

SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol

Das, A., **Gupta, Indranil** and Motivala, A., 2002, June. Swim: Scalable weakly-consistent infection-style process group membership protocol. In *Proceedings International Conference on Dependable Systems and Networks* (pp. 303-312). IEEE.

RAFT Recap



RAFT and similar systems are used to manage ordered logs.



Logs capture the state of a distributed system and so must be handled carefully

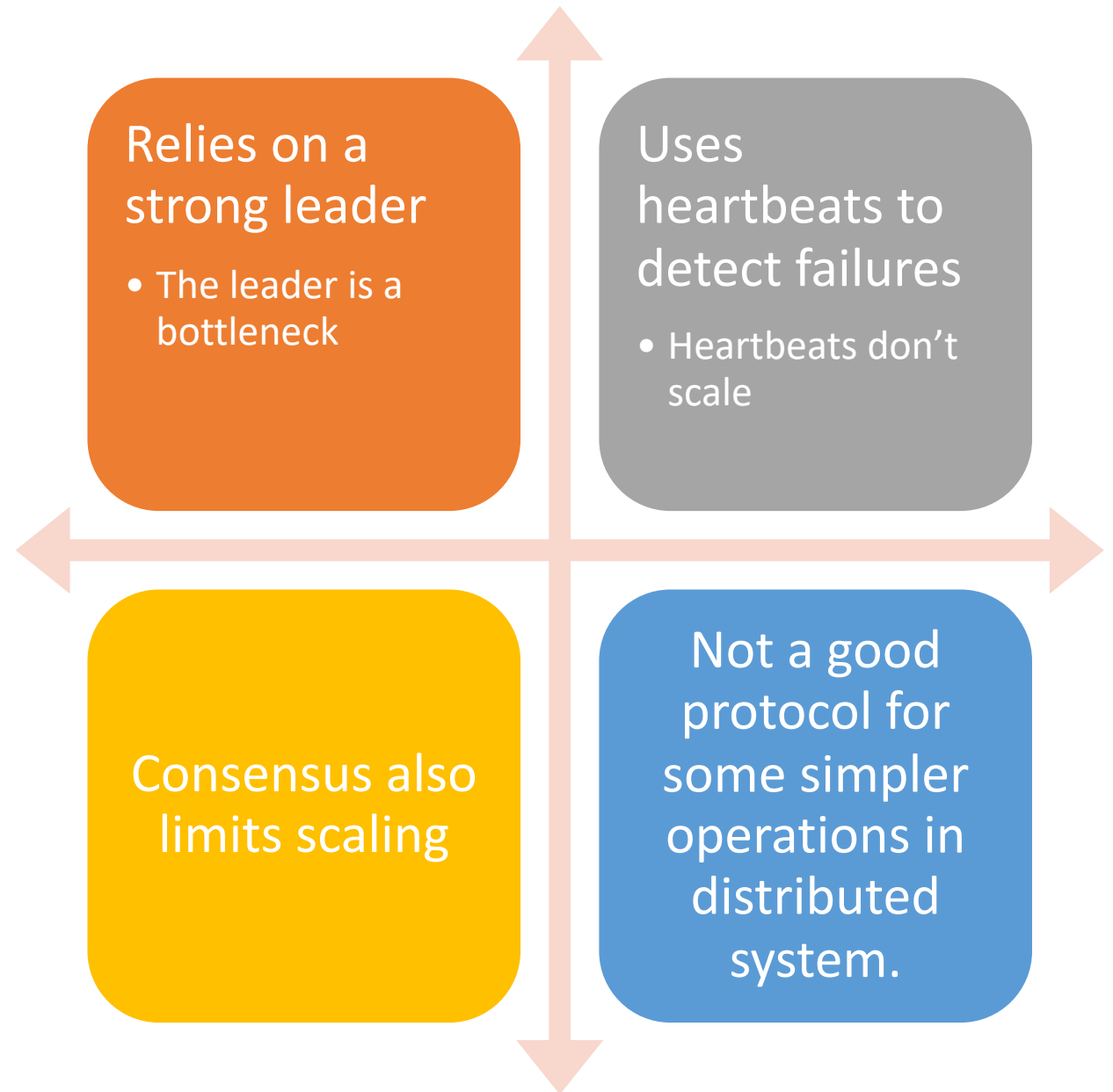


RAFT is fault-tolerant and strongly consistent

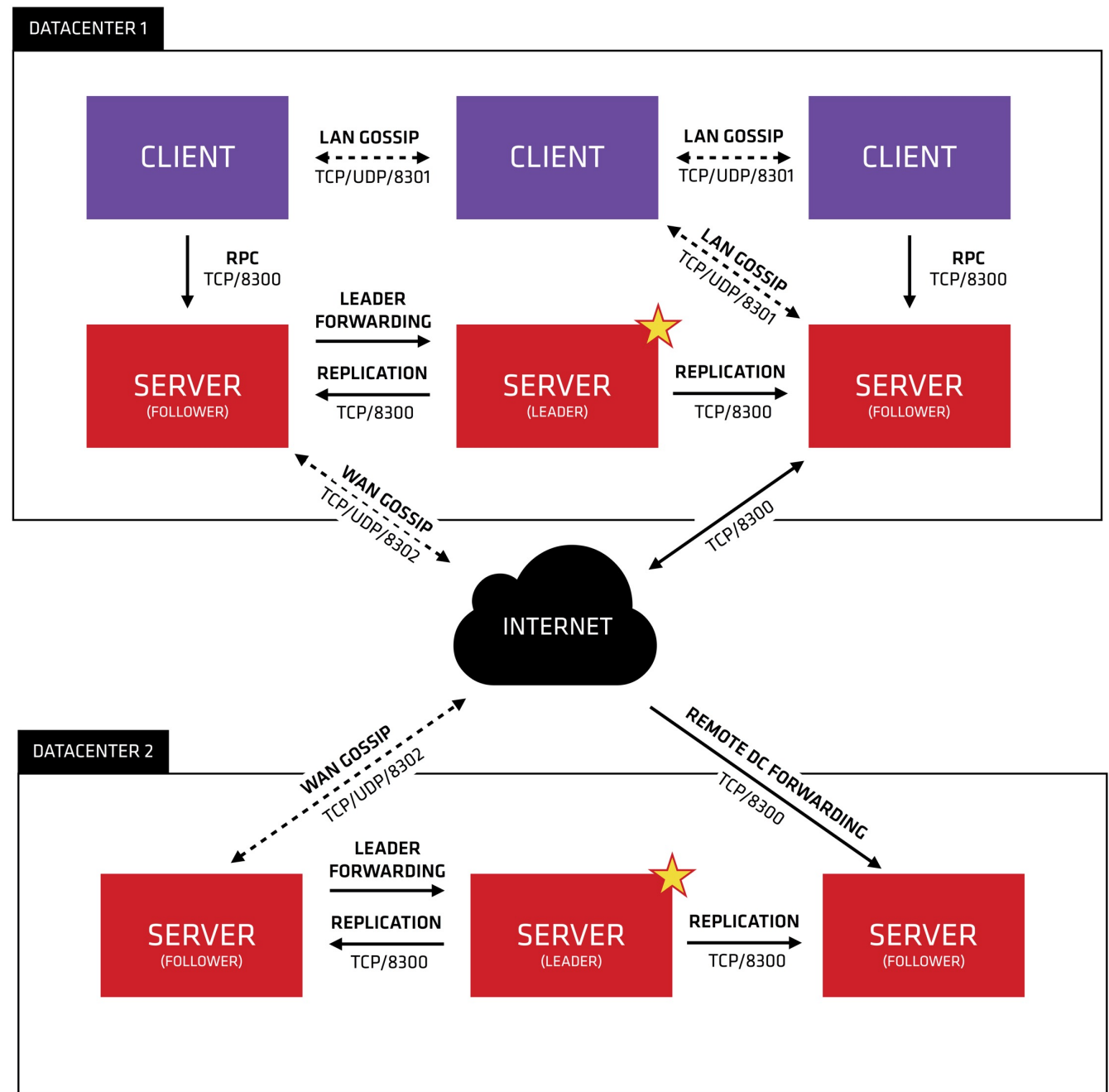


Consensus model

RAFT Scaling Limitations



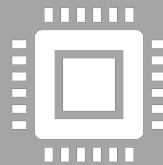
Consul, RAFT, and SWIM/Gossip



Scaling Beyond a Single Data Center



There are simpler services than logs that are needed by distributed systems.



Example: how do you detect server failures in large systems?

Two Key Operations for Distributed Systems

Membership: Each process knows all the other members

Faulty members need to be removed from each member's list



Fault Detection: Failed processes are detected by the other members and communicated throughout the system.

SWIM Insight Compared to Prior Systems



Membership changes and fault detection are separate processes



Monitoring needs to go on all the time, but membership changes because of faults occur on a longer time scale.



Therefore, SWIM has two basic, separate operations: Fault Detection and Dissemination

Problems with All-to-All Heartbeats



Each member of a cluster sends and receives small heartbeat messages with all other members



If M_x doesn't receive a heartbeat from M_y within a timeout, it marks M_y as faulty



