Networking and Errors

Networks: Talking to the Outside World

- Originally sharing I/O devices between computers
  - E.g., printers
- Then communicating between computers
  - E.g., file transfer protocol
- Then communicating between people
  - E.g., e-mail
- Then communicating between networks of computers
  - E.g., file sharing, www, …
- Then turning multiple cheap systems into a single computer
  - Warehouse scale computing

The Internet (1962)

www.computerhistory.org/internet_history

- History
  - 1963: J.C.R. Licklider, while at DoD's ARPA, writes a memo describing desire to connect the computers at various research universities: Stanford, Berkeley, UCLA, ...
  - 1969 : ARPA deploys 4 "nodes" @ UCLA, SRI, Utah, & UCSB
  - 1973 Robert Kahn & Vint Cerf invent TCP, now part of the Internet Protocol Suite
- Internet growth rates
  - Exponential since start

www.greatachievements.org/?id=3736
en.wikipedia.org/wiki/Internet_Protocol_Suite

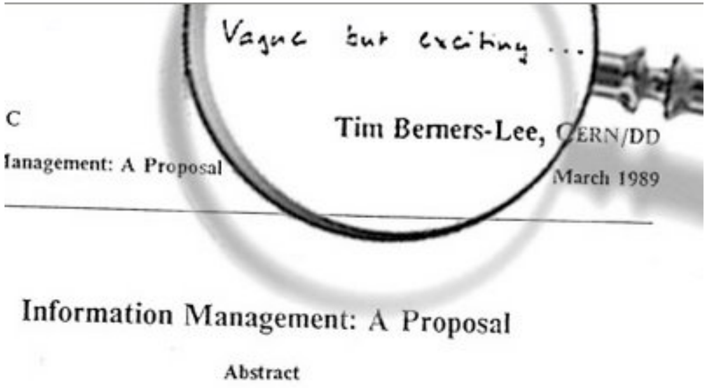
The World Wide Web (1989)

en.wikipedia.org/wiki/History_of_the_World_Wide_Web

- "System of interlinked hypertext documents on the Internet"
- History
  - 1945: Vannevar Bush describes (a hypothetical electromechanical device and) hypertext system called "memex" in article
  - 1989: Sir Tim Berners-Lee proposed and implemented the first successful communication between a Hypertext Transfer Protocol (HTTP) client and server using the internet.
  - 1993: NCSA Mosaic: A graphical HTTP client
  - ~2000 Dot-com entrepreneurs rushed in, 2001 bubble burst
- Today : Access anywhere!

Tim Berners-Lee

World's First web server in 1990



Vague but exciting ...

C

Tim Berners-Lee, CERN/DD

lanagement: A Proposal

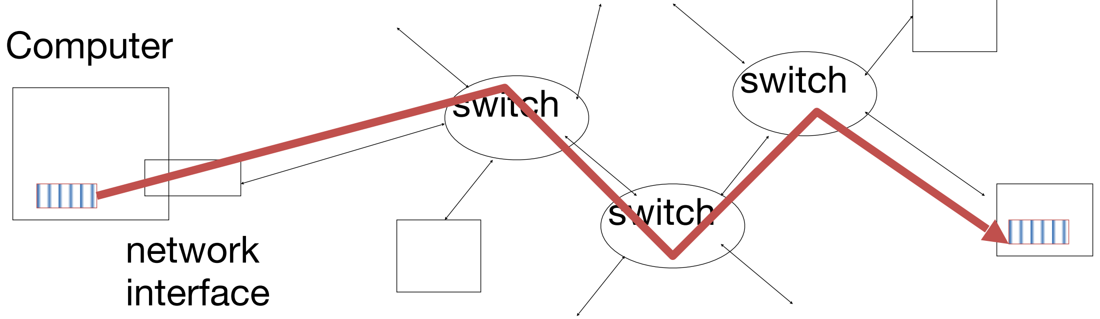
March 1989

Information Management: A Proposal

Abstract

What makes networks work?

- links connecting switches and/or routers to each other and to computers or devices

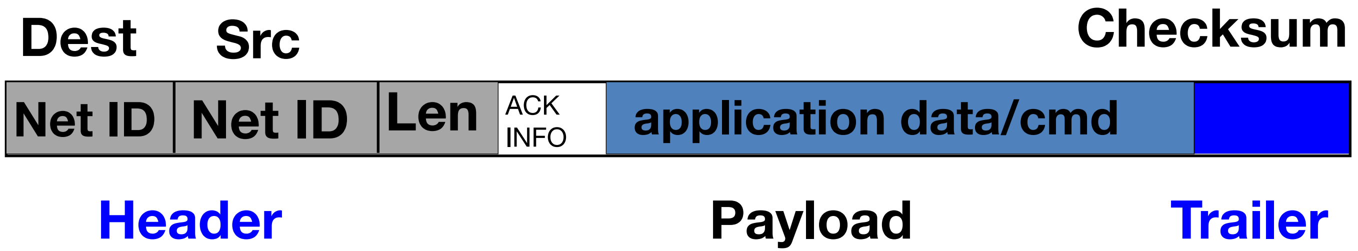


- Ability to name the components and to route packets of information - messages - from a source to a destination

- Layering, redundancy, protocols, and encapsulation as means of abstraction (61C big idea)

Software Protocol to Send and Receive

- SW Send steps
  - 1: Application copies data to OS buffer
  - 2: OS calculates checksum
  - 3: OS sends DMA request to network interface HW and says start
- SW Receive steps
  - 3: Network interface copies data from network interface HW to OS buffer, triggers interrupt
  - 2: OS calculates checksum, if OK, send ACK; if not, delete message (sender resends when timer expires)
  - 1: If OK, OS copies data to user address space, & signals application to continue



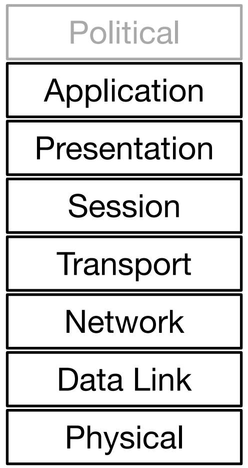
Protocols for Networks of Networks?

What does it take to send packets across the globe?

- Bits on wire or air
- Packets on wire or air
- Delivery packets within a single physical network
- Deliver packets across multiple networks
- Ensure the destination received the data
- Create data at the sender and make use of the data at the receiver

Protocol for Networks of Networks?

- Lots to do and at multiple levels!
- Use abstraction to cope with complexity of communication
- Networks are like onions
  - Hierarchy of layers - network “stack”:
    - Application (chat client, game, etc.)
    - Transport (TCP, UDP)
    - Network (IP)
    - Data Link Layer (ethernet)
    - Physical Link (copper, wireless, etc.)
- OSI 7 layer model actually out of date:
  - Don't really have a "Presentation" or "Session" layer
  - Security is often grafted in as a layer 4.5 "Transport Encryption" on top of the transport layer



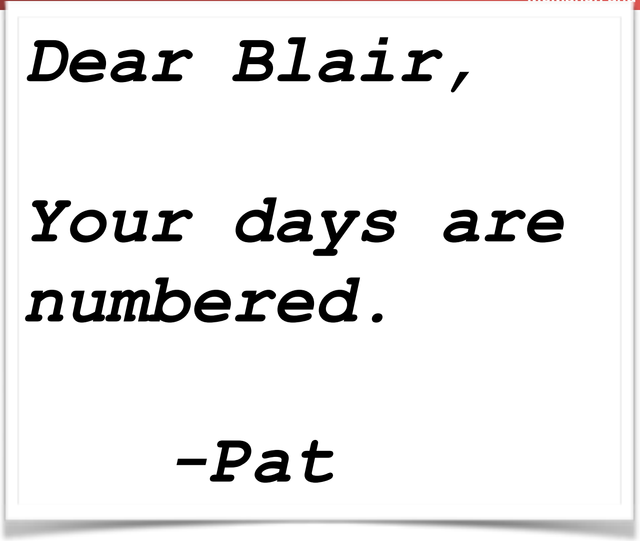
OSI 7 Layer Network Model

Protocol Family Concept

- Protocol: packet structure and control commands to manage communication
- Protocol families (suites): a set of cooperating protocols that implement the network stack
- Key to protocol families is that communication occurs logically at the same level of the protocol, called peer-to-peer…
  …but is implemented via services at the next lower level
- Encapsulation: carry higher level information within lower level "envelope"

Inspiration…

- CEO A writes letter to CEO B
  - Folds letter and hands it to assistant
- Assistant:
  - Puts letter in envelope with CEO B's full name
  - Takes to FedEx
- FedEx Office
  - Puts letter in larger envelope
  - Puts name and street address on FedEx envelope
  - Puts package on FedEx delivery truck
- FedEx delivers to other company



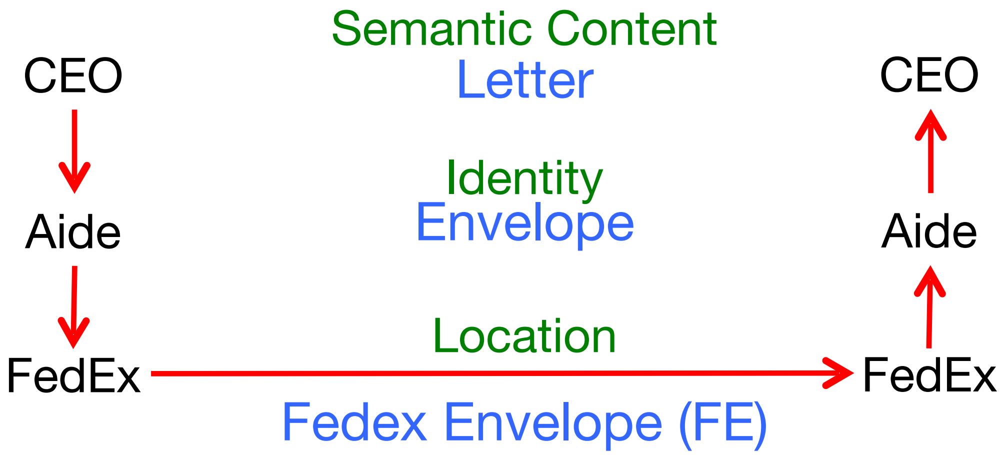
Dear Blair,

Your days are numbered.

