Real-Time System Modeling 1

Clusters, Components & Interfaces

slide credits: H. Kopetz, P. Puschner

Overview

Model Construction

- Clusters
- Component & state
- Interfaces

Model Construction

- Focus on the essential properties – purpose.
- Model assumptions must be stated explicitly (assumption coverage, load & fault hypothesis)
- The elements of the model and the relationships between the elements must be well specified.
- Understandability of structure and functions of the model are important.
- Formal notation to increase the precision.

Elements of an RTS Model

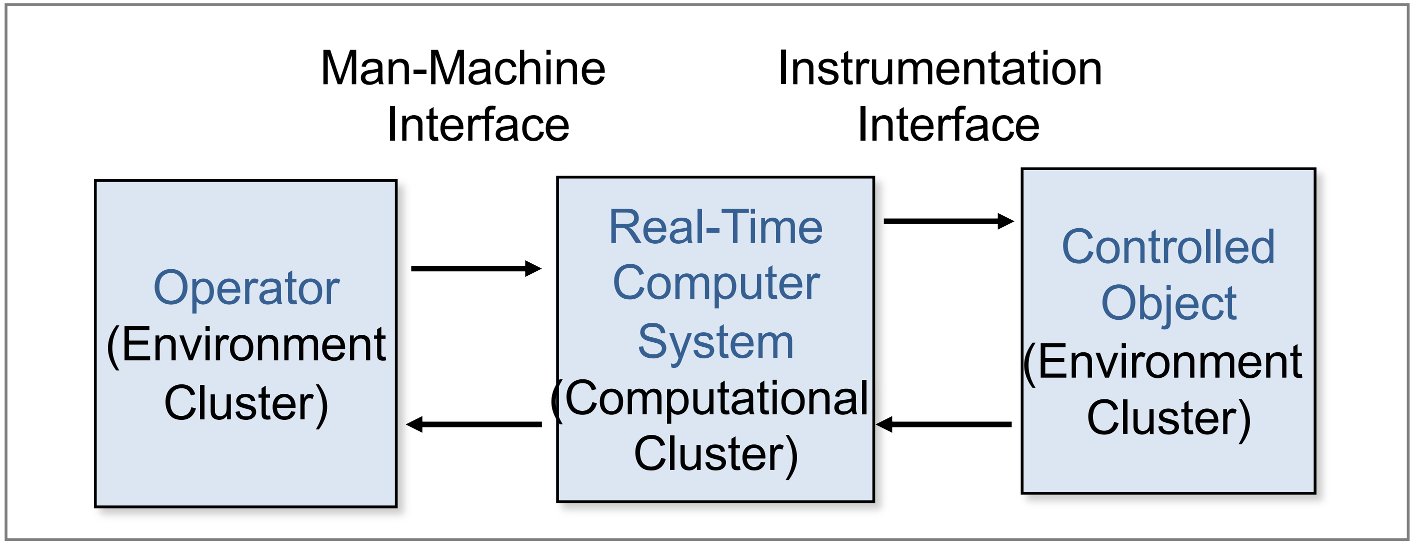
Essential:

- Representation of real-time
- Semantics of data transformations
- Durations of the executions

Unnecessary Detail:

- Representation of information within a system (only important at interfaces – specified by architectural style).
- Detailed characteristics of data transformations
- Time granularity finer than the application requirement

Structure of an RTS



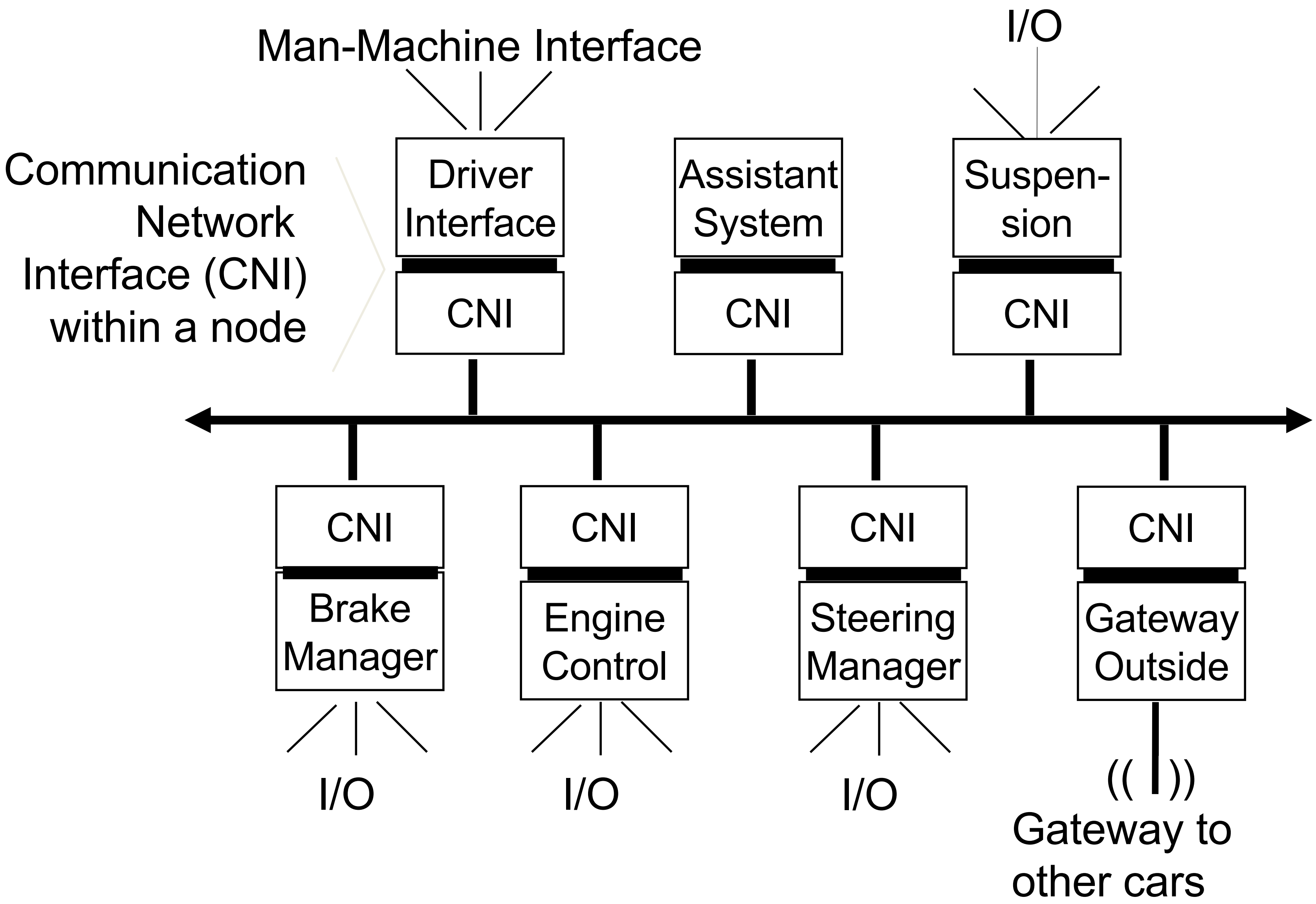
RTS: Controlled Object + Computer System + Operator

Cluster: subsystem of RT-system with high inner connectivity

Node: hardware-software unit of specified functionality

Task: Execution of a program within a component

Example of a Cluster



Computational Cluster

A set of co-operating components that

- provide a specified service to the environment (or some part thereof)
- use a unified representation of the information (messages)
- have high inner connectivity
- provide small interfaces to other clusters
- solve the dependability problem, e.g., by providing Fault Tolerant Units (FTUs)

Component

- Building block of larger system (cluster)
- Provides a clearly defined service
- Service interface specification describes the service ⇔ integration
- Integration must not require knowledge about component internals
- A real-time component has to be time-aware

