Distributed Algorithms

(Part 2)

RiSE Winter School 2012

Ulrich Schmid

Institute of Computer Engineering, TU Vienna

Embedded Computing Systems Group E182/2

s@ecs.tuwien.ac.at

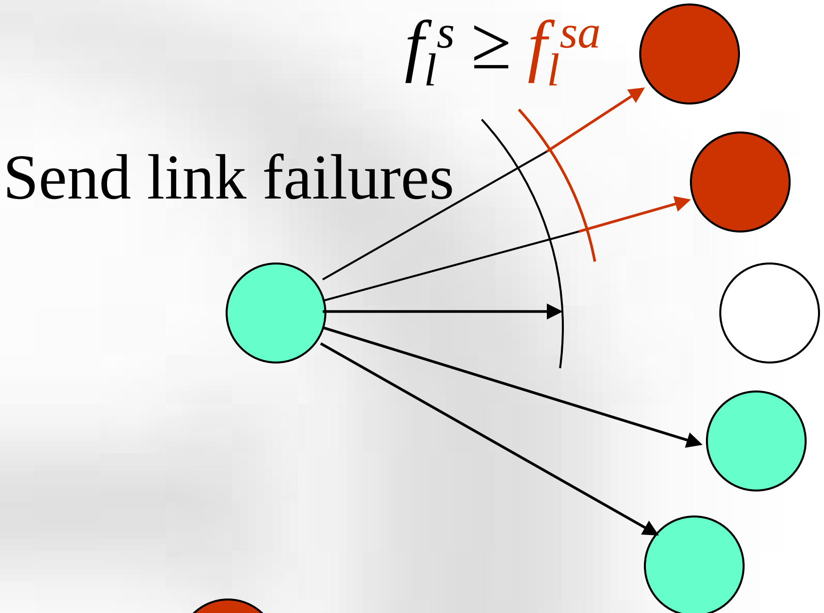
FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

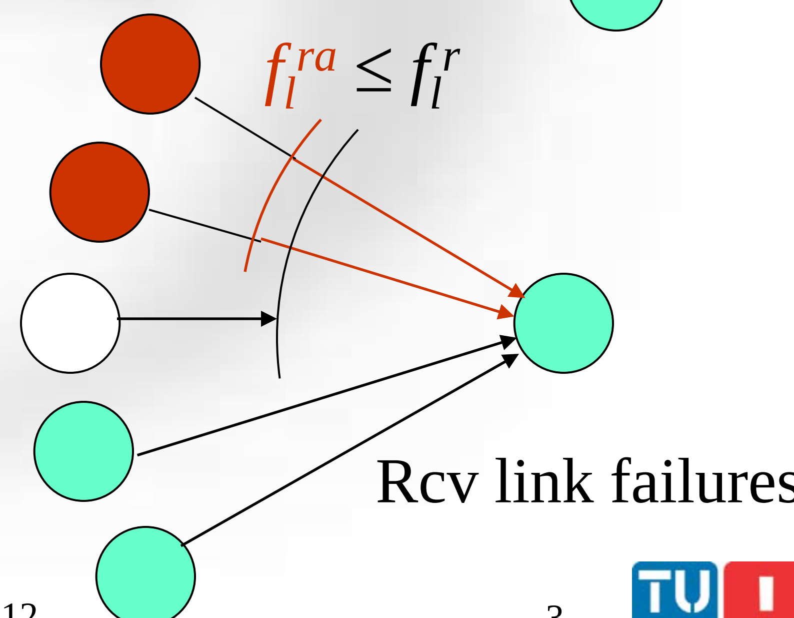
Food for Thoughts

Communcation Failures

- Link failure model:
  1. Distinguish send and receive link failures
  2. Distinguish omission and arbitrary link failures
  3. Indep. for every send/rec to/from all



- Known results:
  - $n > f_l^r + f_l^s$ necessary & sufficient for solving consensus with pure link omission failures
  - $n > f_l^r + f_l^{ra} + f_l^s + f_l^{sa}$ necessary & sufficient for solving consensus with link omission and arbitrary failures



Exercises

1. Find the smallest values for S,R,S',R',S'',R'' in the CB implem. below for arbitrary link failures ($f_l^r = f_l^{ra}$ and $f_l^s = f_l^{sa}$):

if got ($init,p_s,m_s$) from p_s
  $\rightarrow$ send ($echo,p_s,m_s$) to all [once]
if got ($echo,p_s,m_s$) from $Sf_l^{sa} + Rf_l^{ra} + f + 1$
  $\rightarrow$ send ($echo,p_s,m_s$) to all [once]
if got ($echo,p_s,m_s$) from $S'f_l^{sa}+R'f_l^{ra}+2f + 1$
  $\rightarrow$ call **accept**(p_s,m_s)

Required number of procs:

- $n \geq S''f_l^{sa}+R''f_l^{ra}+3f + 1$

Link failure lower bound:

- $n \geq f_l^r + f_l^{ra} + f_l^s + f_l^{sa}$

2. Find an „easy impossibility proof“ that shows that n=4 processors are not enough for solving consensus with $f_l^r = f_l^{ra} = f_l^s = f_l^{sa} = 1$ (and f =0)

Solution

Consistent BC with Comm. Failures (I)

Ulrich Schmid and Christof Fetzer. Randomized asynchronous consensus with imperfect communications. Technical Report 183/1-120, Department of Automation, Technische Universität Wien, January 2002. (Appeared in Proc. SRDS‘02).
http://wwwold.ecs.tuwien.ac.at/W2F/papers/SF02_byzTR.ps

if got $(init, p_s, m_s)$ from p_s

$\rightarrow$ send $(echo, p_s, m_s)$ to all [once]

if got $(echo, p_s, m_s)$ from $f_l^{sa} + f_l^{ra} + f + 1$

$\rightarrow$ send $(echo, p_s, m_s)$ to all [once]

if got $(echo, p_s, m_s)$ from $f_l^{sa} + 3f_l^{ra} + 2f + 1$

$\rightarrow$ call **accept**(p_s, m_s)

Fig. 2

Required number of procs:

- $n \geq 2f_l^{sa} + 4f_l^{ra} + 3f + 1$ (Thm. 2)

Link failure lower bound:

- $n \geq f_l^{r} + f_l^{ra} + f_l^{s} + f_l^{sa}$

Consistent BC with Comm. Failures (II)

The following follows right from the failure model (Lemma 1):

(1) Every correct processor p_i may receive at most $f_l^{ra} + f$ faulty echo msgs

(2) At most f_l^{sa} (correct) processors can emit echo, due to send link failures of p_s

(3) At most $2f_l^{ra} + f$ messages received by a correct processor by time t may not have been received at any other correct processor by time $t + \varepsilon$

Unforgeability:

- For a contradiction, suppose p_i calls **accept** by t+2d, so must have got $f_l^{sa}+3f_l^{ra}+2f+1$ echo msgs
- At most $f_l^{sa} + f_l^{ra} + f$ of these could originate from (1) and (2) → at least one (in fact, more) correct p_j must have emitted echo for good. This happens either
  - if p_j properly received init – which cannot happen as p_s did not call **bcast** by t
  - if p_j received $f_l^{sa} + f_l^{ra} + f + 1$ echo msgs itself, in which case we can repeat the argument above for p_j

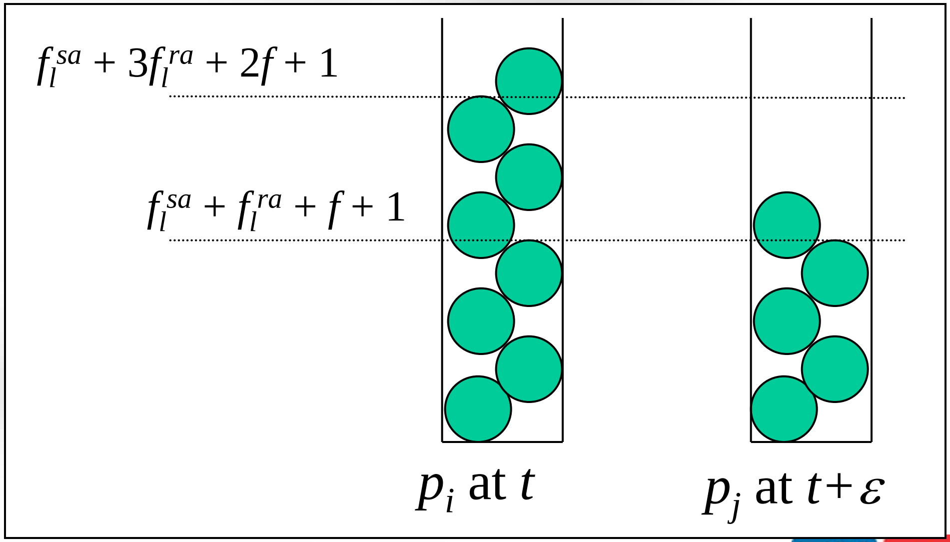
Consistent BC with Comm. Failures (III)

Correctness:

- Since p_s is correct, at least $n - f_l^{sa} - f \geq f_l^{sa} + 4f_l^{ra} + 2f + 1$ will get the init msg and emit echo by time $t+d+\varepsilon$
- Since every receiver could miss at most f_l^{ra} of these messages, at least $f_l^{s} + 3f_l^{ra} + 2f + 1$ are received by time $t+2(d+\varepsilon)$ → **accept** is triggered

Relay:

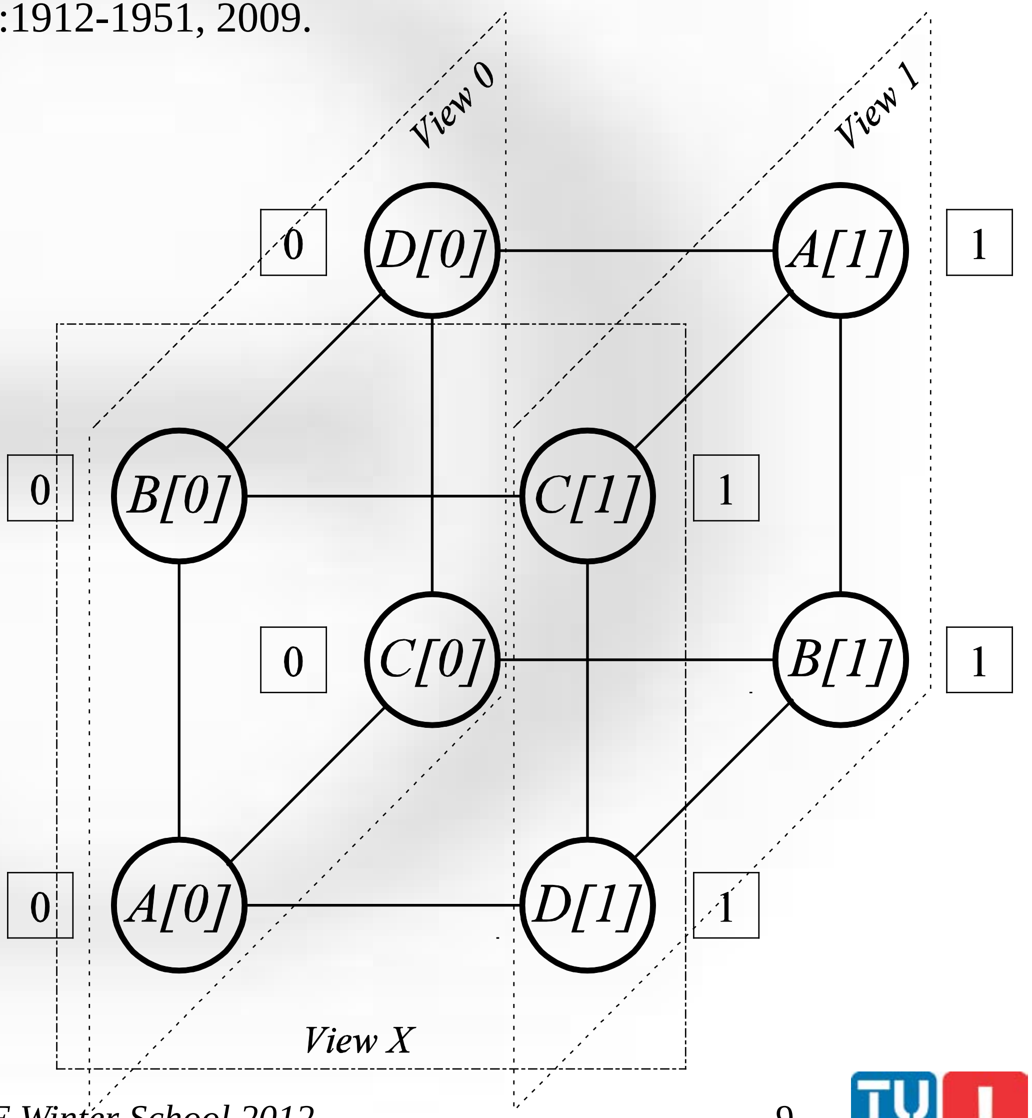
- By (3), at most $2f_l^{ra} + f$ of the echo messages available at p_i by t could be missing at p_j by $t + \varepsilon$
- $f_l^{sa} + f_l^{ra} + f + 1$ remain, so trigger emitting echo at p_j → continue as in original Relay-proof



Easy Impossibility Proof

Ulrich Schmid, Bettina Weiss, and Idit Keidar. Impossibility results and lower bounds for consensus under link failures. *SIAM Journal on Computing*, 38(5):1912-1951, 2009.

