OMG Smart Transducer Specification (I)

H. Kopetz
TU Wien
July 2003

The Time-Triggered Architecture

Take *Time* from the Problem Domain

And move it into the Solution Domain

Basic Concepts

- RT System Requirements
- Model of Time
- Model of a Component
- Temporal Accuracy
- Interfaces

When is a Computer System ‘Real-Time’?

A *real-time computer system* is a computer system in which the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time, when these results are produced.

The point in time when a result has to be produced is called a *deadline*.

Deadlines are dictated by the environment of the real-time computer system.

Some Definitions

If the result has utility even after the deadline, we call the deadline *soft.* Systems with soft deadlines are not the focus of these lectures.

If the result has no utility after the deadline has passed, the deadline is called *firm*.

If a catastrophe could result if a strict deadline is missed, the deadline is called *hard*.

A real-time computer system that has to meet at least one hard deadline is called a *hard real-time system.*

Hard- and soft real-time system design are fundamentally different.

Hard Real Time versus Soft Real Time

Characteristic	Hard Real Time	Soft Real Time
Response time	hard	soft
Pacing	environment	computer
Peak-Load Perform.	predictable	degraded
Error Detection	system	user
Safety	critical	non-critical
Redundancy	active	standby
Time Granularity	millisecond	second
Data Files	small/medium	large
Data Integrity	short term	long term

Fail-Safe versus Fail-Operational

A system is *fail-safe* if there is a safe state in the environment that can be reached in case of a system failure, e.g., ABS, train signaling system.
In a fail-safe application the computer has to have a high *error detection coverage.*
Fail safeness is a characteristic of the application, not the computer system.

A system is *fail operational*, if no safe state can be reached in case of a system failure,e.g., a flight control system aboard an airplane.
In fail-operational applications the computer system has to provide a minimum level of service, even after the occurrence of a fault.

Predictability in Rare Event Situations

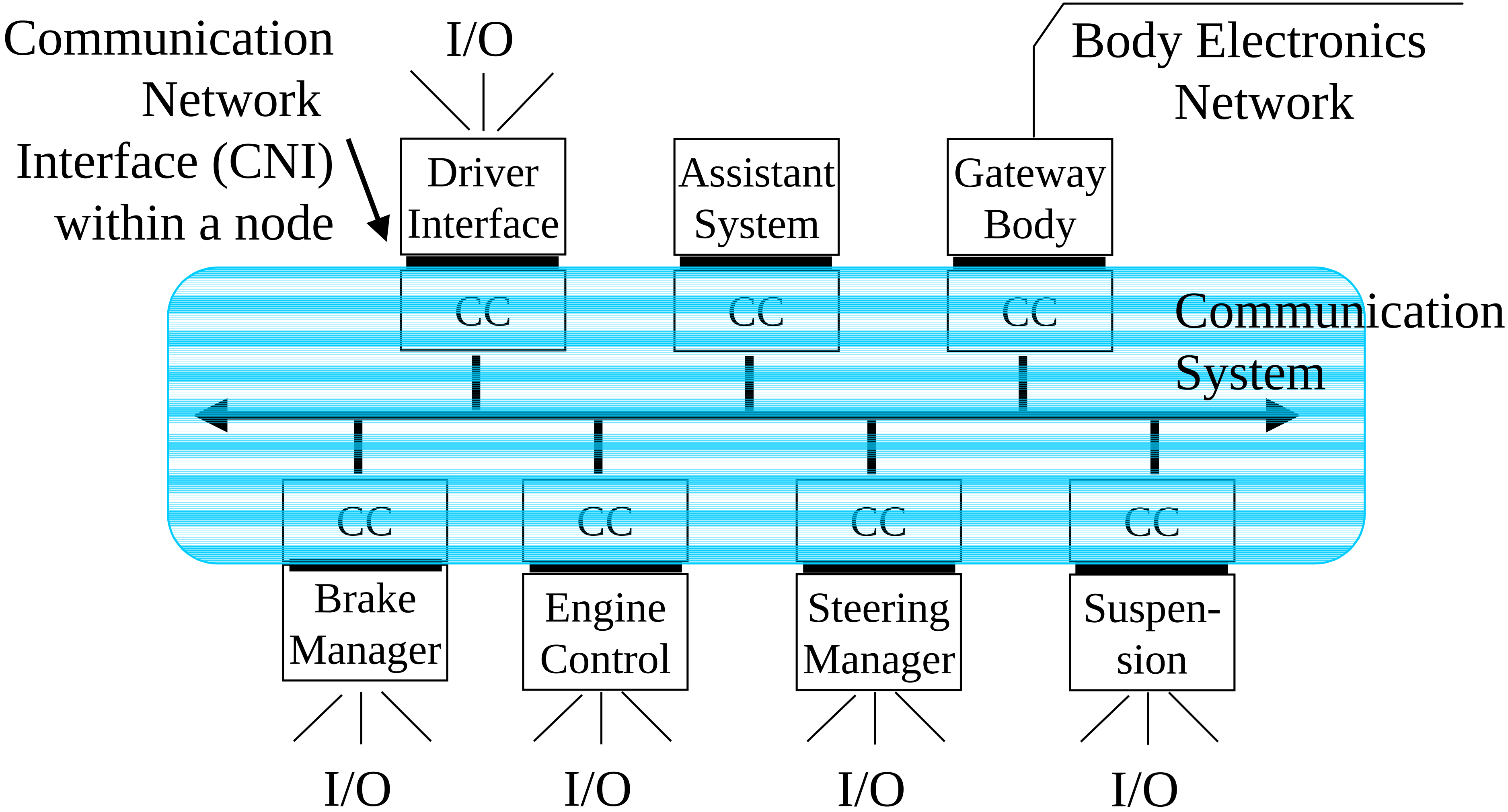
A *rare event* is an important event that occurs very infrequently during the lifetime of a system, e.g., the rupture of a pipe in a nuclear reactor.

A rare event can give rise to many correlated service requests (e.g., an alarm shower).

In a number of applications, the utility of a system depends on the predictable performance in rare event scenarios, e.g. flight control system

In most cases, workload testing will not cover the rare event scenario.

Example of a Distributed System



CC: Communication Controller

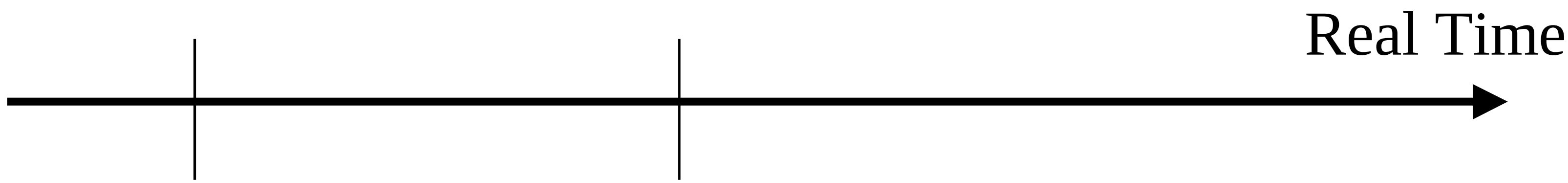
Some Important Concepts in Relation to Time

We assume a (dense) Newtonian time in the environment.

Instant: cut of the timeline

Duration: interval on the timeline

Event: occurrence at an instant--has no duration



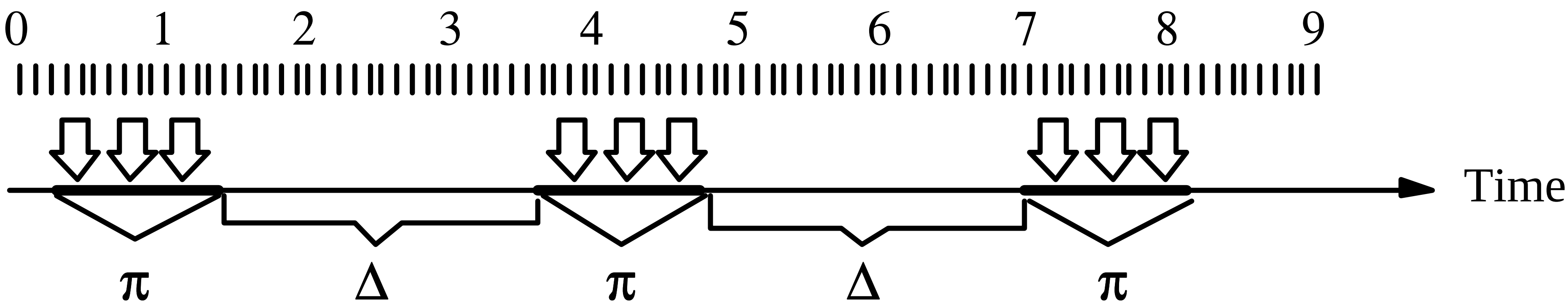
Omniscient Observer: has a reference clock that is in perfect Synchrony with Atomic Time

Absolute Timestamp: Timestamp generated by the reference clock

Global Sparse Time Base

It is assumed that within the distributed system a global time of known precision π is available at every node.

The global time is used to build a sparse time base as follows:



Events ⇩ are only allowed to occur at subintervals of the timeline

What is a "Component"?

In our context, a *component* is complete computer system that is time aware. It consists of

- The hardware
- The system and application software
- The internal state

The component interacts with its environment by the exchange of messages via interfaces.

What is a *software* component?

Closed Component vs. Open Component

- **Closed Component:** Contains no local interface to the *real world*, but can contain local interfaces to other closed components.
  Semi-closed if it is time-aware.
- **Open Component:** Contains an interface to the *real world*.
  Semi-open if no control signals are accepted from the real-world (e.g., a sampling system).

The real world has an unbounded number of properties.

Message-Model–Appropriate Abstraction

Message: An atomic data structure that is formed for the purpose of communication among nodes

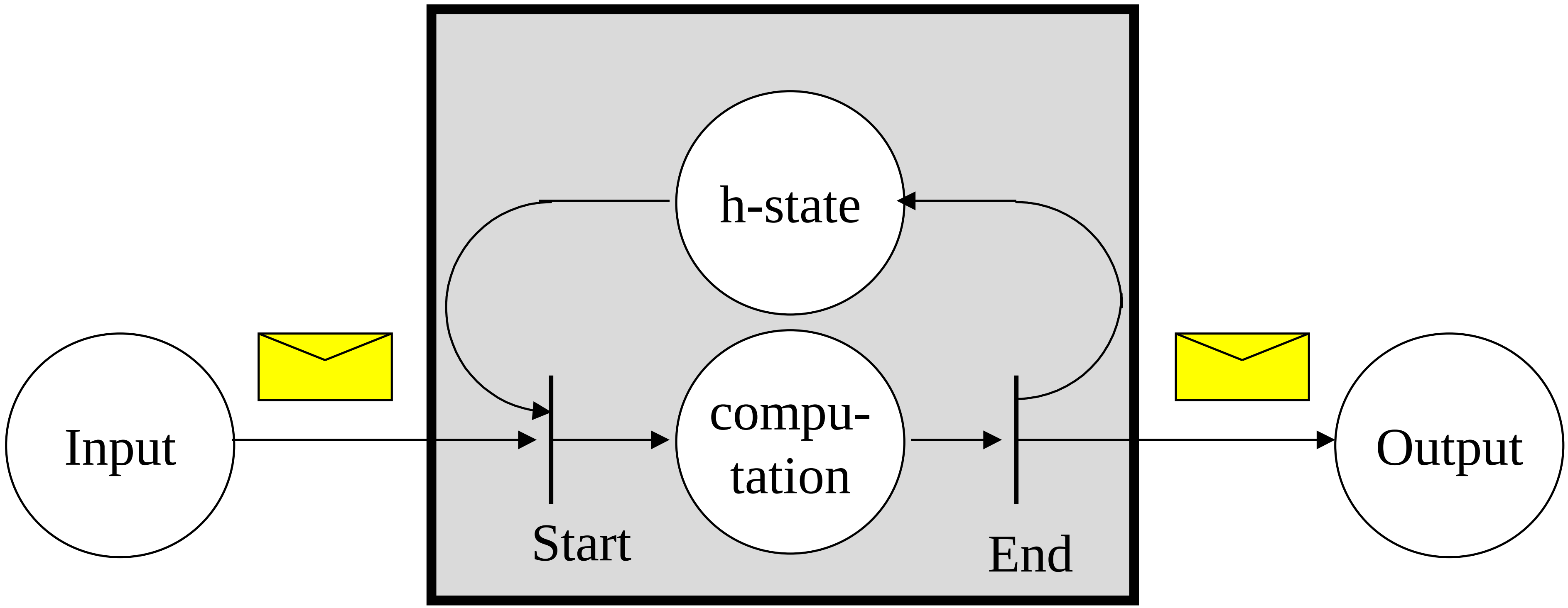
Message Send Instant: The instant when the sending of a message starts at the sender

Message Receive Instant: The instant when the receiving of a message terminates at the receiver

State Message: A (periodic) message that contains state information. Non comsumable read at sender and update in place semantics at receiver.