This works for smaller systems



But it grows quadratically (like N^2) with the size of the cluster

Properties of Failure Detection Protocols



Strong completeness: crash failures are detected by all non-faulty members



Speed of detection: the time interval between a failure and its first detection



Accuracy: How confident are you that a cluster member has failed?



Network Message Load: how many messages are required?

You Can't Have It All

- Failure detection in an asynchronous network cannot be simultaneously 100% accurate and strongly complete
- You need to make a choice
- **Strong completeness** is the usual choice
- Therefore, we need a way to minimize false positives
 - That is, incorrectly marking a process as failed when it has not

SWIM Failure Detection, Step 1


SWIM cluster members only monitor a subset of the other members.



You have two parameters: T_p (protocol time) and K (# of members to monitor)



Every T_p seconds, each member sends out a **PING** to K other members of the cluster.



If M_x receive the ACK from M_y within a timeout period, all is well.



No need to update memberships.

Swim Failure Detection, Step 2



If X doesn't hear from Y before the timeout, it asks for help



X sends a PING-REQ(Y) message to K other members of the cluster



The other members PING(Y) and return their results to X



If Y responds, and if X gets this message back from at least one other member, all is well.



Again, no unnecessary membership updates

SWIM Failure Detection, Step 3

If X cannot confirm that Y is alive even after Step 2, then it needs to tell the rest of the entire cluster that it has detected a failure.