- Suppose correct algorithm $\mathcal{A}$ = (A,B,C,D) for (p_0,p_1,p_2,p_3) existed
- Consider this cube:
  - In View 0: Decision is 0 by Validity
  - In View 1: Decision is 1 by Validity
  - In View X: Violation of Agreement

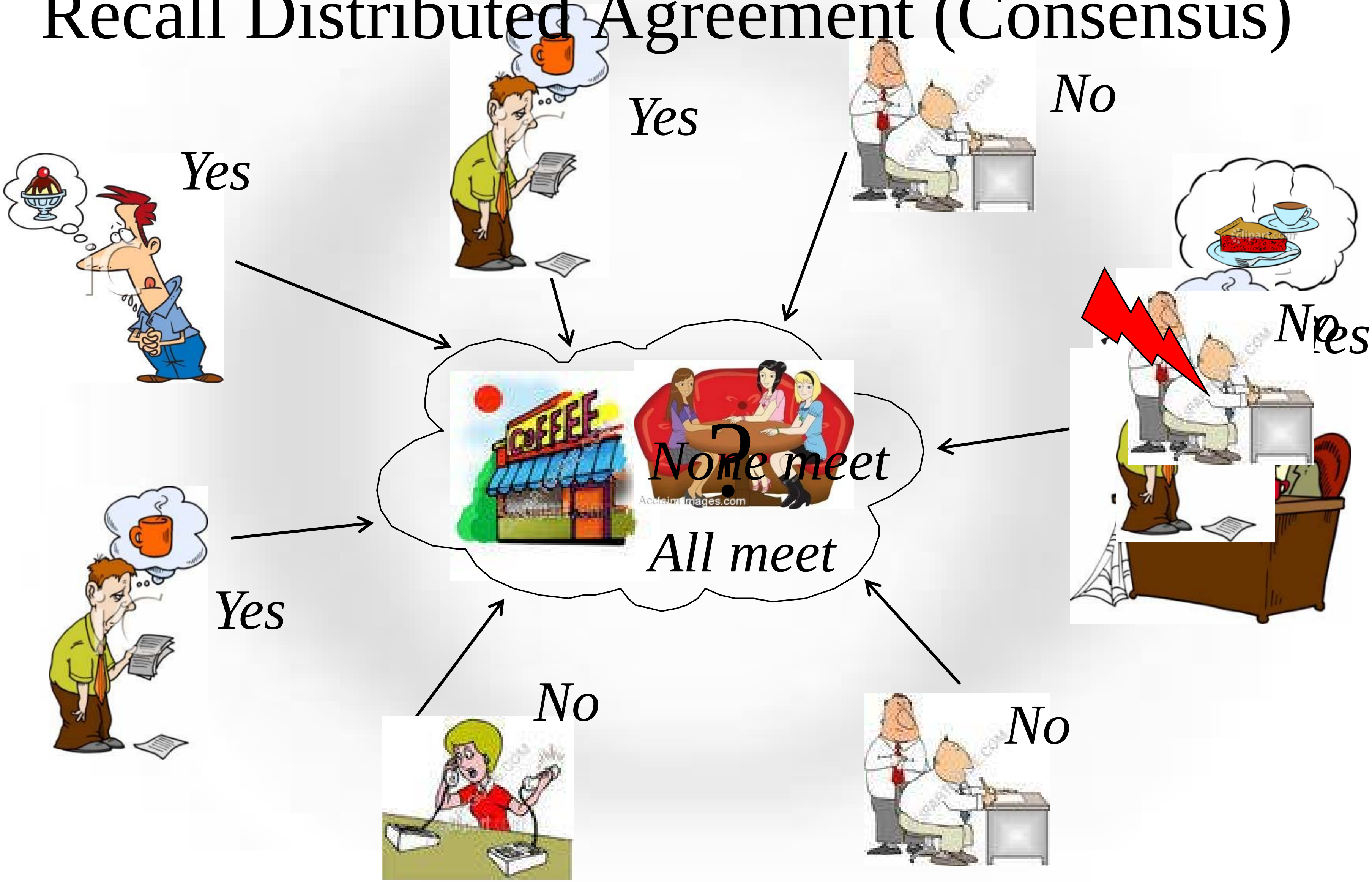


Content (Part 2)

- The Role of Synchrony Conditions:
  - Failure Detectors
  - Real-Time Clocks
- Partially Synchronous Models
  - Models supporting lock-step round simulations
  - Weaker partially synchronous models
- Distributed Real-Time Systems

The Role of Synchrony Conditions

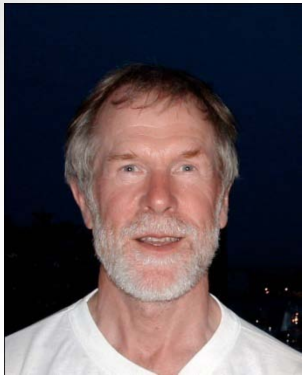
Recall Distributed Agreement (Consensus)



Consensus Impossibility (FLP)

Fischer, Lynch und Paterson [FLP85]:

"There is no deterministic algorithm for solving consensus in an asynchronous distributed system in the presence of a single crash failure."



Key problem:

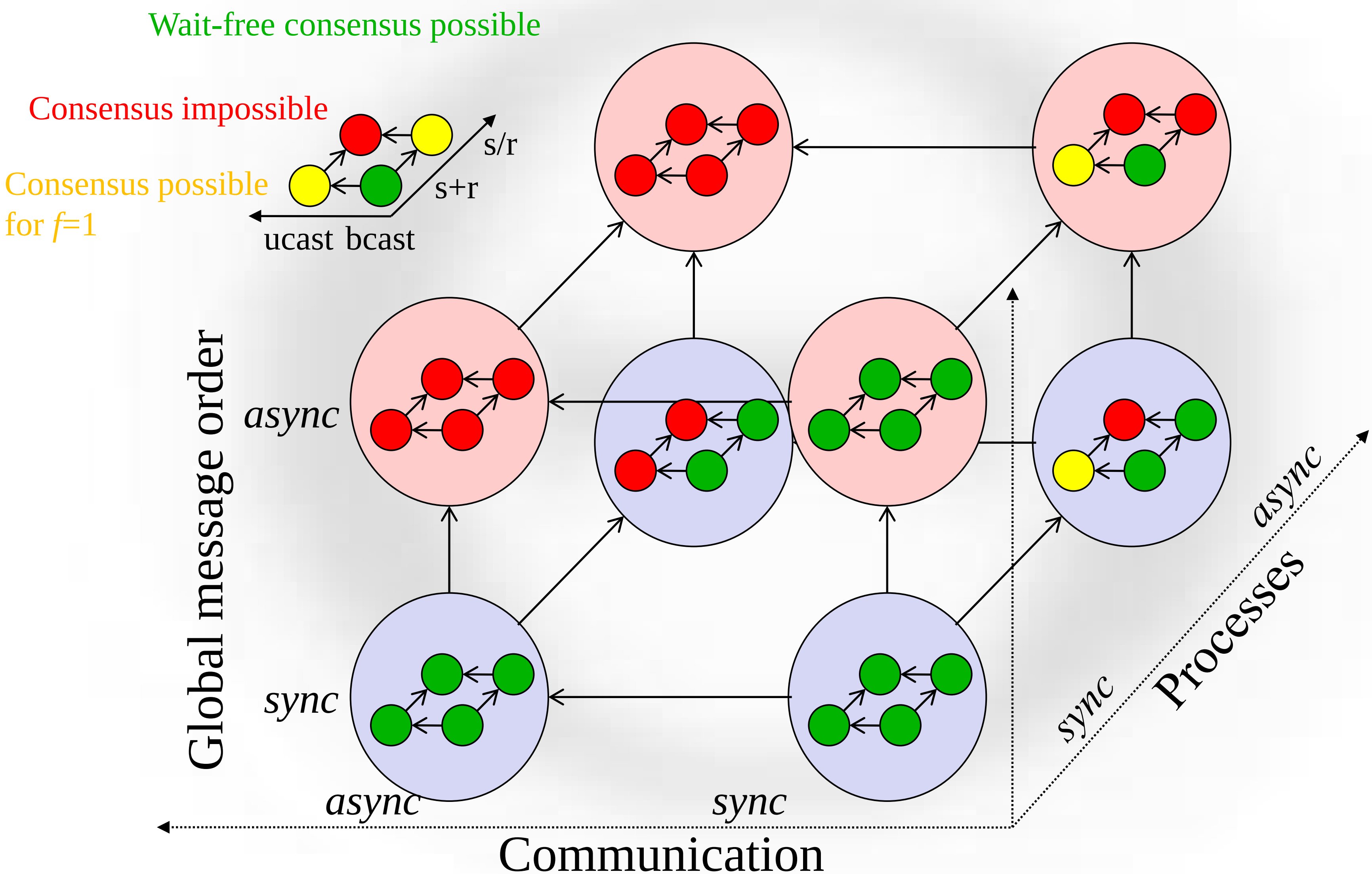
Distinguish slow from dead!

Consensus Solvability in ParSync [DDS87] (I)

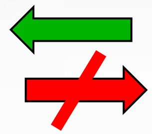
Dolev, Dwork and Stockmeyer investigated consensus solvability in Partially Synchronous Systems (ParSync), varying 5 **„synchrony handles“** :

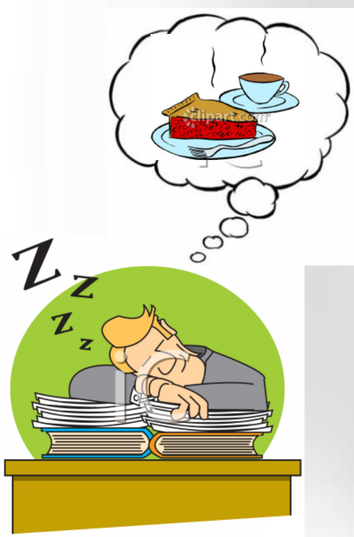
- Processors synchronous / asynchronous
- Communication synchronous / asynchronous
- Message order synchronous (system-wide consistent) / asynchronous (out-of-order)
- Send steps broadcast / unicast
- Computing steps atomic rec+send / separate rec, send

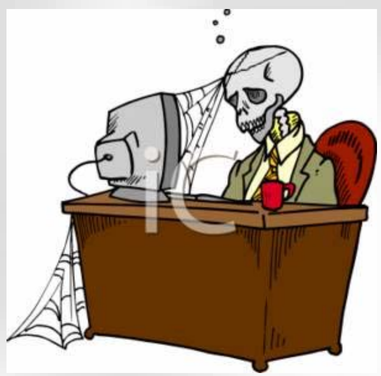
Consensus Solvability in ParSync [DDS87] (II)

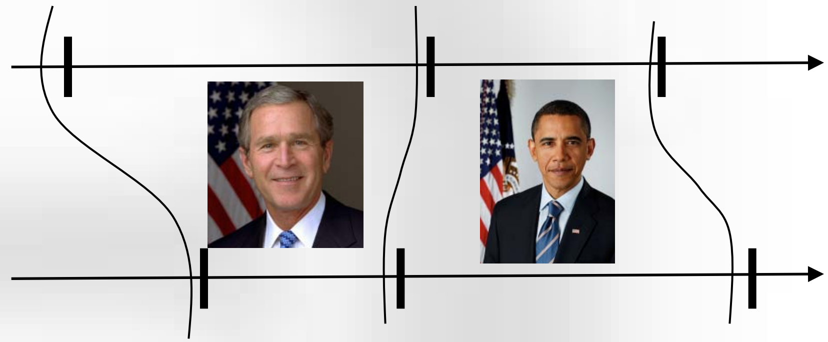


The Role of Synchrony Conditions

Enable failure detection  **Enforce event ordering**







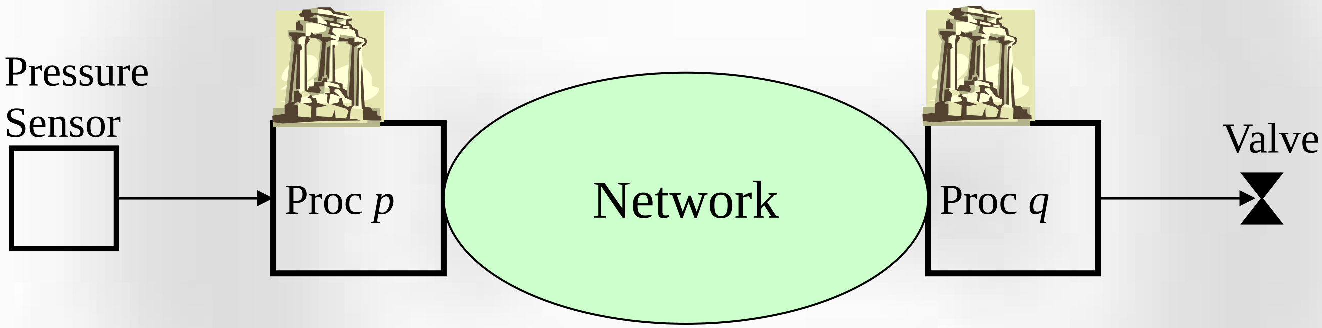
- Distinguish slow from dead

- Distinguish „old“ from „new“
- Ruling out existence of stale (in-transit) information
- Creating non-overlapping „phases of operation“ (rounds)

Failure Detectors

Failure Detectors [CT96] (I)

- Chandra & Toueg augmented purley asynchronous systems with (unreliable) failure detectors (FDs):



- Every processor owns a local FD module (an „**oracle**“ – we do not care about how it is implemented!)
- In every step [of a purely asynchronous algorithm], the FD can be queried for a hint about failures of other procs

Failure Detectors [CT96] (II)

-  make mistakes – the (time-free!) FD specification restricts the allowed mistakes of a FD
- FD hierarchy: A stronger FD specification implies
  - less allowed mistakes
  - more difficult problems to be solved using this FD
  - But: FD implementation more demanding/difficult
- Every problem *Pr* has a weakest FD *W:*
  - There is a purely asynchronous algorithm for solving *Pr* that uses *W*
  - Every FD that also allows to solve *Pr* can be transformed (via a purely asynchronous algorithm) to simulate *W*

Example Failure Detectors (I)

- Perfect failure detector P: Outputs suspect list
  - *Strong completeness*: Eventually, every process that crashes is permanently suspected by every correct process
  - *Strong accuracy*: No process is ever suspected bevor it crashes

- Eventually perfect failure detector ◊P:
  - *Strong completeness*
  - *Eventual strong accuracy*: There is a time after which correct processes are never suspected by correct processes

Example Failure Detectors (II)

- Eventually strong failure detector ◊S:
  - *Strong completeness*
  - *Eventual weak accuracy*: There is a time after which some correct process is never suspected by correct processes
- Leader oracle Ω: Outputs a single process ID
  - There is a time after which every not yet crashed process outputs the same correct process *p* (the „leader“)
- Both are weakest failure detectors for consensus (with majority of correct processes)

Consensus with ◊S: Rotating Coordinator

Task $T1$:

(1) $r_i \leftarrow 0$; $est_i \leftarrow v_i$;
(2) **while** $true$ **do**
(3) $c \leftarrow (r_i \bmod n) + 1$; $r_i \leftarrow r_i + 1$; % $1 \le r_i < +\infty$ %

