

Raft

Mar 26th, 2021

Raft

System for enforcing **strong consistency** (linearizability)

Similar to Paxos and Viewstamped Replication, but much **simpler**

Clear boundary between *leader election* and *consensus*

Leader log is ground truth; log entries only flow in one direction (from leader to followers)

Assignment 3 hints

You will implement the *leader election* portion of Raft in assignment 3

You will implement the *log replication* portion of Raft in assignment 4

Use `time.Timer` and `select` statements to implement timeout

- Need to time out on heartbeats → Start election
- Need to time out on waiting for majority of votes

Raft logs are 1-indexed; add a dummy entry in the first slot to enforce this

When voting for yourself, you can skip the RPC

Importance of readability

A luxury for small projects, but a necessity for large and complex projects

HW4 will build on top of your solution for HW3

HW3 only accounts for about 20% of the work

Some tips:

Duplicate code is *really* bad; avoid at all costs

If a function is more than 30 lines, it is too long → split!

Avoid nested if-else's; use returns and continues where possible

Raft

Leader election

0	currentTerm	0
	votedFor	-1
	commitIndex	0
	lastApplied	0
	nextIndex	[]
	matchIndex	[]

(log entries here)

Logs are 1-indexed

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

(Only 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

0	currentTerm	0
	votedFor	-1
<empty>		

currentTerm latest term server has seen

votedFor candidate ID that received vote in current term,
or -1 if none

State required for election

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

0	currentTerm	0
	votedFor	-1
<empty>		

Timeout

1	currentTerm	0
	votedFor	-1
<empty>		

2	currentTerm	0
	votedFor	-1
<empty>		

0	currentTerm	1
	votedFor	0
<empty>		

RequestVote
Term: 1
CandidateID: 0
LastLogIndex: -1
LastLogTerm: -1

1	currentTerm	0
	votedFor	-1
<empty>		

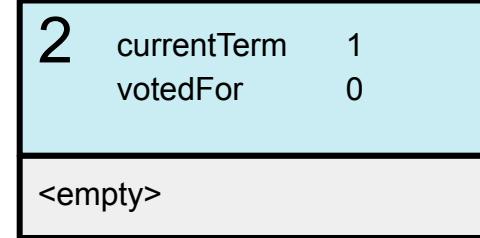
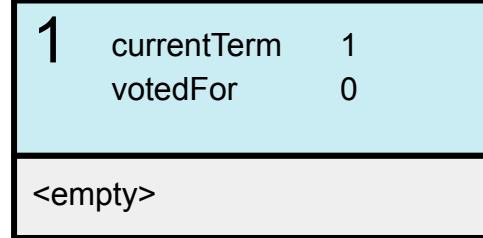
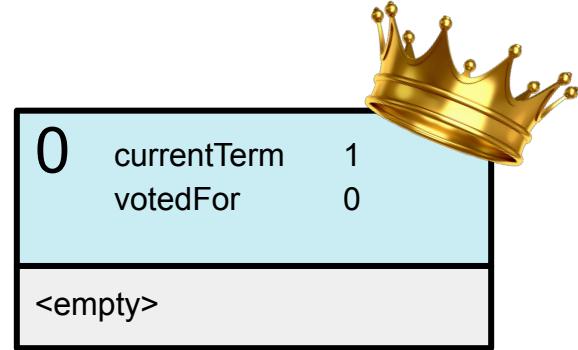
2	currentTerm	0
	votedFor	-1
<empty>		

0	currentTerm	1
votedFor	0	
<empty>		

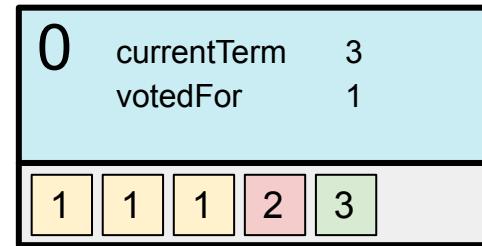
RequestVoteReply
Term: 1
VoteGranted: true

1	currentTerm	1
votedFor	0	
<empty>		

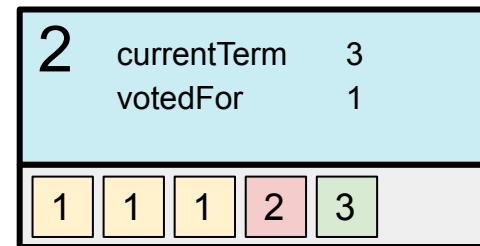
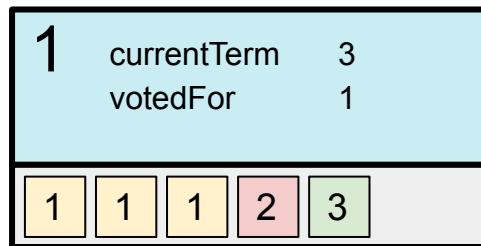
2	currentTerm	1
votedFor	0	
<empty>		

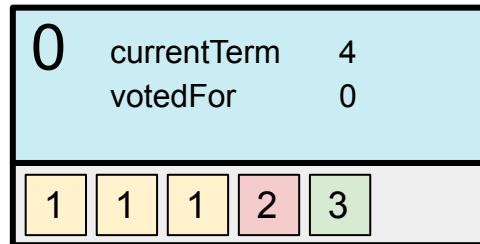


Suppose there are existing log entries...

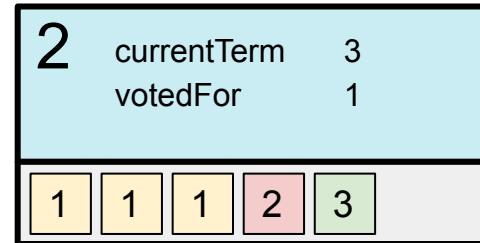
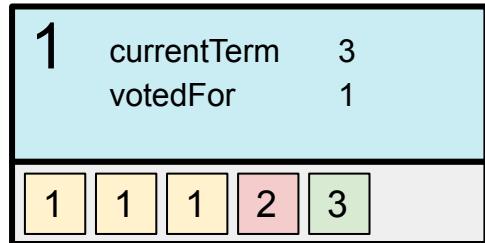


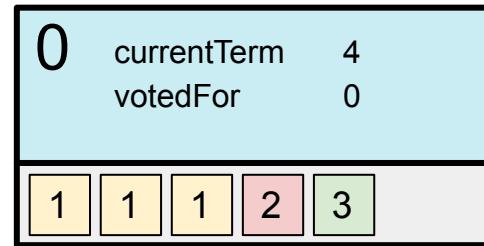
Timeout



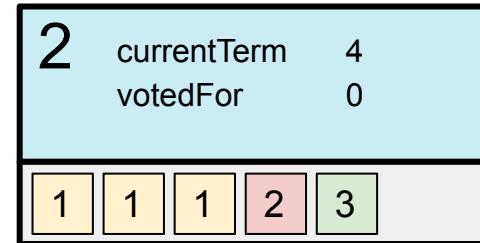
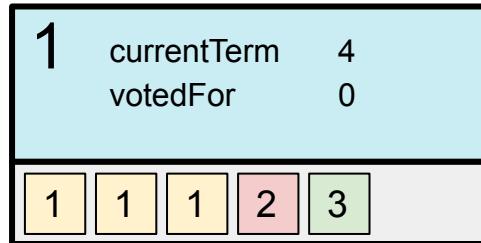


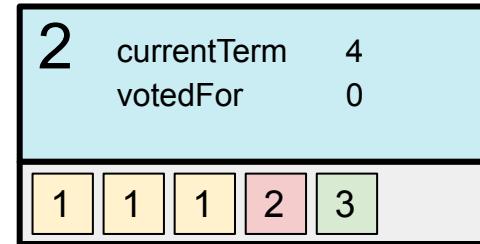
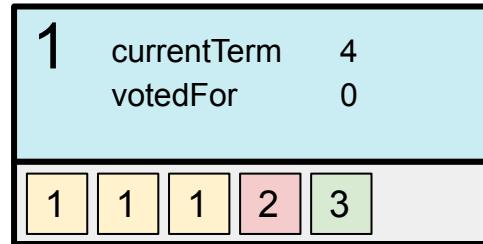
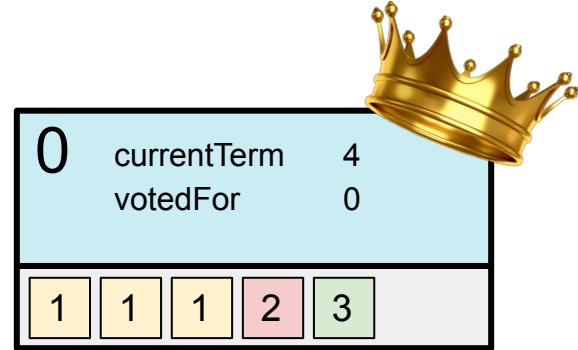
RequestVote
Term: 4
CandidateID: 0
LastLogIndex: 5
LastLogTerm: 3





RequestVoteReply
Term: 4
VoteGranted: True

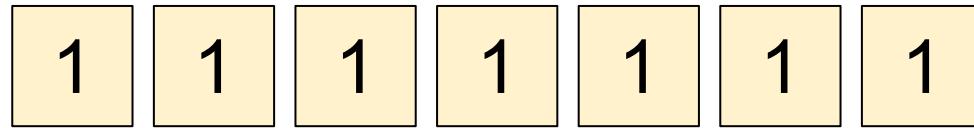
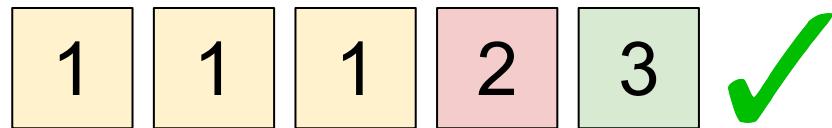




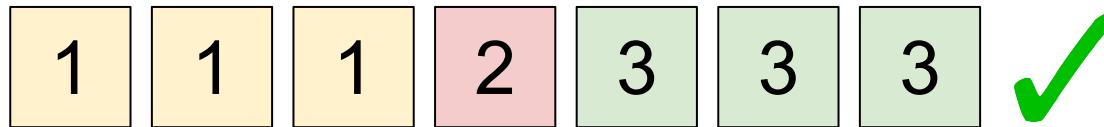
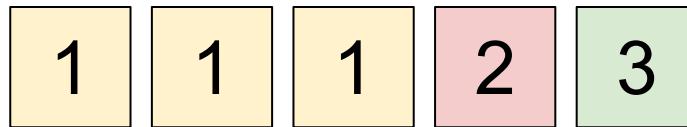
Conditions for granting vote

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?

	1	2	3	4	5
S0	1	1	1	2	3
S1	1	1	1	2	3
S2	1	1	1	2	3
S3	1	1	1		
S4	1	1	1	1	1

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!

