Real-Time System Modeling

slide credits: H. Kopetz, P. Puschner

Overview

- Model Construction
- Real-time components
- Interfaces
- Real-time interfaces and observations
- Temporal accuracy
- Permanence
- Replica determinism

Model Construction

- Focus on the essential properties – eliminate the unnecessary detail (purpose, viewpoint important).
- 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 describe the properties of the model should be introduced to increase the precision.
- Model assumptions must be stated explicitly.

Assumption Coverage

Every model/design is based on a set of assumptions

- about the environment
- about the behavior of the components

Assumption coverage

- Probability that the assumptions cover the real-world scenario
- Limits the dependability of a *perfect design* (also limits the utility of formal verification).

Specification of assumptions is a system-engineering task.

Load and Fault Hypothesis

Requirements specification has to include the following assumptions:

- Load Hypothesis: Specification of the peak load that a system must handle.
- Fault Hypothesis: Specification of number and types of faults that a fault-tolerant system must tolerate.

The fault hypothesis partitions the fault space into two domains: those faults that must be tolerated and those faults that are outside the fault-tolerance mechanisms.

Outside fault hypothesis: Never-give-up (NGU) strategy

Elements of an RTS Model

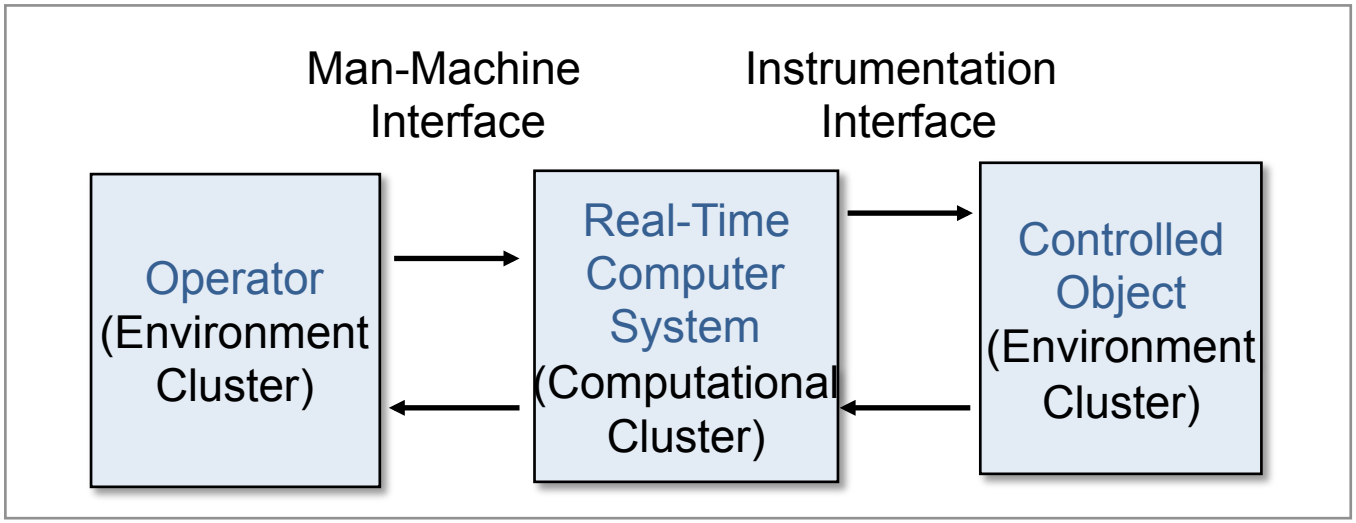
Essential:

- 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



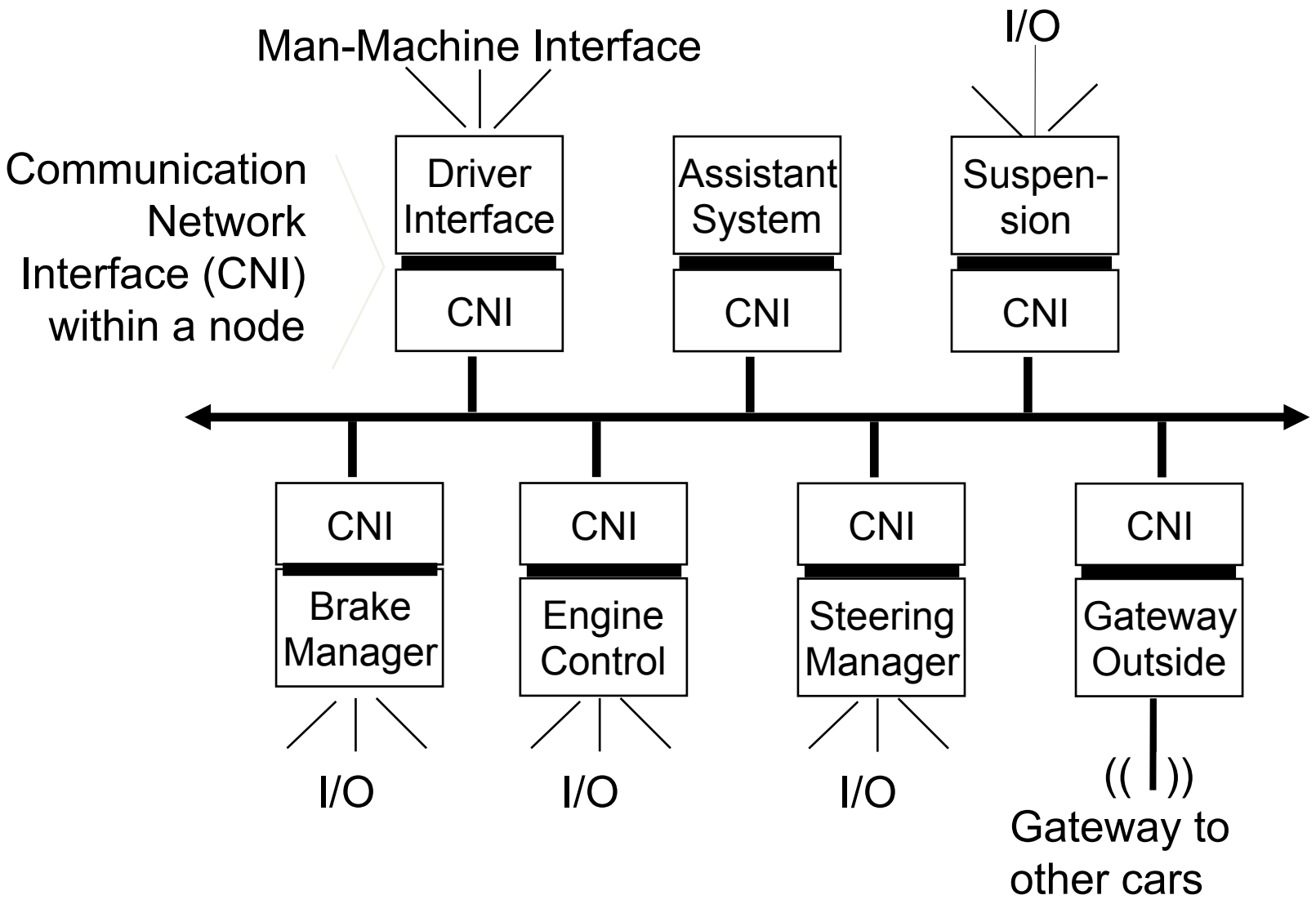
RTS: Controlled Object + Computer System + Operator

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



Cluster

A set of co-operating components that

- provide a specified service to some subsection of the environment
- use a unified representation of the information (messages)
- solve the dependability problem, e.g., by grouping replica determinate components into Fault Tolerant Units
- provide small interfaces to other clusters, not necessarily in a hierarchical fashion

Component

- Building block of large systems
- Provides a clearly defined service
- Service interface specification describes the service for the purpose of integration
- Integration must not require knowledge about component internals
- A real-time component has to be time-aware

Components for RTS

- *Software unit* (software component)
  for independent deployment?
- *Hardware-software unit* (system component)
  characterized by behaviour and state?

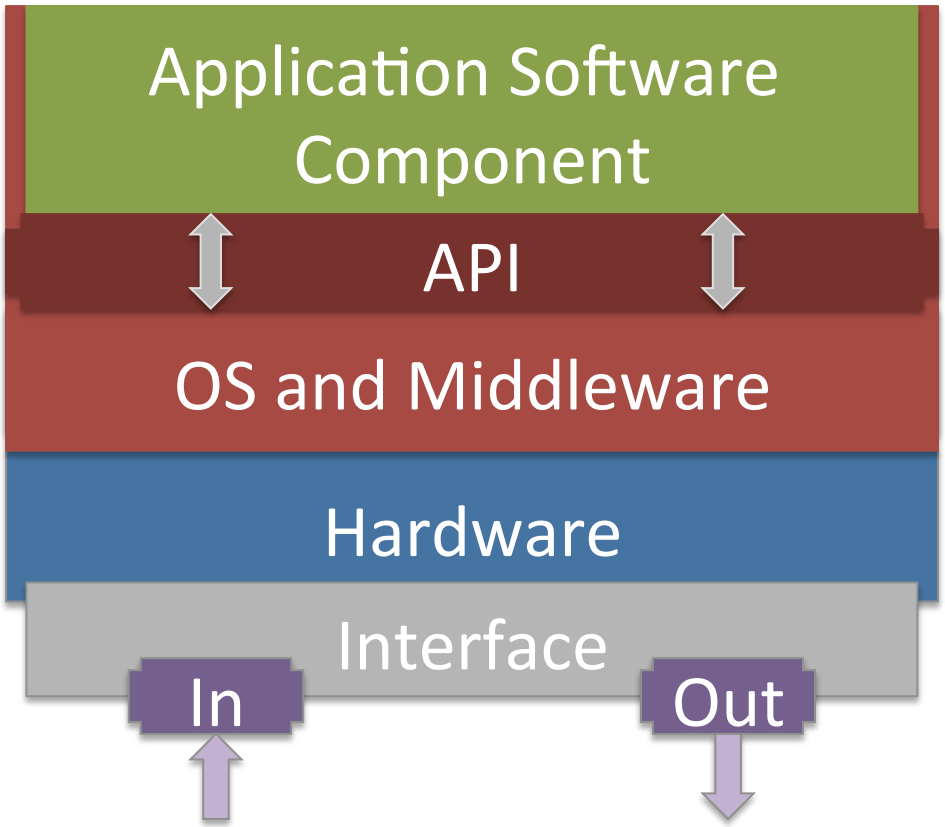
Real-Time Component

Interfaces have defined functionality and timing

RT component

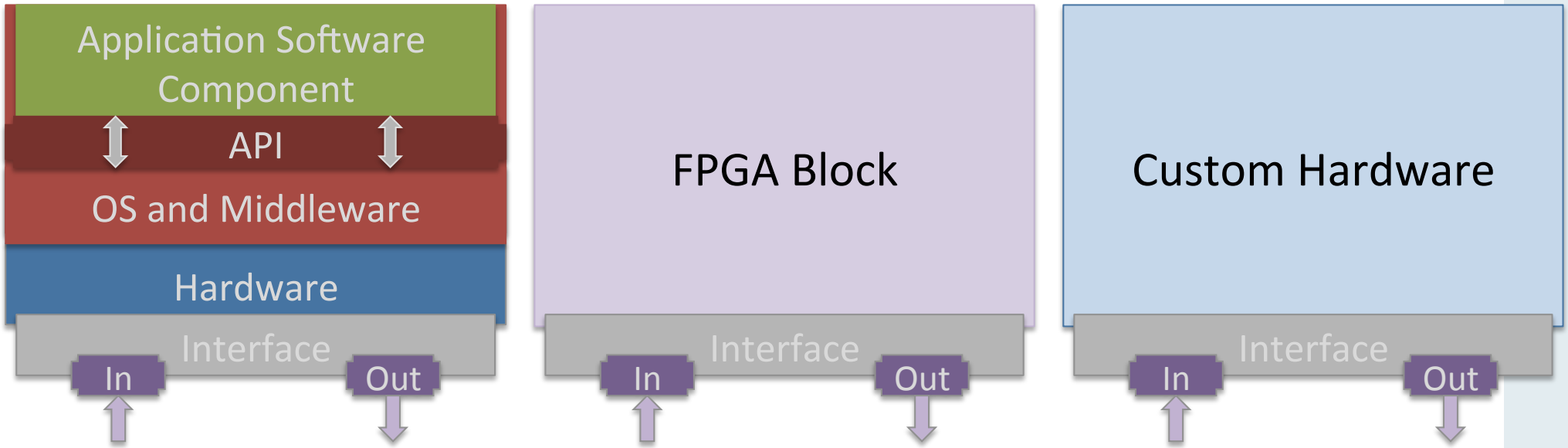
- is a complete computer system – a node / a core?
- is time-aware
- consists of
  - Hardware
  - Software (system and application software)
  - State

