Blockchain Honor Degree Sem VI

HBCC 601 : Blockchain Platforms

Module - 4 : Private Blockchain (6 Hours)

Instructor : Mrs. Lifna C S

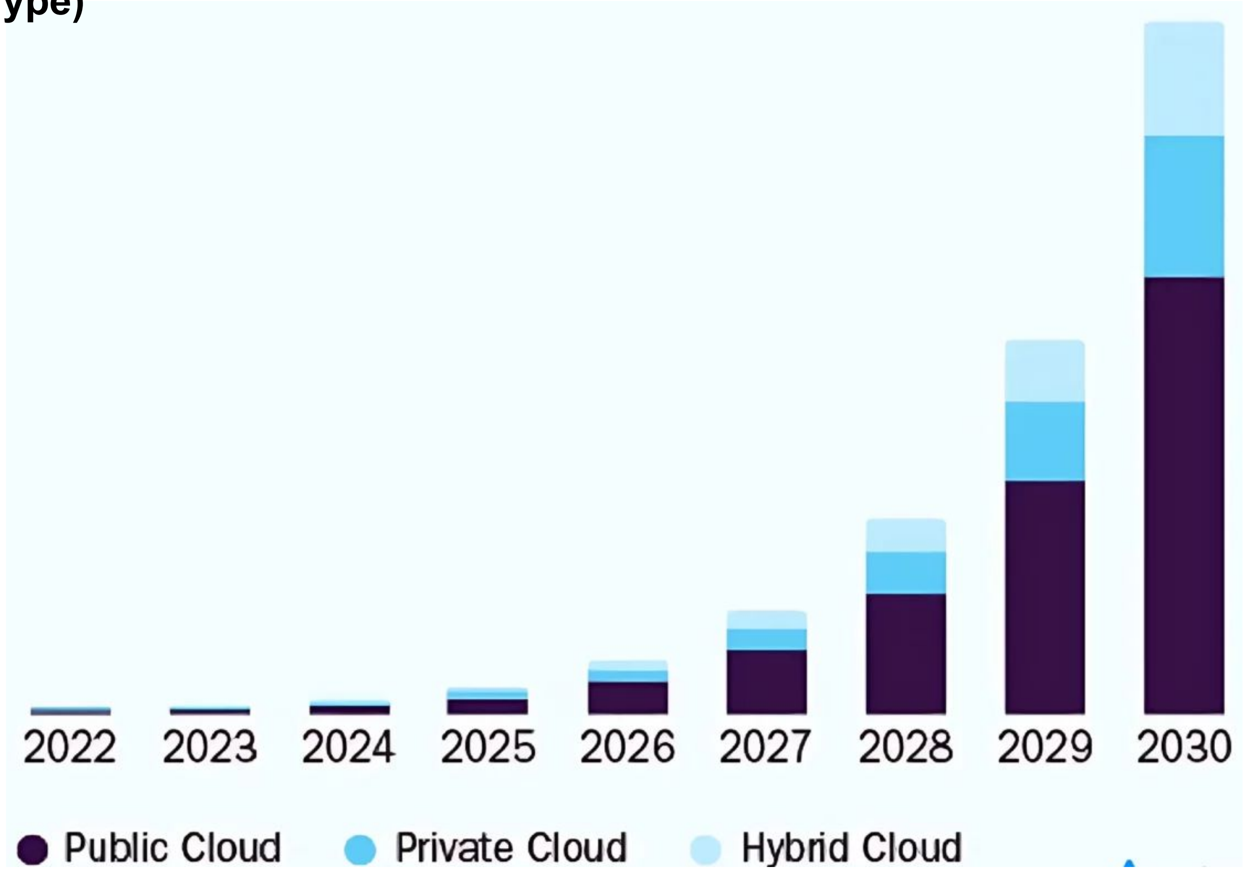
Topics to be covered

- Introduction to Private Blockchain
- Key Characteristics of Private Blockchain
- Need of Private Blockchain.
- Consensus Algorithm for Private Blockchain
  - PAXOS, RAFT
- Smart Contract in Private Blockchain
- Steps to build a Private Blockchain Platform
- Advantages of Private Blockchain for Businesses
- Examples of Private Blockchain
- Use Cases of Private Blockchain
- Best Practices for using a Private Blockchain
- Public Blockchain Vs Private Blockchain

Self-learning Topics: Case study on private Blockchain.

U. S Blockchain Technology Market : 2020-2023 (USD Billion)

(w.r.t Size & Type)



Courtesy : appinventiv

Introduction to Private Blockchain

- Decentralized and distributed digital ledger that **operates within a restricted ecosystem**, accessible only to trusted participants.
- Offer a **more exclusive and secure environment**, ideal for businesses and organizations **seeking confidentiality and control over their data.**
- For businesses, access to the network is confidential, and all the **participants must be granted explicit authorization** to contribute to it.
  - This access control safeguards sensitive information from unauthorized access and manipulation from external malware practices and thefts.
- Widely **used in industries prioritizing data privacy and security,** including finance, healthcare, supply chain management, and government sectors.
  - **Financial institutions** employ private blockchains to enable secure and efficient cross-border transactions among authorized parties, improving operational efficiency while complying with regulations.

Courtesy : appinventiv

Key Characteristics Private Blockchain

1. **Permissioned Access:**
   - access is restricted to known entities or participants.
   - participants are typically known and identified, ensuring a higher level of trust within the network compared to public blockchains.
2. **Centralized or Consortium Control:**
   - controlled either by a single entity (centralized) or a consortium of several organizations (consortium).
   - In a consortium model, multiple organizations agree to collaborate and share authority over the network.
3. **Faster Transaction Processing:**
   - Limited number of participants, they can achieve higher transaction throughput compared to public blockchains.
   - there are fewer nodes involved in reaching consensus, leading to faster transaction processing times.
4. **Enhanced Privacy and Confidentiality:**
   - Often prioritize privacy and confidentiality of data.
   - Participants have control over who can access and view transactions, ensuring sensitive information is not visible to unauthorized parties.

Key Characteristics Private Blockchain

5. **Customizable Consensus Mechanisms:**
    - Organizations are allowed to implement consensus mechanisms tailored to their specific requirements. This flexibility enables greater efficiency and scalability, as consensus algorithms can be optimized for the network's needs.
6. **Enterprise Applications**:
    - widely adopted in enterprise settings, where companies require more control over their blockchain networks.
    - Industries such as finance, supply chain management, healthcare, and logistics utilize private blockchains to streamline operations, reduce costs, and enhance security.
7. **Smart Contract Functionality**:
    - To automate business processes and facilitate transactions without the need for intermediaries, improving efficiency and reducing friction.
8. **Regulatory Compliance:**
    - Designed to comply with specific regulations governing industries or jurisdictions.
    - Participants can enforce rules and regulations within the network, ensuring legal compliance and reducing regulatory risks.

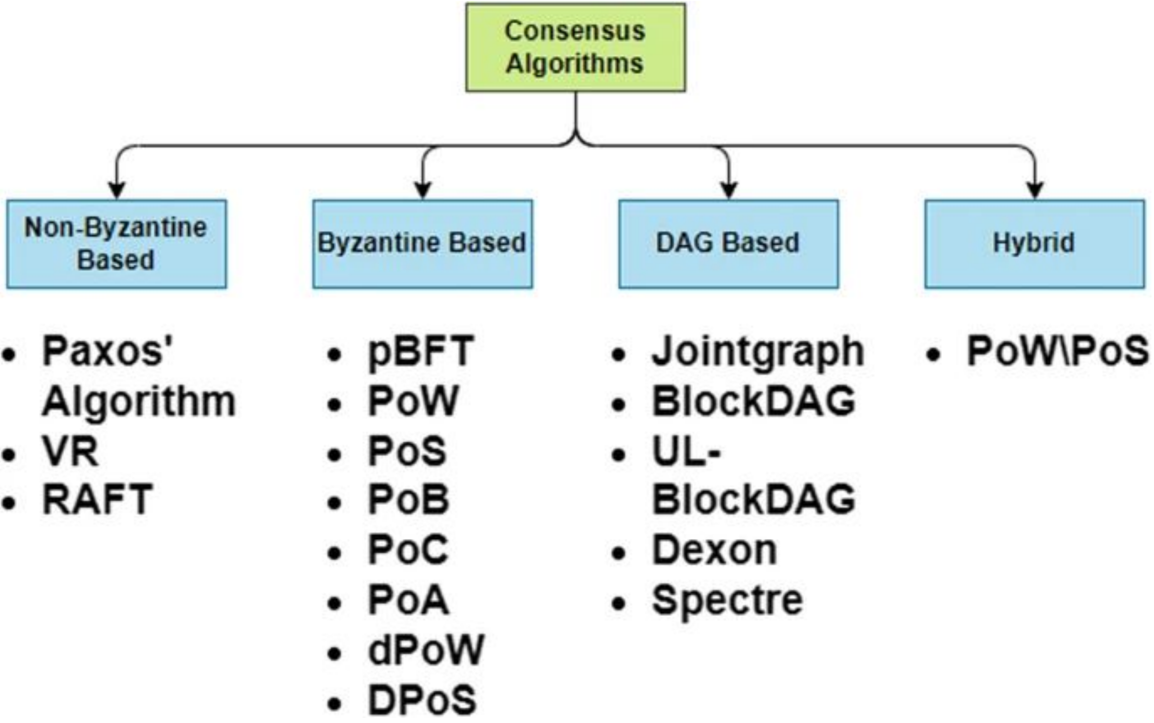
Need for Private Blockchain

1. **Controlled Access:**
   - Enable organizations to restrict access to known and trusted participants.
   - Ensures that **sensitive data and operations are kept within a closed network, enhancing security and confidentiality.**
2. **Privacy and Confidentiality:**
   - **maintaining privacy and confidentiality of data is paramount**.
   - **Allow participants to control who can view and access transactions, protecting sensitive information from unauthorized disclosure.**
3. **Scalability:**
   - Challenge due to the large number of participants and the consensus mechanisms employed.
   - Permissioned nature and customizable consensus algorithms, can be designed **to achieve higher transaction throughput and lower latency,** thus addressing scalability concerns.
4. **Customizable Governance:**
   - Offer flexibility in governance structures.
   - Organizations can establish **governance models tailored to their specific needs**, whether it's a centralized approach with a single controlling entity or a consortium model where multiple organizations collaborate and share authority over the network.

Need for Private Blockchain

5. **Regulatory Compliance:**
   - A significant concern for many industries.
   - Designed to adhere to specific regulations governing data privacy, financial transactions, and other legal requirements.
   - **Participants can enforce rules and policies** within the network, ensuring regulatory compliance.
6. **Efficiency and Cost Reduction:**
   - **Streamline business processes, reduce operational costs, and eliminate intermediaries** by facilitating direct peer-to-peer transactions and automating workflows through smart contracts.
     - **Leads to increased efficiency and improved resource utilization** within organizations.
7. **Interoperability:**
   - Designed to integrate with existing systems and legacy infrastructure, enabling seamless interoperability with other enterprise applications and databases.
   - Allows organizations to leverage the benefits of blockchain technology while **maintaining compatibility with their existing IT environments.**

Consensus Algorithms



Consensus Algorithms

Non-Byzantine Based

- Paxos' Algorithm
- VR
- RAFT

Byzantine Based

- pBFT
- PoW
- PoS
- PoB
- PoC
- PoA
- dPoW
- DPoS

DAG Based

- Jointgraph
- BlockDAG
- UL-BlockDAG
- Dexon
- Spectre

Hybrid

- PoW\PoS

Consensus

- used <u>to guarantee that all nodes on the network agree on the current state</u> of the network and the <u>authenticity of transactions</u>
- vital <u>for preserving the security and integrity</u> of the blockchain.

Courtesy : Springer

Consensus Algorithm in Private Blockchain

- Private blockchains often **prioritize efficiency, scalability, and controlled governance.**
- Choice of consensus algorithm plays a crucial role in determining the network's performance, security, and governance.
- Some of the consensus mechanism used :
  - **Practical Byzantine Fault Tolerance (PBFT)**
  - **Proof of Authority (PoA)**
  - **Raft Consensus**
  - **Proof of Elapsed Time (PoET)**
  - **Quorum Byzantine Fault Tolerance (QBFT)**
  - **Proof of Weight (PoWeight)**

Consensus Algorithm in Private Blockchain

1. **Practical Byzantine Fault Tolerance (PBFT)**
   - Classic consensus algorithm designed for permissioned blockchain networks.
   - It ensures consensus among a set of known, trusted nodes by requiring a two-step process of pre-prepare, prepare, and commit messages.
   - Offers **high throughput and low latency**, making it suitable for enterprise applications where a predefined set of participants can be trusted.
2. **Proof of Authority (PoA):**
   - Block validators are identified and authorized to create new blocks.
   - Validators are typically **known and trusted entities,** such as consortium members or network administrators.
   - **Ensures fast block finality and high transaction throughput,** making it suitable for private blockchains where centralized control is acceptable.

Consensus Algorithm in Private Blockchain

3. **Raft Consensus:**
   - Designed for **fault-tolerant distributed systems,** including private blockchains.
   - Elects a leader among a group of nodes responsible for managing the consensus process.
   - Offers **simplicity, fault tolerance, and efficient leader election,** making it suitable for scenarios where quick consensus and high availability are required.
4. **Proof of Elapsed Time (PoET):**
   - Developed by Intel
   - used in permissioned blockchain networks.
   - Relies on a trusted execution environment (TEE) to select a **leader node randomly** based on a wait time determined by cryptographic processes.
   - Ensures fair leader election and energy efficiency, making it suitable for private blockchains deployed in environments with limited resources.

Consensus Algorithm in Private Blockchain

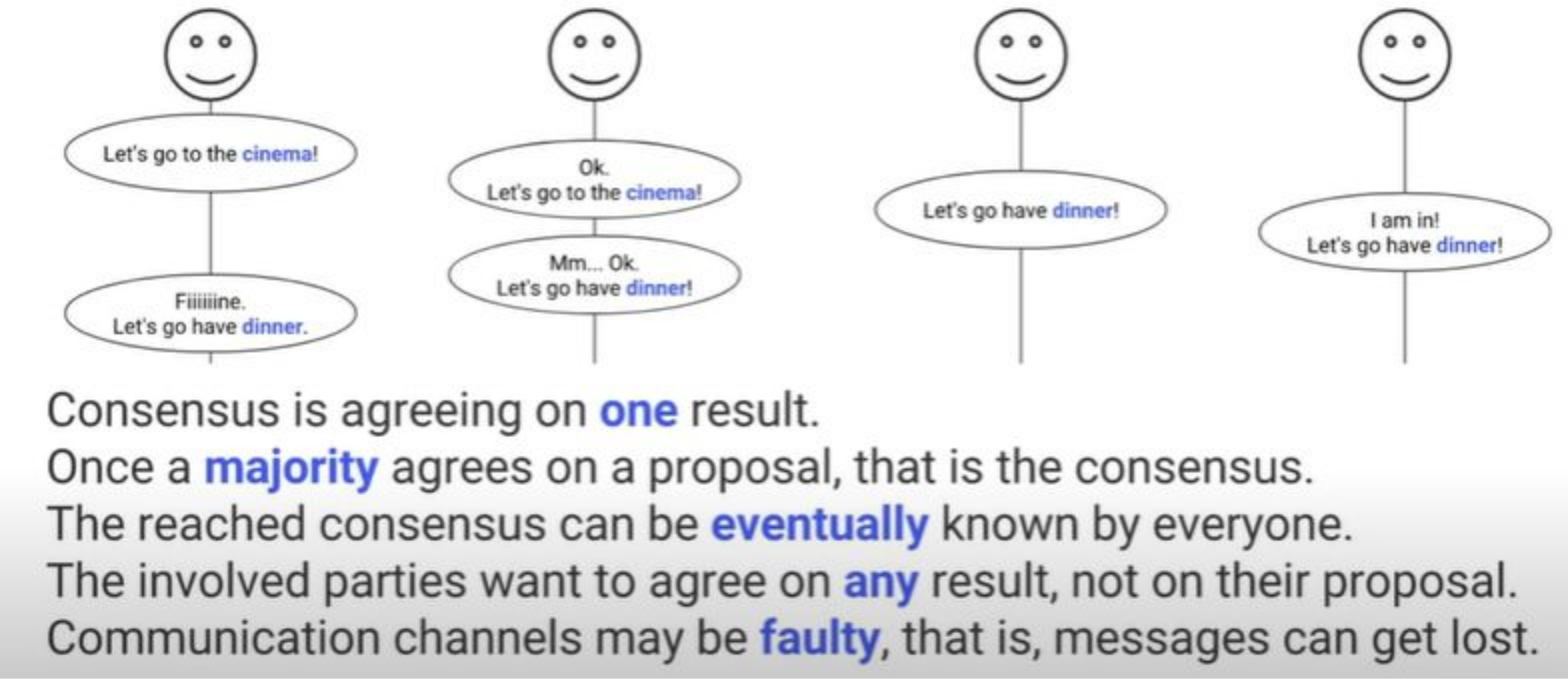
5. **Quorum Byzantine Fault Tolerance (QBFT):**
   - implemented in Quorum,
   - a permissioned blockchain platform built on Ethereum.
   - **Extends PBFT** to **support privacy and confidentiality** features while maintaining **high throughput and low latency.**
   - Ensures consensus among known, trusted nodes in a permissioned network, making it suitable for enterprise applications.
6. **Proof of Weight (PoWeight):**
   - Designed for private blockchain networks
   - **Participants with higher stake or reputation** have greater influence over the consensus process, ensuring security and fairness.
   - Suitable for scenarios where participants' credibility can be verified and weighted accordingly.

Consensus Algorithms - PAXOS Algorithm

- Family of protocols designed for achieving consensus in distributed systems.
- Introduced by **Leslie Lamport (**1998)
- Primary goal of Paxos :
    - To ensure that a group of nodes, despite potential failures and network partitions, can agree on a single value.
    - This is a fundamental problem in distributed computing, particularly in scenarios where nodes must reach agreement on decisions, such as in replicated state machines or distributed databases.
- Operates in several phases, involving a set of nodes (often referred to as acceptors or replicas) that communicate and coordinate to reach consensus.
    - Phase 1: Prepare (Prepare Phase)
    - Phase 2: Accept (Accept Phase)
    - Phase 3: Learn (Learn Phase)

Courtesy : Google Talks

How to reach consensus with PAXOS?



Consensus is agreeing on **one** result.

Once a **majority** agrees on a proposal, that is the consensus.

