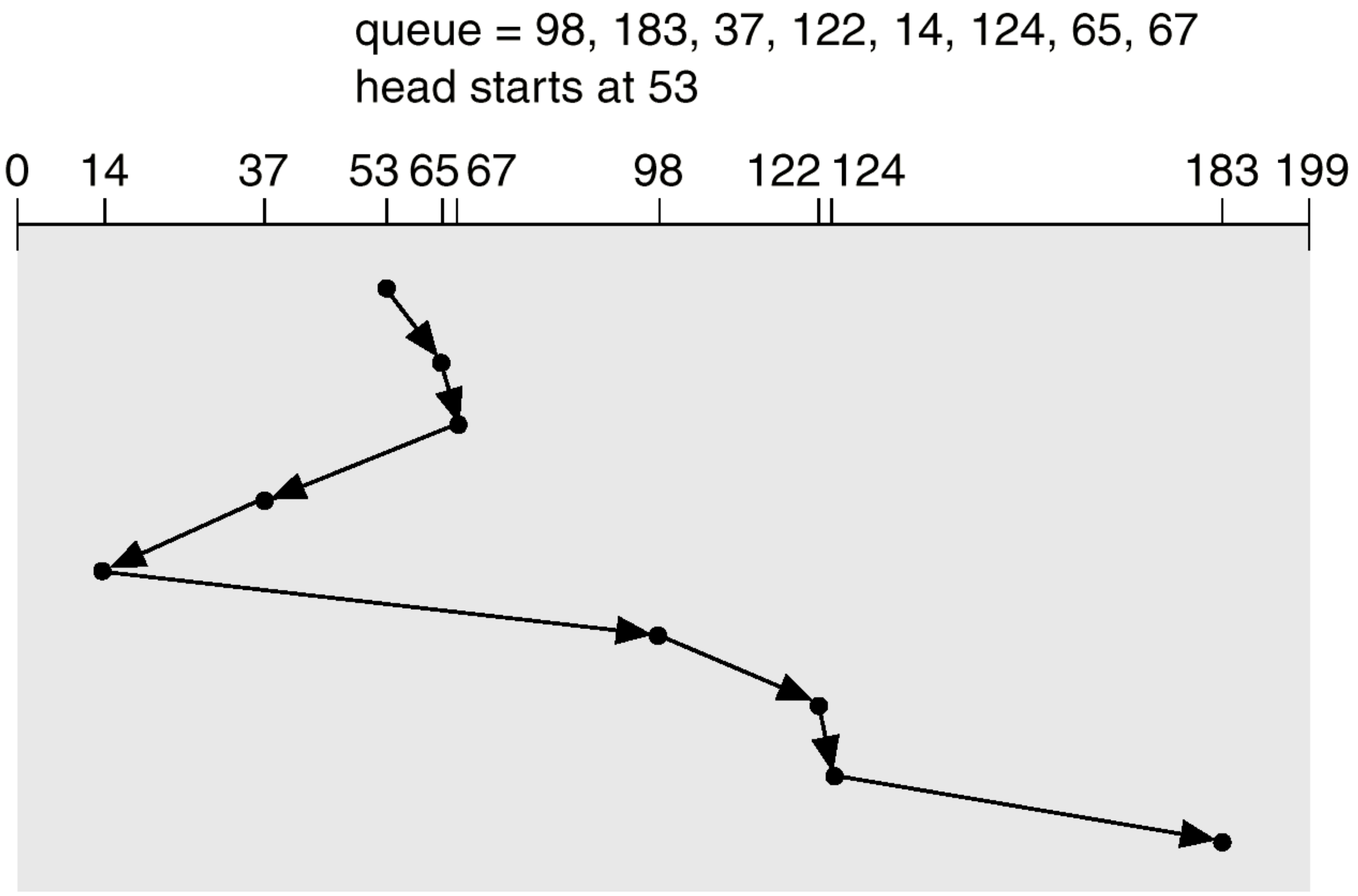
SSTF

Selects the request with the minimum seek time from the current head position.

SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.

Illustration shows total head movement of 236 cylinders.

