# Distributed Algorithms

## (Part 1)

## RiSE Winter School 2012

Ulrich Schmid

Institute of Computer Engineering, TU Vienna

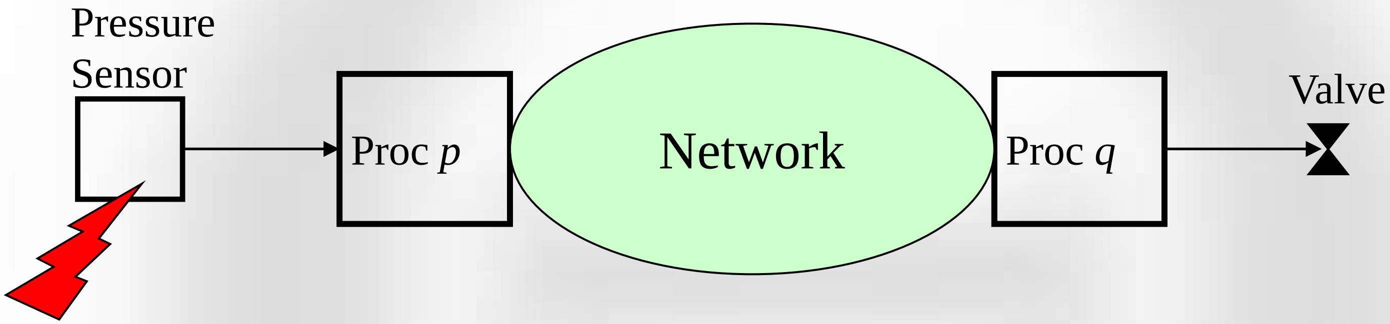
Embedded Computing Systems Group E182/2

*s@ecs.tuwien.ac.at*

TU WIEN FAKULTÄT FÜR INFORMATIK Faculty of Informatics

# Target: Fault-tolerant Distributed RT Systems

Spatially distributed reactive computations



Pressure Sensor

Proc *p*

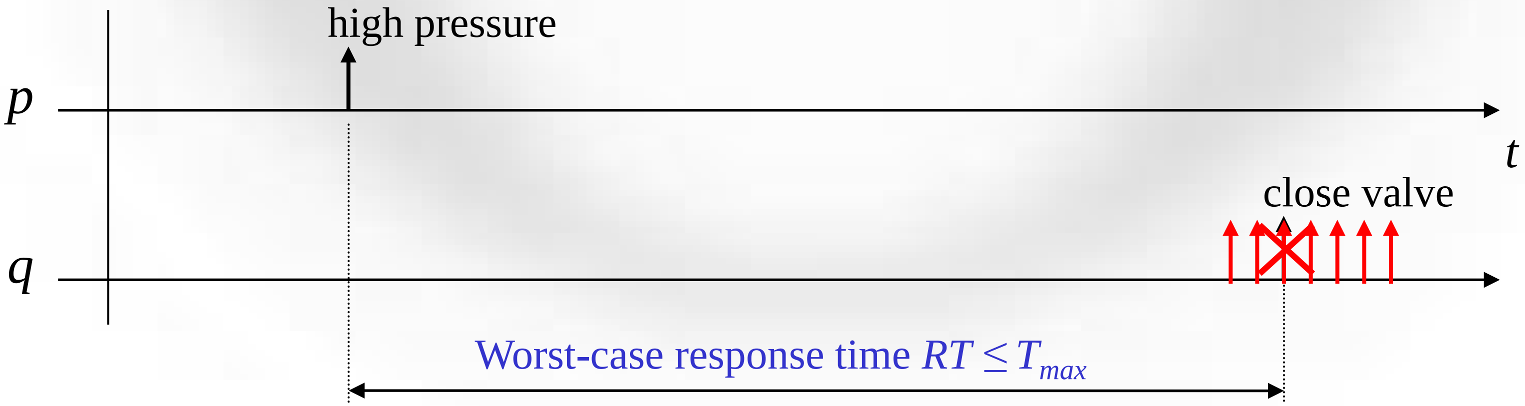
Network

Proc *q*

Valve

Real-time requirements

Partial failures



high pressure

$p$

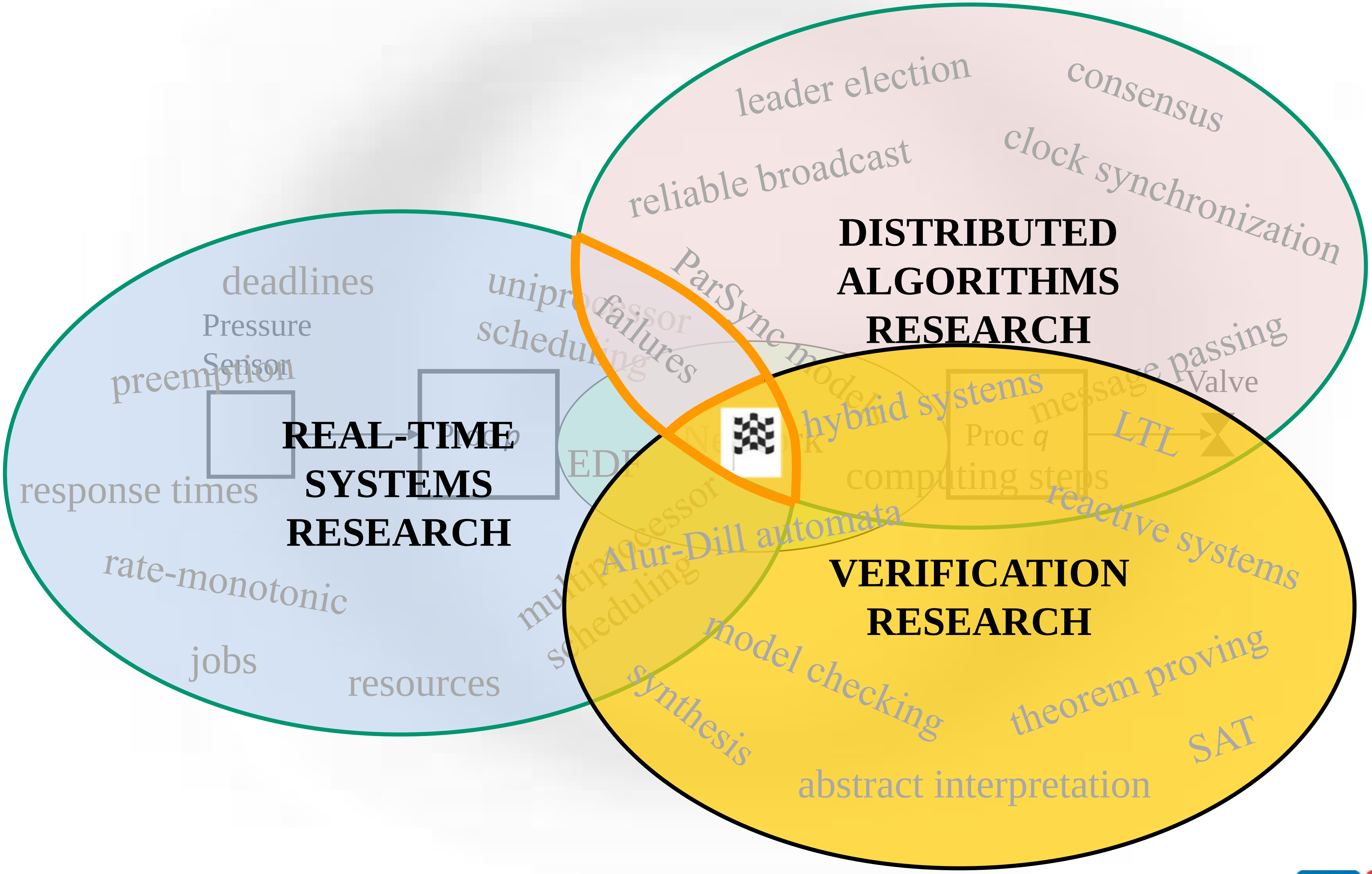
$t$

close valve

$q$

Worst-case response time $RT \leq T_{max}$

# Scattered Research



**DISTRIBUTED ALGORITHMS RESEARCH**

leader election

consensus

reliable broadcast

clock synchronization

message passing

Valve

ParSync model

**REAL-TIME SYSTEMS RESEARCH**

deadlines

Pressure Sensor

preemption

uniprocessor scheduling

Proc *p*

response times

rate-monotonic

jobs

resources

EDF

failures

Network

multiprocessor scheduling

**VERIFICATION RESEARCH**

hybrid systems

Proc *q*

LTL

computing steps

reactive systems

Alur-Dill automata

model checking

theorem proving

synthesis

SAT

abstract interpretation

# Motivation:
# Distributed Fault-Tolerant Clock Generation in Systems-on-Chip

DARTS

Institut für
Technische Informatik

Systems
Computing
Embedded

RUAG Aerospace Austria

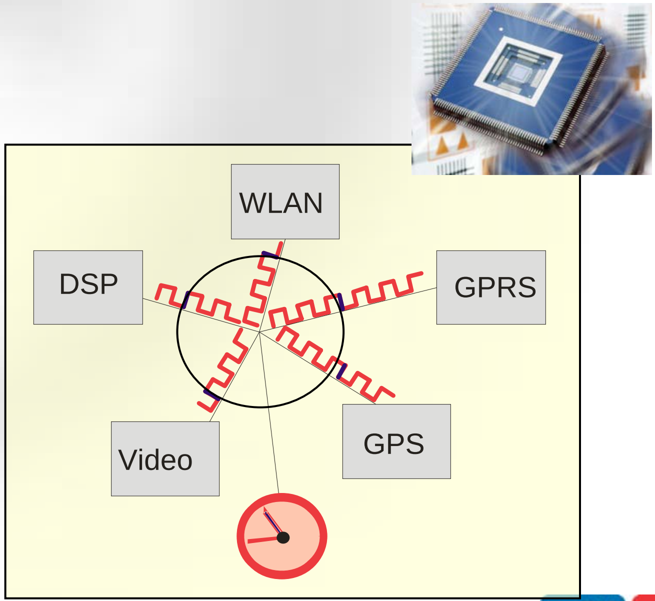
# Clocking in Systems-on-Chip (I)

