Lecture #6: Solving SMR with Byzantine Faults in Partial Synchrony: The Essence of Tendermint

COMS 4995-001:
The Science of Blockchains

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

Tim Roughgarden

Goals for Lecture #6

1. The Tendermint protocol.
   - basis of Cosmos and several other blockchain protocols
   - available more or less off-the-shelf to build on

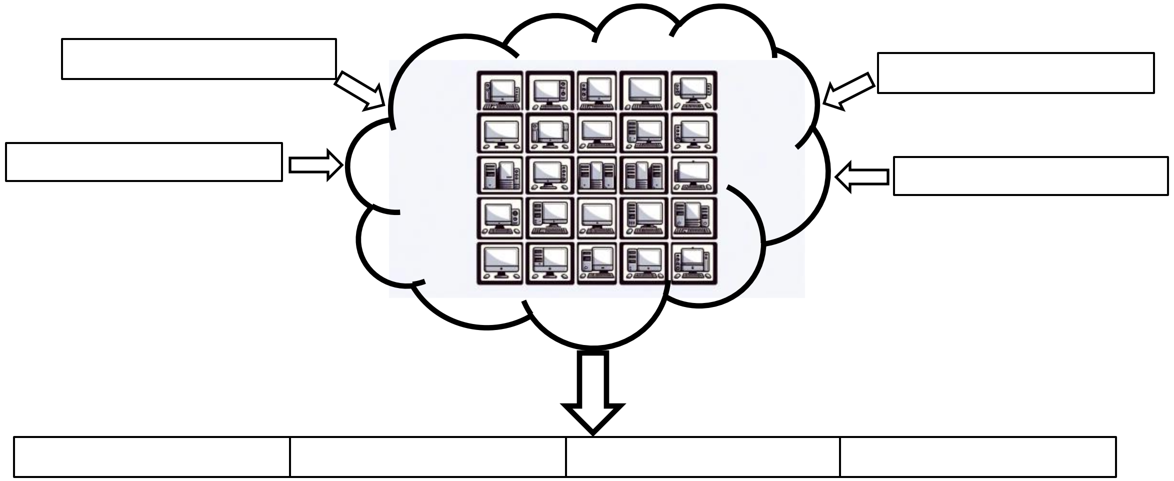
2. Analysis of the Tendermint protocol.
   - achieves optimal Byzantine fault-tolerance in partial synchrony
   - similar structure to Paxos/Raft analysis, but several new ideas

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

crash faults + synchrony [security threshold ≈ 100%]	crash faults + partially synchrony [security threshold = 50%]	Byzantine faults + partially synchrony [security threshold ≤ 33%]

easier → harder

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

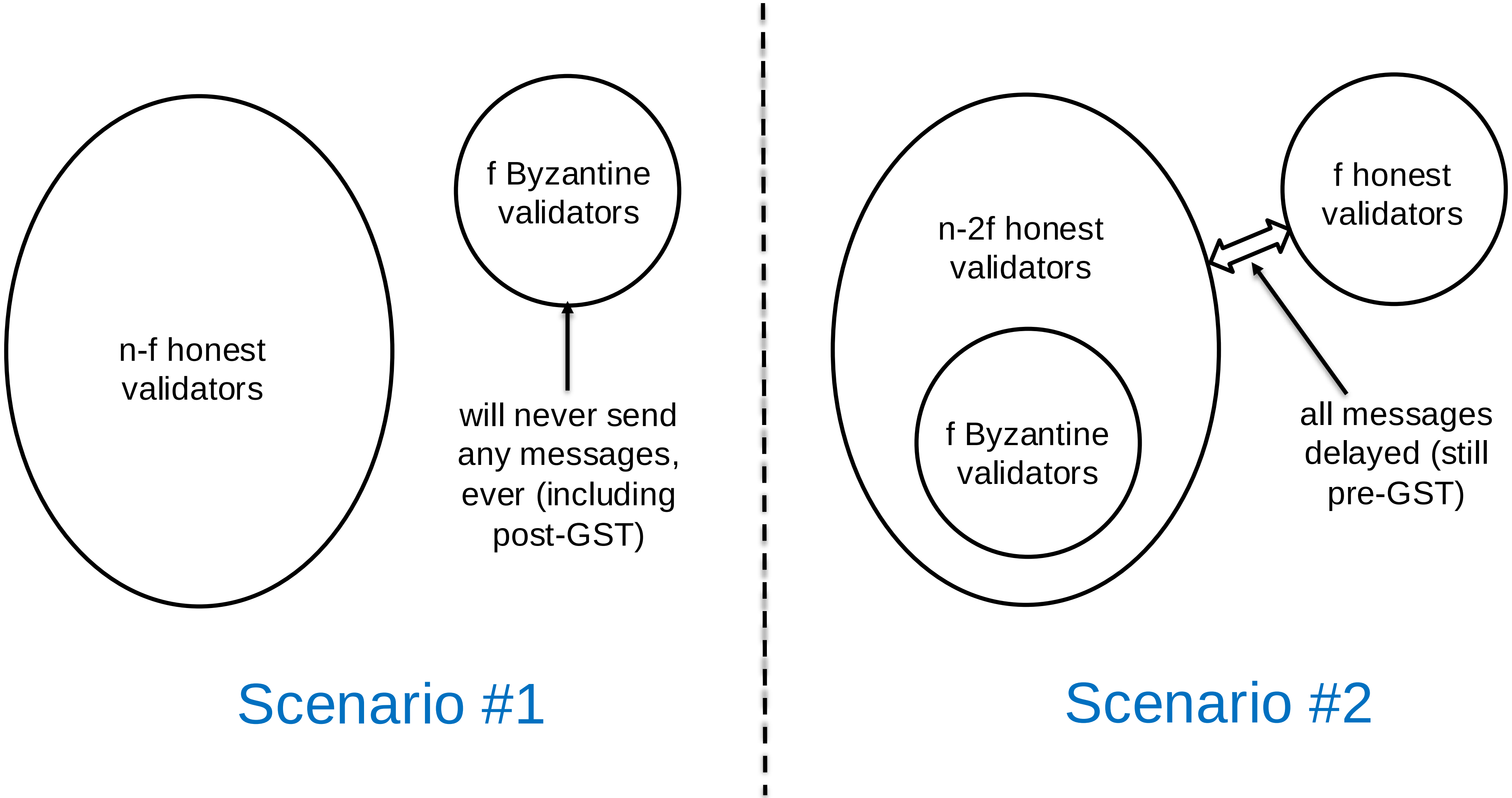
Lecture #4: Paxos/Raft, optimal crash-fault tolerance in partial synchrony.

Lecture #5: can’t achieve >33% Byzantine fault-tolerance in partial synchrony.

Key Ideas in Tendermint

Recall: need to assume < n/3 Byzantine validators.

Post-GST Crashes or Pre-GST Delays?



