Hyperledger Fabric Architecture

Mar. 2018

Sangmoon Oh *(halfface@chollian.net)*

ToC

I. **Background of Enterprise Blockchain**

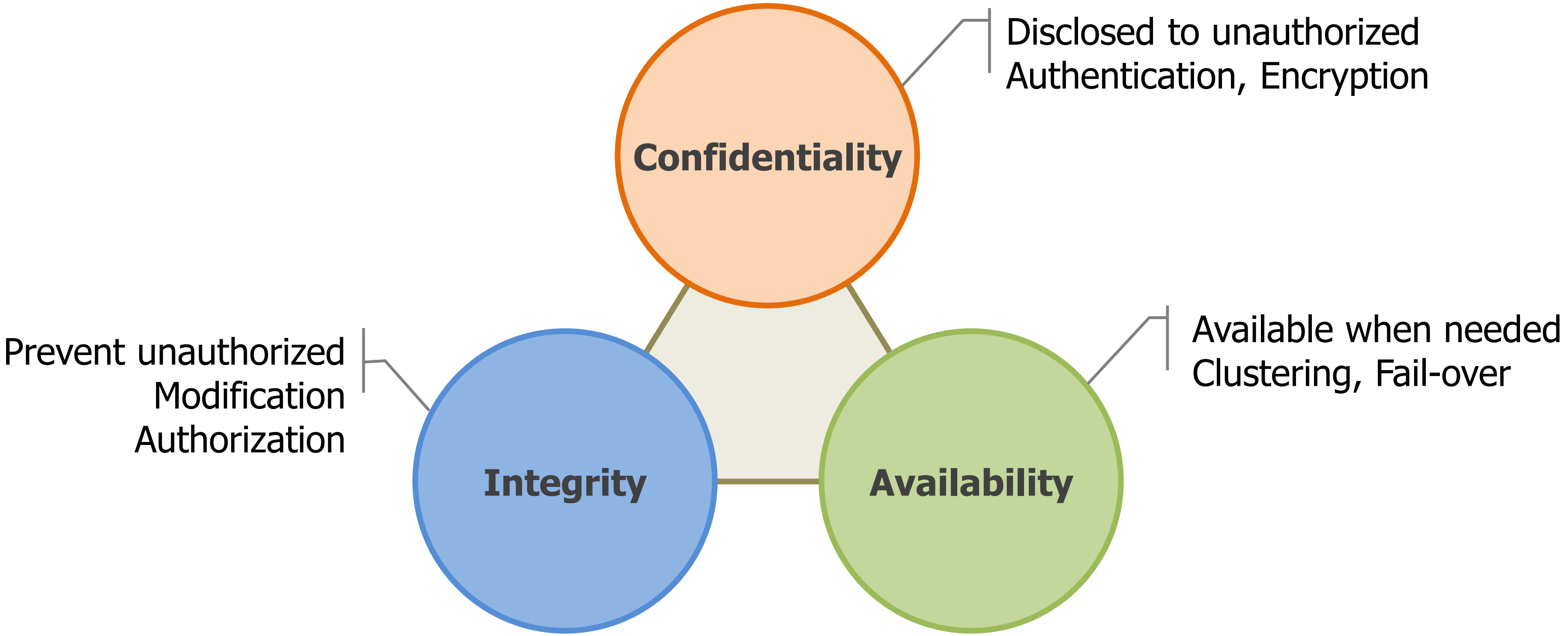
II. **Basic Architecture of Hyperledger Fabric**

III. **Fabric Network Provisioning**

IV. **Architectural Issues of Hyperledger Fabric**

I. Background of Enterprise Blockchain

Information Security

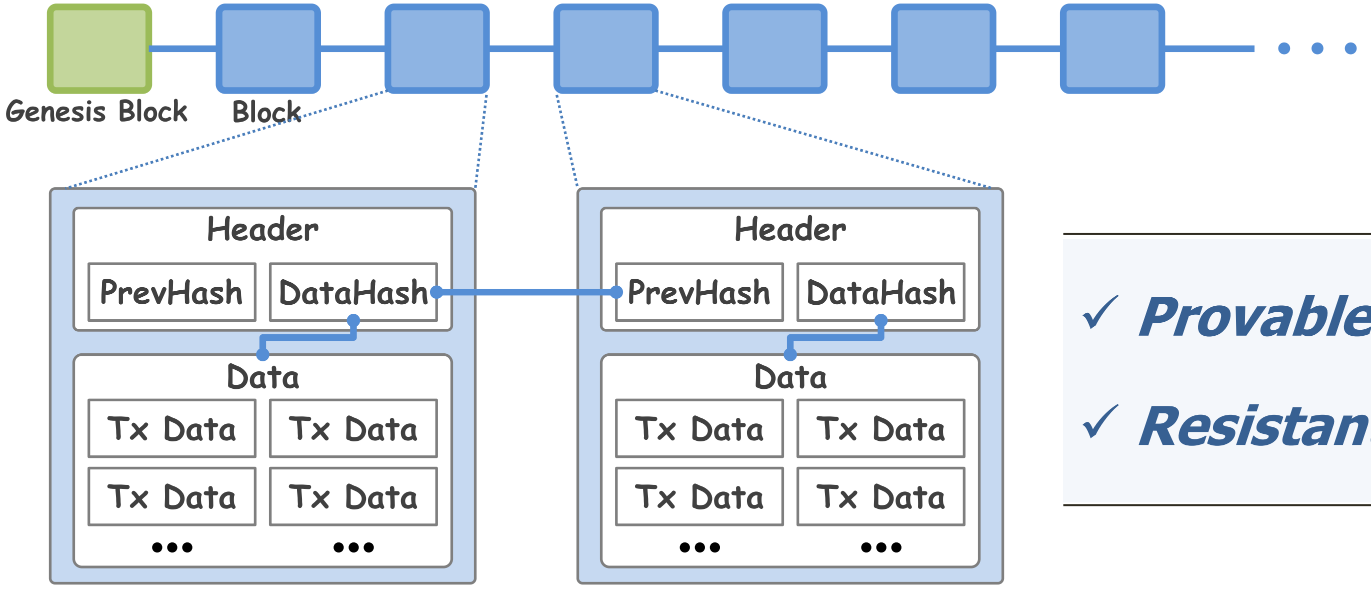


Disclosed to unauthorized
Authentication, Encryption

Prevent unauthorized
Modification
Authorization

Available when needed
Clustering, Fail-over

Public Blockchain / Chain of Block



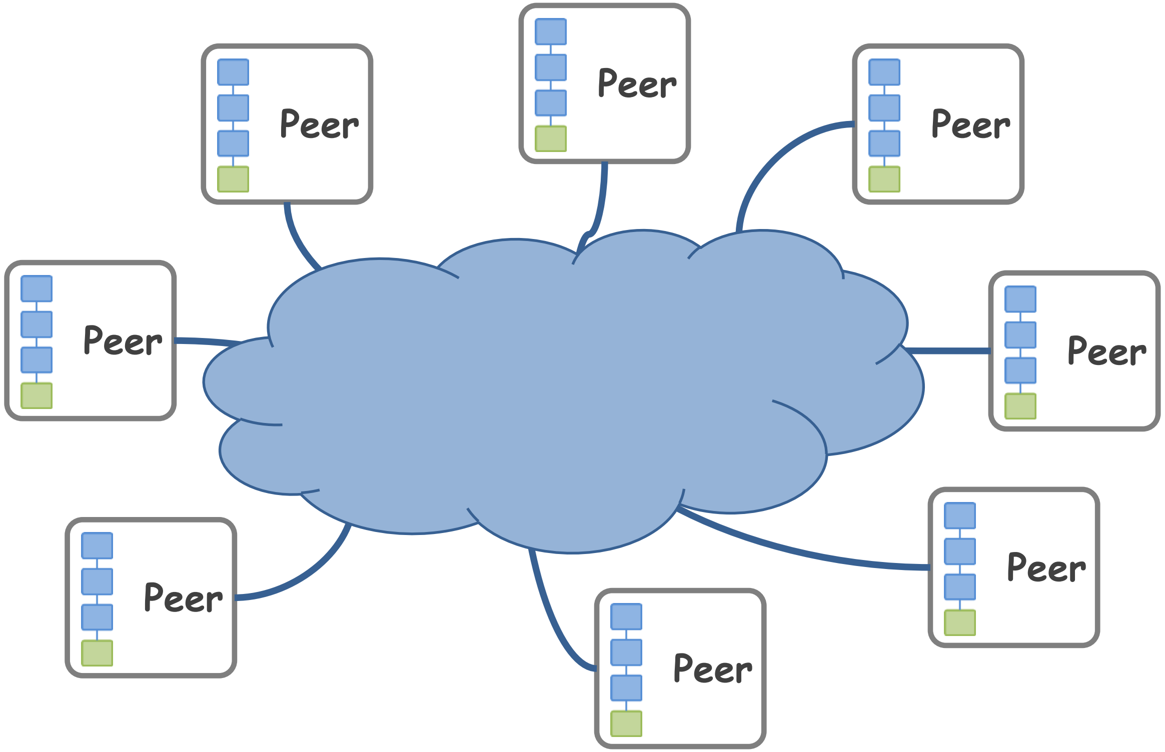
$$block_n.prevHash = hash(block_{n-1}.header)$$