	1	2	3	4	5
S0	1	1	1	2	3
S1	1	1	1	2	3
S2	1	1	1	2	3
S3	1	1	1		
S4	1	1	1	1	1

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

	1	2	3	4	5
S0	1	1	1	2	3
S1	1	1	1	2	3
 S2	1	1	1	2	3
S3	1	1	1	2	3
S4	1	1	1	2	3

Raft

Normal operation

0	currentTerm	0
	votedFor	-1
	commitIndex	0
	lastApplied	0
	nextIndex	[]
	matchIndex	[]

<empty>

Logs are 1-indexed

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

(Only 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

0

currentTerm 0
votedFor -1
commitIndex 0
lastApplied 0
nextIndex []
matchIndex []

<empty>

1

currentTerm 0
votedFor -1
commitIndex 0
lastApplied 0
nextIndex []
matchIndex []

<empty>

2

currentTerm 0
votedFor -1
commitIndex 0
lastApplied 0
nextIndex []
matchIndex []

<empty>



0

currentTerm	1
votedFor	0
commitIndex	0
lastApplied	0
nextIndex	[1, 1, 1]
matchIndex	[0, 0, 0]

<empty>

1

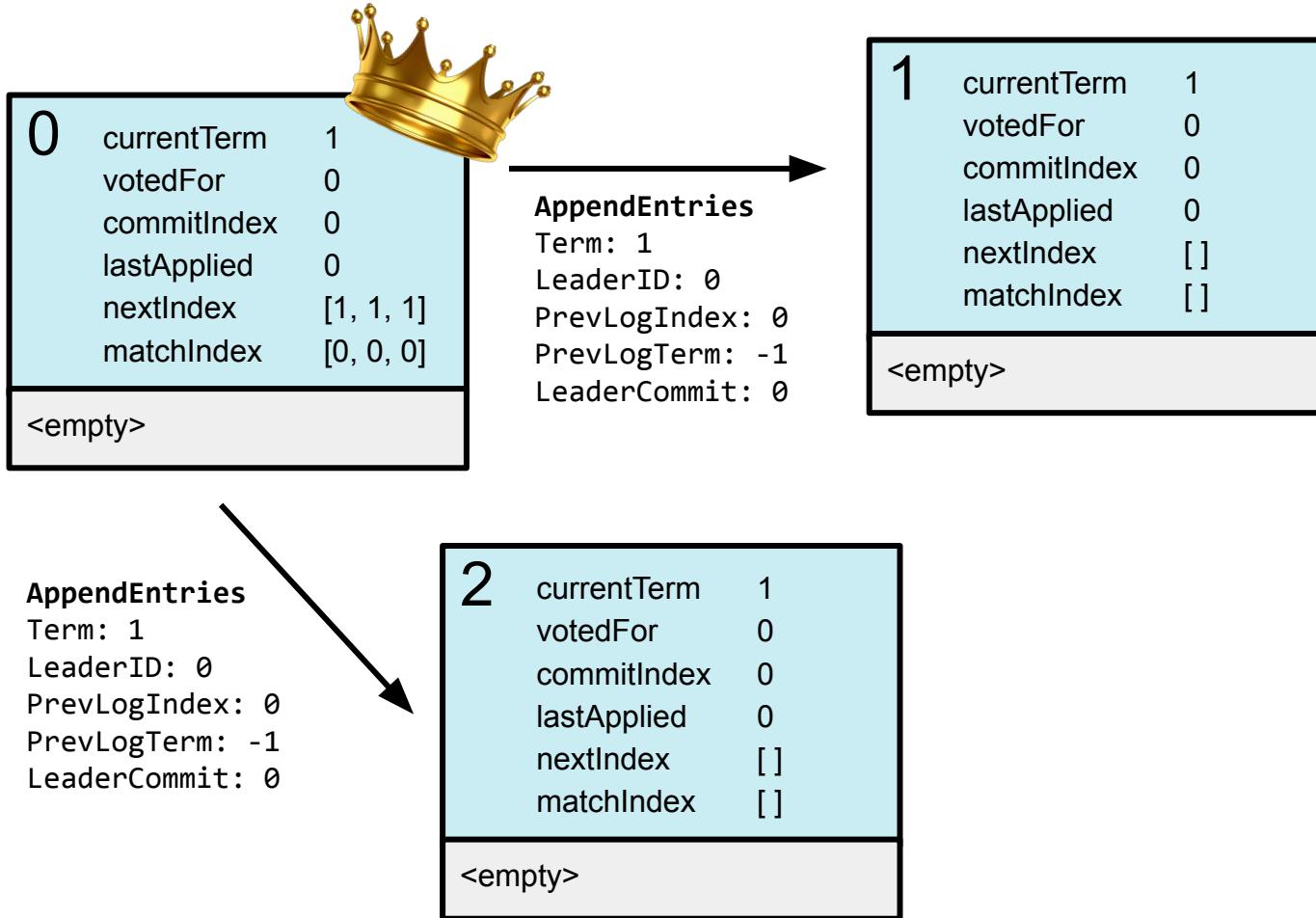
currentTerm	1
votedFor	0
commitIndex	0
lastApplied	0
nextIndex	[]
matchIndex	[]

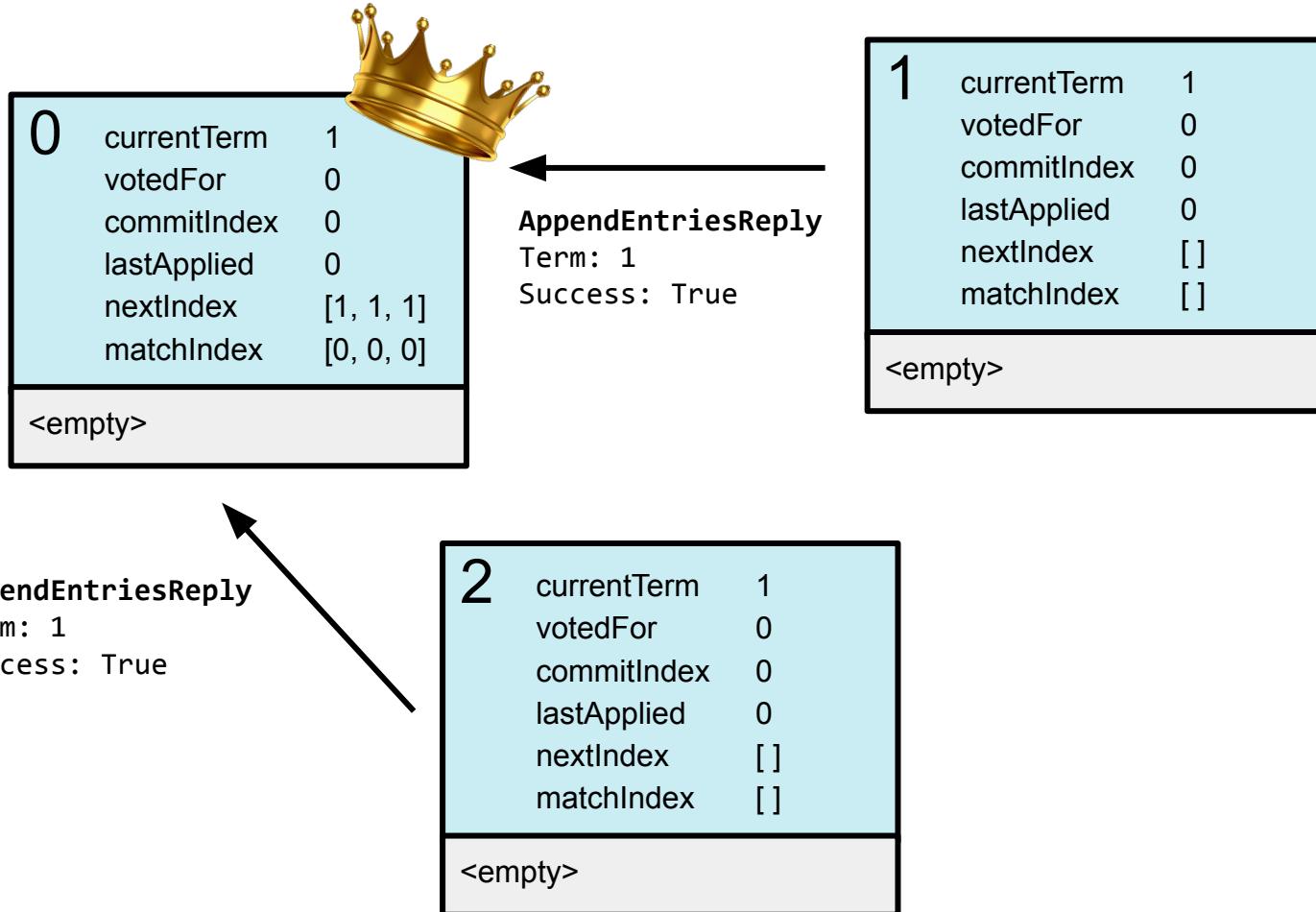
<empty>

2

currentTerm	1
votedFor	0
commitIndex	0
lastApplied	0
nextIndex	[]
matchIndex	[]

