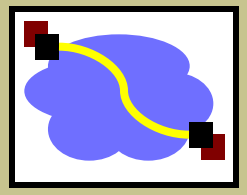
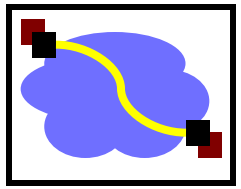


416 Distributed Systems

Errors and Failures

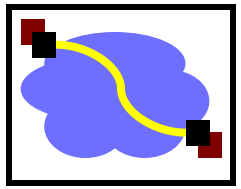
Oct 16, 2018

Types of Errors



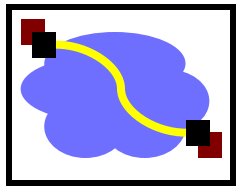
- **Hard errors**: The component is dead.
- **Soft errors**: A signal or bit is wrong, but it doesn't mean the component must be faulty
- Note: You can have recurring soft errors due to faulty, but not dead, hardware

Examples



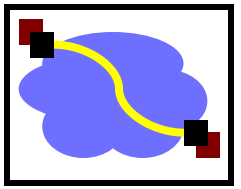
- DRAM errors
  - Hard errors: Often caused by motherboard - faulty traces, bad solder, etc.
  - Soft errors: Often caused by cosmic radiation or alpha particles (from the chip material itself) hitting memory cell, changing value. (Remember that DRAM is just little capacitors to store charge... if you hit it with radiation, you can add charge to it.)

Some fun #s



- Both Microsoft and Google have recently started to identify DRAM errors as an increasing contributor to failures... Google in their datacenters, Microsoft on your desktops.
- We've known hard drives fail for years, of course. :)

Failures across a million consumer PCs:



TACT: Total Accumulated CPU Time

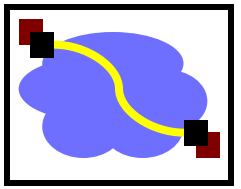
MCE : When CPU issues a machine-check exception (MCE) [Intel], which indicates a detected violation of an internal invariant. Causes include bus errors, microcode bugs, and parity errors in the CPU's caches

Failure	min TACT	Pr[1st failure]	Pr[2nd fail | 1 fail]	Pr[3rd fail | 2 fails]
CPU subsystem (MCE)	5 days	1 in 330	1 in 3.3	1 in 1.8
CPU subsystem (MCE)	30 days	1 in 190	1 in 2.9	1 in 1.7
Memory (DRAM one-bit flip)	5 days	1 in 2700	1 in 9.0	1 in 2.2
Memory (DRAM one-bit flip)	30 days	1 in 1700	1 in 12	1 in 2.0
Disk subsystem	5 days	1 in 470	1 in 3.4	1 in 1.9
Disk subsystem	30 days	1 in 270	1 in 3.5	1 in 1.7

Figure 2. The (conditional) probability of an OS crash from various hardware failures

Millions of consumer PCs; 2016 study
https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/eurosys84-nightingale.pdf

Failures across a million consumer PCs:



TACT: Total Accumulated CPU Time

MCE : When CPU issues a machine-check exception (MCE) [Intel], which indicates a detected violation of an internal invariant. Causes include bus errors, microcode bugs, and parity errors in the CPU's caches

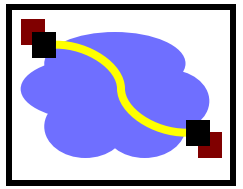
Failure	min TACT	Pr[1st failure]	Pr[2nd fail | 1 fail]	Pr[3rd fail | 2 fails]
CPU subsystem (MCE)	5 days	1 in 330	1 in 3.3	1 in 1.8
CPU subsystem (MCE)	30 days	1 in 190	1 in 2.9	1 in 1.7
Memory (DRAM one-bit flip)	5 days	1 in 2700	1 in 9.0	1 in 2.2
Memory (DRAM one-bit flip)	30 days	1 in 1700	1 in 12	1 in 2.0
Disk subsystem	5 days	1 in 470	1 in 3.4	1 in 1.9
Disk subsystem	30 days	1 in 270	1 in 3.5	1 in 1.7

Figure 2. The (conditional) probability of an OS crash from various hardware failures

Note that failures are not independent!

Millions of consumer PCs; 2016 study
https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/eurosys84-nightingale.pdf

Measuring Availability



- Mean time to failure (MTTF)
- Mean time to repair (MTTR)
- MTBF = MTTF + MTTR (mean time between failure)

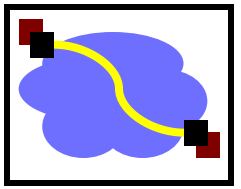
$$Availability = \frac{\text{time system was running}}{\text{time system should have been running}}$$

- Availability = MTTF / (MTTF + MTTR)

$$\text{Down time} = (1 - Availability)$$

  - Suppose OS crashes once per month, takes 10min to reboot.
  - MTTF = 720 hours = 43,200 minutes
    MTTR = 10 minutes
  - Availability = 43200 / 43210 = 0.997 (~“3 nines”)