SSTF (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

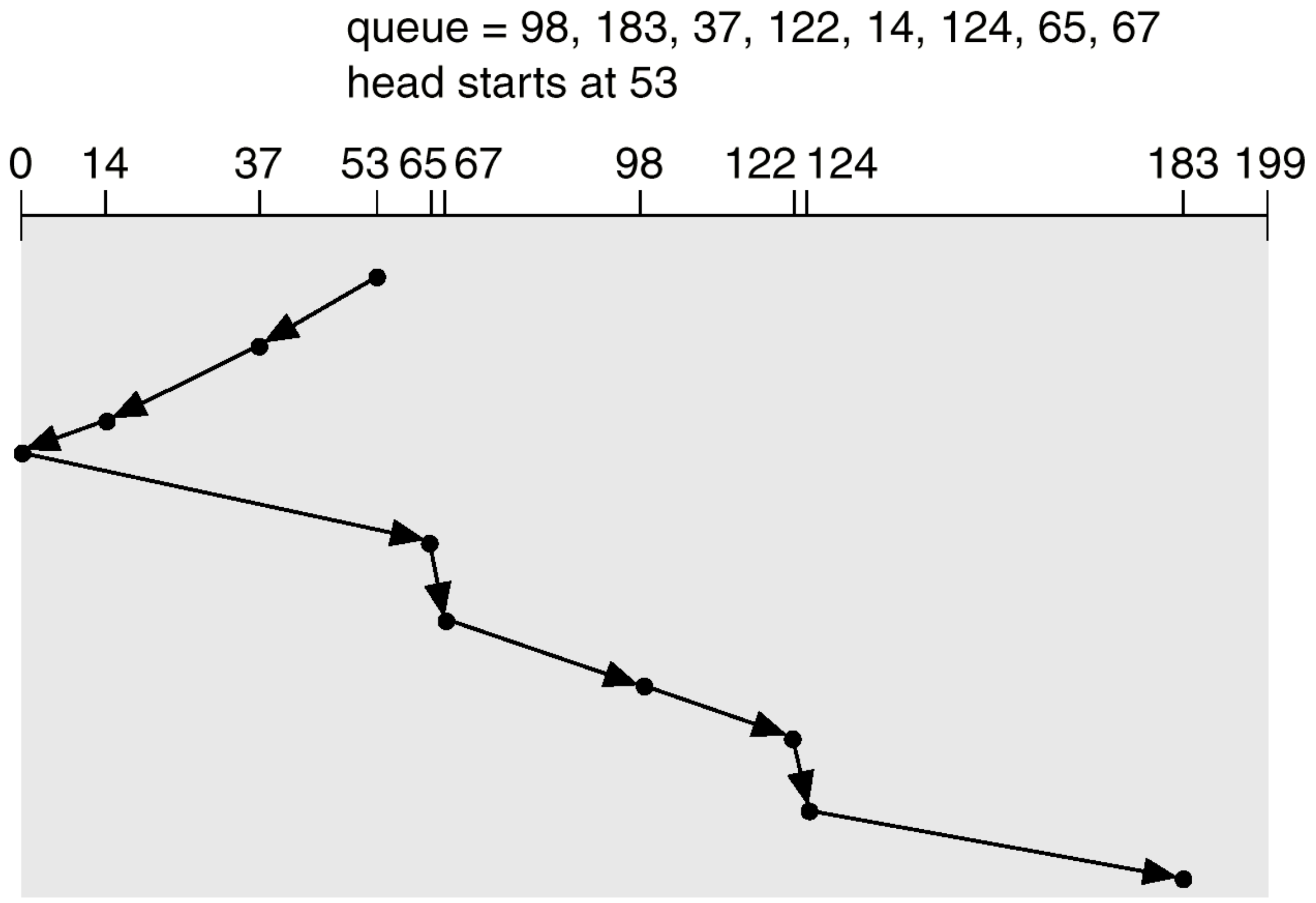
SCAN

The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.

Sometimes called the *elevator algorithm*.

Illustration shows total head movement of 208 cylinders.

SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

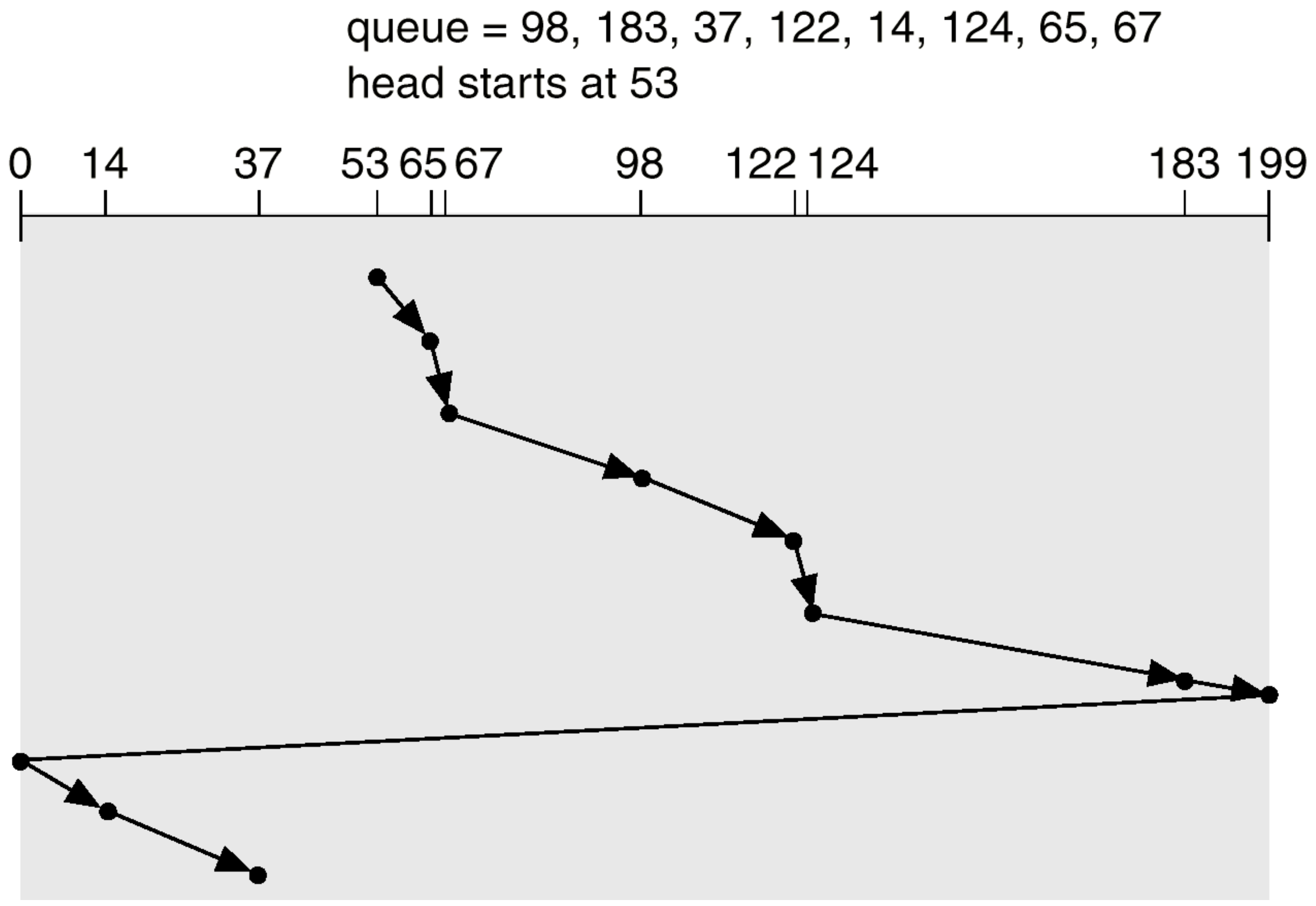
C-SCAN

Provides a more uniform wait time than SCAN.

The head moves from one end of the disk to the other, servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.

Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.

C-SCAN (Cont.)