———— Phase 1 of round r: from p_c to all ————

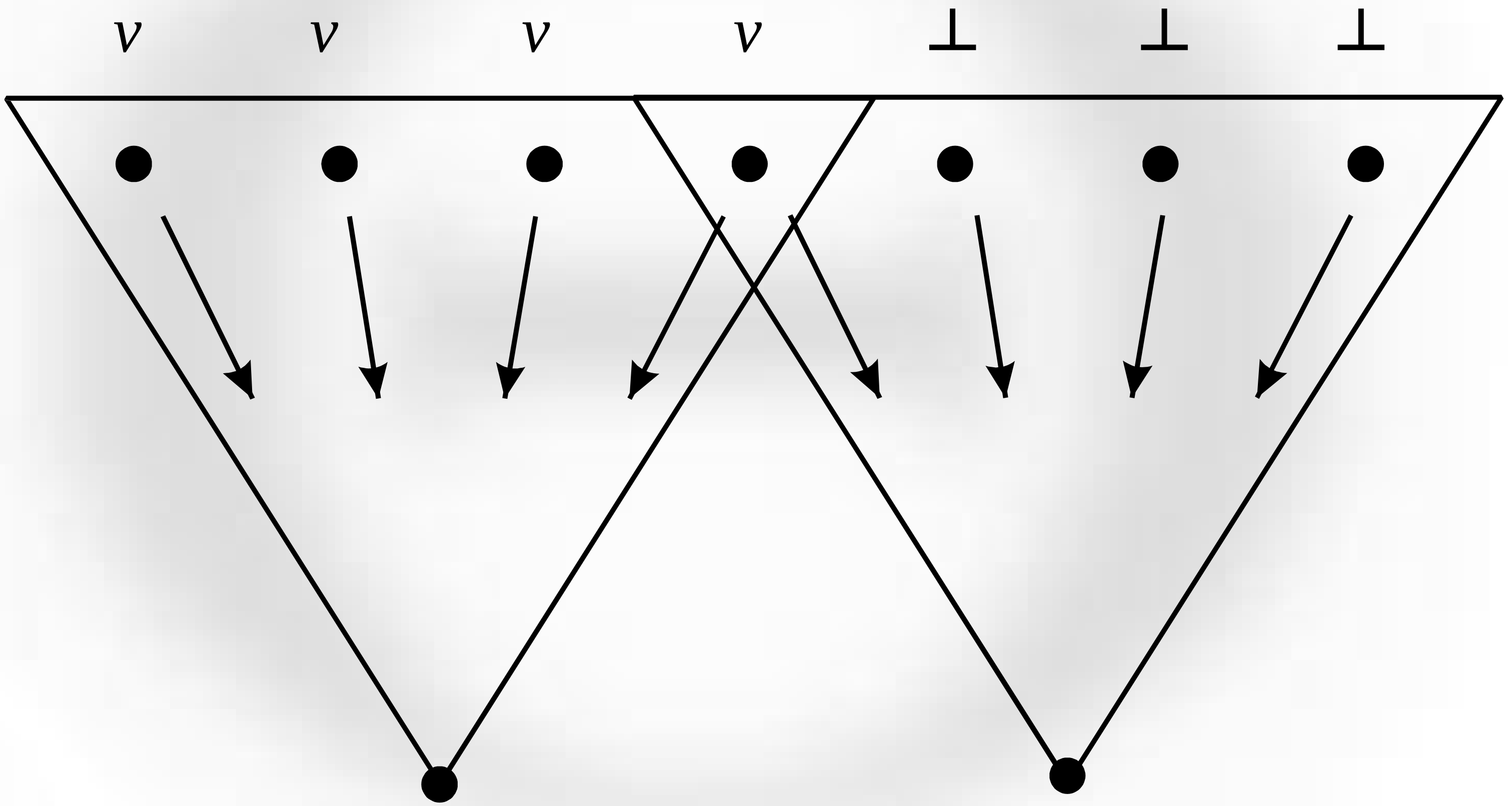
(4) **if** $(i = c)$ **then** *broadcast* PHASE1(r_i, est_i) **endif**;
(5) **wait until** (PHASE1(r_i, v) has been received from $p_c \vee c \in suspected_i$);
(6) **if** (PHASE1(r_i, v) received from p_c) **then** $aux_i \leftarrow v$ **else** $aux_i \leftarrow \perp$ **endif**;

———— Phase 2 of round r: from all to all ————

(7) *broadcast* PHASE2(r_i, aux_i);
(8) **wait until** (PHASE2 (r_i, aux) msgs have been received from a majority of proc.);
(9) **let** rec_i be the set of values received by p_i at line 8;
% We have $rec_i = \{v\}$, or $rec_i = \{v, \perp\}$, or $rec_i = \{\perp\}$ where $v = est_c$ %
(10) **case** $rec_i = \{v\}$ **then** $est_i \leftarrow v$; *broadcast* DECISION(est_i); **stop** $T1$
(11) $rec_i = \{v, \perp\}$ **then** $est_i \leftarrow v$
(12) $rec_i = \{\perp\}$ **then skip**
(13) **endcase**
(14) **endwhile**

Why Agreement? Intersecting Quorums

Intersecting Quorums:



n=7

f=3

p decides *v*

every *q* changes its estimate to *v*

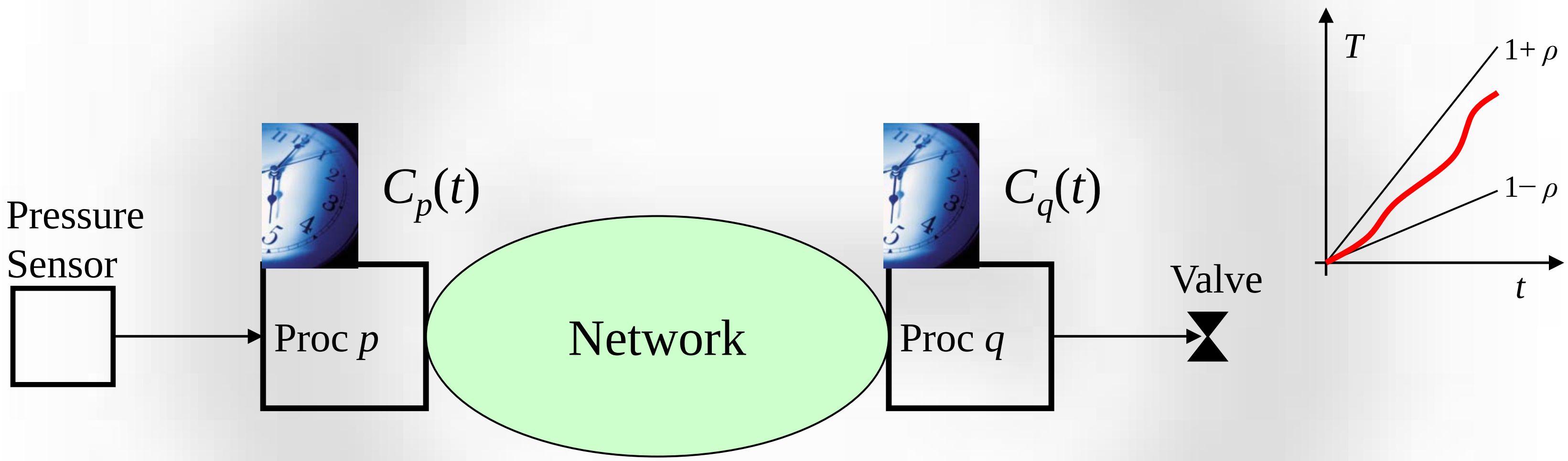
Implementability of FDs

- If we can implement a FD like Ω or ◊S, we can also implement consensus (for $n > 2f$)
- In a purely asynchronous system
  - it is impossible to solve consensus (FLP result)
  - it is hence also impossible to implement Ω or ◊S
- Back at key question: What needs to be added to an asynchronous system to make Ω or ◊S implementable?
  - Real-time constraints [ADFT04, …]
  - Order constraints [MMR03, …]
  - ???

Real-Time Clocks

Distributed Systems with RT Clocks

- Equip every processor p with a local RT clock $C_p(t)$



- Small clock drift ρ → local clocks progress approximately as real-time, with clock rate $\in [1-\rho, 1+\rho]$
- End-to-end delay bounds $[\tau^-, \tau^+]$, *a priori* known

The Role of Real-Time

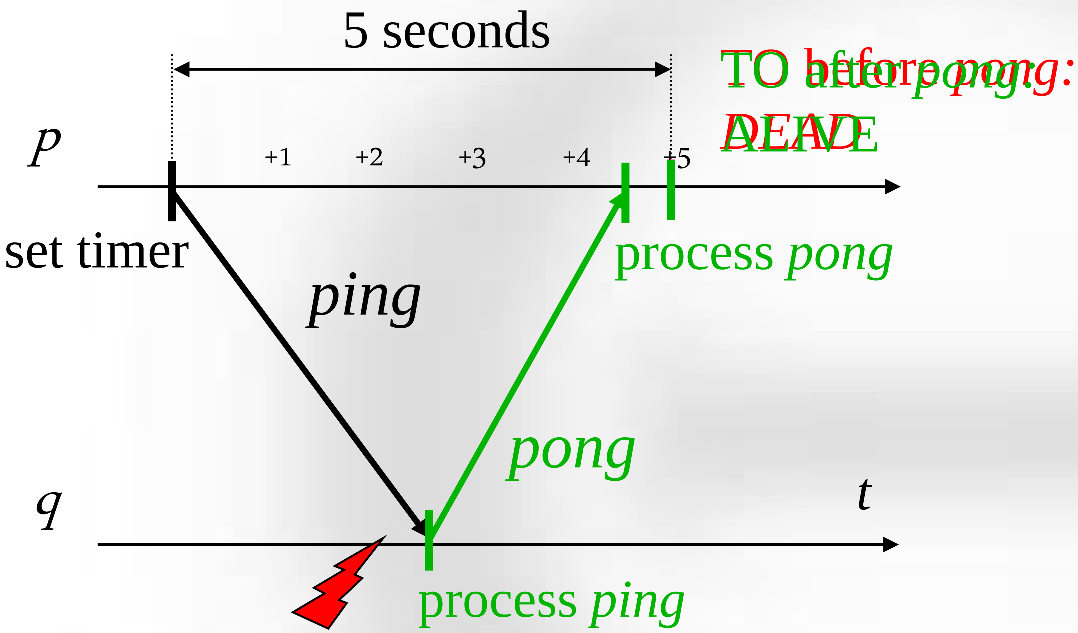
- Real-time clocks enable both:

Failure detection

Event ordering

- [Show later: Real-time clocks are not the only way …]

Failure Detection: Timeout using RT Clock



status = do_roundtrip(q)
{ send *ping* to q
TO := $C_p(t)$ + 5 seconds
wait until $C_p(t)$ = TO
if *pong* did not arrive **then**
return DEAD
else
return ALIVE
}

p can reliably detect whether q has been alive recently, if

- the end-to-end delays are at most $\tau^+ = 2.5$ seconds
- τ^+ is known a priori [at coding time]

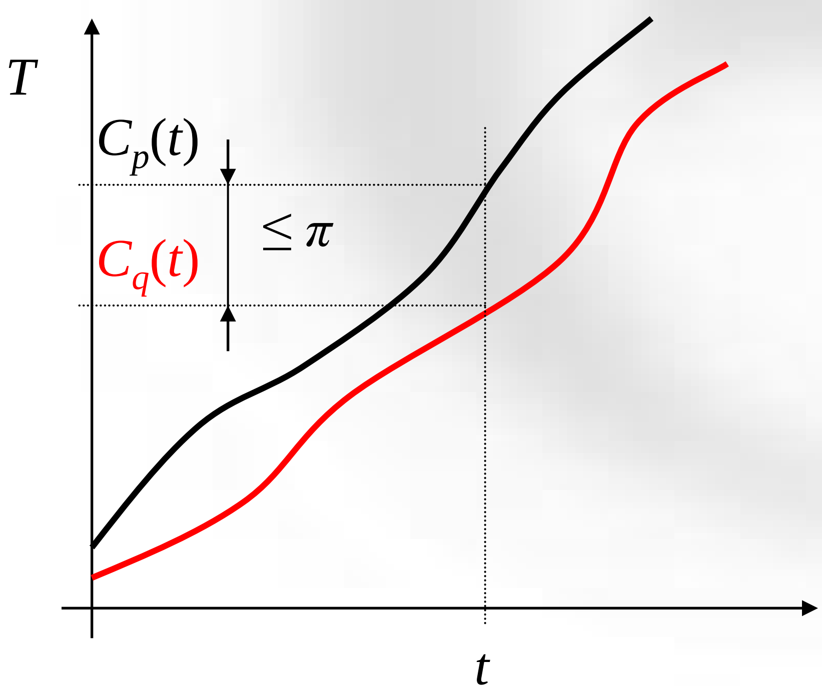
Event Ordering: Via Clock Synchronization

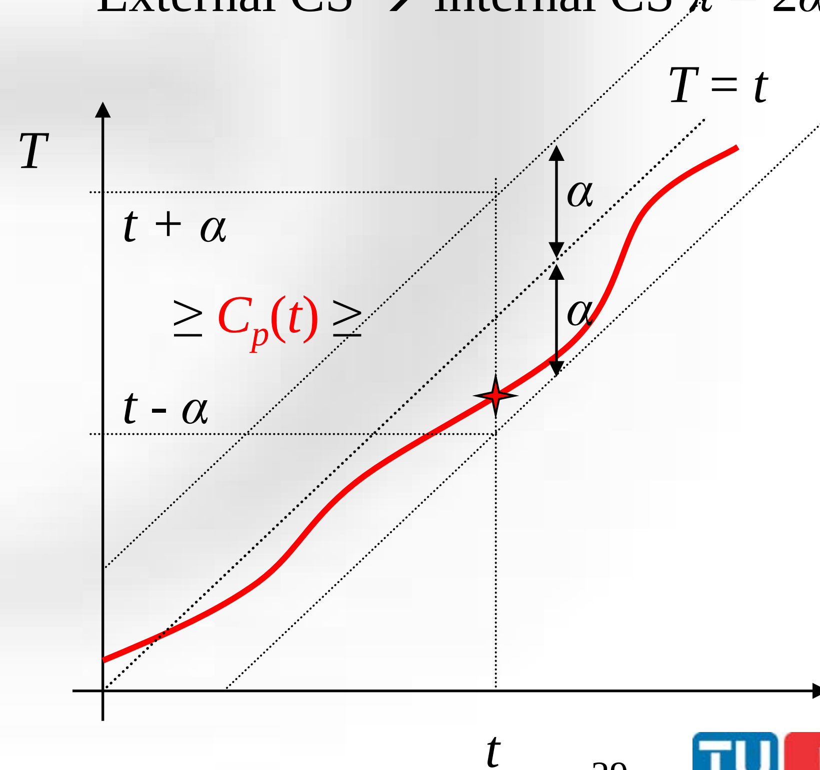
Internal CS:

- Precision $|C_p(t) - C_q(t)| \le \pi$
- Progress like RT (small drift ρ)
- CS-Alg must periodically resynchronize

External CS:

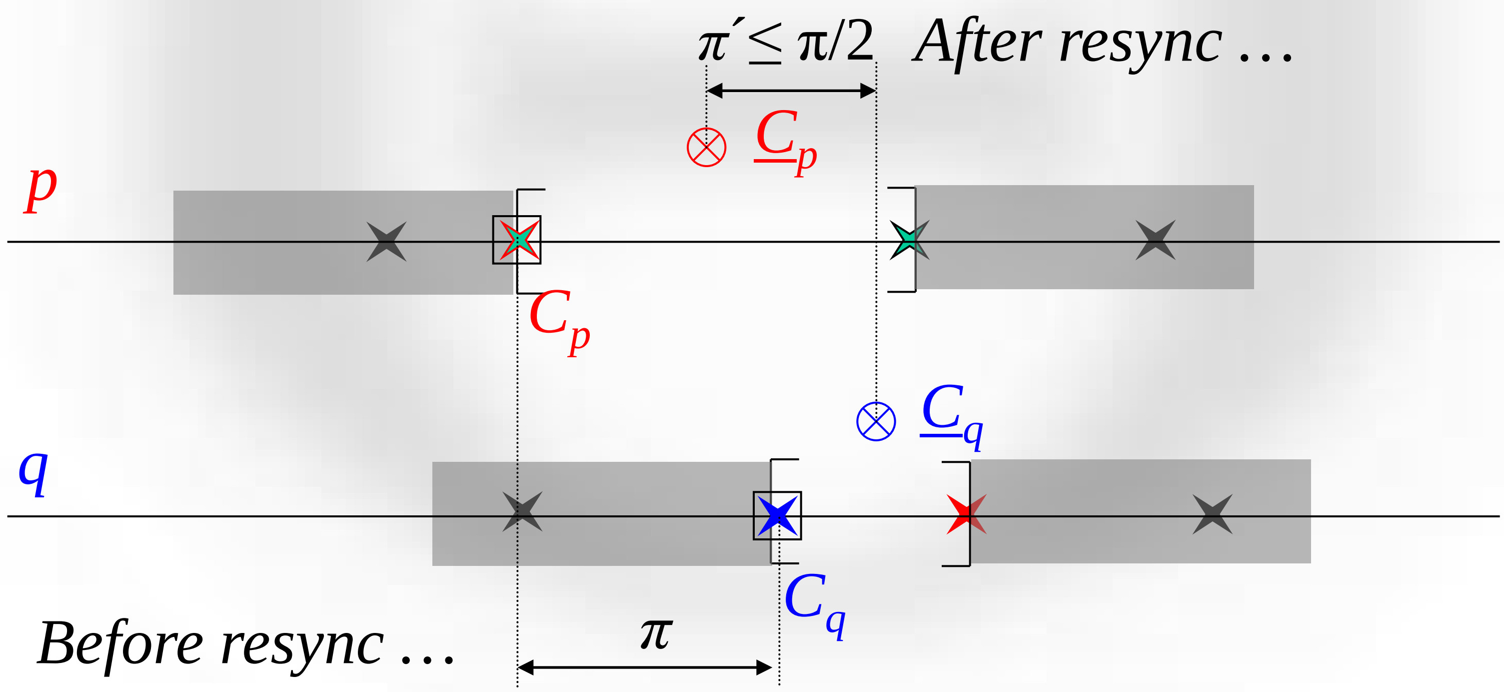
- Accuracy $|C_p(t) - t| \le \alpha$
- CS-Alg needs access to RT
- External CS → internal CS $\pi = 2\alpha$



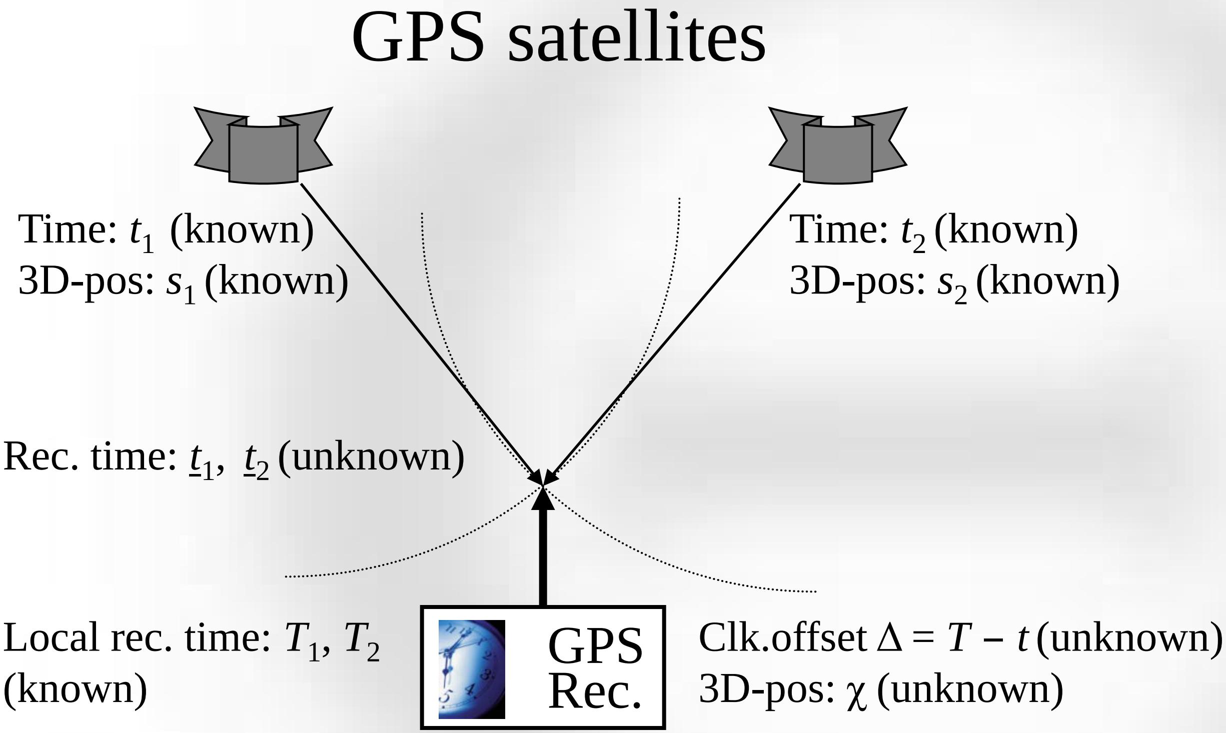


FT Midpoint Internal CS-Alg [LWL88]

- A priori bounded $[\tau^-, \tau^+]$ allows to estimate all remote clocks
- Discard f largest and f smallest clock readings (could be faulty)
- Set local clock to midpoint of remaining interval



Global Positioning System (GPS)



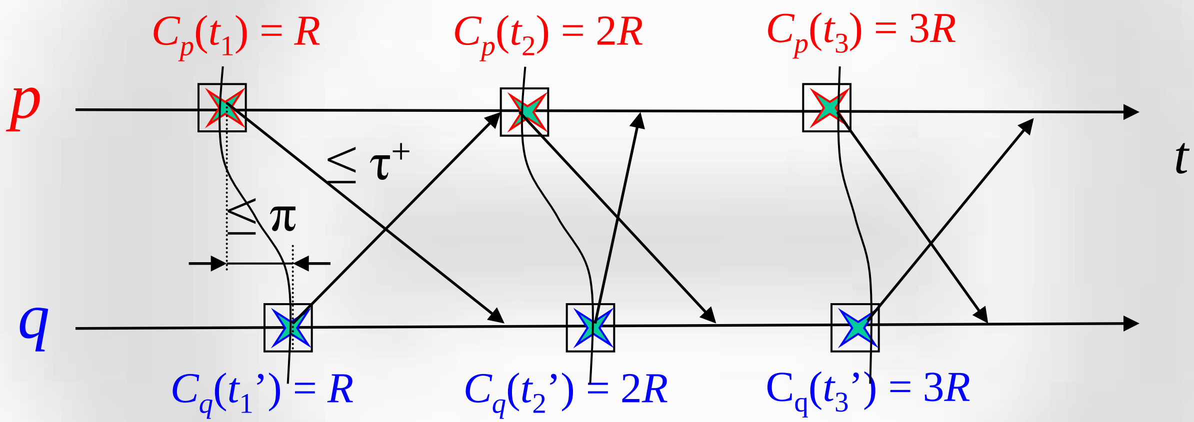
- Satellite clocks synchronized to USNO atomic master clock
- GPS-Receiver solves system of equations

$$t_i + |\chi - s_i|/c + \Delta = T_i$$

- 4 satellites required to determine $\chi = (x, y, z)$ and Δ
- 1 satellite sufficient for Δ if χ is already known

Why are Synchronized Clocks Useful?

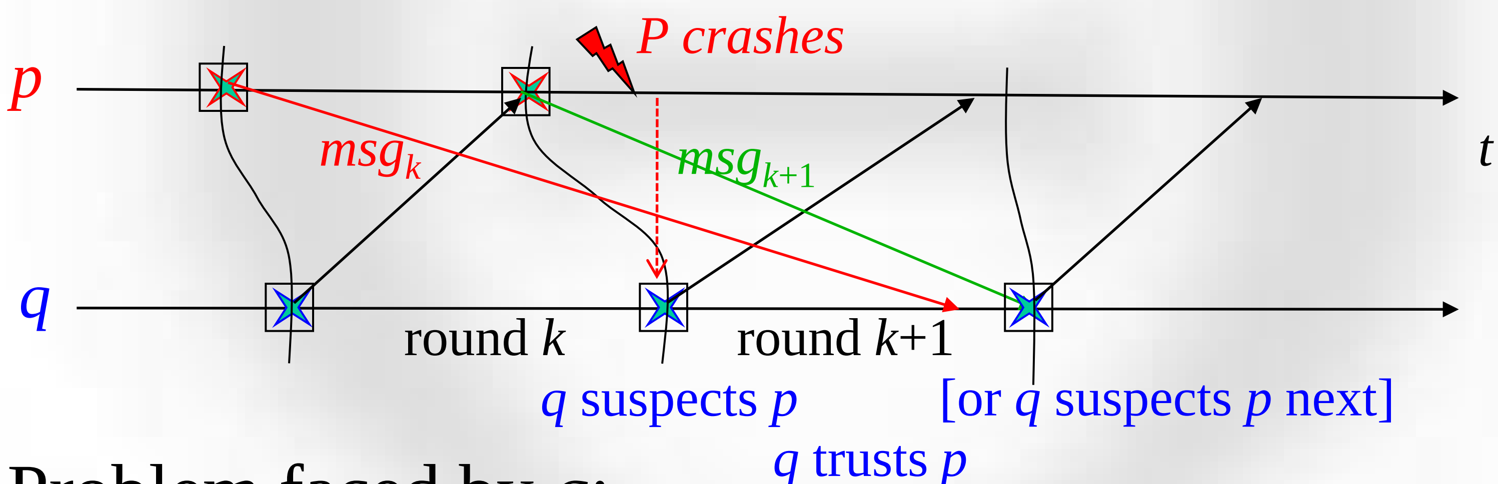
- Synchronized clocks allow to simulate communication-closed lock-step rounds via clock time [NT93]:



- **Only requirement: $R \geq \tau^+ + \pi$ holds!**
- Lock-step rounds ⟹ perfect failure detection at end of rounds

Perfect FD $\not\Rightarrow$ Lock-Step Round Simulation

- Attempt round simulation at *p*: Waiting for either
  - arrival of round message from *q*, or
  - *p*‘s instance of P suspects *q*



- Problem faced by *q*:
  - msg_k not received in round *k*, although *p* alive after round *k*
  - *q* even receives msg_{k+1} in round *k*+1 in this example

Using RT Clocks: Deficiencies

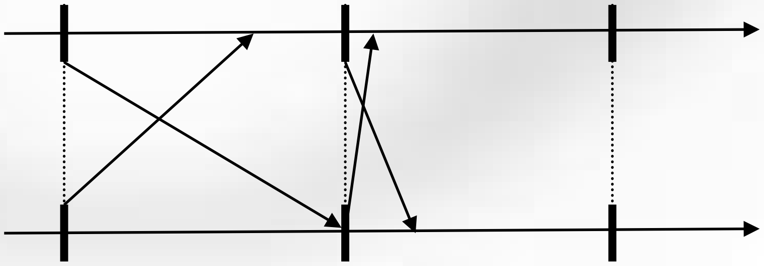
- Algorithms like do_roundtrip(.) have system-dependent time values (unit „seconds“) in their code / variables → not easily portable to e.g. faster hardware
- Fail-operational systems might tolerate occasional loss of timeliness properties – but never of safety properties
- Unfortunately:
  - Safety properties like agreement typically rely on the reliable operation of do_roundtrip(.) and similar primitives
  - End-to-end delay bounds τ^+ that always hold are difficult to determine in real systems

➢ Try to relax timing assumptions in ParSync models …

Partially Synchronous Models

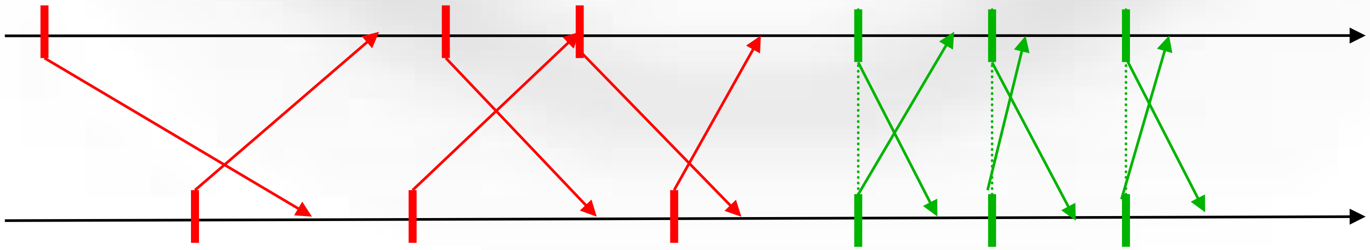
Recall: Synchronous Model

- „The“ classic model
  - Transmission delay bound τ^+
  - Computing step time bound μ^+
  - Bounded-drift local clocks available
- Allows (Byzantine-tolerant) implementation of
  - Internal clock synchronization
  - Lock-step rounds
  - etc.



