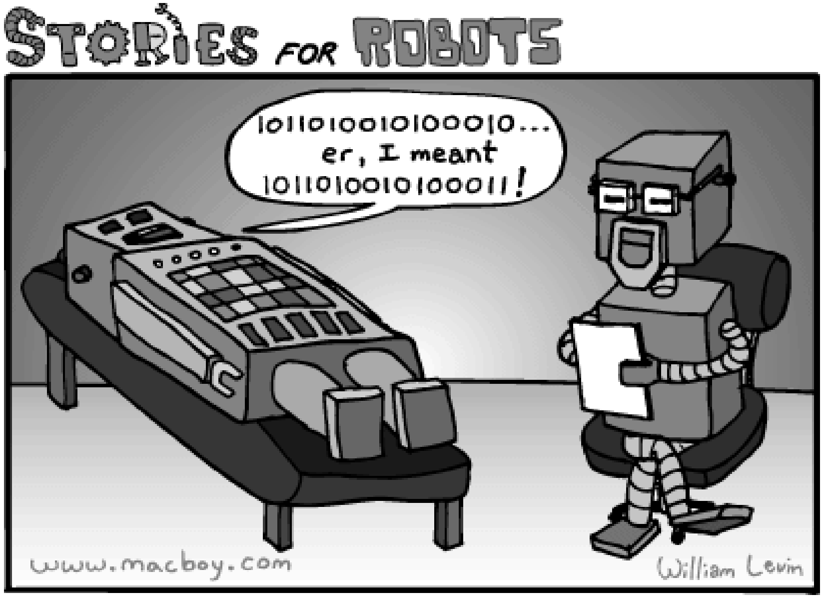
Fault-Tolerant Computing

Basic Concepts and Tools

About This Presentation

This presentation has been prepared for the graduate course ECE 257A (Fault-Tolerant Computing) by Behrooz Parhami, Professor of Electrical and Computer Engineering at University of California, Santa Barbara. The material contained herein can be used freely in classroom teaching or any other educational setting.

Edition	Released	Revised	Revised
First	Sep. 2006	Oct. 2007	



Droidian Slip

"I should get this remote control looked at."

Background and Motivation for Dependable Computing

The Curse of Complexity

Computer engineering is the art and science of translating user requirements we do not fully understand; into hardware and software we cannot precisely analyze; to operate in environments we cannot accurately predict; all in such a way that the society at large is given no reason to suspect the extent of our ignorance.[1]

Microsoft Windows NT (1992): ≈4M lines of code
Microsoft Windows XP (2002): ≈40M lines of code

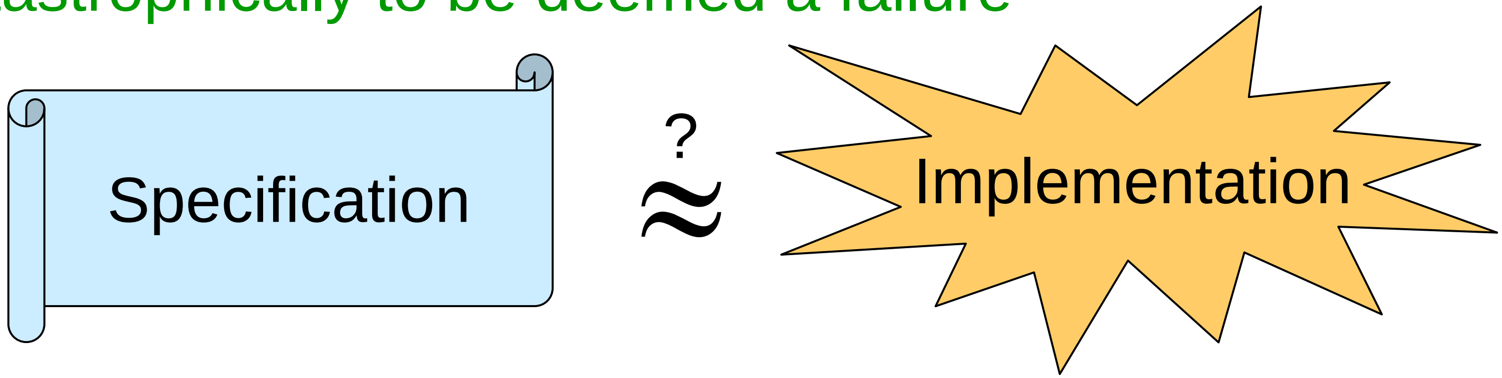
Intel Pentium processor (1993): ≈4M transistors
Intel Pentium 4 processor (2001): ≈40M transistors
Intel Itanium 2 processor (2002): ≈500M transistors

