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. Robust Parallel Processing
Part VI — Degradations: Behavioral Lapses (The Service-Level View)	Methods / Examples	21. Degradation Allowance 22. Degradation Management 23. Resilient Algorithms 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.

Edition	Released	Revised	Revised	Revised	Revised
First	Sep. 2006	Oct. 2007	Dec. 2009	Nov. 2012	Nov. 2013
		Mar. 2015	Nov. 2015	Nov. 2018	Nov. 2019
		Nov. 2020			

25 Failure Confinement

"My goal is to be a failure. If I reach my goal, I'll feel successful and if I don't reach my goal, I'll feel successful too!"

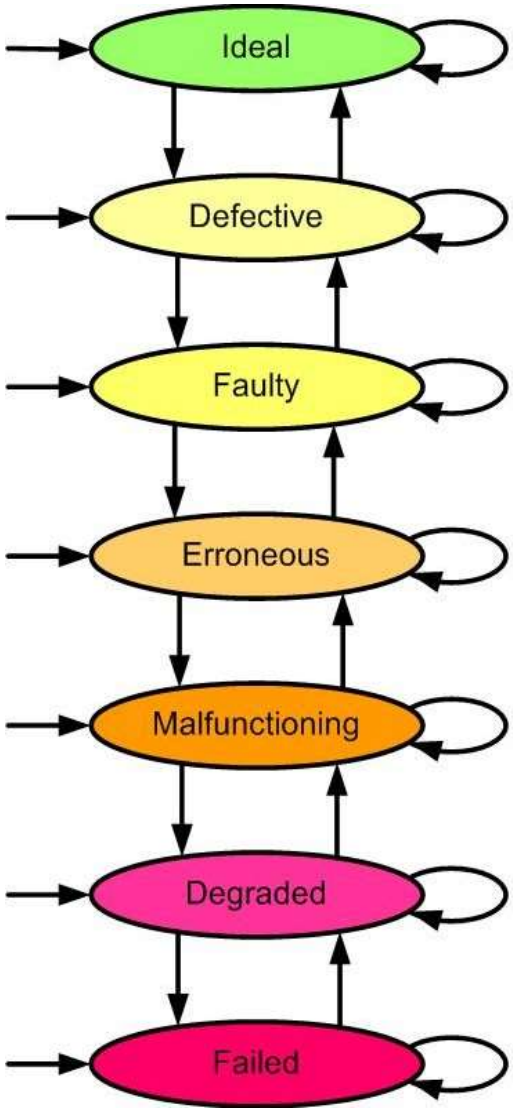
off the mark by Mark Parisi
www.offthemark.com

"Remember son, if at first you don't succeed, assemble a team of lawyers to shield all involved, convince your investors that failure is an essential component of success, create a network for spin control to trivialize any negative consequences, and try, try again."

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. Robust Parallel Processing
Part VI — Degradations: Behavioral Lapses (The Service-Level View)	Methods / Examples	21. Degradation Allowance 22. Degradation Management 23. Resilient Algorithms 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



25.1 From Failure to Disaster

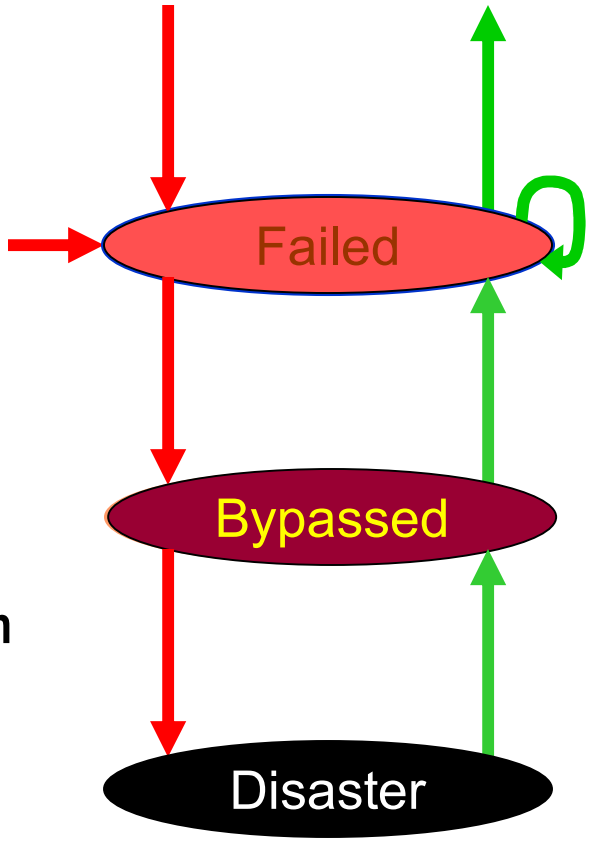
- Computers are components in larger technical or societal systems
- Failure detection and manual back-up system can prevent disaster

Used routinely in safety-critical systems:
Manual control/override in jetliners
Ground-based control for spacecraft
Manual bypass in nuclear reactors

Manual back-up and bypass systems provide a buffer between the failed state and potential disaster

Not just for safety-critical systems:
Amtrak lost ticketing capability on Friday, Nov. 30, 1996, (Thanksgiving weekend) due to a communication system failure and had no up-to-date fare information in train stations to issue tickets manually

Manual system infeasible for e-commerce sites



Space Shuttle Challenger Disaster

O-ring

Second shuttle after Columbia, first mission on April 4, 1983
Exploded 73 seconds after launch on its 10th mission (January 28, 1986)
Seven crew members killed
Temperatures dipped below freezing on launch day
Engineers were concerned about the integrity of seals on the booster

Aftermath:
Presidential commission formed to look into the incident
Cultural problems at NASA, including ineffective communication channels
Major changes enacted at NASA as a result

25.2 Failure Awareness

Importance of collecting experimental failure data

- Indicate where effort is most needed
- Help with verification of analytic models

System outage stats (%)*	Hardware	Software	Operations	Environment
Bellcore [Ali86]	26	30	44	--
Tandem [Gray87]	22	49	15	14
Northern Telecom	19	19	33	28
Japanese Commercial	36	40	11	13
Mainframe users	47	21	16	16
Overall average	**30**	**32**	**24**	**14**

*Excluding scheduled maintenance

Tandem unscheduled outages

Power	53%
Communication lines	22%
Application software	10%
File system	10%
Hardware	5%

Tandem outages due to hardware

Disk storage	49%
Communications	24%
Processors	18%
Wiring	9%
Spare units	1%

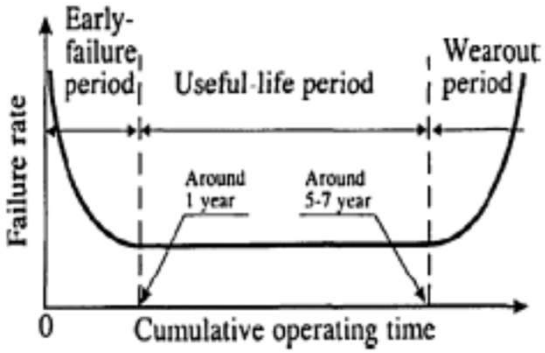
System Failure Data Repositories

Usenix Computer Failure Data Repository
http://usenix.org/cfdr
"The computer failure data repository (CDFR) aims at accelerating research on system reliability by filling the nearly empty collection of public data with detailed failure data from a variety of large production systems. . . .
You first need to register for full access to CFDR data/tools to obtain a user id/password. You can then go to the data overview page."

LANL data, collected 1996-2005: SMPs, Clusters, NUMAs
http://institute.lanl.gov/data/fdata/
From the site's FAQs: "A failure record contains the time when the failure started (start time), the time when it was resolved (end time), the system and node affected, the type of workload running on the node and the root cause."

Memory and Storage Failure Data

Storage failure data: “Disk Failures in the Real World: What does an MTTF of 1,000,000 hours mean to you?” (Schroeder & Gibson, CMU)
http://www.cs.cmu.edu/~bianca/fast07.pdf
From the abstract: “. . . field replacement is a fairly different process than one might predict based on datasheet MTTF.”



OBSERVATION 1. *Variance between datasheet MTTF and disk replacement rates in the field was larger than we expected. The weighted average ARR was 3.4 times larger than 0.88%, corresponding to a datasheet MTTF of 1,000,000 Hours.* [Schr07]

Rochester Memory Hardware Error Research Project
http://www.cs.rochester.edu/research/os/memerror/
“Our research focuses on the characteristics of memory hardware errors and their implications on software systems.” Both soft errors and hard errors are considered.

Software Failure Data

Promise Software Engineering Repository
http://promise.site.uottawa.ca/SERepository/
"Here you will find a collection of publicly available datasets and tools to serve researchers [who build] predictive software models (PSMs) and software engineering community at large."

Software Forensics Centre failure data, Middlesex University
http://www.cs.mdx.ac.uk/research/SFC/
From the website: "The repository of failures is the largest of its kind in the world and has specific details of well over 300 projects (with links to another 2,000 cases)."

[Note: As of November 27, 2013, the large repository seems to have disappeared from the site.]

Failure Data Used to Validate or Tune Models

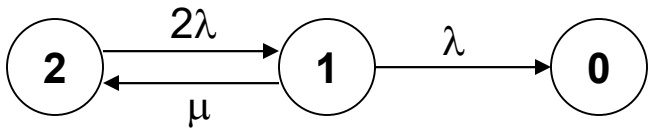
- Indicate accuracy of model predictions (compare multiple models?)
- Help in fine-tuning of models to better match the observed behavior

Example: Two disks, each with MTTF = 50K hr (5.7 yr), MTTR = 5 hr

Disk pair failure rate $\approx 2\lambda^2/\mu$

Disk pair MTTF $\approx \mu/(2\lambda^2)$

$= 2.5 \times 10^8$ hr = 28,500 yr



In 48,000 years of observation (2 years × 6000 systems × 4 disk pairs), 35 double disk failures were reported $\Rightarrow$ MTTF $\approx$ 1400 yr

Problems with experimental failure data:

- Difficult to collect, while ensuring uniform operating conditions
- Logs may not be complete or accurate (the embarrassment factor)
- Assigning a cause to each failure not an easy task
- Even after collection, vendors may not be willing to share data
- Impossible to do for one-of-a-kind or very limited systems

25.3 Failure and Risk Assessment

- Minimum requirement: accurate estimation of failure probability
- Putting in place procedures for dealing with failures when they occur

Failure probability = Unreliability
Reliability models are by nature pessimistic (provide lower bounds)
However, we do not want them to be too pessimistic

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

Frequency may be equated with unreliability or failure probability
Magnitude is estimated via economic analysis (next slide)

Failure handling is often the most neglected part of the process
An important beginning: clean, unambiguous messages to operator/user
Listing the options and urgency of various actions is a good idea
Two way system-user communication (adding user feedback) helpful
Quality of failure handling affects the “Magnitude” term in risk equation

How Much Is Your Life Worth to You?

Thought experiment:

You are told that you have a 1/10,000 chance of dying today

How much would you be willing to pay to buy out this risk, assuming that you're not limited by current assets (can use future earnings too)?

If your answer is $1000, then your life is worth $10M to you

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

Can visualize the risk by imagining that 10,000 people in a stadium are told that one will be killed unless they collectively pay a certain sum

Consciously made tradeoffs in the face of well-understood risks (salary demanded for certain types of work, willingness to buy smoke detector) has been used to quantify the worth of a "statistical human life"

Very Small Probabilities: The Human Factor

Interpretation of data, understanding of probabilities, acceptance of risk

Risk of death / person / year

Influenza	1/5K
Struck by auto	1/20K
Tornado (US MW)	1/455K
Earthquake (CA)	1/588K
Nuclear power plant	1/10M
Meteorite	1/100B

Factors that increase risk of death by $1/10^6$ (deemed acceptable risk)

Smoking 1.4 cigarettes
Drinking 0.5 liter of wine
Biking 10 miles
Driving 300 miles
Flying 1000 miles
Taking a chest X-ray
Eating 100 steaks

US causes of death / 10^6 persons

Auto accident	210
Work accident	150
Homicide	93
Fall	74
Drowning	37
Fire	30
Poisoning	17
Civil aviation	0.8
Tornado	0.4
Bite / sting	0.2

Risk underestimation factors:
Familiarity, being part of our job, remoteness in time or space

Risk overestimation factors:
Scale (1000s killed), proximity

25.4 Limiting the Damage

Prompt failure detection is a prerequisite to failure confinement

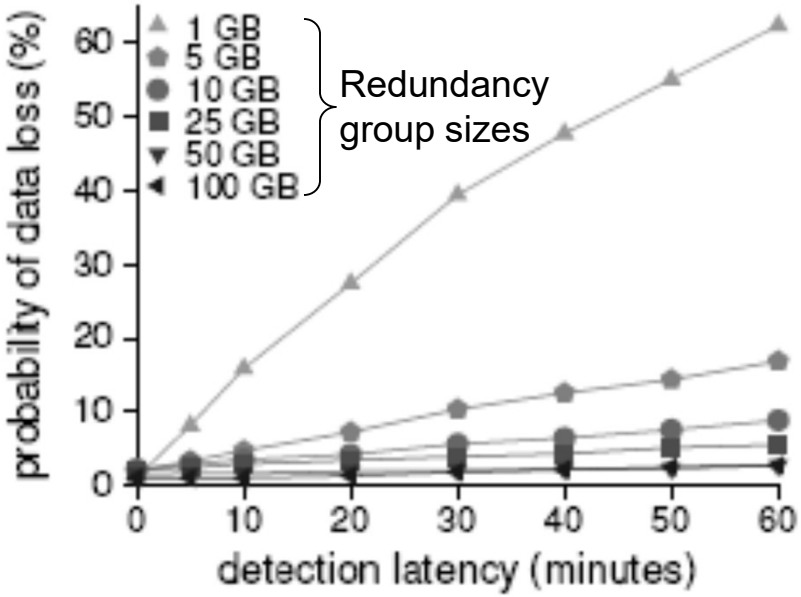
In many cases dealing with mechanical elements, such as wing flaps, reaction time of 10s/100s of milliseconds is adequate (reason: inertia)

In some ways, catastrophic failures that are readily identified may be better than subtle failures that escape detection

Example: For redundant disks with two-way mirroring, detection latency was found to have a significant effect on the probability of data loss

See: http://www.hpdc.org/2004/papers/34.pdf

Failure detection latency can be made negative via "failure prediction" (e.g., in a storage server, increased error rate signals impending failure)



25.5 Failure Avoidance Strategies

Advice for dependable system design and development

– Limit novelty [stick with proven methods and technologies]
– Adopt sweeping simplifications
– Get something simple working soon
– Iteratively add capability
– Give incentives for reporting errors
– Descope [reduce goals/specs] early
– Give control to (and keep it in) a small design team

25.6 Ethical Considerations

Risks must be evaluated thoroughly and truthfully

IEEE Code of Ethics: As IEEE members, we agree to

1. Accept responsibility in making decisions consistent with the safety, health and welfare of the public, and to disclose promptly factors that might endanger the public or the environment
6. Maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations;
7. Seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others

ACM Code of Ethics: Computing professionals must

Minimize malfunctions by following generally accepted standards for system design and testing

Give comprehensive and thorough evaluations of computer systems and their impacts, including analysis of possible risks

National Society of Professional Engineers

Comprehensive code of engineering ethics

I. Fundamental Canons

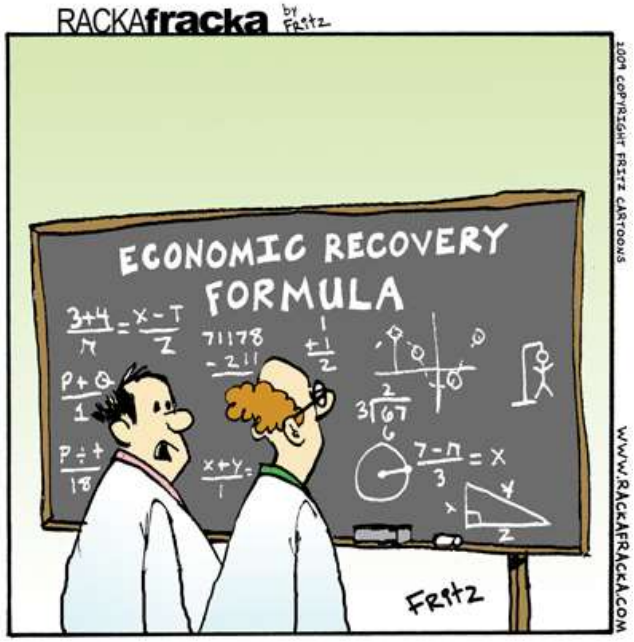
Engineers, in the fulfillment of their professional duties, shall:

1. Hold paramount the safety, health, and welfare of the public
2. Perform services only in areas of their competence
3. Issue public statements only in an objective and truthful manner
4. Act for each employer or client as faithful agents or trustees
5. Avoid deceptive acts
6. Conduct themselves honorably, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession

II. Rules of Practice: e.g., assignments to accept, whistle-blowing

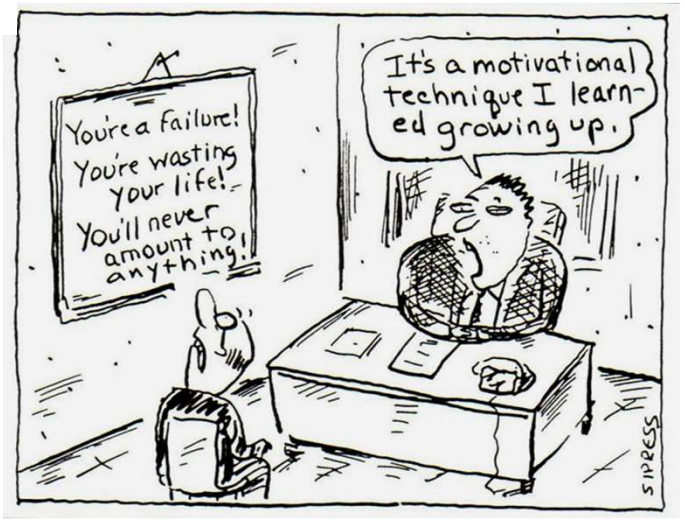
III. Professional Obligations: e.g., ack errors, be open to suggestions

26 Failure Recovery



YOU FORGOT TO CARRY THE "1."

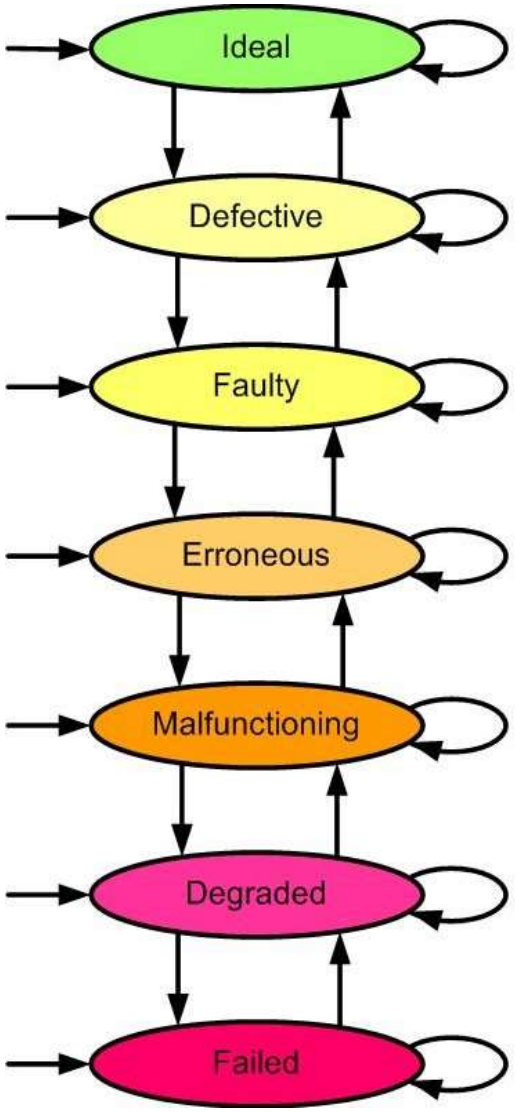
"MEMO: It has come to my attention that every time we solve one problem, we create two more. From now on, all problem solving is forbidden."



