Lumiere: Making Optimal BFT for Partial Synchrony Practical

ANDREW LEWIS-PYE, DAHLIA MALKHI, ODED NAOR, KARTIK NAYAK

Henry Chou
Varun Ringnekar
Chris Ruan
Shaokang Xie
Yubo Bai

Background

Introduction to core concepts

Partially Synchronous BFT model

The Lumiere protocol operates on a Partially Synchronous Byzantine Fault Tolerant model

3f + 1 replicas

f faults

f_a Actual Faults

GST (Global Stabilisation Time)

Δ Known network delay

A message sent at time ***t*** must arrive by time **max{GST, t } + Δ**.

Synchronized Clocks

All the nodes in the system have a local clock ***lc***

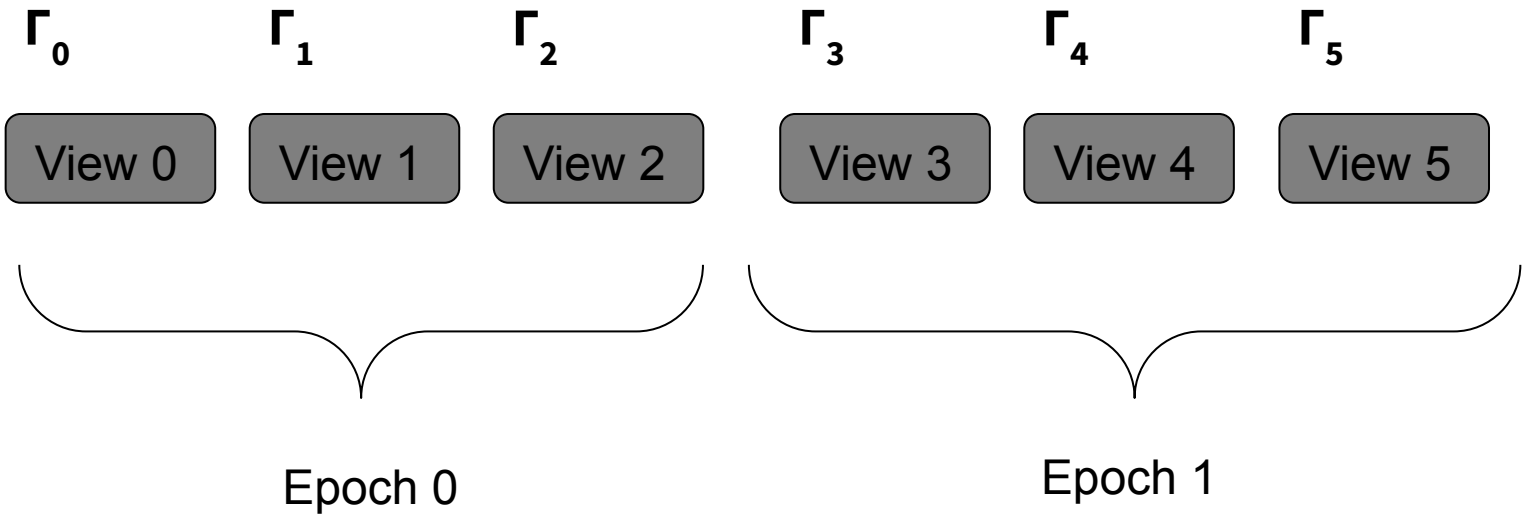
Lumiere tries to keep these clocks synchronized so that at least ***f+1*** clocks are synchronized closely together

