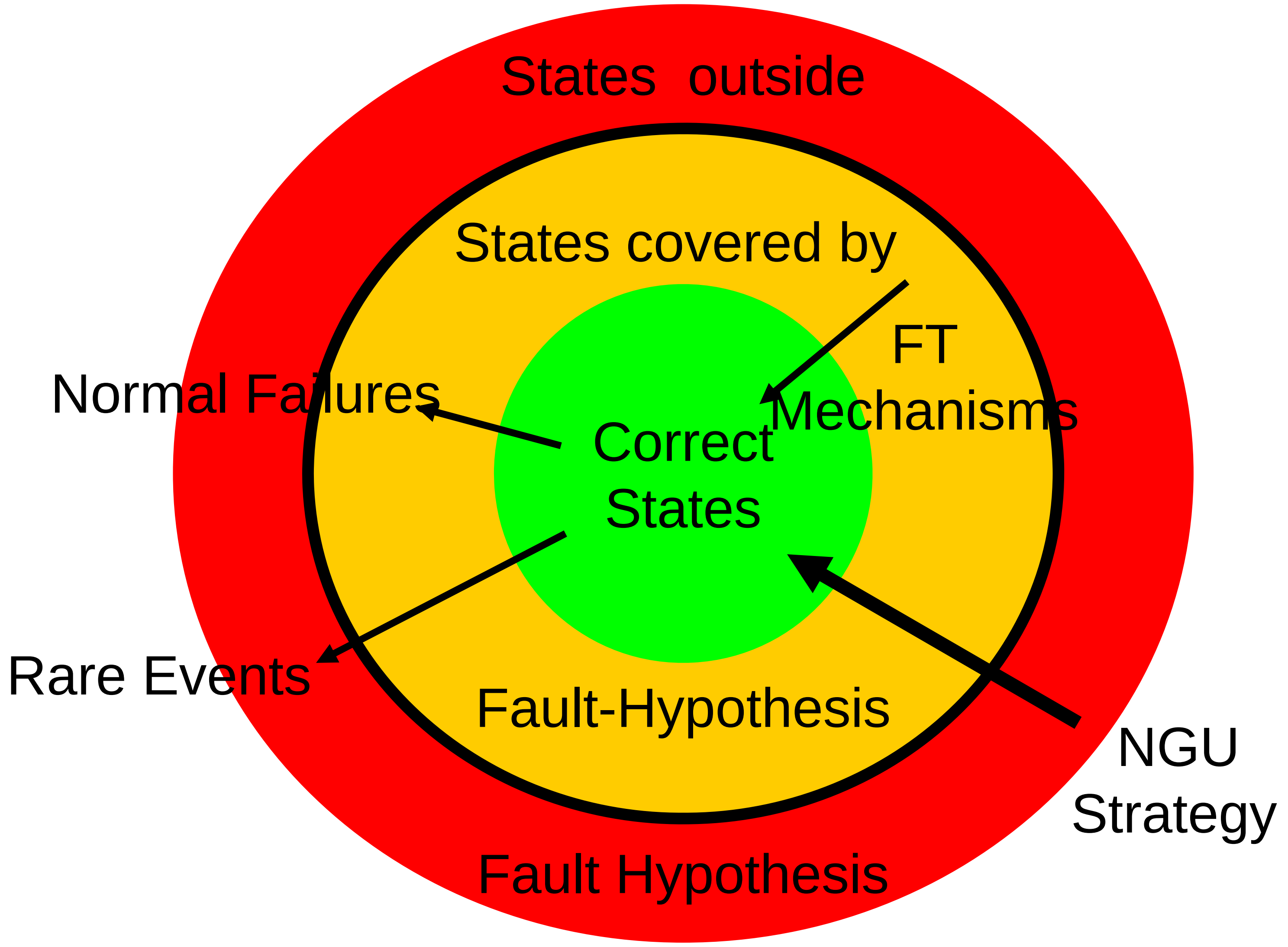
Fault Tolerance

Outline

- ♦ Failures, Errors, and Faults
- ♦ Fault Models
- ♦ Error Detection
- ♦ A Node as a Unit of Failure
- ♦ Fault-Tolerant Unit
- ♦ Reintegration of a Repaired Node
- ♦ Design Diversity

System States of a FT System



Intrinsic Reliability of a Technology

To increase the reliability of a system beyond the intrinsic reliability of a technology

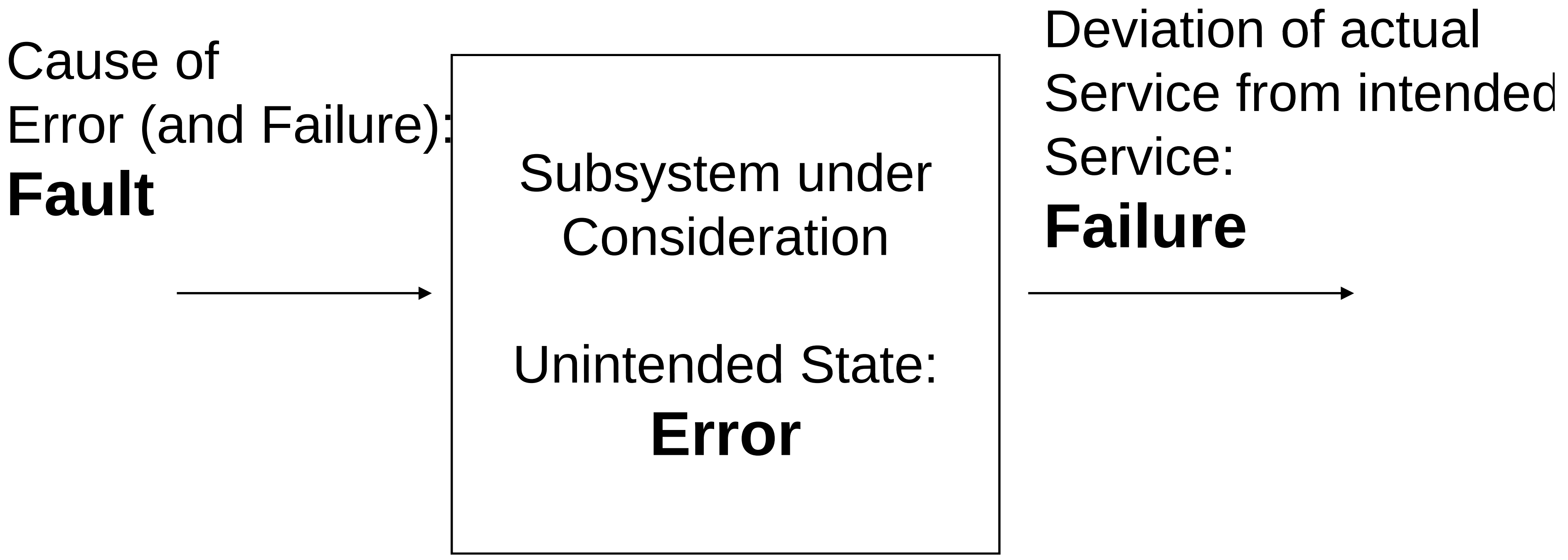
- ♦ use of ultra-high quality components: the cost increase is exponential (e.g., early space program)
- ♦ implementation of fault tolerance

Intrinsic VLSI reliability of single chip microcontrollers in relation to permanent hardware failures:

$$\lambda = 10 \; ... \; 100 \text{ FITS (Failure/}10^{-9}\text{ hours)}$$

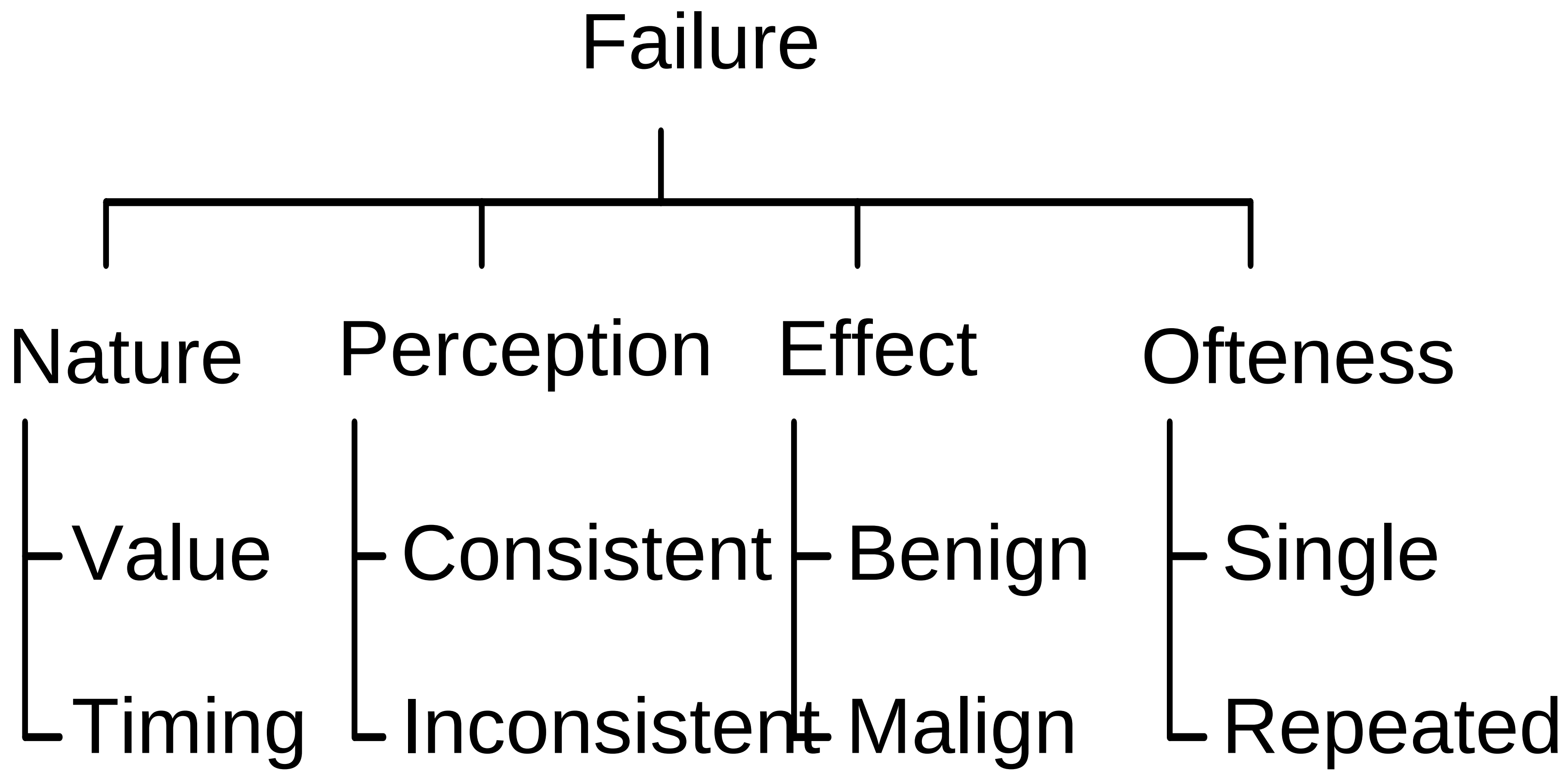
Transient hardware failures: Orders of magnitude larger, depends on the operational environment

Fault-Error-Failure



Faults and Errors are States, Failures are events

Classification of Failures



Component Failures

Crash Failure: After an error has been detected, the component stops silently. It will restart only after the execution of a join protocol.

Omission Failure: Sometimes a result is missing. However, ikf a result is available, it is always as intended.

Consistent Failure: If there are multiple receivers, all receivers see the same erroneous result.

