Elections

Introduction

The elections component includes components that allow accounts to become representatives, and components that allow those representatives to vote for delegates. Staking itself is a closely related but separate component.

Nearly every command supported by the elections component has a one epoch lag before it takes effect. For instance, to become a candidate, an account will announce their candidacy in one epoch, and then potentially be voted for in the following epoch. The thought process behind this decision is that while voting is happening, very few aspects of the system are changing. The system changes all at once during epoch transition. This gives our system some level of stability.

Furthermore, many actions surrounding voting, delegate candidacy and representative status are only allowed once per epoch. A representative can vote once per epoch, a delegate candidate cannot declare candidacy and then renounce that candidacy in the same epoch, etc.

Lastly, delegates serve a fixed length term, currently 4 consecutive epochs. However, delegates are allowed to run for reelection in such a way that they serve consecutive terms. In fact, the long term vision of the network is to have a stable delegate set, that changes very little between epochs.

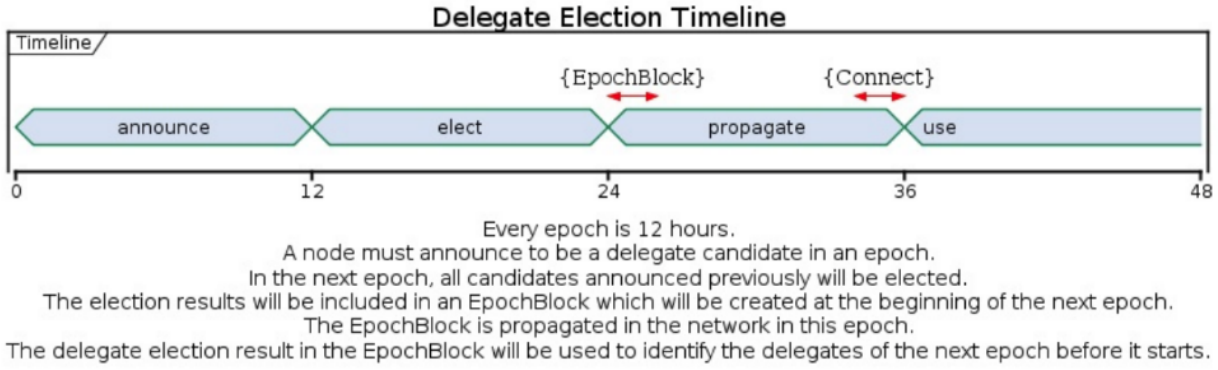
Overview

The elections component is really not a single component but a collection of additions to existing components.

- New structs that inherit from `Request`
- Logic within consensus to process these new `Request` types
- Representatives database and candidates database
- Logic during epoch transition and epoch block creation

The state diagrams and sequence diagrams are at the end of the document, as these are simply visualizing the requirements.

Timeline



Execution Concept

The processing of Requests related to elections follows the same execution model as processing any other type of Request (Send for instance) and spawns no new thread or does anything special.

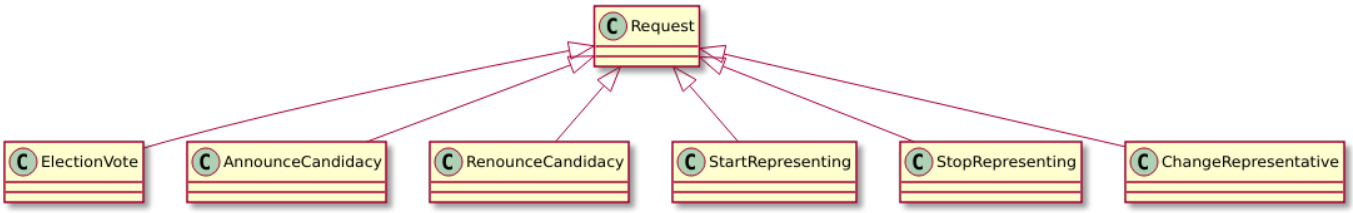
The updating of the candidates database on epoch transition is done as part of applying the updates for an epoch block. We will also delete any retiring representatives from the representatives database when applying the updates for an epoch block, but this will be done asynchronously in another thread, as nothing depends on deleting the representatives. It is simply a pruning operation.

Voting will be rejected during the last ten minutes of an epoch, as well as the first ten minutes. This is to avoid race conditions on the epoch boundary. Candidacy requests and representative requests can still be processed during this time.

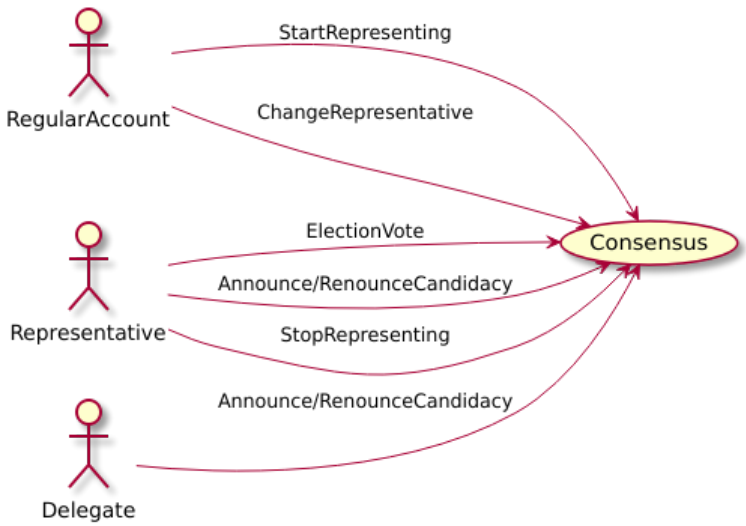
The election results, along with delegate voting power, will be calculated in another thread when the epoch starts. The results are used during epoch block creation and validation ten minutes later.

