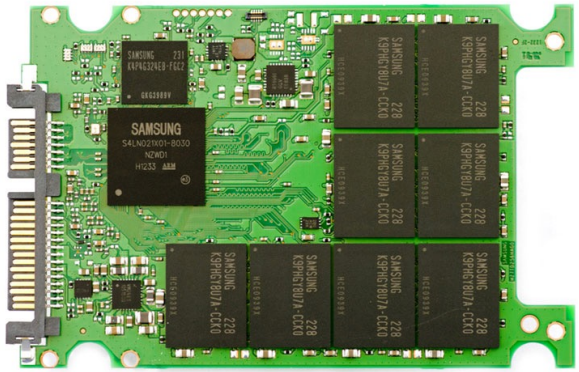
Mass Storage

Chapter 11

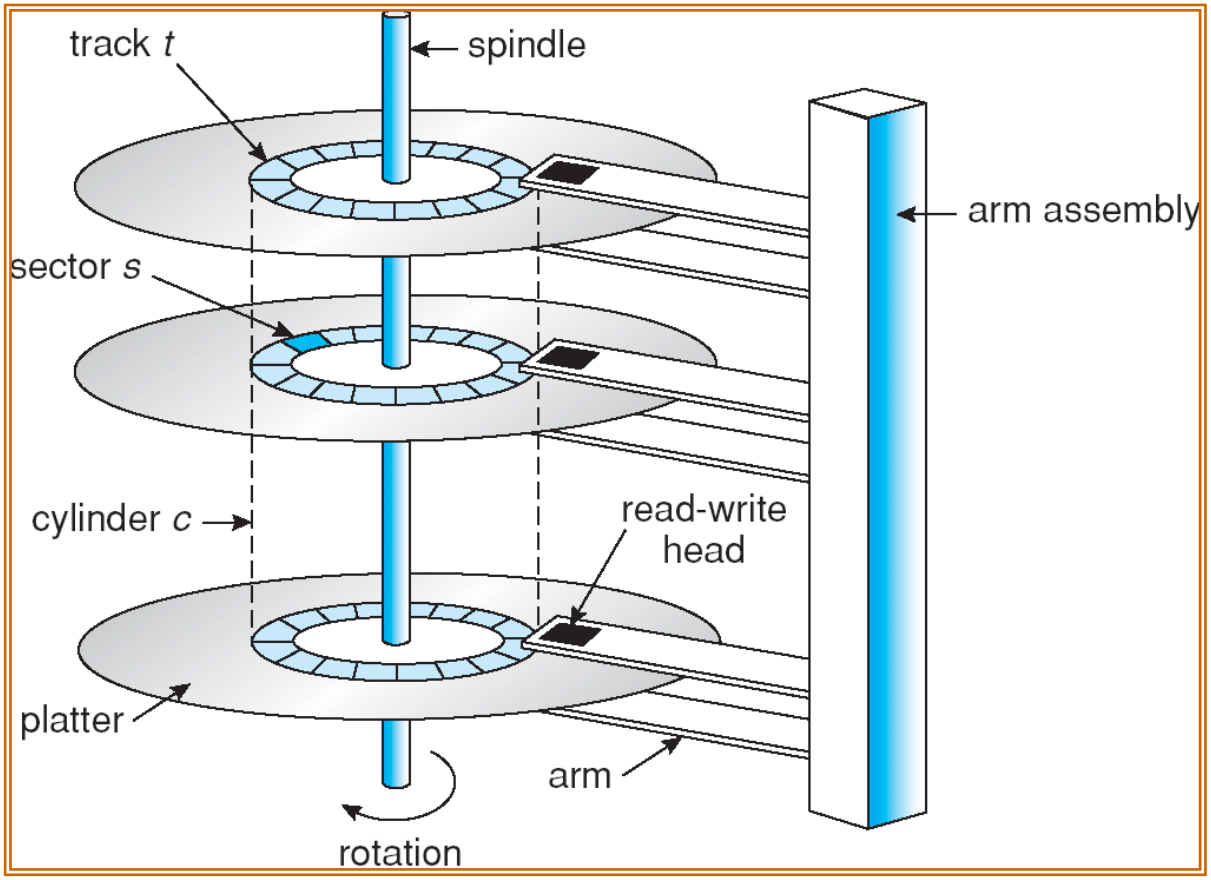
Secondary Storage

- Lots of different technologies are used for secondary storage
  - Magnetic disks (older, slower, larger, cheaper)
  - Solid-State disks (newer, faster, smaller, more expensive)
  - Even RAM disks, use part of your main memory as if it's secondary storage.



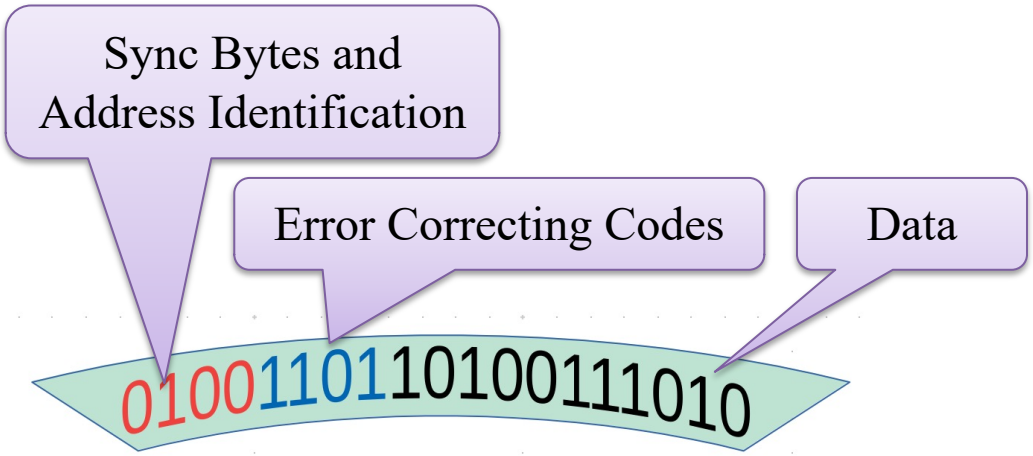
Magnetic Disk Structure

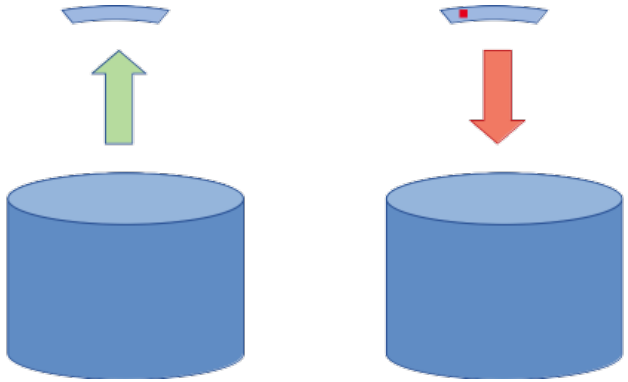
- Still mostly using magnetic disk for secondary storage
- Made of spinning platters and a movable arm assembly
- Organized as sectors, tracks and cylinders



Magnetic Disk Organization

- Physically, a magnetic disk is a large collection of addressable sectors.
  - Read/Write a whole sector at once
  - Sector size typically 512 bytes or larger
  - What if you just need to change one byte?





Magnetic Disk Organization

- Logically, the disk looks like a linear array of blocks
- Blocks are organized to promote locality on the disk.
  - Nearby addresses correspond to nearby disk sectors.
  - Block zero : the first sector of the first track on the innermost cylinder
  - Then, the rest of the sectors on that track …
  - Then the rest of the tracks on that cylinder …
  - Then, the next cylinder, moving toward the edge

Magnetic Disk Access Latency

- *Seek time* : time to move the head to the desired cylinder
  - Initial startup, traversal, settling
- *Rotational latency* : time for the desired sector to rotate under the head (once we reach the right cylinder)
- *Transfer rate* : speed for reading/writing the disk
- Typical parameters:
  - Seek : about 10 – 15 ms
  - Rotational delay : about 4.15 ms for 7200 rpm
  - Transfer rate: 30 MB/s – 100 MB/s