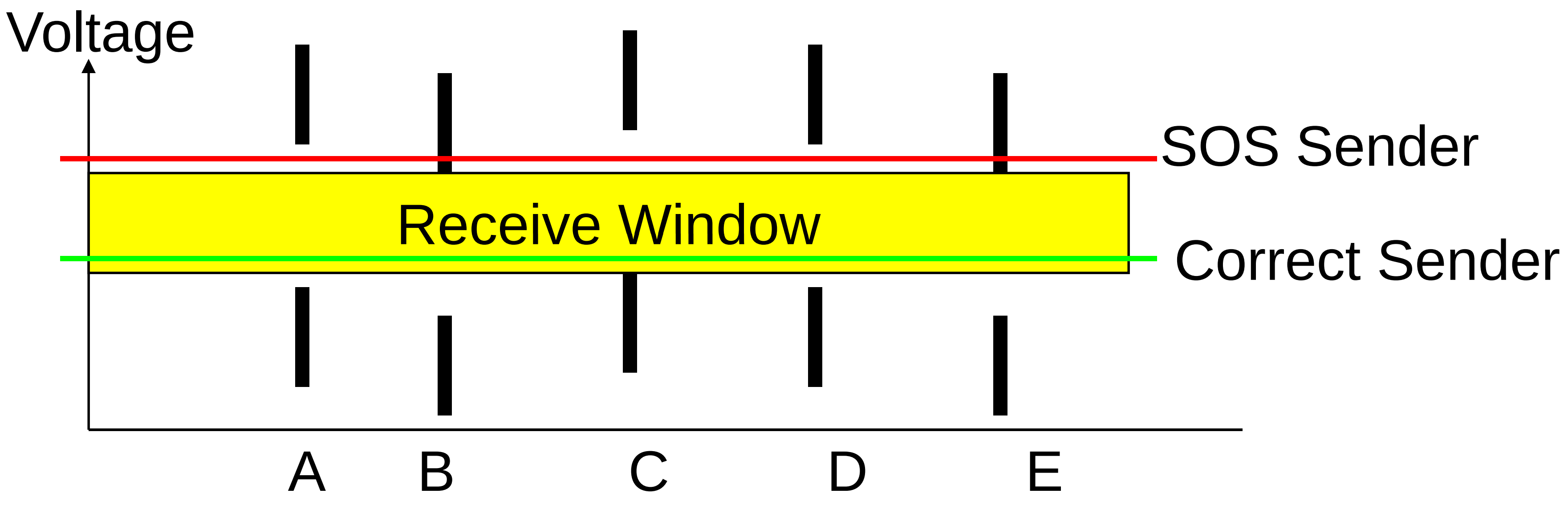
Byzantine (inconsistent, malicious, asymmetric) Failure: In a multiple receiver scenario, the different receivers see differing, possibly incorrect, results.

Slightly off Specification (SOS) Failure

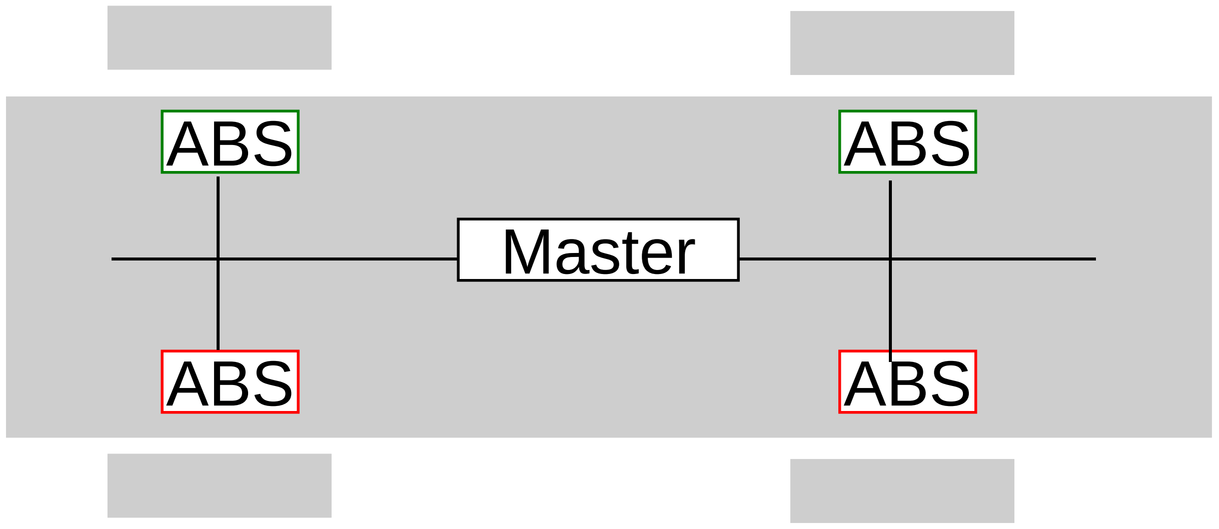
Special type of Byzantine failure:

A component produces an output signal (in the value domain or in the temporal domain) that is slightly outside the specified operating interval.

Some receivers interpret the result correctly, some others cannot interpret the result.



Example: Brake by Wire System



A master with an SOS failure can cause inconsistencies.

Failsilent Component

A component is fail-silent if it either

- ♦ operates correctly by sending correct (both in value and time) messages or
- ♦ sends detectable incorrect messages at the correct instant (e.g. CRC)   or
- ♦ sends no messages at all.

How can we detect if a fail-silent component has an incoming or outgoing link failure?

A component needs a co-operative environment to implement error detection, and thus fail-silence: membership service!

Babbling Idiot

The “Babbling Idiot Failure”, i.e., the transmission of messages at arbitrary points in time, is the most serious node failure in a bus system:

- ♦ Avoidance by hardware means possible, if regularity assumptions (e.g, TDMA) are part of the architecture, e.g. in a TT system
- ♦ Difficult to detect in an ET system, where the *a priori* knowledge of when a node is allowed to send is not available
- ♦ Never grant extra transmission privileges

Error

An error is an unintended state, e.g., corrupted h-state data in memory

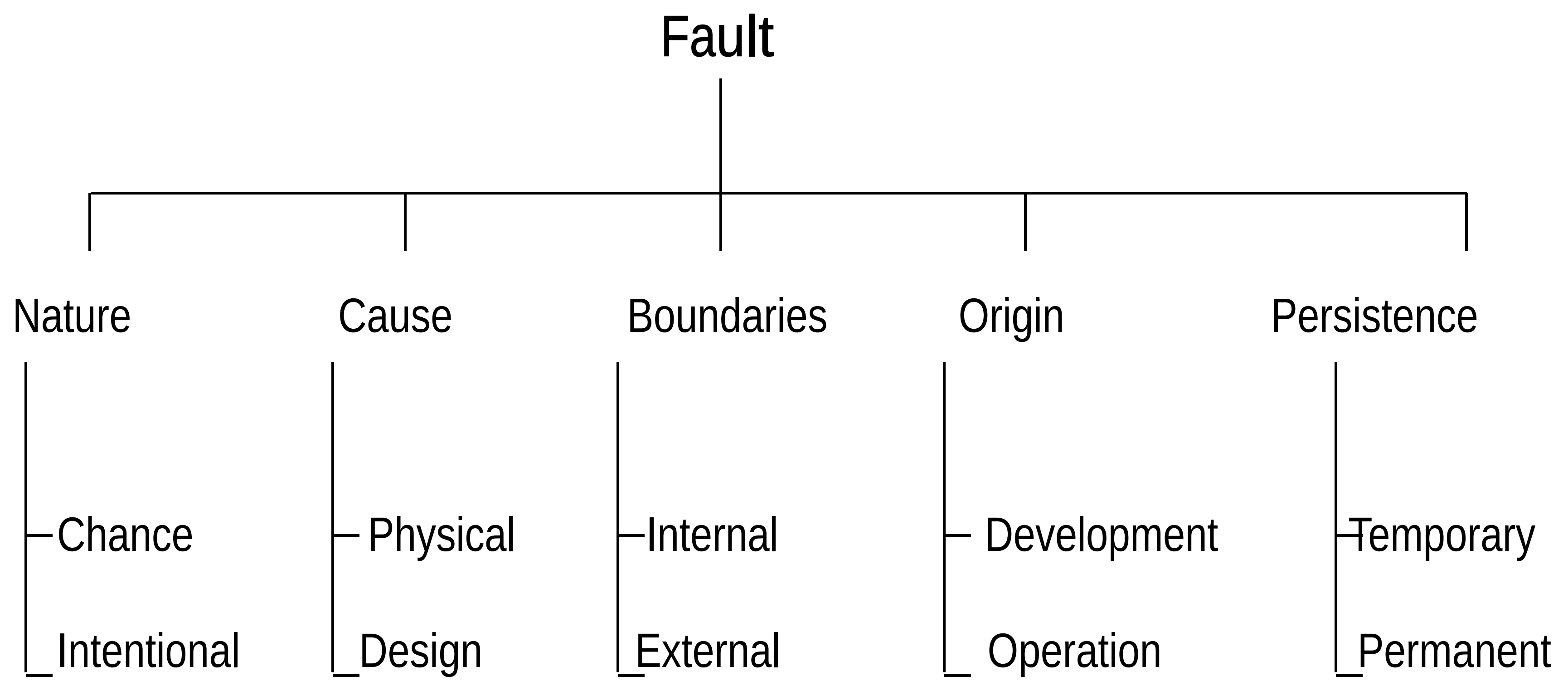
If activated, an error leads to a failure, otherwise it remains *dormant*.

An error is the consequence of a fault.

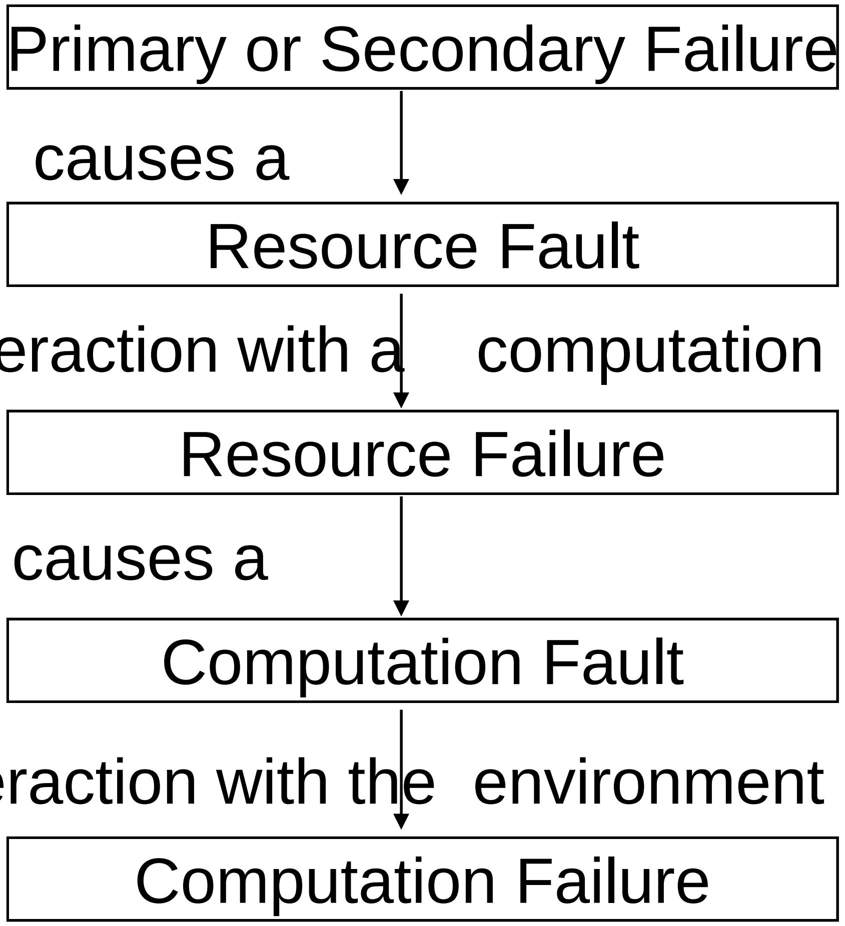
The probability that an error is detected, provided it is present, is called the *error detection coverage*.

The time interval between the start of an error and the detection of an error is the *error detection latency*.

Classification of Faults



Failure - Fault (FF) Model



Programming Failure
Mistake during Hardware Design
Previous Computation Failure

Program Fault, Processor Fault,
h-state error, input error

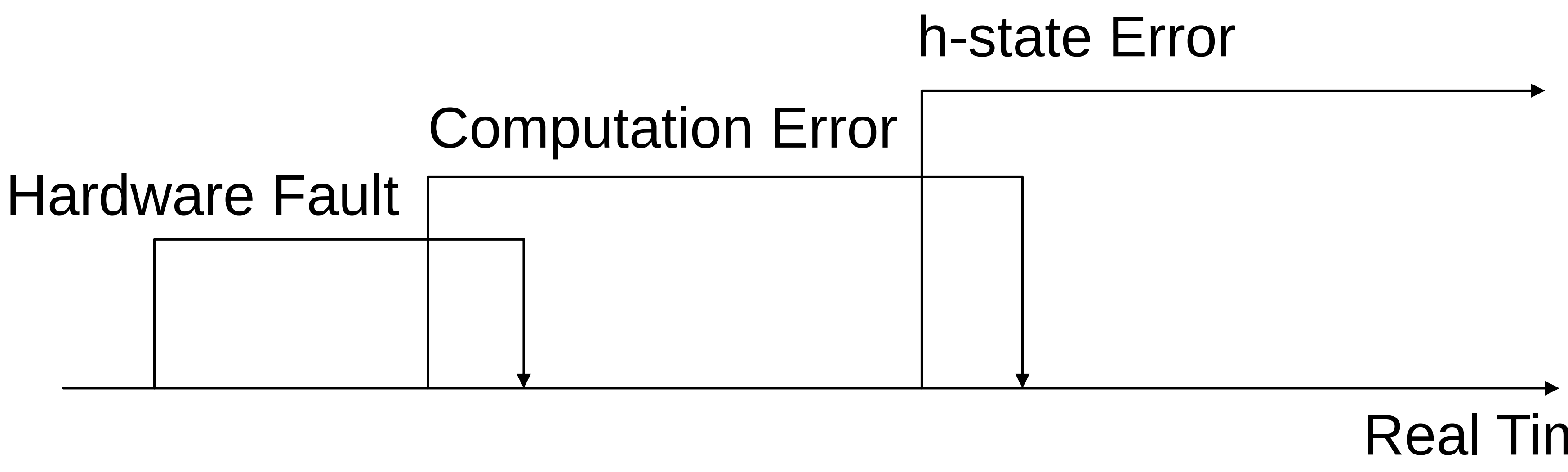
Interaction with a computation can cause a

Program Failure, Processor Failure
h-state failure, Input failure

Erroneous

Interaction with the environment or the h-state can cause a

Transient Fault/Permanent Fault



The interaction of a transient hardware fault with the h-state causes a permanent h-state error: h-state erosion

Four Universe Model

Level IV

IV External Universe: System as perceived by the user: service failure at the user/system interface. ED: End-to-End

Level III

III Informational Universe: System as perceived by the programmer: incorrect data structures. ED: Run time assertions

Level II

II Logical Universe: System as perceived by the logic designer: stuck at logic fault. ED: Error detecting codes

Level I

I Physical Universe: System as seen by the analog engineer: false signal levels. ED: Monitoring of voltage levels

ED: Error Detection

Fault Hypothesis

The fault hypothesis partitions the fault space into two sets

- ♦ Level-1 faults: this is the set of faults that will be tolerated by the fault-tolerance mechanisms.
- ♦ Level-2 faults: this is the set of fault that will ***not*** be tolerated by the fault-tolerance mechanisms. These faults must be ***rare*** events.

