TU Wien

GENESYS– A Cross-Domain Architecture for Dependable Embedded Systems

H.Kopetz

April 2011

Outline

- Introduction: ARTEMIS
- Architecture Overview
- Model of a *Component*
- Message Communication
- Fault Mitigation
- Conclusion

ARTEMIS

In order to meet the future challenges of the Embedded System industry, such as

- Economies of Scale of the Semiconductor Industry
- Convergence of Application Domains (Internet of Things)
- Dependability (Safety, Security, Reliability, Diagnosis)
- Market Fragmentation

the *European ES Industry* in cooperation with the EU and the national authorities have formed the European technology platform ARTEMIS with the intent to improve the world-wide competitiveness of the European ES industry by developing ***a cross-domain embedded system architecture***.

ARTEMIS Requirements

In a two year effort, the following requirements have been identified for the cross-domain embedded system architecture by an ARTEMIS expert group:

- ***Composability***
- ***Networking and Security***
- ***Robustness***
- ***Diagnosis and Maintenance***
- ***Integrated Resource Management***
- ***Evolvability***
- ***Self Organization***

Detailed requirements on the ARTEMIS website

https://www.artemisia-association.org/downloads/RAPPORT_RDA.pdf

The European FP 7 Project *GENESYS*

GENESYS was a joint research project that has developed an architectural framework for the design and implementation of cross-domain embedded systems that meets the ARTEMIS requirements.

Project duration: Jan 2008 - June 2009

The GENESYS Partners (23)

1 (Coordinator) Vienna University of Technology Austria

2 STM Microelectronics Italy

3 Commissariat à l'Énergie Atomique France

4 Nokia Oyj Finland

5 Thalesgroup France

6 Embedded Systems Institute Netherlands

7 Interuniversitair Micro-Elektronica Centrum

IMEC Belgium

8 Technical University Darmstadt Germany

9 European Software Institute ESI-Tecnalia Spain

10 Technical Research Centre of VTT Finland

11 Infineon Germany

12 Centro Ricerche Fiat Italy

13 TTTech Computertechnik AG Austria

14 University of Bologna Italy

15 Verimag UJF France

16 Fraunhofer IGD FhG Germany

17 TU München Germany

18 Vytautas Magnus University Kaunas Lithuania

19 Ikerlan Spain

20 Budapest University of Technology and

Economics Hungary

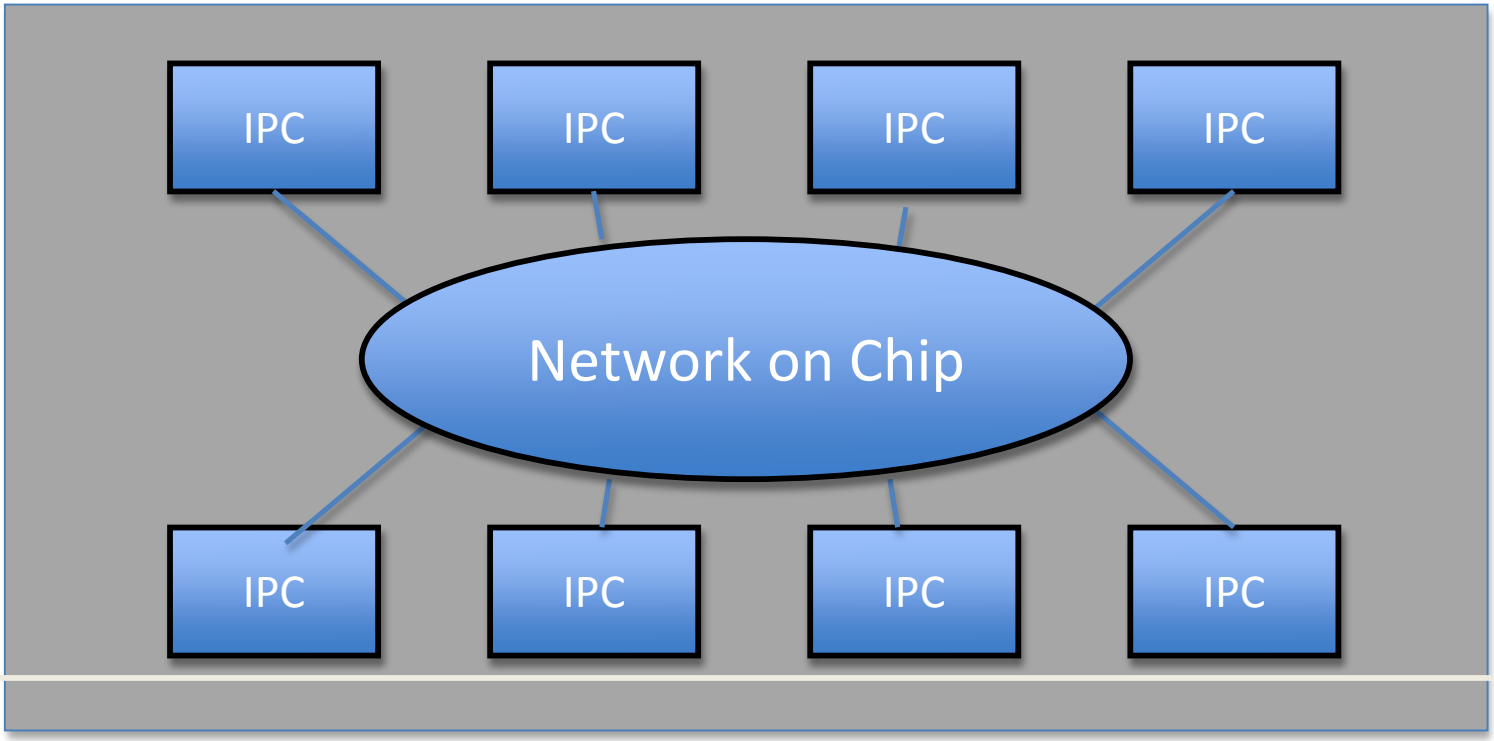
21 Universidad Politecnica de Madrid Spain

22 NXP Semiconductors Netherlands

23 Volvo Technology Sweden

Hardware and OS Trends

The future hardware building block for embedded systems will be a System-on-Chip, where multiple *heterogeneous* IP-Cores (IPC) are interconnected by a Network-on-Chip. Such an SoC is often called an *MPSoC*.



Why the Move to *MPSoCs*

Some reasons for moving to MPSoCs:

- ***Islands of Synchronicity*** are needed since *global time distribution* becomes very costly as we move to high frequencies.
- ***Power and Energy Concerns***: Small physical structures are more energy efficient and can be turned off (power gating) if not needed.
- ***Fault Containment***: A large SoC must be partitioned into independent fault containment regions that can be restarted dynamically.
- ***Design Reuse***: Design, Testing and Certification must move to a higher level of abstraction.

Architecture Alternatives

There are two main alternatives to organize an MPSoc:

- ***Shared Memory (SM)*:** Each IP-core has one or more local caches. There exist a global memory that can be accessed by all IP Cores. Cache coherence is performed at the hardware level.
- ***Message Passing (MP))*:** Each IP-core has a local scratchpad memory. IP cores communicate by messages only. There is no global shared *passive* memory.

Comparison of *SM* and *MP*

Characteristic	Shared Memory	Message Passing
Flexibility	***	*
Cognitive Simplicity	*	***
Energy Efficiency	*	***
Real-Time Guarantees	*	***
Fault Containment	*	***
Legacy Integration	***	*
Scalability	*	***
Certification	*	***

Architectural Principles of *GENESYS*

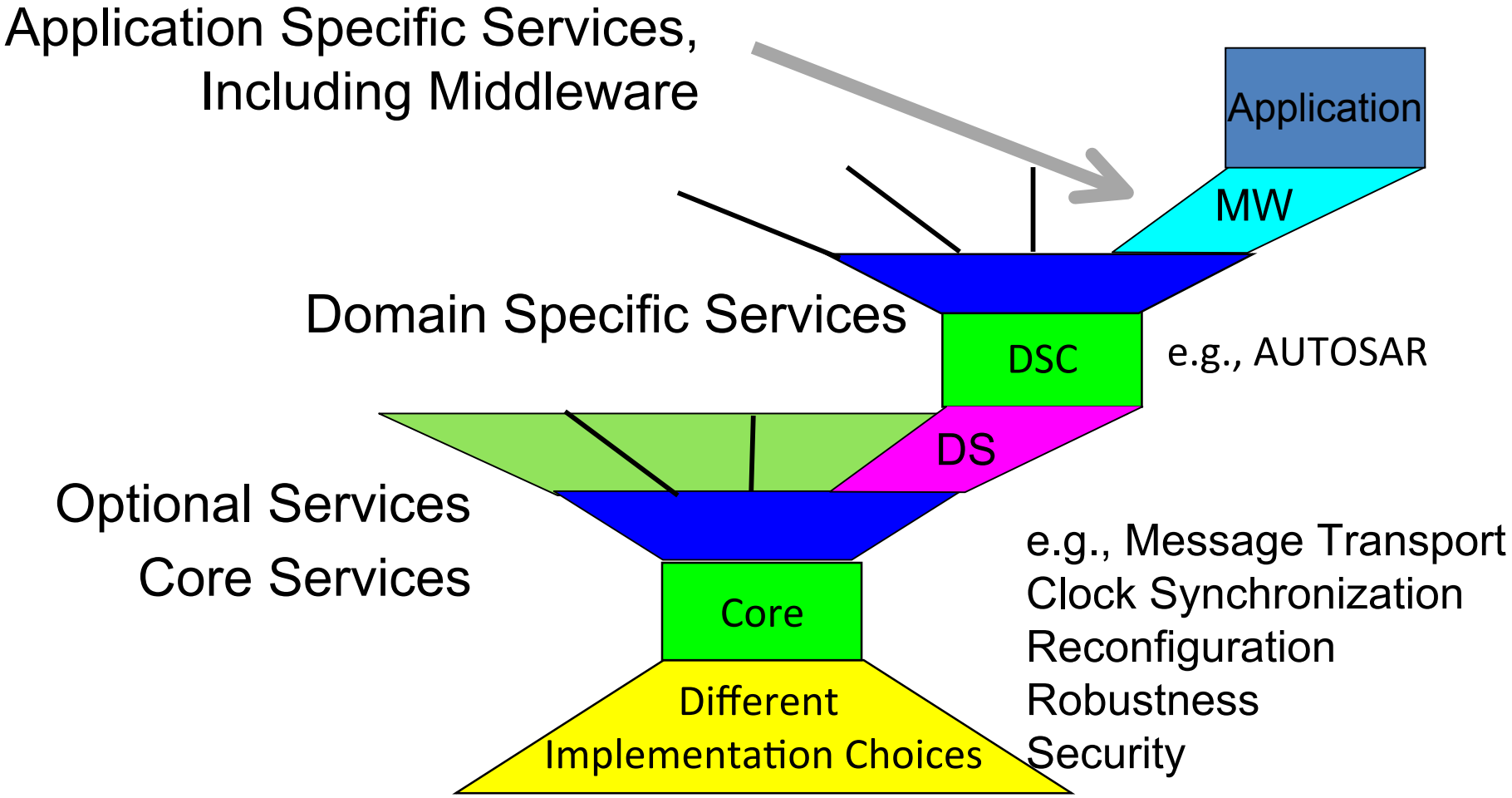
- **Component Orientation**—A component is a hardware/software unit
- **All components are *time-aware***—A global time is provided by the architecture.
- **Separation of Computation from Communication**—Components and Communication systems can be developed independently.
- **Core Services are Deterministic**—Modular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

