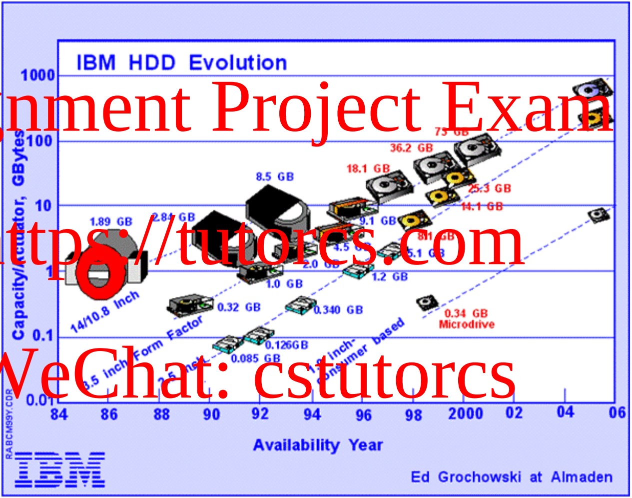
Disk Management

Anandha Gopalan
(with thanks to D. Rueckert, P. Pietzuch, A. Tannenbaum and R. Kolcun)
axgopala@imperial.ac.uk

Imperial College London

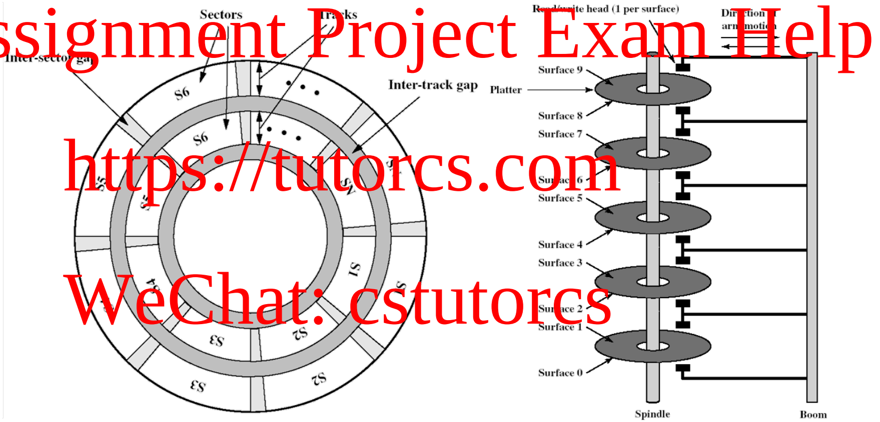
Disk Evolution



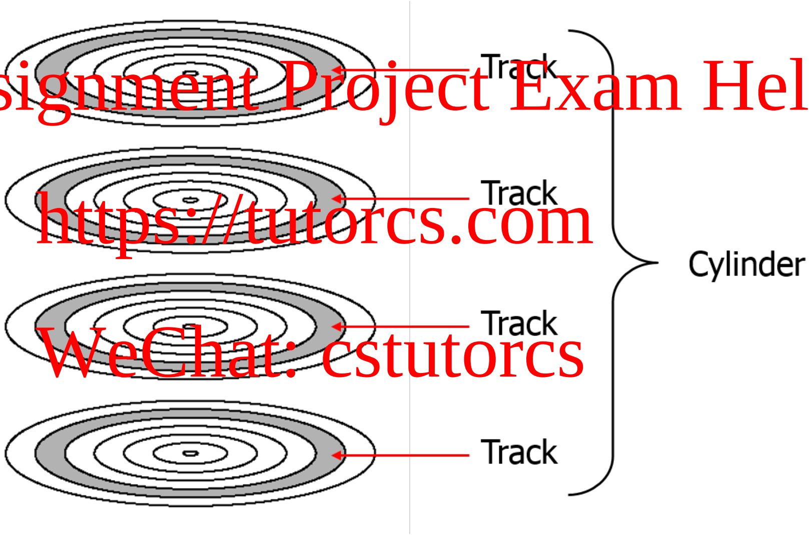
Capacity increases exponentially, but access speeds not so much

