

# **The Fault Hypothesis for the Time-Triggered Architecture**

Hermann Kopetz

TU Wien

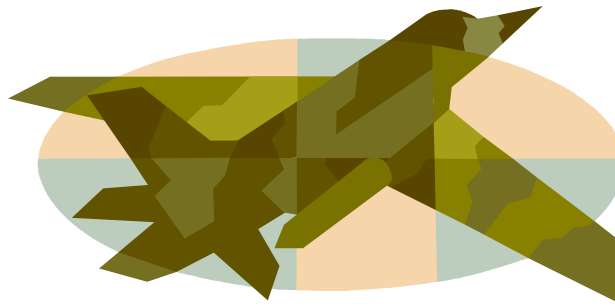
August 2004

- ◆ Introduction--the  $10^{-9}$  Challenge
- ◆ The Time-Triggered Architecture
- ◆ Contents of the Fault Hypothesis
- ◆ Fault Hypothesis of the TTA
- ◆ Transient Failures
- ◆ Design Faults
- ◆ Conclusion

# Examples of Safety Critical Systems--No Backup

---

**Fly-by-wire Airplane:** There is no mechanical or hydraulic connection between the pilot controls and the control surfaces.



**Drive-by-wire Car:** There is no mechanical or hydraulic connection between the steering wheel and the wheels.



# The $10^{-9}$ Challenge

---

- ◆ **Critical system services must be more reliable than any one of the components:** *e.g., System Dependability 1 FIT--Component dependability 1000 FIT (1 FIT: 1 failure in  $10^9$  hours)*
- ◆ **Architecture must be distributed and support fault-tolerance** to mask component failures.
- ◆ System as a whole is **not testable** to the required level of dependability.
- ◆ The safety argument is based on a **combination of experimental evidence** about the expected failure modes and failures rates of **fault-containment regions (FCR)** and a **formal dependability model** that depicts the system structure from the point of view of dependability.
- ◆ **Independence** of the FCRs is a critical issue.

# The Time-Triggered Architecture (TTA)

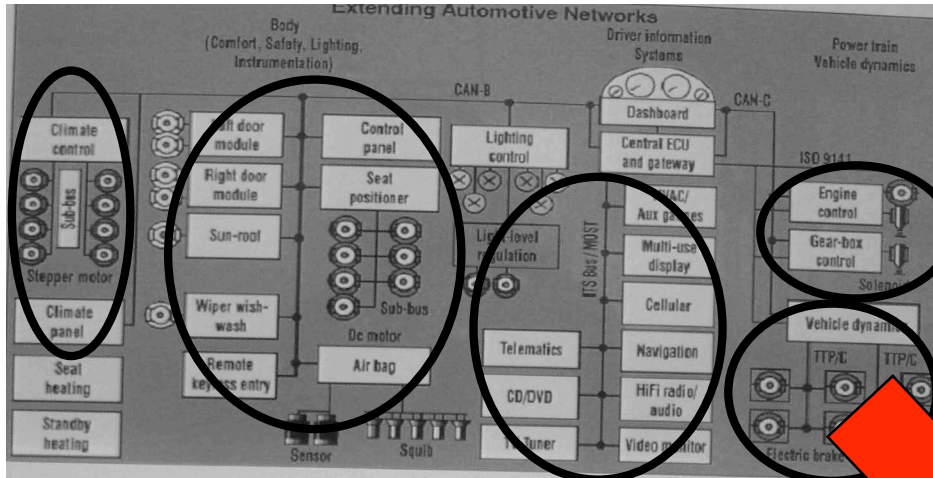
---

provides an execution environment for real-time applications. It is

- ◆ a *distributed architecture*, where a node can be a single chip computer (SoC).
- ◆ an *integrated architecture*, where different application subsystems (DAS) up to the *highest criticality class* can be integrated into a single framework.
- ◆ a *platform architecture* that provides technology invariant interfaces to the application software.
- ◆ a *generic architecture*, which can be deployed in different application domains (e.g., automotive, aerospace, train signaling, process control, multimedia) where real-time performance is an issue.
- ◆ It provides a *fault-tolerant global time-base* of high precision at every node.

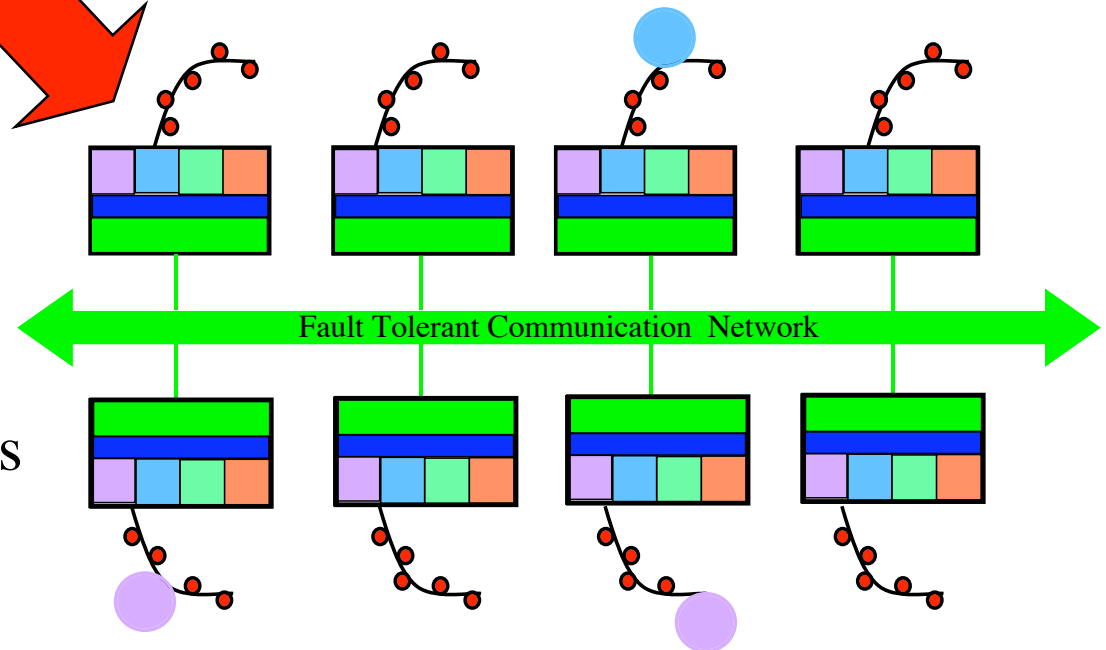
Kopetz, H, Bauer, G. , The Time-Triggered Architecture, Proc. of the IEEE, Jan 2003, Vol 91 p. 112-126

# From a Federated to an Integrated Architecture



**Federated Architecture:**  
*“Every functions has its own ECU”*

**Integrated Architecture:**  
Backbone Network with integrated fault-tolerance  
Intelligent Sensors and Actuators connected by field-buses



