Choosing a Value: Pseudocode

Phase 1:

a. Proposer c:
   i. Selects a unique proposal number $n > crnd_c$, sets $cval_c$ to *none* and $crnd_c$ to n.
   ii. Sends a *prepare*(n) to all acceptors.

b. Acceptor a receives *prepare*(n) from c:
   i. If $n > rnd_a$ then set rnd_a to n and send *promise*(rnd_a, $vrnd_a$, $vval_a$) to c.
   ii. Else ignore request.

Phase 2:

a. Proposer c receives *promise*(rnd_a, $vrnd_a$, $vval_a$) from a majority of acceptors with $rnd_a = crnd_c$:
   i. If all reply with $vrnd_a = 0$, then set $cval_c$ to any proposed value
      Else set $cval_c$ to $vval_a$ associated with largest received value of $vrnd_a$.
   ii. Send *accept*($crnd_c$, $cval_c$) to all acceptors.

b. Acceptor a receives *accept*(n, v):
   i. If $n \geq rnd_a$ and $vrnd_a \neq n$ then set $vrnd_a$ and rnd_a to n and $vval_a$ to v, and send *learn*(n, v).
   ii. Else ignore request.

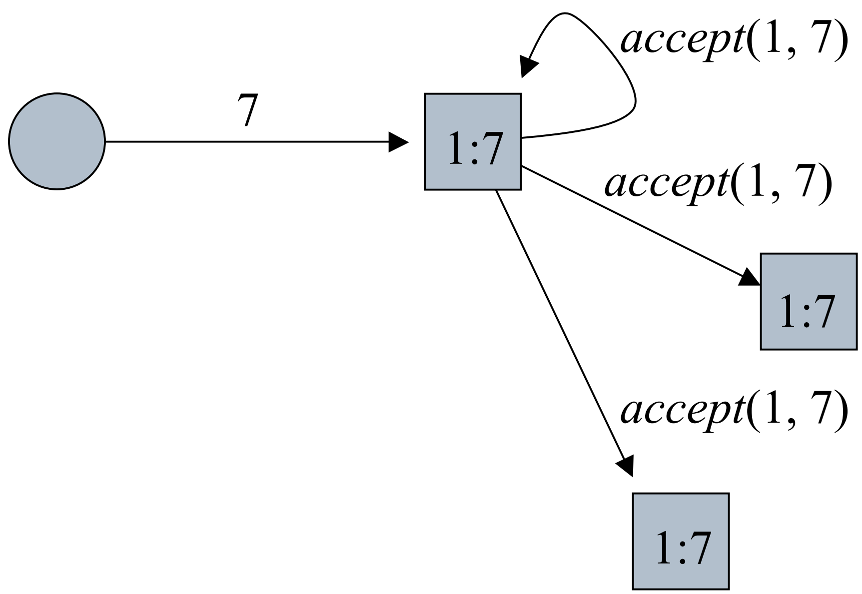
Making Paxos Faster

- The normal-case Paxos communication pattern is proposer → leader → acceptors → learners
  - In the common case for Paxos, the leader is unconstrained in the value it chooses for *accept*(n, v).
  - So, why not have the leader say *take any value* and have acceptors get values directly from a proposer?

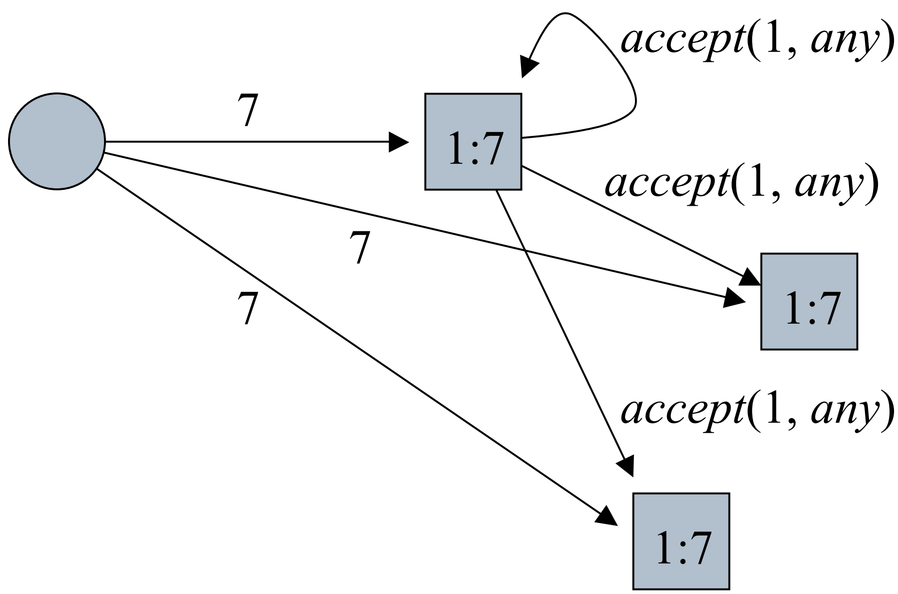
Fast Paxos

- Will call proposal numbers *rounds*.
- Create two kinds of rounds: *fast* and *classic*.
  - In a fast round, if the coordinator can pick any proposed value for 2a, it can instead send *propose*(i, ***any***).
  - When an acceptor receives *propose*(i, ***any***) it can treat any proposer's message proposing a value as if it were an ordinary round i phase 2a message with that value.
    - It can, however, execute a round i phase 2b action only once, for a single value.

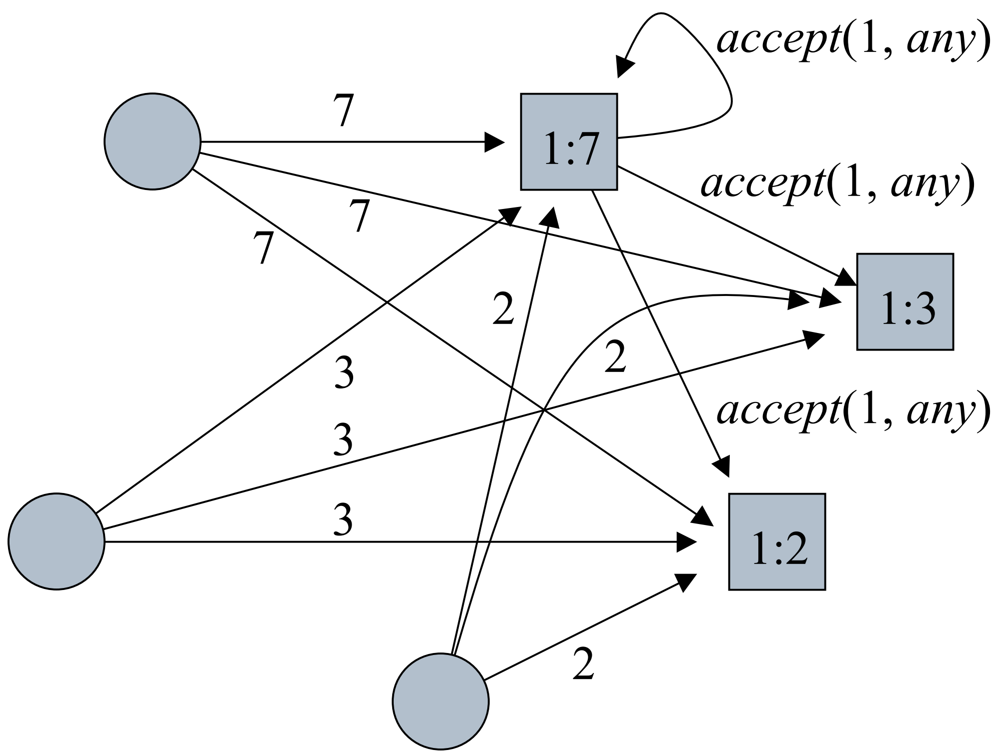
Classic Paxos



Fast Paxos



Fast Paxos



Problems with Fast Paxos

- How does a coordinator pick a value for 2a?
  - An issue if not in round 1...
- What happens if no value is chosen by 2b?
  - How can we tell this happened?

