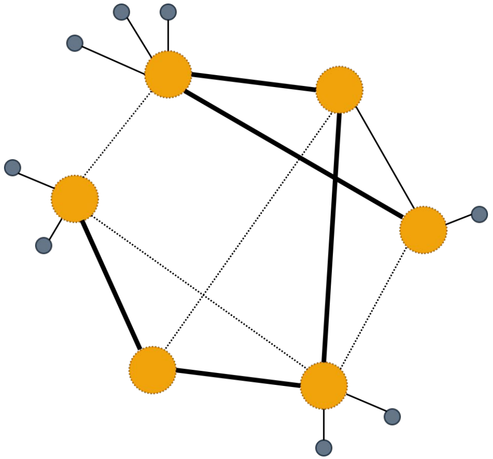
Universidade do Minho
Escola de Engenharia

Decentralized Timeline Service

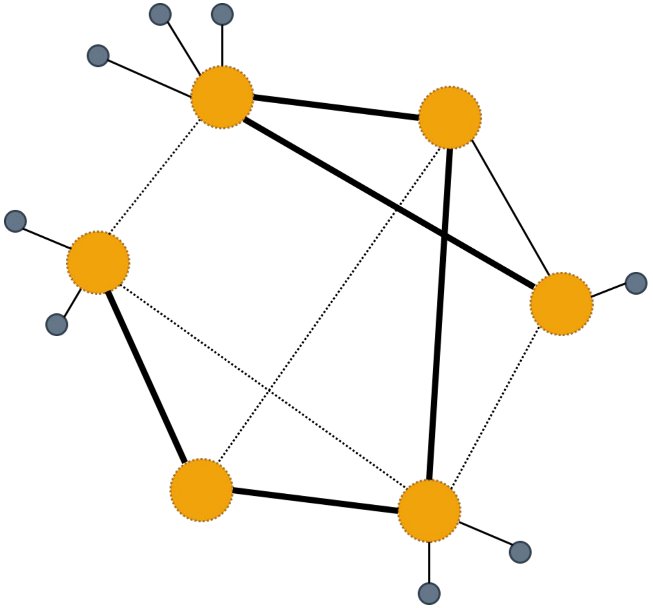
Large Scale Distributed Systems



A84462 | Alexandre Miranda
A84961 | Alexandre Ferreira
A85227 | João Azevedo
A85315 | Miguel Cardoso
A85729 | Paulo Araújo

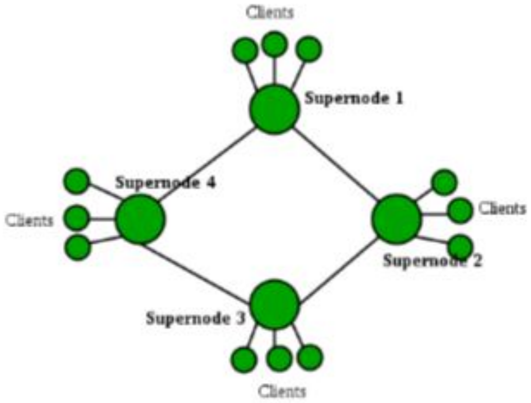
System model

- Each node has **an unique ID**
- Published messages **must reach every active subscriber**
- The timeline should describe a **total order of posts**
- Only **gracious faults** are considered
- Decentralization **only relative** to the timeline posts
- All information stored is **ephemeral**

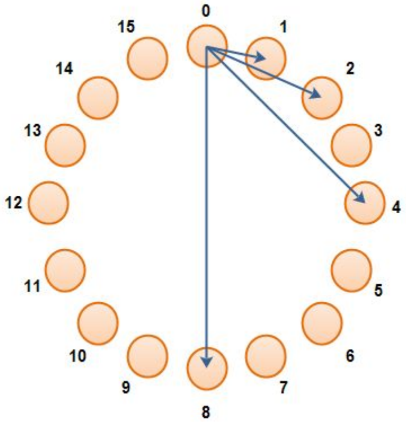


How can we design this system?