The Timed Asynchronous Model

- Cristian & Fetzer [CF99]:
  - Alternating bad and good periods:
    - Transmission delay bound τ^+
    - Computing step time bound μ^+
  - Bounded-drift local RT clocks available
  - Local clocks allow to detect good/bad periods → TA algorithms are always safe and live in good periods
- TA algorithms allow to implement (non-Byzantine) fail-aware services, including eventual lock-step rounds



Classic Partially Synchronous Models (I)

- „The" classic ParSync models
  Dolev, Dwork & Stockmeyer [DDS87]
  Dwork, Lynch & Stockmeyer [DLS88]
  Attiya, Dwork, Lynch & Stockmeyer [ADLS94]
- Semi-synchronous model by Ponzio & Strong [PS92]
- Common system parameters:
  - Bounded processor speed ratio $\Phi = \mu^+/\mu^-$
  - Transmission delay bound Δ
- Archimedean model by Vitanyi [Vit84]
  - Bounded speed ratio $S = \tau^+/\mu^-$

Classic Partially Synchronous Models (II)

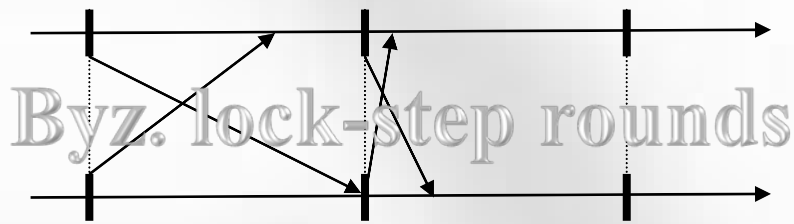
Processes can locally time-out messages:

- The classic ParSync models [DDS87, DLS88] and [ADLS94] assume
  - Δ given in multiples of (unknown) minimal computing step time μ^- [hence $\tau^+ = \Delta \cdot \mu^-$ real-time seconds]
  - spin loop counting f(Φ,Δ) steps allows to time-out messages [implements local clock with real-time rate $\in [1/\Phi,1]$]
- Archimedean model [Vit84] also allows to time-out messages via spin-loop for *S* steps
- Semi-synchonous model [PS92] assumes
  - $\Delta = \tau^+$ given in real-time seconds
  - bounded-drift local RT clocks available for timing-out messages

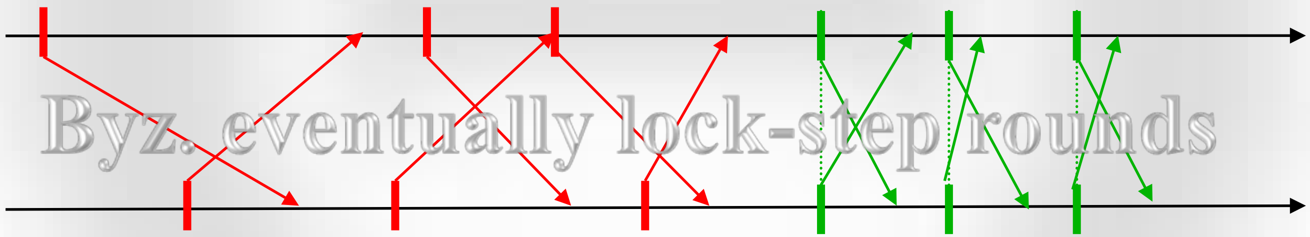
Classic Partially Synchronous Models (III)

Variants of ParSync models: System parameters (Δ, Φ)

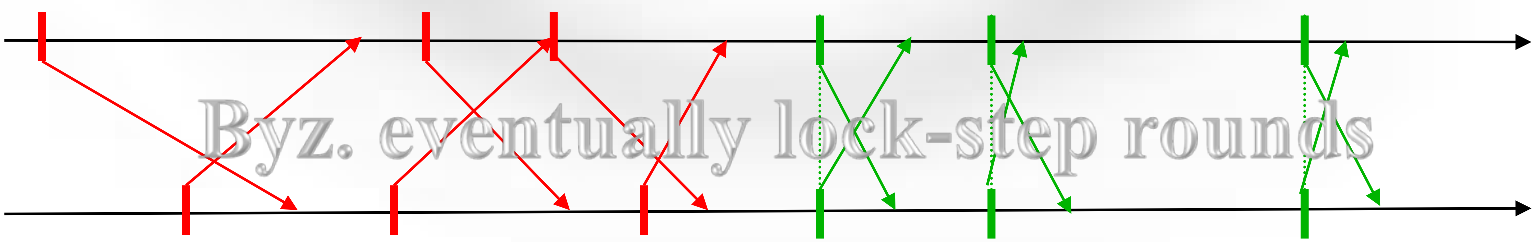
1. known and hold
   from the beginning



2. known and hold from unknown global stabilization time (GST) on



3. unknown and hold from the beginning / from GST on:
   Learn (Δ, Φ), by continuously increasing estimate values



Time-Free Message-Timeout in ParSync ?

- Implementation of do_roundtrip(p) in the ParSync models of [DLS88] or [Vit85]:

```
{  send ping to p
   for i=1 to x do no-op          /* x=f(Δ, Φ) resp. x=f(s) is
                                              dimensionless! */
   if pong did not arrive then
          return DEAD
   else
          return ALIVE
}
```

- But: No obvious correlation between processor step times and message delays → not really time-free …

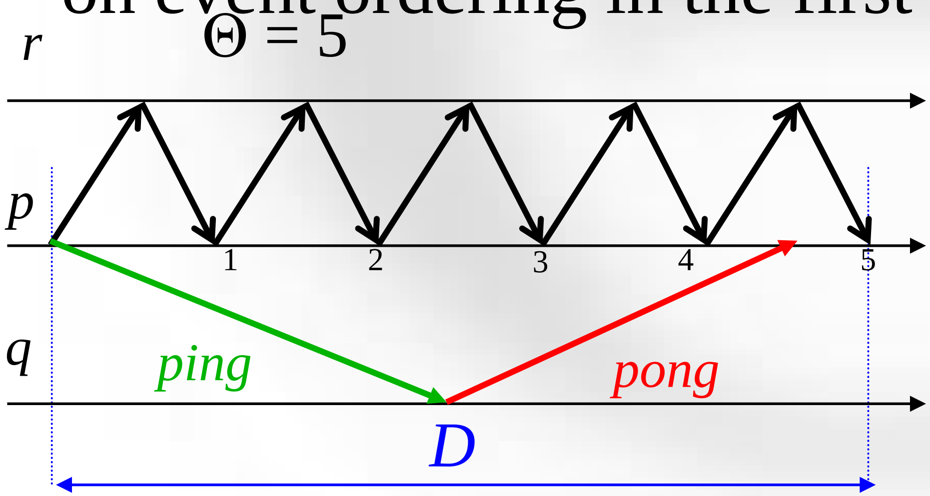
The Θ/ABC-Model

In classic ParSync model:

- only less than Θ roundtrips can occur during any single round-trip
- Timing assumptions are primarily used for ordering events
- Is it possible to define a time-free ParSync model based on event ordering in the first place?

For example: Assume that:

- Actual duration (D) irrelevant

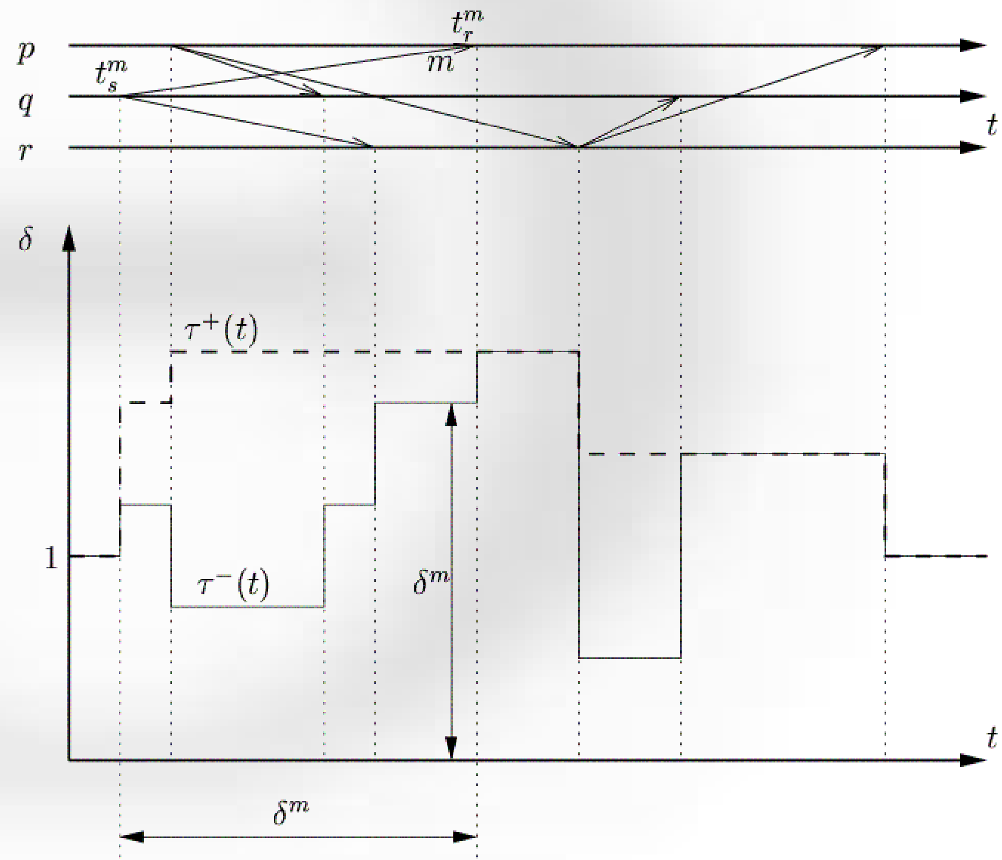



```
status = do_roundtrip(q)
{   send ping to q
    for i=1 to Θ do
    begin
       send delay_ping(i) to r
       wait for delay_pong(i) from r
    end
    if pong did not arrive then
           return DEAD
    else
           return ALIVE
}
```

The Θ-Model: Bounded E-t-E Delay Ratio

LeLann & Schmid [LS03], Widder & Schmid [WS09]

- End-to-end delays of all messages in transit at t
  - minimum $\tau^{-}(t)$
  - maximum $\tau^{+}(t)$
- $\tau^{+}(t)$ and $\tau^{-}(t)$ may vary arbitrarily with time, but:
- Ratio $\tau^{+}(t)/\tau^{-}(t)$ bounded by [known or even unknown] system parameter Θ



Byzantine FT Clock Sync in the Θ-Model

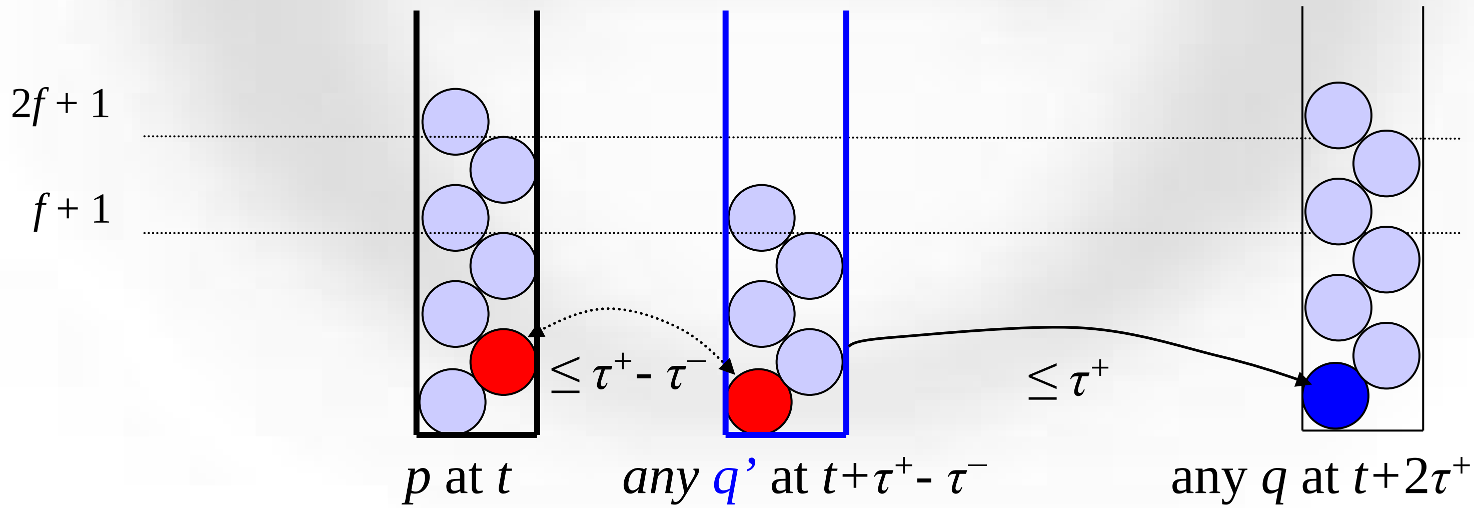
On init
→ send *tick*(0) to all; C := 0;
If got *tick*(l) from f +1 nodes **and** l > C
→ send *tick*(C+1),…, *tick*(l) to all;
C := l;
If got *tick*(C) from 2f +1 nodes
→ send *tick*(C+1) to all;
C := C+1;

For $n \geq 3f + 1$ with up to f Byz. failures:

- Suppose p sends *tick*(C+1) at time t
- Then, q also sends *tick*(C+1) by time $t + 2\tau^+ - \tau^-$

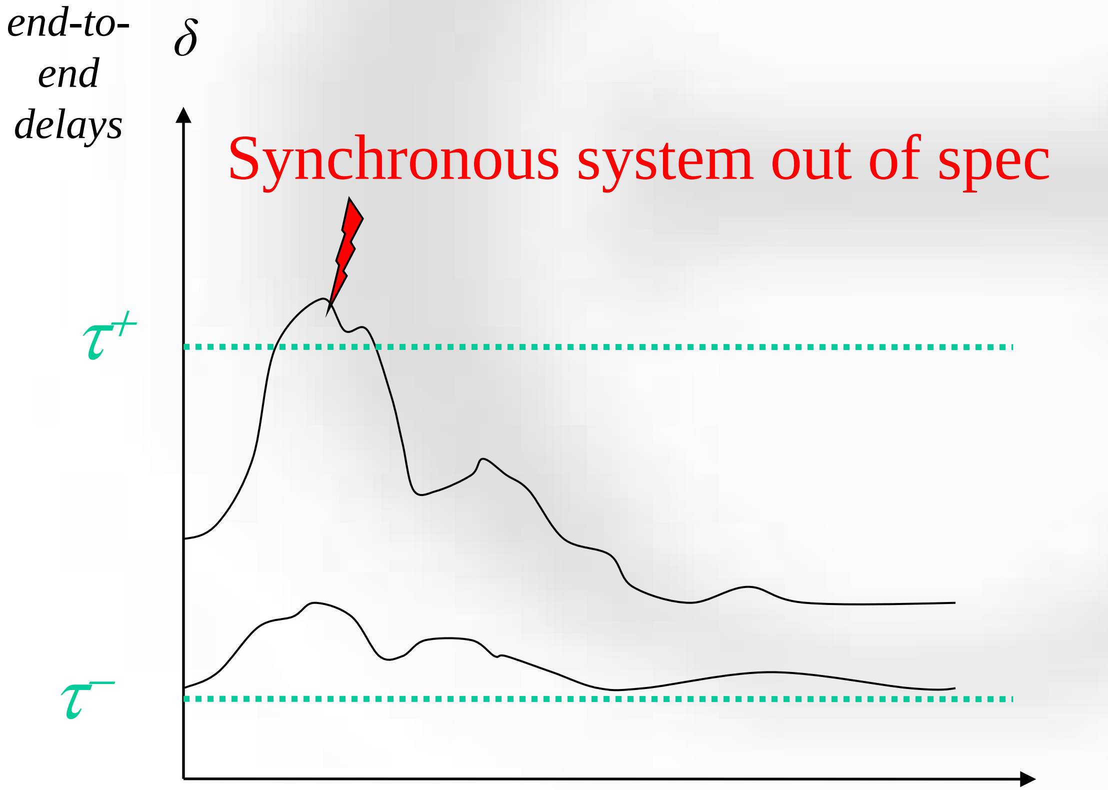
\+ Fastest tick-frequency of any p: $1/\tau^-$

⇒ Clock ticks occur approximately synchronously, with precision $\pi(\Theta)$



Correlation → Coverage Expansion

- Given some bound τ^+ and τ^- assumed during system design (as used in synchronous systems), compute $\Theta = \tau^+ / \tau^-$
- Unanticipated overload: $\tau^+(t) > \tau^+$ — if $\tau^+(t) \leq \Theta\tau^-(t)$, however,



Θ-system still OK

Note:

- $\tau^+(t) = \tau^+ + \alpha(t)$
- $\tau^-(t) = \tau^- + \alpha(t)/\Theta$
  sufficient for Θ to hold!

