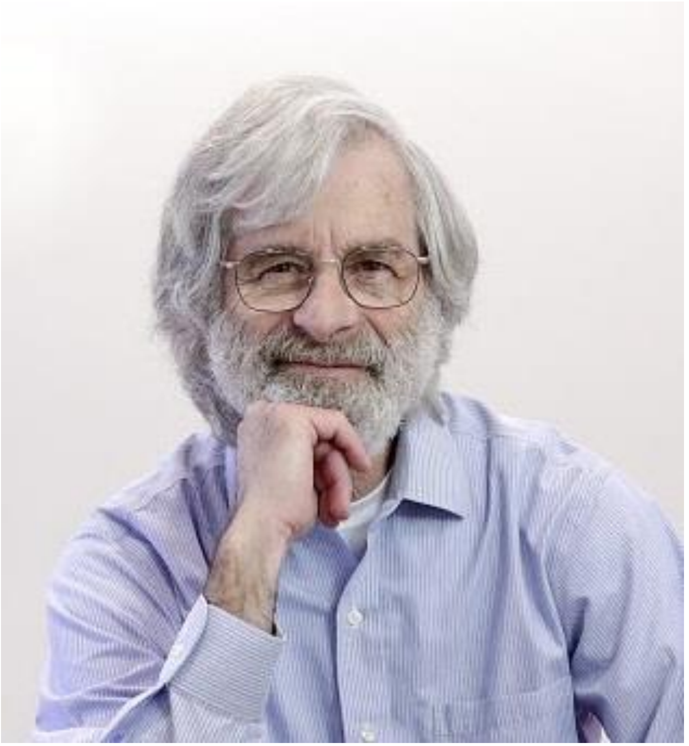
Round model for dist. algo.
From verification to implementation

Cezara Drăgoi
Thomas A. Henzinger
Josef Widder
Damien Zufferey

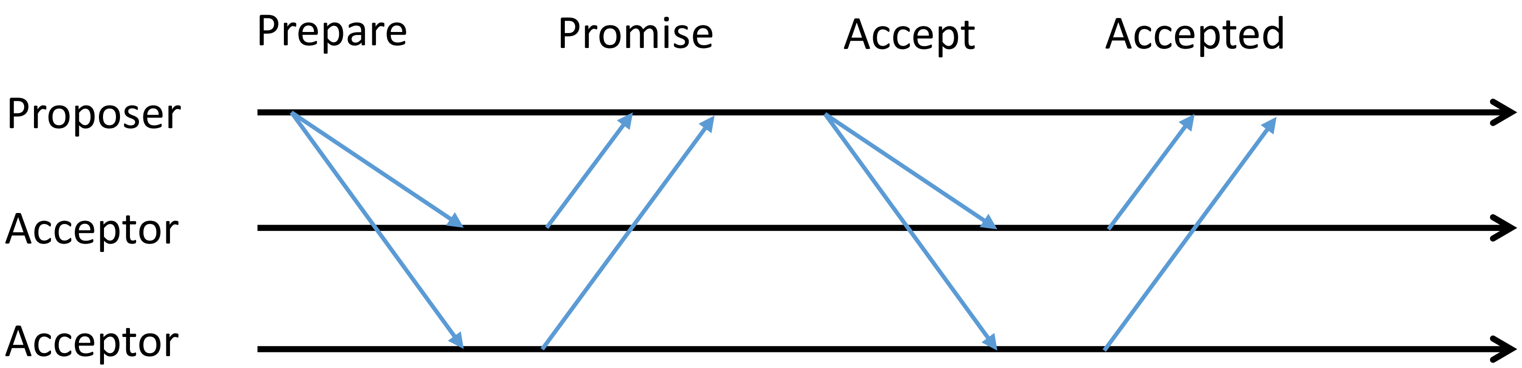
FRIDA, 24.07.2014, Vienna

Our journey starts on the island of Paxos …

… where archeologists made an interesting discovery about a parliament system …



The Paxos Algorithm [Lamport 98]

