

**TU Wien**

---

**System-level Error-Handling Mechanisms  
in the Time-Triggered Architecture**

H.Kopetz  
July 2011

# Outline

---

- Introduction
- Cyber-Physical Systems (CPS)
- TTA Overview
- Error Handling Mechanisms
- Conclusion

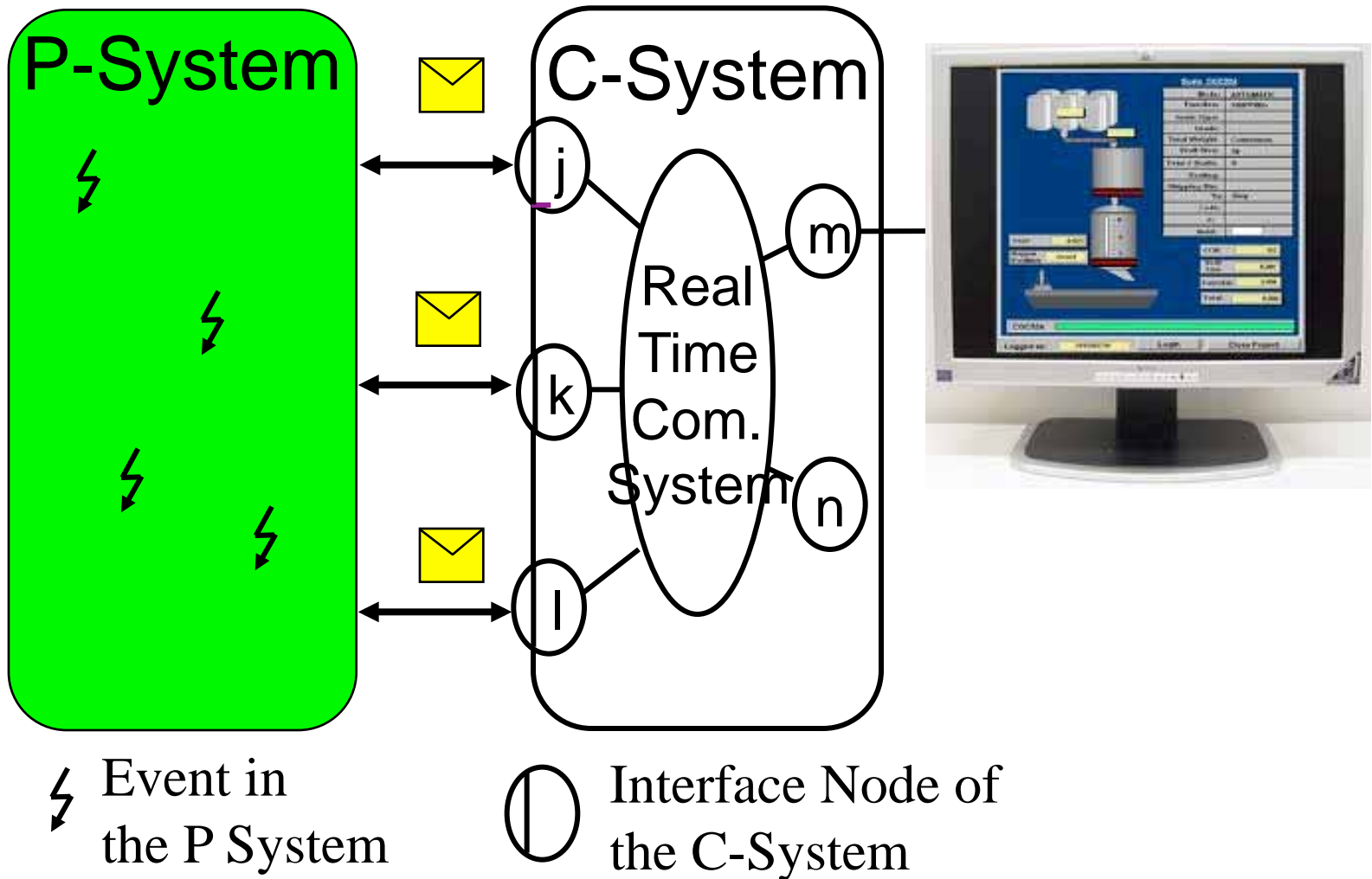
# Component-Based Systems

---

- The TTA establishes a framework for the design of *dependable component-based cyber-physical systems (CPS)*.
- In order to understand the design decisions taken in and the dependability mechanisms provided by the TTA we have to understand the characteristics of CPS.
- The TTA is based on more than twenty years of research on the topic of dependable real-time systems.

# CPS: Physical World meets Cyber World

---



# ***Physical (P) System versus Cyber (C) System***

---

## **P-System**

Controlled by the  
laws of physics

Physical time

Time base dense

## **C-System**

Controlled by  
program execution

Execution time

Time-base sparse

**The model of the P-system that is used by the C-system must be aware of the progression of Real time.**

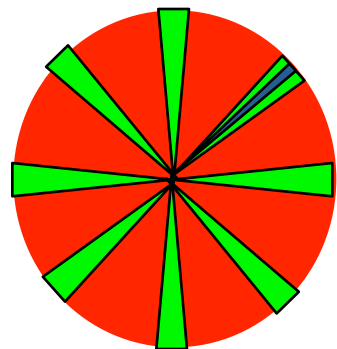
# Models of Time in a CPS



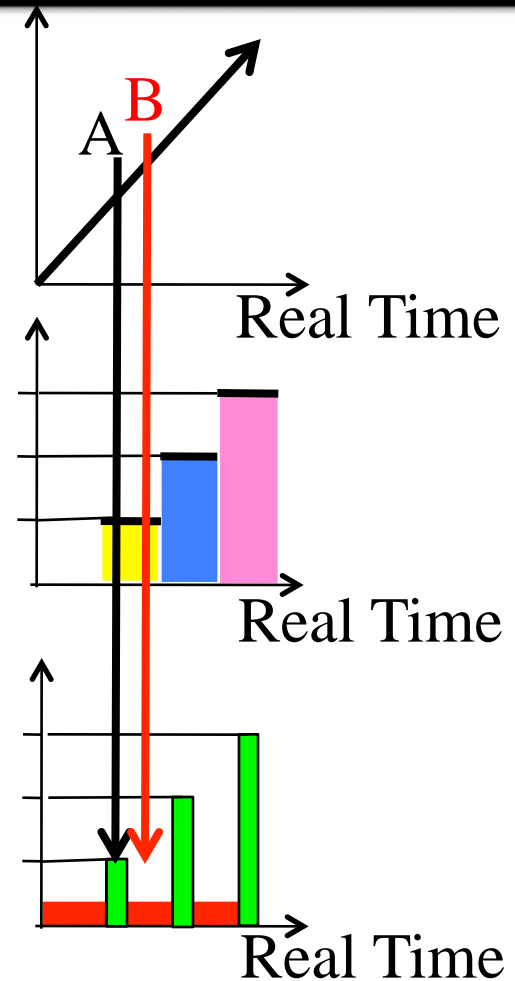
**Dense**  
Physics



**Discrete**  
Central Computer



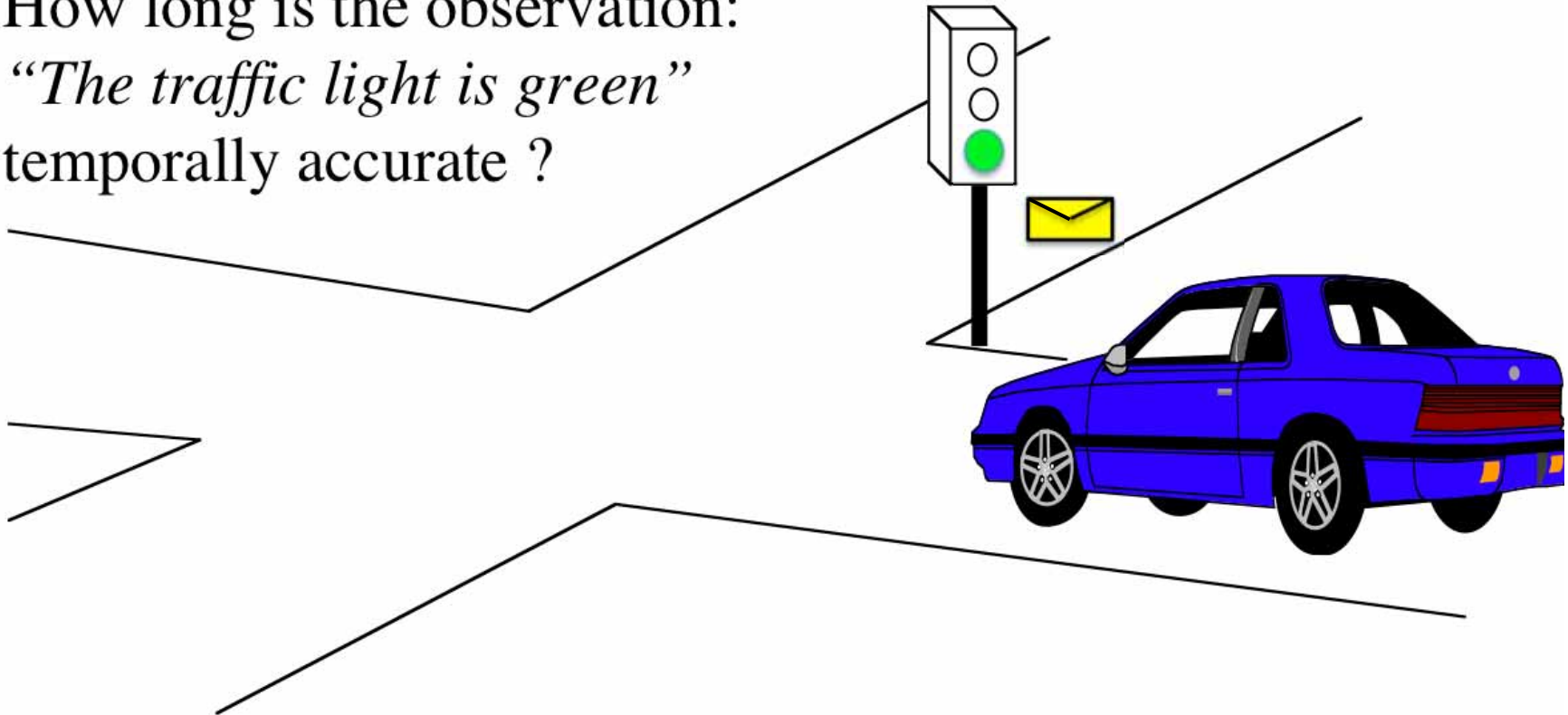
**Sparse**  
Distributed  
Computer



# RT Information has limited temporal validity

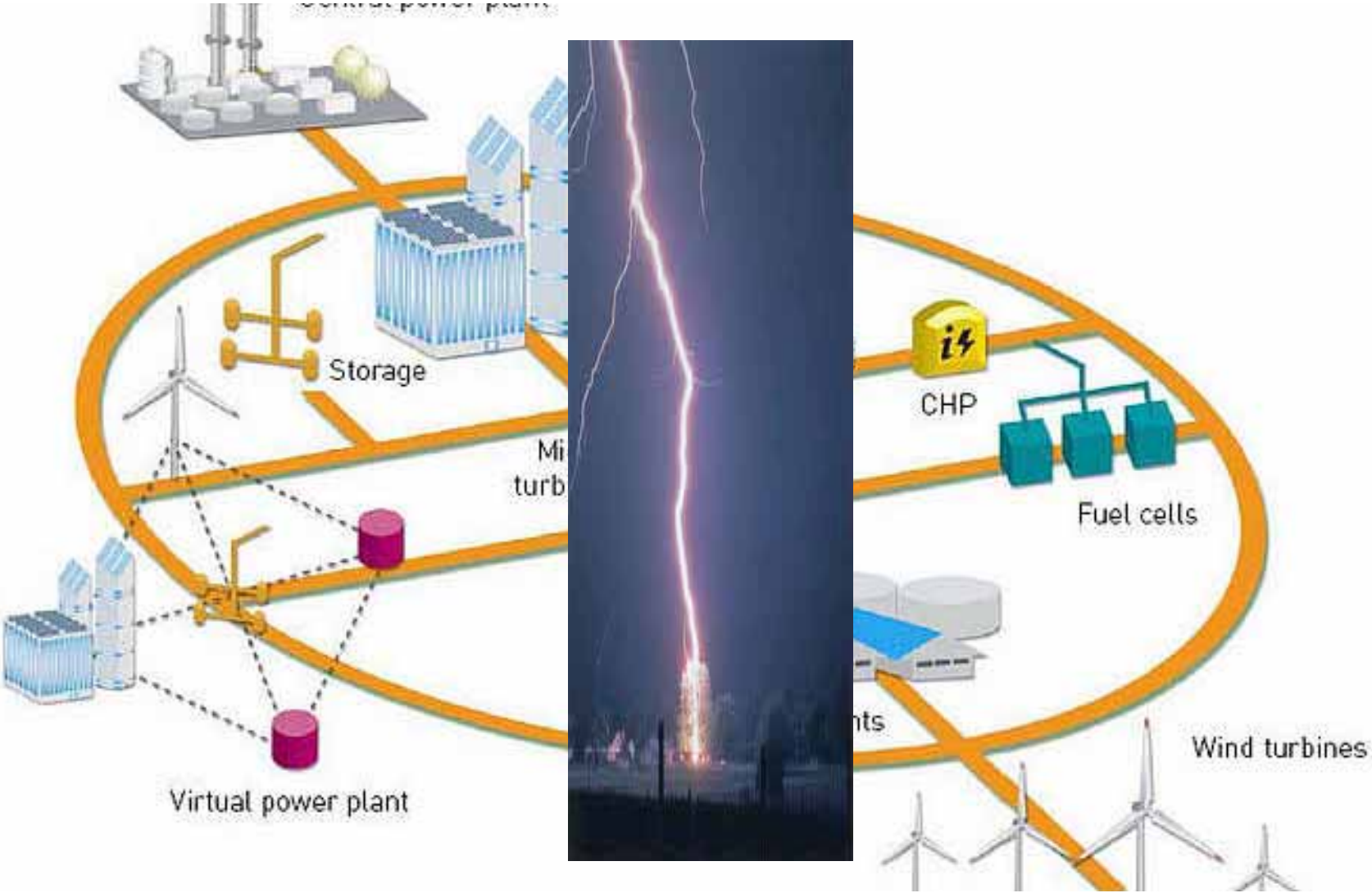
---

How long is the observation:  
*“The traffic light is green”*  
temporally accurate ?



