TECHNISCHE UNIVERSITÄT WIEN

TTTech

Dependable Computer Systems

Part 6b: System Aspects

Contents

- Synchronous vs. Asynchronous Systems
- Consensus
- Fault-tolerance by self-stabilization
- Examples
  - Time-Triggered Ethernet (FT Clock Synchronization)
  - Self-Stabilization in the Time-Triggered Architecture|
  - Traffic Policing
  - System Architectures

Synchronous vs. asynchronous systems

- **Synchronous processors:**
  A processor is said to be *synchronous* if it makes at least one processing step during Δ real-time steps (or if some other computer in the system makes *s* processing steps).
- **Bounded communication:**
  The communication delay is said to be *bounded* if any message sent will arrive at its destination within Φ real-time steps (if no failure occurs).
- **Synchronous system:**
  A system is said to be synchronous if its processors are synchronous **and** the communication delay is bounded.
  Real-time systems are per definition synchronous.
- **Asynchronous systems:**
  A system is said to be asynchronous if either its processors are asynchronous **or** the communication delay is unbounded

Consensus

Consensus

- Each processor starts a protocol with its local input value, which is sent to all other processors in the group, fulfilling the following properties:
  - Consistency: All correct processors agree on the same value and all decisions are final.
  - Non-triviality: The agreed-upon input value must have been some processors input (or is a function of the individual input values).
  - Termination: Each correct processor decides on a value within a finite time interval.

Consensus (cont.)

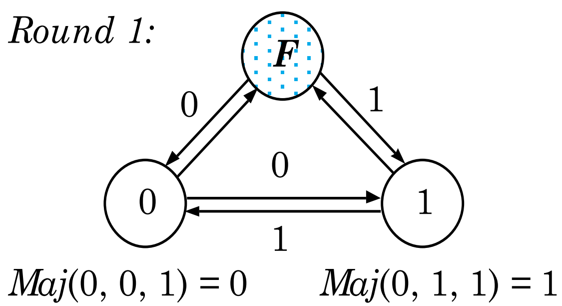
- The consensus problem under the assumption of byzantine failures was first defined in 1980 in the context of the SIFT project which was aimed at building a computer system with ultra-high dependability. Other names are
  - byzantine agreement or byzantine general problem
  - interactive consistency

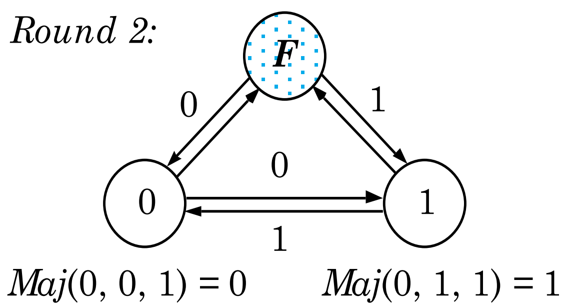
Impossibility of deterministic consensus in asynch. systems

- asynchronous systems cannot achieve consensus by a deterministic algorithm in the presence of even one crash failure of a processor
- it is impossible to differentiate between a late response and a processor crash
- by using coin flips, probabilistic consensus protocols can achieve consensus in a constant expected number of rounds
- failure detectors which suspect late processors to be crashed can also be used to achieve consensus in asynchronous systems

Impossibility of deterministic consensus in asynch. systems (cont.)

$n \geq 3t + 1$ processors are necessary to tolerate t failures





Fault-tolerance by self-stabilization

Fault-tolerance by self-stabilization

Self-Stabilization: A distributed system S is self-stabilizing with respect to some global predicate P if it satisfies the following two properties:

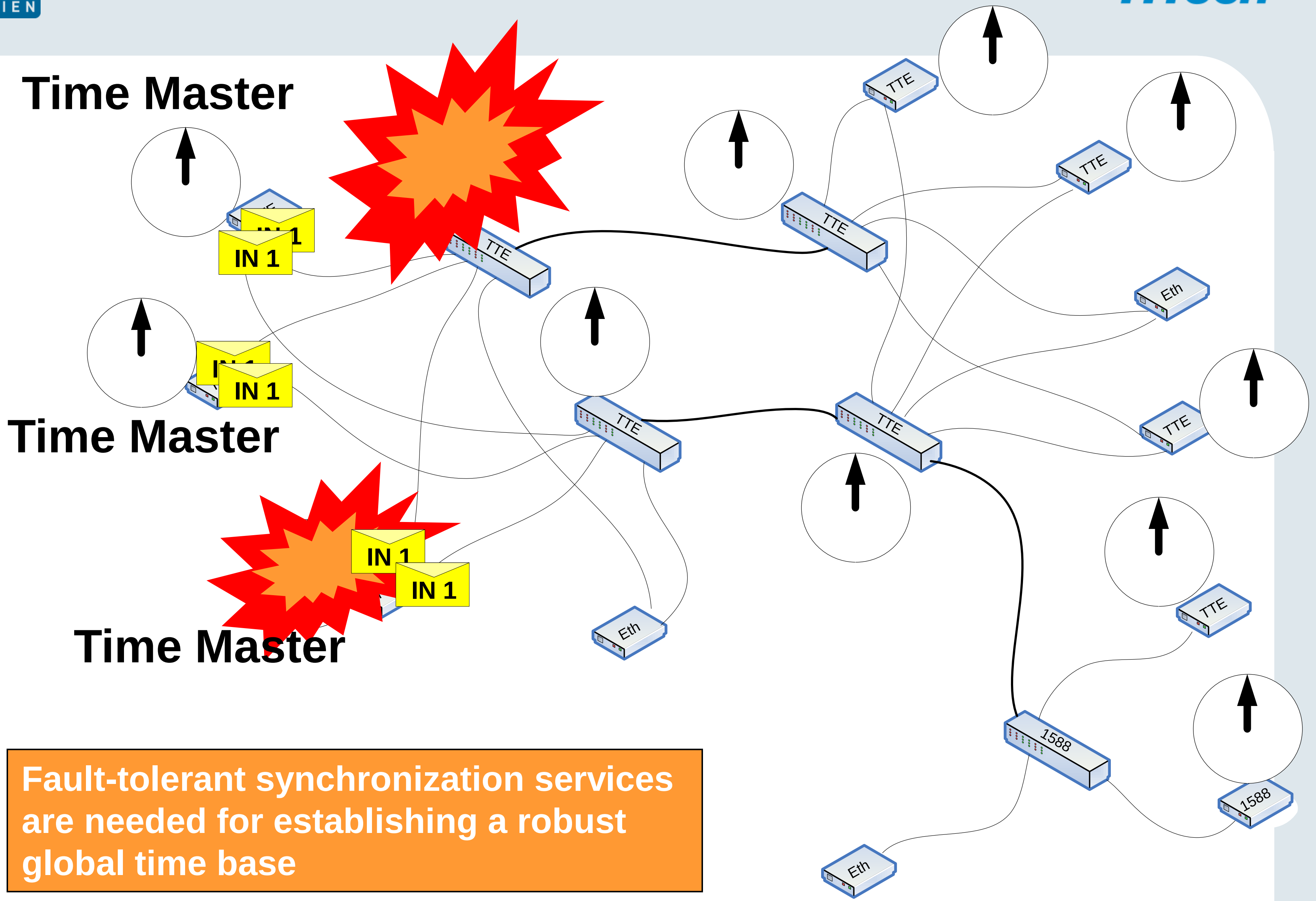
1. Closure: P is closed under the execution of S. That is, once P is established
 in S, it cannot be falsified.
2. Convergence: Starting from an arbitrary global state, S is guaranteed to
 reach a global state satisfying P within a finite number of state transitions

Fault-tolerance by self-stabilization (cont.)

- self-stabilizing systems need not be initialized and they can recover from transient failures (adaptive DSD is self-stabilizing)
- self-stabilization is a different approach to fault-tolerance, it is not based on countering the effects of failures but concentrating on the ability to reach a consistent state
- problems are how to achieve a global property with local actions and local knowledge, lack of theory on how to design self-stabilizing algorithms and how to guarantee timeliness

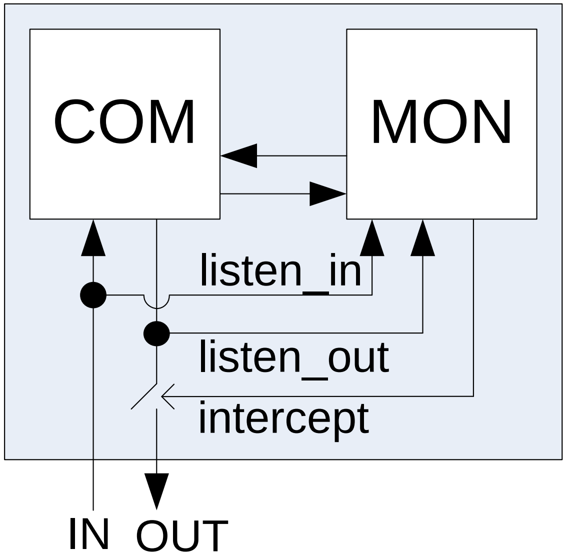
Examples

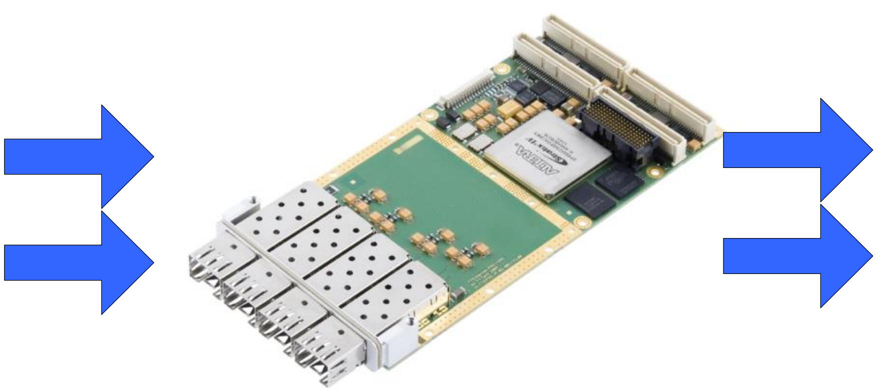
- Time-Triggered Ethernet (FT Clock Synchronization)
- Self-Stabilization in the Time-Triggered Architecture|
- Traffic Policing
- System Architectures



Fault-tolerant synchronization services are needed for establishing a robust global time base

Failure Model for High-Integrity Components: Inconsistent-Omission Faulty





Core COM/MON Assumptions:

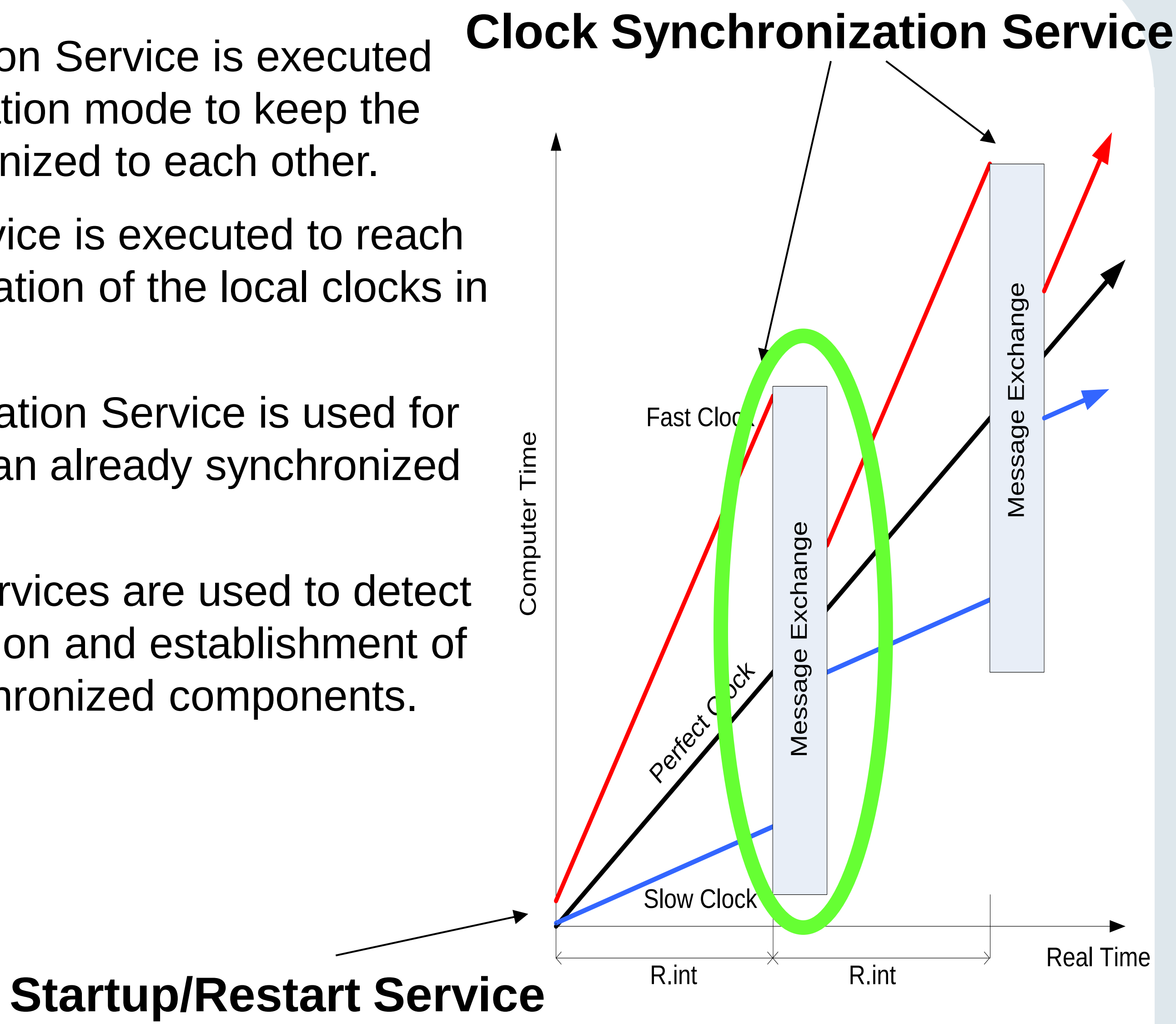
- COM and MON fail independently
- MON can intercept a faulty message produced by the COM
- COM cannot produce a valid message such that this message appears as two different messages on listen_out and OUT; though it may be valid on listen_out but detectable faulty on OUT or vice versa
- MON cannot itself generate a faulty message, neither by inverting listen_out to an output, nor by toggling the intercept signal

Clock Synchronization Service is executed during normal operation mode to keep the local clocks synchronized to each other.

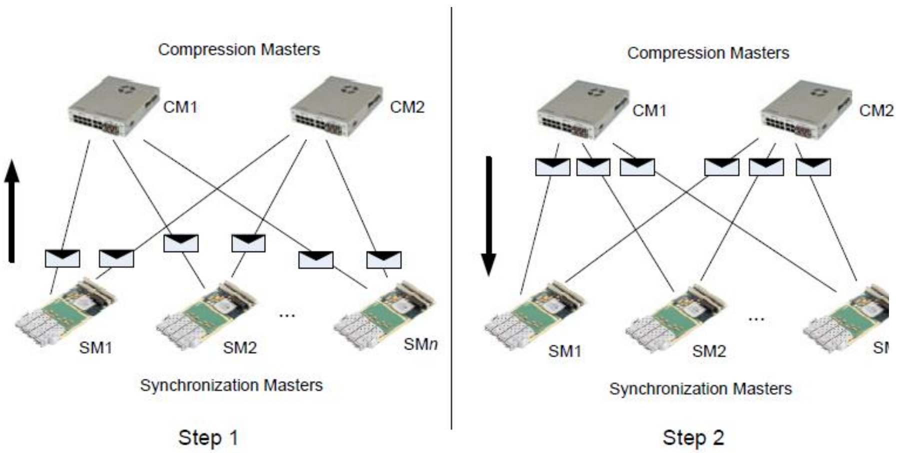
Startup/Restart Service is executed to reach an initial synchronization of the local clocks in the system.

Integration/Reintegration Service is used for components to join an already synchronized system.

Clique Detection Services are used to detect loss of synchronization and establishment of disjoint sets of synchronized components.

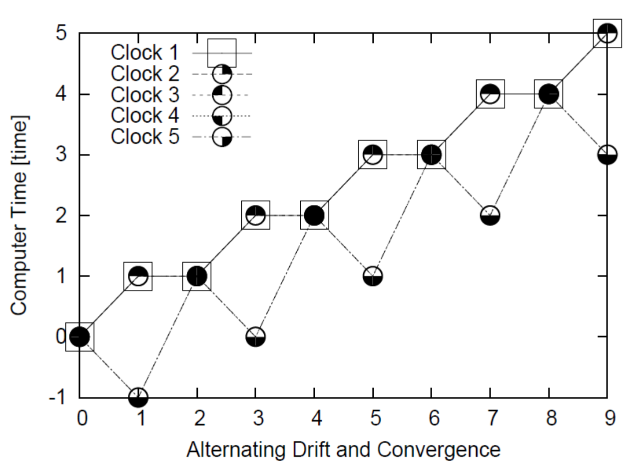


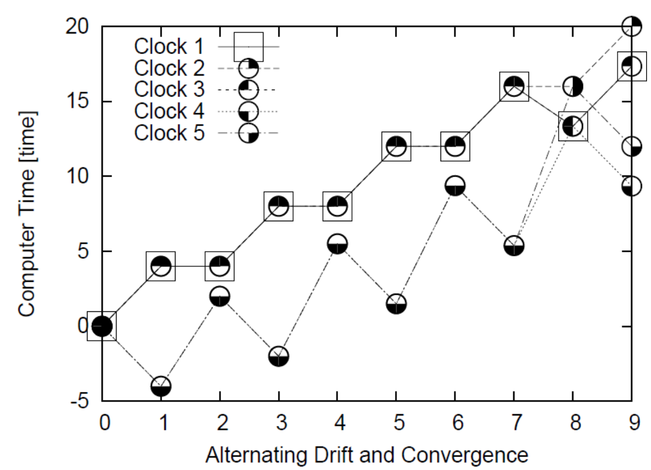
Algorithm Specification



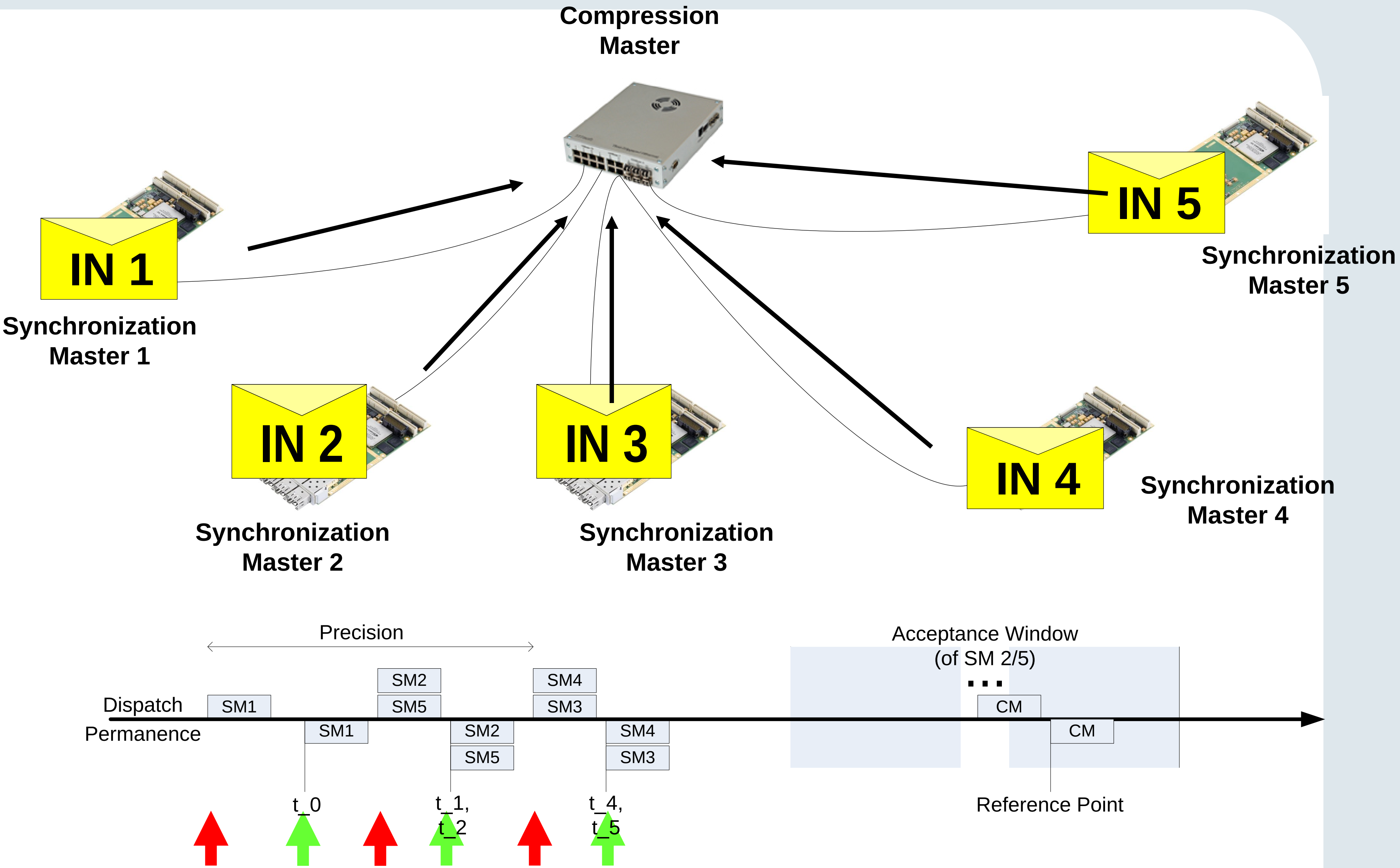
- one SM clock: $compressed\_clock = SM\_clock_1$
- two SM clocks:
  $compressed\_clock = \frac{SM\_clock_1 + SM\_clock_2}{2}$
- three SM clocks: $compressed\_clock = SM\_clock_2$
- four SM clocks:
  $compressed\_clock = \frac{SM\_clock_2 + SM\_clock_3}{2}$
- five SM clocks: $compressed\_clock = SM\_clock_3$
- more than five SM clocks: take the average of the $(k+1)^{th}$ largest and $(k+1)^{th}$ smallest clocks, where k is the number of faulty SMs that have to be tolerated.

