Blockchain Technology and Applications

CS 989

Hyperledger

Sumit Lahiri

HYPERLEDGER FABRIC

Early Web3 Technologies (Cryptocurrencies)

- Computation on the blockchain
- Control transfer and validation
  - Bitcoin (Scripts)
    - stack-based, reverse-polish, Turing incomplete
    - interact with the Bitcoin software
    - Examples: Pay To PubKey Hash (P2PKH), Pay To ScriptHash (P2SH)
  - Ethereum (Solidity)
    - Turing complete
    - Smart Contracts written in a close to C language, Solidity
    - Inter-contract execution, function call, storage etc.

Issues with Solidity/Bitcoin

- Scalability
  - Transaction commit rate
  - High Block time
  - PoW based consensus, leading to high resource usage
  - Transactions come at high cost, depends heavily on cryptocurrency prices.
- Public Anonymity
  - Transactions are publicly visible, account balances are visible
  - Tracking the origin of a cryptocurrency transfer is hard.
- Permissionless
  - Anybody can join the network and start transactions.
- Throughput
  - Lower throughput, block times are increasing.
  - More miners increase throughput but come at higher transaction cost.
- Ledger Forking
  - Divergent ledgers where different participants have different view of the accepted order of the transactions.

HyperLedger Fabric

HYPERLEDGER FABRIC

Solves all the issues!

- High Throughput
- Permissioned (although Permissionless exists)
- Versatility
- Scalability
- Identity
- Authenticity
- Enterprise-grade

[Hyperledger Fabric Model](#)
[Introduction — hyperledger-fabric docs main documentation](#)

HyperLedger Overview

- Multi-party
- Pluggable Consensus Mechanisms
- Language/Platform Agnostic
- Modular/Configurable.
- Privacy (comes by default!)
- Not pegged with a crypto-currency
- Supports writing contract code (chaincode) in multiple popular programming languages (Java/GoLang etc.)

HYPERLEDGER FABRIC

[Hyperledger Fabric Model]
[Introduction — hyperledger-fabric docs main documentation]

Key components – Chaincode

Chaincode is software defining an asset or assets, and the transaction instructions for modifying the asset(s) on the fabric network.

- Core *Business Logic.*
- *Programming* the resources on the blockchain.
- Update/Query of the fabric's *ledger* via the chaincode.
- Invoked via *transactions* similar to Ethereum.
- Ledger records all of the transactions generated by smart contracts (a.k.a chaincode)
    - *Easily verifiable*
    - *Immutable records*

Smart Contracts and Chaincode — hyperledger-fabricdocs main documentation

Key components – Peers

- Manage and deploy chaincode (ready for execution)
- Cache copies of the ledger.
- Keeps updating the ledger from time to time (as new blocks arrive)
- Peers enable the interaction with the underlying fabric network via chaincode. (*Fabric gateway service*)
- Peers are capable of hosting multiple chaincode instances and ledger copies.
- Peers interact with each other on *channels*.
- Peers are uniquely identifiable via *digital certificates* assigned to them via CAs/MSPs.

https://hyperledger-fabric.readthedocs.io/en/release-2.5/peers/peers.html
https://hyperledger-fabric.readthedocs.io/en/release-2.5/peers/peers.html#peers-and-channels

Key components – Orderers

- Ordering Node in the Fabric Network
- Multiple ordering nodes form a *ordering service.*
- Maintain a *single-source-of-truth* ledger by ordering the transactions sequentially.
- Ordering Service maintains follows a deterministic consensus algorithm to order the transactions.
- Ledgers in Fabric cannot *fork*, unlike other traditional blockchains.
- Single ordering node in the ordering service may handle more than one ledger.
- Similar to *peers*, ordering nodes are also assigned *identity* provided by a CAs/MSPs.

The Ordering Service — hyperledger-fabric docs main documentation

Key components – Consensus Mechanisms

- Single Ordering Node Service (SOLO) [depreciated in v2.x]
  - Still useful when setting up the fabric network for testing.
  - Slow and low throughput but best for testing if the ordering is correctly captured.
- Kafka [depreciated in v2.x]
  - Comes from Apache Kafka, based on algorithm similar to *RAFT*.
  - Resilient and robust message queue for storing incoming transactions.
  - Leader-to-follower node configuration.
  - Maintaining a Kafka cluster on top of fabric can be cumbersome.
- RAFT [current and recommended]
  - Fully crash fault tolerant.
  - Easier to setup than Kafka with little to no overhead.

The Ordering Service — hyperledger-fabric docs main documentation

Brief on how RAFT protocol work?

- Set of interconnected servers, peers/users interact with only one of the *servers*.
- Server consists of a *state-machine* and a *log* (*Transaction List* – in our case!)
- State machine needs to be *fault tolerant* and *robust*.
- Log consists instruction to *change/update* the global data store (*Ledger*), stored typically in a *message queue* of each server node (other storages can also be used, but needs to be fast and scalable).