Picking value for 2a, round i?

- **let** Q be a quorum of acceptors that have sent phase 1b messages *promise*(vr_a, vv_a).
- **let** k be the largest vr_a for all $a \in Q$.
- **let** V be the set of values vv_a for all $a \in Q$ with $vr_a = k$.
- **if** $k = 0$ **then** pick any proposed value **else** pick the (only) element of V.

… no longer a sound rule because V can have more than one element…

Picking value for 2a round i

- The coordinator needs to pick a value v: for any round $j < i$, no value other than v has been or might yet be chosen in round j.
  - Recall k be the largest vr_a for all $a \in Q$.
    - if $k = 0$:
      - each $a \in Q$ sent $vr_a = 0$: none had voted for any $j < i$.
      - all $a \in Q$: $rnd_a \geq i$, so no value has been or ever might be chosen for any round $j < i$.
      - thus, coordinator is unconstrained
    - if $k > 0$:
      - three cases: $k < j$, $j = k$, and $j < k$.

Picking value for 2a round i

- if $k < j$: for each $a \in Q$
  - by time sent $promise(vr_a, vv_a)$ had not voted on j.
  - since it promised, a won't vote on j after sending.
- if $k = j$:
  - acceptor only votes for value sent it by coordinator of round k, so all $a \in Q$ either voted for vv_a or didn't vote.
- if $k > j$:
  - by induction, held through round k. So, no value other than v has been or might yet be chosen in round j.

Picking value for 2a round *i*

- if $k < j$: for each $a \in Q$
  - by time sent *promise*(vr_a, vv_a) had not voted on j.
  - since it promised, a didn't vote on j after sending.
- if $k = j$:
  - acceptor only votes for value sent it by coordinator of round k, so all $a \in Q$ either voted for vv_a or didn't vote.
- if $k > j$:
  - by inducti d k. So, no value other than v has chosen in round j.

For a fast round, acceptor can vote for any proposed value!

Picking value for 2a round *i*

- Will be useful to allow different (sized) quorums for different rounds.
  - Quorum for round *i* is an *i-quorum*.
  - For any (not necessarily different) rounds *i* and *j*, any *i*-quorum and *j*-quorum intersect.

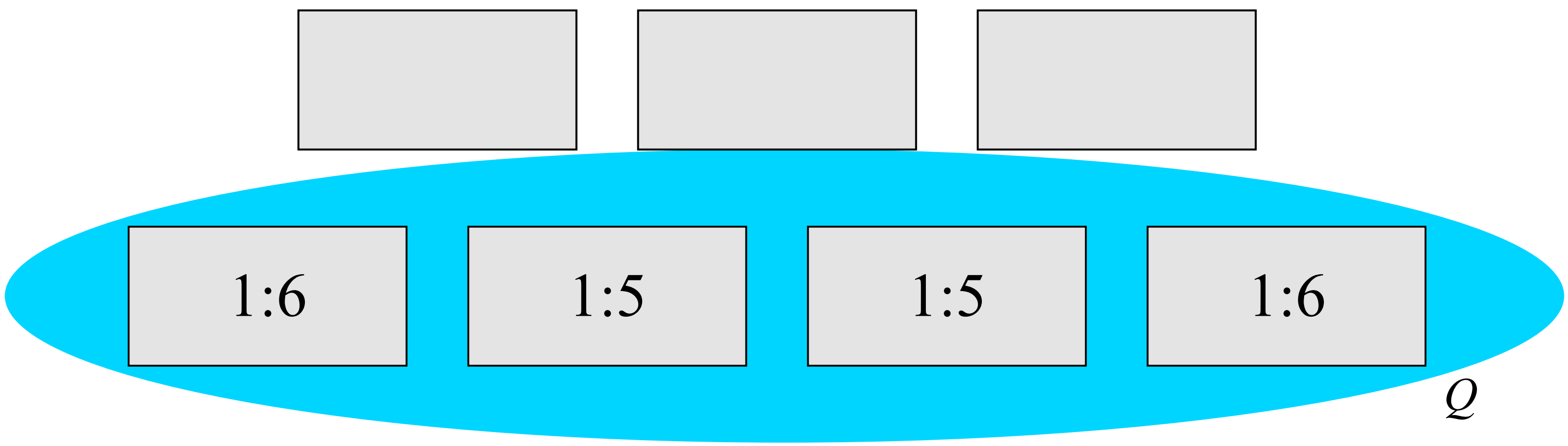
Picking value for 2a round i

- A value v might have been or might yet be chosen in round k only if there is a k-quorum R: for each acceptor a in R, a has $rnd_a \leq k$ or has voted for v in round k.
- Every $a \in Q$ has $rnd_a \geq i > k$ since it sent its promise.
- So, v might have been or might yet be chosen in round k only if there is a k-quorum R: for all $a \in Q \cap R$: $vr_a = k$ and $vv_a = v$.

$A(v)$: There is a k-quorum R:
for all $a \in Q \cap R$: $vr_a = k$ and $vv_a = v$

