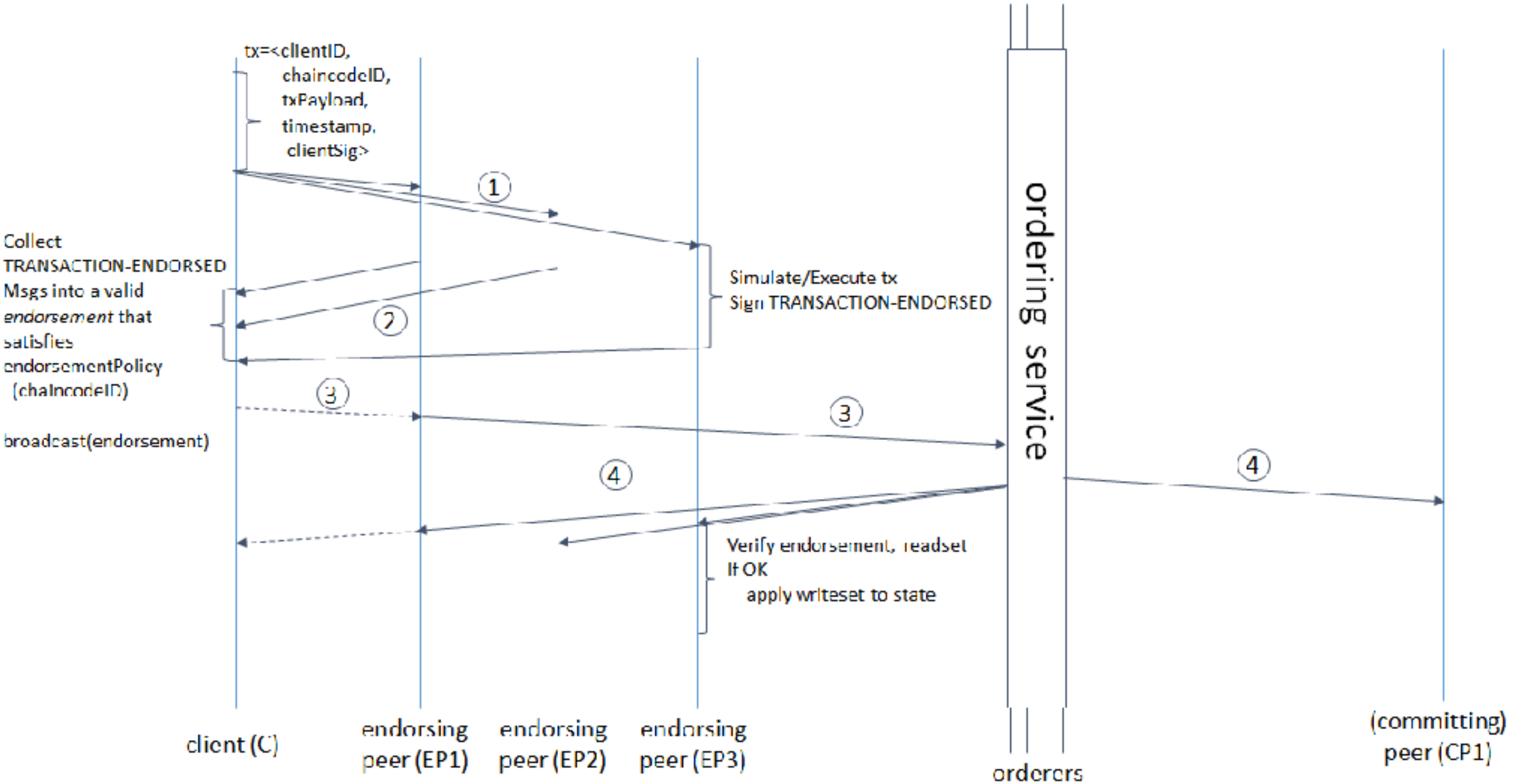
基于Raft的排序服务

郭剑南
@guoger

- Fabric Orderer recap
- Raft - a brief intro
- Raft-based Orderer
- Configuration
- Architecture

Fabric Orderer recap



Fabric Orderer recap

- Orderer provides **Atomic Broadcast** service
- Absolute Finality
- Data consistency (no fork)
- Orderer does not look at tx data

