Reliability in real-time systems

Examples and basic principles

Overview

- Real-Time Systems need to be reliable!
- In this slide set, we will talk about some of the techniques to make fault-tolerant systems (this is a pre-requisite to making a safe system).
- Rather than focus on the large amount of theory in this area, we'll emphasize a few examples:
  - Therac-25
  - CANDU Reactors
  - Space Shuttle
  - Modern Passenger Jets

What should you learn?

- What is the difference between reliability, security and safety?
- What are the steps from the time an error occurs to when a system fails?
- What are some of the causes of errors?
- What are some of the approaches to fault tolerance?
  - What are the differences between hardware and software schemes?

FT.com

Financial Times: March 31, 2005

Mercedes recalls 1.3m cars for quality issues
>By James Mackintosh in London
>Published: March 31 2005 17:49 | Last updated: March 31 2005 17:49

Mercedes-Benz recalled 1 in 3 of the cars it produced in the past 4 years to fix electronic problems…

Mercedes, owned by DaimlerChrysler, has seen profits plummet as a 1.2bn provision last year for the costs of fixing problems with cars already sold added to pressure from the weak dollar and losses at its Smart small car operations. In the final quarter of last year Mercedes profits dropped 97 per cent to 20m, and Eckhard Cordes, head of the division, said this

…Final quarter of last year, Mercedes's profits dropped 97%…

part of a

… sharp rise in breakdowns was due to the failure of the complex electronics in its cars…

The loss of Mercedes' coveted position near the top of quality measures has added to pressure on sales from the ageing of several models. But the launching of new cars and offroaders and a 3bn cost-saving and revenue plan is designed to get the company back to 7 per cent profit margins in two years.

…it is extremely rare to recall more cars than a company builds in a year.

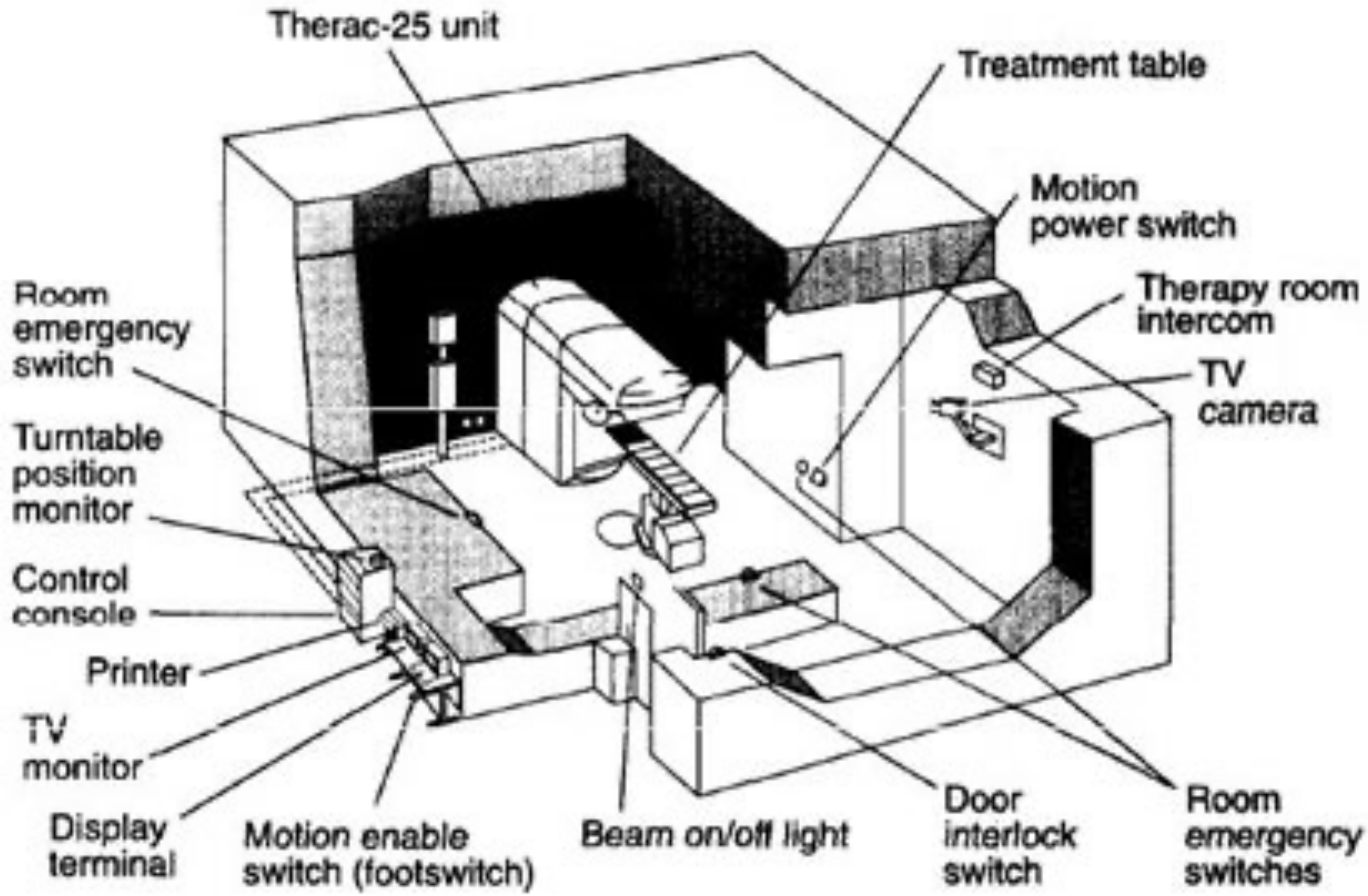
Mercedes declined to say how much the recall would cost, but analysts said it was likely to be covered by the extra warranty provisions taken last year.

Large recalls are becoming increasingly common at big carmakers as regulators and customers become more demanding. But it is extremely rare to recall more cars than a company builds in a year.

Motivating example: Therac-25

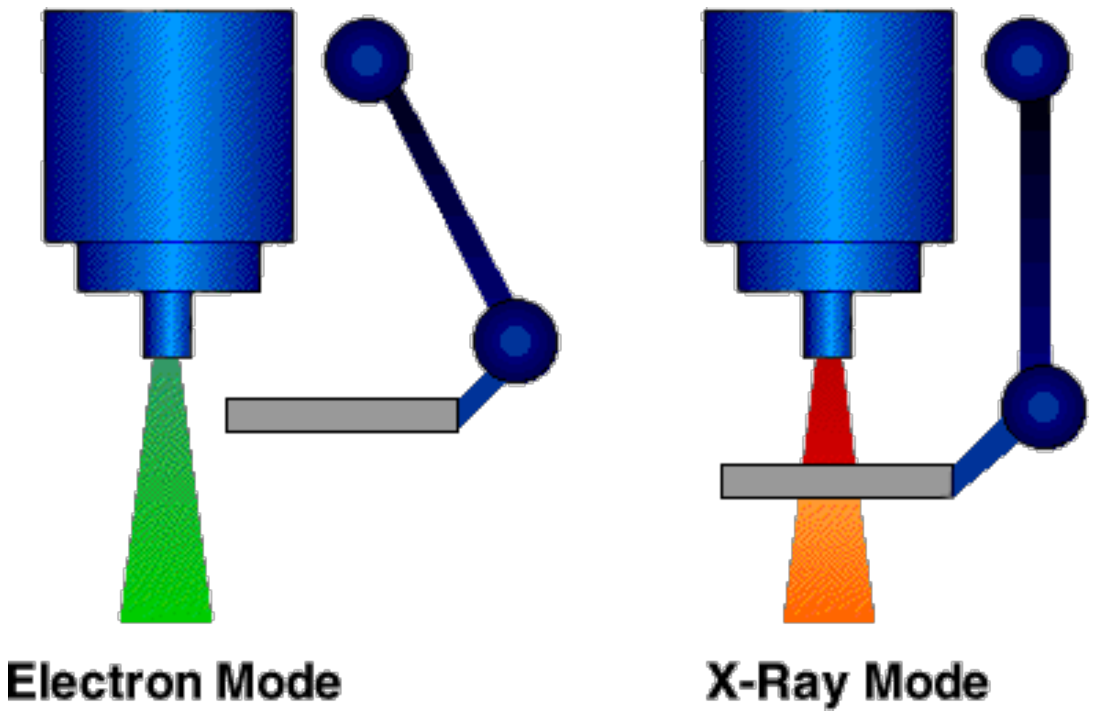
- Medical linear accelerator
  - Used to treat tumors with either:
    - Electron beams for shallow tissue
    - X-Ray beams for deep tissue
- Eleven Therac-25s were installed
  - Six in Canada
  - Five in the United States

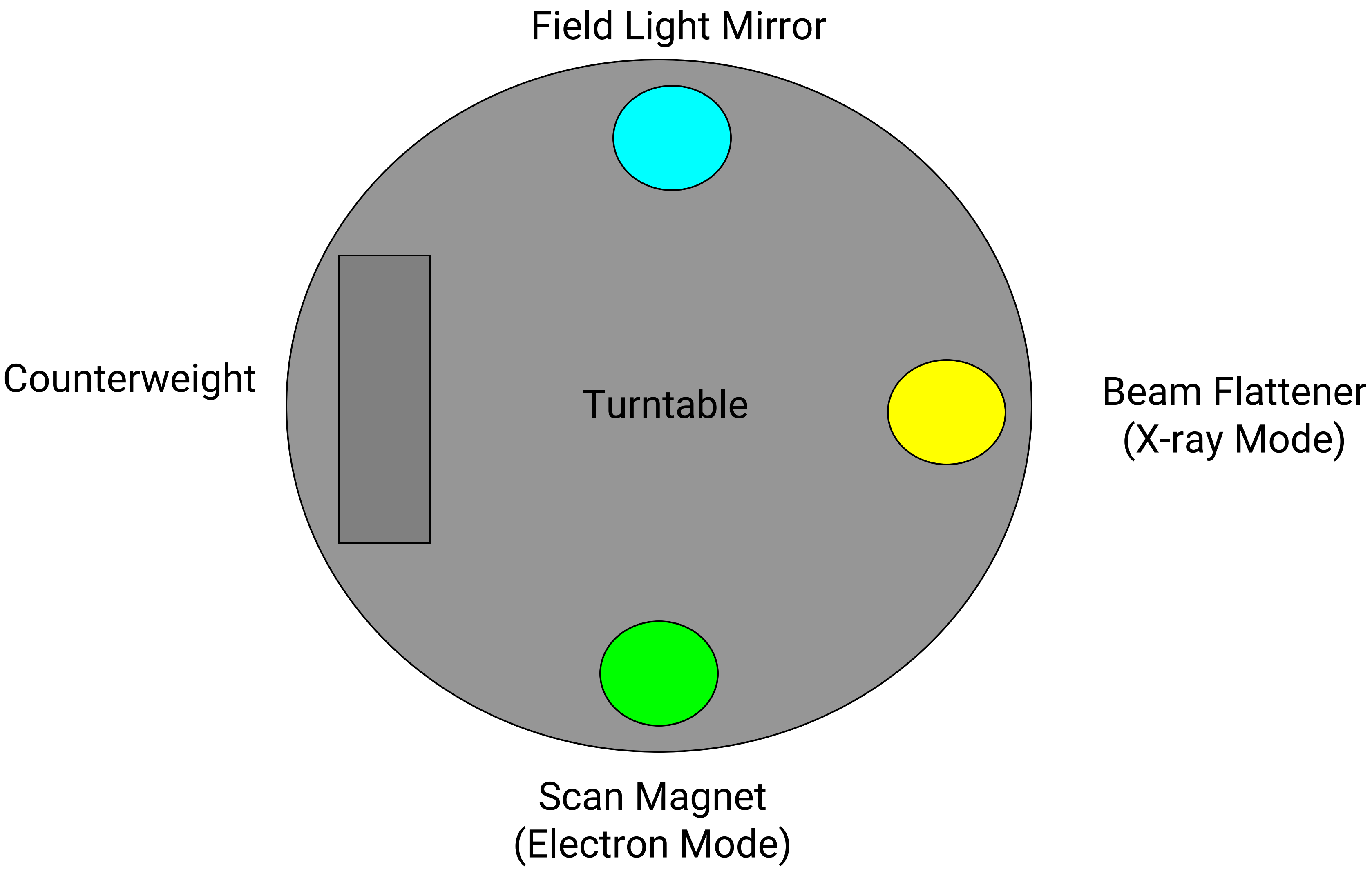
- Developed by Atomic Energy of Canada Limited (AECL).