- one CM clock: $SM\_clock = CM\_clock_1$
- two CM clocks:
  $SM\_clock = \frac{CM\_clock_1 + CM\_clock_2}{2}$
- three CM clocks: $SM\_clock = CM\_clock_2$

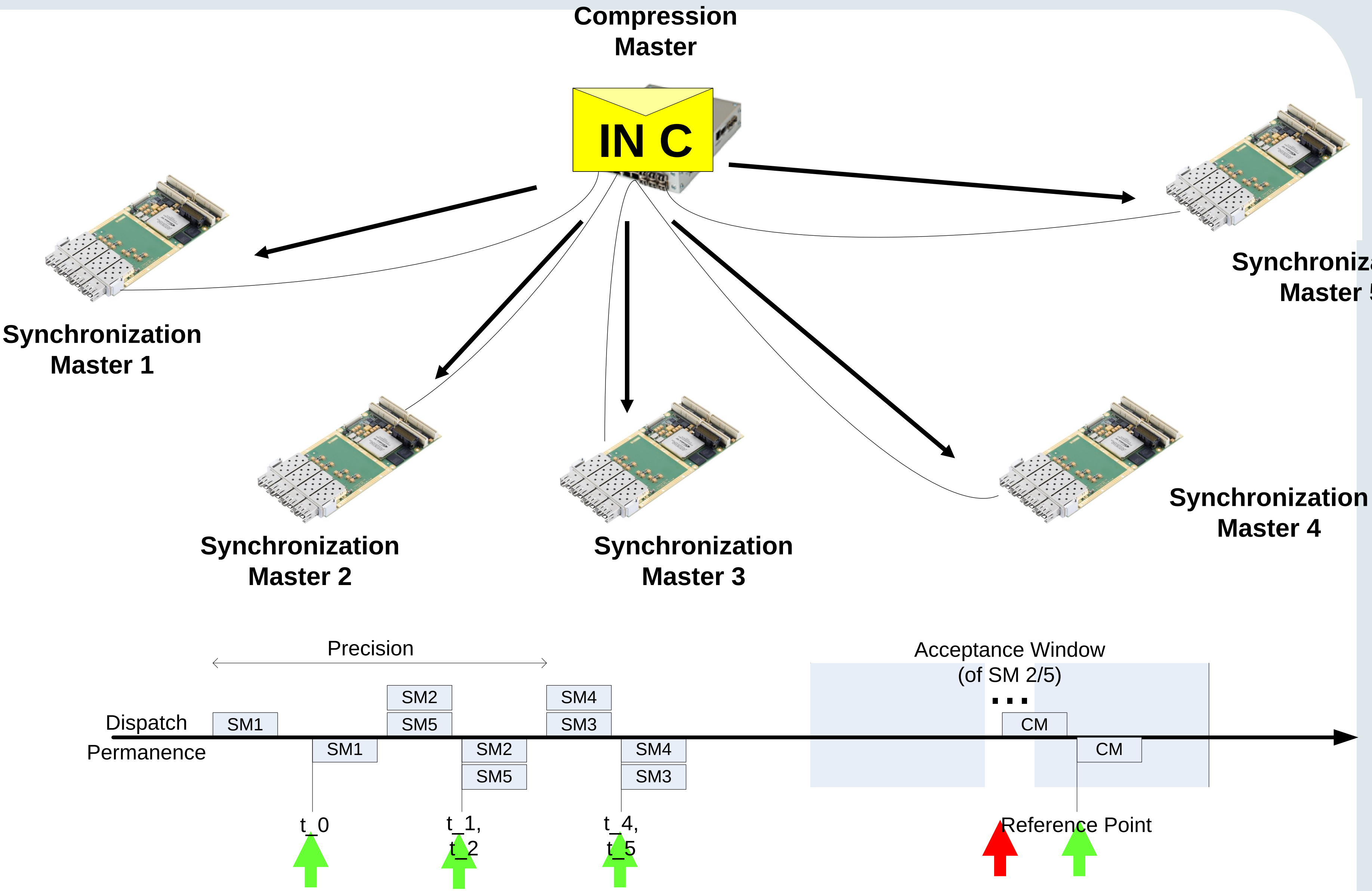




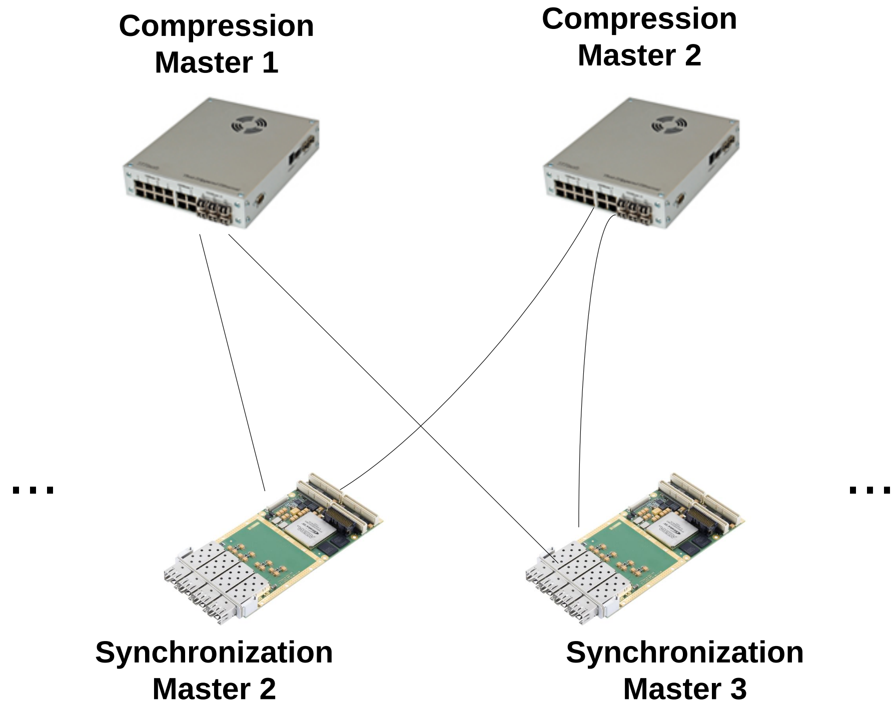
Step 1: SMs send messages to CMs



Step 2: CMs send voted clock values back

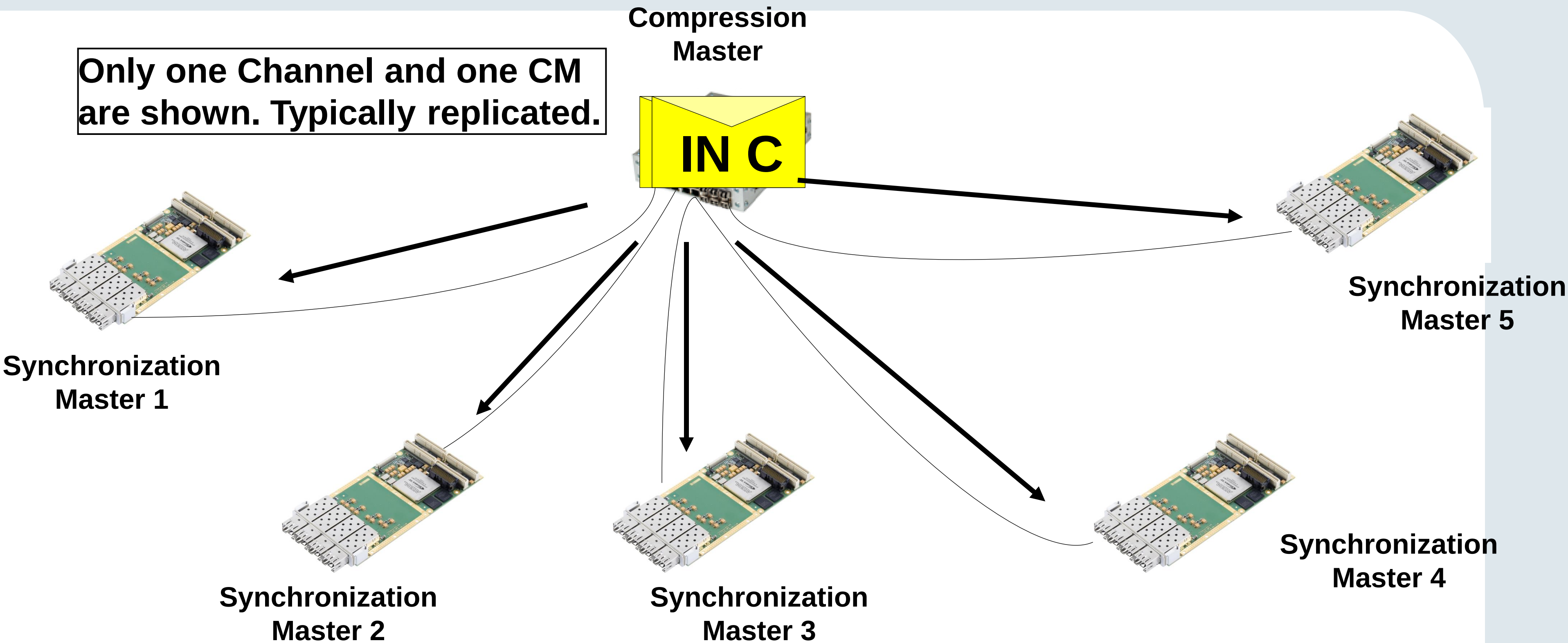


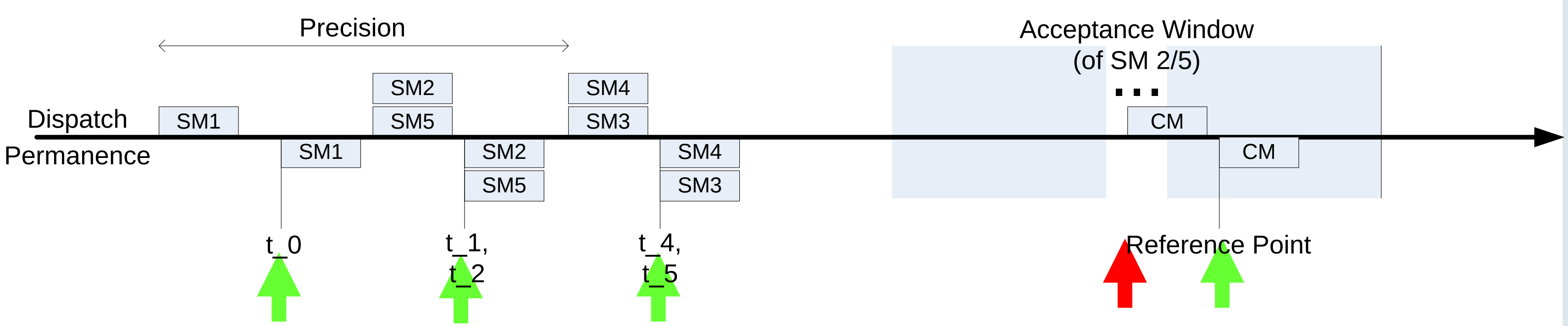
Step 2: Multiple Channels/CMs



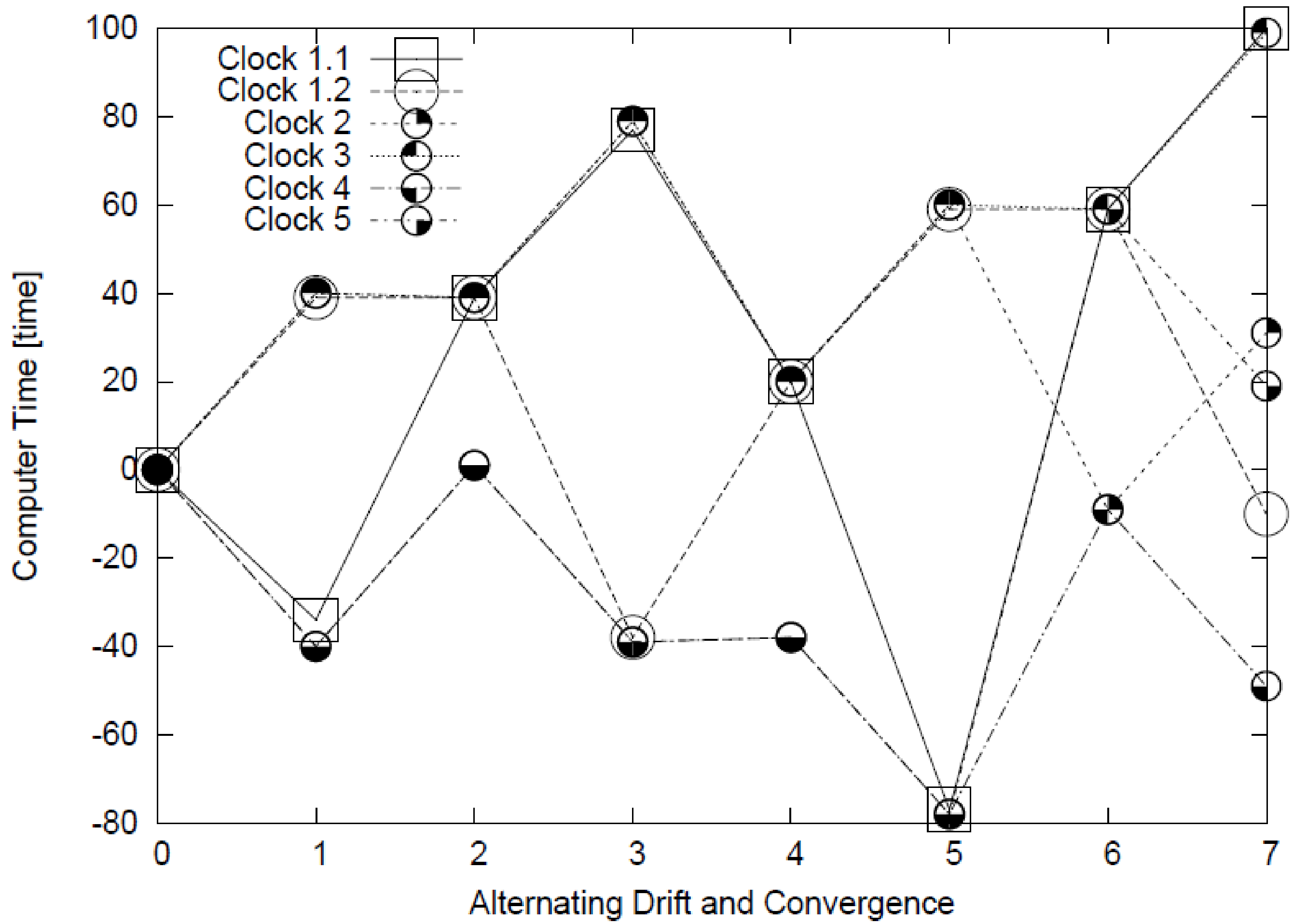