<empty>







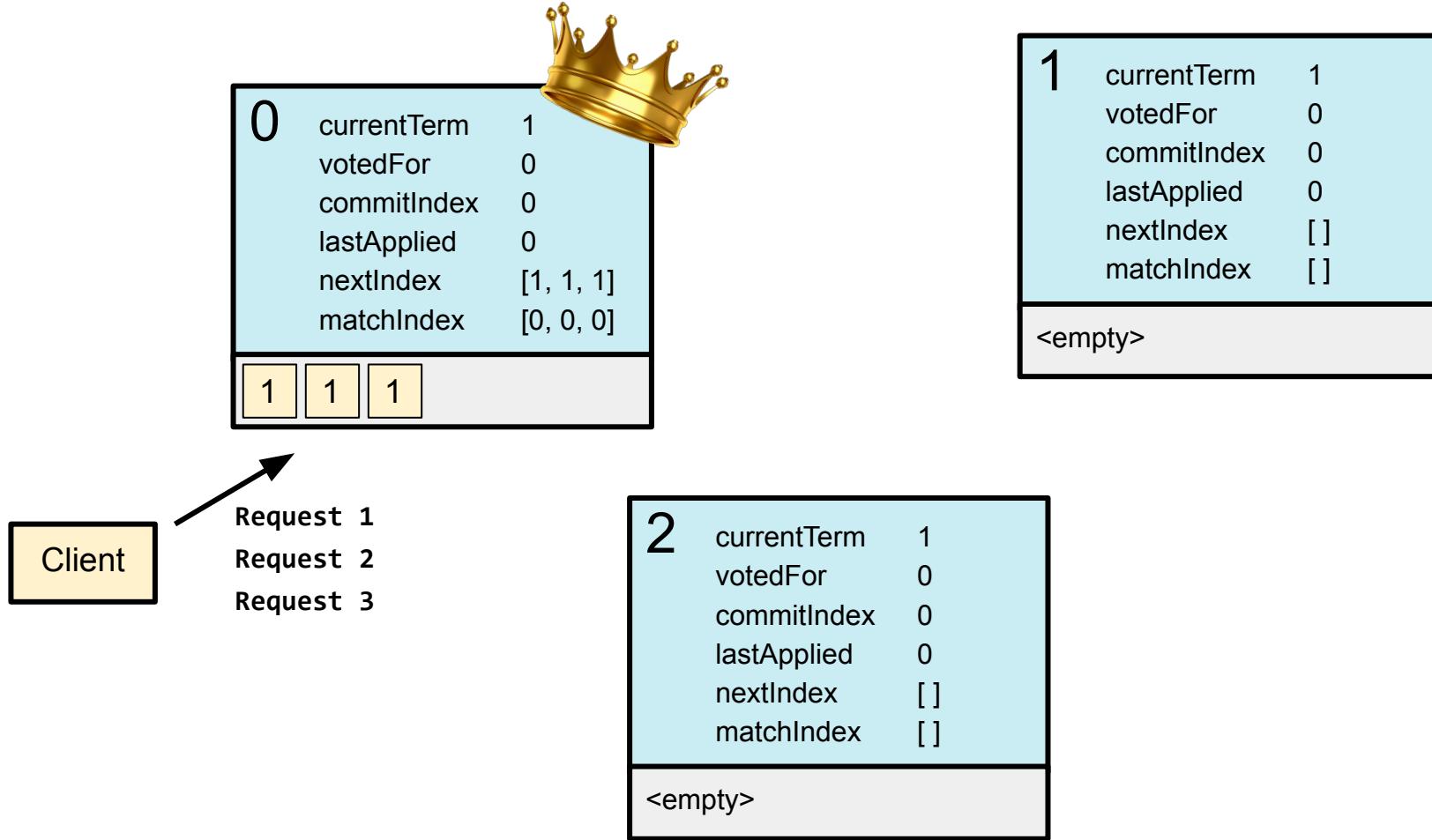
0	currentTerm	1
	votedFor	0
	commitIndex	0
	lastApplied	0
	nextIndex	[1, 1, 1]
	matchIndex	[0, 0, 0]
<empty>		

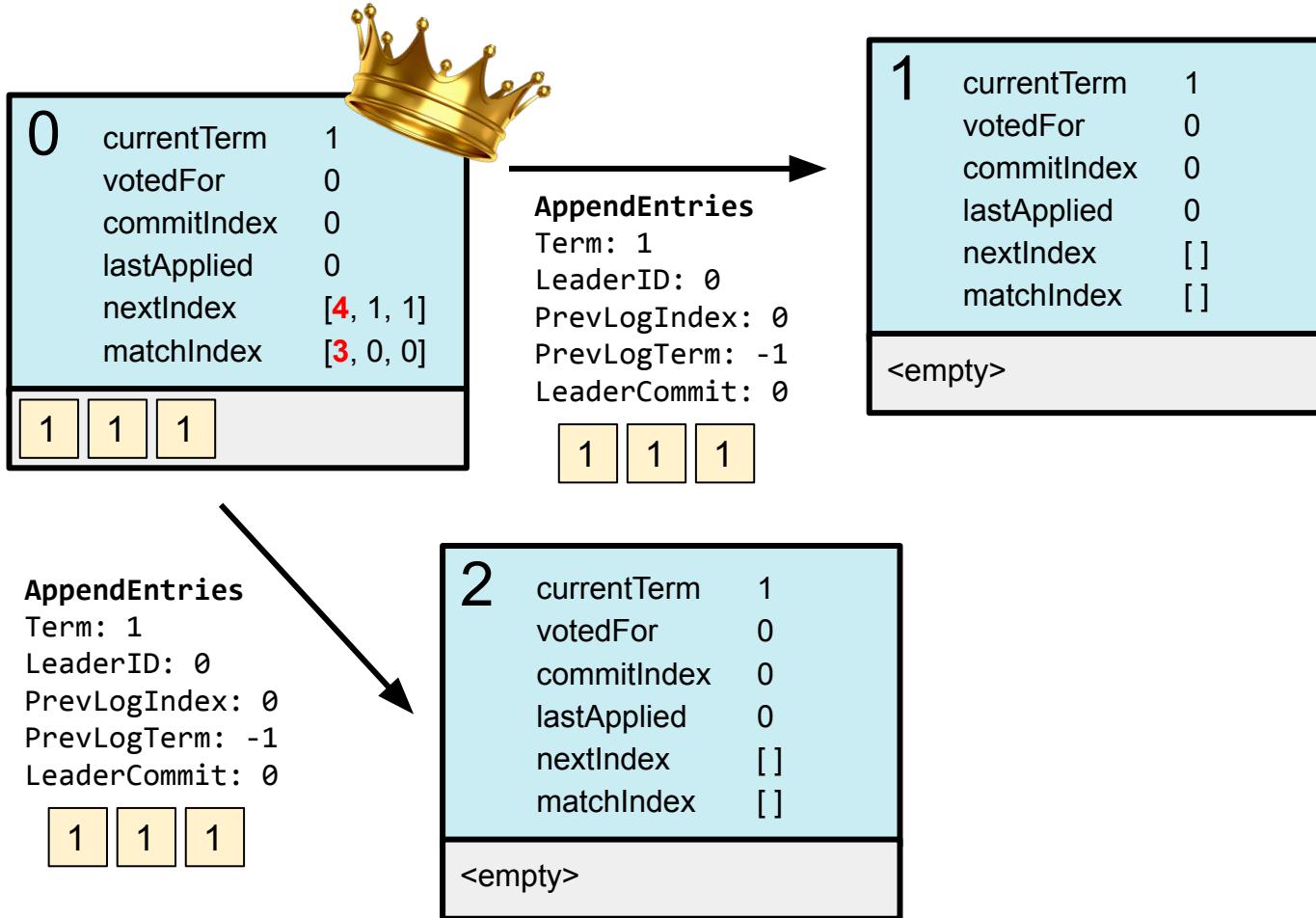
1	currentTerm	1
	votedFor	0
	commitIndex	0
	lastApplied	0
	nextIndex	[]
	matchIndex	[]
<empty>		

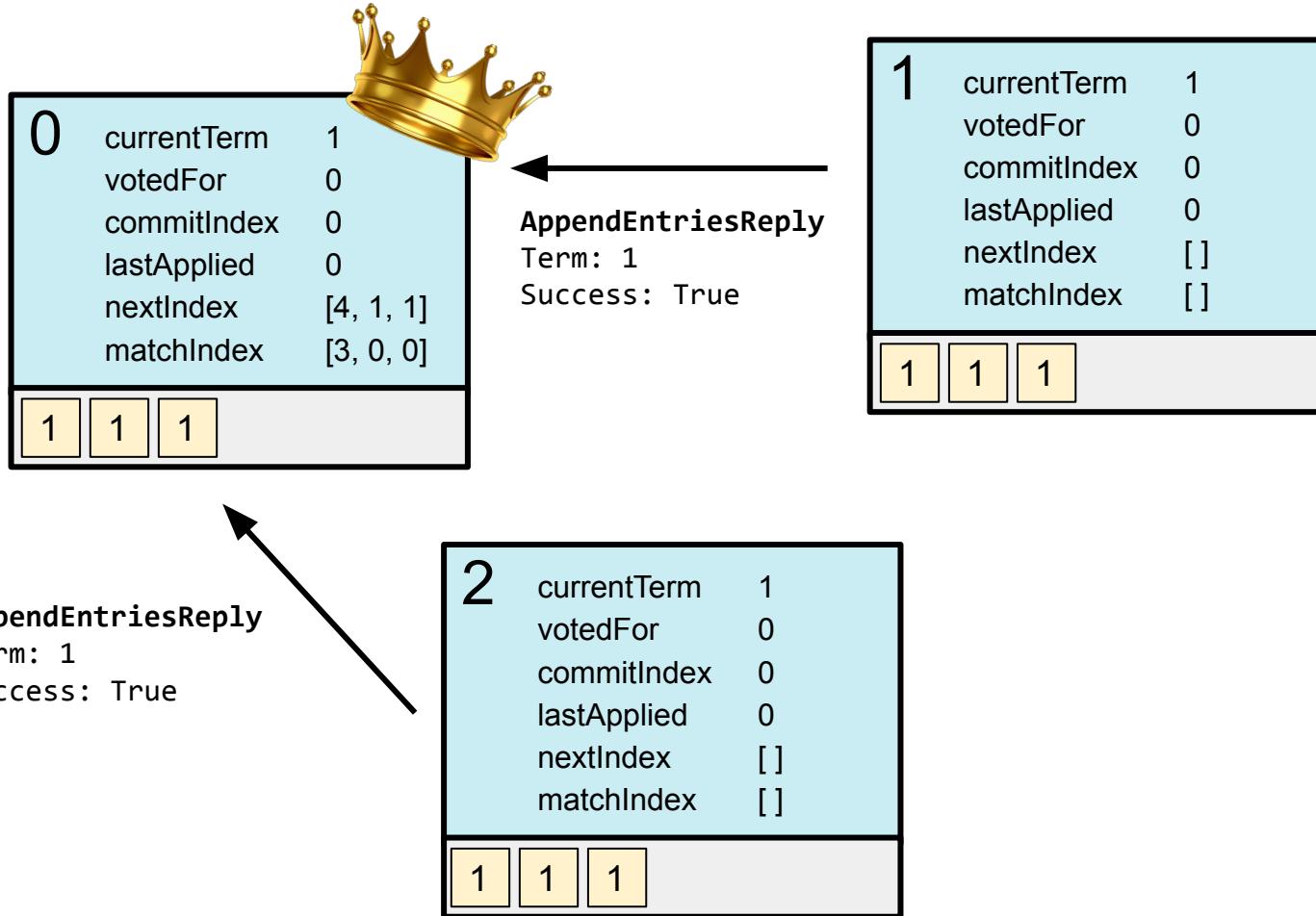
Client

Request 1

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





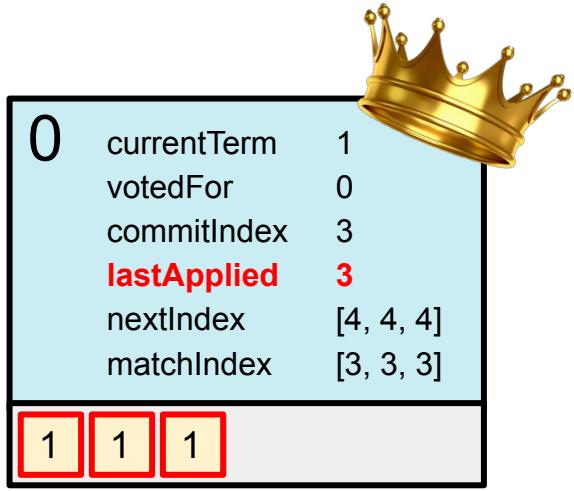




0	currentTerm	1
	votedFor	0
	commitIndex	3
	lastApplied	0
	nextIndex	[4, 4, 4]
	matchIndex	[3, 3, 3]
	1	1
	1	