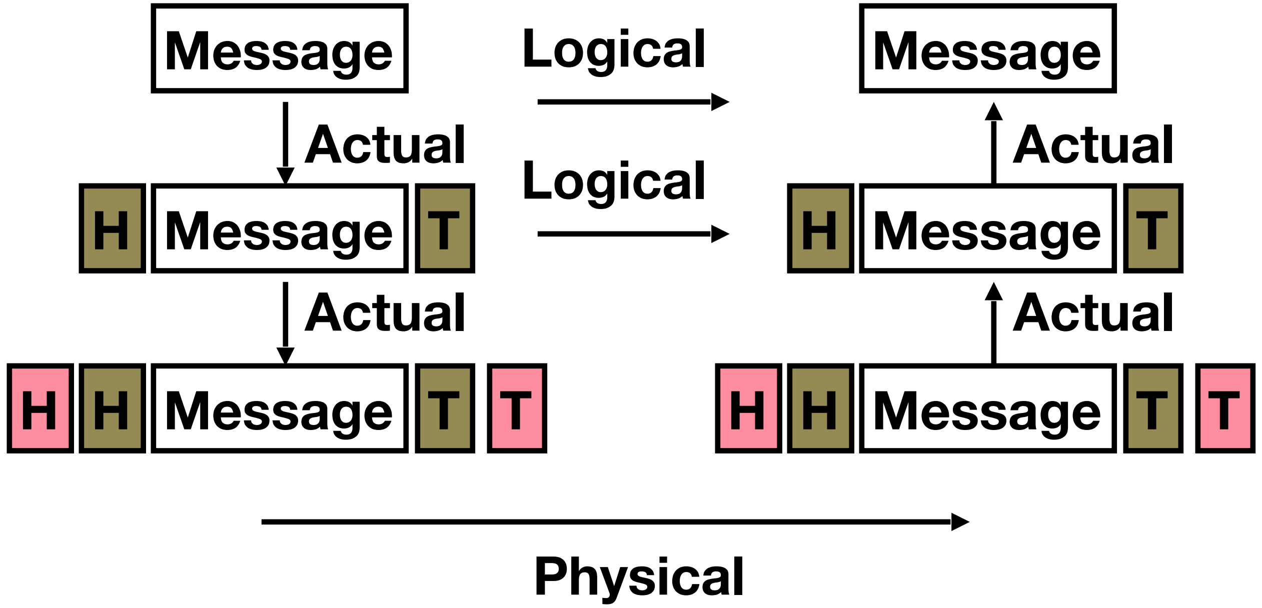
-Pat

The Path of the Letter

"Peers" on each side understand the same things.
No one else needs to.
Lowest level has most packaging.



Protocol Family Concept



Each lower level of stack "encapsulates" information from layer above by adding header and trailer.

Internet and "Best Effort"

- IP (The Internet Protocol) is the common layer 3 protocol for everything
- Designed to send messages to machines identified by IP address, (a 32b (IPv4) or 128b (IPv6) value) anywhere in the world
  - Single global address space: Given an IP address, any router in the network will send a packet towards the right place
- Based around packets (blocks of data) with "Best Effort"
  - "Best Effort" is really more "half-assed job":
    Packets may be dropped, reordered, or corrupted
    - But corrupted packets are turned into dropped packets by various error detection methods

The Foundational Layer 4 Protocols of the Internet: TCP (Transmission Control Protocol)

- Connection Based
  - You establish a communication channel between the client and server and then start communicating
- Ports
  - Communication is to a specific port (attached to a program) on a remote computer
- Reliable:
  - Messages which are sent will be acknowledged when received by the recipient
- In order:
  - Messages are received in the order they are sent
- Bytestream:
  - Abstract of "Stream of bytes" in each direction:
    Can write and receive data just as arbitrarily long arrays of bytes
- Congestion control:
  - TCP views dropped packets as a signal to slow down to enable fair sharing of the network

Example TCP code in Go

```go
// Server side
// Listen for a TCP connection from
// anyone on port 3030
l, _ := net.Listen("tcp", ":3030")

// Accept connections...
for {
  conn, err := l.Accept()
  if err != nil {
     ...
  }
  // Spawn off a NEW thread
  // to handle this particular
  // connection with my own
  // function
  go handleConn(conn);
}
```

```go
// Client side
// Connect to the remote system on port
// 3030
conn, err := net.Dial("tcp", RemoteSystem
+ ":3030")
if err != nil {
  ...
}
// Now just do what I want with the
// connection...
// I can both Read() and Write()
// arrays of bytes to the connection.
handleClientConn(conn);
```

The Other Foundational L4 Protocol: UDP (Universal Datagram Protocol)

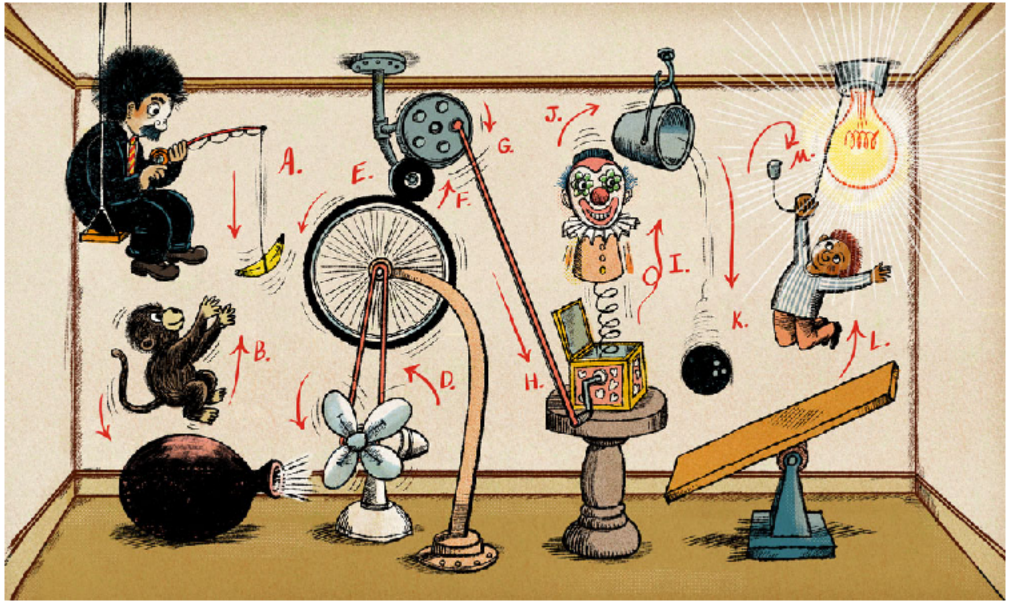
- No connections
  - Just send to particular ports on remote computers and listen on ports for incoming messages
- Datagram
  - A single block of bytes: In practice it should be ~1400B but it can theoretically be a lot larger
- Unreliable
  - Datagrams can get lost
- Out of order
  - Datagrams can be received out of order

When To Use UDP instead of TCP?

- Can't tolerate the latency in initiating a connection
  - Why UDP is primarily used for DNS (The Domain Name System): The service that converts names (www.cs61c.org) to computer addresses (**`104.21.52.249`** or **`2606:4700:3035::ac43:ce10`**)
- Old data is useless data
  - Voice and video games
  - TCP will pause the data stream in the case of a transmission error until the

Dependability: Reliability in Computer Hardware

- What could possibly go wrong?
- How do we mitigate the effects of faults and errors?



Types of Faults in Digital Designs

1. Design Bugs (function, timing, power draw)
    - detected and corrected at design time through testing and verification (simulation, static checks)
2. Manufacturing Defects (violation of design rules, impurities in processing, statistical variations)
    - post production testing for sorting
    - spare on-chip resources for repair
3. Runtime Failures (physical effects and environmental conditions)
    - “Hard Faults”: aging
    - “Soft (transient) Faults”: electro-magnetic interference, cosmic particles

Intel Pentium FDIV Design Bug

A hardware bug affecting the floating point unit of the early Intel Pentium processors

The processor might return incorrect binary floating point results when dividing a number. For example, the chip produced:

$$\frac{4{,}195{,}835}{3{,}145{,}727} = 1.333739068902037589$$

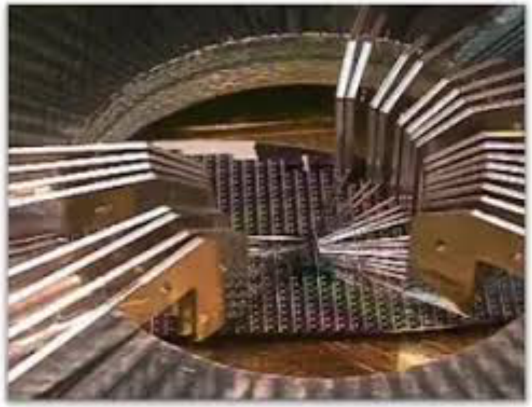
Instead of the correct: $$\frac{4{,}195{,}835}{3{,}145{,}727} = 1.333820449136241002$$

Intel attributed the error to missing entries in the lookup table used by the floating-point division circuitry.

