

# OCaml in Practice : Building Functional systems

Mohan Radhakrishnan<sup>1</sup>

*Programmer*

April 13, 2025  
ver.: 0.1

<sup>1</sup>contact also via my ID `radhakrishnan.mohan@gmail.com`.

# Contents

<b>I</b>	<b>Part 1 : First Steps</b>	<b>4</b>
<b>1</b>	<b>Key Multicore OCaml 5 features used</b>	<b>5</b>
1.1	Introducing OCaml 5 . . . . .	6
1.2	Effect-based direct Style IO(EIO) Programs . . . . .	6
1.3	Remote Procedure Calls using EIO. . . . .	6
1.4	What are effect handlers ? . . . . .	6
1.5	What is structured concurrency ? . . . . .	6
1.6	Summary . . . . .	6
<b>II</b>	<b>Part 2 : Storage Engine</b>	<b>7</b>
<b>2</b>	<b>Log Structured Merge</b>	<b>8</b>
2.1	Construction of LSM Tree . . . . .	9
2.2	SkipLists . . . . .	9
2.3	Bloom Filters . . . . .	9
2.4	MemTable . . . . .	9
2.5	Sorted String Table . . . . .	9
2.6	Summary . . . . .	9
<b>3</b>	<b>Compaction Strategies</b>	<b>10</b>
3.1	Simple Compaction . . . . .	11
3.2	Leveled Compaction . . . . .	11
3.3	Tiered Compaction . . . . .	11
3.4	Summary . . . . .	11

<b>III</b>	<b>Part 3 : Distributed Consensus</b>	<b>12</b>
<b>4</b>	<b>RAFT Distributed consensus protocol</b>	<b>13</b>
4.1	Remote Procedure Calls and State Machine . . . . .	14
4.2	Leader Election . . . . .	14
4.3	The Term . . . . .	14
<b>IV</b>	<b>Part 4 : Collaborative Editor</b>	<b>18</b>
<b>5</b>	<b>Resolving conflicts in distributed editing</b>	<b>19</b>
5.1	Conflict-free Replicate Data Types . . . . .	20
5.2	Logical Clocks . . . . .	20
5.3	Data Structures . . . . .	20
5.4	Developing the editor UI . . . . .	20
5.5	Developing RPCs using capnp-rpc and EIO . . . . .	20
5.6	Summary . . . . .	20
<b>V</b>	<b>Part 5 : Streaming Data</b>	<b>21</b>
<b>6</b>	<b>Basic Distributed Data Pipeline</b>	<b>22</b>
6.1	Introducing Sketching . . . . .	23
6.2	Approximate Distinct Counting . . . . .	23
6.3	Other Sketching Algorithms (Placeholder) . . . . .	23
6.4	Summary . . . . .	23

This file loads all content.

## PART I

# *Part 1 : First Steps*

## CHAPTER 1

# *Key Multicore OCaml 5 features used*

Abstract

- 1.1 Introducing OCaml 5**
- 1.2 Effect-based direct Style IO(EIO) Programs**
- 1.3 Remote Procedure Calls using EIO.**
- 1.4 What are effect handlers ?**
- 1.5 What is structured concurrency ?**
- 1.6 Summary**

## PART II

# *Part 2 : Storage Engine*



## CHAPTER 2

# *Log Structured Merge*

- 2.1 Construction of LSM Tree**
- 2.2 SkipLists**
- 2.3 Bloom Filters**
- 2.4 MemTable**
- 2.5 Sorted String Table**
- 2.6 Summary**

## CHAPTER 3

# *Compaction Strategies*

Abstract

- 3.1 Simple Compaction**
- 3.2 Leveled Compaction**
- 3.3 Tiered Compaction**
- 3.4 Summary**

## PART III

# *Part 3 : Distributed Consensus*

## CHAPTER 4

# *RAFT Distributed consensus protocol*

### Abstract

The Raft consensus Algorithm was designed by Diego Ongaro and John Ousterhout at Stanford University. Apart from other characteristics they argue that it is designed for **Understandability**.

