Paxos:
A fault-tolerant consensus algorithm for highly-available distributed systems

Konrad Iwanicki
University of Warsaw

Supplement for Topic 08: Fault Tolerance
Distributed Systems Course
University of Warsaw

Based on multiple sources.
Consensus

• Assume a collection of processes that can propose values.

• A **consensus algorithm** ensures that only a single value among the proposed ones is chosen.
Consensus

- Assume a collection of processes that can propose values.
- A **consensus algorithm** ensures that only a single value among the proposed ones is chosen.
  - If no value is proposed, then no value should be chosen.
Consensus

- Assume a collection of processes that can propose values.
- A **consensus algorithm** ensures that only a single value among the proposed ones is chosen.
  - If no value is proposed, then no value should be chosen.
  - If a value has been chosen, the processes should be able to learn the chosen value.
Safety: nothing bad will ever happen
Safety: nothing bad will ever happen

- Only a value that has been proposed may be chosen.
Safety: nothing bad will ever happen

- Only a value that has been proposed may be chosen.
- Only a single value is chosen.
Safety: nothing bad will ever happen

• Only a value that has been proposed may be chosen.
• Only a single value is chosen.
• A process never learns that a value has been chosen unless it actually has been.
Liveness: something good will happen
Liveness: something good will happen

- Some proposed value is eventually chosen.
Liveness: something good will happen

- Some proposed value is eventually chosen.
- If a value has been chosen, a process can eventually learn the value.
System model

- Processes communicate by sending messages.
System model

• Processes communicate by sending messages.

• Processes:
  • can operate at arbitrary (different) speeds,
  • may fail (crash), but are not Byzantine,
  • may recover after failures, but need stable storage to this end.
System model

- Processes communicate by sending messages.
- Processes:
  - can operate at arbitrary (different) speeds,
  - may fail (crash), but are not Byzantine,
  - may recover after failures, but need stable storage to this end.
- Messages:
  - can take arbitrarily long to be delivered,
  - may be lost, reordered, or duplicated, but are never corrupted (i.e., the communication channels are not Byzantine).
System model

• Roles:
System model

• Roles:
  • **Proposer** – proposes values.
System model

• Roles:
  • **Proposer** – proposes values.
  • **Acceptor** – acts as a fault-tolerant memory of the protocol.
System model

• Roles:
  • **Proposer** – proposes values.
  • **Acceptor** – acts as a fault-tolerant memory of the protocol.
  • **Learner** – uses the learned values to perform some high level actions.
System model

• Roles:
  • **Proposer** – proposes values.
  • **Acceptor** – acts as a fault-tolerant memory of the protocol.
  • **Learner** – uses the learned values to perform some high level actions.

• A process typically plays all the roles, but this need not be the case.
Our way to Paxos

- We will derive the Paxos algorithm incrementally, by proposing subsequent improvements and properties.
First attempt

• There is a single acceptor, but there can be multiple proposers and learners.
First attempt

- There is a single acceptor, but there can be multiple proposers and learners.

- Algorithm:
  - A proposer sends its value to the acceptor.
  - The acceptor chooses the first proposed value it receives.
  - It sends the chosen value to the learners.
First attempt

- Problem: a single point of failure.
  - A failure of the acceptor makes any further progress impossible.
Deriving Paxos

• There must be multiple acceptors.

• Algorithm sketch:
  • A proposer sends its value to a set of acceptors.
  • An acceptor may accept the value.
  • The value is chosen when a large enough set of acceptors have accepted it.
  • The chosen value is sent to learners.
Deriving Paxos

• How large is “large enough”?
Deriving Paxos

• How large is “large enough”?

• To ensure that only a single value is chosen: a large enough set corresponds to any majority of acceptors.
Deriving Paxos

- How large is “large enough”?

- To ensure that only a single value is chosen: a large enough set corresponds to any majority of acceptors.
  - Any two majorities have at least one acceptor in common.
  - Hence, majority works if an acceptor can accept at most one value.
Deriving Paxos

- How to ensure that a value will be chosen even if there is only one proposer?
Deriving Paxos

• How to ensure that a value will be chosen even if there is only one proposer?

• Recall that:
  • Communication delays are unbounded.
  • Communication need not be reliable.
  • The proposer may be down (and its role has been taken over by another proposer).
Deriving Paxos

• How to ensure that a value will be chosen even if there is only one proposer?

• Recall that:
  • Communication delays are unbounded.
  • Communication need not be reliable.
  • The proposer may be down (and its role has been taken over by another proposer).

P1. An acceptor must accept the *first* proposal it receives.
But now we have another problem: different proposers can propose different values roughly at the same time.
Deriving Paxos

• But now we have another problem: different proposers can propose different values roughly at the same time.

