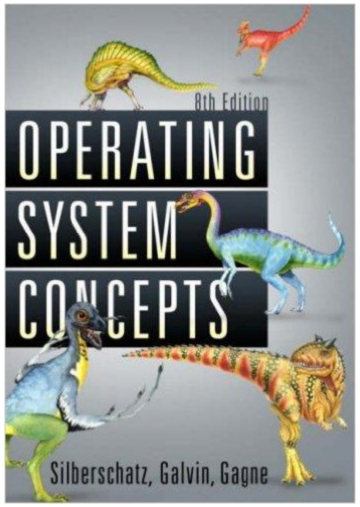
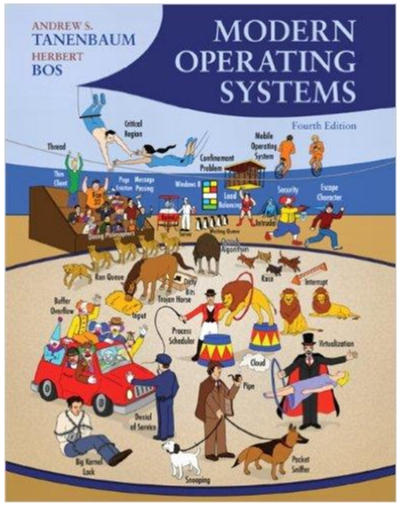
Disks, RAIDs, and Stable Storage





CS-3023 Operating Systems

Worcester Polytechnic Institute

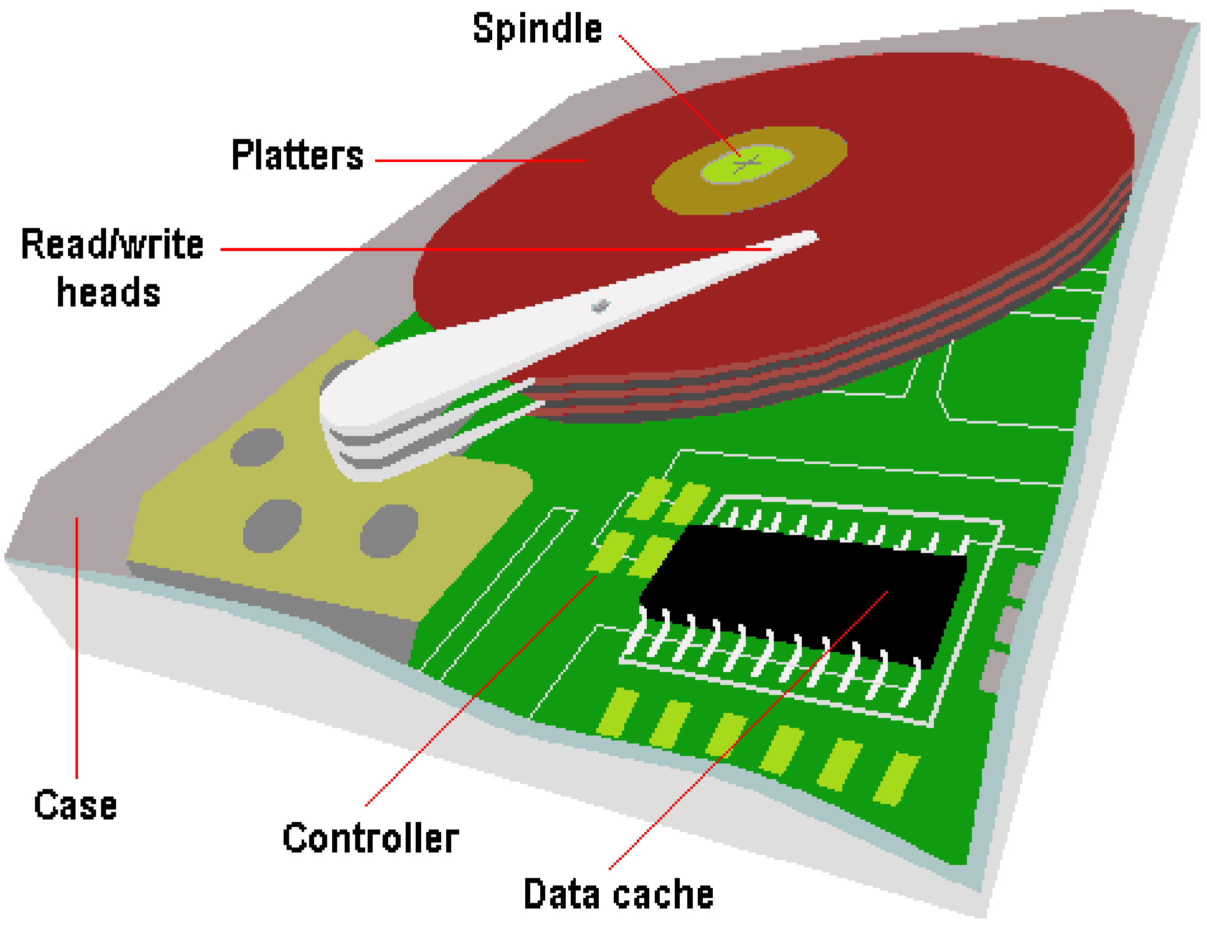
Disks

Context

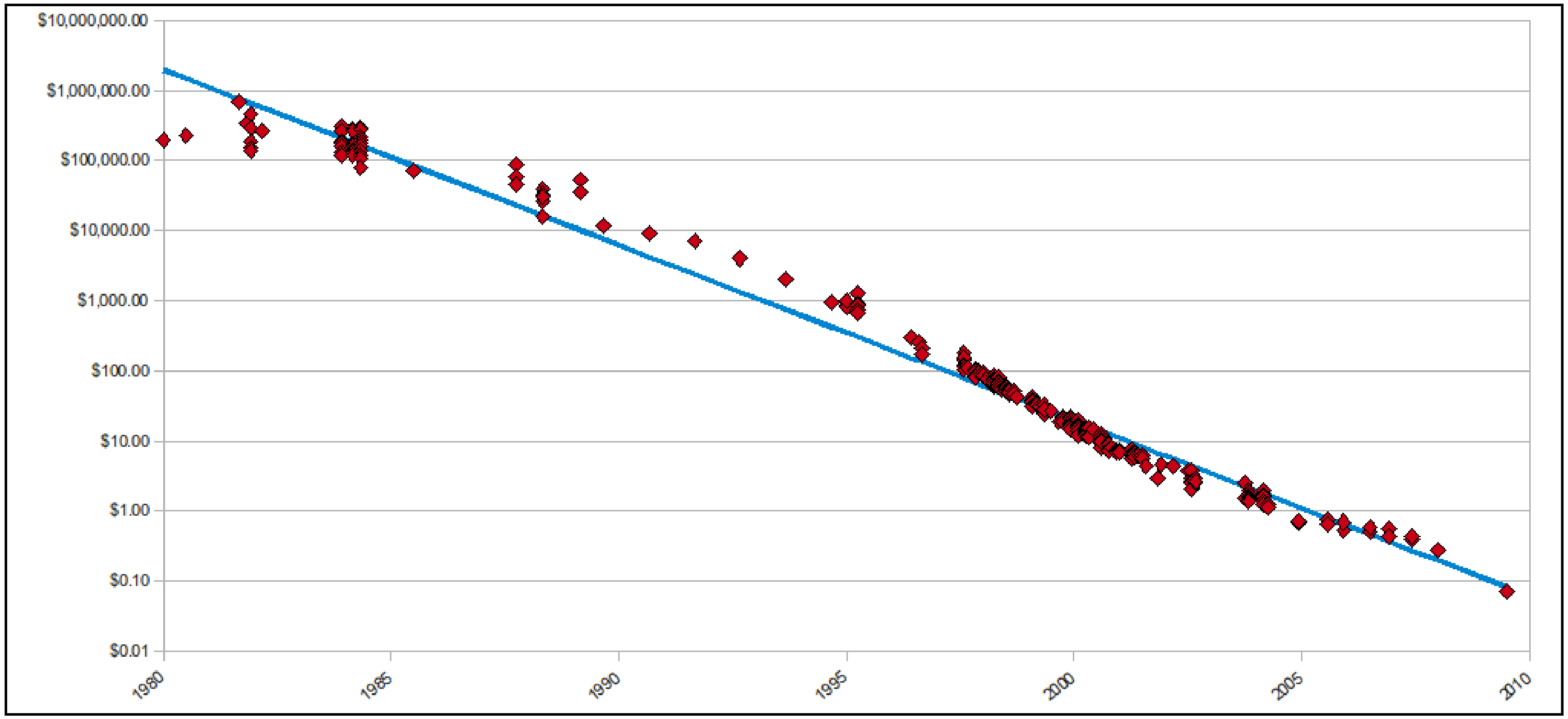
- Early days: disks thought of as I/O devices
    - Controlled like I/O devices
    - Block transfer, DMA, interrupts, etc.
    - Data *in* and *out* of memory (where action is)
- Today: disks as integral part of computing system
    - Implementer of two fundamental abstractions
        - *Virtual Memory*
        - *Files*
    - Long term storage of information *within* system
    - The real center of action

