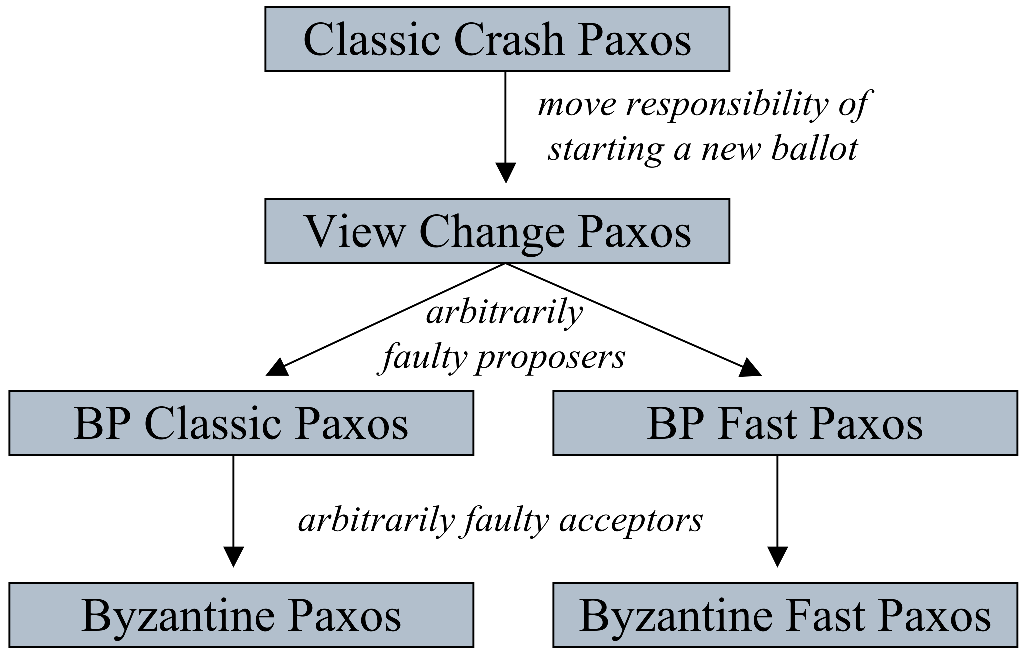
The Six Paxoi



Classic Crash Paxos ($n_p > t_p$, na $> 2t_a$)

Phase 1:

a. Proposer c:
   i. Selects a unique ballot 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.

Moving a responsibility

Change phase 1 so that it is driven by *acceptors* suspecting *proposers*.

- Have each ballot assigned to a proposer.
  - If there are P proposers 0, 1, …, $P-1$ then assign ballot i to proposer $p(i)$ = proposer (i mod P).
- When $trust_a$ changes to i, acceptor a sets rnd_a to the next highest ballot assigned to $p(i)$ and sends $promise(rnd_a, vrnd_a, vval_a)$ to $p(i)$.
- When $p(i)$ receives at least a quorum of such messages, it can choose a value and send the corresponding *accept* message.

… will call this a *view change* (even thought it could better be called a *ballot change*).

View Change Paxos ($n_p > t_p$, $n_a > 2t_a$)

Proposer $p(0)$ with $crnd_{p(0)} = 0$:

set $cval_c$ to any proposed value.

send $accept(crnd_{p(0)}, cral_{p(0)})$ to all acceptors.

Proposer $c = p(rnd_c)$ receives $promise(rnd_a, vrnd_a, vval_a)$ from $n_a - t_a$ acceptors with $rnd_a \geq crnd_c$:

set $crnd_c$ to rnd_a.

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 $vrnd_a$.

send $accept(crnd_c, cval_c)$ to all acceptors.

Acceptor a receives $accept(n, v)$:

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 $learn(n, v)$

else ignore request.

Acceptor a changes trust to $p(i)$:

set rnd_a to the next highest ballot of $p(i)$.

send $promise(rnd_a, vrnd_a, vval_a)$ to $p(rnd_a)$.

Arbitrarily faulty proposers

- Proposer can choose arbitrary value for 2a.
  - Can cause the chosen value to change.
- Proposer can send different *accept* messages to different acceptors.
  - Can lead to starvation or to violation of consensus.

Arbitrarily faulty proposers

Proposer can choose arbitrarily value for 2a.

Acceptors need to detect this case.

- Change *promise* from acceptor *a* to be *promise*($[vrnd_a, vval_a]_a$)
- Change *accept* message to contain the set of these signed values as the proof of why the proposer chose the value it did.
- Acceptors check this proof to validate the proposer's action.