Interfaces

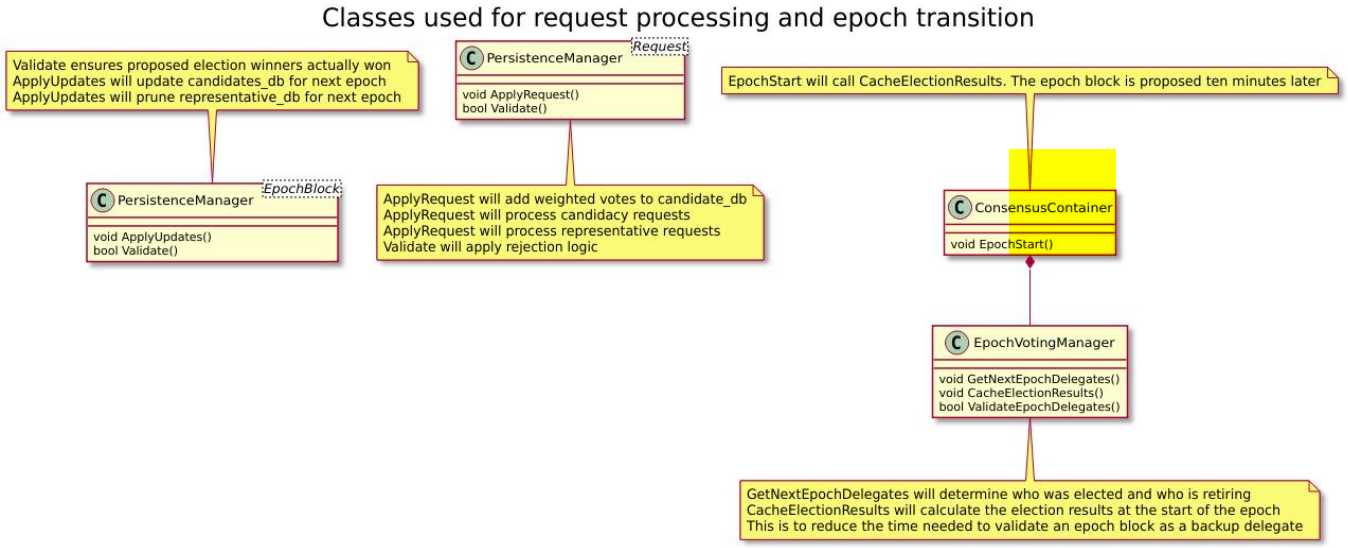
The interface to the election system is through issuing the proper `Request` which is processed through consensus.



Use Case Diagrams



Classes



Resources

Elections will introduce two new databases. An actual database format will be proposed separately from the design document.

- **Representative_db:** this database will store all of the representatives and any additional information about those representatives (voting weight, hash of their most recent vote, representative status)
- **Candidates_db:** this database will store all of the current candidates, along with the amount of weighted votes that they have received. Entries will have a flag whether the candidate is active or pending. Pending candidates are those that have announced this epoch, but cannot receive votes until next epoch

The representatives database will store a hash to the most recent representative status action. If a representative renounces, it can still vote in that epoch, but will be pruned from the database in the next epoch.

Candidate Set

Calculating elections winners:

```
Read candidates_db
Sort by weighted votes_received
Select top 8 as winners
Resolve ties by selecting delegate with most stake, followed by most epochs spent as delegate, followed by
greatest account address
```

Calculating next epochs candidates:

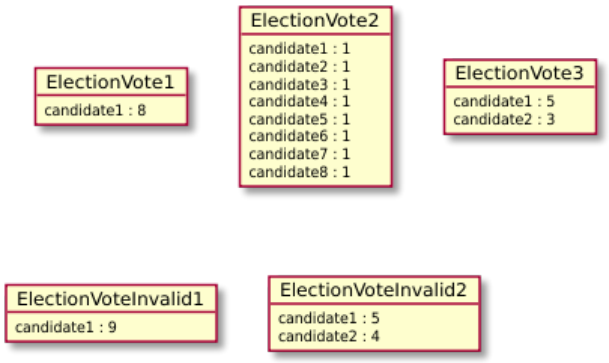
```
for(candidate c in candidates_db):
    if(c was elected or c issued renounce candidacy last epoch)
        remove c from candidates_db
    if(c is marked as pending)
        mark c as active
```

```
for(delegate d in current_delegates):
    if(d is in second to last epoch and hasn't renounced)
        add d to candidates_db
        mark d as active
```

Note that candidates can be tied with 0 votes.

Votes and Vote Weighting

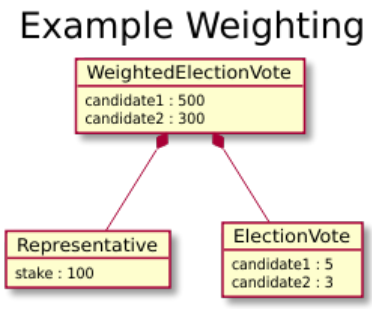
Each epoch we are electing `N/L` delegates, where `N` is the total number of delegates and `L` is the term length. `ElectionVote` can contain votes for up to `N/L` different candidates. The votes are represented like a map from candidates to number of votes. In the diagram below, assuming `N/L == 8`, `ElectionVote1`, `ElectionVote2` and `ElectionVote3` are valid, while the others are not. Any `ElectionVote` that lists an account that is not an active candidate will be rejected entirely.



