Consensus: Paxos

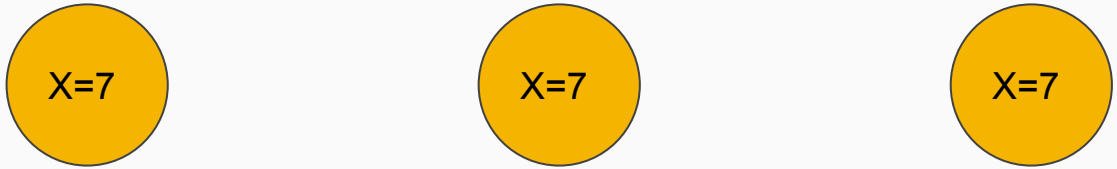
Haobin Ni

Oct 22, 2018

What is consensus?

A group of people go to the same place for the same meal.

→ A set of nodes have the same value for the same variable.



Why is it important?

In state machine replication (SMR):

- We need to keep replicas “the same” = consensus

In blockchain:

- We need to maintain the integrity of global ledger = consensus

What are the goals?

Safety properties:

- Agreement: any two correct nodes decide the same value
- Validity: any value decided should be a value proposed by some node

Liveness properties:

- Termination: eventually all correct nodes should decide

What are the goals? cont’

Performance:

- The number of replicas needed
- The number of message rounds
- The total number of messages
- Overhead (throughput/latency)
- average/optimistic/etc

What are the assumptions?

Failure/adversary model

- Fault-stop/Crash/Byzantine
- Limited computational/mining power
- Rational

What are the assumptions? cont’

Network model

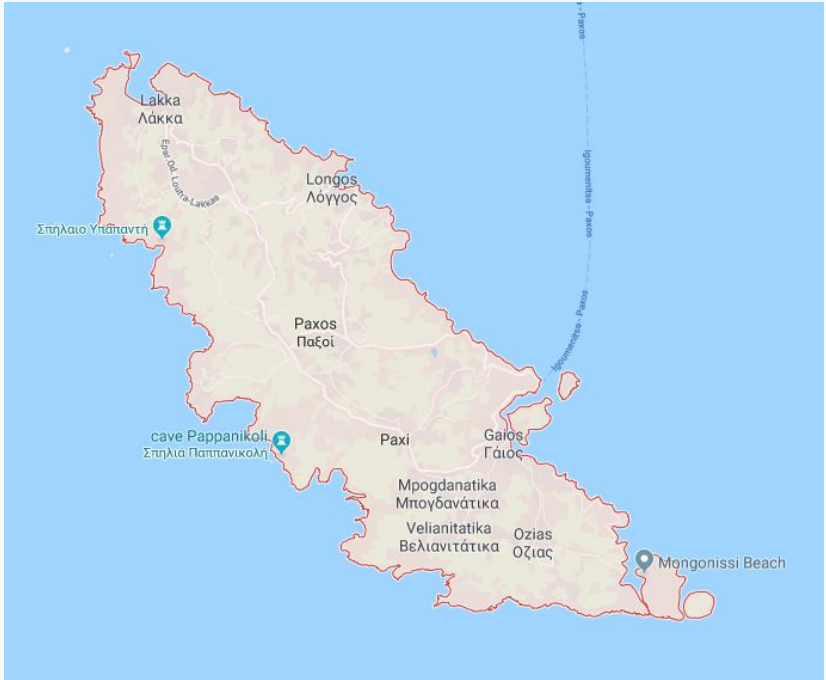
- Delay: Synchronous/Partial synchronous/Asynchronous
- Integrity/Confidentiality: Authenticated channel/Public key infrastructure
- Clock
- Topology

The Part-time Parliament 98’

- Paxos Algorithm
- Assumes:
  - Crash failure
  - Asynchronous network
- Achieves:
  - With f faulty nodes:
  - Needs f+1 for safety
  - Needs 2f+1 for liveness
  - 3 message rounds for each decision made