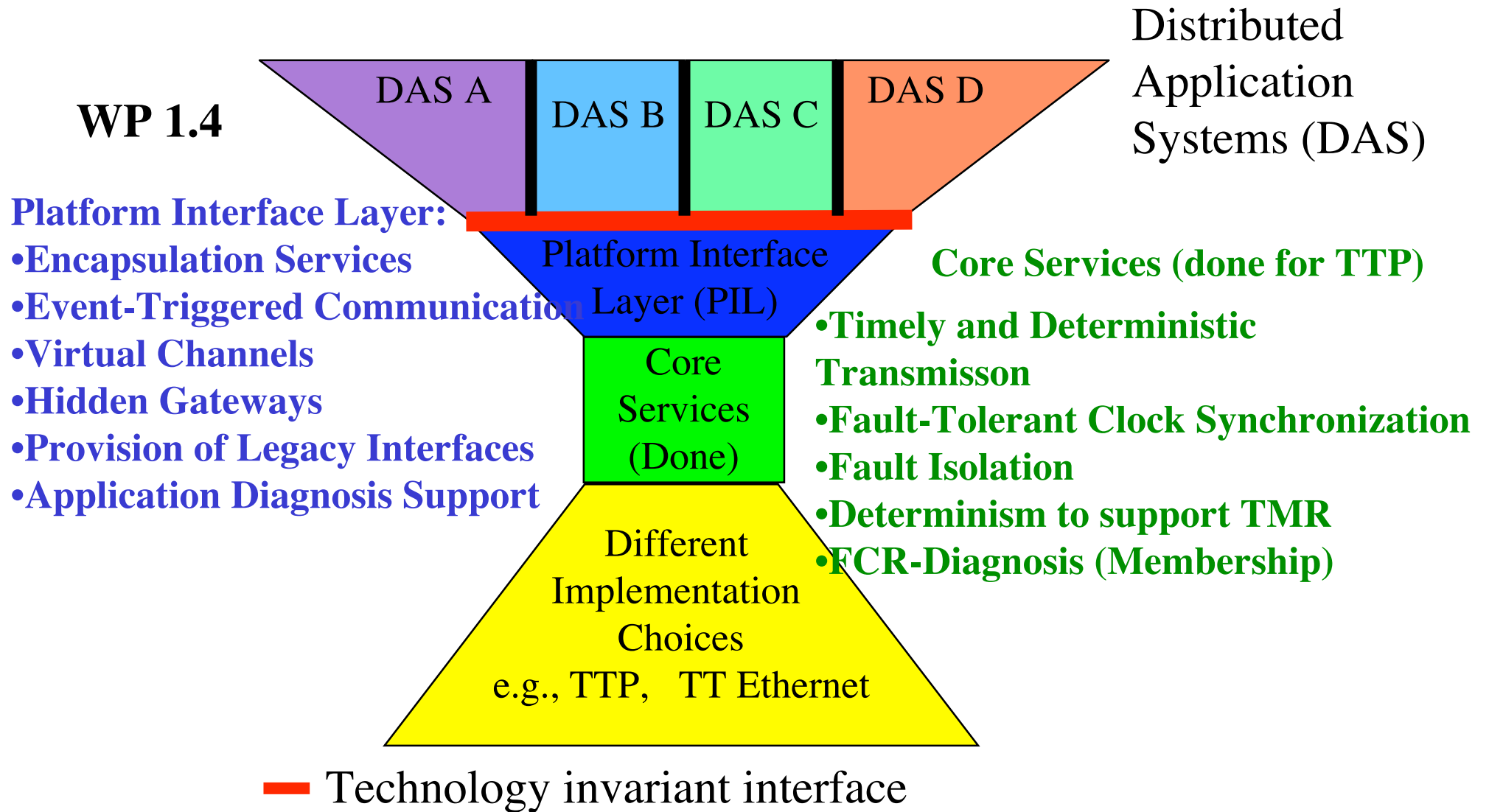
# Federated vs. Integrated System

---

*The ideal future avionics systems would combine the complexity management advantages of the federated approach, but would also realize the functional integration and hardware efficiency benefits of an integrated system.*

Hammett Robert. *Flight Critical Electronics System Design*, IEEE AESS Systems Magazine, June 2003, p.32

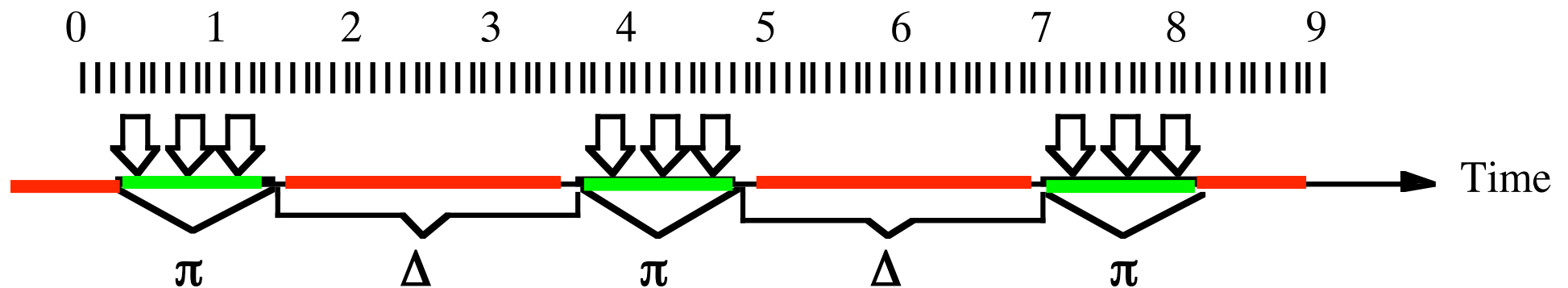
# The TTA is a *Platform Architecture*






# Fault Tolerant *Sparse* Time Base in the TTA

If the occurrence of events is restricted to some active intervals with duration  $\pi$  with an interval of silence of duration  $\Delta$  between any two active intervals, then we call the timebase  $\pi/\Delta$ -sparse, or sparse for short.



Events  are only allowed to occur at subintervals of the timeline

**In a sparse time base, instants can be represented by integers.**

# *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.
- ◆ The fault hypothesis partitions the fault space into *covered* faults and *uncovered* faults.
- ◆ ***The fault hypothesis is the most important document*** in the design of a fault-tolerant system.

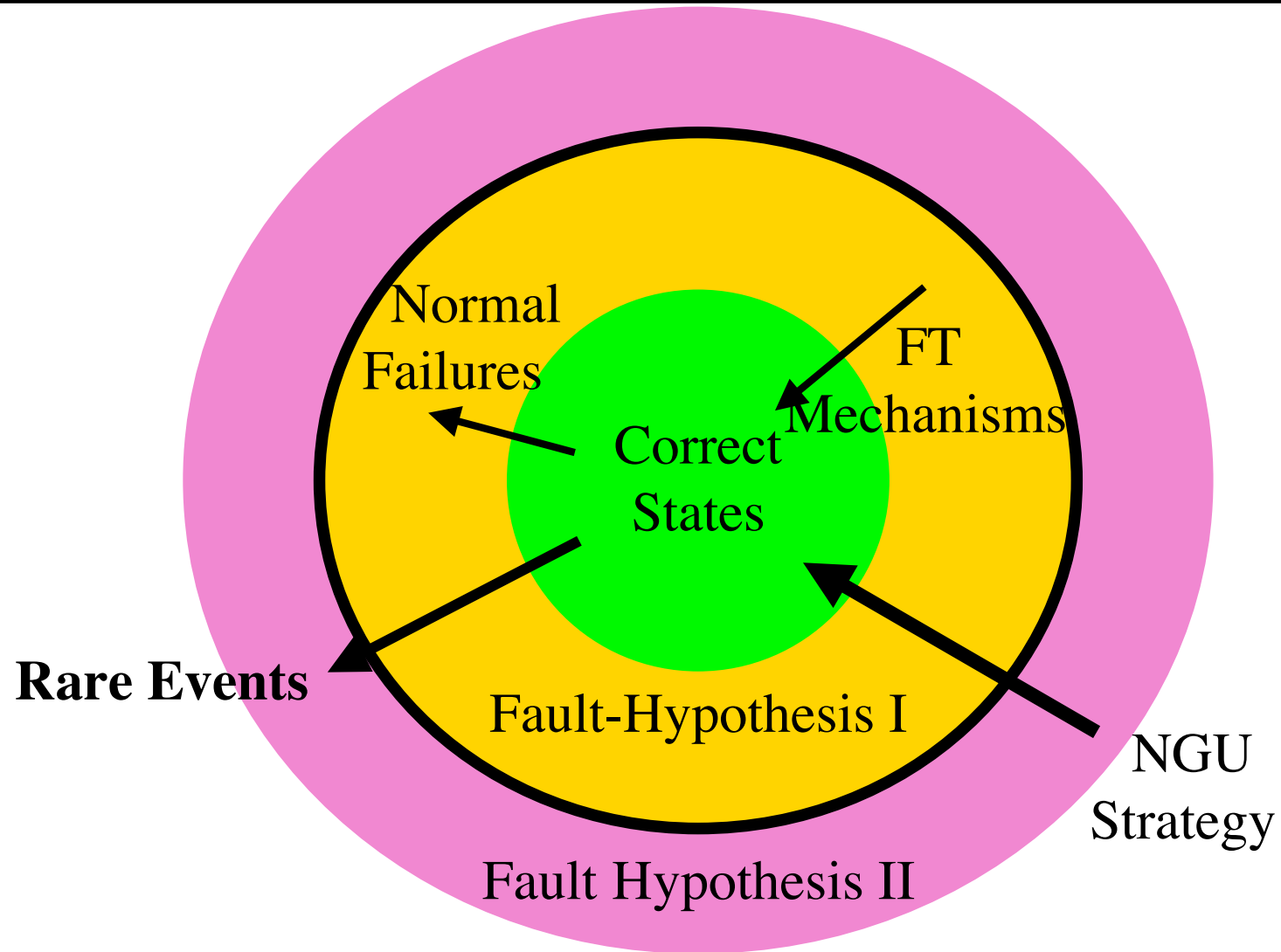
# Why is the Fault Hypothesis Needed?