## Classic synchronous paradigm

- ***Concept:*** Common notion of time for entire chip
- ***Method:*** Single crystal oscillator
  Global, phase-accurate clock tree

## Disadvantages

- Cumbersome clock tree design (physical limits!)
- High power consumption
- Clock is single point of failure!

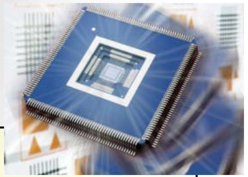


# Clocking in Systems-on-Chip (II)

## Alternative: DARTS clocks

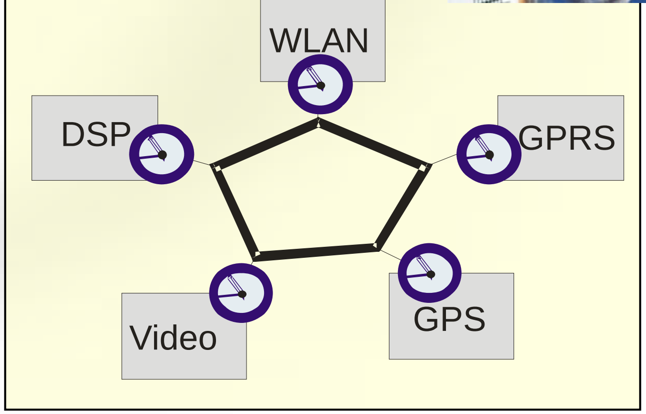
- ***Concept:*** Multiple synchronized tick generators
- ***Method:*** Distributed FT tick generation algorithm
  Implemented in (asynchronous) HW

*http://ti.tuwien.ac.at/ecs/research/projects/darts*



## Advantages

- Reasonable synchrony
- Uncritical clock distribution
- Clock is no single point of failure!



