

Introduction to Deterministic Networks

H Kopetz

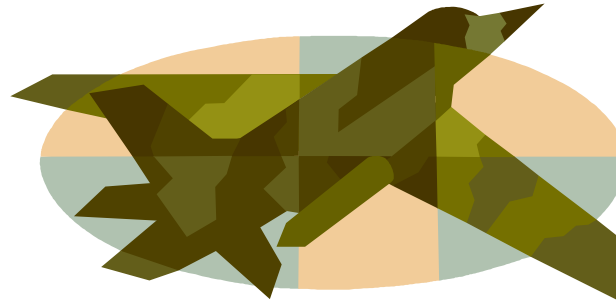
TU Wien

□ October 2013

- ◆ Introduction
- ◆ Distributed Real-Time Systems
- ◆ Timeliness
- ◆ Determinism
- ◆ Fault Containment and Fault Masking
- ◆ Composability
- ◆ Outlook

Focus on Safety Critical Systems--No Backup

Fly-by-wire Airplane: There is no mechanical or hydraulic connection between the pilot controls and the control surfaces.



Drive-by-wire Car: There is no mechanical or hydraulic connection between the steering wheel and the wheels.



No Backup--the 10^{-9} Challenge

- ◆ **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 be distributed and 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 units (FCU)** and a **formal dependability model** that depicts the system structure from the point of view of dependability.
- ◆ **Independence** of the FCUs is a critical issue.

Distributed Real-Time Systems

Dependable Real-Time Systems are Distributed

There are four main reasons for the distribution of intelligence in dependable embedded systems:

- ◆ Mask failures by the application of redundancy.
- ◆ Partition a large system into a set of smaller subsystems in order to reduce the cognitive complexity.
- ◆ Bring local intelligence to the sensors and actuators:
 - Complexity reduction by encapsulating subsystems
 - Reduce the number of cables and cabling points, thus reducing the weight and increasing the reliability
 - Simplification of Installation and Maintenance
- ◆ Avoid spatial proximity: independent fault-containment regions.

. . . we must support the communication among *subsystems of differing criticality* over the same physical wire, otherwise we will be overwhelmed by wires.

RT-Protocol is at the Center of an Architecture

- ◆ At the Center of a Distributed Architecture is the Communication Protocol.
- ◆ The properties of the protocol determine *timeliness*, *composability*, *error containment*, etc.
- ◆ Many of today's RT Protocols have been designed bottom up with limited architectural vision.
(Sometimes with the intention to lock a customer to a supplier).
- ◆ Technology trends force a consolidation.

Mitigation of Soft Errors by Architectural Means

Architectural means to mitigate the consequences of component failures might become a necessity when using the upcoming submicron devices, as stipulated in the 2005 *International Roadmap of Semiconductors* p.6:

Relaxing the requirement of 100% correctness for devices and interconnects may dramatically reduce the costs of manufacturing, verification and test. Such a paradigm shift is likely forced in any case by technology scaling, which leads to more transient and permanent failures of signals, logic values, devices and interconnects.

Present State of RT Communication

- ◆ Many different Protocols
 - Time-Triggered: TTP, FlexRay, TT-CAN, TT Ethernet
 - Fieldbus: 1553, CAN, LIN, Profibus, etc.
 - Ethernet based: AFDX, Powerlink, EtherCAT, etc.
 - Others: MOST, Firewire, Spacewire, etc.
- ◆ At the moment, different protocols are used by different communities.
- ◆ *Glue software* and *gateways* are widely used to achieve some level of interoperability

Technology Trends that Force a Consolidation

- ◆ Interoperability of systems--integration of systems into the *Internet of Things (IoT)*
- ◆ Pressing need to reduce power, number of nodes, spare parts and cables
- ◆ Economies of scale of the semiconductor industry--an SoC cannot support ten different protocols
- ◆ Security in embedded systems
- ◆ Education and limited human resources
- ◆ Investments in software and tools
- ◆ **Increasing number of *Soft-Errors in Semiconductors* suggest system-level fault masking techniques**

Requirements of a RT-Communication System

- ◆ Predictable Transmission of Real-Time Data
 - Small Latency (Timeliness)
- ◆ Fault Containment and Fault Masking
 - Determinism
- ◆ Support of System Integration (Composability)
 - Precise Interface Specification: Value and Time
- ◆ Provision of a Precise Physical Time
 - IEEE 1588 Precision Time Protocol

Timeliness

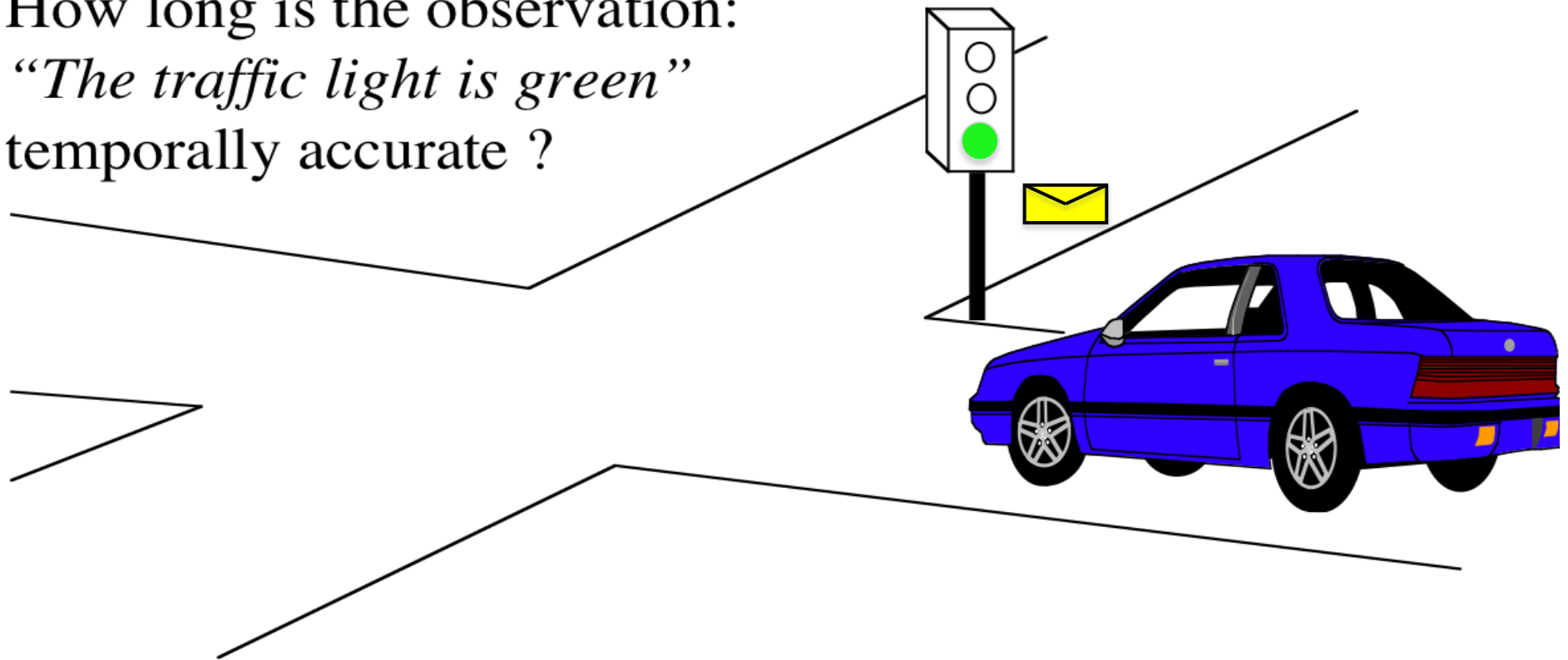
Need for a *Global Physical Time*



A valuable lesson from the August 14 blackout is **the importance of having *time-synchronized system data recorders***. *The Task Force's investigators labored over thousands of data items to determine the sequence of events, much like putting together small pieces of a very large puzzle. That process would have been significantly faster and easier if there had been wider use of synchronized data recording devices.* From Final Report on US-Canadian August 14, 2003 Power Blackout, p.164.

RT Information has limited temporal validity

How long is the observation:
“The traffic light is green”
temporally accurate ?



An appropriate model of RT communication must consider
timeliness as important as *correctness*.

Open-World System

An *open-world system* is a system where an (*unknown*) number of *uncoordinated* clients *compete* for the services of a server (*example: Internet*). In such a system there is always the possibility that *n senders* send messages to the *same receiving port* at the *same instant* resulting in a *temporal conflict*.