---

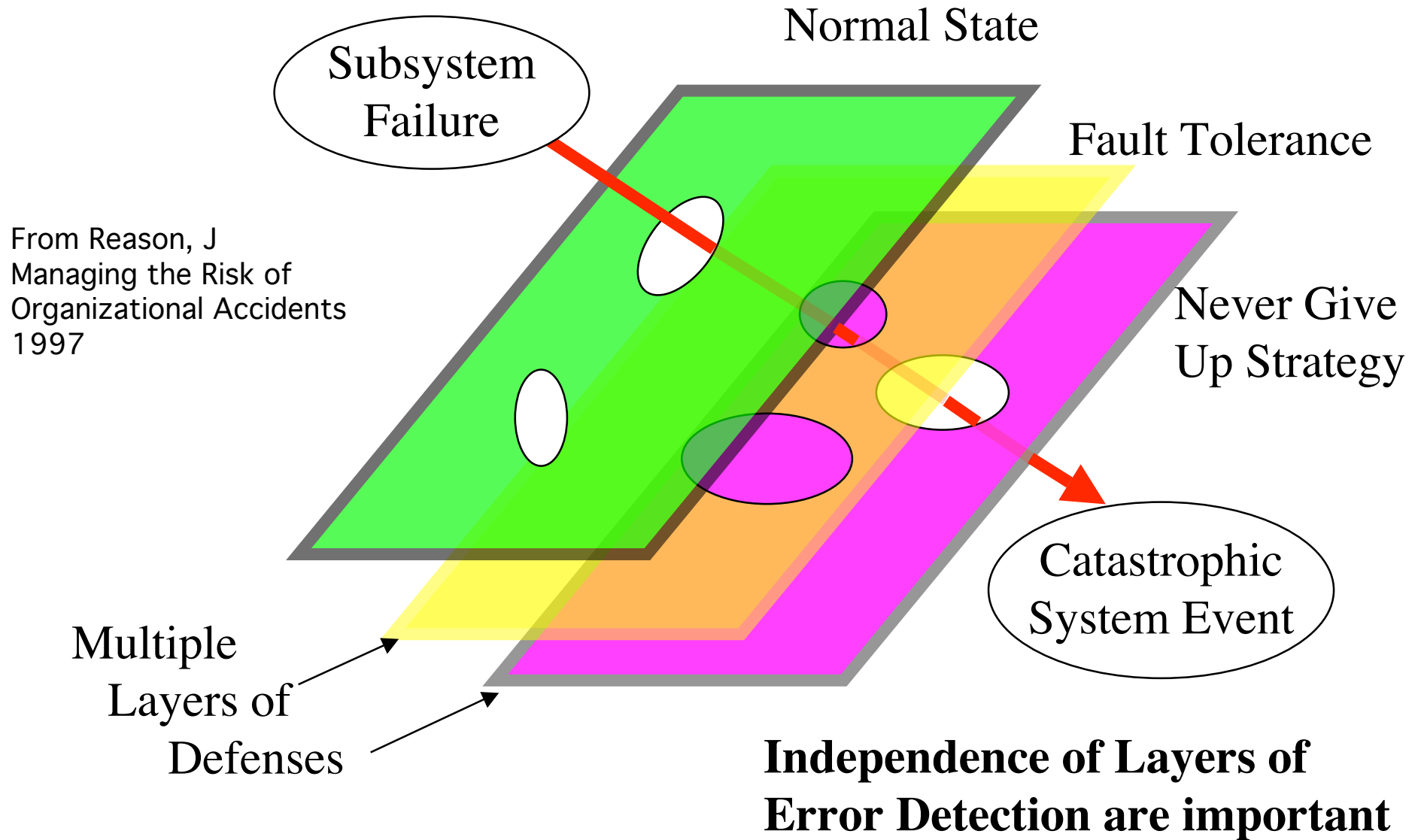
- i. **Design of the Fault-Tolerance Algorithms:** Without a precise fault-hypothesis it is not known which fault-classes must be addressed during the system design.
- ii. **Estimation of the Assumption Coverage:** Probability that the assumptions that are contained in the fault hypothesis are not met by reality.
- iii. **Validation and Certification:** For the validation it must be known which faults are supposed to be tolerated by the given system.
- iv. **Design of the Never-Give-Up (NGU) Strategy:** In case the fault hypothesis is violated the NGU process must be started.

# System States of a FT System

---



# Approach to Safety: The *Swiss-Cheese* Model



# Fault Hypothesis I vs. Fault Hypothesis II

---

## **Fault Hypothesis I:**

Specification of the faults that must be tolerated without any impact on essential system services.

*Example:* Arbitrary failure of an SoC

## **Fault Hypothesis II:**

Specification of faults that can be handled in the rare-event scenario, e.g., for the never-give-up (NGU) strategy

*Example:* massive transients that causes the failure of all communication and more than one node during a limited interval of time

# Contents of the Fault Hypothesis

---

- i. **Unit of Failure:** What is the Fault-Containment Region (FCR)?
- ii. **Failure Modes:** What are the failure modes of the FCR?
- iii. **Frequency of Failures:** What is the assumed MTTF between failures for the different failure modes eg. transient failures vs permanent failures?
- iv. **Detection:** How are failures detected? How long is the detection latency?
- v. **State Recovery:** How long does it take to repair corrupted state (in case of a transient fault)?

# Unit of Failure: Fault Containment Region (FCR)

---

16

A *fault-containment region (FCR)* is a set of subsystems that shares one or more common resources that can be affected by a single fault and is assumed to fail **independently** from other FCRs.

- ◆ Tolerance w.r.t. *spatial proximity faults* requires spatial separation of FCRs: **distributed architectures** required.
- ◆ The fault hypothesis must specify the failure modes of the FCRs and their associated frequencies.
- ◆ Beware of shared resources that compromise the **independence assumption**: common hardware, power supply, oscillator, earthing, single timing source.



