CORBA in the Time-Triggered Architecture

H. Kopetz
TU Wien
July 2003

Outline

Hard Real-Time Computing

Event and State Messages

The Time Triggered Architecture

The Marriage of CORBA with the TTA

Conclusion

What means *Hard Real-Time Computing*?

A real-time computer system must produce results before *deadlines* that are dictated by the enviroment.

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

If a catastrophe could result if a firm 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.***

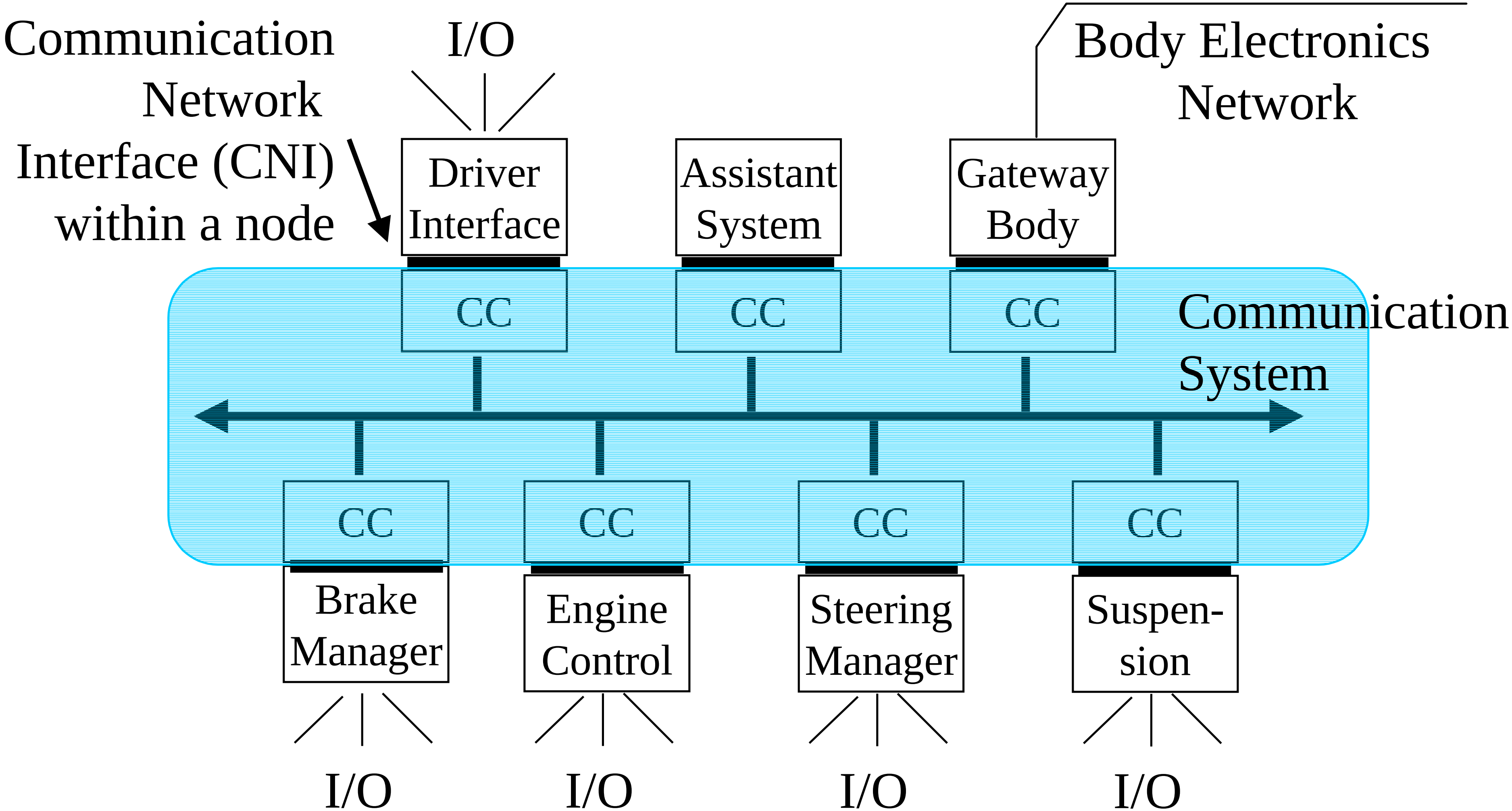
Hard- and soft real-time system design are fundamentally different.

Examples for hard real time systems: Engine Control, X-by-Wire, Nuclear Power, Flight Control, Medical Systems

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

Example of a Hard Real-Time System



CC: Communication Controller

What is a "Component"?

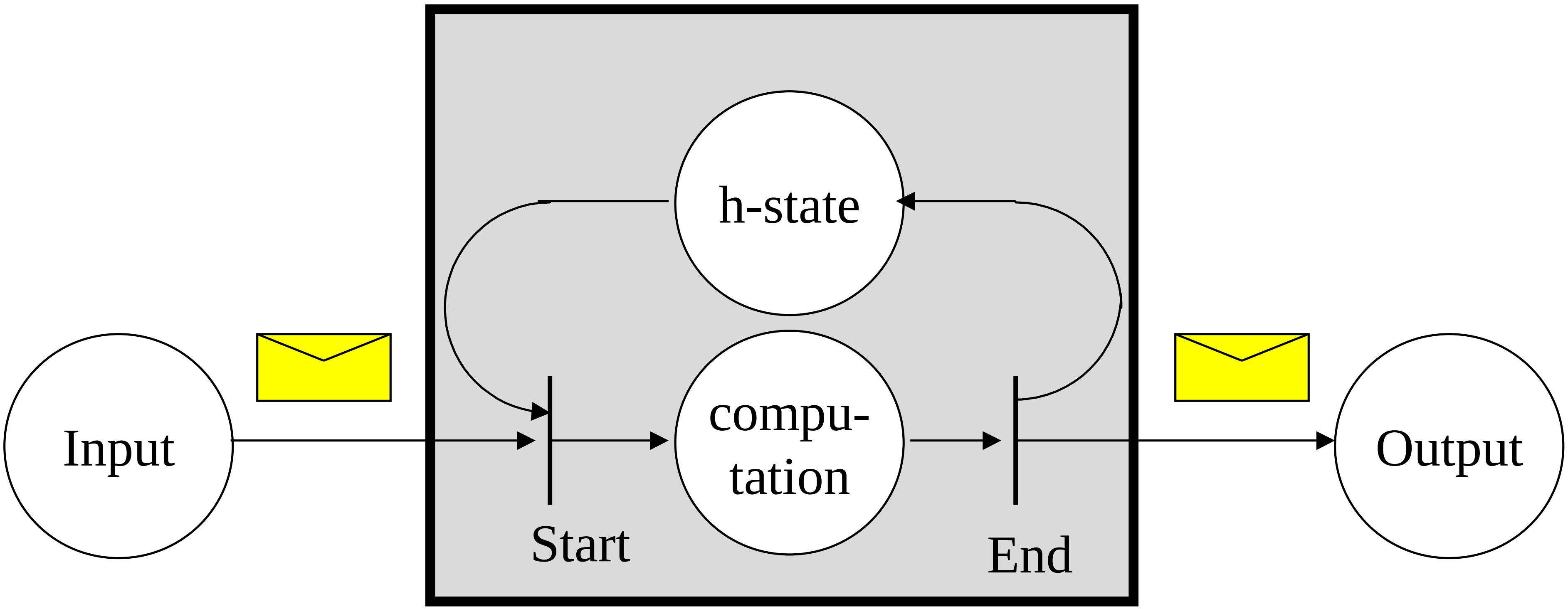
In our context, a *component* is complete computer system that is time aware and forms an independent Fault Containment Region (FCR). It consists of

- ♦ 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? Does it have state? Does it form an FCR?

