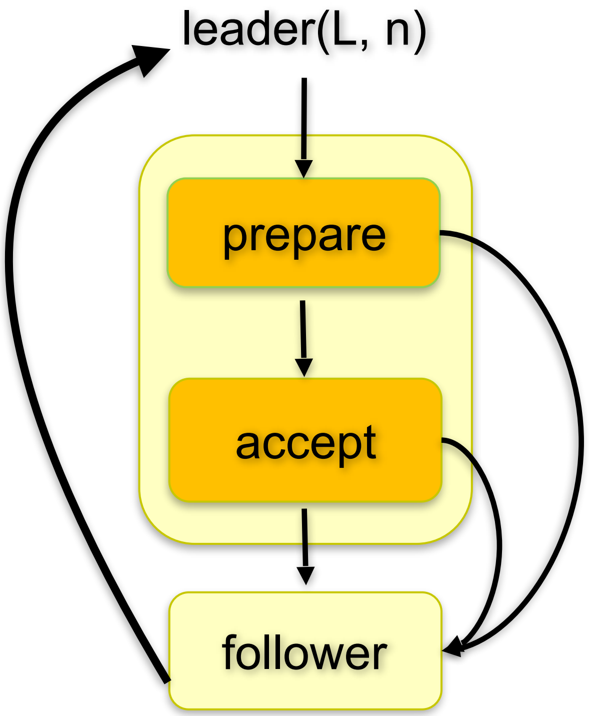
Leader- Based Sequence Paxos

Assumptions

- Assume eventual leader election abstraction with a ballot number BLE ⟨**Leader**, L, n⟩
  - BLE satisfies completeness and eventually accuracy
  - And also monotonically unique ballots
- The Leader-based Sequence Paxos is optimized for the case when a single proposer runs for a longer period of time as a leader
  - Thus, will not be aborted for a while
  - But must guarantee safety if aborted

The state of proposers

- We still have a set of proposers
- Any proposer will be either a leader or a follower
- A leader may be in either:
  - Prepare state, or
  - Accept state
- Until overrun by a higher leader, and moves to a follower state



Prepare phase

- **On** ⟨Propose, C⟩ :
  - n_p := unique higher proposal number
  - S := ∅, acks := 0
  - **send** ⟨Prepare, n_p⟩ **to** all acceptors
- **On** ⟨Promise, n, n', v'⟩ **s.t.** n = n_p:
  - add (n', v') to S (multiset union)
  - **if** |S|= ⌈(N+1)/2⌉:
    - (k, v) := max(S) **// adopt v**
    - v_p := if v ≠ ⊥ then v else C
    - **v_p := v ⊕ ⟨C⟩**
    - **send** ⟨Accept, n_p, v_p⟩ **to** all acceptors
- **On** ⟨Accepted, n⟩ **s.t.** n = n_p:
  - acks := acks + 1
  - **if** acks = ⌈(N+1)/2⌉:
    - **send** ⟨Decide, v_p⟩ **to** all learners

Accept phase

- **On** ⟨**Prepare**, n⟩:
  - **if** n_{prom} < n:
    - n_{prom} := n
    - **send** ⟨Promise, n, n_a, v_a⟩ **to** Proposer
- **On** ⟨**Accept**, n, v⟩:
  - **if** n_{prom} ≤ n:
    - n_{prom} := n
    - (n_a, v_a) := (n, v)
    - **send** ⟨Accepted, n⟩ **to** Proposer

Learner

- **On** ⟨**Decide**, v⟩:
  - **If $|v_d|$ < |v|:**
    - v_d := v
    - **trigger** Decide(v_d)

max(S) is any element (k, v) of S s.t. k is highest proposal number and v is a sequence

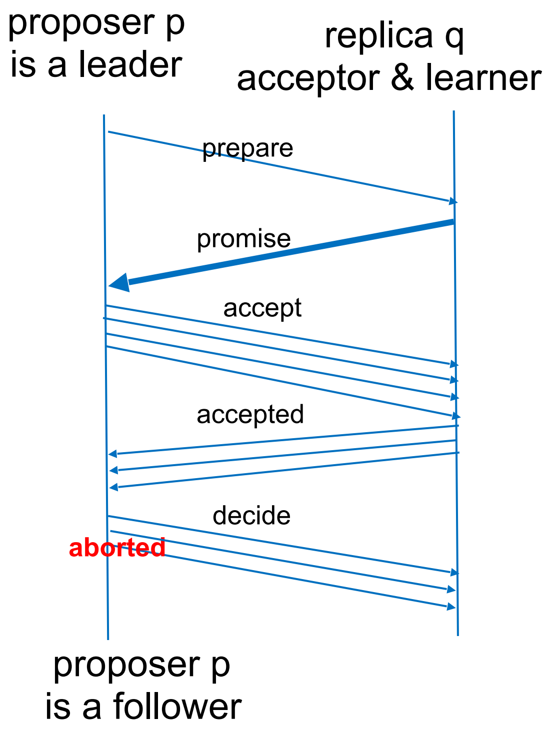
Prepare once and Pipeline Accept

Solution outline

- Current Sequence-Paxos is inefficient:
  - With multiple concurrent proposers, conflicts and restarts are likely (higher load → more conflicts)
  - 2 rounds of messages for each value chosen (Prepare, Accept)

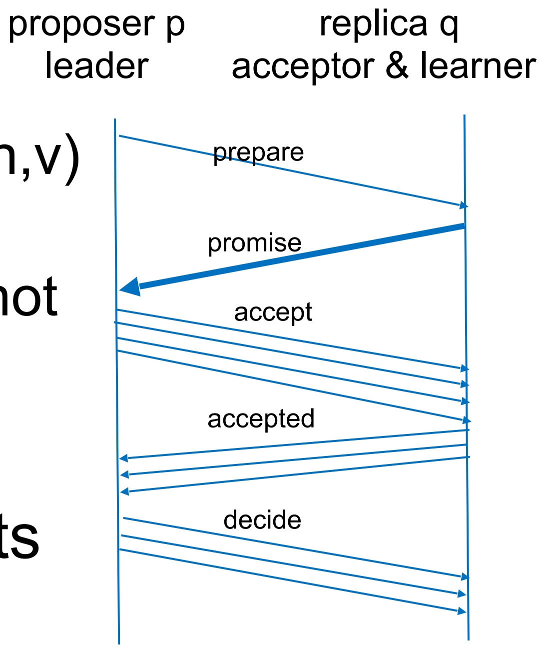
Solution:

- Pick a Leader(L, n) where n is a unique higher round number (leader election algorithm)
- The Leader acts as sole Proposer for round n
- After first Prepare (if not aborted) only perform Accepts until aborted by another Leader(n’), where n’ > n



Prepare Once, Pipeline Accept

- Benefit:
  - Proposer does prepare(n) before first-accept(n,v)
  - After that only one round-trip to decide on an extension of sequence v, as long as round is not aborted
  - (new leader with higher number)
- Allows multiple outstanding accept requests (pipelining)
  - Lower propose-to-decide latency for multiple proposals



Chosen Sequence at round n

- **Sequence v is chosen in round n** if acceptors in a majority set have accepted (in round n) sequences having v as a prefix

Round	Accepted by p_1	Accepted by p_2	Accepted by p_3
n = 5	$\langle C_2,C_3,C_1\rangle$ $\langle C_2,C_3\rangle$	$\langle C_2,C_3\rangle$	$\langle C_2,C_3,C_1,C_4\rangle$ $\langle C_2,C_3,C_1\rangle$ $\langle C_2,C_3\rangle$
...			
n=2		$\langle C_2\rangle$	$\langle C_2\rangle$
n=1	$\langle C_1\rangle$		
n=0	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$

- $\langle C_2,C_3,C_1\rangle$ and all its prefixes are chosen in round 5

Prepare Once, Pipeline Accepts

- After first Prepare
  - Allow issuing and accepting multiple proposals in round n
- We have now multiple (values) v's issued in the same round n?
- Acceptor accepts longer sequences in the same round n as long as $n \geq n_{prom}$ (acceptor's promise)

Prepare at round n, Proposer (Leader) behavior

- Proposer **p** becomes a leader with round **n** (By a leader election algorithm)
  - At this state n is the highest known proposal number
  - But **p** might be aborted by a leader with higher number m > n
  - **n** is **unique**, since only one leader is elected with a given round number **n**, **n** is higher than the rounds of previous leaders
- After successful completion of prepare phase the leader has the sequence v_0, and following invariant holds
  - The longest chosen sequence at any lower round **m < n** is a prefix of v_0 (quorum property guarantee)

Chosen Sequence at round n

- **Sequence v is chosen in round n** if acceptors in a majority set have accepted (in round n) sequences having v as a prefix

Round	Accepted by p_1	Accepted by p_2	Accepted by p_3
n = 5	$\langle C_2,C_3,C_1\rangle$ $\langle C_2,C_3\rangle$	$\langle C_2,C_3\rangle$	$\langle C_2,C_3,C_1,C_4\rangle$ $\langle C_2,C_3,C_1\rangle$ $\langle C_2,C_3\rangle$
…			
n=2		$\langle C_2,C_3\rangle$	$\langle C_2\rangle$
n=1	$\langle C_1\rangle$		
n=0	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$

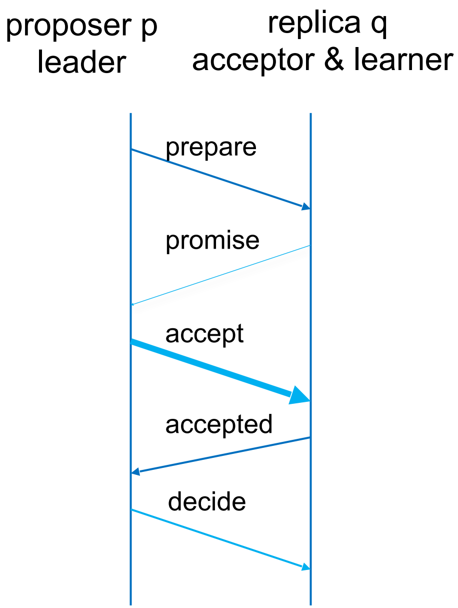
- $\langle C_2,C_3,C_1\rangle$ and all its prefixes are chosen in round 5