This is the **Dissemination** part of the protocol

Disseminating Failures

This part of the protocol needs to propagate efficiently to the entire system.

We assume failures are relatively rare compared to the protocol time T for PINGs

Infection-Style Dissemination

- X includes information about failed members in all of its communications with other members
 - Hey, Z, PING, and by the way, Y has failed
 - Hey, group, PING-REQ(A), and, by the way, Y has failed
 - Hey, Z, I'm alive, here's your ACK, and by the way, Y has failed.
- Anyone receiving this message from X will remove Y from its group list

Reducing False Positives with Suspicion

A healthy cluster member may fail the PING test because of temporary network issues or its own load.

SWIM uses a “Suspicion” subprotocol to reduce these.

Propagating Suspicion



If Y fails X's PING and PING-REQ tests, X marks it as "suspicious" rather than failed.



X propagates the message "X suspects Y" to the rest of the cluster



Z marks Y as suspicious when it receives the message



Suspicious Y is still treated as alive by the cluster

Suspicion, Continued



If Z later successfully pings Y, it moves it back to its “alive” list



Z then propagates the message “Z knows Y is alive” to the rest of the cluster



Other processes remove Y from their suspicious list when they receive this message.



Y can also disseminate this message

Suspicion, Continued

Suspected entries are marked as failed if they haven't responded within a time out.



If X expires Y, it will propagate the message

“X confirms Y has failed”

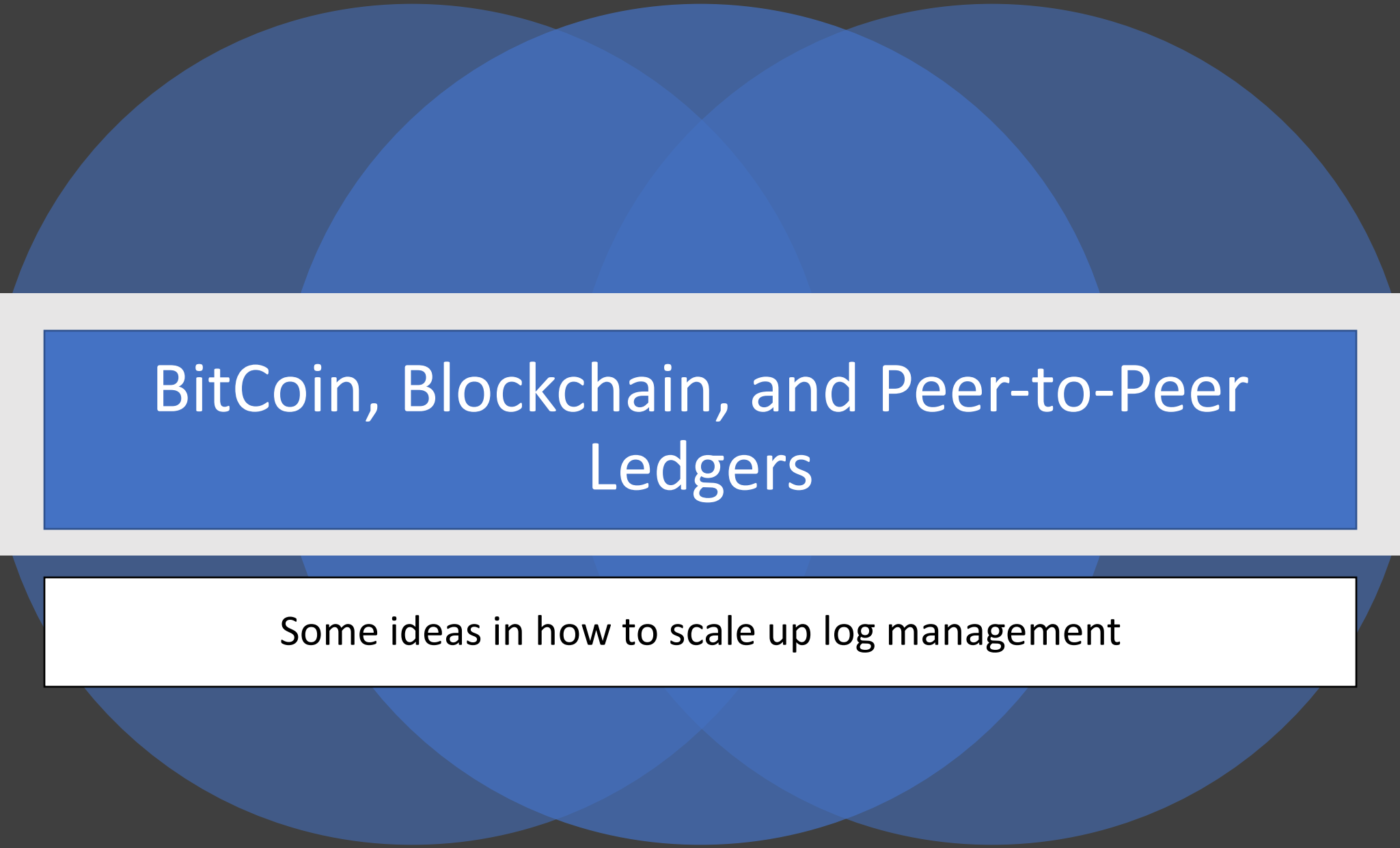
SWIM Scaling Properties

Imposes a constant message load per group member regardless of the cluster size for both failure detection and membership changes

At least one working member detects failures within a constant time.