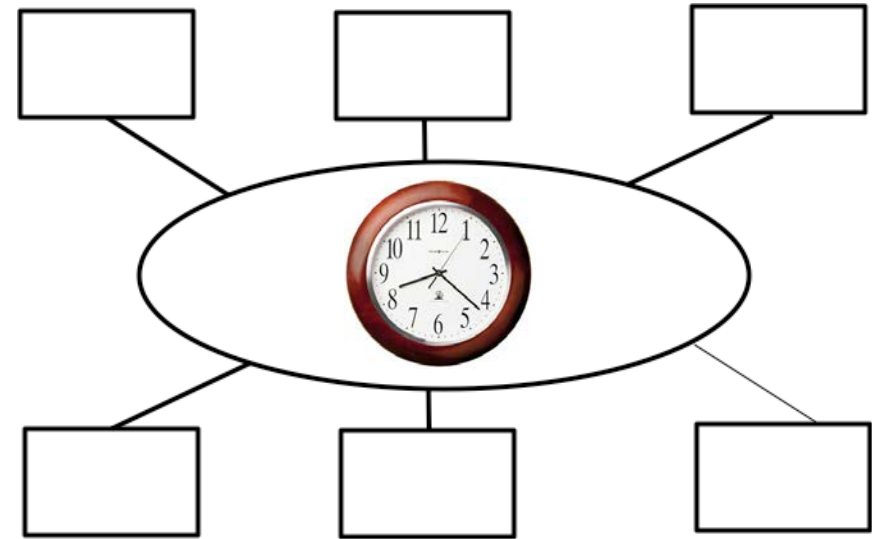
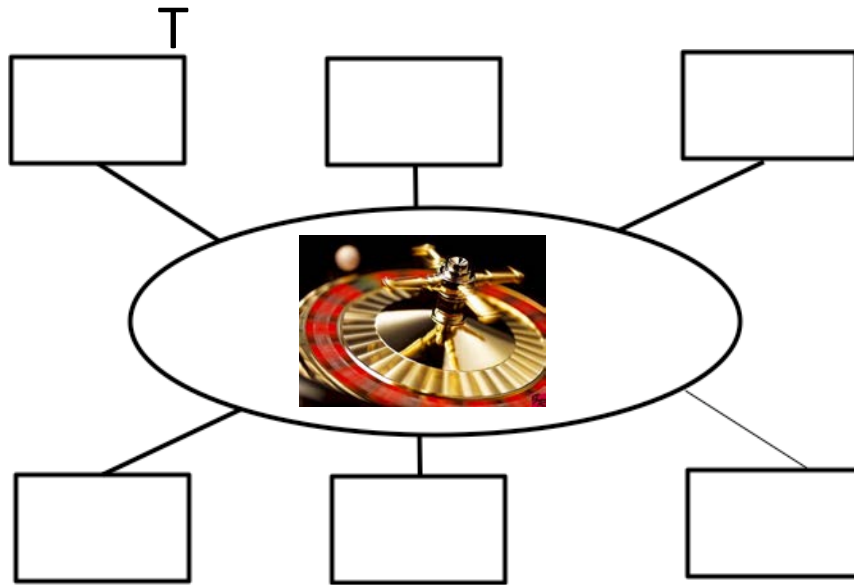
There two alternatives to resolve this temporal conflict:

- ◆ Exercise *back-pressure flow control* on the *n-1 senders* that did not succeed in getting the message through or
- ◆ Deliver *one message* and store the *n-1 other messages* in the network.

Both alternatives are unsuitable for real-time messages.

Closed-World System

- ◆ In a *closed-world system* the number of clients is *limited and known a priori*. The clients *cooperate* with each other such that the server is in the position to meet the requests of all clients within specified temporal bounds.
- ◆ **Temporal guarantees can only be given in a closed-world system.**



Competition **versus**
Event Triggered
Probabilistic

Cooperation
Time Triggered
Deterministic

Timeliness: Distinguish between

Event Triggered (ET)-Messages: (no global time needed)

- ◆ A message is sent, whenever a significant event occurs (e.g., completion of a task, arrival of an interrupt)
- ◆ Open World Assumption—possible conflicting send instants
- ◆ No Guarantee of Timeliness

Time-Triggered (TT)-Messages: (global time needed)

- ◆ An *a priori* planned cycle is associated with every TT message type. A TT message is sent, whenever the global time reaches the start instant of the cycle of this message type.
- ◆ Closed World Assumption—Preplanned conflict-free send instants.
- ◆ Guaranteed *a priori* known latency—Determinism.

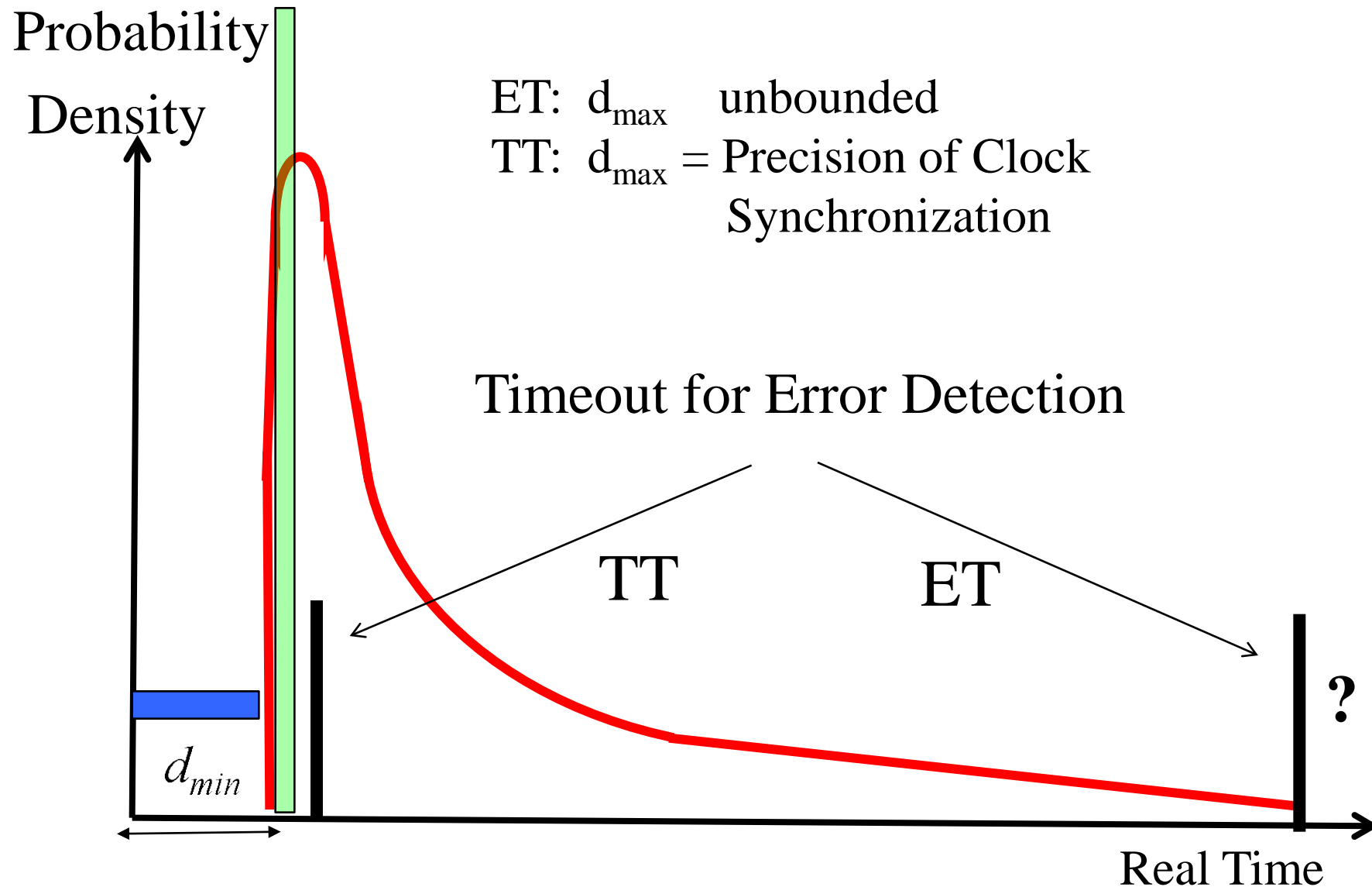
Coordination by Referring to a Global Time

A simple way to coordinate the transmission of hard real-time messages takes advantage of a global time-base:

- ◆ Precision of the Global Time Base determines the inter-message gap.
- ◆ Global Time must be dependable, no reliance on any single clock.
- ◆ Global Time can be used for many other purposes, eg.
 - Consistent time-stamping of data
 - Check the validity of real-time data
 - Coordinate output actions (**Timed Output Action**)
 - Perform prompt error detection

A real-time protocol must provide a clock synchronization service.

