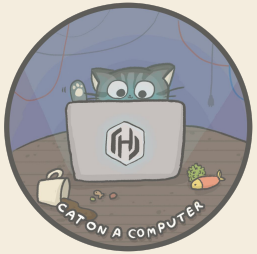
You. Must. Build. A. Raft!

A tidal wave of information


Background...
Layout
Theme...
Transition...
I'm Transgender
@krisnova
Click to add speaker notes
Explore
MacBook Pro

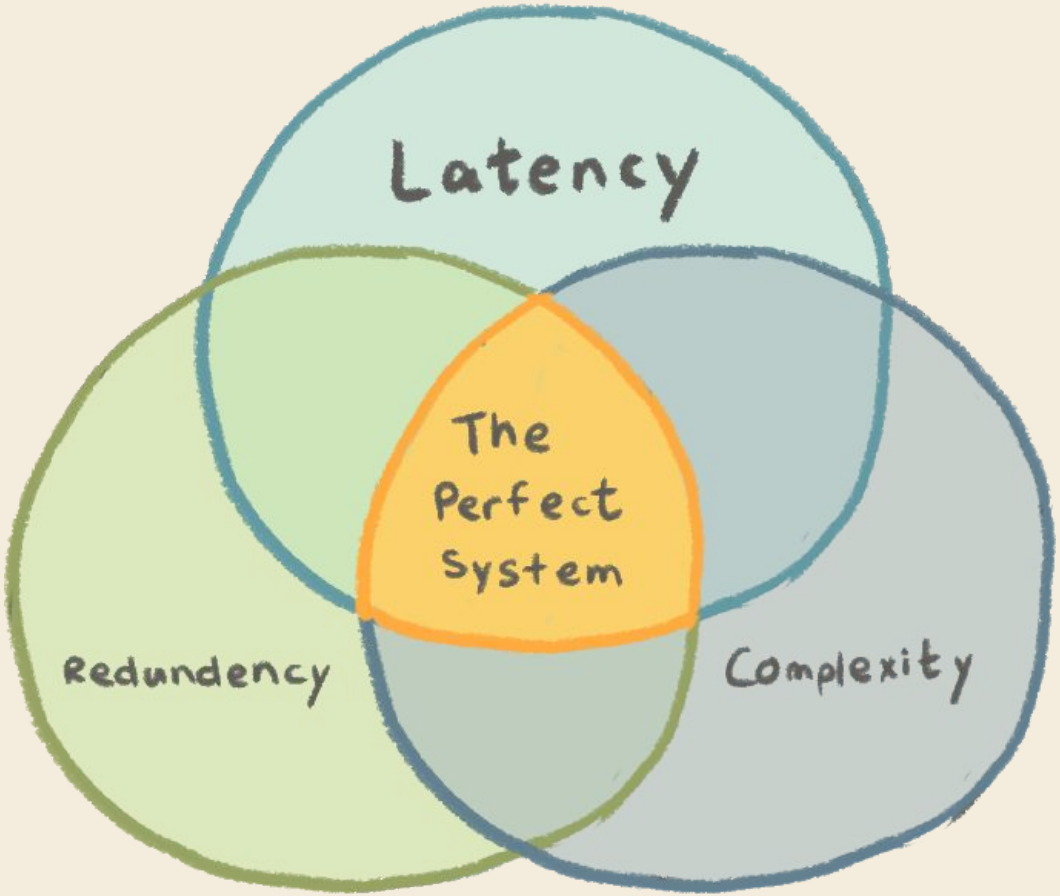
@schristoff25



catonacomputer.com

CLOUDREACH


You Must
Build A Boat
Available on the App Store
STEAM
GET IT ON Google play


Latency
The Perfect System
Redundency
Complexity

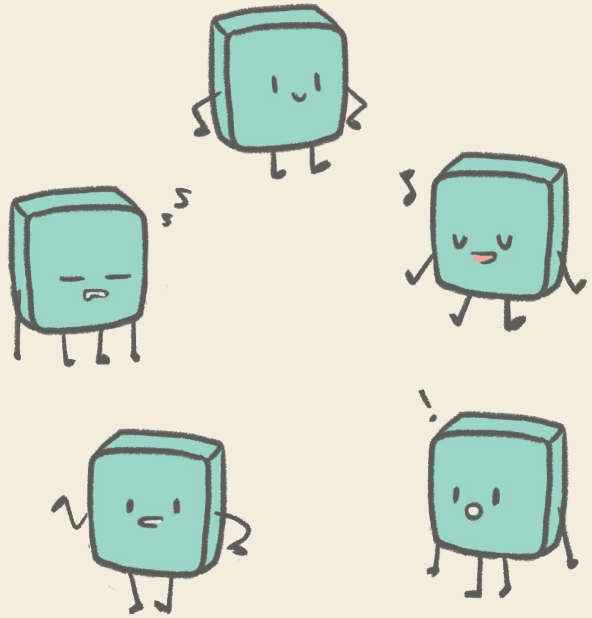
Lead

Me

Lesson Plan:

- Distributed Systems vs Centralized Systems
- Byzantine General’s Problem
- Byzantine Fault Tolerance
- Consensus
  - Paxos
  - Raft

Centralized



Distributed

Byzantine Generals Problem

Fig. 1: A City

Fig. 1: A City

Fig. 2: A Byzantine General
(obviously)

(oh boy.)

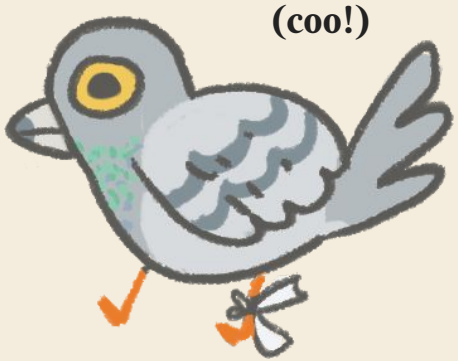
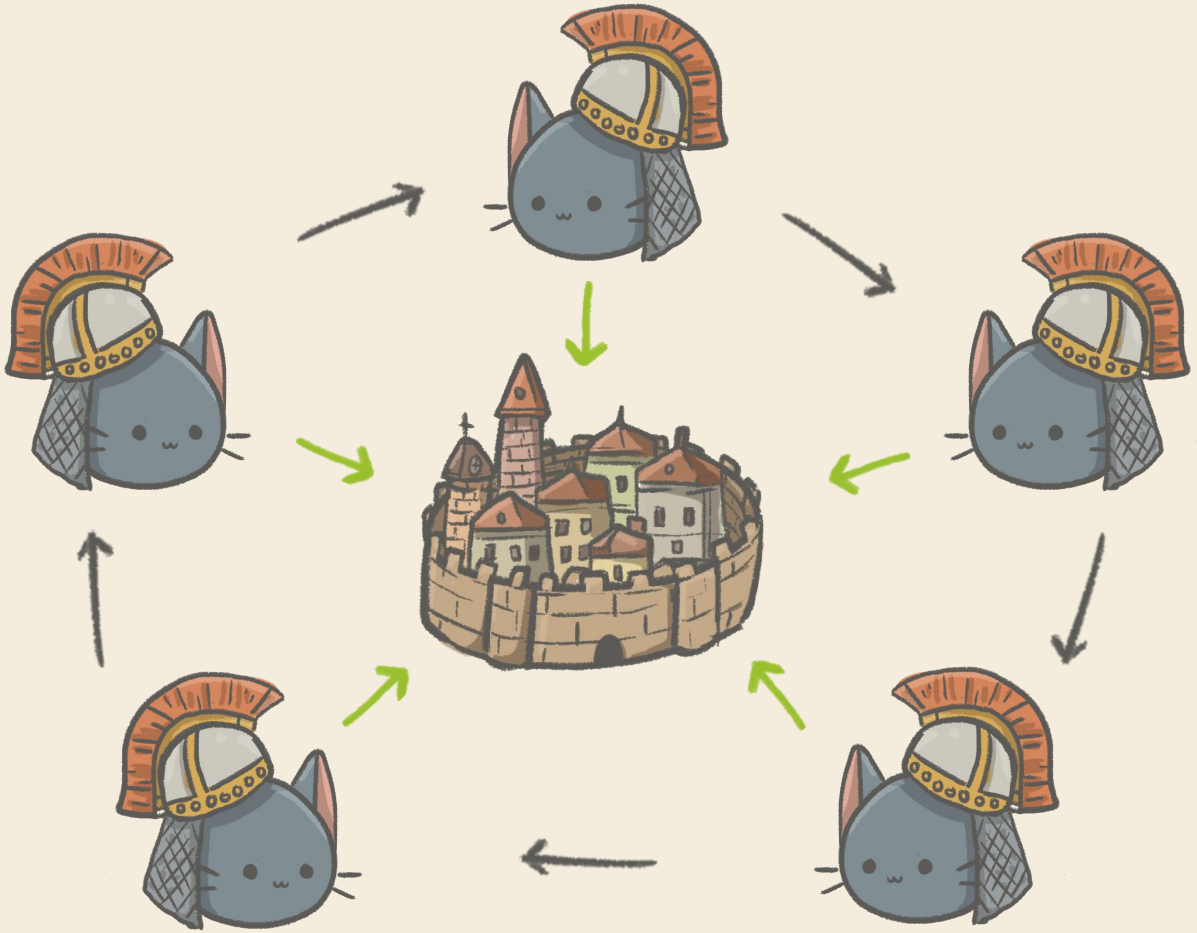


Fig. 3: A Messenger



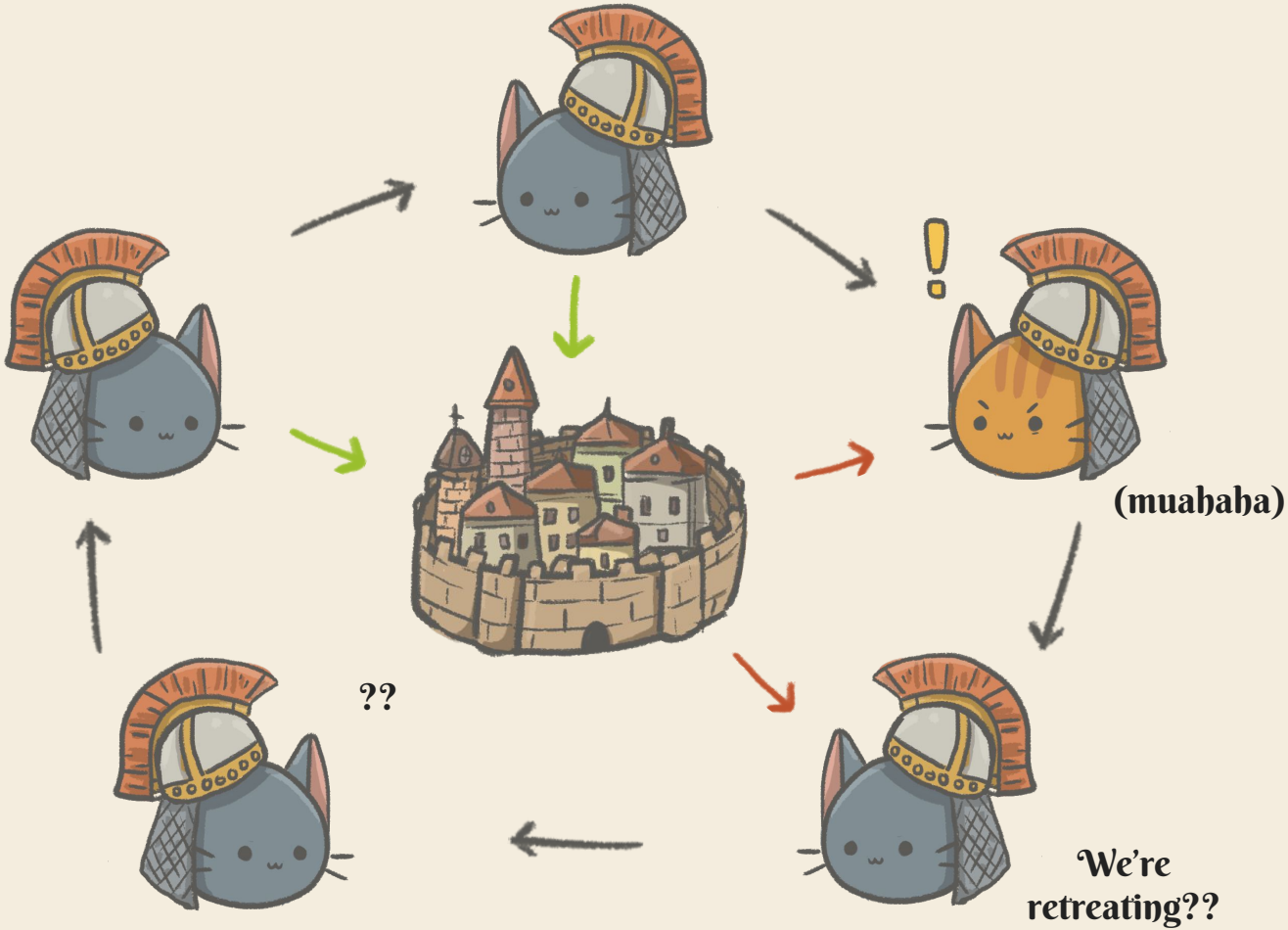

!
(muahaha)
We're retreating??
??