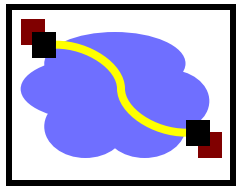
Availability



Availability %	Downtime per year	Downtime per month*	Downtime per week
90% ("one nine")	36.5 days	72 hours	16.8 hours
95%	18.25 days	36 hours	8.4 hours
97%	10.96 days	21.6 hours	5.04 hours
98%	7.30 days	14.4 hours	3.36 hours
99% ("two nines")	3.65 days	7.20 hours	1.68 hours
99.50%	1.83 days	3.60 hours	50.4 minutes
99.80%	17.52 hours	86.23 minutes	20.16 minutes
99.9% ("three nines")	8.76 hours	43.8 minutes	10.1 minutes
99.95%	4.38 hours	21.56 minutes	5.04 minutes
99.99% ("four nines")	52.56 minutes	4.32 minutes	1.01 minutes
99.999% ("five nines")	5.26 minutes	25.9 seconds	6.05 seconds
99.9999% ("six nines")	31.5 seconds	2.59 seconds	0.605 seconds
99.99999% ("seven nines")	3.15 seconds	0.259 seconds	0.0605 seconds

For a reliable component, may have to wait a long time to determine its availability/downtime!

Availability in practice



- Carrier airlines (2002 FAA fact book)
  - 41 accidents, 6.7M departures
  - 99.9993% availability
- 911 Phone service (1993 NRIC report)
  - 29 minutes per line per year
  - 99.994%
- Standard phone service (various sources)
  - 53+ minutes per line per year
  - 99.99+%
- End-to-end Internet Availability
  - 95% - 99.6%

Real Devices

Seagate
We turn on ideas

PRODUCT OVERVIEW

Cheetah 15K.4

Mainstream enterprise disc drive

Simply the best price/ performance, lowest cost of ownership disc drive ever

KEY FEATURES AND BENEFITS

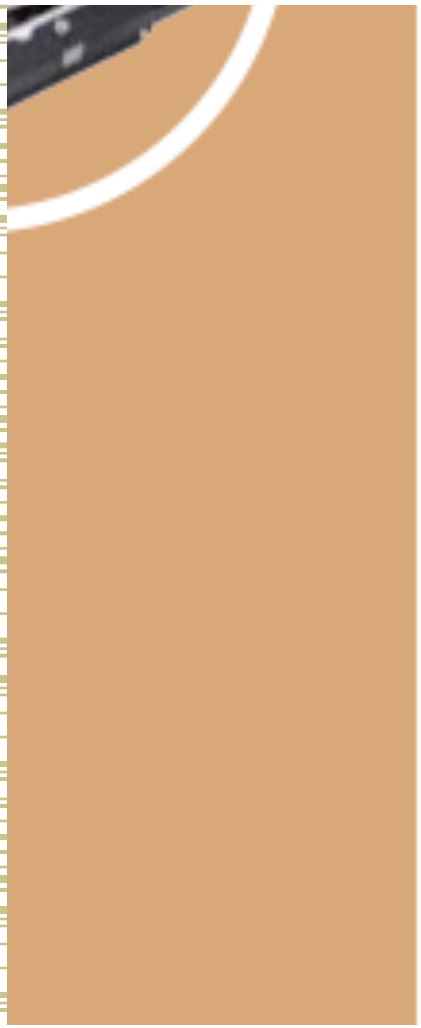
- The Cheetah® 15K.4 is the highest-performance drive ever offered by Seagate®, delivering maximum IOPS with fewer drives to yield lower TCO.
- The Cheetah 15K.4 price-per-performance value united with the breakthrough benefits of serial attached SCSI (SAS) make it the optimal 3.5-inch drive for rock solid enterprise storage.
- Proactive, self-initiated background management functions improve media integrity, increase drive efficiency, reduce incidence of integration failures and improve field reliability.
- The Cheetah 15K.4 shares its electronics architecture and firmware base with Cheetah 10K.7 and Savvio™ to ensure greater factory consistency and reduced time to market.

KEY SPECIFICATIONS

- 146-, 73- and 36-Gbyte capacities
- 3.3-msec average read and 3.8-msec average write seek times
- Up to 96-Mbytes/sec sustained transfer rate
- 1.4 million hours full duty cycle MTBF
- Serial Attached SCSI (SAS), Ultra320 SCSI and 2 Gbits/sec Fibre Channel interfaces
- 5-year warranty

For more information on why 15K is the industry's best price/performance disc drive for use in mainstream storage applications, visit ***http://specials.seagate.com/15k***

