Lecture #4: Solving SMR with Crash Faults in Partial Synchrony: The Essence of Paxos & Raft

COMS 4995-001: The Science of Blockchains

URL: https://timroughgarden.org/s25/

Tim Roughgarden

Goals for Lecture #4

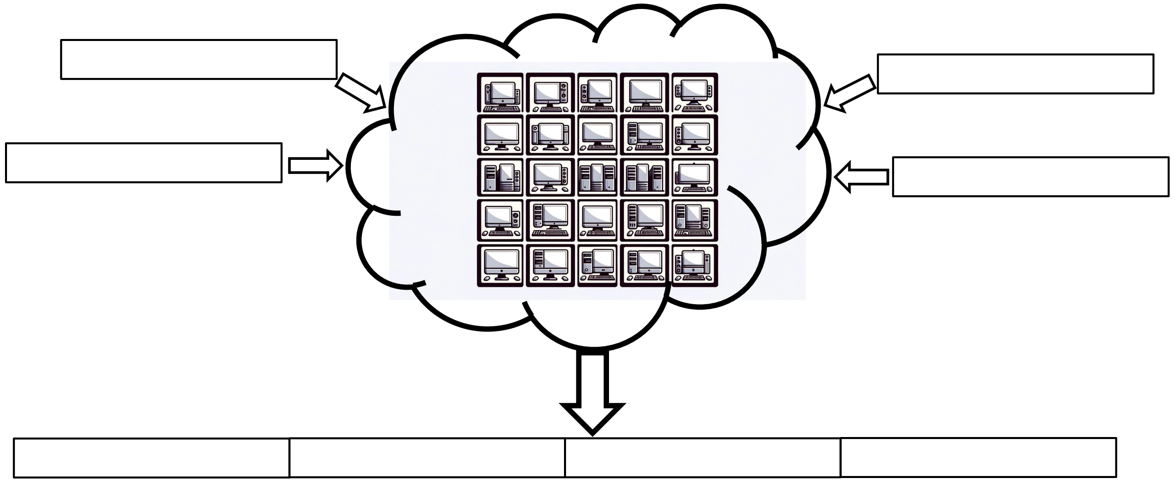
1. Understand the "partially synchronous" model.
   – useful "sweet spot" between the synchronous, asynchronous models
2. Limits on what is possible.
   – no hope unless a strict majority of validators are non-faulty
3. The Paxos/Raft protocol and its guarantees.
   – widely used in production (e.g. see the Raft Wikipedia page)

State Machine Replication (SMR)

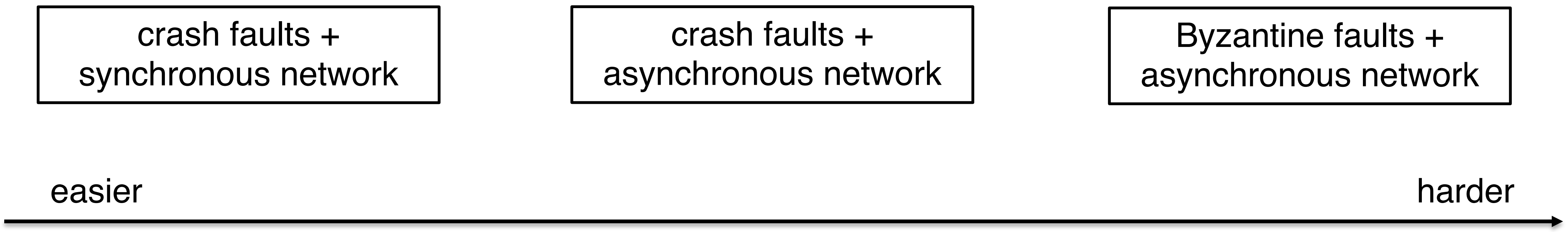
SMR: version of consensus appropriate for a blockchain protocol.

- “state machine” = for us, current state of virtual machine
- “replication” = all validators perform same state transitions
- “clients” submit transactions (“txs”) to validators
- each validator maintains an append-only list of finalized txs (a.k.a. “log” or “history”)

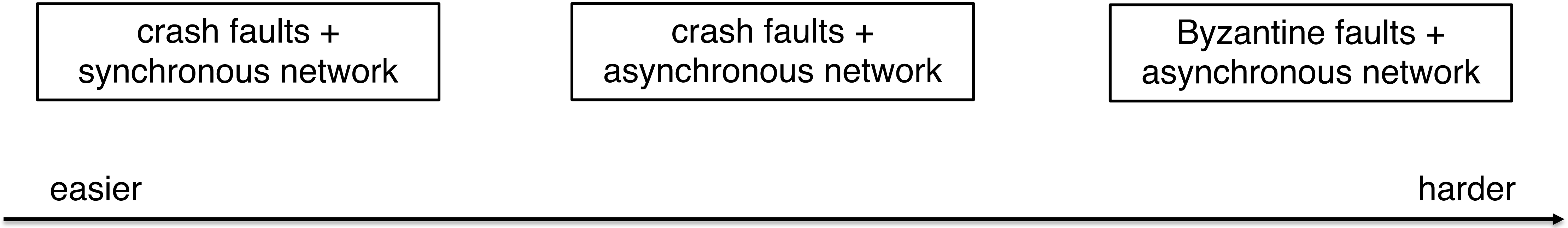
Goal: a protocol that satisfies consistency and liveness.



A Road Map to Practical SMR Protocols

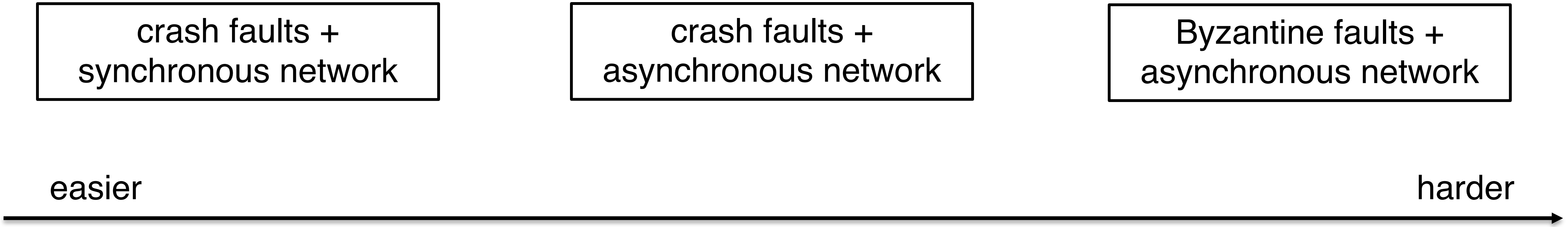


A Road Map to Practical SMR Protocols



Lecture #3: Protocol B solves SMR with crash faults in synchrony.

A Road Map to Practical SMR Protocols



Lecture #3: Protocol B solves SMR with crash faults in synchrony.

FLP Theorem: can’t solve SMR in the asynchronous model with the threat of a single crash fault.

A Road Map to Practical SMR Protocols

crash faults + synchronous network	crash faults + ~~asynchronous~~ partially synchronous network	Byzantine faults + ~~asynchronous~~ partially synchronous network

easier → harder

Lecture #3: Protocol B solves SMR with crash faults in synchrony.

FLP Theorem: can't solve SMR in the asynchronous model with the threat of a single crash fault.

The Partially Synchronous Model (Informal)

Idea: want to accommodate unexpected outages/attacks (unlike synchronous model). But they must end at some point, right?

The Partially Synchronous Model (Informal)

Idea: want to accommodate unexpected outages/attacks (unlike synchronous model). But they must end at some point, right?

Revised goals:

- under "normal conditions," guaranteed consistency + liveness

The Partially Synchronous Model (Informal)

Idea: want to accommodate unexpected outages/attacks (unlike synchronous model). But they must end at some point, right?

Revised goals:

- under "normal conditions," guaranteed consistency + liveness
- under attack/outage, give up liveness only
  - so protocol may stall when there's something wrong
  - FLP theorem implies must give up either consistency or liveness
  - ideally, no assumptions on attack/outage other than finite duration

The Partially Synchronous Model (Informal)

Idea: want to accommodate unexpected outages/attacks (unlike synchronous model). But they must end at some point, right?