Raft - a brief intro

- Raft is a consensus algorithm that is designed to be easy to understand
- **C**rash **F**ault **T**olerant
- Replicated State Machine
- Quorum = 51%

Raft - a brief intro

- Leader election
    - Heartbeats
    - Random timeout
- Log Replication
    - leader coordinates replication
    - 2-phase commit

Raft-based Orderer

- No Kafka/Zookeeper dependencies
  easier to operate

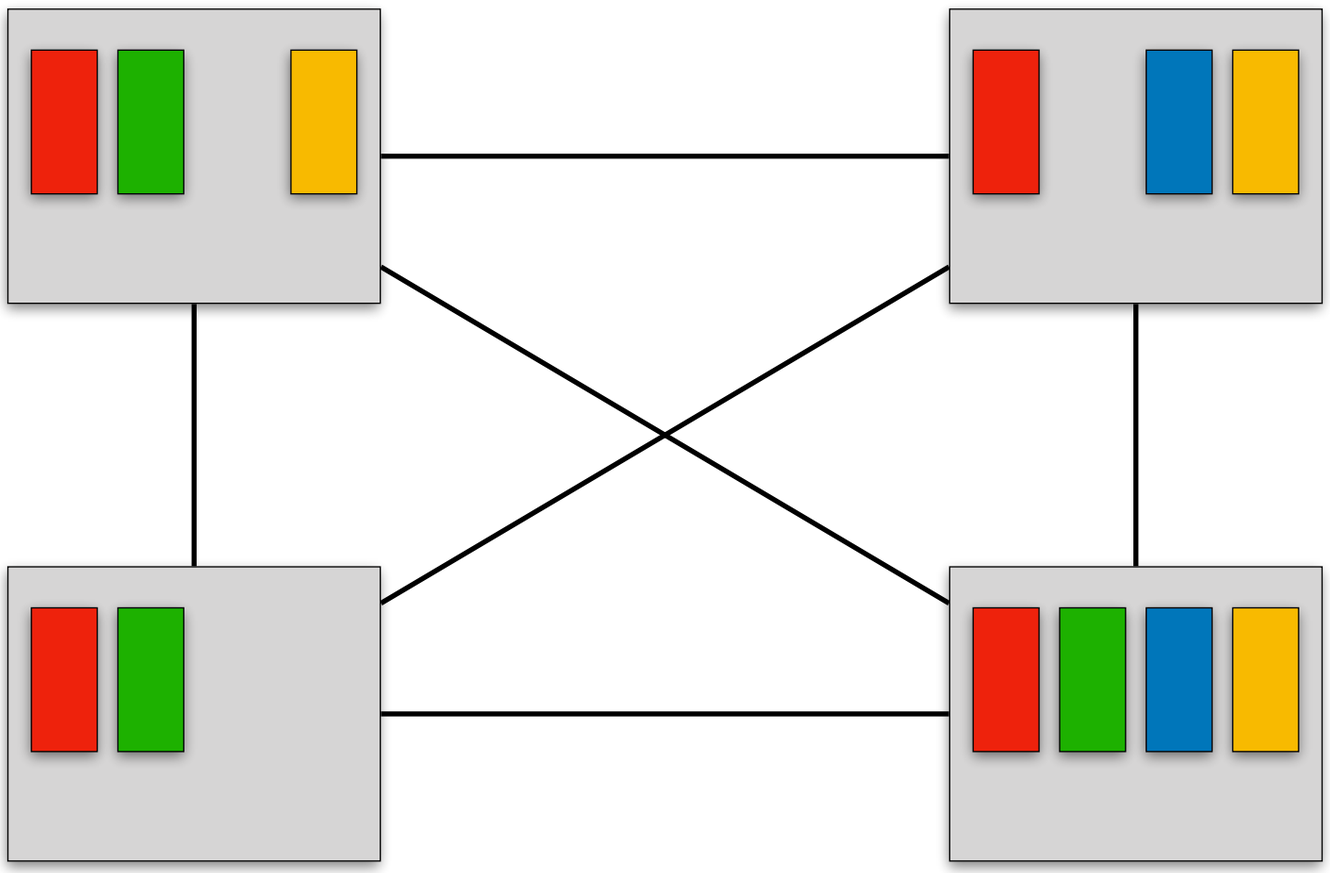
- Get prepared for BFT
  multiple components are independent of consensus plugin (reusable)