Network design



Storage design



Our approach

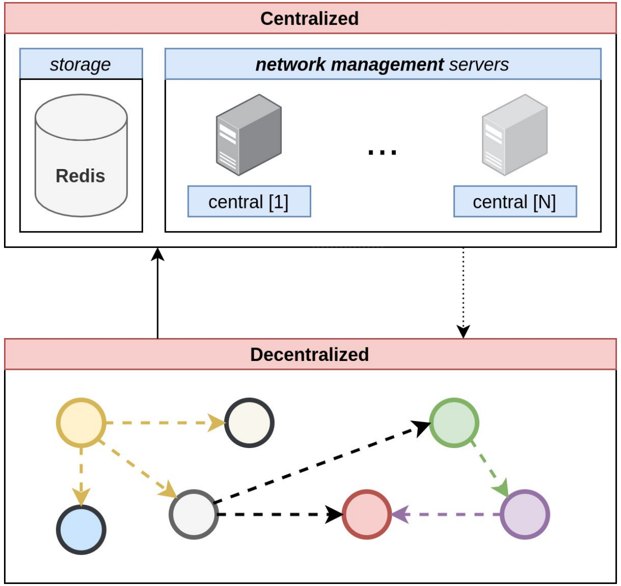
- **Centralized** and **Decentralized** components

- **Network management** (central servers):

  1. Authentication services
  2. Subscription services
  3. Connection management

- **Network peers** (nodes):

