Leader-Based Sequence Paxos
Assumptions

- Assume **eventual leader election abstraction** with a **ballot number** $\langle \text{Leader}, L, n \rangle$
  - 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 \langle Propose, C \rangle:
  - \( n_p := \) unique higher proposal number
  - \( S := \emptyset \), \( \text{acks} := 0 \)
  - send \langle Prepare, n_p \rangle to all acceptors

- On \langle Promise, n, n', v' \rangle s.t. \( n = n_p \):
  - add \((n', v')\) to \( S \) (multiset union)
  - if \(|S| = \lceil (N+1)/2 \rceil \):
    - \((k, v) := \text{max}(S)\) // adopt \( v \)
    - \( v_p := \) if \( v \neq \bot \) then \( v \) else \( C \)
    - \( v_p := v \oplus \langle C \rangle \)
    - send \langle Accept, n_p, v_p \rangle to all acceptors

Accept phase

- On \langle Prepare, n \rangle:
  - if \( n_{\text{prom}} < n \):
    - \( n_{\text{prom}} := n \)
    - send \langle Promise, n, n_a, v_a \rangle to Proposer

- On \langle Accept, n, v \rangle:
  - if \( n_{\text{prom}} \leq n \):
    - \( n_{\text{prom}} := n \)
    - \( (n_a, v_a) := (n, v) \)
    - send \langle Accepted, n \rangle to Proposer

Learner

- On \langle Decide, v \rangle:
  - If \( |v_d| < |v| \):
    - \( v_d := v \)
    - trigger Decide(v_d)

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

S. Haridi, KTHx ID2203.2x
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

<table>
<thead>
<tr>
<th>Round</th>
<th>Accepted by $p_1$</th>
<th>Accepted by $p_2$</th>
<th>Accepted by $p_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 5</td>
<td>$\langle C_2, C_3, C_1 \rangle$</td>
<td>$\langle C_2, C_3 \rangle$</td>
<td>$\langle C_2, C_3, C_1, C_4 \rangle \atop \langle C_2, C_3, C_1 \rangle \atop \langle C_2, C_3 \rangle$</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n=2</td>
<td></td>
<td>$\langle C_2 \rangle$</td>
<td>$\langle C_2 \rangle$</td>
</tr>
<tr>
<td>n=1</td>
<td>$\langle C_1 \rangle$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n=0</td>
<td>$\langle \rangle$</td>
<td>$\langle \rangle$</td>
<td>$\langle \rangle$</td>
</tr>
</tbody>
</table>

- $\langle C_2, C_3, C_1 \rangle$ 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

<table>
<thead>
<tr>
<th>Round</th>
<th>Accepted by $p_1$</th>
<th>Accepted by $p_2$</th>
<th>Accepted by $p_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 5</td>
<td>$\langle C_2, C_3, C_1 \rangle$</td>
<td>$\langle C_2, C_3 \rangle$</td>
<td>$\langle C_2, C_3, C_1, C_4 \rangle$</td>
</tr>
<tr>
<td></td>
<td>$\langle C_2, C_3 \rangle$</td>
<td>$\langle C_2, C_3 \rangle$</td>
<td>$\langle C_2, C_3 \rangle$</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n=2</td>
<td></td>
<td>$\langle C_2, C_3 \rangle$</td>
<td>$\langle C_2 \rangle$</td>
</tr>
<tr>
<td>n=1</td>
<td>$\langle C_1 \rangle$</td>
<td>$\langle C_1 \rangle$</td>
<td>$\langle C_1 \rangle$</td>
</tr>
<tr>
<td>n=0</td>
<td>$\langle \rangle$</td>
<td>$\langle \rangle$</td>
<td>$\langle \rangle$</td>
</tr>
</tbody>
</table>

- $\langle C_2, C_3, C_1 \rangle$ 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 $v_0 < v_1 < v_2 < ...$
  - 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 $\text{prefix}(v_{p,L}, |v_{a,q}|)$

- **on** $\langle \text{Accept, n, v} \rangle$ from $p$:
  - if $n_{prom} \leq n$:
  - $n_{prom} := n$
  - $(n_a, v_a) := \max((n_a, v_a), (n, v))$
  - **send** $\langle \text{Accepted, n, |v_a|} \rangle$ to $p$
Deciding on Sequences
Proposer behavior upon Accepted

- Proposer maintains in $\text{las}[p]$ the length of longest sequence accepted by acceptor $p$

- Sequence $v$ is chosen
  - If for a majority of acceptors $p$: $\text{las}[p] \geq |v|$
  - 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 $\langle\text{Accepted}, n, m\rangle$ from $a$, $n = n_L$:
  - $\text{las}[a] := \max(m, \text{las}[a])$
  - if prefix($v_L$, m) is chosen and $l_c < m$:
    - $l_c := m$
    - send $\langle\text{Decide, prefix}(v_L, m)\rangle$ 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
  - \( \text{propCmds} = \langle \rangle \) Leader’s current set of proposed commands (empty set)
  - \( \text{las} = [0]^N \) Length of longest accepted sequence per acceptor
  - \( l_c = 0 \) Length of longest chosen sequence
  - \( \text{state} = \{(\text{leader, prepare}), (\text{leader, accept}), \text{follower}\} \)

- **Acceptor**
  - \( n_{\text{prom}} = 0 \) 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 \(\langle \text{Leader}, L, n \rangle\):
  - if \(\text{self} = L \text{ and } n > n_L\):
    - \(S := \emptyset\); state := \((\text{leader}, \text{prepare})\)
    - propCmds = \(\langle \rangle\); \((v_L, n_L) := (\langle \rangle, n)\)
    - \(\text{lS} := [0]^N, l_c := 0\)
    - send \(\langle \text{Prepare}, n_L \rangle\) to all acceptors
  - else: \((\text{state}, \text{leader}) := (\text{follower}, L)\)

