Caltech, CMS. CS/IDS 142: Lecture 8.2
Distributed Fault-Tolerant Consensus: Paxos
Mani Chandy
18 November 2019

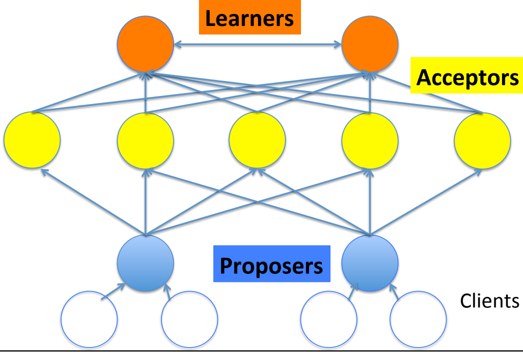
Take away concepts from the course:

- State transition systems.
- Always nothing bad happens.
- Progress. Variant function, Metric. The system never gets further away from its goal and eventually gets closer.
- Map global view to local agent state
- Data structures, e.g. graphs, that change with computation.

Consensus in Faulty Distributed Systems

- **Importance of consensus**: Multiple agents agree on a single sequence of events: banking, games, databases….
- Modifications to usual distributed systems model:
  - Agents may halt and restart from previously saved state.
  - Messages may be lost or duplicated.
- Assumptions: No Byzantine operations
  - Messages are not corrupted
  - Agents are not malicious

Clients, Proposers, Acceptors, Learners



Specification

Safety

1. Learners learn at most one value.

Associated with learner z is a variable learned[z] where learned[z] is either None (representing undefined) or some non-None value.

For v != None: stable(learned[z] = v)

2. Learners learn only proposed values

(learned[z] != None) => learned[z] in set of proposals

3. **All learners learn the same unchanging value**

For all learners z, z': (learned[z]=None) or (learned[z']=None) or (learned[z]=learned[z'])

Example: Group of students want to come to a consensus on an activity

- Some students in a group want to hike, others to swim, and others to dance.

1. Learners must learn a result proposed by a client. A learner must not learn that the result is "sleep" if the clients only propose "hike", "swim", "dance".
2. All learners must learn the same result. All hike, or all swim, or all dance.
3. After a learner learns a result that result remains unchanged. E.g. Learner won't switch from hike to swim.

Student is client, proposer, acceptor, and learner. The same agent plays all roles in many applications.

Progress Not Guaranteed!!

Not guaranteed: learners eventually learn a value.
For all learners z: eventually(learned[z] != None)

Fischer, Lynch, Patterson (FLP) theorem says that ***consensus cannot be achieved with a single faulty process***. Why?

We cannot prove progress; but we will discuss best effort algorithms.

Algorithm has two phases. A phase may be run multiple times.

- Phase 1:
  - **Prepare(t)** message from proposer to acceptor
  - **promise(promise_t, accepted_t, value)** reply from acceptor to proposer
- Phase 2:
  - **Request(t, value)** message from proposer to acceptor
  - **Accept**(**t**, **value**) message from acceptor to learner
- *Learner learns the proposal chosen by majority of acceptors.*
- Different proposers never use the same t value.

Break ties lexicographically

Algorithm for proposer P:

State: (P.t, P.value) Initially (-1, x)
Start timer
While not timed_out:

choose t greater than P.t and set P.t = t
PHASE 1
1. send prepare(P.t) to all acceptors
2. wait for promise(prepare_t, accept_t, value) replies from (at least) a majority of acceptors where prepare_t == P.t
3. If value is not None for one or more of these promise messages then set P.value to the value in the promise message with the largest accept_t

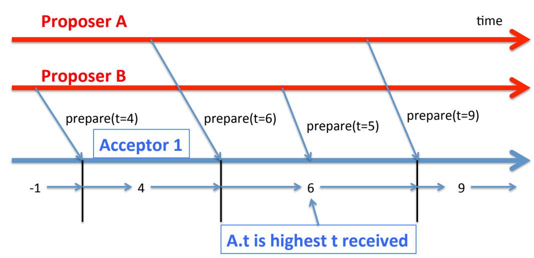
PHASE 2
4. send request(P.t, P.value) to all acceptors.
5. Wait for accepted(t, value) replies from majority of acceptors where (t, value) == (P.t, P.value)