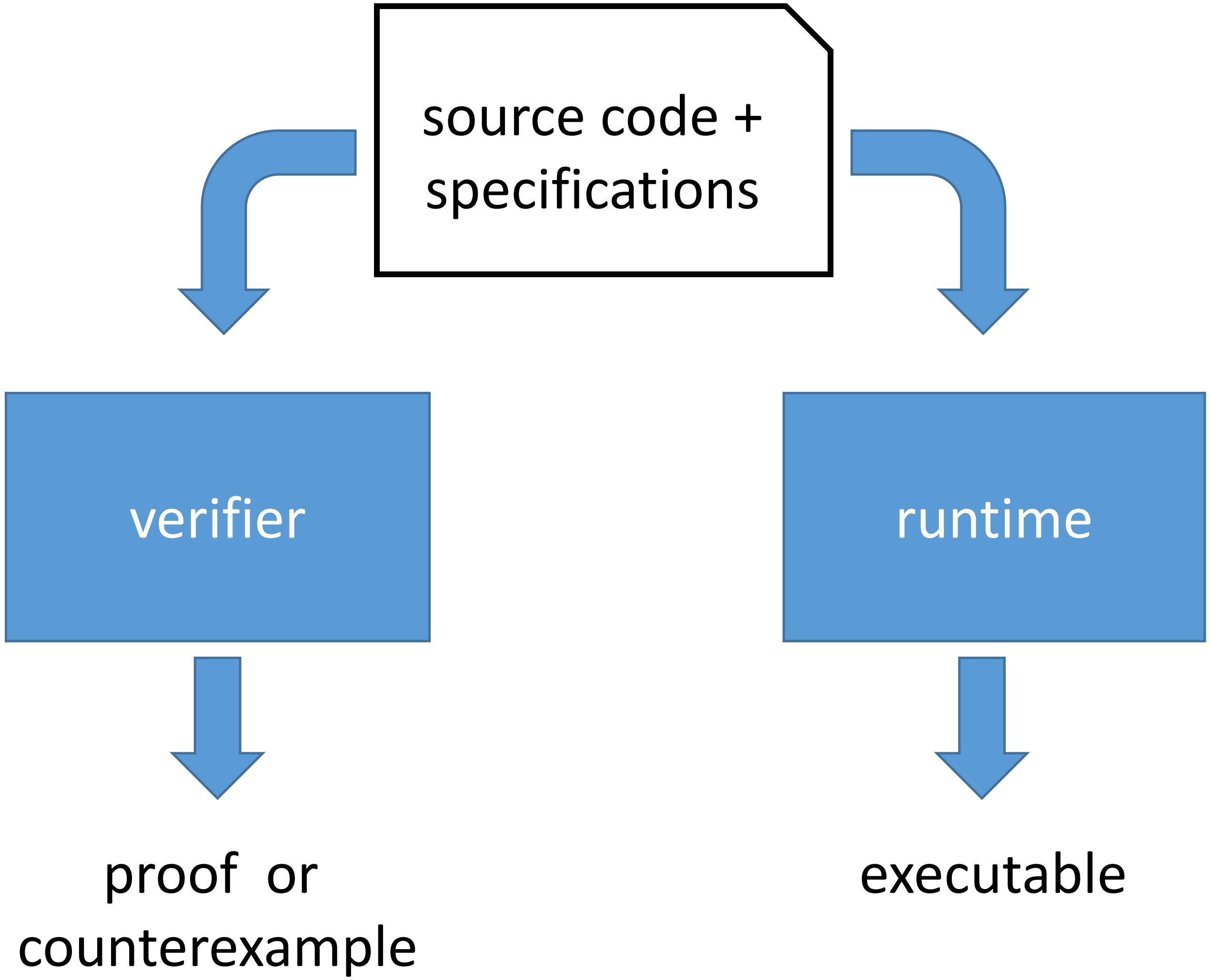


Used at Google (Chubby), Yahoo/Apache (Zookeeper), Microsoft (Autopilot)

What verification can do ?