If there is no precise fault hypothesis available, it is impossible to test the proper behaviour of the fault-tolerance mechanisms.

If during the test and installation phase it is found out that level-2 faults are not rare events, then there exists a fundamental design problem:

- ♦ Either the fault-hypothesis is wrong
- ♦ Or the implementation is deficient.

Fault Hypothesis **and** ***Assumption Coverage***

The Fault-Hypothesis states the *assumptions* about the types and number of faults that a fault-tolerant system must tolerate.

The *assumption coverage* states to what extent are these assumptions met by *reality*.

The *assumption coverage* limits the dependability of a perfect fault-tolerant system.

Without a precise specification of the Fault-Hypothesis it is *impossible*

- ♦ to investigate whether the assumption coverage is realistic.
- ♦ to test the correctness of the fault-tolerance mechanisms

Example

The *fault hypothesis* assumes that a particular transient failure will happen with an MTTF of 10000 hours and that it will take 10 msec to recover from this failure.

These data are an important imput to the reliability model.

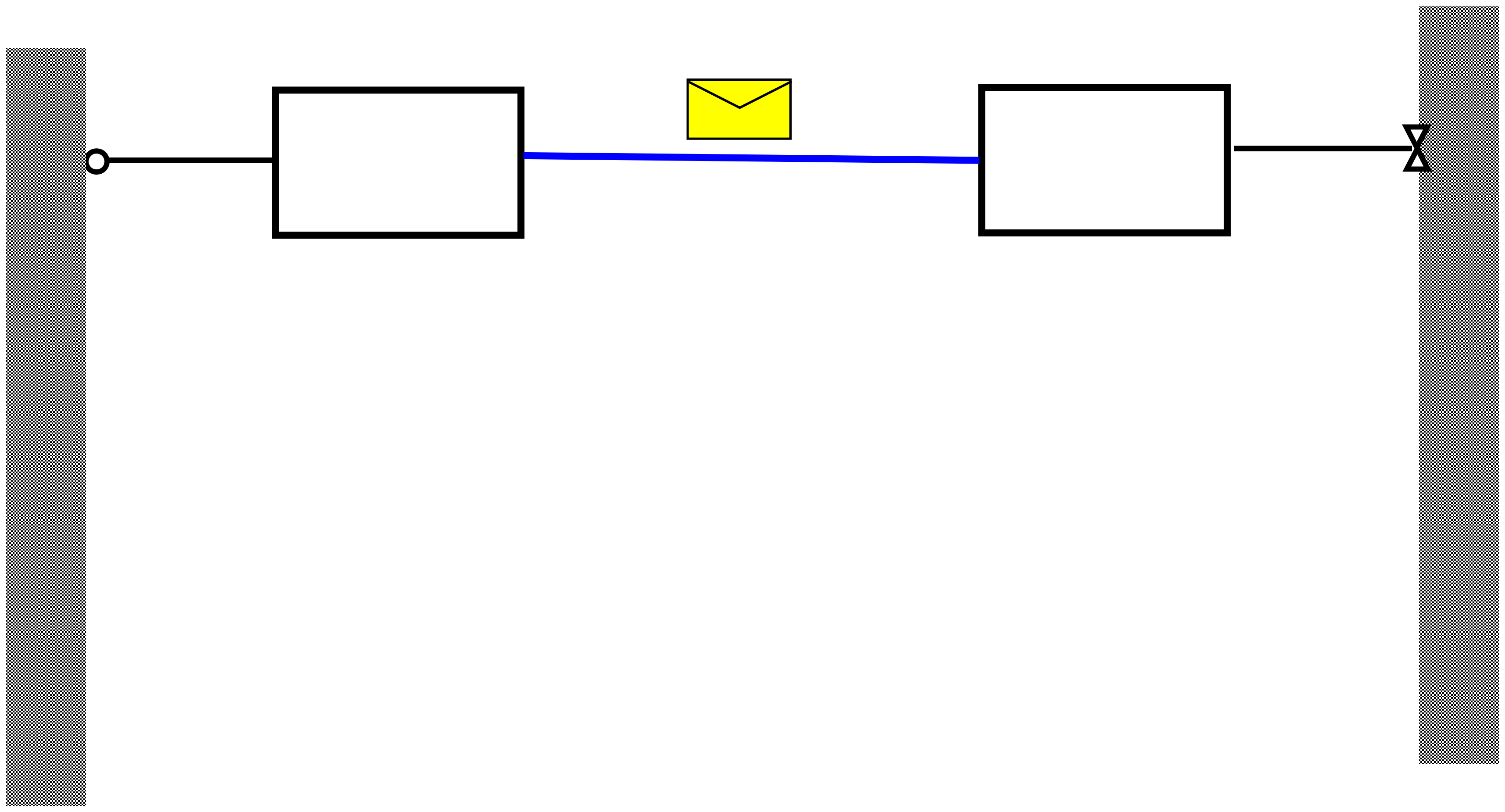
Field data show that such a failure happens every 100 hours and that it takes 50 msec to recover from this failure.

The *assumption coverage* of the fault hypothes is thus a problem. There are two options

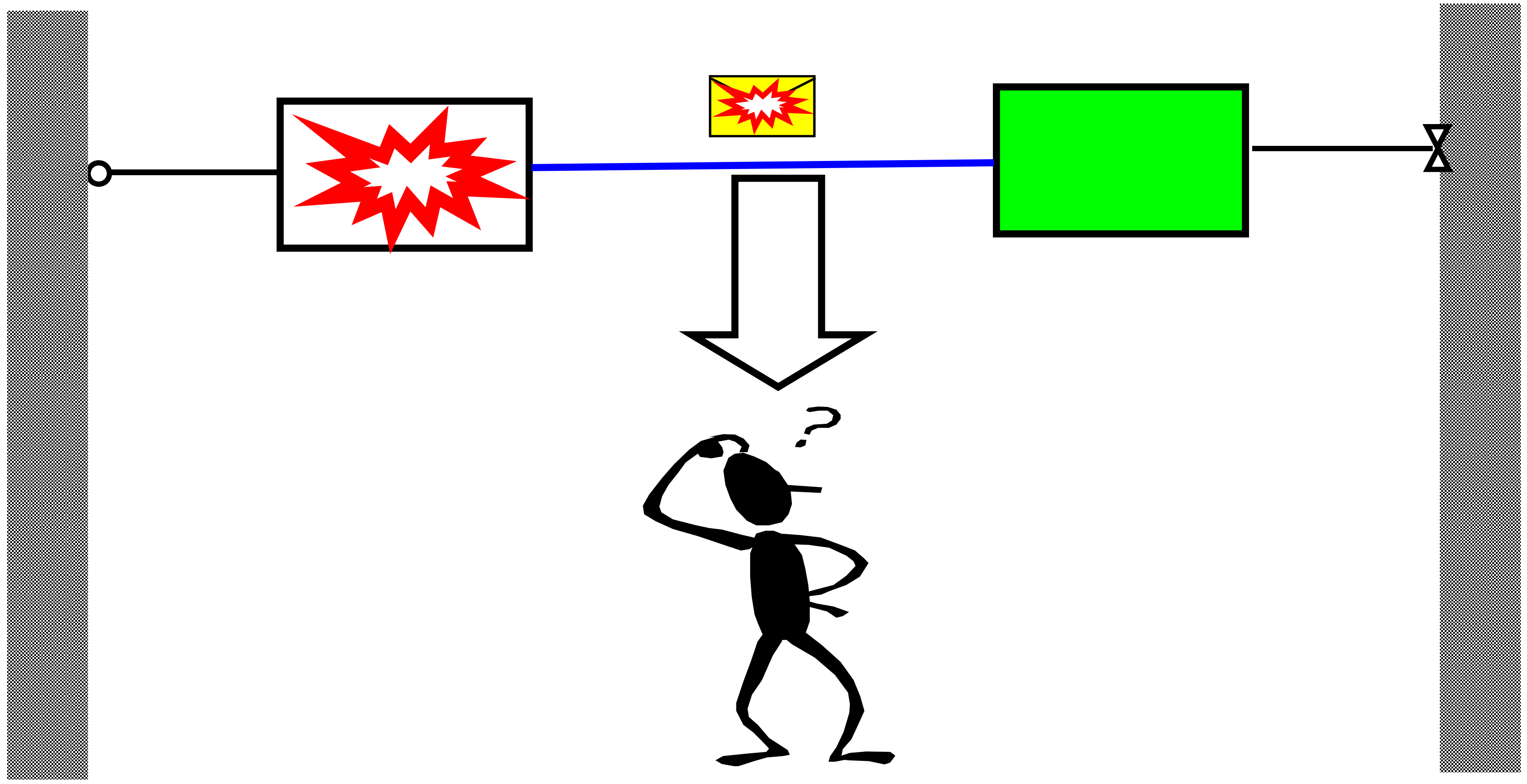
- ♦ Redo the analysis to demonstrate that the overall dependibility goal is still achieved.
- ♦ Modify the design

Case study: Sizewell B Nuclear Reactor in England

An Example: A Non-fault-tolerant Structure



Fall-back to Human Control in Case of Failure

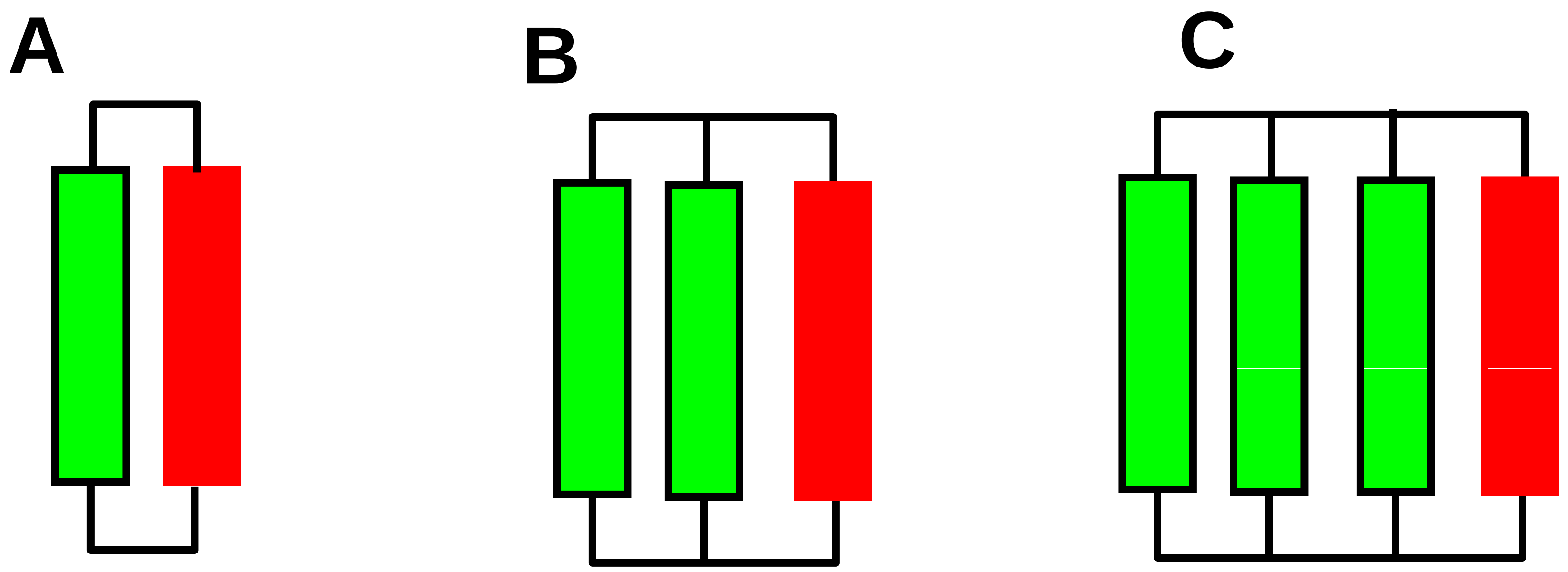


Can humans manage the ***functional difference*** between the ***computer control system*** and the ***manual backup system***?