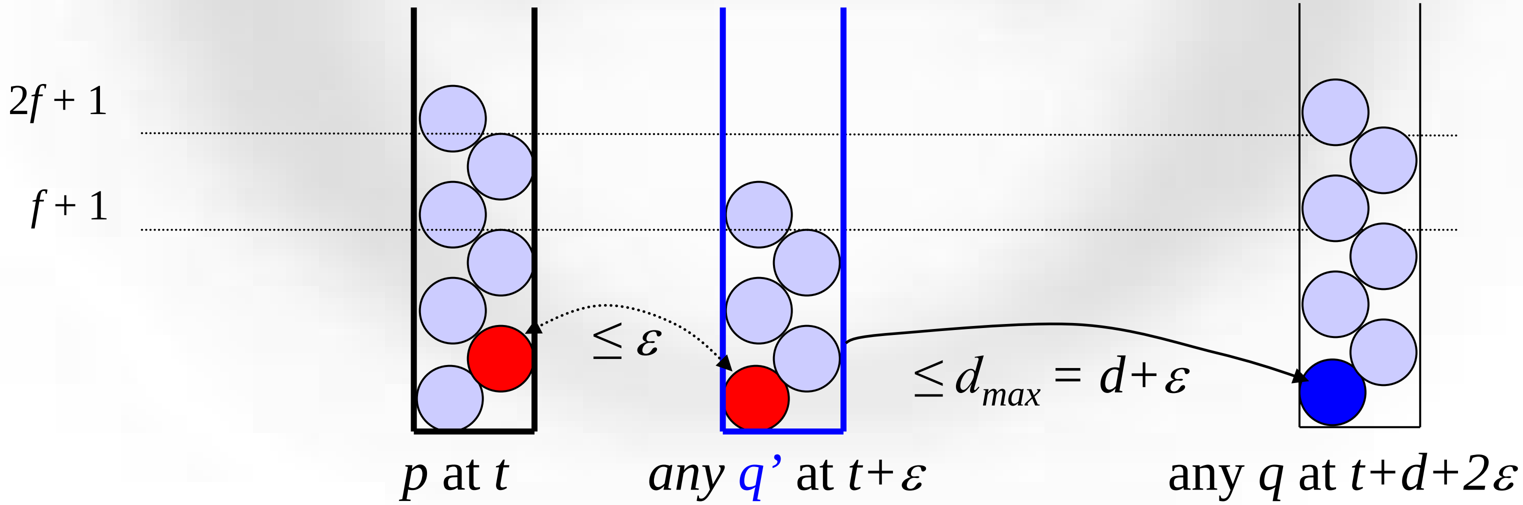
# The DARTS Distributed Algorithm

```
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 $n \geq 3f + 1$ and up to $f$ node failures, with (small) e-t-e delays $\in [d, d+\varepsilon]$:

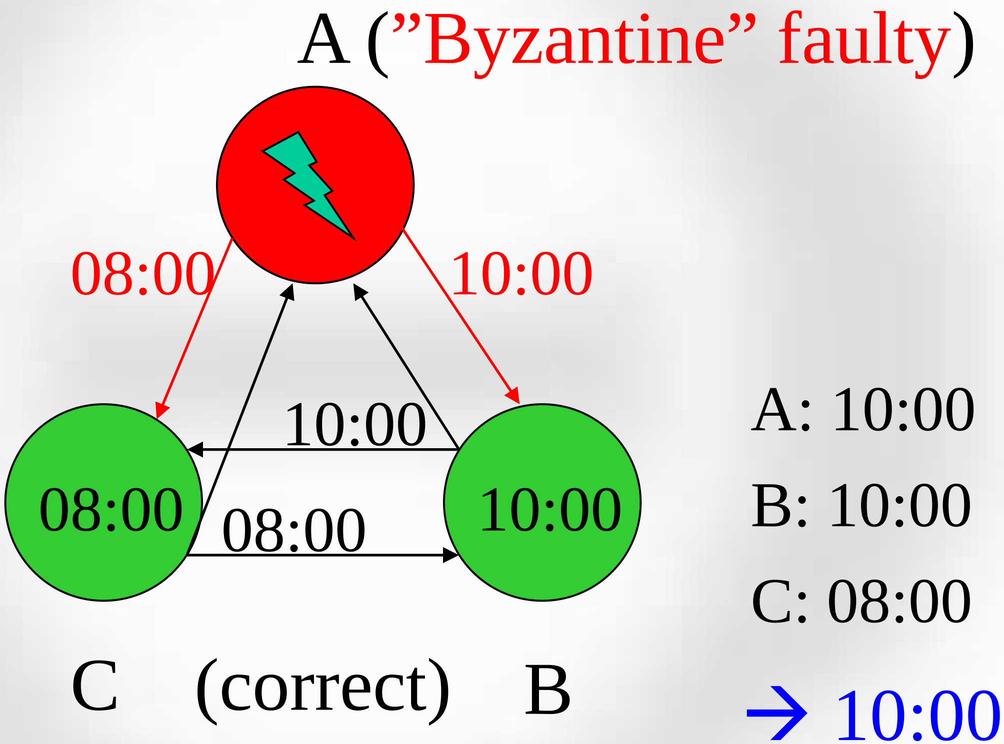
- Suppose node $p$ sends *tick*(C+1) at time $t$
- Then, node $q$ also sends *tick*(C+1) by time $t+d+2\varepsilon$

⇒ **Clock ticks occur approximately at the same time**



# $n \geq 3f+1$: Why do Failures hurt so much ?

**Toy example:**



A: 08:00

B: 10:00

C: 08:00

→ 08:00

A: 10:00

B: 10:00

C: 08:00

→ 10:00

- With this algorithm, B and C never get closer together
- Will prove: Majority $n = 2f + 1$ not enough for $f$ Byz. failures!

# DARTS Correctness Proofs

**Pipe Compare Signal Generators (PCSGs):** There exists a dedicated detection circuit for each pair of pipes which generates the status signals $GEQ^{o/e}_{p,q}(t)$ and $GR^{o/e}_{p,q}(t)$. In particular, $GEQ^{o}_{p,q}(t')$ becomes active (i.e.,

$GEQ^{o}_{p,q}$

previou

(i) $r^{se}_{p,}$

(ii) $[r^{r}_{p}$

Similarl

(i) $r^{self}_{p,q}(t) \in \mathbb{N}_{odd}$ a

**Definition 4.1.** (Direct Causality). *Let $I(t')$ and $O(t)$ be two events of some specific signal input and output, respectively, of a correct component $C$. Then $I(t')$ and $O(t)$ are* directly causally related, *denoted by $I(t') \to O(t)$, if*

*(i) they are*

*(ii) there is*

*i.e., $\nexists I'$*

**Theorem 4.13.** (Precision). *The precision $\pi \geq |b_q(t) - b_p(t)|$ of our algorithm is bounded by $\pi \leq \left\lfloor \frac{T_{sim}}{T^{-}_{first}} \right\rfloor + 1$.*

*Proof.* First of

established for

$k+1$, i.e., $t^p_k \leq$

$b^{max}(t')$

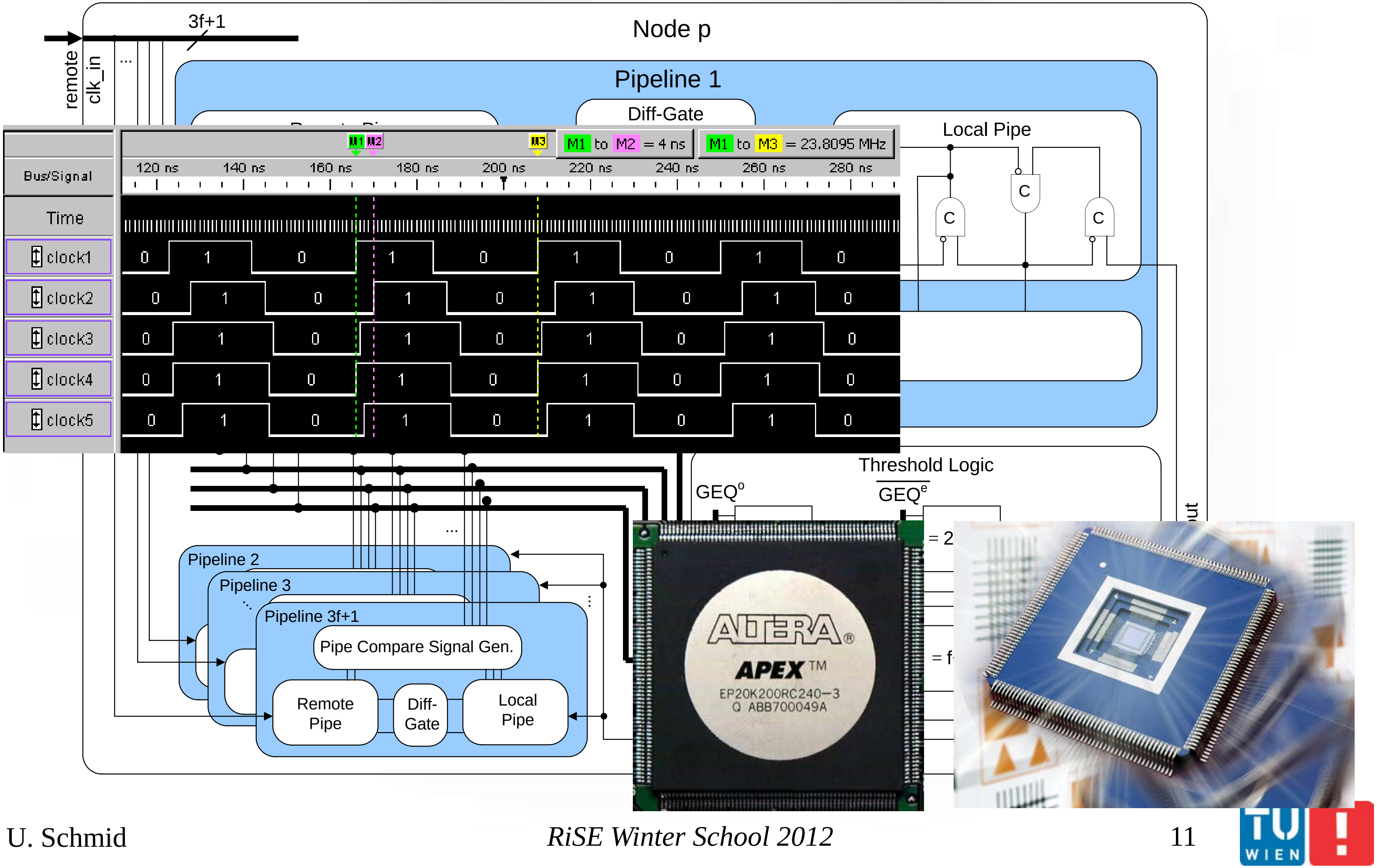
Assume that pr

**Theorem 4.14.** (Accuracy). *Given $\Delta = t_2 - t_1$, the accuracy $|b_p(t_2) - b_p(t_1)|$ of any correct process $p$ is bounded by* $\max\left\{0, \frac{\Delta - T_{sim} - T^{+}}{T^{+}}\right\} \leq |b_p(t_2) - b_p(t_1)| \leq \left\lceil \frac{\Delta}{T^{-}_{first}} \right\rceil + \min\left\{\pi + 1, \left\lceil \frac{\Delta}{D^{-}} - \frac{\Delta}{T^{-}_{first}} \right\rceil\right\}$.

*Proof.* The upper bound for accuracy will be shown first: It is known that $\forall t : b_p(t) \geq b^{max}(t) - \pi + (1 - I_{usync}(t))$ and $\forall t : b_p(t) \leq b^{max}(t)$ from Lemma 4.13 and Lemma 4.11. Thus $b_p(t_2) - b_p(t_1) \leq b^{max}(t_2) - b^{max}(t_1) + \pi - (1 - I_{usync}(t_1))$. By applying Lemma 4.11, $b_p(t_2) - b_p(t_1) \leq \left\lfloor \frac{t_2 - t_1}{T^{-}_{first}} \right\rfloor + 2I_{usync}(t_1) - 1 + \pi \leq \left\lfloor \frac{t_2 - t_1}{T^{-}_{first}} \right\rfloor + \pi + 1 \leq \left\lceil \frac{t_2 - t_1}{T^{-}_{first}} \right\rceil + \pi + 1$. Moreover, from Lemma 4.7 it follows that $b_p(t_2) - b_p(t_1) \leq \lceil \frac{t_2 - t_1}{D^{-}} \rceil$. Hence, $b_p(t_2) - b_p(t_1) \leq \min\left\{\left\lceil \frac{\Delta}{T^{-}_{first}} \right\rceil + \pi + 1, \lceil \frac{\Delta}{D^{-}} \rceil\right\} \leq \left\lceil \frac{\Delta}{T^{-}_{first}} \right\rceil + \min\left\{\pi + 1, \lceil \frac{\Delta}{D^{-}} \rceil - \left\lceil \frac{\Delta}{T^{-}_{first}} \right\rceil\right\} \leq \left\lceil \frac{\Delta}{T^{-}_{first}} \right\rceil + \min\left\{\pi + 1, \left\lceil \frac{\Delta}{D^{-}} - \frac{\Delta}{T^{-}_{first}} \right\rceil\right\}$ since $\lceil x + y \rceil \leq \lceil x \rceil + \lceil y \rceil$.

To prove the lower bound, first define $b_1 = b_p(t_1)$, $b_2 = b_p(t_2)$ and $t^p_{b_1} \leq t_2$, $t^p_{b_2} \leq t_2$ as the points in time when $p$ sends tick $b_1$ and $b_2$. Clearly $t^p_{b_2+1} > t_2$,

# DARTS Implementation



M1 to M2 = 4 ns

M1 to M3 = 23.8095 MHz

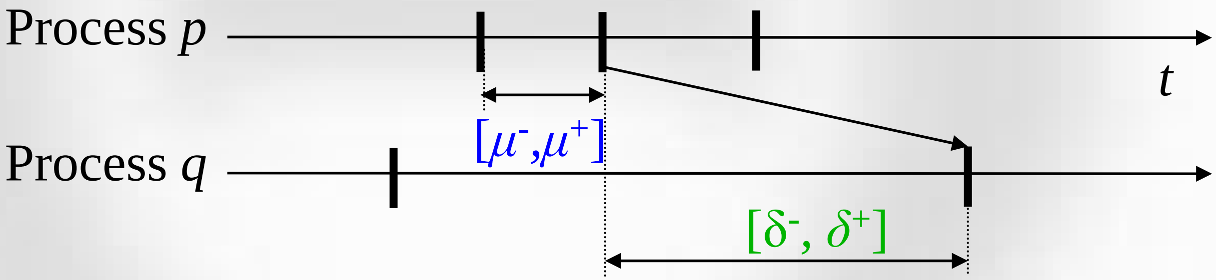
# Introduction to Distributed Algorithms

# Content (Part 1)

- Basics:
    - Distributed Computing Model
    - Synchrony and Fault-Tolerance
    - Correctness Proofs
- Some Appetizers:
    - Consistent Broadcasting
    - Consensus
- Food for Thoughts

