Dependable Computing

A Multilevel Approach

Behrooz Parhami

University of California, Santa Barbara

STRUCTURE AT A GLANCE

Part I — Introduction: Dependable Systems (The Ideal-System View)	Goals / Models	1. Background and Motivation 2. Dependability Attributes 3. Combinational Modeling 4. State-Space Modeling
Part II — Defects: Physical Imperfections (The Device-Level View)	Methods / Examples	5. Defect Avoidance 6. Defect Circumvention 7. Shielding and Hardening 8. Yield Enhancement
Part III — Faults: Logical Deviations (The Circuit-Level View)	Methods / Examples	9. Fault Testing 10. Fault Masking 11. Design for Testability 12. Replication and Voting
Part IV — Errors: Informational Distortions (The State-Level View)	Methods / Examples	13. Error Detection 14. Error Correction 15. Self-Checking Modules 16. Redundant Disk Arrays
Part V — Malfunctions: Architectural Anomalies (The Structure-Level View)	Methods / Examples	17. Malfunction Diagnosis 18. Malfunction Tolerance 19. Standby Redundancy 20. Resilient Algorithms
Part VI — Degradations: Behavioral Lapses (The Service-Level View)	Methods / Examples	21. Degradation Allowance 22. Degradation Management 23. Robust Task Scheduling 24. Software Redundancy
Part VII — Failures: Computational Breaches (The Result-Level View)	Methods / Examples	25. Failure Confinement 26. Failure Recovery 27. Agreement and Adjudication 28. Fail-Safe System Design

Appendix: Past, Present, and Future

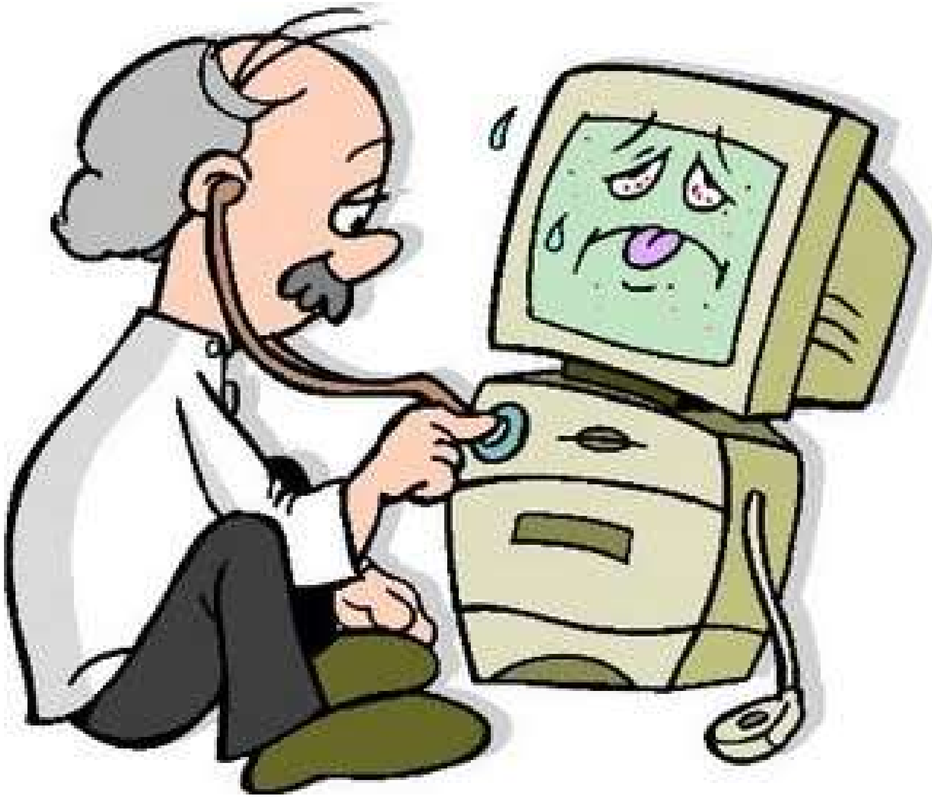
About This Presentation

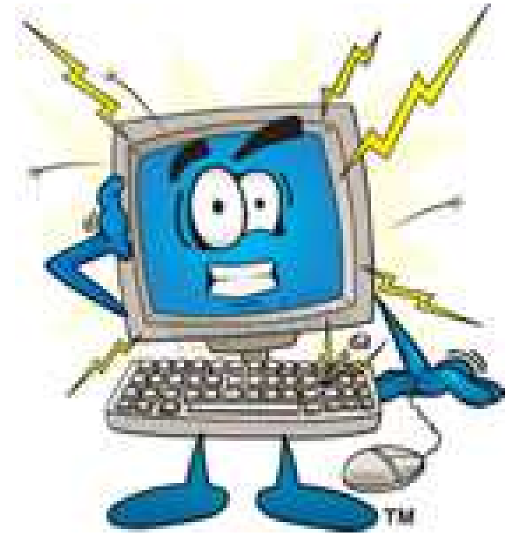
This presentation is intended to support the use of the textbook *Dependable Computing: A Multilevel Approach* (traditional print or on-line open publication, TBD). It is updated regularly by the author as part of his teaching of the graduate course ECE 257A, Fault-Tolerant Computing, at Univ. of California, Santa Barbara. Instructors can use these slides freely in classroom teaching or for other educational purposes. Unauthorized uses, including distribution for profit, are strictly prohibited. © Behrooz Parhami

Edition	Released	Revised	Revised	Revised	Revised
First	Sep. 2006	Oct. 2007	Oct. 2009	Oct. 2012	Sep. 2013
		Jan. 2015	Sep. 2015	Sep. 2018	Sep. 2019
		Sep.02023	Oct. 2023		

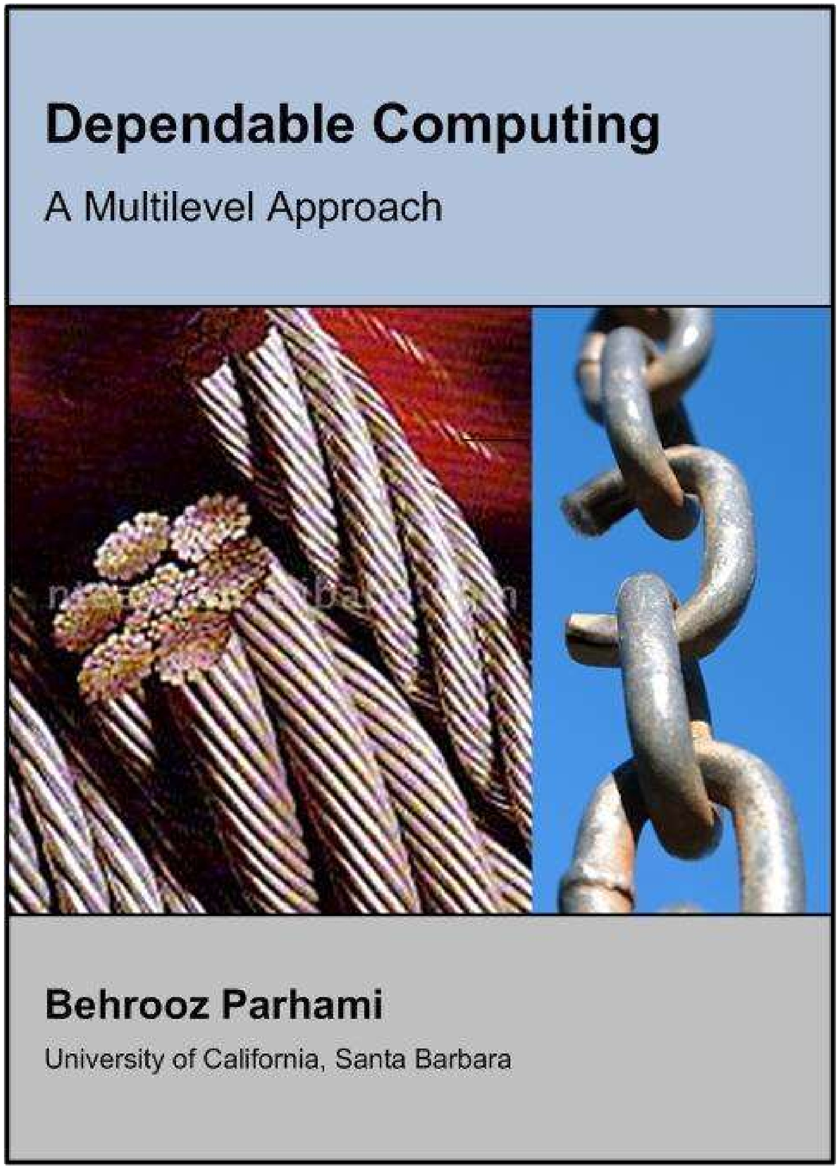
ECE 257A: Fault-Tolerant Computing

Course Introduction





How the Cover Image Relates to Our Course



Dependability as weakest-link attribute:
Under stress, the weakest link will break, even if all other links are superstrong

- Improve the least reliable part first

Safety factor (use of redundancy):
Provide more resources than needed for the minimum acceptable functionality

Additional resources not helpful if:

- failures are not independent
- Some critical component fails

About the Name of This Course

Fault-tolerant computing: a discipline that began in the late 1960s – 1st Fault-Tolerant Computing Symposium (FTCS) was held in 1971

In the early 1980s, the name “dependable computing” was proposed for the field to account for the fact that tolerating faults is but one approach to ensuring reliable computation. The terms “fault tolerance” and “fault-tolerant” were so firmly established, however, that people started to use “dependable and fault-tolerant computing.”

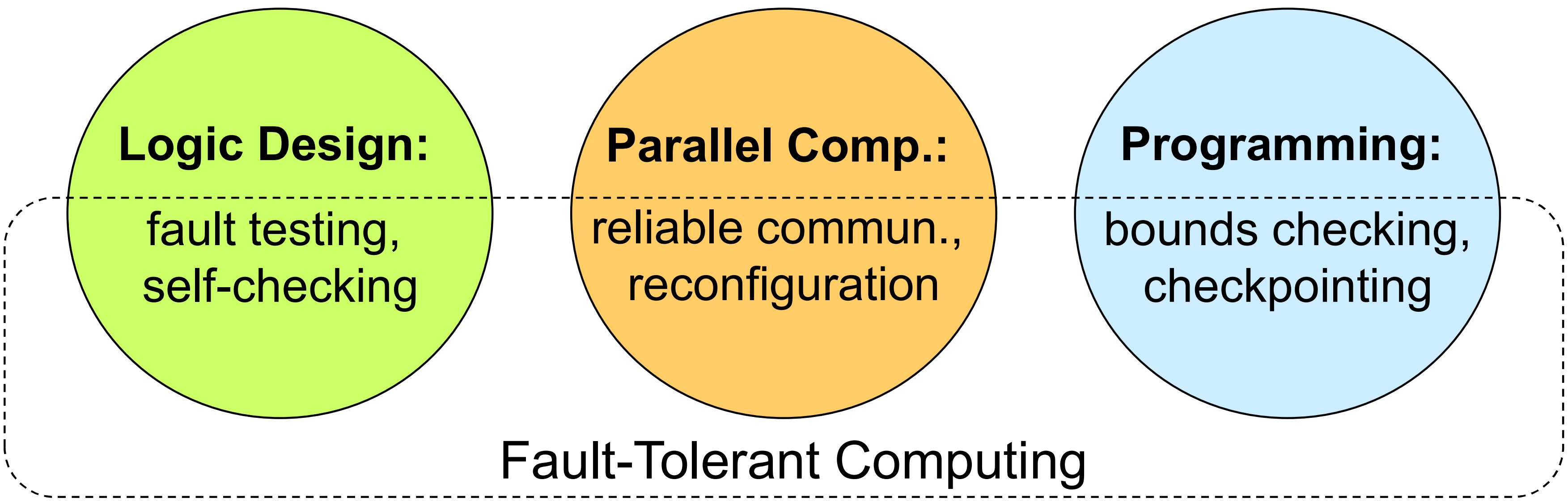
In 2000, the premier conference of the field was merged with another and renamed “Int’l Conf. on Dependable Systems and Networks” (DSN)

In 2004, IEEE began the publication of *IEEE Trans. On Dependable and Secure Systems* (inclusion of the term “secure” is for emphasis, because security was already accepted as an aspect of dependability)

Why This Course Shouldn't Be Needed

In an ideal world, methods for dealing with faults, errors, and other impairments in hardware and software would be covered within every computer engineering course that has a design component

Analogy: We do not teach structural engineers about building bridges in one course and about bridge safety and structural integrity during high winds or earthquakes in another (optional) course



Brief History of Dependable Computing

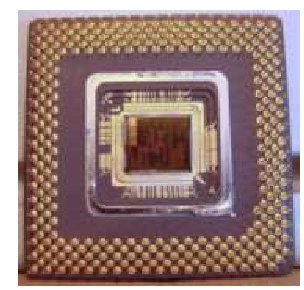
1940s: ENIAC, with 17.5K vacuum tubes and 1000s of other electrical elements, failed once every 2 days (avg. down time = minutes)

1950s: Early ideas by von Neumann (multichannel, with voting) and Moore-Shannon ("crummy" relays)

1960s: NASA and military agencies supported research for long-life space missions and battlefield computing

1970s: The field developed quickly (international conference, many research projects and groups, experimental systems)

1980s: The field matured (textbooks, theoretical developments, use of ECCs in solid-state memories, RAID concept), but also suffered some loss of focus and interest because of the extreme reliability of integrated circuits



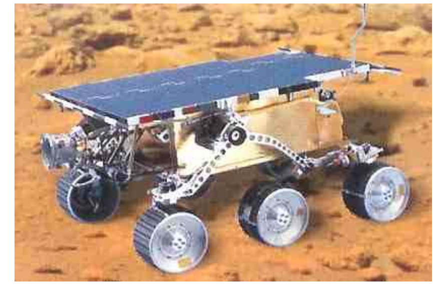
1990s: Increased complexity at chip and system levels made verification, testing, and testability prime study topics

2000s: Resurgence of interest owing to less reliable fabrication at ultrahigh densities and "crummy" nanoelectronic components

2010s: Integration of reliability, safety, privacy, and security concerns, particularly in the cloud, artificial intelligence systems, and IoT

Dependable Computing in the 2020s

There are still ambitious projects; space and elsewhere



Harsh environments (vibration, pressure, temperatures)
External influences (radiation, micrometeoroids)
Need for autonomy (commun. delays, unmanned probes)
Life & death situations (transportation, self-driving cars)

The need is expanding

More complex systems (supercomputers in our pockets)
Critical applications (medicine, transportation, finance)
Expanding pool of unsophisticated users
Continued rise in maintenance costs
Digital-only data (needs more rigorous backup)

The emphasis is shifting

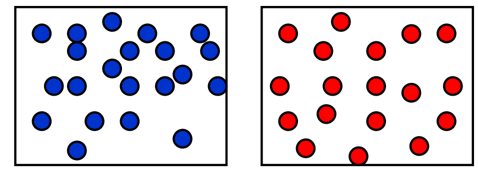
COTS-based hardware, with software assist
Integrated HW/SW/firmware systems-on-chip
Swarms of units with disposable subsystems
Fairness, equity, and social-justice concerns

Pretest: Failures and Probabilities

This test will not be graded or even collected, so answer the test questions truthfully and to the best of your ability / knowledge

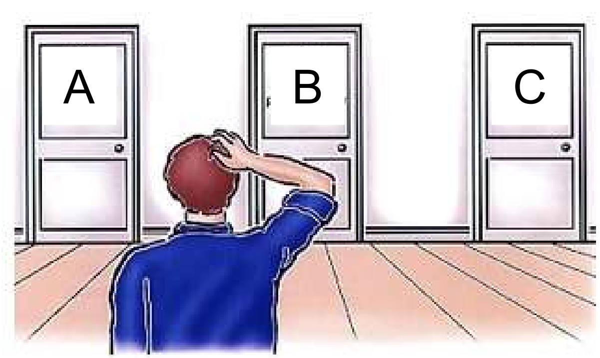
Question 1: Name a disaster that was caused by computer hardware or software failure. How do you define “disaster” and “failure”?

Question 2: Which of these patterns is more random?



Question 3: Which do you think is more likely: the event that everyone in this class was born in the first half of the year or the event that at least two people were born on the same day of the year?

Question 4: In a game show, there is a prize behind one of 3 doors with equal probabilities. You pick Door A. The host opens Door B to reveal that there is no prize behind it. The host then gives you a chance to switch to Door C. Is it better to switch or to stick to your choice?



Pretest (Continued): Causes of Mishaps

Question 5: Does this photo depict a mishap due to design flaw, implementation bug, procedural inadequacies, or human error?

Pretest (Continued): Reliability and Risk

Question 6: Name an emergency backup system (something not normally used unless another system fails) that is quite commonplace

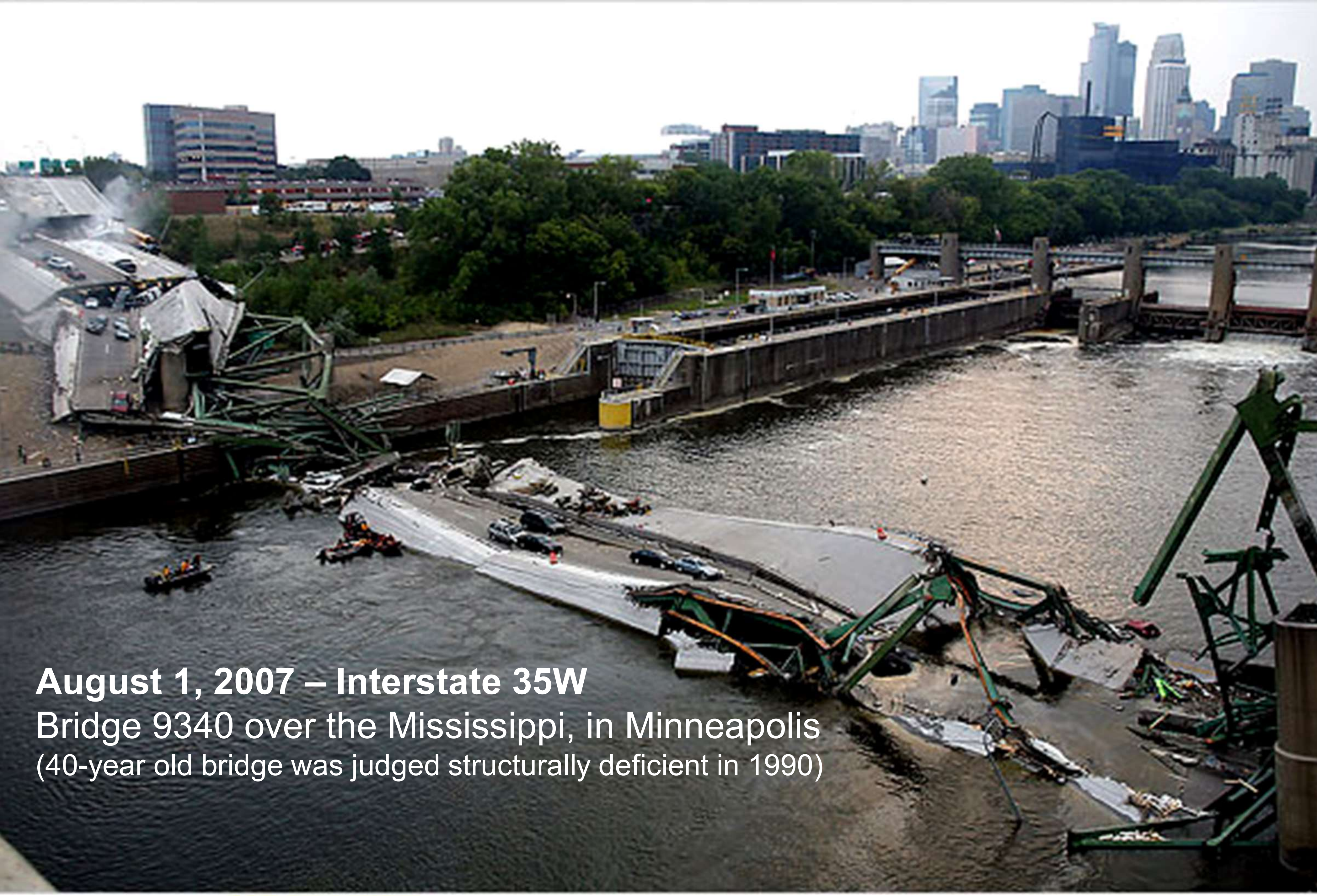
Question 7: Which is more reliable: Plane X or Plane Y that carries four times as many passengers as Plane X and is twice as likely to crash?

Question 8: Which is more reliable: a 4-wheel vehicle with one spare tire or an 18-wheeler with 2 spare tires?

Question 9: Which surgeon would you prefer for an operation that you must undergo: Surgeon A, who has performed some 500 operations of the same type, with 5 of his patients perishing during or immediately after surgery, or Surgeon B, who has a perfect record in 25 operations?

Question 10: Which is more probable at your home or office: a power failure or an Internet outage? Which is likely to last longer?

If you had trouble with 3 or more questions, you really need this course!



August 1, 2007 – Interstate 35W

Bridge 9340 over the Mississippi, in Minneapolis

(40-year old bridge was judged structurally deficient in 1990)

History of Bridge 9340 in Minneapolis

1967: Opens to traffic

1990: Dept. of Transportation classifies bridge as “structurally deficient”

1993: Inspection frequency doubled to yearly

1999: Deck and railings fitted with de-icing system

2001: U. Minn. engineers deem bridge struc. deficient

2004-07: Fatigue potential and remedies studied

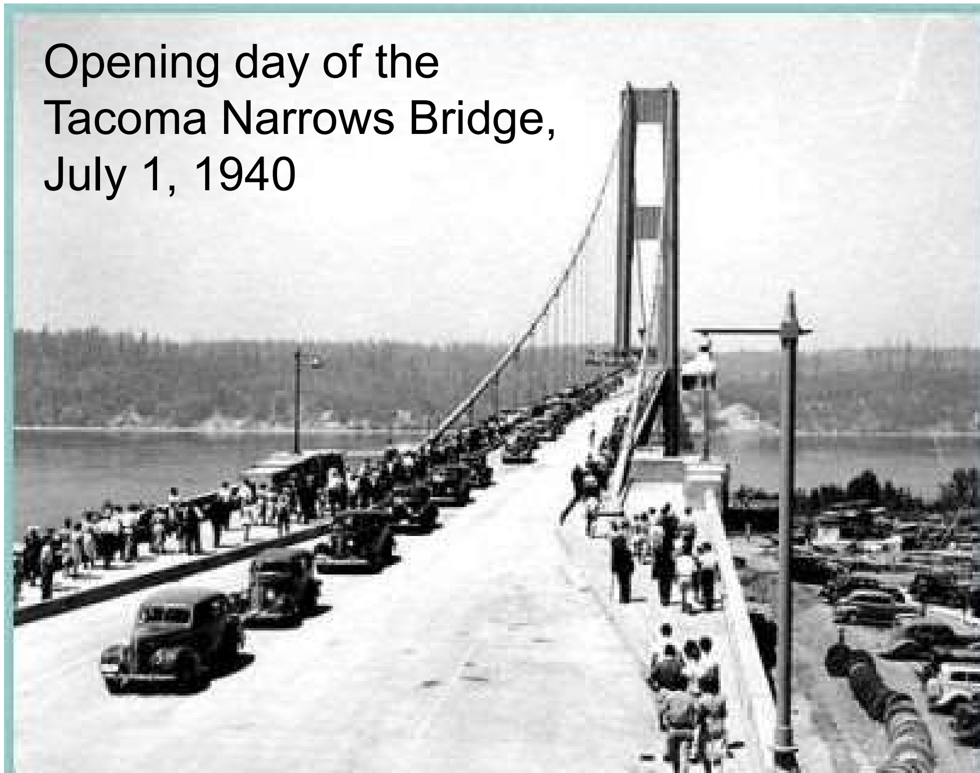
2007: Inspection plan chosen over reinforcements

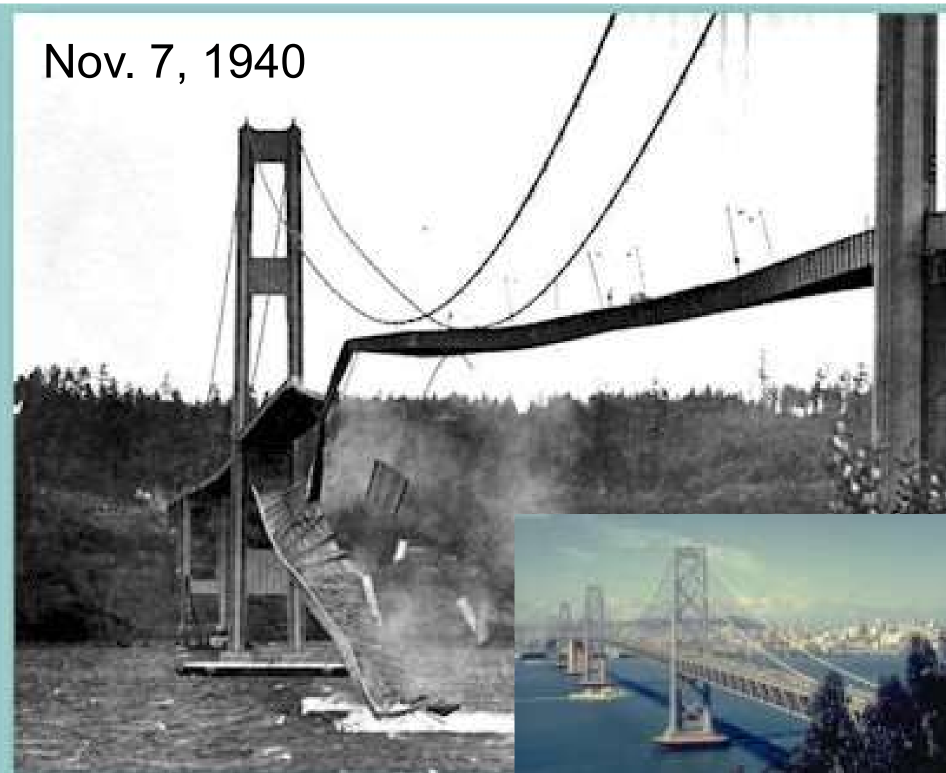
Summer 2007: $2.4M of repairs/maintenance on deck, lights, joints

Aug. 1, 2007: Collapses at 6:05 PM, killing 7

Sep. 18, 2008: Replacement bridge opens

What Do We Learn from Bridges that Collapse?





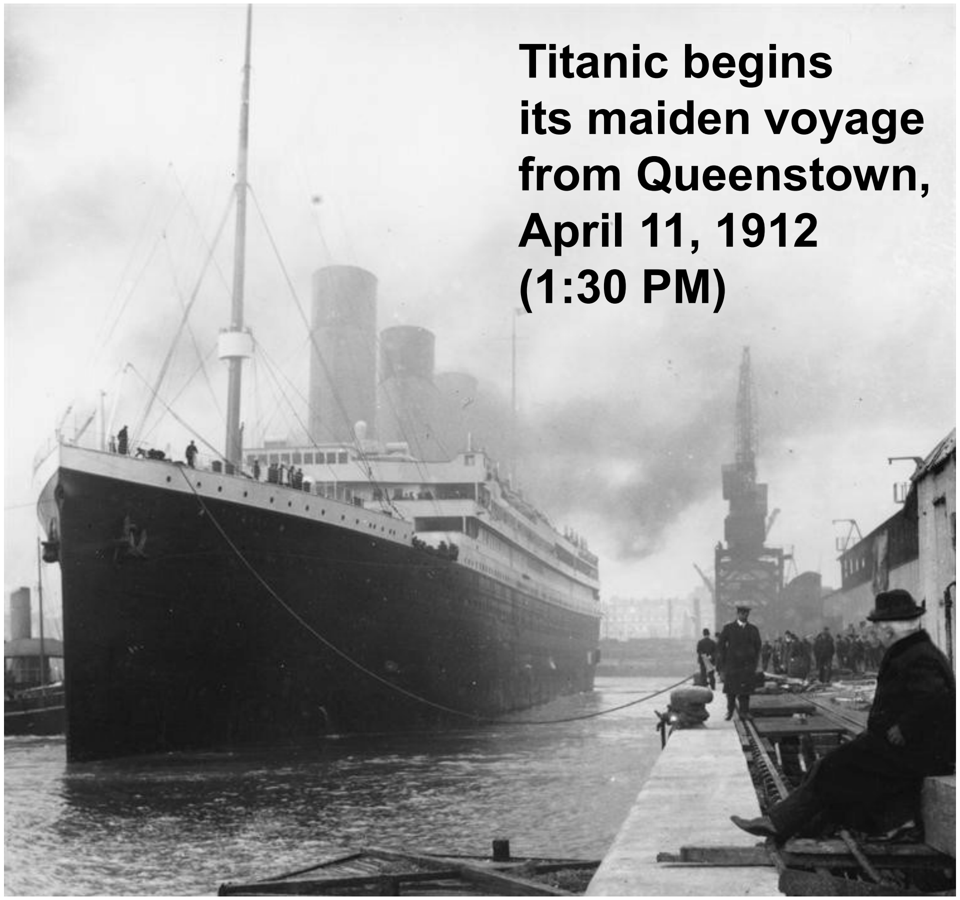
One catastrophic bridge collapse every 30 years or so

See the following amazing video clip (Tacoma Narrows Bridge):
http://www.enm.bris.ac.uk/research/nonlinear/tacoma/tacnarr.mpg

" . . . failures appear to be inevitable in the wake of prolonged success, which encourages lower margins of safety. Failures in turn lead to greater safety margins and, hence, new periods of success."

Henry Petroski, *To Engineer is Human*

. . . or from “Unsinkable” Ships that Sink?



“The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong, it usually turns out to be impossible to get at or repair.”

Douglas Adams, author of *The Hitchhiker’s Guide to the Galaxy*

… or from Poorly Designed High-Tech Trains?

Train built for demonstrating magnetic levitation technology in northwest Germany rams into maintenance vehicle left on track at 200 km/h, killing 23 of 29 aboard

Official investigation blames the accident on human error (train was allowed to depart before a clearance phone call from maintenance crew)

Not a good explanation; even low-tech trains have obstacle detection systems

Even if manual protocol is fully adequate under normal conditions, any engineering design must take unusual circumstances into account (abuse, sabotage, terrorism)

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 0.5 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

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


NO! - Bad User!!!
You've been warned 3 times that this file does not exist.
Now you've made us catch this worthless exception and we're upset.
Do not do this again.
OK


Big error
You really screwed up this time.
I know
What else is new
It's my first day

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

Example from THE RISKS DIGEST

Forum on Risks to the Public in Computers and Related Systems
http://catless.ncl.ac.uk/Risks/
(Peter G. Neumann, moderator)

On August 17, 2006, a class-two incident occurred at the Swedish atomic reactor Forsmark. A short-circuit in the electricity network caused a problem inside the reactor and it needed to be shut down immediately, using emergency backup electricity.

However, in two of the four generators, which run on AC, the AC/DC converters died. The generators disconnected, leaving the reactor in an unsafe state and the operators unaware of the current state of the system for approximately 20 minutes.

A meltdown, such as the one in Chernobyl, could have occurred.

Coincidence of problems in multiple protection levels seems to be a recurring theme in many modern-day mishaps -- emergency systems had not been tested with the grid electricity being off

Worst Stock Market Computer Failure

April 5, 2000: Computer failure halts the trading for nearly 8 hours at the London Stock Exchange on its busiest day (end of financial year)

Firms and individual investors prevented from buying or selling stocks to minimize their capital gains taxes

Delaying end of financial year was considered, but not implemented; eventually, the system became operational at 3:45 PM and trading was allowed to continue until 8:00 PM

London Stock Exchange confirmed it had a fault in its electronic feed that sends the prices to dealers, but it gave no further explanation

A spokesman said the problems were "very technical" and involved corrupt data

A Few News Items in THE RISKS DIGEST

February 2012: Programming Error Doomed Russian Mars Probe

Fails to escape earth orbit due to simultaneous reboot of two subsystems

March 2012: Eighteen Companies Sued over Mobile Apps

Facebook, Apple, Twitter, and Yelp are among the companies sued over gathering data from the address books of millions of smartphone users

May 2012: Automatic Updates Considered Zombieware

Software updates take up much time/space; no one knows what's in them

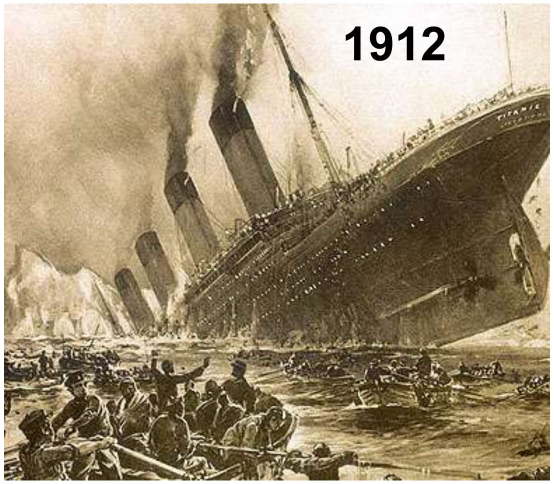
July 2012: A320 Lost 2 of 3 Hydraulic Systems on Takeoff

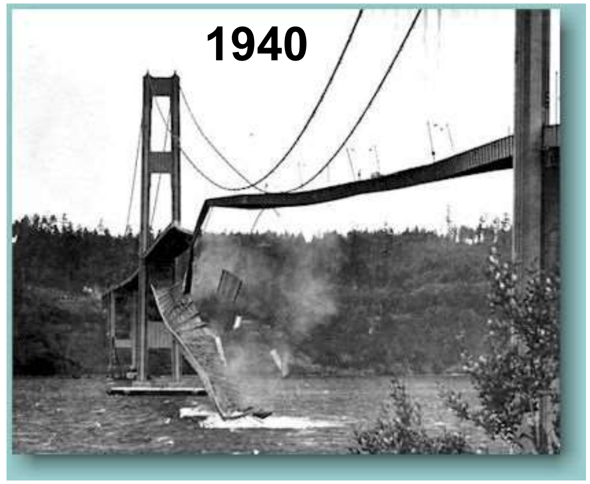
No loss of life; only passenger discomfort. Full account of incident not yet available, but it shows that redundancy alone is not sufficient protection

September 2013: No password Safe from New Cracking Software

A new freely available software can crack passwords of up to 55 symbols by guessing a lot of common letter combinations

How We Benefit from Failures





"When a complex system succeeds, that success masks its proximity to failure. . . . Thus, the failure of the *Titanic* contributed much more to the design of safe ocean liners than would have her success. That is the paradox of engineering and design."