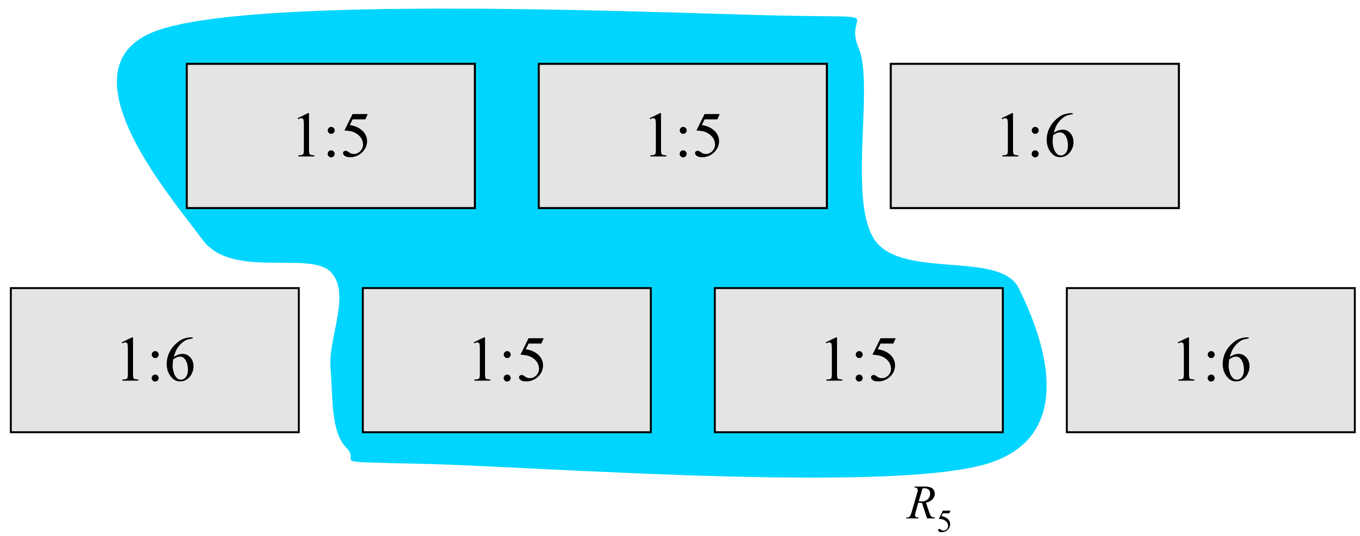
Condition $A(v)$

$N = 7,\ t = 3$, quorum size = 4



Condition $A(v)$

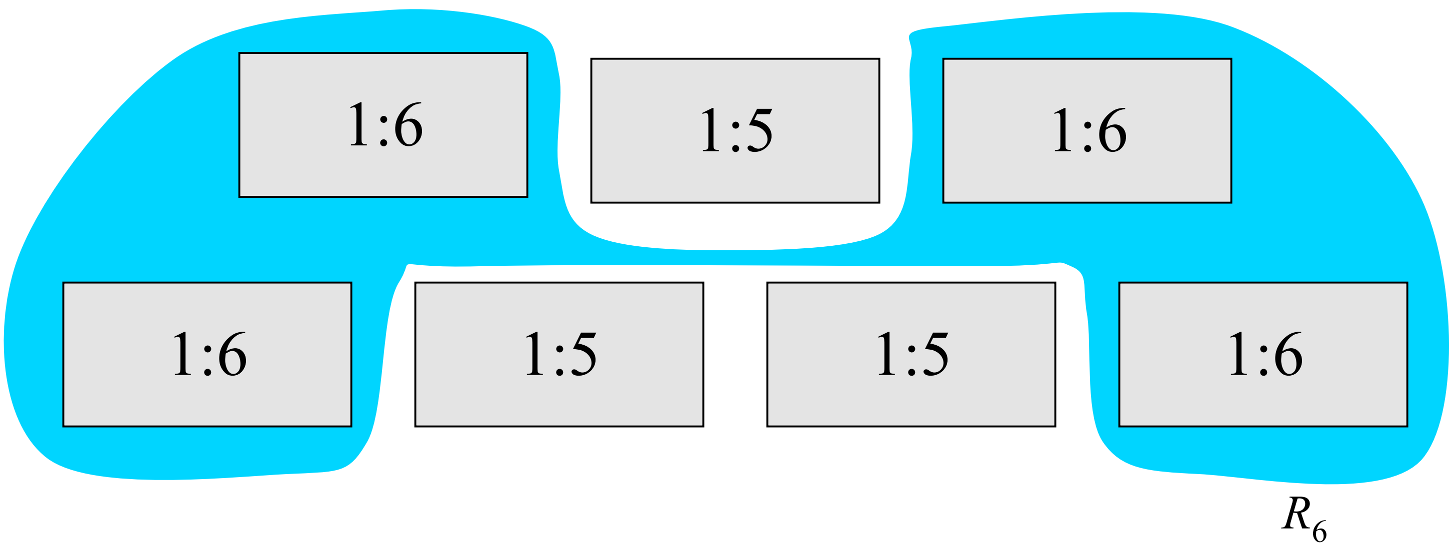
$N = 7,\ t = 3$, quorum size = 4 $\qquad A(5)$



Condition $A(v)$

$N = 7,\ t = 3$, quorum size = 4

$A(5)$ and $A(6)$

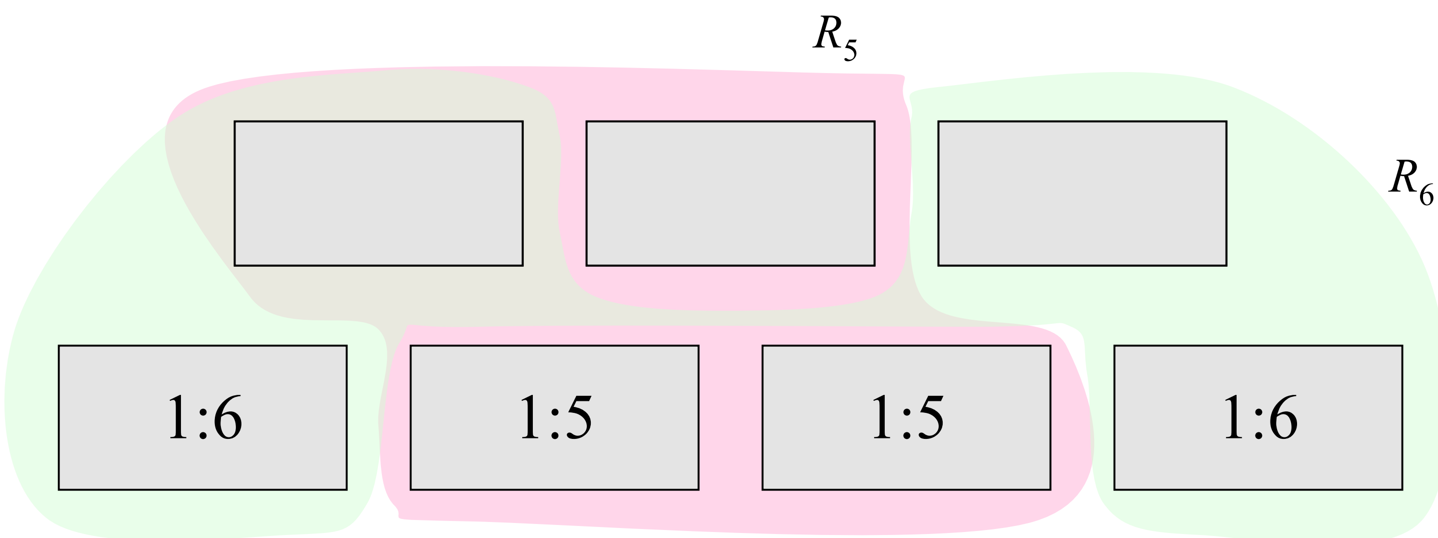


Picking value for 2a round i

- $A(v)$ and $A(w)$ for $v \neq w$:
  - There are two k-quorums R_v and R_w:
    - $\forall\ a \in R_v \cap Q$: $vv_a = v$.
    - $\forall\ a \in R_w \cap Q$: $vv_a = w$.
    - Note that $R_v \cap R_w$ $R_w \cap Q$, and $R_v \cap R_w$ are all non-empty, but $R_v \cap R_w \cap Q$ must be empty for this to hold.