Model of a Component–Messages



Messages must be correct, both in the domains of time and value.

The 10^{-9} Challenge in Hard Real-Time Systems

- **The system as a whole must be more reliable than any one of its components**: *e.g., System Dependability 1 FIT--Component dependability 1000 FIT (1 FIT: 1 failure in 10^9 hours)*
- Architecture **must support fault-tolerance** to mask component failures
- System as a whole is **not testable** to the required level of dependability.
- The safety argument is based on a **combination** of **experimental evidence** about the expected failure modes and failures rates of **fault-containment regions (FCR)** and a **formal dependability model** that depicts the system structure from the point of view of dependability.

Make Certain that Components Fail Independently

A component forms a Fault-Containment Region (FCR).

Any dependence of FCR failures must be reflected in the dependability model--a challenging task!

Independence is a system property. Independence of FCRs can be compromised by

- ♦ Shared physical resources (hardware, power supply, time-base, etc.)
- ♦ External faults (EMI, heat, shock, spatial proximity)
- ♦ Design
- ♦ Flow of erroneous messages
- ♦ Composite Interfaces

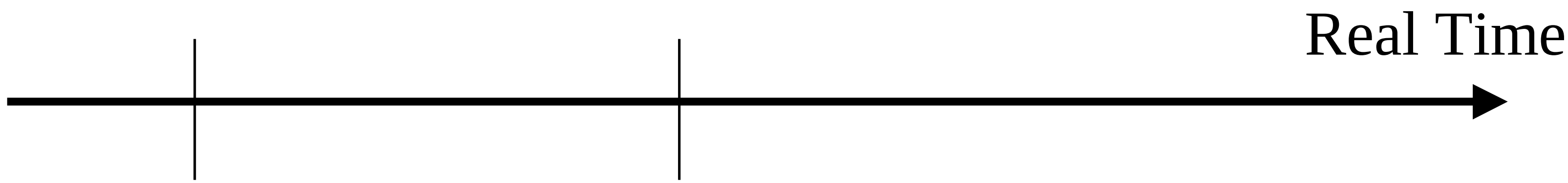
Some Important Concepts in Relation to Time

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

Instant: cut of the timeline

Duration: interval on the timeline

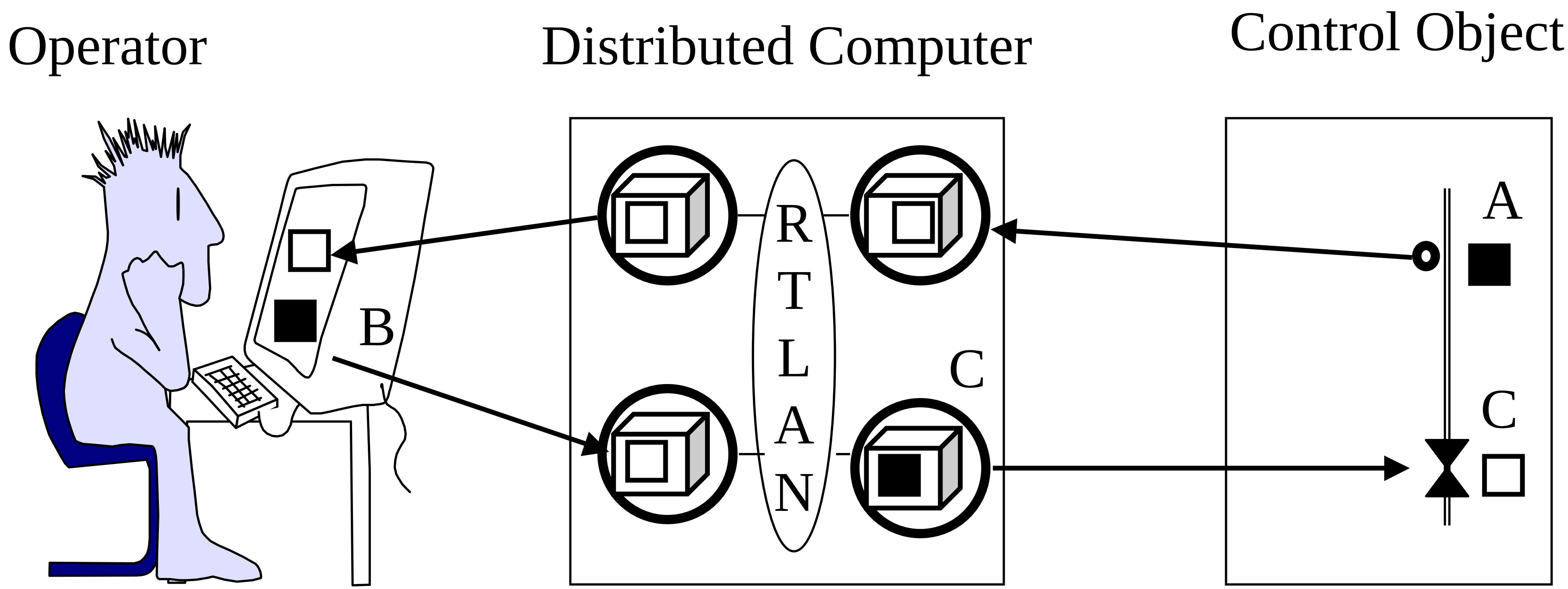
Event: occurrence at an instant--has no duration



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

Absolute Timestamp: Timestamp generated by the reference clock

RT Entities, RT Images and RT Objects



RT Entity RT Image RT Object

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

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:

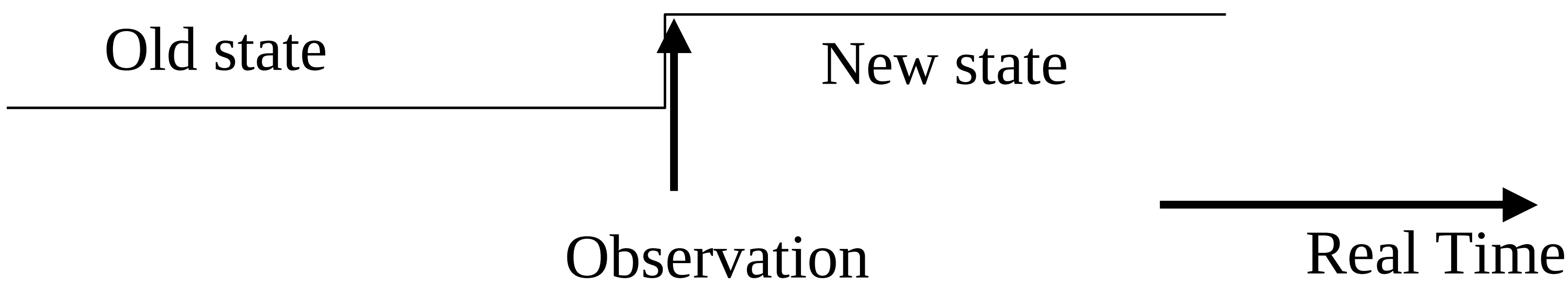
- ♦ The name of the RT-entity
- ♦ The point in real-time when the observation has been made
- ♦ The values of the RT-entity

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 *Left-event* of the "new state".