# Failure Modes of an FCR--Are there Restrictions? 17

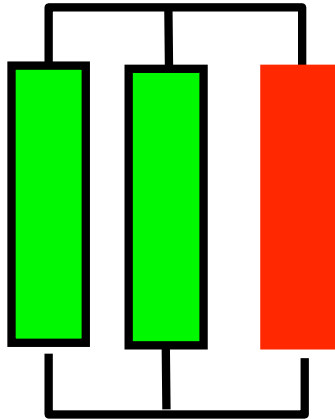
---

A



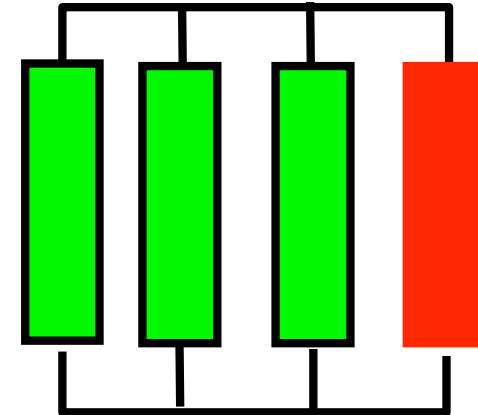
assumption  
*fail-silent*  
 $k+1$

B



assumption  
*synchronized*  
 $2k + 1$

C

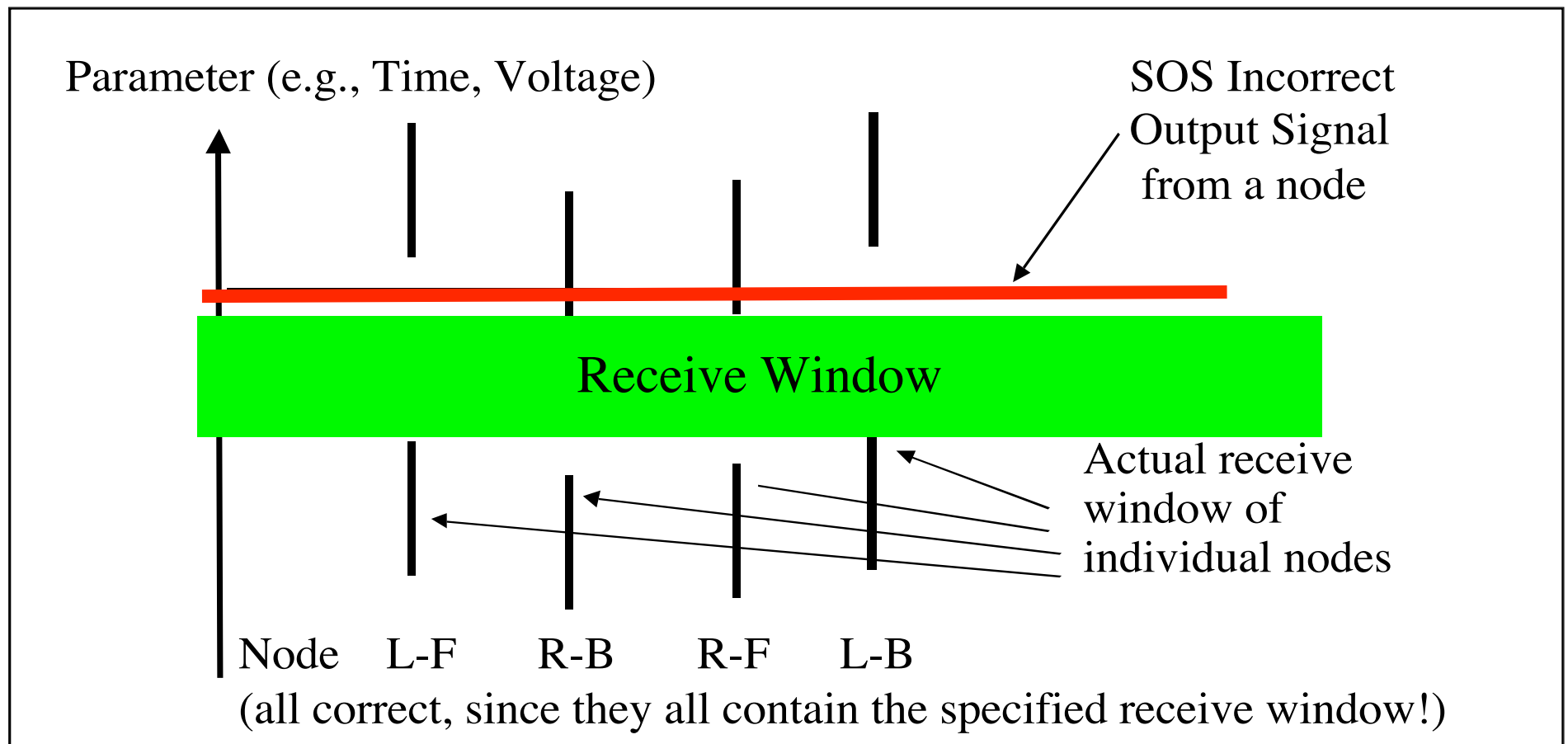


no assumption  
(arbitrary)  
 $3k + 1$

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

# Example: Slightly-out-of-Specification (SOS) Failure<sup>18</sup>

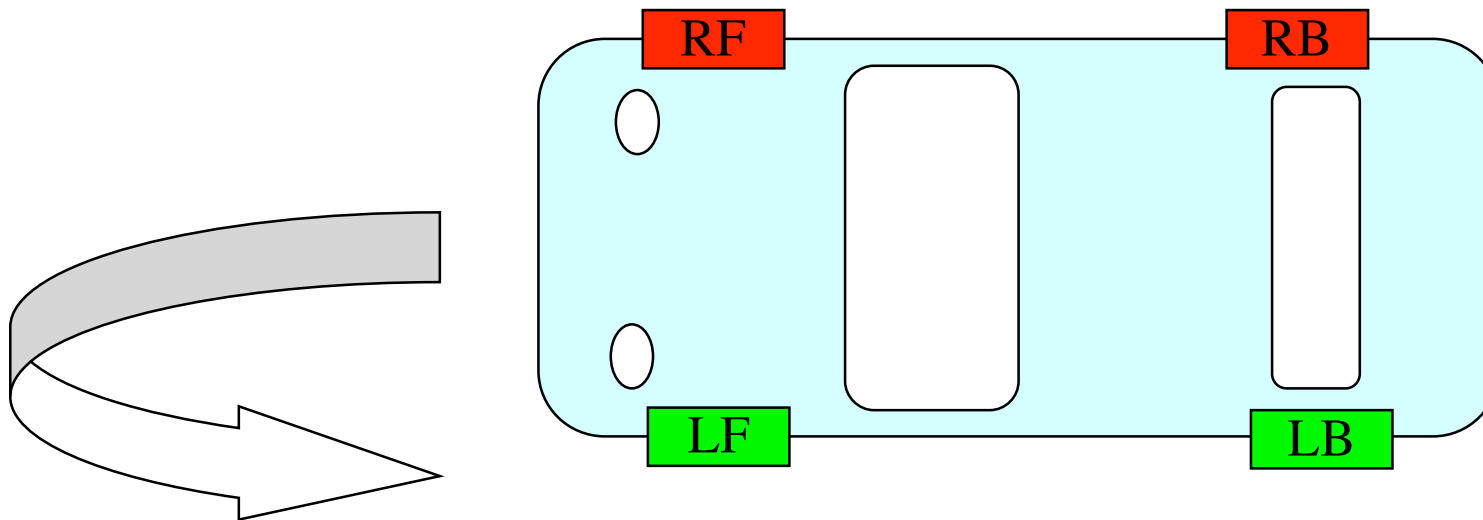
The following is an example for the type of asymmetric non-fail-silent failures that have been observed during experiments:



# Example Brake by Wire Application

---

Consider the scenario where the right two brakes *do not* accept an SOS-faulty brake-command message, while the left two brakes *do* accept this message and brake.



If the two left wheels brake, while the two right wheels do not brake, the car will turn.

