Raft
Leader Election
Logs are 1-indexed

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>currentTerm</code></td>
<td>latest term server has seen</td>
</tr>
<tr>
<td><code>votedFor</code></td>
<td>candidate ID that received vote in current term, or -1 if none</td>
</tr>
<tr>
<td><code>commitIndex</code></td>
<td>index of highest log entry known to be committed</td>
</tr>
<tr>
<td><code>lastApplied</code></td>
<td>index of highest log entry applied to state machine</td>
</tr>
<tr>
<td><code>nextIndex</code></td>
<td>for each server, index of the next log entry to send to that server</td>
</tr>
<tr>
<td><code>matchIndex</code></td>
<td>for each server, index of highest log entry known to be replicated on the server</td>
</tr>
</tbody>
</table>

(Only on leader)
State required for election

<table>
<thead>
<tr>
<th>currentTerm</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>votedFor</td>
<td>-1</td>
</tr>
</tbody>
</table>

currentTerm: latest term server has seen

votedFor: candidate ID that received vote in current term, or -1 if none
Recap: Leader Election

Everyone sets a randomized timer that expires in \([T, 2T]\) (e.g. \(T = 150\text{ms}\))

When timer expires, increment term and send a RequestVote to everyone

Retry this until either:

- You get majority of votes (including yourself): become leader
- You receive an RPC from a valid leader: become follower again
Scenario 1: During System Bootup
Timeout

0

<table>
<thead>
<tr>
<th>currentTerm</th>
<th>votedFor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

<empty>

1

<table>
<thead>
<tr>
<th>currentTerm</th>
<th>votedFor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

<empty>

2

<table>
<thead>
<tr>
<th>currentTerm</th>
<th>votedFor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

<empty>
RequestVote
Term: 1
CandidateID: 0
LastLogIndex: -1
LastLogTerm: -1
RequestVoteReply
Term: 1
VoteGranted: true

0
currentTerm  1
votedFor    0
<empty>

1
currentTerm  1
votedFor    0
<empty>

2
currentTerm  1
votedFor    0
<empty>
Scenario 2: During Normal Execution

(suppose there are existing log entries...)
0 currentTerm 3
votedFor 1

1 1 1 2 3

Timeout

1 currentTerm 3
votedFor 1

1 1 1 2 3

2 currentTerm 3
votedFor 1

1 1 1 2 3
RequestVote
Term: 4
CandidateID: 0
LastLogIndex: 5
LastLogTerm: 3

0
  currentTerm 4
  votedFor 0

1
  currentTerm 3
  votedFor 1

2
  currentTerm 3
  votedFor 1
RequestVoteReply
Term: 4
VoteGranted: True
Conditions for granting vote

1. We did not vote for anyone else in this term

2. Candidate term must be $\geq$ ours

3. Candidate log is at least as up-to-date as ours
   
   a. The log with higher term in the last entry is more up-to-date
   
   b. If the last entry terms are the same, then the longer log is more up-to-date
Which one is more *up-to-date*?
Which one is more *up-to-date*?
Which one is more *up-to-date*?

1 1 1 2 3

1 1 4 ✔
Why reject logs that are not *up-to-date*?

Leader log is always the ground truth

Once someone is elected leader, followers must throw away conflicting entries

Must NOT throw away committed entries!

*Note: Log doesn’t need to be the MOST up-to-date among all servers*
What if we accept logs that are not as up-to-date as ours?
Suppose entries 4-5 have already been committed.

Then previous leader S0 crashes and S3 times out.

If S3 becomes leader then committed entries 4 and 5 may be overwritten!
Why is it OK to throw away these entries?

If these entries had been committed, then it means they must exist on a majority of servers.