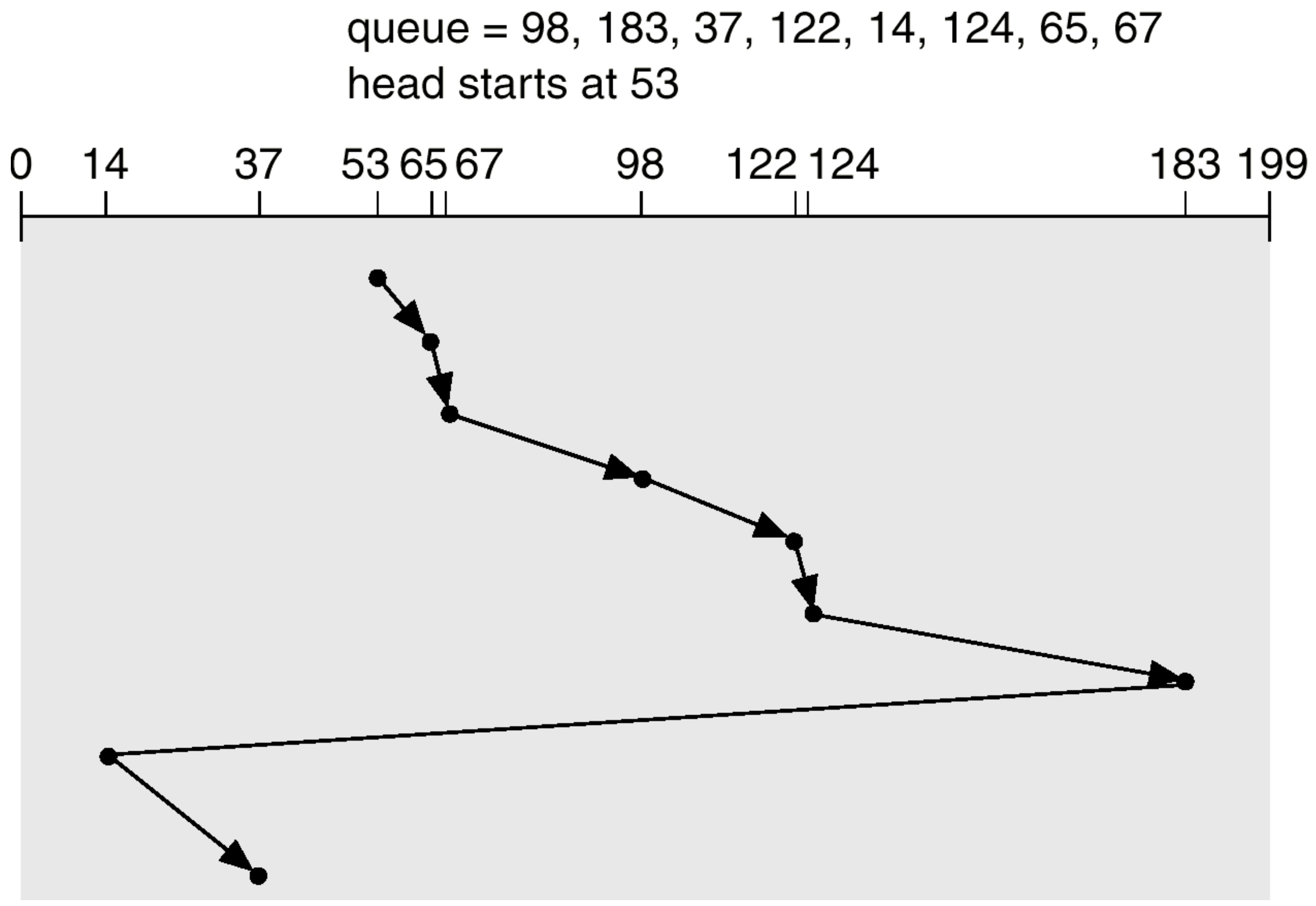

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

C-LOOK

Version of C-SCAN

Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.

C-LOOK (Cont.)



queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

Selecting a Disk Scheduling Policy

SSTF is common and has a natural appeal. SCAN has better service distribution than SSTF. C-SCAN has lower service variability than SCAN.

SCAN and C-SCAN perform better for systems that place a heavy load on the disk.

Performance depends on the number and types of requests.

Requests can be influenced by the file-allocation method.

Scheduling algorithm should be a separate module of the OS, allowing it to be replaced with a different algorithm if necessary.

Either SSTF or LOOK is a reasonable choice for the default algorithm.

Disk Management

Low-level formatting or *physical formatting* — Dividing a disk into sectors that the disk controller can read and write.

To use a disk to hold files, the operating system still needs to record its own data structures on the disk.

- *Partition* the disk into one or more groups of cylinders.
- *Logical formatting* or "making a file system".

Boot block initializes system.

- The bootstrap is stored in ROM.
- *Bootstrap loader* program.