Weighting votes relies on staking. Staking is out of scope for this project, so we will simply assume a representative has a stake. Below is pseudocode for how a single vote is weighted and processed.

```
weightVoteAndAddToDb(ElectionVote v)
    validate v
    stake = sender's stake
    for(Candidate c in v)
        weighted_vote = (num_votes for c in v) * stake
        add weighted_vote to current total votes for c in candidate_db
```

Below is an example representative, a vote that they cast and the resulting weighted vote. Note, WeightedElectionVote is not an actual class, but simply a concept used here to illustrate weighting.



Delegate Voting Power

During consensus, each delegate has a specific voting power, fixed for the entire epoch, that is proportional to the number of votes they receiv during elections. The function to calculate voting power is stated naively here for clarity, but can be optimized. The function is first stated in

english, and then specific pseudocode. We are capping voting power at 1/8 of total voting power, and in the rare case a delegate received 0 votes but was elected still, we give them 1 vote for free

English:

```
if a delegate has 0 votes, give it 1 vote

pick an unprocessed delegate
if delegate has greater than cap votes
    redistribute excess votes to unprocessed delegates, proportional to number of votes those delegates have
mark delegate as processed
repeat until all delegates are processed
```

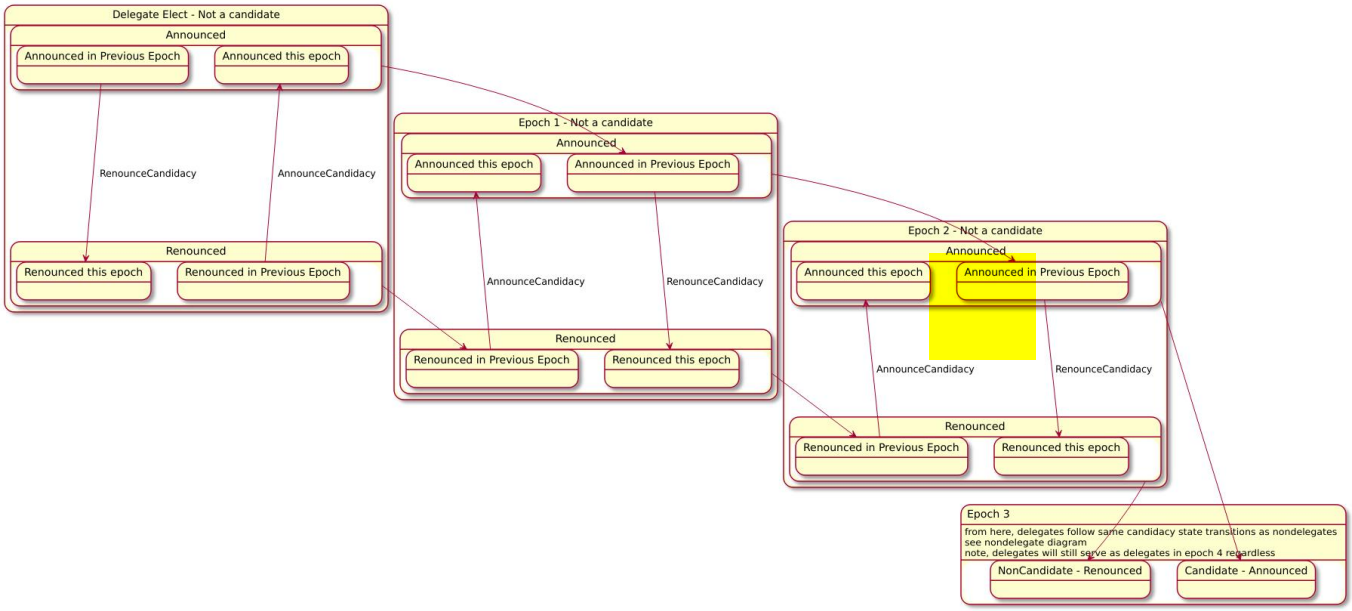
Pseudocode:

```
votes_i = votes delegate i received when elected, or 1 if delegate received 0 votes
total_votes = sum of all votes_i
votes_remaining = total_votes
cap = total_votes * 1/8
for each votes_i in sorted order:
    if votes_i > cap:
        excess = votes_i - cap
        votes_i = cap
        votes_remaining -= votes_i
        for each votes_j not yet processed:
            votes_j_ratio = votes_j / votes_remaining
            votes_j += excess * votes_j_ratio
        votes_remaining += excess
    votes_i.processed = true
```

