Safety-Critical Embedded Systems

EDAN85 Embedded Systems Design - Continuation (Advanced) Course, Lecture 5

Lecture 3 Contents

Compacted and overview version of the full **DTU** course on Safety-Critical Embedded Systems, by Paul Pop

- Terminology
- Examples
- When to deal with faults?
  - forecasting, prevention, removal, tolerance
- How to deal with faults?
  - redundancy in hardware, information, time, software

A few definitions

- **safety**: system property; will not endanger human life or the environment
- **integrity**: system property; able to detect and inform about faults in its own operation
- **safety-critical system**: *safety-related system* (ensures safety) or *high-integrity system* (failure could mean financial loss)
- **risk**: combination of likelihood of an accident (failure) combined with the severity of potential consequences

Ariane 5 Rocket, 1996

- Off course & self-destruct 40s after launch
- **Cause**: overflow due to a 64-bit to 16-bit conversion, because of reuse of a software module (used in Ariane 4) related to horizontal velocity measurement. Both active and backup computers were affected.
- **Loss**: $500 million (rocket), $7 billion (project)

Therac-25 Radiation Therapy Machine, 1985

- Severe overdose of beta radiation (during treatment)
- **Cause**: race condition (improper concurrency) in the software controlling interlocking for safety. Hardware interlocking replaced by this software.
- **Loss**: at least 6 injured, 3 dead

Patriot Missile System, 1991

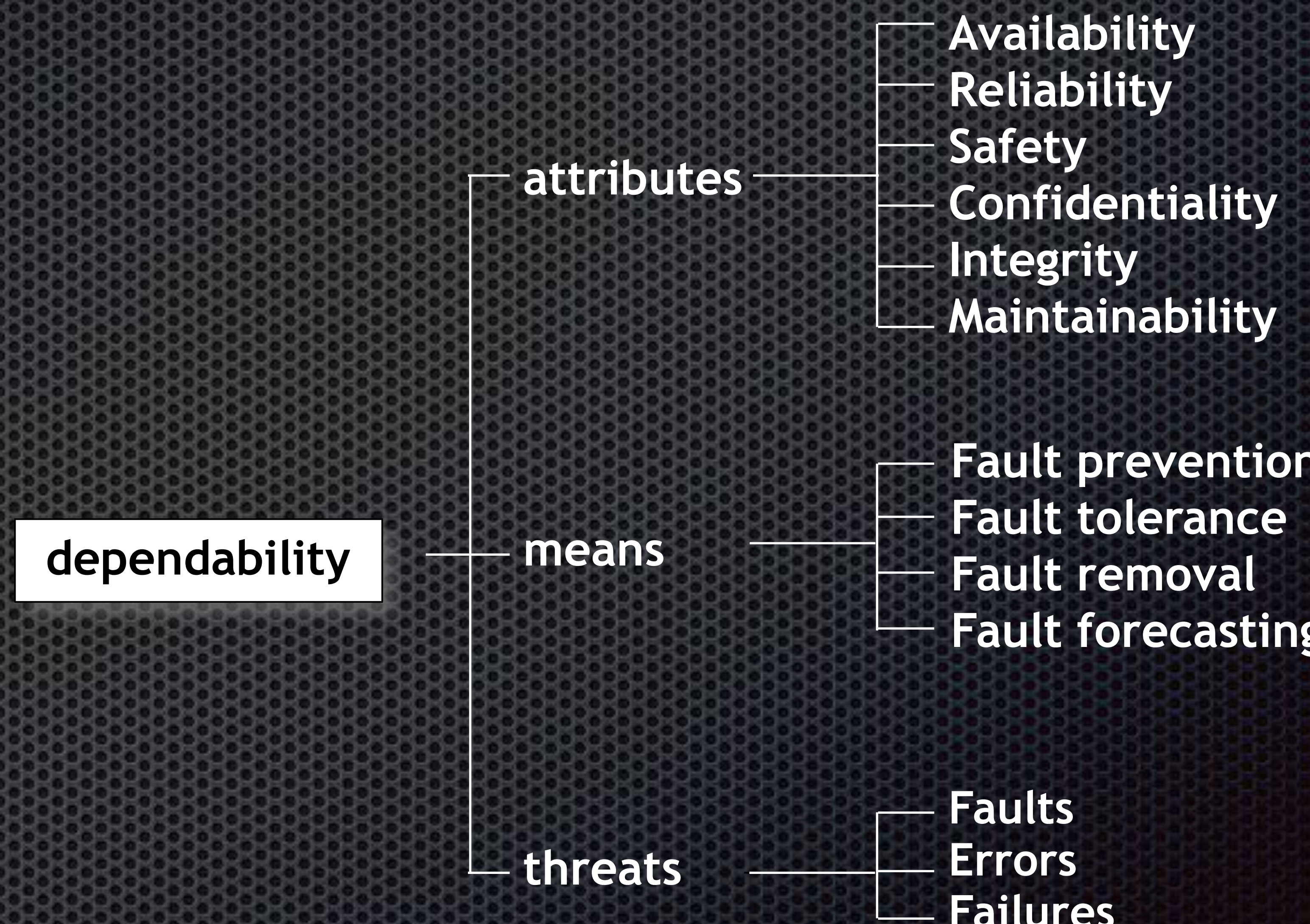
- System fails to intercept incoming missile
- **Cause**: time kept internally in tenths of seconds (badly represented in binary) leading to accumulated error and drifting. After 100h of uptime, the precision error is 0.34s
- **Loss**: 28 soldiers dead, 100 injured

More Examples…

- **Mars Orbiter**, 1998: crash, discrepancy in units used for impulse measurement vs. calculation (pound-seconds vs. newton-seconds). $125 million
- **Infusion pumps**: used to deliver fluids to a patients body in a controlled manner. FDA (US): 56,000 adverse reports of incidents including injuries or deaths (2005-2009).

