Let's Agree to Agree:
Consensus in a Faulty System

Hanane Akeel, Vanessa Heynes, Colin James,
Arisha Khan, Luha Yang, Wesley Yang

Advised by Professor Tanya Amert

Table of contents

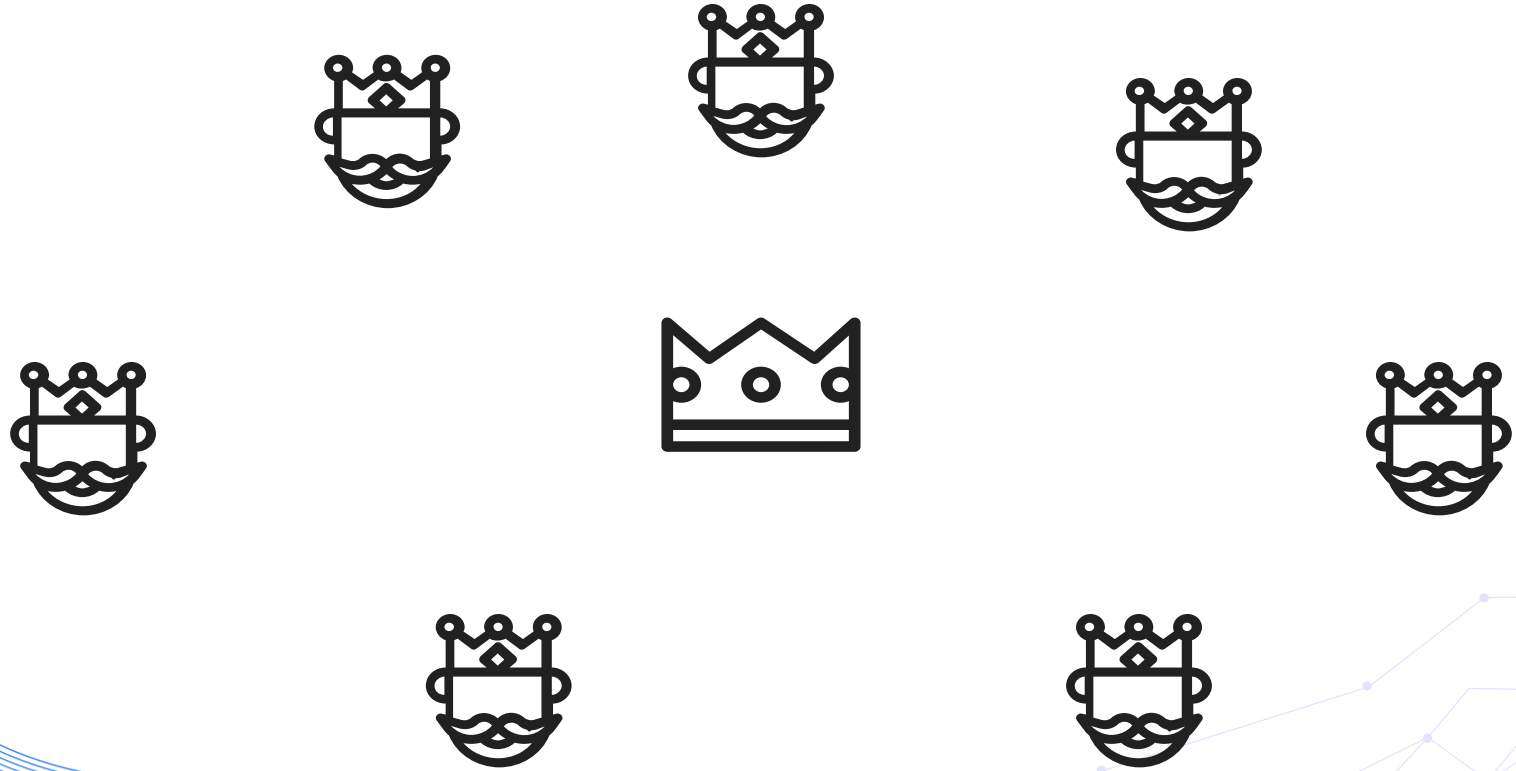
01

Byzantine Generals Problem

Byzantine Generals Problem

- The Byzantine Generals Problem is made up of **Byzantine generals** and the **Roman Empire**

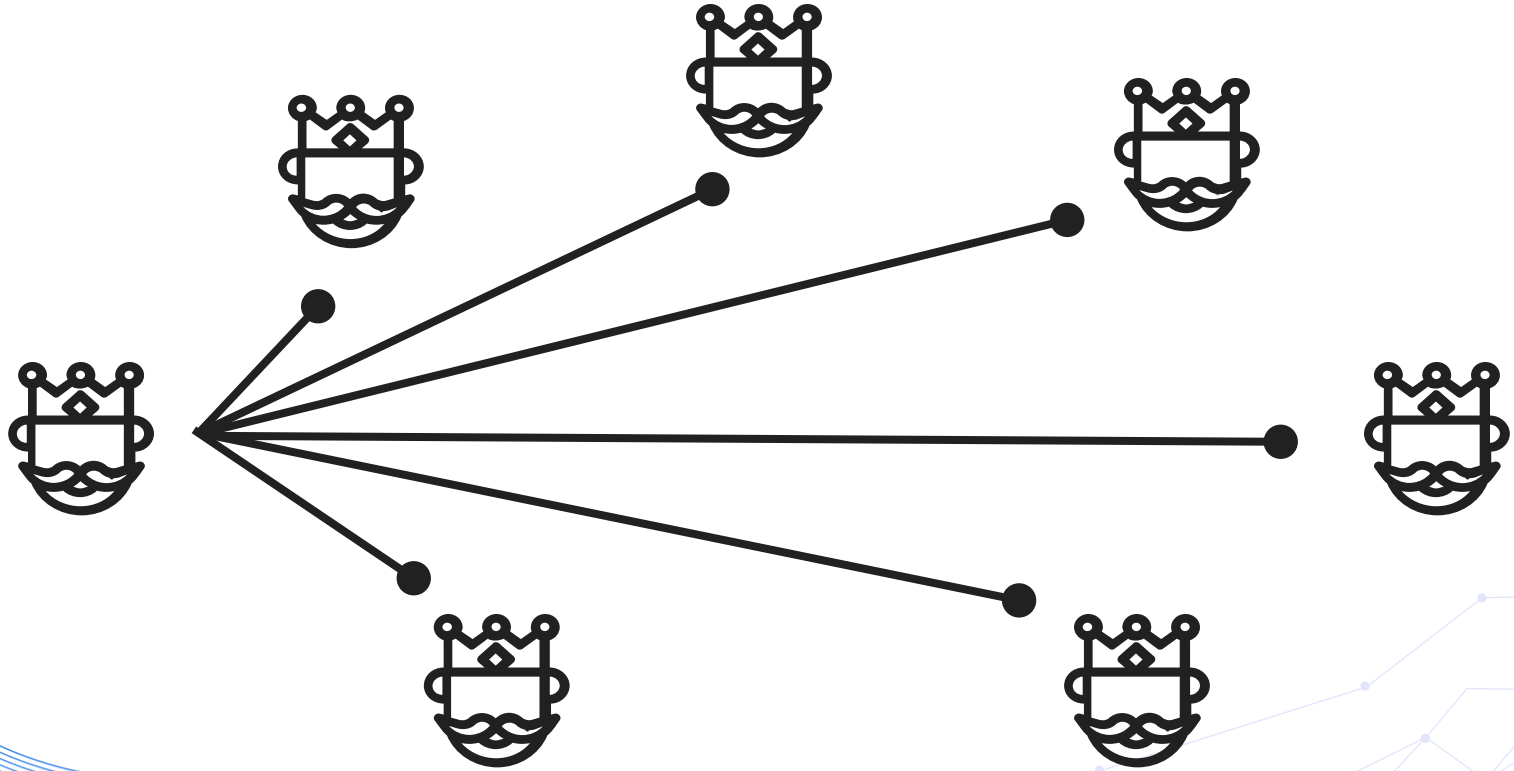
Byzantine Generals Problem



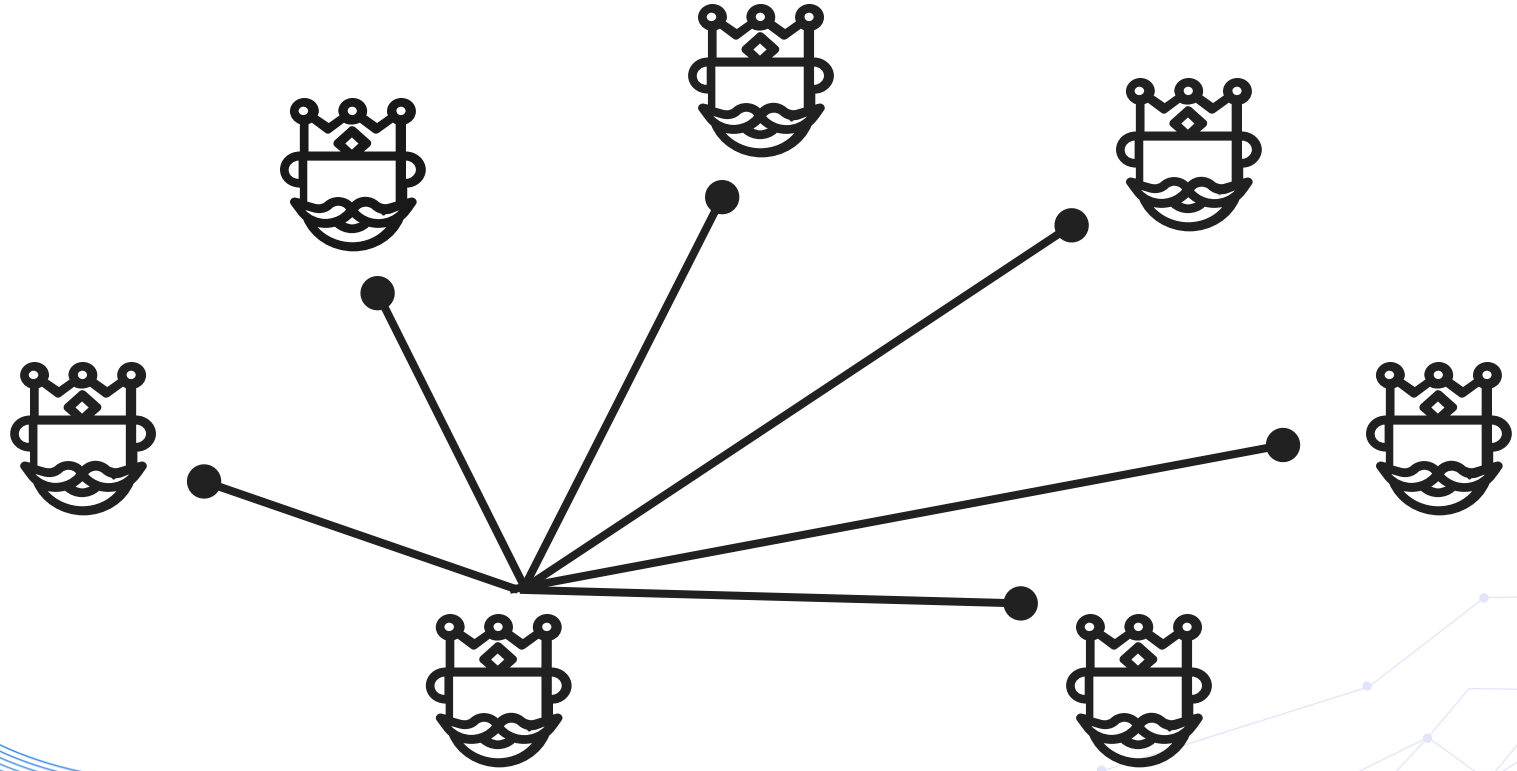
Byzantine Generals Problem

- The Byzantine Generals Problem is made up of Byzantine generals and the Roman Empire
- Each general can send **messages** to **every other general**