All the nodes know which view will have which leader ***lead(v)***

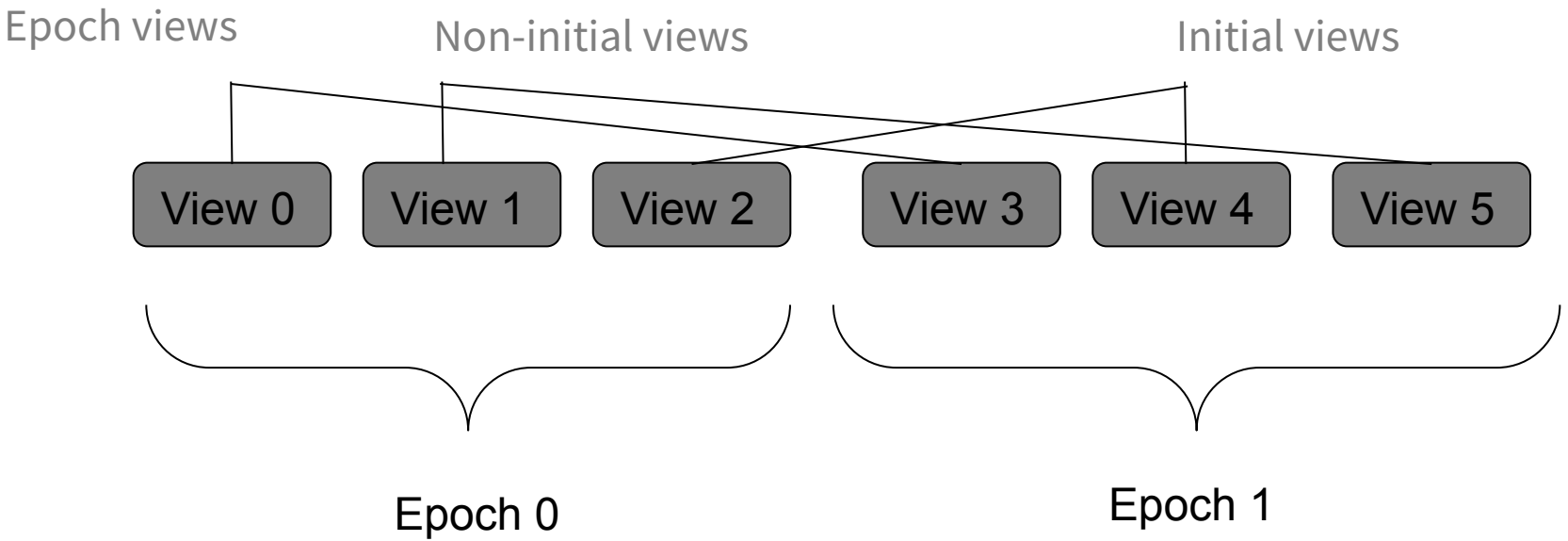
All the nodes in the system know when a view should reach consensus, this time is also called the view timer c_v

All the nodes also know when the next view is supposed to start Γ_v

Epochs


Γ0
Γ1
Γ2
Γ3
Γ4
Γ5
View 0
View 1
View 2
View 3
View 4
View 5
Epoch 0
Epoch 1

Different Types of Views



View Synchronisation

Making sure that all honest processors in a BFT system move between views in a coordinated and consistent manner, even in the presence of byzantine failures

Epoch Certificate(EC): Can be formed by any honest replica, when it receives **2f+1 epoch view messages** from other replicas. $O(n^2)$ communication. Formed to enter a new epoch, ensured heavy synchronization.

Quorum Certificate(QC) : Is formed by the leader when consensus is reached in a view i.e. **2f+1** replicas vote on a block. Created in initial views to enter non initial views

View Certificate(VC) : Is formed by the leader of the next view when enough replicas (**f+1**) send a **view message** to that leader to enter its view. Created to enter initial views

Overview

Quick preview of the upcoming content

Liveness

- **RECAP**: guarantee to make progress, even though faulty node exists
  - Timeout - change view with faulty leader/replica (Fever)
  - Epoch-based Design (LP22)

Optimal Responsiveness

- **RECAP**: make progress at the speed of the network
  - Quorum Certificates (HotStuff) and Clock-bumping (Fever)
  - Epoch-based Design: Light & Heavy Synchronization (LP22)

Fever protocol: Key characteristics

- Makes use of **optimistic responsiveness** - scales with actual network conditions rather than worst case assumptions, clocks of processors can be **bumped forward**
- View synchronization protocol that operates under a **partial synchrony model**
    - **Partial clock synchronization** after GST - clocks may drift before GST, but processor clocks are synchronized after network stabilization
- Designed to handle **frequent view changes** efficiently
- **Initial and non-initial views** - The leader for view v is (*processor (v/2) mod n),* initial views have time based clock synchronization, optimistic responsiveness is achieved in non-initial views with QC
- **Fundamental idea of Fever - Honest Gap**

Fever protocol: Honest Gap

- Measures the disparity in clock synchronization between the most advanced honest processors
- Definition: the i-th honest gap at time C, denoted as $hg_{i,C}$: $hg_{i,C} := lc(p_{1,C}, C) - lc(p_{i,C}, C)$
- Gap needs to stay within bound to allow the processors to stay synchronized
- Despite processors bumping clocks forward, two properties are guaranteed:
  - $\mathsf{hg}_{f+1,t} \leq \Gamma$ for all $t \geq 0$.
  - If $\mathsf{hg}_{f+1,t} \leq \Gamma$ at $t \geq$ GST which is the first time an honest processor enters the view v with honest leader, then the leader will produce a QC.