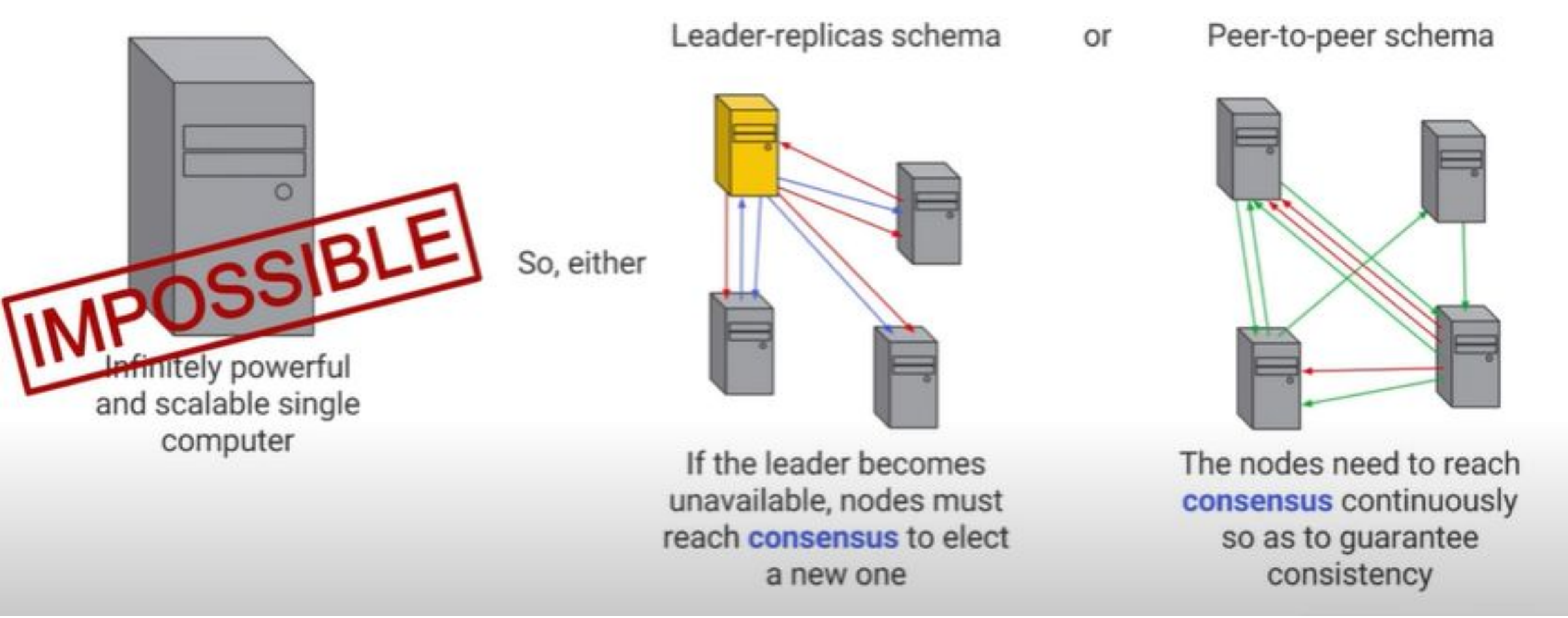
The reached consensus can be **eventually** known by everyone.

The involved parties want to agree on **any** result, not on their proposal.

Communication channels may be **faulty**, that is, messages can get lost.

Courtesy : Google Talks

Why do systems need to reach consensus?



Infinitely powerful and scalable single computer

So, either

Leader-replicas schema

If the leader becomes unavailable, nodes must reach **consensus** to elect a new one

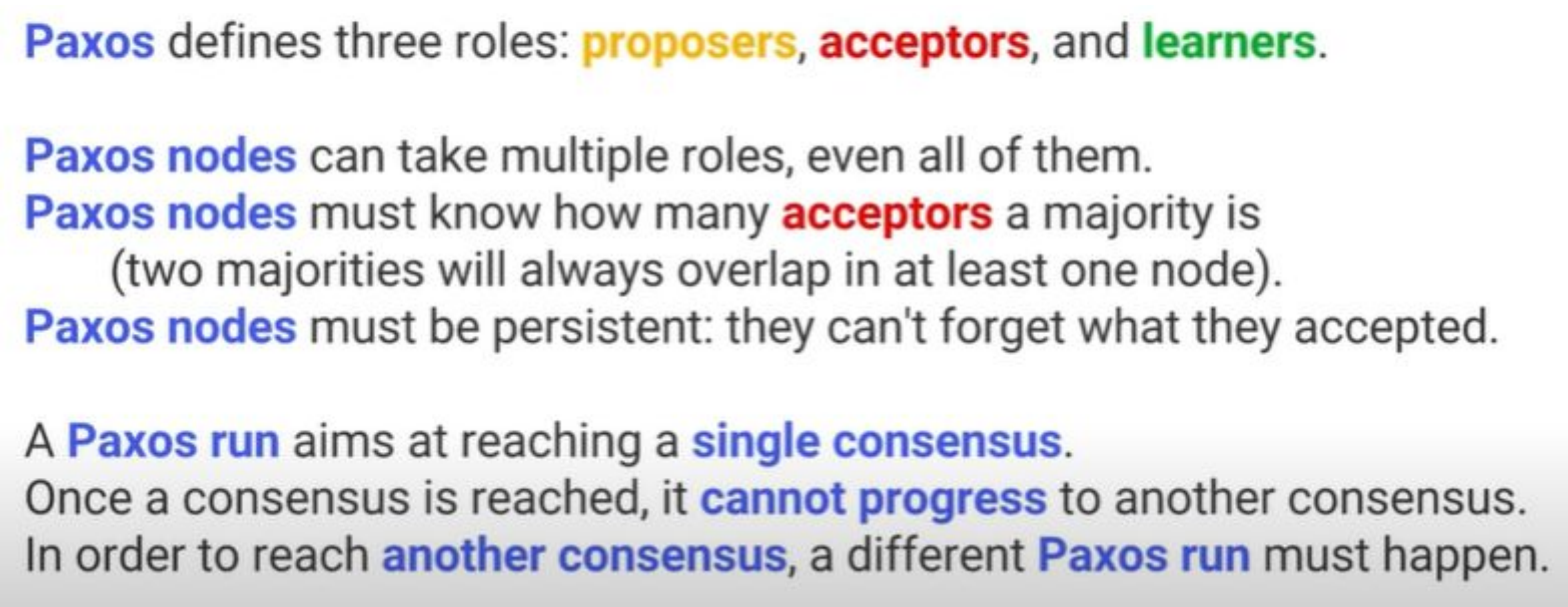
or

Peer-to-peer schema

The nodes need to reach **consensus** continuously so as to guarantee consistency

Courtesy : Google Talks

PAXOS Basics



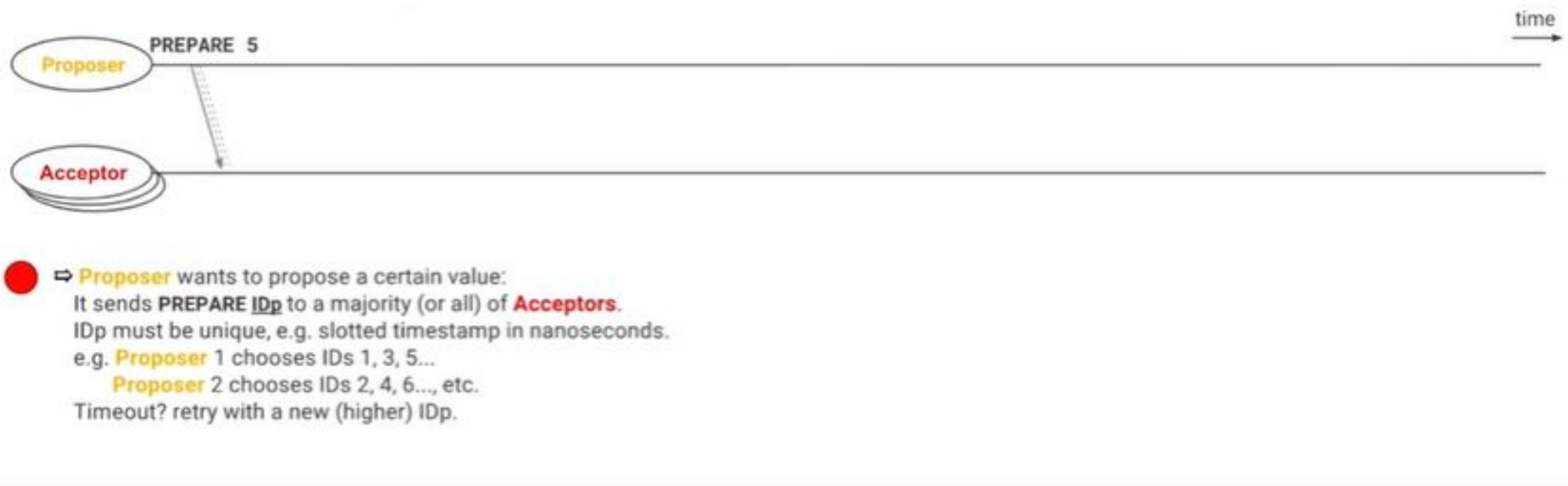
Paxos defines three roles: proposers, acceptors, and learners.

Paxos nodes can take multiple roles, even all of them.
Paxos nodes must know how many acceptors a majority is
(two majorities will always overlap in at least one node).
Paxos nodes must be persistent: they can't forget what they accepted.

A Paxos run aims at reaching a single consensus.
Once a consensus is reached, it cannot progress to another consensus.
In order to reach another consensus, a different Paxos run must happen.

Courtesy : Google Talks

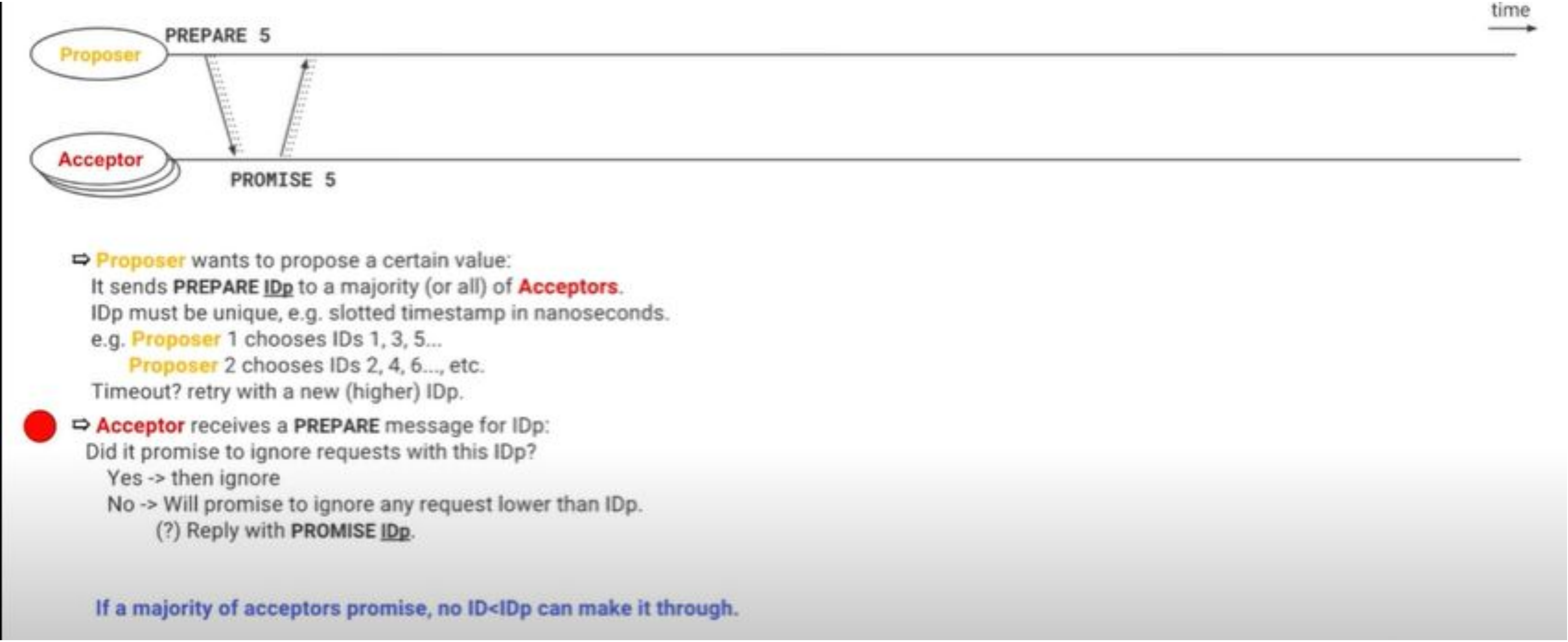
PAXOS Algorithm



● ⇨ Proposer wants to propose a certain value:
It sends **PREPARE IDp** to a majority (or all) of **Acceptors**.
IDp must be unique, e.g. slotted timestamp in nanoseconds.
e.g. Proposer 1 chooses IDs 1, 3, 5...
Proposer 2 chooses IDs 2, 4, 6..., etc.
Timeout? retry with a new (higher) IDp.

Courtesy : Google Talks

PAXOS Algorithm



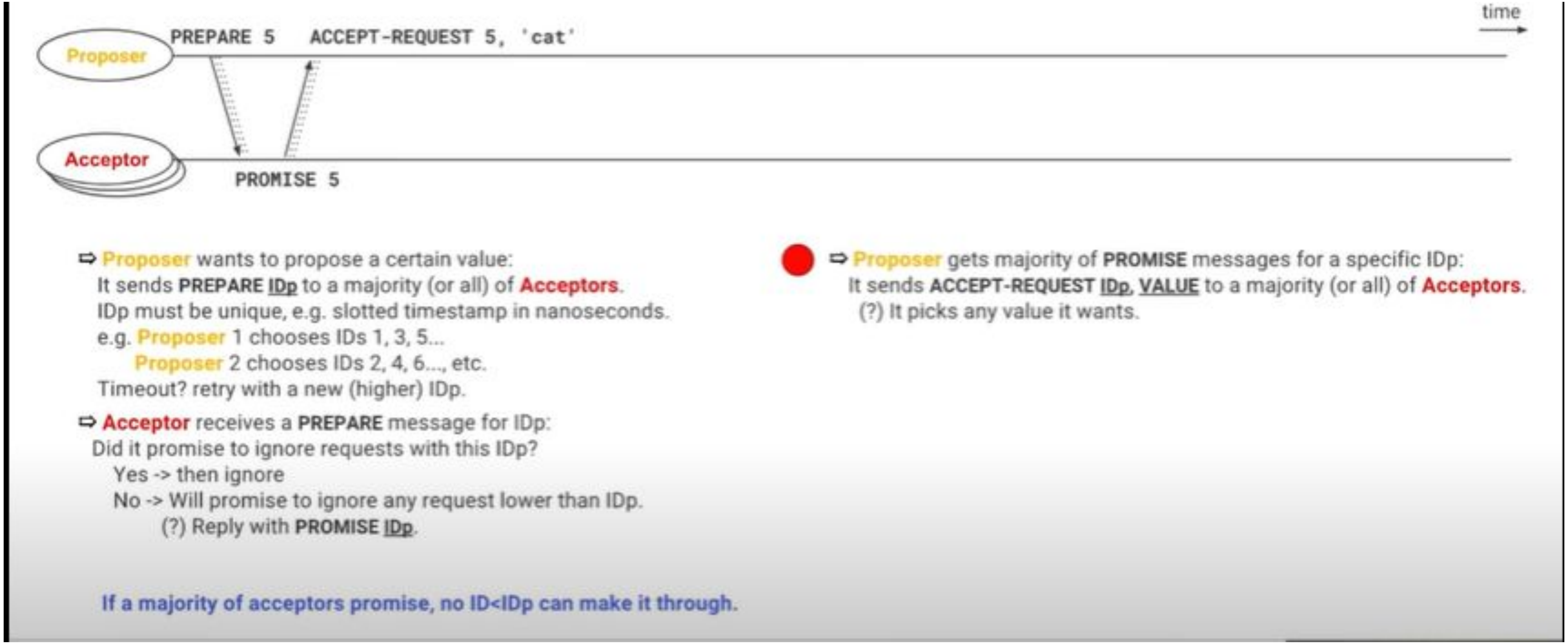
⇨ Proposer wants to propose a certain value:
It sends **PREPARE** **IDp** to a majority (or all) of **Acceptors**.
IDp must be unique, e.g. slotted timestamp in nanoseconds.
e.g. Proposer 1 chooses IDs 1, 3, 5...
Proposer 2 chooses IDs 2, 4, 6..., etc.
Timeout? retry with a new (higher) IDp.

⇨ **Acceptor** receives a **PREPARE** message for IDp:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Will promise to ignore any request lower than IDp.
(?) Reply with **PROMISE** **IDp**.

If a majority of acceptors promise, no ID<IDp can make it through.

Courtesy : Google Talks

PAXOS Algorithm



⇨ Proposer wants to propose a certain value:
It sends **PREPARE** **IDp** to a majority (or all) of **Acceptors**.
IDp must be unique, e.g. slotted timestamp in nanoseconds.
e.g. Proposer 1 chooses IDs 1, 3, 5...
Proposer 2 chooses IDs 2, 4, 6..., etc.
Timeout? retry with a new (higher) IDp.

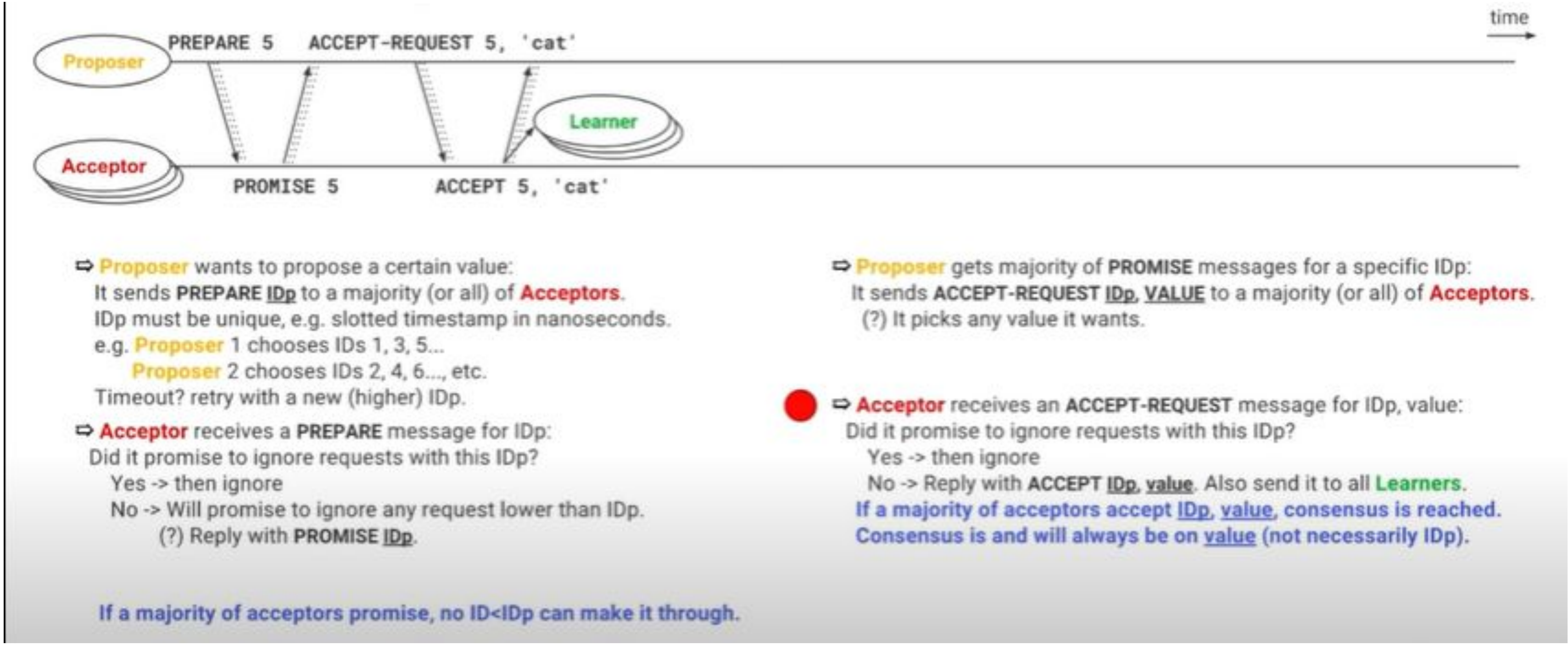
⇨ **Acceptor** receives a **PREPARE** message for IDp:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Will promise to ignore any request lower than IDp.
(?) Reply with **PROMISE** **IDp**.

If a majority of acceptors promise, no ID<IDp can make it through.

● ⇨ Proposer gets majority of **PROMISE** messages for a specific IDp:
It sends **ACCEPT-REQUEST** **IDp**, **VALUE** to a majority (or all) of **Acceptors**.
(?) It picks any value it wants.

Courtesy : Google Talks

PAXOS Algorithm



⇨ Proposer wants to propose a certain value:
It sends **PREPARE IDp** to a majority (or all) of Acceptors.
IDp must be unique, e.g. slotted timestamp in nanoseconds.
e.g. Proposer 1 chooses IDs 1, 3, 5...
Proposer 2 chooses IDs 2, 4, 6..., etc.
Timeout? retry with a new (higher) IDp.