If a server S_i applies some instruction "x" as the n^{th} instruction from it's log, then no other server S_j can apply any other instruction "x" as the n^{th} instruction.

This guarantees consensus!

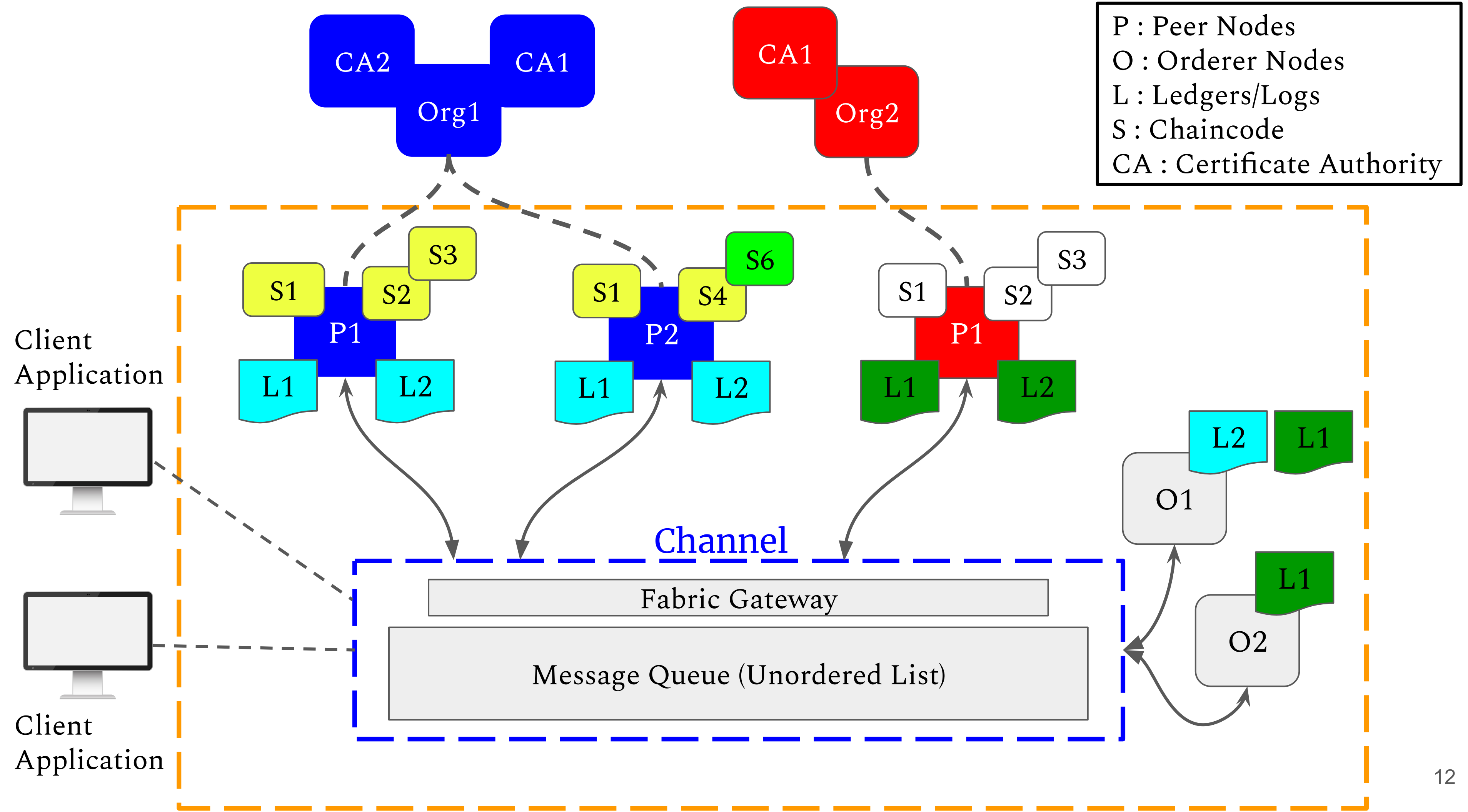
Raft Consensus Algorithm
What is Apache Kafka? | IBM
Apache Kafka Architecture and Internals by Jun Rao

Key components – CAs/MSPs

It is important to determine the exact permissions over resources and access to information that actors have in a blockchain network

- Digital identity provided via a *X.509 digital certificate*.
- Certificate Authority provides the *digital certificates*.
- Membership Service Providers (MSP) manages the certificates and assignes them to peers, ordering nodes, organizations (peers are under organization) etc.
- Solely responsible for identity management and revocations.