Accepts in round n, Proposer behavior

- A proposer (leader) issues multiple proposals in round n extending v_0
  - (n, v_0), (n,v_1), (n,v_2), ...
  - Proposer guarantees that $v_0 < v_1 < v_2 < \ldots$
  - Doesn't have to wait for one proposal to be chosen before the next is issued
  - Continues until aborted

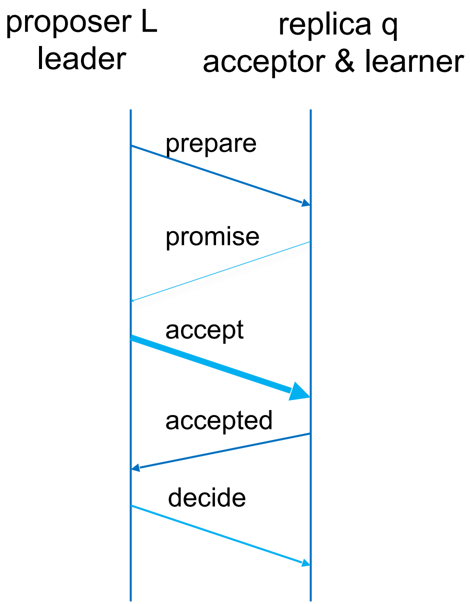
Accepts in round n, Acceptor behavior

- We order proposals in the following way:
  - (n, v) < (n’, v’) iff n < n’ or (n = n’ and |v|<|v’|)
- An acceptor extends its accepted sequence when it receives a new proposal
  - As long as it is a higher proposal according to the ordering above
- Accepted messages include the length of accepted values
  - Since multiple outstanding accept/accepted requests can be delivered out of order



Accepts in round n, Acceptor behavior

- Let $v_{a,q} = v_a$ at acceptor q, and $v_{p,L} = v_p$ at a leader L
- After q has accepted a proposal sent by L, it must be the case that $v_{a,q} \leq v_{p,L}$
  - It is enough for q to send back $|v_{a,q}|$
  - The proposer L can recreate $v_{a,q}$ from its $v_{p,L}$ as prefix($v_{p,L}$, $|v_{a,q}|$)



- **on** ⟨Accept, n, v⟩ from p:
  - **if** $n_{prom} \leq n$:
  - $n_{prom} := n$
  - $(n_a, v_a) :=$ max((n_a, v_a), (n, v))
  - **send** ⟨Accepted, n, $|v_a|$⟩ **to** p

Deciding on Sequences

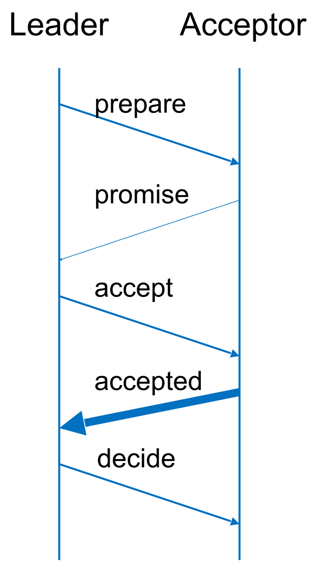
Proposer behavior upon Accepted

- Proposer maintains in *las*[p] the length of longest sequence accepted by acceptor p

- Sequence v is *chosen*
  - If for a majority of acceptors p: *las*[p] ≥ |v|
- If v is longer than previous sequence and chosen:
  - v is Decided and learners notified

Proposer behavior upon Accepted

- rename v_p to v_L the current extended proposed sequence
- At round n_L any value accepted by an acceptor a is a prefix of v_L
- A leader L, maintains l_c :
  - l_c is the length of the longest sequence that L knows is chosen (initially 0)
- On ⟨Accepted, n, m⟩ from a, $n = n_L$:
  - las[a] := max(m, las[a])
  - **if** prefix(v_L, m) is chosen **and** $l_c < m$:
  - l_c := m
  - send ⟨Decide, prefix(v_L, m) ⟩ to learners



Our leader-based Sequence Paxos

Initial State for Sequence Paxos

- Proposers
  - $n_L = 0$, $v_L =$ Leader's current round number, proposed value
  - propCmds = ⟨⟩ Leader's current set of proposed commands (empty set)
  - las = $[0]^N$ Length of longest accepted sequence per acceptor
  - $l_c = 0$ Length of longest chosen sequence
  - state = {(leader, prepare), (leader, accept), follower}
- Acceptor
  - $n_{prom} = 0$ Promise not to accept in lower rounds
  - $n_a = 0$ Round number in which a value is accepted
  - $v_a = \langle\rangle$ Accepted value (empty sequence)
- Learner
  - $v_d = \langle\rangle$ Decided value (empty sequence)

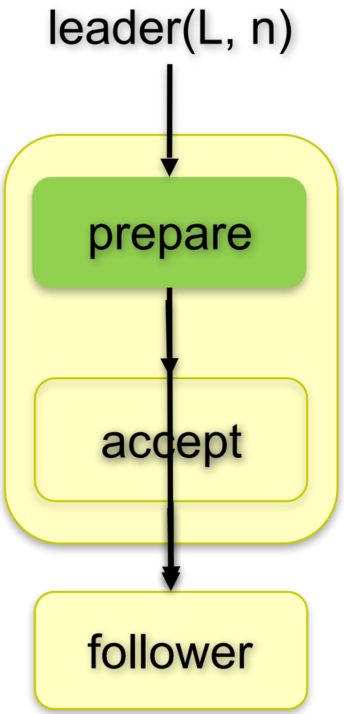
Leader Initiation & Prepare Phase

- **On** ⟨**Leader**, L, n⟩:
  - **if** self = L **and** n > n_L:
    - S := ∅; state := (leader, prepare)
    - propCmds = ⟨⟩; (v_L, n_L) := (⟨⟩, n)
    - las := $[0]^N$, l_c := 0
    - **send** ⟨**Prepare**, n_L⟩ **to** all acceptors
  - **else:** (state, leader) := (follower, L)

- **On** ⟨**Propose**, C⟩ **and.** state = (leader, prepare)
  - propCmds := propCmds ⊕ ⟨C⟩

- **On** ⟨**Promise**, n, n_a, v_a⟩ **s.t.** n = n_L **and** state = (leader, prepare)
- add (n_a, v_a) to S
- If |S|= ⌈(N+1)/2⌉:
- (k, v) := max(S) // **adopt v**
- v_L = v ⊕ propCmds; propCmds = ∅
- **send** ⟨Accept, n_L, v_L⟩ **to** all acceptors
- state := (leader, accept)

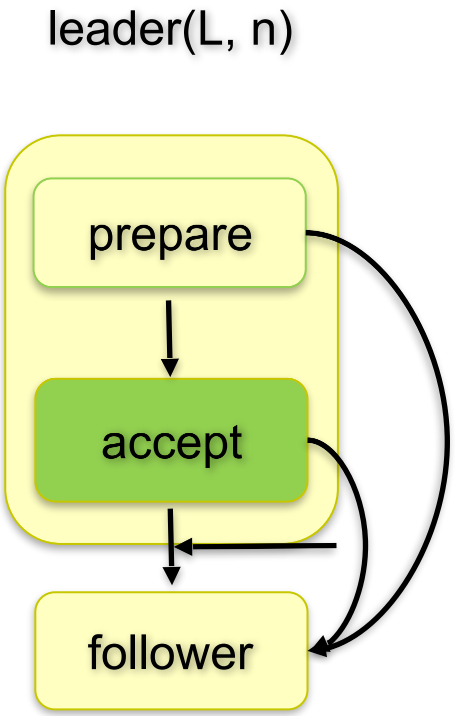
Proposer is in synch with majority of acceptors



Leader Accept Phase

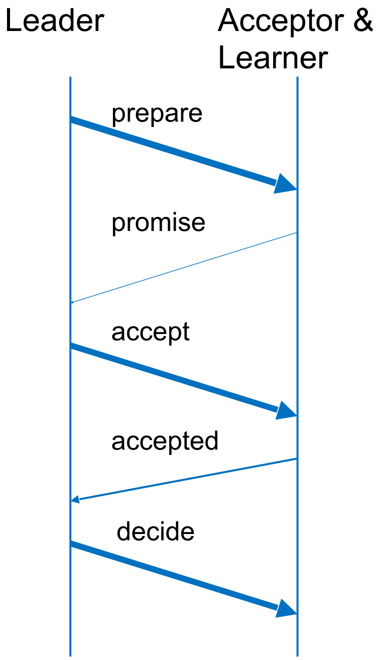
- **On** ⟨**Propose**, C⟩ **and** state = (leader, accept)
  - $v_L := v_L \oplus \langle C \rangle$
  - **send** ⟨**Accept**, n_L, v_L ⟩ **to** all acceptors

- **On** ⟨**Accepted**, n, m⟩ from a, **and** n = n_L **and** state = (leader, accept)
  - las[a] := max(m, las[a])
  - **If** l_c < m **and** prefix(v_L,m) is chosen:
  - l_c := m,
  - **send** ⟨**Decide**, prefix(v_L, m) ⟩ **to** all learners