# Independence of FCRs

---

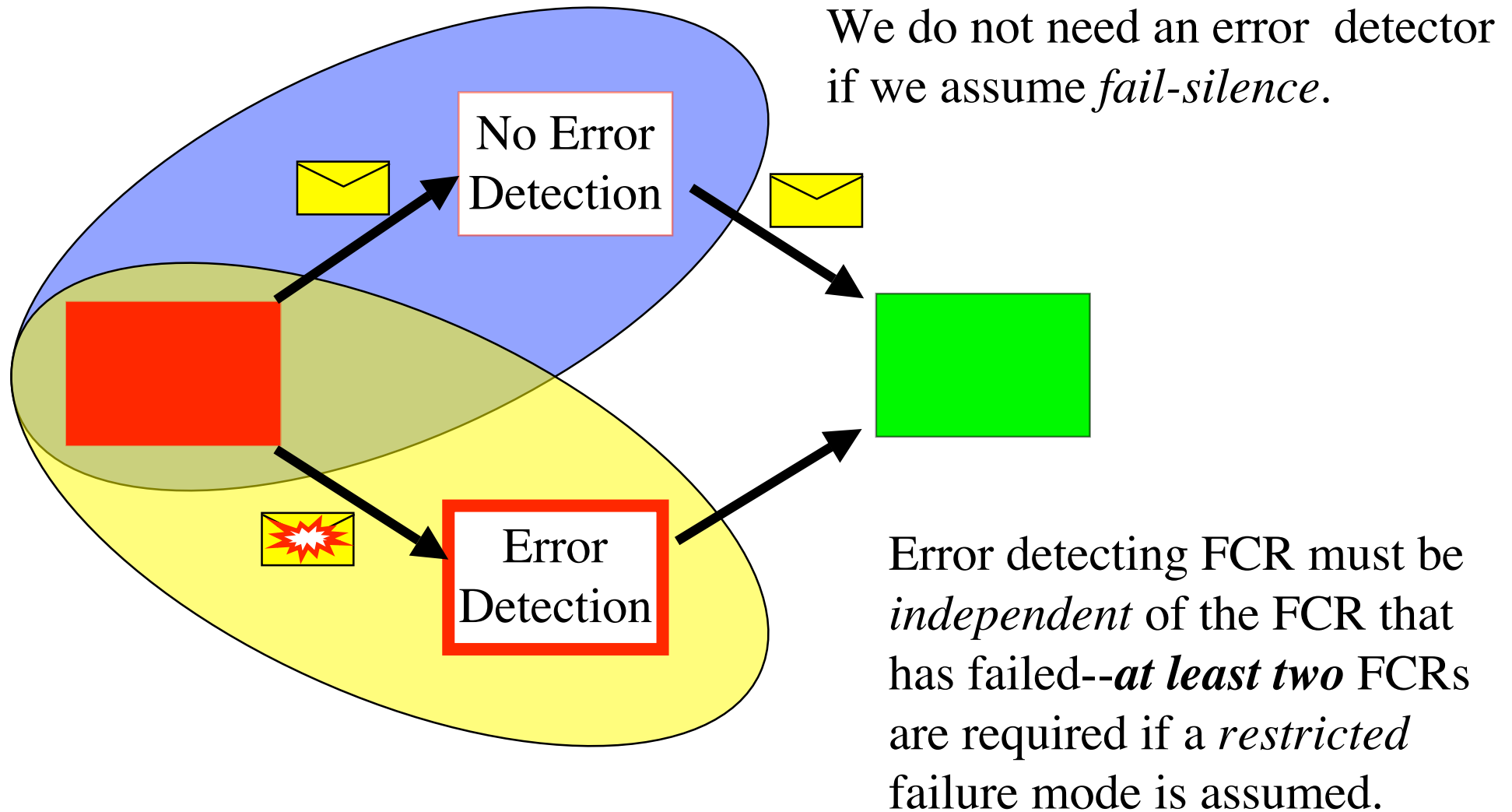
There are two basic mechanisms that compromise the independence of FCRs

- ◆ Missing fault isolation among the FCRs
- ◆ Error propagation--the consequences of a fault, the *ensuing error*, propagates to a healthy FCR by an erroneous message.

**The independence of failures of different FCRs is the most critical issue in the design of an ultra-dependable system.**

# Fault Containment vs. Error Containment

---



# 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**.

# Consequences for an Architecture

---

In a safety-critical application an SoC (System on Chip) must be considered to form a *a single FCR (ie. a single unit of failure)* 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.**

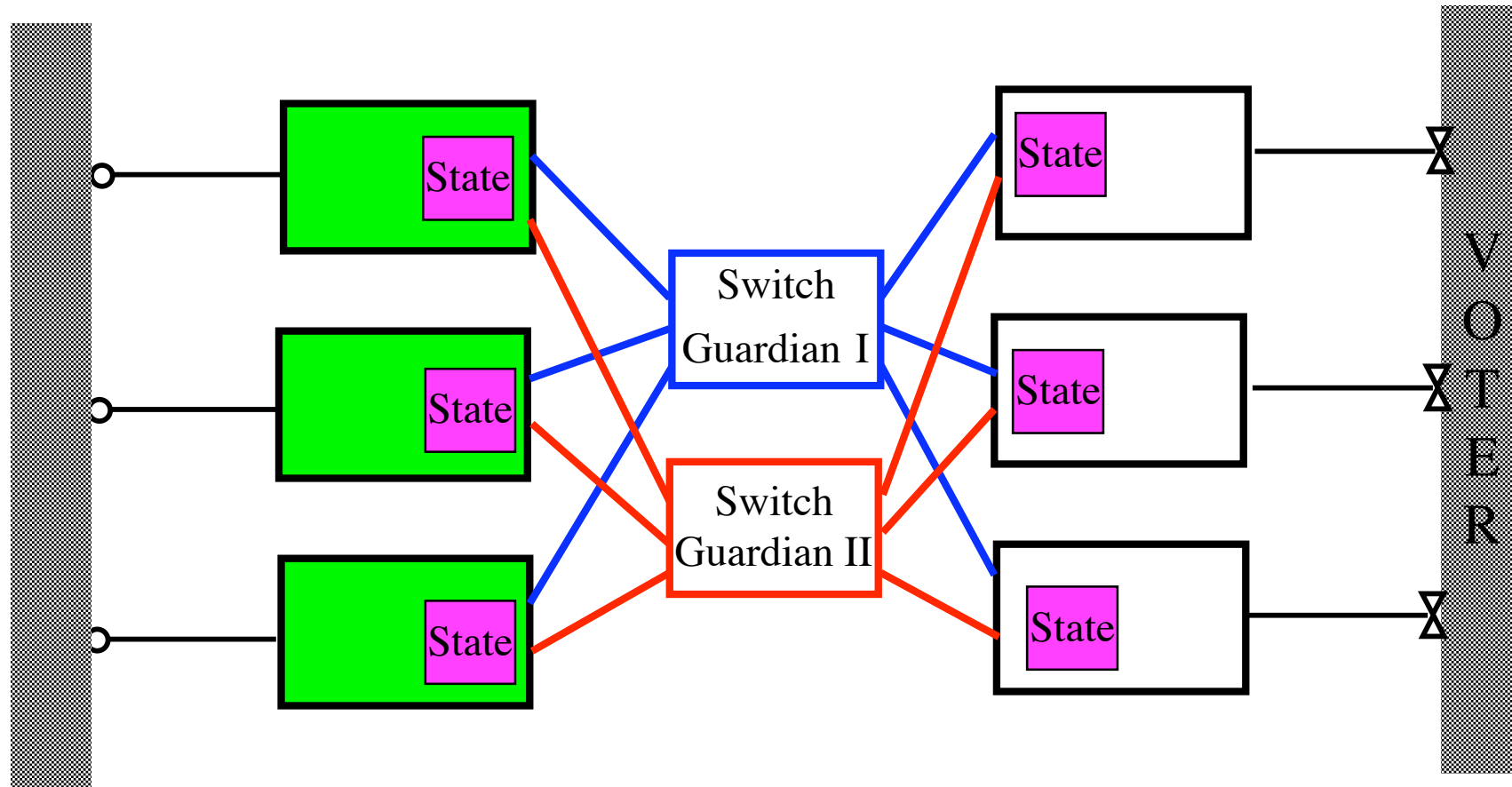
# Fault Hypothesis in the TTA w.r.t. Physical Faults

---

- i. **A Node Computer** forms a single FCR that can fail in an arbitrary failure mode.
- ii. **A communication channel** including the central guardian forms a single FCR that can fail to distribute messages but cannot generate messages on its own.
- iii. **A central guardian in the communication system** transforms (SOS) failures to fail-silent failures in the temporal domain.
- iv. **Error detection** is performed by a membership and clique avoidance algorithms.
- v. **The system can recover** from a single failure within two TDMA rounds.