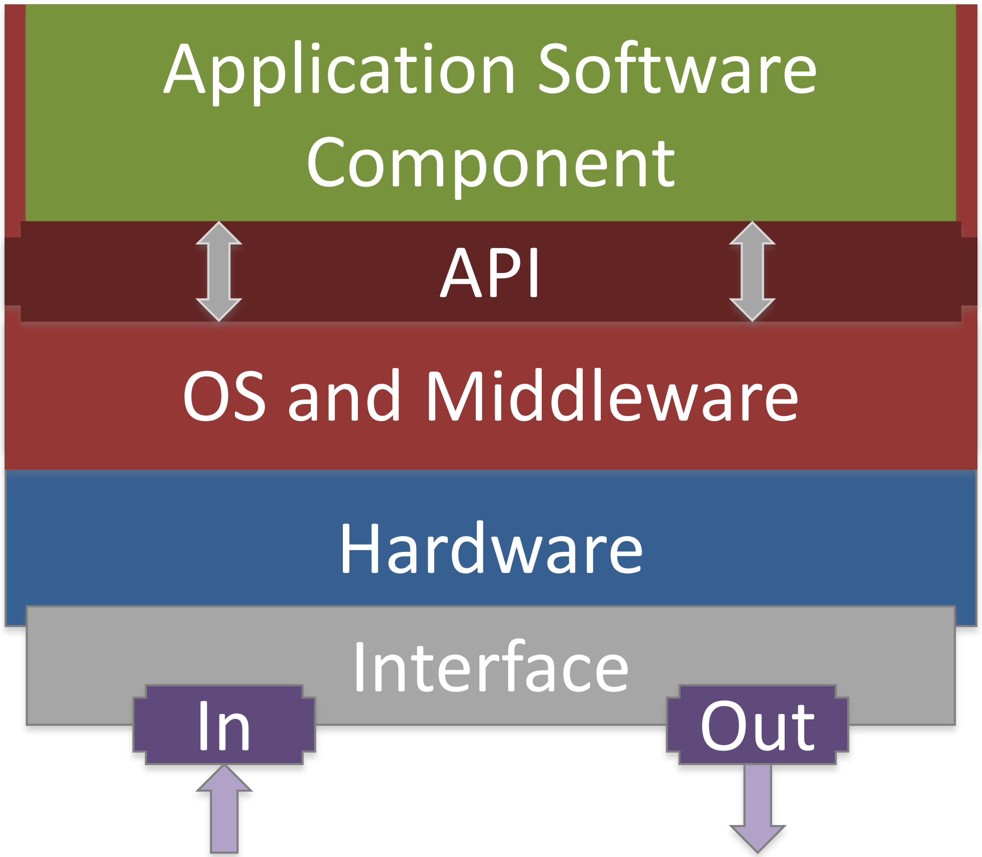
Components for RTS

- Software unit?
- Hardware-software unit?

RT component

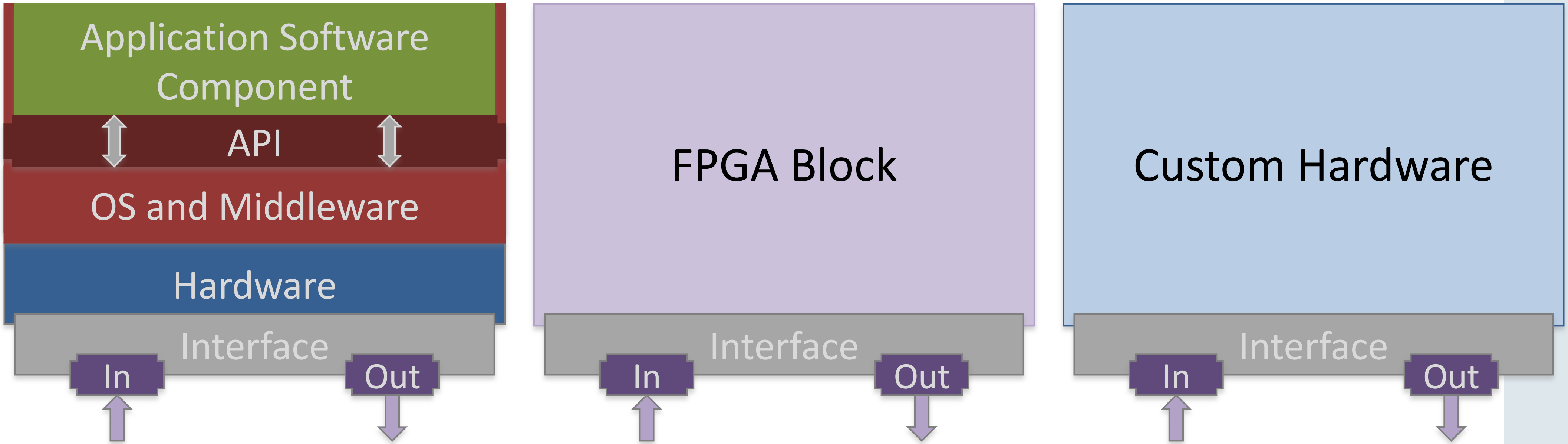
- Complete computer system – a node (… a core?)
  - Hardware + software
- Time-aware

Real-Time Component



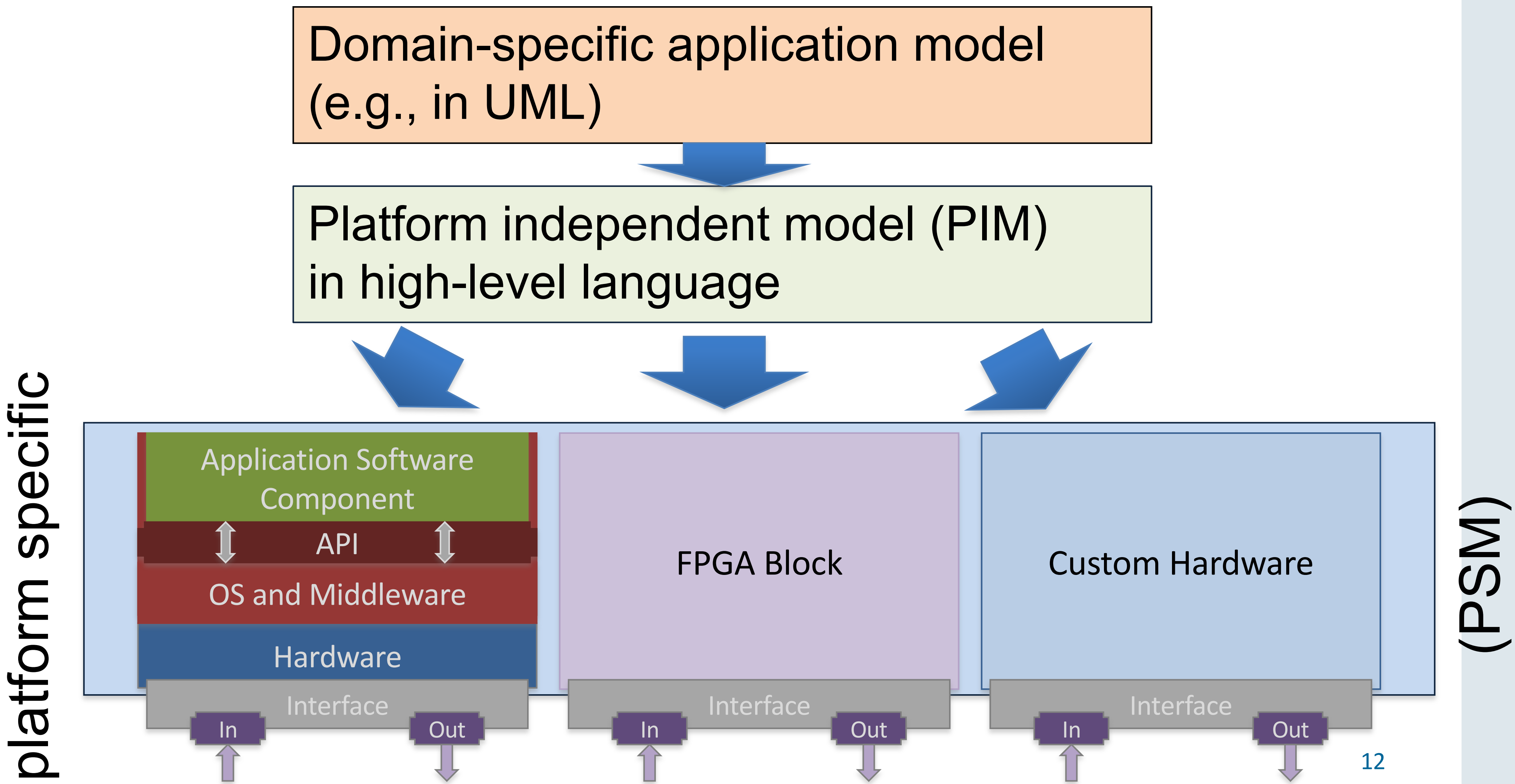
Software + Hardware!

