UNIVERSITY of VIRGINIA

CS 4740 Cloud Computing (Fall 2024)
Overview of Lab 3A and 3B

Background Knowledge

Raft:

Consensus Algorithm

raft.github.io

https://thesquareplanet.com/blog/students-guide-to-raft/

State

Persistent state on all servers:
(Updated on stable storage before responding to RPCs)

currentTerm	latest term server has seen (initialized to 0 on first boot, increases monotonically)
votedFor	candidateId that received vote in current term (or null if none)
log[]	log entries; each entry contains command for state machine, and term when entry was received by leader (first index is 1)

Volatile state on all servers:

commitIndex	index of highest log entry known to be committed (initialized to 0, increases monotonically)
lastApplied	index of highest log entry applied to state machine (initialized to 0, increases monotonically)

Volatile state on leaders:
(Reinitialized after election)

nextIndex[]	for each server, index of the next log entry to send to that server (initialized to leader last log index + 1)
matchIndex[]	for each server, index of highest log entry known to be replicated on server (initialized to 0, increases monotonically)

AppendEntries RPC

Invoked by leader to replicate log entries (§5.3); also used as heartbeat (§5.2).

Arguments:

term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit	leader's commitIndex

Results:

term	currentTerm, for leader to update itself
success	true if follower contained entry matching prevLogIndex and prevLogTerm

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. Reply false if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm (§5.3)
3. If an existing entry conflicts with a new one (same index but different terms), delete the existing entry and all that follow it (§5.3)
4. Append any new entries not already in the log
5. If leaderCommit > commitIndex, set commitIndex = min(leaderCommit, index of last new entry)

RequestVote RPC

Invoked by candidates to gather votes (§5.2).

Arguments:

term	candidate's term
candidateId	candidate requesting vote
lastLogIndex	index of candidate's last log entry (§5.4)
lastLogTerm	term of candidate's last log entry (§5.4)

Results:

term	currentTerm, for candidate to update itself
voteGranted	true means candidate received vote

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. If votedFor is null or candidateId, and candidate's log is at least as up-to-date as receiver's log, grant vote (§5.2, §5.4)

Rules for Servers

All Servers:

- If commitIndex > lastApplied: increment lastApplied, apply log[lastApplied] to state machine (§5.3)
- If RPC request or response contains term T > currentTerm: set currentTerm = T, convert to follower (§5.1)

Followers (§5.2):

- Respond to RPCs from candidates and leaders
- If election timeout elapses without receiving AppendEntries RPC from current leader or granting vote to candidate: convert to candidate

Candidates (§5.2):

- On conversion to candidate, start election:
  - Increment currentTerm
  - Vote for self
  - Reset election timer
  - Send RequestVote RPCs to all other servers
- If votes received from majority of servers: become leader
- If AppendEntries RPC received from new leader: convert to follower
- If election timeout elapses: start new election

Leaders:

- Upon election: send initial empty AppendEntries RPCs (heartbeat) to each server; repeat during idle periods to prevent election timeouts (§5.2)
- If command received from client: append entry to local log, respond after entry applied to state machine (§5.3)
- If last log index ≥ nextIndex for a follower: send AppendEntries RPC with log entries starting at nextIndex
  - If successful: update nextIndex and matchIndex for follower (§5.3)
  - If AppendEntries fails because of log inconsistency: decrement nextIndex and retry (§5.3)
- If there exists an N such that N > commitIndex, a majority of matchIndex[i] ≥ N, and log[N].term == currentTerm: set commitIndex = N (§5.3, §5.4).