Specification of the Fault Hypothesis

1. **Unit of Failure:** What is the Fault-Containment Region (FCR)?
2. **Failure Modes:** What are the failure modes of the FCR?
3. **Frequency of Failures:** What is the assumed MTTF between failures for the different failure modes?
4. **Detection:** How are failures detected? How long is the detection latency?
5. **State Recovery:** How long does it take to repair corrupted state (in case of a transient fault)?

Restricted Failure Modes of SoCs?



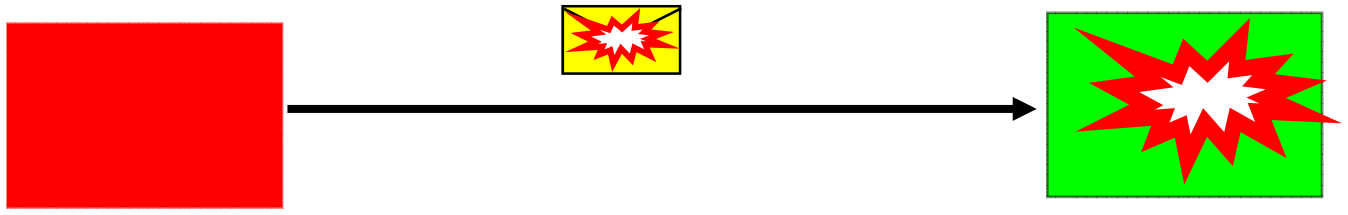
assumption
fail-silent
k+1

assumption
synchronized
2k + 1

no assumption
(arbitrary)
3k + 1

What is the *assumption coverage* in cases A and B?

Error Propagation in Case of *Non-Fail-Silence*



A ***non-fail-silent*** faulty component can corrupt another good component by sending a faulty message:
- Message error in the value domain
- Message error in the temporal domain

If we do not detect and intercept the faulty messages, then a ***single*** that is not properly contained can lead to ***multiple component failur*** thus compromising the ***independence requirement***.

Cognitive Resilience

The human perception system can tolerate many failures that are non acceptable from a logical correctness point of view. Example Pixel processing in multimedia applications.

We can exploit these human perception characteristics to improve user satisfaction despite the occurrence of failures.

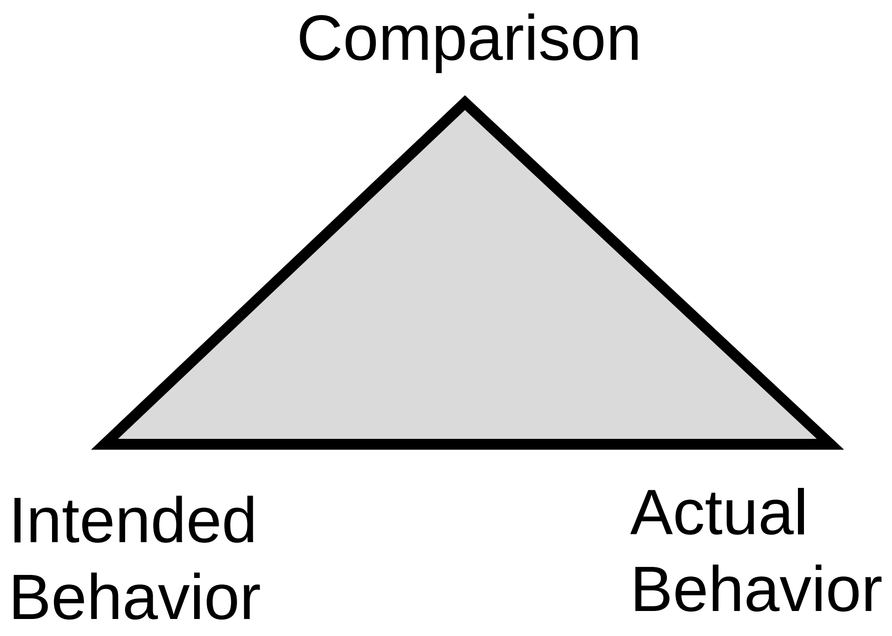
Partition tasks in those that critical and those where the human perception system can tolerate the failures

Error Detection

We distinguish

- ♦ Detection of errors in the value domain
- ♦ Detection of errors in the time domain (the only error type that can occur if every component and communication is fail silent!)

Error detection requires redundant information--how should the system behave?



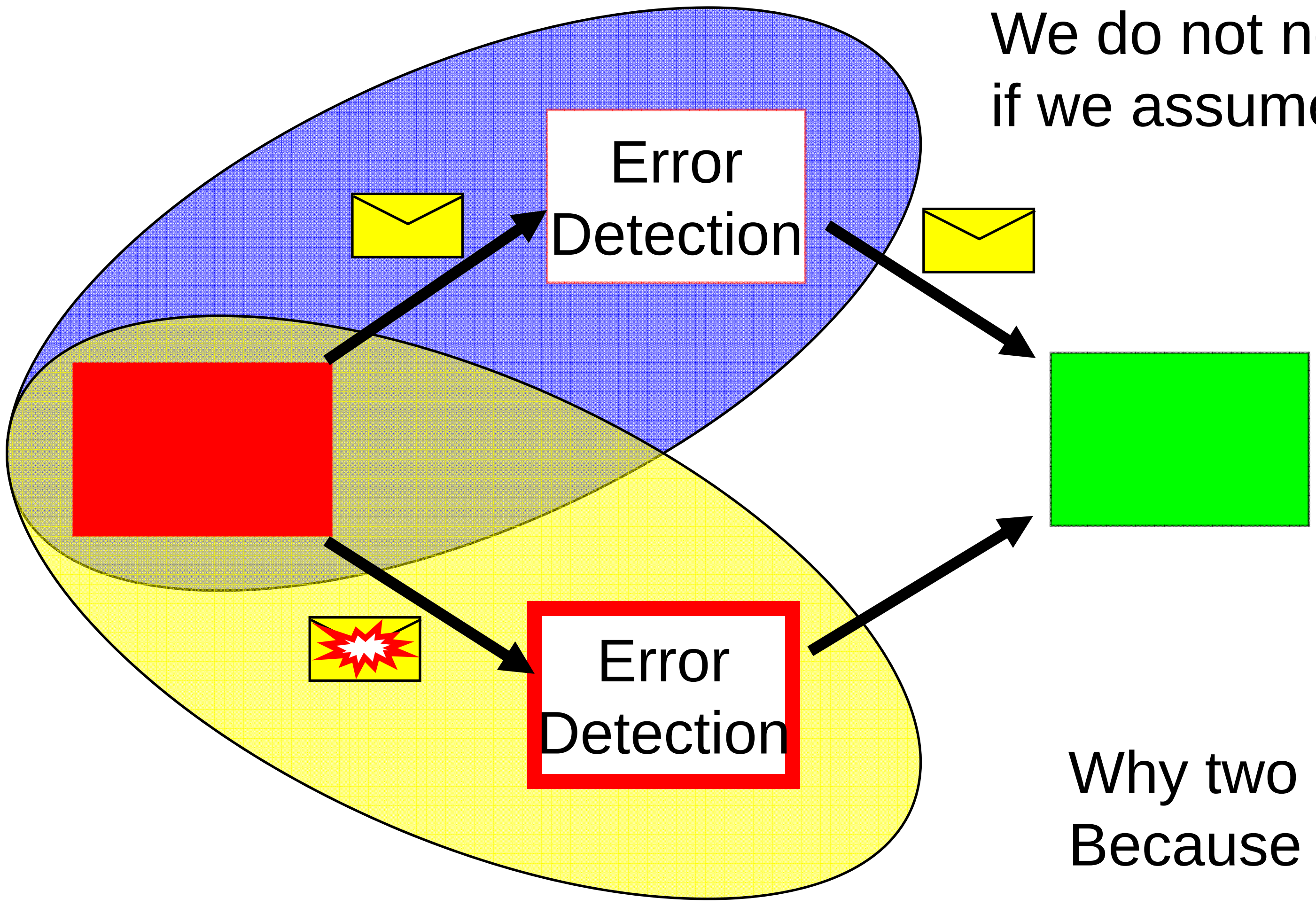
Error Containment Region (ECR)

In a distributed computer system the consequences of a fault, the ensuing error, can propagate outside the originating FCR (Fault Containment Region) by an **erroneous message** of the faulty node to the environment.

- ♦ A propagated error **invalidates** the independence assumption.
- ♦ The error detector must be in a **different FCR** than the faulty unit.
- ♦ Distinguish between architecture-based and application-based error detection
- ♦ Distinguish between error detection in the **time-domain** and error detection in the **value domain**.

Since an Error Containment Region requires at least two independent FCRs, a single die cannot form an Error Containment Region.

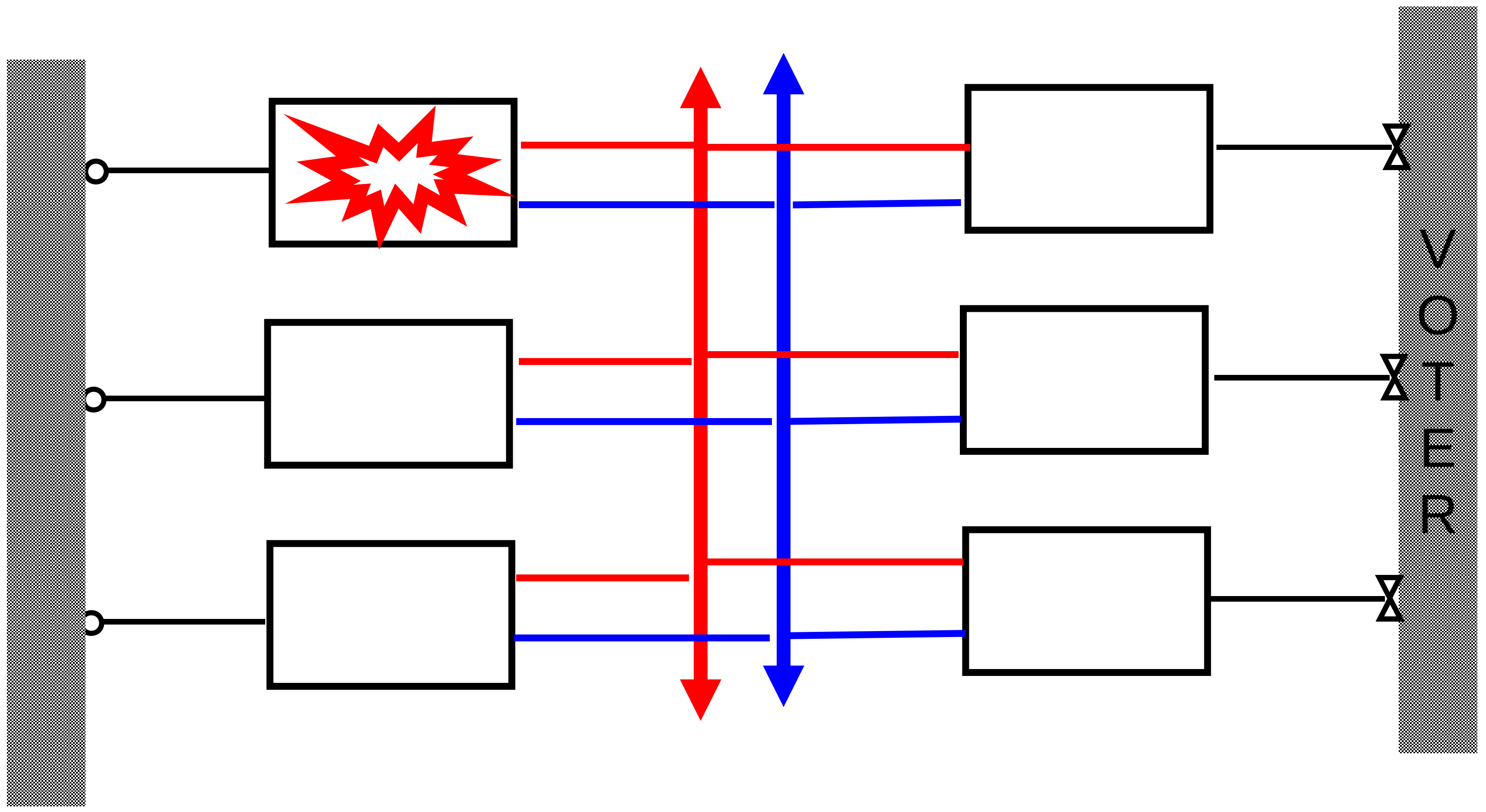
Fault Containment vs. Error Containment



We do not need an error detector
if we assume fail-silence.