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. Robust Parallel Processing
Part VI — Degradations: Behavioral Lapses (The Service-Level View)	Methods / Examples	21. Degradation Allowance 22. Degradation Management 23. Resilient Algorithms 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



26.1 Planning for Recovery

Be prepared for recovery from both mild failures and disasters

- Have documented plans
- Have trained personnel

Recovery from Failures

The recovery block scheme (originally developed for software)

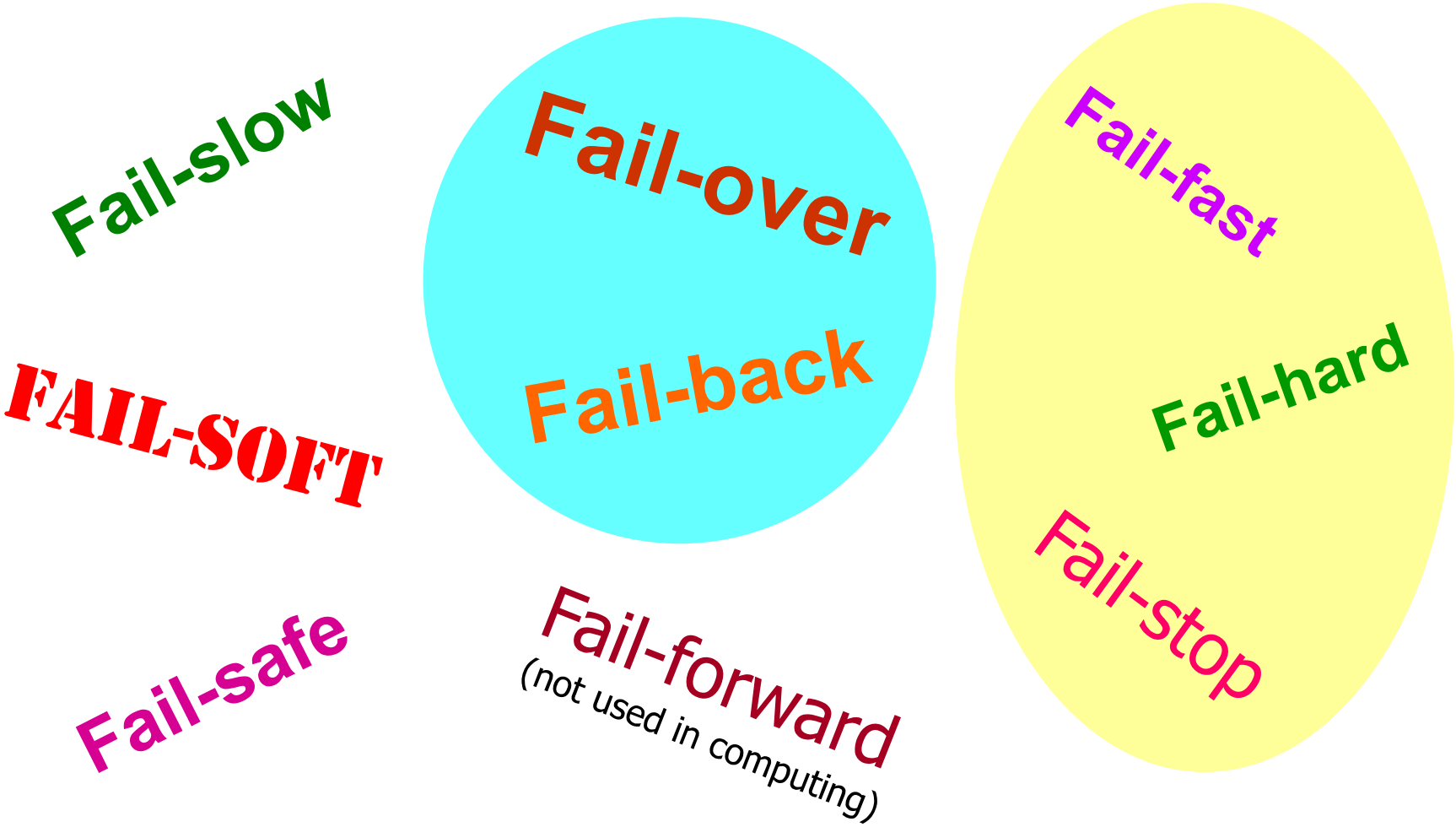
ensure	*acceptance test*	
by	*primary module*	e.g., sorted list
else by	*first alternate*	e.g., quicksort
	.	e.g., bubblesort
	.	.
	.	.
		.
else by	*last alternate*	e.g., insertion sort
else fail		

Computer system with manual backup may be viewed as a one-alternate recovery block scheme, with human judgment constituting the acceptance test

Before resorting to an alternate (hardware/software) module, one may reuse the primary module once or a few times

This scheme is known as “retry” or time redundancy and is particularly effective for dealing with transient or soft failures

26.2 Types of Recovery



Fail-Hard and Fail-Fast Systems

Subtle failures that are difficult to detect can be worse than overt ones that are immediately noticed

Design system so that their failures are difficult to miss (fail-hard system)

Corollary: Unneeded system complexity is to be avoided at all cost

Extended failure detection latency is undesirable because the occurrence of subsequent unrelated failures may overwhelm the system's defenses

Design system to support fast failure detection (fail-fast system)

Concurrent fault / error / malfunction detection better than periodic testing

Fail-Stop and Fail-Over Strategies

Fail-stop systems: Systems designed and built in such a way that they cease to respond or take any action upon internal malfunction detection

Such systems do not confuse the users or other parts of a distributed system by behaving randomly or, worse, maliciously upon failure

A subclass of fail-safe systems (here, stopping is deemed a safe output)

Fail-over: Upon failure detection, often of the fail-stop kind, requests and other tasks are redirected to a backup system

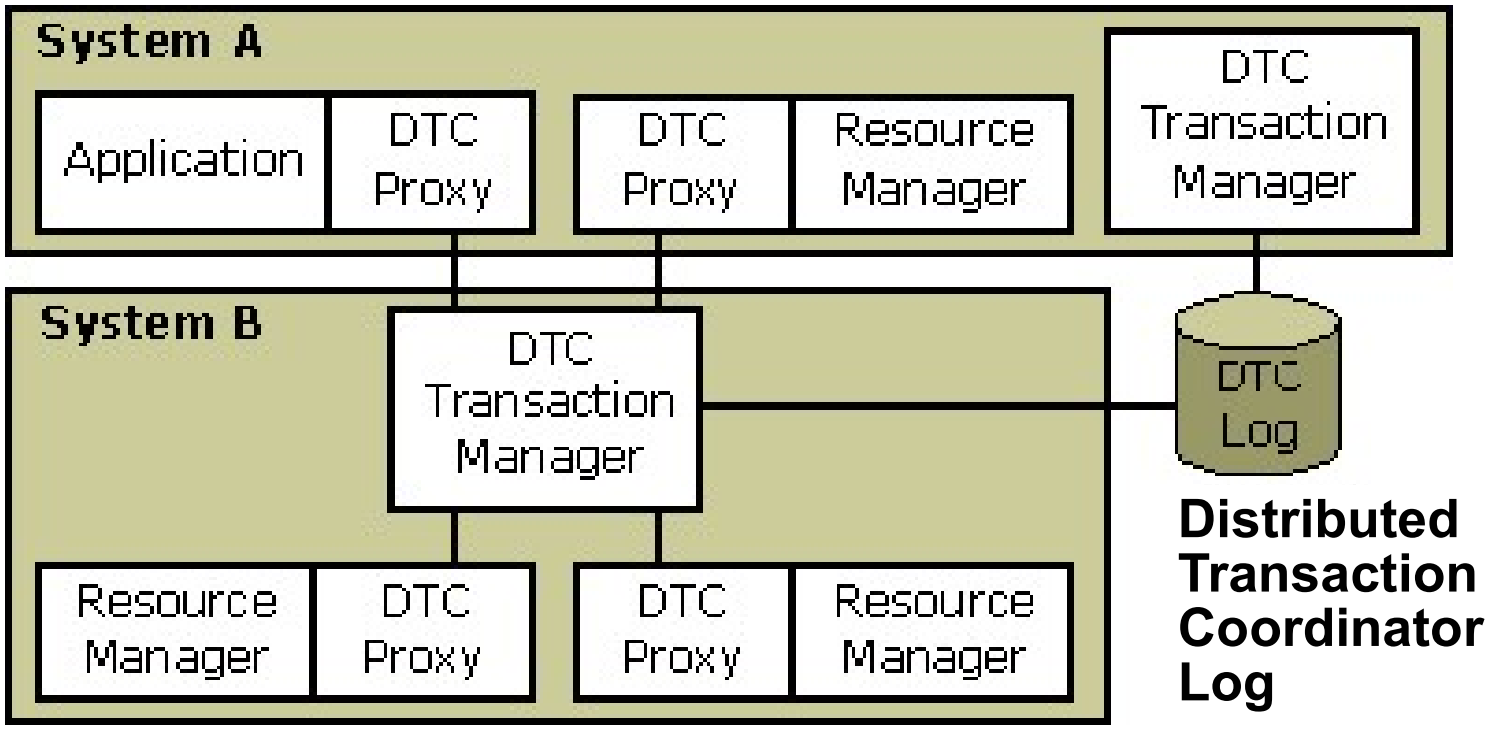
Example – Fail-over on long-running connections for streaming media: all that is needed is to know the file being streamed, and current location within the file, to successfully switch over to a different server

Fail-over software is available for Web servers as part of firewalls for most popular operating systems

It monitors resources and directs requests to a functioning server

Fail-over features of Windows XP: http://msdn2.microsoft.com/en-us/library/ms686091.aspx

Fail-Over Features of Windows Servers



If System B fails, the DTC transaction manager on System A takes over. It reads the DTC log file on the shared disk, performs recovery, and then serves as the transaction manager for the entire cluster.

http://msdn2.microsoft.com/en-us/library/ms686091.aspx

26.3 Interfaces and the Human Link

Believability and helpfulness of failure warnings

> "No warning system will function effectively if its messages, however logically arrived at, are ignored, disbelieved, or lead to inappropriate actions." Foster, H. D., "Disaster Warning Systems," 1987

Unbelievable failure warnings:
Failure event after numerous false alarms
Real failure occurring in the proximity of a scheduled test run
Users or operators inadequately trained (May 1960 Tsunami in Hilo, Hawaii, killed 61, despite 10-hour advance warning via sirens)

Unhelpful failure warnings:
Autos – "Check engine"
Computer systems – "Fatal error"

Human Factors in Automated Systems

"A few motorists have even driven off a cliff or into oncoming traffic after following [GPS] directions explicitly." [Gree09]

The curse of highly successful automation efforts:
Human operators tend to "switch off" mentally,
so they are not prepared when something unexpected happens
(e.g., commuter train operators text-messaging or dozing off)

26.4 Backup Systems and Processes

Backup (*n*): Copy of data made for use after computer system failure or other data loss events [(*v*) to back up, backing up]

Question: What method do you use to back up your data files?

Use removable storage: external hard drive, flash drive, CD/DVD
E-mail file copy to yourself (not usable for very large files)
Run a backup utility periodically (full or incremental backup)
Subscribe to an on-line backup service (plain or encrypted)
Do not overwrite files, create new versions

Businesses also need backups for programs and computer resources

26.5 Blame Assessment and Liability

Computer forensics is used to:

Analyze computers belonging to defendants or litigants in legal cases

Gather evidence against those suspected of wrongdoing

Learn about a system to debug, optimize, or reverse-engineer it

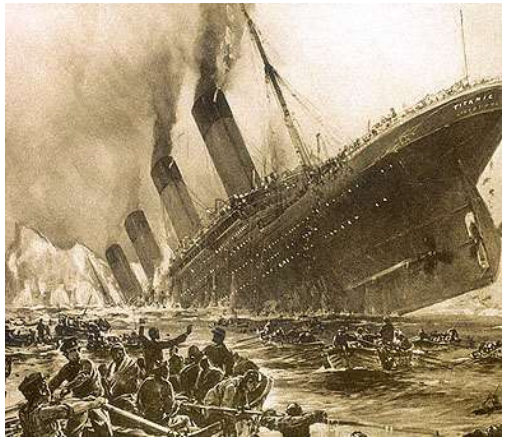
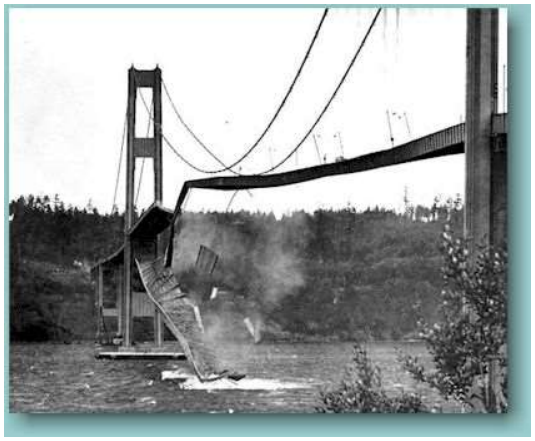
Analyze compromised computer systems to determine intrusion method

Analyze failed computer systems to determine cause of failure

Recover data after computer failures

There are several journals on computer forensics, digital investigations, and e-discovery

26.6 Learning 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

Some Notorious Failed Systems

Source: http://web.mit.edu/6.033/lec/s25.pdf

Automated reservations, ticketing, flight scheduling, fuel delivery, kitchens, and general administration, United Airlines + Univac, started 1966, target 1968, scrapped 1970, $50M

Hotel reservations linked with airline and car rental, Hilton + Marriott + Budget + American Airlines, started 1988, scrapped 1992, $125M

IBM workplace OS for PPC (Mach 3.0 + binary compatibility with AIX + DOS, Mac OS, OS/400 + new clock mgmt + new RPC + new I/O + new CPU), started 1991, scrapped 1996, $2B

US FAA's Advanced Automation System (to replace a 1972 system), started 1982, scrapped 1994, $6B

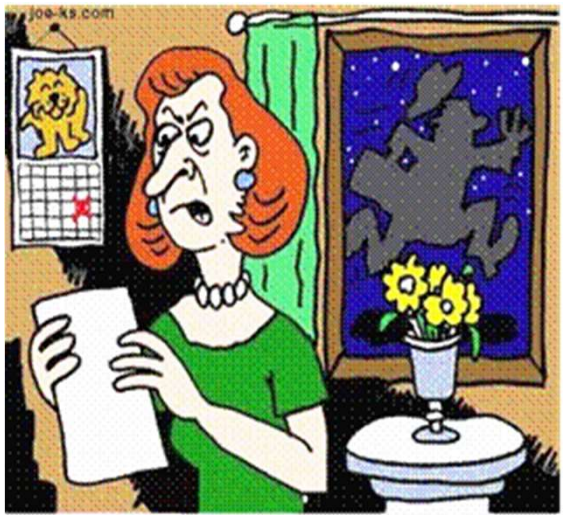
London Ambulance Dispatch Service, started 1991, scrapped 1992, 20 lives lost in 2 days, 2.5M

More Systems that Were Doomed to Failure

Source: http://web.mit.edu/6.033/lec/s25.pdf

- Portland, Oregon, Water Bureau, $30M, 2002
- Washington D.C., Payroll system, $34M, 2002
- Southwick air traffic control system $1.6B, 2002
- Sobey's grocery inventory, Canada, $50M, 2002
- King County financial mgmt system, $38M, 2000
- Australian submarine control system, 100M, 1999
- California lottery system, $52M
- Hamburg police computer system, 70M, 1998
- Kuala Lumpur total airport management system, $200M, 1998
- UK Dept. of Employment tracking, $72M, 1994
- Bank of America Masternet accounting system, $83M, 1988
- FBI virtual case, 2004
- FBI Sentinel case management software, 2006

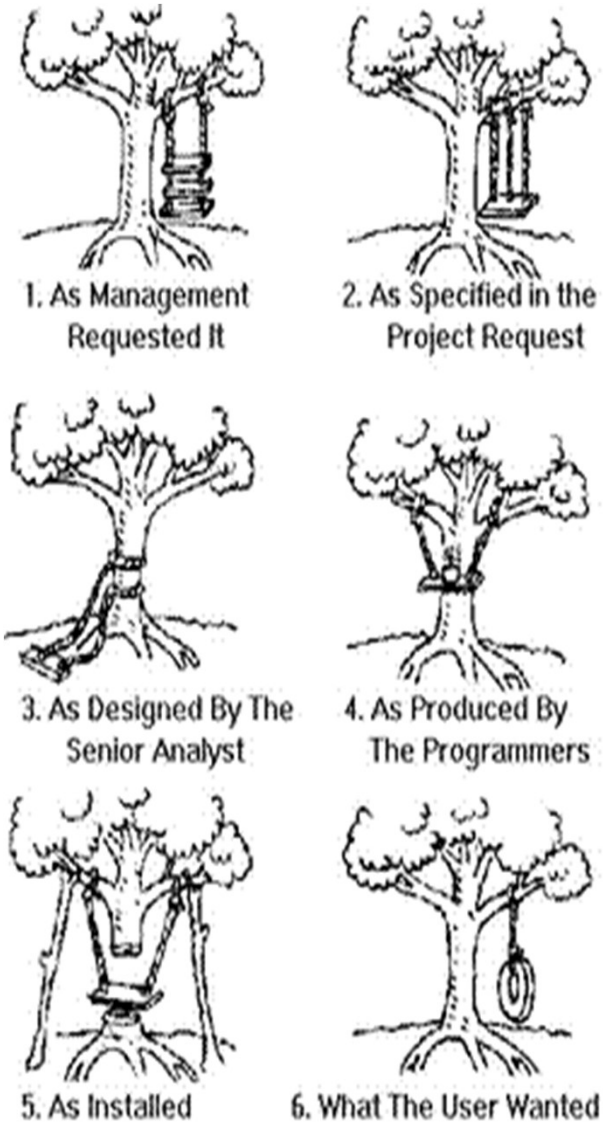
27 Agreement and Adjudication



Abe, I just noticed – our marriage license has an expiration date! And it's today! Do you know anything about this, Abe? . . . Abe? . . . Abe?

"Next case: the Internet economy versus millions of investors who should have known better."

Software Specification

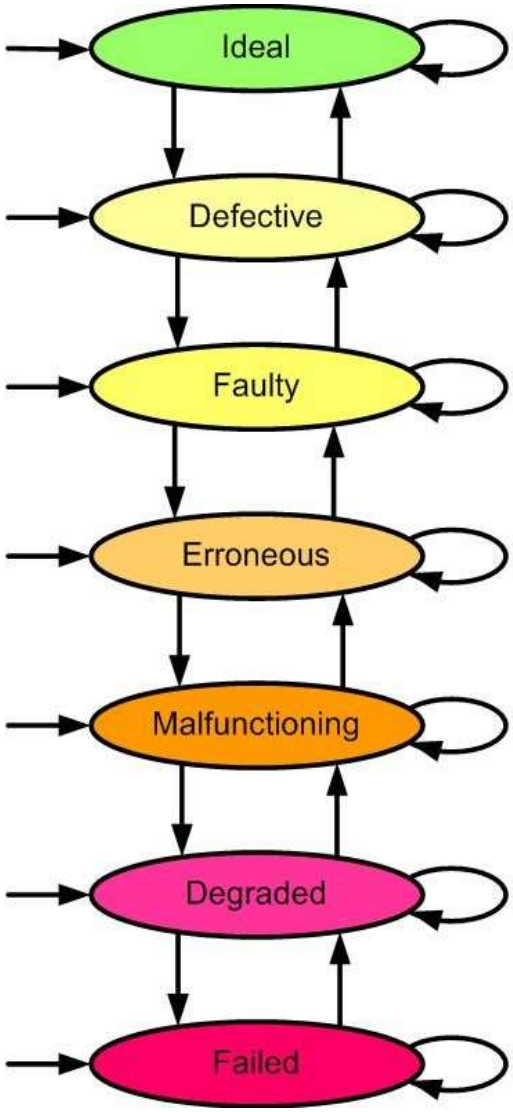


"It was the only mission statement everyone could agree on."

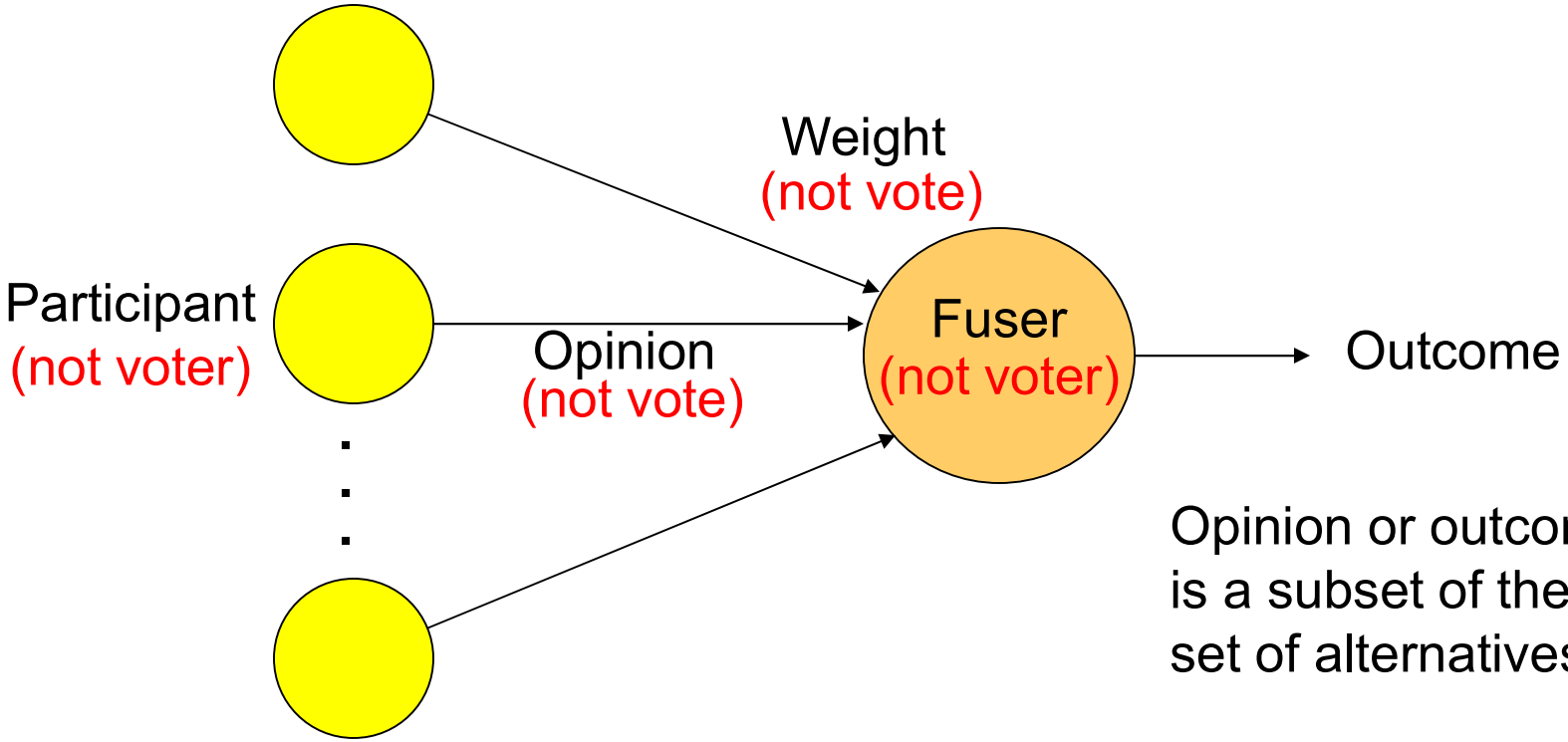
STRUCTURE AT A GLANCE

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. Robust Parallel Processing
Part VI — Degradations: Behavioral Lapses (The Service-Level View)	Methods / Examples	21. Degradation Allowance 22. Degradation Management 23. Resilient Algorithms 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



27.1 Voting and Data Fusion



Need for unambiguous terminology

Social choice research
Data fusion research
Dependable computing research

Introduction to Voting

Voting schemes and associated terminology in dependable computing were originally derived from concepts in sociopolitical elections

With inputs drawn from a small set of integers, the similarity between the two domains is strong

Example: Radar image analysis used to classify approaching aircraft type as civilian (0), fighter (1), bomber (2).