Byzantine Generals Problem



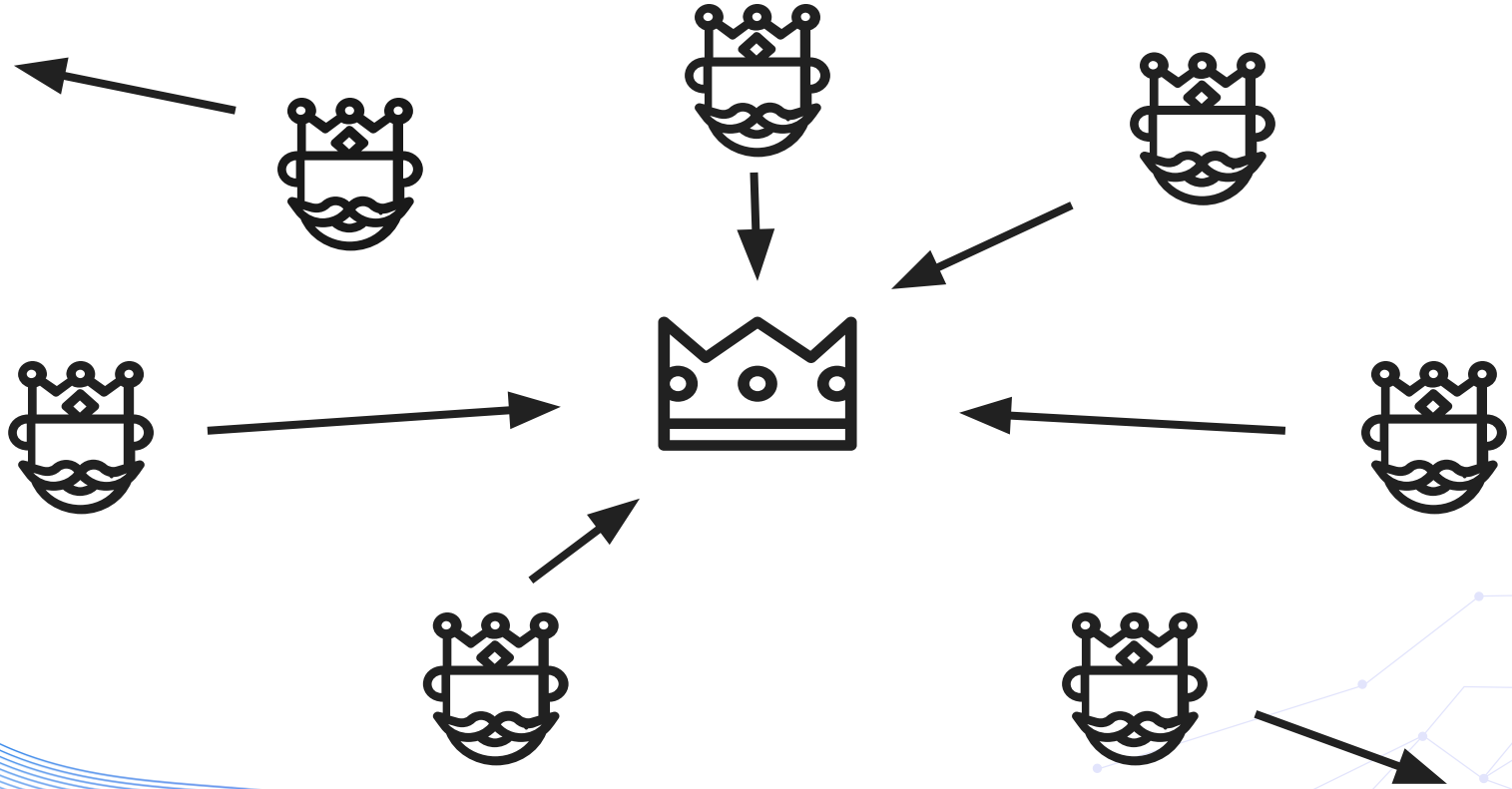
Byzantine Generals Problem



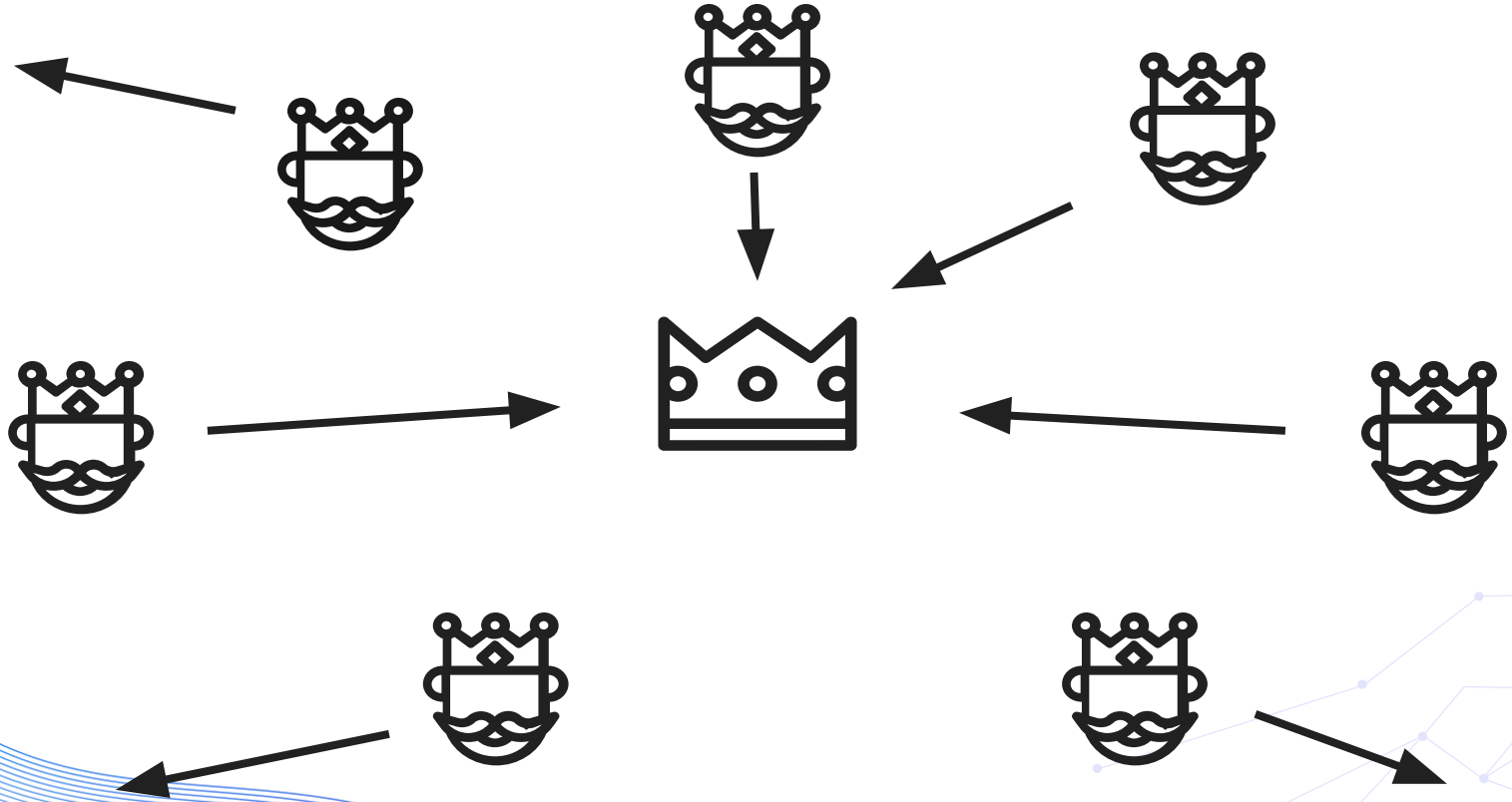
Byzantine Generals Problem

- The Byzantine Generals Problem is made up of Byzantine generals and the Roman Empire
- Each general can send messages to every other general
- Each general can **choose to either attack or withdraw**
- The **goal is to have all of the generals choose the same decision**
- To do so **more than 2/3** must agree

Byzantine Generals Problem



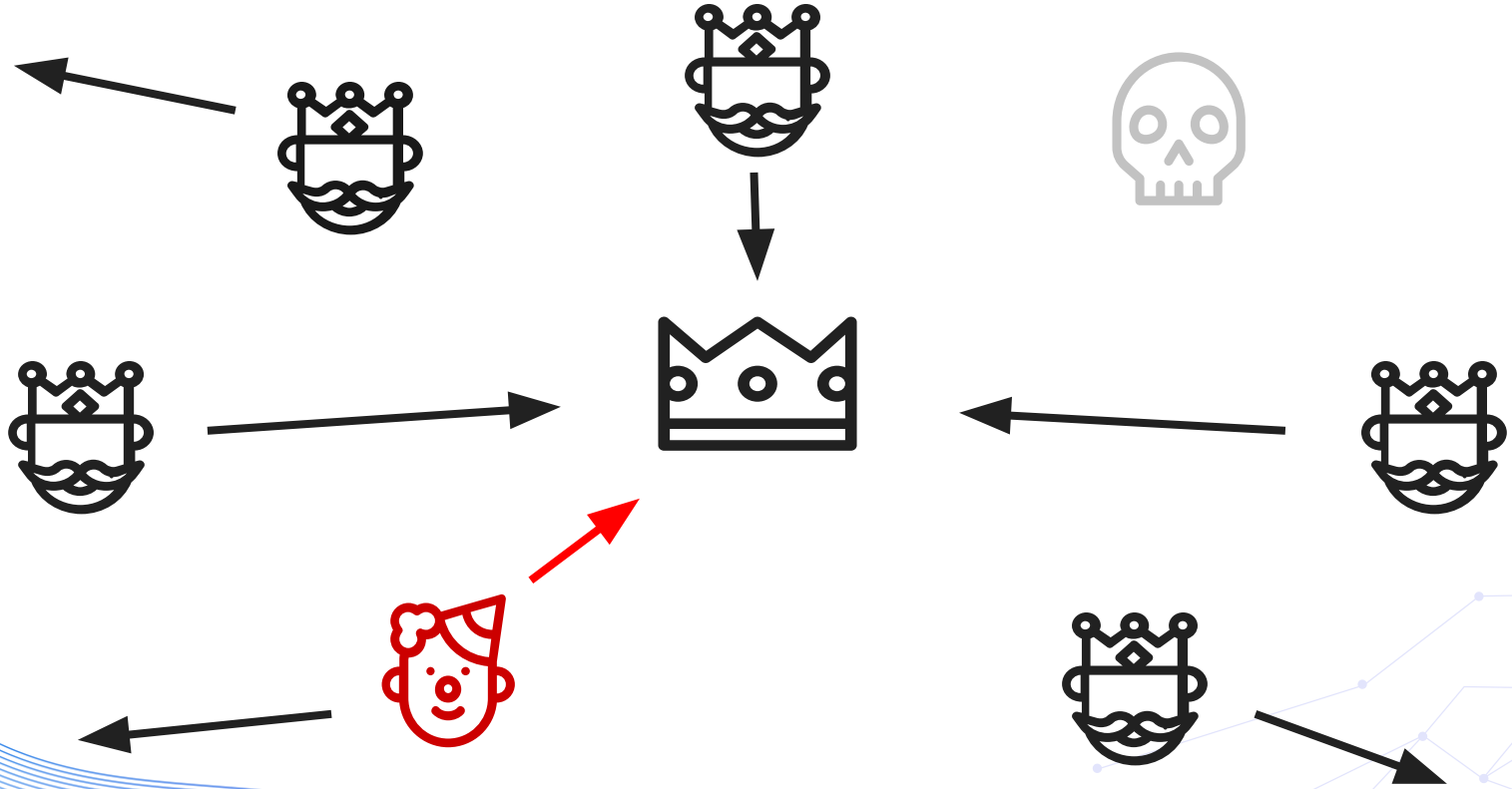
Byzantine Generals Problem



Byzantine Generals Problem

- The Byzantine Generals Problem is made up of Byzantine generals and the Roman Empire
- Each general can send messages to every other general
- Each general can **choose to either attack or withdraw**
- The **goal is to have all of the generals choose the same decision**
- To do so **more than 2/3** must agree
- **But what if within their communications, they can die, or lie**

Byzantine Generals Problem



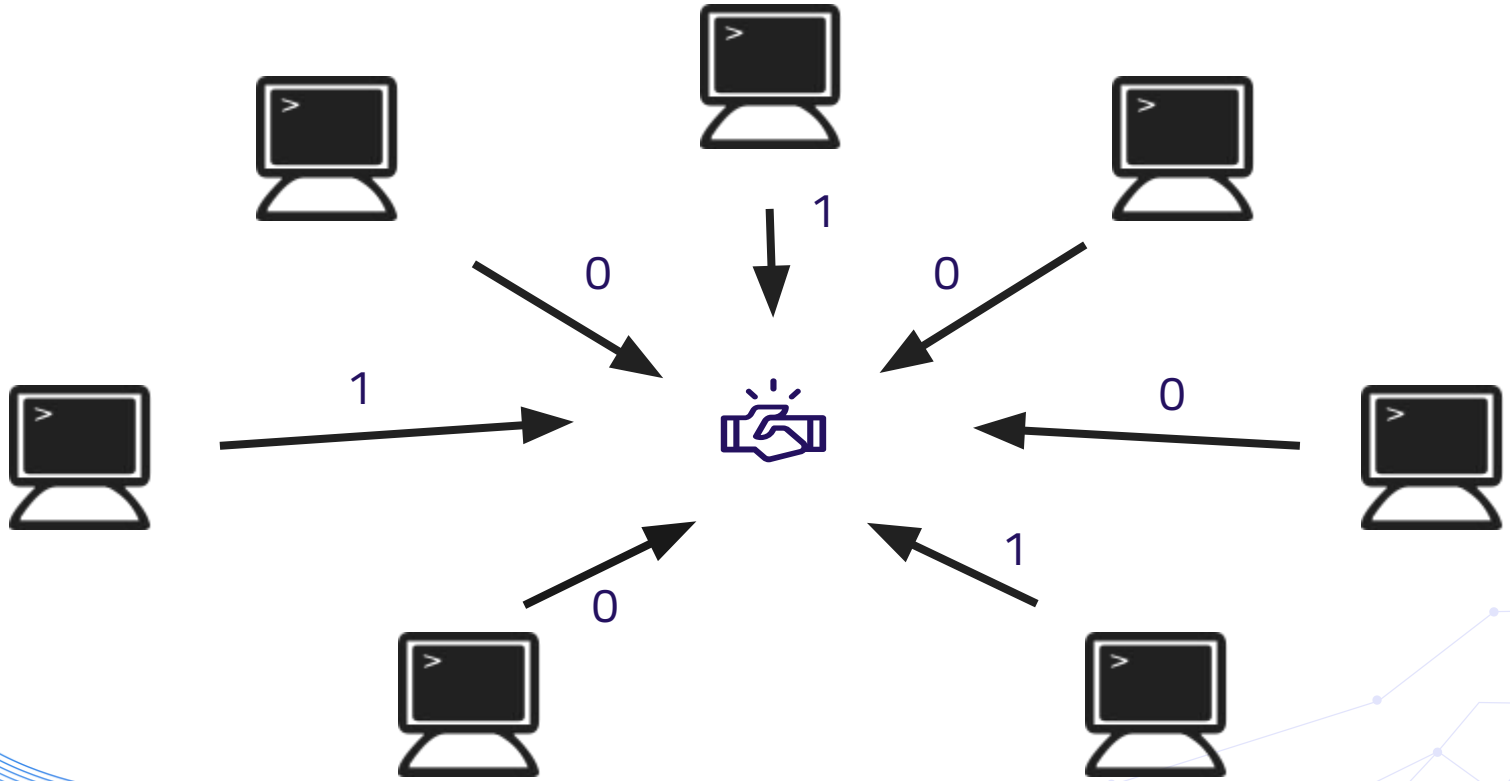
Byzantine Generals Problem

- The Byzantine Generals Problem is made up of Byzantine generals and the Roman Empire
- Each general can send messages to every other general
- Each general can choose to either attack or withdraw
- To win Rome, you must have more than ⅔ generals agree to attack
- But what if within their communications, they can die, or lie
- **Now think about a bunch of connected computers!**

Byzantine Generals Problem

- Instead of Generals, we have **computers** sending out **single values**
  → ex: ID#s, timestamps, true/false, an integer