In that case S4 could receive votes from the same majority and become a valid leader.
Raft
Normal Operation
currentTerm: latest term server has seen
votedFor: candidate ID that received vote in current term, or -1 if none
commitIndex: index of highest log entry known to be committed
lastApplied: index of highest log entry applied to state machine

(next on leader)
nextIndex: for each server, index of the next log entry to send to that server
matchIndex: for each server, index of highest log entry known to be replicated on the server

Logs are 1-indexed
<table>
<thead>
<tr>
<th>Node</th>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>[1, 1, 1]</td>
<td>[0, 0, 0]</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>[]</td>
<td>[]</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>[]</td>
<td>[]</td>
</tr>
</tbody>
</table>

**AppendEntries**
- Term: 1
- LeaderID: 0
- PrevLogIndex: 0
- PrevLogTerm: -1
- LeaderCommit: 0
<table>
<thead>
<tr>
<th></th>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>[1, 1, 1]</td>
<td>[0, 0, 0]</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>[]</td>
<td>[]</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>[]</td>
<td>[]</td>
</tr>
</tbody>
</table>

**AppendEntriesReply**

Term: 1
Success: True
Client

Request 1

currentTerm 1
votedFor 0
commitIndex 0
lastApplied 0
nextIndex [1, 1, 1]
machineIndex [0, 0, 0]
<empty>

1
currentTerm 1
votedFor 0
commitIndex 0
lastApplied 0
nextIndex []
machineIndex []
<empty>

2
currentTerm 1
votedFor 0
commitIndex 0
lastApplied 0
nextIndex []
machineIndex []
<empty>
currentTerm | votedFor | commitIndex | lastApplied | nextIndex | matchIndex
---|---|---|---|---|---
0 | 1 | 0 | 0 | [4, 1, 1] | [3, 0, 0]
1 | 1 | 0 | 0 | [] | []
2 | 1 | 0 | 0 | [] | []
AppendEntriesReply
Term: 1
Success: True

AppendEntriesReply
Term: 1
Success: True
while commitIndex > lastApplied, apply commands to state machine

Entry 3 is now replicated on a majority, so we can commit it
Once leader has applied an entry to state machine, it is safe to tell the client that the entry is committed.
Raft
After new leader election
<table>
<thead>
<tr>
<th>Node</th>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>[4, 4, 4]</td>
<td>[3, 3, 3]</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>[]</td>
<td>[]</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>[]</td>
<td>[]</td>
</tr>
</tbody>
</table>

*Partition!*

*Timeout*
AppendEntries
Term: 2
LeaderID: 1
PrevLogIndex: 3
PrevLogTerm: 1
LeaderCommit: 0

0
- currentTerm: 1
- votedFor: 0
- commitIndex: 3
- lastApplied: 3
- nextIndex: [4, 4, 4]
- matchIndex: [3, 3, 3]

1
- currentTerm: 2
- votedFor: 1
- commitIndex: 0
- lastApplied: 0
- nextIndex: [4, 4, 4]
- matchIndex: [0, 3, 0]
<table>
<thead>
<tr>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>[4, 4, 4]</td>
<td>[3, 3, 3]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>[]</td>
<td>[]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>[4, 4, 4]</td>
<td>[0, 3, 3]</td>
</tr>
</tbody>
</table>

**AppendEntries**
- **Term**: 2
- **LeaderID**: 1
- **PrevLogIndex**: 3
- **PrevLogTerm**: 1
- **LeaderCommit**: 3
<table>
<thead>
<tr>
<th></th>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>[4, 4, 4]</td>
<td>[3, 3, 3]</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>[0, 3, 3]</td>
<td>[0, 3, 3]</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>[]</td>
<td>[]</td>
</tr>
</tbody>
</table>
Committing entries in the new term...
Later, the network partition is fixed …
<table>
<thead>
<tr>
<th>Node</th>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>[4, 4, 4]</td>
<td>[3, 3, 3]</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>[4, 6, 6]</td>
<td>[0, 5, 5]</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>[]</td>
<td>[]</td>
</tr>
</tbody>
</table>

**AppendEntries**
- Term: 1
- LeaderID: 0
- PrevLogIndex: 3
- PrevLogTerm: 1
- LeaderCommit: 3
AppendEntriesReply
Term: 2
Success: false

Rejected request because local term is higher (2 > 1)
Old leader is dethroned!
AppendEntries
Term: 2
LeaderID: 1
PrevLogIndex: 3
PrevLogTerm: 1
LeaderCommit: 5

currentTerm 2
votedFor 1
commitIndex 5
lastApplied 5
nextIndex [4, 6, 6]
machineIndex [0, 5, 5]
0
- currentTerm: 2
- votedFor: -1
- commitIndex: 5
- lastApplied: 5
- nextIndex: []
- matchIndex: []