The following primary characteristics are what the Raft authors mention.

- Consensus is agreement of shared state
- System is up if majority of servers are up
- Needed for consistent, fault-tolerant storage systems

## 4.1 Remote Procedure Calls and State Machine

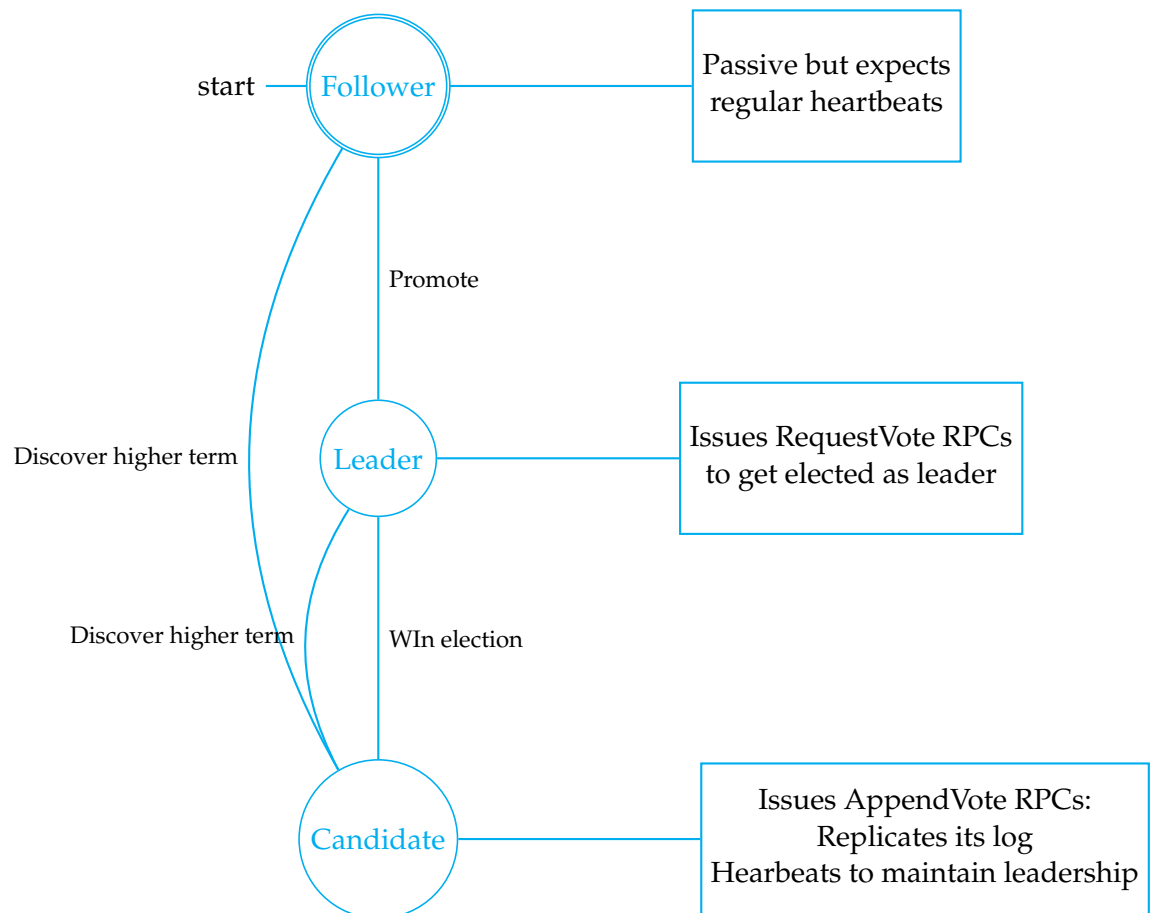


Figure 4.1

## 4.2 Leader Election

## 4.3 The Term

A term is a value that is sent with every RPC and received in every response. It is used to identify obsolete information (*e.g*) If a peer has a later term, the term is updated and the status is reverted to *Follower*. Every server maintains its own term and so there is no-*Global view*.

```
1    let get_state = function
2      | `Leader -> "leader."
3      | `Follower -> "follower."
4      | `Candidate -> "candidate."
5      | `Dead -> "_dead."
```