• With P1, it may happen that:
  • Every acceptor has accepted a value, but none of these values has been accepted by a majority of the acceptors.
Deriving Paxos

• But now we have another problem: different proposers can propose different values roughly at the same time.

• With P1, it may happen that:
  • Every acceptor has accepted a value, but none of these values has been accepted by a majority of the acceptors.
  • A value is accepted by a majority of acceptors, but some of those acceptors are down for ever. Hence, if there is another value accepted by a minority of acceptors, which happen to be up, it may not be clear which of the values has been chosen.
Deriving Paxos

• The majority requirement and P1 suggest that:
  • An acceptor must be allowed to accept values from more than one proposer.
  • A proposer may attempt to propose a value multiple times to have it accepted by the majority.
Deriving Paxos

- The majority requirement and P1 suggest that:
  - An acceptor must be allowed to accept values from more than one proposer.
  - A proposer may attempt to propose a value multiple times to have it accepted by the majority.
- We need to keep track of the different proposed values.
Deriving Paxos

- Each proposed value, \( v \), is assigned a unique sequence number, \( n \): \textit{proposal} = a pair \((v, n)\).
Deriving Paxos

• Each proposed value, \( v \), is assigned a unique sequence number, \( n \): proposal = a pair \((v, n)\).

  • Subsequent values of \( n \) generated by the same proposer must be increasing.
Deriving Paxos

• Each proposed value, \( v \), is assigned a unique sequence number, \( n \): \textit{proposal} = a pair \((v, n)\).
  
  • Subsequent values of \( n \) generated by the same proposer must be increasing.
  
  • All proposal numbers must be totally ordered.
Deriving Paxos

- Each proposed value, $v$, is assigned a unique sequence number, $n$: \textbf{proposal} = a pair $(v, n)$.
  - Subsequent values of $n$ generated by the same proposer must be increasing.
  - All proposal numbers must be totally ordered.
- $n$ is typically itself a pair $<local\_n, prop\_id>$, where:
  - $local\_n$ – local number: 1, 2, 3, ...;
  - $prop\_id$ – is a unique ID of the proposer.
  - The pairs are totally ordered with a natural lexicographical ordering (assuming $prop\_ids$ are totally ordered).
Deriving Paxos

- Back to our algorithm sketch:
  - Value $v$ is chosen when a single proposal, $(v, n)$, has been accepted by a majority of acceptors.
  - In such case, we also say that the proposal, $(v, n)$, has been chosen.
Deriving Paxos

- Back to our algorithm sketch:
  - Value $v$ is chosen when a *single* proposal, $(v, n)$, has been accepted by a majority of acceptors.
  - In such case, we also say that the proposal, $(v, n)$, has been chosen.
- To ensure safety with multiple proposers:
  - We can allow *multiple* proposals to be chosen, but …
Deriving Paxos

● Back to our algorithm sketch:
  ● Value \( v \) is chosen when a *single* proposal, \((v, n)\), has been accepted by a majority of acceptors.
  ● In such case, we also say that the proposal, \((v, n)\), has been chosen.

● To ensure safety with multiple proposers:
  ● We can allow *multiple* proposals to be chosen, but …
  ● we must guarantee that **all of them have the same value.**
Deriving Paxos

• Back to our algorithm sketch:
  • Value $v$ is chosen when a single proposal, $(v, n)$, has been accepted by a majority of acceptors.
  • In such case, we also say that the proposal, $(v, n)$, has been chosen.

• To ensure safety with multiple proposers:
  • We can allow multiple proposals to be chosen, but ...
  • we must guarantee that all of them have the same value.

• By induction on $n$, it is sufficient to guarantee that:
  
  **P2. If a proposal with value $v$ is chosen, then every higher-numbered proposal that is chosen has value $v$.**

• Since proposal numbers are totally ordered, P2 guarantees that only a single value is chosen.
Deriving Paxos

**P2.** If a proposal with value $v$ is chosen, then every *higher-numbered* proposal that is chosen has value $v$. 
Deriving Paxos

P2. If a proposal with value \( v \) is chosen, then every higher-numbered proposal that is chosen has value \( v \).

- To be chosen, a proposal has to be accepted by at least one acceptor.
Deriving Paxos

\[ \textbf{P2.} \text{If a proposal with value } v \text{ is chosen, then every } \textit{higher-numbered} \text{ proposal that is chosen has value } v. \]

- To be chosen, a proposal has to be accepted by at least one acceptor.
- We can thus satisfy \textbf{P2} by satisfying:

\[ \textbf{P2a.} \text{If a proposal with value } v \text{ is chosen, then every } \textit{higher-numbered} \text{ proposal accepted by any acceptor has value } v. \]
Deriving Paxos

- We now have:
  
  **P1.** An acceptor must accept the *first* proposal it receives.

  **P2a.** If a proposal with value \( v \) is chosen, then every *higher-numbered* proposal accepted by *any* acceptor has value \( v \).
We now have:

P1. An acceptor must accept the first proposal it receives.

