Timed Asynchronous System Models for Dependable Mobile/Pervasive/* Systems

Christof Fetzer

Dresden University of Technology

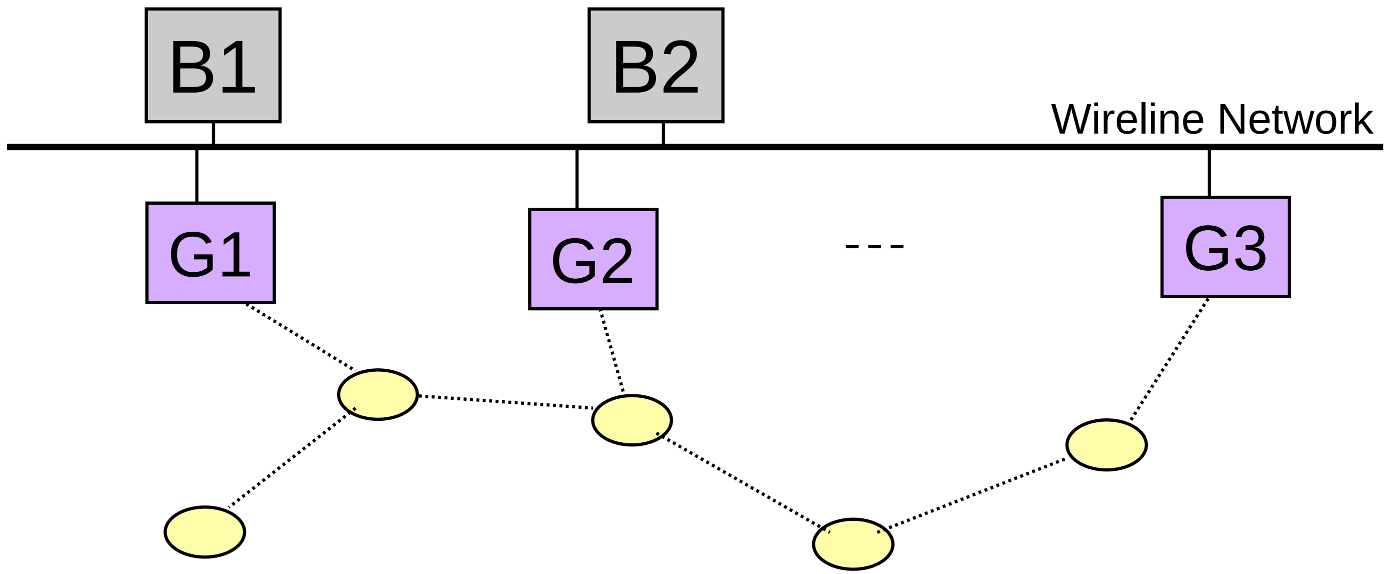
Germany

Application Domain: **Technology Assisted Living**

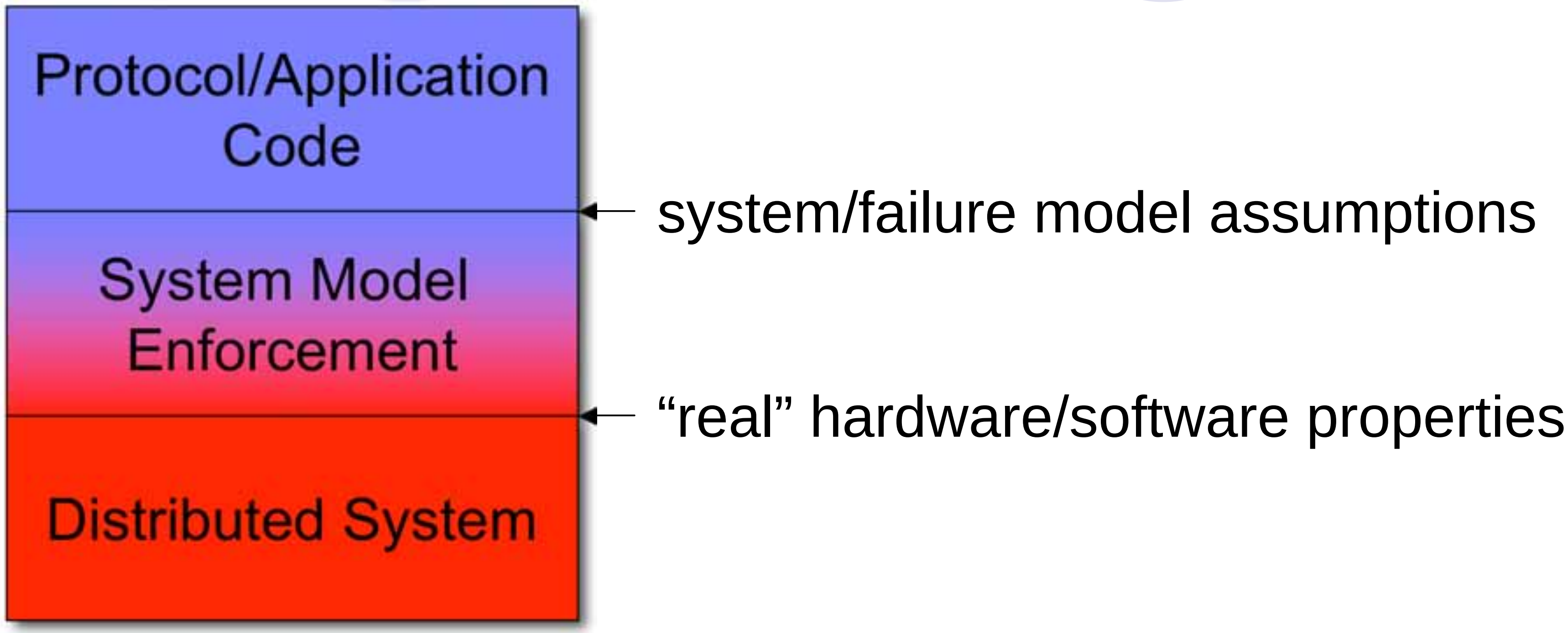
- Home/garden sensor network
  - e.g.: Intel uses motion sensors to check the health status of persons
- Need for dependability
  - application is safety critical...
- Some sort of physical security