⇨ Acceptor receives a **PREPARE** message for IDp:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Will promise to ignore any request lower than IDp.
(?) Reply with **PROMISE IDp**.

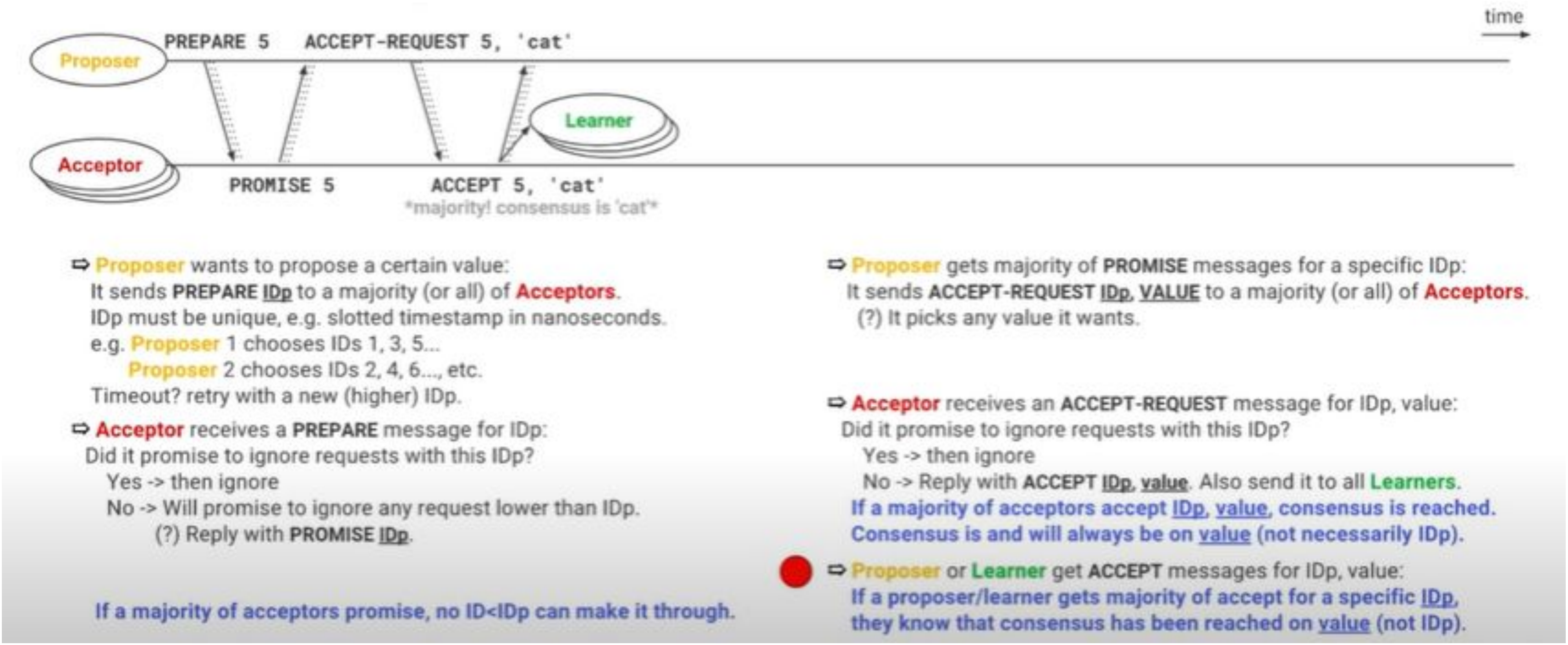
If a majority of acceptors promise, no ID<IDp can make it through.

⇨ Proposer gets majority of **PROMISE** messages for a specific IDp:
It sends **ACCEPT-REQUEST IDp, VALUE** to a majority (or all) of Acceptors.
(?) It picks any value it wants.

⇨ Acceptor receives an **ACCEPT-REQUEST** message for IDp, value:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Reply with **ACCEPT IDp, value**. Also send it to all Learners.
If a majority of acceptors accept IDp, value, consensus is reached.
Consensus is and will always be on value (not necessarily IDp).

Courtesy : Google Talks

PAXOS Algorithm



⇨ Proposer wants to propose a certain value:
It sends **PREPARE IDp** to a majority (or all) of **Acceptors**.
IDp must be unique, e.g. slotted timestamp in nanoseconds.
e.g. Proposer 1 chooses IDs 1, 3, 5...
Proposer 2 chooses IDs 2, 4, 6..., etc.
Timeout? retry with a new (higher) IDp.

⇨ **Acceptor** receives a **PREPARE** message for IDp:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Will promise to ignore any request lower than IDp.
(?) Reply with **PROMISE IDp**.

If a majority of acceptors promise, no ID<IDp can make it through.

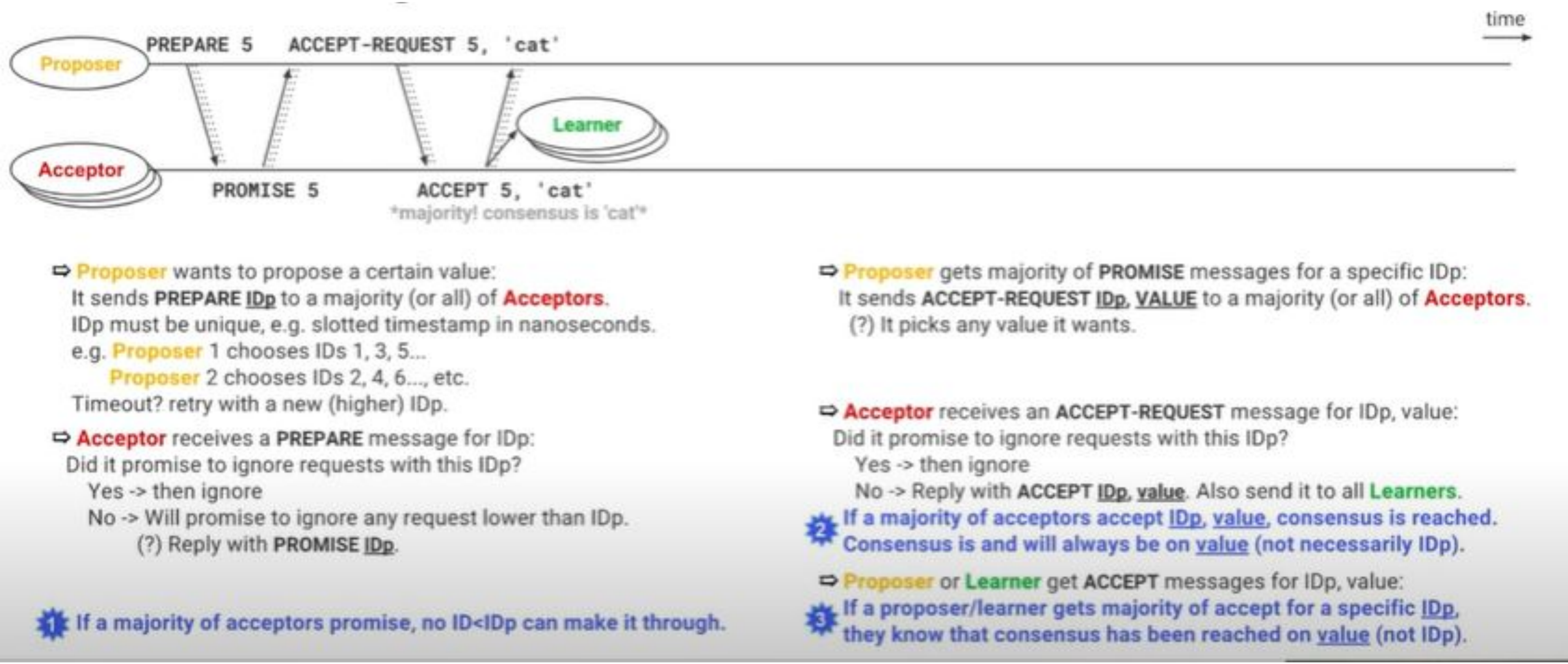
⇨ Proposer gets majority of **PROMISE** messages for a specific IDp:
It sends **ACCEPT-REQUEST IDp, VALUE** to a majority (or all) of **Acceptors**.
(?) It picks any value it wants.

⇨ **Acceptor** receives an **ACCEPT-REQUEST** message for IDp, value:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Reply with **ACCEPT IDp, value**. Also send it to all **Learners**.
If a majority of acceptors accept IDp, value, consensus is reached.
Consensus is and will always be on value (not necessarily IDp).

⇨ Proposer or **Learner** get **ACCEPT** messages for IDp, value:
If a proposer/learner gets majority of accept for a specific IDp,
they know that consensus has been reached on value (not IDp).

Courtesy : Google Talks

PAXOS Algorithm



⇨ Proposer wants to propose a certain value:
It sends **PREPARE IDp** to a majority (or all) of **Acceptors**.
IDp must be unique, e.g. slotted timestamp in nanoseconds.
e.g. Proposer 1 chooses IDs 1, 3, 5...
Proposer 2 chooses IDs 2, 4, 6..., etc.
Timeout? retry with a new (higher) IDp.

⇨ **Acceptor** receives a **PREPARE** message for IDp:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Will promise to ignore any request lower than IDp.
(?) Reply with **PROMISE IDp**.

1 If a majority of acceptors promise, no ID<IDp can make it through.

⇨ Proposer gets majority of **PROMISE** messages for a specific IDp:
It sends **ACCEPT-REQUEST IDp, VALUE** to a majority (or all) of **Acceptors**.
(?) It picks any value it wants.

⇨ **Acceptor** receives an **ACCEPT-REQUEST** message for IDp, value:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Reply with **ACCEPT IDp, value**. Also send it to all **Learners**.

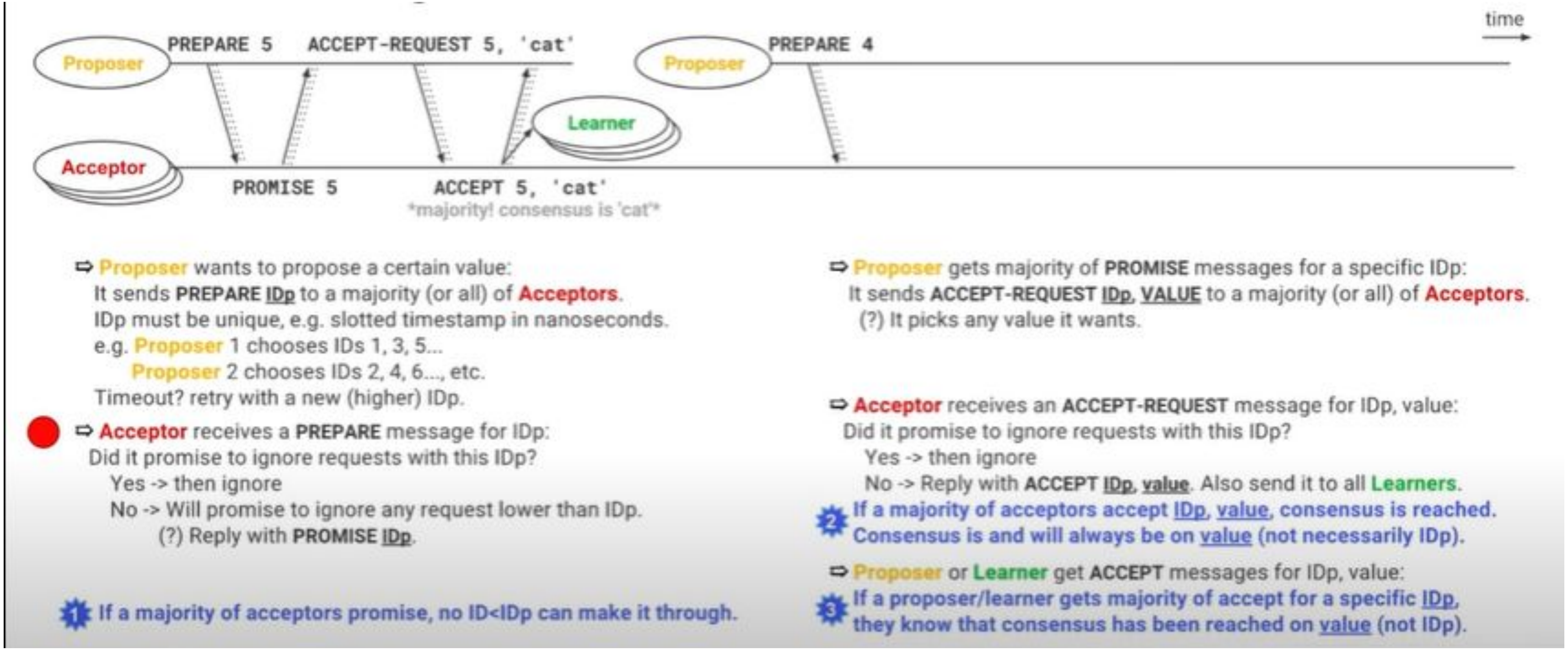
**2 If a majority of acceptors accept IDp, value, consensus is reached.
Consensus is and will always be on value (not necessarily IDp).**

⇨ Proposer or **Learner** get **ACCEPT** messages for IDp, value:

3 If a proposer/learner gets majority of accept for a specific IDp, they know that consensus has been reached on value (not IDp).

Courtesy : Google Talks

PAXOS Algorithm



⇨ Proposer wants to propose a certain value:
It sends **PREPARE** <u>**IDp**</u> to a majority (or all) of **Acceptors**.
IDp must be unique, e.g. slotted timestamp in nanoseconds.
e.g. Proposer 1 chooses IDs 1, 3, 5...
Proposer 2 chooses IDs 2, 4, 6..., etc.
Timeout? retry with a new (higher) IDp.

● ⇨ **Acceptor** receives a **PREPARE** message for IDp:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Will promise to ignore any request lower than IDp.
(?) Reply with **PROMISE** <u>**IDp**</u>.

1 **If a majority of acceptors promise, no ID<IDp can make it through.**

⇨ Proposer gets majority of **PROMISE** messages for a specific IDp:
It sends **ACCEPT-REQUEST** <u>**IDp**</u>, <u>**VALUE**</u> to a majority (or all) of **Acceptors**.
(?) It picks any value it wants.

⇨ **Acceptor** receives an **ACCEPT-REQUEST** message for IDp, value:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Reply with **ACCEPT** <u>**IDp**</u>, <u>**value**</u>. Also send it to all **Learners**.

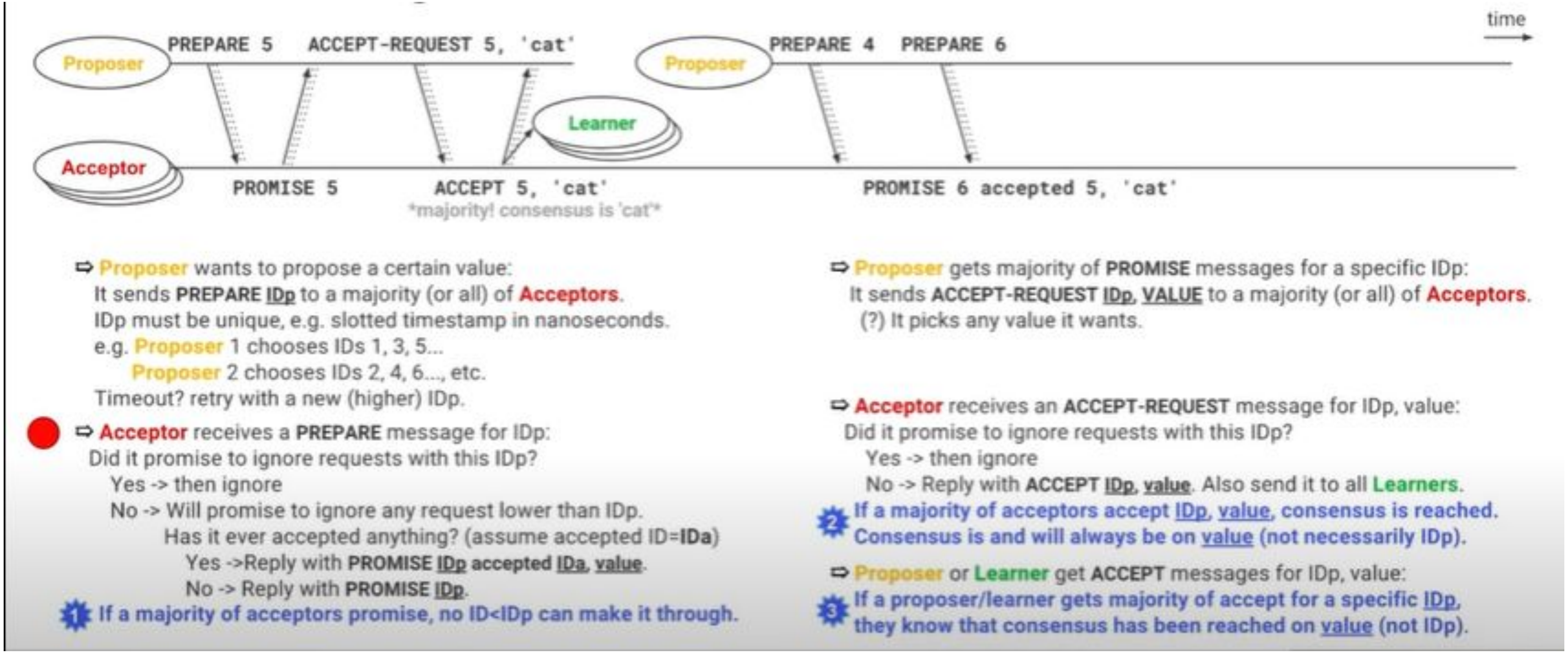
2 **If a majority of acceptors accept <u>IDp</u>, <u>value</u>, consensus is reached.
Consensus is and will always be on <u>value</u> (not necessarily IDp).**

⇨ Proposer or **Learner** get **ACCEPT** messages for IDp, value:

3 **If a proposer/learner gets majority of accept for a specific <u>IDp</u>, they know that consensus has been reached on <u>value</u> (not IDp).**

Courtesy : Google Talks

PAXOS Algorithm



⇨ Proposer wants to propose a certain value:
It sends **PREPARE** **IDp** to a majority (or all) of **Acceptors**.
IDp must be unique, e.g. slotted timestamp in nanoseconds.
e.g. Proposer 1 chooses IDs 1, 3, 5...
Proposer 2 chooses IDs 2, 4, 6..., etc.
Timeout? retry with a new (higher) IDp.

● ⇨ **Acceptor** receives a **PREPARE** message for IDp:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Will promise to ignore any request lower than IDp.
Has it ever accepted anything? (assume accepted ID=**IDa**)
Yes ->Reply with **PROMISE** **IDp** **accepted** **IDa**, **value**.
No -> Reply with **PROMISE** **IDp**.

1 If a majority of acceptors promise, no ID<IDp can make it through.

⇨ Proposer gets majority of **PROMISE** messages for a specific IDp:
It sends **ACCEPT-REQUEST** **IDp**, **VALUE** to a majority (or all) of **Acceptors**.
(?) It picks any value it wants.

⇨ **Acceptor** receives an **ACCEPT-REQUEST** message for IDp, value:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Reply with **ACCEPT** **IDp**, **value**. Also send it to all **Learners**.