Disk Scheduling

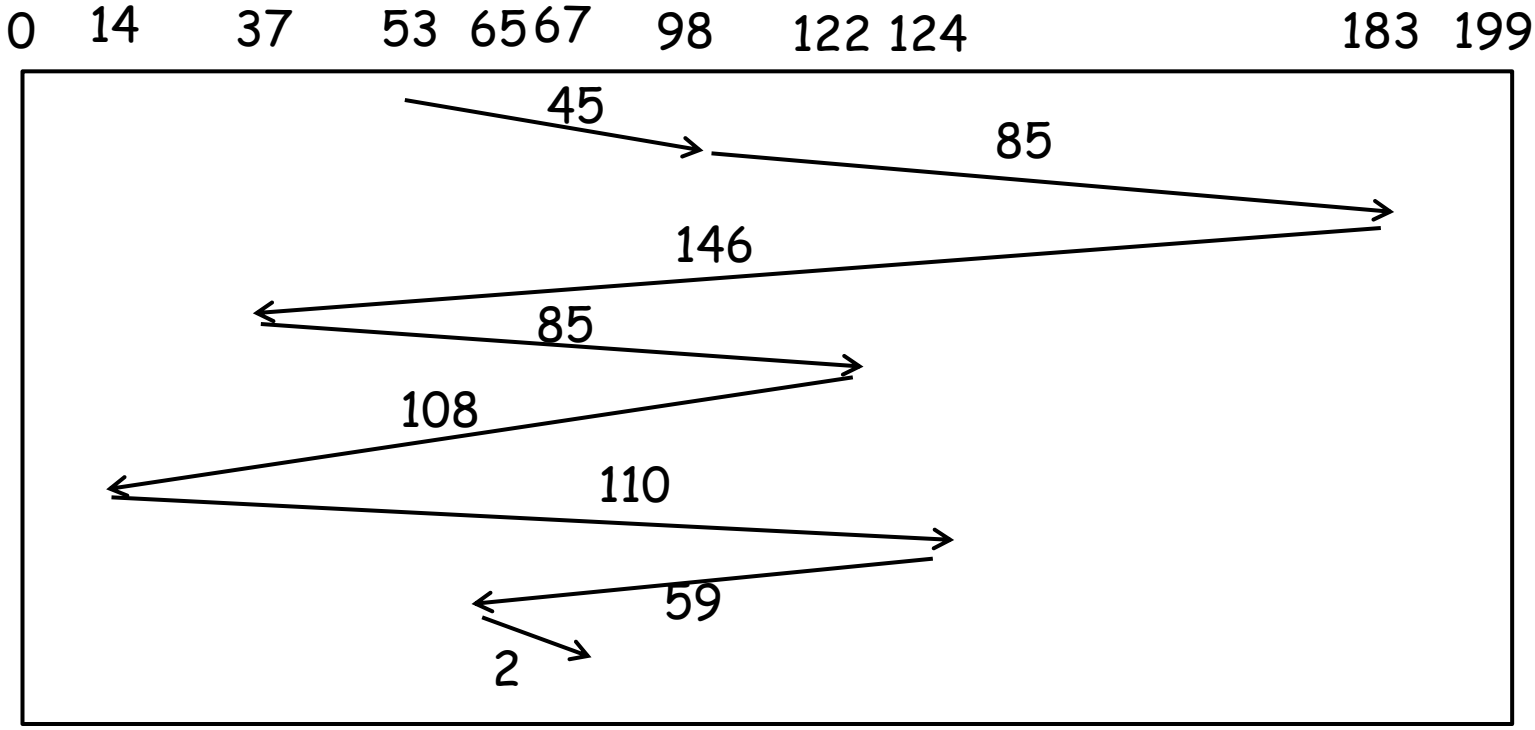
- Disks are much slower than main memory
  - Transfer rate may be more than 100 times slower
  - Most of the cost is just getting to the sector you need
- Access time has two major mechanical components
  - We can't do much about rotational latency
  - But, we can order requests to minimize seek time
  - Assume seek time is proportional to seek distance

Disk Scheduling

- Many *disk scheduling algorithm*s to order requests by cylinder
  - First Come First Served (FCFS)
  - Shortest Seek Time First (SSTF)
  - SCAN
  - LOOK
  - C-SCAN, C-LOOK
- Let's try them out.
  - Pretend the disk queue contains requests for cylinders 98, 183, 37, 122, 14, 124, 64, 67
  - Pretend the disk has cylinders 0 – 199 and the head is on 53

FCFS Example

- Request Queue: 98, 183, 37, 122, 14, 124, 65, 67



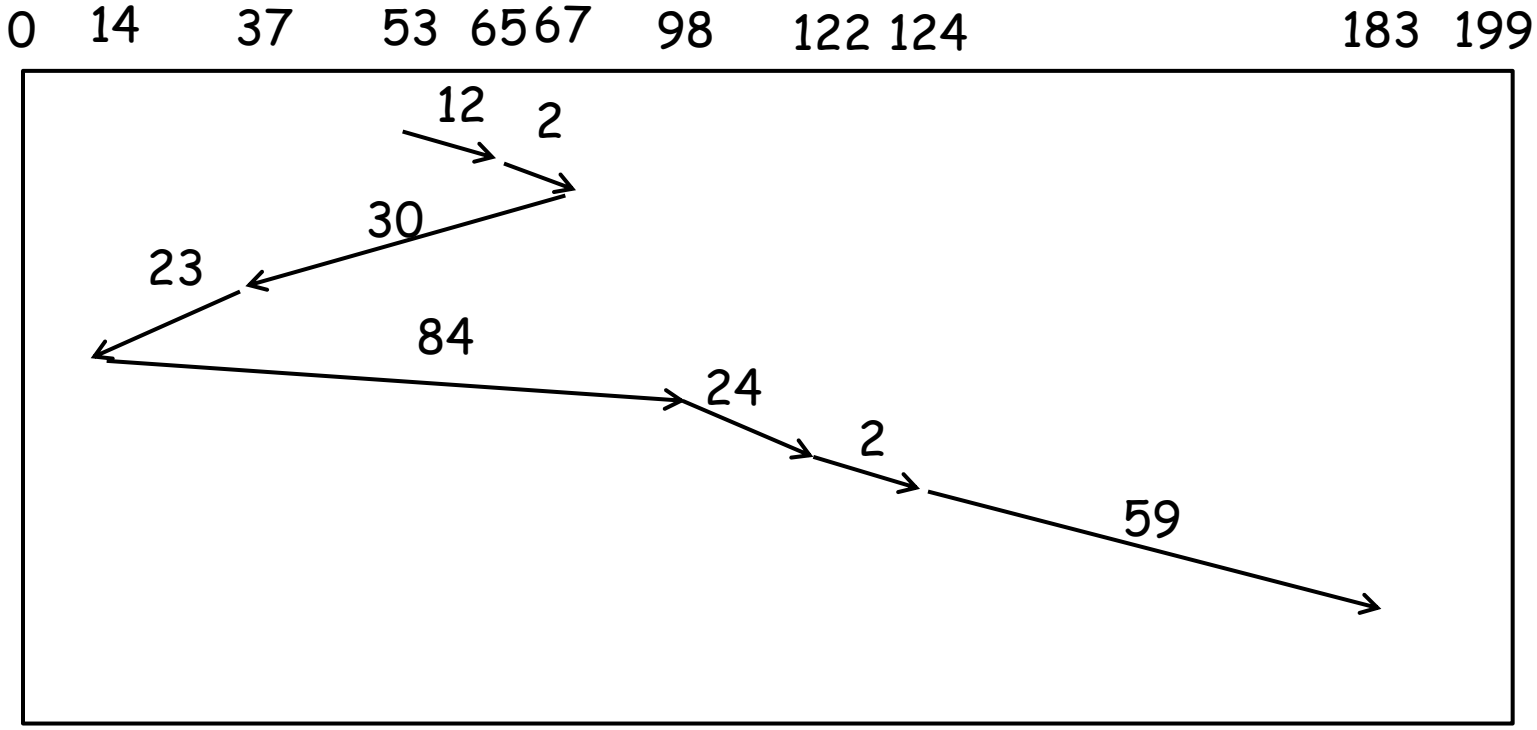
Total Seek Distance = 640

Shortest Seek Time First (SSTF)

- How about a greedy attempt to minimize seek time
- Always select the request with the minimum seek from the current head position
- SSTF is kind of like SJF (but for disk scheduling)
  - Rats! SJF required knowledge of the future
  - Does SSTF require knowledge of the future?
  - Is it optimal with respect to anything?
  - Are there any potential problems with using it?
  - Well, yes. Starvation, but can we fix that with aging.