- Multiple Channels/CMs are required for fault-tolerance.
- Synchronization Masters (SMs) receive synchronization messages from all non-faulty Compression Masters (CMs)
- SMs use either the median or the arithmetic mean on the redundant messages from the CMs.

Step 2: Faulty CMs send to some SMs





Clock Synchronization Two Failures Scenario



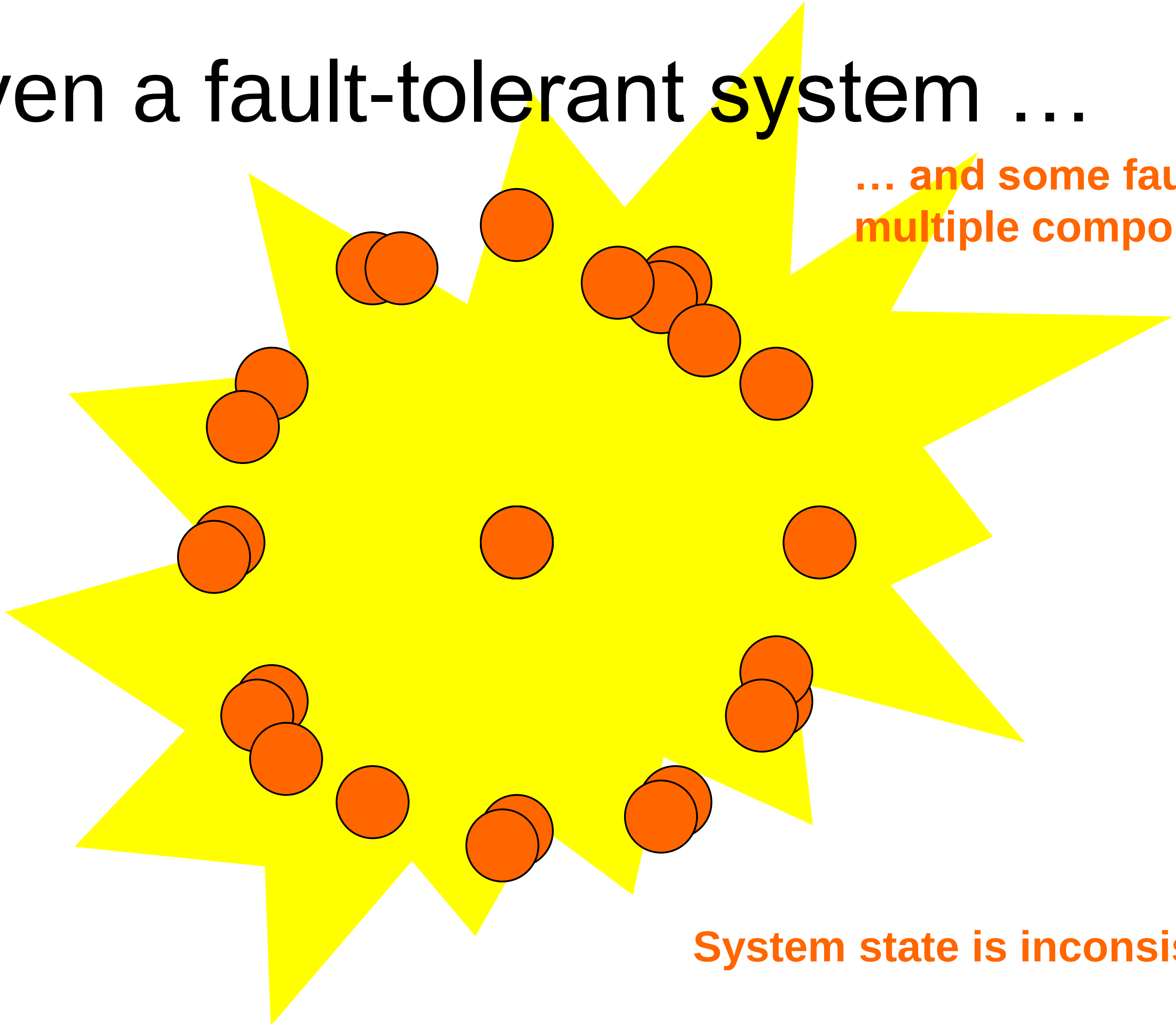
Even in the byzantine failure case, the non-faulty clocks remain synchronized with known bounds.