[1]Adapted from definition of structural engineering: Ralph Kaplan, *By Design: Why There Are No Locks on the Bathroom Doors in the Hotel Louis XIV and Other Object Lessons*, Fairchild Books, 2004, p. 229

Defining Failure

Failure is an unacceptable difference between expected and observed performance.[1]

A structure (building or bridge) need not collapse catastrophically to be deemed a failure



Reasons of typical Web site failures

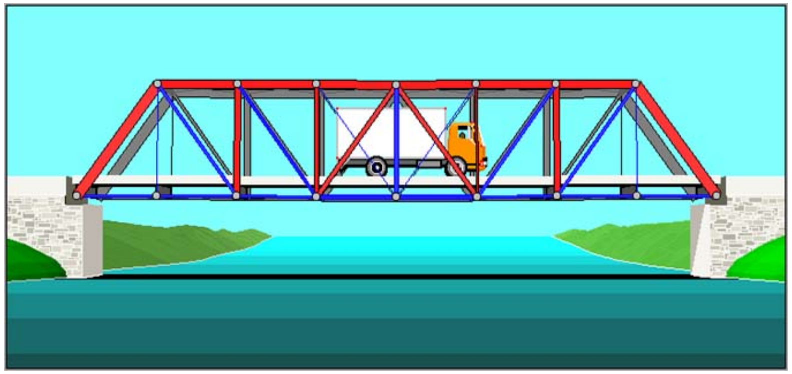
Hardware problems:	15%
Software problems:	34%
Operator error:	51%

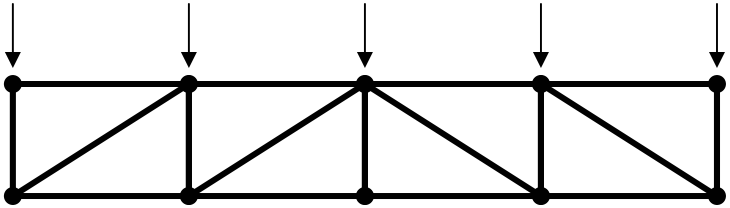
[1] Definition used by the Tech. Council on Forensic Engineering of the Amer. Society of Civil Engineers

Design Flaws: "To Engineer is Human"[1]

Complex systems almost certainly contain multiple design flaws

Redundancy in the form of safety factor is routinely used in buildings and bridges





Example of a more subtle flaw:
Disney Concert Hall in Los Angeles reflected light into nearby building, causing discomfort for tenants due to blinding light and high temperature

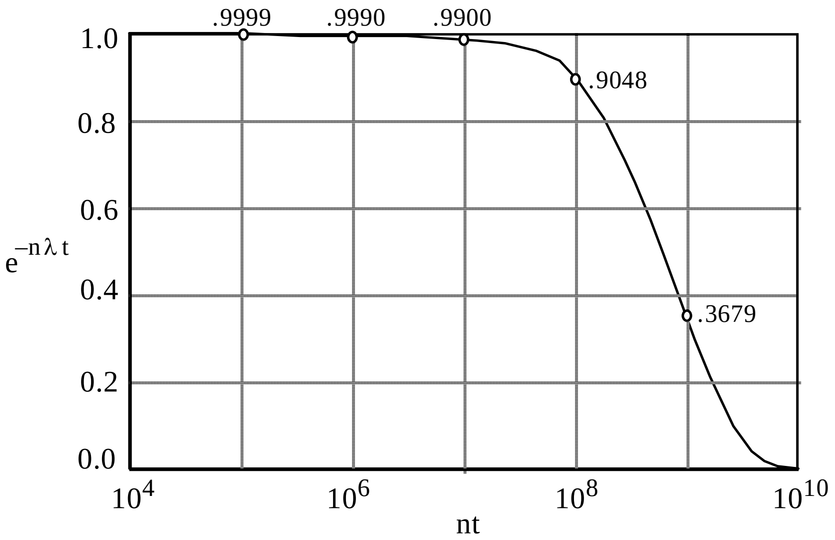
[1] Title of book by Henry Petroski

Concern for Computer System Dependability

The reliability argument

$\lambda = 10^{-9}$ per transistor per hour
Reliability formula $R(t) = e^{-n\lambda t}$

The on-board computer of a 10-year unmanned space mission can contain only $O(10^3)$ transistors if the mission is to have a 90% success probability



The safety argument

Airline's risk: $O(10^3)$ planes × $O(10^2)$ flights × 10^{-2} computer failures / 10 hr × 0.1 crash / failure × $O(10^2)$ deaths × $O(\$10^7)$ / death = \$ billions / yr