- **Non-standard assumption**
  - Clocks assumed to be synchronized within Γ at the beginning for Fever protocol

Drawback of PBFT

PBFT Properties that could be improved with Lumiere

PBFT

- View-change - heavy communication exchange
  - All-to-All broadcast ($O(n^2)$) for each view-change
  - When suffering from f consecutive faulty leader, at worst $O((f+1)*n^2)$ -> $O(n^3)$
- Not optimistically responsive
- Conclusion
  - practical for smaller, more controlled environments, but limited scalability

Contribution of Lumiere:

1. Worst-case communication complexity $O(n^2)$
2. Improve latency - $O(n\Delta)$
3. The protocol is smoothly optimistically responsive
4. Eventual worst case communication complexity $O(f_a n+n)$

Overview of LP22

- Epoch-Based Structure
  - Time is divided into epochs, each consisting of f + 1 views.

- Quadratic Worst-Case Communication Complexity:
  - Communication complexity of O(n^2) worst-case.

Drawbacks of LP22

- Byzantine Leader Delays:
  - Even one Byzantine leader can cause a worst case delay, which is O(f + 1)Γ

- High Communication Cost:
  - Heavy synchronization incurs quadratic cost (O(n^2)), even without Byzantine faults.

Lumiere Design

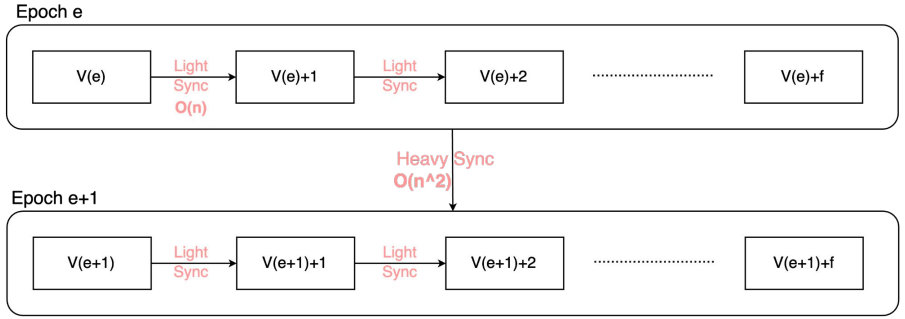
The Design of Lumiere

Lumiere

Key Contributions:

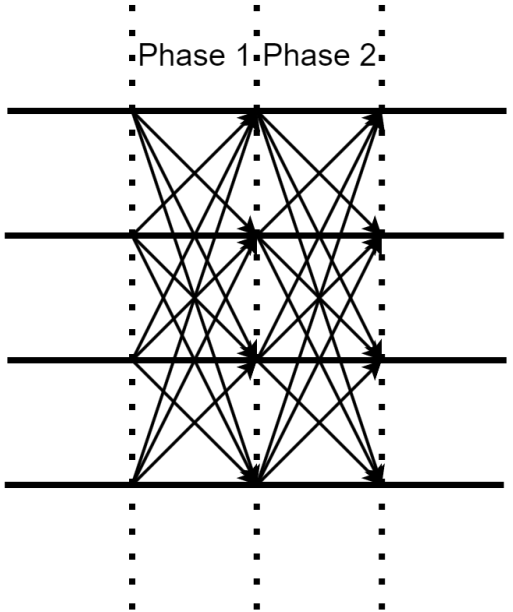
- **Latency Reduction**:
  LP22's view sync process suffers from O((f + 1)Γ) latency due to Byzantine leaders. Lumiere reduces this by combining **LP22's epoch structure** with **Fever's clock bumping**.
- **Communication Optimization**:
  Only a **constant-bounded number of heavy synchronizations (O(n^2))** occur after GST, lowering communication overhead.