Acceptor and Learner behavior

- **On** ⟨**Prepare**, n_p ⟩ **from** (a leader) p:
  - **if** $n_{prom} < n_p$:
  - $n_{prom} := n_p$
  - **send** ⟨**Promise**, n_p, n_a, v_a⟩ **to** p
- **On** ⟨**Accept**, n_p, v⟩ **from** (a leader) p:
  - **If** $n_{prom} \leq n_p$:
  - $n_{prom} := n_p$
  - $(n_a, v_a) := \max((n_a, v_a), (n_p, v))$
  - **send** ⟨**Accepted**, n, $|v_a|$ ⟩ **to** p
- **On** ⟨**Decide**, v⟩:
  - If $|v_d| < |v|$:
  - $v_d := v$
  - **trigger** Decide(v_d)



Leader

- **On** ⟨**Leader**, L, n⟩:
  - **if** self = L **and** $n > n_L$:
    - S := ∅, state := (leader, prepare)
    - propCmds = ⟨⟩; (v_L, n_L) := (⟨⟩, n)
    - las := $[0]^N$, $l_c := 0$
    - **send** ⟨**Prepare**, n_L⟩ **to** all acceptor
  - **else:** state, leader := follower, L
- **On** ⟨**Promise**, n, n_a, v_a⟩ **s.t.** $n = n_L$ **and** state = (leader, prepare)
  - add (n_a, v_a) to S
  - **if** |S|= ⌈(N+1)/2⌉:
    - (k, v) := max(S) // **adopt v**
    - v_L = v + propCmds; propCmds = ∅
    - **send** ⟨**Accept,** n_L, v_L⟩ **to** all acceptors
    - state := (leader, accept)
- **On** ⟨**Propose**, C⟩ **and** state = (leader, accept)
  - $v_L = v_L \oplus$ ⟨C⟩
  - **send** ⟨**Accept**, n_L, v_L ⟩ **to** all acceptors
- **On** ⟨**Propose**, C⟩ **and** state = (leader, prepare)
  - propCmds := propCmds + ⟨C⟩
- **On** ⟨**Accepted**, n, m⟩ from a, **and** $n = n_L$ and state = (leader, accept)
  - las[a] := max(las[a], m)
  - **If** l_c < m and prefix(v_L,m) is supported:
    - l_c := m,
    - **send** ⟨**Decide**, prefix(v_L, m) ⟩ **to** all learners

Acceptor

- **On** ⟨**Prepare**, n_p ⟩ **from** (a leader) p:
  - **if** $n_{prom} < n_p$:
    - $n_{prom} := n_p$
    - **send** ⟨**Promise**, n_p, n_a, v_a⟩ **to** p
- **On** ⟨**Accept**, n_p, v⟩ **from** (a leader) p:
  - **If** $n_{prom} \leq n_p$:
    - $n_{prom} := n_p$
    - (n_a, v_a) := max((n_a, v_a), (n_p, v))
    - **send** ⟨**Accepted**, n, $|v_a|$ ⟩ **to** p

Learner

- **On** ⟨**Decide**, v⟩:
  - If $|v_d| < |v|$:
    - $v_d := v$
    - **trigger** Decide(v_d)

Correctness
Leader Based Algorithm

Correctness

- We must guarantee that:
  - If proposal (n, v) is chosen, then for every higher proposal (n’, v’) that is chosen, v ≤ v’
- We have two cases:
  - n = n’: only successively longer sequences can become chosen within the same round since acceptors accept growing sequences
  - n < n’: the prepare phase guarantees that all chosen sequences in round n will be adopted in round n’, and no new sequences can be chosen in round n after that

Performance

- At this point, the algorithm
  - Pipelines of proposals for each proposer (leader) until losing leader role
  - Only first proposal requires two round-trips once a proposer becomes a leader
- What remains
  - v_L, v_a and v_d are mostly redundant
  - Entire sequences are sent back and forth
- We fix these in the next unit

Removing redundancy of v_L, v_a and v_d

Assumptions so far

  - *A1*: Optimized for the case when a single proposer runs for a longer period of time (leader)
- We add a new assumption
  - *A2:* Each process acts in all roles as proposer, acceptor and learner (replicated state machines)
  - Proposers have access to is own v_a and v_d
  - Acceptors know what is decided v_d

Removing V_L

- The leader **p** has access to its own v_a
- When p becomes a leader, it is possible to remove the need to store the sequences v_L and v_a separately at the leader
- By updating the local replica (acceptor) directly instead of sending a prepare message to itself it is possible to merge v_L into v_a
- At this state when p gets ⟨**Leader**, L, n⟩ and L = p:
  - $n > n_{(prom\ at\ p)}$
  - Hence ⟨**Promise**, n, $n_{(a\ at\ p)}$, $v_{(a\ at\ p)}$⟩ is unnecessary
- From now on the leader is extending his v_a

Leader

- **On** ⟨**Leader**, L, n⟩:
  - **if** **self** = L **and** n > n_L:
    - propCmds = ⟨⟩, **(n_L, n_{prom}) := (n, n)**
    - **S := { (n_a, v_a) }**, state := (leader, prepare)
    - las := $[0]^N$, l_c := 0, leader := self
    - **send** ⟨**Prepare**, n_L⟩ **to all acceptor – { self }**
  - **else:** (state, leader) := (follower, L) abort()
- **On** ⟨**Promise**, n, n_a, v_a⟩ **s.t.** n = n_L **and** state := (leader, prepare)
  - add (n_a, v_a) to S
  - **if** |S|= ⌈(N+1)/2⌉:
  - (k, $\mathbf{v_a}$) := max(S) // **adopt v**
  - $\mathbf{v_a}$ = $\mathbf{v_a}$ ⊕ propCmds; propCmds = ⟨⟩
  - **send** ⟨**Accept,** n_L, $\mathbf{v_a}$ ⟩ **to** all acceptors
  - state := (leader, accept)
- **On** ⟨**Propose**, C⟩ **s.t.** state = (leader, accept)
  - $\mathbf{v_a}$ = $\mathbf{v_a}$ ⊕ ⟨C⟩
  - **send** ⟨**Accept**, n_L, $\mathbf{v_a}$ ⟩ **to** all acceptors
- **On** ⟨**Propose**, C⟩ **and** state = (leader, prepare)
  - propCmds := propCmds ⊕ ⟨C⟩
- **On** ⟨**Accepted**, n, m⟩ from a, **s.t.** n = n_L and state = accept
  - …..

Acceptor

- **On** ⟨**Prepare**, n_p ⟩ **from** (a leader) p:
  - **if** n_{prom} < n_p:
  - n_{prom} := n_p
  - **send** ⟨**Promise**, n_p, n_a, v_a⟩ **to** p
- **On** ⟨**Accept**, n_p, v⟩ **from** (a leader) p:
  - **If** $n_{prom} \le n_p$:
  - n_{prom} := n_p
  - (n_a, v_a) := max((n_a, v_a), (n_p, v))
  - **send** ⟨**Accepted**, n, $|v_a|$ ⟩ **to** p

Learner

- **On** ⟨**Decide**, v⟩:
  - If $|v_d| < |v|$:
  - v_d := v
  - **trigger** Decide(v_d)

Removing redundancy of v_a and v_d

Assumptions so far

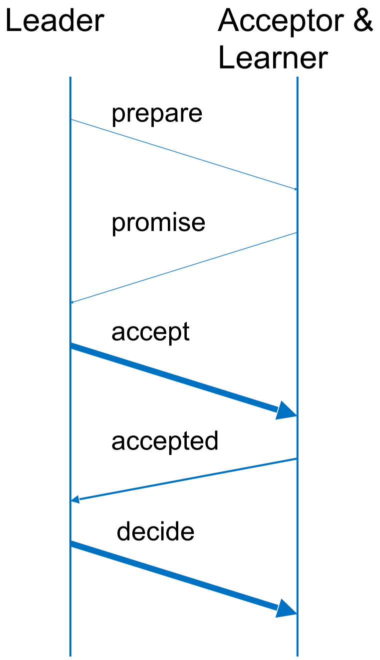
- ***A1***: Optimized for the case when a single proposer runs for a longer period of time (leader)
- ***A2****:* Each process acts in all roles as proposer, acceptor and learner (replicated state machines)
- We add a new assumption
  - ***A3:* FIFO Perfect Links**

The FIFO link assumption

- We assume FIFO Perfect Links (FPL)
  - This will be important for accepting commands incrementally
  - No performance penalties
    - Out of order commands has be buffered before decision
  - Not a too strong assumption in practice
    - In Fail-Silent model you get FPL from PL (Perfect Link) by adding sequence numbers
  - ZooKeeper makes this assumption too
  - If we implement Perfect Links on top of TCP then FIFO is more or less already provided during a session

Removing v_d

- Each replica stores both v_a and v_d, even though they are highly redundant
- Because of FIFO links:
  - At the **same round n** **accept** messages are delivered before corresponding **decide** messages from to any replica :
  - it always holds that at any replica q: $v_{(d\ at\ q)}$ is a prefix of $v_{(a\ at\ q)}$
- Sequence v_d can be replaced with an integer l_d, such that $\mathbf{v_d = prefix(v_a, \mathit{l}_d)}$



Leader

- **On** ⟨**Leader**, L, n⟩:
  - **if** **self** = L **and** n > n_L:
    - **S := { (n_a, v_a) }**, state := (leader, prepare)
    - …
    - **send** ⟨**Prepare**, n_L⟩ **to all acceptor – { self }**
  - **else:** (state, leader) := (follower, L)
- **On** ⟨**Promise**, n, n_a, v_a⟩ **s.t.** n = n_L **and** state := (leader, prepare):
  - …
- **On** ⟨**Propose**, C⟩ **s.t.** state = (leader, accept)
  - $\mathbf{v_a}$ = $\mathbf{v_a}$ + ⟨C ⟩
  - **send** ⟨**Accept**, n_L, $\mathbf{v_a}$ ⟩ **to** all acceptors
- **On** ⟨**Propose**, C⟩ **s.t.** state = (leader, prepare)
  - propCmds := propCmds + ⟨C⟩
- **On** ⟨**Accepted**, n, m⟩ from a, **s.t.** n = n_L and state = (leader, accept)
  - las[a] := max(las[a], m)
  - **If** l_c < m and prefix($\mathbf{v_a}$,m) is supported:
  - l_c := m,
  - **send** ⟨**Decide**, prefix($\mathbf{v_a}$, m), $\mathbf{n_L}$ ⟩ **to** all learners