- etcd/raft library
  well tested, written in Go, compatible license

- Nodes are identified by TLS Cert
  TLS required among orderers, and they authenticate each other via TLS cert pinning

- Support Migrating from Kafka to Raft - coming soon

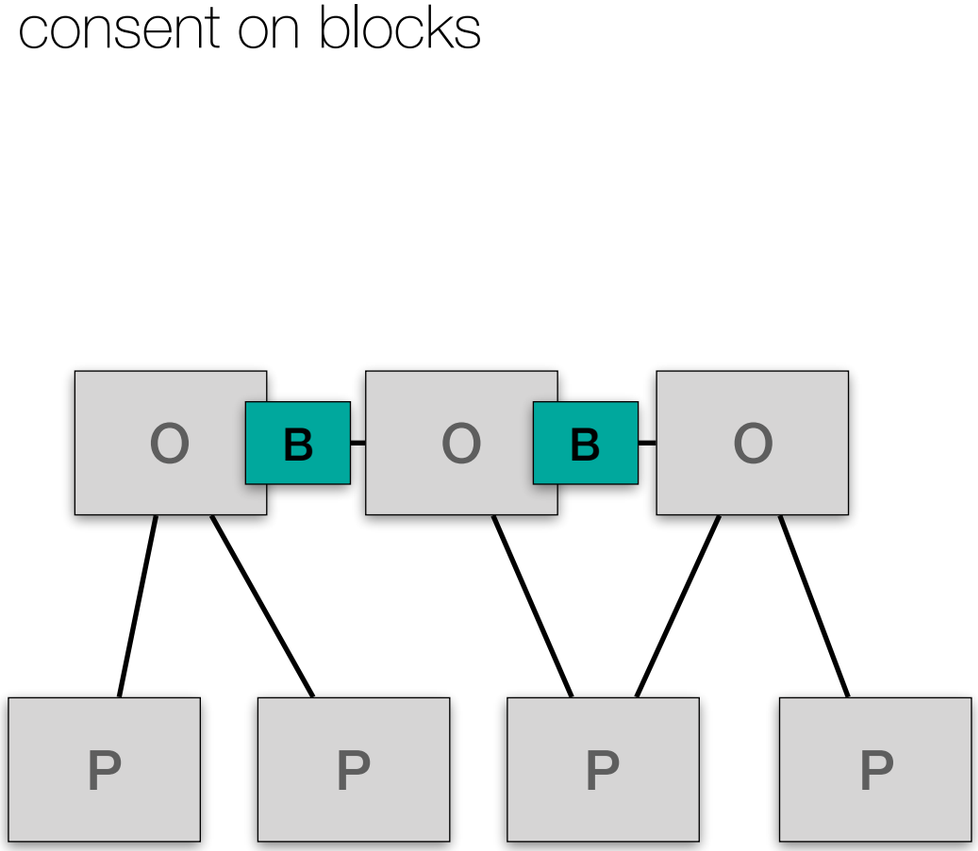
Raft-based Orderer



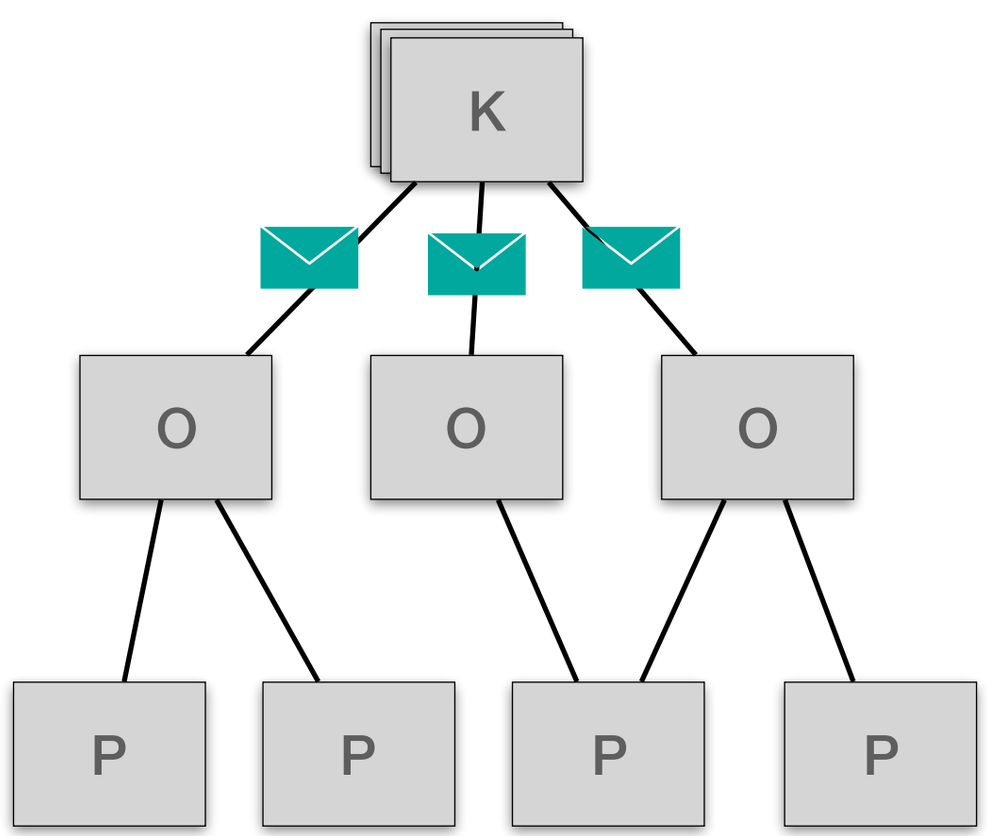
System Channel runs on every Orderer

Raft-based Orderer

- less hop
- p2p network
- consent on blocks

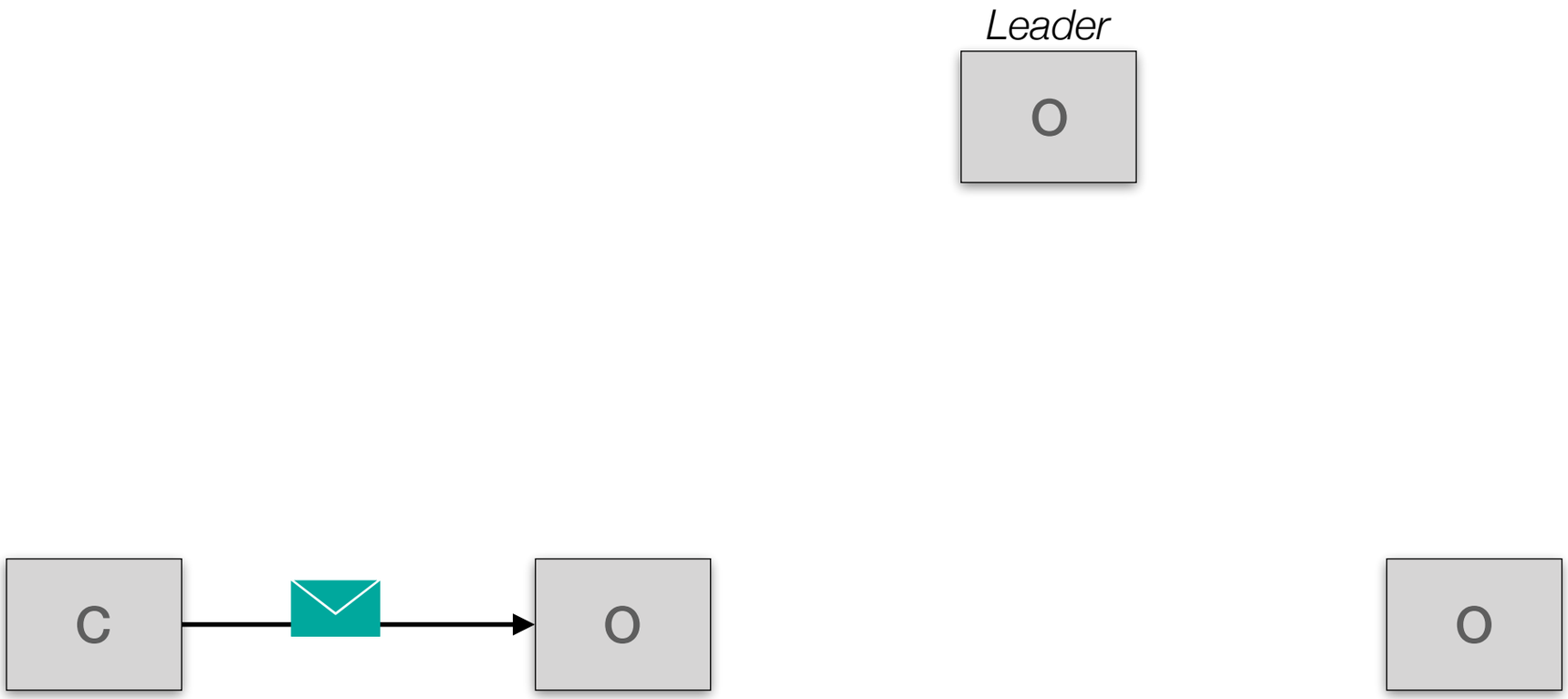


Raft

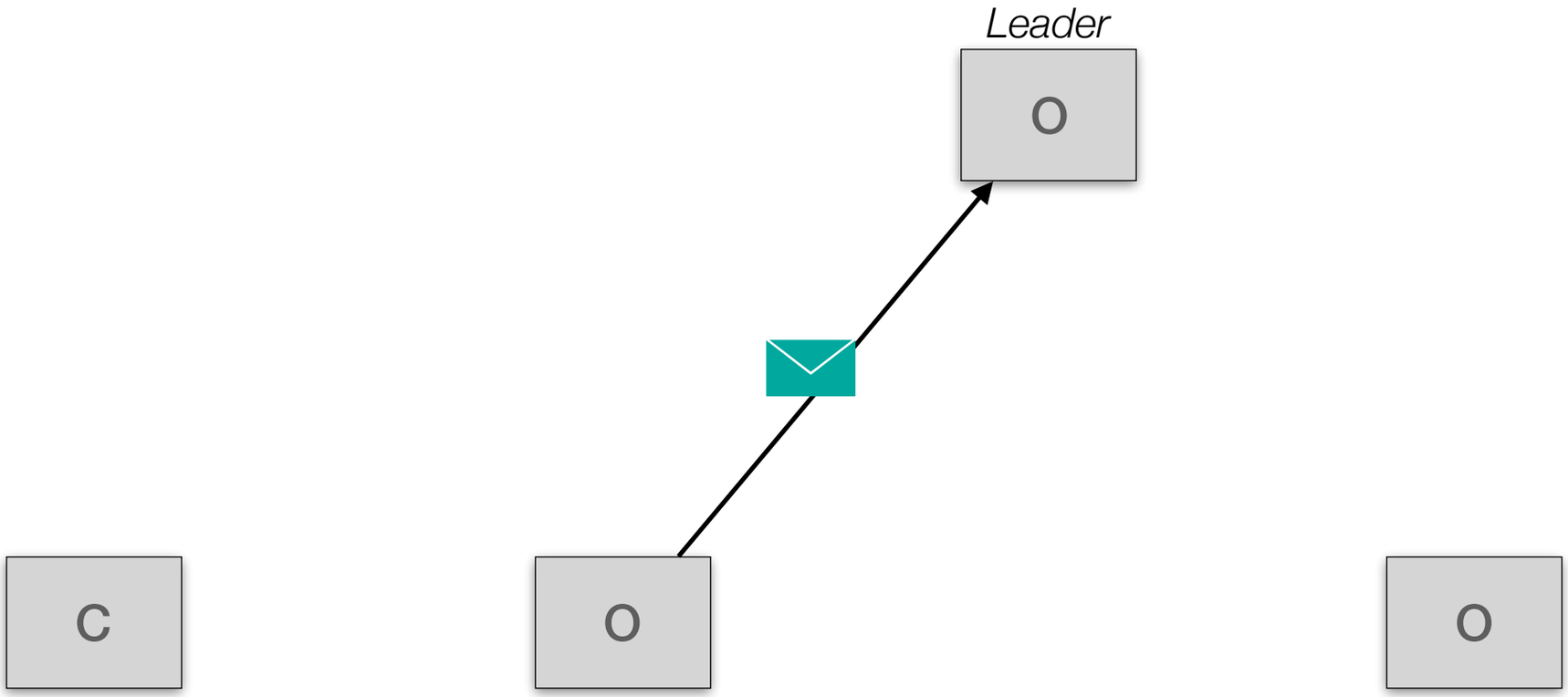


Kafka

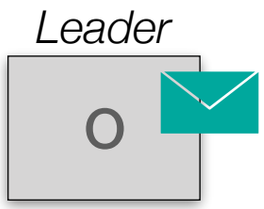
Raft-based Orderer

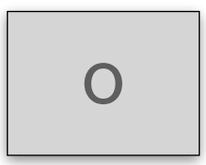


Raft-based Orderer

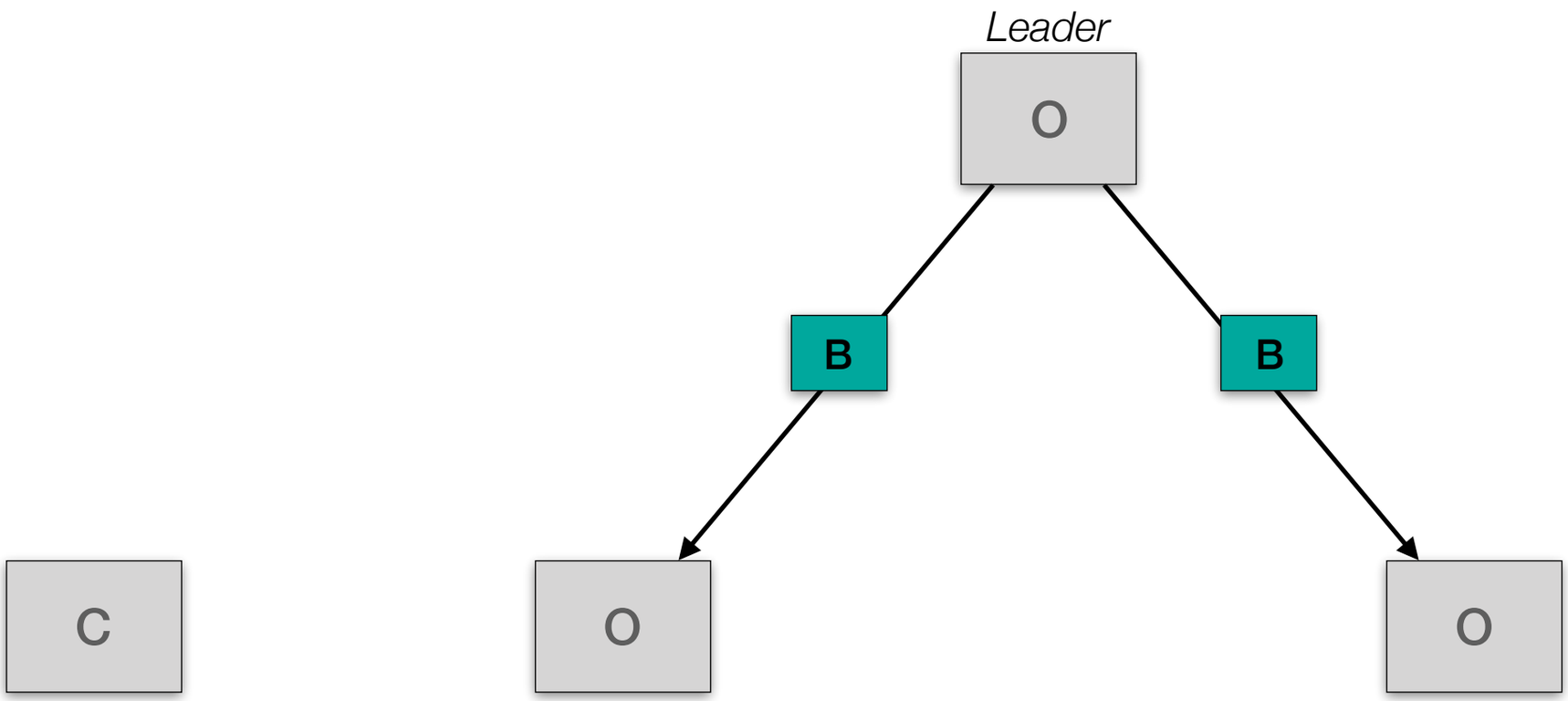


Raft-based Orderer

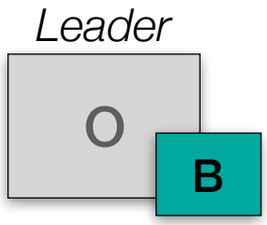


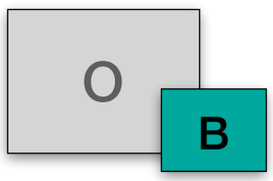