UNIVERSITY of VIRGINIA

3A Leader Election

The goal for Part 3A is

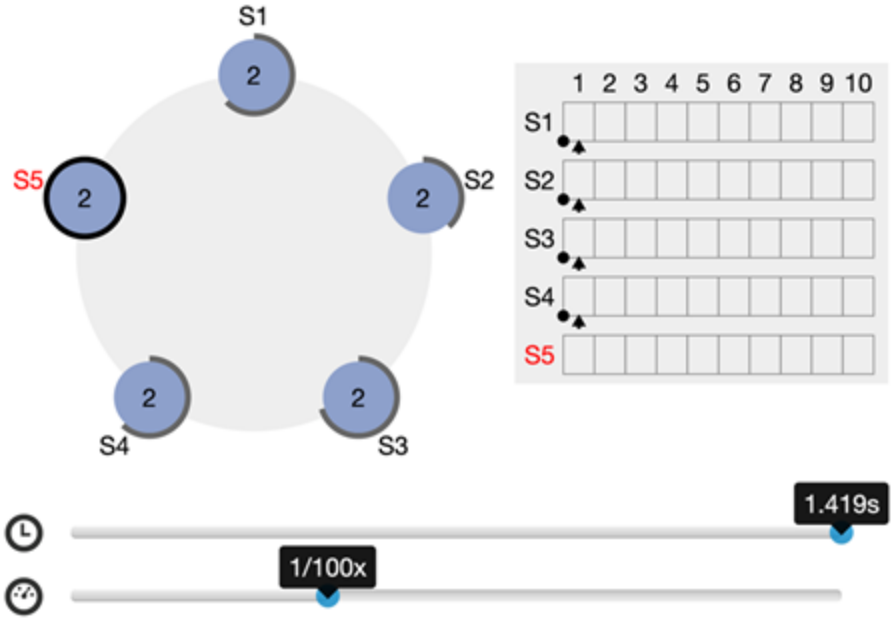
1) for a single leader to be elected

2) for the leader to remain the leader if there are no failures

3) for a new leader to take over if the old leader fails or if packets to/from the old leader are lost.

Understand the Raft progress first

We are using multiple threads to emulate the raft cluster.



Types of nodes

Leader: Handles all client requests and log replication.

Follower: Passively replicates logs from the leader.

Candidate: Seeks to become the leader through an election process.

The steps to finish the lab

1) Define the RPC argument

There are only two types of RPCs:

RequestVote RPCs are initiated by candidates during elections

AppendEntriesRPCs are initiated by leaders to replicate log entries

And to provide a form of heartbeat.（For lab3A, just focus on heartbeat)

2) Implement RequestVote and AppendEntries

Implement RequestVote

RequestVote RPC

Invoked by candidates to gather votes (§5.2).

Arguments:

term	candidate's term
candidateId	candidate requesting vote
lastLogIndex	index of candidate's last log entry (§5.4)
lastLogTerm	term of candidate's last log entry (§5.4)

Results:

term	currentTerm, for candidate to update itself
voteGranted	true means candidate received vote

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. If votedFor is null or candidateId, and candidate's log is at least as up-to-date as receiver's log, grant vote (§5.2, §5.4)

```
type RequestVoteArgs struct {
    // Your data here (3A, 3B).
}

//
// example RequestVote RPC reply structure.
// field names must start with capital letters!
//
type RequestVoteReply struct {
    // Your data here (3A).
}
```

AppendEntries

UNIVERSITY of VIRGINIA

AppendEntries RPC

Invoked by leader to replicate log entries (§5.3); also used as heartbeat (§5.2).

Arguments:

term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit	leader's commitIndex

Results:

term	currentTerm, for leader to update itself
success	true if follower contained entry matching prevLogIndex and prevLogTerm

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. Reply false if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm (§5.3)
3. If an existing entry conflicts with a new one (same index but different terms), delete the existing entry and all that follow it (§5.3)
4. Append any new entries not already in the log
5. If leaderCommit > commitIndex, set commitIndex = min(leaderCommit, index of last new entry)

Raise Election

Candidates (§5.2):

- On conversion to candidate, start election:
  - Increment currentTerm
  - Vote for self
  - Reset election timer
  - Send RequestVote RPCs to all other servers
- If votes received from majority of servers: become leader
- If AppendEntries RPC received from new leader: convert to follower
- If election timeout elapses: start new election

Election Trigger & Campaigning for Leadership

When a follower node does not receive a heartbeat from the current leader within a timeout period, it transitions to a candidate state.