The time it takes to learn about failures increases logarithmically (slowly) with cluster size (good thing)

Reduces false positives (PING-REQ and Suspicion mechanisms) without performance penalties



BitCoin, Blockchain, and Peer-to-Peer Ledgers

Some ideas in how to scale up log management

RAFT: Reliably Managing State with Logs



Has a strong leader



Provides fault tolerance and consistency
with limited scaling



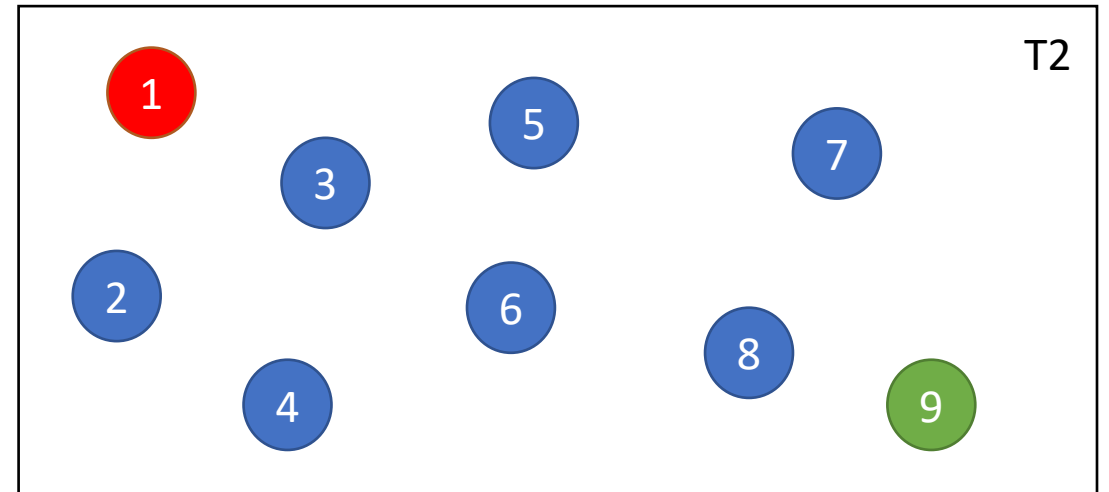
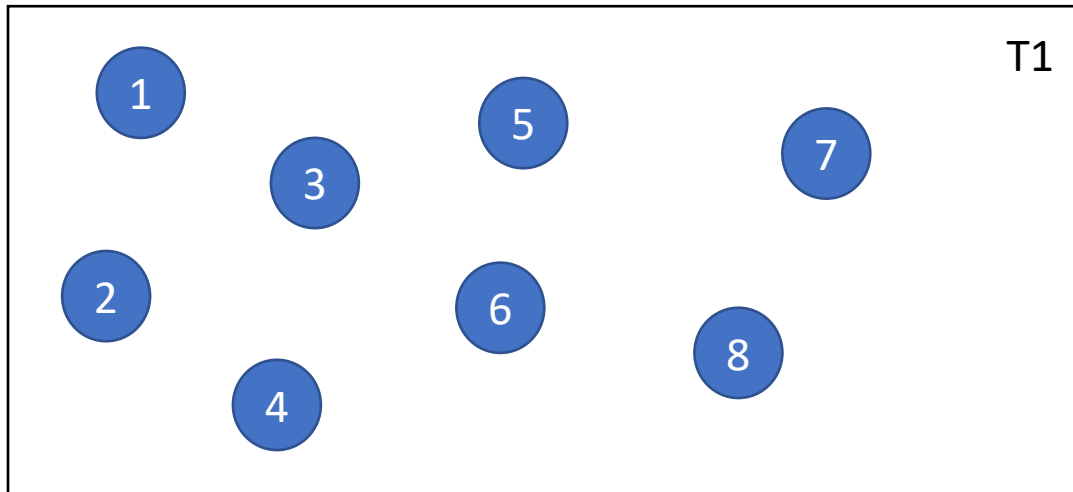
Can coordinate process states in a system
that needs consensus consistency.



Example: transaction logs in a database

SWIM

- Peer-to-peer
- Eventually consistent
- Good for tracking state changes that don't need an audit trail.
- Ex: group membership changes (maybe)



Can We...



Find a process that scales like SWIM, but...



Can create consistent audit trail logs?

Maybe...



<https://bitcoin.org/en/bitcoin-paper>

Bitcoin Overview



Bitcoin uses keys and hashes to create a globally scalable monetary system.



In other words, it is a peer-to-peer transaction system.



So it needs to both scale and be consistent.



How?