Complexity Management in GENESYS

The architectural style of GENESYS 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

Architecture Services Overview

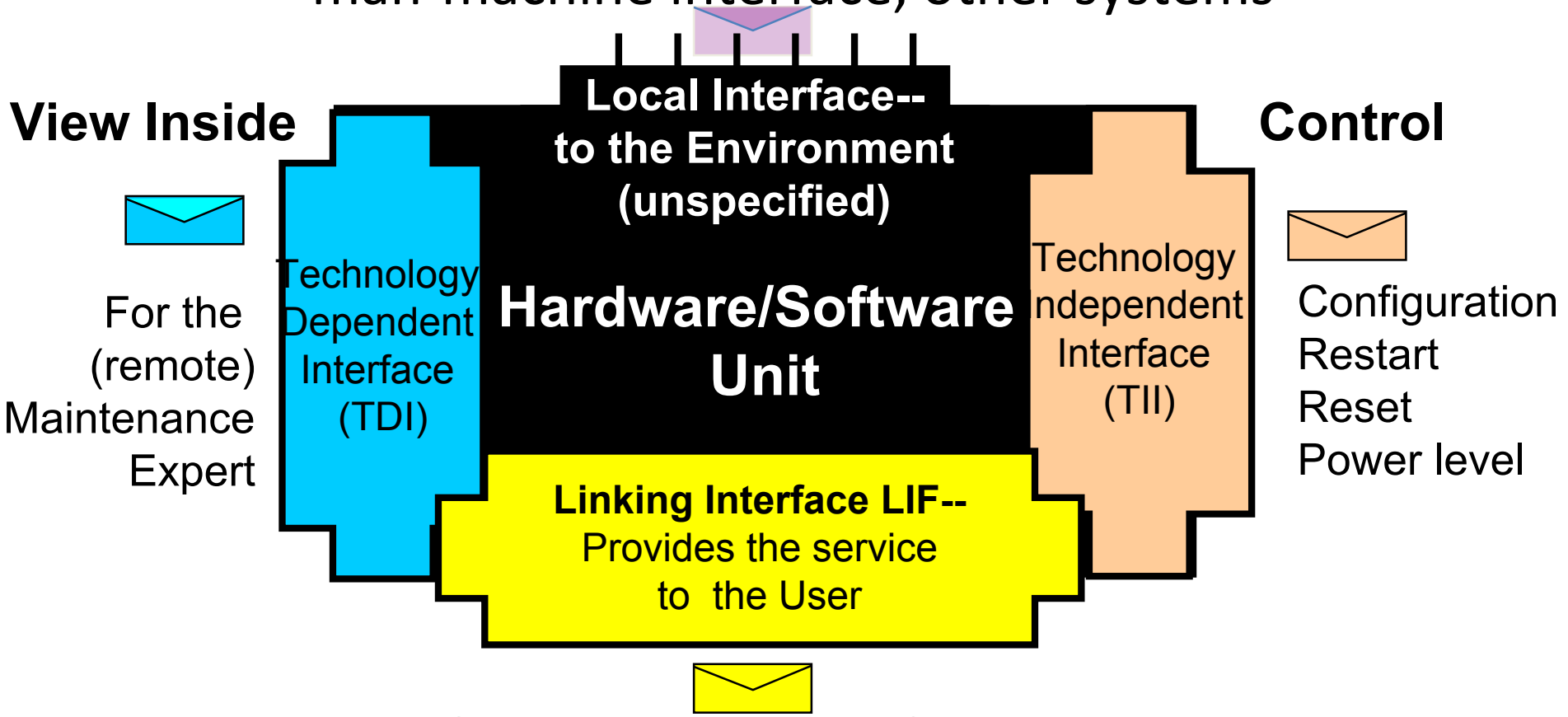


What is a *GENESYS* Component?

- ***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.

