

Chapter 10: Mass-Storage Systems





Chapter 10: Mass-Storage Systems

Overview of Mass Storage Structure

Disk Structure

Disk Attachment

Disk Scheduling

Disk Management

Swap-Space Management

RAID Structure

Stable-Storage Implementation





Objectives

To describe the physical structure of secondary storage devices and its effects on the uses of the devices

To explain the performance characteristics of mass-storage devices

To evaluate disk scheduling algorithms

To discuss operating-system services provided for mass storage, including RAID





Overview of Mass Storage Structure

Magnetic disks provide bulk of secondary storage of modern computers

Drives rotate at 60 to 250 times per second

Transfer rate is rate at which data flow between drive and computer

Positioning time (**random-access time**) is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**)

Head crash results from disk head making contact with the disk surface -- That's bad

Disks can be removable

Drive attached to computer via **I/O bus**

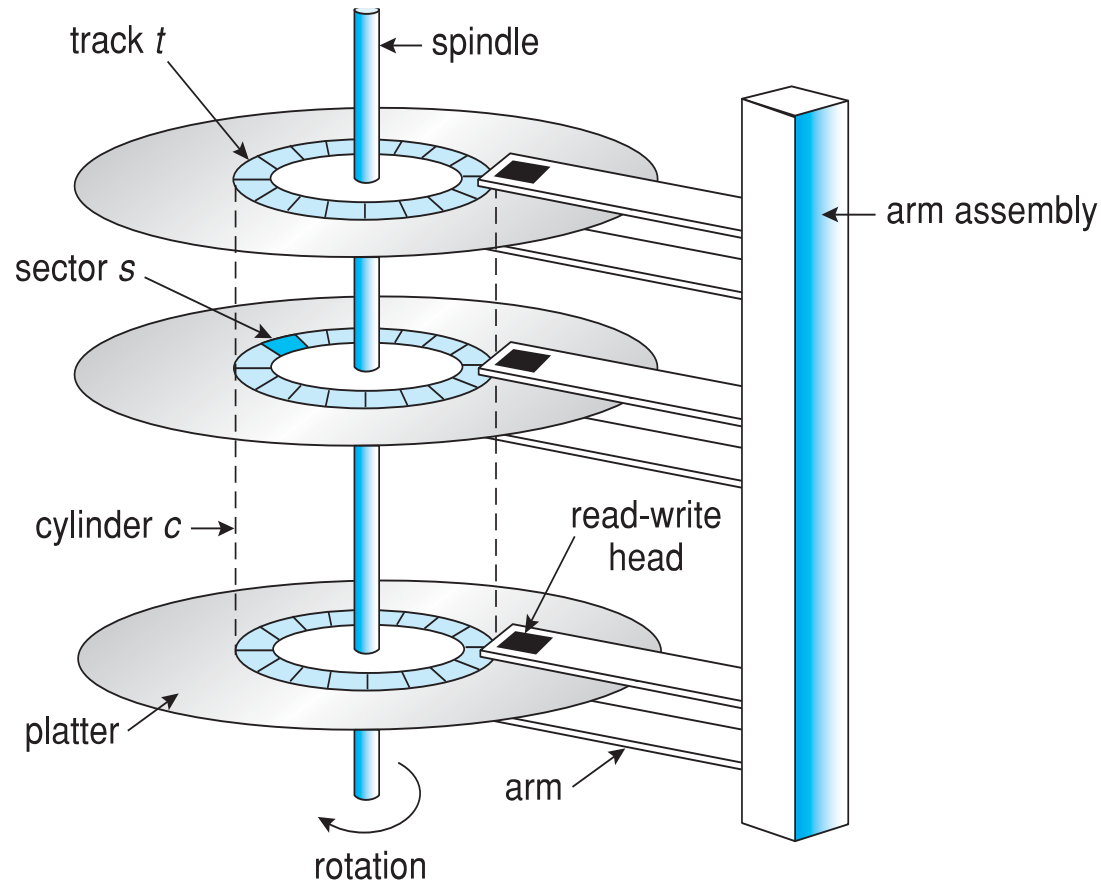
Busses vary, including **EIDE**, **ATA**, **SATA**, **USB**, **Fibre Channel**, **SCSI**, **SAS**, **Firewire**

Host controller in computer uses bus to talk to **disk controller** built into drive or storage array





Moving-head Disk Mechanism





Hard Disks

Platters range from .85" to 14" (historically)

Commonly 3.5", 2.5", and 1.8"

Range from 30GB to 3TB per drive

Performance

Transfer Rate – theoretical – 6 Gb/sec

Effective Transfer Rate – real –
1Gb/sec

Seek time from 3ms to 12ms – 9ms
common for desktop drives

Average seek time measured or
calculated based on 1/3 of tracks

Latency based on spindle speed

▶ $1 / (\text{RPM} / 60) = 60 / \text{RPM}$

Average latency = $\frac{1}{2}$ latency

Spindle [rpm]	Average latency [ms]
4200	7.14
5400	5.56
7200	4.17
10000	3
15000	2

(From Wikipedia)





Hard Disk Performance

Access Latency = **Average access time** = average seek time + average latency

For fastest disk $3\text{ms} + 2\text{ms} = 5\text{ms}$

For slow disk $9\text{ms} + 5.56\text{ms} = 14.56\text{ms}$

Average I/O time = average access time + (amount to transfer / transfer rate) + controller overhead

For example to transfer a 4KB block on a 7200 RPM disk with a 5ms average seek time, 1Gb/sec transfer rate with a .1ms controller overhead =

$5\text{ms} + 4.17\text{ms} + 0.1\text{ms} + \text{transfer time} =$

Transfer time = $4\text{KB} / 1\text{Gb/s} * 8\text{Gb} / \text{GB} * 1\text{GB} / 1024^2\text{KB} = 32 / (1024^2) = 0.031 \text{ ms}$

Average I/O time for 4KB block = $9.27\text{ms} + .031\text{ms} = 9.301\text{ms}$