If three independent units arrive at the conclusions ⟨1, 1, 2⟩, then the presence of a fighter plane may be assumed

Option or candidate 1 “wins” or garners a majority

With a large or infinite input domain, voting takes on a new meaning

Example: There is no strict majority when distance of an approaching aircraft, in km, is indicated as ⟨12.5, 12.6, 14.0⟩, even though the good agreement between 12.5 and 12.6 could lead to a 12.55 km estimate

Role of Voting Units in Dependable Computing

John von Neumann, 1956: “Probabilistic Logic and Synthesis of Reliable Organisms from Unreliable Components”

Hardware voting for multichannel computation

High performance (pipelined)

TMR, NMR

Software voting for multiversion programming

Imprecise results (approximate voting)

NVP, SIFT

Consistency of replicated data

Weighted voting and weight assignment

Quorum, Consensus

Voting schemes in these three contexts share some properties

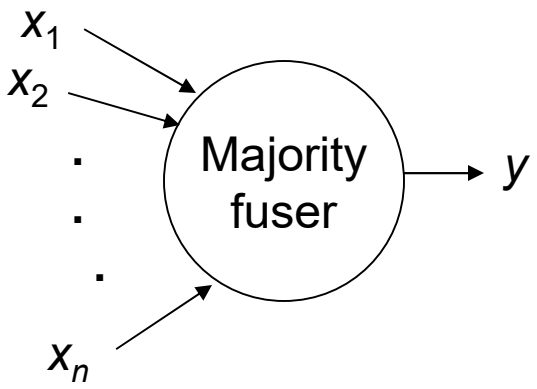
General schemes have been devised to cover all these instances

What We Already Know About Voting

Majority voting: Select the value that appears on at least $\lfloor n/2 \rfloor + 1$ of the n inputs

Number n of inputs is usually odd, but does not have to be

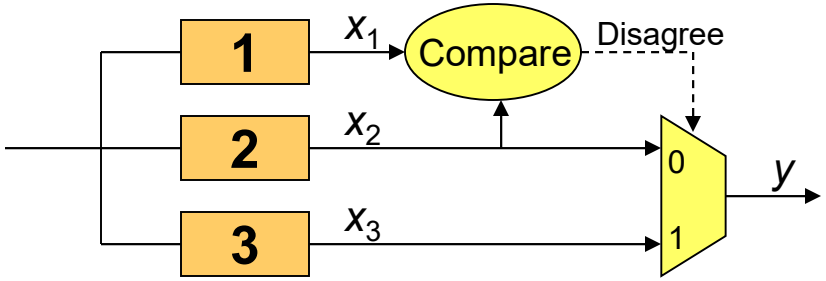
Example: vote(1, 2, 3, 2, 2) = 2



Majority fusers can be realized by means of comparators and muxes

This design assumes that in the case of 3-way disagreement any one of the inputs can be chosen

Can add logic to this fuser so that it signals three-way disagreement



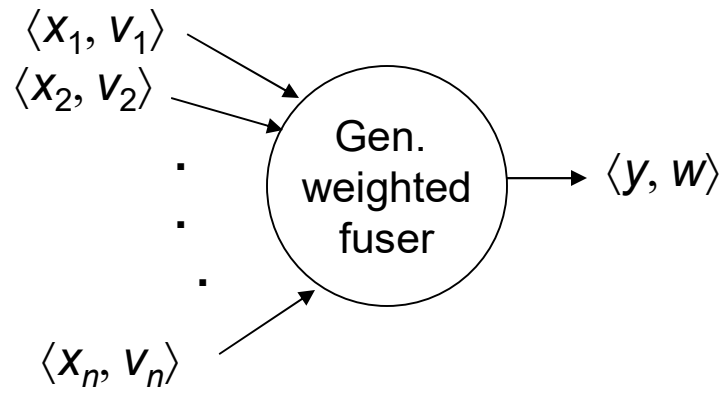
27.2 Weighted Voting

Virtually all voting schemes of practical interest can be formulated in terms of the generalized weighted voting model, as follows:

Given n input data objects $x_1, x_2, \ldots, x_n$ and associated nonnegative real weights $v_1, v_2, \ldots, v_n$, with $\Sigma v_i = V$, compute output y and its weight w such that y is "supported by" a set of input objects with weights totaling w, where w satisfies a condition associated with the voting subscheme

Possible voting subschemes:

Unanimity	$w = V$
Majority	$w > V/2$
Supermajority	$w \geq 2V/3$
Byzantine	$w > 2V/3$
Plurality	(w for y) $\geq$ (w for any $z \neq y$)
Threshold	$w >$ a preset lower bound

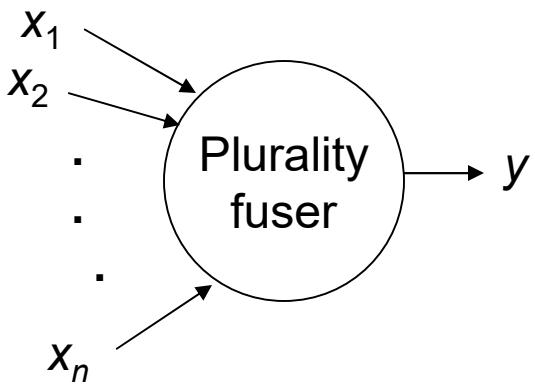


There is More to Voting than Simple Majority

Plurality voting: Select the value that appears on the largest number of inputs

Example: vote(1, 3, 2, 3, 4) = 3

What should we take as the result of vote(1.00, 3.00, 0.99, 3.00, 1.01)?



It would be reasonable to take 1.00 as the result, because 3 participants agree or approximately agree with 1.00, while only 2 agree with 3.00

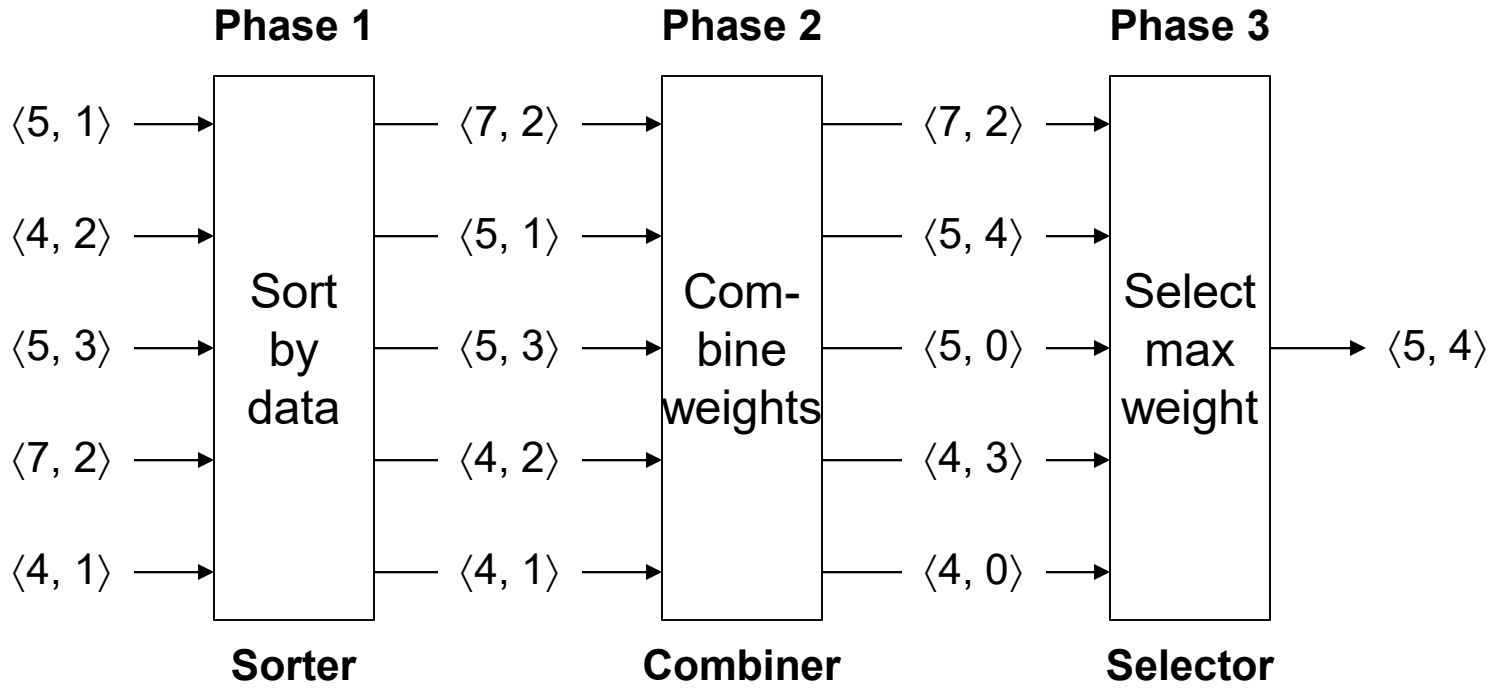
Will discuss approximate voting and a number of other sophisticated voting schemes later

Median voting: one way to deal with approximate values
median(1.00, 3.00, 0.99, 3.00, 1.01) = 1.01

Median value is equal to the majority value when we have majority

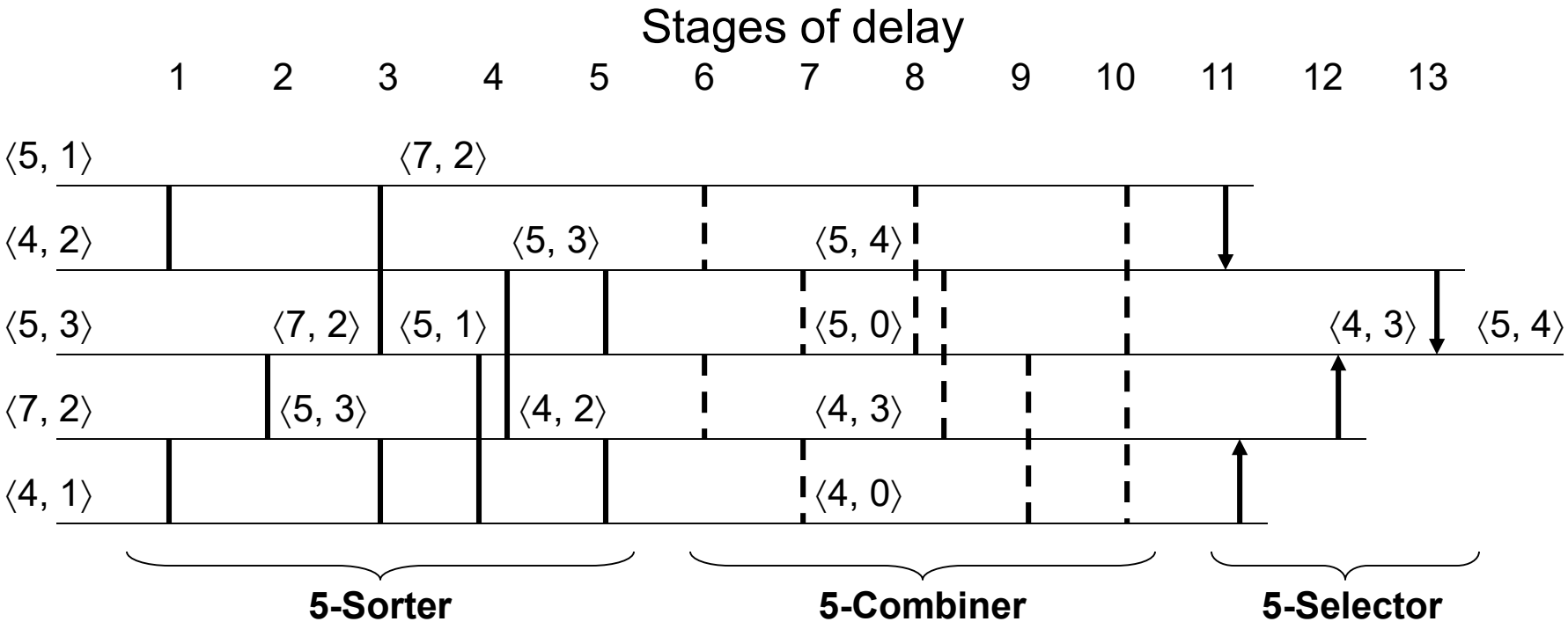
Implementing Weighted Plurality Voting Units

Inputs: Data-weight pairs
Output: Data with maximal support and its associated tally



Source: B. Parhami, *IEEE Trans. Reliability*, Vol. 40, No. 3, pp. 380-394, August 1991

Details of a Sorting-Based Plurality Voting Unit

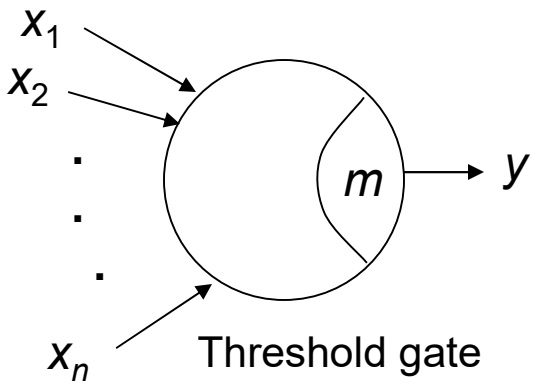


The first two phases (sorting and combining) can be merged, producing a 2-phase design – fewer, more complex cells (lead to tradeoff)

Source: B. Parhami, *IEEE Trans. Reliability*, Vol. 40, No. 3, pp. 380-394, August 1991

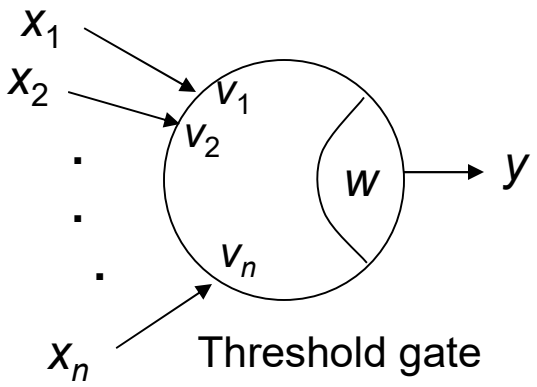
Threshold Voting and Its Generalizations

Simple threshold (m-out-of-n) voting:
Output is 1 if at least m of the n inputs are 1s



Majority voting is a special case of threshold voting: $(\lfloor n/2 \rfloor + 1)$-out-of-$n$ voting

Weighted threshold (w-out-of-Σv_i) voting:
Output is 1 if $\Sigma v_i x_i$ is w or more



Agreement or quorum sets
$\{x_1, x_2\}, \{x_2, x_3\}, \{x_3, x_1\}$ – same as 2-out-of-3
$\{x_1, x_2\}, \{x_1, x_3, x_4\}, \{x_2, x_3, x_4\}$

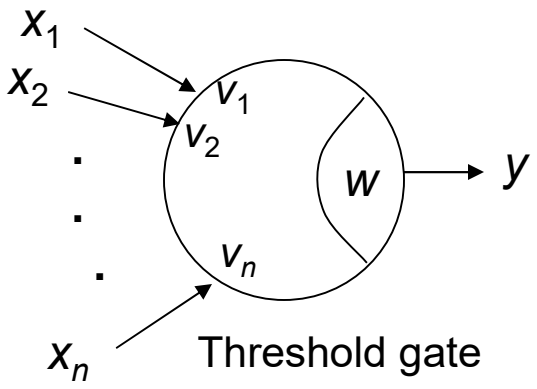
The 2nd example above is weighted voting with
$v_1 = v_2 = 2$, $v_3 = v_4 = 1$, and threshold $w = 4$

Agreement sets are more general than weighted voting in the sense of some agreement sets not being realizable as weighted voting

Usefulness of Weighted Threshold Voting

Unequal weights allow us to take different levels of input reliabilities into account

Zero weights can be used to disable or purge some of the inputs (combined switching-voting)



Maximum-likelihood voting
prob{x_1 correct} = 0.9
prob{x_2 correct} = 0.9
prob{x_3 correct} = 0.8
prob{x_4 correct} = 0.7
prob{x_5 correct} = 0.6

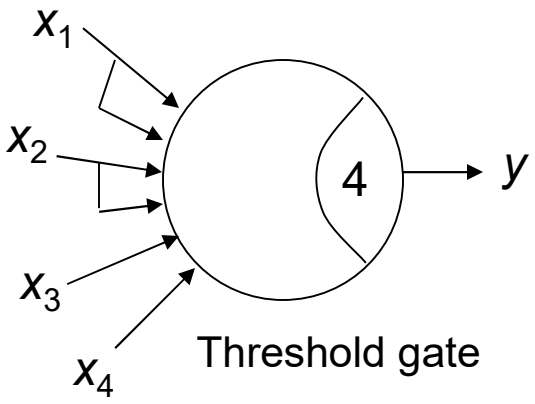
Max-likelihood voting can be implemented in hardware via table lookup or approximated by weighted threshold voting (otherwise, we need software)

Assume $x_1 = x_3 = a$, $x_2 = x_4 = x_5 = b$

prob{a correct} = 0.9 × 0.1 × 0.8 × 0.3 × 0.4 = 0.00864
prob{b correct} = 0.1 × 0.9 × 0.2 × 0.7 × 0.6 = 0.00756

Implementing Weighted Threshold Voting Units

Ex.: Implement a 4-input threshold voting unit with $v_1 = v_2 = 2$, $v_3 = v_4 = 1$, and threshold $w = 4$



Strategy 1: If weights are small integers, fan-out each input an appropriate number of times and use a simple threshold voting unit

Strategy 2: Use table lookup based on comparison results

$x_1=x_2$	$x_1=x_3$	$x_2=x_3$	$x_3=x_4$	Result
1	x	x	x	x_1
0	0	0	1	Error
0	1	0	1	x_1
0	0	1	1	x_2
0	x	x	0	Error

Is this table complete? Why, or why not?