An encompassing concept: Dependability

- system property; justifies placing one's reliance on it



- **Security**: the concurrent existence of (a) availability for authorized users only, (b) confidentiality, and (c) integrity

Faults, Errors, Failures

Fault → **Error** → **Failure**

Fault	Error	Failure
cause of error (and failure)	unintended internal state of the subsystem	deviation from intended service
Physical Universe	Informational Universe	User's Universe

Examples

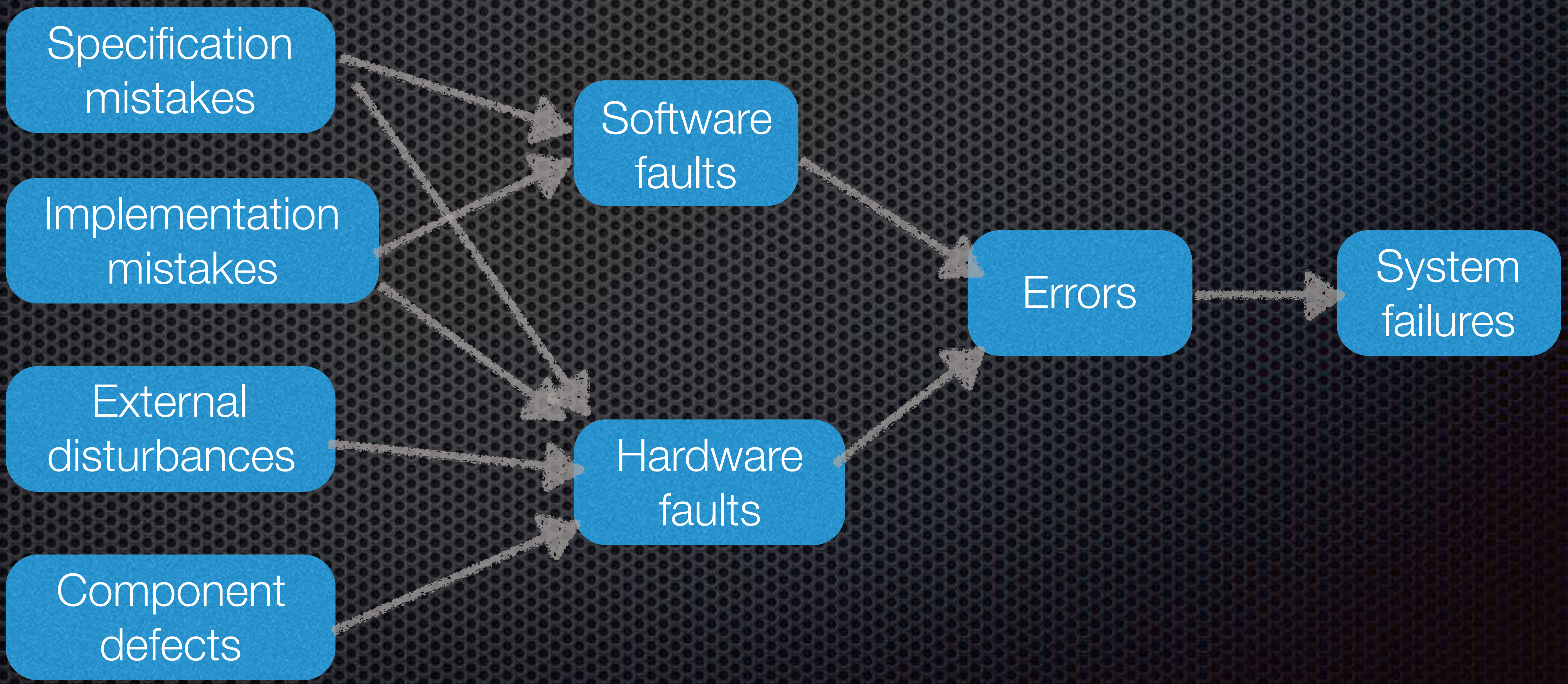
Physical Universe:
- electrical shorts
- imperfections in semiconductors
- unwanted infinite loops in programs

Informational Universe:
- stuck at 1/0
- changed memory contents
- task taking 100% of CPU

User's Universe:
- actuator does not update (always open)
- wrong data output
- unresponsive system

Causes of Faults

- problems at any stage in the design process can result in faults within the system