SSTF Example

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



Total Seek Distance = 236

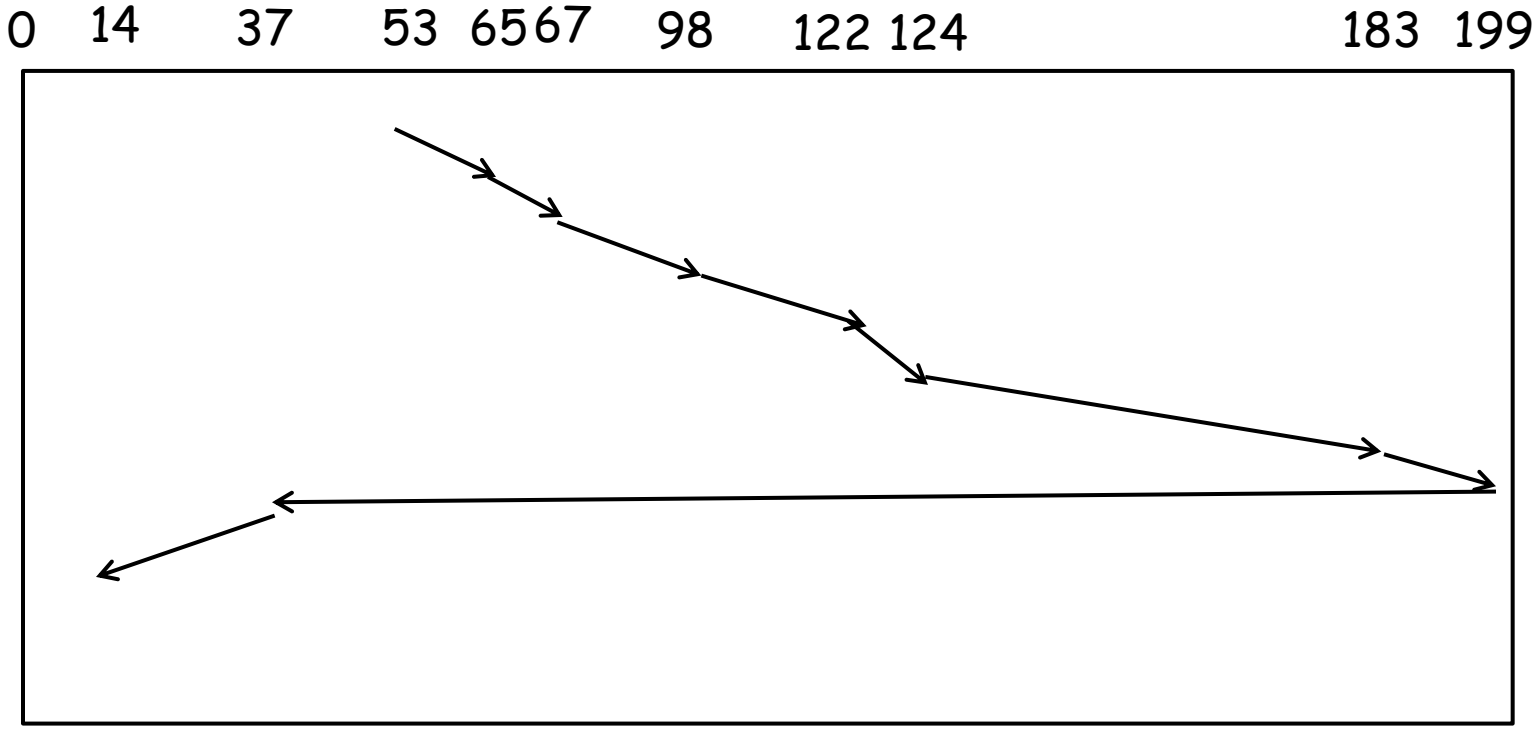
SCAN

- Move the disk arm back and forth, service requests as you go
  - The disk arm starts at one end of the disk (innermost or outermost cylinder)
  - Moves toward the other end, stopping to service requests as it goes
  - Once the head reaches the other end, reverse direction and continue
- Also called the *elevator algorithm*
- Any problems? Risk of starvation?
  - So, with SCAN, everyone (every process) should be happy, right?
  - Well, there's a small risk of starvation, but that's a detail.
  - Also, there are some concerns about fairness, we'll talk about that next

SCAN Example

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

Scan direction



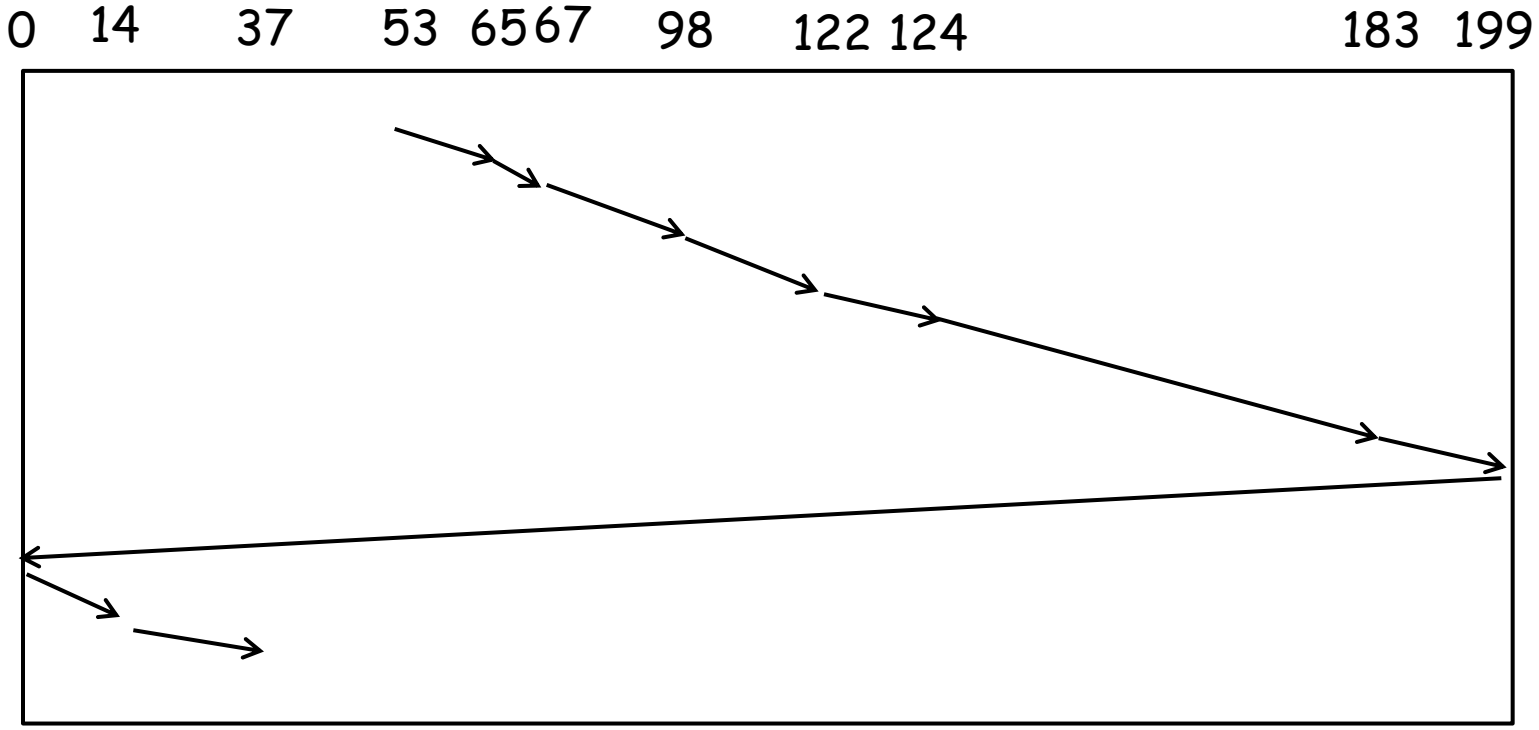
Total Seek Distance = 331

Circular SCAN (C-SCAN)

- Well, with SCAN, some processes may be happier than others
  - Using cylinders near the center ☺
  - Using cylinders near the edge ☹
- Let's fix it, C-SCAN
  - Head starts at one edge
  - Move toward the other edge, servicing requests as it goes
  - When it hits the other edge, run all the way back (without servicing any requests)
  - Start over
- Provides more uniform I/O Wait time than SCAN
- Logically, treats the disk cylinders as a circular list that wraps around from the last cylinder to the first one