Disk Drives

- External Connection
  - IDE/ATA
  - SCSI
  - USB
- Cache – independent of OS
- Controller
  - Details of read/write
  - Cache management
  - Failure management



Price per Gigabyte of Magnetic Hard Disk From 1980 to 2009



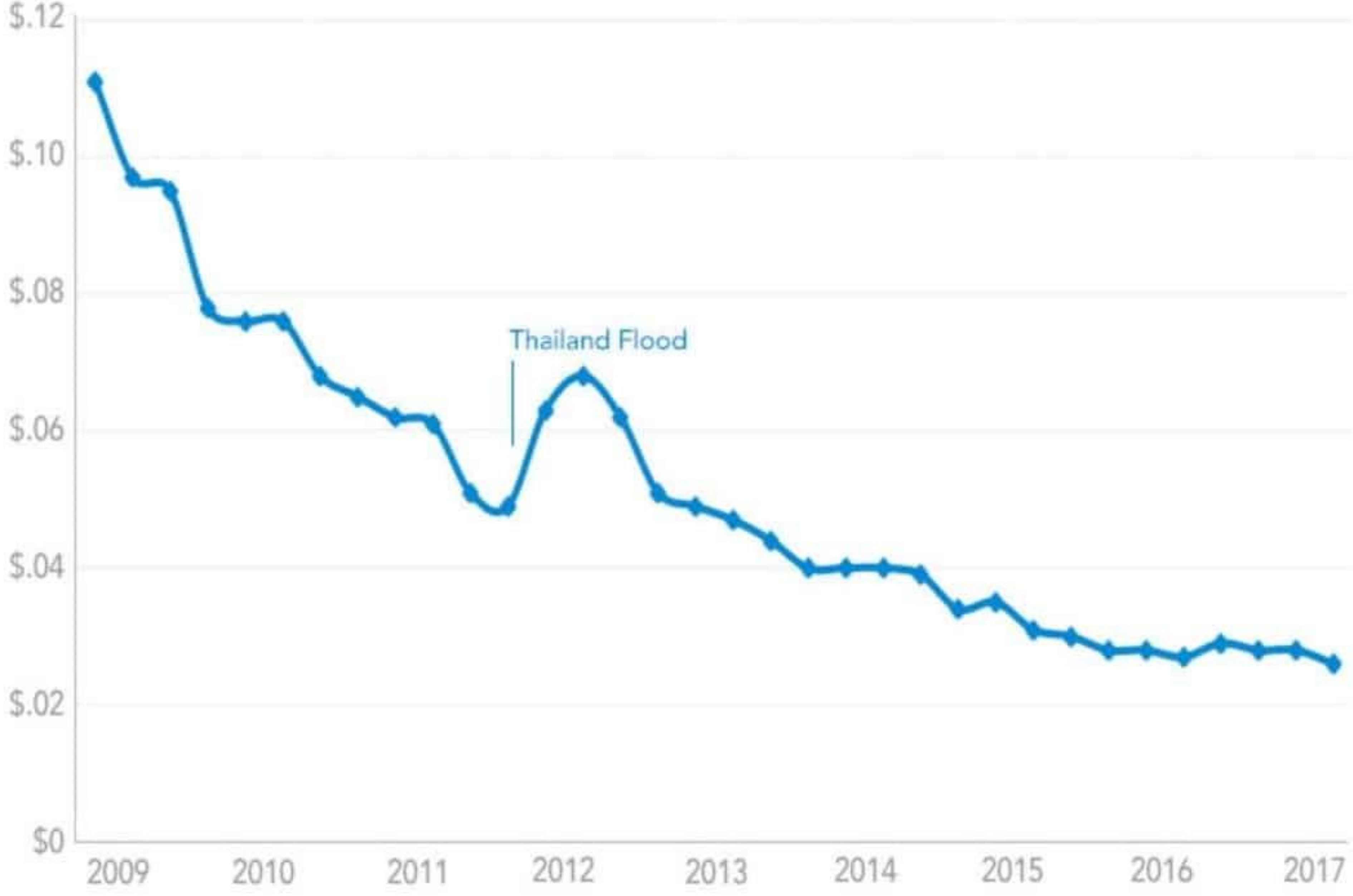
http://www.mkomo.com/cost-per-gigabyte

Prices per GB (March 9, 2006)

- 52¢ per gigabyte – 250 GB Porsche (portable)
  - 7200 rpm, 11 ms. avg. seek time, 2 MB drive cache
  - USB 2.0 port (effective 40 MBytes/sec)
- $1.25 per GB – 40 GB Barracuda
  - 7200 rpm, 8.5 ms. ms. avg. seek time, 2 MB drive cache
  - EIDE (theoretical 66-100 MBytes/sec)
- $4.52 per GB – 72 GB Hot-swap
  - 10,000 rpm, 4.9 ms. avg. seek time
  - SCSI (320 MB/sec)
- $6.10 per GB – 72 GB Ultra
  - 15,000 rpm, 3.8 ms. avg. seek time
  - SCSI (320 MB/sec)

Prices per GB (February 14, 2008)

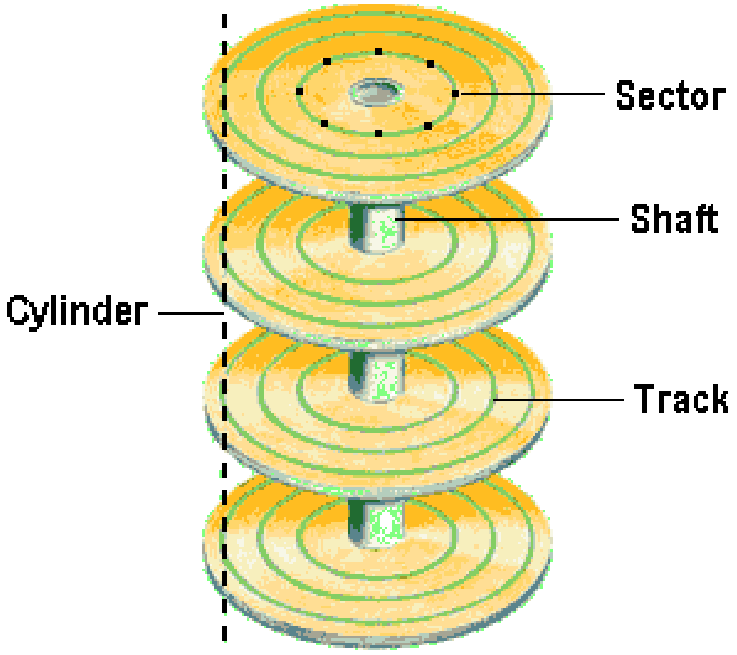
- 19¢ per gigabyte – 500 GB Quadra (portable)
  - 7200 rpm, 10 ms. avg. seek time, 16 MB drive cache
  - USB 2.0 port (effective 40 MBytes/sec)
- 27.3¢ per GB – 1 TB Caviar (internal)
  - 5400-7200 rpm, 8.9 ms. avg. seek time , 16 MB drive cache
  - ATA
- 62¢ per GB – 80 GB Caviar (internal)
  - 7200 rpm, 8.9 ms. ms. avg. seek time, 8 MB drive cache
  - EIDE (theoretical 66-100 MBytes/sec)
- $2.60 per GB – 146.8 GB HP SAS (hot swap)
  - 15,000 rpm, 3.5 ms. avg. seek time
  - ATA