The availability argument

A central phone facility's down time should not exceed a few minutes / yr
Availability formula $A = 1/(n\lambda)$ →
Components $n = O(10^4)$, if we need 20-30 min for diagnosis and repair

Design Flaws in Computer Systems

Hardware example: Intel Pentium processor, 1994

For certain operands, the FDIV instruction yielded a wrong quotient
Amply documented and reasons well-known (overzealous optimization)

Software example: Patriot missile guidance, 1991

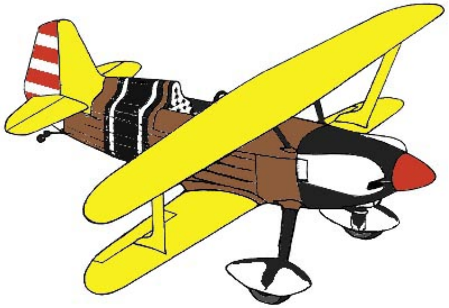
Missed intercepting a scud missile in 1st Gulf War, causing 28 deaths
Clock reading multiplied by 24-bit representation of 1/10 s (unit of time) caused an error of about 0.0001%; normally, this would cancel out in relative time calculations, but owing to ad hoc updates to some (not all) calls to a routine, calculated time was off by 0.34 s (over ≈100 hours), during which time a scud missile travels more than ½ km

User interface example: Therac 25 machine, mid 1980s[1]

Serious burns and some deaths due to overdose in radiation therapy
Operator entered "x" (for x-ray), realized error, corrected by entering "e" (for low-power electron beam) before activating the machine; activation was so quick that software had not yet processed the override

[1] Accounts of the reasons vary

Learning Curve: “Normal Accidents”[1]



Example: Risk of piloting a plane

1903	First powered flight
1908	First fatal accident
1910	Fatalities = 32 (≈2000 pilots worldwide)
1918	US Air Mail Service founded Pilot life expectancy = 4 years 31 of the first 40 pilots died in service
1922	One forced landing for every 20 hours of flight
Today	Commercial airline pilots pay normal life insurance rates

Unfortunately, the learning curve for computers and computer-based systems is not as impressive

[1] Title of book by Charles Perrow (Ex. p. 125)

Mishaps, Accidents, and Catastrophes

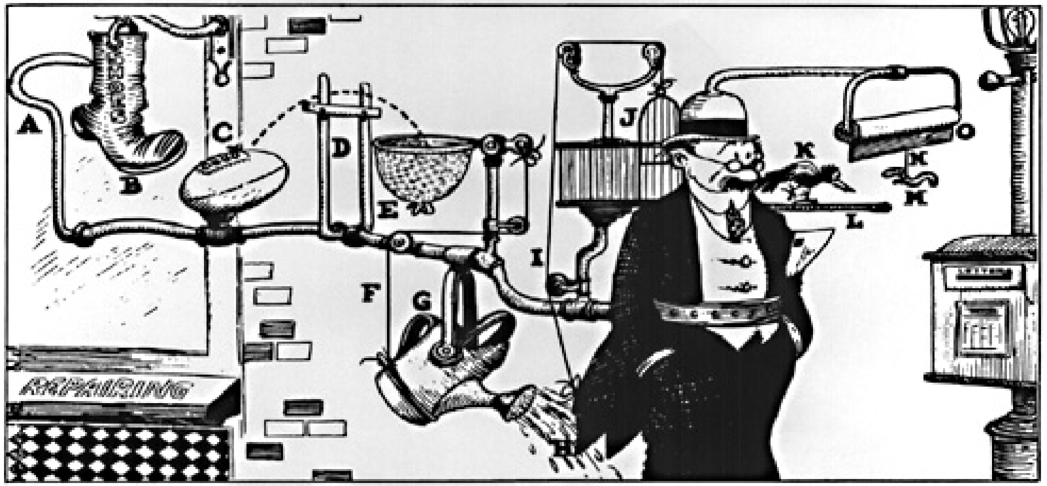
Mishap: misfortune; unfortunate accident

Accident: unexpected (no-fault) happening causing loss or injury

Catastrophe: final, momentous event of drastic action; utter failure

At one time (following the initial years of highly unreliable hardware), computer mishaps were predominantly the results of human error

Now, most mishaps are due to complexity (unanticipated interactions)



Keep You From Forgetting To Mail Your Wife's Letter RUBE GOLDBERG (tm) RGI 049

Rube Goldberg contraptions

The butterfly effect

A Problem to Think About: Perils of Modeling

In a passenger plane, the failure rate of the cabin pressurizing system is 10^{-5}/hr (loss of cabin pressure occurs once per 10^5 hours of flight)