Real Devices – the small print



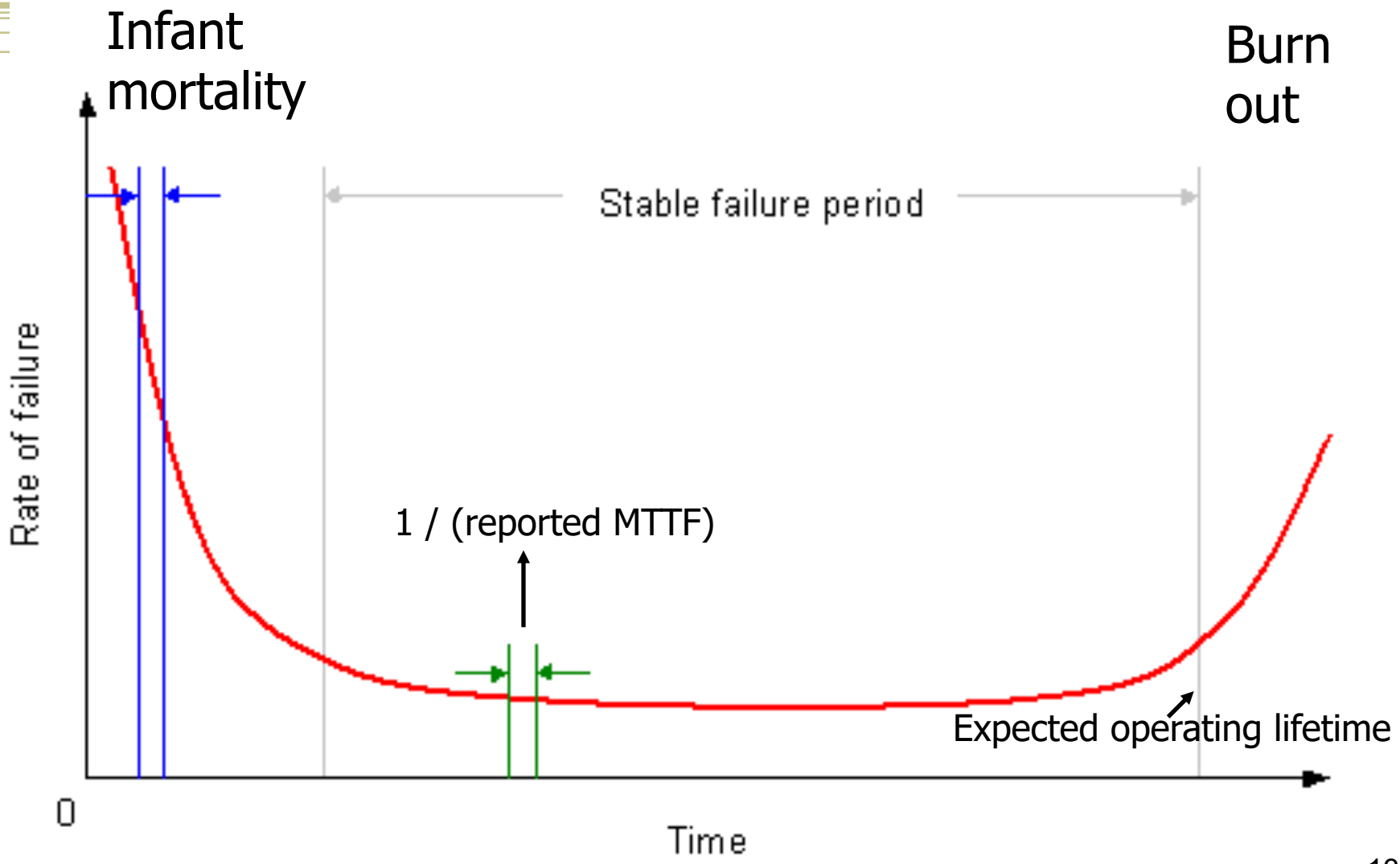
- The Cheetah™ 15K.4 is the highest-performance drive ever offered by Seagate®, delivering maximum IOPS with fewer drives to yield lower TCO.
- The Cheetah 15K.4 price-per-performance value united with the breakthrough benefits of serial attached SCSI (SAS) make it the optimal 3.5-inch drive for rock solid enterprise storage.
- Proactive, self-initiated background management functions improve media integrity, increase drive efficiency, reduce incidence of integration failures and improve field reliability.
- The Cheetah 15K.4 shares its electronics architecture and firmware base with Cheetah 10K.7 and Savvio™ to ensure greater factory consistency and reduced time to market.

KEY SPECIFICATIONS

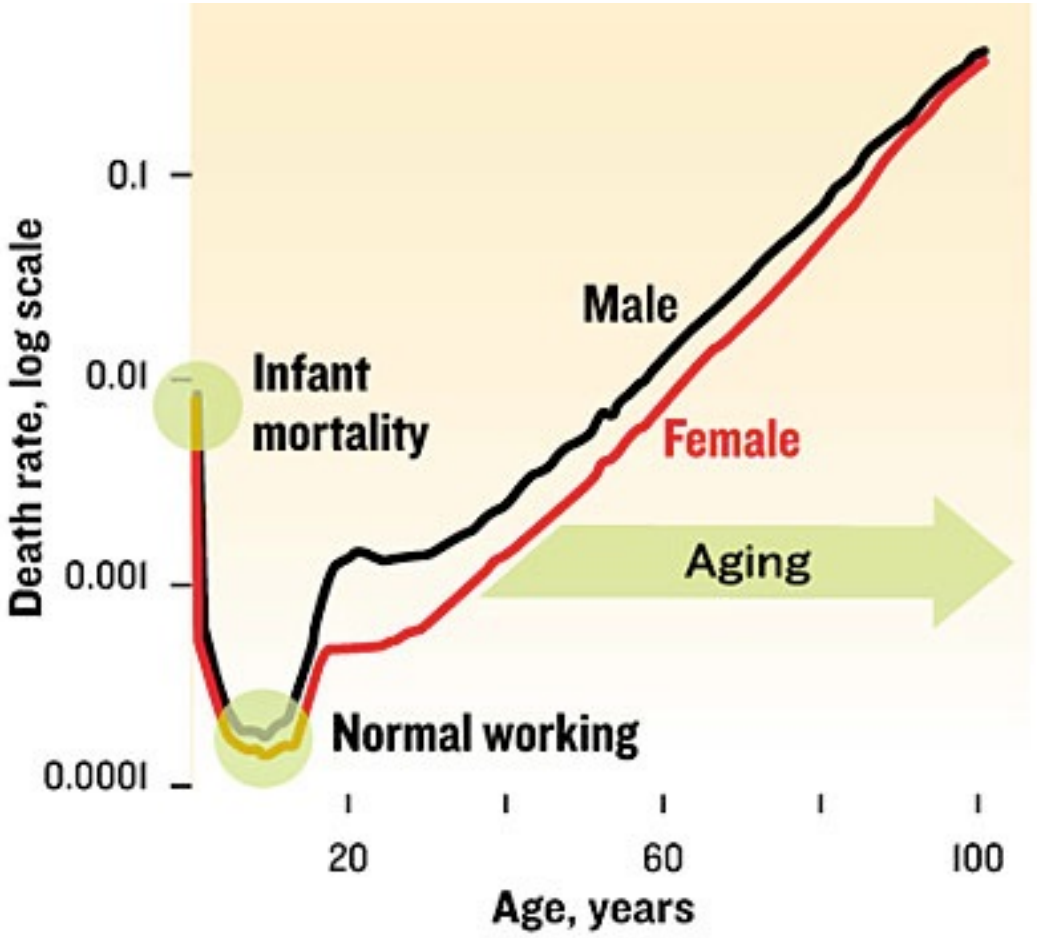
- 146-, 73- and 36-Gbyte capacities
- 3.3-msec average read and 3.8-msec average write seek times
- Up to 96 Mbytes/sec sustained transfer rate
- 1.4 million hours full duty cycle MTBF
- Serial Attached SCSI (SAS), Ultra320 SCSI and 2 Gbits/sec Fibre Channel interfaces
- 5-year warranty