- Hard to implement and get right
  - “The fault-tolerance computing community has not developed the **tools to make it easy to implement** their algorithms.” [Chandra et al. 07]
  - “The fault-tolerance computing community has **not paid enough attention to testing**, a key ingredient for building fault-tolerant systems.” [Chandra et al. 07]
  - ⇒ use for formal verification
- Gap between the theory community and the system community
  - “In order to build a real-world system, an expert needs to use numerous ideas scattered in the literature and make several relatively small protocol extensions. The cumulative effort will be substantial and the **final system will be based on an unproven protocol**.” [Chandra et al. 07]
  - ⇒ use for automated verification

Our goals

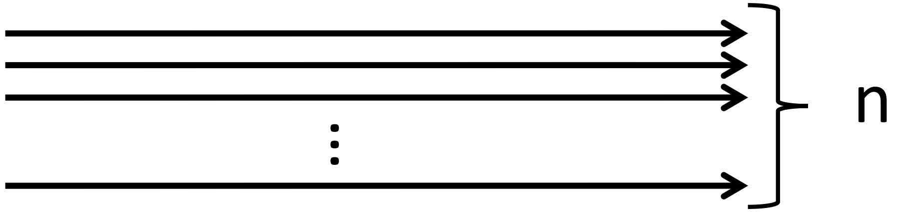

source code +
specifications
verifier
runtime
proof or
counterexample
executable

Our goals

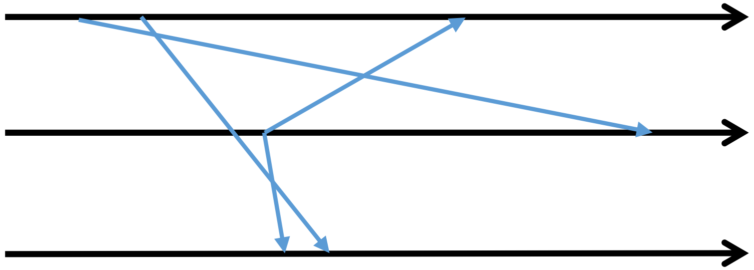
- Language for verified fault tolerant distributed algorithms implementation
  - Algorithms in isolation (as published)
  - Algorithms as part of a bigger system (modified to fit purpose)
- Use the HO model to simplify the reasoning
  - User provides an inductive invariants, then push button
  - Need a very expressive logic for automation (Cezara’s talk)
- Provide an implementation that performs well enough
  - Show that the overhead of rounds is acceptable

Verification Challenges

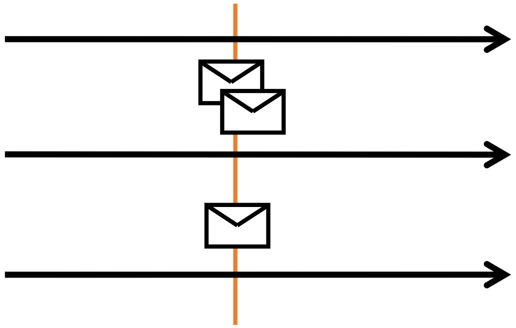
- Parametric systems



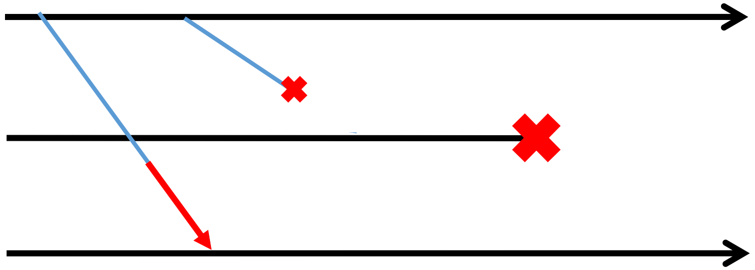
- Asynchrony (Interleaving, delays)