# Classic Modeling and Analysis

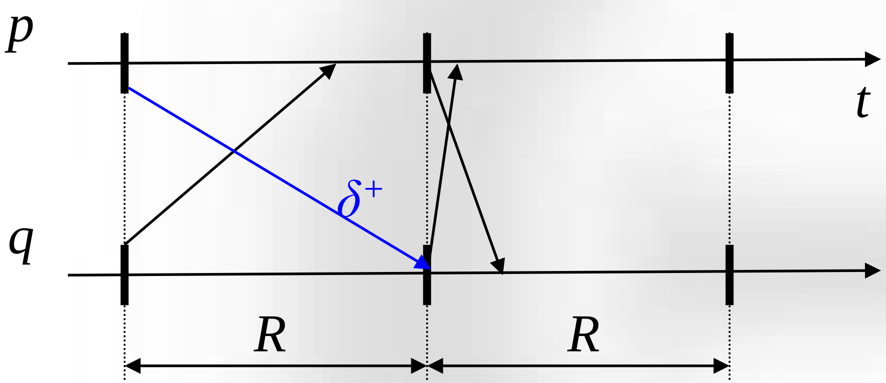
- Processors/processes modeled as interacting state machines
- **Zero-time** atomic computing steps, usually time-triggered
  - Message Passing (MP): [receive] + compute + [send]
  - Shared Memory (SHM): [accessSHM] + compute



- System timing parameters:
  - Operation durations modeled via **inter-step times $\in[\mu^-,\mu^+]$** (often $\mu^- = 0$)
  - Message delays modeled as **end-to-end delays $\in[\delta^-, \delta^+]$** (often $\delta^- = 0$)

# Synchrony Models: 2 Extremes …

## Lock-step synchronous systems



- Computing step times:

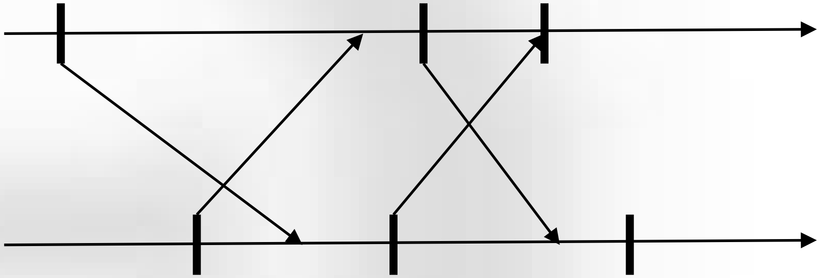
  $\mu^- = \mu^+ = R$

- Message delays

  $0 \leq \delta^- \leq \delta^+ \leq R$

- Perfectly synchronized rounds

## Asynchronous systems



- Computing step times:
  - $\mu^- = 0$
  - $\mu^+$ finite (but unbounded)
- Message delays
  - $\delta^- = 0$
  - $\delta^+$ finite (but unbounded)

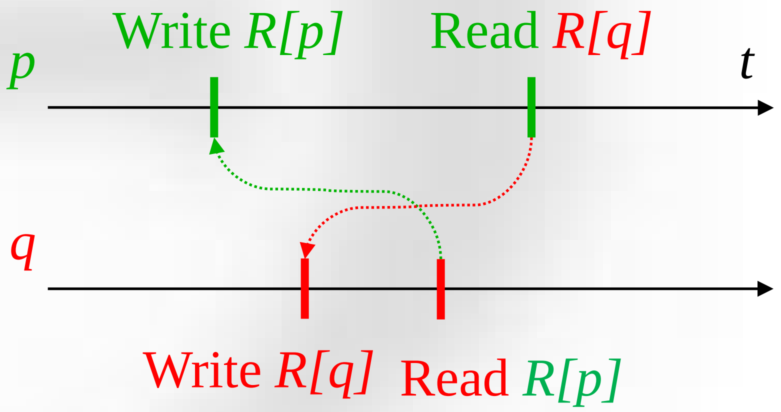
# Failure Models

- „Deterministic“ failure models
  - At most *f* of *n* processors in the system may fail
  - Correct processes do not a priori know who has failed and when and how
- Failure semantics ranging from
  - Crash failures: Processors stop operating, possibly within a step
  - Byzantine failures [LSP82]: Processors can do what they want
- Real processors etc. fail probabilistically → Coverage analysis
- Restrict our attention to message passing systems here:
  - Typically fully connected, with dedicated links between every pair of processors
  - [Communication between correct processes typically considered reliable]

# A Note on Message Passing vs. Shared Memory

- MP can always be simulated in a SHM system
- The opposite is not generally true:
  - AsyncSHM can be simulated in AsyncMP if a majority of processes ($n > 2f$) is correct
  - Not the case for $n \leq 2f$ → AsyncSHM more powerful than AsyncMP
- MP is more elementary than SHM!

- E.g.: Wait-free ($f = n$-1) event ordering possible in AsyncSHM but not in AsyncMP



$p$ knows by the time of its Read whether $q$ has already done its Write

# Correctness Proofs

- Global state transitions
  - Configuration $C$ = vector of processor local states [+ in-transit messages for MP]
  - State transition = result of a single processor taking a step
- Algorithm vs. Adversary
  - Adversary determines which and when events $\varphi$ (like processor $p_i$ takes a step) happen (→ Async. systems: Adv. subject to admissibility (fairness) conditions)
  - Algorithm determines what actually happens in the corresponding step
- Executions and traces
  - Execution $E$ = sequence of configurations alternating with events $C_0,\varphi_1,C_1,\varphi_2,C_2,\varphi_3,C_3, \ldots$
  - Trace $T$ = (sub-)sequence of „interesting" events (or states)
- Correctness proofs: Set of generated traces satisfies
  - Safety properties („something bad never happens")
  - Liveness properties („something good eventually happens")

# Some Appetizers

# Consistent Broadcasting

# Consistent Broadcasting [ST87]

- Want to build **authenticated reliable broadcasting**:
  - Any process $p_s$ may have some message $m_s$ to broadcast: **bcast**($p_s$,$m_s$)
  - Every correct process shall eventually call **accept**($p_s$,$m_s$), and shall be sure that the received $m_s$ originates in $p_s$
  - Do not use real authentication (cryptography)!

- Very useful primitive:
  - Clock synchronization
  - Consensus
  - etc.

# Properties Consistent Broadcasting

## Time-free specification:

- **Correctness:** If a correct processor $p_s$ executes **bcast**($p_s$,$m_s$), then every correct processor eventually calls **accept**($p_s$,$m_s$)
- **Unforgeability:** If a correct processor $p_s$ never executes **bcast**($p_s$,$m_s$), then no correct processor ever calls **accept**($p_s$,$m_s$)
- **Relay:** If some correct processor calls **accept**($p_s$,$m_s$), then every other correct processor eventually also calls **accept**($p_s$,$m_s$)

# Implementation

**bcast**$(p_s,m_s)$ at $p_s$

