CONSISTENT, DISTRIBUTED Elixir

CHRIS KEATHLEY / @CHRISKEATHLEY / C@KEATHLEY.IO
I HAVE A PROBLEM...
THAT'S RIGHT,
SAY IT WITH ME

"PHOENIX IS NOT YOUR ..."
“PHOENIX IS NOT YOUR APPLICATION”
I DON’T KNOW WHAT MY APPLICATION IS ANY MORE
ELIXIR IS AWESOME
PROBLEM:

“WE NEED TO LIMIT ACCESS TO AN EXTERNAL RESOURCE”
SOLUTION:

“LET'S JUST USE PROCESSES!”
GLOBAL LOCK
GLOBAL LOCK

:unlocked
GLOBAL LOCK

CLIENT

:unlocked
GLOBAL LOCK

CLIENT: lock

:unlocked
GLOBAL LOCK

CLIENT

:lock :unlocked
GLOBAL LOCK

CLIENT

:locked
GLOBAL LOCK

CLIENT

:ok

:locked
GLOBAL LOCK

CLIENT :ok

:locked
GLOBAL LOCK

CLIENT

:locked
GLOBAL LOCK

CLIENT

:unlock

:locked
GLOBAL LOCK

CLIENT

:unlock

:locked
GLOBAL LOCK

CLIENT

:locked
GLOBAL LOCK

CLIENT

:unlocked
GLOBAL LOCK

CLIENT

:ok :unlocked
GLOBAL LOCK

CLIENT: ok

:unlocked
GLOBAL LOCK

CLIENT

:unlocked
GLOBAL LOCK

CLIENT

:unlocked
GLOBAL LOCK

CLIENT

CLIENT

:unlocked
GLOBAL LOCK

CLIENT

CLIENT

:unlocked
GLOBAL LOCK

CLIENT

CLIENT

:locked
GLOBAL LOCK

:locked
GLOBAL LOCK

CLIENT

CLIENT

:locked
GLOBAL LOCK

CLIENT

CLIENT

:locked
GLOBAL LOCK

CLIENT

CLIENT

:locked
GLOBAL LOCK

CLIENT

CLIENT

:locked
GLOBAL LOCK

CLIENT

CLIENT

:locked
GLOBAL LOCK

CLIENT

CLIENT

:locked
GLOBAL LOCK

CLIENT

CLIENT

:locked
GLOBAL LOCK

CLIENT

CLIENT

:locked
GLOBAL LOCK

CLIENT

CLIENT

:locked
GLOBAL LOCK

:unlocked
GLOBAL LOCK

CLIENT

CLIENT

:unlocked
GLOBAL LOCK

CLIENT

CLIENT

:unlocked
GLOBAL LOCK

CLIENT

CLIENT

:locked
GLOBAL LOCK

CLIENT

CLIENT

:locked
GLOBAL LOCK

CLIENT

CLIENT

:locked
defmodule Demo.Lock do
  use GenServer
end
defmodule Demo.Lock do
  use GenServer

  def init(:ok) do
    state = {:unlocked, nil}
    {:ok, state}
  end
end
defmodule Demo.Lock do
  use GenServer

  def init(:ok) do
    state = {:unlocked, nil}
    {:ok, state}
  end

  def handle_call({:lock, client}, _from, {:unlocked, nil}) do
    {:reply, :ok, {:locked, client}}
  end
end
defmodule Demo.Lock do
  use GenServer

  def init(:ok) do
    state = {:unlocked, nil}
    {:ok, state}
  end

  def handle_call({:lock, client}, _from, {:unlocked, nil}) do
    {:reply, :ok, {:locked, client}}
  end

  def handle_call({:lock, client}, _from, {:locked, other_client}) do
    {:reply, :error, {:locked, other_client}}
  end
end
defmodule Demo.Lock do
  use GenServer

  def init(:ok) do
    state = {:unlocked, nil}
    {:ok, state}
  end

  def handle_call({:lock, client}, _from, {:unlocked, nil}) do
    {:reply, :ok, {:locked, client}}
  end

  def handle_call({:lock, client}, _from, {:locked, other_client}) do
    {:reply, :error, {:locked, other_client}}
  end

  def handle_call({:unlock, client}, _from, {:locked, client}) do
    {:reply, :ok, {:unlocked, nil}}
  end

  def handle_call({:unlock, client}, _from, {:locked, other_client}) do
    {:reply, :error, {:locked, other_client}}
  end
end
GLOBAL LOCK
PROBLEM:

“WE NEED TO RUN MULTIPLE NODES”
MULTIPLE NODES
MULTIPLE NODES
MULTIPLE NODES

NODE

NODE
MULTIPLE NODES

CLIENT

NODE

CLIENT

NODE
MULTIPLE NODES

CLIENT

NODE

CLIENT

NODE
MULTIPLE NODES

CLIENT

NODE

CLIENT

NODE
MULTIPLE NODES

CLIENT

NODE

NODE
MULTIPLE NODES

CLIENT

NODE

CLIENT

NODE
MULTIPLE NODES

CLIENT

CLIENT
MULTIPLE NODES

CLIENT

NODE

CLIENT

NODE
MULTIPLE NODES

This is bad
SOLUTION:

“LET'S JUST USE A GLOBAL PROCESS”
GenServer.start_link({Lock, :ok, name: Lock})
GenServer.start_link(Lock, :ok, name: {:global, Lock})
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

REMOVE THIS

CLIENT

NODE
SOLUTION: GLOBAL PROCESS
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE
Solution: Global Process

What if this goes away?
SOLUTION: GLOBAL PROCESS

CLIENT

CLIENT

NODE
SOLUTION: GLOBAL PROCESS
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

START A NEW LOCK PROCESS
SOLUTION: GLOBAL PROCESS

START A NEW LOCK PROCESS
SOLUTION: GLOBAL PROCESS

CLIENT

CLIENT

NODE
SOLUTION: GLOBAL PROCESS
SOLUTION: GLOBAL PROCESS

CLIENT

NODE
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT
PROBLEM:

“What if the node isn’t really down?”
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS
SOLUTION: GLOBAL PROCESS

CLIENT

PARTITION

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS

CLIENT

GUESS IT MUST BE DOWN

CLIENT

NODE

NODE
SOLUTION: GLOBAL PROCESS

CLIENT

START A NEW LOCK PROCESS

CLIENT

NODE

NODE
SOLUTION: GLOBAL PROCESS

START A NEW LOCK PROCESS
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS
SOLUTION: GLOBAL PROCESS
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

THIS LOCK STILL EXISTS
SOLUTION: GLOBAL PROCESS
SOLUTION: GLOBAL PROCESS
SOLUTION: GLOBAL PROCESS
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE

NODE
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS

THIS IS BAD
PROBLEM:

“What happens when the partition heals?”
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS

CLIENT

HEALS

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE
SOLUTION: GLOBAL PROCESS

WHO SHOULD WIN?
SOLUTION: GLOBAL PROCESS

CLIENT

NODE

CLIENT

NODE

THIS IS BAD
SOLUTION:

“LET'S JUST USE THE DATABASE”
SOLUTION: DATABASES

CLIENT

NODE

CLIENT

NODE
SOLUTION: DATABASES

CLIENT

NODE

CLIENT

NODE
SOLUTION: DATABASES

CLIENT

NODE

NODE

REDIS

CLIENT
SOLUTION:

“LET'S HAVE A WAY TO CONSISTENTLY MANAGE STATE IN ELIXIR”
When a network is partitioned, you can either be available or consistent.
“EVERY REQUEST RECEIVES A RESPONSE WITHOUT GUARANTEE THAT IT CONTAINS THE MOST RECENT WRITE”
"EVERY READ RECEIVES THE MOST RECENT WRITE OR IT ERRORS"
AP
AVAILABLE DURING PARTITIONS

CP
CONSISTENT DURING PARTITIONS
AVAILABLE
PROBLEM:

“WE NEED TO KEEP TRACK OF COUNTS”
COUNTERS

CLIENT

NODE

CLIENT

NODE
COUNTERS

Client

+θ

Node

Node

+θ
COUNTERS

STORE ADDITIONS

CLIENT

NODE

+θ

CLIENT

NODE

+θ
COUNTERS

CLIENT

+θ

NODE

+θ

CLIENT

NODE
COUNTERS

CLIENT

+2

NODE

+θ

CLIENT

+2

NODE

+θ
COUNTERS

CLIENT

+2

NODE

= θ

CLIENT

+2

NODE

= θ
COUNTERS

CLIENT

NODE

+0

CLIENT

NODE

+0

+2
COUNTERS

CLIENT

NODE

+0

+2

+0

+2
COUNTERS

CLIENT

NODE

+2

+0

CLIENT

NODE

+0

+2
COUNTERS

CLIENT

+0
+2

NODE

CLIENT

+0
+2

NODE
COUNTERS

CLIENT

:read

NODE

+0
+2

NODE

+0
+2
COUNTERS

CLIENT

NODE
+0
+2

:read

CLIENT

NODE
+0
+2
COUNTERS

CLIENT

NODE
+0
+2

CLIENT

NODE
+0
+2
COUNTERS

CLIENT 2

NODE
+2
+0

CLIENT

NODE
+2
+0
COUNTERS

CLIENT

NODE
+0
+2

CLIENT

NODE
+0
+2
COUNTERS

CLIENT

+0
+2

NODE

CLIENT

+0
+2

NODE

---
COUNTERS

CLIENT

+3

NODE

+0
+2

CLIENT

+3

NODE

+0
+2
COUNTERS

CLIENT

NODE
+0
+2

Cliente CLIENT

NODE
+0
+2

NODE
+3
COUNTERS

CLIENT

NODE
+0
+2

CLIENT

NODE
+0
+2
+3
COUNTERS

Client

Node

+0
+2

Node

+3
+0
+2
+3
COUNTERS

Client

Node

+0
+2

Node

+0
+2
+3
COUNTERS

CLIENT

:read

NODE

+0
+2

NODE

+0
+2
+3
COUNTERS

CLIENT

NODE
+0
+2

:read

CLIENT

NODE
+0
+2
+3
COUNTERS

CLIENT

NODE
+0
+2

CLIENT

NODE
+0
+2
+3
COUNTERS

CLIENT

NODE

+0
+2

CLIENT

NODE

+0
+2
+3
COUNTERS

CLIENT +5

NODE
+0
+2

CLIENT

NODE
+0
+2
+3
COUNTERS

CLIENT

NODE
+0
+2
+5

CLIENT

NODE
+0
+2
+3
COUNTERS

CLIENT

NODE
+0
+2
+5

CLIENT

NODE
+0
+2
+3
COUNTERS

CLIENT

INCONSISTENT

NODE
+0
+2
+5

CLIENT

NODE
+0
+2
+3
COUNTERS

CLIENT

NODE
+0
+2
+5

CLIENT

NODE
+0
+2
+3
COUNTERS

CLIENT

NODE

+0
+2
+5

CLIENT

NODE

+0
+2
+3
COUNTERS

CLIENT

+0
+2
+5

CLIENT

+0
+2
+3

NODE

NODE
COUNTERS

NODE
+0
+2
+5
+3

CLIENT

NODE
+0
+2
+3
+5
SOMETIMES YOU’RE GOING TO BE WRONG

(AND THAT’S OK)
PHOENIX
PRESENCE
SOME PROBLEMS NEED CONSISTENCY

DISTRIBUTED LOCKING

DATABASES

DISTRIBUTED SCHEDULING AND COORDINATION

CONFIGURATION AND METADATA STORAGE

transactions
CONSISTENT
PARTITIONS IN CONSISTENT SYSTEMS
PARTITIONS IN CONSISTENT SYSTEMS

- NODE
- NODE
- NODE

The diagram illustrates the concept of partitions in consistent systems, showing how nodes interact and maintain consistency even during partition failures.
PARTITIONS IN CONSISTENT SYSTEMS
PARTITIONS IN CONSISTENT SYSTEMS

![Diagram of partitions in consistent systems](image-url)
PARTITIONS IN CONSISTENT SYSTEMS
The Part-Time Parliament
Leslie Lamport

This article appeared in ACM Transactions on Computer Systems 16, 2 (May 1998), 133-169. Minor corrections were made on 29 August 2000.
Leslie Lamport
PAXOS (BUT SIMPLER)

Paxos Made Simple

Leslie Lamport.