  1. Pub-Sub pattern
  2. Timeline management
     a. posts ordering and recovery services

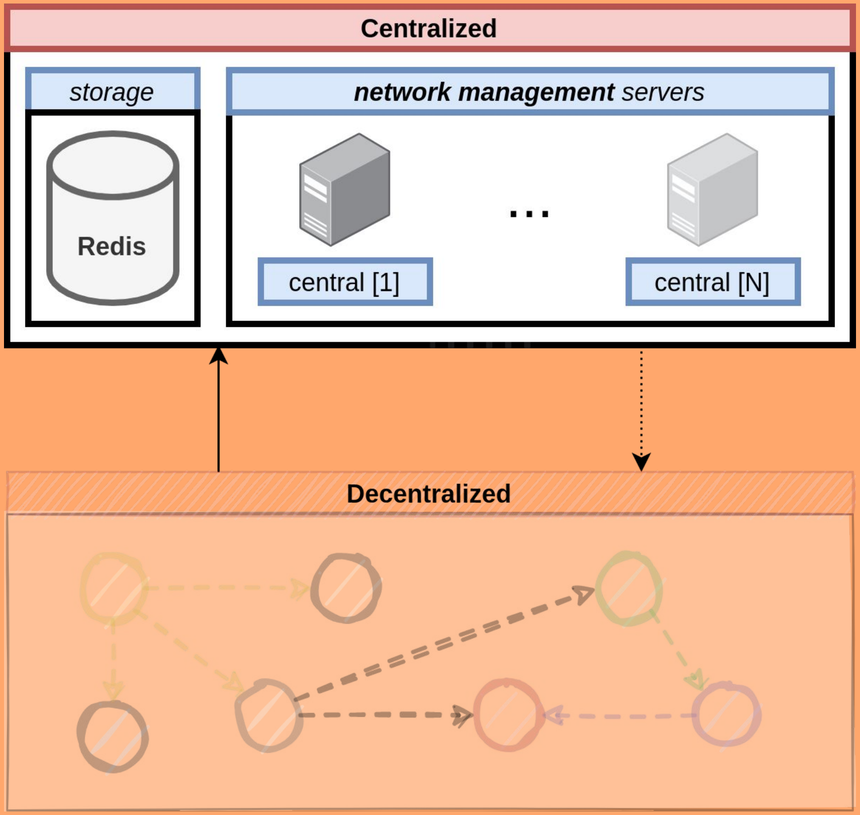


Centralized portion

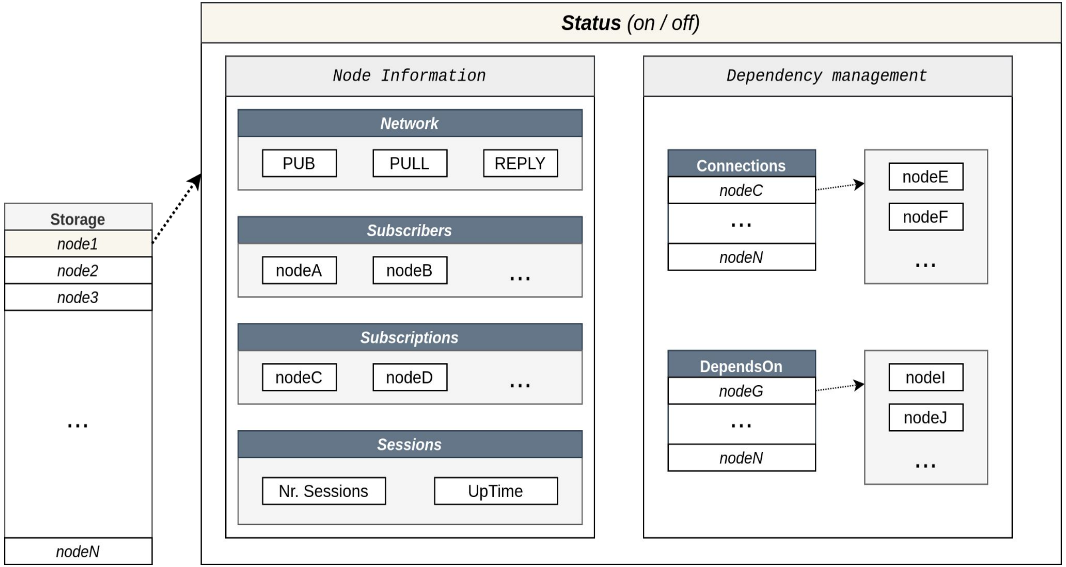
It's used for:

a. Node maintenance
b. Network management
c. Check node status
d. Auth. and Sub. services

"Entry door to the network"



Node Storage Redis Key-Value Store



Use cases Event-Driven server

1. ***Register***
   a. registers new session
2. ***Login***
   a. *new session, manages subscriptions/subscribers*

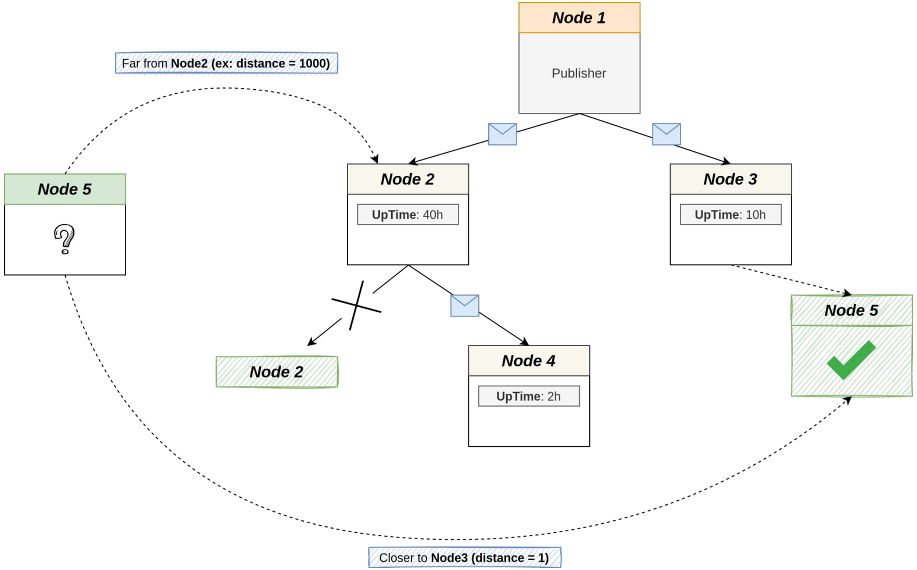
   (we will talk about this one later on)
3. ***Logout***
   a. updates session stats
   b. manages dependencies
4. ***Subscription***
   a. node election

Node election

1. Important for a **reliable subscription service**
2. Candidate with a better score wins
3. How we compute the score?

uptime = upTime(candidate)
distance = diff(**candidate**.port, **subscriber**.port)

score = upTime / distance



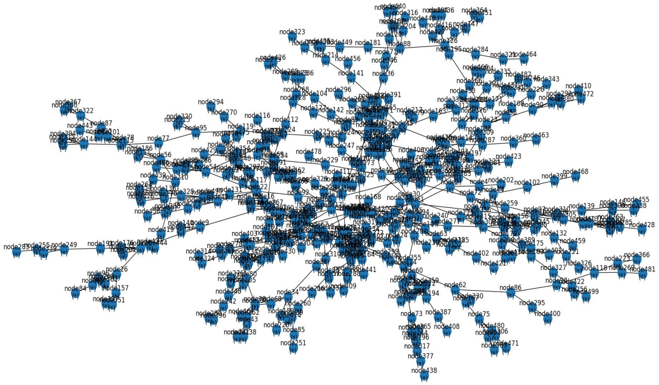
Subscriptions tree

The problem...