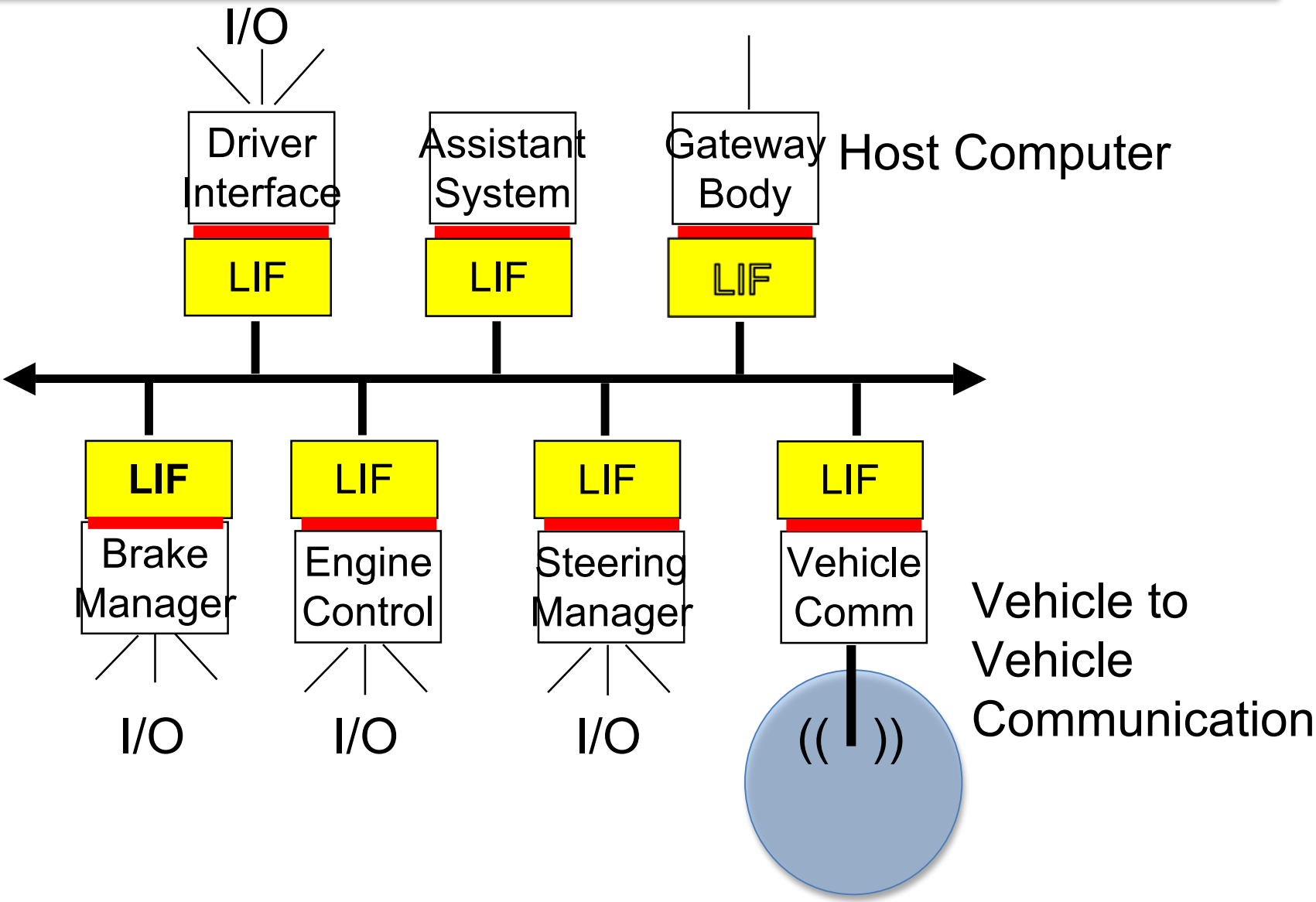
The Interfaces of a *GENESYS* Component

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

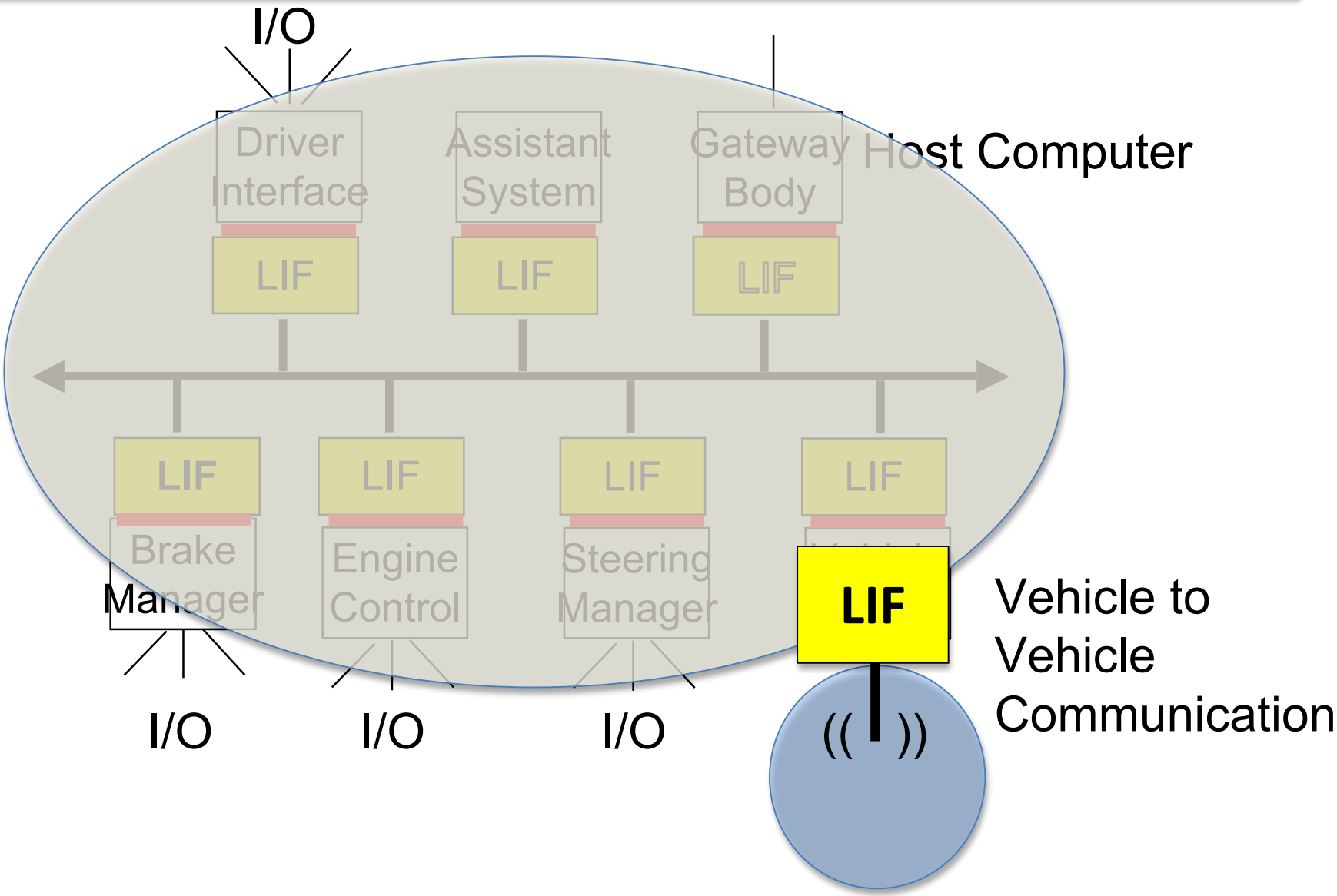


Relevant for the integration of components into a cluster of components

Example of a Cluster



Example of a Cluster--*Recursion*



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.

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.*

Gateway Components

Gateway components connect the *cyber world* to the *physical world*:

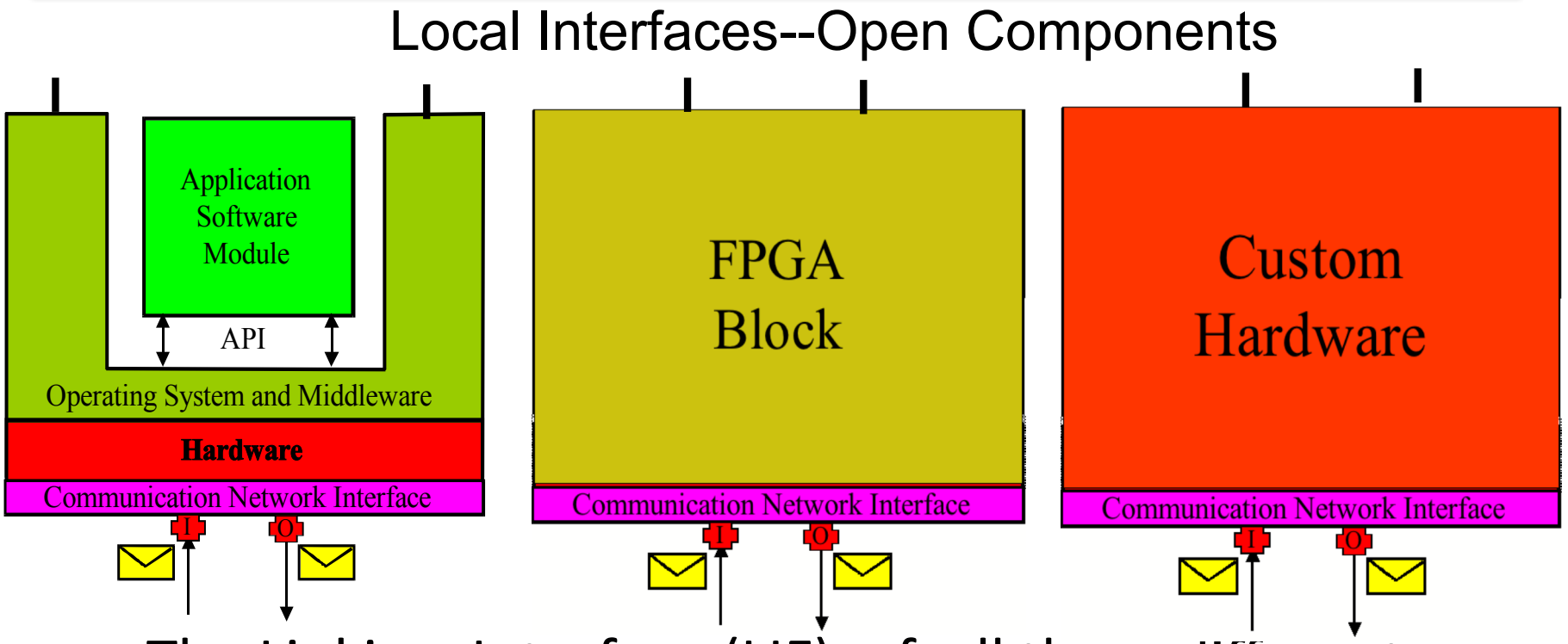
- The representation of the information in the two worlds will be different, but the ***semantic content*** of the *message variables* must be the same.
- The *meta-level specification* of a gateway component requires knowledge about the context of use.

Gateway Components

Gateway components can also 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.

Openness: Soft versus *Hard* Components

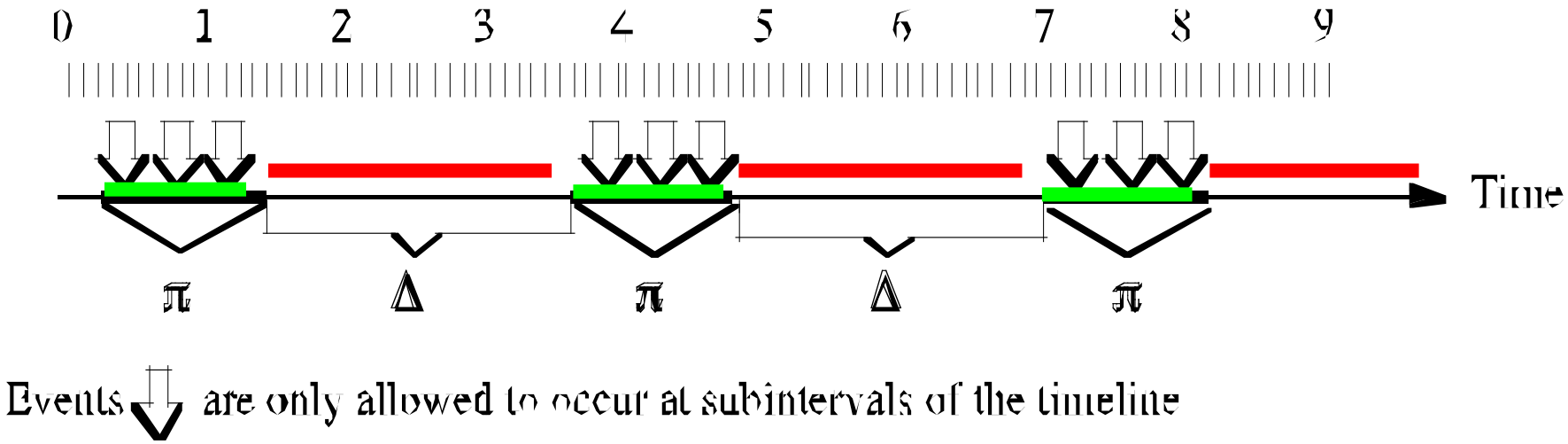


The Linking Interface (LIF) of all three different component implementations should have the same *syntax, timing* and *semantics*. For a user, it should not be discernible which type of component is behind the LIF (***Technology Agnosticism***).