Methods such as *sector sparing* used to handle bad blocks.

Swap Space Management

Virtual memory uses disk space as an extension of main memory.

Swap space is necessary for pages that have been written and then replaced from memory.

Swap space can be carved out of the normal file system, or, more commonly, it can be in a separate disk partition.

Swap space management

- 4.3BSD allocates swap space when process starts; swap space holds *text segment* (the program) and *data segment.* (Stack and heap pages are created in main memory first.)
- Kernel uses *swap maps* to track swap space use.
- Solaris 2 allocates swap space only when a page is forced out of physical memory, not when the virtual memory page is first created.

Disk Reliability

Several improvements in disk use techniques involve the use of multiple disks working cooperatively.

RAID is one important technique currently in common use.

RAID

Redundant Array of Inexpensive Disks (RAID)

- A set of physical disk drives viewed by the OS as a single logical drive
- Replace large-capacity disks with multiple smaller capacity drives to improve the I/O performance (at lower price)
- Data are distributed across physical drives in a way that enables simultaneous access to data from multiple drives
- Redundant disk capacity is used to compensate for the increase in the probability of failure due to multiple drives
  - Improve availability because no single point of failure

Six levels of RAID representing different design alternatives

RAID Level 0

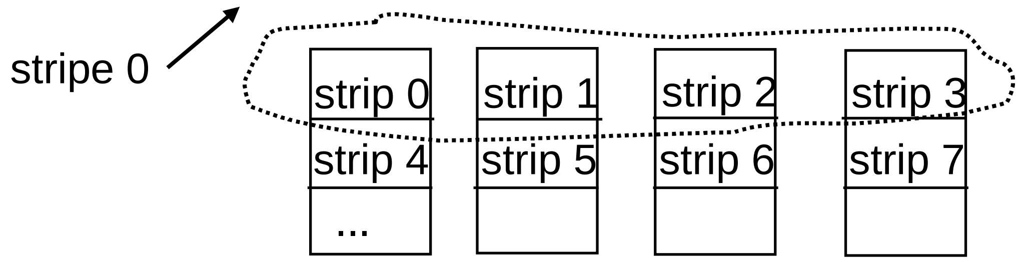
Does not include redundancy

Data are striped across the available disks

- Total storage space across all disks is divided into strips
- Strips are mapped round-robin to consecutive disks
- A set of consecutive strips that map exactly one strip to each disk in the array is called a stripe

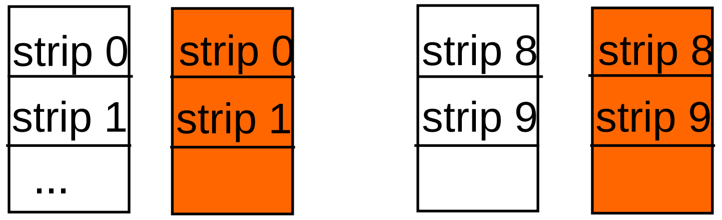
Can you see how this improves the disk I/O bandwidth?

What access pattern gives the best performance?



RAID Level 1

- Redundancy achieved by duplicating all the data
- Every disk has a mirror disk that stores exactly the same data
  - A read can be serviced by either of the two disks which contains the requested data (improved performance over RAID 0 if reads dominate)
  - A write request must be done on both disks but can be done in parallel
  - Recovery is simple but cost is high



RAID Levels 2 and 3

Parallel access: all disks participate in every I/O request

Small strips (1 bit) since size of each read/write = # of disks * strip size

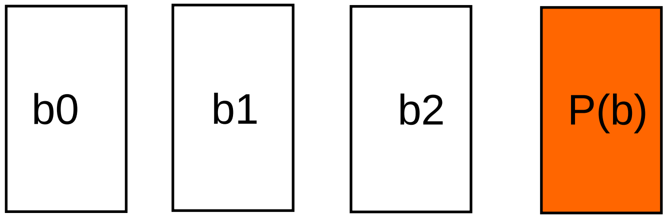
RAID 2: 1-bit strips and error-correcting code. ECC is calculated across corresponding bits on data disks and stored on O(log(# data disks)) ECC disks

- Hamming code: can correct single-bit errors and detect double-bit errors
- Example configurations data disks/ECC disks: 4/3, 10/4, 32/7
- Less expensive than RAID 1 but still high overhead – not needed in most environments