Failure rate of the oxygen-mask deployment system is also 10^{-5}/hr

Assuming failure independence, both systems fail at a rate of 10^{-10}/hr

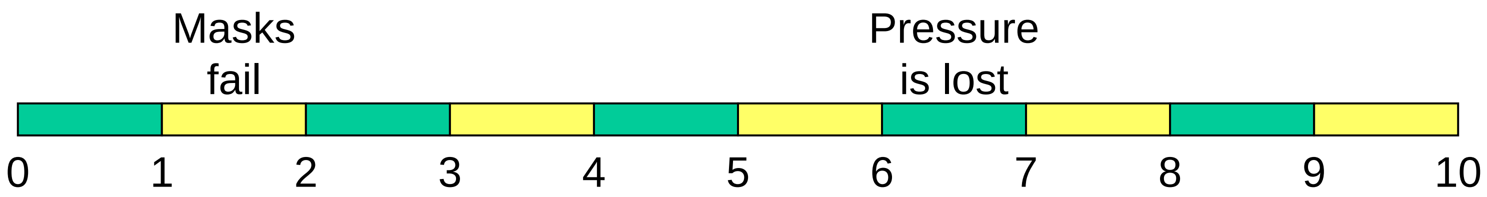
Fatality probability for a 10-hour flight is about $10^{-10} \times 10 = 10^{-9}$ (10^{-9} or less is generally deemed acceptable)

Probability of death in a car accident is $\approx$1/6000 per year ($>10^{-7}$/hr)

Alternate reasoning

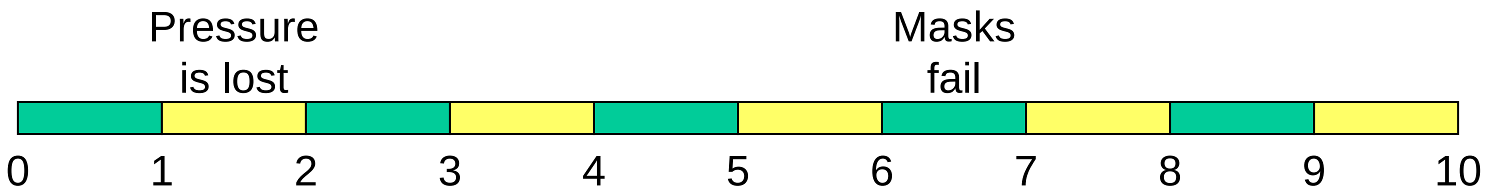
Probability of cabin pressure system failure in 10-hour flight is 10^{-4}
Probability of oxygen masks failing to deploy in 10-hour flight is 10^{-4}
Probability of both systems failing in 10-hour flight is 10^{-8}
Why is this result different from that of our earlier analysis (10^{-9})?
Which one is correct?

Cabin Pressure and Oxygen Masks



When we multiply the two per-hour failure rates and then take the flight duration into account, we are assuming that only the failure of the two systems within the same hour is catastrophic

This produces an optimistic reliability estimate $(1 – 10^{–9})$



When we multiply the two flight-long failure rates, we are assuming that the failure of these systems would be catastrophic at any time

This produces a pessimistic reliability estimate $(1 – 10^{–8})$

Causes of Human Errors in Computer Systems

1. Personal factors (35%): Lack of skill, lack of interest or motivation, fatigue, poor memory, age or disability

2. System design (20%): Insufficient time for reaction, tedium, lack of incentive for accuracy, inconsistent requirements or formats

3. Written instructions (10%): Hard to understand, incomplete or inaccurate, not up to date, poorly organized

4. Training (10%): Insufficient, not customized to needs, not up to date

5. Human-computer interface (10%): Poor display quality, fonts used, need to remember long codes, ergonomic factors

6. Accuracy requirements (10%): Too much expected of operator

7. Environment (5%): Lighting, temperature, humidity, noise

Because "the interface is the system" (according to a popular saying), items 2, 5, and 6 (40%) could be categorized under user interface

Properties of a Good User Interface

1. Simplicity: Easy to use, clean and unencumbered look

2. Design for error: Makes errors easy to prevent, detect, and reverse; asks for confirmation of critical actions

3. Visibility of system state: Lets user know what is happening inside the system from looking at the interface

4. Use of familiar language: Uses terms that are known to the user (there may be different classes of users, each with its own vocabulary)

5. Minimal reliance on human memory: Shows critical info on screen; uses selection from a set of options whenever possible