$$block_n.dataHash = hash(block_n.data)$$

✓ ***Provable Integrity***

✓ ***Resistant to Modification***

Public Blockchain / Distributed Ledger



- ***Replicated***
- ***Synchronized***
- ***Decentralized***
- ***Open***

✓ ***Extreme Availability***

✓ ***Poor Confidentiality***

Public Blockchain / Decentralized Consensus

https://learn.onemonth.com/proof-of-work-vs-proof-of-stake/

https://depositphotos.com/search/bitcoin-mining.html?qview=34106323

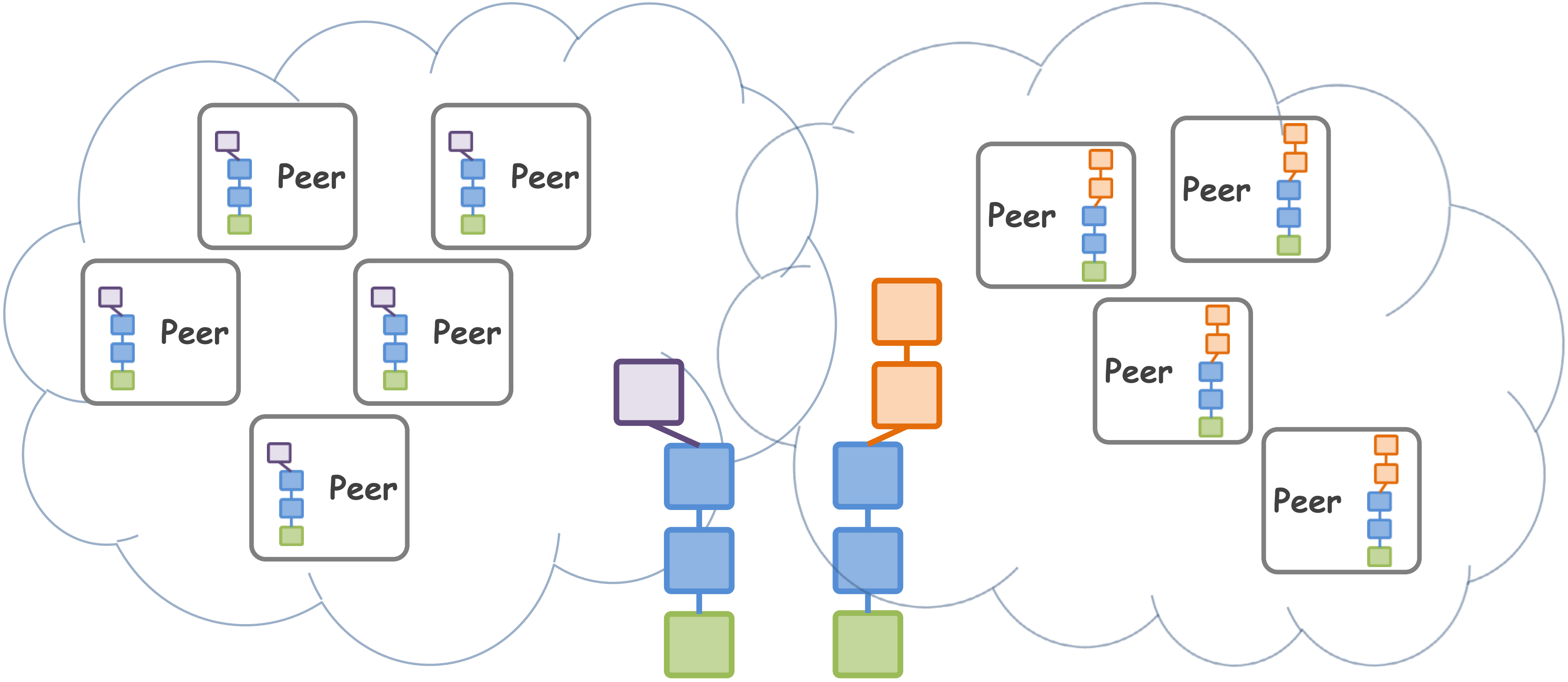
- ***Decentralized Consensus***
- ***PoW, PoS, DPoS, …***
- ***Serialized Change***