Underlying Distributed System

- Mobile nodes
- Network technologies
  - Wireless and wired Ethernet



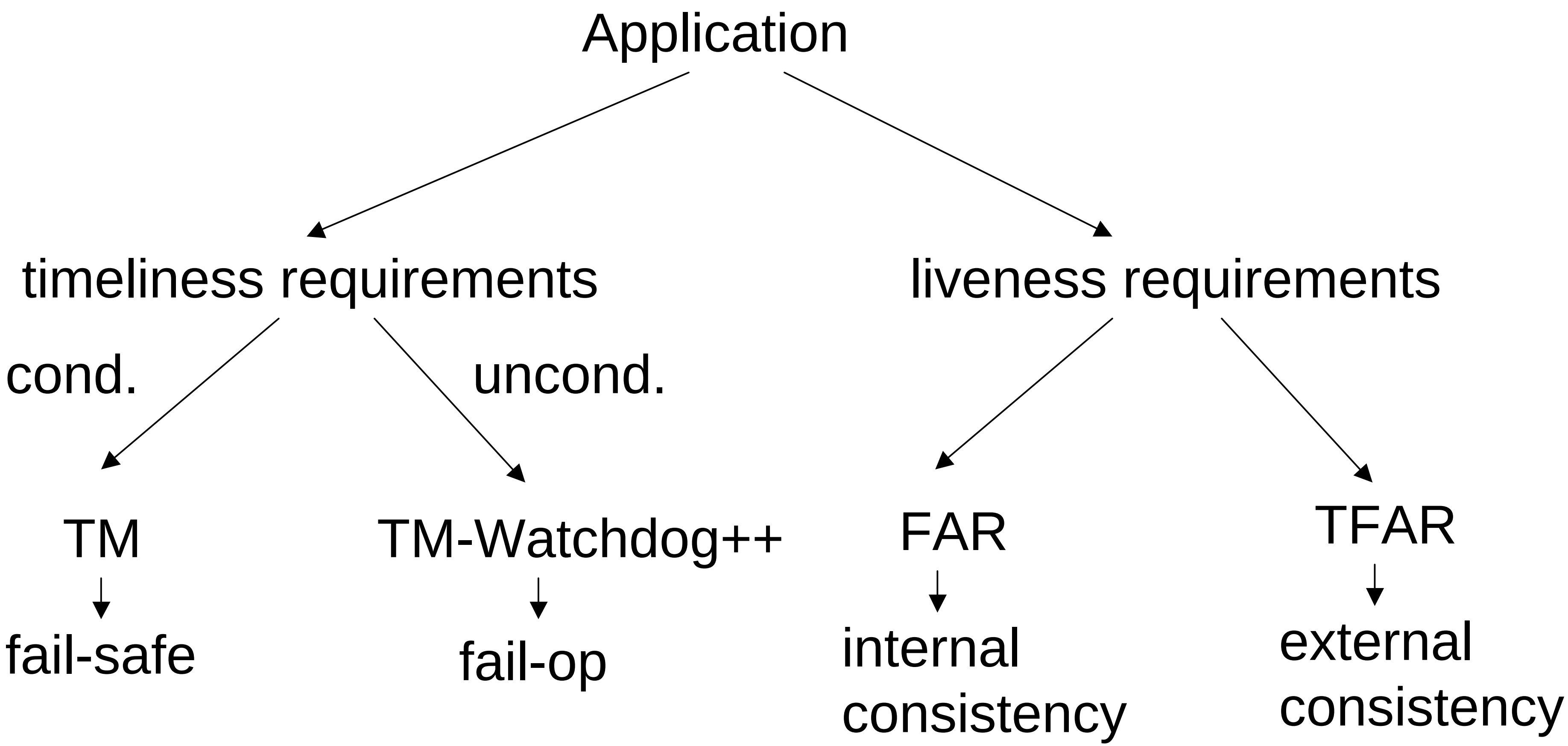
System Model Assumptions



Goals:

1) Simplify protocol development & permit correctness proofs
2) Probability that assumptions are violated are negligible

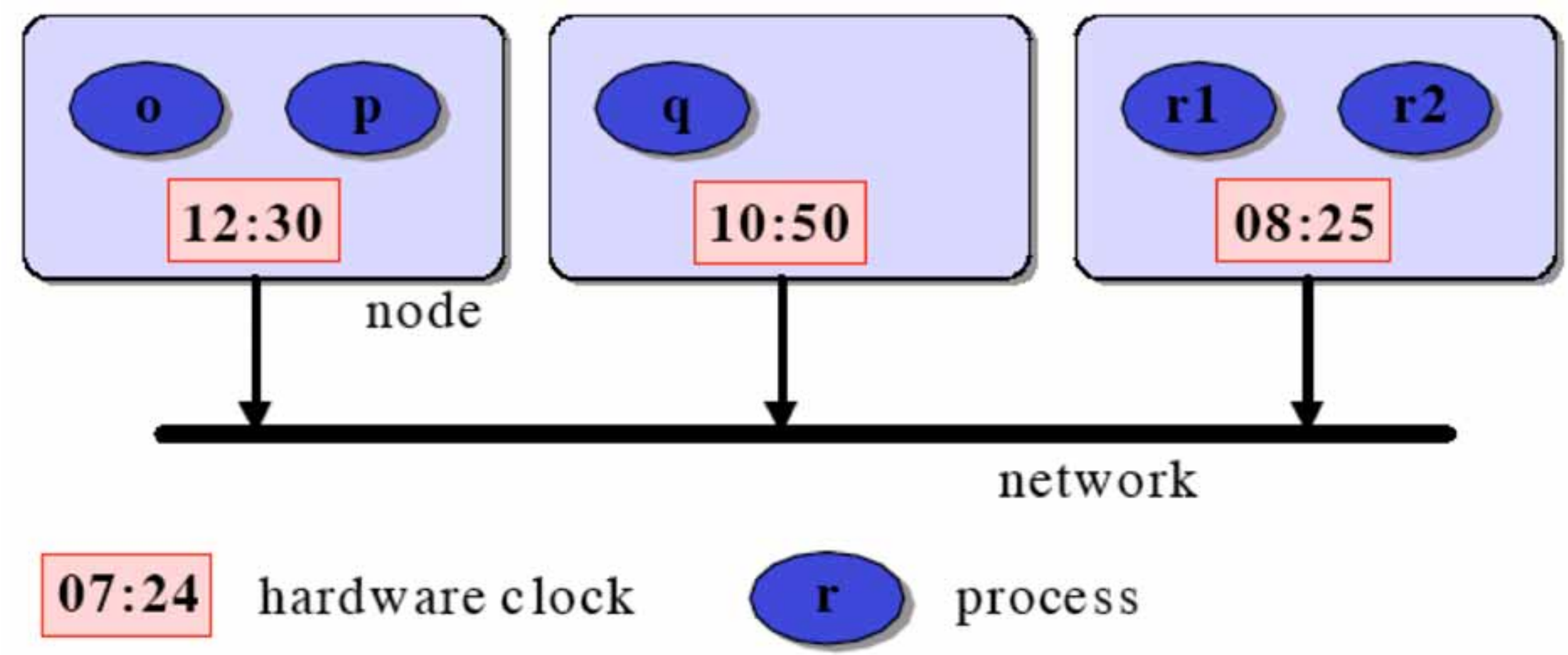
Application Dependency



Timed Asynchronous System Model (TM)

[1]

