Complexity Management in GENESYS

H. Kopetz

January 2009

GENESYS--Driven by the ARTEMIS Requirements [2]

GENESYS is an FP 7 project that is developing an architectural framework for the design and implementation of cross-domain embedded systems that meet the ARTEMIS requirements.

Project Partners (23): TU Vienna (Coordinator), Nokia, Infineon, Thales, STMicroelectronics, NXP, Fiat, Volvo, TTTech, Ikerlan, IMEC, et al.

Project duration: Jan 2008 - June 2009

Requirements for the ARTEMIS Architecture

In a two year effort, 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 document at the ARTEMIS website

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

What are *Characteristics* of GENESYS?

The following properties are characteristic for the *cross-domain architectural style* of GENESYS:

- Strict Component Orientation
- Openness
- Hierarchy of Services
- Deterministic Core
- Standard Internet Integration

Complexity Management

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

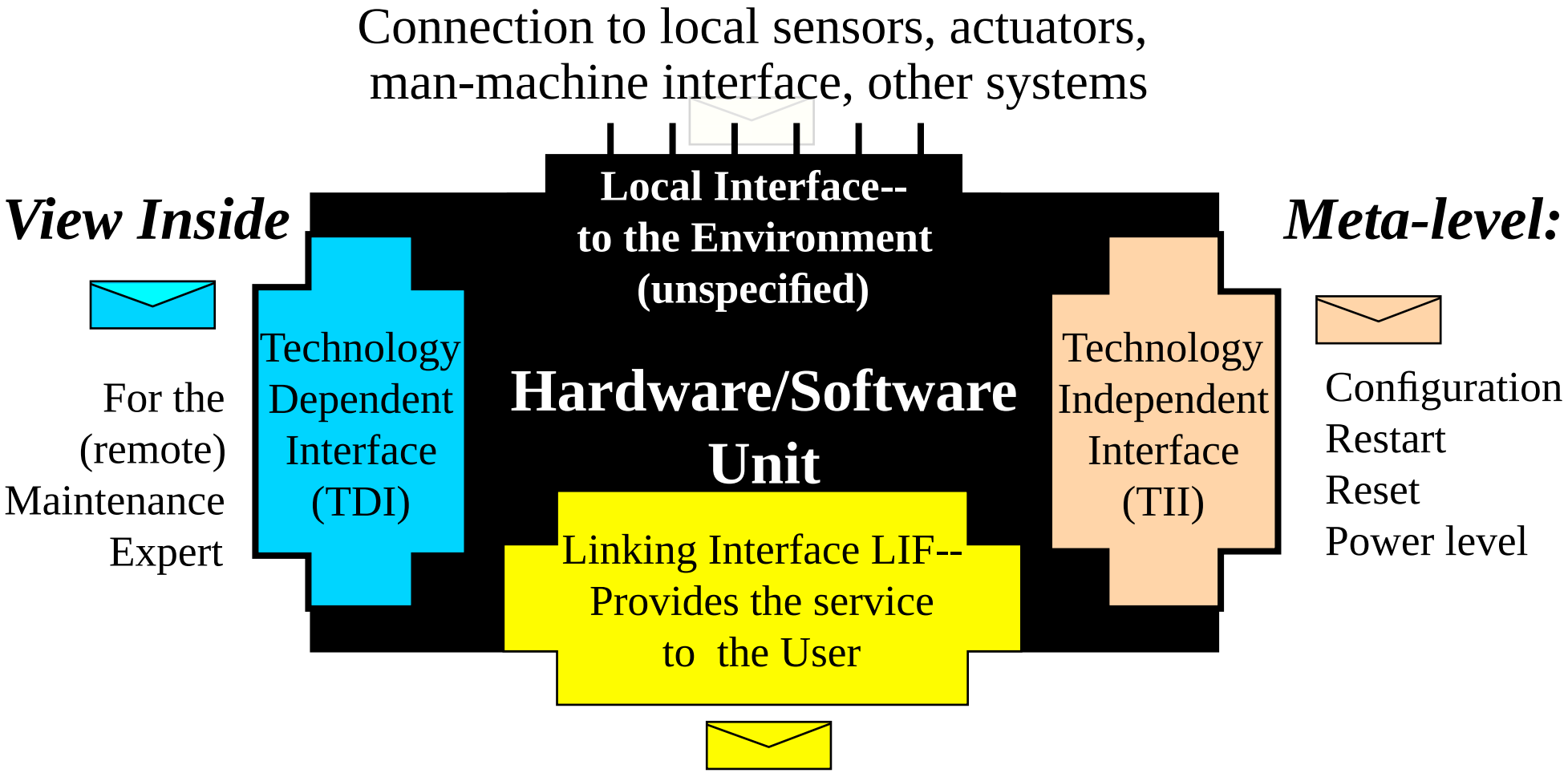
- ***Partitioning***: The partitioning of a system into nearly autonomous subsystems (components).--*Physical Structure*
- ***Abstraction***: The introduction of abstraction layers whereby only the relevant properties of a lower layer are exposed to the upper layer--*Structure and Behavior*
- ***Segmentation***: The *temporal decomposition* of complex behavior into small parts that can be processed sequentially ("step-by-step")--*determinism* helps

What is a *GENESYS* Component?