Algorithm for acceptor A:

State: (A.t, A.accepted_t, A.value) Initially (None, None, None)
Start timer
While not timed_out:
upon receiving prepare(t):
if t >= A.t:
A.t = t
reply with promise(t, A.accepted_t, A.value)
upon receiving request(t, value):
If t >= A.t:
reply with accepted(t, value)

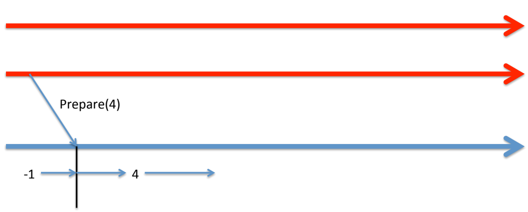
Meaning of **promise**(**n**, **accepted_t**, **value**) sent by an acceptor A

- Acceptor A will ignore all **prepare**(**t**) and **request**(t, v) messages with t < n.



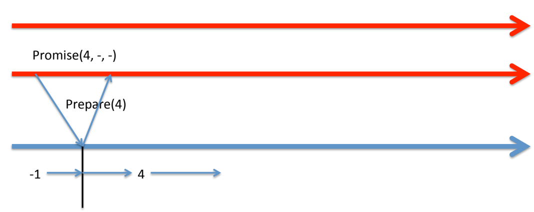
Meaning of **promise**(**n**, **accepted_t=m**, **value**=**v**) sent by an acceptor

- Acceptor will ignore all **prepare**(**id**=**m**) and messages that it receives where m < n.



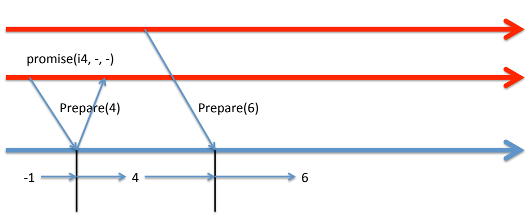
Meaning of **promise**(**id**=**n**, **accepted_id=a**, **value**=**v**) sent by an acceptor V

- Acceptor V will reject (reply with nacks) to all **prepare**(**id**=**m**) messages that it receives where m < n.



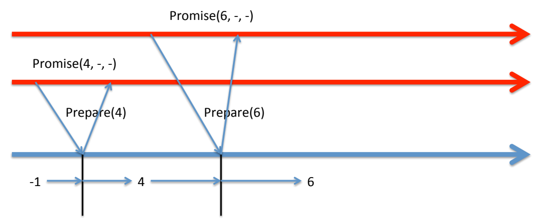
Meaning of **promise(id=n, accepted_id=a, value=v)** sent by an acceptor V

- Acceptor V will reject (reply with nacks) to all **prepare(id=m)** messages that it receives where m < n.



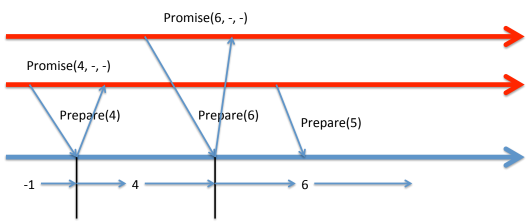
Meaning of **promise(id=n, accepted_id=a, value=v)** sent by an acceptor V

- Acceptor V will reject (reply with nacks) to all **prepare(id=m)** messages that it receives where m < n.



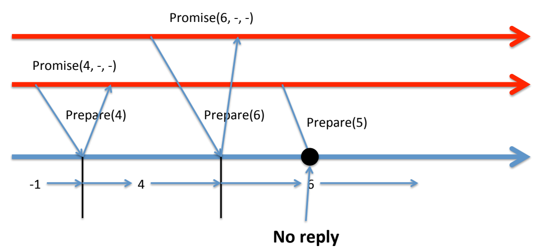
Meaning of **promise(id=n, accepted_id=a, value=v)** sent by an acceptor V

- Acceptor V will reject (reply with nacks) to all **prepare(id=m)** messages that it receives where m < n.



