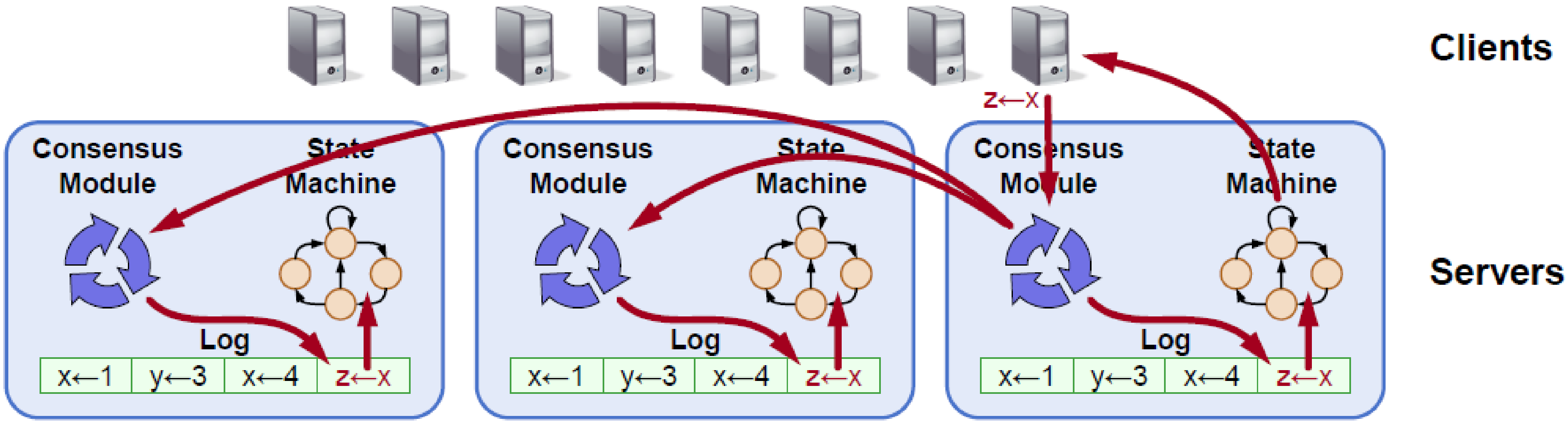
IF 3230 Sistem Paralel dan Terdistribusi

Raft Consensus

Raft

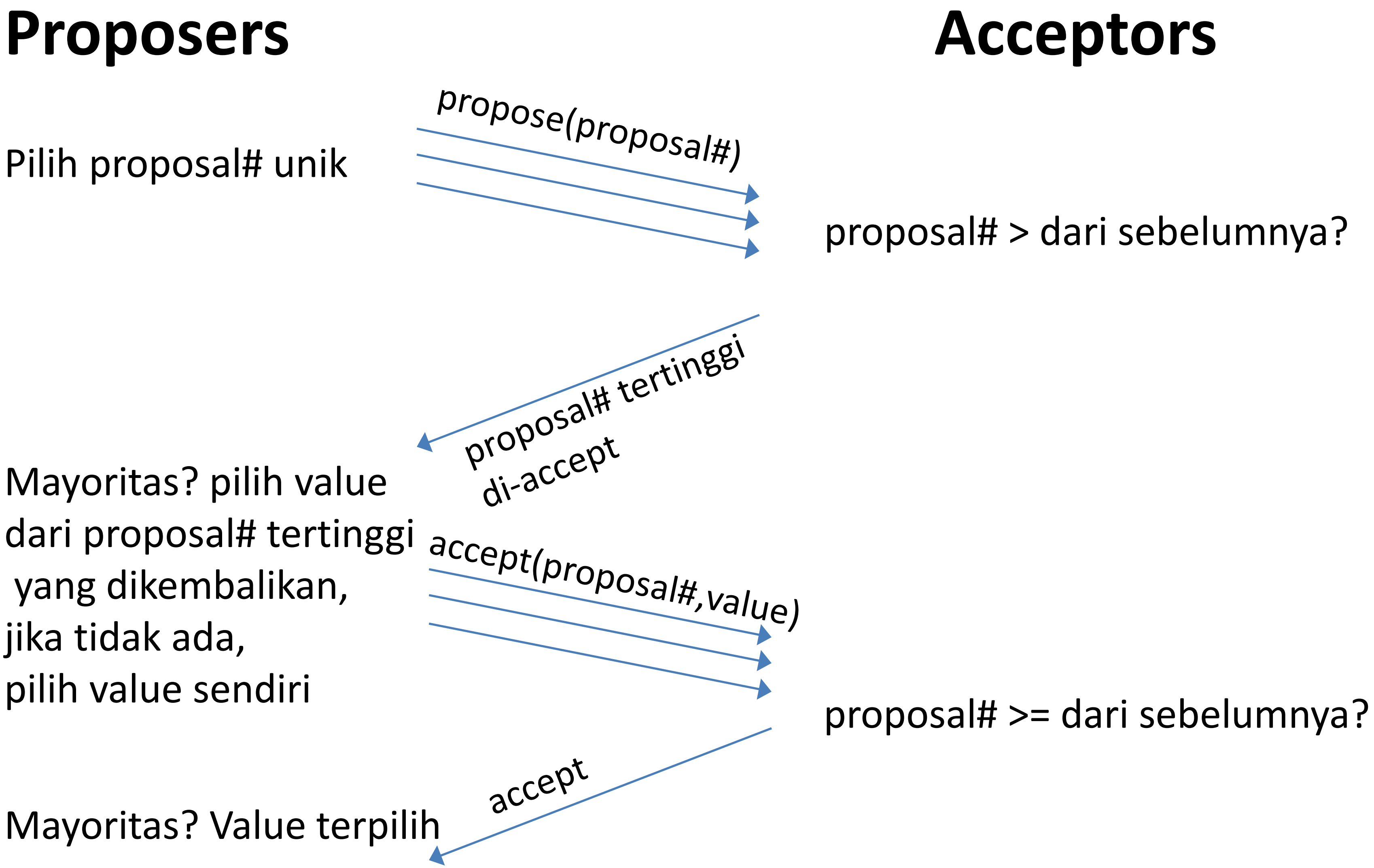
- Dikembangkan oleh Ousterhout & Ongaro
- goal: implementasi konsensus yang lebih mudah dipahami
- premis: algoritma paxos sulit untuk dipahami

Model sistem



- konsensus digunakan untuk membangun replicated state machine
- request client dicatat pada replicated log
- consensus module menjamin log konsisten
- entri log yang committed dieksekusi oleh state machine

Paxos



Proposers

Acceptors

Pilih proposal# unik

propose(proposal#)

proposal# > dari sebelumnya?

proposal# tertinggi
di-accept

Mayoritas? pilih value
dari proposal# tertinggi
yang dikembalikan,
jika tidak ada,
pilih value sendiri

accept(proposal#,value)

proposal# >= dari sebelumnya?