Acceptor

- **On** ⟨**Prepare**, n_p ⟩ **from** (a leader) p:
  - **if** n_{prom} < n_p:
  - n_{prom} := n_p
  - **send** ⟨**Promise**, n_p, n_a, v_a⟩ **to** p
- **On** ⟨**Accept**, n_p, v⟩ **from** (a leader) p:
  - **If** n_{prom} ≤ n_p :
  - n_{prom} := n_p
  - (n_a, v_a) := max((n_a, v_a), (n_p, v))
  - **send** ⟨**Accepted**, n, |v_a| ⟩ **to** p

Learner

- **Initially l_d is 0**
- **On** ⟨**Decide**, v, **n**⟩:
  - If l_d < |v| **and n_{prom} = n**:
  - l_d := |v|
  - **trigger** Decide(prefix(v_a , l_d))

Avoid sending sequences

Leader

- **On** ⟨**Leader**, L, n⟩:
  - **if** **self** = L **and** n > n_L:
    - **S := { (n_a, v_a) }**, state := (leader, prepare)
    - propCmds = ∅, **(n_L, n_{prom}) := (n, n)**
    - las := $[0]^N$, l_c := 0, leader := self
    - **send** ⟨**Prepare**, n_L⟩ **to all acceptor – { self }**
  - **else:** (state, leader) := (follower, L)
- **On** ⟨**Promise**, n, n_a, v_a⟩ **s.t.** n = n_L and state = …prepare…
  - add (n_a, v_a) to S
  - **if** |S|= ⌈(N+1)/2⌉:
  - (k, $\mathbf{v_a}$) := max(S) // **adopt v**
  - $\mathbf{v_a}$ = $\mathbf{v_a}$ + propCmds; propCmds = ∅
  - **send** ⟨**Accept,** n_L, $\mathbf{v_a}$ ⟩ **to** all acceptors
  - state := (leader, accept)
- **On** ⟨**Propose**, C⟩ **s.t.** state = …accept..
  - $\mathbf{v_a}$ = $\mathbf{v_a}$ + ⟨C ⟩
  - **send** ⟨**Accept**, n_L, $\mathbf{v_a}$ ⟩ **to** all acceptors
- **On** ⟨**Propose**, C⟩ **s.t.** state = …prepare..
  - propCmds := propCmds + ⟨C⟩
- **On** ⟨**Accepted**, n, m⟩ from a, **s.t.** n = n_L and state = …
  - las[a] := max(las[a], m)
  - **If** l_c < m and prefix($\mathbf{v_a}$,m) is supported:
  - l_c := m,
  - **send** ⟨**Decide**, prefix($\mathbf{v_a}$, m), $\mathbf{n_L}$ ⟩ **to all learners**

Acceptor

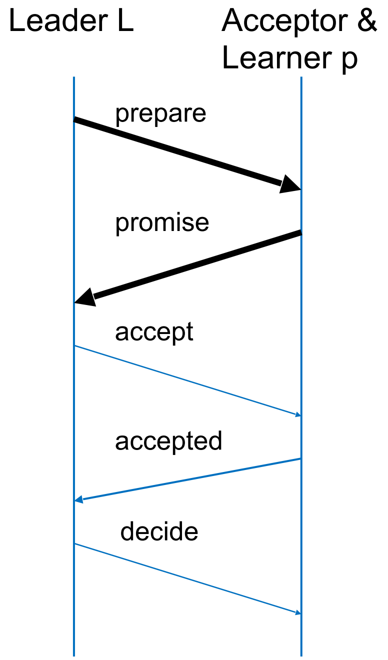
- **On** ⟨**Prepare**, n_p ⟩ **from** (a leader) p:
  - **if** n_{prom} < n_p:
  - n_{prom} := n_p
  - **send** ⟨**Promise**, n_p, n_a, v_a⟩ **to** p
- **On** ⟨**Accept**, n_p, v⟩ **from** (a leader) p:
  - **If** n_{prom} ≤ n_p :
  - n_{prom} := n_p
  - (n_a, v_a) := max((n_a, v_a), (n_p, v))
  - **send** ⟨**Accepted**, n, |v_a| ⟩ **to** p

Learner

- **Initially** l_d **is 0**
- **On** ⟨**Decide**, v, **n**⟩:
  - If l_d < |v| **and n_{prom} = n**:
  - l_d = |v|
  - **trigger** Decide(prefix(v_a , l_d))

Idea of Trim Promise

- Leader L sends a Prepare message to replica p that responds with a Promise msg
- Promise message **currently** contains entire sequence v_a at p
- But L knows that the sequence that will eventually by adopted by all replicas is an extension of v_d at L
- Changes:
  - Prepare message at L includes $(l_d = |v_d|, n_a)_{\text{at L}}$
  - Promise message includes either
    - $(n_a, \text{suffix}(v_a, l_d))_p$ if $n_{a \text{ at p}} \geq n_{a \text{ at L}}$
    - $(n_a, \langle\rangle)_p$ if $n_{a \text{ at p}} < n_{a \text{ at L}}$



- Proposer reconstructs the adopted sequence using max()

Leader at round 3 p1 leader

- If $\mathbf{p_1}$ becomes a leader at 3
  - Its decided sequence is $\langle C_1 \rangle$
    - $(n = 1, \text{suffix} = \langle A, B, D \rangle)_{p1}$
  - p_1 consults a majority, itself and either p_2 or p_3 by sending $|\langle C_1 \rangle|$
    - p_2 sends $(n = 2, \text{suffix} = \langle C_2, C_3 \rangle)_{p2}$
    - p_3 sends $(n = 2, \text{suffix} = \langle C_2 \rangle)_{p3}$
  - If p_2 consulted: $v_{a,p1} = \langle C_1 \rangle + \langle C_2, C_3 \rangle$ and extended locally by $\langle E, F, G \rangle$
    - $v_{a,p1} = \langle C_1, C_2, C_3, E, F, G \rangle$

Round	Accepted by p_1	Accepted by p_2	Accepted by p_3
n = 3	$\langle C_1, C_2, C_3, E, F, G \rangle$		
n = 2		$\langle C_1, C_2, C_3 \rangle$	$\langle C_1, C_2 \rangle$
n = 1	$\langle C_1, A, B, D \rangle$	$\langle C_1 \rangle$	
n = 0	$\langle \rangle$	$\langle \rangle$	$\langle \rangle$

Leader at round 3 p1 leader

- If $\mathbf{p_1}$ becomes a leader at 3
  - Its decided sequence is $\langle C_1 \rangle$
    - $(n = 1, \text{suffix} = \langle A, B, D \rangle)_{p1}$
  - p_1 consults a majority, itself and either p_2 or p_3 by sending $|\langle C_1 \rangle|$
    - p_2 sends $(n = 2, \text{suffix} = \langle C_2, C_3 \rangle)_{p2}$
    - p_3 sends $(n = 2, \text{suffix} = \langle C_2 \rangle)_{p3}$
  - If p_3 consulted: $\langle C_2 \rangle$ is added to $\langle C_1 \rangle$ extended locally by $\langle E, F, G \rangle$
    - $v_{a,p1} = \langle C_1, C_2, E, F, G \rangle$

Round	Accepted by p_1	Accepted by p_2	Accepted by p_3
n = 3	$\langle C_1, C_2,, E, F, G \rangle$		
n = 2		$\langle C_1, C_2, C_3 \rangle$	$\langle C_1, C_2 \rangle$
n = 1	$\langle C_1, A, B, D \rangle$	$\langle C_1 \rangle$	
n = 0	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$

Leader at round 3 p3 leader

- If $\mathbf{p_3}$ becomes a leader at 3
  - Its decided sequence is $\langle C_1, C_2\rangle$
    - $(n_a = 2, \text{suffix} = \langle\rangle)_{p3}$
  - p_3 consults a majority, itself and either p_1 or p_2 by sending $(|v_d| = |\langle C_1, C_2\rangle|, n_a=2)$
    - p_1 sends $(n_a = 1, \text{suffix} = \langle\rangle)_{p1}$
    - p_2 sends $(n_a = 2, \text{suffix} = \langle C_3\rangle)_{p2}$
  - If p_1 consulted: $v_{a,p3} = \langle C_1, C_2\rangle + \langle\rangle$ and extended locally by $\langle E, F, G\rangle$
  - $v_{a,p3} = \langle C_1, C_2, E, F, G\rangle$

Round	Accepted by p_1	Accepted by p_2	Accepted by p_3
n = 3			$\langle C_1, C_2, E, F, G\rangle$
n = 2		$\langle C_1, C_2, C_3\rangle$	$\langle C_1, C_2\rangle$
n = 1	$\langle C_1, A, B, D\rangle$	$\langle C_1\rangle$	
n = 0	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$

Leader at round 3 p3 is a leader

- If $\mathbf{p_3}$ becomes a leader at 3
  - Its decided sequence is $\langle C_1, C_2\rangle$
    - $(n_a = 2, \text{suffix} = \langle\rangle)_{p3}$
  - p_3 consults a majority, itself and either p_1 or p_2 by sending $(|v_d| = |\langle C_1, C_2\rangle|, n_a=2)$
    - p_1 sends $(n_a = 1, \text{suffix} = \langle\rangle)_{p1}$
    - p_2 sends $(n_a = 2, \text{suffix} = \langle C_3\rangle)_{p2}$
  - If p_2 consulted: $\langle C_3\rangle$ is added to $\langle C_1\ C_2\rangle$ and extended locally by $\langle E, F, G\rangle$
  - $v_{a,p3} = \langle C_1, C_2, C_3, E,F,G\rangle$