The Hard Drive

Disk Storage Devices

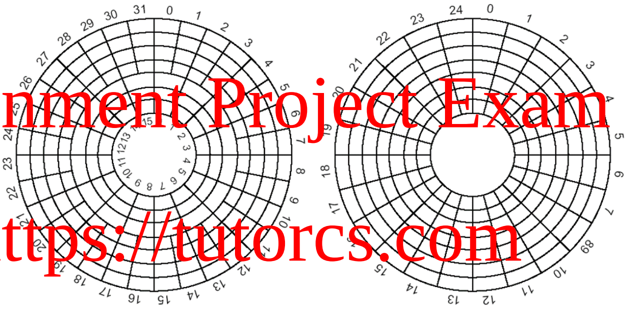


Tracks and Cylinders



Parameter	IBM 360 KB floppy disk	Seagate Barracuda ST3400832AS
No. of cylinders	40	16,383
Tracks/cylinder	2	16
Sectors/track	9	63
Bytes/sector	512	512
Sectors/disk	720	781,422,768
Disk capacity	360 KB	400 GB

http://disctech.com/Seagate-ST3400832AS-SATA-Hard-Drive



(a) Physical geometry (b) Virtual geometry

Surface divided into 20 or more **zones**

- Outer zones have more sectors per track $\rightarrow$ ensures that sectors have same physical length
- Zones hidden using virtual geometry

Physical hardware address: (cylinder, surface, sector)

- But actual geometry complicated $\rightarrow$ hide from OS

Modern disks use logical sector addressing (or logical block addresses LBA)

- Sectors numbered consecutively from 0 … n
- Makes disk management much easier
- Helps work around BIOS limitations
  - Original IBM PC BIOS $\rightarrow$ 8 GB max
  - 6 bits for sector, 4 bits for head, 14 bits for cylinder

Disk capacity statements can be confusing ☺

1 KB $= 2^{10}$ bytes $= 1024$ bytes vs 1 KB $= 10^3$ bytes $= 1000$ bytes

1 MB $= 2^{20}$ bytes $= 1024^2$ bytes vs 1 MB $= 10^6$ bytes $= 1000^2$ bytes

1 GB $= 2^{30}$ bytes $= 1024^3$ bytes vs 1 GB $= 10^9$bytes $= 1000^3$ bytes

If necessary, just make it consistent on the exam ☺

Before a disk can be used, it must be formatted

- **Low level format**
  - Disk sector layout

Preamble	Data	ECC

  - Cylinder skew
  - Interleaving
- **High level format**
  - Boot block
  - Free block list
  - Root directory
  - Empty file system

Amount of cylinder skew depends on the drive geometry

Example Problem

Consider a 10,000 rpm drive with each track having 300 sectors and track to track seek time of 800 μsec

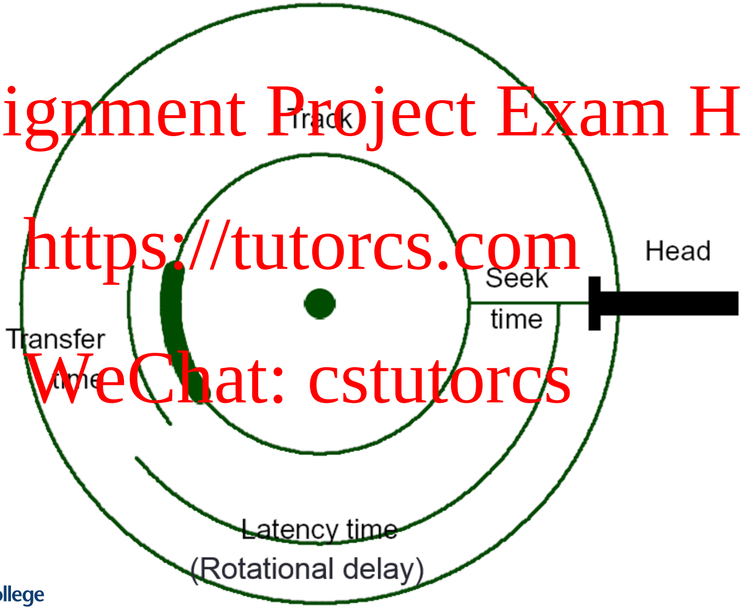
Time taken for 1 rotation $= \frac{60s}{10000} = 6 \text{ x } 10^{-3} = 6$ ms

300 sectors per track $\Rightarrow$ Time taken for 1 sector $= \frac{6ms}{300} = 2 \text{ x } 10^{-5} = 20\ \mu$s

Track to track seek time is 800 μs $\Rightarrow$ Number of sectors that pass in one seek $= \frac{800}{20} = 40$