An appropriate model of RT communication must consider *timeliness* as important as *correctness*.

# Timeliness in Smart Grid Control: $\ll 1$ cycle





# Many CPS are *Resilient*

---



*Resilience*

The Courage to Bounce back!

# Resilience

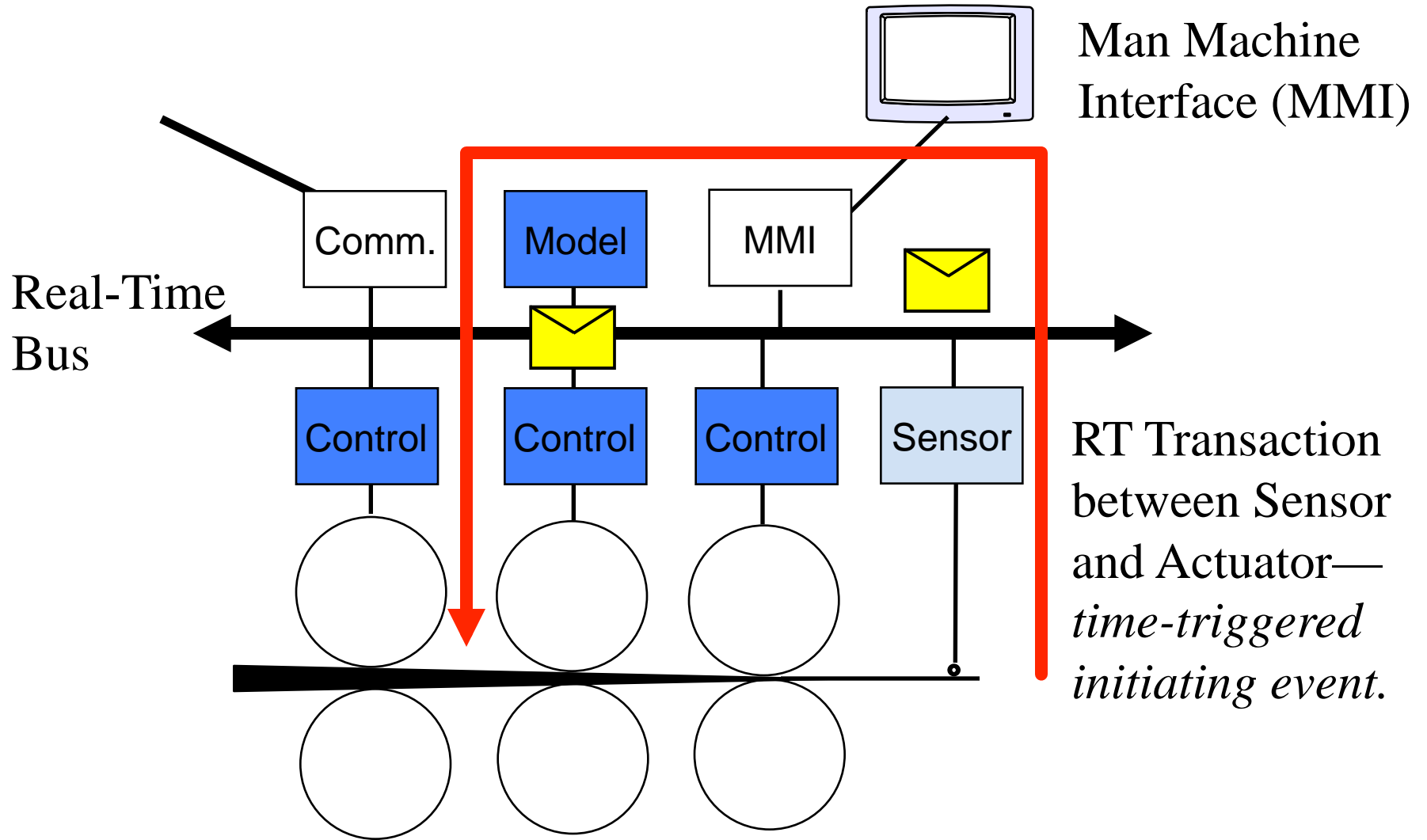
---

In many periodic systems, a single incorrect value will be tolerated by the environment.

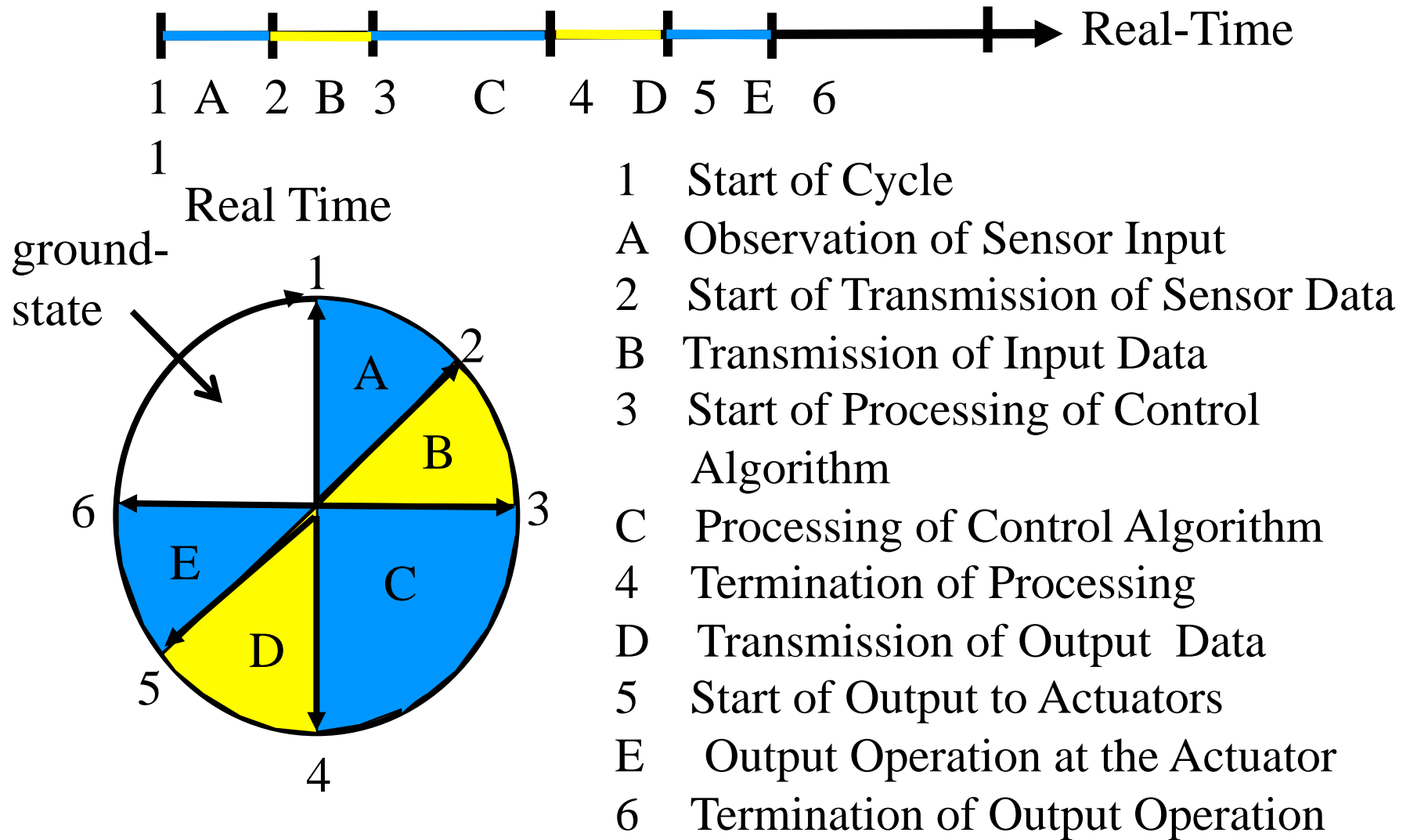
Examples:

- *Control System*: An analog actuator (e.g., a control valve) will move only by a small value in one control cycle.
- *Multimedia System*: A single incorrect pixel or a single missing frame will be masked by the human perception system.

# Control Transactions—*Periodicity*



# Cyclic Representation of Time

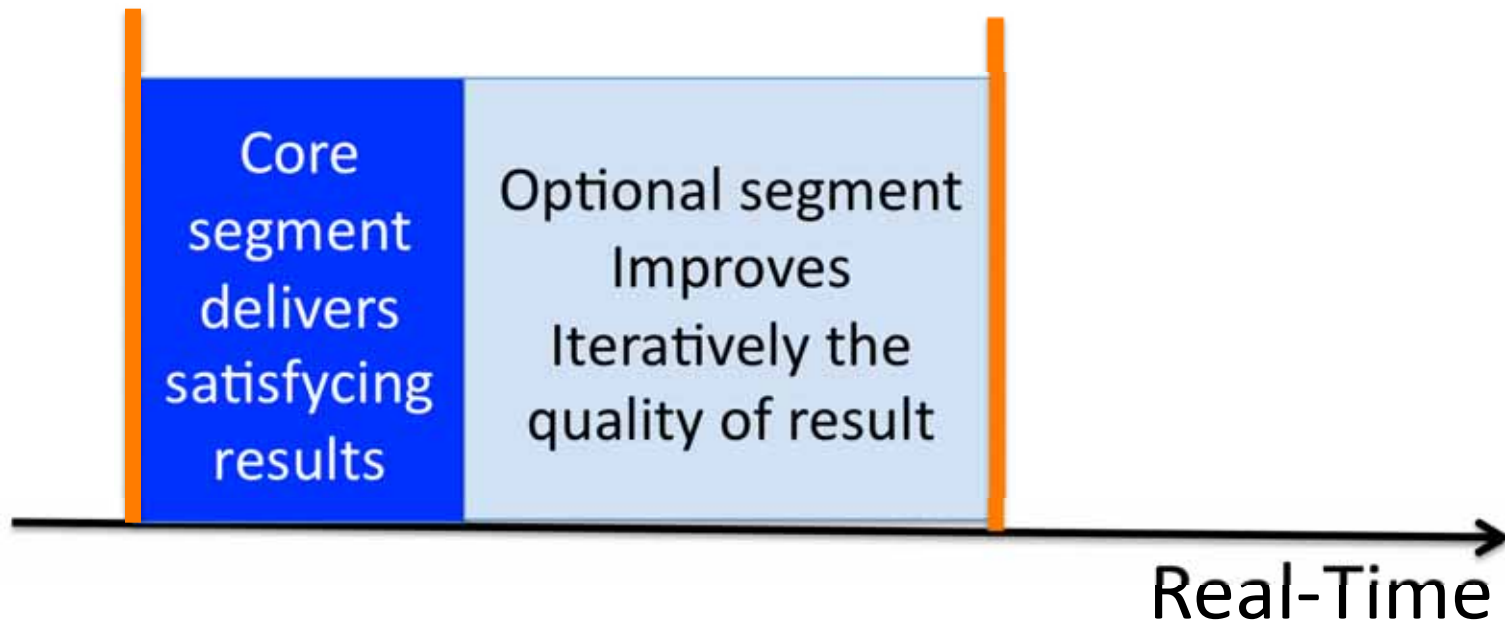


# Anytime Algorithms

---

Start of  
Processing of  
Algorithm

Time-  
Triggered  
Deadline



# Architectural Principles of the *TTA*

---

- ◆ **Component Orientation**—A component is a hardware/software unit
- ◆ **All components are *time-aware***—A global time is provided by the platform.
- ◆ **Separation of Computation from Communication**—Components and Communication systems can be developed independently.
- ◆ **Core Services are **Deterministic****—Modular Certification is supported by the architecture.
- ◆ **Different Integration Levels**—IP-Cores form a Chip, Chips form a Device, Devices form systems
- ◆ **Being Faulty is Normal**

# Time-Awareness

---

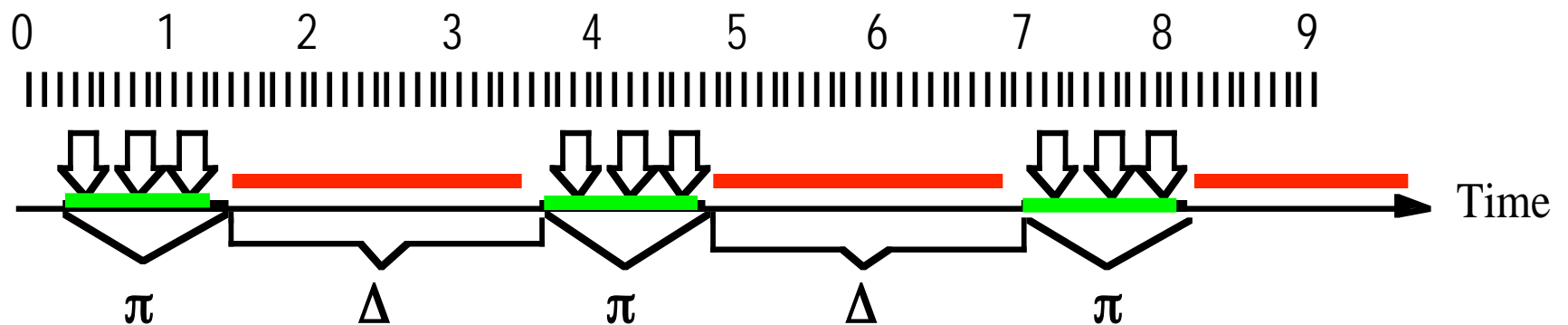
Characteristic for the the TTA is the availability of a global physical time of known precision at every component of the architecture. This global time is used to

- Synchronize actions in the physical world and in the cyber world
- Limit the validity of real-time information
- Control access to shared resources
- Strengthen security protocols
- *Detect failures of fail-silent components*

# Sparse Time Model

Whenever we use the term *time* we mean *physical time* as defined by the international standard of time TAI.

If the occurrence of events is restricted to some active intervals on the timeline with duration  $\pi$  with an interval of silence of duration  $\Delta$  between any two active intervals, then we call the time base  $\pi/\Delta$ -sparse, or **sparse** for short, and events that occur during the active intervals **sparse events**.