# TMR Structure for Safety-Critical Tasks



State

In order to flush out *quasi-permanent state errors* caused by a transient fault, the state must be periodically subject to voting.

# Assumption about the Frequency of Faults of SoCs:<sup>26</sup>

---

Assumed Behavioral Hardware Failure Rates (Orders of Magnitude):

Type of Failure	Failure Rate in Fit	Source
Transient Node Failures (fail silent)	<1 000 000 Fit (MTTF > 1000 hours)	Neutron bombardment Aerospace
Transient Node Failure (non-fail silent)	<10 000 Fit (MTTF > 100 000)	Fault Injection Experiments
Permanent Hardware Failures	<100 Fit (MTTF > 10 000 000)	Automotive Field Data

Tendency: Increase of Transient Failures

# 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 to find out which are realistic assumptions:

- ◆ Software based (TU Vienna, Austria)
- ◆ 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?

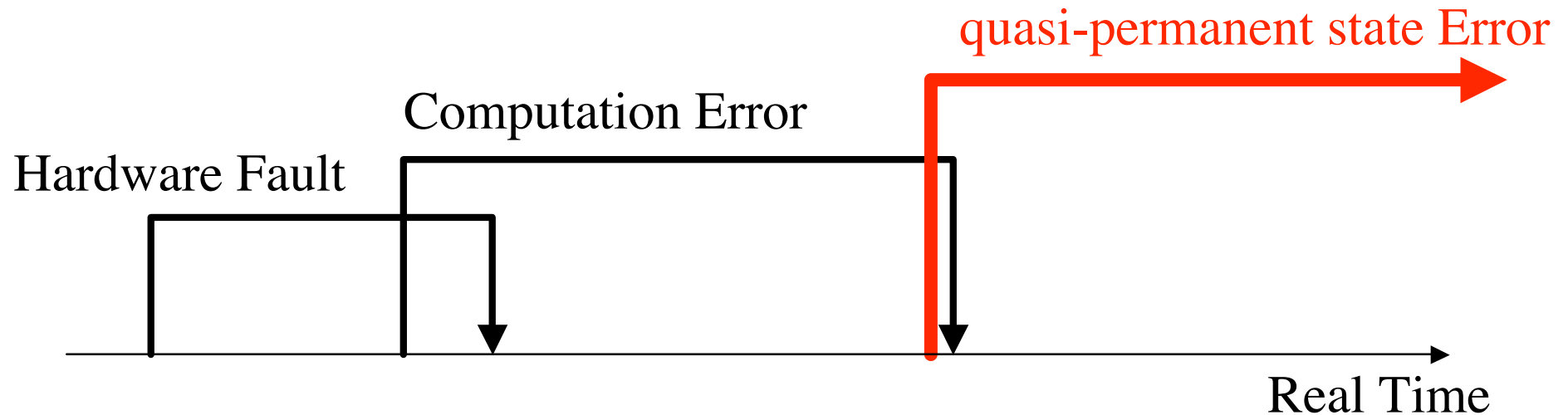
---

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

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

# Transient Faults may cause Permanent State Errors <sup>29</sup>

---



The interaction of a transient hardware fault with the state can cause a quasi-permanent state error: state erosion

Transient failures MTTF: 1000 hours

Permanent failures MTTF: > 1 000 000 hours

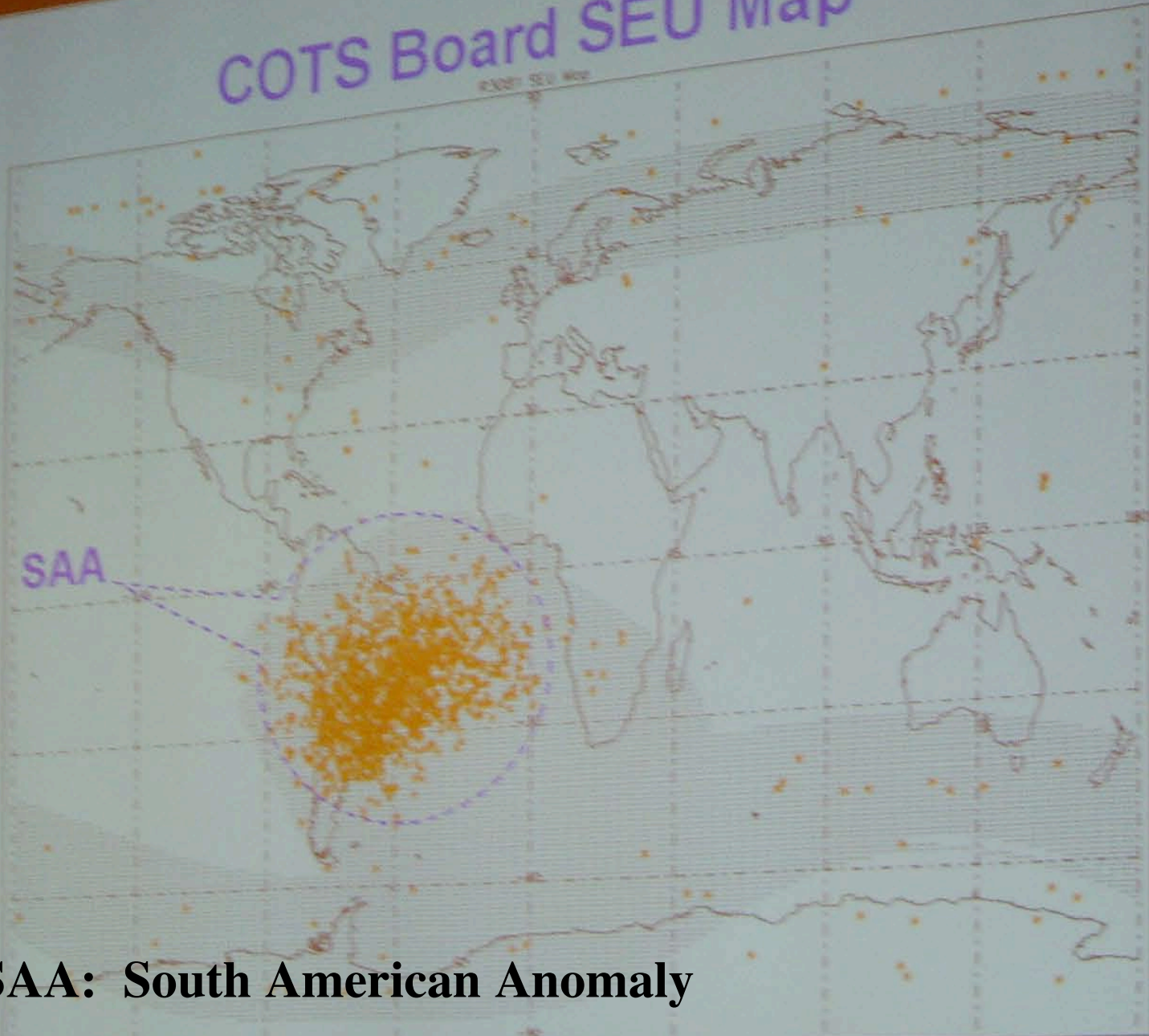
# The Cause of a Transient Fault

---

We have identified the following possible causes of a transient fault

- ◆ **External Disturbances**, e.g., high energy radiation (hardware)
- ◆ **Internal Degradation of the chip hardware**: e.g., corrosion of a PN junction (hardware)
- ◆ **Heisenbugs**, e.g., design error in the synchronization of processes (software)