Start with security basics



Asymmetric Keys

- **Private Keys:** Cryptographically sign messages
- Send the signature along with the message
- Recipients use the **public key** to verify that the message came from the signer

Cryptographic Hashing

A *hash* algorithm is a fast mathematical function that generates a unique, hard-to-guess numerical value from a given input

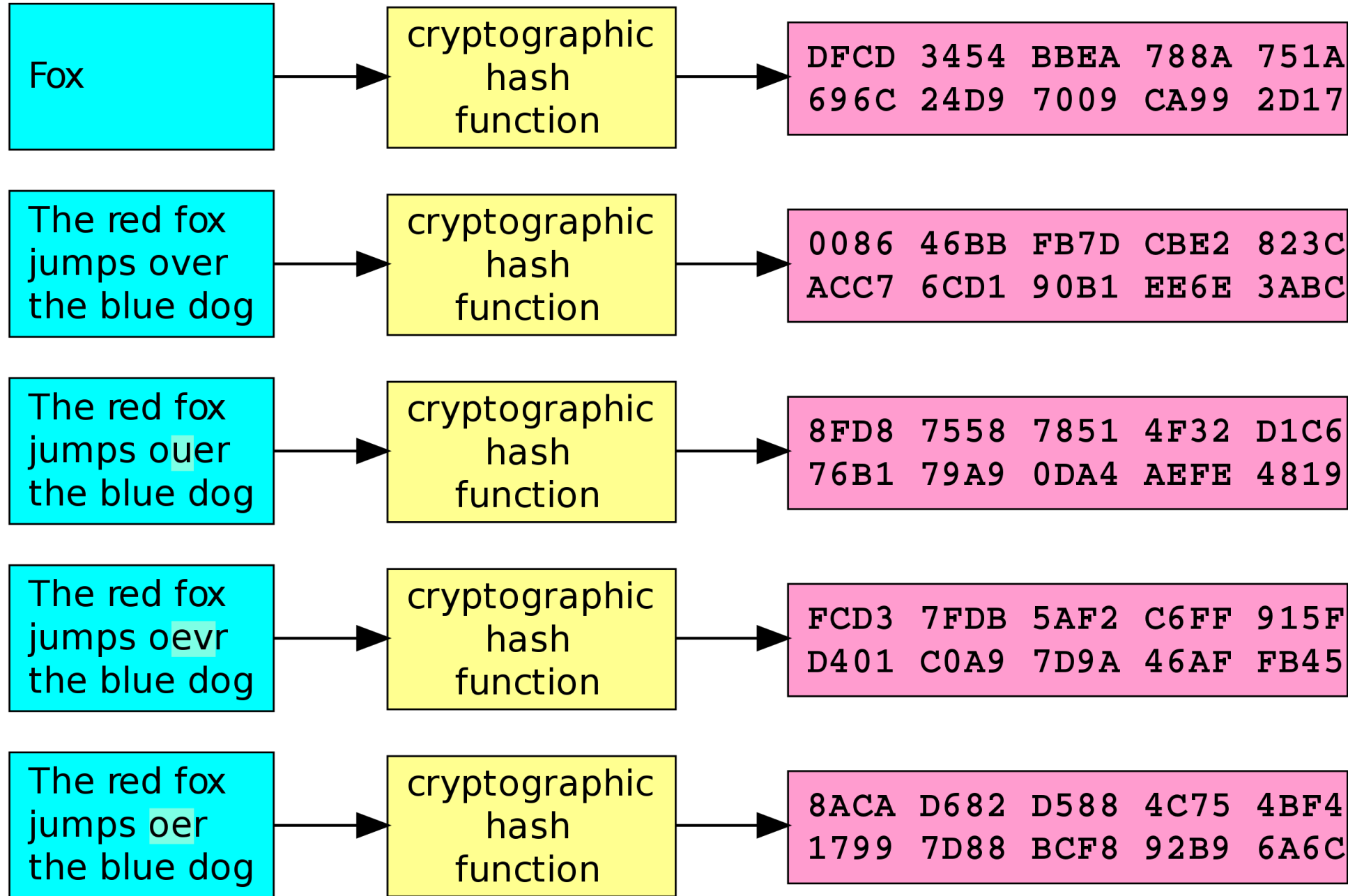
Two messages differing by a single character generate completely different hashes.