Examples

- Time-Triggered Ethernet (FT Clock Synchronization)
- Self-Stabilization in the Time-Triggered Architecture
- Traffic Policing
- System Architectures

Given a fault-tolerant system …

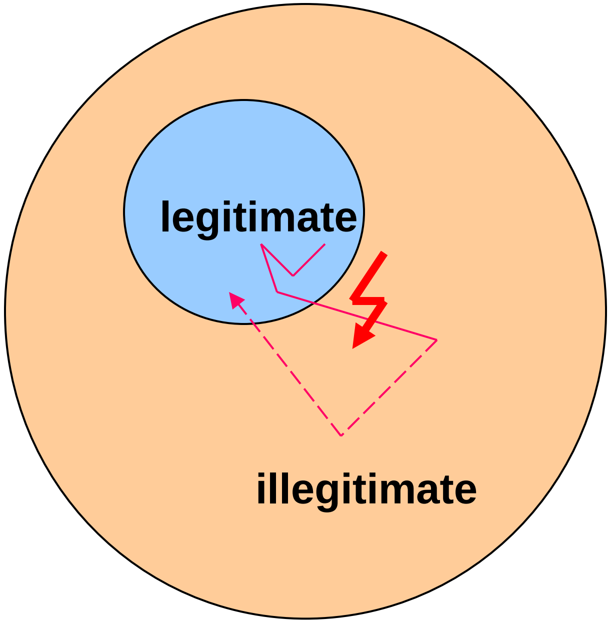
… and some fault affecting multiple components.



System state is inconsistent!

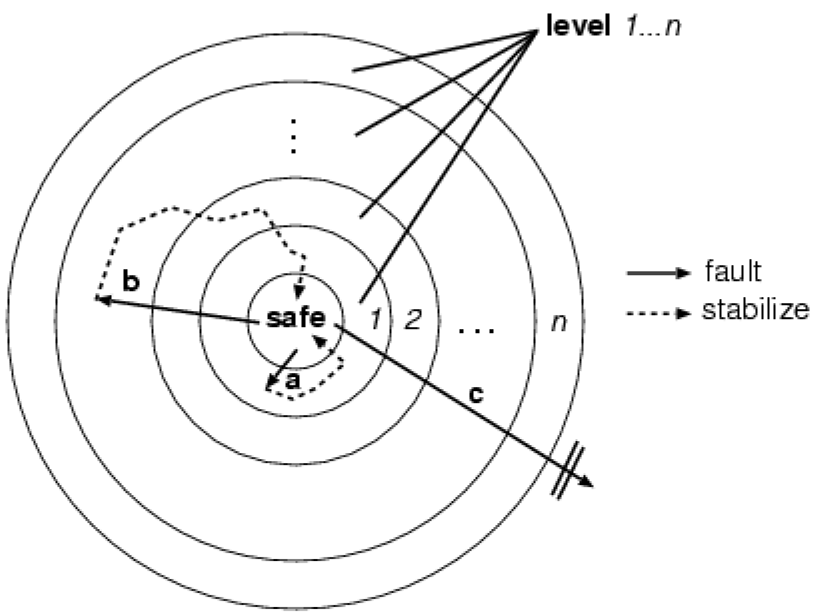
Self-Stabilization (Dijkstra 1974)

- Two properties:
  - Closure
  - Convergence
- Closure: If a system is in a legitimate state it will remain in this state
- Convergence: A system will eventually reach a legitimate state, from an arbitrary state



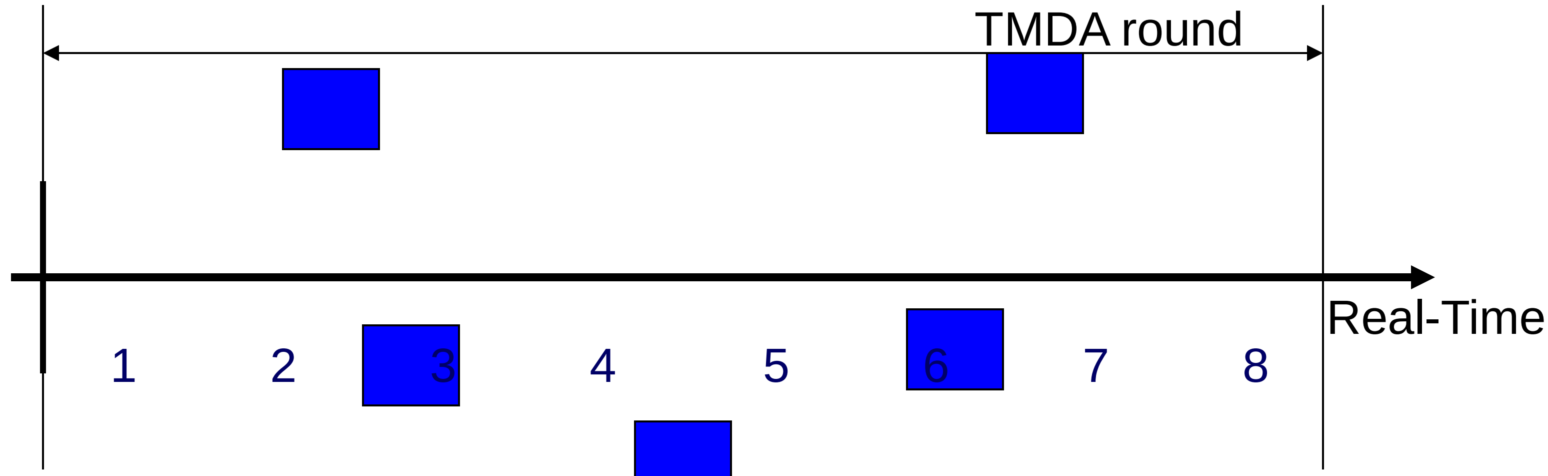
Self-stabilization and fault-tolerance

- Different failure detection mechanism and failure correction mechanisms for different failures
- Using a sequence of algorithms to bring system “nearer” to the safe state



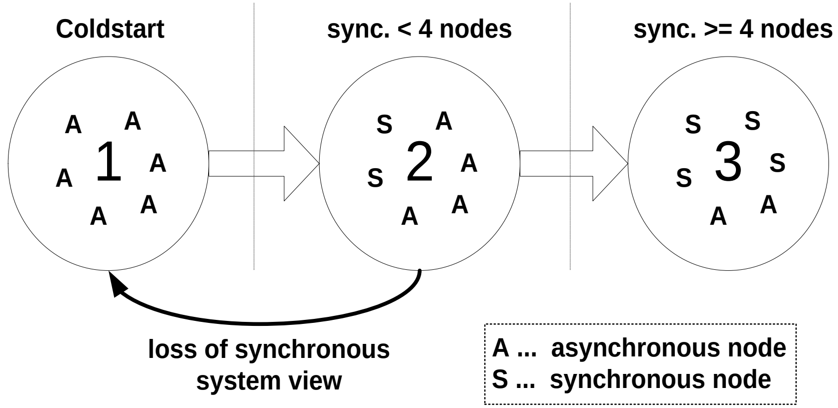
Time-Triggered Architecture (cont.)

- Time is split-up into (not necessarily equal) slots depending on message length
- Slots are grouped into TDMA rounds



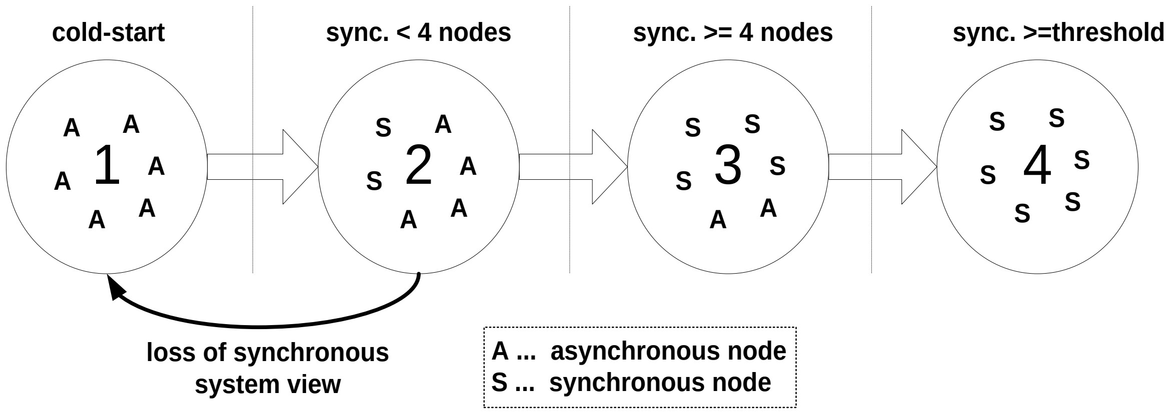
- Each node has assigned exactly one slot in the TDMA round
- This assignment is equal for every TDMA round
- Actual Transmission Phase < Sending Slot

Phases of the TTP



- 3 Phases:
  - coldstart,
  - synchronous < 4 nodes, no guarantees for the system services, sync. messages are broadcasted periodically
  - synchronous >= 4 nodes (normal system operation), sync. messages are broadcasted periodically

Phases of the TTP/C (cont.)



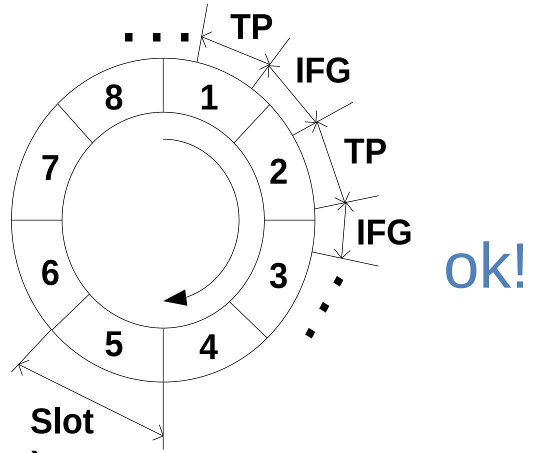
- Identifying a 4th phase of protocol execution:
  - A *sufficient* number of nodes is synchronous

Cliques

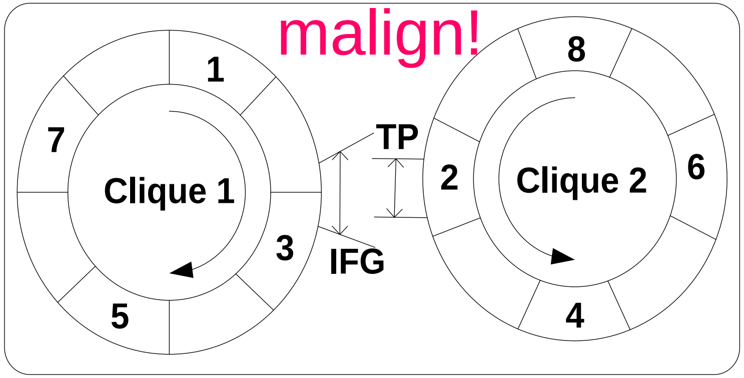
- Nodes that communicate with each other form a clique
- In correct operation mode only one clique
- Possibility of more cliques after multiple transient failures
- Two types of multiple cliques operation:
  - Benign:
    - Synchronous operation
  - Malign:
    - Asynchronous operation

Cliques (cont.)