P2a. If a proposal with value $v$ is chosen, then every higher-numbered proposal accepted by any acceptor has value $v$.

A proposal can be chosen with some acceptor, A, never having received any proposal.

• (e.g., communication is not reliable).
We now have:

P1. An acceptor must accept the first proposal it receives.

P2a. If a proposal with value $v$ is chosen, then every higher-numbered proposal accepted by any acceptor has value $v$.

A proposal can be chosen with some acceptor, $A$, never having received any proposal.

• (e.g., communication is not reliable).

Suppose that a new proposer recovers and issues a higher-numbered proposal with a value different than $v$. 
We now have:

- A proposal can be chosen with some acceptor, A, never having received any proposal.
  - (e.g., communication is not reliable).
- Suppose that a new proposer recovers and issues a higher-numbered proposal with a value different than \( v \).
- **P1** requires A to accept this proposal, which violates **P2a**.

**P1.** An acceptor must accept the first proposal it receives.

**P2a.** If a proposal with value \( v \) is chosen, then every higher-numbered proposal accepted by any acceptor has value \( v \).
Deriving Paxos

• Maintaining both

| P1. | An acceptor must accept the first proposal it receives. |

• and

| P2a. | If a proposal with value $v$ is chosen, then every higher-numbered proposal accepted by any acceptor has value $v$. |

• requires strengthening P2a to:

| P2b. | If a proposal with value $v$ is chosen, then every higher-numbered proposal issued by any proposer has value $v$. |
Deriving Paxos

• Maintaining both

P1. An acceptor must accept the first proposal it receives.

• and

P2a. If a proposal with value \( v \) is chosen, then every higher-numbered proposal accepted by any acceptor has value \( v \).

• requires strengthening P2a to:

P2b. If a proposal with value \( v \) is chosen, then every higher-numbered proposal issued by any proposer has value \( v \).

• Since a proposal must be issued by a proposer before it can be accepted by an acceptor, P2b implies P2a.
Deriving Paxos

P2b. If a proposal with value $v$ is chosen, then every higher-numbered proposal issued by any proposer has value $v$.

- To invent an algorithm that satisfies P2b, let's think how we could prove that an algorithm satisfies P2b.
Deriving Paxos

**P2b.** If a proposal with value $v$ is chosen, then every higher-numbered proposal issued by any proposer has value $v$.

- To invent an algorithm that satisfies P2b, let's think how we could prove that an algorithm satisfies P2b.
- We could assume that a proposal, $(v, m)$, is chosen and show that for any issued proposal $(u, n)$ where $n > m$, we have $u = v$. 
Deriving Paxos

- How to do this?
Deriving Paxos

- How to do this? Use induction on $n$:
Deriving Paxos

- How to do this? Use induction on $n$:
  - Assumption: every proposal issued with number $m...n-1$ has value $v$. (*)
Deriving Paxos

How to do this? Use induction on $n$:

1. Assumption: every proposal issued with number $m...n-1$ has value $v$. (*)
2. Observation: for proposal $(v, m)$ to be chosen, there must be a set, $C$, consisting of a majority of acceptors who have accepted $(v, m)$. (**)

(*) and (**) refer to assumptions and observations within the context of deriving Paxos.
Deriving Paxos

- How to do this? Use induction on $n$:
  - Assumption: every proposal issued with number $m...n-1$ has value $v$. (*)
  - Observation: for proposal $(v, m)$ to be chosen, there must be a set, $C$, consisting of a majority of acceptors who have accepted $(v, m)$. (**) 
  - Combining (*) with (**), the hypothesis that $(v, m)$ is chosen implies that:
Deriving Paxos

- How to do this? Use induction on $n$:
  - Assumption: every proposal issued with number $m...n-1$ has value $v$. (*)
  - Observation: for proposal $(v, m)$ to be chosen, there must be a set, C, consisting of a majority of acceptors who have accepted $(v, m)$. (**) 
  - Combining (*) with (**), the hypothesis that $(v, m)$ is chosen implies that:
    - Every acceptor in C has accepted a proposal with number in $m...n-1$
Deriving Paxos

- How to do this? Use induction on $n$:
  - Assumption: every proposal issued with number $m...n-1$ has value $v$. (*)
  - Observation: for proposal $(v, m)$ to be chosen, there must be a set, C, consisting of a majority of acceptors who have accepted $(v, m)$. (**) 
  - Combining (*) with (**), the hypothesis that $(v, m)$ is chosen implies that:
    - Every acceptor in C has accepted a proposal with number in $m...n-1$ and
    - every proposal with number $m...n-1$ accepted by any acceptor has value $v$. 