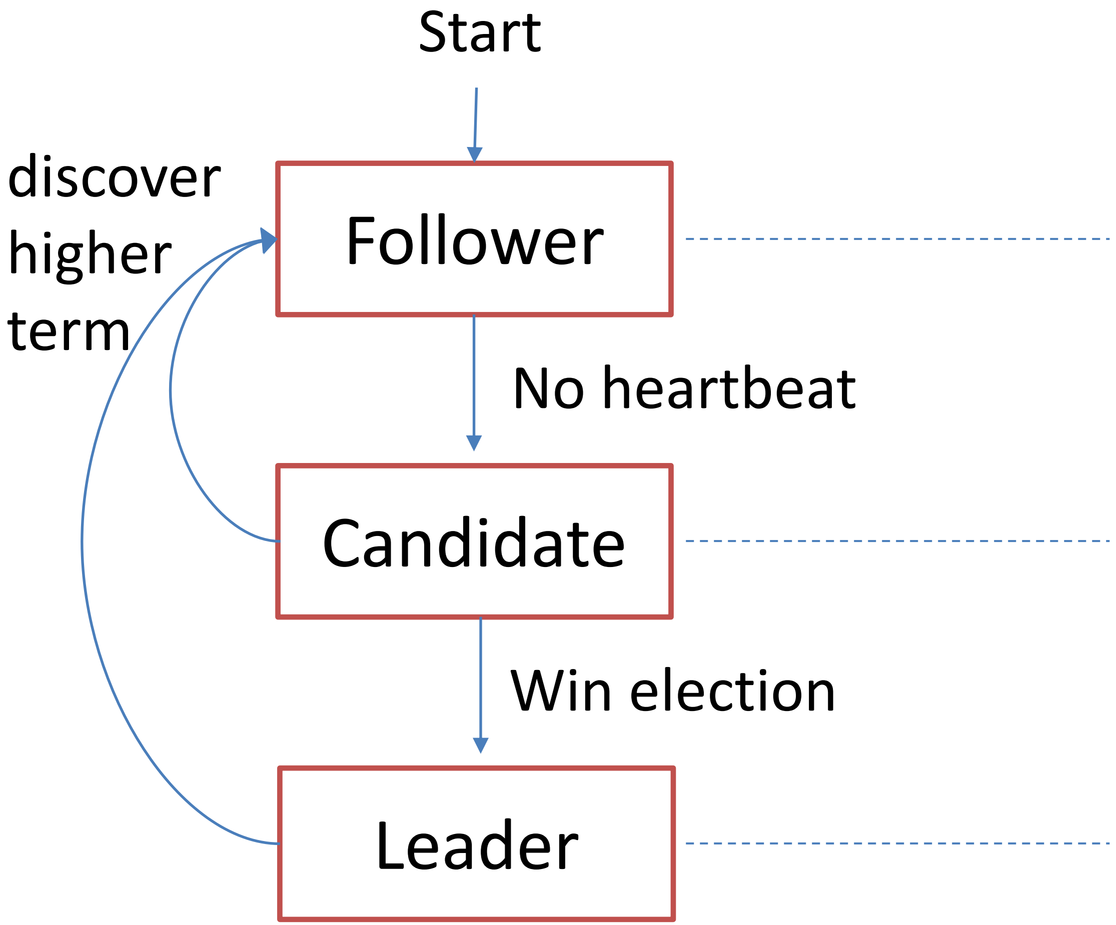
accept

Mayoritas? Value terpilih

Raft

- Leader election
  - pemilihan leader
  - pendeteksian crash
- Log replication
  - leader menerima request dari client, append log
  - replikasi log ke server lain
- Safety
  - menjamin log konsisten
  - server yang memiliki log yang up to date yang dapat menjadi leader