Event Message: A message that contains event information. Consumed on reading and queued at receiver

Model of a Component–Messages



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 message 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 message is *correct* if it is both timely and value-correct.	incorrect
insidious	A message is *insidious* if it is permitted but incorrect	not insidious

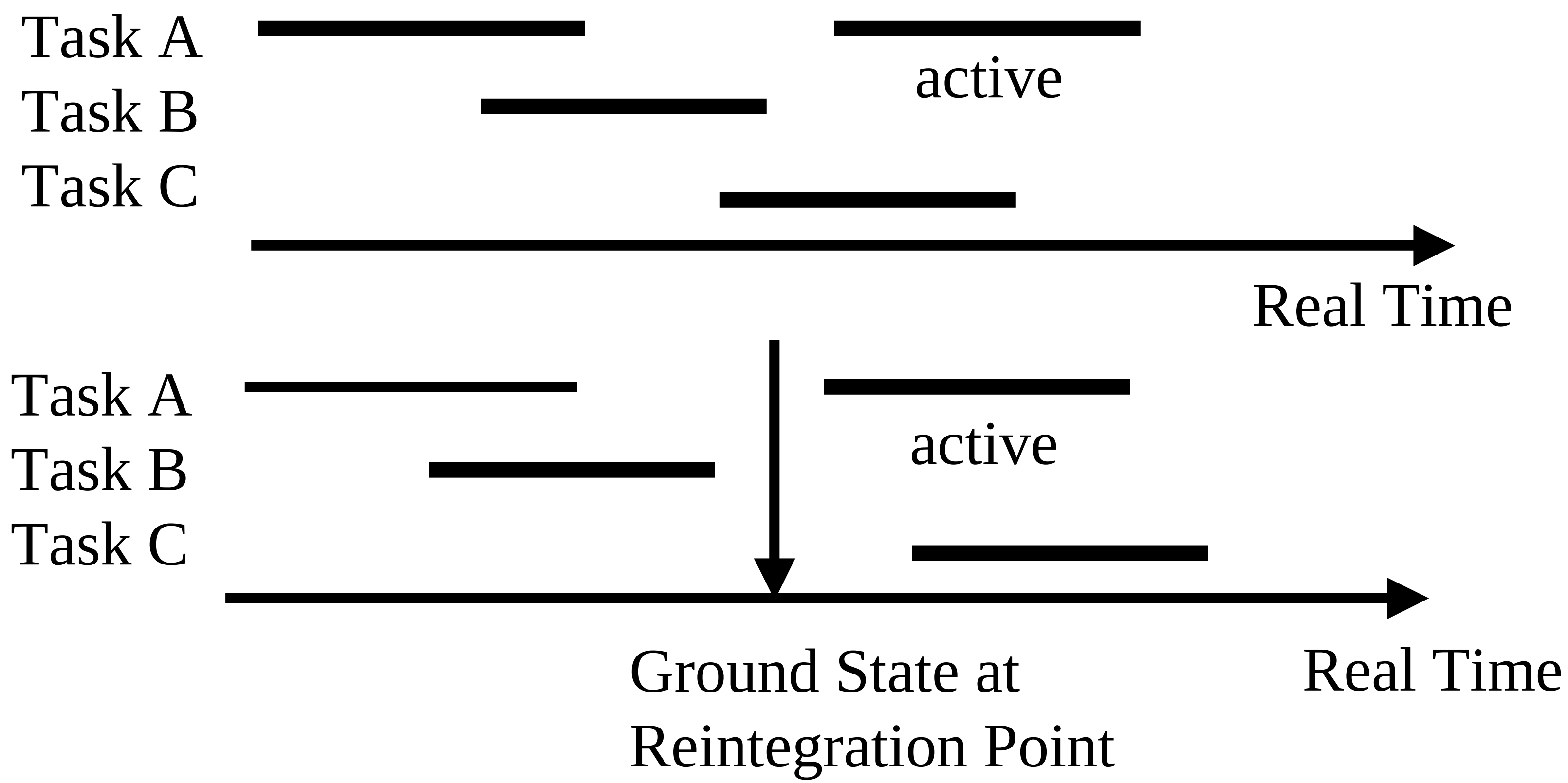
History State (h-state)

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

- ♦ Size of the h-state depends on the point in time chosen
- ♦ relative minimum immediately after a computation (an atomic action) has been completed.
- ♦ System in *ground state*: no messages in transit and no activity occurring.
- ♦ shall be small at reintegration points.

If no h-state has to be stored between successive activations of the node, the node is called “stateless” (at the chosen level of abstraction!).

Ground State (g-state)



g-state: Minimal h-state of a subsystem (node) where are tasks are inactive and all channels are flushed. Needed for reintegration of nodes.

Temporal Requirements

Timeliness: An output message must be submitted to the environment at the specified instant (deadline).

Temporal accuracy of real-time data: the data elements that are used in an a time-sensitive operation must be *temporally accurate*.

Minimal Jitter: The variability between a stimulus and a response should be as small as possible.

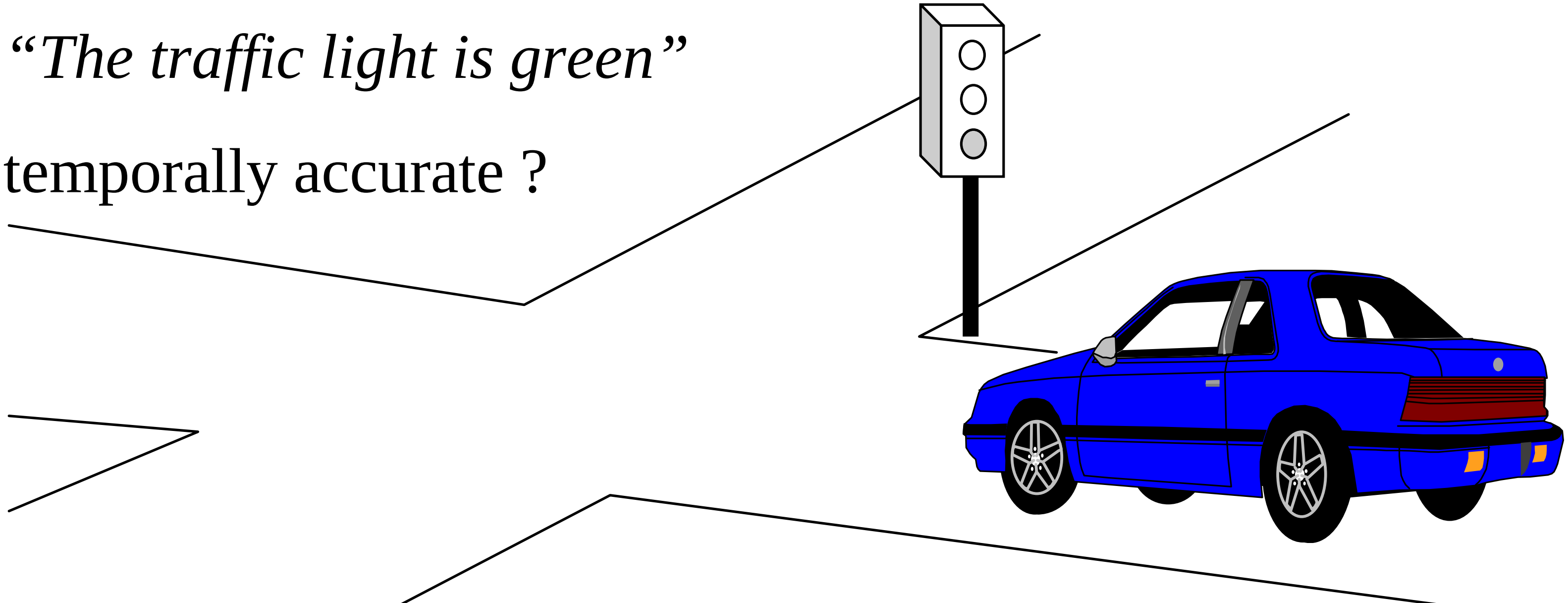
Jitter: The difference between maximum and minimum latencies

Validity of Real-Time Information

How long is the observation:

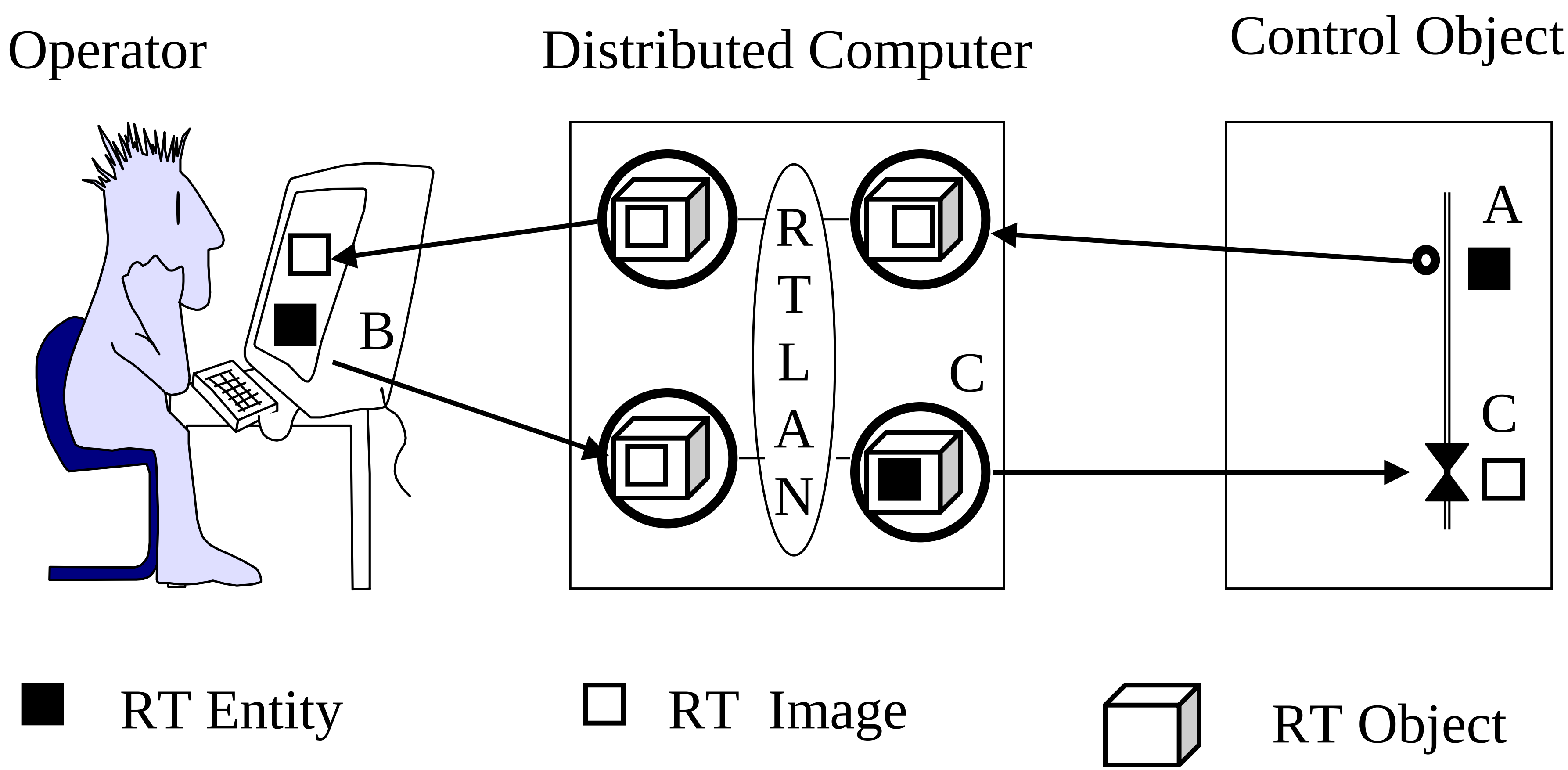
"The traffic light is green"

