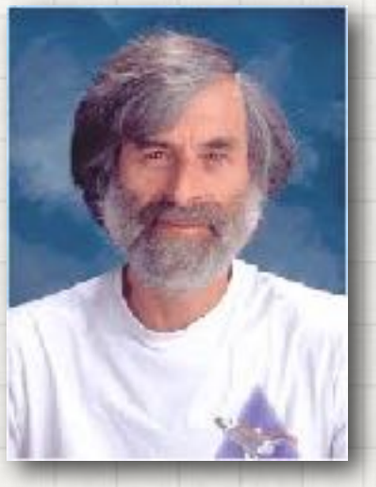
PAXOS (AND A LITTLE ABOUT OTHER CONSENSUSES)

//RUSLAN SHEVCHENKO



Leslie Lamport
The Part-Time Parlament

PAXOS PAPERS

1. Leslie Lamport. The Part-Time Parliament.
ACM Transactions on Computer Systems. 1990 — rejected
resubmitted in 1998 — published
https://www.microsoft.com/en-us/research/publication/part-time-parliament/

2. Leslie Lamport. Paxos Made Simple
ACM SIGACT News (Distributed Computing Column) 32, 4
(Whole Number 121, December 2001) 51-58.
https://lamport.azurewebsites.net/pubs/paxos-simple.pdf

3. Tushar Deepak Chandra, Robert Griesemer, Joshua Redstone
Paxos Made Live - an Engineering Perspective.
ACM, Symposium on Principles of Distributed Computing. 2007
https://ai.google/research/pubs/pub33002

NON-PAXOS PAPERS

// will talk a little, details can be a theme for another PWL.

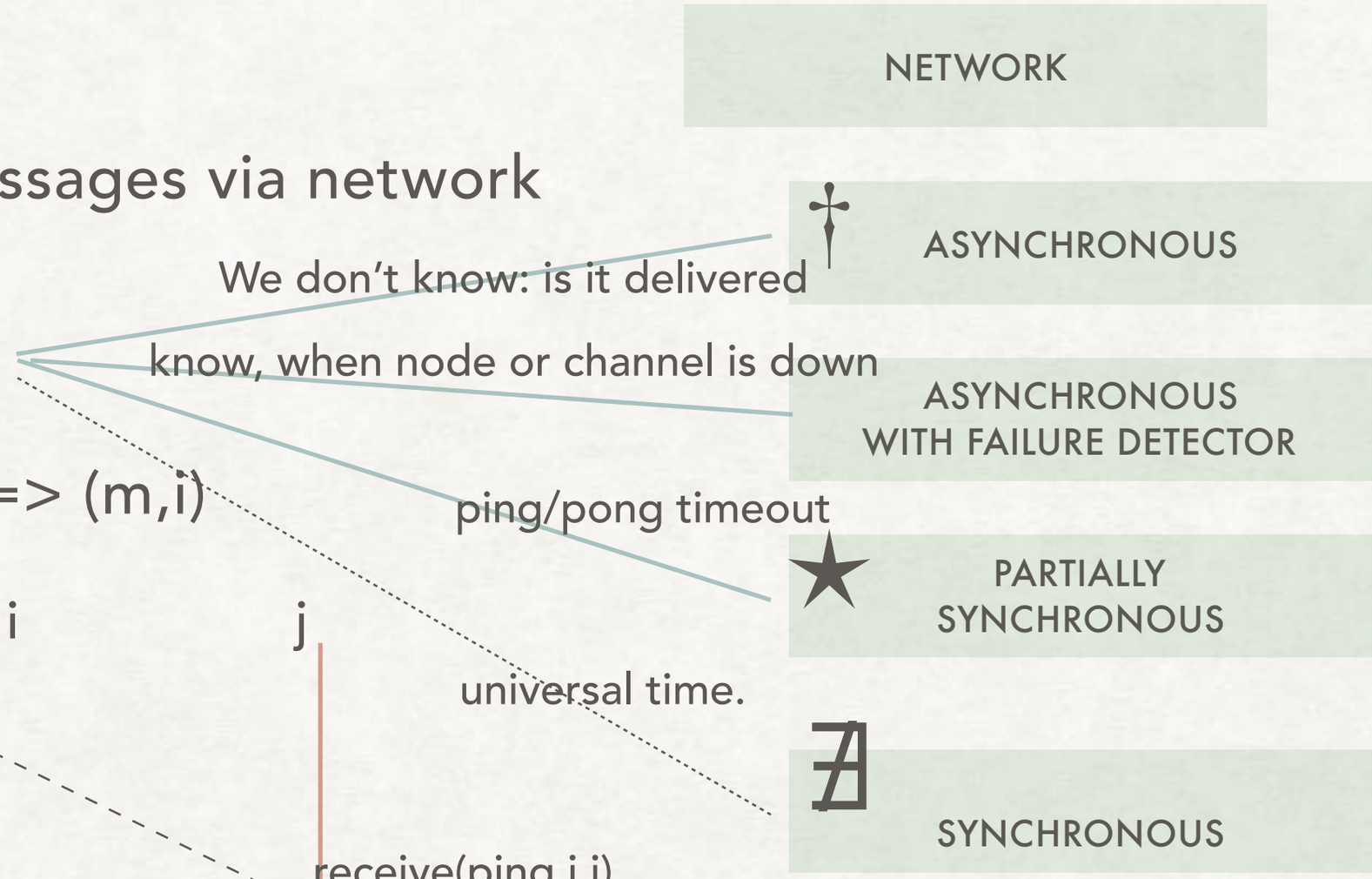
1. Diego Ongaro and John Ousterhout
   In Search of an Understandable Consensus Algorithm
   2014 USENIX Annual Technical Conference.
   https://www.usenix.org/system/files/conference/atc14/atc14-paper-ongaro.pdf
   https://raft.github.io/