Shortcomings Θ-Model

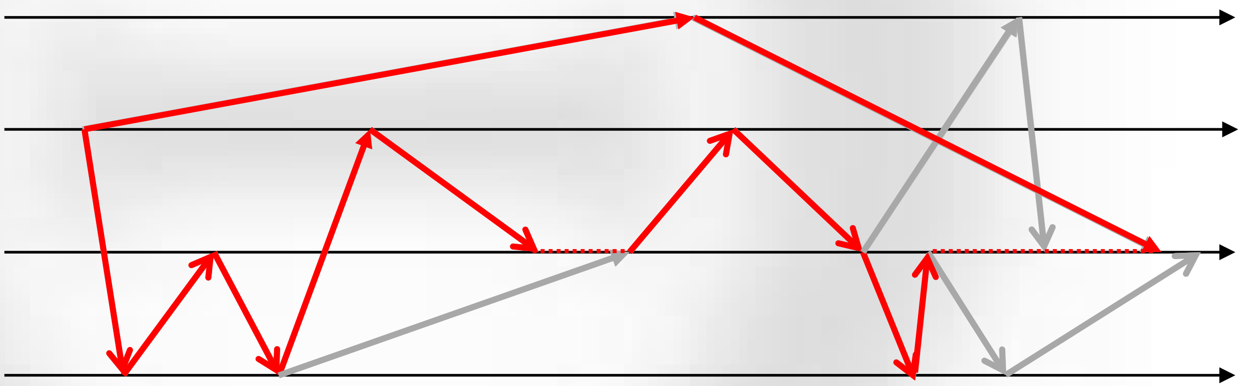
- Correlation between slow and fast messages need not exist for all messages
  - Some very fast messages [even $\tau^- = 0$] may be in transit somewhere in the system during a slow message
  - Correlation and hence coverage expansion does not exist in such cases
- Need a more relaxed definition of the relation between slow and fast messages
  - All that is actually needed is to constrain the number of fast messages during a slow one
  - No need for a correlation of unrelated messages, and at every point in time t

The Asynchronous Bounded Cycle Model

Robinson & Schmid [RS08]

- The ABC Model just bounds the ratio of the **number** of forward and backward-oriented messages in cycles

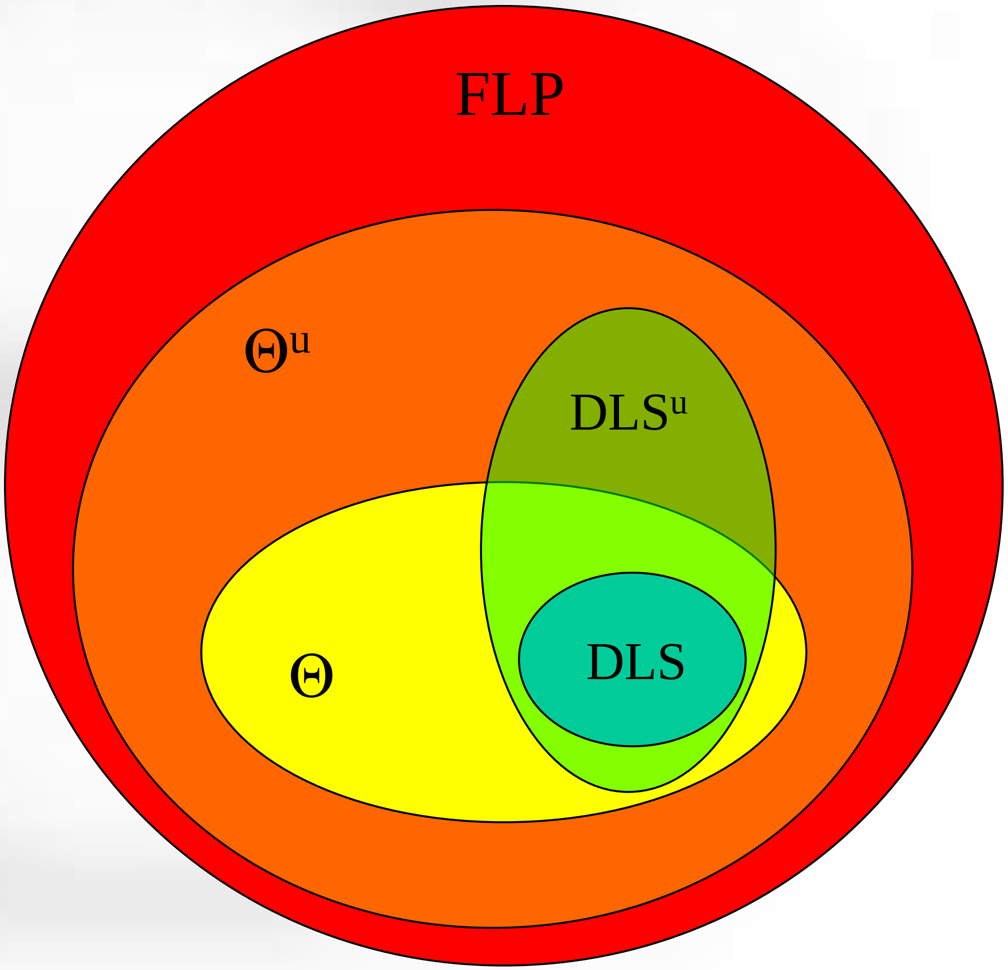
- Example: $\Theta = 4.5$
- 2 consecutive „slow“ messages
- Cycle with 9 enclosed „fast“ messages
- No larger cycles allowed



- No implicit or explicit reference to real-time
  - ✓ Messages with $\tau^{-}(t) = 0$ allowed
  - ✓ No need to relate independent messages in the system
  - ✓ **We proved: Any Θ-algorithm works correctly in the ABC model**

Partial Order of ParSync Models

- DLS … [DLS88] with known Δ, Φ
- Θ … ABC/Θ-Model with known Θ
- DLS^u … [DLS88] with unknown Δ, Φ
- Θ^u … ABC/Θ-Model with unknown Θ
- FLP … FLP-Model



Even Weaker ParSync Models?

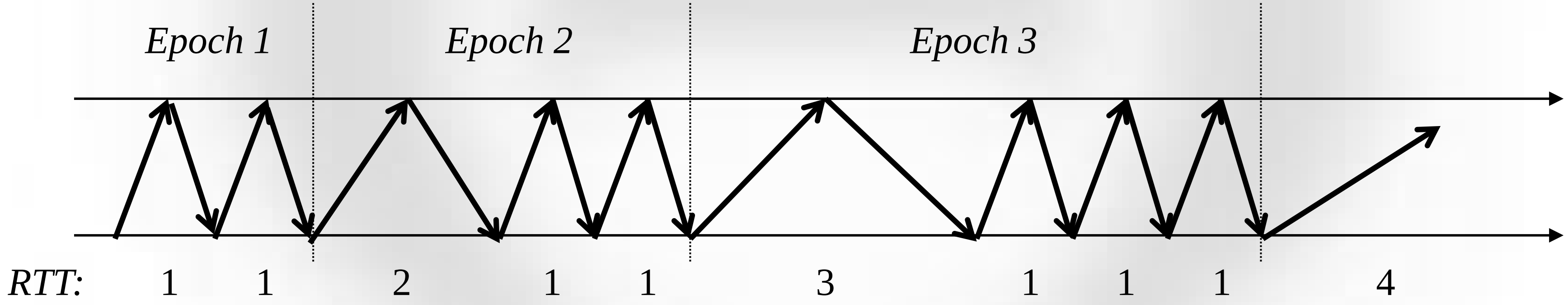
- All the ParSync Models seen so far allow to build
  - lock-step rounds, or at least
  - eventual lock-step rounds
- Solving consensus is easy here.
- We know that lock-step rounds are stronger than failure detectors that are sufficient for solving consensus:
  - Perfect failure detector P
  - Leader oracle Ω
- Are there weaker ParSync models where only such FDs can be implemented?

Weaker Partially Synchronous Models

Finite Average Roundtrip-Time Model (I)

Fetzer, Schmid and Süsskraut [FSS04]

- Asynchronous system with crash failures
- Unknown lower bound μ^- for computing step time
- Unknown average round-trip time bounds $\lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n} RTT(k) < \infty$



- $RTT(k)$ and hence τ^+ unbounded, yet
- Average after n „Epochs“ is $$\frac{n(n+1)}{n(n+1)-(n-1)n/2} = 2\cdot\frac{n^2+n}{n^2+3n} < \infty$$

Finite Average Roundtrip-Time Model (II)

- The FAR model assumptions
  - do not allow to implement lock-step rounds
  - do allow to implement the eventually perfect FD P
  - can solve consensus if $n > 2f$
- Key ideas for P implementation:
  - Implement weak local clock [via spin-loop] for timing-out messages
  - Time-out roundtrips using adaptive timeout value TV
    - If fast RT occurs [before TO]: Increase TV, to prepare for future slow RTs
    - If slow RT occurs [after TO]: (Could) decrease TV, since fast RTs must eventually follow due to finite average RTT

Weak Timely Link Models (I)

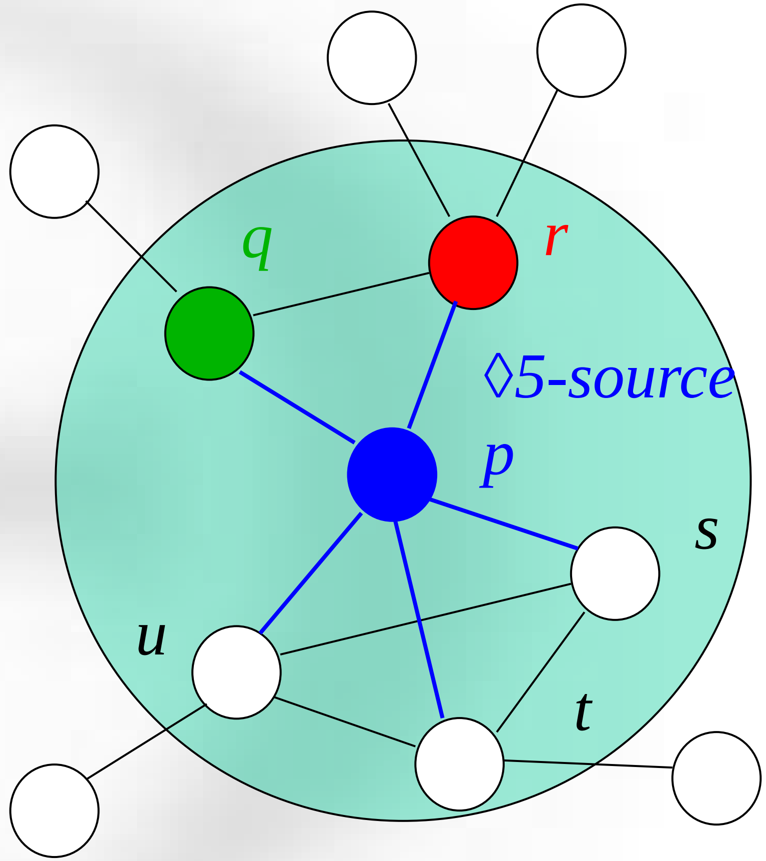
Aguilera, Delporte, Fauconnier, Toueg [ADFT04], Hutle, Malkhi, Schmid, Zhou [HMSZ09]:

- Partially synchronous processors (Φ) with crash failures
- Almost all communication asynchronous, except:
- At least one process *p* must be an $\Diamond f$-source:
  - After some (unknown) time, *p* has timely links to at least *f* neighbors [*No message sent at time t is processed after* $t+\tau^+$ *(unknown)*]
  - Note: A link to a crashed process is timely per definition!
- Allows to implement Ω, and hence solving consensus for $n > 2f$
- An $\Diamond f$-1-source is provably not sufficient
- Currently weakest WTL model [HMSZ09]: A moving $\Diamond f$-source, where the *f* timely links can change with time

Weak Timely Link Models (II)

Ω implementation: Every process

- periodically broadcasts heartbeat message (HB)
- times-out HBs of all neighbors
  - using weak local clock [implemented via step counting in spin-loop]
  - timeout value increased on every TO [= no HB received before expiration]
- broadcasts accusation message *acmsg*(*q*) on every TO for *q*'s HB
- if *n*-*f* *acmsg*(*q*) are received, then increment *acc_count*[*q*]
- Ω-output: *q* with min. *acc_count*[*q*]



- All processes accuse crashed r → *acc_count*[*r*] continuously grows
- 5+1 processes never accuse p → incrementing *acc_count*[*p*] stops

Even Weaker ParSync Models?

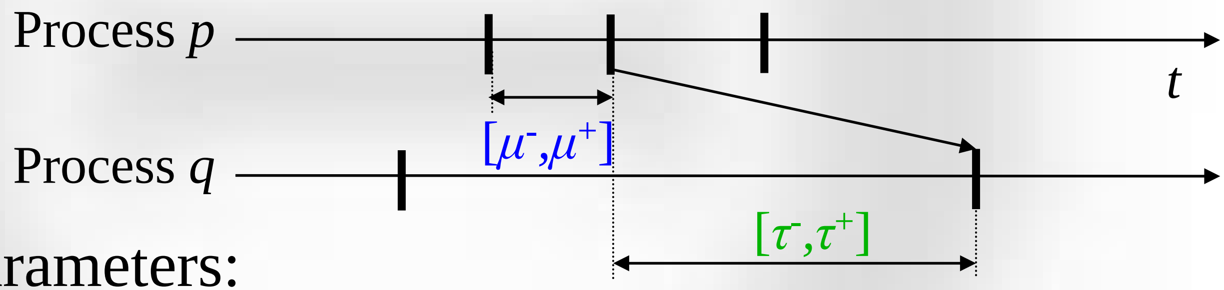
- Investigate models for weaker problems, like *k*-set consensus [Cha93]
  - Biely, Robinson & Schmid [BRS09]: Weak ParSync models
  - Gafni & Kouznetsov [GK09]: Weakest FD
- Open major challenge:

How to quantify and compare the assumption coverage of ParSync models in real systems?