Timeliness: Event-Triggered vs Time-Triggered



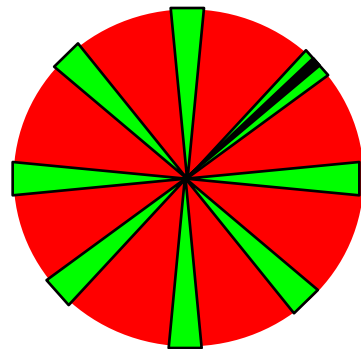
Models of Time in a CPS



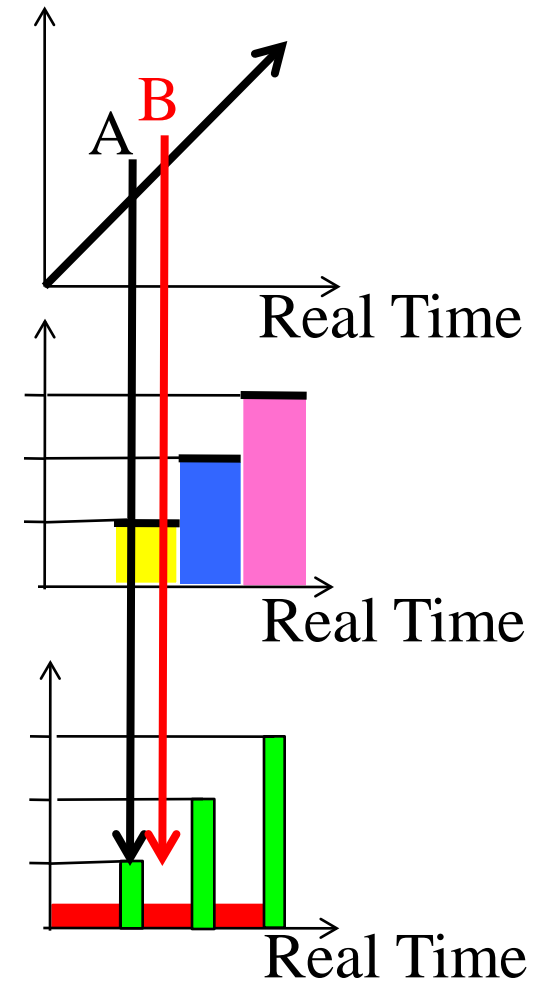
Dense
Physics



Discrete
Central Computer



Sparse
Distributed
Computer



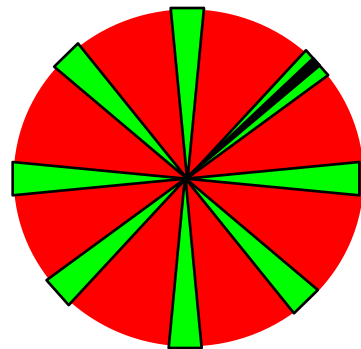
Models of Time in a CPS



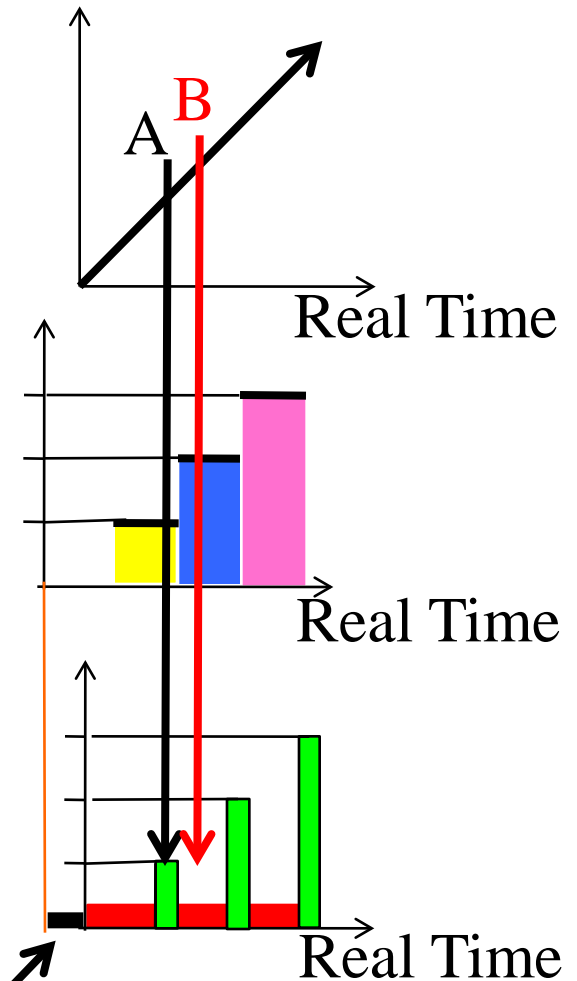
Dense
Physics



Discrete
Central Computer

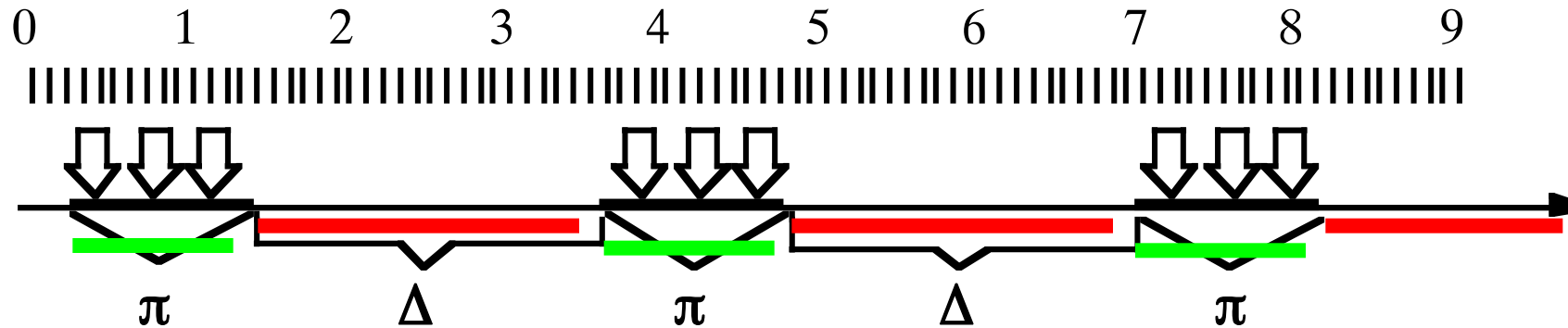



Sparse
Distributed
Computer



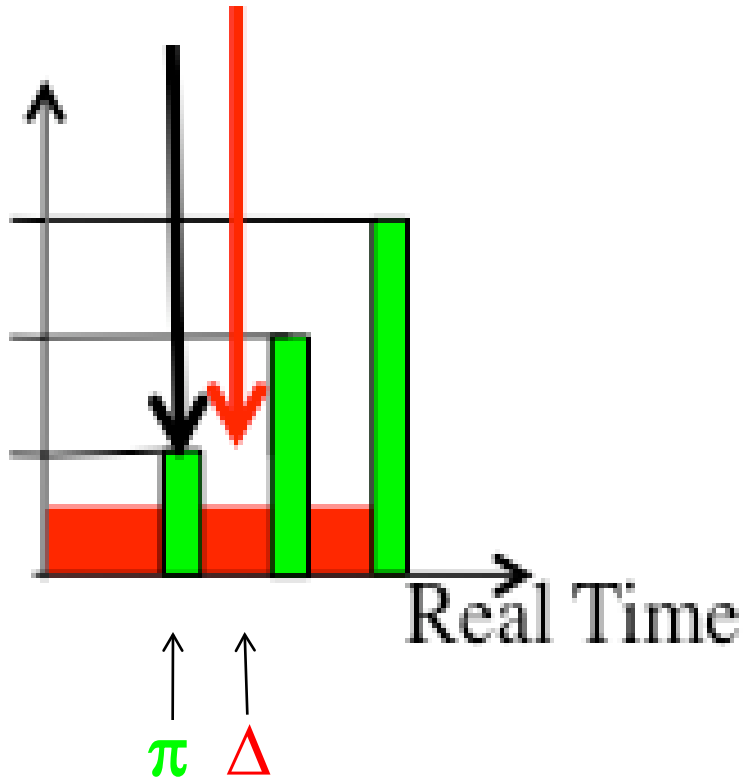
Sparse Time Model

If the occurrence of events is restricted to some active intervals on the timeline with duration π with an interval of silence of duration Δ between any two active intervals, then we call the time base π/Δ -sparse, or *sparse* for short, and events that occur during the active intervals *sparse events*.