Entry 3 is now replicated on a majority, so we can commit it

while `commitIndex > lastApplied`, apply commands to state machine

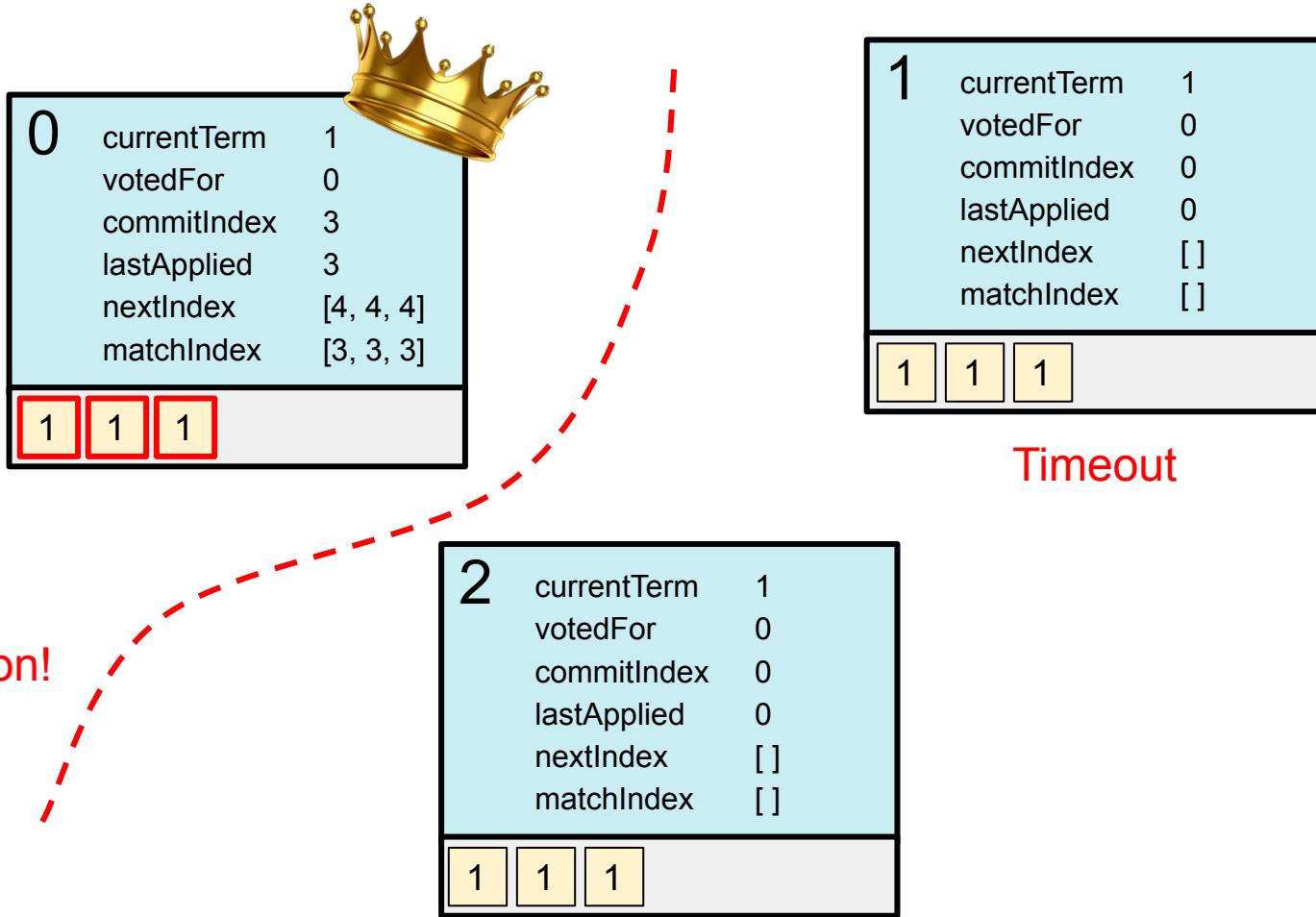


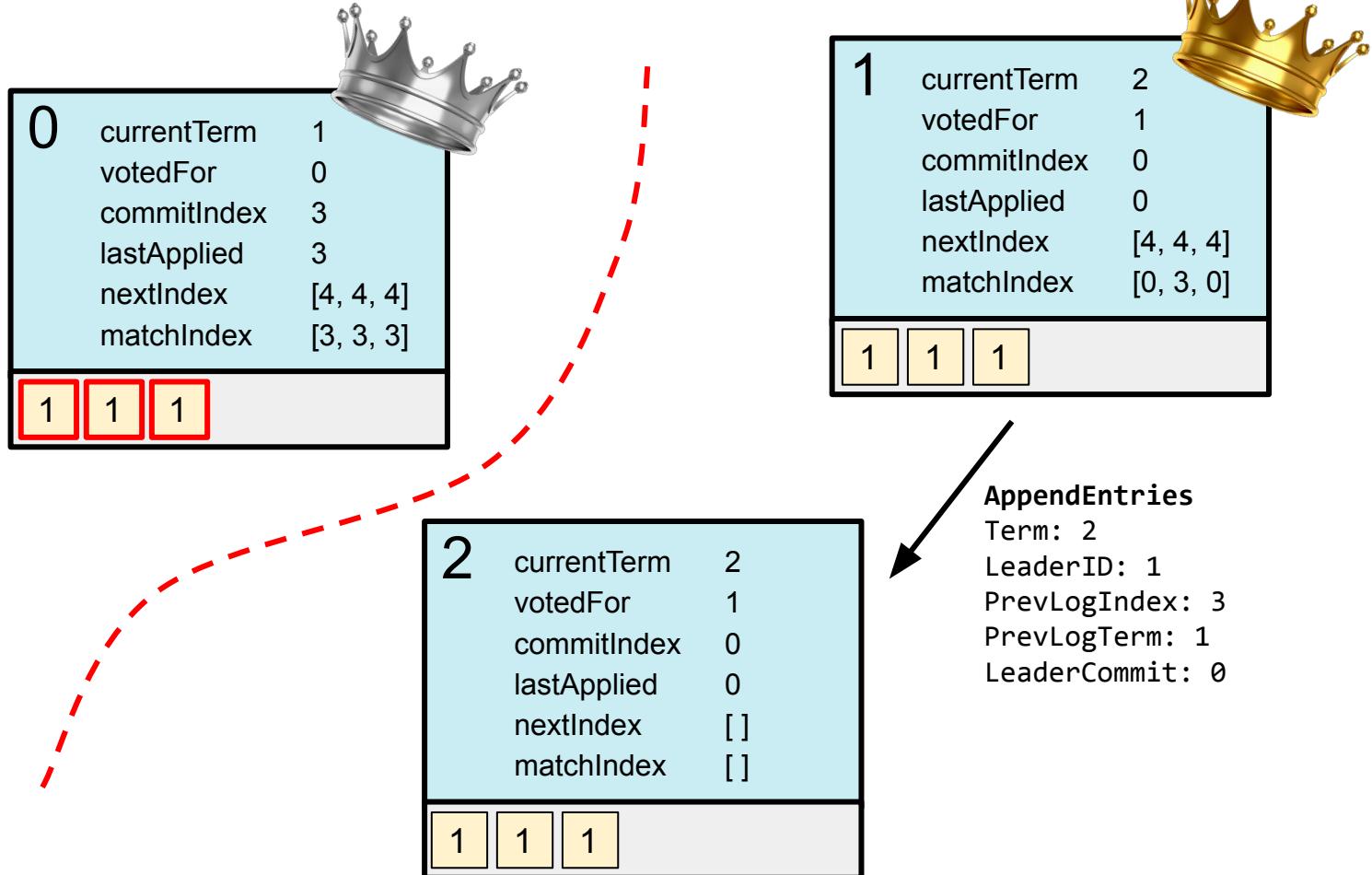
Response 1 2 3

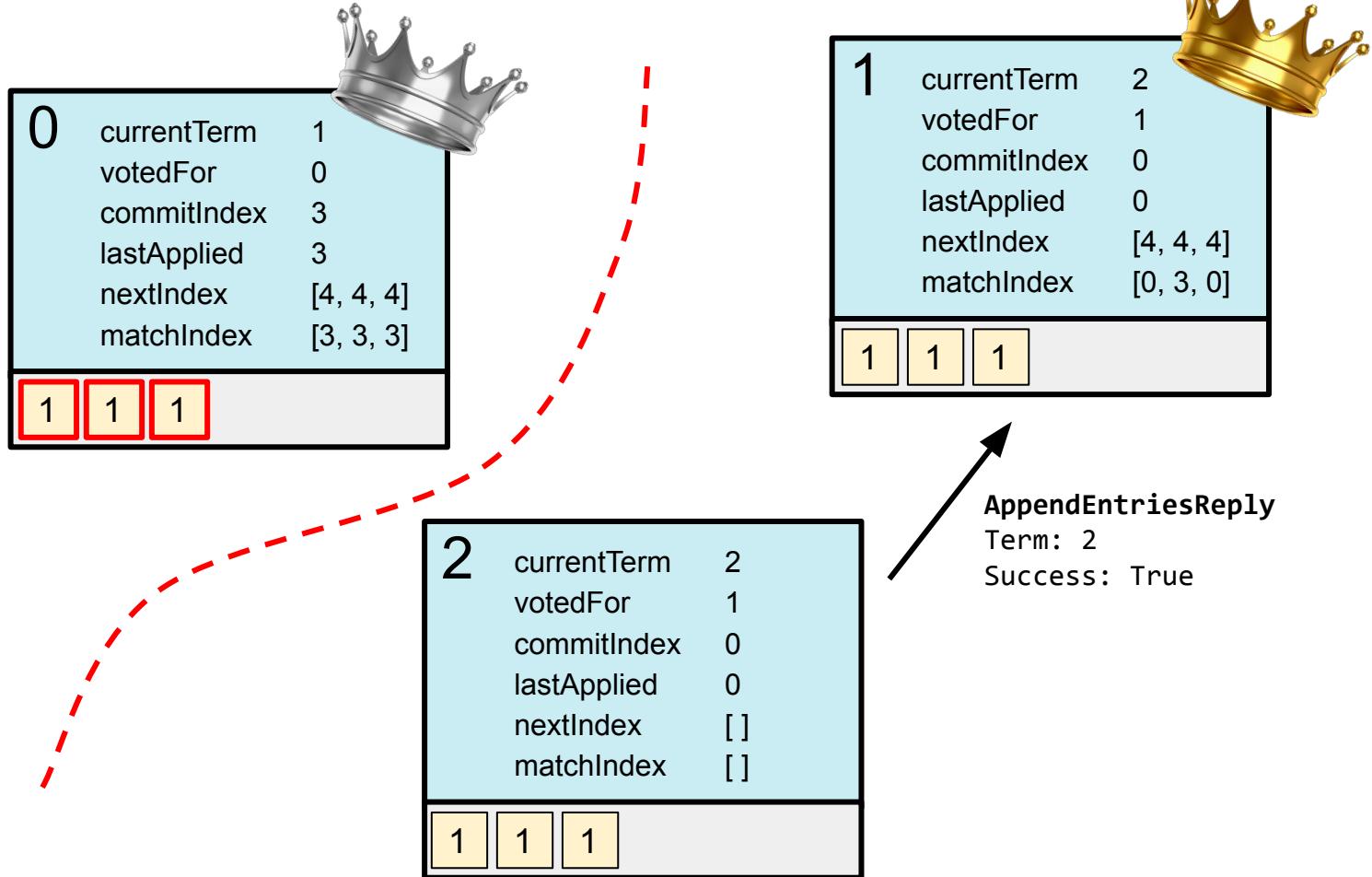
Once leader has applied
an entry to state machine,
it is safe to tell the client
that the entry is committed