C-SCAN Example

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



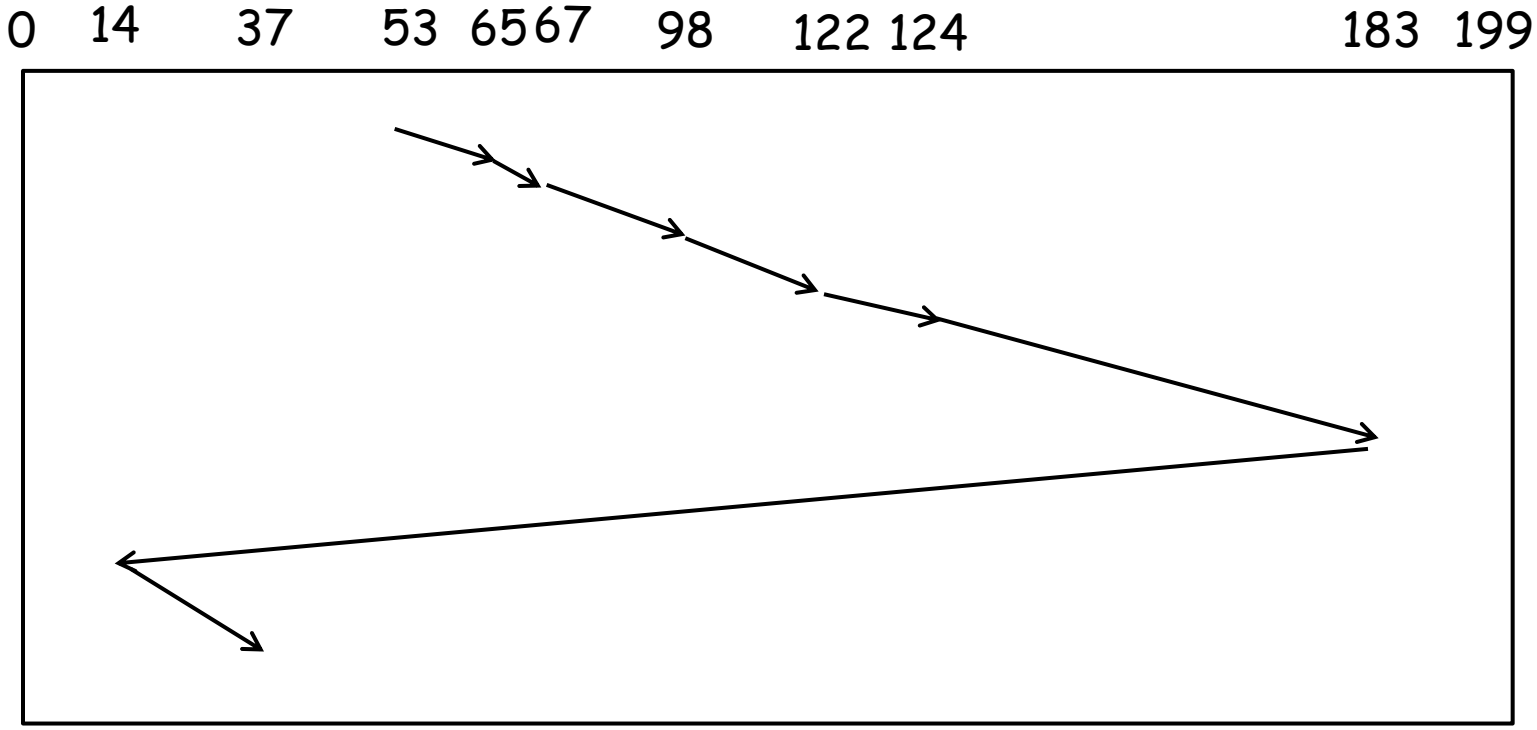
Total Seek Distance = 382

LOOK and C-LOOK

- SCAN is silly
  - Do you really need to go all the way to the edges of the disk (if there are no requests there)?
- We can fix this with LOOK (and C-LOOK)
  - A variant of SCAN
  - The secret: don't go all the way to the edge if you don't need to
  - Stop at the last (innermost or outermost) request
  - Then:
    - For LOOK, reverse directions and resume servicing requests
    - Or, for C-LOOK, go back to the cylinder of the first (e.g., outermost) request

C-LOOK Example

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



Total Seek Distance = 322

Selecting a Disk-Scheduling Algorithm

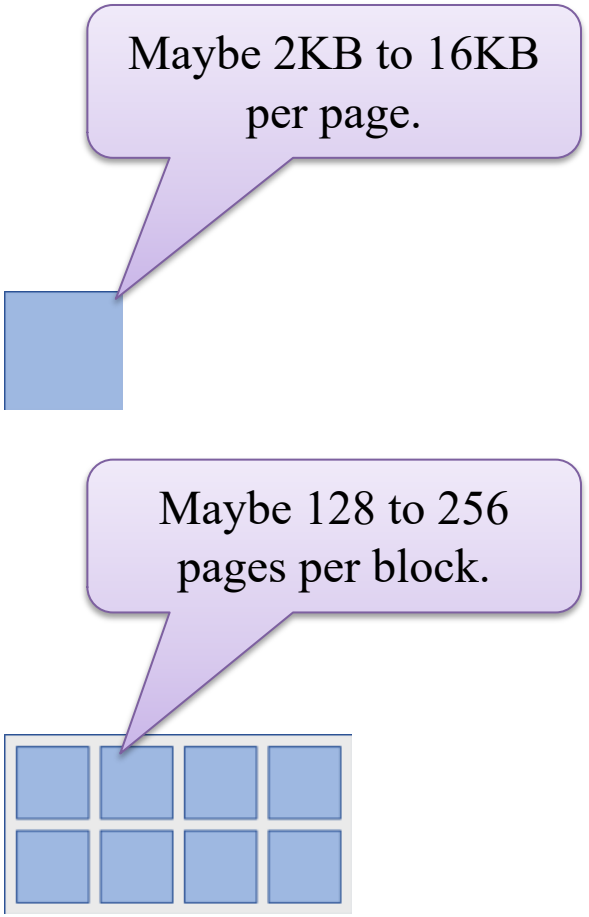
- SSTF is common, it's not optimal, but it does a good job, especially if disk requests are infrequent
- In fact, FCFS works fine if disk activity is low
- SCAN and C-SCAN perform better for systems with lots of disk activity
- Performance depends on number and type of requests
  - How are files stored on the disk?
  - How is backing store implemented?
- Disk scheduling choice should be modular
  - On Linux:
  - cat /sys/block/(disk)/queue/scheduler
  - echo deadline > /sys/block/(disk)/queue/scheduler

Solid-State Drives

- No moving parts
  - So, no seek time, no rotational latency
  - No mechanical wear
- Based on a few different technologies
  - Flash NAND is common

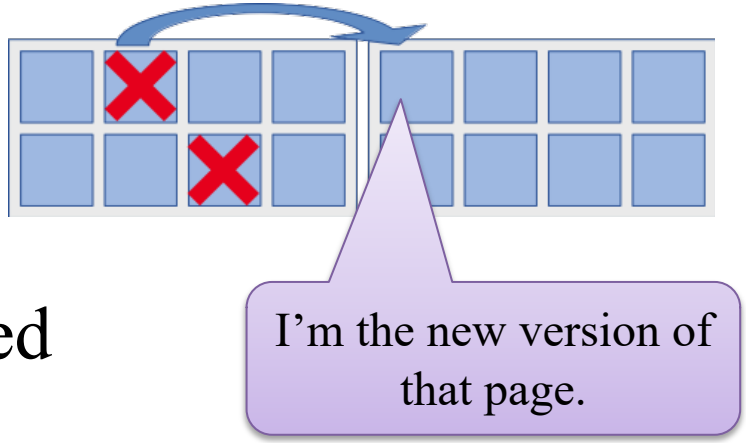
Flash NAND Organization

- Storage consists of fixed-sized regions called *pages*
- Pages can be read (fast) and written (slower), but not overwritten.
- To overwrite a page, it has to be erased (even slower) first.
  - But, multiple pages are organized into *blocks*.
  - A whole block of pages must be erased at once.

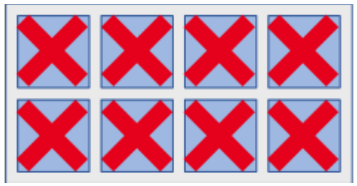


Flash NAND Organization

- To update a page, a new version is written elsewhere.
- … and the old version is marked invalid.
- Device maintains a *flash translation layer* to keep up with where the latest version of each page is.
- Gradually, more and more pages get marked invalid.



Flash NAND Organization



- When all the pages in a block are invalid, it can be erased.
- A garbage collector helps by copying pages out of mostly invalid blocks.
- But, each block has a limited lifetime measured in erasures.
- The device attempts to perform *wear leveling*
  - But, eventually, blocks will start to fail.

Planning for Failure

- Disks can fail
  - Just a few faulty sectors after manufacture
  - Failure of individual sectors or blocks as they are used.
  - Failure of the entire disk after a few years of use
- How can we tell if there is a failure?
- What can we do? Store the data elsewhere.
  - *Sector slipping*
  - *Sector sparing*
  - This can be built into the OS, or hidden from the OS by the device.

RAID

- Redundant Array of Independent Disks (RAID)
  - Try to get the best features out of multiple drives
  - Set of physically distinct drives, logically viewed as one
  - Replace large-capacity disks with many smaller-capacity ones
    - Improved capacity via multiple disks
    - Improved I/O performance via disk parallelism
  - Data distributed across physical drives
    - In a way that enables simultaneous access
    - In a way that duplicates enough data that we can tolerate individual drive failure.
- Lots of raid configurations, representing design alternatives.