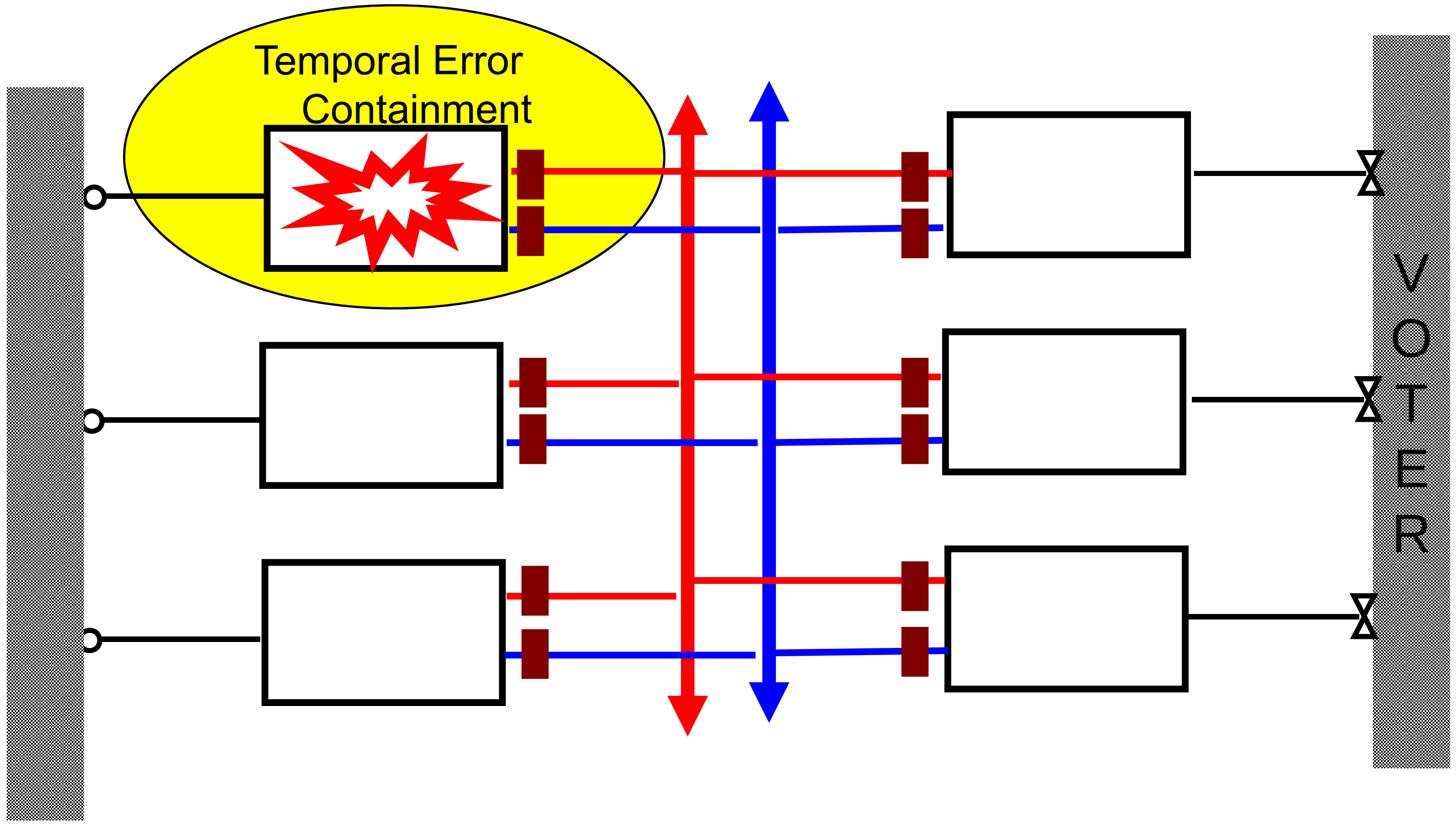
Why two Channels?
Because one can fail

TMR Structure with Bus

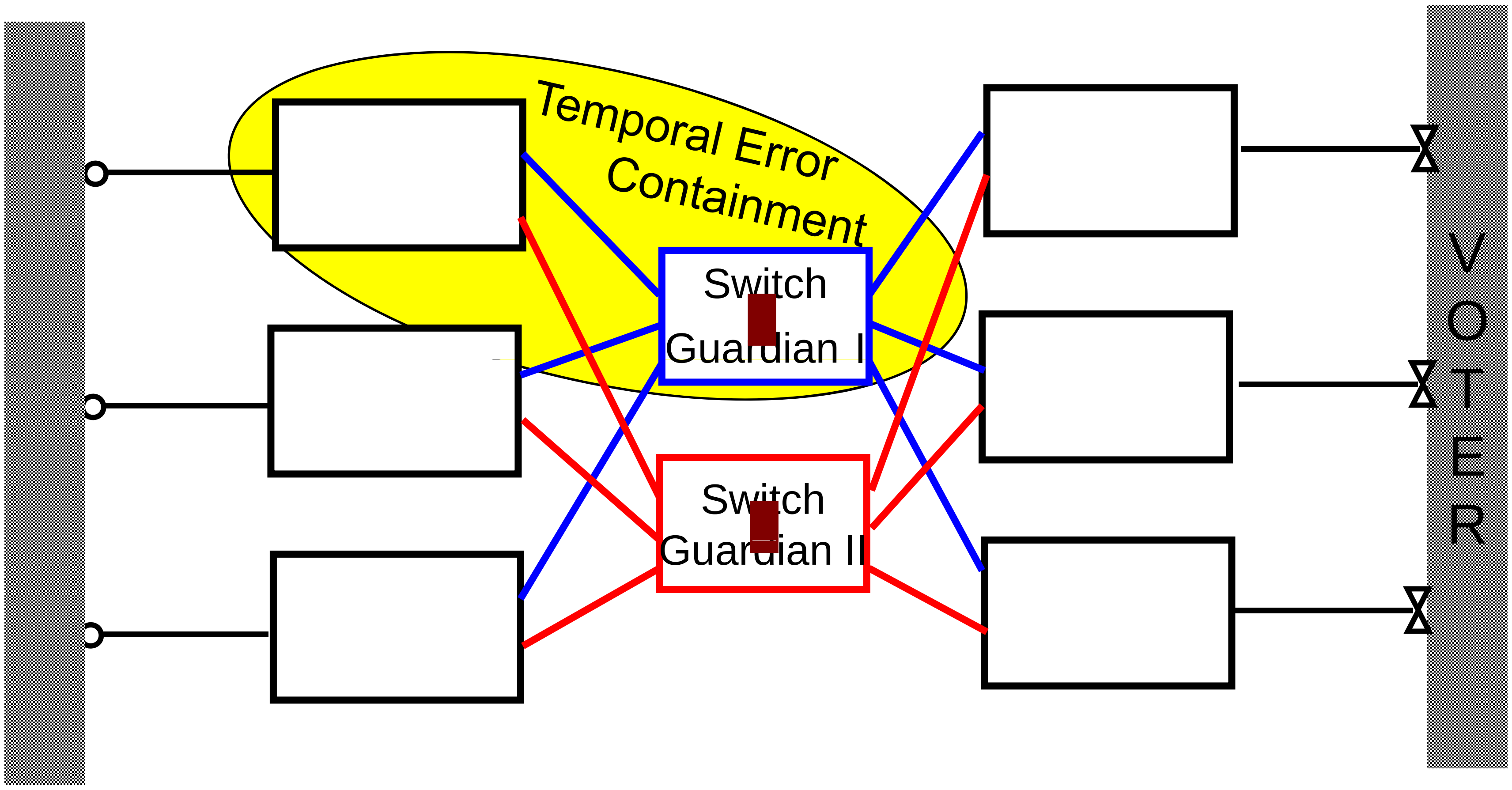


A *babbling* node can kill all communication among good nodes

TMR Structure with Bus and Guardians



A TMR Structure with a Guardian in a Star



3 Fault Containment Regions and 6 Error Containment Regions (temporal

From **Federated** to **Integrated** Architectures

Evidence by Fault Injection

Millions of fault injection experiments have been carried out in the PDCS, TTA and FIT project over a period of more than ten years:

- Software based (TU Vienna, Austia)
- Alpha Particle (Chalmers University, Sweden)
- VLSI-model based (Univ. of Valencia, Spain, Carinthia Tech, Austria)
- Pin Level (LAAS, Toulouse,France, Univ. of Valencia, Spain)

Conclusions:

- Guardians are needed to avoid error propagation in the temporal domain
- Guardians must be fully independent: star coupler

Results are documented in the open literature

What are the Experimental Results?

We observed the following *orders of magnitude* of *fail-silent* versus *non-fail-silent* failures of components:

Error Detection in the temporal domain	**fail-silent to non-fail-silent failures**	**Experimental Evidence**
No Error Detector	**50:1**	FI Measurements in PDCS Project
Local Guardian	**1000: 1**	Fault Injection FIT Project
Autonomous Central Guardian	**no non-fail silent failure observed so far**	Fault Injection in FIT/NEXT TTA

Experimental Results:

TTP/C-C1-based hardware prototype with XILINX 600k FPGA (tested by IST project FIT):

Heavy Ion Experiments (at Chalmers):

Bus topology with guardians: 37036 faults--78 fail silence violations (0.21 %)

Star topology with guardians: 26600 faults-- 0 fail silence violation

Software Implemented Fault Injection (Vienna):

Bus topology with guardians: 562122 faults--14 fail silence violations (0.02 %)

Star topology: 541744 faults-- 0 fail silence violation

To be published at DSN, San Francisco, June 2003

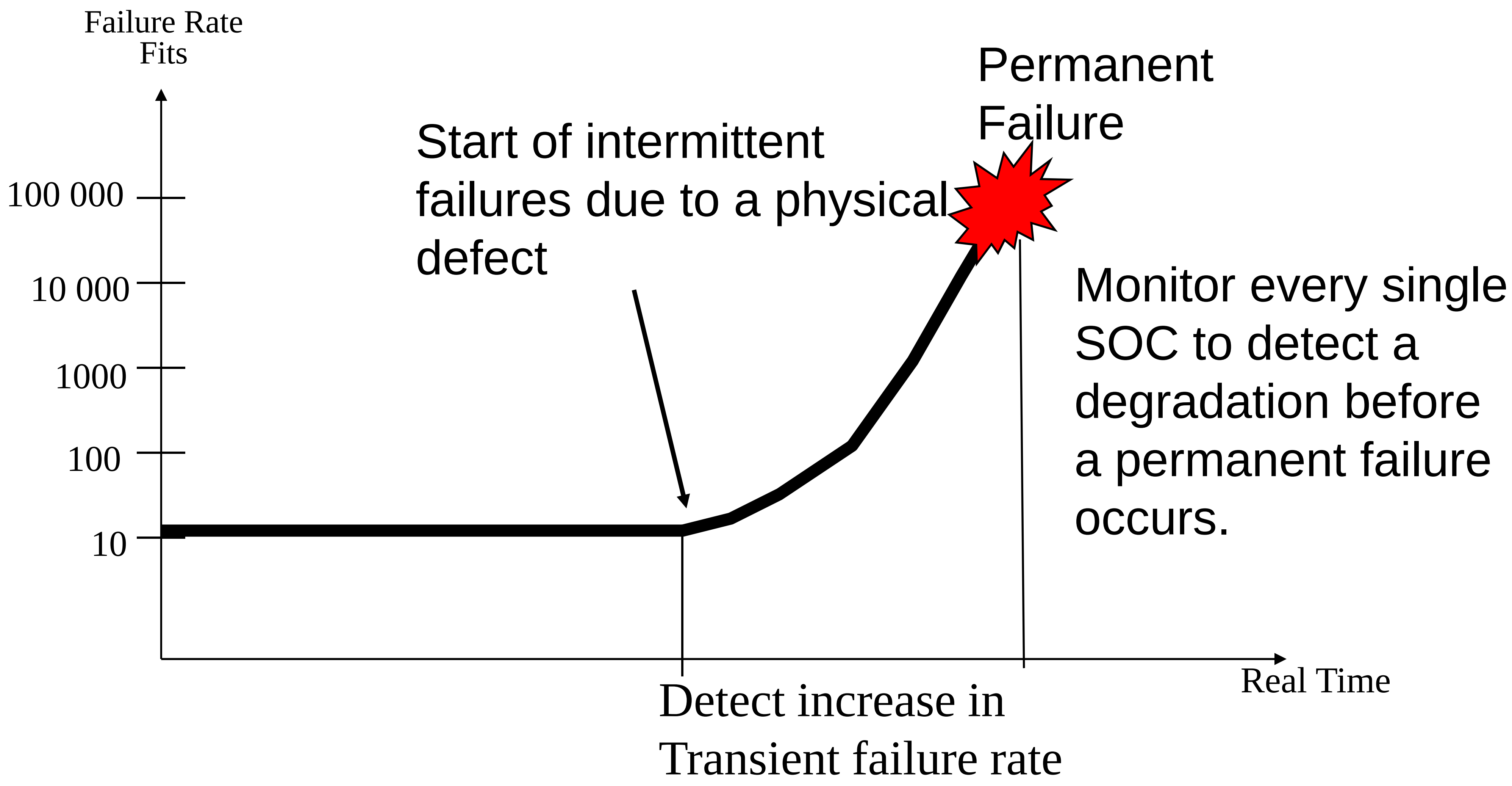
Formal Verification using Model Checking (SAL, UPPAAL2k) and Theorem Proofing (PVS) is ongoing in the NEXT TTA Project.