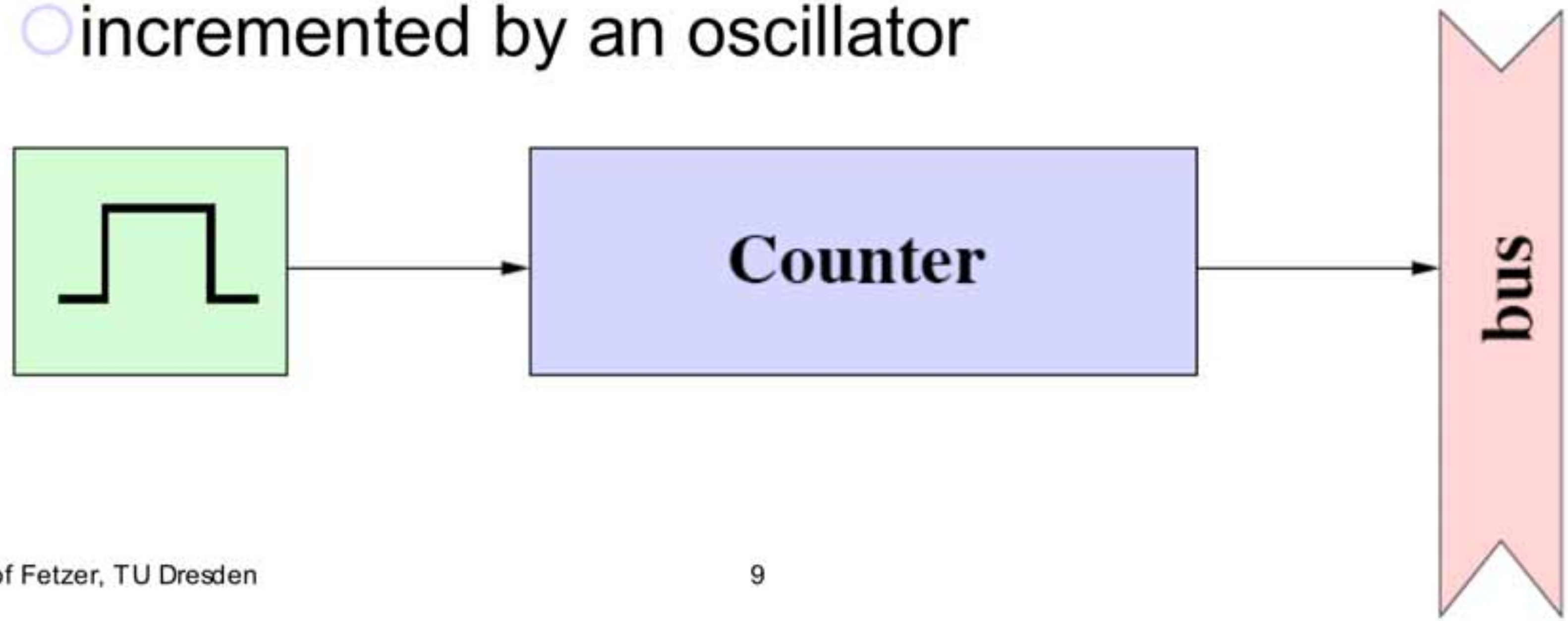
Services


o
p
q
r1
r2
12:30
10:50
08:25
node
network
07:24
hardware clock
r
process

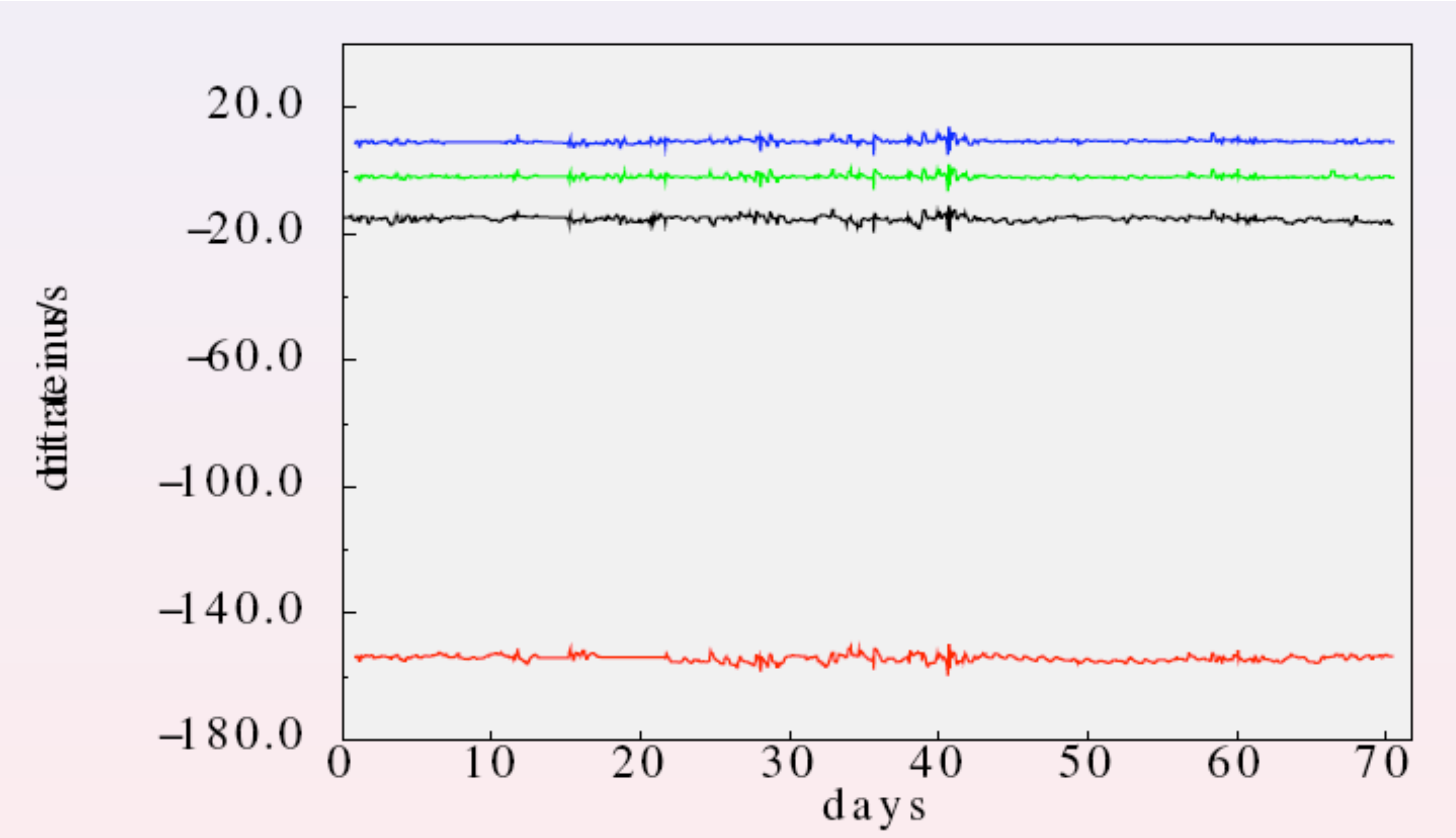
Local Hardware Clock Service

Local Hardware Clocks

- We assume that each computer p has a hardware clock Hp
- A hardware clock can be implemented by a hardware counter
  - incremented by an oscillator



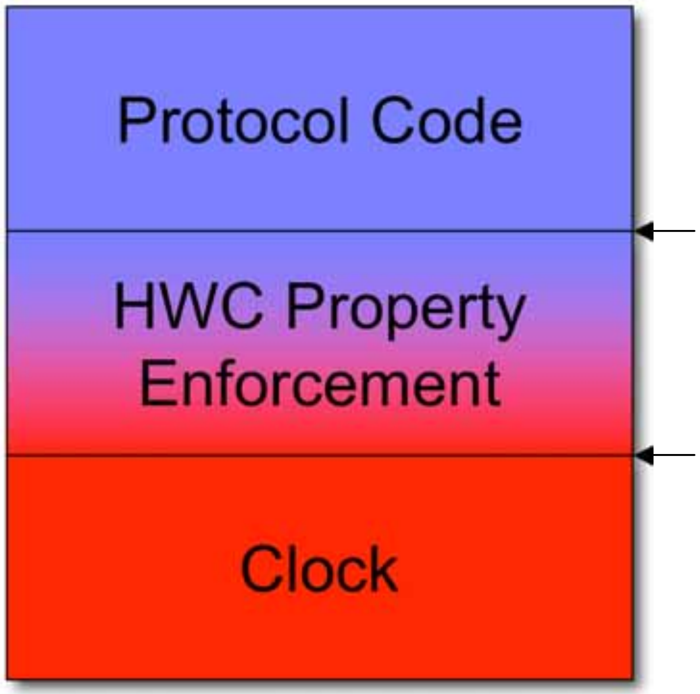
Measurements



Failure Assumption

- Failure Assumption:
  - Each correct process has a correct hardware clock, i.e., clock with a bounded drift rate.
- *Bounded drift rate*:
  - process can measure length of a time interval [s,t] with a max. error of $\rho(t-s)$