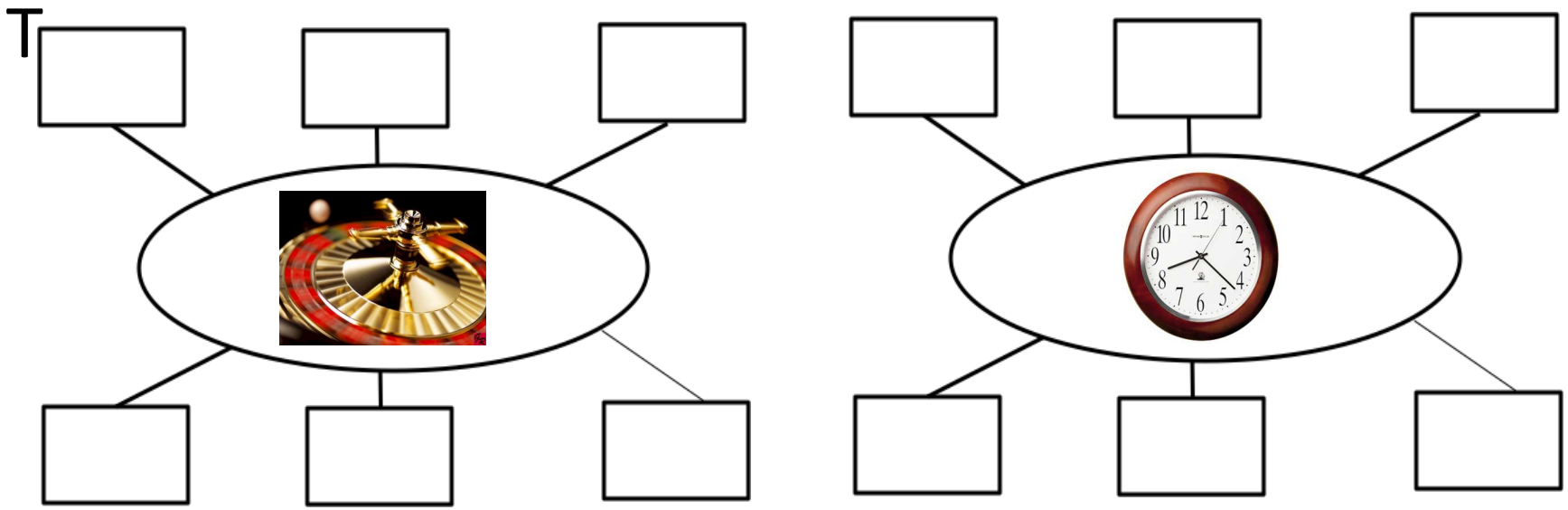
Components are Time Aware: *Sparse Time*

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 π with an interval of silence of duration Δ between any two active intervals, then we call the time base π/Δ -sparse, or ***sparse*** for short, and events that occur during the active intervals ***sparse events***.



Component Integration Framework



Competition versus **Cooperation**

Best-Effort **Time Triggered**

Probabilistic **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 pones*)
  - 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
  - replication of state at multiple components
  - Triple Modular Redundancy

Message Types

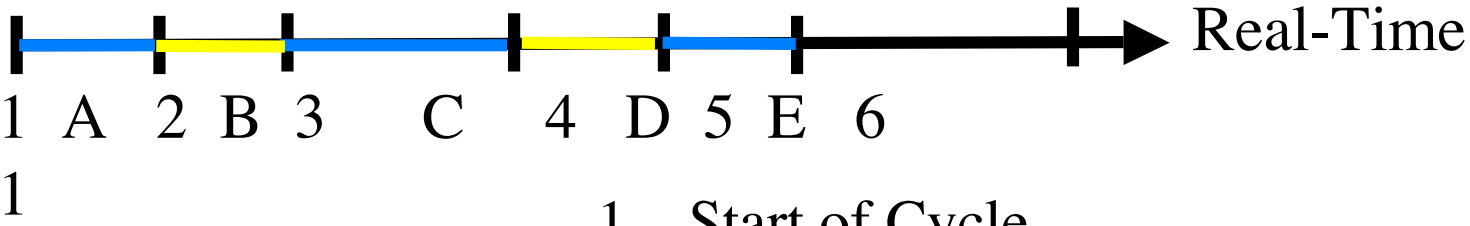
- ***Periodic Messages (TT) (core)***
  - No queues, non-consuming read, update in place
  - Temporal guarantees
- ***Sporadic Messages (ET) (core)***
  - characterized by two queues, one at the sender site and one at the receiver site
  - Exactly once semantics
  - Normally best effort timing
- ***Real-time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

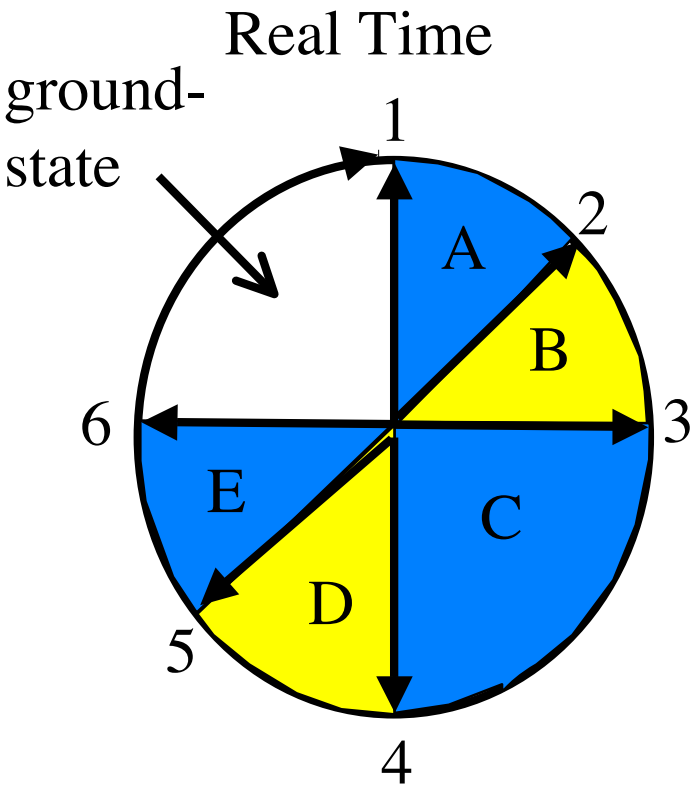
***Openness*: Any communication protocol (wire-bound or wireless) that provides these services can be used**

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.

TT Systems are *Cyclic*



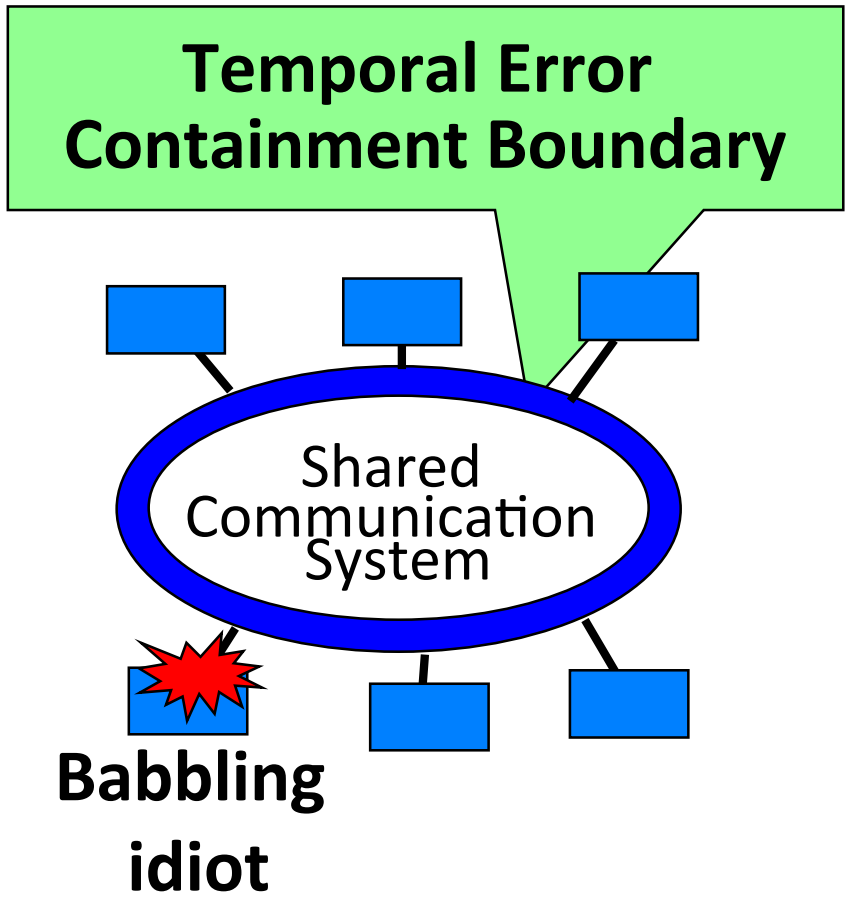


1 Start of Cycle
A Observation of Sensor Input
2 Start of Transmission of Sensor Data
B Transmission of Input Data
3 Start of Processing of Control Algorithm
C Processing of Control Algorithm
4 Termination of Processing
D Transmission of Output Data
5 Start of Output to Actuators
E Output Operation at the Actuator
6 Termination of Output Operation

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* Communication System is a self-contained FCU that has *temporal information* about the allowed behavior of the nodes-- it ***must contain application-specific state***.



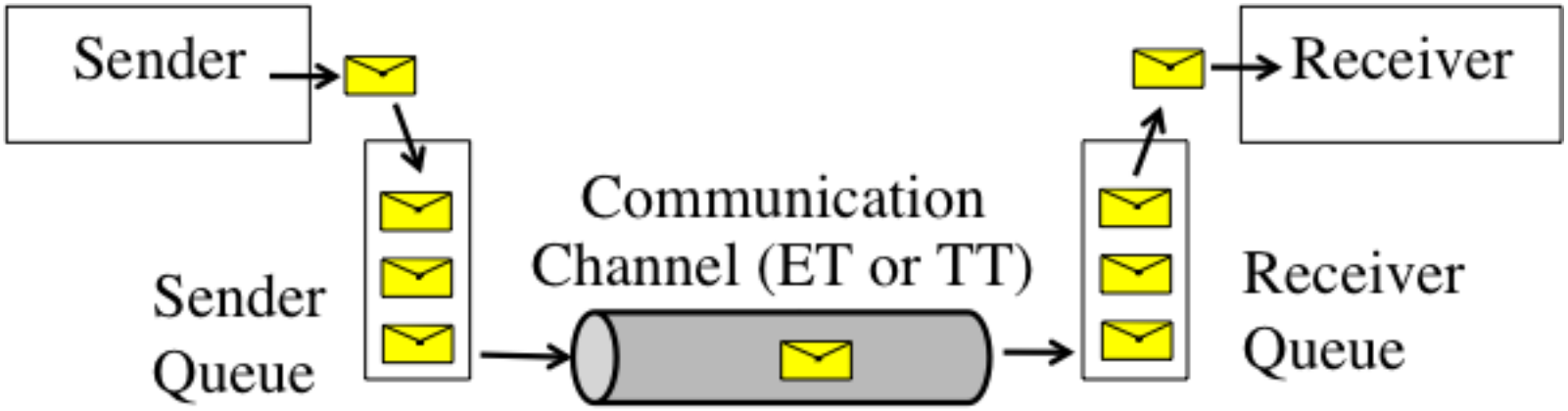
Time-Triggered Protocols

- TTP is very data efficient and operates up to a bandwidth of 20 Mbits/second.
- TTE is fully compatible with standard Ethernet and operates up to 1 Gigabit/second. It has been selected as the standard protocol for the ORION spacecraft.
- TT NoC is a time-triggered network on chip for the connection of the IP-cores on an MPSoc.