Arbitrarily faulty proposers

Proposer can send different *accept* messages to different acceptors.

- Use an echo protocol similar to the Srikanth/Toueg approach.
- Use Fast Paxos approach.

Using echo protocol: I

- Need broadcast with the following properties:
  - If correct *c* broadcasts *m*, then all correct processes deliver *m*.
  - If correct *c* does not broadcast *m*, then all correct processes do not deliver *m*.
  - If faulty *c* broadcasts and correct *p* and *q* deliver messages from this broadcast, then they deliver the same message.

Using echo protocol: II

Assume $n > 3t$.

- c executes *bz-bcast*(m):
  send *pre*(m) to all

- receive *pre*(m) from c:
  if (have not yet sent echo for c)
  send *echo*(c, m) to all

- receive *echo*(c, m) from $n - t$ distinct processes:
  bz-deliver(m)

Using echo protocol: III

- If correct process *c bz-bcast m*, then all correct processes *bz-deliver m.*
  - $n - t$ correct processes will receive *pre*(m) and no other *pre*(m') for $m' \neq m$. All will send *echo*(c, m).
  - All correct processes will receive these $n - t$ *echo*(c, m) messages and *bz-deliver m*.

- If correct *c* does not broadcast *m*, then all correct processes do not deliver *m.*
  - A correct process can not receive more than t *echo*(c, m), one from each faulty process.
  - To *bz-deliver m*, $n - t \leq t$ or $n \leq 2t$, but $n > 3t$.

Using echo protocol: III

- If faulty c broadcasts and correct p and q deliver messages from this broadcast, then they deliver the same message.
  - Suppose p delivers m and q delivers m'.
    - p received $n - t$ $echo(c, m)$, of which at least $n - 2t$ came from nonfaulty processes.
    - Similarly, q received $n - t$ $echo(c, m')$, of which at least $n - 2t$ came from nonfaulty processes.
    - Thus, $n \geq 2(n - 2t) + t$ or $n \leq 3t$, but $n > 3t$.

Using echo protocol: IV

- Proposer $c = p(crnd_c)$ sends *pre-accept*($crnd_c$, $cval_c$, *proof*) to all proposers.

- Upon receipt of *pre-accept*(cr, cv, *proof*) from $c = p(cr)$ and has not yet sent *pre-accept* for this cr, proposer sends *accept*(c, cr, cv, *proof*) to all acceptors.

- Acceptor takes action once it receives $n - t$ identical *accept*(c, cr, cv, *proof*) messages.

BP Classic Paxos ($n_p > 3t_p$, $n_a > 2t_a$)

Proposer $p(0)$ with $crnd_{p(0)} = 0$:

set $cval_c$ to any proposed value.
send *pre-accept*($crnd_{p(0)}$, $cral_{p(0)}$, $\varnothing$) to all proposers.

Proposer $c = p(rnd_c)$ receives *promise*($[vrnd_a, vval_a]_a$) from $n_a - t_a$ acceptors with $rnd_a \geq crnd_c$:

set $crnd_c$ to rnd_a.
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 $vrnd_a$.
set *proof* to the set of received $[vrnd_a, vval_a]_a$.
send *pre-accept*($crnd_c$, $cval_c$, *proof*) to all proposers.

Proposer p receives *pre-accept*(n, v, *proof*) from proposer $c = p(rnd)$:

If not yet sent *accept* for c and n then send *accept*(c, n, v, *proof*) to all acceptors.

Acceptor a receives *accept*(c, n, v, *proof*) from $n - t_p$ proposers:

if *proof* supports c, n and v, $n \geq rnd_a$ and $vrnd_a \neq n$
  then set $vrnd_a$ and rnd_a to n and $vval_a$ to v, and send *learn*(n, v).
  else ignore request.

Acceptor a changes trust to $p(i)$:

set rnd_a to the next highest ballot of $p(i)$.
send *promise*($[vrnd_a, vval_a]_a$) to $p(rnd_a)$.

Using Fast Paxos approach

- Fast Paxos increased n_a to at least $3t_a + 1$ for fast rounds to allow a proposer to choose a unique value that *may have been or will be* chosen.
  - This was to handle the case that not all acceptors received the same proposed value.
  - This is the same situation that occurs here.
  - If acceptors are learners and observe a collision in ballot i, then it's proof that $p(i)$ is faulty.
    - Great time for a view change!