1
- currentTerm: 2
- votedFor: 1
- commitIndex: 5
- lastApplied: 5
- nextIndex: [4, 6, 6]
- matchIndex: [0, 5, 5]

2
- currentTerm: 2
- votedFor: 1
- commitIndex: 5
- lastApplied: 5
- nextIndex: []
- matchIndex: []

AppendEntriesReply
- Term: 2
- Success: true
AppendEntriesReply
Term: 2
Success: True
| 0 | currentTerm | 2 |
|   | votedFor    | -1 |
|   | commitIndex | 5 |
|   | lastApplied | 5 |
|   | nextIndex   | [] |
|   | matchIndex  | [] |
|   | 1 1 1 2 2 2 |

| 1 | currentTerm | 2 |
|   | votedFor    | 1 |
|   | commitIndex | 5 |
|   | lastApplied | 5 |
|   | nextIndex   | [6, 6, 6] |
|   | matchIndex  | [5, 5, 5] |
|   | 1 1 1 2 2 |

| 2 | currentTerm | 2 |
|   | votedFor    | 1 |
|   | commitIndex | 5 |
|   | lastApplied | 5 |
|   | nextIndex   | [] |
|   | matchIndex  | [] |
|   | 1 1 1 2 2 |

Everyone is on the same page again
When log entries don’t match...
When log entries don’t match...

- The leader will find the latest log entry in the follower where the two logs agree
- At the follower:
  - Everything after that entry will be deleted
  - The leader’s log starting from that entry will be replicated on the follower
prevLogIndex = 5
S1 log[5] = 4
S2 log[5] = 2
Mismatch!
<table>
<thead>
<tr>
<th></th>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>[]</td>
<td>[]</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>[6, 6, 6]</td>
<td>[5, 5, 0]</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>-1</td>
<td>3</td>
<td>3</td>
<td>[]</td>
<td>[]</td>
</tr>
</tbody>
</table>

AppendEntriesReply
Term: 5
Success: False
prevLogIndex = 4
S1 log[4] = 3
S2 log[4] = 2
Mismatch!
0
- currentTerm: 5
- votedFor: 1
- commitIndex: 5
- lastApplied: 5
- nextIndex: []
- matchIndex: []

1
- currentTerm: 5
- votedFor: 1
- commitIndex: 5
- lastApplied: 5
- nextIndex: [6, 6, 5]
- matchIndex: [5, 5, 0]

2
- currentTerm: 5
- votedFor: -1
- commitIndex: 3
- lastApplied: 3
- nextIndex: []
- matchIndex: []

AppendEntriesReply
- Term: 5
- Success: False
prevLogIndex = 3
S1 log[3] = 1
S2 log[3] = 1
Match!
currentTerm 5
votedFor 1
commitIndex 5
lastApplied 5
nextIndex []
matchIndex []

AppendEntriesReply
Term: 5
Success: True
<table>
<thead>
<tr>
<th>Node</th>
<th>currentTerm</th>
<th>votedFor</th>
<th>commitIndex</th>
<th>lastApplied</th>
<th>nextIndex</th>
<th>matchIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>[6, 6, 6]</td>
<td>[5, 5, 5]</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>-1</td>
<td>5</td>
<td>5</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>

Everyone is on the same page again
Optimization to reduce number of messages?
Key Idea

- Reduce the number of rejected AppendEntries RPCs
- One RPC per conflicting term, rather than one RPC per conflicting entry

Detailed Algorithm:
- When rejecting an AppendEntries request, the follower can include the term of the conflicting entry and the first index it stores for that term.
- With this information, the leader can decrement nextIndex to bypass all of the conflicting entries in that term.
- See page 7-8 in Raft (extended version)
AppendEntries
Term: 5
LeaderID: 1
PrevLogIndex: 5
PrevLogTerm: 4
LeaderCommit: 5
Specify the term of the conflicting term and the first index of this term.
Leader sends its log entries that are different from the follower’s starting the specified conflicting term.
Key Idea:
Decrement nextIndex one term at a time
Conditions for committing an entry

1. The entry exists on a majority AND it is written in the current term
2. The entry precedes another entry that is committed
Caveat for committing old entries

Can’t assume an old entry has been committed *even if* it exists on a majority

S1 is the leader

S1.log[2] is only partially replicated...