In December 1994, Intel recalled the defective processors. In January 1995, Intel announced a pre-tax charge of $475 million against earnings, the cost associated with replacement of the flawed processors.

Dealing with Manufacturing Faults in ICs

- Designers provide "test vectors"
  - Tools help with ATPG (Automatic Test Pattern Generation)
- Completed ICs are tested and "binned" for correct operation, and speed grade.



- Special on-chip circuits help speed the testing process
  - BIST (built in self test), Scan-chains

Manufacturing Faults and Modern Processors

- A single fault may break an entire processor core
  - But we have 4 or 8 cores on a single chip
- Common solution: Disable the defective core(s)
  - Then sell the chip as a lower-core version
- EG, Nick's Zoom-Cave Beast
  - Basic Zen2 "chiplet" contains 8 processor cores
  - His system has two chiplets in the CPU
  - But only 12 usable processor cores!
    - So deliberately uses defective chiplets

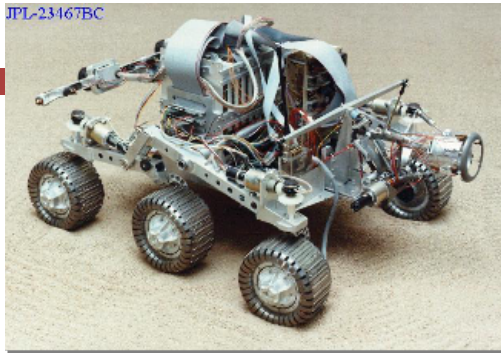
Today: Dealing with Runtime Failures (errors)

- Redundancy
- Measures of Dependability
- Codes for Error Detection/Correction
- Protecting Disk-Drives Against Errors
- And in Conclusion, …

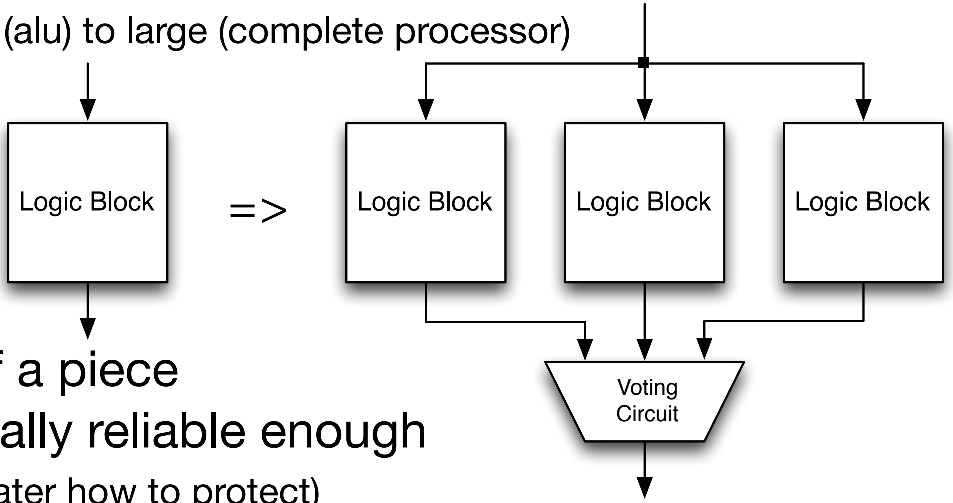
Great Idea #6: Dependability via Redundancy

- Applies to everything from data centers to memory
  - Redundant data centers so that can lose 1 datacenter but Internet service stays online
  - Redundant routes so can lose nodes but Internet doesn't fail
    - Or at least can recover quickly…
  - Redundant disks so that can lose 1 disk but not lose data (Redundant Arrays of Independent Disks (RAID))
  - Redundant memory bits so that can lose 1 bit but no data (Error Correcting Code (ECC) Memory)

A Fault-Tolerant Design Methodology



- Where hardware errors cannot be tolerated
  - Space-craft data systems, life-supporting medical devices, …
- Triple Modular Redundancy
  - Relies on simple reliable voting circuit, assume P(>1 error) is small
  - "Logic Block" could be all the way from small (alu) to large (complete processor)

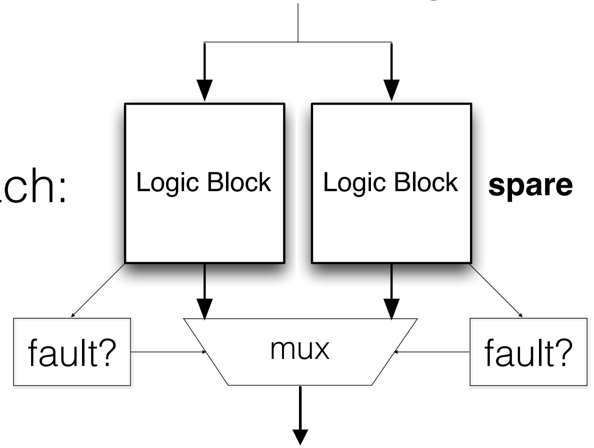


- On earth and for the normal lifetime of a piece of equipment, hardware circuits generally reliable enough
  - except for *large memory circuits* (we will see later how to protect)

Dependability Corollary: Fault Detection

- The ability to determine that ***something*** is wrong is often the key to effectively using redundancy:

Simple "sparing" approach:



- Often error detection is a necessary prerequisite to error correction

Dependability via Redundancy: Time vs. Space

- ***Spatial Redundancy*** – replicated data or extra information or hardware to handle hard and soft (transient) failures
- ***Temporal Redundancy*** – redundancy in time (retry) to handle soft (transient) failures
    - "~~Insanity~~ overcoming soft failures: repeatedly doing the same thing and expecting different results"

Dependability Measures

- *Reliability*: Mean Time To Failure (***MTTF***)
- *Service interruption*: Mean Time To Repair (***MTTR***)
- Mean time between failures (***MTBF***)
  - MTBF = MTTF + MTTR
- Availability = ***MTTF*** / (***MTTF*** + ***MTTR***)
- Improving Availability
  - Increase ***MTTF***: More reliable hardware/software + Fault Tolerance
  - Reduce ***MTTR***: improved tools and processes for diagnosis and repair

Availability Example

- Suppose your laptop dies on average once every 2 years
- Each time you take make an appointment at the repair shop and get it repaired, resulting in a total of 2 days downtime.
- What is the availability of your laptop?

$$MTTF = 2 * 365 \ days$$

$$MTTR = 2 \ days$$

$$availability = \frac{MTTF}{MTTF + MTTR} = \frac{730}{730 + 2} = .997 = 99.7\,\%$$

Availability Measures

- Availability = MTTF / (MTTF + MTTR) as %
- Since we hope things are rarely down, shorthand is "number of 9s of availability per year"
- 1 nine: 90% => 36 days of repair/year
  - Eduroam Reliability?
- 2 nines: 99% => 3.6 days of repair/year
- 3 nines: 99.9% => 526 minutes of repair/year
- 4 nines: 99.99% => 53 minutes of repair/year
- 5 nines: 99.999% => 5 minutes of repair/year
  - And serious $$$ to do

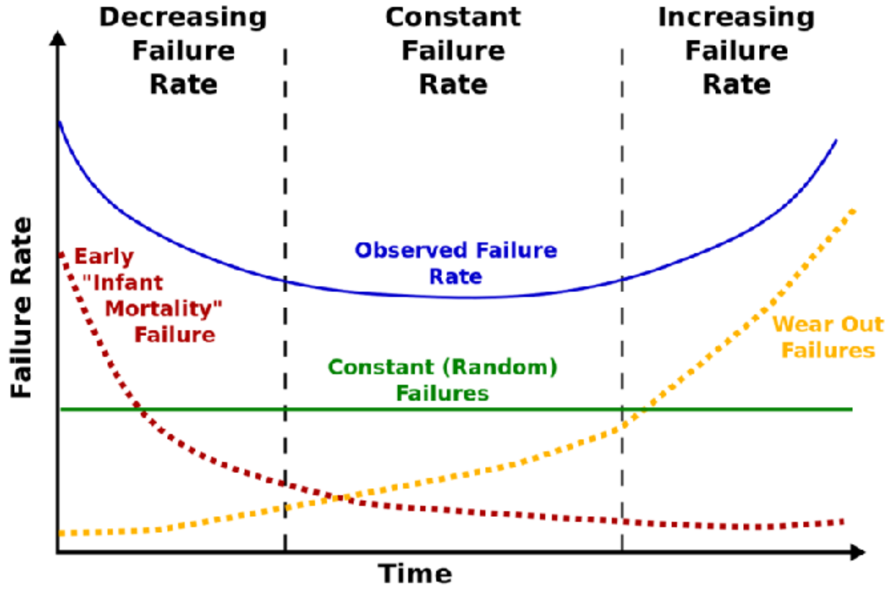
Reliability Measures

- Another is average number of failures per year: ***Annualized Failure Rate (AFR)***
  - E.g., 1000 disks with 100,000 hour MTTF
  - 365 days/yr * 24 hours = 8760 hours/yr
  - (1000 disks * 8760 hours/yr) / 100,000 hours/failure = 87.6 failed disks ***per year*** on average
  - 87.6/1000 = 8.76% annual failure rate
- Google's 2007 study* found that actual AFRs for individual drives ranged from 1.7% for first year drives to over 8.6% for three-year old drives

research.google**.com/archive/disk_failures.pdf*

The "Bathtub Curve"