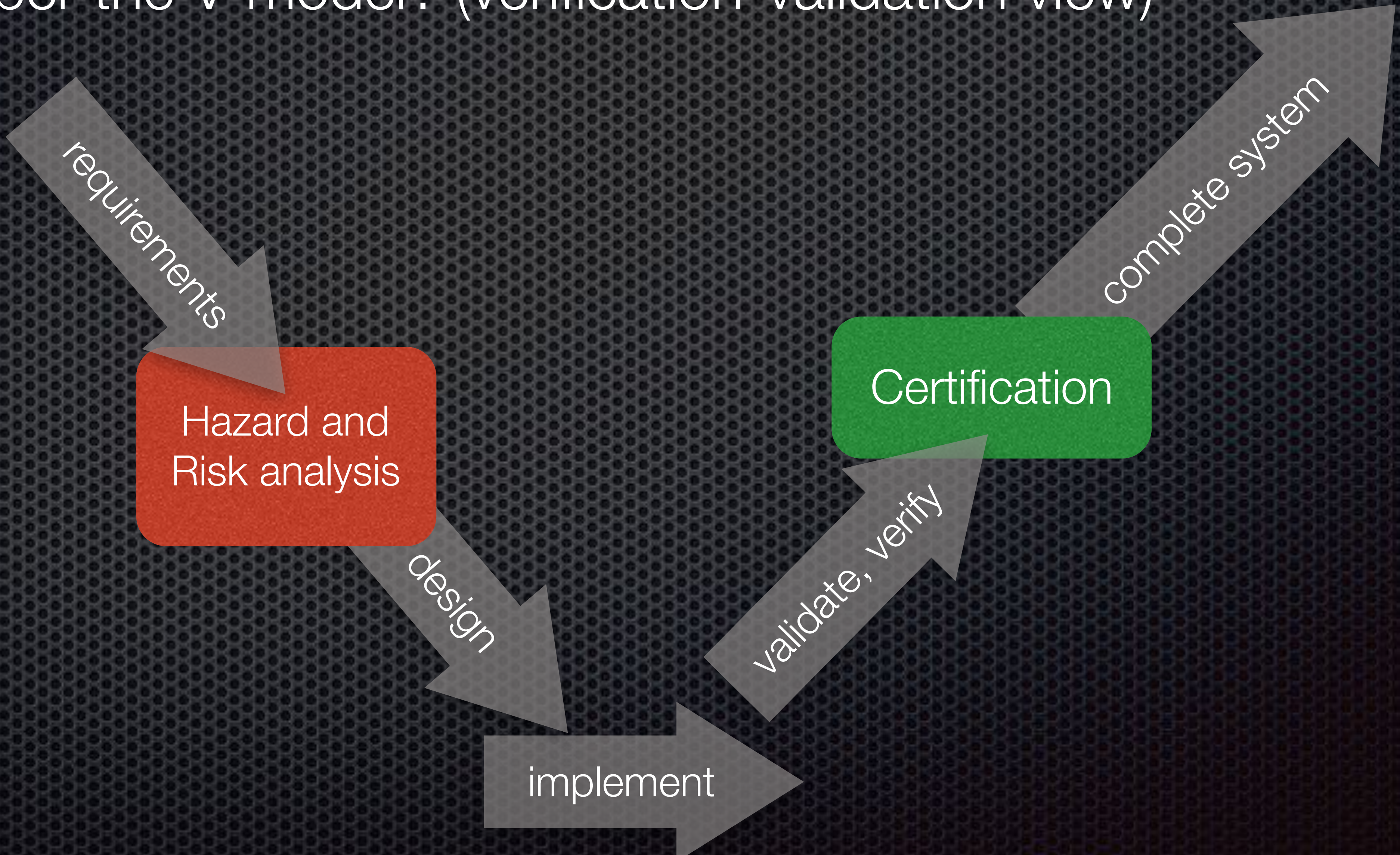


Failure Modes

- Failure **domain**: value/timing
- Failure **consistency**: consistent (all parts see the same result) /inconsistent (byzantine)
- Failure **consequences**: benign (loss of utility) / malign (significantly more severe, catastrophic)
- Failure **ofteness**: permanent/transient (if repeated, intermittent)

Design Life-Cycle for Safety-Critical Systems

- a concern throughout the whole design cycle
- remember the V-model? (verification-validation view)



Dependability Areas

A. **Fault forecasting**: how to minimize, by <u>evaluation</u>, the presence, creation and consequence of faults

B. **Fault prevention**: how to prevent, by <u>construction</u>, fault occurrence

C. **Fault removal**: how to prevent, by <u>validation</u> and <u>verification</u>, the presence of latent faults

D. **Fault tolerance**: how to provide, by <u>redundancy</u>, the service complying a specification despite the occurrence of faults

A. Fault forecasting

Evaluation of the system behavior with respect to fault occurrence.

- **Qualitative** evaluation
  - identifies, classifies, ranks the failure modes and events that lead to system failures
  - Example methods: Failure Mode and Effect Analysis (FMEA), Fault-Tree Analysis (FTA)
- **Quantitative** evaluation
  - evaluates in terms of probabilities the extent to which some the dependability attributes are satisfied (measures dependability)
  - Example methods: Markov chains, reliability block diagrams

An Example: (qualitative) Fault-Tree Analysis

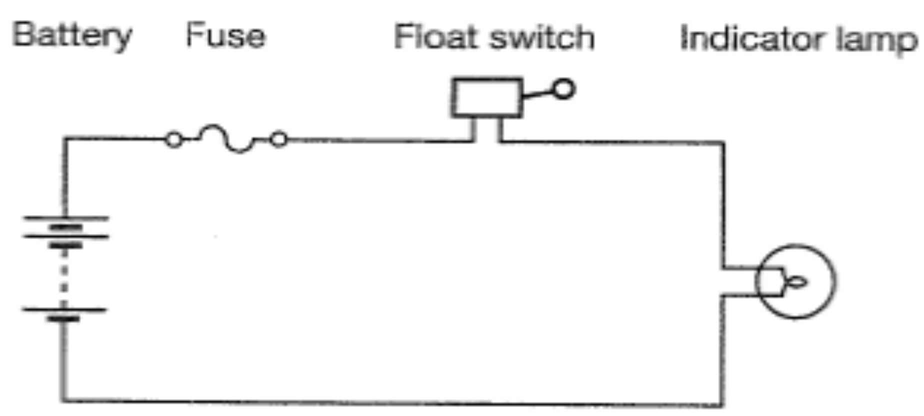


Figure 3.9 An automotive brake fluid warning lamp arrangement.

- construct a fault-tree for an automotive brake fluid warning lamp
- the event is lamp failing to be lit when brake fluid is low