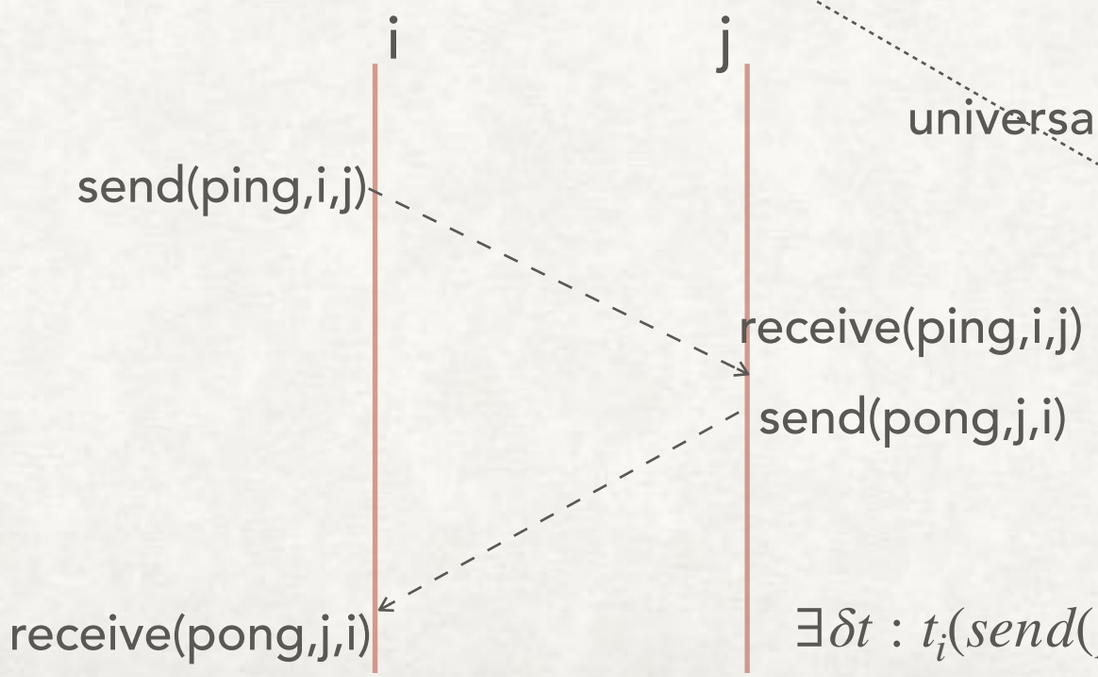
2. Miguel Castro and Barbara Liskov
   Practical Byzantine Fault Tolerance
   Third Symposium on Operating Systems Design and Implementation,
   New Orleans, USA, February 1999
   http://pmg.csail.mit.edu/papers/osdi99.pdf
   https://github.com/luckydonald/pbft

3. Leslie Lamport.
   Byzantizing Paxos by Refinement.
   Distributed Computing: 25th International Symposium: DISC 2011, David Peleg, editor. Springer-Verlag (2011) 211-224.
   http://lamport.azurewebsites.net/pubs/web-byzpaxos.pdf

DISTRIBUTED SYSTEMS: FUNDAMENTALS

- N nodes.
- Can send messages via network
  - Send(m,i,j)
  - Receive: j => (m,i)





$$\exists \delta t : t_i(send(ping, i, j)) - t_i(receive(pong, i)) < \delta_t$$

DISTRIBUTED SYSTEM: FUNDAMENTALS

CONSENSUS PROBLEM

"We have an agreement in principle.
The question is, do we all have the same principles?"

p_i - program on node i.

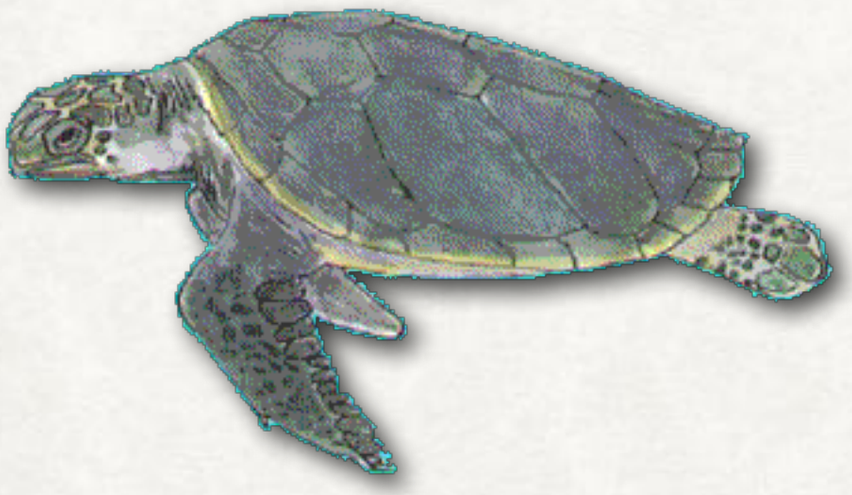
$$\exists c : \forall i, j \in N : result(p_i) = c$$

DISTRIBUTED SYSTEMS: FUNDAMENTALS

ASYNCHRONOUS CONSENSUS IS IMPOSSIBLE

- Fischer, Lynch and Patterson. 'Impossibility of Distributed Consensus with One Faulty Process', J. ACM 32, 2 (April 1985), 374-382. http://cs-www.cs.yale.edu/homes/arvind/cs425/doc/fischer.pdf

- Proof Idea:



≈

PAXOS

PART-TIME PARLIAMENT

- Archeologist, describing voting system.
- Parlament ‘work from home/travel’
- Messages are sent by part-time courier
- Any member at any time can fire issue
- When the majority of votes is collected,
  the decision had to be done
- members can be non-reliable or reply with long delay