Henry Petroski, *Success through Failure: The Paradox of Design*, Princeton U. Press, 2006, p. 95

Take-Home Survey Form: Due Next Class

Personal and contact info: Name, Perm#, e-mail address, phone #(s), degrees & institutions, academic level, GPA, units completed, advisor

Main reason for taking this course

e.g.: interest, advisor's suggestion, have to (not enough grad courses)

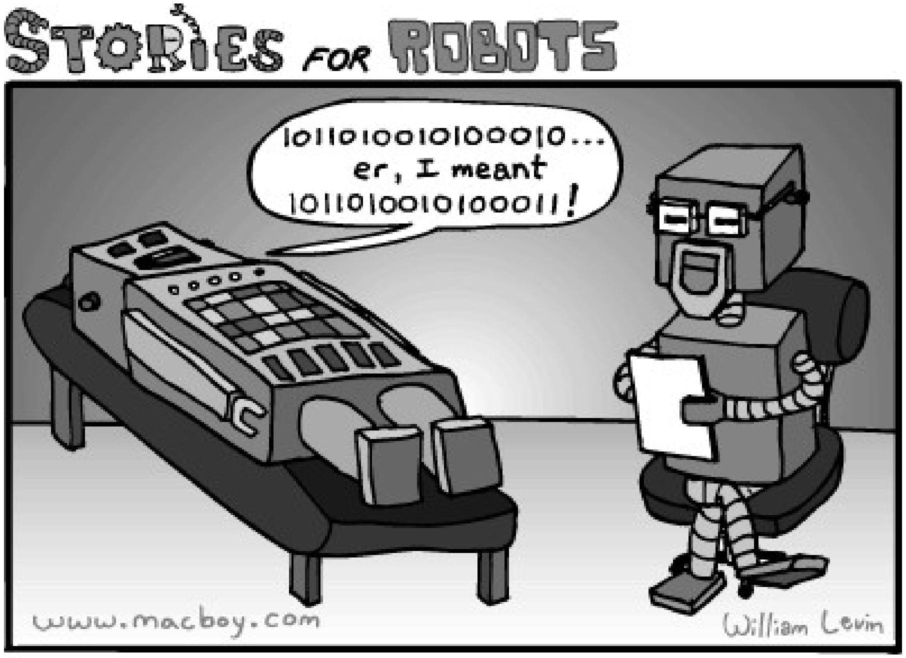
From the lecture topics on the course's website, pick one topic that you believe to be most interesting

List one important fact about yourself that is not evident from your academic record or CV

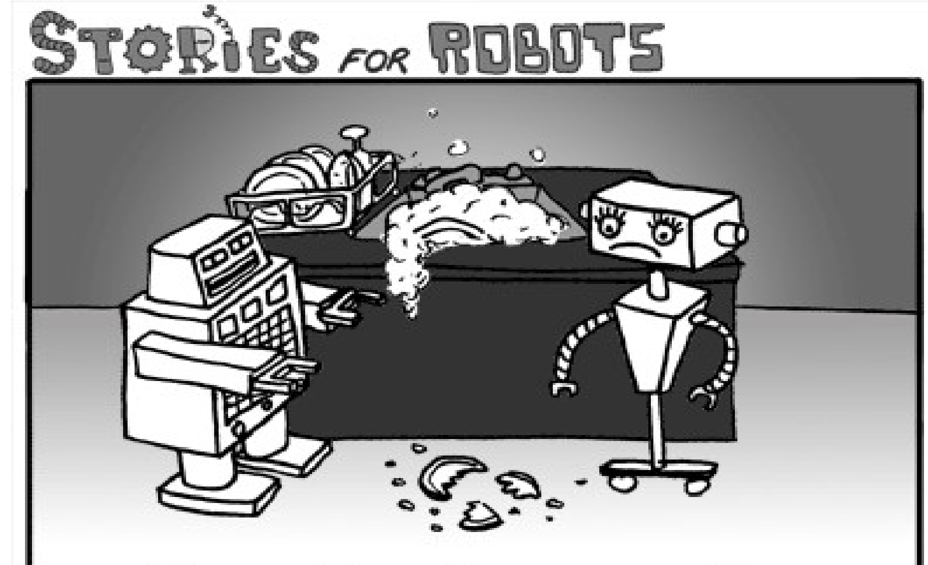
e.g.: I like to solve mathematical, logical, and word puzzles

Use the space below or overleaf for any additional comments on your academic goals and/or expectations from this course

1 Background and Motivation



Droidian Slip



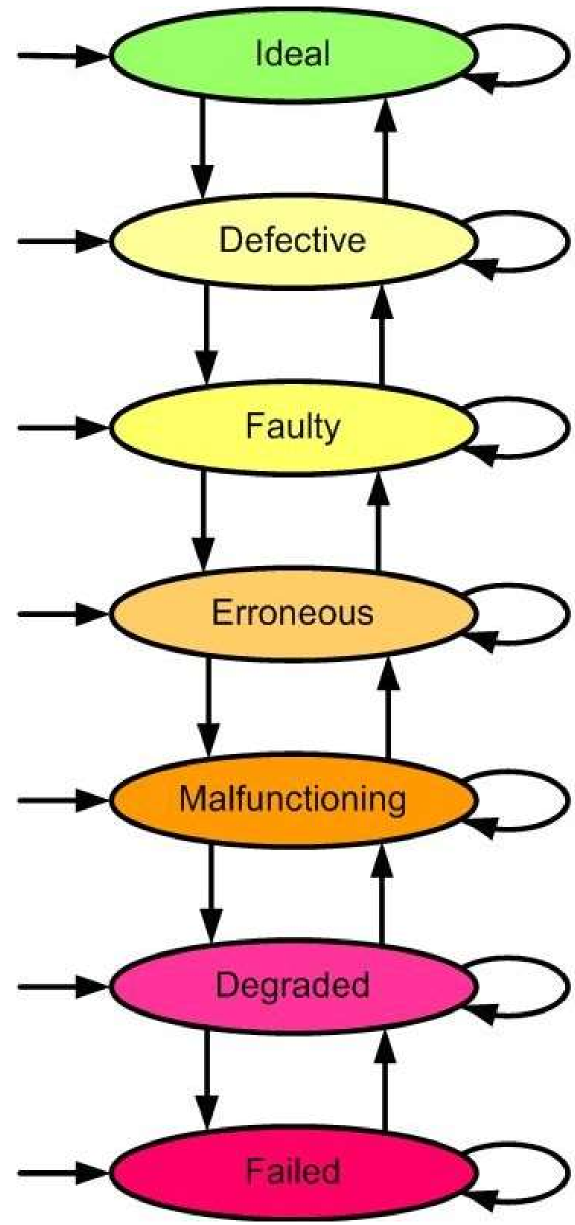
"You can't keep blaming *everything* on human error!"

"I should get this remote control looked at."

STRUCTURE AT A GLANCE

Part I — Introduction: Dependable Systems (The Ideal-System View)	Goals / Models	1. Background and Motivation 2. Dependability Attributes 3. Combinational Modeling 4. State-Space Modeling
Part II — Defects: Physical Imperfections (The Device-Level View)	Methods / Examples	5. Defect Avoidance 6. Defect Circumvention 7. Shielding and Hardening 8. Yield Enhancement
Part III — Faults: Logical Deviations (The Circuit-Level View)	Methods / Examples	9. Fault Testing 10. Fault Masking 11. Design for Testability 12. Replication and Voting
Part IV — Errors: Informational Distortions (The State-Level View)	Methods / Examples	13. Error Detection 14. Error Correction 15. Self-Checking Modules 16. Redundant Disk Arrays
Part V — Malfunctions: Architectural Anomalies (The Structure-Level View)	Methods / Examples	17. Malfunction Diagnosis 18. Malfunction Tolerance 19. Standby Redundancy 20. Resilient Algorithms
Part VI — Degradations: Behavioral Lapses (The Service-Level View)	Methods / Examples	21. Degradation Allowance 22. Degradation Management 23. Robust Task Scheduling 24. Software Redundancy
Part VII — Failures: Computational Breaches (The Result-Level View)	Methods / Examples	25. Failure Confinement 26. Failure Recovery 27. Agreement and Adjudication 28. Fail-Safe System Design

Appendix: Past, Present, and Future



1.1 The Need for Dependability

Hardware problems

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 problems

These can also be classified as design flaws

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

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

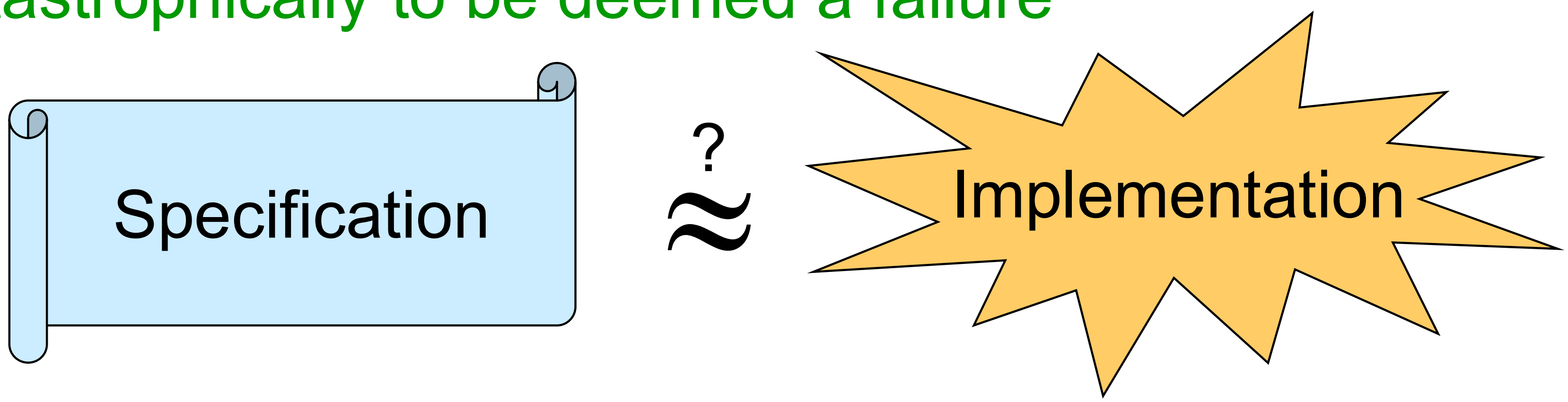
Intel Pentium processor (1993): ≈4M transistors
Intel Pentium 4 processor (2001): ≈40M transistors
Intel Itanium 2 processor (2002): ≈500M transistors

[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]

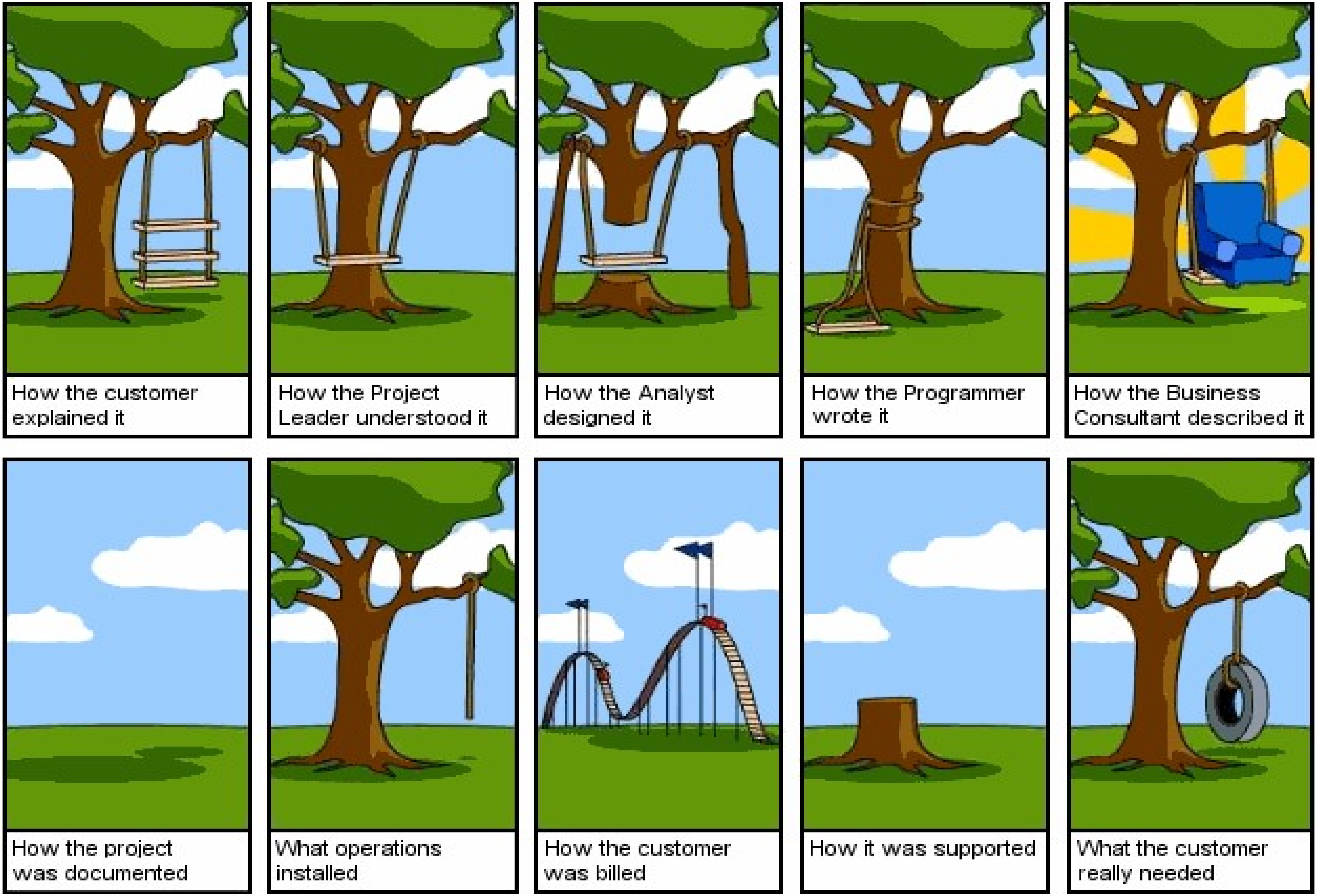
A structure (building or bridge) need not collapse catastrophically to be deemed a failure



Reasons of typical Web site failures

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

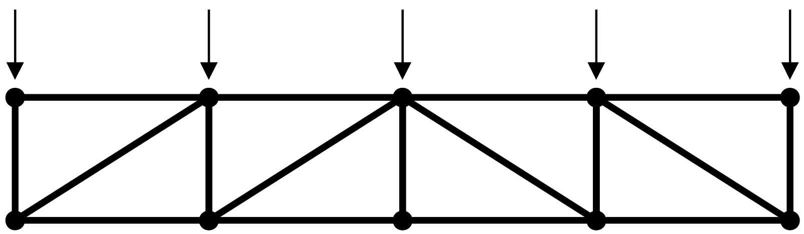
[1] Definition used by the Tech. Council on Forensic Engineering of the Amer. Society of Civil Engineers


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

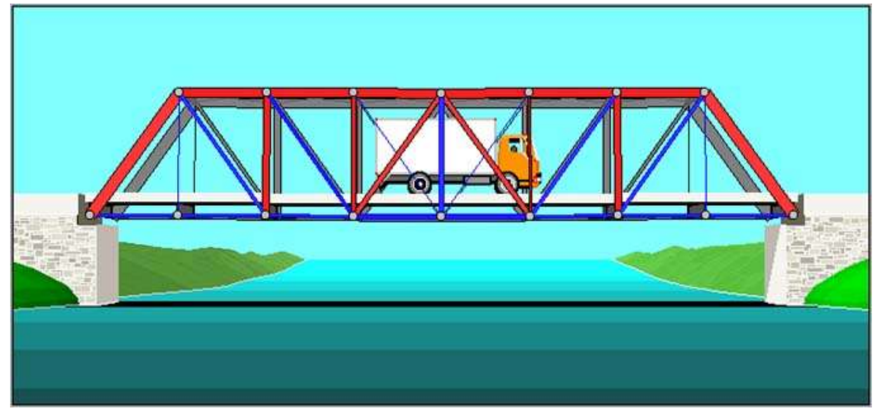
Design Flaws: “To Engineer is Human”[1]

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

Why Dependability Is a Concern

Reliability of *n*-transistor system, each having failure rate λ

$$R(t) = e^{-n\lambda t}$$

There are only 3 ways of making systems more reliable

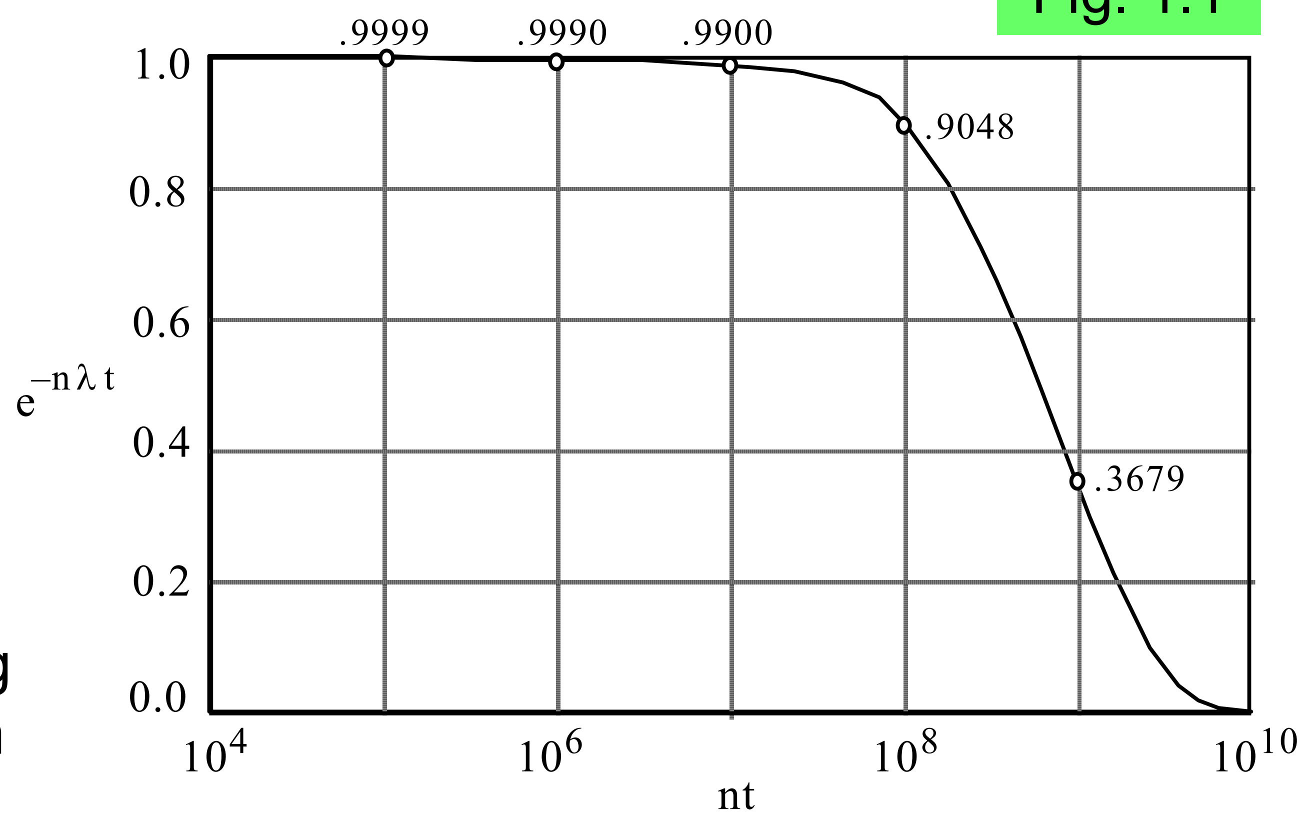
Fig. 1.1

Reduce λ

Reduce *n*

Reduce *t*

Alternative:
Change the reliability formula by introducing redundancy in system

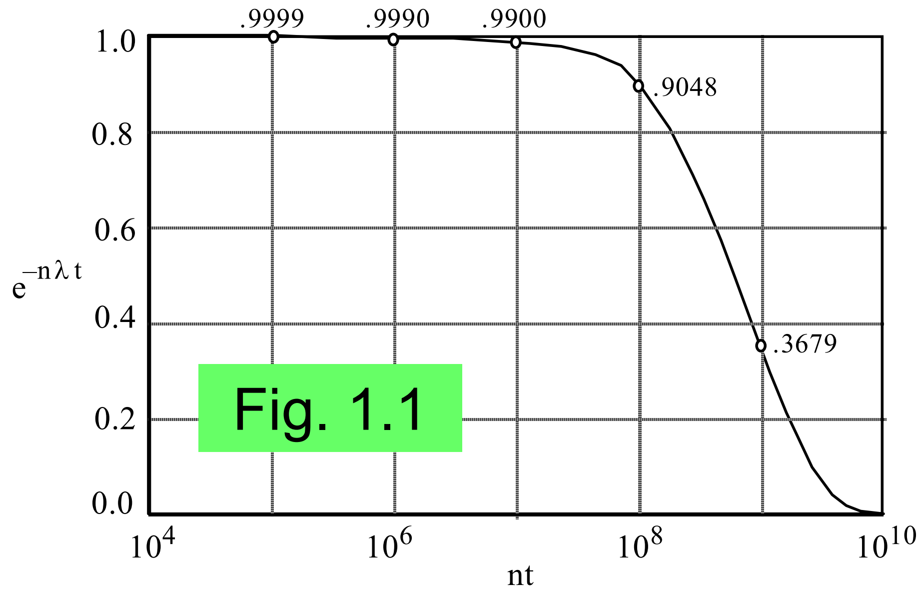


The Three Principal Arguments

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

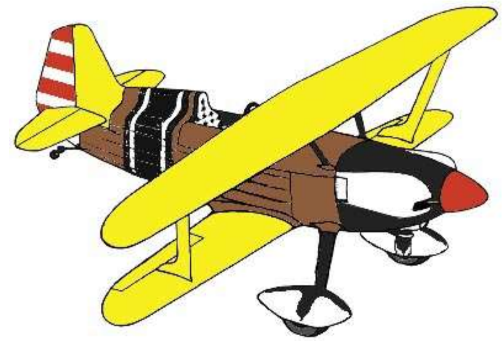
Airline's risk: $O(10^3)$ planes × $O(10^2)$ flights × 10^{-2} computer failures / 10 hr × 0.001 crash / failure × $O(10^2)$ deaths × $O(\$10^7)$ / death = \$ billions / yr

The availability argument

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

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

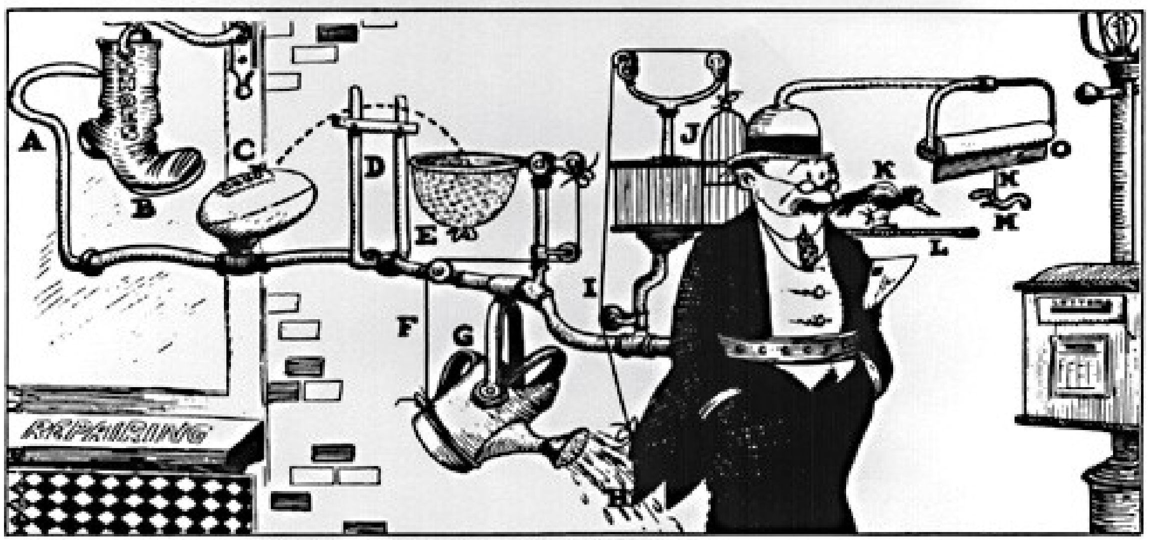
Mishap: misfortune; unfortunate accident

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

1.2 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 = $1 - SL$
= $1 - 0.99 \times 0.95 = 5.95\%$