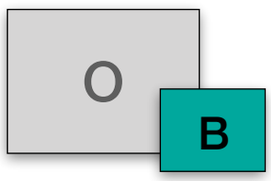
Raft-based Orderer



Raft-based Orderer







Raft-based Orderer

- Write Ahead Log (WAL)
  - Stores data
  - 1 block per entry
    - Disk IO intensive
- Snapshot
  - Each snapshot stores **one** block
  - Lagged orderer receives snapshot and invokes **Deliver** on other nodes
  - Once Snapshot is taken, WAL before that will be pruned

Raft-based Orderer

```
# EtcdRaft defines configuration which must be set when the "etcdraft"
# orderertype is chosen.
EtcdRaft:
    # The set of Raft replicas for this network. For the etcd/raft-based
    # implementation, we expect every replica to also be an OSN. Therefore,
    # a subset of the host:port items enumerated in this list should be
    # replicated under the Orderer.Addresses key above.
    Consenters:
        - Host: raft0.example.com
          Port: 7050
          ClientTLSCert: path/to/ClientTLSCert0
          ServerTLSCert: path/to/ServerTLSCert0
        - Host: raft1.example.com
          Port: 7050
          ClientTLSCert: path/to/ClientTLSCert1
          ServerTLSCert: path/to/ServerTLSCert1
        - Host: raft2.example.com
          Port: 7050
          ClientTLSCert: path/to/ClientTLSCert2
          ServerTLSCert: path/to/ServerTLSCert2
```

Raft Node ID is detected using server certificate

Raft-based Orderer

- Onboard new Orderer to system channel
  - Add properties of new Orderer to consenter set
  - Submit ConfigUpdate to Orderer
  - Start new Orderer **with Latest Config Block**

```
# EtcdRaft defines configuration which must be set w
# ordererertype is chosen.
EtcdRaft:
    # The set of Raft replicas for this network. For