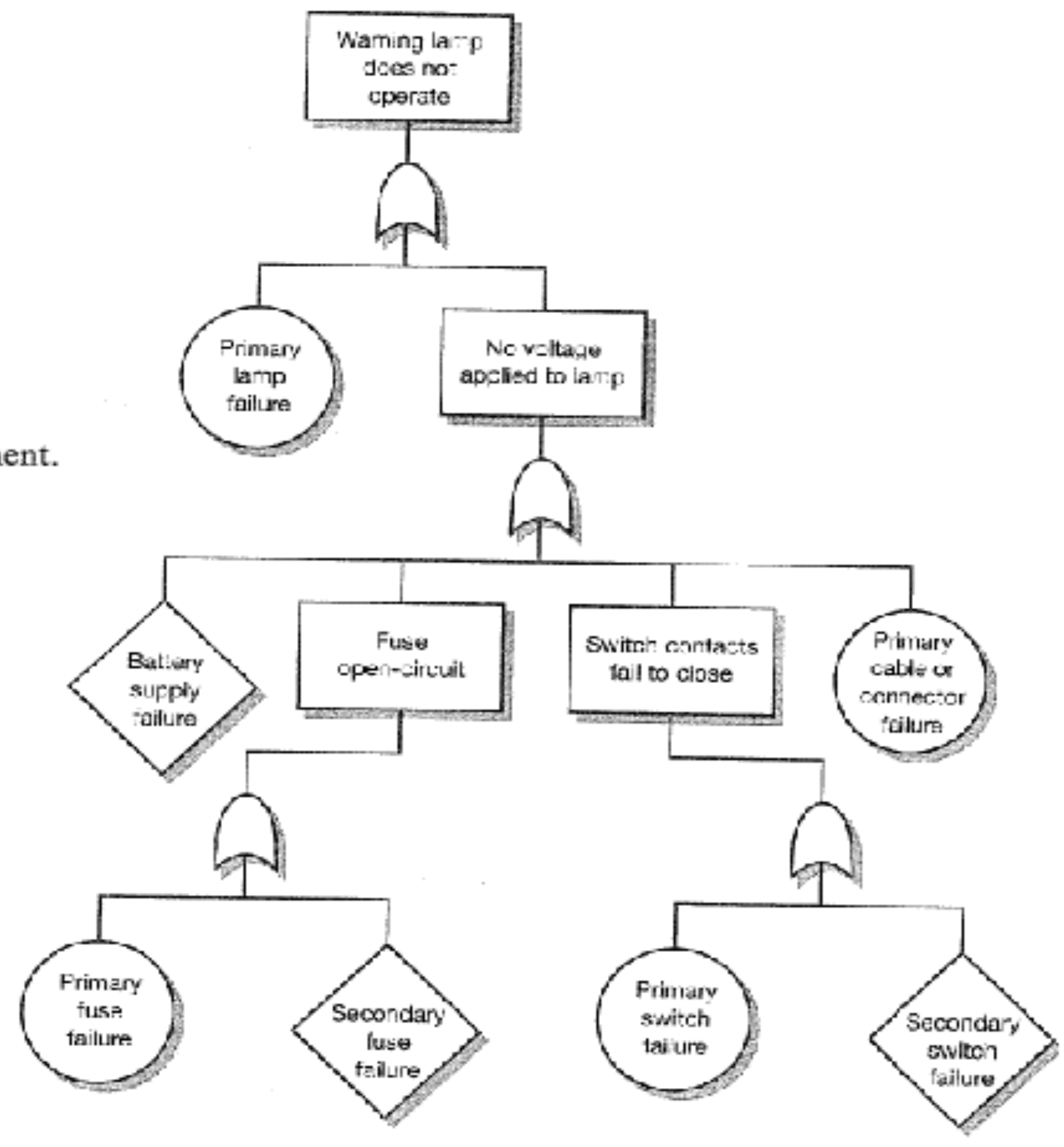
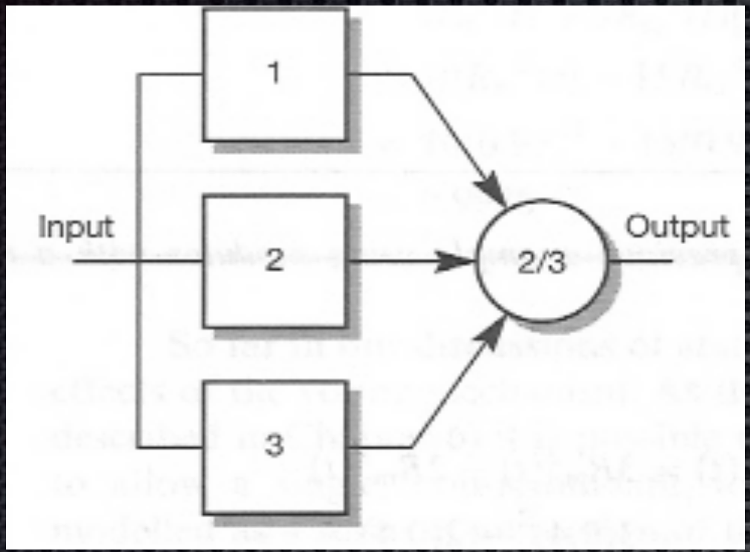


Figure 3.10 A fault tree for a brake fluid warning lamp system.

Intermezzo: more useful terms

- **Reliability**: the probability of a system/component functioning correctly over a period of time under a given set of operating conditions
- **Mean Time to Failure (MTTF)**: the expected duration the system will operate before the first failure
- **Mean Time to Repair (MTTR)**: the average time required to repair the system
- **Mean Time Between Failures (MTBF) =** MTTF + MTTR
- **Availability**: the probability that the system will be functioning correctly at any point in time = MTTF/MTBF

An Example: Reliability Analysis of Triple Modular Redundancy (TMR)



Probability of correct operation = Probability of no failures
+ Probability of only module 1 failing
+ Probability of only module 2 failing
+ Probability of only module 3 failing

- probability of a module working correctly: **R(t)**
  probability of failing: **1-R(t)**
- TMR system reliability

$$R_{\mathrm{TMR}}(t) = R_1(t)R_2(t)R_3(t) + [1 - R_1(t)]R_2(t)R_3(t) + R_1(t)[1 - R_2(t)]R_3(t) + R_1(t)R_2(t)[1 - R_3(t)]$$