Strategy 3: Convert the problem to agreement sets (discussed next)

27.3 Voting with Agreement Sets

3-input majority voting has the agreement sets $\{x_1, x_2\}$, $\{x_2, x_3\}$, $\{x_3, x_1\}$

If participants in any set agree, then their opinion is taken as the output (clearly, the agreement sets cannot include $\{x_1, x_2\}$ and $\{x_3, x_4\}$, say, if we are seeking a unique output)

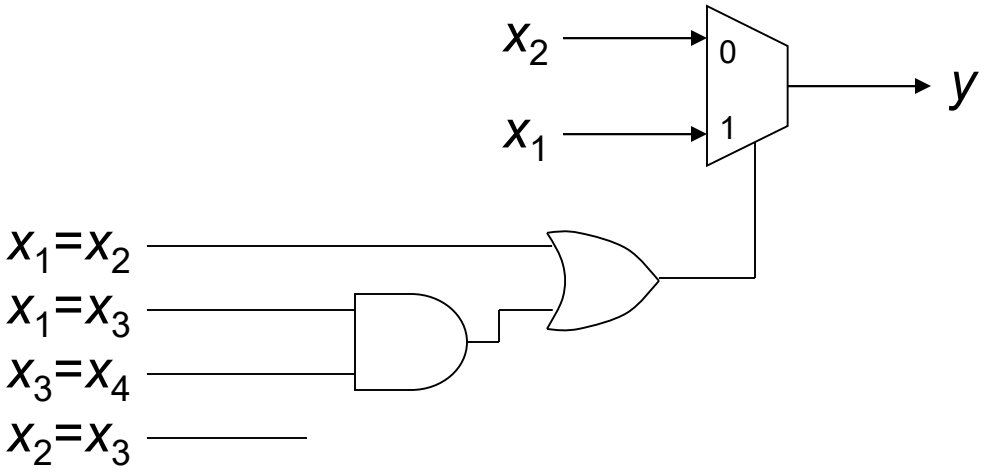
Implementing Agreement-Set Voting Units

Example: Implement a voting unit corresponding to the agreement sets $\{x_1, x_2\}$, $\{x_1, x_3, x_4\}$, $\{x_2, x_3, x_4\}$

Strategy 1: Implement as weighted threshold voting unit, if possible

Strategy 2: Implement directly

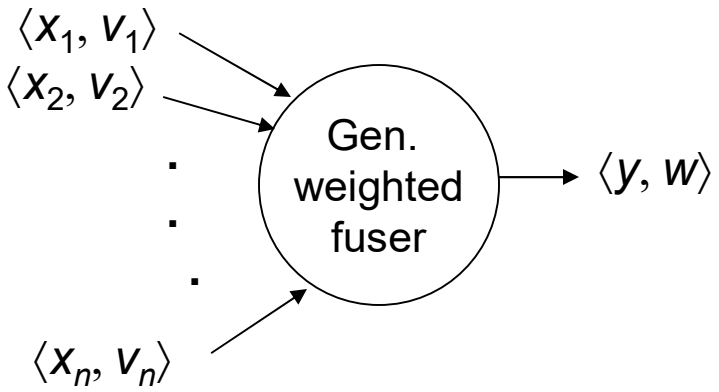
Find a minimal set of comparators that determine the agreement set



Complete this design by producing the "no agreement" signal

27.4 Variations in Voting

One can classify generalized weighted voting schemes into $2^4 = 16$ categories based on dichotomies associated with input data (x_is), output data (y), input weights (v_is), and output weight (w)



Input objects inflexible, or representing flexible "neighborhoods"

First entry in each box is the simpler of the two

	Input	Output
Data	Exact/ Inexact	Consensus/ Mediation
Weight	Oblivious/ Adaptive	Threshold/ Plurality

Total support from a subset of inputs (quorum), or shades of support from all inputs

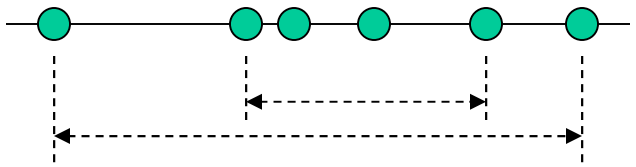
Input weights set at design time, or allowed to change dynamically (adjustable/variable)

A term such as "threshold voting" stands for 8 different methods

Support exceeds a lower bound, or is max over all possible outputs

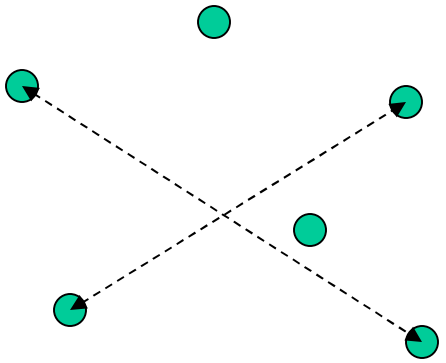
Generalized Median Voting

To find the median of a set of numbers, repeatedly remove largest and smallest numbers, until only one or two remain



If we replace "largest and smallest numbers" by "the two inputs that are furthest apart," we can use an arbitrary distance metric in our screening

A distance metric is any metric (mapping of pairs of inputs into real values) that satisfies the three conditions:



Isolation	$d(x, y) = 0$ iff $x = y$
Symmetry	$d(x, y) = d(y, x)$
Triangle inequality	$d(x, y) + d(y, z) \geq d(x, z)$

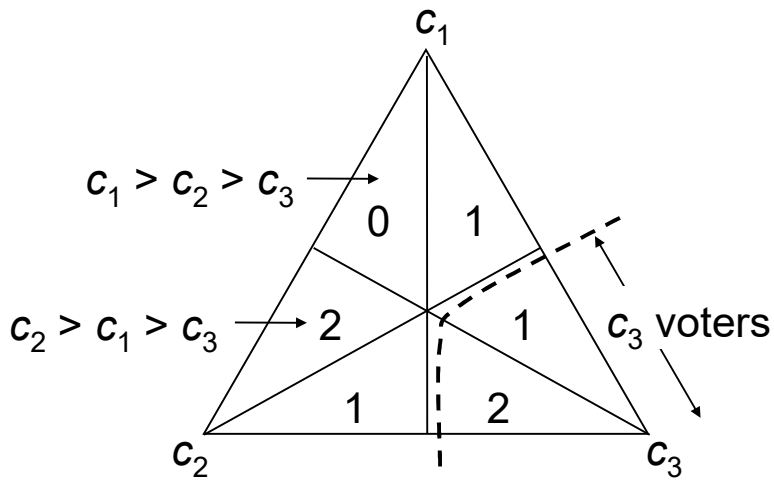
For example, the Hamming distance satisfies these conditions

The Impossibility of Perfect Voting

Properties of an ideal voting scheme:

1. No big brother
   (participants free to express preferences)
2. Independence of irrelevant alternatives
   (preference for one candidate over another is independent of all others)
3. Involvement
   (every outcome is possible)
4. No dictatorship or antidictatorship
   (outcome not always conforming to, or opposite of, one participant's view)

Arrow's Theorem:
No voting scheme exists that satisfies all four conditions



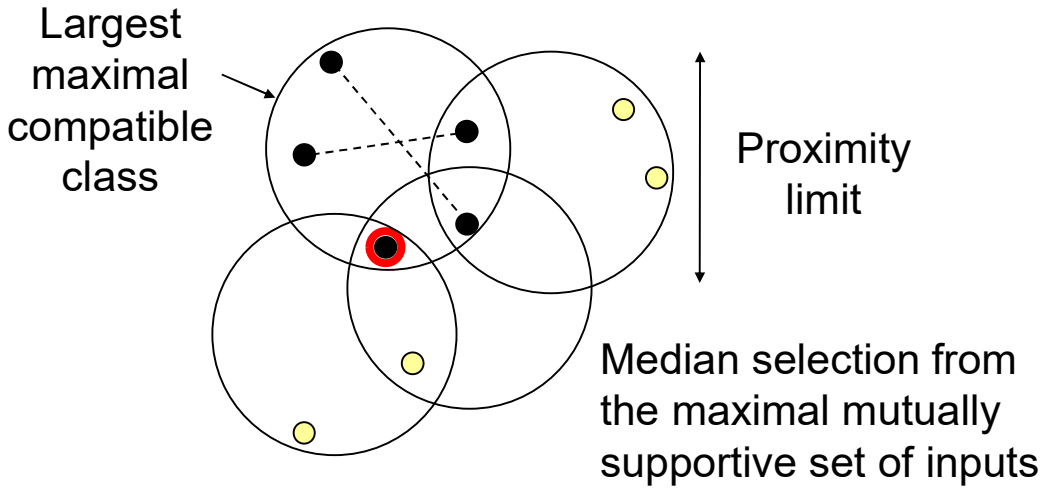
True majority voting scheme:
Each participant orders all the candidates; no circular preference allowed
Choose a candidate who beats every other one in pairwise competitions
(both simple majority and plurality rules fail to choose a candidate)

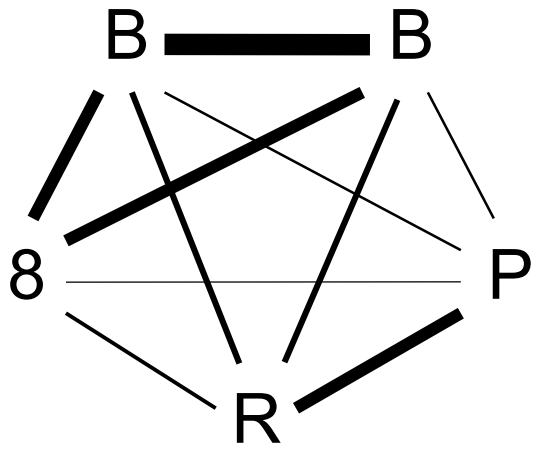
Approximate Voting

The notion of an input object "supporting" a particular output (akin to a hypothesis supporting an end result or conclusion) allows us to treat approximate and exact voting in the same way

Example 1: Input objects are points in the 2D space and the level of "support" between them is a function of their Euclidean distance

Example 2: Input objects are conclusions of character recognizers as to the identity of a character, with varying degrees of mutual support

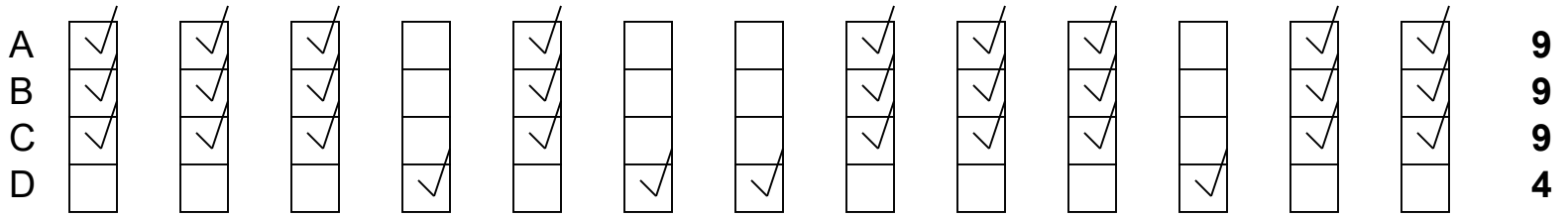




Approval Voting

Approval voting was introduced to prevent the splitting of votes among several highly qualified candidates from leading to the election of a less qualified candidate in plurality voting

In approval voting, a participant divides the candidates into two subsets of “qualified” and “not qualified” and indicates approval of the first subset



In the context of computing, approval voting is useful when a question has multiple answers or when the solution process is imprecise or fuzzy

Example question: What is a safe setting for a particular parameter in a process control system?

When the set of approved values constitute a continuous interval of real values, we have “interval” inputs and “interval” voting

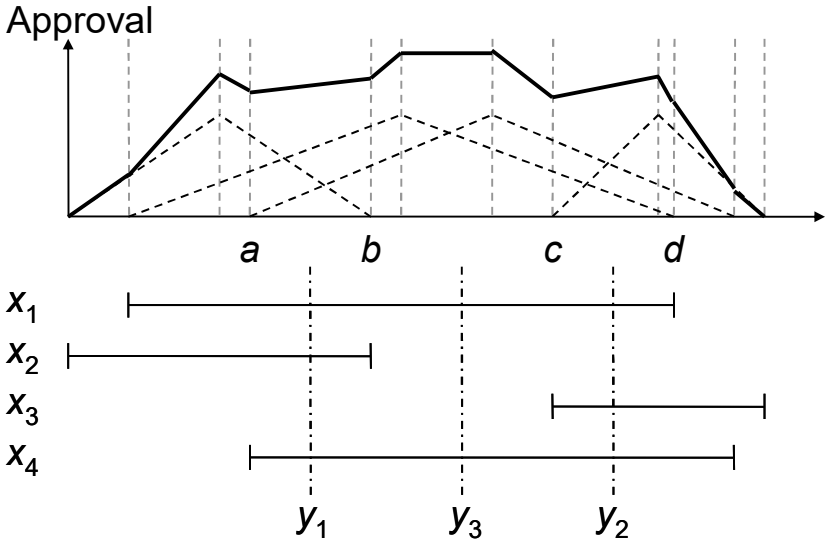
Interval Voting

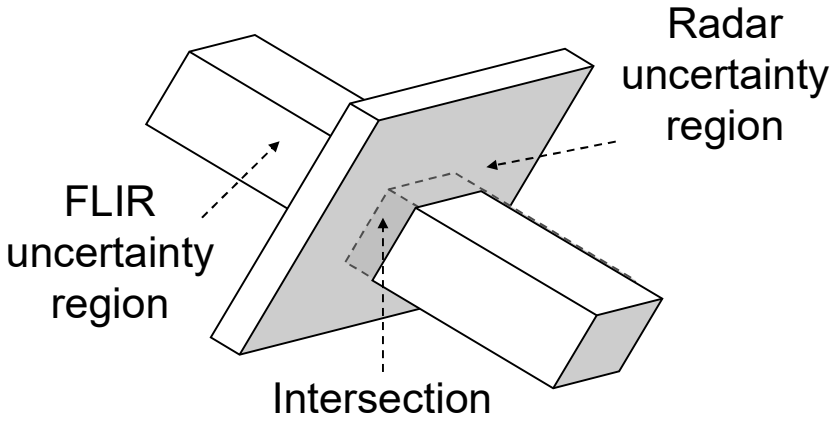
Inputs to the voting process are intervals, representing approved values

How should the voting result be derived from the input intervals?

Various combining rules can be envisaged
If there is overlap among all intervals, then the decision is simple

Depending on context, it may make sense to consider greater levels of approval near the middle of each interval or to associate negative approval levels outside the approved intervals





Combining two 3D intervals

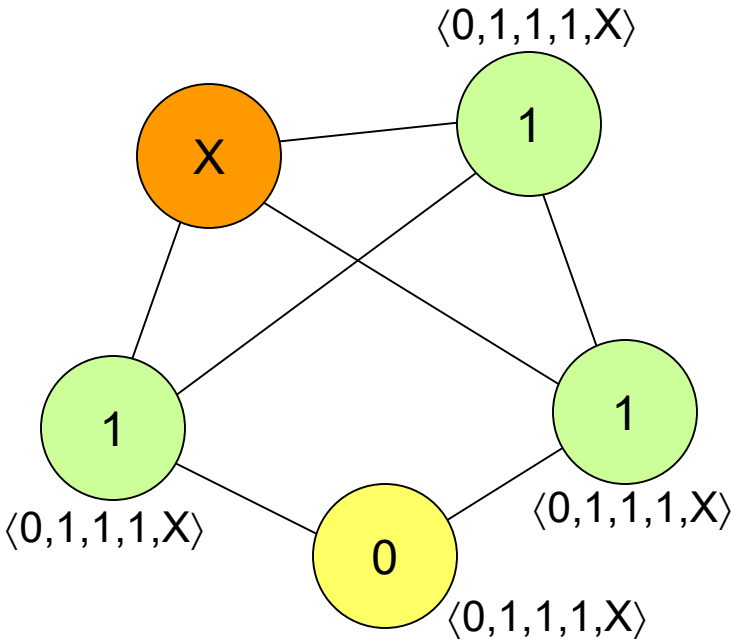
27.5 Distributed Agreement

Problem: Derive a highly reliable value from multiple computation results or stored data replicas at multiple sites

Key challenge: Exchange data among nodes so that all healthy nodes end up with the same set of values; this guarantees that running the same decision process on the healthy nodes produces the same result

Errors are possible in both data values and in their transmission between sites

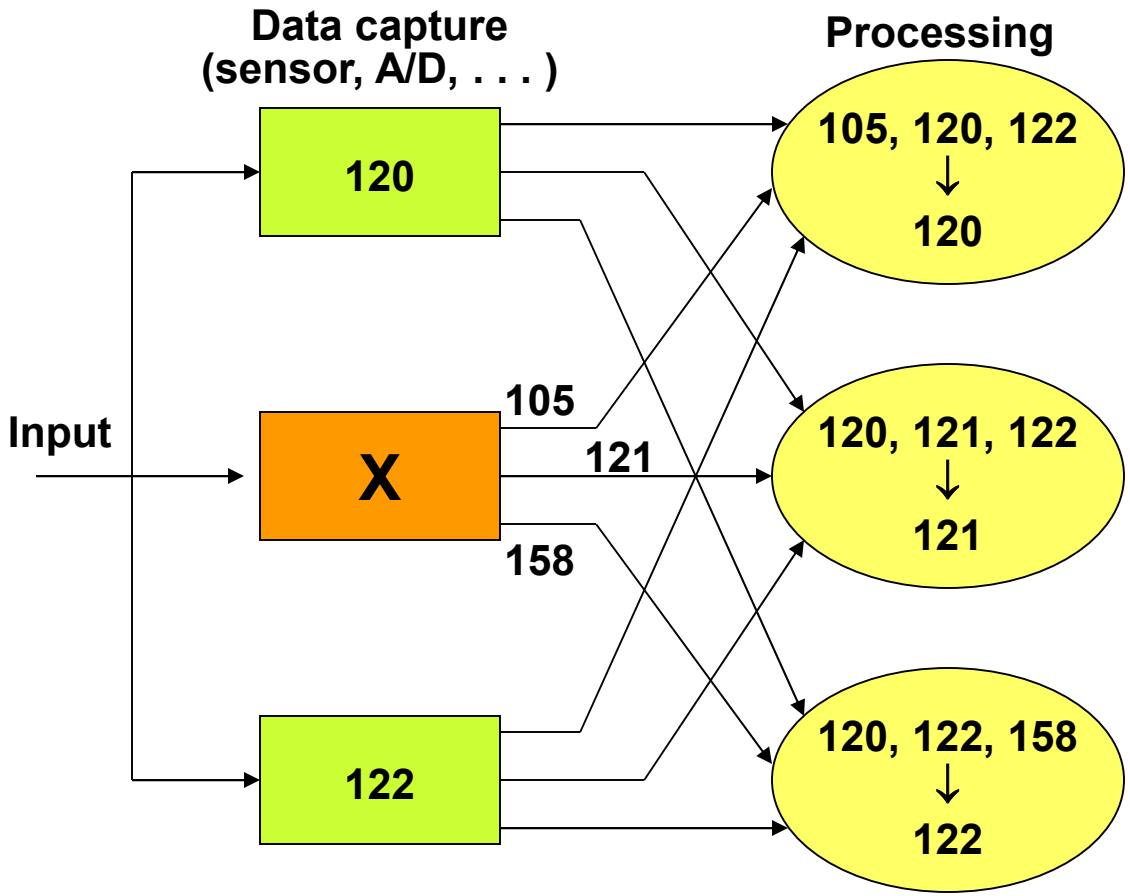
Agreement algorithms generally use multiple rounds of communication, with values held at each site compared and filtered, until the set of values held at all sites converge to the same set



Byzantine Failures in Distributed Voting

Three sites are to collect three versions of some parameter and arrive at consistent voting results

Assume median voting



The Interactive Consistency Algorithm

ICA(0) [no failure]

1. The transmitter sends its value to all other $n - 1$ nodes
2. Each node uses the value received from the transmitter, or a default value Φ if it received no value

ICA(f), $f > 0$ [f failures]

1. The transmitter sends its value to all other $n - 1$ nodes
2. Let v_i be the value received by node i from the transmitter, or a default value Φ if it received no value; node i then becomes the transmitter in its own version of ICA($f - 1$), sending its value to $n - 2$ nodes
3. For each node i, let $v_{i,j}$ be the value it received from node j, or a default value Φ if it received no value from node j. Node i then uses the value majority($v_{i,1}, v_{i,2}, \ldots, v_{i,i-1}, v_{i,i+1}, \ldots, v_{i,n}$)

$O(n^{f+1})$ messages needed, in $f + 1$ rounds, to tolerate f Byzantine failures

Building upon Consensus Protocols

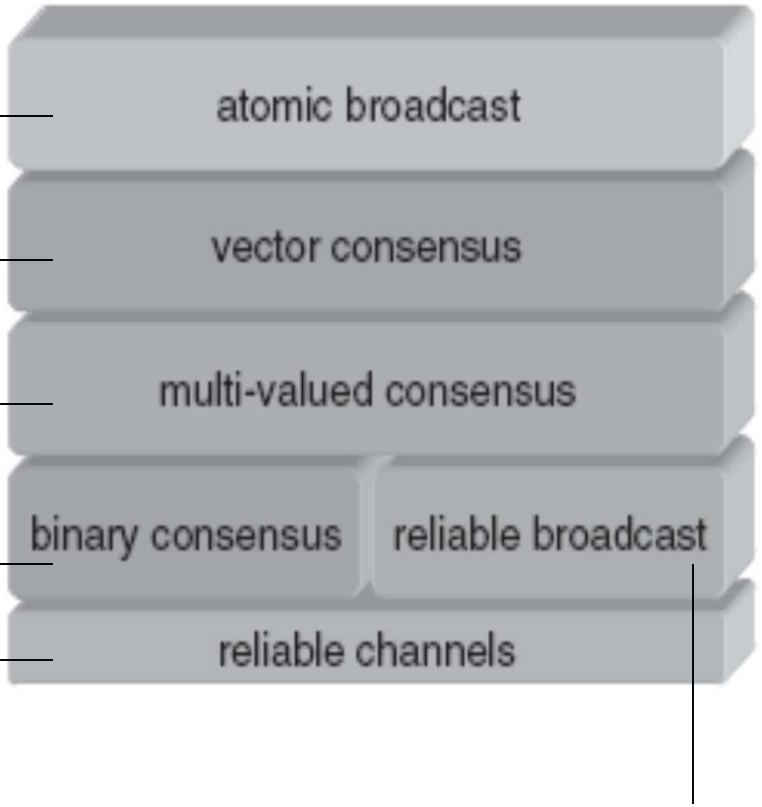
The same messages are delivered in the same order to all participating nodes

All healthy participants arrive at vectors with a majority of elements correct

All healthy participants arrive at vectors with correct value for every healthy node

Agreeing on one of two values, 0 or 1

If source and destination are healthy, message is eventually delivered unmodified



Message from a good node is eventually delivered to all good nodes unmodified

Source: M. Correia, N. Ferreira Neves, P. Veríssimo, "From Consensus to Atomic Broadcast: Time-Free Byzantine-Resistant Protocols without Signatures," *The Computer J.*, 2005.