Revised goals:

- under “normal conditions,” guaranteed consistency + liveness
- under attack/outage, give up liveness only
  - so protocol may stall when there’s something wrong
  - FLP theorem implies must give up either consistency or liveness
  - ideally, no assumptions on attack/outage other than finite duration
- after attack ends, quickly become live again

The Partially Synchronous Model

Formal model:

- shared global clock (timesteps=0,1,2,…)
  - can relax to bounded difference in clock speeds, but won't do so here

The Partially Synchronous Model

Formal model:

- shared global clock (timesteps=0,1,2,…)
  - can relax to bounded difference in clock speeds, but won't do so here
- known upper bound Δ on message delays in normal conditions
  - same as synchronous model

The Partially Synchronous Model

Formal model:

- shared global clock (timesteps=0,1,2,…)
  - can relax to bounded difference in clock speeds, but won't do so here
- known upper bound Δ on message delays in normal conditions
  - same as synchronous model
- unknown transition time GST ("global stabilization time") from asynchrony to synchrony (i.e., end of attack/outage)
  - protocol must satisfy its requirements no matter what the attack length

The Partially Synchronous Model

Formal model:

- shared global clock (timesteps=0,1,2,…)
  - can relax to bounded difference in clock speeds, but won't do so here
- known upper bound Δ on message delays in normal conditions
  - same as synchronous model
- unknown transition time GST ("global stabilization time") from asynchrony to synchrony (i.e., end of attack/outage)
  - protocol must satisfy its requirements no matter what the attack length
- summarizing, the promises on message delivery are:

The Partially Synchronous Model

Formal model:

- shared global clock (timesteps=0,1,2,…)
- known upper bound Δ on message delays in normal conditions
- unknown transition time GST ("global stabilization time") from asynchrony to synchrony (i.e., end of attack/outage)
  - protocol must satisfy its requirements no matter what the attack length
- summarizing, the promises on message delivery are:
  - sent at time $t \leq GST$ ➔ arrives by time $GST + \Delta$
  - sent at time $t \geq GST$ ➔ arrives by time $t + \Delta$

Security Thresholds

Note: The FLP Theorem does not immediately apply to the partially synchronous setting.

- adversary must choose some GST, can’t use unbounded delays after
- but, there will still be limits on what we can hope for

Security Thresholds

Note: The FLP Theorem does not immediately apply to the partially synchronous setting.

- adversary must choose some GST, can't use unbounded delays after
- but, there will still be limits on what we can hope for

Definition: security threshold = the fraction of faulty validators at which guaranteeing consensus flips from possible to impossible.

Security Thresholds

Note: The FLP Theorem does not immediately apply to the partially synchronous setting.

- adversary must choose some GST, can't use unbounded delays after
- but, there will still be limits on what we can hope for

Definition: security threshold = the fraction of faulty validators at which guaranteeing consensus flips from possible to impossible.

- ex: crash faults + synchrony ➔ security threshold ≈ 100%
  - Protocol B consistent and live even if only one validator remains

Security Thresholds

Note: The FLP Theorem does not immediately apply to the partially synchronous setting.

- adversary must choose some GST, can't use unbounded delays after
- but, there will still be limits on what we can hope for

Definition: security threshold = the fraction of faulty validators at which guaranteeing consensus flips from possible to impossible.

- ex: crash faults + synchrony ➔ security threshold ≈ 100%
  - Protocol B consistent and live even if only one validator remains
- ex: crash faults + asynchrony ➔ security threshold ≈ 0%
  - FLP Theorem: already hosed with a single crash fault

What Is Possible in Partial Synchrony?

Fact: crash faults + partial synchrony ➔ security threshold < 50%.

– i.e., no hope unless a strict majority of validators are non-faulty

What Is Possible in Partial Synchrony?

Fact: crash faults + partial synchrony ➔ security threshold < 50%.

- i.e., no hope unless a strict majority of validators are non-faulty

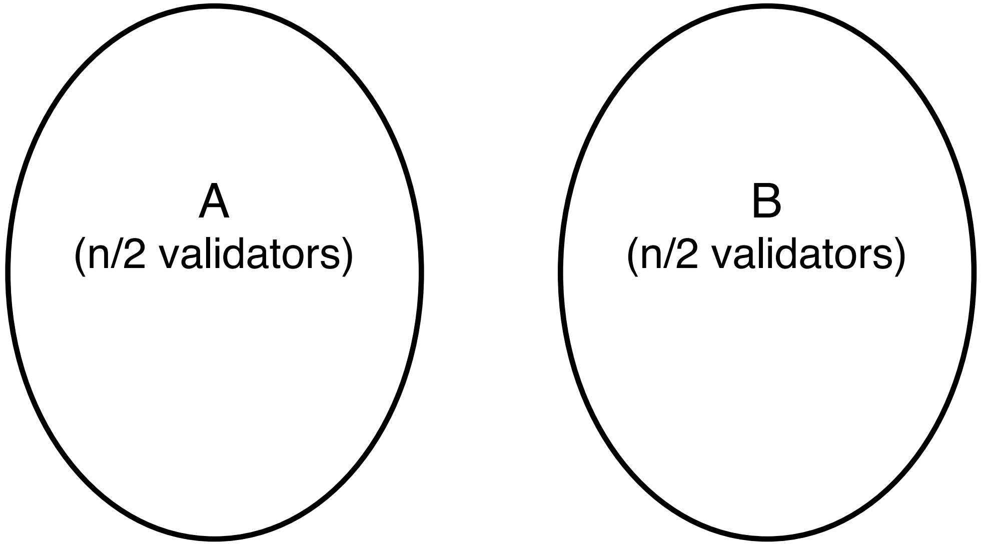
Key challenge: ambiguity between crashed validators and long message delays. [≈ “CAP Principle” from distributed systems]

What Is Possible in Partial Synchrony?

Fact: crash faults + partial synchrony ➔ security threshold < 50%.

- i.e., no hope unless a strict majority of validators are non-faulty

Key challenge: ambiguity between crashed validators and long message delays. [≈ "CAP Principle" from distributed systems]



What Is Possible in Partial Synchrony?

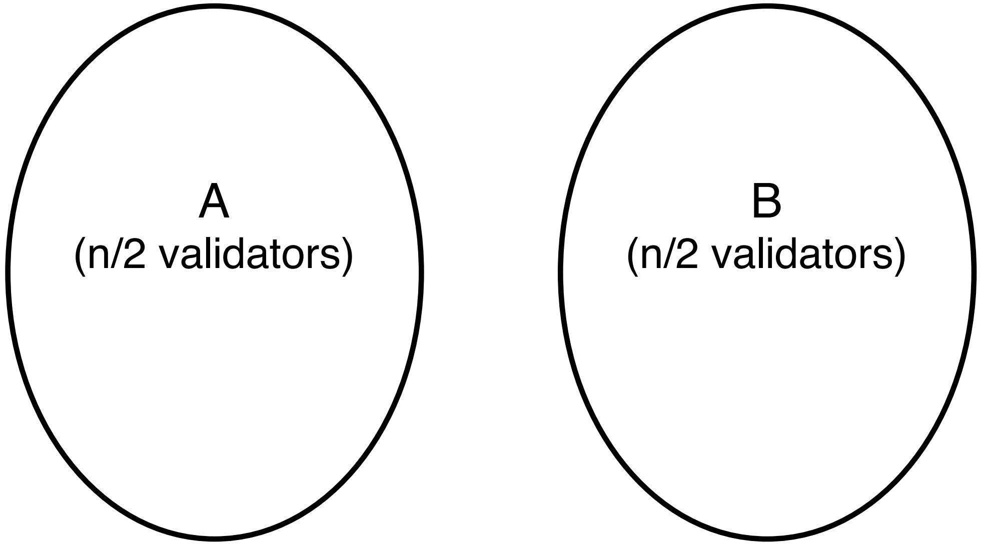
Fact: crash faults + partial synchrony ➔ security threshold < 50%.

- i.e., no hope unless a strict majority of validators are non-faulty

Key challenge: ambiguity between crashed validators and long message delays. [≈ "CAP Principle" from distributed systems]

Suppose: validators in A don't hear from any validators in B for a long time.

- should they finalize any new transactions?



What Is Possible in Partial Synchrony?

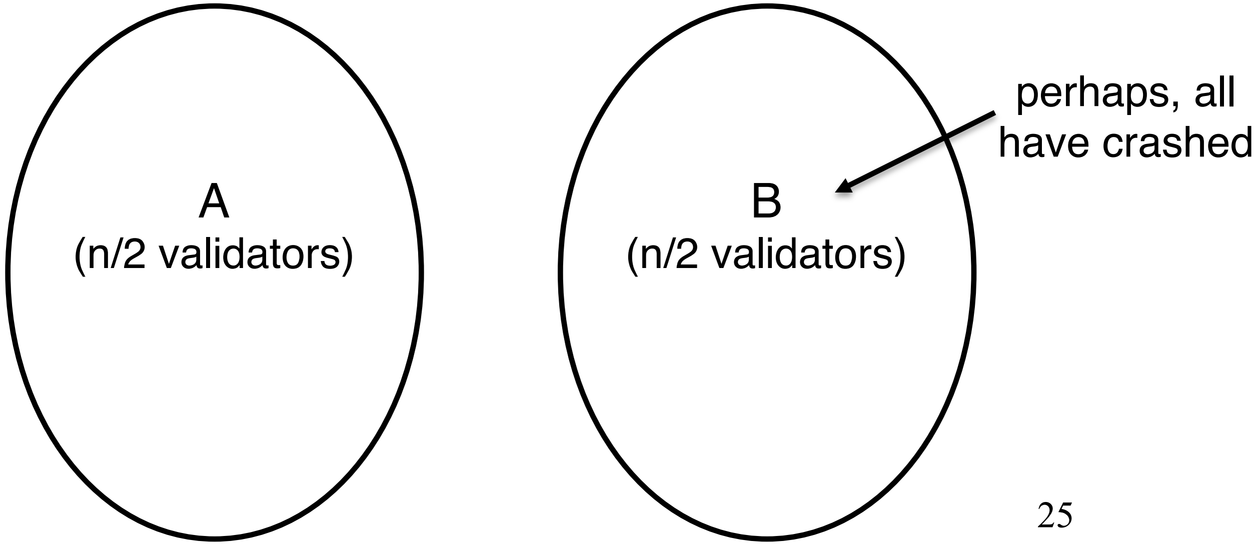
Fact: crash faults + partial synchrony ➔ security threshold < 50%.