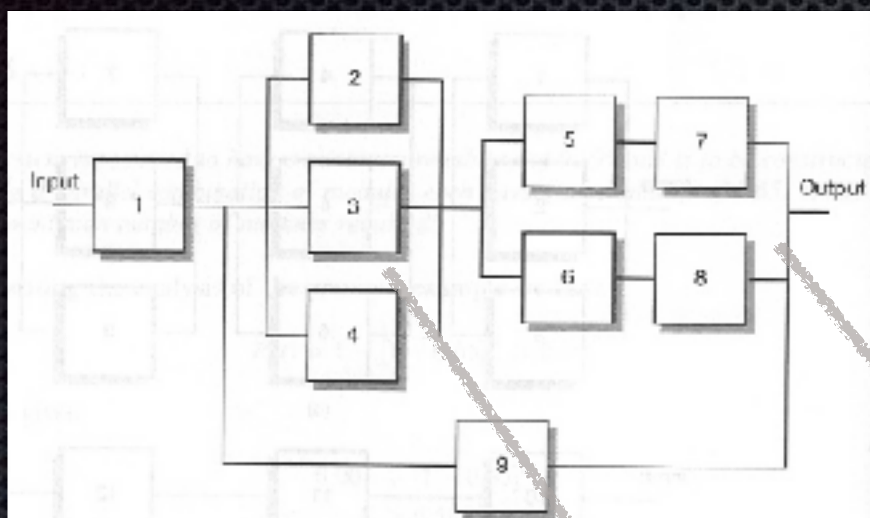
- For three identical modules

$$\begin{aligned} R_{\mathrm{TMR}}(t) &= R_{\mathrm{m}}{}^3(t) + 3R_{\mathrm{m}}{}^2(t)[1 - R_{\mathrm{m}}(t)] \\ &= 3R_{\mathrm{m}}{}^2(t) - 2R_{\mathrm{m}}{}^3(t) \end{aligned}$$

- The voter is a very simple module, allowing for a non-redundant unit

Reliability Analysis for Arbitrary Systems

- series of modules: none should fail
- parallel modules: at least one is OK



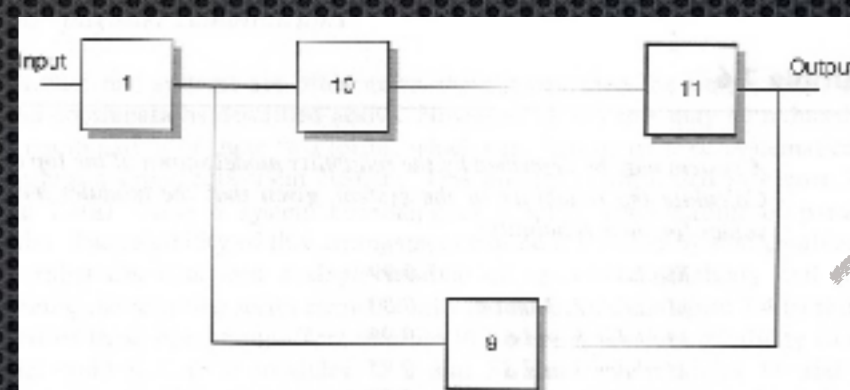
Module 1	0.99
Modules 2, 3 and 4	0.80
Modules 5 and 6	0.90
Modules 7 and 8	0.95
Module 9	0.94

2,3,4

$$R(t) = 1 - [1 - R_m(t)]^N$$
$$= 1 - [1 - 0.8]^3$$
$$= 0.992$$

5,7 and 6,8

$$R(t) = 1 - [1 - R_m(t)]^N$$
$$= 1 - [1 - 0.855]^2$$
$$= 0.979$$



10,11 and 9

$$R(t) = 1 - [1 - R_1(t)][1 - R_2(t)]$$
$$= 1 - [1 - 0.971][1 - 0.94]$$
$$= 0.998$$

Finally 1 and rest: **0.99**

- combinations of <u>series/parallel</u> compositions can be reduced to a single reliability measure in an easy way
- other methods (using paths) for non-series/parallel compositions
- sometimes bounds are enough, if exact values are hard to compute

B. Fault prevention

Use quality control techniques to **avoid faults at construction time**.
(Controlled Design Processes, Guidelines, Standards)

- Software
  - structured/object oriented programming
  - information hiding/modularization
  - support (tools) for compilation/run-time (e.g. GC)
- Hardware
  - rigorous design rules
  - shielding/foolproof packaging
  - radiation hardening
- Note: malicious faults can also be prevented (e.g. firewalls)

C. Fault removal

- **Verification**: “Are building the system right?”
  - Static: does not exercise the system (inspections, walkthroughs, model checking)
  - Dynamic:
    symbolic execution (inputs are symbolic), testing (actual inputs)
  - Fault injection: improve test coverage by forcing faults (in particular error handling)
- **Validation**: “Are we building the right system?”
  - Checking the specification

D. Fault tolerance

The ability of a system to continue operating correctly even when one or more components have failed.

- Masking: sufficient redundancy may allow for recovery without explicit error detection
- Reconfiguration: eliminating a faulty entity from the system and restoring the system to operational state
  1. Error **detection**: recognizing that an error occurred
  2. Error **location**: identifying the module with the error
  3. Error **containment**: preventing errors from propagating
  4. Error **recovery**: regaining operational status