The Part-time Parliament 98’ cont’

Paxos is a Mediterranean island




London
Amsterdam
Netherlands
Hamburg
Berlin
Poland
Belarus
Voronezh
Воронеж
Саратов
Cologne
Germany
Belgium
Luxembourg
Paris
Prague
Czechia
Kyiv
Київ
Kharkiv
Харків
Volgograd
Волгоград
Ukraine
Rostov-on-Don
Ростов-на-Дону
Munich
Vienna
Slovakia
Budapest
Austria
France
Switzerland
Hungary
Moldova
Chisinau
Odesa
Одеса
Krasnodar
Краснодар
Bay of Biscay
Milan
Slovenia
Zagreb
Romania
Makhachkala
Махачкала
Caspian Sea
Croatia
Belgrade
Београд
Bucharest
Tbilisi
თბილისი
Porto
Monaco
Bosnia and
Herzegovina
Serbia
Georgia
Andorra
Black Sea
Baku
Azerbaijan
Portugal
Madrid
Barcelona
Italy
Sarajevo
Armenia
Podgorica
Подгорица
Kosovo
Bulgaria
Rome
Spain
Valencia
Macedonia
(FYROM)
Istanbul
Tirana
Albania
Bursa
Ankara
Seville
Tyrrhenian Sea
Turkey
Málaga
Gibraltar
Greece
Athens
Αθήνα
Izmir
Algiers
مدينة الجزائر
Adana
Rabat
الرباط
Tunis
تونس
Antalya
Baghdad
بغداد
Casablanca
الدار البيضاء
Syria
Malta
Cyprus
Lebanon
Morocco
Iraq
Beirut
بيروت
Tunisia
Mediterranean Sea
Tripoli
طرابلس
Jordan
Jerusalem
Kuwait
Israel
Cairo
القاهرة
Algeria
Riyadh
الرياض
Egypt
Medina

The Part-time Parliament 98’ cont’

Paxos is a Mediterranean island

The Part-time Parliament is a fictional parliamentary system of ancient Paxos where people experience asynchrony and crash failures.

“Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators…”

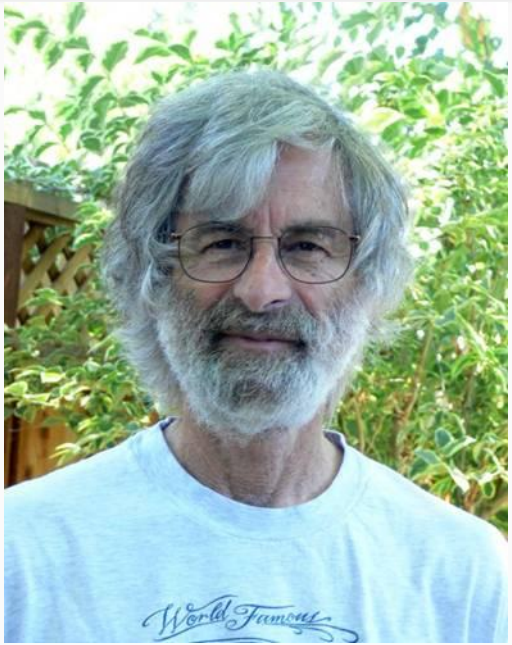
Leslie Lamport

Hey, we've met before!

Time, Clocks, and the Ordering of Events in a Distributed System

And we'll meet again!

The Byzantine Generals Problem



Leslie Lamport on *The Part-time Parliament*

“My attempt at inserting some humor into the subject was a dismal failure… I submitted the paper to TOCS in 1990. All three referees said that the paper was mildly interesting, though not very important, but that all the Paxos stuff had to be removed. I was quite annoyed at how humorless everyone working in the field seemed to be, so I did nothing with the paper…”

http://lamport.azurewebsites.net/pubs/pubs.html#lamport-paxos

“how humorless everyone working in the field seemed to be…”

“Legislators could communicate only by messenger and were provided with funds to hire as many messengers as they needed. A messenger could be counted on not to garble messages, but he might forget that he had already delivered a message, and deliver it again. Like the legislators they served, messengers devoted only part of their time to parliamentary duties. A messenger might leave the Chamber to conduct some business—perhaps taking a six-month voyage—before delivering a message. He might even leave forever, in which case the message would never be delivered.”