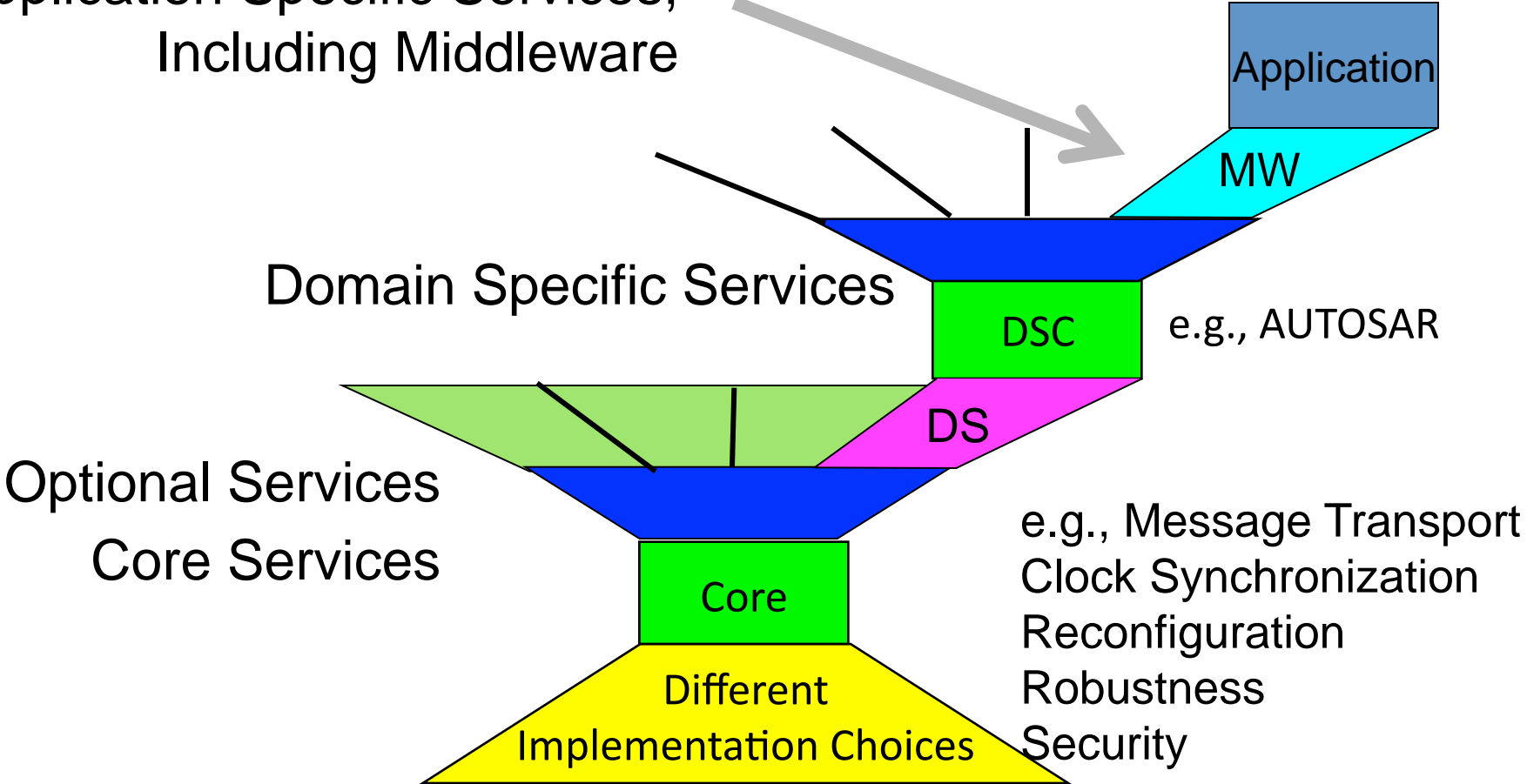


Events  are only allowed to occur at subintervals of the timeline



# TTA Architecture Services Overview

Application Specific Services,  
Including Middleware



# Complexity Management in the TTA

---

The architectural style of the TTA deploys the following *simplification strategies* to reduce the complexity of a design:

- ◆ **Partitioning**: The partitioning of a system into nearly autonomous subsystems (components).--*Physical Structure*
- ◆ **Abstraction**: The introduction of abstraction layers whereby only the relevant properties of a lower layer are exposed to the upper layer--*Structure and Behavior*
- ◆ **Isolation**: The logical and physical containment of subsystems, such that errors are confined.
- ◆ **Segmentation**: The *temporal decomposition* of complex behavior into small parts that can be processed sequentially (“step-by-step”)--*determinism* helps!
- ◆ **Recursion**: Use of the same general concepts and mechanisms at different levels of abstraction

# What is a *TTA* Component?

---

- ◆ ***Hardware/software unit*** that accepts input messages, provides a useful service, maintains internal state, and produces after some *elapsed time* output messages containing the results. It is aware of the progression of *physical time*
- ◆ ***Unit of abstraction***, the behavior of which is captured in a *high-level concept* that is used to capture the services of a subsystem.
- ◆ ***Fault-Containment-Unit (FCU)*** that maintains the abstraction in case of fault occurrence and contains the immediate effects of a fault (a fault can propagate from a faulty component to a component that has not been affected by the fault only by erroneous messages).
- ◆ ***Unit of restart, replication and reconfiguration*** in order to enable the implementation of robustness and fault-tolerance.

# Operating System in the TTA

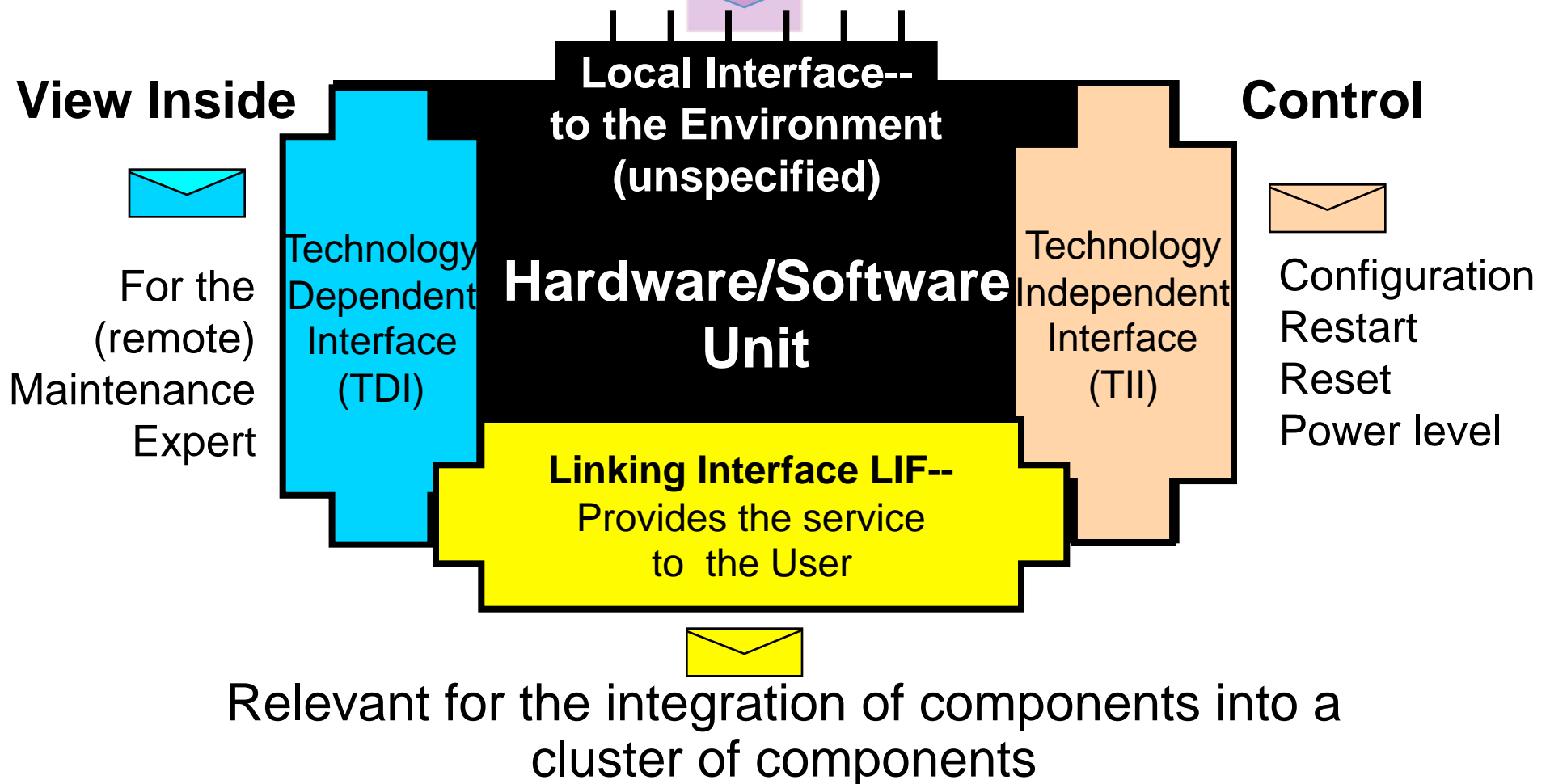
---

In the TTA, the functions of a monolithic operating system are partitioned into

- ◆ A **Mini-RTOS** in each node, and
- ◆ An open-ended set of autonomous **OS Components** that provide operating system services such as
  - Device Controllers (Gateway Components)
  - Integrated Resource Management
  - Security
  - Diagnosis and Robustness
  - Shared Memory Component

# The Interfaces of a *TTA* Component

Connection to local sensors, actuators,  
man-machine interface, other systems



# Software within a Component

---



- ◆ Downloading of the component software, the *Job*, into the component hardware via the *TII interface*.
- ◆ Communicate with other System Components to establish *ports* and *dynamic links*, to reintegrate components after a transient fault etc.
- ◆ Global time management
- ◆ Provision of API Services (e.g., *send* and *receive* of a message)
- ◆ Scheduling of the tasks within a component
- ◆ Service of the *TII Interface* to *reset*, *start*, and *terminate* the operation of a component
- ◆ Provision of Generic Middleware Services (GEM)

# Linking Interface Specification

---

The Linking interface is a *message-based* interface. Its specification consists of three parts:

- ◆ **Transport Specification:** contains the information needed to transport a message from the sender to the receiver. Temporal properties are part of the transport specification.
- ◆ **Operational Specification:** specifies the *syntactic structure* of the bit stream contained in the message and establishes the *message variable names* that point to the concepts at the meta level.
- ◆ **Meta-Level Specification:** assigns meaning to the message variable names established by the operational specification.

# Operational Specification

---

## Operational Specification:

- ◆ **Operational Input Interface Specification**
  - Syntactic Specification (e.g. by IDL)
  - **Input Assertion**
- ◆ **Operational Output Interface Specification**
  - Syntactic Specification (e.g. by IDL)
  - **Output Assertion**
- ◆ **Interface State:** contents of data structure at ground state .

