Tim McGilchrist *@lambda_foo* *lambdafoo.com*

Raft

Implementing Distributed Consensus with Erlang

Outline

- Goals
- The consensus problem
- Outline RAFT algorithm
- Implementing in Erlang

Goals

What I want you to get out of this talk?

- Understand core ideas in RAFT
- Erlang / OTP as a tool for building systems
- Build your own implementation

Consensus

In a distributed system, agreement among multiple processes on a single data value, despite failures.

Once they reach a decision on a value, that decision is final.

Potential Use Case

- Configuration Management
- Distributed Transactions
- Distributed Lock Manager
- DNS and Resource Discovery

RAFT

Raft is a consensus algorithm that is designed to be easy to understand.

Goals

- Design for understandability
- Strong leader
- Practical to implement

Messages

- RAFT only needs 2 messages.
- RequestVote includes term
- AppendEntries includes term and log entries
- Term acts as a logical clock

States

3 states a node can be in.

Follower

Candidate

Leader

Leader

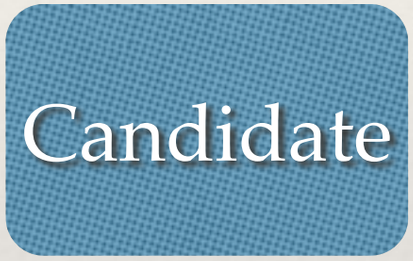
- Only a single leader within a cluster
- Receives commands from client
- Commits commands to the log

Follower

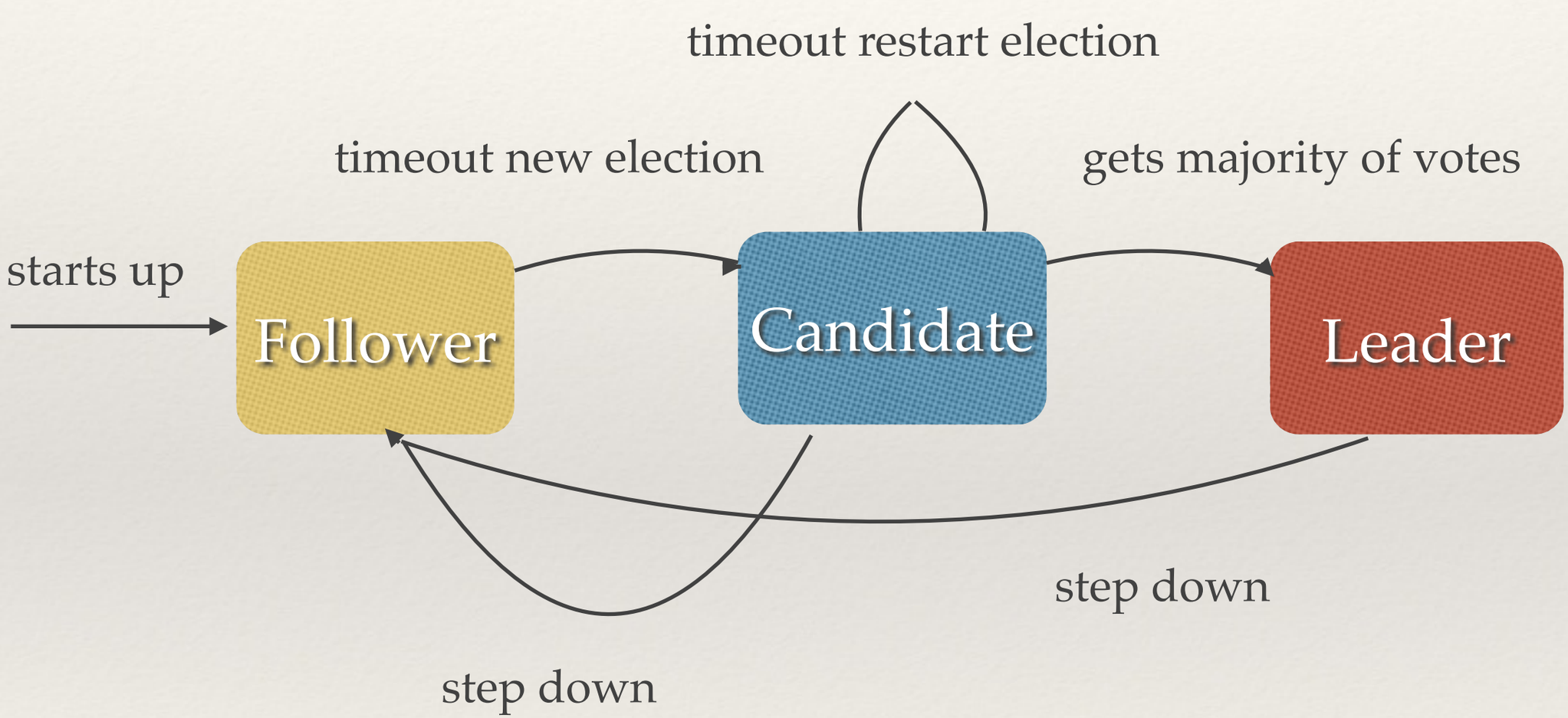
- Appends commands to log
- Votes for candidates
- Otherwise passive