Key Ideas in Tendermint

Recall: need to assume < n/3 Byzantine validators.

Idea #1: every validator signs every message it sends.

- assume all validators know each others public keys (+ IDs + IP addrs)
- called a "public key infrastructure (PKI)" assumption

Key Ideas in Tendermint

Recall: need to assume < n/3 Byzantine validators.

Idea #1: every validator signs every message it sends.

Idea #2: increase all quorum sizes to > 2n/3 validators.

Key Ideas in Tendermint

Recall: need to assume < n/3 Byzantine validators.

Idea #1: every validator signs every message it sends.

Idea #2: increase all quorum sizes to > 2n/3 validators.

- note: does not immediately threaten liveness

Key Ideas in Tendermint

Recall: need to assume < n/3 Byzantine validators.

Idea #1: every validator signs every message it sends.

Idea #2: increase all quorum sizes to > 2n/3 validators.

- note: does not immediately threaten liveness
- key point: any two quorums have non-faulty validator in common

Key Ideas in Tendermint

Recall: need to assume < n/3 Byzantine validators.

Idea #1: every validator signs every message it sends.

Idea #2: increase all quorum sizes to > 2n/3 validators.

- note: does not immediately threaten liveness
- key point: any two quorums have non-faulty validator in common
- consequence #1: leader proposal that respects a read quorum is up-to-date (includes non-faulty participants from all previous write quorums)

Key Ideas in Tendermint

Recall: need to assume < n/3 Byzantine validators.

Idea #1: every validator signs every message it sends.

Idea #2: increase all quorum sizes to > 2n/3 validators.

- note: does not immediately threaten liveness
- key point: any two quorums have non-faulty validator in common
- consequence #1: leader proposal that respects a read quorum is up-to-date (includes non-faulty participants from all previous write quorums)
  - prevents inconsistencies between updates in different views

Key Ideas in Tendermint

Recall: need to assume < n/3 Byzantine validators.

Idea #1: every validator signs every message it sends.

Idea #2: increase all quorum sizes to > 2n/3 validators.

- key point: any two quorums have non-faulty validator in common
- consequence #1: leader proposal that respects a read quorum is up-to-date (includes non-faulty participants from all previous write quorums)
  - prevents inconsistencies between updates in different views
- consequence #2: can't have write quorums for different proposals in the same view (even with equivocating leader and Byzantine validators)

Key Ideas in Tendermint

Recall: need to assume < n/3 Byzantine validators.

Idea #1: every validator signs every message it sends.

Idea #2: increase all quorum sizes to > 2n/3 validators.

- key point: any two quorums have non-faulty validator in common
- consequence #1: leader proposal that respects a read quorum is up-to-date (includes non-faulty participants from all previous write quorums)
  - prevents inconsistencies between updates in different views
- consequence #2: can't have write quorums for different proposals in the same view (even with equivocating leader and Byzantine validators)
  - prevents inconsistencies between updates in the same view

Key Ideas in Tendermint

Recall: need to assume $< n/3$ Byzantine validators.

Idea #1: every validator signs every message it sends.

Idea #2: increase all quorum sizes to $> 2n/3$ validators.

Idea #3: before acking a proposal, validator assembles its own read quorum (recorded with a "quorum certificate (QC)").

- attestations by $> 2n/3$ validators that proposal appears up-to-date

Key Ideas in Tendermint

Recall: need to assume < n/3 Byzantine validators.

Idea #1: every validator signs every message it sends.

Idea #2: increase all quorum sizes to > 2n/3 validators.

Idea #3: before acking a proposal, validator assembles its own read quorum (recorded with a "quorum certificate (QC)").

- attestations by > 2n/3 validators that proposal appears up-to-date
- reason: can't trust Byzantine leader to assemble/respect a read quorum

Key Ideas in Tendermint

Recall: need to assume < n/3 Byzantine validators.

Idea #1: every validator signs every message it sends.

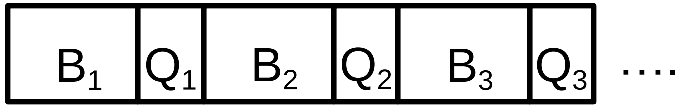
Idea #2: increase all quorum sizes to > 2n/3 validators.

Idea #3: before acking a proposal, validator assembles its own read quorum (recorded with a "quorum certificate (QC)").

- attestations by > 2n/3 validators that proposal appears up-to-date
- reason: can't trust Byzantine leader to assemble/respect a read quorum
- will add extra round to each view (not strictly necessary)

Key Ideas in Tendermint

Recall: need to assume < n/3 Byzantine validators.



Idea #1: every validator signs every message it sends.

Idea #2: increase all quorum sizes to > 2n/3 validators.

Idea #3: before acking a proposal, validator assembles its own read quorum (recorded with a "quorum certificate (QC)").

- attestations by > 2n/3 validators that proposal appears up-to-date
- reason: can't trust Byzantine leader to assemble/respect a read quorum
- will add extra round to each view (not strictly necessary)
- QCs included as metadata alongside blocks

The Tendermint Protocol

Protocol D (≈ Tendermint) [code run by every validator]

The Tendermint Protocol

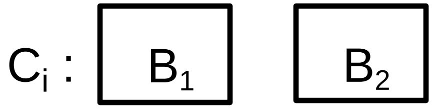
Protocol D (≈ Tendermint) [code run by every validator]

- define view = 4Δ consecutive timesteps
  - – *extra phase for validators to assemble read quorum before acking*

The Tendermint Protocol

Protocol D (≈ Tendermint) [code run by every validator]

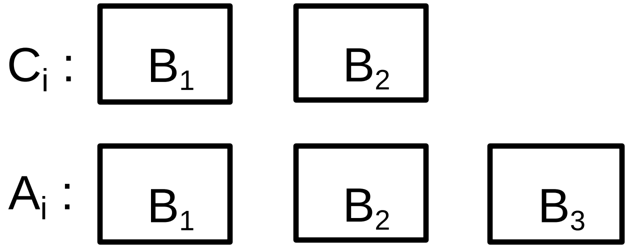
- define view = 4Δ consecutive timesteps
  - *extra phase for validators to assemble read quorum before acking*
- validator i maintains
  - a local chain C_i of finalized txs [append-only]



The Tendermint Protocol

Protocol D (≈ Tendermint) [code run by every validator]

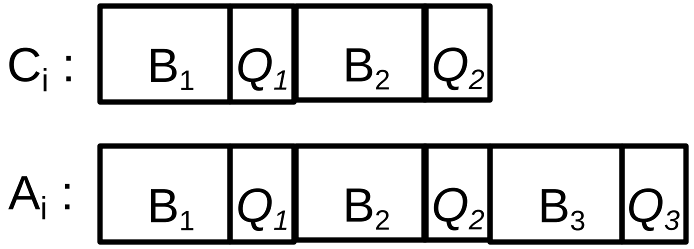
- define view = 4Δ consecutive timesteps
  - *extra phase for validators to assemble read quorum before acking*
- validator i maintains
  - a local chain C_i of finalized txs [append-only]
  - a possibly longer chain A_i that it knows about



