Ensuring Reliable Networks TTTech

TTA and PALS: Formally Verified Design Patterns for Distributed Cyber-Physical

DASC 2011, Oct/19

CoMMiCS

Wilfried Steiner
wilfried.steiner@tttech.com
TTTech Computertechnik AG

John Rushby
rushby@csl.sri.com
SRI International

Ensuring Reliable Networks TTTech

Introduction

- Cyber-Physical Systems
- Models of Computation
  - Synchronous MoC
  - Time-Aware MoC
  - Time-Synchronized MoC

The Physically-Asynchronous Logically-Synchronous Protocol (PALS)

- PALS Notation and Principles of Operation
- Formal Verification and discussion of imprecise definitions

The Time-Triggered Architecture (TTA)

- Layers of Time
- Fundamental Limits in Time Measurement
- Sparse Time

Conclusion

Introduction

Cyber-Physical Systems

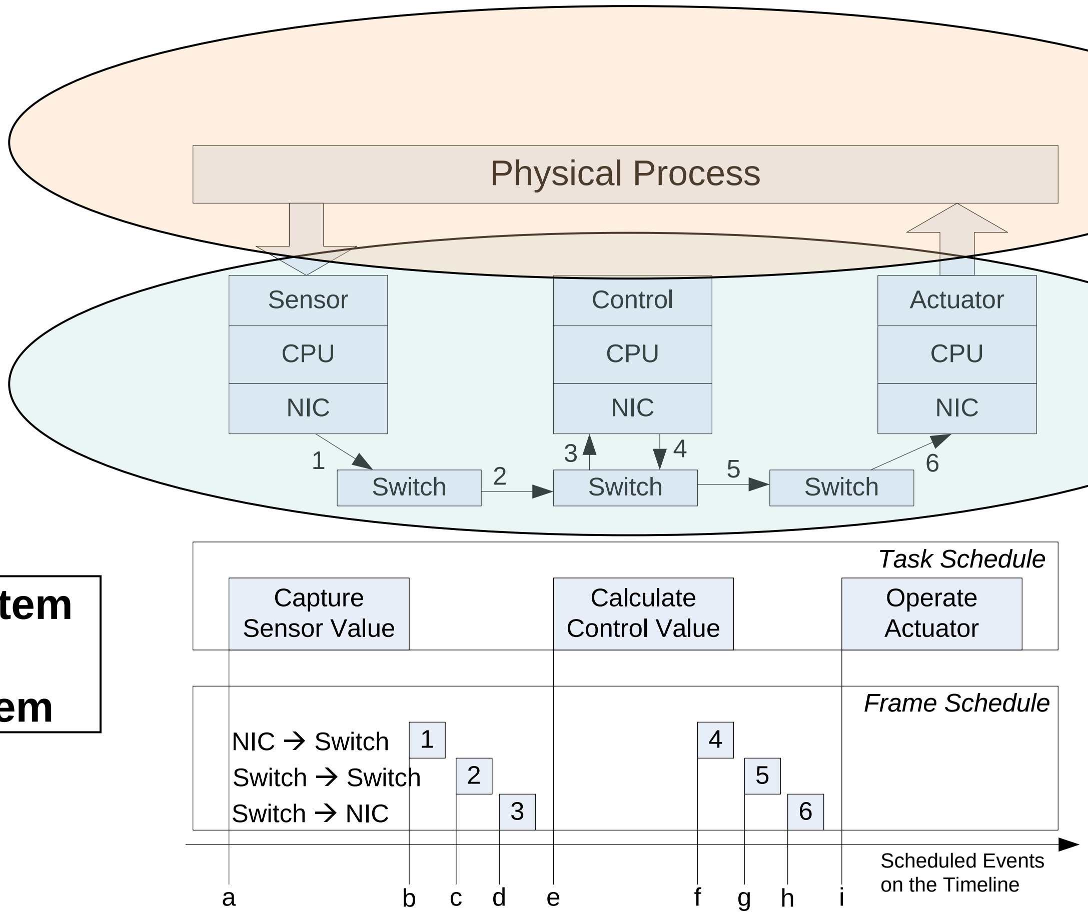
Distributed Cyber-Physical Systems

Ensuring Reliable Networks TTTech

Physical Subsystem

Cyber Subsystem

Time-Triggered System Implementing the Cyber Subsystem



Models of Computation (MoC)

A Model of Computation (MoC) is an abstract interface that hides implementation details from the *application* developer.

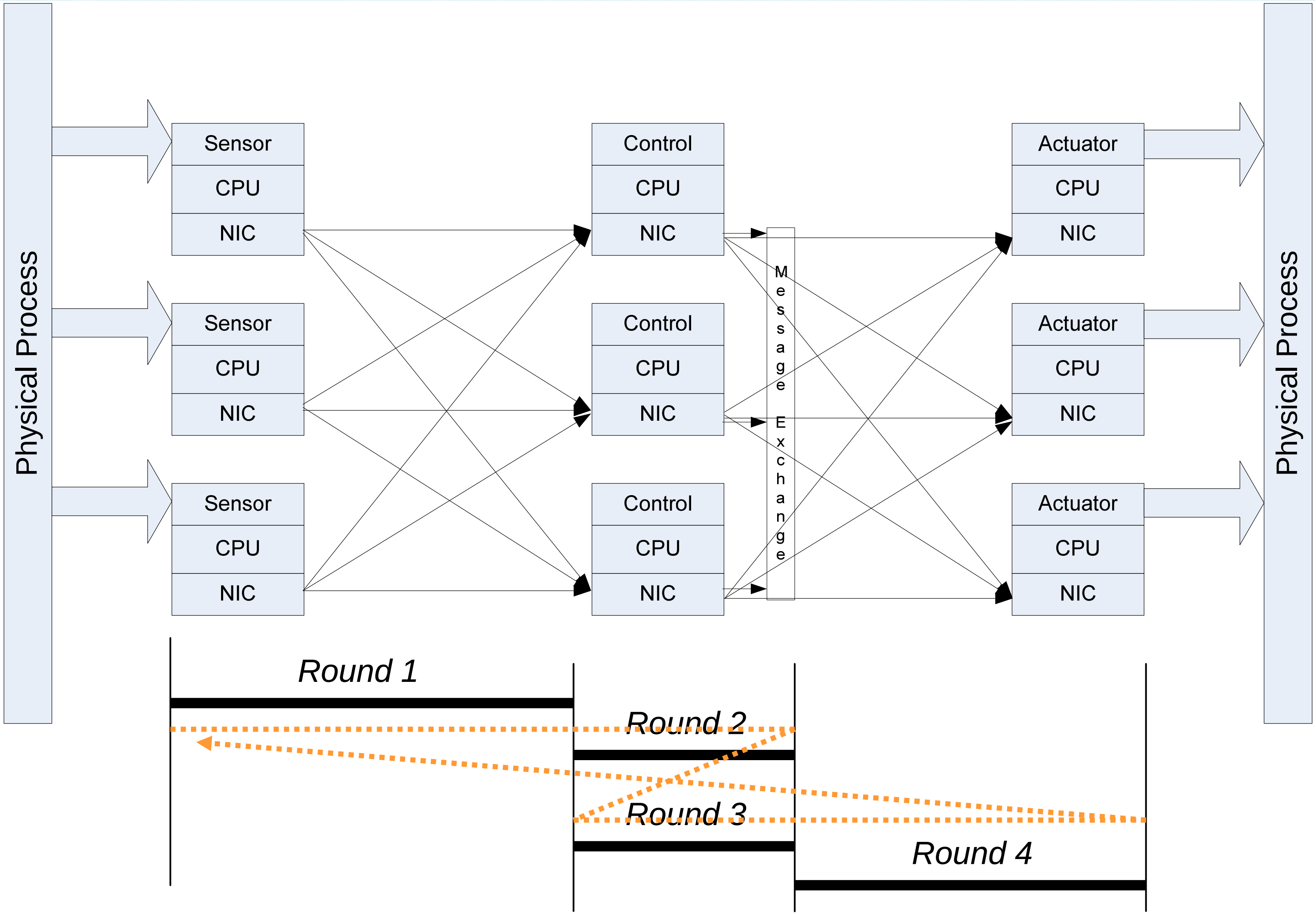
- E.g., a protocol on top of AFDX; TTEthernet