Therac-25 unit
Treatment table
Motion power switch
Therapy room intercom
TV camera
Room emergency switch
Turntable position monitor
Control console
Printer
TV monitor
Display terminal
Motion enable switch (footswitch)
Beam on/off light
Door interlock switch
Room emergency switches

Therac-25

- Could deliver either electron therapy or X-ray therapy





Source: W. Leubke, 2003

- Six patients were delivered severe overdoses of radiation between 1985 and 1987.
  - Four of these patients died.
- Why?
  - The turntable was in the wrong position.
  - Patients were receiving x-rays without beam-scattering.

Therac-25

- How could this happen?
  - Race conditions in the software
  - Multiple threads did not lock variables properly
- Overflow error.
  - The turntable position was not checked every 256th time a certain variable was incremented.
- No hardware safety interlocks.
- User interface errors, and wrong information on console.
- Non-descriptive error messages.
  - “Malfunction 54”
  - “H-tilt”
- Too easy to just hit “P” (Proceed)

Patriot Missile 1991

A report of the General Accounting office, entitled *Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia,* begins:

> *"On February 25, 1991, a Patriot missile defense system operating at Dhahran, Saudi Arabia, during Operation Desert Storm <u>failed to track and intercept an incoming Scud</u>. This Scud subsequently hit an Army barracks, killing* ***28*** *Americans"*

Patriot Missile 1991 Failure Root Cause

- Cause was an inaccurate calculation of the time since boot due to computer arithmetic errors
  - Time is measured by the system's internal clock in tenths of second (stored as an integer) and then multiplied by 1/10 to produce the time in seconds
  - This calculation was performed using a 24 bit fixed-point register. The value 1/10 has a non-terminating binary expansion and was chopped at 24 bits after the radix point
  - The small chopping error, when multiplied by the large number giving the time in tenths of a second, led to a significant error
  - The Patriot battery had been up around 100 hours, and the resulting time error due to the magnified chopping error was about 0.34 seconds
  - The range gate's prediction of where the Scud will next appear is a function of the Scud's known velocity and the time of the last radar detection
  - A Scud travels at about 1,676 m/s and so travels more than half a **km** in this time
  - This was far enough that the incoming Scud was outside the "range gate" that the Patriot tracked

http://www-users.math.umn.edu/~arnold/disasters/patriot.html

Some definitions

- Security: A measure of confidence that the system can resist attempts to modify its behavior.

- Reliability: A measure of confidence that the system produces accurate and consistent results **over a specific period of time**

- Safety: A measure of confidence that the system will not cause accidents/harm
  - Absence of unreasonable risk due to hazards resulting from malfunctioning of the E/E, insufficiencies of the intended functionality (not necessarily failure of the function), or by reasonably foreseeable misuse by persons

- Is reliability either sufficient or necessary for safety?

Note that safety and reliability can be in conflict.

"The safest plane is one that never leaves the ground".

Good engineering always involves a tradeoff between safety and reliability.

Reliability Engineering

The objectives of reliability engineering are:

- To apply engineering knowledge and specialist techniques to **prevent** or to reduce the likelihood or frequency of failures.
- To identify and **correct the causes** of failures that do occur despite the efforts to prevent them.
- To determine ways of **coping** with failures that do occur, if their causes have not been corrected.
- To apply methods for estimating the likely reliability of new designs, and for analysing reliability data.

What should we be worried about?

- Computers are composed of hardware, software, and data
  - The software defines the operations
  - The hardware performs the operations
  - The data records the results of the operations
- We *have* to worry about all three of these

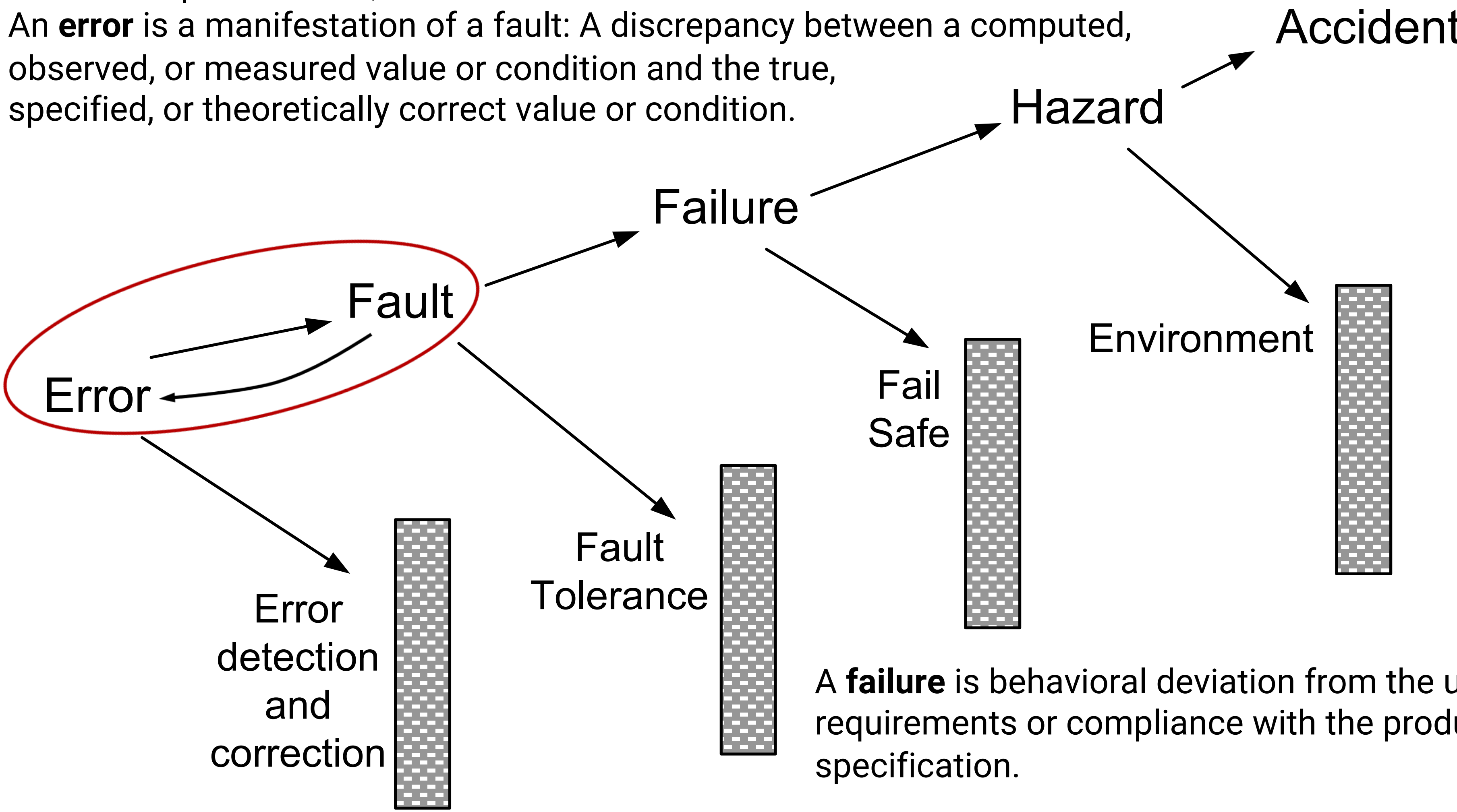
	Hardware	Software	Data
Cause of Failure	Deficiencies in design, production or maintenance	Design (logic) errors	Transient Events
Occurrences	Will eventually fail	Might never fail	Might never fail
Failure Rates	Can be predicted in theory from physical principles	Can **not** be predicted from physical principles	Some upset rates can be predicted from test
Redundancy	Will improve reliability, but might be susceptible to <u>common cause failures</u>	Will not improve reliability, since this will only replicate same failure	Might improve reliability
Diversity	Likely to improve reliability, should be less susceptible to common cause failures	Likely to improve reliability since it minimizes possibility of same error occurring in separate modules	Likely to improve reliability since it minimizes possibility of same error occurring in separate modules

Modified from A. Tribble, 23rd Digital Avionics System Conference

	Hardware	Software	Data
Environmental Factors	Dependent on temperature, humidity, stress, etc.	Dependent on internal environment of computer (memory, clock speed, etc.)	Dependent on both external (radiation, EMI, etc.) and internal environment (memory, clock speed, etc.)
Time Dependence	Is time dependent. Failure rates can be increasing, constant or decreasing	Not time dependent. Failures occur when path that contains fault is executed	Is time dependent. Failure rates can be increasing, constant or decreasing
Wear-Out	Responsible for some failures. Could be preceded by a warning.	Not responsible for any failures	Not responsible for any failures
Preventative Maintenance	Can improve reliability	Will not improve reliability, and might actually **worsen** it	Will not improve reliability

Modified from A. Tribble, 23rd Digital Avionics System Conference

A **fault** is a passive flaw;
An **error** is a manifestation of a fault: A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.



A **failure** is behavioral deviation from the user requirements or compliance with the product specification.

A **failure** is an observable effect <u>outside the system boundary</u> arising from an internal error or fault

An error or fault does **not** always lead to a failure

Modified from A. Tribble, 23rd Digital Avionics System Conference

Examples

- A programmer types `char x[11]` to define a variable when `char x[10]` was intended
- Is this a fault, error, or failure?
- This is fault, until the system becomes really short of memory
  - Becomes an error then

- A programmer created a system that opened a file every 10 seconds, but failed to close those files
- What is the fault, error, failure? When do they happen?
- Fault: not closing open files → causes an error (file descriptor leakage) every 10 seconds
- When would this error cause a failure?
  - When the allowed # of open file descriptors in the OS is reached

Faults vs. Errors vs. Failures

- The Patriot Missile battery failure:
  - **Fault**: not programming correctly the time since boot
  - **Error:** time truncation error that led to Scud distance prediction error that is not tolerable (might have been tolerable for shorter duration since boot)
  - **Failure:** incorrectly track and intercept an incoming Scud missile

- **Faults do not always result in failures**. They might be occurring without even knowing about them (Fault might stay dormant for a long time before it manifests as an error) but failures might happen depending on the conditions in which the fault occurred
  - memory bit got stuck but CPU does not access this data

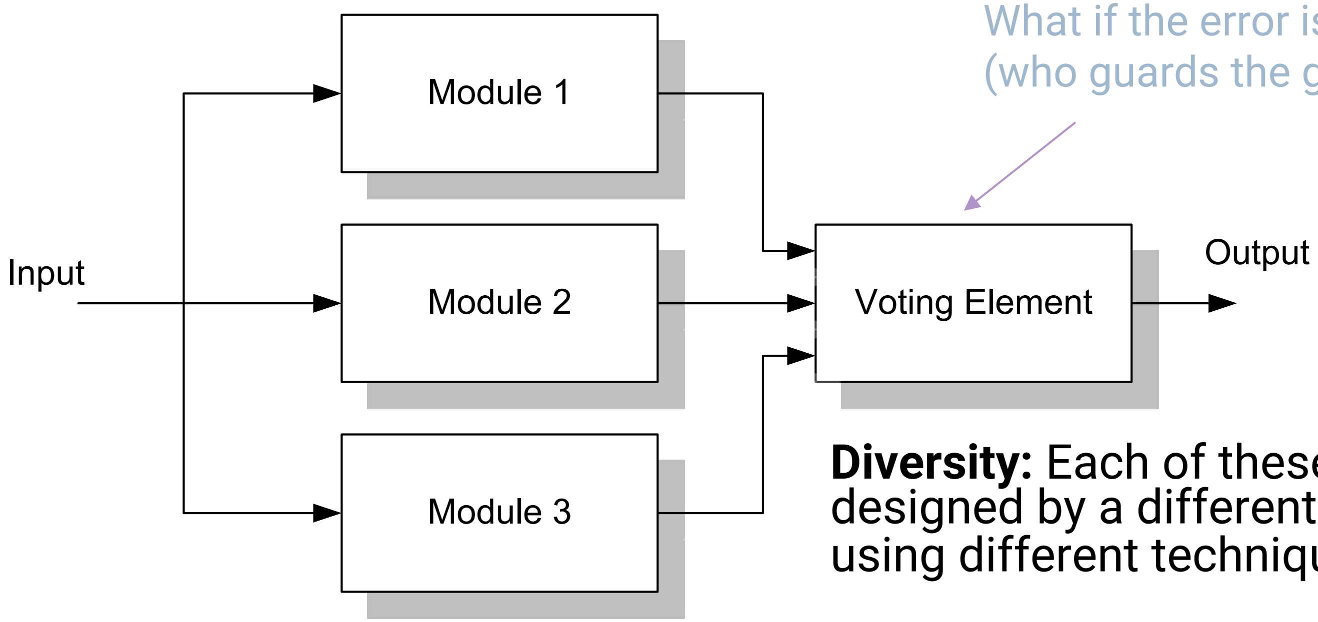
- **Not all errors cause failures**. In AI algorithms there’s an error due to approximation but it is within tolerable limits that it does not cause failures.