Fig. 4: The Traitor

Solution

Algorithm OM(0).

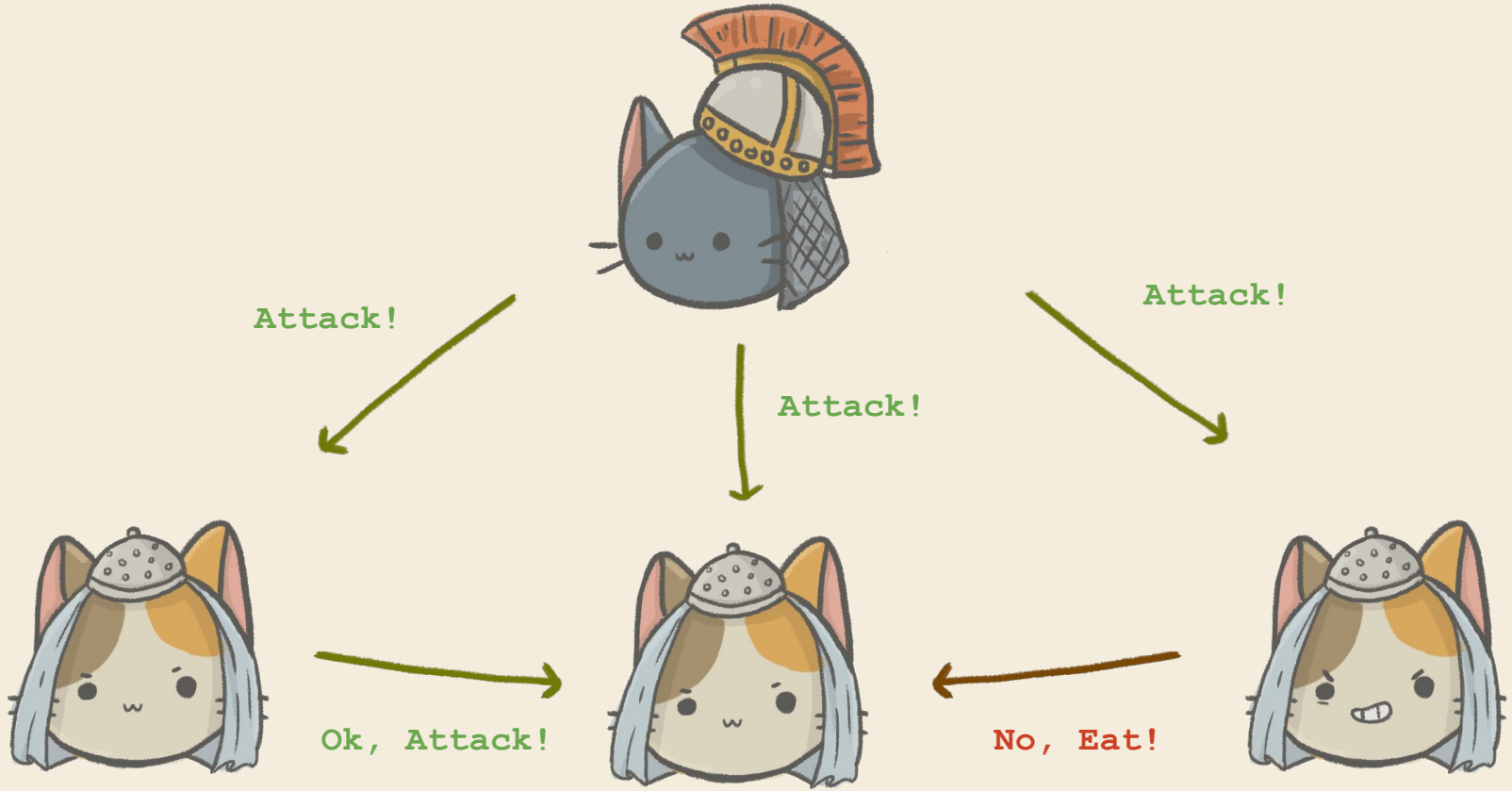
(1) The commander sends his value to every lieutenant.
(2) Each lieutenant uses the value he receives from the commander, or uses the value RETREAT if he receives no value.

Algorithm OM(m), $m > 0$.

(1) The commander sends his value to every lieutenant.
(2) For each i, let v_i be the value Lieutenant i receives from the commander, or else be RETREAT if he receives no value. Lieutenant i acts as the commander in Algorithm OM($m - 1$) to send the value v_i to each of the $n - 2$ other lieutenants.
(3) For each i, and each $j \neq i$, let v_j be the value Lieutenant i received from Lieutenant j in step (2) (using Algorithm OM($m - 1$)), or else RETREAT if he received no such value. Lieutenant i uses the value *majority*($v_1, \ldots, v_{n-1}$).

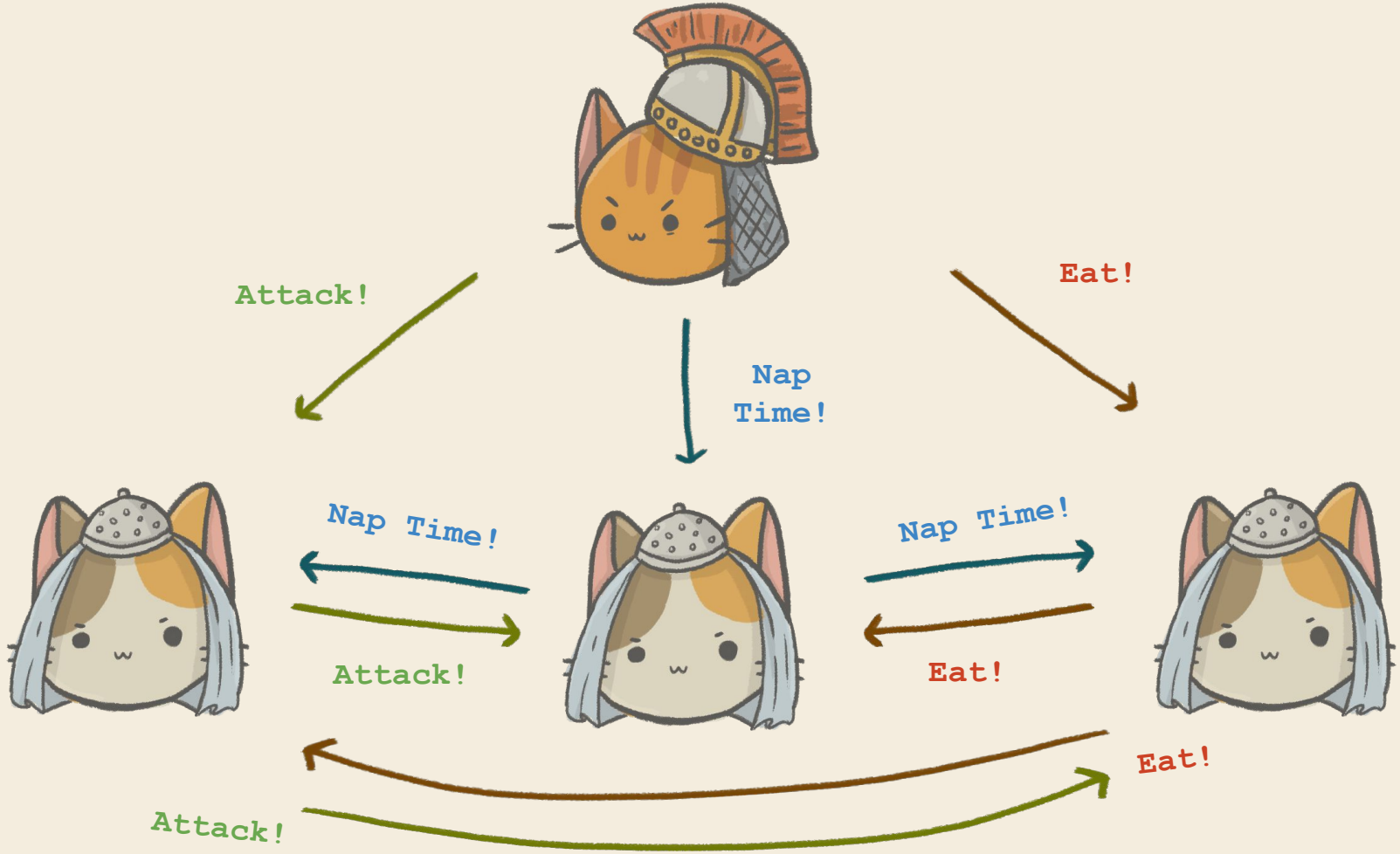
Fig. 5: A Lieutenant

The generals can reach consensus
if *2/3* of them are honest.


Attack!
Attack!
Attack!
Ok, Attack!
No, Eat!

So.. 2 commands for **Attack** and 1 for **Eat**.

I'll **ATTACK!**


Attack!
Nap Time!
Eat!
Nap Time!
Attack!
Nap Time!
Eat!
Attack!
Eat!

Lieutenant 1

Attack, Nap, Eat

Lieutenant 2

Attack, Nap, Eat

Lieutenant 3

Attack, Nap, Eat

So.. Attack, Nap, and Eat.

We'll **RETREAT!** (as a default action)

Byzantine Fault Tolerance

“In a **"Byzantine failure"**, a component such as a server can inconsistently appear both failed and functioning to failure-detection systems, presenting different symptoms to different observers.”

Byzantine Fault

Any fault that presents different symptoms to different observes

Byzantine Failure

The loss of system agreement due to a Byzantine Fault

Actual Byzantine Fault

First Picture of a Byzantine Fault?

Honeywell

At 12:12 GMT 13 May 2008, a NASA Space Shuttle was loading hypergolic fuel for mission STS-124 when a 3-1 split of its four control computers occurred. Three seconds later, the split became 2-1-1. During troubleshooting, the remaining two computers disagreed (1-1-1-1 split). ***Complete system disagreement.*** But, none of the computers or their intercommunications were faulty! The ***single fault**** was in a box (MDM FA2) that sends messages to the 4 computers via a multi-drop data bus that is similar to the MIL STD 1553 data bus. This fault was a simple crack (fissure) through a diode in the data link interface.

Figure 1. Two views (90 degrees apart) of a fissure that appears to go through the silicon - Red arrows.