Real-Time Component



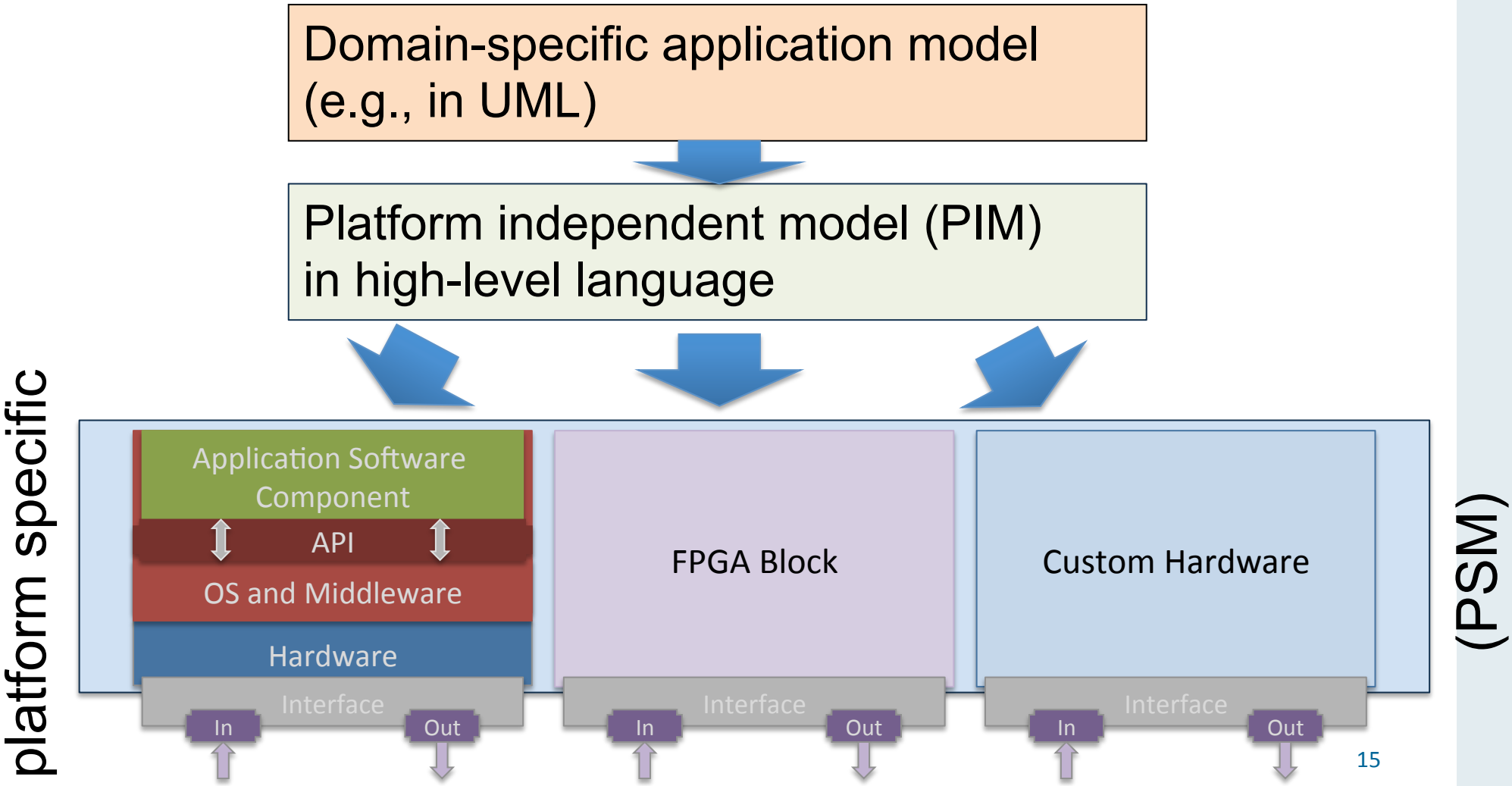
Software + Hardware!

Real-Time Component Realizations



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

Model Driven Design: from PIM to PSM



Component State

A hardware-software component consist of

- Hardware
- Operating System
- 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 the current and past computations (in RAM)

A system-wide consistent notion of time – sparse time – is necessary to build a consistent notion of 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 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, Abstract System Theory, p.45]

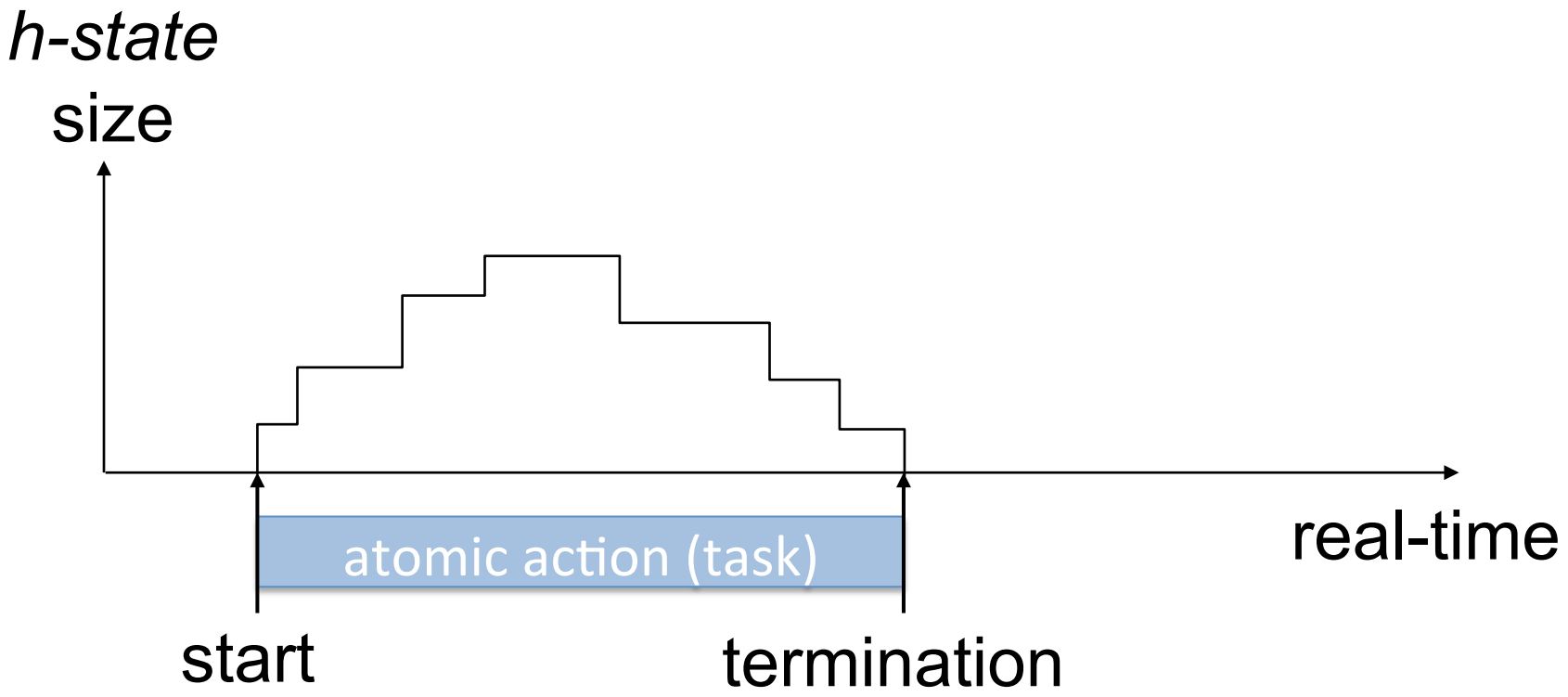
History State (H-State)

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

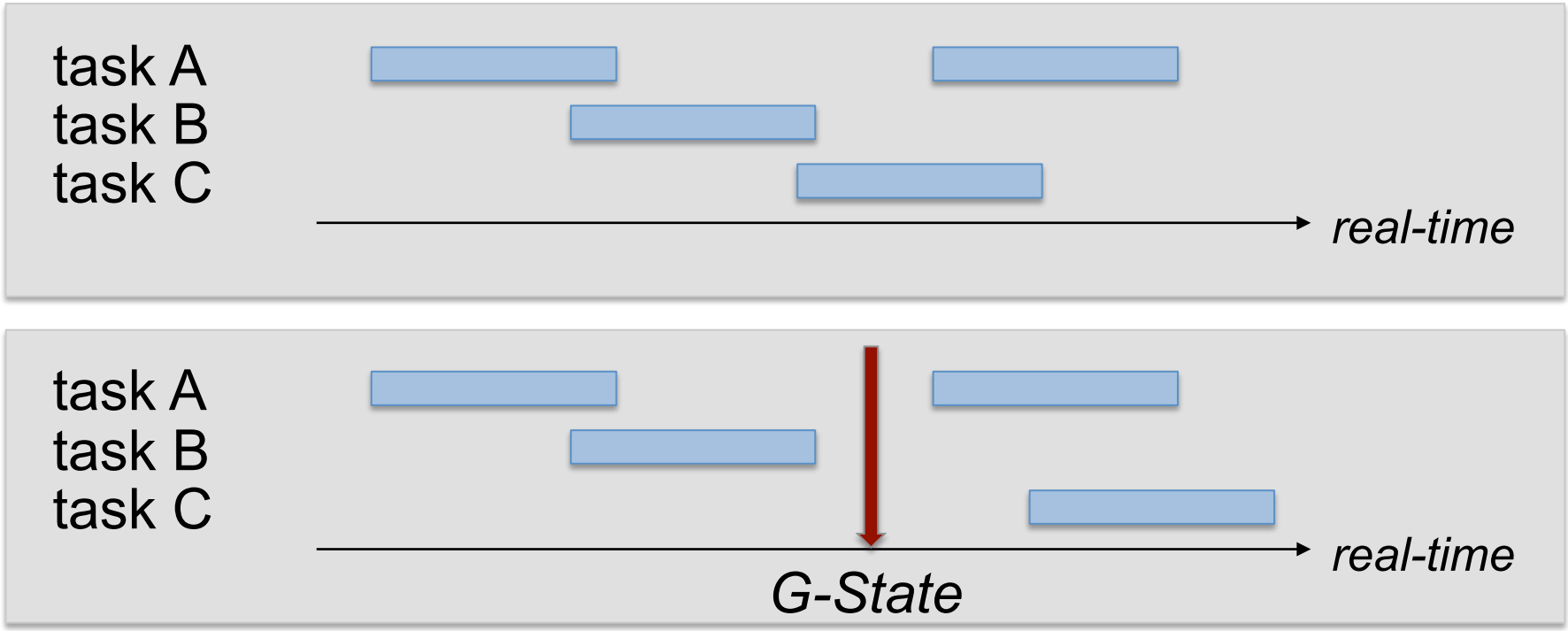
- Size of the *h-state* changes over time
- relative minimum immediately after a computation (an atomic action) has been completed.
- shall be small at reintegration points.
- *g-state* (ground state) of a system: minimal h-state, when all tasks are inactive and all channels are flushed (no messages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

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

H-State Size during Atomic Action



Ground State (G-State)



- Minimal h-state of subsystem
- Tasks are inactive, comm. channels are flushed

⇨ Re-integration point

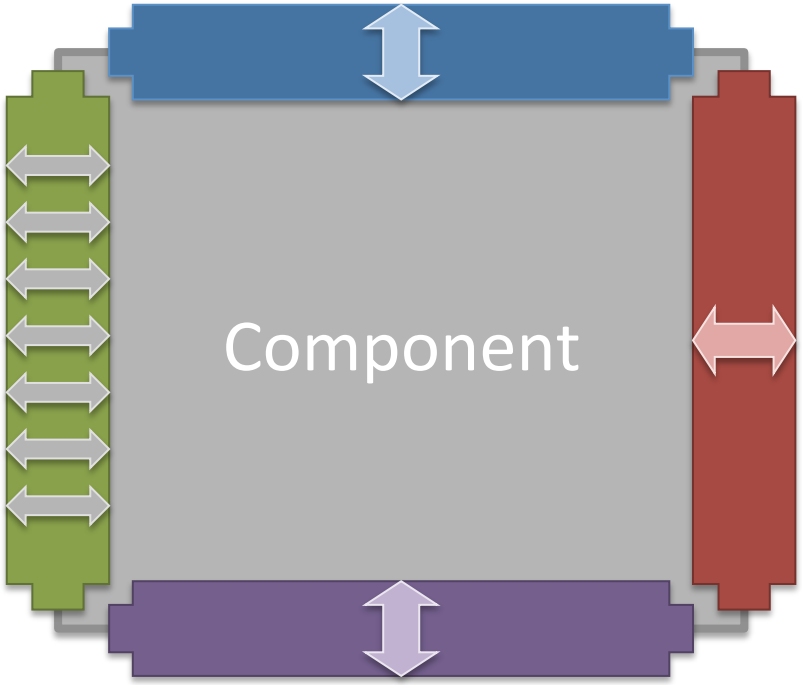
Interface

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

Diagnosis & Management
(boundary scan in HW design)



Local
Interfaces
(open component)

Linking
Interface – LIF
(Composability)

Configuration & Planning

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

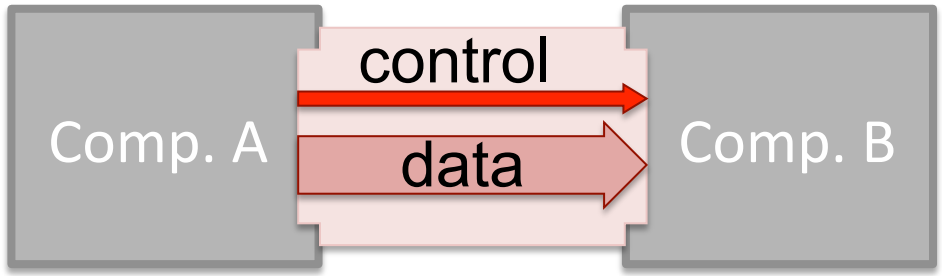
Component Communication via LIF

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

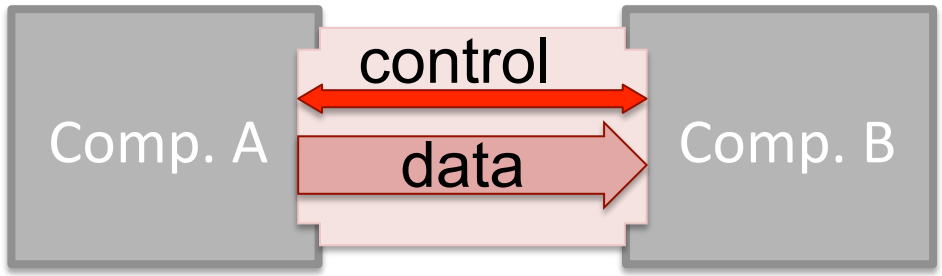
Elementary Interface



unidirectional control flow

Example:
Write to dual-ported RAM

Composite Interface



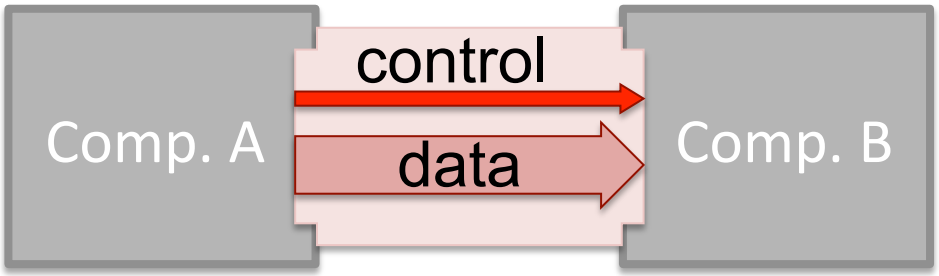
bidirectional control flow

e.g.,
queue of event messages

Elementary interfaces are simpler!