- Often failures follow the "bathtub curve"
- Brand new devices, higher prob of failure
- Old devices, higher prob of failure
- In between, lower
- Some manufacturers will “burn in” products to eliminate early failures



https://upload.wikimedia.org/wikipedia/commons/7/78/Bathtub_curve.svg

Dependability Design Principle

- Design Principle: No single points of failure
  - “Chain is only as strong as its weakest link”
- Dependability behaves like speedup of Amdahl’s Law
  - Doesn’t matter how dependable you make one portion of system
  - Dependability limited by part you do not improve

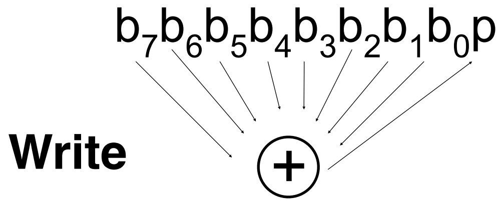
Time For a Loafing Alden (From @CalFalconsCam)

Error Detection/Correction Codes (EDC/ECC)

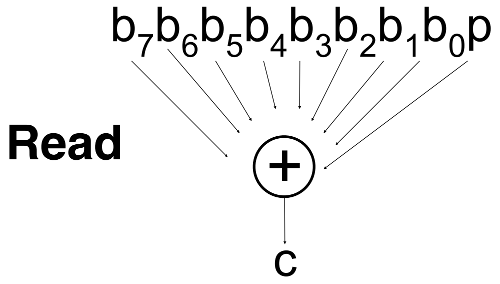
- Memory systems generate errors (accidentally flipped-bits)
  - DRAMs store very little charge per bit
  - "***Soft***" errors occur occasionally when cells are struck by alpha particles or other environmental upsets
  - "***Hard***" errors can occur when chips permanently fail
  - Problem gets worse as memories get denser and larger
- Memories protected against failures with EDC/ECC
- Extra bits are added to each data-word
  - Used to detect and/or correct faults in the memory system
  - Each data word value mapped to unique code word
  - A fault changes valid code word to invalid one, which can be detected

Simple Error Detection Coding: Parity Bit

- Each data value, before it is written to memory is "tagged" with an extra bit to force the stored word to have *even parity*:



- Each word, as it is read from memory is "checked" by finding its parity (including the parity bit).



- A non-zero parity check, c, indicates an error occurred:
  - two errors (on different bits) is not detected (nor any even number of errors)
  - odd numbers of errors are detected.

Parity Example

- X = 0101 0101
- 4 ones, *even* parity now
- Write to memory:
  0101 0101 0
  to keep parity even
- Y = 0110 0111
- 5 ones, *odd* parity now
- Write to memory:
  0110 0111 1
  to make parity even
- Read X from memory
  0101 0101 0
- 4 ones => even parity, so no error
- Read X from memory
  1101 0101 0
- 5 ones => odd parity, so error

What if error is parity bit?

Suppose We Want to Correct Errors?

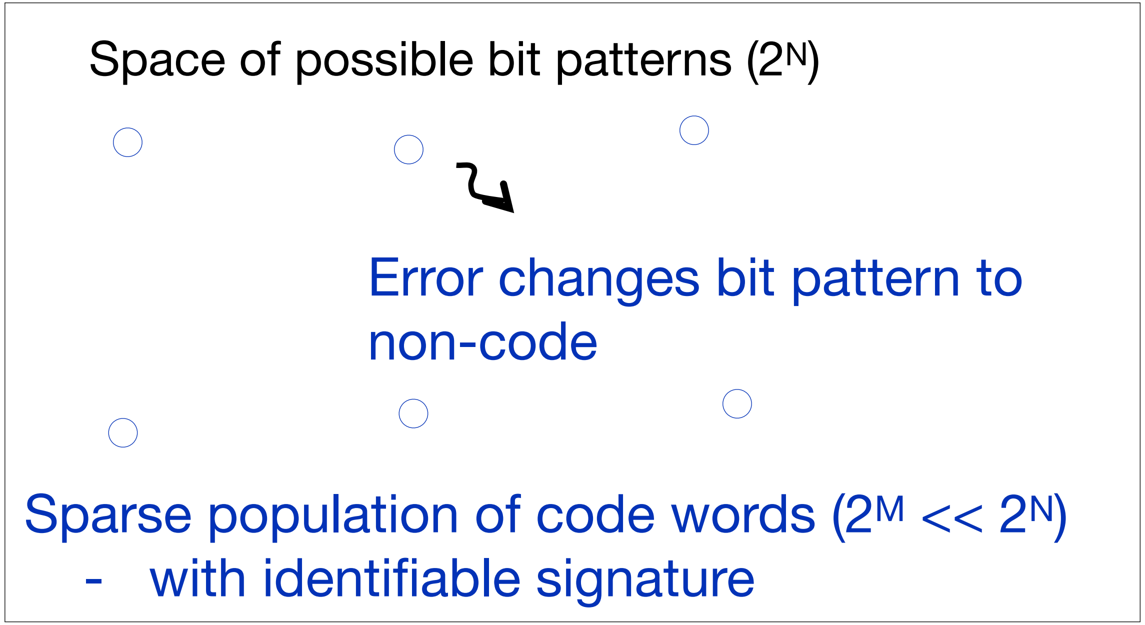
- Hamming came up with simple method using multiple parity bits to enable Error Detection *and Correction*
- Called “Hamming ECC”
  - Worked weekends on relay computer with unreliable card reader, frustrated with manual restarting
  - Got interested in error correction; published 1950
  - R. W. Hamming, “Error Detecting and Correcting Codes,” The Bell System Technical Journal, Vol. XXVI, No 2 (April 1950) pp 147-160.

Richard Hamming
Turing Award Winner

Detecting/Correcting Code Concept

Use N-bits per symbol, and M-bits per valid code-word

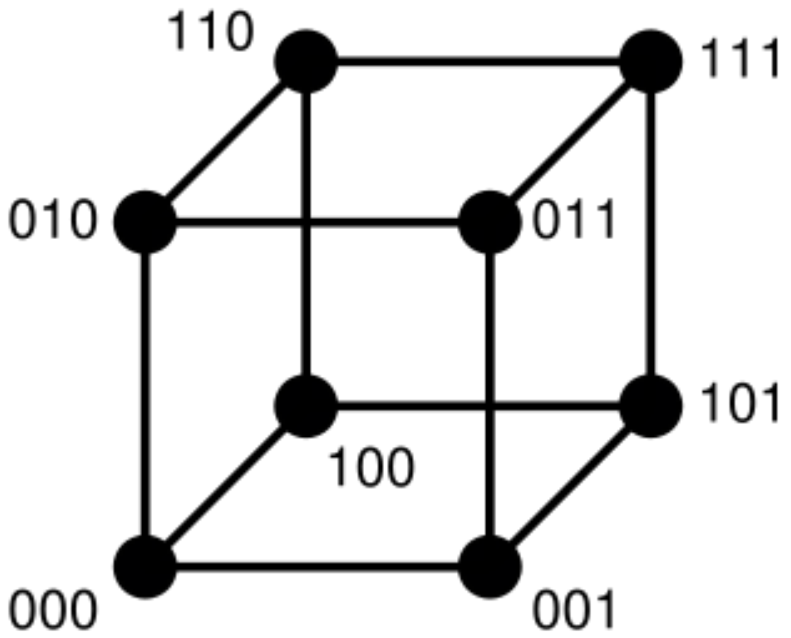
- Detection: bit pattern fails codeword check
- Correction: map to nearest valid code word


Space of possible bit patterns (2^N)
Error changes bit pattern to non-code
Sparse population of code words ($2^M << 2^N$)
- with identifiable signature