“how humorless everyone working in the field seemed to be…”

“We assume the messages could not be changed and an asynchronous network where messages could be duplicated or delayed indefinitely.”

Paxos Made Simple 01’

“The Paxos algorithm, when presented in plain English, is very simple.”

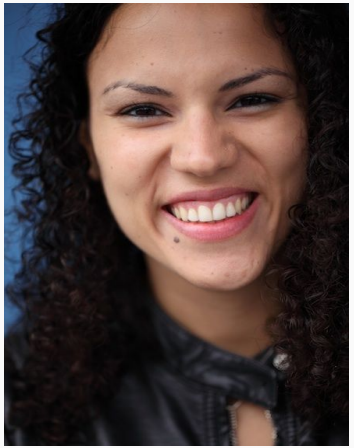
The description of the algorithm is greatly simplified by deleting all the Paxos stuff and use modern terms. (11pages vs 37pages)

But the author is still hand-waving many details (such as leader) and how to implement SMR with Paxos.

“...Do not try to implement the algorithm from this paper. Use [parliament] instead.”

Paxos Made Moderately Complex 15’

Robbert van Renesse and Deniz Altinbuken



SMR implemented with a practical version of multi-Paxos. Specified and explained in detail with invariants, pseudocode and a Python implementation.

Disclaimer: I will now try to explain Paxos in a different and ~~maybe~~ very confusing way

Paxos Outline

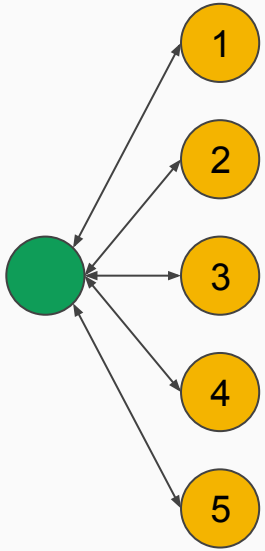
- “Single-decree synod” (Single value consensus)
  - Basic protocol ← In this talk
  - Leader selection
- Multi-decree synod
  - Running multiple parallel “Single-decree synod” protocols
  - Each instance decide one value
    - The decided values are ordered by their instance
- + optimization + state machine repilication = Moderately complex

Single-decree Synod

- Run many rounds of 3-phase commit protocol
  - Each round either decides some value or abort

- Maintains agreement across rounds
  - If a round decides X, all other rounds can only decide X
  - The execution of each instance may interleave each other.

3 Phase Commit - Setting



Phase 1: Prepare

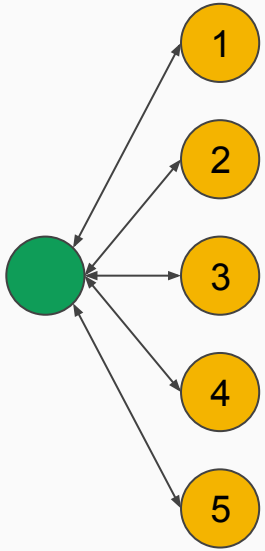
Each round of 3 phase commit will have a node to initiate this round.

This node is called the leader of that round.

We can make sure each round only have 1 node act as the leader by enforcing different nodes use different round numbers.

This also defines a total order on all rounds. (with smaller/larger round number).