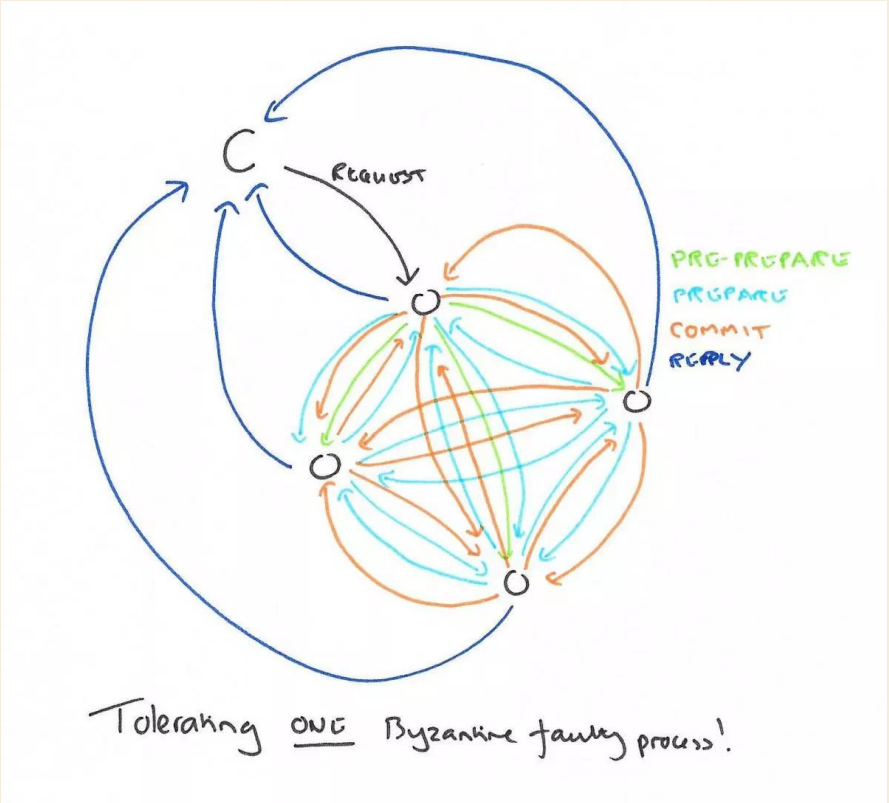
* the Byzantine Assassin

Practical Byzantine Fault Tolerance

History

- Written by Castro and Liskov in 1999

- Tests show PBFT is only 3% slower than the standard NFS daemon

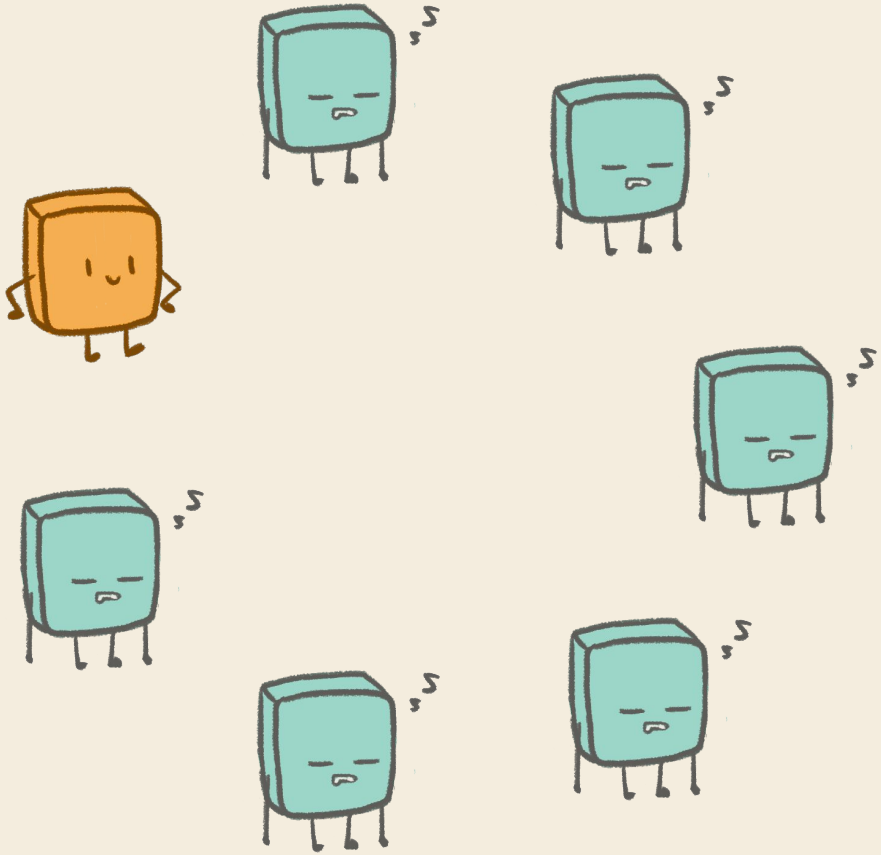


Breakdown

PBFT uses $3f+1$, f for failures, which requires more replicas than non-Byzantine consensus modules.

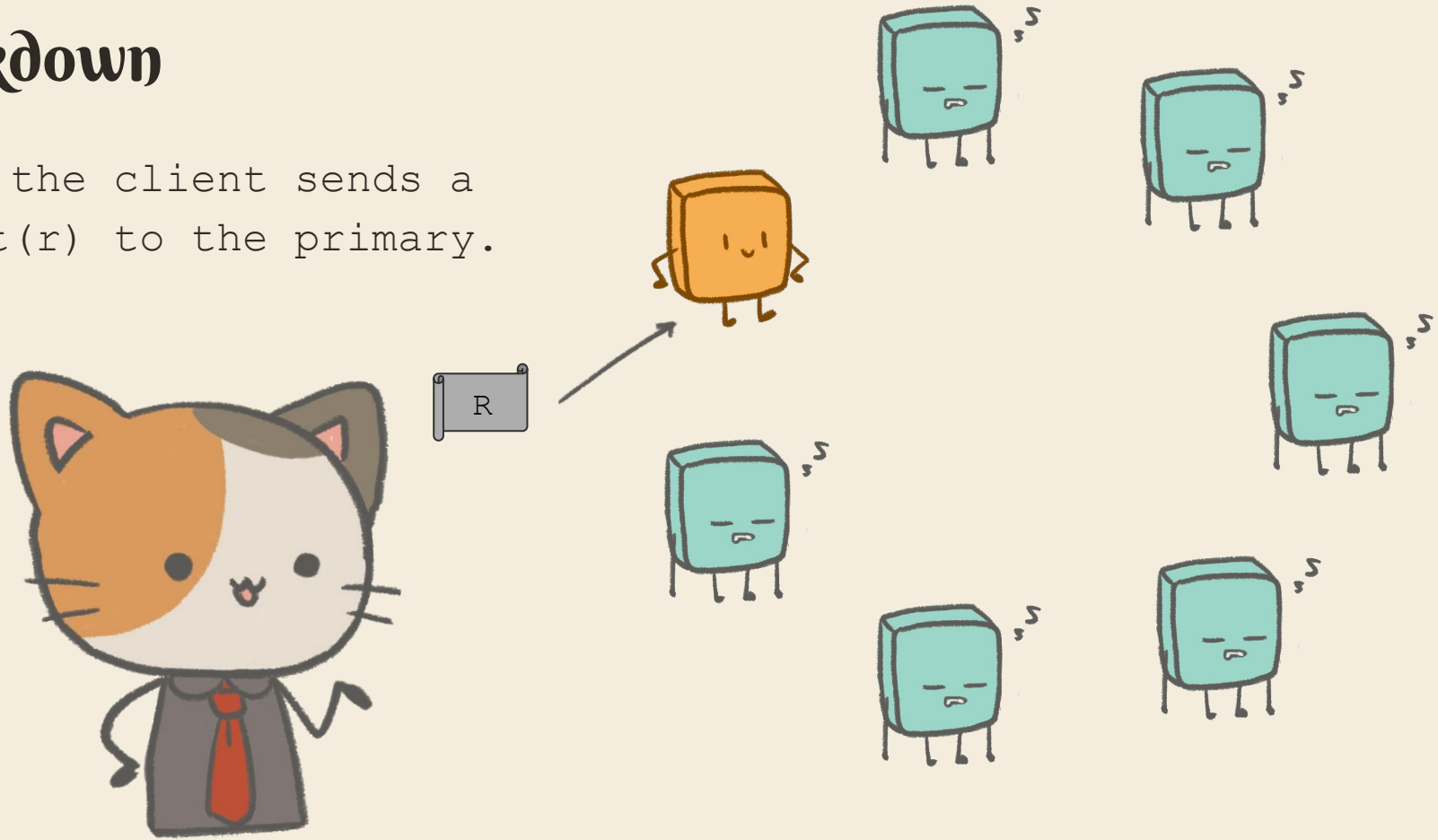
Breakdown

- We have a system that we want to withstand two failures.
  - 3 * 2 + 1 = 7



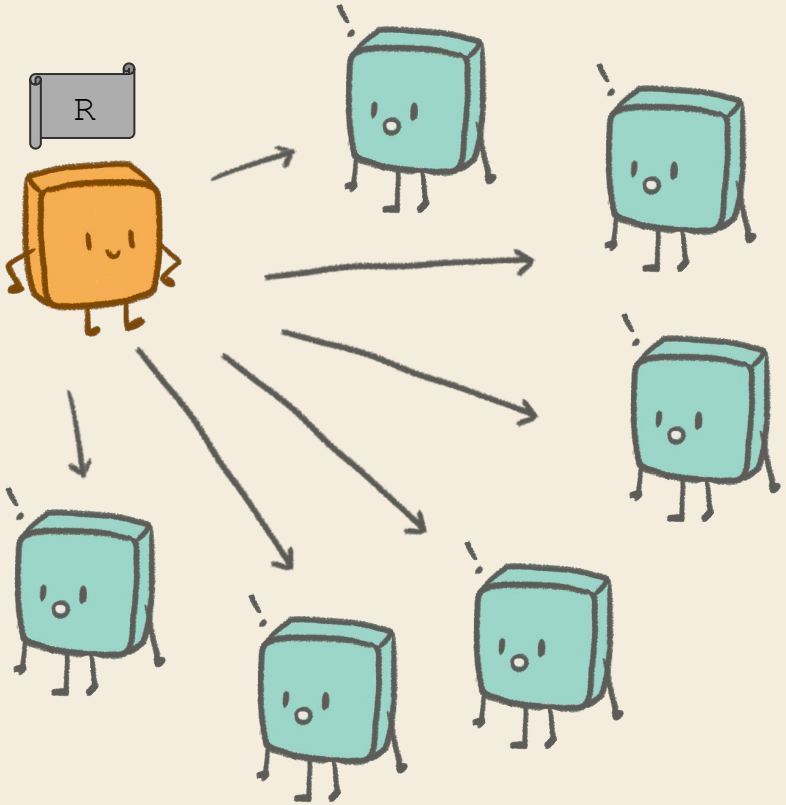
Breakdown

First, the client sends a request(r) to the primary.



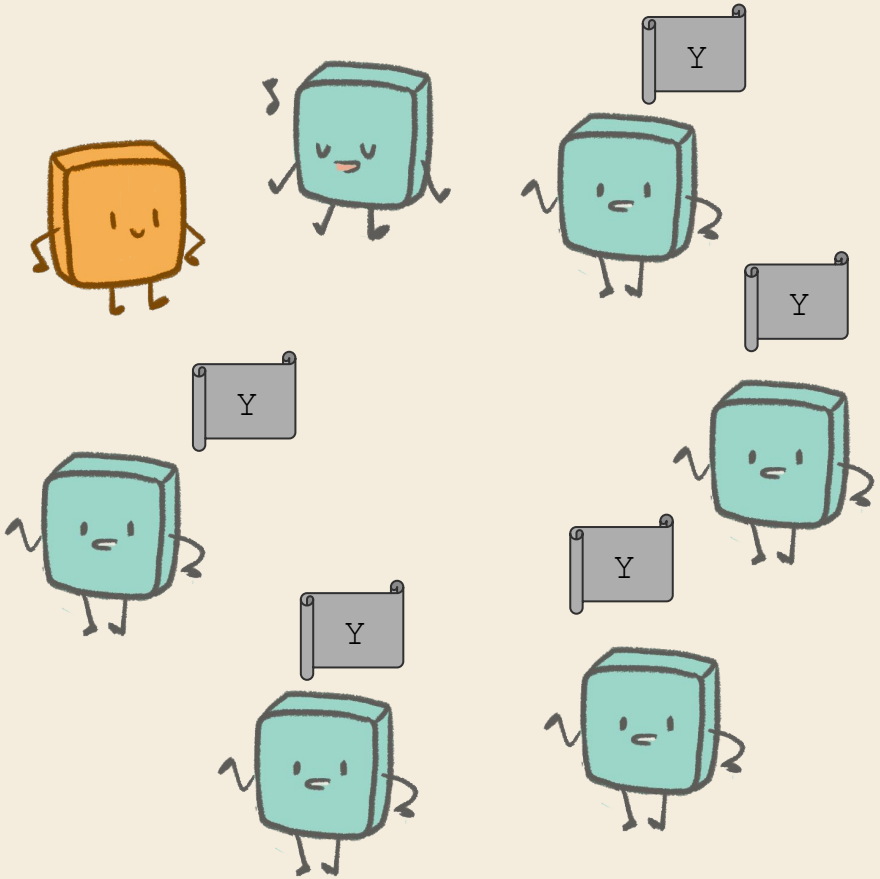
Breakdown

The primary then sends the request to all backups.