↓

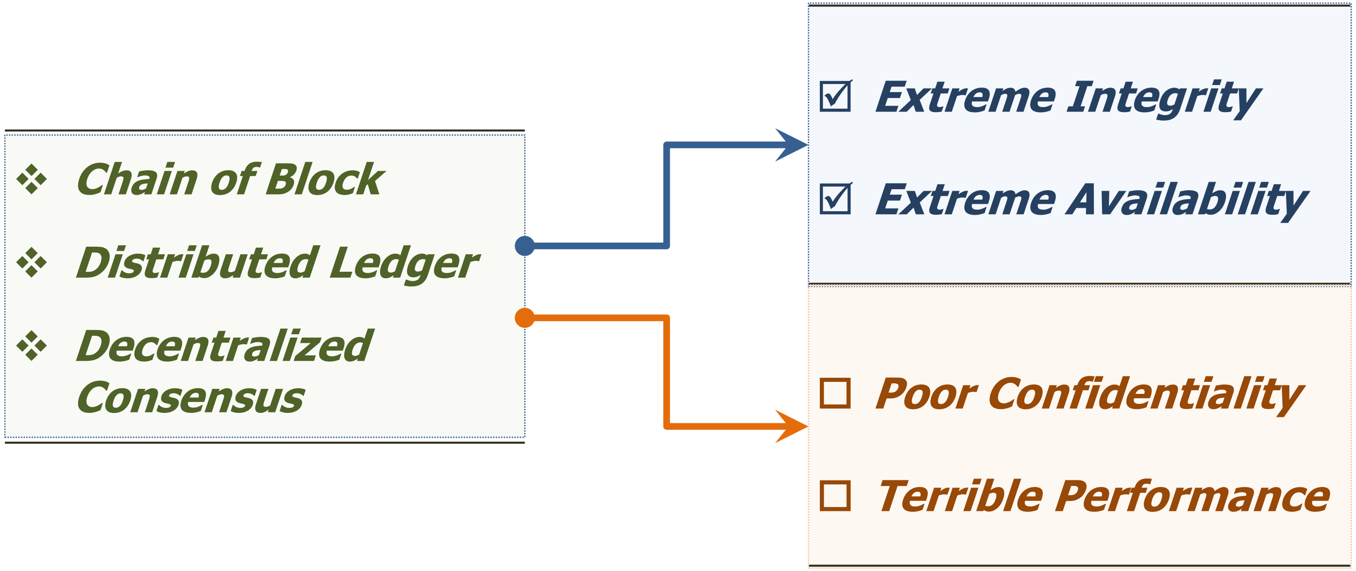
✓ ***Extreme Integrity***

✓ ***Terrible Performance***

Proof-of-Work • Mining : Can Make Conflict and Need Merge



Public Blockchain / Summary



- ❖ *Chain of Block*
- ❖ *Distributed Ledger*
- ❖ *Decentralized Consensus*

☑ ***Extreme Integrity***

☑ ***Extreme Availability***

□ ***Poor Confidentiality***

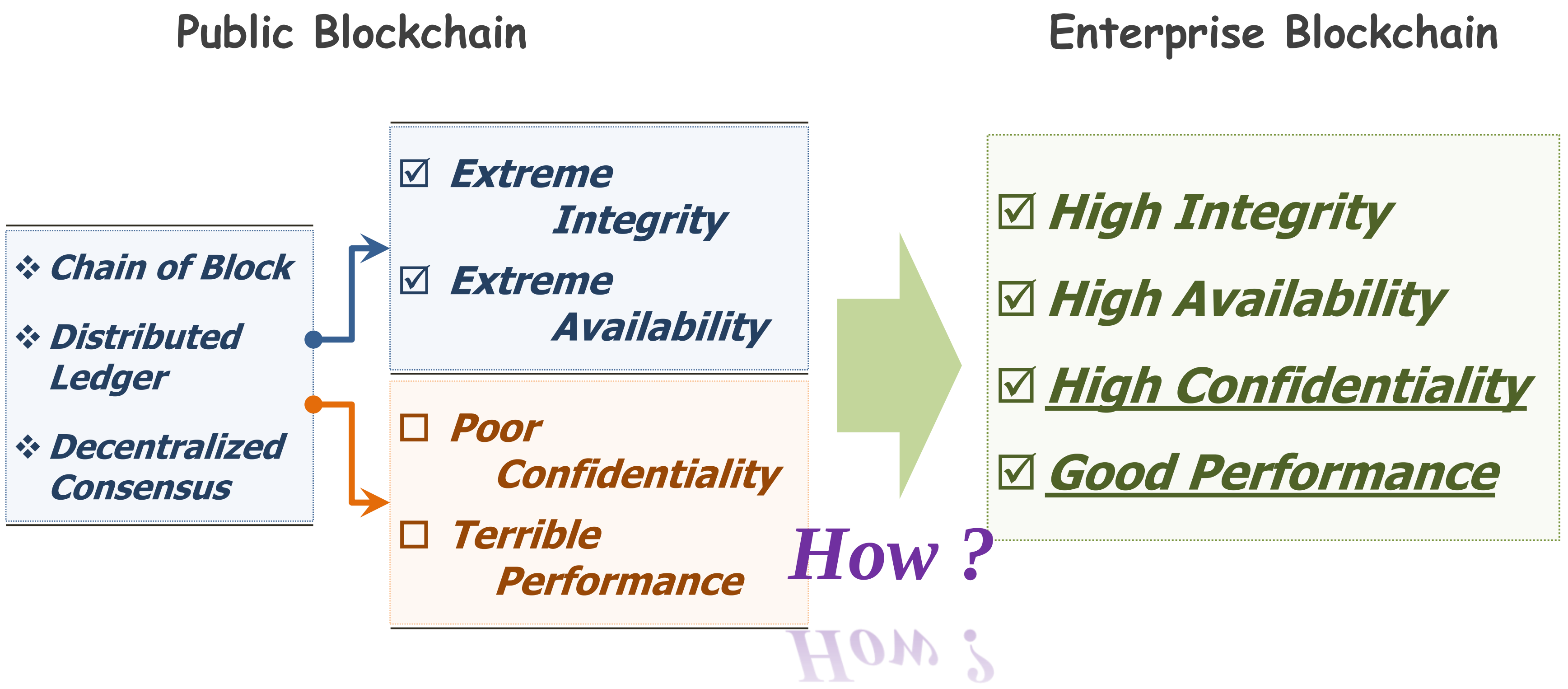
□ ***Terrible Performance***

ethereum

bitcoin

ripple

Enterprise Blockchain / Requirements



Public Blockchain

Enterprise Blockchain

- ❖ *Chain of Block*
- ❖ *Distributed Ledger*
- ❖ *Decentralized Consensus*

☑ ***Extreme Integrity***

☑ ***Extreme Availability***

□ ***Poor Confidentiality***

□ ***Terrible Performance***

☑ ***High Integrity***

☑ ***High Availability***

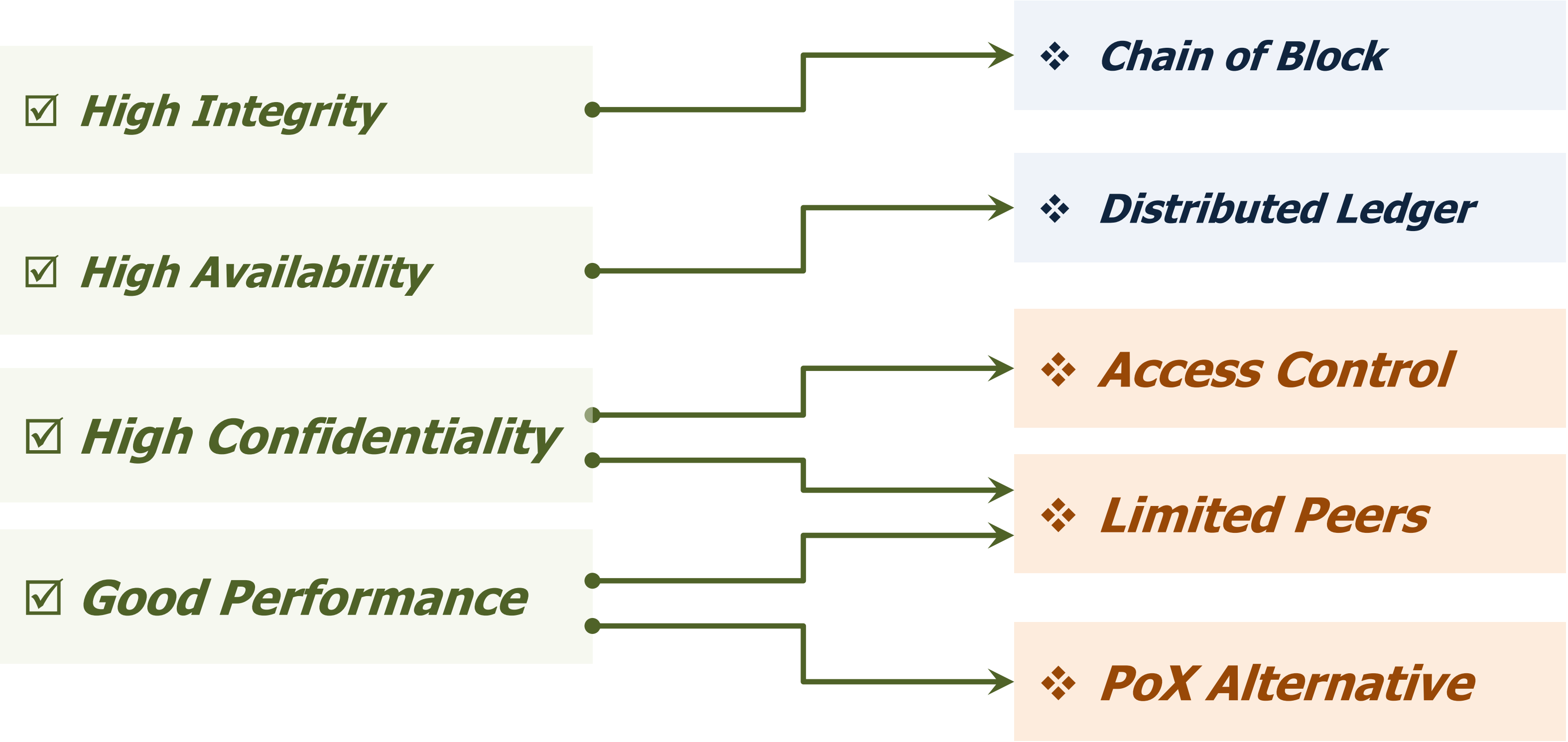
☑ ***High Confidentiality***

☑ ***Good Performance***

How ?

How ?

Enterprise Blockchain / Core Strategy



☑ *High Integrity*

☑ *High Availability*

☑ *High Confidentiality*

☑ *Good Performance*

❖ *Chain of Block*

❖ *Distributed Ledger*

❖ *Access Control*

❖ *Limited Peers*

❖ *PoX Alternative*

II. Basic Architecture of Hyperledger Fabric

Hyperledger Fabric / Intro

HYPERLEDGER

HYPERLEDGER
FABRIC

☑ ***Permissioned Blockchain***

☑ ***Open-source*** *(Apache-2.0)*

☑ ***led by IBM and Linux Foundation***

☑

1.1.0	:	*Mar 2018*
1.0.0	:	*Jul 2017*
1.0.0-alpha	:	*Mar 2017*
0.6.0-preview	:	*Sep 2016*

☑ ***implemented in Go***

☑ ***Identity management***
- ***Membership service***

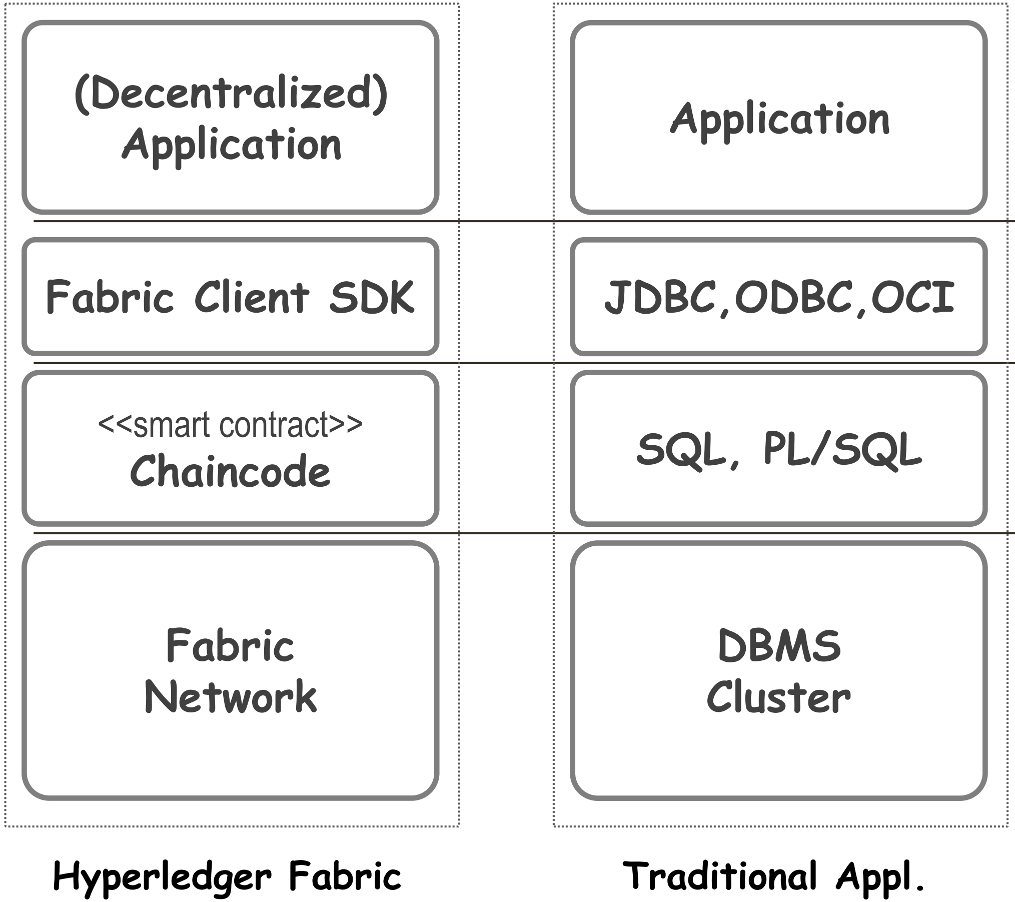
☑ ***Privacy and confidentiality***

☑ ***Chaincode as smart contract***

☑ ***Endorsement policy***

☑ ***CouchDB as state database***

Fabric / Full Stack



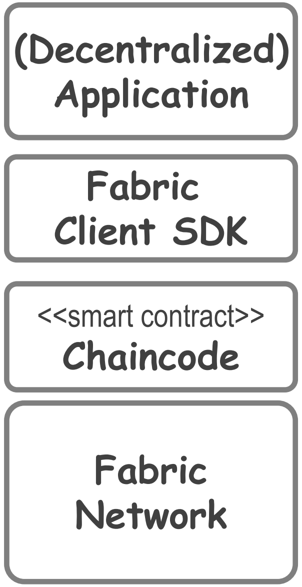
- ✓ Visualize data
- ✓ Calculate and analyze data
- ✓ Generate data