temporally accurate ?



Temporal parameters are associated with real-time data.

RT Entities, RT Images and RT Objects



A: Measured Value of Flow

B: Setpoint for Flow C: Intended Valve Position

Real Time (RT) Entity

A Real-Time (RT) Entity is a state variable of interest for the given purpose that changes its state as a function of real-time.

We distinguish between:

- ♦ Continuous RT Entities
- ♦ Discrete RT Entities

Examples of RT Entities:

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

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:

- ♦ Setpoint is in the SOC of the operator
- ♦ Actual Flow is in the SOC of the control object
- ♦ Intended Valve Position is in the SOC of the Computer

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

At this level of abstraction, changes in the representation of a RT-entity are not significant.

Observation

Information about the state of a RT-entity at a particular point in time is captured in an observation.

An observation is an atomic triple

Observation = <Name, Time, Value>

consisting of:

- The name of the RT-entity
- The point in real-time when the observation has been made
- The values of the RT-entity

Observations are transported in messages.

State and Event Observation

An observation is a *state observation*, if the value of the observation contains the full or partial state of the RT-entity. The time of a state observation denotes the point in time when the RT-entity was sampled.

An observation is an *event observation*, if the value of the observation contains the difference between the “old state” (the last observed state) and the “new state”. The time of the event information denotes the point in time of the L-event of the “new state”.

RT Images

A RT-Image is a picture of a RT Entity. A RT image is valid at a given point in time, if it is an accurate representation, both in the domains of value and time, of the corresponding RT Entity.

RT-Images

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

RT Object

A RT-object is a “container” for a RT-Image or a RT-Entity in the Computer System.

A RT-object k

- has an associated real-time clock which ticks with a granularity t_k. This granularity must be in agreement with the dynamics of the RT-entity this object is to represent.
- Activates an object procedure if the 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 a *synchronous* RT object.

Temporal Accuracy (II)

The temporal accuracy of a RT image is defined by referring to the recent history of observations of the related RT entity. A recent history RH_i at time t_i is an ordered set of time points $<t_i, t_{i-1}, t_{i-2}, \ldots t_{i-k}>$, where the length of the recent history

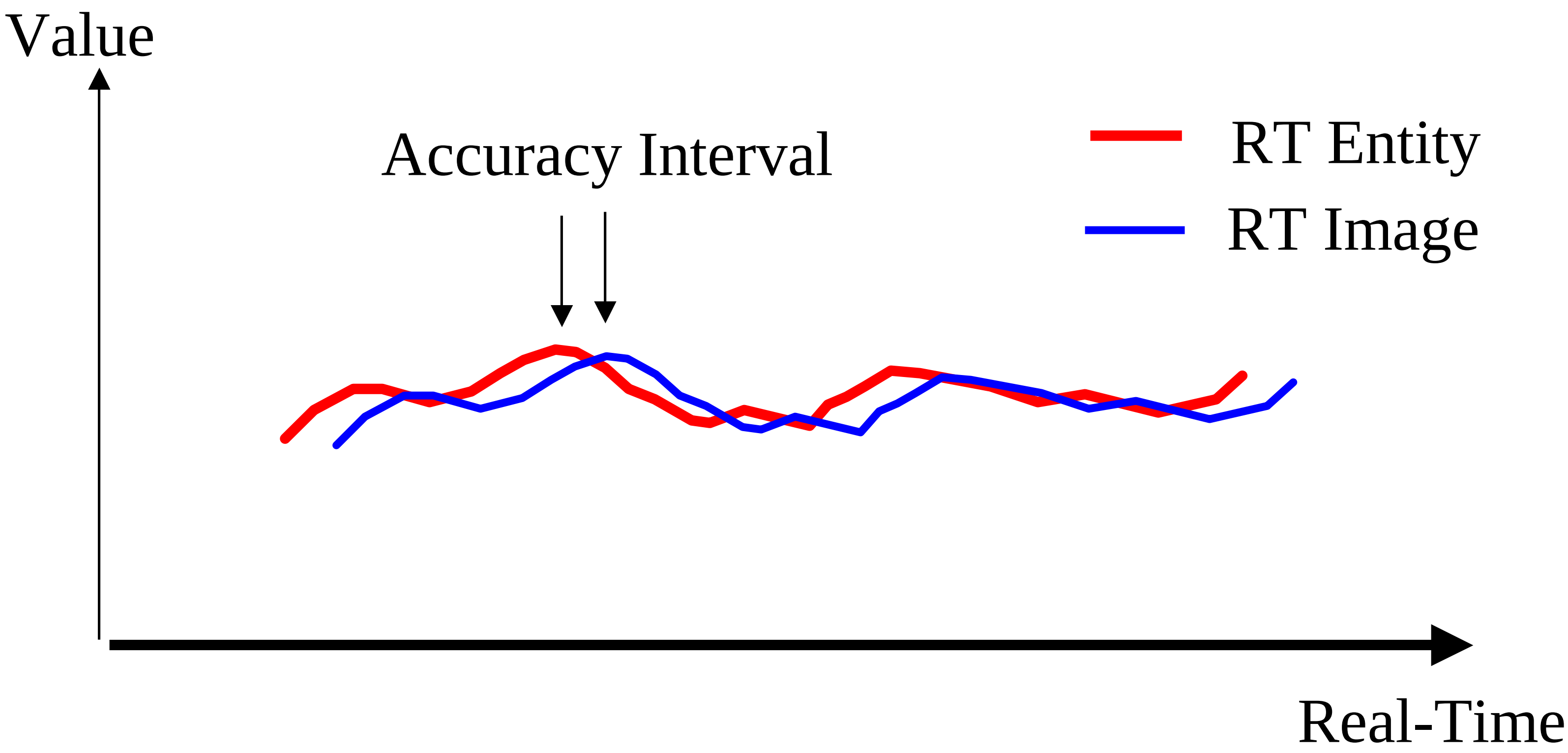
$$d_{acc} = t_i - t_{i-k}$$

is called the temporal accuracy. Assume that the RT entity has been observed at every time point of the recent history. A RT image is temporally accurate at the present time t_i

if

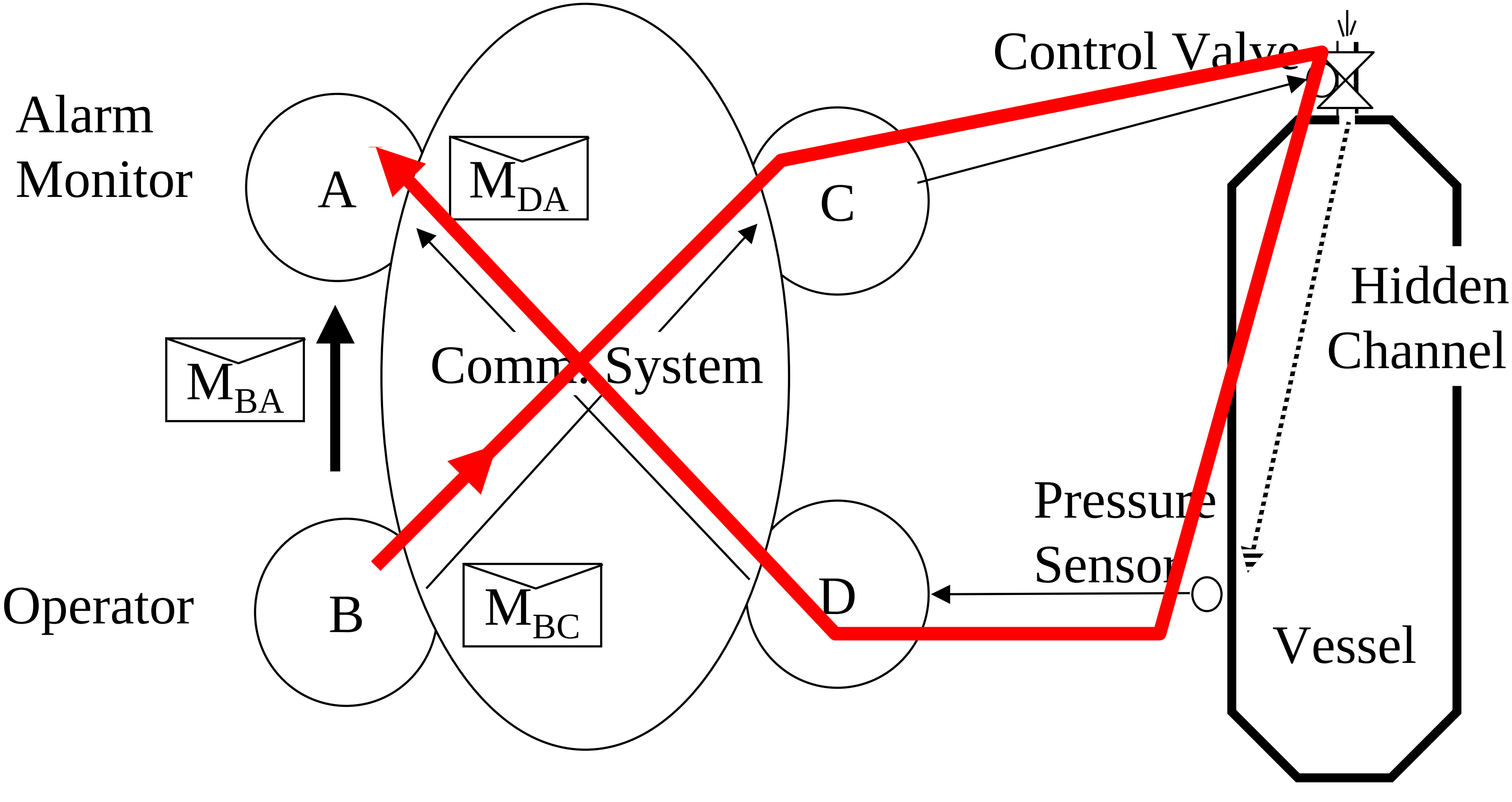
$$\exists t_j \in RH_i : Value(RT image at t_i) = Value(RT entity at t_j)$$