Synchronous MoC

- Multiple (probably all) nodes in the system progress synchronously in “rounds.”
- A round consists of a communication phase followed by a computation phase.
- The computation phase is only entered when all nodes received all (non-faulty) messages during the preceding communication phase.
- Synchronous MoC aims to systematically avoid race conditions, e.g., scenarios in which messages are not received consistently.

Fault-Tolerant Cyber Subsystem

Ensuring Reliable Networks TTTech



Synchronous Model of Computation (MoC)

Models of Computation (MoC)

Ensuring Reliable Networks TTTech

A Model of Computation (MoC) is an abstract interface that hides implementation details from the *application* developer.

- E.g., a protocol on top of AFDX; TTEthernet

Synchronous MoC

- Multiple (probably all) components in the system progress synchronously in “rounds.”
- A round consists of a communication phase followed by a computation phase.
- The computation phase is only entered when all nodes received all (non-faulty) messages during the preceding communication phase.
- Synchronous MoC aims to systematically avoid race conditions, e.g., scenarios in which messages are not received consistently.

Typically, the <u>Synchronous MoC</u> does not exist per se, but <u>has to be established by protocols</u> (like PALS) executed in the network.

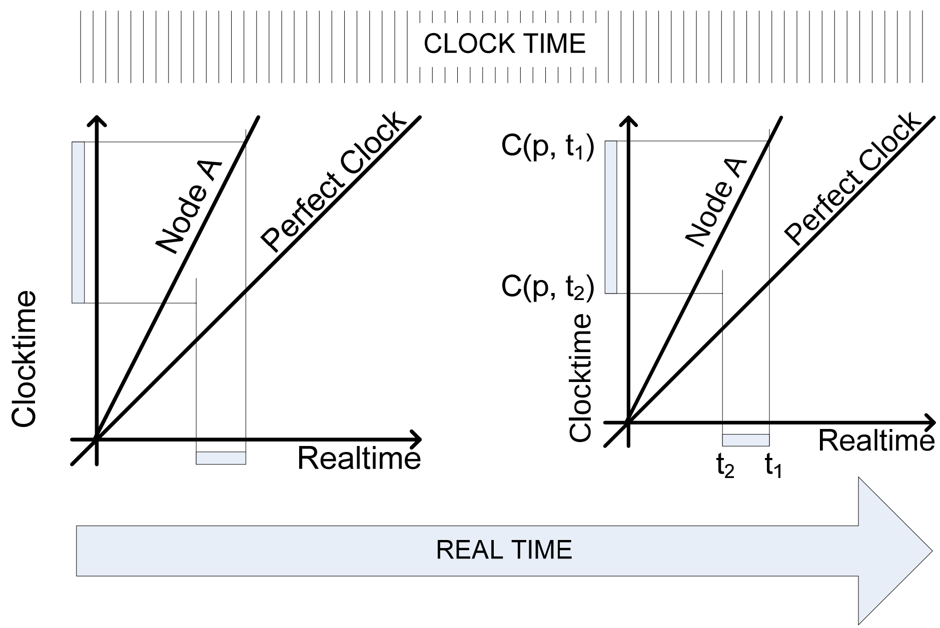
On the other hand, the Synchronous MoC is also not directly usable for Cyber-Physical Systems, because it does not provide a formal relation between a “round” and real-time.

The <u>Synchronous MoC</u> has, thus, to be <u>extended by real-time clocks</u>.

Time-Aware MoC

- All components are equipped with real-time clocks that can measure the progress in real-time.
- These real-time clocks are assumed to be not perfect, but to have a non-zero “drift rate” ρ.
  - I.e., an interval in real-time of duration *d* will appear inside a node as

$$(1-\rho)*d <= d <= (1+\rho)*d$$



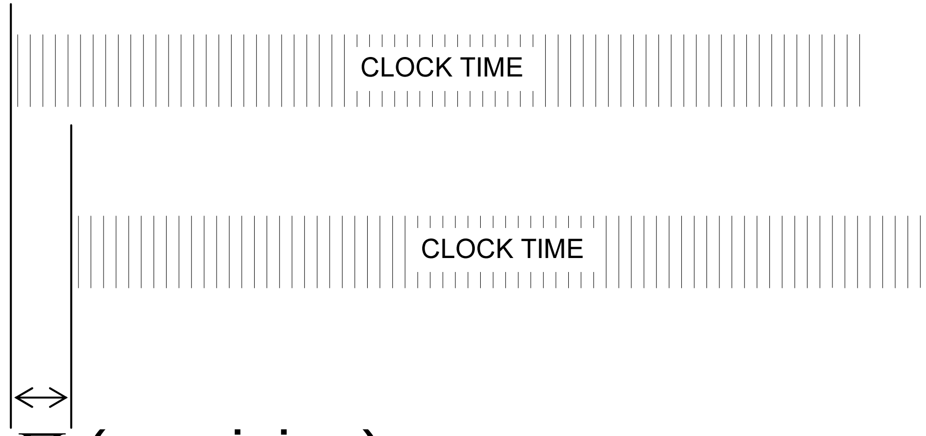
Clock Time is a simulation of Real Time inside a computer.