Block Code Principle

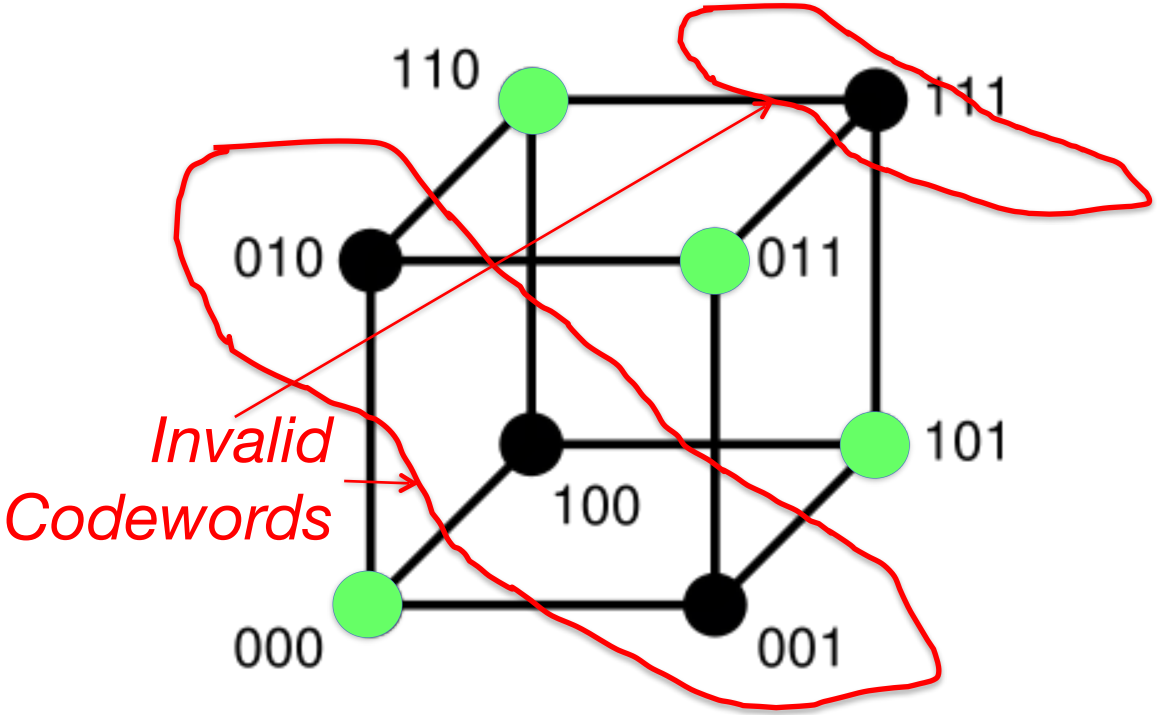
- Hamming distance = # of bits positions where words differ:
- p = 0<u>1</u>1<u>0</u>11, q = 0<u>0</u>1<u>1</u>11, Hamming distance (p,q) = 2
- p = 011011,
  q = 110001, Hamming distance (p,q) = ?
- Can think of adding extra bits to data to create a code word
- What if minimum distance between members of code is 2 and get a 1-bit error?

Hamming Distance 1: 8 code words



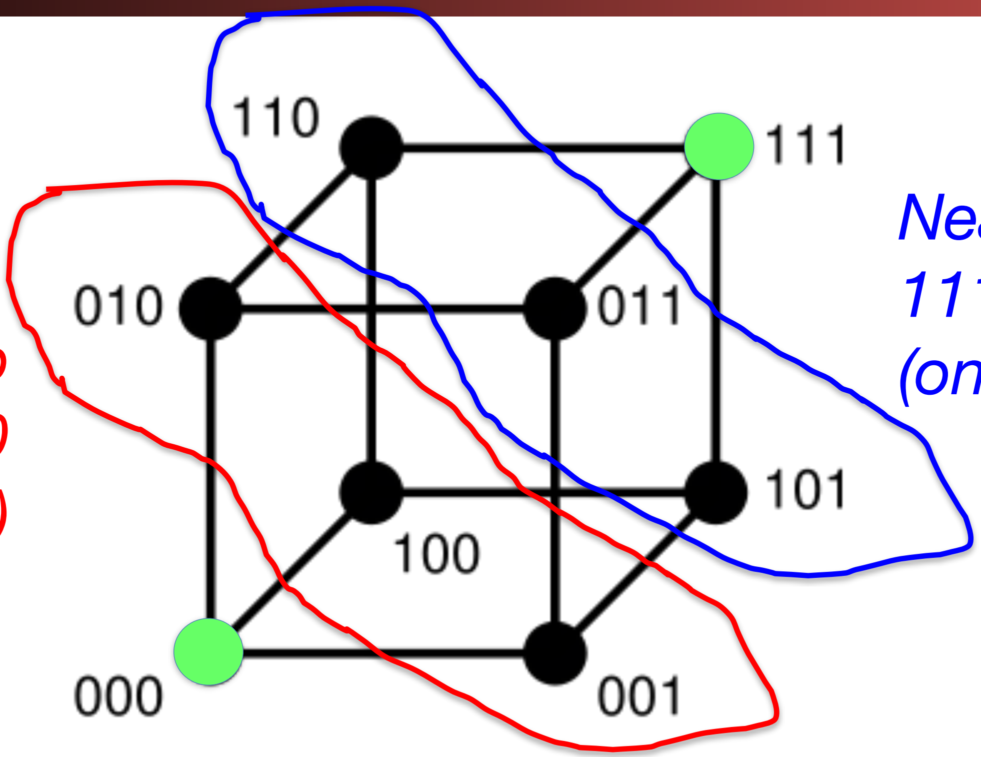
- Any single bit-error goes from valid codeword to another

Hamming Distance 2: Detect Single Bit-Errors



- No single bit-error goes to another valid codeword
- ½ of symbols are valid codewords (in this case, simple parity)

Hamming Distance 3: Correct Single Bit-Errors



Nearest to 000 (one 1)

Nearest to 111 (one 0)

- Any 1 bit-error can be corrected;
- 1/4 symbols are valid code-words

- Correct is voting in this case

Hamming Error Correcting Code

- Use more parity bits to pinpoint bit(s) in error, so they can be corrected.
- Example: Single error correction (SEC) on 4-bit data
  - use 3 parity bits, with 4-data bits results in 7-bit code word
  - 3 parity bits sufficient to identify any one of 7 code word bits
  - overlap the assignment of parity bits so that a single error in the 7-bit word can be corrected
- **Procedure**: group parity bits so they correspond to subsets of the 7 bits:
  - p_1 protects bits 1,3,5,7
  - p_2 protects bits 2,3,6,7
  - p_3 protects bits 4,5,6,7

1	2	3	4	5	6	7
p_1	p_2	d_1	p_3	d_2	d_3	d_4

Note: number bits from left to right.

Bit position number

$$\left.\begin{array}{l} 001 = 1_{10} \\ 011 = 3_{10} \\ 101 = 5_{10} \\ 111 = 7_{10} \end{array}\right\} p_1$$

$$\left.\begin{array}{l} 010 = 2_{10} \\ 011 = 3_{10} \\ 110 = 6_{10} \\ 111 = 7_{10} \end{array}\right\} p_2$$

$$\left.\begin{array}{l} 100 = 4_{10} \\ 101 = 5_{10} \\ 110 = 6_{10} \\ 111 = 7_{10} \end{array}\right\} p_3$$

Hamming Code Example

1	2	3	4	5	6	7
p_1	p_2	d_1	p_3	d_2	d_3	d_4

- Note: parity bits occupy power-of-two bit positions in code-word.
- On writing to memory:
  - parity bits are assigned to force even parity over their respective groups.
- On reading from memory:
  - check bits (c_3,c_2,c_1) are generated by finding the parity of the group and its parity bit. If an error occurred in a group, the corresponding check bit will be 1, if no error the check bit will be 0.
  - check bits (c_3,c_2,c_1) form the position of the bit in error.

- Example: $c = c_3c_2c_1 = 101$
  - error in 4,5,6, or 7 (by c_3=1)
  - error in 1,3,5, or 7 (by c_1=1)
  - no error in 2, 3, 6, or 7 (by c_2=0)
- Therefore error must be in bit 5.
- *Note the check bits point to 5*
- By our clever positioning and assignment of parity bits, the check bits always address the position of the error!
- c=000 indicates no error

Hamming Error Correcting Code

- Adding on extra parity bit covering the entire word can provide double error detection

1	2	3	4	5	6	7	8
p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4

- On reading, the C bits are computed (as usual) plus the parity over the entire word, P:

C=0 P=0, no error

C!=0 P=1, correctable single error

C!=0 P=0, a double error occurred

C=0 P=1, an error occurred in p_4 bit

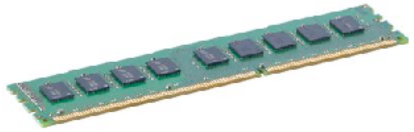
- Overhead involved in single error correction code:
  - let *p* be the total number of parity bits and *d* the number of data bits in a *p* + *d* bit word.
  - If p error correction bits are to point to the error bit (*p* + *d* cases) plus indicate that no error exists (1 case), we need:

$2^p >= p + d + 1,$

thus $p >= \log(p + d + 1)$

for large *d*, *p* approaches log(*d*)

Typical modern codes in DRAM memory systems on servers:
64-bit data blocks (8 bytes) with 72-bit code words (9 bytes), results in SEC, DED.