Candidate

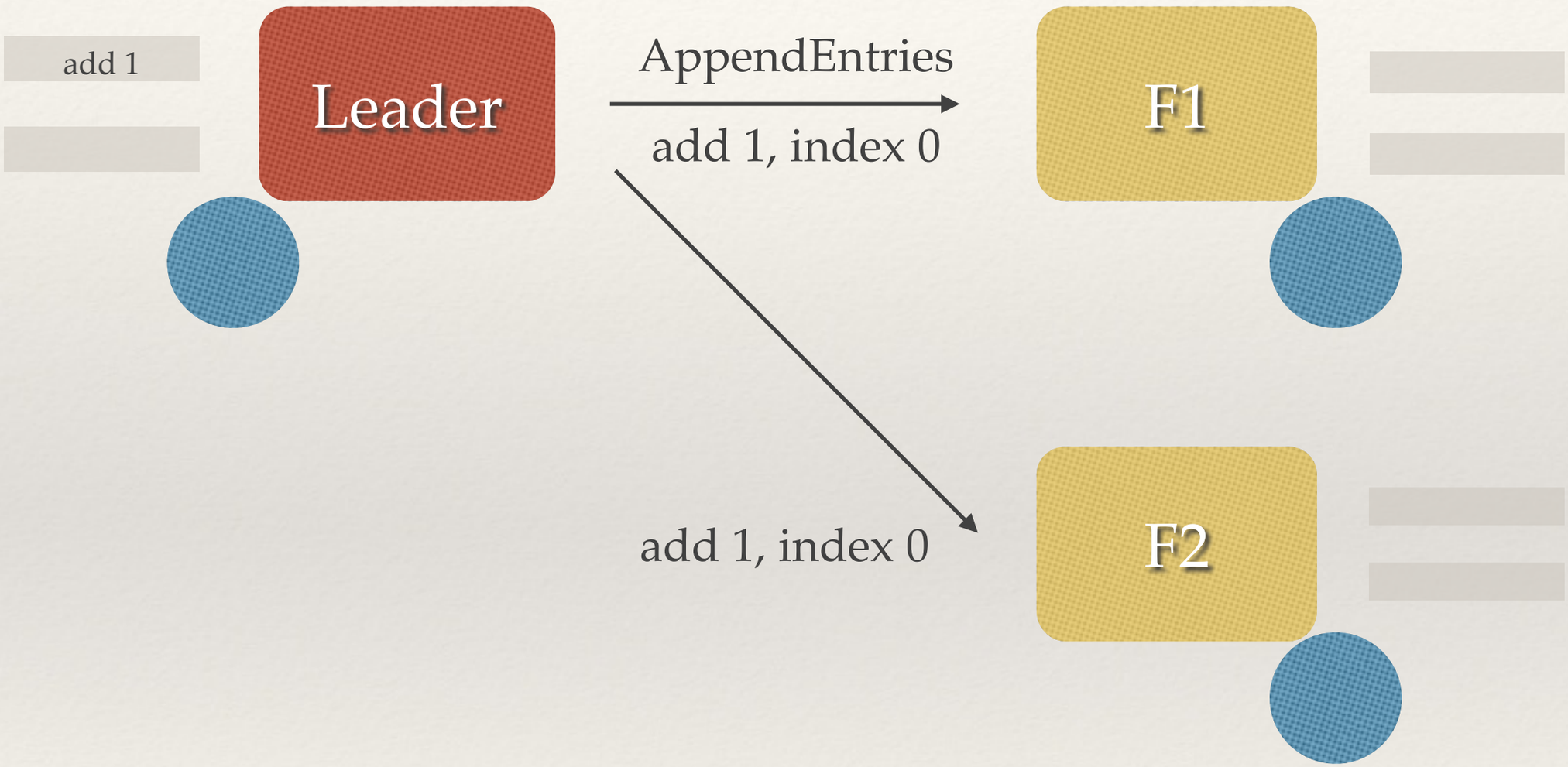
- Initiates Election
- Coordinates Votes