- On \(\langle \text{Propose}, C \rangle\) and. state = \((\text{leader}, \text{prepare})\)
  - propCmds := propCmds \(\oplus \langle C \rangle\)

- On \(\langle \text{Promise}, n, n_a, v_a \rangle\) s.t. \(n = n_L\) and state = \((\text{leader}, \text{prepare})\)
  - add \((n_a, v_a)\) to \(S\)
  - If \(|S| = \lceil (N+1)/2 \rceil\):
    - \((k, v) := \max(S)\) // adopt \(v\)
    - \(v_L = v \oplus \text{propCmds}; \text{propCmds} = \emptyset\)
    - send \(\langle \text{Accept}, n_L, v_L \rangle\) to all acceptors
  - state := \((\text{leader}, \text{accept})\)
**Leader Accept Phase**

- On \(\langle\text{Propose}, C\rangle\) and state = (leader, accept)
  - \(v_L := v_L \oplus \langle C \rangle\)
  - send \(\langle\text{Accept}, n_L, v_L\rangle\) to all acceptors

- On \(\langle\text{Accepted}, n, m\rangle\) from \(a\), and \(n = n_L\) and state = (leader, accept)
  - \(\text{las}[a] := \max(m, \text{las}[a])\)
  - If \(l_c < m\) and \(\text{prefix}(v_L, m)\) is chosen:
    - \(l_c := m\),
    - send \(\langle\text{Decide}, \text{prefix}(v_L, m)\rangle\) to all learners
Acceptor and Learner behavior

- On \(\langle \text{Prepare}, n_p \rangle\) from (a leader) \(p\):
  - if \(n_{prom} < n_p\):
  - \(n_{prom} := n_p\)
  - send \(\langle \text{Promise}, n_p, n_a, v_a \rangle\) to \(p\)

- On \(\langle \text{Accept}, n_p, v \rangle\) 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 \(\langle \text{Accepted}, n, |v_a| \rangle\) to \(p\)

- On \(\langle \text{Decide}, v \rangle\):
  - If \(|v_d| < |v|\):
  - \(v_d := v\)
  - trigger Decide\((v_d)\)
Leader

On \(\text{Leader}, L, n\):
  • if self = L and n > n_L:
    • S := \emptyset, state := (leader, prepare)
    • propCmds := \langle \rangle; (v_L, n_L) := (\langle \rangle, n)
    • las := \lfloor 0 \rfloor N, l_c := 0
    • send \langle \text{Prepare}, n_L \rangle to all acceptor
  • else: state, leader := follower, L

On \langle Promise, n, n_a, v_a \rangle s.t. n = n_L and state = (leader, prepare)
  • add \((n_a, v_a)\) to S
  • if \(|S|\leq \lfloor (N+1)/2 \rfloor:
    • \((k, v) := \max(S) \text{ // adopt } v\)
    • v_L := v + propCmds; propCmds := \emptyset
    • send \langle \text{Accept}, n_L, v_L \rangle to all acceptors
    • state := (leader, accept)

On \langle Propose, C \rangle and state = (leader, accept)
  • v_L := v_L + \langle C \rangle
  • send \langle \text{Accept}, n_L, v_L \rangle to all acceptors

On \langle Propose, C \rangle and state = (leader, prepare)
  • propCmds := propCmds + \langle C \rangle

On \langle Accepted, n, m \rangle 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 \langle \text{Decide}, \text{prefix}(v_L, m) \rangle to all learners

Acceptors

On \langle \text{Prepare}, n_p \rangle from (a leader) p:
  • if \(n_{\text{prom}} < n_p:\)
    • \(n_{\text{prom}} := n_p\)
    • send \langle \text{Promise}, n_p, n_a, v_a \rangle to p

On \langle \text{Accept}, n_p, v \rangle from (a leader) p:
  • If \(n_{\text{prom}} \leq n_p:\)
    • \(n_{\text{prom}} := n_p\)
    • \((n_a, v_a) := \max((n_a, v_a), (n_p, v))\)
    • send \langle \text{Accepted}, n, |v_a| \rangle to p

Learner

On \langle \text{Decide}, v \rangle:
  • 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, \(v \leq v'\)

- 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 $\langle \text{Leader}, L, n \rangle$ and $L = p$:
  - $n > n_{(\text{prom at } p)}$
  - Hence $\langle \text{Promise}, n, n_{(a \text{ at } p)}, v_{(a \text{ at } p)} \rangle$ is unnecessary
- From now on the leader is extending his $v_a$
Leader

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

On $\langle \text{Promise}, n, n_a, v_a \rangle$ s.t. $n = n_L$ and state := (leader, prepare)
- add $(n_a, v_a)$ to $S$
- if $|S| = \lceil (N+1)/2 \rceil$:
  - $(k, v_a) := \text{max}(S)$ // adopt $v$
  - $v_a = v_a \oplus \text{propCmds}$; propCmds := $\langle \rangle$
  - send $\langle \text{Accept}, n_L, v_a \rangle$ to all acceptors
  - state := (leader, accept)

On $\langle \text{Propose}, C \rangle$ s.t. state = (leader, accept)
- $v_a = v_a \oplus \langle C \rangle$
- send $\langle \text{Accept}, n_L, v_a \rangle$ to all acceptors

On $\langle \text{Propose}, C \rangle$ and state = (leader, prepare)
- propCmds := propCmds $\oplus \langle C \rangle$

On $\langle \text{Accepted}, n, m \rangle$ from a, s.t. $n = n_L$ and state = accept
- ......