3 Phase Commit - Prepare



Phase 1: Prepare

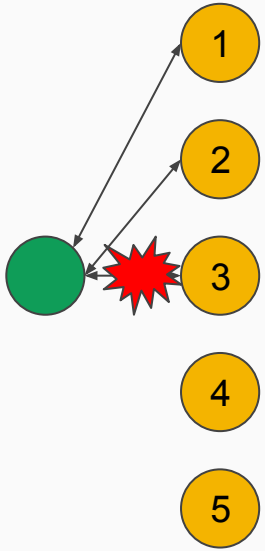
A leader choose a majority/quorum set.

Poll that set on what value to propose.

It should always propose a value if that value could become the decision.

The acceptors will abort previous rounds unless it has accepted that round. In this case it informs the leader that value could still become the decision.

3 Phase Commit - Prepare



Phase 1: Prepare

Leader: We now start round 8. What do you feel like for lunch?

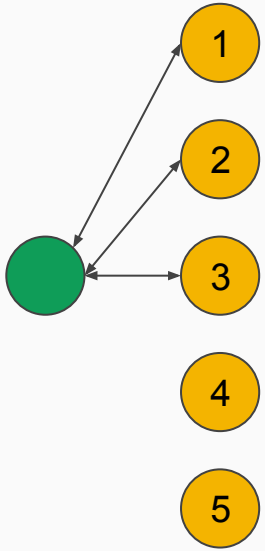
Acceptor 1: I have no idea.

Acceptor 2: I already accepted Thai in round 6.

Acceptor 3: … (Never receive this message)

Acceptor 1,2: Now we give up all rounds < 8.

3 Phase Commit - Prepare



Phase 1: Prepare

Leader: We now start round 10. What do you feel like for lunch?

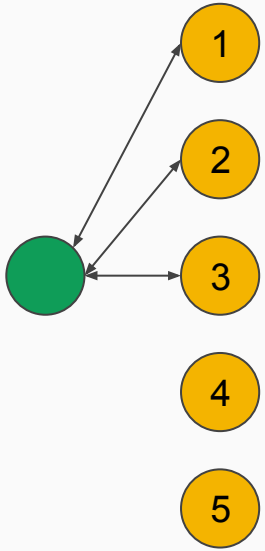
Acceptor 1: I have no idea.

Acceptor 2: I already accepted Thai in round 6.

Acceptor 3: I already accepted Taco in round 7.

Acceptor 1,2,3: Now we give up all rounds < 10.

3 Phase Commit - Propose



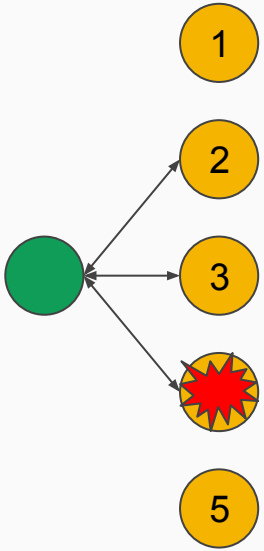
Phase 2: Propose

The leader now propose the latest value some acceptors has accepted to **the same majority set**.

This value gives safety. (we'll explain this later)

The acceptors will respond with an accept message unless it has aborted this round.

3 Phase Commit - Propose



Phase 2: Propose

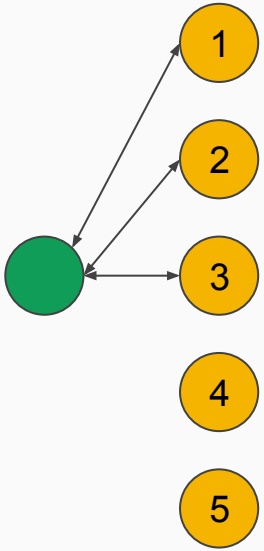
Leader': Let's try agree on Thai in round 6.

Acceptor 2: OK. Accept Thai in round 6.

Acceptor 3: No I aborted since I'm now participating round 10.