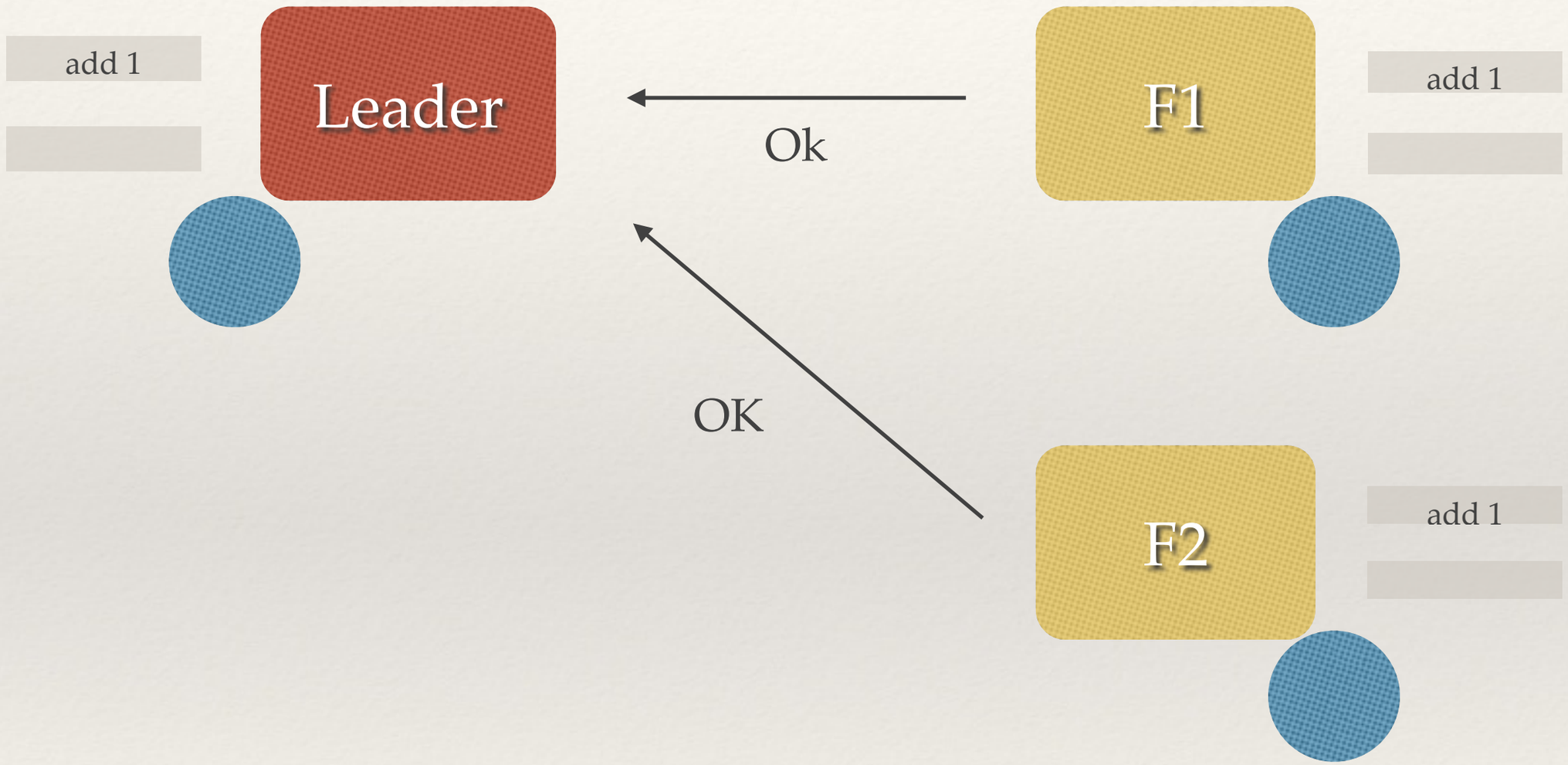
Leader Election



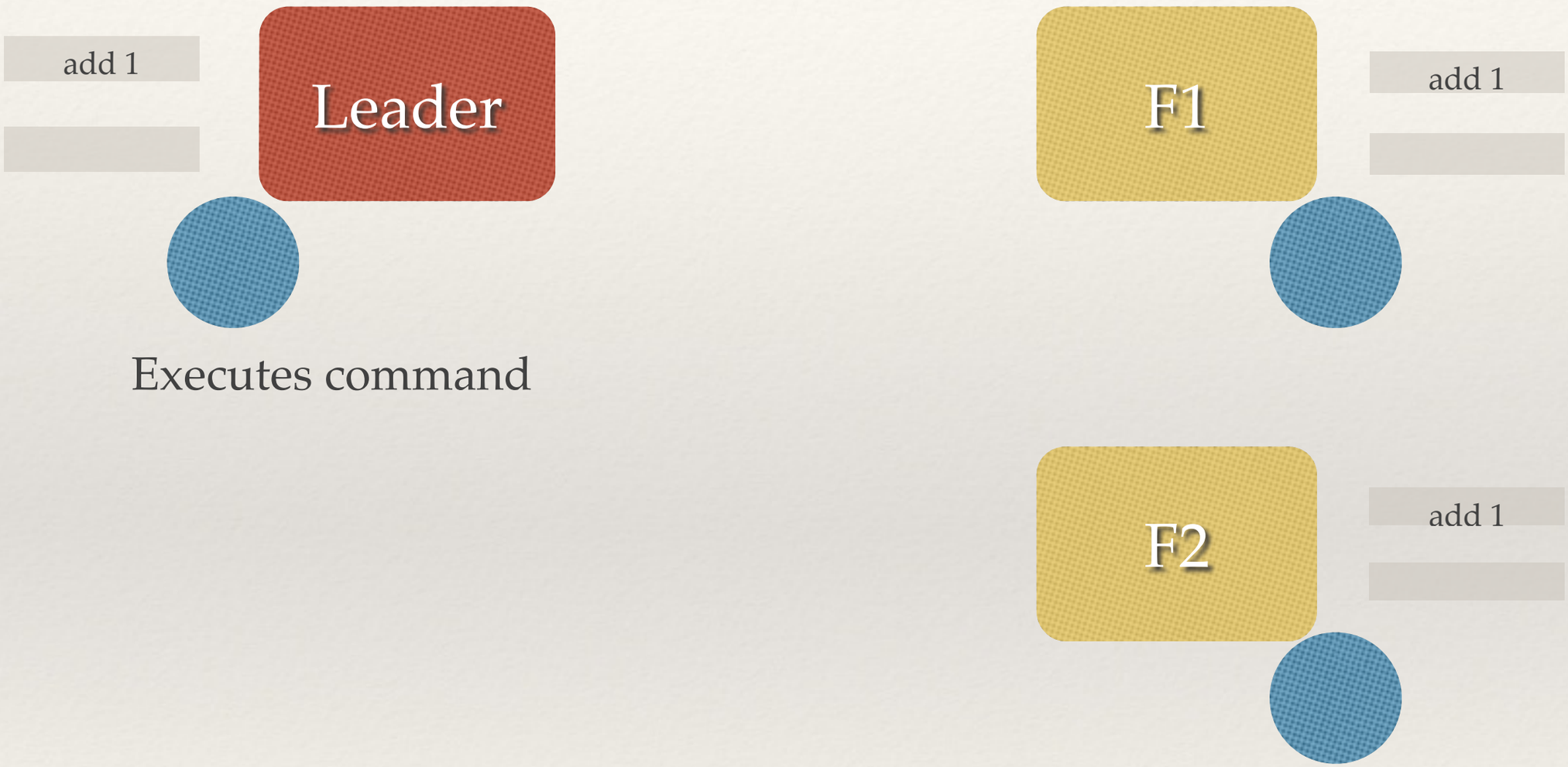
Log Replication



Log Replication

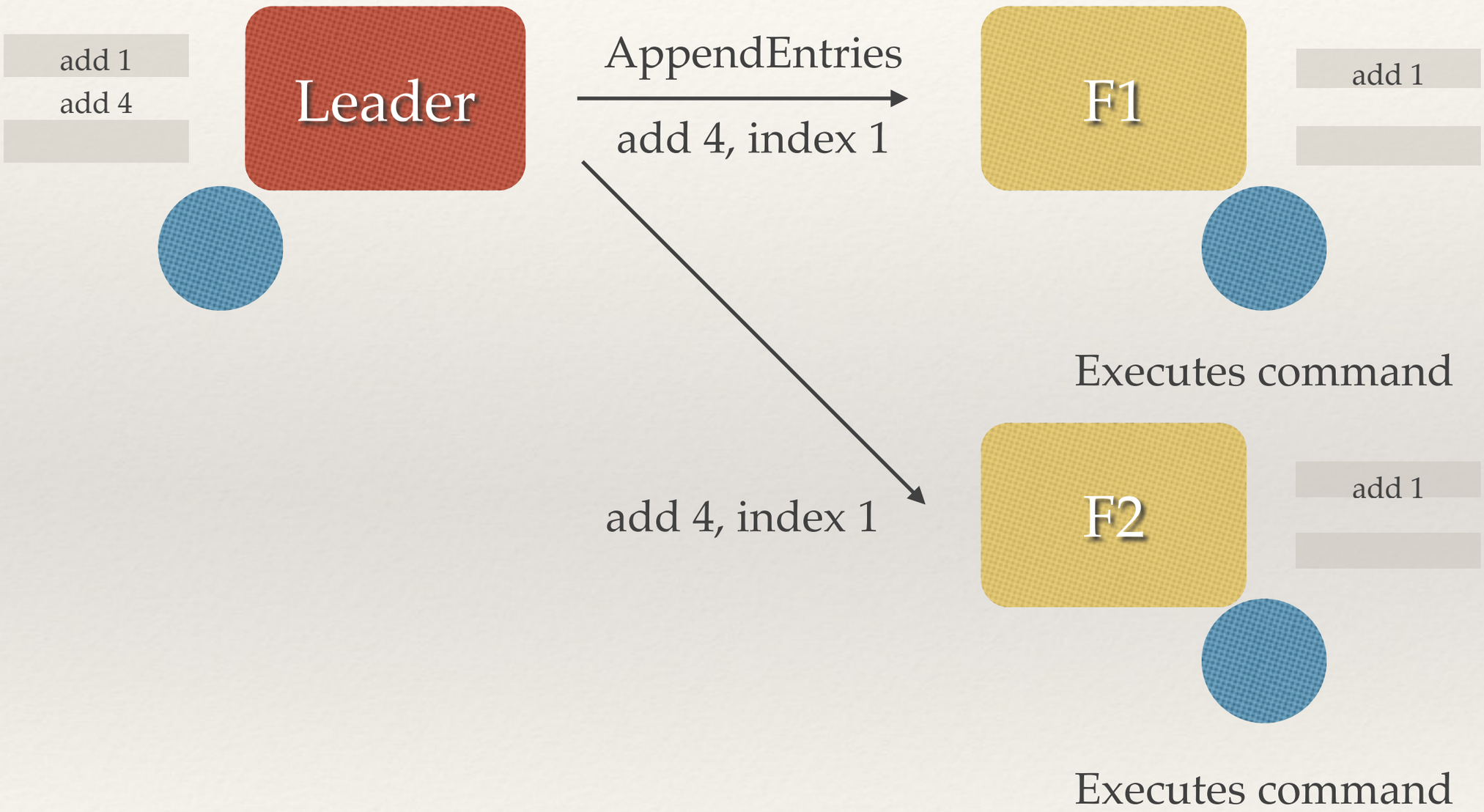


Log Replication



Executes command

Log Replication



RAFT Summary

- 2 types of messages, RequestVote and AppendEntries
- 3 states, Leader, Follower and Candidate
- Save Entries to persistent log

Erlang

- Functional language
- Fundamentally a concurrent language
- Actor model as basic abstraction
- No shared state between actors
- OTP behaviours like supervisors and gen_fsm
- Location independent message sending

Implementation Overview

- github.com/tmcgilchrist/sloop
- github.com/andrewjstone/rafter
- Each node has 2 supervised behaviours
- gen_fsm implementing the consensus protocol
- gen_server wraps the log store
- passes erlang terms as messages