A GENESYS *component* is a

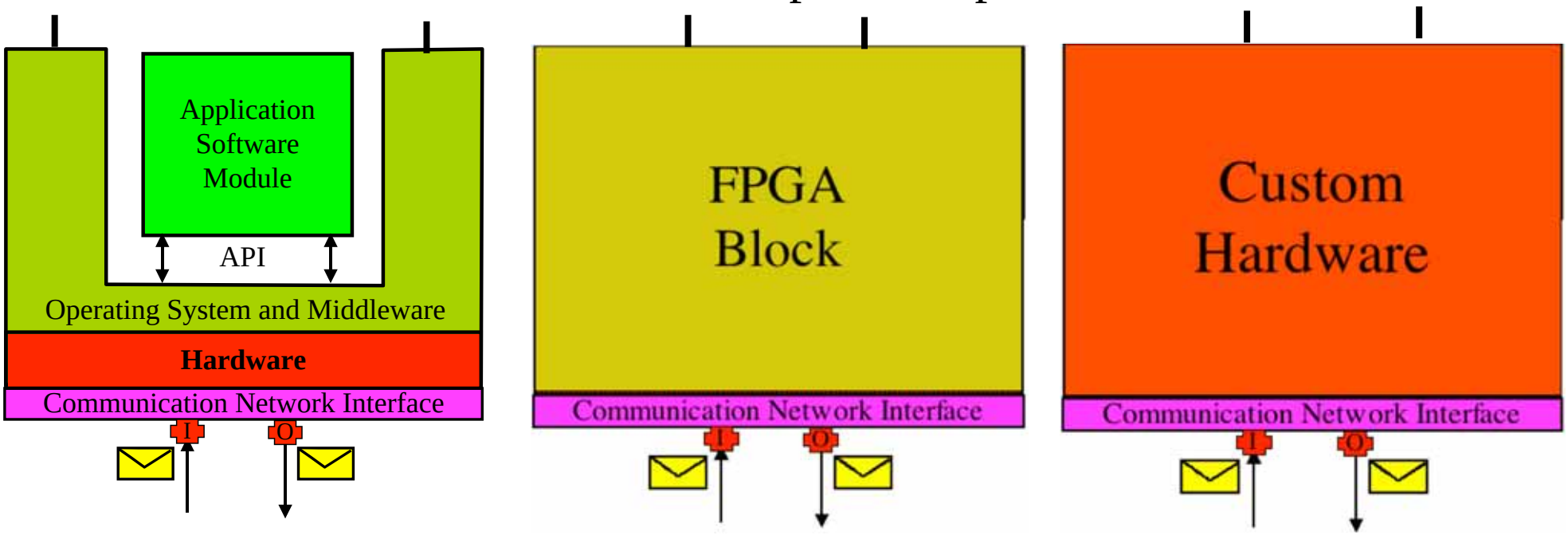
- ***Hardware/software unit*** that accepts input messages, provides a useful service, maintains internal state, and produces after some *elapsed time* output messages containing the results. It is aware of the progression of *physical time*.
- ***Unit of abstraction***, the behavior of which is captured 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 (i)