Hardware Clock Enforcement



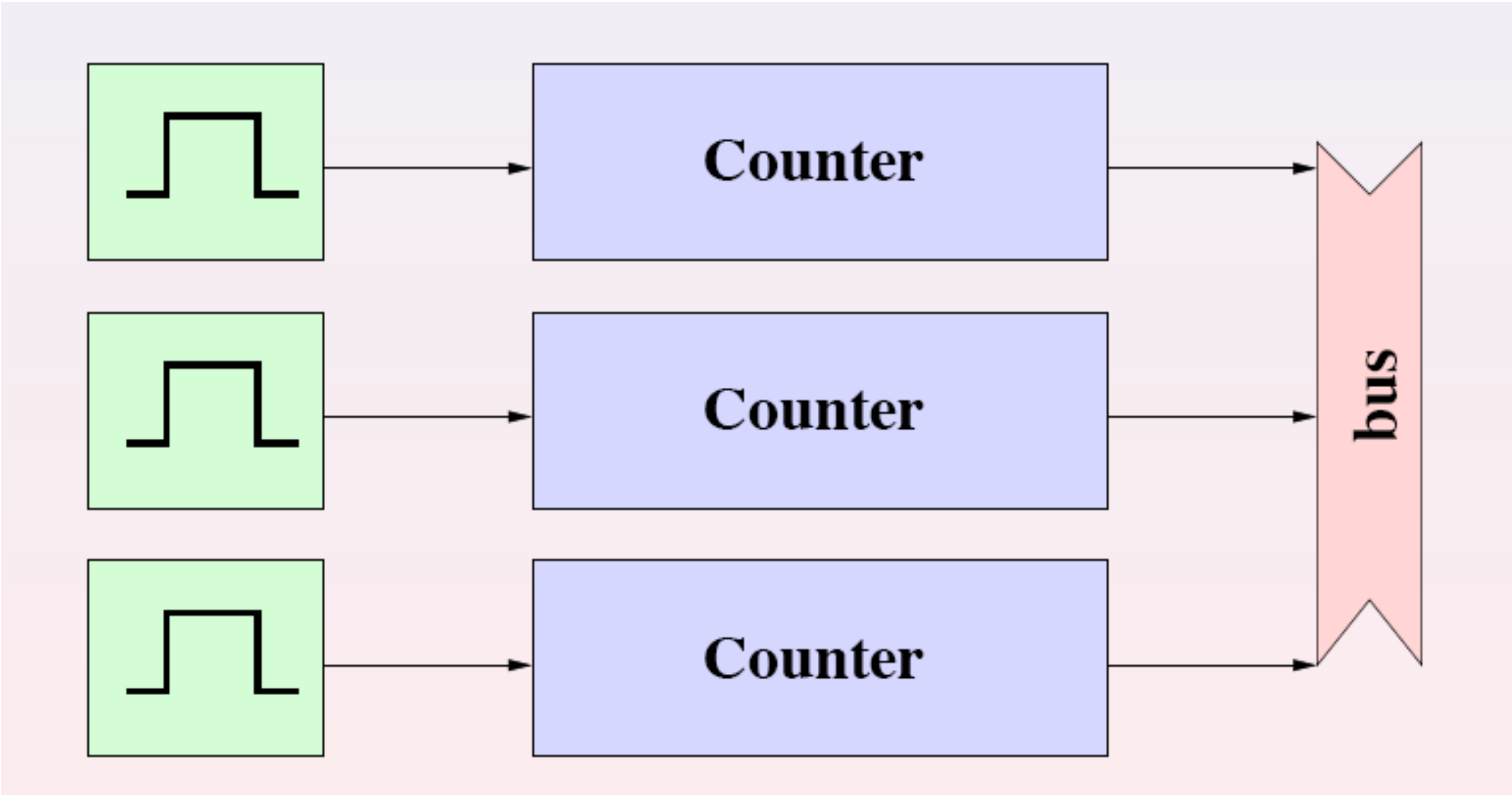
correct HWC

“real” properties

Clock Failure Semantics Enforcement

- We can try to detect clock failures and force a process to
  - crash if its hardware clock is faulty
- We can try to mask clock failures
- We can try to do both

Replicated Hardware Clock [2]



Replicated Hardware Clock

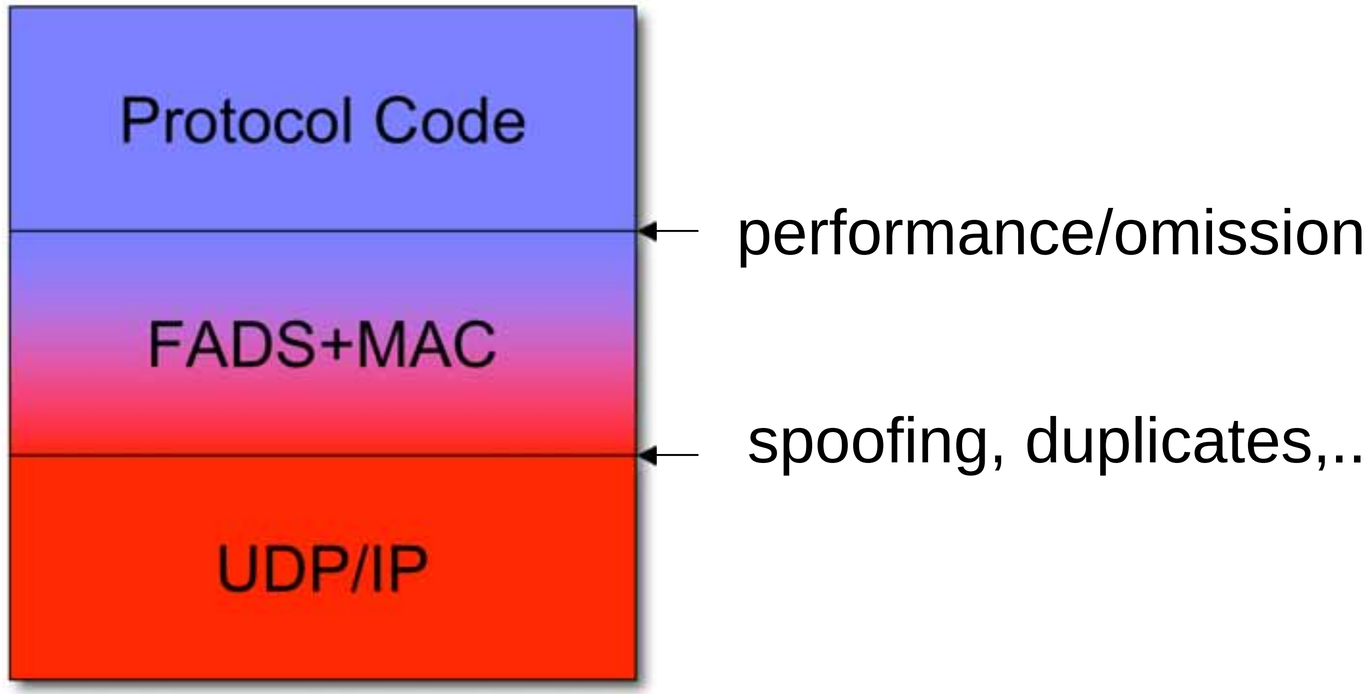
- Pentium processor has counter that is incremented in each cycle
  - Read counter with instruction: `rdtsc`
- Computers have hardware real-time clock
- Approach:
  - Can use different on-board clocks to enforce clock failure assumption