RAID Basics

- Two basic RAID techniques
  - Data striping
    - Store some data on each disk (e.g., some part of every file)
    - Helps to enhance I/O bandwidth via parallel reads/writes
  - Data redundancy
    - Store duplicate data across multiple disks
    - Helps to enhance reliability
    - Mirroring, error-correcting encodings, parity

Data Striping

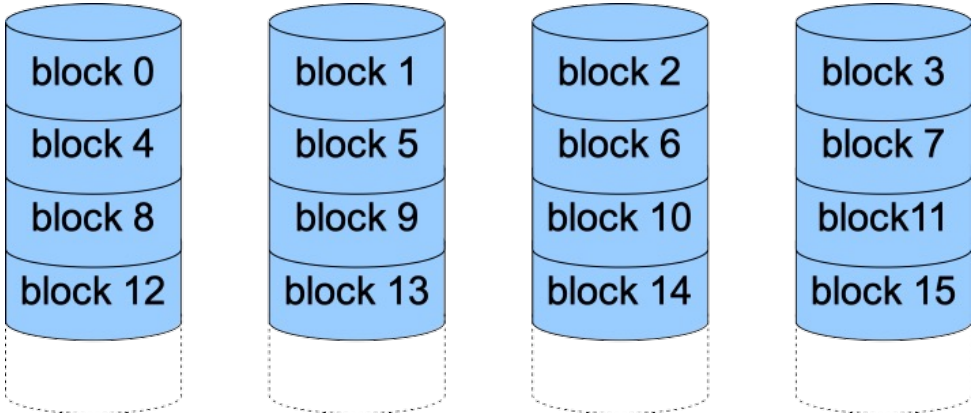
- Alternatives for how we interleave data across the disks
- Granularity of data interleaving
  - Fine grained (bit-interleaved)
    - Small units on each disk
    - Any I/O request will access every disk
    - Only one logical I/O request at a time, but every request will benefit from parallelism
  - Coarse grained (block-interleaved)
    - Large units on each disk
    - Multiple small I/O requests may be served concurrently
    - Large requests can still access all disks in parallel

Data Redundancy

- Alternatives for how we maintain redundant information
  - Just a plain-old copy
  - Or, something fancy:
    Parity, Hamming or Reed-Solomon codes
- Alternatives for where we store the redundant information
  - Concentrate it all on one or a few disks
  - Or, distribute it across all the disks

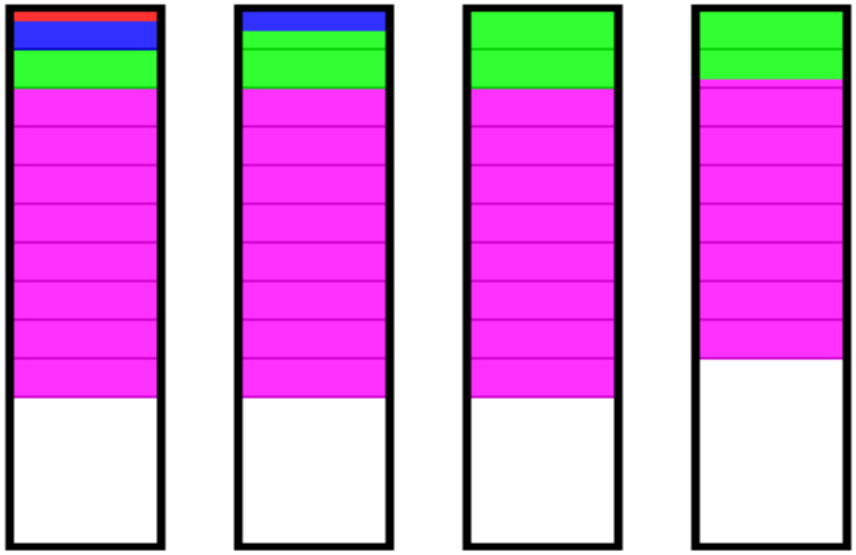
Raid Level 0

- No redundancy, just striping
- Data striped across all the disks
  - Logical storage divided into fixed-sized stripes
  - Stripes mapped round-robin to consecutive disks
- Improved performance, likely to be able to access disks in parallel for:
  - Lots of small reads from multiple processes
  - One large read from a single process



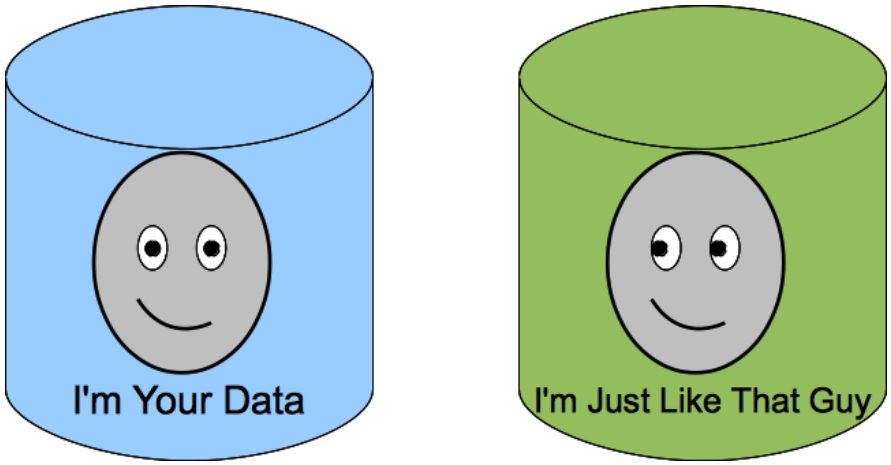
Raid Level 0 Example

- Pretend we have a 16 KB stripe size
- Let's say we have 4 files, all *contiguously allocated* to the array
  - A 4KB red file
  - A 20KB blue file
  - A 100KB Green file
  - A 500KB Magenta file



Raid Level 1

- Use *mirroring* for data redundancy, just keep two copies of every disk
- That's good, right?
  - We can have parallel reads, read from either copy
  - Of course, writes have to update both disks, so that limits parallelism
  - You can afford to lose any disk, but your cost per byte is twice as high

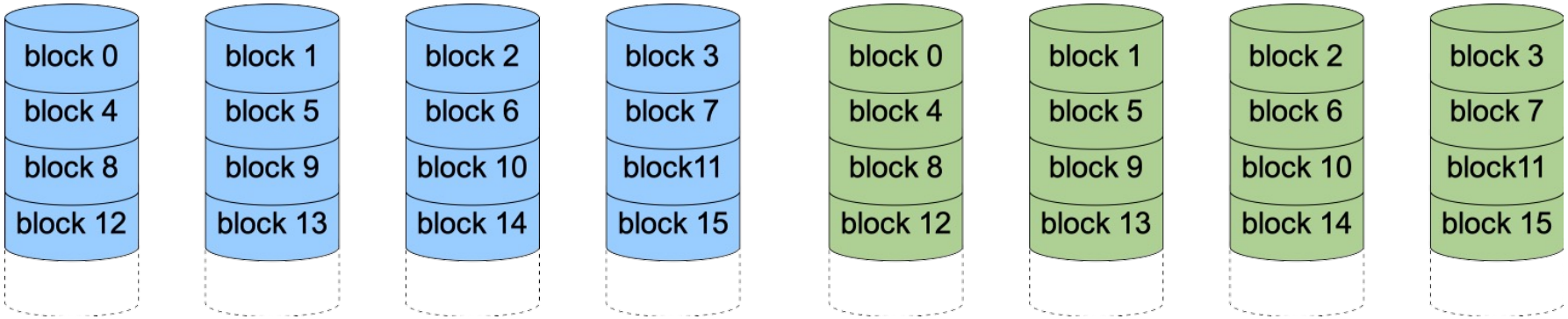


Fun with Failure Probability

- Let's pretend
  - Expected time to failure for a disk: 500 days ☹
  - Chance of a disk failing today: 1 / 500
  - Let's always keep two copies
  - Chance of two disks failing today: 1 / 500 * 1 / 500
  - That's an expected time to 2-disk failure of 250,000 days = 685 years ☺
- But, there's an assumption of independence here

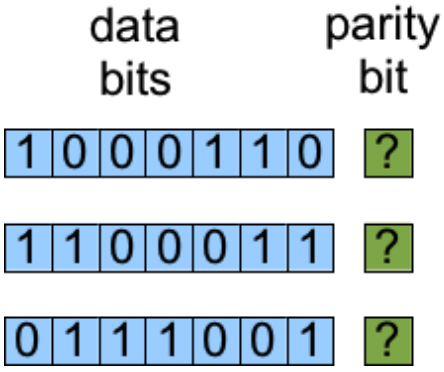
Raid 0+1

- If mirroring is good for fault tolerance …
- … and striping is good for performance
- Has the advantages of both, but storage costs twice as much.



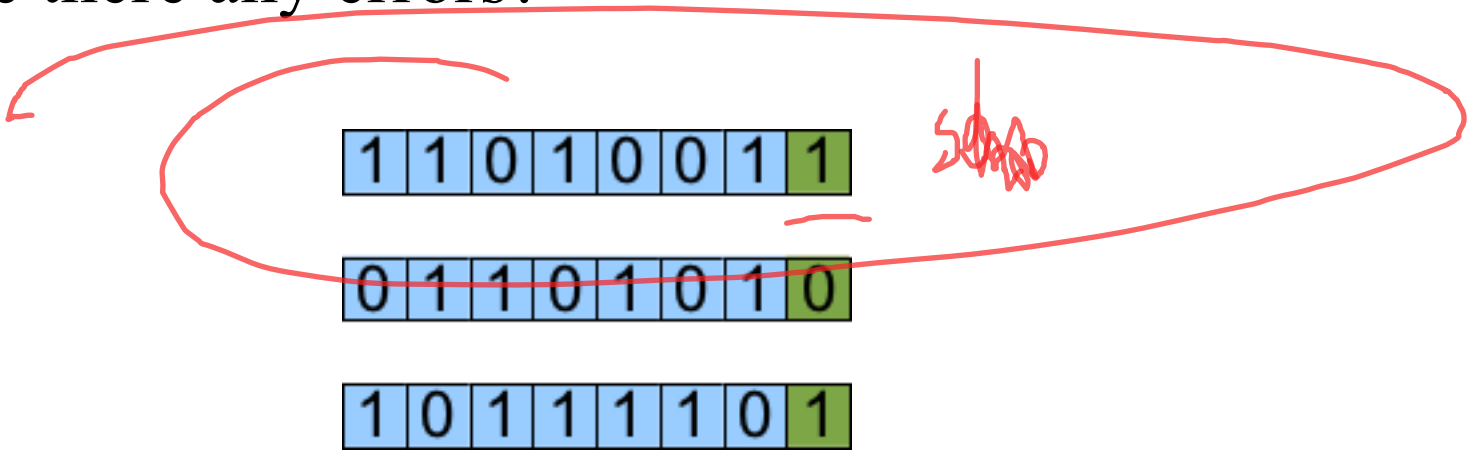
Error Detection / Correcting Codes

- We'll consider two mechanisms
  - Parity
  - Hamming codes (well, a particular hamming code)
- Parity: add an extra, redundant bit to help detect errors
  - Set the extra bit based on the values of the others
  - Even parity : want an even number of ones (across all bits, including parity)
  - Odd parity : want an odd number of ones
  - So, we can tell if just one bit changes (even if it's the parity bit)