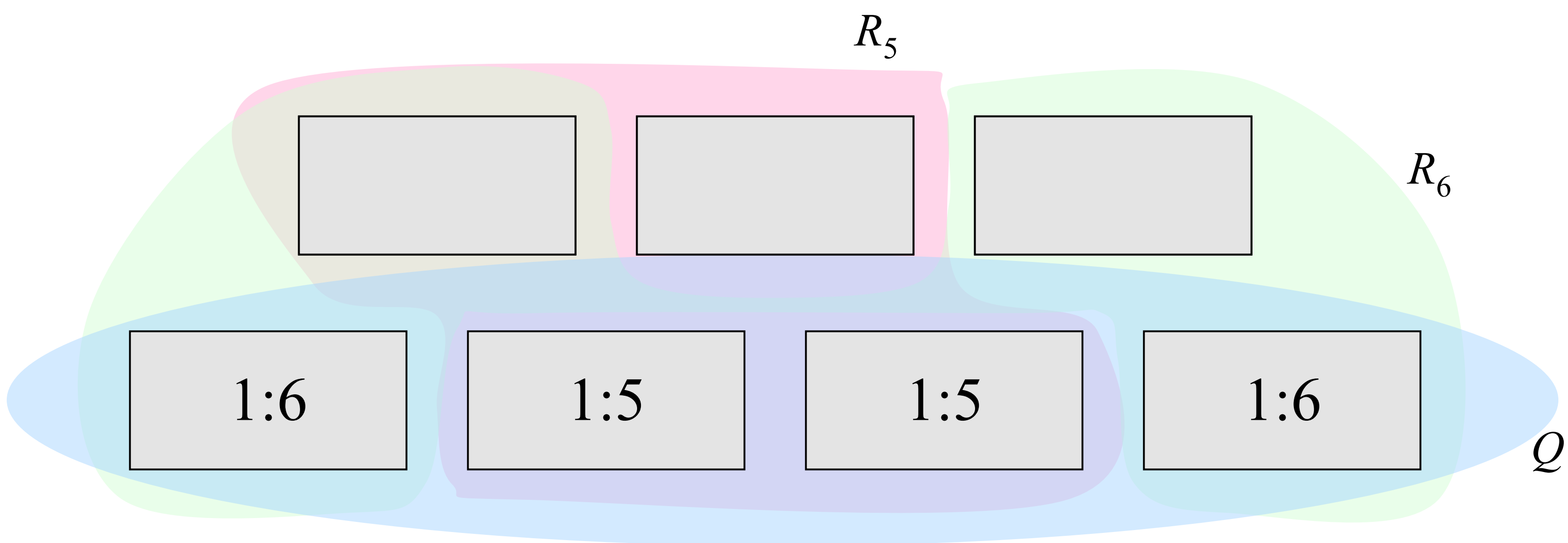
Condition $A(v)$

$N = 7,\ t = 3$, majority = 4



Condition $A(v)$

$N = 7,\ t = 3$, majority = 4



Picking value for 2a round *i*

… so, let's ensure $R_v \cap R_w \cap Q$ is not empty!

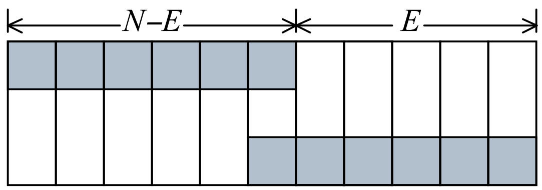
For any rounds *i* and *j*:

- Any *i*-quorum and *j*-quorum have a non-empty intersection.
- If *j* is a fast round, then any *i*-quorum and two *j*-quorums have a non-empty intersection.

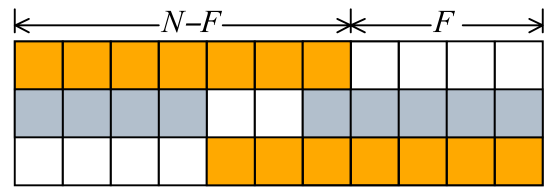
Classic Quorums and Fast Quorums

- If there are N acceptors, choose E and F:
  - $N - E$ acceptors are a classic quorum
  - $N - F$ acceptors are a fast quorum
  - Since fast quorums have more stringent requirements, they should be at least as large as classic quorums: $E \geq F$.

Classic Quorums and Fast Quorums



$2(N\text{–}E) > N$, or $N > 2E$



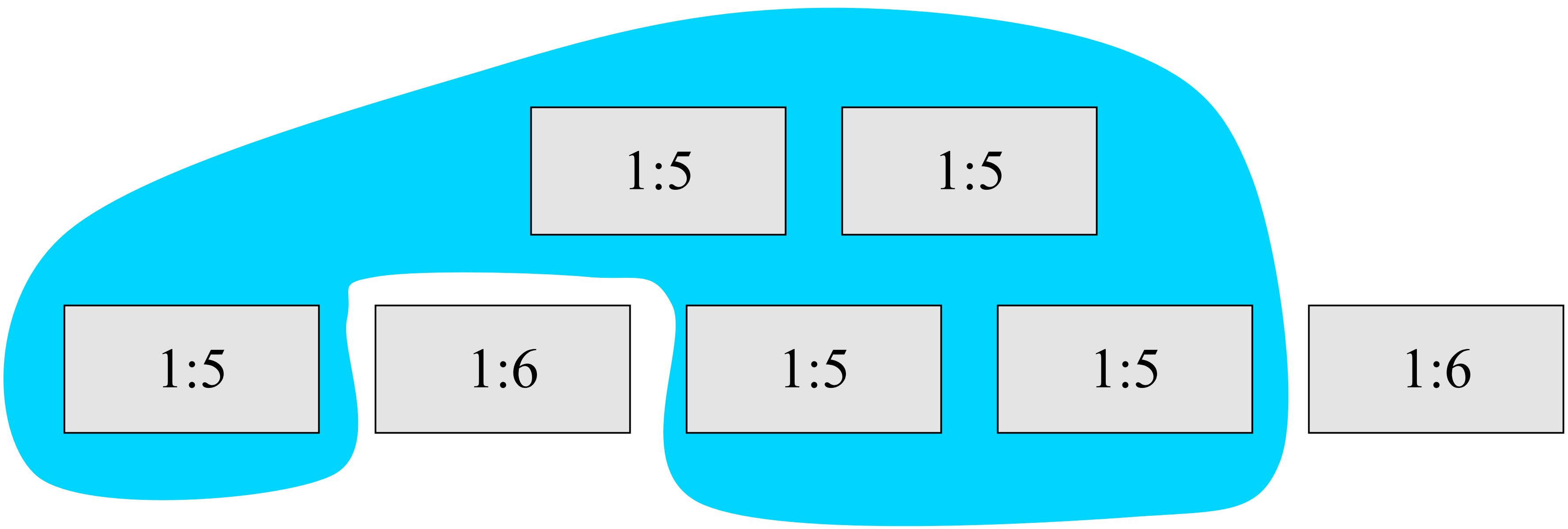
$N\text{–}E > 2F$, or $N > 2F + E$

- Maximize F: $E = F = \lceil N/3 \rceil$
- Maximize E: $E = \lceil N/2 \rceil - 1$, $E = \lfloor N/4 \rfloor$
  - eg, if $t = 2$, can have $N = 7$, $E = F = 2$.
  - or, can have $N = 5$, $E = 2$, $F = 1$ and run only classic runs if appears that there are two failures.

Condition $A(v)$

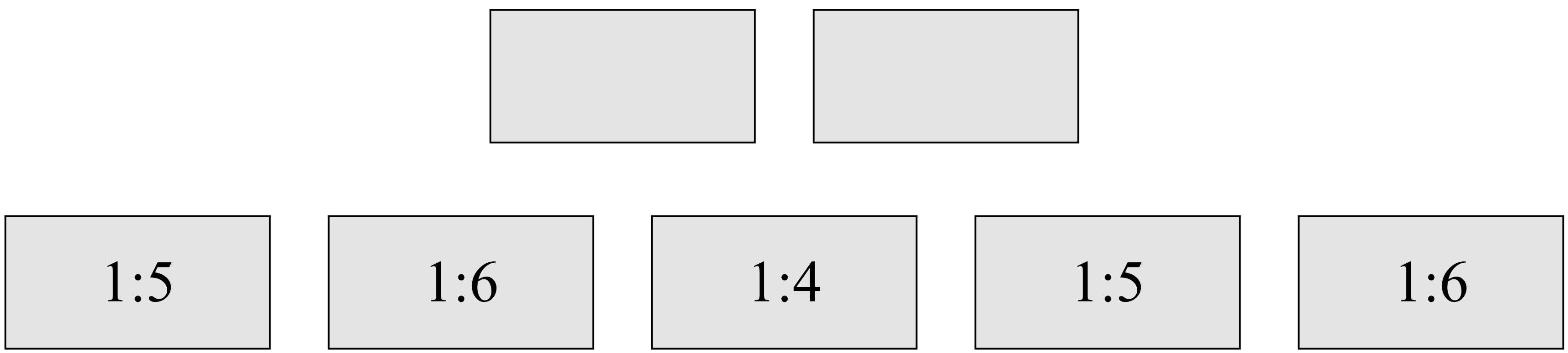
$N = 7,\ E = F = 2$

$A(5)$ and $\neg A(6)$



Condition $A(v)$

$N = 7,\ E = F = 2$ $\qquad$ $\neg A(4),\ \neg A(5),\ \neg A(6)$



Picking value for 2a round i

- **let** Q be an i-quorum of acceptors that have sent round i phase 1b messages $promise(vr_a, vv_a)$.
- **let** k be the largest vr_a for all $a \in Q$.
- **let** V be the set of values vv_a for all $a \in Q$ with $vr_a = k$.
- **if** $k = 0$ **then** pick any proposed value or pick *any*
  else if V contains only one element w then pick w
  else if $w \in V$ that satisfies $A(w)$ then pick w
  else pick any value in V

Collisions

- Two or more proposers send proposals at about the same time, and they are received in different orders by different acceptors.
  - This may result in no value being chosen.
  - A collision in round *i* will be noticed by learners if they do not receive an *i*-quorum of identical values.
  - Coordinators can notice if they are also learner.
  - Acceptors can notice if they are learners, too.