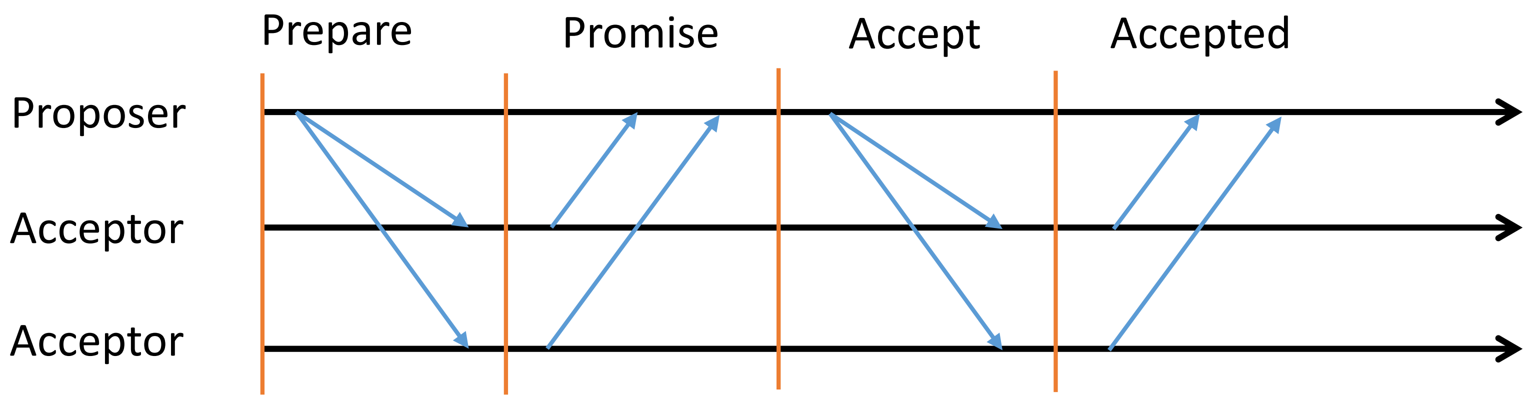
- Channels



- Faults



Communication-closed Rounds

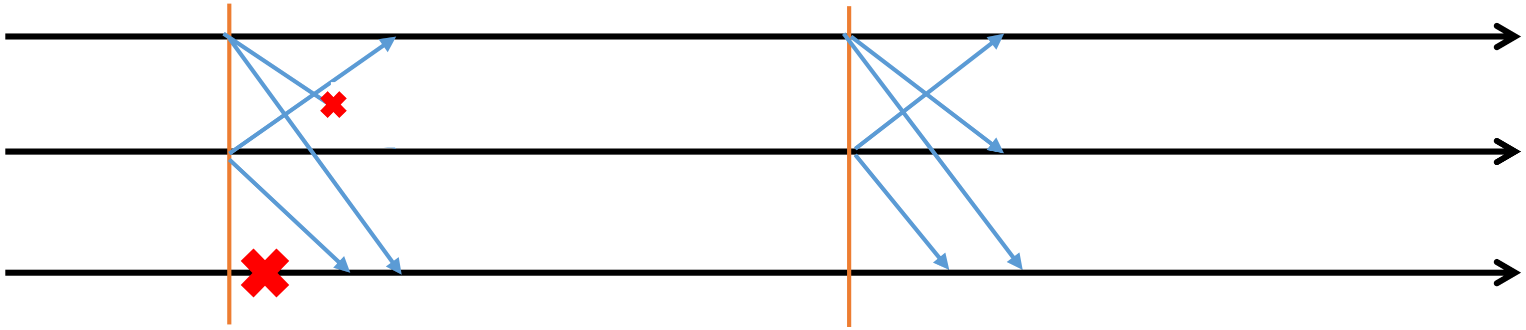


[Elrad & Francez 82]: decomposition of algorithm in communication-closed rounds.

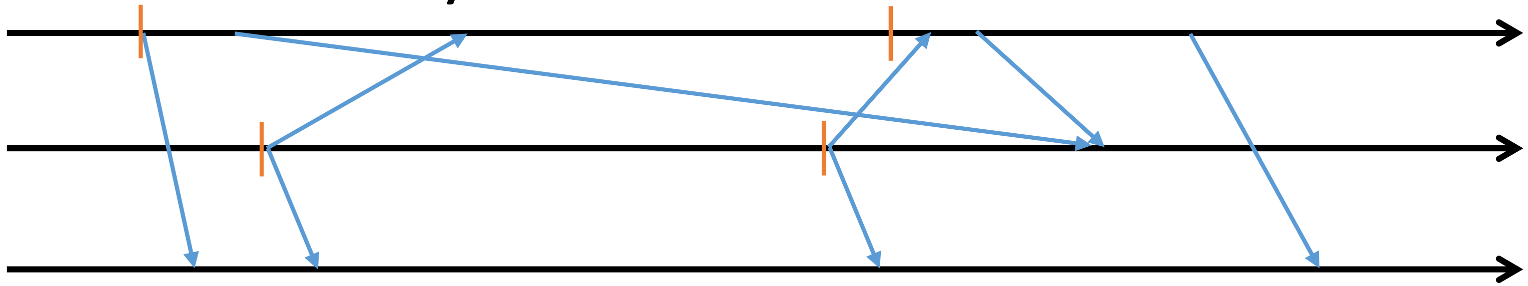
[Dwork & Lynch & Stockmeyer, 88] defines round model for non-synchronous models: partial synchrony