Fun with Error Detection

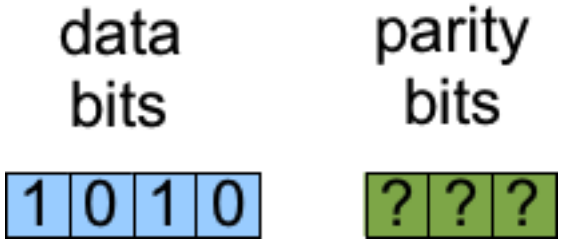
- Consider the following cases
  - 7 Data bits, one parity bit
  - Pretend we're using even parity
  - Are there any errors?



  - Can you tell which bits are bad?

Error Detection / Correcting Codes

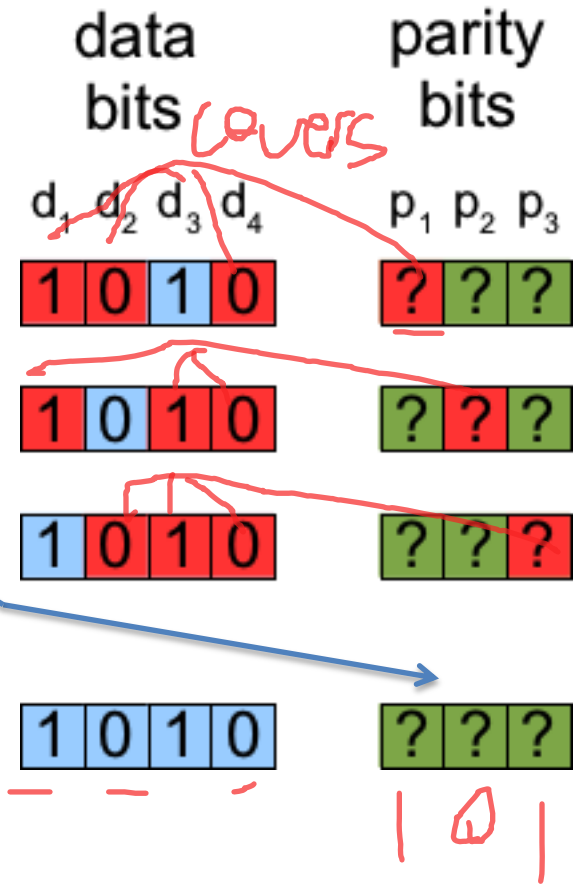
- Hamming codes
  - A general technique for building error correcting codes
  - We'll consider one case
    - Four data bits, three parity bits
    - Can correct single-bit errors, in any bit



Error Detection / Correcting Codes

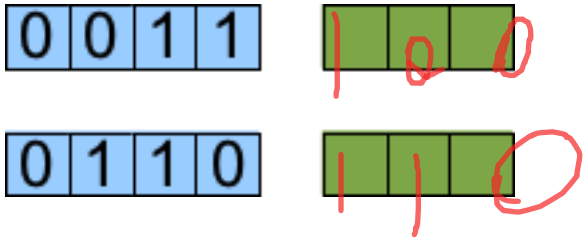
- Each parity bit covers a (different) subset of the data bits
- So, what goes here (under even parity)?

- Things to notice:
  - Every data bit is covered by a different combination of two or more parity bits
  - So, if two or more parity bits don't match the data, the pattern tells us which data bit is bad
  - What if just one parity bit is wrong?

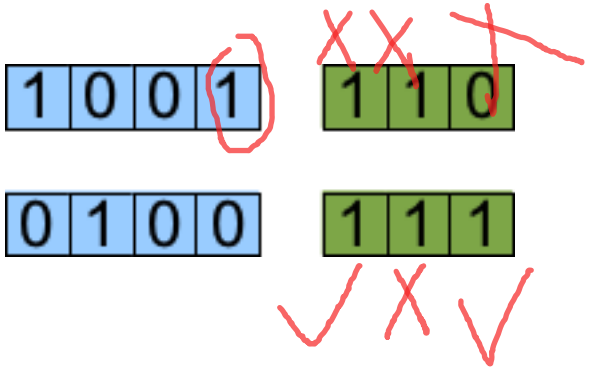


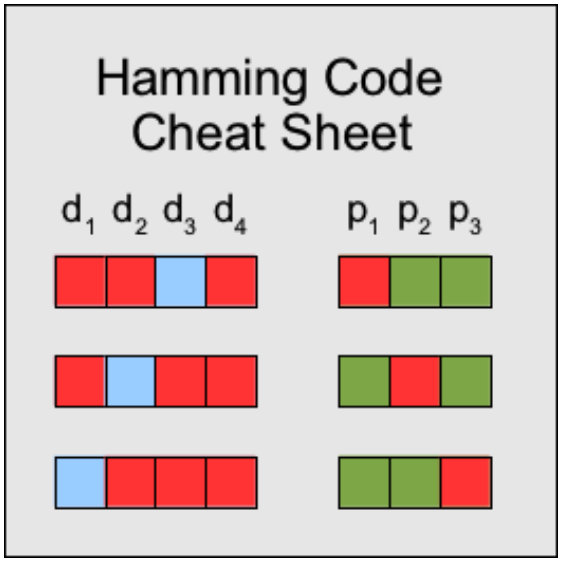
Fun with Hamming Codes

- Fill in the parity bits for the following



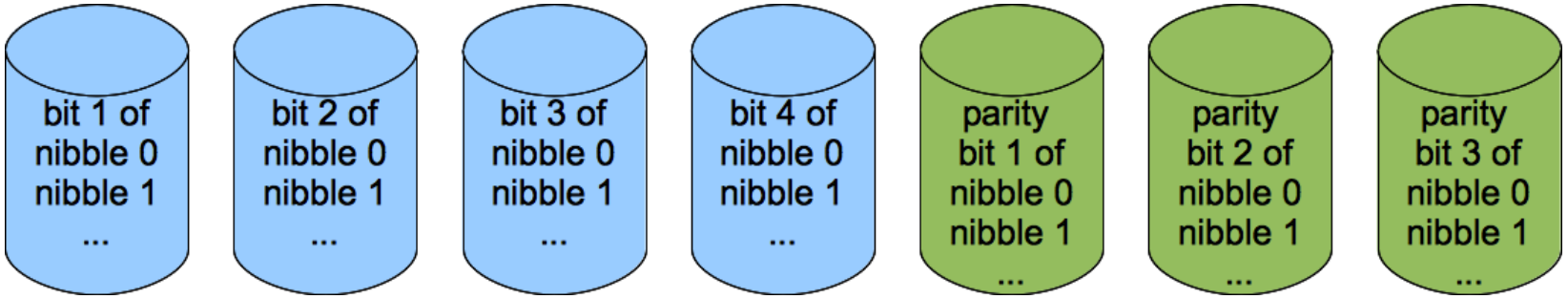
- Are there errors in the following? Where?