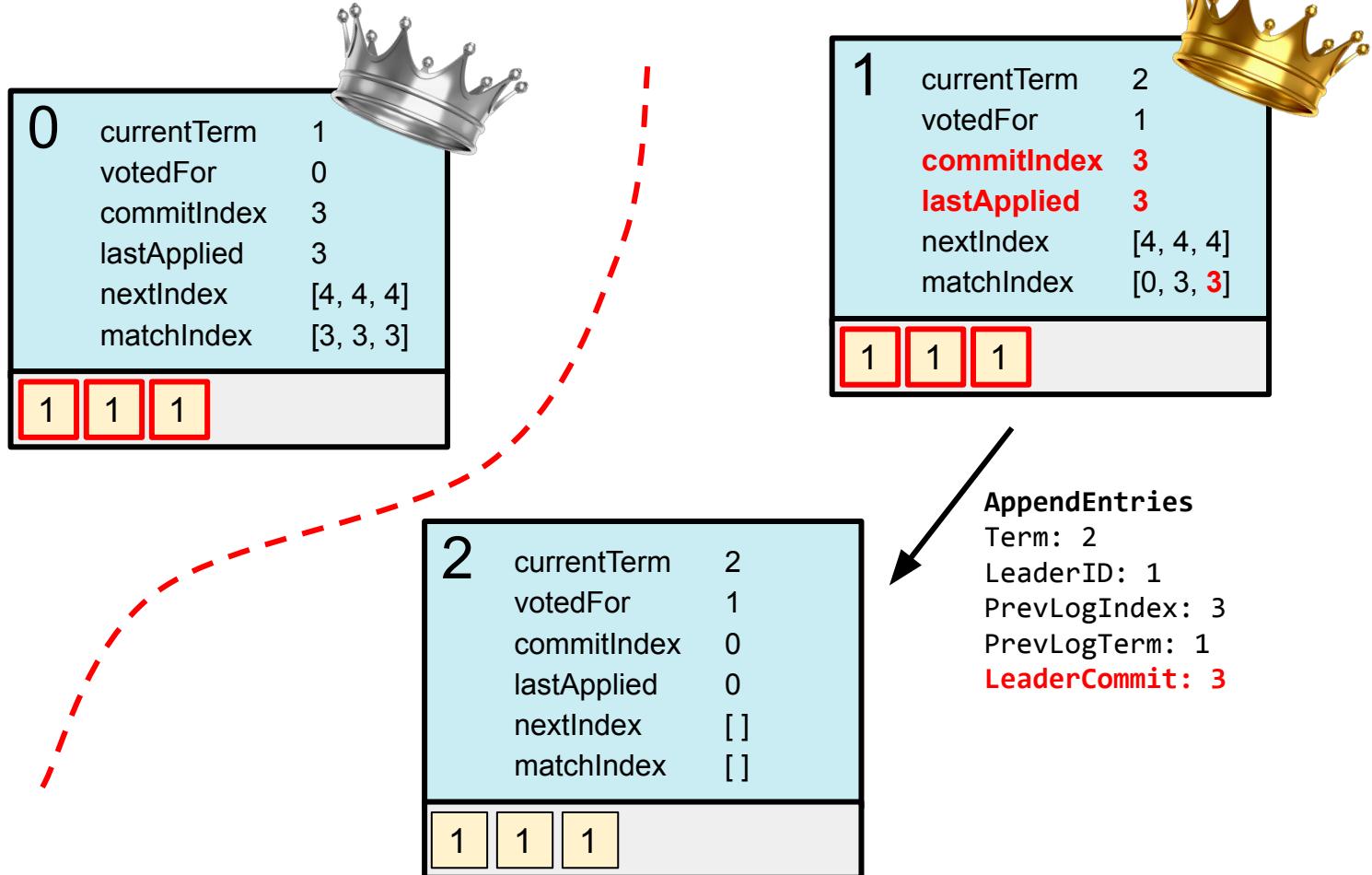
Raft

After new leader election







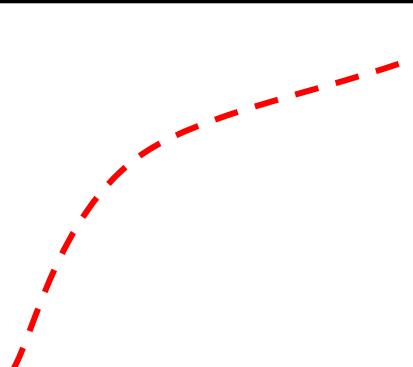




0	currentTerm	1
	votedFor	0
	commitIndex	3
	lastApplied	3
	nextIndex	[4, 4, 4]
	matchIndex	[3, 3, 3]
		
1 1 1		



1	currentTerm	2
	votedFor	1
	commitIndex	3
	lastApplied	3
	nextIndex	[4, 4, 4]
	matchIndex	[0, 3, 3]
		
1 1 1		



2	currentTerm	2
	votedFor	1
	commitIndex	3
	lastApplied	3
	nextIndex	[]
	matchIndex	[]
		
1 1 1		



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



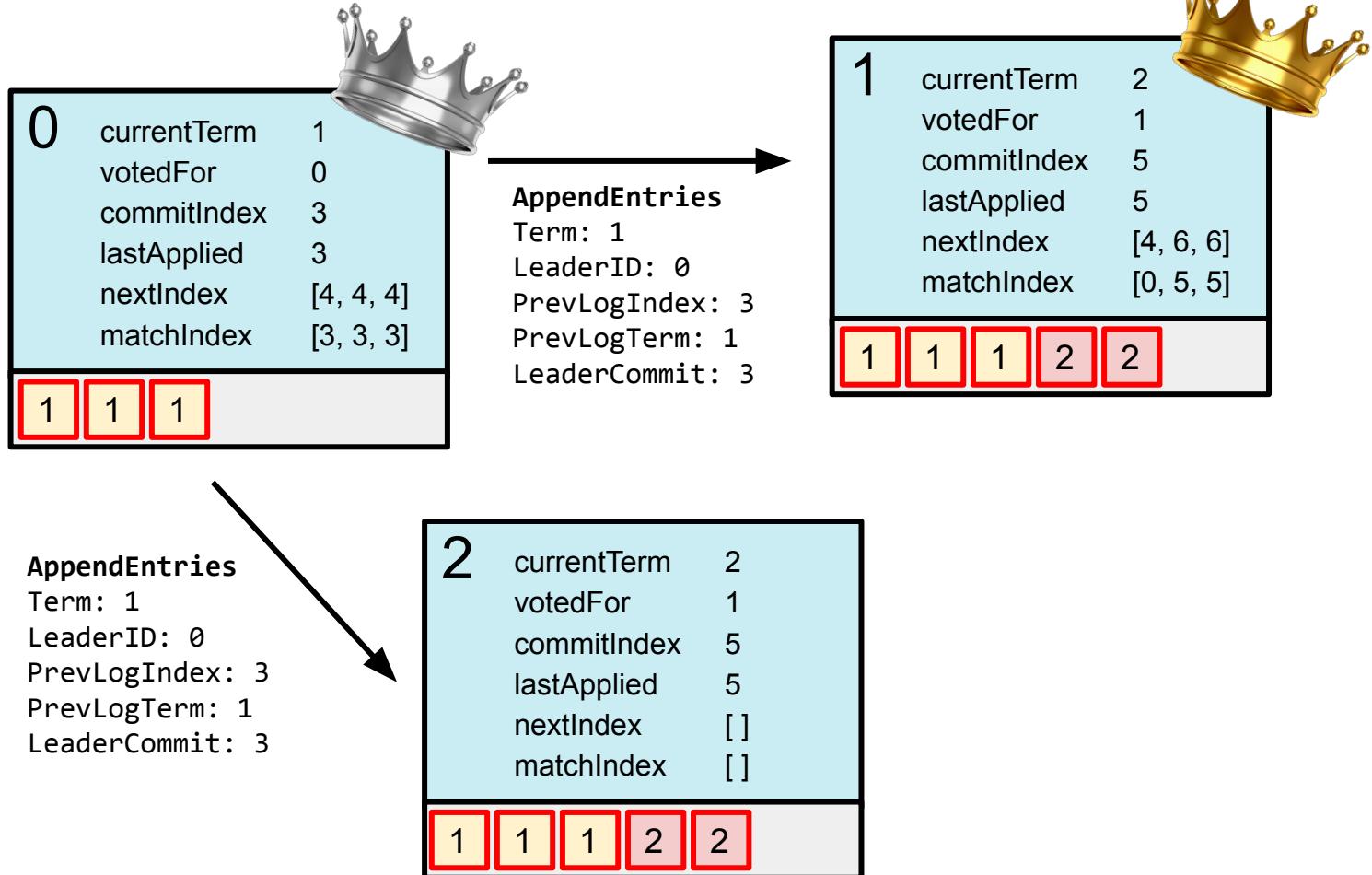
1	currentTerm	2
	votedFor	1
	commitIndex	5
	lastApplied	5
	nextIndex	[4, 6, 6]
	matchIndex	[0, 5, 5]
		