Observation

- Suppose c receives round i phase 2b *learn*(i, v) from a followed by round i+1 phase 1b *promise*(r, v) from a.
  - Both report a's vote for round i and promise that a will not vote on any round less than i+1.
- So, if coordinator c has received round i *learn*(i, v) and is starting round i +1, it doesn't need a's round i+1 *promise* message to send *accept*(i +1, $cval_c$).

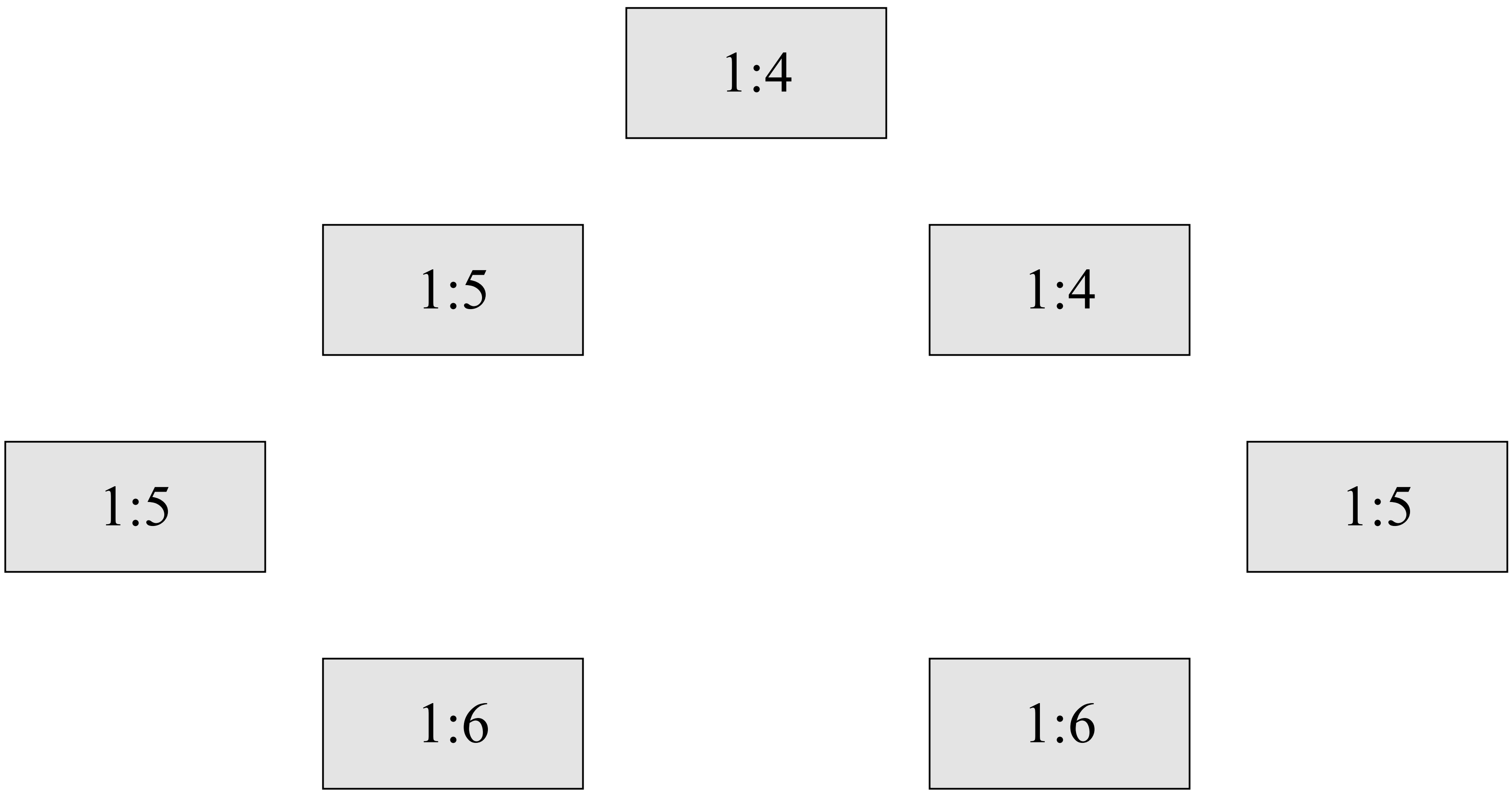
Coordinated Recovery

- *i* is a fast round and *c* coordinates both rounds *i* and *i* + 1.
  - Once *c* receives round *i* *learn* messages from an (*i* + 1)-quorum it starts phase 2a of round *i* + 1.
  - Since *V* is non-empty, *c* does not send *accept*(*i* + 1, *any*).
  - This will succeed if the acceptors in a nonfaulty (*i* + 1)-quorum receive these messages before receiving any message from a higher round.