Distributed Real-Time Systems

Recall Classic DC Modeling and Analysis

- Processors/processes modeled as interacting state machines
- **Zero-time** atomic computing steps, usually time-triggered
  - Message Passing (MP): [receive] + compute + [send]
  - Shared Memory (SHM): [accessSHM] + compute

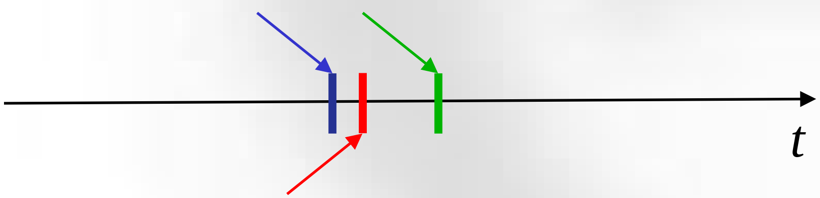


- System timing parameters:
  - Operation durations modeled via **inter-step times $[\mu^-,\mu^+]$** (often $\mu^- = 0$)
  - Message delays modeled as **end-to-end delays $[\tau^-, \tau^+]$** (often $\tau^- = 0$)
- DC research established a wealth of results:
  - Correctness proofs of distributed algorithms
  - Impossibility & lower bound results

Real-Time Properties ?

Classic modeling:

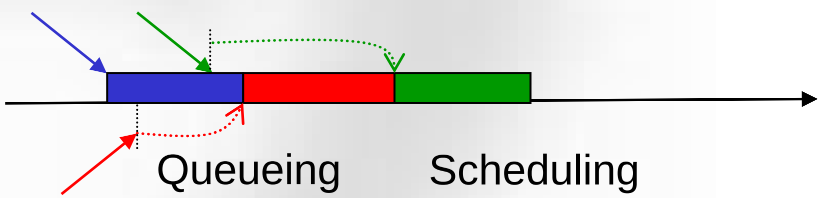
- $[\mu^-,\mu^+]$, $[\tau^-,\tau^+]$ are *a priori given* system parameters (alg-indep.)
- Analysis considers occurrence times of steps independently of each other:



- No queueing & scheduling in the picture
- Too optimistic time complexity

Reality:

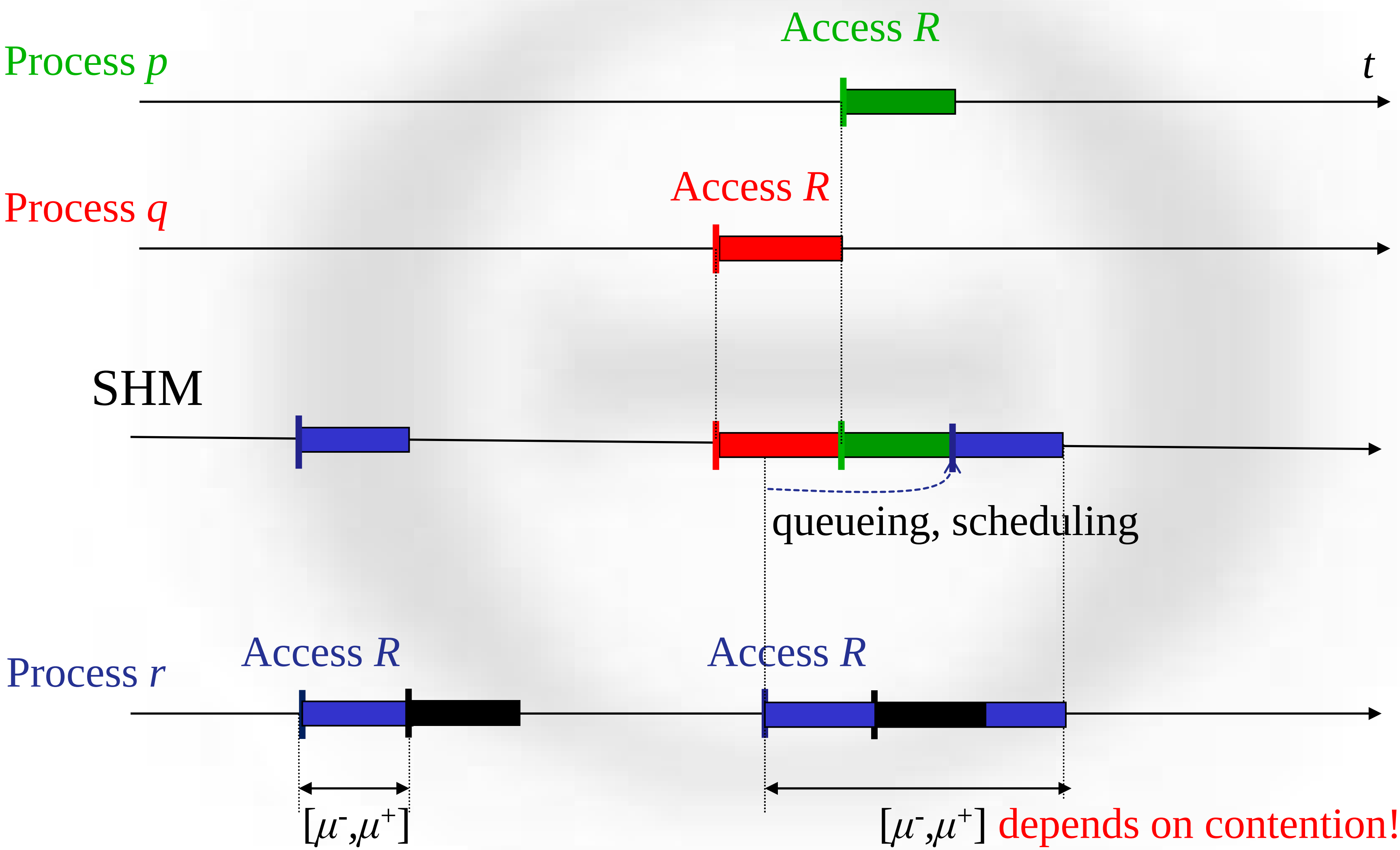
- $[\mu^-,\mu^+]$, $[\tau^-,\tau^+]$ depend on algorithms + scheduling policies
- Non-preemptible operations → steps not independent:



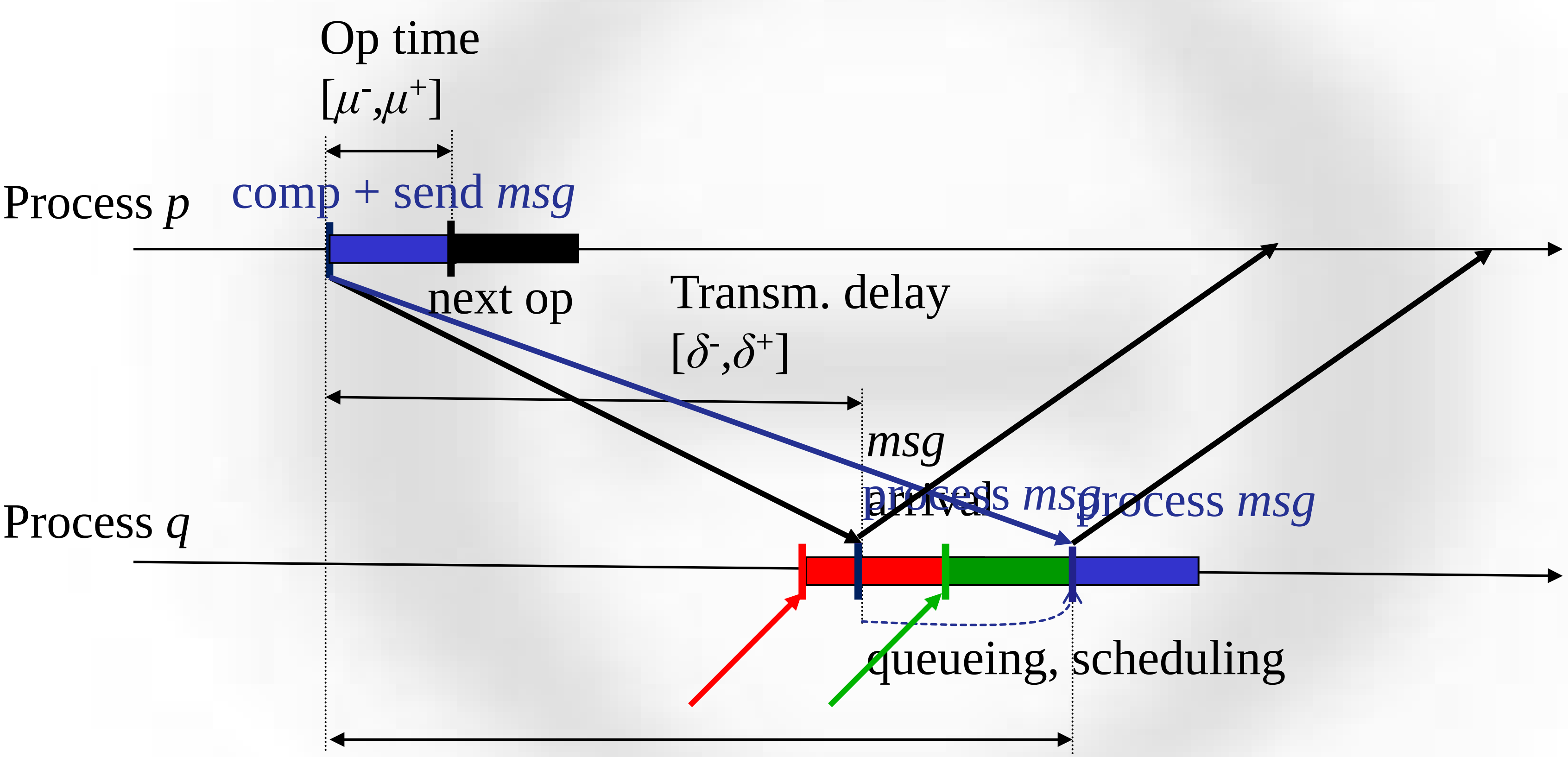
- Time complexity analysis involves real-time analysis

✓ Moser & Schmid [MS06,MS08]

Fixed Step Times in SHM Systems ?



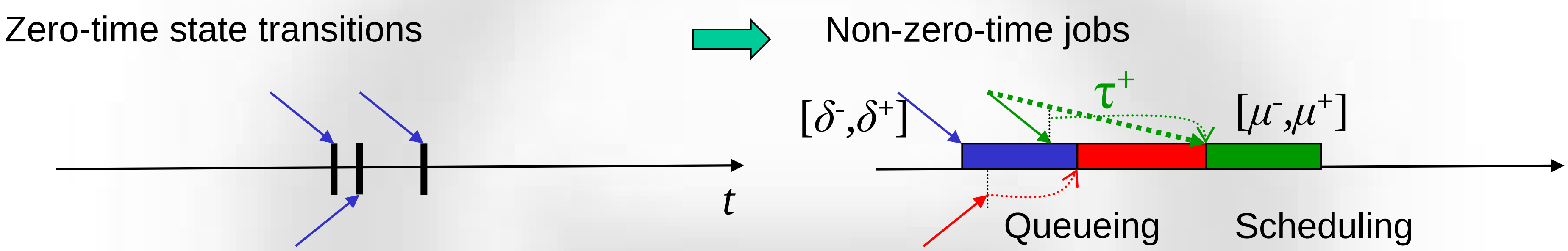
Fixed End-to-End Delays in MP Systems ?



E-t-e delay [τ^-, τ^+] depends on receiver load!

Real-Time Distributed Computing Model

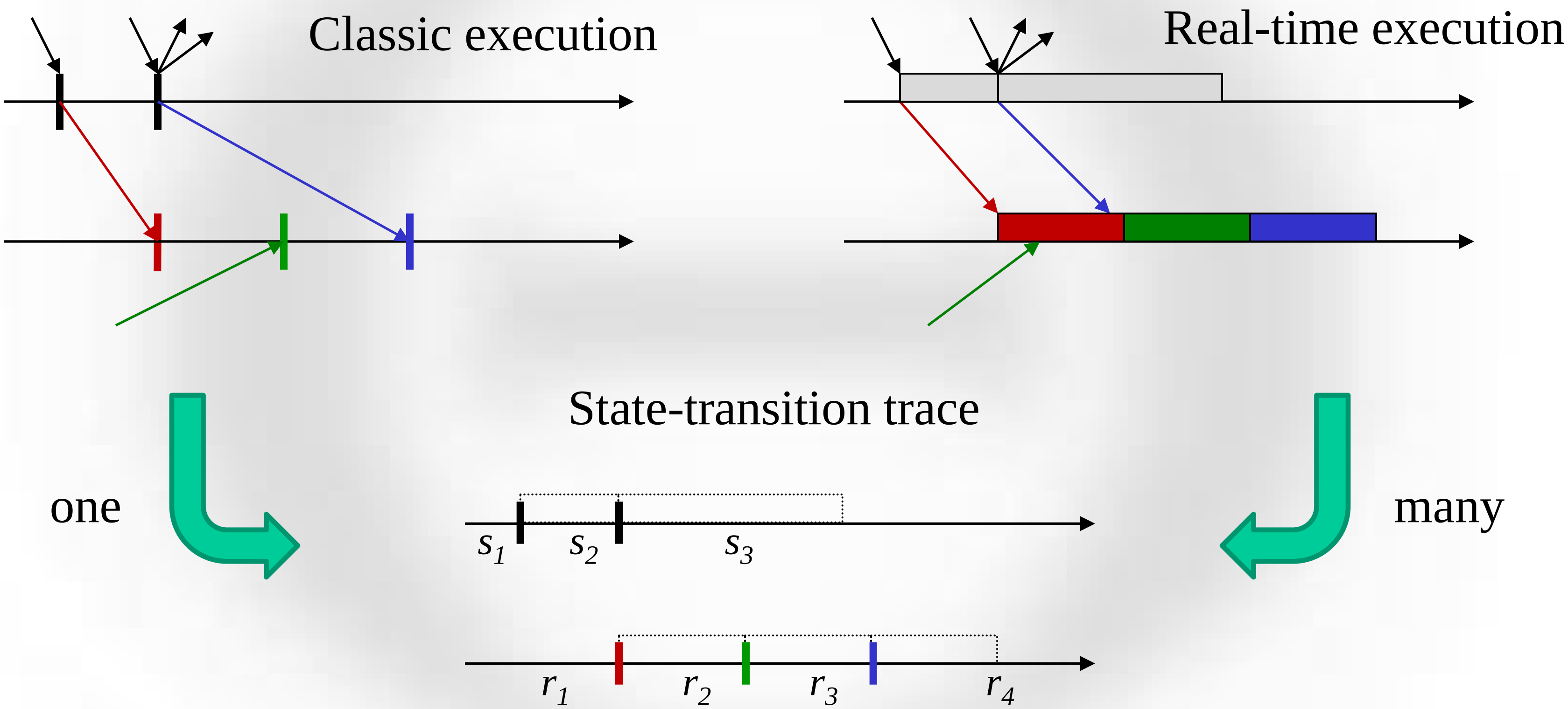
- RT model core features [Moser & Schmid, OPODIS'06]



- Investigate relation classic vs. RT model
  - Carry over classic failure models ?
  - Carry over classic correctness proofs ?
  - Carry over classic time complexity results ?
  - Carry over classic impossibility & lower bound results ?
- Conduct real-time analysis for e-t-e delays τ^+

State-Transition Problems

Can be defined for both models in the same way:



State-transition problem = a set of state-transition traces

Example: Problem Definition

Deterministic Drift-Free Clock Synchronization

$is\_finalstate(g) :\Leftrightarrow \forall g' \succ g : \forall p : s_p(g) = s_p(g')$

Termination: All processors eventually terminate.