PACTS

PART-TIME PARLIAMENT

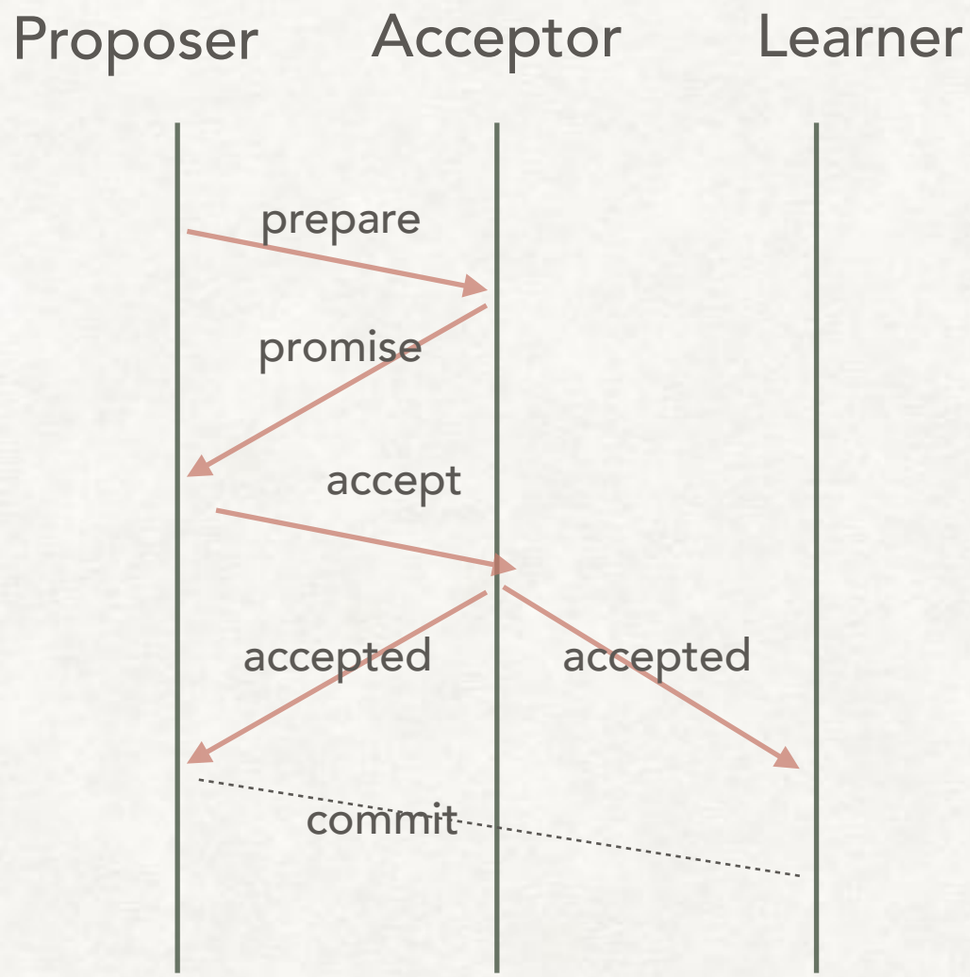
Roles:

- Proposer,
- Acceptor
- Learner

On practice, merged in one.

Messages:

- prepare
- promise
- accept
- accepted
- optional, implicit



PACTS

PART-TIME PARLIAMENT

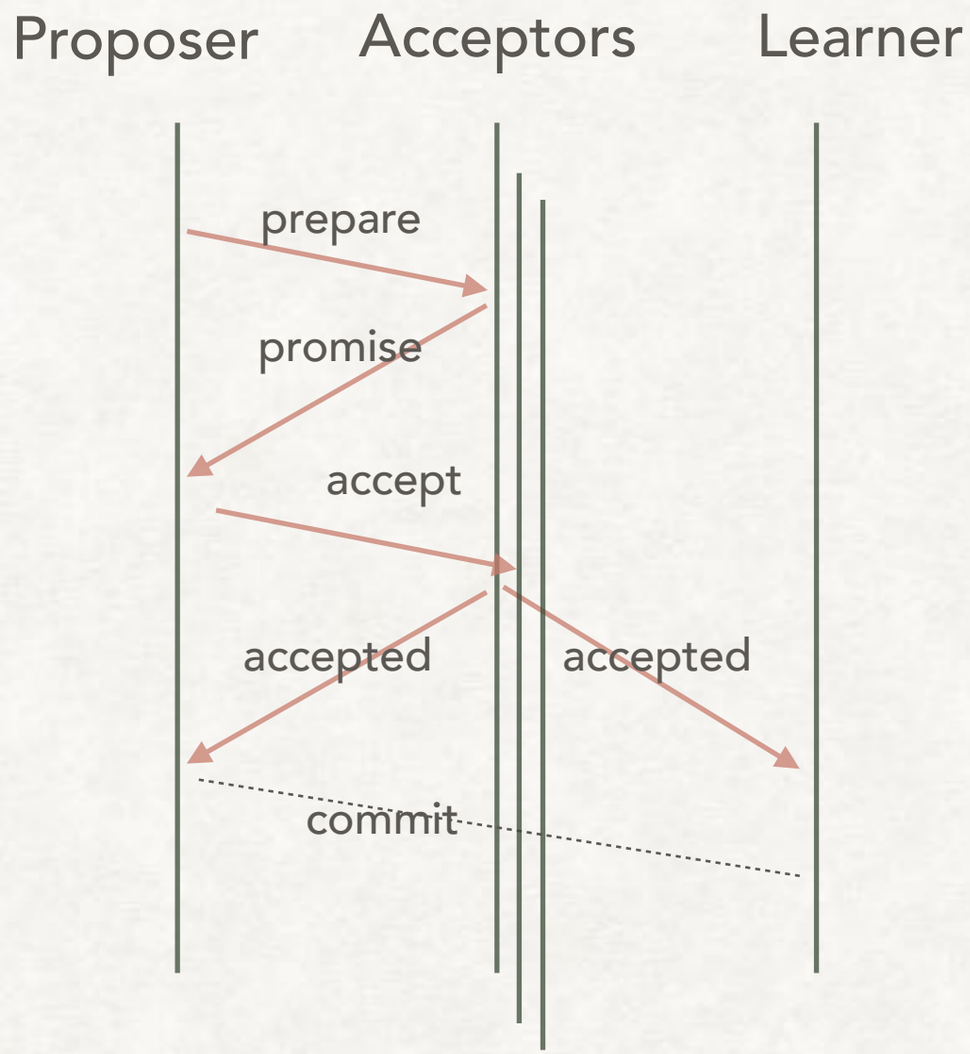
Roles:

- Proposer,
- Acceptor
- Learner

On practice, merged in one.