6. Frequent feedback: Messages indicate consequences of actions

7. Good error messages: Descriptive, rather than cryptic

8. Consistency: Similar/different actions produce similar/different results and are encoded with similar/different colors and shapes

Operational Errors in Computer Systems

Hardware examples

Permanent incapacitation due to shock, overheating, voltage spike
Intermittent failure due to overload, timing irregularities, crosstalk
Transient signal deviation due to alpha particles, external interference

Software examples

These can also be classified as design errors

Counter or buffer overflow
Out-of-range, unreasonable, or unanticipated input
Unsatisfied loop termination condition

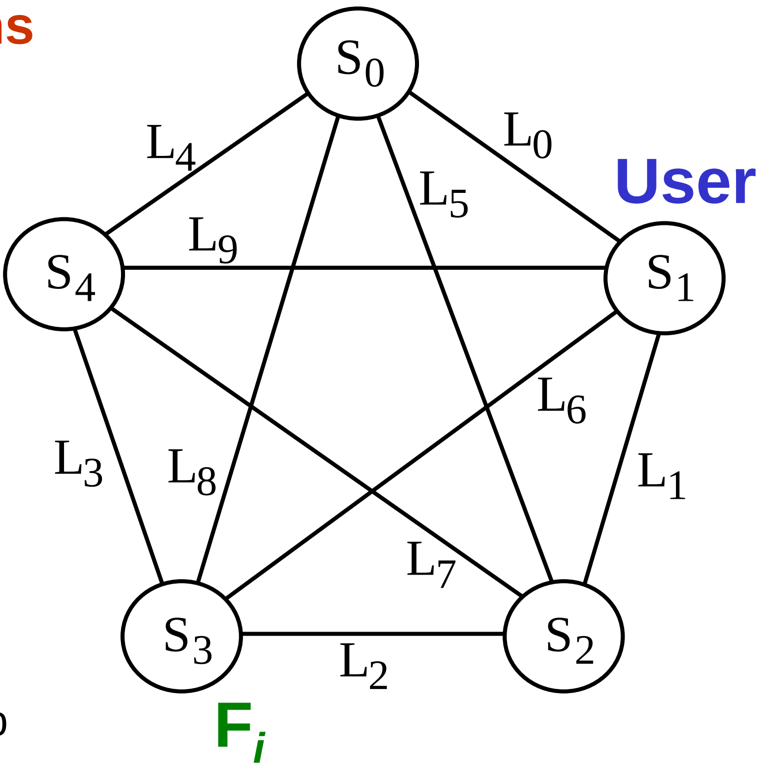
Dec. 2004: "Comair runs a 15-year old scheduling software package from SBS International (www.sbsint.com). The software has a hard limit of 32,000 schedule changes per month. With all of the bad weather last week, Comair apparently hit this limit and then was unable to assign pilots to planes."
It appears that they were using a 16-bit integer format to hold the count.

June 1996: Explosion of the Ariane 5 rocket 37 s into its maiden flight was due to a silly software error. For an excellent exposition of the cause, see: http://www.comp.lancs.ac.uk/computing/users/dixa/teaching/CSC221/ariane.pdf)

A Motivating Case Study

Data availability and integrity concerns

Distributed DB system with 5 sites
Full connectivity, dedicated links
Only direct communication allowed
Sites and links may malfunction
Redundancy improves availability



S: Probability of a site being available
L: Probability of a link being available

Single-copy availability = SL
Unavailability = $1 - SL$
= $1 - 0.99 \times 0.95 = 5.95\%$

Data replication methods, and a challenge

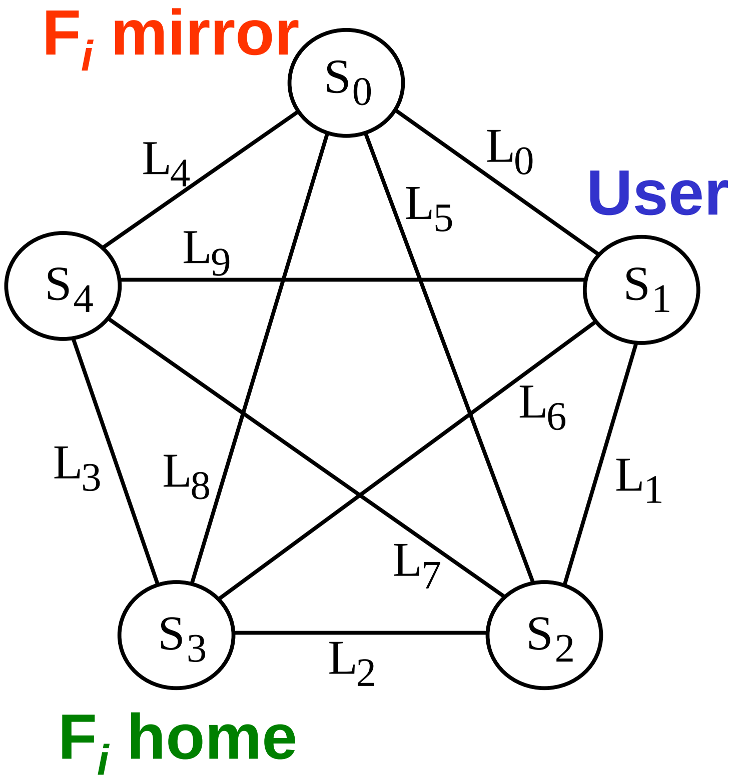
File duplication: home / mirror sites
File triplication: home / backup 1 / backup 2
Are there availability improvement methods with less redundancy?