- ✓ General library and connector
- ✓ Low-level API

- ✓ Define data structure
- ✓ Manipulate data (CRUD)

- ✓ Persist data
- ✓ Share data
- ✓ Basic I/O

Fabric / Projects



(Decentralized) Application

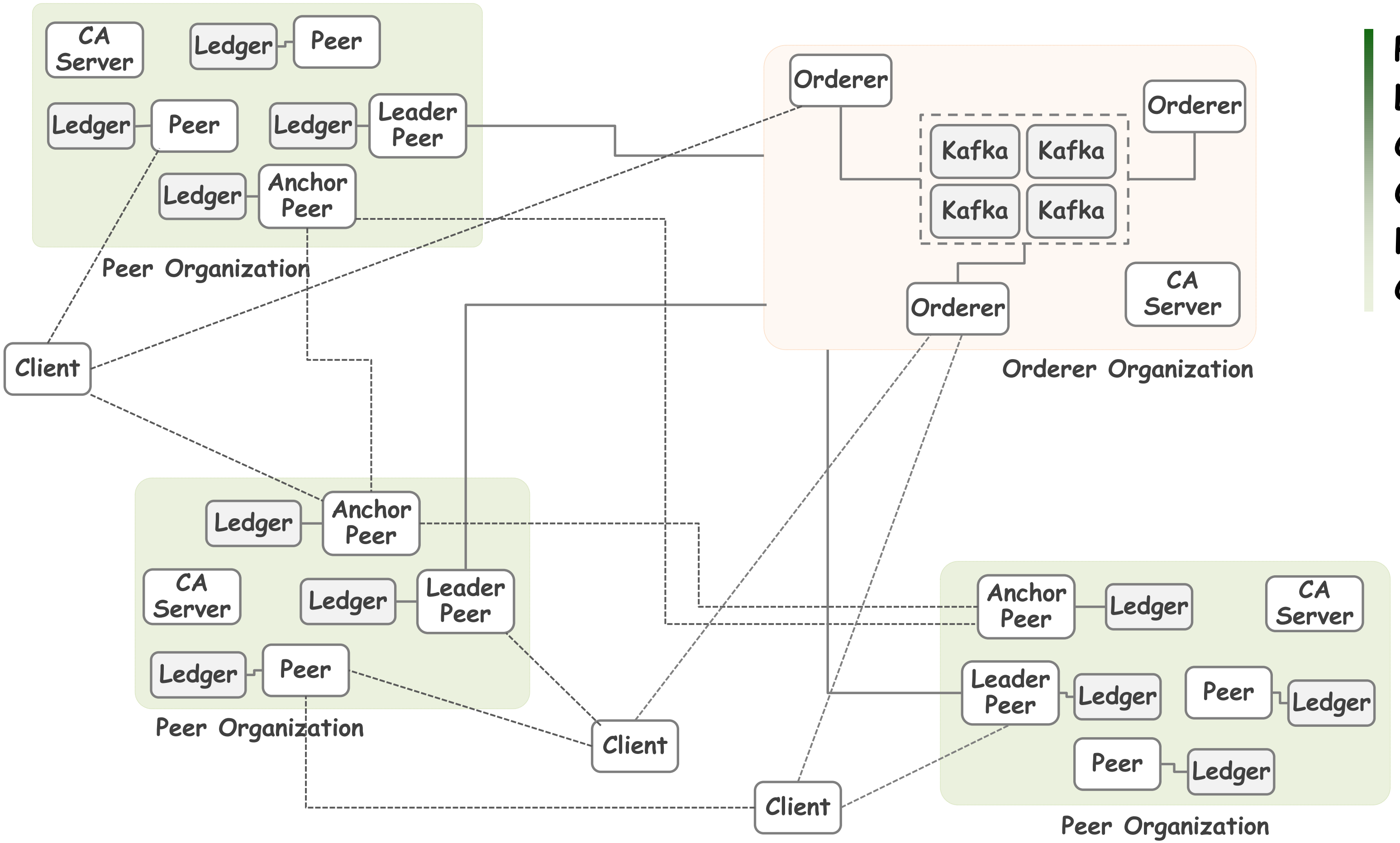
Fabric Client SDK

<<smart contract>> Chaincode

Fabric Network

Project	Description	Commits	Releases
fabric	Platform, Network, Runtime	5,843	18
fabric-samples	Sample codes		
fabric-sdk-node	Client SDK in Node.js	758	14
fabric-sdk-java	Client SDK in Java	295	6
fabric-sdk-go	Client SDK in Go	779	3
fabric-sdk-py	Client SDK in Python	221	1
fabric-sdk-rest	Client SDK in REST	110	0
fabric-chaincode-node	Chaincode in Node.js	60	3
fabric-chaincode-java	Chaincode in Java	24	0
composer	Application development framework	4,749	75

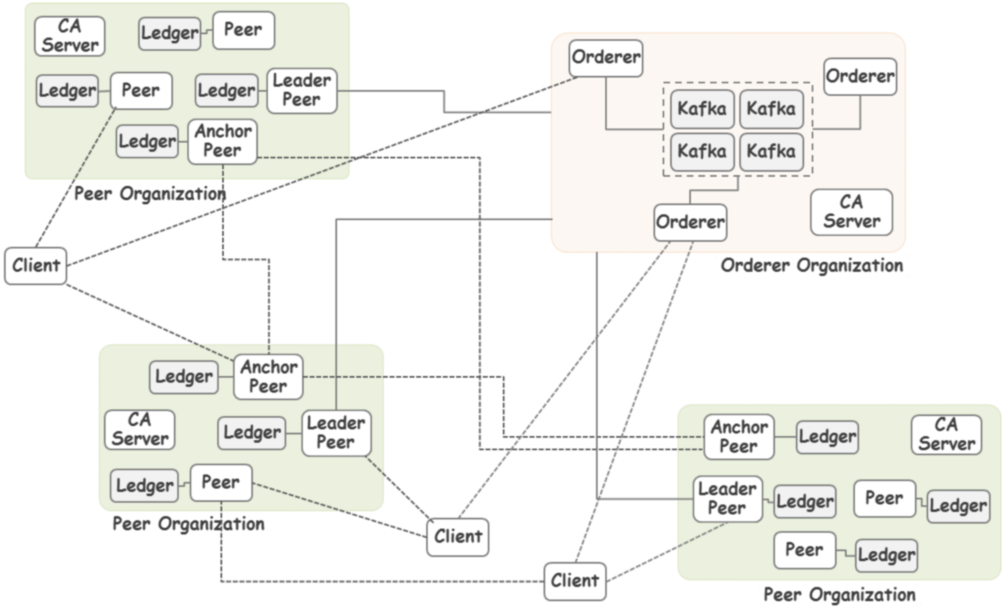
Fabric Network / Software Architecture



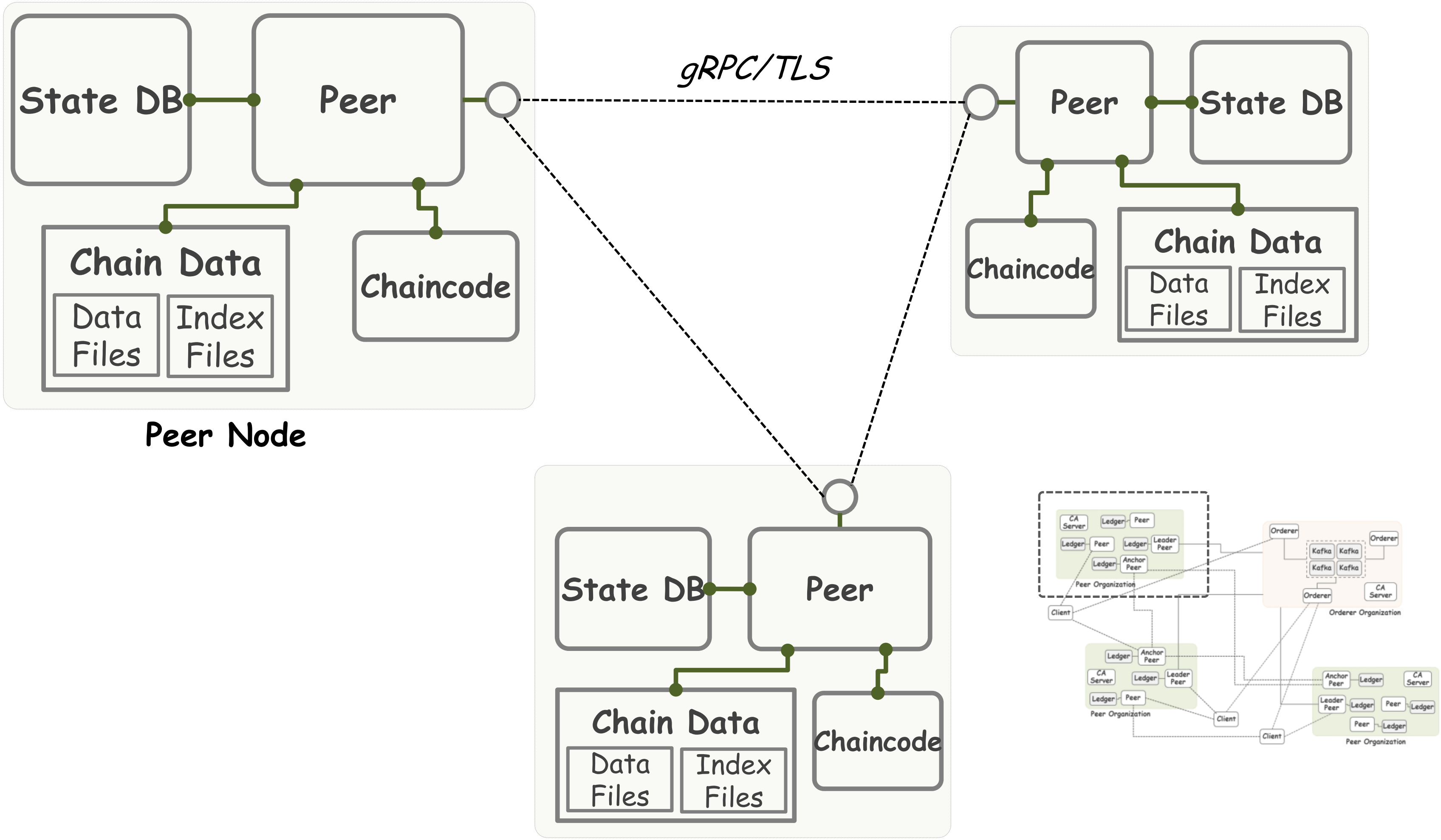
Peer
Ledger
Client
Orderer
Kafka
CA Server