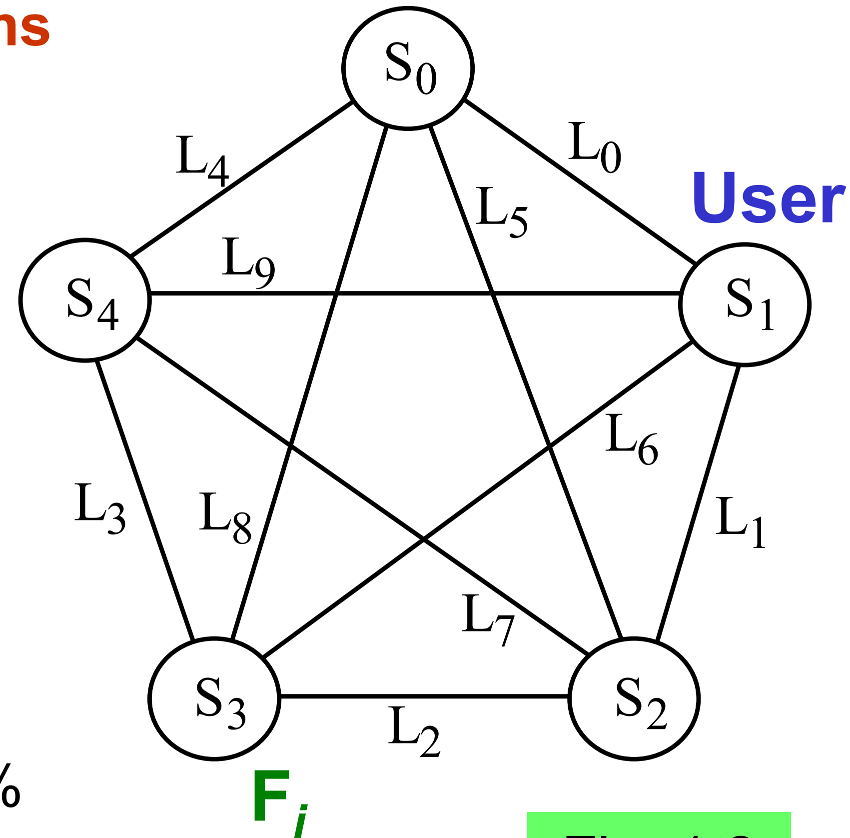


Fig. 1.2

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%

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

Primary site can be reached

Primary site inaccessible

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%

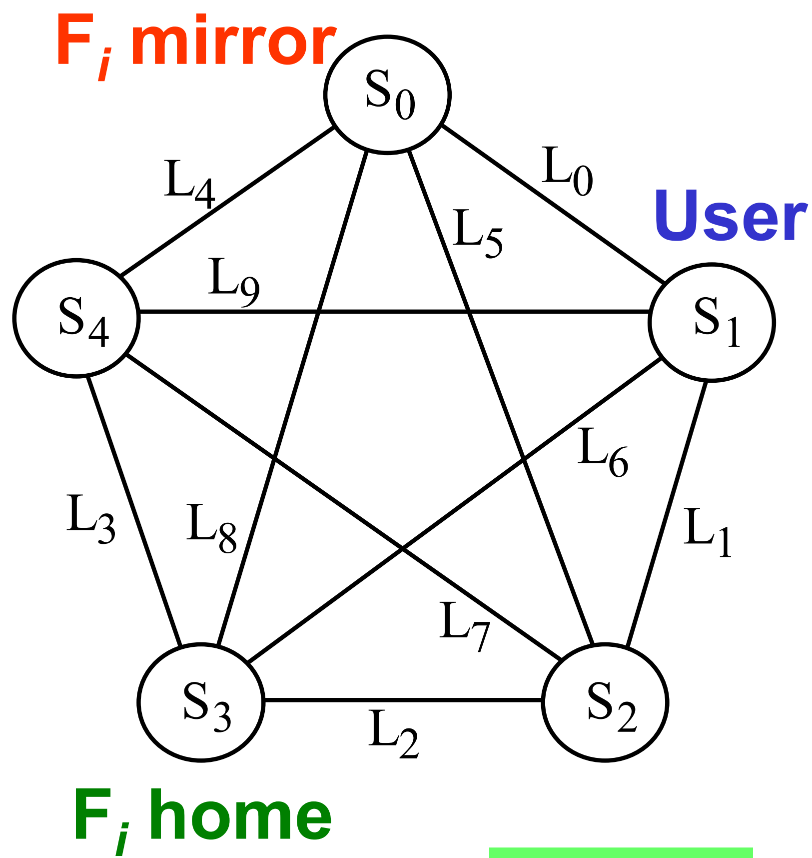


Fig. 1.2

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

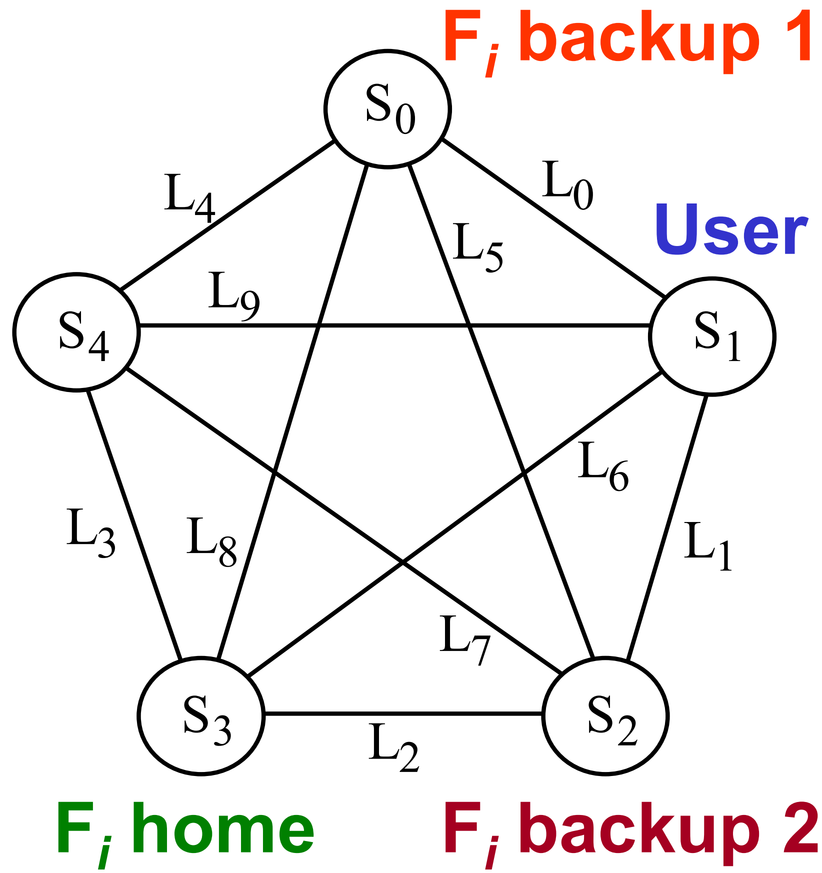


Fig. 1.2

Triplicated avail. $= 3SL - 3(SL)^2 - (SL)^3$

Unavailability $= 1 - 3SL - 3(SL)^2 + (SL)^3$

$= (1 - SL)^3 = 0.02\%$

Data unavailability reduced from 5.95% to 0.02%

Availability improved from ≈ 94% to 99.98%

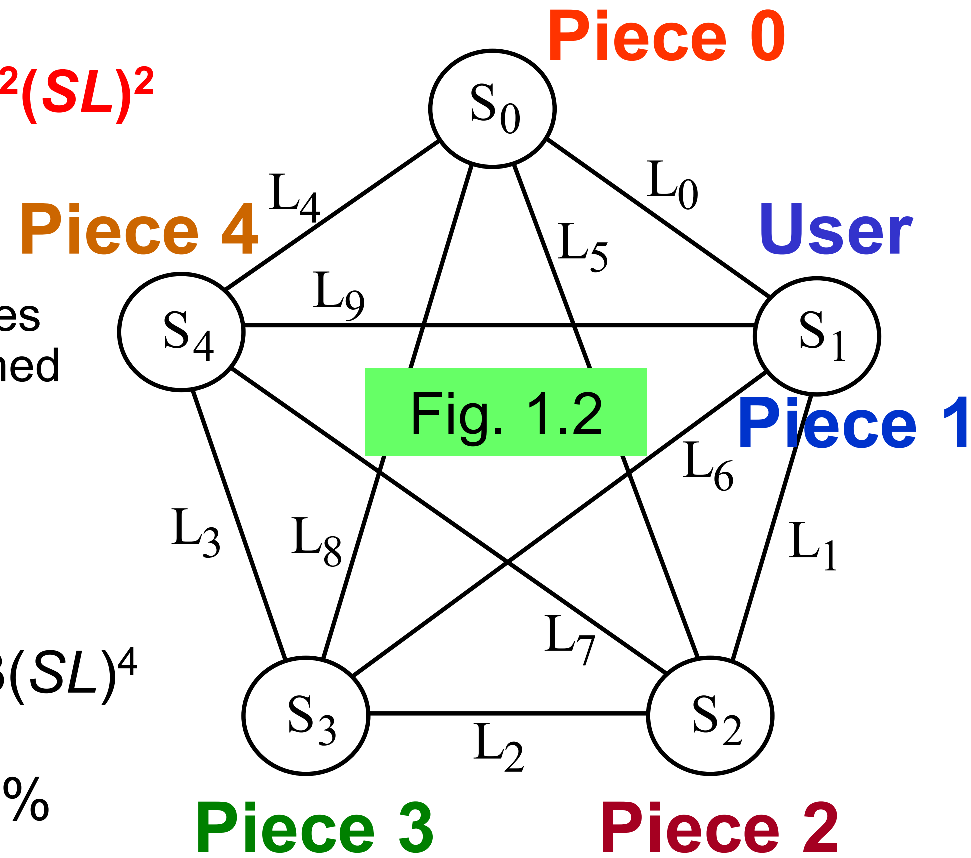
Data Dispersion: Three of Five Pieces

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

All 4 pieces can be reached

Exactly 3 pieces can be reached

Only 2 pieces can be reached

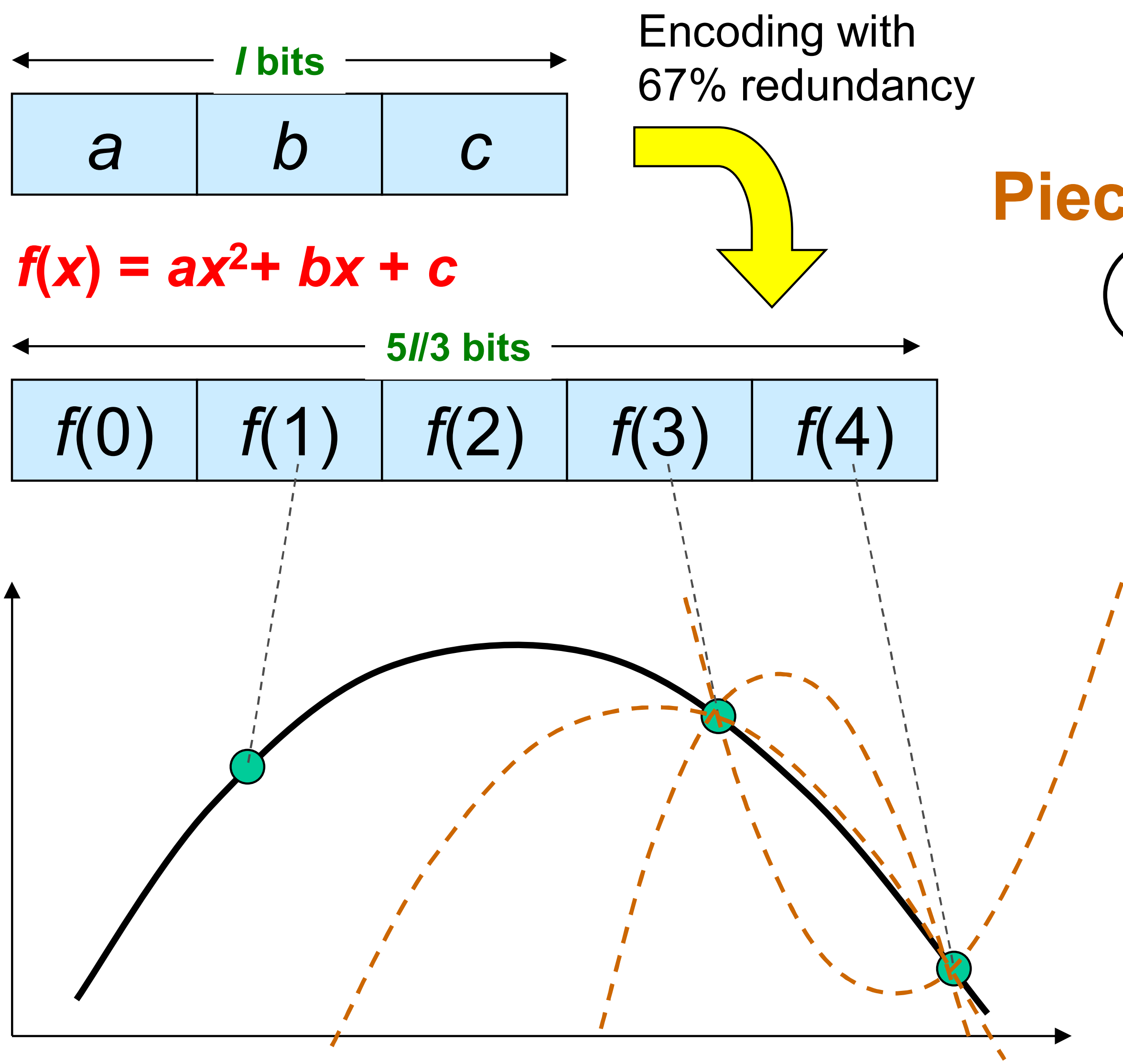


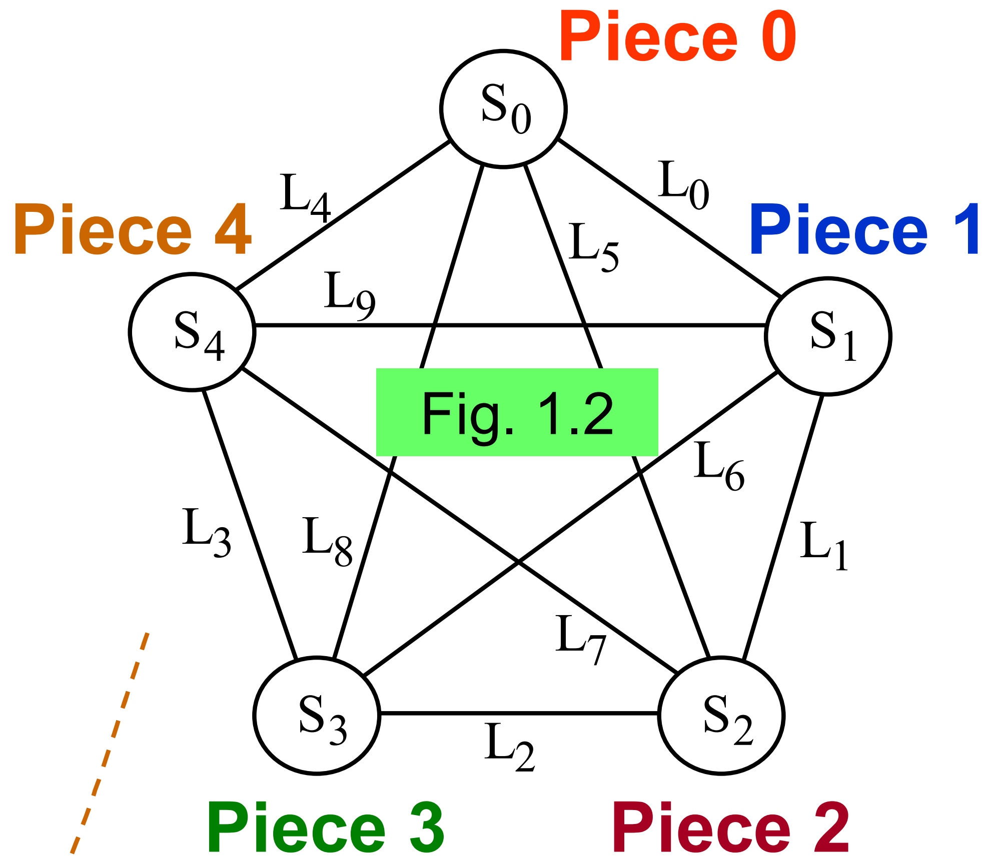
S: Site availability e.g., 99%
L: Link availability e.g., 95%

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

1.3 Impairments to Dependability

Flaw

Fault

Hazard

Bug

ERROR

Degradation

Defect

Failure

Intrusion

Crash

Malfunction

The Fault-Error-Failure Cycle

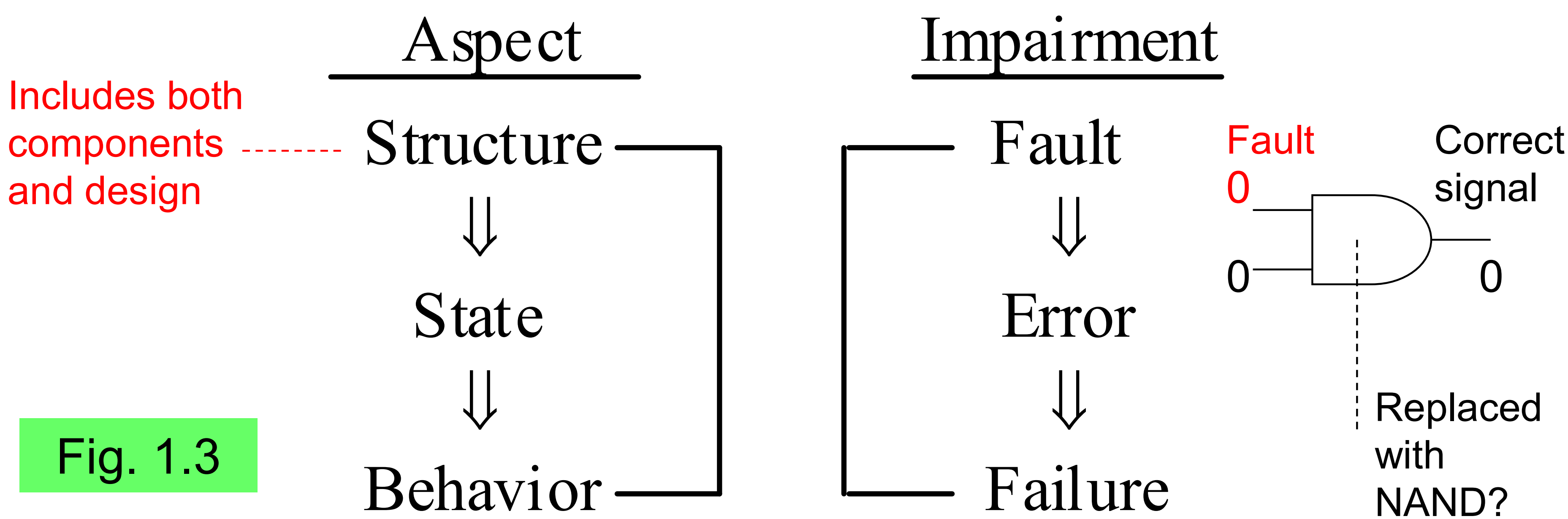


Fig. 1.3

Schematic diagram of the Newcastle hierarchical model and the impairments within one level.

The Four-Universe Model

Universe	Impairment
Physical	Failure
$\Downarrow$	$\Downarrow$
Logical	Fault
$\Downarrow$	$\Downarrow$
Informational	Error
$\Downarrow$	$\Downarrow$
External	Crash

Fig. 1.4

Cause-effect diagram for Avižienis’ four-universe model of impairments to dependability.

Unrolling the Fault-Error-Failure Cycle

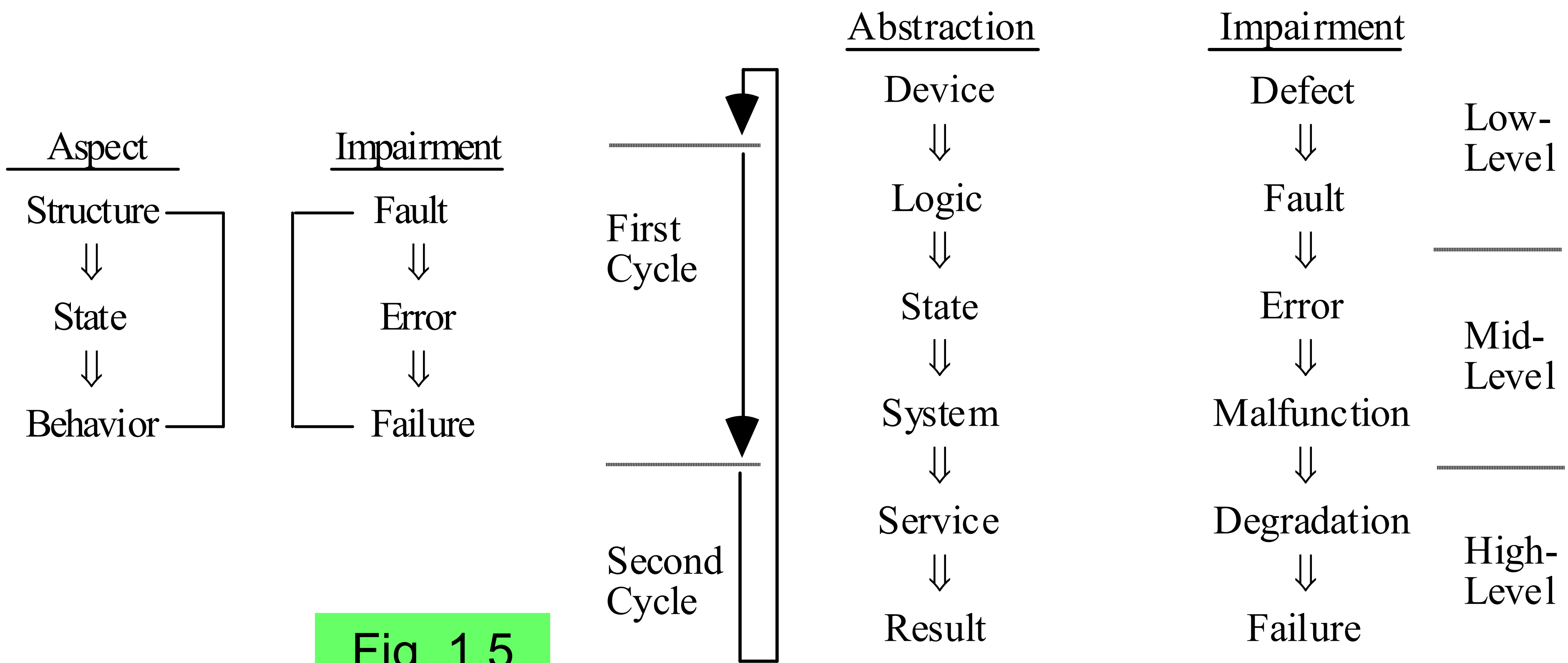


Fig. 1.5

Cause-effect diagram for an extended six-level view of impairments to dependability.

1.4 A Multilevel Model

Device

Logic

State

System

Service

Result

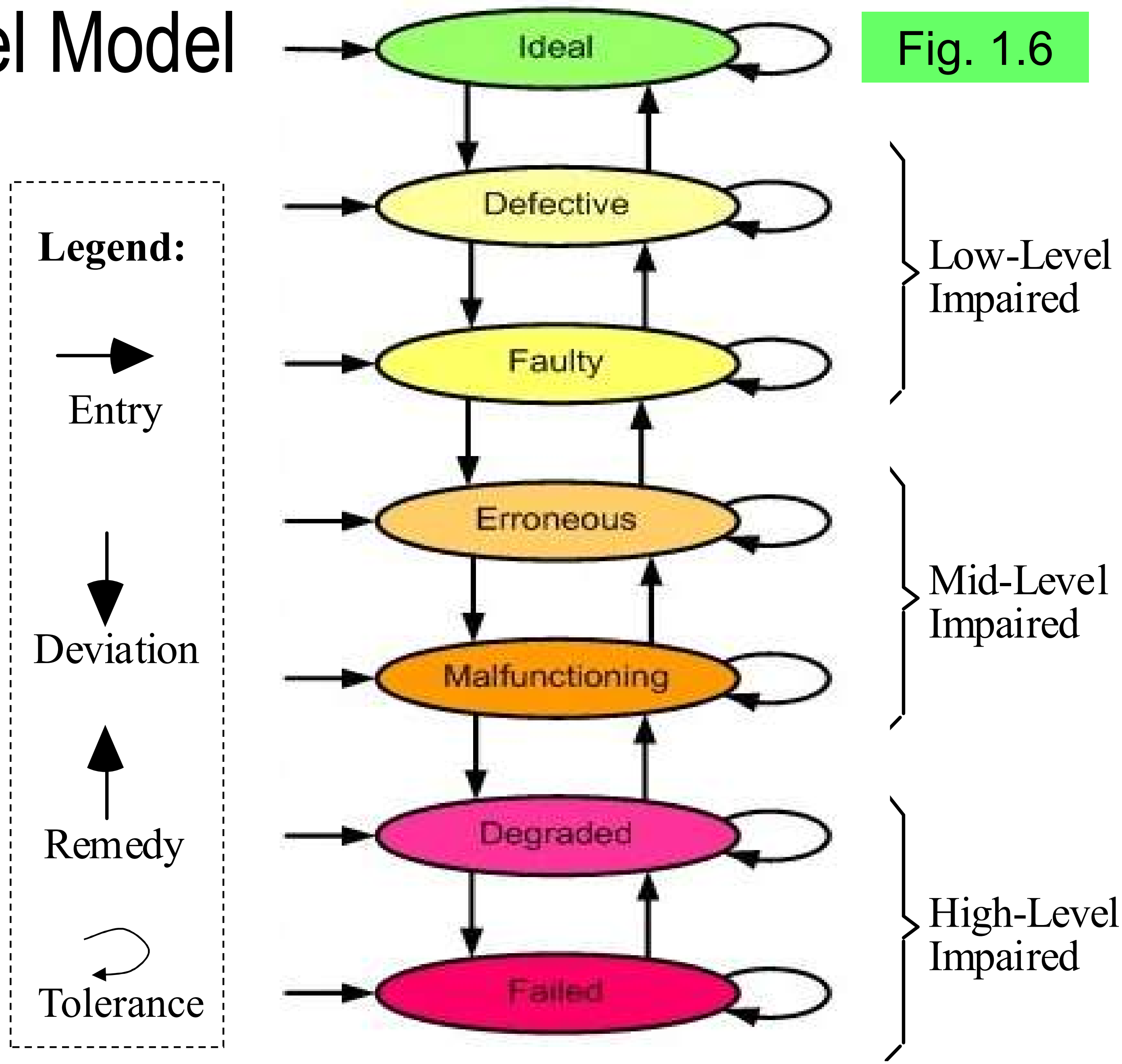


Fig. 1.6

1.5 Examples and Analogies

Example 1.4: **Automobile brake system**

Defect	Brake fluid piping has a weak spot or joint
Fault	Brake fluid starts to leak out
Error	Brake fluid pressure drops too low
Malfunction	Braking force is below expectation
Degradation	Braking requires higher force or takes longer
Failure	Vehicle does not slow down or stop in time

Note in particular that not every defect, fault, error, malfunction, or degradation leads to failure

Analogy for the Multilevel Model

An analogy for our multi-level model of dependable computing.
Defects, faults, errors, malfunctions, degradations, and failures are represented by pouring water from above. Valves represent avoidance and tolerance techniques. The goal is to avoid overflow.

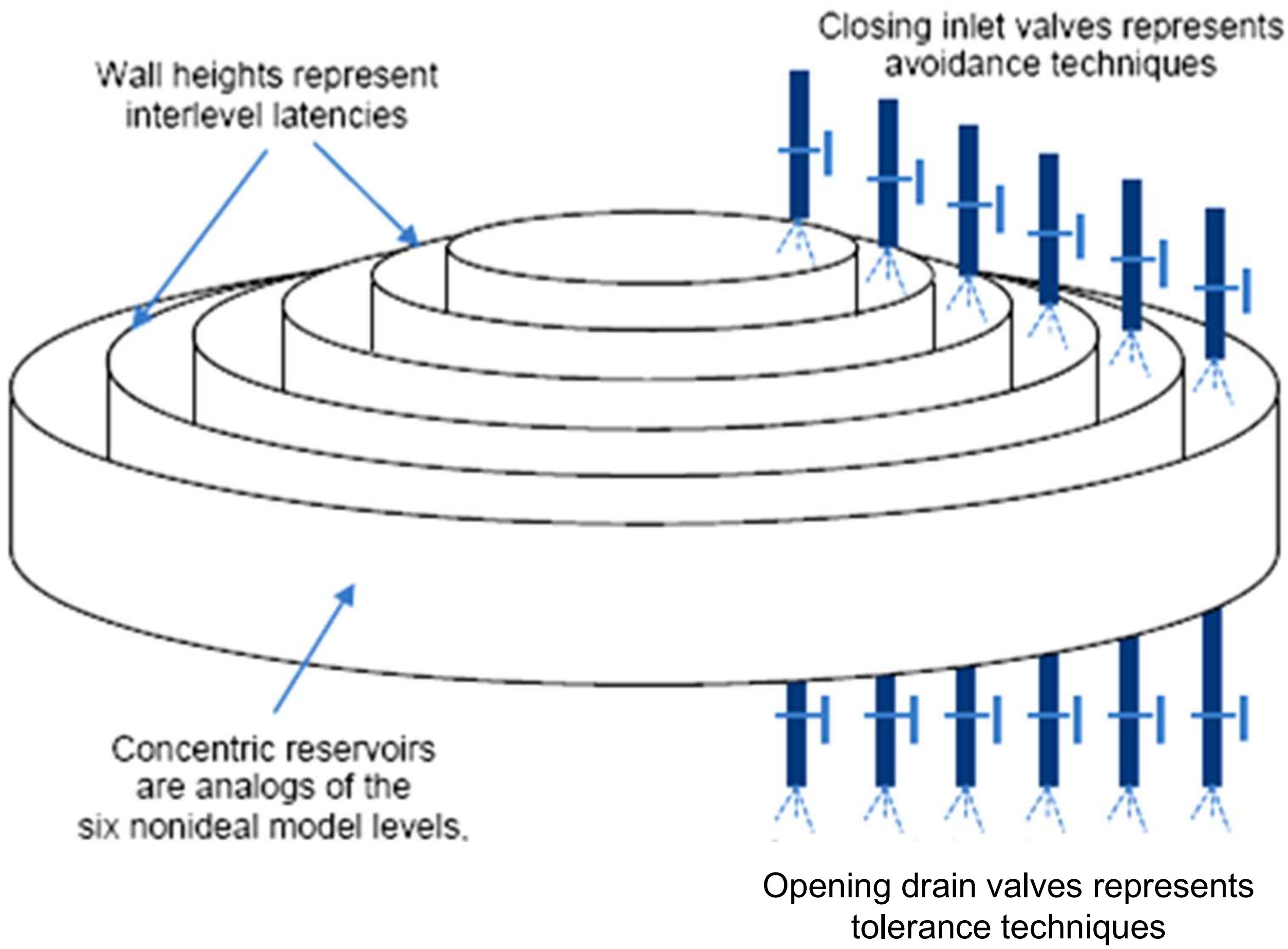


Fig. 1.7

1.6 Dependable Computer Systems

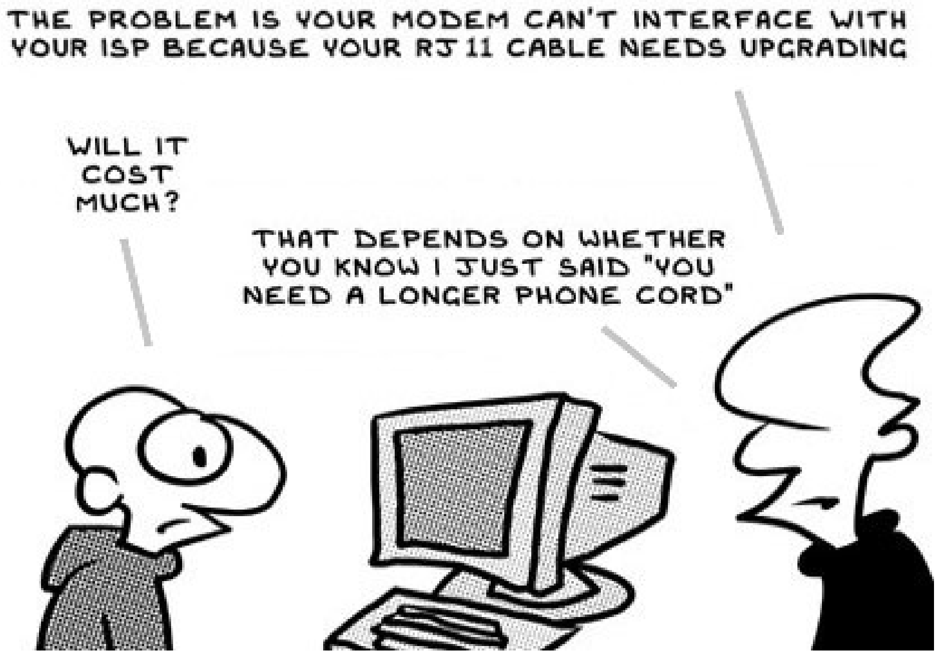
Long-life systems: Fail-slow, Rugged, High-reliability
Spacecraft with multiyear missions, systems in inaccessible locations
Methods: Replication (spares), error coding, monitoring, shielding

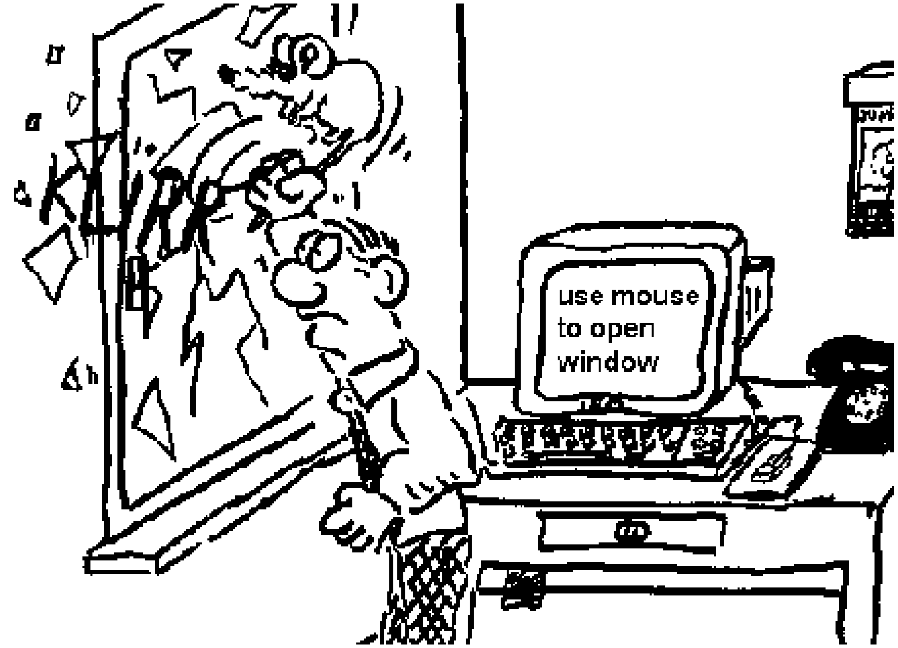
Safety-critical systems: Fail-safe, Sound, High-integrity
Flight control computers, nuclear-plant shutdown, medical monitoring
Methods: Replication with voting, time redundancy, design diversity

Non-stop systems: Fail-soft, Robust, High-availability
Telephone switching centers, transaction processing, e-commerce
Methods: HW/info redundancy, backup schemes, hot-swap, recovery

Just as performance enhancement techniques gradually migrate from supercomputers to desktops, so too dependability enhancement methods find their way from exotic systems into personal computers

2 Dependability Attributes





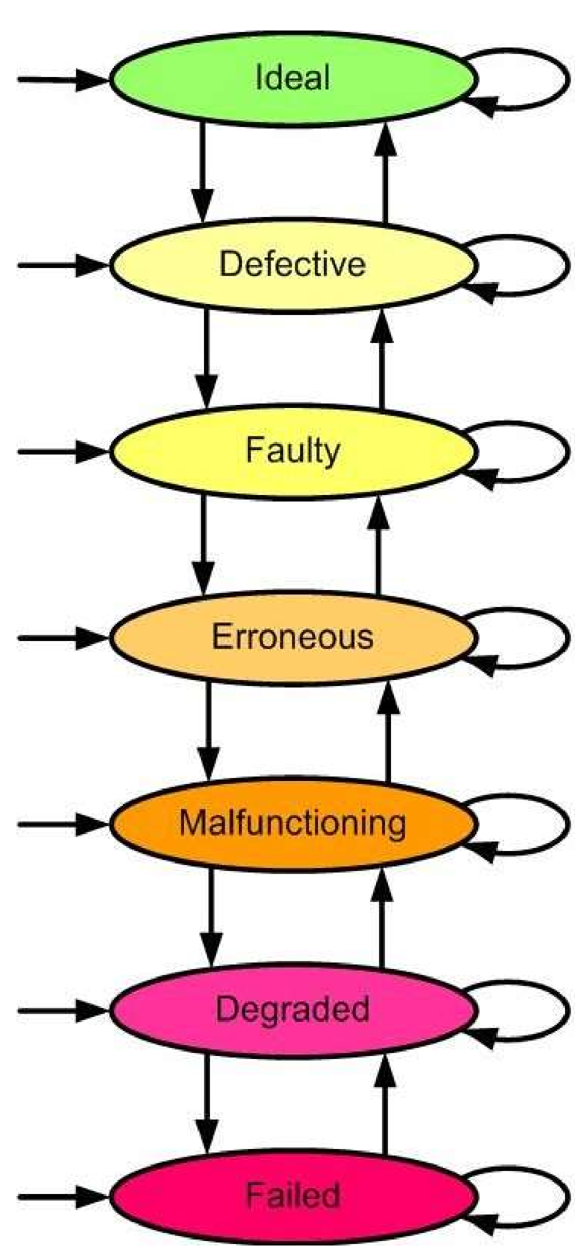
"Every time we successfully recover from a technical problem, the computer likes a high five."

"That foul smell is coming from your computer. You've got some old data in there that's gone bad."