**Specification of the assertions is important.**

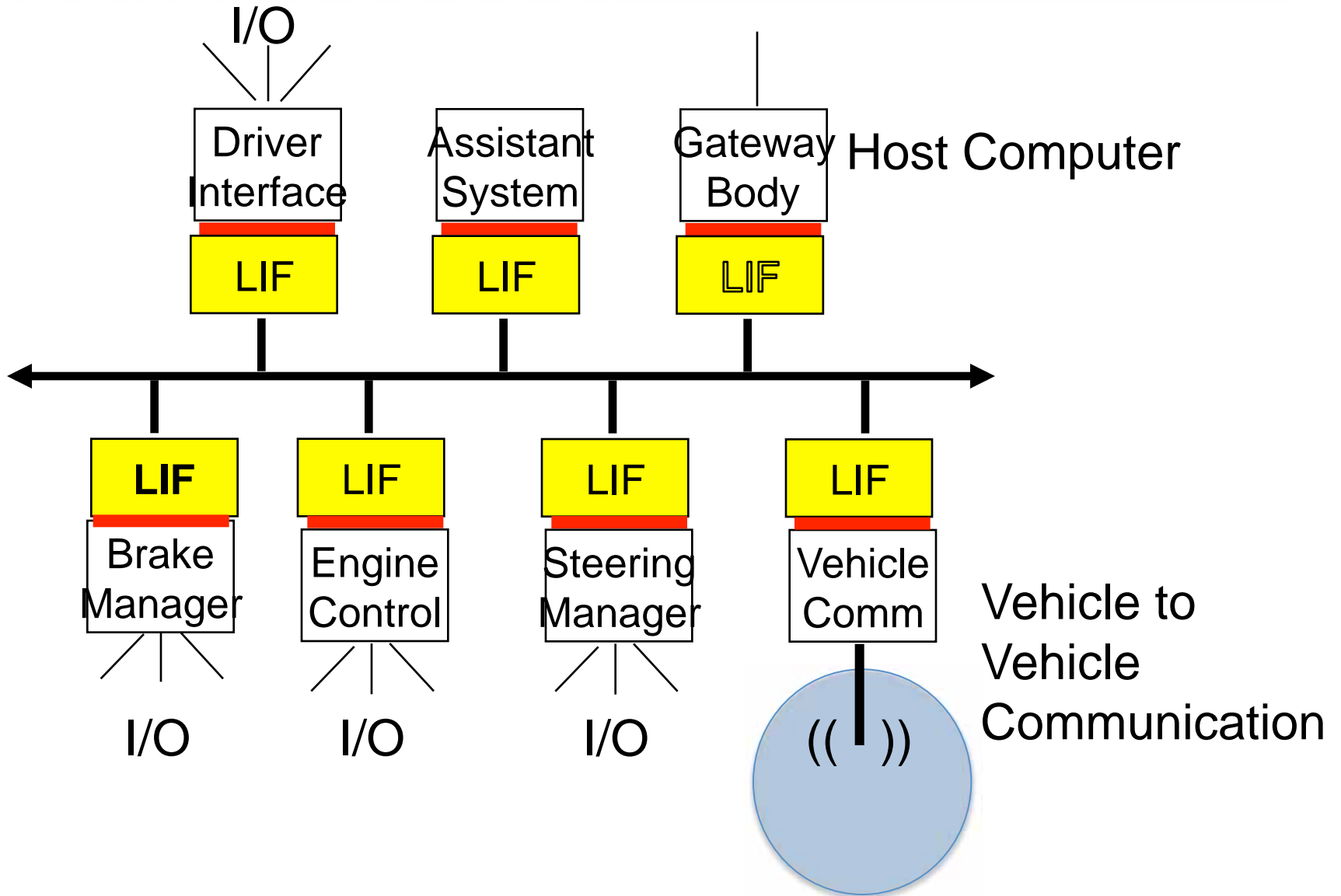


# Meta-Level Specification: Interface Model

---

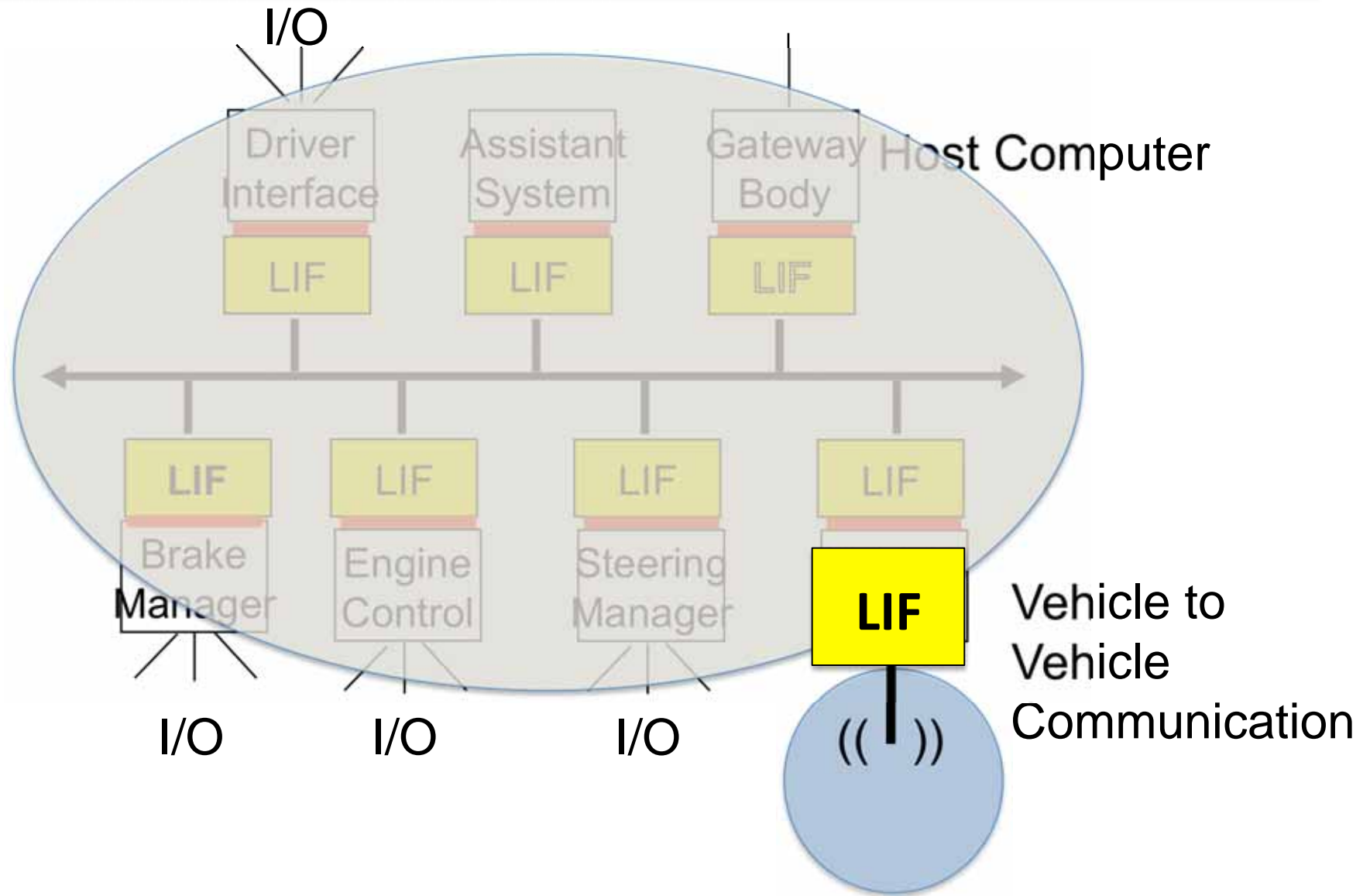
- ◆ specifies the relationship between the *real world* and the meaning of the *message variables*.
- ◆ must be expressed with concepts that are familiar to the conceptual world of the intended users.
- ◆ must include the context of use, i.e. a (constrained) model of the environment.
- ◆ The brittleness of natural language cannot be avoided in open components.
- ◆ Meta-level specification remain often informal -- *Formalization increases the precision, but at the same time increases the distance to reality (Chargaff)*
- ◆ Beware of *pseudo-formalism*.

# Example of a *Cluster* of Components



# Example of a Cluster--*Recursion*

---



# Gateway Components

---

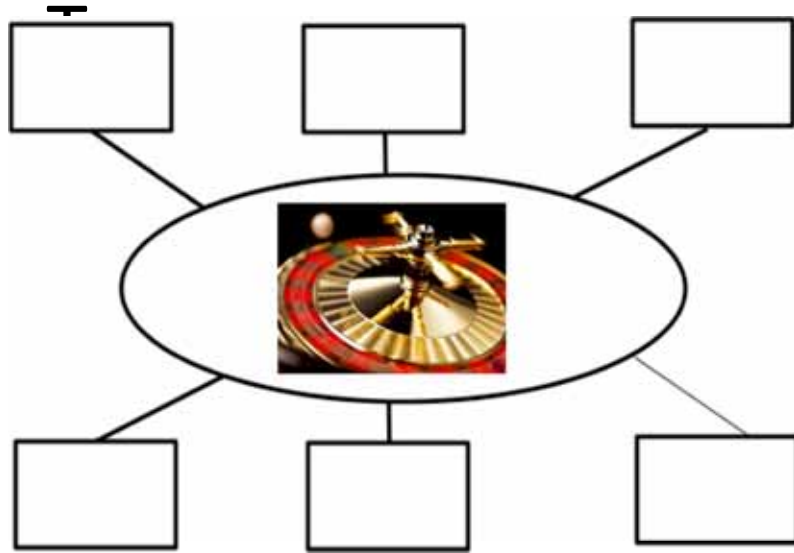
Gateway components can connect two clusters that may be based on different *architectural styles*:



- The representation of the information in the two clusters will be different, but the ***semantic content*** of the *message variables* must be the same.

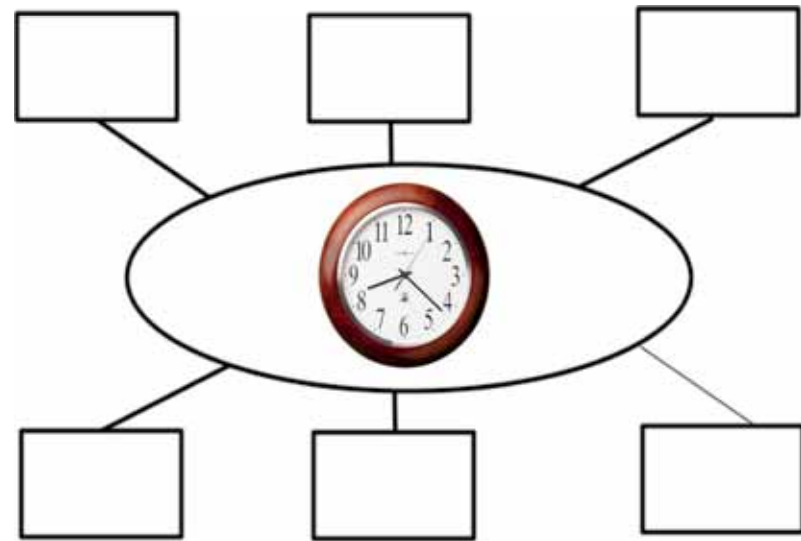
# Communication in the TTA

---



**Competition**  
**Best-Effort**  
**Probabilistic**

versus



**Cooperation**  
**Time Triggered**  
**Deterministic**

# ***Unidirectional Deterministic Multi-cast Message***

---

- ◆ ***Uni-directionality*** is required to
  - decouple communication from computation
  - decouple the sender behavior from the receiver behavior
- ◆ ***Determinism*** is required to
  - establish timeliness
  - simplify the reasoning about the behavior (*modus ponens*)
  - simplify testing (repeatable test cases)
  - be able to implement active replication (TMR)
  - support the certification
- ◆ ***Multi-cast*** is required to support
  - the independent observation of the component behavior
  - the ***support of diagnosis***
  - replication of state at multiple components
  - Triple Modular Redundancy

# Message Types

---

- ◆ ***Periodic Messages (TT) (core)***
  - No queues, non-consuming read, update in place
  - Temporal guarantees
- ◆ ***Sporadic Messages (ET) (core)***
  - characterized by two queues, one at the sender site and one at the receiver site
  - Exactly once semantics
  - Normally best effort timing
- ◆ ***Real-time Data Streams***
  - Guaranteed bandwidth and timing (core)
  - Queues with watermark management (optional)

***Openness: Any communication protocol (wire-bound or wireless) that provides these services can be used***

# Core: Time-Triggered Communication

---

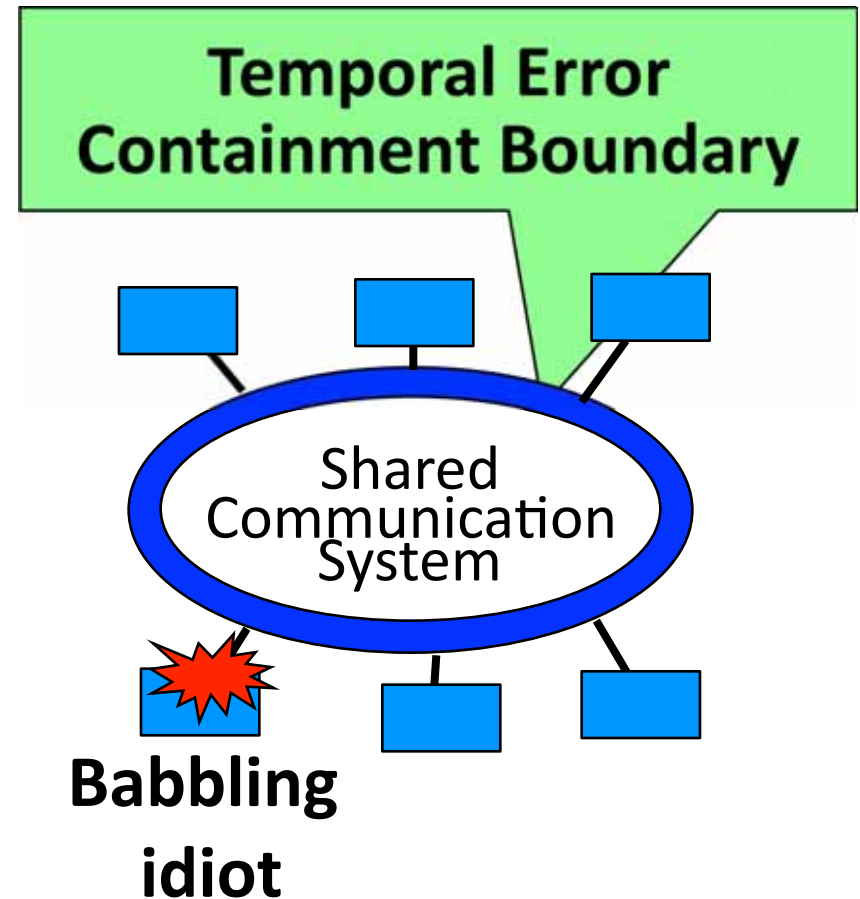
- In a time-triggered communication system, the sender and receiver(s) agree a priori on a cyclic time-controlled conflict-free communication schedule for the sending of time-triggered messages.
- In every period, a message is sent at exactly the same phase.
- The control and data flow is unidirectional.
- Error detection is in the sphere of control of the receiver, based on his *a priori* knowledge of the arrival instants of messages.



# Temporal Error Containment by the CS

It is *impossible* to maintain the communication among the correct components of a RT-cluster if the temporal errors caused by a faulty component are not contained.

*Error containment of an arbitrary temporal node failure requires that the shared Comm. System is a self-contained FCU that has temporal information about the allowed behavior of the nodes—*  
***It must contain application-specific state.***



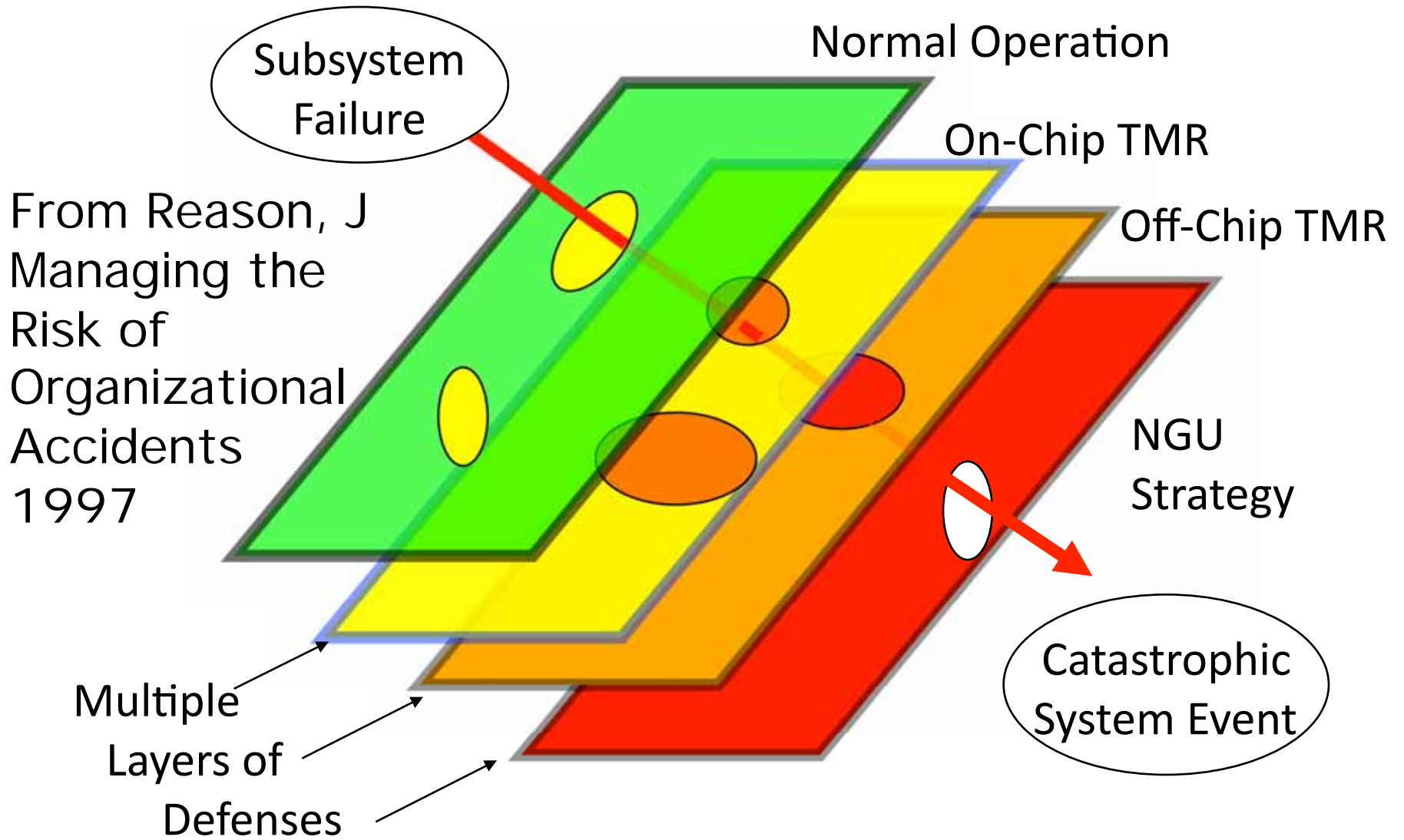
# Levels of Fault Mitigation in the TTA

---

- I. Normal Operation
- II. Swift Component Recovery after a transient fault
- III. On-Chip TMR: to handle transient and permanent faults within a chip
- IV. Off-Chip TMR: to handle a transient and permanent fault of a total chip.
- V. NGU (Never-Give-Up) Strategy: to handle multiple correlated transient faults.