Acceptor 4: … (Crashes)

3 Phase Commit - Propose



Phase 2: Propose

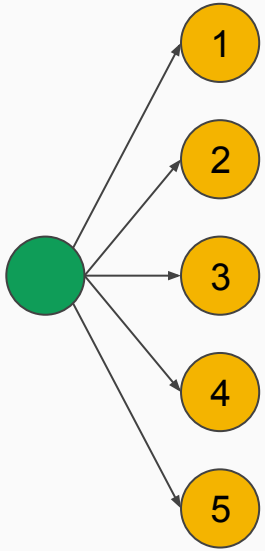
Leader: Let’s try agree on Taco in round 10.

Acceptor 1: OK. Accept Taco in round 10.

Acceptor 2: OK. Accept Taco in round 10.

Acceptor 3: OK. Accept Taco in round 10.

3 Phase Commit - Learn



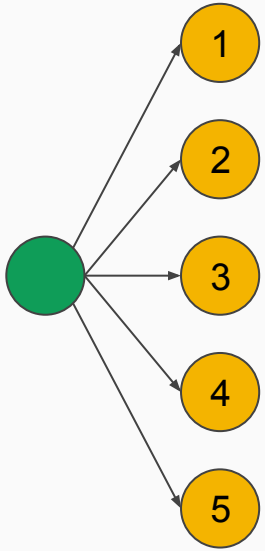
Phase 3: Learn

If all nodes in the majority/quorum set of a round accepted that value, anyone is safe to decide that value.

Someone collects the accept messages and shares the information “I have seen a round of 3 phase commit succeeds”.

The decision will be the output of a single-decree synod instance.

3 Phase Commit - Learn

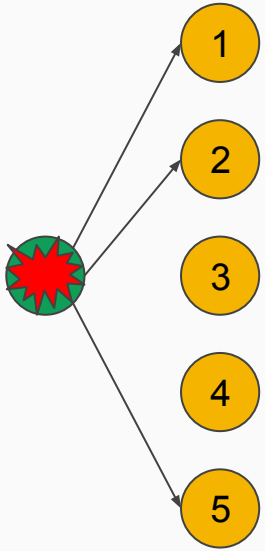


Phase 3: Learn

Leader (learner): I see we have agreed on Taco in round 10.

Everyone: Yay, Taco!

3 Phase Commit - Learn



Phase 3: Learn

Leader (learner): I see we have agreed on Taco in round 10 … and it crashes!

Acceptor 1,2,5: Yay, Taco!

Acceptor 3,4: ???

But Acceptor 3,4 will learn that decision eventually as all other rounds can now only decide Taco.

Single-decree Synod

We now analyze the single-decree synod protocol.

Remember our goals:

- Agreement
- Validity (trivial)
- Liveness

Agreement

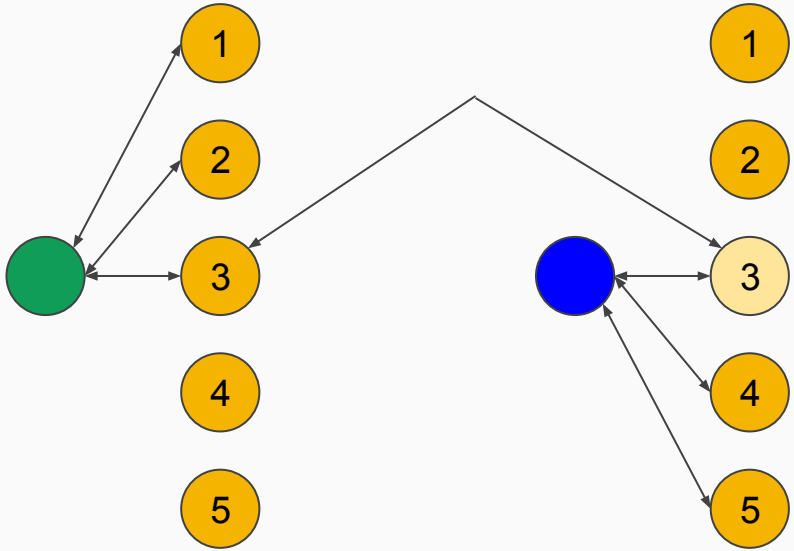
Each round of 3 phase commit either succeeds or aborts.