Real-Time Component Realizations



- Interfaces must have the same syntax, semantics, timing
- Implementations are not distinguishable by the user

Model Driven Design: from PIM to PSM



Description of Component Behaviour

- Clearly describe dynamic behaviour/evolution of state in the time domain
- Support fault tolerance (dynamic service transfer, hot standby)

⇨ State

State

State separates the *past* from the *future*

"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 words, 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, Abstract System Theory, p.45]

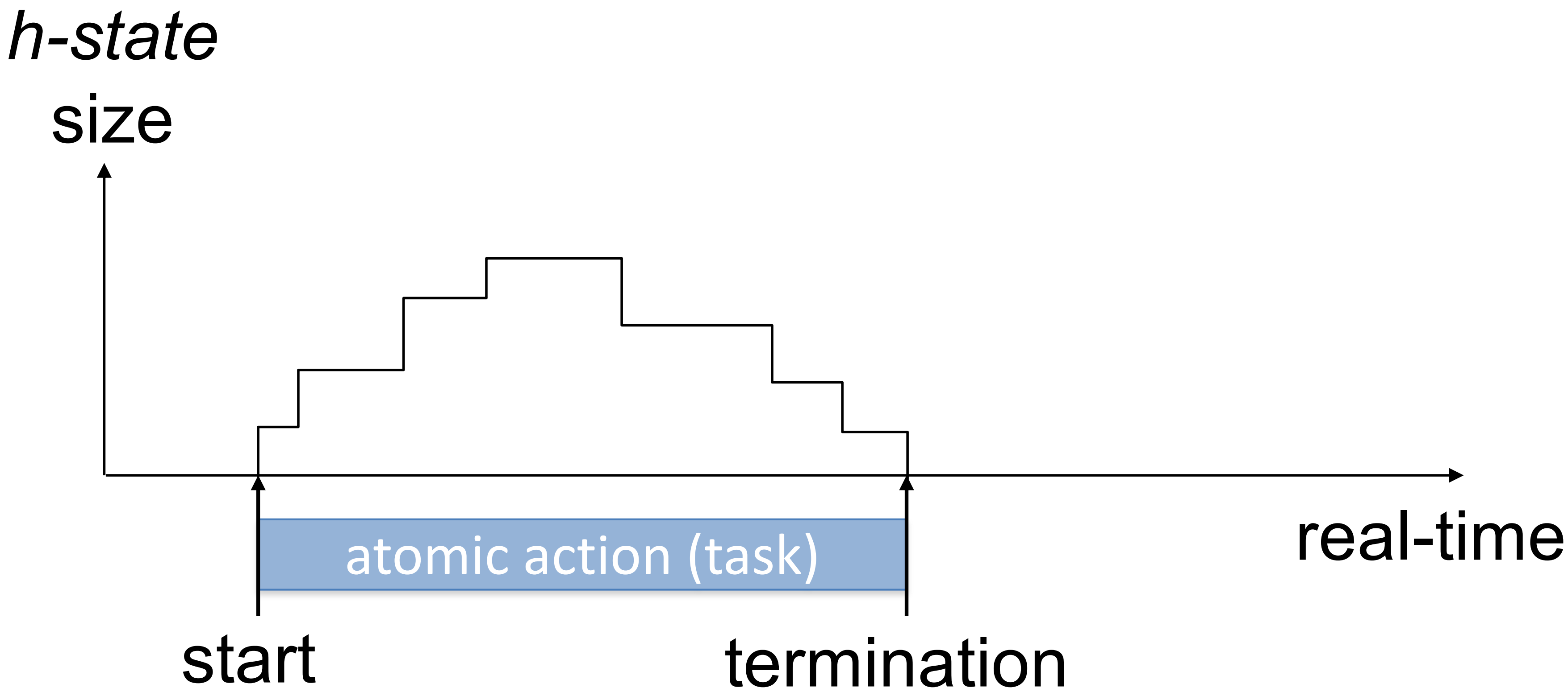
Component State

A hardware-software component consists of

- Hardware
- Operating System (pre-installed system software)
- State
  - *i-state* (initialization state): static data structure, i.e., application program code, initialization data (e.g., in ROM)
  - *h-state* (history state): dynamic data structure that contains information about current and past computations (in RAM)

A system-wide consistent notion of time – sparse time – is necessary to build a consistent notion of state

History-State Size during Atomic Action



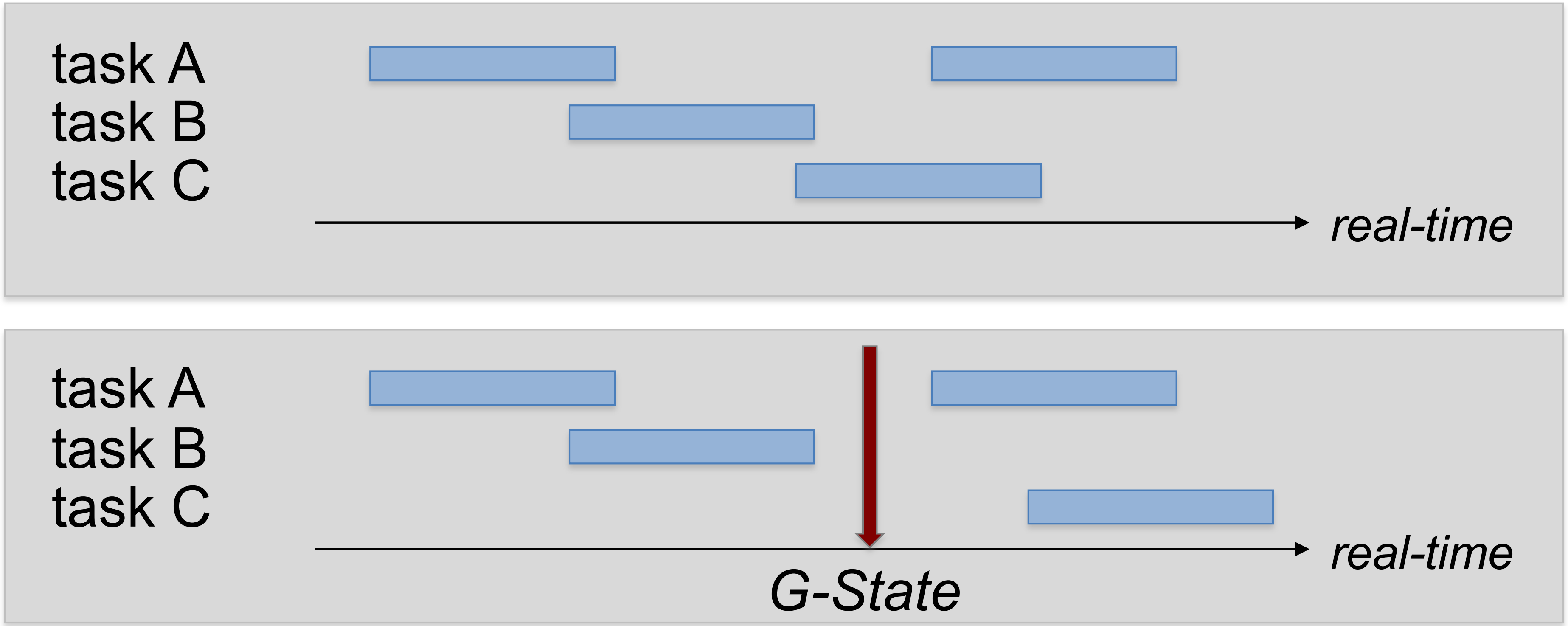
History State (H-State)

The *h-state* comprises all information that is required to start an initialized node or task (*i-state*) at a given *point in time*

- Size of the *h-state* changes over time
- relative minimum immediately after end of computation (atomic action).
- shall be small at reintegration points.
- *g-state* (ground state) of a system: minimal h-state, when all tasks are inactive and all channels are flushed (no messages in transit)
  ➪ ideal for *reintegration*.