BP Fast Paxos ($n_p > t_p$, $n_a > 3t_a$)

Proposer $p(0)$ with $crnd_{p(0)} = 0$:

- set $cval_c$ to any proposed value.
- send $accept(crnd_{p(0)}, cral_{p(0)}, \varnothing)$ to all acceptors.

Proposer $c = p(rnd_c)$ receives $promise(rnd_a, [vrnd_a, vval_a]_a)$ from $n_a - t_a$ acceptors with $rnd_a \geq crnd_c$:

- Set $crnd_c$ to rnd_a.
- let MV be the multiset $[vrnd_a, vval_a]_a$ from the acceptors.
- discard values from V that do not occur at least $n_a - 2t_a$ times.
- let V be the set of $vval_a$ in MV with the largest value of $vrnd_a$.
- if $|V| = 1$ then set $cval_c$ to the value in V.
  - else if $|V| = 1$ then set $cval_c$ to any proposed value.
- set $proof$ to the set of $[vrnd_a, vval_a]_a$ from the acceptors.
- send $accept(crnd_c, cval_c, proof)$ to all proposers.

Acceptor a receives $accept(n, v, proof)$ from c:

- if $proof$ supports c, n and v, $n \geq rnd_a$ and $vrnd_a \neq n$
  - then set $vrnd_a$ and rnd_a to n and $vval_a$ to v, and send $learn(n, v)$.
  - else ignore request.

Acceptor a changes trust to $p(i)$:

- set rnd_a to the next highest ballot of $p(i)$.
- send $promise(rnd_a, [vrnd_a, vval_a]_a)$ to $p(rnd_a)$.

Arbitrarily Faulty Acceptors

- Can send fabricated *promise* messages
- Can send fabricated *learn messages*
  - Problems with picking value for *accept* message.

Arbitrarily Faulty Acceptors

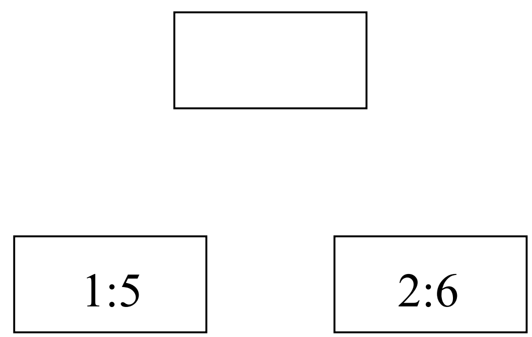
Can send fabricated *promise* messages

Can send fabricated *learn messages*

- The problem looks different for the *classic* and the *fast* protocols.

Classic approach, $n_a = 2t_a + 1$

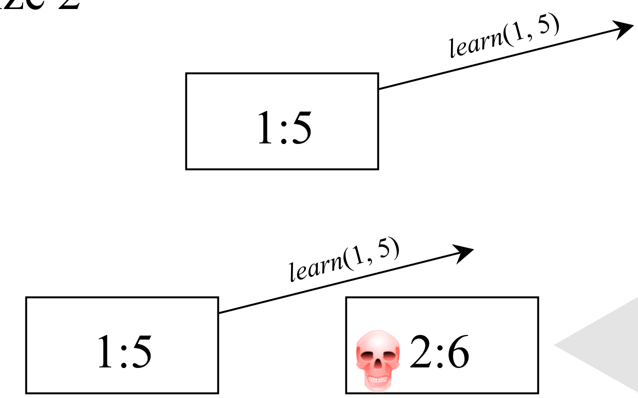
quorum size $t_a + 1$
eg, $n_a = 3$, $t_a = 1$, quorum size 2



proposer picks 6

Classic approach, $n_a = 2t_a + 1$

quorum size $t_a + 1$
eg, $n_a = 3$, $t_a = 1$, quorum size 2

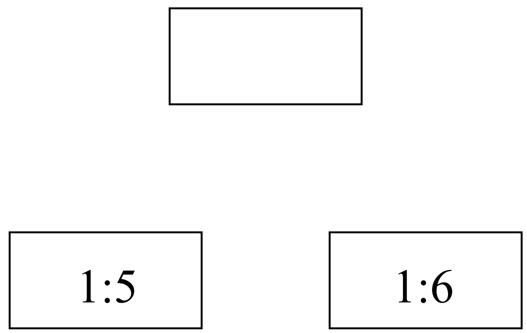


Even having p(2) digitally sign its *accept* messages does not help!

proposer picks 6
but 5 already learned!