Events  are only allowed to occur at subintervals of the timeline

The Intervals π and Δ in a *Sparse* Timebase

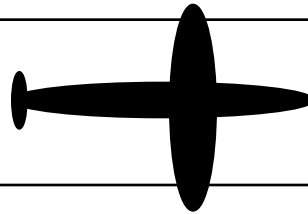


- Depend on the *precision* P of the clock synchronization.
- In reality, the precision is always larger than zero—in a distributed system clocks can never be fully synchronized.
- The precision depends on the stability of the oscillator, the length of the resynchronization interval and the accuracy of interval measurement.
- *On a discrete time-base, there is always the possibility that the same external event will be observed by a tick difference.*

Determinism

Replica Determinism: Airplane on Takeoff

Consider an airplane that is taking off from a runway with a flight control system consisting of *three independent channels* without a global time. Consider the system at the *critical instant* before takeoff:



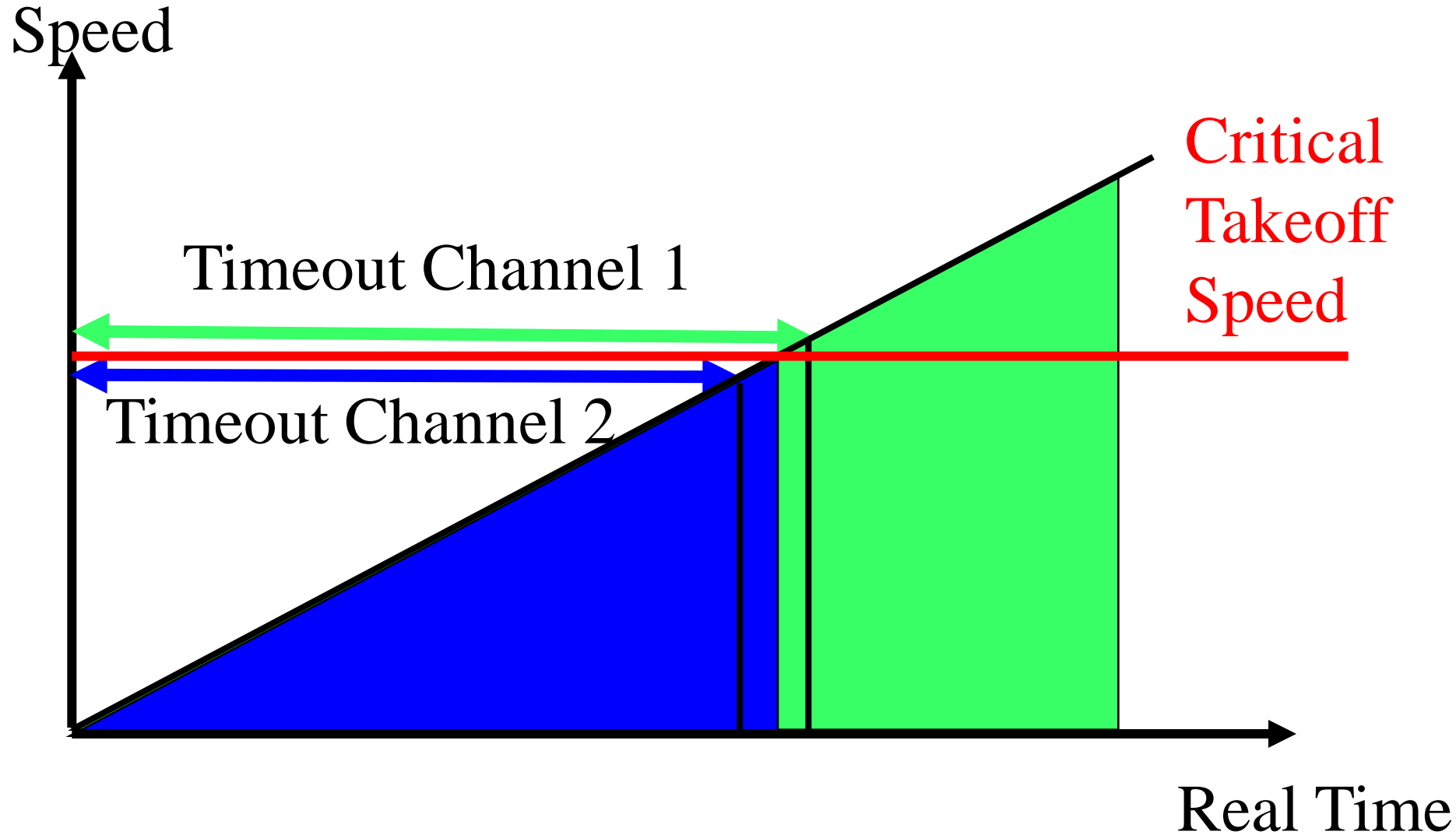
Channel 1 Take off

Accelerate Engine

Channel 2 Abort

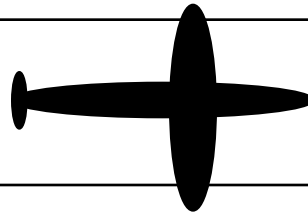
Stop Engine

The Critical Role of Time



Replica Determinism: Airplane on Takeoff

Consider an airplane that is taking off from a runway with a flight control system consisting of *three independent channels*. Consider the system at the *critical instant* before takeoff:



Channel 1 Take off

Accelerate Engine

Channel 2 Abort

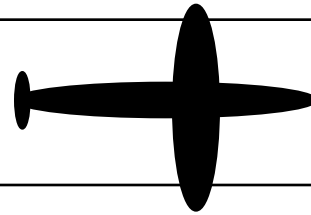
Stop Engine

Channel 3 Take off

Stop Engine (Fault)

Replica Determinism: Airplane on Takeoff

Consider an airplane that is taking off from a runway with a flight control system consisting of *three independent channels*. Consider the system at the *critical instant* before takeoff:



Channel 1	Take off	Accelerate Engine
Channel 2	Abort	Stop Engine
Channel 3	Take off	<i>Stop Engine (Fault)</i>
<i>Majority</i>	Take off	<i>Stop Engine (Fault)</i>

Determinism of a Communication Channel

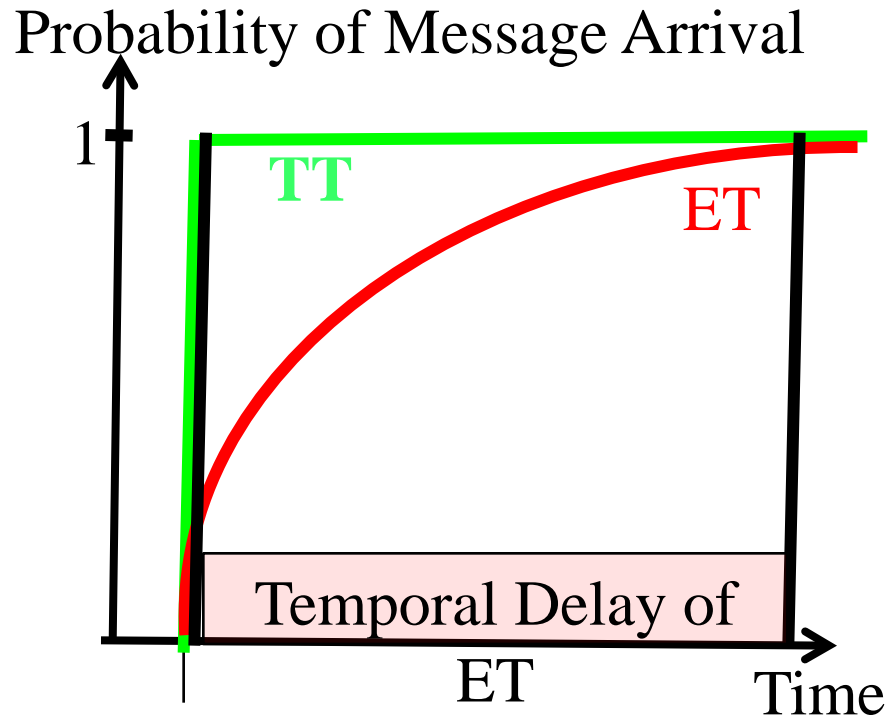
The behavior of a communication channel is called *deterministic* if (as seen from an omniscient external observer):

- ◆ A message is delivered before an *a priori* known instant (timeliness).
- ◆ The *receive order* of the messages is same as the *send order*. The *send order* among all messages is established by the *temporal order* of the *send instants* of the messages as observed by an omniscient observer.
- ◆ If the *send instants* of n ($n > 1$) messages are the *same*, then an order of the n messages will be established in an *a priori* known manner.