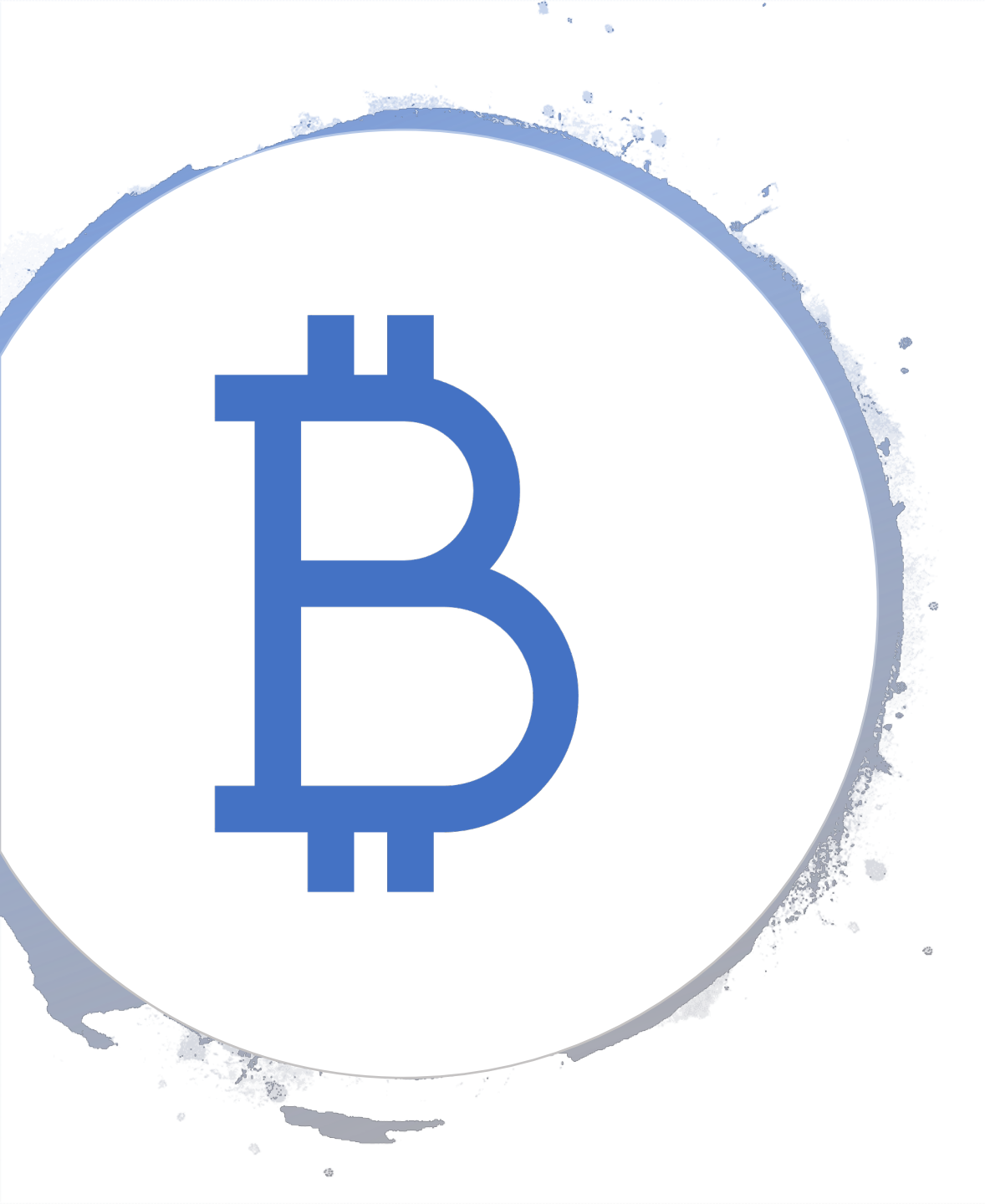
Hashes are not reversible: given a value, you can't easily guess the original input

Hashes are a simple way to verify that data hasn't been corrupted or modified during transmission

Input

Digest





Bitcoin Basic Transactions

- Assume for the moment that bitcoins exist
- Marlon wants to buy a pizza from Suresh and pay in 1 bitcoin.
- Marlon transfers the bitcoin to Suresh in exchange for the pizza.

Steps in a Transaction



Only Marlon (Owner 1) can spend the coin because his public key is embedded in the last coin.



Marlon transfers the coin to Suresh (Owner 2) by digitally signing a hash of the previous transaction and Suresh's public key



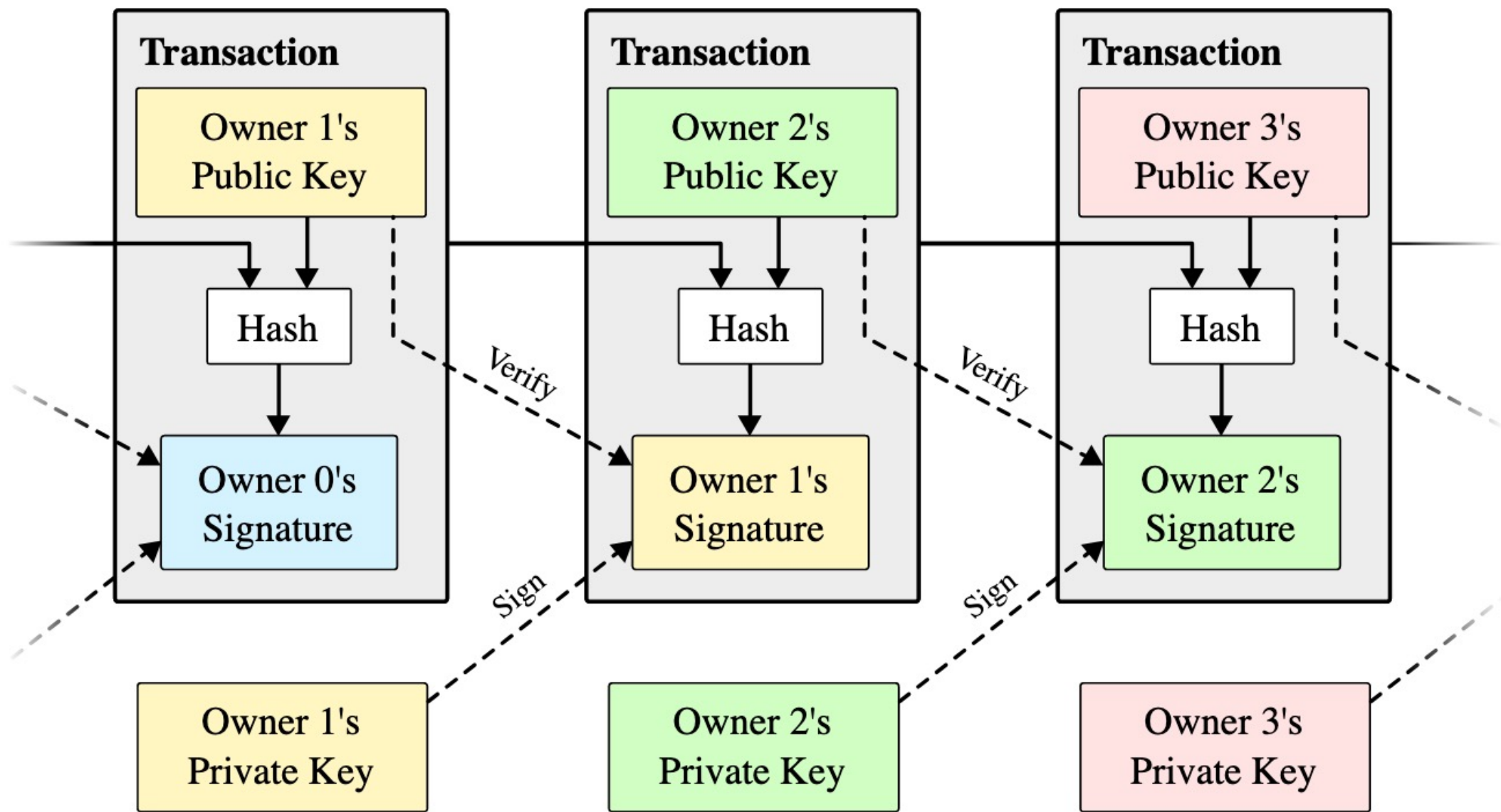
Marlon adds these to the end of the coin.