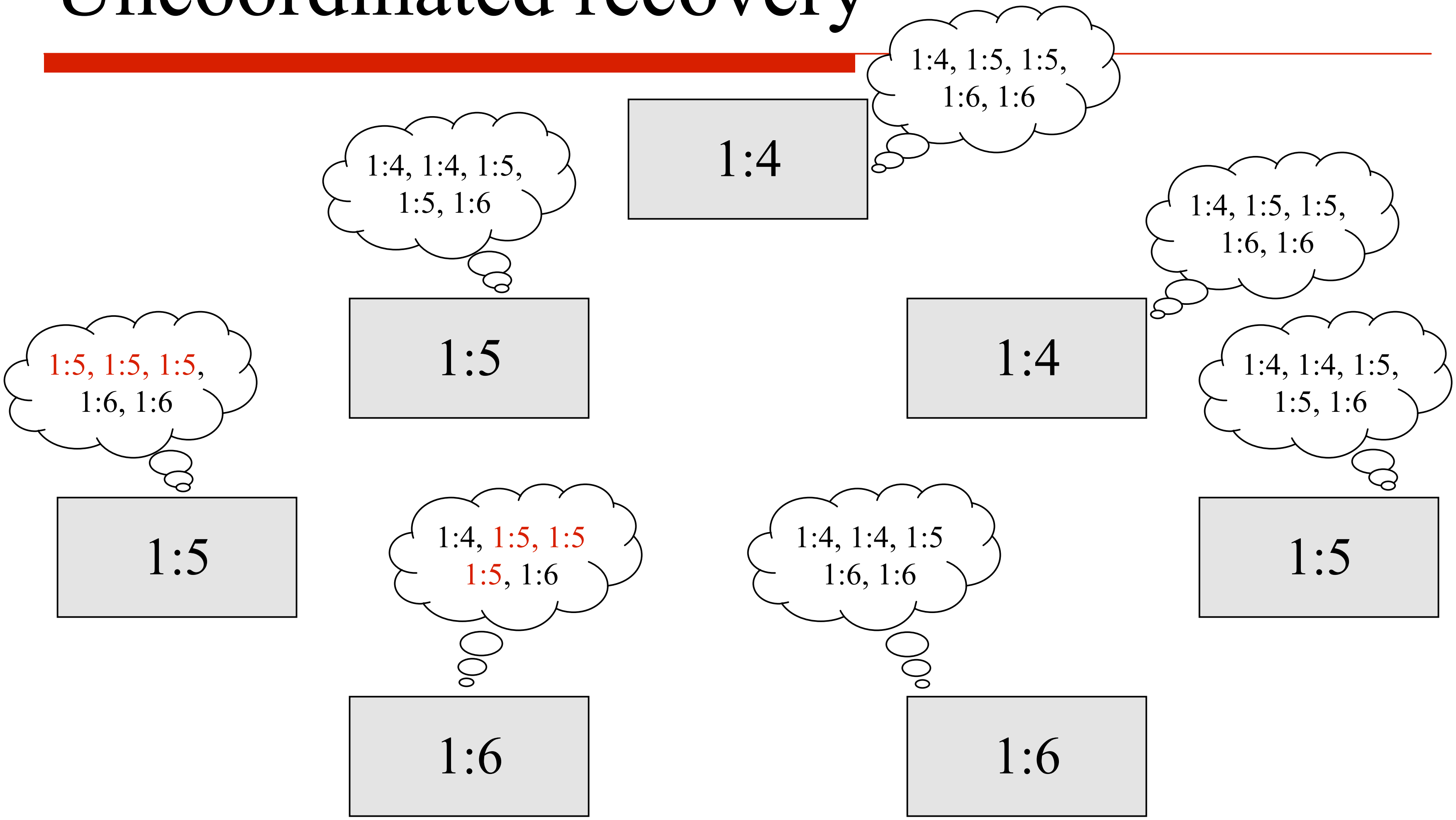
Uncoordinated Recovery

- Both i and $i + 1$ are fast rounds.
  - acceptors send *learn* messages to all other acceptors.
  - Each acceptor uses the same procedure as in coordinated recovery to pick a value v the the coordinator could send in a round ($i + 1$) *accept* message.