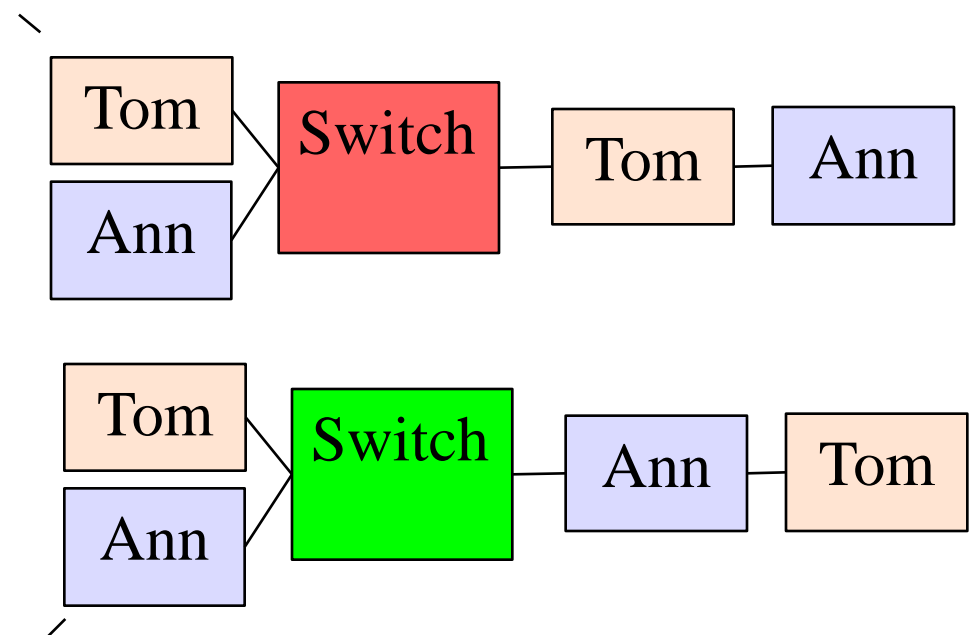
Two *independent* deterministic channels will deliver messages *always* in the same order before an *a priori* known instant.

Determinism: *Timeliness* and *Order*

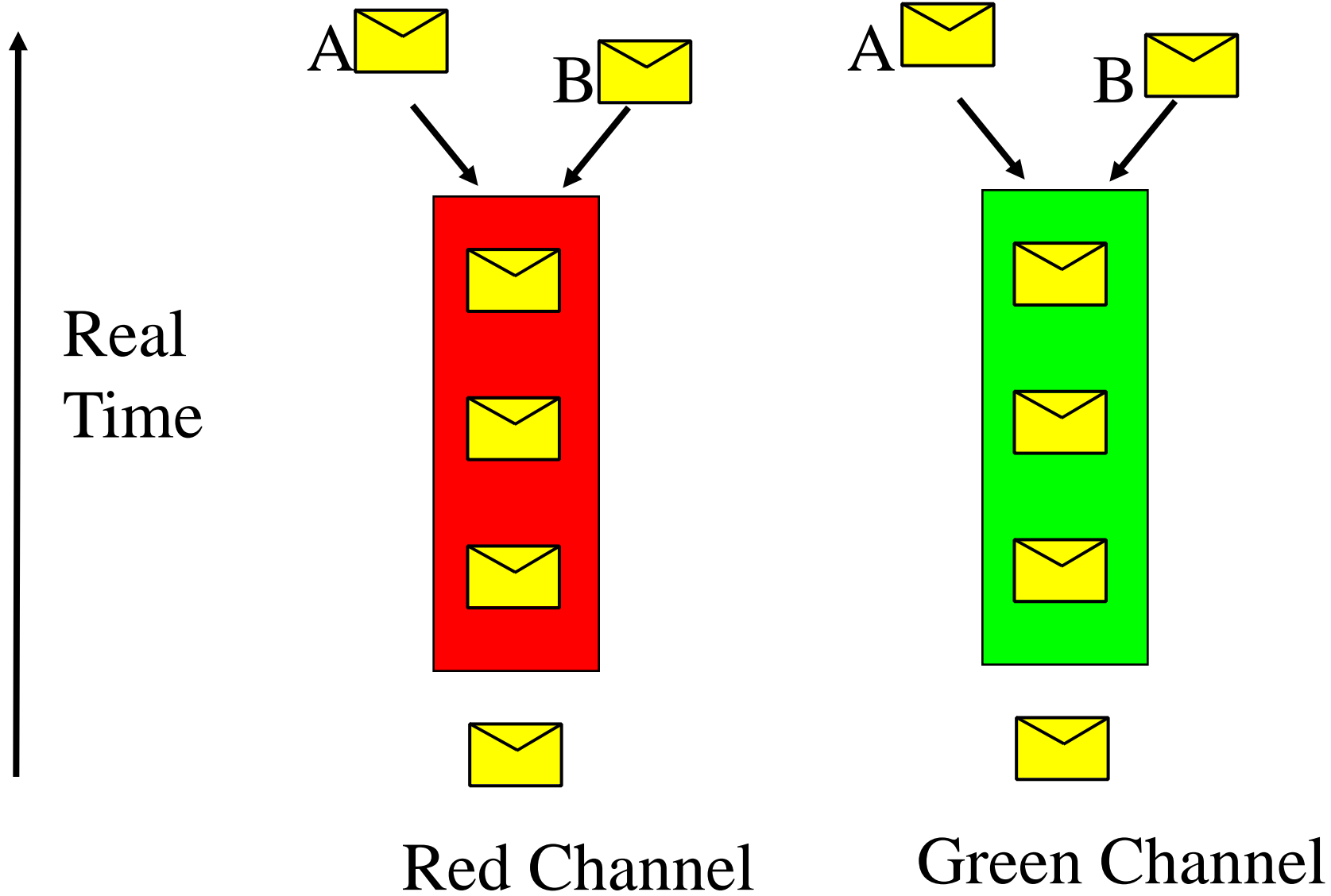
Timeliness



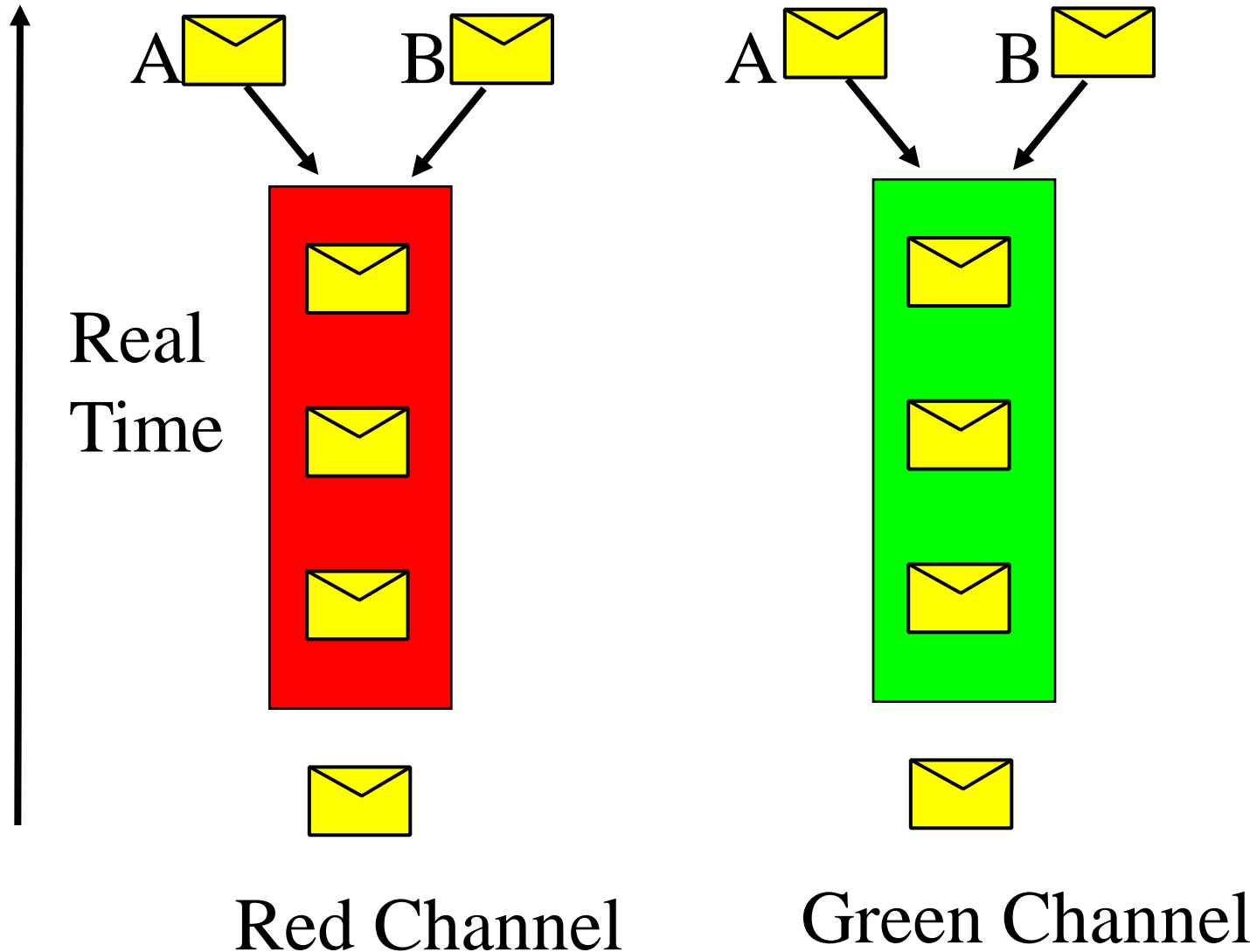
Consistent Temporal Order



Temporal Order is Obvious



Determinism: Simultaneity--Who Wins?