Classic approach, $n_a = 2t_a + 1$

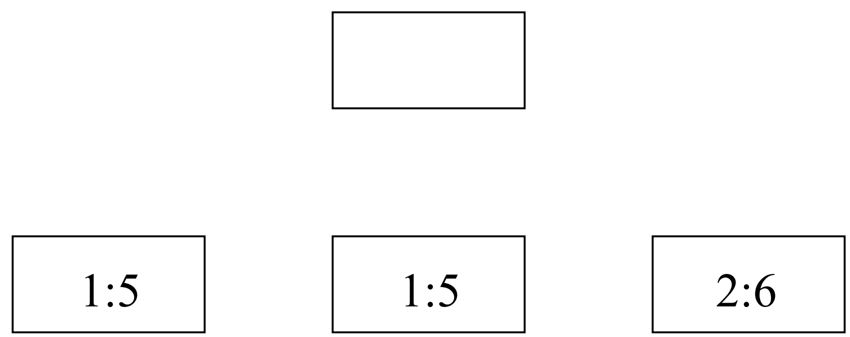
quorum size $t_a + 1$
eg, $n_a = 3$, $t_a = 1$, quorum size 2



what should proposer pick?

Classic approach, $n_a = 3t_a + 1$

quorum size $2t_a + 1$
eg, $n_a = 4$, $t_a = 1$, quorum size 3



2:6 must be from a faulty acceptor!
choose a value with the highest round that occurs at least $t_a + 1$ times (in general, at least $n - 2t_a$ times).
… this is the same choice rule for BP Fast Consensus.

Byzantine Paxos ($n_p > 3t_p$, $n_a > 3t_a$)

Proposer $p(0)$ with $crnd_{p(0)} = 0$:

set $cval_c$ to any proposed value.
send *pre-accept*($crnd_{p(0)}$, $cral_{p(0)}$, $\emptyset$) to all proposers.

Proposer $c = p(rnd_c)$ receives *promise*(rnd_a, $[vrnd_a, vval_a]_a$) from $n_a - t_a$ acceptors with $rnd_a \geq crnd_c$:

Set $crnd_c$ to rnd_a.
let MV be the multiset $[vrnd_a, vval_a]_a$ from the acceptors.
discard values from V that do not occur at least $n_a - 2t_a$ times.
let V be the set of $vval_a$ in MV with the largest value of $vrnd_a$.
if $|V| = 1$ then set $cval_c$ to the value in V.
 else if $|V| = 1$ then set $cval_c$ to any proposed value.
set *proof* to the set of $[vrnd_a, vval_a]_a$ from the acceptors.
send *accept*($crnd_c$, $cval_c$, *proof*) to all proposers.

Proposer p receives *pre-accept*(n, v, *proof*) from proposer $c = p(rnd)$:

If not yet sent *accept* for c and n then send *accept*(c, n, v, *proof*) to all acceptors.

Acceptor a receives *accept*(c, n, v, *proof*) from $n - t_p$ proposers:

if *proof* supports c, n and v, $n \geq rnd_a$ and $vrnd_a \neq n$
 then set $vrnd_a$ and rnd_a to n and $vval_a$ to v, and send *learn*(n, v).
 else ignore request.

Acceptor a changes trust to $p(i)$:

set rnd_a to the next highest ballot of $p(i)$.
send *promise*($[vrnd_a, vval_a]_a$) to $p(rnd_a)$.

Byzantine Paxos to BFT

Miguel Castro and Barbara Liskov. Proactive Byzantine Fault Tolerance and Proactive Recovery. *ACM TOCS* 20(4):398-461 (November 2002).

- A core Byzantine Paxos protocol as part of a state machine solution.
  - *n* processes, each taking on the role of *proposer*, *acceptor* and *learner*.
  - pre-accept → pre-prepare
  - accept → prepare
  - learn → commit
  - promise → view-change
  - use message authentication codes to protect all messages against attack.

BFT state machine

- Each process maintains
  - service state and implements service operations.
  - a *message log* that contains:
    - messages the replica has accepted or sent;
    - the replica's current view.
    - *checkpointing* is used to keep log's length bounded.