Real Time is Newtonian Time, a continuous entity.

Ensuring Reliable Networks TTTech

Time-Synchronized MoC

- Any two real-time clocks are synchronized with each other with an *a priori* known maximum distance (called the precision).



Clock Time is a simulation of Real Time inside a computer.

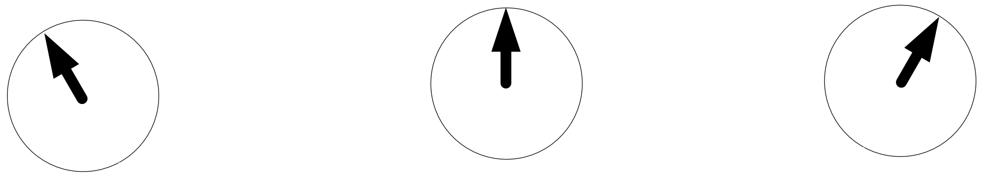
Clock Time is a simulation of Real Time inside a computer.

Π (precision)

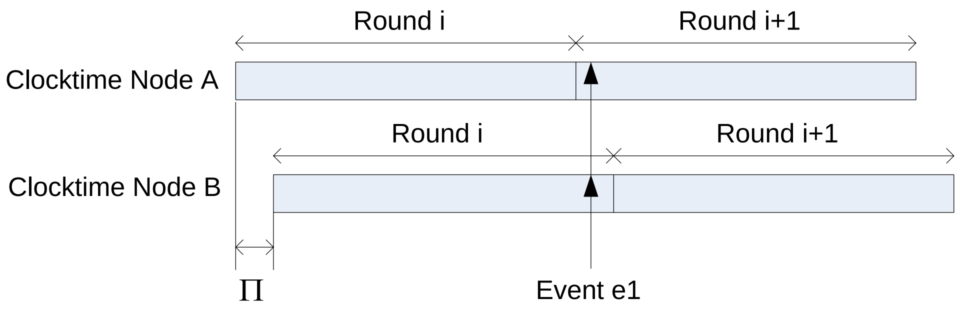
Definition: Precision

Ensuring Reliable Networks TTTech

In an ensemble of clocks, the precision Π is defined as the maximum distance between any two synchronized non-faulty clocks at any point in real time.



Late Clock **Perfect Clock** **Early Clock**



TTA satisfies the Time-Synchronized MoC

- The main problem is to show that a network with synchronized clocks can "simulate" the Synchronous MoC (all nodes consistently progress through communication and computation phases)
- Formally verified by Rushby and Pike by restricting when messages may be communicated
  - *Rushby, J., 1999, "Systematic formal verification for fault tolerant time-triggered algorithms. IEEE Transactions on Software Engineering, 25(5):651–660."*
  - *Pike, L., 2006, "A note on inconsistent axioms in Rushby's "Systematic formal verification for fault-tolerant time-triggered algorithms". IEEE Transactions on Software Engineering, 32(5)*

PALS defines a similar pattern to TTA and we reuse Rushby's formal model to prove that also PALS satisfies the Time-Synchronized MoC

- A Time Layer called the PALS clocks (or PALS Time)
- Rules that specify when messages may be sent and when not

We then show how the formal model can also be reused to verify further properties of the TTA

- We formally prove the four fundamental limits of time measurement (FLTM)
- We formally prove properties of the Sparse Timebase:
  - A paradigm that specifies intervals when events may be generated.

The Physically-Asynchronous Logically-Synchronous Protocol (PALS)

PALS Definitions i

G2: Real Time Network. The network has a network queueing (scheduling) delay *q* bound by

$$0 < q_{min} <= q <= q_{max}$$

and a network transmission delay μ bounded by

$$0 < \mu_{min} <= \mu <= \mu_{max}$$

G3: Real Time Machine. . . . The task completion time α, including real time scheduling, computation, and I/O is bounded by $0 < \alpha_{min} <= \alpha <= \alpha_{max}$ . . .

PALS Clocks. All the local clocks used by PALS for global computation are synchronized with the global clock with skews of at most ε (note: $2 * \varepsilon = \Pi$).

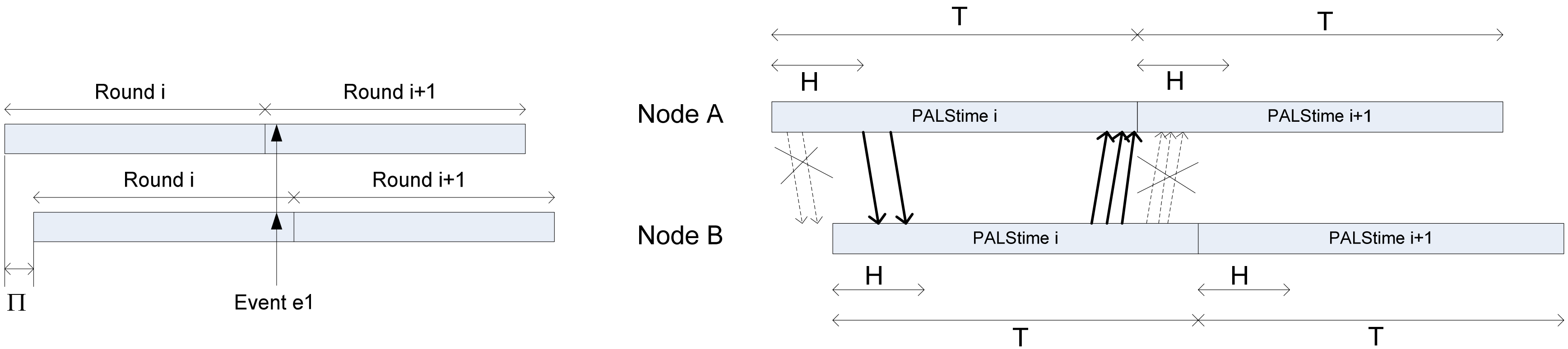
PALS Definitions ii

G6: PALS Causality Rule. A machine at (PALS) clock period *j* cannot send earlier than

$$(C_i = j) + H, \text{ where } H = 2^*\varepsilon - \mu_{min}.$$

G7: PALS Clock Period. PALS clock period

$$T > 2^*\varepsilon + \max(\alpha_{max} + q_{max}, H) + \mu_{max}$$



To be shown → Fact 3. A message sent during sender's j^{th} clock period will be received by all machines when they are still in their j^{th} clock period.

Formal Proof found an imprecision in this argument.