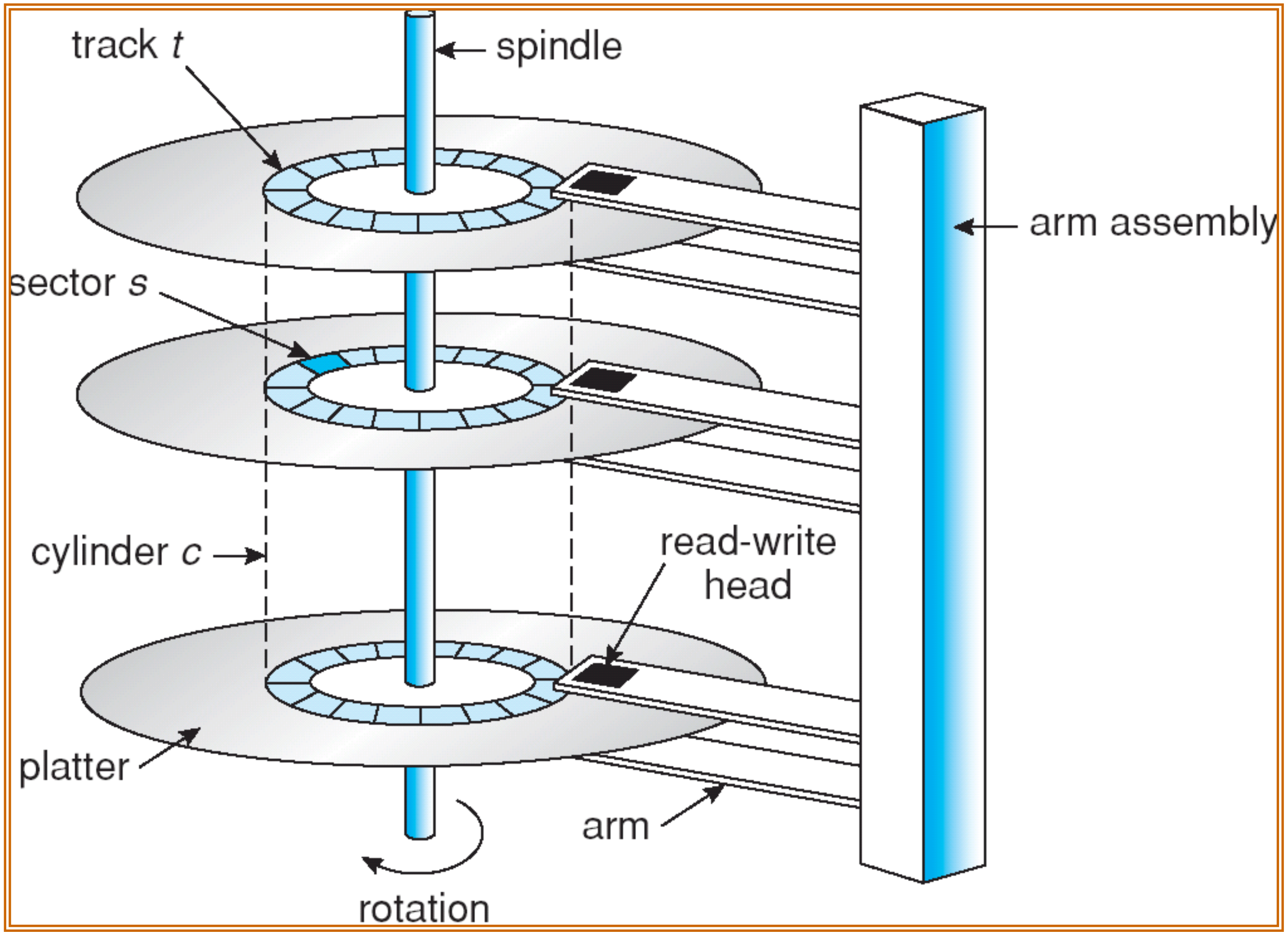
https://www.cloudwards.net/history-of-the-hard-drive/

Hard Disk Geometry

- Platters
  - Two-sided magnetic material
  - 1-16 per drive, 3,000 – 15,000 RPM
- Tracks
  - Concentric rings bits laid out serially
  - Divided into *sectors* (addressable)
- Cylinders
  - Same track on each platter
  - Arms move together
- Operation
  - *Seek:* move arm to track
  - *Read/Write:*
    - Wait till sector arrives under head
    - Transfer data



Moving-head Disk Mechanism



More on Hard Disk Drives

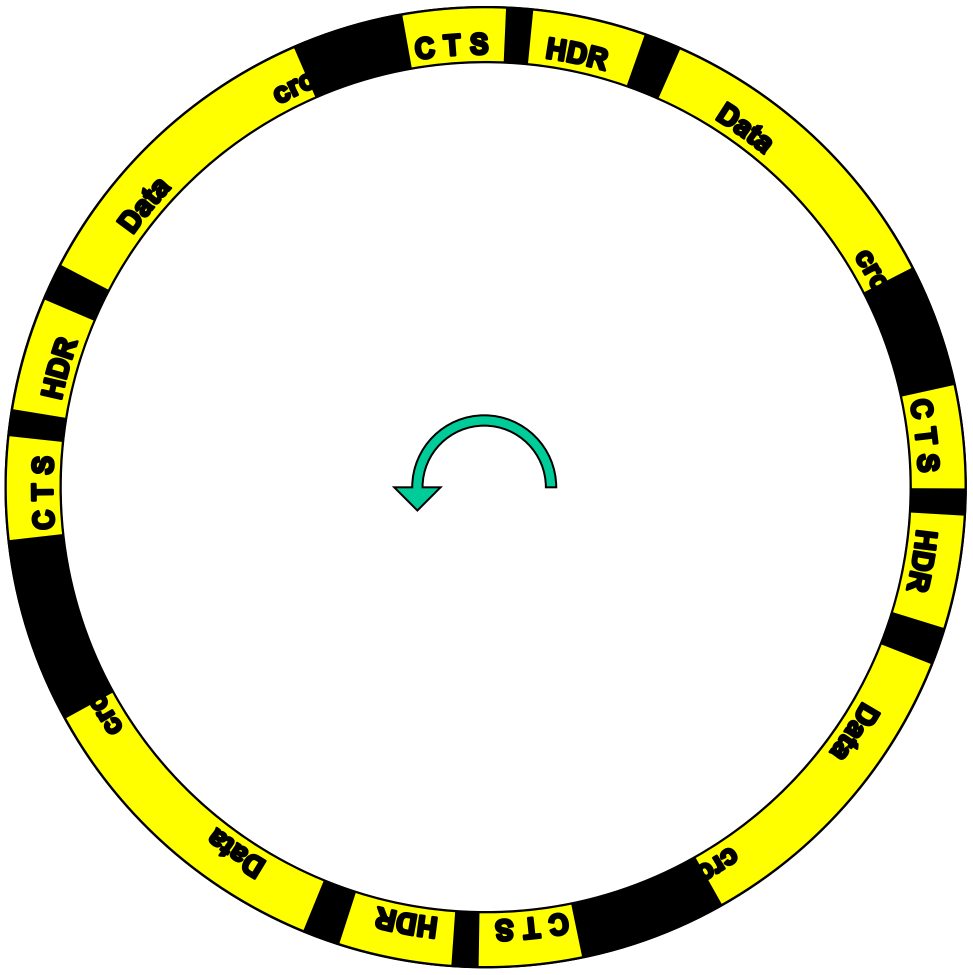
- Manufactured in clean room
- Permanent, air-tight enclosure
    - "Winchester" technology
    - Spindle motor integral with shaft
- "Flying heads"
    - Aerodynamically "float" over moving surface
    - Velocities > 100 m/s
    - Parking position for heads during power-off
- Excess capacity
    - Sector re-mapping for bad blocks
    - Managed by OS or by drive controller
- 20,000-100,000 hours mean time between failures
    - Disk failure (usually) means total destruction of data!