Acceptor

On $\langle \text{Prepare}, n_p \rangle$ from (a leader) p:
- if $n_{\text{prom}} < n_p$:
  - $n_{\text{prom}} := n_p$
  - send $\langle \text{Promise}, n_p, n_a, v_a \rangle$ to p

On $\langle \text{Accept}, n_p, v \rangle$ from (a leader) p:
- If $n_{\text{prom}} \leq n_p$:
  - $n_{\text{prom}} := n_p$
  - $(n_a, v_a) := \text{max}((n_a, v_a), (n_p, v))$
  - send $\langle \text{Accepted}, n, |v_a| \rangle$ to p

Learner

On $\langle \text{Decide}, v \rangle$:
- 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 $q$: $v_{(d \text{ at } q)}$ is a prefix of $v_{(a \text{ at } q)}$
  - Sequence $v_d$ can be replaced with an integer $l_d$, such that $v_d = \text{prefix}(v_a, l_d)$
Leader

- On \(\langle\text{Leader}, L, n\rangle\):
  - if \(\text{self} = L\) and \(n > n_L\):
    - \(S := \{(n_a, v_a)\}\), state := (leader, prepare)
    - ...
    - send \(\langle\text{Prepare}, n_L\rangle\) to all acceptor – \{self\}
  - else: (state, leader) := (follower, L)

- On \(\langle\text{Promise}, n, n_a, v_a\rangle\) s.t. \(n = n_L\) and state := (leader, prepare):
  - ...

- On \(\langle\text{Propose}, C\rangle\) s.t. state = (leader, accept)
  - \(v_a = v_a + \langle C \rangle\)
  - send \(\langle\text{Accept}, n_L, v_a\rangle\) to all acceptors

- On \(\langle\text{Propose}, C\rangle\) s.t. state = (leader, prepare)
  - propCmds := propCmds + \langle C \rangle

- On \(\langle\text{Accepted}, n, m\rangle\) from \(a\), s.t. \(n = n_L\) and state = (leader, accept)
  - las[a] := max(las[a], m)
  - If \(l_c < m\) and prefix\((v_a, m)\) is supported:
    - \(l_c := m\)
    - send \(\langle\text{Decide}, \text{prefix}(v_a, m), n_L\rangle\) to all learners

Acceptor

- On \(\langle\text{Prepare}, n_p\rangle\) from (a leader) \(p\):
  - if \(n_{\text{prom}} < n_p\):
    - \(n_{\text{prom}} := n_p\)
    - send \(\langle\text{Promise}, n_p, n_a, v_a\rangle\) to \(p\)

- On \(\langle\text{Accept}, n_p, v\rangle\) from (a leader) \(p\):
  - If \(n_{\text{prom}} \leq n_p\):
    - \(n_{\text{prom}} := n_p\)
    - \((n_a, v_a) := \max((n_a, v_a), (n_p, v))\)
    - send \(\langle\text{Accepted}, n, \lvert v_a \rvert\rangle\) to \(p\)

Learner

- Initially \(l_d\) is 0
- On \(\langle\text{Decide}, v, n\rangle\):
  - If \(l_d < |v|\) and \(n_{\text{prom}} = n\):
  - \(l_d := |v|\)
  - trigger \(\text{Decide}(\text{prefix}(v_a, l_d))\)
Avoid sending sequences
Leader

- On \(\langle \text{Leader}, L, n \rangle\):
  - if \(\text{self} = L \text{ and } n > n_L\):
    - \(S := \{ (n_a, v_a) \}\), state := (leader, prepare)
    - propCmds = \(\emptyset\), \((n_L, n_{prom}) := (n, n)\)
    - las := \([0]^N\), \(l_c := 0\), leader := self
    - send \(\langle \text{Prepare}, n_L \rangle\) to all acceptor – \{ self \}
  - else:
    - \((\text{state}, \text{leader}) := (\text{follower}, L)\)

- On \(\langle \text{Promise}, n, n_a, v_a \rangle\) s.t. \(n = n_L\) and state = ...prepare...
  - add \((n_a, v_a)\) to \(S\)
  - if \(|S| = \lfloor (N+1)/2 \rfloor\):
    - \((k, v_a) := \max(S)\) // adopt \(v\)
    - \(v_a = v_a + \text{propCmds}; \text{propCmds} = \emptyset\)
    - send \(\langle \text{Accept}, n_L, v_a \rangle\) to all acceptors
    - state := (leader, accept)

- On \(\langle \text{Propose}, C \rangle\) s.t. state = ...accept..
  - \(v_a = v_a + \langle C \rangle\)
  - send \(\langle \text{Accept}, n_L, v_a \rangle\) to all acceptors

- On \(\langle \text{Propose}, C \rangle\) s.t. state = ...prepare..
  - propCmds := \text{propCmds} + \langle C \rangle

- On \(\langle \text{Accepted}, n, m \rangle\) from \(a\), s.t. \(n = n_L\) and state = ...
  - \(\text{las}[a] := \max(\text{las}[a], m)\)
  - If \(l_c < m\) and prefix(\(v_a\), \(m\)) is supported:
    - \(l_c := m\)
    - send \(\langle \text{Decide}, \text{prefix}(v_a, m), n_L \rangle\) to all learners