The candidate requests votes from other nodes by incrementing the term and broadcasting a vote request.

Nodes vote for the first candidate they receive a valid request from, based on the latest known term.

Leader Responsibilities & Handle Failure

Sending Heartbeats: The leader sends regular heartbeats to maintain authority and ensure followers stay in sync.
Log Replication: Handles client requests and replicates changes to all follower nodes.

Leader Failure: When a leader fails, followers start a new election after the timeout period expires. A new leader is elected to continue the operation.
Candidate Failure: If a candidate fails or cannot achieve a majority, it reverts to a follower state, and the election is restarted.

3B Log Replication

Goal: implement leader & follower code to append new log entries

- Review 5.3 and 5.4 of the Raft paper
- Understand the Raft guarantees (Figure 3)

Implementation Step

- Define the log
- Add election restriction in RequestVote
- Update AppendEntries code to support logs
- Apply the logs to state machine

Define the Log

The log structure should contain

- Command interface{} (see Start)
- Term

5.3 Log replication

Once a leader has been elected, it begins servicing client requests. Each client request contains a command to be executed by the replicated state machines. The leader appends the command to its log as a new entry, then issues AppendEntries RPCs in parallel to each of the other servers to replicate the entry. When the entry has been safely replicated (as described below), the leader applies the entry to its state machine and returns the result of that execution to the client. If followers crash or run slowly, or if network packets are lost, the leader retries AppendEntries RPCs indefinitely (even after it has responded to the client) until all followers eventually store all log entries.

Logs are organized as shown in Figure 6. Each log entry stores a state machine command along with the term number when the entry was received by the leader. The term numbers in log entries are used to detect inconsistencies between logs and to ensure some of the properties in Figure 3. Each log entry also has an integer index iden-

Define the Log

The Start function send a command

- If sending to leader
  - Package it to a log
  - Add the log to its log slice
- If sending to other nodes
  - Return false

5.3 Log replication

Once a leader has been elected, it begins servicing client requests. Each client request contains a command to be executed by the replicated state machines. The leader appends the command to its log as a new entry, then issues AppendEntries RPCs in parallel to each of the other servers to replicate the entry. When the entry has been safely replicated (as described below), the leader applies the entry to its state machine and returns the result of that execution to the client. If followers crash or run slowly, or if network packets are lost, the leader retries AppendEntries RPCs indefinitely (even after it has responded to the client) until all followers eventually store all log entries.

Logs are organized as shown in Figure 6. Each log entry stores a state machine command along with the term number when the entry was received by the leader. The term numbers in log entries are used to detect inconsistencies between logs and to ensure some of the properties in Figure 3. Each log entry also has an integer index iden-

Election Restriction

Please refer to section 5.4.1

Only vote for up-to-date candidate

Add restriction in RequestVote

Before granting vote to a candidate

Raft uses the voting process to prevent a candidate from winning an election unless its log contains all committed entries. A candidate must contact a majority of the cluster in order to be elected, which means that every committed entry must be present in at least one of those servers. If the candidate's log is at least as up-to-date as any other log in that majority (where "up-to-date" is defined precisely below), then it will hold all the committed entries. The RequestVote RPC implements this restriction: the RPC includes information about the candidate's log, and the voter denies its vote if its own log is more up-to-date than that of the candidate.

Raft determines which of two logs is more up-to-date by comparing the index and term of the last entries in the logs. If the logs have last entries with different terms, then the log with the later term is more up-to-date. If the logs end with the same term, then whichever log is longer is more up-to-date.

Implement AppendEntries RPC

What to implement →

Leaders:

- Upon election: send initial empty AppendEntries RPCs (heartbeat) to each server; repeat during idle periods to prevent election timeouts (§5.2)
- If command received from client: append entry to local log, respond after entry applied to state machine (§5.3)
- If last log index ≥ nextIndex for a follower: send AppendEntries RPC with log entries starting at nextIndex
  - If successful: update nextIndex and matchIndex for follower (§5.3)
  - If AppendEntries fails because of log inconsistency: decrement nextIndex and retry (§5.3)
- If there exists an N such that N > commitIndex, a majority of matchIndex[i] ≥ N, and log[N].term == currentTerm: set commitIndex = N (§5.3, §5.4).