– i.e., no hope unless a strict majority of validators are non-faulty

Key challenge: ambiguity between crashed validators and long message delays. [≈ "CAP Principle" from distributed systems]

Suppose: validators in A don't hear from any validators in B for a long time.

- should they finalize any new transactions?



What Is Possible in Partial Synchrony?

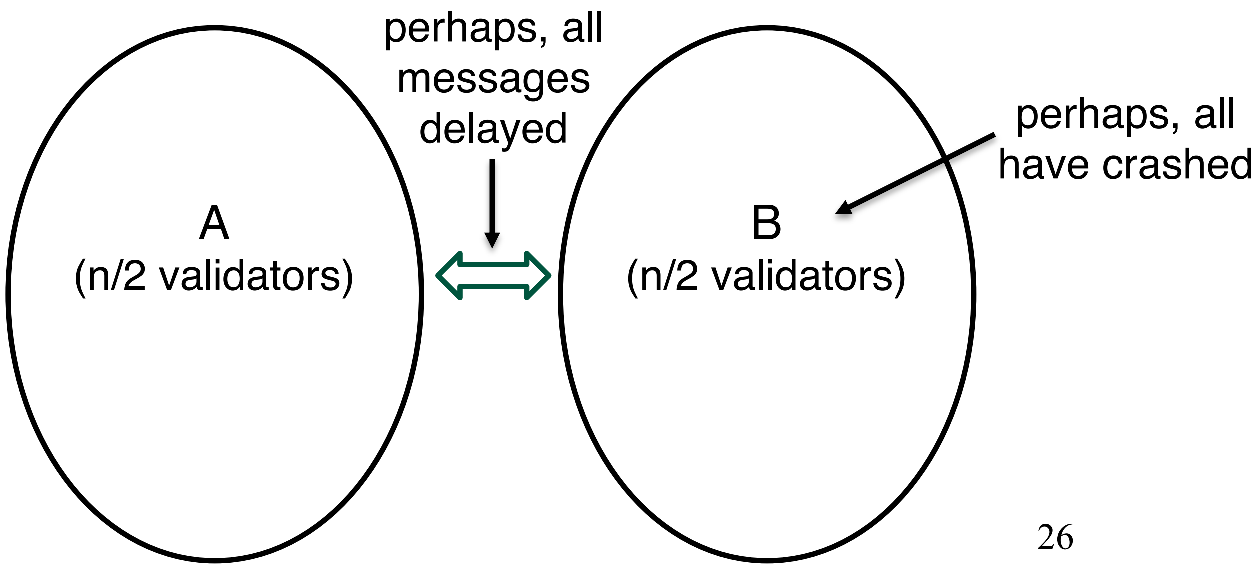
Fact: crash faults + partial synchrony ➔ security threshold < 50%.

– i.e., no hope unless a strict majority of validators are non-faulty

Key challenge: ambiguity between crashed validators and long message delays. [≈ "CAP Principle" from distributed systems]

Suppose: validators in A don't hear from any validators in B for a long time.

- should they finalize any new transactions?



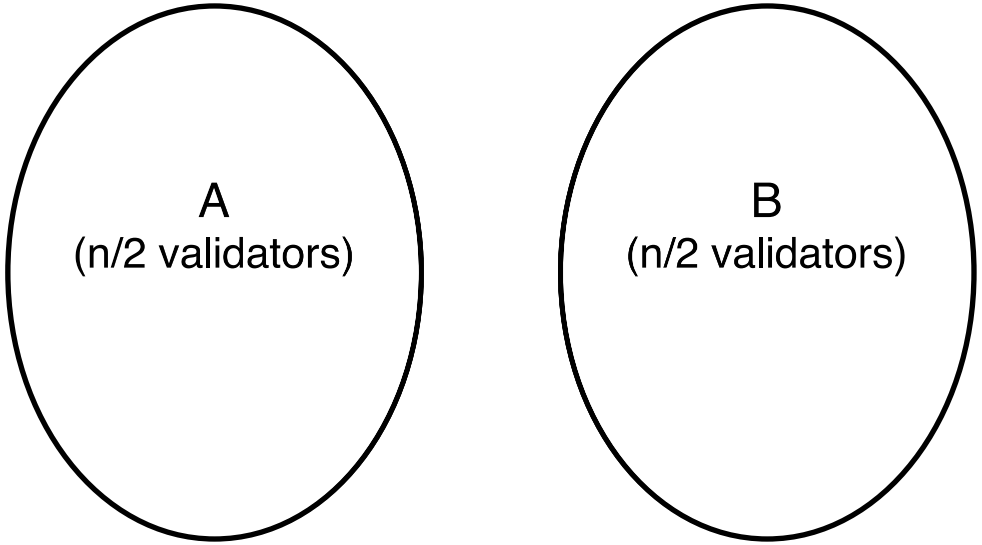
What Is Possible in Partial Synchrony?

Fact: crash faults + partial synchrony ➔ security threshold < 50%.

Suppose: validators in A don't hear from any validators in B for a long time.

- should they finalize any new txs?

Catch-22:

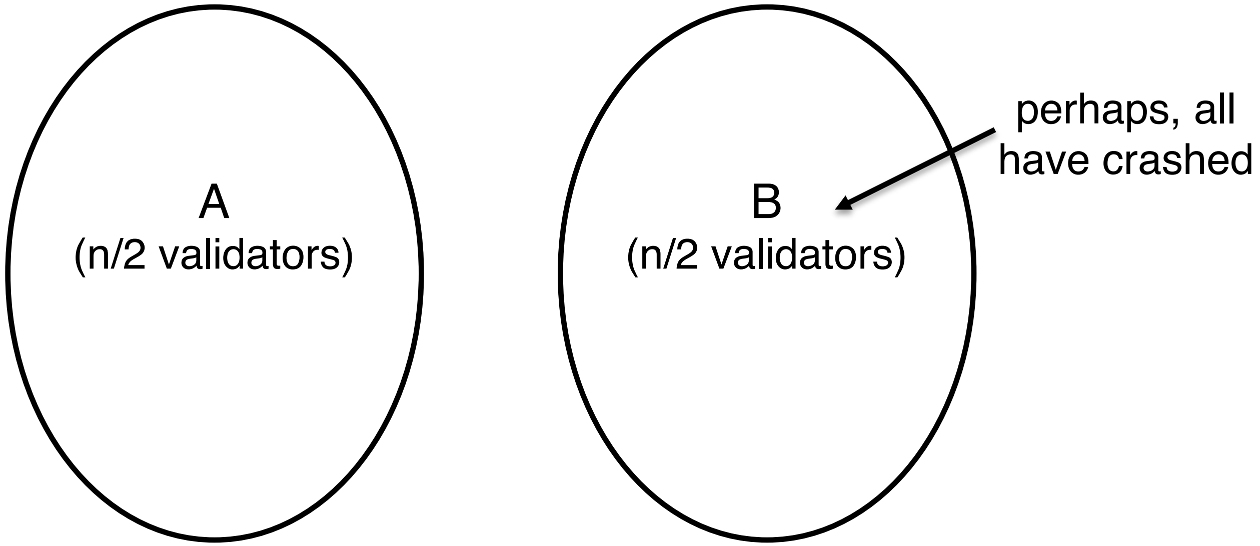


What Is Possible in Partial Synchrony?

Fact: crash faults + partial synchrony → security threshold < 50%.

Suppose: validators in A don't hear from any validators in B for a long time.

- should they finalize any new txs?



Catch-22:

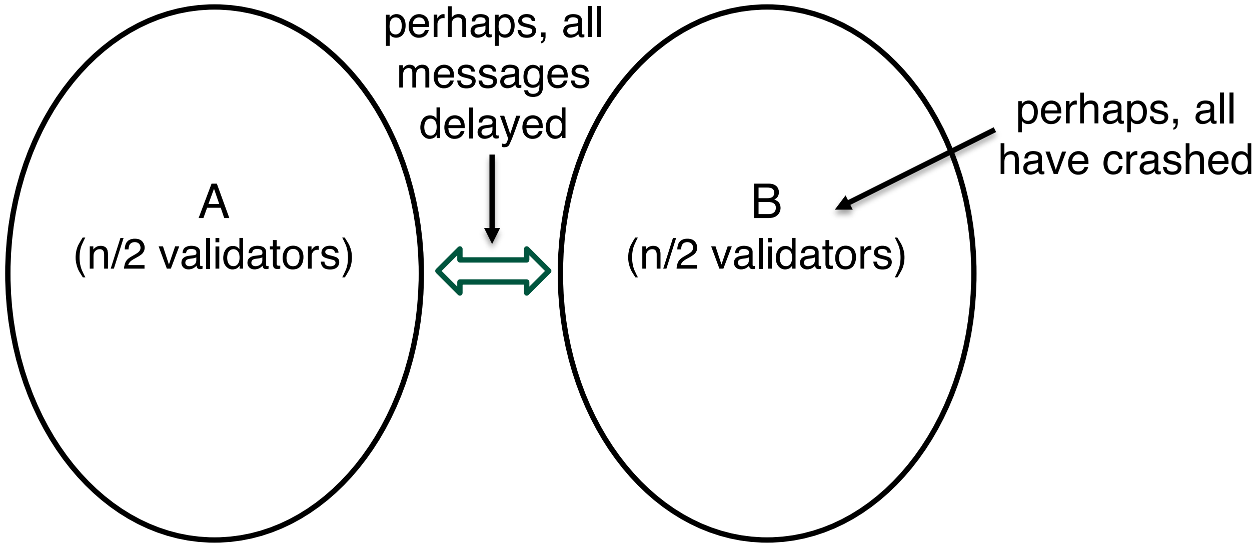
- if validators in A wait → possible liveness violation
  - if post-GST and all validators in B have crashed (will wait forever)

What Is Possible in Partial Synchrony?