More on Hard Disk Drives (continued)

- Early days
  - Read/write platters in parallel for higher bandwidth
- Today
  - Extremely narrow tracks, closely spaced
    - tolerances < 5-20 microns
  - Thermal variations prevent precise alignment from one cylinder to the next
- Seek operation
  - Move arm to approximate position
  - Use feedback control for precise alignment
  - Seek time $\cong$ *k* * *distance*

Raw Disk Layout

- Track format – *n* sectors
  - 200 < *n* < 2000 in modern disks
  - Some disks have fewer sectors on inner tracks
- Inter-sector gap
  - Enables each sector to be read or written independently
- Sector format
  - Sector address: *C*ylinder, *T*rack, *S*ector (or some equivalent code)
  - <u>Optional</u> header (*HDR*)
  - Data
  - Each field separated by small gap and with its own CRC
- Sector length
  - Almost all operating systems specify uniform sector length
  - 512 – 4096 bytes



Formatting the Disk

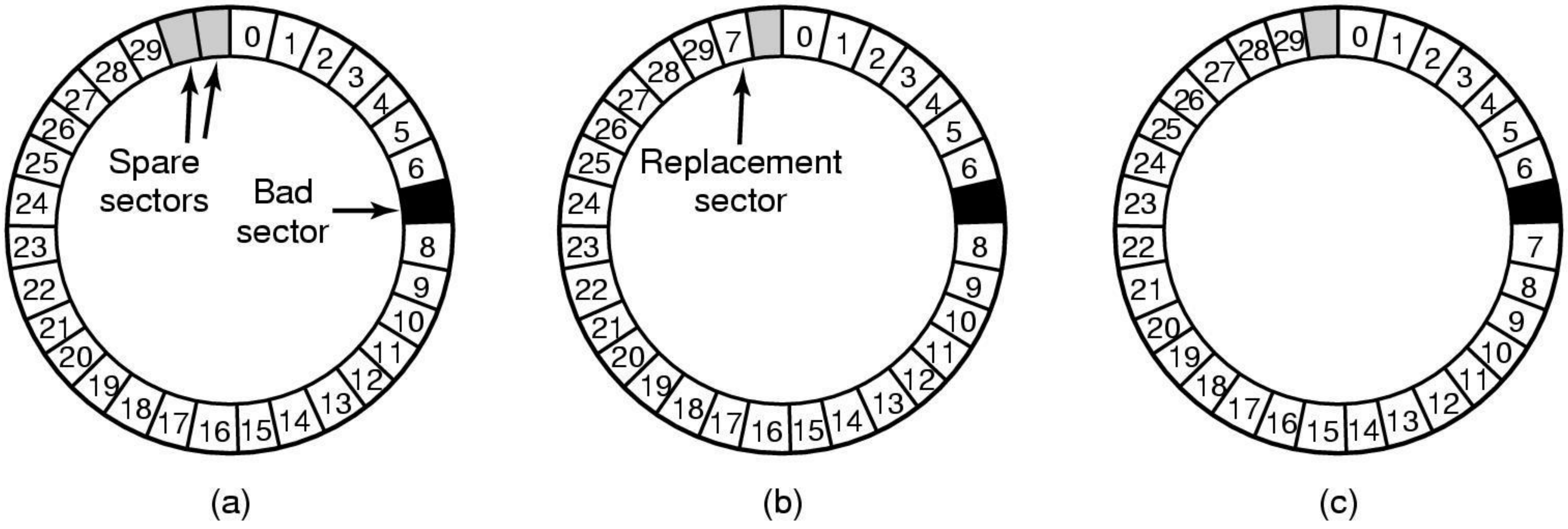
- Write all sector addresses
- Write and read back various data patterns on all sectors
    - Test all sectors
    - Identify *bad blocks*

- *Bad block*
    - Any sector that does not reliably return the data that was written to it!

Bad Block Management

- Bad blocks are inevitable
    - Part of manufacturing process (less than 1%)
        - Detected during formatting
    - Occasionally, blocks become bad during operation
- Manufacturers add extra tracks to all disks
    - Physical capacity = (1 + *x*) * rated_capacity
- Who handles them?
    - *Disk controller:* Bad block list maintained internally
        - Automatically substitutes good blocks
    - *Formatter:* Re-organize track to avoid bad blocks
    - *OS:* Bad block list maintained by OS, bad blocks never used

Bad Sector Handling – within track



a) A disk track with a bad sector
b) Substituting a spare for the bad sector
c) Shifting all the sectors to bypass the bad one

Logical *vs.* Physical Sector Addresses

- Many modern disk controllers convert

  [*cylinder*, *track*, *sector*]

  addresses into *logical sector numbers*
  - Linear array
  - No gaps in addressing
  - Bad blocks concealed by controller
- *Reason:*
  - Backward compatibility with older PC's
  - Limited number of bits in *C*, *T*, and *S* fields

Disk Drive – Performance

- Seek time
  - Position heads over a cylinder – 1 to 25 ms
- Rotational *latency*
  - Wait for sector to rotate under head
  - Full rotation - 4 to 12 ms (15000 to 5400 RPM)
  - Latency averages ½ of rotation time
- Transfer rate
  - approx 40-380 MB/s (aka *bandwidth*)
- Transfer of 1 KB
  - Seek (4 ms) + rotational latency (2ms) + transfer (40 μs) = 6.04 ms
  - Effective BW here is about 170 KB/sec (misleading!)

Disk Reading Strategies

- Read and cache a whole track
  - Automatic in many modern controllers
  - Subsequent reads to same track have zero rotational latency – good for locality of reference!
  - Disk arm available to seek to another cylinder
- Start from current head position
  - Start filling cache with first sector under head
  - Signal completion when desired sector is read
- Start with requested sector
  - When no cache, or limited cache sizes

Disk Writing Strategies

- There are none
- The best one can do is
  - collect together a sequence of contiguous (or nearby) sectors for writing
  - Write them in a single sequence of disk actions
- Caching for later writing is (usually) a *bad idea*
  - Application has no confidence that data is actually written before a failure
  - Some *network disk systems* provide this feature, with battery backup power for protection

Disk Arm Scheduling

- A lot of material in textbooks on this subject.

- Goal
  - Minimize seek time by minimizing seek distance

However …

- In real systems, average disk queue length is often 1-2 requests
  - All strategies are approximately equal!
- If your system typically has queues averaging more than 2 entries, something is seriously wrong!

- Disk arm scheduling used only in a few very specialized situations
  - Multi-media; some transaction-based systems

Performance metrics

- Transaction & database systems
    - Number of transactions per second
    - Focus on seek and rotational latency, not bandwidth
    - Track caching may be irrelevant (except read-modify-write)
- Many little files (e.g., Unix, Linux)
    - Same
- Big files
    - Focus on bandwidth and contiguous allocation
    - Track caching important; seek time is secondary concern
- Paging support for VM
    - A combination of both
    - Track caching is highly relevant – locality of reference

Problem

- Question:
  - If *mean time between failures* of a disk drive is 100,000 hours,
  - and if your system has 100 identical disks,
  - what is mean time between need to replace a drive?
- Answer:
  - 1000 hours (i.e., 41.67 days ≅ 6 weeks)
- I.e.:
  - You lose 1% of your data every 6 weeks!
- But don’t worry – you can restore *most* of it from backup!

Can we do better?

- Yes, *mirrored*
  - Write every block twice, on two separate disks
  - Mean time between simultaneous failure of *both* disks is >57,000 years
- Can we do even better?
  - E.g., use fewer extra disks?
  - E.g., get more performance?