STRUCTURE AT A GLANCE

Part I — Introduction: Dependable Systems (The Ideal-System View)	Goals	1. Background and Motivation 2. Dependability Attributes
	Models	3. Combinational Modeling 4. State-Space Modeling
Part II — Defects: Physical Imperfections (The Device-Level View)	Methods	5. Defect Avoidance 6. Defect Circumvention
	Examples	7. Shielding and Hardening 8. Yield Enhancement
Part III — Faults: Logical Deviations (The Circuit-Level View)	Methods	9. Fault Testing 10. Fault Masking
	Examples	11. Design for Testability 12. Replication and Voting
Part IV — Errors: Informational Distortions (The State-Level View)	Methods	13. Error Detection 14. Error Correction
	Examples	15. Self-Checking Modules 16. Redundant Disk Arrays
Part V — Malfunctions: Architectural Anomalies (The Structure-Level View)	Methods	17. Malfunction Diagnosis 18. Malfunction Tolerance
	Examples	19. Standby Redundancy 20. Resilient Algorithms
Part VI — Degradations: Behavioral Lapses (The Service-Level View)	Methods	21. Degradation Allowance 22. Degradation Management
	Examples	23. Robust Task Scheduling 24. Software Redundancy
Part VII — Failures: Computational Breaches (The Result-Level View)	Methods	25. Failure Confinement 26. Failure Recovery
	Examples	27. Agreement and Adjudication 28. Fail-Safe System Design

Appendix: Past, Present, and Future



2.1 Aspects of Dependability

The -ilities

Serviceability

Security

Safety
Risk, consequence

Resilience

RELIABILITY
Reliability, MTTF = MTFF

Availability
Pointwise av., Interval av.,
MTBF, MTTR

Testability
Controllability,
observability

Performability
Performability, MCBF

Maintainability

Integrity

Robustness

Concepts from Probability Theory

Probability density function: pdf

$f(t) = \text{prob}[t \le x \le t + dt] \,/\, dt = dF(t) \,/\, dt$

Cumulative distribution function: CDF

$F(t) = \text{prob}[x \le t] = \int_0^t f(x)\,dx$

Expected value of *x*

$E_x = \int_{-\infty}^{+\infty} x\,f(x)\,dx = \sum_k x_k f(x_k)$

Variance of *x*

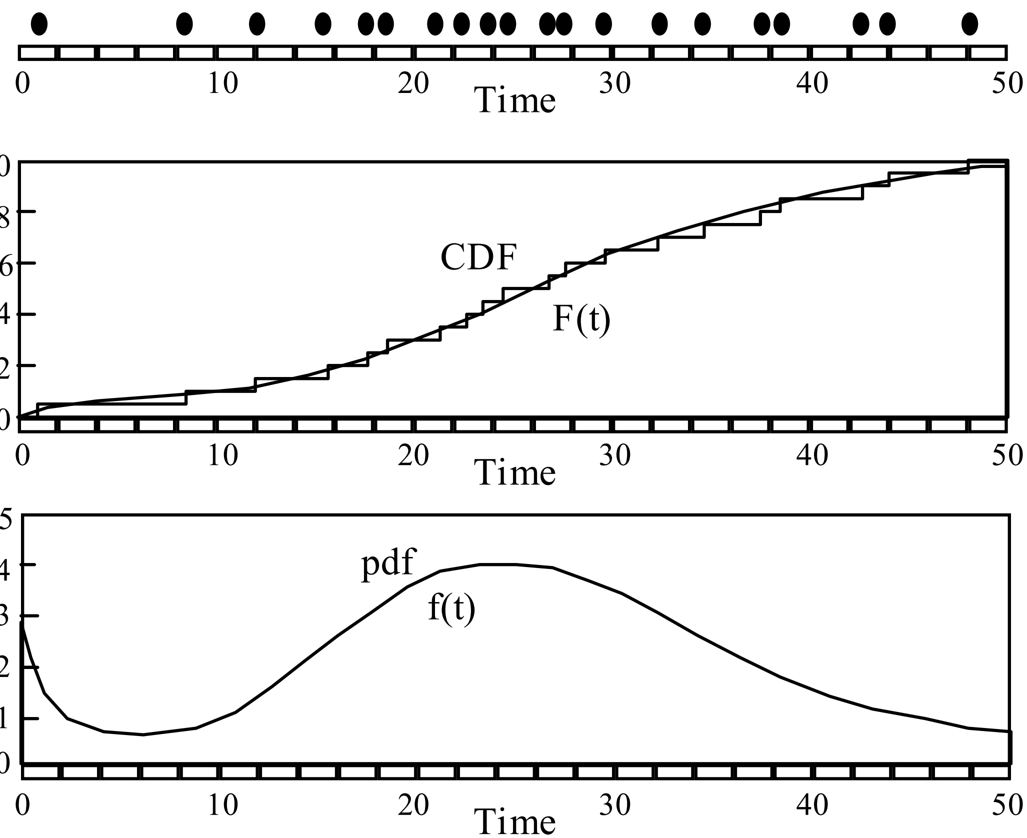
$$\sigma_x^2 = \int_{-\infty}^{+\infty} (x - E_x)^2 f(x)\,dx = \sum_k (x_k - E_x)^2 f(x_k)$$

Covariance of *x* and *y*

$$\psi_{x,y} = E[(x - E_x)(y - E_y)] = E[xy] - E_x E_y$$

Fig. 2.1

Lifetimes of 20 identical systems



Some Simple Probability Distributions

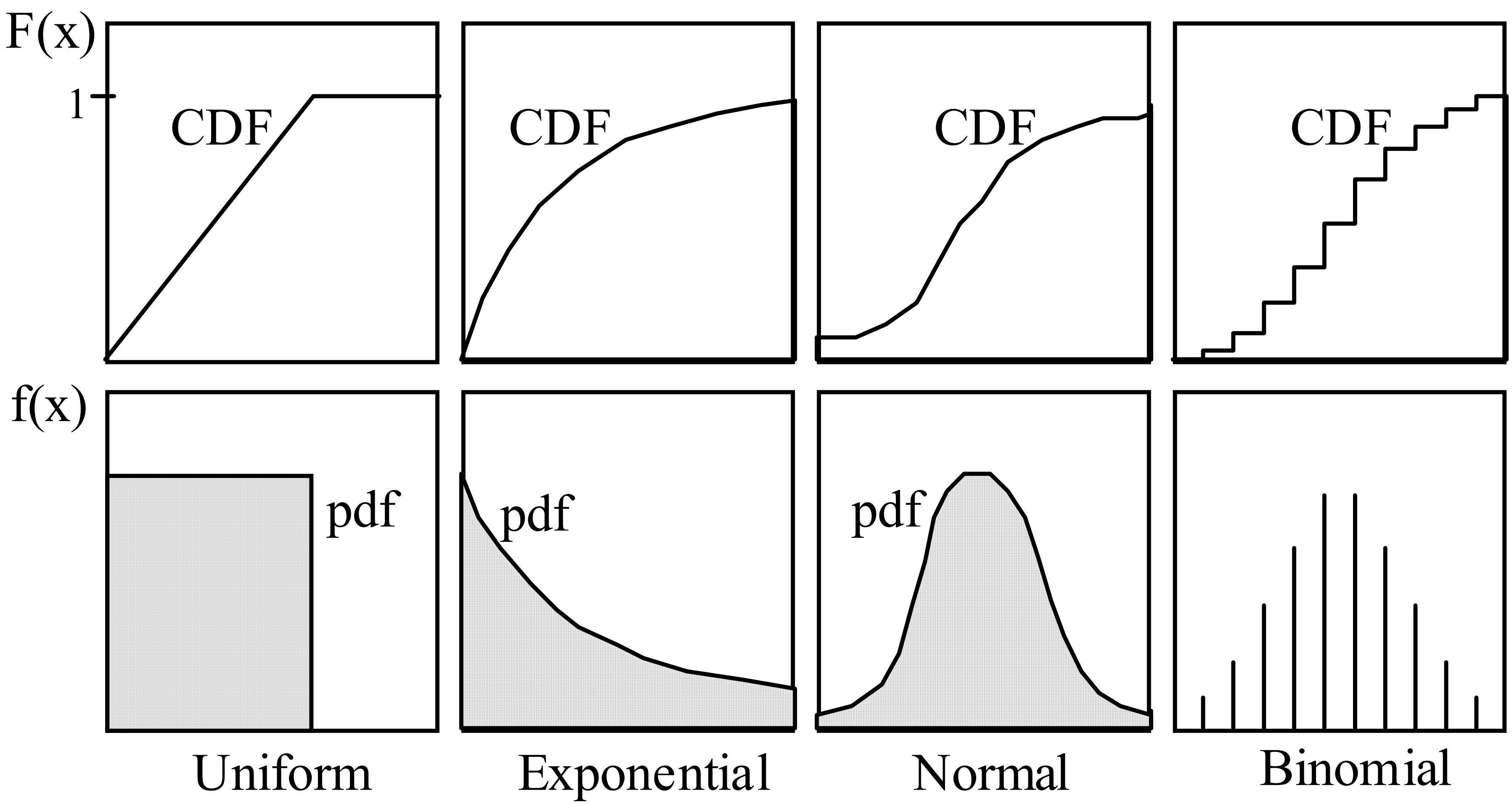
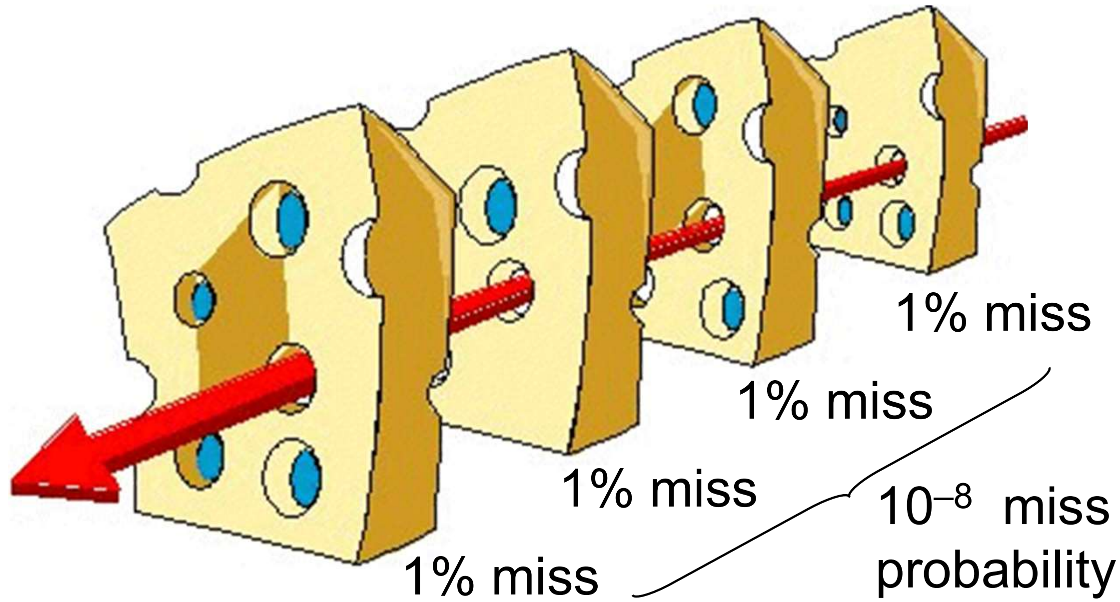


Fig. 2.2

Layers of Safeguards



With multiple layers of safeguards, a system failure occurs only if warning symptoms and compensating actions are missed at every layer, which is quite unlikely

Is it really?

The computer engineering literature is full of examples of mishaps when two or more layers of protection failed at the same time

Multiple layers increase the reliability significantly only if the “holes” in the representation above are fairly randomly and independently distributed, so that the probability of their being aligned is negligible

Dec. 1986: ARPANET had 7 dedicated lines between NY and Boston; A backhoe accidentally cut all 7 (they went through the same conduit)

2.2 Reliability and MTTF

Reliability: $R(t)$

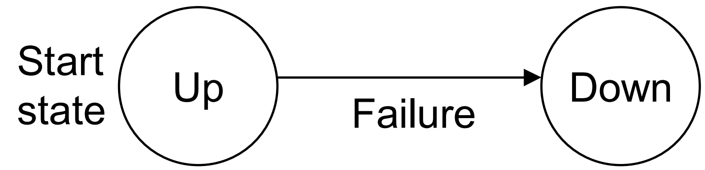
Probability that system remains in the "Good" state through the interval [0, *t*]

$R(t + dt) = R(t)\,[1 - z(t)\,dt]$

Hazard function

Fig. 2.3

Two-state nonrepairable system



$R(t) = 1 - F(t)$ CDF of the system lifetime, or its unreliability

Constant hazard function $z(t) = \lambda \Rightarrow R(t) = e^{-\lambda t}$
(system failure rate is independent of its age)

Exponential reliability law

Mean time to failure: MTTF

$$\mathrm{MTTF} = \underbrace{\int_0^{+\infty} t\,f(t)\,dt}_{\text{Expected value of lifetime}} = \underbrace{\int_0^{+\infty} R(t)\,dt}_{\text{Area under the reliability curve (easily provable)}}$$

Failure Distributions of Interest

Discrete versions

Exponential: $z(t) = \lambda$

$R(t) = e^{-\lambda t}$ MTTF = $1/\lambda$

Geometric

$R(k) = q^k$

Rayleigh: $z(t) = 2\lambda(\lambda t)$

$R(t) = e^{(-\lambda t)^2}$ MTTF = $(1/\lambda)\sqrt{\pi} / 2$

Weibull: $z(t) = \alpha\lambda(\lambda t)^{\alpha-1}$

$R(t) = e^{(-\lambda t)^\alpha}$ MTTF = $(1/\lambda)\,\Gamma(1 + 1/\alpha)$

Discrete Weibull

Erlang:

Gen. exponential MTTF = k/λ

Gamma:

Gen. Erlang
(becomes Erlang for b an integer)

Normal:

Reliability and MTTF formulas are complicated

Binomial

Elaboration on Weibull Distribution

Weibull: $z(t) = \alpha\lambda(\lambda t)^{\alpha-1}$

$R(t) = e^{(-\lambda t)^\alpha}$

$\ln \ln[1/R(t)] = \alpha(\ln t + \ln \lambda)$

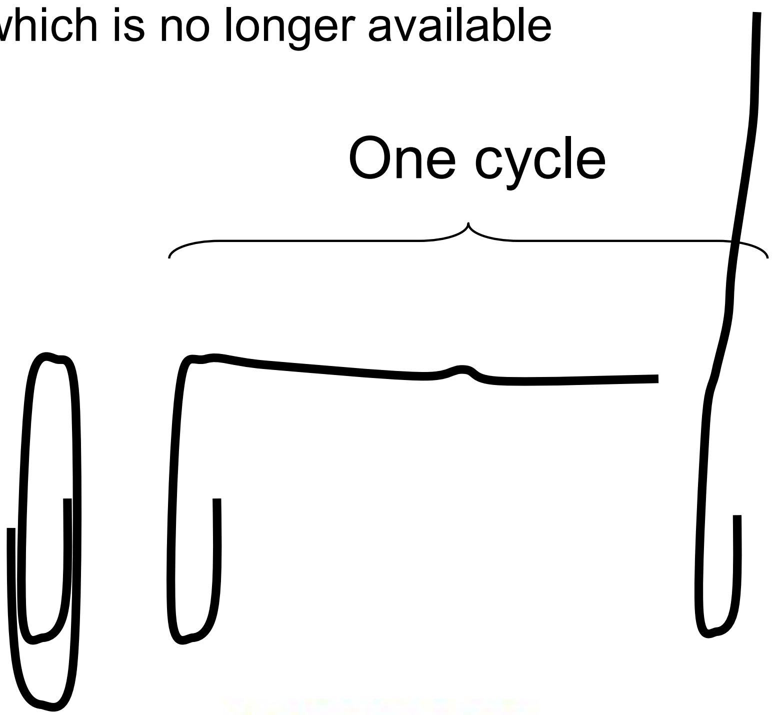
$\alpha < 1$, Infant mortality

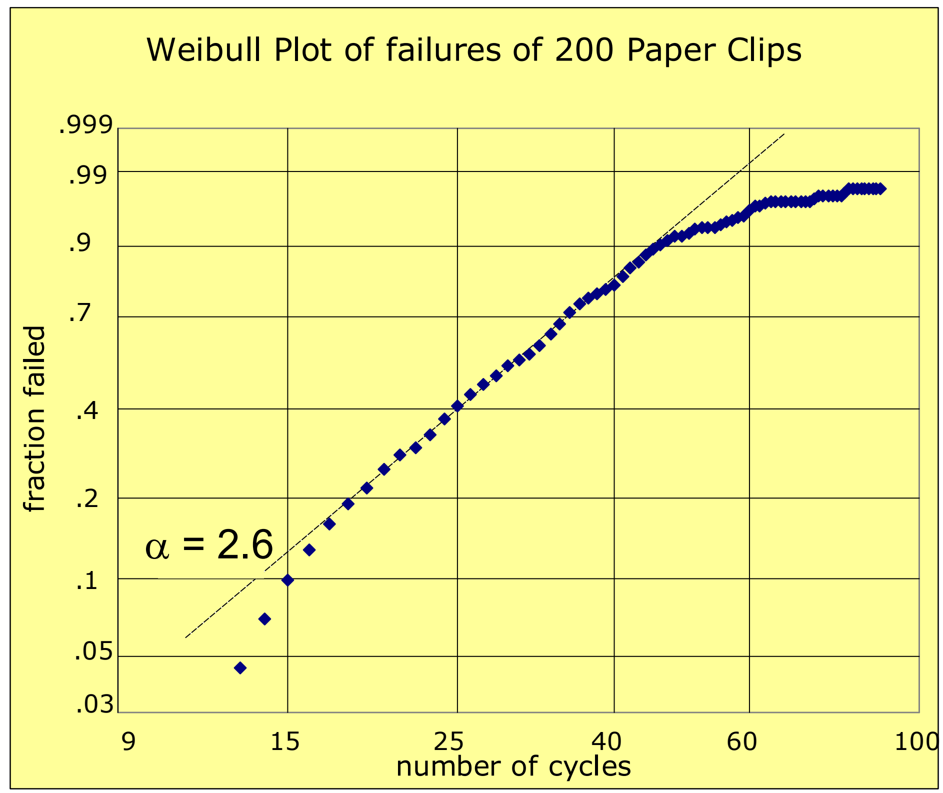
$\alpha = 1$, Constant hazard rate (exponential)

$1 < \alpha < 4$, Rising hazard (fatigue, corrosion)

$\alpha > 4$, Rising hazard (rapid wearout)

The following diagrams were taken from
http://www.rpi.edu/~albenr/presentations/Reliabilty.ppt
which is no longer available





Comparing Reliabilities

Reliability difference: $R_2 - R_1$

Reliability gain: R_2 / R_1

Reliability improvement factor

$\mathrm{RIF}_{2/1} = [1 - R_1(t_M)] / [1 - R_2(t_M)]$

Example:
$[1 - 0.9] / [1 - 0.99] = 10$

Reliability improv. index

$\mathrm{RII} = \log R_1(t_M) / \log R_2(t_M)$

Mission time extension

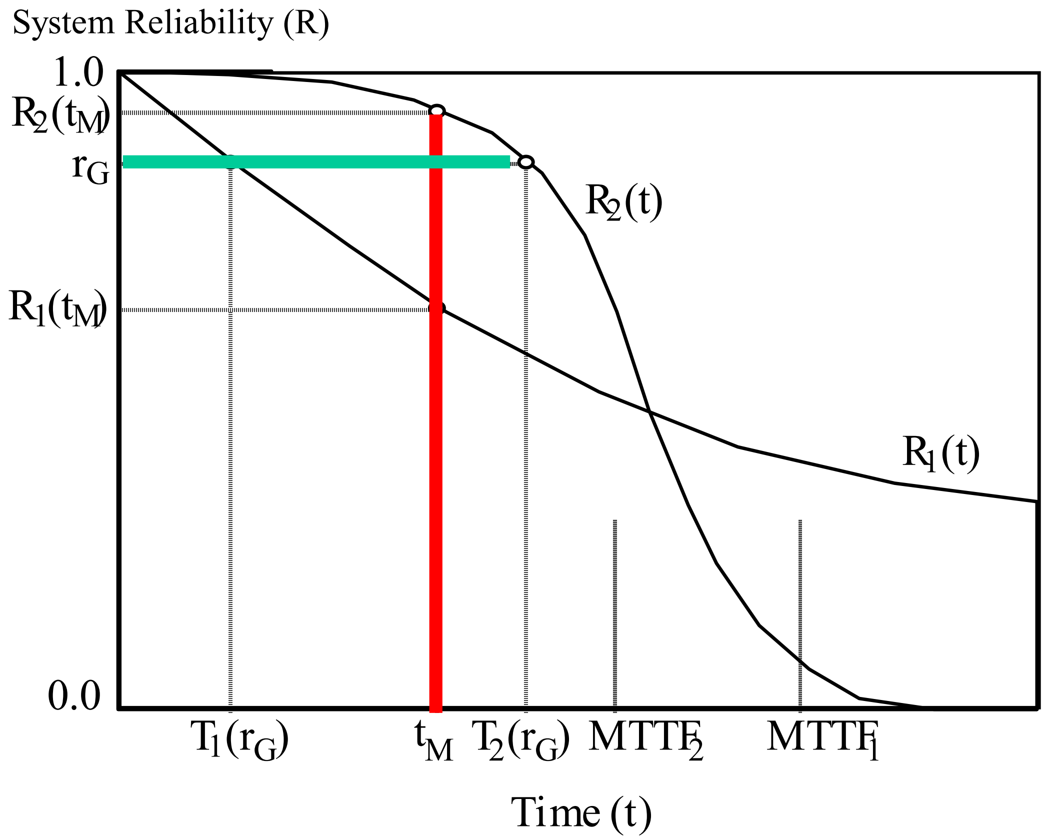
$\mathrm{MTE}_{2/1}(r_G) = T_2(r_G) - T_1(r_G)$

Mission time improv. factor:

$\mathrm{MTIF}_{2/1}(r_G) = T_2(r_G) / T_1(r_G)$

Fig. 2.4

Reliability functions for Systems 1 and 2



Analog of Amdahl's Law for Reliability

Amdahl's law: If in a unit-time computation, a fraction f doesn't change and the remaining fraction $1 - f$ is speeded up to run p times as fast, the overall speedup will be $s = 1 / (f + (1 - f)/p)$

Consider a system with two parts, having failure rates ϕ and $\lambda - \phi$

Improve the failure rate of the second part by a factor p, to $(\lambda - \phi)/p$

$R_{\text{original}} = \exp(-\lambda t)$

$R_{\text{improved}} = \exp[-(\phi + (\lambda - \phi)/p)t]$

Reliability improv. index

$\text{RII} = \log R_{\text{original}} / \log R_{\text{improved}}$

$\text{RII} = \lambda / (\phi + (\lambda - \phi)/p)$

See B. Parhami's paper in July 2015 *IEEE Computer*

Letting $\phi / \lambda = f$, we have: $\text{RII} = 1 / (f + (1 - f)/p)$

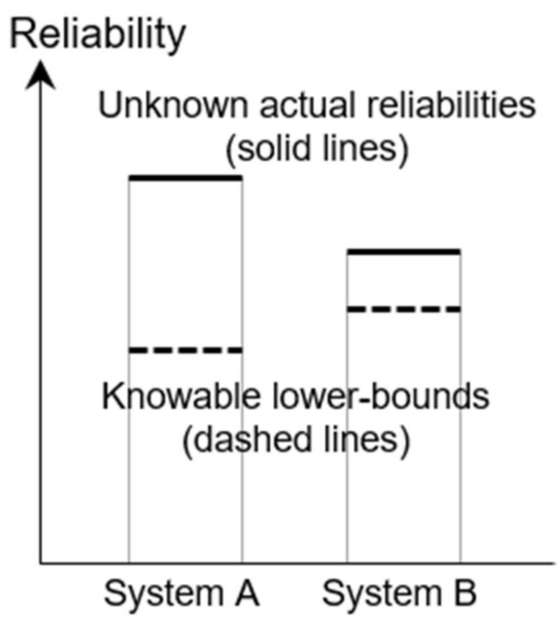
Reliability Inversion

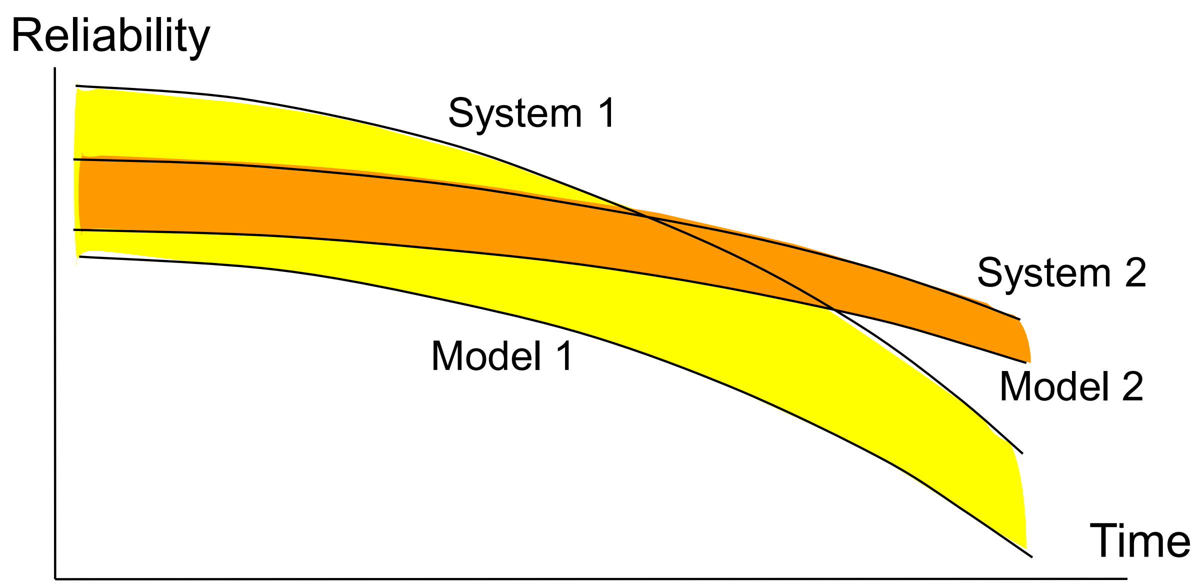
Actual reliability is unknowable

We derive a pessimistic lower bound, which can be tight or loose

The more pessimistic the assumptions, the looser the bounds

But pessimism is dictated by our concern for safety





2.3 Availability, MTTR, and MTBF

(Interval) Availability: *A*(*t*)

Fraction of time that system is in the "Up" state during the interval [0, *t*]

Steady-state availability: $A = \lim_{t\to\infty} A(t)$

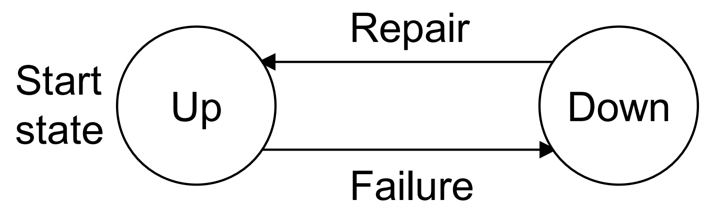
Pointwise availability: *a*(*t*)

Probability that system available at time *t*

$A(t) = (1/t) \int_0^t a(x)\, dx$

Fig. 2.5

Two-state repairable system



Availability = Reliability, when there is no repair

Availability is a function not only of how rarely a system fails (reliability) but also of how quickly it can be repaired (time to repair)

$$A = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} = \frac{\text{MTTF}}{\text{MTBF}} = \frac{\mu}{\lambda + \mu}$$

Repair rate
$1/\mu$ = MTTR
(Will justify this equation later)

In general, $\mu >> \lambda$, leading to $A \cong 1$

System Up and Down Times

Short repair time implies
good **maintainability (serviceability)**

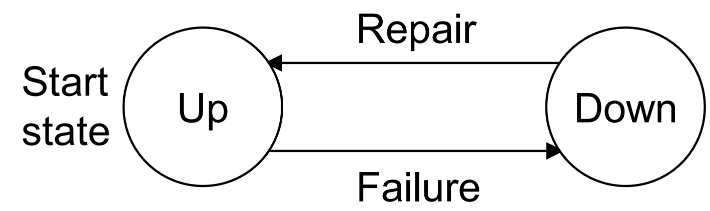
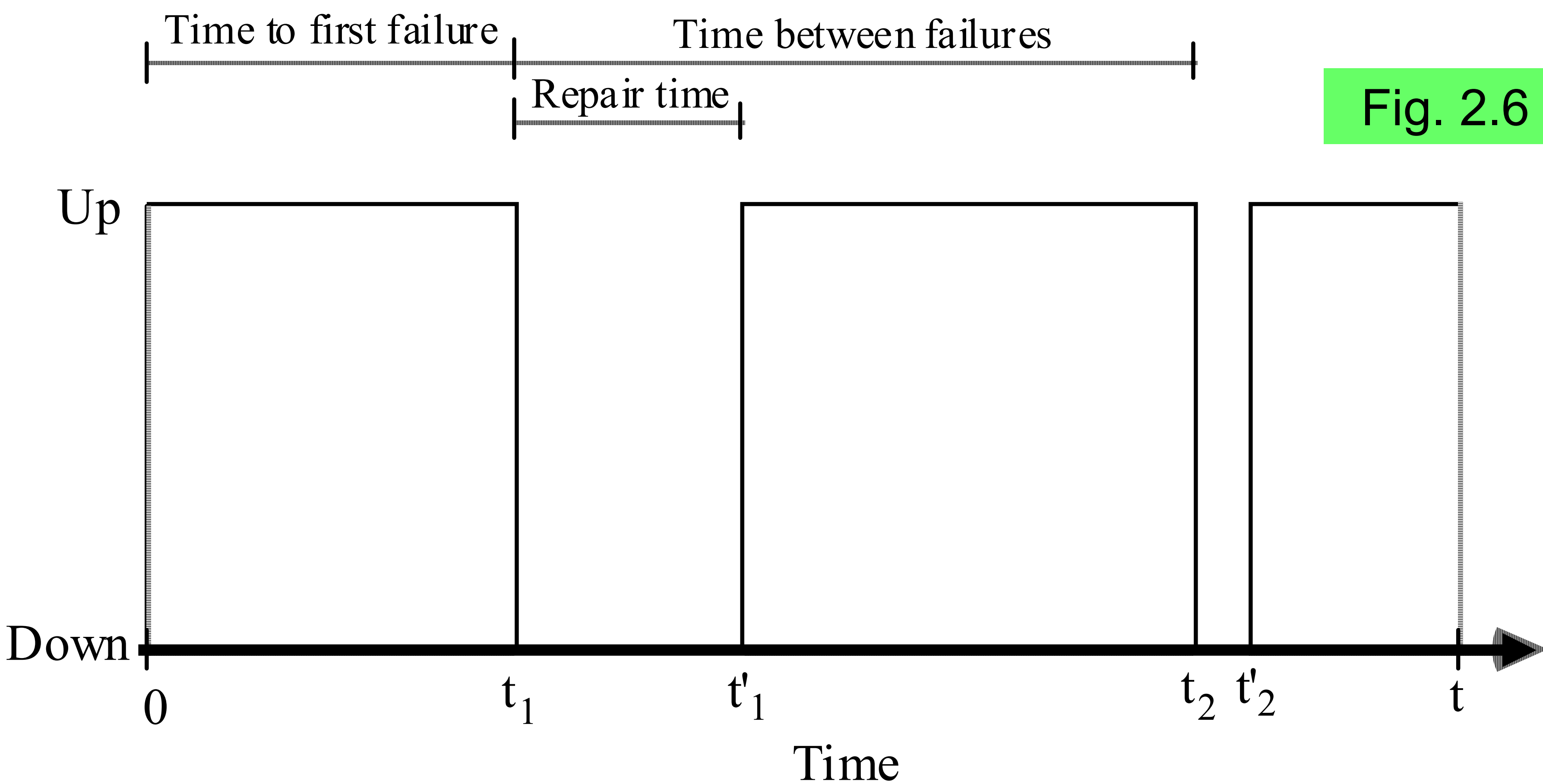


Fig. 2.6



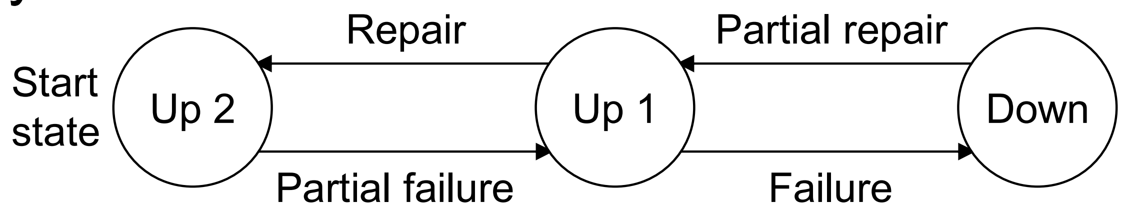
2.4 Performability and MCBF

Performability: *P*

Composite measure, incorporating both performance and reliability

Fig. 2.7

Three-state degradable system



Simple example

Worth of “Up2” twice that of “Up1”

$p_{\mathrm{Up}i}$ = probability system is in state Upi

$P = 2p_{\mathrm{Up2}} + p_{\mathrm{Up1}}$

$p_{\mathrm{Up2}} = 0.92$, $p_{\mathrm{Up1}} = 0.06$, $p_{\mathrm{Down}} = 0.02$, $P = 1.90$

(system performance equiv. To that of 1.9 processors on average)

Question:
What is system availability here?

Performability improvement factor of this system (akin to RIF) relative to a fail-hard system that goes down when either processor fails:

PIF = $(2 - 2 \times 0.92) / (2 - 1.90) = 1.6$

System Up, Partially Up, and Down Times

Important to prevent direct transitions to the "Down" state (**coverage**)

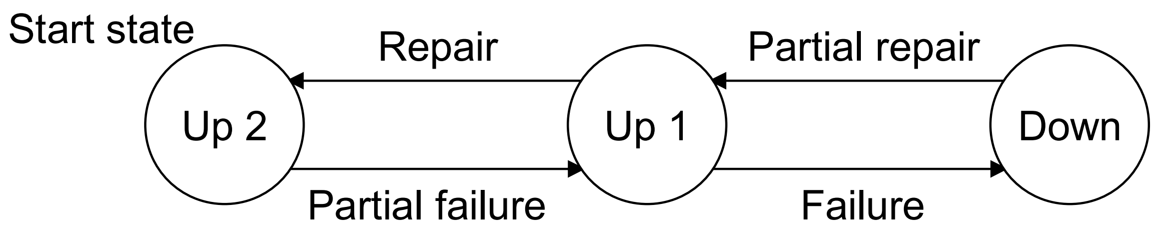
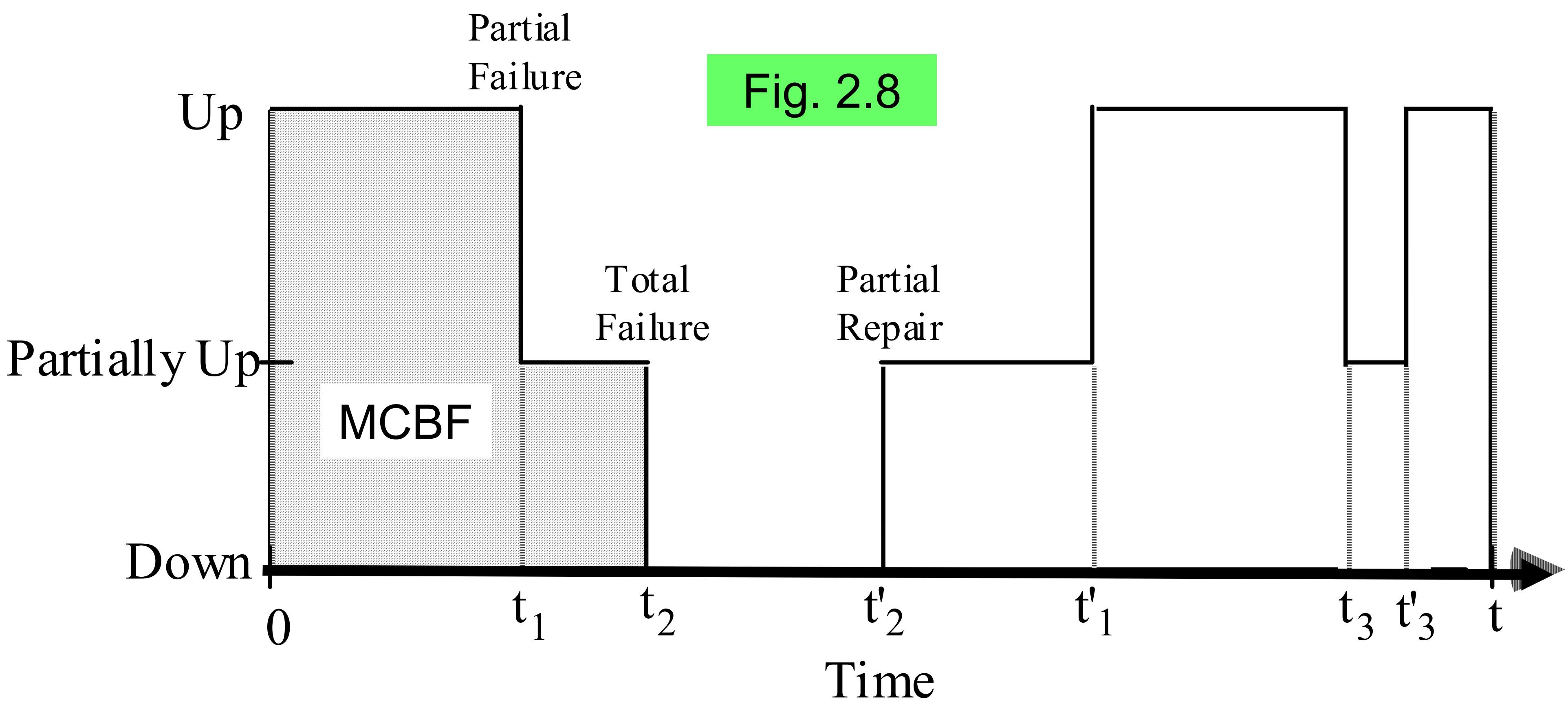


Fig. 2.8



2.5 Integrity and Safety

Integrity and safety are similar

Integrity is inward-looking: capacity to protect system resources (e.g., data)