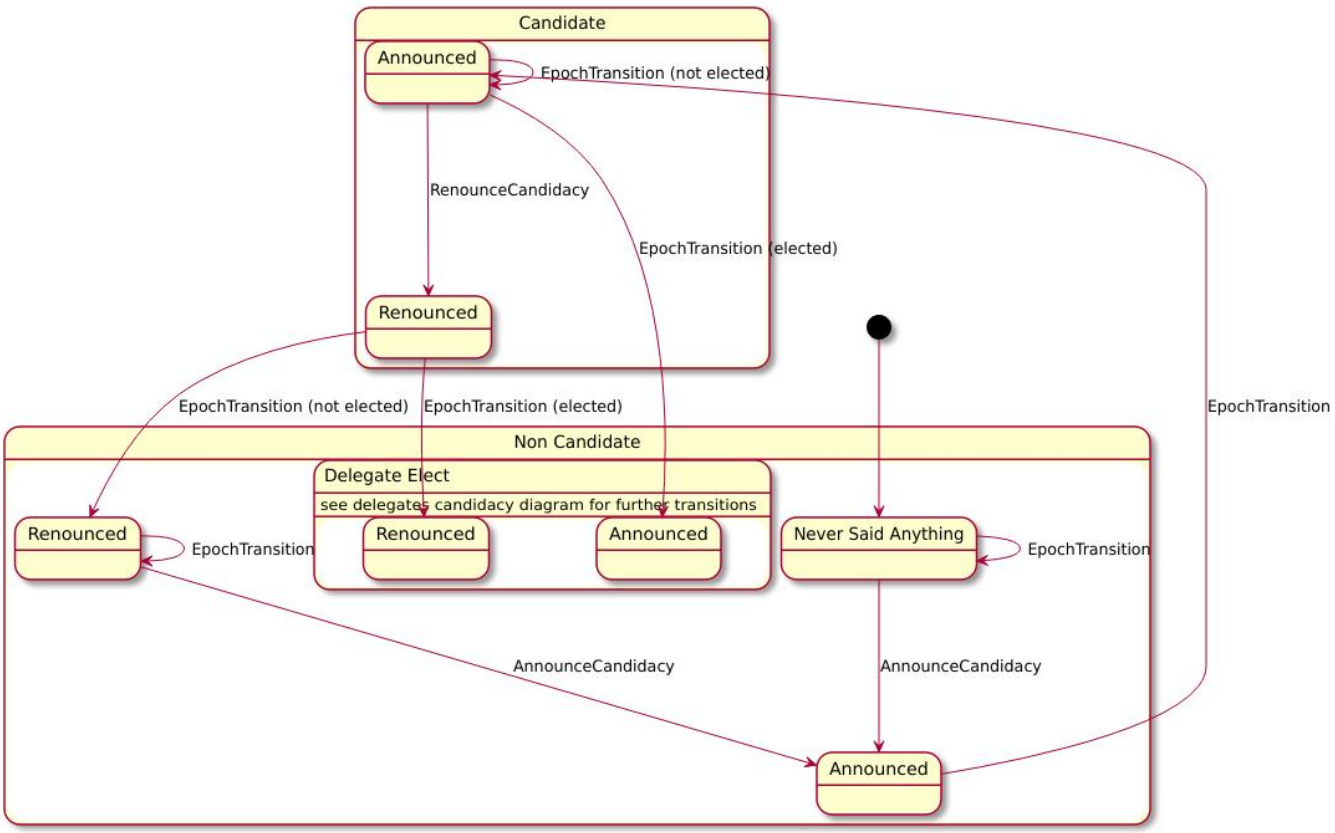
This function ensures no delegate receives more than 1/8 voting power, every delegate has greater than 0 voting power, and, if two delegates have less than 1/8 voting power after redistribution, their voting power relative to each other is the same before and after redistribution. When giving 1 vote as a freebie, that delegate is receiving at most 1/32 of voting power, since every other delegate will also have at least 1 vote as well. If you want to change the cap, simply replace 1/8 with whatever you would want the proportional cap to be.