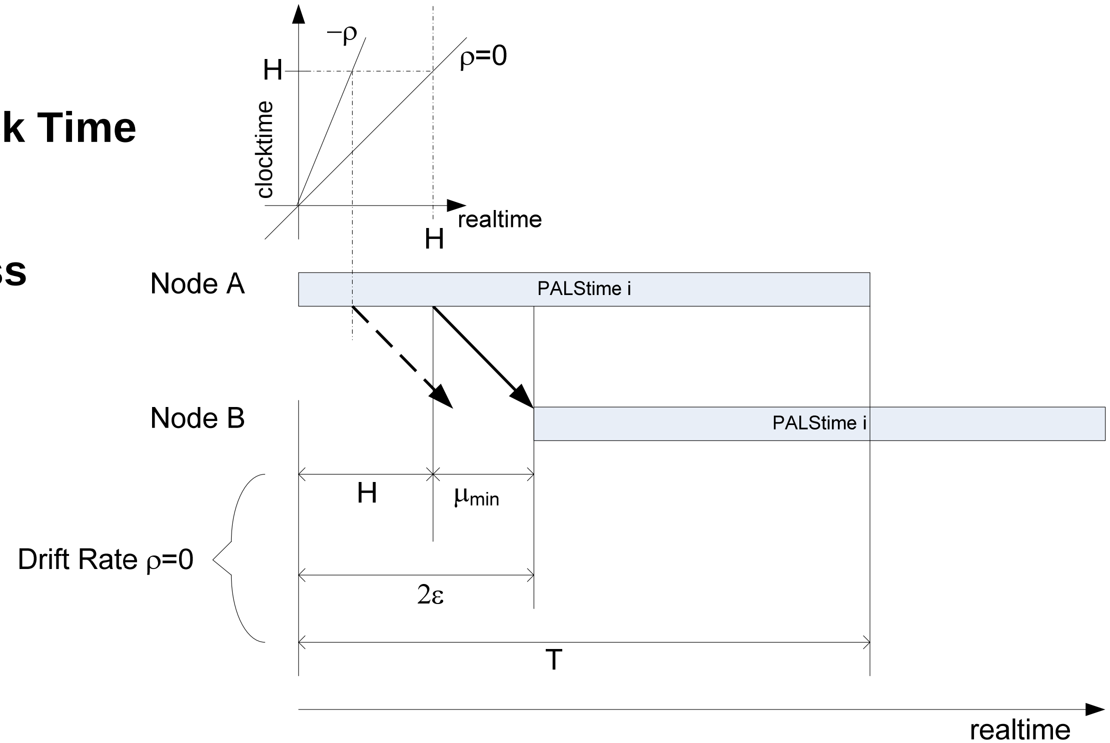
Formal Verification of Fact 3

Imprecision 1: H cannot be negative, thus $H = 2*\varepsilon - \mu_{min,}$ needs to be modified to $H = \max(0, 2*\varepsilon - \mu_{min})$

- **Has been corrected in publications following the original PALS description**

Imprecision 2:

- **H and ε are of type Clock Time**
- **μ_{min} is of type Real Time**
- **Clock Time may progress faster than Real Time**



We modify $H = 2*\varepsilon - \mu_{min,}$ to

$$H = 2*\varepsilon - \text{floor}(\mu_{min} * (1\text{-rho}))$$

✓ **Formally Verified**

Ensuring Reliable Networks TTTech

The Time-Triggered Architecture (TTA)

In the Synchronous MoC events (messages, computations) can be assigned to particular rounds.

However, the duration in real-time of a round may become quite high, e.g., tens of milliseconds.

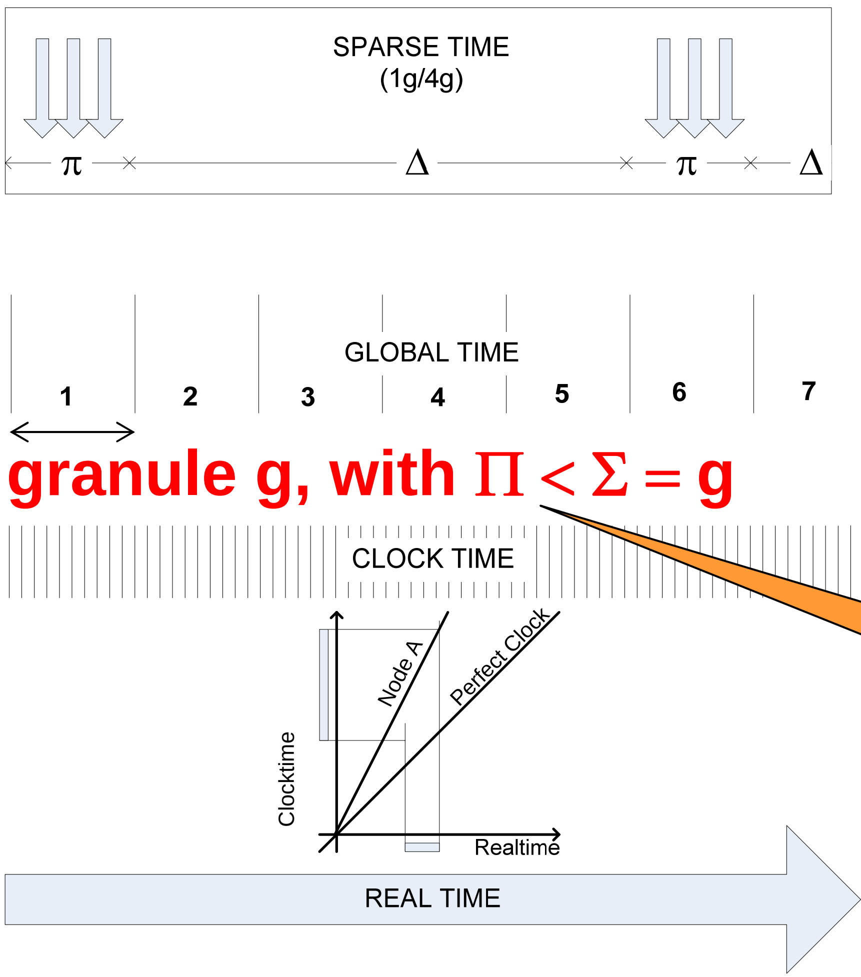
We can leverage the synchronized global time also for a generic time-stamping service.

- E.g., different nodes will assign “similar” time-stamps to an external event
- E.g., time-stamps can help to determine the temporal and causal order of events

We can go even further and agree that our system generates events only at certain times, such that the order of their occurrence can be easily determined by an observing system.

Layers of Time in the TTA

Ensuring Reliable Networks TTTech



Sparse Time is a design guideline according which a computer generates events only during pre-defined intervals.