Fact: crash faults + partial synchrony ➔ security threshold < 50%.

Suppose: validators in A don't hear from any validators in B for a long time.

- should they finalize any new txs?



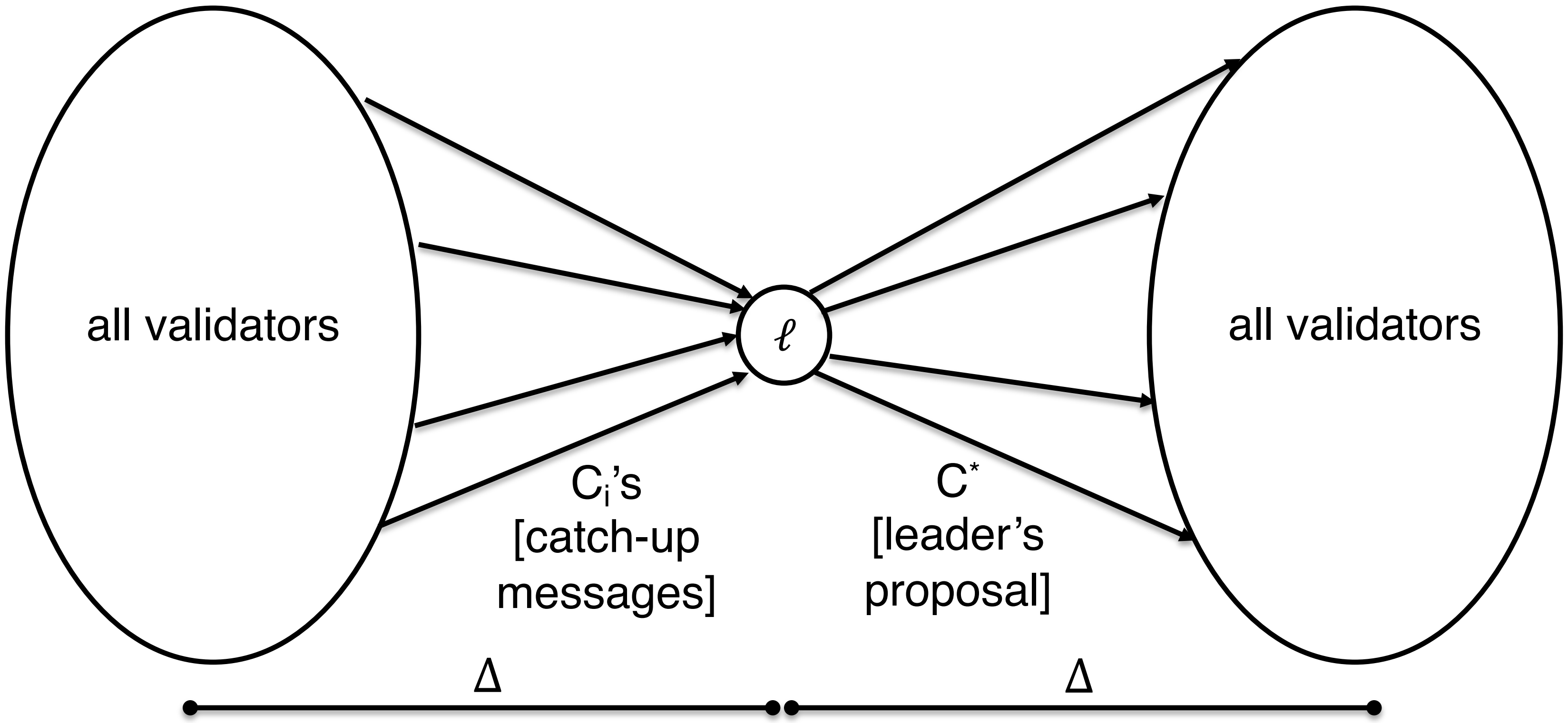
Catch-22:

- if validators in A wait ➔ possible liveness violation
  - if post-GST and all validators in B have crashed (will wait forever)
- if validators in A proceed ➔ possible consistency violation
  - if pre-GST and all messages A ⬄ B have been delayed

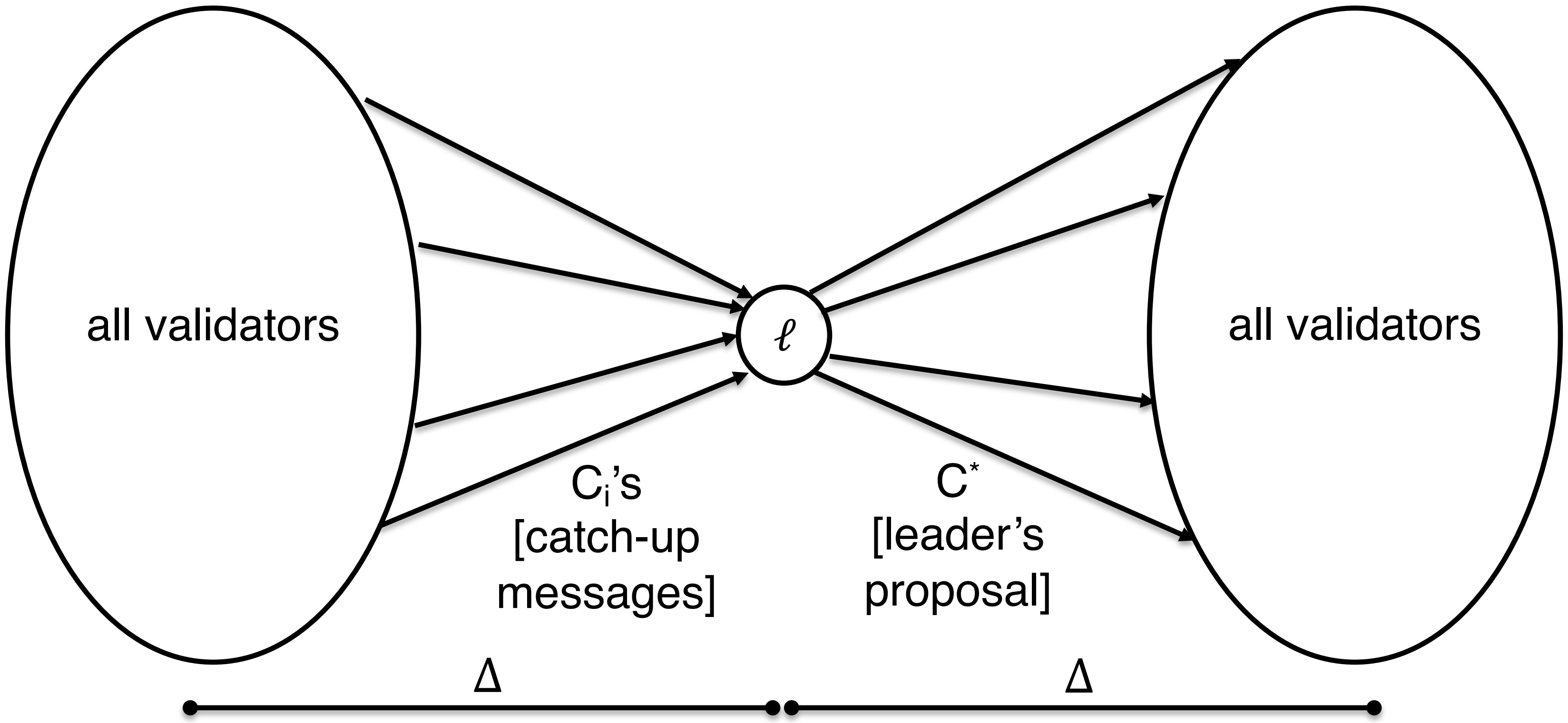
Design Patterns

1. views = repeated attempts to finalize new transactions.
2. leaders = coordinate the transactions proposed in each view.
   – chosen e.g. round-robin (variation: chosen randomly)
3. view may end with non-faulty validators in different states.
   – leader may need to “clean up the mess” left by previous view
4. leader should be as up-to-date as all non-faulty validators.
   – *otherwise, leader’s out-of-date proposal might conflict with the local chains of more up-to-date non-faulty validators*
   – reason for the “catch-up” messages in first half of view in Protocol B
5. distributed computing is hard! [no proof ➔ probably buggy!]