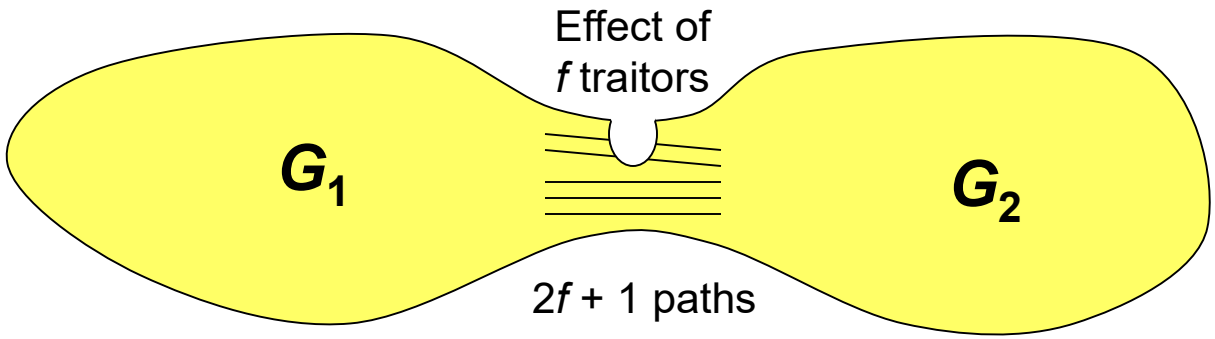
Correctness and Performance of ICA

Theorem 1: With ICA(f), all nonfailed nodes will agree on a common value, provided that $n \geq 3f + 1$ (proof is by induction on f)

ICA works correctly, but it needs an exponential number of messages:
$(n-1) + (n-1)(n-2) + (n-1)(n-2)(n-3) + \ldots + (n-1)(n-2) \ldots (n-m)$

More efficient agreement algorithms exist, but they are more difficult to describe or to prove correct; $f + 1$ rounds of message exchange is the least possible, so some algorithms trade off rounds for # of messages

Theorem 2: In a network G with f failed nodes, agreement is possible only if the connectivity is at least $2f + 1$



The Byzantine Generals Problem

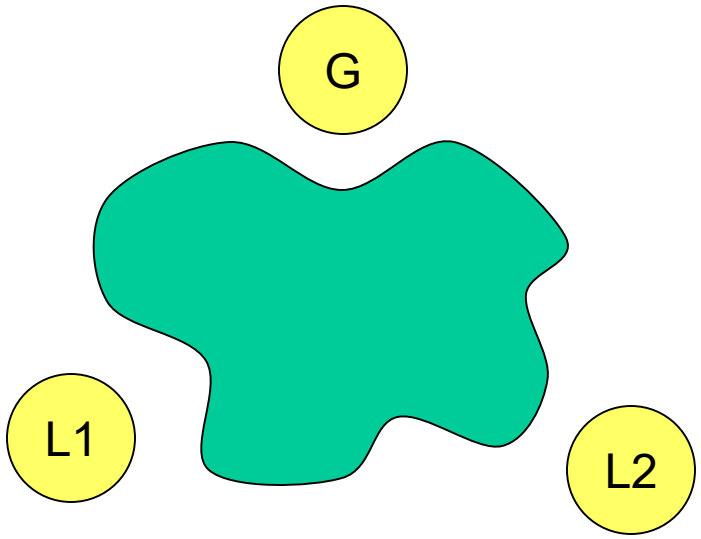
A general and n – 1 lieutenants lead n divisions of the Byzantine army camped on the outskirts of an enemy city

The n divisions can only communicate via messengers

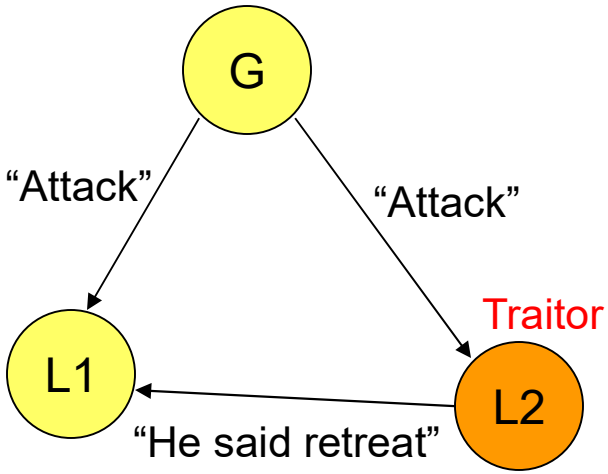
We need a scheme for the generals to agree on a common plan of action (attack or retreat), even if some of the generals are traitors who will do anything to prevent loyal generals from reaching agreement

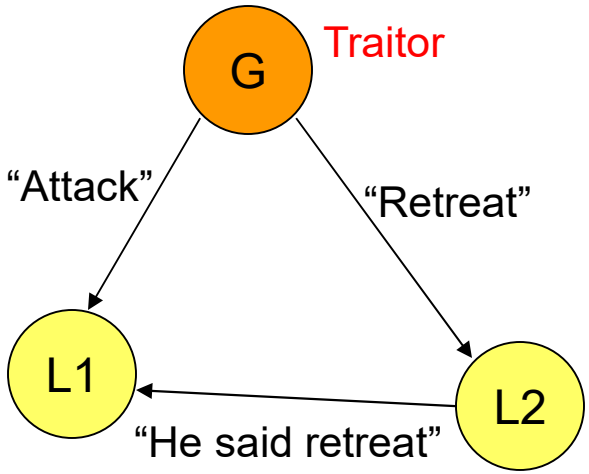
The problem is nontrivial even if messengers are totally reliable

With unreliable messengers, the problem is very complex

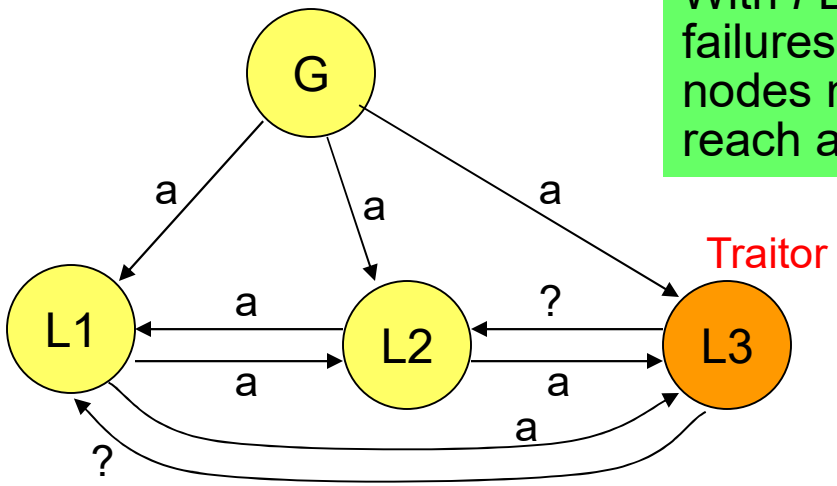


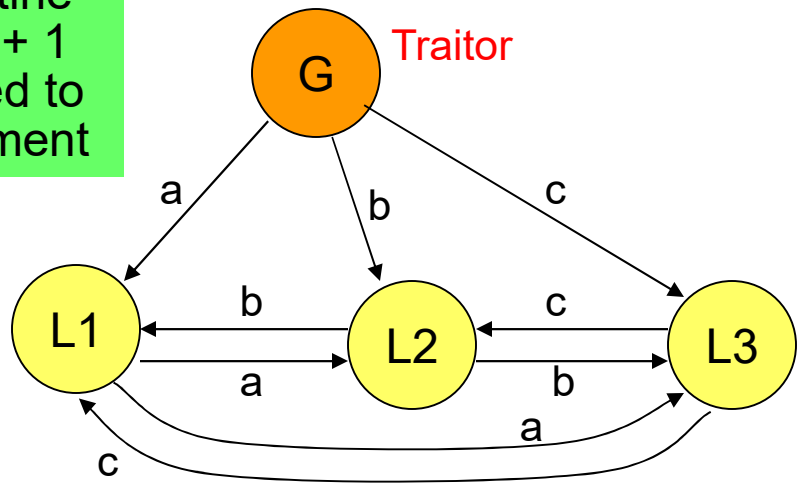
Byzantine Generals with Reliable Messengers





With *f* Byzantine failures, ≥ 3*f* + 1 nodes needed to reach agreement





27.6 Byzantine Resiliency

To tolerate *f* Byzantine failures:

We need $3f + 1$ or more FCRs (fault containment regions)

FCRs must be interconnected via at least $2f + 1$ disjoint paths

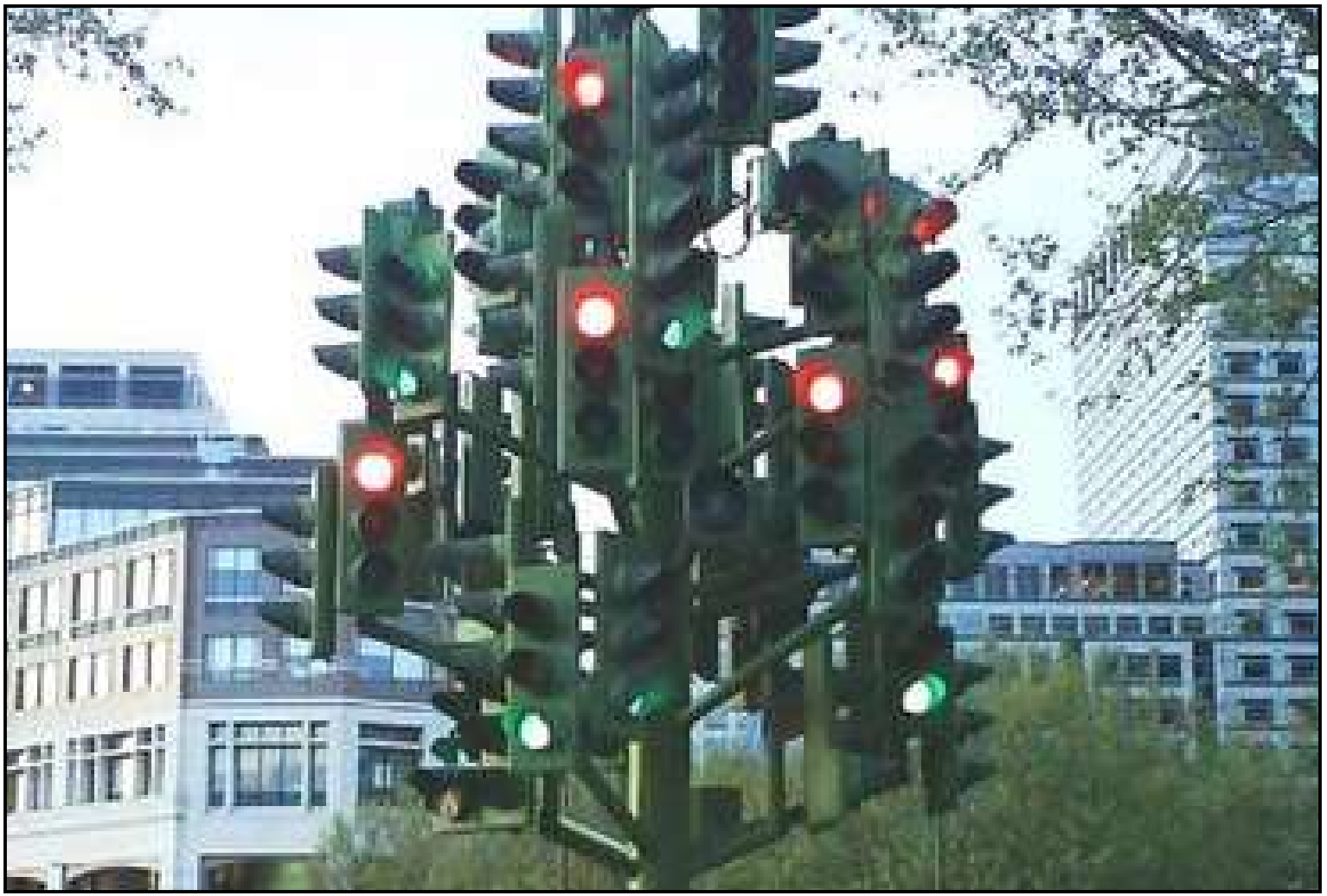
Inputs must be exchanged in at least $f + 1$ rounds

Corollary 1: Simple 3-way majority voting is not Byzantine resilient

Corollary 2: Because we need $2f + 1$ good nodes out of a total of $3f + 1$ nodes, a fraction $(2f + 1)/(3f + 1) = 2/3 + 1/(9f + 3)$ of the nodes must be healthy

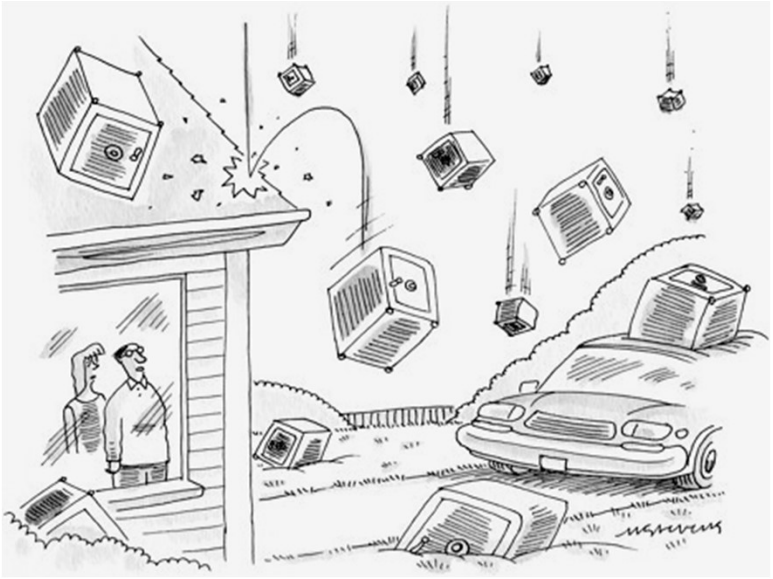
This is greater than a supermajority (2/3) requirement

28 Fail-Safe System Design



"Have you seen the use-by date on this porridge?"

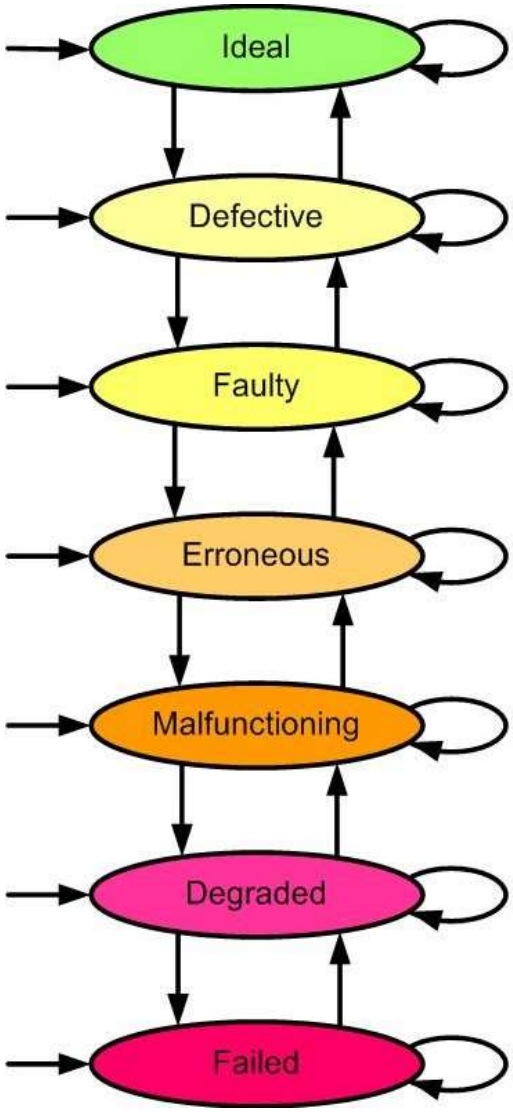
"Damn those Health & Safety guys."



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. Robust Parallel Processing
Part VI — Degradations: Behavioral Lapses (The Service-Level View)	Methods / Examples	21. Degradation Allowance 22. Degradation Management 23. Resilient Algorithms 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



28.1 Fail-Safe System Concepts

Fail-safe: Produces one of a predetermined set of safe outputs when it fails as a result of "undesirable events" that it cannot tolerate

Fail-safe traffic light: Will remain stuck on red

Fail-safe gas range/furnace pilot flame: Cooling off of the pilot assembly due to the flame going out will shut off the gas intake valve

A fail-safe digital system must have at least two binary output lines, together representing the normal outputs and the safe failure condition

Reason: If we have a single output line, then even if one value (say, 0) is inherently safe, the output stuck at the other value would be unsafe

Two-rail encoding is a possible choice: **0**: 01, **1**: 10, **F**: 00, 11, or both

Totally fail-safe: Only safe erroneous outputs are produced, provided another failure does not occur before detection of the current one

Ultimate fail-safe: Only safe erroneous output is produced, forever

28.2 Principles of Safety Engineering

Principles for designing a safe system (J. Goldberg, 1987)

1. Use barriers and interlocks to constrain access to critical system resources or states

2. Perform critical actions incrementally, rather than in a single step

3. Dynamically modify system goals to avoid or mitigate damages

4. Manage the resources needed to deal with a safety crisis, so that enough will be available in an emergency

5. Exercise all critical functions and safety features regularly to assess and maintain their viability

6. Design the operator interface to provide the information and power needed to deal with exceptions

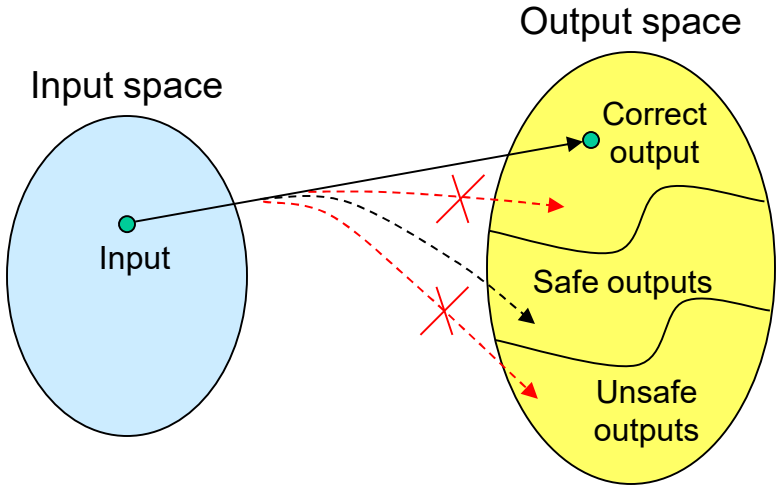
7. Defend the system against malicious attacks

28.3 Fail-Safe Specifications

Amusement park train safety system

Signal s_B when asserted indicates that the train is at beginning of its track (can move forward, but should not be allowed to go back)

Signal s_E when asserted indicates that the train is at end of its track (can go back, but should not move forward)



Is the specification above consistent and complete?

No, because it does not say what happens if $s_B = s_E = 1$; this would not occur under normal conditions, but because such sensors are often designed to fail in the safe mode, the combination is not impossible

Why is this a problem, though? (Train simply cannot be moved at all)

Completeness will prevent potential implementation or safety problems

Simple Traffic Light Controller

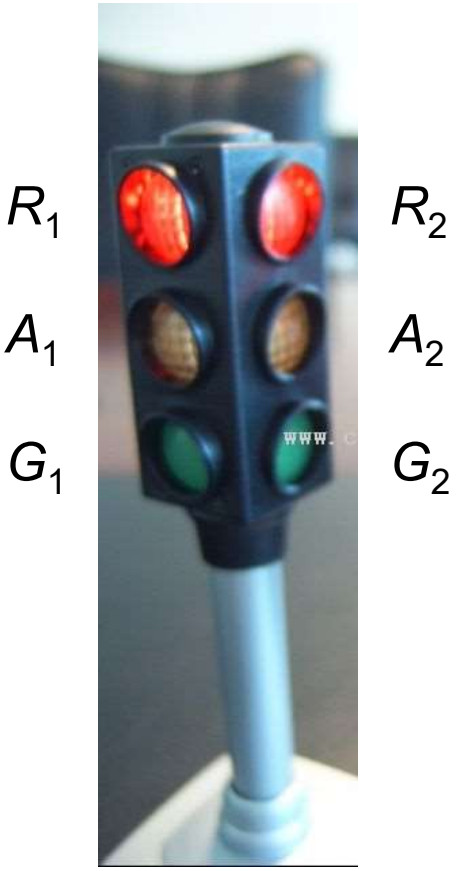
Six signals: red, amber, and green light control for each direction (no left turn signals)

Safety flip-flop, when set, forces flashing red lights in both directions

$S = (A_1 \vee G_1) \wedge (A_2 \vee G_2)$

Let g_1 be the computed green light signal for direction 1, and so on for other signals, and G_1 be the signal that controls the light

$G_1 = \overline{S} \wedge g_1 \wedge (\overline{A_2 \vee G_2})$



28.4 Fail-Safe Combinational Logic

Similar to the design of self-checking circuits:
Design units so that for each fault type of interest, the output is either correct or safe

Totally fail-safe: Fail-safe and self-testing [Nico98]

Strongly fail-safe: Intermediate between fail-safe and totally fail-safe
For each fault x it is either totally fail-safe *or* it is fail-safe with respect to x and strongly fail-safe with respect to $x \cup y$, where y is another fault