Datagram Service

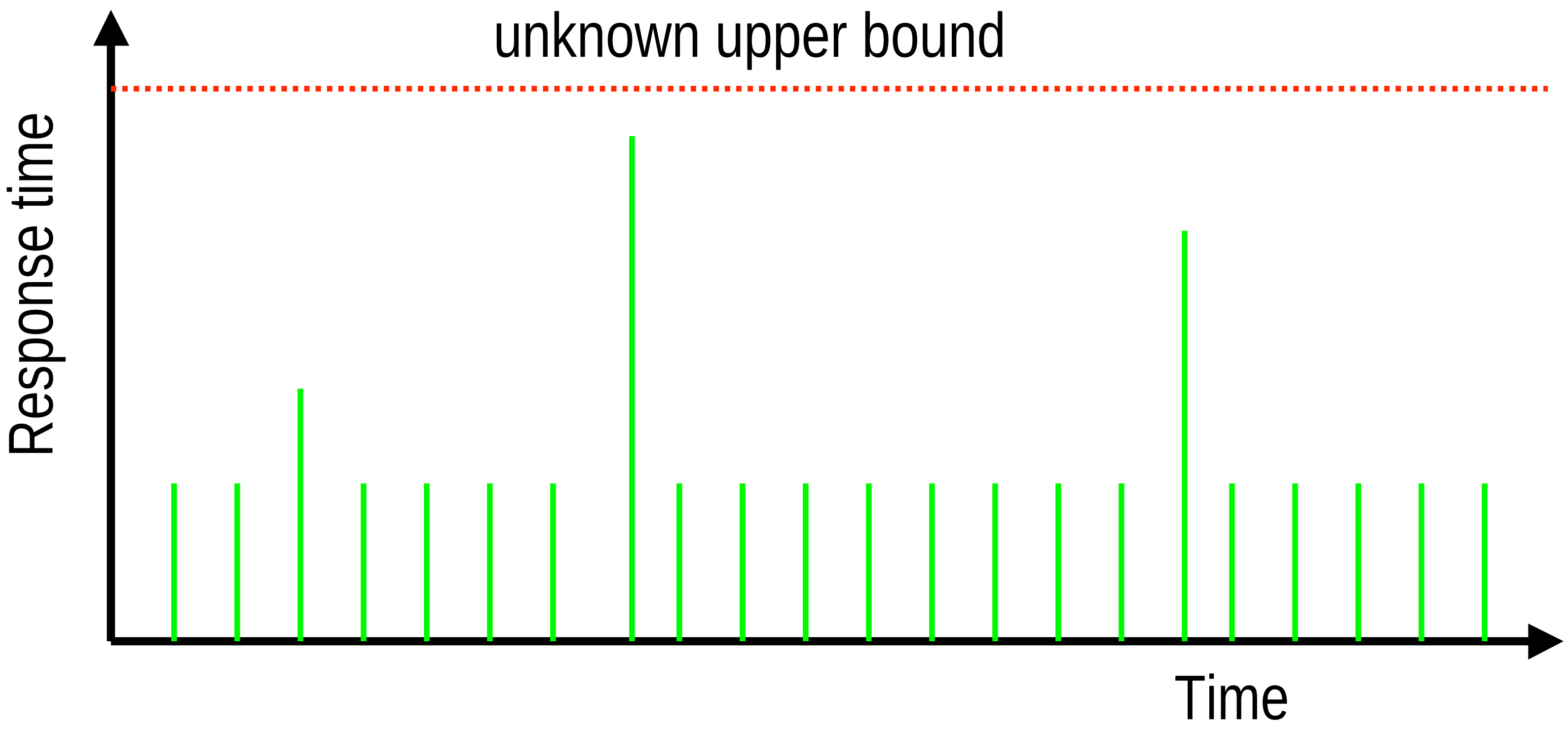
Datagram Service

- Semantics:
  - At most once delivery of messages
- Performance failure:
  - message transmission delay > δ.
- Omission failure:
  - message transmission delay = ∞
- Note: No bound on the number of failures!

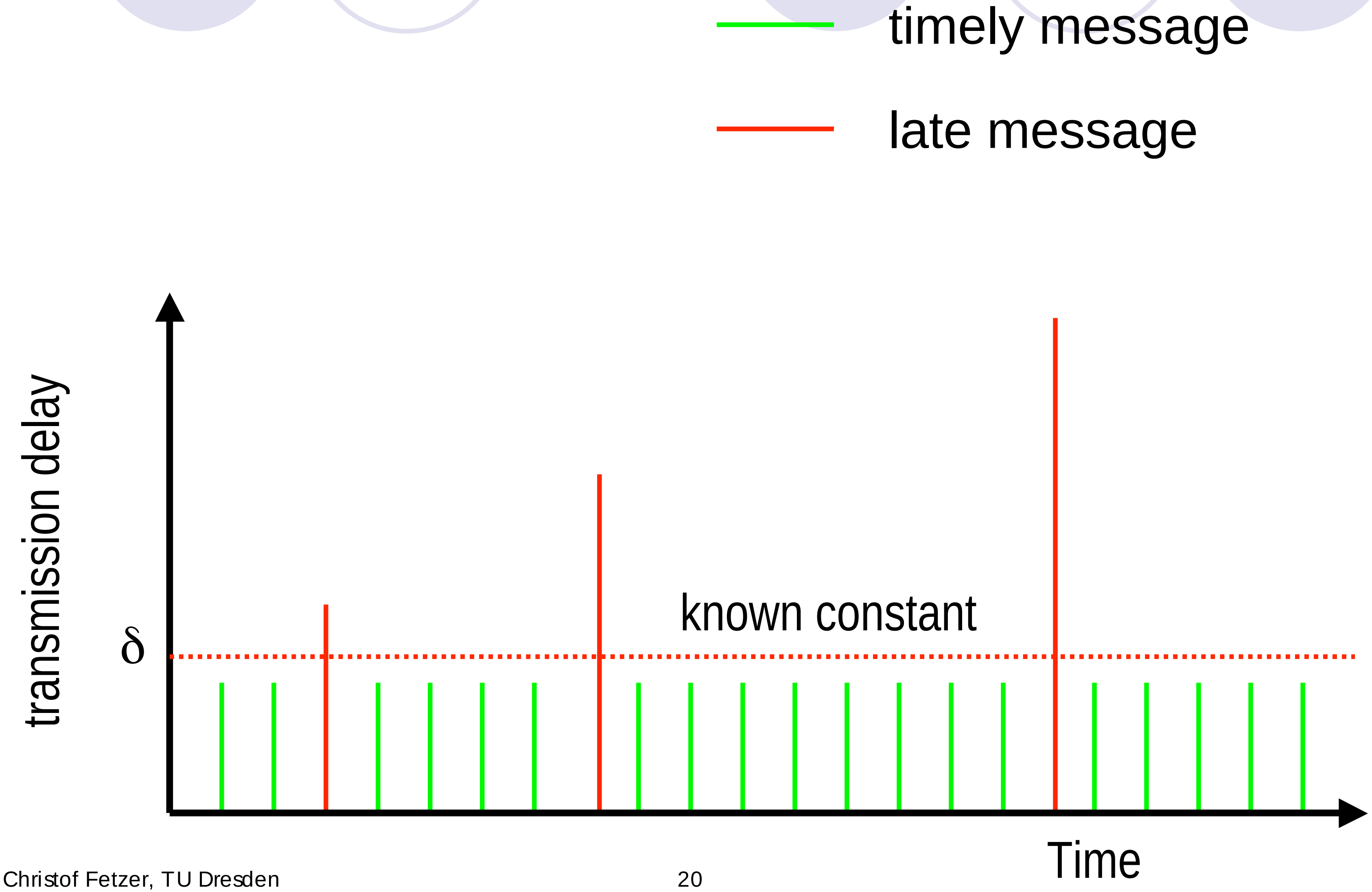
Datagram Failure Semantics Enforcement



Partially Synchronous Systems



Timed Model: No Upper Bound



Conditional Timeliness Requirements

- Timeliness Requirement:
  - have to achieve something good in **D** seconds

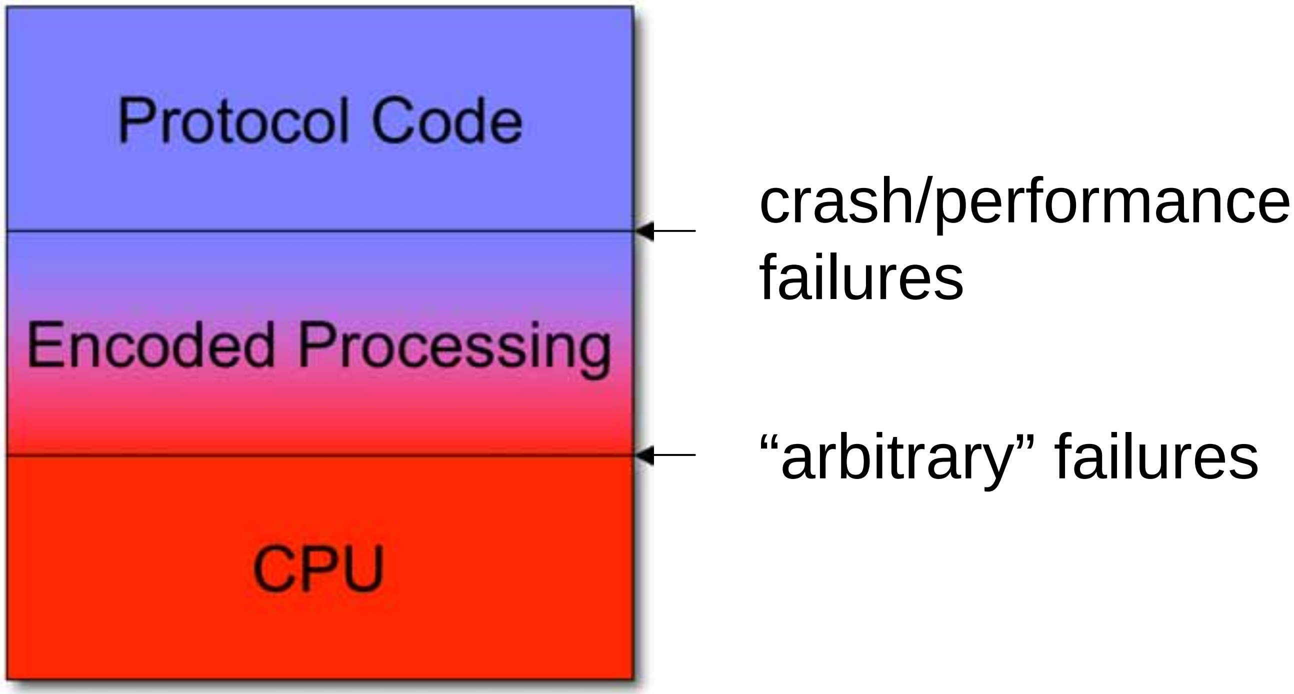
- Conditional Timeliness Requirement:
  - have to achieve something good in **D** seconds if system is *stable*.