BFT, normal case ($n > 3t$)

Client c wishes to invoke operation o with timestamp t:

send $[\text{REQUEST}, o, t, c]_c$ to $p_0, p_1, \ldots, p_{n-1}$;

primary p receives $[m]_c$:

if (message authenticates)

assign sequence number n.

send $[\text{PRE-PREPARE}, v, n, D(m)]_p$ to $p_0, p_1, \ldots, p_{n-1}$; // view v, $D(m)$ is digest of m

// q has pre-prepared the request

replica q receives $[\text{PRE-PREPARE}, v, n, D(m)]_p$ from replica p

if (message authenticates) and (q is in view v) and ($p = p(v)$)

and (q has not accepted $[\text{PRE-PREPARE}, v, n, D(m')]_p$ where $m' \neq m$)

send $[\text{PREPARE}, v, n, D(m), q]_q$ to $p_0, p_1, \ldots, p_{n-1}$;

// q has pre-prepared the request

replica q receives matching authenticated $[\text{PREPARE}, v, n, D(m), q]_q$ from $2t + 1$ replicas://

// a has prepared the request

if (q is in view v)

send $[\text{COMMIT}, v, n, q]_q$ to $p_0, p_1, \ldots, p_{n-1}$;

replica q receives matching authenticated $[\text{COMMIT}, v, n, q]_q$ from $2t + 1$ replicas:

// q has committed request

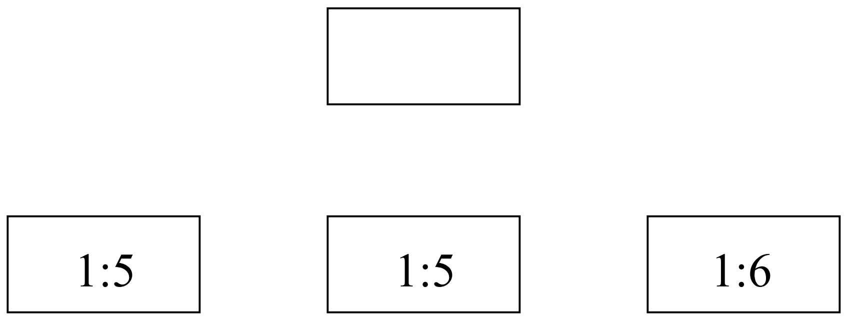
q executes o when all commands numbered less than n are executed;

BFT View Change

- Significantly more complex.
  - Processes exchange sets of pre-prepared and prepared messages.
  - New primary uses these to choose what to propose for each sequence number, filling with *no-ops* if choice is unconstrained.
  - Backups use same information to verify choices of new primary.

Fast approach, $n_a = 3t_a + 1$

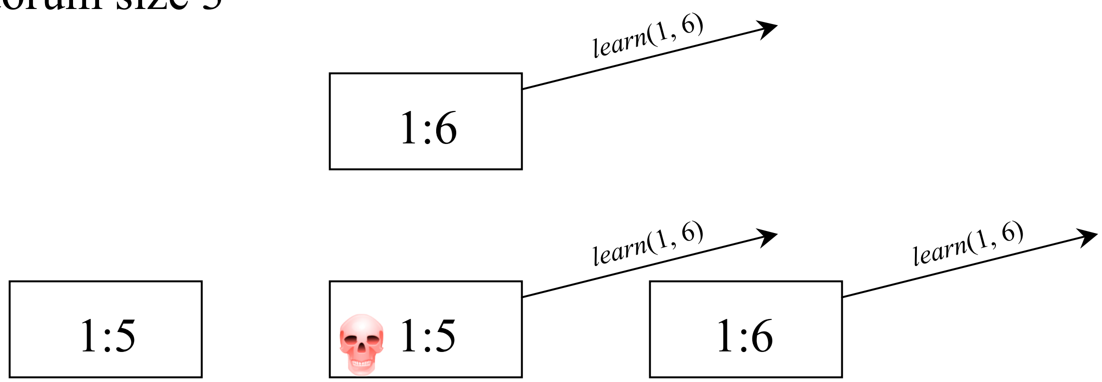
quorum size $2t_a + 1$
eg, $n_a = 4$, $t_a = 1$, quorum size 3