Fault Tolerance

The ability of a functional unit to continue to perform a required function in the presence of faults or errors

Fault Tolerance – Relies on Redundancy

- Triple Modular Redundancy (TMR) [**passive** redundancy]



What if the error is in the voting unit? (who guards the guard?)

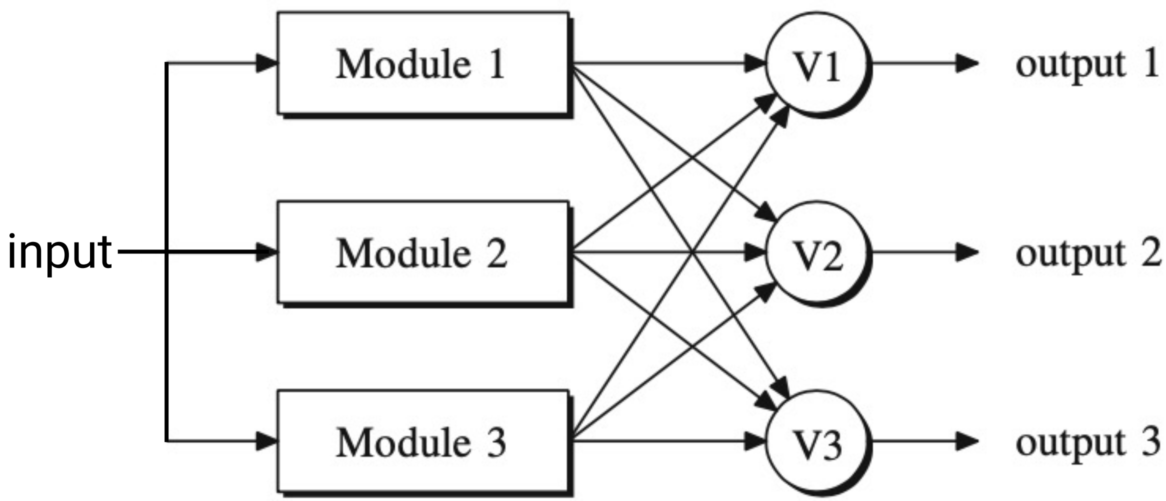
Diversity: Each of these blocks could be designed by a different design team using different techniques.

- Hardware Redundancy
- Software Redundancy
- Information Redundancy
- Temporal (Time) Redundancy

Example: Used in the logic section of the launch vehicle digital computer (LVDC) of Saturn 5. Saturn 5 is a rocket carrying Apollo space crafts to the orbit. Reliability of the logic section for a 250-hr mission is approx. 20x larger than the reliability of an equivalent simplex system.

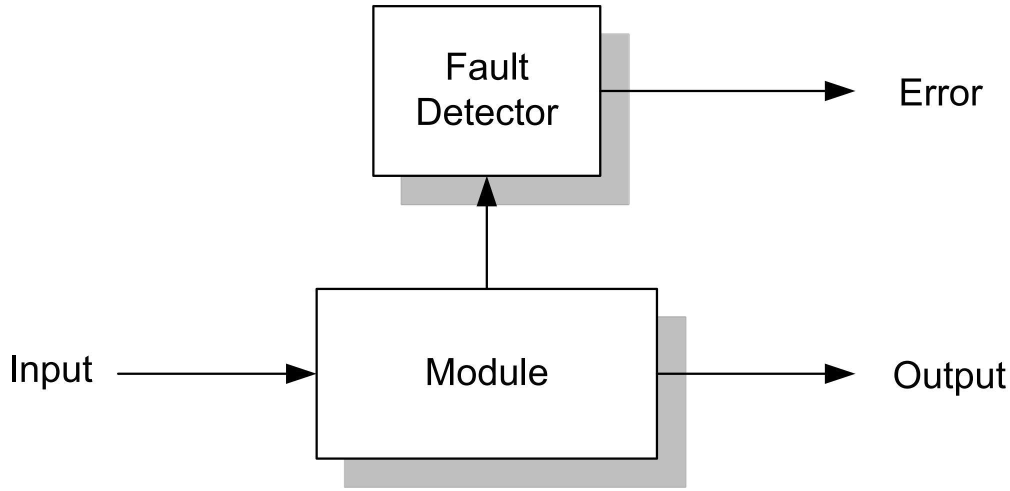
TMR with triplicate voters

- Can replicate voters if single voter is imperfect
- Removes **single point of failure** (simplex voter)
- Can tolerate two module failures + 1 voter failure



> Requires **consensus** to be established among three voters to decide final output
> In order to reach consensus, the voters typically exchange several rounds of messages

Detecting Faults (Errors)



- For some applications, this might be OK.
- But … how do we detect a fault?

Detecting Faults (Errors)

1. Functionality Checks:
   - Periodically execute code and check results
   - For example, write to then read from RAM
2. Consistency Checking:
   - Example: Range checking
3. Signal Comparison:
   - In some systems, you can compare signals at various points within a module
4. **Information Redundancy:**
   - Checksums, CRC, parity checking

Detecting Faults (Errors)

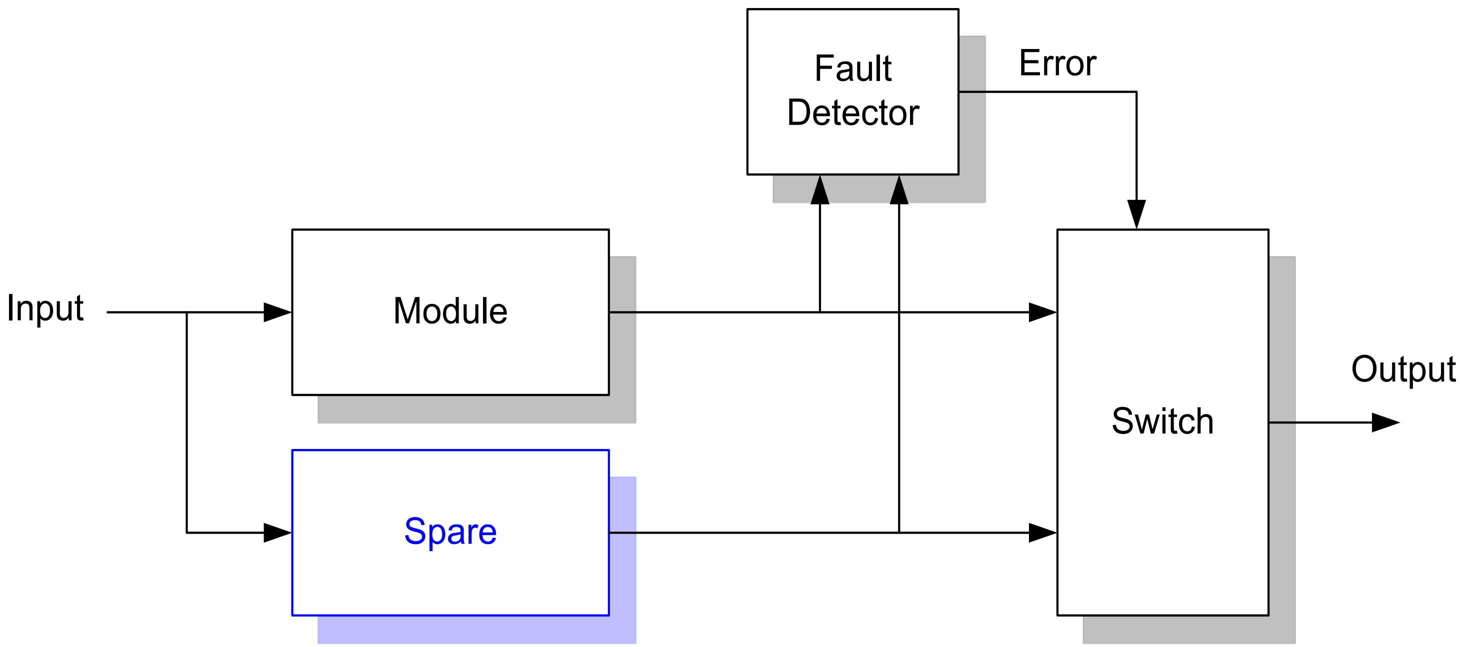
5. Instruction Monitoring:
   - If the processor fetches an illegal instruction, something is probably wrong
6. Loop-back Testing:
   - Useful for testing communication channels. Make sure what is received is the same as what is sent
7. Bus Monitoring:
   - Watch the bus and make sure that the program accesses memory within an allowable range
8. Power Supply Monitoring:
   - Possibly a dead system will draw less power
   - Also, the power supply might fail: this would cause major system failure. There might be a warning you could watch for

Detecting Faults (Errors)

- 9. Watchdog Timers:
  - Detect the crash of a microprocessor by arranging a timer such that it will cause a reset (or error condition) if it is allowed to time-out
  - While the processor is operating normally, it periodically loads a value into this register
  - **Problem:** Time delay until fault is detected
  - **Problem:** It is conceivable that the system could crash in such a way that the timer is still loaded with a value, or it is possible that the watchdog timer fails

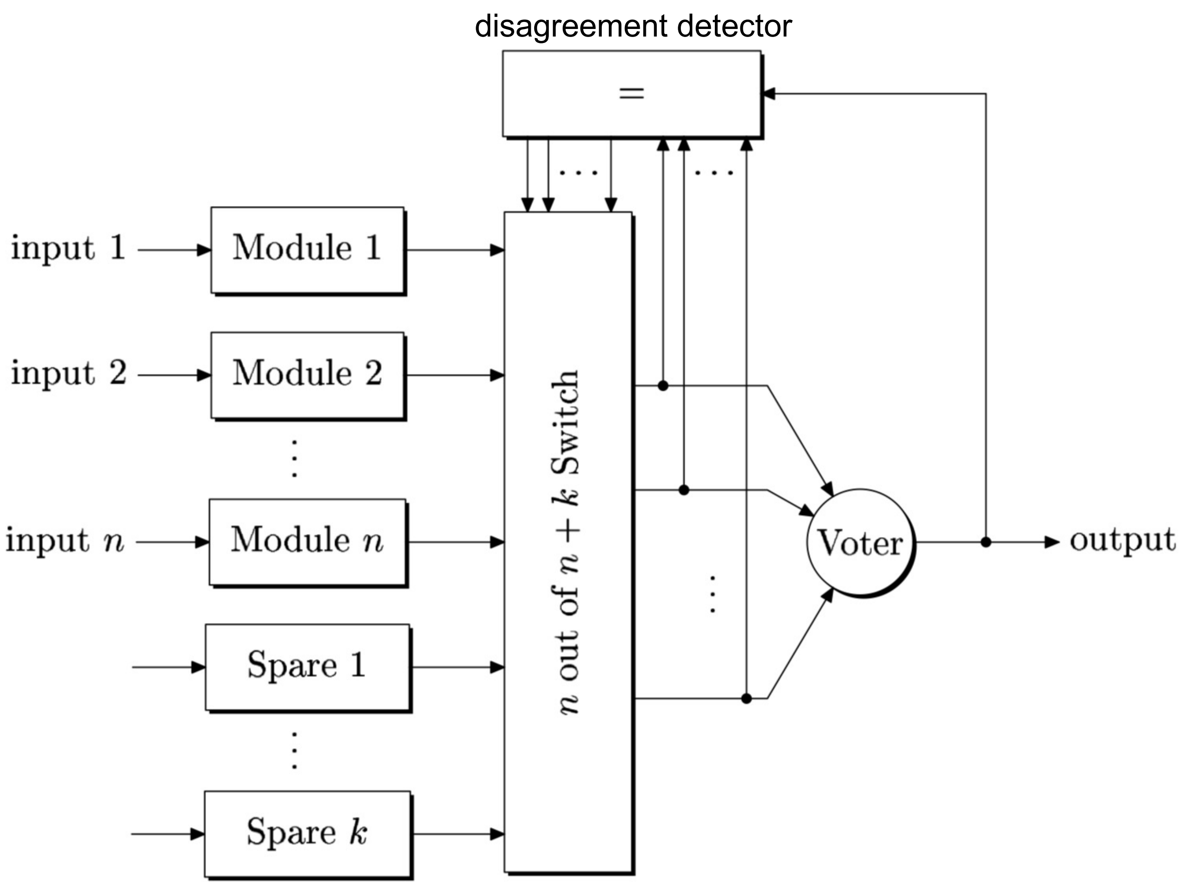
Standby Spares (Active redundancy)

- When an error is detected, switch in a spare:



- **Hot Standby:** during normal operation, run the spare in parallel with the active unit. Allows for a fast transfer of control with minimum of delay
- Used in systems where infrequent, occasional errors are allowed, as long as the system recovers back to normal operation in a specified interval of time.