```
send ($init,p_s,m_s$) to all processors
```

**accept**$(p_s,m_s)$ at every $p_i$

```
if got ($init,p_s,m_s$) from $p_s$
    → send ($echo,p_s,m_s$) to all [once]
if got ($echo,p_s,m_s$) from $f + 1$
    → send ($echo,p_s,m_s$) to all [once]
if got ($echo,p_s,m_s$) from $2f + 1$
    → call accept($p_s,m_s$)
```

## System model:

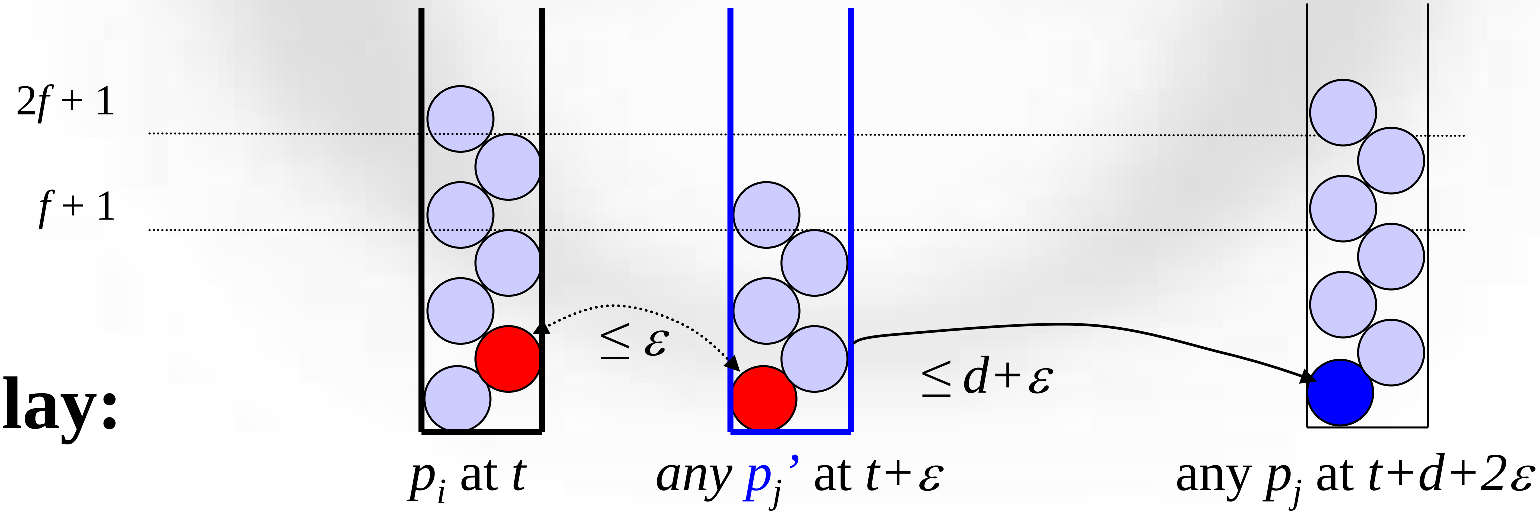
- At most $f$ Byzantine faulty processors
- $n \geq 3f + 1$
- E-t-e delays $\in [d,d+\varepsilon]$:

- Message sent by correct proc at $t$ got by correct receiver proc within $[t+d,t+d+\varepsilon]$
- Every proc gets at most $f$ faulty echo/init messages from different procs
- At most $f$ echo messages available at $p_i$ by $t$ could be missing at $p_j$ by $t + \varepsilon$

# Correctness Proof (Time-dependent Version)

- **Correctness:** If a correct proc $p_s$ executes **bcast**($p_s$,$m_s$) by $t$, then every correct processor eventually calls **accept**($p_s$,$m_s$) by $t$+2($d$+$\varepsilon$)
- **Unforgeability:** If a correct proc $p_s$ does not execute **bcast**($p_s$,$m_s$) by $t$, then no correct processor calls **accept**($p_s$,$m_s$) by $t$+2$d$