AppendEntries RPC

Invoked by leader to replicate log entries (§5.3); also used as heartbeat (§5.2).

Arguments:

term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit	leader's commitIndex

Results:

term	currentTerm, for leader to update itself
success	true if follower contained entry matching prevLogIndex and prevLogTerm

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. Reply false if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm (§5.3)
3. If an existing entry conflicts with a new one (same index but different terms), delete the existing entry and all that follow it (§5.3)
4. Append any new entries not already in the log
5. If leaderCommit > commitIndex, set commitIndex = min(leaderCommit, index of last new entry)

Leader Side

If command received from client: append entry to local log, respond after entry applied to state machine (§5.3)

- The Start function would send the entry
- Leader append the log to its local log
- sendAppendEntries

Leader Side

If last log index ≥ nextIndex for a follower: send AppendEntries RPC with log entries starting at nextIndex

```
Args = AppendEntriesArgs{
        Term: …
        LeaderId: …
        PrevLogIndex: rf.nextIndex[server] - 1
        PrevLogTerm: rf.log[PrevLogIndex].Term
        Entries: rf.log[rf.nextIndex[server]: ]
        LeaderCommit: …
}
```

Empty entries can serve as heartbeat

Arguments:

term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit	leader's commitIndex

Leader Side

If last log index ≥ nextIndex for a follower: send AppendEntries RPC with log entries starting at nextIndex

- If successful: update nextIndex and matchIndex for follower
- If AppendEntries fails because of log inconsistency: decrement nextIndex and retry
  - nextIndex[follower]-- (or decrease further if optimized)
  - Resend the RPC

Leader Side

If there exists an N such that N > commitIndex, a majority of matchIndex[i] ≥ N, and log[N].term == currentTerm: set commitIndex = N

- Leader append (start) → servers append (appendEntriesRPC) → Leader commit (majority replicated) → servers commit (see leaderCommit)

Leader Side

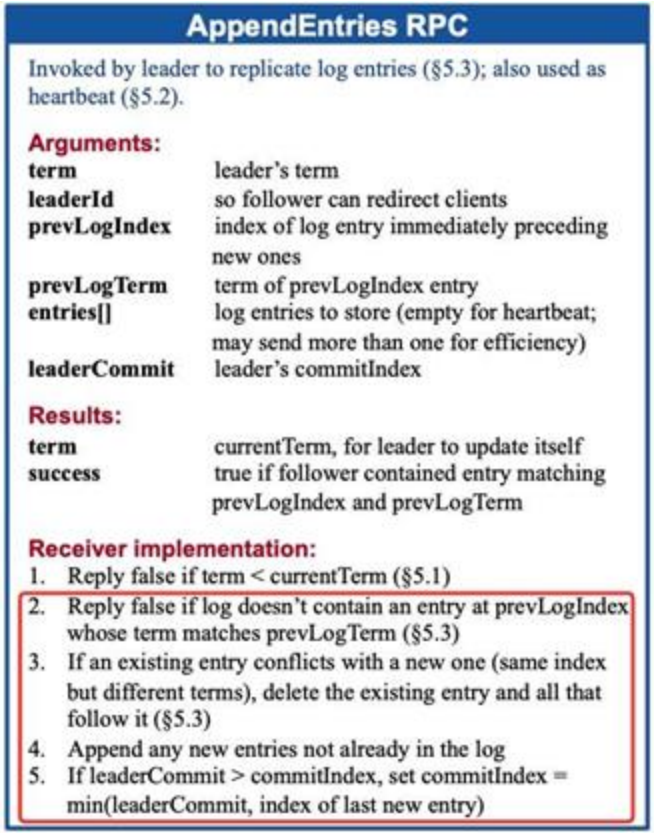
If there exists an N such that N > commitIndex, a majority of matchIndex[i] ≥ N, and log[N].term == currentTerm: set commitIndex = N