Assumption: Frequency of Failures

Type of Failure	Failure Rate in Fit	Source
Transient Node Failures (fail silent)	> 1 000 000 Fit (MTTF = 1000 hours) (depends on environment)	Neutron bombardment Aerospace
Transient Node Failure (non-fail silent)	> 20 000 Fit (MTTF= 50 000) about 5 years	Fault Injection Experiments
Permanent Hardware Failures	< 100 Fit (MTTF= 10 000 000 I.e. about 1000 years)	Automotive Field Data

Basic frequency assumption in Aerospace: Any chip can fail in an arbitrary failure mode with a probalitity of 10^{-6}.

Intermittent Failures: Increase of Transients



Consequences for the Architecture

In a safety-critical application an SoC (System on Chip) must be considered to form a *single unit of failure*, *ie.*, *a single FCR* that can fail in an ***arbitrary*** failure mode because of:

- Same Physical Space (Physical Proximity Failures)
- Same Wafer Production Process and Mask (Mask Alignment Issues)
- Same Bulk Material
- Same Power Supply and Same Earthing
- Same Timing Source
-

Although some of these dependencies can be eliminated, others cannot.

We cannot assume an *independent* error detector on the same die.

Never-Give-Up (NGU) Strategy

In a fail-operational environment (e.g., brake-by-wire system) the system must try to reach a safe state, even if faults outside the specified fault-hypothesis have occurred. The sequence of actions that handles the faults outside the fault-hypothesis is called never-give-up strategy.

In order to realize a NGU strategy, the control system must

- ♦ Contain an algorithm that detects a violation of the fault-hypothesis.
- ♦ Activate the NGU strategy promptly to bring the system into the safe state.

In the TTA the membership algorithm, executed in the hardware of the TTP controller, detects a violation of the fault hypothesis.

Ariane 5 rocket failure

The failure to provide an NGU mechanism was the cause of the well-publisized ARIANE accident on June 4, 1996 . The designers of the ARIANE 5 rocket assumed that the software is free of design faults and therefore no mechanisms for handling software failures were provided (no level-2 fault analysis). The occurrence of the same software failure on replicated hardware components caused the loss of the rocket.

NGU Strategy in the TTA

The membership algorithm of TTP informs all nodes after every TDMA cycle (i.e., after about 1 msec) about the state of each node.

As soon as the fault hypothesis is violated, (or the node loses its membership), the NGU strategy is activated.

Depending on the application, different NGU strategies are possible:

- ♦ Restart and reintegration of a failed node
- ♦ Restart of a complete cluster
- ♦ Selection of alternate set-points

Faults in Automotive Environment

Three important fault classes must be considered in the automotive environment:

(i) internal physical faults caused by single event upsets (SEUs). *Solution*: replication, fast restart and reintegration.

(ii) external physical faults caused by the environment (EMI, power surges). *Solution*: Quality engineering (replication not appropriate)

(iii) design faults in software (and possibly in hardware). *Solution*: Reduction of complexity, disciplined design and testing, plausibility checks, formal validation

Fast and Consistent Detection of Transients

In many application the fast and consistent detection of transient faults provides more than half of the solution!

Examples

- ♦ Distributed ABS: If within a very short latency, a clean (fail-silent) failure of one node is consistently detected by the other nodes, all other nodes can disengage the ABS immediately to provide the normal braking service consistently.
- ♦ Stop light at the tail: If the local node detects swiftly the loss of communication to its client, it can enter the safe state (stop light on) autonomously. If the client recovers within 100 msec, it can turn it off again.

How should a system behave?

- ♦ Specify assertions about the expected correct behavior (e.g., plausibility checks).
- ♦ The most important information for error detection comes from the *a priori* common knowledge about the regularity of the behavior (e.g., a message must arrive every full second).
- ♦ Provide redundant channels and compare the results!
- ♦ The failure of a fail-silent system can only be detected in the temporal domain.

Levels of Error Detection

Error Detection Mechanism can be situated at any level of the four universe model:

- ♦ An error will only be detected if it occurs at the considered level or below
- ♦ Experimental data has shown that End-to-End Error detection at level IV (application level) is most effective and is absolutely necessary in ultra reliable systems
- ♦ Intermediate level error detection is useful for diagnosis if the system operation is not very reliable--however in an intrinsically reliable architecture these intermediate level error detection mechanisms are of questionable cost-effectiveness.

Signature Analysis

Signature Analysis is an ED method in the value domain:

- ♦ The characteristic features of a data stream (message) are captured in a (redundant) signature
- ♦ Comparison of the data stream with the signature for error detection
- ♦ Faulty data stream must produce a different signature then a correct one (with very high probability), i.e. all important features of the data stream are to be considered.
- ♦ Very effective for detection of communication errors and storage errors (e.g., CRC, parity codes, check digits)--should be used to protect the i-state.

Signatures in the TTA

The Time Triggered Protocol TTP makes extensive use of signatures:

- ♦ The CRC polynomial is calculated over the message contents concatenated with the c-state of the controller
- ♦ A second level End-to-End signature in class C messages reflects, in addition to the message contents, a message key and the point in time when the message has been generated.
- ♦ Fault injection experiments have shown that this elaborate signature calculation is needed to get a very high coverage (>99 %) of EMI induced failures

Detection of Transient Value Errors

Transient errors have the following characteristics:

- ♦ The critical transient is between 10 and 100 machine cycles.
- ♦ If they last longer than 100 machine cycles, the probability of detection by built in hardware mechanisms is high.
- ♦ Duplicated (time-redundant) execution of a task at slightly different times and comparison of the results is an effective technique for transient error detection.
- ♦ The redundant task execution can be combined with the signature analysis mentioned previously.

ED in the Value Domain: Other Techniques

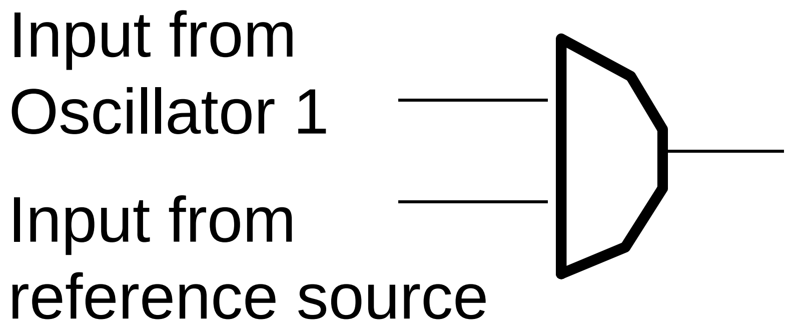
- ♦ Plausibility Checks, such as
  - application specific invariants (very effective)
  - trend analysis (requires h-state)
- ♦ Structural checks, e.g.,
  - run time assertions
  - robust data structures
- ♦ Comparison of diverse computations
  - time redundancy
  - resource redundancy
  - different software versions

ED in the Time Domain: Babbling Idiot

In a TT System, the time interval when a component is allowed to send a message on the bus is known apriori. This knowledge can be used to suppress erroneous control signals.

A redundant reference signal can be generated by:

- ♦ an independent oscillator
- ♦ a fixed delay after the arrival of a previous message



There is no similar error handling technique possible in ET systems.

Bus Guardian in the TTA

In the TTA a bus guardian is provided to detect the babbling idiot behavior of a node:

- ♦ In order to be independent, must be an independent unit (chip) with its own power supply, oscillator and knowledge base.
- ♦ Independent Bus Guardian must have its own distributed clock synchronization.
- ♦ It is most economically to integrate the Bus Guardian into a (replicated) central star coupler.

If the Bus Guardian is integrated into the node and shares the node's oscillator and power supply, it is not fully independent.

ED in the Time Domain: Other Techniques

- ♦ Monitor the maximum execution time of operating system processes and application processes (possible in ET and TT systems)
- ♦ In TT-systems: use the information about the temporal regularity of the TDMA cycle to provide a distributed membership service with short latency error detection (as in TTP)
- ♦ Use *a priori* application information about what has to happen at a specific point in time or after specified intervals.

Fault-Tolerant Unit (FTU)

A fault-tolerant unit (FTU) is a set of actively redundant components that provide a fault tolerant service to its environment:

- ♦ FTUs have to receive identical input messages in the same order
- ♦ FTUs have to operate in replica determinism
- ♦ The output messages of FTUs should be idempotent
- ♦ As long as a defined subset of the components of the FTU is operational, the FTU is considered operational

FTUs provide the continuous service by fault masking.

Types of Redundancy

Cold Standby Redundancy

At any single point in time, only a single component provides the service. If the service provider fails, the failure has to be detected by a failure detector and a spare component is started.

Hot Standby Redundancy

At any single point in time, only a single component provides the service. If a failure is detected, an active spare component replaces the failed component.

Active Redundancy

Two or more components provide the service concurrently.

What is Needed to Implement TMR?

What architectural services are needed to implement Triple Modular Redundancy (TMR) at the architecture level?

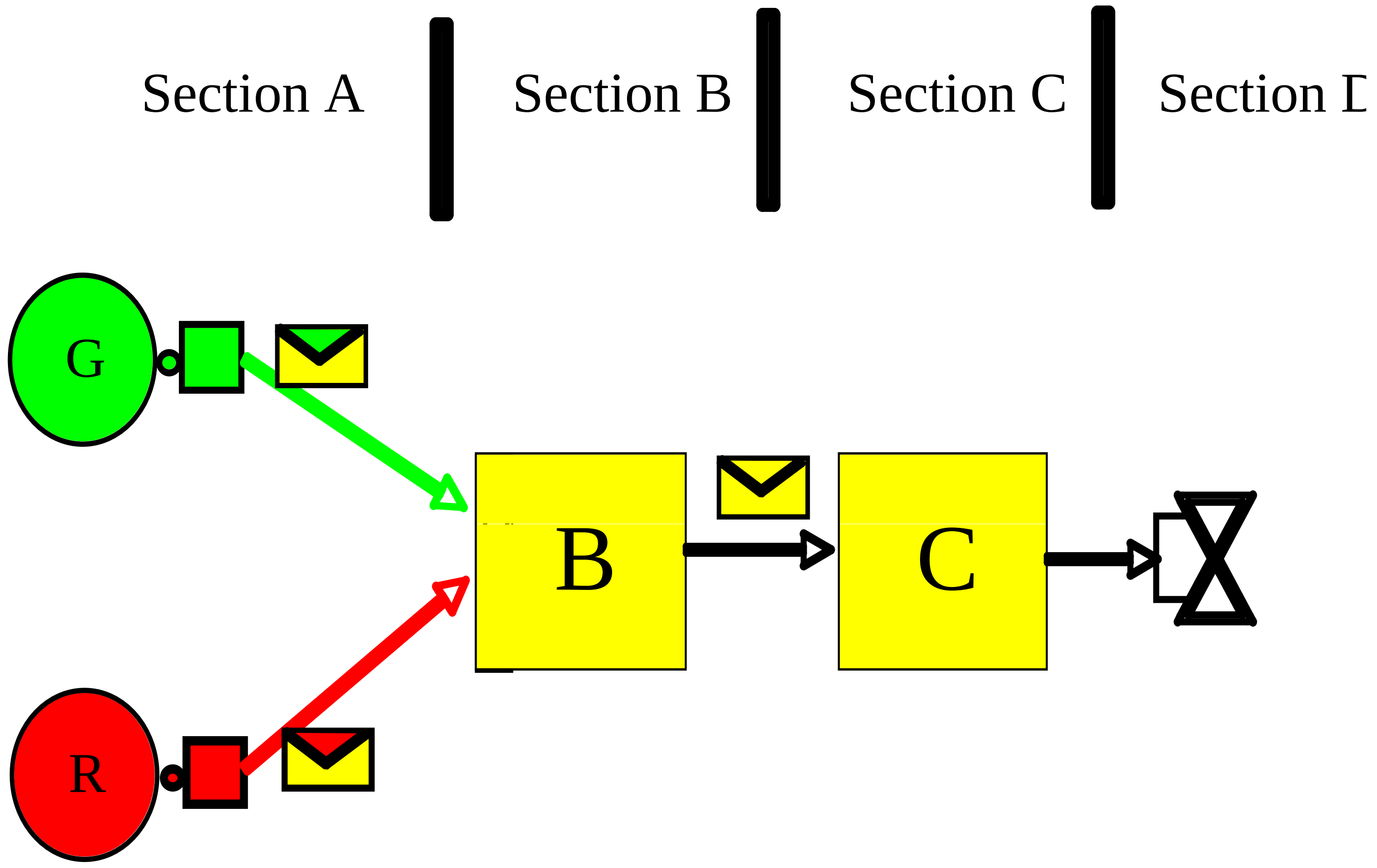
- ♦ Provision of an Independent Fault-Containment Region for each one of the replicas
- ♦ Synchronization Infrastructure
- ♦ Multicast communication
- ♦ Replicated Communication Channels
- ♦ Support for Voting
- ♦ Timely and Deterministic Operation