Messages:

- prepare
- promise
- accept
- accepted
- optional, implicit



Quorum. $|i \in Q| > N/2$

PACTOS

FORMALIZATION

- Nodes: $n_i \in N$, each node have number

$issue \in I$

$value \in V$

$prepare(n_{proposer} : N, issue : I, round : N)$

$promise(n_{acceptor} : N, issue : I, round : N, value : Option[V])$

$accept(n_{proposer} : N, issue : I, round : N, value : V)$

Optional

$accepted(n_{acceptor} : N, issue : I, round : N, value : V)$

$acceptNack(n_{acceptor} : N, issue : I, round : N)$

$ballot(message) = (message\,.\,value, message\,.\,round)$

PAXOS

PROPOSER

```
    State: myProposal: Option[Value], round:Nat,

round <- myN
broadcast(Prepare(me,issue,round))

receive{
  case p:Promise if checkRound(p) =>
     myProposal=best(myProposal, ballot(p))
     if (prepare-quorum-ready) {
       value = myProposal.getOrElse(random)
       broadcast Accept(me,issue,round,value)
     }
     if (quorum-failed) {
       fail-round
     }
  case m: Accepted if (checkRound(m)) =>
     if (accepted-quorum-ready(m)){
       //(local Commit(myProposal,round))
       finish(myProposal)
     } else if (quorum-failed) {
       failRound
     }
  case m:Commit =>  finish(m.value)
}
```

$prepare(n_{proposer} : N, issue : I, round : N)$

$promise(n_{acceptor} : N, issue : I, round : N, value : Opti$

$accept(n_{proposer} : N, issue : I, round : N, value : V)$

$accepted(n_{acceptor} : N, issue : I, round : N, value : V)$

$commit(n_{proposer} : N, issue : I, round : N, value : V)$

```
def fail-round() = {

   round = ((round/N)+1)*N + myN
   myPropsal = None

   wait-some-time
   restart()

}
```

PAXOS

ACCEPTOR

```
    State: myValue: Option[Value], round:N,

receive{
  case p:Prepare =>
       r = if (myValue.isEmpty) p.round
             else this.round
       reply Promise(me,p.issue,r,myValue)

  case m: Accept if (p.round > round) =>
     myValue match {
       case None =>
         myValue = p.value
         round = p.round
         reply Accepted(me,m.issue,round,m.value)
       case Some(v) =>
         if (m.value == v) {
           reply Accepted(me,m.issue,round,m.value)
         } else {
           reply AcceptedNack
         }
  case m:Commit =>  finish(m.value)
}
```

$$Accepted(v, r) \Rightarrow \forall k > r : Promise(v, k)$$

PAXOS

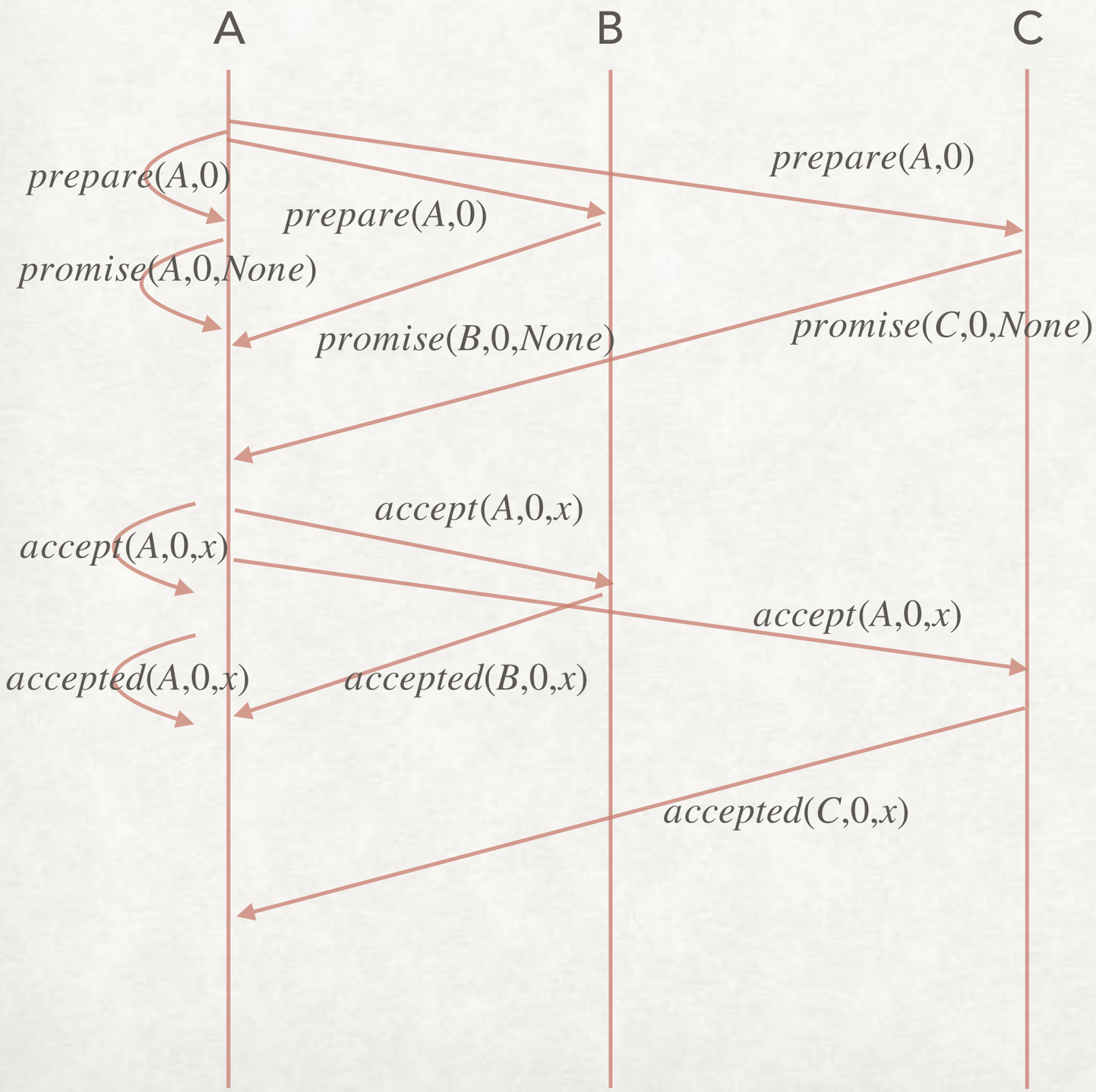
LEARNER

```
    State: myValue: Option[Value], round:N,


receive{
  case m:Accepted =>
      if (checkQuorum(m)) {
        learn(m.issue,m.value)
                // generate local commit(m)
      }

}
```

// On practice, Proposal, Acceptor, Learner — the same.