proposer picks 5

Fast approach, $n_a = 3t_a + 1$

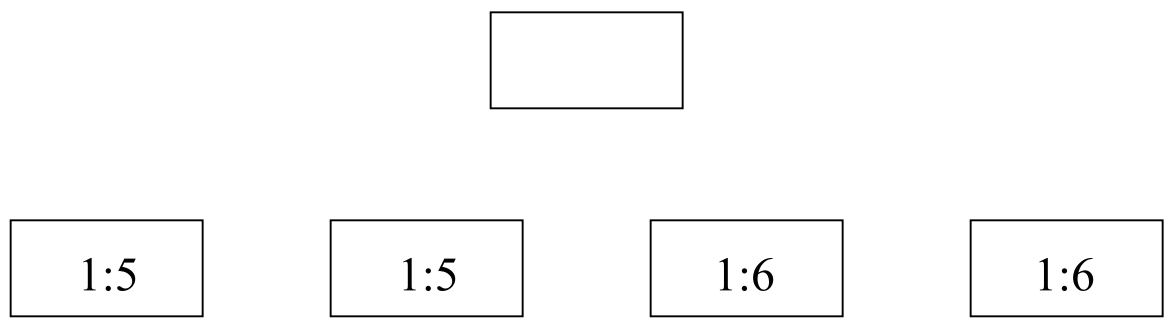
quorum size $2t_a + 1$
eg, $n_a = 4$, $t_a = 1$, quorum size 3



proposer picks 5
but 6 already learned!

Fast approach, $n_a = 4t_a + 1$

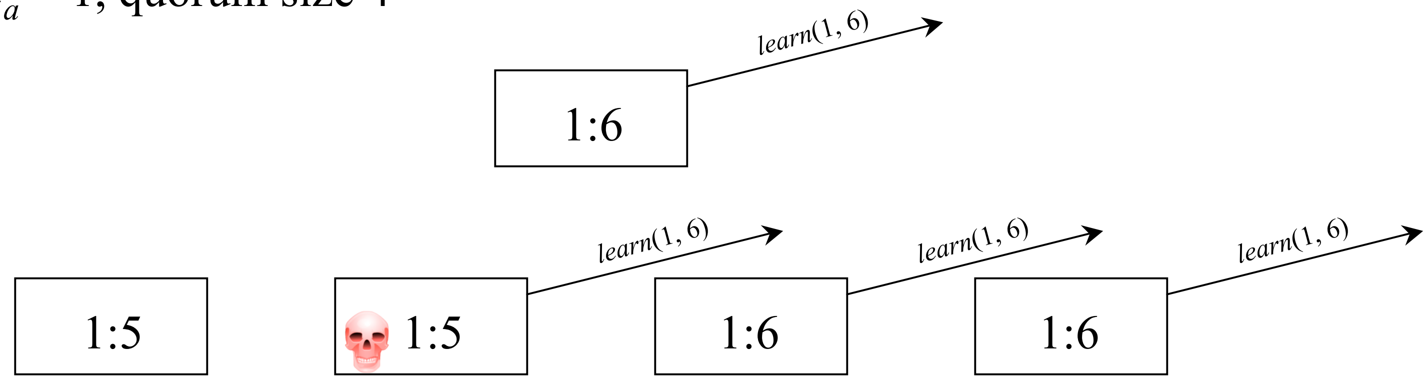
quorum size $3t_a + 1$
eg, $n_a = 5$, $t_a = 1$, quorum size 4



which value should proposer pick?

Fast approach, $n_a = 4t_a + 1$

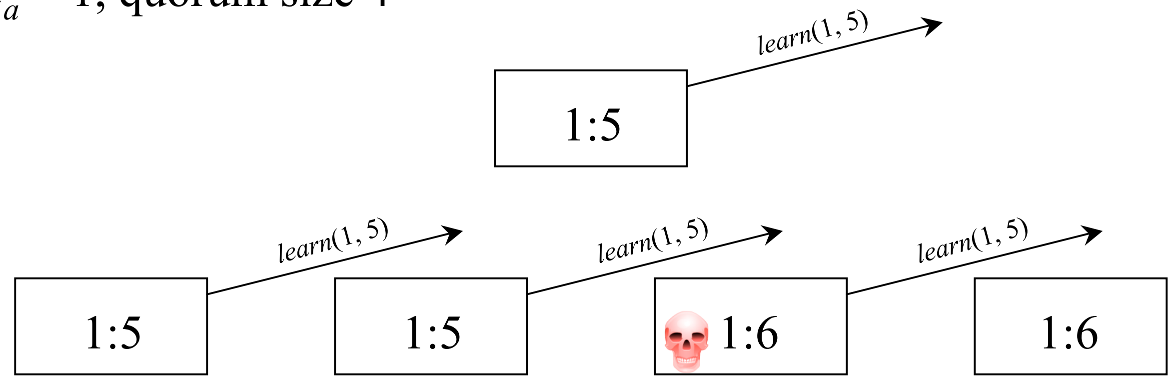
quorum size $3t_a + 1$
eg, $n_a = 5$, $t_a = 1$, quorum size 4