- Leader append (start) → servers append (appendEntriesRPC) → Leader commit (majority replicated) → servers commit (see leaderCommit)
- After handling the reply from EACH server, check the highest index entry, where more than a half servers successfully replicate
- If so, leader can set commitIndex and apply to its state machine

Follower Side

2 & 3

- log[prevLogIndex].Term != prevLogTerm
- success = false



AppendEntries RPC

Invoked by leader to replicate log entries (§5.3); also used as heartbeat (§5.2).

Arguments:

term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit	leader's commitIndex

Results:

term	currentTerm, for leader to update itself
success	true if follower contained entry matching prevLogIndex and prevLogTerm

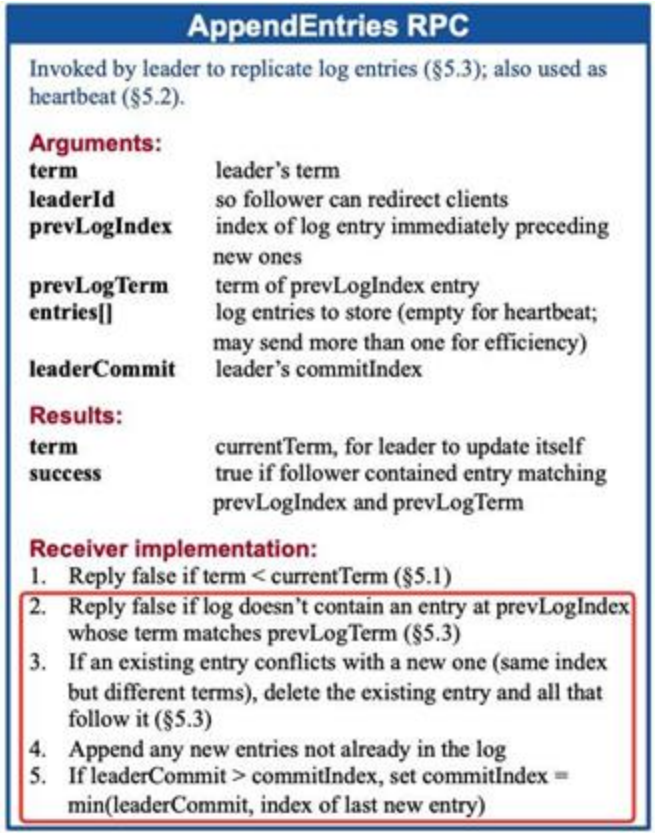
Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. Reply false if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm (§5.3)
3. If an existing entry conflicts with a new one (same index but different terms), delete the existing entry and all that follow it (§5.3)
4. Append any new entries not already in the log
5. If leaderCommit > commitIndex, set commitIndex = min(leaderCommit, index of last new entry)

Follower Side

4

- Logs <= prevLogIndex are good
- restLog = logs > prevLogIndex
- Compare restLog with entries[]
  - len(restlog) < len(entries) or any term inconsistency
  - Otherwise, use restLog



AppendEntries RPC

Invoked by leader to replicate log entries (§5.3); also used as heartbeat (§5.2).

Arguments:

term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit	leader's commitIndex

Results:

term	currentTerm, for leader to update itself
success	true if follower contained entry matching prevLogIndex and prevLogTerm

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. Reply false if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm (§5.3)
3. If an existing entry conflicts with a new one (same index but different terms), delete the existing entry and all that follow it (§5.3)
4. Append any new entries not already in the log
5. If leaderCommit > commitIndex, set commitIndex = min(leaderCommit, index of last new entry)