Acceptors

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

Learner

- Initially \(l_d \) is 0

On \(\langle \text{Decide}, v, n \rangle\):
  - If \(l_d < |v|\) and \(n_{prom} = n\):
  - \(l_d := |v|\)
  - trigger \(\text{Decide(prefix}(v_a, l_d))\)
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 \((l_d = |v_d|, n_a)\) at L
  - Promise message includes either
    - \((n_a, \text{suffix}(v_a, l_d))_p\) if \(n_a\) at \(p\) \(\geq n_a\) at L
    - \((n_a, \langle \rangle)_p\) if \(n_a\) at \(p\) \(< n_a\) at L
- Proposer reconstructs the adopted sequence using \(\max()\)
Leader at round 3 p1 leader

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

<table>
<thead>
<tr>
<th>Round</th>
<th>Accepted by $p_1$</th>
<th>Accepted by $p_2$</th>
<th>Accepted by $p_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 3$</td>
<td>$\langle C_1, C_2, C_3, E, F, G \rangle$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n = 2$</td>
<td>$\langle C_1, C_2, C_3 \rangle$</td>
<td>$\langle C_1, C_2 \rangle$</td>
<td></td>
</tr>
<tr>
<td>$n = 1$</td>
<td>$\langle C_1, A, B, D \rangle$</td>
<td></td>
<td>$\langle C_1 \rangle$</td>
</tr>
<tr>
<td>$n = 0$</td>
<td>$\langle \rangle$</td>
<td>$\langle \rangle$</td>
<td>$\langle \rangle$</td>
</tr>
</tbody>
</table>
**Leader at round 3 p1 leader**

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

<table>
<thead>
<tr>
<th>Round</th>
<th>Accepted by $p_1$</th>
<th>Accepted by $p_2$</th>
<th>Accepted by $p_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 3$</td>
<td>$\langle C_1, C_2, E, F, G \rangle$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n = 2$</td>
<td>$\langle C_1, C_2, C_3 \rangle$</td>
<td>$\langle C_1, C_2 \rangle$</td>
<td></td>
</tr>
<tr>
<td>$n = 1$</td>
<td>$\langle C_1, A, B, D \rangle$</td>
<td>$\langle C_1 \rangle$</td>
<td></td>
</tr>
<tr>
<td>$n = 0$</td>
<td>$\langle \rangle$</td>
<td>$\langle \rangle$</td>
<td>$\langle \rangle$</td>
</tr>
</tbody>
</table>
If \( p_3 \) becomes a leader at 3

- Its decided sequence is \( \langle C_1, C_2 \rangle \)
  - \( (n_a = 2, \text{ suffix } = \langle \rangle)_p^3 \)
- \( p_3 \) 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 \( (n_a = 1, \text{ suffix } = \langle \rangle)_p^1 \)
  - \( p_2 \) sends \( (n_a = 2, \text{ suffix } = \langle C_3 \rangle)_p^2 \)
- If \( p_1 \) consulted: \( v_{a,p^3} = \langle C_1, C_2 \rangle + \langle \rangle \) and extended locally by \( \langle E, F, G \rangle \)
- \( v_{a,p^3} = \langle C_1, C_2, E, F, G \rangle \)

<table>
<thead>
<tr>
<th>Round</th>
<th>Accepted by ( p_1 )</th>
<th>Accepted by ( p_2 )</th>
<th>Accepted by ( p_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n = 3 )</td>
<td></td>
<td></td>
<td>( \langle C_1, C_2, E, F, G \rangle )</td>
</tr>
<tr>
<td>( n = 2 )</td>
<td>( \langle C_1, C_2, C_3 \rangle )</td>
<td>( \langle C_1, C_2 \rangle )</td>
<td></td>
</tr>
<tr>
<td>( n = 1 )</td>
<td>( \langle C_1, A, B, D \rangle )</td>
<td></td>
<td>( \langle C_1 \rangle )</td>
</tr>
<tr>
<td>( n = 0 )</td>
<td>( \langle \rangle )</td>
<td>( \langle \rangle )</td>
<td>( \langle \rangle )</td>
</tr>
</tbody>
</table>
Leader at round 3 p3 is a leader

- If $p_3$ becomes a leader at 3
  - Its decided sequence is $\langle C_1, C_2 \rangle$
    - $(n_a = 2, \text{suffix} = \langle \rangle)_p$3
  - $p_3$ 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)_p$1
    - $p_2$ sends $(n_a = 2, \text{suffix} = \langle C_3 \rangle)_p$2
  - If $p_2$ consulted: $\langle C_3 \rangle$ is added to $\langle C_1, C_2 \rangle$ and extended locally by $\langle E, F, G \rangle$
  - $v_{a,p} = \langle C_1, C_2, C_3, E, F, G \rangle$

<table>
<thead>
<tr>
<th>Round</th>
<th>Accepted by $p_1$</th>
<th>Accepted by $p_2$</th>
<th>Accepted by $p_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 3$</td>
<td></td>
<td>$\langle C_1, C_2, C_3, E, F, G \rangle$</td>
<td></td>
</tr>
<tr>
<td>$n = 2$</td>
<td>$\langle C_1, C_2, C_3 \rangle$</td>
<td>$\langle C_1, C_2 \rangle$</td>
<td></td>
</tr>
<tr>
<td>$n = 1$</td>
<td>$\langle C_1, A, B, D \rangle$</td>
<td></td>
<td>$\langle C_1 \rangle$</td>
</tr>
<tr>
<td>$n = 0$</td>
<td>$\langle \rangle$</td>
<td>$\langle \rangle$</td>
<td>$\langle \rangle$</td>
</tr>
</tbody>
</table>
Leader at round 3 p2 is a leader