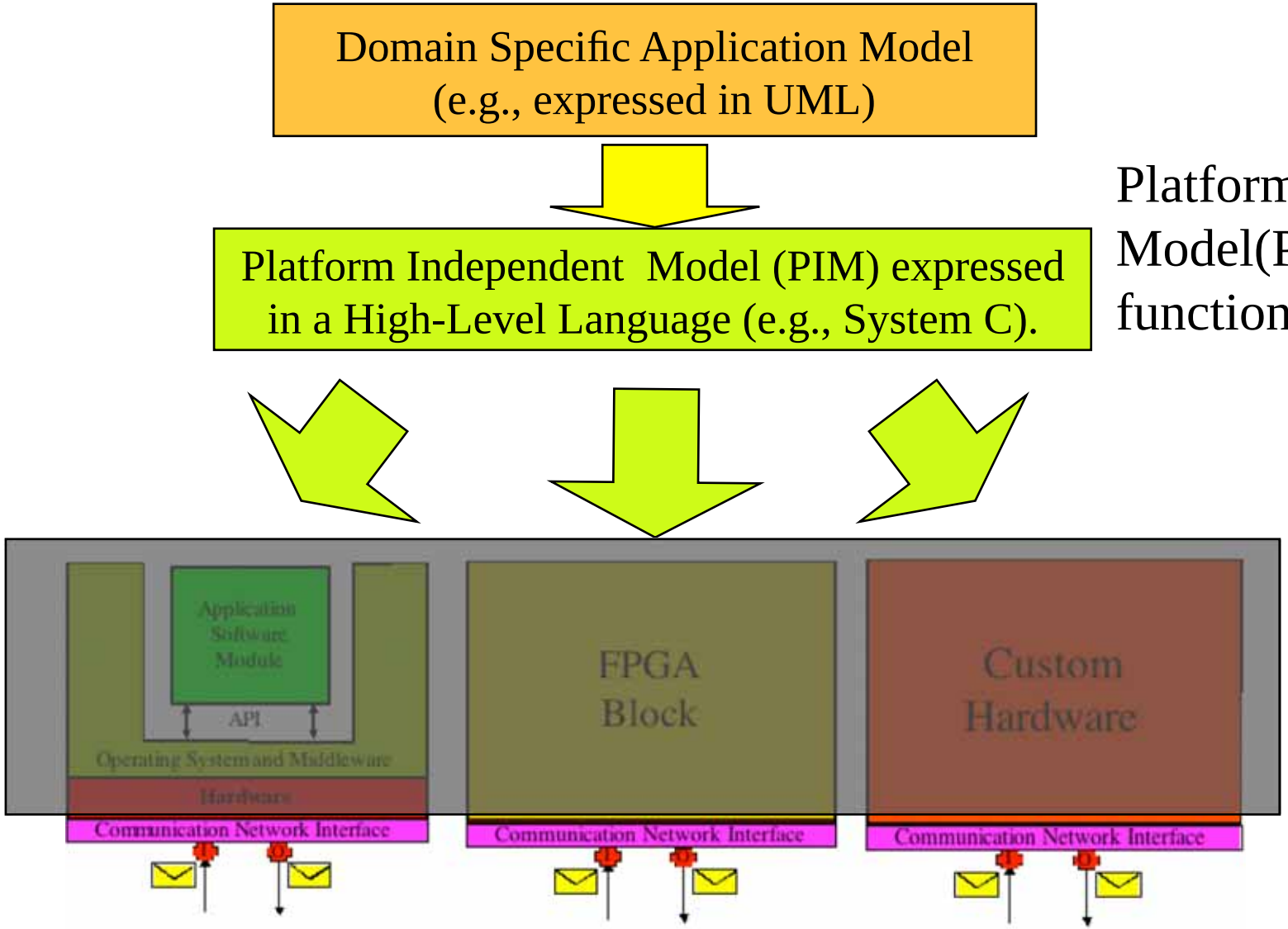
Openness: Soft versus *Hard* Components

Local Interfaces--Open 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.

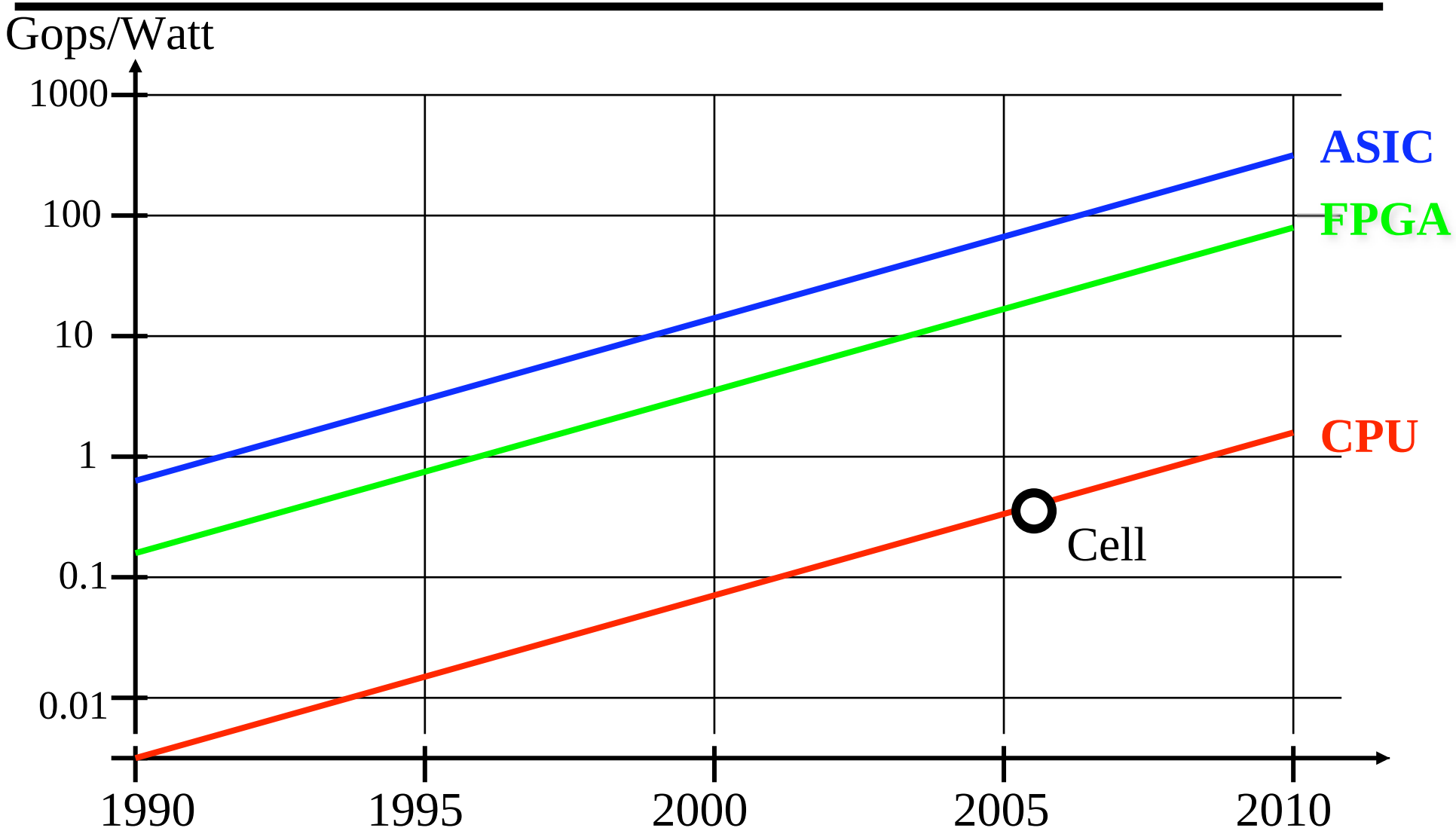
Abstraction: *Model Driven Design*



Platform Independent Model(PIM) focuses on functionality and time

Platform Specific Model (PSM) (nonfunctional properties)

Performance Trends--Power



Ref: Lauwereins, Imec, MSOP 2006

Component 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.