Safety is outward-looking: consequences of incorrect actions to users

A high-integrity system is robust

Data is not corrupted by low-severity causes

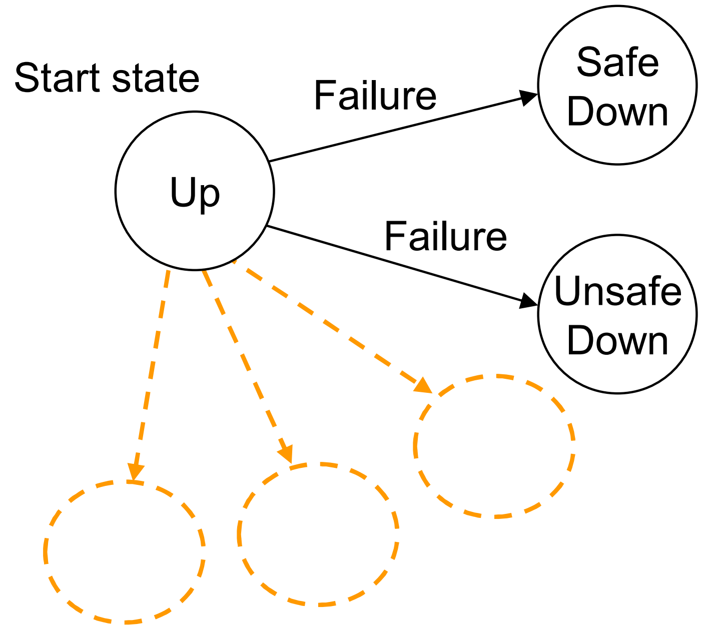
Safety is distinct from reliability: a fail-safe system may not be very reliable in the traditional sense

Basic Safety Assessment

Risk: Prob. of being in "Unsafe Down" state
There may be multiple unsafe states, each with a different consequence (cost)

Fig. 2.9

Three-state fail-safe system



Simple analysis
Lump "Safe Down" state with "Up" state; proceed as in reliability analysis

More detailed analysis
Even though "Safe Down" state is more desirable than "Unsafe Down", it is still not as desirable as the "Up" state; so keeping it separate makes sense

We may have multiple unsafe states

Quantifying Safety

Risk	=	Frequency	×	Magnitude
Consequence / Unit time		Events / Unit time		Consequence / Event

Risk	=	Probability	×	Severity

Magnitude or severity is measured in some suitable unit (say, dollars)

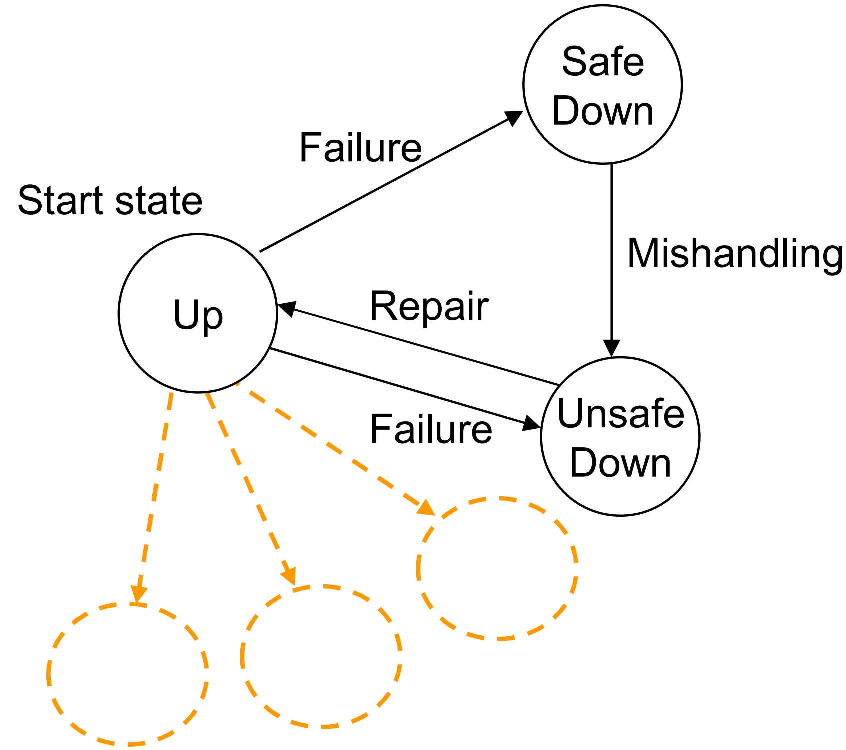
When there are multiple unsafe outcomes, the probability of each is multiplied by its severity (cost) and the results added up

Safety Assessment with More Transitions

If a repair transition is introduced between “Safe Down” and “Up” states, we can tackle questions such as the expected outage of the system in safe mode, and thus its availability

Fig. 2.10

Three-state fail-safe system



Modeling safety procedures

A safe failure can become unsafe or an unsafe failure can turn into a more severe safety problem due to mishandling or human error

This can be easily modeled by adding appropriate transitions

Fallacies of Risk*

1. *Sheer size*: X is accepted. Y is a smaller risk than X. ∴ Y should be accepted.
2. *Converse sheer size*: X is not accepted. Y is a larger risk than X.
   ∴ Y should not be accepted.
3. *Naturalness*: X is natural. ∴ X should be accepted.
4. *Ostrich's*: X has no detectable risk. ∴ X has no unacceptable risks.
5. *Proof-seeking*: There is no scientific proof that X is dangerous.
   ∴ No action should be taken against X.
6. *Delay*: If we wait, we will know more about X.
   ∴ No decision about X should be made now.
7. *Technocratic*: It is a scientific issue how dangerous X is.
   ∴ Scientists should decide whether or not X is acceptable.
8. *Consensus*: We must ask the experts about X.
   ∴ We must ask the experts about a consensus opinion on X
9. *Pricing*: We have to weigh the risk of X against its benefits.
   ∴ We must put a price on the risk of X
10. *Infallibility*: Experts and the public do not have the same attitude about X.
    ∴ The public is wrong about X

*Hansson, S. O., "Fallacies of Risk," *Journal of Risk Research*, Vol. 7, pp. 353-360, 2004.

2.6 Privacy and Security

Privacy and security impairments are human-related

Accidental: operator carelessness, improper reaction to safety warnings

Malicious attacks: Hackers, viruses, and the like

Privacy is compromised when

confidential or personal data are disclosed to unauthorized parties

Security is breached when

account information in a bank is improperly modified, say

Security is distinct from both reliability and safety: a system that automatically locks up when a security breach is suspected may not be very reliable or safe in the traditional sense

Quantifying Security

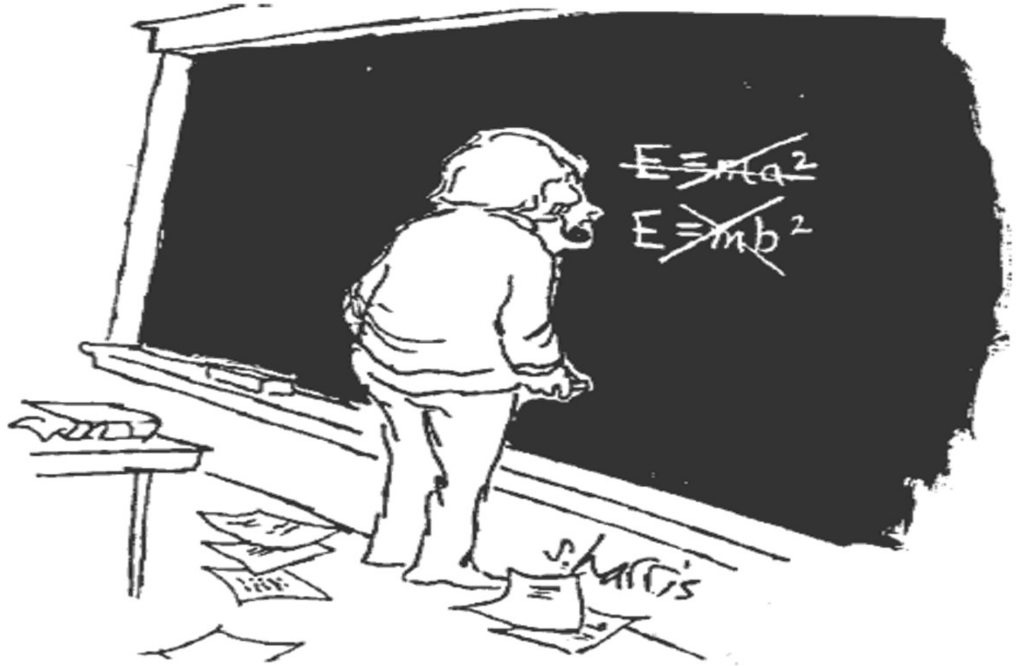
In theory, security can be quantified in the same way as safety:

$$\text{Risk} = \text{Frequency} \times \text{Magnitude}$$

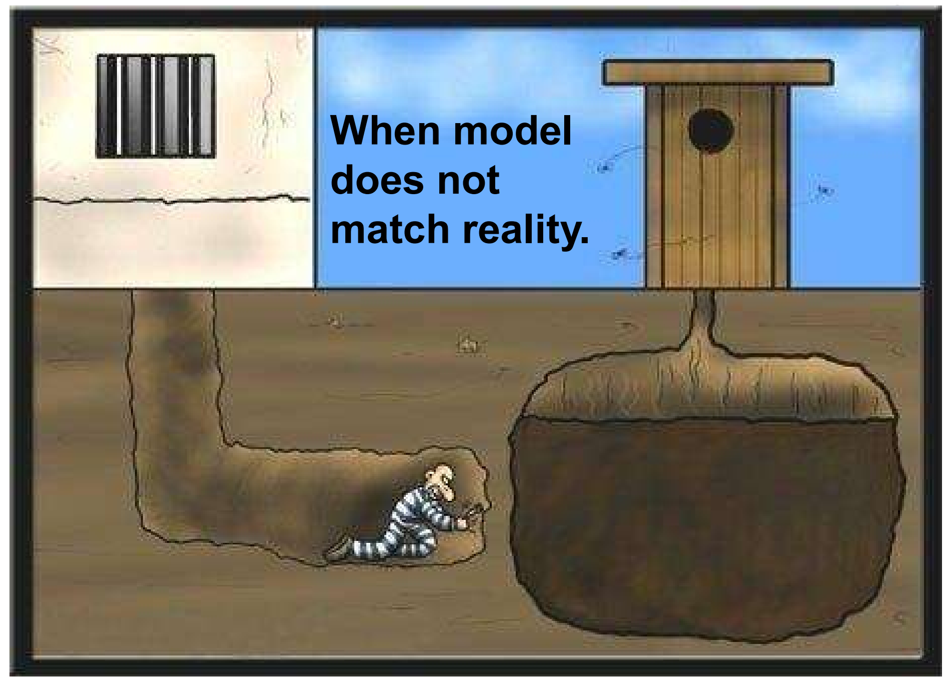
$$\text{Risk} = \text{Probability} \times \text{Severity}$$

But because security breaches are often not accidental, they are ill-suited to probabilistic treatment

3 Combinational Modeling



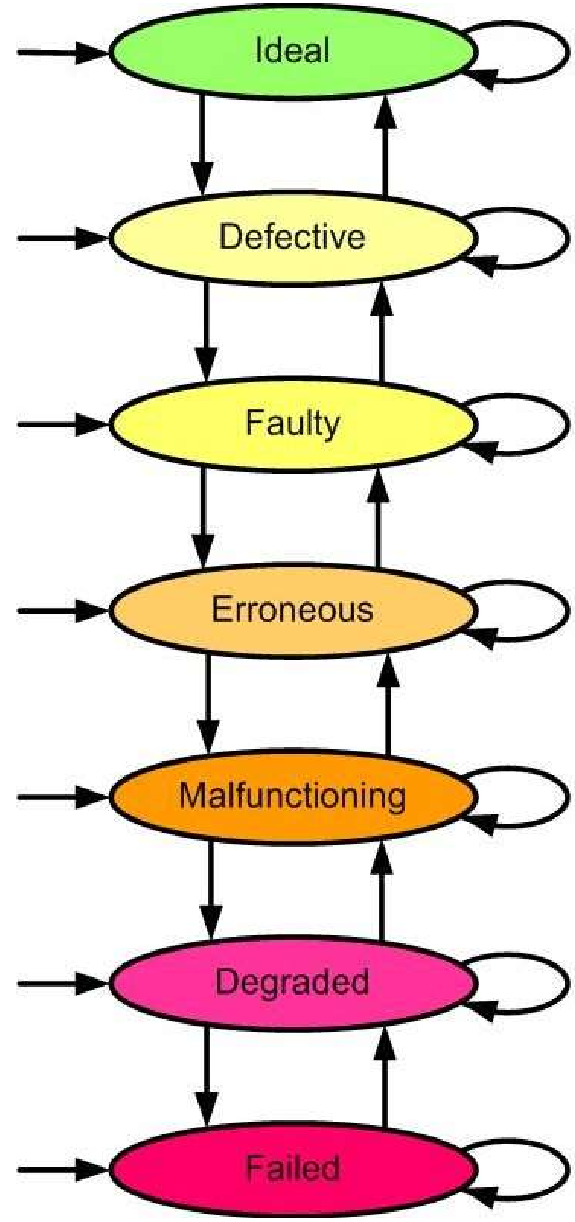
"We installed little monitors because they make all of our problems look smaller."



STRUCTURE AT A GLANCE

Part I — Introduction: Dependable Systems (The Ideal-System View)	Goals / Models	1. Background and Motivation 2. Dependability Attributes 3. Combinational Modeling 4. State-Space Modeling
Part II — Defects: Physical Imperfections (The Device-Level View)	Methods / Examples	5. Defect Avoidance 6. Defect Circumvention 7. Shielding and Hardening 8. Yield Enhancement
Part III — Faults: Logical Deviations (The Circuit-Level View)	Methods / Examples	9. Fault Testing 10. Fault Masking 11. Design for Testability 12. Replication and Voting
Part IV — Errors: Informational Distortions (The State-Level View)	Methods / Examples	13. Error Detection 14. Error Correction 15. Self-Checking Modules 16. Redundant Disk Arrays
Part V — Malfunctions: Architectural Anomalies (The Structure-Level View)	Methods / Examples	17. Malfunction Diagnosis 18. Malfunction Tolerance 19. Standby Redundancy 20. Resilient Algorithms
Part VI — Degradations: Behavioral Lapses (The Service-Level View)	Methods / Examples	21. Degradation Allowance 22. Degradation Management 23. Robust Task Scheduling 24. Software Redundancy
Part VII — Failures: Computational Breaches (The Result-Level View)	Methods / Examples	25. Failure Confinement 26. Failure Recovery 27. Agreement and Adjudication 28. Fail-Safe System Design

Appendix: Past, Present, and Future



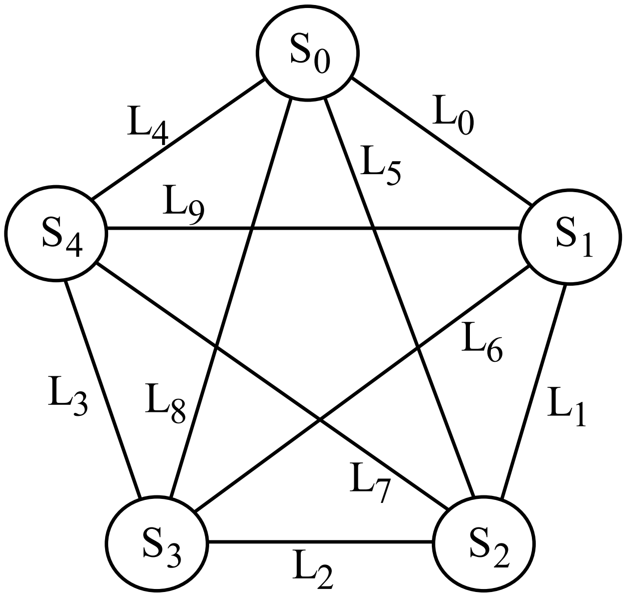
3.1 Modeling by Case Analysis

Revisiting the motivating example: Data files to be stored on five sites so that they remain available despite site and link malfunctions

S = Site availability (a_S in textbook)

L = Link availability (a_L in textbook)

Five-site distributed computer system



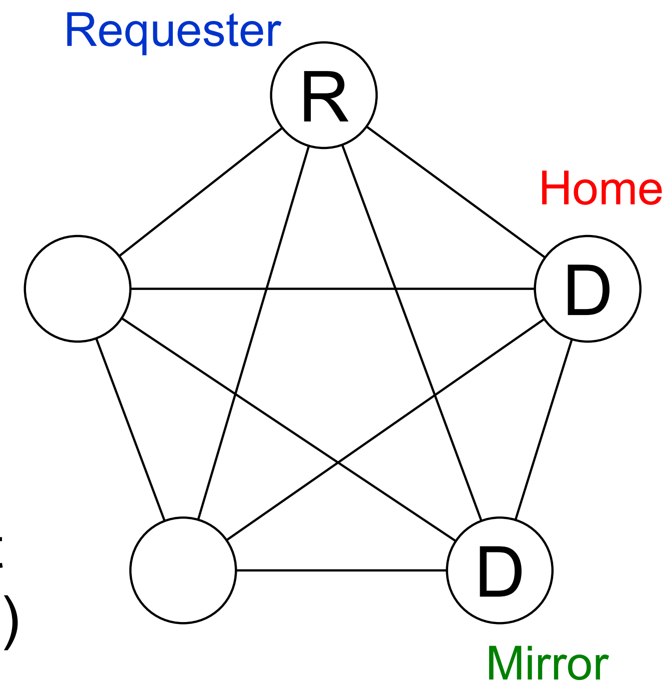
Some possible strategies:

- Duplication on home site and mirror site
- Triplication on home site and 2 backups
- Data dispersion through coding

Here, we ignore the important problem of keeping the replicas consistent and do not worry about malfunction detection and attendant recovery actions

Data Availability with Home and Mirror Sites

Assume data file must be obtained directly from a site that holds it

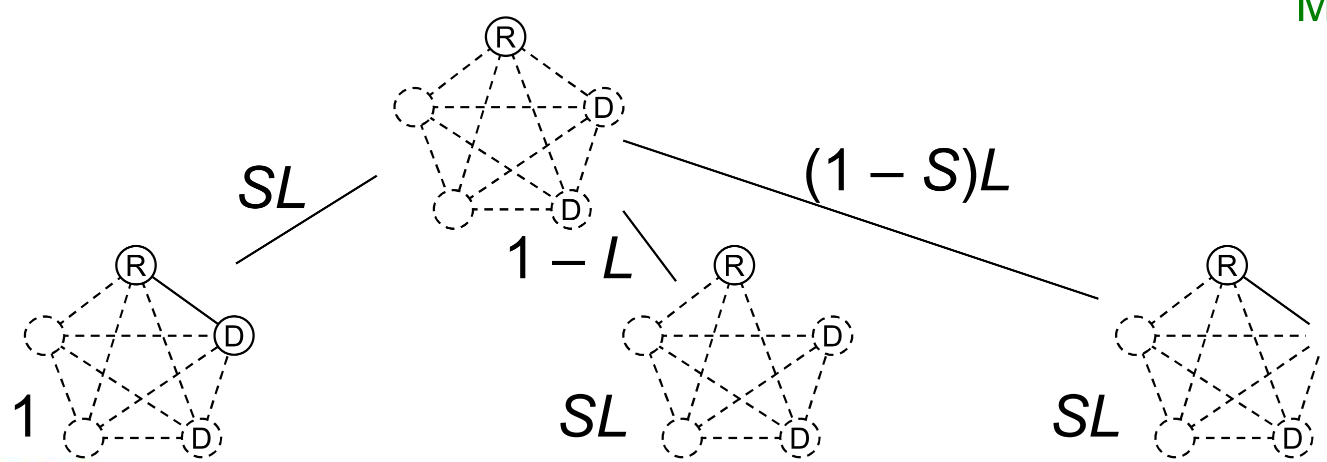


$A = SL + (1 - SL)SL = 2SL - (SL)^2$

For example, $S = 0.99$, $L = 0.95$, $A = 0.9965$
With no redundancy, $A = 0.99 \times 0.95 = 0.9405$

Combinational modeling:
Consider all combinations of circumstances that lead to availability/success (unavailability/failure)

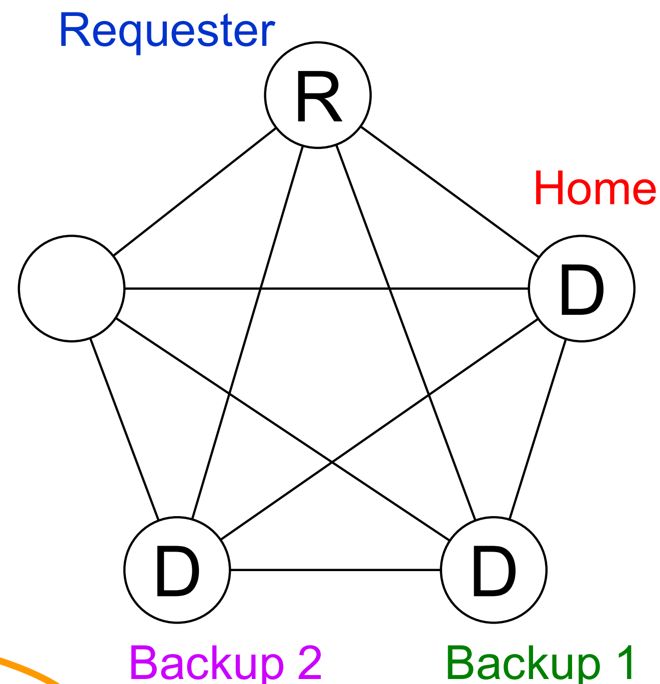
Analysis by considering mutually exclusive subcases

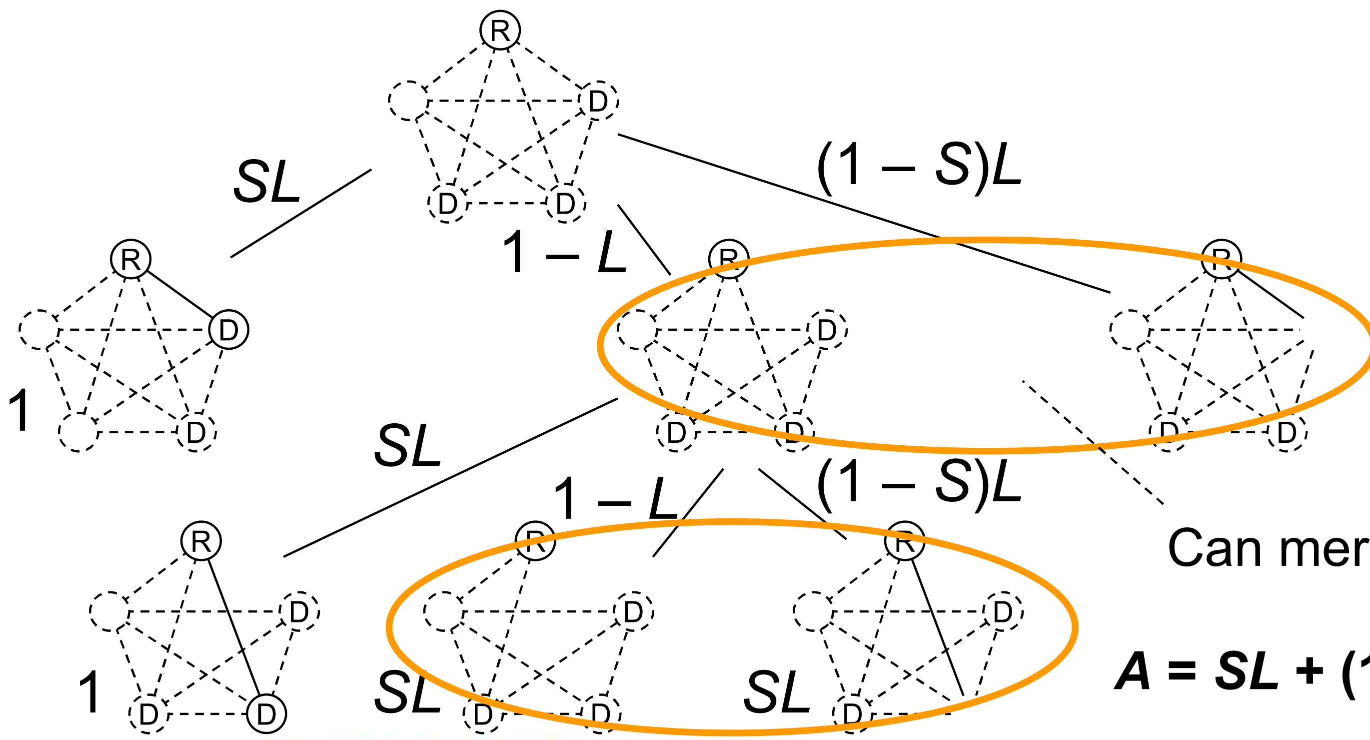


Data Availability with Triplication

$A = SL + (1 - SL)SL + (1 - SL)^2 SL$
$\quad = 3SL - 3(SL)^2 + (SL)^3$

For example, $S = 0.99$, $L = 0.95$, $A = 0.9998$
With duplication, $A = 0.9965$
With no redundancy, $A = 0.9405$





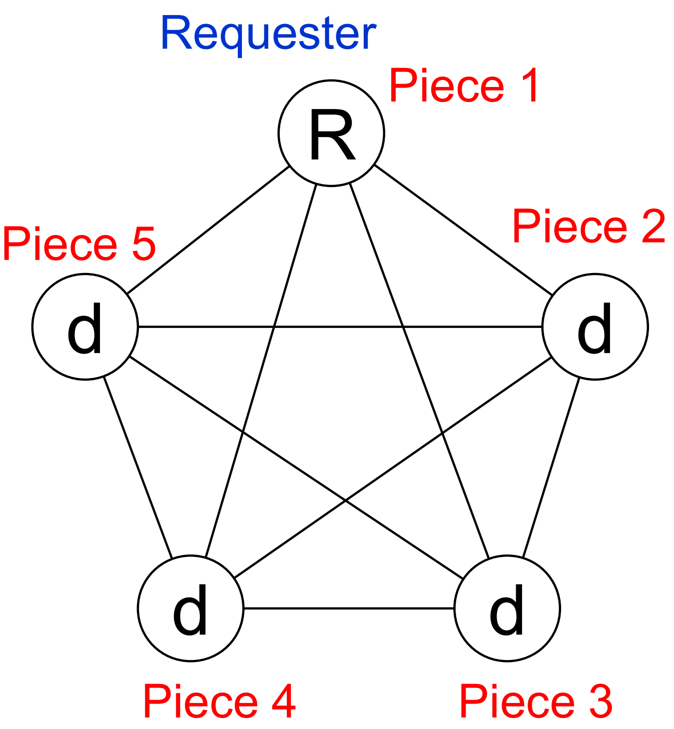
Can merge these two cases

$\boldsymbol{A = SL + (1 - SL)[SL + (1 - SL)SL]}$

Data Availability with File Dispersion

Encode an l-bit file into ≈$5l/3$ bits (67% redund.)
Break encoded file into 5 pieces of length $l/3$
Store each piece on one of the 5 sites

Any 3 of the 5 pieces can be used to reconstruct the original file



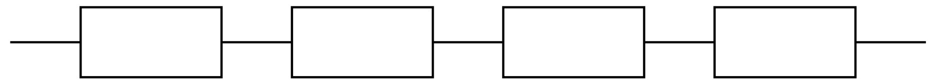
File accessible if 2 out of 4 sites accessible

$$A = (SL)^4 + 4(1 - SL)(SL)^3 + 6(1 - SL)^2(SL)^2$$
$$= 6(SL)^2 - 8(SL)^3 + 3(SL)^4$$

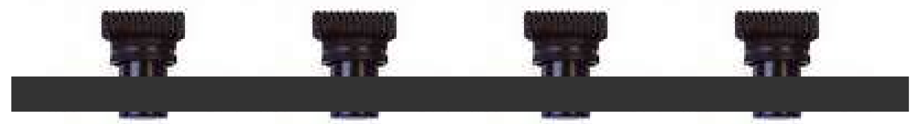
For example, $S = 0.99$, $L = 0.95$, $A = 0.9992$, Redundancy = 67%
With duplication, $A = 0.9965$, Redundancy = 100%
With triplication, $A = 0.9998$, Redundancy = 200%
With no redundancy, $A = 0.9405$

3.2 Series and Parallel Systems

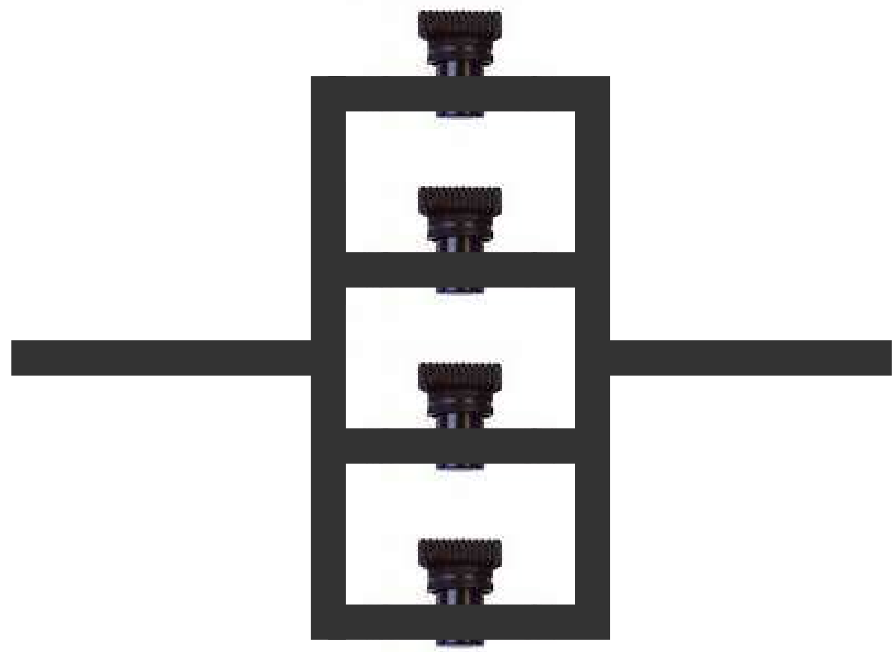
A series system is composed of n units all of which must be healthy for the system to function properly



$R = \prod R_i$



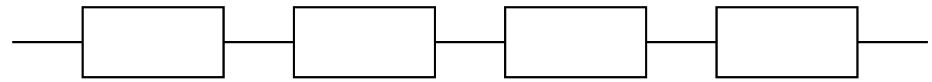
Example: Redundant system of valves in series with regard to stuck-on-shut malfunctions (tolerates stuck-on-open valves)



Example: Redundant system of valves in parallel with regard to to stuck-on-open malfunctions (tolerates stuck-on-shut valves)

Series System: Implications to Design

Assume exponential reliability law



$R_i = \exp[-\lambda_i t]$

$R = \prod R_i = \exp[-(\Sigma\lambda_i)\, t]$

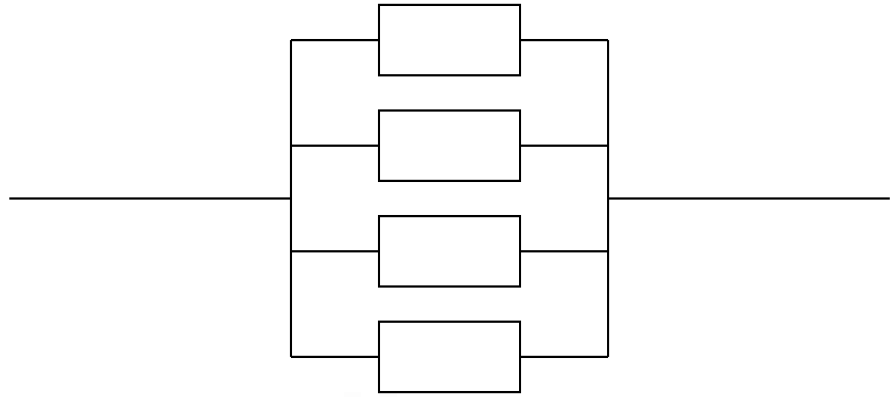
Given the reliability goal r, find the required value of $\Sigma\lambda_i$

Assign a failure rate "budget" to each unit and proceed with its design

May have to reallocate budgets if design proves impossible or costly

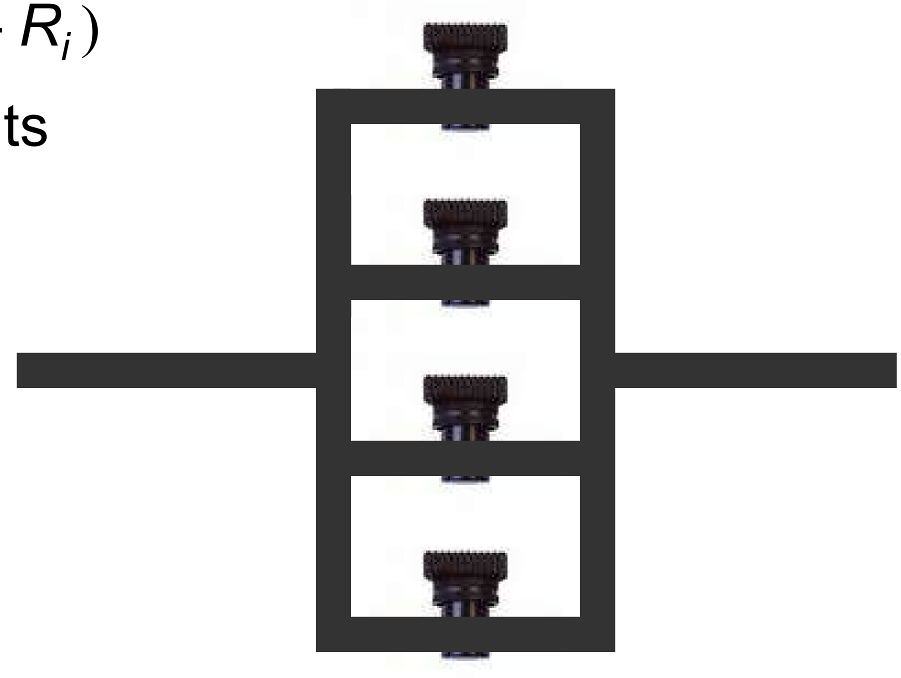
Parallel System

A parallel system is composed of n units, the health of one of which is enough for proper system operation



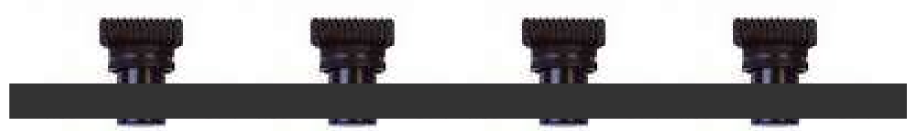
$$1 - R = \prod(1 - R_i) \rightarrow R = 1 - \prod(1 - R_i)$$