Round	Accepted by p_1	Accepted by p_2	Accepted by p_3
n = 3			$\langle C_1,C_2, C_3\ E, F, G\rangle$
n = 2		$\langle C_1,C_2, C_3\rangle$	$\langle C_1,C_2\rangle$
n = 1	$\langle C_1,A, B, D\rangle$	$\langle C_1\rangle$	
n = 0	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$

Leader at round 3 p2 is a leader

- If $\mathbf{p_2}$ becomes a leader at 3
  - Its decided sequence is $\langle C_1, C_2\rangle$
    - $(n_a = 2, \text{suffix} = \langle C_3 \rangle)_{p3}$
  - p_2 consults a majority, itself and either p_1 or p_3 by $(|v_d| = |\langle C_1, C_2\rangle|, n_a=2)_{p2}$
    - p_1 sends $(n_a = 1, \text{suffix} = \langle\rangle)_{p1}$
    - p_3 sends sends $(n_a = 2, \text{suffix} = \langle\rangle)_{p2}$
  - If p_1 consulted: $v_{a,p2} = \langle C_1, C_2\rangle + \langle C_3 \rangle$ and extended locally by $\langle E, F, G\rangle$
  - $v_{a,p2} = \langle C_1, C_2, C_3, E, F, G\rangle$
  - If p_2 consulted: $v_{a,p2} = \langle C_1, C_2\rangle + \langle C_3 \rangle$ and extended locally by $\langle E, F, G\rangle$
  - $v_{a,p2} = \langle C_1, C_2, C_3, E, F, G\rangle$

Round	Accepted by p_1	Accepted by p_2	Accepted by p_3
n = 3		$\langle C_1, C_2, C_3\ E, F, G\rangle$	
n = 2		$\langle C_1, C_2, C_3\rangle$	$\langle C_1, C_2\rangle$
n = 1	$\langle C_1, A, B, D\rangle$	$\langle C_1\rangle$	
n = 0	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$

Implementation

- **On** ⟨**Leader**, L, n⟩:
  - **if** **self** = L **and** n > n_L:
    - leader := self, state := prepare
    - S := {(n_a ,suffix(v_a , l_d)) },
    - propCmds = ⟨⟩, (n_L, n_{prom}) := (n, n)
    - las := $[0]^N$, l_c := 0, leader := self
    - **send** ⟨**Prepare**, n_L, l_d, n_a⟩ **to** all acceptor – { self }
  - **else:** (state, leader) := (follower, L) abort()

- **On** ⟨**Prepare**, n_p , ld, n⟩ **from** (a leader) p:
  - **if** n_{prom} < n_p:
  - n_{prom} := n_p
  - suffx := **if** $n_a \geq n$: suffix($v_{a,}$, ld) **else** ⟨⟩
  - **send** ⟨**Promise**, n_p, n_a, suffx⟩ **to** p

Implementation

- **On** $\langle$**Promise**, n, n_a, suffx$_a\rangle$ **s.t.** n = n_L and state = prepare
  - add (n_a, suffx$_a$) to S
  - **if** |S|= ⌈(N+1)/2⌉:
  - (k, suffx) := max(S) // **adopt v**
  - $\mathbf{v_a}$ = prefix($\mathbf{v_a}$, l_d) + suffx + propCmds;
  - propCmds = $\langle\rangle$
  - **send** $\langle$**Accept,** n_L, $\mathbf{v_a}\rangle$ **to** all acceptors
  - state := accept

- S = {(n1, v1), …., (n_k,vk)}
- **fun** max(S):
  - (n,v) =: (0,$\langle\rangle$)
  - **for** (n',v') **in** S:
    - **if** n < n' **or** (n = n' **and** |v| < |v'|):
    - (n,v) := (n',v')
  - **return** (n,v)

- **On** $\langle$**Prepare**, n_p , *ld*, n$\rangle$ **from** (a leader) p:
  - **if** n_{prom} < n_p:
  - n_{prom} := n_p
  - suffx := **if** $n_a \geq n$: suffix($v_{a,}$, *ld*) **else** $\langle\rangle$
  - **send** $\langle$**Promise**, n_p, n_a, suffx$\rangle$ **to** p

Leader

- On ⟨**Leader**, L, n⟩:
  - **if** **self** = L **and** n > n_L:
    - leader := self, state := …**prepare**…
    - S := {(n_a ,suffix(v_a , l_d)) },
    - propCmds = ⟨⟩, (n_L, n_{prom}) := (n, n)
    - $l$as := $[0]^N$, l_c := 0, leader := self
    - **send** ⟨**Prepare**, n_L, l_d, n_a ⟩ **to** all acceptor – { self }
  - **else:** (state, leader) := (follower, L)
- **On** ⟨**Promise**, n, n_a, suffx$_a$⟩ **s.t.** n = n_L and state = **prepare…**
  - add (n_a, suffx$_a$) to S
  - **if** |S|= ⌈(N+1)/2⌉:
  - (k, suffx) := max(S) // **adopt v**
  - $\mathbf{v_a}$ = prefix($\mathbf{v_a}$, l_d) + suffx + propCmds;
  - propCmds = ∅
  - **send** ⟨**Accept,** n_L, $\mathbf{v_a}$ ⟩ **to** all acceptors
  - state := …**accept…**
- **On** ⟨**Propose**, C⟩ **s.t.** state = **…accept..**
  - $\mathbf{v_a}$ = $\mathbf{v_a}$ + ⟨C ⟩
  - **send** ⟨**Accept**, n_L, $\mathbf{v_a}$ ⟩ **to** all acceptors
- **On** ⟨**Propose**, C⟩ **s.t.** state = …**prepare..**
  - propCmds := propCmds + ⟨C⟩
- **On** ⟨**Accepted**, n, m⟩ from a, **s.t.** n = n_L and state = **…accept.**
  - las[a] := max(las[a], m)
  - **If** l_c < m and prefix($\mathbf{v_a}$,m) is supported:
  - l_c := m,
  - **send** ⟨**Decide**, prefix($\mathbf{v_a}$, m), $\mathbf{n_L}$ ⟩ **to** all learners

Acceptor

- **On** ⟨**Prepare**, n_p , ld, n⟩ **from** (a leader) p:
  - **if** n_{prom} < n_p:
  - n_{prom} := n_p
  - suffx := **if** n_a ≥ n : suffix(v_a, ld) **else** ⟨⟩
  - **send** ⟨**Promise**, n_p, n_a, suffx⟩ **to** p
- **On** ⟨**Accept**, n_p, v⟩ **from** (a leader) p:
  - **If** n_{prom} ≤ n_p :
  - n_{prom} := n_p
  - (n_a, v_a) := max((n_a, v_a), (n_p, v))
  - **send** ⟨**Accepted**, n, |v_a| ⟩ **to** p

Learner

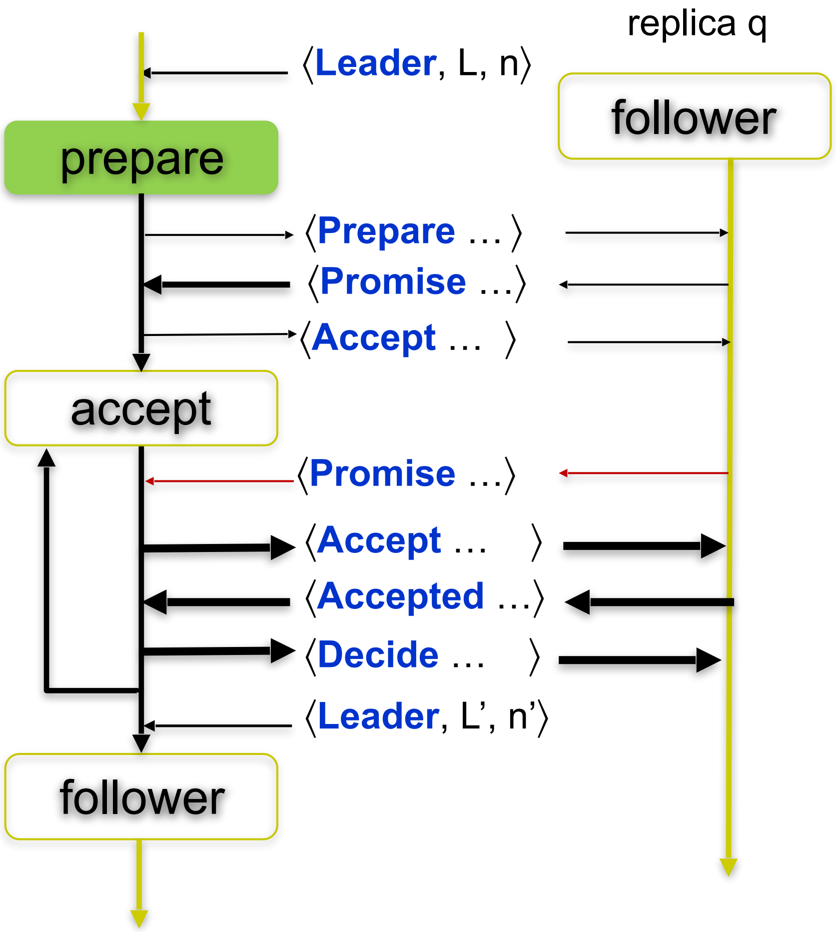
- **Initially** l_d **is 0**
- **On** ⟨**Decide**, v, **n**⟩:
  - If l_d < |v| **and** $\mathbf{n_{prom}}$ **= n**:
  - l_d = |v|
  - **trigger** Decide(prefix(v_a , l_d))