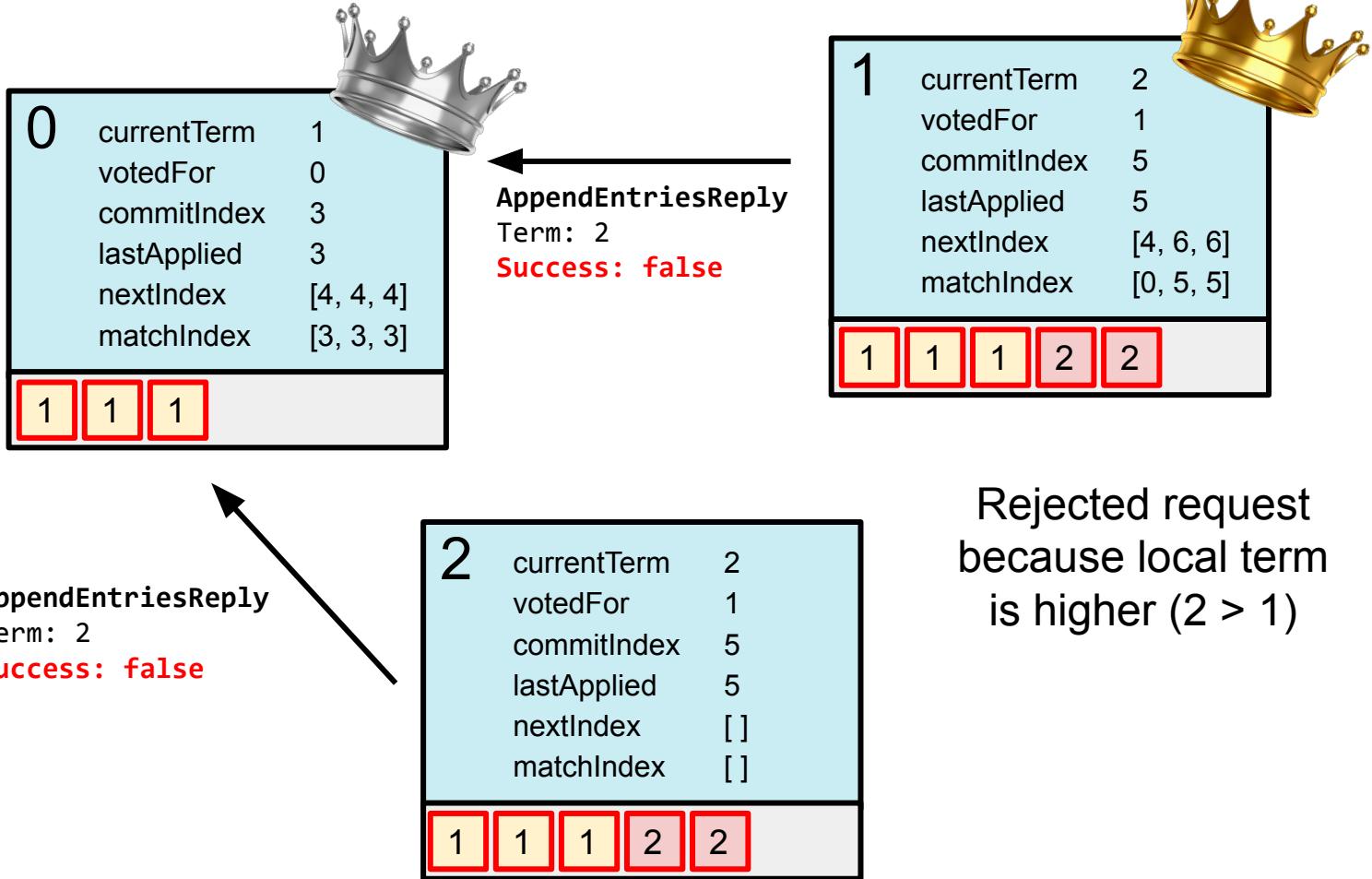


2	currentTerm	2
	votedFor	1
	commitIndex	5
	lastApplied	5
	nextIndex	[]
	matchIndex	[]
		

Committing entries
in the new term...

Let's fix the partition...







0	currentTerm	2			
	votedFor	-1			
	commitIndex	3			
	lastApplied	3			
	nextIndex	[]			
	matchIndex	[]			
<table border="1"><tr><td>1</td><td>1</td><td>1</td></tr></table>			1	1	1
1	1	1			

1	currentTerm	2					
	votedFor	1					
	commitIndex	5					
	lastApplied	5					
	nextIndex	[4, 6, 6]					
	matchIndex	[0, 5, 5]					
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>2</td><td>2</td></tr></table>			1	1	1	2	2
1	1	1	2	2			

Old leader is dethroned!

2	currentTerm	2					
	votedFor	1					
	commitIndex	5					
	lastApplied	5					
	nextIndex	[]					
	matchIndex	[]					
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>2</td><td>2</td></tr></table>			1	1	1	2	2
1	1	1	2	2			



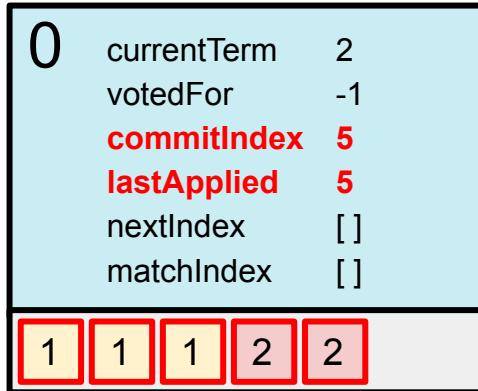
0	currentTerm	2			
	votedFor	-1			
	commitIndex	3			
	lastApplied	3			
	nextIndex	[]			
	matchIndex	[]			
<table border="1"><tr><td>1</td><td>1</td><td>1</td></tr></table>			1	1	1
1	1	1			

AppendEntries
Term: 2
LeaderID: 1
PrevLogIndex: 3
PrevLogTerm: 1
LeaderCommit: 5

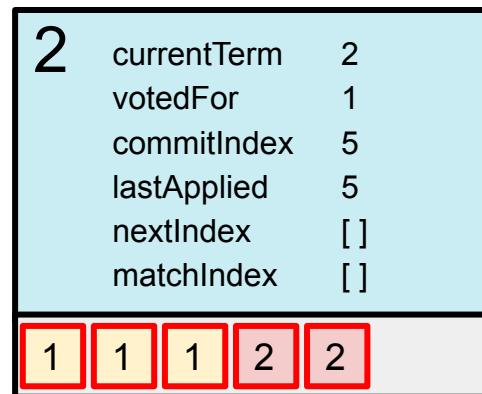
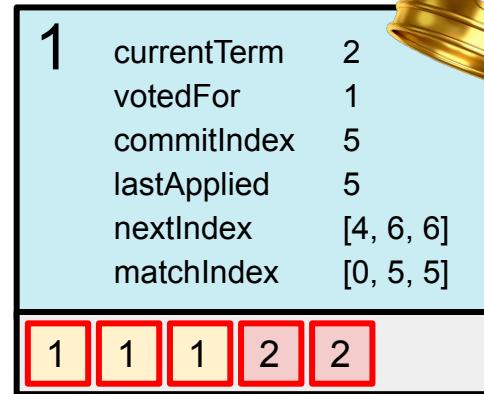
2 2

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

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



AppendEntriesReply
Term: 2
Success: true





0	currentTerm	2					
	votedFor	-1					
	commitIndex	5					
	lastApplied	5					
	nextIndex	[]					
	matchIndex	[]					
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>2</td><td>2</td></tr></table>			1	1	1	2	2
1	1	1	2	2			

1	currentTerm	2					
	votedFor	1					
	commitIndex	5					
	lastApplied	5					
	nextIndex	[6, 6, 6]					
	matchIndex	[5, 5, 5]					
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>2</td><td>2</td></tr></table>			1	1	1	2	2
1	1	1	2	2			

Everyone is on the same page again

2	currentTerm	2					
	votedFor	1					
	commitIndex	5					
	lastApplied	5					
	nextIndex	[]					
	matchIndex	[]					
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>2</td><td>2</td></tr></table>			1	1	1	2	2
1	1	1	2	2			

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 up to that point will be replicated on the follower

0	currentTerm	5			
	votedFor	1			
	commitIndex	5			
	lastApplied	5			
	nextIndex	[]			
	matchIndex	[]			
	1	1	1	3	4

1	currentTerm	5			
	votedFor	1			
	commitIndex	5			
	lastApplied	5			
	nextIndex	[6, 6, 6]			
	matchIndex	[5, 5, 0]			
	1	1	1	3	4

2	currentTerm	3				
	votedFor	2				
	commitIndex	3				
	lastApplied	3				
	nextIndex	[]				
	matchIndex	[]				
	1	1	1	2	2	2



0	currentTerm	5
	votedFor	1
	commitIndex	5
	lastApplied	5
	nextIndex	[]
	matchIndex	[]

prevLogIndex = 5
S1 log[5] = 4
S2 log[5] = 2

Mismatch!



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



2	currentTerm	3
	votedFor	2
	commitIndex	3
	lastApplied	3
	nextIndex	[]
	matchIndex	[]

AppendEntries
 Term: 5
 LeaderID: 1
 PrevLogIndex: 5
 PrevLogTerm: 4
 LeaderCommit: 5



0	currentTerm	5					
	votedFor	1					
	commitIndex	5					
	lastApplied	5					
	nextIndex	[]					
	matchIndex	[]					
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>3</td><td>4</td></tr></table>			1	1	1	3	4
1	1	1	3	4			

1	currentTerm	5					
	votedFor	1					
	commitIndex	5					
	lastApplied	5					
	nextIndex	[6, 6, 6]					
	matchIndex	[5, 5, 0]					
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>3</td><td>4</td></tr></table>			1	1	1	3	4
1	1	1	3	4			

2	currentTerm	5						
	votedFor	-1						
	commitIndex	3						
	lastApplied	3						
	nextIndex	[]						
	matchIndex	[]						
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>2</td><td>2</td><td>2</td></tr></table>			1	1	1	2	2	2
1	1	1	2	2	2			

AppendEntriesReply
Term: 5
Success: False

0	currentTerm	5
	votedFor	1
	commitIndex	5
	lastApplied	5
	nextIndex	[]
	matchIndex	[]
		

prevLogIndex = 4
S1 log[4] = 3
S2 log[4] = 2

Mismatch!