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

<table>
<thead>
<tr>
<th>Round</th>
<th>Accepted by p_1</th>
<th>Accepted by p_2</th>
<th>Accepted by p_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 3</td>
<td>( \langle C_1, C_2, C_3, E, F, G \rangle )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n = 2</td>
<td>( \langle C_1, C_2, C_3 \rangle )</td>
<td>( \langle C_1, C_2 \rangle )</td>
<td></td>
</tr>
<tr>
<td>n = 1</td>
<td>( \langle C_1, A, B, D \rangle )</td>
<td>( \langle C_1 \rangle )</td>
<td></td>
</tr>
<tr>
<td>n = 0</td>
<td>( \langle \rangle )</td>
<td>( \langle \rangle )</td>
<td>( \langle \rangle )</td>
</tr>
</tbody>
</table>
Implementation

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

- **On** \(\langle \text{Prepare}, n_p, l_d, n \rangle\) from (a leader) \(p\):
  - if \(n_{\text{prom}} < n_p\):
    - \(n_{\text{prom}} := n_p\)
    - suffix := if \(n_a \geq n : \text{suff}(v_a, l_d)\) else \(\langle \rangle\)
  - send \(\langle \text{Promise}, n_p, n_a, \text{suff} \rangle\) to \(p\)
Implementation

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

- S = \{(n_1, v_1), \ldots, (n_k, v_k)\}

- \text{fun max}(S):
  - \((n,v) := (0,\langle \rangle)\)
  - \text{for} \((n',v)\) \text{ in} \(S\):
    - if \(n < n'\) or \((n = n' \text{ and } |v| < |v'|)\):
      - \((n,v) := (n',v')\)
    - \text{return} \((n,v)\)

- On \(\text{Prepare}, \ n_p, \ l_d, \ n\) from \text{(a leader)} \(p\):
  - if \(n_{prom} < n_p\):
    - \(n_{prom} := n_p\)
  - suffix := if \(n_a \geq n\) : suffix(v_a, l_d) else \langle \rangle\)
  - \text{send} \\langle \text{Promise}, \ n_p, \ n_a, \ \text{suffx} \rangle \text{ to} \ p\)
Leader

On \(\langle \text{Leader}, \text{L}, n \rangle\):
- if \(\text{self} = \text{L} \text{ and } n > n_L\):
  - leader := self, state := \ldots \text{prepare}\ldots
  - \(S := \{ (n_a, \text{suffix}(v_a, l_d)) \}\)
  - propCmds = \(\langle \rangle\), \(n_{\text{prom}} := (n, n)\)
  - \(l_a := [0]^N, l_c := 0\), leader := self
  - \(\text{send} \langle \text{Prepare}, n_L, l_d, n_a \rangle\) to all acceptor \{- self \}
- else: (state, leader) := (follower, L)

On \(\langle \text{Promise}, n, n_a, \text{suffix}_a \rangle\) s.t. \(n = n_L\) and state = \text{prepare}…
- add \((n_a, \text{suffix}_a)\) to \(S\)
- if |\(S|\geq \lceil (N+1)/2 \rceil:\)
  - \((k, \text{suffix}) := \text{max}(S) // \text{adopt } v\)
  - \(v_a = \text{prefix}(v_a, l_d) + \text{suffix} + \text{propCmds};\)
  - propCmds := \(\varnothing\)
  - \(\text{send} \langle \text{Accept}, n_L, v_a \rangle\) to all acceptors
  - state := \ldots \text{accept}…

On \(\langle \text{Propose}, C \rangle\) s.t. state = \text{accept}…
- \(v_a = v_a + \langle C \rangle\)
- \(\text{send} \langle \text{Accept}, n_L, v_a \rangle\) to all acceptors

On \(\langle \text{Propose}, C \rangle\) s.t. state = \text{prepare}…
- propCmds := propCmds + \(\langle C \rangle\)

On \(\langle \text{Accepted}, n, m \rangle\) from \(a\), s.t. \(n = n_L\) and state = \text{accept}.
- las[a] := max(las[a], m)
  - If \(l_c < m\) and prefix(v_a, m) is supported:
    - \(l_c := m\)
  - \(\text{send} \langle \text{Decide}, \text{prefix}(v_a, l_d), n_L \rangle\) to all learners

Acceptor

On \(\langle \text{Prepare}, n_p, l_d, n \rangle\) from (a leader) \(p\):
- if \(n_{\text{prom}} < n_p:\)
  - \(n_{\text{prom}} := n_p\)
  - \(n_{\text{prom}} := n_p\)
  - \(\text{suffix} := \text{if } n_a \geq n : \text{suffix}(v_a, l_d) \text{ else } \varnothing\)
  - \(\text{send} \langle \text{Promise}, n_p, n_a, \text{suffix} \rangle\) to \(p\)

On \(\langle \text{Accept}, n_p, v \rangle\) from (a leader) \(p\):
- If \(n_{\text{prom}} \leq n_p:\)
  - \(n_{\text{prom}} := n_p\)
  - \((n_a, v_a) := \text{max}((n_a, v_a), (n_p, v))\)
  - \(\text{send} \langle \text{Accepted}, n, |v_a| \rangle\) to \(p\)

Learner

Initially \(l_d\) is 0

On \(\langle \text{Decide}, v, n \rangle\):
- If \(l_d < |v|\) and \(n_{\text{prom}} = n:\)
  - \(l_d := |v|\)
  - \(\text{trigger} \text{Decide(prefix}(v_a, l_d))\)
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