For more information on why 15K is the industry's best price/performance disc drive for use in mainstream storage applications, visit ***http://specials.seagate.com/15k***

Disk failure conditional probability distribution - Bathtub curve



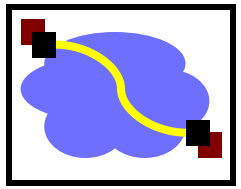
Other Bathtub Curves



Human Mortality Rates (US, 1999)

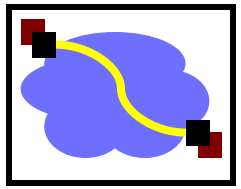
From: L. Gavrilov & N. Gavrilova, "Why We Fall Apart," IEEE Spectrum, Sep. 2004.
Data from http://www.mortality.org

So, back to disks...



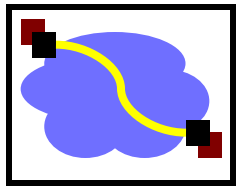
- How can disks fail?
  - Whole disk failure (power supply, electronics, motor, etc.)
  - Sector errors - soft or hard
    - Read or write to the wrong place (e.g., disk is bumped during operation)
    - Can fail to read or write if head is too high, coating on disk bad, etc.
    - Disk head can hit the disk and scratch it.

Coping with failures...



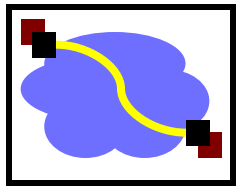
- A failure
  - Let's say one bit in your DRAM fails.
- Propagates
  - Assume it flips a bit in a memory address the kernel is writing to. That causes a big memory error elsewhere, or a kernel panic.
  - Your program is running one of a dozen storage servers for your distributed filesystem.
  - A client can't read from the DFS, so it hangs.

Recovery Techniques



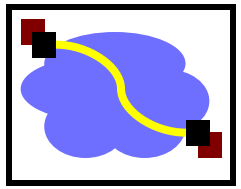
- We've already seen some: e.g., retransmissions in TCP and in your RPC system
- Modularity can help in failure isolation: preventing an error in one component from spreading.
  - Analogy: The firewall in your car keeps an engine fire from affecting passengers
- Redundancy and Retries
  - Later lectures: Specific techniques used in file systems, disks (RAID)
  - This time: Understand how to quantify reliability
  - Understand basic techniques of replication and fault masking

What are our options?



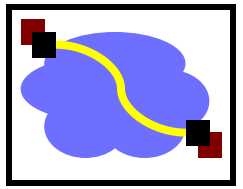
1. Silently return the wrong answer.
2. Detect failure.
3. Correct / mask the failure