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	[]
		

AppendEntries
Term: 5
LeaderID: 1
PrevLogIndex: 4
PrevLogTerm: 3
LeaderCommit: 5




0	currentTerm	5					
	votedFor	1					
	commitIndex	5					
	lastApplied	5					
	nextIndex	[]					
	matchIndex	[]					
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>3</td><td>4</td></tr></table>			1	1	1	3	4
1	1	1	3	4			

1	currentTerm	5					
	votedFor	1					
	commitIndex	5					
	lastApplied	5					
	nextIndex	[6, 6, 5]					
	matchIndex	[5, 5, 0]					
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>3</td><td>4</td></tr></table>			1	1	1	3	4
1	1	1	3	4			

2	currentTerm	5						
	votedFor	-1						
	commitIndex	3						
	lastApplied	3						
	nextIndex	[]						
	matchIndex	[]						
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>2</td><td>2</td><td>2</td></tr></table>			1	1	1	2	2	2
1	1	1	2	2	2			

AppendEntriesReply
Term: 5
Success: False

0	currentTerm	5
	votedFor	1
	commitIndex	5
	lastApplied	5
	nextIndex	[]
	matchIndex	[]
		

prevLogIndex = 3
S1 log[3] = 1
S2 log[3] = 1

Match!



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

AppendEntries
Term: 5
LeaderID: 1
PrevLogIndex: 3
PrevLogTerm: 1
LeaderCommit: 5


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



0	currentTerm	5					
	votedFor	1					
	commitIndex	5					
	lastApplied	5					
	nextIndex	[]					
	matchIndex	[]					
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>3</td><td>4</td></tr></table>			1	1	1	3	4
1	1	1	3	4			

1	currentTerm	5					
	votedFor	1					
	commitIndex	5					
	lastApplied	5					
	nextIndex	[6, 6, 4]					
	matchIndex	[5, 5, 0]					
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>3</td><td>4</td></tr></table>			1	1	1	3	4
1	1	1	3	4			

2	currentTerm	5						
	votedFor	-1						
	commitIndex	5						
	lastApplied	5						
	nextIndex	[]						
	matchIndex	[]						
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>2</td><td>2</td><td>2</td></tr></table>			1	1	1	2	2	2
1	1	1	2	2	2			

AppendEntriesReply
Term: 5
Success: True



0	currentTerm	5
	votedFor	1
	commitIndex	5
	lastApplied	5
	nextIndex	[]
	matchIndex	[]
	1	1
	1	3
	4	

1	currentTerm	5
	votedFor	1
	commitIndex	5
	lastApplied	5
	nextIndex	[6, 6, 6]
	matchIndex	[5, 5, 5]
	1	1
	1	3
	4	

Everyone is on the same page again

2	currentTerm	5
	votedFor	-1
	commitIndex	5
	lastApplied	5
	nextIndex	[]
	matchIndex	[]
	1	1
	1	3
	4	

Optimization to reduce
number of messages?

0	currentTerm	5			
	votedFor	1			
	commitIndex	5			
	lastApplied	5			
	nextIndex	[]			
	matchIndex	[]			
	1	1	1	3	4



1	currentTerm	5			
	votedFor	1			
	commitIndex	5			
	lastApplied	5			
	nextIndex	[6, 6, 6]			
	matchIndex	[5, 5, 0]			
	1	1	1	3	4



2	currentTerm	3				
	votedFor	2				
	commitIndex	3				
	lastApplied	3				
	nextIndex	[]				
	matchIndex	[]				
	1	1	1	2	2	2

AppendEntries
 Term: 5
 LeaderID: 1
 PrevLogIndex: 5
 PrevLogTerm: 4
 LeaderCommit: 5



0	currentTerm	5					
	votedFor	1					
	commitIndex	5					
	lastApplied	5					
	nextIndex	[]					
	matchIndex	[]					
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>3</td><td>4</td></tr></table>			1	1	1	3	4
1	1	1	3	4			

1	currentTerm	5					
	votedFor	1					
	commitIndex	5					
	lastApplied	5					
	nextIndex	[6, 6, 6]					
	matchIndex	[5, 5, 0]					
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>3</td><td>4</td></tr></table>			1	1	1	3	4
1	1	1	3	4			

2	currentTerm	5						
	votedFor	-1						
	commitIndex	3						
	lastApplied	3						
	nextIndex	[]						
	matchIndex	[]						
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>2</td><td>2</td><td>2</td></tr></table>			1	1	1	2	2	2
1	1	1	2	2	2			

Specify index of first log entry in the new term

AppendEntriesReply
Term: 5
Success: False
RequestedIndex: 4

0	currentTerm	5			
	votedFor	1			
	commitIndex	5			
	lastApplied	5			
	nextIndex	[]			
	matchIndex	[]			
	1	1	1	3	4



1	currentTerm	5			
	votedFor	1			
	commitIndex	5			
	lastApplied	5			
	nextIndex	[6, 6, 4]			
	matchIndex	[5, 5, 0]			
	1	1	1	3	4

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

AppendEntries
 Term: 5
 LeaderID: 1
 PrevLogIndex: 3
 PrevLogTerm: 1
 LeaderCommit: 5





0	currentTerm	5
	votedFor	1
	commitIndex	5
	lastApplied	5
	nextIndex	[]
	matchIndex	[]
	1	1
	1	3
	4	

1	currentTerm	5
	votedFor	1
	commitIndex	5
	lastApplied	5
	nextIndex	[6, 6, 6]
	matchIndex	[5, 5, 5]
	1	1
	1	3
	4	

2	currentTerm	5
	votedFor	-1
	commitIndex	5
	lastApplied	5
	nextIndex	[]
	matchIndex	[]
	1	1
	1	3
	4	

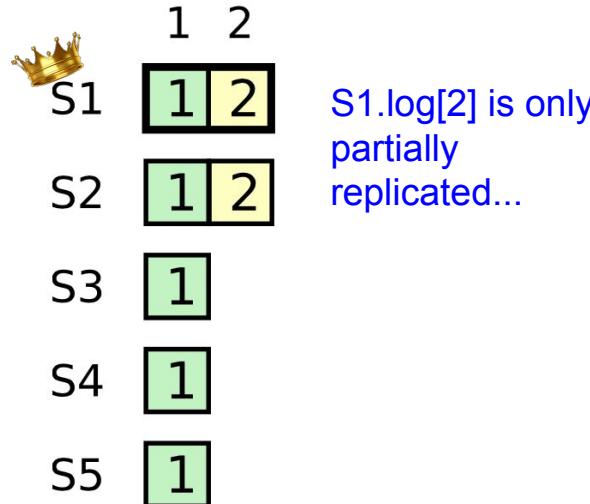
Decrement nextIndex
one term at a time

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

Caveat for committing old entries

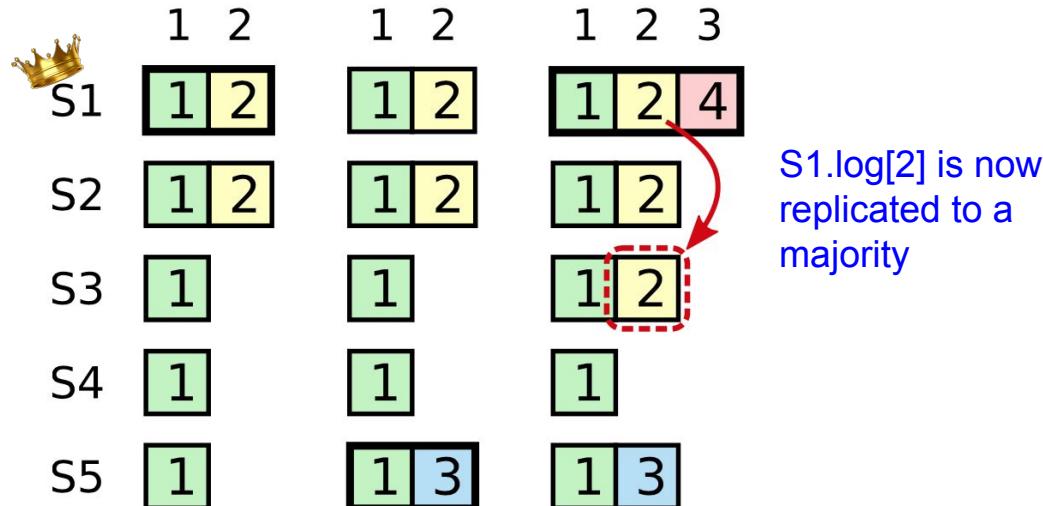
Can't assume an old entry has been committed *even if* it exists on a majority

	1	2	1	2
S1	1	2	1	2
S2	1	2	1	2
S3	1		1	
S4	1		1	
S5	1		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