Stateless node: no h-state has to be stored between successive activations (at the chosen level of abstraction!)

Ground State (G-State)



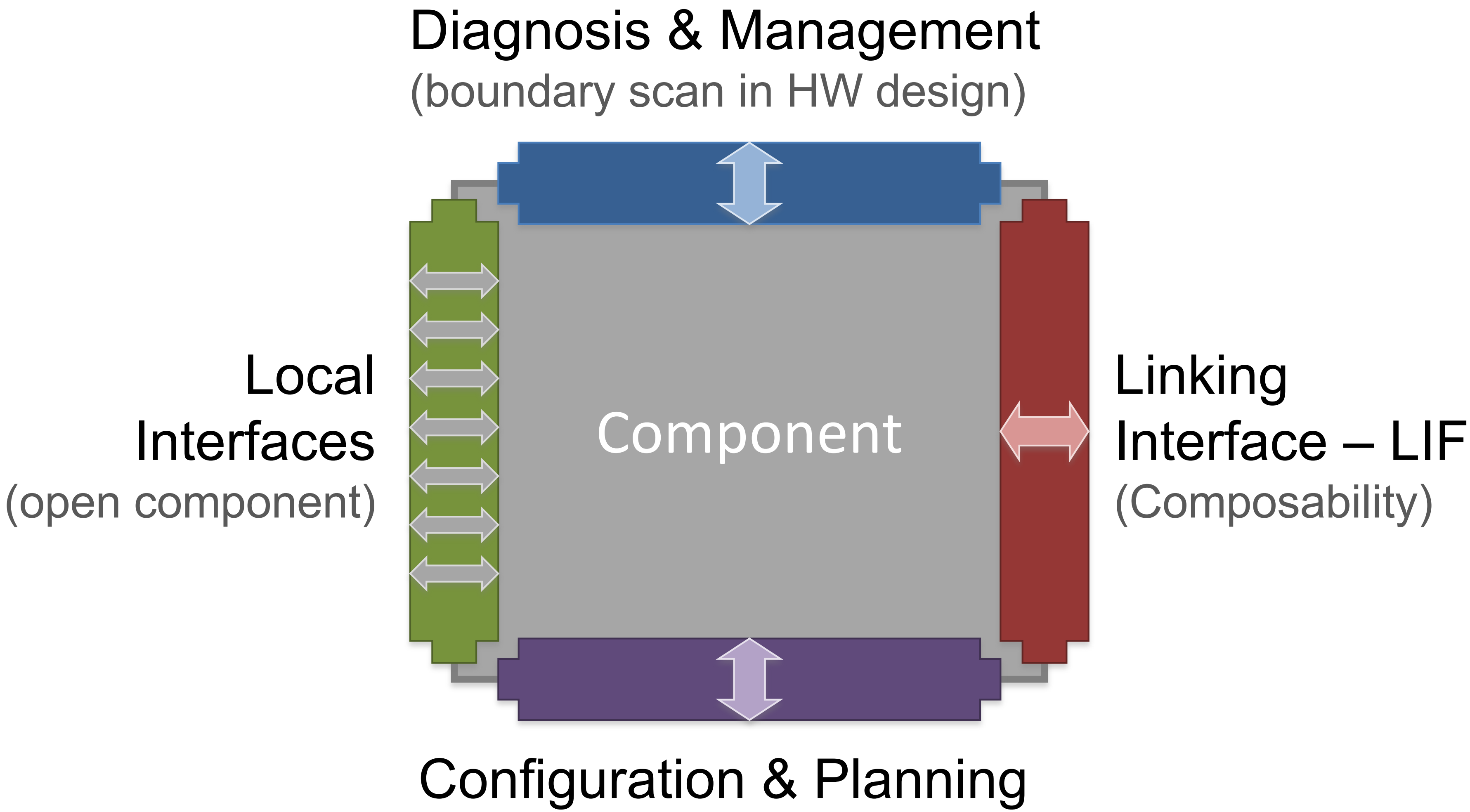
- Minimal h-state of a subsystem
- Tasks are inactive, communication channels are flushed

Interface

Common boundary between two systems, characterized by

- *Data properties*
  structure and semantics of the data items crossing the interface, including the *functional intent*
- *Temporal properties*
  temporal conditions that have to be satisfied, e.g., update rate and temporal data validity
- *Control properties*
  strategy for controlling the data transfer between communicating entities

Component Interfaces



The Four Interfaces of a Component (1)

Realtime Service (RS) or Linking Interface (LIF):

- In control applications periodic
- Contains RT observations
- Time sensitive

Diagnostic and Maintenance (DM) Interface – Technology Dependent (TDI):

- Sporadic access
- Requires knowledge about internals of a node
- Not time sensitive

The Four Interfaces of a Component (2)

Configuration Planning (CP) Interface –
Technology Independent (TII):

- Sporadic access
- Used to install a node into a new configuration
- Not time sensitive

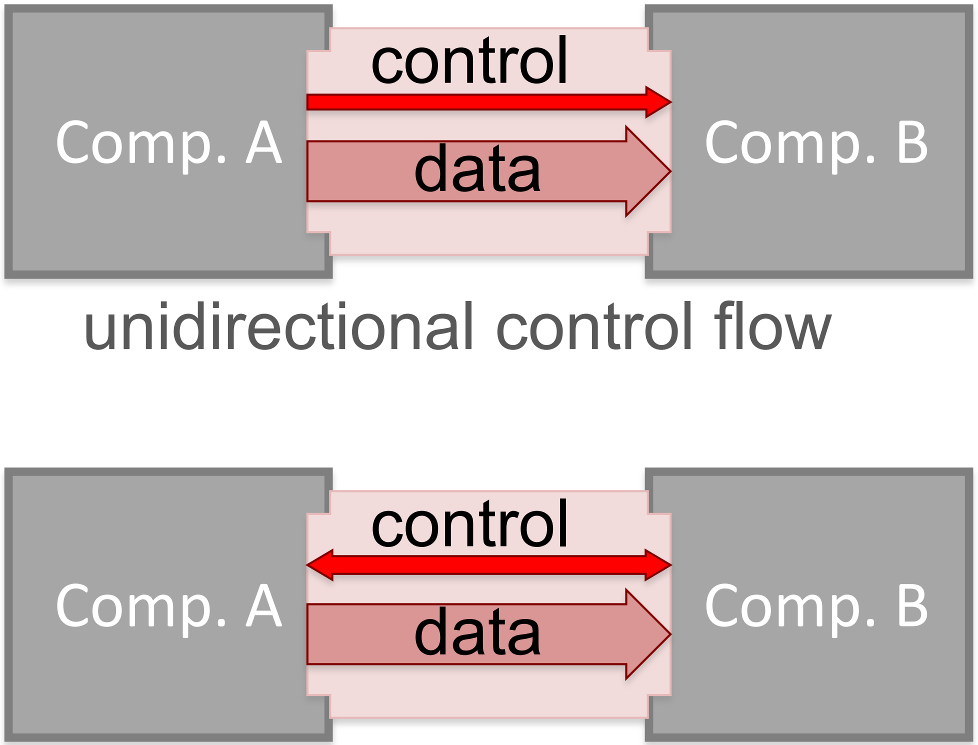
Local Interface to the Component Environment

Component Communication via LIF

LIF must provide temporal composability, it specifies:

- Temporal preconditions
  points in time when component inputs are available
  (time instants, rates, order, phase relationship)
- Temporal post-conditions
  points in time when component outputs are available
- Functional properties of the information transformation performed by the component (proper model)
- Syntactic units
- Interface state
- Interface control strategy

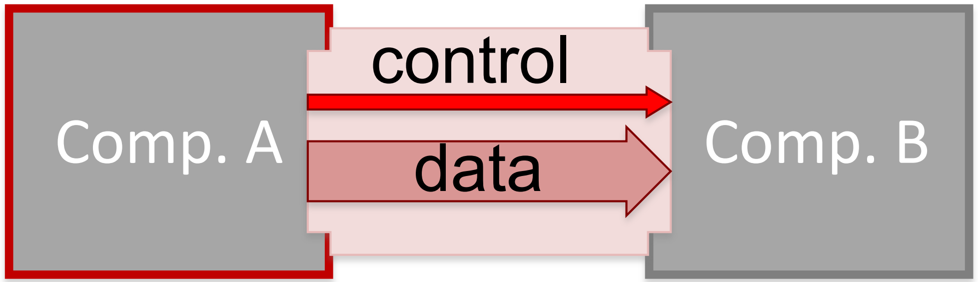
Interfaces and Control



Elementary interfaces are simpler!