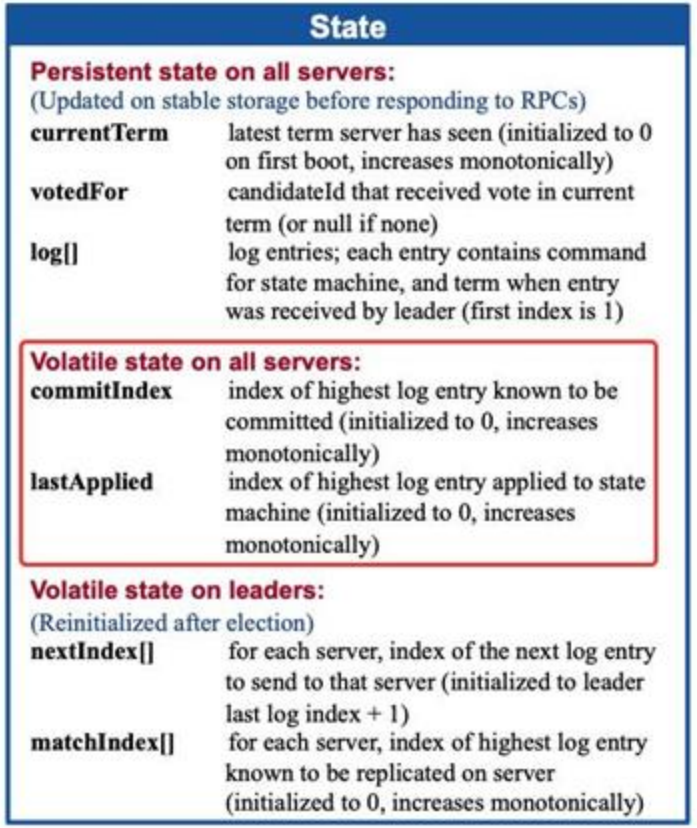
Apply to State Machine

Committed: safe to apply to state machine

When commitIndex increase,

Apply committed, but no applied logs

To state machine

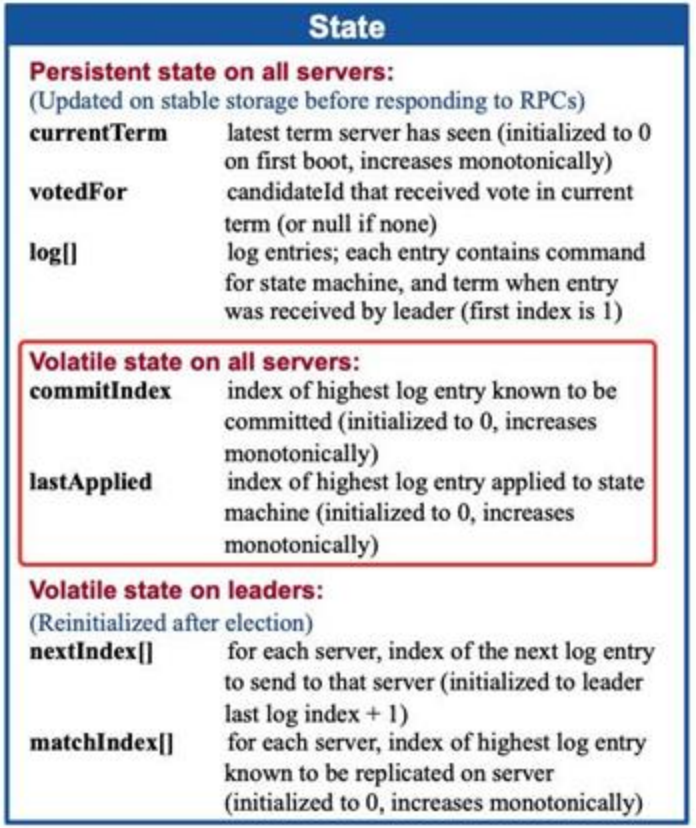


Apply to State Machine

How?

For logs to apply

YourChannel <- ApplyMsg{...}



Architecture Suggestion

- In Make, after init, start a go routine to run a infinite loop
  - Follower: check heartbeat timeout
  - Leader: broadcast AppendEntries/heartbeat RPCs
  - Candidate: broadcast RequestVote RPCs, status change
  - Sleep for some time