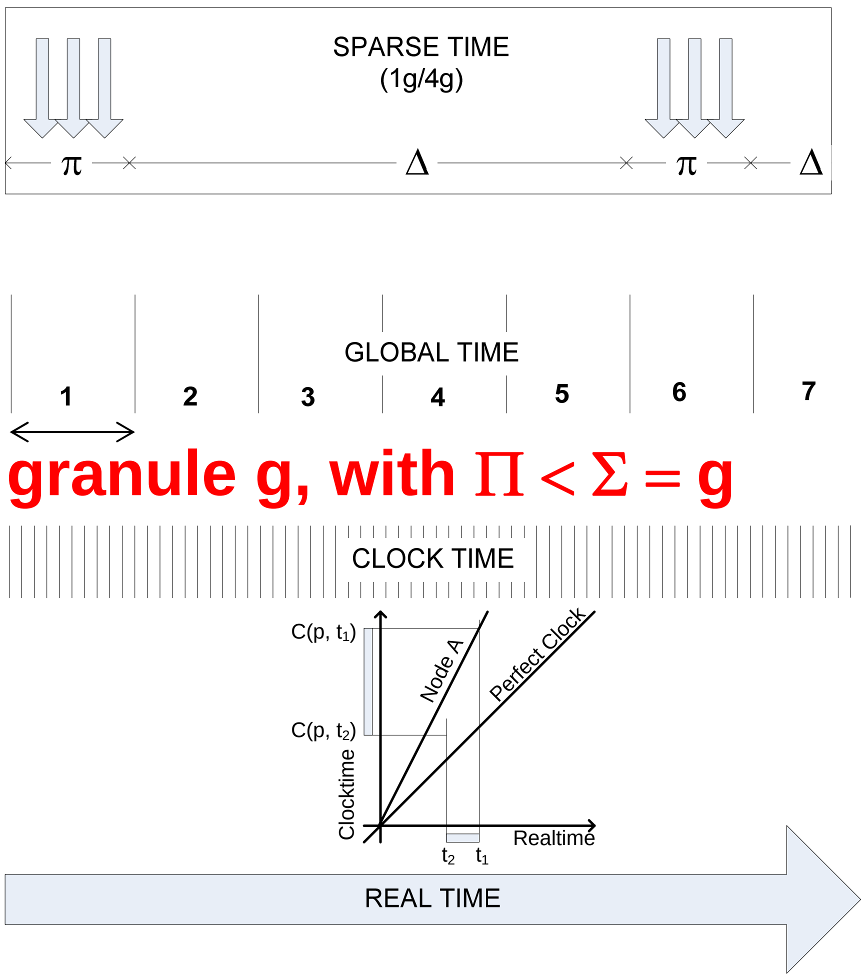
Global Time groups a configurable number of ticks in Clock Time into a coarser tick granularity.

Clock Time is a simulation of Real Time inside a computer.

"Reasonableness Condition"

Real Time is Newtonian Time, a continuous entity.

Formalization of Time




```
sparsetime_b: THEOREM
z(e2) >= z(e1) AND
floor((z(e2)-z(e1))*(1-rho)) >= 4*granularity
  => G(p, C(p, z(e2))) - G(q, C(q, z(e1))) >= 3
```

```
GLOBALtime: TYPE = nat
granularity: clocktime = Sigma
G(p, c): GLOBALtime = floor(c/granularity)
```

```
Sigma: clocktime
clock_sync: AXIOM abs(C(p, t) - C(q, t)) < Sigma
```

```
clocktime: TYPE = nat
```

```
rho: {x: real | 0 < x AND x < 1}
drift_rate: AXIOM t1 >= t2 IMPLIES
floor((1-rho)*(t1-t2)) <= C(p,t1)-C(p,t2) AND
C(p,t1)-C(p,t2) <= ceiling((1+rho)*(t1-t2))
```

```
t: VAR realtime
C(p, t): clocktime
```

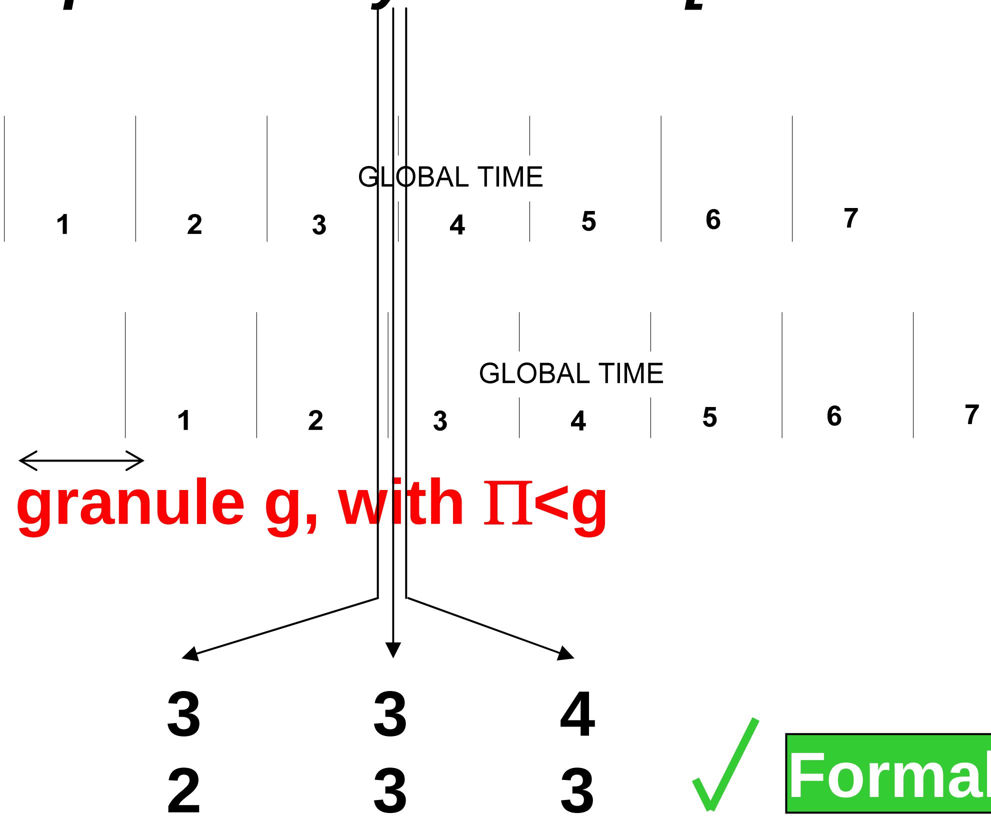
```
realtime: TYPE = nonneg_real
```

Ensuring Reliable Networks TTTech

FLTM i: ***If a single event is observed by two different nodes, there is always the possibility that the timestamps differ by one tick [in Global Time].***