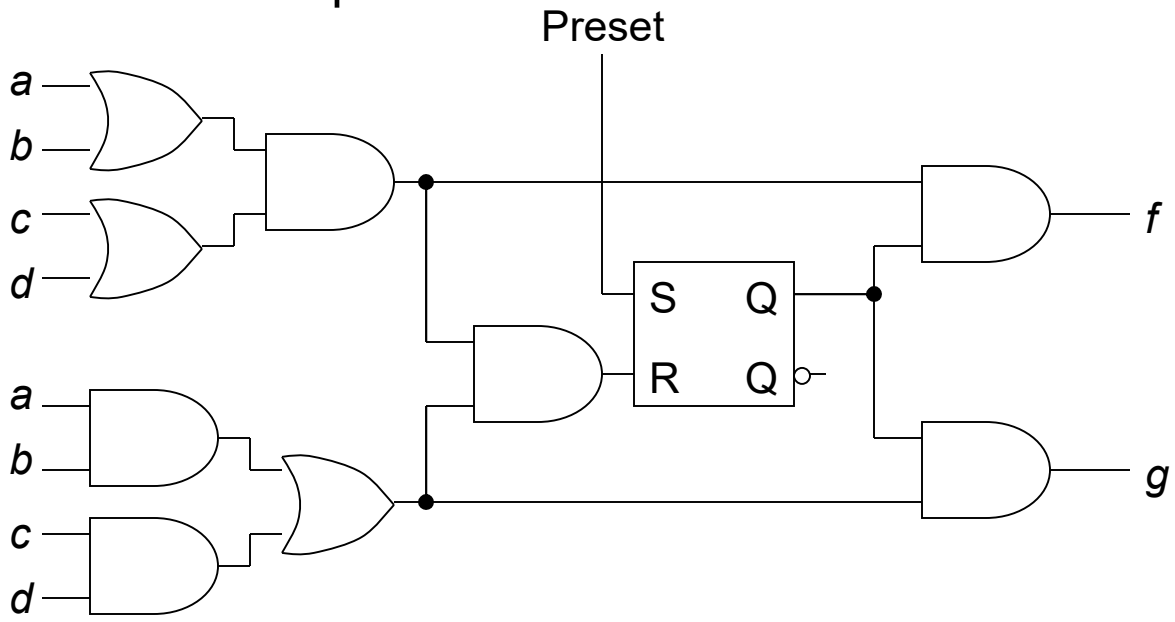
Fail-Safe 2-out-of-4 Code Checker

Input: 4 bits *abcd*, exactly 2 of which must be 1s

Output: *fg* = 01 or 10, if the input is valid
00 safe erroneous output
11 unsafe erroneous output

Codewords

a	*b*	*c*	*d*
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0



Output will become permanently 00 upon the first unsafe condition

28.5 Fail-Safe State Machines

Use an error code to encode states

Implement the next-state logic so that the machine is forced to an error state when something goes wrong

Possible design methodology:

Use Berger code for states, avoiding the all 0s state with all-1s check, and vice versa

Implement next-state logic equations in sum-of-products form for the main state bits and in product-of-sums form for the check state bits

The resulting state machine will be fail-safe under unidirectional errors

	Input	
State	x=0	x=1
A	E	B
B	C	D
C	A	D
D	E	D
E	A	D

State	Encoding
A	001 10
B	010 10
C	011 01
D	100 10
E	101 01

Hardware overhead for n-state machine consists of O(log log n) additional state bits and associated next-state logic, and a Berger code checker connected to state FFs

28.6 System- and User-Level Safety

Principle: False alarms are tolerable, but missed failures are not

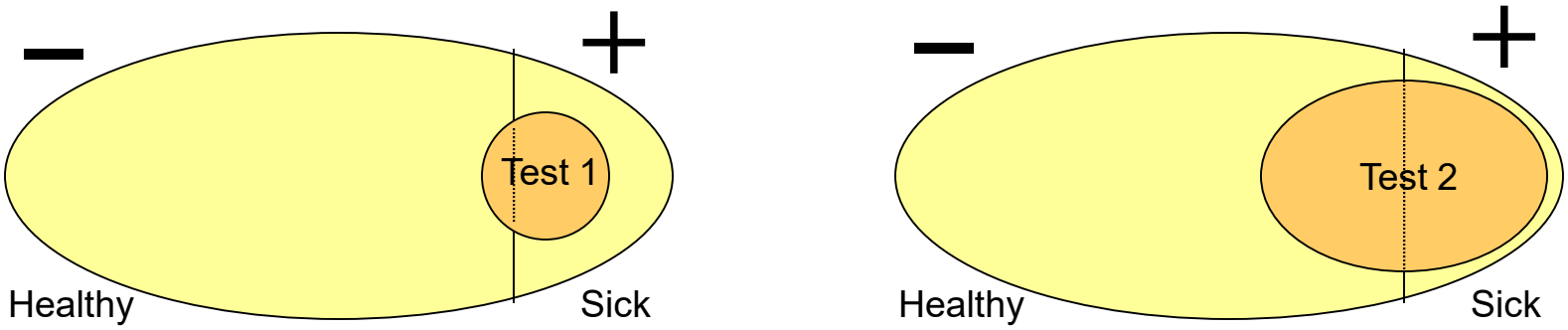
A binary test is characterized by its sensitivity and specificity

Sensitivity is the fraction of positive cases (e.g., people having a particular disease) that the test identifies correctly

Specificity is the fraction of negative cases (e.g., healthy people) that are correctly identified

There exists a fundamental tradeoffs between sensitivity and specificity

We want to err on the side of too much sensitivity



Interlocks and Other Safety Mechanisms

Dead-man's switch (kill switch): Used in safety-critical systems that should not be operable without the presence of an operator

Usually, takes the form of a handle or pedal that the operator must touch or press continuously

First used in trains to stop them in the event of operator incapacitation

A Past, Present, and Future

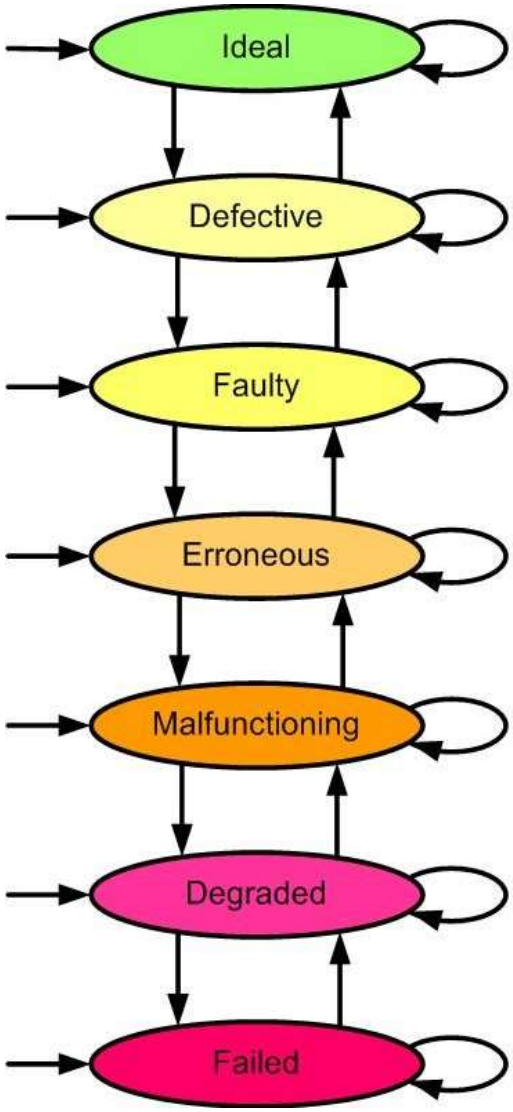
"I am not disorganized — I know *exactly* where everything is! The newer stuff is on top and the older stuff is on the bottom."

"We couldn't afford faster computers, so we just made them *sound* faster."

"I couldn't do my homework because my computer has a virus and so do all my pencils and pens."

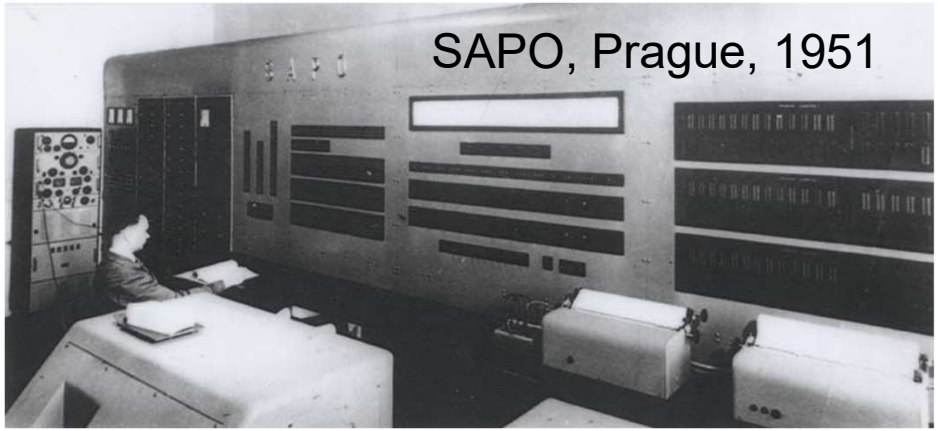
STRUCTURE AT A GLANCE

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. Robust Parallel Processing
Part VI — Degradations: Behavioral Lapses (The Service-Level View)	Methods / Examples	21. Degradation Allowance 22. Degradation Management 23. Resilient Algorithms 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		



A.1 Historical Perspective

Antonin Svoboda (1907-80), built SAPO, the first fault-tolerant computer—it used triplication with voting for error correction; necessitated by poor component quality



Motivated by NASA's planned Solar System exploration taking 10 years (the Grand Tour), Algirdas Avizienis built Jet Propulsion Lab's self-testing-and-repairing (STAR) computer

Dependable Computing in the 1950s

SAPO, built in Czechoslovakia in 1951 by Antonin Svoboda
Used magnetic drums and relays

SAPO fault tolerance features were motivated by [Renn84]:
Low quality of components available to designers
Severe (political) pressure to get things right

With the advent of transistors, which were much more reliable than relays and vacuum tubes, redundancy methods took a back seat (used only for exotic or highly critical systems)

Dependable Computing in the 1960s

NASA began extensive development in long-life, no-maintenance computer systems for (manned) space flight

Orbiting Astronomical Observatory (OAO), transistor-level fault masking

The Apollo Guidance System, triplication and voting

Mars Voyager Program (late 1960s, cancelled)

Deep space missions (e.g., the cancelled Solar System Grand Tour)
JPL self-testing-and-repairing computer (STAR)

Also:
AT&T Electronic Switching System (ESS), down time of minutes/decade
Serviceability features in mainframes, such as IBM System/360

Dependable Computing in the 1970s

Two influential avionics systems:
Software-implemented fault tolerance (SIFT), SRI, Stanford Univ.
  Spun off a commercial venture, August Systems (process control)
Fault-tolerant multiprocessor (FTMP), Draper Lab., MIT

The Space Shuttle (4 identical computers, and a 5th one running different software developed by a second contractor)

Redundancy in control computers for transportation (trains)

Also:
First Fault-Tolerant Computing Symposium, FTCS (1971)
Systems for nuclear reactor safety
Tandem NonStop (1976), transaction processing

Dependable Computing in the 1980s

1980: IFIP WG 10.4 on Dependable Computing and Fault Tolerance

Continued growth in commercial multiprocessors (Tandem)

Advanced avionics systems (Airbus)

Automotive electronics

Reliable distributed systems (banking, transaction processing)

Dependable Computing in the 1990s

DCCA conferences, started in 1989, continued until 1999 (DCCA-7); subsequently merged with FTCS to become DSN Conf.

SEU tolerance

BIST

IBM G4 fault-tolerant mainframe

Robust communication and collaboration protocols
(routing, broadcast, f-t CORBA)

Dependable Computing in the 2000s

2000: First DSN Conf. (DSN-2000, 30th in series going back to FTCS)
Now composed of two symposia
DCCS: Dependable Computing and Communications Symp.
PDS: Performance and Dependability Symp.

2004: *IEEE Trans. Dependable and Secure Computing*

Deep submicron effects

F-T ARM, Sparc

Intrusion detection and defense

Widespread use of server farms for directory sites and e-commerce (Google, e-Bay, Amazon)

Dependable Computing in the 2010s

2010: DSN Conf. held its 40th in series, going back to FTCS

Cloud computing reliability focus of an *IEEE TDSC* 2013 special issue

Greater integration of reliability and security concerns

End of decade: Half-century mark of the field and of the DSN Conf.

A.2 Long-Life Systems

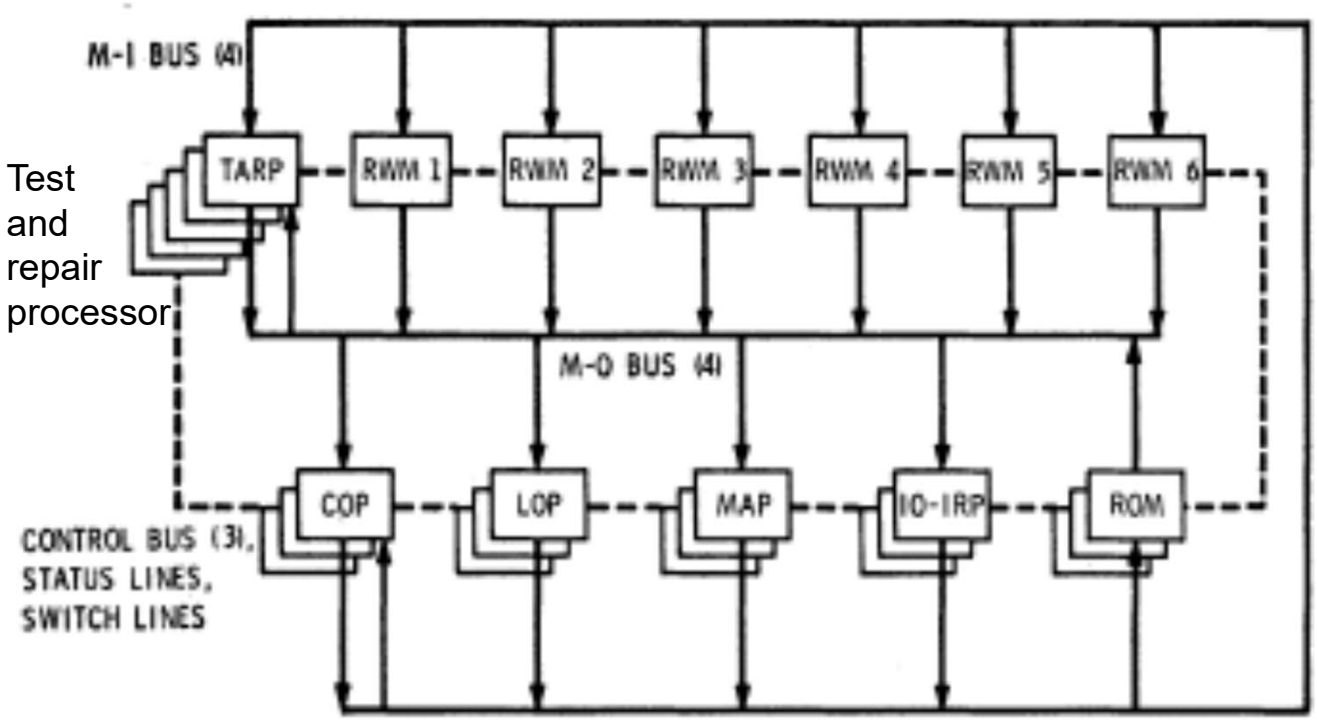
Genesis: Computer systems for spacecraft on multiyear missions, with no possibility of repair

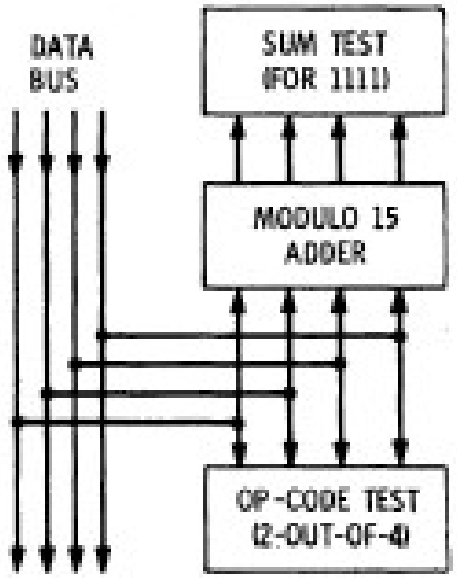
Today: More of the same, plus remotely located, hard-to-access systems for intelligence gathering or environmental monitoring

Typical systems for case studies:
NASA OAO, Galileo, JPL STAR, . . . , Space Station
Communication satellites
Remote sensor networks

The JPL STAR Computer

Became operational in 1969, following studies that began in 1961
Standby redundancy for most units, 3 + 2 hybrid redundancy for TARP
mod-15 inverse residue code to check arithmetic ops and bus transfers
Also used other codes, plus various hardware/software sanity checks





Bus checker

A.3 Safety-Critical Systems

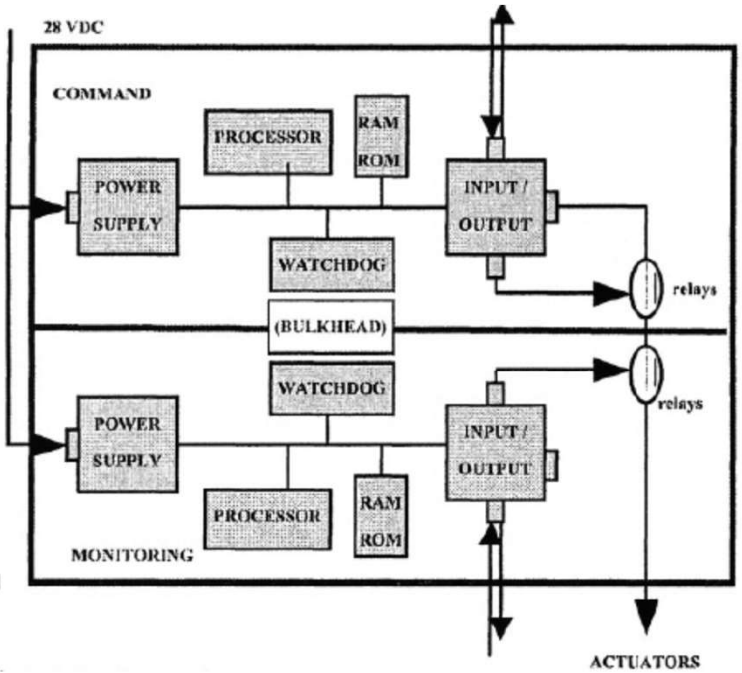
Genesis: Flight control, nuclear reactor safety, factory automation

Today: More of the same, plus high-speed transportation, health monitoring, surgical robots

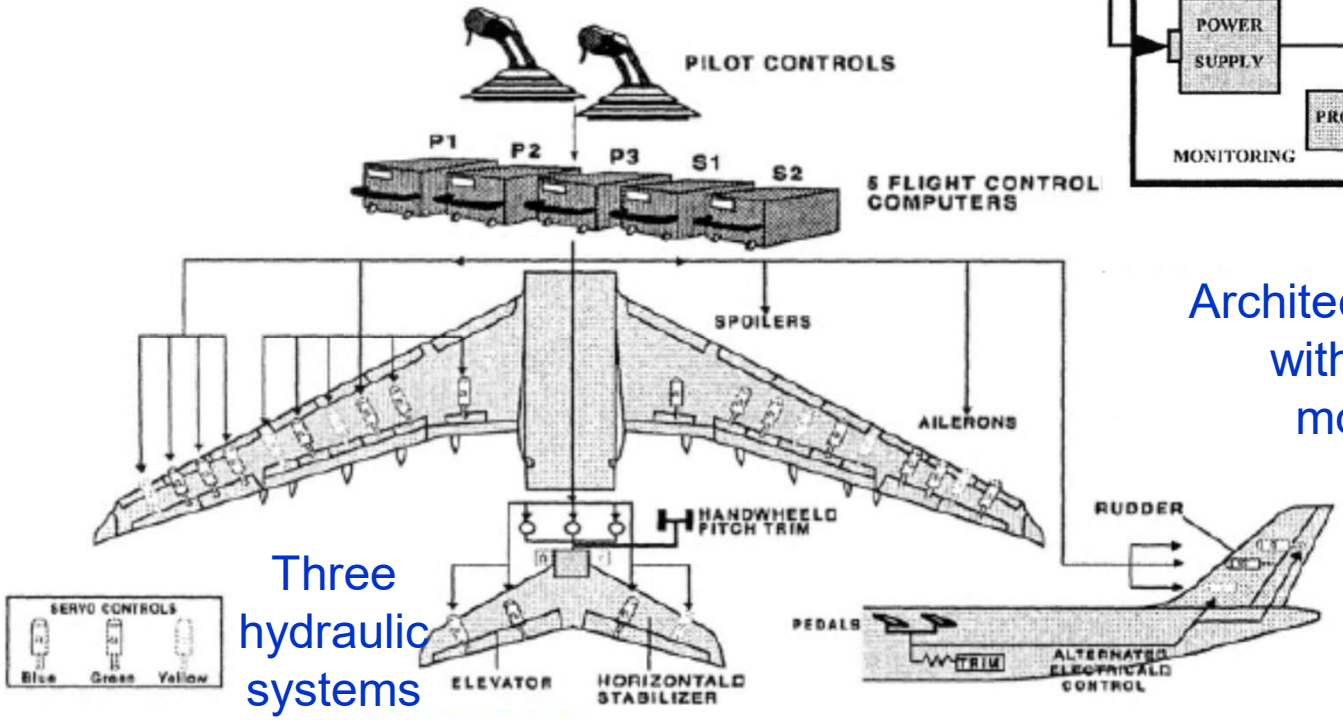
Typical systems for case studies:
CMU C.vmp
Stanford SIFT
MIT FTMP
August Systems
High-speed train controls
Automotive computers

Avionic Fly-by-Wire Systems

Airbus A320 entered operation in 1988
Other models include A340 and A380
Primary (P) and secondary (S) computers
(different designs and suppliers)
Multiple redundant software modules