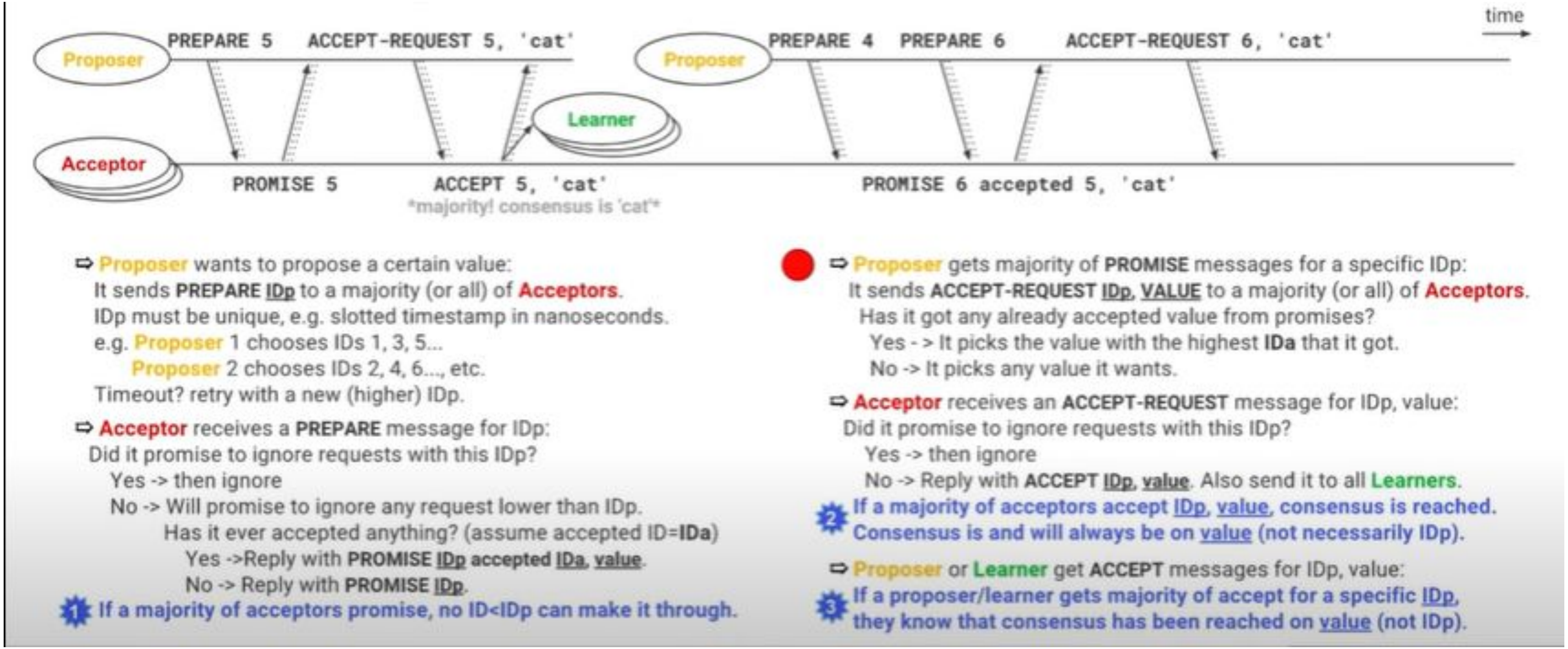
**2 If a majority of acceptors accept IDp, value, consensus is reached.
Consensus is and will always be on value (not necessarily IDp).**

⇨ Proposer or **Learner** get **ACCEPT** messages for IDp, value:

3 If a proposer/learner gets majority of accept for a specific IDp, they know that consensus has been reached on value (not IDp).

Courtesy : Google Talks

PAXOS Algorithm



⇨ Proposer wants to propose a certain value:
It sends **PREPARE** **IDp** to a majority (or all) of **Acceptors**.
IDp must be unique, e.g. slotted timestamp in nanoseconds.
e.g. Proposer 1 chooses IDs 1, 3, 5...
Proposer 2 chooses IDs 2, 4, 6..., etc.
Timeout? retry with a new (higher) IDp.

⇨ **Acceptor** receives a **PREPARE** message for IDp:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Will promise to ignore any request lower than IDp.
Has it ever accepted anything? (assume accepted ID=**IDa**)
Yes ->Reply with **PROMISE** **IDp** **accepted** **IDa**, **value**.
No -> Reply with **PROMISE** **IDp**.

1 **If a majority of acceptors promise, no ID<IDp can make it through.**

● ⇨ Proposer gets majority of **PROMISE** messages for a specific IDp:
It sends **ACCEPT-REQUEST** **IDp**, **VALUE** to a majority (or all) of **Acceptors**.
Has it got any already accepted value from promises?
Yes - > It picks the value with the highest **IDa** that it got.
No -> It picks any value it wants.

⇨ **Acceptor** receives an **ACCEPT-REQUEST** message for IDp, value:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Reply with **ACCEPT** **IDp**, **value**. Also send it to all Learners.

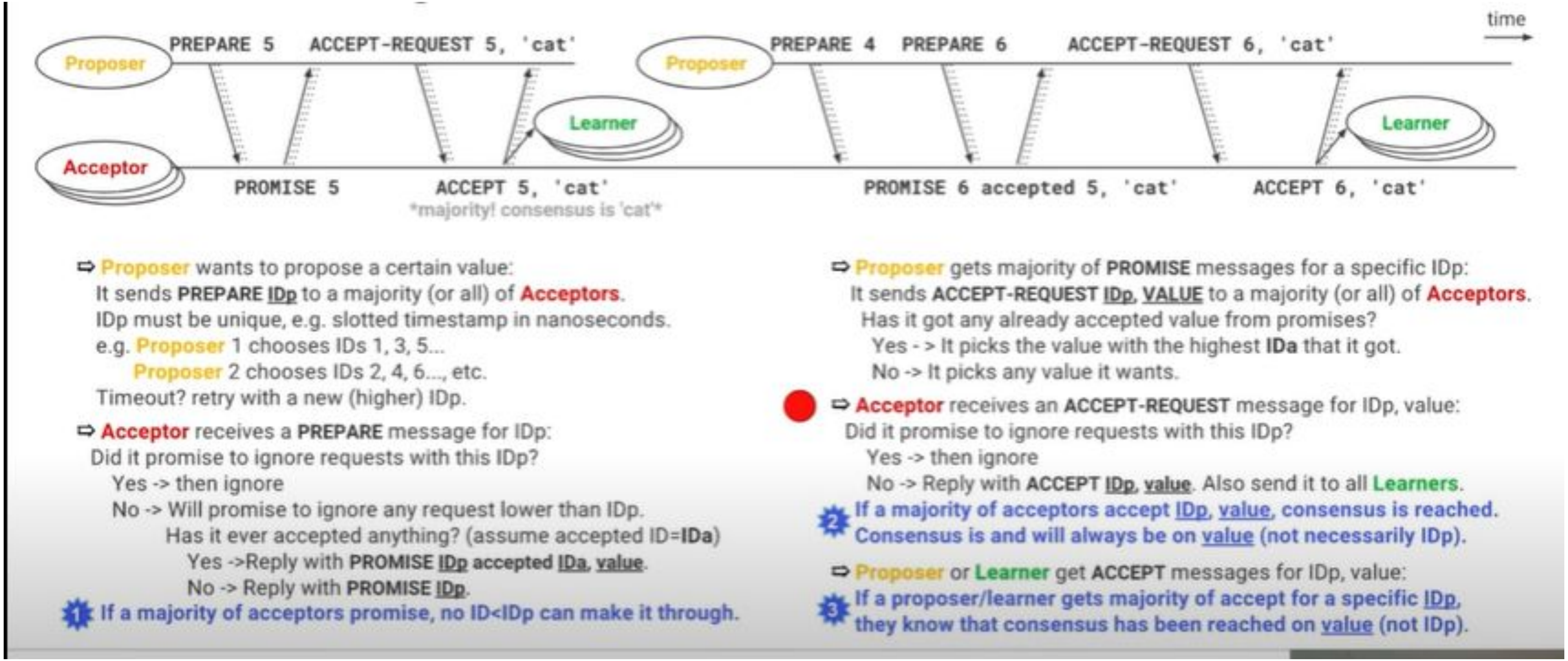
2 **If a majority of acceptors accept IDp, value, consensus is reached.**
Consensus is and will always be on value (not necessarily IDp).

⇨ Proposer or Learner get **ACCEPT** messages for IDp, value:

3 **If a proposer/learner gets majority of accept for a specific IDp, they know that consensus has been reached on value (not IDp).**

Courtesy : Google Talks

PAXOS Algorithm



⇨ Proposer wants to propose a certain value:
- It sends **PREPARE IDp** to a majority (or all) of **Acceptors**.
- IDp must be unique, e.g. slotted timestamp in nanoseconds.
- e.g. Proposer 1 chooses IDs 1, 3, 5...
  - Proposer 2 chooses IDs 2, 4, 6..., etc.
- Timeout? retry with a new (higher) IDp.

⇨ **Acceptor** receives a **PREPARE** message for IDp:
- Did it promise to ignore requests with this IDp?
  - Yes -> then ignore
  - No -> Will promise to ignore any request lower than IDp.
    - Has it ever accepted anything? (assume accepted ID=**IDa**)
      - Yes ->Reply with **PROMISE IDp accepted IDa, value**.
      - No -> Reply with **PROMISE IDp**.

1 If a majority of acceptors promise, no ID<IDp can make it through.

⇨ Proposer gets majority of **PROMISE** messages for a specific IDp:
- It sends **ACCEPT-REQUEST IDp, VALUE** to a majority (or all) of **Acceptors**.
- Has it got any already accepted value from promises?
  - Yes - > It picks the value with the highest **IDa** that it got.
  - No -> It picks any value it wants.

● ⇨ **Acceptor** receives an **ACCEPT-REQUEST** message for IDp, value:
- Did it promise to ignore requests with this IDp?
  - Yes -> then ignore
  - No -> Reply with **ACCEPT IDp, value**. Also send it to all **Learners**.

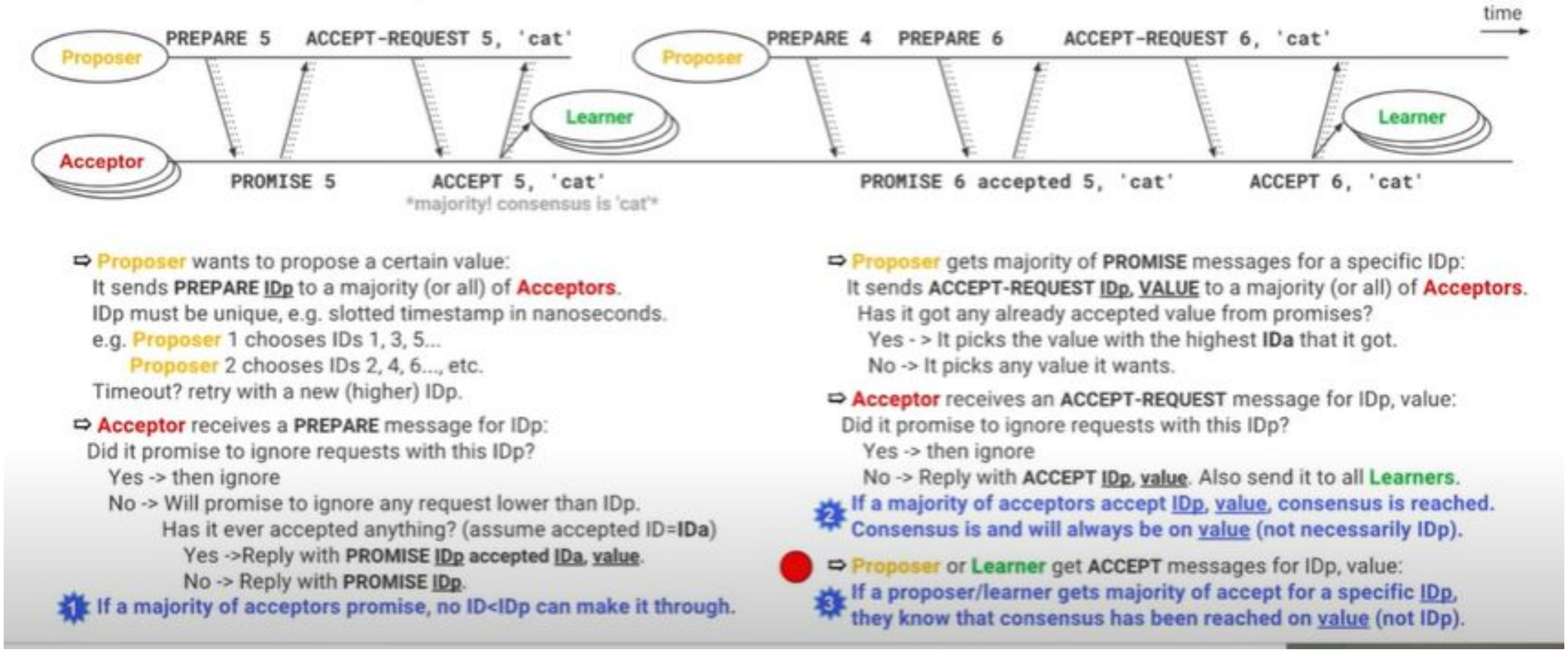
2 If a majority of acceptors accept IDp, value, consensus is reached. Consensus is and will always be on value (not necessarily IDp).

⇨ Proposer or **Learner** get **ACCEPT** messages for IDp, value:

3 If a proposer/learner gets majority of accept for a specific IDp, they know that consensus has been reached on value (not IDp).

Courtesy : Google Talks

PAXOS Algorithm



⇨ Proposer wants to propose a certain value:
It sends **PREPARE** <u>**IDp**</u> to a majority (or all) of **Acceptors**.
IDp must be unique, e.g. slotted timestamp in nanoseconds.
e.g. Proposer 1 chooses IDs 1, 3, 5...
Proposer 2 chooses IDs 2, 4, 6..., etc.
Timeout? retry with a new (higher) IDp.

⇨ **Acceptor** receives a **PREPARE** message for IDp:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Will promise to ignore any request lower than IDp.
Has it ever accepted anything? (assume accepted ID=**IDa**)
Yes ->Reply with **PROMISE** <u>**IDp**</u> **accepted** <u>**IDa**</u>, <u>**value**</u>.
No -> Reply with **PROMISE** <u>**IDp**</u>.

1 **If a majority of acceptors promise, no ID<IDp can make it through.**

⇨ Proposer gets majority of **PROMISE** messages for a specific IDp:
It sends **ACCEPT-REQUEST** <u>**IDp**</u>, <u>**VALUE**</u> to a majority (or all) of **Acceptors**.
Has it got any already accepted value from promises?
Yes - > It picks the value with the highest **IDa** that it got.
No -> It picks any value it wants.

⇨ **Acceptor** receives an **ACCEPT-REQUEST** message for IDp, value:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Reply with **ACCEPT** <u>**IDp**</u>, <u>**value**</u>. Also send it to all **Learners**.

2 **If a majority of acceptors accept <u>IDp</u>, <u>value</u>, consensus is reached. Consensus is and will always be on <u>value</u> (not necessarily IDp).**

⇨ Proposer or **Learner** get **ACCEPT** messages for IDp, value:

3 **If a proposer/learner gets majority of accept for a specific <u>IDp</u>, they know that consensus has been reached on <u>value</u> (not IDp).**

Courtesy : Google Talks

Consensus Algorithms - PAXOS Algorithm

Phase 1: Prepare (Prepare Phase):

- A proposer (node) initiates the consensus process by sending a "prepare" message with a proposal number (n) to a majority of acceptors.
- Upon receiving a "prepare" message, each acceptor checks if the proposal number (n) is greater than any proposal number it has seen before.
  - If so, it responds with a "promise" message indicating its acceptance of the proposal number (n) and the highest-numbered proposal (if any) it has accepted.
- If an acceptor has already accepted a proposal with a higher number,
  - it responds with the highest-numbered proposal it has accepted.
- Otherwise, it responds with a promise to not accept any proposal with a lower number.

Courtesy : Google Talks

Consensus Algorithms - PAXOS Algorithm

Phase 2: Accept (Accept Phase):

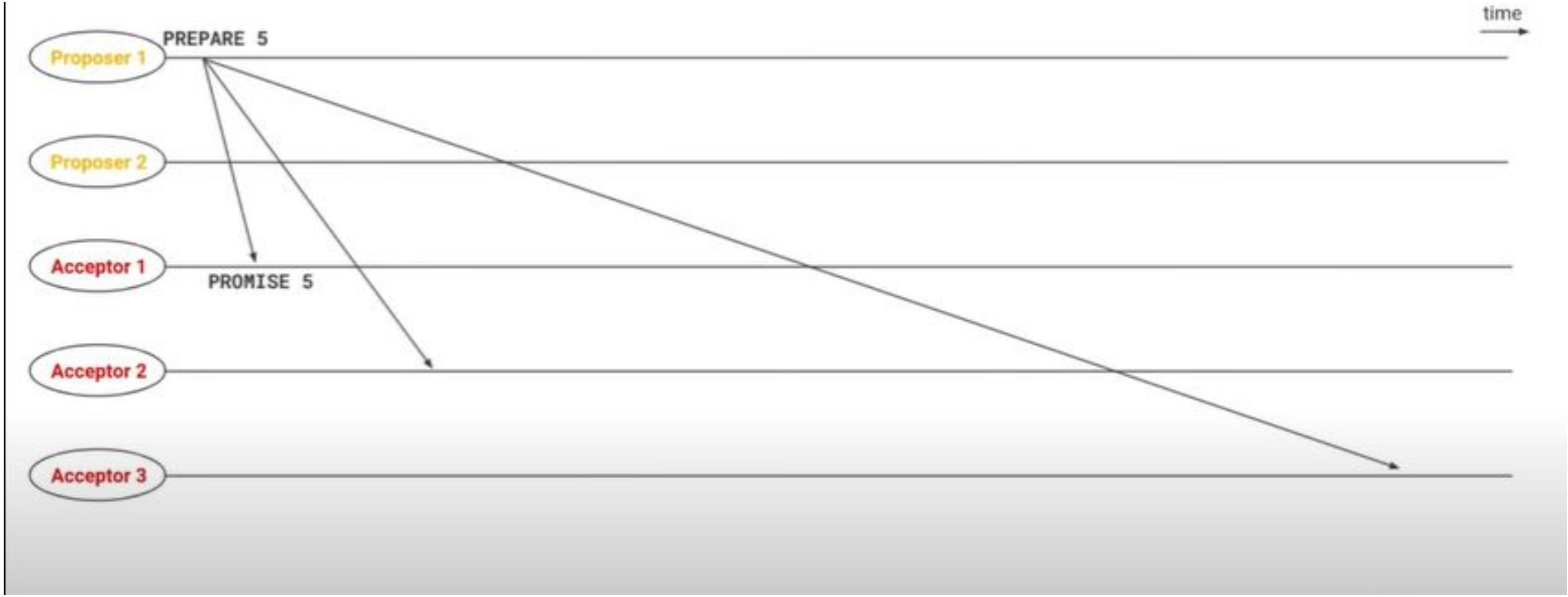
- If the proposer receives "promise" messages from a majority of acceptors, it can **proceed to the accept phase.**
- The proposer sends an "accept" message with the proposed value and the proposal number (n) it chose in the prepare phase to a majority of acceptors.
- Upon receiving an "accept" message, each acceptor checks if it has already **promised not to accept any proposal with a lower number**.
  - If not, it accepts the proposal by updating its accepted value and responds with an "accepted" message to the proposer.

Phase 3: Learn (Learn Phase):

- Once the proposer receives "accepted" messages from a majority of acceptors, **it knows that consensus has been reached.**
- The proposer sends a "learn" message to all nodes, informing them of the agreed-upon value.
- Upon receiving a "learn" message, each node updates its state with the agreed-upon value.