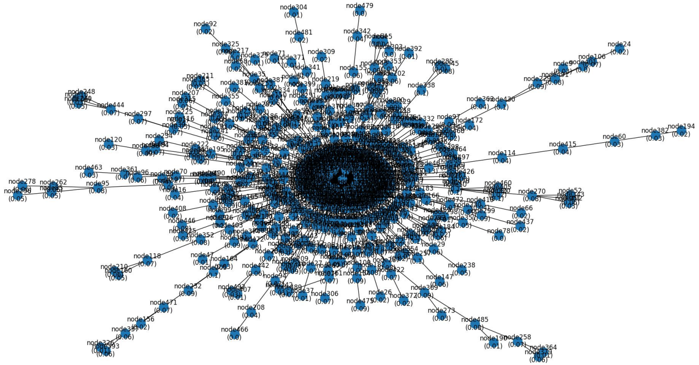
1. A node with a lot of subscribers **restarts its session**
2. How to **reestablish the network**?
    - Recursive node election?
    - Recursive random connections?
    - Preferential Attachment?

Maybe not! We must balance the <u>total load</u> on the central server and the <u>connection reliability</u>...

Random connected



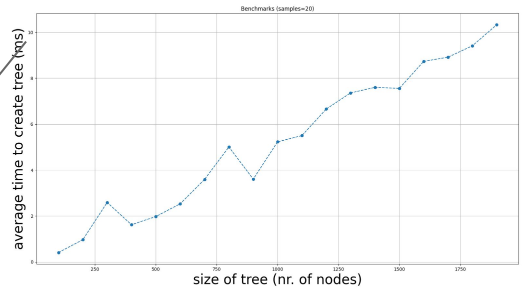
Score based + Random



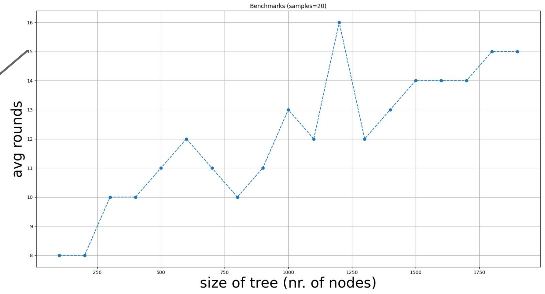
Tree **construction** and **broadcast** (2000 nodes)

Random tree

10 (ms)

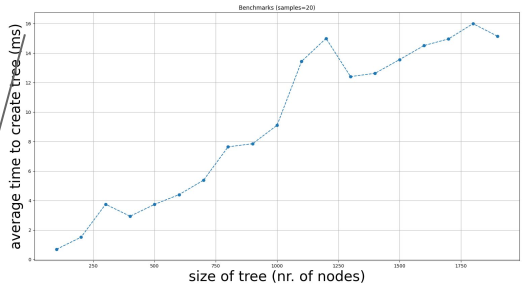


16 rounds

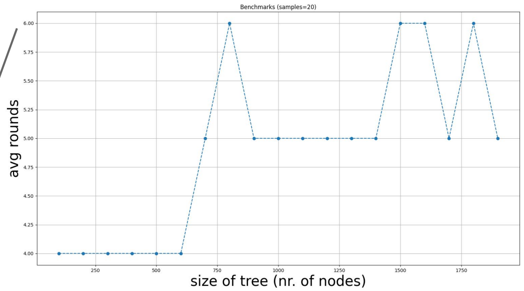


Point-based tree + random attachment

16 (ms)