Hence, cylinder skew $= 40$

Disk Delays I



Disk Delays II

Typical disk

Sector size	512 bytes
Seek time (adjacent cylinder)	$<$ 1 ms
Seek time (average)	8 ms
Rotation time (average latency)	4 ms
Transfer rate	up to 100 MB/s

Disk Scheduling

- Minimise seek and/or latency times
- Order pending disk requests with respect to head position

Seek time $\approx$ 2 – 3 times larger than latency time $\rightarrow$ more important to optimise

Disk Performance

Given

b – number of bytes to be transferred

N – number of bytes per track

r – rotation speed in revolutions per second

Seek time	t_{seek}
Latency time (rotational delay)	$t_{latency} = \frac{1}{2 \times r}$
Transfer time	$t_{transfer} = \frac{b}{N \times r}$
Total access time (t_{access})	$t_{seek} + t_{latency} + t_{transfer}$

Example Problem

Disk Performance

Average seek time: 10ms

Rotation speed: 10,000 rpm

512 byte sectors

320 sectors per track

File size: 2560 sectors (1.25 MB)

Calculate the time taken to:

1. read file stored as compactly as possible on disk (i.e. file occupies all sectors on 8 adjacent tracks $\rightarrow$ 8 tracks x 320 sectors/track = 2560 sectors)
2. read file with all sectors randomly distributed across disk

Example Problem

Answer: Disk Performance

1. Average seek $= 10$ ms

   Latency time $= 3$ ms $= \frac{1}{2 \times (\frac{10000}{60})}$

   Read 320 sectors $= 6$ ms $= \frac{b}{N \times (\frac{10000}{60})}$

   Total (for first track) $= 19$ ms

   Time to read next track $= 3$ ms $+ 6$ ms $= 9$ ms

   Total time $= 19$ ms $+ 7 \times 9$ ms $= 82$ ms $= 0.082$ seconds

2. Average seek and latency time same as above

   Read 1 sector $= 0.01875$ ms $= \frac{512}{512 \times 320 \times (\frac{10000}{60})}$

   Total $= 13.01875$ ms

   Total time $= 2560 \times 13.01875$ ms $= 33.328$ seconds

First Come First Served (FCFS)

No ordering of requests → random seek patterns

- OK for lightly-loaded disks
- But poor performance for heavy loads
- Fair scheduling

Queue: 98, 183, 37, 122, 14, 130, 60, 67 (head starts at 53)

Shortest Seek Time First (SSTF)

Order requests according to shortest seek distance from current head position

- Discriminates against innermost/outermost tracks
- Unpredictable and unfair performance

Queue: 98, 183, 37, 122, 14, 130, 60, 67 (head starts at 53)

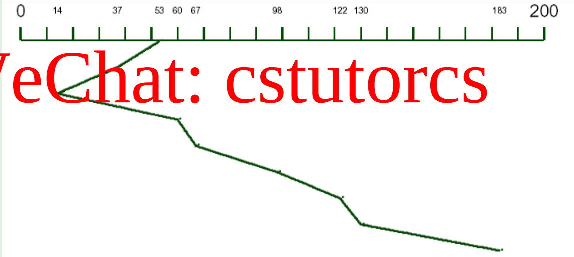
If, when handling request at 14, new requests arrive for 50, 70, 100 → long delay before 183 serviced

SCAN Scheduling

Choose requests which result in shortest seek time in preferred direction

- Only change direction when reaching outermost/innermost cylinder (or no further requests in preferred direction)
- Most common scheduling algorithm (also called elevator scheduling)
- Long delays for requests at extreme locations

Queue: 98, 183, 37, 122, 14, 130, 60, 67 (head starts at 53 and direction is towards 0)



Services requests in one direction only

- When head reaches innermost request, jump to outermost request
- Lower variance of requests on extreme tracks
- May delay requests indefinitely (though less likely)

Queue: 98, 183, 37, 122, 14, 130, 60, 67 (head starts at 53)

N-Step SCAN

As for SCAN, but services only requests waiting when sweep began

- Requests arriving during sweep serviced during return sweep
- Doesn't delay requests indefinitely

Queue: 98, 183, 37, 122, 14, 130, 60, 67 (head starts at 53; direction $\rightarrow$ 0); requests 80, 140 arrive when head moving outwards

I/O requests placed in request list

- One request list for each device in system
- **bio** structure: associates memory pages with requests