PAXOS - RUN

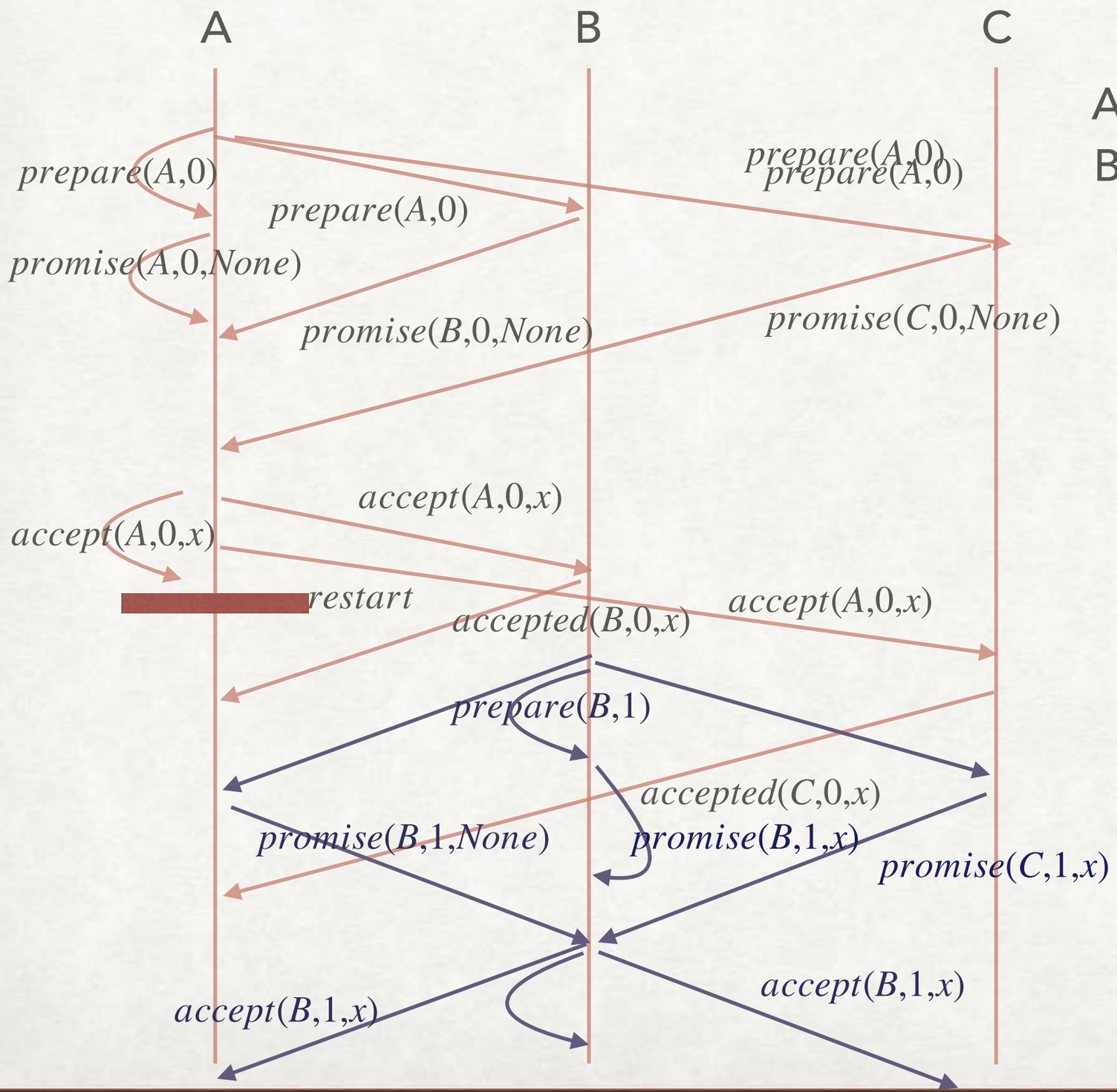
LET WE HAVE 3 NODES, ALL PLAY BOTH ROLES



Happy path

PAXOS - RUN

LET WE HAVE 3 NODES, ALL PLAY BOTH ROLES



A fail,

B initiate next round

PROPERTIES

PROOFS

- Safety:
  - Only a value that has been proposed may be chosen.
  - Only a single value is chosen. (non-trivial)
  - Only chosen values are learned.
- Liveness impossible (due to FLP), eventual progress instead:
  - Proposal will learn previous highest number

PAXOS SAFETY

ONLY A SINGLE VALUE HAS CHOSEN

$$\forall n \in N : Accepted(n, round, v) \rightarrow \forall k > round : Accepted(n, k, v)$$

$$\exists Q \in Qourm(N) : \forall n \in Q : Accepted(n, r, v) \rightarrow Chosen(v, r)$$

$$Chosen(v, r) \rightarrow \forall k > r : Chosen(v, k)$$

Let we have 2 values chosen:

$$Chosen(v1,r)\ \&\ Chosen(v2,k) \rightarrow Chosen(v1,max(r,k)) \wedge Chosen(v2,max(r,k))$$

$$\exists Q_1 \in Quorum(N), Q_2 \in Quorum(N) : \forall n_1 \in Q_1 \forall n_2 in Q_2 :$$
$$Accepted(n_1, v_1, r) \wedge Accepted(n_2, v_2, r)$$

$$v_1 \neq v_2 \rightarrow Q_1 \cap Q_2 = \varnothing$$

But any two Quorums should be intersect

$$Q_1 \in Qourum(N), Q_2 \in Qourum(N) \rightarrow (Q_1 \wedge Q_2 \neq \varnothing)$$

!Contradiction!

PAXOS SAFETY

ONLY A SINGLE VALUE HAS CHOSEN

$$\forall n \in N : Accepted(n, round, v) \rightarrow \forall k > round : Accepted(n, k, v)$$

$$\exists Q \in Qourm(N) : \forall n \in Q : Accepted(n, r, v) \rightarrow Chosen(v, r)$$

$$Chosen(v, r) \rightarrow \forall k > r : Chosen(v, k)$$

Let we have 2 values chosen:

$$Chosen(v1,r)\ \&\ Chosen(v2,k) \rightarrow Chosen(v1,max(r,k)) \wedge Chosen(v2,max(r,k))$$

$$\exists Q_1 \in Quorum(N), Q_2 \in Quorum(N) : \forall n_1 \in Q_1 \forall n_2 in Q_2 :$$
$$Accepted(n_1, v_1, r) \wedge Accepted(n_2, v_2, r)$$

$$v_1 \neq v_2 \rightarrow Q_1 \cap Q_2 = \varnothing$$

But any two Quorums should be intersect

$$Q_1 \in Qourum(N), Q_2 \in Qourum(N) \rightarrow (Q_1 \wedge Q_2 \neq \varnothing)$$!Contradiction!