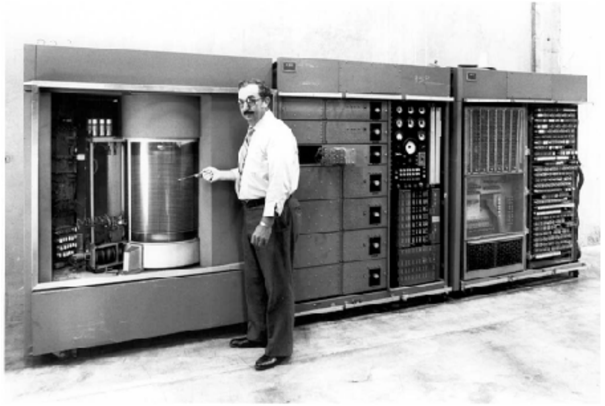
Other Codes

- **Many** different types of codes exist for applications in communications and data storage:
  - Cyclic Redundancy Code (CRC), for error detection Ethernet packets
  - “forward” error correction (FEC) codes in wireless telecomm and WiFi:
    - Low Density Parity Codes (LDPC), Verterbi/Turbo Codes, Polar Codes
  - Reed-Solomon Codes for communicating with Satellites billions of kilometers away, and for storing data on CDs (able to correct defects up to 1mm in size)
  - Cryptographic hash-functions, to guard against malicious tampering (e.g. SHA256)
    - An attacker is not be able to change, add, or remove any bits without changing the hash output
- All use redundancy (extra bits) to represent or summarize the data

RAID: Redundancy for Disks

- Spinning disk is still a critical technology
  - Although worse latency than SSD…
- Disk has equal or greater bandwidth and an order of magnitude better storage density (bits/cm^3) and cost density (bits/$)
- So when you need to store a petabyte or three…
  - You need to use disk, not SSDs
- Oh, and SSDs can fail too

Evolution of the Disk Drive


IBM RAMAC 305, 1956

First commercial computer that used a moving-head hard disk drive (magnetic disk storage) for secondary storage.

up to 22.7 billion bytes (gigabytes) of storage

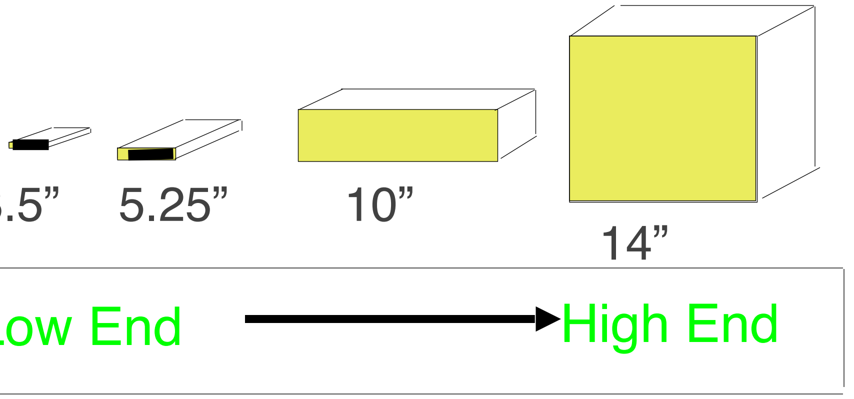
IBM 3390K, 1989

Apple SCSI, 1986

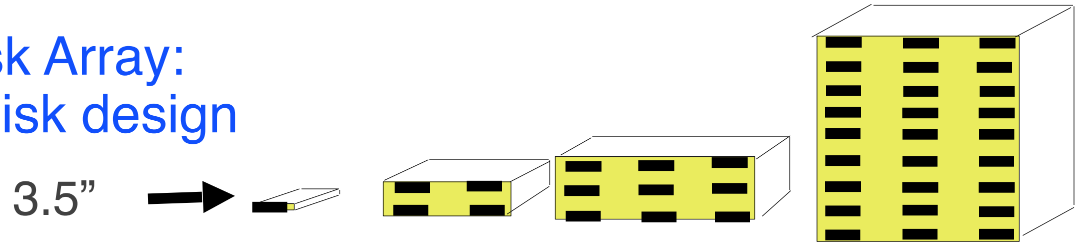
Arrays of Small Disks

Can smaller disks be used to close gap in performance between disks and CPUs?

Conventional:
4 disk designs



Disk Array:
1 disk design



Replace Small Number of Large Disks in 1988

	IBM 3390K	IBM 3.5" 0061	x70	
Capacity	20 GBytes	320 MBytes	23 GBytes	
Volume	97 cu. ft.	0.1 cu. ft.	11 cu. ft.	9X
Power	3 KW	11 W	1 KW	3X
Data Rate	15 MB/s	1.5 MB/s	105 MB/s	7X
I/O Rate	600 I/Os/s	55 I/Os/s	3900 IOs/s	6X
MTTF	250 KHrs	50 KHrs	??? Hrs	
Cost	$250K	$2K	$150K	

Disk Arrays have potential for large data and I/O rates, high MB per cu. ft., high MB per KW, but what about reliability?

But MTTF goes through the roof…

- If 1 disk as MTTF of 50k hours…
  - 70 disks will have a MTTF of ~700 hours!!!
  - This is assuming failures are independent…
- But fortunately we know when failures occur!
  - Disks use a lot of CRC coding, so we don't have corrupted data, just no data
- We can have both "Soft" and "Hard" failures
  - Soft failure just the read is incorrect/failed, the disk is still good
  - Hard failures kill the disk, necessitating replacement
    - Most RAID setups are "Hot swap":
      Unplug the disk and put in a replacement while things are still going
    - Most modern RAID arrays also have "hot spares":
      An already installed disk that is used automatically if another disk fails.

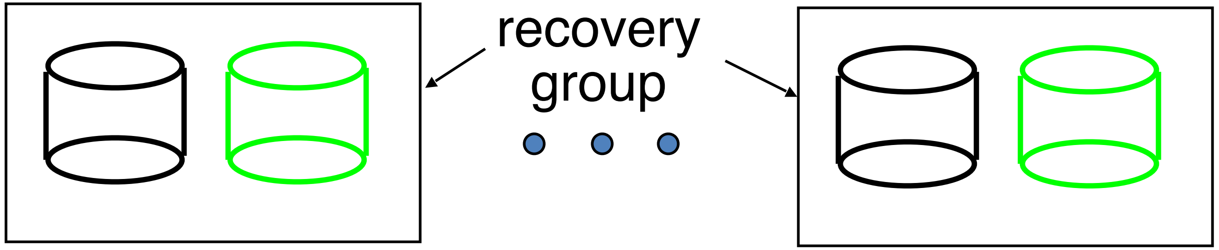
RAID: Redundant Arrays of (Inexpensive) Disks

- Files are "striped" across multiple disks, ex:
- Redundancy yields high data availability
  - Availability: service still provided to user, even if some components failed
- Disks will still fail
- Contents reconstructed from data redundantly stored in the array
  - Capacity penalty to store redundant info
  - Bandwidth penalty to update redundant info on writes
- 6 Raid *Levels,* 0, 1, 5, 6 most common today

RAID 0: Striping

- "RAID 0" is not actually RAID (no redundancy)
  - It is simply spreading the data across multiple disks
- So, e.g, for 4 disks, stripe-unit address 0 is on disk 0, address 1 is on disk 1, address 2 is on disk 2, address 4 on disk 0...
- Improves bandwidth linearly
  - With 4 disks you have 4x the disk bandwidth
- Doesn't really help latency
  - Still have the individual disks seek and rotation time
- Failures will happen...

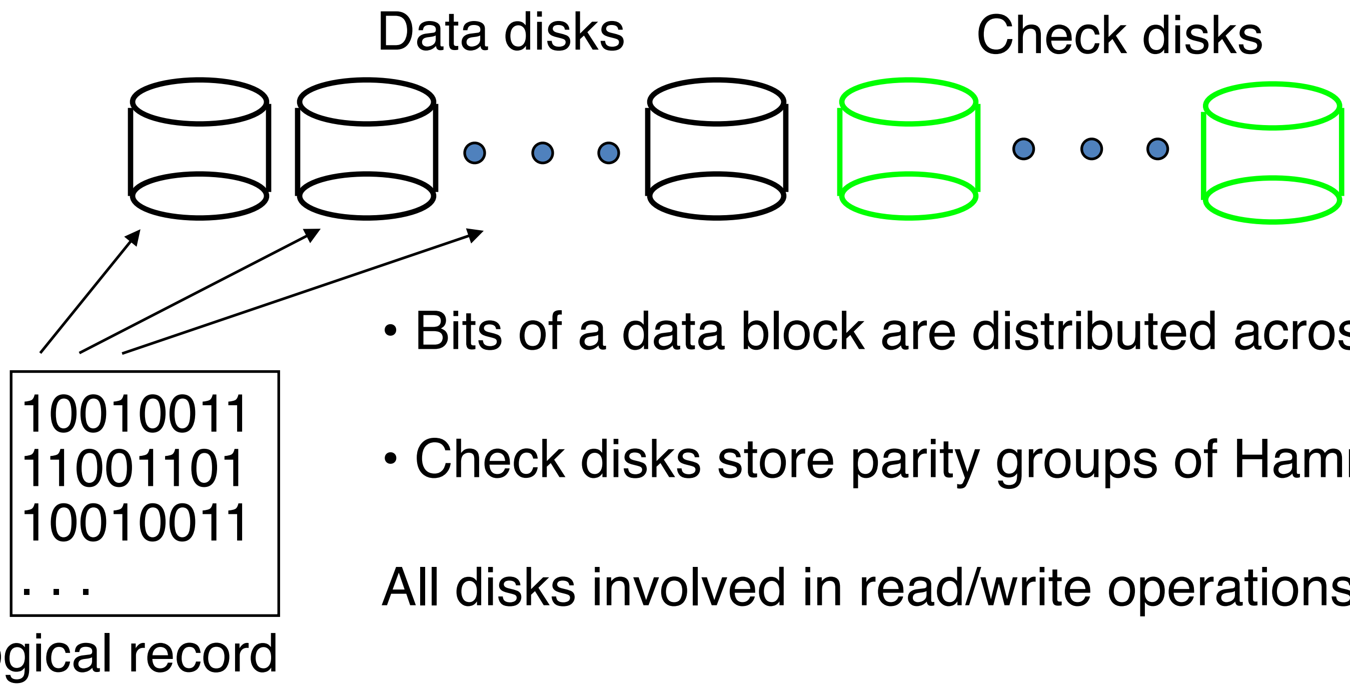
RAID 1: Disk Mirroring/Shadowing (online sparing)