- **Relay:** If a correct processor calls **accept**($p_s$,$m_s$) at $t$, then every other correct processor also calls **accept**($p_s$,$m_s$) by $t+d+2\varepsilon$

**Relay:**

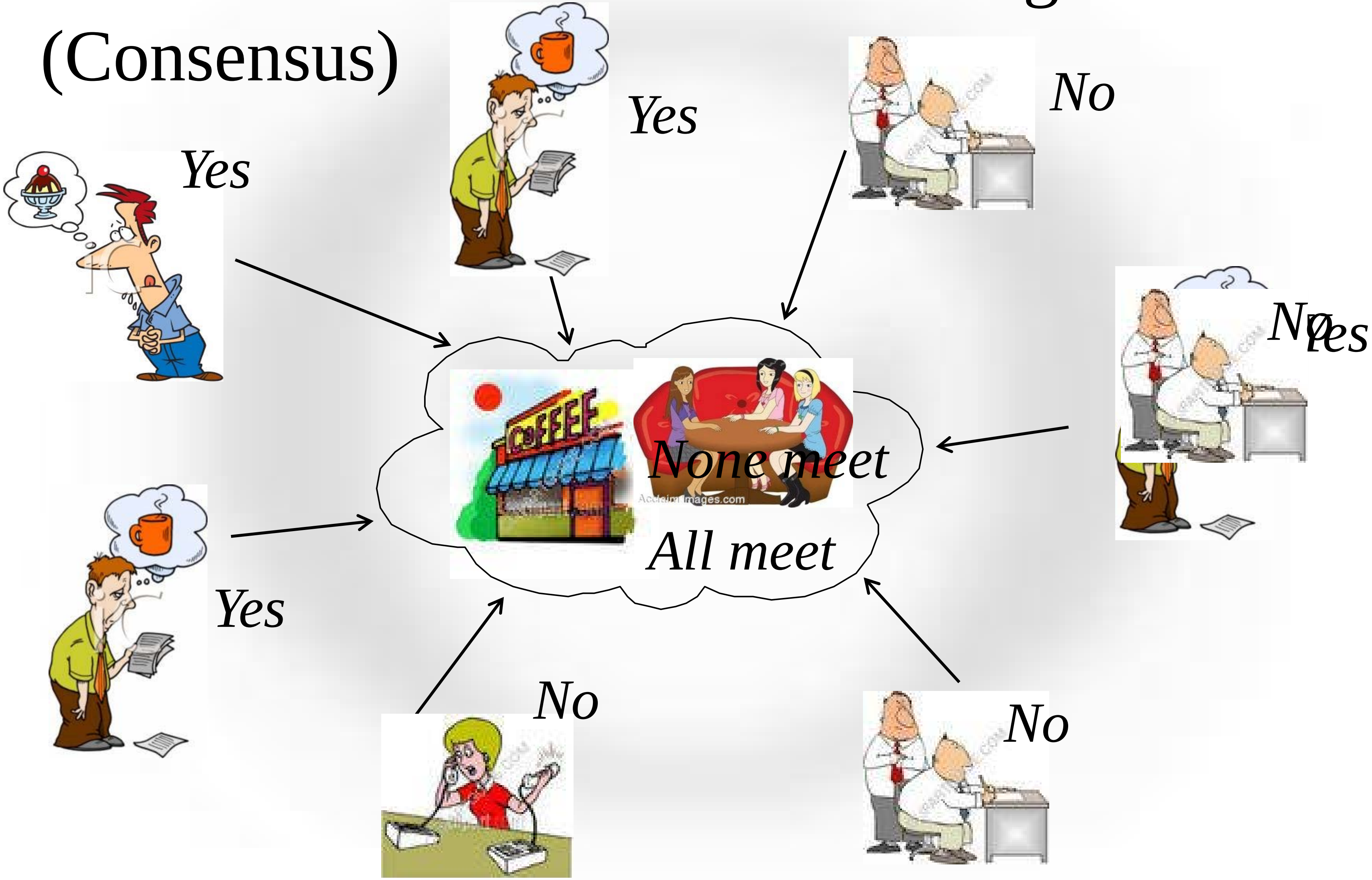


# Verification Challenges

- Typical distributed algorithms proofs are „handwaving“, compared to verification standards
- Try do make it rigorous is challenging, even for simple problems like CB:
  - Parameterization ($n$, $f$)
  - Asynchronous systems
  - Failures
- We are working on this in the context of RiSE …

# Consensus

# A Classic Problem: Distributed Agreement (Consensus)



*Yes*

*No*

*Yes*

*No*

*Yes*

*None meet*

*All meet*

*Yes*

*No*

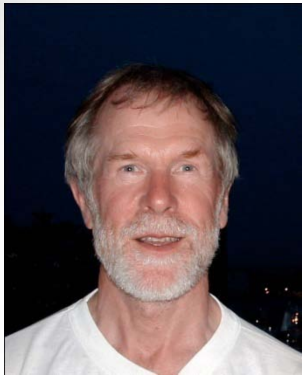
*No*

# Consensus Properties

- Every process $p_i$
  - has initial value $x_i$ chosen from some finite set $V$
  - shall irrevocably decide on output value $y_i$

- **Termination:** Every correct processor eventually decides
- **Agreement:** Every two correct processors $p_i$ , $p_j$ decide on the same value $y_i = y_j$
- **Validity:** If all correct processors have the same input value $x$, then $x$ is the only possible decision value

# Asynchronous Consensus Impossibility

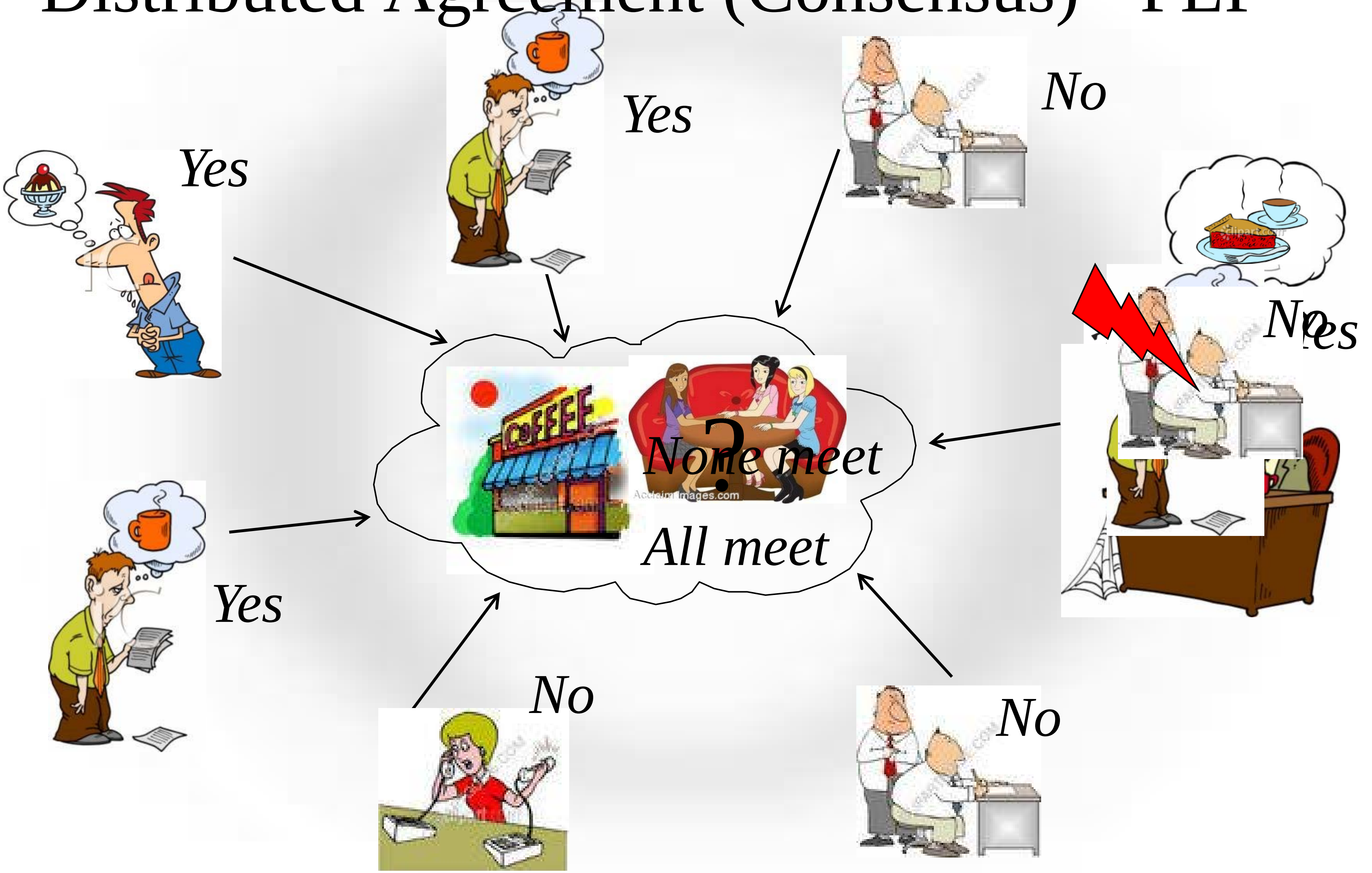


Fischer, Lynch and Paterson [FLP85]:

> *"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!

# Distributed Agreement (Consensus) - FLP



*Yes*

*Yes*

*No*

*No*
*Yes*

*None meet*

?

*All meet*

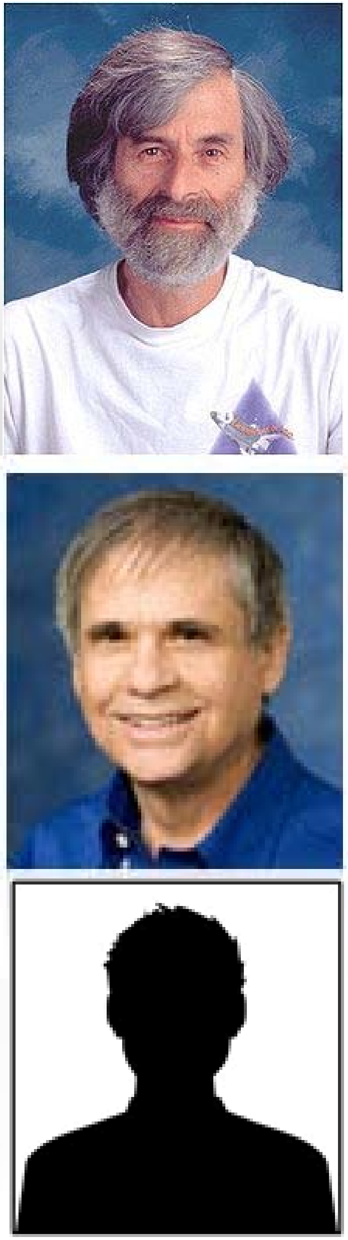
*Yes*

*No*

*No*

# Synchronous Consensus

Lamport, Shostak and Pease [LSP82]:



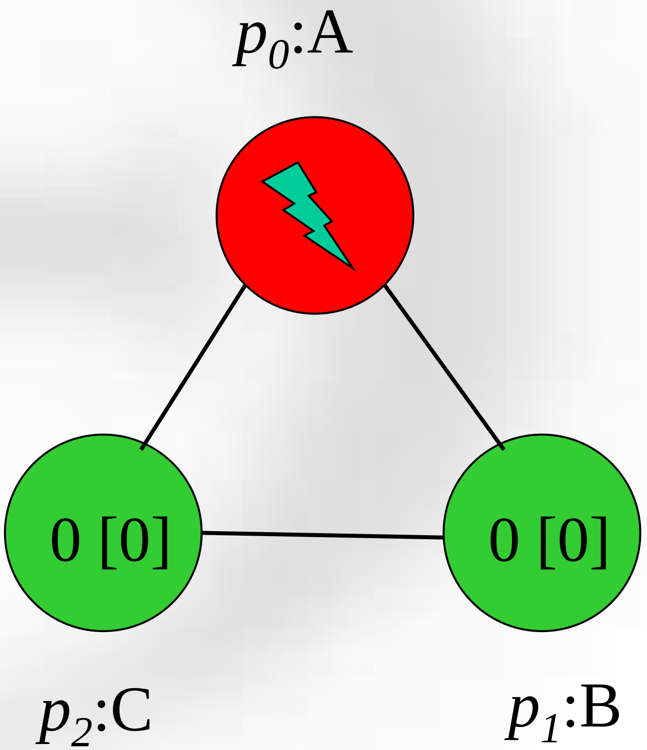
> *"There is a deterministic algorithm for solving consensus in a synchronous distributed system of $n \geq 3f+1$ processors in the presence of at most $f$ Byzantine failures."*

But:

It is impossible to solve consensus if $n = 3f$ !

# Impossibility of Consensus for $f = 1$, $n = 3$

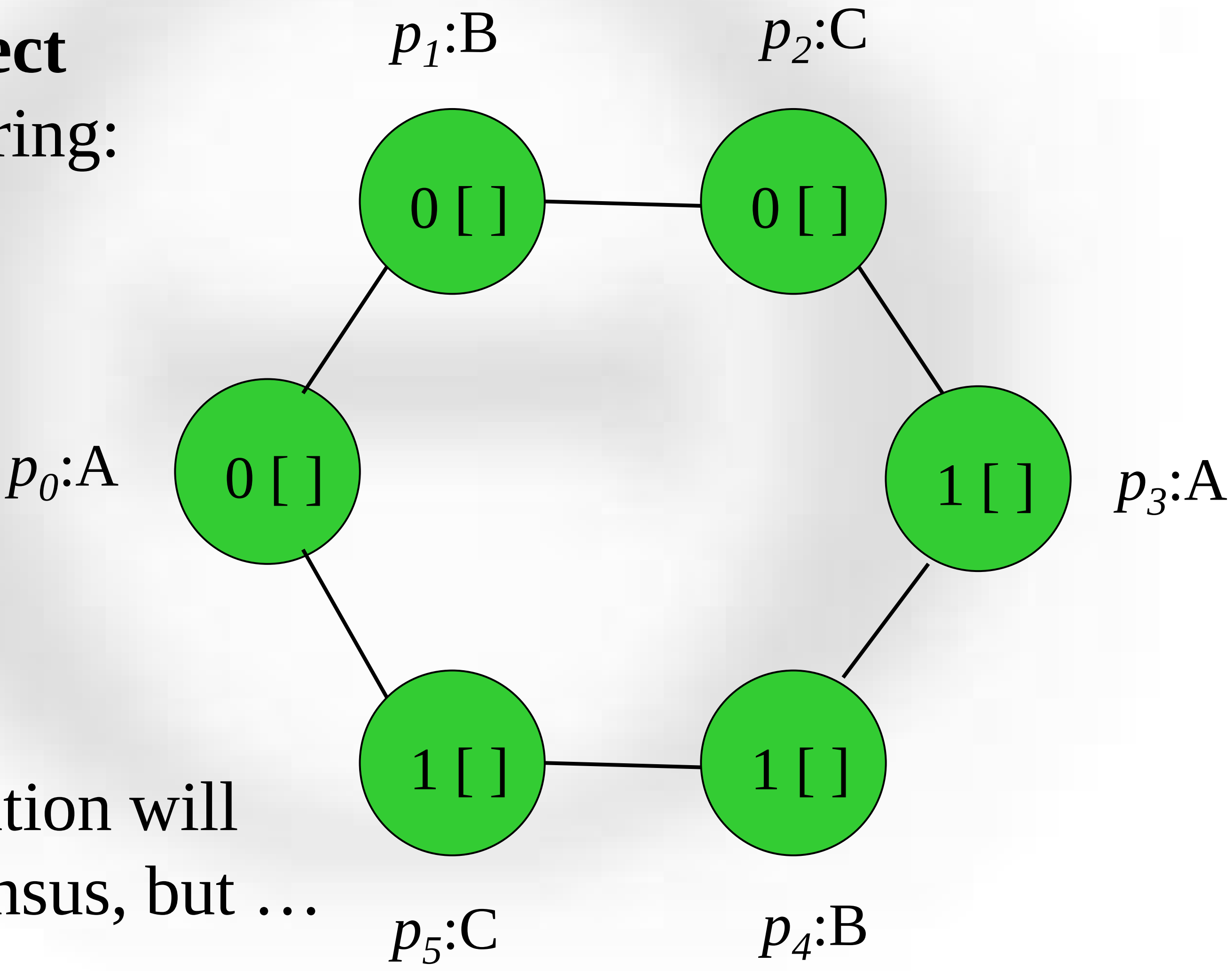
- Suppose correct algorithm $\mathcal{A} = (A,B,C)$ for $(p_0,p_1,p_2)$ existed



- Assume $p_0$ faulty
- By Validity:
  - $x_1 = x_2 = 0 \rightarrow y_1 = y_2 = 0$
  - $x_1 = x_2 = 1 \rightarrow y_1 = y_2 = 1$
- By Agreement:
  - $x_1 \neq x_2 \rightarrow y_1 = y_2$

# „Easy Impossibility Proofs“ [FLM86] (I)

Arrange 6 **correct** processors in a ring:



$p_1$:B

$p_2$:C

0 [ ]

0 [ ]

$p_0$:A

0 [ ]

1 [ ]

$p_3$:A

1 [ ]

1 [ ]

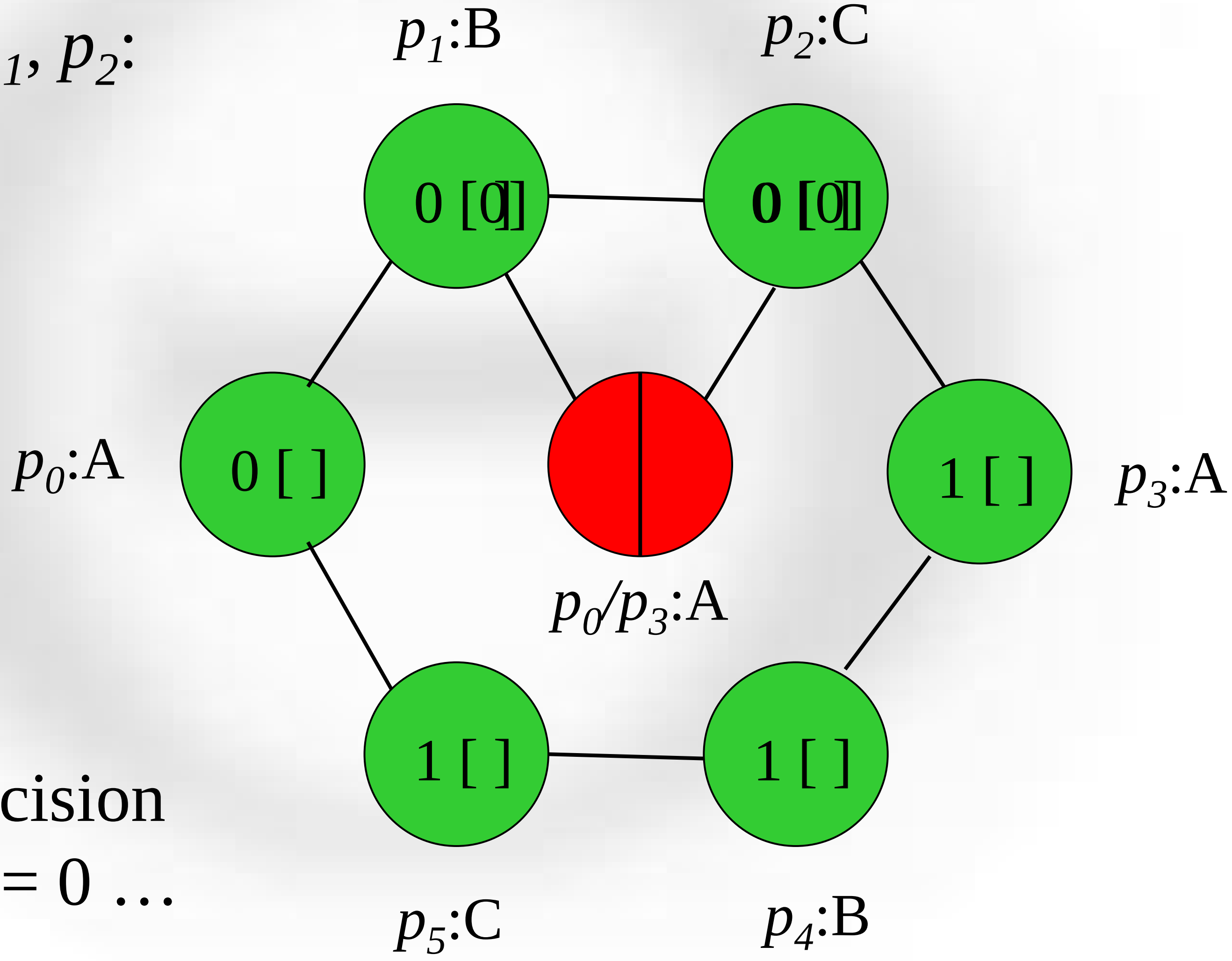
$p_5$:C

$p_4$:B

Resulting execution will not solve consensus, but …