Information Push vs. Information Pull

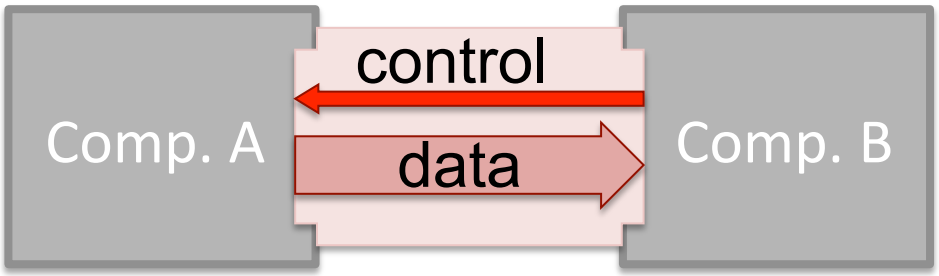
Information Push



Example: telephone, interrupt

Producer pushes information on consumer

Information Pull



Example: email

Consumer retrieves information when required

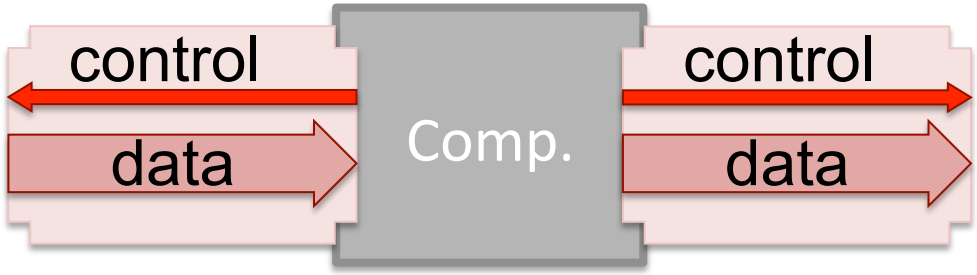
Temporal Firewall

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

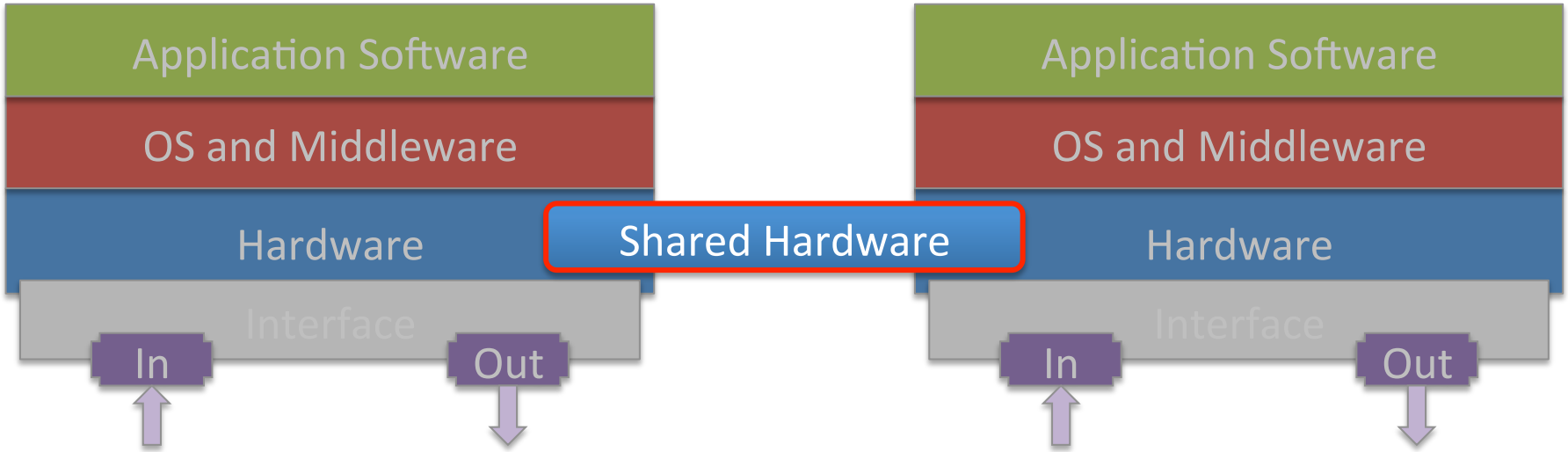
Open component

- Local real-world interface

Semi-open component

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

Shared Hardware on RT Component



Memory ⇨ Mutual Exclusion
Bus ⇨ Arbitration
Cache ⇨ Pollution

⇨ **Timing,** Error Containment

NO!

Component Interfaces

Real-time interface = message interface

Network communication

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

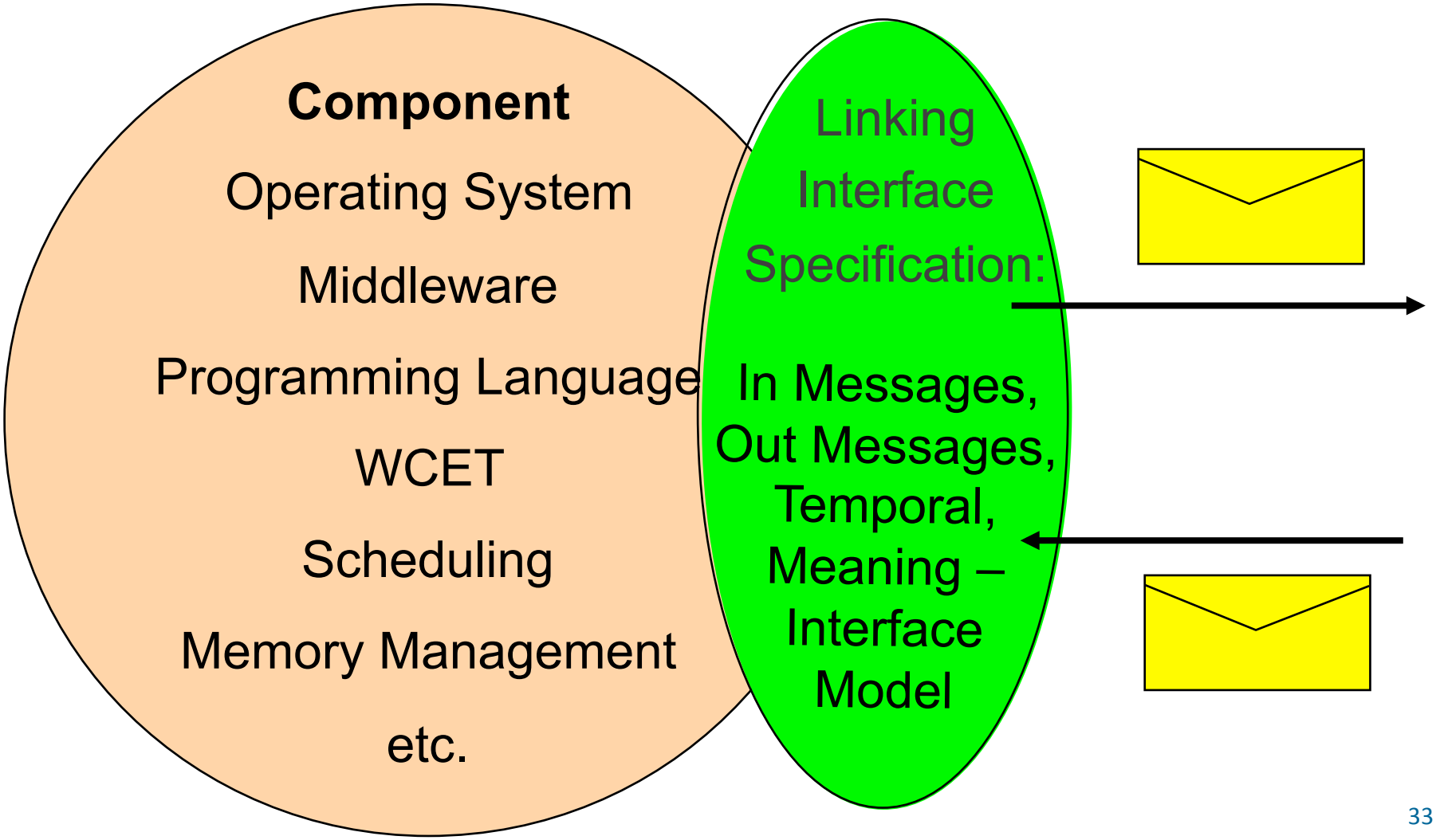
System Design = Message Specificaton

System Design
interactions among all components are specified

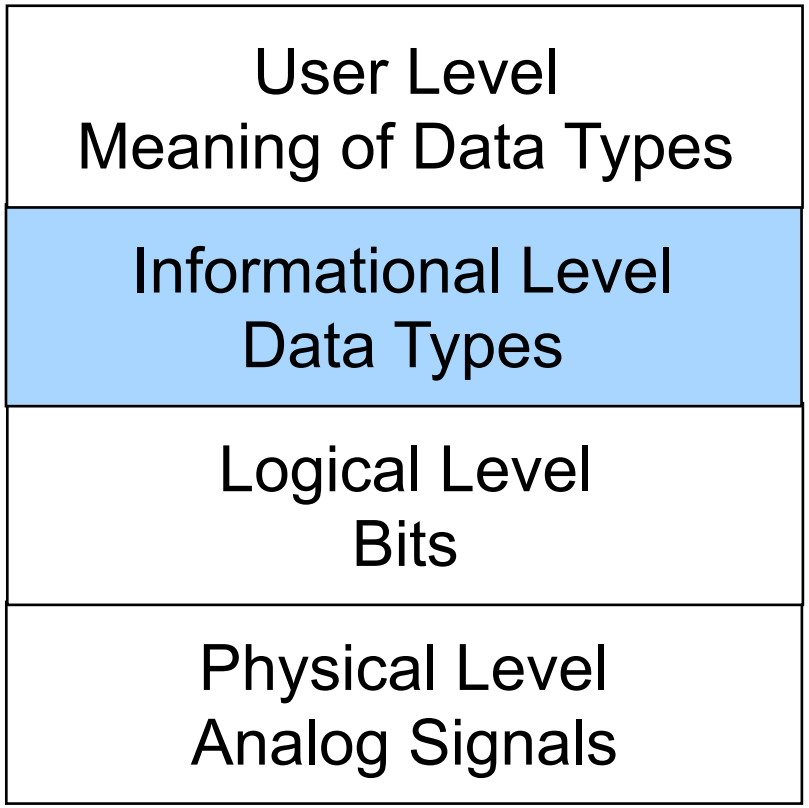
- Abstract message interface → message data structures
- Timing (phase, period) and control semantics of messages
- Response time of nodes
- Ground state of nodes