Information Push vs. Information Pull

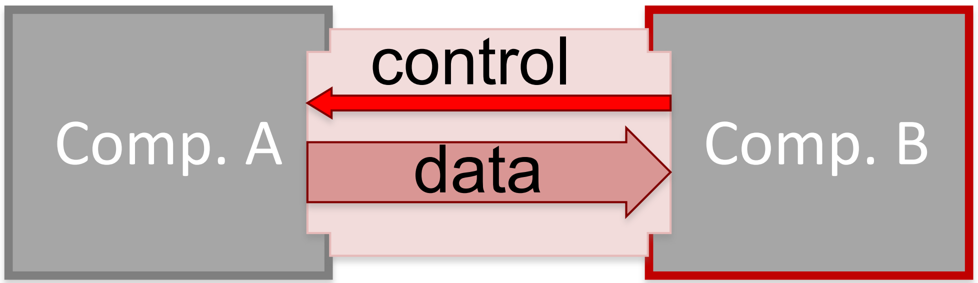
Information Push



Example: telephone call, interrupt

Producer pushes information on consumer

Information Pull



Example: checking email

Consumer retrieves information when required

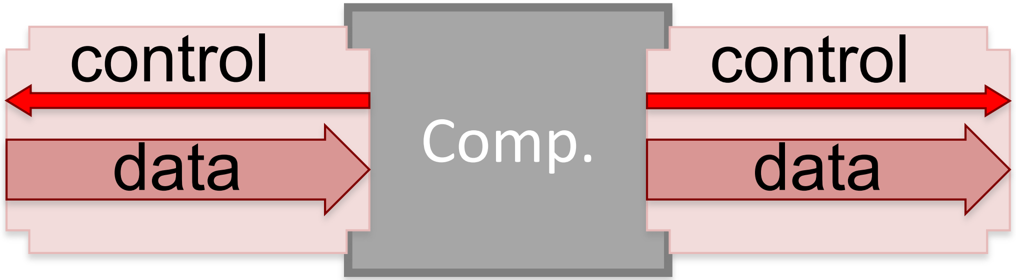
Temporal Firewall

Desirable control semantics at RT interfaces

- Producer → information push
- Consumer → information pull

Temporal Firewall

Interface that prohibits external control on a component



component with two temporal-firewall interfaces

Component Categories

Closed Component

- Linking interfaces to other components
- No local interface to the controlled environment (real world)

Semi-closed Component

- Time-aware, closed component

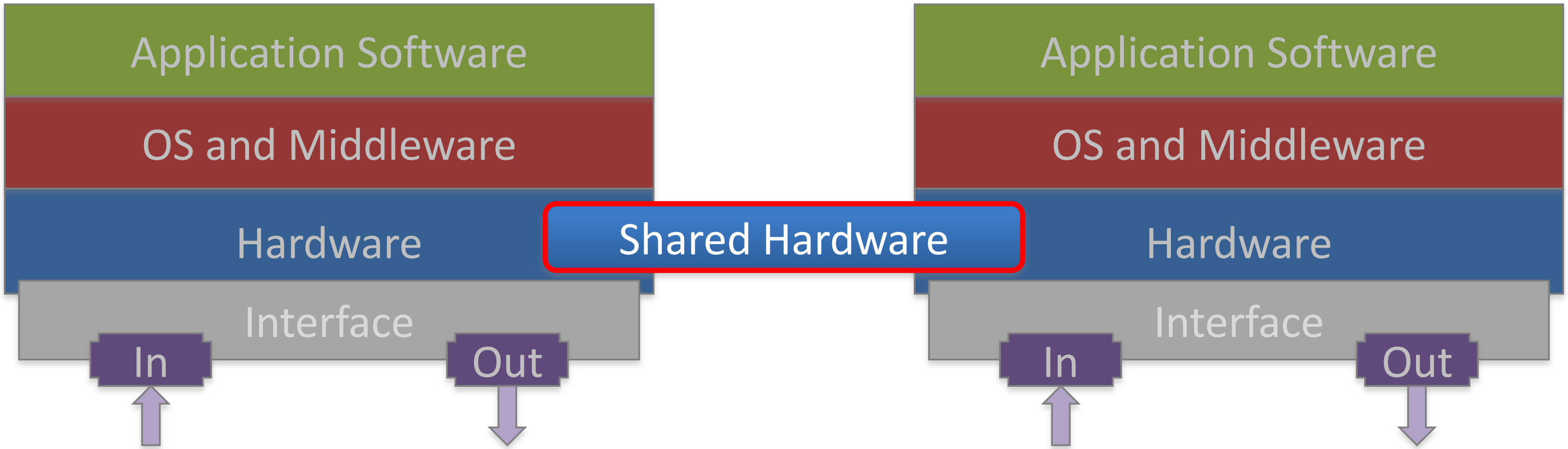
Open component

- Local real-world interface

Semi-open component

- Does not allow control signals from the real world (e.g., sampling, polling)

Shared HW on RT Component



Memory ⇨ Mutual Exclusion
Bus ⇨ Arbitration
Cache ⇨ Pollution

Control of **Timing**,
Error Containment

NO !

Component Interfaces

Real-time interface = message interface

Network communication

- Cluster level: real-time network of cluster;
  Gateway components link local interfaces to system
- Node level (multicore): network on chip