Determinism:

If A wins on the red channel

then A must also win on the green channel

Handling of *Simultaneity*—A Fundamental Problem³⁴

In the hardware : *meta-stability*

In operating systems: *mutual exclusion*

In communication systems: *order of messages*

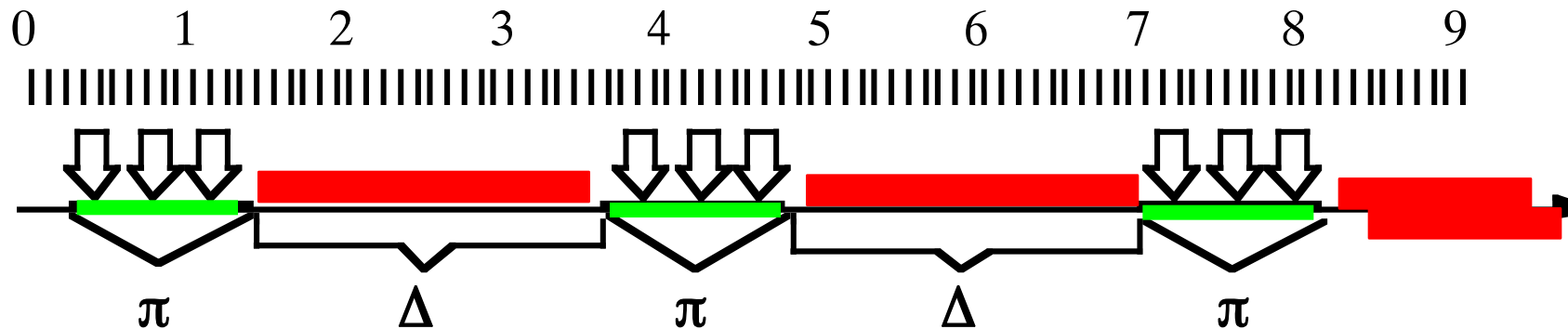
A two-step solution:

- (i) Provide consistent view of simultaneity—distinguish between events that are in the sphere of control (SoC) of the system and events that are outside the SoC--*difficult*
- (ii) Order simultaneous events according to some *a priori* established criterion--*easy*

Events *inside* the SoC: Generate *Sparse Events*

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 time base π/Δ -sparse, and the events that occur in the active intervals *sparse events*.

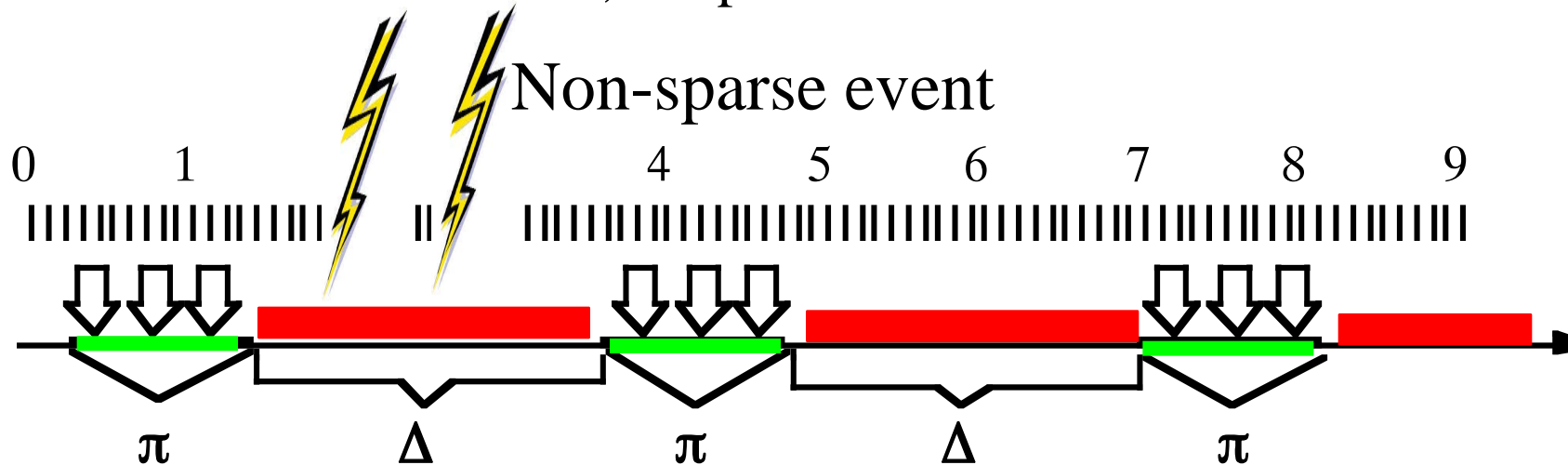
π/Δ are determined by the precision of the clock synchronization



Events  are only allowed to occur at subintervals of the timeline

Events *outside* the SoC: Agreement Protocols

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 time base π/Δ -sparse, or sparse for short.



Events  are only allowed to occur at subintervals of the timeline

Agreement Protocol to Generate *Sparse Events*

To arrive at a consistent view of the temporal order of non-sparse events within a distributed computer system (which does not necessarily reflect the temporal order of event occurrence), the nodes must execute an *agreement protocol*.

- (i) exchange information about the observations among all nodes, such that all nodes have the same data set.
- (ii) every node executes the same algorithm on this data set to arrive at a consistent value and at a sparse interval of the observation.

An Impossibility Results

It is impossible to represent the temporal properties of the dynamic *analog* physical world with *true fidelity* in *digital* cyberspace.

- The conflict between fidelity and consistency can be reduced, but can never be fully resolved.
- The better the precision of the clock synchronization, the smaller the error introduced by digitalization and synchronization.

Fault and Error Containment

Error Propagation in Case of *Non-Fail-Silence*



A *non-fail-silent* faulty component can corrupt another good component by sending an erroneous message:

- Message error in the value domain
- Message error in the temporal domain

If we do not detect and intercept the faulty messages, then a *single fault* that is not properly contained can lead to *multiple component failures*, thus compromising the *independence requirement*.

Error Containment Region (ECR)

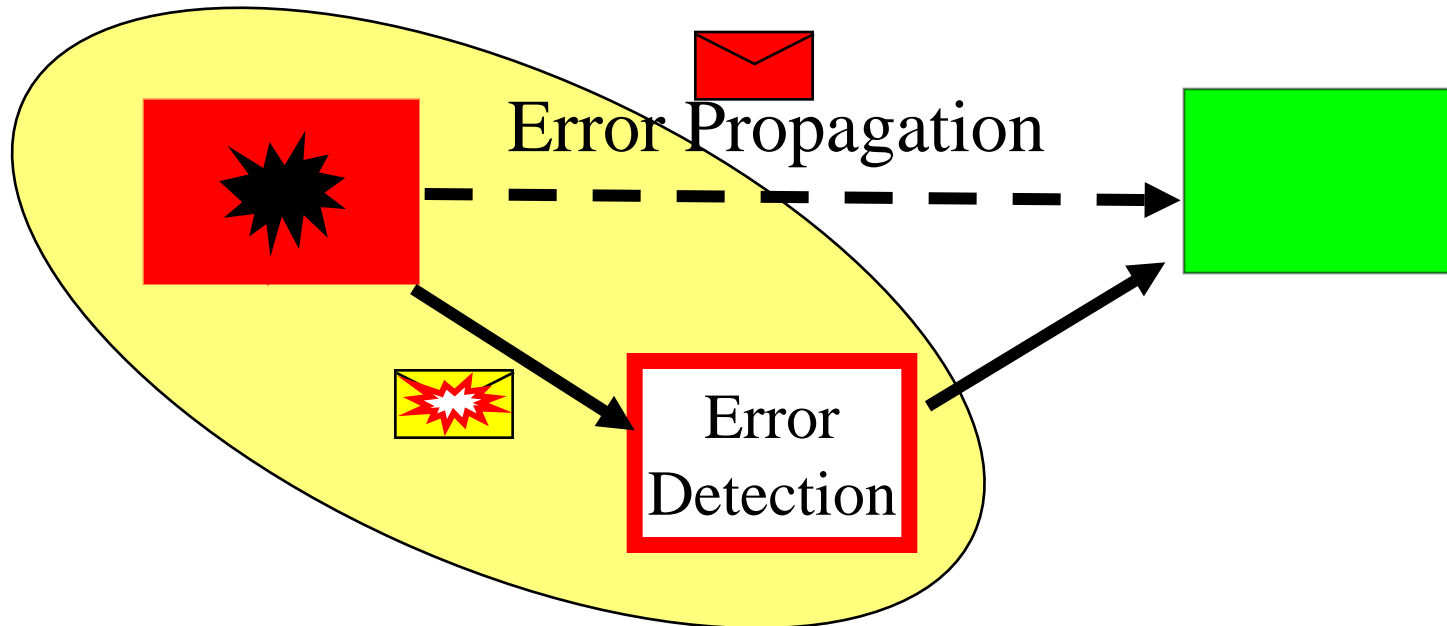
In a distributed computer system the consequences of a fault, the ensuing error, can propagate outside the originating FCU (Fault Containment Unit) by an **erroneous message** of the faulty node to the environment.