We only need to prove if any round succeeds, then any round with a larger round number either aborts or proposes the same value.

Suppose round i succeeds, round j is some round after i.

Agreement cont’

There must exist a node that belongs to the intersection of two quorums of two rounds.



Agreement cont’

There must exist a node that belongs to the intersection of two quorums of two rounds.

For all such nodes, each round it only receives prepare and propose messages and it must have replied both messages of round i for it to succeed.

Since i < j, it must receive round j’s prepare message after it have accepted in round i. (Otherwise it will abort round i) Now we induct on j.

Agreement cont’

Base case: j = i + 1

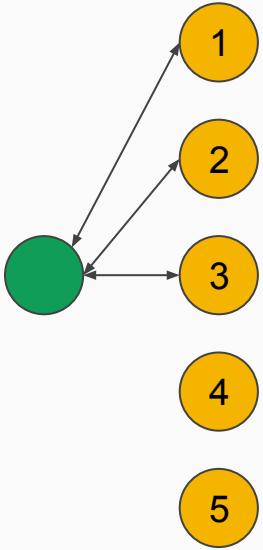
The only possible scenarios (for a node in the intersection) are:

Prepare i (OK _ _) → Propose i v (Accept i v) → Prepare j (Abort)

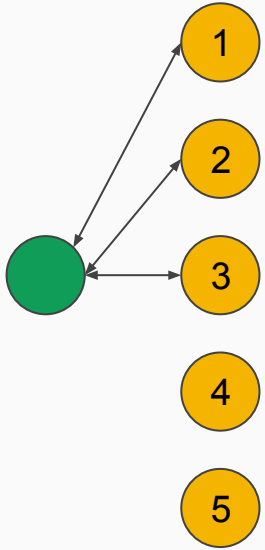
Prepare i (OK _ _) → Propose i v (Accept i v) → Prepare j (OK i v)

In the latter case, round j is either aborted by some other node or has value v since i is the largest round number smaller than j.

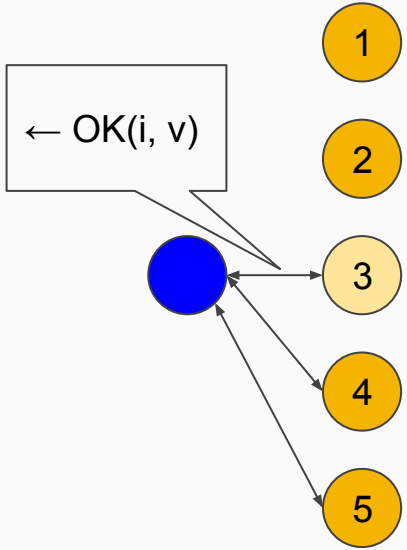
Agreement cont’



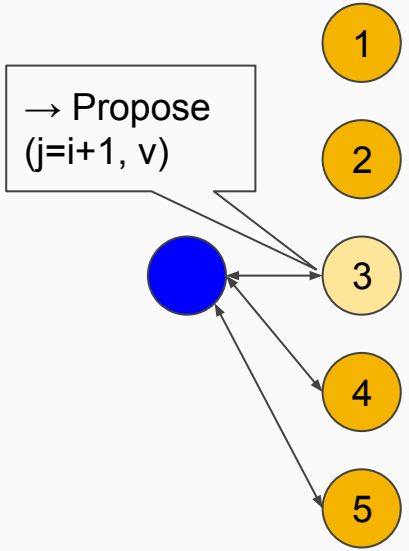
Round i: Prepare



Round i: Propose



Round j: Prepare



Round j: Propose

Agreement cont’

Induction case: j > i + 1, and all rounds i + 1, …, j - 1 satisfies the claim.