[Identity — hyperledger-fabric docs main documentation]


P : Peer Nodes
O : Orderer Nodes
L : Ledgers/Logs
S : Chaincode
CA : Certificate Authority
CA2
CA1
Org1
CA1
Org2
S1
S2
S3
P1
L1
L2
S1
S4
S6
P2
L1
L2
S1
S2
S3
P1
L1
L2
Client Application
Client Application
Channel
Fabric Gateway
Message Queue (Unordered List)
L2
L1
O1
L1
O2

Transaction Overview

- Phase 1 - Transaction Proposal and Endorsement
    - Clients submit transaction for next execution
    - Endorsing organizations must validate that the transaction is correct and authorized via the peer.
    - Peer is selected for chaincode execution via the fabric gateway.
    - For the transaction, peer returns the execution result as a response back to the fabric gateway.
    - Steps are repeated for each liable organization.

https://hyperledger-fabric.readthedocs.io/en/release-2.5/peers/peers.html#phase-1-transaction-proposal-and-endorsement

Transaction Overview

- Phase 2 - Transaction Submission and Ordering
  - Fabric Gateway submits the response from peer back to the message queue (network)
  - On submission, forwards it to a ordering node and returns a *success* token to the peer network.
  - Ordering service picks the orderer and, verifies the authenticity of the transactions and orders it for addition to the block.
  - Newly formed block is sent back to the gateway for distribution to all active peer nodes for ledger updation.

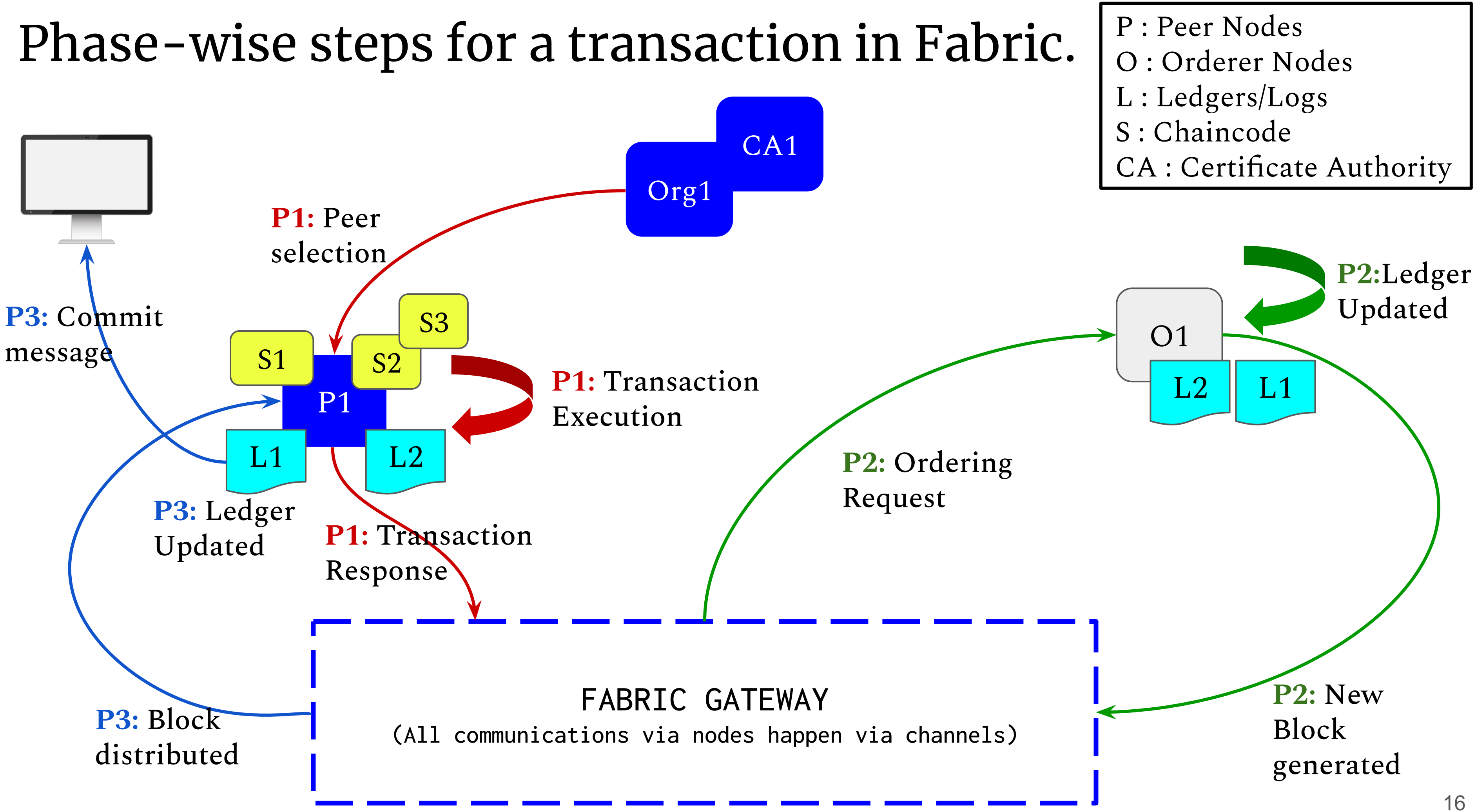
https://hyperledger-fabric.readthedocs.io/en/release-2.5/peers/peers.html#phase-2-transaction-submission-and-ordering