Subsequent component design is constrained by the message specifications

LIF Specification Hides Implementation



System Views: Four-Universe Model



Meta-level Specification
Interpretation by the User

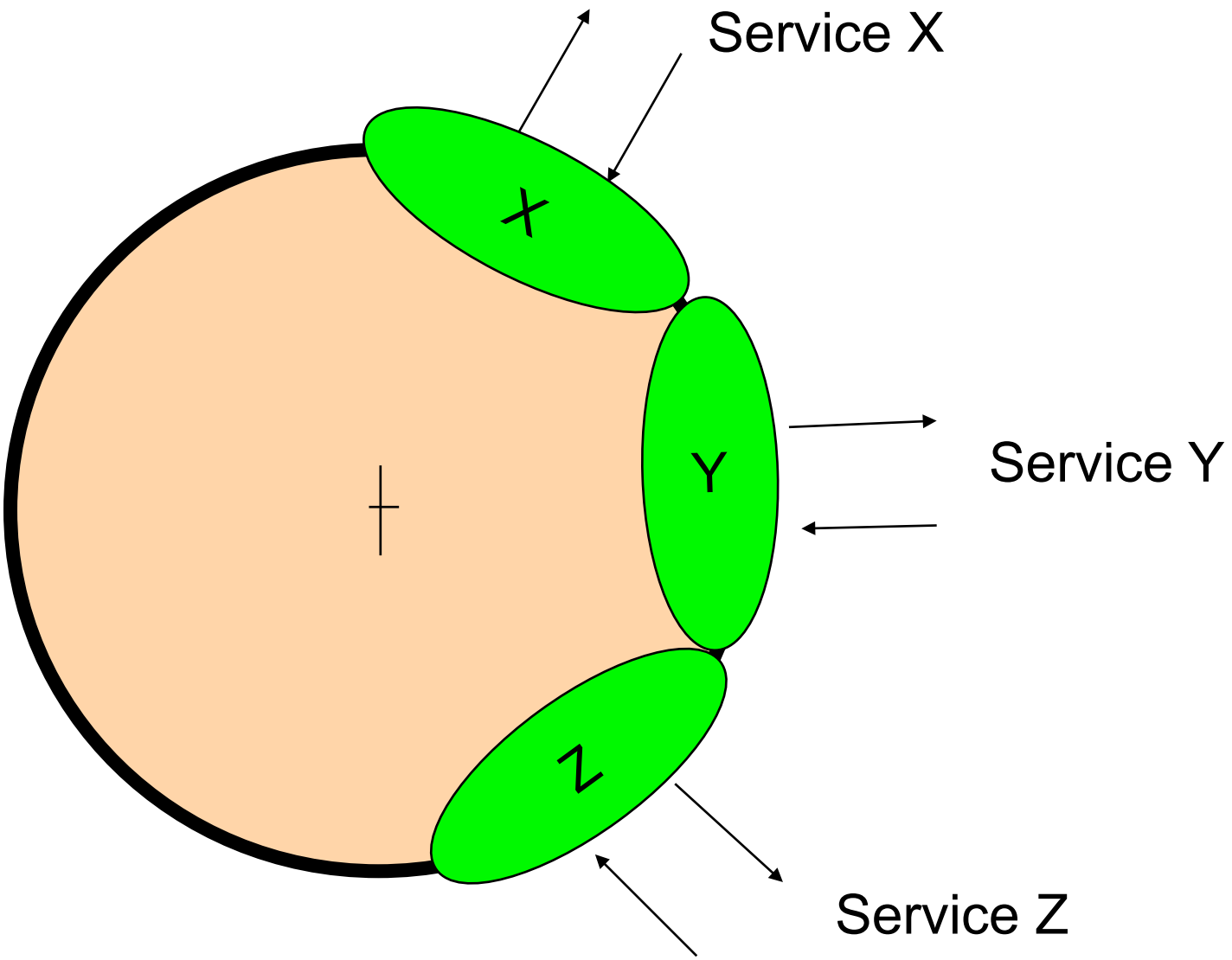
Operational Interface Specification,
Value, Temporal, Control Properties

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



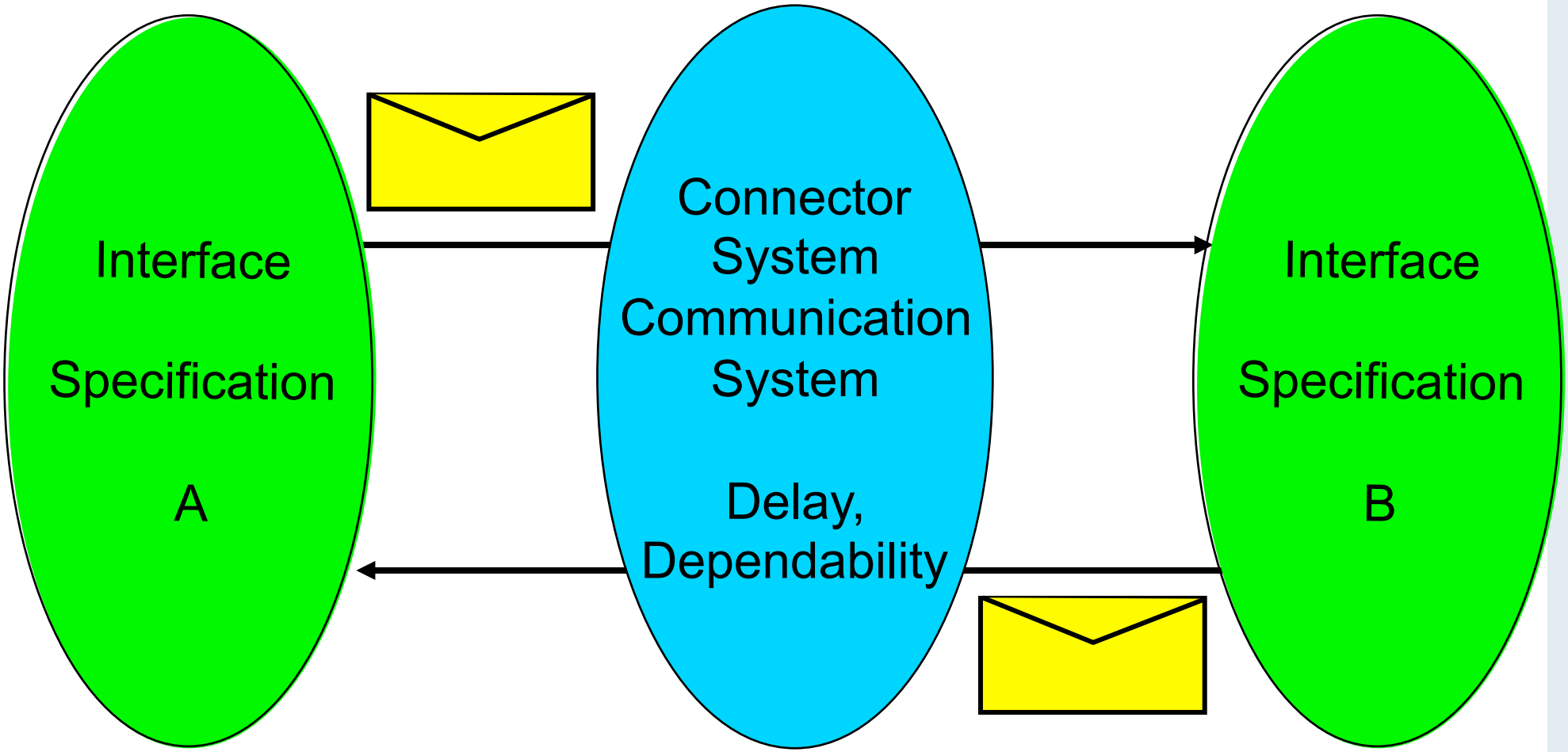
Interfaces – Property Mismatches

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

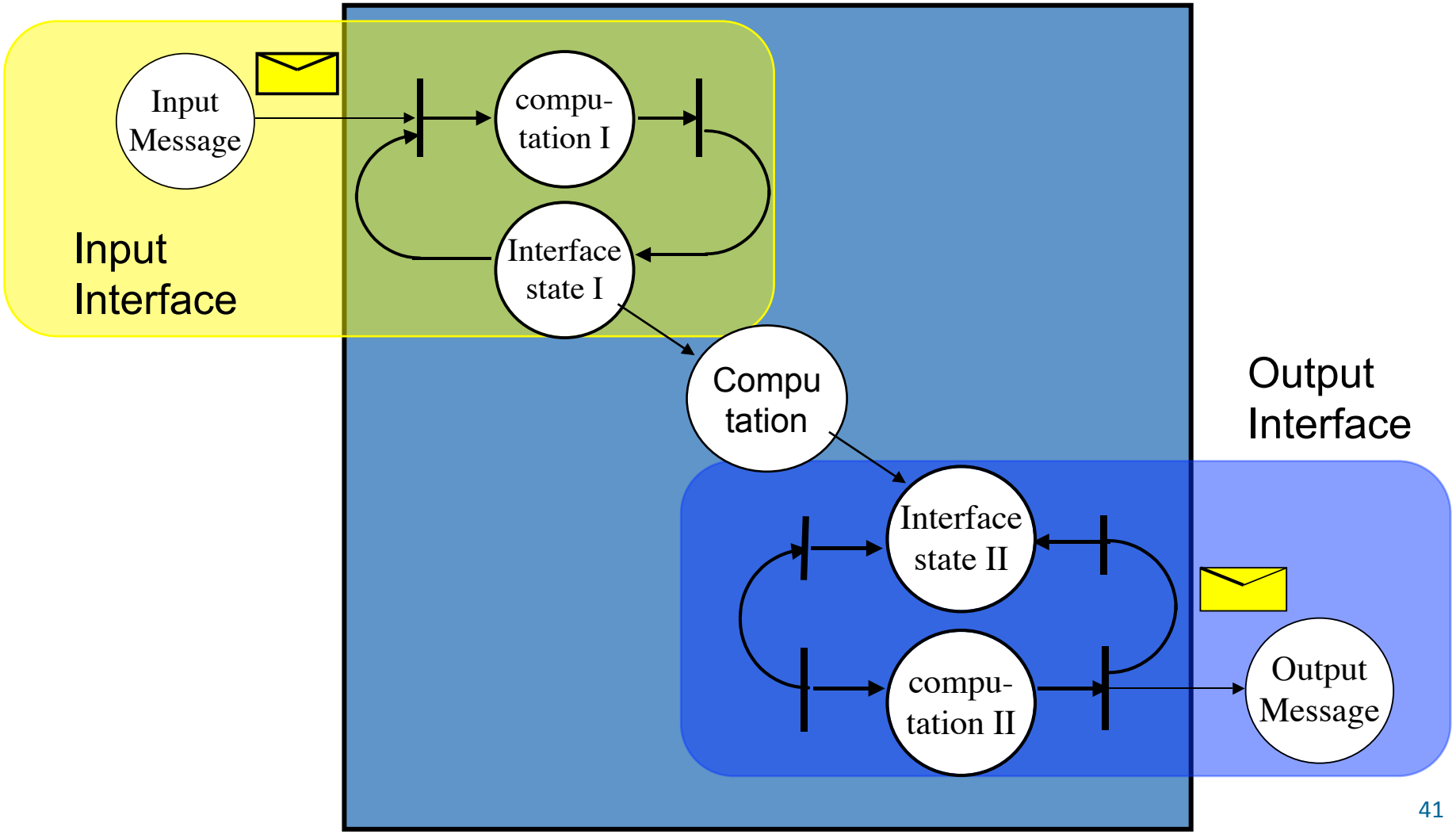
Input Interface:

- 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

Example: Text-to-Speech (2)



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

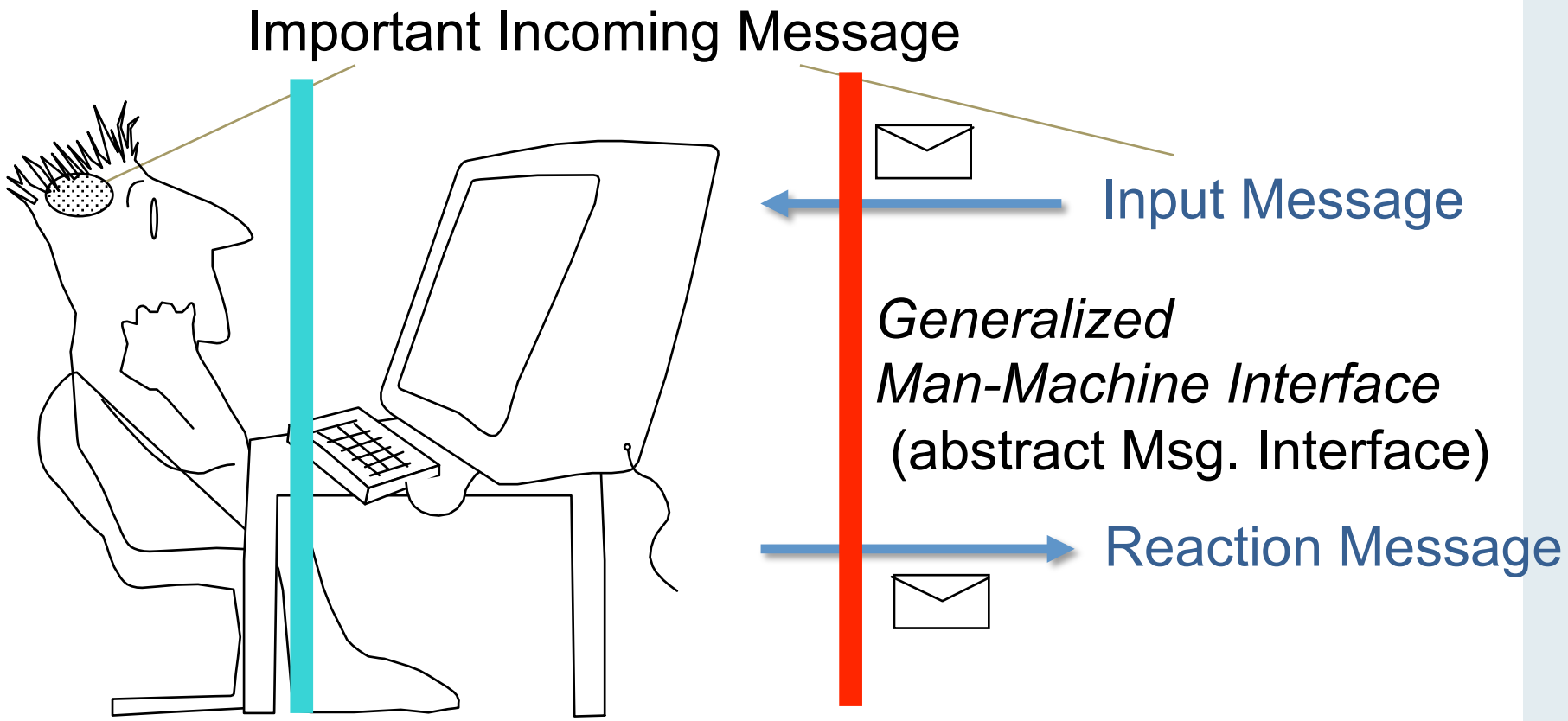
Message interface: internal, abstract message-based interface of a cluster