Temporal Accuracy of an RT Object

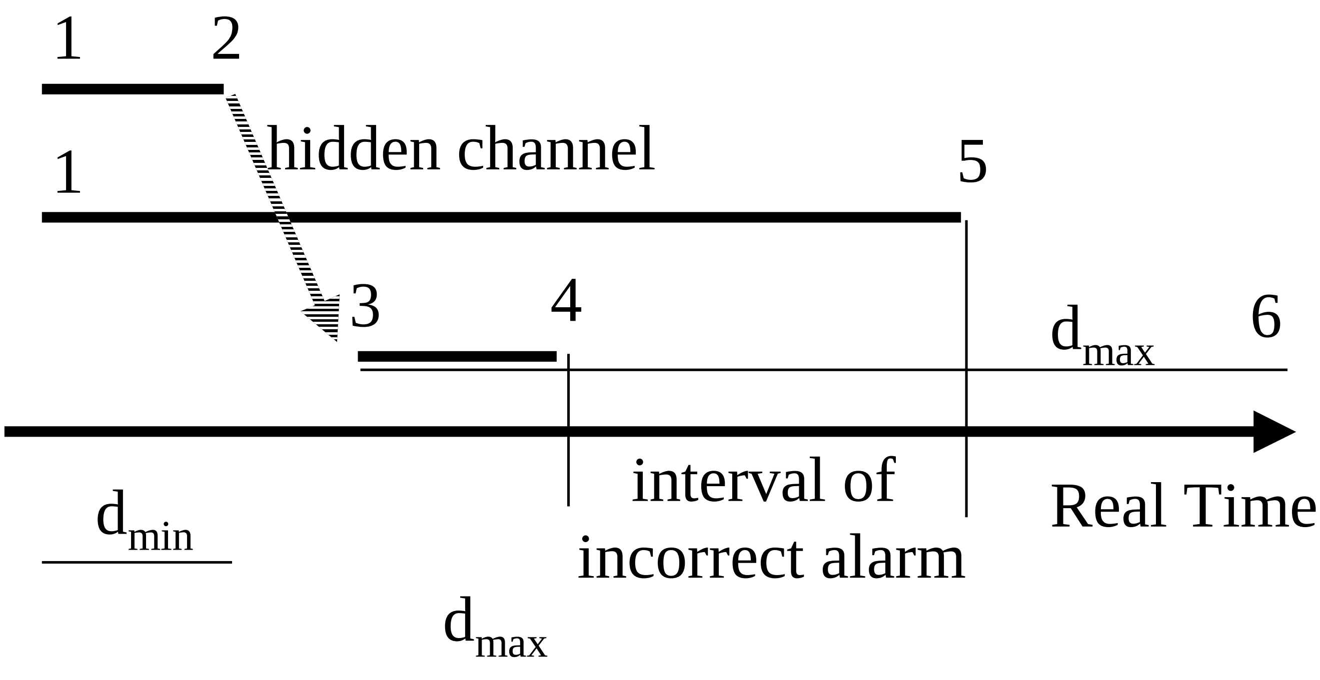


If a RT-object is updated by observations, then there will always be a delay between the state of the RT entity and that of the RT object

Hidden Channel (red)



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

Permanence is a relation between a given message M_i that has arrived at a RT-object O and all messages M_{i-1}, M_{i-2}, . . . that have been sent to this object before (in the temporal order) message M_i.

The message M_i becomes *permanent* at object O as soon as all previously sent messages have arrived at O.

If actions are taken on non-permanent messages, then an inconsistent behavior may result.

The *action delay* is the 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.

How long does it take until a message becomes permanent?

Action Delay

In distributed RT systems without a global time base the

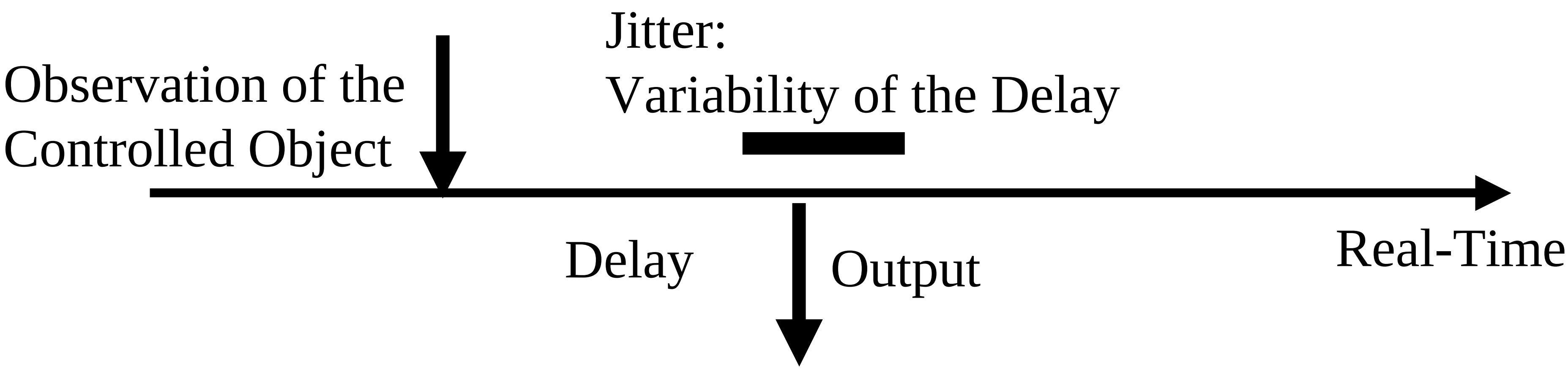
maximum action delay: $d_{max} + \varepsilon = 2\,d_{max} - d_{min}$

but the consistent order problem is not yet solved!

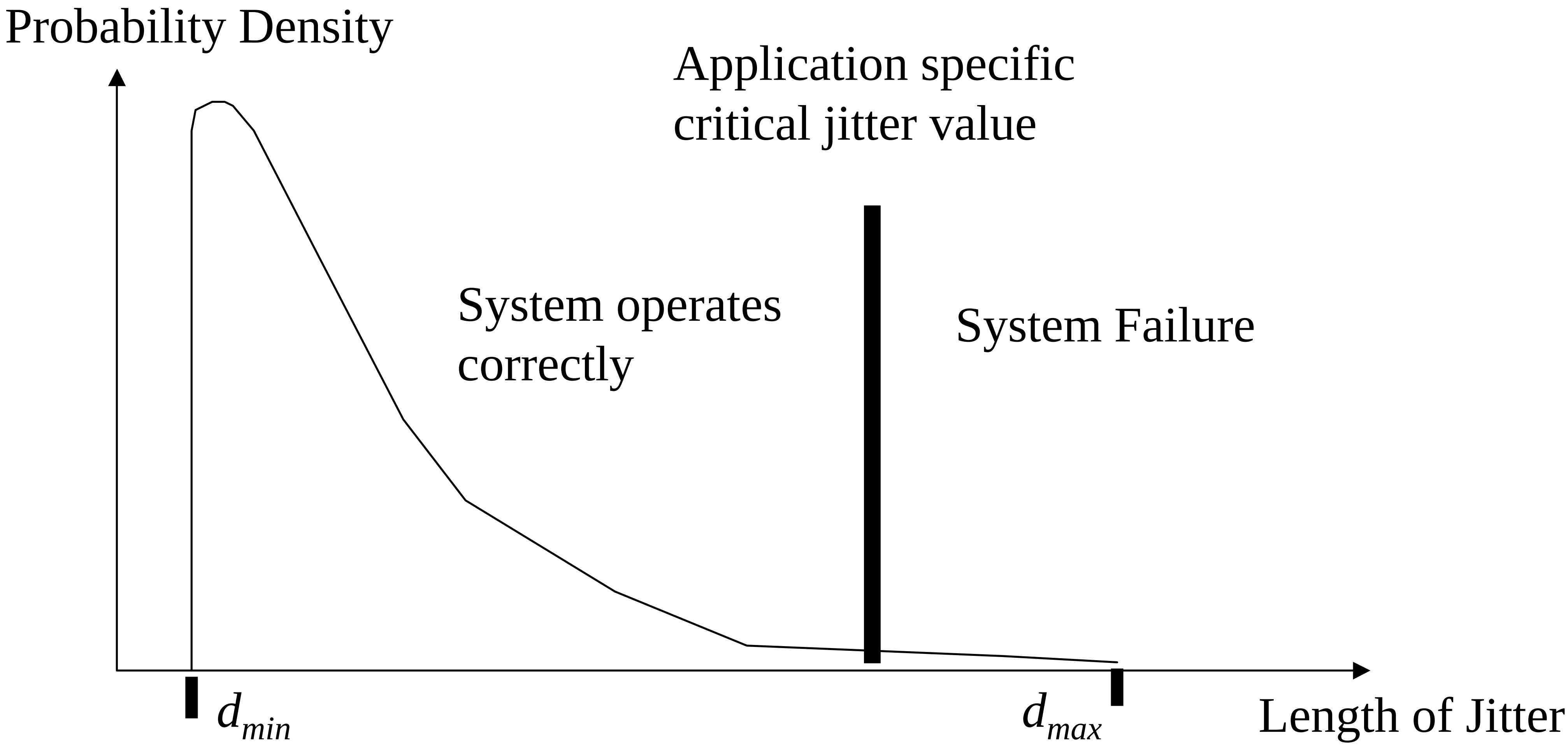
In systems with a global time the maximum action delay: $d_{max} + 2g$

In distributed real time system the maximum protocol execution time and not the "median" protocol execution time determines the responsiveness.