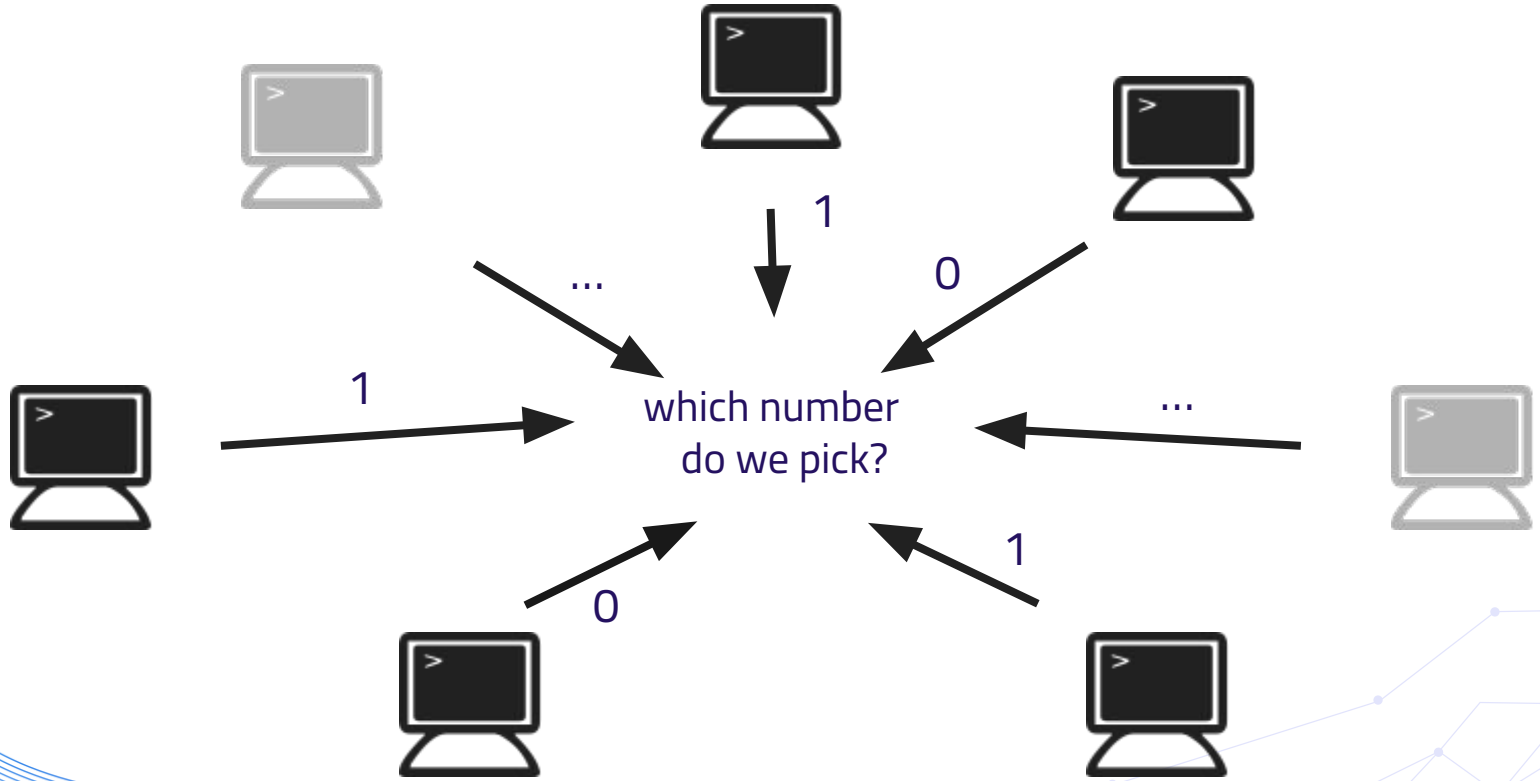
Byzantine Generals Problem



Byzantine Generals Problem

- “How do we reach agreement when computers…?”
  → Go offline
  → Send misinformation

Byzantine Generals Problem



02

Definitions

Defining Common Terms

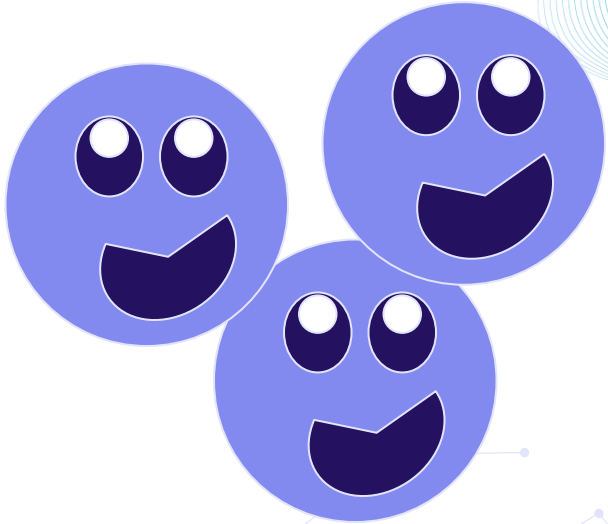
Node

- **Single computer** in a larger system
- Nodes work **independently** but can communicate and make decisions with other nodes

Defining Common Terms

Distributed System

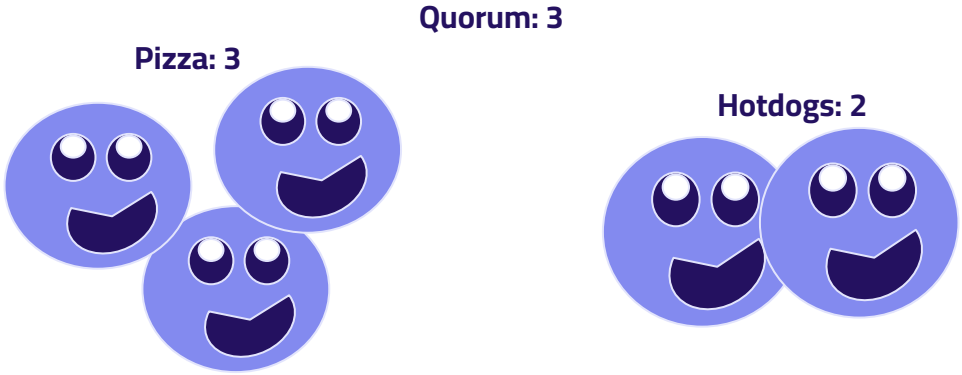
- **Group of nodes** working together as one
- Shares the work between all the nodes



Defining Common Terms

Quorum

- Minimum number of nodes that must agree in order to make a decision



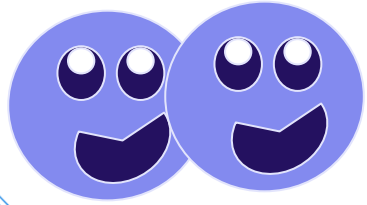
Defining Common Terms

Term

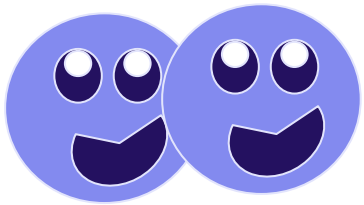
- A **round** in the decision making process

Term 1

Pizza: 2

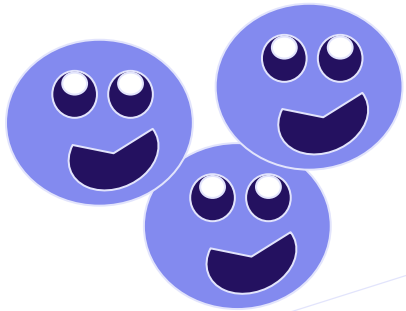


Hotdogs: 2



Term 2

Pizza: 3



Hotdogs: 1

Defining Common Terms

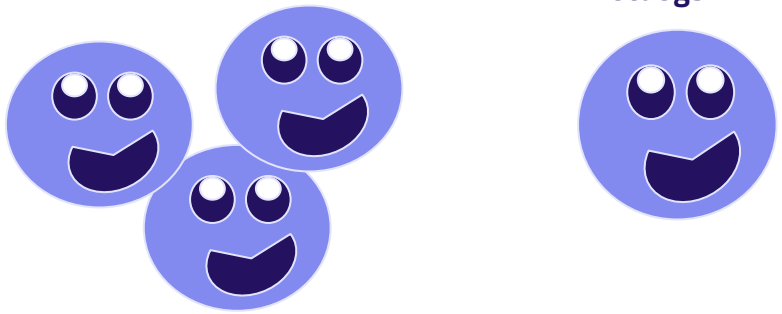
Consensus

- Majority of nodes are non-lying and non-dead nodes and they all **agree** on a value

Consensus reached!

Pizza: 3

Hotdogs: 1



Defining Common Terms

Node

An independent computer in a system

Distributed System

Network of nodes working together

Quorum

Minimum number of votes needed to make a decision

Term

Round in the decision making process

Consensus

Agreement on a value

Defining Common Terms

Synchronous

- Strict amount of time in which messages must be sent

Asynchronous

- No limit to how long a node can take to send a message

Defining Common Terms

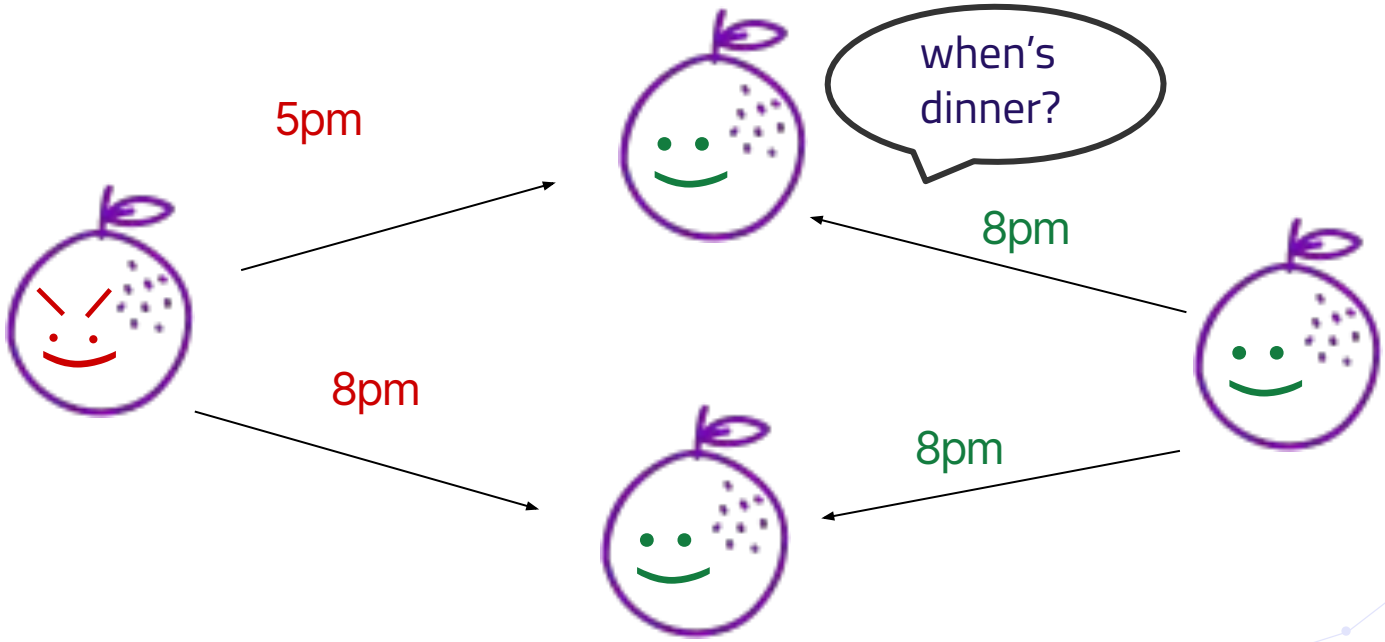
Synchronous

- Strict amount of time in which messages must be sent

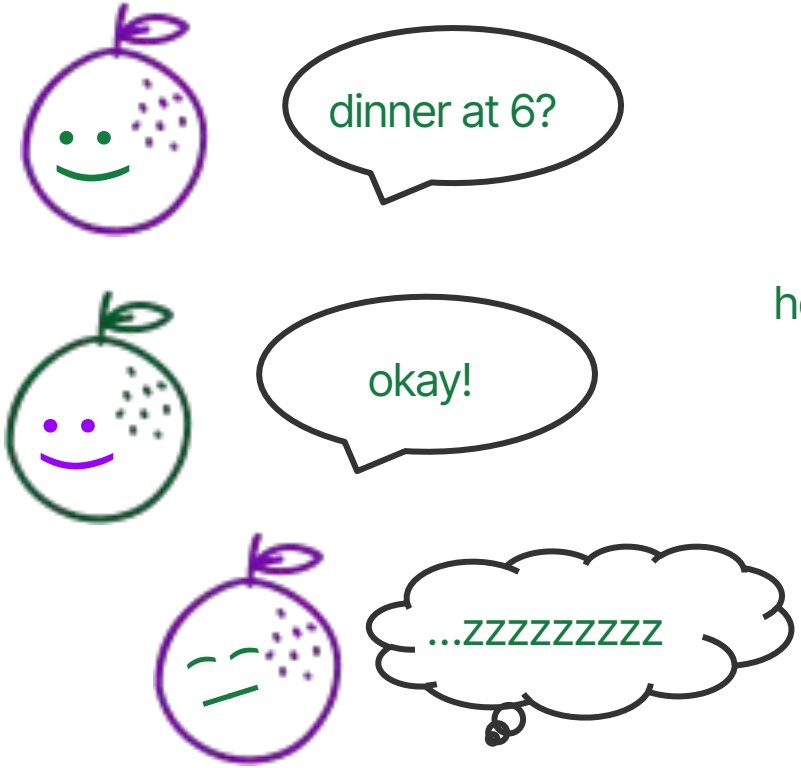
Asynchronous

- No limit to how long a node can take to send a message
- More applicable to real world applications

Defining Fault 1 - Byzantine Fault (lying node)

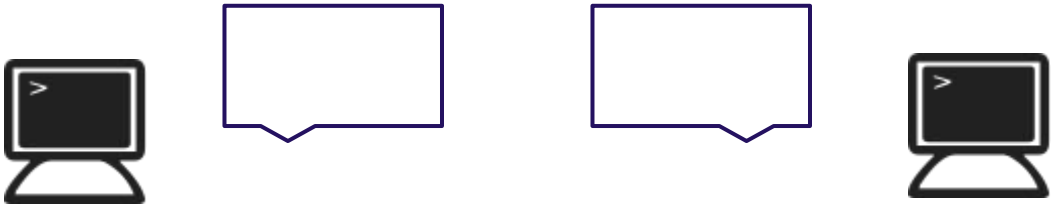


Defining Fault 2 - Network Failure (dead node)



Types of Consensus Algorithms

- With large clusters of nodes, a large part of consensus algorithm is coordinating how they communicate with one another.