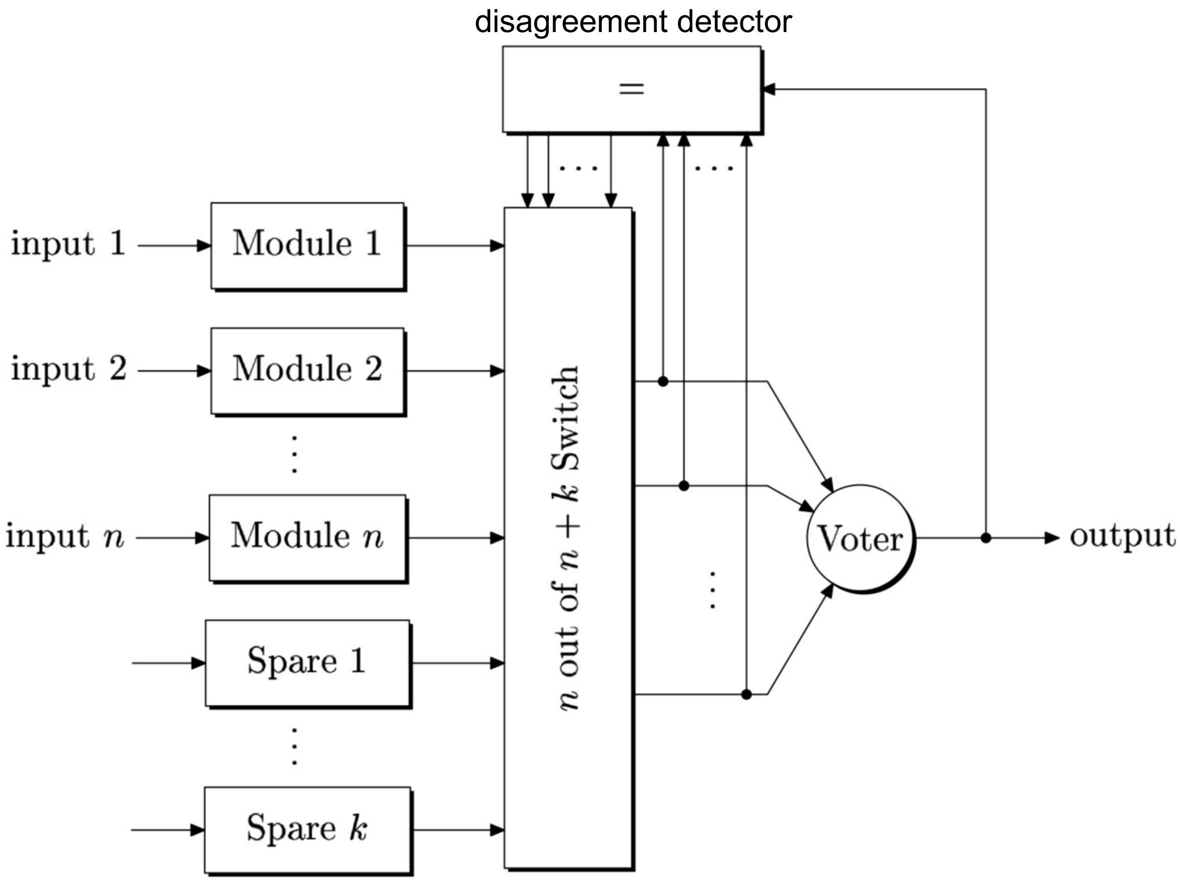
N-Modular Redundancy with Spares



Normal Operation: Use first *n* modules

- output of the voter is compared to the individual outputs of modules
- module which disagrees is labeled as faulty and removed from the NMR core
- spare is switched to replace it

N-Modular Redundancy with Spares



Note: fault tolerance capability depends on voter implementation (fixed input vs variable input [threshold])

Fixed input voter:

- Initially *k* faults are tolerated by means of the *k* spares
- After k faults, disagreement detector turned off, tolerating extra (n-1)/2 faults (becomes regular NMR)

Software Fault-Tolerant Techniques

- Main difference between S/W and H/W fault tolerance
  - Simply replicating code and executing it 3 times doesn't help
  - If there is a bug in the code, it will happen each time

- Thus, with software, it really only makes sense if you have *different* implementations of the module [diversity]

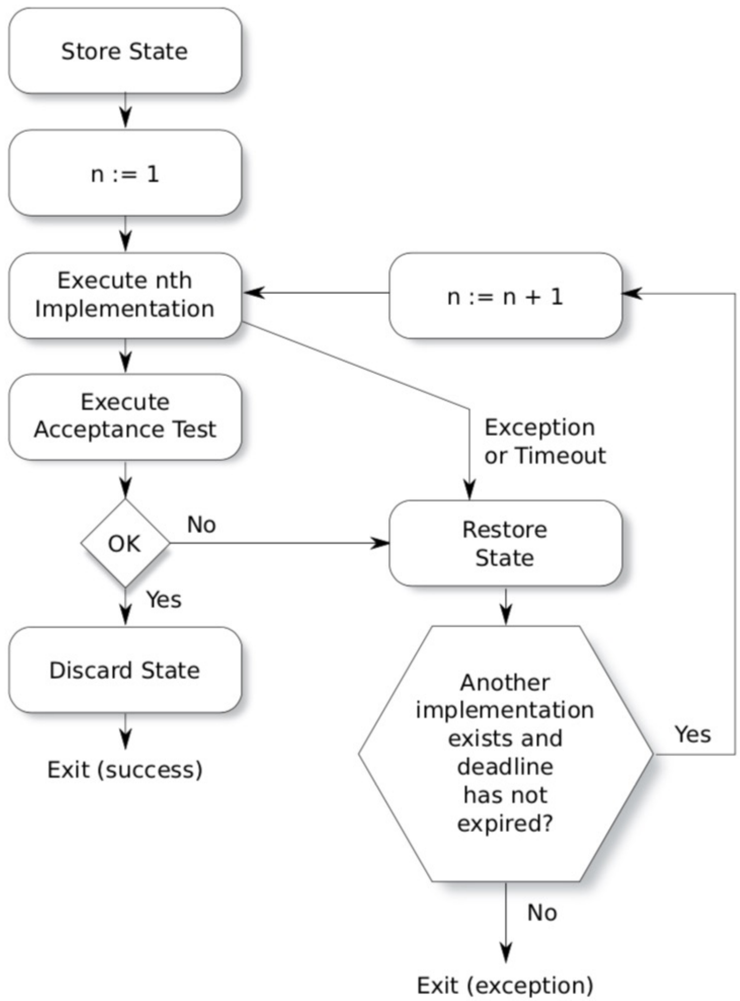
- *N*-Versions programming: have *N* different versions of the software, and execute all *N* versions
  - Significant Run-time overhead
  - Development time/cost overhead

Handling Software Errors: **Recovery Blocks**

Problem: primary module might have changed the state of the system → need to *checkpoint* at the start so we can "roll back" system state

```
primary module
acceptance test
if (acceptance test failed) {
    secondary module   //different implementation, simpler
    acceptance test
    if (acceptance test failed) give up   //prepare for failure
                                            in as controlled a
                                            manner as possible
}
```

Recovery Block Structure



From Hobbs, Embedded Software Development for Safety-Critical Systems

Data Errors

- Even if the hardware and software is fine, data can be corrupted:
- One mechanism: Single-Event Upset faults
  - Radiation continuously strikes earth
  - It is possible that a bit can be flipped
- Flipping a bit can cause memory errors, or if you are using reconfigurable logic, it can even cause circuit/processor errors

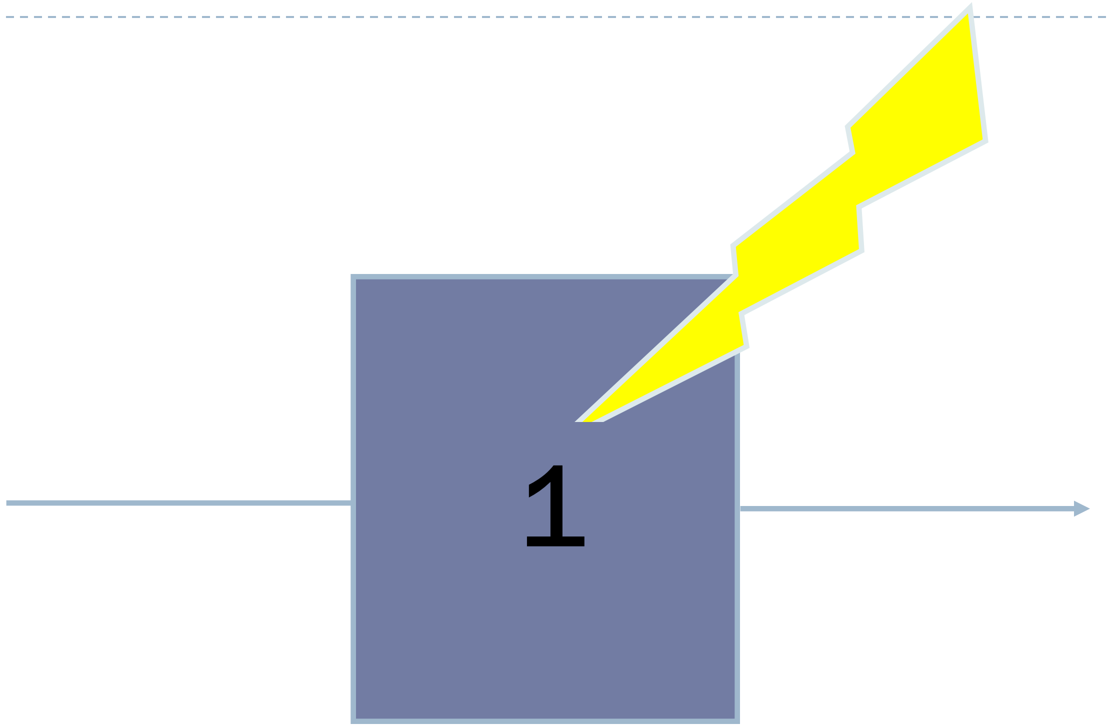
Does it really happen?

- Documented strikes in large servers found in error logs
  - Normand, “Single Event Upset at Ground Level,” IEEE Transactions on Nuclear Science, Vol. 43, No. 6, December 1996.

- Sun Microsystems, 2000 (R. Baumann, Workshop talk)
  - Cosmic ray strikes on L2 cache with defective error protection
    - caused Sun’s flagship servers to suddenly and mysteriously crash!
  - Companies affected
    - Bell, America Online, Ebay, & dozens of other corporations