Fabric Network / Key Concepts

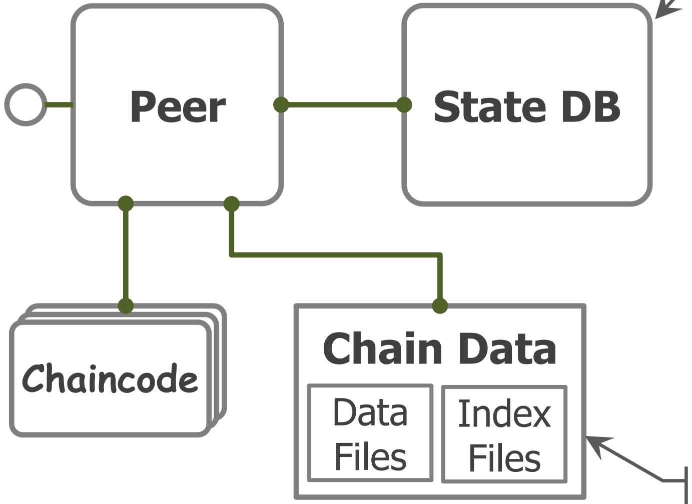
- □ **Peer and Leger**
- □ **Orderer Cluster**
- □ **Consensus**
- □ **Membership Service**
- □ **Channel**



Fabric Network / Peer



Fabric Network / Ledger



current/latest state data

(k, v_n)
(k', v'_m)
(k'', v''_s)

```
("BTC", {price: 10,000,000, at: 3/27})
("ETH", {price: 800,000, at: 3/27})
```

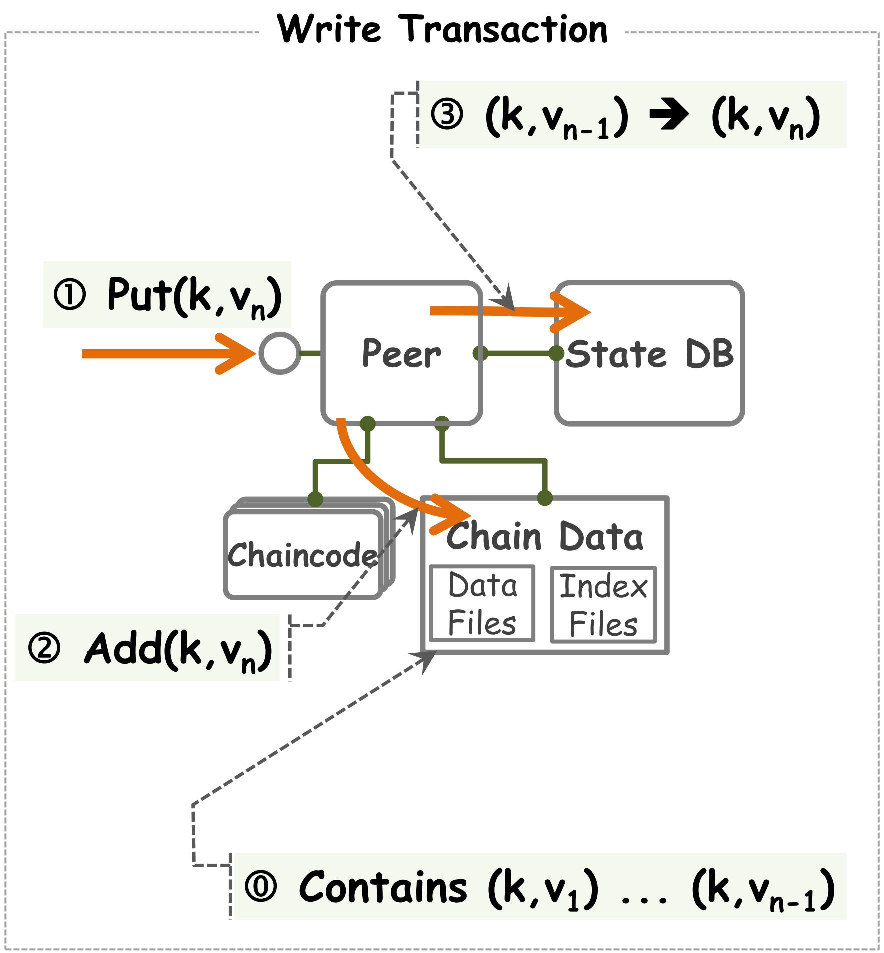
all data

$(k, v_1), (k, v_2), (k, v_3), \ldots. (k, v_n)$
$(k', v'_1), (k', v'_2), (k', v'_3), \ldots. (k', v'_m)$
$(k'', v''_1), (k'', v''_2), (k'', v''_3), \ldots. (k'', v''_s)$

```
("BTC", {price: 10,000,000, at: 3/27})
("BTC", {price:  9,500,000, at: 3/26})
("BTC", {price:  9,000,000, at: 3/25})
("BTC", {price:  8,000,000, at: 3/24})
                ...
```

Fabric Network / Data Access

Write Transaction



③ (k, v_{n-1}) ➔ (k, v_n)

① Put(k, v_n)

Peer

State DB

Chaincode

Chain Data

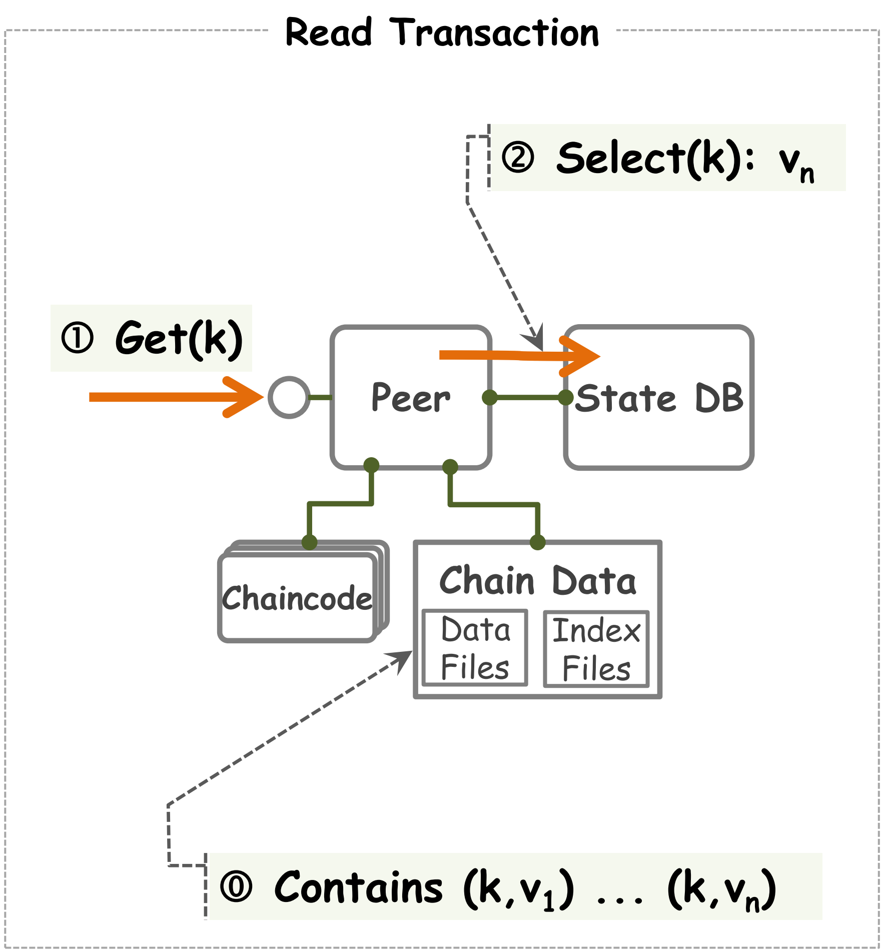
Data Files

Index Files

② Add(k, v_n)

⓪ Contains (k, v_1) … (k, v_{n-1})

Read Transaction



② Select(k): v_n

① Get(k)