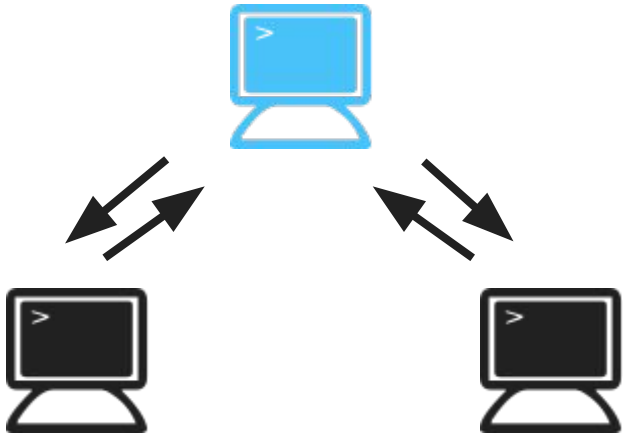


Types of Consensus Algorithms

- With large clusters of nodes, a large part of consensus algorithm is coordinating how they communicate with one another.
- Every sent message needs to be correctly received

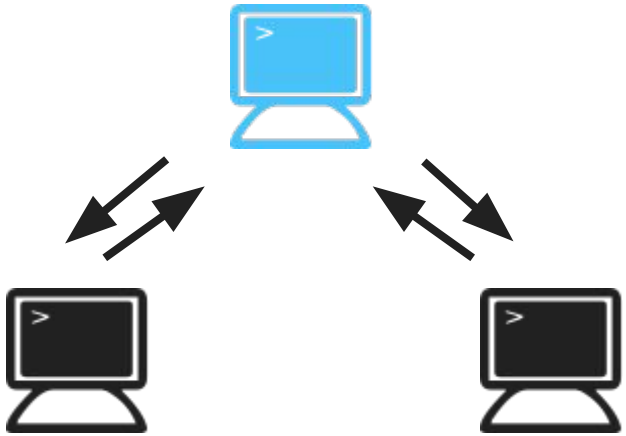
Types of Consensus Algorithms

- **Leader-based** consensus algorithms



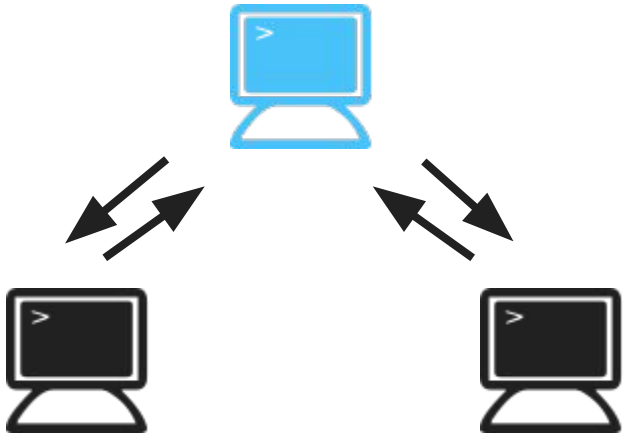
Types of Consensus Algorithms

- **Leader-based** consensus algorithms
  → easier to implement, higher efficiency



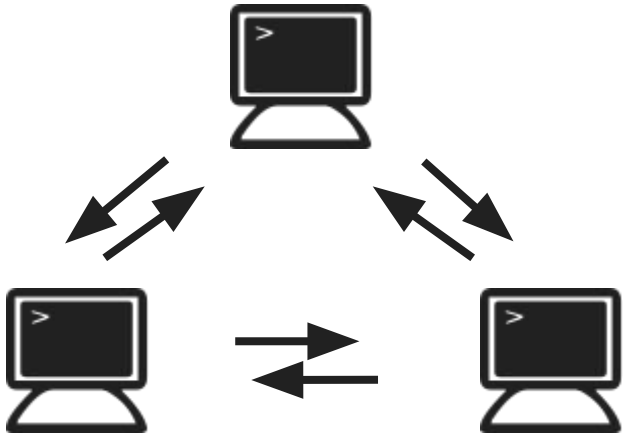
Types of Consensus Algorithms

- **Leader-based** consensus algorithms
  → easier to implement, higher efficiency
  → single point of failure



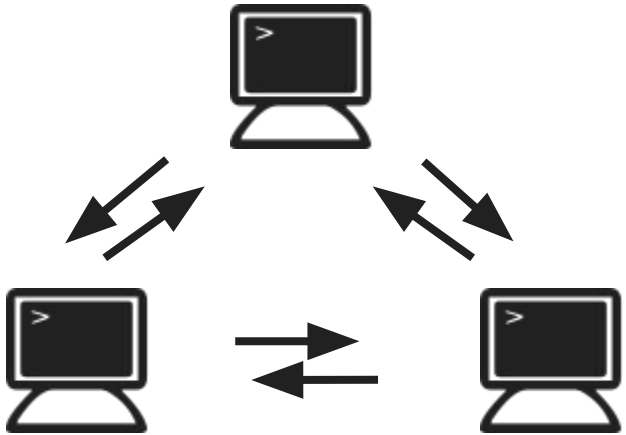
Types of Consensus Algorithms

- **Leaderless** consensus algorithms



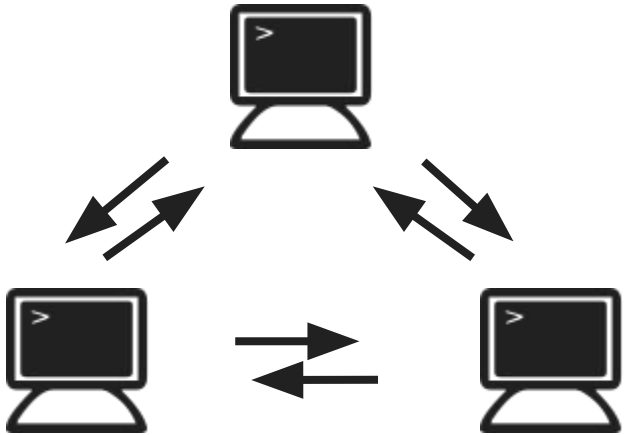
Types of Consensus Algorithms

- **Leaderless** consensus algorithms
  → no single point of failure, distributed workload



Types of Consensus Algorithms

- **Leaderless** consensus algorithms
  → no single point of failure, distributed workload
  → abstract communication



What are Consensus Protocols?

- Algorithms that allow multiple nodes in a distributed system to agree on a value or decision
- Protocols ensure that the system will remain **reliable** and **consistent** despite faults or failures in some nodes

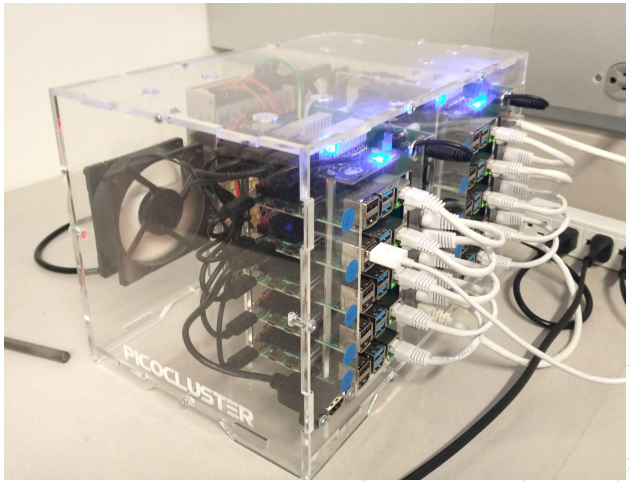
03

Project

What is our project about?

Our project is to learn about consensus protocols by implementing common consensus algorithms and comparing their performance.

- Use of the Raspberry Pi’s
- Open MPI



Motivations

Everyday, you use distributed systems, whether that be through your favorite social media app, banking, or blockchain!

Motivations

What if they were to fail?

- If bank servers don’t agree on transactions, money could disappear
- Facebook Outage (2021)
- **GRADUATE**

04

Algorithms

Algorithms

4.1

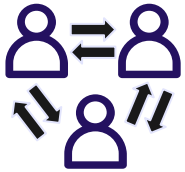
Leaderless Byzantine Paxos

4.2

Raft

4.3

HotStuff



4.1

Leaderless Byzantine Paxos

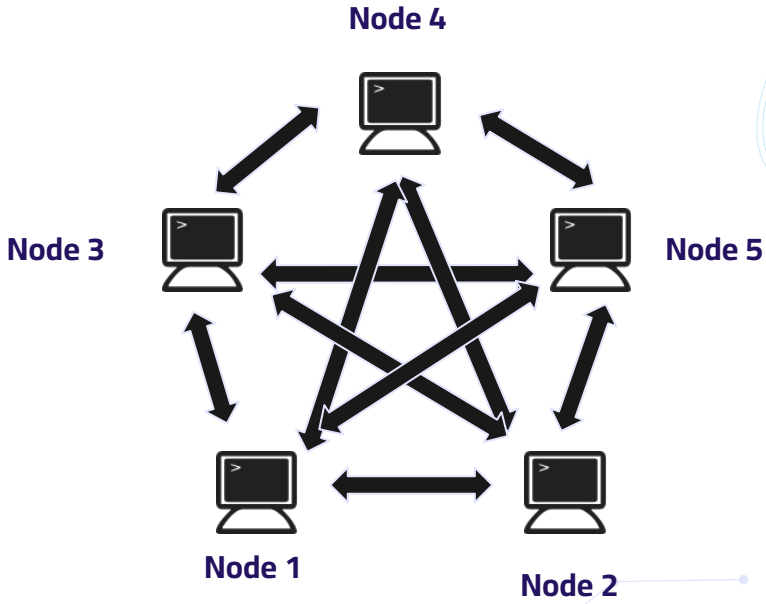
Leaderless Byzantine Paxos: Consensus Protocol

- Part of the **Paxos Consensus Protocols**
- Consensus algorithm that helps nodes reach consensus even in if some **nodes fail** or act **maliciously**
- Does not rely on a central leader, making it more **resilient** to failures

Leaderless Byzantine Paxos: Consensus Protocol

Steps:

- Nodes vote
- Vote broadcast
- Count votes and decide
- Final decision



Leaderless Byzantine Paxos: Summary Output

Number of nodes = **10**
Number of byzantine faults = **3**

```
==========================

Node 1 sees majority vote as 1, old vote: 0 new vote 1
There were 10 total nodes in this round of consensus
There were 3 faulty nodes that had a value of 0
There were 7 correct nodes that had a value of 1
Since there was over the qourum size of 7, there was the consensus decision of 1
Consensus was reached and the non-lying nodes decided 1

==========================
```

Leaderless Byzantine Paxos: Faults

- Doesn't rely on a leader - any node can propose values
- Can handle **Byzantine Faults**
- **Resilient** - no node is a single point of failure
- **More communication** and **complex** to implement

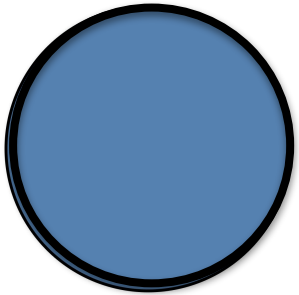
4.2

Raft:

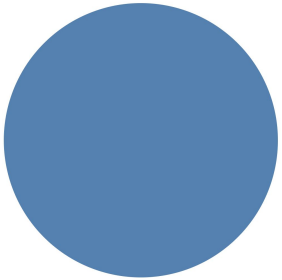
Reliable, Replicated, Redundant, and Fault-Tolerant

RAFT: Leader-Based System

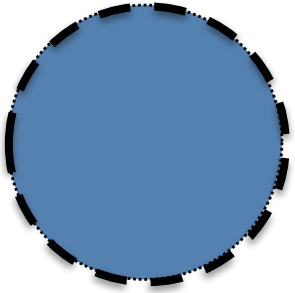
Leader State



Follower State



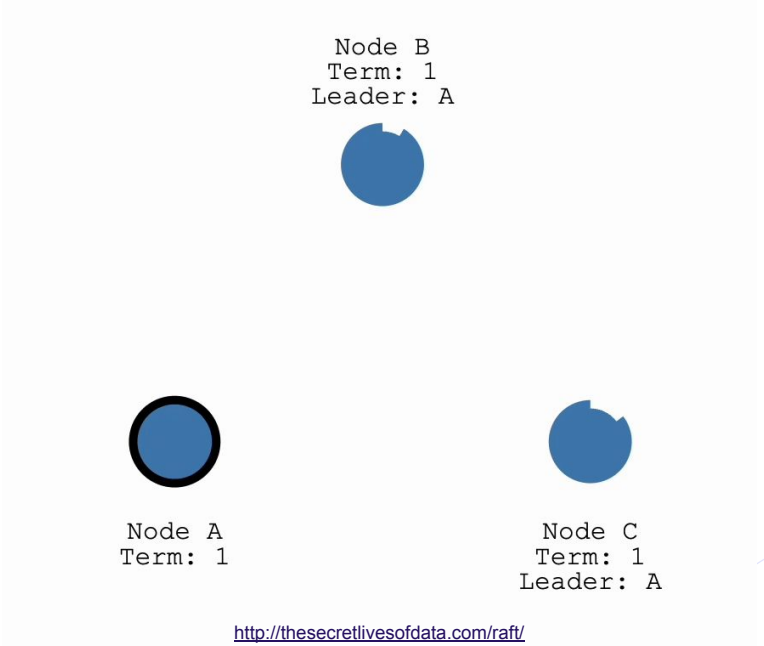
Candidate State



http://thesecretlivesofdata.com/raft/

RAFT: Heartbeat Mechanism

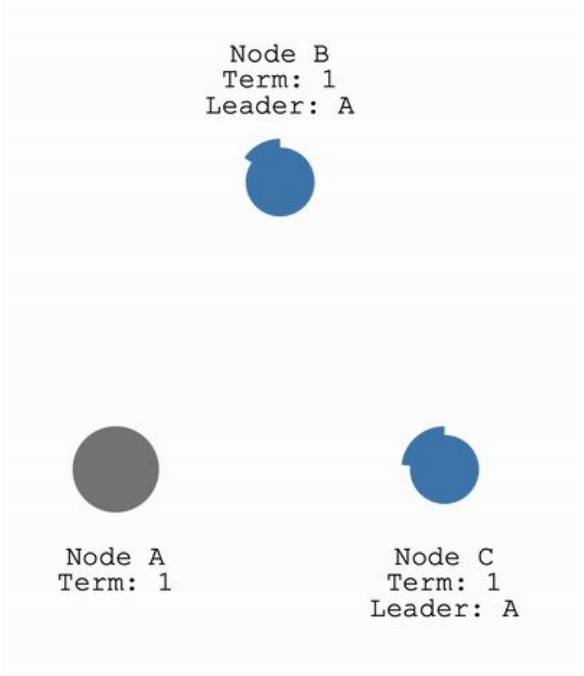
- Leader sends Heartbeat messages to followers periodically telling that it's alive



http://thesecretlivesofdata.com/raft/

RAFT: Election Process

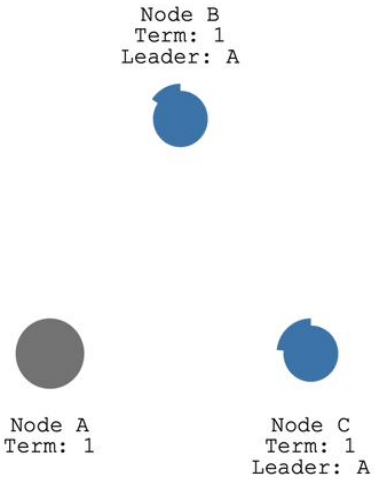
- Grey node: dead leader
- Green dots: vote requests
- Green circle: vote
- White outer circle: election timeout



http://thesecretlivesofdata.com/raft/

RAFT: Election Process

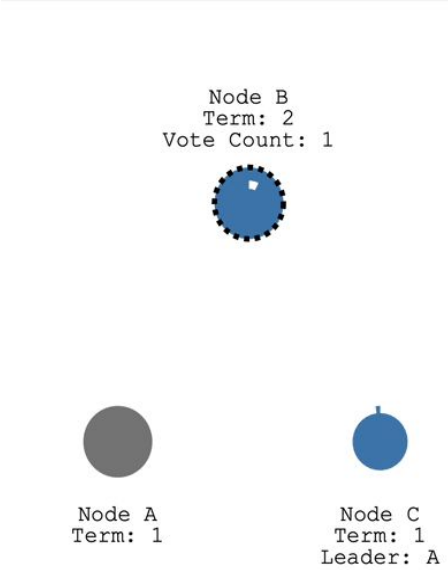
- **Election timeout** expires
  → Node B becomes candidate
  → Vote requests are sent
  and B votes for itself
  → Vote Count is 1



http://thesecretlivesofdata.com/raft/

RAFT: Election Process

- Node C **votes** to Node B
  → Node B is elected
  → Node B becomes leader in Term 2



http://thesecretlivesofdata.com/raft/

RAFT: Faults

- **Simple** and **intuitive**
- **Straightforward** to implement
- Only handles **network failures**