server state



Start

Follower

No heartbeat

Candidate

Win election

Leader

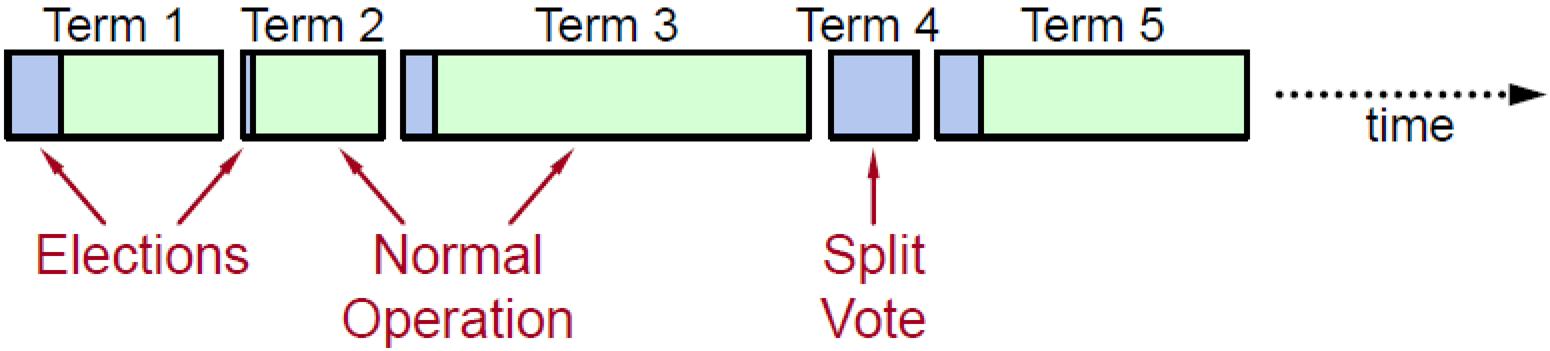
discover
higher
term

Pasif,
menunggu heartbeat
dan time out

issue RequestVote RPC
pemilihan leader

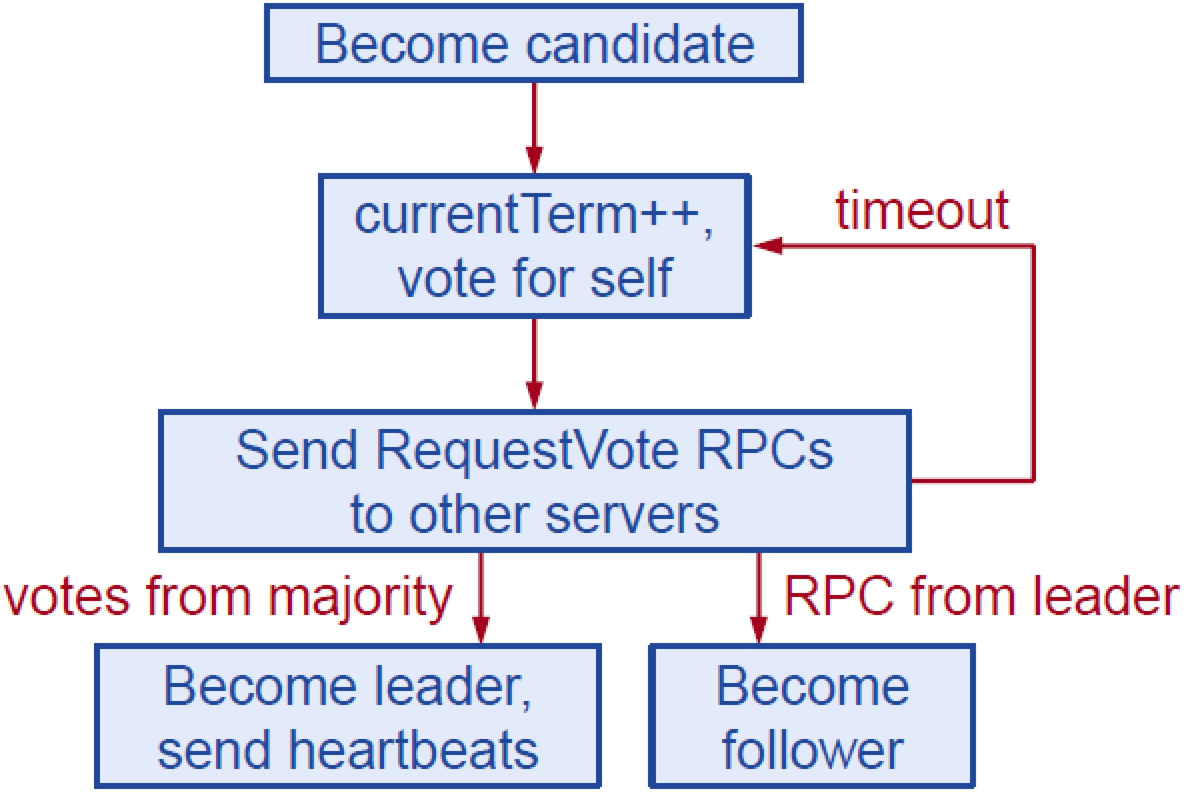
issue AppendEntries RPC
replikasi log
heartbeat

Terms



- hanya max 1 leader per term
- ada term yang tidak memiliki leader
- setiap server mengelola nilai current term
  - dipertukarkan pada setiap RPC
  - jika peer memiliki term lebih baru, update term, berubah menjadi follower
  - jika incoming RPC memiliki term lebih lama, reply dengan error

Leader election



Become candidate

currentTerm++,
vote for self

timeout

Send RequestVote RPCs
to other servers

votes from majority

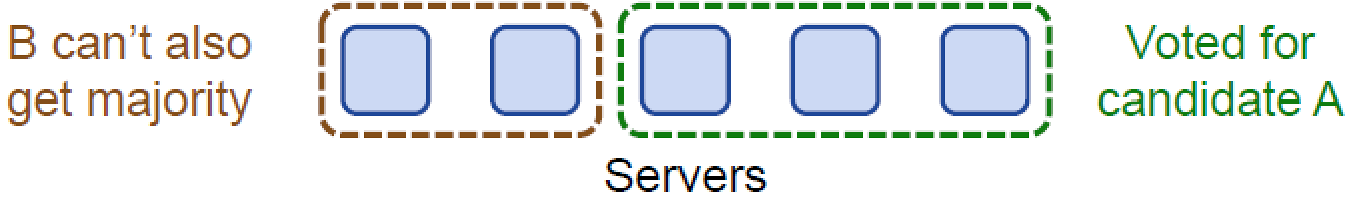
RPC from leader

Become leader,
send heartbeats

Become
follower

Election correctness

- Safety: hanya at most 1 winner per term
  - setiap server hanya memberikan 1 vote per term
  - perlu mayoritas