*Code 4.1: A example with parameter in a environment.*

- RequestVote : Solicits votes from other members of the cluster
- AppendEntries : Replicates the log and can also server as a heartbeat



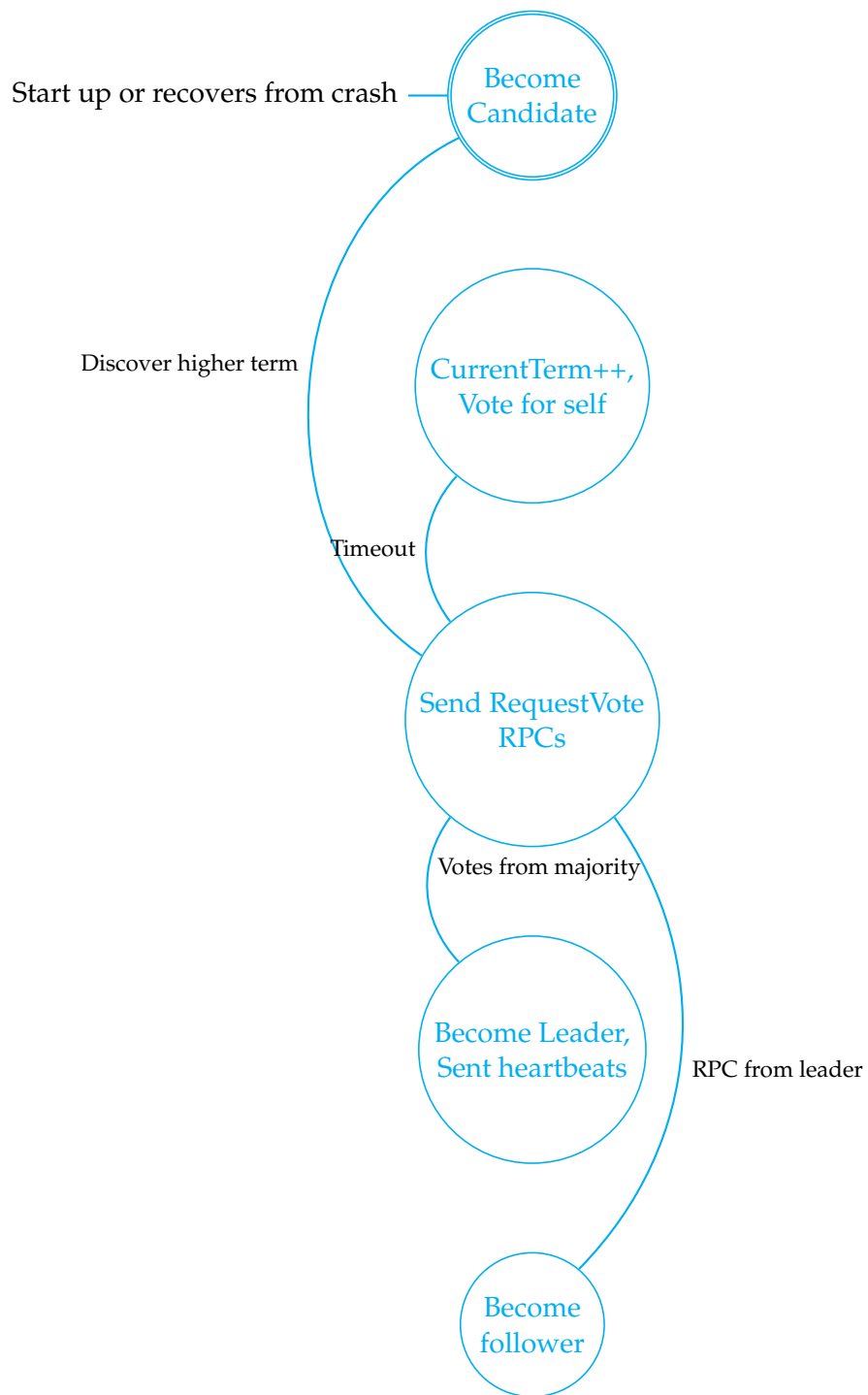


Figure 4.2: John Ousterhout's presentation.

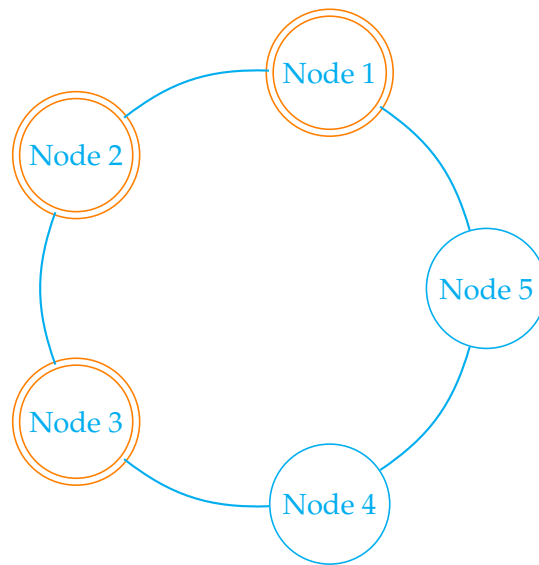


Figure 4.3: John Outershout's presentation

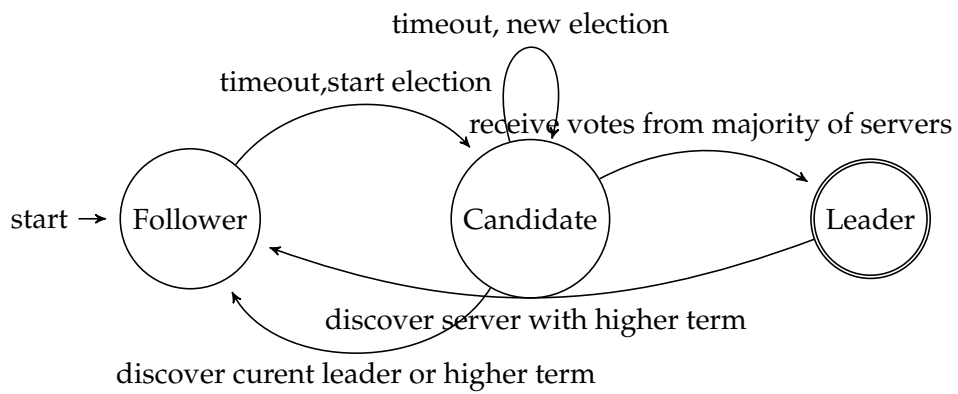


Figure 4.4

**PART IV**

*Part 4 :  
Collaborative  
Editor*

## CHAPTER 5

# *Resolving conflicts in distributed editing*

Abstract

- 5.1 Conflict-free Replicate Data Types**
- 5.2 Logical Clocks**
- 5.3 Data Structures**
- 5.4 Developing the editor UI**
- 5.5 Developing RPCs using capnp-rpc and EIO**
- 5.6 Summary**

## PART V

# *Part 5 : Streaming Data*

## CHAPTER 6

# *Basic Distributed Data Pipeline*

Abstract

- 6.1 Introducing Sketching**
- 6.2 Approximate Distinct Counting**
- 6.3 Other Sketching Algorithms (Placeholder)**
- 6.4 Summary**