Options in dealing with failure

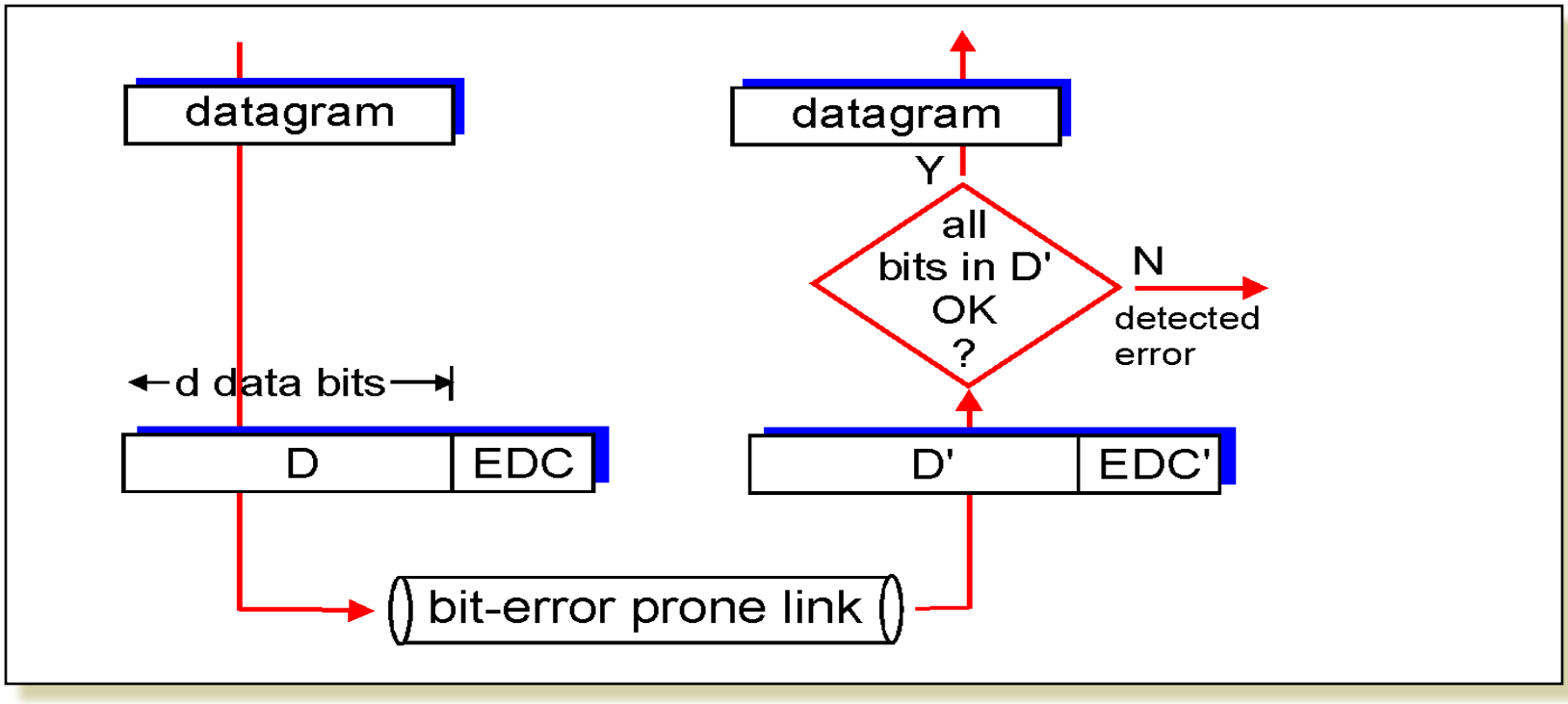


1. Silently return the wrong answer.
2. Detect failure.
3. Correct / mask the failure

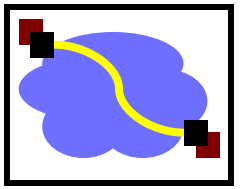
Block error detection/correction



- EDC= Error Detection and Correction bits (redundancy)
- D = Data protected by error checking, may include header fields
- Error detection not 100% reliable!
  - Protocol may miss some errors, but rarely
  - Larger EDC field yields better detection and correction

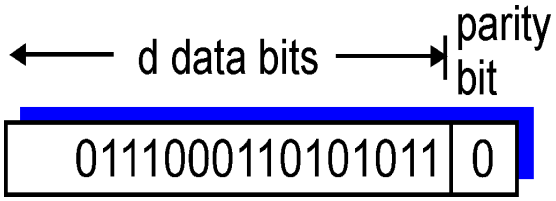


Parity Checking



Single Bit Parity:

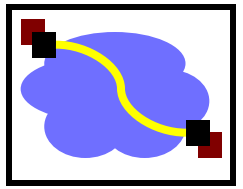
Detect single bit errors



Calculated using XOR over data bits:

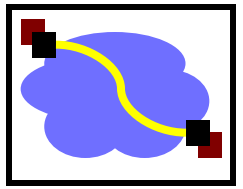
- 0 bit: even number of 0s
- 1 bit: odd number of 0s

Error Detection - Checksum



- Used by TCP, UDP, IP, etc..
- Ones complement sum of all 16-bits in packet
- Simple to implement
  - Break up packet into 16-bits strings
  - Sum all the 16-bit strings
  - Take complement of sum = checksum; add to header
  - One receiver, compute same sum, add sum and checksum, check that the result is 0 (no error)
- Relatively weak detection
  - Easily tricked by typical loss patterns (bursty errors)

Example: Internet Checksum



- Goal: detect “errors” (e.g., flipped bits) in transmitted segment

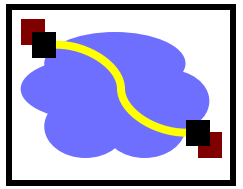
Sender

- Treat segment contents as sequence of 16-bit integers
- Checksum: addition (1’s complement sum) of segment contents
- Sender puts checksum value into checksum field in header

Receiver

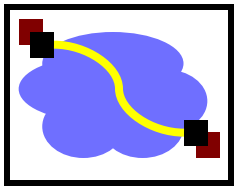
- Compute checksum of received segment
- Check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. But maybe errors nonethless?