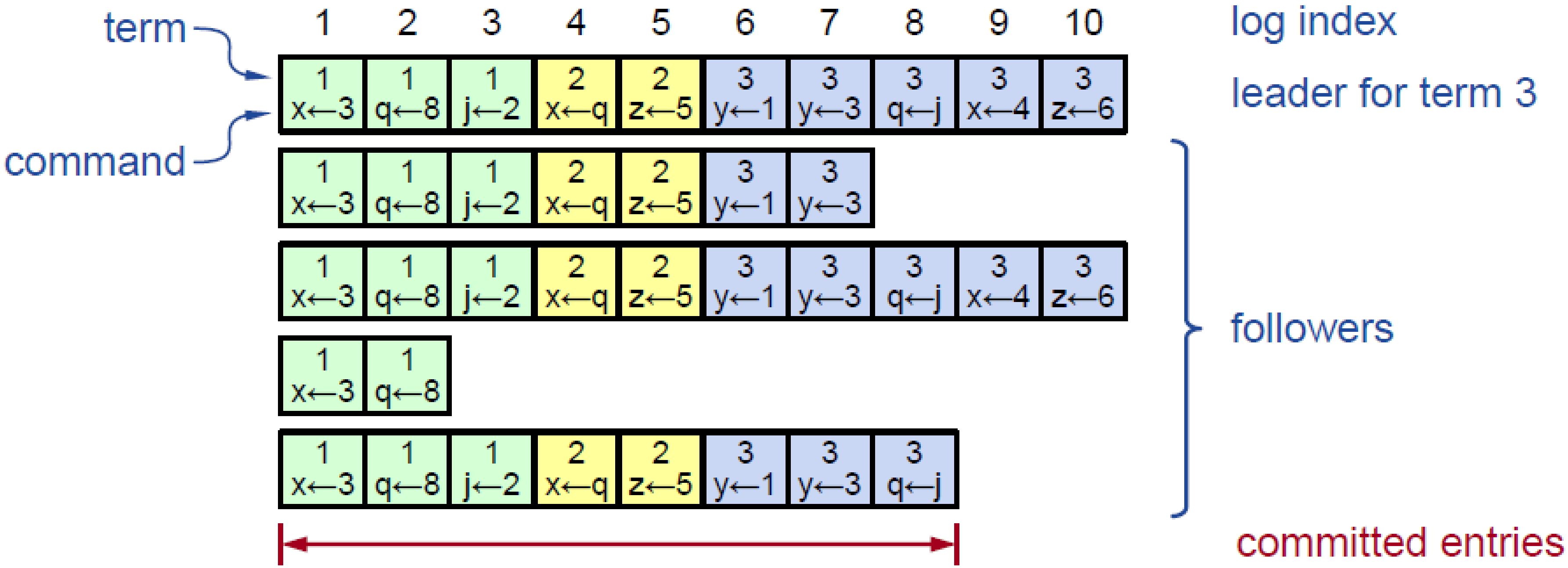


- Liveness: eventually, akan ada kandidat yang menang
  - menggunakan timeout random [T, 2T]
  - satu server akan timeout dan win election sebelum server lainnya timeout
  - T >> broadcast time

Operasi normal

- client sends command ke leader
- leader append command ke log pada leader
- leader sends AppendEntries RPC ke semua follower
- saat entri baru sudah di-commit
  - leader mengeksekusi command, dan mengirimkan hasilnya ke client
  - leader memberitahu follower ttg committed entries pada AppendEntries RPC berikutnya
  - follower mengeksekusi committed command

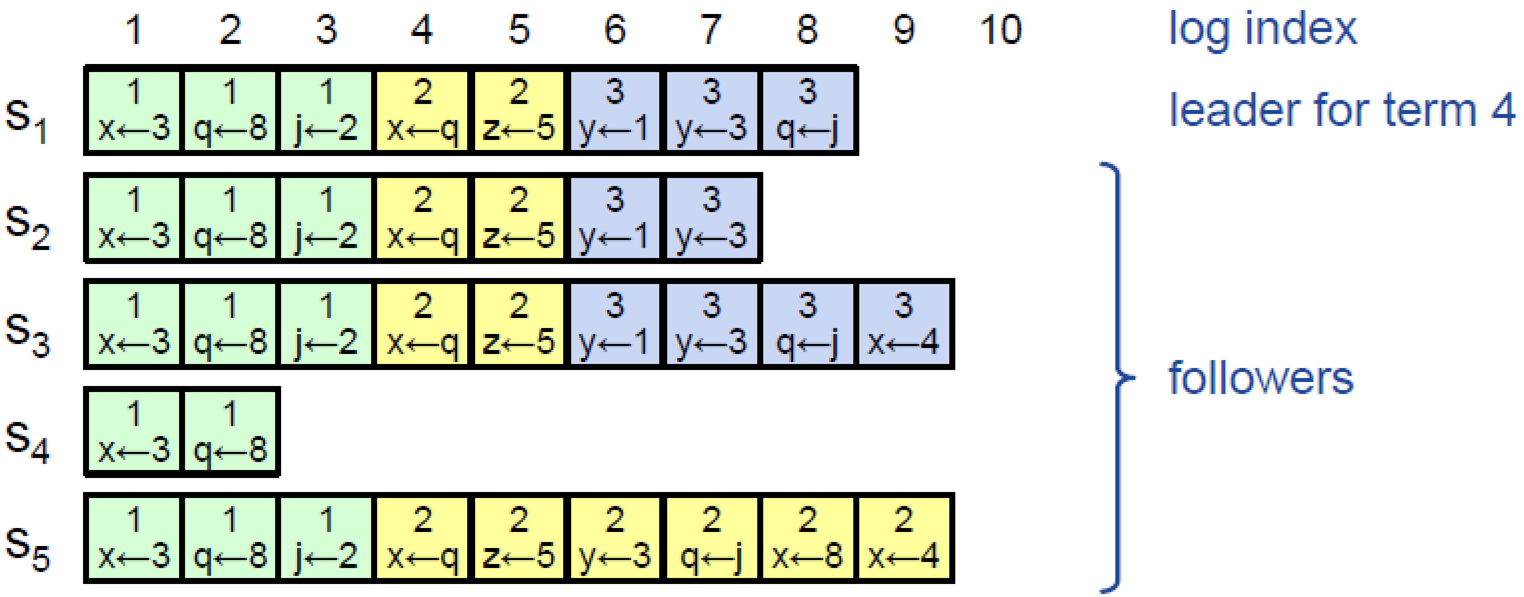
Struktur Log



	1	2	3	4	5	6	7	8	9	10
leader for term 3	1 x←3	1 q←8	1 j←2	2 x←q	2 z←5	3 y←1	3 y←3	3 q←j	3 x←4	3 z←6
followers	1 x←3	1 q←8	1 j←2	2 x←q	2 z←5	3 y←1	3 y←3			
	1 x←3	1 q←8	1 j←2	2 x←q	2 z←5	3 y←1	3 y←3	3 q←j	3 x←4	3 z←6
	1 x←3	1 q←8								
	1 x←3	1 q←8	1 j←2	2 x←q	2 z←5	3 y←1	3 y←3	3 q←j		