- Benign Cliques:
  - Act still “slot-synchronously”
  - accept and reject counters are used to determine the amount of nodes in the own clique
- Malign Cliques:
  - Multiple cliques act “slot-asynchronously”
  - Cliques do not “see” each other
    - Clique A sends in the IFGs of Clique B and vice versa



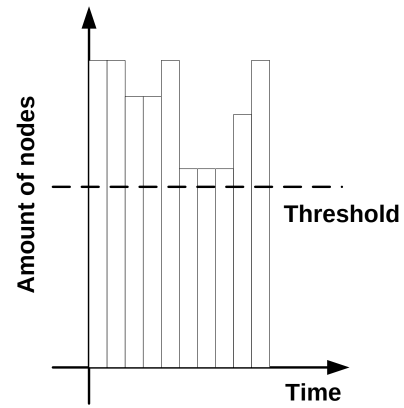
a)

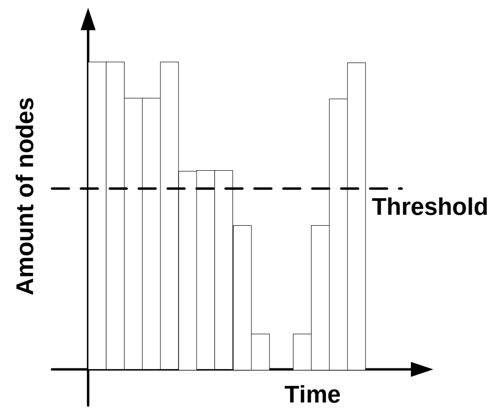


b)

Self-Stabilization and the TTA (cont.)

- Detect system misbehavior using the Membership Vector:
  - If >= (n/2)+1 nodes set in the membership: ok!
  - If < (n)/2+1 nodes set in membership: restart!
- Bring the system in a safe state again by using the startup algorithm and reintegration of TTP



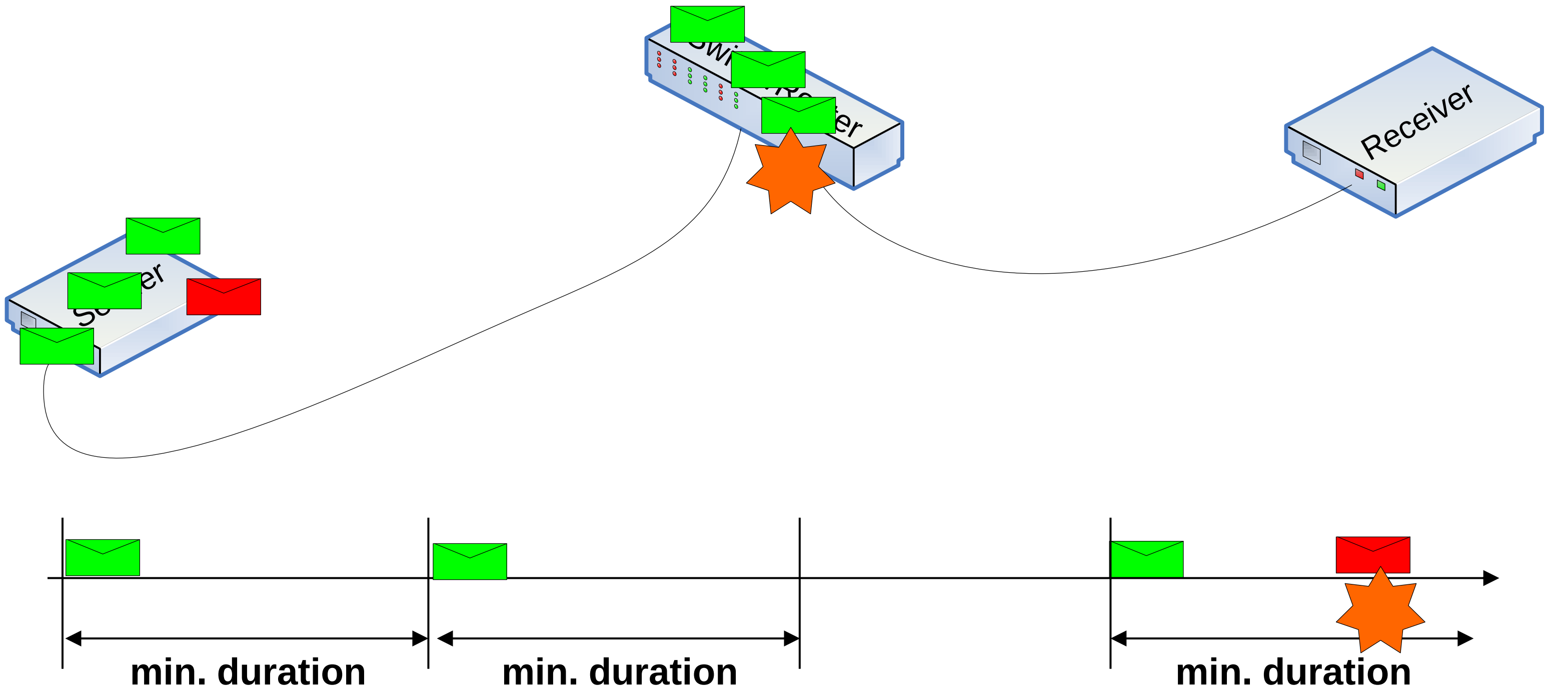


Examples

- Time-Triggered Ethernet (FT Clock Synchronization)
- Self-Stabilization in the Time-Triggered Architecture
- Traffic Policing
- System Architectures

Leaky-Bucket Traffic Policing

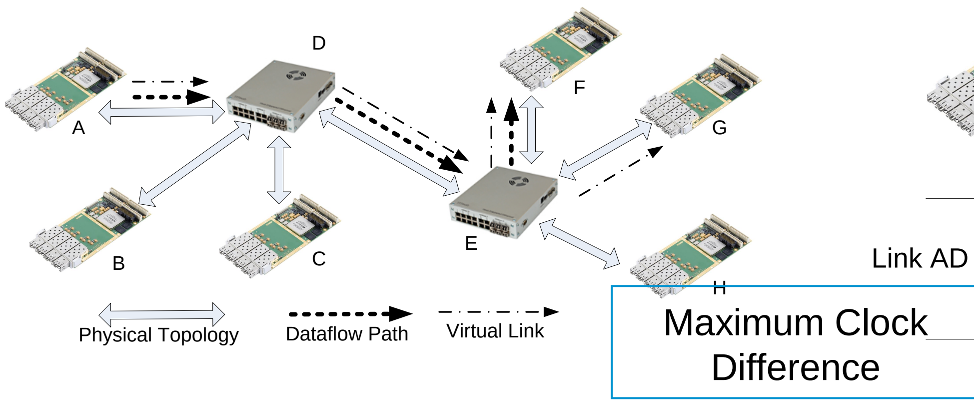
Rate-Constrained Traffic (RC)

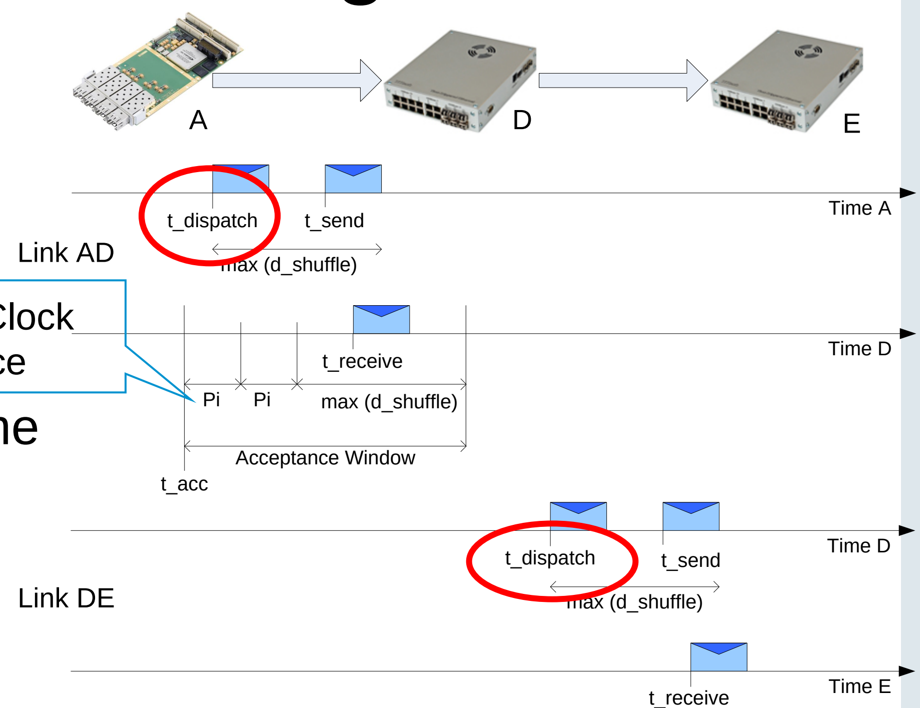


Leaky-Bucket Traffic Policing (cont.)

- Token-Bucket / Leaky-Bucket algorithm are implemented to control the behavior of rate-constrained traffic.
- In the case when a faulty end point attempts to send frames too close back-to-back, then the token/leaky-bucket algorithm will detect this behavior and drop the frame.
- Token/leaky-bucket algorithms may be expensive to implement as they require to track the timing on a per VL basis.