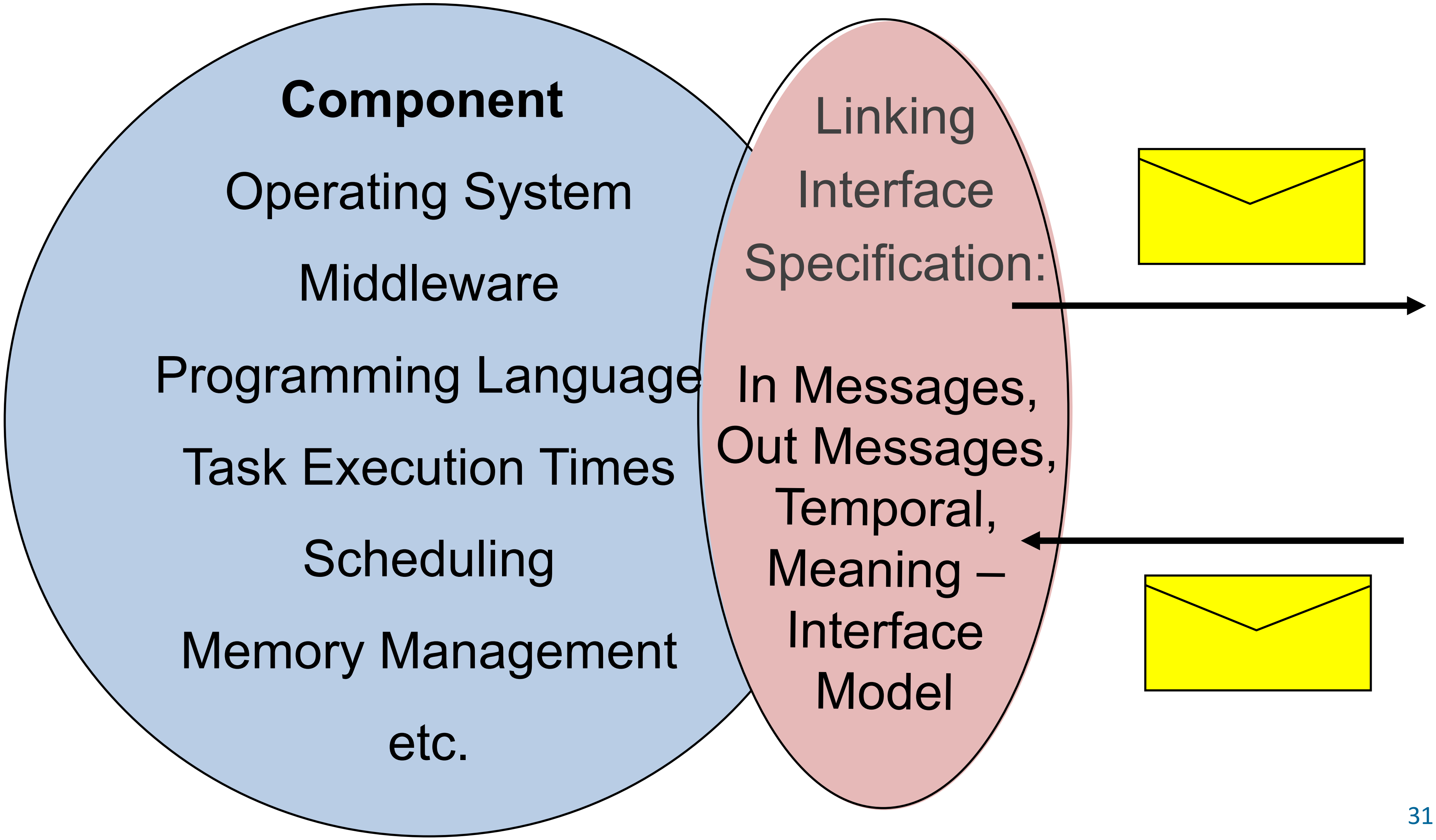
System Design = Message Specificaton

System Design
interactions among all components are specified

- Abstract message interface → message data structures
- Timing (period, phase) and control semantics of messages
- Response time of nodes
- Ground state of nodes

Subsequent component design is constrained by the message specifications

LIF Specification Hides Implementation



System Views: Four-Universe Model

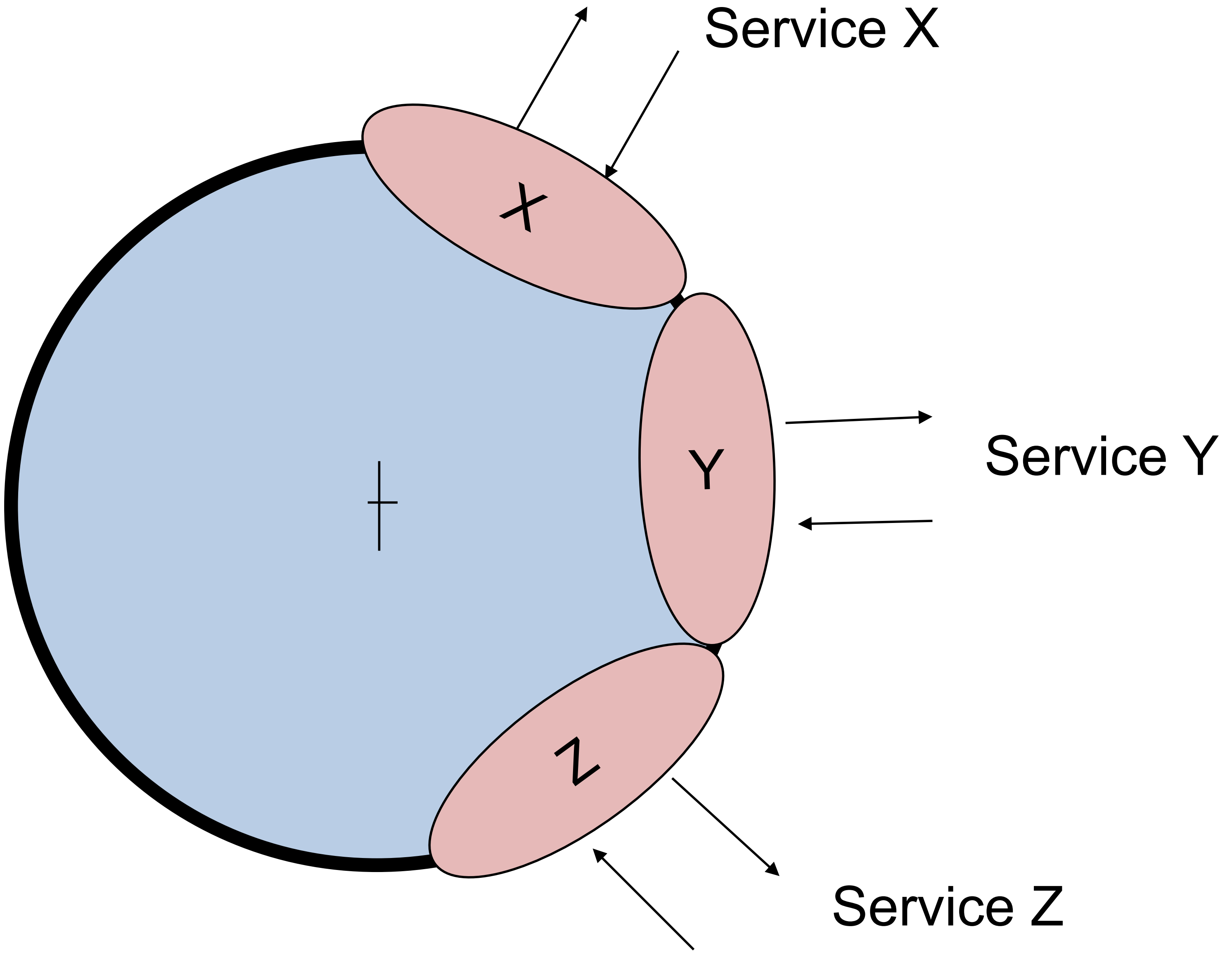
User Level Meaning of Data Types	Meta-level Specification Interpretation by the User
Informational Level Data Types	Operational Interface Specification, Value, Temporal, Control Properties
Logical Level Bits	
Physical Level Analog Signals	

Avizienis, FTCS 12, 1982

Metalevel Specification

- Assigns a meaning to the syntactic units of the operational specification by referring to a LIF service model.
- Bridges the gap between information level and user level (means-and-ends model)