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

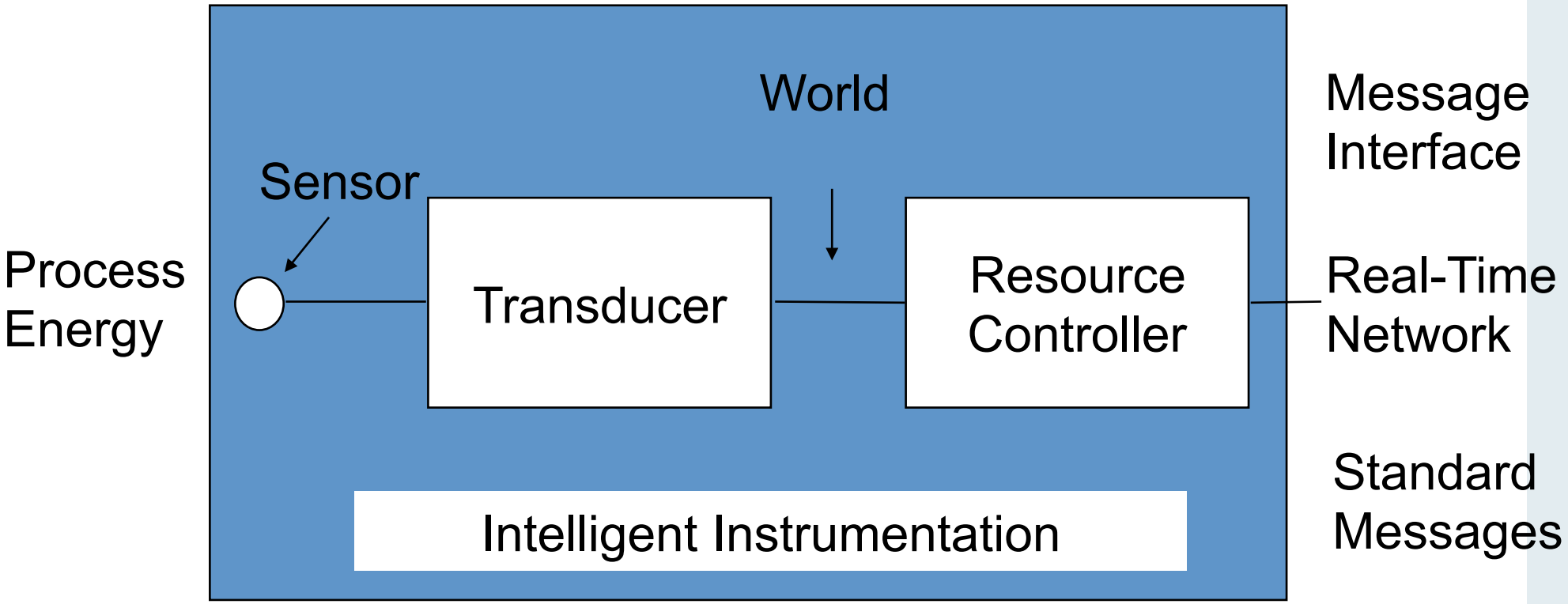
[Transducer (Webster): device that receives energy from one system, and retransmits it, often in a different form, to another].

Example: Text-to-Speech (3)

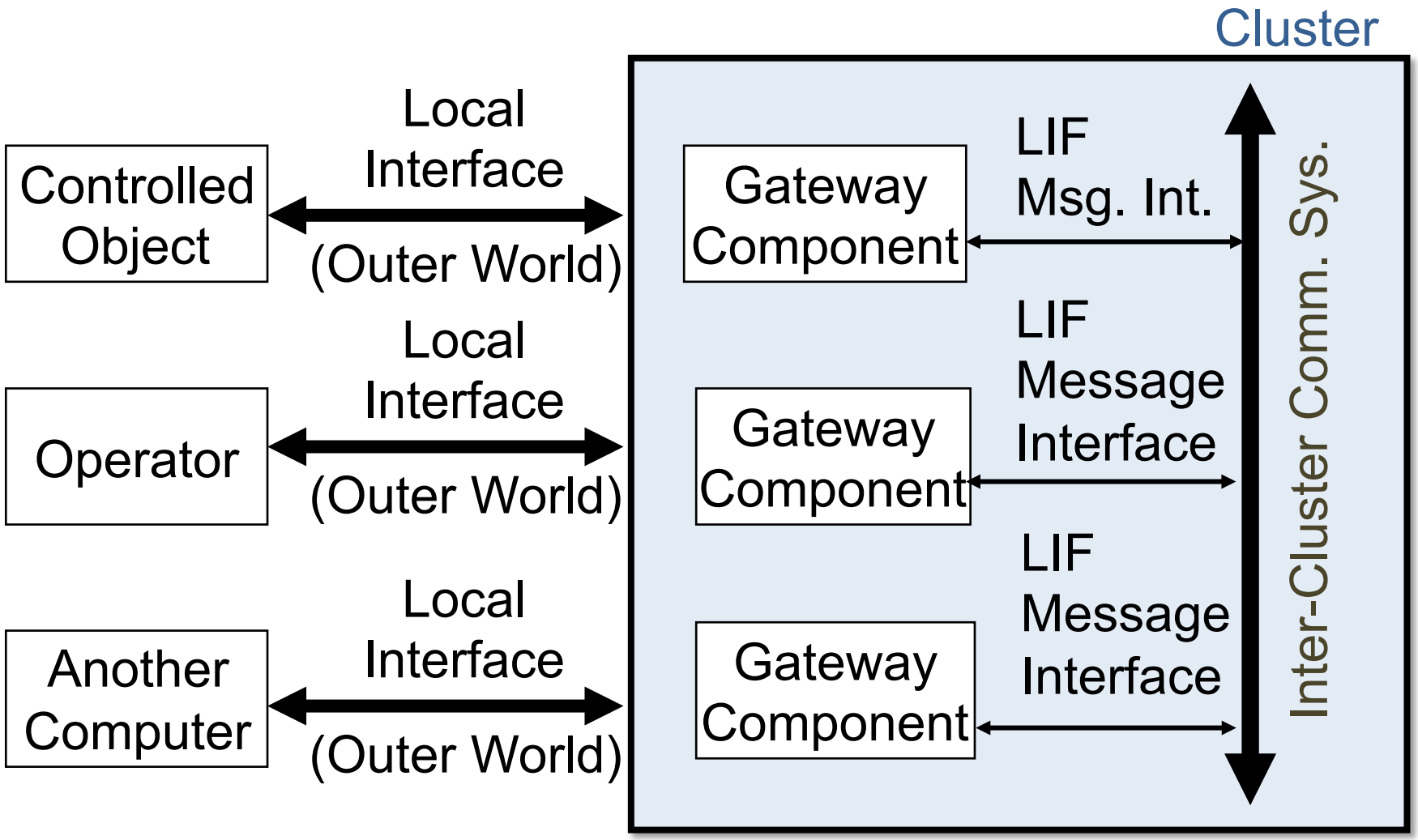


Specific Man-Machine Interface (Graphics, Sound)
(concrete World Interface)

Example: Intelligent Instrumentation



World vs. Message Interface



World/Message Interface Characteristics

Characteristic	World Interface	Message Interface
Info. Representation	unique	standardized
Coupling	tight	weaker
Coding	ana./digital	digital
Time Base	dense	sparse
Responsiveness	tight	weaker
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

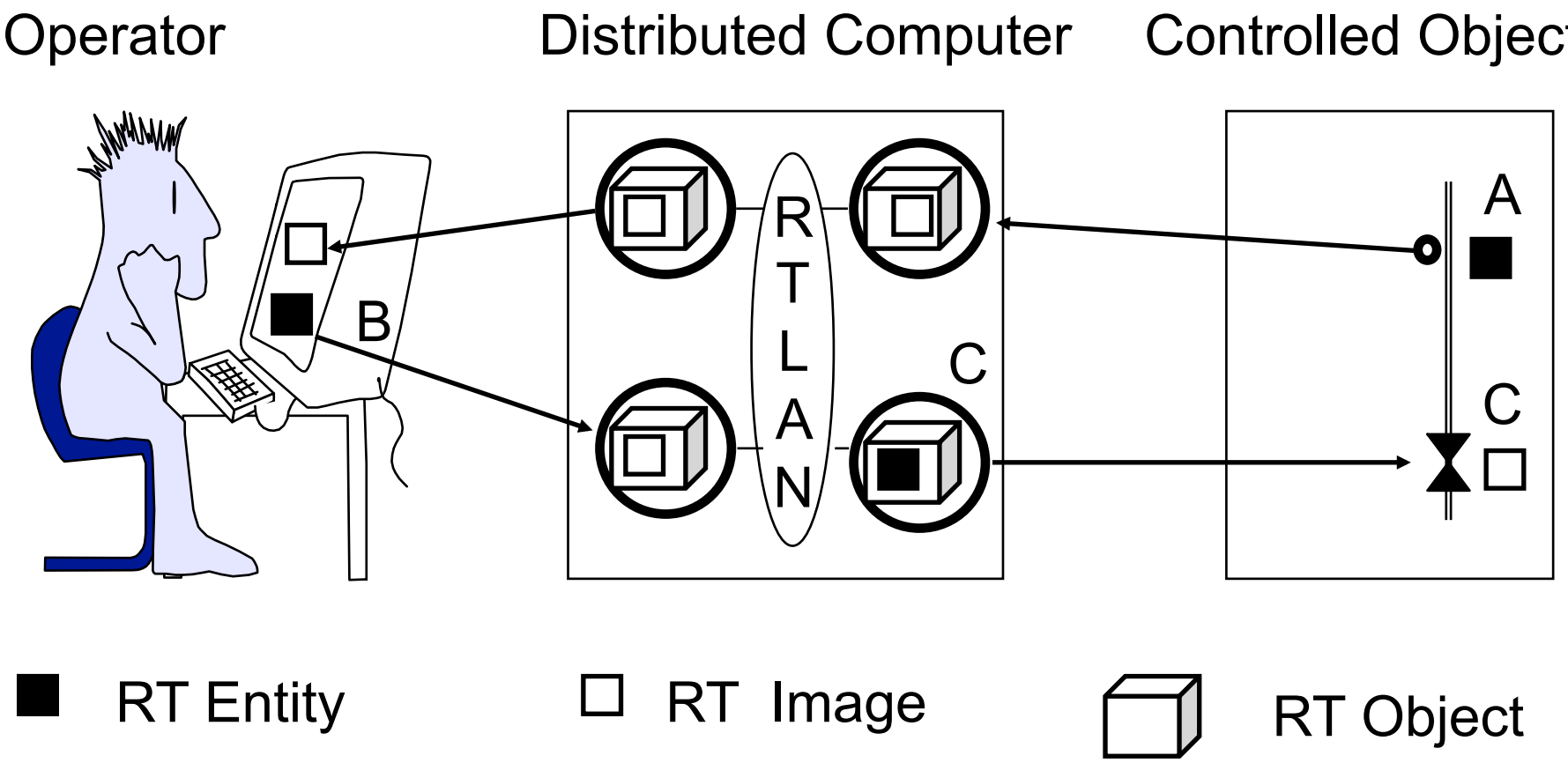
Message Classification

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

RT Entities, RT Images, and RT Objects

This is not a pipe.

RT Entities, RT Images, and RT Objects



■ RT Entity □ RT Image RT Object

A: Value of Flow being measured

B: Setpoint for Flow C: Intended Valve Position

Real-time (RT) Entity

Real-Time (RT) Entity:

- state variable/property of interest
- for a given purpose
- changes its state as a function of real-time
- may be *continuous* or *discrete*

Examples of RT Entities:

- Flow in a Pipe (Continuous)
- Position of a Switch (Discrete)
- Setpoint selected by an Operator
- Intended Position of an Actuator

Attributes of RT-Entities

Static attributes

- Name (meaning)
- Type
- Value Domain
- Maximum Rate of Change

Dynamic attributes

- Actual value at a particular point in time

Sphere of Control

Every RT-Entity is in the Sphere of Control (SOC) of a subsystem that has the authority to set the value of the RT-entity:

RT-Entity	SOC
Setpoint	Operator
Actual flow	Control object
Intended valve position	Computer

Outside its SOC an RT-entity can only be observed, but not modified.

Observation

Captures information about the state of an RT-entity

Observation = <*Name*, *Time*, *Value*>

Name: name of the RT-entity

Time: point in real-time when the observation was made

Value: value of the RT-entity

Observations are transported in messages.

Observations, States, and Events

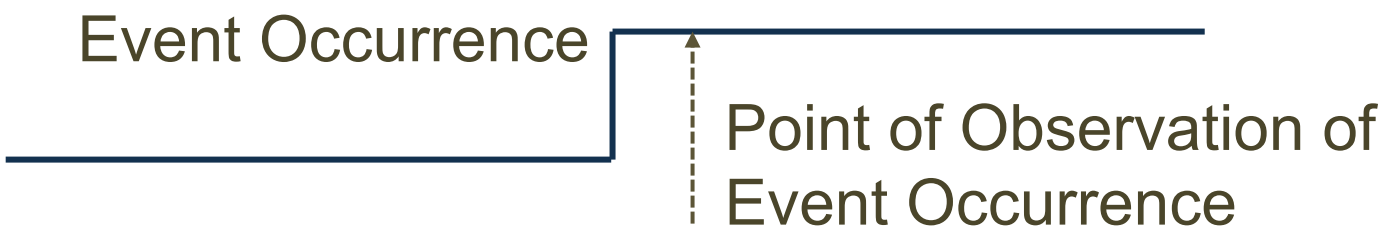
State … section of the timeline.

Event … cut of the timeline.

➪ every change of state is an event.

Observations

- States can be observed
- An event cannot be observed
  - ➪ only the new state can be observed.