# The *Swiss-Cheese* Model

---



# Fault Classification

---

We distinguish between:

- **Transient faults** (hardware, Heisenbugs, operation): Recovery by a ground state monitor
- **Permanent faults**: masking by replicated self-checking components or by TMR.

In the TTA we focus on the fast recovery after a transient fault. Diagnosis of the fault is postponed and allocated to a diagnostic system.

# State of a CPS

---

*The state enables the determination of a future output solely on the basis of the future input and the state the system is in. In other word, the state enables a “decoupling” of the past from the present and future. The state embodies all past history of a system. Knowing the state “supplants” knowledge of the past. . . . Apparently, for this role to be meaningful, the **notion of past and future** must be relevant for the system considered.*

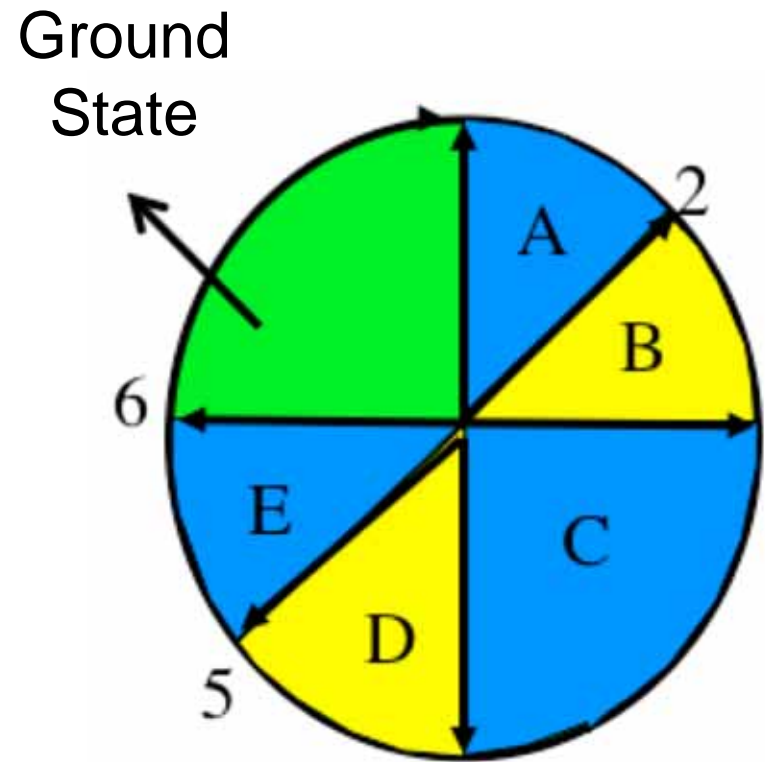
Mesarovic, p.45

# Ground State

---

The ground state of a component is the state of a component at an instant when all tasks are inactive and where all communication channels are empty.

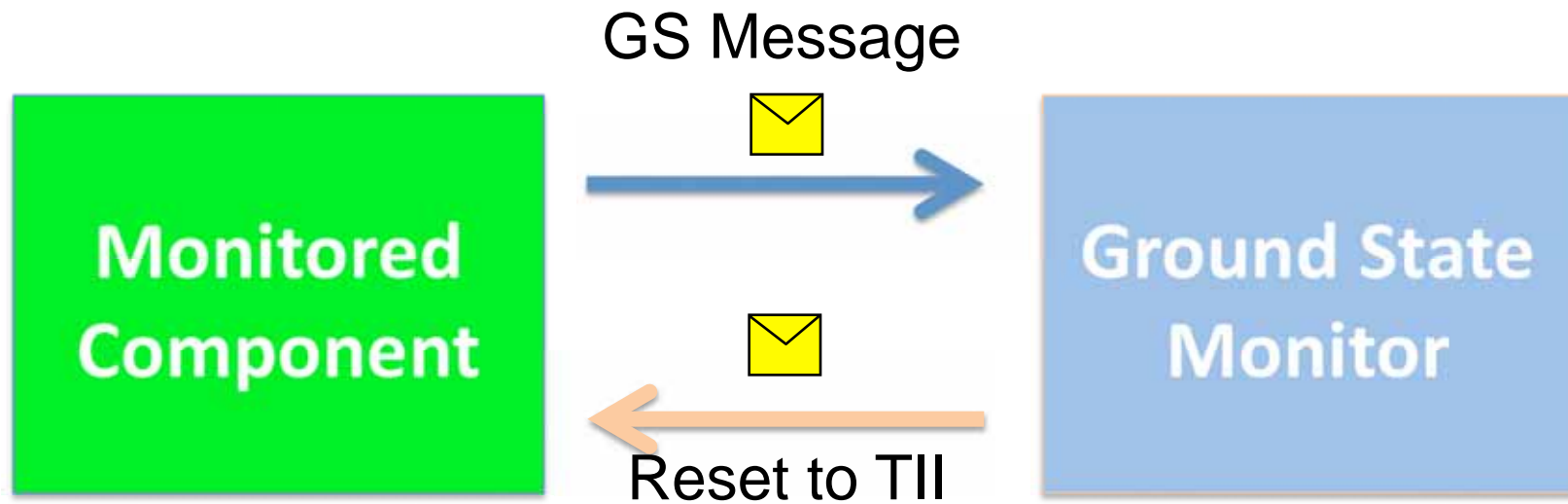
A relevant ground state must be provided to reintegrate a component after a hardware reset.



# Ground State Monitor

---

The ground state monitor is an independent FCU that receives periodically a ground-state message, checks the message and, if erroneous, resets the component and provides an estimated ground state for the next recovery instant.



# Restart after a Failure

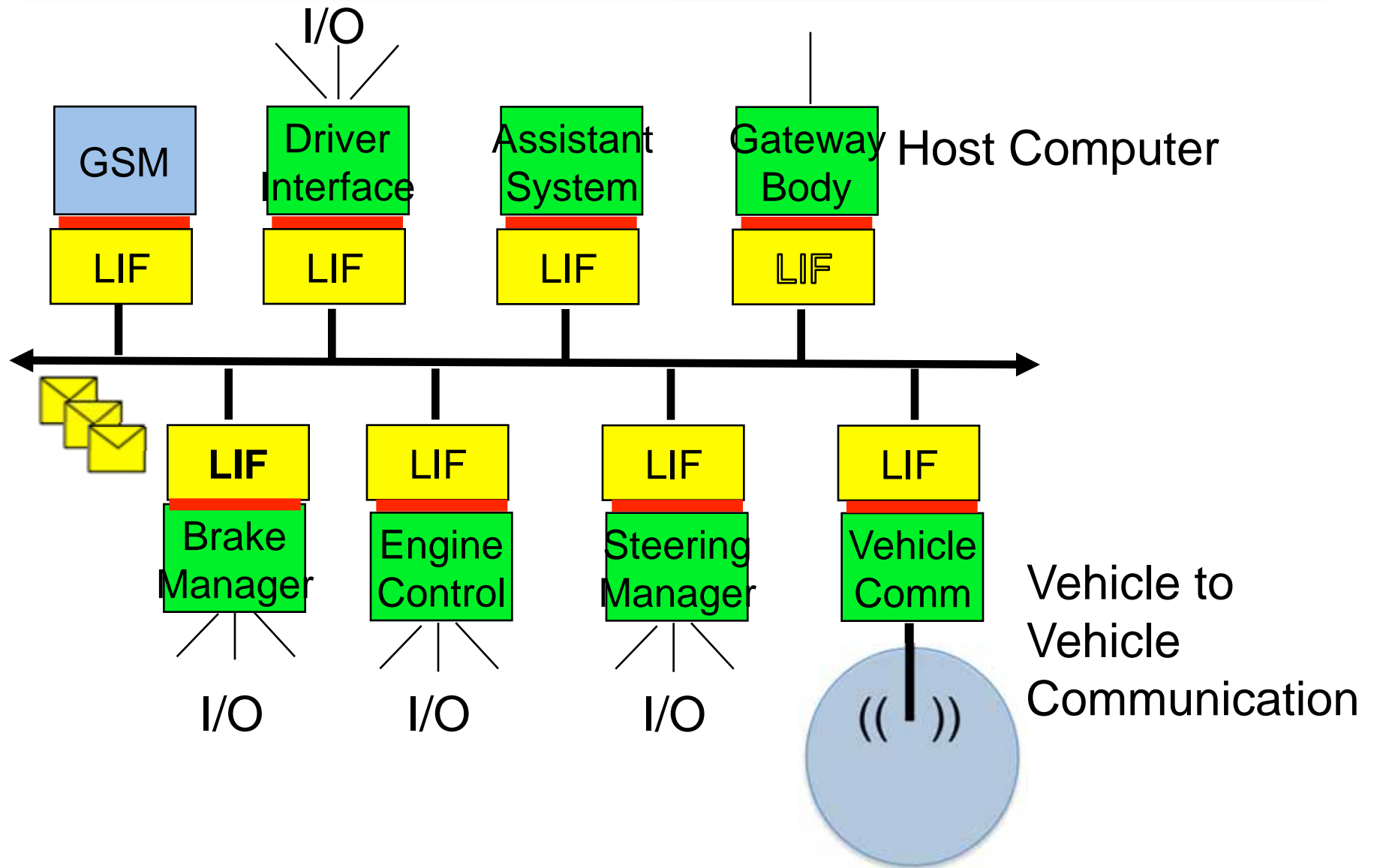
---

- The Monitored Component sends periodically the G-State to the GS-Monitor.
- The GS-Monitor performs state-estimation to the next restart instant.
- In case a GS-state message is missing (fail silence) or wrong (non-fail silence) the GS monitor sends a reset/restart message to the failed component.
- After the receipt of a reset message at the TII Interface, the component performs a hardware reset and restarts its operation with the ground-state that is contained in the restart message.

TII-Addr	Time	Ground State at Reintegration Instant
----------	------	---------------------------------------