Time-Triggered Ethernet

- Supports two message times, standard (ET) Ethernet messages and TT Ethernet messages.
- Message Format and Addressing adhere to the Ethernet Standard.
- Standard Ethernet controller in end systems
- TT messages are transported with small constant delay.
- Schedule for TT messages contained in the TTE switch to protect the network from babbling idiots.

Event-Triggered Communication



In the GENESYS Network-on-Chip, the communication channel is time-triggered, even for *event-triggered* messages. There are no intermediate buffers in the network—they are at the memory spaces of the end points.

Integration: Principles of Composability

(1) **Independent Development of the Components** (Architecture)
The interfaces of the components must be *precisely specified* in the value domain and in the *temporal domain* in order that the component systems can be developed in isolation.

(2) **Stability of Prior Services** (Component Implementation)
The prior services of the components must be maintained after the integration and should not fail if a partner fails.

(3) **Non-Interfering Interactions** (Communication System)
The communication system transporting the messages must meet the given temporal requirements under all specified operating conditions.

(4) **Preservation of the Component Abstraction in the case of failures** (Architecture) and provision of a communication system with error containment.

Operating System in GENESYS

In GENESYS 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

Functions of the Mini OS 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)

Integration Levels

In Genesys we introduce three integration levels:

- **Chip Level:** the components are IP-cores, interconnected by a NoC (network on Chip) to form a Chip
- **Device Level**: the component are chips interconnected by an inter-chip communication system to form a Device. A device can be an addressable entity in the Internet and can have an IP-Address (as well as a chip, if desired).
- **System Level**: The components are devices that are interconnected by a wire-bound or wireless communication service:
  - *Closed Systems*: System structure is *static*.
  - *Open Systems*: System structure is *dynamic*, i.e., devices can come and go

Chip Level: Why not take the *Cell*?

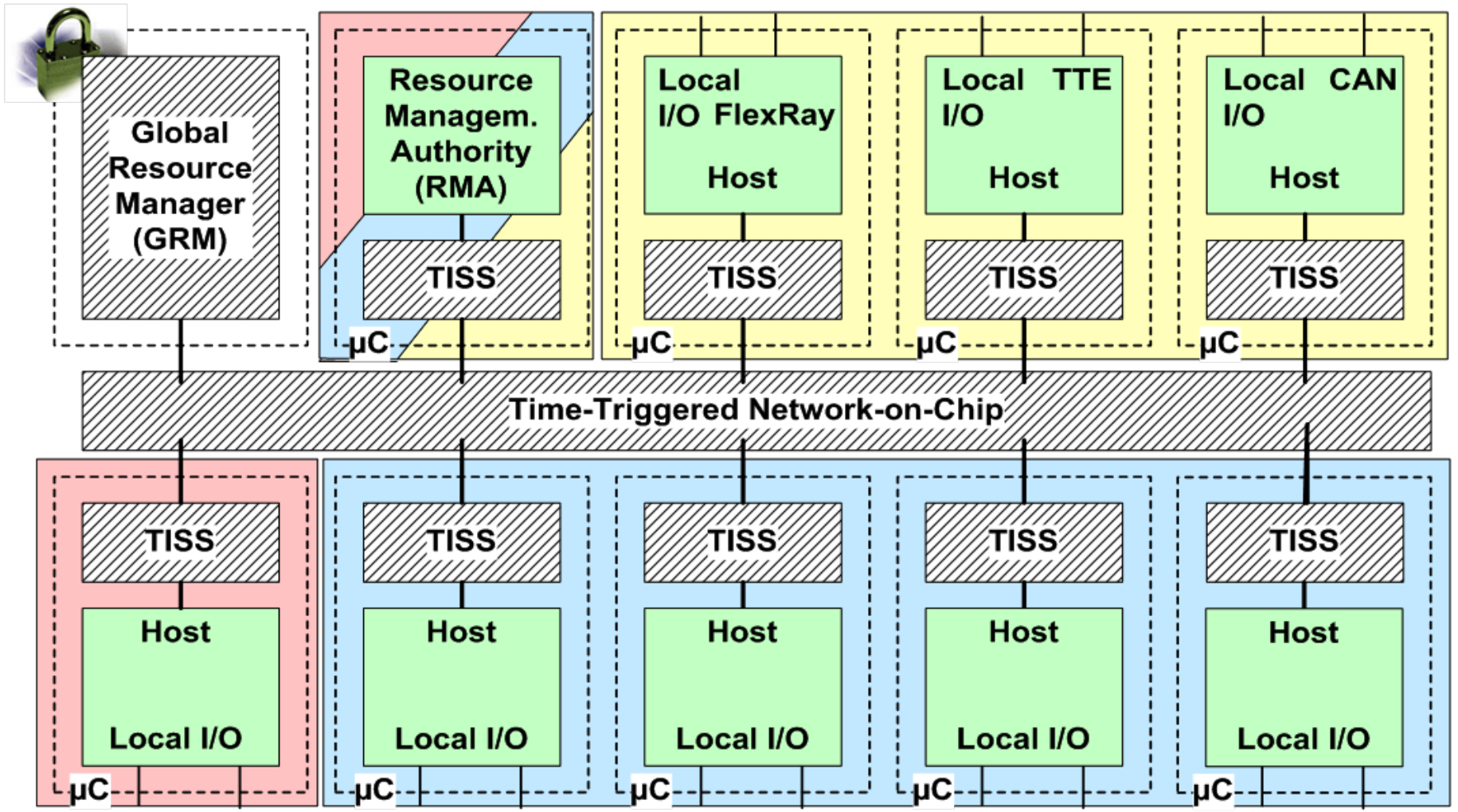
Cell Processor	GENESYS
Designed to run a *single monolithic application*	Designed to provide an execution execution environment for *nearly independent jobs*.
Decomposition of the application into parallel modules is the critical issue.	Fault Isolation and Error Containment between the jobs executing on different cores are the critical issues
Asynchronous on-chip communication	Time-triggered deterministic on-chip Communication
Single Distributed Operating System	Multiple heterogeneous Operating Systems (one in each core)

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 (harware or software) of a non-trusted IP core

Chip Level: GENESYS Prototype



Examples of *Optional OS Services*

Optional OS Service are implemented by self-contained System Components in cooperation with the GEM (Generic Middleware) within a component:

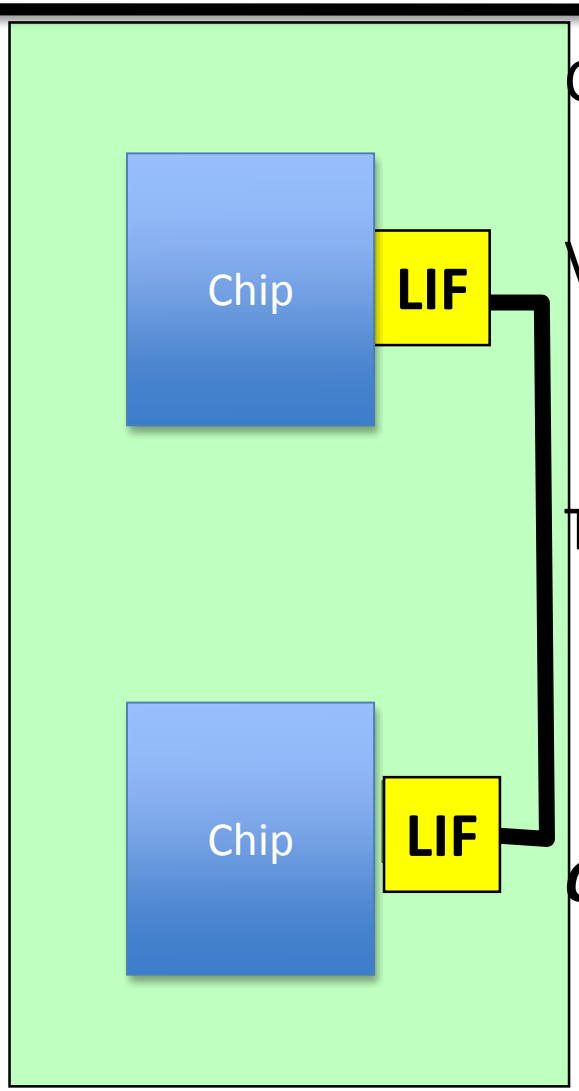
- External Memory Management
- Security
- Diagnostics and Robustness
- Integrated Resource Management (scheduling)
- Standard Internet Connection

Core Services at the Chip Level

The core services are provided by the trusted subsystem at the chip level

- ***Platform Configuration Service***—boot service to generate components by binding software (job) with IP core hardware
- ***Channel Configuration Service***—establishes the ports and channels among IP cores
- ***Clock Synchronization Services***—global time
- ***Execution Service***—control the execution of IP Cores (start, terminate, reset)
- ***Communication Service—***provides the capability to send and receive messages.