Deriving Paxos

- How to do this? Use induction on $n$:
  - Assumption: every proposal issued with number $m...n-1$ has value $v$. (*)
  - Observation: for proposal $(v, m)$ to be chosen, there must be a set, $C$, consisting of a majority of acceptors who have accepted $(v, m)$. (**)
  - Combining (*) with (**), the hypothesis that $(v, m)$ is chosen implies that:
    - Every acceptor in $C$ has accepted a proposal with number in $m...n-1$ and
    - every proposal with number $m...n-1$ accepted by any acceptor has value $v$.
  - Since any set $S$ consisting of a majority of acceptors contains a member of $C$, we could conclude that for $(u, n), u = v$ by ensuring the invariant:
Deriving Paxos

- How to do this? Use induction on $n$:
  - Assumption: every proposal issued with number $m...n-1$ has value $v$. (*)
  - Observation: for proposal $(v, m)$ to be chosen, there must be a set, $C$, consisting of a majority of acceptors who have accepted $(v, m)$. (**) 
  - Combining (*) with (**), the hypothesis that $(v, m)$ is chosen implies that:
    - Every acceptor in $C$ has accepted a proposal with number in $m...n-1$ and 
    - every proposal with number $m...n-1$ accepted by any acceptor has value $v$.
  - Since any set $S$ consisting of a majority of acceptors contains a member of $C$, we could conclude that for $(u, n)$, $u = v$ by ensuring the invariant:

**P2c.** For any $v$ and $n$, if $(v, n)$ is issued, then there is a set, $S$, consisting of a majority of acceptors such that either (a) no acceptor in $S$ has accepted any proposal numbered less than $n$, or (b) $v$ is the value of the highest-numbered proposal among all proposals less than $n$ accepted by the acceptors in $S$.
Deriving Paxos

• We can thus satisfy:

**P2b.** If a proposal with value \( v \) is chosen, then every higher-numbered proposal issued by any proposer has value \( v \).

• by maintaining the invariance of:

**P2c.** For any \( v \) and \( n \), if \((v, n)\) is issued, then there is a set, \( S \), consisting of a majority of acceptors such that either (a) no acceptor in \( S \) has accepted any proposal numbered less than \( n \), or (b) \( v \) is the value of the highest-numbered proposal among all proposals less than \( n \) accepted by the acceptors in \( S \).
P2c. For any \( v \) and \( n \), if \((v, n)\) is issued, then there is a set, \( S \), consisting of a majority of acceptors such that either (a) no acceptor in \( S \) has accepted any proposal numbered less than \( n \), or (b) \( v \) is the value of the highest-numbered proposal among all proposals less than \( n \) accepted by the acceptors in \( S \).
P2c. For any \( v \) and \( n \), if \((v, n)\) is issued, then there is a set, \( S \), consisting of a majority of acceptors such that either (a) no acceptor in \( S \) has accepted any proposal numbered less than \( n \), or (b) \( v \) is the value of the highest-numbered proposal among all proposals less than \( n \) accepted by the acceptors in \( S \).

- To maintain the invariance of P2c, before issuing a proposal numbered \( n \), a proposer must learn the highest-numbered proposal with number less than \( n \) (if any) that has been or will be accepted by each acceptor in some majority of acceptors.
P2c. For any $v$ and $n$, if $(v, n)$ is issued, then there is a set, $S$, consisting of a majority of acceptors such that either (a) no acceptor in $S$ has accepted any proposal numbered less than $n$, or (b) $v$ is the value of the highest-numbered proposal among all proposals less than $n$ accepted by the acceptors in $S$.

- To maintain the invariance of P2c, before issuing a proposal numbered $n$, a proposer must learn the highest-numbered proposal with number less than $n$ (if any) that has been or will be accepted by each acceptor in some majority of acceptors.
  - Why “will be”? 

Deriving Paxos

**P2c.** For any $v$ and $n$, if $(v, n)$ is issued, then there is a set, $S$, consisting of a majority of acceptors such that either (a) no acceptor in $S$ has accepted any proposal numbered less than $n$, or (b) $v$ is the value of the highest-numbered proposal among all proposals less than $n$ accepted by the acceptors in $S$.

- To maintain the invariance of **P2c**, before issuing a proposal numbered $n$, a proposer must learn the highest-numbered proposal with number less than $n$ (if any) that has been or will be accepted by each acceptor in some majority of acceptors.
  - Why “will be”?
  - Because other proposer can be issuing their proposals at the same time.
Deriving Paxos

**P2c.** For any \( v \) and \( n \), if \( (v, n) \) is issued, then there is a set, \( S \), consisting of a majority of acceptors such that either (a) no acceptor in \( S \) has accepted any proposal numbered less than \( n \), or (b) \( v \) is the value of the highest-numbered proposal among all proposals less than \( n \) accepted by the acceptors in \( S \).