PACTS

PROGRESS

$$fail(n, r) \rightarrow \exists n \in N, r_1 > r : Prepare(n, r_1)$$

// scheduling run on randomized timeout

Modifications:

Multi-Paxos: same leader process events until failure

not need in Prepare, until we have no failures.

PACT-OS IMPLEMENTATION

CHUBBY – GOOGLE LOCK SERVICE

3. Tushar Deepak Chandra, Robert Griesemer, Joshua Redstone
   Paxos Made Live - an Engineering Perspective.
   ACM, Symposium on Principles of Distributed Computing. 2007
   https://ai.google/research/pubs/pub33002

2 pages pseudocode => k* 10^2 LOC in C++.

Code generated from own specification language

+ Determenistics variant for testing.

Performance superior to 3DB

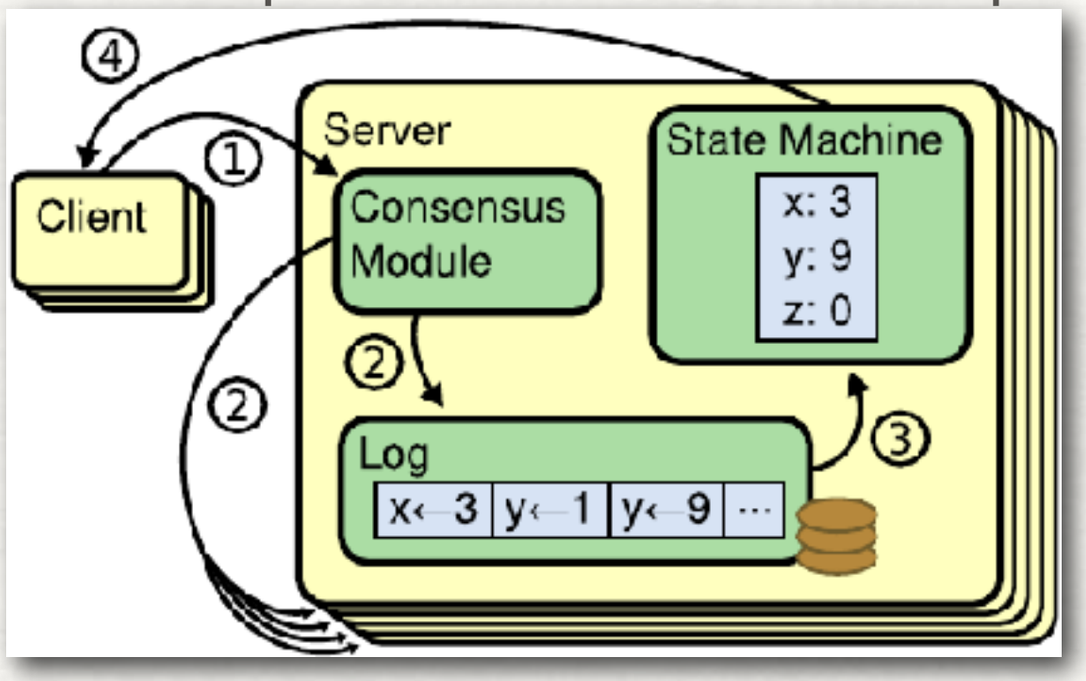
MultiPaxos + optimizations
- lagging.

Testing fail-over solutions is hard.

RAFT

IN SEARCH FOR UNDESTANDABLE CONSENSUS ALGORITHM (2014)

- MultiPaxos usually used as part of state-machine replication.



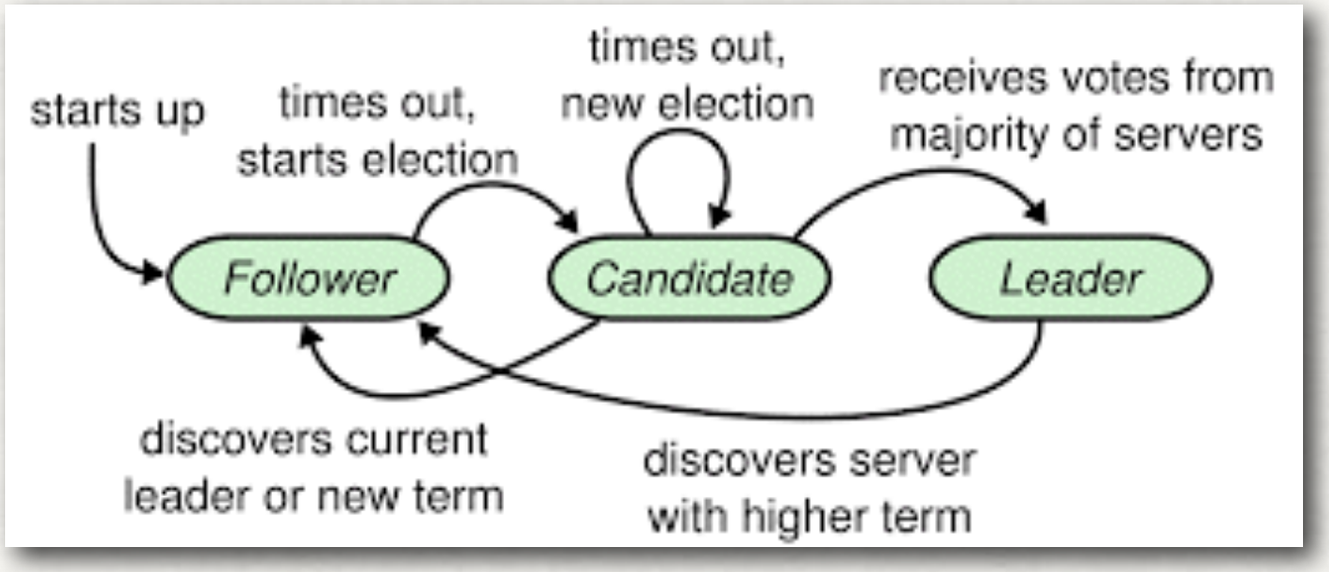
- One writer (leader), do something and send to followers. When leader failed, new leader is elected.

- RAFT - state machine replication, simpler (in theory) than paxos with similar performance characteristics.