The concept of redundancy

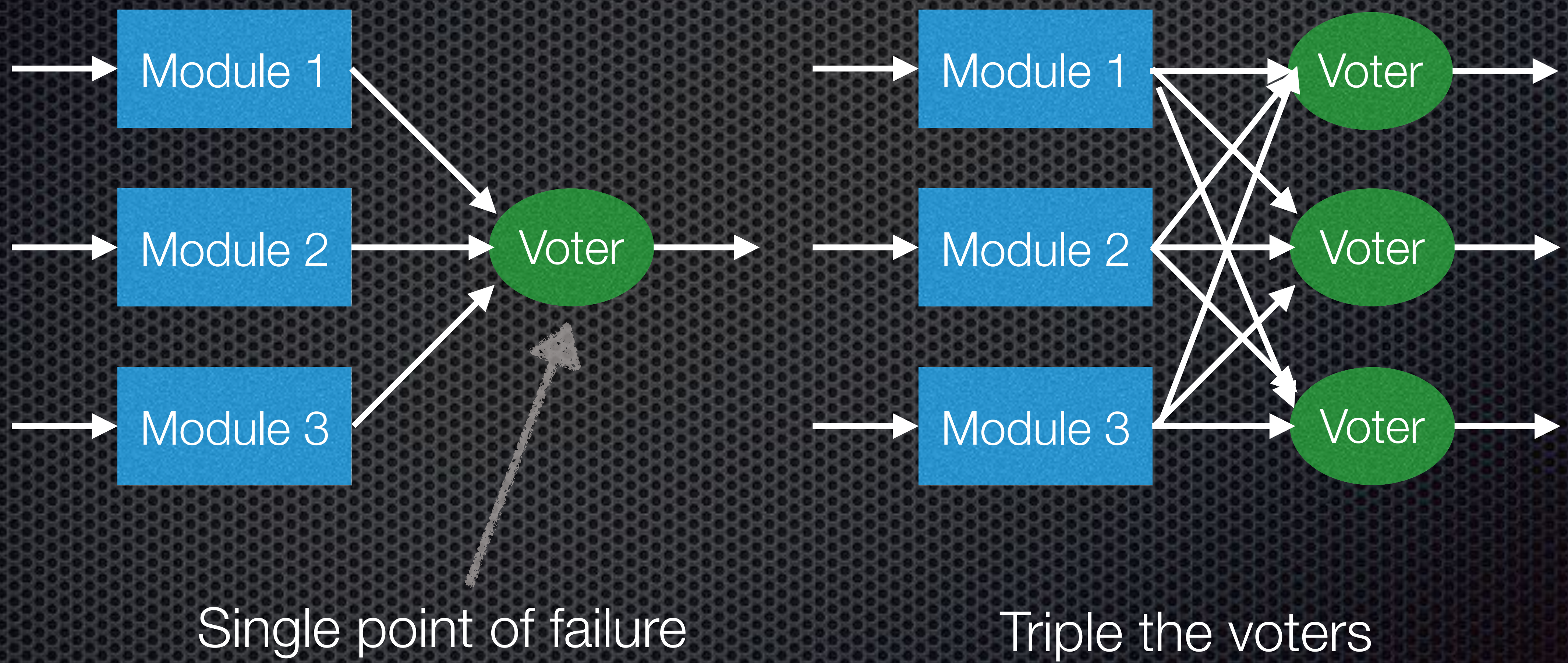
- **Redundancy** is the addition of information, resources, or time beyond what is needed for normal system operation

- Example for a digital filter
  1. software redundancy: lines of code to perform validity checks
  2. hardware redundancy: if more memory is needed for checks
  3. time redundancy: each filter calculation performed twice to detect (transient) faults
  4. information redundancy: using a parity check bit in the output