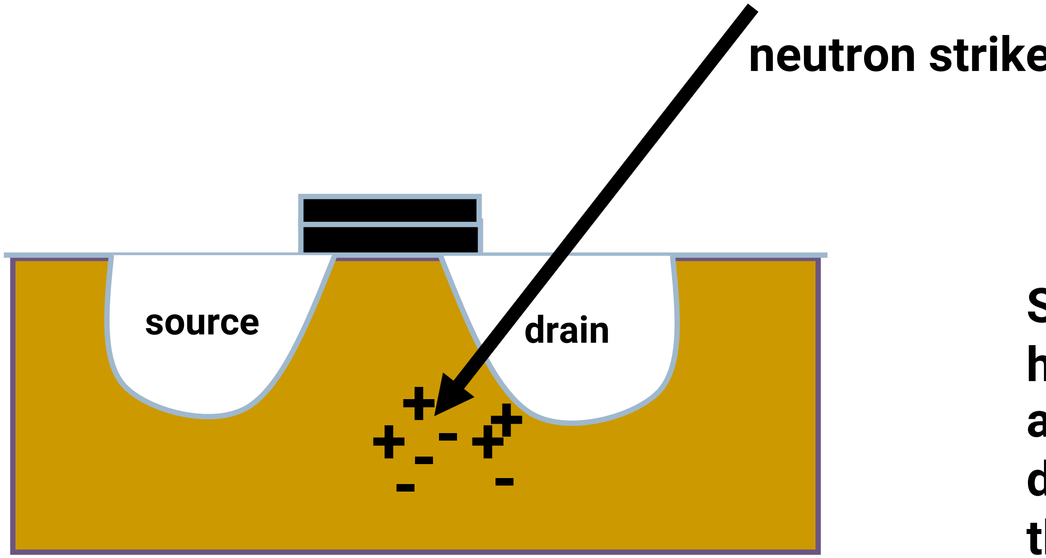
From: S. Mukherjee, Intel, HPCA 2005

Strike Changes State of a Single Bit



From: S. Mukherjee, Intel, HPCA 2005

Impact of Neutron Strike on a Si Device

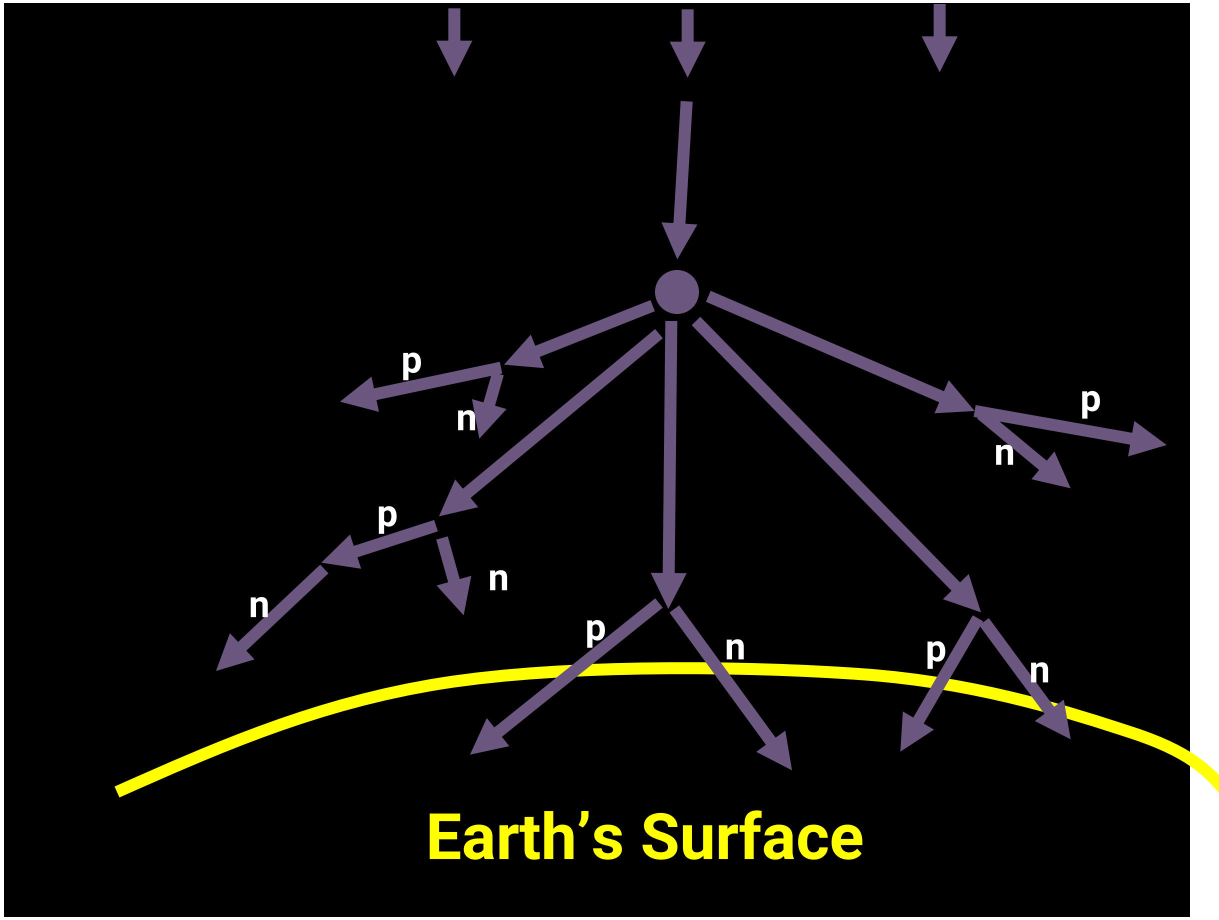


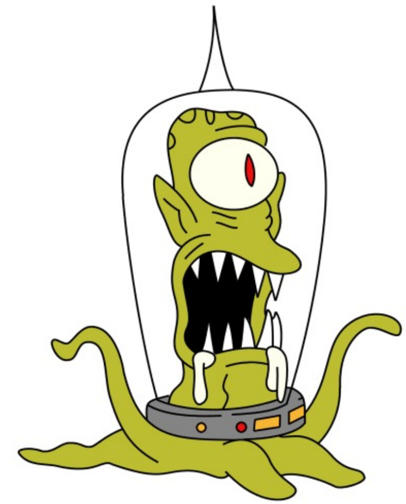
Strikes release electron & hole pairs that can be absorbed by source & drain to alter the state of the device

Secondary source of upsets: alpha particles from packaging

From: S. Mukherjee, Intel, HPCA 2005

Cosmic Rays Come From Deep Space





Neutron flux is higher in higher altitudes

From: S. Mukherjee, Intel, HPCA 2005

Impact of Elevation

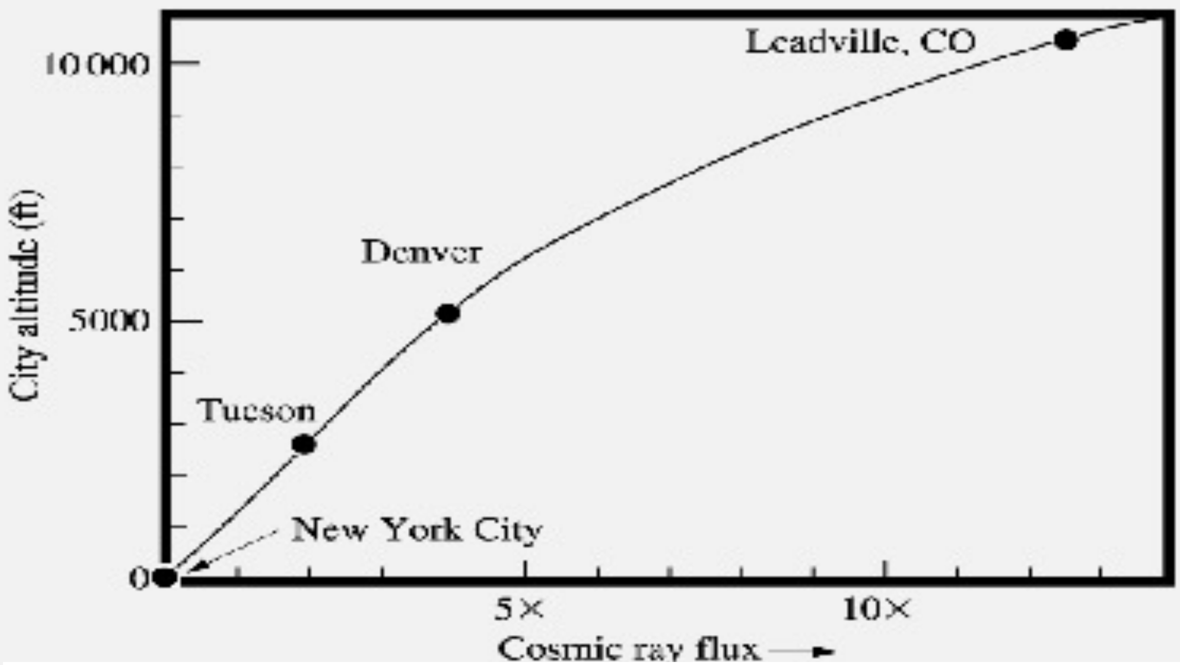


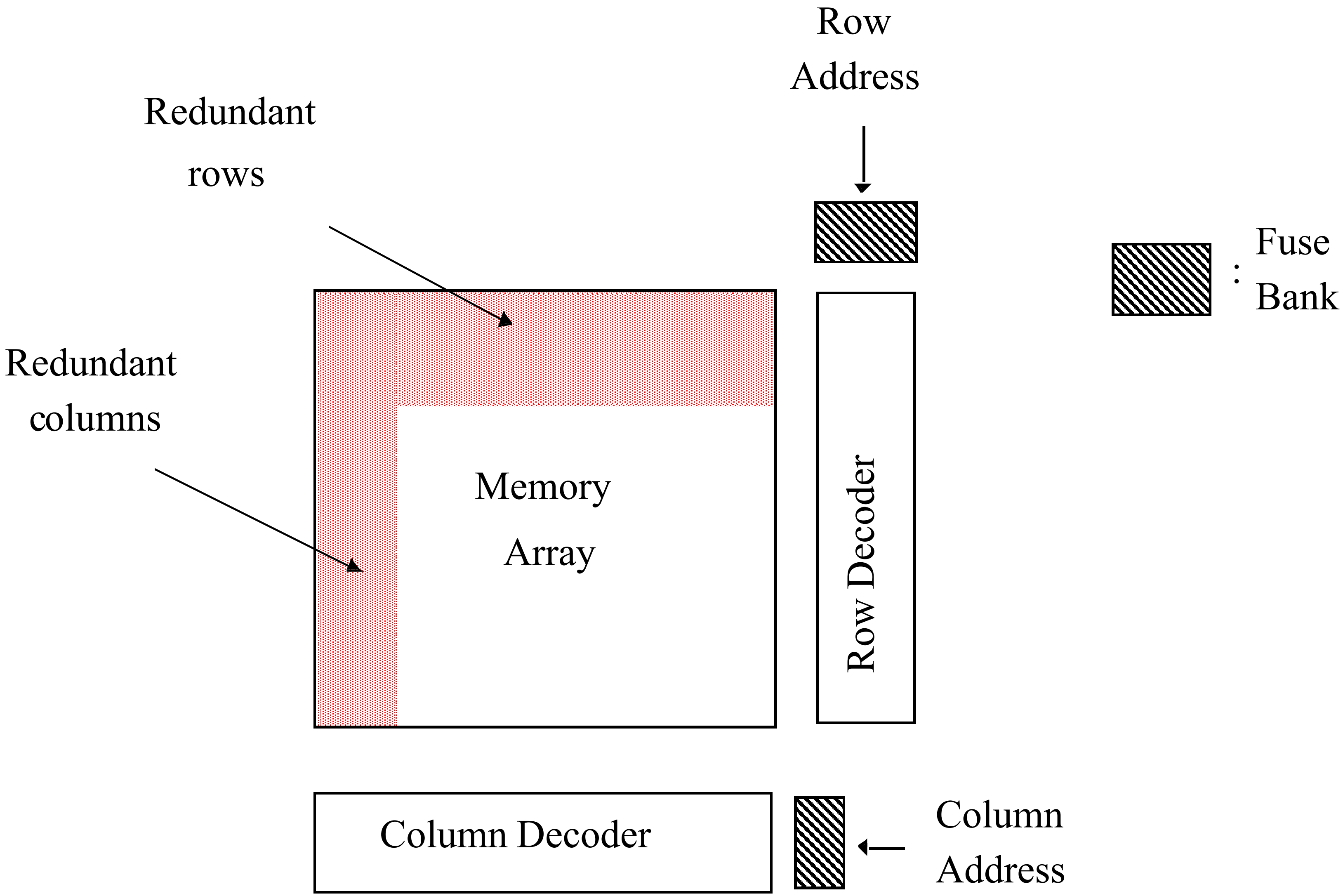
Figure 8, Ziegler, et al., "IBM experiments in soft fails in computer electronics (1978 - 1994)," IBM J. of R. & D., Vol. 40, No. 1, Jan. 1996.