4.3

HotStuff

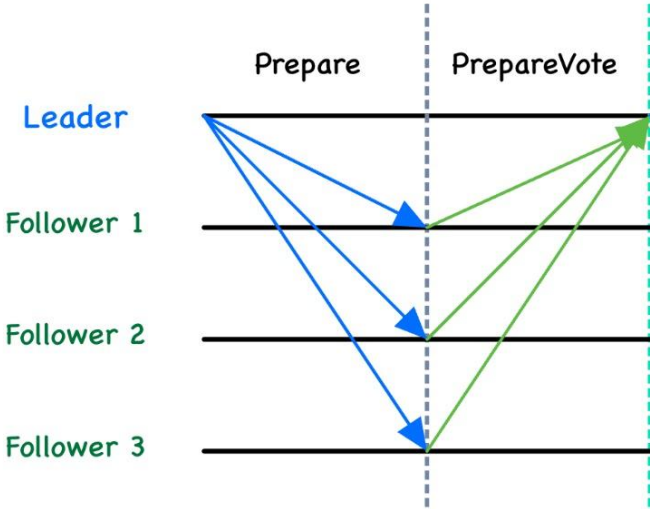
What's HotStuff

- Leader-based
- Handle **Byzantine faults** and **network failure**
- Decision is made through **multiple validation phases**
- **Faster term change** with leader rotation and term synchronization

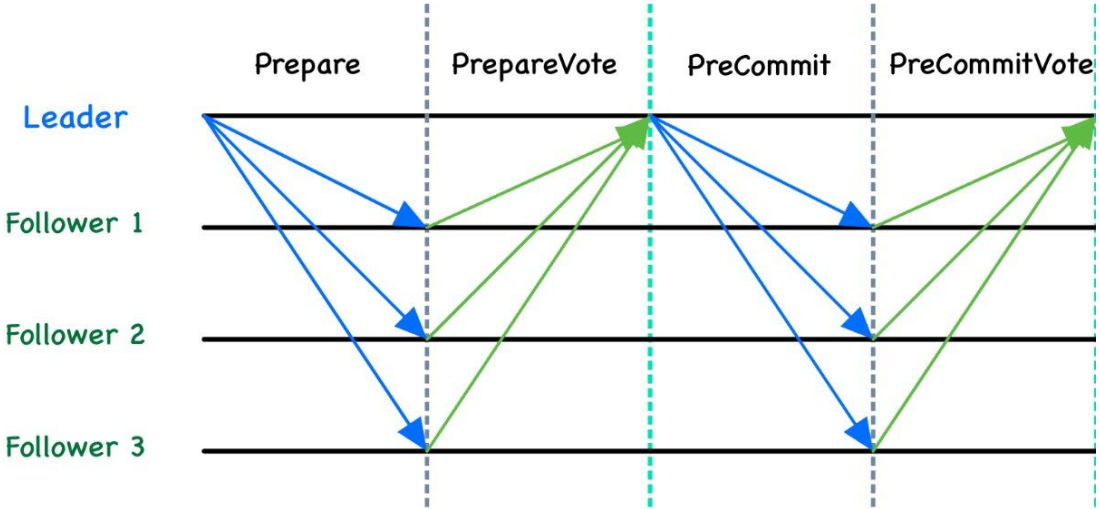
HotStuff: Term Overview

- Leader announces **new term number** and synchronizes it across system → Ensure that system operates under same term
- System goes through four phases (**prepare, precommit, commit, and decide**) to build agreement
- After finalizing decision, **next scheduled leader** takes over and starts new term

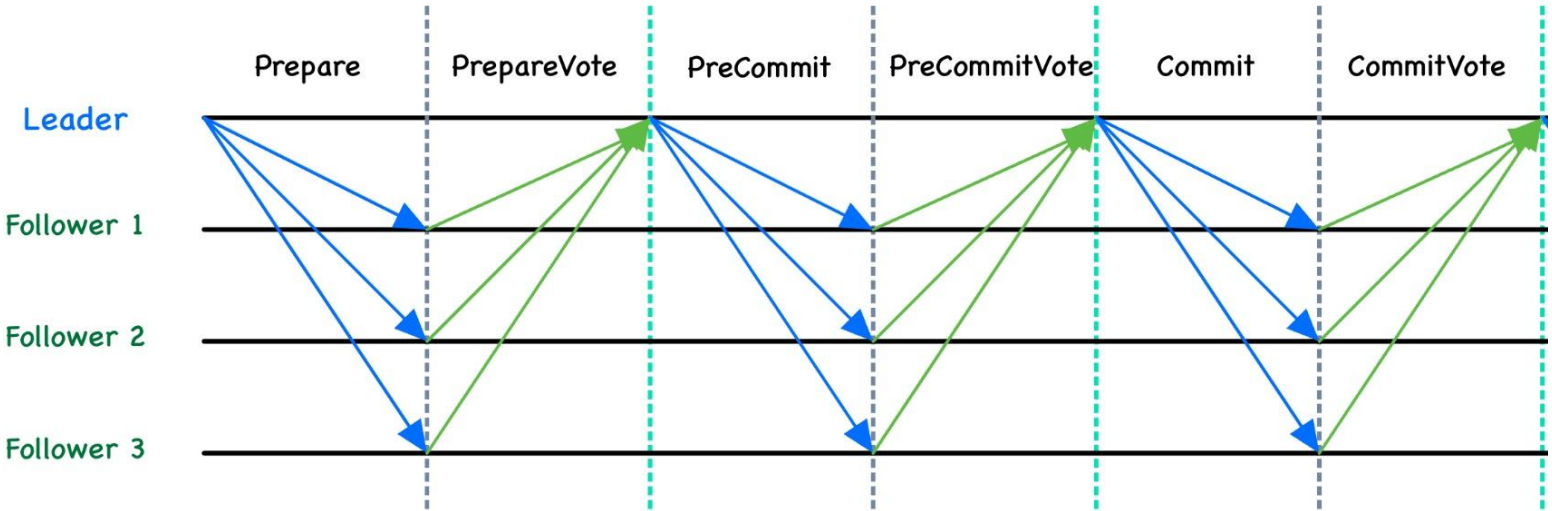
HotStuff: Four Phases



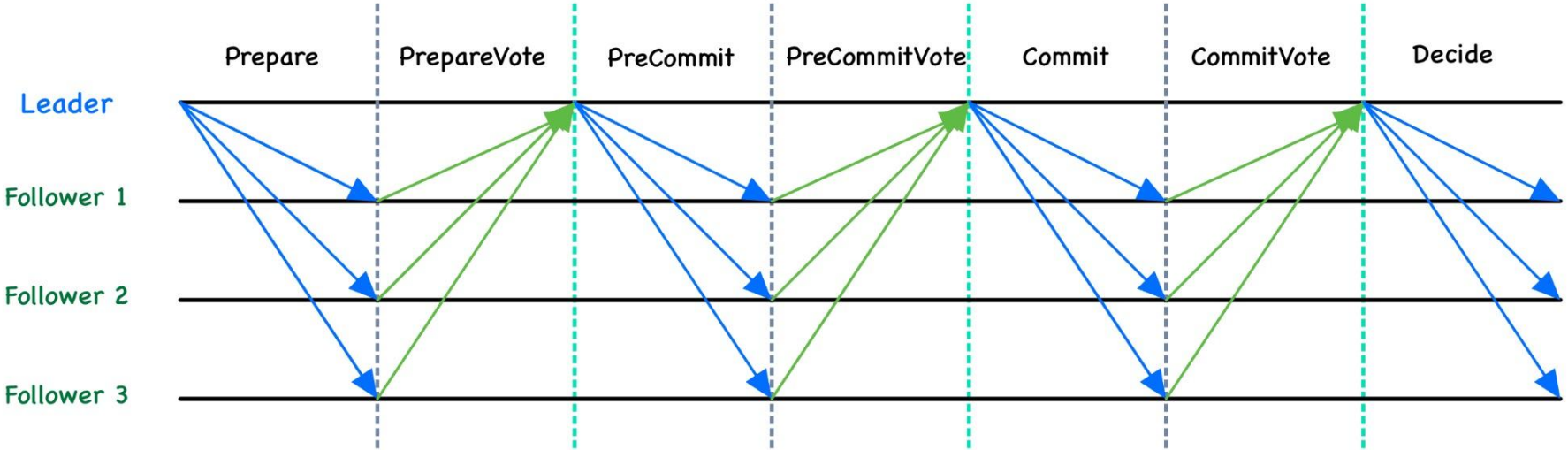
HotStuff: Four Phases



HotStuff: Four Phases



HotStuff: Four Phases



HotStuff: Round-Robin Leader Rotation

- Leader is rotated following a **fixed round-robin schedule**

Leader Schedule

Node 0 → Node 1 → Node 2 → Node 3 → Node 0 → Node 1 → Node 2 → ...

Term 0 Term 1 Term 2 Term 3 Term 4 Term 5 Term 6 ...

HotStuff: Round-Robin Leader Rotation

- Leader is rotated following a **fixed round-robin schedule**
- **Fault-tolerance**: even if a leader fails, it's replaced by next leader
- **Security**: less risk of leader-based attacks

HotStuff: Why? It's Better!

In 2022 Jalazai et al. released a paper, publishing HotStuff Consensus Protocol

HotStuff: Why? It's Better!

In 2022 Jalazai et al. released a paper, publishing HotStuff Consensus Protocol

- **Linear Term Change**: The process of switching leadership requires linearly scaling communication complexity
  - (The leader changes quickly)

HotStuff: Why? It's Better!

In 2022 Jalazai et al. released a paper, publishing HotStuff Consensus Protocol

- Linear Term Change: The process of switching leadership requires linearly scaling communication complexity
  - (The leader changes quickly)

- **Responsiveness**: Latency is based on network delays rather than average time bounds
  - (HotStuff is as fast as the wifi you're using)

4.3.1

HotStuff Implementation: **Consensus Reached**

HotStuff Output 1

Number of nodes = **4**
Number of byzantine faults = **1**

```
=== Term 1, Leader 0 ===
All phases reached quorum ✅
Lying Node 1: Sent message 2 🐍
All honest nodes (0, 2, 3) sent 42 👍
Nodes logged value 42 📝
Consensus reached! 🎉
==========================
```

HotStuff Output 1

Number of nodes = **4**
Number of byzantine faults = **1**

```
=== Term 2, Leader 1 ===
All phases reached quorum ✅
Lying Node 1: Sent message 2 🐍
All honest nodes (0, 2, 3) sent 42 👍
Nodes logged value 42 📝
Consensus reached! 🎉
==========================
```

HotStuff Output 1

Number of nodes = **4**
Number of byzantine faults = **1**

```
=== Term 3, Leader 2 ===
All phases reached quorum ✅
Lying Node 1: Sent message 2 🐍
All honest nodes (0, 2, 3) sent 42 👍
Nodes logged value 42 📝
Consensus reached! 🎉
==========================
```

4.3.2

HotStuff Implementation: **Consensus NOT Reached**

HotStuff Output 2

Number of nodes = **4**
Number of byzantine faults = **2**

```
=== Term 1, Leader 0 ===
All phases did not reach quorum ❌
Lying Node 1: Sent message 2 🐍
Lying Node 3: Sent message 2 🐍
All honest nodes (0, 2) sent 42 👍
Consensus was not reached 😔 too many evil lying nodes
============================
```

4.4

Quorums

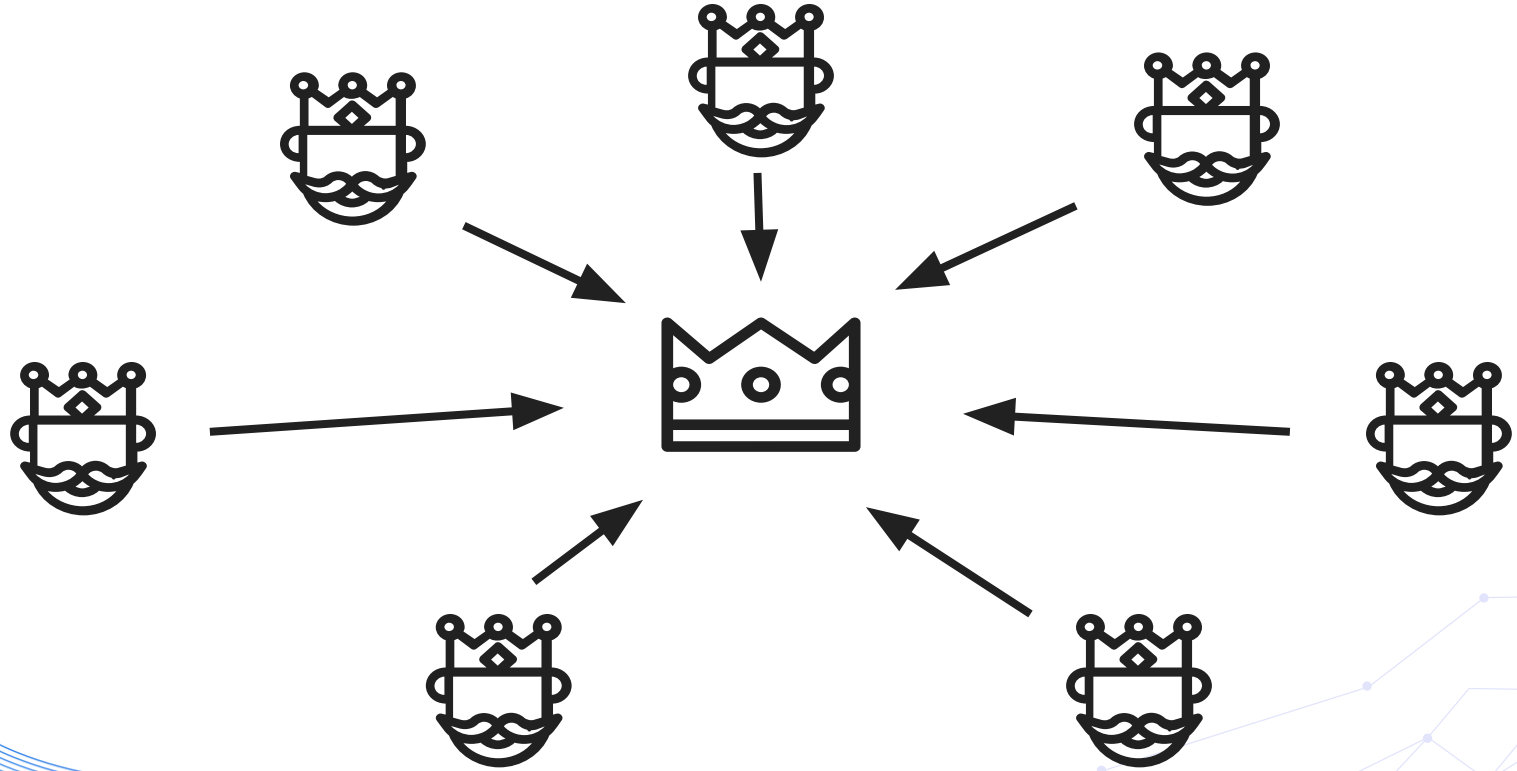
Quorums Explained

Quorum:

Minimum number of agreeing nodes in order to prove consensus!