Architecture of one computer, with its command and monitoring sections

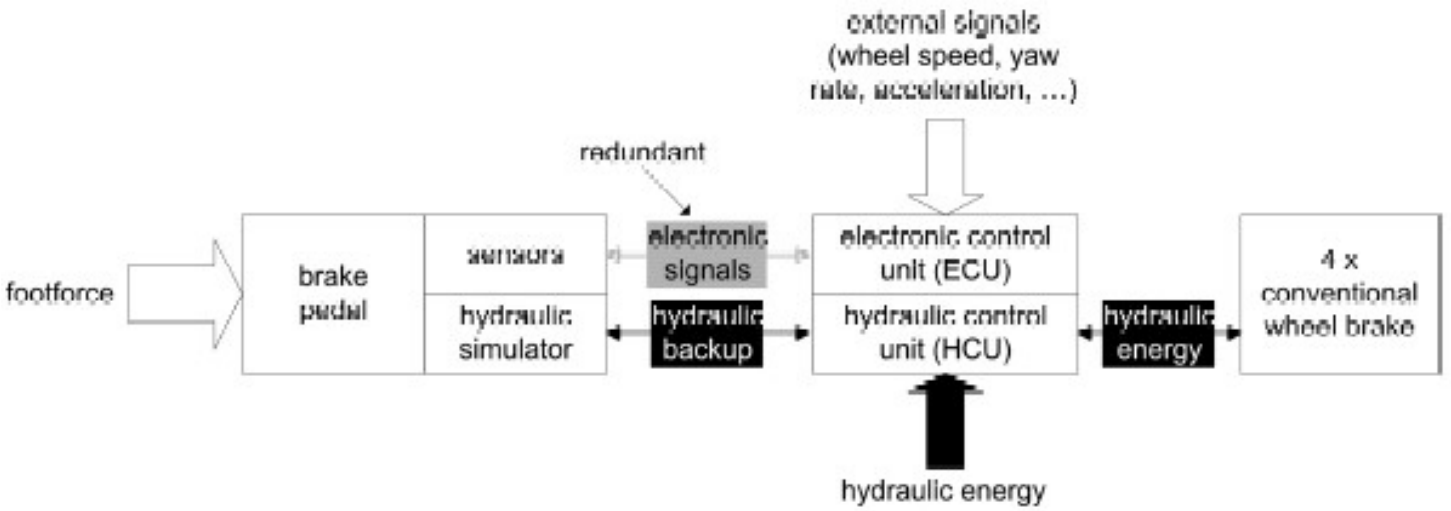


Three hydraulic systems

Source: [Trav04]

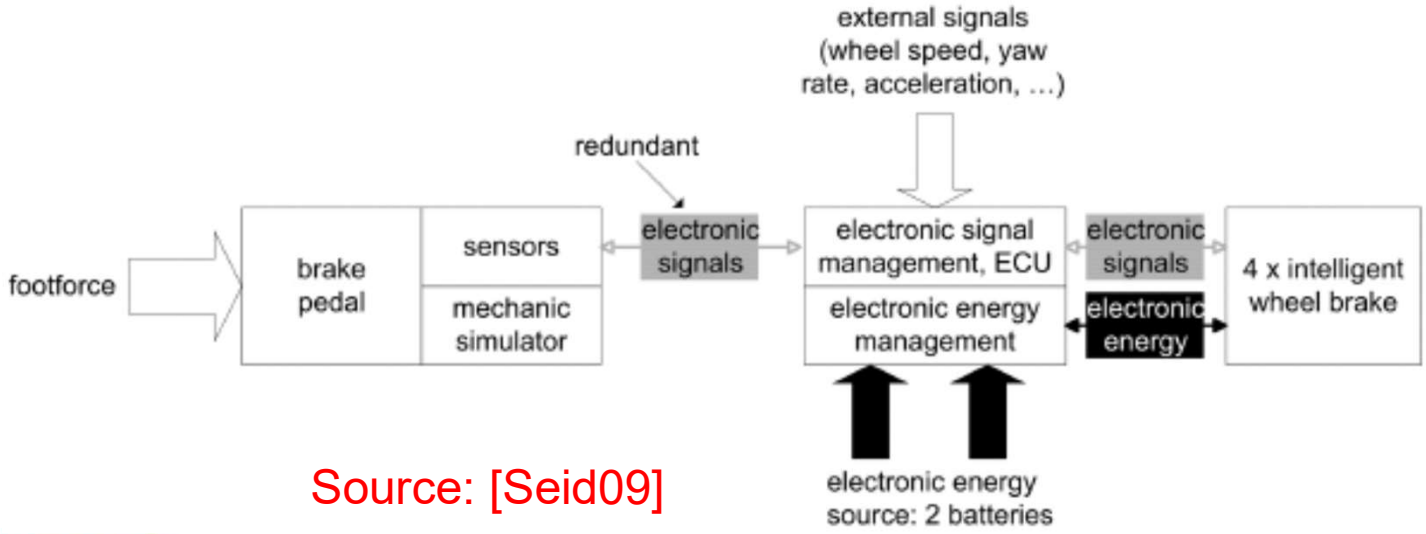
Automotive Drive-by-Wire Systems

Interim braking solution with mixed electronics and hydraulics



Input Processing Actuation

Fully electronic, with redundant buses, and power supplies



Source: [Seid09]

A.4 High-Availability Systems

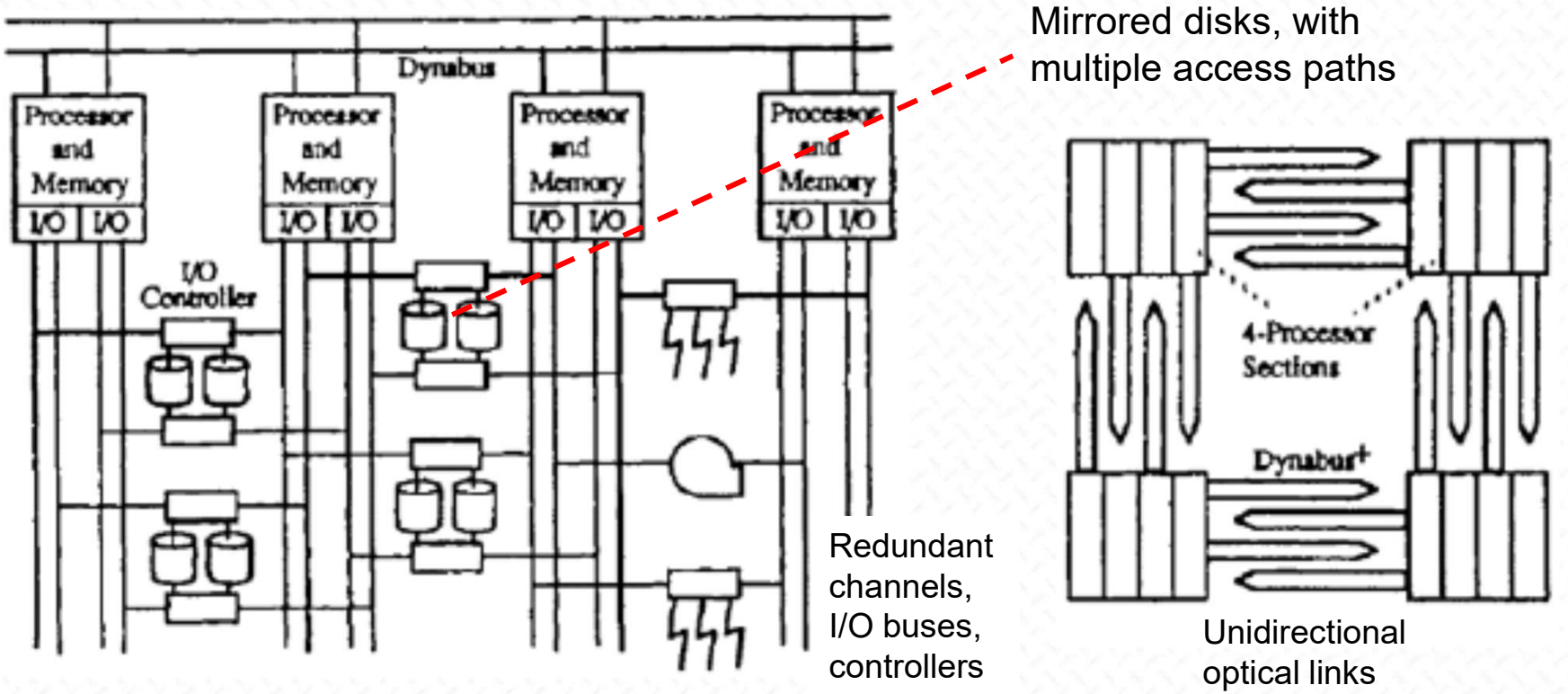
Genesis: Electronic switching systems for telephone companies

Today: More of the same, plus banking, e-commerce, social networking, and other systems that can ill-afford even very short down times

Typical systems for case studies:
AT&T ESS 1-5, telephone switching, 1965-1982
Tandem NonStop I/II- . . . -Cyclone-CLX800, 1976-1991
Stratus FT200-XA2000, 1981-1990
Banking systems
Portals and e-commerce (Google, Amazon)

Tandem NonStop Cyclone

Announced in 1989 for database and transaction-processing applications (descendant of the first NonStop system announced in 1976)

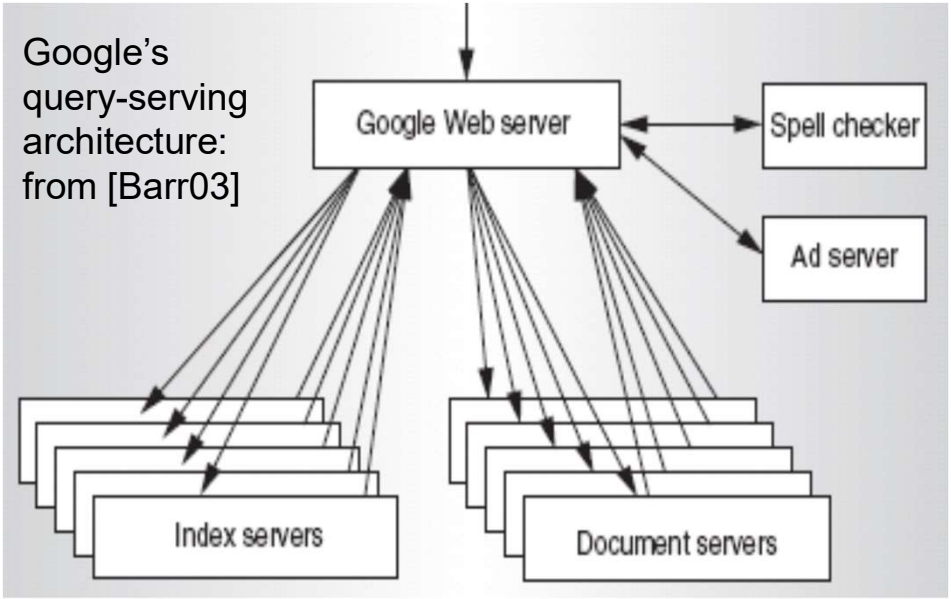


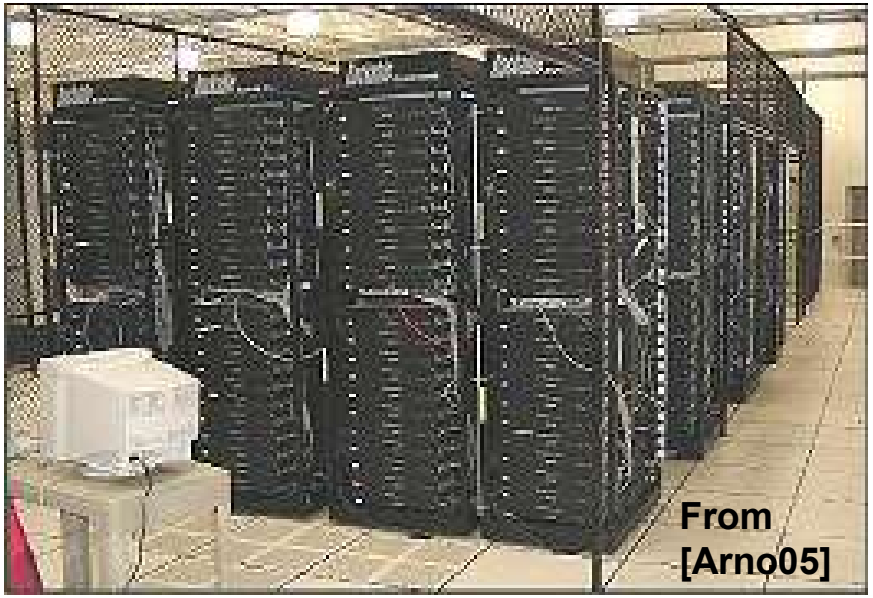
(a) One section of NonStop Cyclone system

(b) Dynabus+ connecting 4 sections

Google Data Center Architecture

Uses commodity (COTS) processing and communication resources
Data replication and software used to handle reliability issues
Massive hardware replication, needed for capacity and high performance, also leads to fault tolerance
Consistency problems are simplified by forcing read-only operation most of the time and shutting down sections of the system for infrequent updates





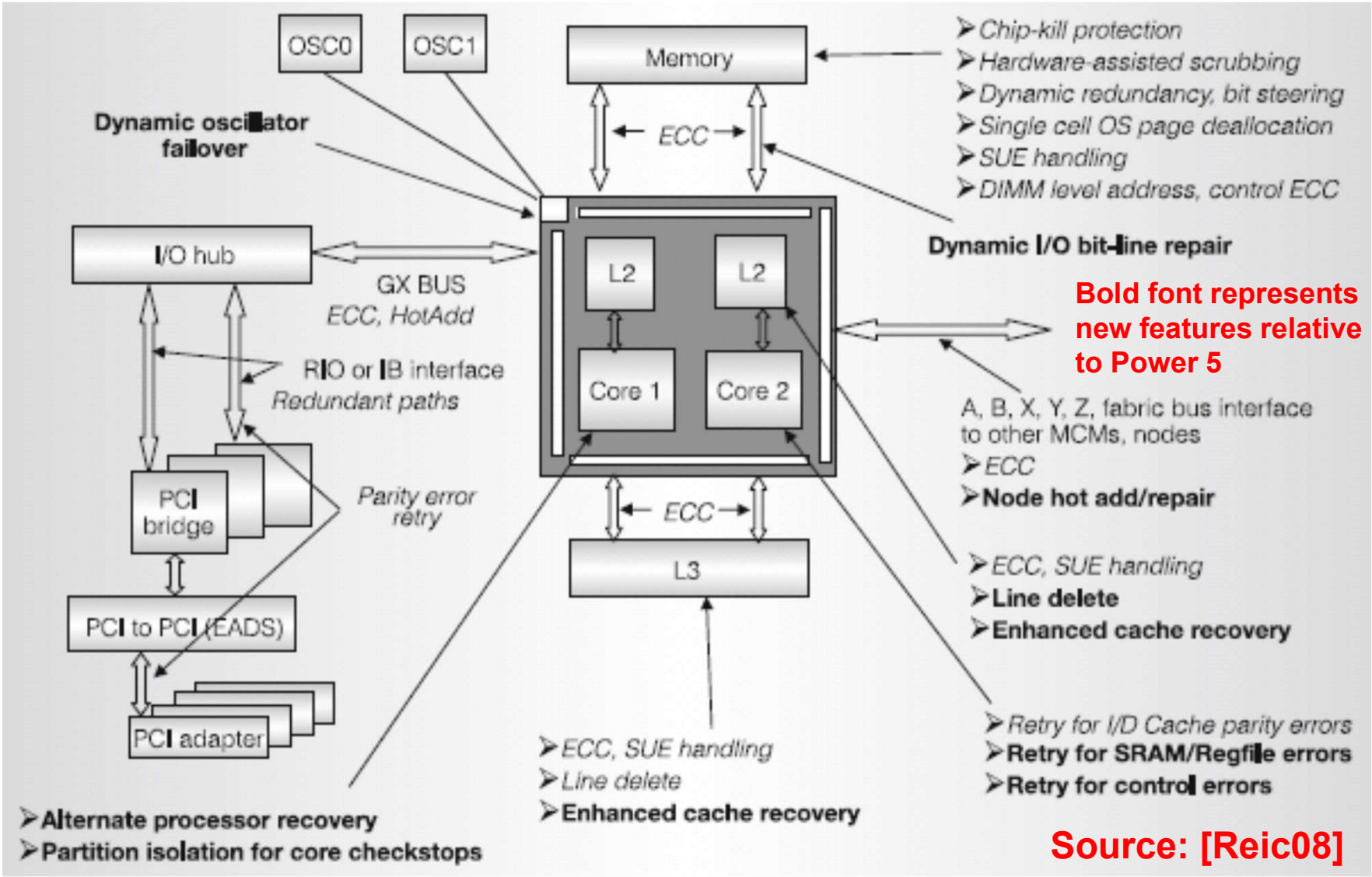
A.5 Commercial and Personal Systems

Genesis: Due to highly unreliable components, early computers had extensive provisions for fault-masking and error detection/correction

Today: Components are ultrareliable, but there are so many of them that faults/errors/malfunctions are inevitable

Typical systems for case studies:
SAPO
IBM System 360
IBM POWER6 microprocessor

The IBM Power6 Microprocessor



A.6 Trends, Outlook, and Resources

- Nobel Laureate Physicist Niels Bohr said:
  "Prediction is very difficult, especially if it's about the future."
  [Paraphrased by Yogi Berra in his famous version]
- Anonymous quotes on the perils of forecasting:
  "Forecasting is the art of saying what will happen, and then explaining why it didn't."
  "There are two kinds of forecasts: lucky and wrong."
  "A good forecaster is not smarter than everyone else; he merely has his ignorance better organized."
- Henri Poincare was more positive on prediction:
  "It is far better to foresee even without certainty than not to foresee at all."

2020s | 2030s | 2040s | 2050s

Current Challenges in Dependable Computing

Dependable computer systems and design methods continue to evolve

Current trend: Building dependable systems from mass-produced, commercial off-the-shelf (COTS) subsystems, not custom-made units (that is, dependability is viewed as a layer put around otherwise untrustworthy computing elements)

This is similar to the RAID approach to dependable mass storage

Challenge 1: The curse of shrinking electronic devices (nanotechnology)

Challenge 2: Reliability in cloud computing (opportunities & problems)

Challenge 3: Smarter and lower-power redundancy schemes (brainlike)

Challenge 4: Ensuring data longevity over centuries, even millennia

Challenge 5: Dependability verification (reasoning about uncertainty)

Challenge 6: Counteracting combined effects of failures and intrusions

Challenge 7: Reliability as roadblock for exascale systems (reliability wall)

The Curse of Shrinking Electronic Devices

Size shrinkage leads to

- Unreliable operation due to “crummy” devices and interconnects
- Need for modeling parameter fluctuations due to process variations
- Difficulties in testing, particularly with regard to delay faults

Reliability issues

- Some of the area/power gains must be reinvested in reliability features

Circuit modeling

- Can no longer model by considering only worst-case latencies

Testing challenges

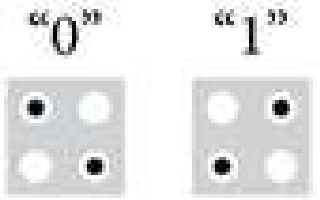
- Need for more tests to cover combinations of parameter values

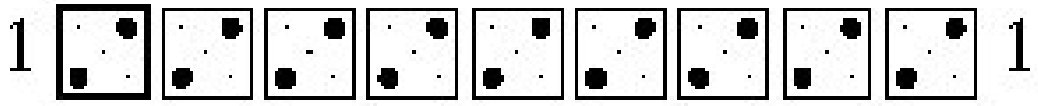
http://webhost.laas.fr/TSF/WDSN10/WDSN10_files/Slides/WDSN10_Wunderlich-Becker-Polian-Hellebrand.pdf

Computing with Majority Elements

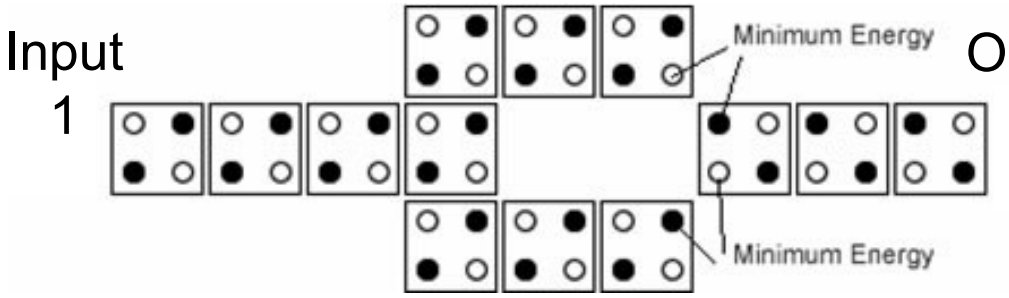
Several new technologies offer simple and power-efficient majority gates

In quantum-dot cellular automata (QCA), each cell contains 2 electrons that can tunnel between corner spots (dots), but can't leave the cell