Transaction Overview

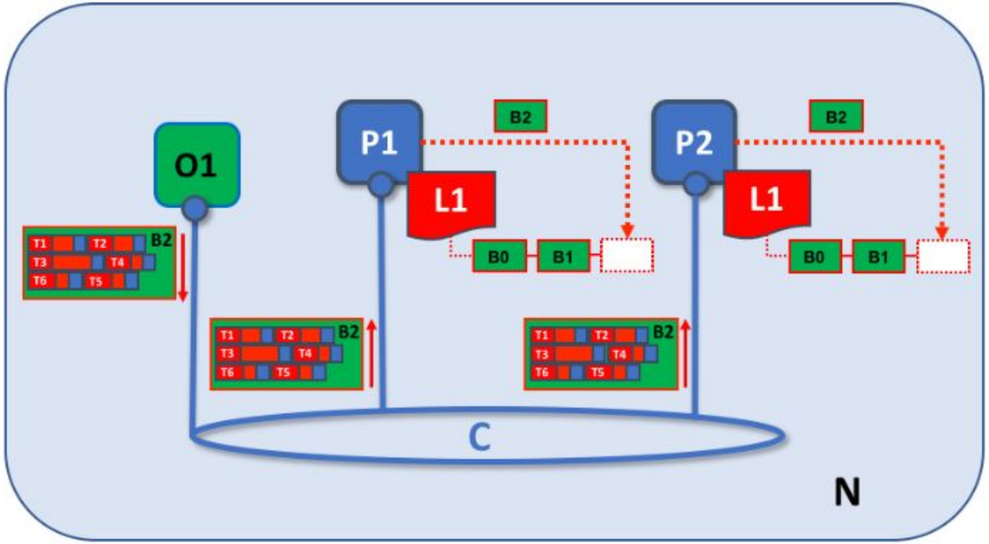
- Phase 3 - Transaction Validation and Commitment
  - Each peer receiving the block, verifies if the transaction in it are valid.
  - The ordering service ordering the *transaction* is validated
  - Ledger is updated and *commit* event is emitted back by the peer to the fabric gateway.

https://hyperledger-fabric.readthedocs.io/en/release-2.5/peers/peers.html#phase-3-transaction-validation-and-commitment

Phase-wise steps for a transaction in Fabric.



Transaction Validation**



N	Blockchain Network	P	Peer
C	Channel	O	Orderer
L	Ledger	B	Block B
L1 B0 B1	Ledger L1 has blockchain with blocks B0, B1	T1 T2 T3 B1	Block B1 contains transactions T1, T2, T3…
B1 C	Block B1 flows on channel C	PA C	Principal PA (P1, P2) communicates via channel C.

**courtesy: https://hyperledger-fabric.readthedocs.io/en/release-2.5/smartcontract/smartcontract.html

https://hyperledger-fabric.readthedocs.io/en/release-2.5/peers/peers.html#phase-3-transaction-validation-and-commitment

Fabric Example – Chaincode Initialization

Chaincode is a program, written in Go, Node.js, or Java that implements a prescribed interface. Chaincode runs in a separate process from the peer and initializes and manages the ledger state through transactions submitted by applications.

```go
package main

import (
  "encoding/json"
  "fmt"
  "log"

  "github.com/hyperledger/fabric-contract-api-go/contractapi"
)

// SmartContract provides functions for managing an Asset
type SmartContract struct {
  contractapi.Contract
}
```

Fabric Example – Asset creation and updations.

Consider a car dealership setting up hyperledger to manage and transact with customers.

```go
// Asset describes basic details of what makes up a simple asset
type Car struct {
  Name          string `json:"name"`
  Color         string `json:"color"`
  Manufactyear  int    `json:"manctYear"`
  Owner         string `json:"owner"`
  BasePrice     int    `json:"basePrice"`
}
```

Demo-1

- https://hyperledger-fabric.readthedocs.io/en/latest/install.html
- Asset Example: https://hyperledger-fabric.readthedocs.io/en/latest/test_network.html
- Real World Example: https://github.com/Research-Tools-PAVT/fabric-verify.git (This is an example fabric code that I worked on earlier for an organization. Some parts of it have been open-sourced. There are other details that have not been added to this repo i.e. is closed-source.)