Example of State and Event Observation

State observation (blue):

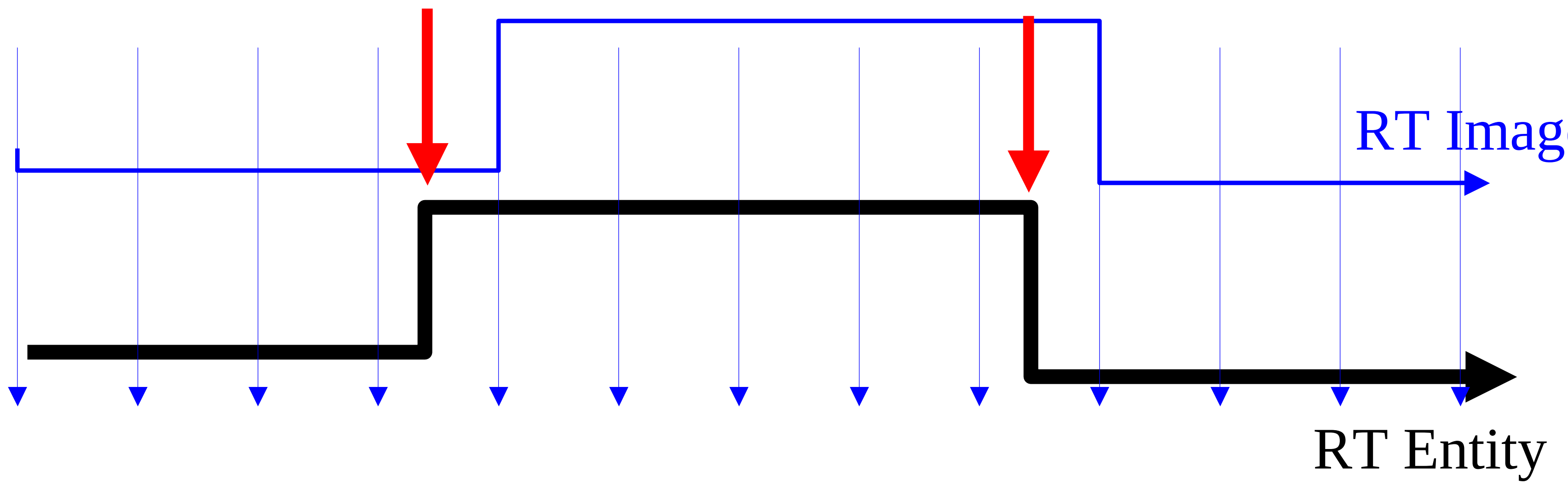
<Name of RT entity, Time of observation, full value>

The flow is at 5 l/sec a 10:45 a.m.

Event Observation (red):

<Name of Event, Time of event occurrence, state difference>

The flow changed by 1 l/sec at 10:45 a.m.



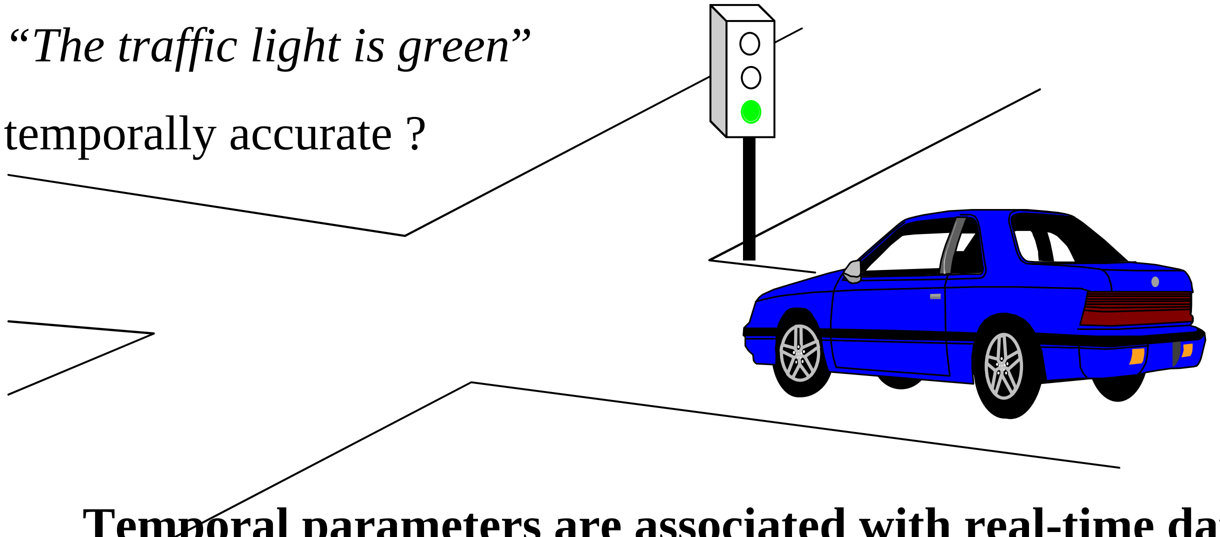
RT Image

A RT-Image is a picture of a RT Entity. A RT image is *accurate* 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.

How long is the observation:

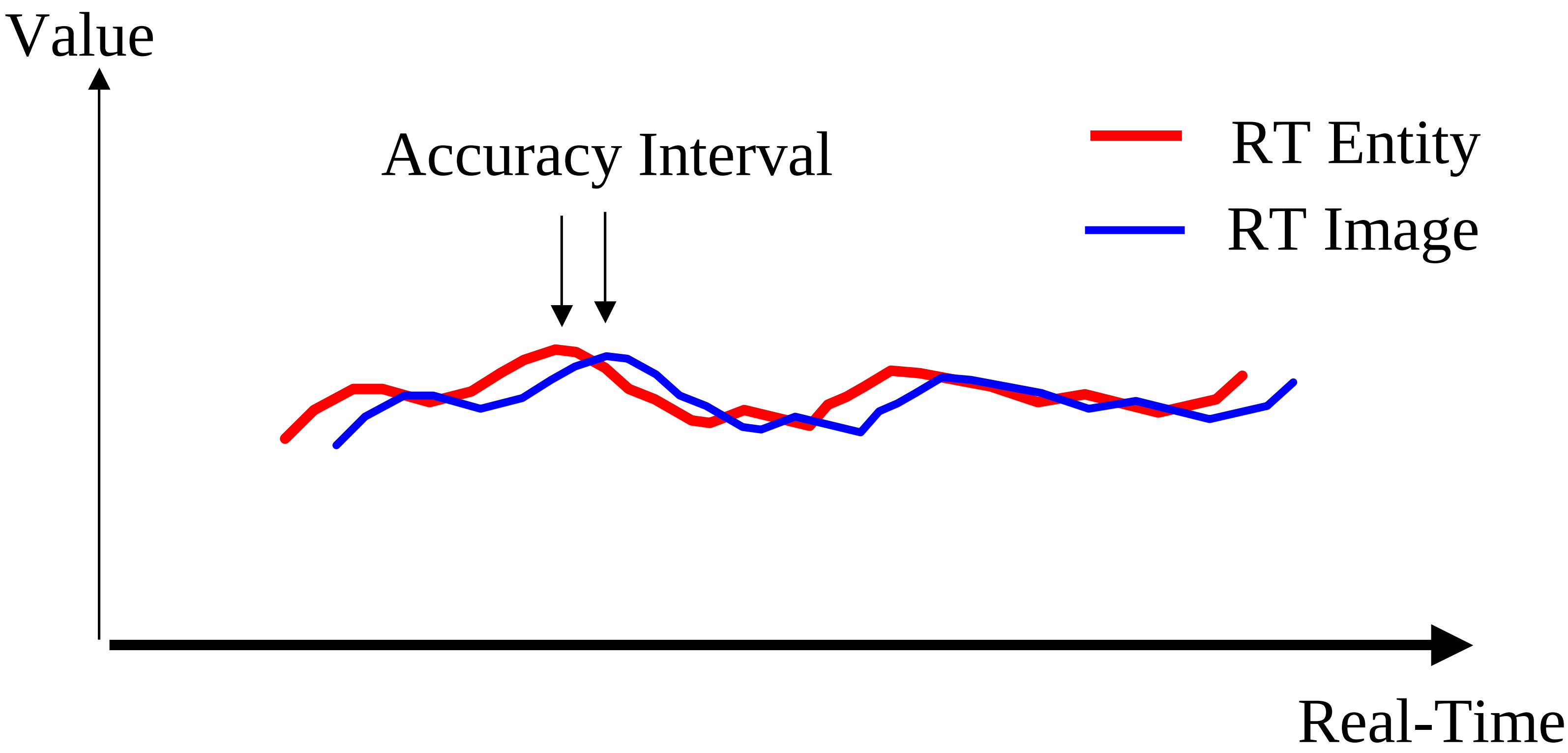
"The traffic light is green"