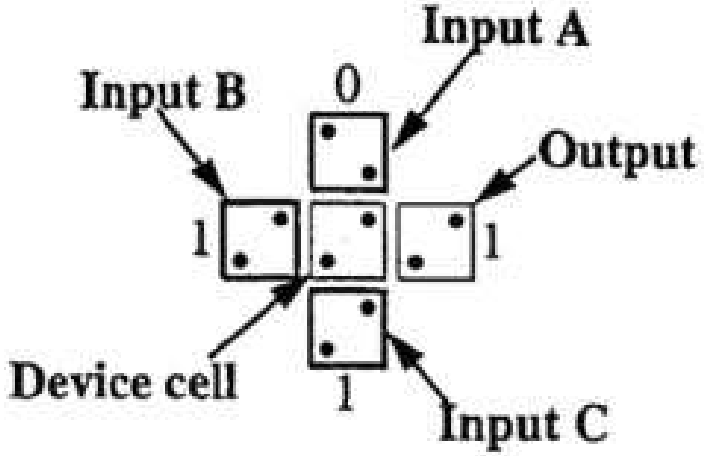




Wire



Inverter

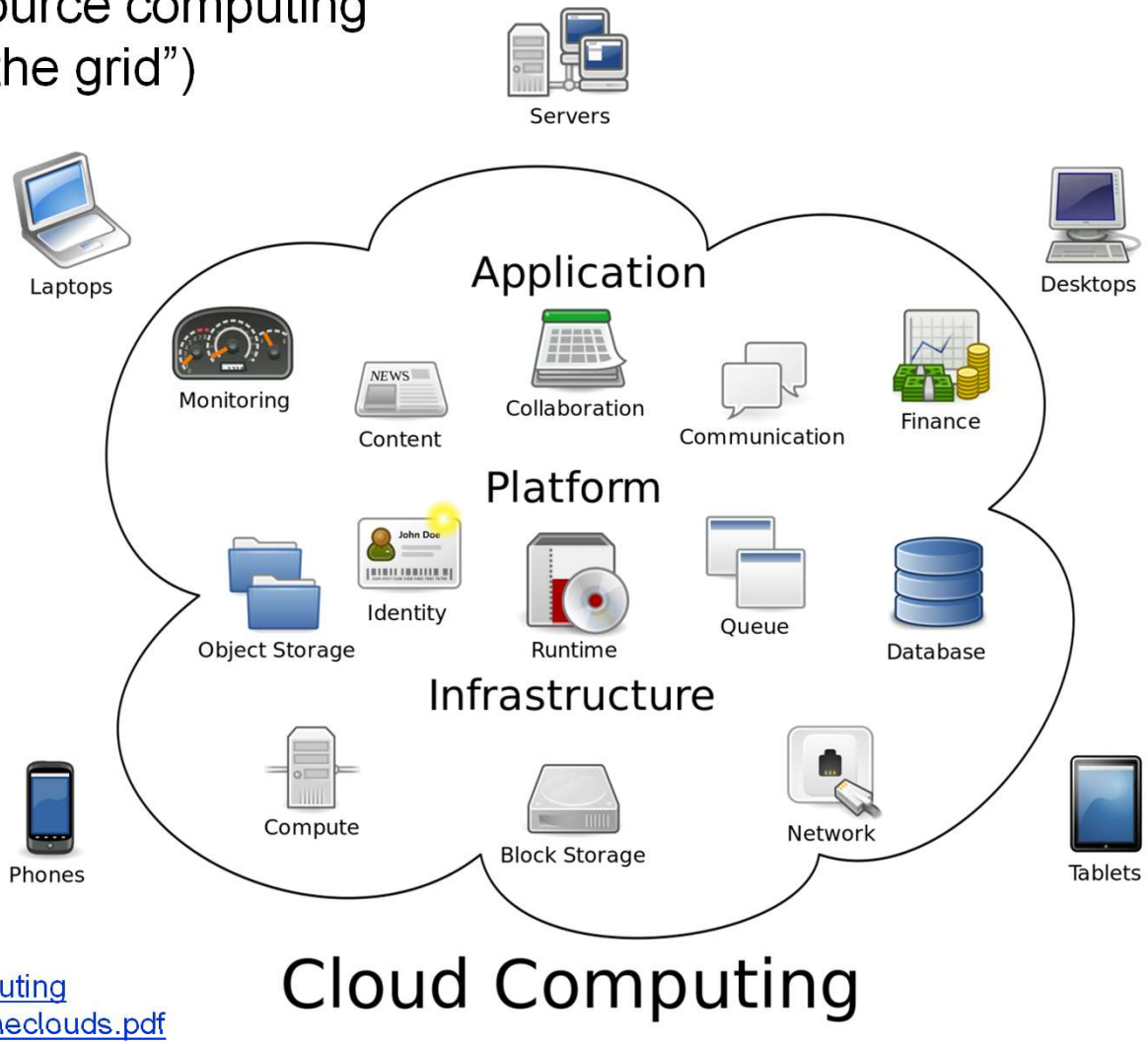


Majority

Cloud Computing Overview

On-demand, shared-resource computing
(formerly referred to as “the grid”)

“The fifth utility”



Sources

https://en.wikipedia.org/wiki/Cloud_computing
http://www.moonther.com/cis492/abovetheclouds.pdf

Reliability in Cloud Computing

Theoretically, the cloud can be highly reliable (no single point of failure)
- Challenge of integrating diverse resources from multiple providers
- Identifying weakest links; assigning blame in the event of outages
- Risk assessment in e-commerce and safety-critical processing

Planning for and modeling outages
- Accidental or deliberate

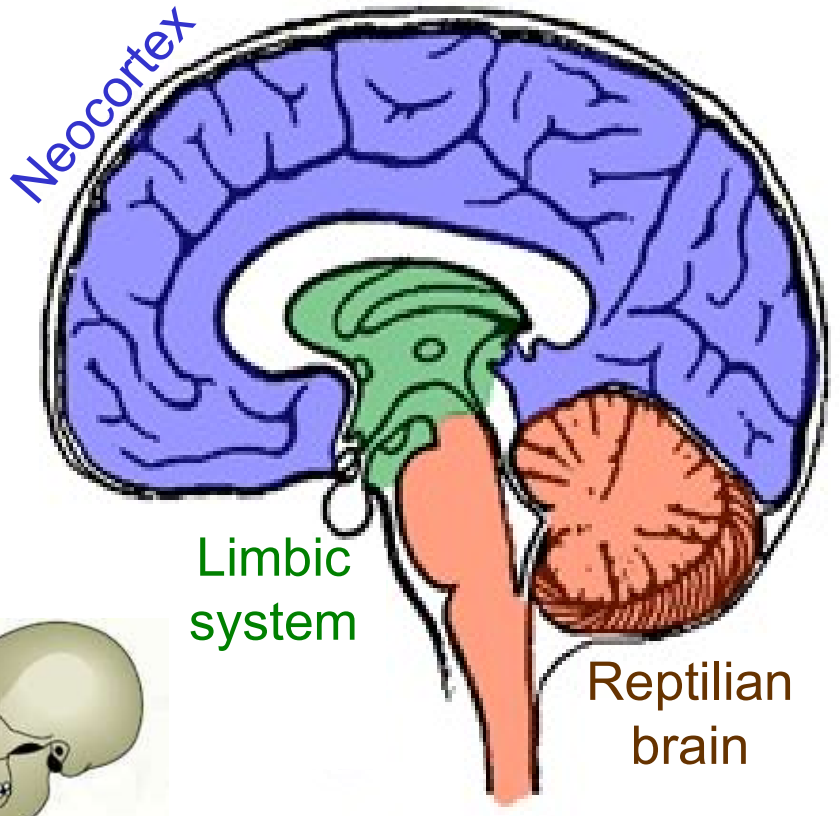
Balance and accountability
- Recall Amdahl's law of reliability

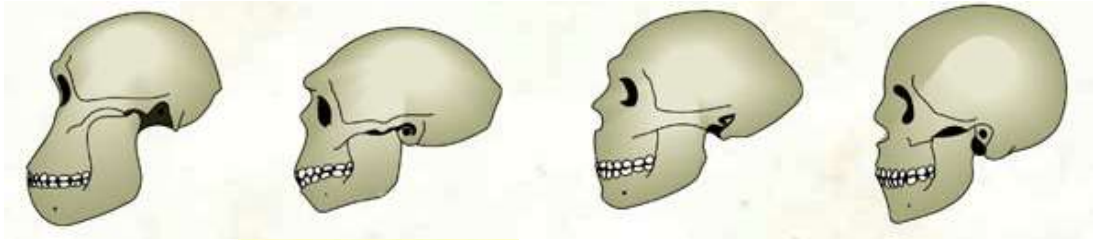
Risk assessment

Smarter and Lower-Power Redundancy Schemes

Brain-inspired computing

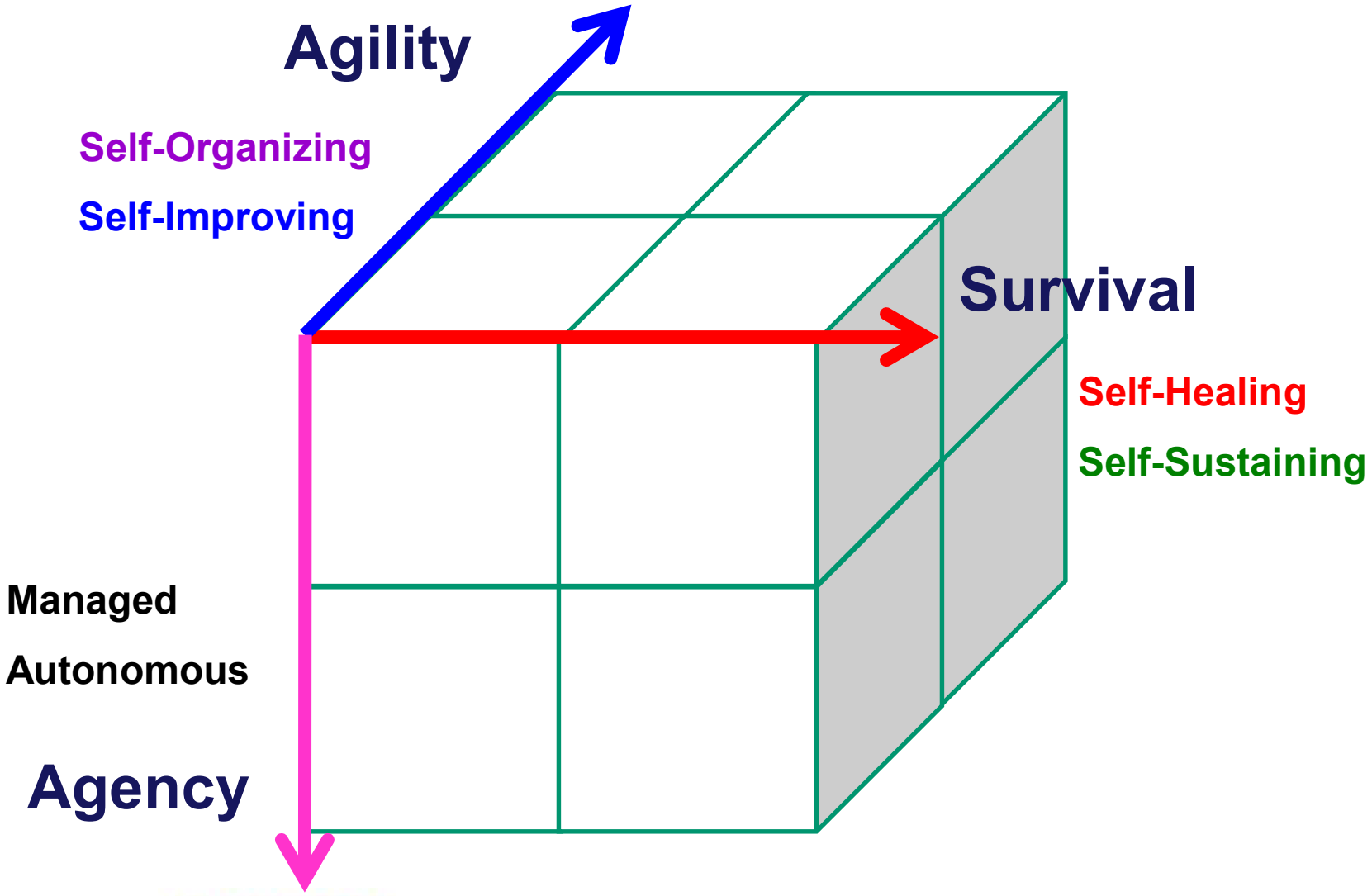
- Human brain has a very inelegant design, but it operates on 20 W (a supercomputer doing a small fraction of the brain's tasks needs MWs)
- Ad-hoc additions over eons
- Functional redundancy
- Layered improvement
- Electrochemical signals
- Slow signaling
- Yet, functionally effective!





Engineering the Future: System Attributes

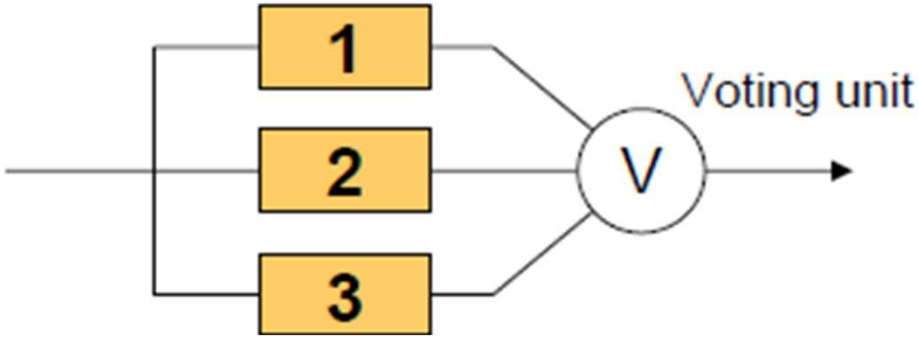
Toward **Self-Organizing**, **Self-Improving**, **Self-Healing**, and **Self-Sustaining** Systems



Example: Learning Voter

Suppose the voter/fuser keeps track of disagreements by units 1-3 so that it learns which units are more trustworthy

If unit 1, say, disagrees often, then the first disagreement between units 2 and 3 may be cause for the system to shut itself down



Ensuring Data Longevity

Typical media used for storing data have lifespans of 3-20 years

We can lose data to both media decay and format obsolescence

We're not yet aware of, or sensitive to,
decay and obsolescence,
because we move data
to newer devices every few years
as part of routine upgrades

Reference:
Springer Encyclopedia of Big Data Technologies, Article on "Data Longevity and Compatibility," 2019

Reasoning About Uncertainty

Uncertainty can exist in data (facts)
- Data not available
- Data unreliable (measurement / representation errors, estimation

Uncertainty can exist in rules (knowledge)
- Rules may be empirical
- Rules may be based on typical or average cases

Reasoning about uncertainty requires
- Representing uncertain data or rules
- Combining two or more items uncertain data items
- Drawing inferences based on uncertain data and rules

Categories of approaches
- Probabilistic analysis (Bayesian logic)
- Certainty/Confidence factors
- Evidence/Belief theory (Dempster-Shafer theory)
- Continuous truth values (Fuzzy logic)

Reference:
Int'l J. Approx. Reasoning, Special Issue 40 Years of Research Dempster-Shafer Theory, December 2016

Interaction Between Failures and Intrusions

Intruders can cause failures as a way of disrupting systems
- Self-driving cars
- Fly by wire
- Electronic terrorism

Failures may facilitate security breaches

Overcoming the Reliability Wall

We have known about memory wall limiting performance for some time

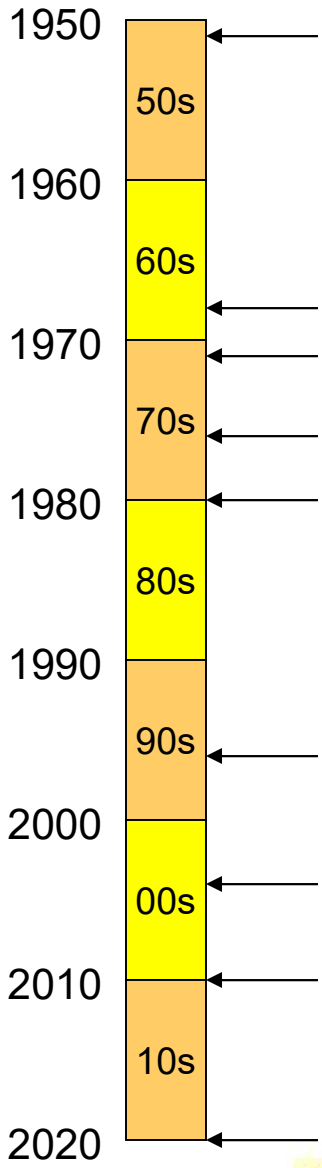
Reliability wall limits the scalability of parallel applications

Yang et al's two case studies [Yang12]
- Intrepid supercomputer
- ASCI White supercomputer

The notion of reliability wall is introduced in Problem 3.17
- Amdahl's constant-task speed-up formula: $s = p/[1 + f(p - 1)]$
  - Speed-up is upper-bounded by $1/f$
- Gustafson's constant-running-time speed-up formula: $s = f + p(1 - f)$
  - Speed-up is unbounded for any $f < 1$

If the reliability overhead is a superlinear function of p, then a reliability wall exists under Amdahl's interpretation but not under Gustafson's

Dependable Computing Through the Decades



SAPO, the first fault-tolerant computer, built in Czechoslovakia

JPL STAR Computer designed and built for deep space missions

First FTCS held in Pasadena, CA

Tandem NonStop multiprocessor announced

IFIP WG 10.4 on Dependable Computing and Fault Tolerance formed

25th FTCS held in Pasadena, CA

***IEEE Trans. Dependable and Secure Systems* started publication**

40th DSN held in Chicago

This timeline to be completed

50th DSN held in Valencia, Spain

Resources for Dependable Computing

IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance
IFIP WG 10.4 — http://www.dependability.org/wg10.4/

IEEE/IFIP Int'l Conf. Dependable Syst's and Networks (2021, Taipei)
DSN — http://www.dsn.org/

European Dependable Computing Conf. (2020, Munich)
EDCC — http://edcc.dependability.org/

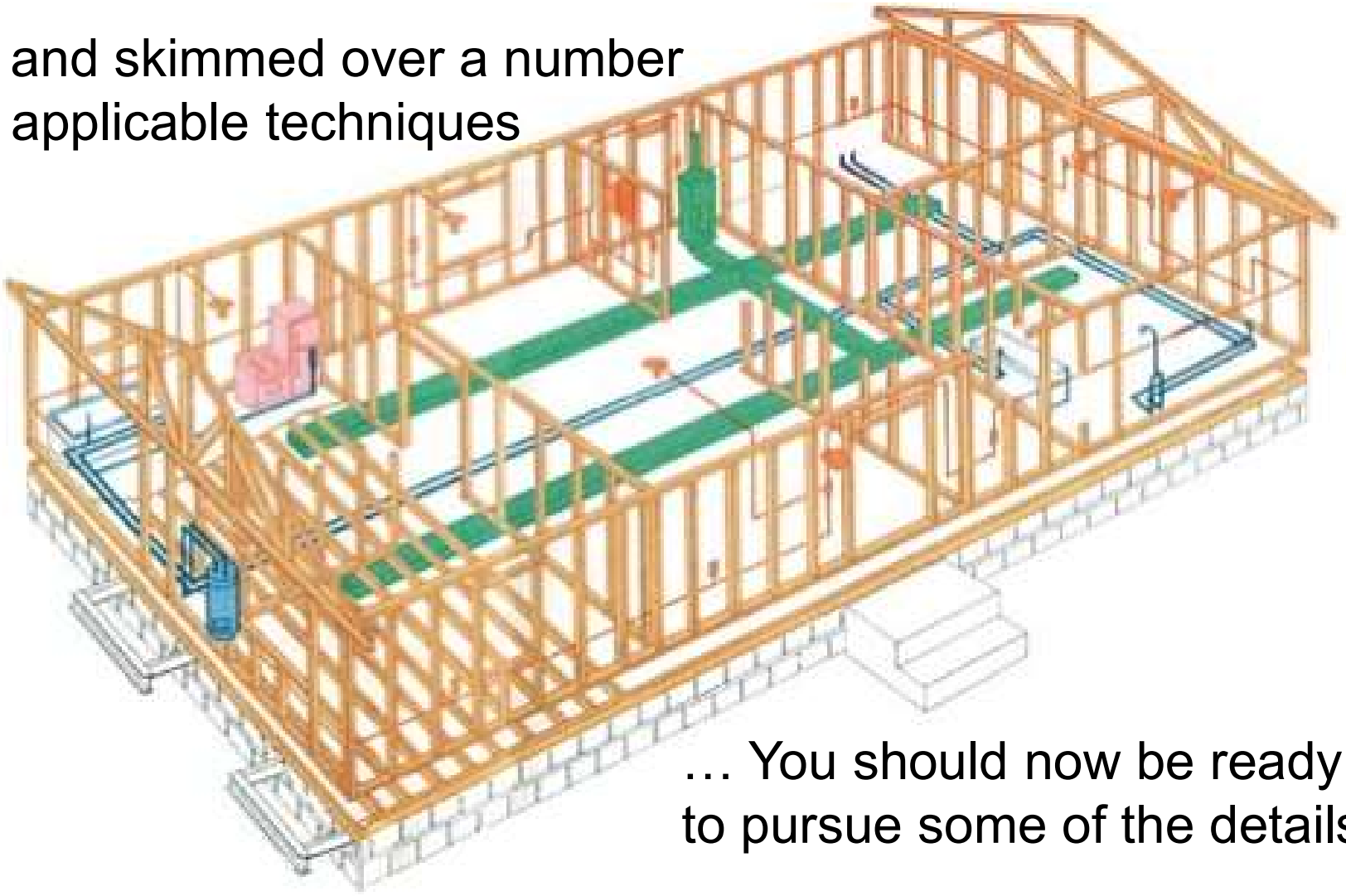
IEEE Pacific Rim Int'l Symp. Dependable Computing (2021, Perth)
PRDC — http://prdc.dependability.org/

Int'l Conf. Computer Safety, Reliability and Security (2021, York, UK)
SAFECOMP — http://www.safecomp.org/

IEEE Trans. Dependable and Secure Computing (since 2004)
IEEE TDSC — https://www.computer.org/csdl/journal/tq

We Have Built a Framework

... and skimmed over a number of applicable techniques



… You should now be ready to pursue some of the details