3x - 5x increase in Denver at 5,000 feet

100x increase in airplanes at 30,000+ feet

From: S. Mukherjee, Intel, HPCA 2005

Redundancy in Memory Arrays



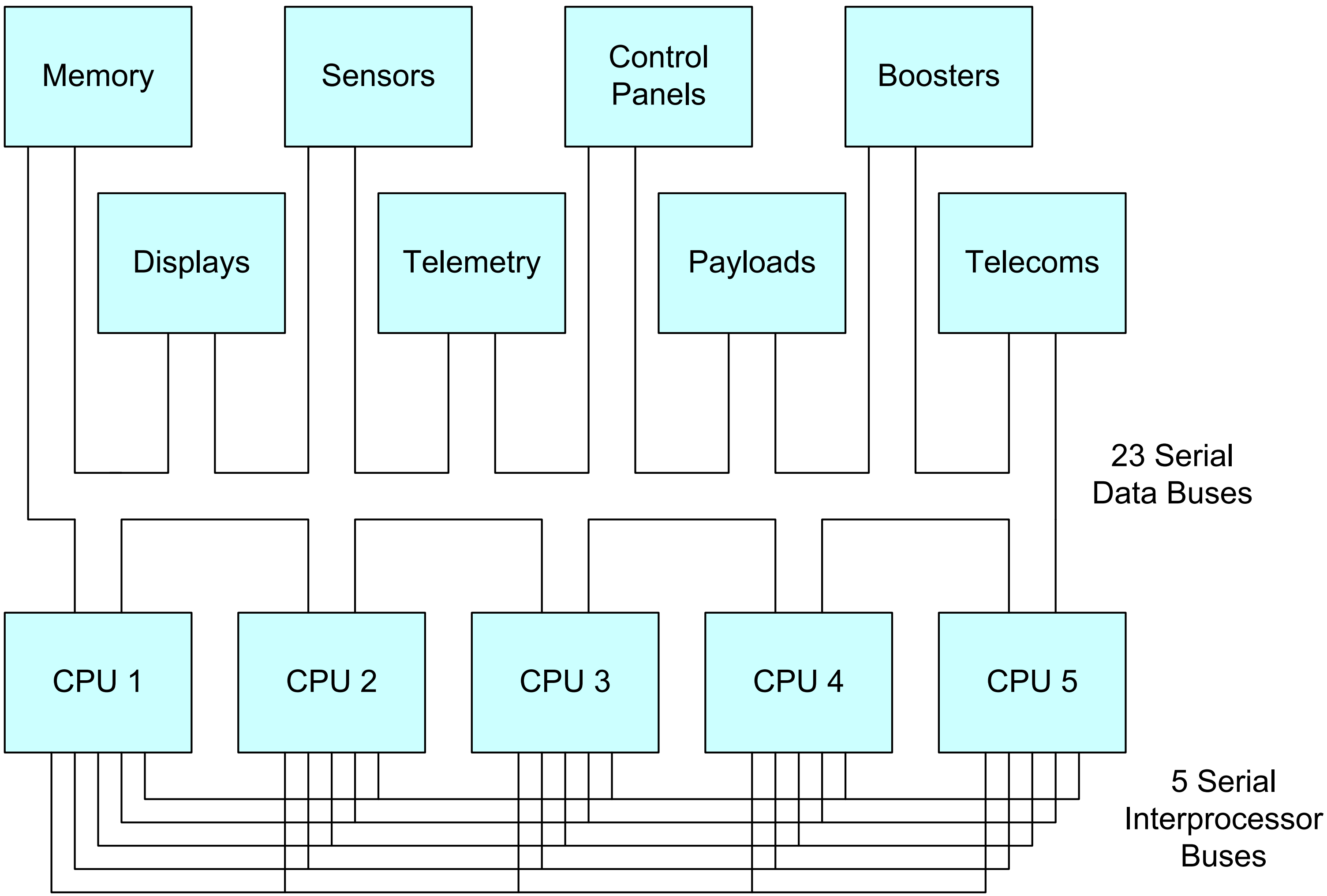
Some Example Fault-Tolerant Systems

Example: Darlington Nuclear Power Plant

- Two ways of shutting down reaction:
  - SDS1: Drop Neutron-absorbing shut-off rods into the reaction
  - SDS2: Injects liquid Gadolinium Nitrate into the reaction
- Both systems use separate sensors and separate software:
  - SDS1: 7000 lines of Fortran
  - SDS2: 13000 lines of Pascal
- Written by two different teams (but managed by the same person)

Example: Space Shuttle

Space Shuttle Computer Systems



Space Shuttle Computer Systems

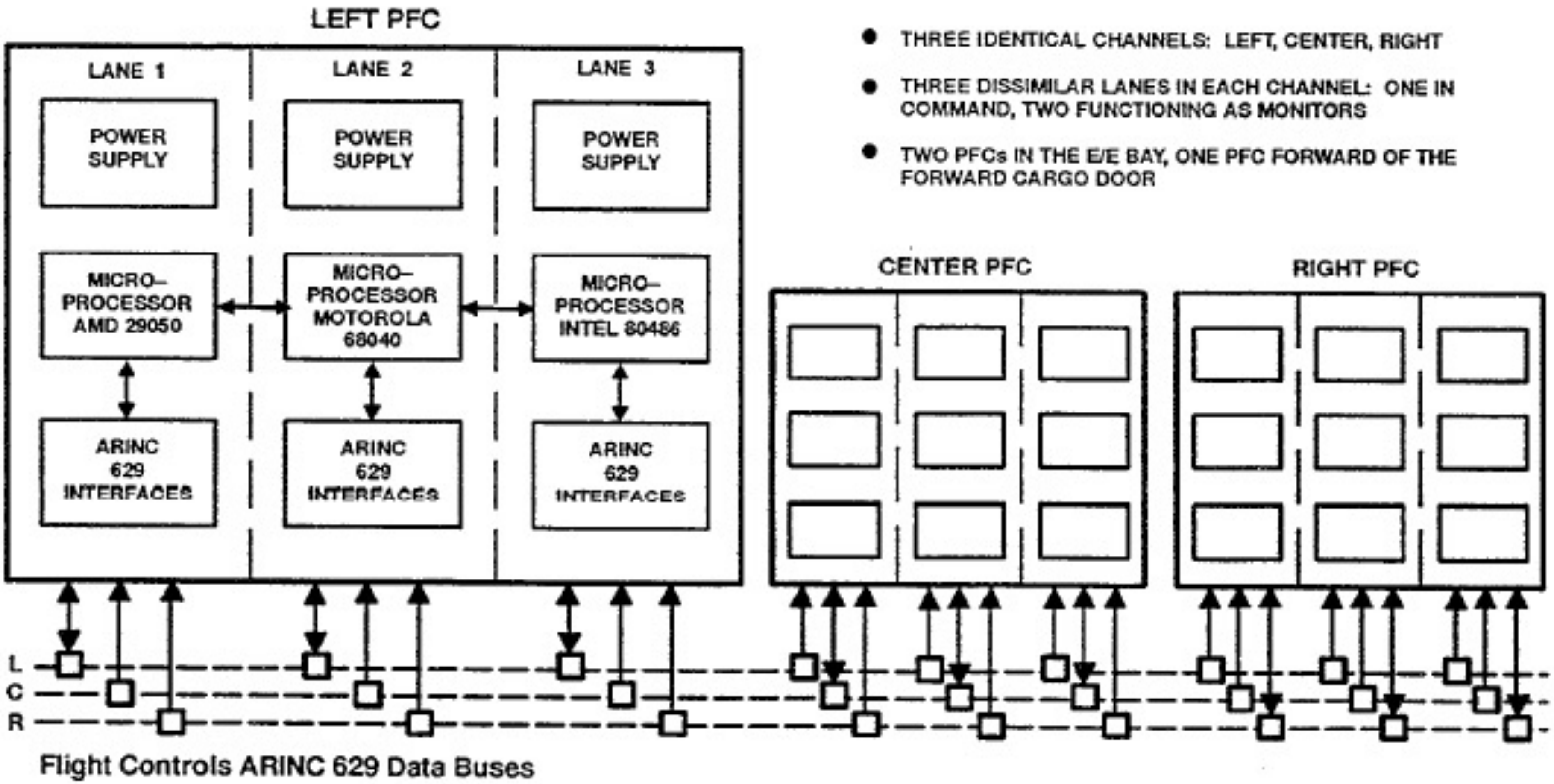
- Four CPUs are configured in a N-Way Modular Redundancy scheme
  - Each CPU executes the same code
  - Hardware voting is done, but each processor also compares its results to those from its neighbour
  - If there is a disagreement, voting is used to remove the offending computer
- When one computer fails, there are three left
  - Use TMR (Triple Modular Redundancy) techniques
- When another computer fails, there are two left:
  - Two remaining computers compare their results to detect failure
- When another computer fails, there is one left:
  - Inform the crew, try to detect what the problem is and maybe fix it?

Space Shuttle Computer Systems

- The fifth computer is normally used for non-critical functions such as communications.

- In an emergency, it can take over critical operations
  - It contains flight control software written by a different contractor
  - Provides some software diversity
  - However, all processors are of the same type (potential problem)

Modern Passenger Jet Processor Architecture



- THREE IDENTICAL CHANNELS: LEFT, CENTER, RIGHT
- THREE DISSIMILAR LANES IN EACH CHANNEL: ONE IN COMMAND, TWO FUNCTIONING AS MONITORS
- TWO PFCs IN THE E/E BAY, ONE PFC FORWARD OF THE FORWARD CARGO DOOR

Figure 5 Primary Flight Computer Channel Architecture

Mariner 1 Venus Probe

- Hardware/Software problems go way back…
- Mars 1 Venus Probe: Launched in July of 1962: “The first American attempt to send a probe to Venus. Guidance instructions from the ground stopped reaching the rocket due to a problem with its antenna, so the onboard computer took control. However, a bug in the guidance software caused the rocket to veer off course and it was destroyed by the range safety officer.”
- The problem was traced to the following line of Fortran code:
  - DO 5 K = 1. 3
- The period should have been a comma.
- An $18.5 million space exploration vehicle was lost

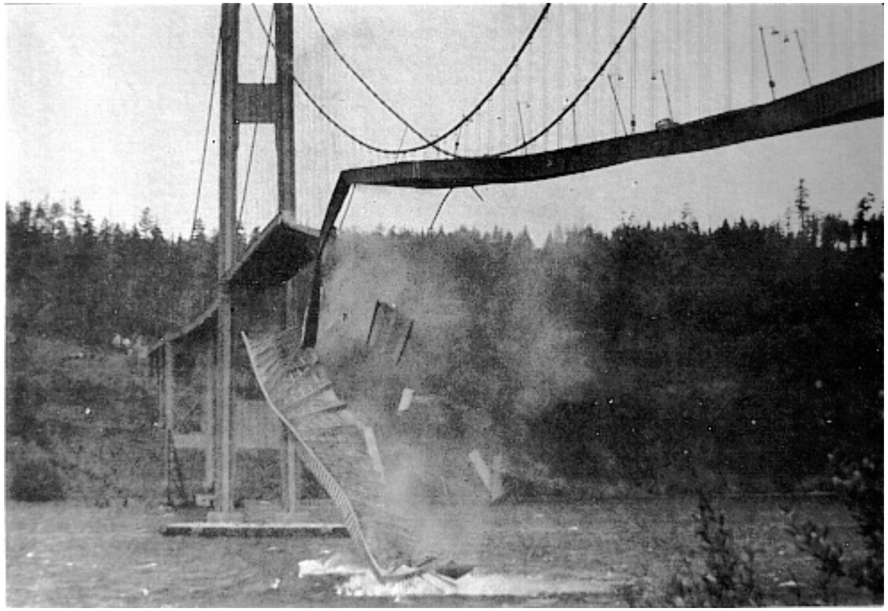
Learning from Other Fields…

In the 1800's, ¼ of iron truss railroad bridges failed!

Today: safety is now part of the Civil Engineering culture

- margin of safety: 3x-6x vs. calculated load
- What is the EE/CE margin of safety?

What will people in the future think of our computers?



Modified from Patterson, 2004

Estimating system reliability

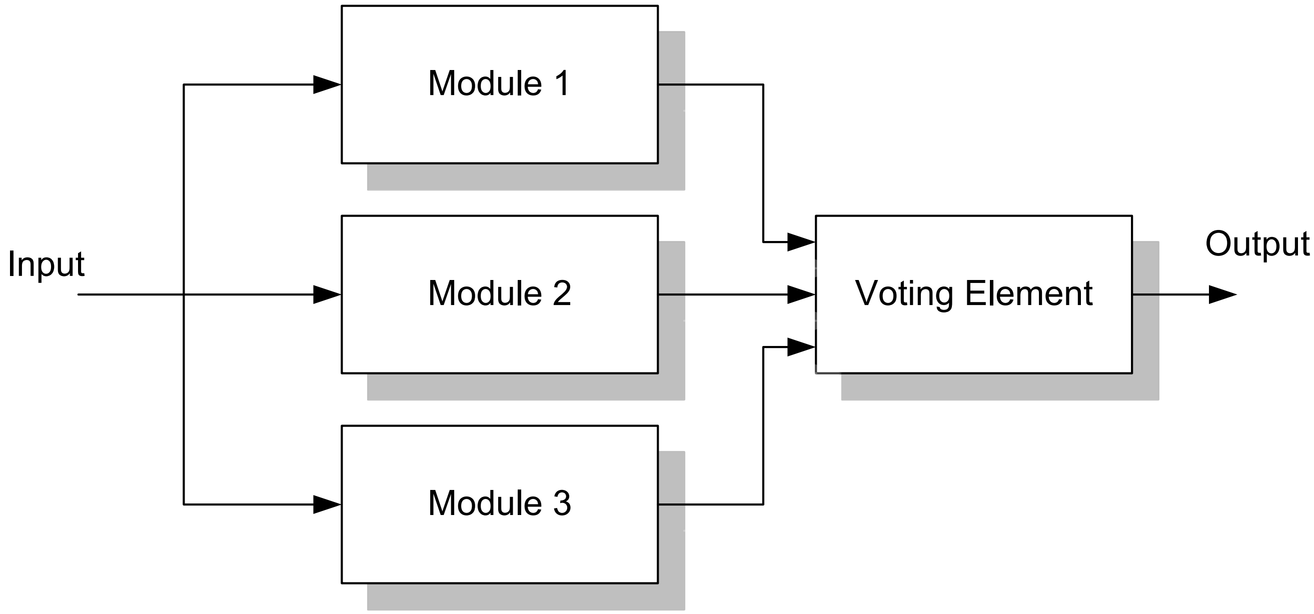
Definitions and basic mathematical modeling