Scheduled Traffic Policing





Transmission Sequence of a TT frame

1. Frame dispatch at t_dispatch
2. Frame start sending at t_send
3. Frame start receiving at t_receive

Temporal Correctness is checked via Acceptance Window Test

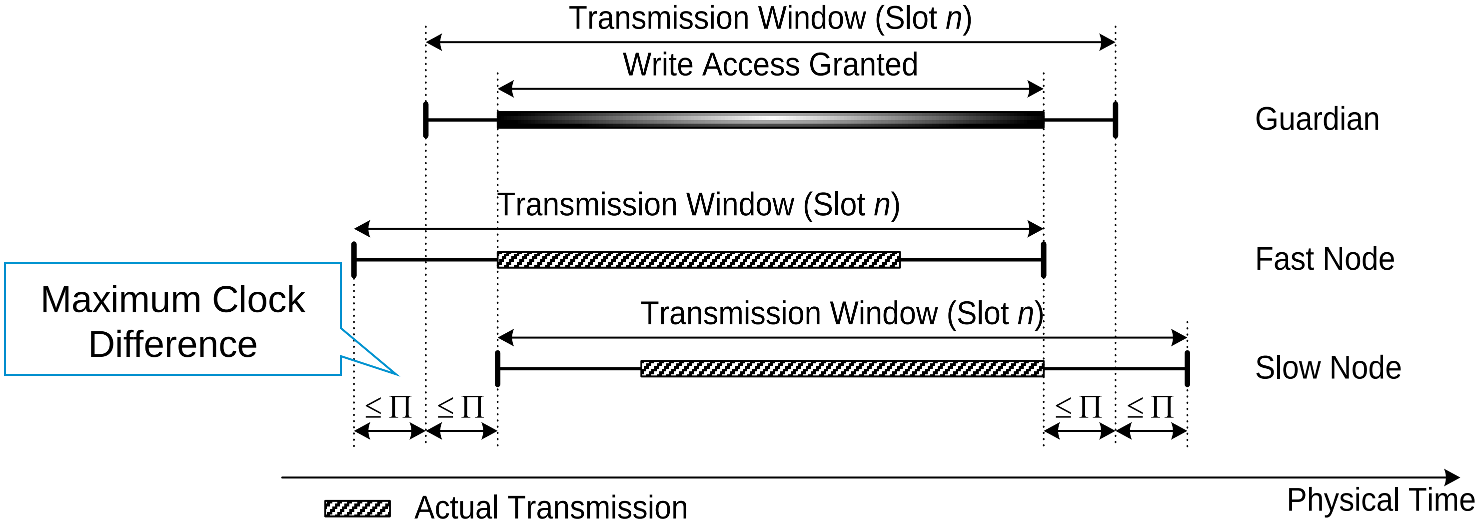
- t_receive = t_send + l_link (l_link … link latency)
- t_acc = t_dispatch + l_link – Pi
  (Pi … maximum distance of any two synchronized correct clocks in the system)

Scheduled Traffic Policing (cont.)

- Scheduled traffic policing checks the correctness of a received message with respect to a synchronized global time.
- Scheduled traffic policing enforces minimum durations as well as maximum durations in between two successive messages of the same stream.

Central Bus Guardian

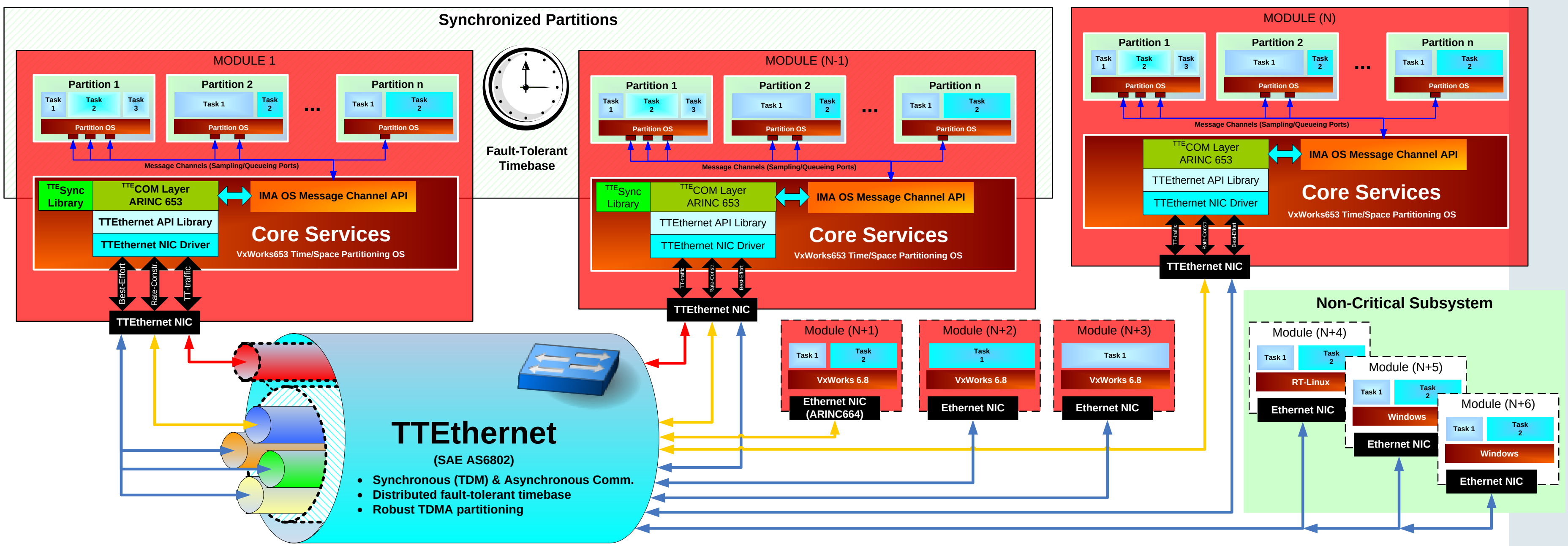
- Nodes and switches are synchronized to each other with a precision Pi.
- In a cut-through setting the TDMA slot needs to account four times the precision (4*Pi) as a safety margin.
- Only then it is guaranteed that traffic policing in the switch does not truncate correct transmissions.



Examples

- Time-Triggered Ethernet (FT Clock Synchronization)
- Self-Stabilization in the Time-Triggered Architecture
- Traffic Policing
- System Architectures

Distributed Integrated Modular Avionics (Distributed IMA)

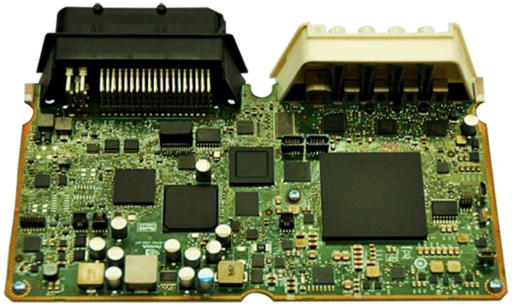


Distributed Integrated Modular Avionics (Distributed IMA) (cont.)

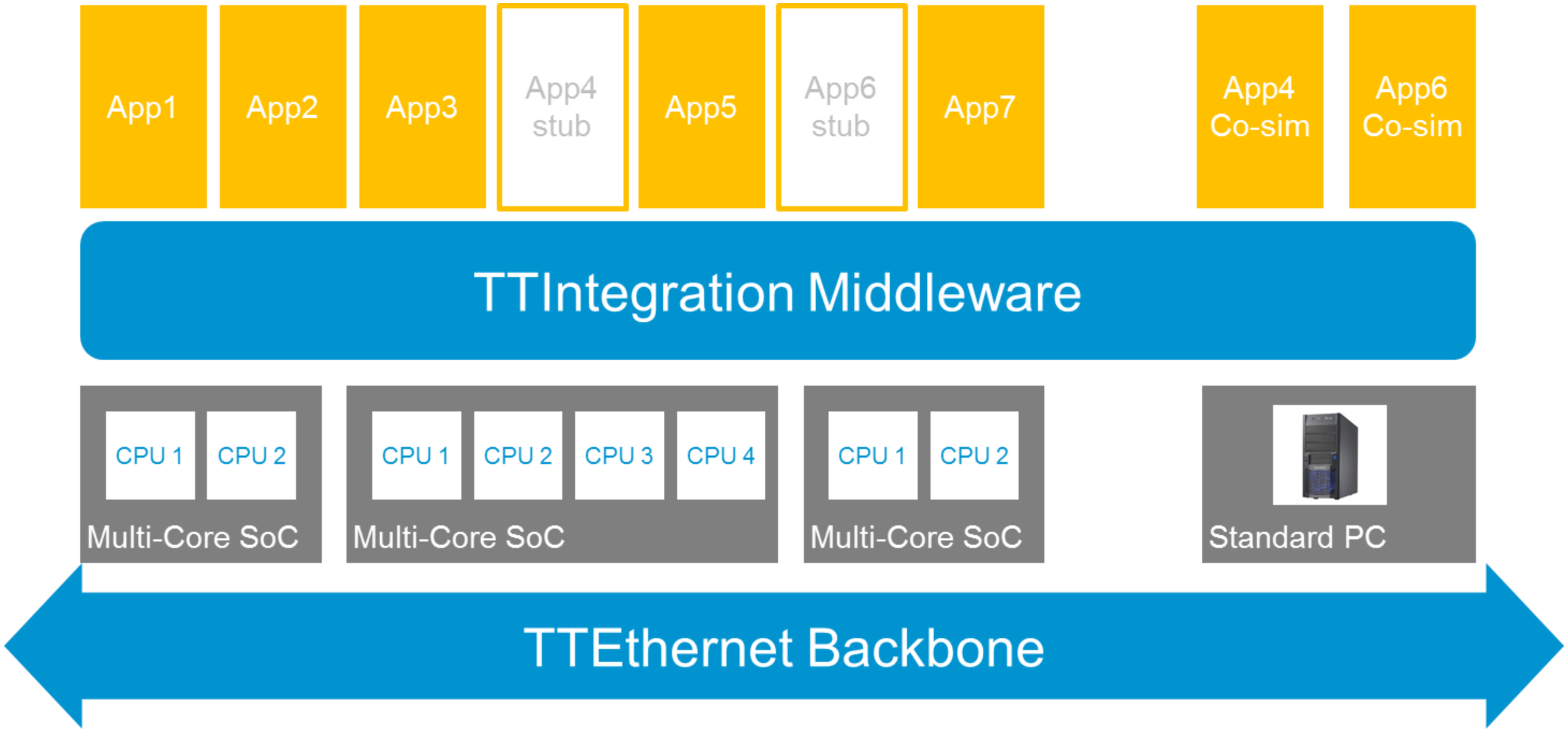
- The network is a distributed fault-tolerant embedded computer and executes a set of partitions
  - synchronous TDMA communication for Ethernet allows integration of low-latency, low-jitter VLs in complex networks
  - „System-level partitioning“ closes the gap between federated and integrated architectures
- The partitions can be mutually aligned and synchronized to system time