The First Commercial Disk Drive



1956
IBM RAMDAC computer
included the IBM Model
350 disk storage system

5M (7 bit) characters
50 x 24" platters
Access time = < 1 second





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

Busses can be too slow -> connect directly to PCI for example

No moving parts, so no seek time or rotational latency





Magnetic Tape

Was early secondary-storage medium

Evolved from open spools to cartridges

Relatively permanent and holds large quantities of data

Access time slow

Random access ~1000 times slower than disk

Mainly used for backup, storage of infrequently-used data, transfer medium between systems

Kept in spool and wound or rewound past read-write head

Once data under head, transfer rates comparable to disk

140MB/sec and greater

200GB to 1.5TB typical storage

Common technologies are LTO- $\{3,4,5\}$ and T10000





Disk Structure

Disk drives are addressed as large 1-dimensional arrays of **logical blocks**, where the logical block is the smallest unit of transfer

Low-level formatting creates **logical blocks** on physical media

The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially

Sector 0 is the first sector of the first track on the outermost cylinder

Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost

Logical to physical address should be easy

- ▶ Except for bad sectors
- ▶ Non-constant # of sectors per track via constant angular velocity





Disk Attachment

Host-attached storage accessed through I/O ports talking to I/O busses

SCSI itself is a bus, up to 16 devices on one cable, **SCSI initiator** requests operation and **SCSI targets** perform tasks

Each target can have up to 8 **logical units** (disks attached to device controller)

FC is high-speed serial architecture

Can be switched fabric with 24-bit address space – the basis of **storage area networks (SANs)** in which many hosts attach to many storage units

I/O directed to bus ID, device ID, logical unit (LUN)





Storage Array

Can just attach disks, or arrays of disks

Storage Array has controller(s), provides features to attached host(s)

Ports to connect hosts to array

Memory, controlling software (sometimes NVRAM, etc)

A few to thousands of disks

RAID, hot spares, hot swap (discussed later)

Shared storage -> more efficiency

Features found in some file systems

- ▶ Snapshots, clones, thin provisioning, replication, deduplication, etc

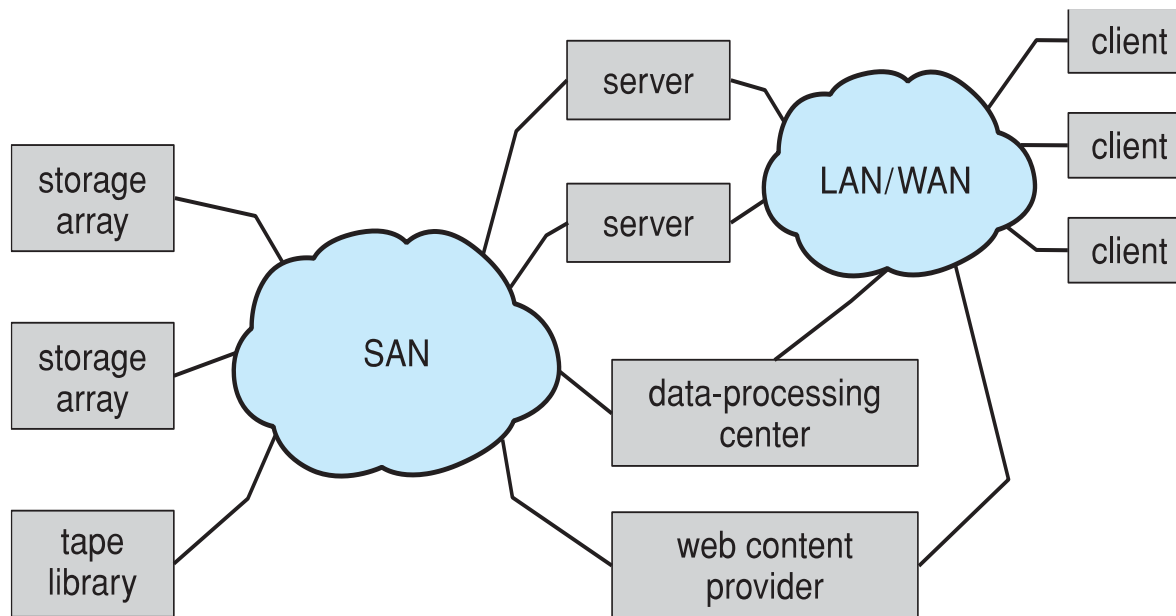




Storage Area Network

Common in large storage environments

Multiple hosts attached to multiple storage arrays - flexible





Storage Area Network (Cont.)

SAN is one or more storage arrays

Connected to one or more Fibre Channel switches

Hosts also attach to the switches

Storage made available via **LUN Masking** from specific arrays to specific servers

Easy to add or remove storage, add new host and allocate it storage

Over low-latency Fibre Channel fabric

Why have separate storage networks and communications networks?

Consider iSCSI, FCOE





Network-Attached Storage

Network-attached storage (**NAS**) is storage made available over a network rather than over a local connection (such as a bus)

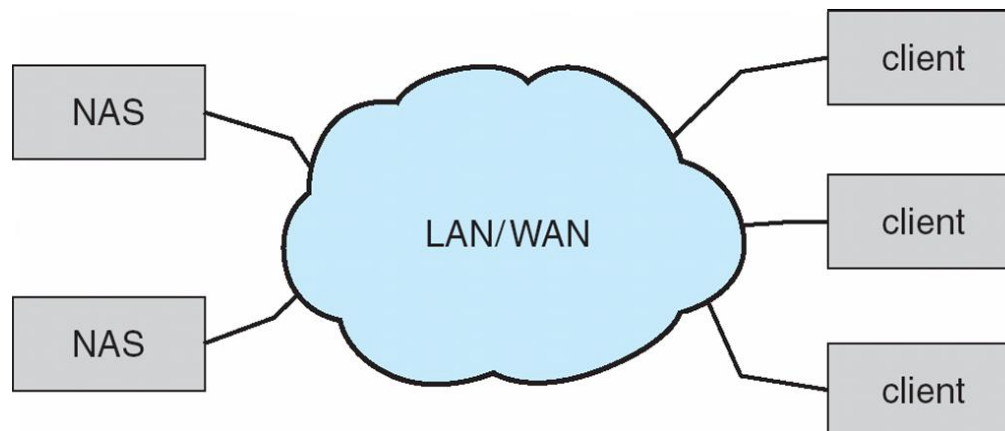
Remotely attaching to file systems

NFS and CIFS are common protocols

Implemented via remote procedure calls (RPCs) between host and storage over typically TCP or UDP on IP network

iSCSI protocol uses IP network to carry the SCSI protocol

Remotely attaching to devices (blocks)





Disk Scheduling

The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth

Minimize seek time

Seek time \approx seek distance

Disk **bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer





Disk Scheduling (Cont.)