Protocol B: Picture of One View

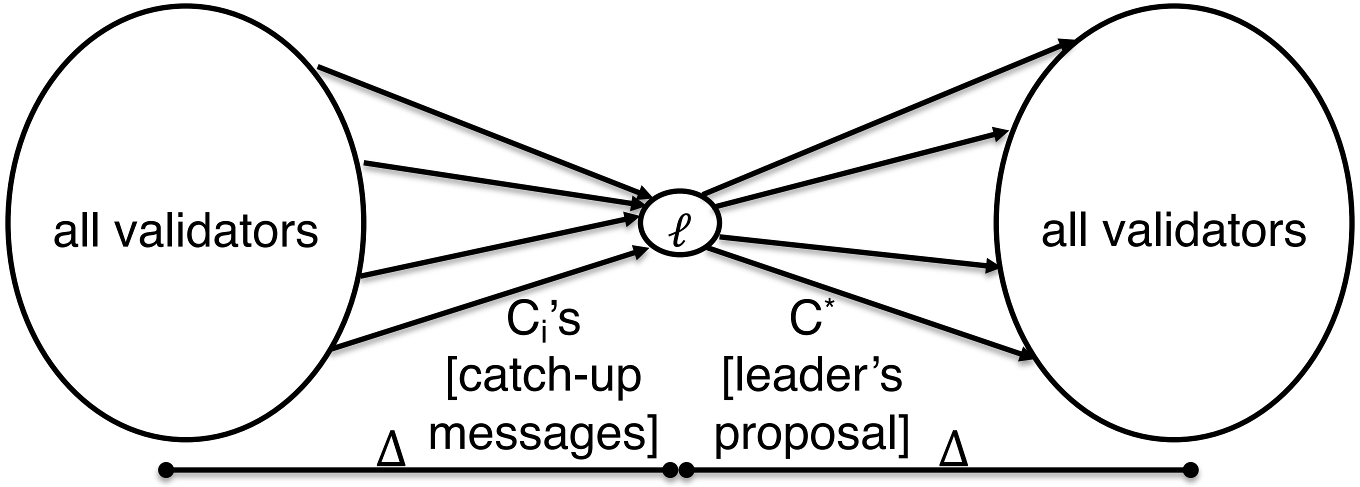


Protocol B: Picture of One View



Problem: in partial synchrony, if pre-GST, no guarantee that the C_i's will reach ℓ before it makes its proposal.

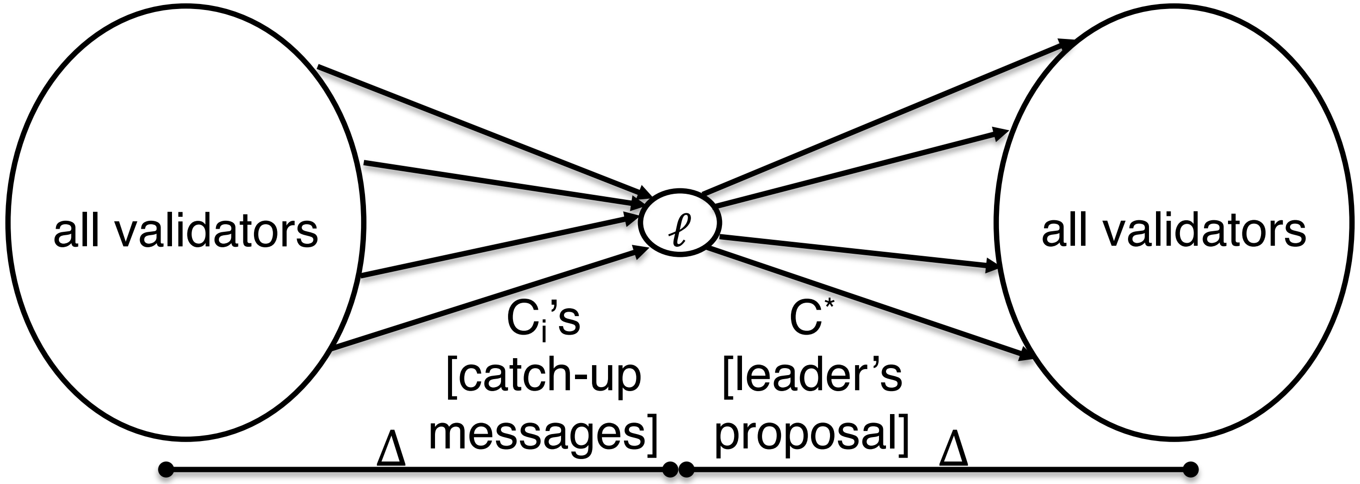
Protocol B: Picture of One View



Problem: in partial synchrony, if pre-GST, no guarantee that the C_i's will reach ℓ before it makes its proposal.

Solution: will add restrictions on when:

Protocol B: Picture of One View

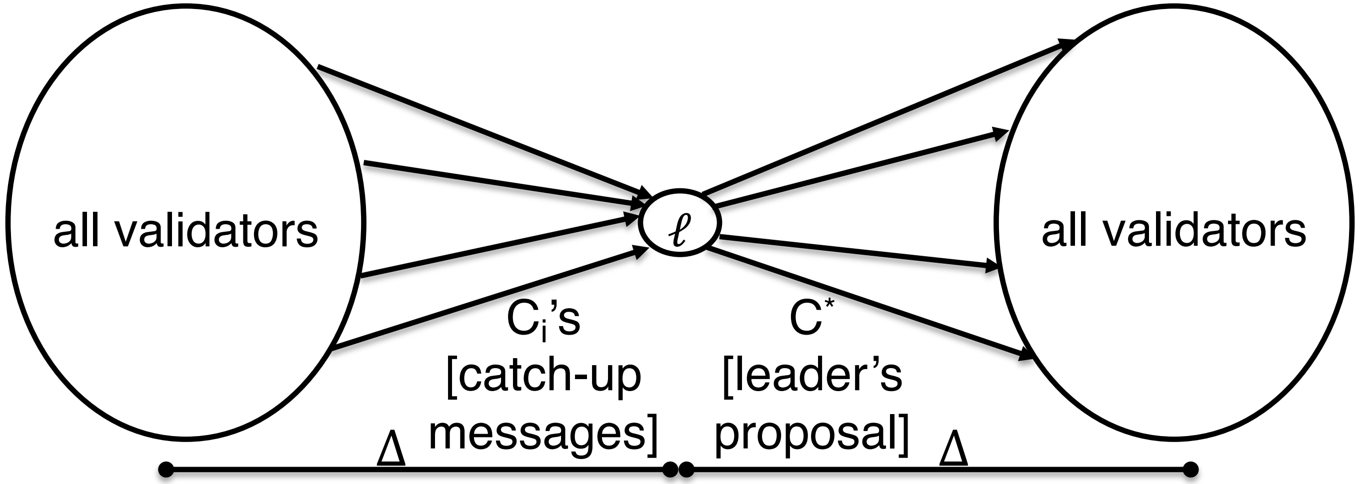


Problem: in partial synchrony, if pre-GST, no guarantee that the C_i's will reach ℓ before it makes its proposal.

Solution: will add restrictions on when:

- a validator can finalize new txs (requires a "write quorum")