temporally accurate ?



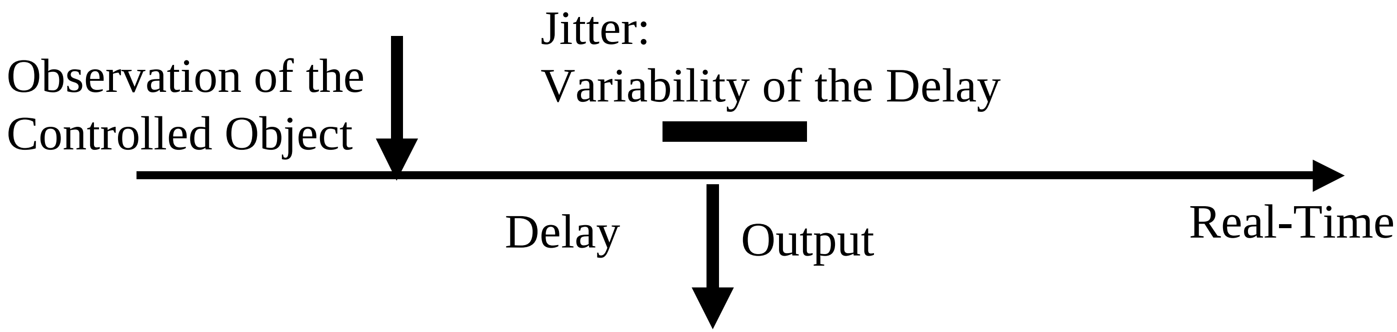
Temporal parameters are associated with real-time data.

Temporal Accuracy of an RT Image



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

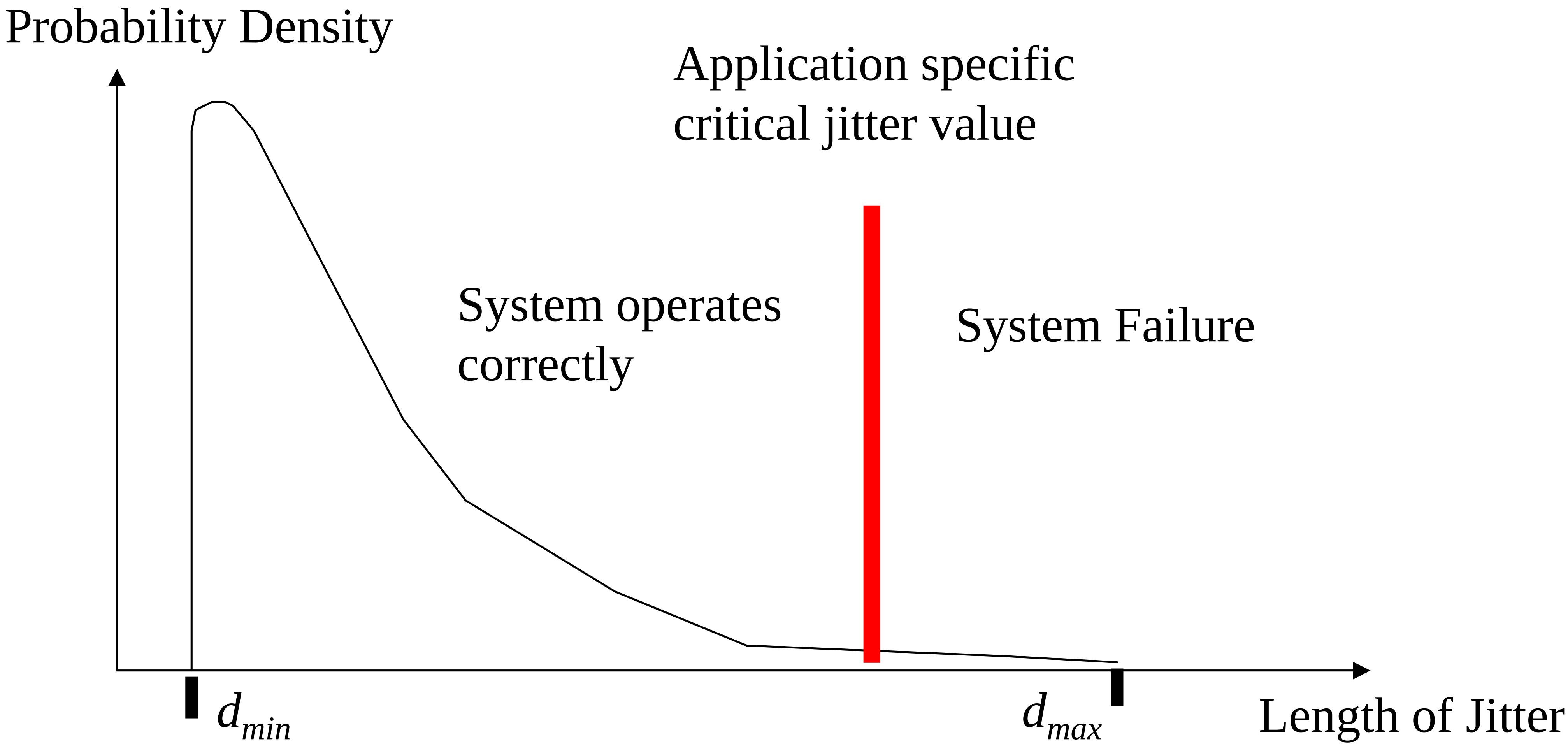
Jitter is Bad in Control Systems:



The consequences of a significant jitter:

- ♦ Measurement error increases
- ♦ Probability of Orphans
- ♦ Action Delay increases
- ♦ Clock Synchronization difficult
- ♦ Sporadic Failures in time-critical szenarios