The Accept phase

The first Accept AcceptSync

First Accept

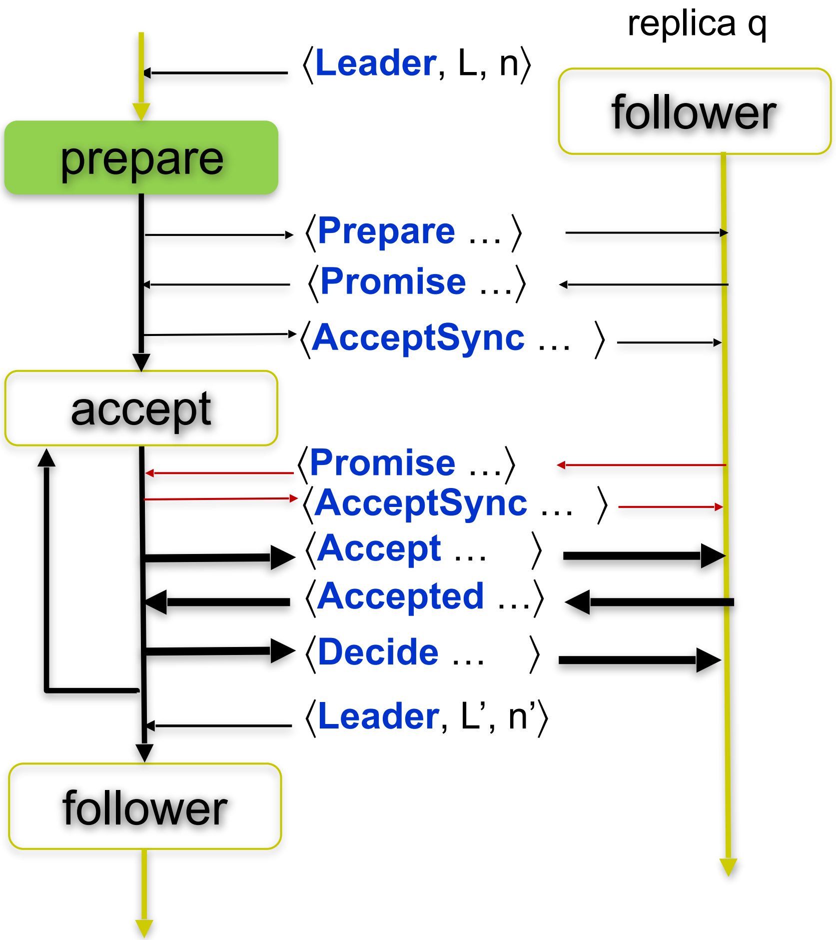
- After getting Promise messages from a majority, The leader L updates the state of its accepted sequence v_a
- Leader needs to update the accepted sequence v_a's of the replicas
- We have two cases
  - Replica q_i from which L received a promise message in **state prepare**
  - Replicas q_i from which L received a promise message in **state accept**
- In both cases the leader needs to know the length of decided sequence at each replica



AcceptSync

- In both cases the first accept is special
- It synchronizes the state of the replicas to reflect the state of the leader

- We call the first Accept AcceptSync
- We extend the state of a follower to distinguish the first accept from subsequent accepts
  - (follower, ⊥) initially
  - (follower, prepare) after **Prepare** message
  - (follower, accept) after **AcceptSync** message



AcceptSync, leader in prepare state

- Leader L has acquired the knowledge of the length of decided sequence from a majority of replicas through promise messages
  - Each replica q sends the length of its decided sequence $l_{\text{d at q}}$ in the promise
  - Leader L reconstructs his own v_a
  - For each replica q in the majority: L sends an AcceptSync message suffix($v_{\text{a at L}}$, $l_{\text{d at q}}$) and $l_{\text{d at q}}$

Implementation

- **On** ⟨**Promise**, n, n_a, $suffx_a$, ld_a⟩ **from** a **s.t.** n = n_L and state = **…prepare…**
  - add (n_a, $suffx_a$) to S, lds[a] := ld_a
  - **if** |S|= ⌈(N+1)/2⌉:
  - (k, suffx) := max(S) // **adopt v**
  - v_a = prefix(v_a, l_d) + suffx + propCmds;
  - $l$as[self] := |v_a| /** selecting chosen sequence */
  - propCmds = ∅, state := (leader, accept)
  - **for** p **in** π- {self} **s.t.** lds[p] ≠ ⊥ :
  - **send** ⟨**AcceptSync,** n_L, suffix(v_a, lds[p]), lds[p]⟩ **to** p

- **On** ⟨**Prepare**, n_L , ld, n⟩ **from** (a leader) L:
  - **if** n_{prom} < n_L:
  - n_{prom} := n_L
  - state := (follower, prepare)
  - suffx := **if** n_a ≥ n : suffix(v_a, ld) **else** ⟨⟩
  - **send** ⟨**Promise**, n_L, n_a, suffx, l_d⟩ **to** p

Implementation

- **On** ⟨**Promise**, n, n_a, $suffx_a$, ld_a⟩ **from** a **s.t.** n = n_L and state = **(leader, prepare)**
  - add (n_a, $suffx_a$) to S, lds[a] := ld_a
  - **if** |S|= ⌈(N+1)/2⌉:
  - (k, suffx) := max(S) // **adopt v**
  - v_a = prefix(v_a, l_d) + suffx + propCmds;
  - $l$as[self] := |v_a| /** selecting chosen sequence */
  - propCmds = ∅**,** state := (leader, accept)
  - **for** p **in** π- {self} **s.t.** lds[p] ≠ ⊥ :
  - **send** ⟨**AcceptSync,** n_L, suffix(v_a, lds[p]), lds[p]⟩ **to** p

- **On** ⟨**AcceptSync**, n_L, suffxv, ld⟩ **from** L and state = (**follower**, **prepare**):
  - **If** n_{prom} = n_L :
  - n_a := n_L
  - v_a := prefix(v_a, ld) + suffxv
  - **send** ⟨**Accepted**, n_L, |v_a|⟩ **to** p
  - state = (**follower**, **accept**)

Leader at round 3

- If $\mathbf{p_1}$ becomes a leader at 3
  - Its decided sequence is $\langle C_1 \rangle$
    - ($n = 1$, suffix = $\langle A, B, D\rangle)_{p1}$
  - p_1 consults itself and p_2 by sending $|\langle C_1\rangle|$
    - p_2 sends ($n = 2$, suffix = $\langle C_2, C_3\rangle)_{p2}$, $l_{d,p2} = 2$
  - P_1 constructs $v_{a,p1} = \langle C_1\rangle + \langle C_2, C_3\rangle$ extended locally by $\langle E, F, G\rangle$
    - $v_{a,p1} = \langle C_1, C_2, C_3, E, F, G\rangle$
  - p_1 sends
    - suffix($v_{a,p1}$, $l_{d,p2}$) = $\langle C_3, E, F, G\rangle$
    - $l_{d,p2} = 2$
  - p_2 reconstructs its v_a at round 3
    - $v_{a,p2} = \langle C_1, C_2, C_3, E, F, G\rangle$

Round	Accepted by p_1	Accepted by p_2	Accepted by p_3
n = 3	$\langle C_1, C_2, C_3, E, F, G\rangle$	$\langle C_1, C_2, C_3, E, F, G\rangle$	
n = 2		$\langle C_1, C_2, C_3\rangle$	$\langle C_1, C_2\rangle$
n = 1	$\langle C_1, A, B, D\rangle$	$\langle C_1\rangle$	
n = 0	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$

Leader at round 3

- If $\mathbf{p_1}$ becomes a leader at 3
  - Its decided sequence is $\langle C_1 \rangle$
    - $(n = 1, \text{suffix} = \langle A, B, D \rangle)_{p1}$
  - p_1 consults a majority
  - If p_3 consulted: $v_{a,p1} = \langle C_1 \rangle + \langle C_2 \rangle$ and extended locally by $\langle E, F, G \rangle$
    - $v_{a,p1} = \langle C_1, C_2, E, F, G \rangle$
  - p_1 sends
    - $\text{suffix}(v_{a,p1}, l_{d,p3}) = \langle E, F, G \rangle$ +
    - $l_{d,p2} = 2$
  - p_3 reconstructs its v_a at round 3
    - $v_{a,p2} = \langle C_1, C_2, E, F, G \rangle$

Round	Accepted by p_1	Accepted by p_2	Accepted by p_3
n = 3	$\langle C_1, C_2, E, F, G \rangle$		$\langle C_1, C_2, E, F, G \rangle$
n = 2		$\langle C_1, C_2, C_3 \rangle$	$\langle C_1, C_2 \rangle$
n = 1	$\langle C_1, A, B, D \rangle$	$\langle C_1 \rangle$	
n = 0	$\langle \rangle$	$\langle \rangle$	$\langle \rangle$