The Tendermint Protocol

Protocol D (≈ Tendermint) [code run by every validator]

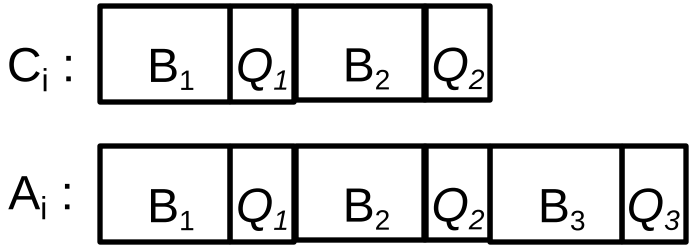
- define view = 4Δ consecutive timesteps
  - *extra phase for validators to assemble read quorum before acking*
- validator i maintains
  - a local chain C_i of finalized txs [append-only]
  - a possibly longer chain A_i that it knows about
  - *a QC for each block of C_i and A_i*



The Tendermint Protocol

Protocol D (≈ Tendermint) [code run by every validator]

- define view = 4Δ consecutive timesteps
  - *extra phase for validators to assemble read quorum before acking*
- validator i maintains
  - a local chain C_i of finalized txs [append-only]
  - a possibly longer chain A_i that it knows about
  - *a QC for each block of C_i and A_i*

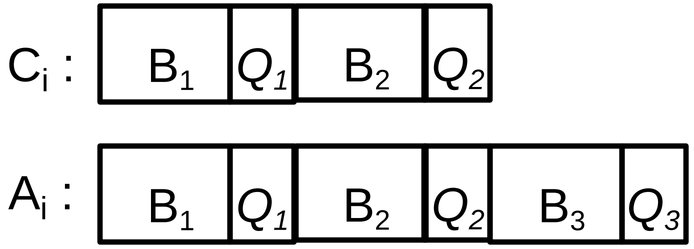


- validators take turns as leader (round-robin, one per view)

The Tendermint Protocol

Protocol D (≈ Tendermint) [code run by every validator]

- define view = 4Δ consecutive timesteps
  - *extra phase for validators to assemble read quorum before acking*
- validator i maintains
  - a local chain C_i of finalized txs [append-only]
  - a possibly longer chain A_i that it knows about
  - *a QC for each block of C_i and A_i*

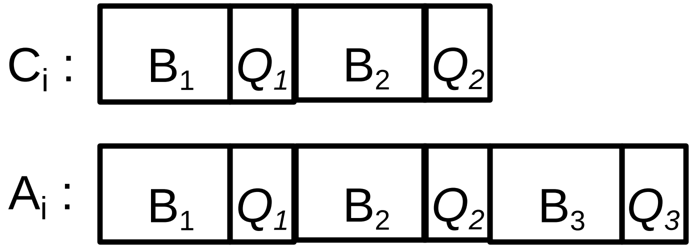


- validators take turns as leader (round-robin, one per view)
- *validators sign all messages*

The Tendermint Protocol

Protocol D (≈ Tendermint) [code run by every validator]

- define view = 4Δ consecutive timesteps
  - *extra phase for validators to assemble read quorum before acking*
- validator i maintains
  - a local chain C_i of finalized txs [append-only]
  - a possibly longer chain A_i that it knows about
  - *a QC for each block of C_i and A_i*



- validators take turns as leader (round-robin, one per view)
- *validators sign all messages*
- all messages annotated with current view number

The Tendermint Protocol (con’d)

Protocol D (≈ Tendermint) [code run by every validator]

- at time $4\Delta \cdot v$: [i.e., at beginning of view v]
  – each validator i sends its current chain A_i to v’s leader ℓ

The Tendermint Protocol (con'd)

Protocol D (≈ Tendermint) [code run by every validator]

- at time $4\Delta \cdot v$: [i.e., at beginning of view v]
  - each validator i sends its current chain A_i to v's leader ℓ
- at time $4\Delta \cdot v + \Delta$: [performed only by v's leader ℓ]
  - let A = of the A_i's received, the most recently created one
  - let B := all not-yet-included (in A) valid txs ℓ knows about
  - ℓ sends proposal (A,B) to all other validators

The Tendermint Protocol (con’d)

Protocol D (≈ Tendermint) [code run by every validator]

- at time $4\Delta \cdot v + \Delta$: [performed only by v’s leader ℓ]
  - let A = of the A_i’s received, the most recently created one
  - let B := all not-yet-included (in A) valid txs ℓ knows about
  - ℓ sends proposal (A,B) to all other validators

The Tendermint Protocol (con'd)

Protocol D (≈ Tendermint) [code run by every validator]

- at time $4\Delta \cdot v + \Delta$: [performed only by v's leader ℓ]
  - let A = of the A_i's received, the most recently created one
  - let B := all not-yet-included (in A) valid txs ℓ knows about
  - ℓ sends proposal (A,B) to all other validators
- at time $4\Delta \cdot v + 2\Delta$:
  - *if validator i receives a proposal (A,B) from ℓ with A = A_i or with A more recent than A_i by this time:*
    - *send "(A,B) is up-to-date" message to all validators*

The Tendermint Protocol (con’d)

Protocol D (≈ Tendermint) [code run by every validator]

- at time $4\Delta \cdot v + 2\Delta$:
  - *if validator i receives a proposal (A,B) from ℓ with A = A_i or with A more recent than A_i by this time:*
    - *send “(A,B) is up-to-date” message to all validators*

The Tendermint Protocol (con'd)

Protocol D (≈ Tendermint) [code run by every validator]

- at time $4\Delta \cdot v + 2\Delta$:
  - *if validator i receives a proposal (A,B) from ℓ with A = A_i or with A more recent than A_i by this time:*
    - *send "(A,B) is up-to-date" message to all validators*
- at time $4\Delta \cdot v + 3\Delta$:
  - *if validator i has heard > 2n/3 "up-to-date" msgs for (A,B) by this time:*
    - *package these messages into a quorum certificate (QC), Q*
    - send "ack (A,B,*Q*)" message to all validators
    - reset A_i := (A,B,*Q*)

The Tendermint Protocol (con’d)

Protocol D (≈ Tendermint) [code run by every validator]

- at time $4\Delta \cdot v + 3\Delta$:
  - *if validator i has heard > 2n/3 “up-to-date” msgs for (A,B) by this time:*
    - *package these messages into a quorum certificate (QC), Q*
    - send “ack (A,B,*Q*)” message to all validators
    - reset A_i := (A,B,*Q*)