S1 is the leader
Caveat for committing old entries

Can’t assume an old entry has been committed even if it exists on a majority

```
S1
1 2
```
```
S2
1 2
```
```
S3
1
```
```
S4
1
```
```
S5
1 3
```
S1 crashes,
S5 becomes leader
Caveat for committing old entries

Can’t assume an old entry has been committed even if it exists on a majority

S5 crashes,
S1 becomes leader

S1.log[2] is now replicated to a majority
Caveat for committing old entries

Can’t assume an old entry has been committed *even if* it exists on a majority

S1 crashes, S5 becomes leader

S5 replicates S5.log[2] to all other nodes...

S1 crashes, S5 becomes leader
Caveat for committing old entries

Can’t assume an old entry has been committed *even if* it exists on a majority.

Entry 2 was overwritten even though it was replicated on a majority!

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Cannot assume entry 2 was committed.
Caveat for committing old entries

Can’t assume an old entry has been committed even if it exists on a majority

Commit old entries indirectly

Entry 2 is committed once entry 3 is committed

S1 commits entry 3

S1
1 2
1 2
1 2 4
1 2 4

S2
1 2
1 2
1 2
1 2 4

S3
1
1
1 2
1 2 4

S4
1
1
1
1

S5
1
1 3
1 3
1 3
Exercise...
Exercise...

Rules for deciding which log is more up-to-date:

- Compare **index** and **term** of last entries in the logs
- If the terms are different: log with **later term is more up-to-date**
- If the terms are the same: **longer log is more up-to-date**
Q1: Is this a possible configuration?
Trace the steps...

```
1
S0  1
S1  1
S2  1
S3  1
S4  1
```

```
1  2  3  4  5
S0  1  1  2  3
S1  1  1  2  3
S2  1  1  2  3
S3  1  1  
S4  1  1  1  1  1
```
Trace the steps...

Trace the steps of S0, S1, S2, S3, and S4.
Trace the steps...

S0: 1 1
S1: 1 1
S2: 1 1
S3: 1 1
S4: 1 1 1 1 1

S0: 1 1 2 3
S1: 1 1 2 3
S2: 1 1 2 3
S3: 1 1
S4: 1 1 1 1 1 1
Trace the steps...

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>1</td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
</tr>
<tr>
<td>S4</td>
<td>1</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S0</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Trace the steps...

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S4</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Trace the steps...

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S4</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Trace the steps...

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The steps are traced through the states S0 to S4, with each state represented by a box. The numbers 1, 2, 3, 4, 5 represent different states or conditions within the process.
Trace the steps...
Q2: Is this a possible configuration?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

S3 cannot become leader in term 4
(Who’s going to vote for him?)

NO!
Q3: Is this a possible configuration?

Yes

What happened to terms 2 and 3?
1. Split vote: no one became leader
2. Partitions: no one became leader
3. Simply no requests in these terms
Q4: Is this a possible configuration?

NO!

Let's try tracing the steps...
Q4: Is this a possible configuration?
Q4: Is this a possible configuration?

No one becomes leader in term 2...
Q4: Is this a possible configuration?
Q4: Is this a possible configuration?

<table>
<thead>
<tr>
<th></th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

The configuration on the right is possible because it meets the constraints. The configuration on the left does not because it violates the constraints (e.g., S0 in column 4).
Q4: Is this a possible configuration?

S0 previously voted for S2 in term 3
S0 can only vote for S1 for term 4!
Q4: Is this a possible configuration?

The two entries in term 3 are in different positions.

S1 and S2 could not have written these entries without being leaders.

But they can’t both be leaders in the same term!
Q5: Is entry 2 (term 2) guaranteed to be committed?

Yes!
Entry 2 is on a majority of nodes
No one else has a more *up-to-date* log
Q6: Is entry 3 (term 2) guaranteed to be committed?

S3 could become leader if S0 crashes

Entry 3 is an entry from an old term
(See Figure 8 in Raft paper)

NO!
Q7: Is entry 3 (term 2) guaranteed to be committed?

NO!

S3 could still become leader if S0 crashes (votes from S2, S3 and S4)
Q8: Is entry 3 (term 2) guaranteed to be committed?

Yes!

Entry 4 is guaranteed to be committed because no one else has a more up-to-date log.

All entries before entry 4 are safe.