Leader at round 3 p2 is leader

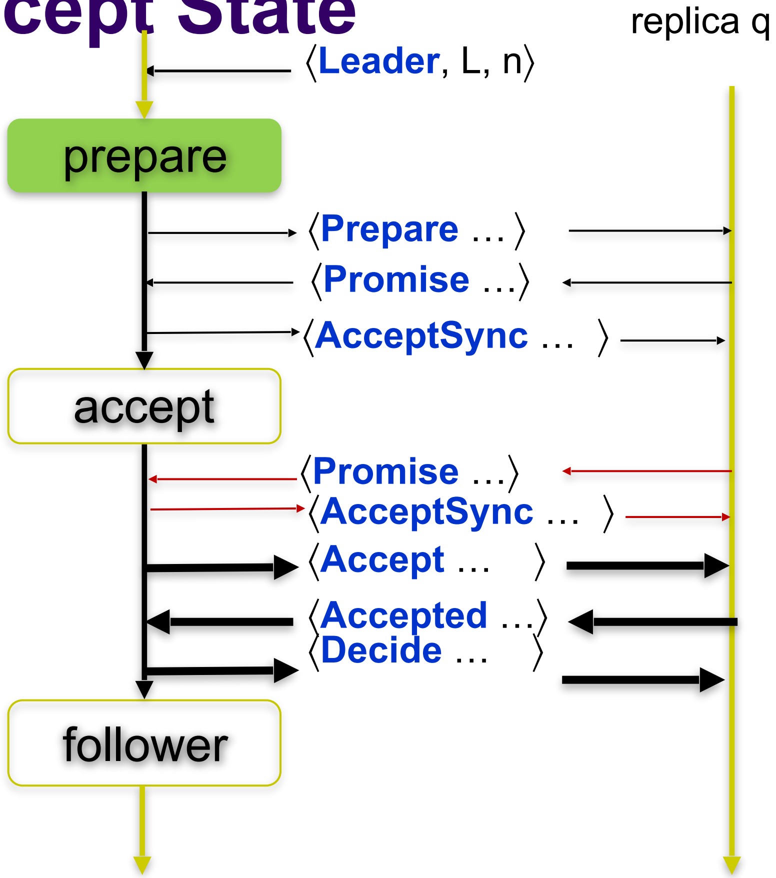
- If $\mathbf{p_2}$ becomes a leader at 3
  - Its decided sequence is $\langle C_1, C_2 \rangle$
    - $(n_a = 2, \text{suffix} = \langle\ C_3\ \rangle)_{p3}$
  - p_2 consults a majority, itself and either p_1 or p_3 by $(|v_d| = |\langle C_1, C_2 \rangle|, n_a{=}2)_{p2}$
    - p_1 sends $(n_a = 1, \text{suffix} = \langle\rangle\ )_{p1}$, $l_{d,p1} = 1$
  - p_2 sends to p_1
    - $\text{suffix}(v_{a,p1}, l_{d,p1}) = \langle\ C_2, C_3\ E,F,G \rangle$
    - $l_{d,p1} = 1$
  - p_1 reconstructs its v_a at round 3
    - $v_{a,p1} = \langle\ C_1, C_2, C_3\ , E, F, G \rangle$

Round	Accepted by p_1	Accepted by p_2	Accepted by p_3
n = 3	$\langle\ C_1, C_2, C_3\ E, F, G \rangle$	$\langle\ C_1, C_2, C_3\ E, F, G \rangle$	
n = 2		$\langle C_1, C_2, C_3 \rangle$	$\langle C_1, C_2 \rangle$
n = 1	$\langle C_1, A, B, D \rangle$	$\langle C_1 \rangle$	
n = 0	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$

Leader at the Accept Phase II

First Accept, leader in Accept State

- After getting Promise msgs from a majority, The leader L updates the state of its accepted sequence v_a
- Leader needs to update the accepted sequence v_a's of the replicas
- We have two cases
  - Replica q_i from which L received a promise message in **state prepare**
  - Replicas q_i from which L received a promise message in **state accept**
- In both cases the leader needs to know the length of decided sequence at each replica

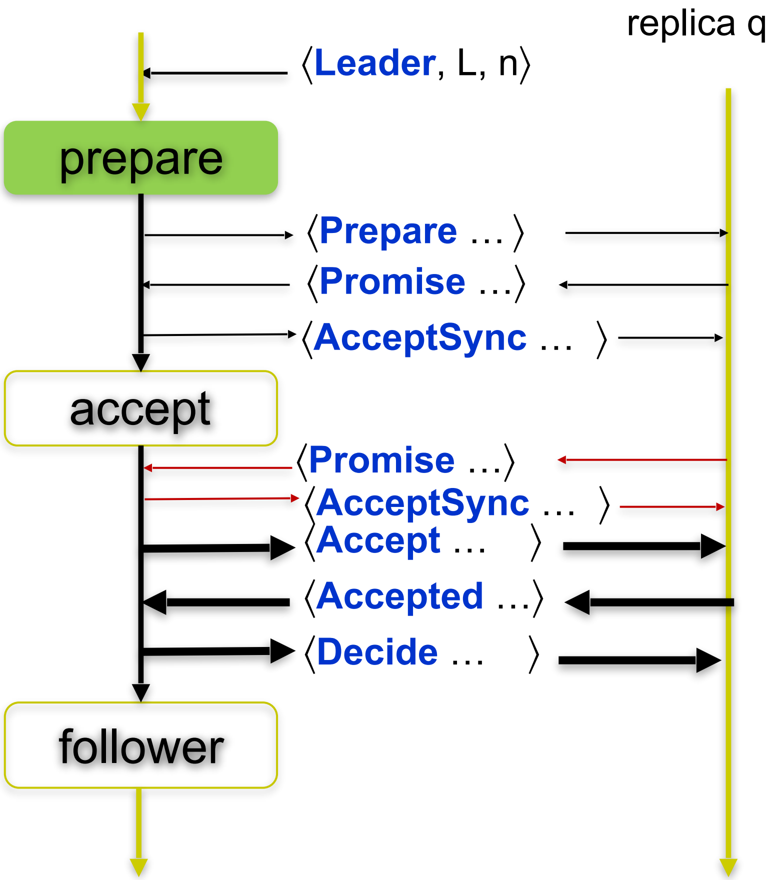


AcceptSync, leader in accept state

- Leader L receives a promise from replica q while in the accept state
  - Each replica q sends the length of its decided sequence $l_{d \text{ at } q}$ in the Promise
  - Leader has already reconstructed his sequence v_a
  - For each other replica q after receiving a promise, L sends an AcceptSync message:
    - suffix($v_{a \text{ at } L}$, $l_{d \text{ at } q}$) and $l_{d \text{ at } q}$
  - If some sequence is already decided it sends the decide index $l_{d \text{ at } L}$

AcceptSync, leader in accept state

- Other replicas
  - Leader L waits until it receives Promise msg from q before sending AcceptSync message to q
  - Receiving a promise synchronizes L's knowledge about q
  - Maintain invariant at q: $v_d \leq v_a$
  - L may not send Decide msg or subsequent Accept msgs to q until AcceptSync msg is sent to q
- If some sequence has been chosen before L received promise from q then L must send Decide msg to q **after** first Accept
  - This is indicated by $l_c \neq 0$: Length of longest chosen (learned) sequence



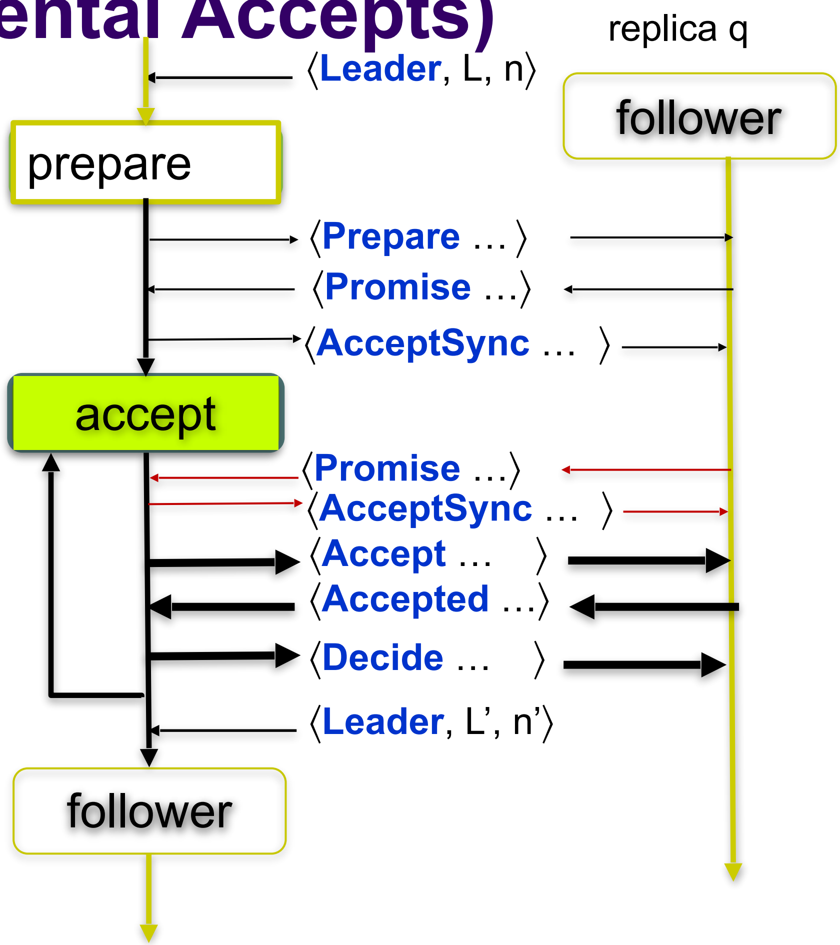
Implementation

- **On** ⟨**Promise**, n, n_a, $suffx_a$, ld_a⟩ **from** a **and** n = n_L **and** state = (leader, accept)
  - lds[a] := ld_a
  - **send** ⟨**AcceptSync**, n_L, suffix(v_a, lds[a]), lds[a]⟩ **to** a
  - if $l_c \neq 0$:
  - **send** ⟨**Decide**, l_d, $\mathbf{n_L}$⟩ **to** a

- **On** ⟨**AcceptSync**, n_L, suffxv, ld⟩ **from** L and state = (**follower**, **prepare**):
  - **If** $n_{prom} = n_L$:
  - $n_a := n_L$
  - v_a := prefix(v_a, ld) + suffxv
  - **send** ⟨**Accepted**, n_L, $|v_a|$⟩ **to** p
  - state = (**follower**, **accept**)