Automotive Integrated Safety Platform

- zFAS, co-developed with TTTech, enabling Audi to integrate a variety of innovative functions with multiple safety criticality levels.
- zFAS uses numerous technology components from TTTech. For example, the individual CPU cores are connected based on Deterministic Ethernet communication



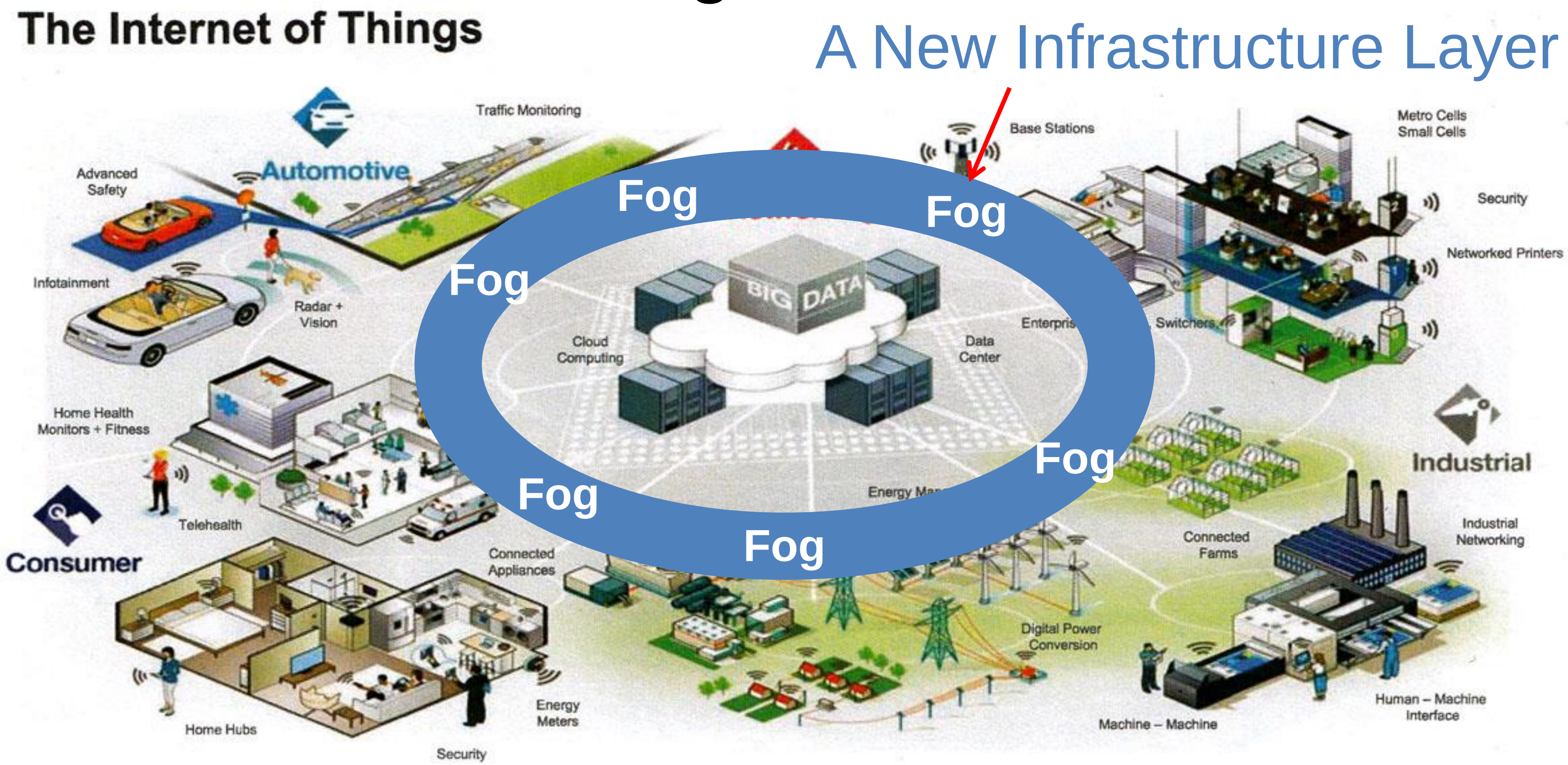
Automotive Integrated Safety Platform (cont.)



Automotive Integrated Safety Platform (cont.)

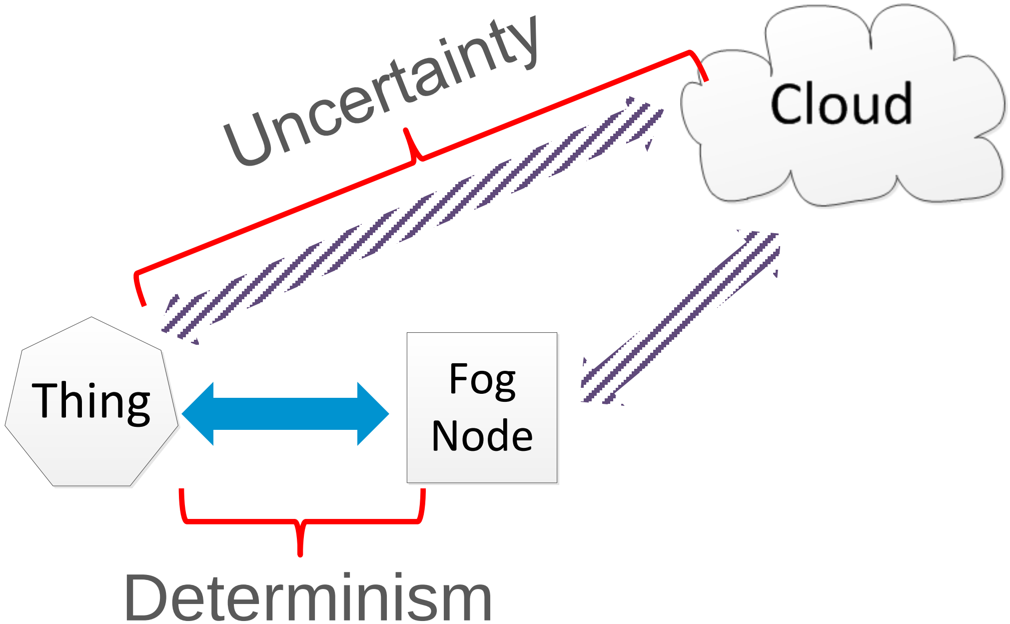
- Combines high-performance computing with functional safety
- Highly efficient deterministic SW integration
- Applications can be moved between embedded cores
- Supports various SoCs (System-on-a-Chip) and operating systems
- Accelerated development due to PC-based co-simulation

Fog Computing for the Dependable Internet of Things



Fog Computing for the Dependable Internet of Things (cont.)

Fog computing is an architecture approach that provides non-functional knowledge to enable dependability in the IoT.



Fog Computing for the Dependable Internet of Things (cont.)

+

+ ?

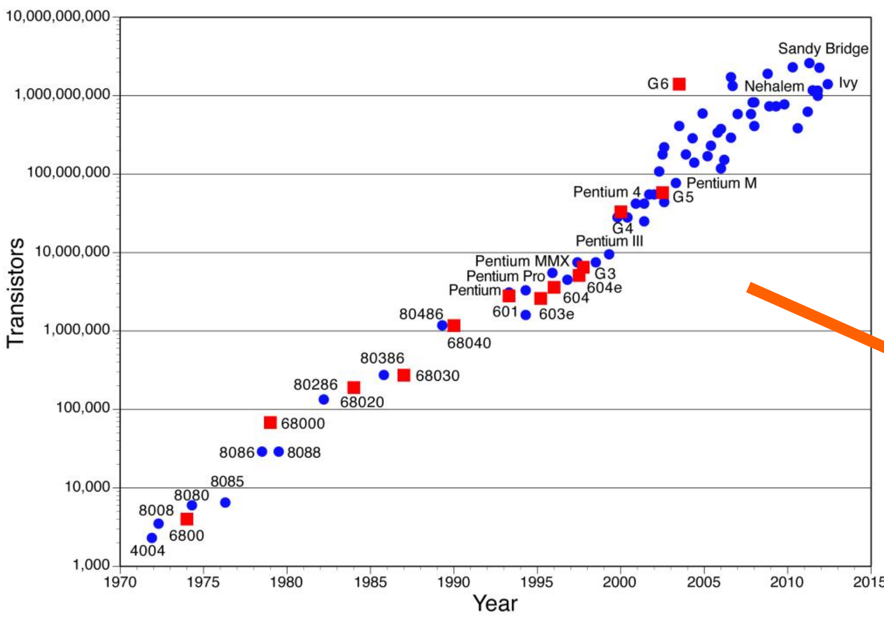
Multiple Components

Modularized & Cross-Industry

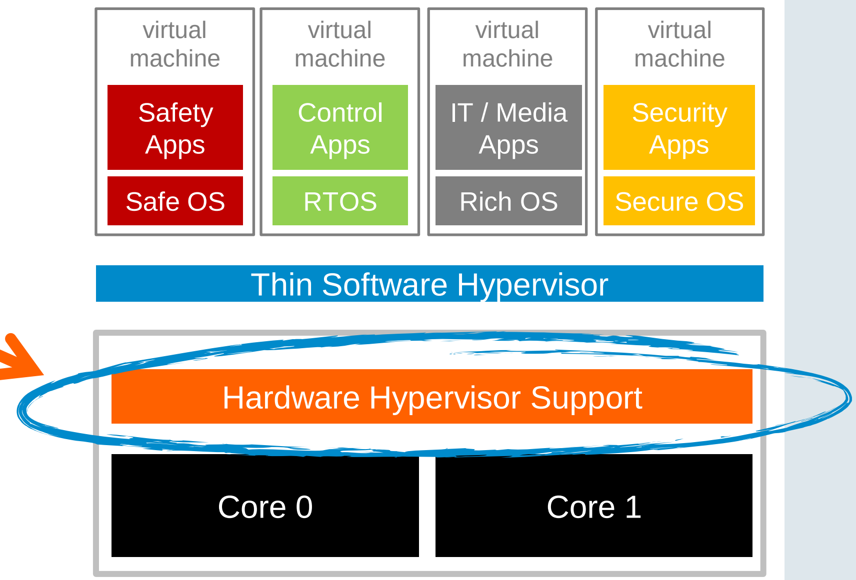
Fog Computing for the Dependable Internet of Things (cont.)

Moore's Law Alive and Well

Heterogeneous Multi-Core



Hardware Supported Virtualization at Chip Level Possible



“Fog Node on Wheels”