Basically majority?

Quorums Explained



Quorums Explained

How do you decide quorums for different consensus protocols?

Quorums Explained

How do you decide quorums for different consensus protocols?

Turns out it depends on the type of faults you're accounting for.

Quorums Explained

Network Failure

Maximum Network Failure Nodes (*f*):
(n-1)/2

Necessary Correct Nodes:
***f*+1**

Byzantine Nodes

Maximum Byzantine Failure Nodes (*f*):
(n-1)/3

Necessary Correct Nodes:
2*f*+1

Quorums Explained: Network Failure

Theorem 1

There is no consensus protocol that can handle (n/2) faulty nodes

Theorem 2

There is a consensus protocol that can handle ((n-1)/2) faulty nodes

Quorums Explained: Network Failure

Theorem 1

There is no consensus protocol that can handle (n/2) faulty nodes

Theorem 2

There is a consensus protocol that can handle ((n-1)/2) faulty nodes

Proven due to lack of
Consistency

Quorums Explained: Network Failure

Theorem 1

There is no consensus protocol that can handle (n/2) faulty nodes

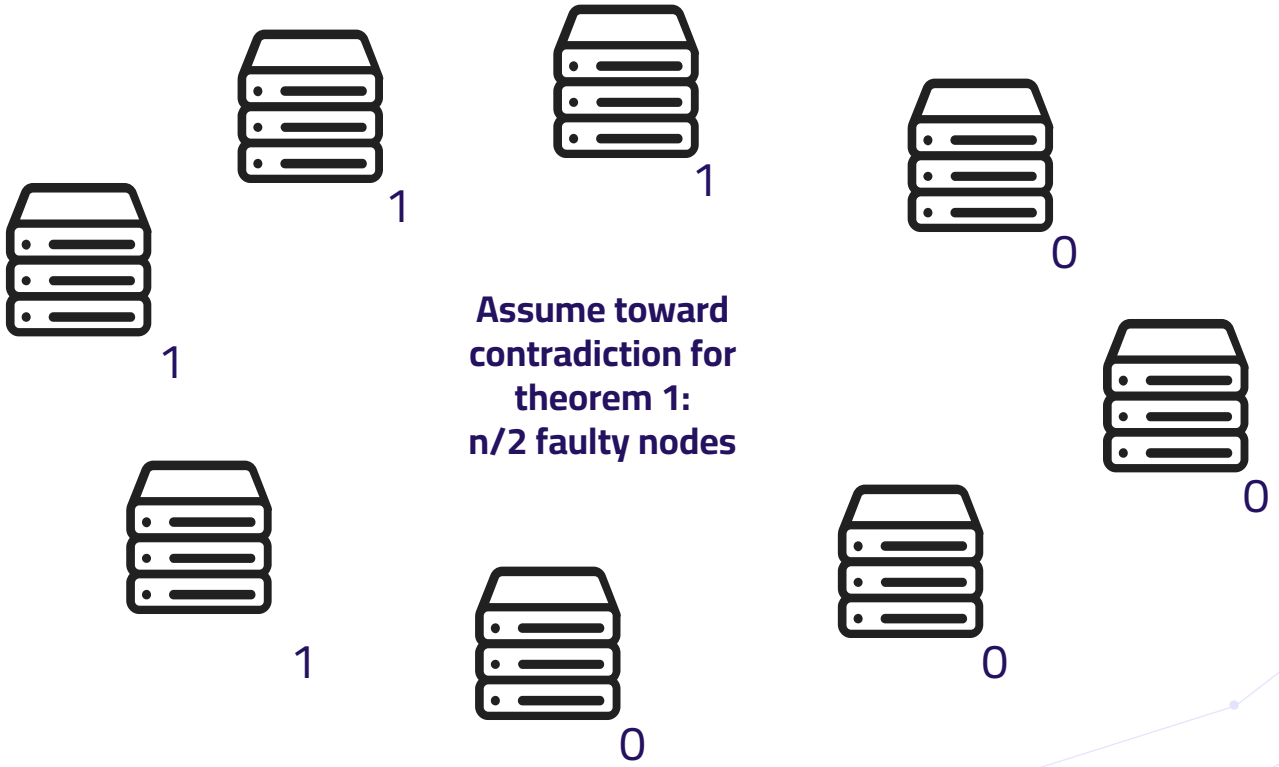
Theorem 2

There is a consensus protocol that can handle ((n-1)/2) faulty nodes

Proven due to lack of Consistency

The idea that there is no reachable configuration where correct processes decide different values.

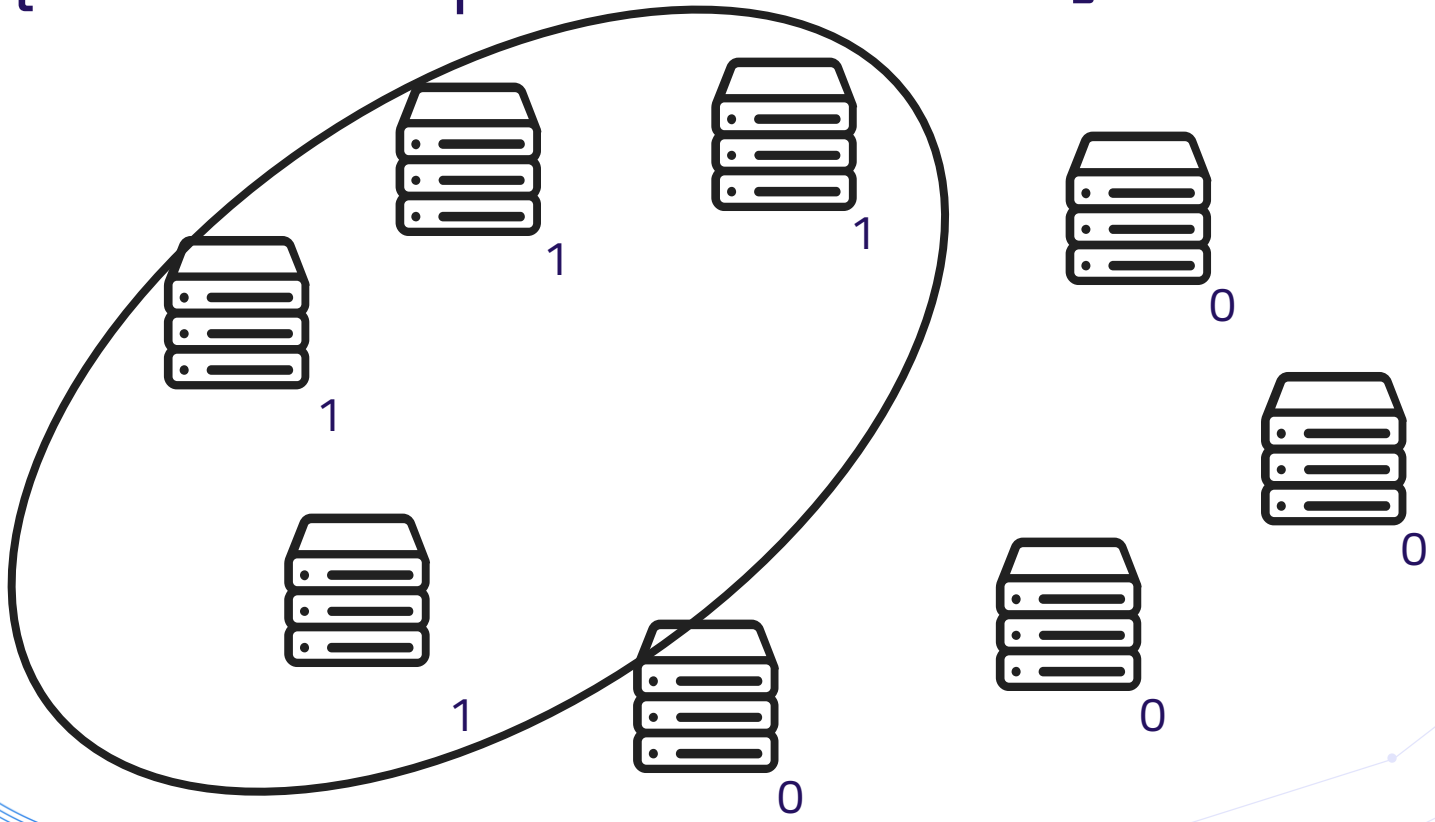
Quorums Explained: Consistency



Assume toward contradiction for theorem 1: n/2 faulty nodes

Quorums Explained: Consistency

Quorums Explained: Consistency



Quorums Explained: Byzantine Nodes

Theorem 3

There is no consensus protocol that can handle n/3 byzantine nodes.

Theorem 4

There is a consensus protocol that can handle ((n-1)/3) byzantine nodes.

Quorums Explained: Byzantine Nodes

Theorem 3

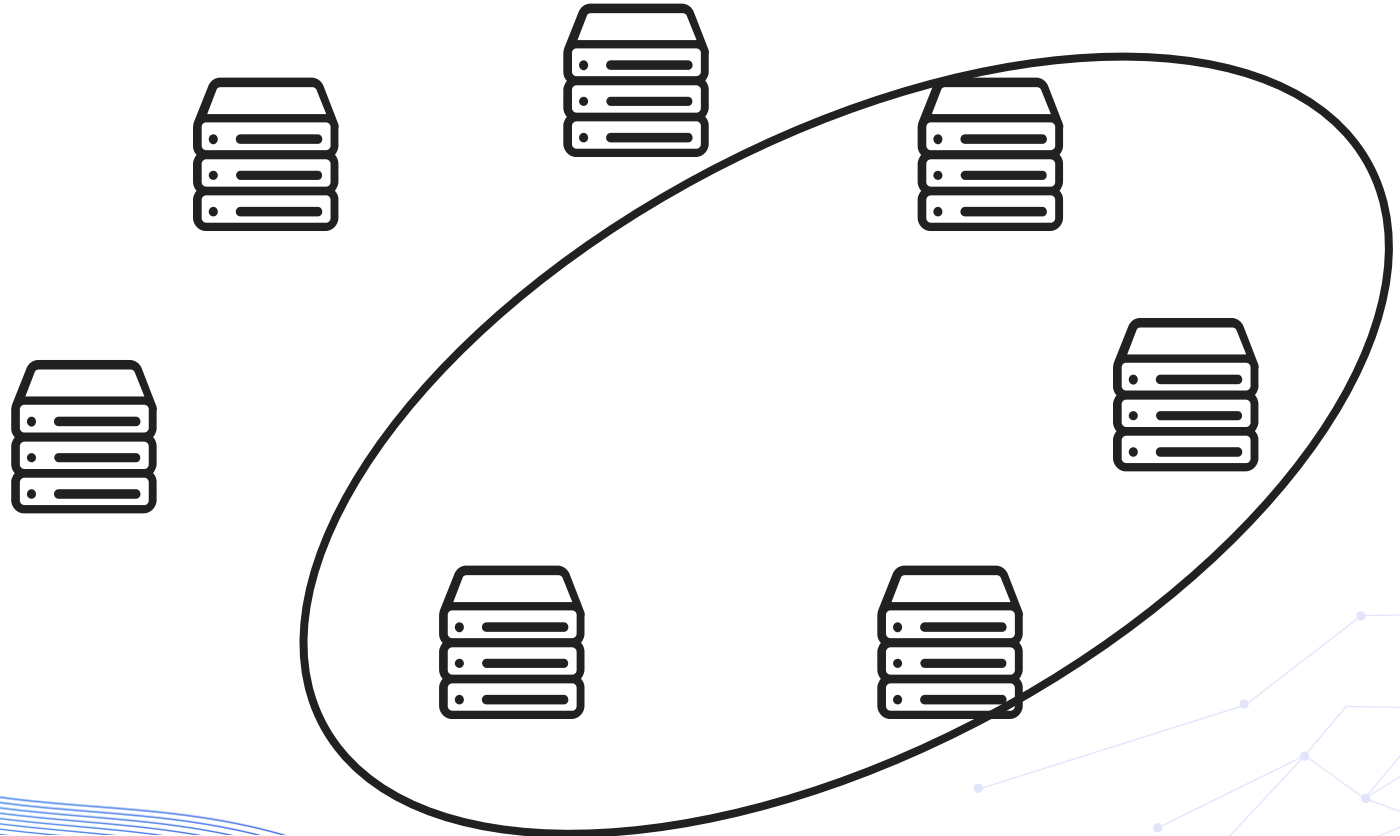
There is no consensus protocol that can handle n/3 byzantine nodes.

Theorem 4

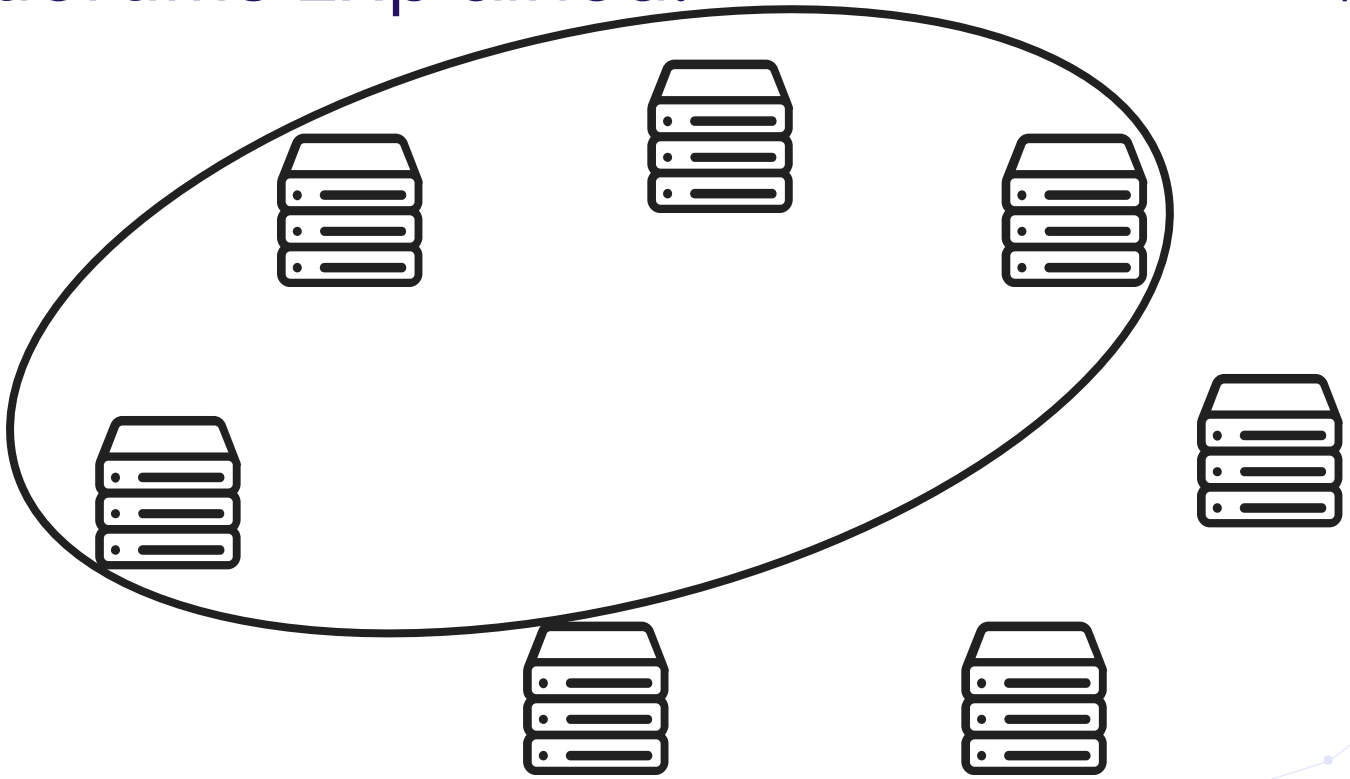
There is a consensus protocol that can handle ((n-1)/3) byzantine nodes.