Updating replicas (incremental Accepts)

- Subsequent Accept messages:
  - Let $m_1 = \langle$Accept, $n_L, v_1\rangle$ and $m_2 = \langle$Accept, $n_L, v_2\rangle$, and m_1 is sent before m_2 from leader L to a replica q
  - L knows that at the time when q processes m_2, q will have accepted v_1, or blocked round n_L
    - Holds because of FIFO links
  - Therefore L will send vs = suffix(v_2, $|v_1|$) and offset = $|v_1|$ instead of v_2
    - In particular if $v_2 = v_1 + \langle C\rangle$: m_2 is $\langle$Accept, n_L, $\langle C\rangle$, $|v_1|\rangle$



Implementation

- When a leader L in the accept state gets a new command C
  - Updates its accepted sequence and its $l$as[L]
  - Sends Accept messages to all replicas that passed the **prepare** phase

- **On** ⟨**Propose**, C⟩ **and** state = (**leader, accept**)
  - $\mathbf{v_a} = \mathbf{v_a} \oplus \langle C \rangle$
  - $l$as[self] := $l$as[self] + 1
  - **for** p **in** π- {self} **s.t.** lds[p] ≠ ⊥ :
  - **send** ⟨**Accept,** n_L, ⟨C⟩⟩ **to** p
- A replica that moved to the **accept** phase will accept the command if leader is in the current round as the promise, extends its accepted sequence and acknowledges to the leader

- **On** ⟨**Accept**, n_L, ⟨C⟩ ⟩ **from** (a leader) L **and** state = (**follower**, **accept**)
  - **If** $n_{prom} = n_L$:
  - $v_a := v_a \oplus \langle C \rangle$
  - **send** ⟨**Accepted**, n_p, $|v_a|$ ⟩ **to** L

How to Decide

Implementation

- The leader maintains
- $las[0]$: the leader's knowledge of the longest accepted sequence per replica
- l_c : the longest learned sequence so far
- If m the length of the acknowledged sequence is greater than l_c , a majority of replicas responded : a longer sequence is chosen (supported)
- A decision is sent to all replicas in the accept phase

- **On** ⟨**Accepted**, n, m⟩ **from** a, **s.t.** n = n_L and state = (**leader**, **accept**)
  - $las[a]$:= m
  - **If** l_c < m **and** |{p **in** π : $las[a] \geq m$}| ≥ $\lceil (N+1)/2 \rceil$
  - l_c := m,
  - **for** p **in** π **s.t.** lds[p] ≠ ⊥
  - **send** ⟨**Decide**, l_c , $\mathbf{n_L}$ ⟩ **to** p

Deliver One Command At A Time

- Currently every decided sequence is handed to the application in its entirety
- It makes more sense to change the API and decide one command at a time

- **Initially** l_d **is 0 //** zero-based indexing

- **On** ⟨**Decide**, l, $\mathbf{n_L}$⟩:
  - **if** $n_{prom} = n_L$:
  - **while** $l_d < l$:
  - **trigger** Decide($v_a[l_d]$)
  - $l_d := l_d + 1$

Initially l_d **is 0**
On ⟨**Decide**, v, **n**⟩:
- **if** $l_d < |v|$ **and** $n_{prom} = n$:
- $l_d = |v|$
- **trigger** Decide(prefix(v_a , l_d))

The final algorithm

The final Sequence Paxos algorithm

- **The algorithm use**
  - BallotLeaderElection
  - FIFOPerfectPointToPointLinks
  - The algorithm works in the asynchronous model
  - but requires BLE which works in the partially synchronous model

Initial Replica for Sequence Paxos

- Leader specific
  - propCmds = $\langle\rangle$ Leader's current set of proposed commands (empty set)
  - las = $[0]^N$ Length of longest accepted sequence per acceptor
  - lds = $[\bot]^N$ Length of longest known decided sequence per acceptor
  - $l_c = 0$ Length of longest chosen (learned) sequence
  - acks = $[\bot]^N$ Promise acks per acceptor p $\longmapsto$ (n, v)
- Replica (including Acceptor and Learner)
  - (n_L, leader) = (0, $\bot$) Leader's current round number, leader process
  - state = ({**follower**, **leader**}, {**prepare**, **accept**, $\bot$}) initially (**follower**, $\bot$)
  - $n_{prom} = 0$ Promise not to accept in lower rounds
  - $n_a = 0$ Round number in which a value is accepted
  - $v_a = \langle\rangle$ Accepted value (empty sequence)
  - $l_d = 0$ Length of decided value (length of empty sequence)

Replicas

On ⟨**Leader**, L, n⟩:
  if $n > n_L$ **:**
    leader := L , n_L := n
    if **self** = L **and** $n_L > n_{prom}$ **:**
      state := (**leader**, **prepare**)
      propCmds = ⟨⟩; las := $[0]^N$; lds := $[\perp]^N$
      acks := $[\perp]^N$; l_c := 0,
      send ⟨**Prepare**, n_L, l_d, n_a⟩ **to all** π – { **self** }
      acks[L] := (n_a ,suffix(v_a , l_d))
      lds[**self**] := l_d; n_{prom}:= n_L
  else:
      state = (**follower**, state[2])

On ⟨**Prepare**, n_L , ld, n⟩ **from** L:
  if $n_{prom} < n_L$:
    n_{prom} := n_L ; state := (**follower**, **prepare**)
    suffx := **if** $n_a \geq n$: suffix(v_a, ld) **else** ⟨⟩
    send ⟨**Promise**, n_L, n_a, suffx, l_d⟩ **to** L

On ⟨**Promise**, n, n_a, suffx$_a$, ld⟩ **from** a
  s.t. $n = n_L$ **and** state = (**leader**, **prepare**):
  acks[a] := (n_a, suffx$_a$) , lds[a] := ld
  P := {p **in** π : acks[p] ≠ ⊥}
  if $|P| = \lceil (N+1)/2 \rceil$:
    (k, suffx) := max({acks[p]: p **in** P}) // **adopt v**
    v_a = prefix(v_a, l_d) + suffx + propCmds;
    las[**self**] := $|v_a|$
    propCmds := ⟨⟩; state := (**leader**, **accept**)
    for p **in** π- {**self**} **and** lds[p] ≠ ⊥**:**
      sufx := suffix(v_a, lds[p])
      send ⟨**AcceptSync**, n_L, sufx, lds[p]⟩ **to** p

On ⟨**Promise**, n, n_a, suffx$_a$, ld⟩ **from** a
  s.t. $n = n_L$ **and** state = (**leader**, **accept**)**:**
  lds[a] := ld
  send ⟨**AcceptSync**, n_L, suffix(v_a, lds[a]), lds[a]⟩ **to** a
  if $l_c \neq 0$:
    send ⟨**Decide**, l_d, $\mathbf{n_L}$⟩ **to** a

Replicas

On ⟨**AcceptSync**, n_L, sufx, ld ⟩ **from** p
  s.t. state = (**follower**, **prepare**):
  If $n_{prom} = n_L$:
    $n_a := n_L$
    $v_a :=$ prefix(v_a, ld) + sufx
    send ⟨**Accepted**, n_L, $|v_a|$ ⟩ **to** p
    state = (**follower**, **accept**)

On ⟨**Accept**, n_L, ⟨C⟩, ld ⟩ **from** p
  s.t. state = (**follower**, **accept**):
  If $n_{prom} = n_L$:
    $v_a := v_a$ + ⟨C⟩
    send ⟨**Accepted**, n_p, $|v_a|$ ⟩ **to** p

On ⟨**Decide**, l, $\mathbf{n_L}$⟩:
  if $n_{prom} = n_L$:
    while $l_d < l$:
      trigger Decide($v_a[l_d]$)
      $l_d := l_d + 1$

On ⟨**Propose**, C⟩
  s.t. state **=** (**leader**, **prepare**):
  propCmds := propCmds + ⟨C⟩

On ⟨**Propose**, C⟩
  s.t. state = (**leader**, **accept**):
  $\mathbf{v_a} = \mathbf{v_a}$ + ⟨C ⟩
  las[self] := las[self] + 1
  for p **in** π- {self} **s.t.** lds[p] ≠ ⊥ :
    send ⟨**Accept**, n_L, ⟨C⟩⟩ **to** p

On ⟨**Accepted**, n, m⟩ **from** a,
  s.t. n = n_L **and** state = (**leader**, **accept**) :
  las[a] := m
  If l_c < m **and** |{p **in** π : las[a] ≥ m}| ≥ ⌈(N+1)/2⌉ **:**
    l_c := m,
    for p **in** l **s.t.** lds[p] ≠ ⊥**:**
      send ⟨**Decide**, l_c , $\mathbf{n_L}$ ⟩ **to** p

The final Sequence Paxos algorithm

- We developed a complete, simple and efficient Sequence Paxos algorithm in the fail-silent model (asynchronous model) that creates a consistent replicated **log** v_a
- The algorithm guarantees the safety properties of sequence consensus as long as the following assumptions hold
  - FIFO perfect links
  - An eventual leader election abstraction that guarantees for any indication (response) event ⟨**Leader**, L, n⟩ the combination (L,n) is **unique** (same requirement as single value Paxos)

The final Sequence Paxos algorithm

- Most of the time once a command C is delivered to the leader, one round trip is needed for deciding on C
- For liveness (progress) the leader election should satisfy
  - For any process p: if p is elected by ⟨**Leader**, p, n⟩, then for any for previous event and process q:
    - ⟨**Leader**, q, n'⟩: n' < n should hold
  - A leader p should stay and be considered as a leader by a majority of processes "for a sufficient time" before overtaken by a higher numbered process
  - **No requirement** on strong accuracy on the leader election algorithm otherwise.