Error Detection – Cyclic Redundancy Check (CRC)

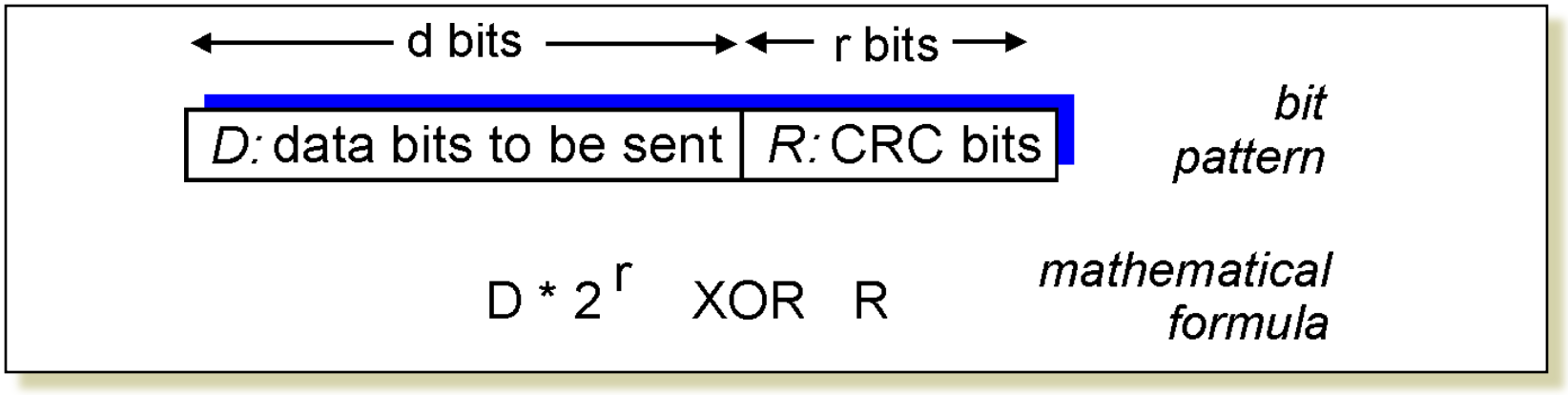


- Polynomial code
  - Treat packet bits a coefficients of n-bit polynomial
  - Choose r+1 bit generator polynomial (well known – chosen in advance)
  - Add r bits to packet such that message is divisible by generator polynomial
- Better loss detection properties than checksums
  - Cyclic codes have favorable properties in that they are well suited for detecting burst errors
  - Therefore, used on networks/hard drives

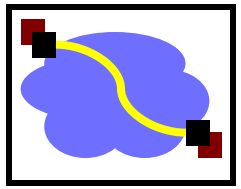
Error Detection – CRC



- View data bits, D, as a binary number
- Choose r+1 bit pattern (generator), G
- Goal: choose r CRC bits, R, such that
  - <D,R> exactly divisible by G (modulo 2)
  - *Receiver knows G*, divides <D,R> by G. If non-zero remainder: error detected!
  - Can detect all burst errors less than r+1 bits
- Widely used in practice



CRC Example



Want:

$D \cdot 2^r$ XOR R = nG

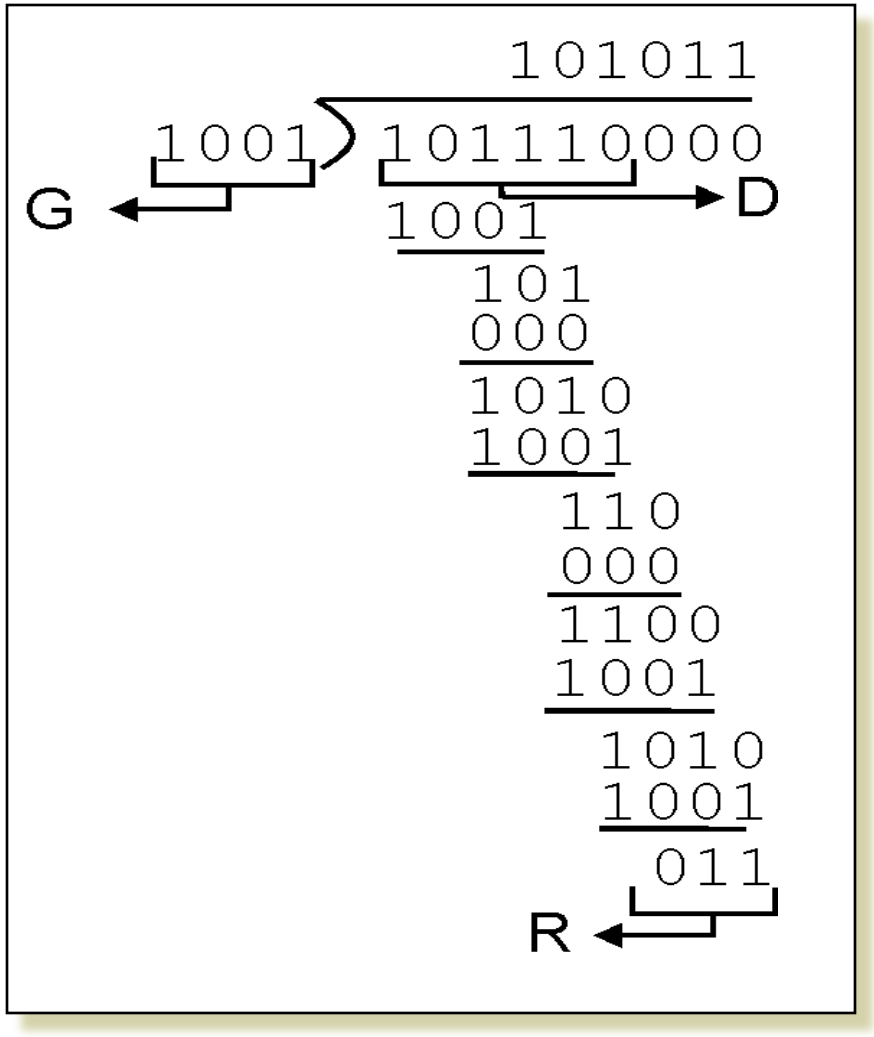
equivalently:

$D \cdot 2^r$ = nG XOR R

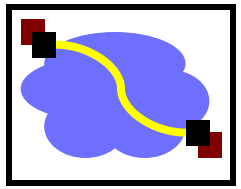
equivalently:

if we divide $D \cdot 2^r$ by G, want reminder R

$$R = \text{remainder}\left[\frac{D \cdot 2^r}{G}\right]$$

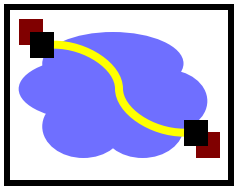


CRC notes



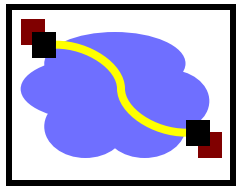
- n-bit CRC = appended value is n-bits long
- Typical CRCs:
  - CRC-8, CRC-16, CRC-32, CRC-64
- CRC-1 = parity bit (degenerate CRC case!)
- Error detection, but not correction
- Usage:
  - RFID (CRC-5)
  - Ethernet, PNG, Gzip, MPEG-2.. (CRC-32)
  - 2G/GSM (CRC-40)
- Many practical considerations:
  - https://en.wikipedia.org/wiki/Computation_of_cyclic_redundancy_checks

Options in dealing with failure



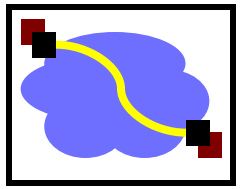
1. Silently return the wrong answer.
2. Detect failure.
3. Correct / mask the failure

Error Recovery



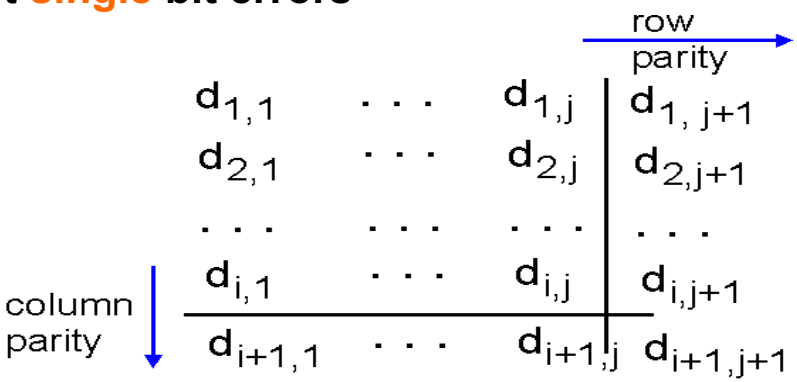
- Two forms of error recovery
  - Redundancy
    - Error Correcting Codes (ECC)
    - Replication/Voting
  - Retry
- ECC
  - Keep encoded redundant data to help repair losses
  - Forward Error Correction (FEC) – send bits in advance
    - Reduces latency of recovery at the cost of bandwidth

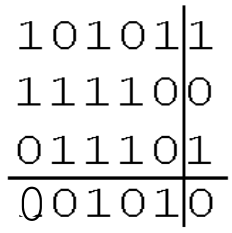
Error Recovery – Error Correcting Codes (ECC)



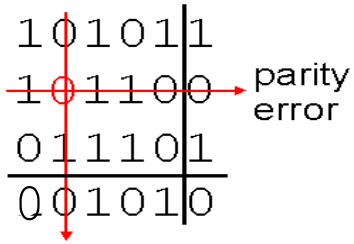
Two Dimensional Bit Parity:

Detect *and correct* single bit errors



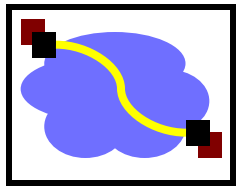


no errors



correctable single bit error

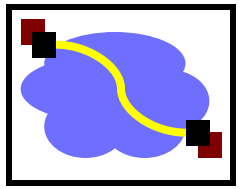
Replication/Voting



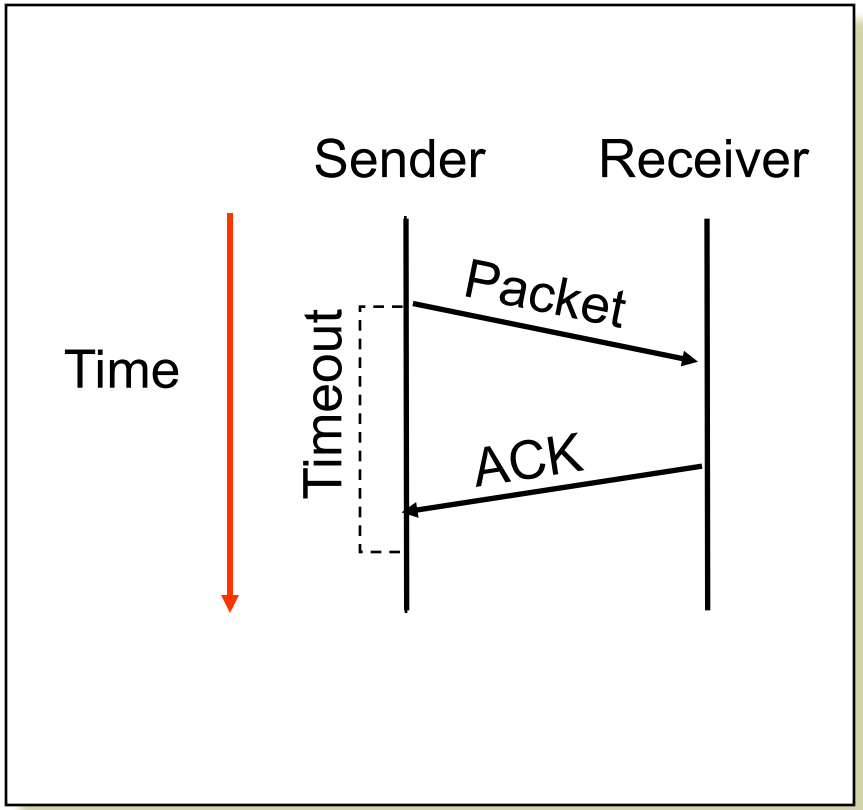
- If you take this to the extreme, three software versions:
  [r1] [r2] [r3]
- Send requests to all three versions of the software: Triple modular redundancy
  - Compare the answers, take the majority
  - Assumes no error detection
- In practice - used mostly in space applications; some extreme high availability apps (stocks & banking? maybe. But usually there are cheaper alternatives if you don't need real-time)
  - Stuff we cover later: surviving malicious failures through voting (byzantine fault tolerance)

31

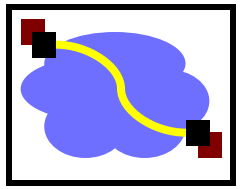
Retry – Network Example



- Sometimes errors are transient / need to mask
- Need to have error detection mechanism
  - E.g., timeout, parity, checksum
  - No need for majority vote

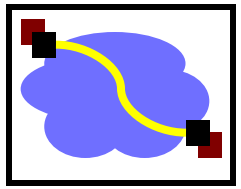


One key question



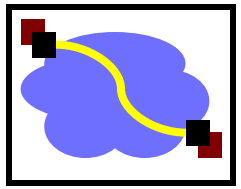
- How correlated are failures?
- Can you assume independence?
  - If the failure probability of a computer in a rack is p,
  - What is p(computer 2 failing) | computer 1 failed?
    - Maybe it's p... or maybe they're both plugged into the same UPS...
- Why is this important?

Back to Disks… What are our options?



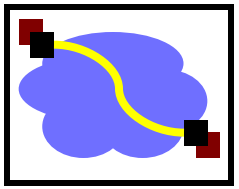
1. Silently return the wrong answer.
2. Detect failure.
   - Every sector has a header with a checksum. Every read fetches both, computes the checksum on the data, and compares it to the version in the header. Returns error if mismatch.
3. Correct / mask the failure
   - Re-read if the firmware signals error (may help if transient error, may not)
   - Use an error correcting code (what kinds of errors do they help?)
     - Bit flips? Yes. Block damaged? No
   - Have the data stored in multiple places (RAID)

Fail-fast disk



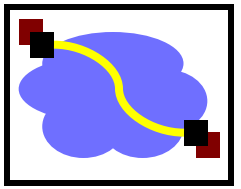
```
failfast_get (data, sn) {
        get (sector, sn);
        if (checksum(sector.data) = sector.cksum) {
                data ← sector.data;
                return OK;
        } else {
                return BAD;
        }
}
```

Careful disk (try 10 times on error)



```
careful_get (data, sn) {
        r ← 0;
        while (r < 10) {
                r ← failfast_get (data, sn);
                if (r = OK) return OK;
                r++;
        }
        return BAD;
}
```

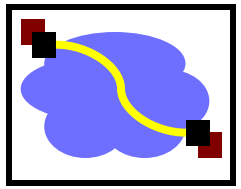
"RAID"



- Redundant Array of {Inexpensive, Independent} disks
- Replication! Idea: Write everything to two disks ("RAID-1")
  - If one fails, read from the other
- write(sector, data) ->
  - write(disk1, sector, data)
  - write(disk2, sector, data)
- read(sector, data)
  - data = read(disk1, sector)
  - if error
    - data = read(disk2, sector)
    - if error, return error
  - return data
- Not perfect, though... doesn't solve all uncaught errors.

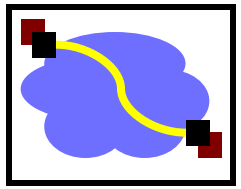
37

Durable disk (RAID 1)



```
durable_get (data, sn) {
        r ← disk1.careful_get (data, sn);
        if (r = OK) return OK;
        r ← disk2.careful_get (data, sn);
        signal(repair disk1);
        return r;
}
```

Summary



- Definition of MTTF/MTBF/MTTR: Understanding availability in systems.
- Failure detection and fault masking techniques
- Engineering tradeoff: Cost of failures vs. cost of failure masking.
  - At what level of system to mask failures?
  - Leading into replication as a general strategy for fault tolerance (more RAID next time)
- Thought to leave you with:
  - What if you have to survive the failure of entire machine? Of a rack of machines? Of a datacenter?

39