Requirement for Active Redundancy: *Determinism*

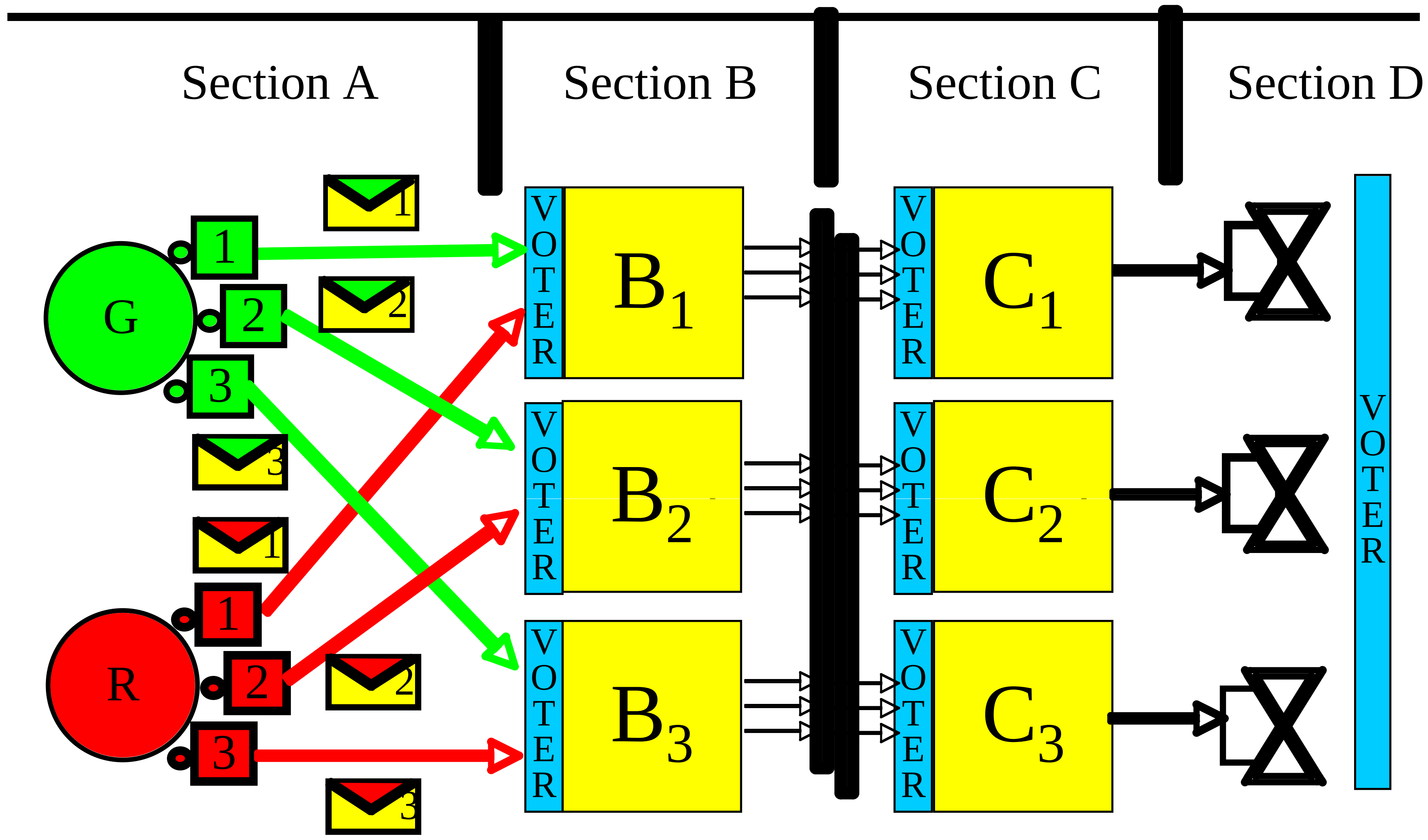
If voting at the logical level is performed correct active channels must produce the same result (same bit vector)

- ♦ Input: Sparse time and agreed values--Two round agreement protocol at the interface of analogue (dense) and digitial (discrete) world in time and value domain
- ♦ Communication: Same message order on independent channels--simultaneity is the challenge
- ♦ Processing: e.g.. Deterministic algorithms, No race condition on semaphores
- ♦ Timeouts: Reference to a global sparse timebase