Component with Multiple LIFs



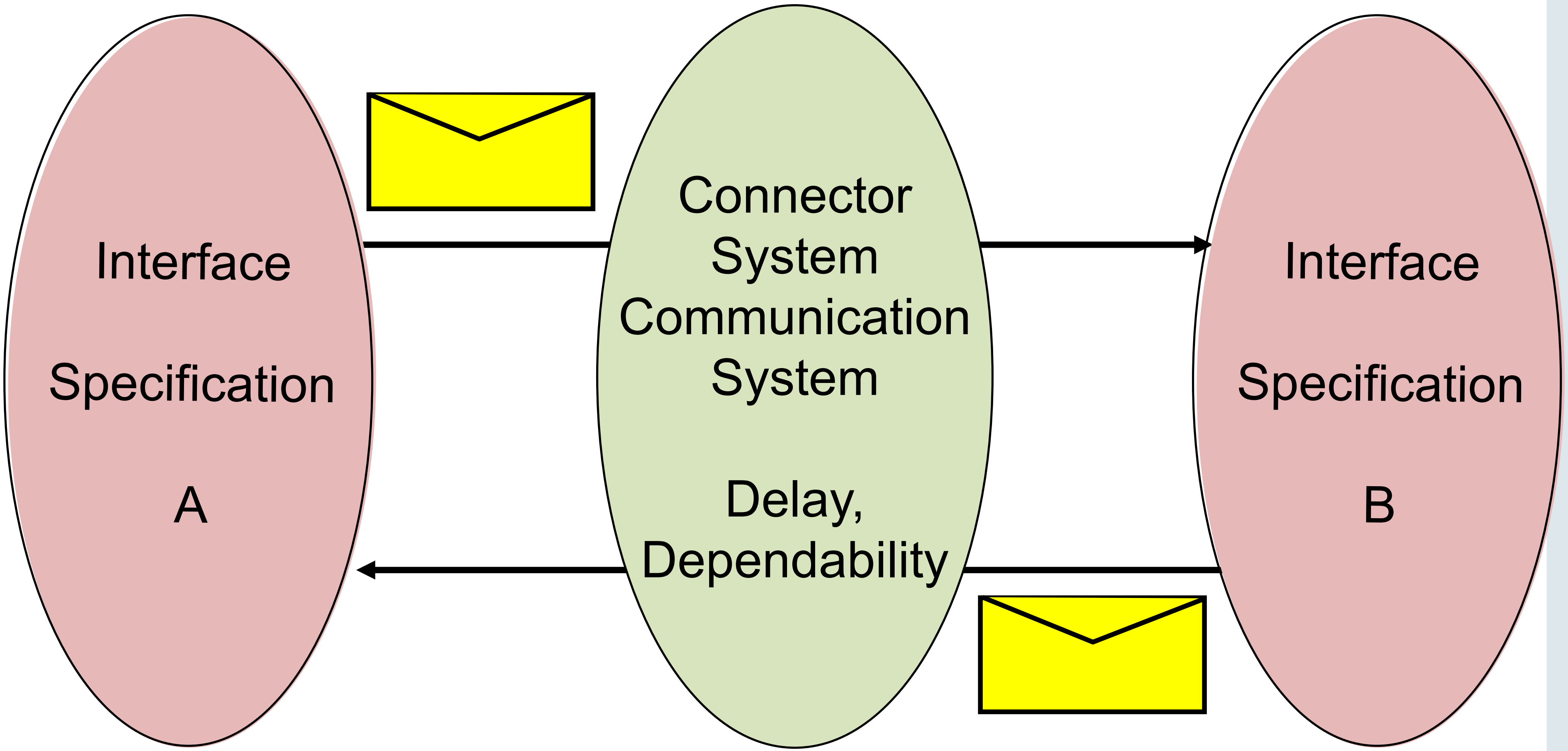
Interfaces – Property Mismatches

Property	Example
Physical, Electrical Communication protocol	Line interface, plugs, CAN versus J1850
Syntactic	Endianness of data
Flow control	Implicit or explicit, Information push or pull
Incoherence in naming	Same name for different entities
Data representation	Different styles for data representation Different formats for date

Interfaces – Property Mismatches (2)

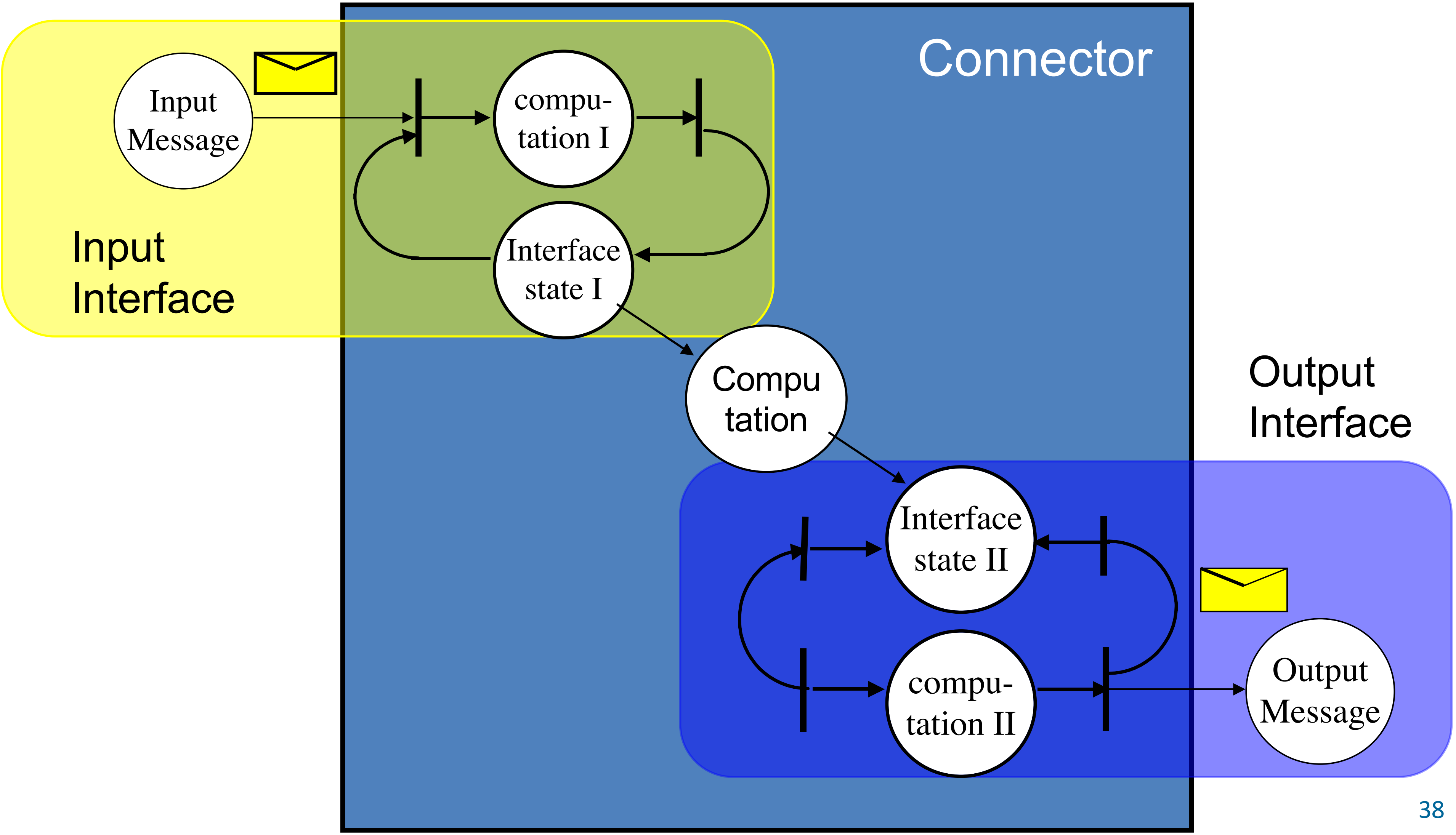
Property	Example
Temporal	Different time bases Inconsistent timeouts
Dependability	Different failure-mode assumptions
Semantics	Differences in the meaning of the data

Connector System



A connector system resolves interface mismatches

Example: Text-to-Speech



Example: Text-to-Speech (2)

Input Interface:

- Accepts text, following client-server paradigm
- Requests are event-triggered
- Composite interface

Output Interface:

- Produces a bit-stream that encodes sound
- Output is time-triggered
- Elementary, temporal firewall interface

Information Representation at Interfaces

- Every interface belongs to two subsystems
- Information may be coded differently within these subsystems

Abstract Interface

- Differences in the *information representation* are of no concern, as long as the semantic contents and the temporal properties of the information are maintained across the interface.

Low-level Interface

- *Information representation* between different subsystems is relevant (not within a properly designed subsystem, e.g., a cluster, with a unique information-representation standard).