Epoch Change in Lumiere

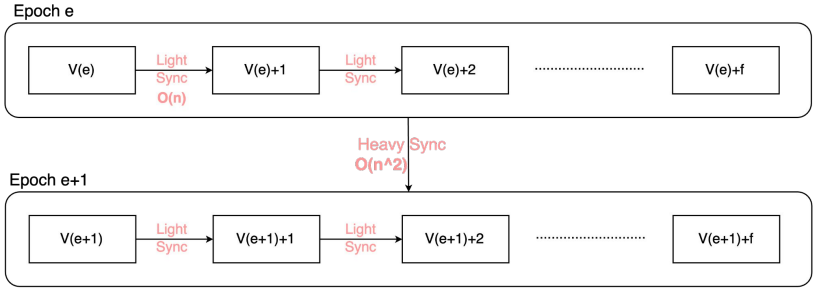


- **Heavy Sync** in Epoch change: **(O(n^2))**
  - Phase 1: Send "**epoch e**" msg to all replicas
  - Phase 2.1: Any honest replica combine **2f+1** "**epoch e**" msg as **EC**(Epoch Certificate), then broadcast EC
  - Phase 2.2: When sb see a **EC**, enter **epoch e**

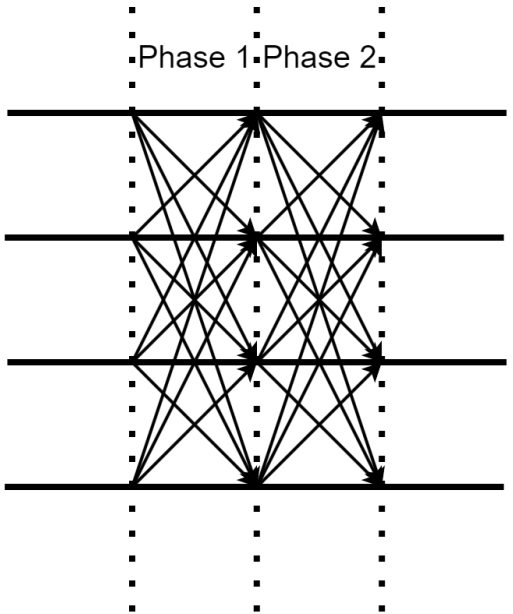
- **Clock Synchronization:** If a correct replica ***i*** enters epoch ***e*** at time ***t***, then all correct replicas will have entered an epoch **e′** ≥ **e** by time **max{t, GST} + Δ**.
  - Each replica will enter epoch e in a time interval **Δ**.



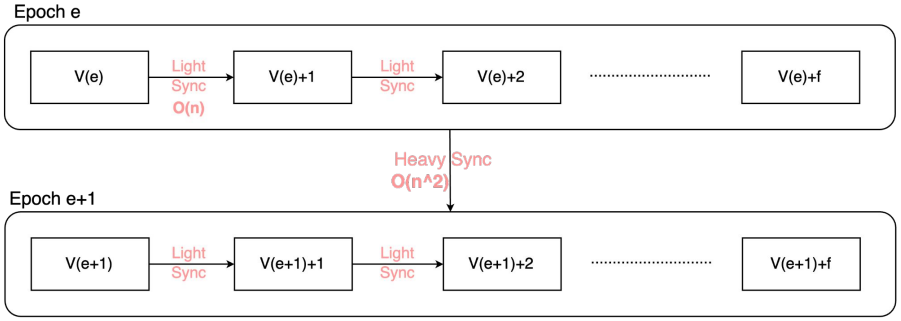
Epoch Change in Lumiere



- **Worst-case Message Complexity: O(n^2)**
  - **Heavy sync** in each Epoch Change
- **Heavy Sync** is **not always necessary** after **GST**, once it satisfies **"success criterion" (see QCs).**
  - **At least f+1 honest replicas have synchronized clock.**
- **After GST, the Message Complexity: O(f_an+n)**
  - In each epoch, **f_a** faulty leader will cause **f_an** message complexity, and **n** for a true leader

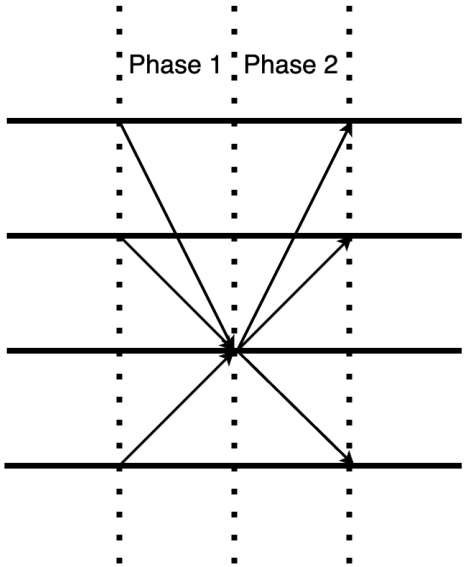


View Change in Lumiere



- **Whenever receiving QC, it would perform “Clock Bumping”.**

- **Light Sync** in View change **(O(n))**
  - Phase 1: Send “**view v**” msg to the certain leader
  - Phase 2.1: Leader combining **f+1** “**view v**” msg as VC(View Certificate)
  - Phase 2.2: When sb see a **VC**, enter view v