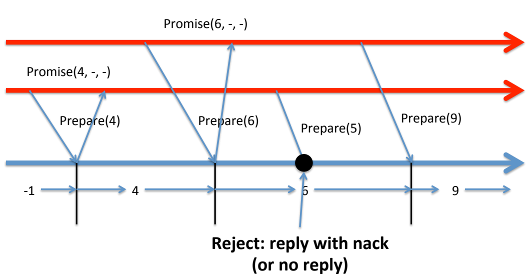
Meaning of **promise(id=n, accepted_id=a, value=v)** sent by an acceptor V

- Acceptor V will reject (reply with nacks) to all **prepare(id=m)** messages that it receives where m < n.



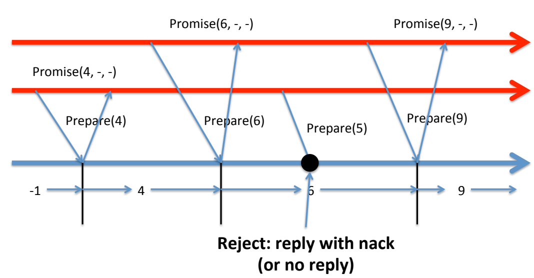
Meaning of **promise(id=n, accepted_id=a, value=v)** sent by an acceptor V

- Acceptor V will reject (reply with nacks) to all **prepare(id=m)** messages that it receives where m < n.

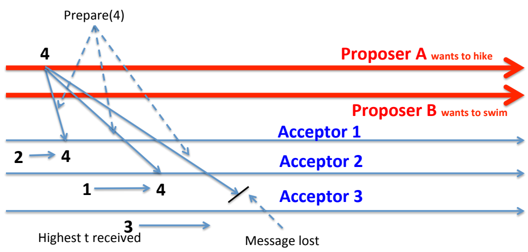


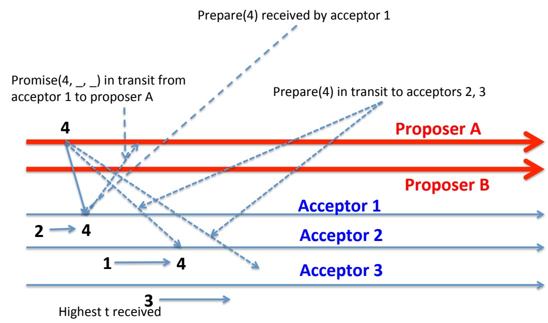
Meaning of **promise(n, accepted_t=T, value=v)** sent by an acceptor

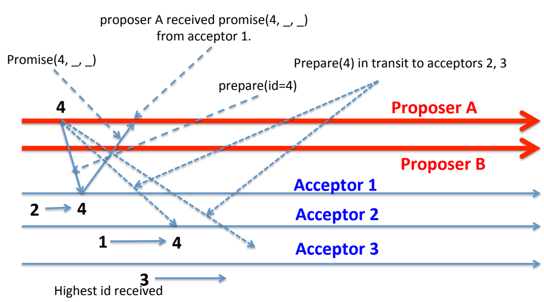
- The acceptor will ignore all **prepare(m)** messages that it receives where m < n.

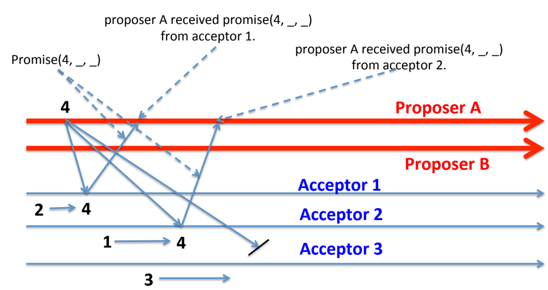


A group of students wants to come to a consensus on a common activity. Some students want to hike, others to swim, others to dance, …..
And they must finally agree on exactly one activity.
Picture shows only two proposers for brevity.
Proposer A sends prepare(4) to acceptors.