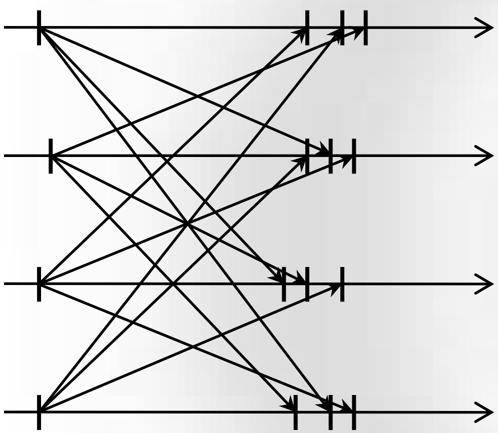
$\exists g : is\_finalstate(g)$

Agreement: After all processors have terminated, all processors have adjusted clocks within γ of each other.

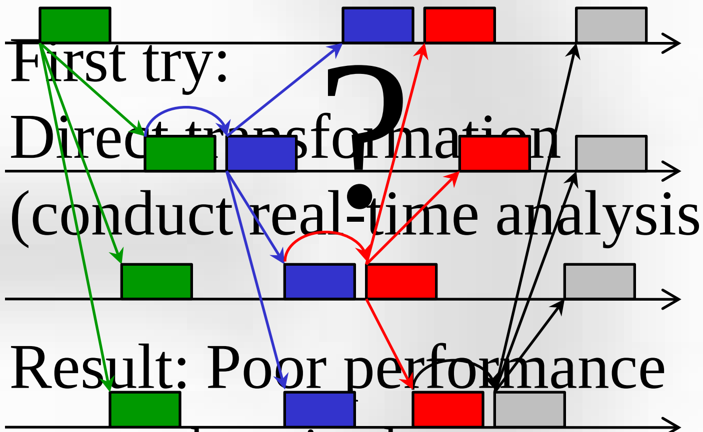
$\forall g : is\_finalstate(g) \Rightarrow (\forall p, q : |AC_p(g) - AC_q(g)| \leq \gamma)$

Example: Drift-Free Clock Sync

Classic Model:



Real-time Model:



First try:

Direct transformation

(conduct real-time analysis)

Result: Poor performance

- sub-optimal worst-case precision
- O(n) time

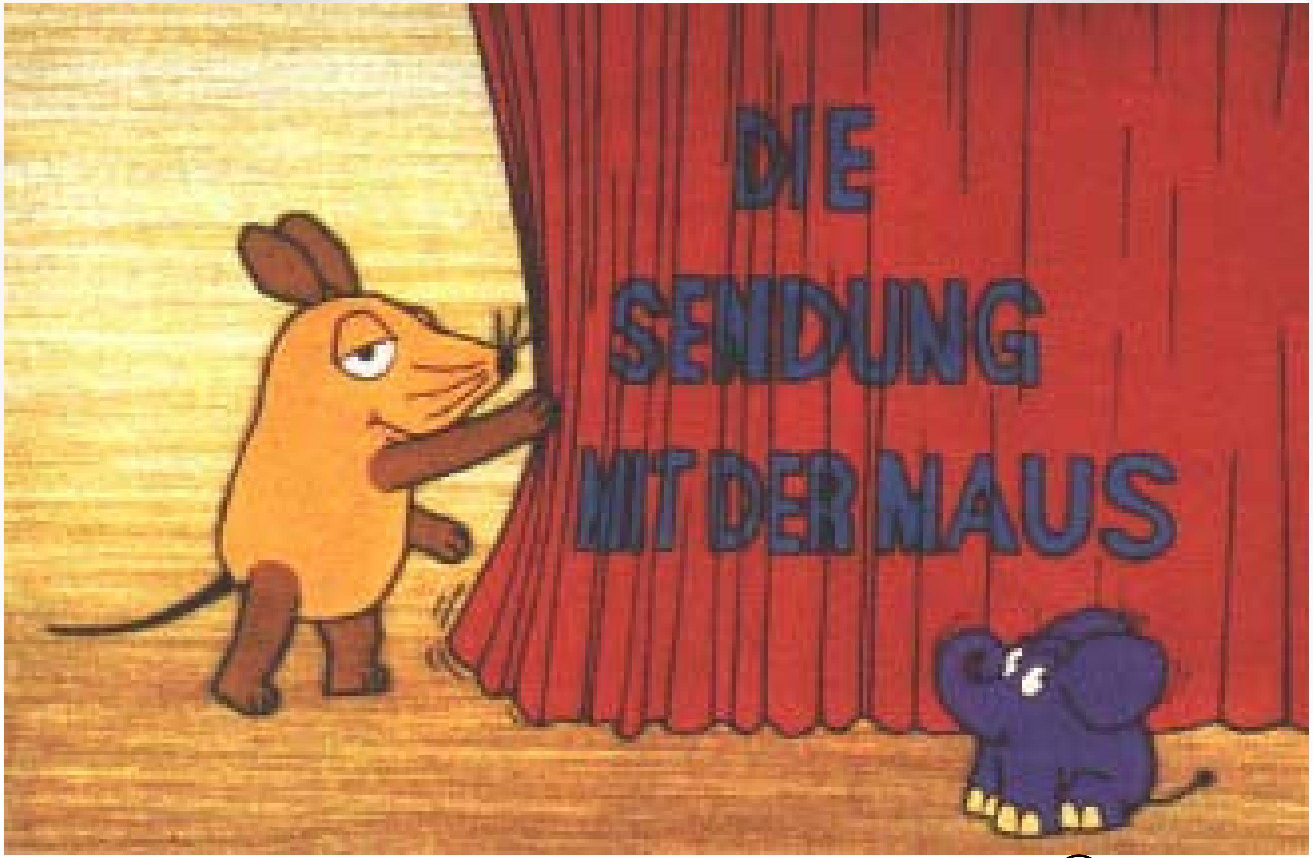
Result:

- Optimal worst-case precision
- Optimal running time O(1)

Result:

- Optimal worst-case precision
- Achievable only in time O(n)
- O(1) time algorithm with sub-optimal precision also exists

The End
(Part 2)



References

- [ADFT03] Marcos K. Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. On implementing Omega with weak reliability and synchrony assumptions. In Proceeding of the 22nd Annual ACM Symposium on Principles of Distributed Computing (PODC’03), pages 306–314, New York, NY, USA, 2003. ACM Press.
- [ADFT04] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Communication-efficient leader election and consensus with limited link synchrony. In Proceedings of the 23th ACM Symposium on Principles of Distributed Computing (PODC’04), pages 328–337, St. John’s, Newfoundland, Canada, 2004. ACM Press.
- [ADLS94] Hagit Attiya, Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. Journal of the ACM (JACM), 41(1):122–152, 1994.
- [BRS09] M. Biely, P. Robinson, and U. Schmid, Weak synchrony models and failure detectors for message passing k-set agreement,”in Proceedings of the International Conference on Principles of Distributed Systems (OPODIS’09), ser. LNCS. Nimes, France: Springer Verlag, Dec 2009.
- [Cha93] S. Chaudhuri, “More choices allow more faults: set consensus problems in totally asynchronous systems,” Inf. Comput., vol. 105, no. 1, pp. 132–158, 1993.
- [CT96] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. Journal of the ACM, 43(2):225–267, March 1996.
- [CF99] Flaviu Cristian and Christof Fetzer. The timed asynchronous distributed system model. IEEE Transactions on Parallel and Distributed Systems, 10(6):642–657, 1999.
- [DDS87] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. Journal of the ACM, 34(1):77–97, January 1987.
- [DHS86] Danny Dolev, Joseph Y. Halpern and H. Raymond Strong. On the Possibility and Impossibility of Achieving Clock Synchronization 32:230-250, 1986.
- [DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. Journal of the ACM, 35(2):288–323, April 1988.

References

- [FC97b] Christof Fetzer and Flaviu Cristian. Integrating external and internal clock synchronization. J. Real-Time Systems, 12(2):123--172, March 1997.
- [FSS05] Christof Fetzer, Ulrich Schmid, and Martin Süßkraut. On the possibility of consensus in asynchronous systems with finite average response times. In Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS'05), pages 271–280, Washington, DC, USA, June 2005. IEEE Computer Society.
- [FML86]] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy Impossibility Proofs for Distributed Consensus Problems, Distributed Computing 1(1), 1986, p. 26—39.
- [FLP85] Michael J. Fischer, Nancy A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. Journal of the ACM, 32(2):374–382, April 1985.
- [FSFK06] Matthias Fuegger, Ulrich Schmid, Gottfried Fuchs, and Gerald Kempf. Fault-Tolerant Distributed Clock Generation in VLSI Systems-on-Chip. In Proceedings of the Sixth European Dependable Computing Conference (EDCC-6), pages 87–96. IEEE Computer Society Press, October 2006.
- [Gaf98] Eli Gafni. Round-by-round fault detectors (extended abstract): unifying synchrony and asynchrony. In Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, pages 143–152, Puerto Vallarta, Mexico, 1998. ACM Press.
- [GK09] E. Gafni and P. Kuznetsov, "The weakest failure detector for solving ksetagreement," in 28th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2009), 2009.
- [HL02] Jean-Francois Hermant and Gerard Le Lann. Fast asynchronous uniform consensus in real-time distributed systems. IEEE Transactions on Computers, 51(8):931–944, August 2002.
- [HW05] Jean-Francois Hermant and Josef Widder. Implementing reliable distributed real-time systems with the Θ-model. In Proceedings of the 9th International Conference on Principles of Distributed Systems (OPODIS 2005), volume 3974 of LNCS, pages 334–350, Pisa, Italy, December 2005. Springer Verlag.
- [HMSZ09] Martin Hutle, Dahlia Malkhi, Ulrich Schmid, and Lidong Zhou. Chasing the weakest system model for implementing Omega and consensus. IEEE Transactions on Dependable and Secure Computing 6(4), 2009

References

- [HS97] Dieter Hoechtl and Ulrich Schmid. Long-term evaluation of GPS timing receiver failures. In Proceedings of the 29th IEEE Precise Time and Time Interval Systems and Application Meeting (PTTI'97), pages 165--180, Long Beach, California, December 1997.
- [Lam84] Leslie Lamport. Using Time Instead of Timeout for Fault-Tolerant Distributed Systems. ACM Transactions on Programming Languages and Systems 6(2), April 1984, p. 254-280
- [LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. ACM Transactions on Programming Languages and Systems, 4(3):382–401, July 1982.
- [LS03] Gerard LeLann and Ulrich Schmid. How to implement a timer-free perfect failure detector in partially synchronous systems. Technical Report 183/1-127, Department of Automation, Technische Universit¨at Wien, January 2003. (Replaced by Research Report 28/2005, Institut f¨ur Technische Informatik, TU Wien, 2005.).
- [LWL88] Jennifer Lundelius-Welch and Nancy A. Lynch. A new fault-tolerant algorithm for clock synchronization. Information and Computation, 77(1):1--36, 1988.
- [Mil95] David L. Mills. Improved algorithms for synchronizing computer network clocks. IEEE Transactions on Networks, pages 245--254, June 1995.
- [MMR03] Achour Mostefaoui, Eric Mourgaya, and Michel Raynal. Asynchronous implementation of failure detectors. In Proceedings of the International Conference on Dependable Systems and Networks (DSN'03), San Francisco, CA, June 22–25, 2003.
- [Mos09] Heinrich Moser, Towards a real-time distributed computing model, *Theoretical Computer Science, vol. 410, no. 6–7, pp. 629–659, Feb 2009.*
- [MS06] Heinrich Moser and U. Schmid, Optimal clock synchronization revisited: Upper and lower bounds in real-time systems, in *Proceedings of the International Conference on Principles of Distributed Systems (OPODIS), ser. LNCS 4305. Bordeaux &* Saint-Emilion, France: Springer Verlag, Dec 2006, pp. 95–109.
- [MS08] Heinrich Moser and Ulrich Schmid. Optimal deterministic remote clock estimation in real-time systems. In Proceedings of the International Conference on Principles of Distributed Systems (OPODIS), pages 363–387, Luxor, Egypt, December 2008.
- [NT93] Gil Neiger and Sam Toueg. Simulating Synchronized Clocks and Common Knowledge in Distributed Systems. JACM 40(3), April 1993, p. 334-367.

References

- [PS92] Stephen Ponzio and Ray Strong. Semisynchrony and real time. In Proceedings of the 6th International Workshop on Distributed Algorithms (WDAG’92), pages 120–135, Haifa, Israel, November 1992.
- [RS08] Peter Robinson and Ulrich Schmid. The Aynchronous Bounded Cycle Model. Proceedings of the 10th Internlation Symposium on Stailization, Safety and Security of Distribted Systems (SSS‘08), Detroit, USA. Springer LNCS 5340, p. 246-262.
- [SAACBBBCLM04] L. Sha, T. Abdelzaher, K.-E. Arzen, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, “Real time scheduling theory: A historical perspective,” *Real-Time Systems Journal, vol. 28, no. 2/3, pp. 101–155, 2004.*
- [Sch86] Fred B. Schneider. A paradigm for reliable clock synchronization. In Proceedings Advanced Seminar of Local Area Networks, pages 85--104, Bandol, France, April 1986.
- [SKMNCK99] Ulrich Schmid, Johann Klasek, Thomas Mandl, Herbert Nachtnebel, Gerhard R. Cadek, and Nikolaus Keroe. A Network Time Interface M-Module for distributing GPS-time over LANs. J. Real-Time Systems, 18(1), 2000, p. 24-57.
- [SS97] Ulrich Schmid and Klaus Schossmaier. Interval-based clock synchronization. J. Real-Time Systems, 12(2):173--228, March 1997.
- [SS99] Ulrich Schmid and Klaus Schossmaier. How to reconcile fault-tolerant interval intersection with the Lipschitz condition. Distributed Computing 14(2):101-111. 2001.
- [ST87] T. K. Srikanth and Sam Toueg. Optimal clock synchronization. Journal of the ACM, 34(3):626--645, July 1987.
- [Vit84] Paul M.B. Vitányi. Distributed elections in an Archimedean ring of processors. In proceedings of the sixteenth annual ACM symposium on theory of computing, pages 542-547. ACM Press, 1984.
- [WLS95] Josef Widder, Gerard Le Lann, and Ulrich Schmid. Failure detection with booting in partially synchronous systems. In Proceedings of the 5th European Dependable Computing Conference (EDCC-5), volume 3463 of LNCS, pages 20–37, Budapest, Hungary, April 2005. Springer Verlag.
- [WS09] Josef Widder and Ulrich Schmid. The Theta-Model: Achieving Synchrony without Clocks. Distributed Computing 22(19; 2009, p. 29-47