Block device drivers define request operation called by kernel

- Kernel passes ordered request list
- Driver must perform all operations in list
- Device drivers do not define read/write operations

Some devices drivers (e.g. RAID) order their own requests

- Bypass kernel for request list ordering

Default: variation of SCAN algorithm

- Kernel attempts to merge requests to adjacent blocks
- But synchronous read requests may starve during large writes

Deadline scheduler: ensures reads performed by deadline

- Eliminates read request starvation

Anticipatory scheduler: delay after read request completes

- Idea: process will issue another synchronous read operation before its quantum expires
- Reduces excessive seeking behaviour
- Can lead to reduced throughput if process does not issue another read request to nearby location
    - Anticipate process behaviour from past behaviour

Problem

- CPU performance doubling every 18 months
- Disk performance has increased only 10 times since 1970

Solution

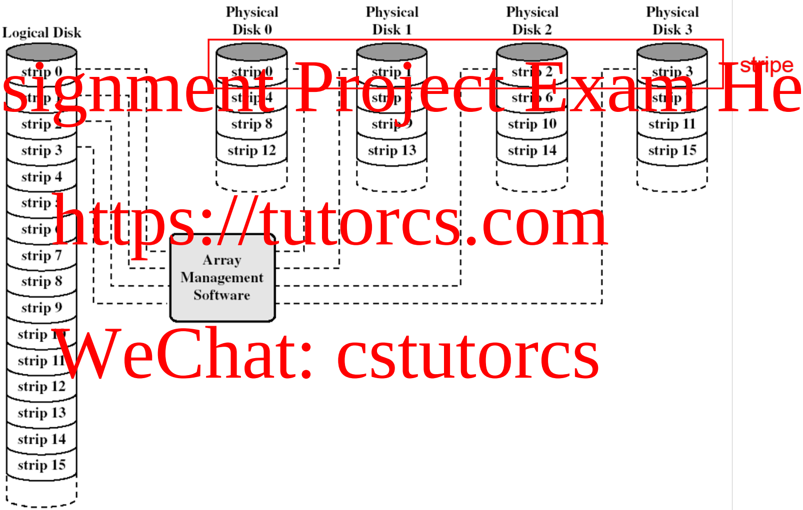
- Use parallel disk I/O → appears to OS as a single disk

RAID (Redundant Array of Inexpensive Disks)

- Array of physical drives appearing as single virtual drive
- Stores data distributed over array of physical disks to allow parallel operation (called striping)

Use redundant disk capacity to respond to disk failure

- More disks → lower mean-time-to-failure (MTTF)



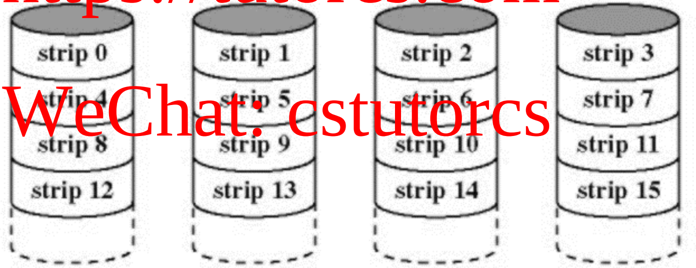
Logical Disk
strip 0
strip 3
strip 4
strip 7
strip 8
strip 11
strip 12
strip 13
strip 14
strip 15
Physical Disk 0
strip 4
strip 8
strip 12
Physical Disk 1
strip 1
strip 13
Physical Disk 2
strip 10
strip 14
Physical Disk 3
strip 3
strip 11
strip 15
stripe
Array Management Software

Use multiple disks and spread out data

Disks can seek/transfer data concurrently

- May also balance load across disks

No redundancy $\rightarrow$ no fault tolerance



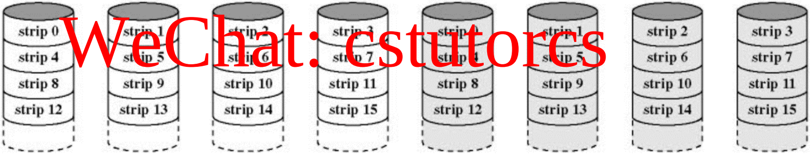
Mirror data across disks

Reads can be serviced by either disk (fast)

Writes update both disks in parallel (slower)

Failure recovery easy

High storage overhead (high cost)