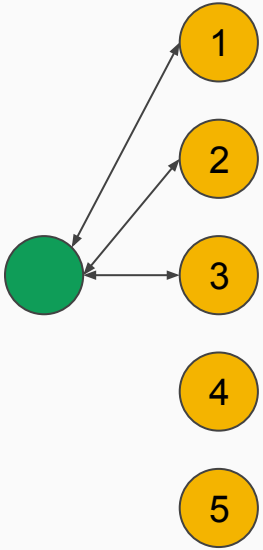
The only possible scenarios are the sane:

Prepare i (OK _ _) → Propose i v (Accept i v) → Prepare j (Abort)

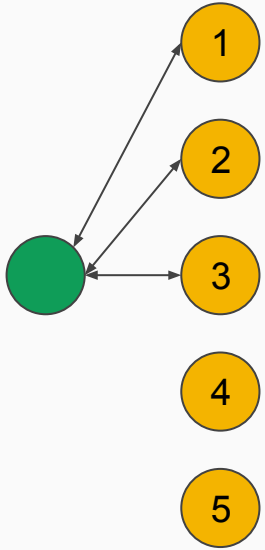
Prepare i (OK _ _) → Propose i v (Accept i v) → Prepare j (OK ? ?)

In the latter case, any node in the intersection will only propose a value it has accepted with a round number >= i.

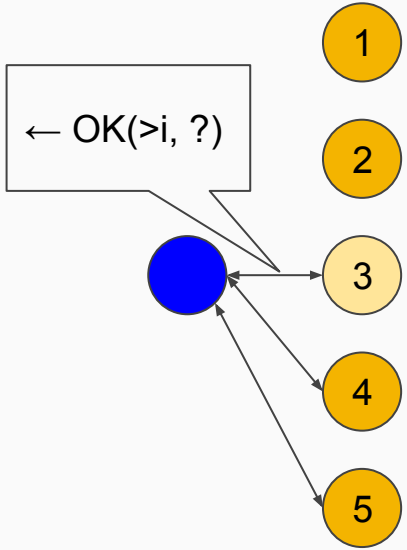
Agreement cont’



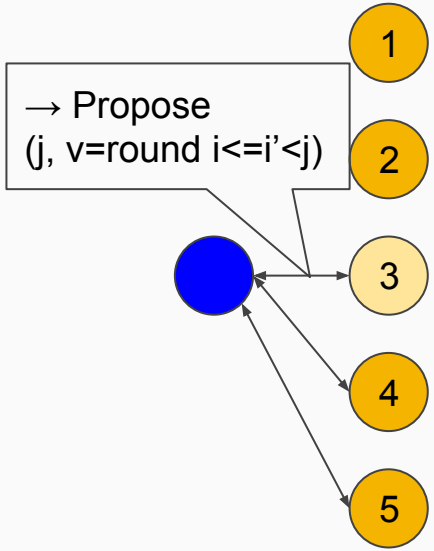
Round i: Prepare



Round i: Propose



Round j: Prepare



Round j: Propose

Agreement cont’

Thus round j can only propose a value accepted in round no earlier than i. By our IH, this value can only be v.

Liveness

For the single-decree synod to succeed, some round of 3 phase commit must succeed, which requires the leader and the majority of that round to be non-faulty. This gives us the 2f+1 bound.

However, even all nodes are non-faulty, we can still make no progress by having 2 conflicting leaders trying to outrun each other:

Liveness cont’

Leader 1: prepare 1 (Acceptors: OK)

Leader 2: prepare 2 (Acceptors: OK, abort 1)

Leader 1: propose 1(Acceptors: Aborted)

Leader 1: prepare 3 (Acceptors: OK, abort 2)

Leader 2: propose 2 (Acceptors: Aborted)