01 Nov 2001
In Search of an Understandable Consensus Algorithm (Extended Version)

Diego Ongaro and John Ousterhout
Stanford University

Abstract
Raft is a consensus algorithm for managing a replicated log. It produces a result equivalent to Paxos, and it is as efficient as Paxos, but its structure is different from Paxos: this makes Raft more understandable than Paxos and also provides a better foundation for building practical systems. In order to enhance understandability, Raft separates the key elements of consensus, such as leader election, log replication, and safety, and it enforces a stronger degree of coherency to reduce the number of states that must be considered. Results from a user study demonstrate that Raft is easier for students to learn than Paxos. Raft also includes a new mechanism for changing the cluster membership, which uses overlapping majorities to guarantee safety.

1 Introduction
Consensus algorithms allow a collection of machines to work as a coherent group that can survive the failures of some of its members. Because of this, they play a key role in building reliable large-scale software systems.
RAFT (This time a lot simpler)
PROBLEM:

“We need to limit access to an external resource”
defmodule Demo.Lock do
  use GenServer

  def init(:ok) do
    {:ok, {:unlocked, nil}}
  end

  def handle_call({:lock, client}, _from, {:unlocked, nil}) do
    {:reply, :ok, {:locked, client}}
  end

  def handle_call({:lock, client}, _from, {:locked, other_client}) do
    {:reply, :error, {:locked, other_client}}
  end

  def handle_call({:unlock, client}, _from, {:locked, client}) do
    {:reply, :ok, {:unlocked, nil}}
  end

  def handle_call({:unlock, client}, _from, {:locked, other_client}) do
    {:reply, :error, {:locked, other_client}}
  end
end
defmodule Demo.Lock do
  use Raft.StateMachine

  def init(:ok) do
    {:ok, {:unlocked, nil}}
  end

  def handle_call({:lock, client}, _from, {:unlocked, nil}) do
    {:reply, :ok, {:locked, client}}
  end

  def handle_call({:lock, client}, _from, {:locked, other_client}) do
    {:reply, :error, {:locked, other_client}}
  end

  def handle_call({:unlock, client}, _from, {:locked, client}) do
    {:reply, :ok, {:unlocked, nil}}
  end

  def handle_call({:unlock, client}, _from, {:locked, other_client}) do
    {:reply, :error, {:locked, other_client}}
  end
end
defmodule Demo.Lock do
  use Raft.StateMachine

  def init(:ok) do
    {:unlocked, nil}
  end

  def handle_call({:lock, client}, _from, {:unlocked, nil}) do
    {:reply, :ok, {:locked, client}}
  end
  def handle_call({:lock, client}, _from, {:locked, other_client}) do
    {:reply, :error, {:locked, other_client}}
  end

  def handle_call({:unlock, client}, _from, {:locked, client}) do
    {:reply, :ok, {:unlocked, nil}}
  end
  def handle_call({:unlock, client}, _from, {:locked, other_client}) do
    {:reply, :error, {:locked, other_client}}
  end
end
defmodule Demo.Lock do
  use Raft.StateMachine

  def init(:ok) do
    {:unlocked, nil}
  end

  def handle_write({...lock, client}, _from, {:unlocked, nil}) do
    {:reply, :ok, {:locked, client}}
  end

  def handle_call({...lock, client}, _from, {:locked, other_client}) do
    {:reply, :error, {:locked, other_client}}
  end

  def handle_call({...unlock, client}, _from, {:locked, client}) do
    {:reply, :ok, {:unlocked, nil}}
  end

  def handle_call({...unlock, client}, _from, {:locked, other_client}) do
    {:reply, :error, {:locked, other_client}}
  end
end
defmodule Demo.Lock do
  use Raft.StateMachine

  def init(:ok) do
    {:unlocked, nil}
  end

  def handle_write({:lock, client}, {:unlocked, nil}) do
    {:reply, :ok, {:locked, client}}
  end

  def handle_call({:lock, client}, _from, {:locked, other_client}) do
    {:reply, :error, {:locked, other_client}}
  end

  def handle_call({:unlock, client}, _from, {:locked, client}) do
    {:reply, :ok, {:unlocked, nil}}
  end

  def handle_call({:unlock, client}, _from, {:locked, other_client}) do
    {:reply, :error, {:locked, other_client}}
  end
end
defmodule Demo.Lock do
  use Raft.StateMachine