That is, the system fails only if all units malfunction



Example: Redundant system of valves in parallel with regard to stuck-on-shut malfunctions (tolerates stuck-on-shut valves)

Example: Redundant system of valves in series with regard to stuck-on-open malfunctions (tolerates stuck-on-open valves)

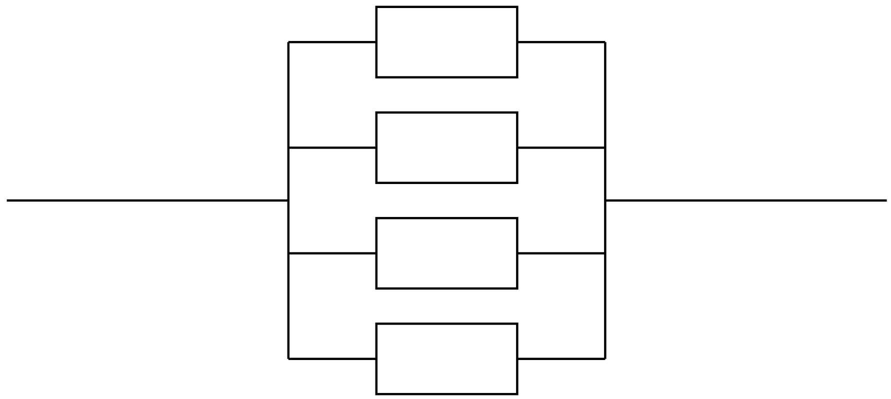


Parallel System: Implications to Design

Assume exponential reliability law

$R_i = \exp[-\lambda_i t]$

$1 - R = \prod(1 - R_i)$



Given the reliability goal r, find the required value of $1 - r = \prod(1 - R_i)$

Assign a failure probability "budget" to each unit

For example, with identical units, $1 - R_m = \sqrt[n]{1 - r}$
Assume $r = 0.9999$, $n = 4 \rightarrow 1 - R_m = 0.1$ (module reliability must be 0.9)
Conversely, for $r = 0.9999$ and $R_m = 0.9$, $n = 4$ is needed

The Perils of Modeling

An example two-way parallel system:

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

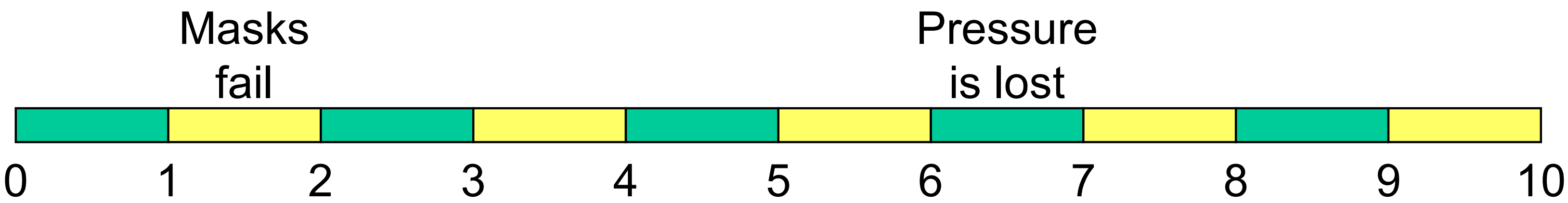
Fatality probability for a 10-hour flight is about $10^{-10} \times 10 = 10^{-9}$ (10^{-9} or less is generally deemed acceptable)

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

Alternate reasoning

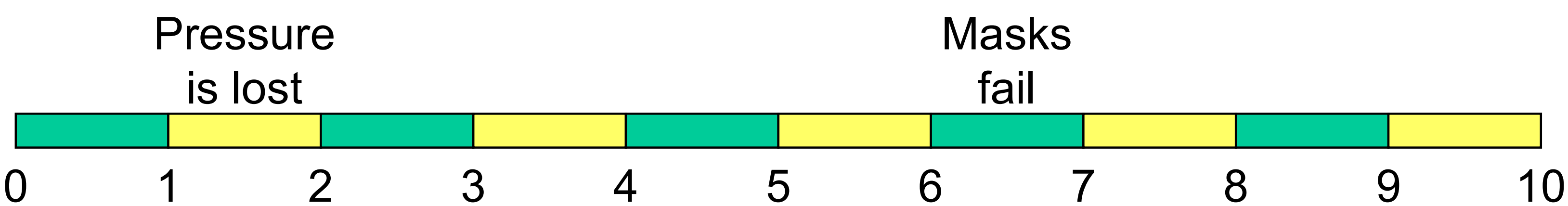
Probability of cabin pressure system failure in 10-hour flight is 10^{-4}
Probability of oxygen masks failing to deploy in 10-hour flight is 10^{-4}
Probability of both systems failing in 10-hour flight is 10^{-8}
Why is this result different from that of our earlier analysis (10^{-9})?
Which one is 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 ($1 - 10^{-9}$)

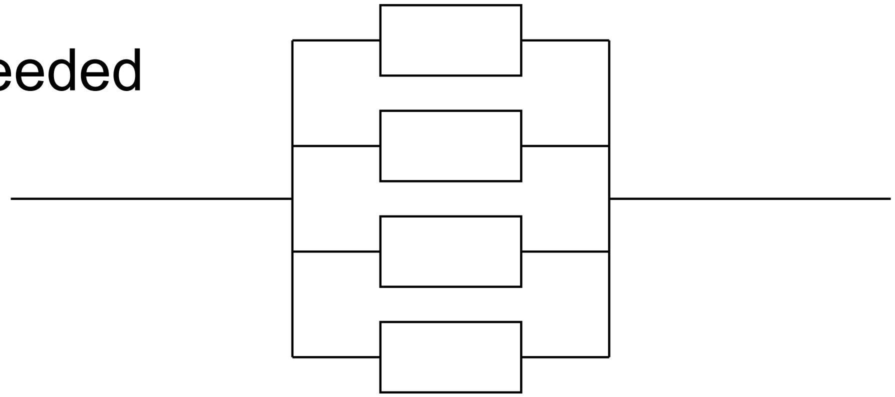


When we multiply the two flight-long failure rates, we are assuming that the failure of these systems would be catastrophic at any time

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

The Concept of Coverage

For $r = 0.9999$ and $R_i = 0.9$, $n = 4$ is needed



Standby sparing: One unit works; others are also active concurrently or they may be inactive (spares)

When a malfunction of the main unit is detected, it is removed from service and an alternate unit is brought on-line; our analysis thus far assumes perfect malfunction detection and reconfiguration

$$R = 1 - (1 - R_\mathrm{m})^n = R_\mathrm{m} \, \frac{1 - (1 - R_\mathrm{m})^n}{1 - (1 - R_\mathrm{m})}$$

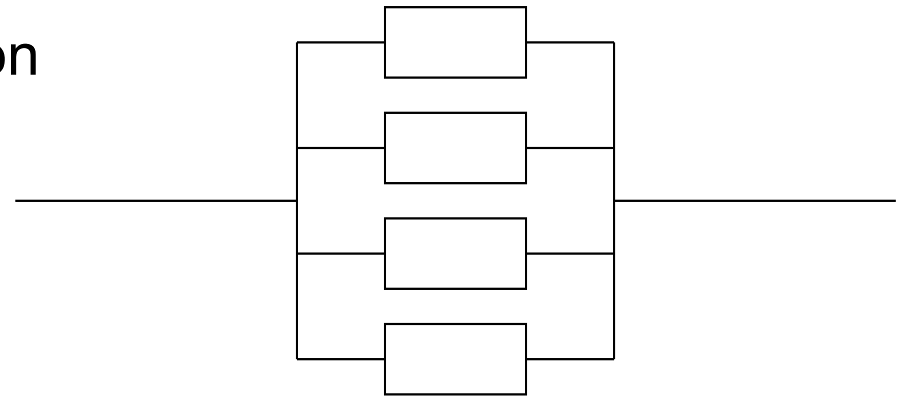
Let the probability of correct malfunction detection and successful reconfiguration be c (coverage factor, $c < 1$)

$$R = R_\mathrm{m} \, \frac{1 - c^n (1 - R_\mathrm{m})^n}{1 - c(1 - R_\mathrm{m})}$$

See [Siew92], p. 288

Impact of Coverage on System Reliability

c: prob. of correct malfunction detection and successful reconfiguration

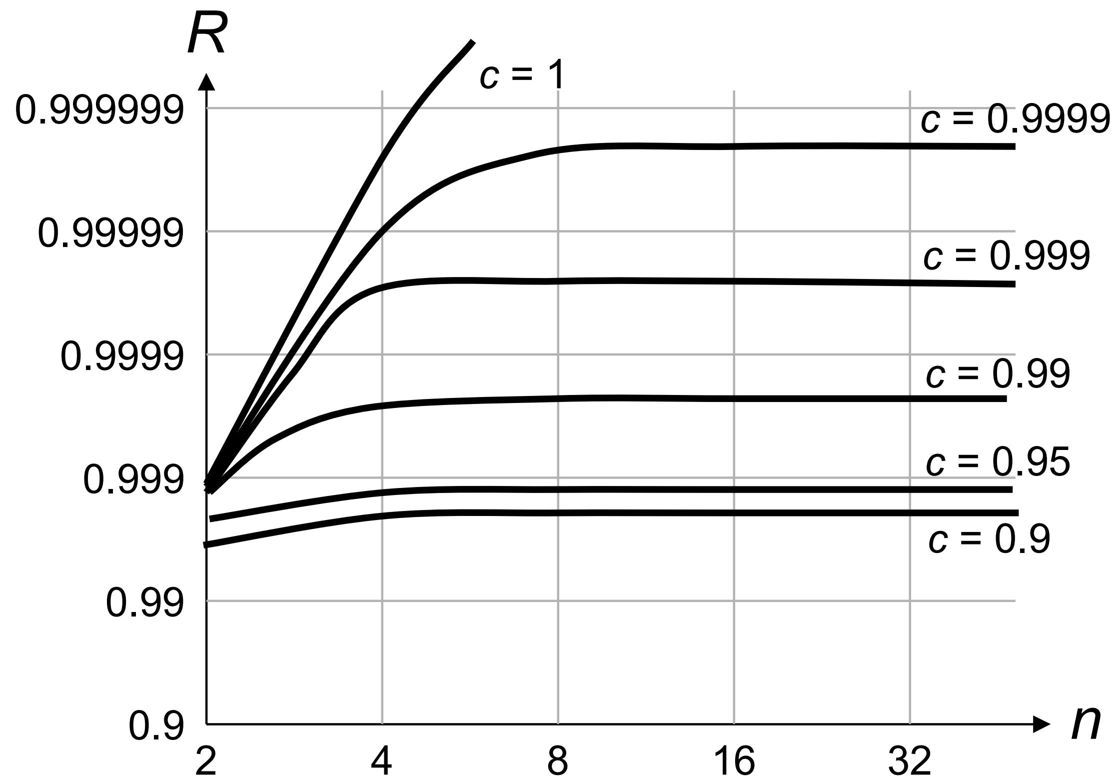


$$R = R_m \frac{1 - c^n(1 - R_m)^n}{1 - c(1 - R_m)}$$

Assume $R_m = 0.95$
Plot R as a function of n for
c = 0.9, 0.95, 0.99, 0.999, 0.9999, 1

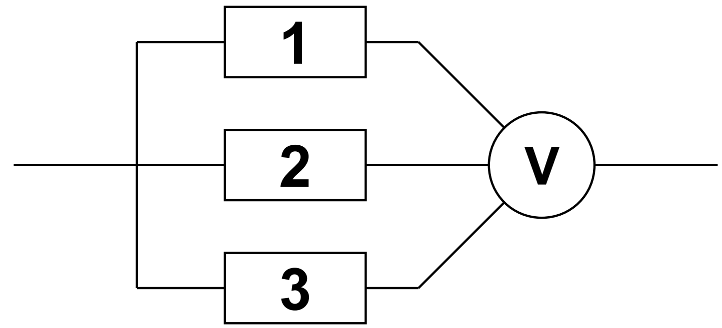
Unless c is near-perfect, adding more spares has no significant effect on reliability

In practice c is not a constant and may deteriorate with more spares; so too many spares may be detrimental to reliability



3.3 Classes of *k*-out-of-*n* Systems

There are *n* modules, any *k* of which are adequate for proper system functioning



Example: System with 2-out-of-3 voting
Assume perfect voter

$$R = R_1 R_2 R_3 + R_1 R_2 (1 - R_3) + R_2 R_3 (1 - R_1) + R_3 R_1 (1 - R_2)$$

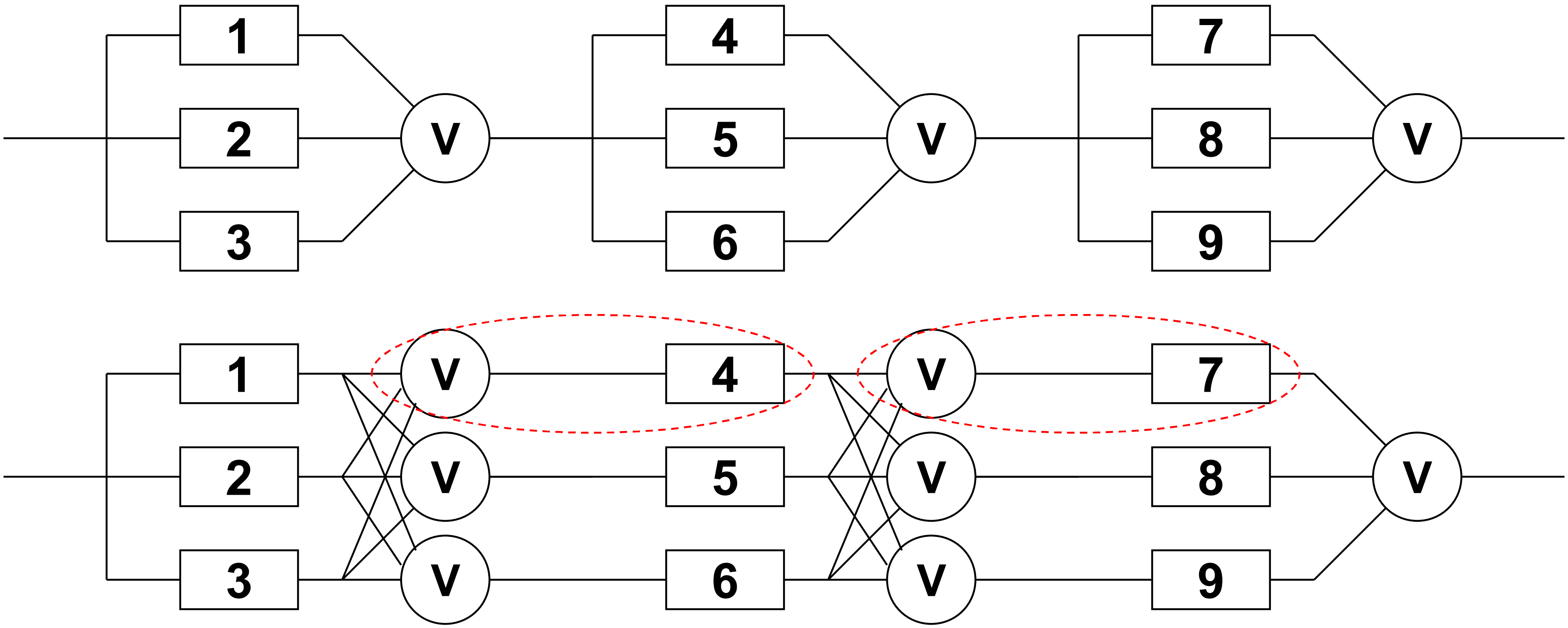
With all units having the same reliability R_m and imperfect voter:

$$R = (3R_m^2 - 2R_m^3) R_v$$ Triple-modular redundancy (TMR)

$$R = \Sigma_{j = k \text{ to } n} \binom{n}{j} R_m^j (1 - R_m)^{n-j}$$ *k*-out-of-*n* system in general

Assuming that any 2 malfunctions in TMR lead to failure is pessimistic
With binary outputs, we can model compensating errors
(when two malfunctioning modules produce 0 and 1 outputs)

n-Modular Redundancy with Replicated Voters



Voters (all but the final one in a chain) no longer critical components

Can model as a series system of 2-out-of-3 subsystems

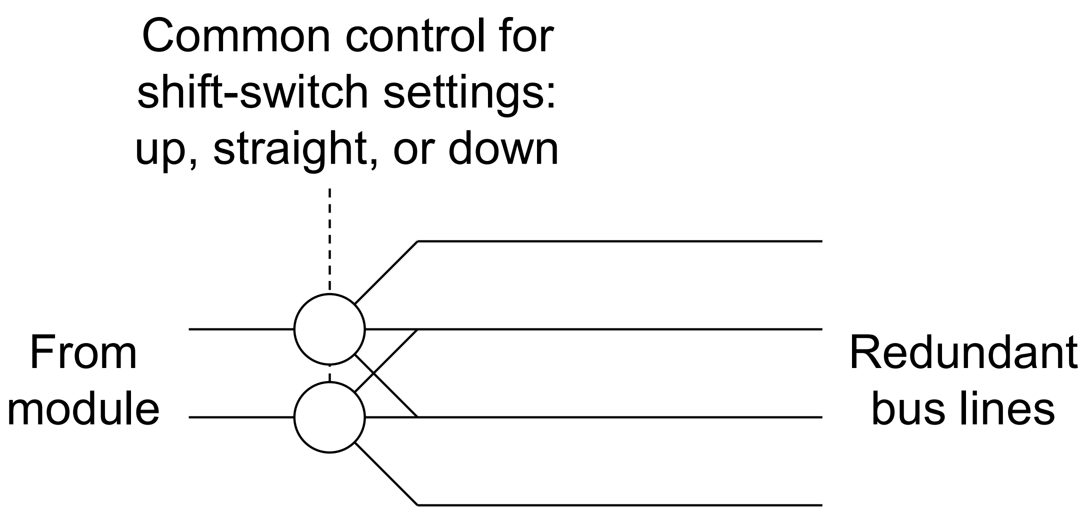
Consecutive *k*-out-of-*n*:G (*k*-out-of-*n*:F) System

Units are ordered and the functioning (failure) of *k* consecutive units leads to proper system function (system failure)

Ordering may be linear (usual case) or circular

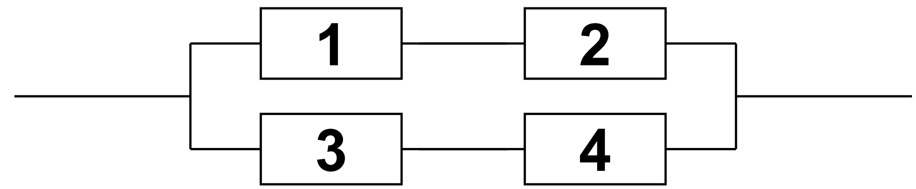
Example: System of street lights may be considered a consecutive 2-out-of-*n*:F system

Example: The following redundant bus reconfiguration scheme is a consecutive 2-out-of-4:G system

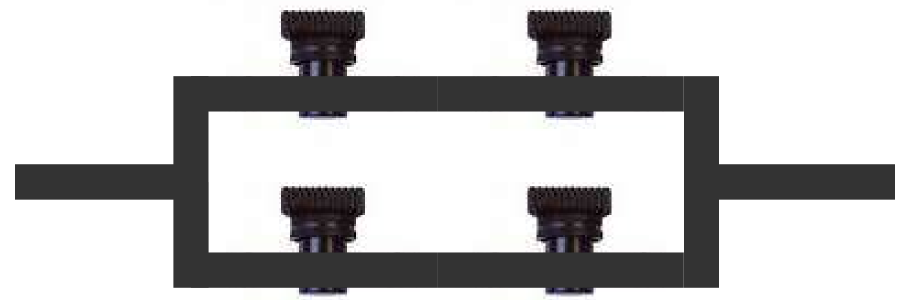


3.4 Reliability Block Diagrams

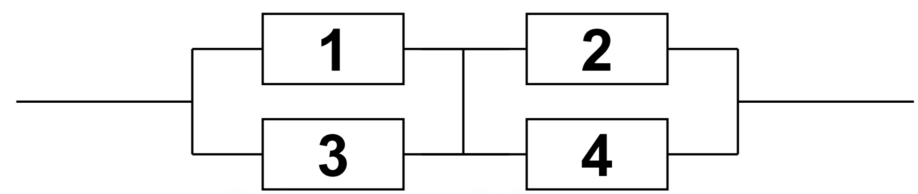
The system functions properly if a string of healthy units connect one side of the diagram to the other



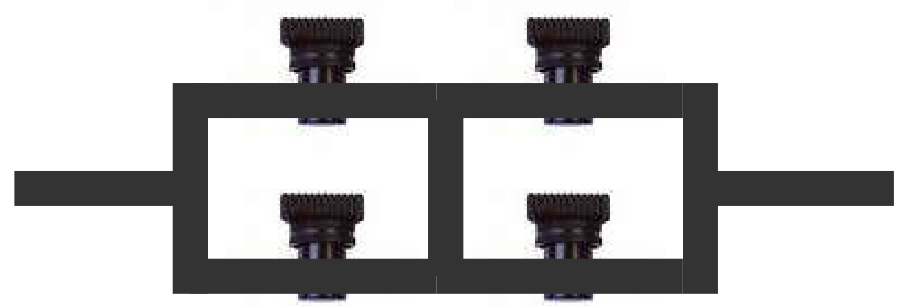
$1 - R = (1 - R_1 R_2)(1 - R_3 R_4)$



Example: Parallel connection of series pairs of valves (tolerates one stuck-on-shut and one stuck-on-open valve)



Example: Series connection of parallel pairs of valves (tolerates one stuck-on-shut and one stuck-on-open valve)

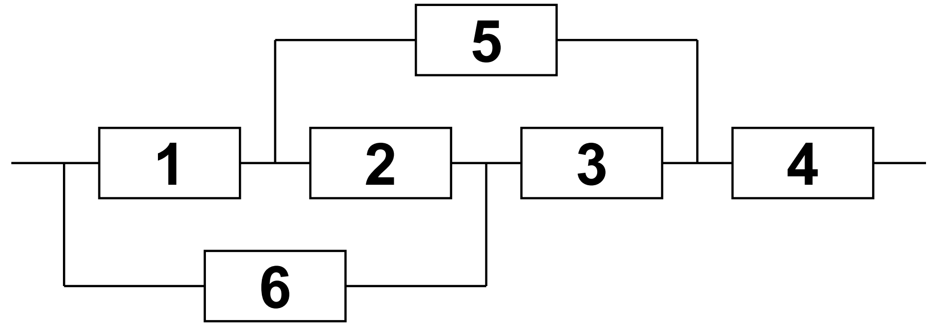


$$R = [1 - (1 - R_1)(1 - R_3)] \times [1 - (1 - R_2)(1 - R_4)]$$

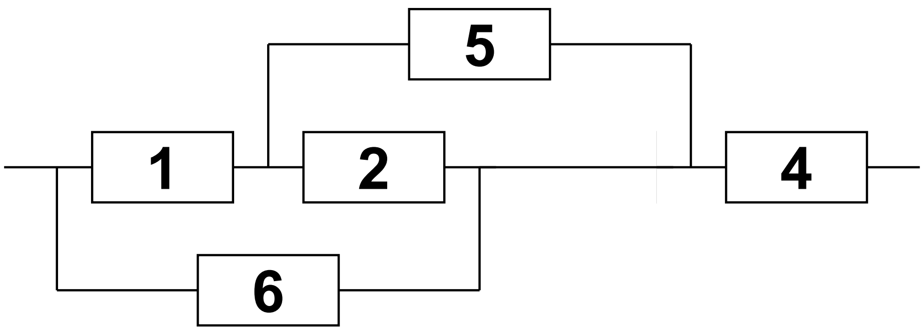
Non-Series/Parallel Systems

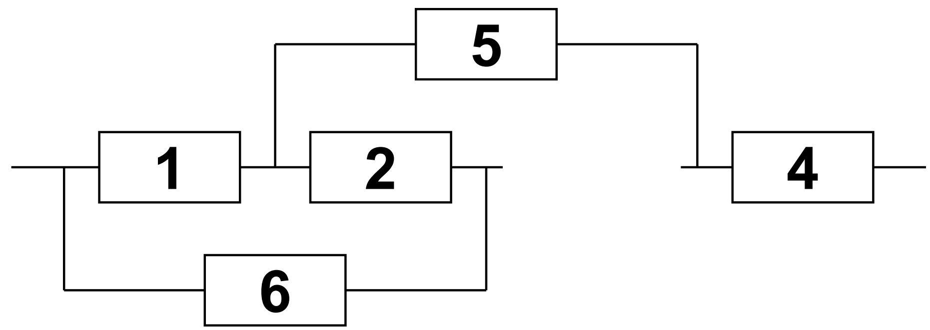
The system functions properly if a string of healthy units connect one side of the diagram to the other

We can think of Unit 5 as being able to replace Units 2 and 3



$$R = R_3 \times \text{prob(system OK | Unit 3 OK)} \quad R_{3\text{OK}}$$
$$+ (1 - R_3) \times \text{prob(system OK | Unit 3 not OK)} \quad R_{3\neg\text{OK}}$$



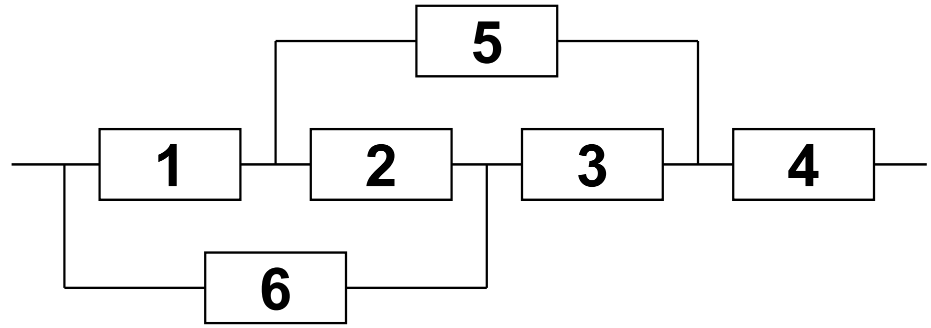


$$R_{3\text{OK}} = [1 - [1 - R_1(\underbrace{1 - (1 - R_2)(1 - R_5)}_{\text{Units 2 and 5 in parallel}})](1 - R_6)]\,R_4 \qquad R_{3\neg\text{OK}} = R_1 R_5 R_4$$

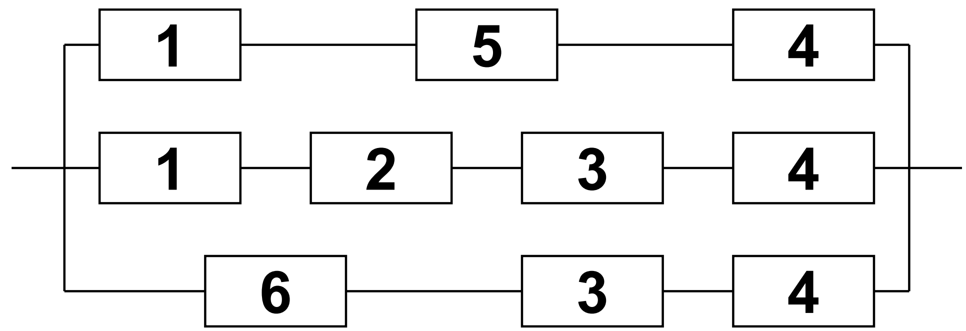
Analysis Using Success Paths

$R \leq 1 - \Pi_i \, (1 - R_{i\text{th success path}})$

This yields an upper bound on reliability because it considers the paths to be independent



$R \leq 1 - (1 - R_1 R_5 R_4)$ [*]
$\quad (1 - R_1 R_2 R_3 R_4)(1 - R_6 R_3 R_4)$



With equal module reliabilities:
$R \leq 1 - (1 - R_m^{\;3})^2 \, (1 - R_m^{\;4})$

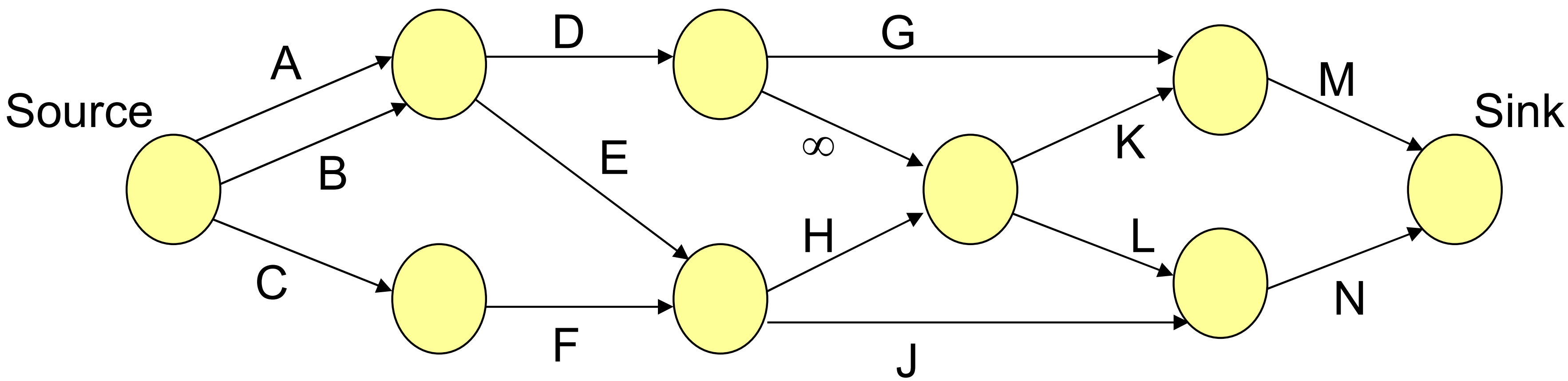
If we expand [*] by multiplying out, removing any power for the various reliabilities, we get an exact reliability expression

$R = 1 - (1 - R_1 R_4 R_5)(1 - R_3 R_4 R_6 - R_1 R_2 R_3 R_4 + R_1 R_2 R_3 R_4 R_6)$
$\quad = R_3 R_4 R_6 + R_1 R_2 R_3 R_4 - R_1 R_2 R_3 R_4 R_6 + R_1 R_4 R_5 - R_1 R_3 R_4 R_5 R_6$
$\quad\quad - R_1 R_2 R_3 R_4 R_5 + R_1 R_2 R_3 R_4 R_5 R_6$ (Verify for the case of equal R_j)

3.5 Reliability Graphs

A reliability graph is a schematic representation of system components, their interactions, and their roles in proper system operation

Use generalized series-parallel connections to visualize success paths, which are directed paths from a source node to a sink node (both unique)



Each module name labels one edge: module failure = edge disconnect

An edge labeled "∞" is never disconnected

3.6 The Fault-Tree Method

Top-down approach to failure analysis:

Start at the top (tree root) with an undesirable event called a "top event" and then determine all the possible ways that the top event can occur

Analysis proceeds by determining how the top event can be caused by individual or combined lower-level undesirable events

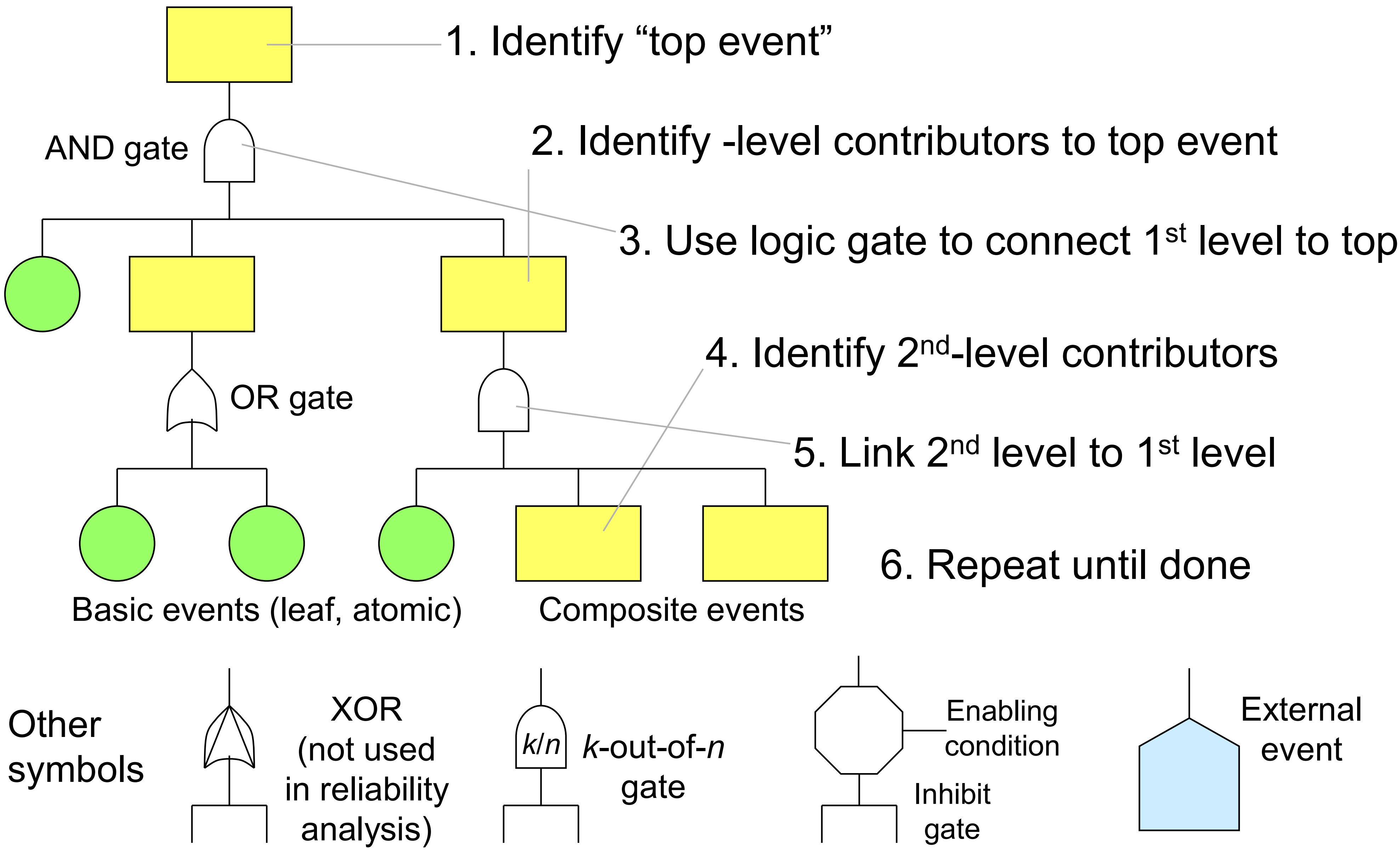
Example:
Top event is "being late for work"
Clock radio not turning on, family emergency, bus not running on time
Clock radio won't turn on if there is a power failure and battery is dead