Probability of "Long" Jitter in PAR Protocols

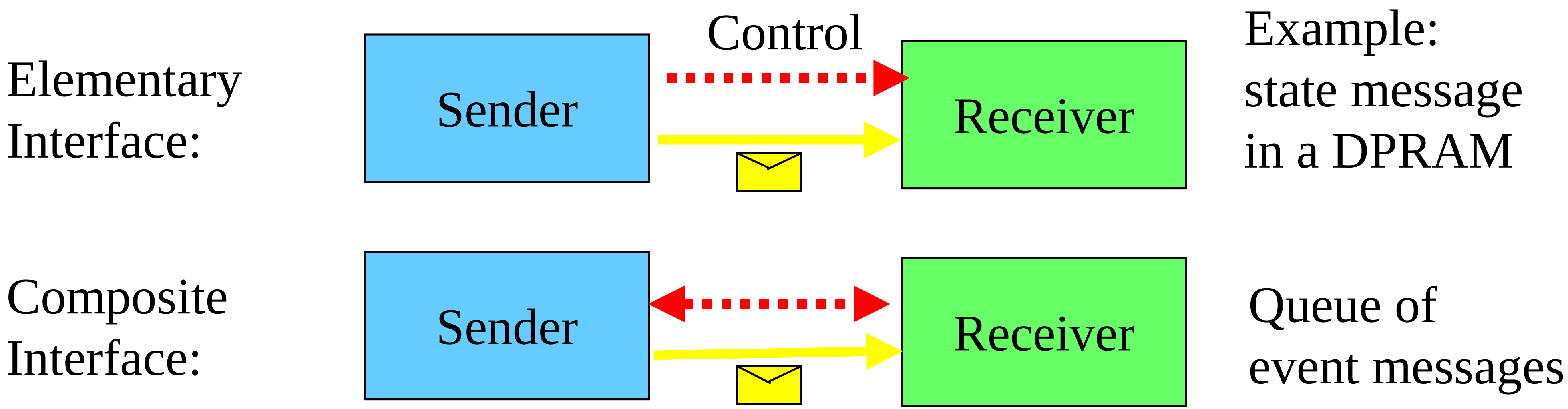


Most of the time, the system will operate correctly.

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.

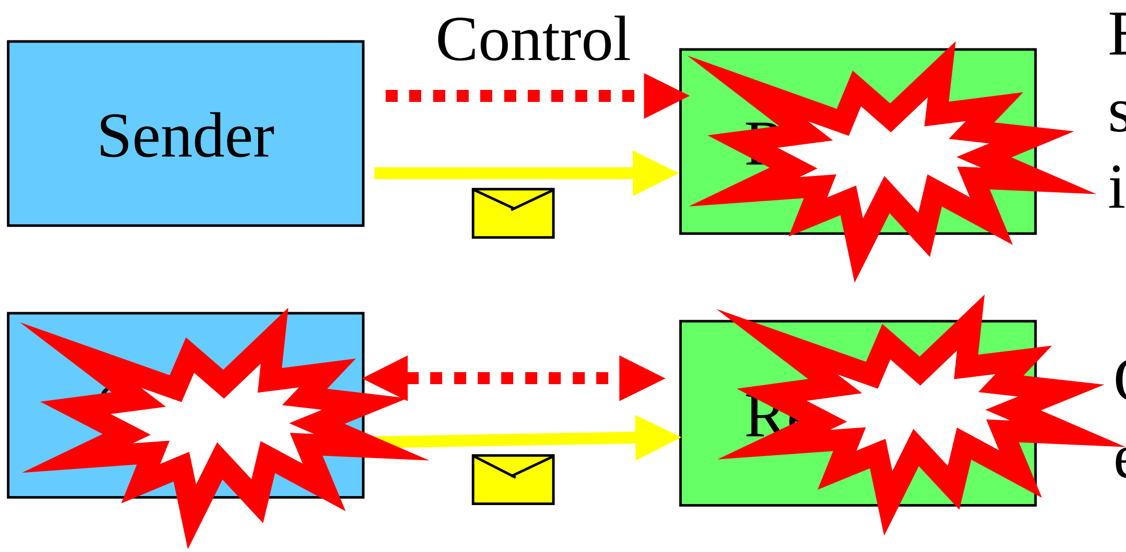
Elementary vs. Composite Interface

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

We distinguish between:

Elementary Interface:

Composite Interface:



Example: state message in a DPRAM

Queue of event messages

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

State Message versus Event Message

- **State Message**: A periodic time-triggered message that contains *state observations* (synchronous).
  Message handling: update in place and non-consuming read.
  Periodic state messages can be implemented as an *elementary* interface (*no dependence* of sender on receivers) with error detection at the receiver.

- **Event Message:** An event-triggered message that contains *event observations* (asynchronous).
  Message handling: exactly-once semantics, realized by message queues. Requires a *composite* interface (*dependence* of sender on receivers) for error detection at the sender.

(Compare “sampled message” and “queued message” in ARINC)

Examples of ET and TT Messages

ET Message:

Client Server Request

Interrupt

Alarm Message

Diagnostic Message

Flexibility, Openness. Best Effort Performance

TT Message:

Data Sampling

Control Loop

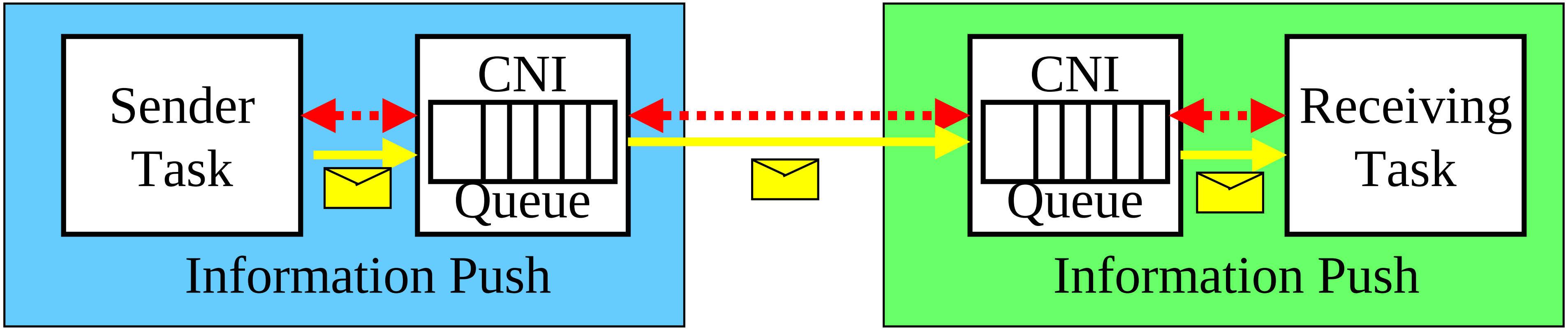
Periodic Display Refresh

Multimedia

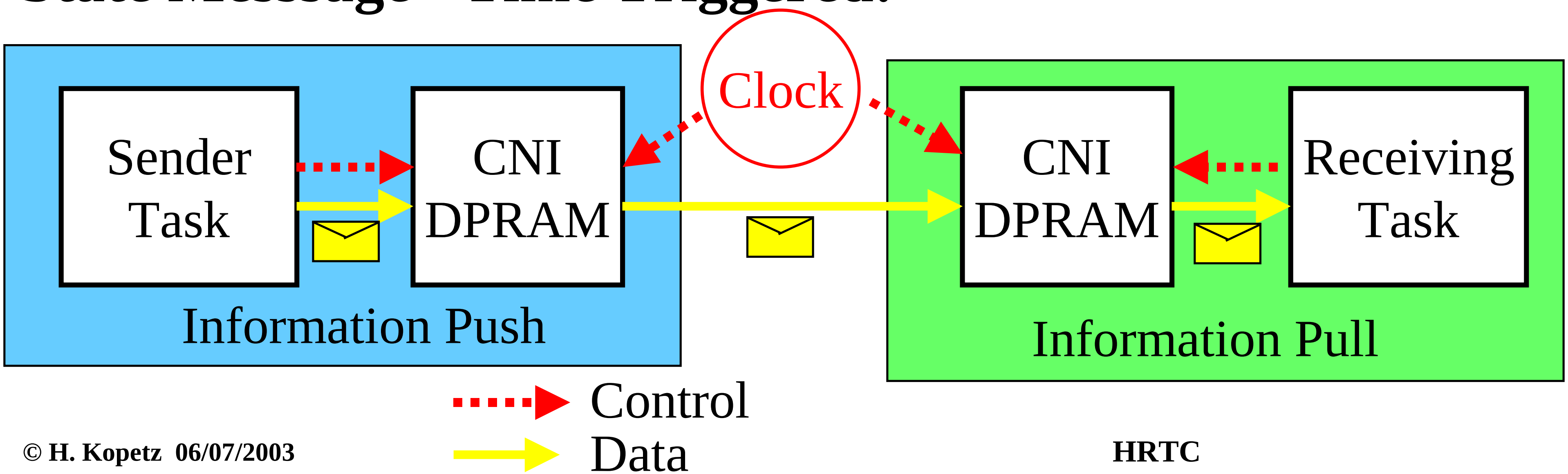
Temporal Predictability, Minimal Jitter

Unidirectional Information Transfer

Event-Message- Event Triggered:



State Messsage- Time Triggered:



Comparison Event and State Message

	Event Messsage:	**State Messsage:**
Information Type	Event Information	State Information
Temporal Pattern	Sporadic	Periodic
Semantics	Exactly Once	At least Once
Sender Access	Queue--Add Mess.	Read from DPRAM
Receiver Access	Dequeue--Consume	Overwrite DPRAM
Synchronization	Tight	Loose
Min. Temp. Distance	Unknown	Constant
Flow Control	Explicit (PAR Prot.)	Implicit
Error Detection	At Sender	At Receiver
Loss of Message	Loss of State Synchr.	Loss of Period
Control Pattern	Bidirectional (Queue)	Unidirectional
Multicast Topology	Difficult, Composite	Easy, Elementary
Jitter	Significant	Minimal

What Simplifies Open Interconnect?

	Event Messsage:	State Messsage:
Information Type	Event Information	State Information
Temporal Pattern	Sporadic	Periodic
Semantics	Exactly Once	At least Once
Sender Access	Queue--Add Mess.	Read from DPRAM
Receiver Access	Dequeue--Consume	Overwrite DPRAM
Synchronization	Tight	Loose
Min. Temp. Distance	Unknown	Constant
Flow Control	Explicit (PAR Prot.)	Implicit
Error Detection	At Sender	At Receiver
Loss of Message	Loss of State Synchr.	Loss of Period
Control Pattern	Bidirectional (Queue)	Unidirectional
Multicast Topology	Difficult	Easy
Jitter	Significant	Minimal

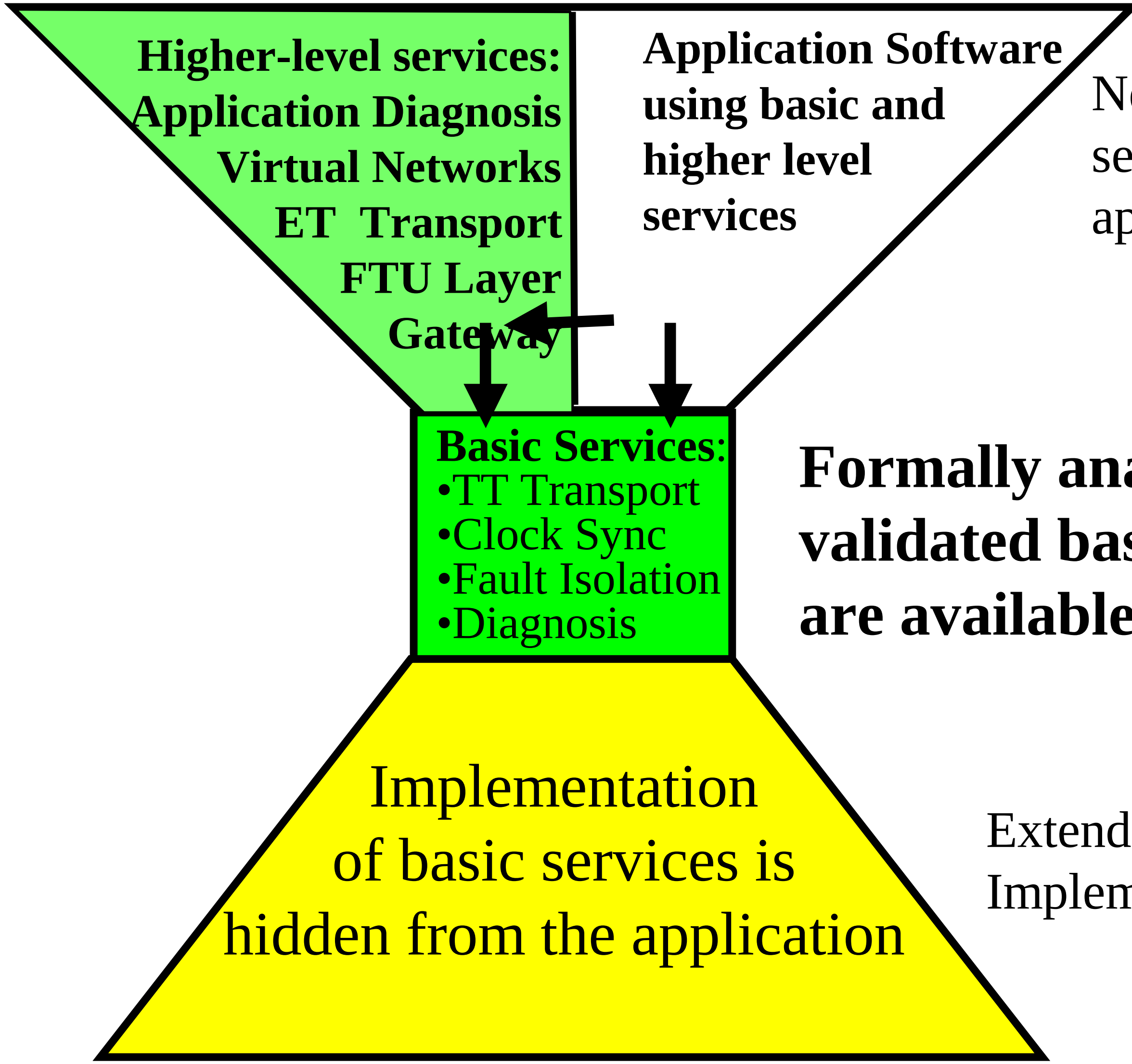
What Simplifies Fault Tolerance (TMR)?

	Event Messsage:	State Messsage:
Information Type	Event Information	State Information
Temporal Pattern	Sporadic	Periodic
Semantics	Exactly Once	At least Once
Sender Access	Queue--Add Mess.	Read from DPRAM
Receiver Access	Dequeue--Consume	Overwrite DPRAM
Synchronization	Tight	Loose
Min. Temp. Distance	Unknown	Constant
Flow Control	Explicit (PAR Prot.)	Implicit
Error Detection	At Sender	At Receiver
Loss of Message	Loss of State Synchr.	Loss of Period
Control Pattern	Bidirectional (Queue)	Unidirectional
Multicast Topology	Difficult	Easy
Jitter	Significant	Minimal

Preliminary Conclusion

- In hard real-time systems, we need **both services**
  - Event Message Transport
  - State Message Transport
- Time-critical information should be transported in State Messages
- Event Messages provide open and flexible mechanisms for the transport of *non-time critical* information
- The Architecture must support analytical reasoning about its safety, since ultradependability is not testable.