RAFT

STATE MACHINE REPLICATION

- https://raft.github.io/
- Instead

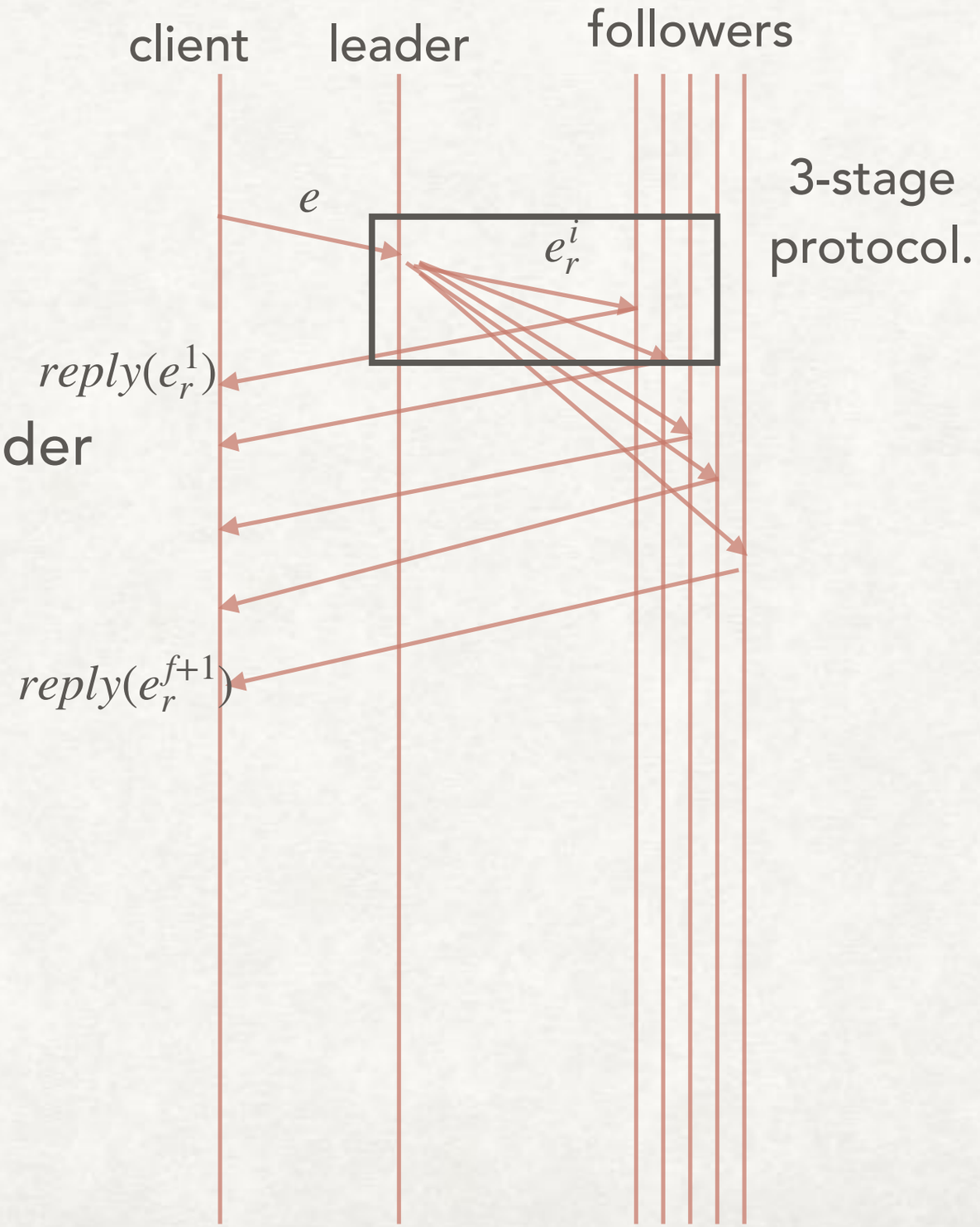


- Can be viewed as ‘refactoring’ of PaxOS: merging roles into one,
- use heartbeat for failure detection
- on leader failure, node become a candidate and request other node to vote (choose one) as leader.

BPFT

BYZANTINE FAULT TOLERANCE

- Replicated state machine
- Some nodes can lie (f):
  - Client submit item to leader
  - Gather f+1 replies.
- If all replies
  - are the same: all ok
  - otherwise - resend.