Suresh can verify the signatures to verify the chain of transactions.



Compare with BFT Raft's logs



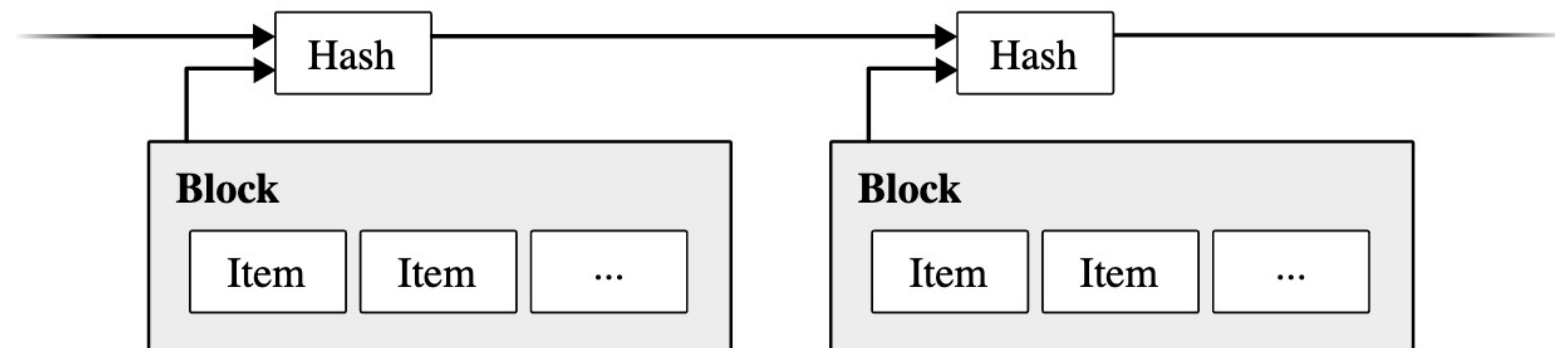


But Can Suresh
Trust Marlon?

- What if Marlon also used the same bitcoin to pay Dimuthu for tacos at the same time?
- This is called double-spending.

Bitcoin's Solution, Part 1: Timestamping and Ledgers

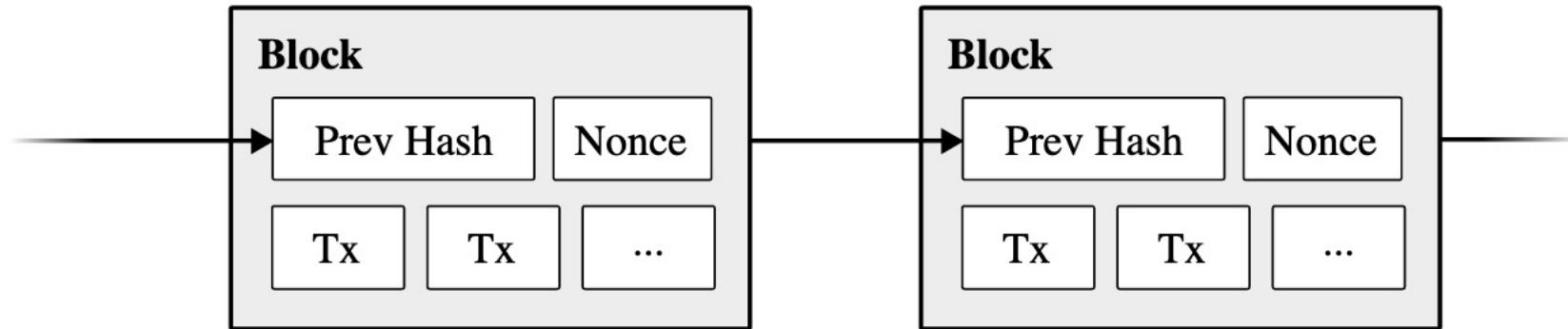
- We publish this information broadly to public ledgers
 - We include a time stamp
- The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash.
- Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.