6 rounds

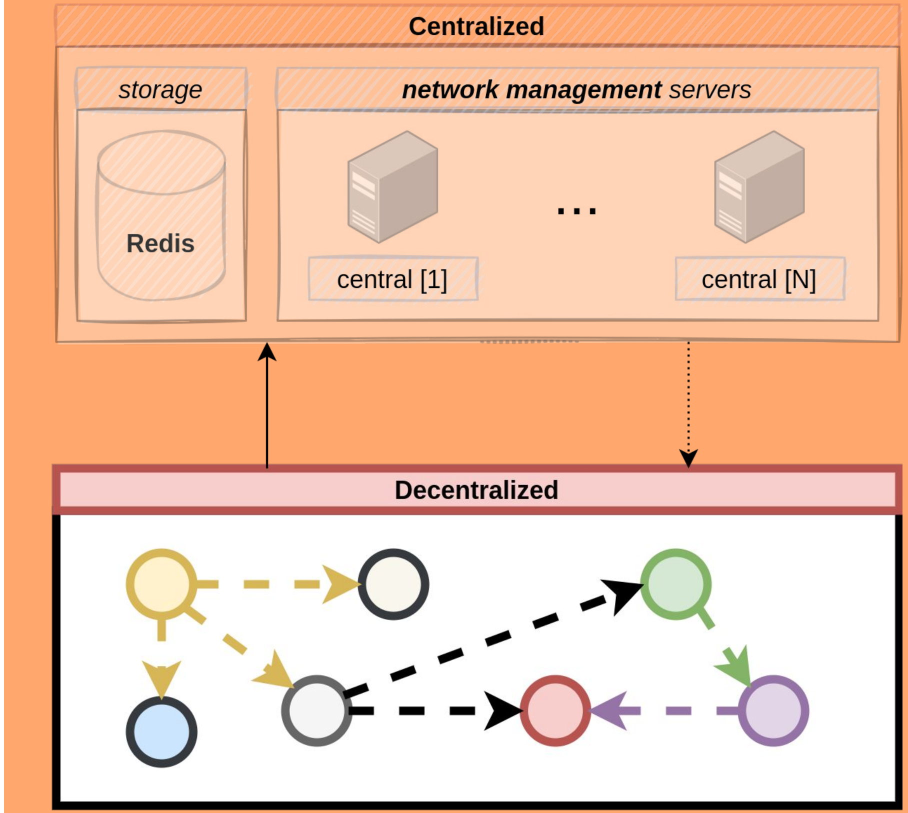


Decentralized portion

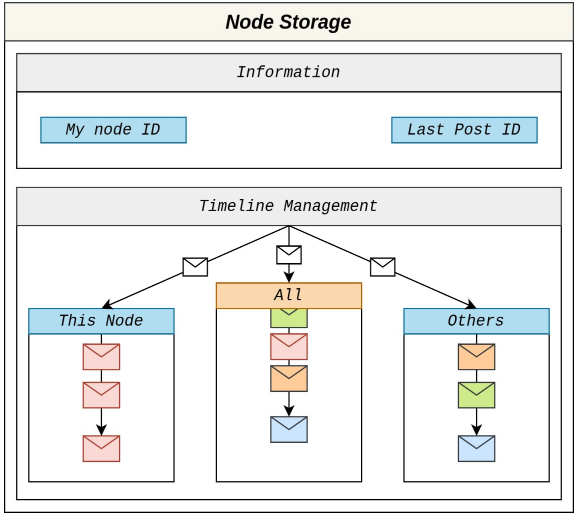
It’s used for:

a. Propagation of posts
b. Timeline management

“Peer-to-Peer portion of the network”



Node information
Local storage



Use cases Services

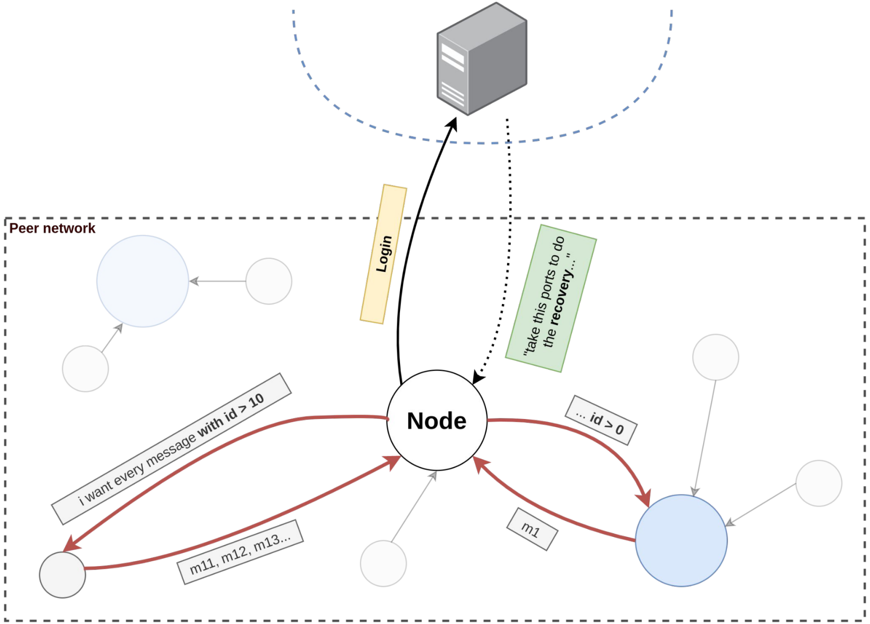
1. ***Register***
2. ***Login***
   a. *gets the **elected candidates** ports for subscriptions and recovery*
3. ***Logout***
   a. notifies central server that the node is **no longer online**
4. ***Subscription***
   a. node election
5. ***Timeline management***
   a. publish, forward, recover and post ordering