CPEN 432– Design of Real-Time Systems

Overview

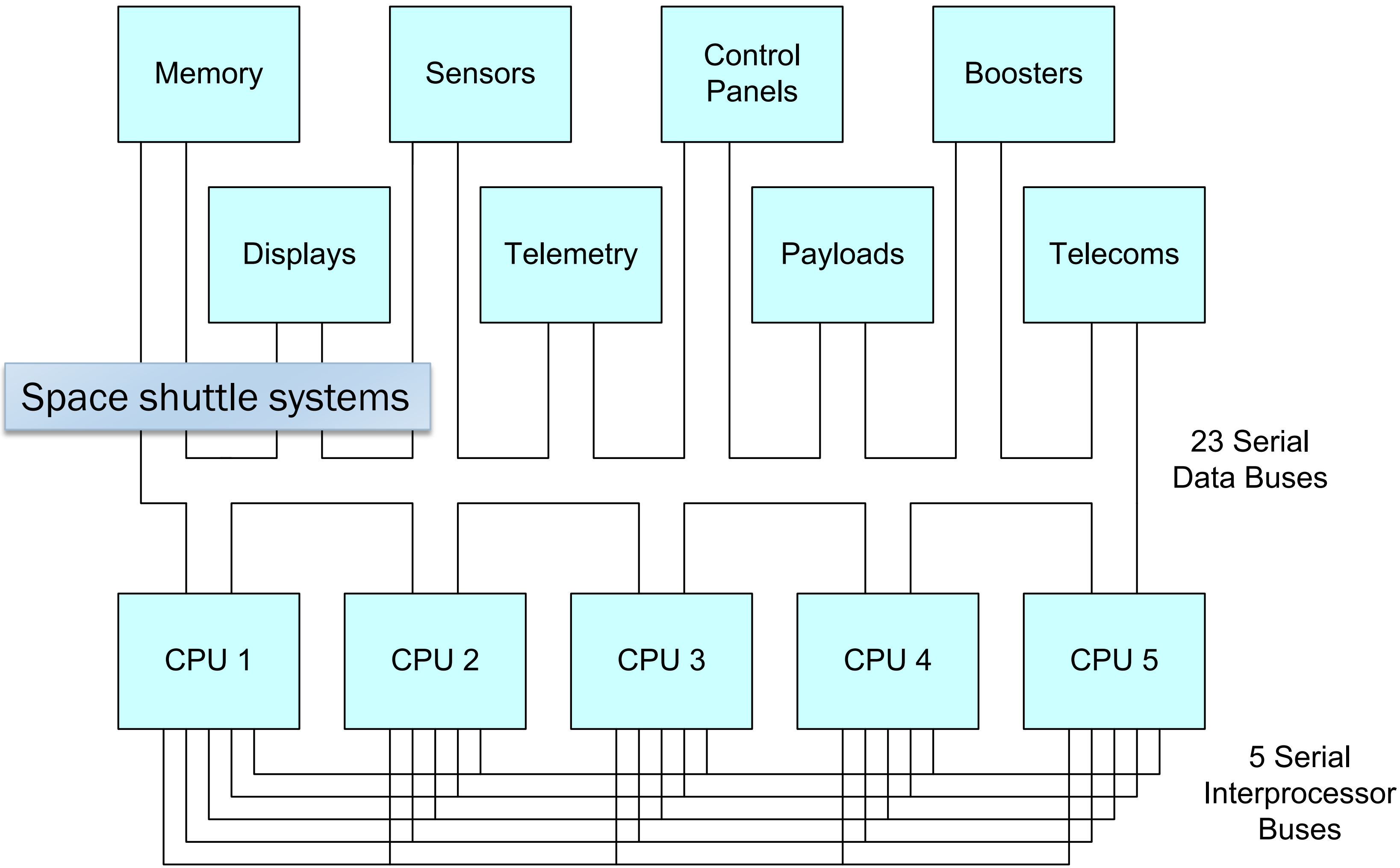
- Basic concepts
- Reliability expressions
- Mean time to failure (& mean time between failures)
- Availability
- Maintainability

Improving reliability via redundancy



Triple modular redundancy

Improving reliability via redundancy



Definitions

- Reliability: The probability that the given system will perform its required function under specified conditions for a *specified period of time*.

- MTBF (Mean Time Between Failures): Average time a system will run between failures. The MTBF is usually expressed in hours. This metric is more useful to the user than the reliability measure.

Increased system reliability

- Worst case design
- Use high quality components
- Strict quality control procedures

- Redundancy
- Typically employed
- Less expensive

Reliability expressions - Assumptions

- If a device is operable (up) then it produces **perfect** responses
  - Reliability defined in terms of *lifetime* of device
  - Applicable to hardware only
  - Not suitable for software and many lower-grade or “imperfect” systems → not practical
  - However, mathematically convenient
  - Will be relaxed later

Reliability expressions

T : positive real-valued random variable on a probability space (Ω, P)
lifetime of system

$$\boxed{\text{Reliability } R(t) = P(\text{system does not fail at time } t) = P(T > t)}$$

If T is exponential with rate λ failures/time unit : $R(t) = e^{-\lambda t}$

Failure Rate

- **Failure Rate:** probability that device fails in a finite interval of time, say of length Delta, given that the device has not failed until time *t*

$$\varphi(t,\Delta) = P(T \le t+\Delta | T > t)$$
$$= \frac{F(t+\Delta) - F(t)}{1 - F(t)}$$

Failure distribution function = 1 – *R*(*t*)

- If *T* is exponential with rate $\lambda > 0$:

$$\underbrace{\varphi(t,\Delta) = \frac{1 - e^{-\lambda(t+\Delta)} - 1 + e^{-\lambda t}}{e^{-\lambda t}} = 1 - e^{-\lambda\Delta} = \varphi(0,\Delta)}_{\text{Device starts anew if it is observed that it has not failed}}$$

Device starts anew if it is observed that it has not failed

Memoryless Property

Failure Rate Function

- Instantaneous failure rate

$$\lambda(t) = \lim_{\Delta \to 0} \frac{\varphi(t, \Delta)}{\Delta} = \lim_{\Delta \to 0} \frac{F(t + \Delta) - F(t)}{R(t)\Delta}$$

- **Interpretation:** probability that device with age *t* fails in a very short period of time around *t*

- If *T* is continuous and has density *f* that is continuous on its domain:

$$\lambda(t) = \frac{\frac{d}{dt}F(t)}{R(t)} = \frac{f(t)}{R(t)}$$

- **Example:** If *T* is exponential with rate $\lambda > 0$

$$\lambda(t) = \frac{\lambda e^{-\lambda t}}{e^{-\lambda t}} = \lambda$$ Constant Failure Rate!

Failure rate function examples

- Weibull

$$f(t) = \lambda \alpha t^{\alpha-1} e^{-\lambda t^{\alpha}}, \quad \lambda, \alpha > 0, t \geq 0$$

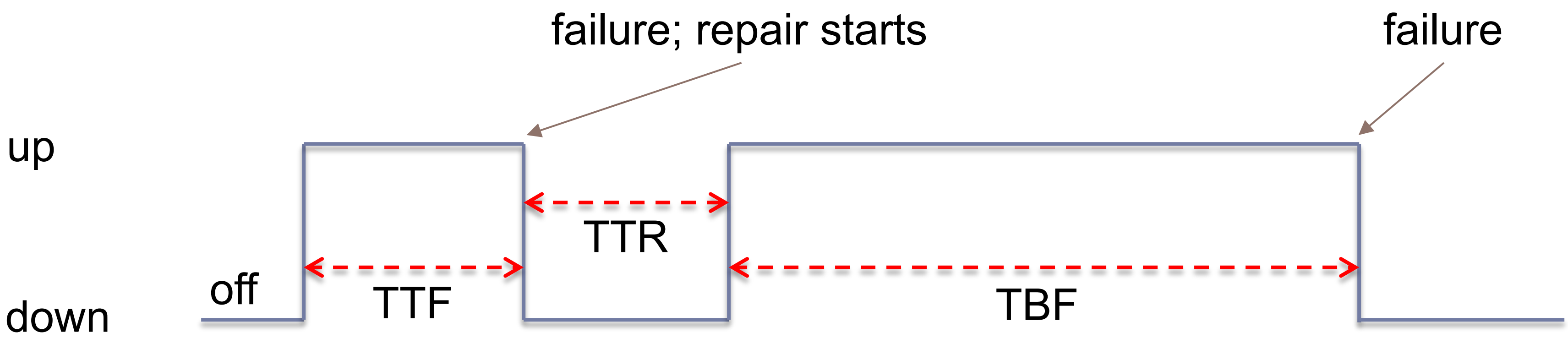
$$\lambda(t) = \lambda \alpha t^{\alpha-1}$$ Increasing Failure Rate (IFR) for $\alpha > 1$

- Log-normal

$$f(t) = \frac{1}{t\sigma\sqrt{2\pi}} e^{\frac{-(\log t - \mu)^2}{2\sigma^2}}$$

  - Check that $\lambda(t)$ is Decreasing Failure Rate (DFR) for sufficiently large t (in the long run)
  - Not suitable for devices; only material with hardening properties

Mean Time Between Failures (MTBF)



TTF: Time To Failure
TBF: Time Between Failures
TTR: Time To Repair

- MTBF is the expected (average) length of contiguous uptime (it is the expected lifetime if system is not repairable)

$$MTBF = \int_0^\infty R(t)dt = \int_0^\infty e^{-\lambda t}dt = \frac{1}{\lambda}$$

- The mean time between failures is the reciprocal of the failure rate
- If λ is the number of failures per hour, the MTBF is expressed in hours/failure
- If system not repairable then MTBF = MTTF
- If, after repair, a system behaves like it was new, there is no difference between MTTF and MTBF. Else there might be some difference

Example

- A system has 4000 components, each with a failure rate of 0.02% per 1000 hours. Calculate λ and the MTBF.

Assumption: The components fail independently, and the <u>entire system fails as soon as one component fails</u>

System lifetime $T = \min(T_1, \dots T_n)$

$$R(t) = P(T > t) = P(T_1 > t, T_2 > t, \dots, T_n > t)$$

$$= P(T_1 > t)P(T_2 > t) \cdots P(T_n > t)$$

$$= \exp[-(\lambda_1 + \dots + \lambda_n)t]$$

$$\lambda = \frac{(0.02/100)}{1000} \times 4000 = 8 \times 10^{-4} \text{ failures/hour}$$

$$\text{MTBF} = \frac{1}{8 \times 10^{-4}} = 1250 \text{ hours}$$

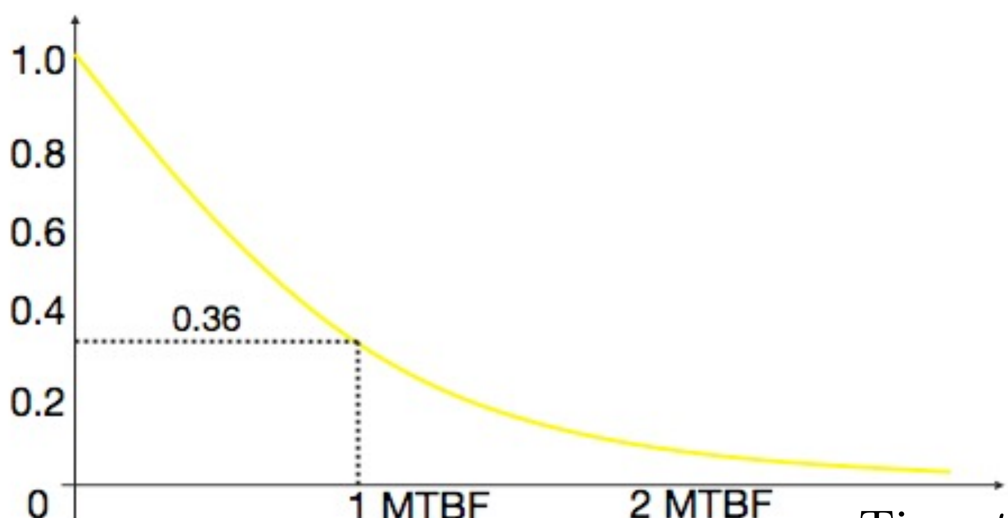
Relation between MTBF and reliability

- When λt is small: $R(t) = e^{-\lambda t} \approx 1 - \lambda t = 1 - (t/MTBF)$
- $MTBF = t/(1-R(t))$

$$\text{In particular: } 1 - \lambda t \le e^{-\lambda t}$$
$$\text{(underestimate; good)}$$