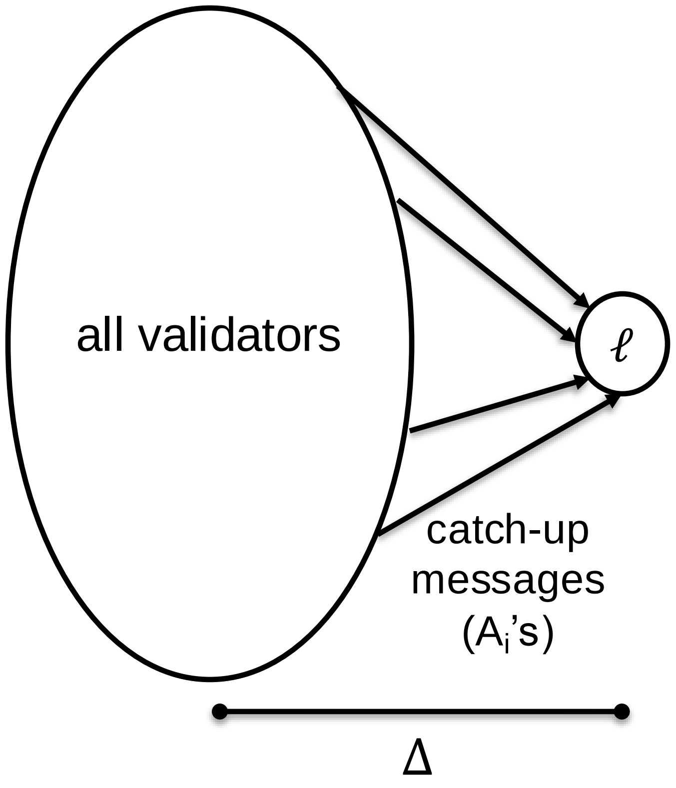
The Tendermint Protocol (con’d)

Protocol D (≈ Tendermint) [code run by every validator]

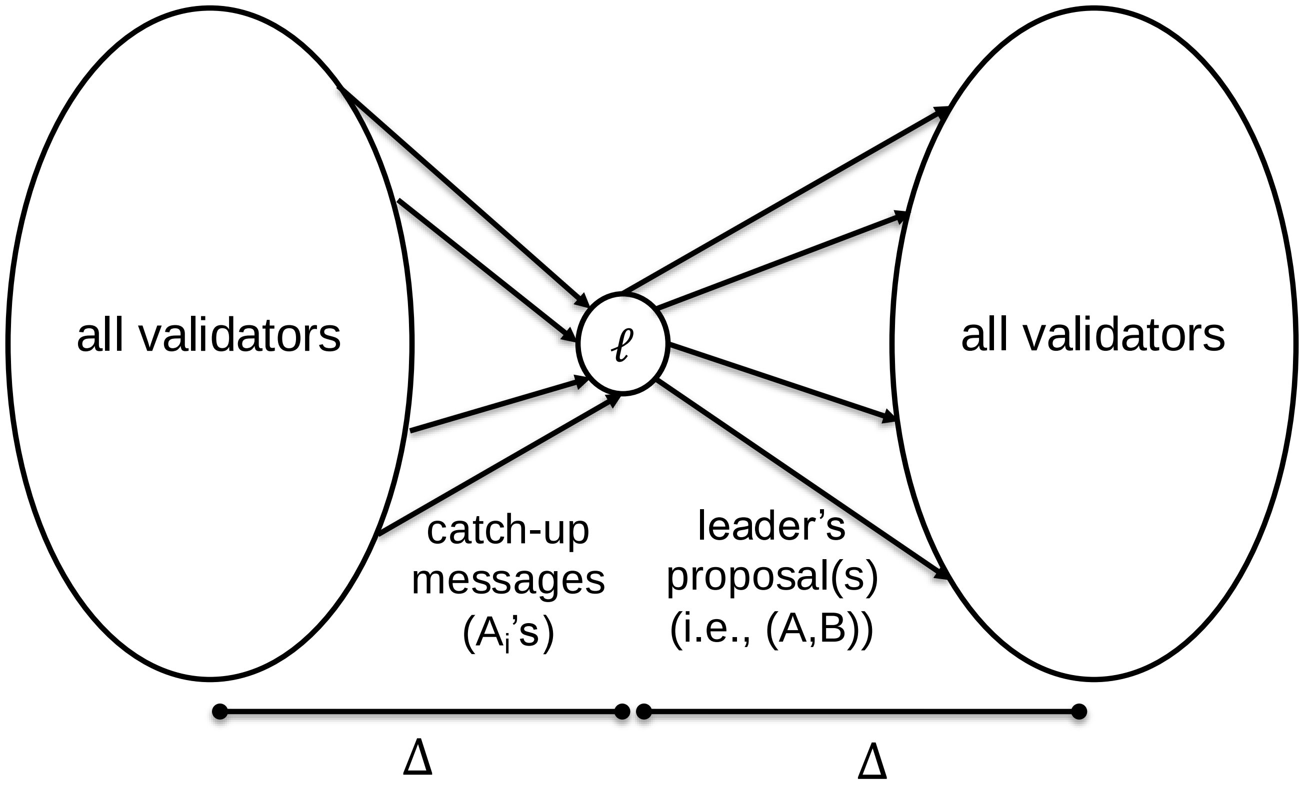
- at time $4\Delta \cdot v + 3\Delta$:
  - *if validator i has heard > 2n/3 “up-to-date” msgs for (A,B) by this time:*
    - *package these messages into a quorum certificate (QC), Q*
    - send “ack (A,B,*Q*)” message to all validators
    - reset A_i := (A,B,*Q*)
- at time $4\Delta \cdot v + 4\Delta$:
  - if validator i has received > *2n/3* “ack (A,B,*Q*)” messages:
    - reset C_i := (A,B,*Q*) (and also A_i := (A,B,*Q*), if necessary)