Communication in GENESYS

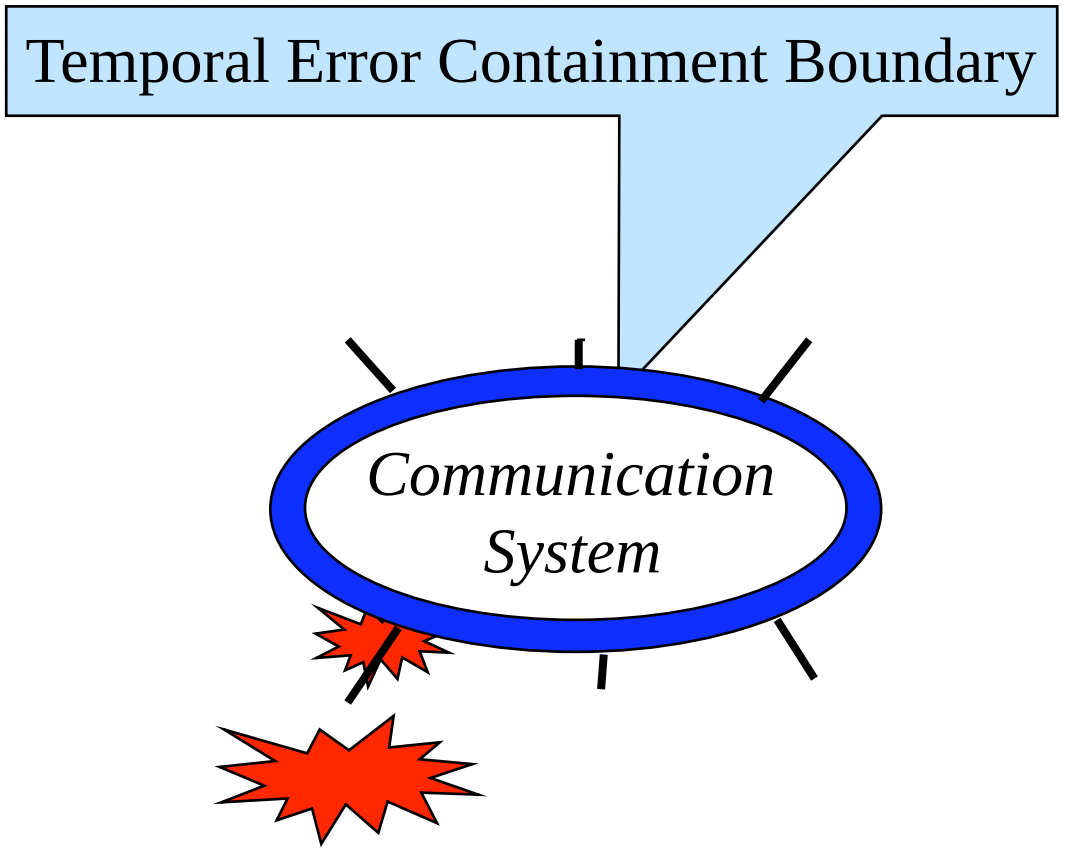
The core communication primitive in GENESYS is the ***unidirectional deterministic multi-cast*** **message:**

- *Uni-directionality* is required in order to decouple communication from computation (*fate-sharing* principle).
- *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 of certification
- *Multi-cast* is required to support the independent observation of the component behavior