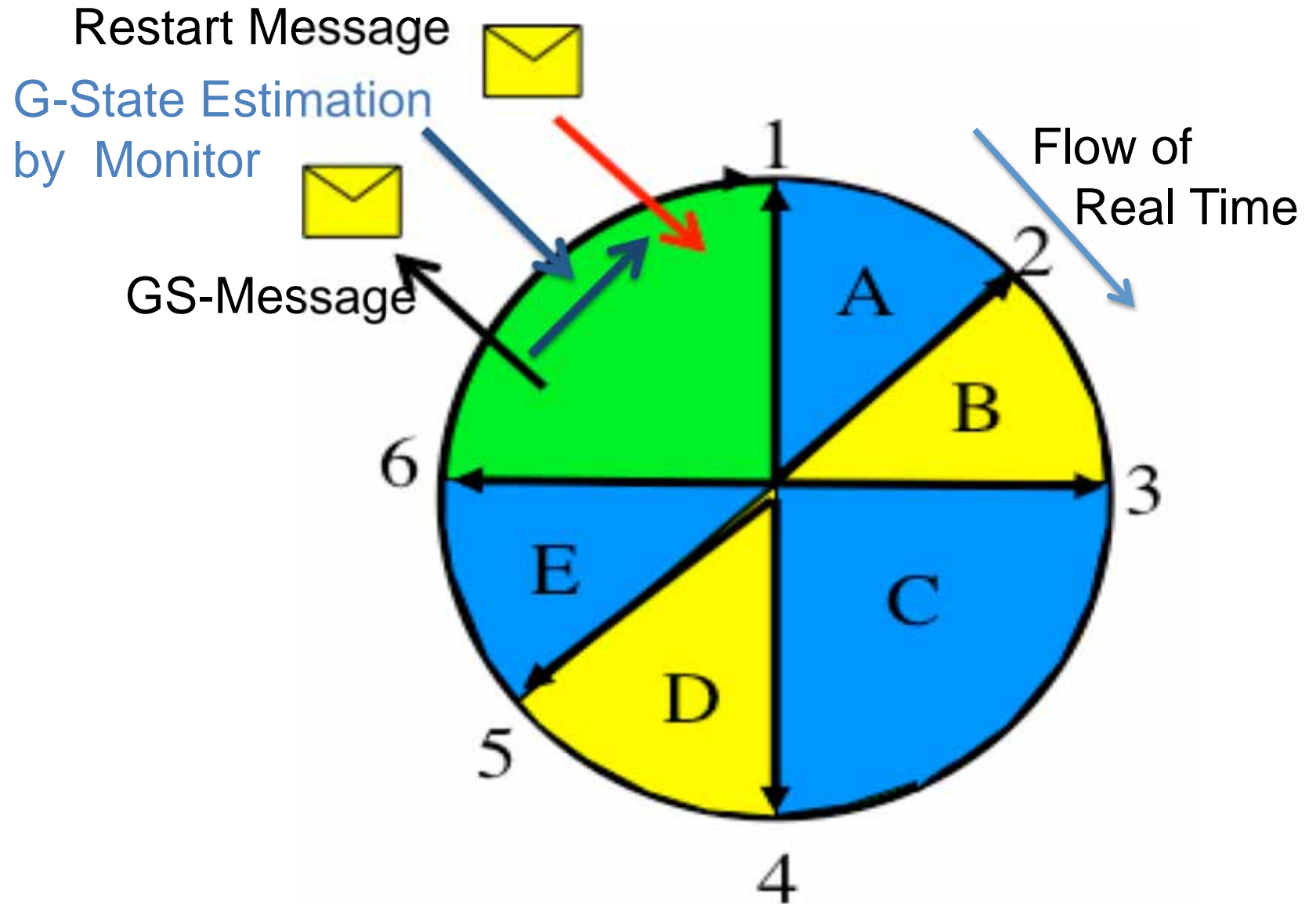


# Example of a *Cluster* of Components

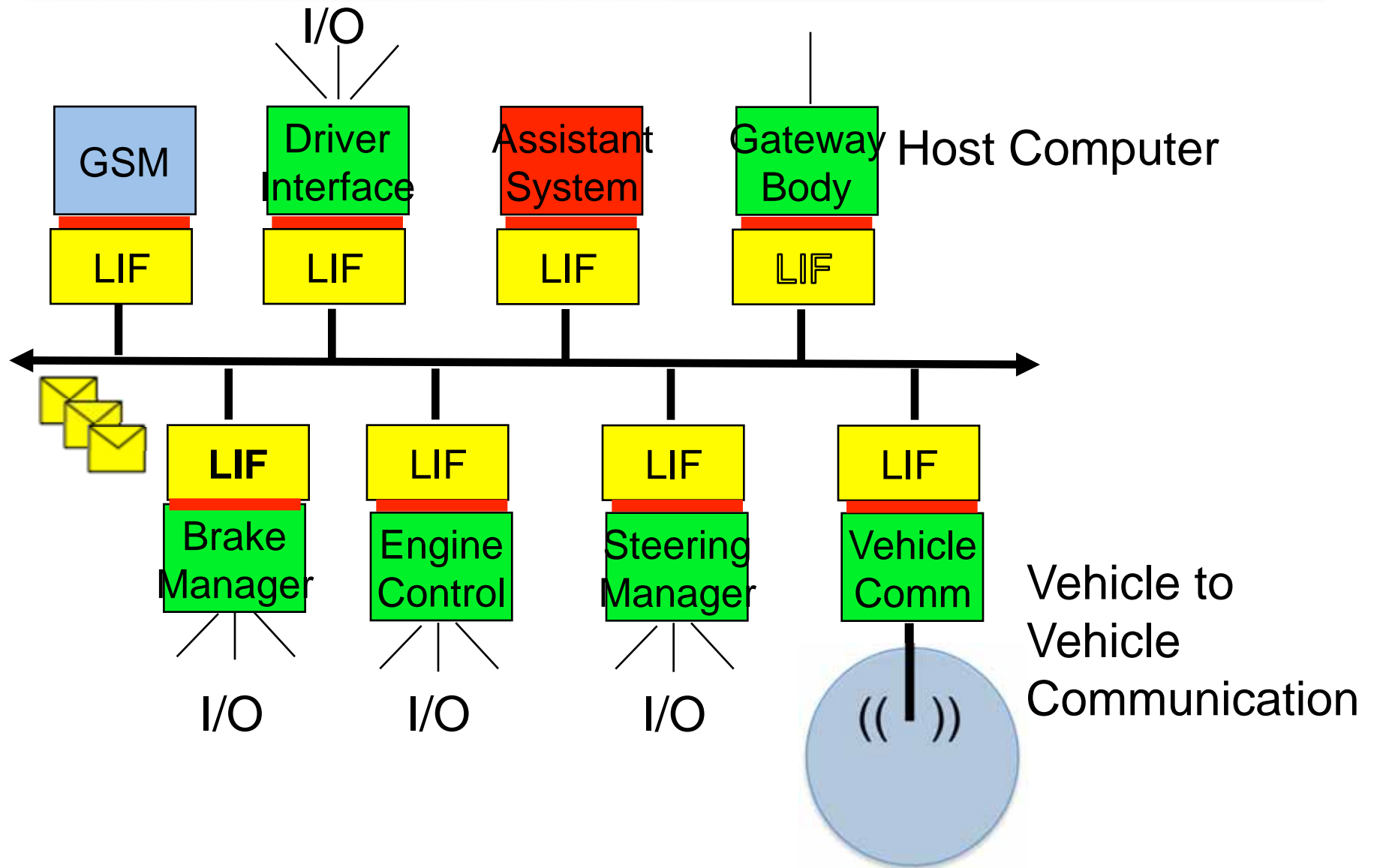


# Ground State Recovery

---



# Example of a *Cluster* of Components



# Case Distinction

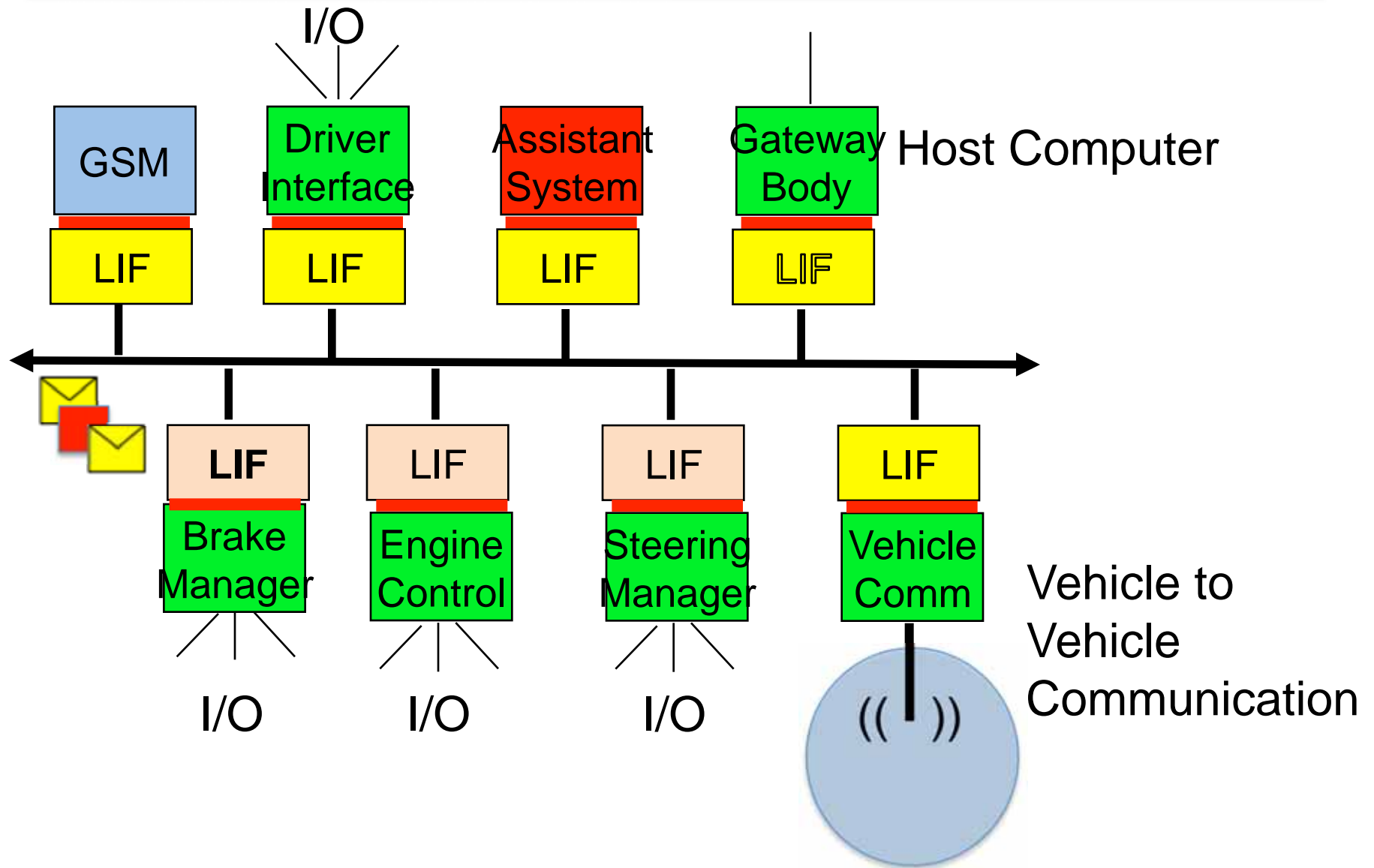
---

A failure of the faulty component is

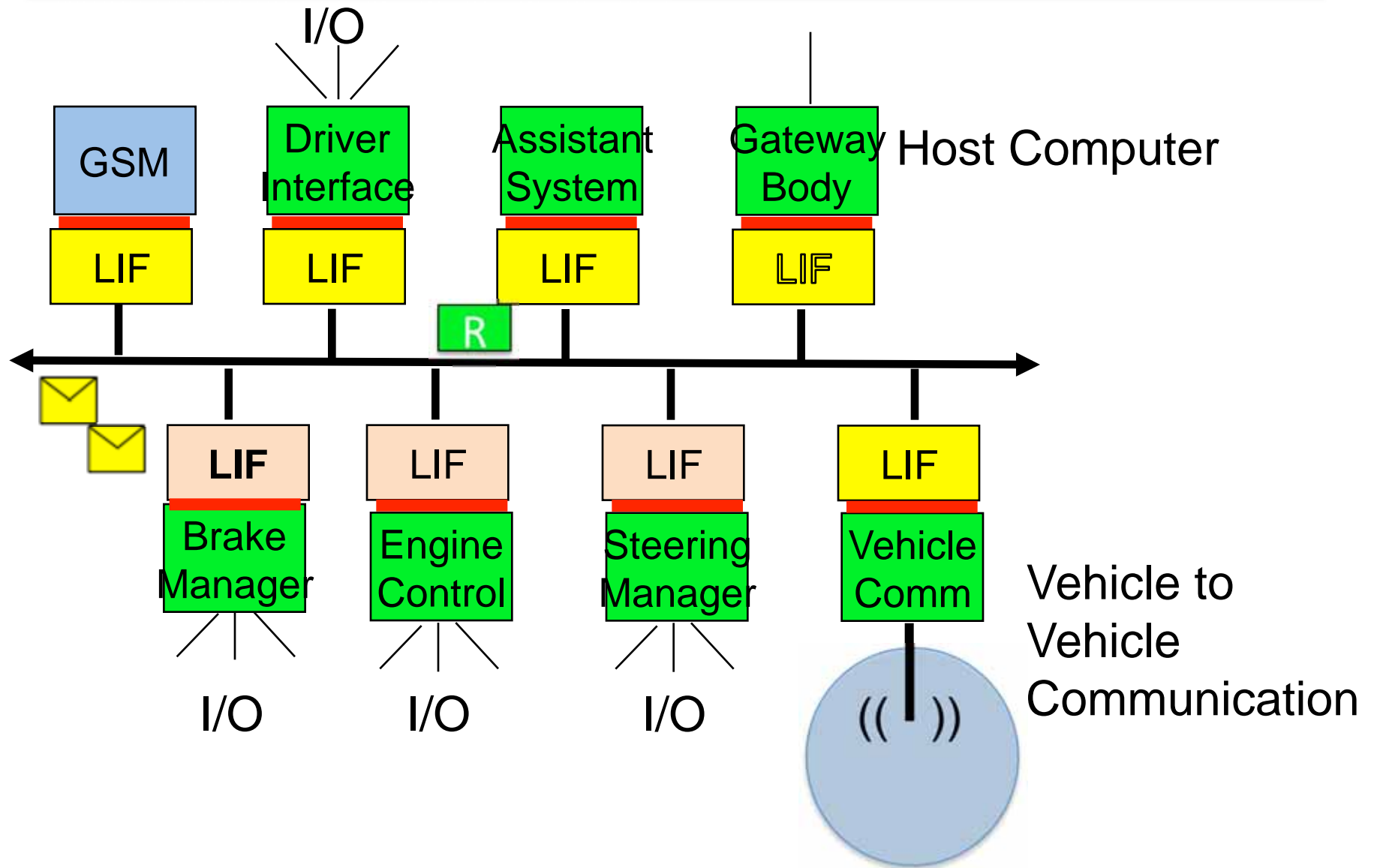
1. *Fail silent*: no output message is produced
2. *Non-fail silent*: the incorrect output message does not pass the output assertion of the sender
3. *Non-fail silent*: the incorrect output message does not pass the input assertion of the receiver
4. ***Non-fail silent*: the incorrect output message passes all assertions.**

What is the probability of for each one of these cases?