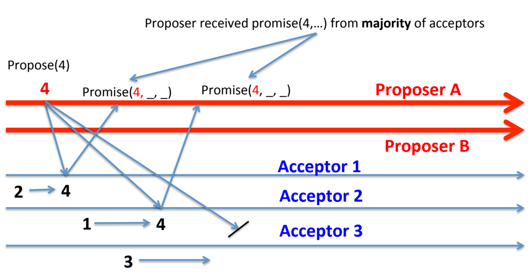




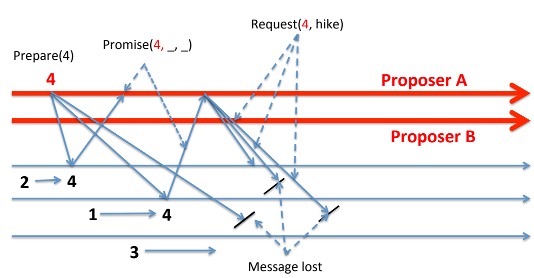




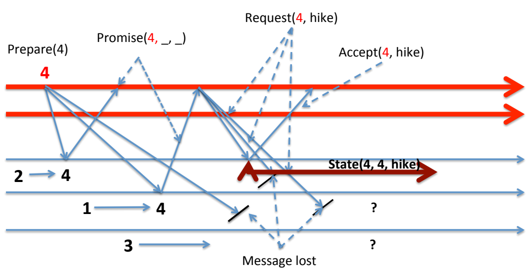
Proposer sets its value when it receives promise messages from majority of acceptors. Since promise messages have value = None, the proposer's value remains its initial value (e.g., hike)



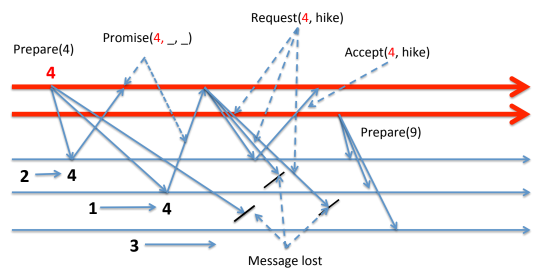
Proposer A sends request(4, hike) messages to all acceptors.
Request to Acceptor 1 arrives but requests to acceptors 2, 3 are lost.
Acceptor 1 accepts request(4, hike) and replies with accept(4, hike)



Proposer A sends request(4, hike) messages to all acceptors.
Request to Acceptor 1 arrives but requests to acceptors 2, 3 are lost.
Acceptor 1 accepts request(4, hike) and replies with accept(4, hike)
Problem? Acceptor 1 has accepted "hike" but acceptors 2, 3 have not

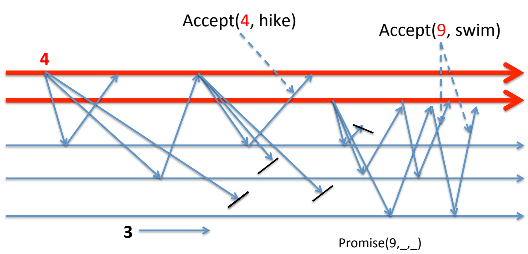


Acceptor 1 accepts request(4, hike) and replies with accept(4, hike)
Now proposer B (who wants to swim) sends prepare(9)



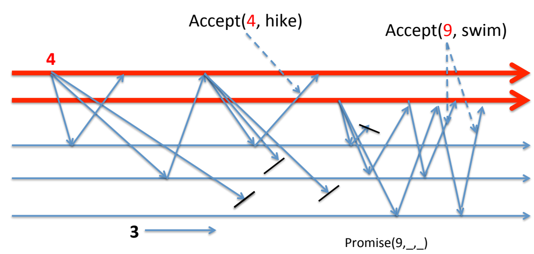
Acceptor 1 accepts (4, hike)
Acceptors 2, 3 accept (9, swim)

Is that a problem?



Acceptor 1 accepts (4, hike)
Acceptors 2, 3 accept (9, swim)

Is that a problem? No see the theorem!



Theorem: stable(majority of acceptors accept the same (t, value)
Proof idea: Consider an acceptor A outside this majority.
Either A has not accepted any value or
the timestamp (t-value) of A is less than the timestamps of this majority