Proven due to not holding **Quorum Intersection Property** resulting in **lack of consistency.**

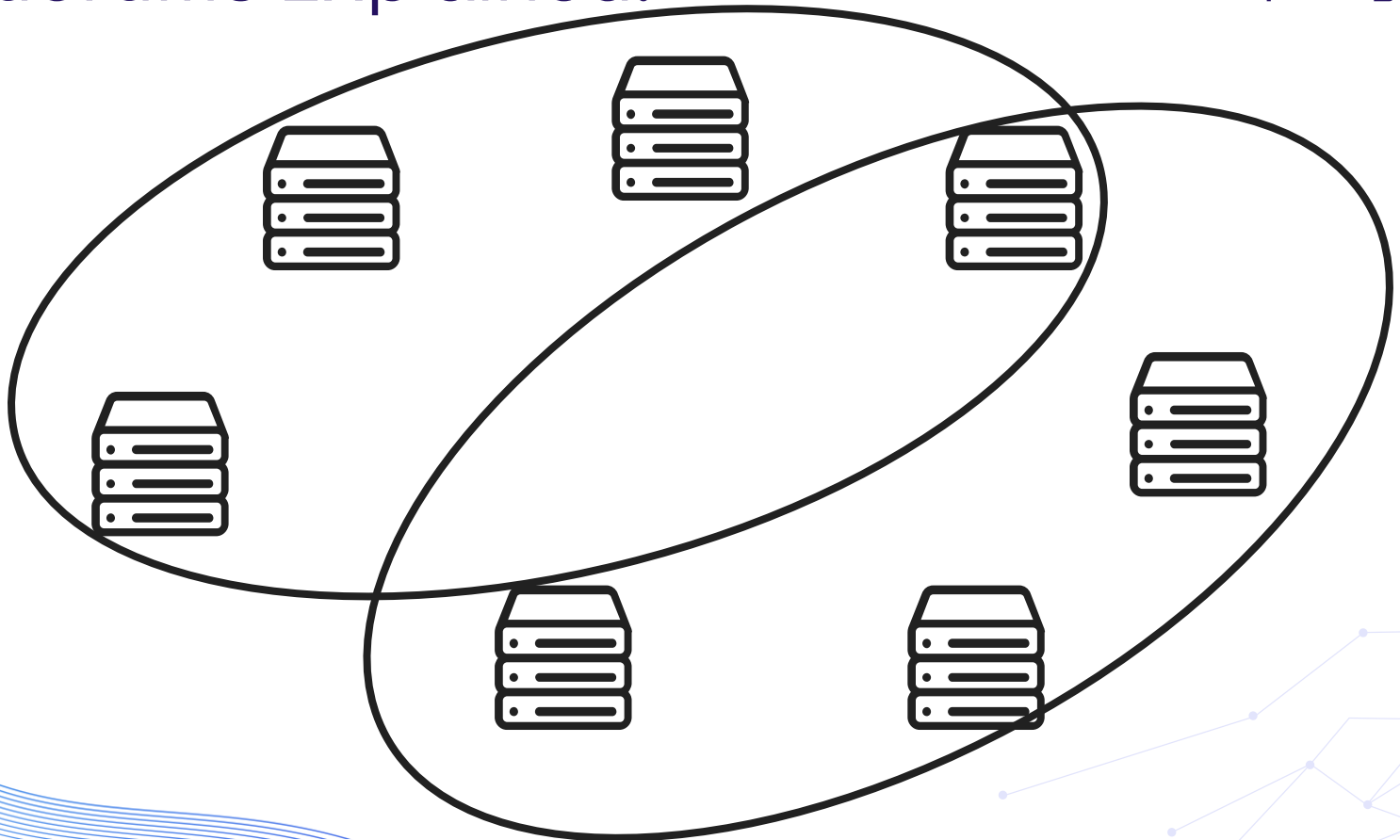
Quorums Explained: Quorum Intersection Property



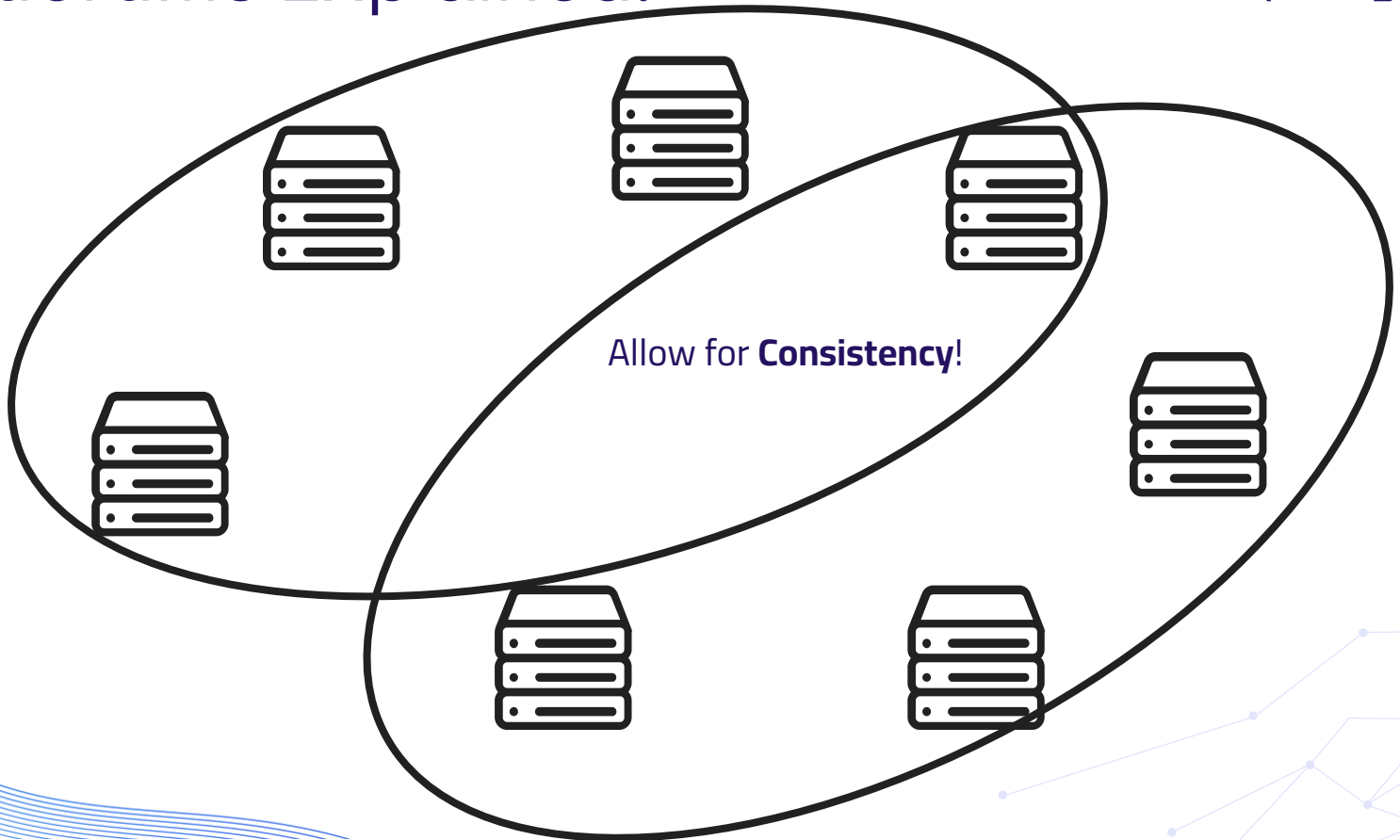
Quorums Explained: Quorum Intersection Property



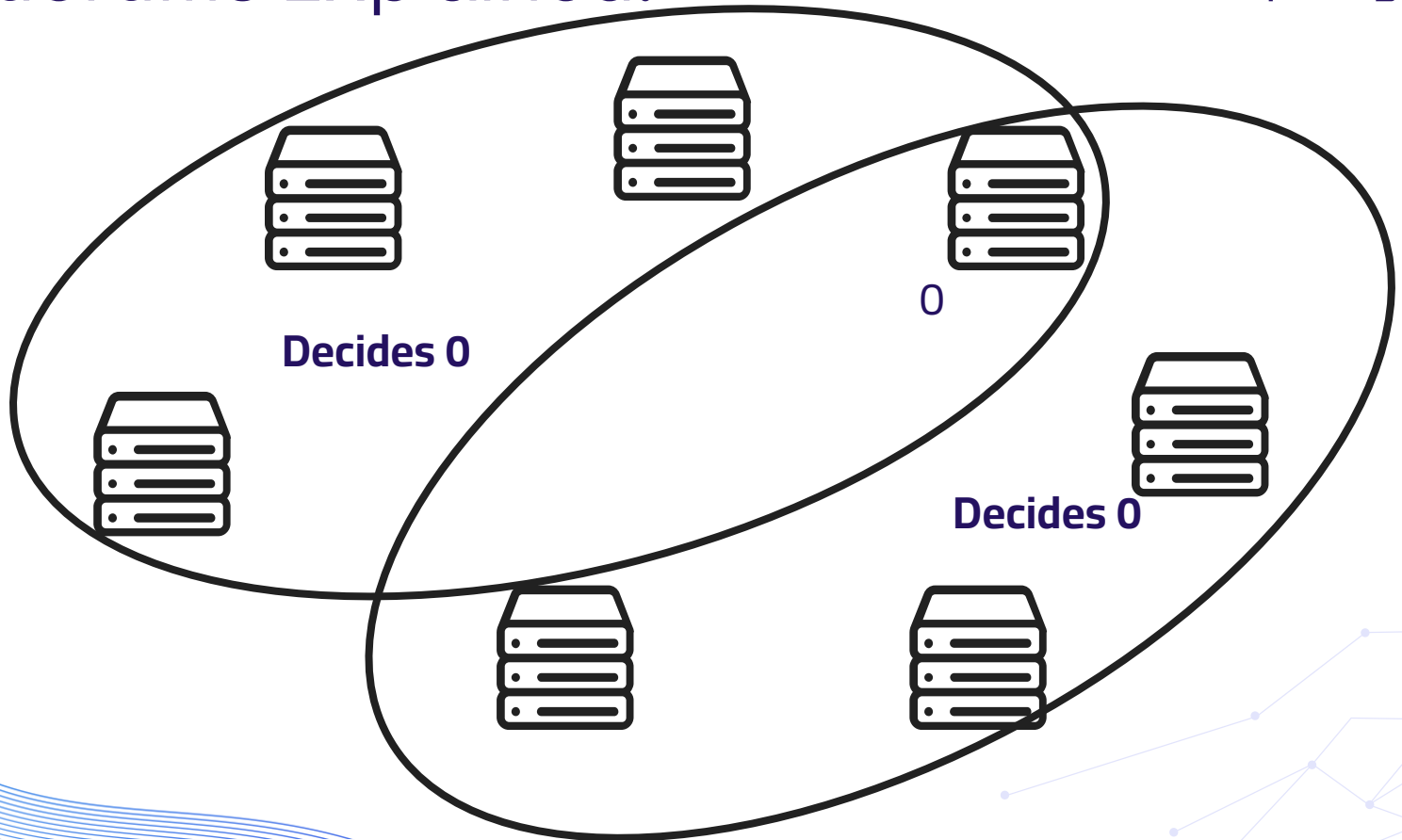
Quorums Explained: Quorum Intersection Property



Quorums Explained: Quorum Intersection Property



Quorums Explained: Quorum Intersection Property



Quorums Explained: Quorum Intersection Property

Why do we care if this property holds for a consensus protocol that accounts for Byzantine nodes?

Quorums Explained: Quorum Intersection Property

Why do we care if this property holds for a consensus protocol that accounts for Byzantine nodes?

Best Case: (one overlapping node)

$$|Q_1 \cap Q_2| \geq 1$$

Worst case: (Maximum faults overlapping)

$$|Q_1 \cap Q_2| \geq 2q - n$$

Quorums Explained: Quorum Intersection Property

Why do we care if this property holds for a consensus protocol that accounts for Byzantine nodes?

Best Case: (one overlapping node)

$$|Q_1 \cap Q_2| \geq 1$$

We want to solve for q!