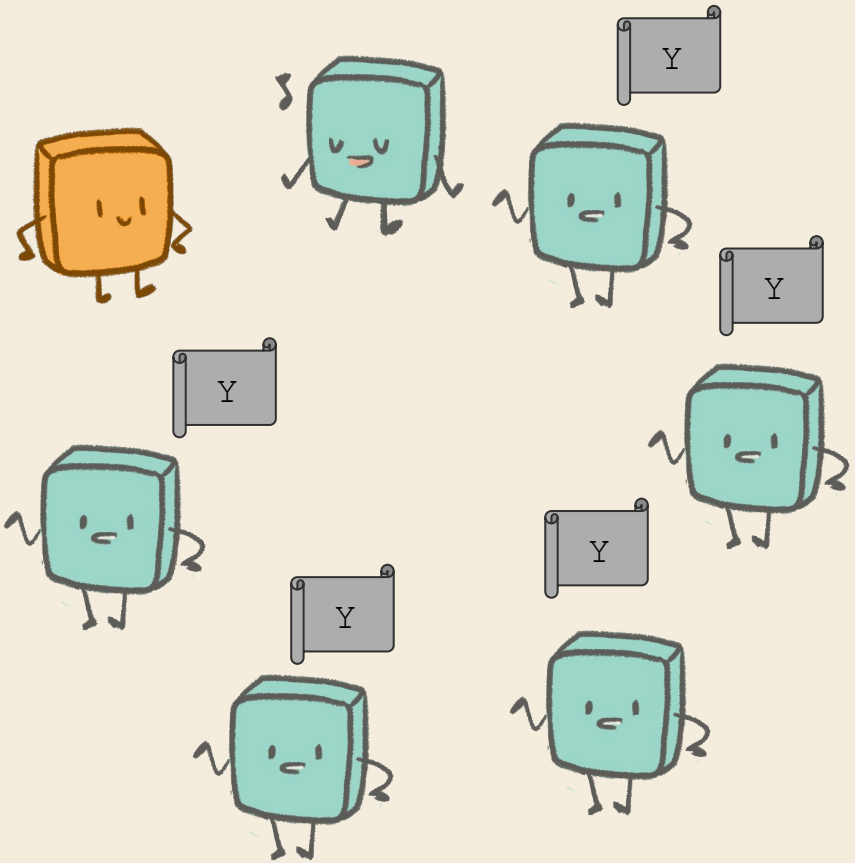
Breakdown

Each backup executes the request and then replies to the client



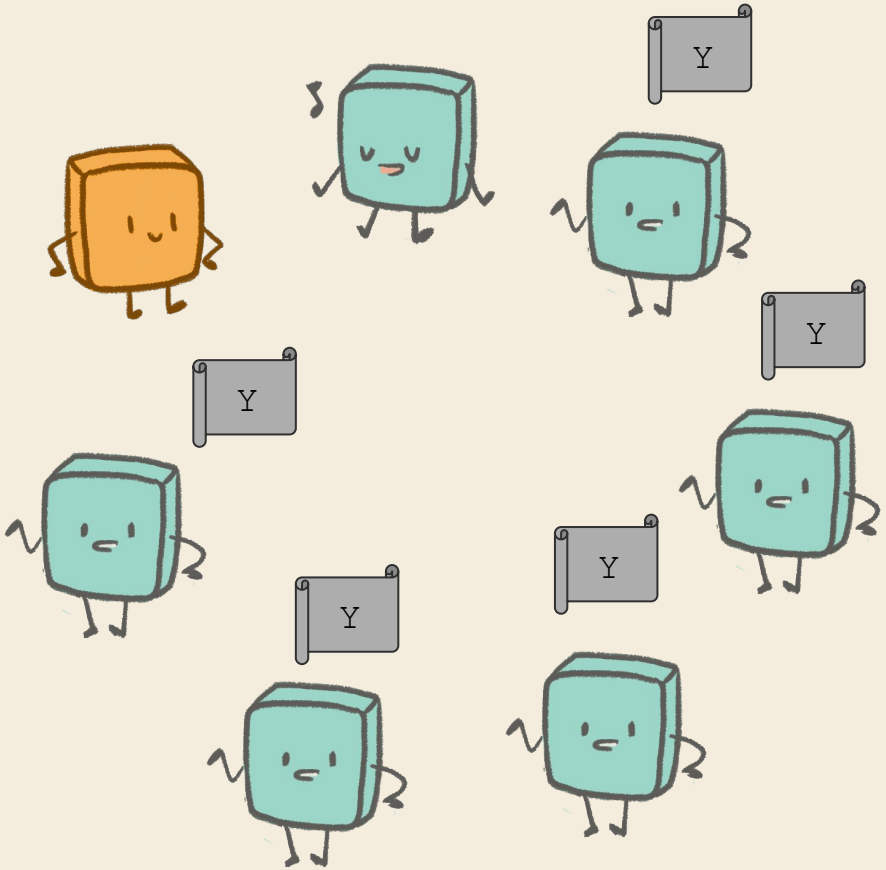
Breakdown

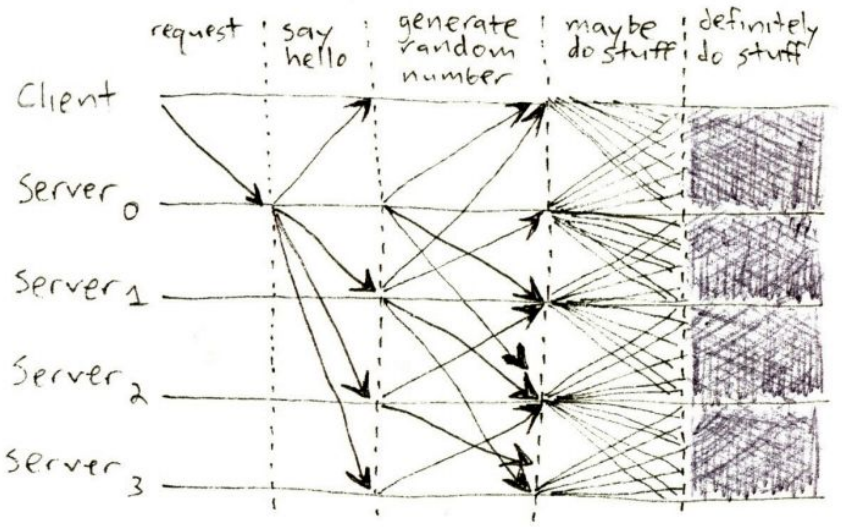
Client waits for *f*+1 replies from **different** replicas with the same result.

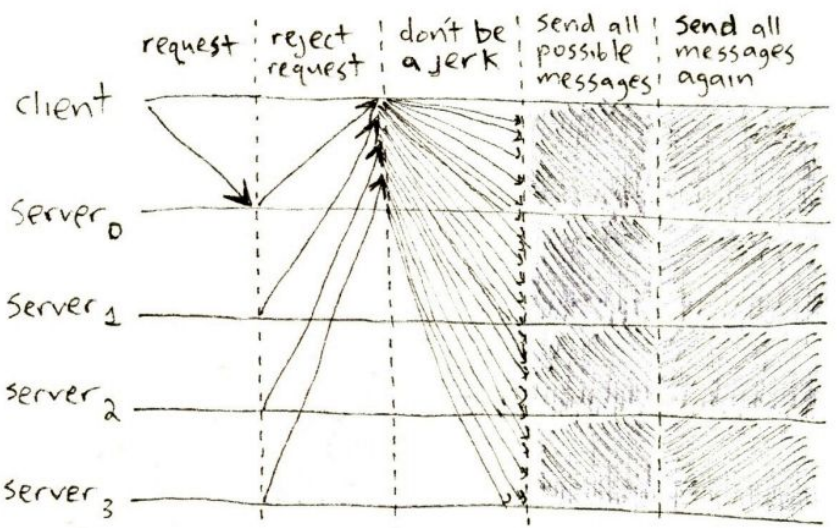


Breakdown

If the client doesn’t receive replies soon enough, it will send the request to all replicas.





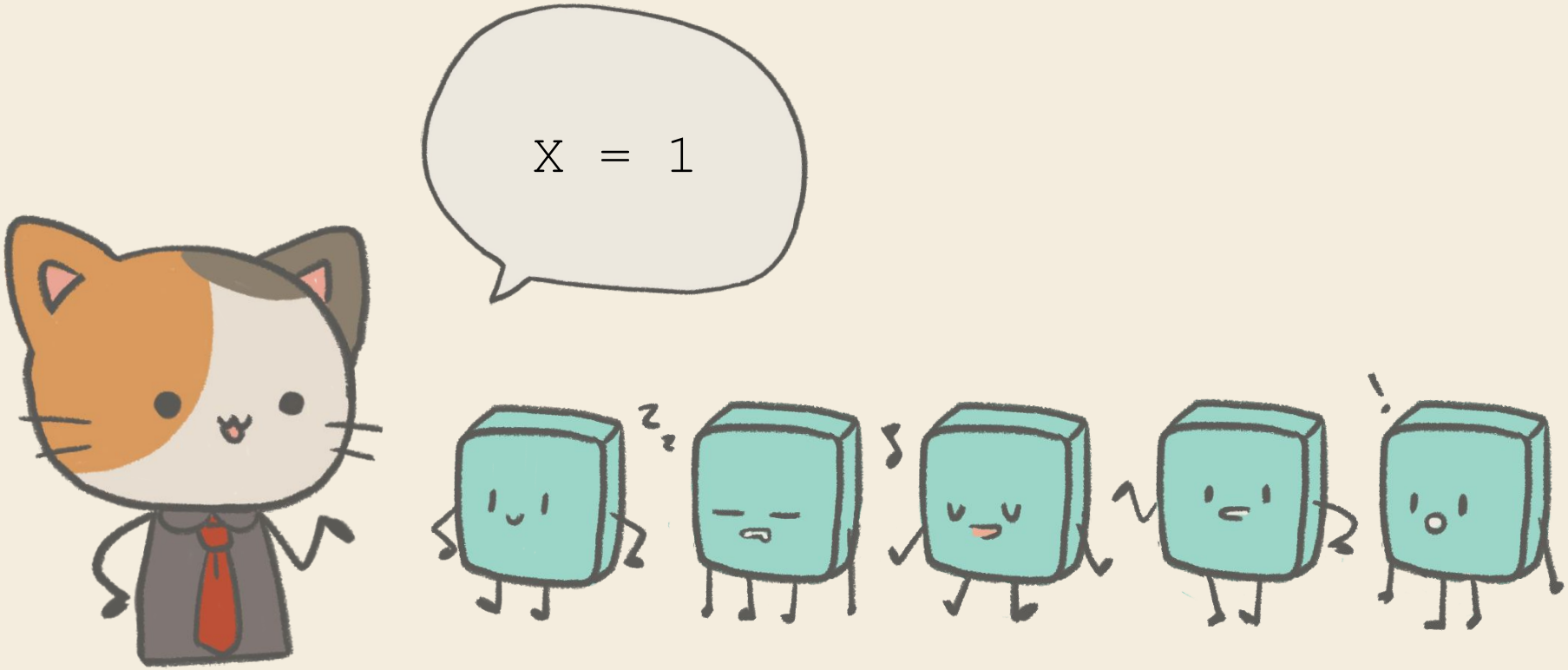


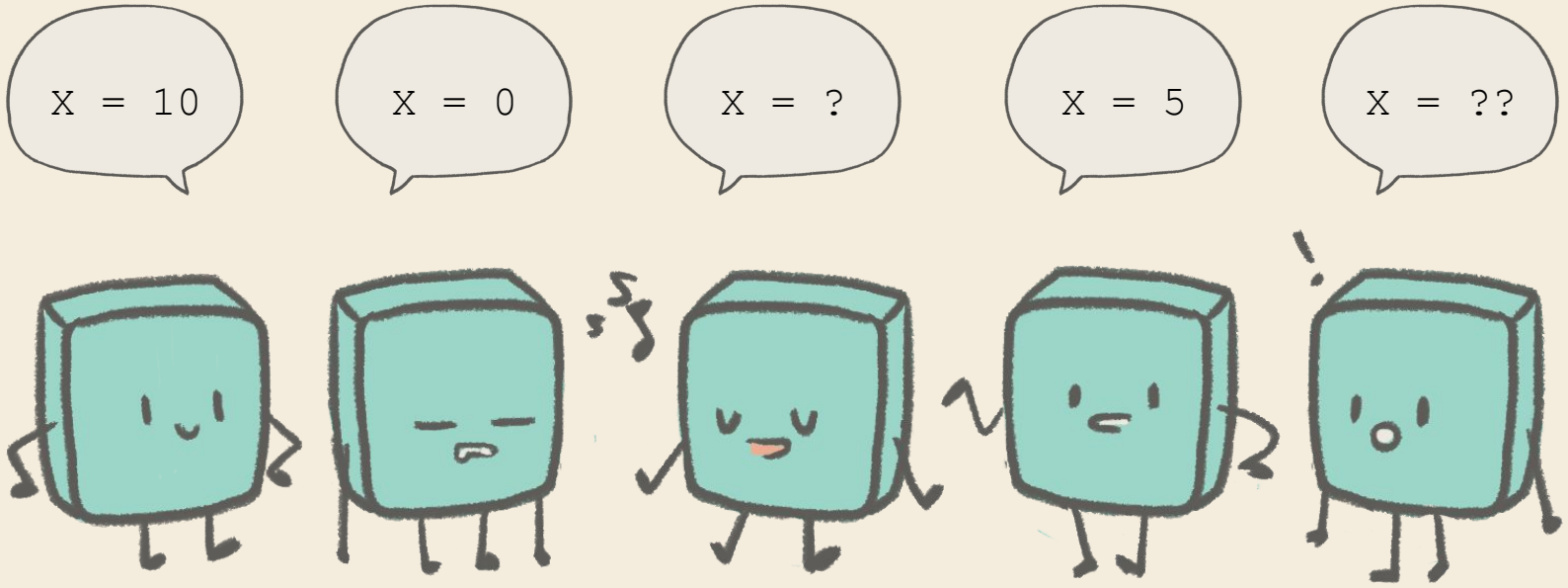
(The Saddest Moment)

“*How can you make a reliable computer service?*” the presenter will ask in an innocent voice before continuing, “*It may be difficult if you can’t trust anything and the entire concept of happiness is a lie designed by unseen overlords of endless deceptive power.*”

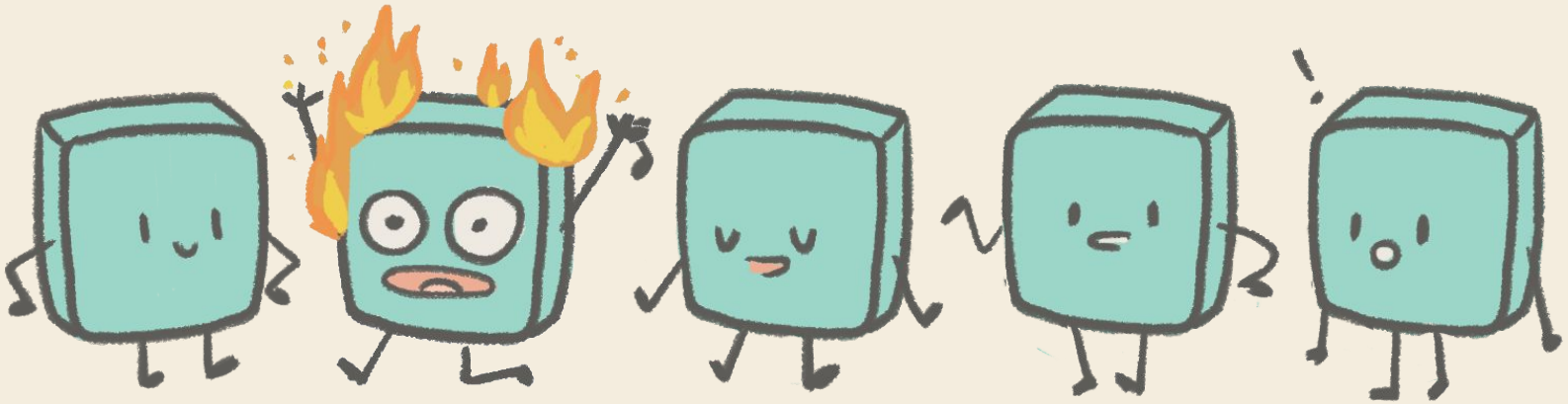
- James Mickens

Consensus Problem


X = 1


X = 10
X = 0
X = ?
X = 5
X = ??