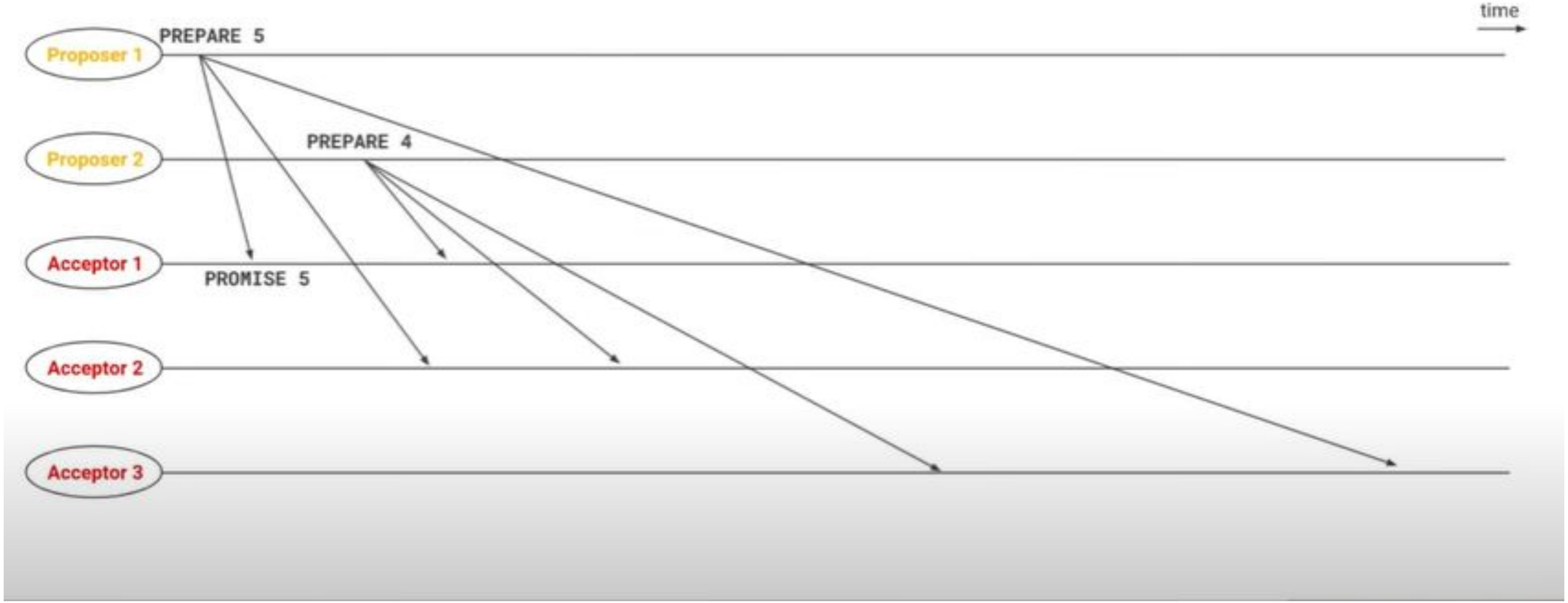
Courtesy : Google Talks

PAXOS Algorithm : Detailed Review with Majority of Promises



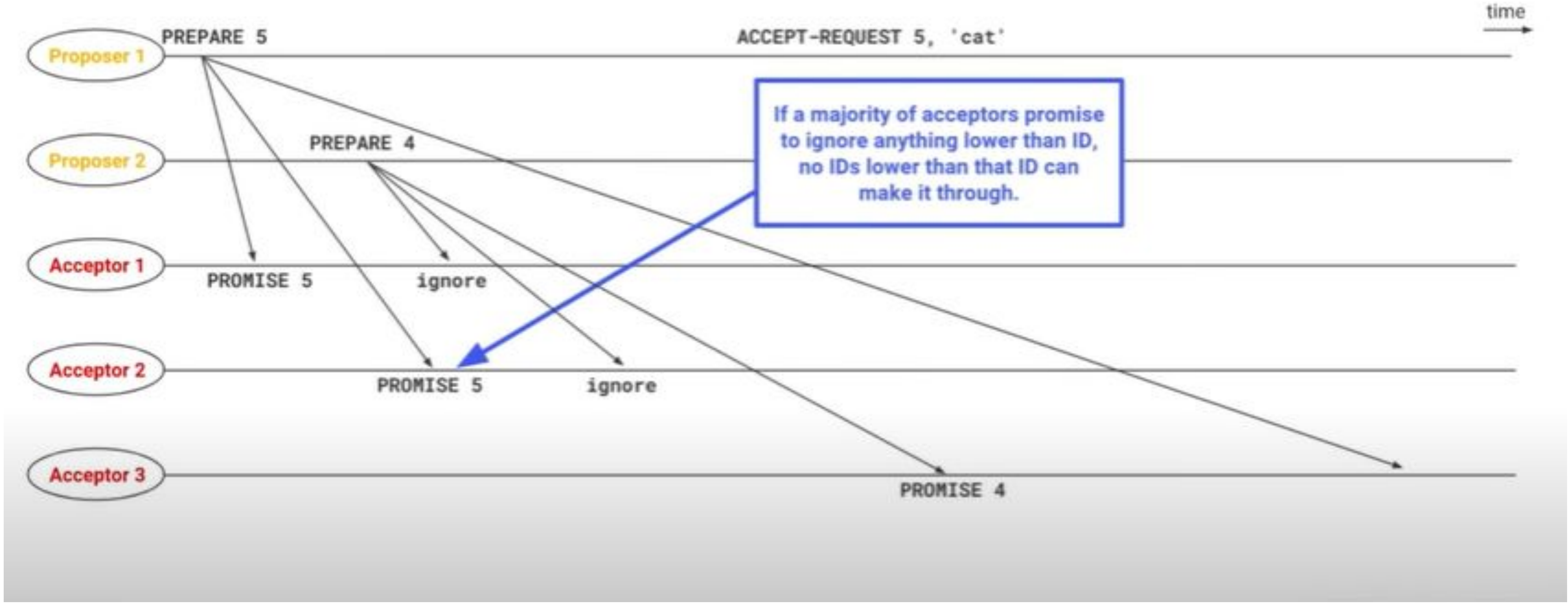
Courtesy : Google Talks

PAXOS Algorithm : Detailed Review with Majority of Promises



Courtesy : Google Talks

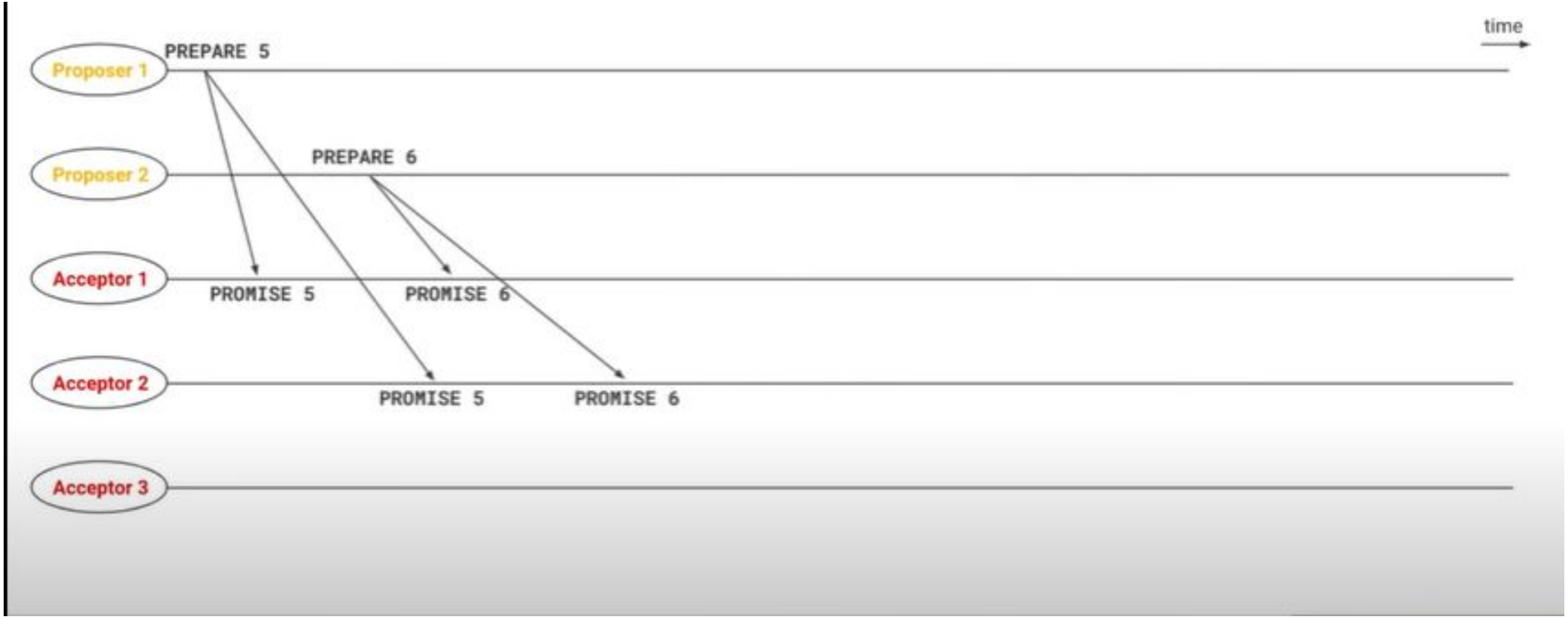
PAXOS Algorithm : Detailed Review with Majority of Promises



If a majority of acceptors promise to ignore anything lower than ID, no IDs lower than that ID can make it through.

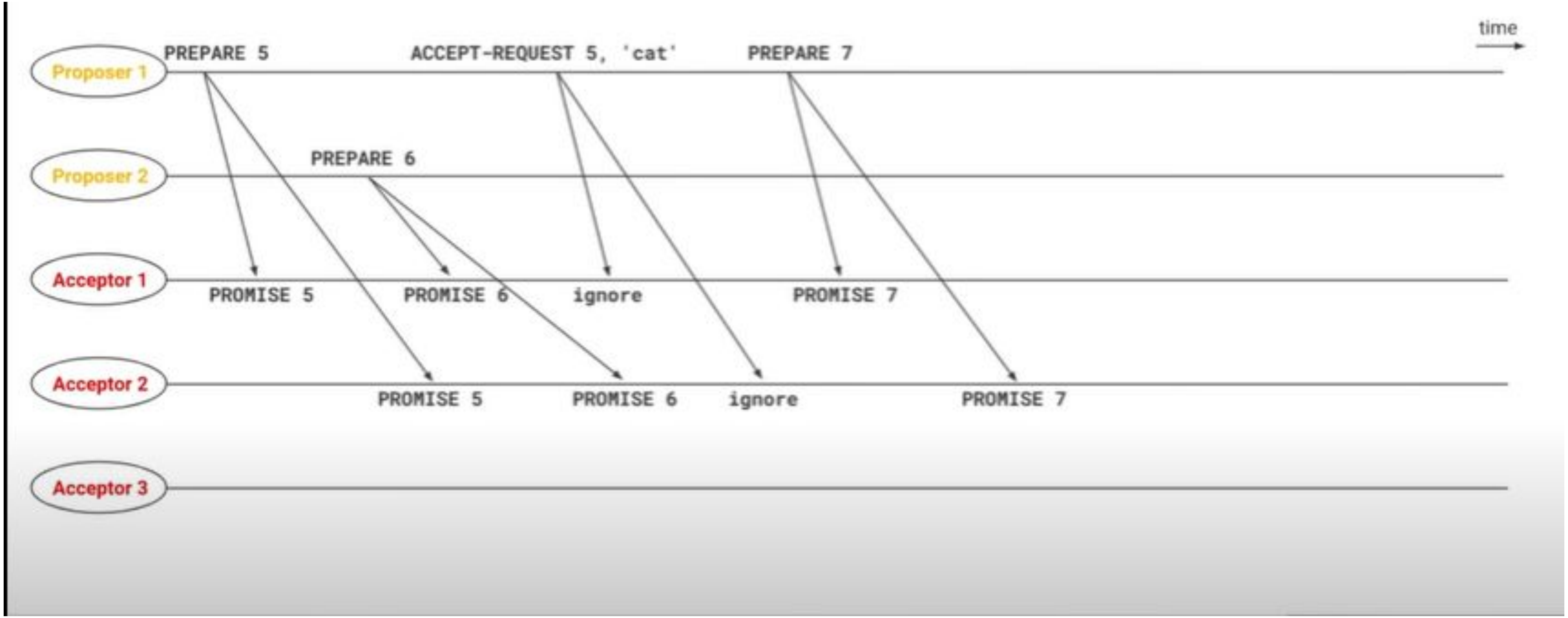
Courtesy : Google Talks

PAXOS Algorithm : Detailed Review with Contention



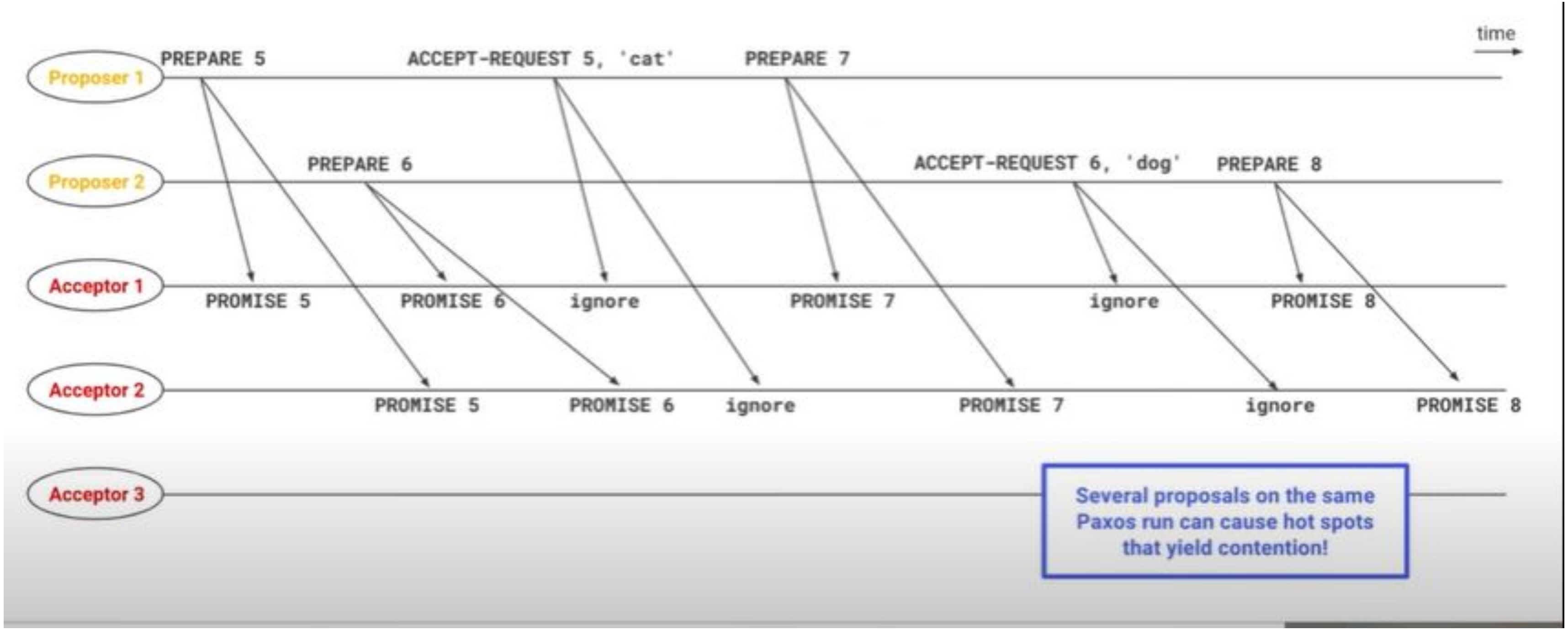
Courtesy : Google Talks

PAXOS Algorithm : Detailed Review with Contention



Courtesy : Google Talks

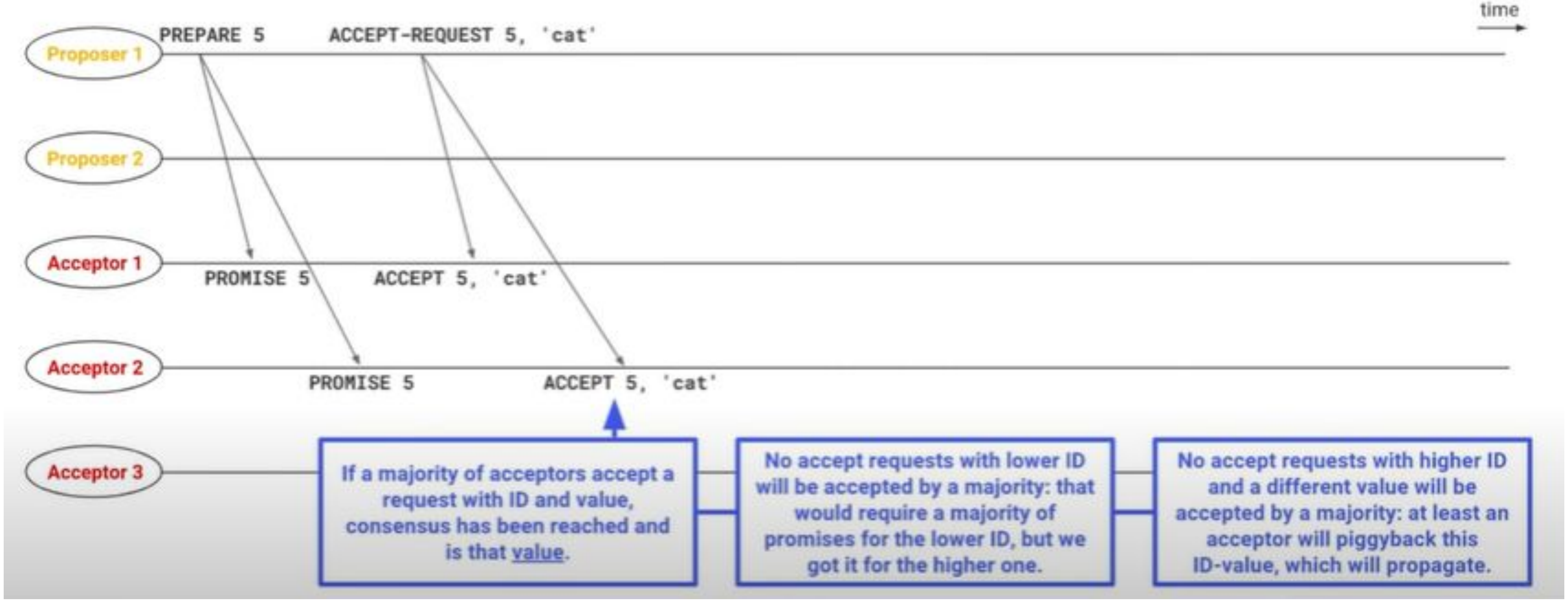
PAXOS Algorithm : Detailed Review with Contention



Several proposals on the same Paxos run can cause hot spots that yield contention!

Courtesy : Google Talks

PAXOS Algorithm : Detailed Review with Majority of Accepts



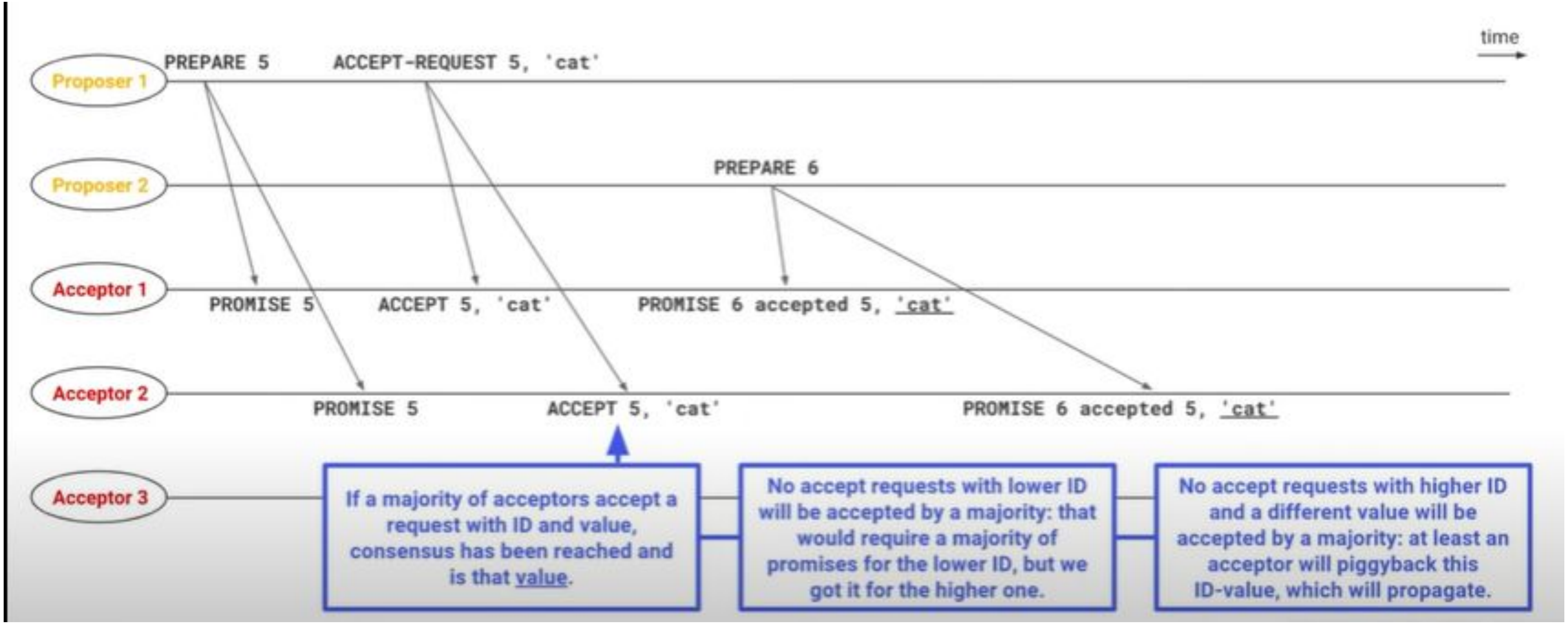
If a majority of acceptors accept a request with ID and value, consensus has been reached and is that value.