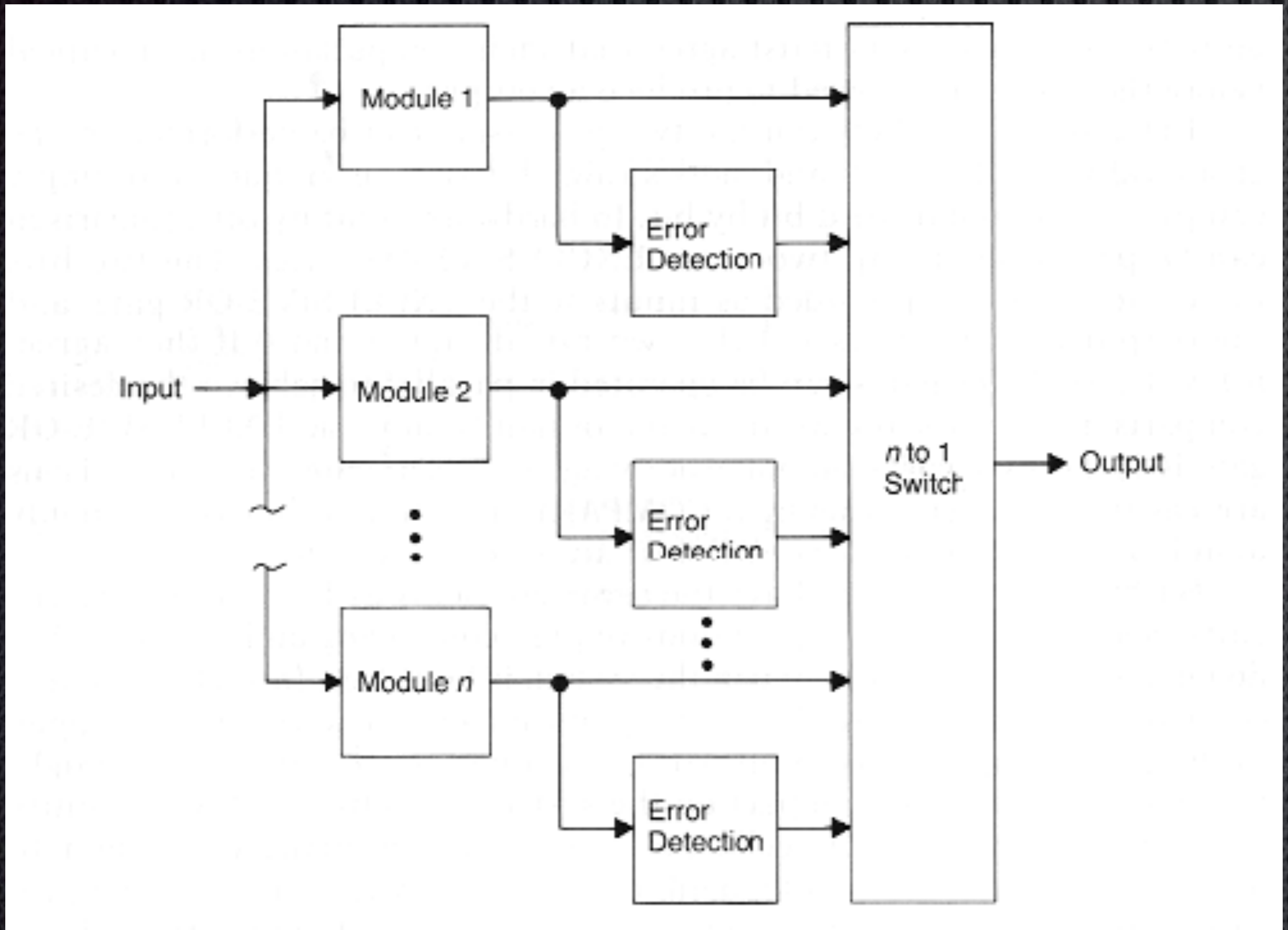
Hardware redundancy

- **Passive redundancy**: employ extra hardware to instantly mask errors
  - <u>M-of-N and voting</u>: systems with N identical modules, at least M need to function properly
- **Active (dynamic) redundancy**: no fault masking, instead detect, locate and recover
  - Standby sparing, duplication with comparison
- **Hybrid redundancy**: a combination of the above

M-of-N example: (passive) Triple Modular Redundancy



Standby sparing (active)

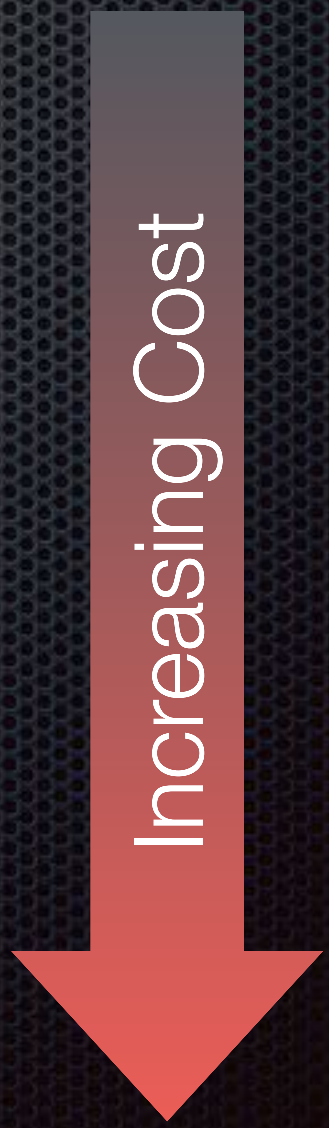


One module is operational while one or more modules are spares.

- **error detection** used to identify when a module is faulty
- **error location** is used to determine which module is faulty
- faulty modules are **removed** and **replaced** by a spare

The choice of hardware redundancy…

- **Active**: when temporary erroneous results are acceptable; most important is that the system can return to operational state in short enough time (e.g. satellite systems)
- **Passive**: critical-computations where momentary erroneous outputs are not acceptable
- **Hybrid**: applications requiring extremely high integrity of the computations



Information redundancy

Encode (decode) data using redundant bits in order to achieve detection/correction of (bit) faults.

Typical codes:

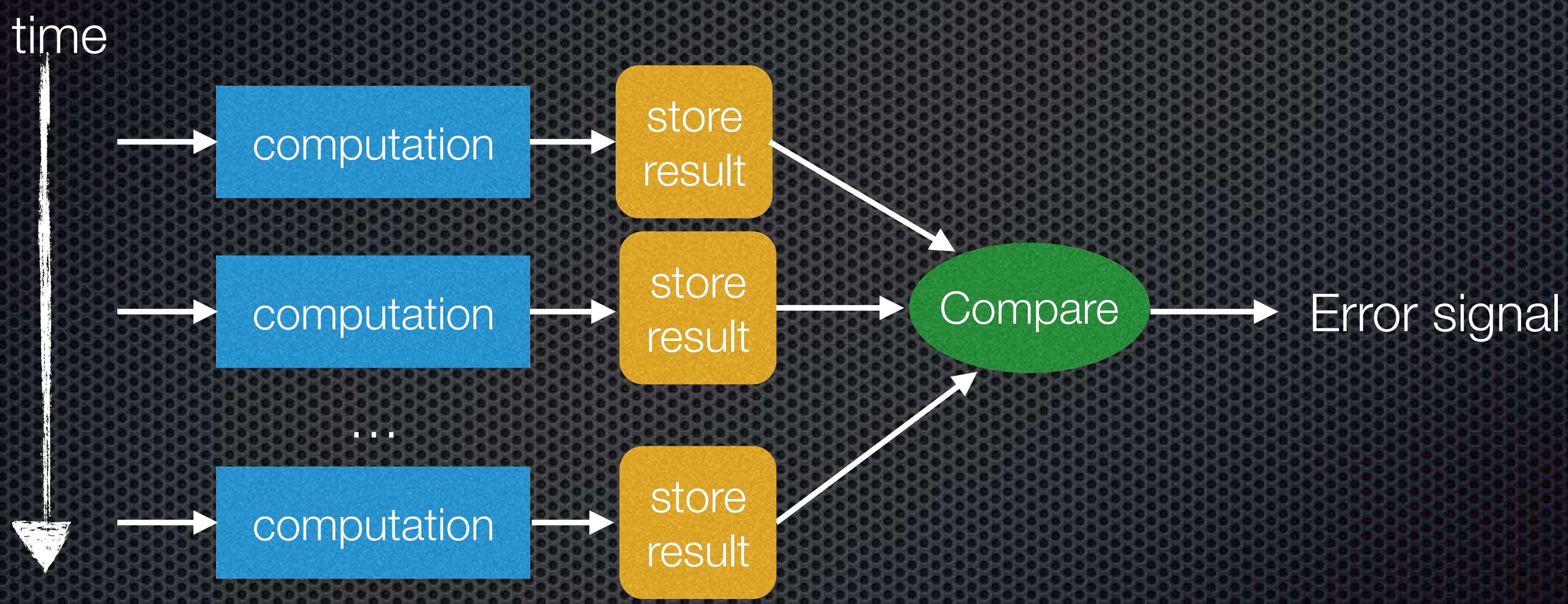
- checksum codes (e.g. parity)
- m-of-n codes
- Berger codes
- Hamming codes
- …

Time redundancy

Recomputing/resending the same results (possibly in a different way) in order to check for faults.