Parallel access by striping at bit-level

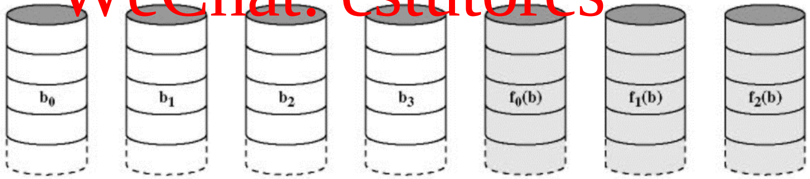
- Use Hamming error-correcting code (ECC)
- Corrects single-bit errors (and detect double-bit errors)

Very high throughput for reads/writes

- But all disks participate in I/O requests (no concurrency)
- Read-modify-write cycle

Only used if high error rates expected

- ECC disks become bottleneck
- High storage overhead



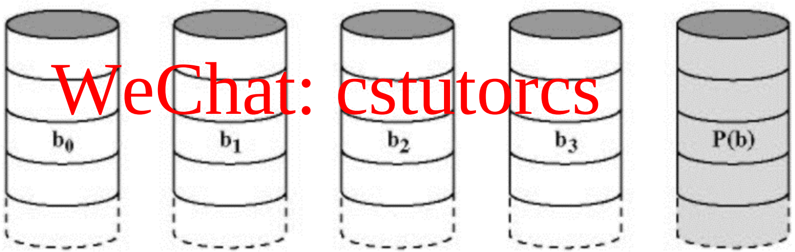
RAID
Level 3 (Byte-level XOR)

Only single parity strip used

- Parity = data1 $\oplus$ data2 $\oplus$ data3 ...
- Reconstruct missing data from parity and remaining data

Lower storage overhead than RAID Level 2

- But still only one I/O request can take place at a time



RAID

Level 4 (Block-level XOR)

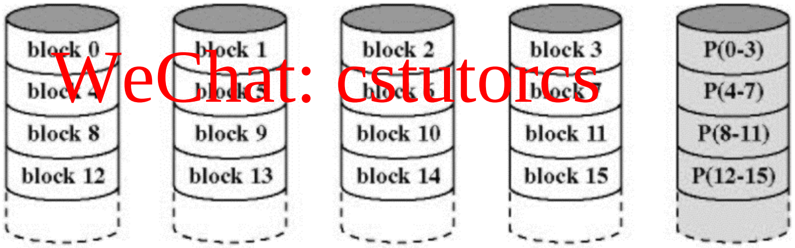
Parity strip handled on block basis

- Each disk operates independently

Potential to service multiple reads concurrently

Parity disk tends to become bottleneck

- Data and parity strips must be updated on each write



RAID
Level 5 (Block-level Distributed XOR)

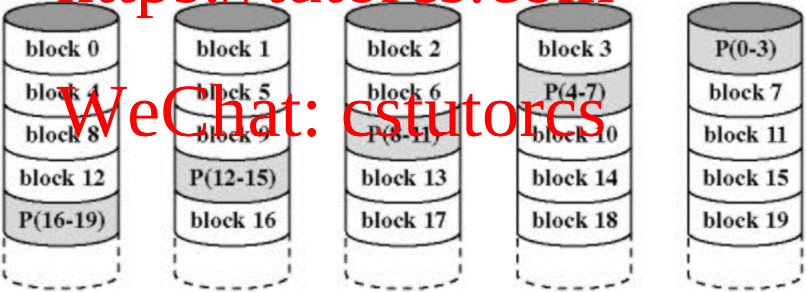
Like RAID Level 4, but distribute parity

- Most commonly used

Some potential for write concurrency

Good storage efficiency/redundancy trade-off

- Reconstruction of failed disk non-trivial (and slow)



Category	Level	Description	I/O Data Transfer (R/W)	I/O Request rate (R/W)
Striping	0	Non-redundant	+/+	+/+
Mirroring	1	Mirrored	+/0	+/0
Parallel access	2	Redundant via Hamming code	++/++	0/0
	3	Bit interleaved parity	++/++	0/0
Independent access	4	Block interleaved parity	+/-	+/-
	5	Block interleaved distributed parity	+/-	+/- or 0

better than single disk (+) / same (0) / worse (-)

Use Piazza for Q & A

Marked coursework will be returned in January

Provide feedback on PG SOLE ☺

Feedback also possible through Mentimeter (94 41 03)

If time permits, possible guest lecture on Security