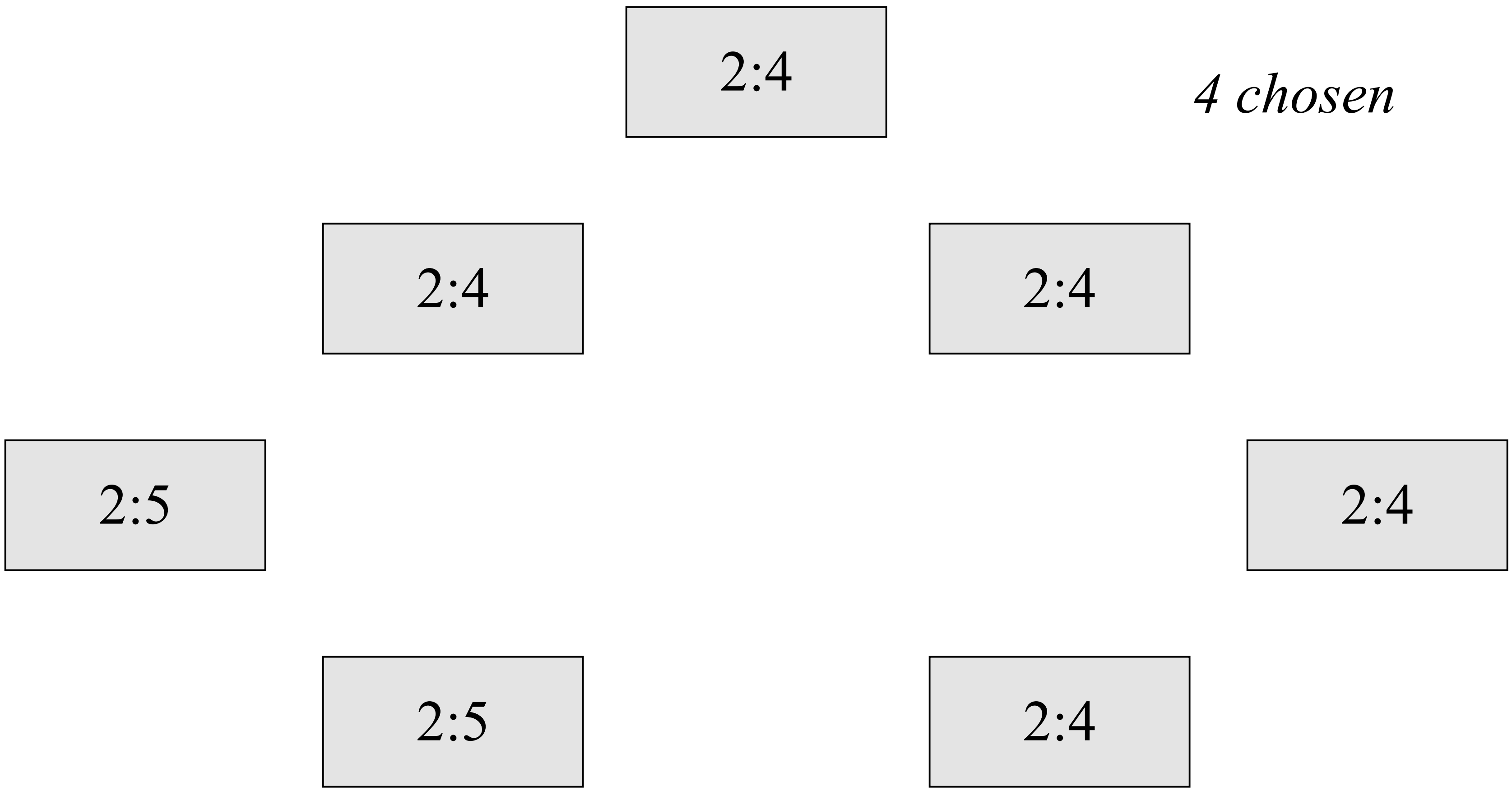
Uncoordinated recovery



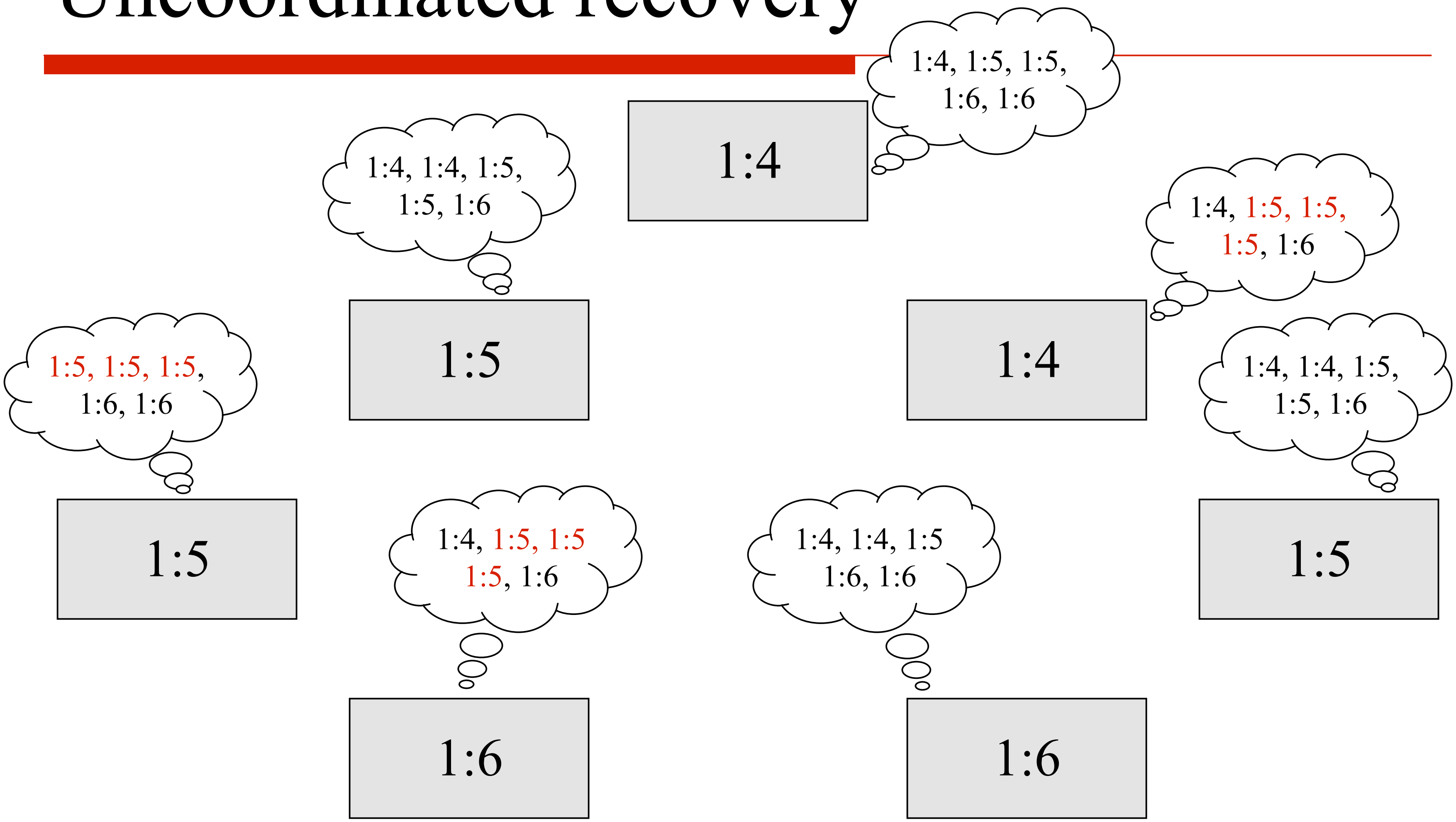
Uncoordinated recovery



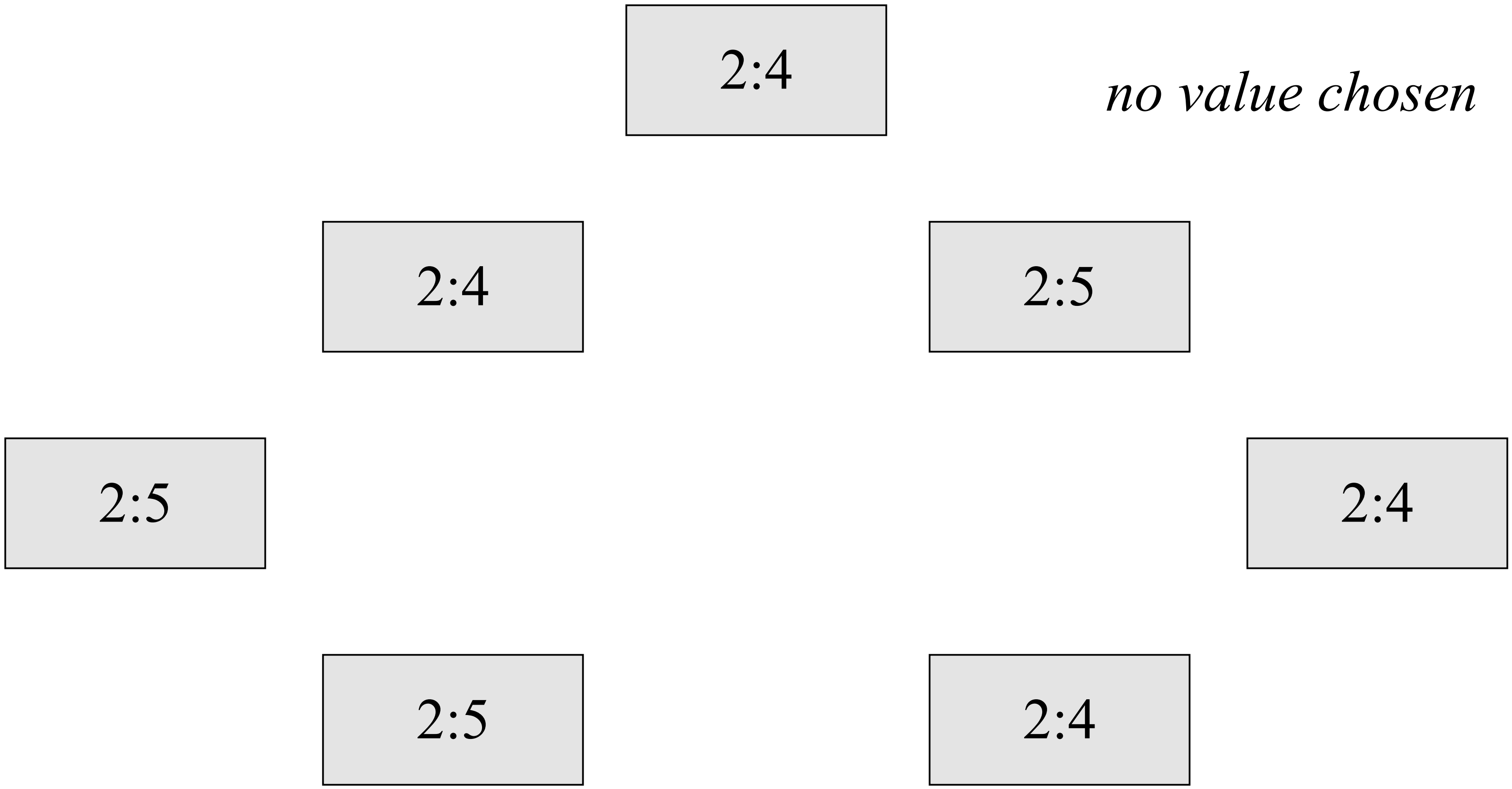
Uncoordinated recovery



Uncoordinated recovery



Uncoordinated recovery



no value chosen

Default recovery

- Any proposer is free to start some round greater than $i + 1$ with a *propose* message.
  - This will run phase 1, and a classic phase 2 will reach consensus (barring conflicting proposers).

… is it worth it?

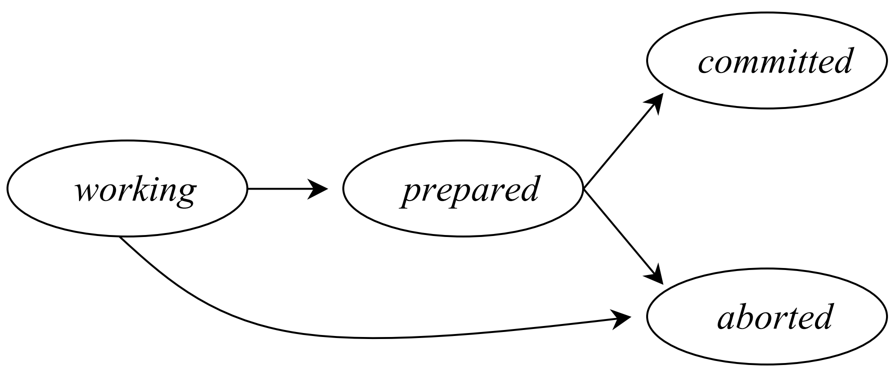
- Common case is faster.
- Requires more bookkeeping or larger quorums.
- Recovering from collisions takes time.
  - Local area networks and using network-level multicast could make collisions unlikely.