Tendermint: Picture of One View

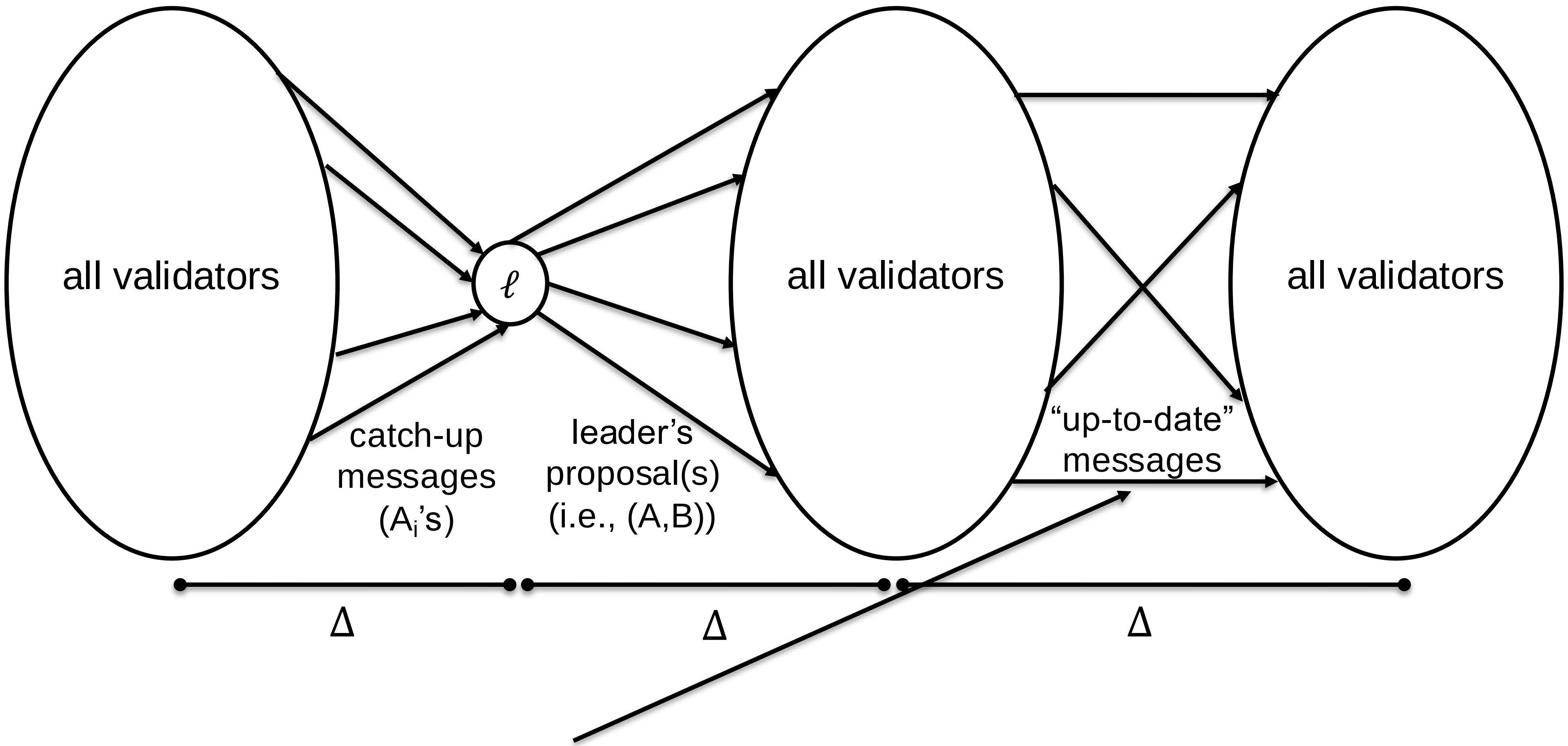
Tendermint: Picture of One View



Tendermint: Picture of One View

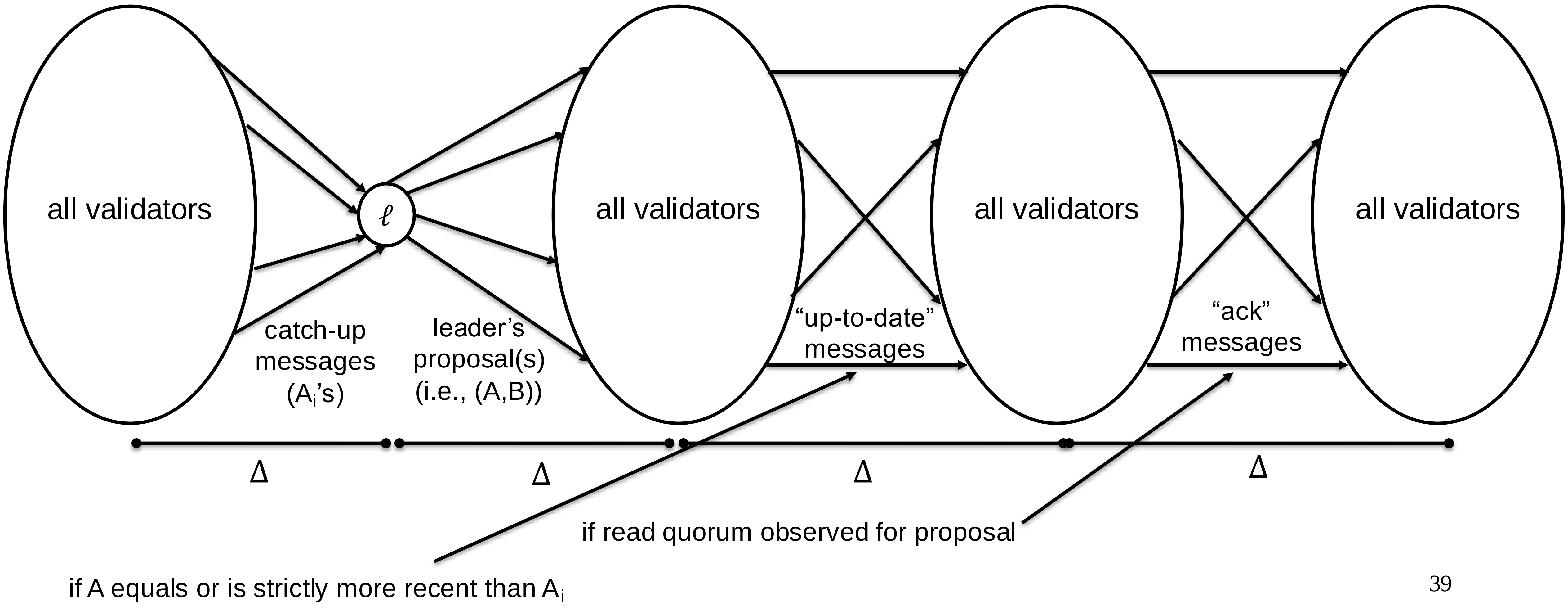


Tendermint: Picture of One View

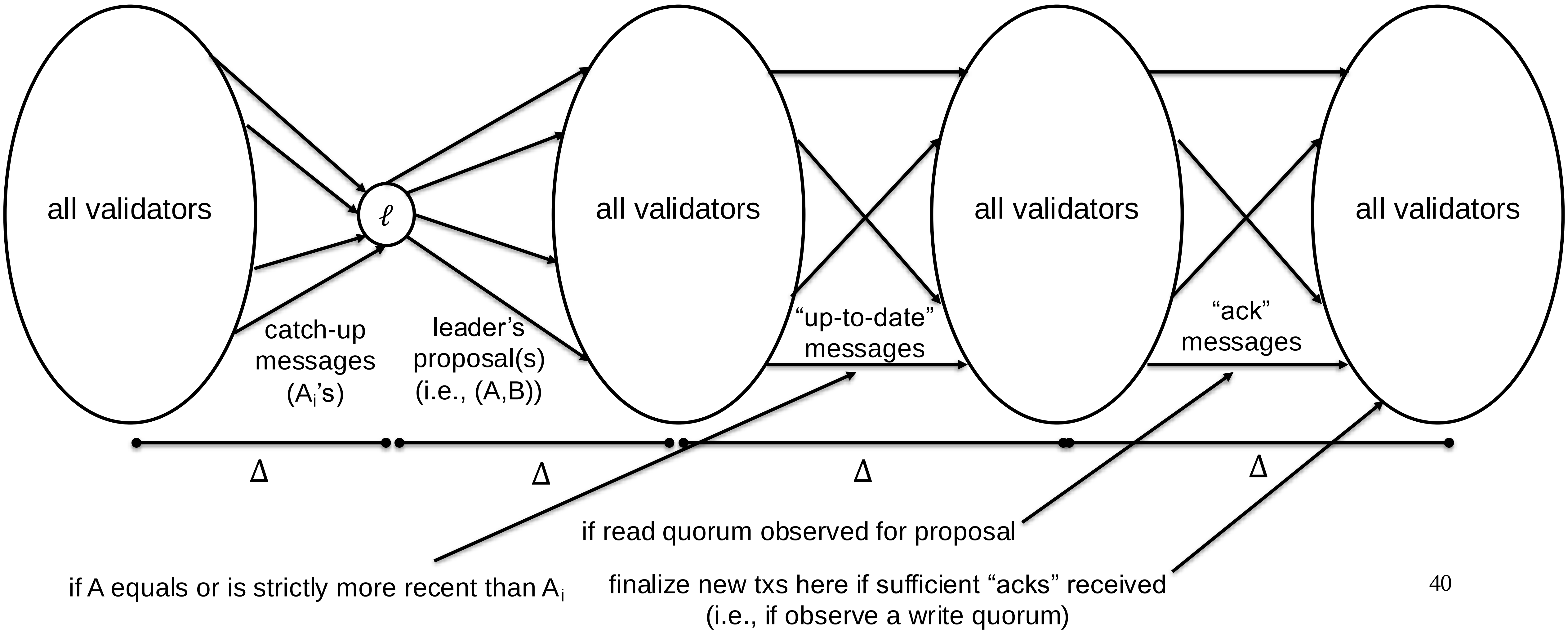


if A equals or is strictly more recent than A_i

Tendermint: Picture of One View



Tendermint: Picture of One View



Protocol D (≈ Tendermint)

- at time $4\Delta \cdot v$:
  - each validator i sends its current chain A_i to v's leader ℓ
- at time $4\Delta \cdot v + \Delta$:
  - let A = of the A_i's received, the most recently created one; let B := all not-yet-included (in A) valid txs ℓ knows about
  - ℓ sends proposal (A,B) to all other validators
- at time $4\Delta \cdot v + 2\Delta$:
  - if validator i receives a proposal (A,B) from ℓ with A = A_i or with A more recent than A_i by this time:
    - send "(A,B) is up-to-date" message to all validators
- at time $4\Delta \cdot v + 3\Delta$:
  - if validator i has heard > 2n/3 "up-to-date" msgs for (A,B) by this time (a *read quorum)*:
    - package these messages into a quorum certificate (QC), Q
    - send "ack (A,B,Q)" message to all validators and reset A_i := (A,B,Q)
- at time $4\Delta \cdot v + 4\Delta$:
  - if validator i has received > 2n/3 "ack (A,B,Q)" messages (a *write quorum)*:
    - reset C_i := (A,B,Q) (and also A_i := (A,B,Q), if necessary)

Recap: The Partially Synchronous 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 work no matter what GST is

Recall goals:

- consistency, always (even pre-GST/"under attack")
- liveness soon after GST (once "normal conditions" resume)
  - FLP ➔ need to give up one of consistency, liveness before GST

Tendermint: Proof of Consistency

Tendermint: Proof of Consistency

Key claim: for each view v:

1. All QCs formed in view v are for the same proposal (A,B).

Tendermint: Proof of Consistency

Key claim: for each view v:

1. All QCs formed in view v are for the same proposal (A,B).
   - consequence: if any non-faulty C_i's get updated in this view, all get updated to the same proposal A^* = (A,B,Q) made by v's leader.

Tendermint: Proof of Consistency

Key claim: for each view v:

1. All QCs formed in view v are for the same proposal (A,B).
   - consequence: if any non-faulty C_i's get updated in this view, all get updated to the same proposal A^* = (A,B,Q) made by v's leader.
2. in this case (i.e., ≥1 update in v), every QC created in a view v' > v is for a chain that extends A^*.