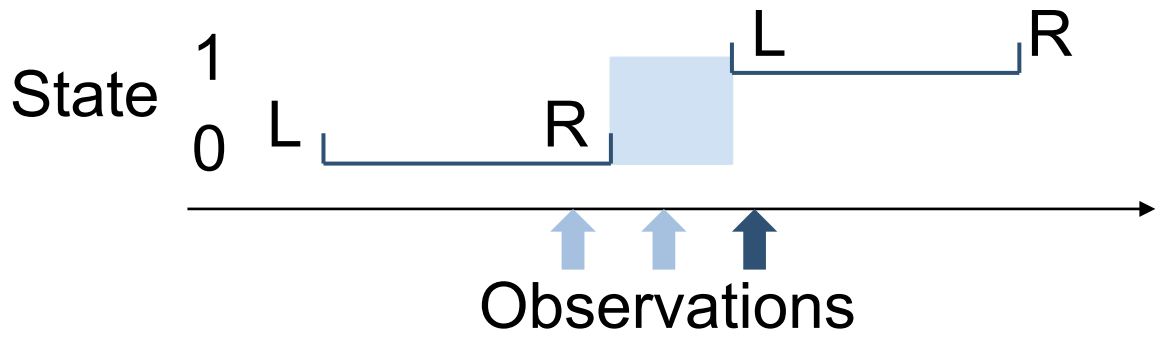
State and Event Observation

State observation

- Value contains the full or partial state of the RT-entity.
- Observation time: point in time when the RT-entity was sampled.

Event observation

- Value characterizes difference between the “old state” (the last observed state) and “new state”.
- Observation time: point in time of the L-event of the “new state”.



RT-Image

RT-Image

- picture of an RT-entity,
- valid at a given point in time, if it is an accurate representation of the corresponding RT-entity, in value and time.

RT-Image

- is only valid during a specified interval of real time,
- can be based on an observation or on state estimation,
- can be stored in data objects, either inside a computer (in an RT-object) or outside, in an actuator.

RT-Object

RT-object

- “container” for RT-image or RT-entity in the computer system.

An RT-object *k*

- has an associated real-time clock that ticks with granularity t_k. t_k must be in agreement with the dynamics of the RT-entity the object represents,
- activates an object procedure upon occurrence of defined events, e.g., when time reaches a preset value.
- If there is no other way to activate an object procedure than by the periodic clock tick, we call the RT-object *synchronous.*

Distributed RT-Object

Distributed RT-object

- set of replicated RT-objects located at different sites.
- every local instance of a distributed RT-object provides a specified service to its local site.

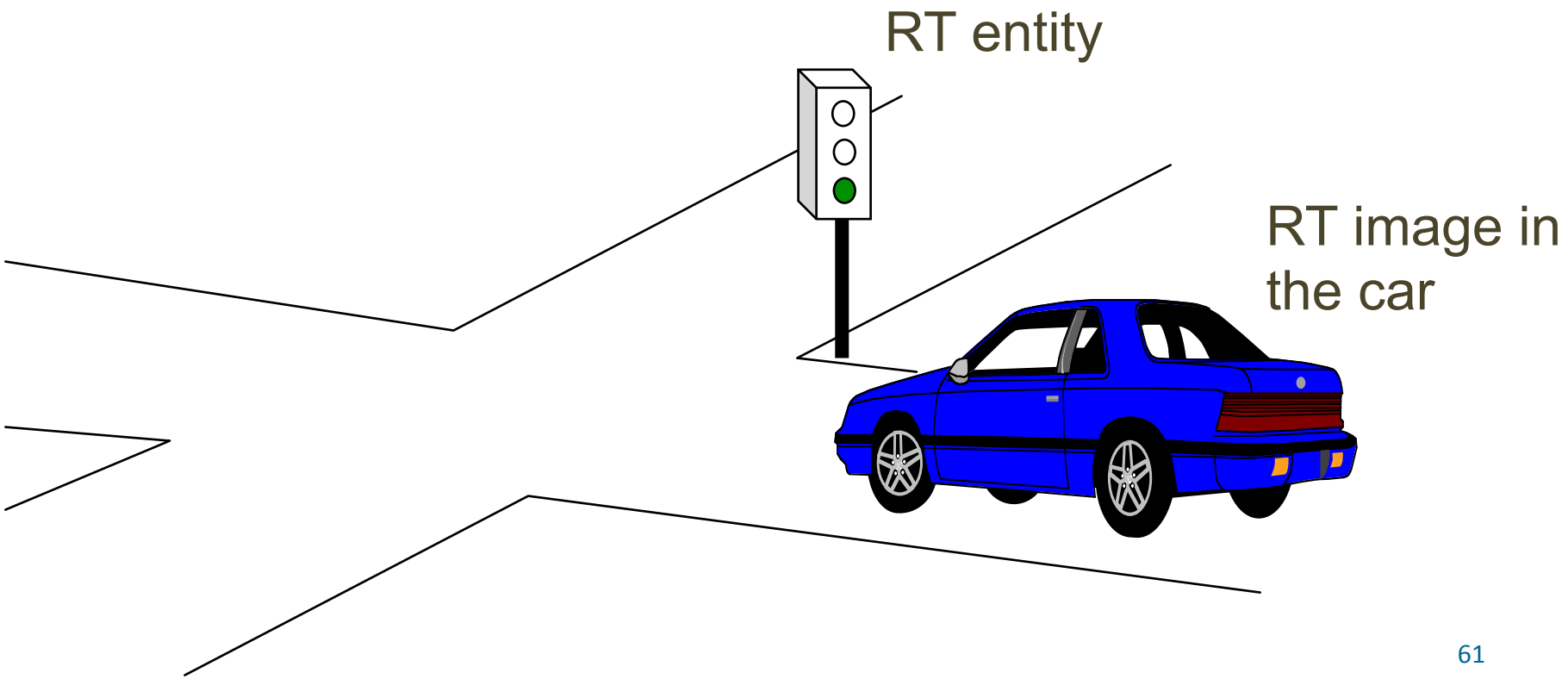
The *quality of service* of a distributed RT-object must be in conformance with some specified consistency constraint.

Examples:

- Clock synchronization within a specified precision
- Membership service with a specified delay

Temporal Accuracy of RT-Information

How long is the RT-image, based on the observation *"The traffic light green"*, temporally accurate?



Temporal Accuracy

Recent history RH_i at time t_i:

- Ordered set of time points $<t_{i-k}, \ldots, t_{i-1}, t_i>$

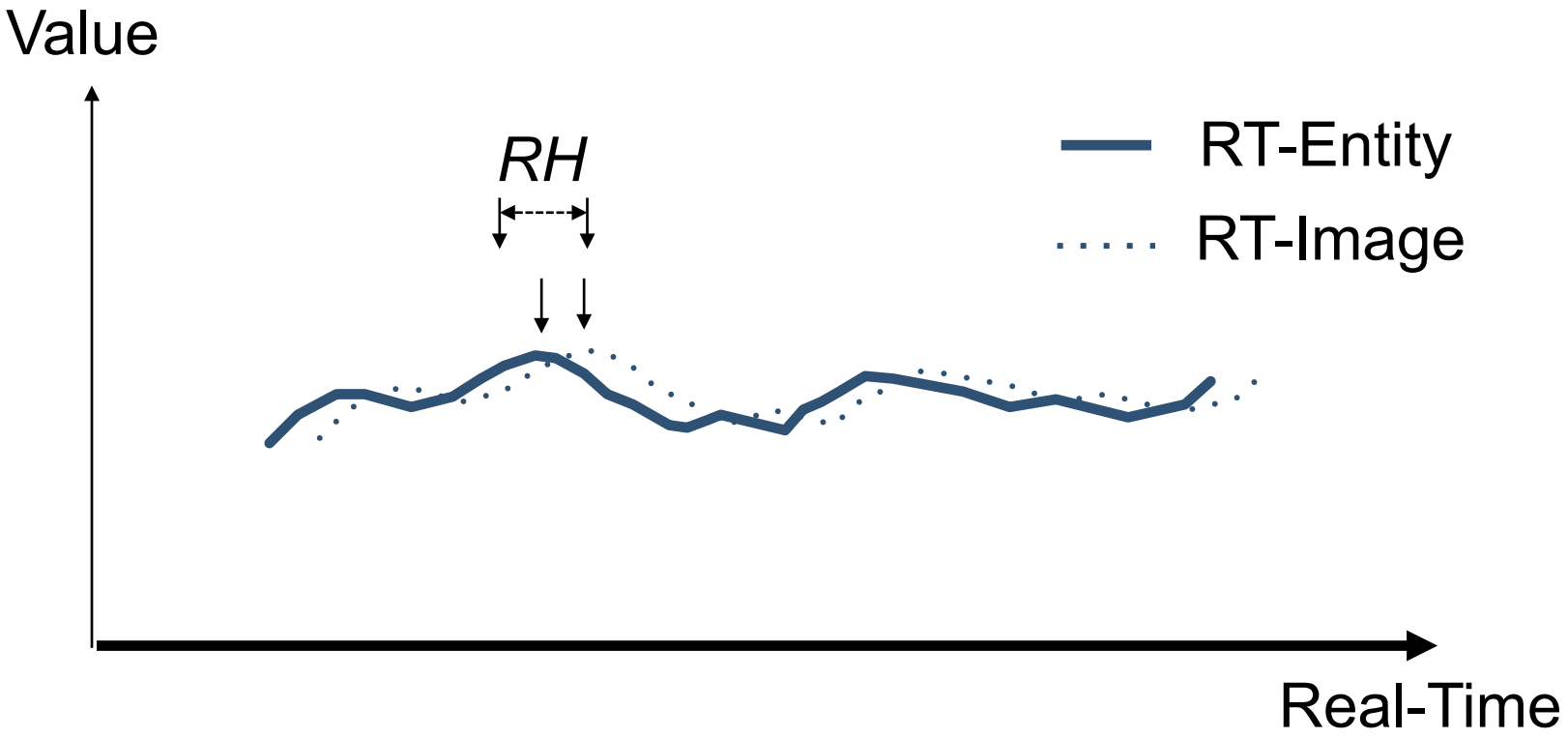
Temporal accuracy d_{acc}:

- Length of the recent history, $d_{acc} = t_i - t_{i-k}$

Assume that the RT-entity has been observed at every time point of the recent history.
The RT-image is temporally accurate at the present time t_i if

$$\exists\, t_j \in RH_i : Value\,(RT\ image\ at\ t_i) = Value\,(RT\ entity\ \ at\ t_j)$$

Temporal Accuracy of RT-Objects



For an RT-object, updated by observations, there will always be a delay between the state of the RT-entity and that of the RT-object.

Temporal Accuracy and RT-Image Error

The delay between observation (at t_{obs}) and use (at t_{use}) of the value v of an RT-entity causes an error of the RT-image:

$$error(v, t_{obs}, t_{use}) = v(t_{use}) - v(t_{obs}).$$

Approximation of worst-case error at the time of use of a temporally valid RT-image:

$$error_{max}(v) = \max (dv(t)/dt)\ d_{acc}$$

$error_{max}$ should be in the same order of magnitude as the worst-case measurement error in the value domain.

d_{acc} is therefore determined by the dynamics of the RT entity in the controlled object.

Example: Combustion Engine

The ignition time is a function of the following parameters:

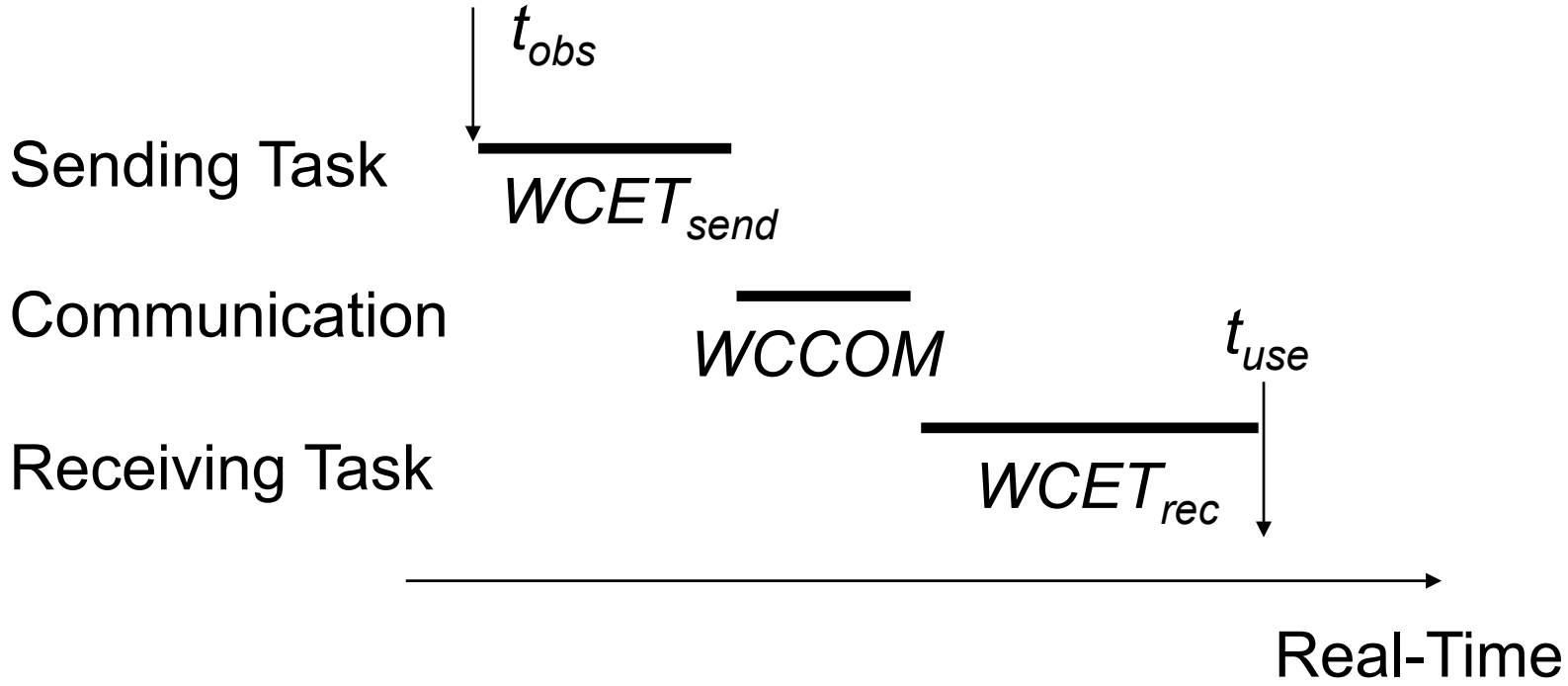
Parameter	Recent History Accuracy(sec)
crank position	10^{-6}
h-state	10^{-3}
gas pedal position	10^{-2}
load	1
temperature	10^{+1}

There are seven orders of magnitude difference in the required temporal accuracy of the parameters.