- To maintain the invariance of **P2c**, before issuing a proposal numbered \( n \), a proposer must learn the highest-numbered proposal with number less than \( n \) (if any) that has been or will be accepted by each acceptor in some majority of acceptors.
  - Why “will be”?
    - Because other proposer can be issuing their proposals at the same time.

- Learning about the accepted proposals is easy.
Deriving Paxos

**P2c.** For any \(v\) and \(n\), if \((v, n)\) is issued, then there is a set, \(S\), consisting of a majority of acceptors such that either (a) no acceptor in \(S\) has accepted any proposal numbered less than \(n\), or (b) \(v\) is the value of the highest-numbered proposal among all proposals less than \(n\) accepted by the acceptors in \(S\).

- To maintain the invariance of **P2c**, before issuing a proposal numbered \(n\), a proposer must learn the highest-numbered proposal with number less than \(n\) (if any) that has been or will be accepted by each acceptor in some majority of acceptors.
  - Why “will be”?
    - Because other proposer can be issuing their proposals at the same time.
- Learning about the accepted proposals is easy.
- Predicting future acceptances is hard.
Deriving Paxos

**P2c.** For any $v$ and $n$, if $(v, n)$ is issued, then there is a set, $S$, consisting of a majority of acceptors such that either (a) no acceptor in $S$ has accepted any proposal numbered less than $n$, or (b) $v$ is the value of the highest-numbered proposal among all proposals less than $n$ accepted by the acceptors in $S$.

- To maintain the invariance of **P2c**, before issuing a proposal numbered $n$, a proposer must learn the highest-numbered proposal with number less than $n$ (if any) that has been or will be accepted by each acceptor in some majority of acceptors.
  - Why “will be”?
    - Because other proposer can be issuing their proposals at the same time.
- Learning about the accepted proposals is easy.
- Predicting future acceptances is hard.
- Rather than predicting the future, let's create it!
Deriving Paxos

- To issue a proposal, a proposer:
Deriving Paxos

• To issue a proposal, a proposer:
  • Chooses a new proposal number, $n$, and sends a prepare request to a set of acceptors, asking each acceptor to respond with:
Deriving Paxos

- To issue a proposal, a proposer:
  - Chooses a new proposal number, $n$, and sends a *prepare request* to a set of acceptors, asking each acceptor to respond with:
    - the proposal with the highest number less than $n$ that the acceptor has accepted (if any)
Deriving Paxos

- To issue a proposal, a proposer:
  - Chooses a new proposal number, $n$, and sends a *prepare request* to a set of acceptors, asking each acceptor to respond with:
    - the proposal with the highest number less than $n$ that the acceptor has accepted (if any) and
    - a *promise* never again to accept a proposal numbered less than $n$. 
Deriving Paxos

To issue a proposal, a proposer:

- Chooses a new proposal number, \( n \), and sends a \textit{prepare request} to a set of acceptors, asking each acceptor to respond with:
  - the proposal with the highest number less than \( n \) that the acceptor has accepted (if any) and
  - a \textit{promise} never again to accept a proposal numbered less than \( n \).

- If the proposer receives responses from a majority of acceptors, it can issue a proposal with number \( n \) and value \( v \), where:
Deriving Paxos

● To issue a proposal, a proposer:

  ● Chooses a new proposal number, \( n \), and sends a \textit{prepare request} to a set of acceptors, asking each acceptor to respond with:

    – the proposal with the highest number less than \( n \) that the acceptor has accepted (if any) and
    – a \textit{promise} never again to accept a proposal numbered less than \( n \).

  ● If the proposer receives responses from a majority of acceptors, it can issue a proposal with number \( n \) and value \( v \), where:

    – \( v \) is the value of the highest-numbered proposal among the responses,
Deriving Paxos

- To issue a proposal, a proposer:
  - Chooses a new proposal number, \( n \), and sends a \textit{prepare request} to a set of acceptors, asking each acceptor to respond with:
    - the proposal with the highest number less than \( n \) that the acceptor has accepted (if any) and
    - a \textbf{promise} never again to accept a proposal numbered less than \( n \).
  - If the proposer receives responses from a majority of acceptors, it can issue a proposal with number \( n \) and value \( v \), where:
    - \( v \) is the value of the highest-numbered proposal among the responses,
    - or is any value selected by the proposer if the responders reported no proposals.
To issue a proposal, a proposer:

- Chooses a new proposal number, $n$, and sends a *prepare request* to a set of acceptors, asking each acceptor to respond with:
  - the proposal with the highest number less than $n$ that the acceptor has accepted (if any) and
  - a *promise* never again to accept a proposal numbered less than $n$.
- If the proposer receives responses from a majority of acceptors, it can issue a proposal with number $n$ and value $v$, where:
  - $v$ is the value of the highest-numbered proposal among the responses,
  - or is any value selected by the proposer if the responders reported no proposals.
- The proposer issues proposal $(v, n)$ by sending an accept request that the proposal be accepted to some set of acceptors (can be a different set than earlier).
Deriving Paxos

- What about an acceptor?
Deriving Paxos

• What about an acceptor?