Quick guide to fault trees: http://www.weibull.com/basics/fault-tree/index.htm

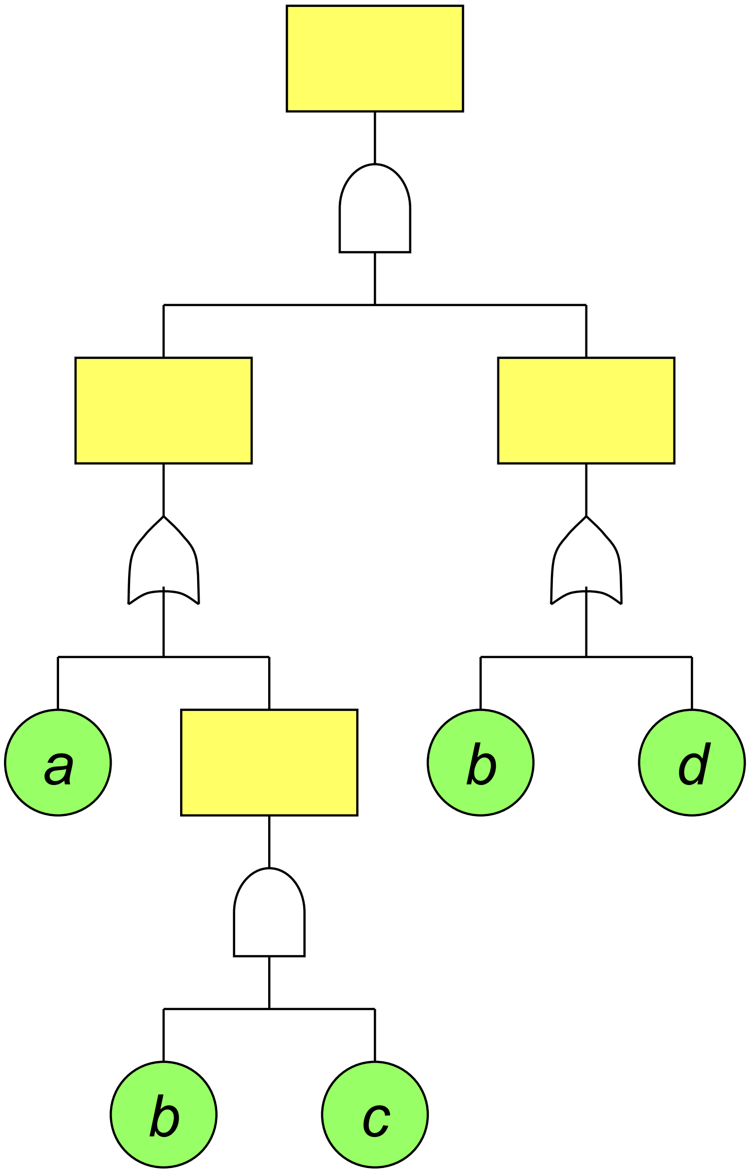
Chapter 38 in *Handbook of Performability Engineering*, Springer, 2008

Fault tree handbook:
http://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0492/sr0492.pdf

Fault Tree Analysis: The Process



Fault Tree Analysis: Cut Set



A cut set is any set of initiators so that the failure of all of them induces the top event

Minimal cut set: A cut set for which no subset is also a cut set

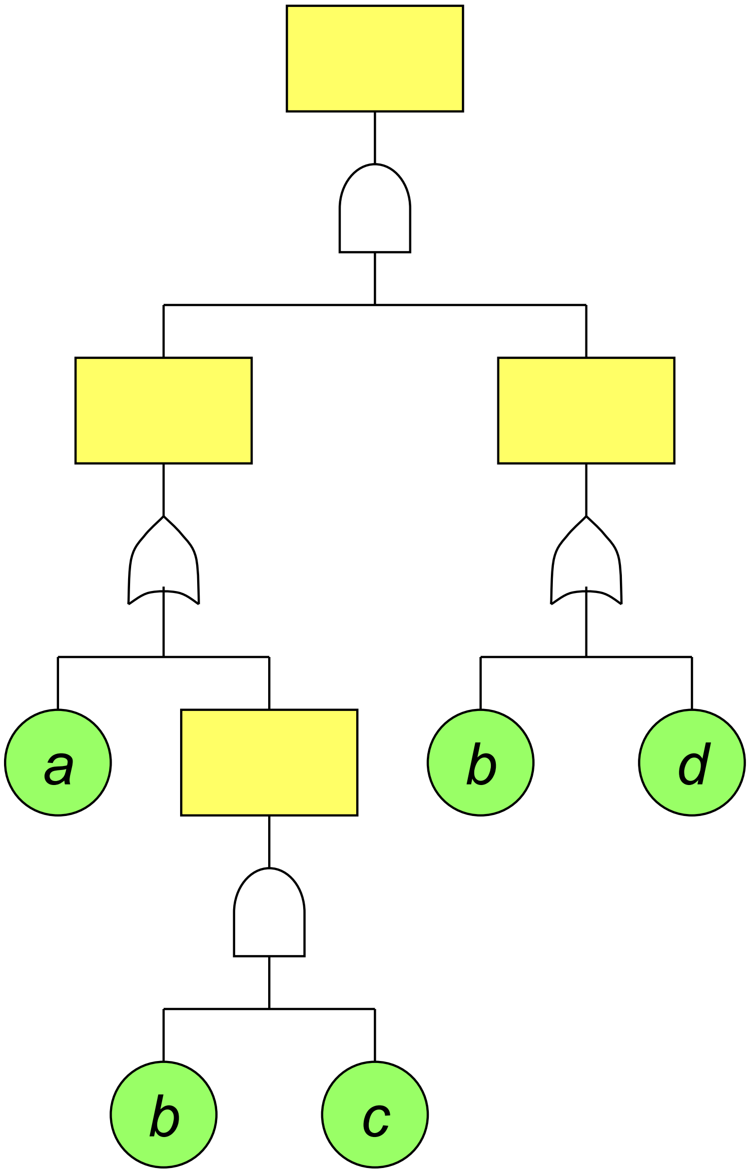
Minimal cut sets for this example:
{*a*, *b*}, {*a*, *d*}, {*b*, *c*}

Just as logic circuits can be transformed to different (simpler) ones, fault trees can be manipulated to obtain equivalent forms

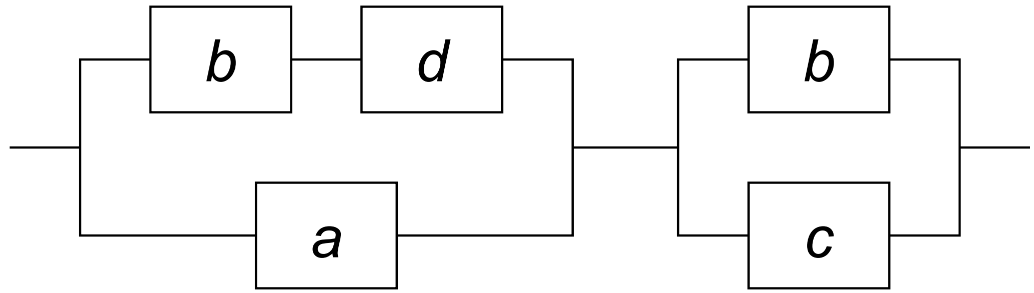
Path set: Any set of initiators so that if all are failure-free, the top event is inhibited (to derive path sets, exchange AND gates and OR gates and then find cut sets)

What are the path sets for this example?

Converting Fault Trees to Reliability Block Diagrams



Minimal cut sets for this example:
{*a*, *b*}, {*a*, *d*}, {*b*, *c*}

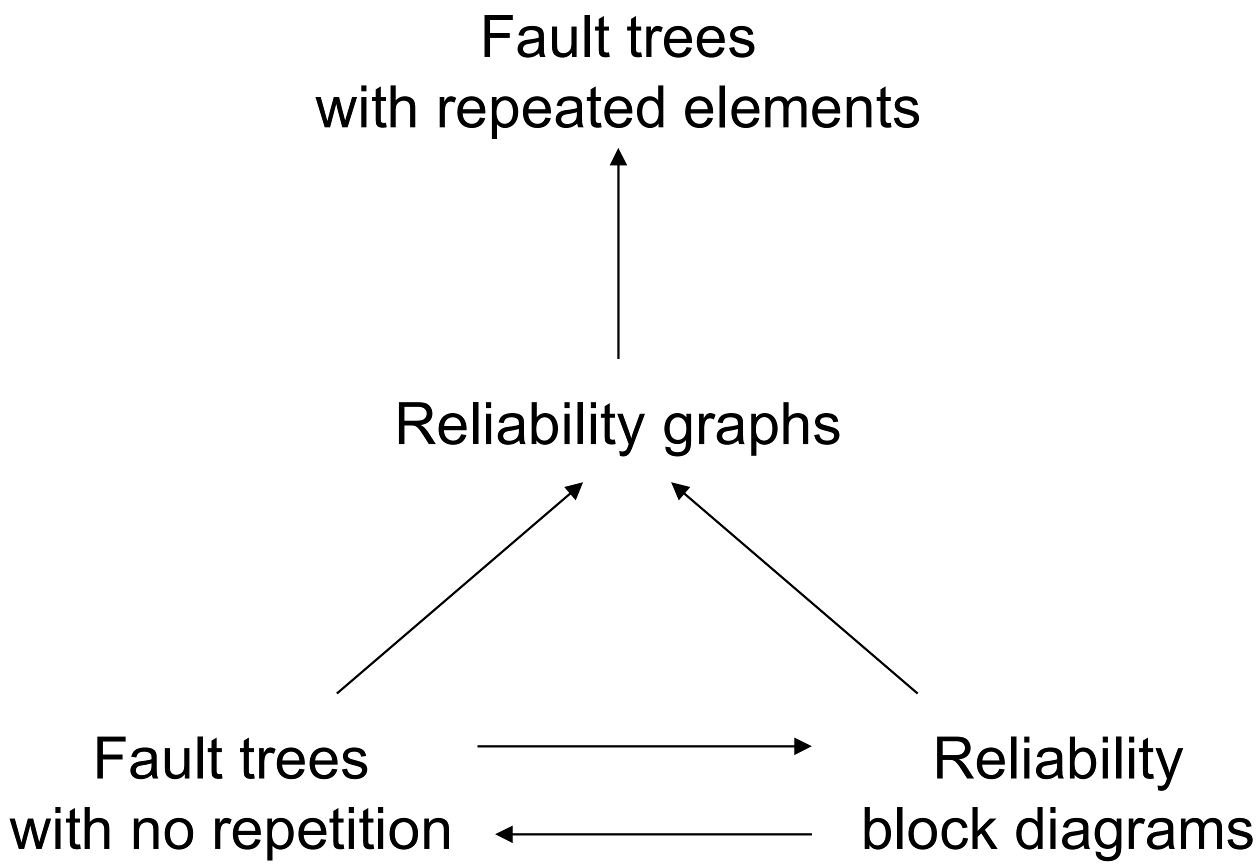


Another example:
Minimal cut set {*a*, *b*}, {*a*, *c*}, {*a*, *d*}, {*c*, *d*, *e*, *f*}

Construct a fault tree for the above
Derive a reliability block diagram
What are the path sets for this example?

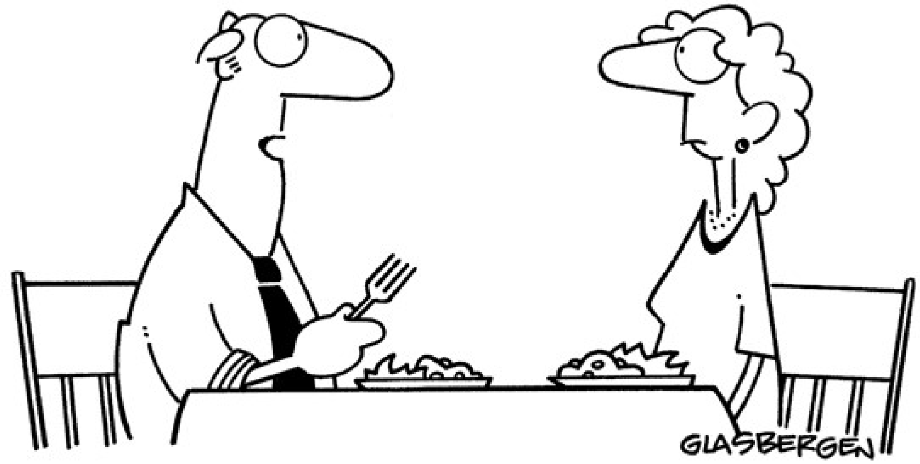
Applications of cut sets:
1. Evaluation of reliability
2. Common-cause failure assessment
3. Small cut set → high vulnerability

Hierarchy of Combinational Models



4 State-Space Modeling

"It's the latest innovation in office safety.
When your computer crashes, an air bag is activated
so you won't bang your head in frustration."

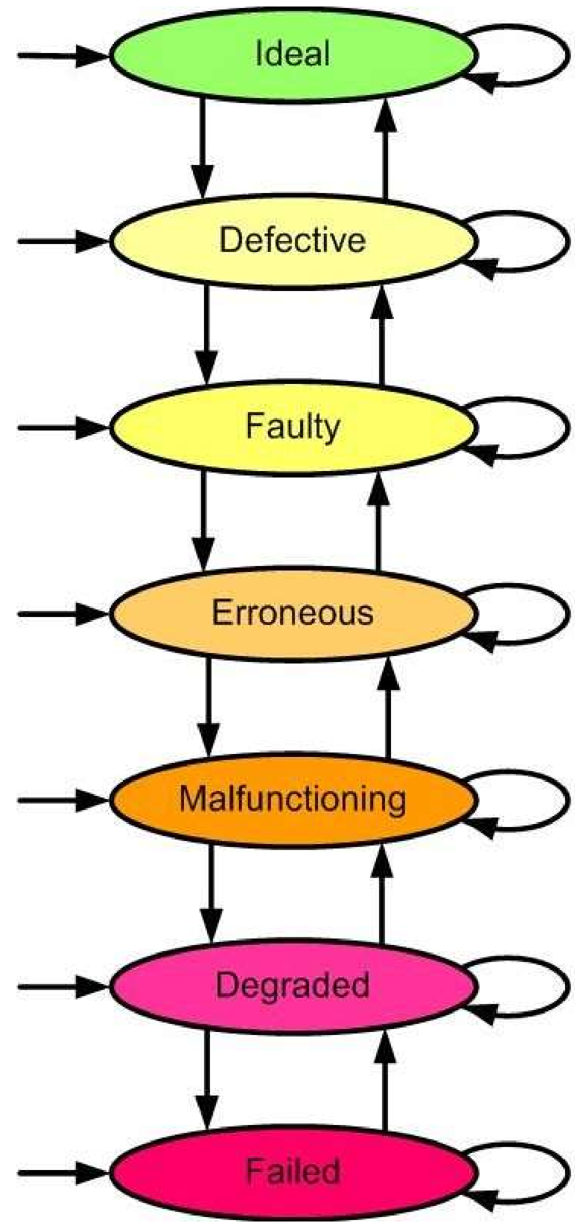


"An amazing thing happened at work today. For 8 minutes, my computer and I were both functional at the same time!"

STRUCTURE AT A GLANCE

Part I — Introduction: Dependable Systems (The Ideal-System View)	Goals / Models	1. Background and Motivation 2. Dependability Attributes 3. Combinational Modeling 4. State-Space Modeling
Part II — Defects: Physical Imperfections (The Device-Level View)	Methods / Examples	5. Defect Avoidance 6. Defect Circumvention 7. Shielding and Hardening 8. Yield Enhancement
Part III — Faults: Logical Deviations (The Circuit-Level View)	Methods / Examples	9. Fault Testing 10. Fault Masking 11. Design for Testability 12. Replication and Voting
Part IV — Errors: Informational Distortions (The State-Level View)	Methods / Examples	13. Error Detection 14. Error Correction 15. Self-Checking Modules 16. Redundant Disk Arrays
Part V — Malfunctions: Architectural Anomalies (The Structure-Level View)	Methods / Examples	17. Malfunction Diagnosis 18. Malfunction Tolerance 19. Standby Redundancy 20. Resilient Algorithms
Part VI — Degradations: Behavioral Lapses (The Service-Level View)	Methods / Examples	21. Degradation Allowance 22. Degradation Management 23. Robust Task Scheduling 24. Software Redundancy
Part VII — Failures: Computational Breaches (The Result-Level View)	Methods / Examples	25. Failure Confinement 26. Failure Recovery 27. Agreement and Adjudication 28. Fail-Safe System Design

Appendix: Past, Present, and Future



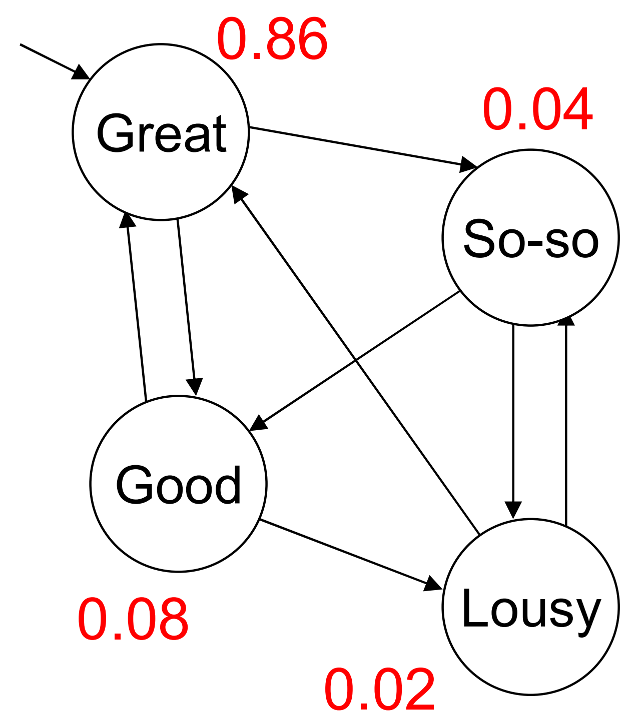
What Is State-Space Modeling?

With respect to availability of resources and computational capabilities, a system can be viewed as being in one of several possible states

The number of states can be large, if we want to make fine distinctions, or it can be relatively small if we lump similar states together

State transitions:
System moves from one state to another as resource availability and computational power change due to various events

State-space modeling entails quantifying transition probabilities so as to determine the probability of the system being in each state; from this, we derive reliability, availability, safety, and other desired parameters



4.1 Markov Chains and Models

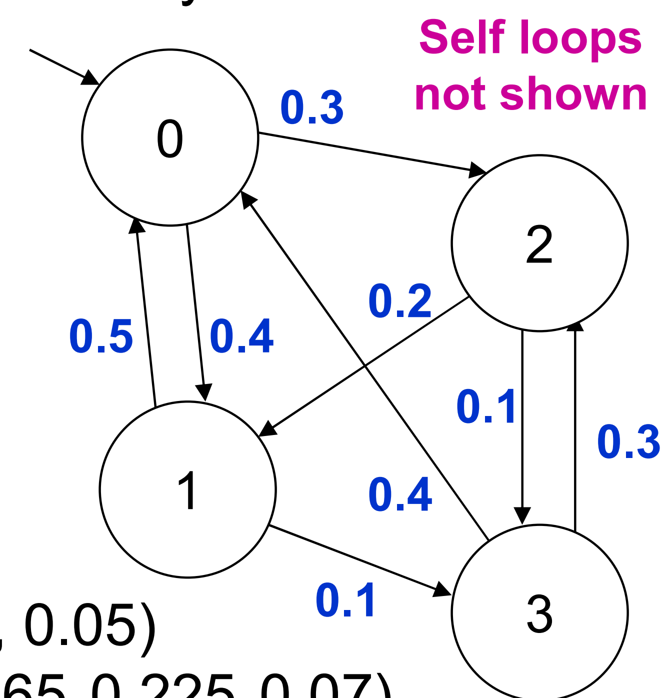
Represented by a state diagram with transition probabilities
Sum of all transition probabilities out of each state is 1

The state of the system is characterized by the vector (s_0, s_1, s_2, s_3)
(1, 0, 0, 0) means that the system is in state 0 Must sum to 1
(0.5, 0.5, 0, 0) means that the system is in state 0 or 1 with equal prob's
(0.25, 0.25, 0.25, 0.25) represents complete uncertainty

Transition matrix: $M = \begin{pmatrix} 0.3 & 0.4 & 0.3 & 0 \\ 0.5 & 0.4 & 0 & 0.1 \\ 0 & 0.2 & 0.7 & 0.1 \\ 0.4 & 0 & 0.3 & 0.3 \end{pmatrix}$

Markov matrix
(rows sum to 1)

$s(t+1) = s(t)\,M$
$s(t+h) = s(t)\,M^h$



Example:
$(s_0, s_1, s_2, s_3) = (0.5, 0.5, 0, 0)\,M = (0.4, 0.4, 0.15, 0.05)$
$(s_0, s_1, s_2, s_3) = (0.4, 0.4, 0.15, 0.05)\,M = (0.34, 0.365, 0.225, 0.07)$

Stochastic Sequential Machines

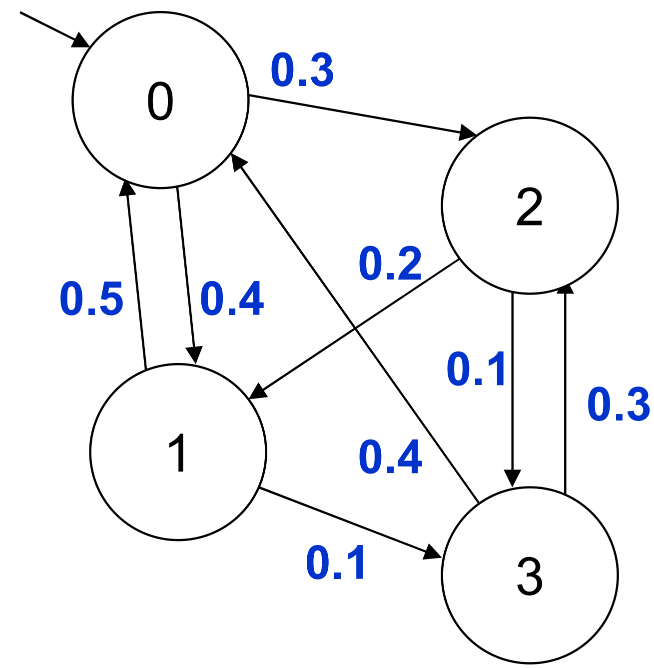
Transition taken from state *s* under input *j* is not uniquely determined
Rather, a number of states may be entered with different probabilities

There will be a separate transition (Markov) matrix for each input value

Transitions, j = 0: $M = \begin{pmatrix} 0.3 & 0.4 & 0.3 & 0 \\ 0.5 & 0.4 & 0 & 0.1 \\ 0 & 0.2 & 0.7 & 0.1 \\ 0.4 & 0 & 0.3 & 0.3 \end{pmatrix}$

Transitions, j = 1: $M = \begin{pmatrix} 0.5 & 0.2 & 0.1 & 0.2 \\ 0.1 & 0.4 & 0.4 & 0.1 \\ 0.3 & 0 & 0.2 & 0.5 \\ 0.2 & 0.6 & 0 & 0.2 \end{pmatrix}$

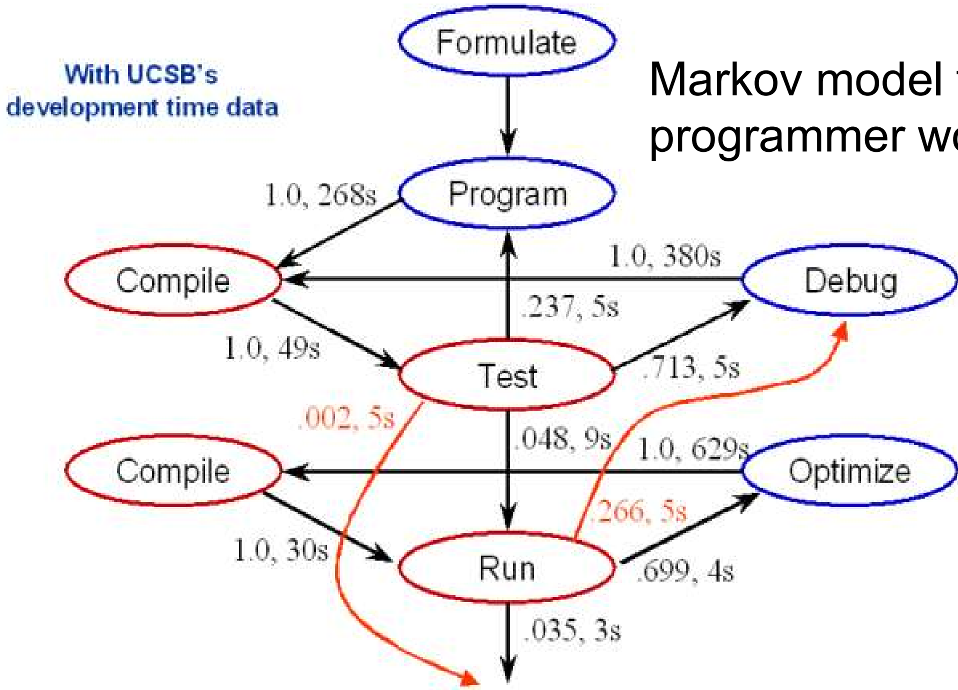
Self loops and transitions for j = 1 not shown

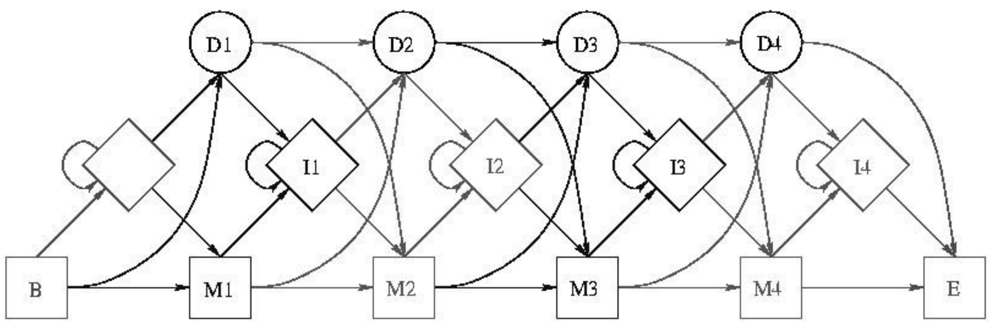


A Markov chain can be viewed as a stochastic sequential machine with no input

Sample Applications of Markov Modeling

Markov model for programmer workflow





"Hidden Markov Model" for recognition problems

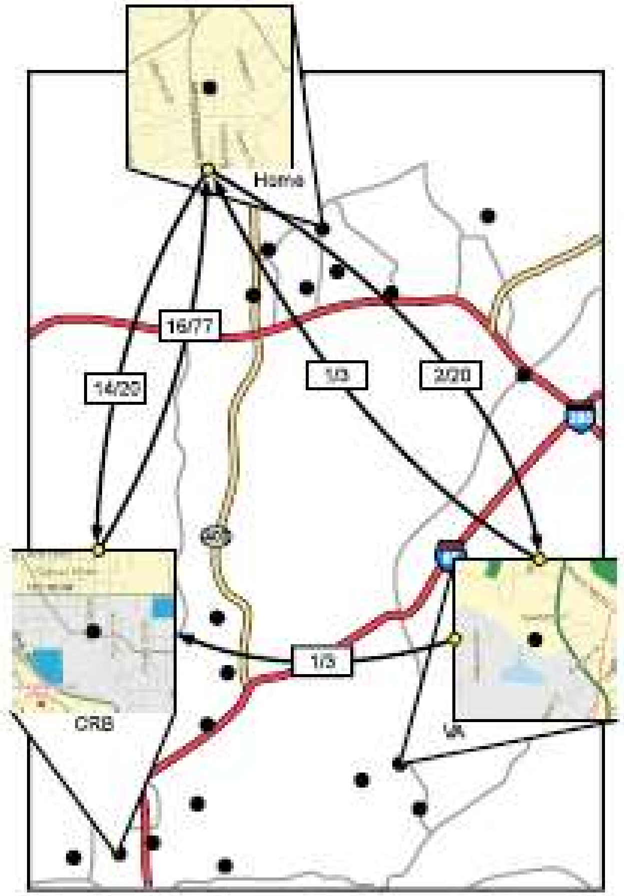
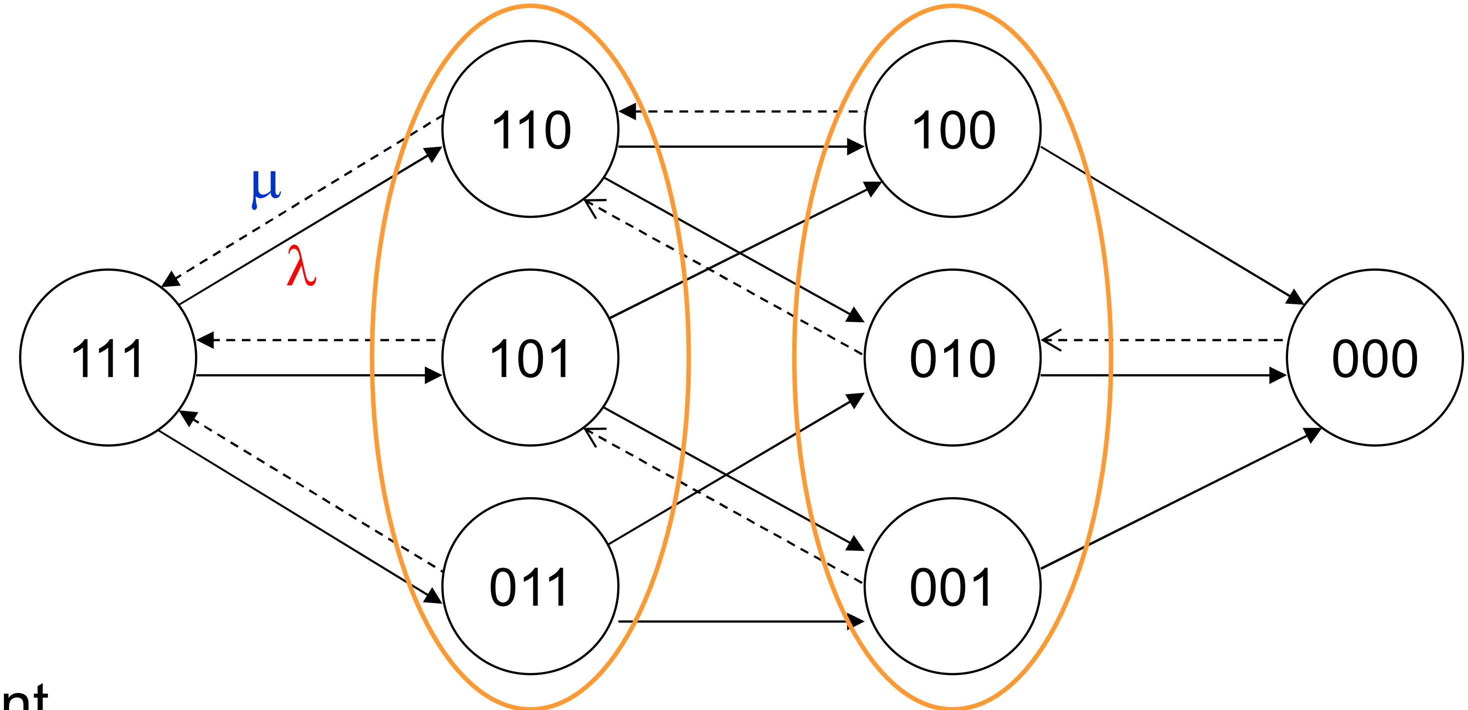


Figure 6: *Partial Markov model of trips made between home, Centennial Research Building (CRB), and Dept. of Veterans Affairs (VA). Because some paths are not shown, the ratios do not sum to 1.*