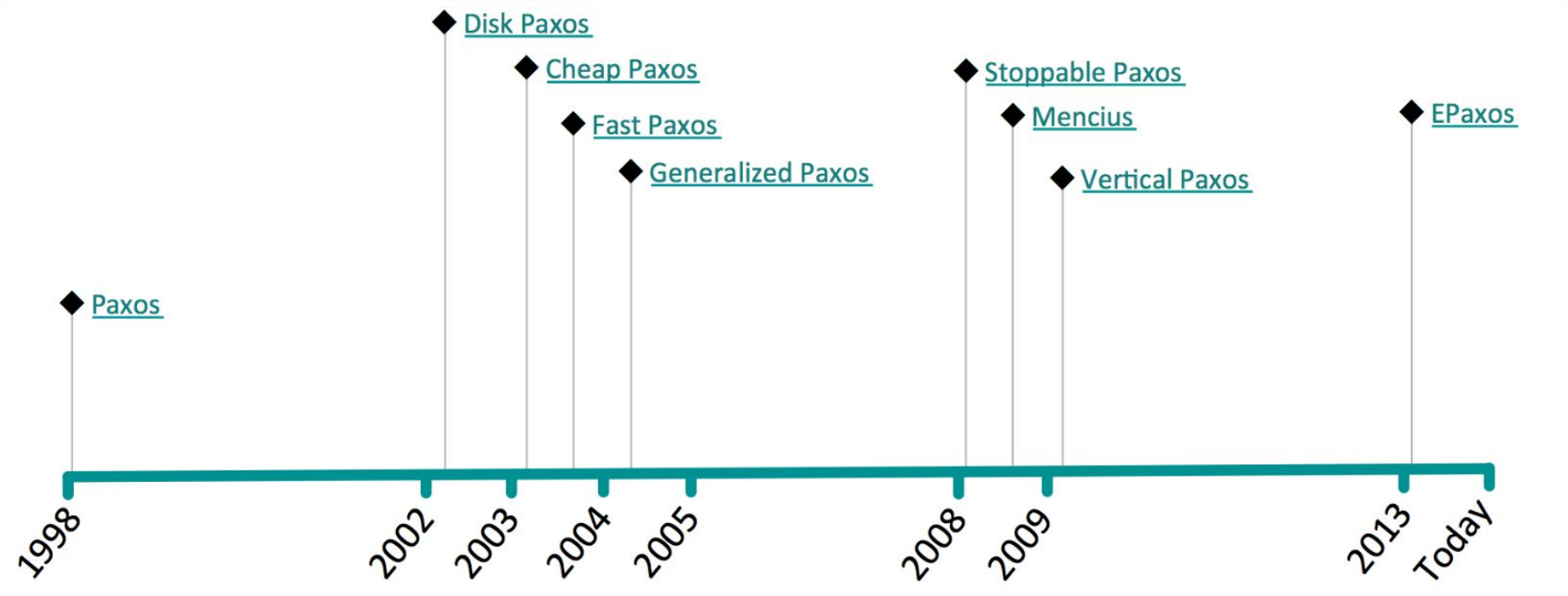
…

Liveness cont’

Due to FLP, we cannot prevent the latter scenario from happening if we want the leaders to be fault tolerant as well.

One can circumvent this by strengthen the assumptions: physical clocks, fault-detection, randomness etc and use a leader selection sub-protocol.

Paxos Variants http://paxos.systems/variants.html



Conclusion: Paxos

- Paxos is a consensus protocol that defends against f crash failures with 2f+1 replicas.

- It is a useful technique to implement state machine replication and has many variants which optimized its different aspects.

CORFU: A Distributed Shared Log

Corfu is another Mediterranean island on the North of Paxos

By Mahesh Balakrishnan, Dahlia Malkhi, John Davis, Vijayan Prabhakaran, Michael Wei, and Ted Wobber

CORFU: A Distributed Shared Log

It is a distributed log designed for high write throughput. (read can scale with the number of replicas)

A writer of the log first requests a new write position then write at the position without conflicts.

Possible holes due to slow writes/crashes are filled by empty writes.