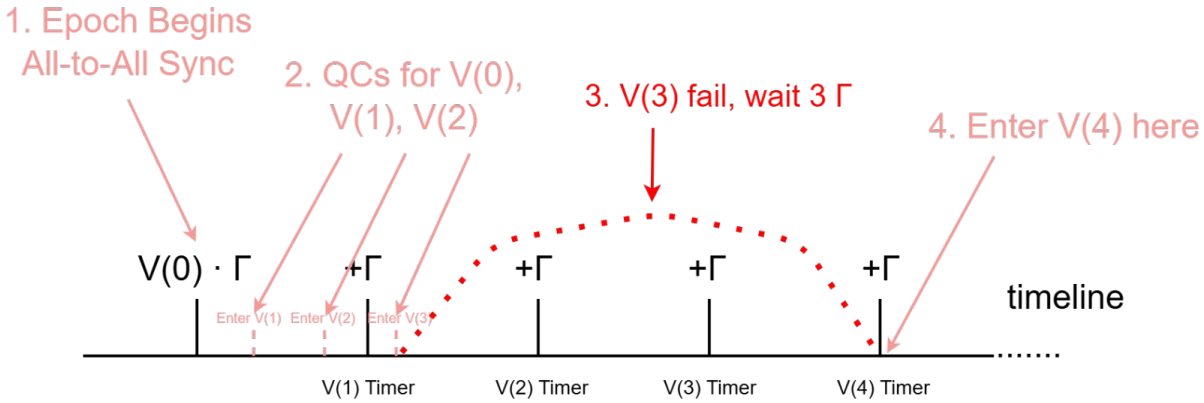
Clock Bumping (Forwarding) overview

- Purpose of forwarding clocks
  - Ensures that nodes can adjust their clocks to stay in sync with the rest of the network despite potential network delays or Byzantine behavior
- Occurs when receiving a QC in a non initial view or VC for the initial view
- Importance of clock forwarding
  - Prevent nodes with delayed clocks from being stuck in old views

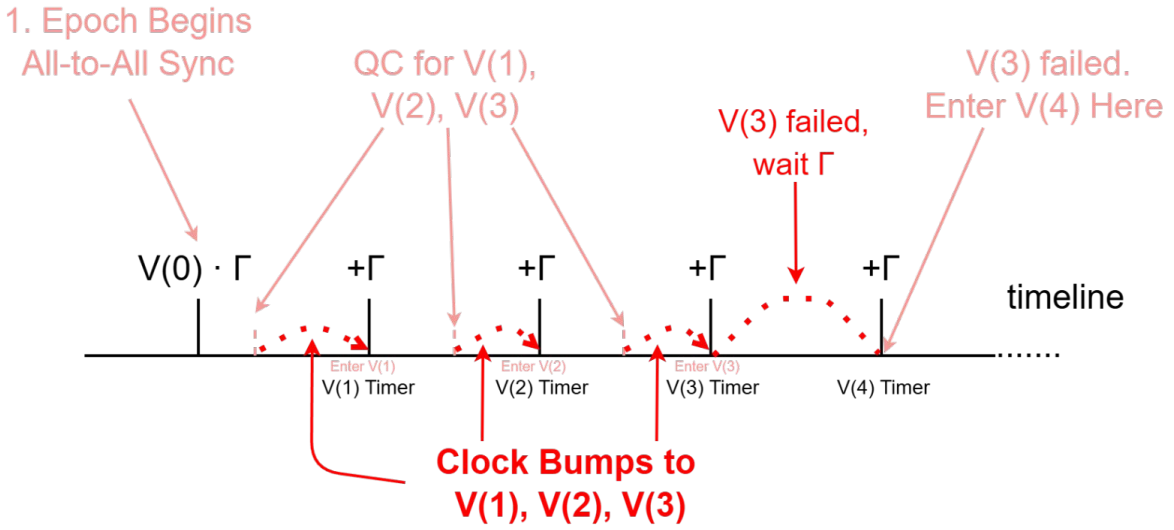
Clock Bumping (Clock Forwarding) in Lumiere:

Γ: Expectation time for a view

LP22:

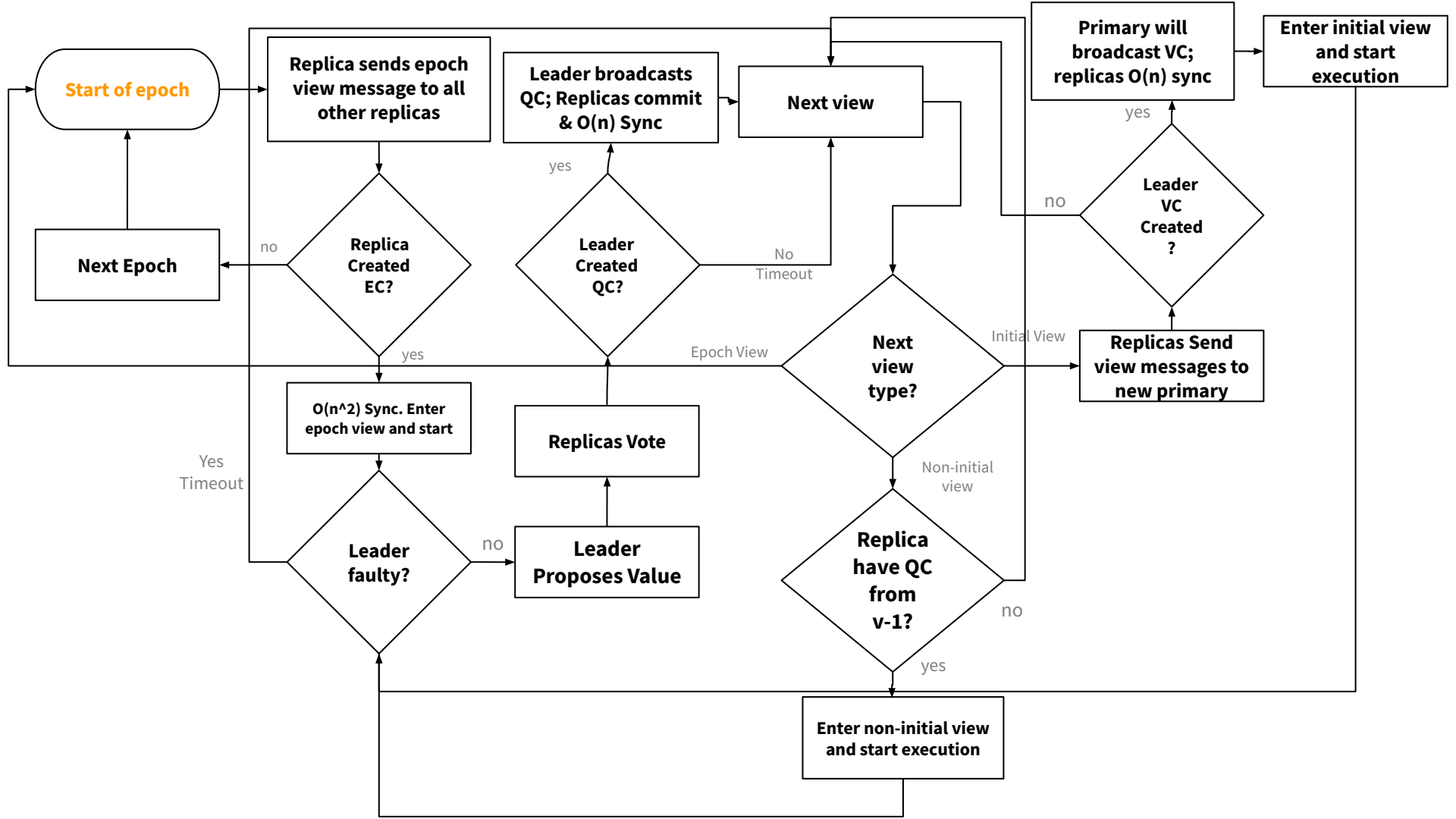


Lumiere:



The Flow

Flowchart of the protocol


Start of epoch
Replica sends epoch view message to all other replicas
Leader broadcasts QC; Replicas commit & O(n) Sync
Next view
Primary will broadcast VC; replicas O(n) sync
Enter initial view and start execution
yes
Next Epoch
no
Replica Created EC?
yes
Leader Created QC?
yes
No Timeout
no
Leader VC Created ?
Epoch View
Next view type?
Initial View
Replicas Send view messages to new primary
O(n^2) Sync. Enter epoch view and start
Yes Timeout
Leader faulty?
no
Leader Proposes Value
Replicas Vote
Non-initial view
Replica have QC from v-1?
no
yes
Enter non-initial view and start execution

Thank You