There are many sources of disk I/O request

- OS

- System processes

- Users processes

I/O request includes input or output mode, disk address, memory address, number of sectors to transfer

OS maintains queue of requests, per disk or device

Idle disk can immediately work on I/O request, busy disk means work must queue

Optimization algorithms only make sense when a queue exists





Disk Scheduling (Cont.)

Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying “depth”)

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

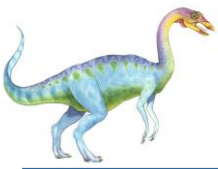
The analysis is true for one or many platters

We illustrate scheduling algorithms with a request queue (0-199)

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

Head pointer 53



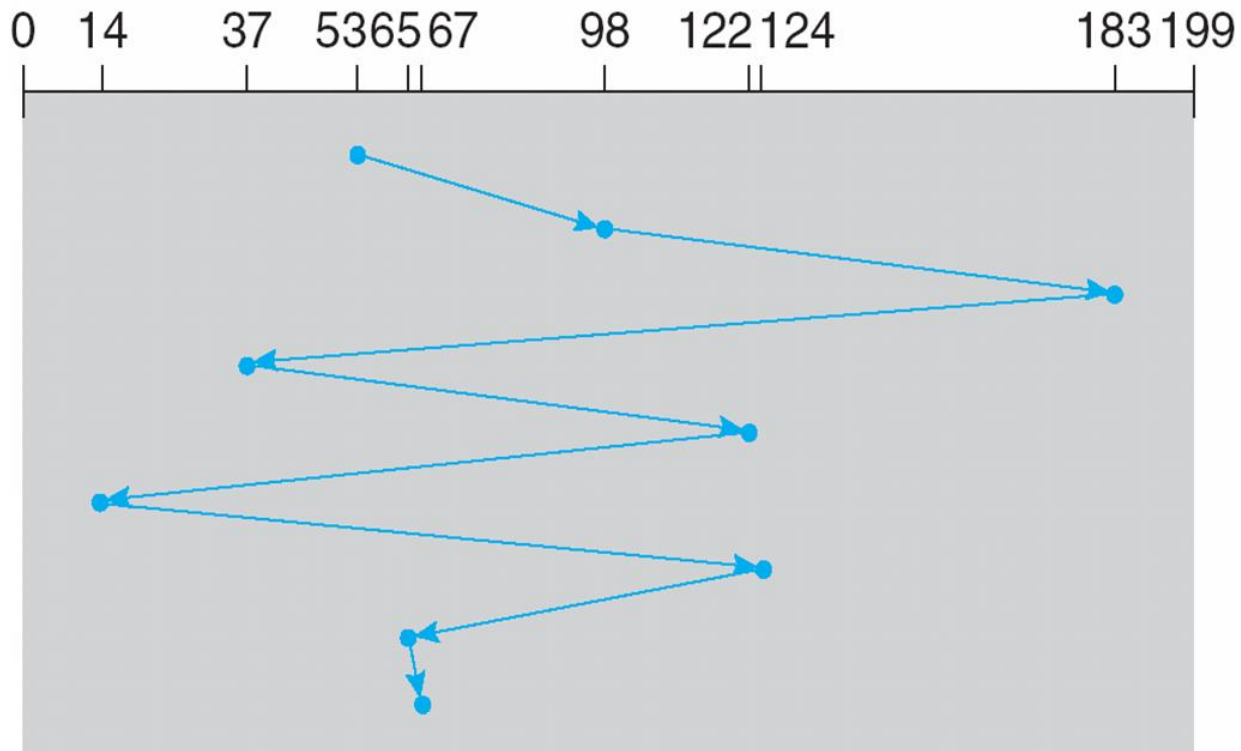


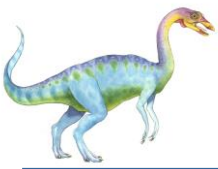
FCFS

Illustration shows total head movement of 640 cylinders

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

head starts at 53



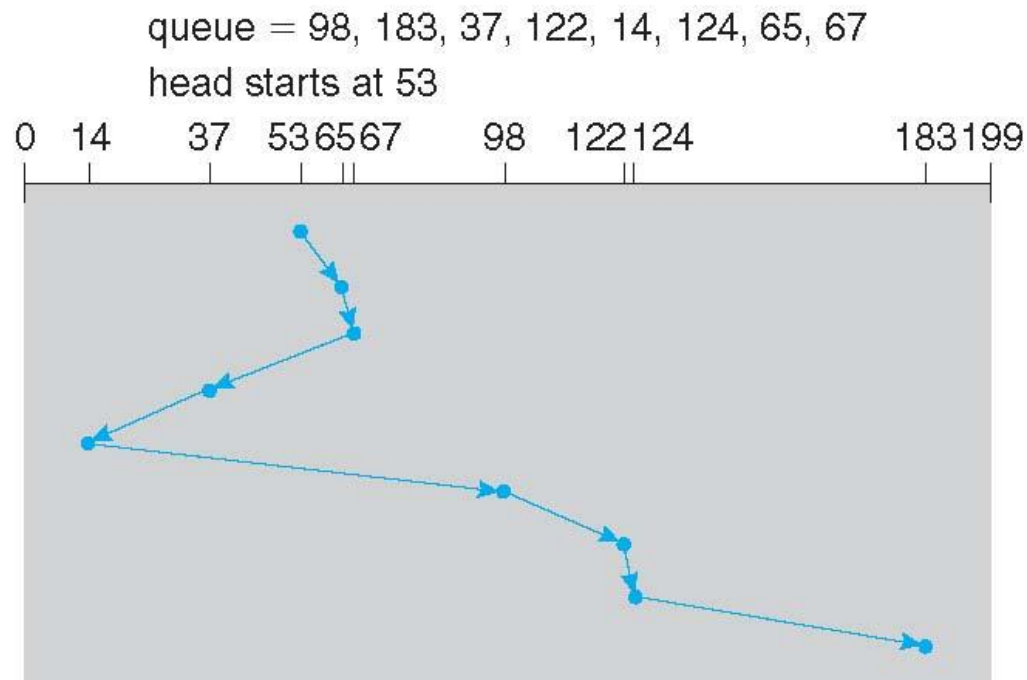


SSTF

Shortest Seek Time First 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





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.

SCAN algorithm Sometimes called the **elevator algorithm**

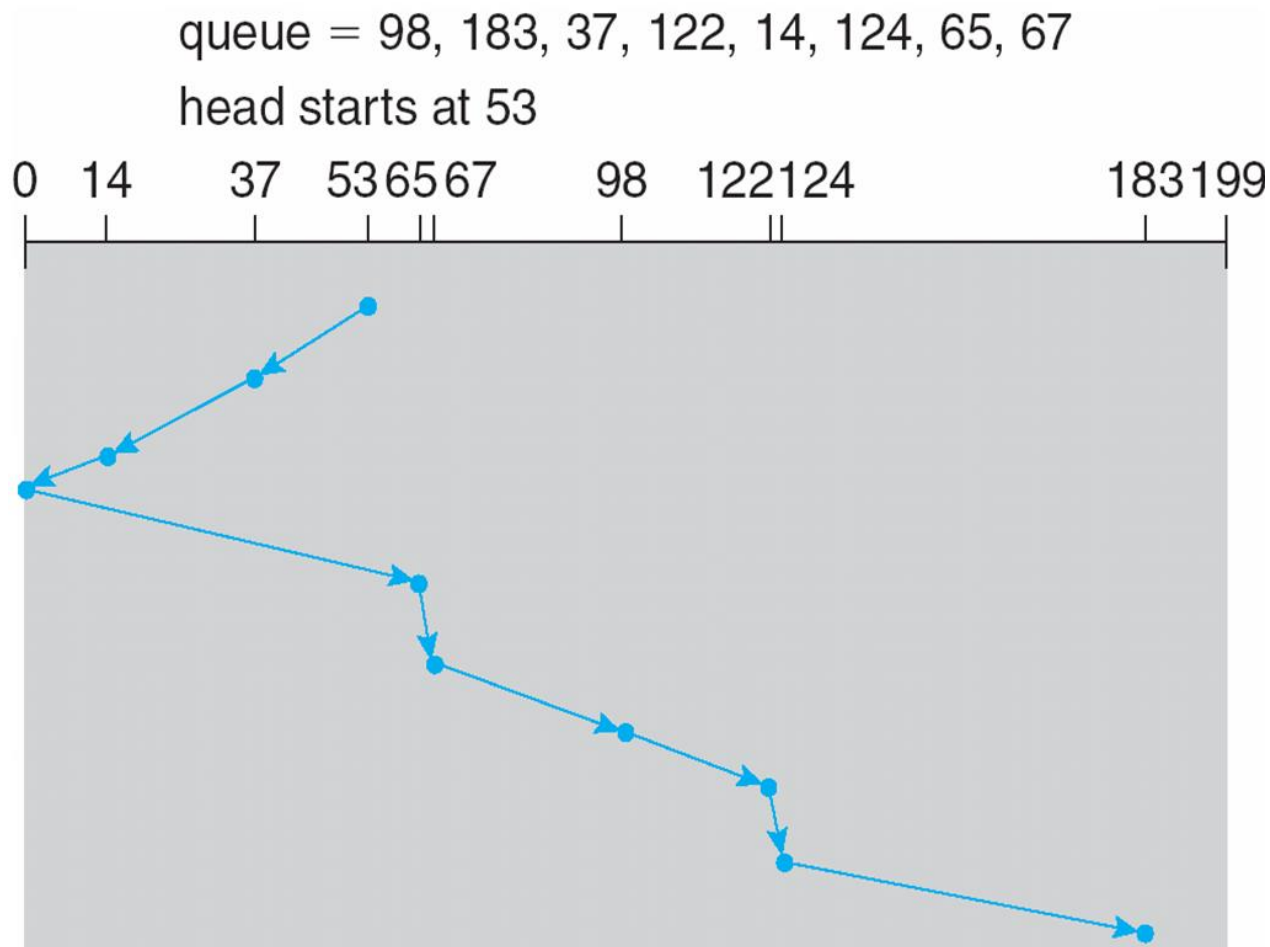
Illustration shows total head movement of 236 cylinders

But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest





SCAN (Cont.)





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

Total number of cylinders?