Protocol B: Picture of One View

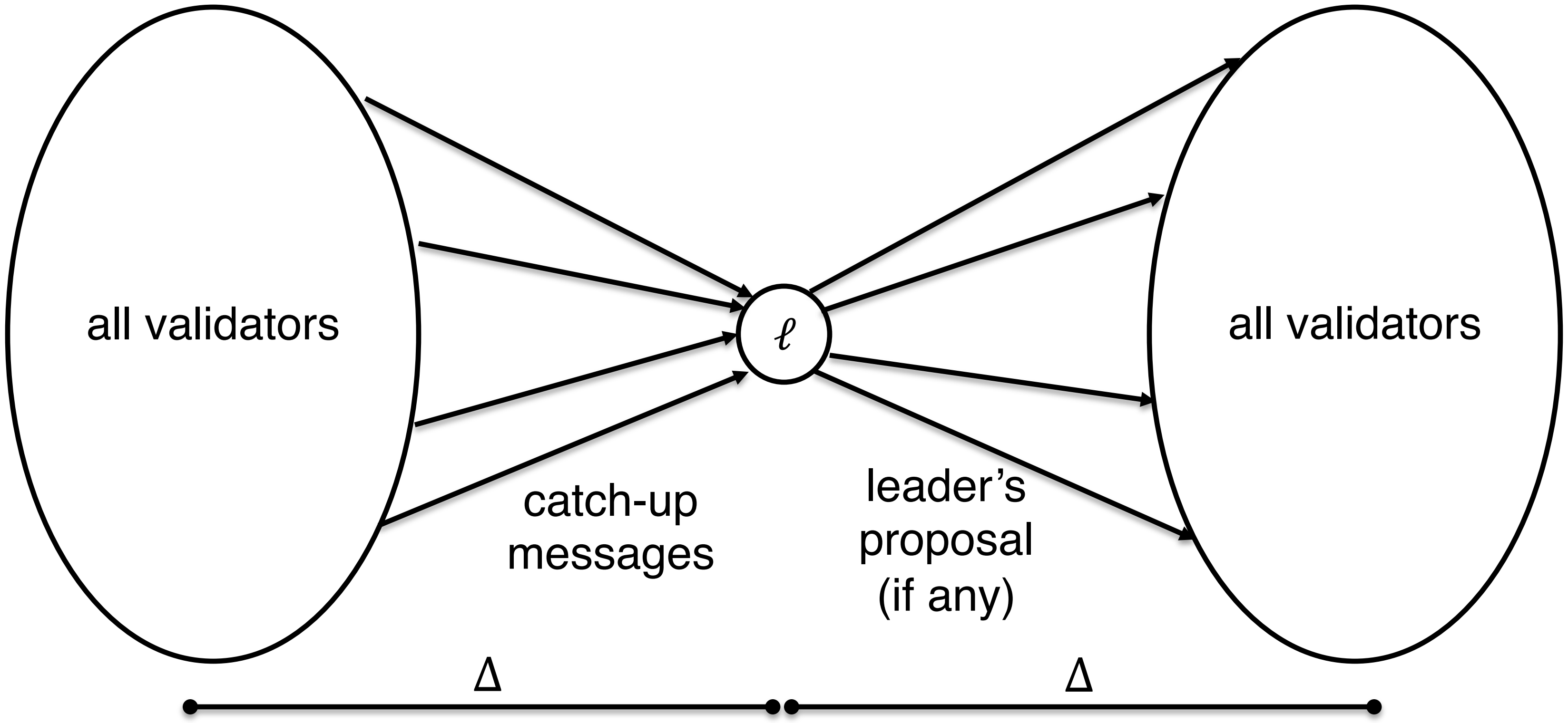


Problem: in partial synchrony, if pre-GST, no guarantee that the C_i's will reach ℓ before it makes its proposal.

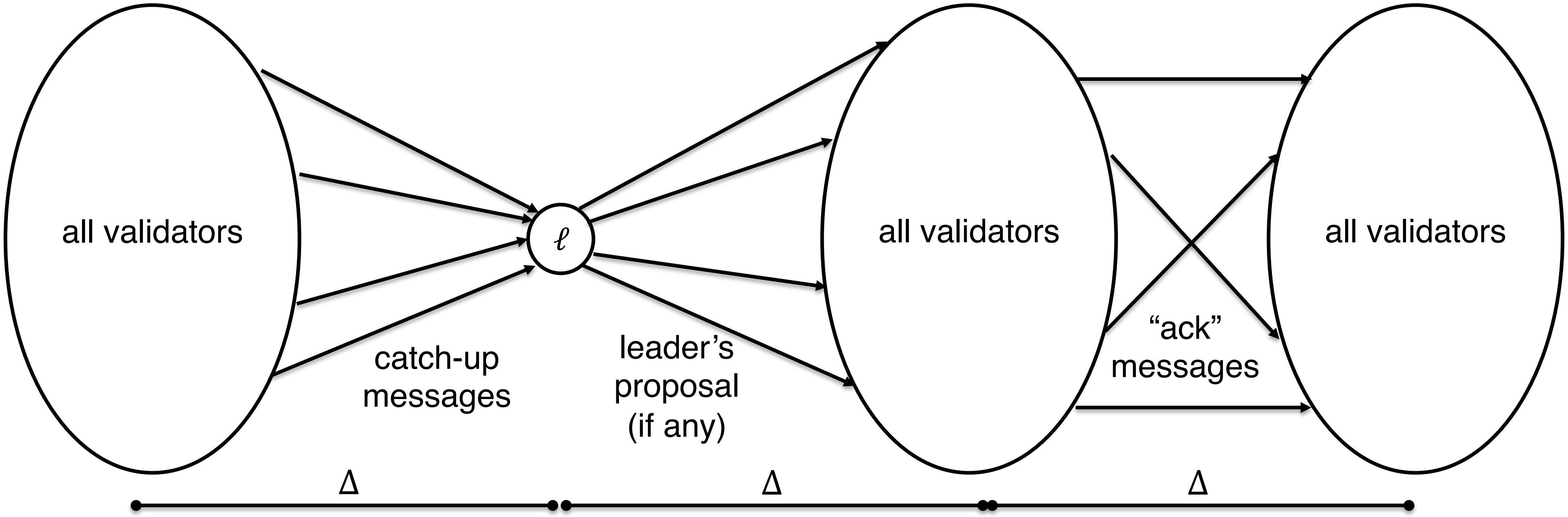
Solution: will add restrictions on when:

- a validator can finalize new txs (requires a "write quorum")
- a leader can make a proposal (requires a "read quorum")

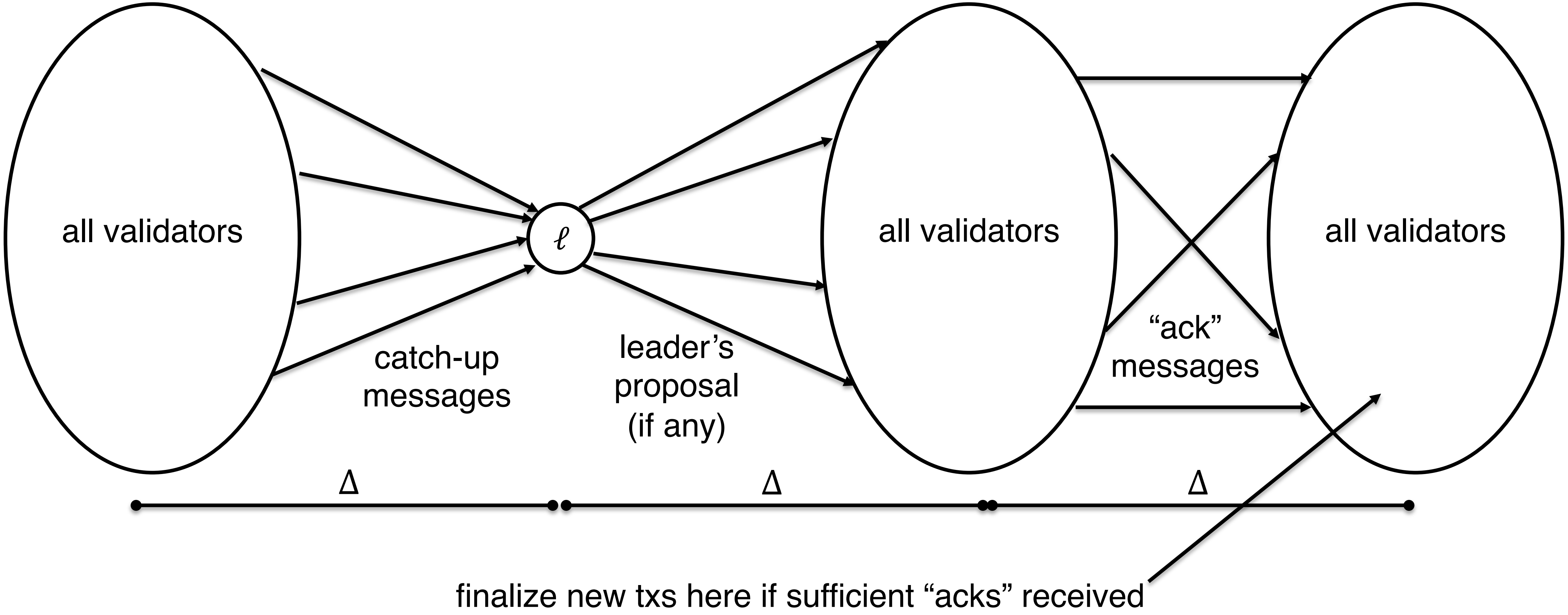
Picture of One View



Picture of One View



Picture of One View



SMR: Crash Faults and Partial Synchrony

Protocol C (≈ Paxos/Raft) [code run by every validator]

SMR: Crash Faults and Partial Synchrony

Protocol C (≈ Paxos/Raft) [code run by every validator]

- define view = 3Δ consecutive timesteps
  - extra phase for validators to assemble write quorums (see below)
- validators take turns as leader (round-robin, one per view)

SMR: Crash Faults and Partial Synchrony

Protocol C (≈ Paxos/Raft) [code run by every validator]

- define view = 3Δ consecutive timesteps
  - extra phase for validators to assemble write quorums (see below)
- validators take turns as leader (round-robin, one per view)
- all messages annotated with view number

SMR: Crash Faults and Partial Synchrony

Protocol C (≈ Paxos/Raft) [code run by every validator]

- define view = 3Δ consecutive timesteps
  - extra phase for validators to assemble write quorums (see below)
- validators take turns as leader (round-robin, one per view)
- all messages annotated with view number
- validator i maintains
  - a local chain C_i (i.e., sequence of blocks) of finalized txs [append-only]
  - a possibly longer chain A_i that it knows about

SMR: Crash Faults and Partial Synchrony

Protocol C (≈ Paxos/Raft) [code run by every validator]

- at time $3\Delta \cdot v$: [i.e., at beginning of view v]
  - each validator i sends its current chain A_i to v's leader ℓ