RAID

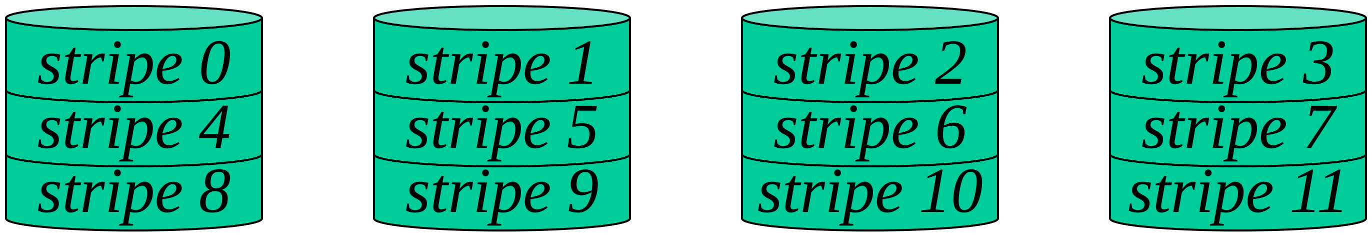
Redundant Array of Independent Disks

- Distribute a file system intelligently across multiple disks to
  - Maintain high reliability *and* availability
  - Enable fast recovery from failure
  - Increase performance

“Levels” of RAID

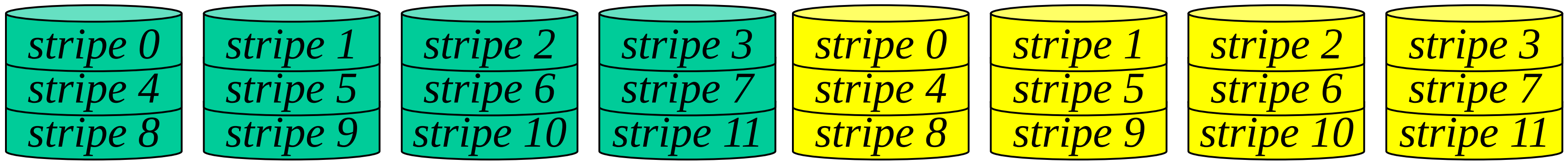
- Level 0 – striping of blocks across disk
- Level 1 – **simple mirroring**
- Level 2 – striping of bytes or bits with ECC
- Level 3 – Level 2 with parity, not ECC
- Level 4 – Level 0 with parity block
- Level 5 – Level 4 with distributed parity blocks
- …

RAID Level 0 – Simple Striping



- Each stripe is one or a group of contiguous blocks
- Block/group *i* is on disk (*i* **mod** *n*)
- Advantage
  - Read/write *n* blocks in parallel; *n* times bandwidth
- Disadvantage
  - No redundancy at all. System MBTF is *1/n* disk MBTF!

RAID Level 01 – Striping and Mirroring



- Each stripe is written twice
    - Two separate, identical disks
- Block/group *i* is on disks (*i* **mod** 2*n*) & (*i*+*n* **mod** 2*n*)
- Advantages
    - Read/write *n* blocks in parallel; *n* times bandwidth
    - Redundancy: System MBTF = (Disk MBTF)^2 at twice the cost
    - Failed disk can be replaced by copying
- Disadvantage
    - A lot of extra disks for much more reliability than we need

RAID Levels 2 & 3

- Bit- or byte-level striping
- Requires synchronized disks
    - Highly impractical
- Requires fancy electronics
    - For ECC calculations
- Not used; academic interest only

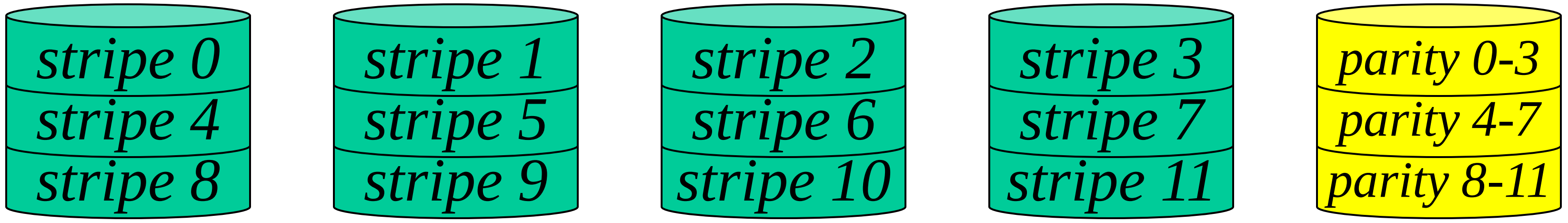
Observation

- When a disk or stripe is read incorrectly,

  we know which one failed!

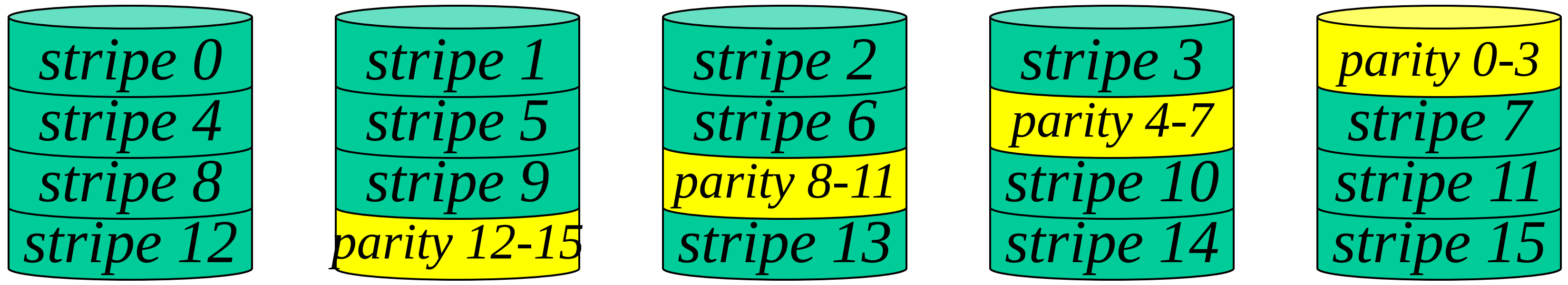
- Conclusion:
  - A simple parity disk can provide very high reliability
    - (unlike simple parity in memory)

RAID Level 4 – Parity Disk



- parity 0-3 = stripe 0 **xor** stripe 1 **xor** stripe 2 **xor** stripe 3
- *n* stripes plus parity are written/read in parallel
- If any disk/stripe fails, it can be reconstructed from others
  - E.g., stripe 1 = stripe 0 **xor** stripe 2 **xor** stripe 3 **xor** parity 0-3
- Advantages
  - *n* times read bandwidth
  - System MBTF = (Disk MBTF)^2 at *1/n* additional cost
  - Failed disk can be reconstructed "on-the-fly" (hot swap)
  - Hot expansion: simply add *n* + *1* disks all initialized to zeros
- However
  - Writing requires read-modify-write of parity stripe ⇒ only 1x write bandwidth.

RAID Level 5 – Distributed Parity



- Parity calculation is same as RAID Level 4
- Advantages & Disadvantages – Mostly same as RAID Level 4
- Additional advantages
  - Avoids beating up on parity disk
  - Some writes in parallel (if no contention for parity drive)

- Writing individual stripes (RAID 4 & 5)
  - Read existing stripe and existing parity
  - Recompute parity
  - Write new stripe and new parity

RAID 4 & 5

- Very popular in data centers
  - Corporate and academic servers
- Built-in support in Windows and Linux
  - Connect a group of disks to fast SCSI port (320 MB/s bandwidth)
  - OS RAID support does the rest!