? ?



A consensus protocol must..

Termination

Every correct process decides some value

Integrity

If all the correct processes proposed the same value x, then any correct process decide x

Validity

If a process decides a value x, then x must have been proposed by some correct processes.

Agreement

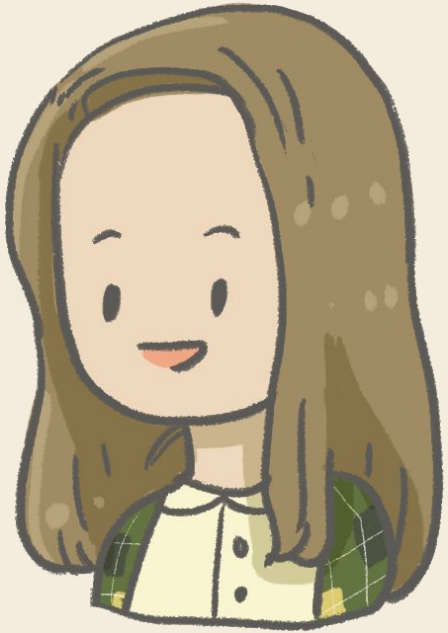
Every correct process must agree on the same value

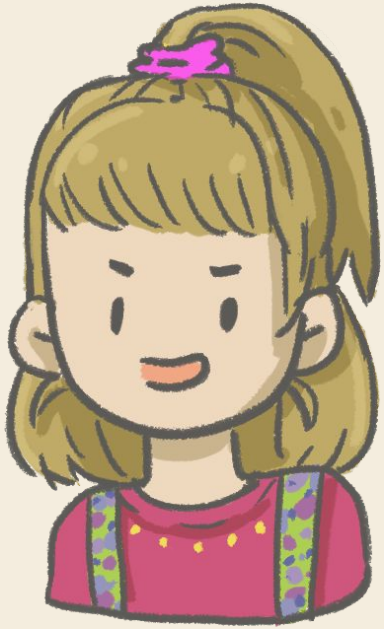
Paxos

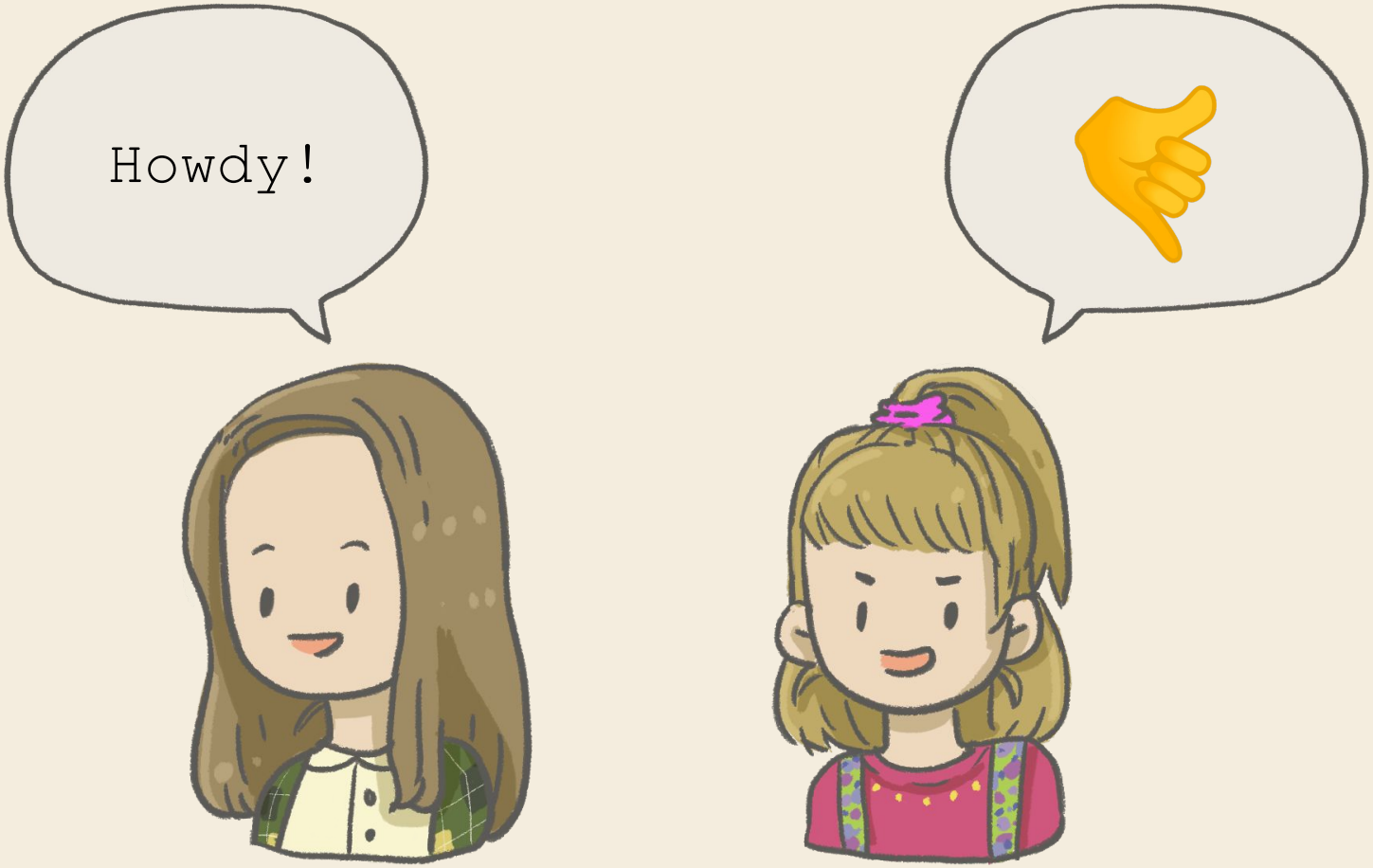
History

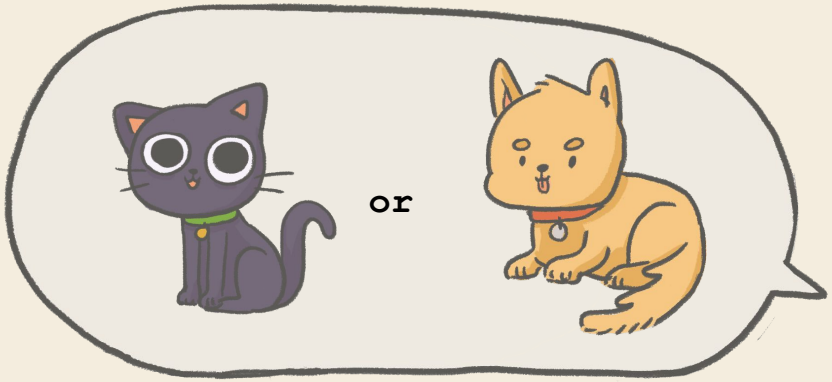
- Used Lynch and Liskov’s work as a base, 1988
- The Part Time Parliament, Lamport 1989-1998
  - No one understood it so it took ten years to get it published
- Paxos Made Simple, Lamport 2001
  - Also not easy to understand..

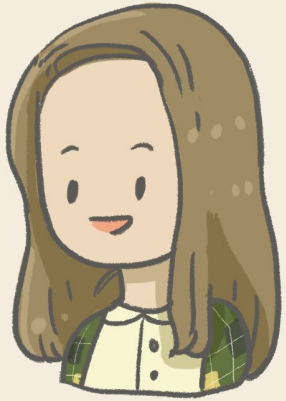
Breakdown

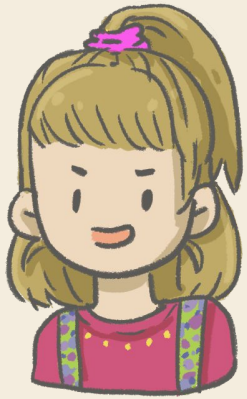





Howdy!



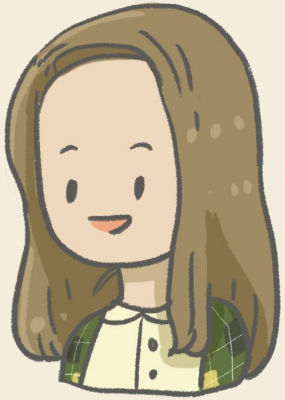




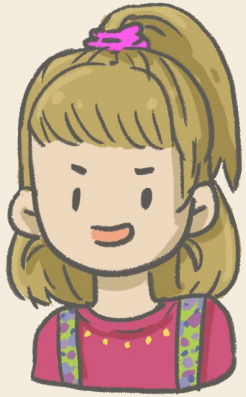
¯_(ツ)_/¯

¯_(ツ)_/¯

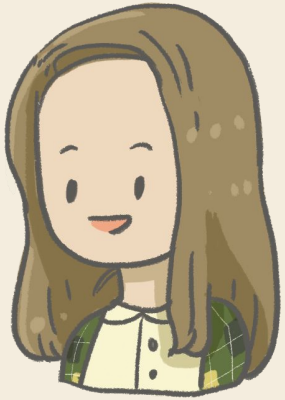
How about this majestic amazing cat?

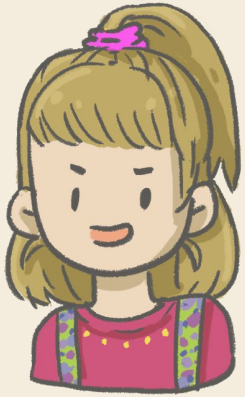


Wow, yes, she’s really cool.




¯_(ツ)_/¯





Paxos has three roles:

Proposers
propose values

Acceptors
accept them

Learners
learn the agreed upon value

Paxos has four phases:

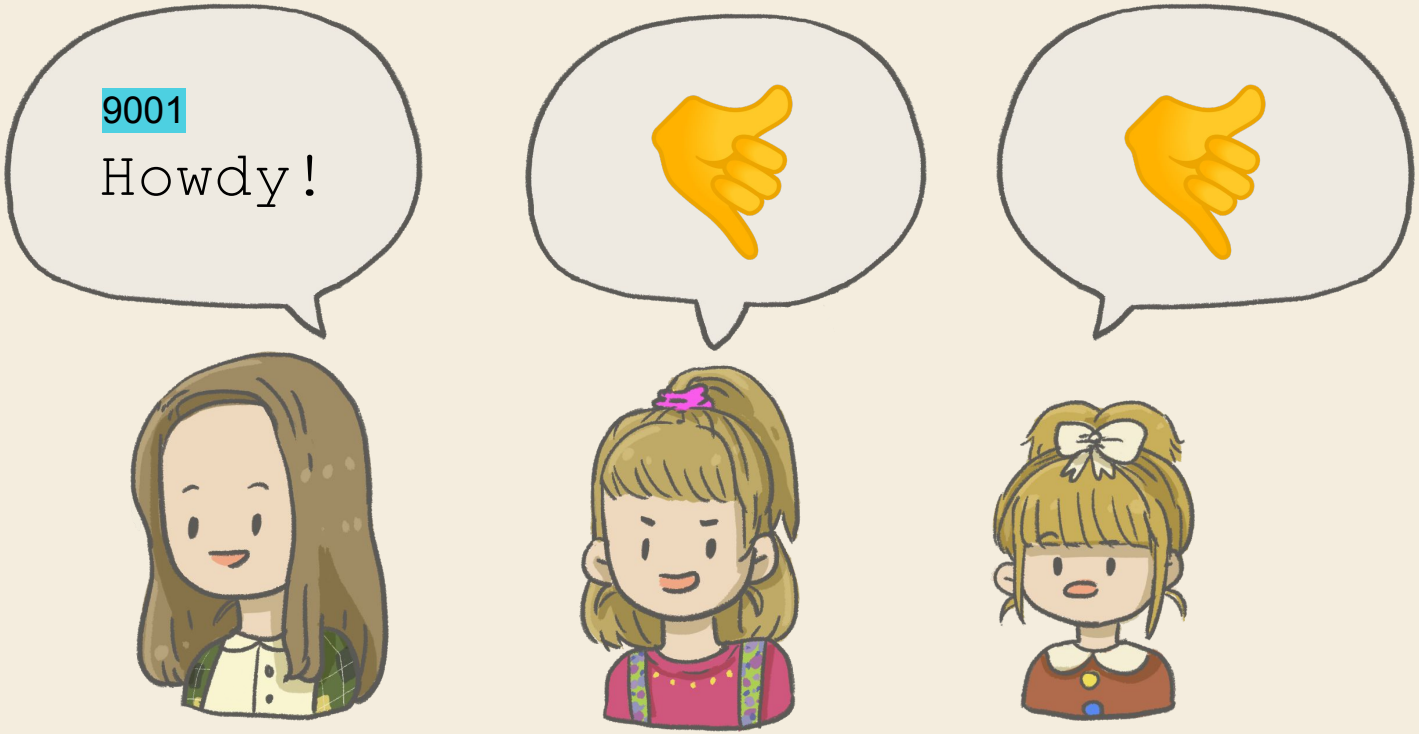
Prepare

Promise

Accept

Accepted

Prepare

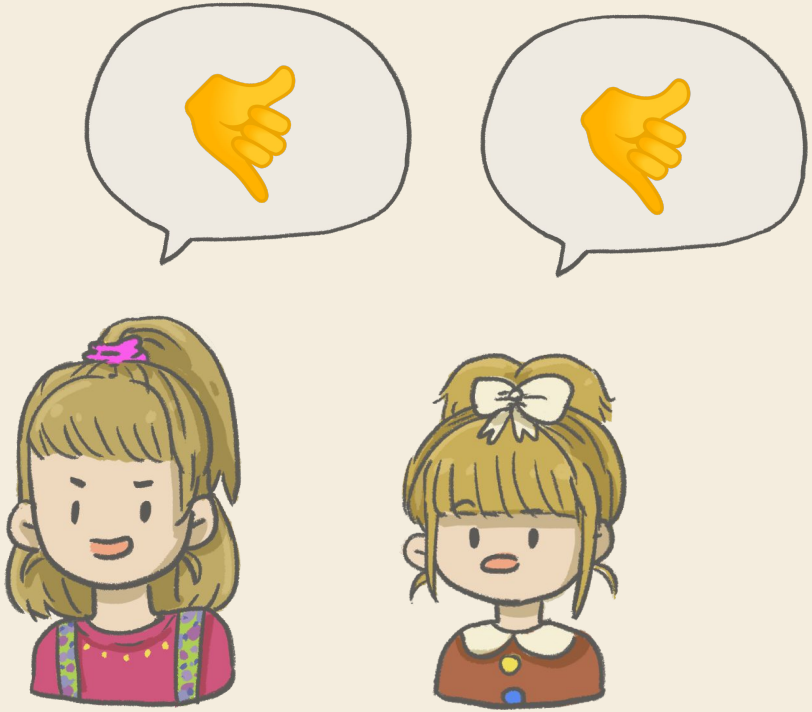


Promise

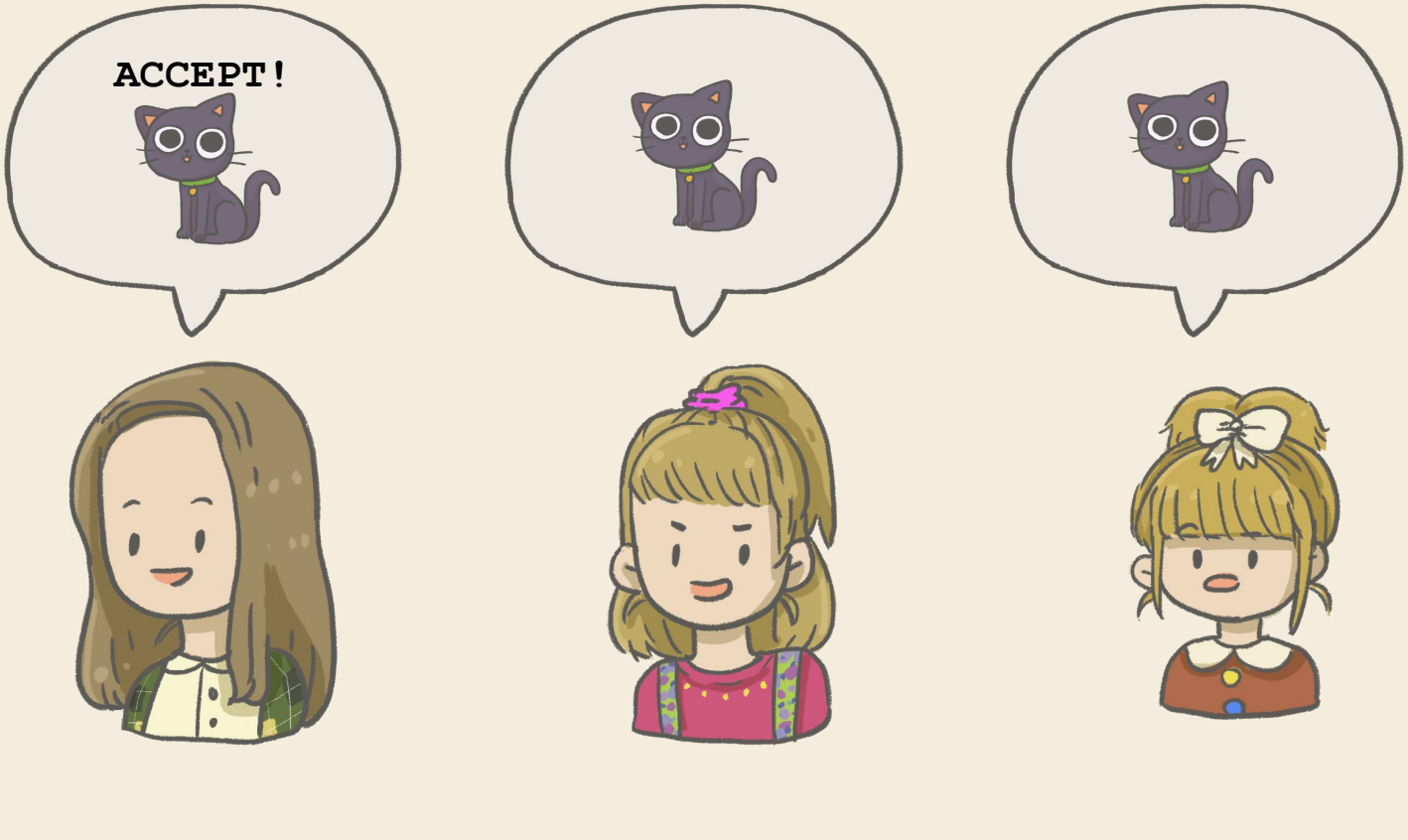
Accepters ask themselves:

- Have we seen a proposal number higher than 9001?

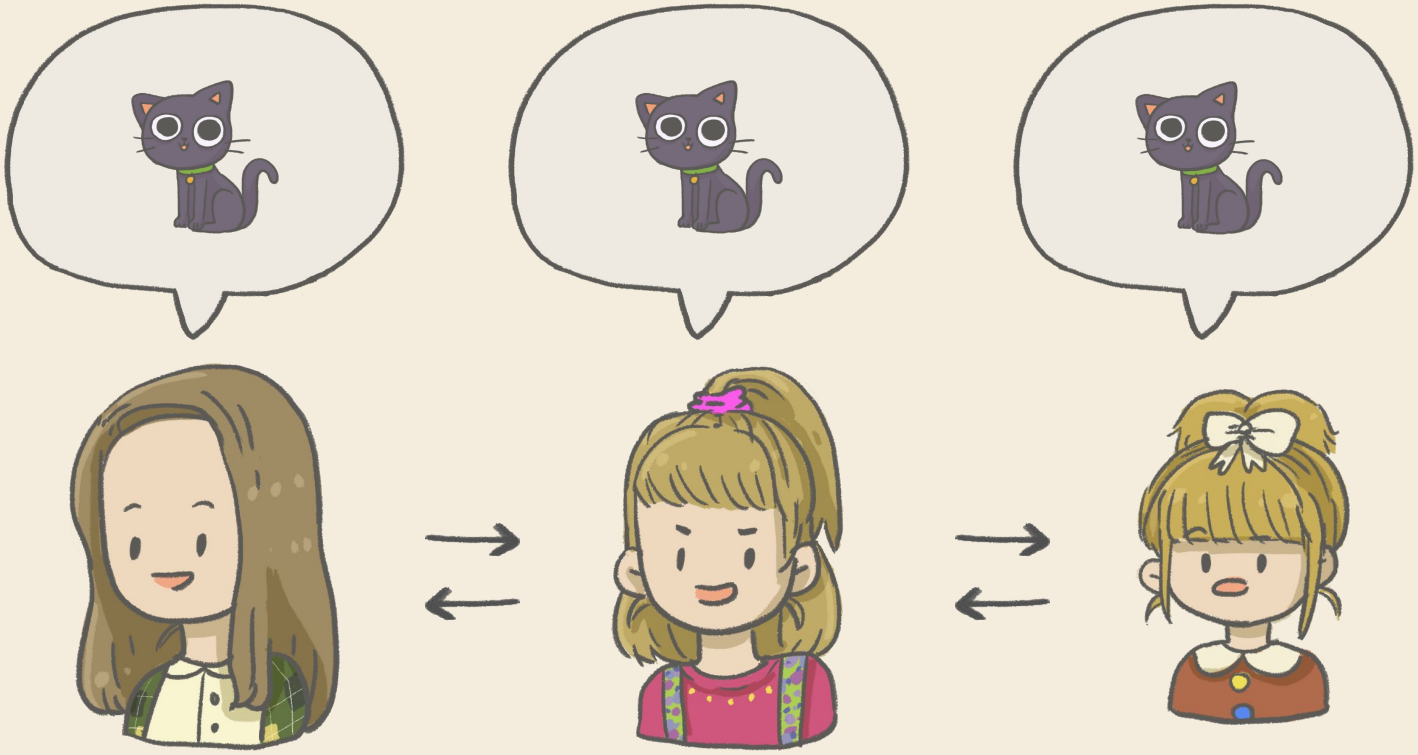
- If not, let’s promise never to accept something > 9001, and say our highest proposal is 8999.



Accept



Accepted



Breakdown

Paxos needs 2m+1 servers to tolerate m failures.

Ex. : I want my cluster to tolerate 2 failures, m = 2
2 * 2 + 1 = 5

I need five servers.

Breakdown: Failures

Proposer fails during prepare phase:

- Proposer is now unable to accept promise, therefore it doesn’t complete.
- A new proposer can take over though.

Breakdown: Failures

Proposer fails during accept phase

- Another proposer tries to overwrite the job, but someone will eventually tell the new proposer about the previously unfinished business. The new proposer will update their value to the previous value.

Breakdown: Failures

Acceptor fails

- Keeps running unless majority fails

Leaderless Byzantine Paxos

- Leaders are huge pain point to become Byzantine Fault Tolerant
  - Once a leader is malicious it is difficult to choose a new leader
  - Lamport calls Castro and Liskov's method for detecting this "ad hoc".
- Each server is a virtual leader
  - The message is sent out to each server
  - All virtual leaders synchronously send back their responses

“If the system does not behave synchronously, then the synchronous Byzantine agreement algorithm may fail, causing different servers to choose different virtual-leader messages. This is equivalent to a malicious leader sending conflicting messages to different processes.”

- Leslie Lamport

“There are significant gaps between the description of Paxos and the needs of the real world system...”