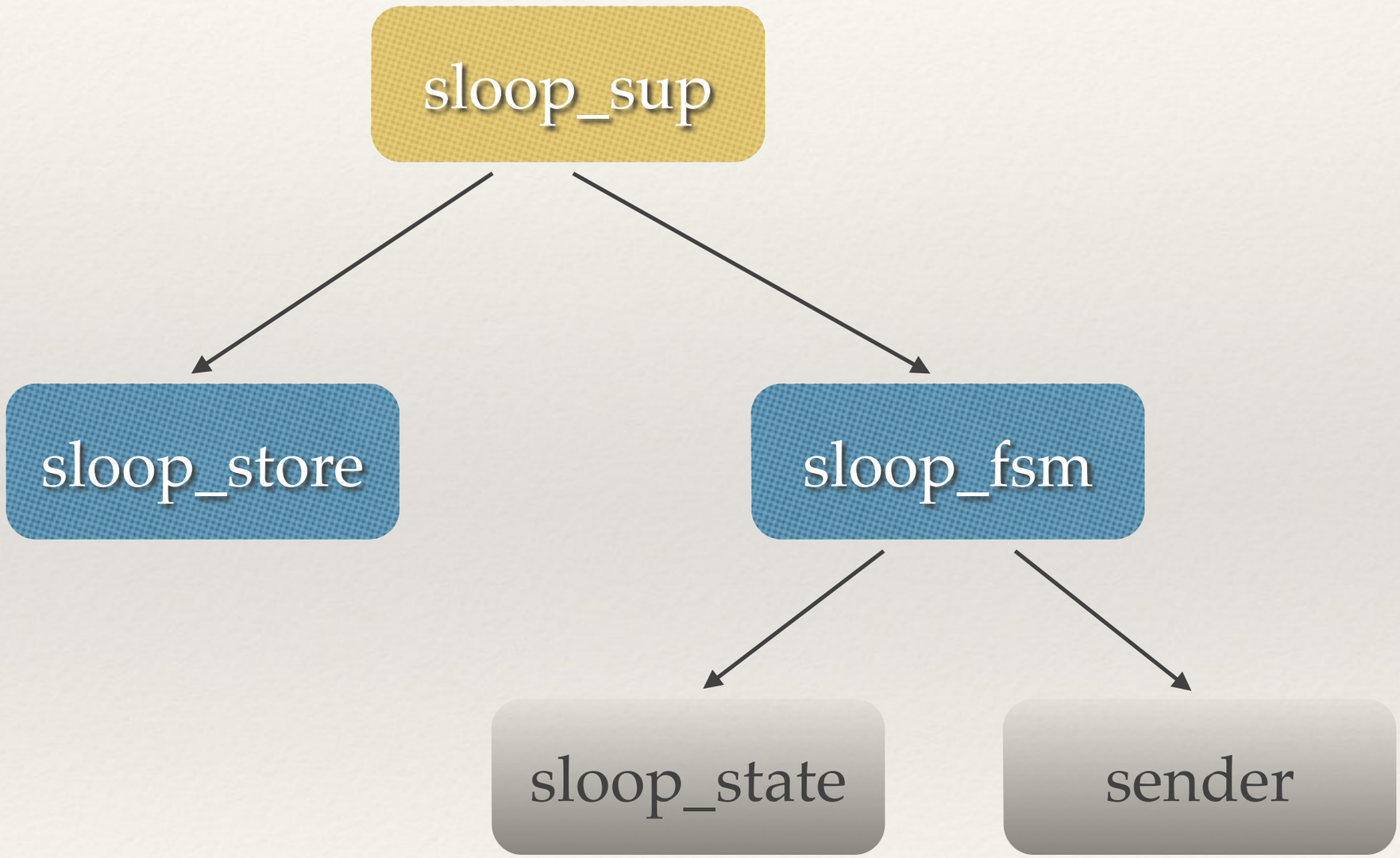
sloop_fsm

- state machine implements leader election and log replication
- each state is a function with multiple clauses

```
follower(timeout, State) ->
    reset_timer(election_timeout()),
    request_votes(State),
    {next_state, candiate, State};
follower(_Event, Data) ->
    {next_state, follower, Data}.

candidate({vote_rpc, From}, State) ->
    Responses = store_vote(From),
    case quorum_reached(Responses, State) of
        true ->
            NewState = become_leader(State),
            {next_state, leader, NewState};
        false ->
            NewState = State#state{responses=Responses},
            {next_state, candidate, NewState}
    end;
candidate(_Event, Data) ->
    {next_state, candidate, Data}.
```

Supervisors



Implementations

- raftconsensus.github.io
- github.com/tmcgilchrist/sloop
- github.com/andrewjstone/rafter
- github.com/goraft/raft

Summary

- Defined Distributed Consensus
- Looked at core ideas of RAFT
- Erlang suits distributed systems
- Map Erlang to RAFT

Thanks!