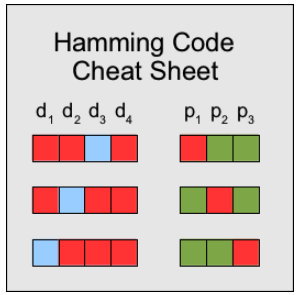


Raid Level 2

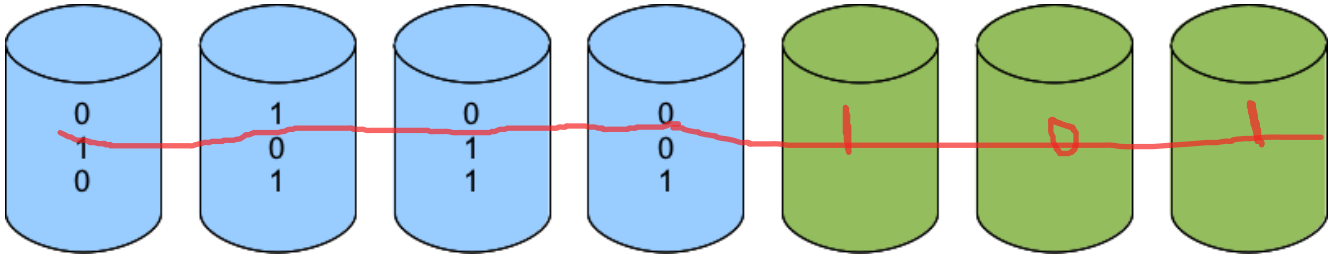
- Levels 2 and 3 are bit interleaved
  - A bit on one disk, the next on the next disk, so-on
  - All disks participate on every read/write.
  - This gives us a guarantee of data parallelism
  - But, just one read/write operation at a time
- Use the bit-level error detection/correction we just covered
- Raid 2, apply an error correction code across all disks
  - For example, use the hamming code we just discussed
  - Four data disks, three parity disks
  - That's a little less overhead than Raid level 1



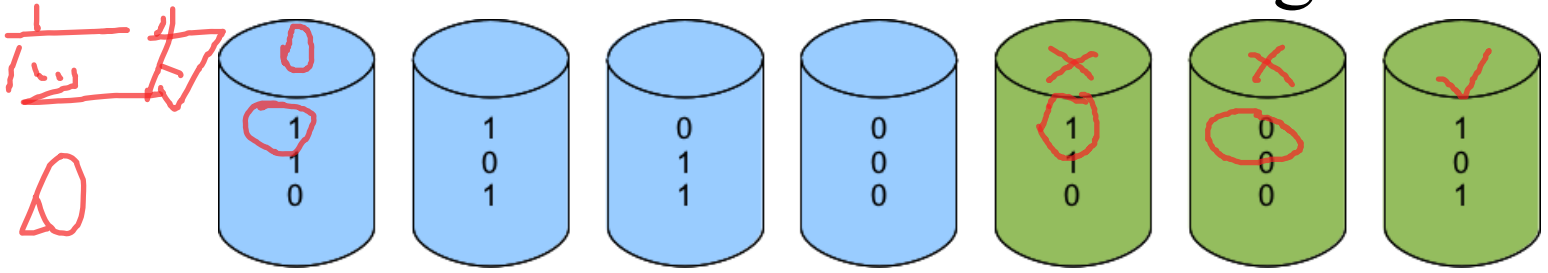
Be the Raid Controller



- Fill in the parity bits in the following disks



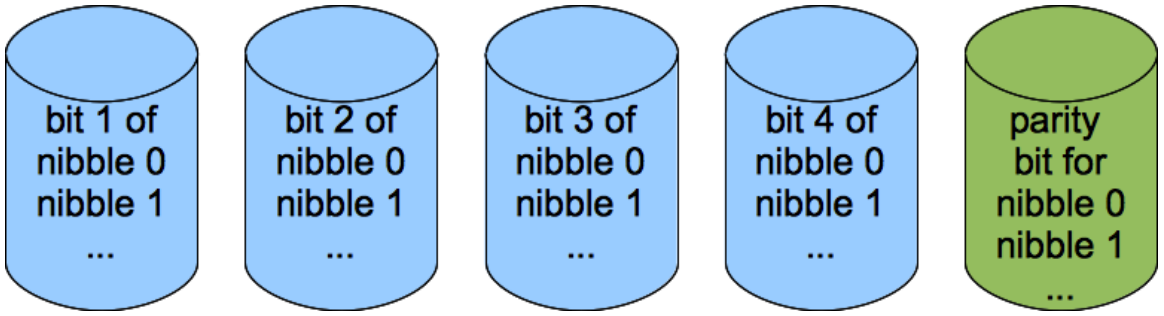
- Correct the errors in the following data



- See, we can correct any one bit (per nibble), on any disk

Raid Level 3

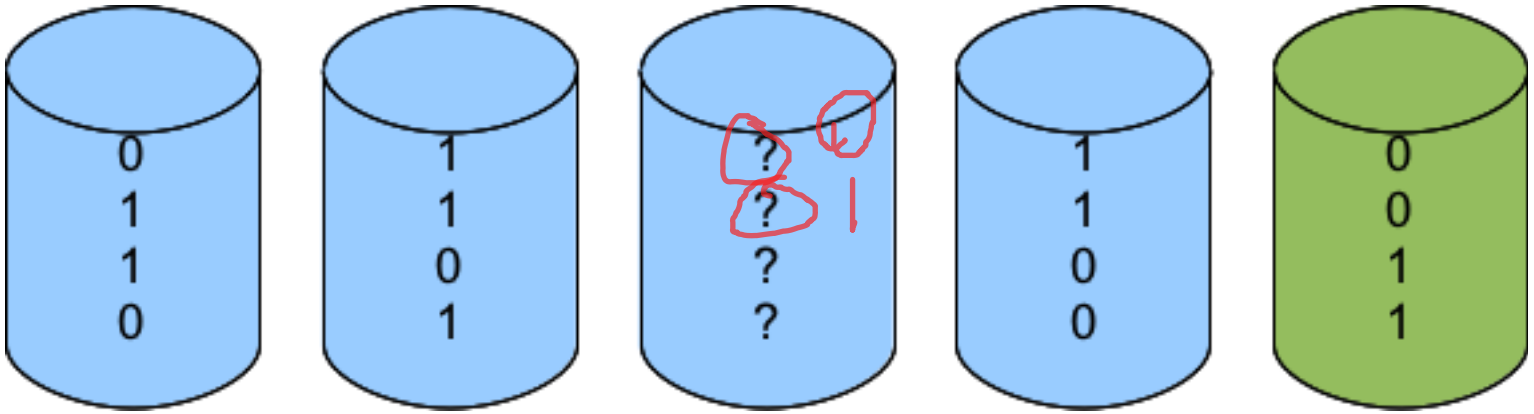
- But, disks don't fail like this, at least not as much
- What if we lose a whole disk at a time?
  - That's actually easier to deal with, we know were the errors/missing data are.
- Raid level 3
  - Bit-interleaved data, one parity bit for the whole array
  - Tolerates failure of a single, whole disk
  - Less storage overhead than level 2



Be the Raid

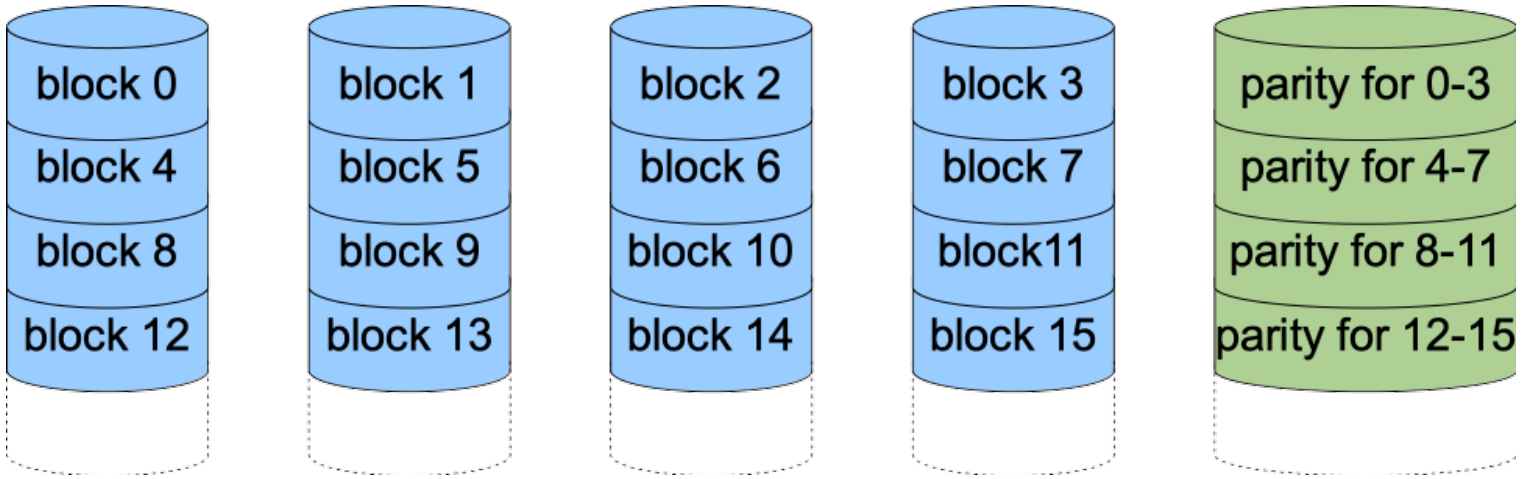
- Restore the missing data
  - Assume even parity was used