Device-Level: *Integration of Chips*



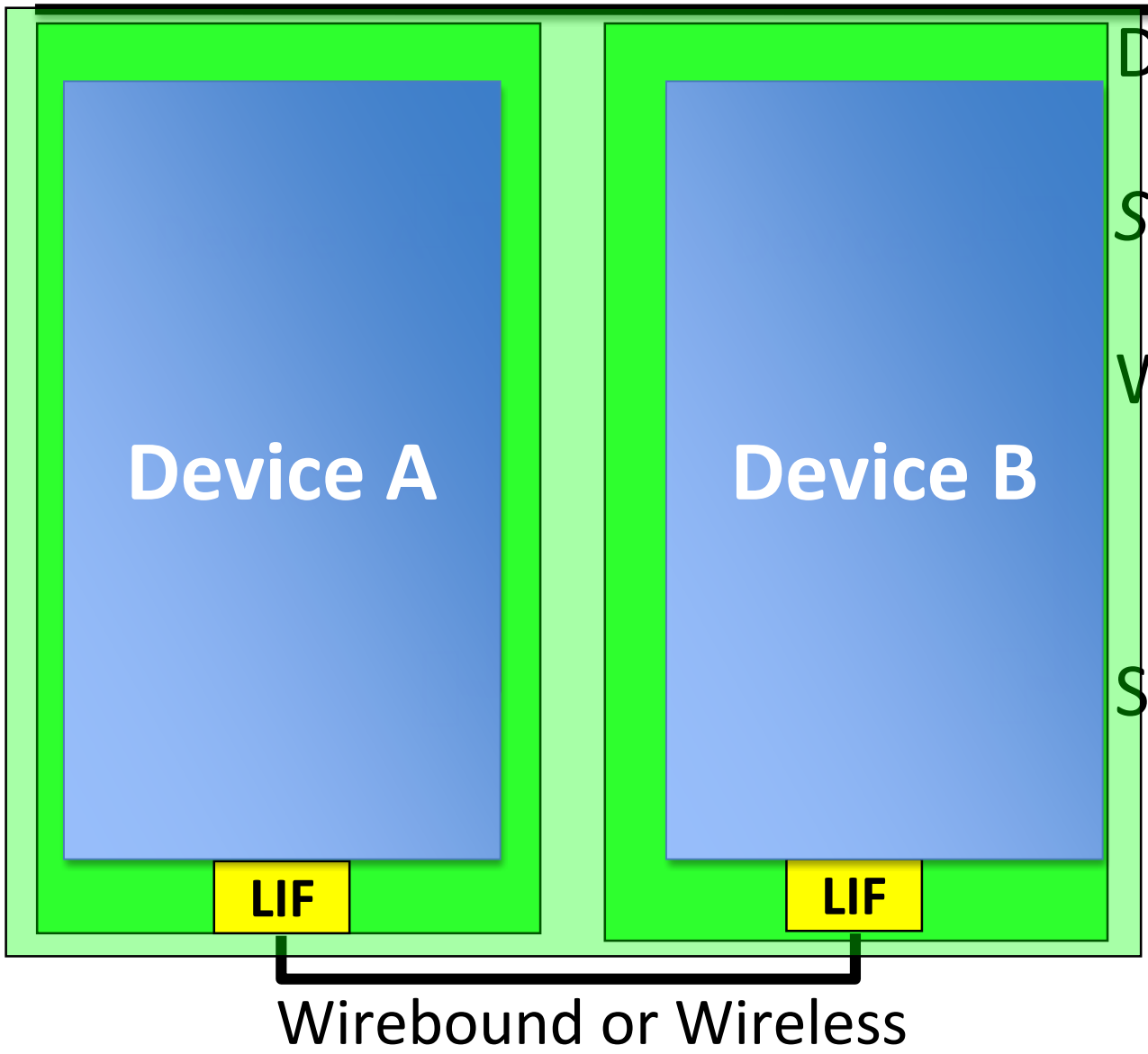
Chips are linked by ***intra-device-level LIFs*** to form a device.

Viewed from the *intra-chip level*, the *intra-device level LIF* is a *local interface (and vice versa)*.

The intra-device level LIF carries its own LIF Specification that comprises all subsystems that are connected to this device.

***Openness*: The open LIF specification makes it possible to integrate legacy systems.**

System Level: *Integration of Devices*



Devices are linked by *System Level LIFs*

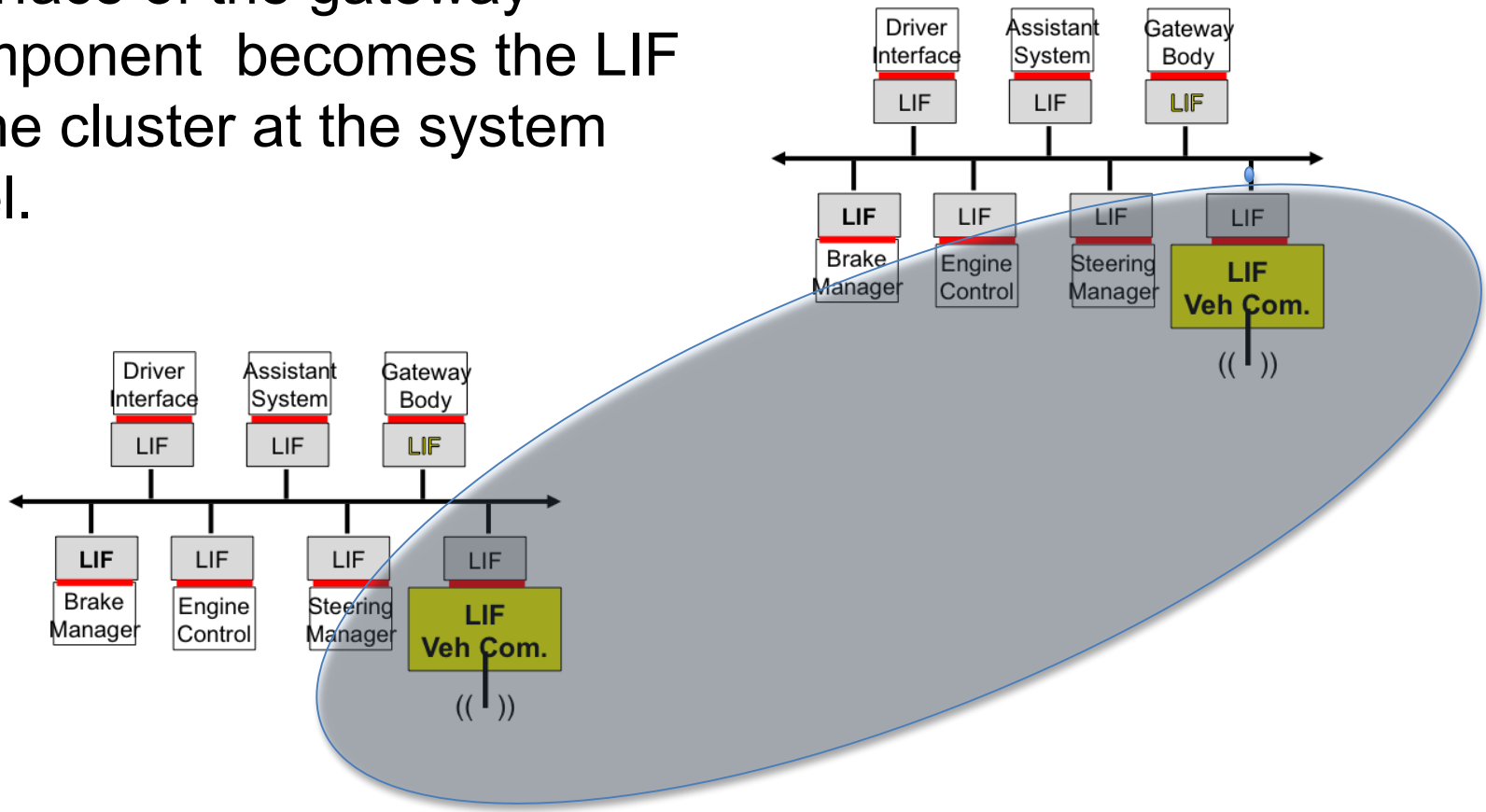
We distinguish between

- Open
- Closed

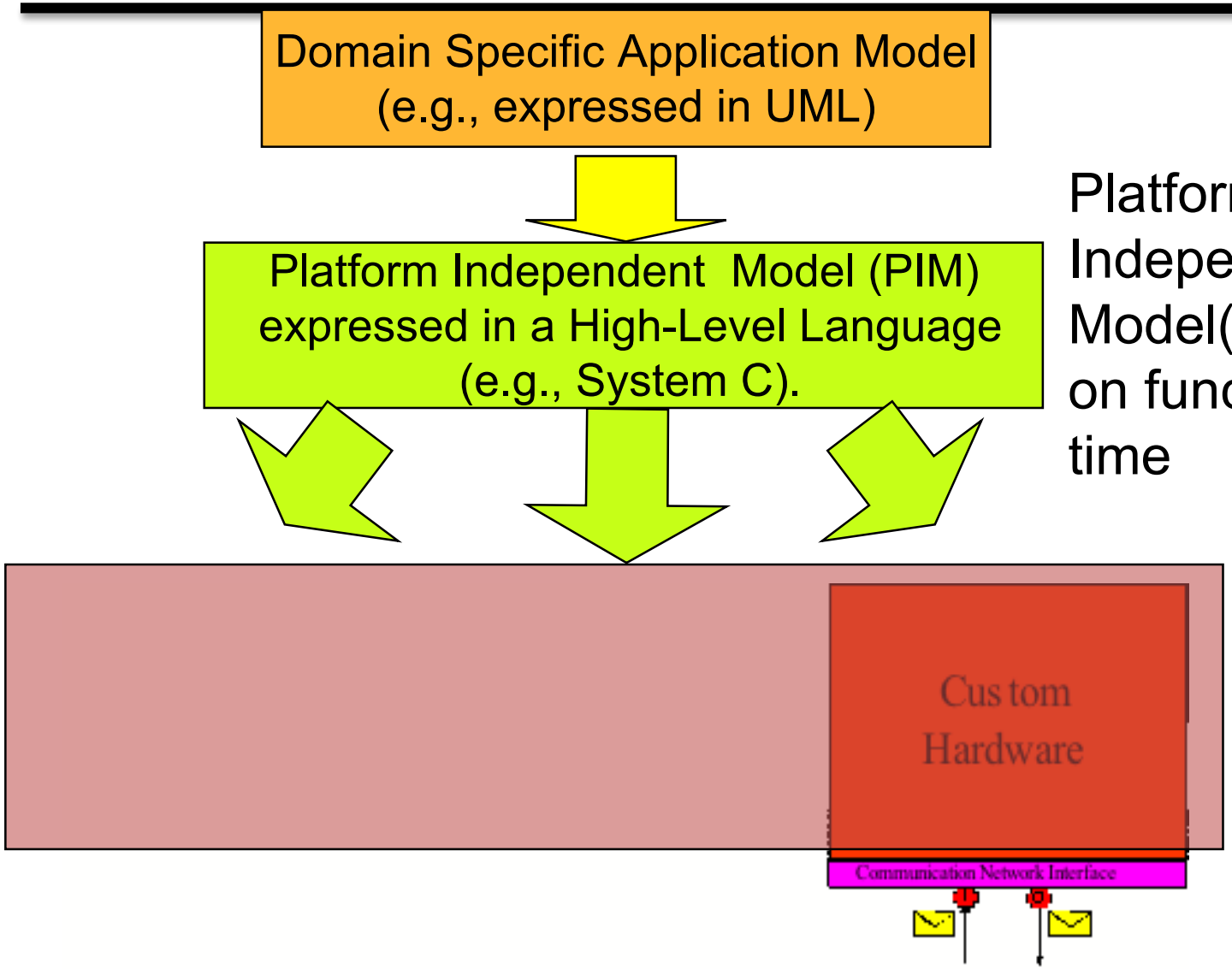
Systems.