Error Containment by the Communication System

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



Babbling idiot

GENESYS Communication Services

GENESYS introduces three communication services

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

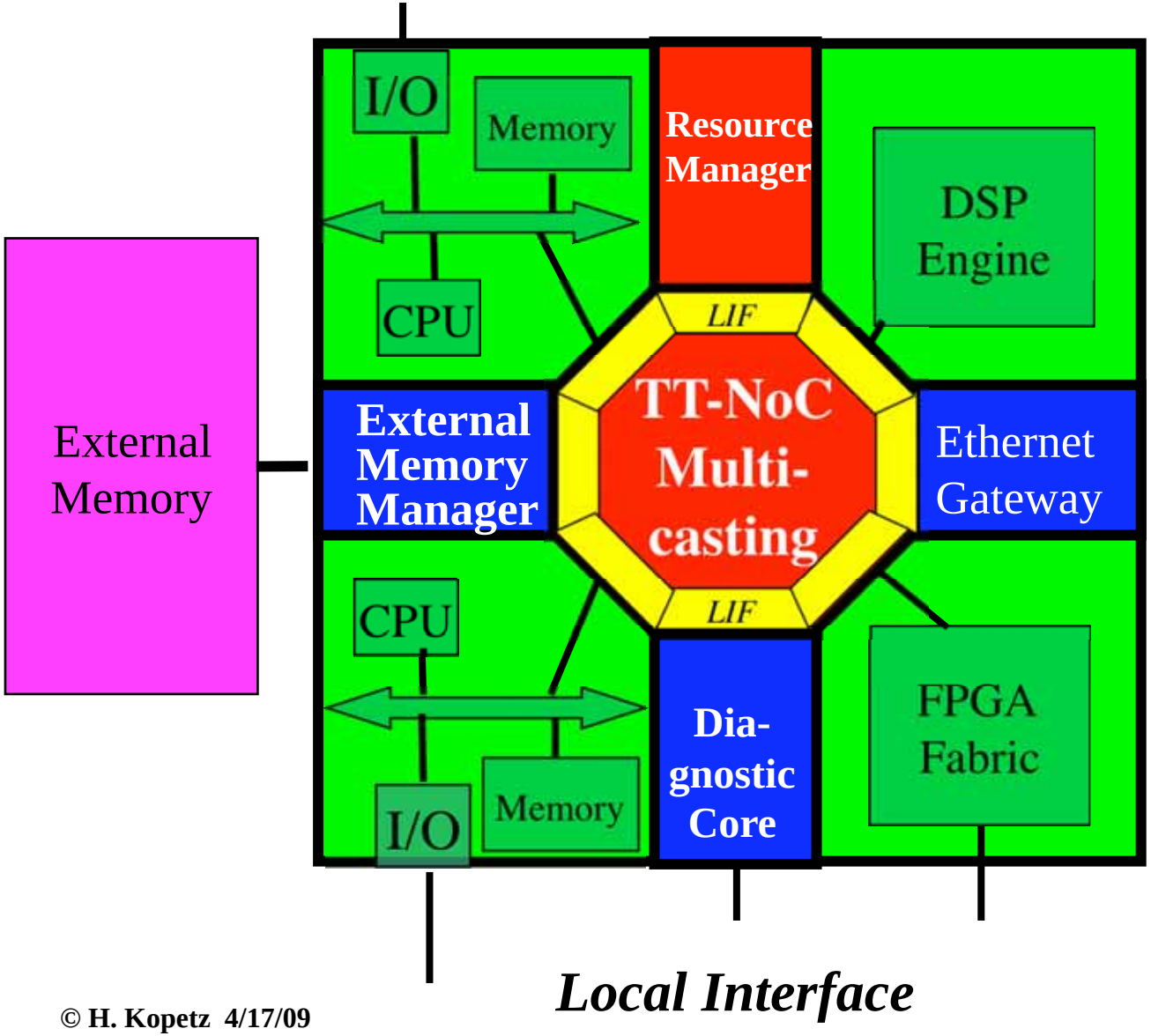
***Openness*: Any communication protocol (wire-bound or wireless) that provides these services can be used in GENESYS**

Partitioning: *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

A GENESYS System-on-a-Chip: *Chip-Level*



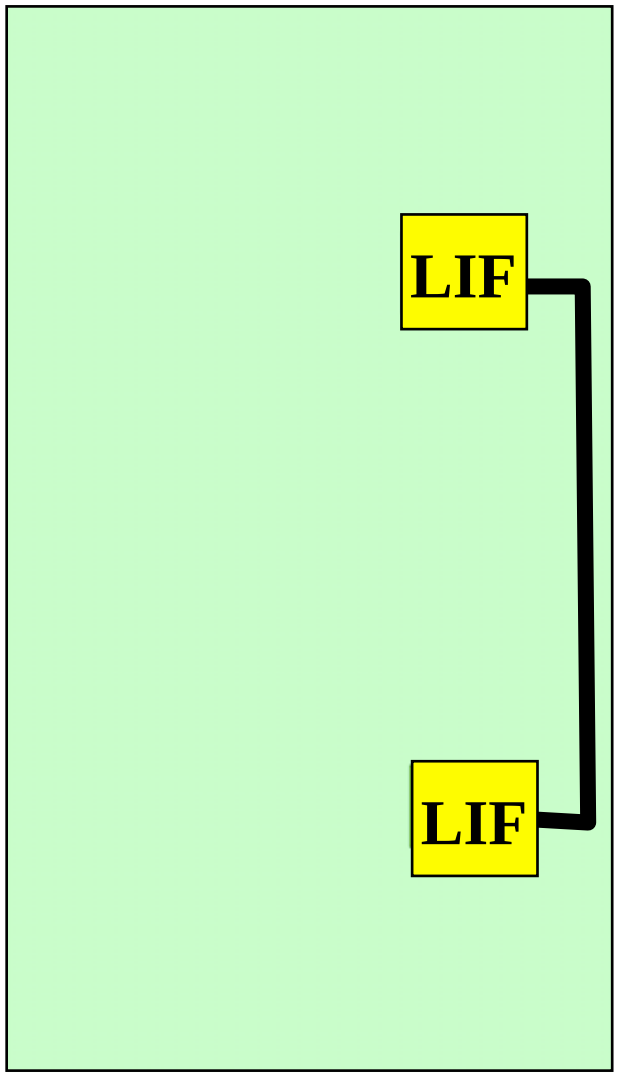
Standard components (IP-cores):

- External Memory Manager
- Resource Manager
- Diagnostic Core
- Ethernet Gateway

Application components

- Two CPUs with application software
- DSP Engine
- FPGA Fabric

Device-Level:



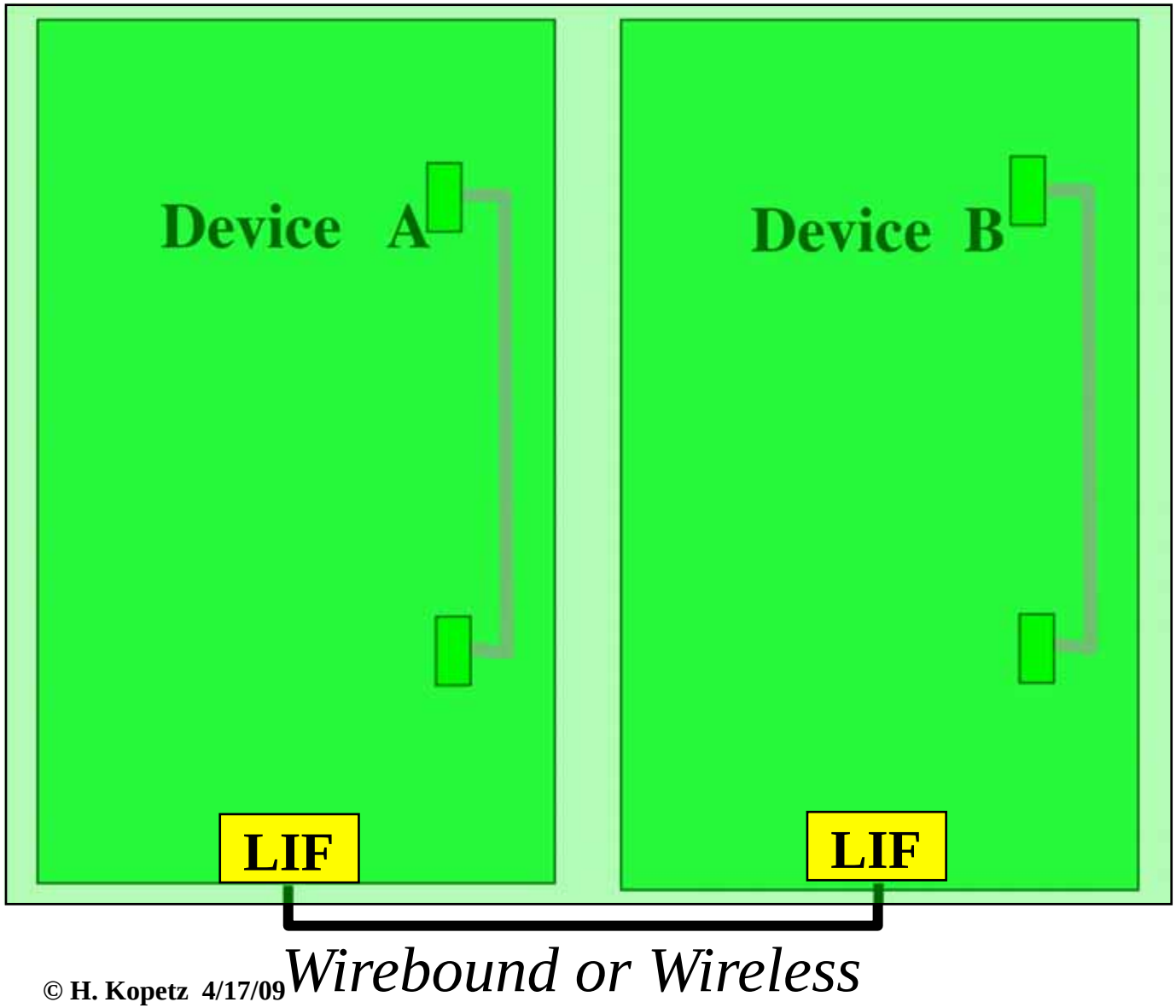
Chips are linked by ***intra-device-level LIFs*** to form a device.

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:



Devices are linked by

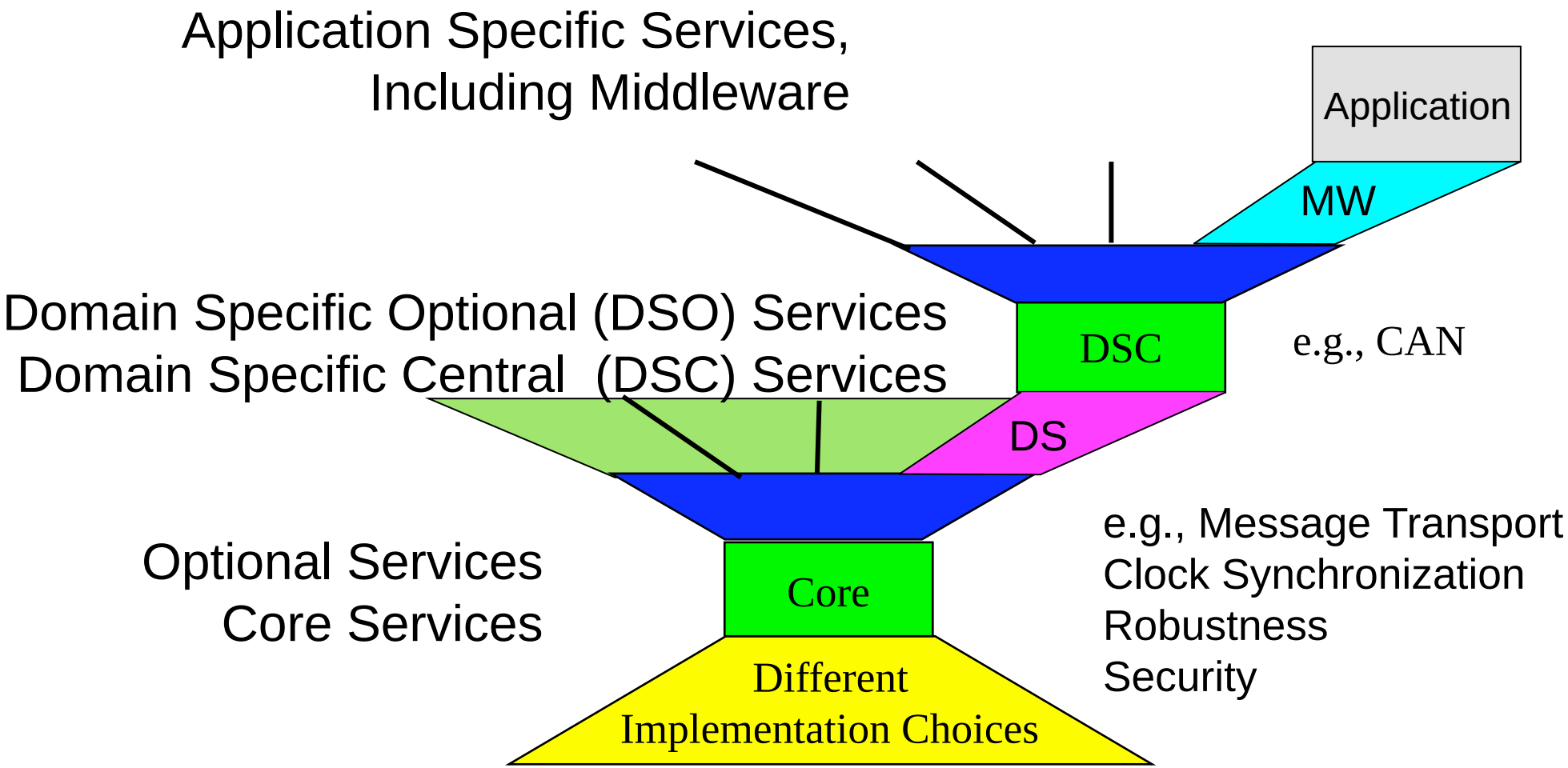
System Level LIFs

We distinguish between

- Open
- Closed

Systems.