Process Service

Process Service

- Failure assumption:
  - **Processes have crash / performance failure semantics**

Process Failure Semantics Enforcement



Possibilities and Impossibilities in the Timed Model

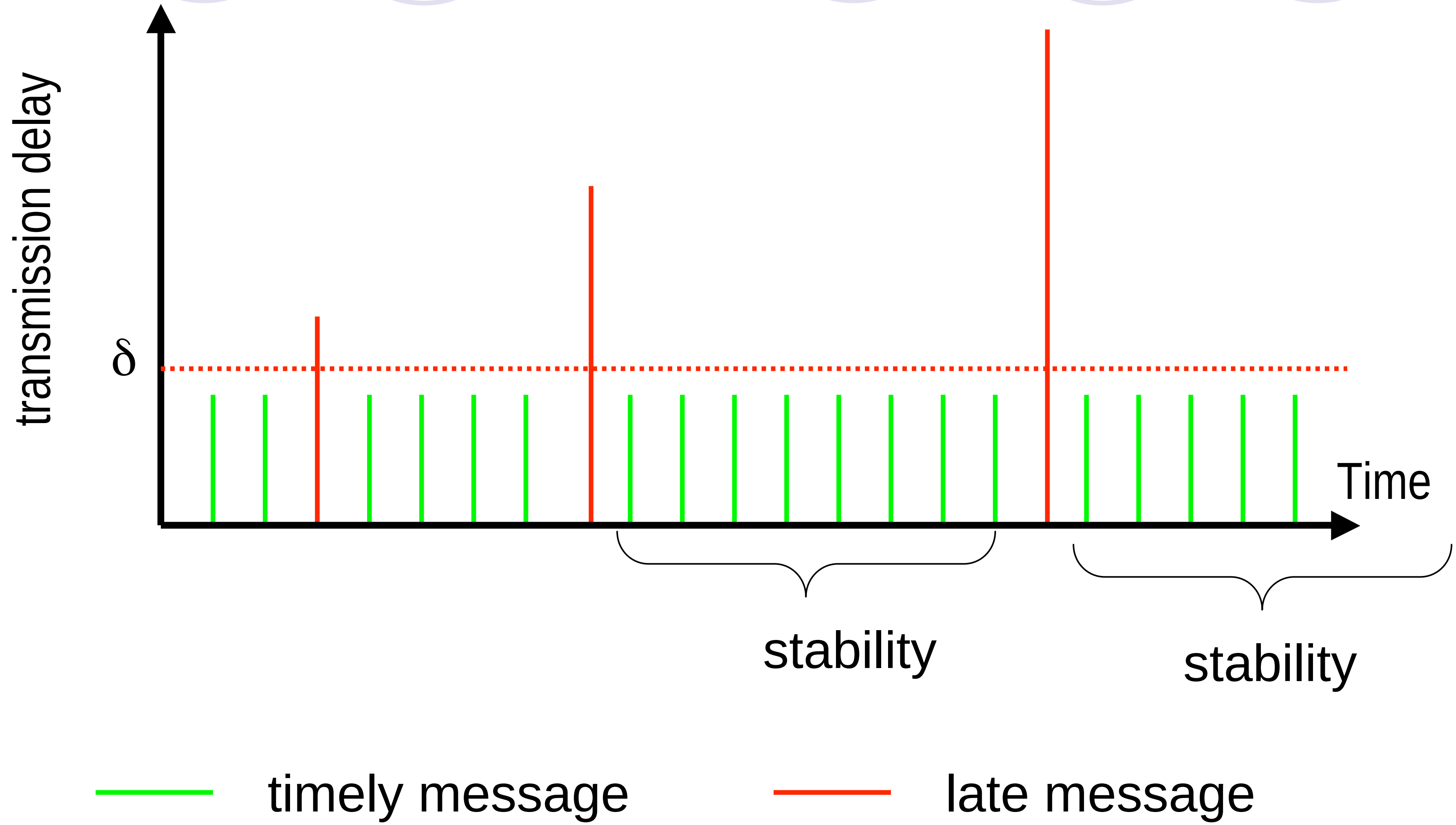
Most Standard Problems are impossible to solvable in TM

- For example, cannot solve
  - *consensus*,
  - *strong leader election*
  - eventually perfect failure detector
  - ...
- Reason:
  - Timed Model permits runs in which no message is delivered!

Two Approaches

- Change the problem:
  - enforce service properties whenever the underlying system is stable (synchronous)
  - if properties might be violated, signal to clients that properties are not guaranteed
    - we call that fail-awareness [3]
- *Add additional assumptions:*
  - *infinitely often the system is **stable***

Stability and instability periods



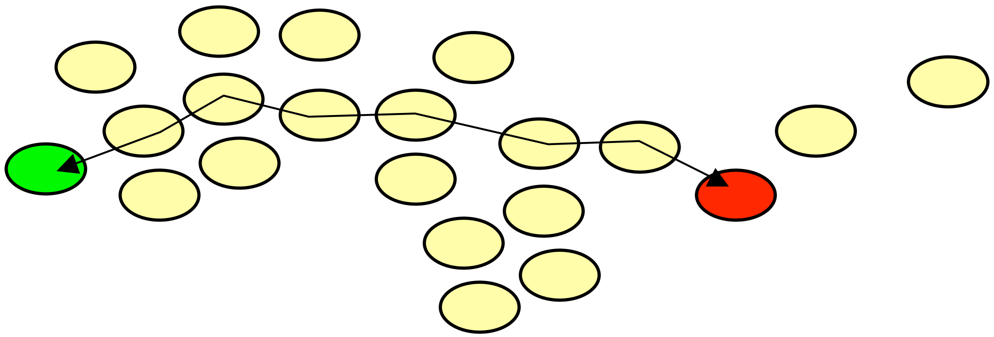
Conditional Timeliness Requirements

- Timeliness Requirement:
  - have to achieve something good in **D** seconds

- Conditional Timeliness Requirement:
  - have to achieve something good in **D** seconds if system is *stable*.

Transmission delay...

- depends on diameter, density, ...
  - expect more variance in mobile/* systems
- How could nodes dynamically adjust δ?



Need to agree on a new δ

- Do we need the system to stabilize?
  - need to adjust δ when the system is unstable
- Do we really need a hardware clock?
  - e.g., change of clock frequency in mobile systems might complicate things...
  - use of minimal assumptions