• It can receive two types of request from proposers:
  • Prepare requests
  • Accept requests
Deriving Paxos

• What about an acceptor?
• It can receive two types of request from proposers:
  • Prepare requests
    – Always allowed to respond.
  • Accept requests
Deriving Paxos

• What about an acceptor?
• It can receive two types of request from proposers:
  • Prepare requests
    – Always allowed to respond.
  • Accept requests
    – Allowed to respond only iff it has not promised not to.
Deriving Paxos

- What about an acceptor?
- It can receive two types of request from proposers:
  - Prepare requests
    - Always allowed to respond.
  - Accept requests
    - Allowed to respond only iff it has not promised not to.

**P1a.** An acceptor can accept a proposal numbered $n$ iff it has not responded to a prepare request having a number greater than $n$. 
Deriving Paxos

• What about an acceptor?
• It can receive two types of request from proposers:
  • Prepare requests
    - Always allowed to respond.
  • Accept requests
    - Allowed to respond only iff it has not promised not to.

P1a. An acceptor can accept a proposal numbered $n$ iff it has not responded to a prepare request having a number greater than $n$.

• It can ignore any request without compromising safety.
Deriving Paxos

• An optimization:
  • Suppose that an acceptor
    – receives a prepare request numbered $n$, but
    – has already responded to a prepare request numbered greater than $n$. 
Deriving Paxos

• An optimization:
  • Suppose that an acceptor
    – receives a prepare request numbered $n$, but
    – has already responded to a prepare request numbered greater than $n$.
  • It can then ignore the request numbered $n$. 
Deriving Paxos

• An optimization:
  • Suppose that an acceptor
    – receives a prepare request numbered $n$, but
    – has already responded to a prepare request numbered greater than $n$.
  • It can then ignore the request numbered $n$.
  • It can also ignore a request for a proposal it has already accepted.
Deriving Paxos

• An optimization:
  • Suppose that an acceptor
    - receives a prepare request numbered $n$, but
    - has already responded to a prepare request numbered greater than $n$.
  • It can then ignore the request numbered $n$.
  • It can also ignore a request for a proposal it has already accepted.

• In effect:
  • An acceptor has to remember only:
    - The highest-numbered proposal it has ever accepted.
    - The highest-numbered prepare request to which it has responded.
Deriving Paxos

• An optimization:
  • Suppose that an acceptor
    – receives a prepare request numbered $n$, but
    – has already responded to a prepare request numbered greater than $n$.
  • It can then ignore the request numbered $n$.
  • It can also ignore a request for a proposal it has already accepted.

• In effect:
  • An acceptor has to remember only:
    – The highest-numbered proposal it has ever accepted.
    – The highest-numbered prepare request to which it has responded.
  • This information must be persistent (P2c under failures).
The Paxos algorithm

- Two phases:
  - **Phase 1**: preparing a proposal.
  - **Phase 2**: issuing and accepting the proposal.

**P1a.** An acceptor can accept a proposal numbered $n$ iff it has not responded to a prepare request having a number greater than $n$.

**P2c.** For any $v$ and $n$, if $(v, n)$ is issued, then there is a set, $S$, consisting of a majority of acceptors such that either (a) no acceptor in $S$ has accepted any proposal numbered less than $n$, or (b) $v$ is the value of the highest-numbered proposal among all proposals less than $n$ accepted by the acceptors in $S$. 
The Paxos algorithm: Phase 1

- A proposer:
The Paxos algorithm: Phase 1

- A proposer:
  - selects a proposal number, \( n \), and
The Paxos algorithm: Phase 1

- A proposer:
  - selects a proposal number, \( n \), and
  - sends a prepare request with number \( n \) to at least a majority of acceptors.
The Paxos algorithm: Phase 1

• A proposer:
  • selects a proposal number, $n$, and
  • sends a prepare request with number $n$ to at least a majority of acceptors.

• If an acceptor receives a prepare request:
The Paxos algorithm: Phase 1

- A proposer:
  - selects a proposal number, $n$, and
  - sends a prepare request with number $n$ to at least a majority of acceptors.

- If an acceptor receives a prepare request:
  - If the request's number $n$ is greater than that of any prepare request to which the acceptor has responded and
The Paxos algorithm: Phase 1

- A proposer:
  - selects a proposal number, \( n \), and
  - sends a prepare request with number \( n \) to at least a majority of acceptors.