No accept requests with lower ID will be accepted by a majority: that would require a majority of promises for the lower ID, but we got it for the higher one.

No accept requests with higher ID and a different value will be accepted by a majority: at least an acceptor will piggyback this ID-value, which will propagate.

Courtesy : Google Talks

PAXOS Algorithm : Detailed Review with Majority of Accepts



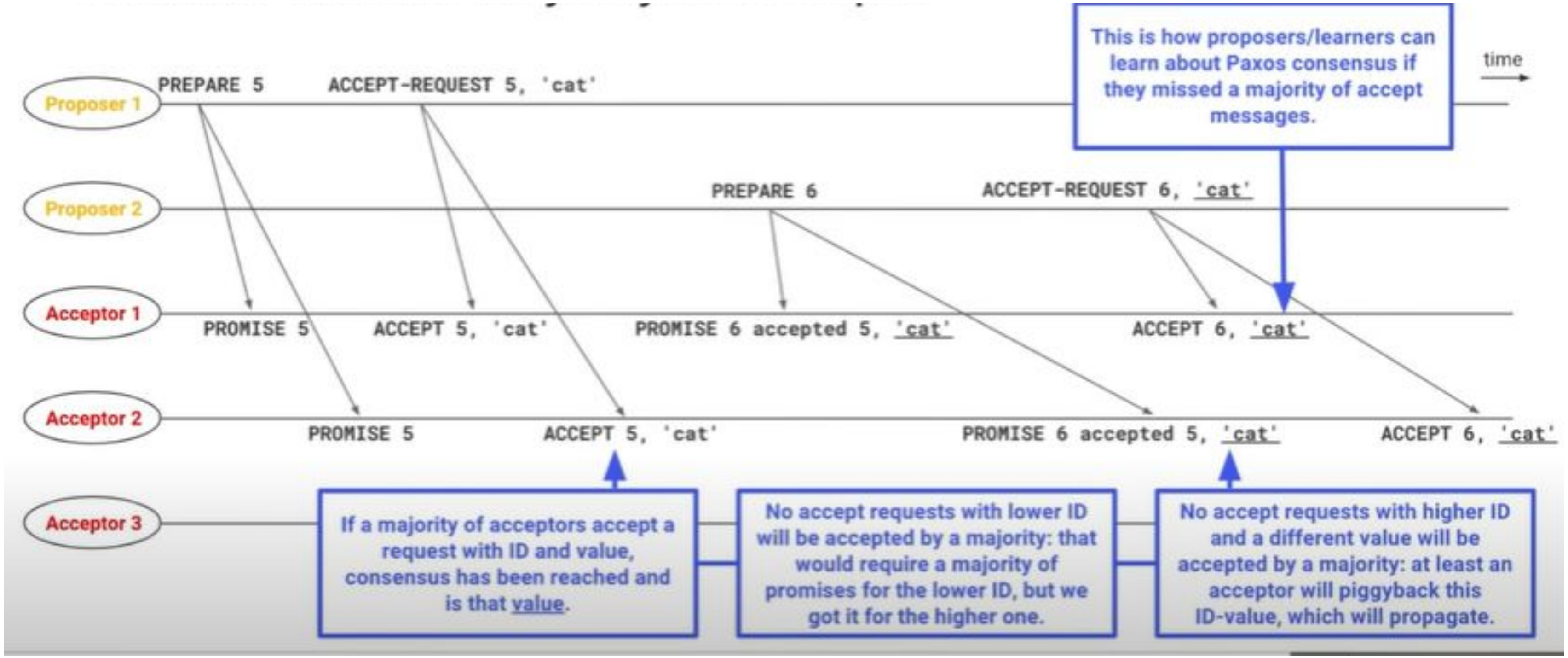
If a majority of acceptors accept a request with ID and value, consensus has been reached and is that value.

No accept requests with lower ID will be accepted by a majority: that would require a majority of promises for the lower ID, but we got it for the higher one.

No accept requests with higher ID and a different value will be accepted by a majority: at least an acceptor will piggyback this ID-value, which will propagate.

Courtesy : Google Talks

PAXOS Algorithm : Detailed Review with Majority of Accepts



This is how proposers/learners can learn about Paxos consensus if they missed a majority of accept messages.

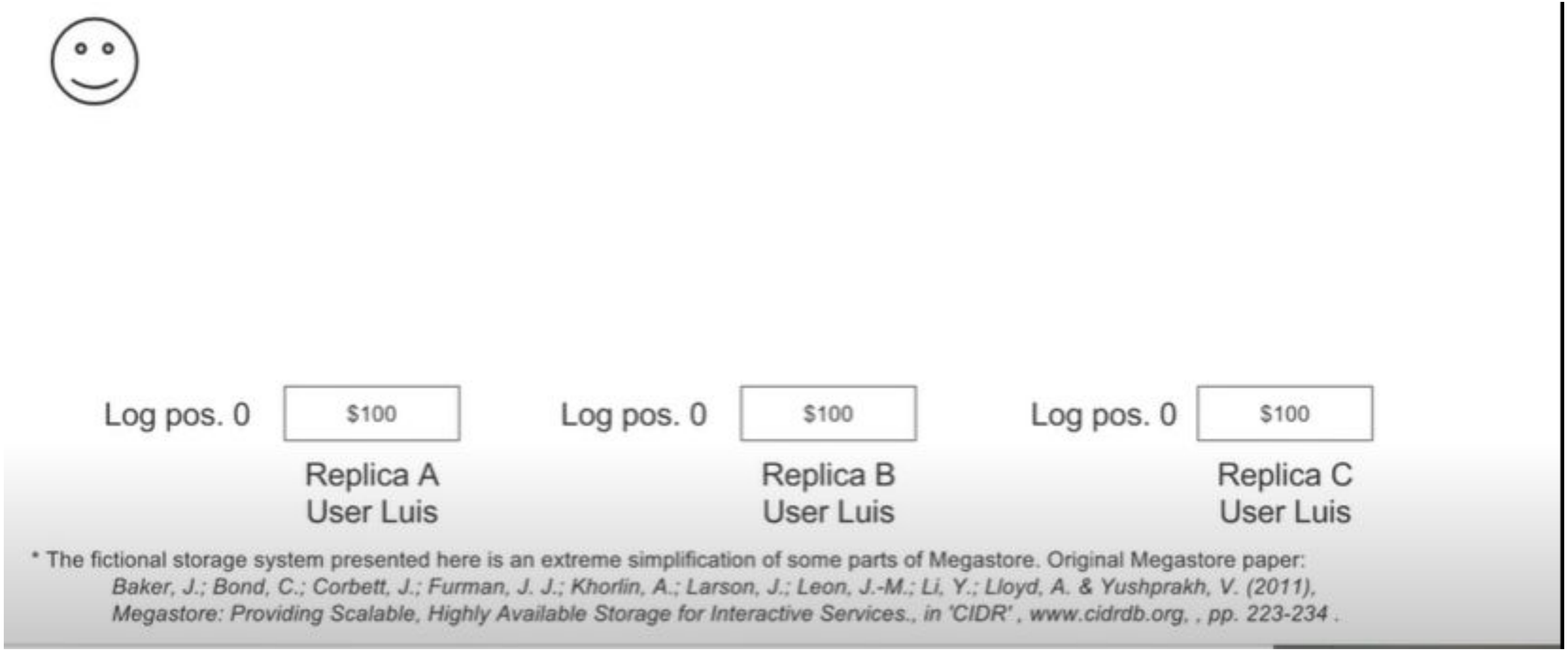
If a majority of acceptors accept a request with ID and value, consensus has been reached and is that value.

No accept requests with lower ID will be accepted by a majority: that would require a majority of promises for the lower ID, but we got it for the higher one.

No accept requests with higher ID and a different value will be accepted by a majority: at least an acceptor will piggyback this ID-value, which will propagate.

Courtesy : Google Talks

Practical Case : Distributed Storage System based on PAXOS



Log pos. 0 $100

Replica A
User Luis

Log pos. 0 $100

Replica B
User Luis

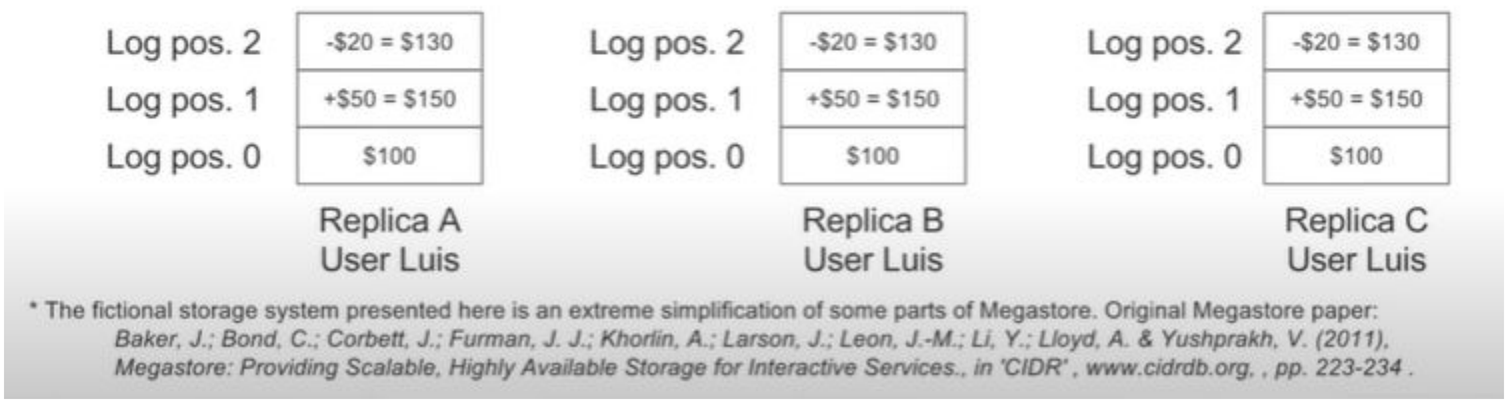
Log pos. 0 $100

Replica C
User Luis

* The fictional storage system presented here is an extreme simplification of some parts of Megastore. Original Megastore paper: *Baker, J.; Bond, C.; Corbett, J.; Furman, J. J.; Khorlin, A.; Larson, J.; Leon, J.-M.; Li, Y.; Lloyd, A. & Yushprakh, V. (2011), Megastore: Providing Scalable, Highly Available Storage for Interactive Services., in 'CIDR' , www.cidrdb.org, , pp. 223-234 .*

Courtesy : Google Talks

Practical Case : Distributed Storage System based on PAXOS

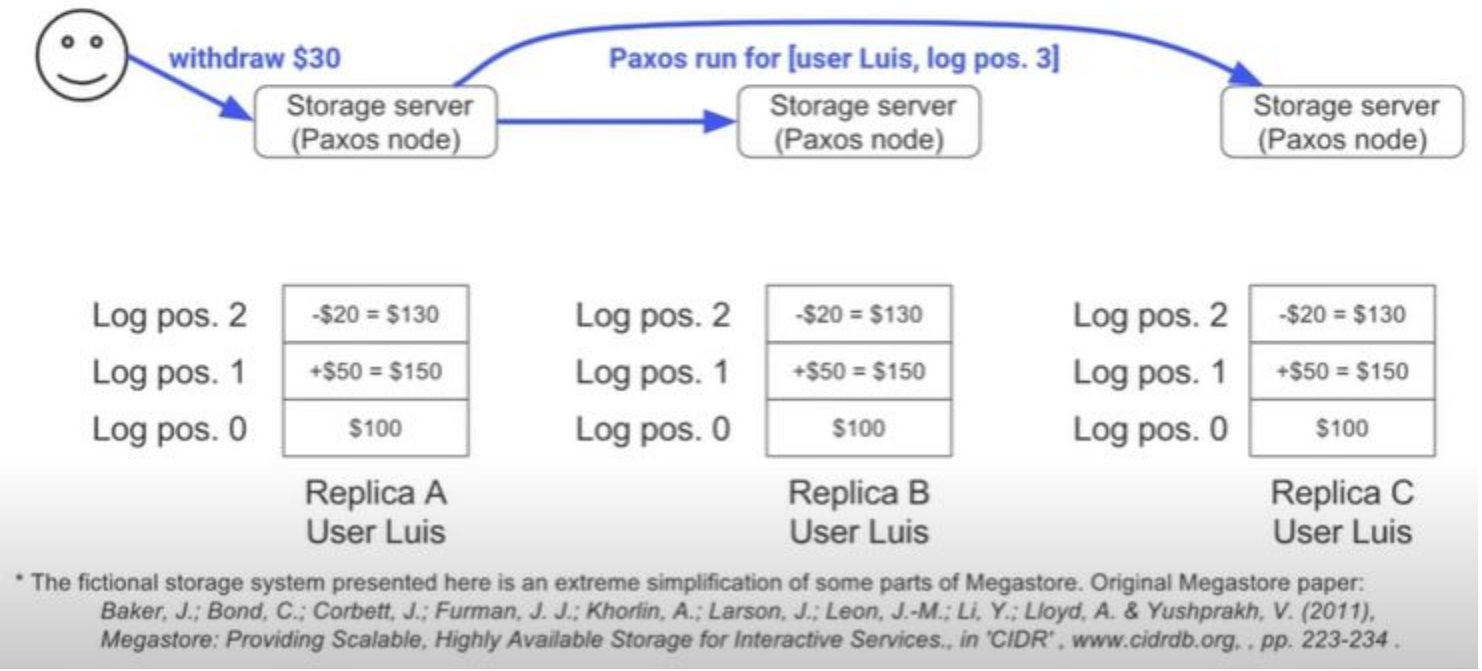


	Replica A User Luis	Replica B User Luis	Replica C User Luis
Log pos. 2	-$20 = $130	-$20 = $130	-$20 = $130
Log pos. 1	+$50 = $150	+$50 = $150	+$50 = $150
Log pos. 0	$100	$100	$100

* The fictional storage system presented here is an extreme simplification of some parts of Megastore. Original Megastore paper: *Baker, J.; Bond, C.; Corbett, J.; Furman, J. J.; Khorlin, A.; Larson, J.; Leon, J.-M.; Li, Y.; Lloyd, A. & Yushprakh, V. (2011), Megastore: Providing Scalable, Highly Available Storage for Interactive Services., in 'CIDR' , www.cidrdb.org, , pp. 223-234 .*

Courtesy : Google Talks

Practical Case : Distributed Storage System based on PAXOS

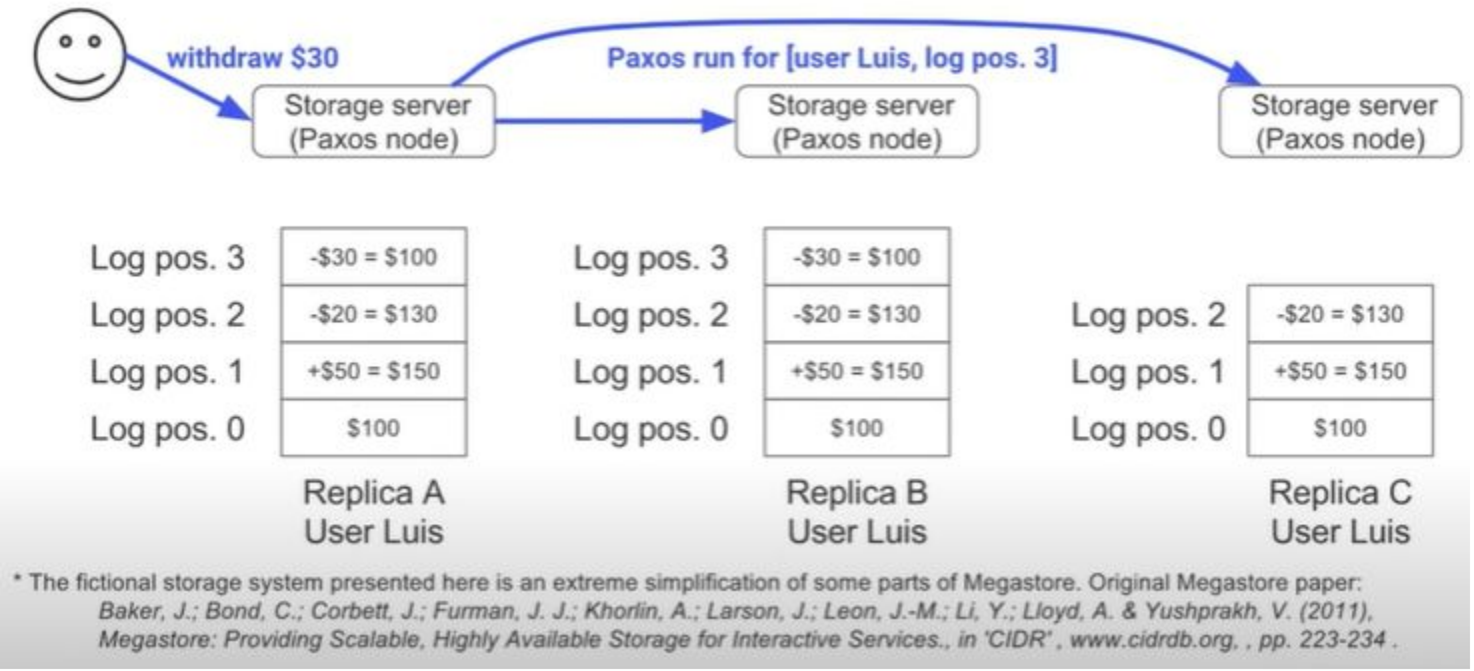


	Replica A User Luis	Replica B User Luis	Replica C User Luis
Log pos. 2	-$20 = $130	-$20 = $130	-$20 = $130
Log pos. 1	+$50 = $150	+$50 = $150	+$50 = $150
Log pos. 0	$100	$100	$100

* The fictional storage system presented here is an extreme simplification of some parts of Megastore. Original Megastore paper:
Baker, J.; Bond, C.; Corbett, J.; Furman, J. J.; Khorlin, A.; Larson, J.; Leon, J.-M.; Li, Y.; Lloyd, A. & Yushprakh, V. (2011), Megastore: Providing Scalable, Highly Available Storage for Interactive Services., in 'CIDR' , www.cidrdb.org, , pp. 223-234 .