    # implementation, we expect every replica to als
    # a subset of the host:port items enumerated in
    # replicated under the Orderer.Addresses key abo
    Consenters:
    - Host: raft0.example.com
      Port: 7050
      ClientTLSCert: path/to/ClientTLSCert0
      ServerTLSCert: path/to/ServerTLSCert0
    - Host: raft1.example.com
      Port: 7050
      ClientTLSCert: path/to/ClientTLSCert1
      ServerTLSCert: path/to/ServerTLSCert1
    - Host: raft2.example.com
      Port: 7050
      ClientTLSCert: path/to/ClientTLSCert2
      ServerTLSCert: path/to/ServerTLSCert2
    - Host: raft3.example.com
      Port: 7050
      ClientTLSCert: path/to/ClientTLSCert3
      ServerTLSCert: path/to/ServerTLSCert3
```

Configuration

```
################################################################################
#
#   Consensus Configuration
#
#   - This section contains config options for a consensus plugin. It is opaque
#     to orderer, and completely up to consensus implementation to make use of.
#
################################################################################
Consensus:
    # The allowed key-value pairs here depend on consensus plugin. For etcd/raft,
    # we use following options:

    # WALDir specifies the location at which Write Ahead Logs for etcd/raft are
    # stored. Each channel will have its own subdir named after channel ID.
    WALDir: /var/hyperledger/production/orderer/etcdraft/wal

    # SnapDir specifies the location at which snapshots for etcd/raft are
    # stored. Each channel will have its own subdir named after channel ID.
    SnapDir: /var/hyperledger/production/orderer/etcdraft/snapshot
```