- If an acceptor receives a prepare request:
  - If the request's number \( n \) is greater than that of any prepare request to which the acceptor has responded and
  - (This also implies that \( n \) is greater than that of any proposal which the acceptor has accepted),
The Paxos algorithm: Phase 1

- A proposer:
  - selects a proposal number, $n$, and
  - sends a prepare request with number $n$ to at least a majority of acceptors.

- If an acceptor receives a prepare request:
  - If the request's number $n$ is greater than that of any prepare request to which the acceptor has responded and
  - (This also implies that $n$ is greater than that of any proposal which the acceptor has accepted),
  - Then acceptor responds with:
    - A promise not to accept any proposals numbered less than $n$ and
    - The highest-numbered proposal it has accepted.
The Paxos algorithm: Phase 2

- If the proposer receives responses to its prepare requests numbered $n$ from a majority of acceptors,
The Paxos algorithm: Phase 2

• If the proposer receives responses to its prepare requests numbered \( n \) from a majority of acceptors,
  • then it sends an accept request for proposal \((v, n)\) to each of those acceptors, where \( v \):
    − is the value of the highest-numbered proposal among the responses or
    − is any value if the responses reported no proposals.
The Paxos algorithm: Phase 2

- If the proposer receives responses to its prepare requests numbered $n$ from a majority of acceptors,
  - then it sends an accept request for proposal $(v, n)$ to each of those acceptors, where $v$:
    - is the value of the highest-numbered proposal among the responses or
    - is any value if the responses reported no proposals.

- If an acceptor receives an accept request for proposal $(v, n)$,
The Paxos algorithm: Phase 2

- If the proposer receives responses to its prepare requests numbered \( n \) from a majority of acceptors,
  - then it sends an accept request for proposal \((v, n)\) to each of those acceptors, where \( v \):
    - is the value of the highest-numbered proposal among the responses or
    - is any value if the responses reported no proposals.
- If an acceptor receives an accept request for proposal \((v, n)\),
  - then, it accepts the proposal,
The Paxos algorithm: Phase 2

- If the proposer receives responses to its prepare requests numbered \( n \) from a majority of acceptors,
  - then it sends an accept request for proposal \((v, n)\) to each of those acceptors, where \( v \):
    - is the value of the highest-numbered proposal among the responses or
    - is any value if the responses reported no proposals.

- If an acceptor receives an accept request for proposal \((v, n)\),
  - then, it accepts the proposal,
  - unless it has already accepted it or responded to a prepare request with a number greater than \( n \).
The Paxos algorithm: Remarks

• A proposer can make multiple proposals (but must follow the algorithm).

• It can abandon a proposal at any time.
  • e.g., when it can learn that the proposal will not pass.
The Paxos algorithm: Learning

- To learn that a value has been chosen, a learner must find out that a proposal has been accepted by a majority of acceptors.
- However, recall that:
  - Processes can crash,
  - Messages can be delayed or lost.
The Paxos algorithm: Learning

• Possible approaches to learning the chosen value:
The Paxos algorithm: Learning

- Possible approaches to learning the chosen value:
  - Whenever an acceptor accepts a proposal, it informs all learners.
The Paxos algorithm: Learning

- Possible approaches to learning the chosen value:
  - Whenever an acceptor accepts a proposal, it informs all learners.
  - Whenever an acceptor accepts a proposal, it informs a distinguished learner; the learner, in turn, informs all other learners.
The Paxos algorithm: Learning

- Possible approaches to learning the chosen value:
  - Whenever an acceptor accepts a proposal, it informs all learners.
  - Whenever an acceptor accepts a proposal, it informs a distinguished learner; the learner, in turn, informs all other learners.
  - Whenever an acceptor accepts a proposal, it informs a set of distinguished learners; these learners, in turn, inform all other learners.
The Paxos algorithm: Learning

- Possible approaches to learning the chosen value:
  - Whenever an acceptor accepts a proposal, it informs all learners.
  - Whenever an acceptor accepts a proposal, it informs a distinguished learner; the learner, in turn, informs all other learners.
  - Whenever an acceptor accepts a proposal, it informs a set of distinguished learners; these learners, in turn, inform all other learners.
  - A learner can ask acceptors what proposals they have accepted.
The Paxos algorithm: Learning

• Possible approaches to learning the chosen value:
  • Whenever an acceptor accepts a proposal, it informs all learners.
  • Whenever an acceptor accepts a proposal, it informs a distinguished learner; the learner, in turn, informs all other learners.
  • Whenever an acceptor accepts a proposal, it informs a set of distinguished learners; these learners, in turn, inform all other learners.
  • A learner can ask acceptors what proposals they have accepted.
  • A learner can have a proposer/multiple proposers issue a new proposal.
The Paxos algorithm: Progress
The Paxos algorithm: Progress