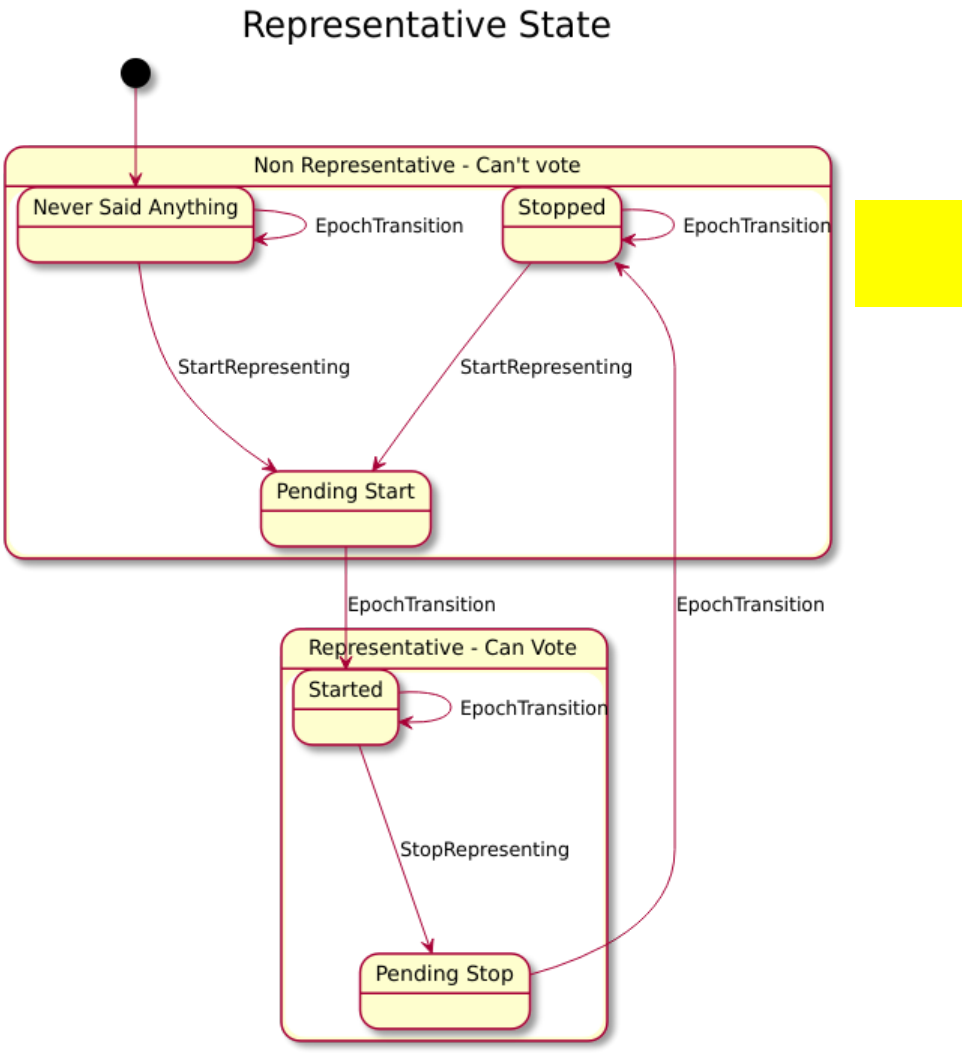
State Diagram

Candidacy State for Delegates

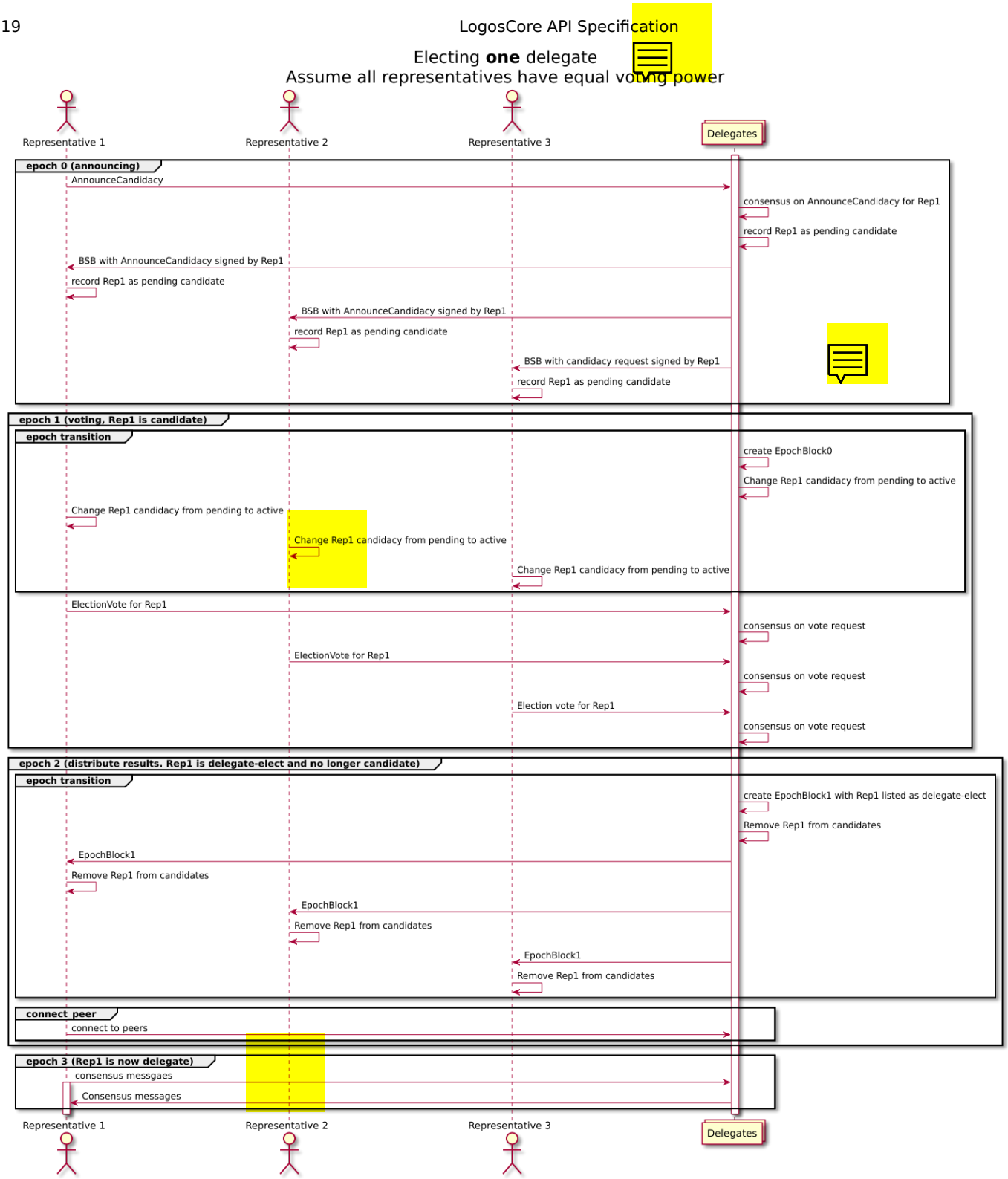


Candidacy State for NonDelegates

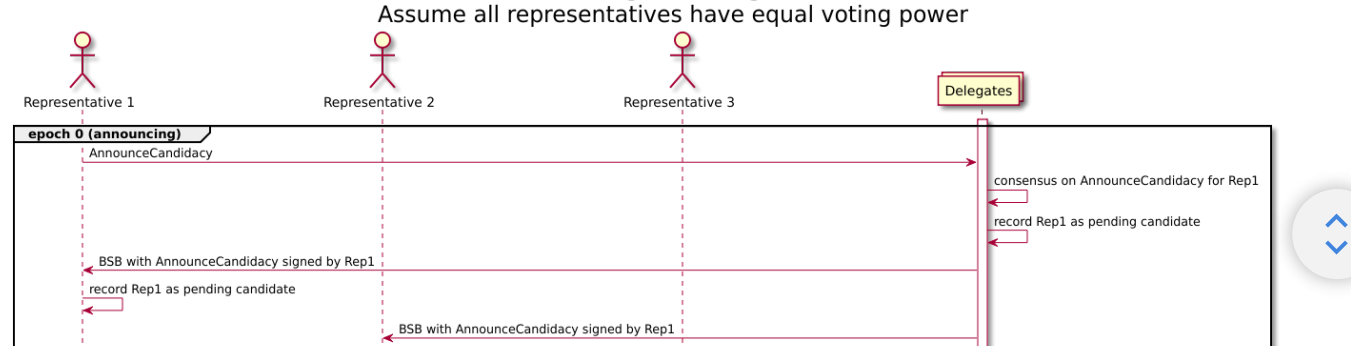


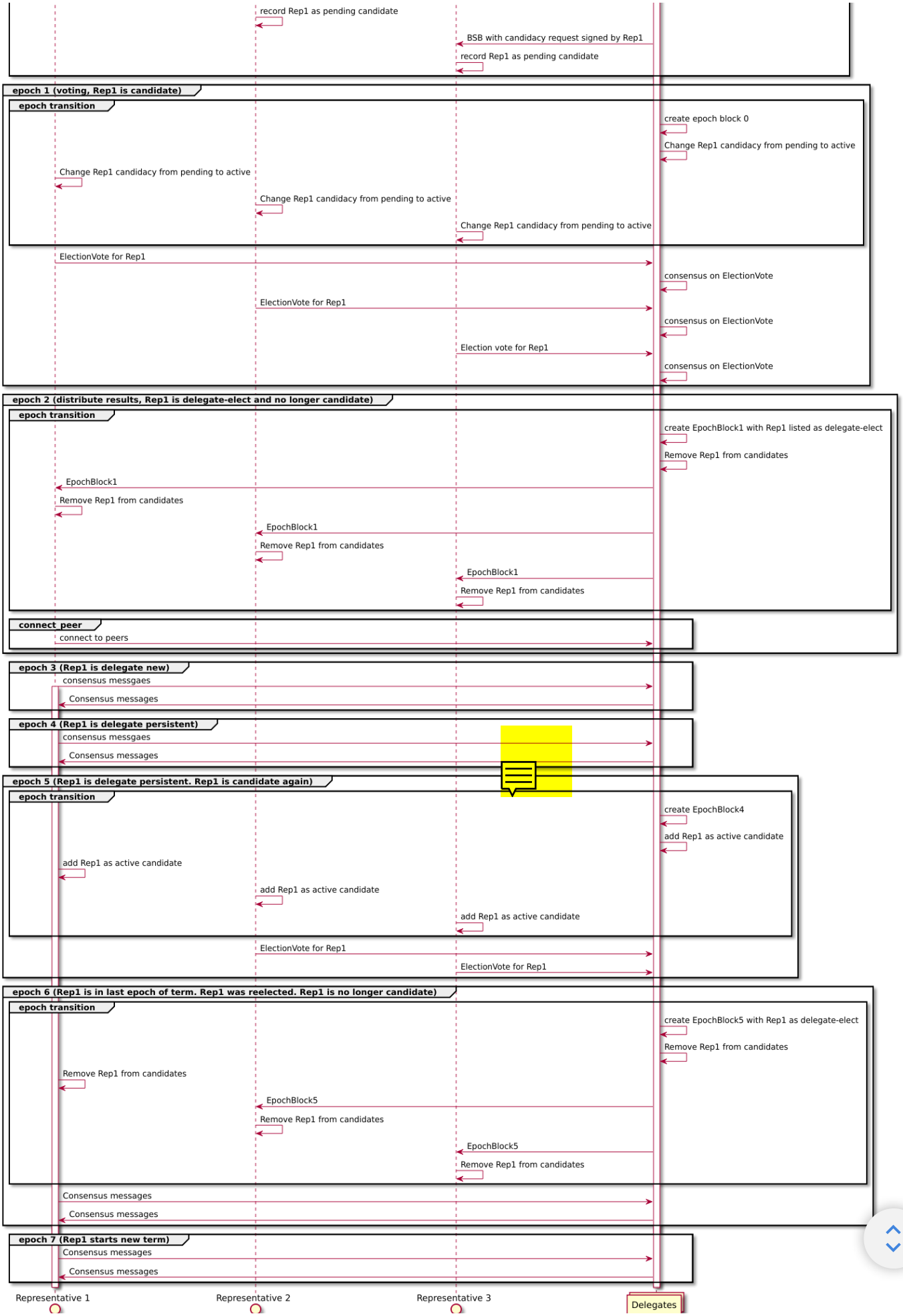


Scenarios and Sequence Diagrams

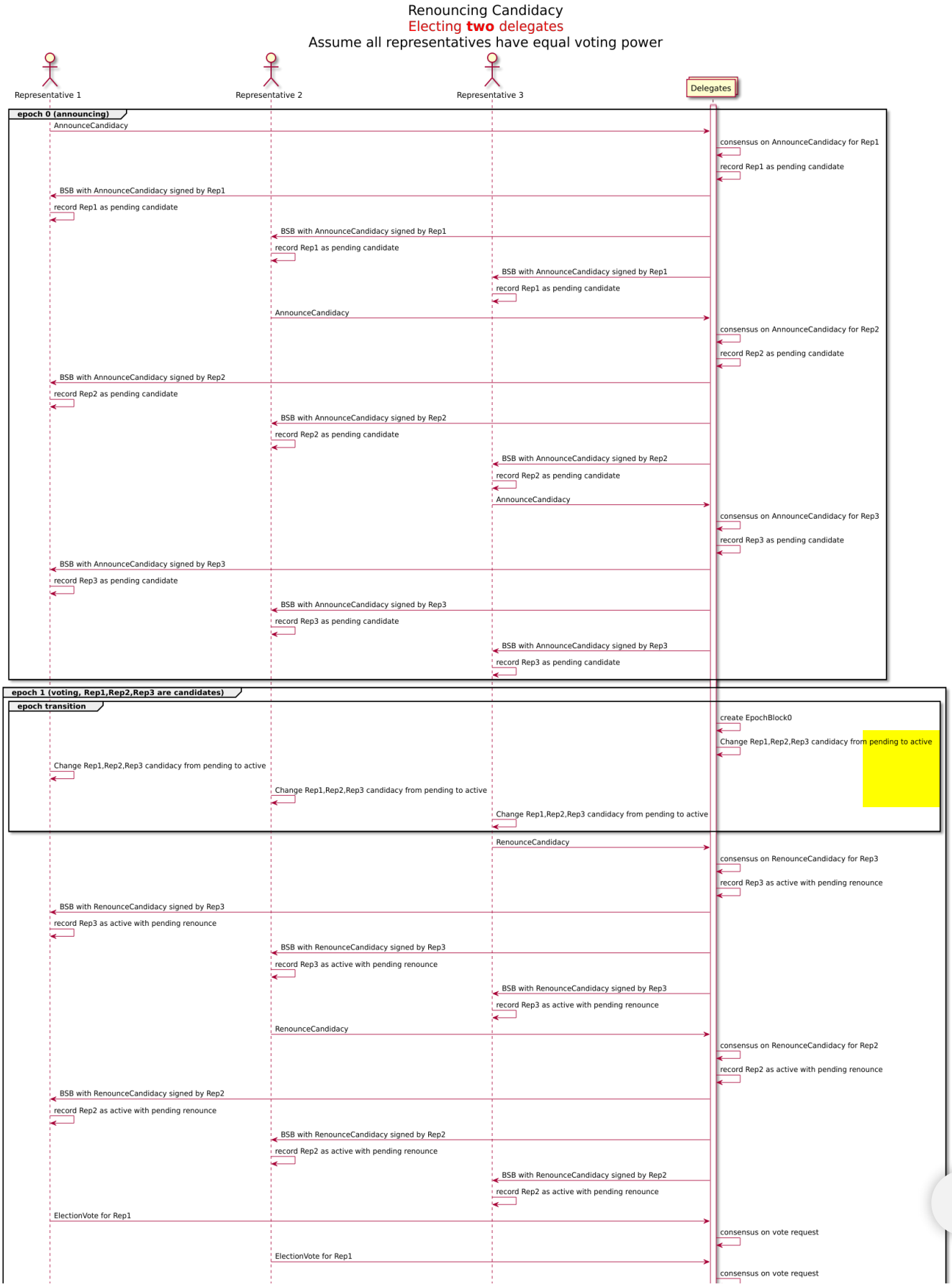


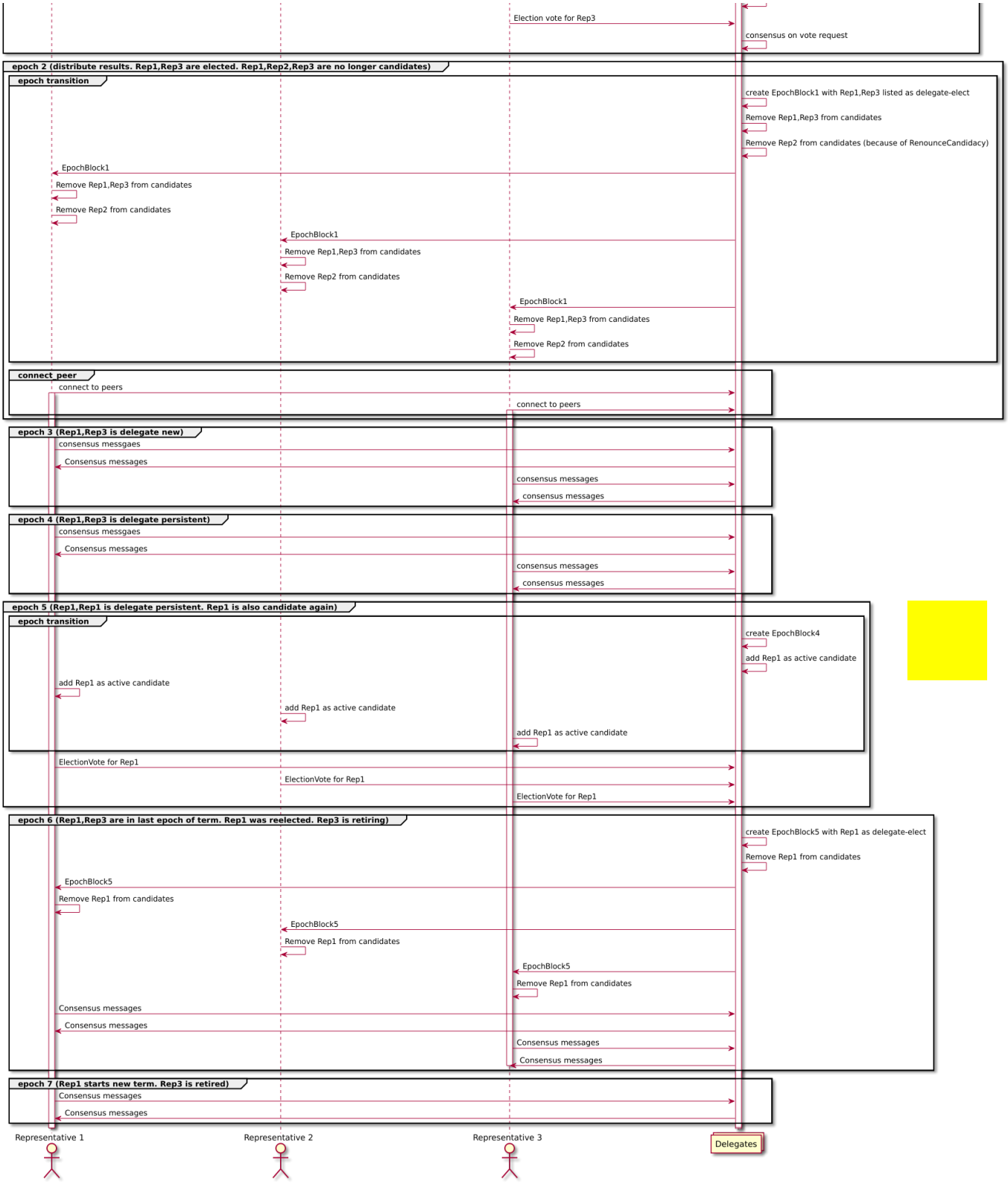
Reelections




record Rep1 as pending candidate
BSB with candidacy request signed by Rep1
record Rep1 as pending candidate
epoch 1 (voting, Rep1 is candidate)
epoch transition
create epoch block 0
Change Rep1 candidacy from pending to active
Change Rep1 candidacy from pending to active
Change Rep1 candidacy from pending to active
Change Rep1 candidacy from pending to active
ElectionVote for Rep1
consensus on ElectionVote
ElectionVote for Rep1
consensus on ElectionVote
Election vote for Rep1
consensus on ElectionVote
epoch 2 (distribute results, Rep1 is delegate-elect and no longer candidate)
epoch transition