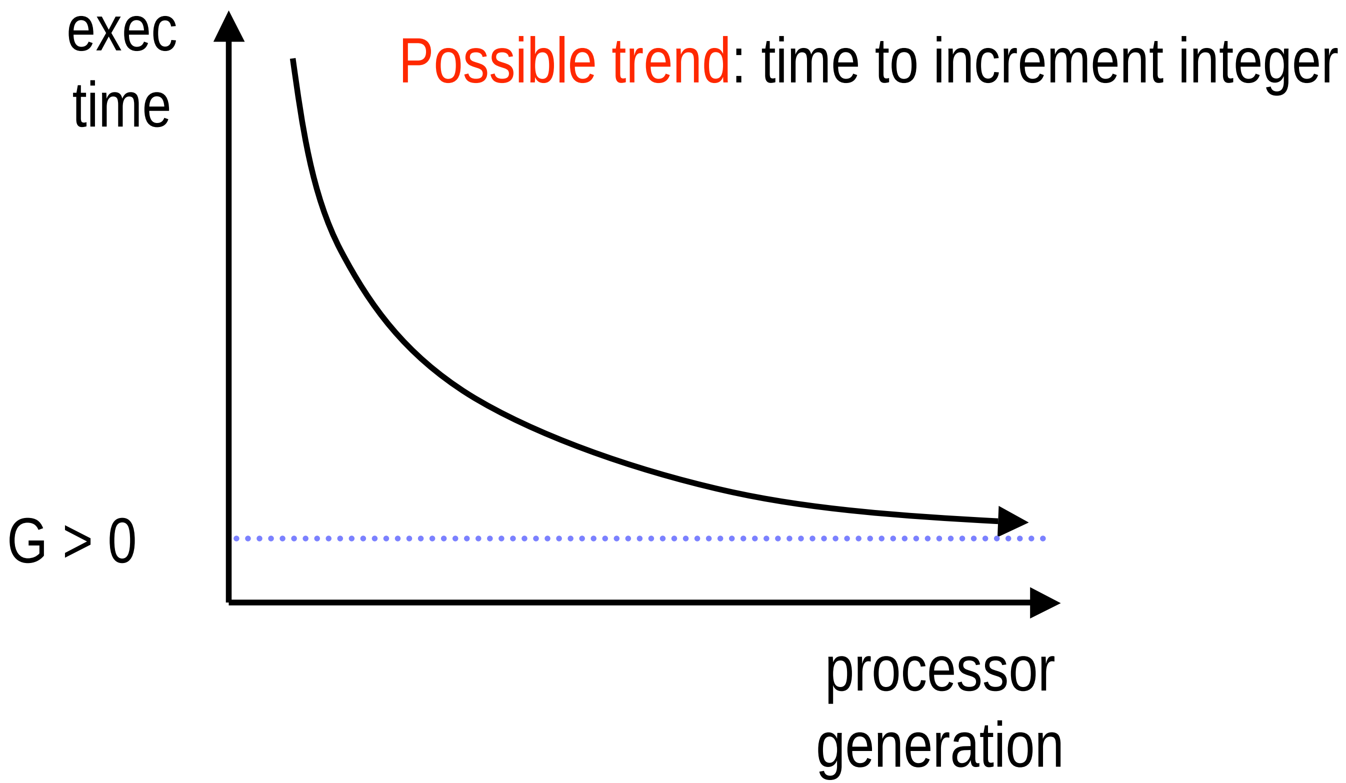
Finite Average Response Time Model (Far) Model

[5]

Observation 1:

Computers are not infinitely fast!

Max. Speed of ++ is bounded



Weak Clock

- Clock with some max. unknown speed:

```
int tick = 0 ;
process Tick() {
    forever { tick++; }
}

int ReadClock() { return tick; }
```

Arbitrary Clock Failures

```
int tick = 0, last = 0; const int maxd = ...;
process Tick() { forever { tick++; } }

int ReadClock() {
    if (H() > tick) {
            tick = min(H(), tick+(tick-last)*maxd);
    } else { tick = max(H(), last); }
    last = ++tick;
    return last;
}
```

Weak Clock Semantics

- For each clock tick, at least some minimum unknown time G has passed
- What is it good for?
  - timeouts!

Observation 2:

In all well engineered systems(*), average transmission delay is finite.

(*) we need to take care of protocols without flow control

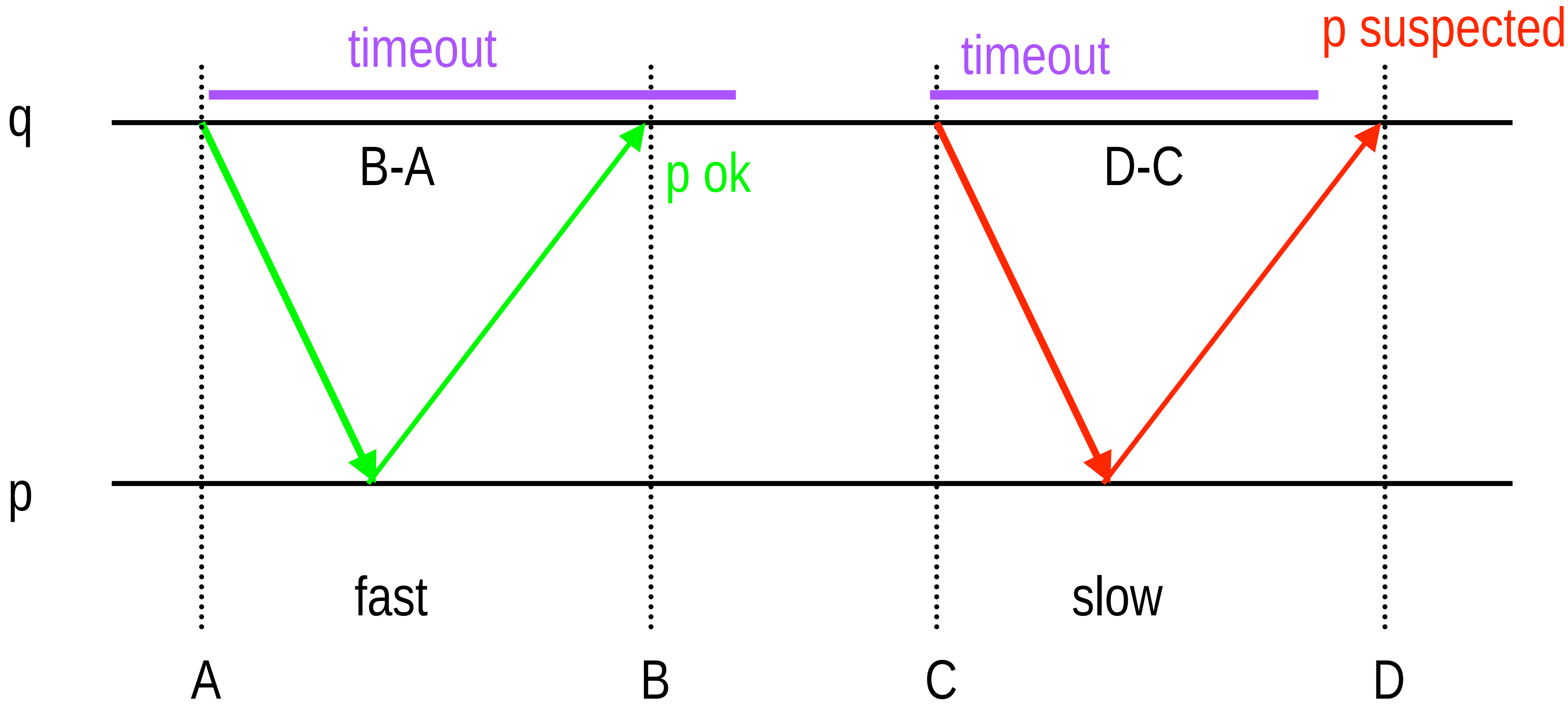
Communication System

- We use **stubborn channels**
  - only reliable transmission of last message is guaranteed
  - need to wait for delivery of last message before transmitting new message