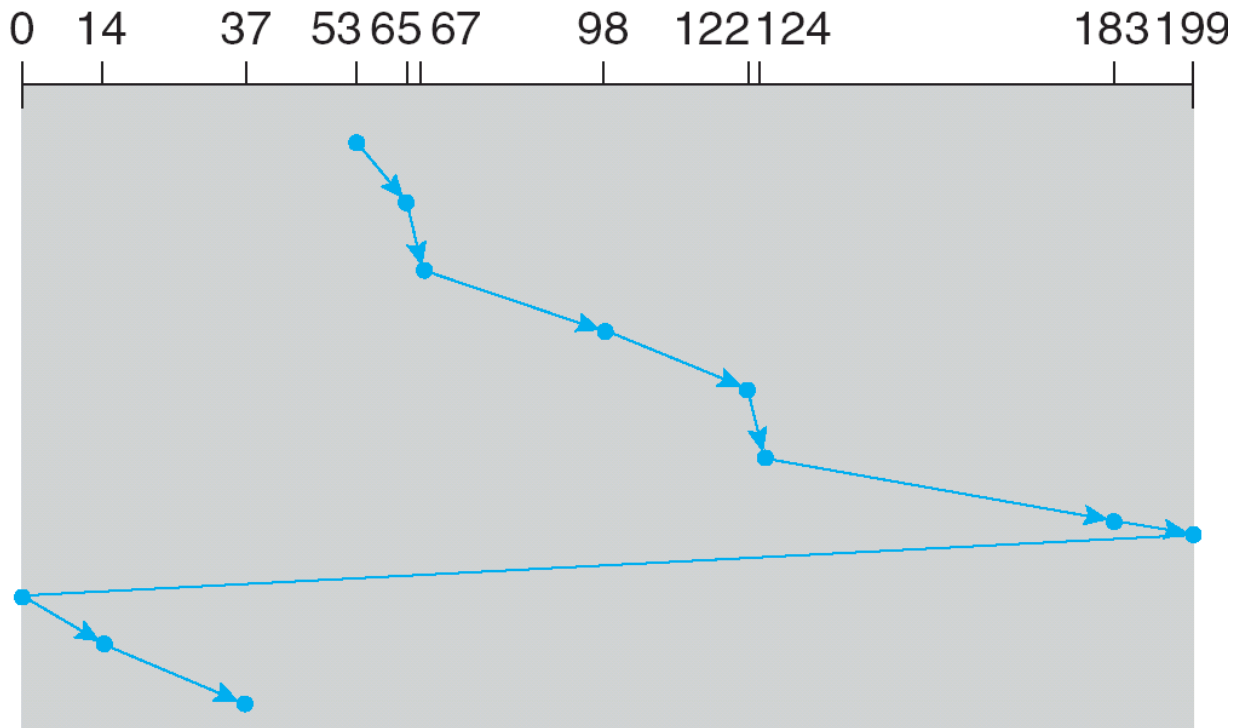


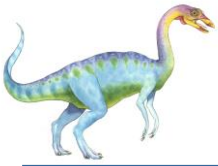


C-SCAN (Cont.)

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

head starts at 53





C-LOOK

LOOK a version of SCAN, C-LOOK a 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

Total number of cylinders?

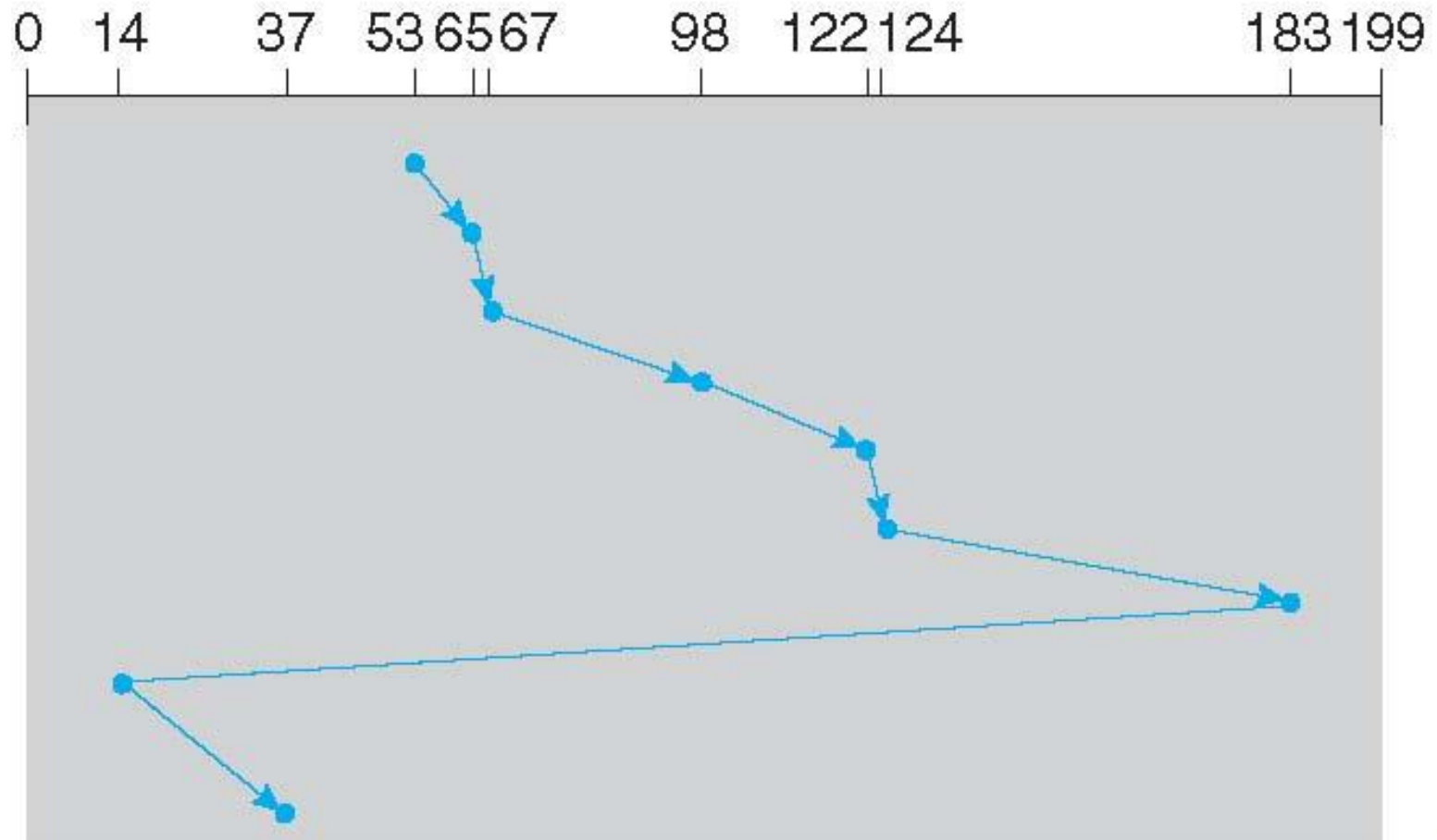




C-LOOK (Cont.)