Courtesy : Google Talks

Practical Case : Distributed Storage System based on PAXOS



Log pos. 3	-$30 = $100
Log pos. 2	-$20 = $130
Log pos. 1	+$50 = $150
Log pos. 0	$100

Replica A
User Luis

Log pos. 3	-$30 = $100
Log pos. 2	-$20 = $130
Log pos. 1	+$50 = $150
Log pos. 0	$100

Replica B
User Luis

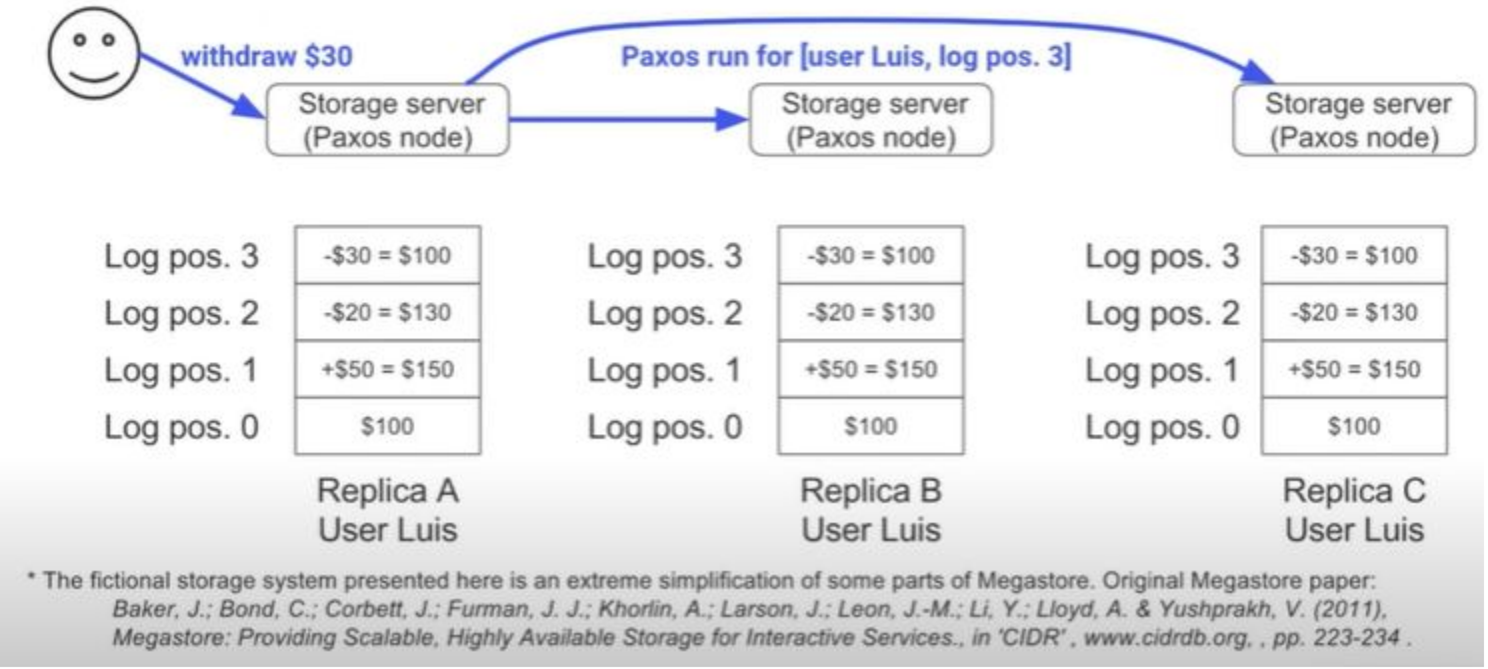
Log pos. 2	-$20 = $130
Log pos. 1	+$50 = $150
Log pos. 0	$100

Replica C
User Luis

* The fictional storage system presented here is an extreme simplification of some parts of Megastore. Original Megastore paper: *Baker, J.; Bond, C.; Corbett, J.; Furman, J. J.; Khorlin, A.; Larson, J.; Leon, J.-M.; Li, Y.; Lloyd, A. & Yushprakh, V. (2011), Megastore: Providing Scalable, Highly Available Storage for Interactive Services., in 'CIDR' , www.cidrdb.org, , pp. 223-234 .*

Courtesy : Google Talks

Practical Case : Distributed Storage System based on PAXOS


withdraw $30
Paxos run for [user Luis, log pos. 3]
Storage server (Paxos node)
Storage server (Paxos node)
Storage server (Paxos node)

Log pos. 3	-$30 = $100
Log pos. 2	-$20 = $130
Log pos. 1	+$50 = $150
Log pos. 0	$100

Replica A
User Luis

Log pos. 3	-$30 = $100
Log pos. 2	-$20 = $130
Log pos. 1	+$50 = $150
Log pos. 0	$100

Replica B
User Luis

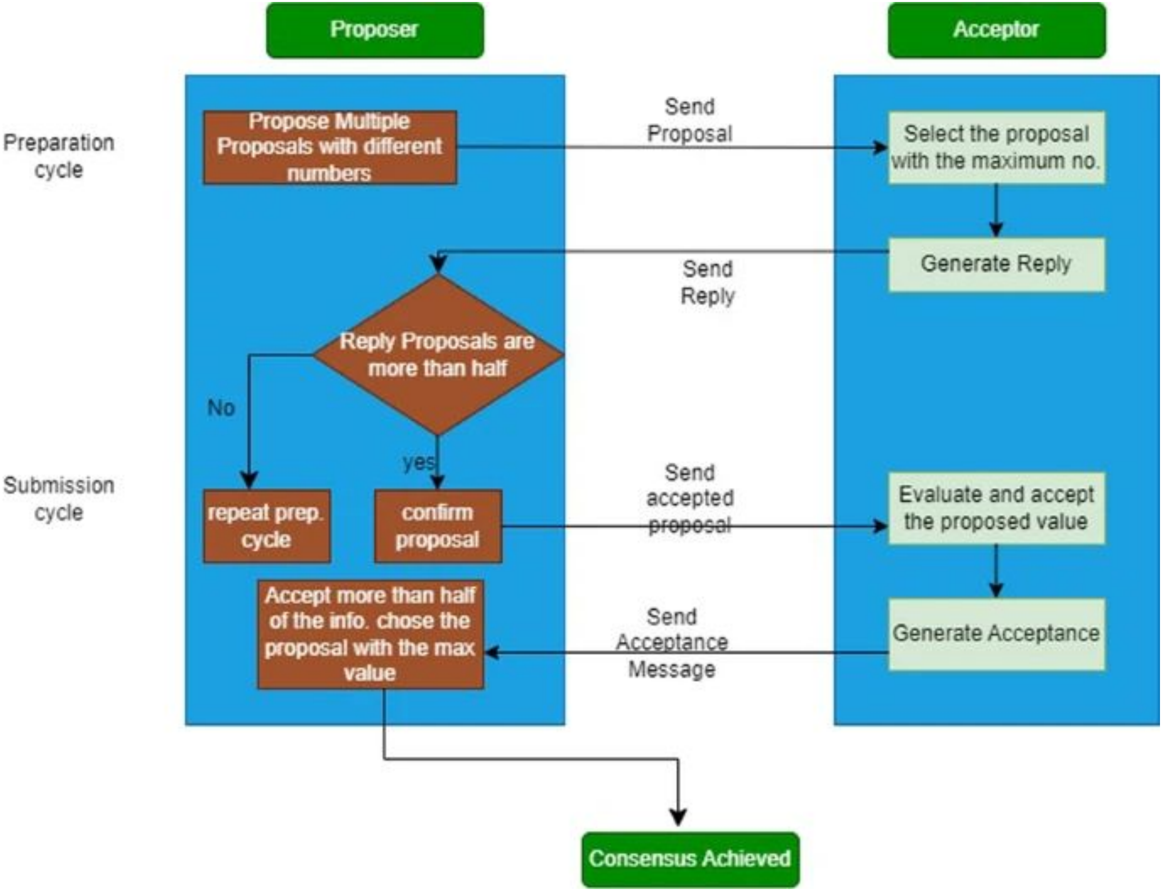
Log pos. 3	-$30 = $100
Log pos. 2	-$20 = $130
Log pos. 1	+$50 = $150
Log pos. 0	$100

Replica C
User Luis

* The fictional storage system presented here is an extreme simplification of some parts of Megastore. Original Megastore paper: *Baker, J.; Bond, C.; Corbett, J.; Furman, J. J.; Khorlin, A.; Larson, J.; Leon, J.-M.; Li, Y.; Lloyd, A. & Yushprakh, V. (2011), Megastore: Providing Scalable, Highly Available Storage for Interactive Services., in 'CIDR' , www.cidrdb.org, , pp. 223-234 .*

Courtesy : Google Talks

PAXOS Algorithm in Nutshell



Proposer

Acceptor

Preparation cycle

Propose Multiple Proposals with different numbers

Send Proposal

Select the proposal with the maximum no.

Generate Reply

Send Reply

Reply Proposals are more than half

No

yes

Submission cycle

repeat prep. cycle

confirm proposal

Send accepted proposal

Evaluate and accept the proposed value

Generate Acceptance

Send Acceptance Message

Accept more than half of the info. chose the proposal with the max value

Consensus Achieved

Courtesy : Springer

Consensus Algorithms - RAFT (Replicated and Fault Tolerant)

- distributed consensus protocol designed **to achieve fault tolerance and consensus** among a cluster of nodes in a distributed system.
- Introduced by Diego Ongaro and John Ousterhout in 2014 as an alternative to more complex consensus algorithms like Paxos.
- known for its simplicity, making it easier to understand, implement, and maintain compared to other consensus algorithms.
- Operates by electing a leader among the nodes in the cluster, which coordinates the replication of log entries across all nodes
- Key aspects of the Raft consensus algorithm include:
  - Leader-Based Approach: A single leader node coordinates the replication of log entries. This simplifies the protocol and reduces the likelihood of conflicts and inconsistencies.
  - Leader Lease: A leader node is elected for a certain period (typically referred to as the election timeout). This ensures that leader elections are triggered periodically, even in the absence of failures.
  - Membership Changes: Raft supports dynamic cluster membership changes, allowing nodes to join or leave the cluster without interrupting ongoing operations.

Consensus Algorithms - RAFT (Replicated and Fault Tolerant)

1. **Leader Election:**
   - Initially, all nodes in the cluster are in the follower state.
   - Nodes periodically send "heartbeat" messages to other nodes to indicate that they are still alive.
   - If a follower node does not receive a heartbeat within a certain timeout period, it transitions to the candidate state and starts a new leader election term.
   - The candidate node increments its current term and requests votes from other nodes in the cluster by sending "RequestVote" messages.
   - If a majority of nodes grant their vote to the candidate, it becomes the new leader for the current term.
2. **Log Replication:**
   - Once a leader is elected, it coordinates the replication of log entries across all nodes in the cluster.
   - Client requests (such as appending a new entry to the log) are sent to the leader.
   - The leader adds the new entry to its log and sends "AppendEntries" messages to followers to replicate the entry.
   - Followers append the new entry to their logs and respond to the leader with success or failure.
   - If a follower's log is inconsistent with the leader's log, the leader brings the follower's log up-to-date by sending missing entries.

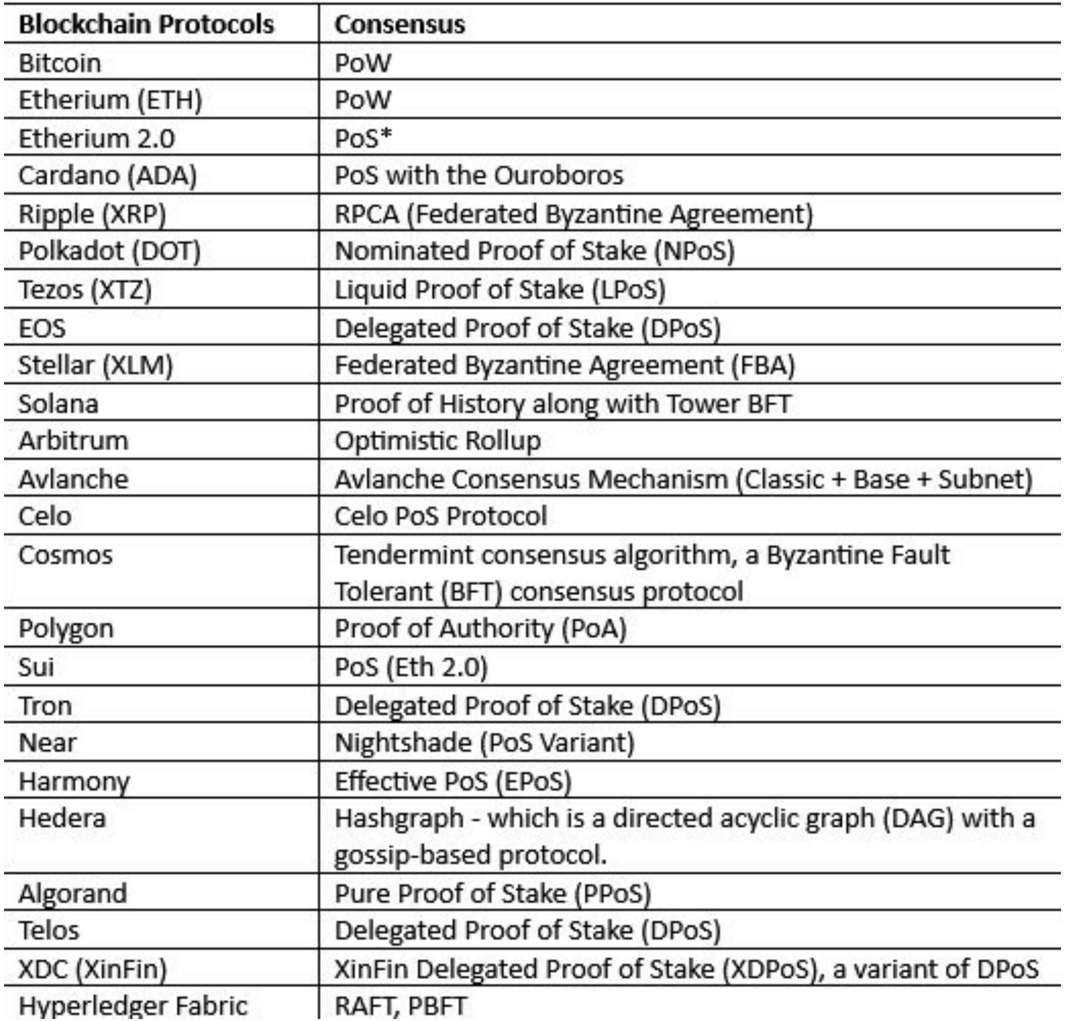
Courtesy : LinkedIn

Consensus Algorithms - RAFT (Replicated and Fault Tolerant)

3. **Commitment:**
   - Once a leader receives acknowledgment from a majority of followers that they have appended an entry to their logs, the entry is considered committed.
   - Committed entries can be applied to the state machine (e.g., executing commands, updating data) and responded to clients.
4. **Safety and Liveness:**
   - Raft ensures safety by guaranteeing that only one leader can be elected for a given term, and all committed log entries are consistent across all nodes.
   - Liveness is guaranteed by the leader's regular heartbeat messages, which allow nodes to detect leader failures and initiate new leader elections.

Courtesy : LinkedIn

Consensus Algorithms



Blockchain Protocols	Consensus
Bitcoin	PoW
Etherium (ETH)	PoW
Etherium 2.0	PoS*
Cardano (ADA)	PoS with the Ouroboros
Ripple (XRP)	RPCA (Federated Byzantine Agreement)
Polkadot (DOT)	Nominated Proof of Stake (NPoS)
Tezos (XTZ)	Liquid Proof of Stake (LPoS)
EOS	Delegated Proof of Stake (DPoS)
Stellar (XLM)	Federated Byzantine Agreement (FBA)
Solana	Proof of History along with Tower BFT
Arbitrum	Optimistic Rollup
Avlanche	Avlanche Consensus Mechanism (Classic + Base + Subnet)
Celo	Celo PoS Protocol
Cosmos	Tendermint consensus algorithm, a Byzantine Fault Tolerant (BFT) consensus protocol
Polygon	Proof of Authority (PoA)
Sui	PoS (Eth 2.0)
Tron	Delegated Proof of Stake (DPoS)
Near	Nightshade (PoS Variant)
Harmony	Effective PoS (EPoS)
Hedera	Hashgraph - which is a directed acyclic graph (DAG) with a gossip-based protocol.
Algorand	Pure Proof of Stake (PPoS)
Telos	Delegated Proof of Stake (DPoS)
XDC (XinFin)	XinFin Delegated Proof of Stake (XDPoS), a variant of DPoS
Hyperledger Fabric	RAFT, PBFT

Courtesy : LinkedIn

Smart Contracts in Private Blockchain

- Private blockchains are permissioned networks where participation is restricted to authorized entities. Smart contracts are **designed and deployed within this controlled environment.**
- In private blockchains, access to **smart contracts can be restricted to authorized participants.**
- Smart Contracts play a pivotal role in both public and private blockchain ecosystems, facilitating automation, security, and trust within the network.
- self-executing contracts with the terms of the agreement directly written into code.
- They **automatically enforce and execute the terms of the contract when predefined conditions are met, without the need for intermediaries.**
- Can be written in various programming languages depending on the blockchain platform being used.

Smart Contracts in Private Blockchain - Key Characteristics

- **Automation:**
  - Smart contracts are automatically executed when predefined actions when specific conditions are met.
  - In a supply chain scenario, payment can be automatically released to the supplier upon successful delivery of goods.
- **Transparency:** Smart contracts ensure transparency as their code is stored on the blockchain and is visible to all authorized participants.
- Immutability: Once deployed on the blockchain, smart contracts cannot be altered or tampered with, ensuring the integrity of agreements.
- Security: Smart contracts leverage cryptographic techniques for secure execution and validation of transactions, eliminating the risk of fraud or manipulation.
- Cost Efficiency: By eliminating intermediaries and automating processes, smart contracts reduce transaction costs associated with traditional contract execution.

Smart Contracts in Private Blockchain

Use Cases in Private Blockchains:

- Supply Chain Management: Smart contracts can automate various supply chain processes such as order fulfillment, payment settlement, and inventory tracking, enhancing efficiency and transparency.
- Financial Services: In private blockchain networks used for financial transactions, smart contracts can automate loan disbursements, asset transfers, and trade settlements, reducing manual intervention and operational costs.
- Insurance: Smart contracts can automate insurance claims processing, triggering payouts automatically when specific conditions, such as the occurrence of a predefined event, are met.
- Legal Contracts: Smart contracts can be utilized to automate various legal agreements, such as property transfers, lease agreements, and intellectual property rights management, streamlining contract execution and enforcement.

Smart Contracts in Private Blockchain

Development and Deployment Process:

- Writing Code: Developers write smart contract code using programming languages supported by the blockchain platform being used (e.g., Solidity for Ethereum).
- Testing: Smart contracts undergo rigorous testing to ensure they function as intended and are free from vulnerabilities.
- Deployment: Once tested, smart contracts are deployed onto the private blockchain network, where they become immutable and enforceable.
- Interaction: Authorized participants interact with smart contracts by sending transactions to trigger their execution or query their state.