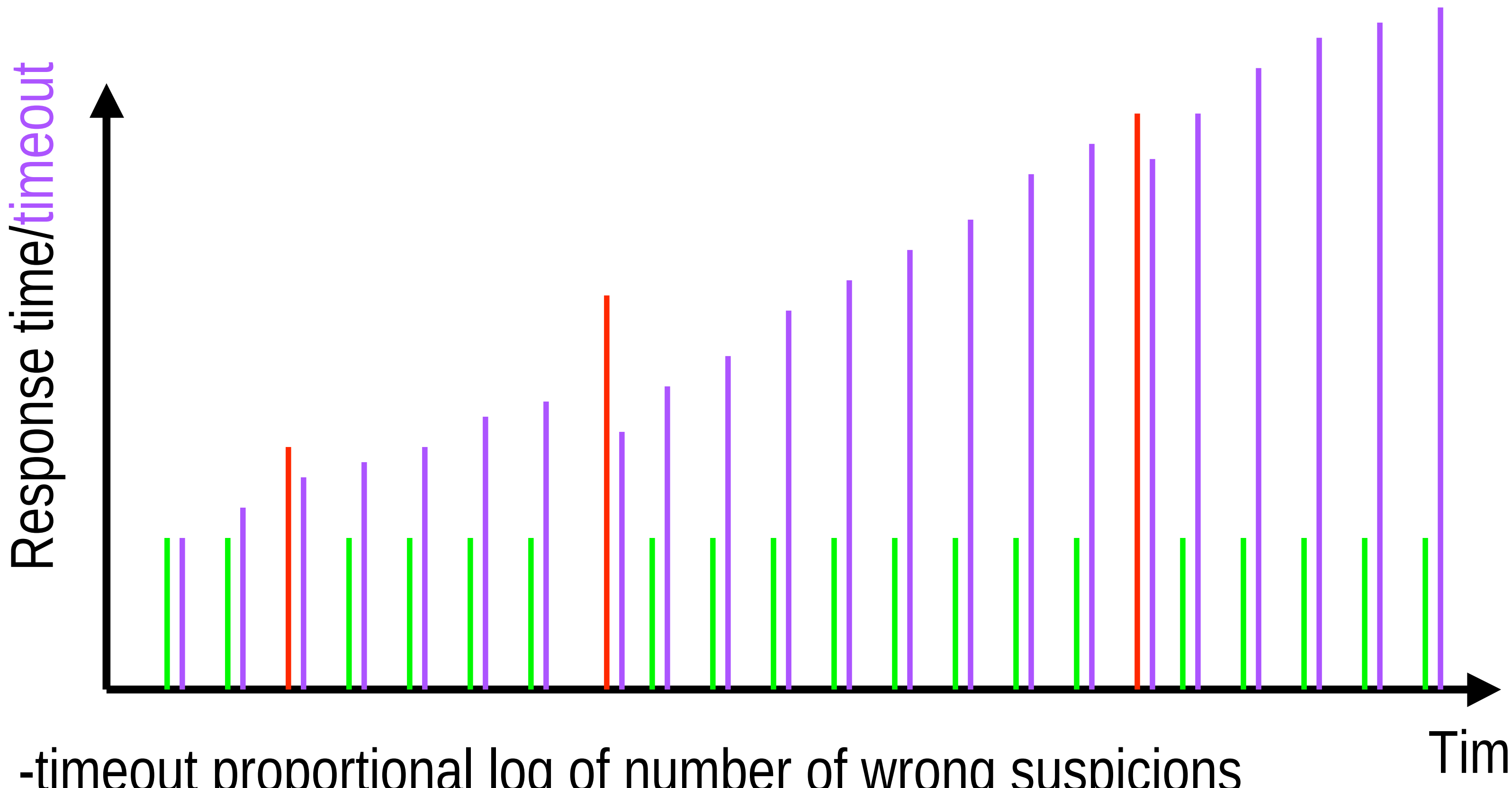
Finite Average Response Time

- **Assumption**:
  - average response time of link between any two correct processes is finite
  - average: $\lim_{k \to \infty}$(average of k first responses)
- Result:
  - Assumptions 1+2 sufficient to implement an eventually perfect failure detector [5]

Eventually Perfect Failure Detector



Timeout Adaptation



-timeout proportional log of number of wrong suspicions

-timeout proportional number fast messages since last slow message

Finite Average Response (FAR) Model [5]

- Eventually perfect failure detector (and hence consensus protocol) can be implemented in a system with
  - NO upper/relative bound on transmission delay
  - NO upper/relative bound on processing delay
  - NO assumption that system stabilizes
  - NO clocks, failure detectors, etc
- But
  - average response time must be finite
  - unknown min exec time for some operation

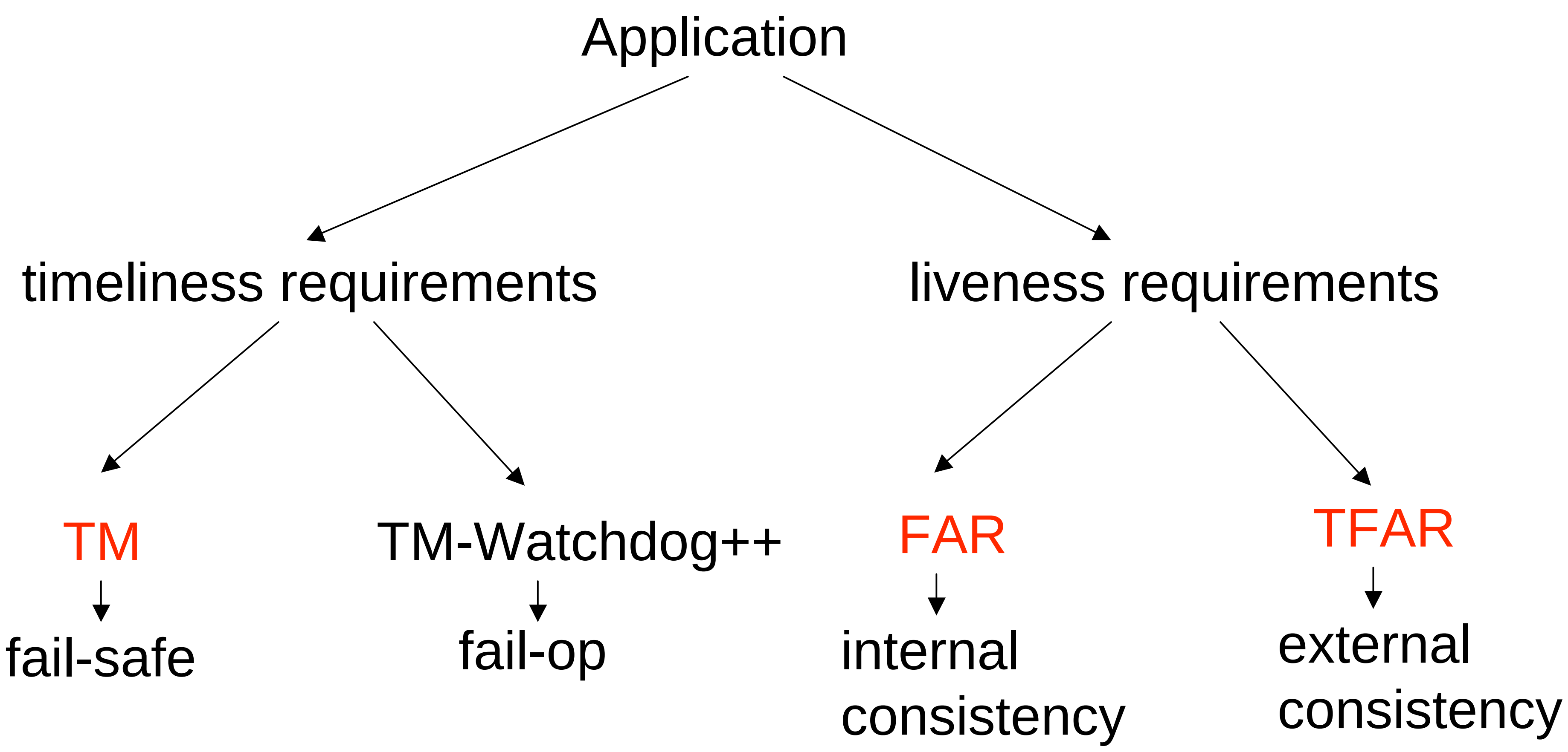
Timed Far Model

FAR Model [6]

Impossibility Result

- *Strong leader election* problem, i.e.,
  - infinitely often there is a leader
  - at any point in time there is at most one leader
- impossible to solve in FAR model [6]
  - adding a clock solves the problem
  - →Timed FAR model

Conclusion


Application
timeliness requirements
liveness requirements
TM
fail-safe
TM-Watchdog++
fail-op
FAR
internal consistency
TFAR
external consistency

References

[1] F. Cristian and C. Fetzer, *The Timed Asynchronous Distributed System Model*, IEEE Transactions on Parallel and Distributed Systems

[2] C. Fetzer, F. Cristian, *Building Fault-Tolerant Hardware Clocks*, DCCA1999

[3] C. Fetzer and F. Cristian, *Fail-Awareness: An Approach to Construct Fail-Safe Applications*, Journal of Real-Time Systems, 2003

[4] C. Fetzer, F. Cristian, *A Fail-Aware Datagram Service*, IEE Proceedings - Software Engineering, 1999

[5] C. Fetzer, U. Schmid, M. Süßkraut, *On the Possibility of Consensus in Asynchronous Systems with Finite Average Response Times*, ICDCS 2005

[6] M. Süßkraut, C. Fetzer, *Leader Election in the Timed Finite Average Response Time Model*