Tendermint: Proof of Consistency

Key claim: for each view v:

1. All QCs formed in view v are for the same proposal (A,B).
   - consequence: if any non-faulty C_i's get updated in this view, all get updated to the same proposal A^* = (A,B,Q) made by v's leader.
2. in this case (i.e., ≥1 update in v), every QC created in a view v' > v is for a chain that extends A^*.
   - consequence: all updates to non-faulty C_i's in views v' > v are to chains that extend A^*. [reason: never update without a QC]

Key Claim Implies Consistency

- consequence (1): if any non-faulty C_i's get updated in this view, all get updated to the same proposal $A^* = (A,B,Q)$.
- consequence (2): all updates to non-faulty C_i's in views v' > v are to chains that extend A^*.

Note: these consequences imply consistency:

Key Claim Implies Consistency

- consequence (1): if any non-faulty C_i's get updated in this view, all get updated to the same proposal $A^* = (A,B,Q)$.
- consequence (2): all updates to non-faulty C_i's in views v' > v are to chains that extend A^*.

Note: these consequences imply consistency:

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

Key Claim Implies Consistency

- consequence (1): if any non-faulty C_i's get updated in this view, all get updated to the same proposal $A^* = (A,B,Q)$.
- consequence (2): all updates to non-faulty C_i's in views v' > v are to chains that extend A^*.

Note: these consequences imply consistency:

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

Key Claim Implies Consistency

- consequence (1): if any non-faulty C_i's get updated in this view, all get updated to the same proposal $A^* = (A,B,Q)$.
- consequence (2): all updates to non-faulty C_i's in views $v' > v$ are to chains that extend A^*.

Note: these consequences imply 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

Proof of Claim (Part 1)

Claim: All QCs formed in view v are for the same proposal (A,B).

Proof of Claim (Part 1)

Claim: All QCs formed in view v are for the same proposal (A,B).

Proof: Let Q_1, Q_2 = two QCs formed in view v.

Proof of Claim (Part 1)

Claim: All QCs formed in view v are for the same proposal (A,B).

Proof: Let Q_1, Q_2 = two QCs formed in view v.

- let S = validators with "up-to-date" messages in Q_1
- let T = validators with "up-to-date" messages in Q_2

Proof of Claim (Part 1)

Claim: All QCs formed in view v are for the same proposal (A,B).