Peer

State DB

Chaincode

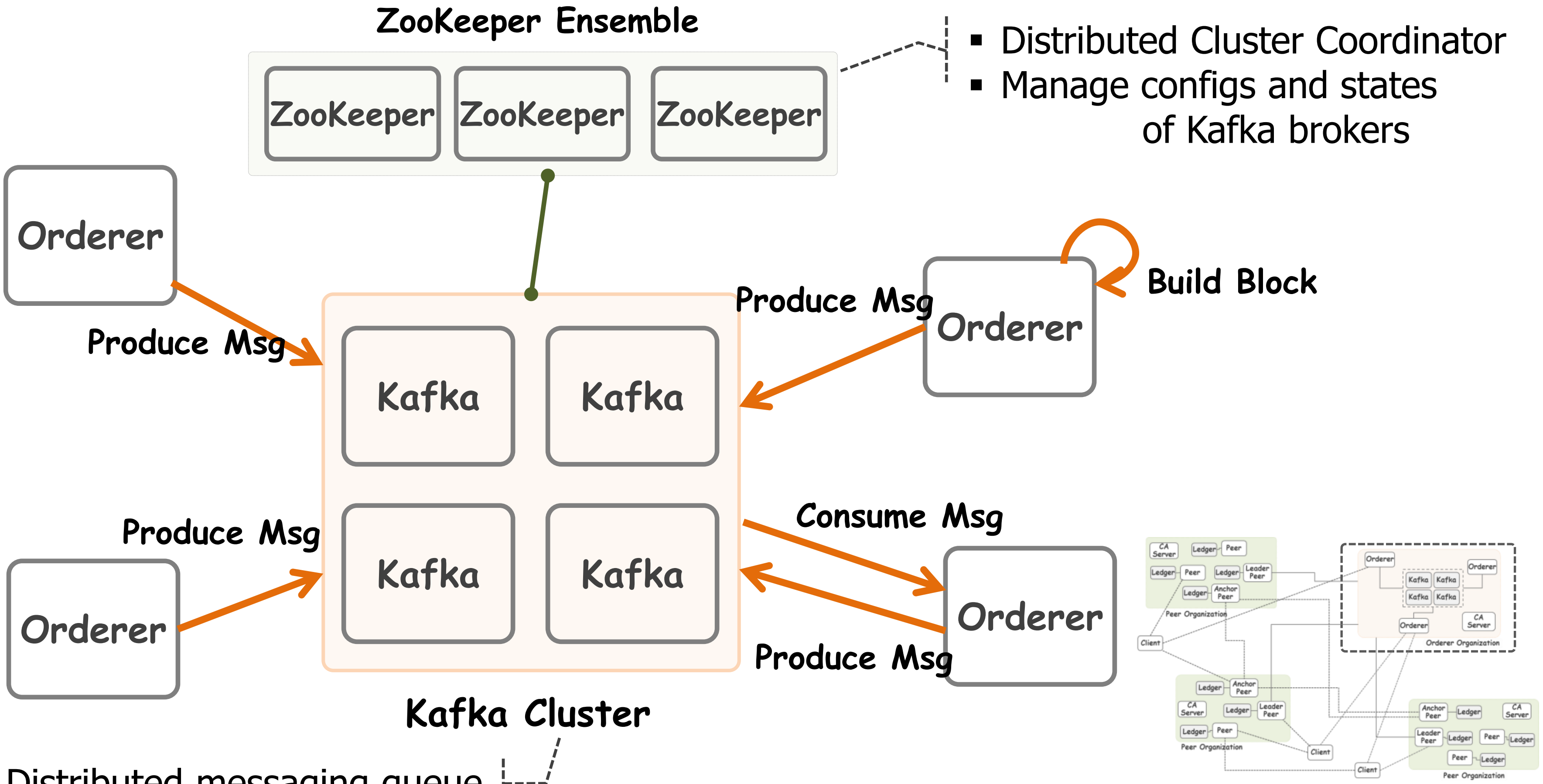
Chain Data

Data Files

Index Files

⓪ Contains (k, v_1) … (k, v_n)