- entri di-commit jika sudah safe untuk dieksekusi
  - entri ter-replikasi pada mayoritas server

Inkonsisten log

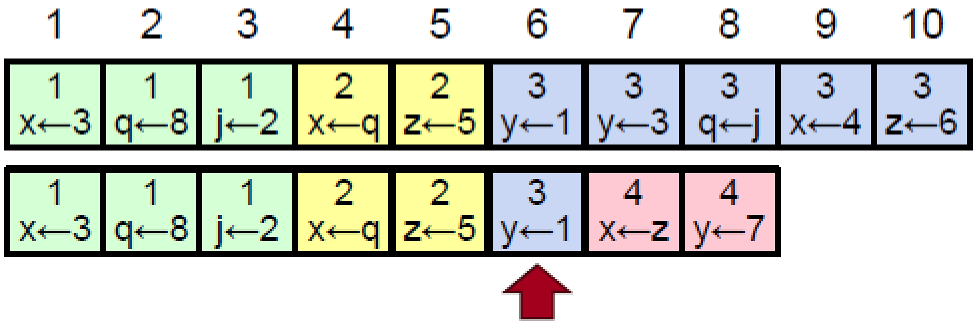


	1	2	3	4	5	6	7	8	9	10	log index
s_1	1 x←3	1 q←8	1 j←2	2 x←q	2 z←5	3 y←1	3 y←3	3 q←j			leader for term 4
s_2	1 x←3	1 q←8	1 j←2	2 x←q	2 z←5	3 y←1	3 y←3				followers
s_3	1 x←3	1 q←8	1 j←2	2 x←q	2 z←5	3 y←1	3 y←3	3 q←j	3 x←4		
s_4	1 x←3	1 q←8									
s_5	1 x←3	1 q←8	1 j←2	2 x←q	2 z←5	2 y←3	2 q←j	2 x←8	2 x←4		

- crash dapat mengakibatkan log inkonsisten
- Raft meminimal perbaikan inkonsistensi
  - leader selalu berasumsi log nya benar
  - normal operation akan memperbaiki semua inkonsistensi

Log Matching property

- jika log entri pada server yang berbeda memiliki term dan indeks yang sama:
  - log tersebut memiliki command yang sama
  - identik untuk semua entri sebelumnya

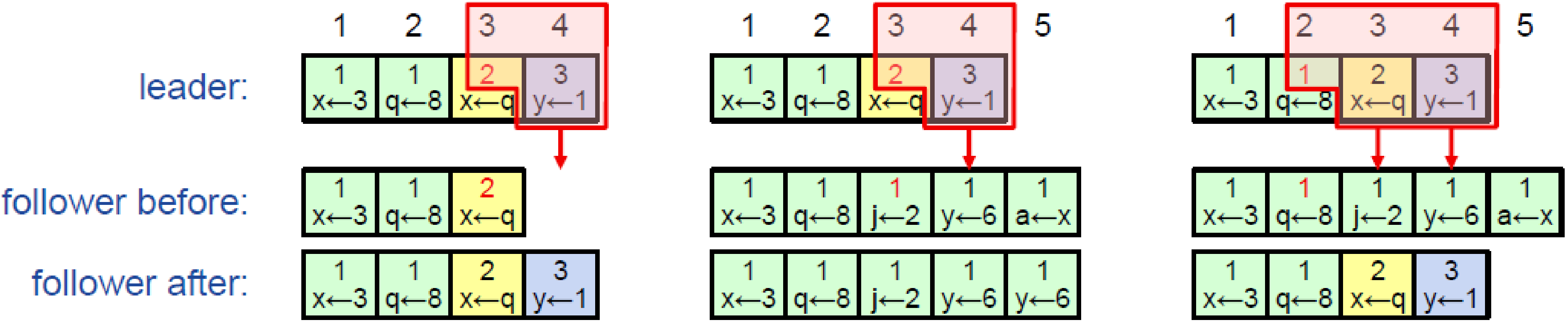


1	2	3	4	5	6	7	8	9	10
1 x←3	1 q←8	1 j←2	2 x←q	2 z←5	3 y←1	3 y←3	3 q←j	3 x←4	3 z←6
1 x←3	1 q←8	1 j←2	2 x←q	2 z←5	3 y←1	4 x←z	4 y←7		

- jika sebuah entri committed, maka semua entri sebelumnya juga committed

AppendEntries consistency check

- AppendEntries menyertakan <index, term> untuk entri sebelumnya
- follower harus berisi entri yang sesuai, jika tidak, request akan di-reject
  - leader akan mengulang dengan entri dengan indeks yg lebih kecil



	1	2	3	4
leader:	1 x←3	1 q←8	2 x←q	3 y←1
follower before:	1 x←3	1 q←8	2 x←q	
follower after:	1 x←3	1 q←8	2 x←q	3 y←1

Example #1: success

	1	2	3	4	5
leader:	1 x←3	1 q←8	2 x←q	3 y←1	
follower before:	1 x←3	1 q←8	1 j←2	1 y←6	1 a←x
follower after:	1 x←3	1 q←8	1 j←2	1 y←6	1 y←6

Example #2: mismatch

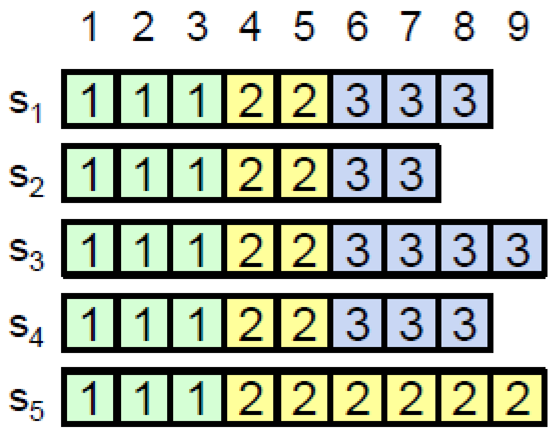
	1	2	3	4	5
leader:	1 x←3	1 q←8	2 x←q	3 y←1	
follower before:	1 x←3	1 q←8	1 j←2	1 y←6	1 a←x
follower after:	1 x←3	1 q←8	2 x←q	3 y←1	