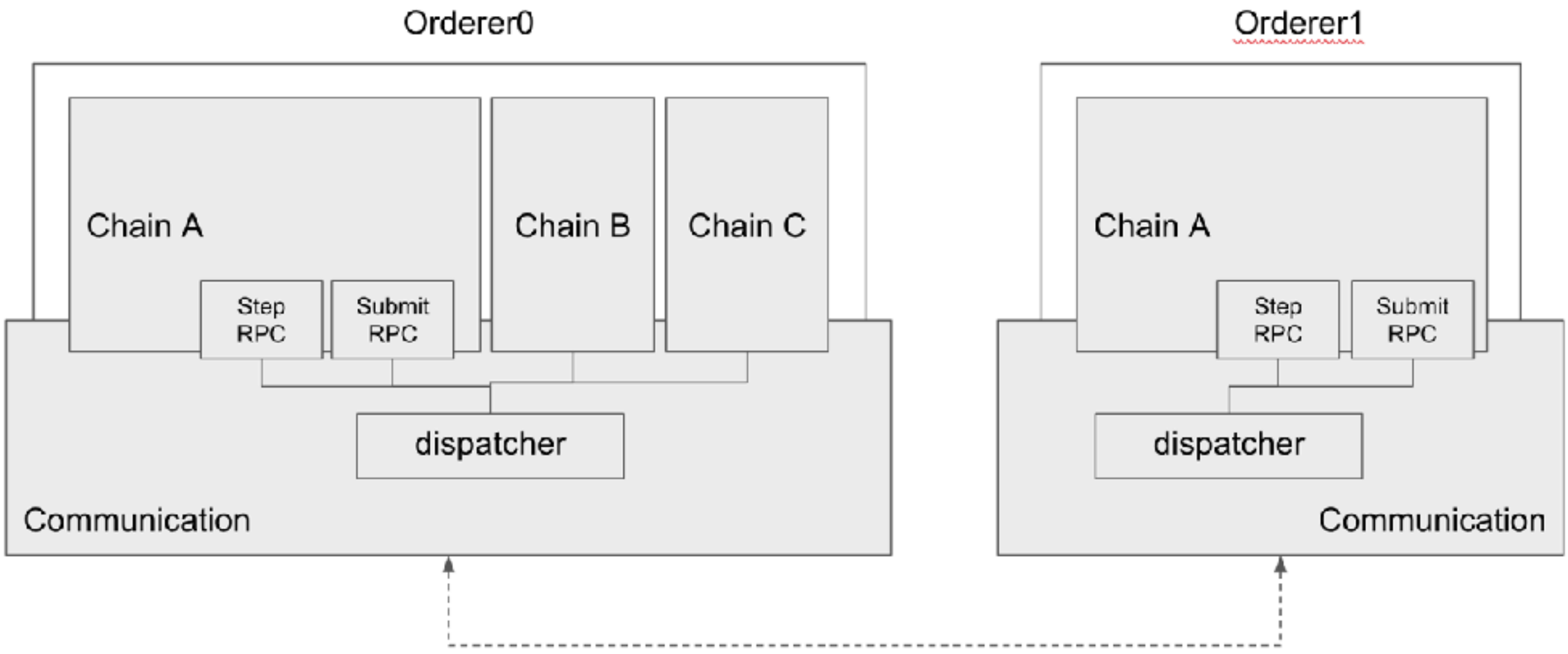
Orderer.yaml

Configuration

```yaml
# EtcdRaft defines configuration which must be set when the "etcdraft"
# ordererтype is chosen.
EtcdRaft:

    # Options to be specified for all the etcd/raft nodes. The values here
    # are the defaults for all new channels and can be modified on a
    # per-channel basis via configuration updates.
    Options:
        # TickInterval is the time interval between two Node.Tick invocations.
        TickInterval: 500ms
        # ElectionTick is the number of Node.Tick invocations that must pass
        # between elections. That is, if a follower does not receive any
        # message from the leader of current term before ElectionTick has
        # elapsed, it will become candidate and start an election.
        # ElectionTick must be greater than HeartbeatTick.
        ElectionTick: 10

        # HeartbeatTick is the number of Node.Tick invocations that must
        # pass between heartbeats. That is, a leader sends heartbeat
        # messages to maintain its leadership every HeartbeatTick ticks.
        HeartbeatTick: 1

        # MaxInflightBlocks limits the max number of in-flight append messages
        # during optimistic replication phase.
        MaxInflightBlocks: 5

        # SnapshotIntervalSize defines number of bytes per which a snapshot is taken
        SnapshotIntervalSize: 20 MB
```

Configtx.yaml

Architecture



Thanks!