$\langle \text{Leader, L, n} \rangle$

prepare

$\langle \text{Prepare ...} \rangle$

$\langle \text{Promise ...} \rangle$

$\langle \text{Accept ...} \rangle$

accept

$\langle \text{Promise ...} \rangle$

follower

$\langle \text{Leader, L', n'} \rangle$

follower

replica q

S. Haridi, KTHx ID2203.2x
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 $l_d$ at q in the promise
  - Leader L reconstructs his own $v_a$
  - For each replica q in the majority: L sends an AcceptSync message $\text{suffix}(v_a \text{ at } L, l_d \text{ at } q)$ and $l_d$ at q
Implementation

- On \langle \text{Promise}, n, n_a, \text{suff}_a, l_d \rangle \text{ from a s.t. } n = n_L \text{ and state } = \ldots\text{prepare}\ldots
  
  add \ (n_a, \text{suff}_a) \text{ to } S, \ lds[a] := l_d

  if \ |S| = \lceil \frac{N+1}{2} \rceil:
    \ (k, \text{suff}) := \max(S) // \text{adopt v}
    \ v_a = \text{prefix}(v_a, l_d) + \text{suff} + \text{propCmds};

  \ lds[\text{self}] := |v_a| //** selecting chosen sequence */
  \ \text{propCmds} = \emptyset, \ \text{state} := (\text{leader, accept})

  \text{for } p \text{ in } \pi- \{\text{self}\} \text{ s.t. } lds[p] \neq \bot:
  
    \text{send } \langle \text{AcceptSync}, n_L, \text{suff}(v_a, lds[p]), lds[p] \rangle \text{ to } p

- On \langle \text{Prepare}, n_L, l_d, n \rangle \text{ from } (\text{a leader}) L:
  
  if \ n_{\text{prom}} < n_L:
    \ n_{\text{prom}} := n_L

  \ \text{state} := (\text{follower, prepare})

  \text{suff} := \text{if } n_a \geq n : \text{suff}(v_a, l_d) \ \text{else } \langle \rangle

  \text{send } \langle \text{Promise}, n_L, n_a, \text{suff}, l_d \rangle \text{ to } p
Implementation

- On \( \langle \text{Promise}, n, n_a, \text{suff}x_a, ld_a \rangle \) from a \text{s.t.} \( n = n_L \) and state = (leader, prepare):
  - add \((n_a, \text{suff}x_a)\) to \( S \), \( lds[a] := ld_a \)
  - if \(|S|=(N+1)/2\):
    - \((k, \text{suff}) := \max(S) \) // adopt \( v \)
    - \( v_a := \text{prefix}(v_a, l_d) + \text{suff}x + \text{propCmds} \)
    - \( las[self] := |v_a| \) /* selecting chosen sequence */
    - \( \text{propCmds} = \emptyset \), state := (leader, accept)
  - for \( p \) in \( \pi - \{\text{self}\} \) \text{s.t.} \( lds[p] \neq \bot \):
    - send \( \langle \text{AcceptSync}, n_L, \text{suff}x(v_a, lds[p]), lds[p] \rangle \) to \( p \)

- On \( \langle \text{AcceptSync}, n_L, \text{suff}x_v, ld \rangle \) from \( L \) and state = (follower, prepare):
  - If \( n_{prom} = n_L \):
    - \( n_a := n_L \)
    - \( v_a := \text{prefix}(v_a, ld) + \text{suff}x_v \)
    - send \( \langle \text{Accepted}, n_L, |v_a| \rangle \) to \( p \)
  - state = (follower, accept)
Leader at round 3

- If \( p_1 \) becomes a leader at 3
  - Its decided sequence is \( \langle C_1 \rangle \)
    - \( (n = 1, \text{suffix} = \langle A, B, D \rangle)_{p_1} \)
  - \( p_1 \) consults itself and \( p_2 \) by sending \( \langle C_1 \rangle \)
    - \( p_2 \) sends \( (n = 2, \text{suffix} = \langle C_2, C_3 \rangle)_{p_2}, l_{d,p2} = 2 \)
  - \( P_1 \) 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
    - \( v_{a,p2} = \langle C_1, C_2, C_3, E, F, G \rangle \)
Leader at round 3

- If $p_1$ becomes a leader at 3
  - Its decided sequence is $\langle C_1 \rangle$
    - $(n = 1, \text{suffix} = \langle A, B, D \rangle)_p$ $p_1$
  - $p_1$ consults a majority
  - If $p_3$ consulted: $v_{a,p_1} = \langle C_1 \rangle + \langle C_2 \rangle$ and extended locally by $\langle E, F, G \rangle$
    - $v_{a,p_1} = \langle C_1, C_2, E, F, G \rangle$
  - $p_1$ sends
    - $\text{suffix}(v_{a,p_1}, l_{d,p_3}) = \langle E, F, G \rangle +$
    - $l_{d,p_2} = 2$
  - $p_3$ reconstructs its $v_a$ at round 3
    - $v_{a,p_2} = \langle C_1, C_2, E, F, G \rangle$