- ◆ A propagated error **invalidates** the independence assumption.
- ◆ The error detector must be in a **different FCU** than the faulty unit.
- ◆ Distinguish between architecture-based and application-based error detection
- ◆ Distinguish between error detection in the **time-domain** and error detection in the **value domain**.

Since an Error Containment Region requires at least two independent FCUs, a single die cannot form an Error Containment Region.

Fault Containment vs. Error Containment

We do not need an error detector in the value domain if we assume fail-silence.



Error Detector must be in a separate FCU

Consequences for the Architecture

In a safety-critical application an SoC (System on Chip) must be considered to form a *single unit of failure*, ie., a *single FCR* that can fail in an *arbitrary* failure mode because of:

- ◆ Same Physical Space (Physical Proximity Failures)
- ◆ Same Wafer Production Process and Mask (Mask Alignment Issues)
- ◆ Same Bulk Material
- ◆ Same Power Supply and Same Earthing
- ◆ Same Timing Source
- ◆

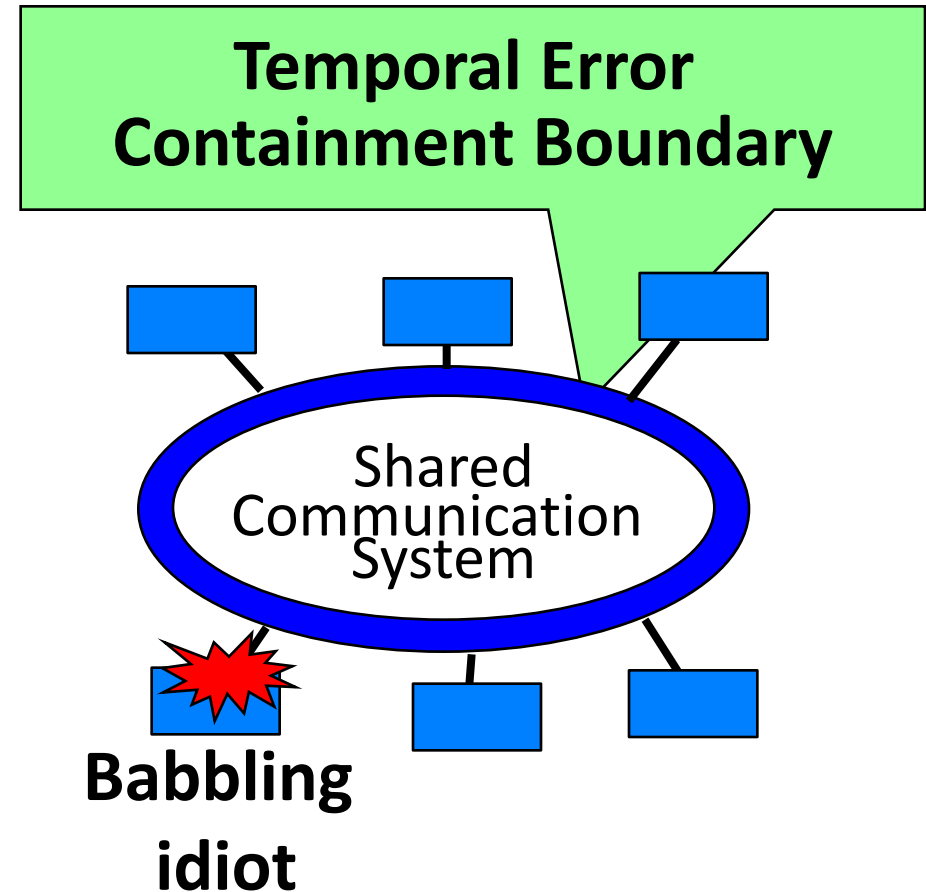
Although some of these dependencies can be eliminated, others cannot.

We cannot assume an *independent* error detector on the same die.

Containment of Temporal Error by the CS

It is *impossible* to maintain the communication among the correct components of a RT-cluster **if the temporal errors caused by a faulty component are not contained.**

Error containment of an arbitrary temporal node failure requires that the shared Communication System is a self-contained FCU that has information about the allowed temporal behavior of the nodes—application specific.



High priority message
in a CAN System?

Fault Tolerance

Fault Containment Unit (FCU)

A *Fault-Containment Unit (FCU)* is an independent subsystem with well-defined interfaces such that the immediate consequences of a single fault (e.g., hardware, software) are contained within this subsystem.

Examples:

A fail-silent component

A micro-processor including the software with a well-defined message interface.

Fault-Tolerant Unit (FTU)

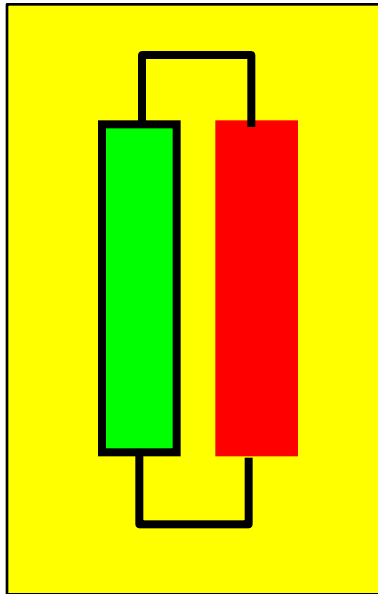
A fault-tolerant unit (FTU) is a set of actively redundant FCUs (components) that provide a fault tolerant service to its environment:

- ◆ FTUs have to receive identical input messages in the same order
- ◆ FTUs have to operate in replica determinism
- ◆ The output messages of FTUs should be idempotent
- ◆ As long as a defined subset of the components of the FTU is operational, the FTU is considered operational

FTUs provide the continuous service by fault masking.

Fault Masking: How Many FCUs in an FTU?

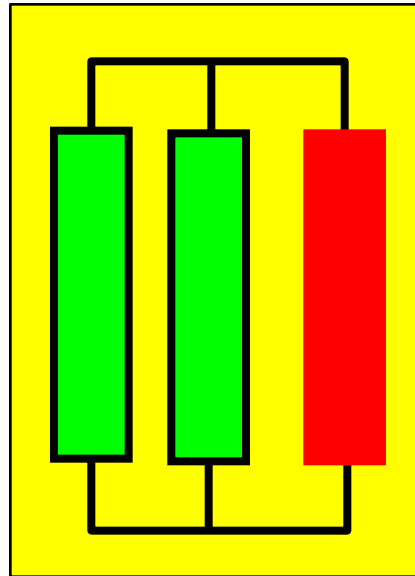
A



assumption
fail-silent FCUs

$$k+1$$

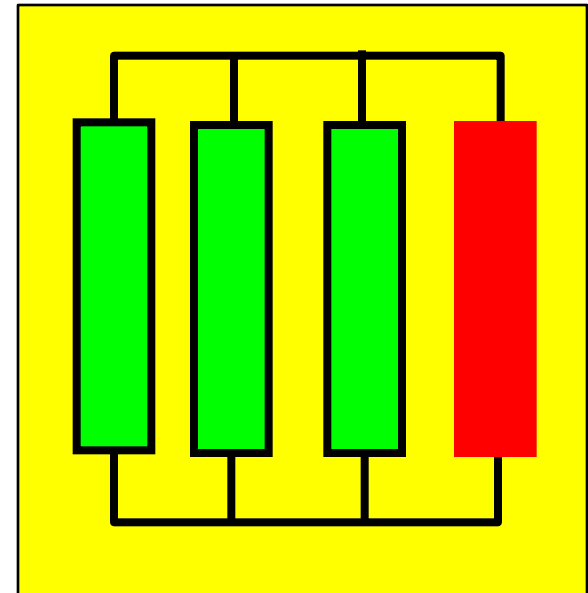
B



assumption
Synchronized FCUs

$$2k + 1$$

C



no assumption about FCU
(arbitrary)

$$3k + 1$$

What is the *assumption coverage* in cases A and B?