- Each disk is fully duplicated onto its "mirror"
  Very high availability can be achieved
- Writes go to disk and mirror - limited by single-disk speed
- Reads from original disk, unless failure

Most expensive solution: 100% (2x) capacity overhead

RAID 2: Hamming Code for Error Correction



```
10010011
11001101
10010011
. . .
```

- Bits of a data block are distributed across all disks
- Check disks store parity groups of Hamming code

All disks involved in read/write operations

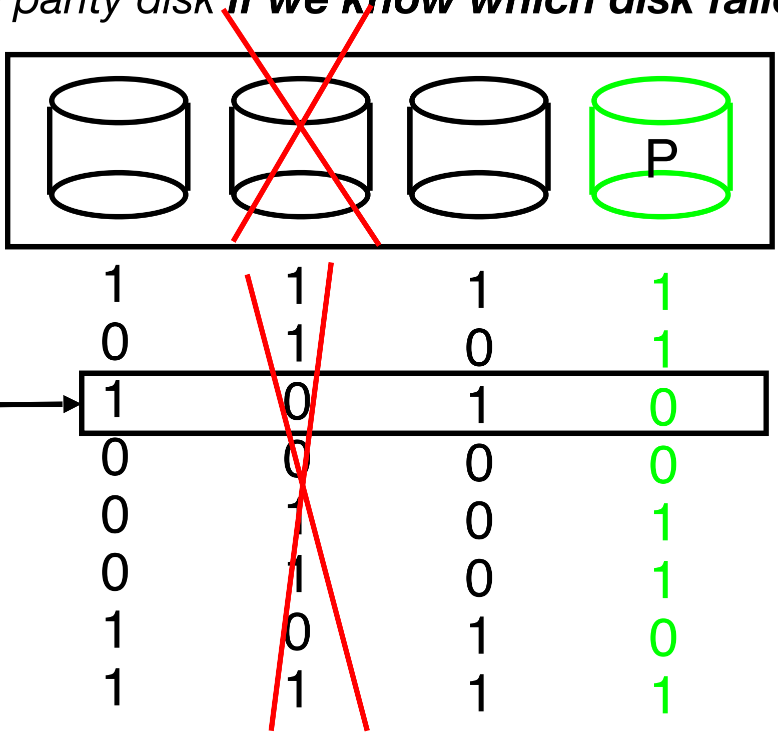
Not actually used, don't worry about remembering it!

RAID 3: Single Parity Disk

- *Disk drives themselves code data and detect failures*
- *Reconstruction of data can be done with single parity disk **if we know which disk failed***
- *Writes change data disk and P disk*

```
10010011
11001101
10010011
. . .
```

logical record



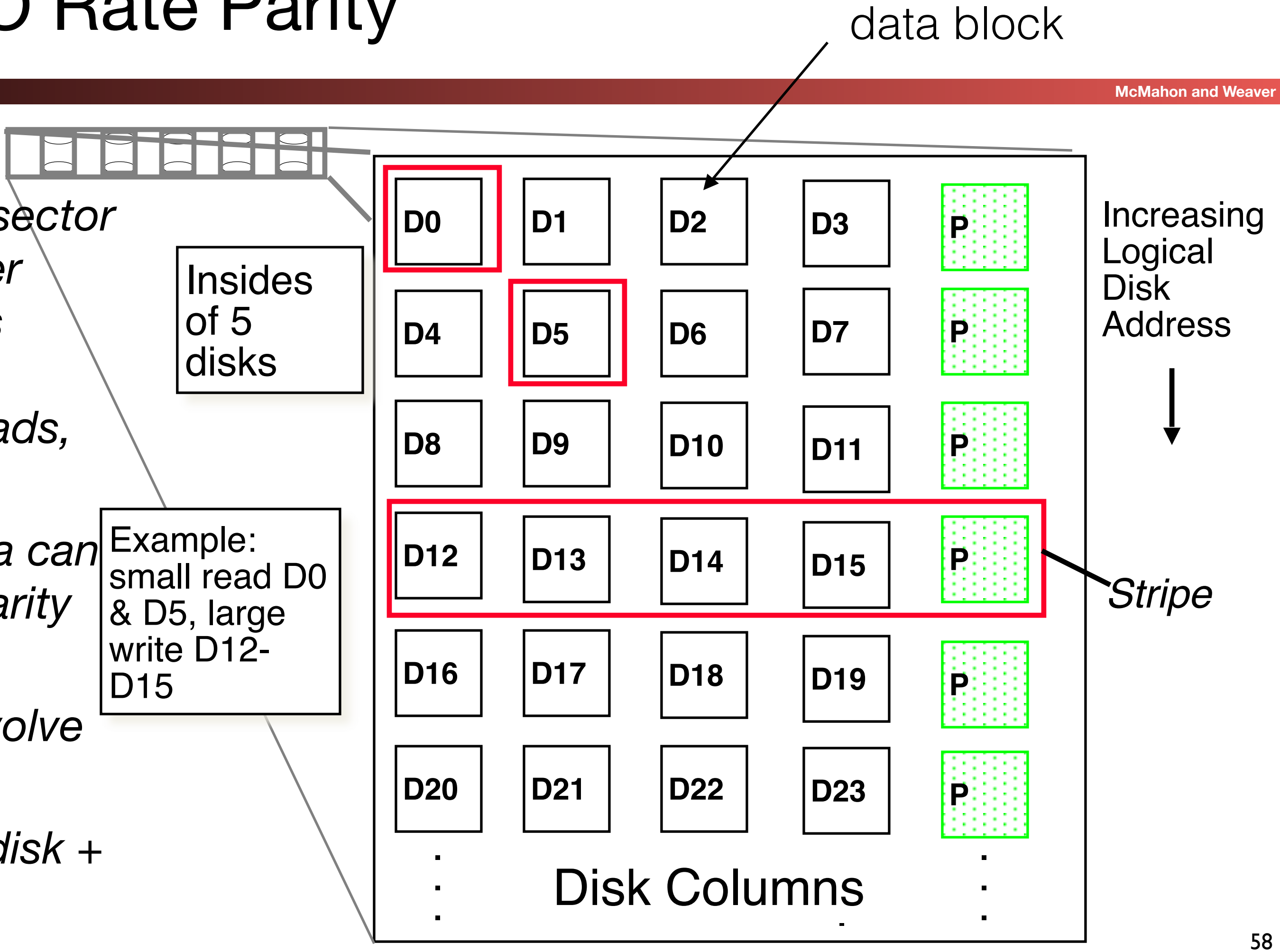
Striped physical records

P contains parity of other disks per stripe.

If disk fails, use P and other disks to find missing information

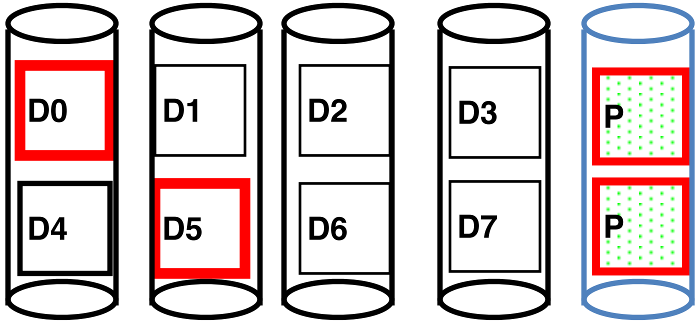
RAID 4: High I/O Rate Parity

- *Interleave data at the sector level (data block) rather than bit level - permits more parallelism (independent small reads, parallel large reads)*
- *Reconstruction of data can be done with single parity disk*
- *Reading (w/o) fault involve only data disk*
- *Writing involves data disk + parity disk*