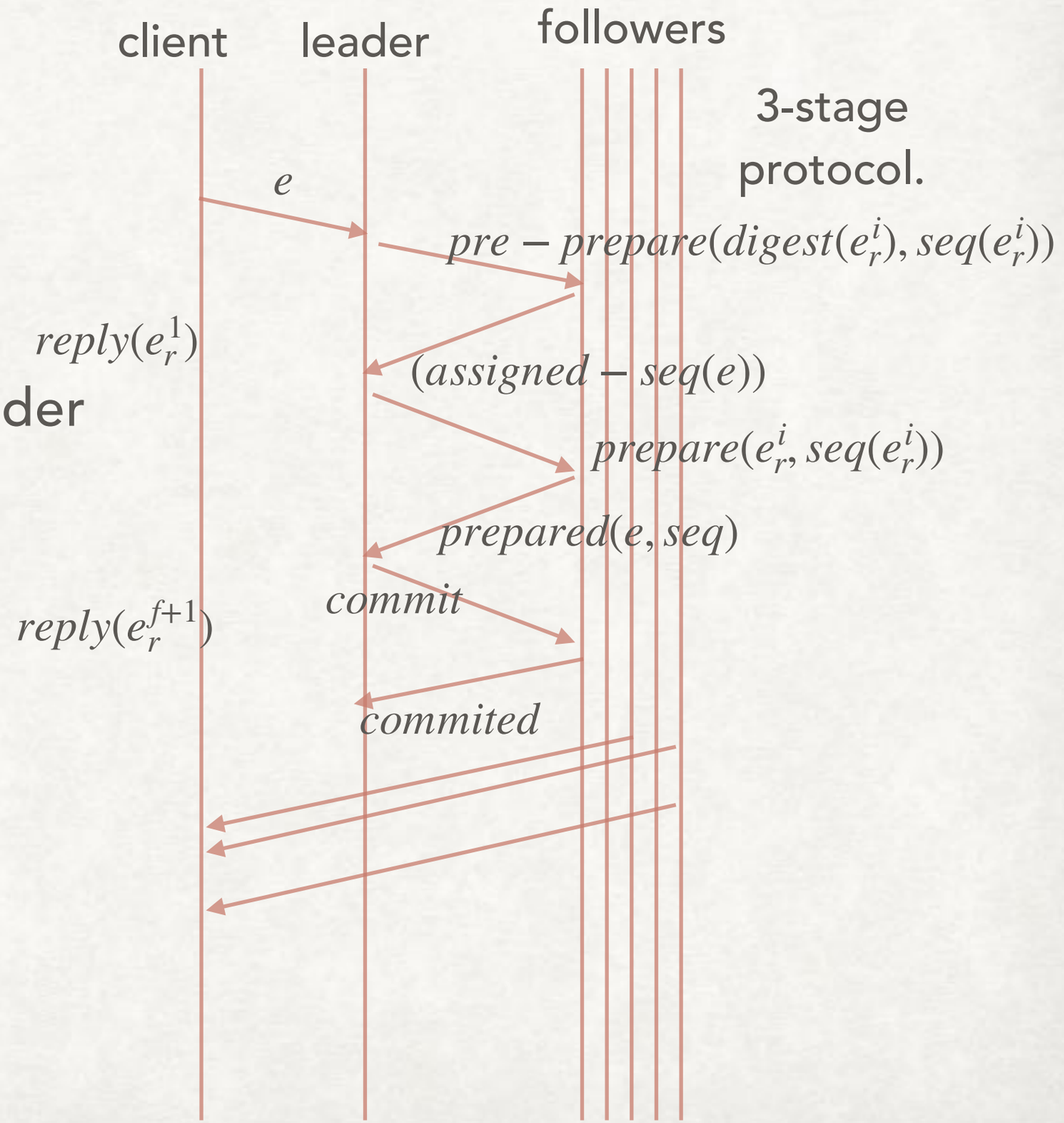


3-stage protocol.

BPFT

BYZANTINE FAULT TOLERANCE

- Replicated state machine
- Some nodes can lie (f):
  - Client submit item to leader
  - Gather f+1 replies.
  - If all replies
    - are the same: all ok
    - otherwise - resend.



BYZANTIZING PAXOS BY REFINEMENT

LESLI LAMPORT, 2011

- BPCon — Byzantine fault tolerance (prev. slides)
- PCon — Paxos
- Byzantine: Algorithm(N) => Algorithm(N+f)
- $Byzantine(PCon) = BPCon \qquad N > 3f$

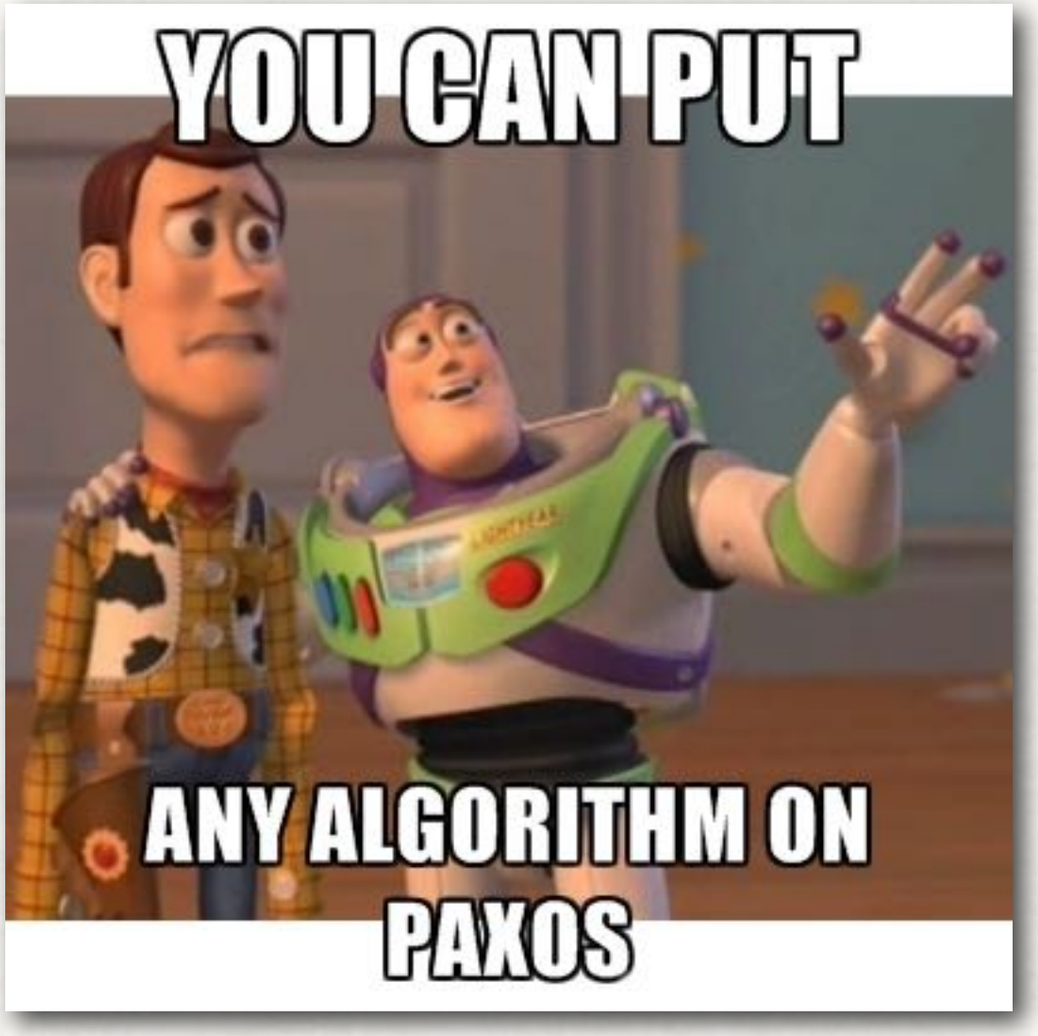
  f+1 round (in worst case)

Idea: $S \to M \Rightarrow BzQourum(S) \to M$

$S \to M \Rightarrow BzQourum(S) \to Proof(BzQourm(S), M)$

THANKS FOR LISTENING

PAPERS WE LOVE, KYIV, 27.08.2018



//The Story, told by Ruslan Shevchenko.

ruslan@shevchenko.kiev.ua
@rssh1