Jitter at the Application Level

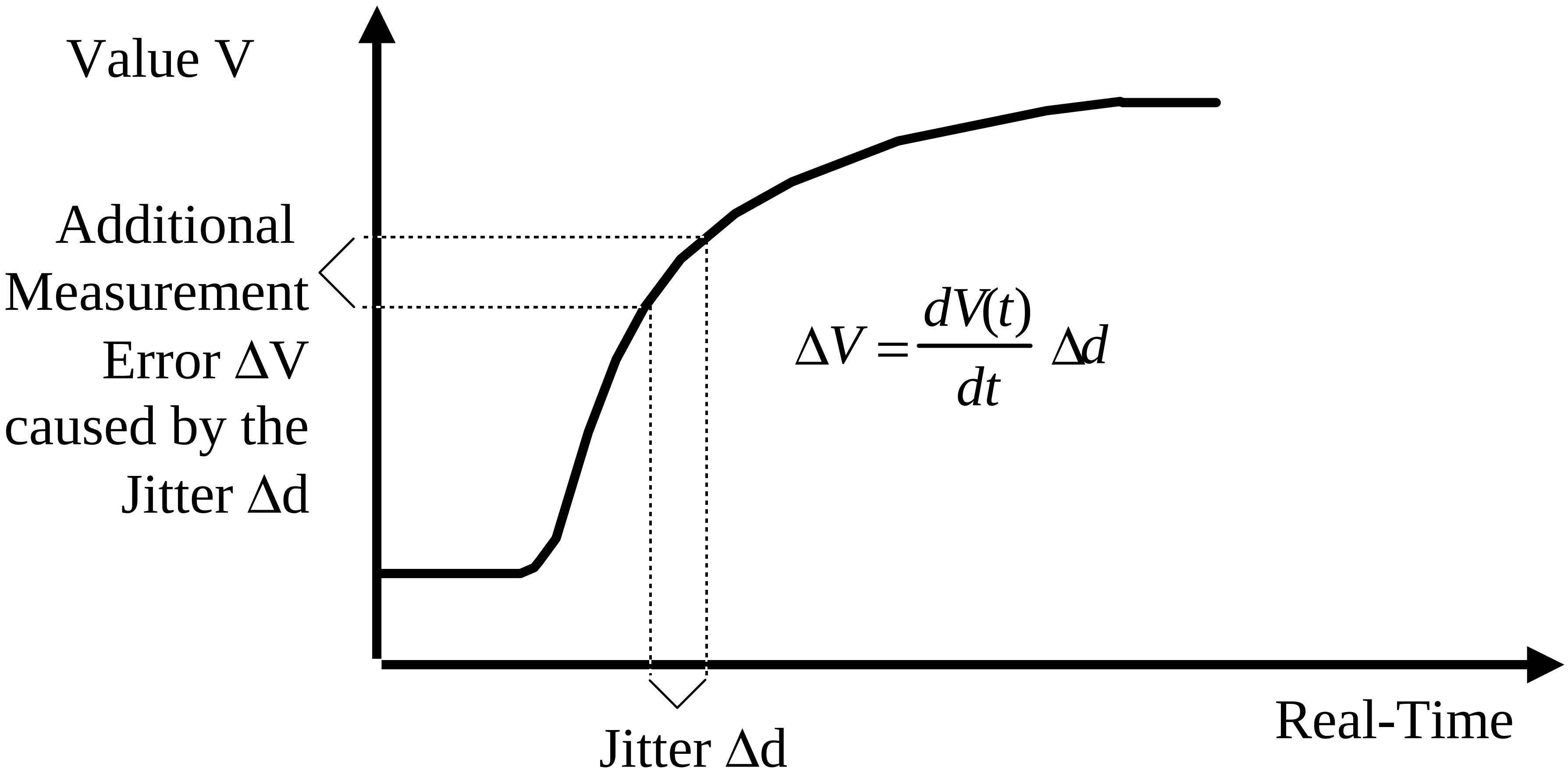


Probability of “Long” Jitter in PAR Protocols



Most of the time, the system will operate correctly.

The Effect of Jitter: Measurement Error

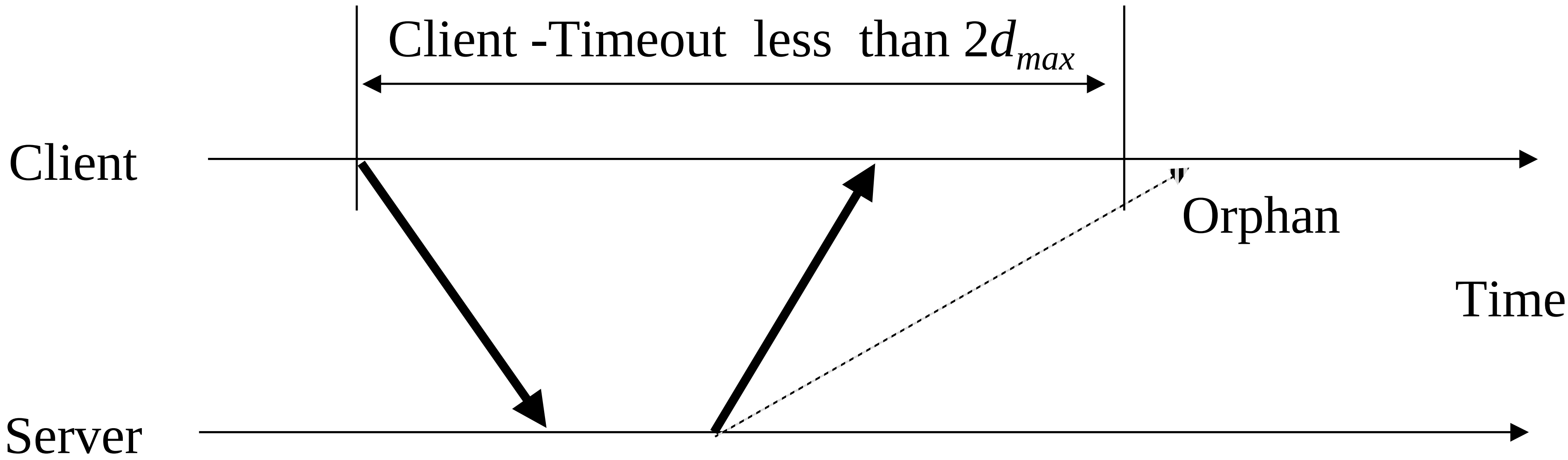


$$\Delta V = \frac{dV(t)}{dt} \Delta d$$

Jitter in Control Loops causes a degradation of control quality.

The Effect of Jitter: Orphans

Request Response Transaction between a Client and a Server:



How large is d_{max} ?

It is not contained in the interface specification, available at the sub-supplier.

ET Systems: Jitter at Critical Instant

A *critical instant* is a point in time, when all hosts in the ECUs try to send a message simultaneously. There is no phase control possible in ET system.

The message at the lowest priority level must wait until all higher priority messages have been sent (assume that all message have the same length).

Protocol execution time at critical instant (n ECUs):

$$d_{max} = n\, d_{trans}$$

Protocol execution time if the channel is free:

$$dmin = d_{trans}$$

Jitter of the lowest priority message:

$$Jitter = (n-1)\, d_{trans}$$

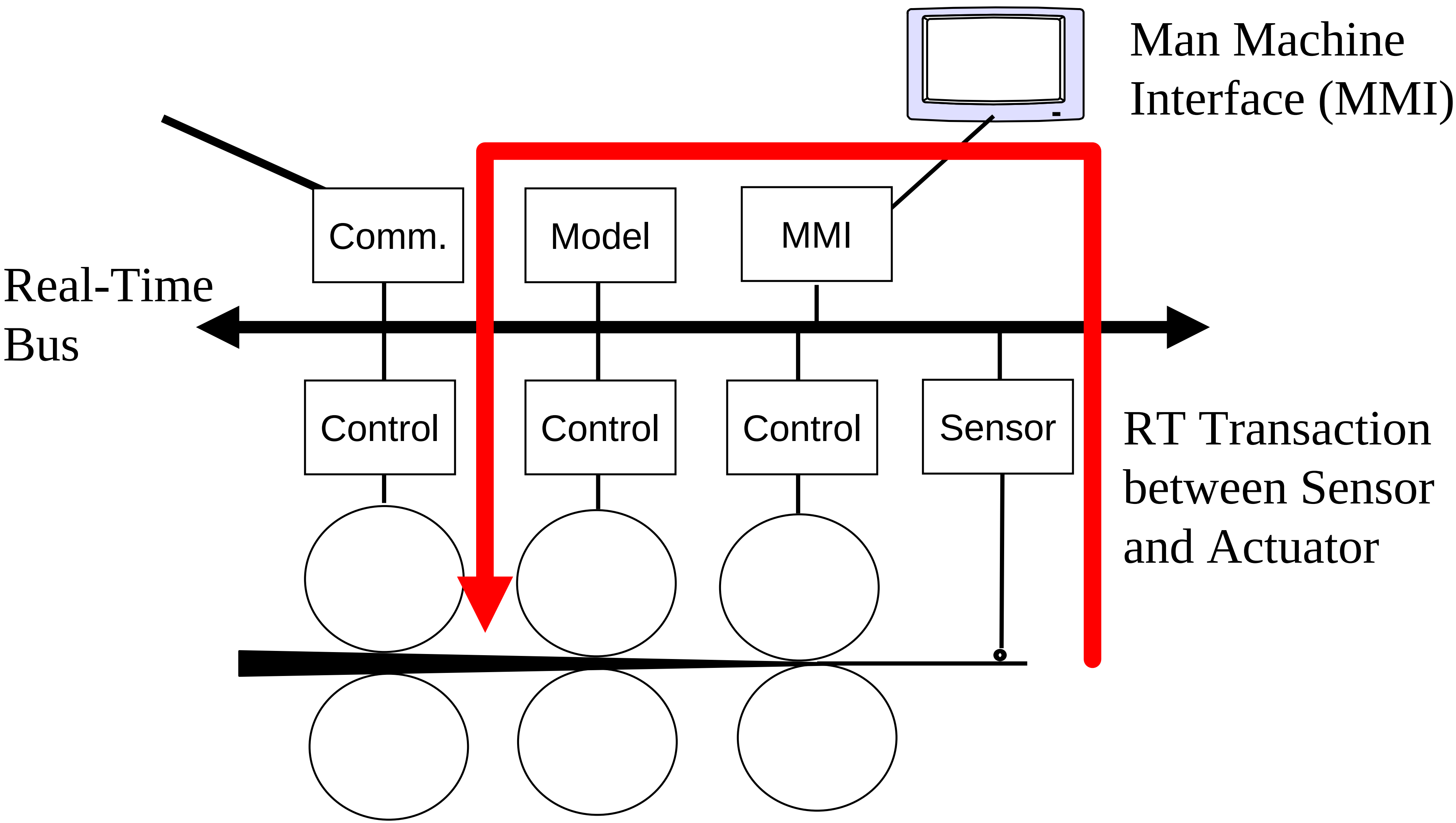
The jitter depends on the number of ECUs in the system.

Summary: Jitter is Bad

The consequences of a long jitter:

- ♦ Measurement error increases
- ♦ Probability of Orphans
- ♦ Action Delay increases
- ♦ Clock Synchronization difficult

Real-Time Transaction



Rolling Mill Example

An alarm monitoring component should raise an alarm

WHEN $p_1 < p_2$ THEN raise alarm;

At a first glance, this specification of an alarm condition looks reasonable. A further analysis leads to the following open questions:

- What is the maximum$_1$ and p_2 ?
- At what points in time must the alarm condition be evaluated?

Logical versus Temporal Control

The control scheme determines at what point in time the execution of a selected action will start. In RT systems it is necessary to distinguish between:

- *Logical Control* is concerned with the control flow within a task to realize the specified data transformation
- *Temporal Control* is concerned with the point in time when a task is to be started or when it has to be preempted by a more urgent task. Temporal control is closely related to scheduling

Synchronous Programming Languages

In the past twenty years, synchronous programming languages have been developed that distinguish clearly between *temporal control* and *logical control*:

Initialize Memory
For each clock tick (or input event) do
Read Inputs
Compute Outputs
Update Memory

LUSTRE: Used for the development of the flight critical control software in Airbus planes

ESTEREL: Used in telecommunication

Ref: Beneviste, A et. Al.: The Synchronous Languages, Twelve years later

Proc. of the IEEE Vol 91, Nr. 1, Jan 2003, p. 64-84

Interface

Interface: A point of interaction between a system and its environment

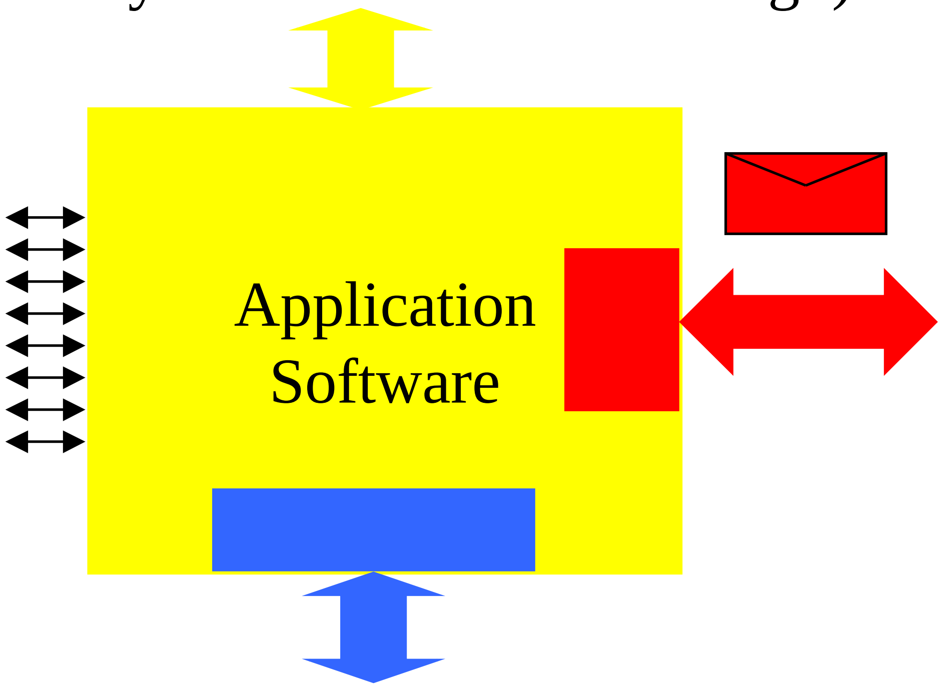
Linking Interface: An interface of a component through which it is connected to other components.