Inspiration for RAID 5

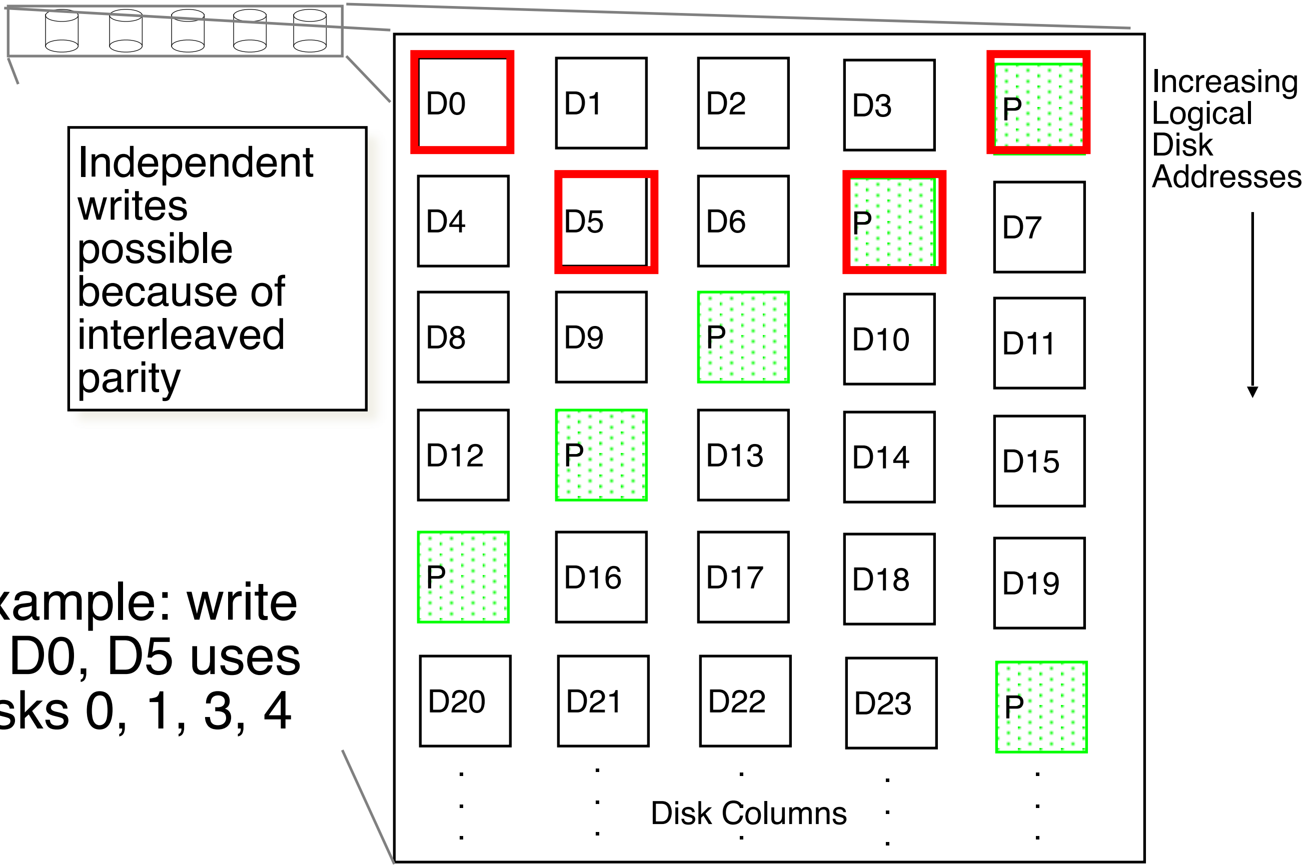
- RAID 4 works well for reads, but
- Parity Disk is the bottleneck for writes: Write to D0, D5 both also write to P disk



RAID 5: High I/O Rate Interleaved Parity

Independent writes possible because of interleaved parity

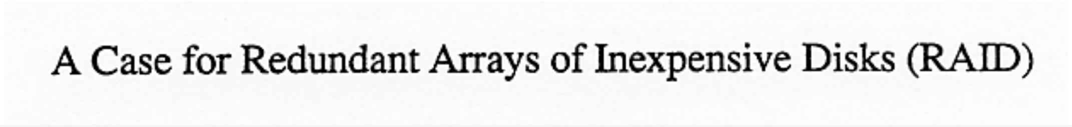
Example: write to D0, D5 uses disks 0, 1, 3, 4



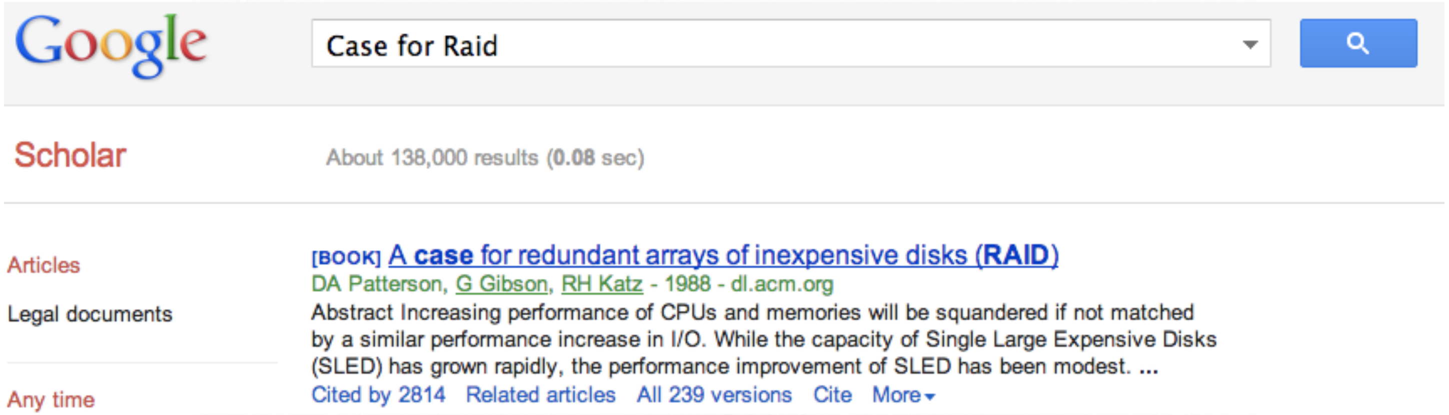
RAID 6

- RAID 5 is no longer the “gold standard”
- Can experience 1 disk failure and continue operation
  - RAID array is in a “degraded” state
- But disk failures are not actually independent!
  - When one disk has failed, there’s a decent chance another will fail soon
- RAID 6: Add another parity block per stripe
  - Now 2 blocks per stripe rather than 1
  - Sacrifice capacity for increased redundancy
  - Now the array can tolerate **2** disk failures and continue operating

Berkeley's Role in Definition of RAID (December 1987)



A Case for Redundant Arrays of Inexpensive Disks (RAID)



Google

Case for Raid

Scholar

About 138,000 results (0.08 sec)

Articles

Legal documents

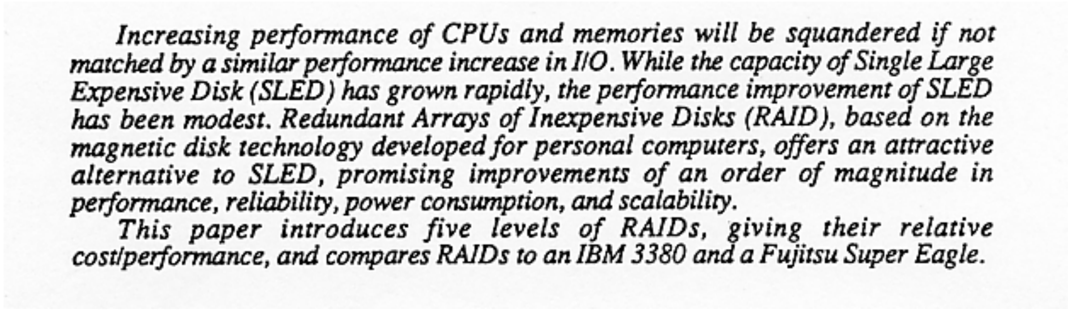
Any time

[BOOK] A case for redundant arrays of inexpensive disks (RAID)

DA Patterson, G Gibson, RH Katz - 1988 - dl.acm.org

Abstract Increasing performance of CPUs and memories will be squandered if not matched by a similar performance increase in I/O. While the capacity of Single Large Expensive Disks (SLED) has grown rapidly, the performance improvement of SLED has been modest. ...

Cited by 2814 Related articles All 239 versions Cite More▾



Increasing performance of CPUs and memories will be squandered if not matched by a similar performance increase in I/O. While the capacity of Single Large Expensive Disk (SLED) has grown rapidly, the performance improvement of SLED has been modest. Redundant Arrays of Inexpensive Disks (RAID), based on the magnetic disk technology developed for personal computers, offers an attractive alternative to SLED, promising improvements of an order of magnitude in performance, reliability, power consumption, and scalability.

This paper introduces five levels of RAIDs, giving their relative cost/performance, and compares RAIDs to an IBM 3380 and a Fujitsu Super Eagle.

RAID Version 1

- RAID-I (1989)
  - Consisted of a Sun 4/280 workstation with 128 MB of DRAM, four dual-string SCSI controllers, 28 5.25-inch SCSI disks and specialized disk striping software

RAID Version 2

- 1990-1993
- Early Network Attached Storage (NAS) System running a Log Structured File System (LFS)
- Impact:
  - $25 Billion/year in 2002
  - Over $150 Billion in RAID device sold since 1990-2002
  - 200+ RAID companies (at the peak)
  - Software RAID a standard component of modern OSs

RAID Is Not Enough By Itself

- You don't just have one disk die...
  - You can have more die in a short period of time
  - Thank both the "bathtub curve" and common environmental conditions
- If you care about your data, RAID isn't sufficient
  - You need to also consider a separate backup solution
- A good practice in clusters/warehouse scale computers:
  - RAID-6 in each cluster node with auto-failover and a hot spare
  - Distributed filesystem on top
    - Replicates amongst the cluster nodes so that nodes can fail
  - And then distribute to a different WSC...

In Conclusion …

- We have methods to mitigate faults in electronic systems:
  - Design bugs, Manufacturing defects, and Runtime Faults
- Dependability Measures let us quantify
- Dealing with Runtime Faults requires redundancy
  - either more hardware (cost) or more time (performance)
- Redundancy most commonly used in memory systems (DRAM, SRAM, Disks, SSD), also for communications