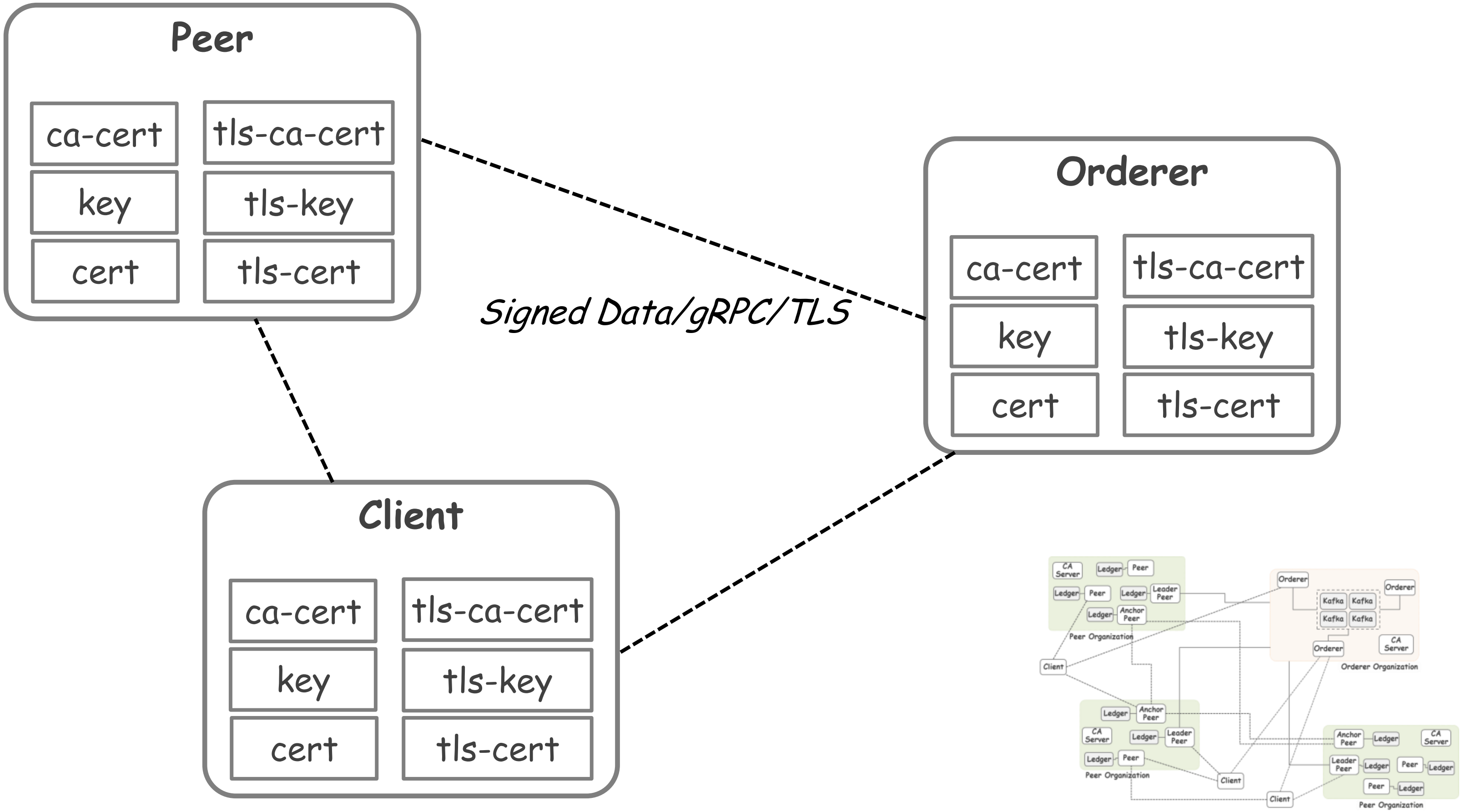
Fabric Network / Orderer Cluster



- Distributed Cluster Coordinator
- Manage configs and states of Kafka brokers

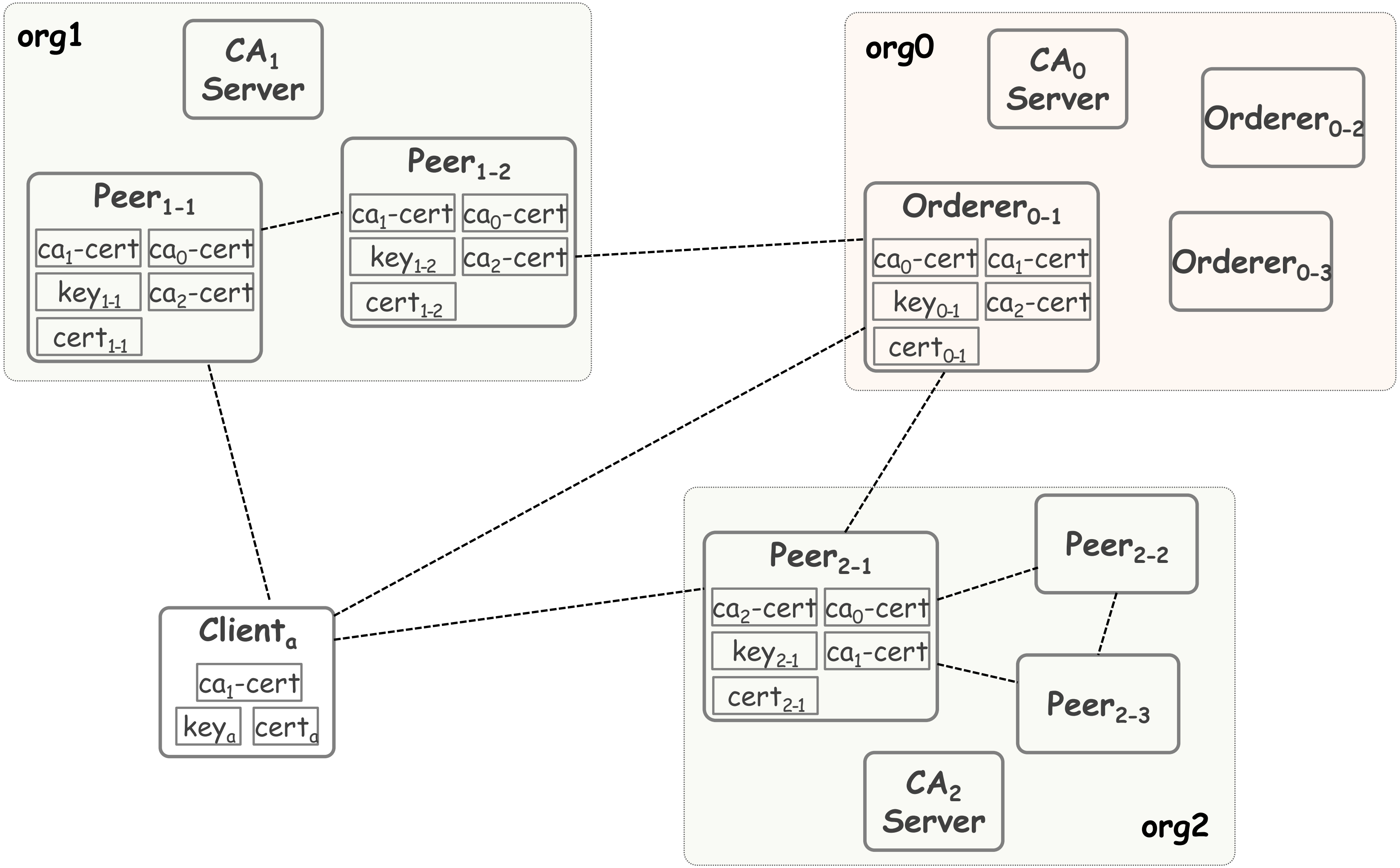
Distributed messaging queue
Queuing transactions

Fabric Network / Membership Service

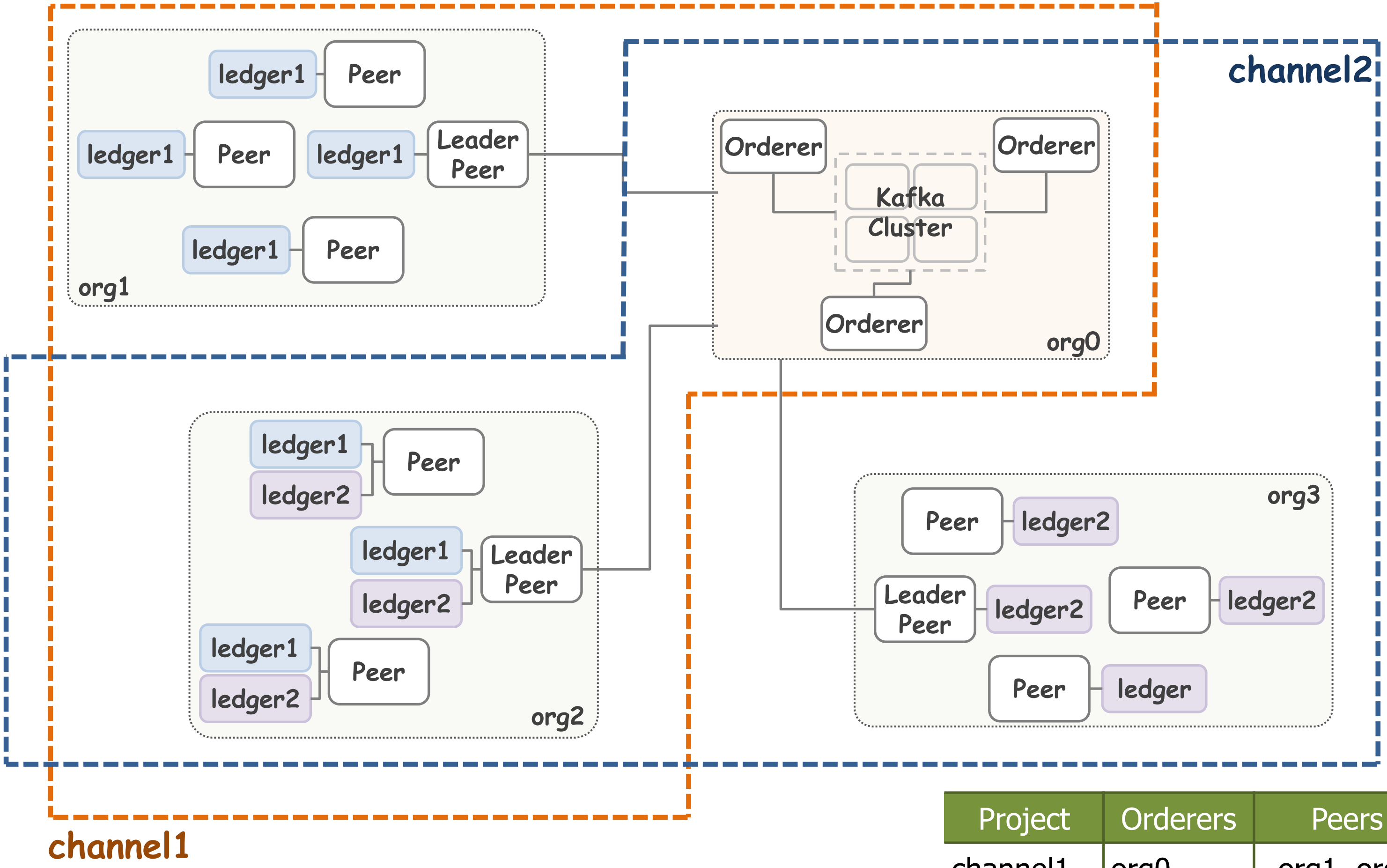


※ Hyperledger Fabric V1.0: Block Structure

Fabric Network / Membership Service

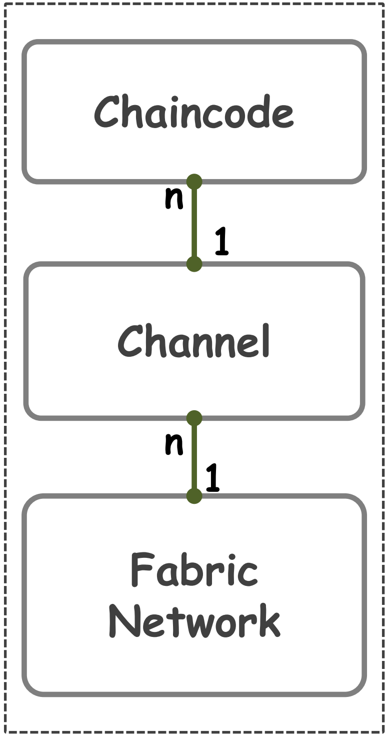


Fabric Network / Channel

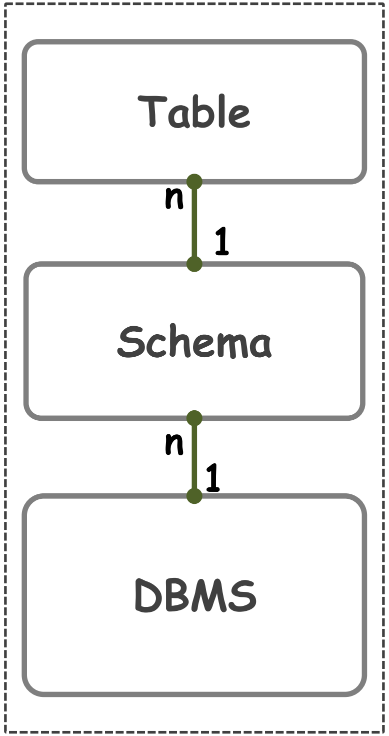


Project	Orderers	Peers	Ledger
channel1	org0	org1, org2	ledger1
channel1	org0	org2, org3	ledger2

Fabric Network / Channel (2)