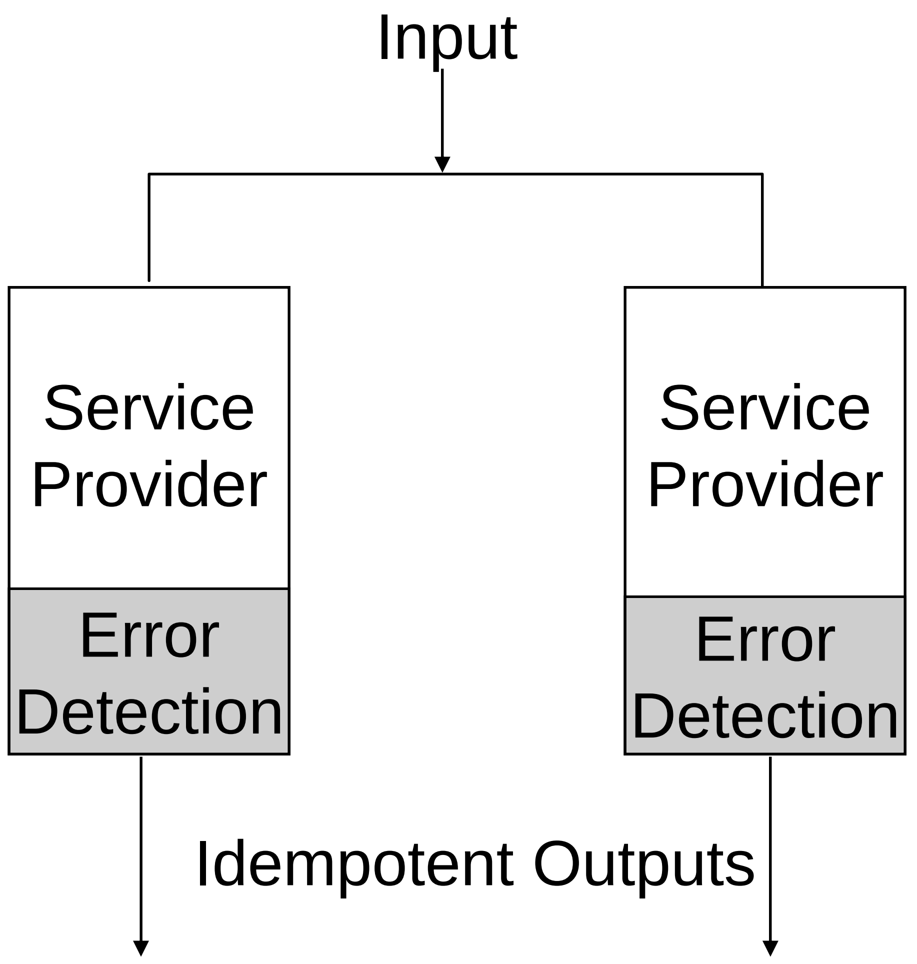
Non-Redundant Application



TMR Configuration

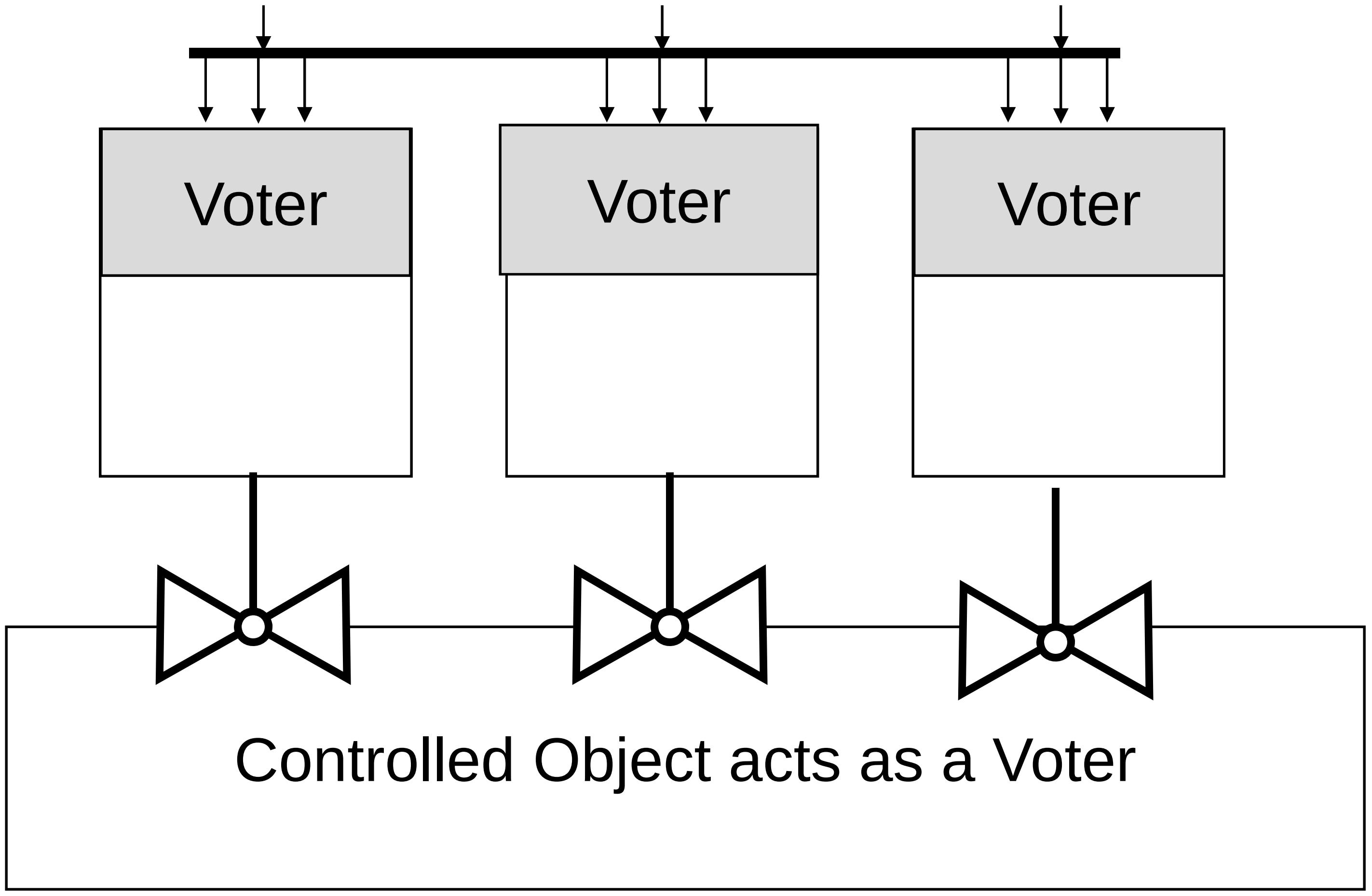


Active Redundancy: Self-checking Components

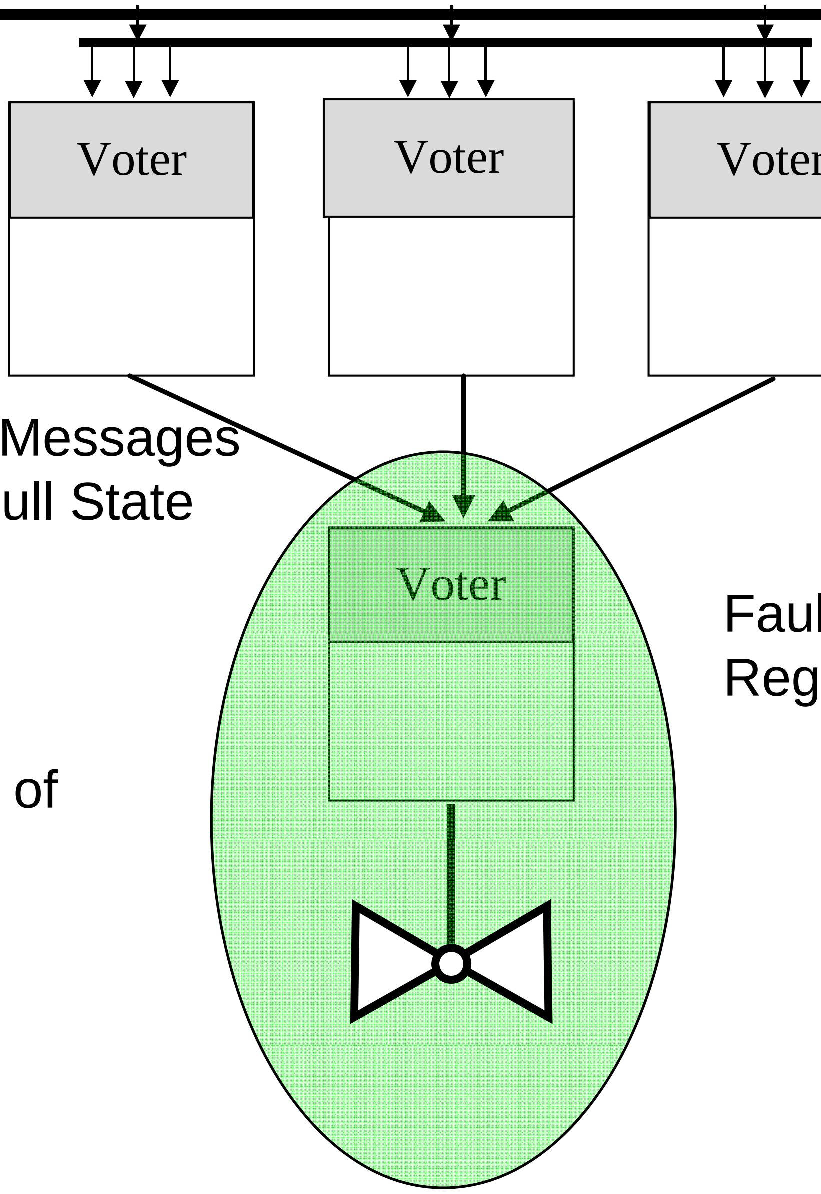


A FTU has to have k+1 selfchecking components to tolerate k fault

Active Redundancy: Fault Tolerant Actuators



Active Redundancy: Control of a Single Actuator

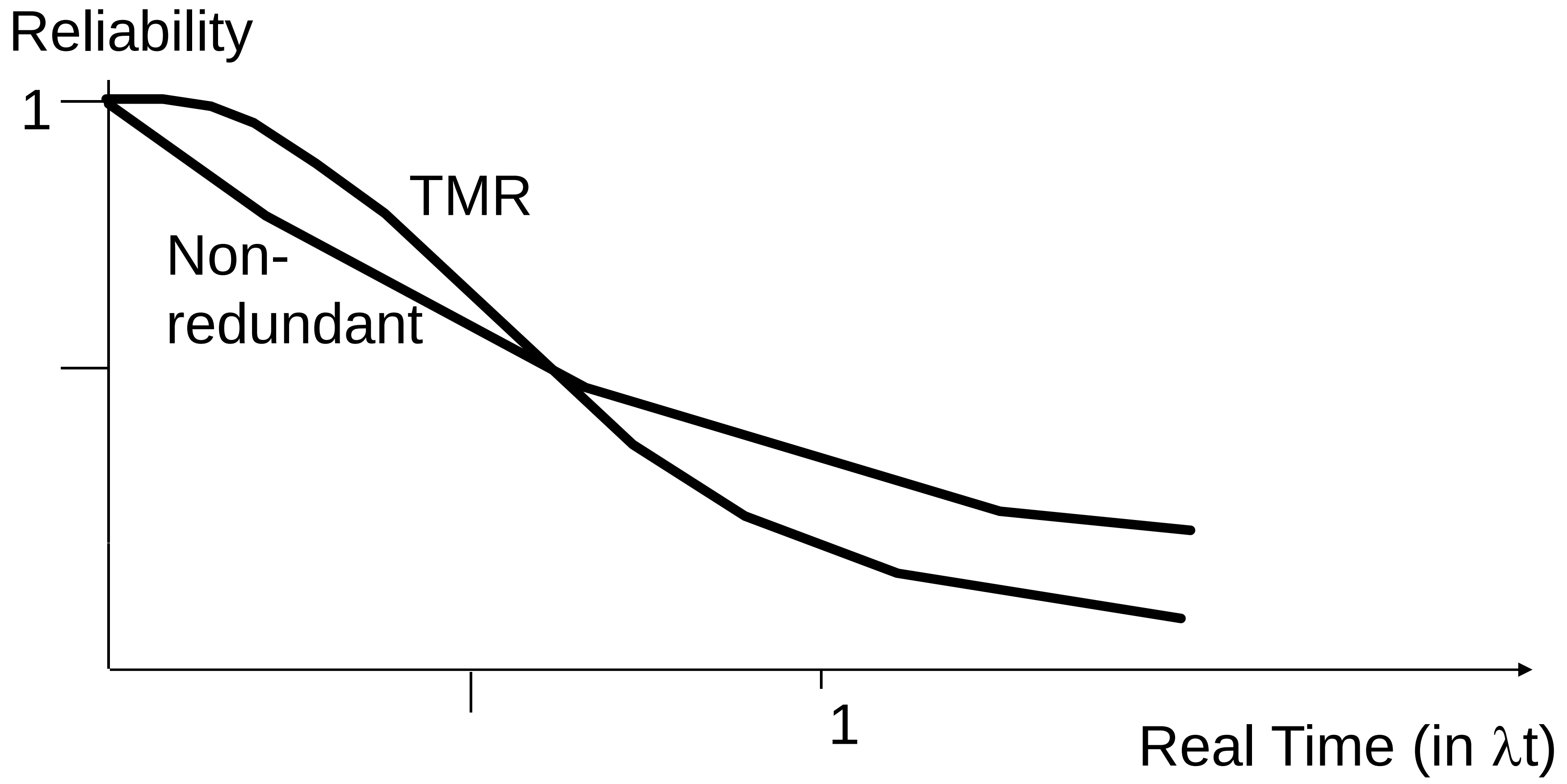


State Messages with Full State

Fault Containment Region without h-state

Masking of Transient Errors of the Computer System

Reliability of TMR Systems



On-line Maintainability increases the reliability of redundant systems by a factor of about MTTF/MTTR. Non repairable redundant systems are important if the mission time is short.

Complexity of Fault-Tolerance Mechanisms

Implementing Fault-Tolerance at the application level without proper architectural support

- ♦ results in complex application code that is prone to software errors
- ♦ this code is difficult to test, since the activation of the faults is non-trivial

In the end it can happen that the unreliability of the fault-tolerance mechanisms beats the purpose: the fault-tolerant system may be less reliable than the system without any special fault handling mechanisms!

Reintegration Point

Select the reintegration point

- at the end of a component cycle, when all atomic actions are complete
- when all channels are empty and the state is recorded either at the sender or at the receiver (ground state)
- immediately before reading fresh input data and after the output of the output data
- when most activities have been completed and the stack contains minimal information

In many automotive applications it is possible to find a reintegration point with an empty h-state!

Reintegration of Repaired Components in a FTU

- Perform a self-test of the component
- Load the static i-state (if not already in ROM)
- Listen to the traffic in the cluster and wait for clock synchronization messages to synchronize the local clock
- Wait for an g-state message from the partner component to reinitialize the h state of the reintegrating component
- Continue the normal processing function

Reintegration in a non-redundant system

In a non-redundant system, h-state cannot be recovered from a replicated partner.

Consistency between the state a system and of an actuator in the environment is established by a restart vector (e.g., in a traffic control system, force all traffic lights to red)

- ♦ Define different restart vectors for different modes
- ♦ Try to store the remaining h-state in some other component redundantly
- ♦ During restart, read the input values, determine the operating mode and then output the proper restart vector to the actuators

Design (Software) Faults

The application of fault-tolerance techniques to tolerate software faults by design diversity is still an open research area:

- ♦ If a disciplined software development process is followed most remaining failures are due to incorrect specification
- ♦ Even independent programming teams tend to make similar errors
- ♦ Replica Determinism can get lost if different algorithms are used

However, an independent check of safety assertions makes sense.

Fail-Safe Applications

In a fail-safe application an independent watchdog can monitor the operation of the computer.

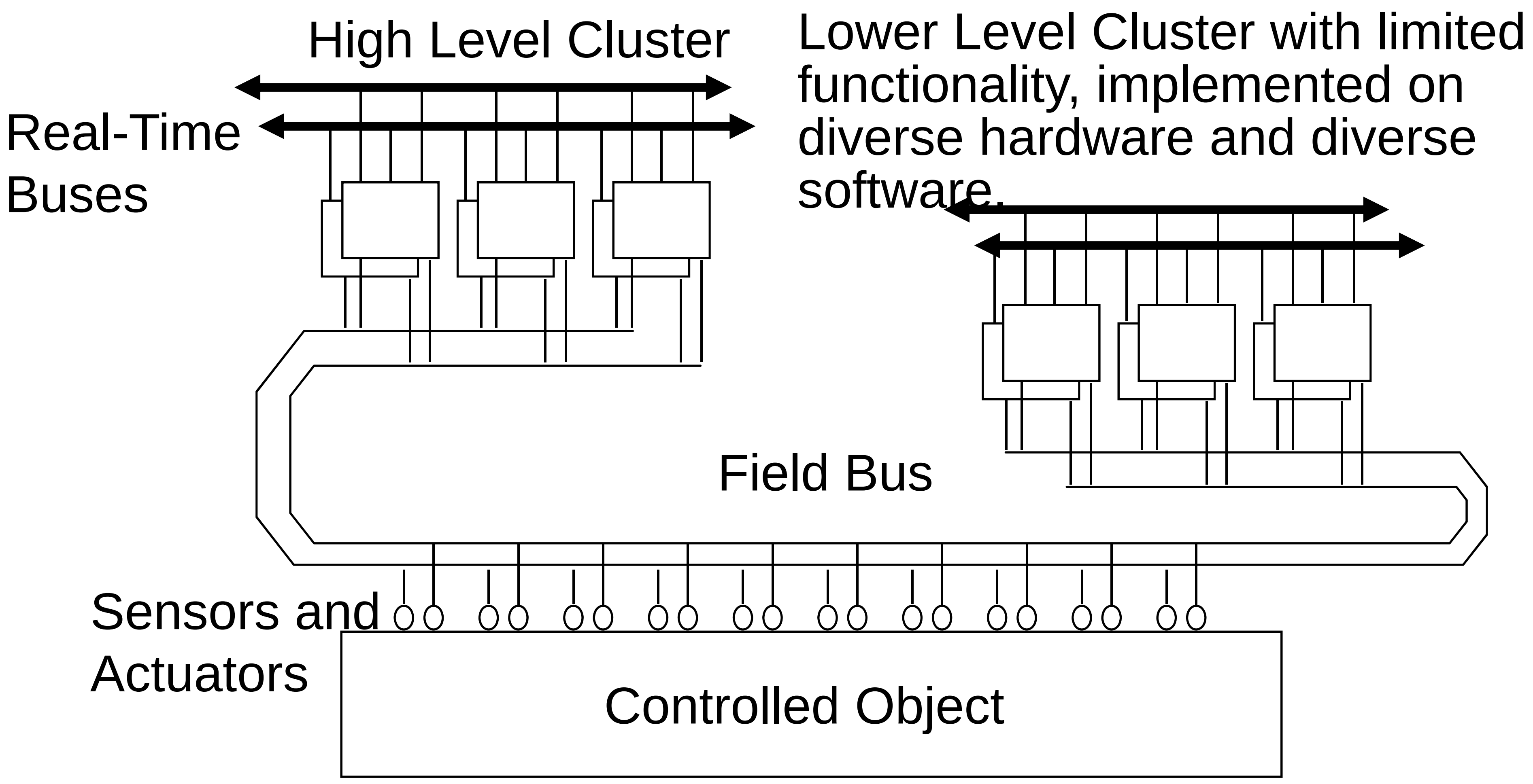
If the computer fails silently or is late, the watchdog can initiate a transition into the safe state (e.g., in a train control scenario: all signals to red, all trains stop).

The computer system must have a *high error detection coverage*.

For safety, timeliness is not the prime concern.

These techniques are not applicable *to fail-operational* systems.

Multilevel Architecture


High Level Cluster
Lower Level Cluster with limited functionality, implemented on diverse hardware and diverse software.
Real-Time Buses
Field Bus
Sensors and Actuators
Controlled Object