- Other RAID variations also available

Stable storage

Incomplete Operations

- Problem – how to protect against disk write operations that don’t finish
  - Power or CPU failure in the middle of a block
  - Related series of writes interrupted before all are completed

- Examples:
  - Database update of charge and credit
  - RAID 1 failure between redundant writes

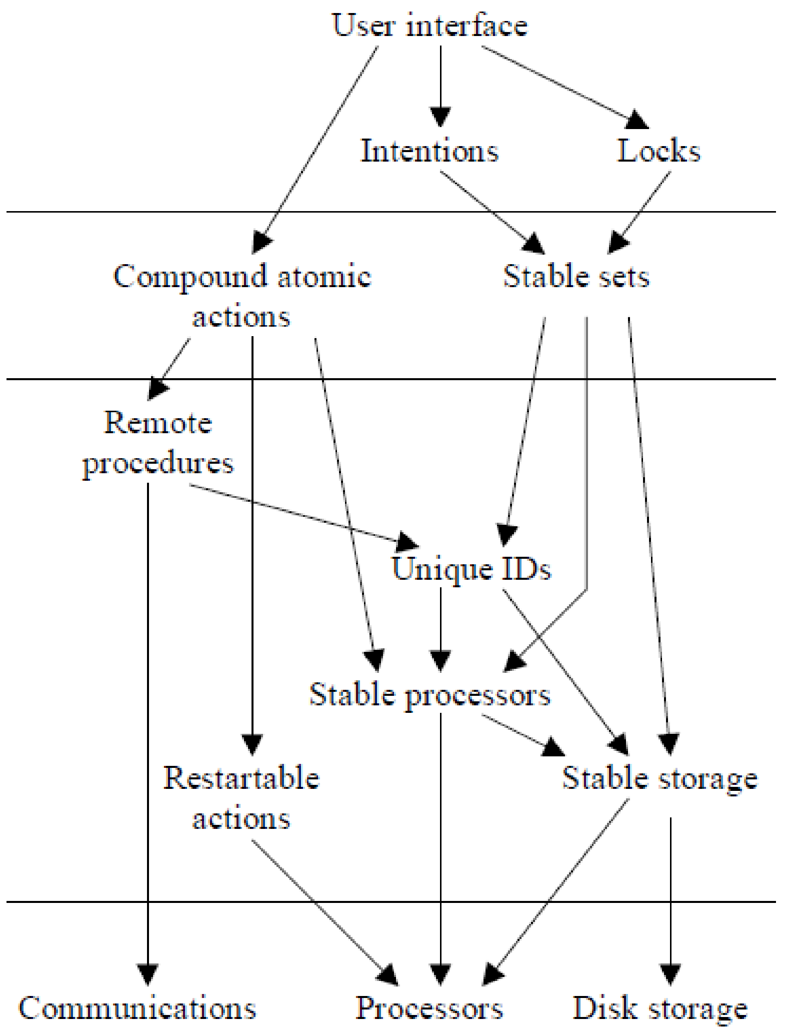
Atomic Transactions

Lampson, B. (1981). *Distributed Systems: Architecture and Implementation, Lecture Notes in Computer Science, Vol. 105*, pp. 246–265.

Butler Lampson

1993 Turing award - For contributions to the development of distributed, personal computing environments and the technology for their implementation: workstations, networks, operating systems, programming systems, displays, security and document publishing.

Abstractions for Transactions



Stable system

Physical system

Abstractions for Transactions

- By eliminating errors, we convert disk storage into an ideal device for recording state, called *stable storage*.

- Processor → A stable processor makes use of stable storage to save its state in order to be able to survive crashes.
- Communications → RPC (over TCP).

Lampson, B., & Sturgis, H. (1979). Crash recovery in a distributed data storage system, Unpublished.

Errors

- We divide the events which occur in the physical system into two categories: *desired* and *undesired*: in a fault-free system only desired events will occur.

- Undesired events are subdivided into expected ones, called *errors*, and unexpected ones, called *disasters*.

Errors

- Our algorithms are designed to work in the presence of any number of errors, and no disasters.
- We have tried to represent as an error, rather than a disaster, any event with a significant probability of occurring.

Disk Storage

- Procedures
  - *Put*(*at*: *Address*, *data*: *Dblock*)
  - *Get*(*at*: *Address*) returns (*status*: (*good*, *looksBad*), *data*: *Dblock*)

Disk Storage

- Results of a *Get*(*at*: *a*):
  - (desired) Page *a* is (*good*, *d*), and *Get* returns (*good*, *d*).
  - (desired) Page *a* is *bad*, and *Get* returns *looksBad*.
  - (error) *Soft read error*: Page *a* is *good*, and *Get* returns *looksBad*, provided this has not happened too often in the recent past: this is made precise in the next event.

Disk Storage

- Results of a *Get*(*at*: *a*)
  - (disaster) *Persistent read error*: Page *a* is *good*, and *Get* returns *looksBad*, and n_R successive *Gets* within a time T_R have all returned *looksBad*.
  - (disaster) *Undetected error*: Page *a* is *bad*, and *Get* returns *good*, or if page *a* is (*good*, *d*), then returns (*good*, *d'*) with *d'* ? *d*.

Disk Storage

- Effects of a *Put*(*at*: *a*, *data*: *d*)
  - (desired) Page *a* becomes (*good*, *d*).
  - (error) *Null write*: Page *a* is unchanged.
  - (error) *Bad write*: Page *a* becomes (*bad*, *d*).

Disk Storage

- Spontaneous events:
  - (error) *Infrequent decay*: a decay preceded and followed by an interval T_D during which there is no other decay in the same unit, and the only bad writes on that unit are to pages in the characteristic set of the decay.
  - (error) *Revival*: a page goes from (*bad*, *d*) to (*good*, *d*).

Disk Storage

- Spontaneous events:
  - (disaster) *Frequent decay*: two decays in the same unit within an interval T_D.
  - (disaster) *Undetected error*: some page changes from (s, d) to (s, d') with $d' \neq d$.

Stable Storage

- The disk storage not used as volatile storage for processor state is converted into *stable storage* with the same actions as disk storage, but with the property that no errors can occur.

Stable Storage

- To construct stable storage, we introduce two successive abstractions, each of which eliminates two of the errors associated with disk storage.
  - Careful disk storage
  - Stable page

Careful disk storage

- Its state and actions are specified exactly like those of disk storage, except that the only errors are a bad write immediately followed by a crash, and infrequent decay.

- A careful page is represented by a disk page.

Careful page

- *CarefulGet* repeatedly does *Get* until it gets a *good* status, or until it has tried *n* times.

- *CarefulPut* repeatedly does *Put* followed by *Get* until the *Get* returns *good* with the data being written.

Stable page

- It is represented by an ordered pair of careful disk pages, chosen from the same unit but not decay-related.

- The value of the data is the data of the first representing page if that page is *good*, otherwise the data of the second page.

Stable page

- A *StableGet* does a *CarefulGet* from one of the representing pages, and if the result is bad does a *CarefulGet* from the other one.

- A *StablePut* does a *CarefulPut* to each of the representing pages in turn; the second *CarefulPut* must not be started until the first is complete.

Stable page

- A third procedure is called *Cleanup*:

```
Do a CarefulGet from each of the two pages;
if both return good and the same data then
        Do nothing
else if one returns bad then
        Do a CarefulPut of the data block obtained from the
        good address to the bad address
else if both return good, but different data then
        Choose either one of the pages and do a CarefulPut of
        its data to the other page
```

Stable Storage

- Data residing in stable storage is never lost (due to a failure)
- To implement stable storage:
  - Replicate data on more than one nonvolatile storage media with independent failure modes
  - Update data in a controlled manner to ensure that we can recover the stable data after any failure during an update (or recovery)

Stable Storage

- Write everything twice to separate disks
    - Be sure 1st write does not invalidate previous 2nd copy
    - RAID 1 is okay
    - Read blocks back to validate; then report completion
- Reading both copies
    - If 1st copy okay, use it – i.e., newest value
    - If 2nd copy different or bad, update it with 1st copy
    - If 1st copy is bad; update it with 2nd copy – i.e., *old value*

Stable Storage (continued)

- Crash recovery
  - Scan disks, compare corresponding blocks
  - If one is bad, replace with good one
  - If both good but different, replace 2nd with 1st copy
- Result:
  - If 1st block is good, it contains latest value
  - If not, 2nd block still contains previous value
- An *abstraction* of an *atomic disk write* of a single block
  - Uninterruptible by power failure, etc.