Worst case: (Maximum faults overlapping)

$$|Q_1 \cap Q_2| \geq 2q - n$$

Quorums Explained: Quorum Intersection Property

Worst case: (Maximum faults overlapping)

$$2q - n$$

Quorums Explained: Quorum Intersection Property

Worst case: (Maximum faults overlapping)

$$2q - n$$

In order for 1 correct overlapping node:

$$2q - n > f$$

Quorums Explained: Quorum Intersection Property

Inequality we're focusing on:

$$2q - n > f$$

Rearrange:

$$2q > n + f$$

Substitute known Byzantine Fault Node Requirement for n (3f + 1):

$$2q > 3f + 1 + f$$

Simplify for:

$$q \geq 2f + 1$$

Quorums Explained

Network Failure

Maximum Network Failure Nodes (*f*):
(n-1)/2

Necessary Correct Nodes:
f+1

Byzantine Nodes

Maximum Byzantine Failure Nodes (*f*):
(n-1)/3

Necessary Correct Nodes:
2*f*+1

05

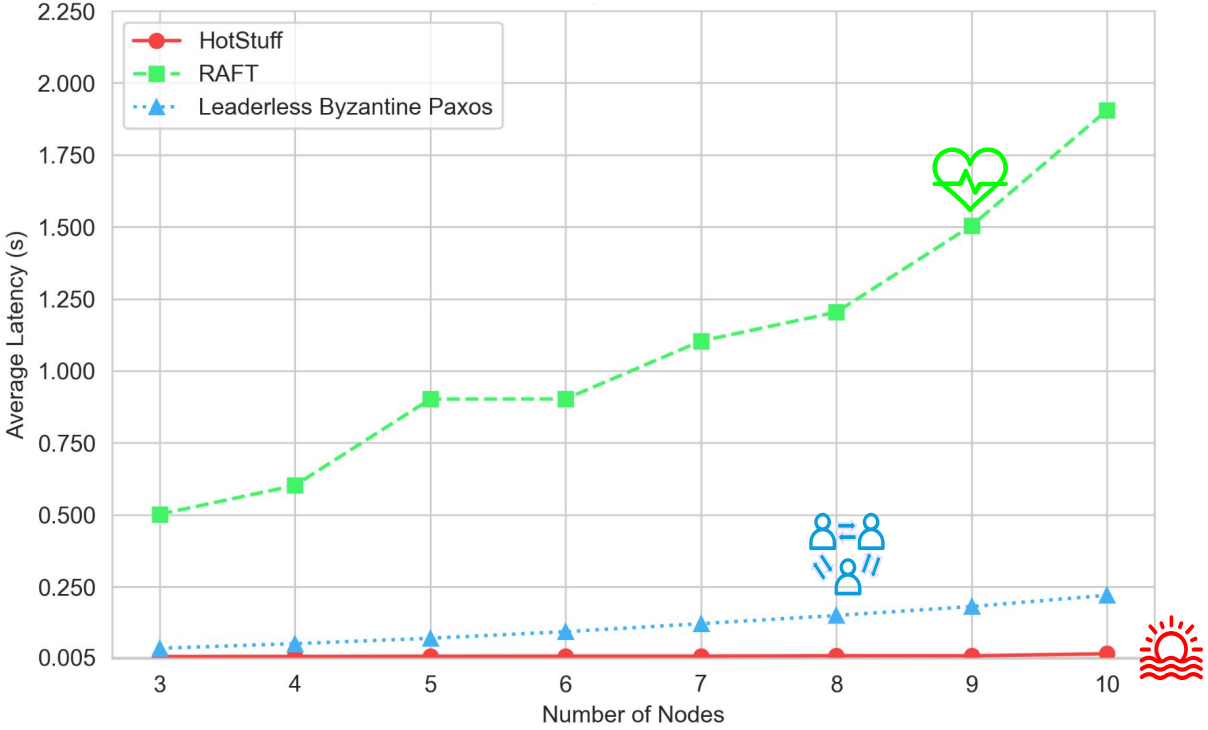
Data Analysis

Latency

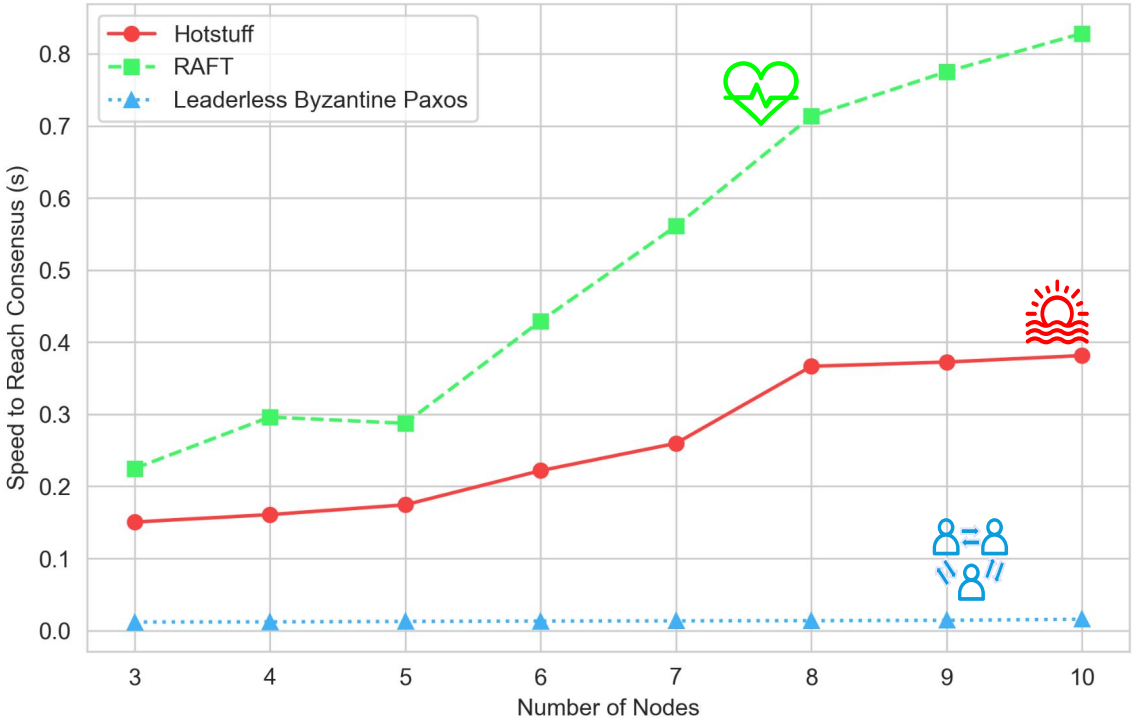
Int **MPI_Send** (void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Status *status)

Int **MPI_Recv** (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

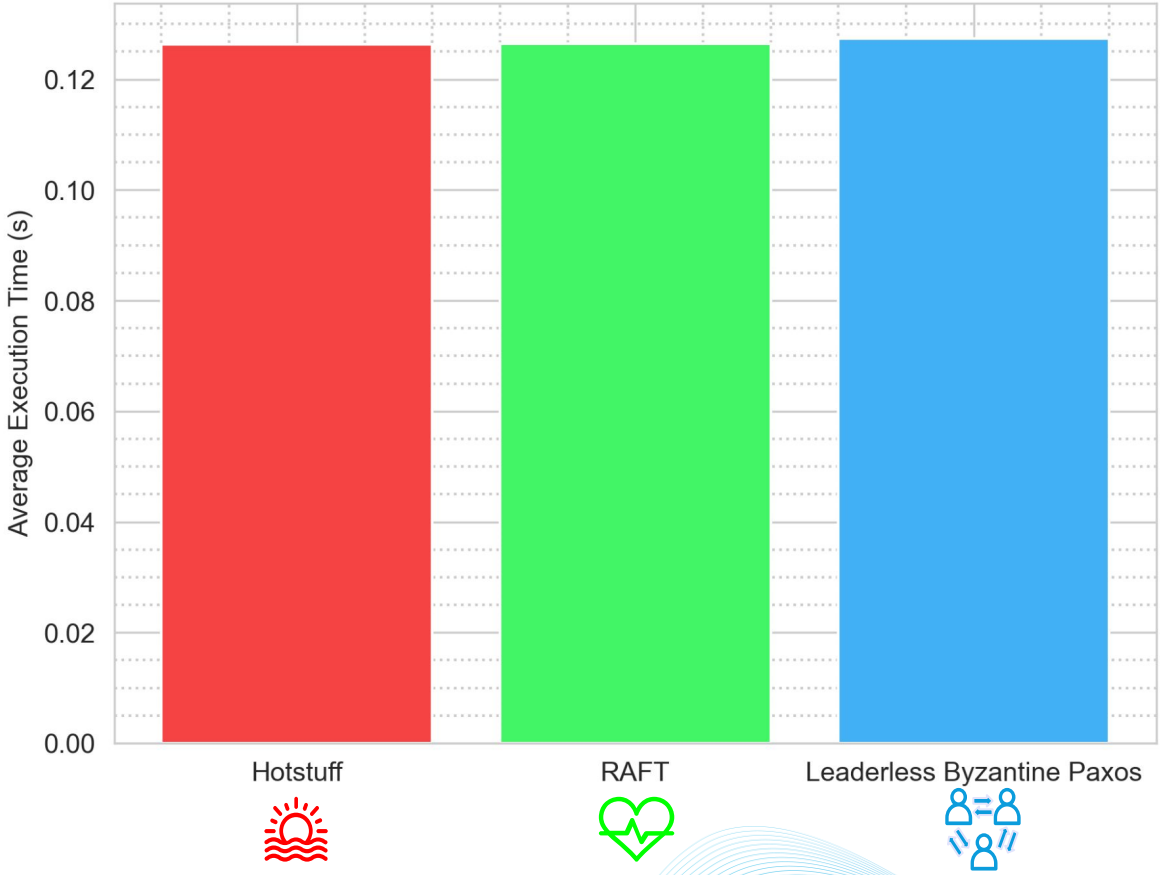
Latency Comparison



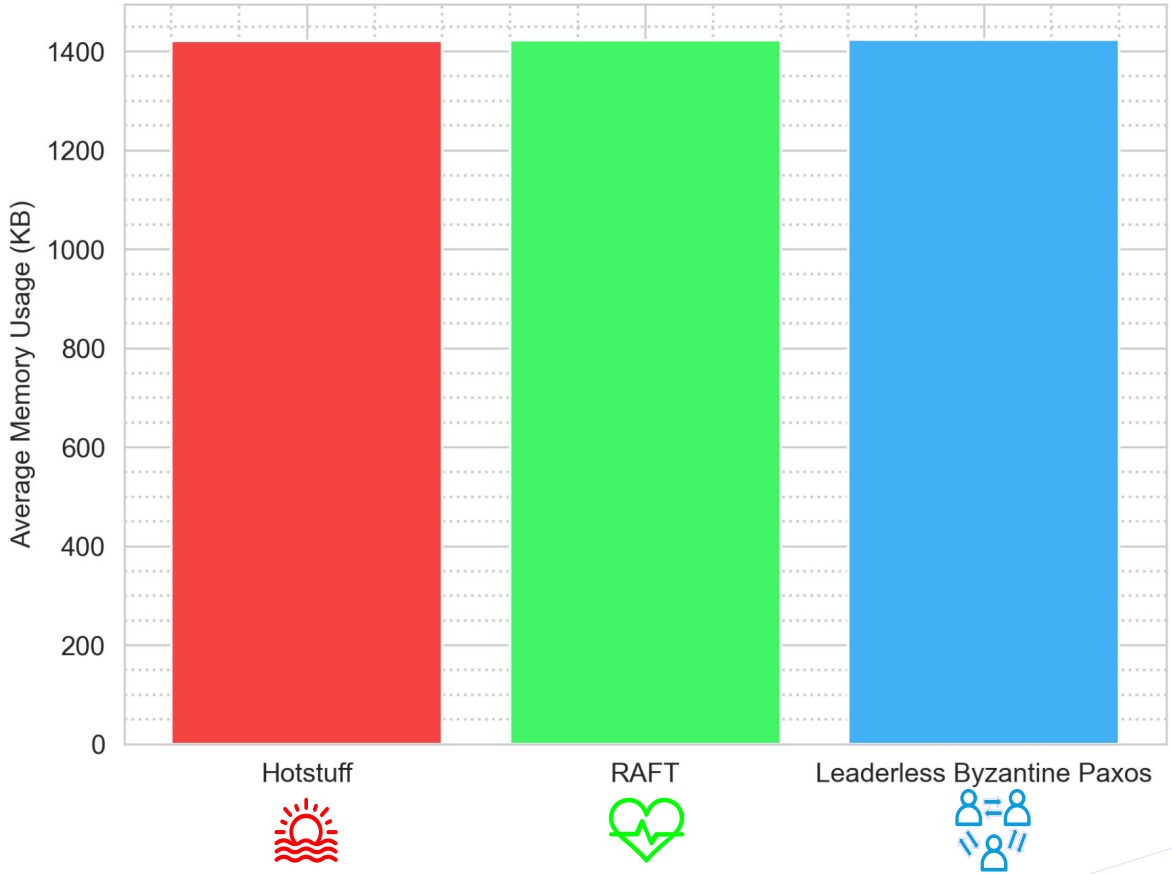
Consensus Speed Comparison



Execution Time Comparison



Memory Usage Comparison



06

Conclusion

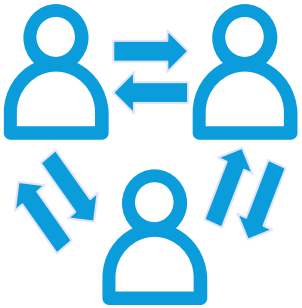
Summary: Why Do We Care?

- Consensus is important in your day to day life!
- Banking, social media apps, cryptocurrency proof of work, cloud computing, and anything that uses servers

Summary: What We Did

- We implemented Leaderless Byzantine Paxos, Raft, and Hotstuff
  - We used the raspberry pi's as our distributive system





Summary: What We Did

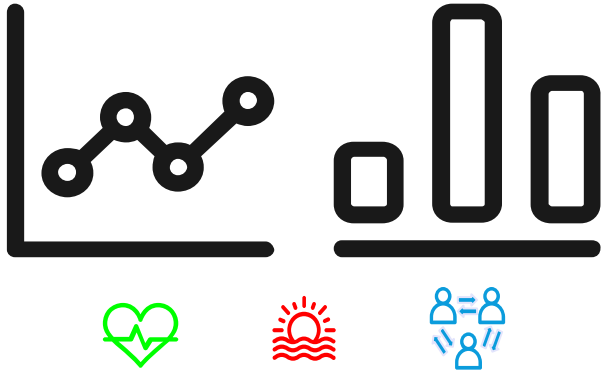
- Implemented a logging system for each node
  - Many real world uses of consensus algorithms use logs to debug and when something went wrong

Node 1

- Term 1: Acknowledged new term
- Term 1: Received heartbeat
- Term 1: Failed to receive heartbeat

Summary: What We Did

- Compared and the latency, execution time, and memory usage of our consensus algorithms and we proved some stuff



Acknowledgements

We'd like to thank Tanya Amert, our wonderful comps advisor. We'd also like to thank all of the Computer Science Professors and the Computer Science Department (especially Mike Tie).

We'd like to thank all of our friends and family for getting us here. And most of all we'd like to thank you all for coming here to listen, hopefully without zoning out.

Sources

- G. Bracha and S. Toueg, “Asynchronous consensus and broadcast protocols,” *Journal of the ACM (JACM)*, vol. 32, no. 4, pp. 824–840, Oct. 1985, doi: https://doi.org/10.1145/4221.214134.
- Ongaro, Diego; Ousterhout, John (2013). "In Search of an Understandable Consensus Algorithm"
- Lamport, Leslie. “Leaderless Byzantine Paxos.” (2016).
- Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness.
- Gabriel Bracha and Sam Toueg. 1985. Asynchronous consensus and broadcast protocols. J. ACM 32, 4 (Oct. 1985), 824–840. https://doi.org/10.1145/4221.214134
- Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. 1985. Impossibility of distributed consensus with one faulty process. J. ACM 32, 2 (April 1985), 374–382. https://doi.org/10.1145/3149.214121
- Yin, Maofan et al. “HotStuff: BFT Consensus in the Lens of Blockchain.” arXiv: Distributed, Parallel, and Cluster Computing (2018): n. pag.

Summary: What We Did

- We implemented Leaderless Byzantine Paxos, Raft, and Hotstuff
  - We used the raspberry pi's as our distributive system