Data Duplication: Home and Mirror Sites

S: Site availability e.g., 99%
L: Link availability e.g., 95%

$A = SL + (1 - SL)SL$

- SL: Primary site can be reached
- $(1 - SL)$: Primary site inaccessible
- SL: Mirror site can be reached



Duplicated availability $= 2SL - (SL)^2$
Unavailability $= 1 - 2SL + (SL)^2$
$= (1 - SL)^2 = 0.35\%$

Data unavailability reduced from 5.95% to 0.35%

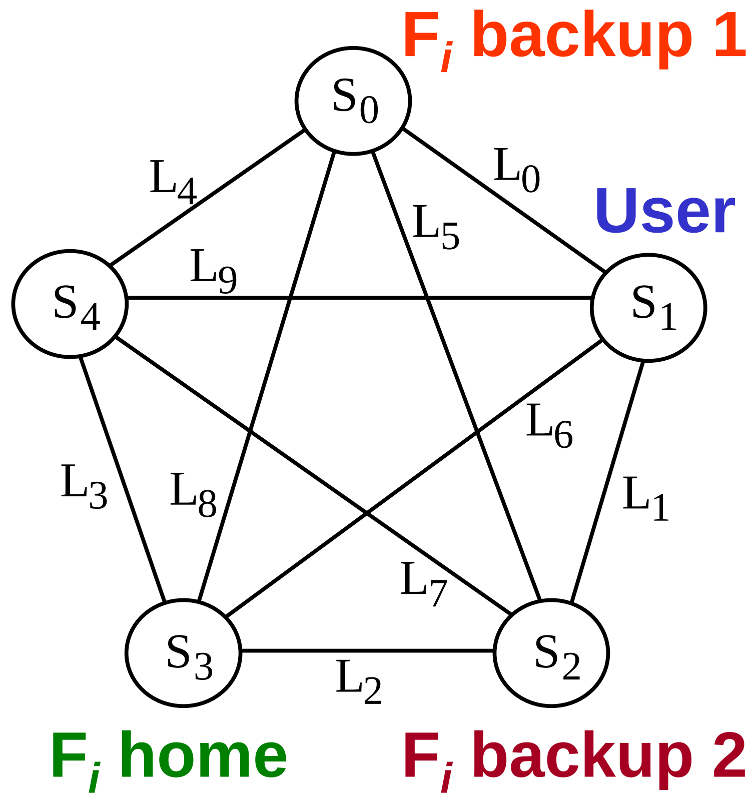
Availability improved from ≈ 94% to 99.65%

Data Triplication: Home and Two Backups

S: Site availability — e.g., 99%
L: Link availability — e.g., 95%

$$A = SL + (1 - SL)SL + (1 - SL)^2 SL$$

- SL: Primary site can be reached
- $(1 - SL)$: Primary site inaccessible
- SL: Backup 1 can be reached
- $(1 - SL)^2$: Primary and backup 1 inaccessible
- SL: Backup 2 can be reached



Triplicated avail. $= 3SL - 3(SL)^2 - (SL)^3$
Unavailability $= 1 - 3SL - 3(SL)^2 + (SL)^3$
$= (1 - SL)^3 = 0.35\%$

