on RequestVote request from peer

```
if currentTerm < m.term:
| stepDown(m.term)
if (currentTerm == m.term and
    votedFor in [None, peer] and
    (m.lastLogTerm > logTerm(len(log)) or
    (m.lastLogTerm == logTerm(len(log)) and
     m.lastLogIndex >= len(log)))):
| granted := True
| votedFor = peer
| electionAlarm = now() + rand(1.0, 2.0) *
|                         ELECTION_TIMEOUT
else:
| granted := False
reply {term: currentTerm,
       granted: granted}
```

on RequestVote response from peer

```
if currentTerm < m.term:
| stepDown(m.term)
if (state == CANDIDATE and
    currentTerm == m.term):
| rpcDue[peer] = INFINITY
| voteGranted[peer] = m.granted
```

on AppendEntries request from peer

```
if currentTerm < m.term:
| stepDown(m.term)
if currentTerm > m.term:
| reply {term: currentTerm,
|        success: False}
else:
| leader = peer
| state = FOLLOWER
| electionAlarm = now() + rand(1.0, 2.0) *
|                         ELECTION_TIMEOUT
| success := (m.prevIndex == 0 or
|    (m.prevIndex <= len(log) and
|     log[m.prevIndex].term == m.prevTerm))
| if success:
| | index := m.prevIndex
| | for j := 1..len(m.entries):
| | | index += 1
| | | if getTerm(index) != m.entries[j].term:
| | | | log = log[1..(index-1)] + m.entries[j]
| | commitIndex = min(m.commitIndex, index)
| else:
| | index = 0
| reply {term: currentTerm,
|        success: success,
|        matchIndex: index}
```

on AppendEntries response from peer

```
if currentTerm < m.term:
| stepDown(m.term)
elif state == LEADER and currentTerm == m.term:
| if m.success:
| | matchIndex[peer] = m.matchIndex
| | nextIndex[peer] = m.matchIndex + 1
| else:
| | nextIndex[peer] = max(1, nextIndex[peer] - 1)
```

on StateMachine request from client

```
if state == LEADER:
| log.append({term: currentTerm,
|             command: m.command})
```

helper functions

```
def stepDown(newTerm):
| currentTerm = newTerm
| state = FOLLOWER
| votedFor = None
| if electionAlarm < now():
| | electionAlarm = now() + rand(1.0, 2.0) *
| |                         ELECTION_TIMEOUT
def logTerm(index):
| if index < 1 or index > len(log):
| | return 0
| else:
| | return log[index].term
```

start new election

```
on (state in [FOLLOWER, CANDIDATE] and
    electionAlarm < now()):
| electionAlarm = now() + rand(1.0, 2.0) *
|                         ELECTION_TIMEOUT
| currentTerm += 1
| votedFor = serverID
| state = CANDIDATE
| foreach peer:
| | # reset all state for peer
```

send RequestVote to peer

```
on (state == CANDIDATE and
    rpcDue[peer] < now()):
| rpcDue[peer] = now() + RPC_TIMEOUT
| send RequestVote to peer {
|   term: currentTerm,
|   lastLogTerm: logTerm(len(log)),
|   lastLogIndex: len(log)}
```

become leader

```
on (state == CANDIDATE and
    sum(voteGranted) + 1 > NUM_SERVERS / 2:
| state = LEADER
| leader = localhost
| foreach peer:
| | nextIndex[peer] = len(log) + 1
```

send AppendEntries to peer

```
on (state == LEADER and
    (matchIndex[peer] < len(log) or
     rpcDue[peer] < now()):
| rpcDue[peer] = now() + ELECTION_TIMEOUT / 2
| lastIndex := choose in (nextIndex[peer] - 1)..len(log)
| nextIndex[peer] = lastIndex
| send AppendEntries to peer {
|   term: currentTerm,
|   prevIndex: nextIndex[peer] - 1,
|   prevTerm: getTerm(nextIndex[peer] - 1),
|   entries: log[nextIndex[peer]..lastIndex],
|   commitIndex: commitIndex}
```

advance commit index

```
n := sorted(matchIndex + [len(log)])[NUM_SERVERS / 2ish]
on (state == LEADER and
    logTerm(n) == currentTerm):
| commitIndex = n
```

advance state machine

```
on lastApplied < commitIndex:
| lastApplied += 1
| result := stateMachine.apply(log[lastApplied])
| if (state == Leader and
|       logTerm(lastApplied) == currentTerm):
| | # send result to client
```

server state

```
# FOLLOWER, CANDIDATE, or LEADER
state := FOLLOWER
# latest term server has seen (increases monotonically)
currentTerm := 1
# candidate that received vote in current term
votedFor := None
# log entries; each entry contains command for state
# machine, and term when entry was received by leader
log := [] # indexed from 1
# index of highest log entry known to be committed
commitIndex := 0
# time after which to start new election
electionAlarm := 0.0
# applies committed commands in log order
stateMachine := new SM()
# identity of last known leader
leader := None

# State per peer, valid only for the current term
foreach peer:
| # time after which to send another RPC
| # (RequestVote or heartbeat)
| rpcDue[peer] := 0.0
| # True if peer has granted this server its vote
| voteGranted[peer] := False
| # index of highest log entry known to be replicated
| # on peer
| matchIndex[peer] := 0
| # index of next log entry to send to peer
| nextIndex[peer] := 1
```

Figure 1: alternative cheatsheet