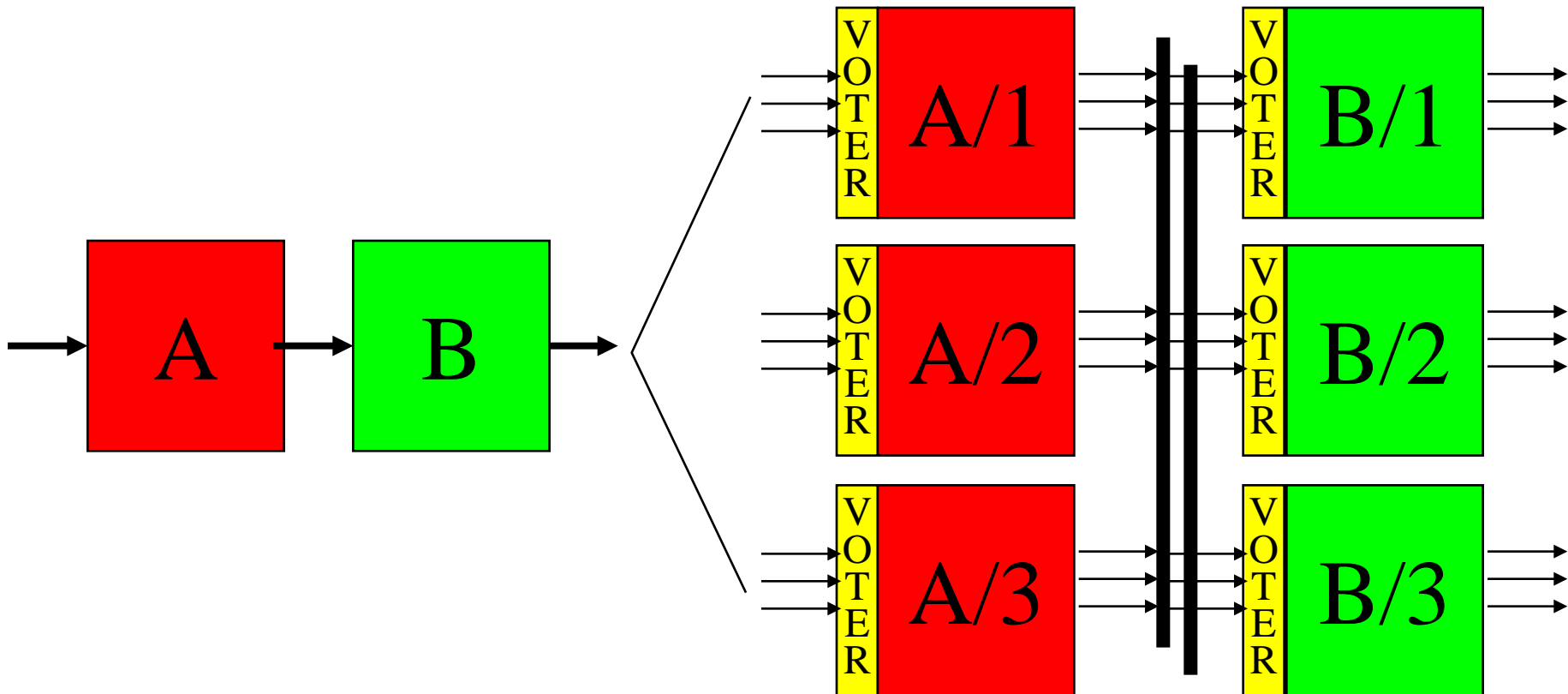
Another *Impossibility* Result

It is impossible to synchronize three clocks if one of the clocks can exhibit an arbitrary failure mode.

Pease, M., R. Shostak, & L. Lamport, Reaching Agreement in the Presence of Faults. *Journal of the ACM*, 1980. 27(2): pp. 228-234.

Triple Modular Redundancy (TMR)

Triple Modular Redundancy (TMR) is the generally accepted technique for the mitigation of component failures at the system level:



What is Needed to Implement TMR?

What architectural services are needed to implement Triple Modular Redundancy (TMR) at the architecture level?

- ◆ *Replica Deterministic (which includes timely)* Operation of the three Lanes
- ◆ Provision of an Independent Fault-Containment Region for each one of the Replicas
- ◆ Replicated *Independent* Communication Channels
- ◆ Synchronization Infrastructure (requires *four* clocks)
- ◆ Predictable Multicast Communication
- ◆ Support for Voting

Composability

System Integration: Composability

Construction:

1. *Precise Interface Specifications* of the component behavior (the input and output messages) in the domains of value and time
2. *Stability of Prior Properties* of the components
3. *Non-interfering interactions* among components in the domains of value and time

Dependability:

4. *Error Containment* of faulty components
5. *Fault-Masking* of erroneous components

**If we agree with these principles,
then they become *design challenges*.**

Legacy Integration

A candidate for a winning protocol is required to be compatible with main-line existing legacy applications in hardware and software. Ethernet is by far the most important important existing protocol for local area communication.

- ◆ Message format in full conformance with Ethernet standard
- ◆ Standard Ethernet traffic must be supported in all configurations
- ◆ Real-time Ethernet functionality can be implemented on COTS Ethernet controllers in software (limited temporal precision)
- ◆ Existing Ethernet controller hardware can coexist with special RT Ethernet controllers.
- ◆ Overlay networks can be implemented on top of RT Ethernet to establish the legacy interface at the API (e.g., CAN).

The Outlook

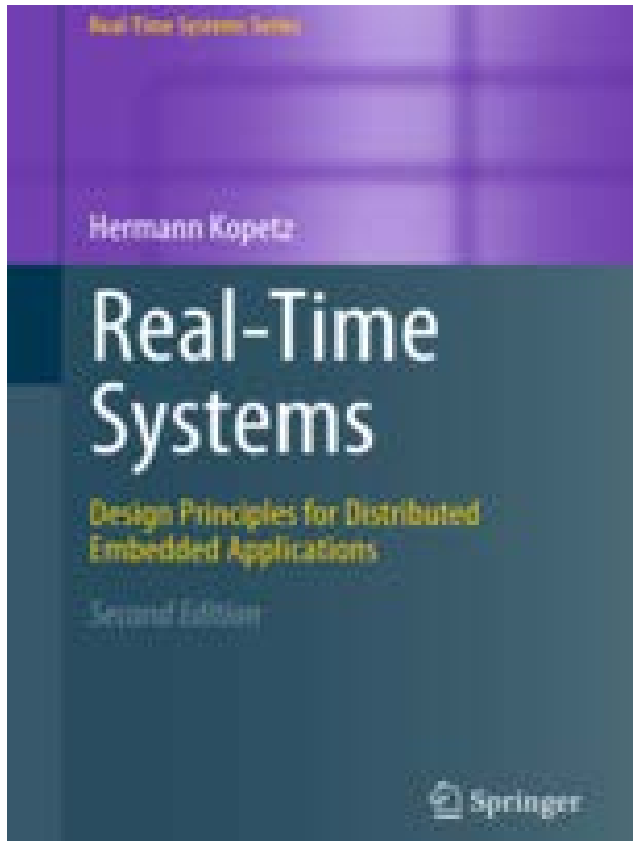
- ◆ Within the next decade, many of today's real-time protocols will disappear.
- ◆ Real-time traffic must originate from a closed cooperating environment, while non-real-time traffic originates from an open competing environment.
- ◆ A winning protocol must be based on a sound theoretical footing and must support legacy installations with event-triggered and time-triggered traffic
- ◆ Speed by itself will not solve all problems.
- ◆ Global time must be supported.

.

Summary: Characteristics of a Winner

- ◆ **Sound Theoretical Basis:** Conceptual clarity that supports all types of real-time systems--from simple data collection to high-dependability systems that require TMR
- ◆ **Support of Global Time:** Many real-time applications require a global time
- ◆ **Support of Fault-Tolerance:** Determinism and Error Containment.
- ◆ **Ethernet based:** Ethernet is a quasi world-wide standard in non-real-time applications
- ◆ **Controller available on a System on Chip:** Standard Ethernet controllers are preferred candidates. FPGAs opens interesting alternatives.
- ◆ **Controlled Openness:** eg., CAN, Ethernet

More Information



Background information can be found in the second revised edition of my book

Real-Time Systems—Design Principles for Distributed Embedded Applications

published by ***Springer Verlag*** on April 27 , 2011.