The TTA is Waist-line Architecture



New high-level services to ease application development

Formally analyzed and validated basic services are available and stable

Extend the range of Implementation choices

Basic Services versus High-Level Services

The TTA distinguishes between four basis services and an open-ended set of high-level services. The basic services are:

(1) Time Triggered Transport of Messages

(2) Fault-Tolerant clock synchronization

(3) Strong Fault-Isolation Services

(4) Diagnostic service

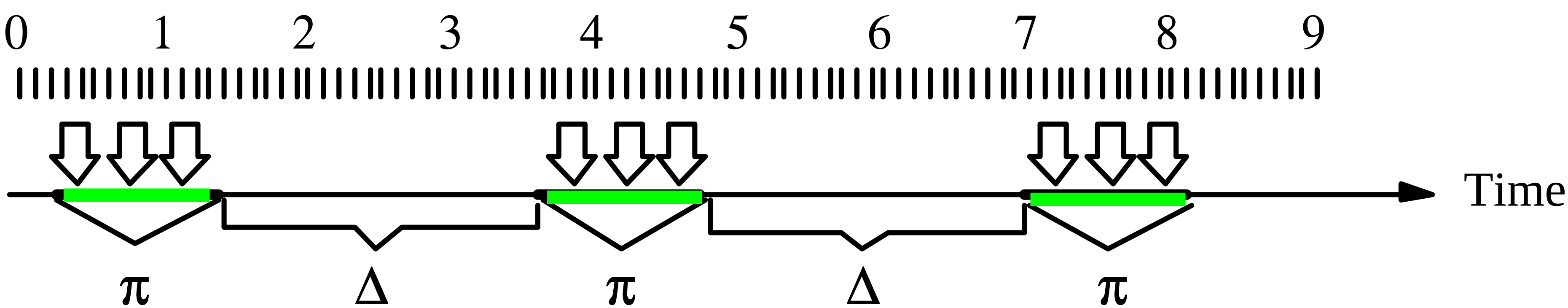
The high level services depend on the basic services, while the basic services do not depend on the high-level services!

Basic Service 1: Message Transport by TTP/C

- TTP (Time-Triggered Protocol) generates a fault-tolerant global time-base.
- Media access is controlled by TDMA, based on this time. ET messages are piggy-packed on the basic TT messages.
- Information identified by the common knowledge of the send/receive times.
- Two independent intelligent star couplers provide fault isolation in the temporal domain.
- Membership service to detect crash/omission (CO) failures. Also used to detect violations of the fault hypothesis.

Basic Service 2: Fault-Tolerant Sparse Time Base

If the occurrence of events is restricted to some active intervals with duration π with an interval of silence of duration Δ between any two active intervals, then we call the timebase π/Δ-sparse, or sparse for short.



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

Basic Service 3: Fault Isolation

In the Time-Triggered Architecture Fault-Containment Regions (FCRs) communicate by the exchange of messages:

- ♦ In a properly configured system, any FCR (node) can fail in an without disrupting the operation of the nodes that have not been directly affected by the fault.
- ♦ Error Detection in the **Time Domain** is in the responsibility of the architecture. It is performed by independent replicated guardians which are part of the architecture.
- ♦ Error Detection in the **Value Domain** is in the responsibility of the fault-tolerance layer or of the application (e.g., by TMR), supported by **post condition checks** at the guardians.
- ♦ TTP/C contains also a clique avoidance service, based on a membership service to detect a violation from the fault hypothesis.

Basic Service 4: Diagnosis

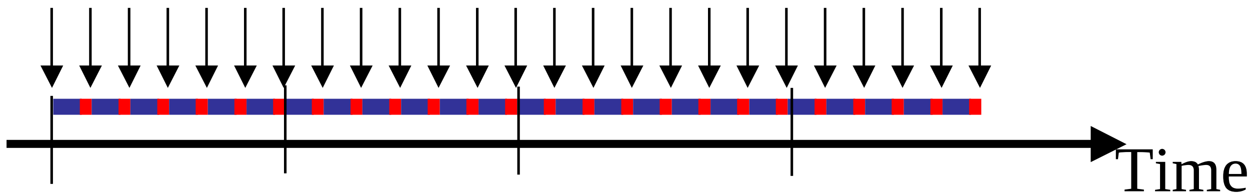
The TTP/C membership service checks continuously, which node is alive and which node has failed. It monitors the correctness of the distributed computing base.

- ♦ The periodic TT message of each node is interpreted as a life sign of the sender.
- ♦ In order to distinguish between a sender fault and a receiver fault, the view of a third node is considered to be the judge (single fault assumption)
- ♦ Delay of the membership service < 2 TDMA rounds.

HL Service: ET Transport

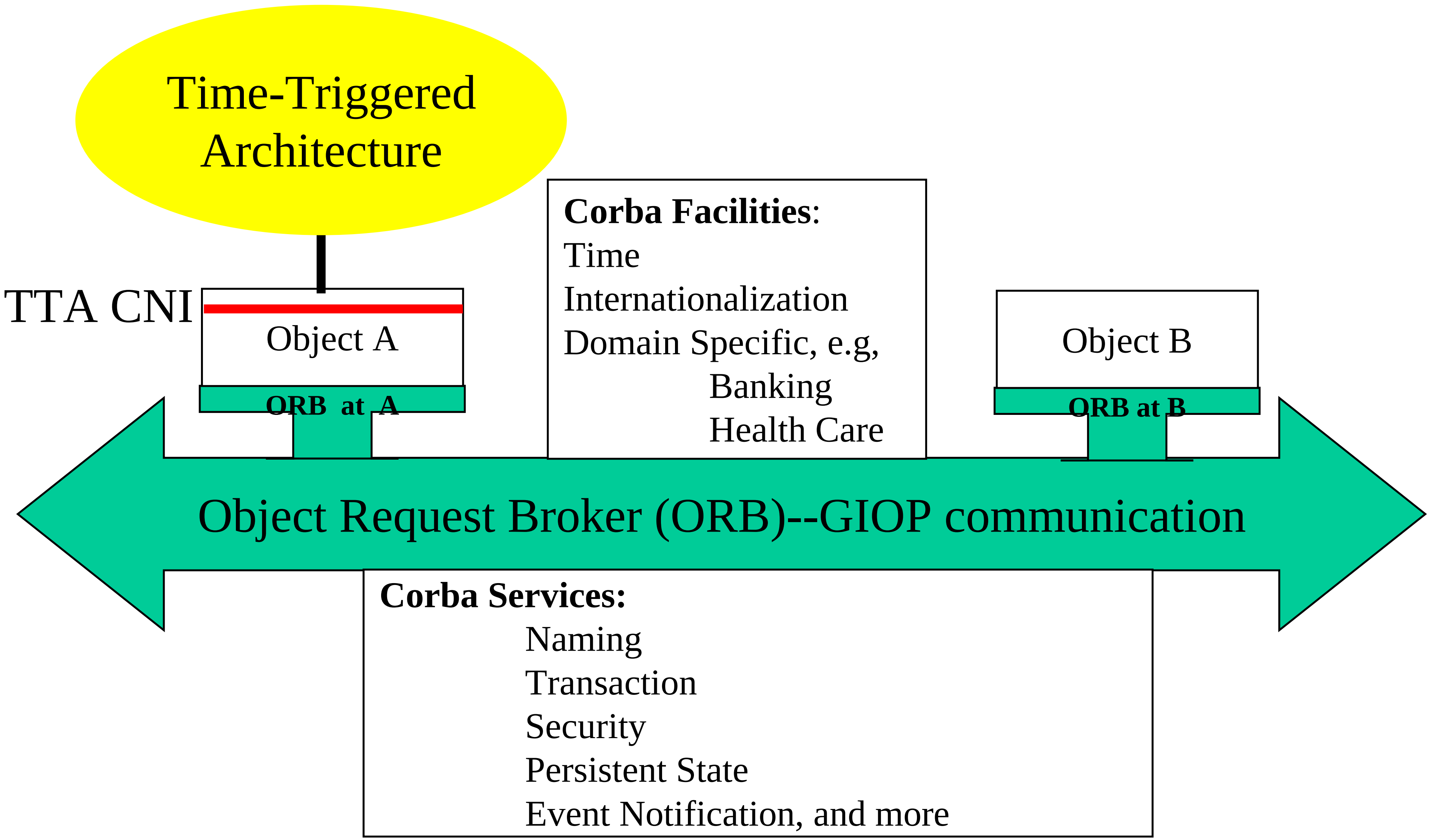
Layered: ET service is implemented on top of a TT protocol
Single time triggered access media access protocol.