Timeline Management

- **Publish** a post

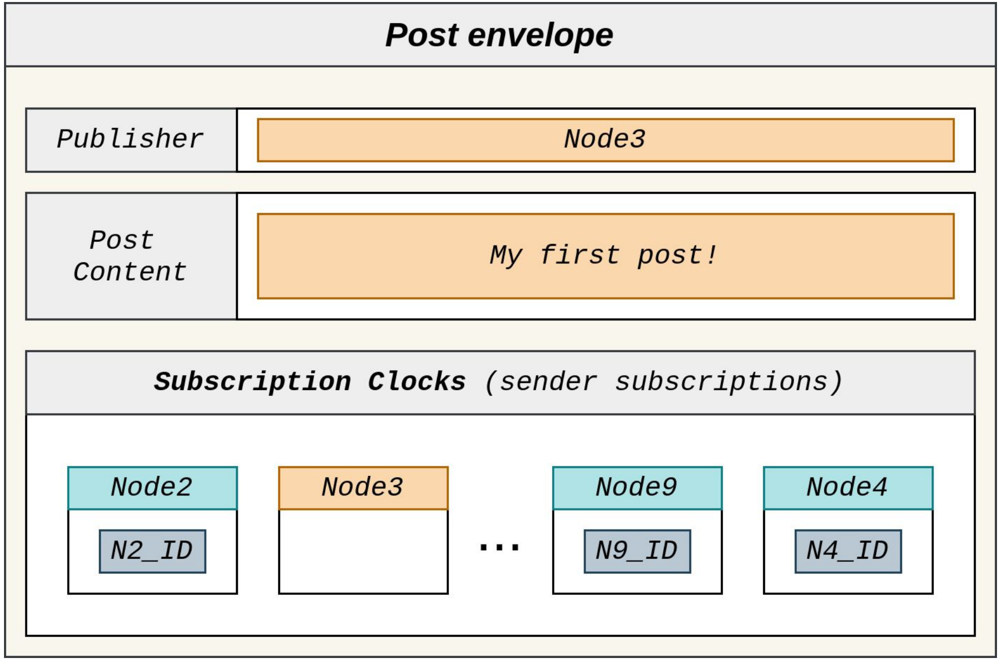
- **Reception** and **Forwarding**

- **Timeline Recovery** (for all subscriptions)
  1- Always after Login
  2- Recovers only from the last message of each subscription
  3- Some messages can be lost due to ephemerality


Peer network
Login
"take this ports to do the **recovery**..."
Node
i want every message **with id > 10**
m11, m12, m13...
... id > 0
m1