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

head starts at 53





Selecting a Disk-Scheduling Algorithm

SSTF is common and has a natural appeal

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

Less starvation

Performance depends on the number and types of requests

Requests for disk service can be influenced by the file-allocation method

And metadata layout

The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary

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

What about rotational latency?

Difficult for OS to calculate

How does disk-based queueing effect OS queue ordering efforts?





Disk Management

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

Each sector can hold header information, plus data, plus error correction code (**ECC**)

Usually 512 bytes of data but can be selectable

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, each treated as a logical disk

Logical formatting or “making a file system”

To increase efficiency most file systems group blocks into **clusters**

- ▶ Disk I/O done in blocks
- ▶ File I/O done in clusters





Disk Management (Cont.)

Raw disk access for apps that want to do their own block management, keep OS out of the way (databases for example)

Boot block initializes system

The bootstrap is stored in ROM

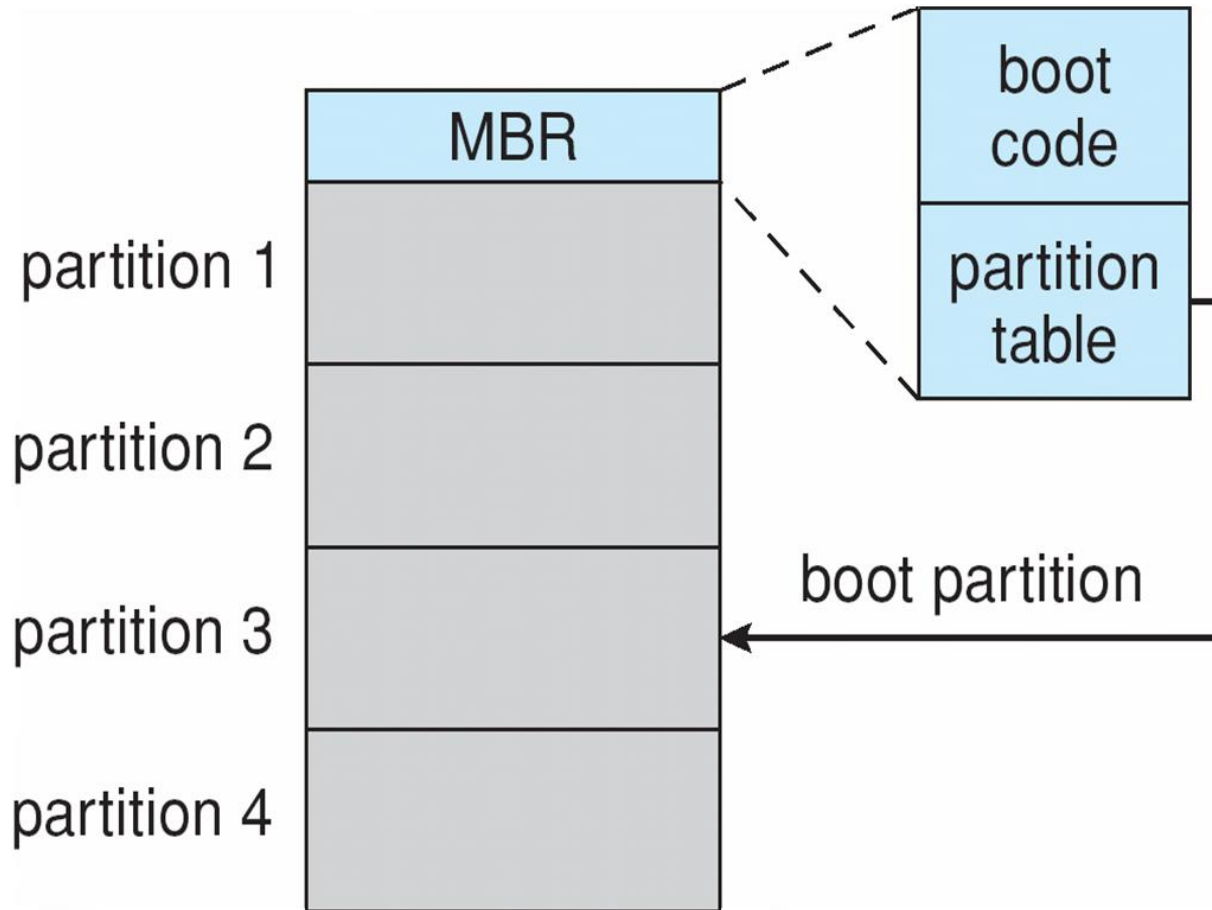
Bootstrap loader program stored in boot blocks of boot partition

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





Booting from a Disk in Windows





Swap-Space Management

Swap-space — Virtual memory uses disk space as an extension of main memory

Less common now due to memory capacity increases

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

Swap-space management

4.3BSD allocates swap space when process starts; holds text segment (the program) and data segment

Kernel uses **swap maps** to track swap-space use

Solaris 2 allocates swap space only when a dirty page is forced out of physical memory, not when the virtual memory page is first created

- ▶ File data written to swap space until write to file system requested
- ▶ Other dirty pages go to swap space due to no other home
- ▶ Text segment pages thrown out and reread from the file system as needed