Mapping asynchrony to faults

Reasoning in round-based model (partial synchrony)



Implementation in an asynchronous world



The Heard-Of model [Charron-Bost & Schiper 09]

- $p \in HO(q, r)$: message send by *p* to *q* at round *r* is delivered
- Maps every faults to message faults
  - A crashed process is the same as a process whose messages are dropped.
  - Byzantine faults can be simulated altering messages
  - Simplify the proofs: does not need to case split on (in)correct processes
  - Handling transient/permanent faults is transparent at the algorithm level
- However, in practice, ...

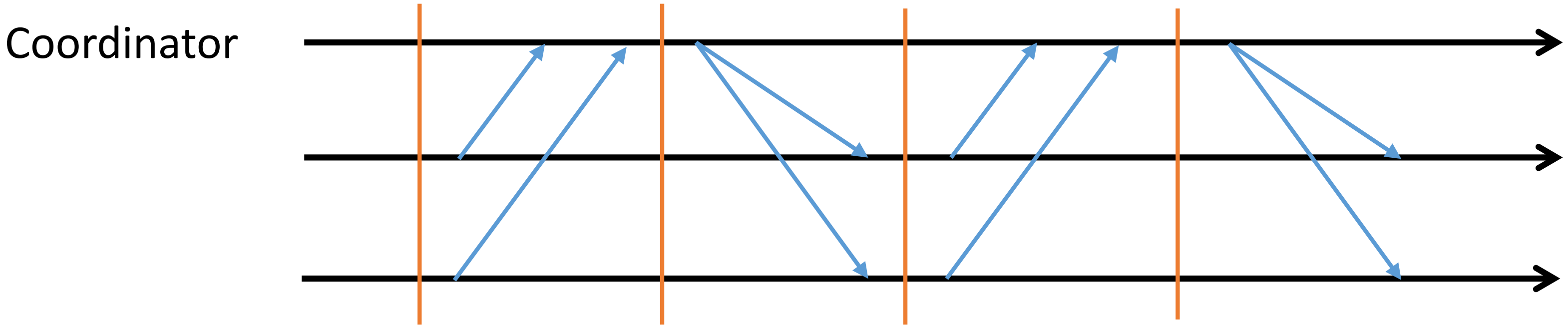
Last Voting Algorithm

Round $r = 4\phi - 3$ **:**
S_p^r :
send $\langle x_p \,,\, ts_p \rangle$ to $Coord(p, \phi)$

T_p^r :
if $p = Coord(p, \phi)$ **and**
number of $\langle \nu \,,\, \theta \rangle$ received $> n/2$ **then**
let $\overline{\theta}$ be the largest θ from $\langle \nu \,,\, \theta \rangle$ received
$vote_p$:= one ν such that $\langle \nu \,,\, \overline{\theta} \rangle$ is received
$commit_p := \mathtt{true}$

Round $r = 4\phi - 1$ **:**
S_p^r :
if $ts_p = \phi$ **then**
send $\langle ack \rangle$ to $Coord(p, \phi)$

T_p^r :
if $p = Coord(p, \phi)$ **and**
number of $\langle ack \rangle$ received $> n/2$ **then**
$ready_p := \mathtt{true}$



Round $r = 4\phi - 2$ **:**
S_p^r :
if $p = Coord(p, \phi)$ **and** $commit_p$ **then**
send $\langle vote_p \rangle$ to all processes

T_p^r :
if received $\langle v \rangle$ from $Coord(p, \phi)$ **then**
$x_p := v$; $ts_p := \phi$

Round $r = 4\phi$ **:**
S_p^r :
if $p = Coord(p, \phi)$ **and** $ready_p$ **then**
send $\langle vote_p \rangle$ to all processes

T_p^r :
if received $\langle v \rangle$ from $Coord(p, \phi)$ **then**
DECIDE(v)
if $p = Coord(p, \phi)$ **then**
$ready_p := \mathtt{false}$
$commit_p := \mathtt{false}$