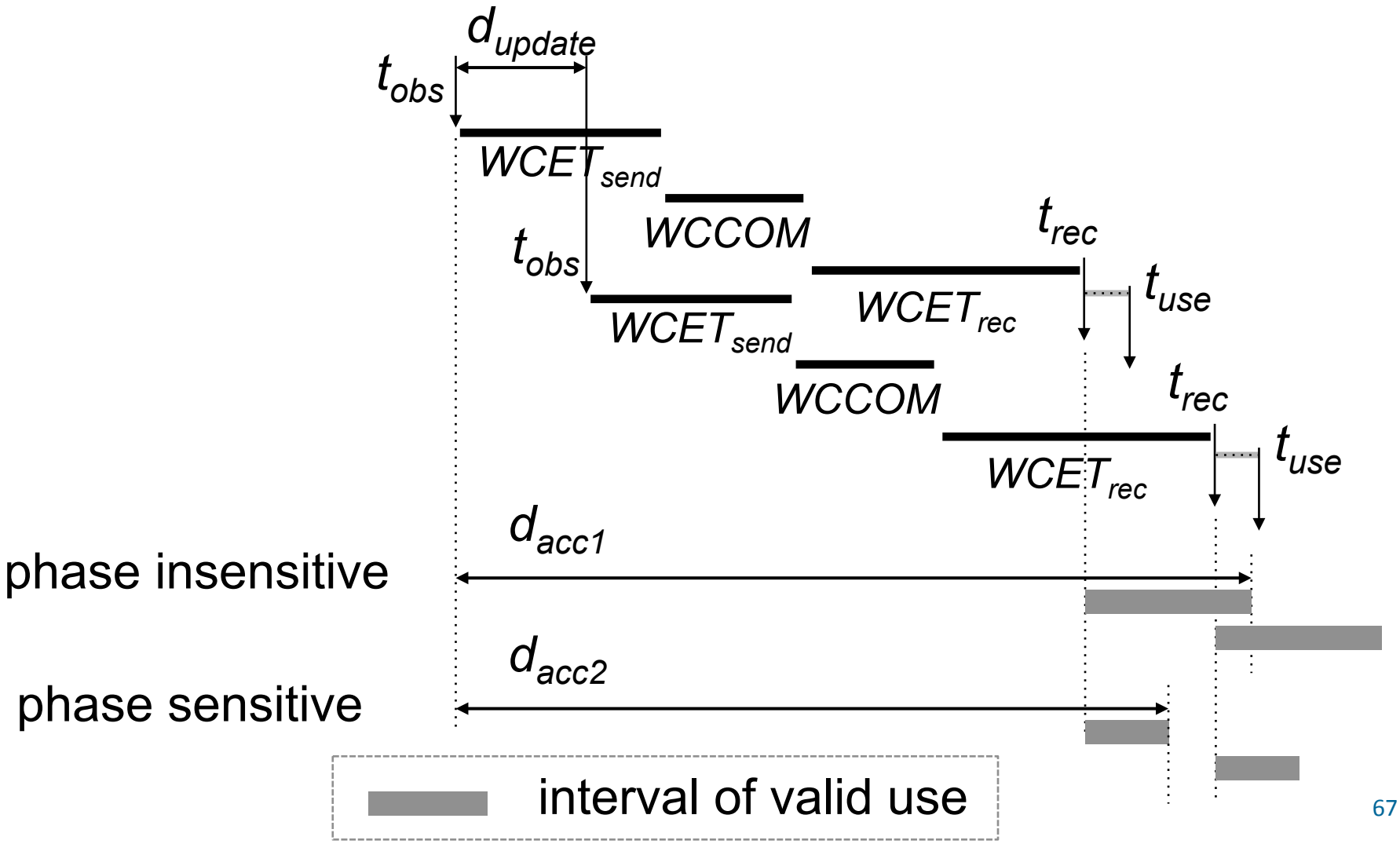
Synchronized Actions

If an RT entity changes its value quickly, d_{acc} must be short.

Phase-aligned transactions to guarantee: $t_{use} - t_{obs} \leq d_{acc}$



Phase Sensitivity of RT-Images



Phase Sensitivity of RT-Images

Assume a periodic update of an RT image with period d_{update}.

An RT-image is *parametric* or *phase insensitive* if:

$$d_{acc} > d_{update} + WCET_{send} + WCCOM + WCET_{rec}$$

An RT-image is *phase sensitive* if:

$$d_{acc} \leq d_{update} + WCET_{send} + WCCOM + WCET_{rec}$$
$$\text{and } d_{acc} > WCET_{send} + WCCOM + WCET_{rec}$$

State Estimation

A good accuracy of an RT-object can be obtained either by

- frequent sampling of the RT-entity or
- by state estimation.

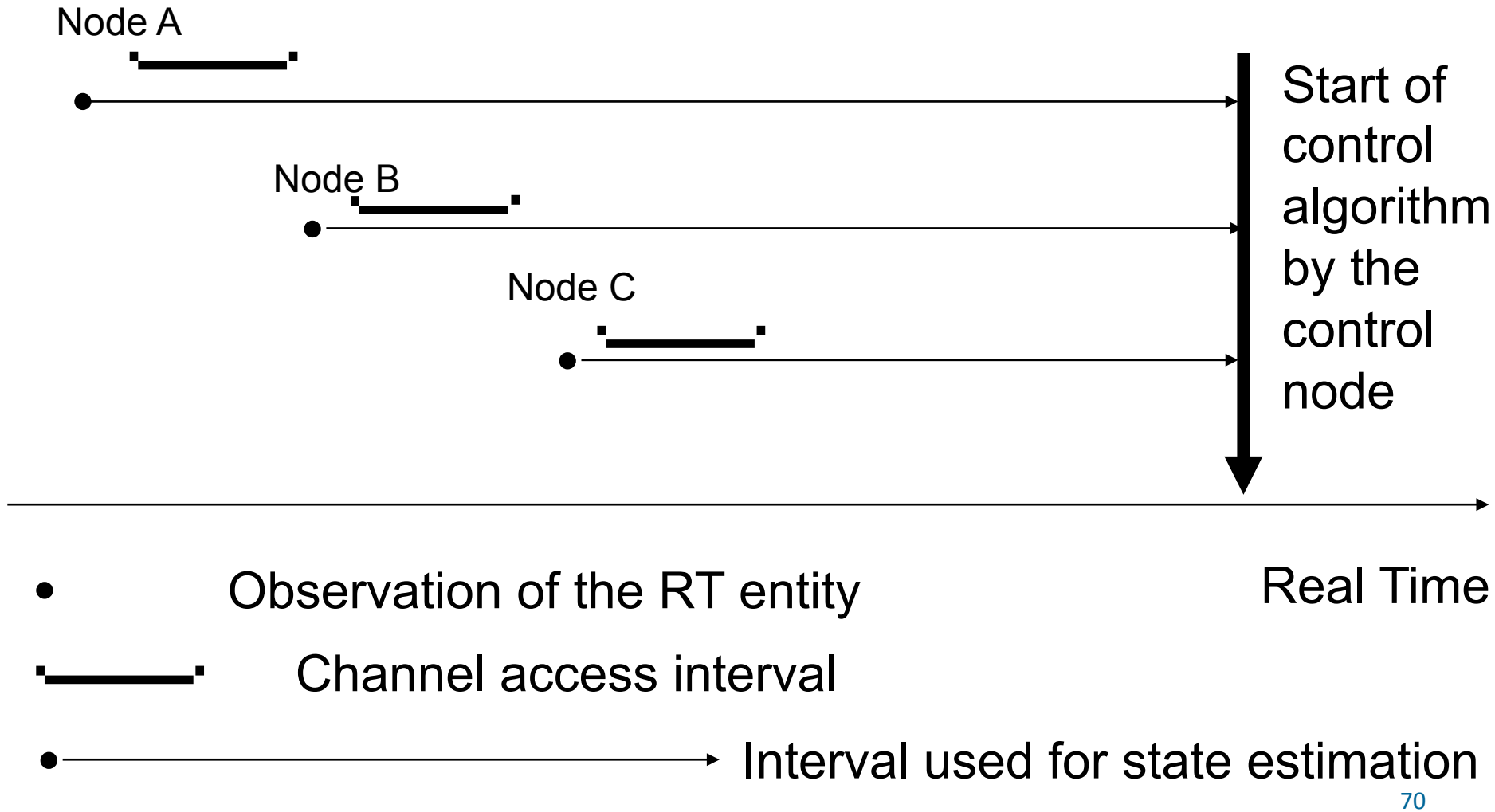
State estimation

- estimation of the current state of the RT-entity,
- periodically calculated within an RT-object,
- based on computational model of the dynamics of RT-entity.

$$v(t_{use}) \approx v(t_{obs}) + (t_{use} - t_{obs})\, dv/dt\,(t_{obs})$$

Often tradeoff possibility: computational vs. comm. resources.

State Estimation of Sensor Observations



Latency Jitter at Sender

Knowledge about latency at sender improves control quality
➪ receiver can use known latency for state estimation.

Approaches

- Latency guarantee: sender guarantees latency between point of sampling and point of transmission.
- Timed messages: sender transmits messages that contain the interval between observation and transmission.

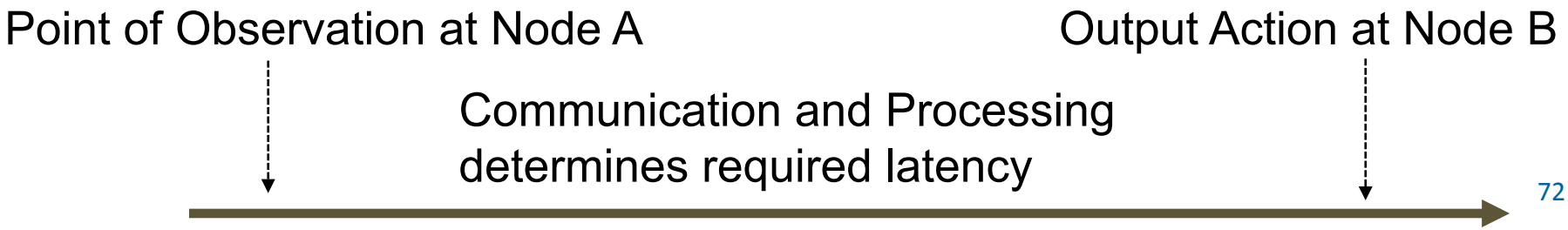
Timing Requirements for State Estimation

To compensate for the delay, a state estimation program needs

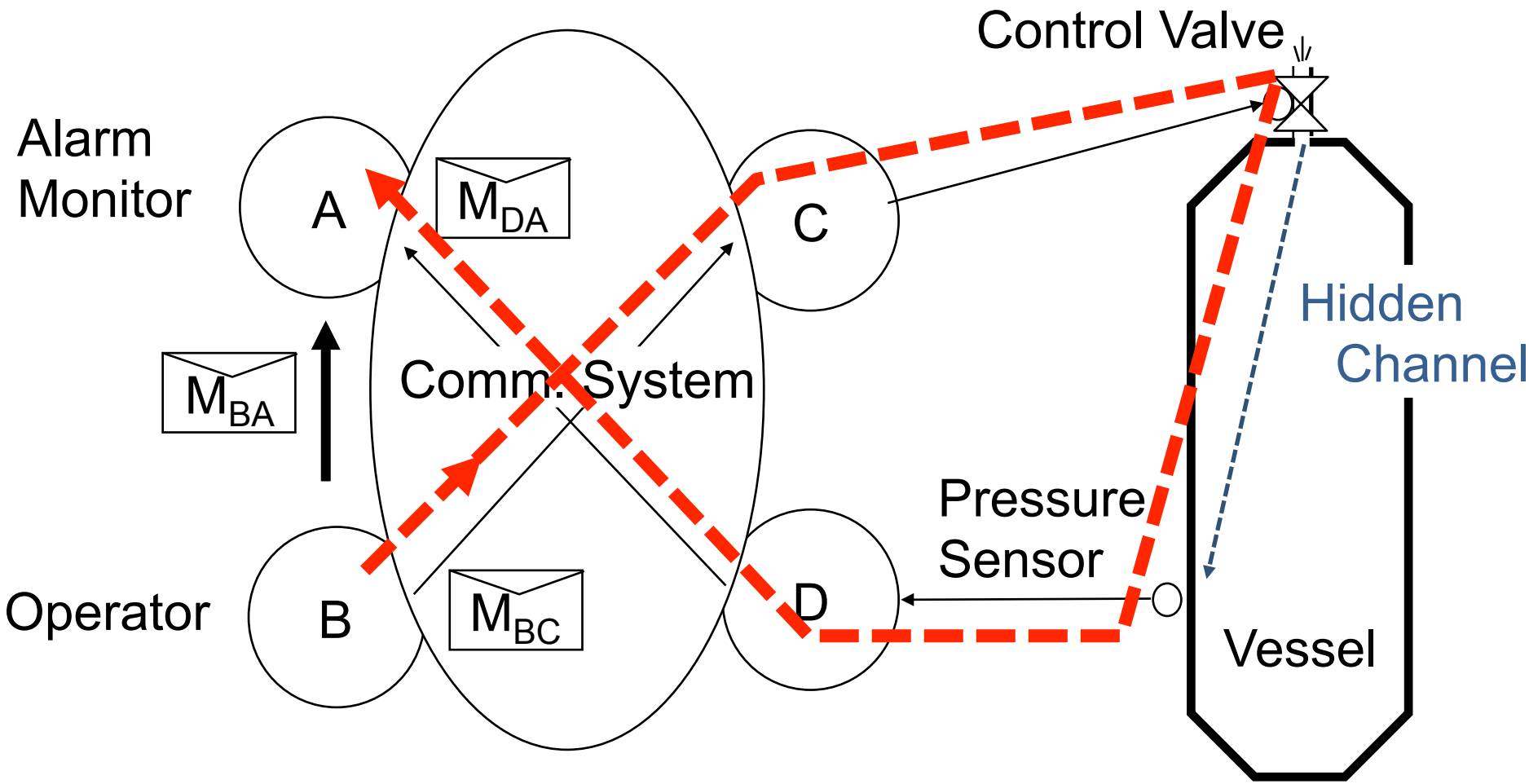
- the time of observation of an RT-entity,
- the planned time of actuation.