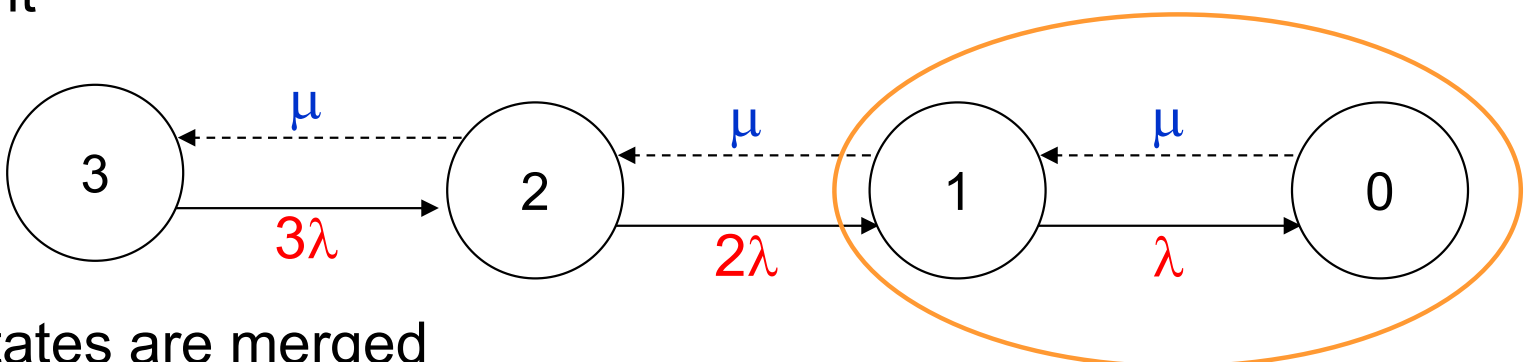
Merging States in a Markov Model

There are three identical units
1 = Unit is up
0 = Unit is down

All solid lines λ
Dashed lines μ



Simpler equivalent model for 3-unit fail-soft system



Whether or not states are merged depends on the model's semantics

Failed state if TMR

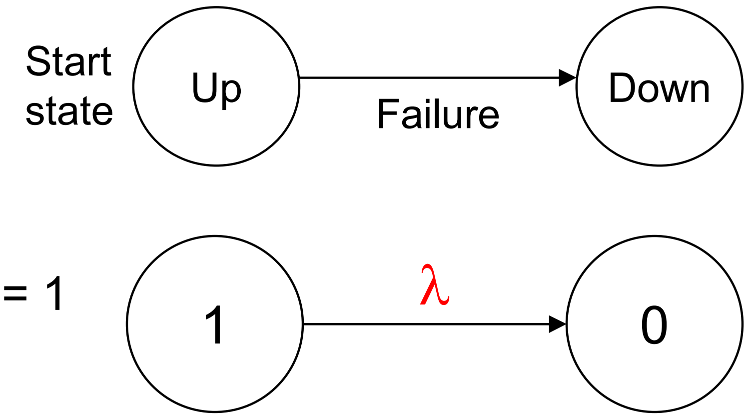
4.2 Modeling Nonrepairable Systems

Rate of change for the probability of being in state 1 is $-\lambda$

$$p'_1 = -\lambda p_1$$
$$p_1 + p_0 = 1$$

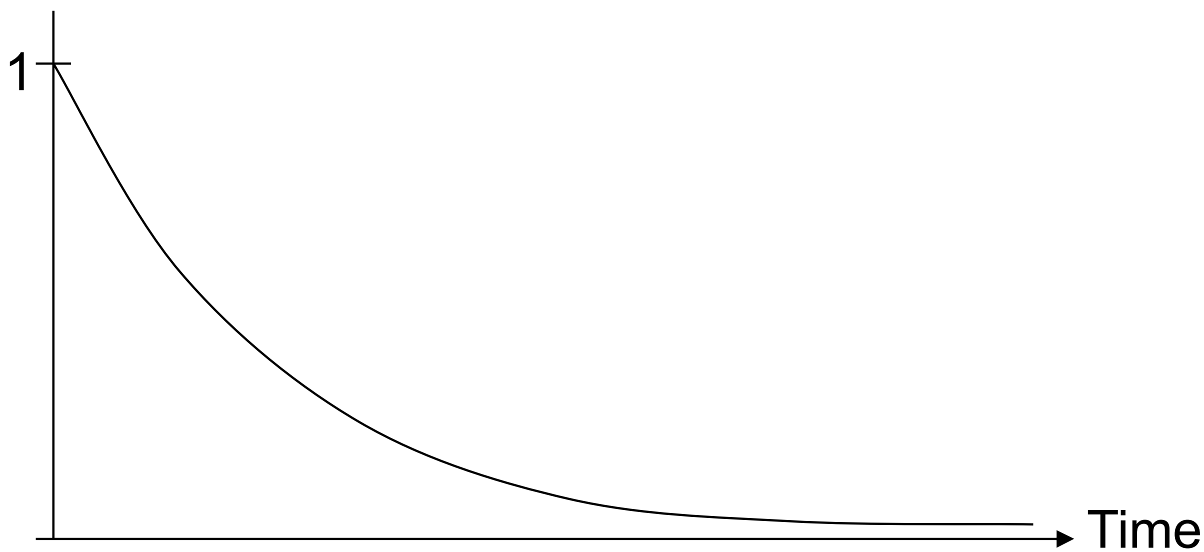
$$p_0 = 1 - e^{-\lambda t}$$
$$p_1 = e^{-\lambda t}$$

Initial condition: $p_1(0) = 1$



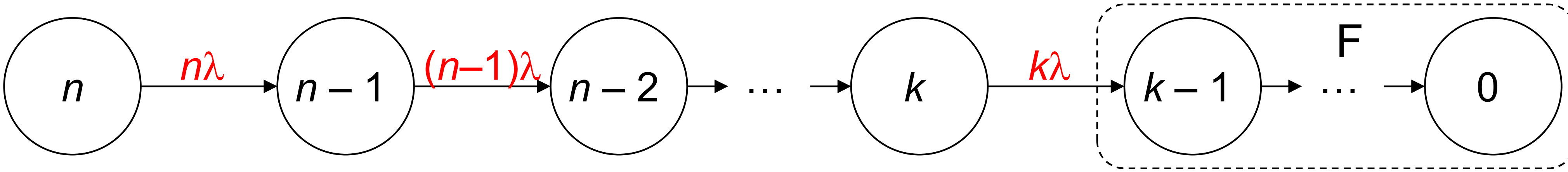
Reliability as a function of time:

$R(t) = p_1(t) = e^{-\lambda t}$



Two-state system: the label λ on this transition means that over time dt, the transition will occur with probability λdt (we are dealing with a continuous-time Markov model)

k-out-of-n Nonrepairable Systems



$$p'_n = -n\lambda p_n$$
$$p'_{n-1} = n\lambda p_n - (n-1)\lambda p_{n-1}$$
$$\vdots$$
$$p'_k = (k+1)\lambda p_{k+1} - k\lambda p_k$$
$$p_n + p_{n-1} + \ldots + p_k + p_F = 1$$

$$p_n = e^{-n\lambda t}$$ Initial condition: $p_n(0) = 1$
$$p_{n-1} = n e^{-(n-1)\lambda t}(1 - e^{-\lambda t})$$
$$\vdots$$
$$p_k = \binom{n}{k} e^{-(n-k)\lambda t}(1 - e^{-\lambda t})^k$$
$$p_F = 1 - \textstyle\sum_{j=k \text{ to } n} p_j$$

In this case, we do not need to resort to more general method of solving linear differential equations (LaPlace transform, to be introduced later)

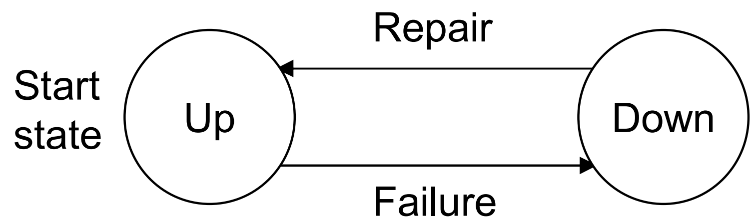
The first equation is solvable directly, and each additional equation introduces only one new variable

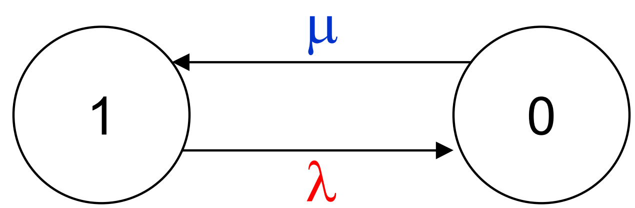
4.3 Modeling Repairable Systems

In steady state (equilibrium), transitions into/out-of each state must “balance out”

$$-\lambda p_1 + \mu p_0 = 0$$
$$p_1 + p_0 = 1$$

$$p_1 = \mu/(\lambda + \mu)$$
$$p_0 = \lambda/(\lambda + \mu)$$



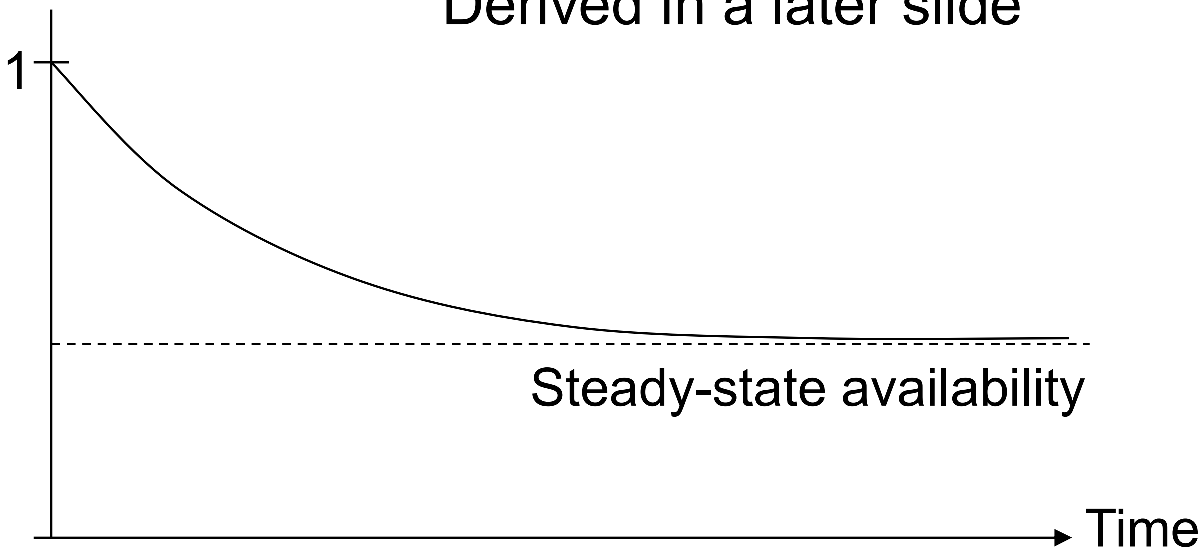


The label μ on this transition means that over time dt, repair will occur with probability μdt (constant repair rate as well as constant failure rate)

Availability as a function of time:

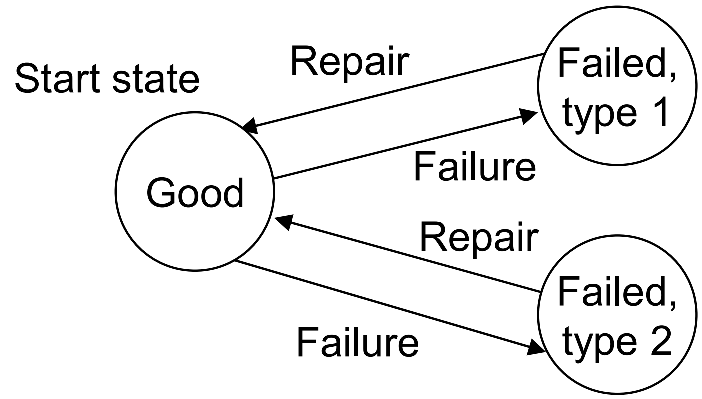
$$A(t) = p_1(t) = \mu/(\lambda + \mu) + \lambda/(\lambda + \mu)\, e^{-(\lambda+\mu)t}$$

Derived in a later slide



Multiple Failure States

In steady state (equilibrium), transitions into/out-of each state must "balance out"



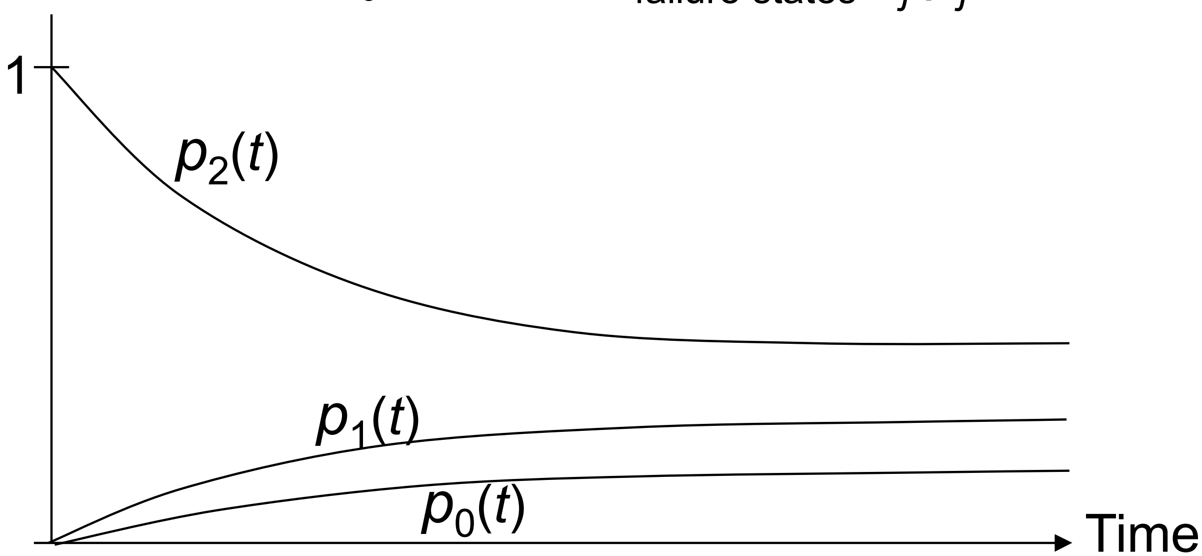
$$-\lambda p_2 + \mu p_1 + \mu p_0 = 0$$
$$-\mu p_1 + \lambda_1 p_2 = 0$$
$$p_2 + p_1 + p_0 = 1$$

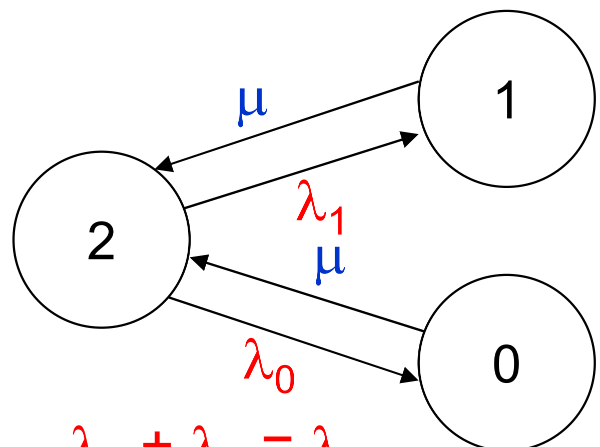
$\Rightarrow$

$$p_2 = \mu/(\lambda + \mu)$$
$$p_1 = \lambda_1/(\lambda + \mu)$$
$$p_0 = \lambda_0/(\lambda + \mu)$$

Safety evaluation:

Total risk of system is $\Sigma_{\text{failure states}}\ c_j\, p_j$



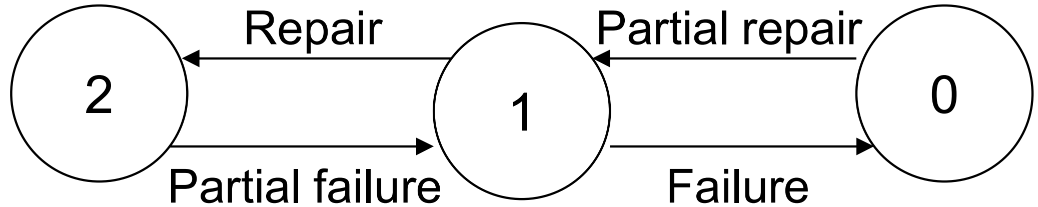


$\lambda_1 + \lambda_0 = \lambda$

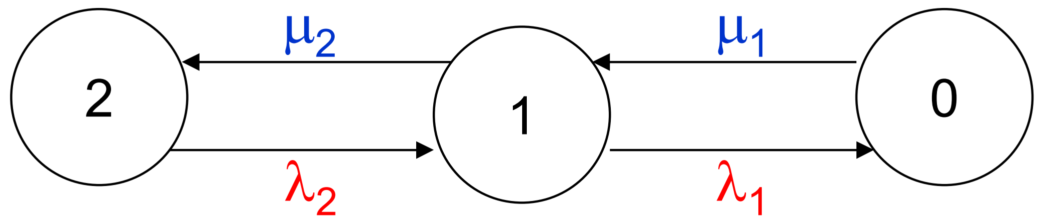
Failure state j has a cost (penalty) c_j associated with it

4.4 Modeling Fail-Soft Systems

$-\lambda_2 p_2 + \mu_2 p_1 = 0$
$\lambda_1 p_1 - \mu_1 p_0 = 0$
$p_2 + p_1 + p_0 = 1$



Let $\delta = 1/[1 + \lambda_2/\mu_2 + \lambda_1\lambda_2/(\mu_1\mu_2)]$



$p_2 = \delta$
$p_1 = \delta\lambda_2/\mu_2$
$p_0 = \delta\lambda_1\lambda_2/(\mu_1\mu_2)$

Operational state j has a benefit b_j associated with it

Performability evaluation:

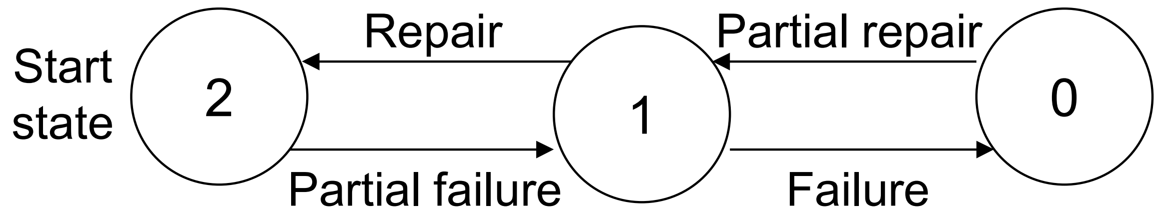
Performability $= \Sigma_{\text{operational states}}\, b_j\, p_j$

Example: $\lambda_2 = 2\lambda$, $\lambda_1 = \lambda$, $\mu_1 = \mu_2 = \mu$ (single repairperson or facility), $b_2 = 2$, $b_1 = 1$, $b_0 = 0$

$P = 2p_2 + p_1 = 2\delta + 2\delta\lambda/\mu = 2(1 + \lambda/\mu)/(1 + 2\lambda/\mu + 2\lambda^2/\mu^2)$

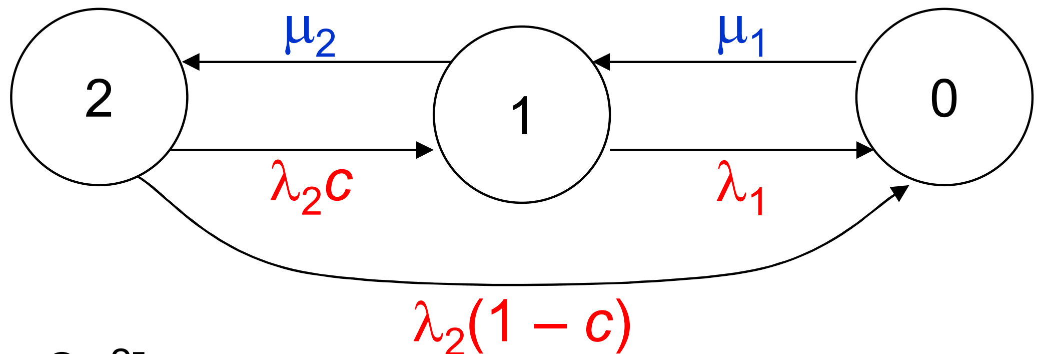
Fail-Soft System with Imperfect Coverage

$-\lambda_2 p_2 + \mu_2 p_1 = 0$
$\lambda_2(1-c)p_2 + \lambda_1 p_1 - \mu_1 p_0 = 0$
$p_2 + p_1 + p_0 = 1$



We solve this in the special case of $\lambda_2 = 2\lambda$, $\lambda_1 = \lambda$, $\mu_2 = \mu_1 = \mu$

Let $\rho = \mu / \lambda$



$p_0 = 2[(1 - c)\rho + 1] / [1 + (4 - 2c)\rho + 2\rho^2]$

$p_1 = 2\rho / [1 + (4 - 2c)\rho + 2\rho^2]$

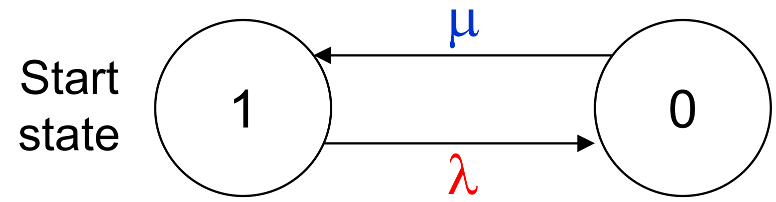
$p_2 = \rho^2 / [1 + (4 - 2c)\rho + 2\rho^2]$

If a unit's malfunction goes undetected, the system fails

We can also consider coverage for the repair direction

4.5 Solving Markov Models

$$p'_1(t) = -\lambda p_1(t) + \mu p_0(t)$$
$$p'_0(t) = -\mu p_0(t) + \lambda p_1(t)$$



To solve linear differential equations with constant coefficients:

1. Convert to algebraic equations using LaPlace transform
2. Solve the algebraic equations
3. Use inverse LaPlace transform to find original solutions

$$sP_1(s) - \overset{1}{\cancel{p_1(0)}} = -\lambda P_1(s) + \mu P_0(s)$$
$$sP_0(s) - \underset{0}{\cancel{p_0(0)}} = -\mu P_0(s) + \lambda P_1(s)$$

$$P_1(s) = (s + \mu) / [s^2 + (\lambda + \mu)s]$$
$$P_0(s) = \lambda / [s^2 + (\lambda + \mu)s]$$

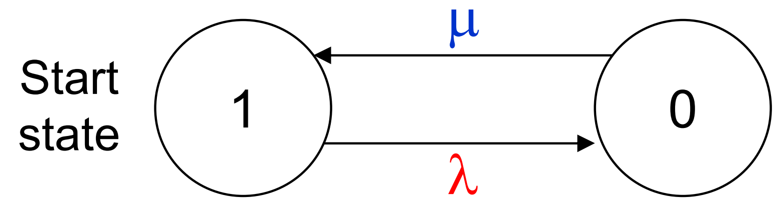
$$p_1(t) = \mu/(\lambda + \mu) + \lambda/(\lambda + \mu)\ e^{-(\lambda+\mu)t}$$
$$p_0(t) = \lambda/(\lambda + \mu) - \lambda/(\lambda + \mu)\ e^{-(\lambda+\mu)t}$$

LaPlace Transform Table

Time domain	Xform domain
k	k/s
e^{-at}	$1/(s + a)$
$t^{n-1}e^{-at}/(n-1)!$	$1/(s + a)^n$
$k\,h(t)$	$k\,H(s)$
$h(t) + g(t)$	$H(s) + G(s)$
$h'(t)$	$s\,H(s) - h(0)$

Inverse LaPlace Transform

$P_1(s) = (s + \mu) / [s^2 + (\lambda + \mu)s]$
$P_0(s) = \lambda / [s^2 + (\lambda + \mu)s]$



To find the solutions via inverse LaPlace transform:

1. Manipulate expressions into sum of terms, each of which takes one of the forms shown under $H(s)$
2. Find the inverse transform for each term

$(s + \mu) / [s^2 + (\lambda + \mu)s] =$
$1/[s + (\lambda + \mu)] + \mu/[s^2 + (\lambda + \mu)s]$

$1/[s^2 + (\lambda + \mu)s] = a/s + b/[s + (\lambda + \mu)]$

$1 = a[s + (\lambda + \mu)] + bs \quad \rightarrow \quad a + b = 0$

$a = 1/(\lambda + \mu) \qquad b = -1/(\lambda + \mu)$

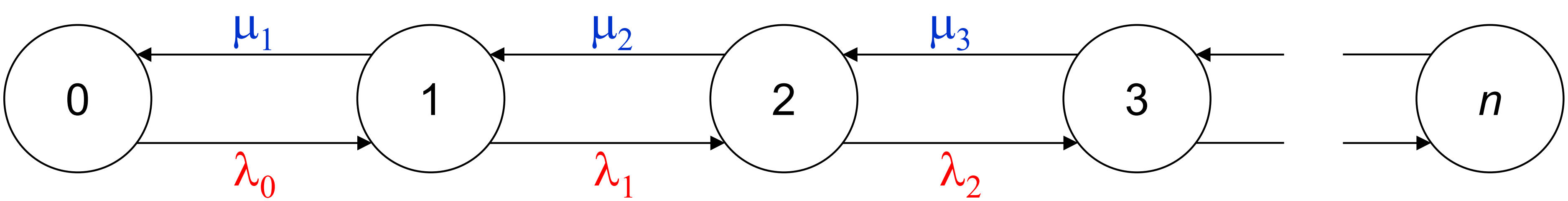
LaPlace Transform Table

Time domain	Xform domain
k	k/s
e^{-at}	$1/(s + a)$
$t^{n-1}e^{-at}/(n-1)!$	$1/(s + a)^n$
$k\,h(t)$	$k\,H(s)$
$h(t) + g(t)$	$H(s) + G(s)$
$h'(t)$	$sH(s) - h(0)$

4.6 Dependability Modeling in Practice

A birth-and-death process is a special case of Markov model with states appearing in a chain and transitions allowed only between adjacent states

Number of states is finite or infinite



This model is used in queuing theory, where the customers' arrival rate and provider's service rate determine the queue size and waiting time

Transition from state j to state $j + 1$ is an *arrival* or *birth*

Transition from state j to state $j - 1$ is a *departure* or *death*

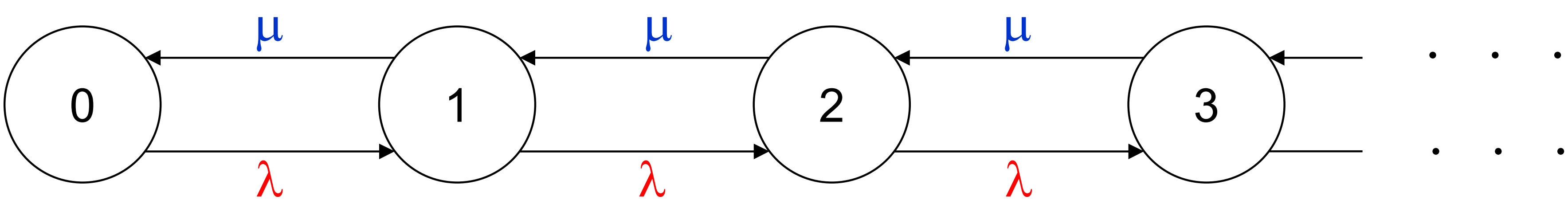
Closed-form solution for state probabilities are difficult to obtain in general

Steady-state prob.'s are easily obtained: $p_j = p_0\, \lambda_0\lambda_1 \ldots \lambda_{j-1} / (\mu_1\mu_2 \ldots \mu_j)$

Birth-and-Death Process: Special Case 1

Constant arrival (birth) and departure (death) rates, infinite chain

Ex.: Bank customers arriving at random, and a single teller serving them
(State number is the customer queue size)



Let $\rho = \lambda / \mu$ be the ratio of birth and death rates

Steady-state prob.'s for the general case: $p_j = p_0\, \lambda_0\lambda_1 \ldots \lambda_{j-1} / (\mu_1\mu_2 \ldots \mu_j)$

When $\lambda_i = \lambda$ and $\mu_i = \mu$, we have: $p_j = p_0(\lambda/\mu)^j = p_0\, \rho^j$

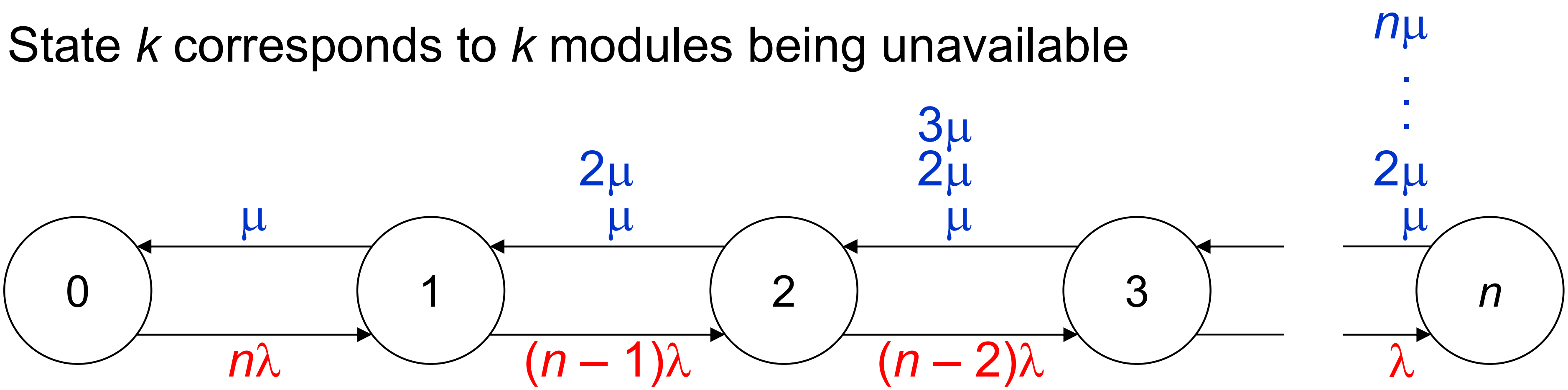
$p_0(1 + \rho + \rho^2 + \ldots) = 1$ yields $p_0 = 1 - \rho$ and $p_j = (1 - \rho)\rho^j$

Finite chain: If n is the last state, then $p_n = (1 - \rho)(\rho^n + \rho^{n+1} + \ldots) = \rho^n$

Birth-and-Death Process: Special Case 2

Gracefully degrading system with *n* identical modules

State *k* corresponds to *k* modules being unavailable



If there are *s* identical service providers (repair persons), the departure or death transition rate is capped at $s\mu$

Steady-state probabilities for the $n + 1$ states with *s* service providers (M/M/*s*/*n*/*n* queue) can be found:

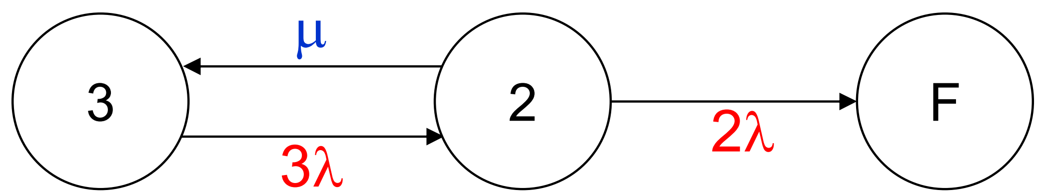
$p_j = (n - j + 1)(\lambda/\mu)\, p_{j-1}/j \quad$ for $j = 1, 2, \ldots, s$

$p_j = (n - j + 1)(\lambda/\mu)\, p_{j-1}/s \quad$ for $j = s + 1, s + 2, \ldots, n$

Equation for p_0
[Siew92], p. 347

TMR System with Repair

$-3\lambda p_3 + \mu p_2 = 0$
$-(\mu + 2\lambda)p_2 + 3\lambda p_3 = 0$
$p_3 + p_2 + p_F = 1$



Assume the voter is perfect
Upon first module malfunction, we switch to duplex operation with comparison

Steady-state analysis of no use
$p_3 = p_2 = 0,\ p_F = 1$

Mean time to failure evaluation:
See Textbook's Example 4.11 for derivation

$$\text{MTTF} = \underbrace{5/(6\lambda)}_{\text{MTTF for TMR}} + \underbrace{\mu/(6\lambda^2)}_{\text{Improvement due to repair}} = [5/(6\lambda)]\underbrace{(1 + 0.2\mu/\lambda)}_{\text{Improvement factor}}$$

MTTF Comparisons	$(\lambda = 10^{-6}/\text{hr},\ \mu = 0.1/\text{hr})$	
Nonredundant	$1/\lambda$	1 M hr
TMR	$5/(6\lambda)$	0.833 M hr
TMR with repair	$[5/(6\lambda)](1 + 0.2\mu/\lambda)$	16,668 M hr

The Dependability Modeling Process

Choose modeling approach
- Combinational
- State-space

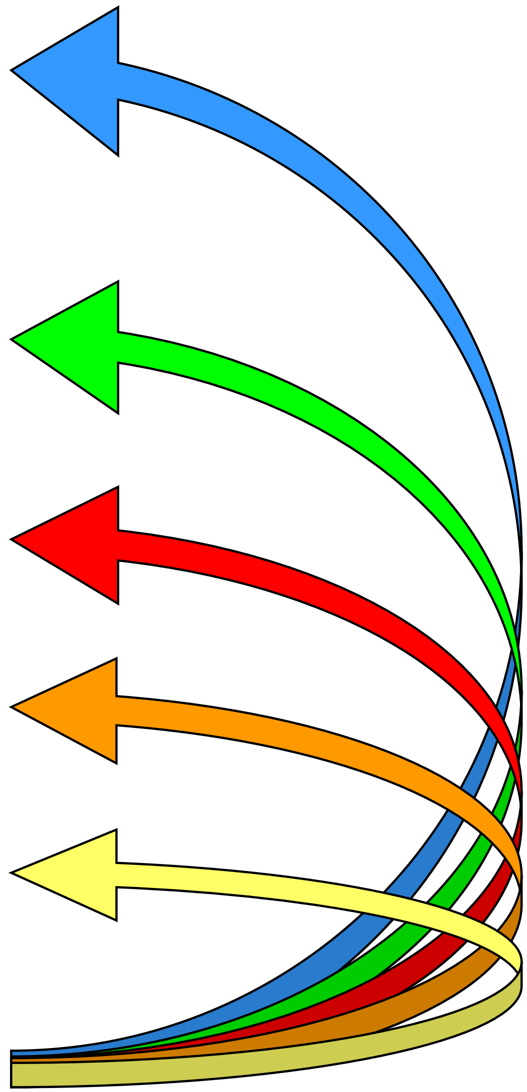
Construct model

Derive model parameters

Solve model

Interpret results

Validate model and results



Iterate until results are satisfactory

Software Aids for Reliability Modeling

PTC Windchill (formerly Relex; specializes in reliability engineering)
Fault tree analysis; Markov analysis
https://www.ptc.com/en/products/windchill

University of Virginia
Galileo (manual): http://www.cs.virginia.edu/~ftree/

Iowa State University
HIMAP: http://ecpe.ece.iastate.edu/dcnl/Tools/tools_HIMAP.htm

See Appendix D, pp. 504-518, of [Shoo02] for more programs

More limited tools from MATLAB or some MATLAB-based systems
Nanolab: *IEEE Trans. Nanotechnology*, Vol. 4, No. 4, pp. 381-394, July 2005

Virginia Tech thesis (2004): “Tools and Techniques for Evaluating Reliability Trade-offs for Nano-Architectures”

https://vtechworks.lib.vt.edu/bitstream/handle/10919/9918/bhaduri_debayan_thesis.pdf