Fast Node:

Slow Node:

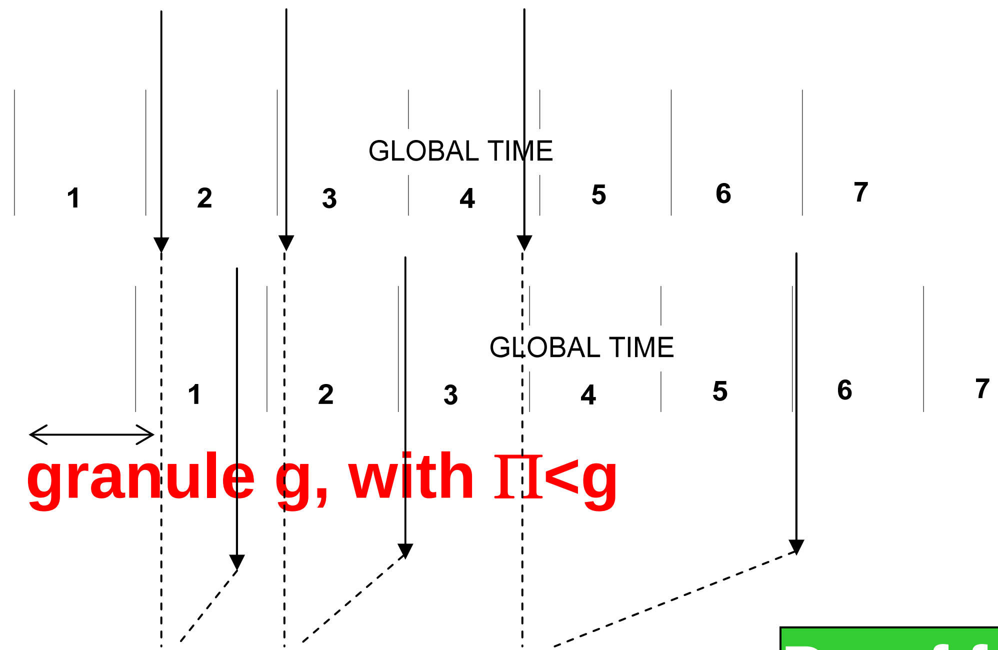


Formally Verified

FLTM iii: ***The temporal order of events can be recovered from their timestamps, if the difference between their timestamps is equal to or greater than 2 ticks.***

Fast Node:

Slow Node:



Observing Node:

	2	**3**	**4**
	1	**3**	**6**
Can temporal order be established?	no	no	yes

✓

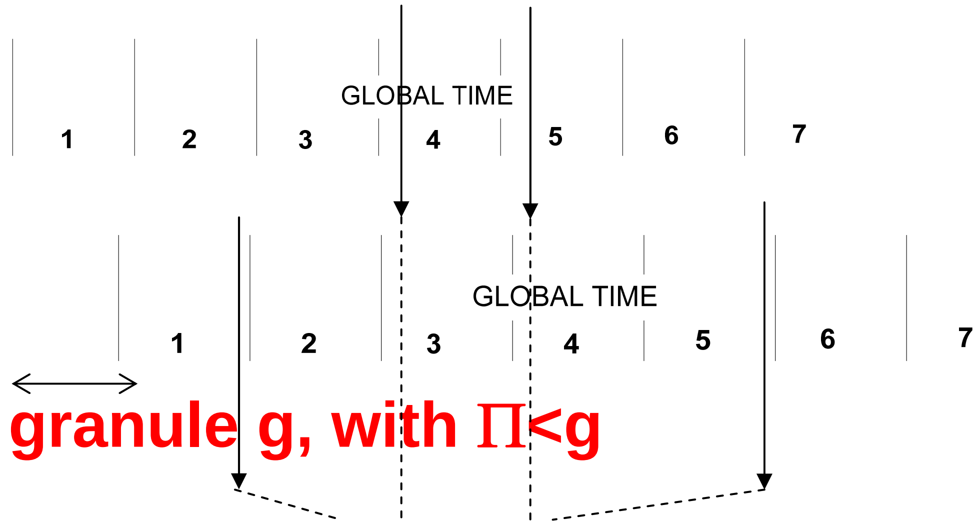
Proof follows from FLTM i

Formally Verified

FLTM ii: If the observed duration of an interval is d_{obs}, (in Global Time) then the true duration d_{true} (in Real Time) is bounded by: $(d_{obs} - 2g) < d_{true} < (d_{obs} + 2g)$

Fast Node:

Slow Node:



granule g, with Π<g

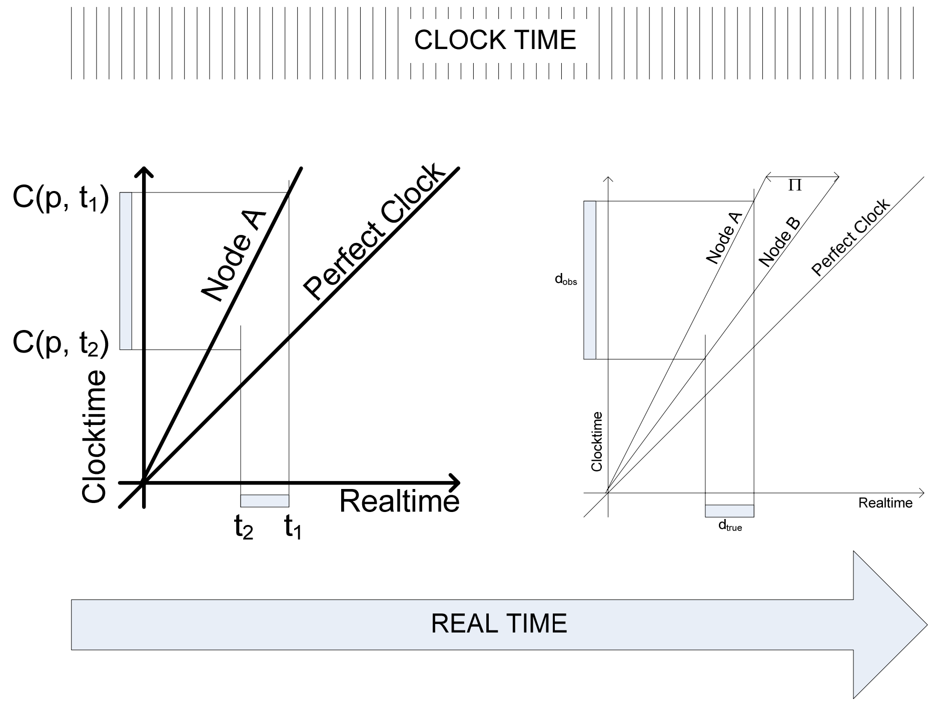
Observing Node:

$$\begin{array}{r} -1 \\ 4 \\ \hline 3 \end{array} \qquad \begin{array}{r} -5 \\ 5 \\ \hline 0 \end{array}$$

Formal Proof found an imprecision in this argument.