Example #3: success

Safety: leader completeness

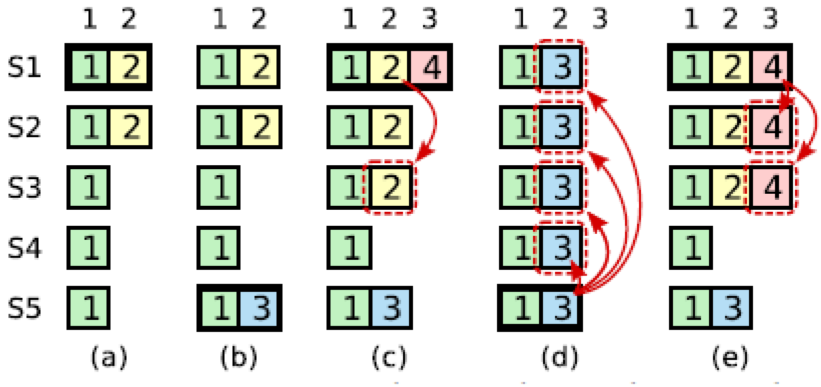
- Jika sebuah entri committed, semua leader berikutnya harus menyimpan entri tersebut
- server dengan log incomplete tidak boleh dipilih
  - kandidat menyertakan index dan term dari last entri pada RequestVote RPC
  - voting server menolak vote jika log yang dimiliki lebih uptodate
  - log diranking berdasarkan <lastTerm, lastIndex>

Leader election for term 4:

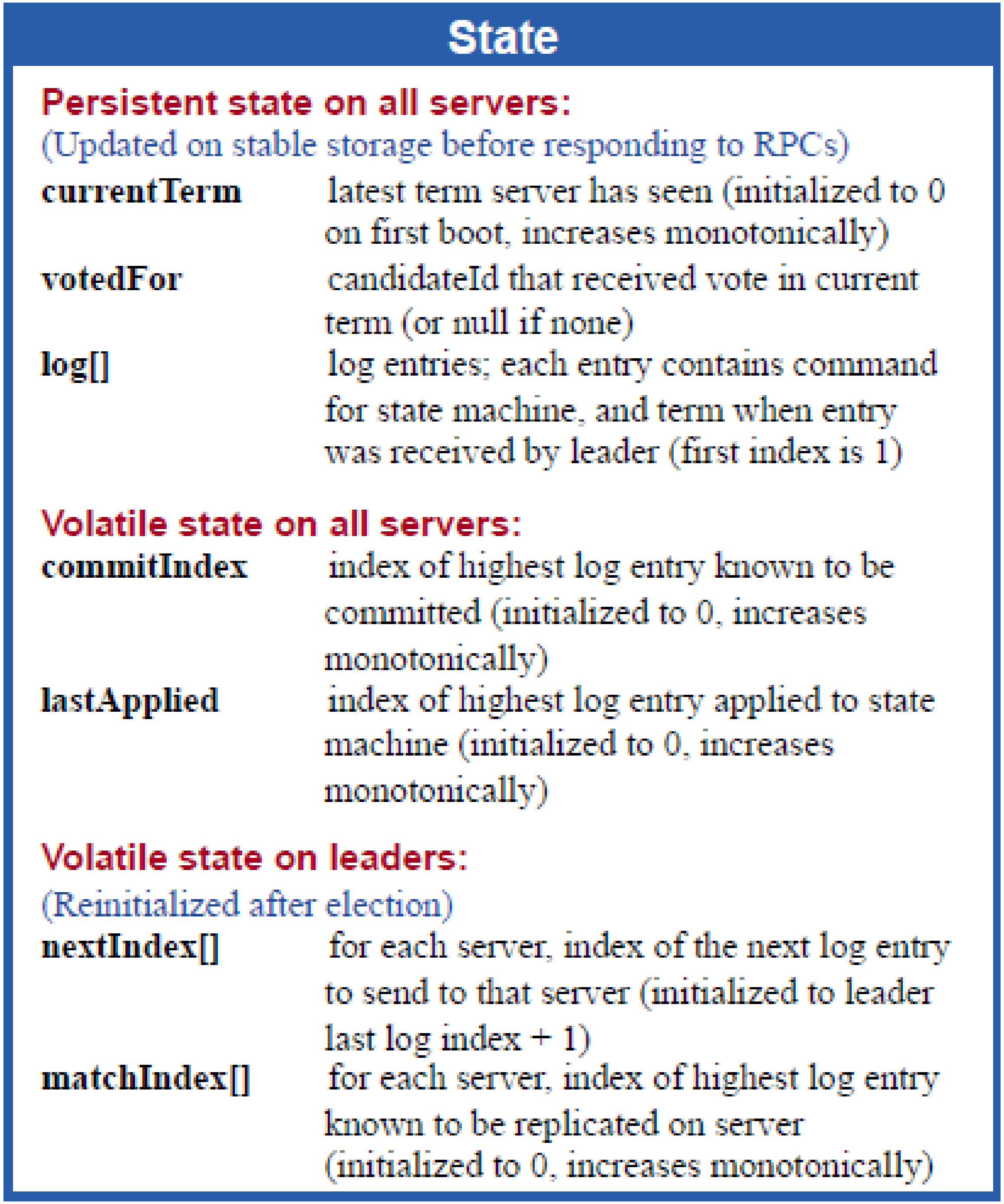


	1	2	3	4	5	6	7	8	9
s_1	1	1	1	2	2	3	3	3	
s_2	1	1	1	2	2	3	3		
s_3	1	1	1	2	2	3	3	3	3
s_4	1	1	1	2	2	3	3	3	
s_5	1	1	1	2	2	2	2	2	2