Abstraction: *Hierarchy of GENESYS Services*

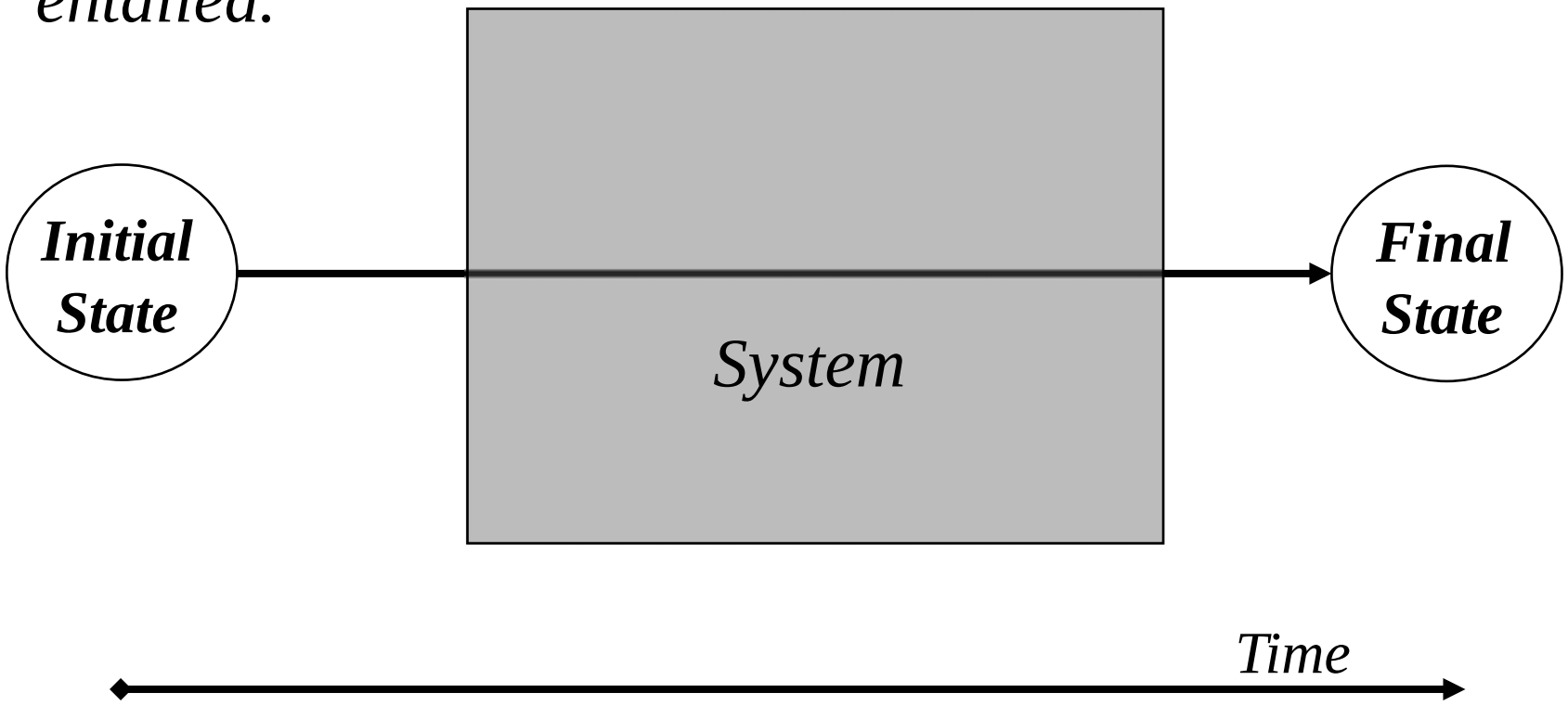


Typical Optional Services

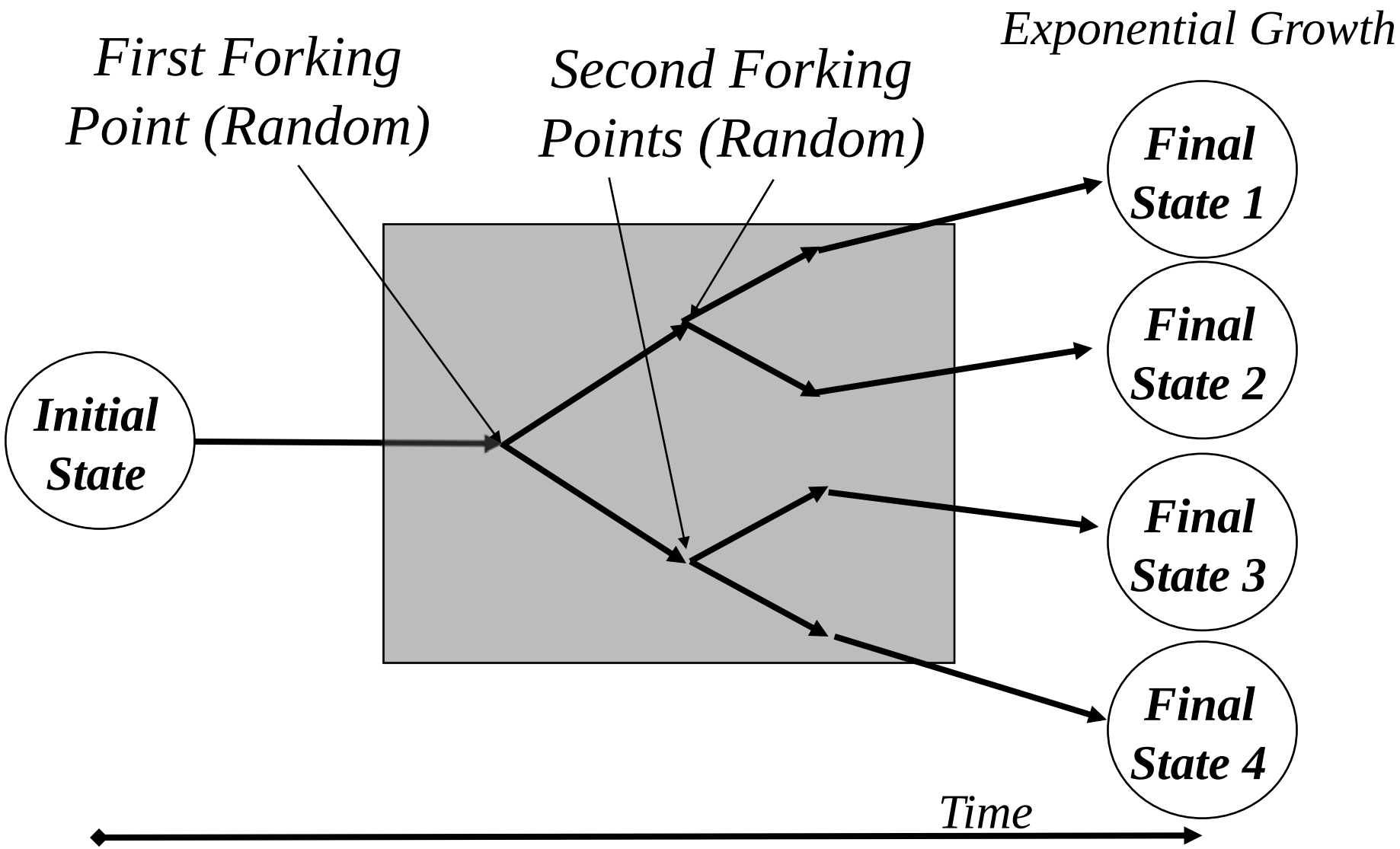
- Integrated Resource Management
- Diagnostic and Robustness Services
- Security Services
- Man-Machine Interface via a *Web Browser*
- Internet Connectivity-- *Internet of Things*
- Voting Service for Triple Modular Redundancy
- Overlay Network Service for domain-specific protocols (e.g., CAN, Most, ETHERNET)
- . . . .

Segmentation: *Determinism Helps*

Determinism: *A model behaves deterministically if and only if, given a full set of initial conditions (the initial state) at time t_o, and a sequence of future timed inputs, the outputs at any future instant t are entailed.*



Behavior of an Indeterminate System



Conclusion

- The cross domain Architectural Style of GENESYS supports a strict *component-based design style* and supports the straightforward composition of systems out of components.
- The *stable core-services* of GENESYS, which can be implemented cost-effectively in hardware, are the basis for the realization of *flexible* domain specific higher-level services.
- The Architectural Style of GENESYS supports the established simplification strategies of *Partitioning*, *Abstraction* and *Segmentation*
- The deterministic core services of GENESYS simplify the reasoning about the behavior (segmentation).