Corrected Version – next Slide

Fundamental Limits of Time Measurement ii




```
Sigma: clocktime
clock_sync: AXIOM abs(C(p, t) - C(q, t)) < Sigma
```

```
clocktime: TYPE = nat
```

```
rho: {x: real | 0 < x AND x < 1}
drift_rate: AXIOM t1 >= t2 IMPLIES
floor((1-rho)*(t1-t2)) <= C(p,t1)-C(p,t2) AND
C(p,t1)-C(p,t2) <= ceiling((1+rho)*(t1-t2))
```

```
t: VAR realtime
C(p, t): clocktime
```

```
realtime: TYPE = nonneg_real
```

~~**FLTM ii: … $(d_{obs} - 2g) < d_{true} < (d_{obs} + 2g)$ ….**~~

FLTM ii: … $(d_{obs} - 2g) < (1+rho) * d_{true}$ and
$(d_{obs} + 2g) > (1-rho) * d_{true}$ …

Formally Verified

So far the FLTMs were of the logic: given that events happen, what can we conclude from their timestamps?

FLTM iv is of opposite nature: how shall we generate events such that their timestamps are in a certain relation?

FLTM iv: The temporal order of events can always be recovered from their timestamps, if the event set is *0/3g*-precedent.

Def.: 0/3g-precedent

- Events occur either at the same instant in Real Time or at least 3g apart.

Fundamental Limits of Time Measurement iv

Ensuring Reliable Networks

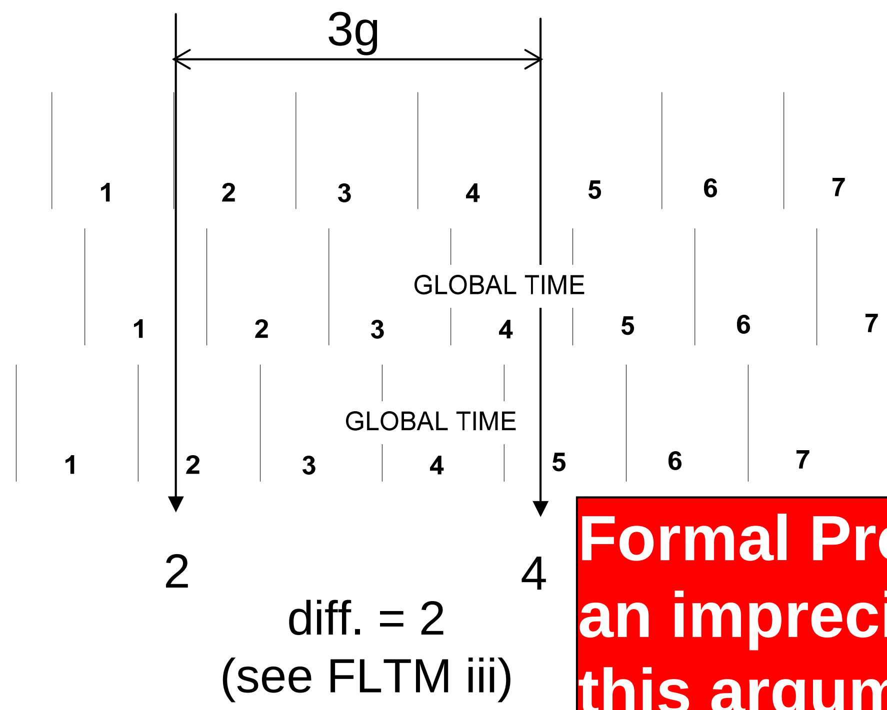
FLTM iv: The temporal order of events can always be recovered from their timestamps, if the event set is *0,3g*-precedent.

Source of Events:
(e.g., unsynchronized clock)

Event Time-stamping Node 1:

Event Time-stamping Node 2:

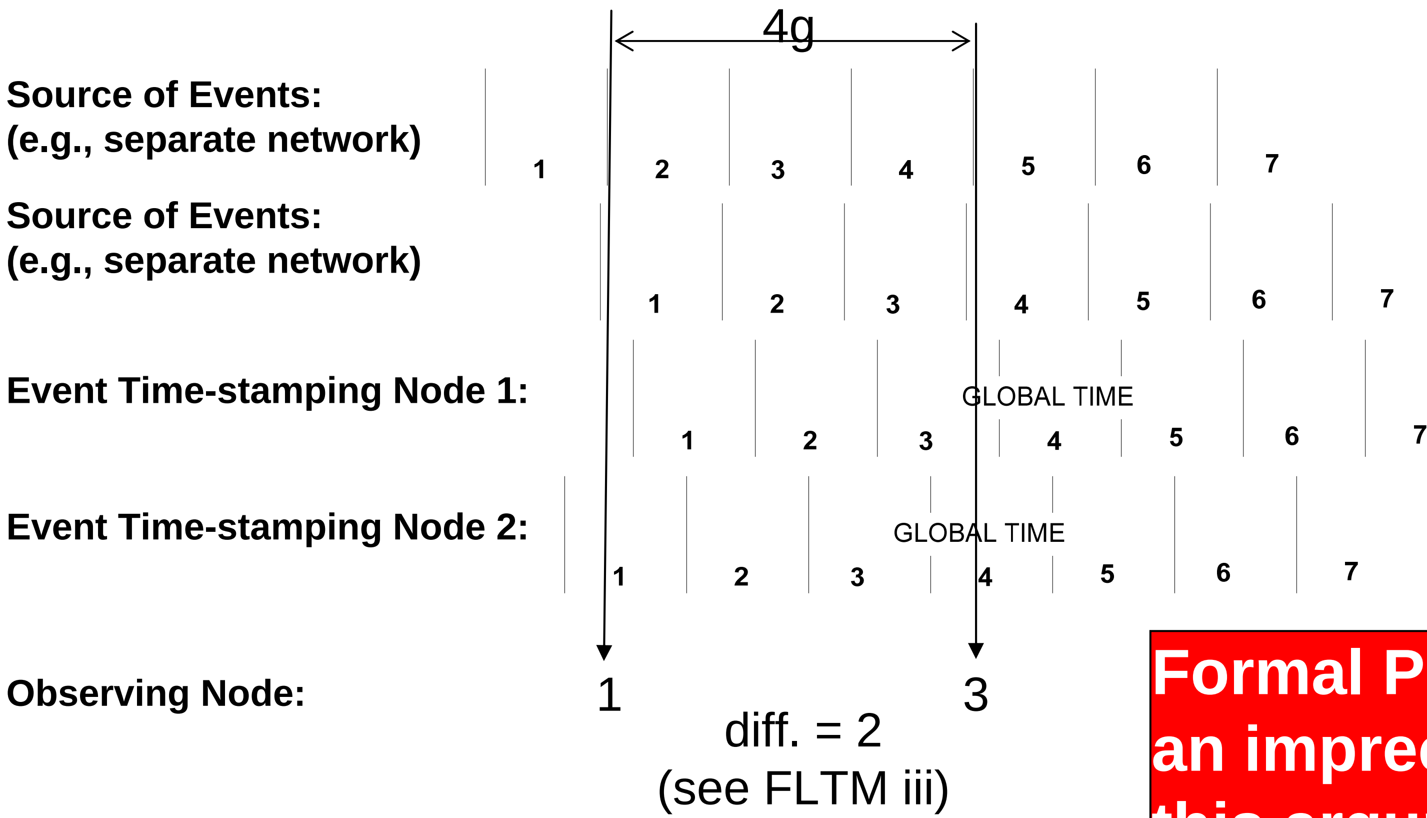
Observing Node:



Formal Proof found an imprecision in this argument.

Corrected Version → ***0 , 3g*1/(1*-rho) precedent** ✓ **Formally Verified**

The temporal order of events can always be recovered from their timestamps, if the event set is *1g,3g*-precedent.



Formal Proof found an imprecision in this argument.

Corrected Version → ***1g/(1+rho),4g/(1**-rho) precedent** ✓ **Formally Verified**

Original Reasonableness Condition:

- granule g, with $\Pi < \Sigma = g$

Generalized Reasonableness Condition:

- g = round-up [(Σ + 1) / (1- number_granules *rho)]
  - rho … drift rate
  - number_granules … {3, 4, …}

Generalized form allows again to use:

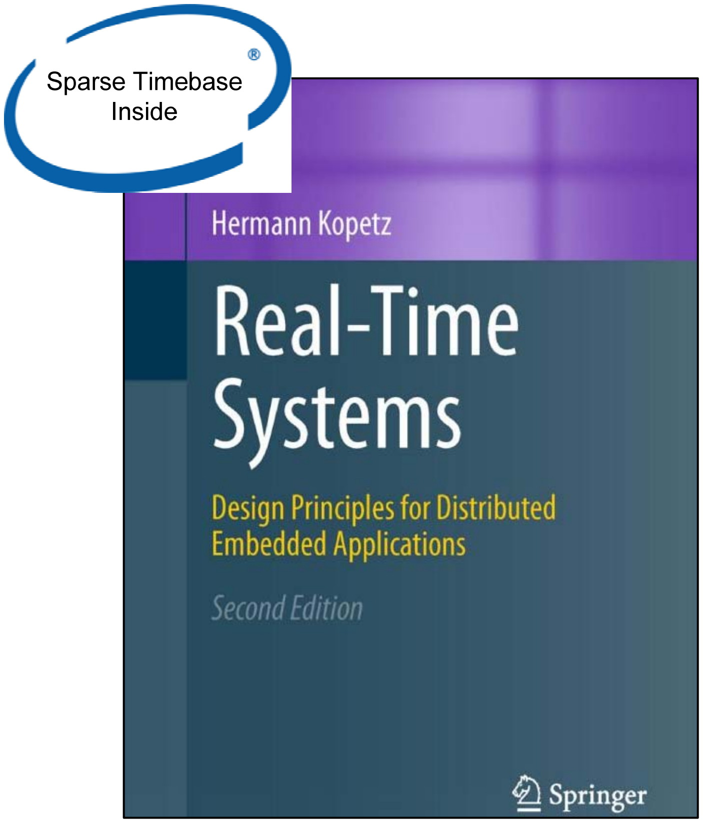
- 0,3g – precedence in FLTM iv
- 1g,4g – precedence for the Sparse Timebase

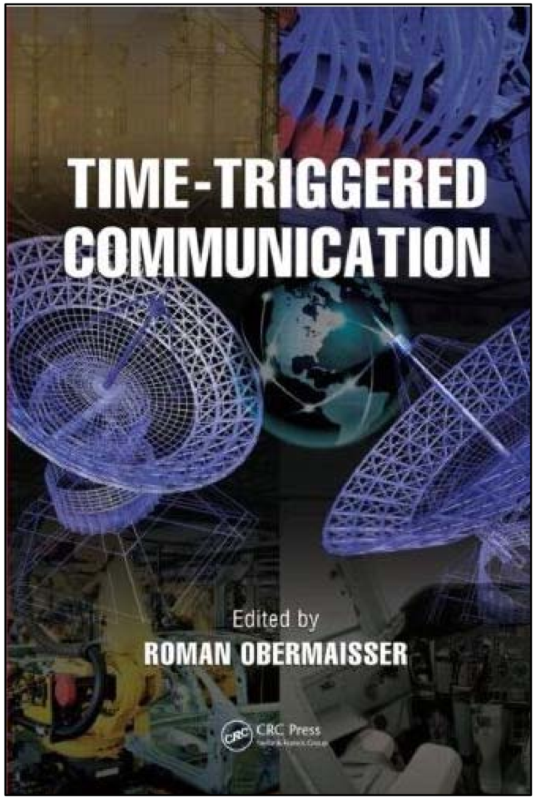
Conclusion and Summary

Summary and Conclusion

Ensuring Reliable Networks

- **We have formally verified (modified versions) of PALS and TTA and found imprecise definitions.**
- **Subtle dependencies between Real Time and Clock Time can be overlooked by pencil proofs and even by formal verification on a too abstract model.**
  - A similar imprecision has been found in PALS and the TTA.
  - Clock Time is seen as an abstraction layer that hides all details from Real Time.
  - Yet, as we have demonstrated Clock Time is not a perfect abstraction when reasoning about synchronized actions in a distributed system.
- **Type system of PVS detects mismatches between Clock Time and Real Time early on.**
- **We have reused PVS model from over 15 years ago as a basis for our formal proofs.**


Sparse Timebase Inside
Hermann Kopetz
Real-Time Systems
Design Principles for Distributed Embedded Applications
Second Edition
Springer


TIME-TRIGGERED COMMUNICATION
Edited by
ROMAN OBERMAISSER
CRC Press

Ensuring Reliable Networks

www.tttech.com

Wilfried Steiner
wilfried.steiner@tttech.com
TTTech Computertechnik AG