But how do I keep ledgers in synch?

What if I send different information to different ledgers?

Bitcoin's Solution, Part 2: Proof of Work



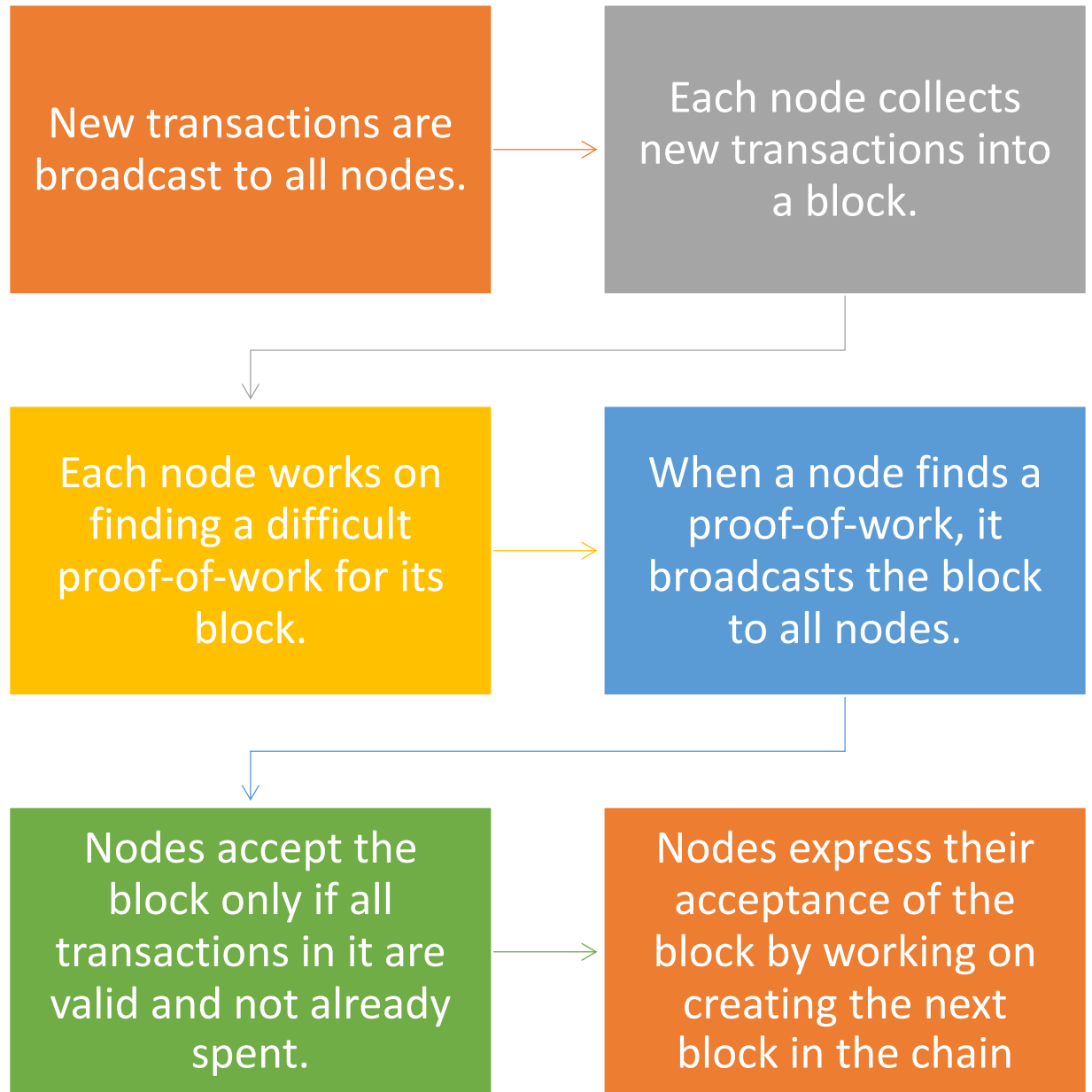
X-Hashcash: 1:52:380119:calvin@comics.net:::9B760005E92F0DAE

Nonce

00000000000000756af69e2ffbdb930261873cd71

Find a nonce that
produces 13 0's

Steps in a Bitcoin Network



Trust the Longest Chain

Nodes always consider the longest chain to be the correct one and will keep working on extending it.

If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first.

In that case, they work on the first one they received, but save the other branch in case it becomes longer.

The tie will be broken when the next proof-of-work is found and one branch becomes longer;

The nodes that were working on the other branch will then switch to the longer one.

Lecture Takeaways

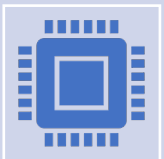


Logs are at the heart of distributed systems



SWIM is a membership management protocol that can scale beyond a data center.

Assumes membership changes don't need to be reliably logged



Blockchain is a log (ledger) system that can scale globally and across organizations

Assumption: members want log availability \gg log completeness
That is, time delays are significant