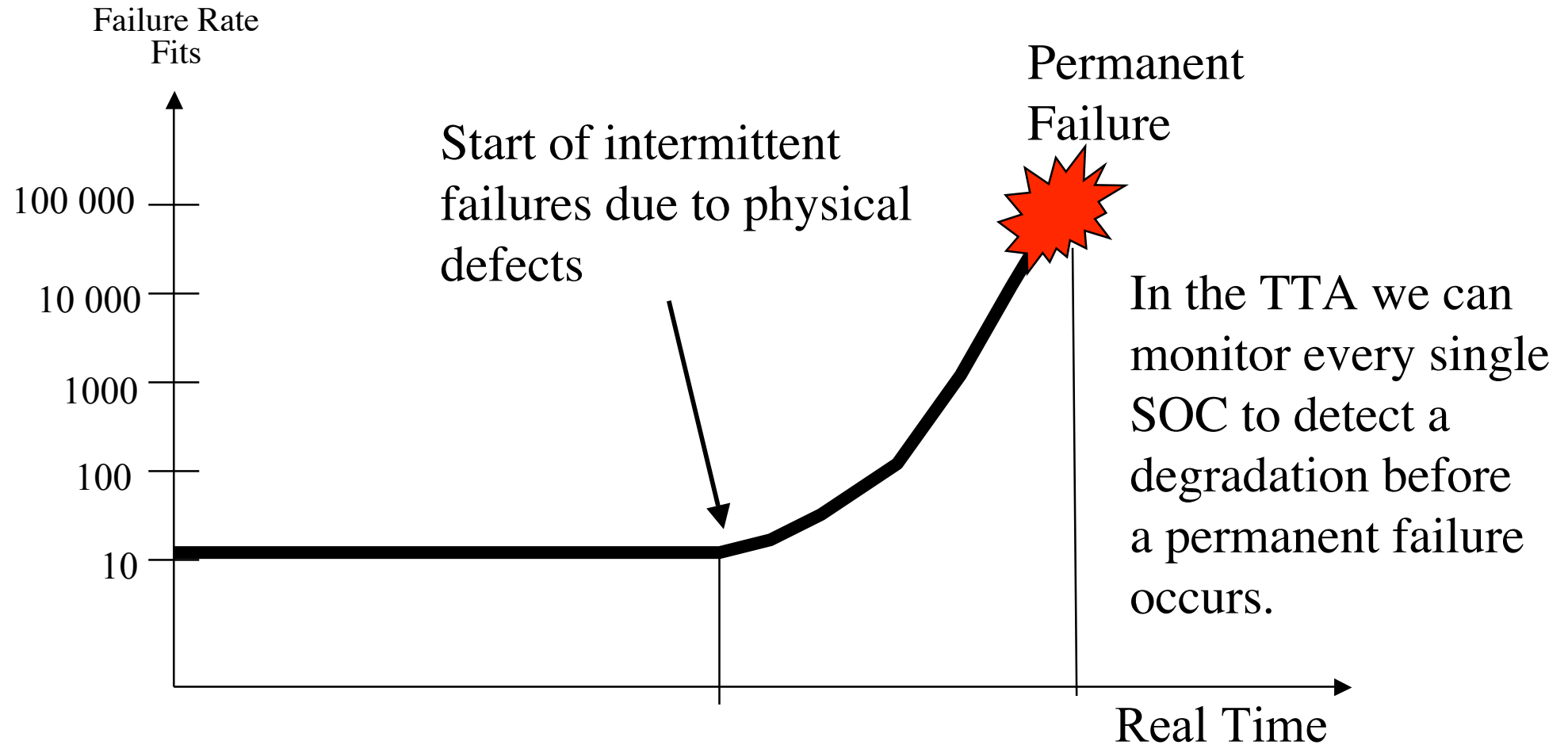
# COTS Board SEU Map



**SAA: South American Anomaly**

# Intermittent Failures of a Chip causes Transients

---



More than half of the transients may be caused by intermittents.



# The Distinction between *Bohrbugs* and *Heisenbugs*\*

33

- 
- ◆ *Bohrbugs* are design errors in the software that cause reproducible failures. E.g., a logic error in a program.
  - ◆ *Heisenbugs* are design errors in the software that seem to generate quasi-random failures. E.g., a synchronization error that will cause the occasional violation of an integrity condition.
  - ◆ From a phenomenological point of view, a failure that is caused by a *Heisenbug* cannot be distinguished from a failure caused by transient hardware malfunction.
  - ◆ Experience shows that it is much more difficult to find and eliminate the *Heisenbugs* than it is to eliminate the *Bohrbugs* from a large software system.

\*J. Gray, "Why do Computers Stop and What can be done about it?," Proc. 5th Symp. on Reliability in Distributed Software and Database Systems, Los Angeles, CA, USA, 1986

# The Replacement Strategy

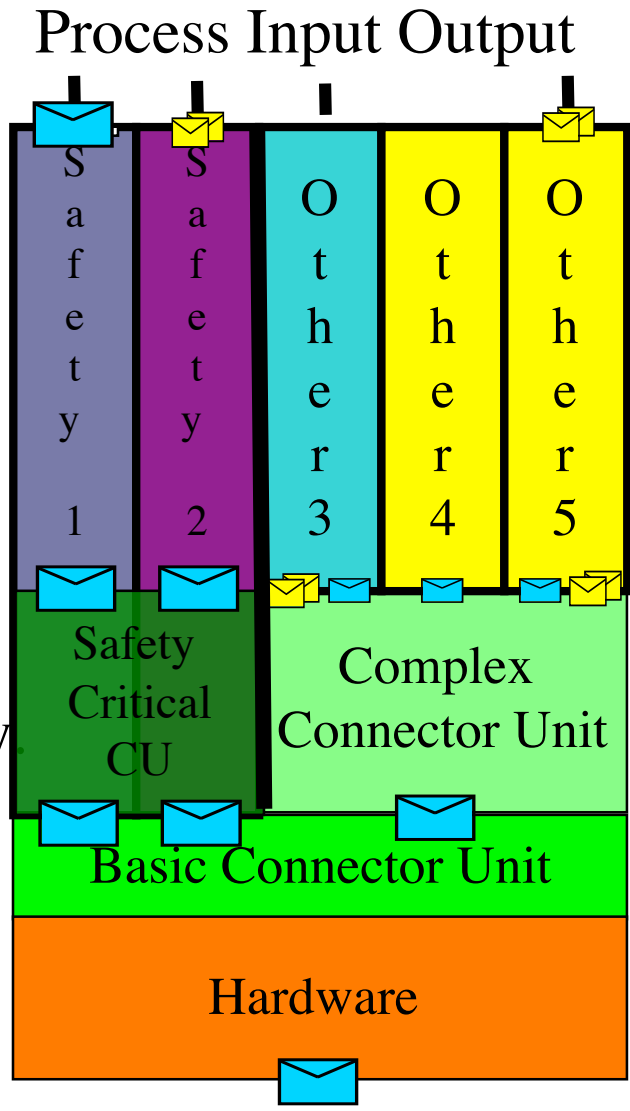
---

- ◆ From the observation of a transient failure of a node, it *impossible* to identify in a *single function node* the cause of the transient.
- ◆ It is possible to reason about the cause of the transient if a *population of nodes* is observed over time.
- ◆ It is also possible to reason about the cause of the transient if the malfunctions of a *single multifunction node* are observed over time.