Example: *Car to Car* Communication

The Vehicle Communication Interface of the gateway Component becomes the LIF of the cluster at the system level.



Abstraction: *Model Driven Design*



Platform Independent Model(PIM) focuses on functionality and time

Platform Specific Model (PSM)

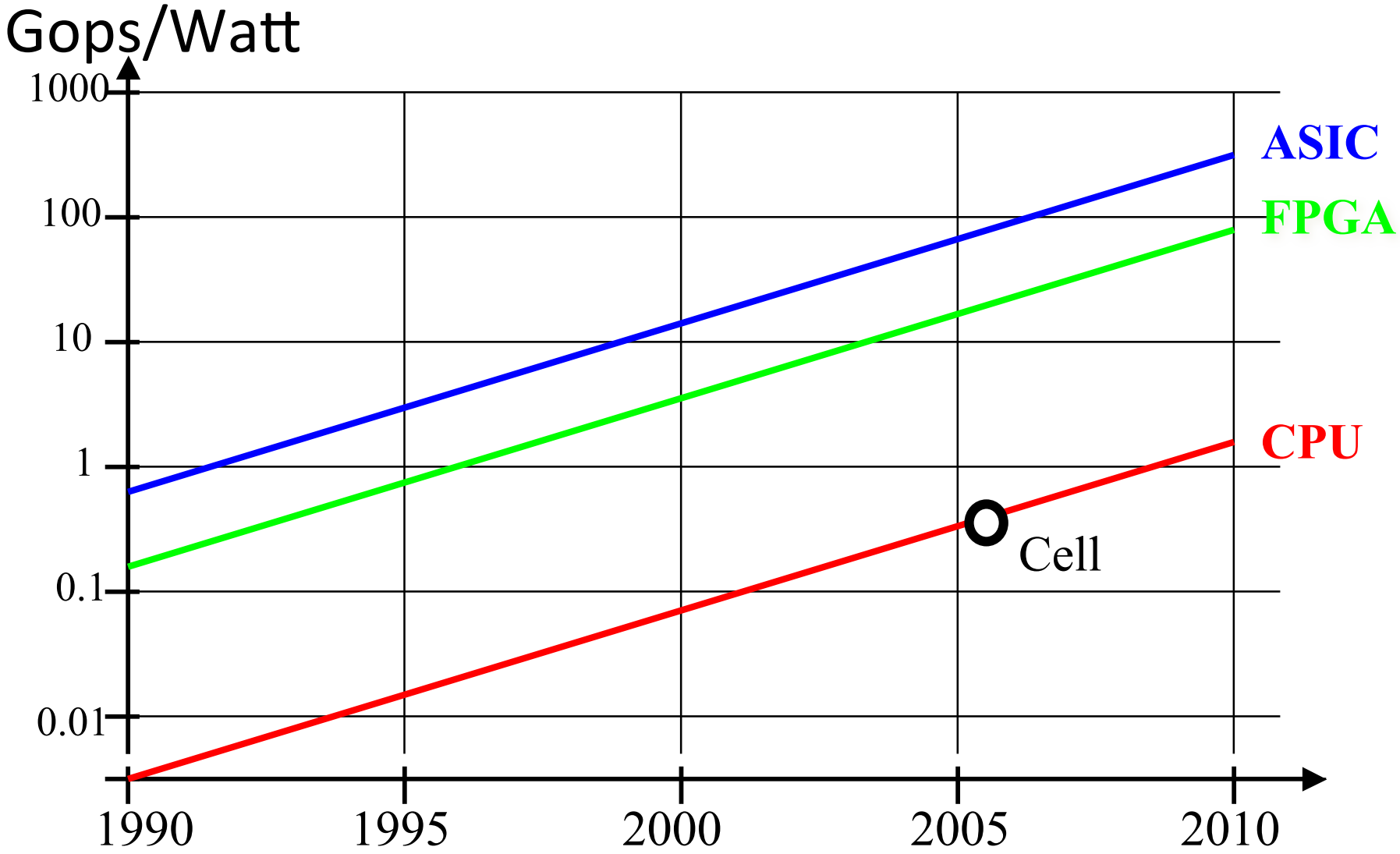
System Components can be *Hard*

An *OS component*, that has reached a high level of stability, can be implemented in hardware:

- Reduction of the power requirement by a factor of one thousand and more
- Reduction of the silicon real-estate
- Standardization by hardware

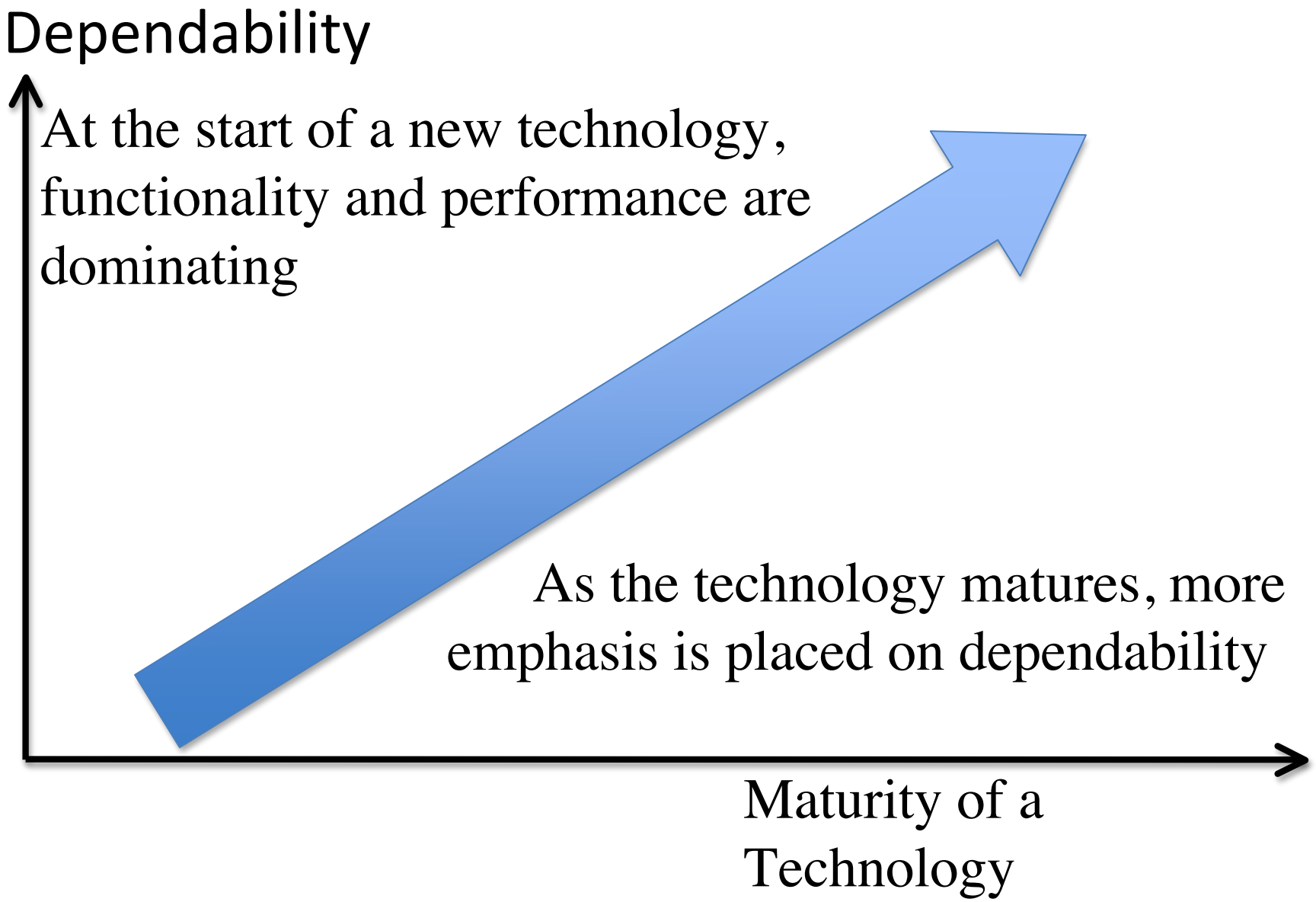
An OS component maps ideally into an IP-core of a multiprocessor-system-on-chip (MPSoC)