Smart Contracts in Private Blockchain

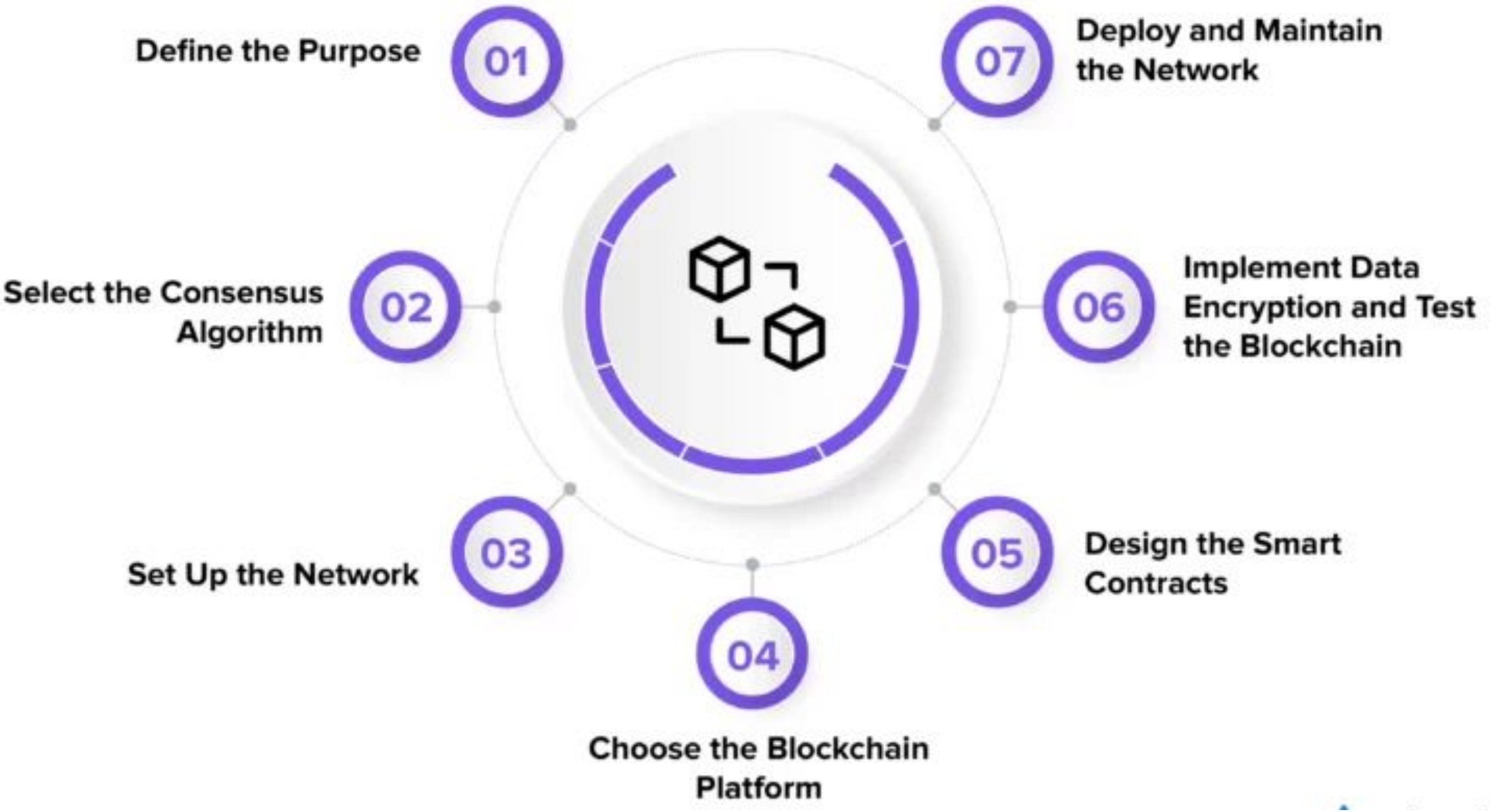
Advantages:

- Efficiency: Smart contracts automate processes, reducing manual intervention and speeding up transaction execution.
- Security: The cryptographic nature of smart contracts ensures secure and tamper-proof execution of agreements.
- Transparency: Smart contracts enhance transparency by making contract terms and execution visible to all authorized participants.
- Cost Savings: By eliminating intermediaries and automating processes, smart contracts reduce transaction costs associated with traditional contract execution.

7. Challenges:

- Complexity: Developing and deploying smart contracts requires specialized knowledge of blockchain technology and programming languages, which can be challenging for non-technical users.
- Security Risks: Smart contracts are susceptible to security vulnerabilities, such as coding errors and exploits, which can lead to financial losses or contract failures.
- Legal and Regulatory Issues: The legal enforceability of smart contracts and their compliance with existing regulations vary across jurisdictions and may pose legal challenges.

Steps to build a Private Blockchain Platform



01 Define the Purpose

02 Select the Consensus Algorithm

03 Set Up the Network

04 Choose the Blockchain Platform

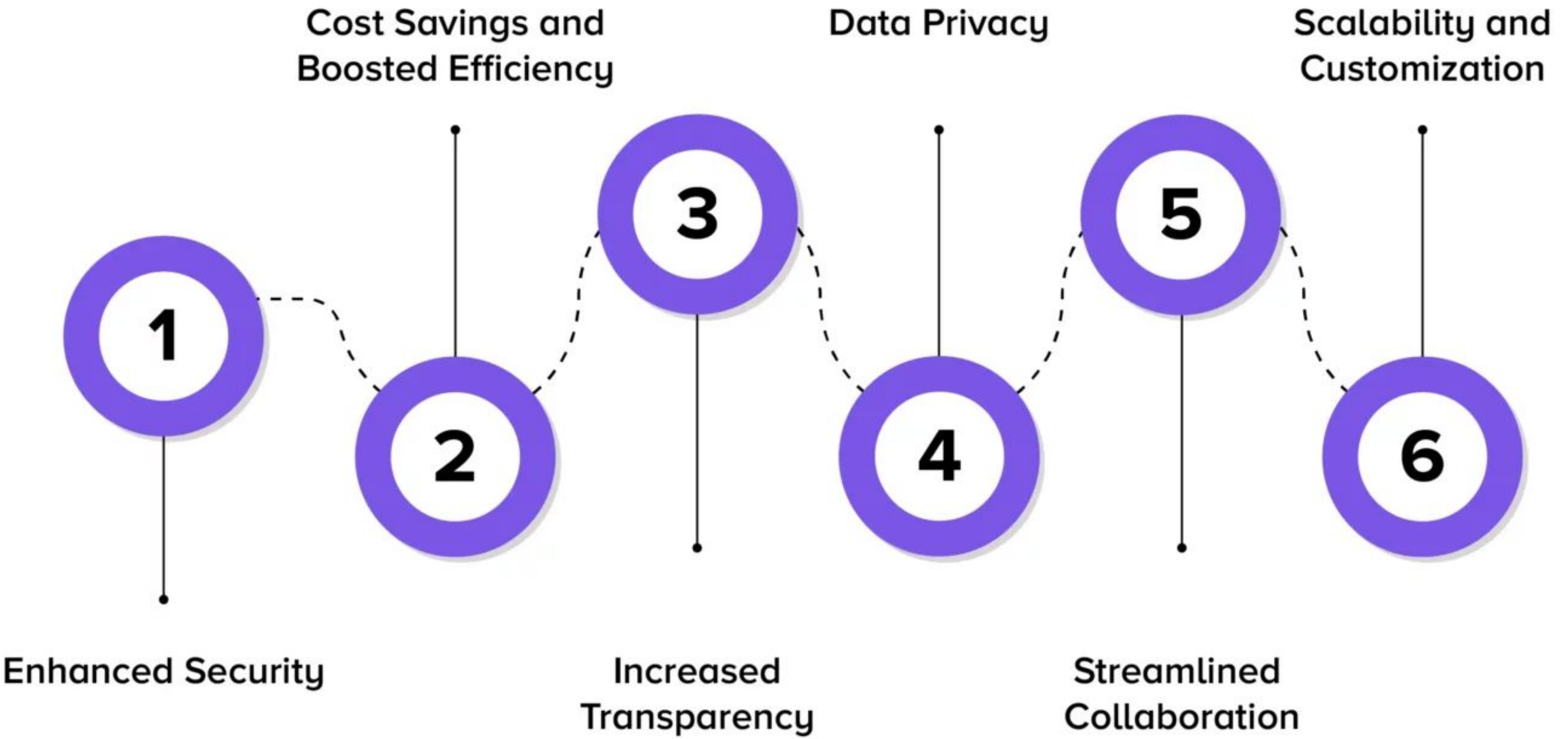
05 Design the Smart Contracts

06 Implement Data Encryption and Test the Blockchain

07 Deploy and Maintain the Network

Courtesy : appinventiv

Advantages of Private Blockchain for Businesses



1. Enhanced Security
2. Cost Savings and Boosted Efficiency
3. Increased Transparency
4. Data Privacy
5. Streamlined Collaboration
6. Scalability and Customization

Courtesy : appinventiv

Examples of Private Blockchain - HyperLedger Fabric

- prominent example of a private blockchain
- widely used for supply chain management.
- It allows businesses to securely share data and information, streamlining the complexities of supply chain operations.
- Walmart utilized Hyperledger Fabric private blockchain
  - to increase food supply chain transparency.
  - only authorized participants are allowed to securely access the shared data in a permissioned network, thereby ensuring accuracy and traceability.
  - improved food safety and reduced tracking time from days to seconds.
  - Transparency facilitated the quick identification of contamination sources, enhancing supply chain efficiency and consumer trust.

Courtesy : appinventiv

Examples of Private Blockchain - Corda & Quorum

Corda

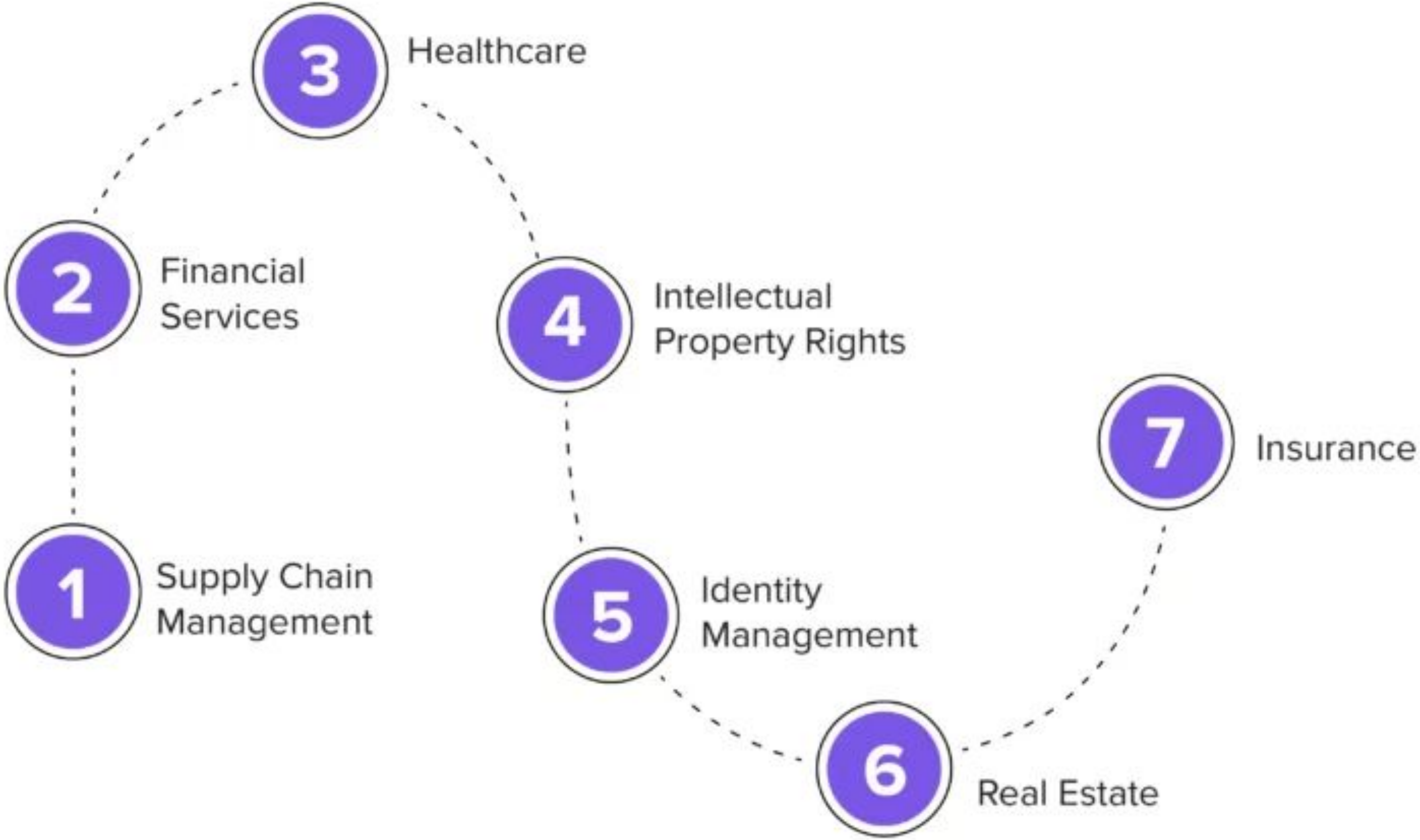
- famous private blockchain example that has made it large in the industry.
- In the BFSI ecosystem, Corda is a notable private blockchain financial organizations use.
- to securely and efficiently share sensitive financial information, making way for smoother transactions and improved collaboration within the industry.

Quorum

- For financial transactions, Quorum has emerged as a popular permissioned private blockchain. \
- Many large corporations use Quorum for its enhanced privacy and security features, making it an attractive alternative to traditional financial platforms.

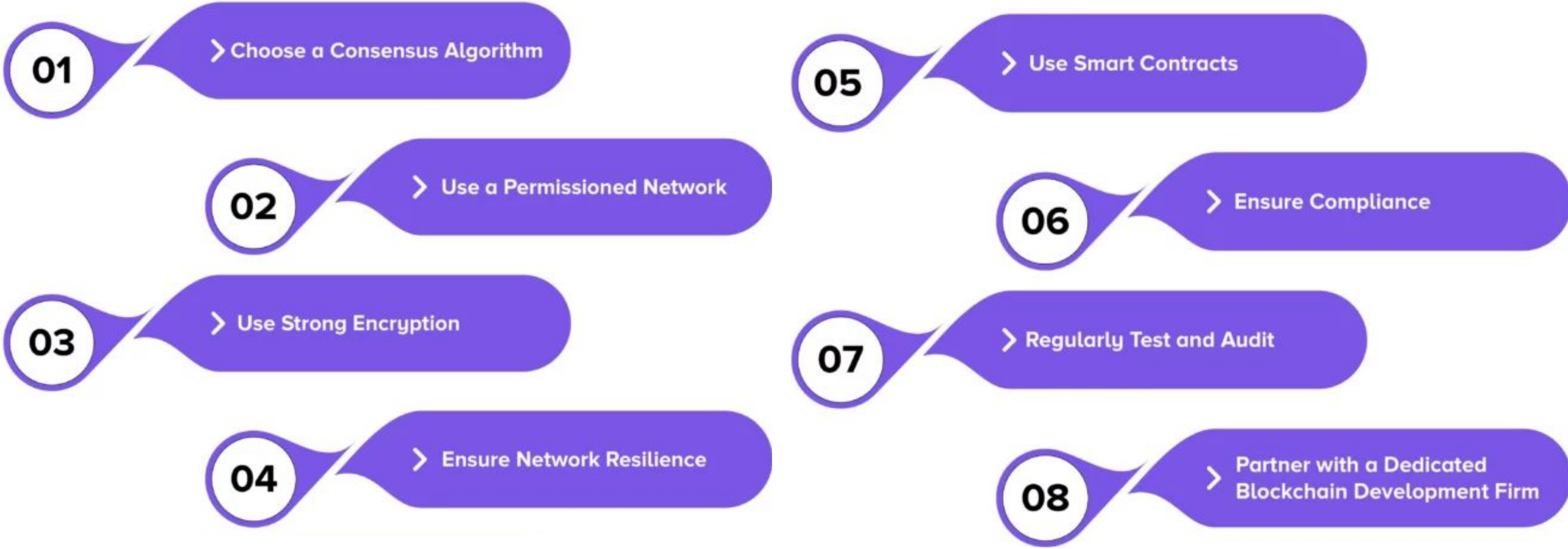
Courtesy : appinventiv

Use Cases of Private Blockchain



Courtesy : appinventiv

Best Practices for using a Private Blockchain



01 Choose a Consensus Algorithm

02 Use a Permissioned Network

03 Use Strong Encryption

04 Ensure Network Resilience

05 Use Smart Contracts

06 Ensure Compliance

07 Regularly Test and Audit

08 Partner with a Dedicated Blockchain Development Firm

Courtesy : appinventiv

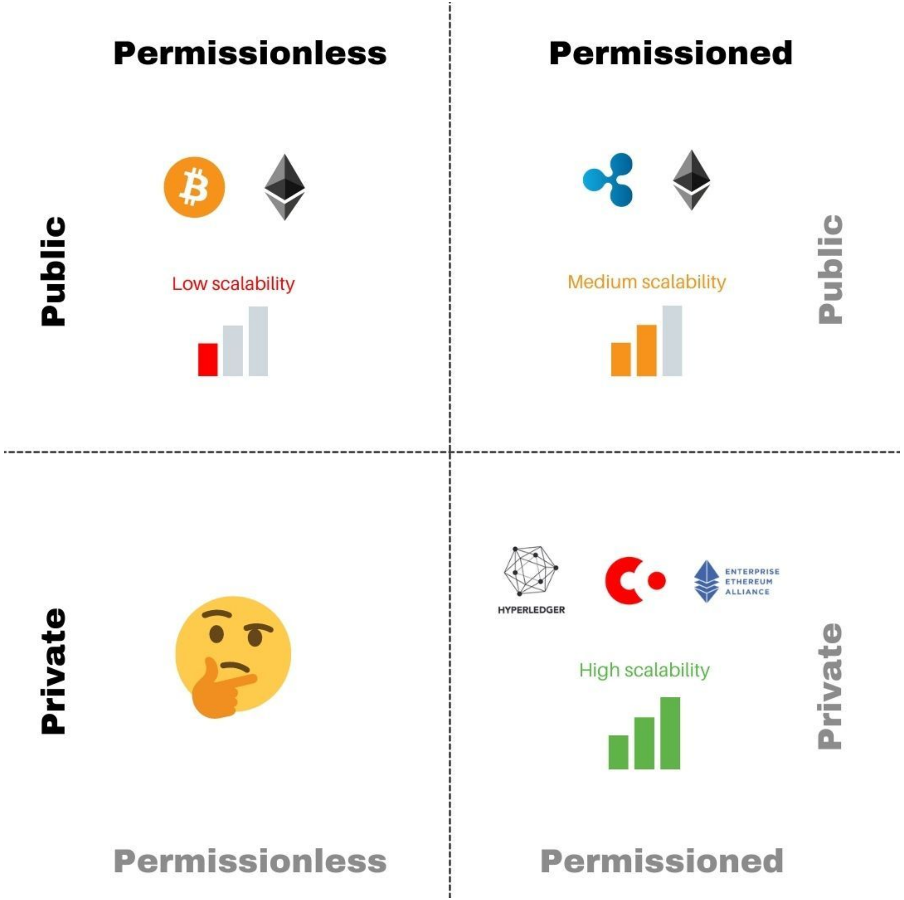
Public Blockchain Vs Private Blockchain

Feature	Public Blockchain	Private Blockchain
Access	Permissionless - Anyone can participate	Permissioned - Access restricted to authorized participants
Consensus Mechanism	Typically Proof of Work (PoW) or Proof of Stake (PoS)	Customizable - Consensus mechanism can be tailored to requirements
Governance	Decentralized - No single entity controls the network	Centralized or Consortium - Controlled by a single entity or group of organizations
Privacy	Transparent - Transactions and data are publicly visible	Enhanced Privacy - Participants have control over who can access and view transactions
Scalability	Slower due to open participation and consensus mechanisms	Faster due to limited participants and customized consensus algorithms
Transaction Throughput	Lower throughput due to consensus mechanisms and scalability issues	Higher throughput due to fewer participants and optimized consensus mechanisms

Public Blockchain Vs Private Blockchain

Feature	Public Blockchain	Private Blockchain
Use Cases	Cryptocurrency, decentralized applications (dApps), tokenization	Enterprise applications, supply chain management, financial services, consortium networks
Regulatory Compliance	Compliance may be challenging due to decentralization	Easier compliance with regulations as participants can enforce rules within the network
Network Security	Relies on incentives for security (e.g., mining rewards)	Relies on identity verification and access control for security
Example	Bitcoin, Ethereum	Hyperledger Fabric, Corda, Quorum

Public Blockchain Vs Private Blockchain



Permissionless

Permissioned

Public

Low scalability

Medium scalability

Public

Private

High scalability

Private

Permissionless

Permissioned

Courtesy : Softwaremill.com