- Google Chubby Authors

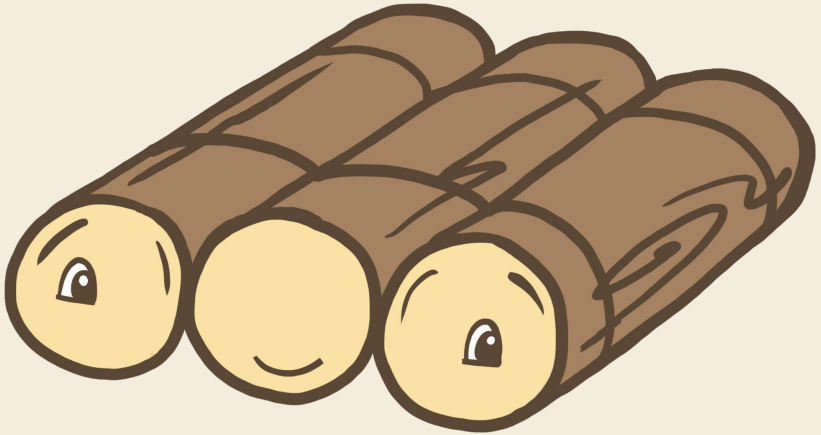
Raft

History

In Search of an Understandable Consensus Algorithm, Diego Ongaro and John Ousterhout est. 2013

College students set out to make a more understandable consensus algorithm

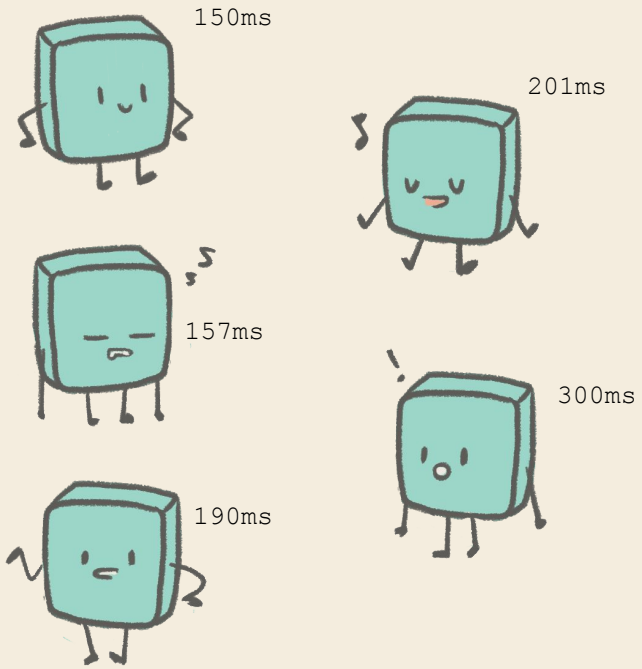
Stands for “Replicated and Fault Tolerant”



Breakdown

In the beginning Raft has to decide a leader.

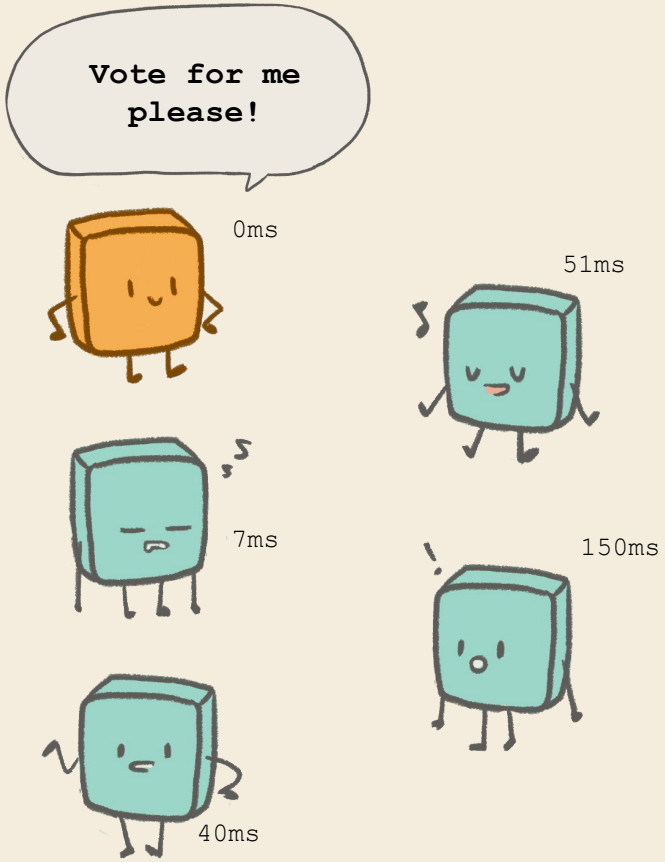
Each node will have a randomized timeout set



Breakdown

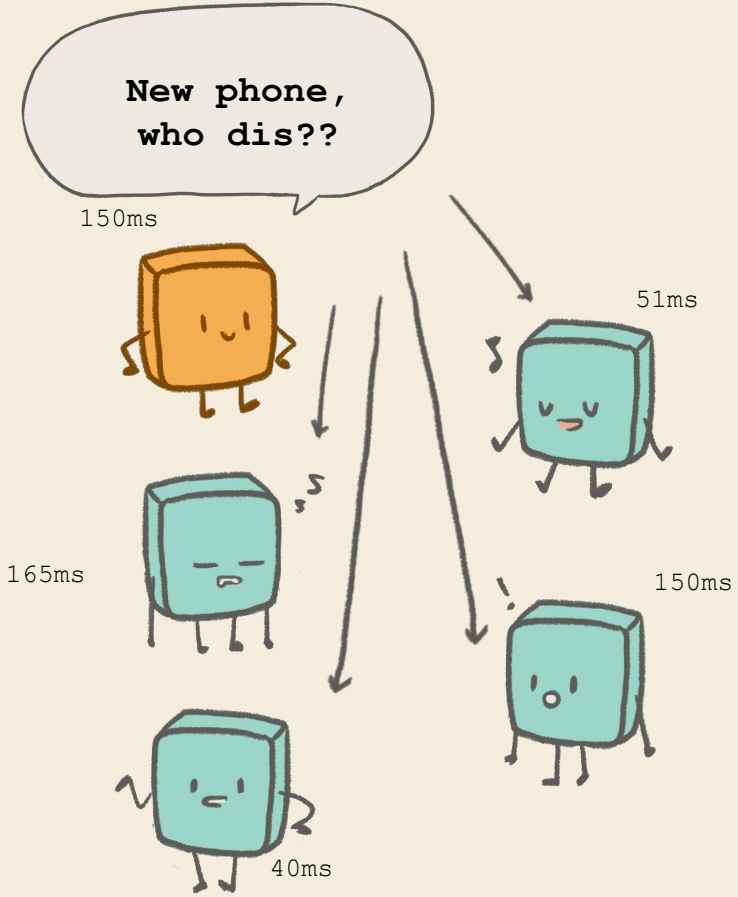
The first node to reach the end of it’s timeout will request to be leader

A node will typically reach the end of it’s timeout when it doesn’t get a message from the leader



Breakdown

The elected leader will send out health checks which will restart the other node's timeouts.



Server can be in any of the
three states at any given time:

Follower

Listening for heartbeats

Candidate

Polling for votes

Leader

Listening for incoming commands, sending out heartbeats
to keep term alive

Breakdown

Raft is divided into **terms**, where at most there is one leader per term.

- Some terms can have no leaders

"*Terms identify obsolete information*"

- John Ousterhout

- Leader's log is seen as the **truth**, and is the most up to date log.

Breakdown: Leader Election

Timeout occurs after not receiving heartbeat from leader

Request others to vote for you

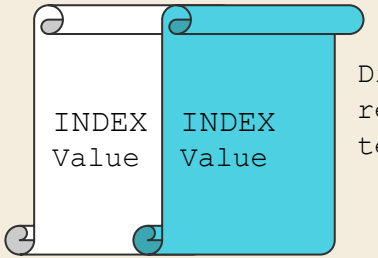
Becomes leader, send out heartbeats

Somebody else becomes leader, become a follower

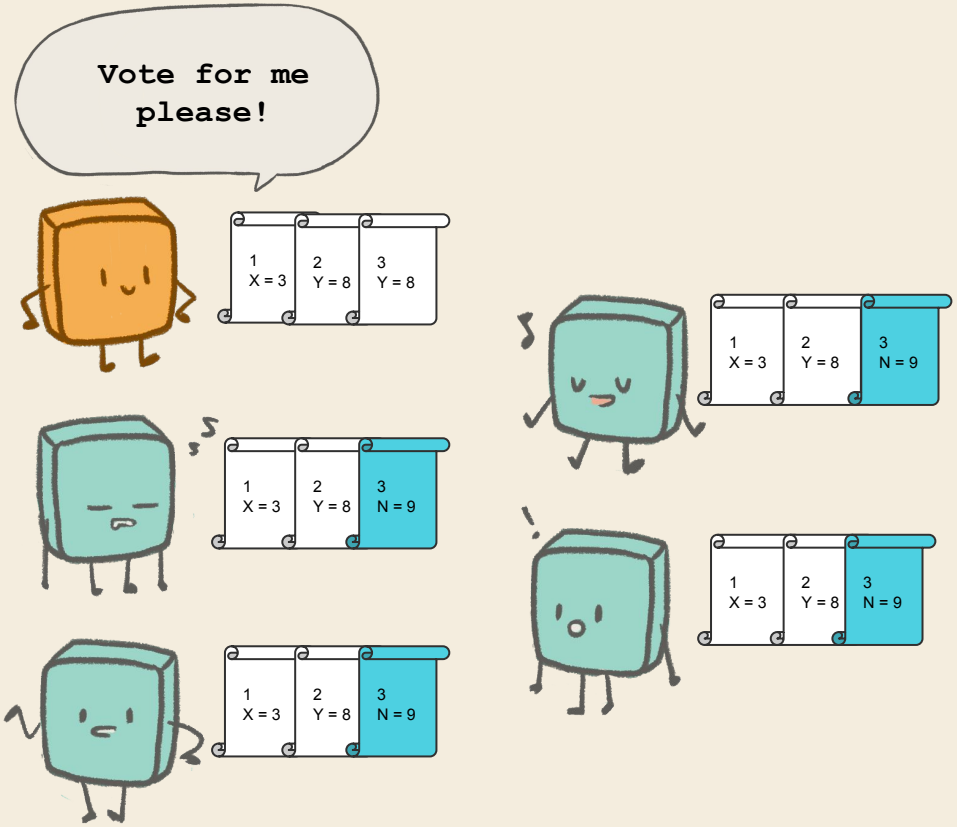
Vote split, nobody wins. New term

Leader Election

- Candidates will deny a leader if their log has a higher term, higher index then the proposed-leaders log.



Different color represents new term



Breakdown: Log Replication

"Keeping the replicated log consistent is the job of the consensus algorithm."

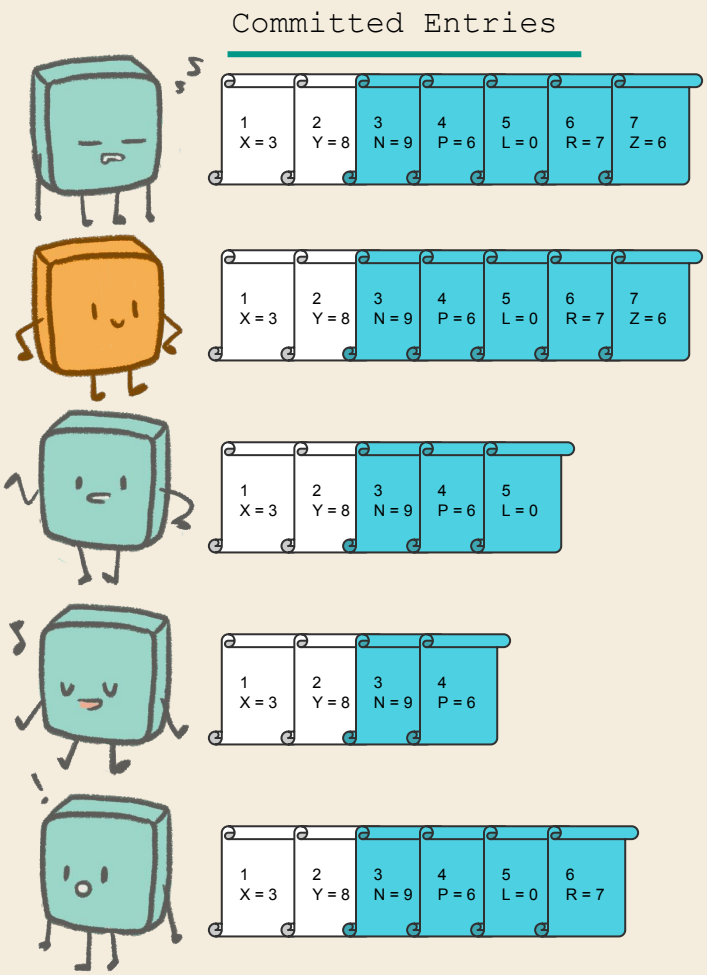
- Raft is designed around the log.