  def init(:ok) do
    {:unlocked, nil}
  end

  def handle_write({:lock, client}, {:unlocked, nil}) do
    {:ok, {:locked, client}}
  end
  def handle_call({:lock, client}, _from, {:locked, other_client}) do
    {:reply, :error, {:locked, other_client}}
  end
  def handle_call({:unlock, client}, _from, {:locked, client}) do
    {:reply, :ok, {:unlocked, nil}}
  end
  def handle_call({:unlock, client}, _from, {:locked, other_client}) do
    {:reply, :error, {:locked, other_client}}
  end
end
defmodule Demo.Lock do
  use Raft.StateMachine

  def init(:ok) do
    {:unlocked, nil}
  end

  def handle_write({:lock, client}, {:unlocked, nil}) do
    {:ok, {:locked, client}}
  end
  def handle_write({:lock, client}, {:locked, other_client}) do
    {:error, {:locked, other_client}}
  end

  def handle_call({:unlock, client}, _from, {:locked, client}) do
    {:reply, :ok, {:unlocked, nil}}
  end
  def handle_call({:unlock, client}, _from, {:locked, other_client}) do
    {:reply, :error, {:locked, other_client}}
  end
end
defmodule Demo.Lock do
  use Raft.StateMachine

  def init(:ok) do
    {:unlocked, nil}
  end

  def handle_write({:lock, client}, {:unlocked, nil}) do
    {:ok, {:locked, client}}
  end
  def handle_write({:lock, client}, {:locked, other_client}) do
    {:error, {:locked, other_client}}
  end
  def handle_write({:unlock, client}, {:locked, client}) do
    {:ok, {:unlocked, nil}}
  end
  def handle_write({:unlock, client}, {:locked, other_client}) do
    {:error, {:locked, other_client}}
  end
end
Raft.start_peer(Demo.Lock, name: :s1)
Raft.start_peer(Demo.Lock, name: :s2)
Raft.start_peer(Demo.Lock, name: :s3)
Raft.start_peer(Demo.Lock, name: :s1)
Raft.start_peer(Demo.Lock, name: :s2)
Raft.start_peer(Demo.Lock, name: :s3)
Raft.set_configuration(:s1, [:s1, :s2, :s3])
Raft.start_peer(Demo.Lock, name: :s1)
Raft.start_peer(Demo.Lock, name: :s2)
Raft.start_peer(Demo.Lock, name: :s3)
Raft.set_configuration(:s1, [:s1, :s2, :s3])
:ok = Raft.write(:s1, {:lock, :s1})
Raft.start_peer(Demo.Lock, name: :s1)
Raft.start_peer(Demo.Lock, name: :s2)
Raft.start_peer(Demo.Lock, name: :s3)
Raft.set_configuration(:s1, [:s1, :s2, :s3])

:ok     = Raft.write(:s1, {:lock, :s1})
:error  = Raft.write(:s2, {:lock, :s2})
:error  = Raft.write(:s2, {:unlock, :s2})
Raft.start_peer(Demo.Lock, name: :s1)
Raft.start_peer(Demo.Lock, name: :s2)
Raft.start_peer(Demo.Lock, name: :s3)
Raft.set_configuration(:s1, [[:s1, :s2, :s3]])

:ok = Raft.write(:s1, {lock, :s1})
:error = Raft.write(:s2, {lock, :s2})
:error = Raft.write(:s2, {unlock, :s2})
:ok = Raft.write(:s1, {unlock, :s1})
Raft.start_peer(Demo.Lock, name: :s1)
Raft.start_peer(Demo.Lock, name: :s2)
Raft.start_peer(Demo.Lock, name: :s3)
Raft.set_configuration(:s1, [:s1, :s2, :s3])

:ok = Raft.write(:s1, {:lock, :s1})
:error = Raft.write(:s2, {:lock, :s2})
:error = Raft.write(:s2, {:unlock, :s2})
:ok = Raft.write(:s1, {:unlock, :s1})
:ok = Raft.write(:s2, {:lock, :s2})
Demo
HOW DOES THIS WORK?
CONSENSUS & LEADER ELECTION
CONSENSUS & LEADER ELECTION