# „Easy Impossibility Proofs“ [FLM86] (II)

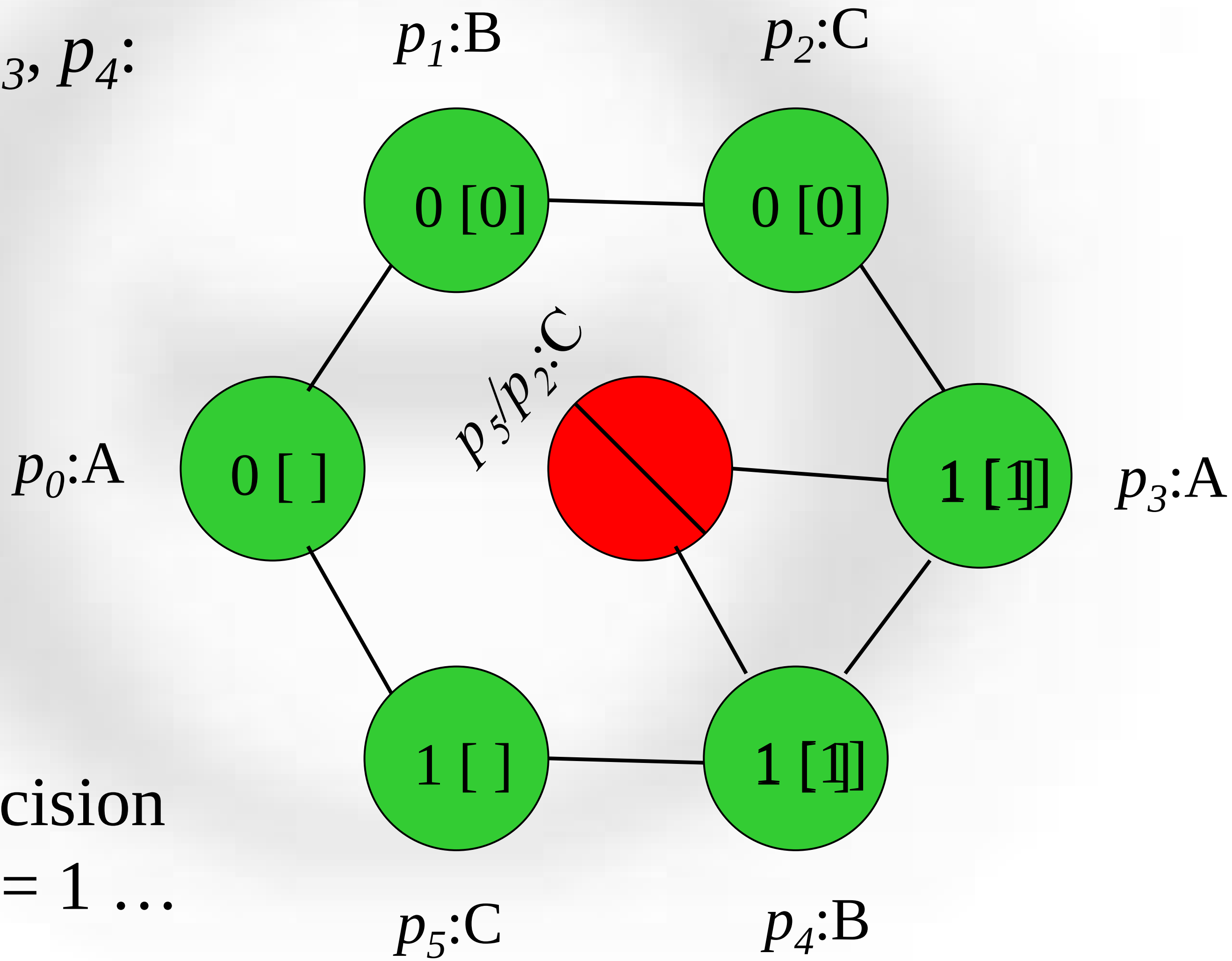
Local view of $p_1$, $p_2$:



By Validity: Decision must be $y_1 = y_2 = 0$ …

# „Easy Impossibility Proofs“ [FLM86] (III)

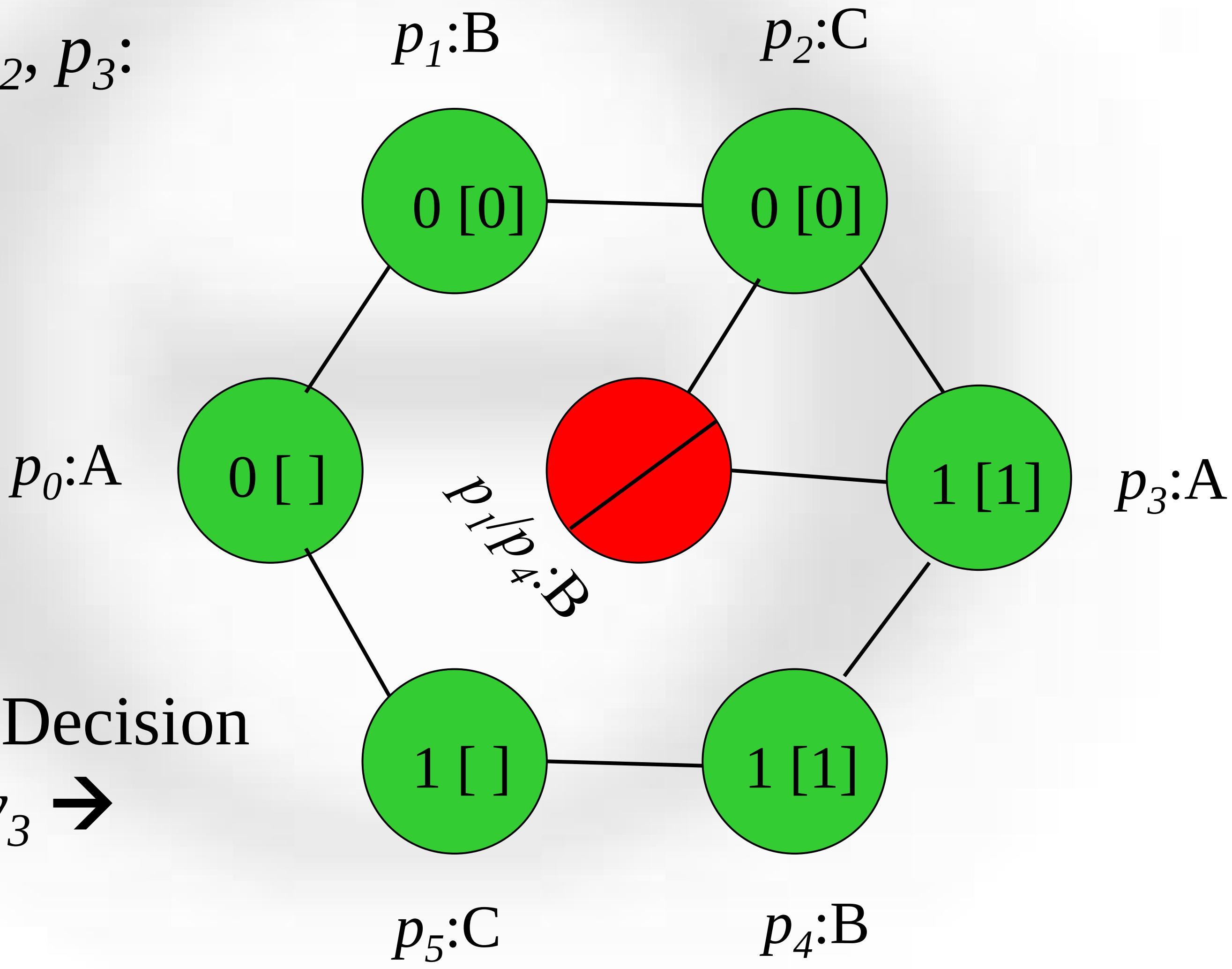
Local view of $p_3$, $p_4$:



By Validity: Decision must be $y_3 = y_4 = 1$ …

# „Easy Impossibility Proofs“ [FLM86] (IV)

Local view of $p_2$, $p_3$:

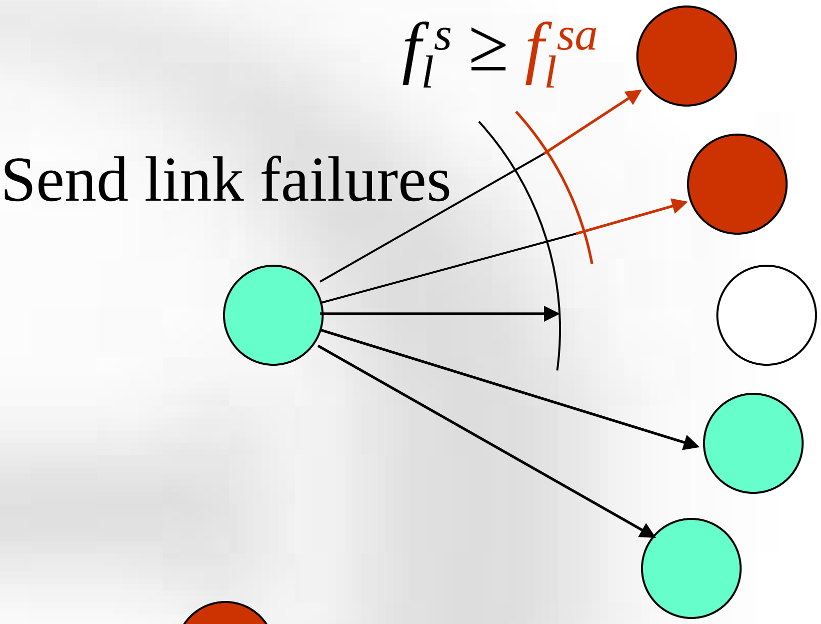


By Agreement: Decision should be $y_2 = y_3$ → **Contracdicion**

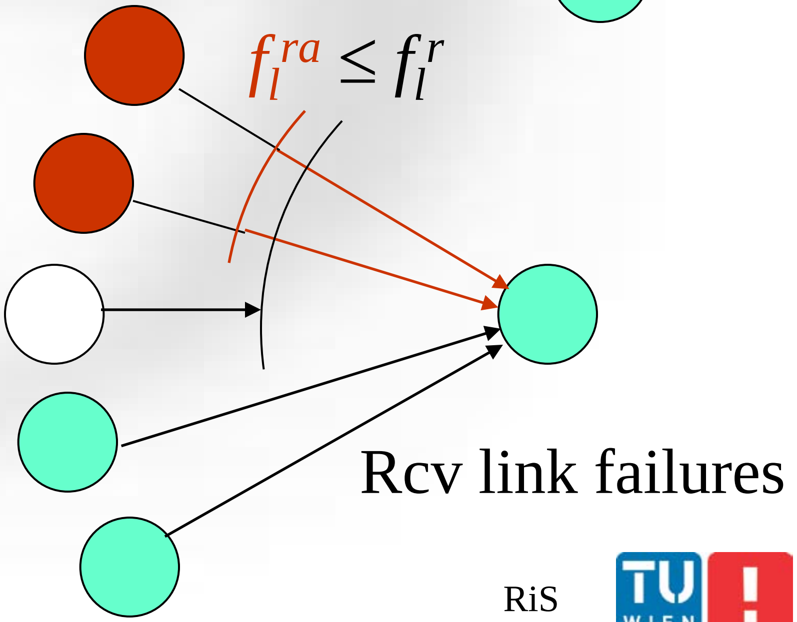
# Food for Thoughts

# 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 $p_s$
    → send $(echo, p_s, m_s)$ to all [once]
if got $(echo, p_s, m_s)$ from $S f_l^{sa} + R f_l^{ra} + f + 1$
    → send $(echo, p_s, m_s)$ to all [once]
if got $(echo, p_s, m_s)$ from $S' f_l^{sa} + R' f_l^{ra} + 2f + 1$
    → call accept$(p_s, m_s)$
```

Required number of procs:

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

Recall lower bound:

- $n \geq f_l^r + f_l^{ra} + f_l^s + f_l^{sa} + 3f + 1$

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 $f = 0$)

# The End

## (Part 1)

# 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