- Even just two proposer can inhibit any progress.
The Paxos algorithm: Progress

- Even just two proposer can inhibit any progress.
- Solution: have a distinguished proposer.
The Paxos algorithm: Progress

- Even just two proposer can inhibit any progress.
- Solution: have a distinguished proposer.
  - If enough of the system is working properly for a sufficient amount of time, this ensures liveness.
The Paxos algorithm: Progress

- Even just two proposer can inhibit any progress.
- Solution: have a distinguished proposer.
  - If enough of the system is working properly for a sufficient amount of time, this ensures liveness.
  - However, the election requires using randomness or real time (timeouts).
The Paxos algorithm: Progress

• Even just two proposer can inhibit any progress.

• Solution: have a distinguished proposer.
  • If enough of the system is working properly for a sufficient amount of time, this ensures liveness.
  • However, the election requires using randomness or real time (timeouts).
    – The protocol will thus no longer be asynchronous.
The Paxos algorithm: Progress

- Even just two proposer can inhibit any progress.
- Solution: have a distinguished proposer.
  - If enough of the system is working properly for a sufficient amount of time, this ensures liveness.
  - However, the election requires using randomness or real time (timeouts).
    - The protocol will thus no longer be asynchronous...
    - But this is correct, because no asynchronous consensus with our assumptions is feasible.
Classic Paxos Application

• Consider a server that operates as a deterministic state machine:
  • The server has a current state.
  • A client issues a command to the server.
  • The server executes the command and as a result transits to a new state and produces an output for the client.
• Multiple clients send their commands concurrently, but the server executes these commands one after another.
Classic Paxos Application

- To ensure fault tolerance, the state machine has to be replicated on multiple servers.
- Each of the servers implement the state machine independently.
- However, since the machine is deterministic, if all the servers see the clients' commands in the same order, they all perform the same state transitions and produce the same output.
- A client can then use the output generated for it by any server.
- This approach is called **state machine replication**.
State machine replication

- To guarantee that all servers see the same sequence of commands, we use Paxos:
  - One instance of the Paxos consensus algorithm per command.
  - The value chosen by the $i^{th}$ instance is the $i^{th}$ state machine command in the sequence.
  - Every server plays all roles: proposer, acceptor, learner.
  - One server is elected to be a leader: a distinguished proposer in all instances of the algorithm.
State machine replication

- Clients send commands to the leader.
- The leader decides at which position in the sequence the commands appear.
- If the leader decides that a given command should be $i^{th}$, it tries to have that command chosen as the value of the $i^{th}$ instance of the consensus algorithm.
  - It will usually succeed.
  - It may fail because of failures.
  - It may fail also because another server believes it is the leader.
  - In any case, however, the consensus algorithm ensures that at most one command is chosen as the $i^{th}$ one.
State machine replication

• What is important for efficiency is that:
  • In any instance of Paxos, no command need to be chosen before Phase 2.
  • Put differently, having completed Phase 1 of a Paxos instance, the leader is normally free to propose any command.
State machine replication

- Suppose that the previous leader has failed, and a new one has just been elected.
- Being a learner in all instances of Paxos, the new leader knows most of the commands that have already been chosen in various instances.
- Suppose those instances are:
  - 1...31,
  - 35, and
  - 36.
- What should the leader do?
State machine replication

- It executes Phase 1 for the remaining instances:
  - 32...34,
  - and all instances greater than 36.
- What can the outcome be?
- Some instances may have their commands chosen.
  - Suppose those are 32 and 37.
- Others must have no associated commands.
  - Those are: 33, 34, and all greater than 37.
State machine replication

- The leader can than (re)execute Phase 2 for instances 32 and 37, thereby choosing their commands.

- The leader, and any other process that learns all the commands the leader knows:
  - can execute commands 1...32,
  - cannot execute subsequent commands, because command 33 and 34 has not been chosen.

- What should the leader do?
  - It can wait for subsequent client commands.
  - It can make 33 and 34 commands “NO OP”.
State machine replication

- In effect, commands 1...37 can be executed.
- The leader has also completed Phase 1 for instances greater than 37.
- It can wait for subsequent clients commands and propose them in Phase 2 of those instances.
  - Commands need not be chosen in the order of instances.
- Since we execute Phase 1 for infinitely many instances, the cost of the consensus is only Phase 2.
Final remarks

• There are many other issues:
  • Node composition.
  • Leader election.
  • Dozens of optimizations.

• Implementing Paxos in a real system requires expertise.
Final remarks

• Nevertheless, Paxos gains popularity:
  • Chubby lock service.
  • Petal: distributed virtual disks.
  • Frangipani: a distributed file system.
  • Spanner: globally distributed database.
  • ...