Even parity rule



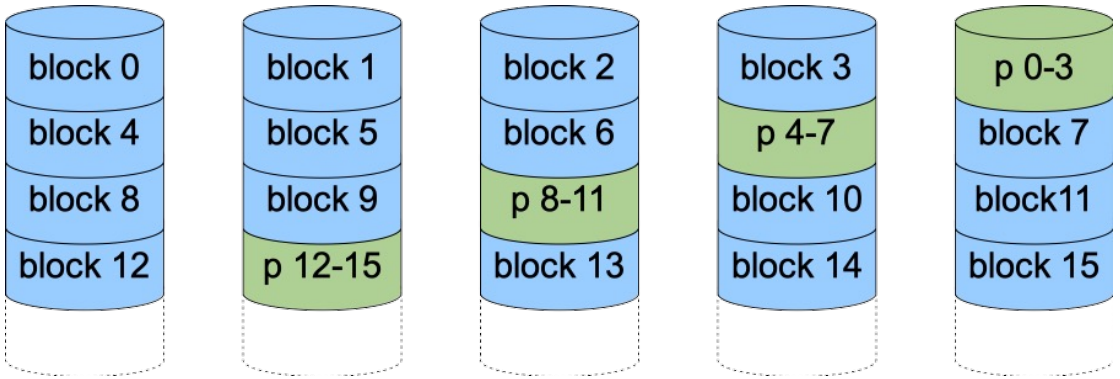
Raid Levels 4 - 6

- Levels 4 – 6 permit independent disk access
  - Block interleaved data (not bit-by-bit)
  - Block 0 on first disk, then block 1 on the next and so on
  - Multiple I/O requests can be done in parallel
    - Maybe I need disk 2 while you need disk 0, etc.
- Level 4
  - Large stripe, with all parity on a single disk
  - Good for concurrent reads
  - Not so much for concurrent writes, they all have to update the single parity disk



Level 5 Distribute the Parity

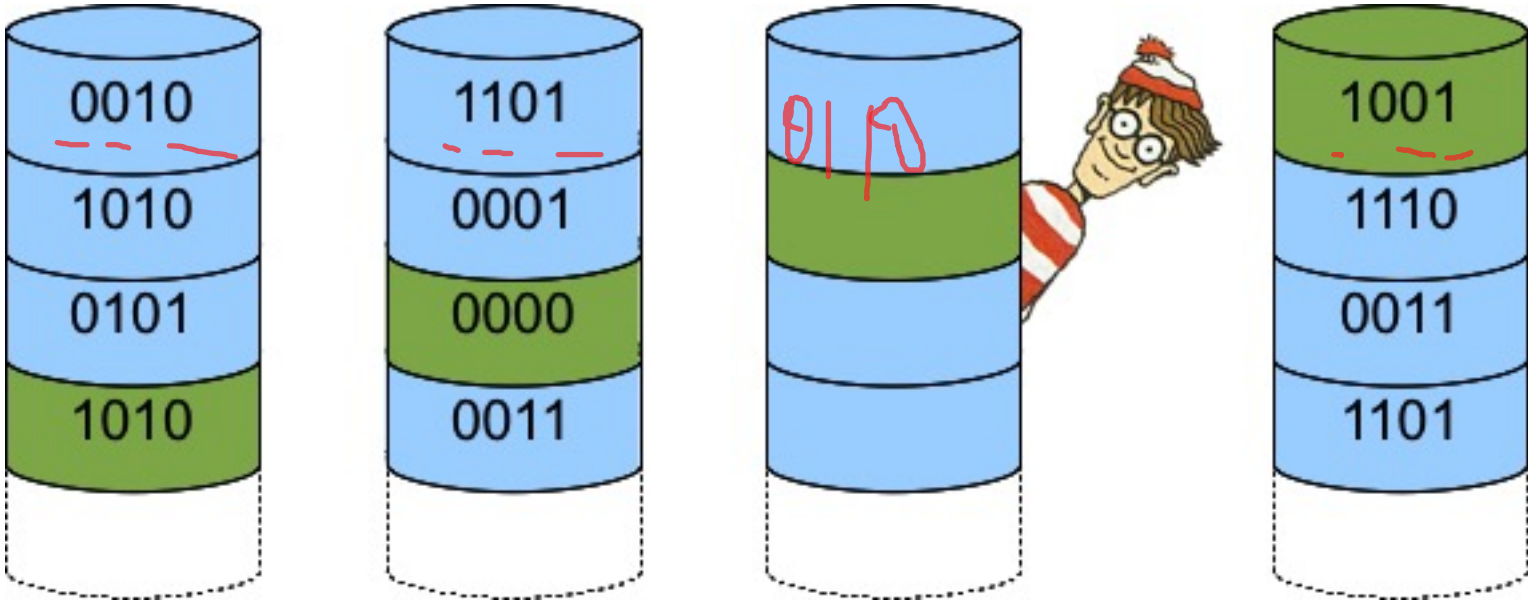
- Level 5, parity stripes distributed among disks



- More opportunities for concurrency
  - Processes P_1 and P_2 can read concurrently (e.g. blocks 2 and 10)
  - They may even be able to write concurrently (e.g., blocks 4 and 9)

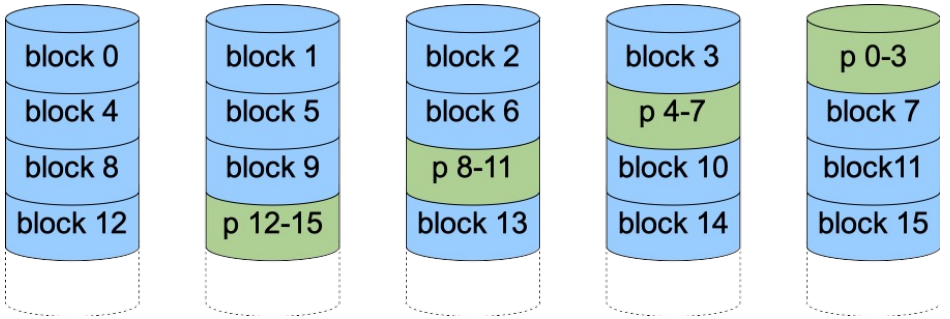
Be the Raid

- Consider Raid Level 5
  - Assume even parity was used
  - Find the missing data



Failure Probability Revisited

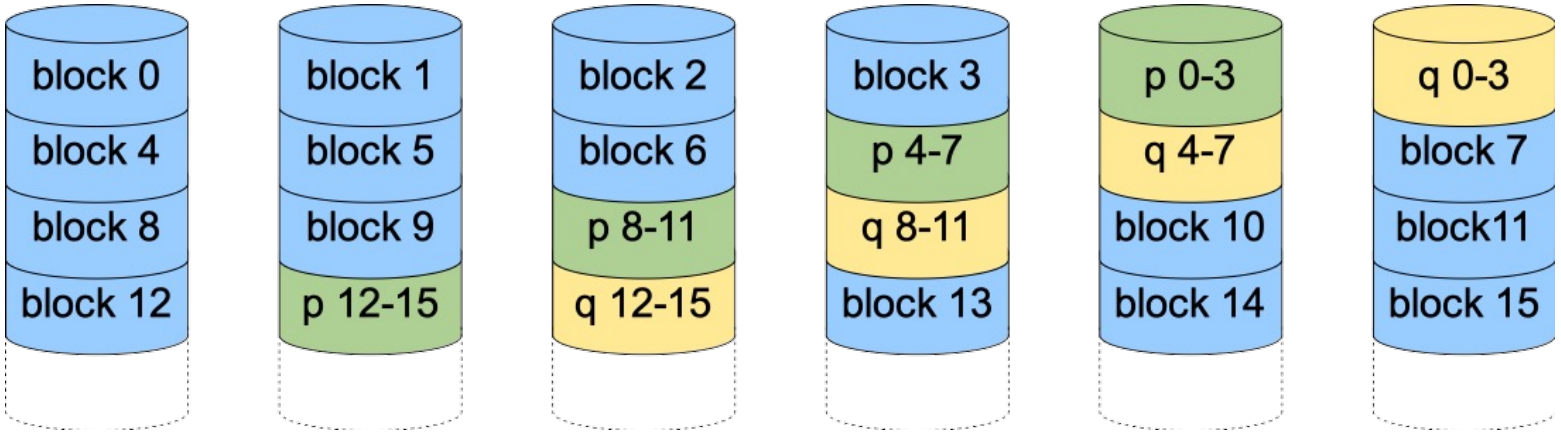
- So, we can cover any number of disks with just one extra disk, right?



- No so fast. More disks means more ways they can fail
  - Disks 1 and 2 could fail on the same day, or disks 3 and 4 or disks 2, 4 and 5, etc.
  - With our previous assumptions, this configuration has a mean time to data loss of just 68.7 years

Level 6 : Additional Parity Bits

- What can we do?
  - Store more redundant information
  - Raid level 6 stores two (different) redundant blocks per stripe
  - We can afford to lose any two disks
  - Now, mean time to data loss is up to 17,000 years



Raid Highlights

- Seven disk organizations
  - But, nobody uses level 2
  - Tradeoff, make one operation really fast, or permit multiple operations in parallel
  - Different schemes for managing redundant information.

What We've Learned

- Secondary storage is implemented:
  - Via different types of devices
  - With different performance, size and volatility characteristics.
  - Logically, the OS sees all these the same way, as an array of fixed sized blocks.
- We need to plan for eventual failure of storage devices fail
  - This is done internally by the device.
  - Raid gives us good ways to increase performance and recover from catastrophic failure.