SMR: Crash Faults and Partial Synchrony

Protocol C (≈ Paxos/Raft) [code run by every validator]

- at time $3\Delta \cdot v$: [i.e., at beginning of view v]
  - each validator i sends its current chain A_i to v's leader ℓ
- at time $3\Delta \cdot v + \Delta$:

SMR: Crash Faults and Partial Synchrony

Protocol C (≈ Paxos/Raft) [code run by every validator]

- at time $3\Delta \cdot v$: [i.e., at beginning of view v]
  - each validator i sends its current chain A_i to v's leader ℓ
- at time $3\Delta \cdot v + \Delta$:
  - if ℓ has received $> n/2$ A_i's by this time (i.e., received a *read quorum*):

SMR: Crash Faults and Partial Synchrony

Protocol C (≈ Paxos/Raft) [code run by every validator]

- at time $3\Delta \cdot v$: [i.e., at beginning of view v]
  - each validator i sends its current chain A_i to v's leader ℓ
- at time $3\Delta \cdot v + \Delta$:
  - if ℓ has received > n/2 A_i's by this time (i.e., received a *read quorum*):
    - let A = most recently proposed of these (i.e., with max view number)

SMR: Crash Faults and Partial Synchrony

Protocol C (≈ Paxos/Raft) [code run by every validator]

- at time $3\Delta \cdot v$: [i.e., at beginning of view v]
  – each validator i sends its current chain A_i to v's leader ℓ
- at time $3\Delta \cdot v + \Delta$:
  – if ℓ has received > n/2 A_i's by this time (i.e., received a *read quorum*):
    - let A = most recently proposed of these (i.e., with max view number)
    - let B := all not-yet-included (in A) valid txs ℓ knows about
    - ℓ sends A^* := (A,B) to all other validators

SMR: Crash Faults and Partial Synchrony

Protocol C (≈ Paxos/Raft) [code run by every validator]

- at time $3\Delta \cdot v + \Delta$:
  - if ℓ has received > n/2 A_i's by this time (i.e., received a *read quorum*):
    - let A = most recently proposed of these (i.e., with max view number)
    - let B := all not-yet-included (in A) valid txs ℓ knows about
    - ℓ sends A^* := (A,B) to all other validators

SMR: Crash Faults and Partial Synchrony

Protocol C (≈ Paxos/Raft) [code run by every validator]

- at time $3\Delta \cdot v + \Delta$:
  - if ℓ has received > n/2 A_i's by this time (i.e., received a *read quorum*):
    - let A = most recently proposed of these (i.e., with max view number)
    - let B := all not-yet-included (in A) valid txs ℓ knows about
    - ℓ sends A^* := (A,B) to all other validators
- at time $3\Delta \cdot v + 2\Delta$:
  - if validator i has received a proposal A^* from ℓ by this time:

SMR: Crash Faults and Partial Synchrony

Protocol C (≈ Paxos/Raft) [code run by every validator]

- at time $3\Delta \cdot v + \Delta$:
  - if ℓ has received > n/2 A_i's by this time (i.e., received a *read quorum*):
    - let A = most recently proposed of these (i.e., with max view number)
    - let B := all not-yet-included (in A) valid txs ℓ knows about
    - ℓ sends A^* := (A,B) to all other validators
- at time $3\Delta \cdot v + 2\Delta$:
  - if validator i has received a proposal A^* from ℓ by this time:
    - i sends "ack A^*" message to all other validators
    - reset $A_i := A^*$

SMR: Crash Faults and Partial Synchrony

Protocol C (≈ Paxos/Raft) [code run by every validator]

- at time $3\Delta \cdot v + 2\Delta$:
  – if validator i has received a proposal A^* from ℓ by this time:
    - i sends "ack A^*" message to all other validators
    - reset $A_i := A^*$

SMR: Crash Faults and Partial Synchrony

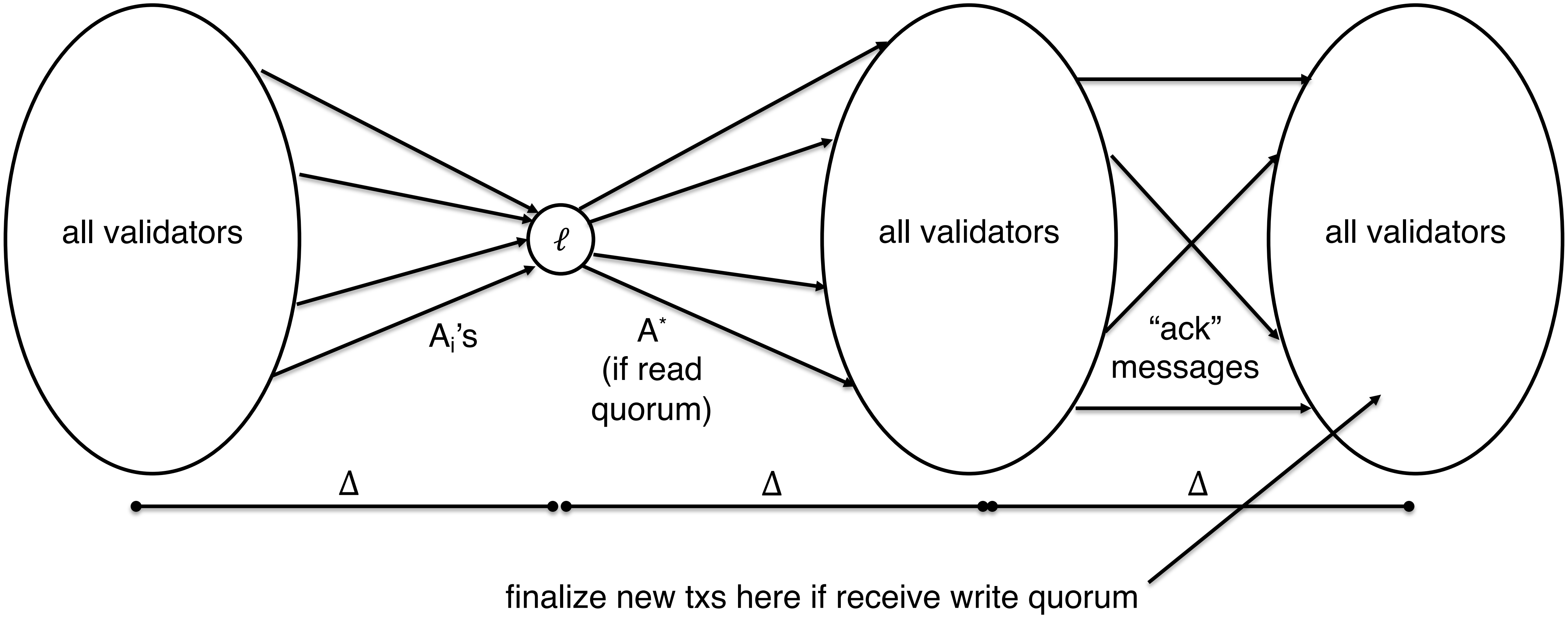
Protocol C (≈ Paxos/Raft) [code run by every validator]

- at time $3\Delta \cdot v + 2\Delta$:
  - if validator i has received a proposal A^* from ℓ by this time:
    - i sends "ack A^*" message to all other validators
    - reset $A_i := A^*$
- at time $3\Delta \cdot v + 3\Delta$:
  - if validator i has received > n/2 "ack A^*" messages (a *write quorum*):
    - reset $C_i := A^*$ (and also $A_i := A^*$, if necessary)

Protocol C

- at time $3\Delta \cdot v$:
  - each validator i sends its current chain A_i to v's leader ℓ
- at time $3\Delta \cdot v + \Delta$:
  - if ℓ has received > n/2 A_i's by this time (i.e., received a *read quorum*):
    - let A = most recently proposed of these (i.e., with max view number)
    - let B := all not-yet-included (in A) valid txs ℓ knows about
    - ℓ sends A^* := (A,B) to all other validators
- at time $3\Delta \cdot v + 2\Delta$:
  - if validator i has received a proposal A^* from ℓ by this time:
    - i sends "ack A^*" message to all other validators
    - reset $A_i := A^*$
- at time $3\Delta \cdot v + 3\Delta$:
  - if validator i has received > n/2 "ack A^*" messages (a *write quorum)*:
    - reset $C_i := A^*$ (and also $A_i := A^*$, if necessary)

Picture of One View



Protocol C: Proof of Consistency

Protocol C: Proof of Consistency

Key claim: for each view v:

1. if any non-faulty C_i's get updated in this view, all get updated to the proposal A^* made be v's leader. [immediate, see code]

Protocol C: Proof of Consistency

Key claim: for each view v:

1. if any non-faulty C_i's get updated in this view, all get updated to the proposal A^* made be v's leader. [immediate, see code]
2. in this case (i.e., ≥1 update in v), all updates to non-faulty C_i's in views v' > v are to chains that extend A^*. [need to prove]

Protocol C: Proof of Consistency

Key claim: for each view v:

1. if any non-faulty C_i's get updated in this view, all get updated to the proposal A^* made be v's leader. [immediate, see code]
2. in this case (i.e., ≥1 update in v), all updates to non-faulty C_i's in views v' > v are to chains that extend A^*. [need to prove]