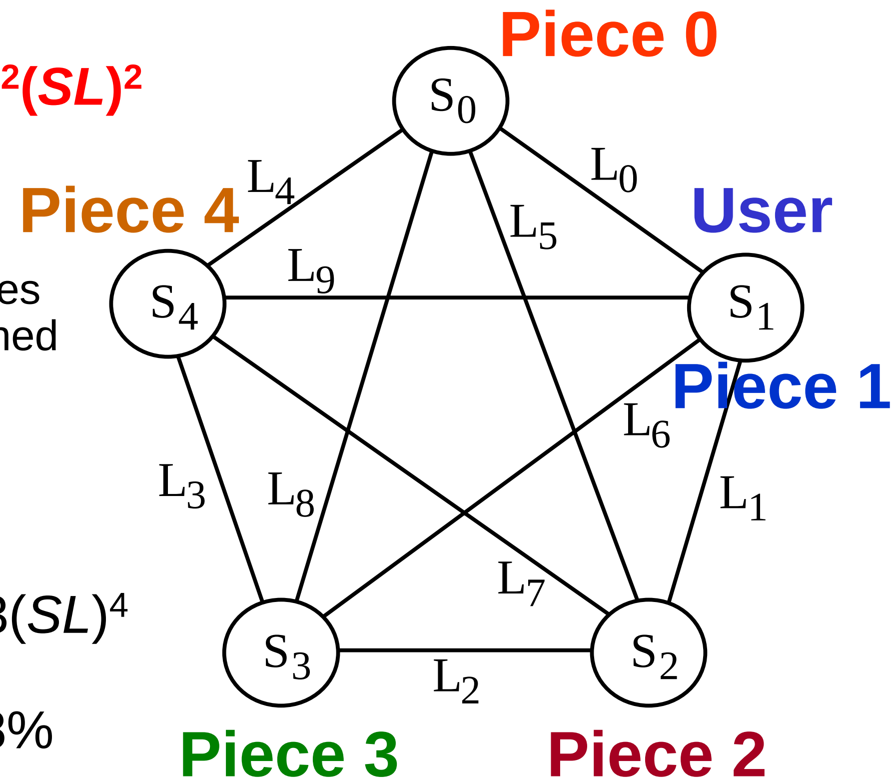
Data unavailability reduced from 5.95% to 0.02%

Availability improved from ≈ 94% to 99.98%

Data Dispersion: Three of Five Pieces

$A = (SL)^4 + 4(1 - SL)(SL)^3 + 6(1 - SL)^2(SL)^2$

All 4 pieces can be reached | Exactly 3 pieces can be reached | Only 2 pieces can be reached

