

State	
<b>Persistent state on all servers:</b> (Updated on stable storage before responding to RPCs)	
<b>currentTerm</b>	latest term server has seen (initialized to 0 on first boot, increases monotonically)
<b>votedFor</b>	candidateId that received vote in current term (or null if none)
<b>log[]</b>	log entries; each entry contains command for state machine, and term when entry was received by leader (first index is 1)
<b>Volatile state on all servers:</b>	
<b>commitIndex</b>	index of highest log entry known to be committed (initialized to 0, increases monotonically)
<b>lastApplied</b>	index of highest log entry applied to state machine (initialized to 0, increases monotonically)
<b>Volatile state on leaders:</b> (Reinitialized after election)	
<b>nextIndex[]</b>	for each server, index of the next log entry to send to that server (initialized to leader last log index + 1)
<b>matchIndex[]</b>	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).	
<b>Arguments:</b>	
<b>term</b>	leader's term
<b>leaderId</b>	so follower can redirect clients
<b>prevLogIndex</b>	index of log entry immediately preceding new ones
<b>prevLogTerm</b>	term of prevLogIndex entry
<b>entries[]</b>	log entries to store (empty for heartbeat; may send more than one for efficiency)
<b>leaderCommit</b>	leader's commitIndex
<b>Results:</b>	
<b>term</b>	currentTerm, for leader to update itself
<b>success</b>	true if follower contained entry matching prevLogIndex and prevLogTerm
<b>Receiver implementation:</b>	
<ol style="list-style-type: none"> <li>1. Reply false if term &lt; currentTerm (§5.1)</li> <li>2. Reply false if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm (§5.3)</li> <li>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)</li> <li>4. Append any new entries not already in the log</li> <li>5. If leaderCommit &gt; commitIndex, set commitIndex = min(leaderCommit, index of last new entry)</li> </ol>	

RequestVote RPC	
Invoked by candidates to gather votes (§5.2).	
<b>Arguments:</b>	
<b>term</b>	candidate's term
<b>candidateId</b>	candidate requesting vote
<b>lastLogIndex</b>	index of candidate's last log entry (§5.4)
<b>lastLogTerm</b>	term of candidate's last log entry (§5.4)
<b>Results:</b>	
<b>term</b>	currentTerm, for candidate to update itself
<b>voteGranted</b>	true means candidate received vote
<b>Receiver implementation:</b>	
<ol style="list-style-type: none"> <li>1. Reply false if term &lt; currentTerm (§5.1)</li> <li>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)</li> </ol>	

Rules for Servers	
<b>All Servers:</b>	
<ul style="list-style-type: none"> <li>• If commitIndex &gt; lastApplied: increment lastApplied, apply log[lastApplied] to state machine (§5.3)</li> <li>• If RPC request or response contains term T &gt; currentTerm: set currentTerm = T, convert to follower (§5.1)</li> </ul>	
<b>Followers (§5.2):</b>	
<ul style="list-style-type: none"> <li>• Respond to RPCs from candidates and leaders</li> <li>• If election timeout elapses without receiving AppendEntries RPC from current leader or granting vote to candidate: convert to candidate</li> </ul>	
<b>Candidates (§5.2):</b>	
<ul style="list-style-type: none"> <li>• On conversion to candidate, start election: <ul style="list-style-type: none"> <li>• Increment currentTerm</li> <li>• Vote for self</li> <li>• Reset election timer</li> <li>• Send RequestVote RPCs to all other servers</li> </ul> </li> <li>• If votes received from majority of servers: become leader</li> <li>• If AppendEntries RPC received from new leader: convert to follower</li> <li>• If election timeout elapses: start new election</li> </ul>	
<b>Leaders:</b>	
<ul style="list-style-type: none"> <li>• Upon election: send initial empty AppendEntries RPCs (heartbeat) to each server; repeat during idle periods to prevent election timeouts (§5.2)</li> <li>• If command received from client: append entry to local log, respond after entry applied to state machine (§5.3)</li> <li>• If last log index ≥ nextIndex for a follower: send AppendEntries RPC with log entries starting at nextIndex <ul style="list-style-type: none"> <li>• If successful: update nextIndex and matchIndex for follower (§5.3)</li> <li>• If AppendEntries fails because of log inconsistency: decrement nextIndex and retry (§5.3)</li> </ul> </li> <li>• If there exists an N such that N &gt; commitIndex, a majority of matchIndex[i] ≥ N, and log[N].term == currentTerm: set commitIndex = N (§5.3, §5.4).</li> </ul>	

A condensed summary of the Raft consensus algorithm (excluding membership changes and log compaction), extracted from the original Raft paper.