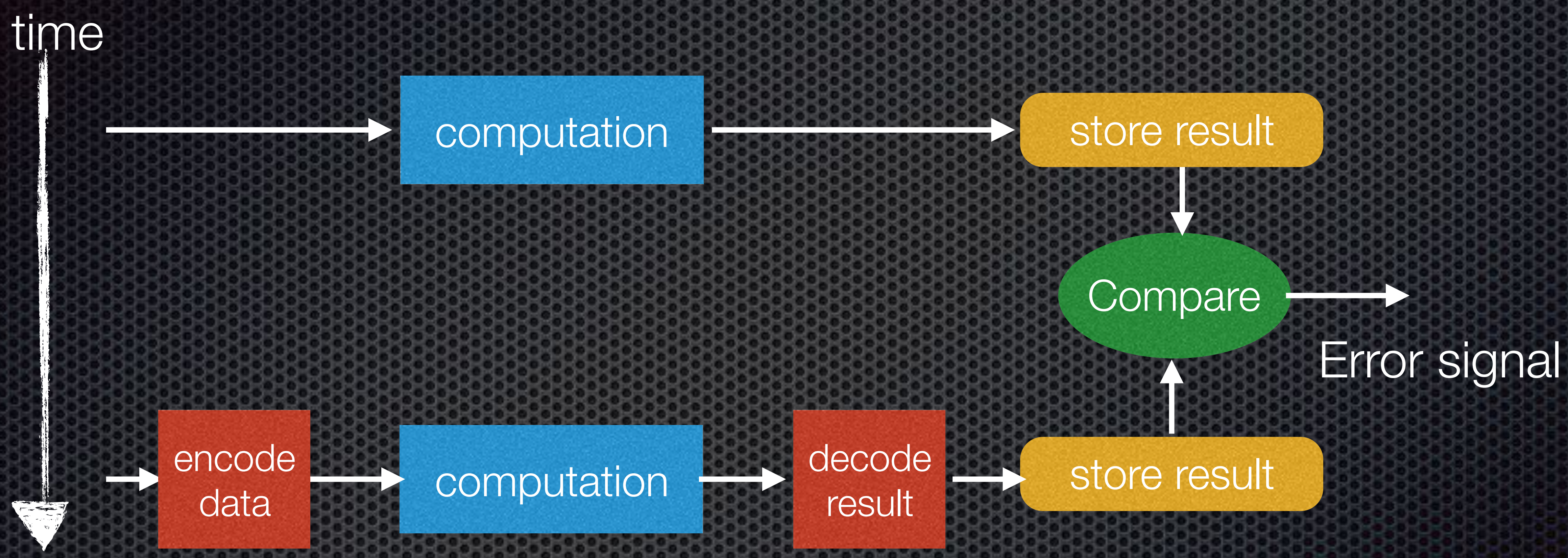
- Uses fewer resources than hardware and information redundancy, at the expense of more time (which may be possible in some applications)
- Can address transient or permanent faults

Transient fault detection



Identical computations are repeated over time.

Permanent fault detection



- used to detect permanent errors in the module performing the computation
- second computation uses recoded data (swap operands, shifts,…)

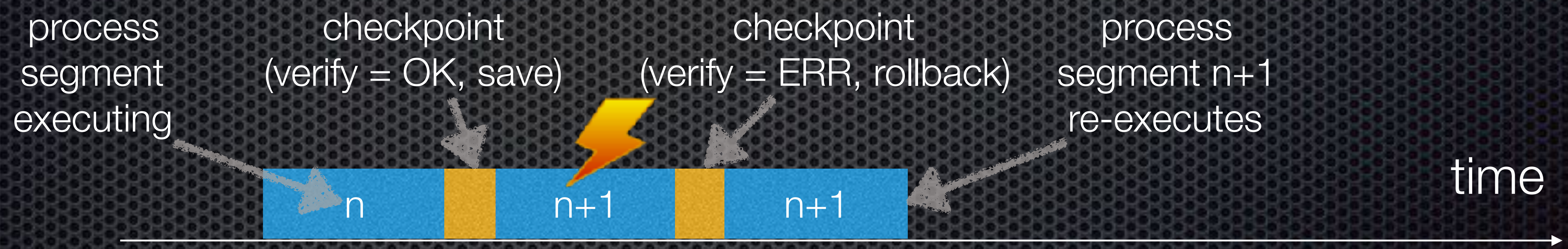
Software fault-tolerance

Software almost inevitably contains defects/bugs

- **Formal proof** of correctness:
  Not practical for large code bases…
- **Instead, use “software fault-tolerance”:**
  - acceptance tests, timing checks:
    output in range, in time, inputs in range
  - single-version vs. N-version programming:
    run a number of versions, developed by independent teams

Checkpointing

- Long running applications may fail at any time: time is wasted if a fault is only detected at the end
- **Checkpoint**: a snapshot of the process state (everything needed to restart a process from that state)



- Verify: use an oracle (acceptance tests) or run the same segment on several processors
- Issues: how many? where? overhead? distribution?…

There's More… (selfstudy for interested)

- Fault causes and fault models
- Hazard analysis
- Standards and regulations (IEC 61505, SIL)
- Fault-tolerant networks
- …

Bibliography

- Fault-Tolerant Systems, I. Koren and C. M. Krishna, Morgan Kaufman, 2007 (popular textbook)
- Fundamental Concepts of Dependability, A. Avizenis, J. C. Laprie, et al., 2001 (short article)
- Safety Critical Computer Systems, N. Storey, 1996