Note: Implies consistency:

Protocol C: Proof of Consistency

Key claim: for each view v:

1. if any non-faulty C_i's get updated in this view, all get updated to the proposal A^* made be v's leader. [immediate, see code]
2. in this case (i.e., ≥1 update in v), all updates to non-faulty C_i's in views v' > v are to chains that extend A^*. [need to prove]

Note: Implies consistency:

- (2) ➔ each C_i is append-only (finalized txs never rolled back)

Protocol C: Proof of Consistency

Key claim: for each view v:

1. if any non-faulty C_i's get updated in this view, all get updated to the proposal A^* made be v's leader. [immediate, see code]
2. in this case (i.e., ≥1 update in v), all updates to non-faulty C_i's in views v' > v are to chains that extend A^*. [need to prove]

Note: Implies consistency:

- (2) ➔ each C_i is append-only (finalized txs never rolled back)
- (1) ➔ simultaneous updates (i.e., in same view) are consistent

Protocol C: Proof of Consistency

Key claim: for each view v:

1. if any non-faulty C_i's get updated in this view, all get updated to the proposal A^* made be v's leader. [immediate, see code]
2. in this case (i.e., ≥1 update in v), all updates to non-faulty C_i's in views v' > v are to chains that extend A^*. [need to prove]

Note: Implies consistency:

- (2) ➔ each C_i is append-only (finalized txs never rolled back)
- (1) ➔ simultaneous updates (i.e., in same view) are consistent
- (2) ➔ every update extends all updates from all previous views

Protocol C: Proof of Consistency

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ all updates in views v' > v are to chains that extend A^*.

Protocol C: Proof of Consistency

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ all updates in views v' > v are to chains that extend A^*.

- i updated to C_i in view v ➔ let S = validators in its write quorum

Protocol C: Proof of Consistency

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ all updates in views v' > v are to chains that extend A^*.

- i updated to C_i in view v ➔ let S = validators in its write quorum
- note: all j in S reset $A_j := A^*$ in this view [A^* is a view-v proposal]

Protocol C: Proof of Consistency

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ all updates in views v' > v are to chains that extend A^*.

- i updated to C_i in view v ➔ let S = validators in its write quorum
- note: all j in S reset $A_j := A^*$ in this view [A^* is a view-v proposal]

Consider view v+1:

Protocol C: Proof of Consistency

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ all updates in views v' > v are to chains that extend A^*.

- i updated to C_i in view v ➔ let S = validators in its write quorum
- note: all j in S reset A_j := A^* in this view [A^* is a view-v proposal]

Consider view v+1:

- if leader makes proposal A' ➔ let T = validators in read quorum

Protocol C: Proof of Consistency

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ all updates in views v' > v are to chains that extend A^*.

- i updated to C_i in view v ➔ let S = validators in its write quorum
- note: all j in S reset $A_j := A^*$ in this view [A^* is a view-v proposal]

Consider view v+1:

- if leader makes proposal A' ➔ let T = validators in read quorum
- *quorum intersection:* because |S|,|T|>n/2, S and T overlap

Protocol C: Proof of Consistency

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ all updates in views v' > v are to chains that extend A^*.

- i updated to C_i in view v ➔ let S = validators in its write quorum
- note: all j in S reset $A_j := A^*$ in this view [A^* is a view-v proposal]

Consider view v+1:

- if leader makes proposal A' ➔ let T = validators in read quorum
- *quorum intersection:* because |S|,|T|>n/2, S and T overlap
- leader of view receives A^* from at least one validator

Protocol C: Proof of Consistency

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ all updates in views v' > v are to chains that extend A^*.

- i updated to C_i in view v ➔ let S = validators in its write quorum
- note: all j in S reset $A_j := A^*$ in this view [A^* is a view-v proposal]

Consider view v+1:

- if leader makes proposal A' ➔ let T = validators in read quorum
- *quorum intersection:* because |S|,|T|>n/2, S and T overlap
- leader of view receives A^* from at least one validator
- leader's proposal will extend A^* (nothing could be more recent)

Protocol C: Proof of Consistency

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ all updates in views v' > v are to chains that extend A^*.

- i updated to C_i in view v ➔ let S = validators in its write quorum
- note: all j in S reset $A_j := A^*$ in this view [A^* is a view-v proposal]

Consider view v+2:

- if leader makes proposal A' ➔ let T = validators in read quorum
- *quorum intersection:* because |S|,|T|>n/2, S and T overlap
- leader receives an A_j from a view ≥v from at least one validator
- leader's proposal will extend A^* (everything from view ≥v does)

Protocol C: Proof of Consistency

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ all updates in views v' > v are to chains that extend A^*.

- i updated to C_i in view v ➔ let S = validators in its write quorum
- note: all j in S reset $A_j := A^*$ in this view [A^* is a view-v proposal]

In general (by induction on v' > v):

- if leader makes proposal A' ➔ let T = validators in read quorum
- *quorum intersection:* because |S|,|T|>n/2, S and T overlap
- leader receives an A_j from a view ≥v from at least one validator
- leader's proposal will extend A^* (everything from view ≥v does)

Protocol C: Proof of Liveness

Suppose tx z known to some non-faulty validator i at time step t.

Protocol C: Proof of Liveness

Suppose tx z known to some non-faulty validator i at time step t.

- let v be the next view that begins after GST and for which i is the leader (must exist, why?)

Protocol C: Proof of Liveness

Suppose tx z known to some non-faulty validator i at time step t.

- let v be the next view that begins after GST and for which i is the leader (must exist, why?)
- post-GST ➔ by time $3\Delta \cdot v + \Delta$, i will receive A_j's from all not-yet-crashed validators (of which there are > n/2 !)

Protocol C: Proof of Liveness

Suppose tx z known to some non-faulty validator i at time step t.

- let v be the next view that begins after GST and for which i is the leader (must exist, why?)
- post-GST ➔ by time $3\Delta \cdot v + \Delta$, i will receive A_j's from all not-yet-crashed validators (of which there are > n/2 !)
  - i's will make a proposal A* := (A,B) in view v will include the tx z
    - if not already in A, will put it in the new block B

Protocol C: Proof of Liveness

Suppose tx z known to some non-faulty validator i at time step t.

- let v be the next view that begins after GST and for which i is the leader (must exist, why?)
- post-GST ➔ by time $3\Delta \cdot v + \Delta$, i will receive A_j's from all not-yet-crashed validators (of which there are > n/2 !)
  - i's will make a proposal A* := (A,B) in view v will include the tx z
    - if not already in A, will put it in the new block B
- post-GST ➔ all non-faulty validators get A* by $3\Delta \cdot v + 2\Delta$

Protocol C: Proof of Liveness

Suppose tx z known to some non-faulty validator i at time step t.

- let v be the next view that begins after GST and for which i is the leader (must exist, why?)
- post-GST ➔ by time $3\Delta \cdot v + \Delta$, i will receive A_j's from all not-yet-crashed validators (of which there are > n/2 !)
  - i's will make a proposal A* := (A,B) in view v will include the tx z
    - if not already in A, will put it in the new block B
- post-GST ➔ all non-faulty validators get A* by $3\Delta \cdot v + 2\Delta$
  - all send "ack A* messages at that time

Protocol C: Proof of Liveness

Suppose tx z known to some non-faulty validator i at time step t.

- let v be the next view that begins after GST and for which i is the leader

- post-GST ➔ by time $3\Delta \cdot v + \Delta$, i will receive A_j's from all not-yet-crashed validators (of which there are > n/2 !)
  - i's will make a proposal A* := (A,B) in view v will include the tx z
- post-GST ➔ all non-faulty validators get A* by $3\Delta \cdot v + 2\Delta$
  - all send "ack A* messages at that time

Protocol C: Proof of Liveness

Suppose tx z known to some non-faulty validator i at time step t.

– let v be the next view that begins after GST and for which i is the leader

- post-GST ➔ by time $3\Delta \cdot v + \Delta$, i will receive A_j's from all not-yet-crashed validators (of which there are > n/2 !)

  – i's will make a proposal A* := (A,B) in view v will include the tx z

- post-GST ➔ all non-faulty validators get A* by $3\Delta \cdot v + 2\Delta$

  – all send "ack A* messages at that time

- post-GST ➔ all non-faulty validators get all these ack messages by time $3\Delta \cdot v + 3\Delta$ (of which there are > n/2 !)

Protocol C: Proof of Liveness

Suppose tx z known to some non-faulty validator i at time step t.

- let v be the next view that begins after GST and for which i is the leader

- post-GST ➔ by time $3\Delta \cdot v + \Delta$, i will receive A_j's from all not-yet-crashed validators (of which there are > n/2 !)
  - i's will make a proposal A* := (A,B) in view v will include the tx z
- post-GST ➔ all non-faulty validators get A* by $3\Delta \cdot v + 2\Delta$
  - all send "ack A^* messages at that time
- post-GST ➔ all non-faulty validators get all these ack messages by time $3\Delta \cdot v + 3\Delta$ (of which there are > n/2 !)
  - ➔ all such validators set C_j := A^* at this time