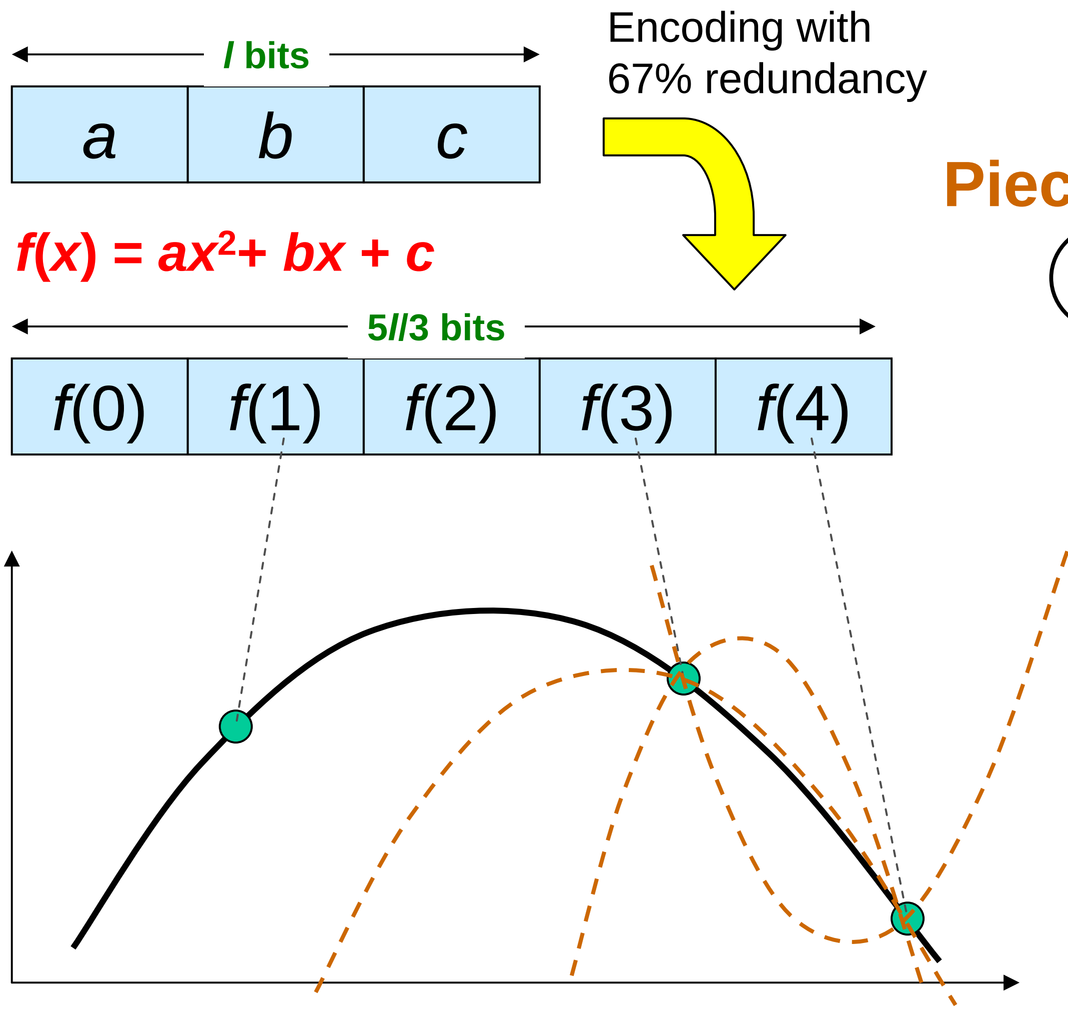


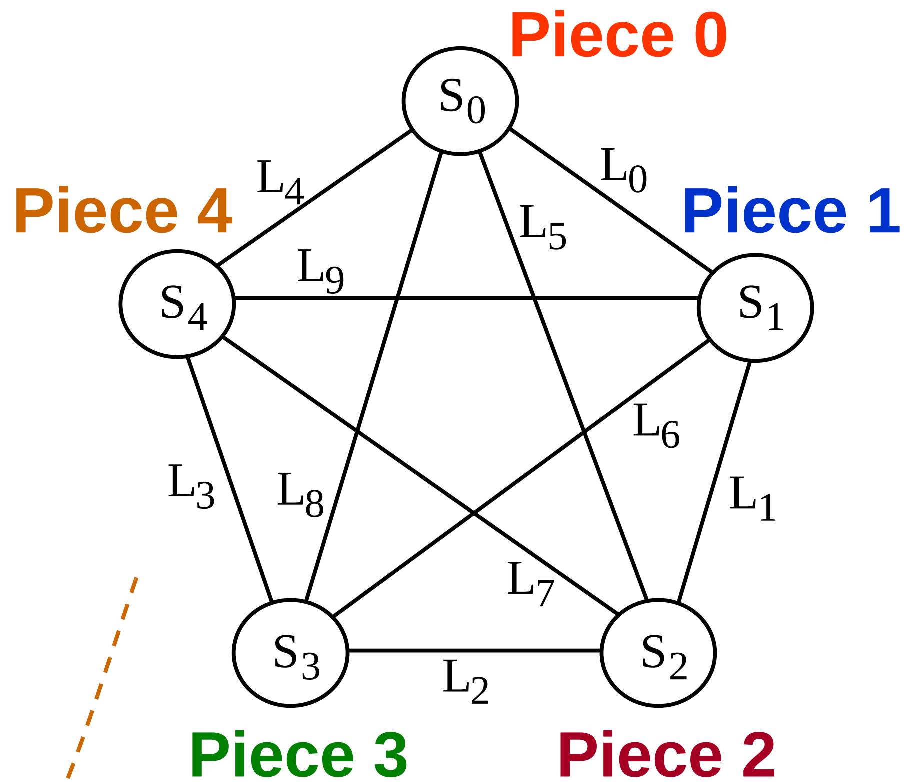
S: Site availability e.g., 99%
L: Link availability e.g., 95%

Dispersed avail. = $6(SL)^2 - 8(SL)^3 + 3(SL)^4$
Availability = 99.92%
Unavailability = 1 – Availability = 0.08%

Scheme →	Nonredund.	Duplication	Triplication	Dispersion
Unavailability	5.95%	0.35%	0.02%	0.08%
Redundancy	0%	100%	200%	67%

Dispersion for Data Security and Integrity





Note that two pieces would be inadequate for reconstruction

Questions Ignored in Our Simple Example

1. How redundant copies of data are kept consistent
When a user modifies the data, how to update the redundant copies (pieces) quickly and prevent the use of stale data in the meantime?

2. How malfunctioning sites and links are identified
Malfunction diagnosis must be quick to avoid data contamination

3. How recovery is accomplished when a malfunctioning site / link returns to service after repair
The returning site must be brought up to date with regard to changes

4. How data corrupted by the actions of an adversary is detected
This is more difficult than detecting random malfunctions

The example does demonstrate, however, that:

- Many alternatives are available for improving dependability
- Proposed methods must be assessed through modeling
- The most cost-effective solution may be far from obvious