CLIENT

LEADER

FOLLOWER

FOLLOWER
Consensus & Leader Election

Client

Leader

Follower

Follower
CONSENSUS & LEADER ELECTION

CLIENT

LEADER

FOLLOWER

FOLLOWER
Consensus & Leader Election

Replicated
CONSENSUS & LEADER ELECTION

Client

Leader

Follower

Follower
CONSENSUS & LEADER ELECTION
CONSENSUS & LEADER ELECTION

COMM ITTED

LEADER

FOLLOWER

FOLLOWER

CLIENT
CONSENSUS & LEADER ELECTION
CONSSENSUS & LEADER ELECTION

CLIENT

LEADER

FOLLOWER

FOLLOWER
CONSENSUS & LEADER ELECTION

CLIENT

LEADER

FOLLOWER

FOLLOWER
Consensus & Leader Election

Client

Leader

Heartbeats

Follower

Follower
CONSENSUS & LEADER ELECTION

CLIENT

LEADER

FOLLOWER

FOLLOWER
Consensus & Leader election

Client

Leader

Follower

Follower
CONSENSUS & LEADER ELECTION

CLIENT

LEADER

FOLLOWER

FOLLOWER
CONSENSUS & LEADER ELECTION

CLIENT

LEADER

FOLLOWER

FOLLOWER
CONSENSUS & LEADER ELECTION

LEADER

CLIENT

FOLLOWER

FOLLOWER
Consensus & Leader Election

Starts a new election
CONSENSUS & LEADER ELECTION

LEADER

CLIENT

FOLLOWER

FOLLOWER
CONSENSUS & LEADER ELECTION

LEADER

CLIENT

CANDIDATE

FOLLOWER
CONSENSUS & LEADER ELECTION

LEADER

CLIENT

CANDIDATE

FOLLOWER
CONSENSUS & LEADER ELECTION
CONSENSUS & LEADER ELECTION

LEADER

CLIENT

CANDIDATE

FOLLOWER
CONSENSUS & LEADER ELECTION

CLIENT

LEADER

FOLLOWER

LEADER
CONSENSUS & LEADER ELECTION

CLIENT

LEADER

LEADER

FOLLOWER
CONSENSUS & LEADER ELECTION

CLIENT

LEADER

LEADER

FOLLOWER
CONSENSUS & LEADER ELECTION

Leader -> Follower

Client

LEADER

FOLLOWER
CONSENSUS & LEADER ELECTION

CLIENT

LEADER

LEADER

FOLLOWER
CONSENSUS & LEADER ELECTION

Client

Leader

Follower
CONSENSUS & LEADER ELECTION

CLIENT

FOLLOWER

LEADER

FOLLOWER
CONSENSUS & LEADER ELECTION

CLIENT

FOLLOWER

LEADER

FOLLOWER
CONSENSUS & LEADER ELECTION

CLIENT

FOLLOWER

LEADER

FOLLOWER
CONSENSUS & LEADER ELECTION

CLIENT

LEADER

FOLLOWER

FOLLOWER
CONSENSUS & LEADER ELECTION

Client

Leader

Follower

Follower
CONSENSUS & LEADER ELECTION

CLIENT

LEADER

FOLLOWER

FOLLOWER
Consensus & Leader Election

- Leader
- Follower
- Client
CONSENSUS & LEADER ELECTION

CLIENT

FOLLOWER

LEADER

FOLLOWER
CONSENSUS & LEADER ELECTION

CLIENT

FOLLOWER

LEADER

FOLLOWER
CONSENSUS & LEADER ELECTION

CLIENT

LEADER

FOLLOWER

FOLLOWER
CONSENSUS & LEADER ELECTION

CLIENT

LEADER

FOLLOWER

FOLLOWER
Logs

Linearized Writes

Replicated across all nodes

RocksDB
TESTING
PROPERTY TESTS
JEPSEN
WHAT CAN WE DO NOW?
KV Store
Service Discovery
Distributed Lock Manager
Database Transactions
Configuration Management
LINKS

https://github.com/toniqsystems/raft
https://github.com/toniqsystems/raft_dem
https://toniq.sh
https://speakerdeck/keathley
TODO
MORE TESTING
DYNAMIC NODE CONFIGURATIONS
LMDB STORAGE ADAPTER
NOW WE CAN BUILD APPLICATIONS AND MANAGE STATE SAFELY WITHOUT NEEDING TO LEAVE ELIXIR.