RAID 3: 1-bit strips and a single redundant disk for parity bits

- $P(i) = X2(i) \oplus X1(i) \oplus X0(i)$

On a failure, data can be reconstructed. Only tolerates one failure at a time



$X2(i) = P(i) \oplus X1(i) \oplus X0(i)$

RAID Levels 4 and 5

RAID 4

Large strips with a parity strip like RAID 3

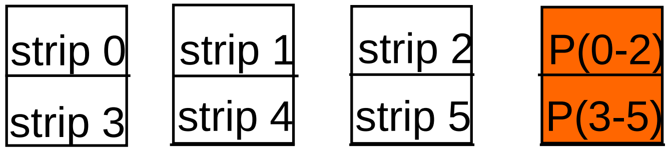
Independent access - each disk operates independently, so multiple I/O request can be satisfied in parallel

Independent access → small write = 2 reads + 2 writes

Example: if write performed only on strip 0:

$$
\begin{aligned}
P'(i) &= X2(i) \oplus X1(i) \oplus X0'(i) \\
&= X2(i) \oplus X1(i) \oplus X0'(i) \oplus X0(i) \oplus X0(i) \\
&= P(i) \oplus X0'(i) \oplus X0(i)
\end{aligned}
$$

Parity disk can become bottleneck

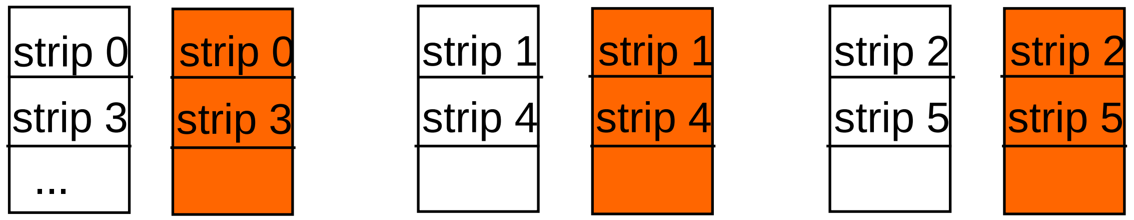


RAID 5

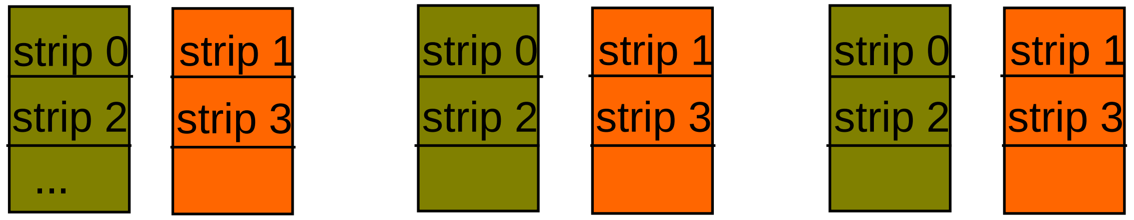
Like RAID 4 but parity strips are distributed across all disks

Other Popular RAID Organizations

RAID 0 + 1: Stripe first and then mirror

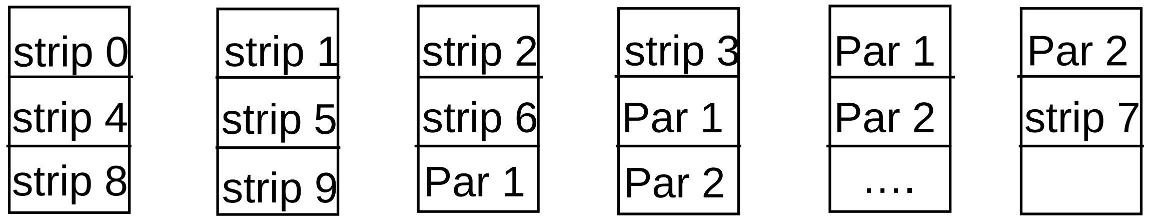


RAID 1 + 0: Mirror first and then stripe



Other Popular RAID Organizations

RAID 6: Two parity blocks for each stripe tolerate any two faults



Parity 1 is the XOR of the data strips

Parity 2 involves first shifting each strip left and then using XOR (this idea comes from Galois Field theory)