Maintains
Temporal
Composability



The CAN Protocol and the TCP/IP Protocol have been implemented on top of basic TTP/C in order to be able to use legacy software and to support the integration of CORBA.

HL Service: CORBA Integration



Constraints on a Proposed Solution in CORBA

- Interoperability with traditional ORBs
- Provide Hard Real-Time capabilities by supporting State Message Transport in addtion to Event Message Transport
- Provide Composability
- Support of Fault Tolerance
- Support Analytical Reasoning about Dependability at the level of the base Architecture
- Should be viable on embedded systems (small footprint)

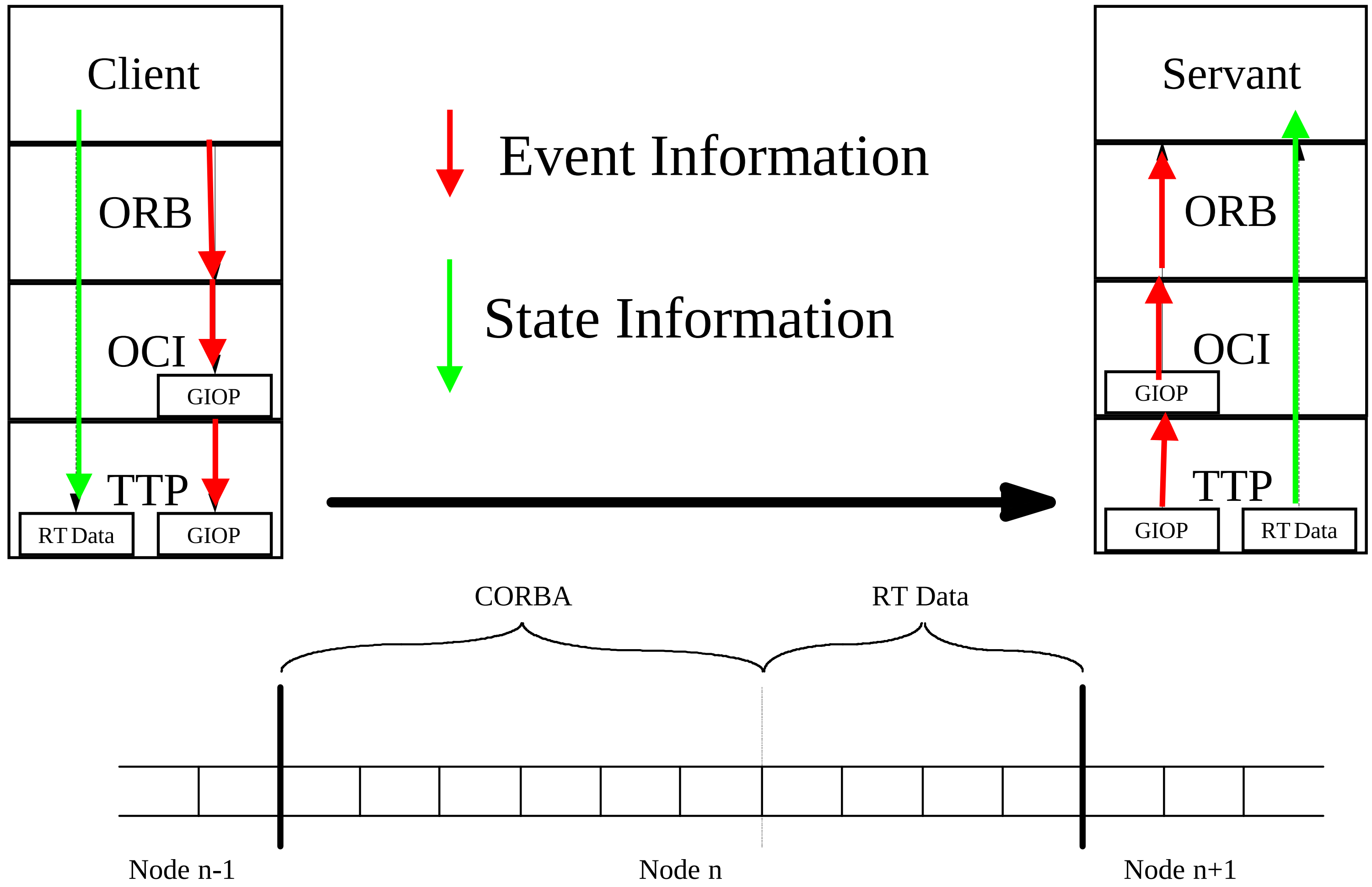
Proposed Solution

- No changes required at ORB
- Application dependent Extensible Transport Plugin (could be generated by a tool)
- Additional Overhead in the Extensible Transport Plugin (can be neglected if CPU power is significantly greater than network performance)
- For a prototype we use the Open Communication Interface (OCI) as Extensible Transport

Proposed Mechanisms

- Communication Infrastructure provides both ET Message Channel and TT Message Channel
- IIOP works over ET Message Channel without any modification.
- Extensible Transport Plugin on Client‘s Side decides if information is available locally or must be requested from the remote CORBA object.

Flow of State and Event Information in CORBA



Delay and Jitter of a TT Message

At present TT implementations up to 25 Mbit/second are available:

This implementations achieves:

- ♦ TDMA round (8 nodes) about 1 msec
- ♦ Transport delay about 125 μsec
- ♦ Jitter about 1 μsec

Delay and Jitter at the application level depend on the internal structure of the node local operating system and middleware.

Conclusion

The proposed integration of CORBA in the TTA (time-triggered architecture) as developed within the HRTC project provides:

- ♦ An architecture which meets the safety requirements of ultradependable hard real-time application.
- ♦ The seamless integration of this architecture into the open information infrastructure by providing full compatibility with exisiting CORBA standards.
- ♦ A new mechanism for the transport of time-critical information within dedicated CORBA subsystems,