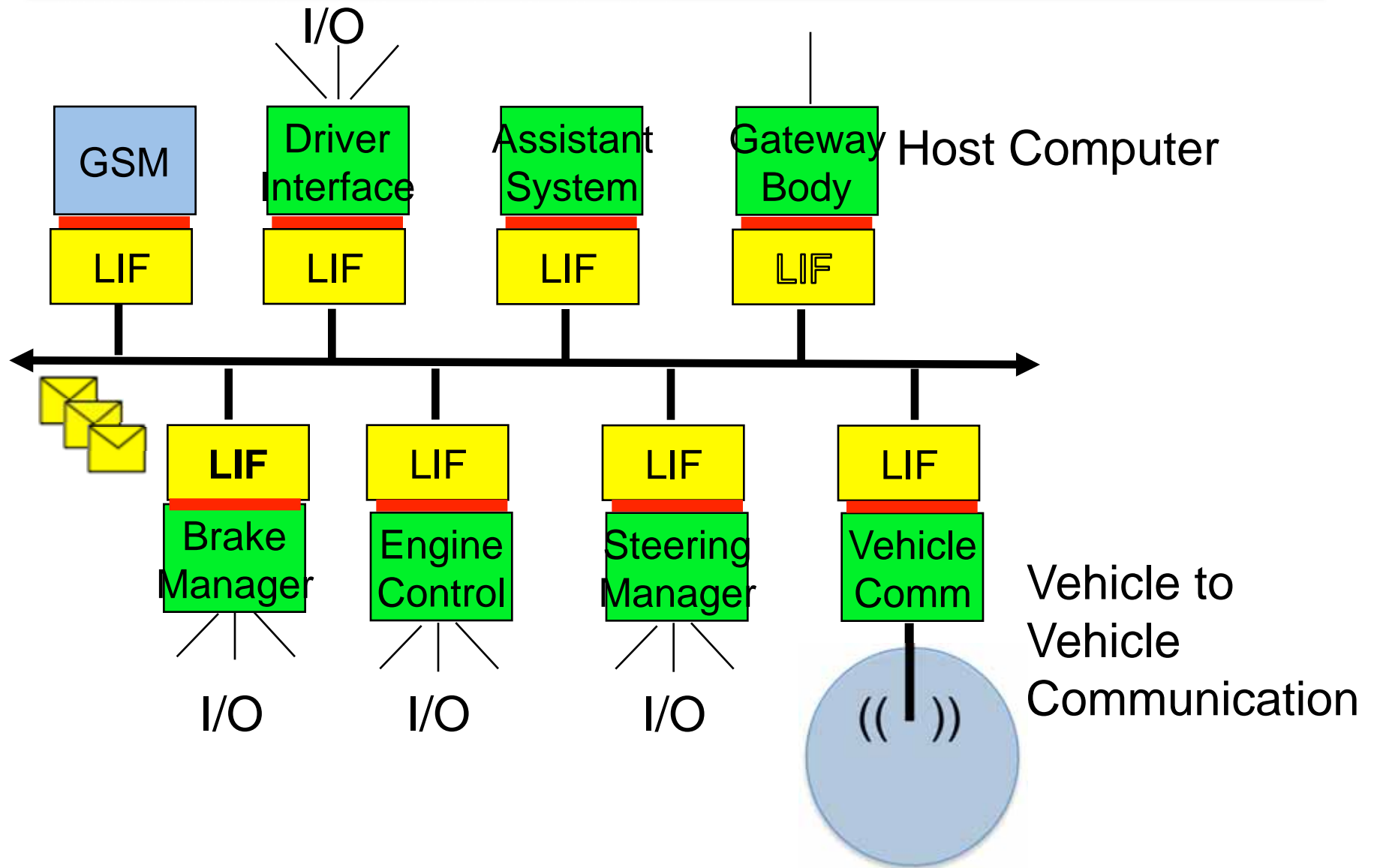
# Example of a *Cluster* of Components



# Example of a *Cluster* of Components



# Example of a *Cluster* of Components



# Structure of the *GENESYS Chip*

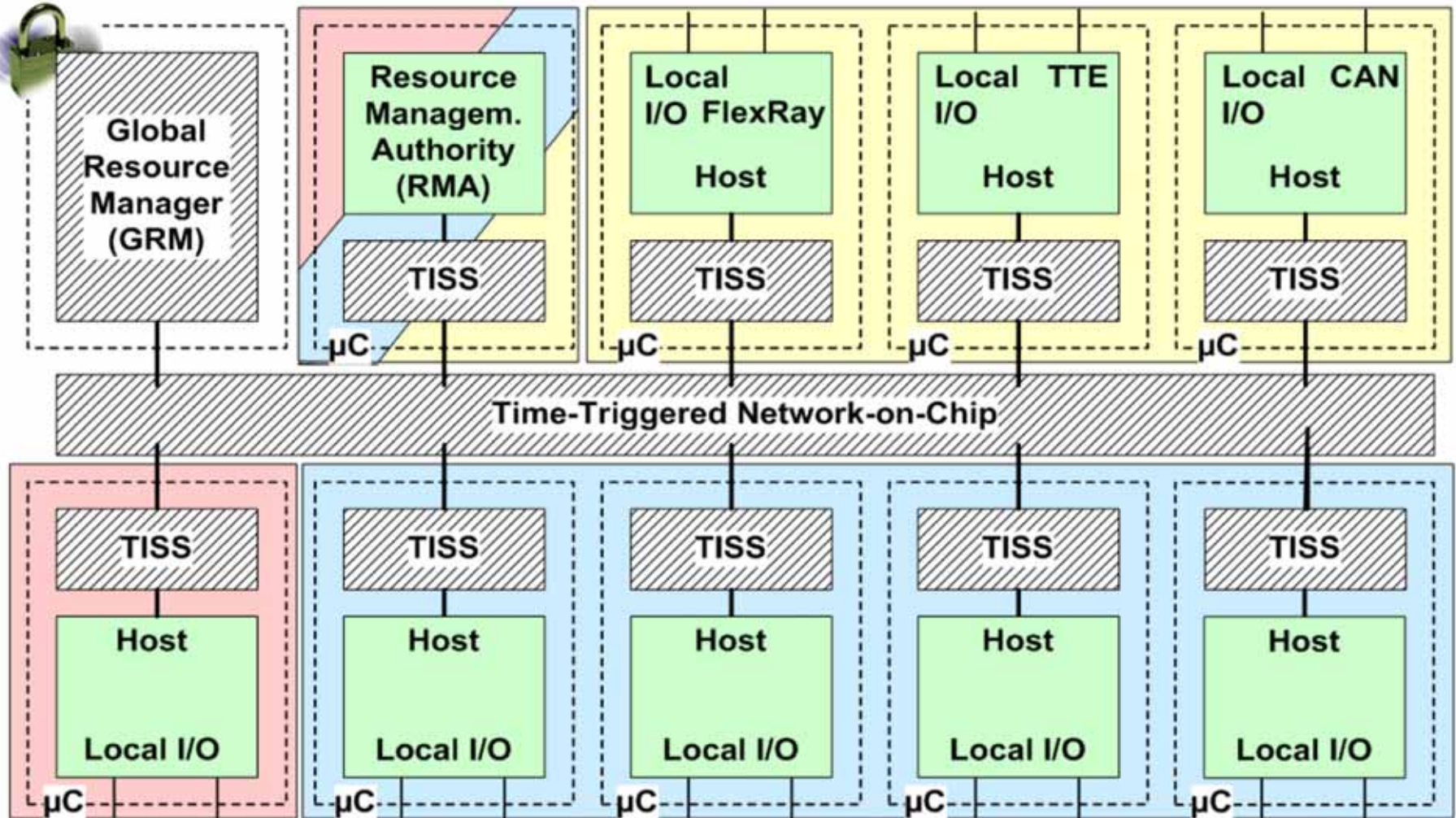
---

The GENESYS Chip consists of the *Trusted Subsystem* and IP Cores:

- ◆ The *Trusted Subsystem* is formed by the Trusted Resource Monitor (TRM) and Trusted Interface Subsystems (TISS) to the components and the TTNoC.
- ◆ The IP-cores are connected to the TISSes. The TISS will contain an arbitrary temporal failure (hardware or software) of a non-trusted IP core



# Chip Level: GENESYS Prototype



# What is Needed to Implement TMR?

---

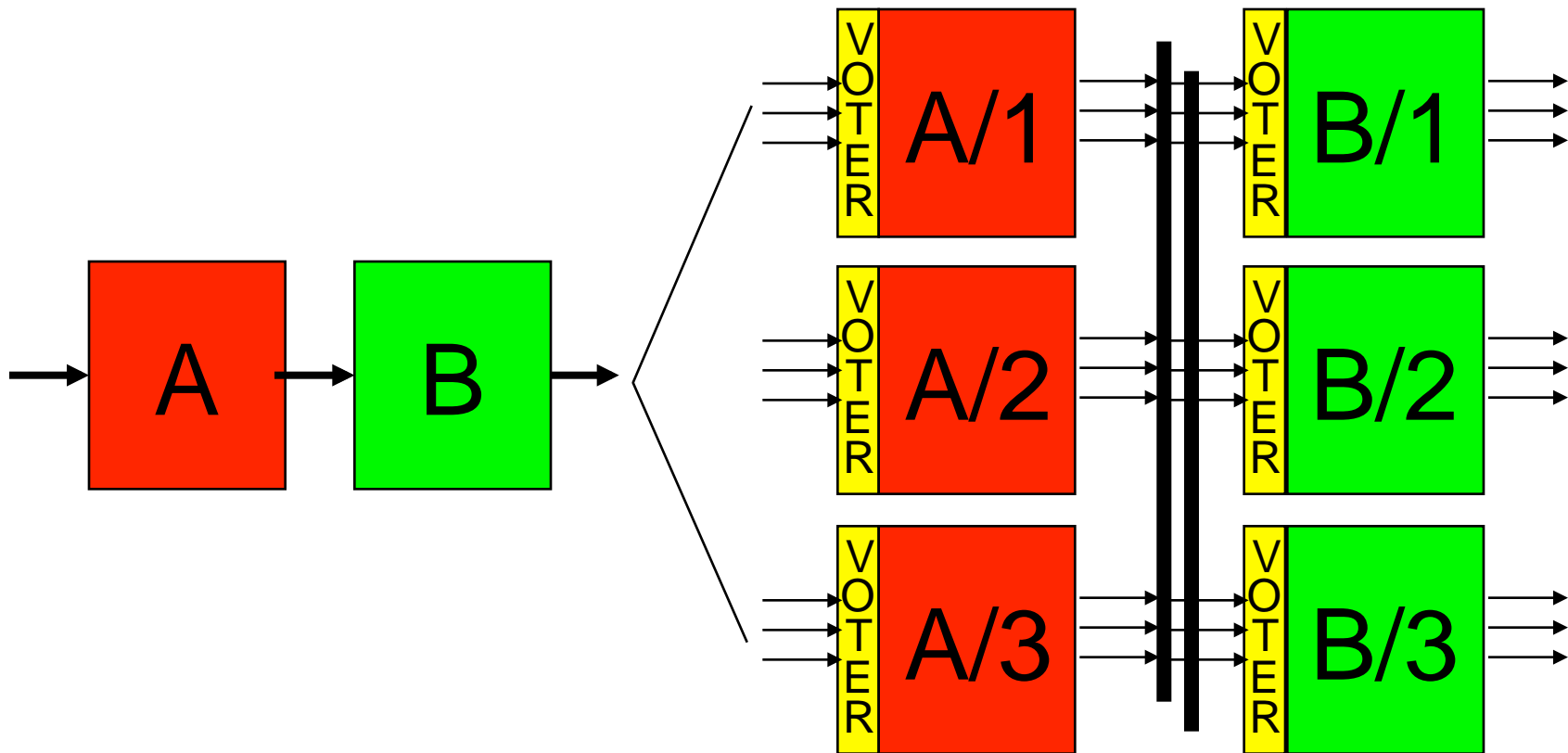
What architectural services are needed to implement Triple Modular Redundancy (TMR) at the architecture level?

- ◆ Provision of an Independent Fault-Containment Region for each one of the replicated components
- ◆ Synchronization Infrastructure for the components
- ◆ Predictable Multicast Communication
- ◆ Replicated Communication Channels
- ◆ Support for Voting
- ◆ Deterministic (*which includes timely*) Operation
- ◆ Identical state in the distributed components

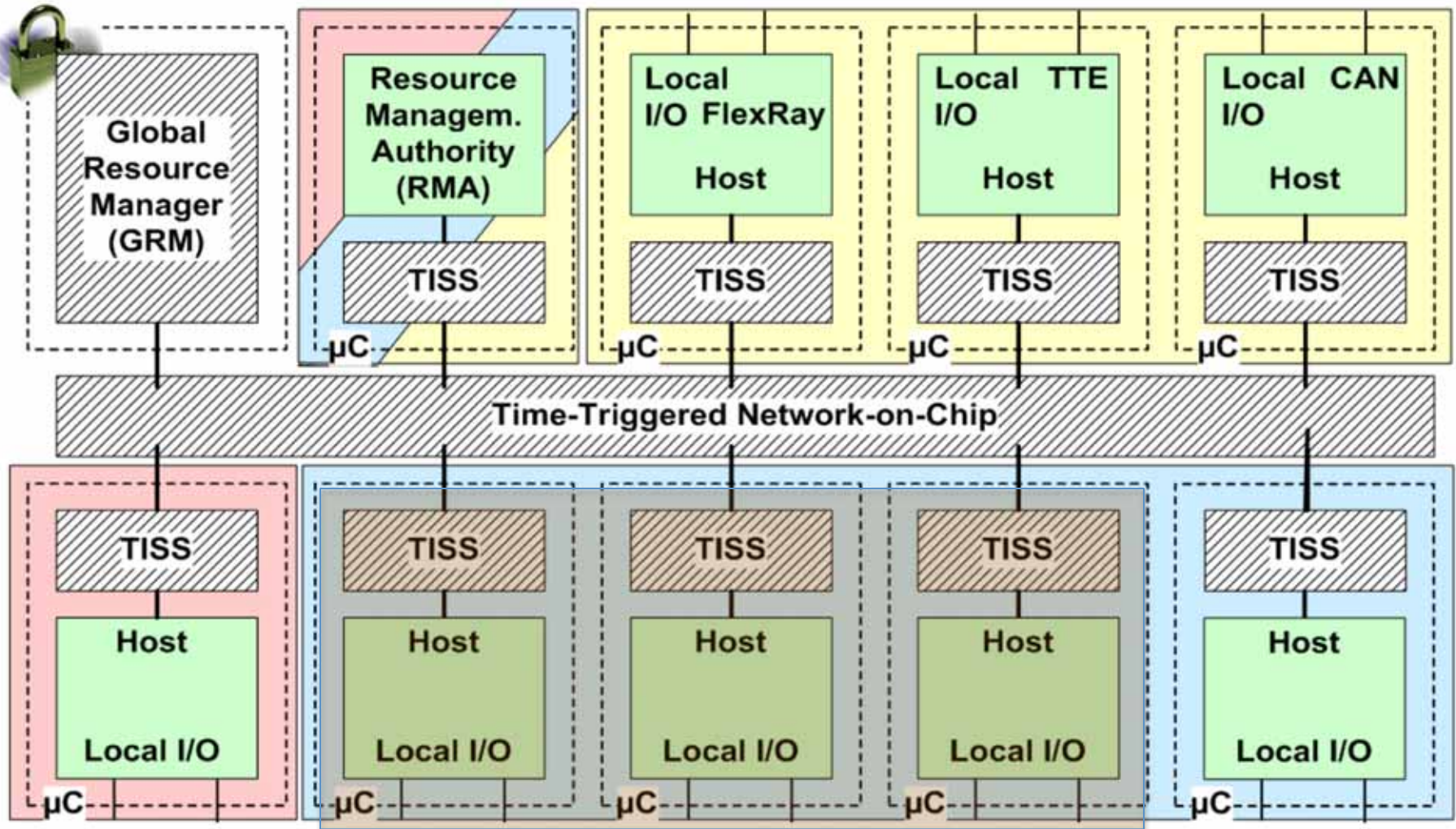
# Triple Modular Redundancy (TMR)