Scheduling

Disk Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and large disk bandwidth.
- Access time has two major components
  - *Seek time* is the time for the disk to move the heads to the cylinder containing the desired sector.
  - *Rotational latency* is the additional time waiting for the disk to rotate the desired sector to the disk head.
- Minimize seek time
- Seek time ~ seek distance

Disk Arm Scheduling

- Seek time dominates the performance of disk drive transfers
- Can the disk arm be moved to improve the effective disk performance?
- Assume a request queue (0-199)

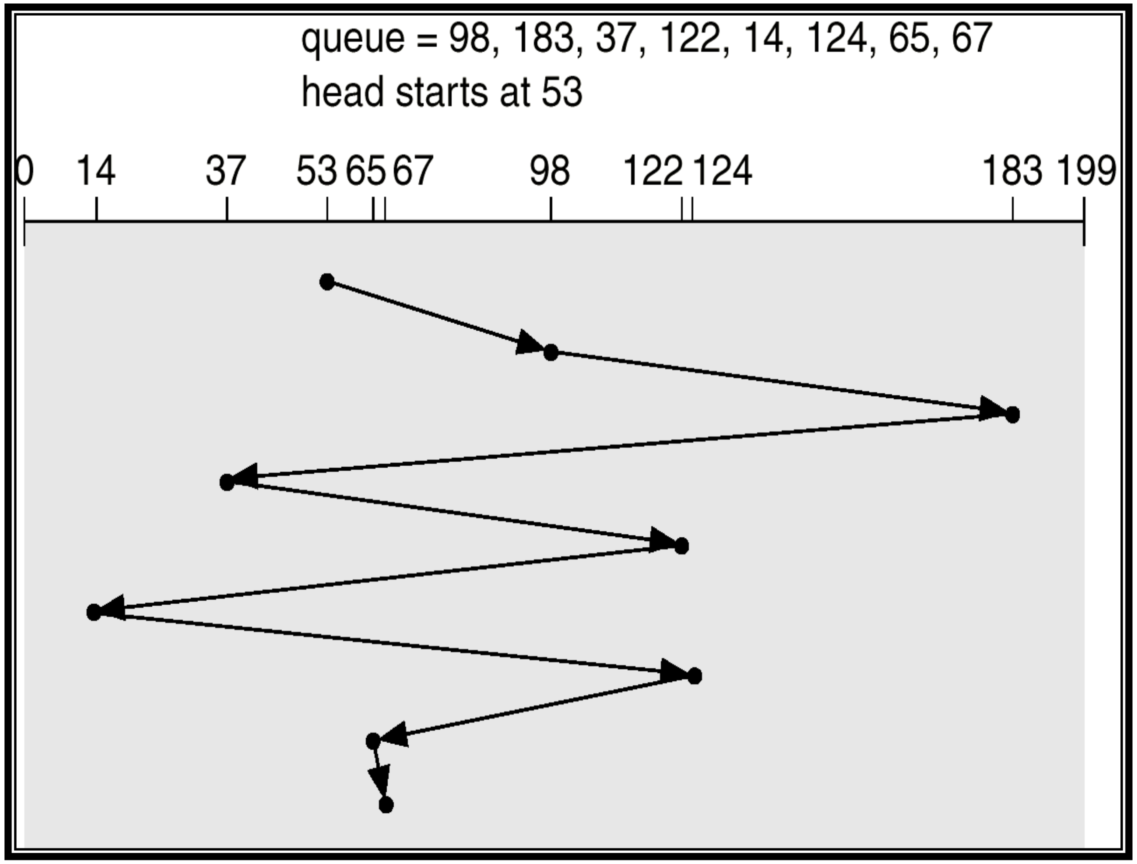
  98, 183, 37, 122, 14, 124, 65, 67

  with current head pointer at 53

Textbook solutions

- *FCFS* – First-come, first-served
- *SSTF* – Shortest seek time first
- *SCAN* (aka *Elevator*) – scan one direction, then the other
- *C-SCAN* – scan in one direction only
- …

FCFS – First come, first served

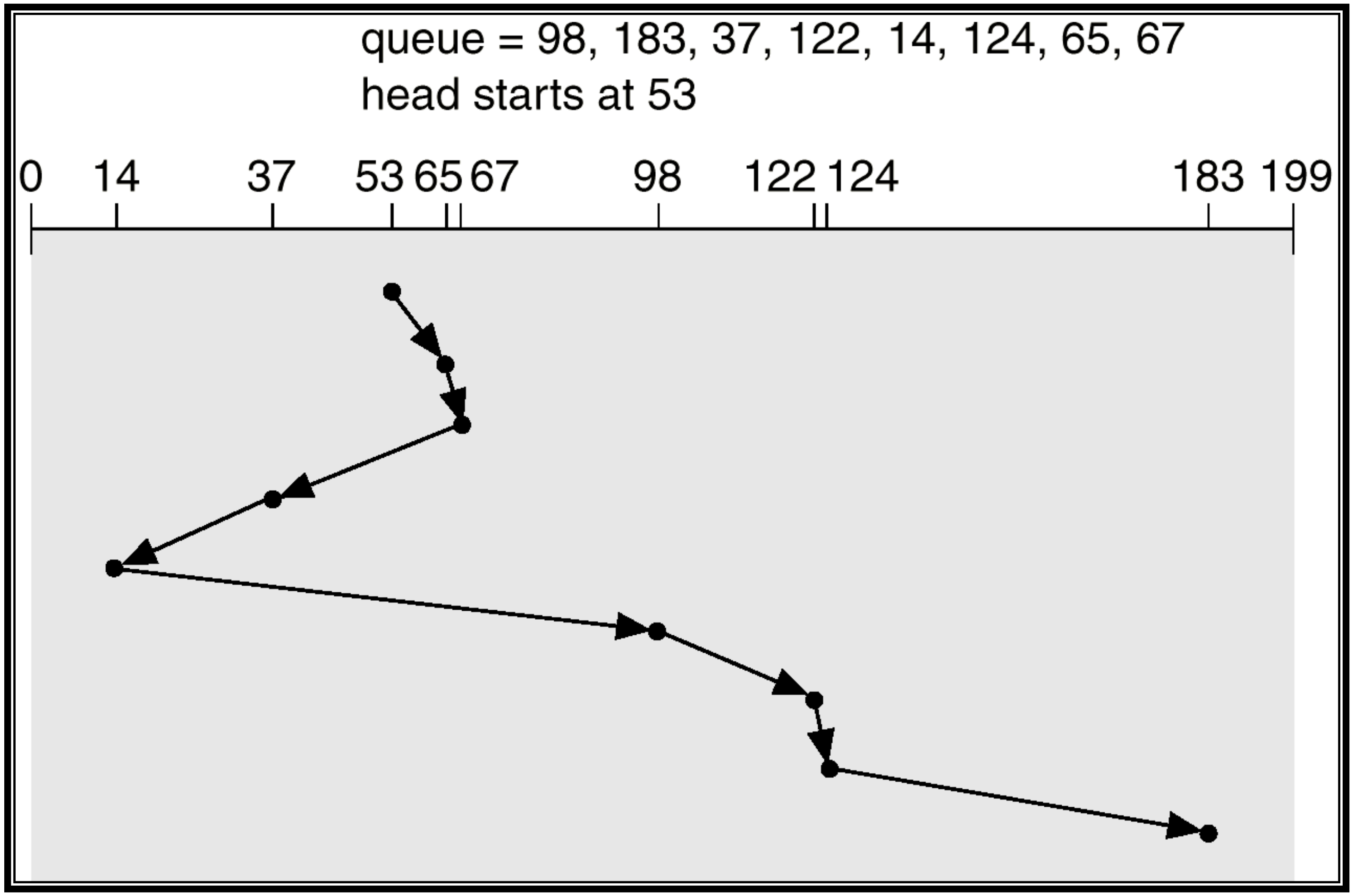


- Example
  - Total head movement of 640 cylinders for request queue
- Pros
  - In order of applications
  - Fair to all requests
- Cons
  - Long seeks

SSTF

- Shortest Seek Time First – Selects request with the minimum seek time from current head position.
- Pro
  - Minimize seek times
- Cons
  - Lingers in areas of high activity
  - Starvation, particularly at edges of disk
- Example
  - Total head movement of 236 cylinders for request queue