Contoh Kasus



- a) S1 leader
- b) S1 crashes, S5 leader
- c) S5 crash, S1 leader
- d) S1 crash sebelum menambahkan committed entri
- e) S1 crash setelah menambahkan committed entri



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)

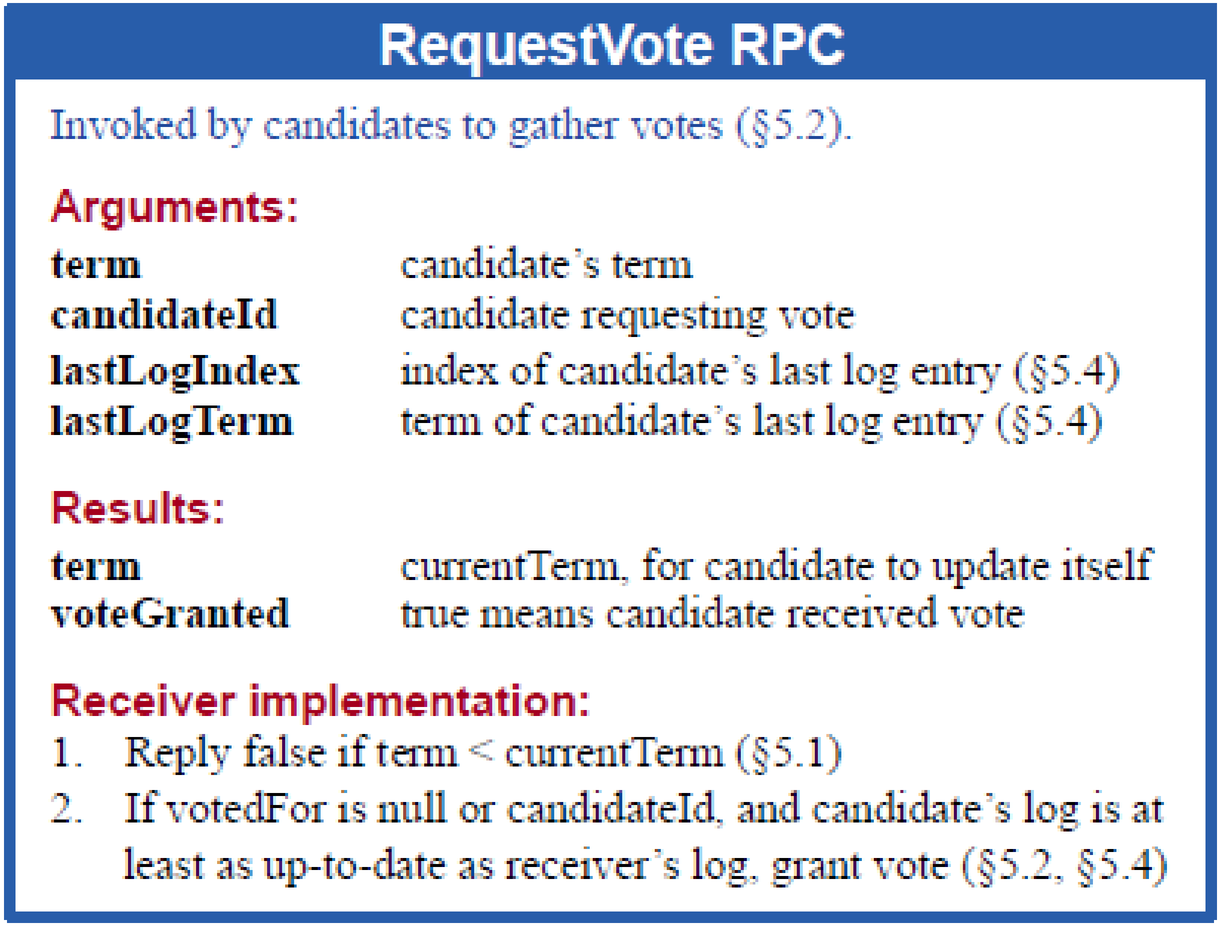
lastApplied index of highest log entry applied to state machine (initialized to 0, increases monotonically)

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)



RequestVote RPC

Invoked by candidates to gather votes (§5.2).

Arguments:

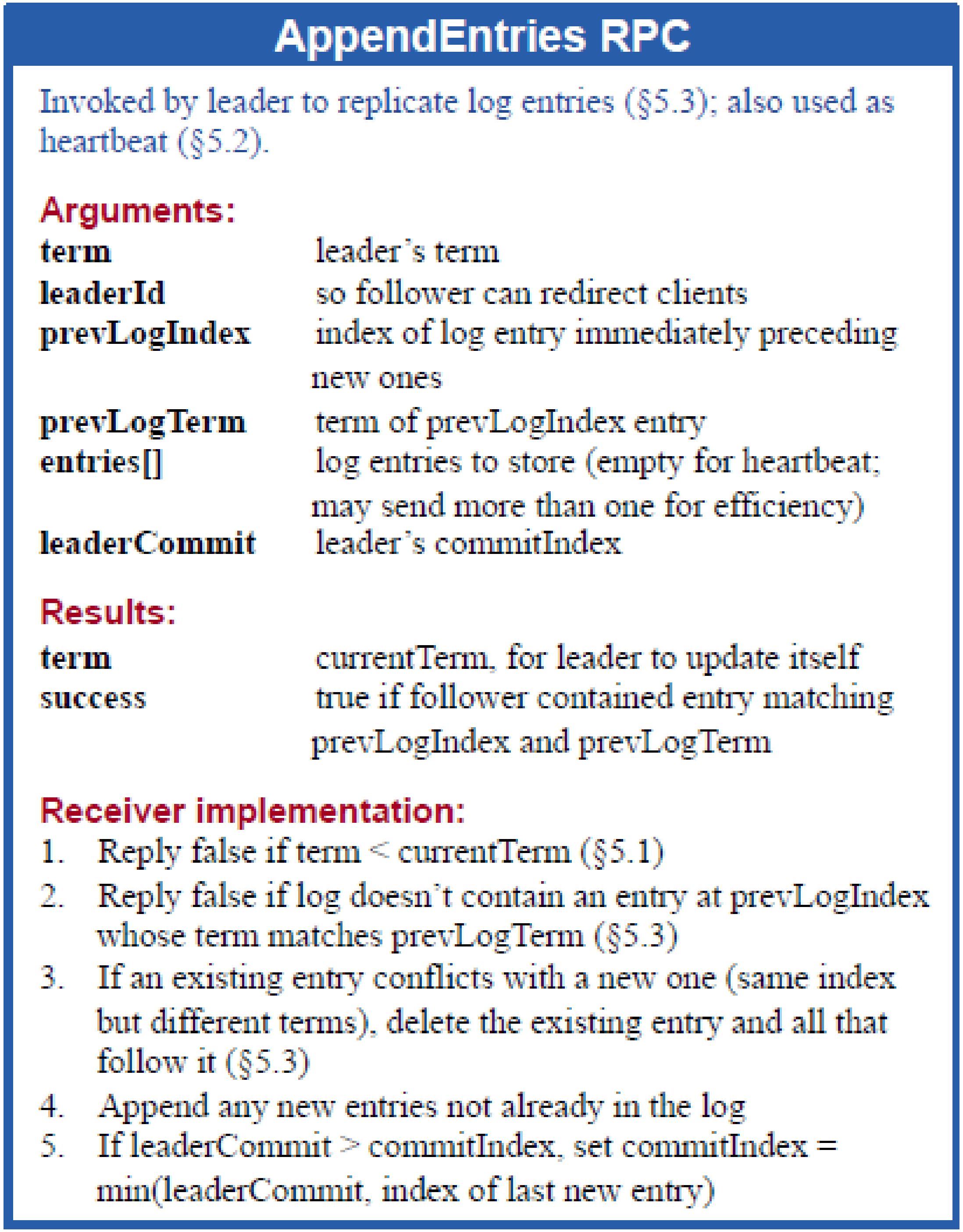
term	candidate's term
candidateId	candidate requesting vote
lastLogIndex	index of candidate's last log entry (§5.4)
lastLogTerm	term of candidate's last log entry (§5.4)

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)



AppendEntries RPC

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

Arguments:

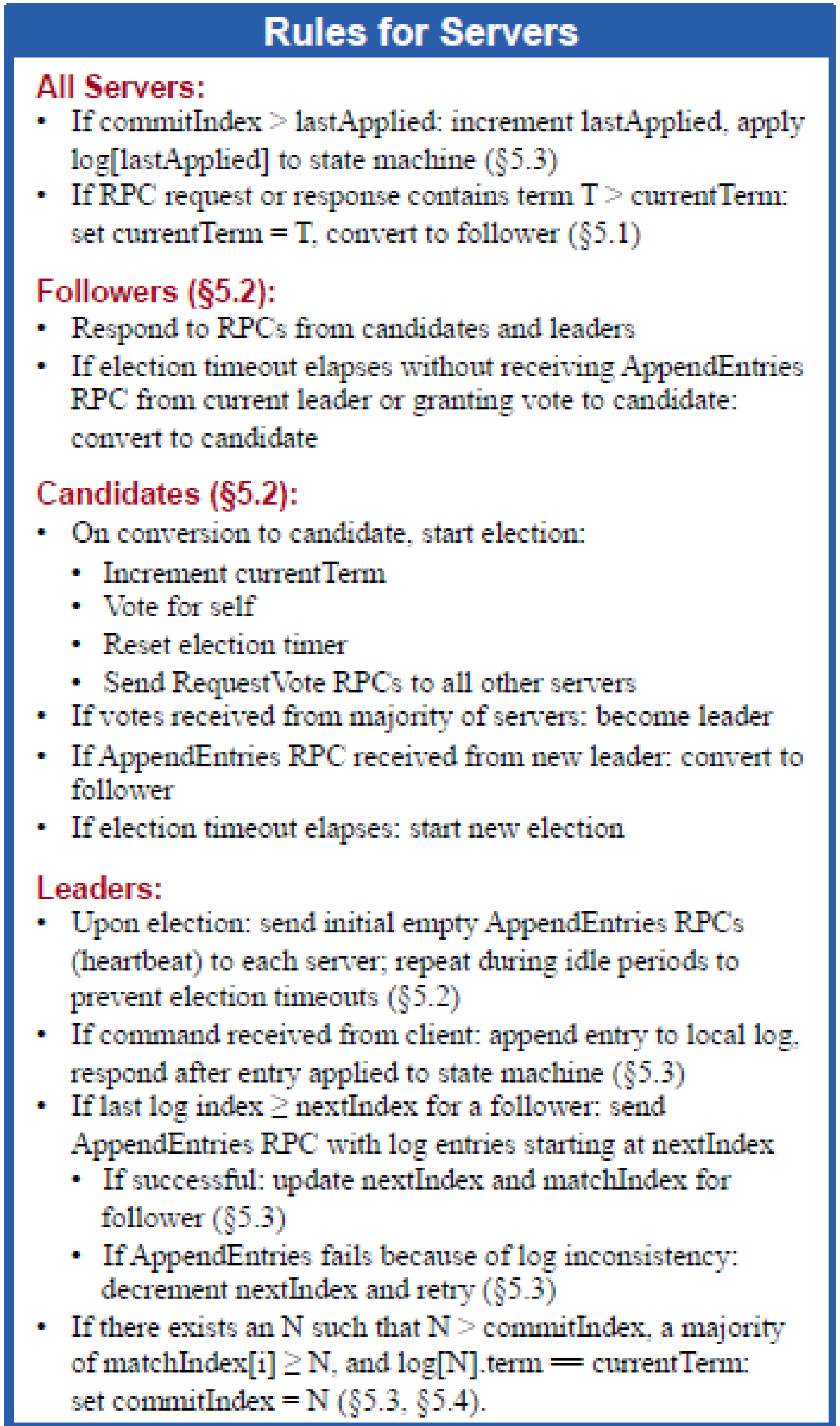
term	leader’s term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit	leader’s commitIndex

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)



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).