---

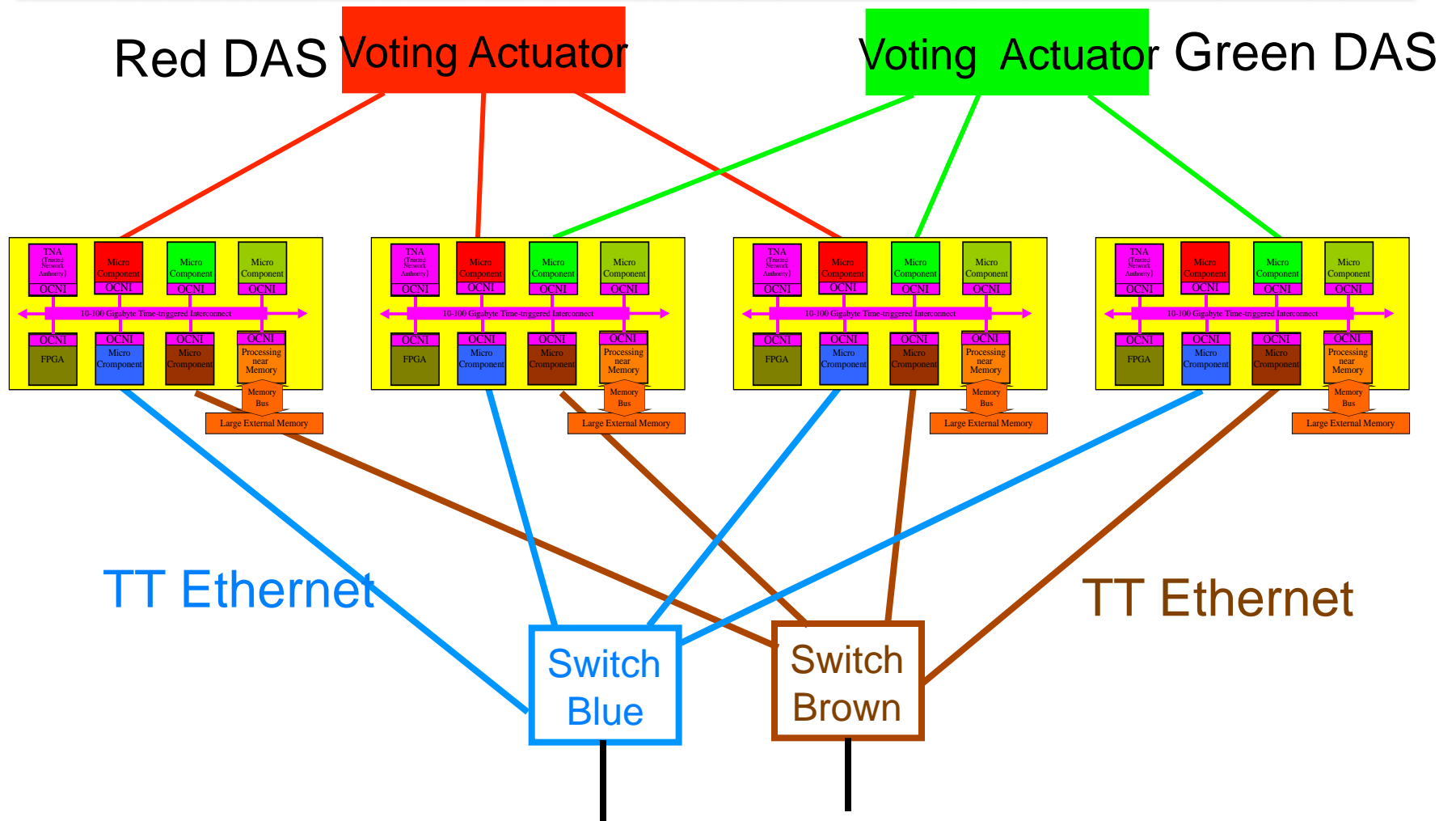
*Triple Modular Redundancy (TMR)* is the generally accepted technique for the mitigation of component failures at the system level:



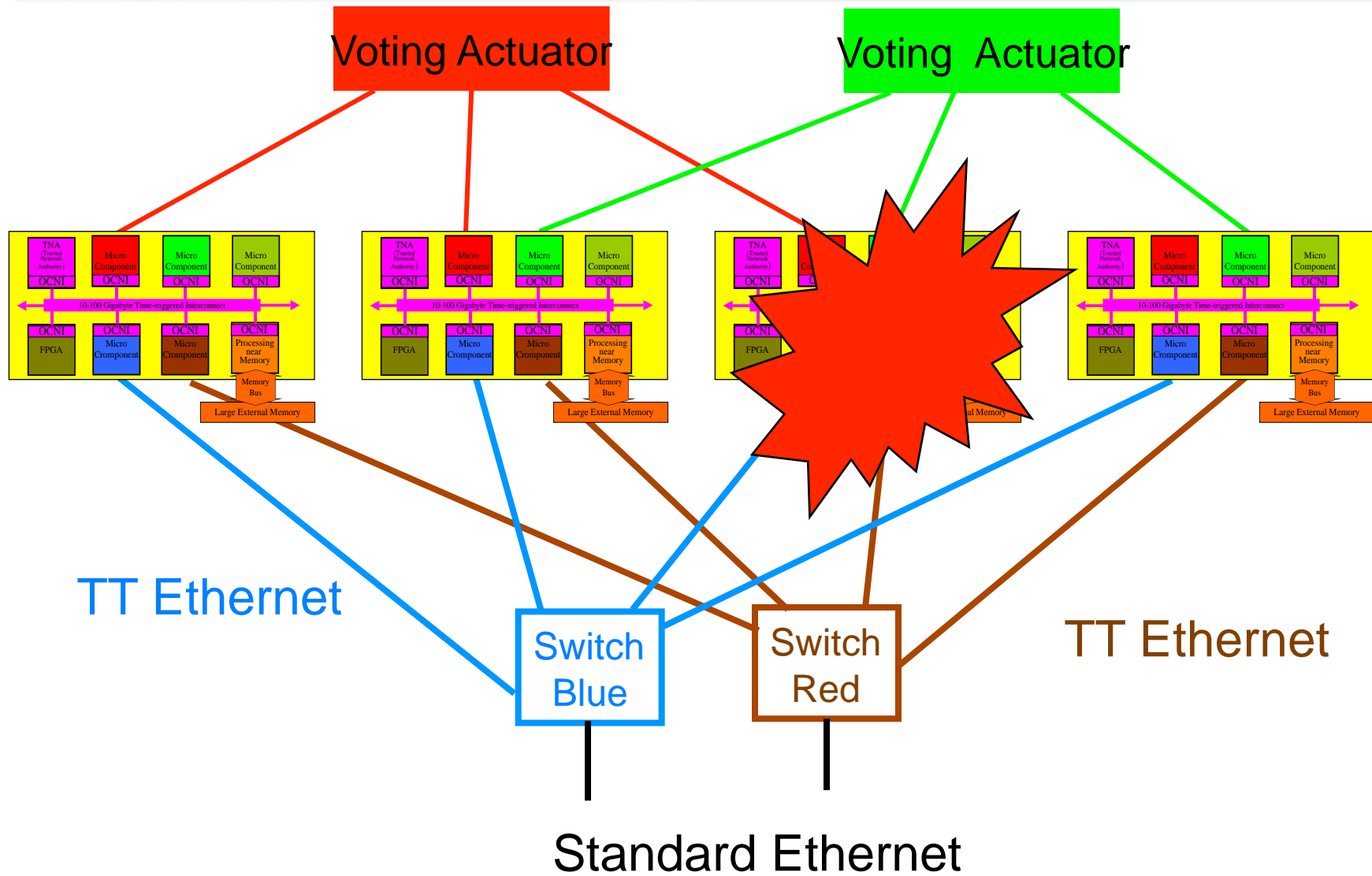
# On-Chip TMR



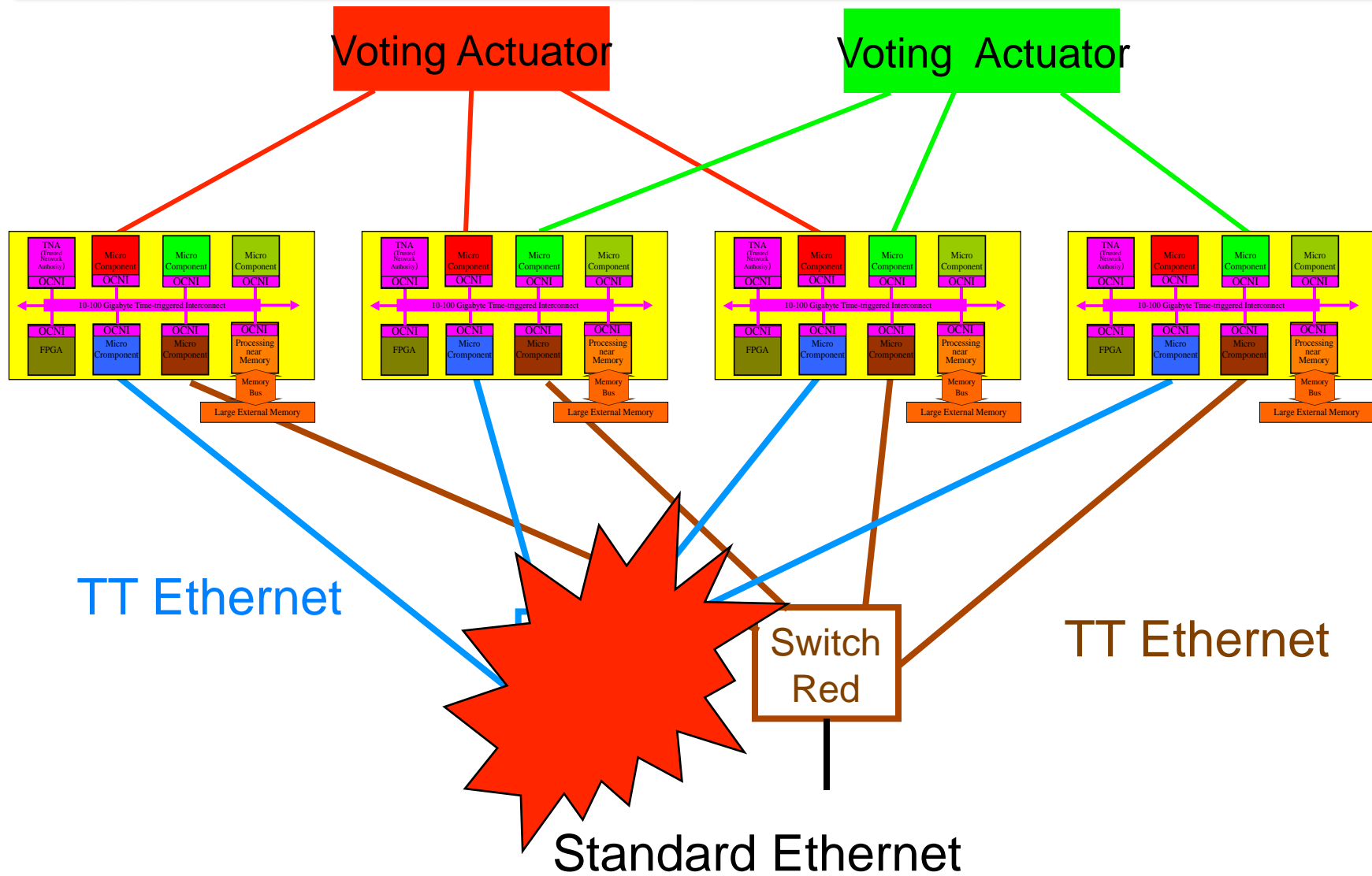
# Off-Chip TMR



# Example: TMR Configuration



# Example: TMR Configuration



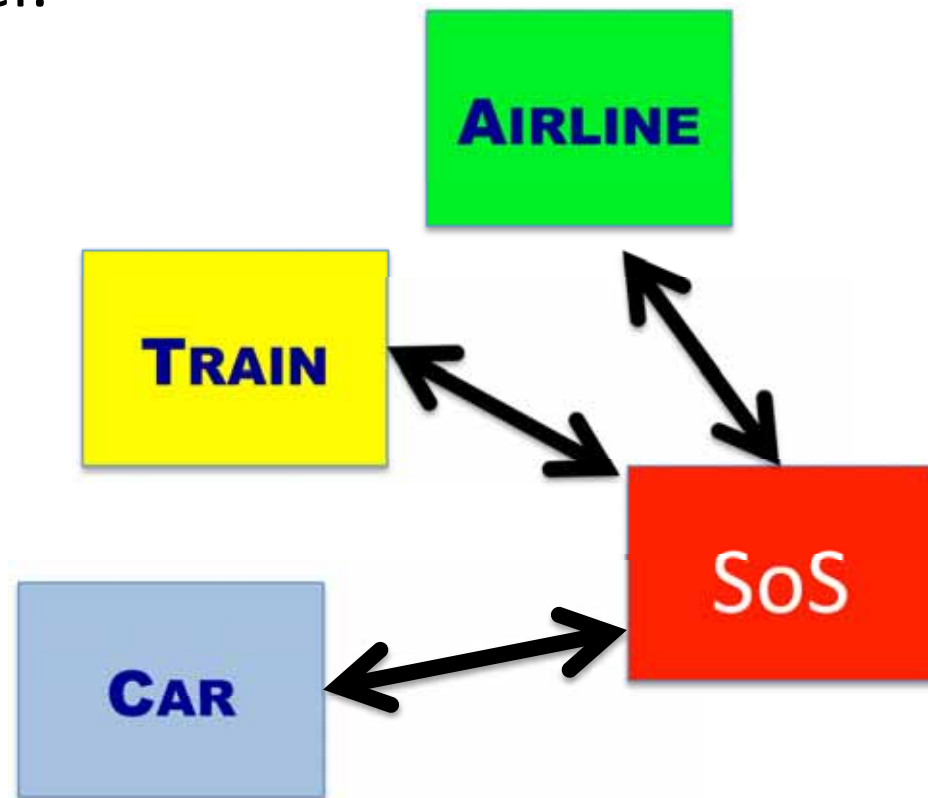
# Outlook: System of Systems

---

In many fields there are *multiple provider systems* for a requested service. A SoS can ***dynamically select*** the most appropriate provider.

## Example: Transportation

- Airline
- Train
- Car





# Conclusion

---

- The increasing dependence of society on the functioning of the ICT systems requires new approaches to dependability.
- System-level dependability techniques can cover transient hardware failures and Heisenbugs in the Software.
- The conceptual simplicity of the TTA supports the implementation of fast recovery actions in cyber-physical systems.

# More Information

---

The GENESYS Architecture is described in a book that can be downloaded freely from the Internet:

[http://www.genesys-platform.eu/genesys\\_book.pdf](http://www.genesys-platform.eu/genesys_book.pdf)



Background information can be found in the second revised edition of my book

***Real-Time Systems—Design Principles for Distributed Embedded Applications***

published by ***Springer Verlag*** on April 27 , 2011.