Servers with inconsistent logs will never get elected as leader

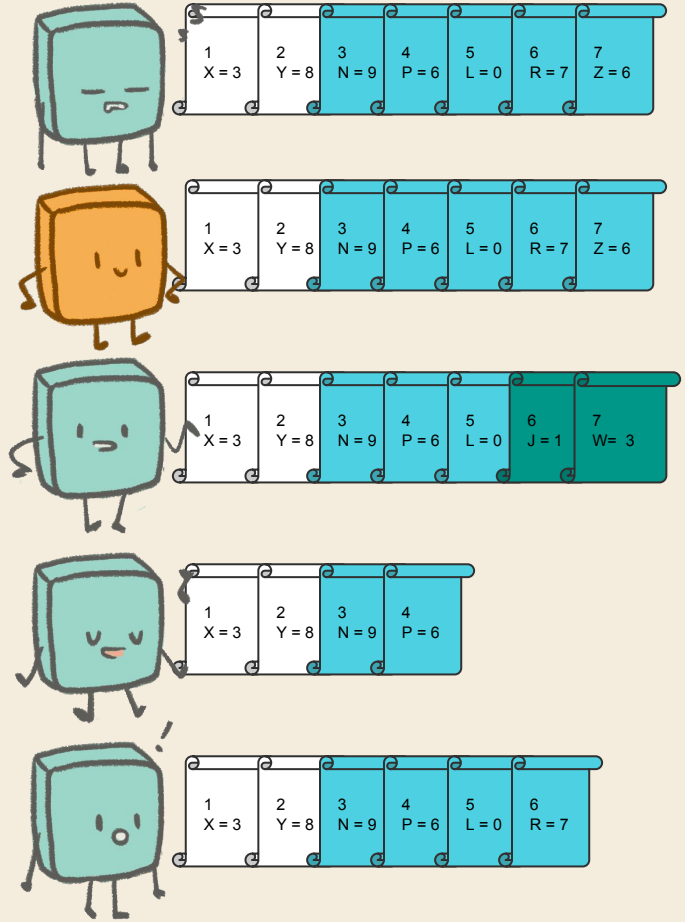
- Normal operation of Raft will repair inconsistencies

Breakdown: Log Replication

- Logs must persist through crashes
- Any committed entry is safe to execute in state machines
  - A committed entry is replicated on the majority of servers

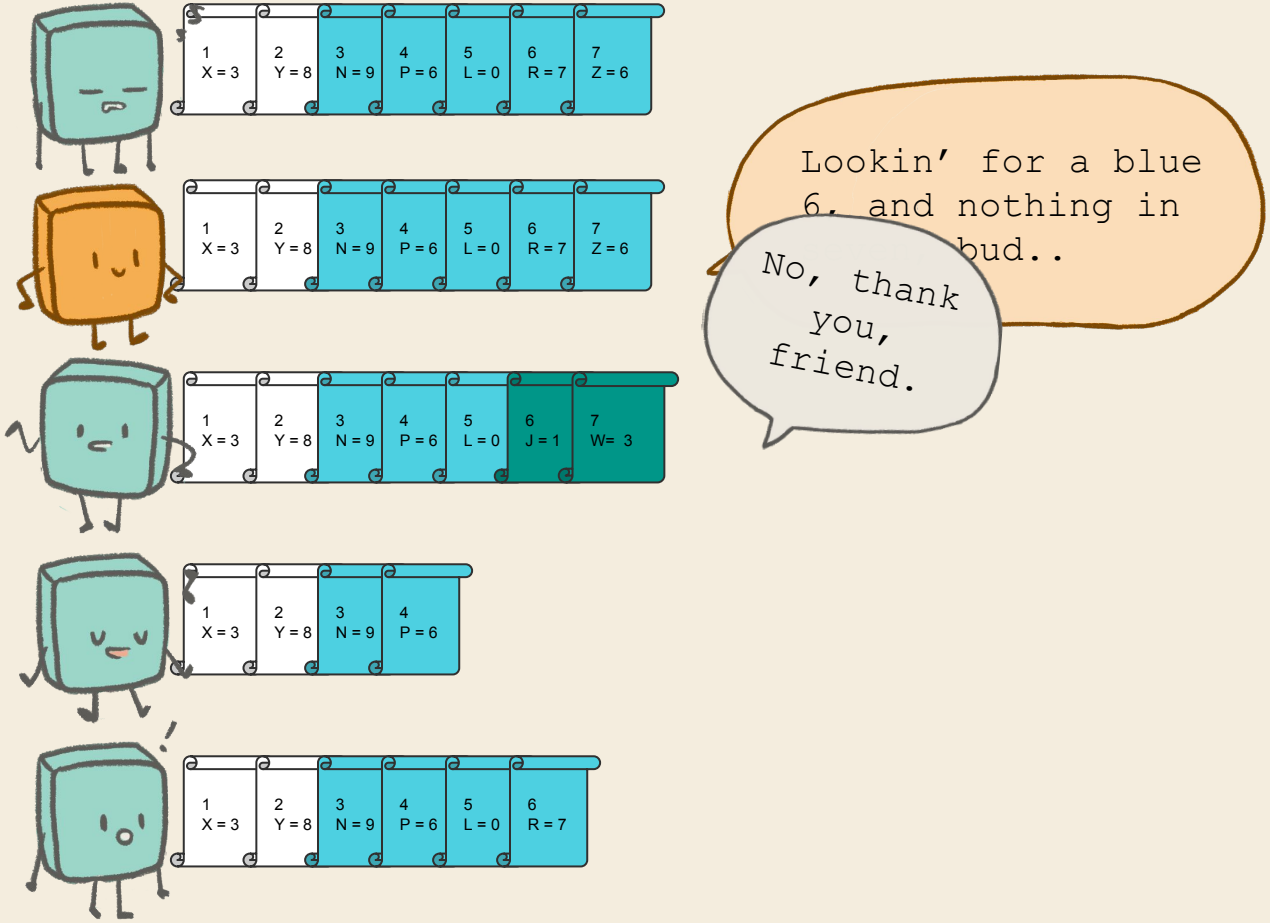


Breakdown: Log Replication

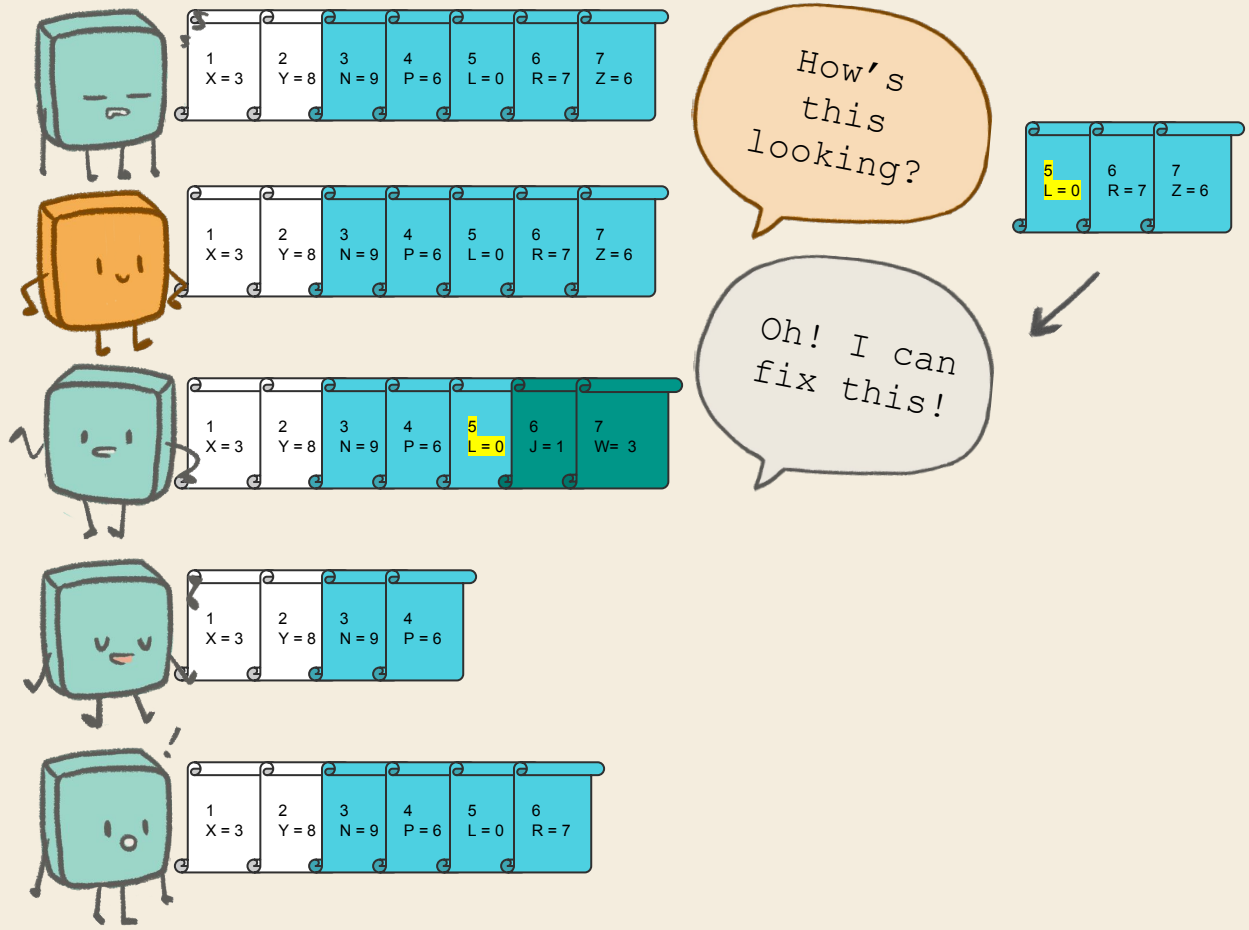


Lookin' for a blue 6, and nothing in seven, bud..

Breakdown: Log Replication



Breakdown: Log Replication



Breakdown: Failures

Normal operations will heal all log inconsistencies

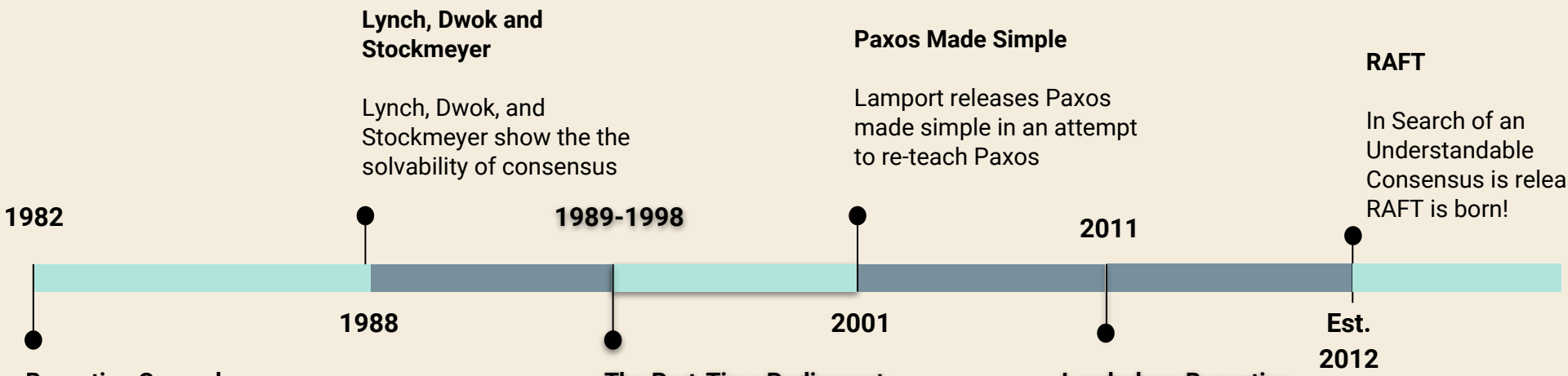
- If leader fails before sending out new entry, then that entry will be lost

A Byzantine Fault Tolerant Raft

Tangaroa: a Byzantine Fault Tolerant Raft:

- Uses digital signatures to authenticate messages
- Clients can interrupt current leadership if it fails to make progress. Disallows unloyal leaders from starving the system.
- Nodes broadcast each entry they would like to commit to each other, not just the leader

Summary


1982
1988
1989-1998
2001
2011
Est.
2012
Byzantine Generals Problem
Lamport, Shostak, and Pease release the Byzantine Generals Problem to describe failures in reliable computing.
Lynch, Dwok and Stockmeyer
Lynch, Dwok, and Stockmeyer show the the solvability of consensus
The Part-Time Parliament
Lamport spends almost ten years trying to get his paper accepted.
Paxos Made Simple
Lamport releases Paxos made simple in an attempt to re-teach Paxos
Leaderless Byzantine Paxos
Lamport releases three page paper describing how to make a Byzantine Fault Tolerant Paxos
RAFT
In Search of an Understandable Consensus is relea
RAFT is born!

- When something talks about Byzantine, you know what it means
  - You can use this in meetings to sound really cool

- If anyone says “Let’s use Paxos” you can tell them why it’s probably not a good idea

- If someone tells you that X problem is occuring because of Raft, you may be able to tell them they’re wrong.

Lead

Me

ARTWORK BY

Lookmai Rattana

@ cosmicmeoww

cosmicmeow.works
@ gmail.com

cosmicmeow.com

Thank you

Wikipedia:
Paxos
Raft
Consensus Problem
Byzantine Fault Tolerance

Medium:
Loom Network

Whitepapers:
Practical Byzantine Fault Tolerance

Byzantine Leaderless Paxos

Whitepapers:
The Part-time Parliament

The Byzantine Generals Problem

Paxos Made Simple

In Search of an Understandable Consensus Algorithm

Consensus in the Cloud: Paxos Demystified

Tangaroa: A Byzantine Fault Tolerant Raft

Thank you

Misc.

James Aspnes Notes - Paxos
The Saddest Moment
Mark Nelson - Byzantine Fault
Good Math - Paxos
Byzantine Failures - NASA
GoogleTech Talk - Paxos
Talk on Raft
Raft Website
CSE452 at Washington State
Practical Byzantine Fault Tolerance