Paxos and Atomic Commit

- Recall *atomic commit* protocols.
  - 2 phase commit is blocking in the face of one (or two) well-placed failures.
  - 3 phase commit does not have this weakness, but it requires leader election (which is a perfect failure detector).
  - Consensus only requires $\diamond$W…

Atomic Commit

- Resource managers (RM) agree on *commit* or *abort*.
- Decision directed by a transaction manager (TM), which can be a resource manager as well.
  - *Stability*: Once an RM has entered *committed* or *aborted* state, it remains in that state.
  - *Consistency*: It is impossible for one RM to be in the *committed* state and another to be in the *aborted* state.



Paxos Commit

- In 2PC, one TM collects the abort/commit votes from the RMs, decides the outcome, and disseminates the result to the RMs.
- In 3PC, if the TM fails, then its failure is detected and a new TM is elected.
  - Created a *precommitted* state to ensure there was no state in which it was possible for an RM to be *committed* and in which it was possible for an RM to be *aborted*.
- We could have TM use consensus once it knows all RMs are in *prepared* state.
  - … but there is a more message efficient approach: we have $2f + 1$ acceptors in the role of transaction managers.

Paxos Commit: algorithm I

- Each resource manager *i* has its own instance of Paxos *Pax*(*i*) to agree to *prepare* or *abort*.
  - If each instance chooses *prepare* then commit; else abort.
  - All instances use the same leader and set of acceptors.
    - Acceptors/leader know the set of RM involved in the transaction.

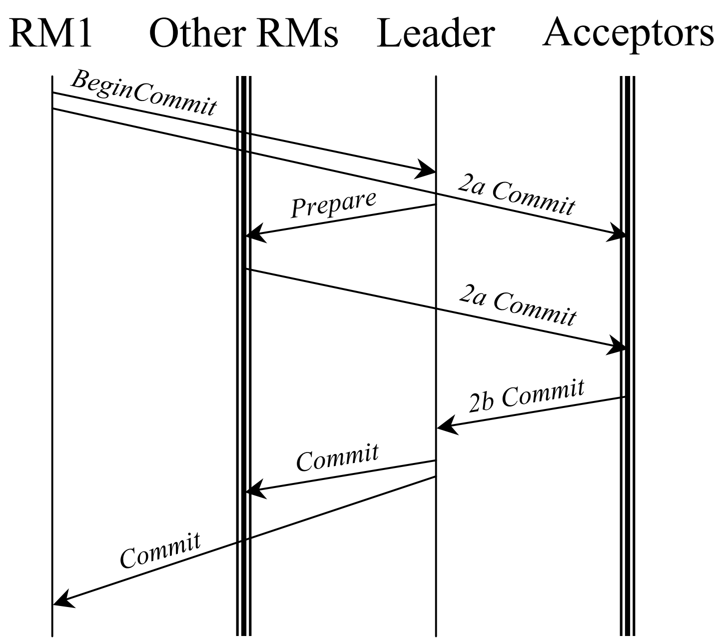
Paxos Commit: algorithm III

- When an RM *i* decides to prepare, it sends on behalf of the leader a phase 2a message *accept*(1, $vote_i$) in *Pax*(*i*).
  - This is a phase 2a ballot 1 message. There's no need for this to have to come from the leader nor to run phase 1.
- When leader receives $f + 1$ *learn* messages for *Pax*(*i*), it can send phase 3 messages announcing outcome to RMs.
- Transaction committed iff every RM chooses prepared; otherwise is aborted.
  - For efficiency, each acceptor can bundle phase 2b *learn* messages for all instances into one message.
  - Similarly, the leader can distill all phase 3 *learn* messages into one message declared *commit* or *abort*.

Paxos Commit: algorithm II

- If a new leader starts ballot > 1, then it runs phase 1.
  - If it finds its choice unconstrained, then it should propose *abort* in phase 2.
- In fact, the only way that an instance of Paxos will decide *prepared* is if the associated RM sends *accept*(1, *prepare*).
  - Thus, if an RM i has $vote_i = abort$, then can short-circuit and inform all processes that the decision is to abort.

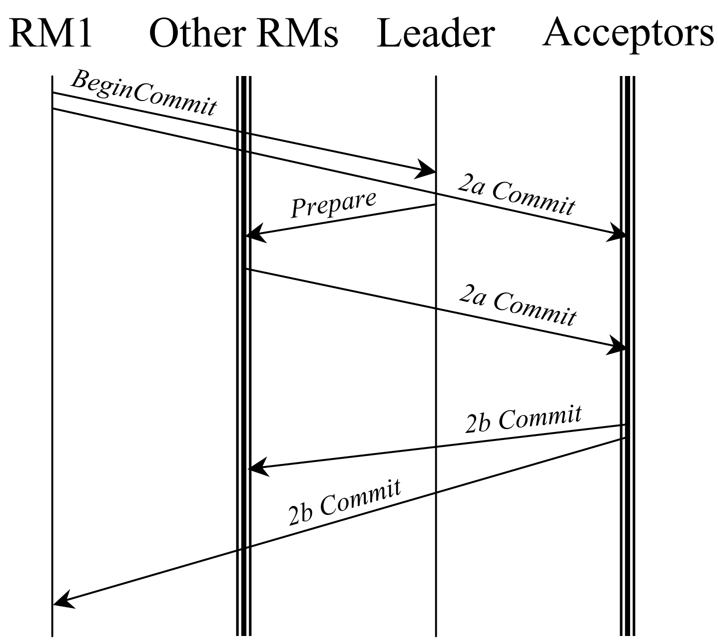
Cost: I



5 message delays, $(n+1)(t+3) - 4$ messages (with the leader an acceptor).

If each acceptor is on the same node as an RM, and the leader on RM1, then $n(t+3) - 3$ messages, but still 5 message delays.

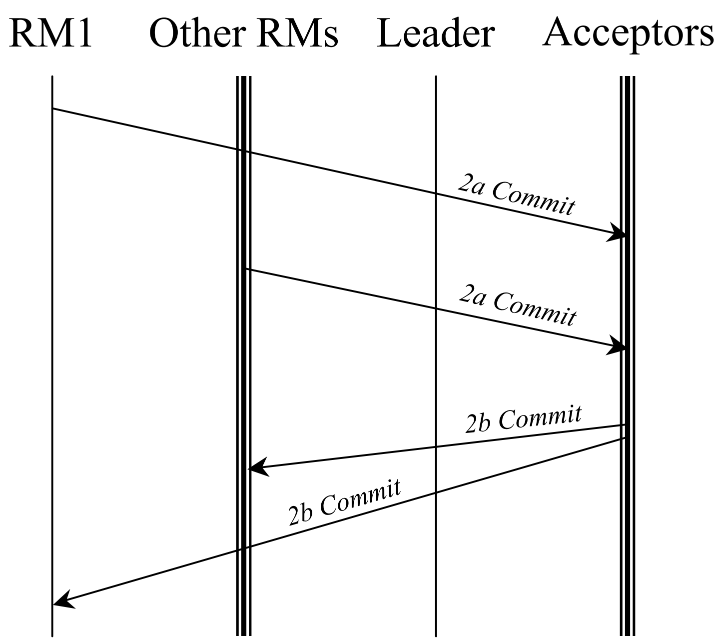
Cost: II



4 message delays, $n(2t+3) - 1$ messages

If each acceptor is on the same node as an RM, and the leader on RM1, then $(n-1)(2t+3)$ messages, but still 4 message delays.

Cost: III



2 message delays, which is optimal