<table>
<thead>
<tr>
<th>Round</th>
<th>Accepted by $p_1$</th>
<th>Accepted by $p_2$</th>
<th>Accepted by $p_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 3$</td>
<td>$\langle C_1, C_2, E, F, G \rangle$</td>
<td>$\langle C_1, C_2, E, F, G \rangle$</td>
<td>$\langle C_1, C_2, E, F, G \rangle$</td>
</tr>
<tr>
<td>$n = 2$</td>
<td>$\langle C_1, C_2, C_3 \rangle$</td>
<td>$\langle C_1, C_2 \rangle$</td>
<td>$\langle C_1, C_2 \rangle$</td>
</tr>
<tr>
<td>$n = 1$</td>
<td>$\langle C_1, A, B, D \rangle$</td>
<td>$\langle C_1 \rangle$</td>
<td>$\langle C_1 \rangle$</td>
</tr>
<tr>
<td>$n = 0$</td>
<td>$\langle \rangle$</td>
<td>$\langle \rangle$</td>
<td>$\langle \rangle$</td>
</tr>
<tr>
<td>Round</td>
<td>Accepted by p₁</td>
<td>Accepted by p₂</td>
<td>Accepted by p₃</td>
</tr>
<tr>
<td>---------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>n = 3</td>
<td>〈C₁, C₂, C₃ E, F, G〉</td>
<td>〈C₁, C₂, C₃ E, F, G〉</td>
<td></td>
</tr>
<tr>
<td>n = 2</td>
<td>〈C₁, C₂, C₃〉</td>
<td>〈C₁, C₂〉</td>
<td></td>
</tr>
<tr>
<td>n = 1</td>
<td>〈C₁, A, B, D〉</td>
<td>〈C₁〉</td>
<td></td>
</tr>
<tr>
<td>n = 0</td>
<td>〈〉</td>
<td>〈〉</td>
<td>〈〉</td>
</tr>
</tbody>
</table>

If \(p₂\) becomes a leader at 3

- Its decided sequence is \(\langle C₁, C₂ \rangle\)
  - \((n_a = 2, \text{suffix } = \langle C₃ \rangle)\) \(p₃\)
  - \(p₂\) consults a majority, itself and either \(p₁\) or \(p₃\) by \(|v_d| = |\langle C₁, C₂ \rangle|, n_a = 2\) \(p₂\)
  - \(p₁\) sends \((n_a = 1, \text{suffix } = \langle \rangle)\) \(p₁\), \(l_{d,p₁} = 1\)
  - \(p₂\) sends to \(p₁\)
    - suffix\((v_{a,p₁}, l_{d,p₁}) = \langle C₂, C₃ E,F,G \rangle\)
    - \(l_{d,p₁} = 1\)
  - \(p₁\) reconstructs its \(v_a\) at round 3
    - \(v_{a,p₁} = \langle C₁, C₂, C₃, E, F, G \rangle\)
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 $l_d$ 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:
    - $\text{suffix}(v_a \text{ at } L, \ l_d \text{ at } q)$ and $l_d$ at $q$
  - If some sequence is already decided it sends the decide index $l_d$ 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 \( \langle Promise, n, n_a, \text{suffix}_a, ld_a \rangle \) from a and \( n = n_L \) and state = (leader, accept)
  - \( \text{lds}[a] := ld_a \)
  - send \( \langle AcceptSync, n_L, \text{suffix}(v_a, \text{lds}[a]), \text{lds}[a] \rangle \) to a
  - if \( l_c \neq 0 \):
    - send \( \langle Decide, l_d, n_L \rangle \) to a

- On \( \langle AcceptSync, n_L, \text{suffix}_v, ld \rangle \) from L and state = (follower, prepare):
  - If \( n_{\text{prom}} = n_L \):
  - \( n_a := n_L \)
  - \( v_a := \text{prefix}(v_a, ld) + \text{suffix}_v \)
  - send \( \langle Accepted, n_L, |v_a| \rangle \) to p
  - state = (follower, accept)
Updating replicas (incremental Accepts)

- Subsequent `Accept` messages:
  - Let \( m_1 = \langle \text{Accept}, n_L, v_1 \rangle \) and \( m_2 = \langle \text{Accept}, n_L, v_2 \rangle \), 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 \( vs = \text{suffix}(v_2, |v_1|) \) and \( \text{offset} = |v_1| \) instead of \( v_2 \)
    - In particular if \( v_2 = v_1 + \langle C \rangle \): \( m_2 \) is \( \langle \text{Accept}, n_L, \langle C \rangle, |v_1| \rangle \)
Implementation

- When a leader $L$ in the accept state gets a new command $C$
  - Updates its accepted sequence and its $\text{las}[L]$
  - Sends Accept messages to all replicas that passed the prepare phase

- On $\langle\text{Propose}, C\rangle$ and state = (leader, accept)
  - $v_a = v_a \oplus \langle C\rangle$
  - $\text{las}[\text{self}] := \text{las}[\text{self}] + 1$
  - for $p$ in $\pi$- {self} s.t. $\text{lds}[p] \neq \bot$
    - send $\langle\text{Accept}, n_L, \langle C\rangle\rangle$ 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 $\langle\text{Accept}, n_L, \langle C\rangle\rangle$ from (a leader) $L$ and state = (follower, accept)
  - If $n_{\text{prom}} = n_L$:
  - $v_a := v_a \oplus \langle C\rangle$
  - send $\langle\text{Accepted}, n_p, |v_a|\rangle$ to $L$
How to Decide
Implementation

- The leader maintains
  - $\textit{l}\text{as}[0]$ : 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 $\langle \text{Accepted}, n, m \rangle$ from a, s.t. $n = n_L$ and state = (leader, accept)
  - $\textit{l}\text{as}[a] := m$
  - If $l_c < m \text{ and } |\{p \in \pi : \textit{l}\text{as}[a] \geq m\}| \geq \lfloor (N+1)/2 \rfloor$
  - $l_c := m$
  - for $p$ in $\pi$ s.t. $\text{lds}[p] \neq \perp$
  - send $\langle \text{Decide}, l_c, n_L \rangle$ 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 \( \langle \text{Decide}, l, n_L \rangle \):
  - if \( n_{\text{prom}} = n_L \):
  - while \( l_d < l \):
    - trigger Decide\( (v_a[l_d]) \)
  - \( l_d := l_d + 1 \)