Service Providing LIF (**SPLIF** or LIF, for short): A LIF where the real-time service of a component is provided to its environment

Service Requesting LIF (SRLIF): A LIF where a service is requested from another component

Interface Model: The model of the concepts a user has in mind when he/she relates the meaning of the chunks of information in a message to his/her conceptual world.

Interfaces of a Node--Messages



The Three Interfaces

The three interfaces of an embedded system node:

Realtime Service (RS) Interface--LIF:

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

Diagnostic and Maintenance (DM) Interface:

- Sporadic access
- Requires knowledge about internals of a node (Restrictions in order to protect IP)
- Not time sensitive

Configuration Planning (CP) Interface:

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

SPLIF or LIF is Important for Composability

For the temporal composability, only the LIF interface is relevant.

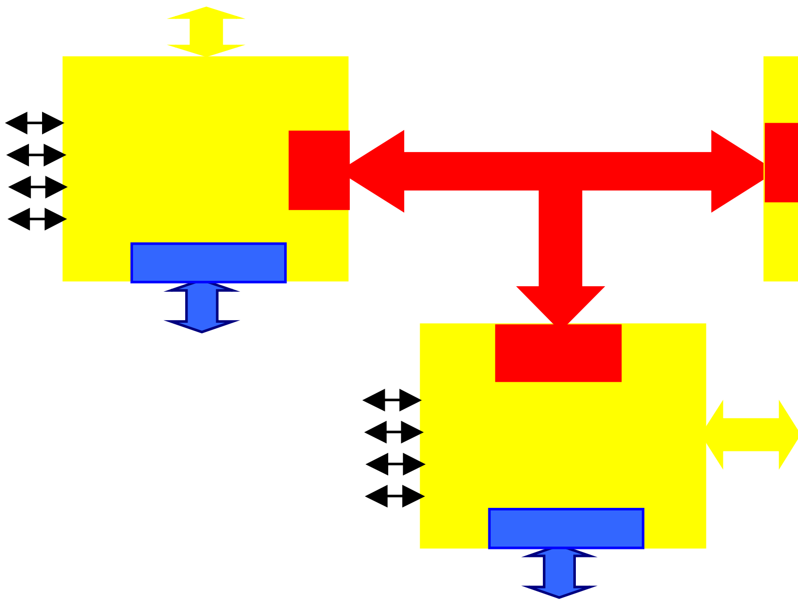
An LIF (e.g., a control algorithm) must specify:

- ♦ At what point in time the input information is delivered to a module (temporal pre-conditions)
- ♦ At what point in time the output information must be produced by the module (temporal post-conditions).
- ♦ The properties of the intended information transformation provided by the module (a proper model)

Focus on Message Based Interfaces!

A Composition Involving three LIFs

Linking Interfaces



Four Principles of Composability (LIF)

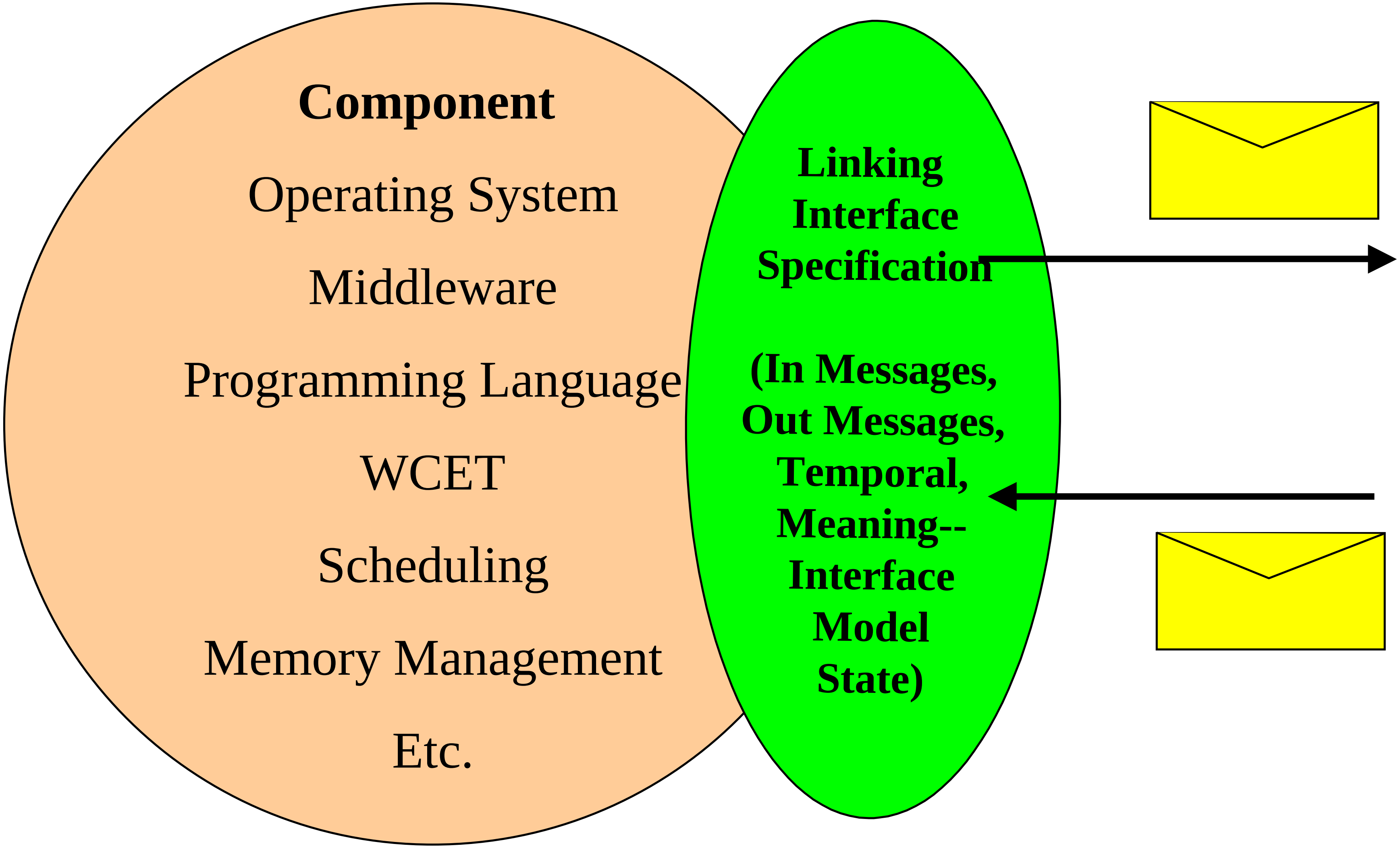
(1) Independent Development of the Components (Architecture)
The message interfaces of the components must be *precisely specified* in the value domain and in the *temporal domain* in order that the component systems can be developed in isolation.

(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) Performability of the Communication System (Comm. System)
The communication system transporting the messages must meet the given temporal requirements under all specified operating conditions.

(4) Replica Determinism (Architecture)
Replica Determinism is required for the transparent implementation of fault tolerance

The LIF Specification hides the Implementation



Views of a System: Four Universe Model

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

Avizienis, FTCS 12, 1982

Interface Specification

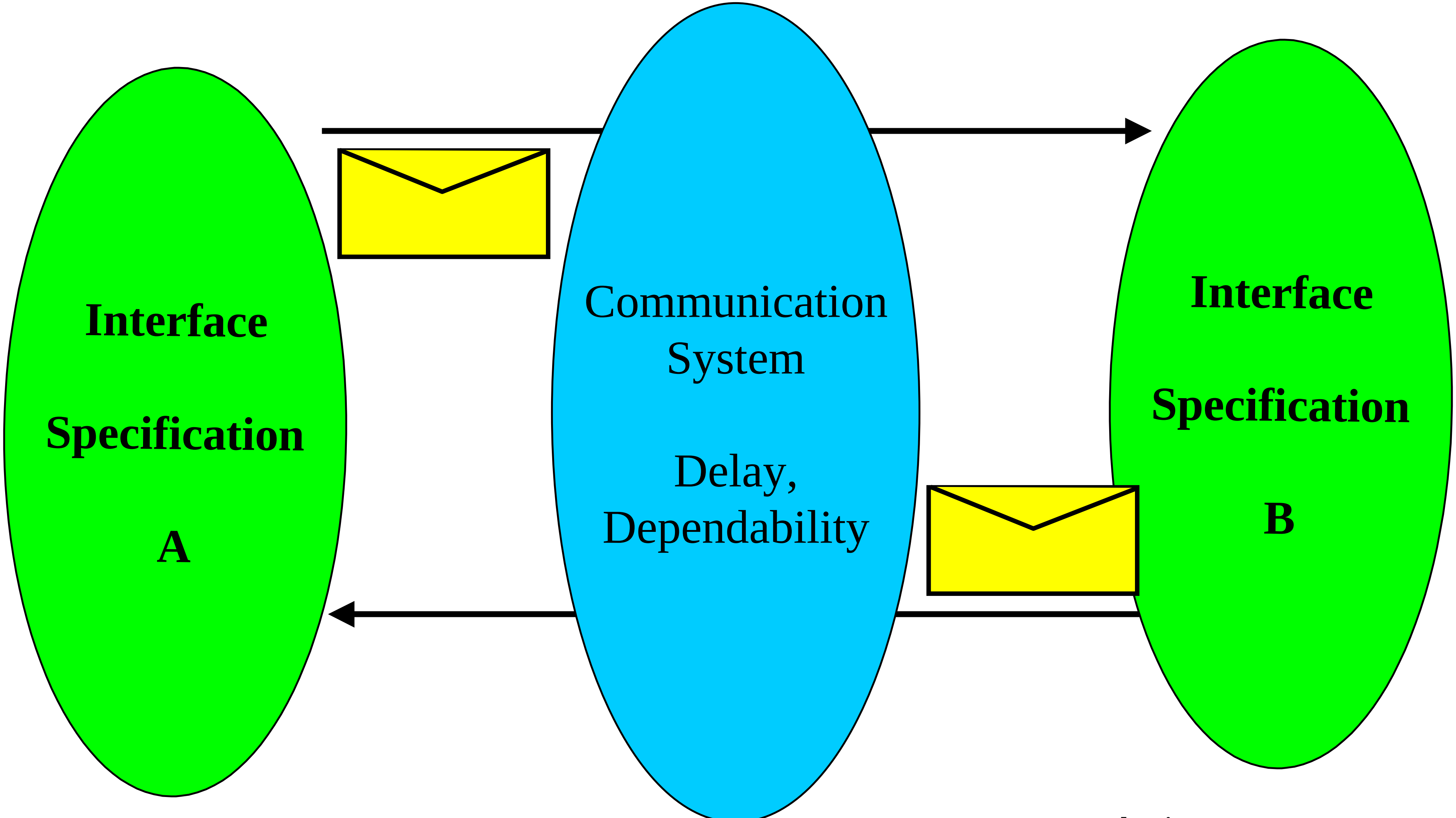
Operational Specification:

- ♦ **Operational Input Interface Specification**
  - Syntactic Specification
  - Temporal Specification
  - Input Assertion
- ♦ **Operational Output Interface Specification**
  - Syntactic Specification
  - Temporal Specification
  - Output Assertion
- ♦ **Interface State**

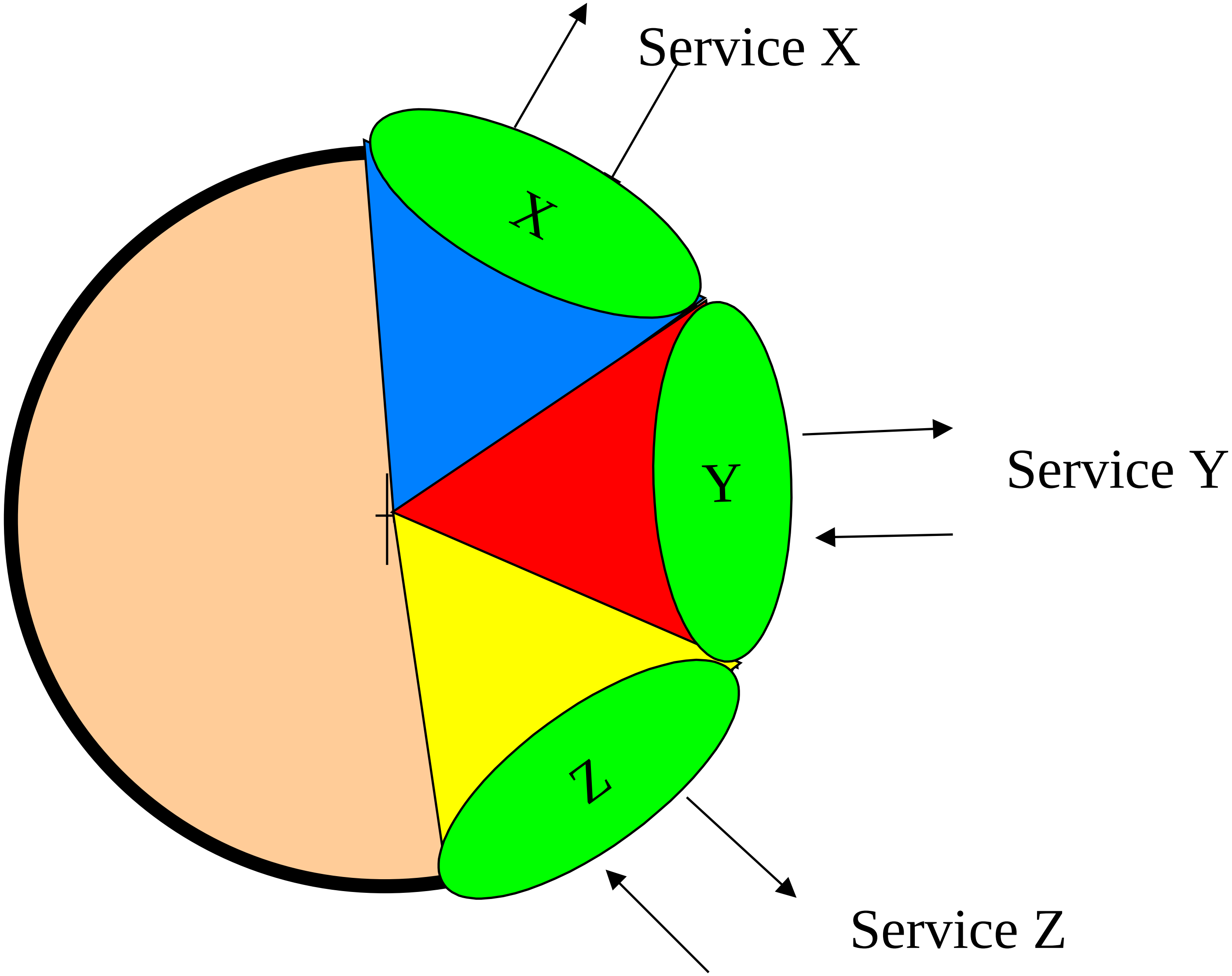
Meta-level Specification:

- ♦ Meaning of the data elements: Means-and-ends model

Composability in Distributed Systems



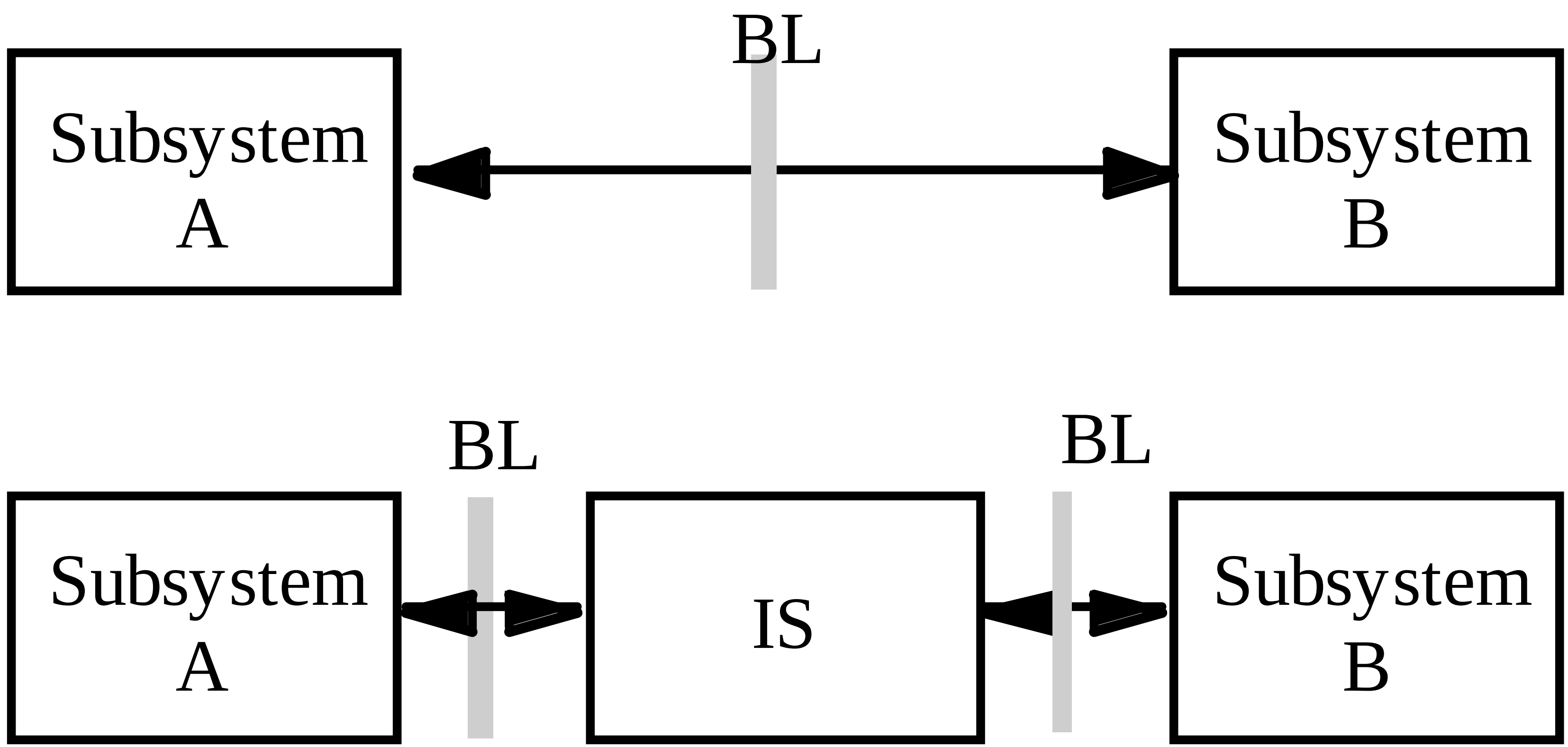
A Component may support many LIFs



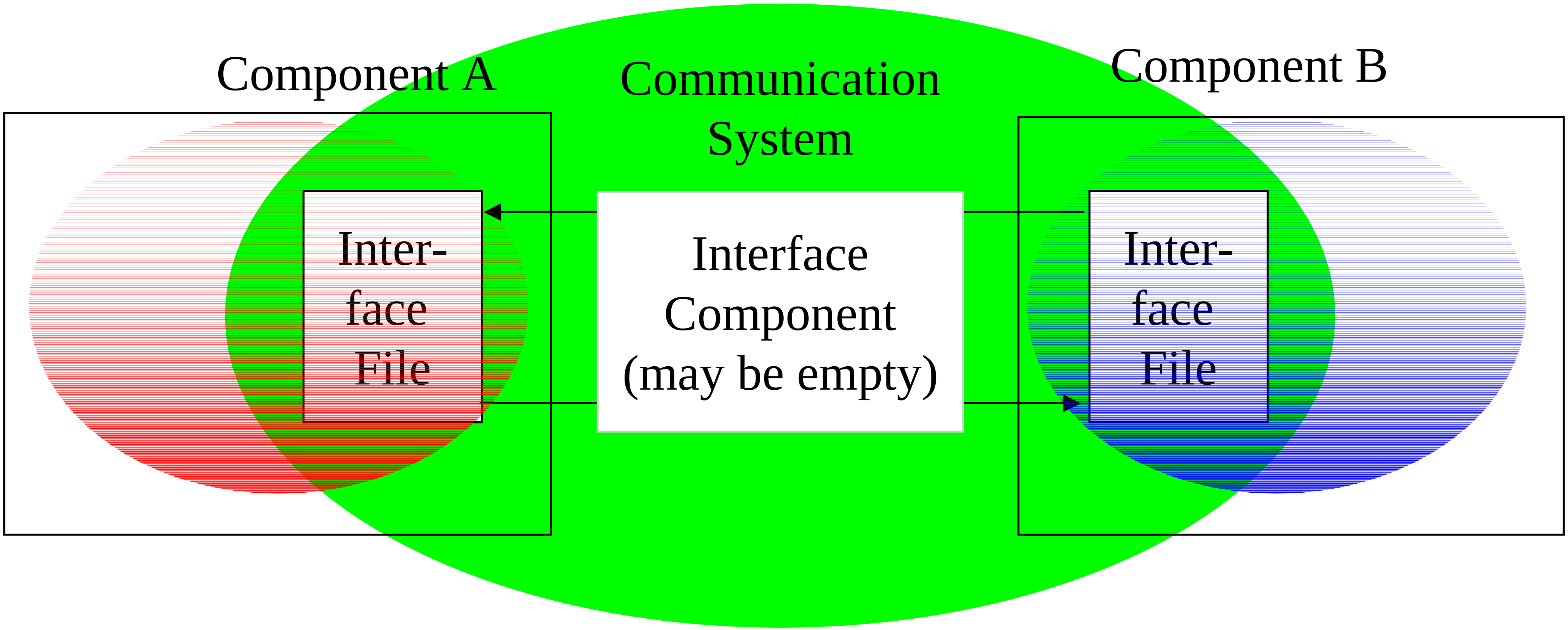
Property Mismatches at Interfaces

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

Boundary Line versus Interface System (IS)



Distributed Interface File

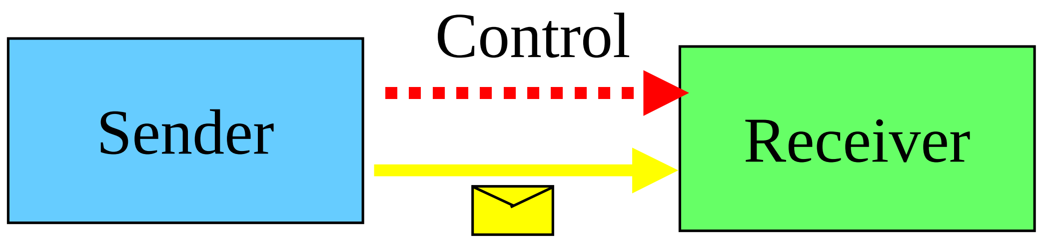


Elementary vs. Composite Interface

Consider a **unidirectional data flow** between two subsystems (e.g., data flow from sensor node to processing node).