6 could have been learned

Fast approach, $n_a = 4t_a + 1$

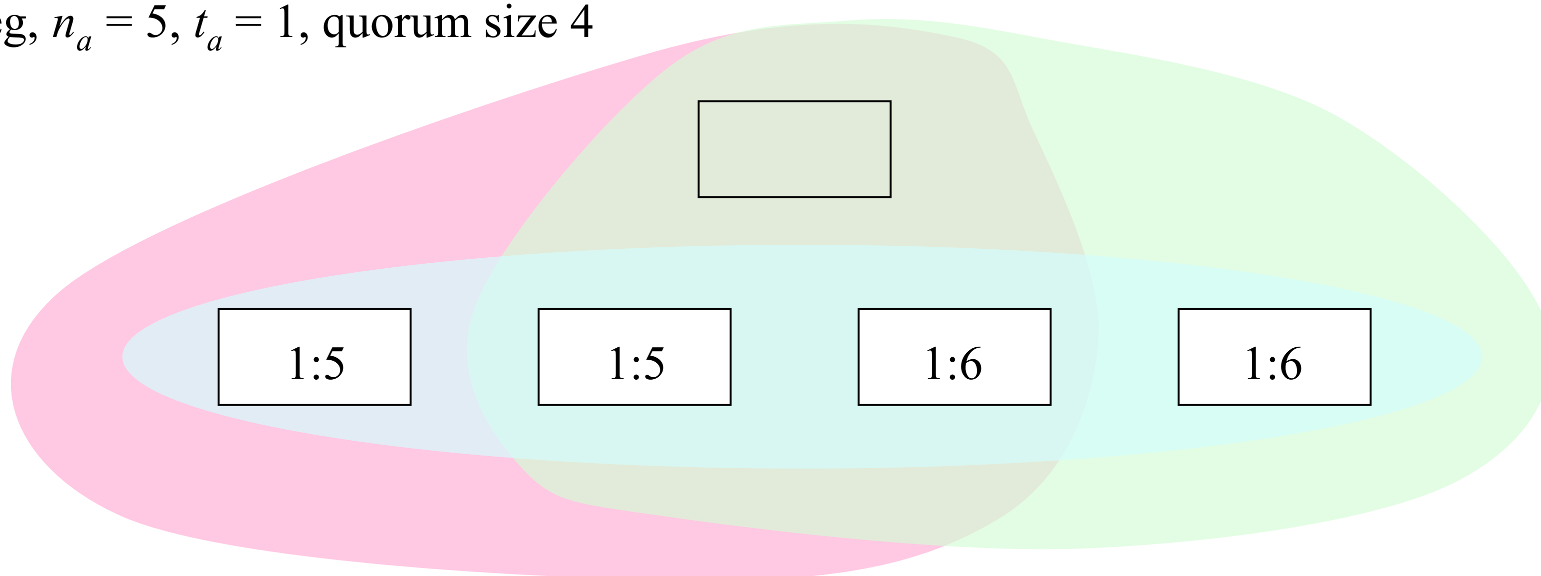
quorum size $3t_a + 1$
eg, $n_a = 5$, $t_a = 1$, quorum size 4