Message Interface vs. Real-world Interface

World interface: concrete, low-level interface to devices

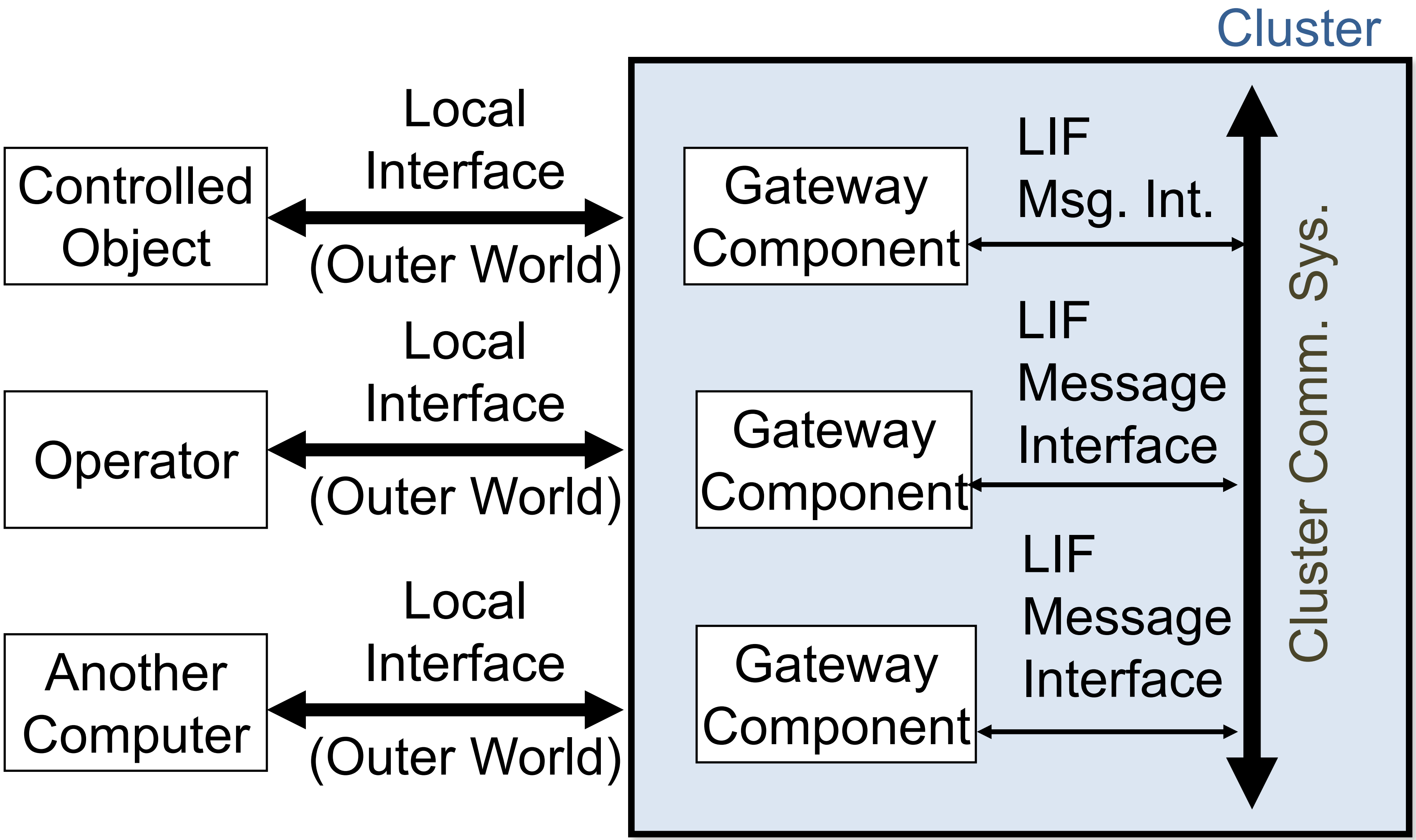
Message interface: internal, abstract message-based interface of a cluster

Resource controller:

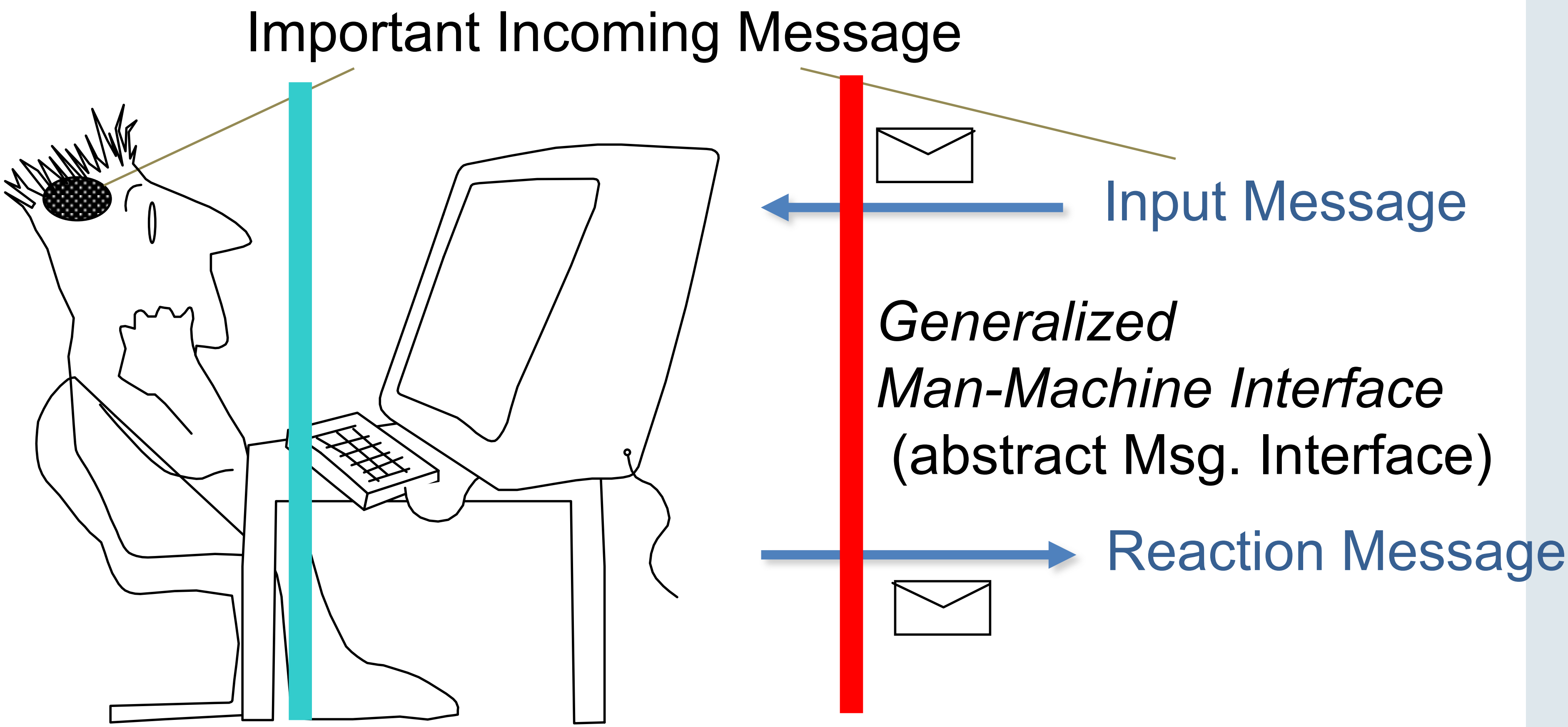
- Interface component between message and world interface
- acts as an "information transducer"
- hides the concrete physical interface of real-world devices from the standardized information representation within a cluster
- is a kind of gateway

[Transducer (Webster): device that receives energy from one system, and retransmits it, often in a different form, to another].

World vs. Message Interface



Example: Text-to-Speech (3)



Specific Man-Machine Interface (Graphics, Sound)
(concrete World Interface)

World/Message Interface Characteristics

Characteristic	World Interface	Message Interface
Info. Representation	unique	standardized
Coupling/Responsiveness	tight	weaker
Coding	ana./digital	digital
Time Base	dense	sparse
Communication topology	1-to-1	multicast
Design Freedom	limited	given

Example: SAE J1587 Message Specification

The SAE has defined message standards, e.g., for heavy-duty vehicle applications (SAE J1587):

- Standardized Message IDs and Parameter IDs for many significant variables in the application domain
- Standardized data representation
- Definition of ranges of variables
- Update frequency
- Priority information

Message Classification

Attribute	Explanation	Antonym
valid	A message is *valid* if its checksum and contents are in agreement.	invalid
checked	A message is *checked at source* (or, in short, *checked*) if it passes the output assertion.	not checked
permitted	A msg. is *permitted* with respect to a receiver if it passes the input assertion of that receiver.	not permitted
timely	A message is *timely* if it is in agreement with the temporal specification.	untimely
value-correct	A message is *value-correct* if it is in agreement with the value specification.	not value-correct
correct	A msg. is *correct* if it is both timely and value-correct.	incorrect
insidious	A msg. is *insidious* if it is permitted but incorrect.	not insidious

Points to Remember

- Modeling – purpose & coverage
- RTS cluster model
- RT Component = hardware + software + state
- Interfaces and their properties