Proof: Let Q_1, Q_2 = two QCs formed in view v.

- let S = validators with "up-to-date" messages in Q_1
- let T = validators with "up-to-date" messages in Q_2
- $|S|, |T| > 2n/3$ [due to revised quorum thresholds]

Proof of Claim (Part 1)

Claim: All QCs formed in view v are for the same proposal (A,B).

Proof: Let Q_1, Q_2 = two QCs formed in view v.

- let S = validators with “up-to-date” messages in Q_1
- let T = validators with “up-to-date” messages in Q_2
- |S|, |T| > 2n/3 [due to revised quorum thresholds]
- > n/3 validators are in both S and T (<n/3 not in S, <n/3 not in T)

Proof of Claim (Part 1)

Claim: All QCs formed in view v are for the same proposal (A,B).

Proof: Let Q_1, Q_2 = two QCs formed in view v.

- let S = validators with “up-to-date” messages in Q_1
- let T = validators with “up-to-date” messages in Q_2
- $|S|, |T| > 2n/3$ [due to revised quorum thresholds]
- $> n/3$ validators are in both S and T ($<n/3$ not in S, $<n/3$ not in T)
- some non-faulty validator i is in both Q_1 and Q_2

Proof of Claim (Part 1)

Claim: All QCs formed in view v are for the same proposal (A,B).

Proof: Let Q_1, Q_2 = two QCs formed in view v.

- let S = validators with "up-to-date" messages in Q_1
- let T = validators with "up-to-date" messages in Q_2
- $|S|, |T| > 2n/3$ [due to revised quorum thresholds]
- $> n/3$ validators are in both S and T ($<n/3$ not in S, $<n/3$ not in T)
- some non-faulty validator i is in both Q_1 and Q_2
- since i sent an up-to-date message for only one leader proposal (A,B), Q_1 and Q_2 must both be for (A,B)

Proof of Claim (Part 2)

Need to show: if any non-faulty C_i is updated to A^* in view v → every QC created in a view v' > v is for a chain that extends A^*.

Proof of Claim (Part 2)

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ every QC created in a view v' > v is for a chain that extends A^*.

- i updated C_i to A^* in view v ➔ heard > 2n/3 "ack A^*" messages

Proof of Claim (Part 2)

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ every QC created in a view v' > v is for a chain that extends A^*.

- i updated C_i to A^* in view v ➔ heard > 2n/3 "ack A^*" messages, including > n/3 from a set U of non-faulty validators

Proof of Claim (Part 2)

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ every QC created in a view v' > v is for a chain that extends A^*.

- i updated C_i to A^* in view v ➔ heard > 2n/3 "ack A^*" messages, including > n/3 from a set U of non-faulty validators
- all j in U set A_j := A^* at time $4\Delta \cdot v + 3\Delta$

Proof of Claim (Part 2)

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ every QC created in a view v' > v is for a chain that extends A^*.

- i updated C_i to A^* in view v ➔ heard > 2n/3 "ack A^*" messages, including > n/3 from a set U of non-faulty validators
- all j in U set A_j := A^* at time $4\Delta \cdot v + 3\Delta$

In view v+1:

Proof of Claim (Part 2)

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ every QC created in a view v' > v is for a chain that extends A^*.

- i updated C_i to A^* in view v ➔ heard > 2n/3 "ack A^*" messages, including > n/3 from a set U of non-faulty validators
- all j in U set A_j := A^* at time $4\Delta \cdot v + 3\Delta$

In view v+1: a validator j in U will send an "up-to-date" message for a proposal (A,B) only if: (i) A := A^* OR (ii) A more recent than A^*

Proof of Claim (Part 2)

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ every QC created in a view v' > v is for a chain that extends A^*.

- i updated C_i to A^* in view v ➔ heard > 2n/3 "ack A^*" messages, including > n/3 from a set U of non-faulty validators
- all j in U set A_j := A^* at time $4\Delta \cdot v + 3\Delta$

In view v+1: a validator j in U will send an "up-to-date" message for a proposal (A,B) only if: (i) A := A^* OR ~~(ii) A more recent than A^*~~

  – (ii) is impossible (because A^* created in the most recent view, v)

Proof of Claim (Part 2)

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ every QC created in a view v' > v is for a chain that extends A^*.

- i updated C_i to A^* in view v ➔ heard > 2n/3 "ack A^*" messages, including > n/3 from a set U of non-faulty validators
- all j in U set $A_j := A^*$ at time $4\Delta \cdot v + 3\Delta$

In view v+1: a validator j in U will send an "up-to-date" message for a proposal (A,B) only if: (i) $A := A^*$ OR ~~(ii) A more recent than A^*~~

  – (ii) is impossible (because A^* created in the most recent view, v)

- if (A,B) doesn't extend A^* ➔ receives < 2n/3 "up-to-date" msgs

Proof of Claim (Part 2)

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ every QC created in a view v' > v is for a chain that extends A^*.

- i updated C_i to A^* in view v ➔ heard > 2n/3 "ack A^*" messages, including > n/3 from a set U of non-faulty validators
- all j in U set $A_j := A^*$ at time $4\Delta \cdot v + 3\Delta$

In view v+1: a validator j in U will send an "up-to-date" message for a proposal (A,B) only if: (i) A := A^* OR ~~(ii) A more recent than A^*~~

  – (ii) is impossible (because A^* created in the most recent view, v)

- if (A,B) doesn't extend A^* ➔ receives < 2n/3 "up-to-date" msgs
  – no QC for such a proposal can be formed in this view

Proof of Claim (Part 2)

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ every QC created in a view v' > v is for a chain that extends A^*.

- i updated C_i to A^* in view v ➔ heard > 2n/3 "ack A^*" messages, including > n/3 from a set U of non-faulty validators
- all j in U set $A_j := A^*$ at time $4\Delta \cdot v + 3\Delta$

In view v+2: for each j in U, A_j is either A^* or a chain + QC created in view v+1 (which, as we just saw, must extend A^*).

Proof of Claim (Part 2)

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ every QC created in a view v' > v is for a chain that extends A^*.

- i updated C_i to A^* in view v ➔ heard > 2n/3 "ack A^*" messages, including > n/3 from a set U of non-faulty validators
- all j in U set A_j := A^* at time $4\Delta \cdot v + 3\Delta$

In view v+2: for each j in U, A_j is either A^* or a chain + QC created in view v+1 (which, as we just saw, must extend A^*).

- if proposal (A,B) doesn't extend A^* ➔ receives < 2n/3 "up-to-date" msgs [none from the > n/3 validators of U]

Proof of Claim (Part 2)

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ every QC created in a view v' > v is for a chain that extends A^*.

- i updated C_i to A^* in view v ➔ heard > 2n/3 "ack A^*" messages, including > n/3 from a set U of non-faulty validators
- all j in U set A_j := A^* at time $4\Delta \cdot v + 3\Delta$

In view v+2: for each j in U, A_j is either A^* or a chain + QC created in view v+1 (which, as we just saw, must extend A^*).