… and 5 could have been learned

Fast approach, $n_a = 4t_a + 1$

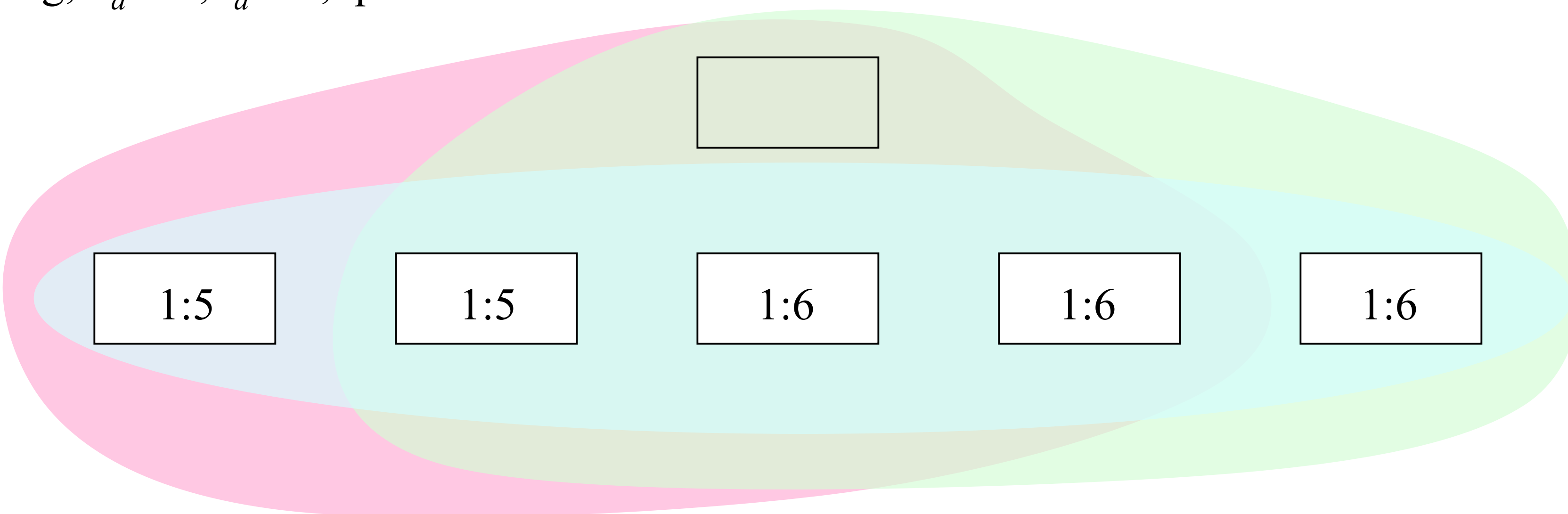
quorum size $3t_a + 1$
eg, $n_a = 5$, $t_a = 1$, quorum size 4



Three quorums whose intersection differ in x: $1 \leq x \leq t$ values.
… so ensure that intersection has at least $2t + 1$ values.

Fast approach, $n_a = 5t_a + 1$

quorum size $4t_a + 1$
eg, $n_a = 6$, $t_a = 1$, quorum size 5



Intersection has at least $2t + 1$ values, and so a correct acceptor has 1:6.
Thus, pick v if it occurs at least $2t + 1$ times (in general, $n - 3t$ times)

Byzantine Paxos ($n_p > t_p$, $n_a > 5t_a$)

Proposer $p(0)$ with $crnd_{p(0)} = 0$:

set $cval_c$ to any proposed value.
send $accept(crnd_{p(0)}, cral_{p(0)}, \varnothing)$ to all proposers.

Proposer $c = p(rnd_c)$ receives $promise(rnd_a, [vrnd_a, vval_a]_a)$ from $n_a - t_a$ acceptors with $rnd_a \geq crnd_c$:

Set $crnd_c$ to rnd_a.
let MV be the multiset $[vrnd_a, vval_a]_a$ from the acceptors.
discard values from V that do not occur at least $n_a - 3t_a$ times.
let V be the set of $vval_a$ in MV with the largest value of $vrnd_a$.
if $|V| = 1$ then set $cval_c$ to the value in V.
  else if $|V| = 1$ then set $cval_c$ to any proposed value.
set $proof$ to the set of $[vrnd_a, vval_a]_a$ from the acceptors.
send $accept(crnd_c, cval_c, proof)$ to all proposers.

Acceptor a receives $accept(n, v, proof)$ from c

if $proof$ supports c, n and v, $n \geq rnd_a$ and $vrnd_a \neq n$
  then set $vrnd_a$ and rnd_a to n and $vval_a$ to v, and send $learn(n, v)$.
  else ignore request.

Acceptor a changes trust to $p(i)$:

set rnd_a to the next highest ballot of $p(i)$.
send $promise([vrnd_a, vval_a]_a)$ to $p(rnd_a)$.

Fast Byzantine Consensus

Martin, J. and L. Alvisi. Fast Byzantine Consensus. *IEEE Transactions on Dependable Secure Computing* 3(3):202-215, July 2006.

- The duty of leader election is moved to the proposers as part of support for retransmission.
  - Learners report to proposer c that they have learned so c can stop transmitting.
  - Proposers need to know that c is making progress and so learners notify other proposers as well.
    - When a proposer p receives $2t_l + 1$ notifications it notifies c
      - Otherwise it suspects c
      - When $2t_p + 1$ suspect c then a new leader is elected.
        - $n_p > 3t_p$: echo protocol bounds.
    - When c receives $2t_p + 1$ notifications, it stops retransmitting.
- Thus $n_p > 3t_p$, $n_a > 5t_a$, $n_l > 3t_l$
  - Can be reduced to $n_l > 2t_l$ by signing *learn* messages: when $t_l + 1$ learners generate and sign messages: once $t_l + 1$ have signed, forward to c.