Reliability

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



Time t

Another example

- A first generation computing system contains 10,000 components, each with λ = 0.5% per 1000 hours. What is the period of 99% reliability? (All components have to function correctly.)

$$N = 10,000 \text{ (the number of components)}$$

$$\lambda = N \times \frac{0.5/100}{1000} = 5 \times 10^{-2} \text{ failures/hour}$$

$$\text{MTBF} = \frac{1}{\lambda} = 20 \text{ hours/failure}$$

Using the approximation:

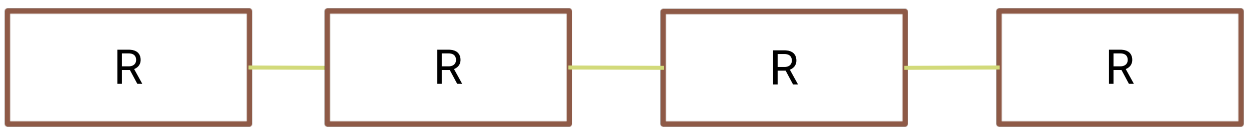
$t_{0.99}$

$$\text{MTBF} = \frac{t}{1 - R(t)} = \frac{t}{1 - 0.99} = 100t$$

$$t_{0.99} = 0.2 \text{ hours} = 12 \text{ minutes}$$

Reliability of different system configurations

- Series configuration



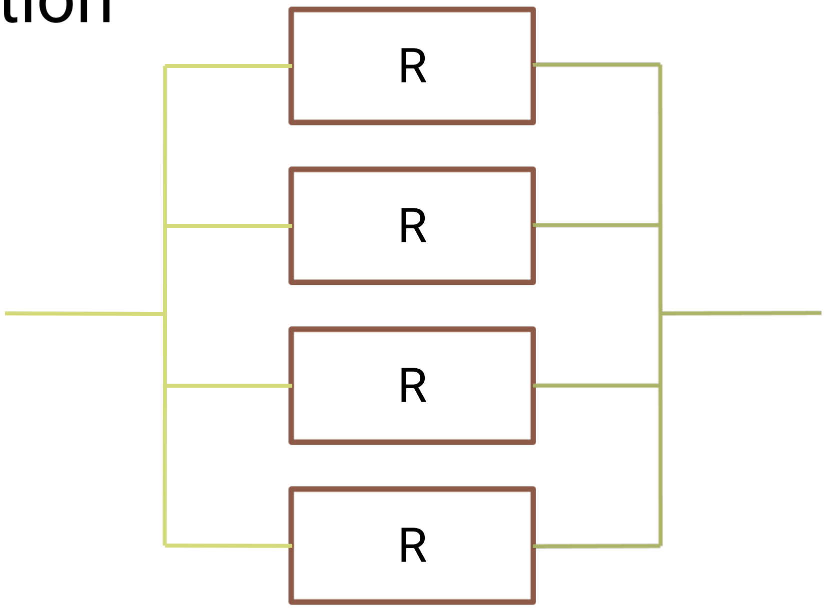
System Lifetime $T = \min(T_1, \ldots, T_n)$

$$P(T > t) = P(T_1 > t, T_2 > t, \ldots, T_n > t)$$
$$= P(T_1 > t)\, P(T_2 > t) \ldots P(T_n > t)$$

Overall reliability, $R_o = R \times R \times \ldots \times R = R^n$
(Assuming independent failures)

Reliability of different system configurations

- Parallel configuration

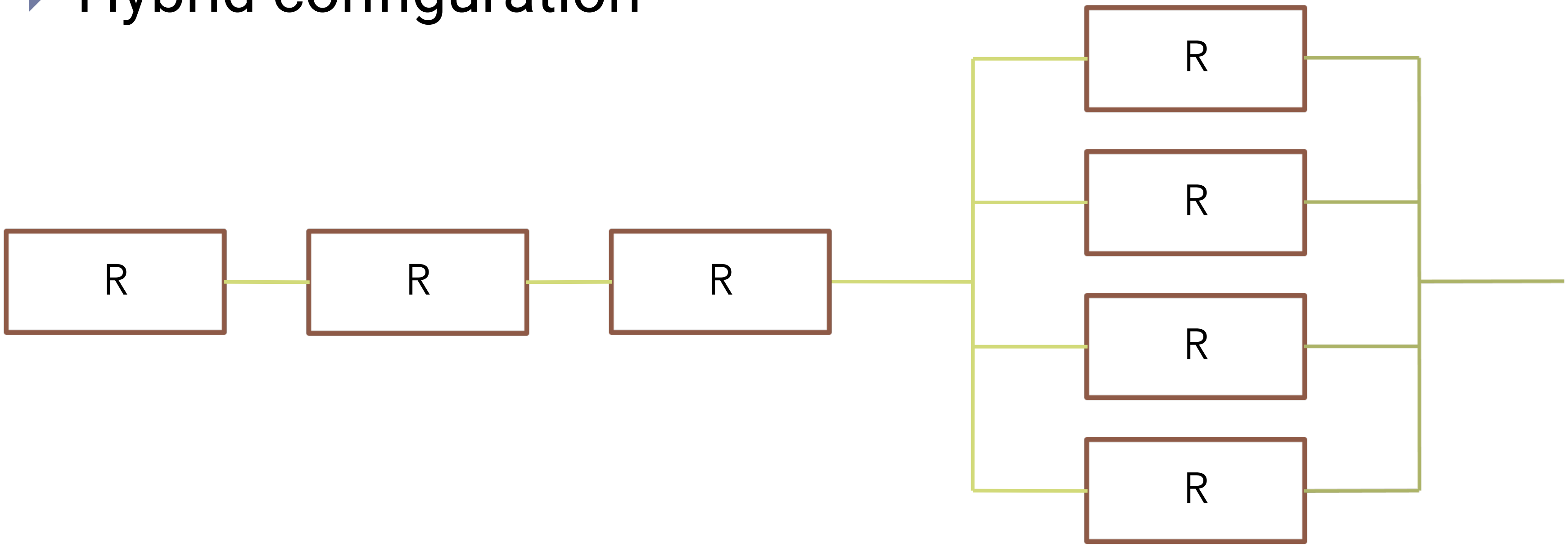


System Lifetime T = max(T_1, …, T_n)

Overall reliability R_o = 1 - (probability of all failures) = $1 - (1 - R)^n$

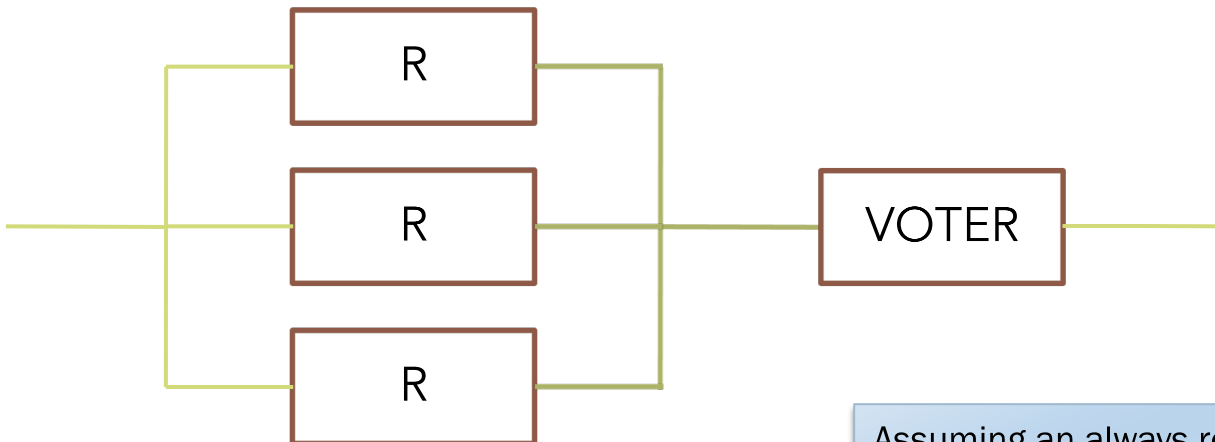
Reliability of different system configurations

- Hybrid configuration



Reliability of different system configurations

- Triple modular redundancy



Assuming an always reliable voter

$$R_o = \mathbf{P}(\text{at least 2 devices working})$$

$$R_o \quad = \quad \left[\binom{3}{2} \times R^2 \times (1 - R)\right] + R^3$$

An exercise

- You have five modules of the same type to design a fault-tolerant system. Each module has a failure rate λ_m. You have several possibilities for building redundancy. You can use a 5-input majority voter (with failure rate λ_5), a 3-input majority voter (λ_3), a 2-to-1 selection circuit (λ_{2s}), and a 2-input comparator (λ_{2c}). Assume that
  - $\lambda_5 = 2\lambda_3$
  - $\lambda_3 = 2\lambda_{2s}$
  - $\lambda_{2s} = \lambda_{2c}$

- What would be the best configuration to connect the modules?
  - You may make reasonable assumptions to ease calculations.

Maintainability

- Maintainability of a system is the probability of isolating and repairing a fault in the system within a given time duration.

- $M(t) = \mathbf{P}(S \leq t)$, where S is the **service time** random variable
  - If S is exponentially distributed, then $M(t) = 1 - e^{-\mu t}$, where μ is the repair rate and t is the permissible time for the maintenance action

- $M(t)$ is the probability that the system has been repaired at time t.

- $\mu = 1/MTTR$
  - MTTR is the Mean Time To Repair

Availability – Repairable Units

- So far we assumed that if a module fails it *stays in the failed state forever*
- If system is **repairable** (maintenance, replacement, etc) then its behavior in time will not only depend on its lifetime and on failure mode but also on the way in which it can be recovered and the duration of recovery time
- One possible definition of **availability**: A measure of the degree in which a system is in the operable (up) state at the start of mission when the mission is called for at an unknown random point in time. "MIL-STD-721"
- Repair action/policy is crucial to achievable availability
- In this context, a **highly available** system is a system with a low probability of breaking down (**high reliability**) and a high probability of being quickly repaired (**high maintainability**)
- Under the assumption that an up (operable) system is reliable, if system is not repairable then availability = reliability

Availability – Repairable Units

- Most common metric: **Steady state Availability**

- Steady state Availability ≈ $\frac{E[uptime]}{E[operating\ time]} = \frac{E[uptime]}{E[uptime]+E[downtime]}$
- If system not repairable, then Steady state Availability = 0 [HW will eventually fail]

- Downtime = (number of failures) x MTTR
- Downtime = (uptime) x λ x MTTR [Why? λ: # failures/normal operation time (uptime)]

- Steady state availability ≈ (uptime)/(uptime + ((uptime) x λ x MTTR))
  = 1/(1+ (λ x MTTR))
- When system is repairable: λ = 1/MTBF
- **Steady state Availability ≈ MTBF/(MTBF + MTTR)**

A Broader View of Reliability

- So far we assumed that if a system is up (operable) then it provides perfect answers (i.e., is "reliable")

- This is rarely the case, however:
  - Sensors are noisy (imperfect measurements)
  - Object detection algorithms (false positives/negatives)
  - A medical equipment that analyses blood or urine samples might produce inaccurate results
    - (most dangerous of which are **false negatives**: missing a diagnosis/disease when it is present in the sample!)

A Broader View of Reliability

- Should incorporate “accuracy/quality/correctness” of response

- Reliability is not lifetime

- Reliability becomes much harder to measure but availability still relatively easy

- In this more realistic reliability definition, availability and reliability can be at conflict

Reliability Redefined

- A system might fail in one of two ways:
  - It might fail to give a timely response at all **[availability problem]**, OR
  - It might respond in a timely manner but with the wrong answer **[reliability problem]**
- Increasing availability might reduce reliability and vise versa
  - Example: Increasing reliability by using two subsystems that both compute answers and then should compare their answers to produce final result
- It is important to understand the main objective of system design (does reliability have priority over availability?)
- Software testing is almost always concerned with reliability, checking availability just in passing

Software systems

- How do we improve reliability?
- Apart from N-versions programming and recovery blocks…

- Better operating systems support
- Stricter programming language primitives
- Rigorous engineering practice
- Formal verification of correctness

Redundancy… but at what cost?

- Discussion: What is an important factor affecting most engineering decisions that has been excluded in reliability estimation? How would you account for the impact of this factor?

- **The price of redundancy**. If we had a finite budget, is it better to improve the reliability of a component or rely on redundant (and less reliable) versions of that component?

Summary

- Building a reliable system involves integrating several redundant components.
- In this lecture we discussed:
  - Reliability and failure rates;
  - MTTF and MTBF;
  - Maintainability;
  - Availability.
- We also analyzed the reliability of different configurations.
  - Parallel, Serial, Hybrid, TMR.