Initially \( l_d \) is 0
On \( \langle \text{Decide}, v, n \rangle \):
  - if \( l_d < |v| \) and \( n_{\text{prom}} = n \):
  - \( l_d = |v| \)
  - trigger Decide\( (\text{prefix}(v_a, l_d)) \)
The final algorithm
The final Sequence Paxos algorithm

- The algorithm uses:
  - 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 \(p \mapsto (n, v)\)

- **Replica (including Acceptor and Learner)**
  - \((n_L, \text{leader}) = (0, \bot)\) Leader’s current round number, leader process
  - state = \(\{(\text{follower, leader}), \{\text{prepare, accept, } \bot\}\}\) initially \((\text{follower, } \bot)\)
  - \(n_{\text{prom}} = 0\) 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 \(\text{Leader}, \ L, \ n\):
  if \(n > n_L\):
    leader := L \_ n_L := n
  if self = L and \(n_L > n_{\text{prom}}\):
    state := (leader, prepare)
    propCmds = \(\langle \rangle\); las := [0]^N; lds := [\bot]^N
    acks := [\bot]^N; \(l_c := 0\),
    send \(\langle \text{Prepare}, \ n_L, l_d, n_a \rangle\) to all \(\pi \_ \{\text{self}\}\)
    acks[L] := (n_a, suffix(v_a, l_d))
    lds[\text{self}] := l_d; n_{\text{prom}} := n_L
  else:
    state = (follower, state[2])

On \(\text{Prepare}, \ n_L, l_d, n\) from L:
  if \(n_{\text{prom}} < n_L\):
    n_{\text{prom}} := n_L; state := (follower, prepare)
    suffix := if \(n_a \geq n\) : suffix(v_a, l_d) else \(\langle \rangle\)
    send \(\langle \text{Promise}, n_L, n_a, \text{suffix}, l_d \rangle\) to L

On \(\text{Promise}, n, n_{a}, \text{suffix}_a, l_d\) from a
  s.t. \(n = n_L\ and\ state = (\text{leader, prepare})\):
  acks[a] := (n_a, \text{suffix}_a), lds[a] := l_d
  \(P := \{p \in \pi : \text{acks}[p] \neq \bot\}\)
  if \(|P| = \lceil (N+1)/2 \rceil\):
    \((k, \text{suffix}) := \max\{\text{acks}[p] : p \in P\}\) // adopt \(v\)
    \(v_a = \text{prefix}(v_a, l_d) + \text{suffix} + \text{propCmds};\)
    las[\text{self}] := \(|v_a|\)
    propCmds := \(\langle \rangle\); state := (leader, accept)
    for p in \(\pi\_\{\text{self}\}\) and lds[p] \(\neq \bot\):
      suf := suffix(v_a, lds[p])
      send \(\langle \text{AcceptSync}, n_L, \text{suf}, \text{lds}[p] \rangle\) to p

On \(\text{Promise}, n, n_{a}, \text{suffix}_a, l_d\) from a
  s.t. \(n = n_L\ and\ state = (\text{leader, accept})\):
  lds[a] := l_d
  send \(\langle \text{AcceptSync}, n_L, \text{suffix}(v_a, lds[a]), \text{lds}[a] \rangle\) to a
  if \(l_c \neq 0\):
    send \(\langle \text{Decide}, l_d, n_L \rangle\) to a

S. Haridi, KTHx ID2203.2x
On \(\texttt{AcceptSync}, n_L, \text{sufx}, ld\) from p

s.t. state = (follower, prepare):

If \(n_{prom} = n_L:

n_a := n_L

v_a := \text{prefix}(v_a, ld) + \text{sufx}

send \(\texttt{Accepted}, n_L, |v_a|\) to p

state = (follower, accept)

On \(\texttt{Accept}, n_L, \langle C\rangle, ld\) from p

s.t. state = (follower, accept):

If \(n_{prom} = n_L:

v_a := n_L + \langle C\rangle

send \(\texttt{Accepted}, n_L, |v_a|\) to p

On \(\texttt{Decide}, l, n_L\):  

if \(n_{prom} = n_L:

while \(l_d < l:

\text{trigger} \ \text{Decide}(v_a[l_d])

l_d := l_d + 1

On \(\texttt{Propose}, C\)  

s.t. state = (leader, prepare):

\(\text{propCmds} := \text{propCmds} + \langle C\rangle\)

On \(\texttt{Propose}, C\)  

s.t. state = (leader, accept):

\(v_a = v_a + \langle C\rangle\)

\(l\text{as}[\text{self}] := l\text{as}[\text{self}] + 1\)

for \(p\) in \(\pi\) - \{self\} s.t. \(lds[p] \neq \perp:\n
\quad \text{send} \(\texttt{Accept}, n_L, \langle C\rangle\) to p\)

On \(\texttt{Accepted}, n, m\) from a,  

s.t. \(n = n_L\) and state = (leader, accept):

\(l\text{as}[a] := m\)

If \(l_c < m\) and \(|\{p \in \pi : l\text{as}[a] \geq m\}| \geq \lceil (N+1)/2 \rceil :\n
\quad l_c := m,

\quad \text{for} \ p \ \text{in} \ l\ s.t. \ lds[p] \neq \perp:\n
\quad \quad \text{send} \(\texttt{Decide}, l_c, n_L\) 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 $<\text{Leader}, L, n>$ 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 $\langle \text{Leader}, p, n \rangle$, then for any for previous event and process $q$:
    - $\langle \text{Leader}, q, n' \rangle$: $n' < n$ 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.