Verification

Goals for the verification

- Safety and liveness properties
  - Agreement, Validity, Irrevocability
  - Termination
- User provided invariants
  - Quite simple
    - No channels
    - All the processes have the same PC
- Flexibility
  - Being able to handle a large class of algorithms

Invariant for the Last Voting example

$$\forall i. \neg decided(i) \land \neg ready(i)$$

$$\lor \quad \exists\, v, t, A.\ A = \{\, i.\ ts(i) > t\} \land |A| > n/2$$

$$\land \qquad \forall i. i \in A \Rightarrow x(i) = v$$

$$\land \qquad \forall i. decided(i) \Rightarrow x(i) = v$$

$$\land \qquad \forall i. commit(i) \lor ready(i) \Rightarrow vote(i) = v$$

$$\land \qquad t \leq \Phi$$

$$\land \qquad \forall i.\ ts(i) = \Phi \Rightarrow commit(coord(i)) = v$$

Implementing a system

Architecture

User application

Algorithm

Round abstraction
(Predicate)

Network

Algorithms

- Does not terminate!
  - Related to recovery procedures
- How does it start ?
  - Praise the benevolent sysadmin
- Integration in the user program
  - Relating the guarantees of the algorithms to the rest of the system.

Round $r = 4\phi - 3$ **:**
  S_p^r :
    send $\langle x_p , ts_p \rangle$ to $Coord(p, \phi)$

  T_p^r :
    if $p = Coord(p, \phi)$ **and** number of $\langle v , \theta \rangle$ received $> n/2$ **then**
      let $\overline{\theta}$ be the largest θ from $\langle v , \theta \rangle$ received
      $vote_p :=$ one v such that $\langle v , \overline{\theta} \rangle$ is received
      $commit_p :=$ `true`

Round $r = 4\phi - 2$ **:**
  S_p^r :
    if $p = Coord(p, \phi)$ **and** $commit_p$ **then**
      send $\langle vote_p \rangle$ to all processes

  T_p^r :
    if received $\langle v \rangle$ from $Coord(p, \phi)$ **then**
      $x_p := v$; $ts_p := \phi$

Round $r = 4\phi - 1$ **:**
  S_p^r :
    if $ts_p = \phi$ **then**
      send $\langle ack \rangle$ to $Coord(p, \phi)$

  T_p^r :
    if $p = Coord(p, \phi)$ **and** number of $\langle ack \rangle$ received $> n/2$ **then**
        $ready_p :=$ `true`

Round $r = 4\phi$ **:**
  S_p^r :
    if $p = Coord(p, \phi)$ **and** $ready_p$ **then**
      send $\langle vote_p \rangle$ to all processes

  T_p^r :
    if received $\langle v \rangle$ from $Coord(p, \phi)$ **then**
      DECIDE(v)
    if $p = Coord(p, \phi)$ **then**
      $ready_p :=$ `false`
      $commit_p :=$ `false`

Boundary conditions

- Assumes that every replicas runs forever
  - Starting consensus among some replicas is not so different from a consensus problem itself ?
    - Difference between not deciding and not knowing we were supposed to decide
    - What are the assumption you need to get to the point where you can run consensus ?
  - Sometime you don’t even know how many replicas there are.
    - … coming from industry people who are running data centers …

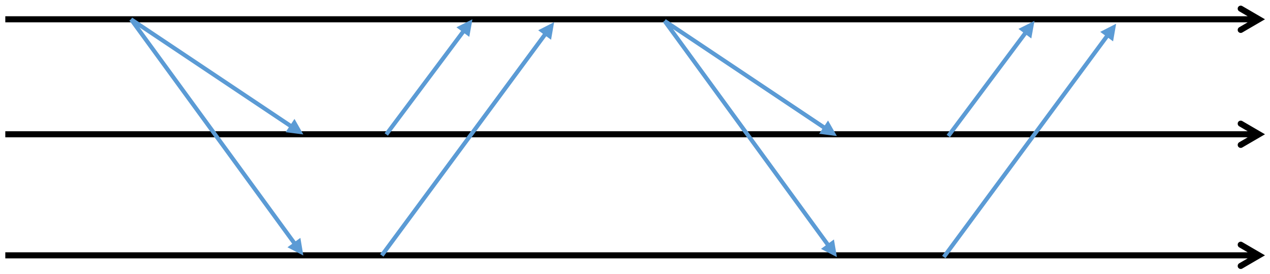
Consensus vs Atomic broadcast

- In theory, we can solve atomic broadcast by reduction to consensus.
  - Reverse is also true
- In practice, we need another algorithm.