- if proposal (A,B) doesn't extend A^* ➔ receives < 2n/3 "up-to-date" msgs [none from the > n/3 validators of U]
  - no QC for such a proposal can be formed in this view

Proof of Claim (Part 2)

Need to show: if any non-faulty C_i is updated to A^* in view v ➔ every QC created in a view v' > v is for a chain that extends A^*.

- i updated C_i to A^* in view v ➔ heard > 2n/3 "ack A^*" messages, including > n/3 from a set U of non-faulty validators
- all j in U set A_j := A^* at time $4\Delta \cdot v + 3\Delta$

In general (by induction on v' > v): for each j in U, A_j is either A^* or a chain+QC created in a view > v (which, inductively, extends A^*).

- if proposal (A,B) doesn't extend A^* ➔ receives < 2n/3 "up-to-date" msgs [none from the > n/3 validators of U]
  - no QC for such a proposal can be formed in this view

Tendermint: Proof of Liveness

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

Tendermint: 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?)

Tendermint: 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 $4\Delta \cdot v + \Delta$, i will receive A_j's from all non-faulty validators (+ possibly some Byzantine validators)

Tendermint: 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 $4\Delta \cdot v + \Delta$, i will receive A_j's from all non-faulty validators (+ possibly some Byzantine validators)
- let A = most recent of these (i.e., QC from the largest view)

Tendermint: 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 $4\Delta \cdot v + \Delta$, i will receive A_j's from all non-faulty validators (+ possibly some Byzantine validators)
- let A = most recent of these (i.e., QC from the largest view)
  - note: because all QCs from the same view are for the same proposal (by part 1 of the consistency claim), A is unique (i.e., no ties possible)

Tendermint: 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 $4\Delta \cdot v + \Delta$, i will receive A_j's from all non-faulty validators (+ possibly some Byzantine validators)
- let A = most recent of these (i.e., QC from the largest view)
  - note: because all QCs from the same view are for the same proposal (by part 1 of the consistency claim), A is unique (i.e., no ties possible)
  - i makes a proposal (A,B) that includes the tx z (if not in A, then in B)

Tendermint: Proof of Liveness

- post-GST ➔ by time $4\Delta \cdot v + \Delta$, i will receive A_j's from all non-faulty validators (+ possibly some Byzantine validators)
- let A = most recent of these (i.e., QC from the largest view)
  – note: because all QCs from the same view are for the same proposal (by part 1 of the consistency claim), A is unique (i.e., no ties possible)
  – i makes a proposal (A,B) that includes the tx z (if not in A, then in B)

Tendermint: Proof of Liveness

- post-GST ➔ by time $4\Delta \cdot v + \Delta$, i will receive A_j's from all non-faulty validators (+ possibly some Byzantine validators)
- let A = most recent of these (i.e., QC from the largest view)
  - note: because all QCs from the same view are for the same proposal (by part 1 of the consistency claim), A is unique (i.e., no ties possible)
  - i makes a proposal (A,B) that includes the tx z (if not in A, then in B)
- post-GST ➔ all non-faulty validators get (A,B) by $4\Delta \cdot v + 2\Delta$
  - by choice of A, all send "(A,B) up-to-date" messages at that time

Tendermint: Proof of Liveness

- post-GST ➔ by time $4\Delta \cdot v + \Delta$, i will receive A_j's from all non-faulty validators (+ possibly some Byzantine validators)
- let A = most recent of these (i.e., QC from the largest view)
  - i makes a proposal (A,B) that includes the tx z (if not in A, then in B)
- post-GST ➔ all non-faulty validators get (A,B) by $4\Delta \cdot v + 2\Delta$
  - by choice of A, all send "(A,B) up-to-date" messages at that time

Tendermint: Proof of Liveness

- post-GST ➔ by time $4\Delta \cdot v + \Delta$, i will receive A_j's from all non-faulty validators (+ possibly some Byzantine validators)
- let A = most recent of these (i.e., QC from the largest view)
  - i makes a proposal (A,B) that includes the tx z (if not in A, then in B)
- post-GST ➔ all non-faulty validators get (A,B) by $4\Delta \cdot v + 2\Delta$
  - by choice of A, all send "(A,B) up-to-date" messages at that time
- post-GST ➔ all non-faulty validators get > 2n/3 "(A,B) up-to-date" messages by time $4\Delta \cdot v + 3\Delta$

Tendermint: Proof of Liveness

- post-GST ➔ by time $4\Delta \cdot v + \Delta$, i will receive A_j's from all non-faulty validators (+ possibly some Byzantine validators)
- let A = most recent of these (i.e., QC from the largest view)
  - i makes a proposal (A,B) that includes the tx z (if not in A, then in B)
- post-GST ➔ all non-faulty validators get (A,B) by $4\Delta \cdot v + 2\Delta$
  - by choice of A, all send “(A,B) up-to-date” messages at that time
- post-GST ➔ all non-faulty validators get > 2n/3 “(A,B) up-to-date” messages by time $4\Delta \cdot v + 3\Delta$
  - all send “ack (A,B,Q)” messages at that time

Tendermint: Proof of Liveness

- post-GST ➔ by time $4\Delta \cdot v + \Delta$, i will receive A_j's from all non-faulty validators (+ possibly some Byzantine validators)
- let A = most recent of these (i.e., QC from the largest view)
  - i makes a proposal (A,B) that includes the tx z (if not in A, then in B)
- post-GST ➔ all non-faulty validators get (A,B) by $4\Delta \cdot v + 2\Delta$
  - by choice of A, all send "(A,B) up-to-date" messages at that time
- post-GST ➔ all non-faulty validators get > 2n/3 "(A,B) up-to-date" messages by time $4\Delta \cdot v + 3\Delta$
  - all send "ack (A,B,Q)" messages at that time

Tendermint: Proof of Liveness

- post-GST ➔ by time $4\Delta \cdot v + \Delta$, i will receive A_j's from all non-faulty validators (+ possibly some Byzantine validators)
- let A = most recent of these (i.e., QC from the largest view)
  – i makes a proposal (A,B) that includes the tx z (if not in A, then in B)
- post-GST ➔ all non-faulty validators get (A,B) by $4\Delta \cdot v + 2\Delta$
  – by choice of A, all send "(A,B) up-to-date" messages at that time
- post-GST ➔ all non-faulty validators get > 2n/3 "(A,B) up-to-date" messages by time $4\Delta \cdot v + 3\Delta$
  – all send "ack (A,B,Q)" messages at that time
- post-GST ➔ all non-faulty validators j get > 2n/3 "ack (A,B,Q)" messages by time $4\Delta \cdot v + 4\Delta$, set C_j := (A,B,Q) [thereby finalizing tx z]