The quality of state estimation depends on the

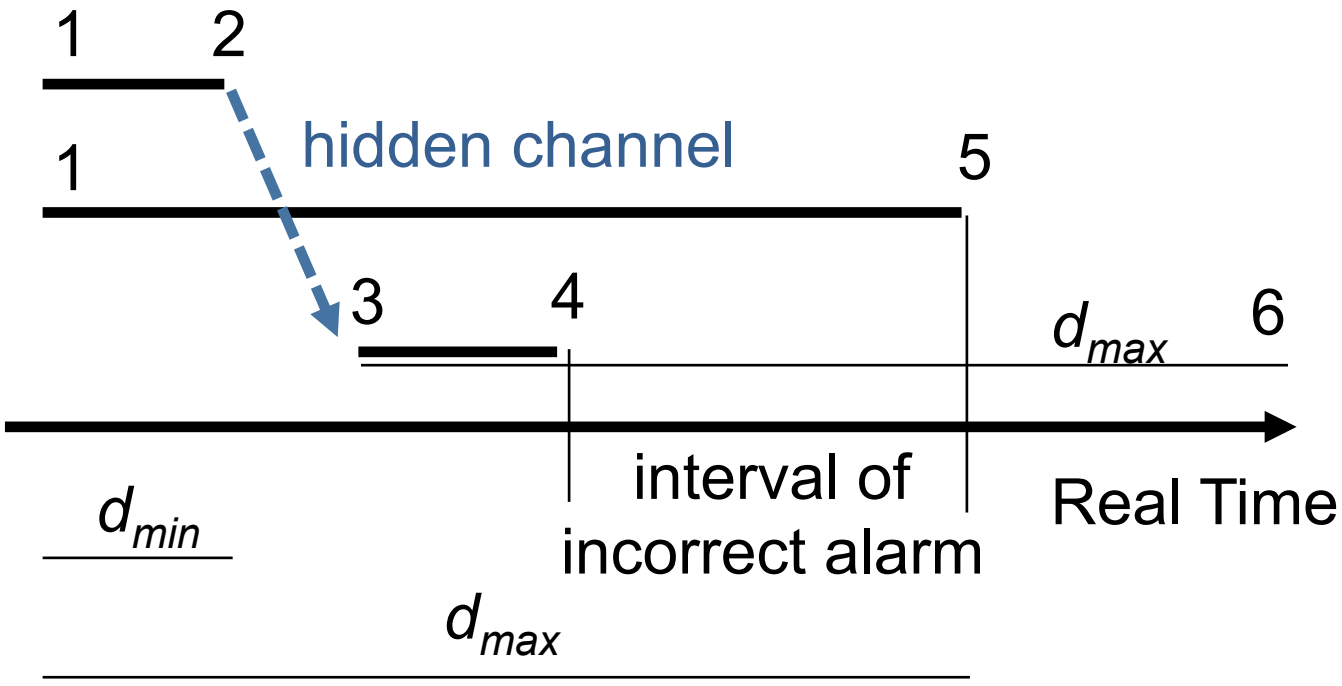
- Precision of the clock synchronization,
- Size of latency and quality of latency measurement,
- Quality of state-estimation model.



Hidden Channel



Hidden Channel (2)



1 Sending of M_{BC}

Sending of M_{BA}

2 Arrival of M_{BC}

3 Sending of M_{DA}

4 Arrival of M_{DA}

5 Arrival of M_{BA}

6 Permanence of M_{DA}

Permanence

A message M_i becomes *permanent* at object *O* as soon as all messages M_{i-1}, M_{i-2}, … that have been sent to *O* before M_i (in temporal order) have arrived at *O*.

Actions taken on non-permanent messages may cause errors or inconsistencies!

Action delay:

Interval between the point in time when a message is sent by the sender and the point in time when the receiver knows that the message is permanent.

Action Delay

Distributed RT systems without global time base:

maximum action delay: $d_{max} + \varepsilon = 2d_{max} - d_{min}$

Systems with global time (timestamped messages):

action delay: $d_{max} + 2g$

In distributed real time system the maximum protocol execution time and not the “median” protocol execution time determines the responsiveness!

Accuracy vs. Action Delay

In a properly designed RT system

$$\text{Action Delay} < d_{acc}$$

- Accuracy (d_{acc}) is an application specific parameter.
- The action delay is an implementation-specific parameter.

What happens if this condition is violated?

➪ Then we need state estimation!

Idempotence

A set of messages is *idempotent*, if the effect of receiving more than one messages of this set is the same as the effect of receiving a single message.

- Duplicated state messages are idempotent.
- Duplicated event messages are not idempotent.

Idempotence of redundant messages simplifies the design of fault-tolerant systems.

Determinism – First Attempt

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.

- Definition of determinism is intuitive,
- neglects the fact that in a real (physical) distributed system clocks cannot be precisely synchronized,
- therefore a system-wide consistent representation of time (and consequently state) cannot be established.

Determinism

Let us assume

- Q *is a finite set of symbols denoting states*
- Σ *is a finite set symbols denoting the possible inputs*
- Δ *is a finite set of symbols denoting the possible outputs*
- $q_0 \in Q$ *is the initial state*
- $t_i, \in N$ *is the infinite set of active sparse time intervals*

then a model (*processing, communication*) is said to behave *deterministically* **iff, given** a sequence of *active sparse real-time intervals* t_i, the initial state of the system $q_0(t_0) \in Q$ *at* t_0 *(now),* and a sequence of *future* inputs $in_i(t_i) \in \Sigma$ **then** the sequence of *future* outputs $out_j(t_j) \in \Delta$ and the sequence of future states $q_j(t_j) \in Q$ is *entailed.*

Replica Determinism

A set of replicated RT-objects is replica determinate if all objects of this set visit the same state within a specified interval of real time and produce identical outputs.

The time interval of this definition is determined by the precision of the clock synchronization.

Replica Determinism

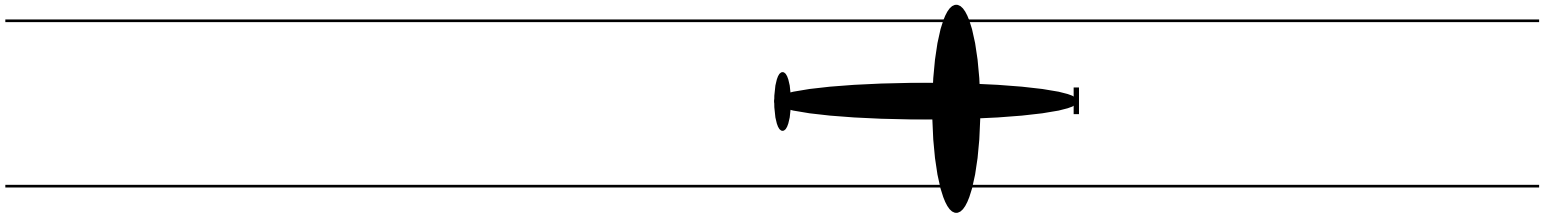
Replica Determinism is needed for the following reasons:

- To implement consistent distributed actions.
- To improve the testability of systems – tests are only reproducible if the replicas are deterministic.
- To facilitate the implementation of fault tolerance by active replication.

Replica Determinism helps to make systems more intelligible!

Example: Airplane Takeoff

Consider an airplane with a three channel flight control system taking off from a runway:



Channel 1	Take off	Accelerate Engine
Channel 2	Abort	Stop Engine
Channel 3	Take off	*Stop Engine (Fault)*
Majority	*Take off*	

The Simultaneity Problem

The ordering of simultaneous events is a fundamental problem of computer science:

- Hardware level: metastability
- Node level: semaphor operation
- Distributed system: ordering of messages

There are two solutions *within* a distributed system to solve the simultaneity problem:

- Distributed consensus – takes real-time and requires bandwidth (atomic broadcast)
- Sparse time

Replica Determinism: Destroying Factors

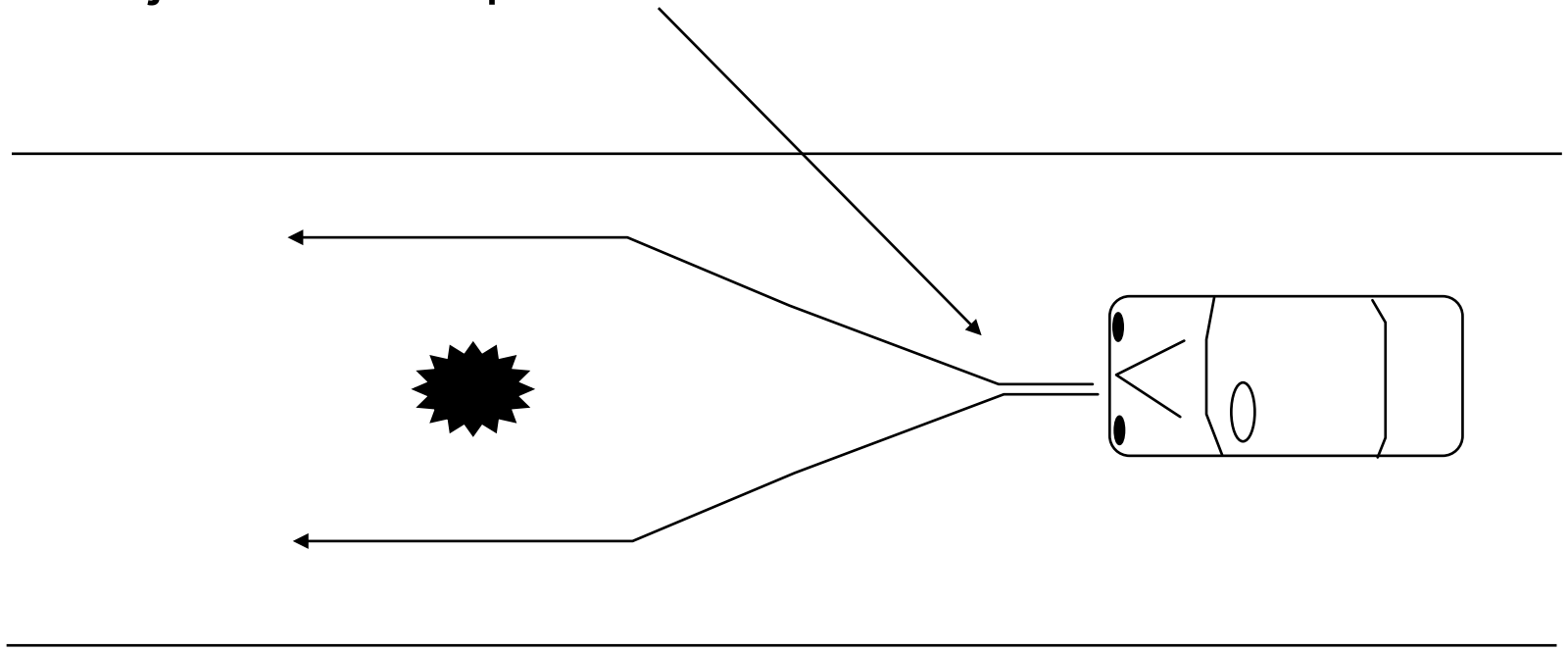
Replica determinism can be destroyed by:

- Inconsistent inputs
- Inconsistent order
- Non-deterministic program constructs
- Explicit synchronization statements (e.g., *Wait*)
- Uncontrolled access to the global time and timeouts
- Dynamic preemptive scheduling decisions
- Consistent comparison problem (software diversity)

This list is not complete!

Major Decision Point

How can we make sure, that both replicas take the same decision at this major decision point?



Basic Causes of Replica Indeterminism

At the root of replica indeterminism are:

- Differing processing speeds (diff. crystal resonators, clocks)
- Finite representation of real values (e.g., real arithmetic)
- Differing inputs (inconsistent order or sensor variations)

Solutions:

- Sparse value/time base
- Static or non-preemptive scheduling
- exact arithmetic
- agreement on input data and order

Points to Remember

- Modeling – assumption coverage determines value
- RTS cluster model
- Component = hardware + software + state
- Interfaces and their properties
- RT-entity vs. RT-image, RT-object
- Observation – only state is observable
- Temporal accuracy and state estimation
- Action delay limits responsiveness
- Replica determinism supports fault tolerance