Performance Trends--Power



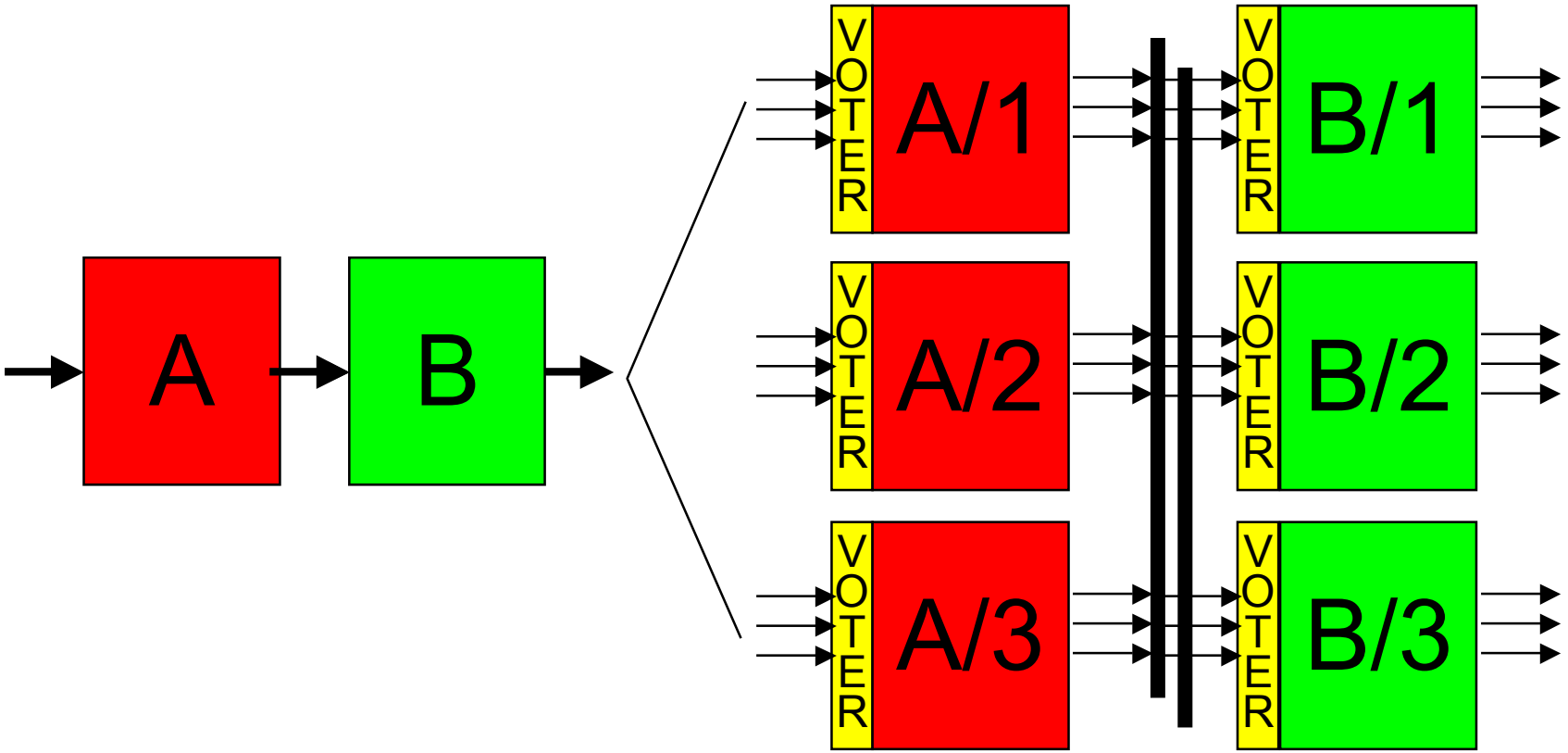
Ref: Lauwereins, Imec, MSOP 2006

Dependability versus *Maturity*



Triple Modular Redundancy (TMR)

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



What is Needed to Implement TMR?

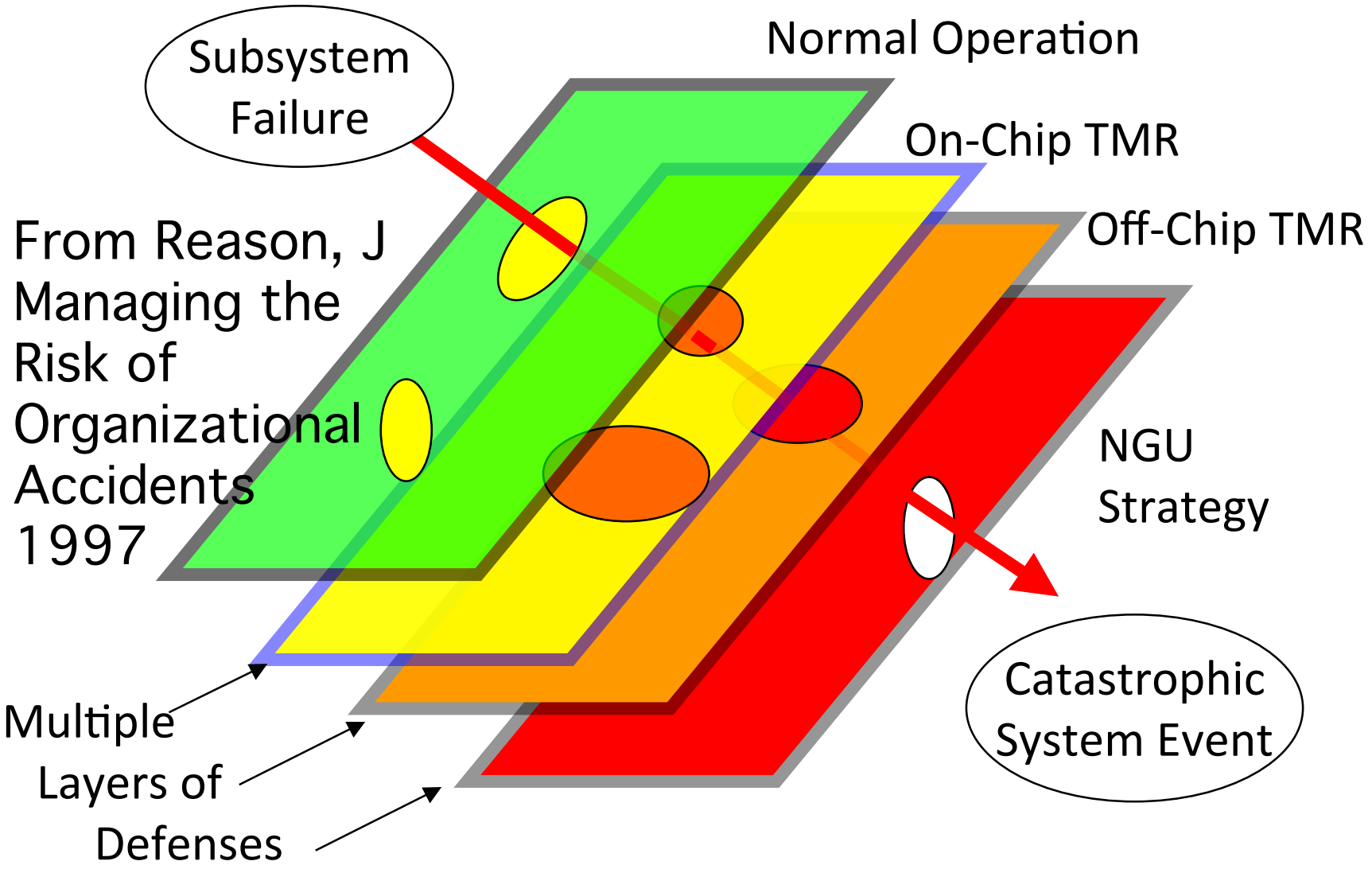
What architectural services are needed to implement Triple Modular Redundancy (TMR) at the architecture level?

- 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

Levels of Fault Mitigation in GENESYS

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.

Approach to Safety: The *Swiss-Cheese* Model

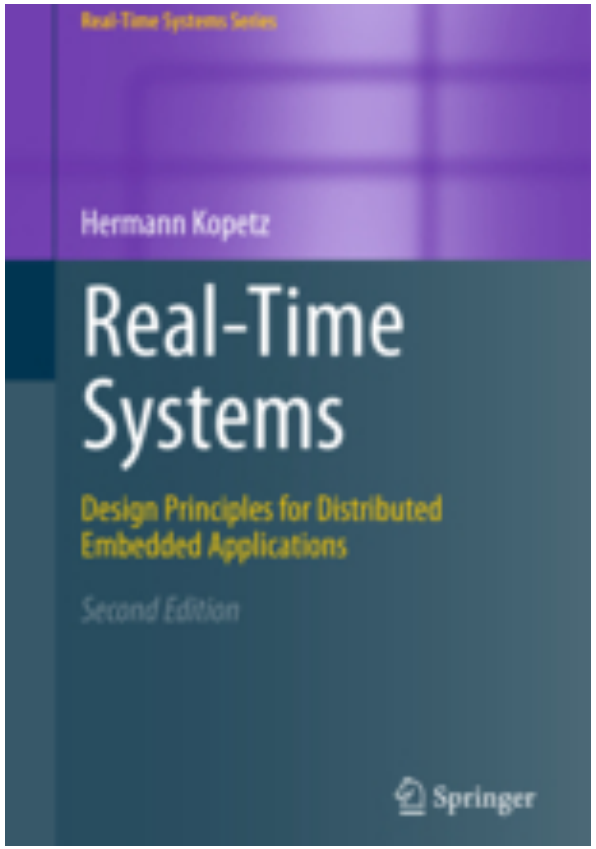


Conclusion

- The GENESYS architecture is a cross-domain architecture that distinguishes between core services and optional services.
- The *certifiable core services* must be available in any instantiation of the architecture and can be implemented in hardware.
- The *optional service* can be used to establish a high-level framework for specific application domains.
- GENESYS does not need a large monolithic operating system.

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



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 (two days ago).