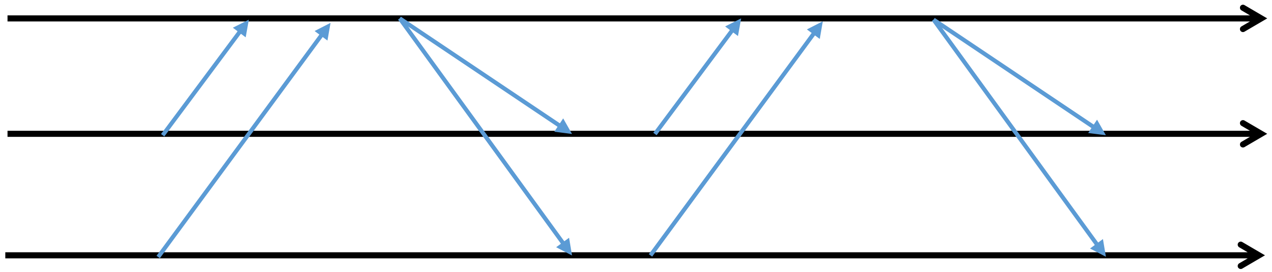
- Paxos vs Zab
  - Zab [Junqueira et al. 11]: atomic broadcast algorithm used in Zookeeper
  - For performance reason
  - Share many ideas but Zab is designed to enable quick recovery
  - Also customized properties (stronger than atomic broadcast ?)

Same principle, different results ?

Paxos



Last Voting



Predicate

- Interface between the "synchronous" rounds and "asynchronous" world.
  - Relative speed of replicas (δ)
  - Network delay (ϕ)
- example [Hutle & Schiper 07]

Algorithm 2 Ensuring $\mathcal{P}_{su}(\pi_0, -, -)$ with a '$\overline{\pi_0}$-down" good period

Reception policy: Highest round number first
$msgsRcv_p \leftarrow \emptyset$ {*set of messages received*}
$r_p \leftarrow 1$ {*round number*}
$next\_r_p \leftarrow 1$ {*next round number*}
$s_p \leftarrow init_p$ {*state of the consensus algorithm*}
while $true$ **do**
  $msg \leftarrow S_p^{r_p}(s_p)$
  send $\langle msg, r_p \rangle$ to all
  $i_p \leftarrow 0$
  while $next\_r_p = r_p$ **do**
    $i_p \leftarrow i_p + 1$
    if $i_p \geq 2\delta + n + 2\phi$ **then**
      $next\_r_p \leftarrow r_p + 1;$
    receive a message
    if message is $\langle msg, r' \rangle$ from q **then**
      $msgsRcv_p \leftarrow msgsRcv_p \cup \{\langle msg, r', q \rangle\}$
      if $r' > r_p$ **then**
        $next\_r_p \leftarrow r'$
  $R \leftarrow \{\langle msg', q' \rangle \mid \langle msg', r_p, q' \rangle \in msgsRcv_p\}$
  $s_p \leftarrow T_p^{r_p}(R, s_p)$
  forall r' **in** $[r_p{+}1, next\_r_p{-}1]$ **do** $s_p \leftarrow T_p^{r'}(\emptyset, s_p)$
  $r_p \leftarrow next\_r_p$

Related work

- TLA+ [Lamport 91] that comes with a model checker and proof assistant [Chaudhuri et al. 08]
- Isabelle formalization of algorithms in the HO model [Charron-Bost & Merz 09]
- Mace [Killian et al. 07] a DSL for distributed systems, comes with a model-checker.
- Declarative networking: (logic) programming [Loo et al. 06] and verification [Wang et al. 09]

Summary

Implementing a correct fault-tolerant system is hard:

- We have the tool to help:
  - ATPs, SMT-solvers, model-checkers, …
- Where to position ourselves ?
  - Verification/programming abstraction: more tractable vs closer to real systems
  - Formalization of the boundaries of the systems
    - Start/stop
    - Interactions between main algorithms and recovery procedures
    - Customization of algorithms and properties of the system which uses the algorithm