create EpochBlock1 with Rep1 listed as delegate-elect
Remove Rep1 from candidates
EpochBlock1
Remove Rep1 from candidates
EpochBlock1
Remove Rep1 from candidates
EpochBlock1
Remove Rep1 from candidates
connect_peer
connect to peers
epoch 3 (Rep1 is delegate new)
consensus messgaes
Consensus messages
epoch 4 (Rep1 is delegate persistent)
consensus messgaes
Consensus messages
epoch 5 (Rep1 is delegate persistent. Rep1 is candidate again)
epoch transition
create EpochBlock4
add Rep1 as active candidate
add Rep1 as active candidate
add Rep1 as active candidate
add Rep1 as active candidate
ElectionVote for Rep1
ElectionVote for Rep1
epoch 6 (Rep1 is in last epoch of term. Rep1 was reelected. Rep1 is no longer candidate)
epoch transition
create EpochBlock5 with Rep1 as delegate-elect
Remove Rep1 from candidates
Remove Rep1 from candidates
EpochBlock5
Remove Rep1 from candidates
EpochBlock5
Remove Rep1 from candidates
Consensus messages
Consensus messages
epoch 7 (Rep1 starts new term)
Consensus messages
Consensus messages
Representative 1
Representative 2
Representative 3
Delegates

The below diagram shows two examples of renouncing. Representative 2 renounces, and is removed from the candidate list. Representative 3 renounces, but is elected and must still serve as a delegate. However, this renounce prevents Representative 3 from being added as a candidate for reelection.




Election vote for Rep3
consensus on vote request
epoch 2 (distribute results. Rep1,Rep3 are elected. Rep1,Rep2,Rep3 are no longer candidates)
epoch transition
create EpochBlock1 with Rep1,Rep3 listed as delegate-elect
Remove Rep1,Rep3 from candidates
Remove Rep2 from candidates (because of RenounceCandidacy)
EpochBlock1
Remove Rep1,Rep3 from candidates
Remove Rep2 from candidates
EpochBlock1
Remove Rep1,Rep3 from candidates
Remove Rep2 from candidates
EpochBlock1
Remove Rep1,Rep3 from candidates
Remove Rep2 from candidates
connect_peer
connect to peers
connect to peers
epoch 3 (Rep1,Rep3 is delegate new)
consensus messgaes
Consensus messages
consensus messages
consensus messages
epoch 4 (Rep1,Rep3 is delegate persistent)
consensus messgaes
Consensus messages
consensus messages
consensus messages
epoch 5 (Rep1,Rep1 is delegate persistent. Rep1 is also candidate again)
epoch transition
create EpochBlock4
add Rep1 as active candidate
add Rep1 as active candidate
add Rep1 as active candidate
add Rep1 as active candidate
ElectionVote for Rep1
ElectionVote for Rep1
ElectionVote for Rep1
epoch 6 (Rep1,Rep3 are in last epoch of term. Rep1 was reelected. Rep3 is retiring)
create EpochBlock5 with Rep1 as delegate-elect
Remove Rep1 from candidates
EpochBlock5
Remove Rep1 from candidates
EpochBlock5
Remove Rep1 from candidates
EpochBlock5
Remove Rep1 from candidates
Consensus messages
Consensus messages
Consensus messages
Consensus messages
epoch 7 (Rep1 starts new term. Rep3 is retired)
Consensus messages
Consensus messages
Representative 1
Representative 2
Representative 3
Delegates

→ Next to **Echo**