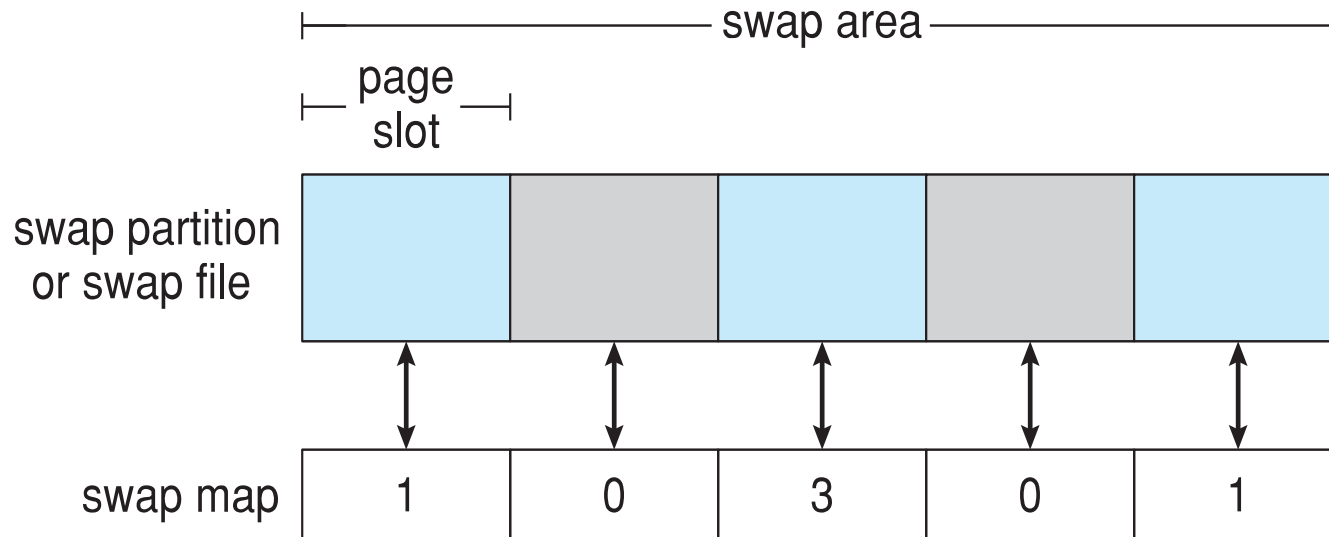
What if a system runs out of swap space?

Some systems allow multiple swap spaces





Data Structures for Swapping on Linux Systems





RAID Structure

RAID – redundant array of inexpensive disks

multiple disk drives provides reliability via **redundancy**

Increases the **mean time to failure**

Mean time to repair – exposure time when another failure could cause data loss

Mean time to data loss based on above factors

If mirrored disks fail independently, consider disk with 1300,000 mean time to failure and 10 hour mean time to repair

Mean time to data loss is $100,000^2 / (2 * 10) = 500 * 10^6$ hours, or 57,000 years!

Frequently combined with **NVRAM** to improve write performance

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





RAID (Cont.)

Disk **striping** uses a group of disks as one storage unit

RAID is arranged into six different levels

RAID schemes improve performance and improve the reliability of the storage system by storing redundant data

Mirroring or **shadowing** (**RAID 1**) keeps duplicate of each disk

Striped mirrors (**RAID 1+0**) or mirrored stripes (**RAID 0+1**) provides high performance and high reliability

Block interleaved parity (**RAID 4, 5, 6**) uses much less redundancy

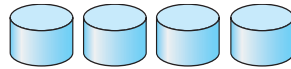
RAID within a storage array can still fail if the array fails, so automatic **replication** of the data between arrays is common

Frequently, a small number of **hot-spare** disks are left unallocated, automatically replacing a failed disk and having data rebuilt onto them





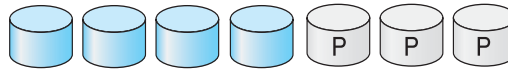
RAID Levels



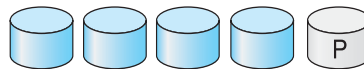
(a) RAID 0: non-redundant striping.



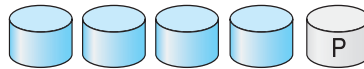
(b) RAID 1: mirrored disks.



(c) RAID 2: memory-style error-correcting codes.



(d) RAID 3: bit-interleaved parity.



(e) RAID 4: block-interleaved parity.



(f) RAID 5: block-interleaved distributed parity.

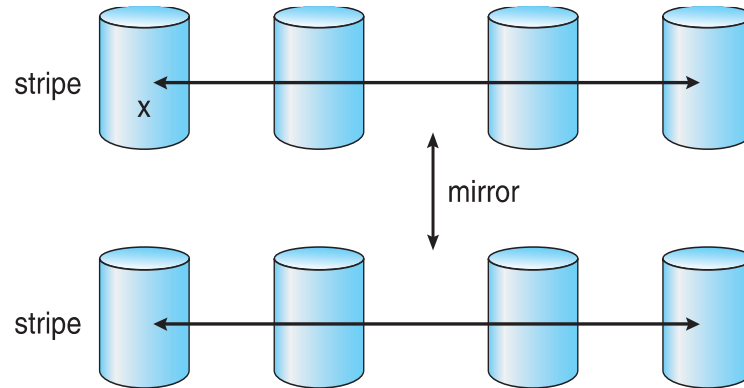


(g) RAID 6: P + Q redundancy.

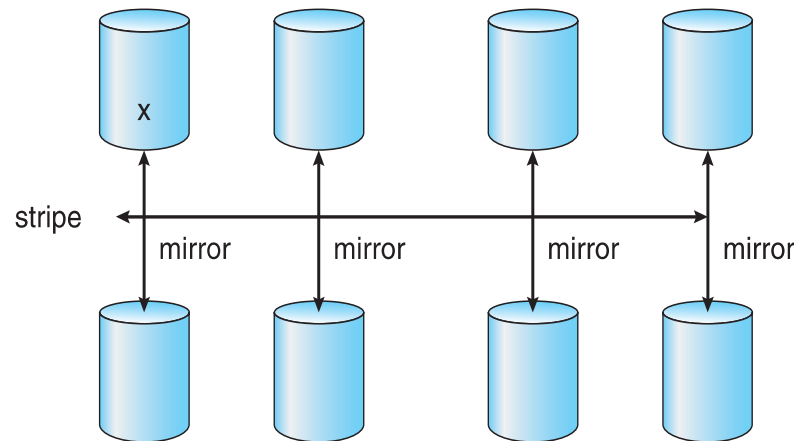




RAID (0 + 1) and (1 + 0)



a) RAID 0 + 1 with a single disk failure.



b) RAID 1 + 0 with a single disk failure.