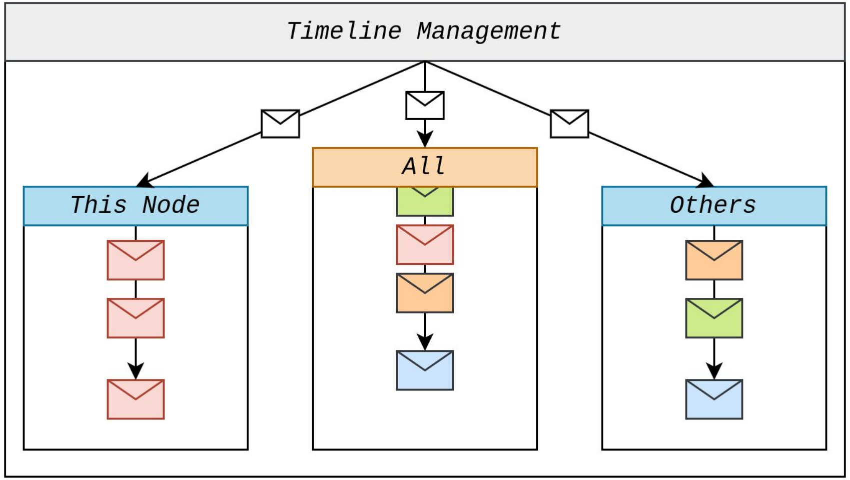
Post structure

- **Publisher** used to identify who sent the post
- **Post Content** that contains the message that the sender wants to deliver
- **Subscription Clocks** are important for ordering the messages



Message ordering

- We use the vector clock and order the posts **using the shared vector id's**
- Posts that are **concurrent** are distinguished using the **id of the sender**
- This solution can generate many paradoxical situations witch we didn't found a solution.



Architecture **overview**

Java

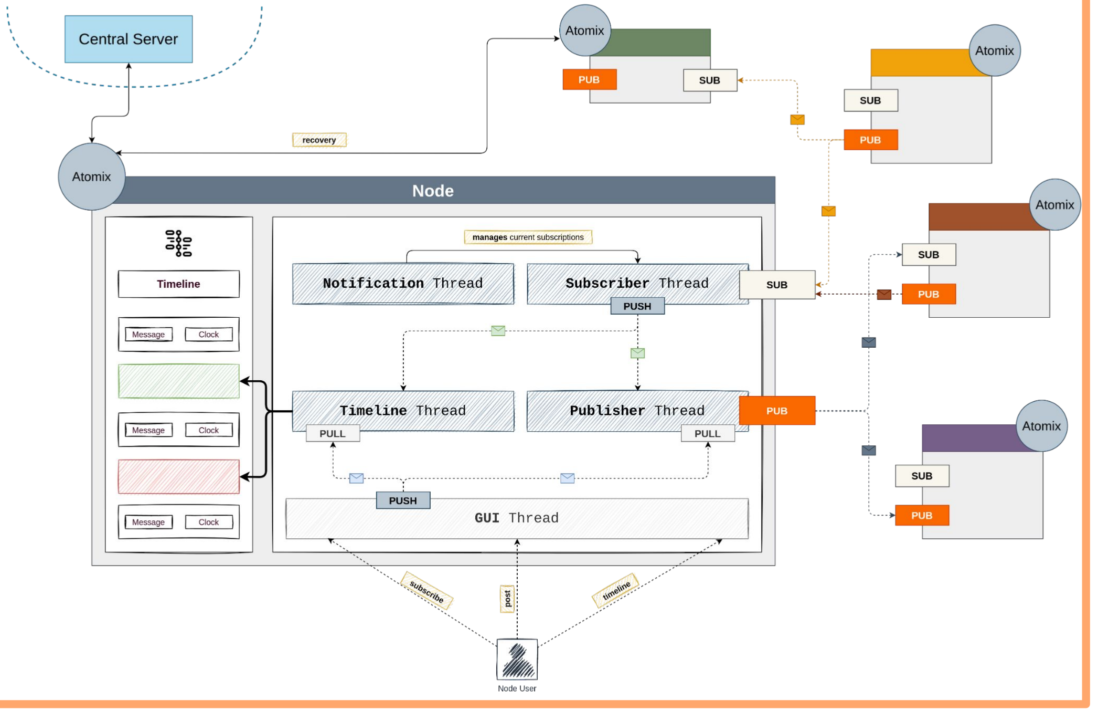
ZeroMQ

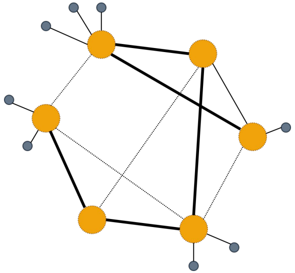
Pub-Sub pattern

Push-Pull pattern

Atomix

Event-driven services





CAP Theorem

Model assumptions

Conclusion

A84462 | Alexandre Miranda
A84961 | Alexandre Ferreira
A85227 | João Azevedo
A85315 | Miguel Cardoso
A85729 | Paulo Araújo