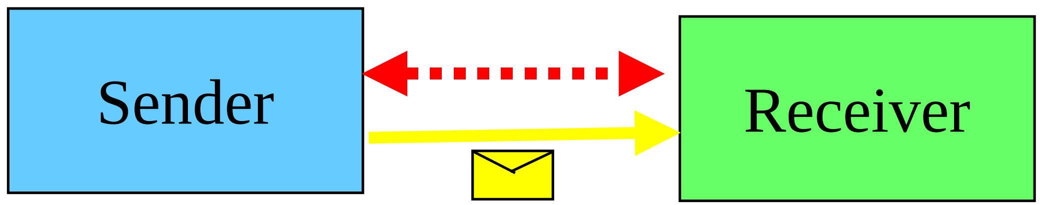
We distinguish between:

Elementary Interface:



Example: state message in a DPRAM

Composite Interface:



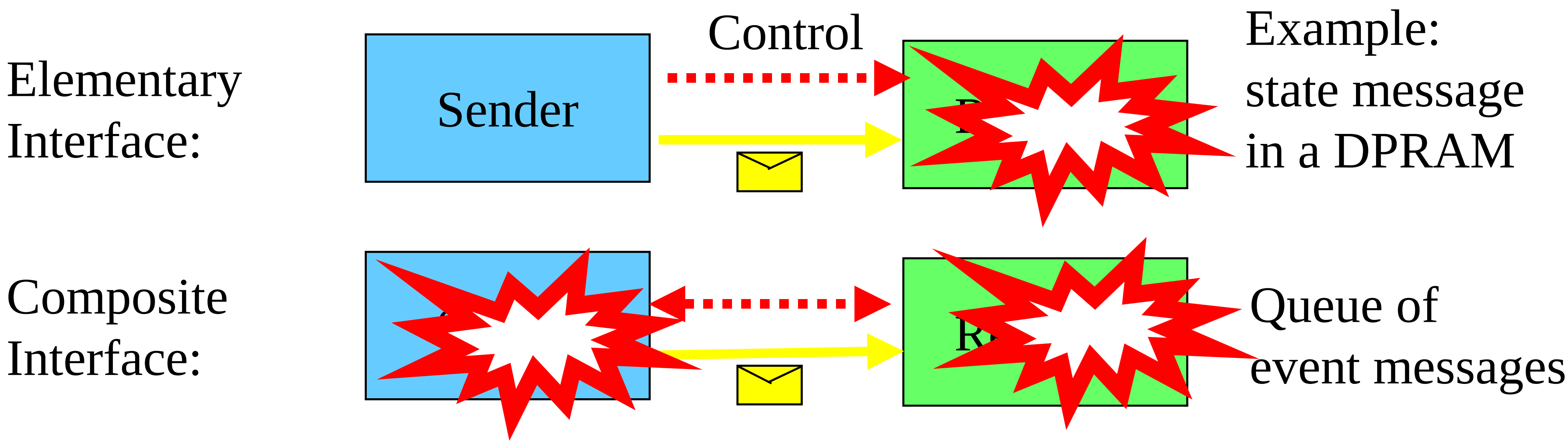
Queue of event messages

A composite Interface introduces a control dependency between the Sender and Receiver and thus compromises their independence.

Elementary vs. Composite Interface

Consider a **unidirectional data flow** between two subsystems (e.g., data flow from sensor node to processing node).

We distinguish between:



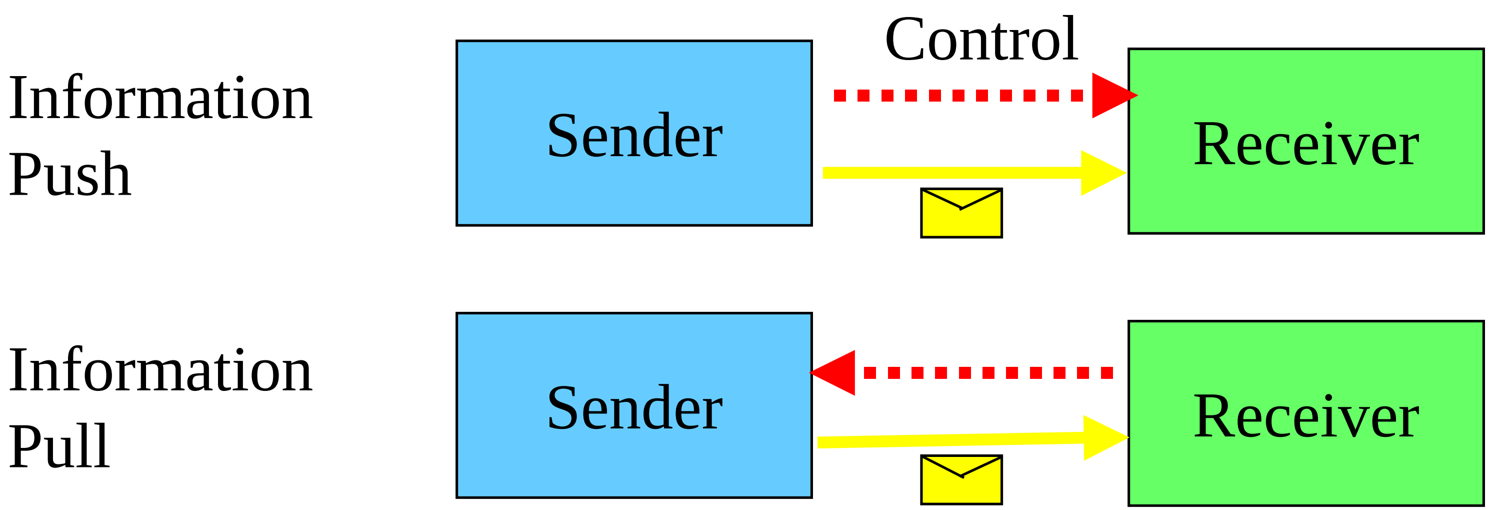
A composite Interface introduces a control dependency between the Sender and Receiver and thus compromises their independence.

Information Push vs. Information Pull

Information Push Interface: Information producer pushes information on information consumer (e.g., telephone, interrupt)

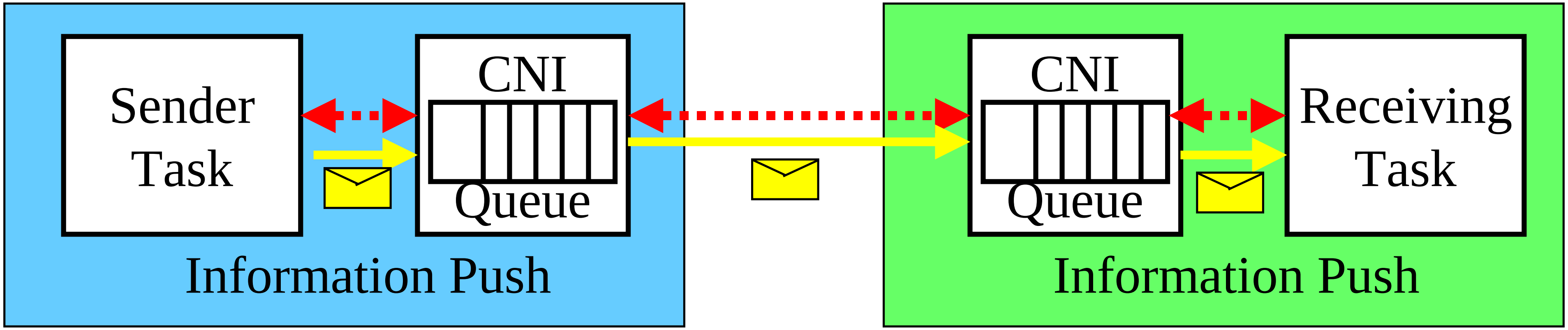
Information Pull Interfaces: Information consumer requests information when required (e.g, email).

What is better in real-time systems?--For whom?

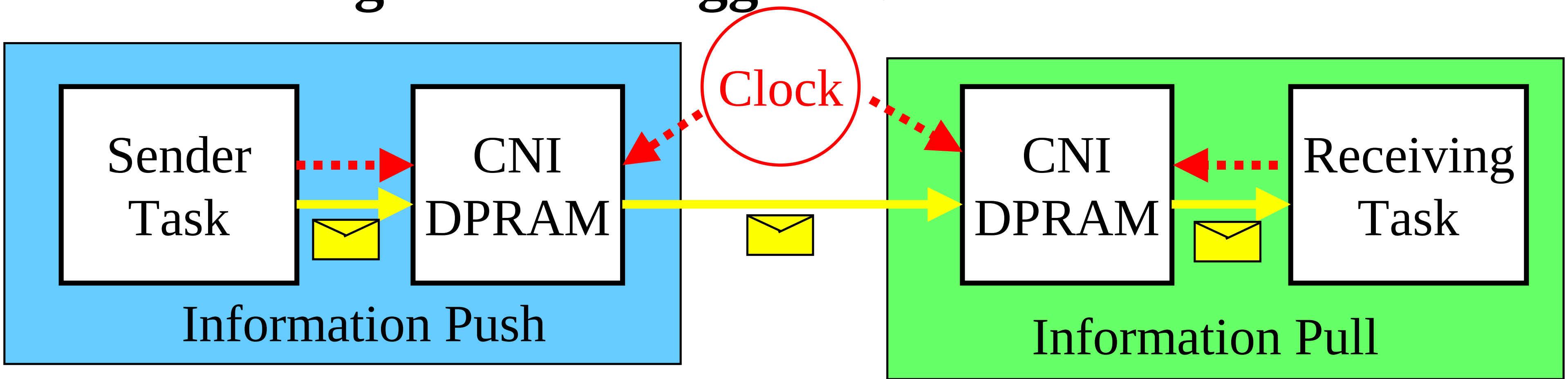


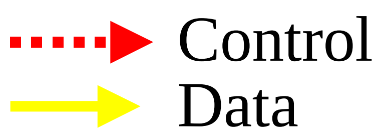
Unidirectional Information Transfer

Event-Message- Event Triggered:



State Messsage- Time Triggered:





Architecture Design *is* Interface Design

A good interface within a distributed real-time system

- is *precisely* specified in the value domain and in the temporal domain,
- provides the relevant abstractions of the interfacing subsystems and hides the irrelevant details,
- leads to minimal coupling between the interfacing subsystems,
- limits error propagation across the interface,
- Conforms to the established architectural style

and thus introduces ***structure*** into a system.