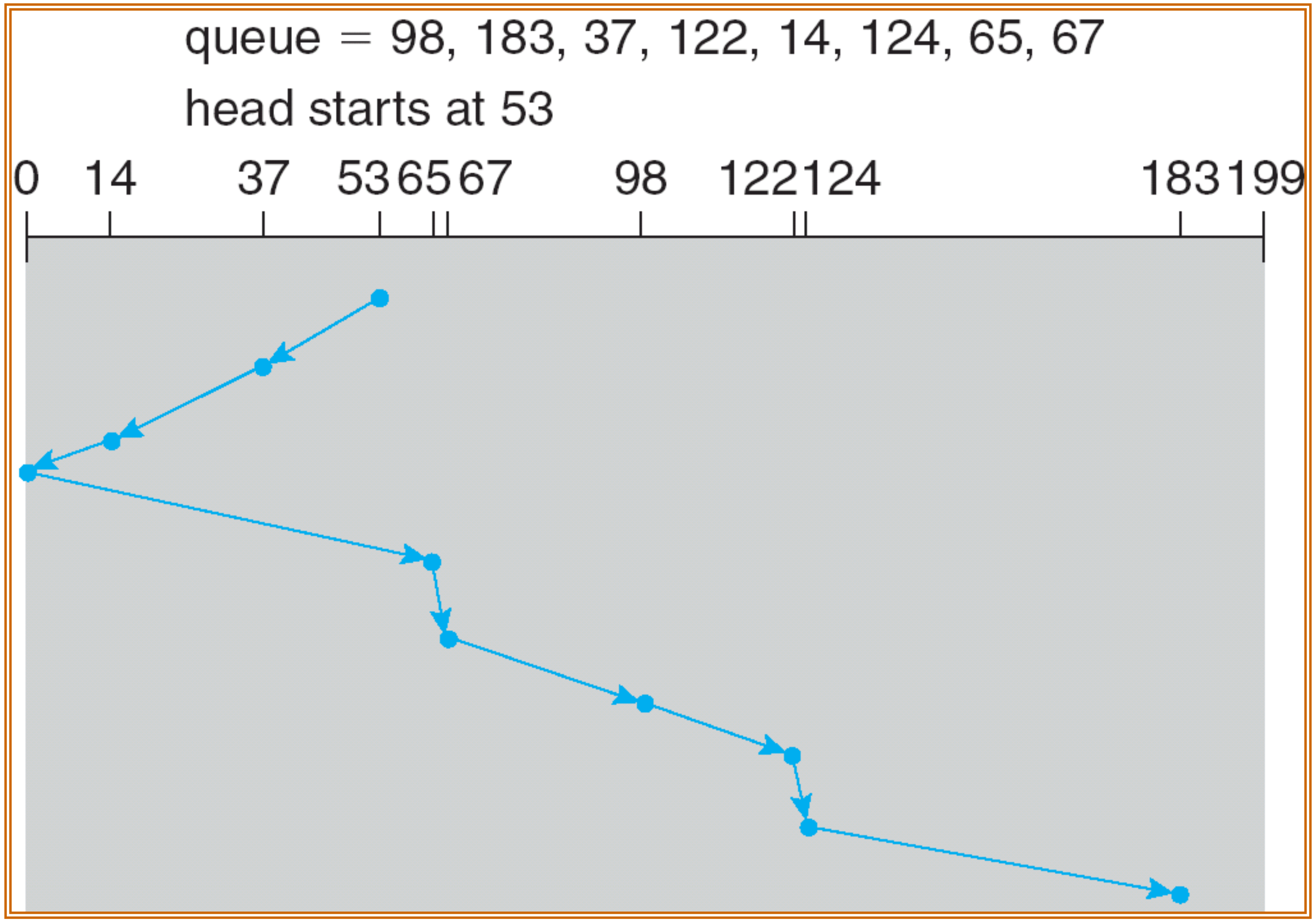
SSTF


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 or Elevator

- The disk arm starts at one end of the disk, moves toward the other end, servicing requests until reaching end of disk, then the head reverses and servicing continues (pick the closest request in same direction as last arm motion)
- Con
  - More arm motion than SSTF
- Pros
  - Fair
  - Avoids starvation
- Example
  - Total head movement of 208 cylinders.

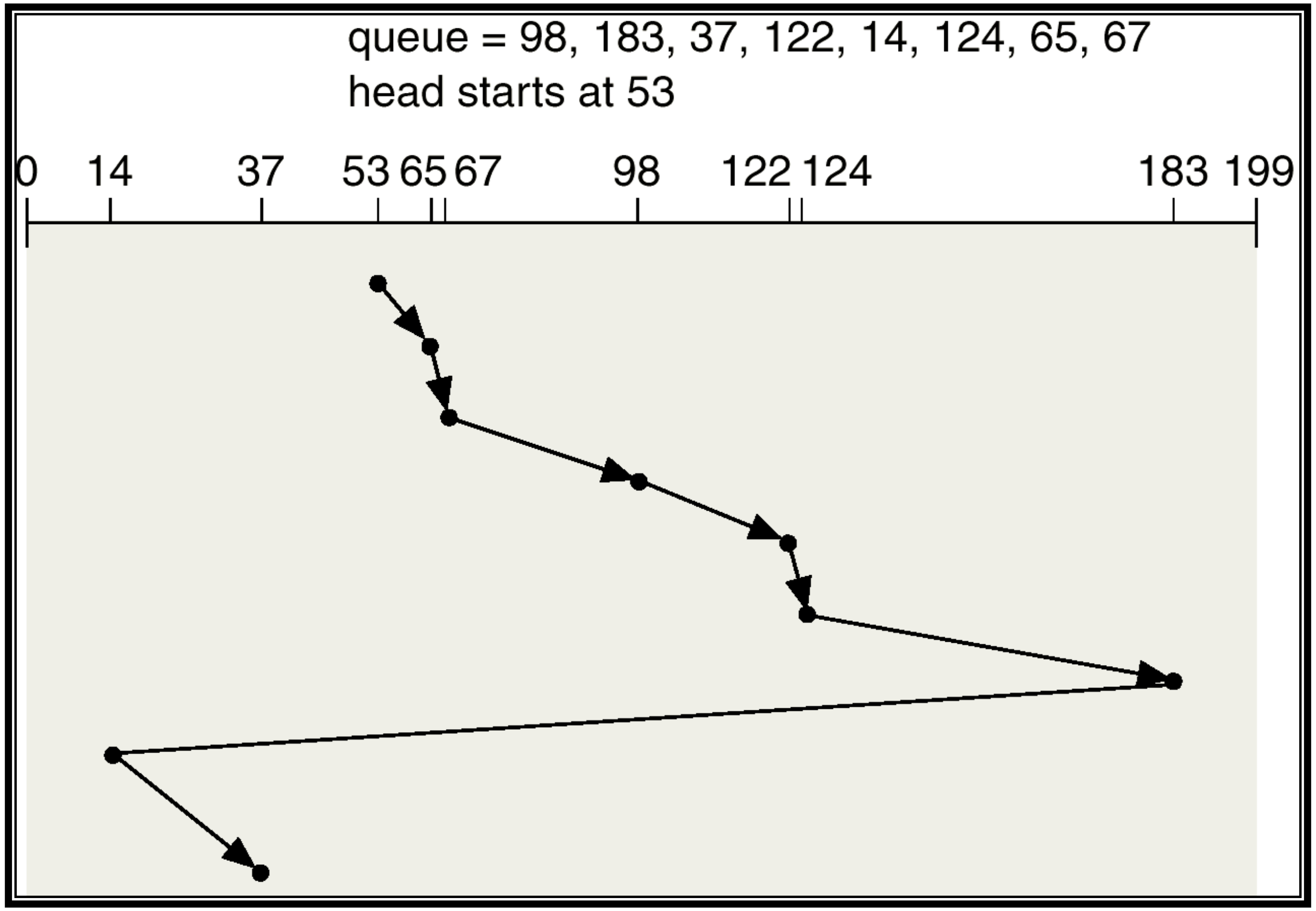
Scan (continued)



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, 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

Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal.
- SCAN and C-SCAN perform better for systems that place heavy load on the disk.
- Performance depends on the number and types of requests.
- Requests for disk service are influenced by the file-allocation method.

Solid-State Disks

- Nonvolatile memory used like a hard drive (many technology variations)
- Can be more reliable than HDDs
- More expensive per MB
- Maybe have shorter life span
- Less capacity, but much faster
- No moving parts, so no seek time or rotational latency