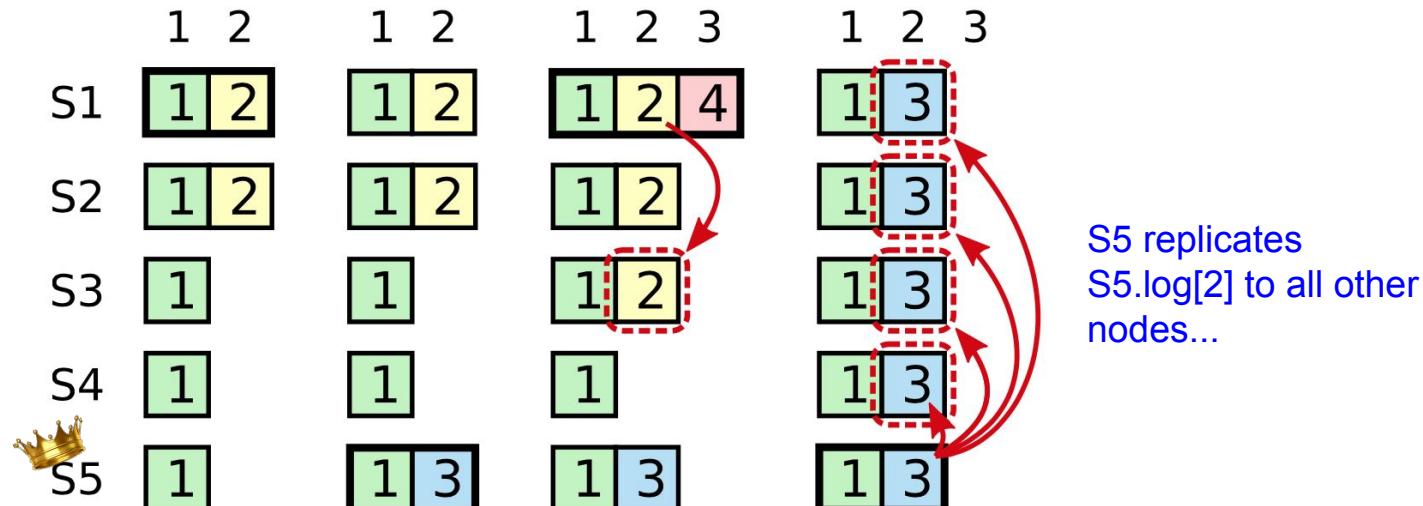


S1.log[2] is now replicated to a majority

S5 crashes,
S1 becomes leader

Caveat for committing old entries

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

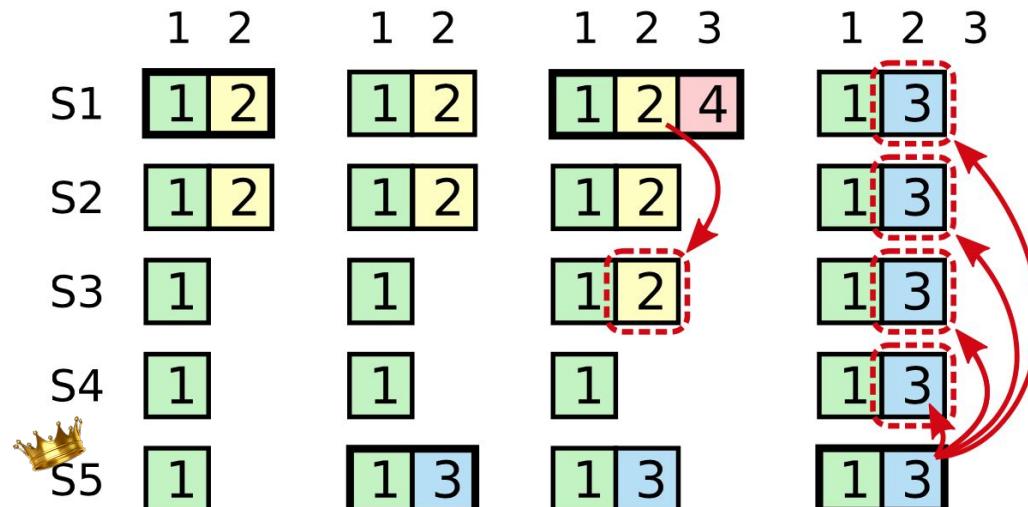


S1 crashes,
S5 becomes leader

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

Caveat for committing old entries

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

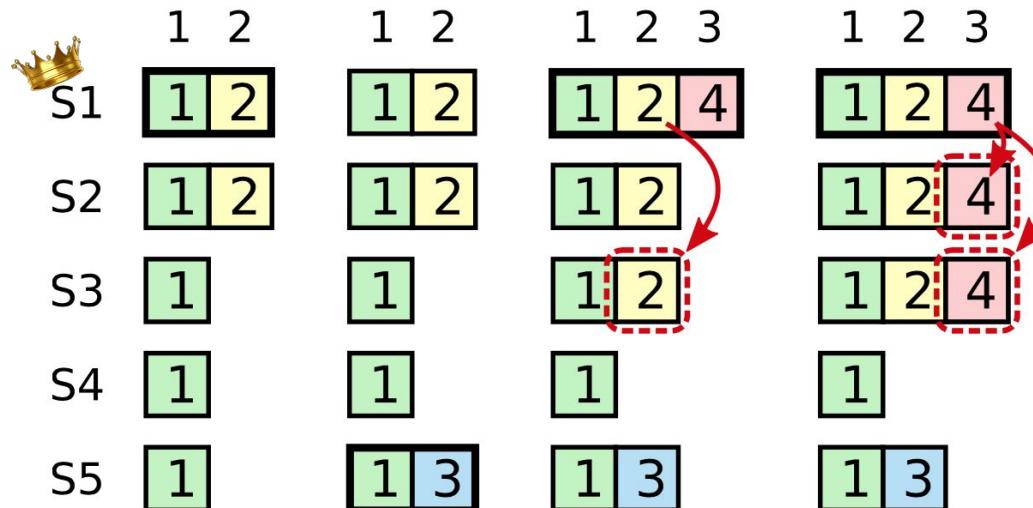


Entry 2 was overwritten
even though it was
replicated on a majority!

**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



Entry 2 is committed once
entry 3 is committed

**Commit old entries
indirectly**

S1 commits entry 3

Exercise...

Exercise...

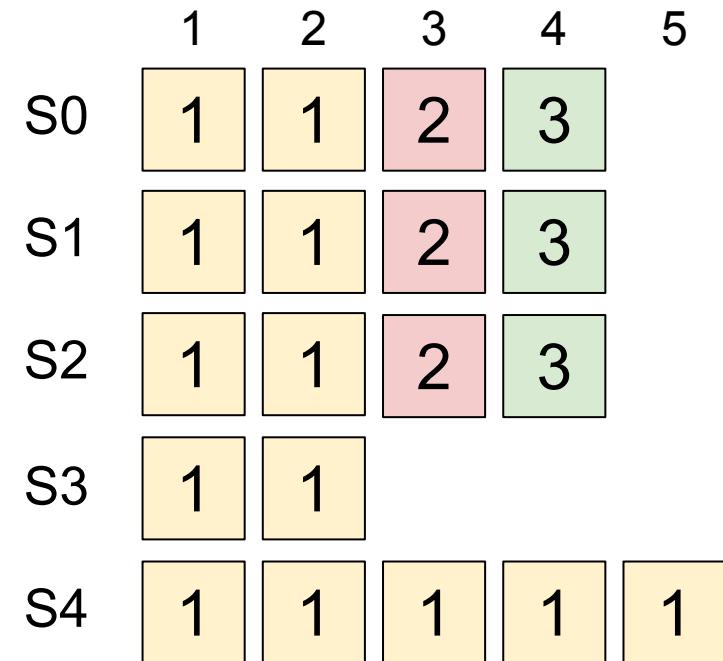
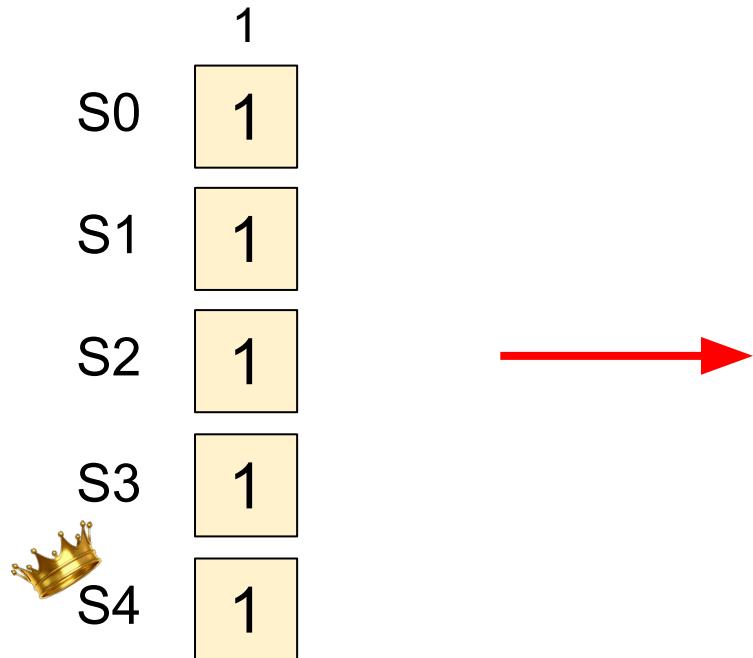
Rules for deciding which log is more up-to-date:

- 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?

	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...

	1	2
S0	1	1
S1	1	1
S2	1	1
S3	1	1
S4	1	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...

	1	2
S0	1	1
S1	1	1
S2	1	1
S3	1	1
	1	1
1	1	1
1	1	1
1	1	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...



	1	2			
S0	1	1			
S1	1	1			
S2	1	1			
S3	1	1			
S4	1	1	1	1	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...



	1	2		
S0	1	1	2	
S1	1	1	2	
S2	1	1	2	
S3	1	1		
S4	1	1	1	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...

	1	2		
S0	1	1	2	
S1	1	1	2	
S2	1	1	2	
S3	1	1		
S4	1	1	1	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...

	1	2	
S0	1	1	2
S1	1	1	2
S2	1	1	2
 S3	1	1	
 S4	1	1	1
	1	1	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...

	1	2		
S0	1	1	2	3
S1	1	1	2	3
S2	1	1	2	3
S3	1	1		
S4	1	1	1	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



Q2: Is this a possible configuration?

	1	2	3	4	5
S0	1	1	2	3	
S1	1	1	2	3	
S2	1	1	2	3	
S3	1	1	4		
S4	1	1	1	1	1

NO!

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

Q3: Is this a possible configuration?

	1	2	3	4	5
S0	1	1	5	6	
S1	1	1	5	6	
S2	1	1	5	6	
S3	1	1	4		
S4	1	1	1	1	1

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?

	1	2	3	4
S0	1	1		
S1	1	1	1	3
S2	1	1	3	

NO!

Let's try tracing the steps...

Q4: Is this a possible configuration?

	1	2	3	4
S0	1	1		
S1	1	1	1	
S2	1	1		

	1	2	3	4
S0	1	1		
S1	1	1	1	3
S2	1	1	3	

Q4: Is this a possible configuration?

	1	2	3	4
S0	1	1		
S1	1	1	1	
S2	1	1		

	1	2	3	4
S0	1	1		
S1	1	1	1	3
S2	1	1	3	

No one becomes leader in term 2...

Q4: Is this a possible configuration?

	1	2	3	4
S0	1	1		
	1	1	1	
S2	1	1	3	

	1	2	3	4
S0	1	1		
S1	1	1	1	3
S2	1	1	3	

Q4: Is this a possible configuration?

	1	2	3	4
S0	1	1		
X	1	1	1	
X	1	1	3	

	1	2	3	4
S0	1	1		
S1	1	1	1	3
S2	1	1	3	

Q4: Is this a possible configuration?

	1	2	3	4
S0	1	1		
S1	1	1	1	4
 S2	1	1	3	

	1	2	3	4
S0	1	1		
S1	1	1	1	3
S2	1	1	3	

**S0 previously voted for S2 in term 3
S0 can only vote for S1 for term 4!**

Q4: Is this a possible configuration?

	1	2	3	4
S0	1	1		
S1	1	1	1	3
S2	1	1	3	

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?

	1	2
S0	1	2
S1	1	2
S2	1	2
S3	1	
S4	1	

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?

	1	2	3
S0	1	1	2
S1	1	1	2
S2	1	1	2
S3	1	3	
S4	1		

NO!

S3 could become leader if S0 crashes

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

Q7: Is entry 3 (term 2) guaranteed to be committed?

	1	2	3	4
S0	1	1	2	4
S1	1	1	2	4
S2	1	1	2	
S3	1		3	
S4	1			

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?

	1	2	3	4
S0	1	1	2	4
S1	1	1	2	4
S2	1	1	2	
S3	1	3		
S4	1	1	2	4

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