# Mixed-Criticality TTA Node

Mixed-Criticality Node with 6 Partitions, controlled by connector units.

The two safety-critical partitions depend on the correctness of the Basic Connector Unit only



malign failures

DAS 1 Safety Critical

DAS 2 Safety Critical

benign failures

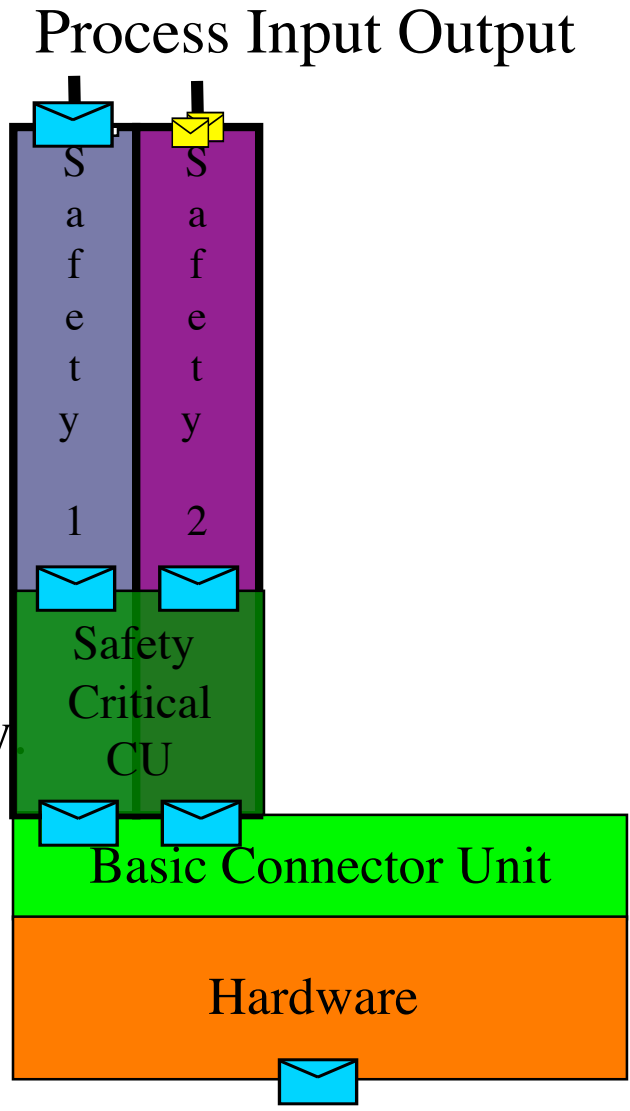
DAS 3

DAS 4

# Critical Parts of a Mixed-Criticality TTA Node

Mixed-Criticality Node with 6 Partitions, controlled by connector units.

The two safety-critical partitions depend on the correctness of the Basic Connector Unit only



malign failures

DAS 1 Safety Critical

DAS 2 Safety Critical

benign failures

DAS 3

DAS 4

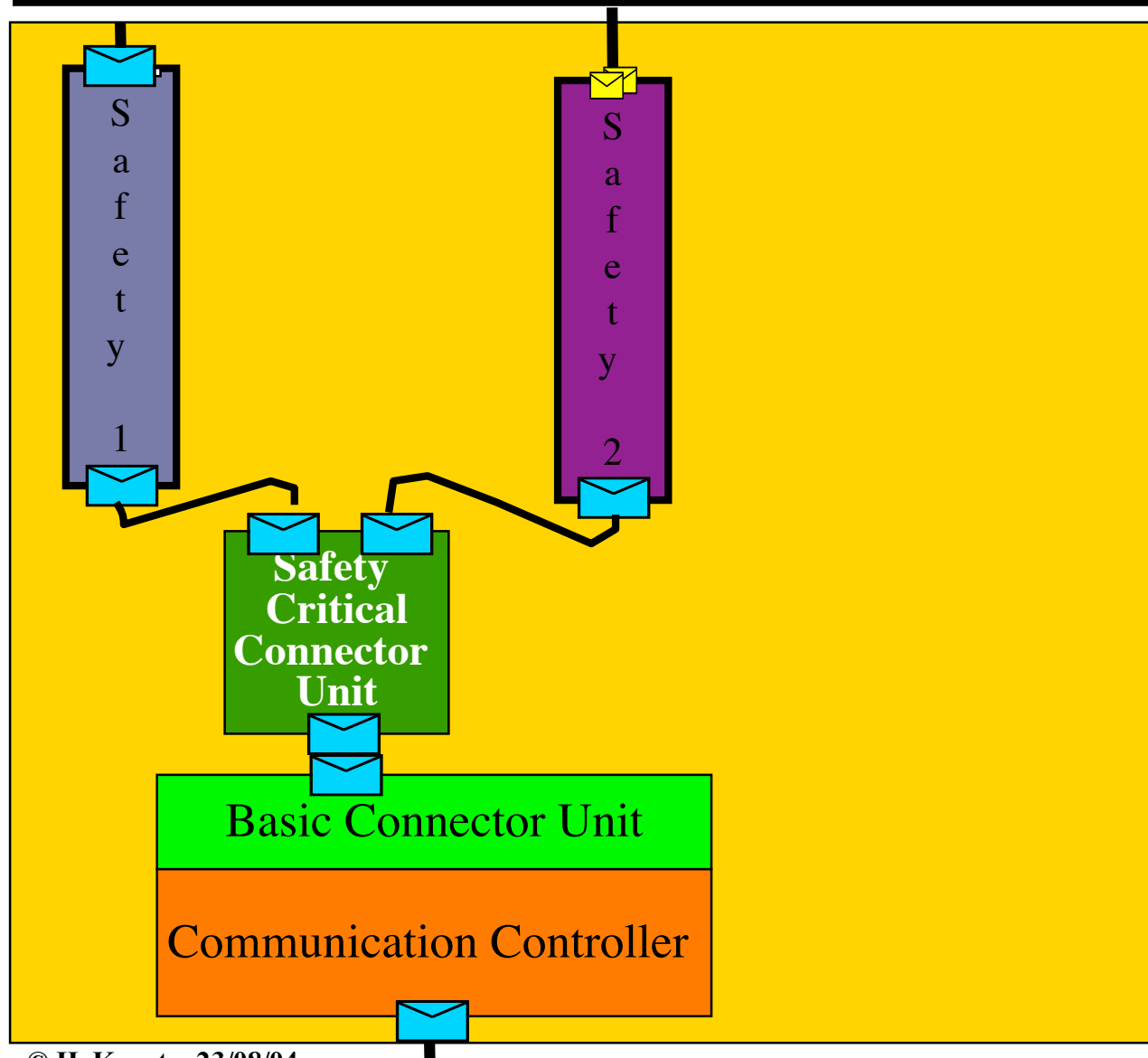
# Formal Analysis of the Core Services

---

Minimize the complexity in the hardware and the software of the core services of the TTA in order that they can be formally analyzed

- ◆ Partition the system such that modular certification is supported.
- ◆ Time-triggered (TT) communication with elementary interfaces--no backpressure
- ◆ Basic Connector Unit supports static mechanisms only
- ◆ Temporal encapsulation by design (hardware)
- ◆ Avoid algorithms which are not amenable to formal certification (e.g., feedback in the clock-synchronization)

# Modular Certification of a TTA Node



Each unit  
can be certified  
in isolation  
from each other  
unit.

Unintended  
interactions are  
avoided by  
design.

# Conclusion

---

- ◆ The Design of Safety-Critical Computing Systems requires a fault-tolerant architecture and a rigorous design methodology
- ◆ The precise specification of the fault hypothesis is the key document in the design of a fault-tolerant systems.
- ◆ The architecture of for a safety critical application must tolerate the arbitrary failure of any single VLSI Chip since we cannot assume that a chip contains two independent fault containment regions.