Other Features

Regardless of where RAID implemented, other useful features can be added

Snapshot is a view of file system before a set of changes take place (i.e. at a point in time)

More in Ch 12

Replication is automatic duplication of writes between separate sites

For redundancy and disaster recovery

Can be synchronous or asynchronous

Hot spare disk is unused, automatically used by RAID production if a disk fails to replace the failed disk and rebuild the RAID set if possible

Decreases mean time to repair





Extensions

RAID alone does not prevent or detect data corruption or other errors, just disk failures

Solaris ZFS adds **checksums** of all data and metadata

Checksums kept with pointer to object, to detect if object is the right one and whether it changed

Can detect and correct data and metadata corruption

ZFS also removes volumes, partitions

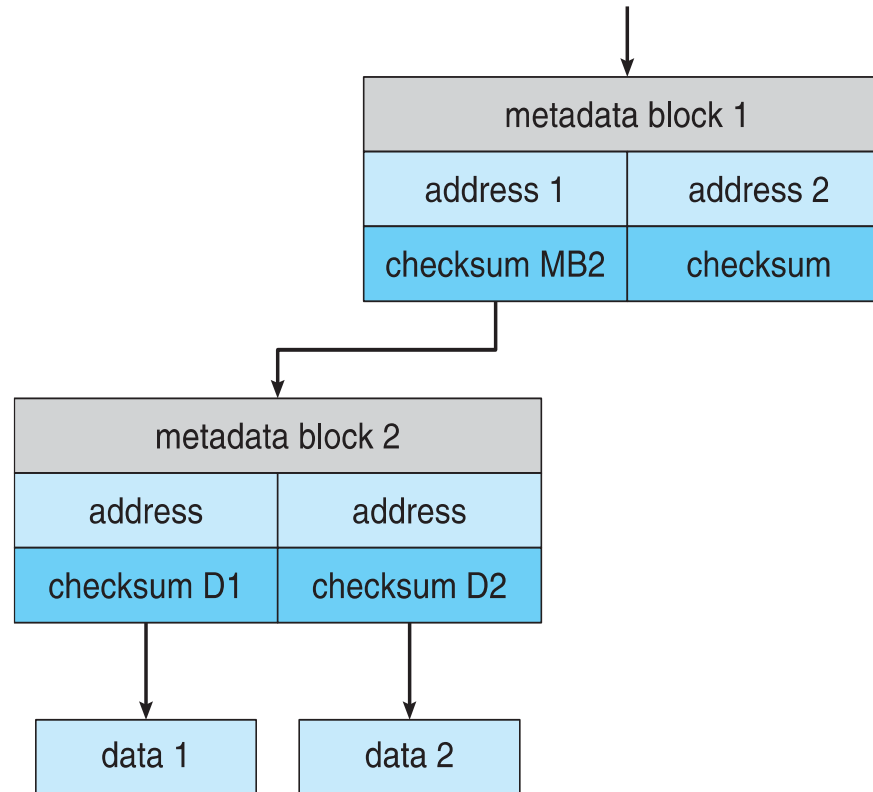
Disks allocated in **pools**

Filesystems with a pool share that pool, use and release space like `malloc()` and `free()` memory allocate / release calls



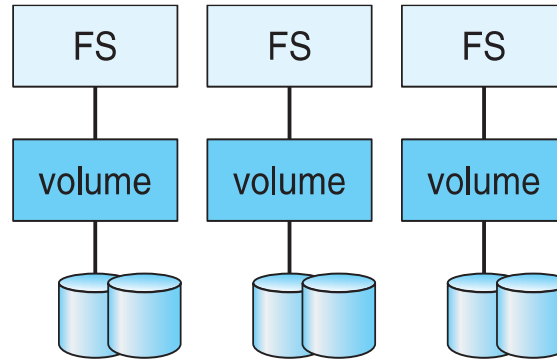


ZFS Checksums All Metadata and Data

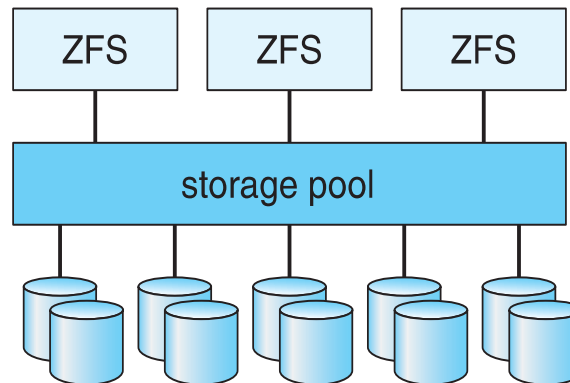




Traditional and Pooled Storage



(a) Traditional volumes and file systems.



(b) ZFS and pooled storage.





Stable-Storage Implementation

Write-ahead log scheme requires stable storage

Stable storage means data is never lost (due to failure, etc)

To implement stable storage:

Replicate information on more than one nonvolatile storage media with independent failure modes

Update information in a controlled manner to ensure that we can recover the stable data after any failure during data transfer or recovery

Disk write has 1 of 3 outcomes

1. **Successful completion** - The data were written correctly on disk
2. **Partial failure** - A failure occurred in the midst of transfer, so only some of the sectors were written with the new data, and the sector being written during the failure may have been corrupted
3. **Total failure** - The failure occurred before the disk write started, so the previous data values on the disk remain intact





Stable-Storage Implementation (Cont.)

If failure occurs during block write, recovery procedure restores block to consistent state

System maintains 2 physical blocks per logical block and does the following:

1. Write to 1st physical
2. When successful, write to 2nd physical
3. Declare complete only after second write completes successfully

Systems frequently use NVRAM as one physical to accelerate



End of Chapter 10