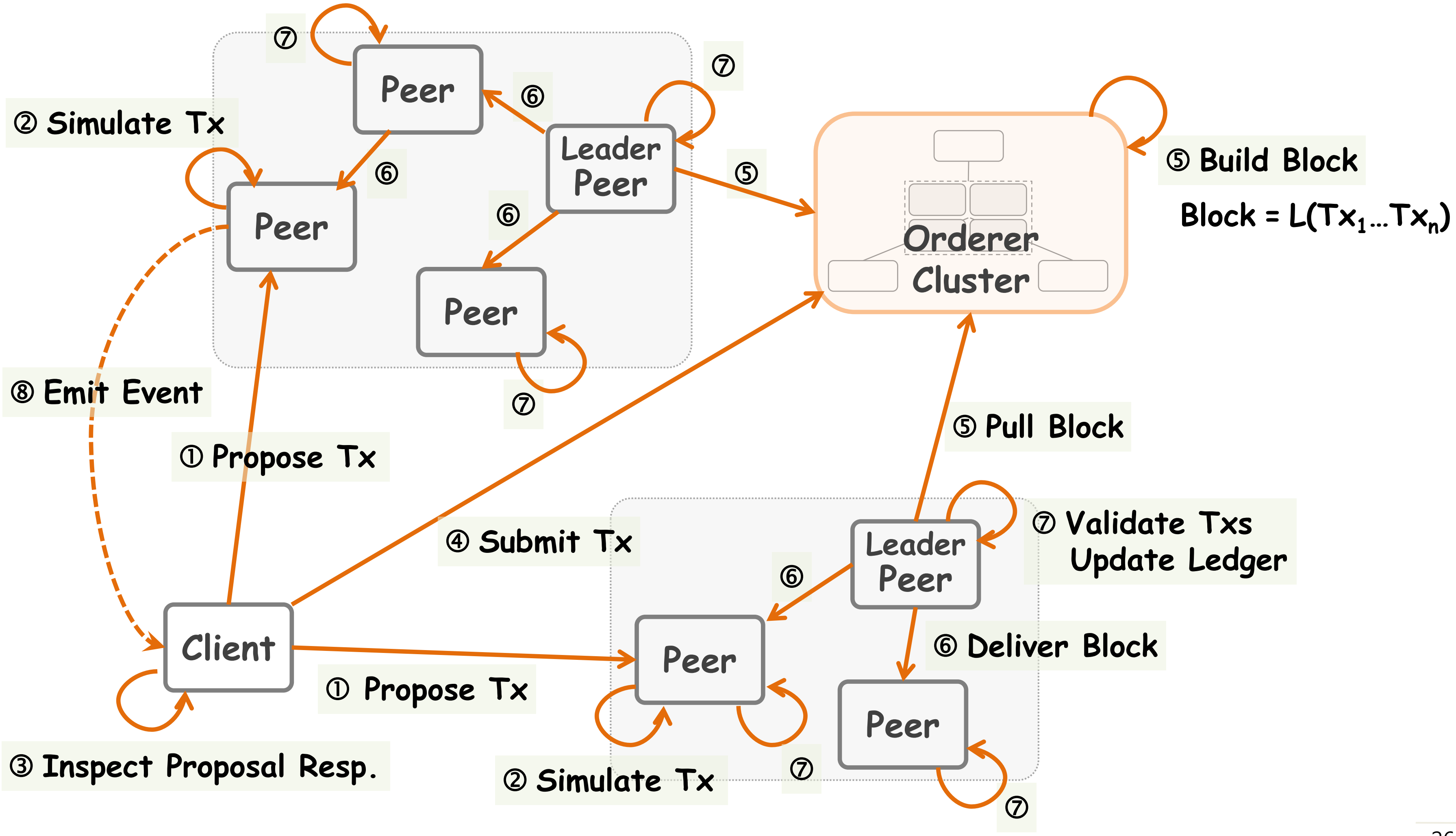
Hyperledger Fabric



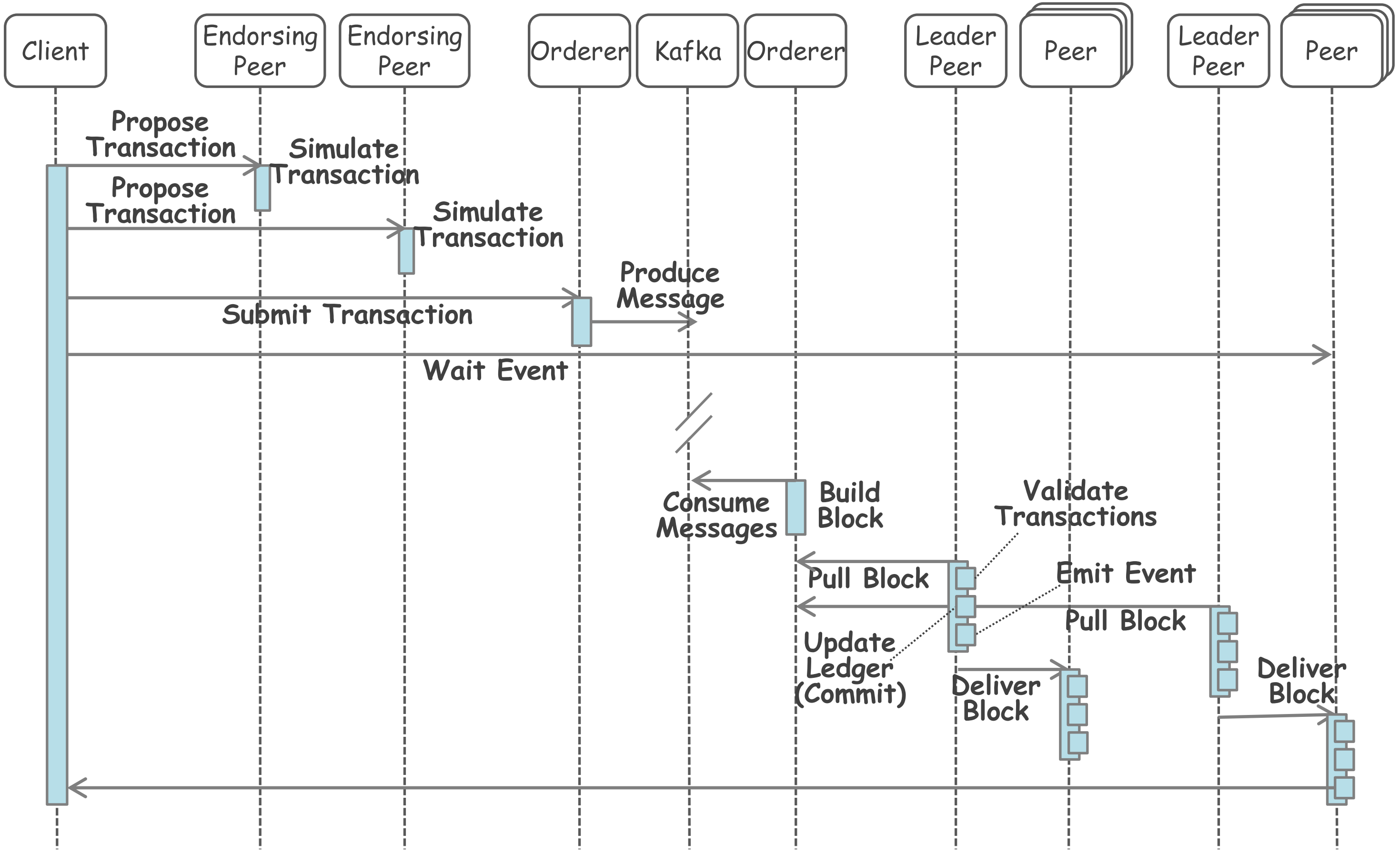
RDBMS

Channel	Chaincode
Supply Management	▪ Part ▪ Supplier ▪ Warehouse ▪ Procurement
Order Management	▪ Customer ▪ Product ▪ Order ▪ Delivery

Fabric / Consensus / Interaction



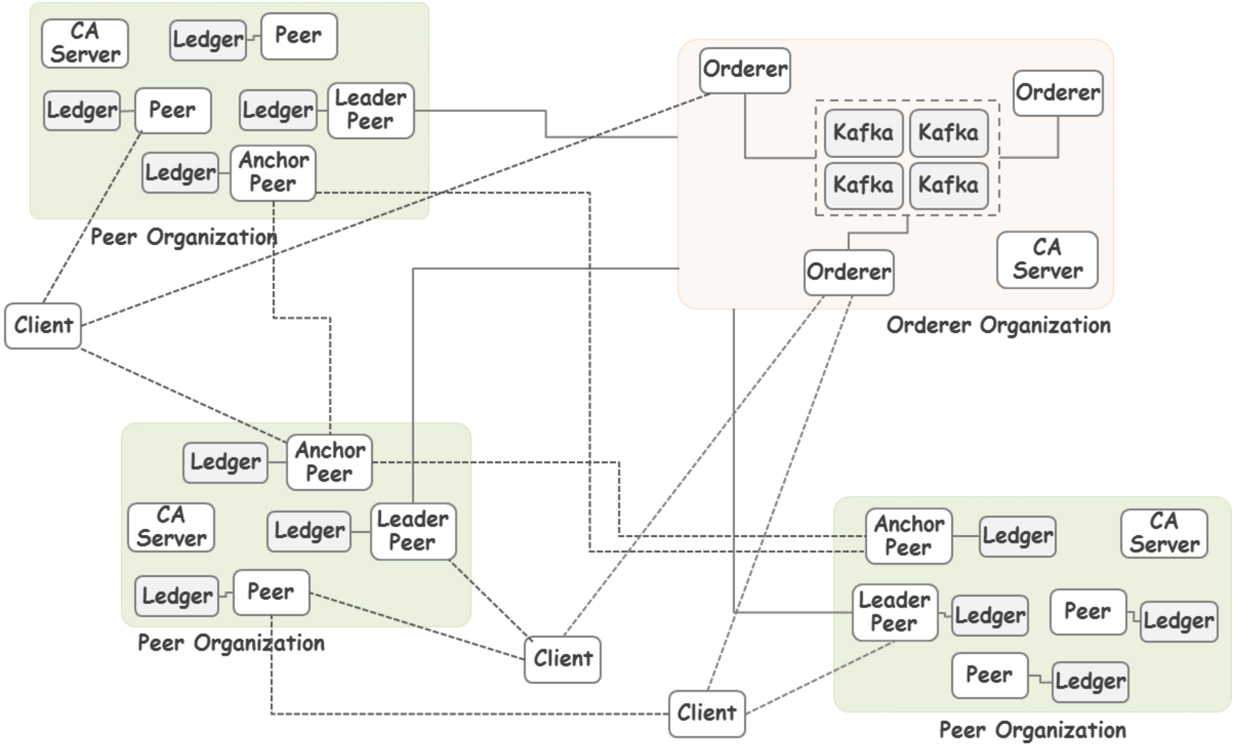
Fabric / Consensus / Sequence



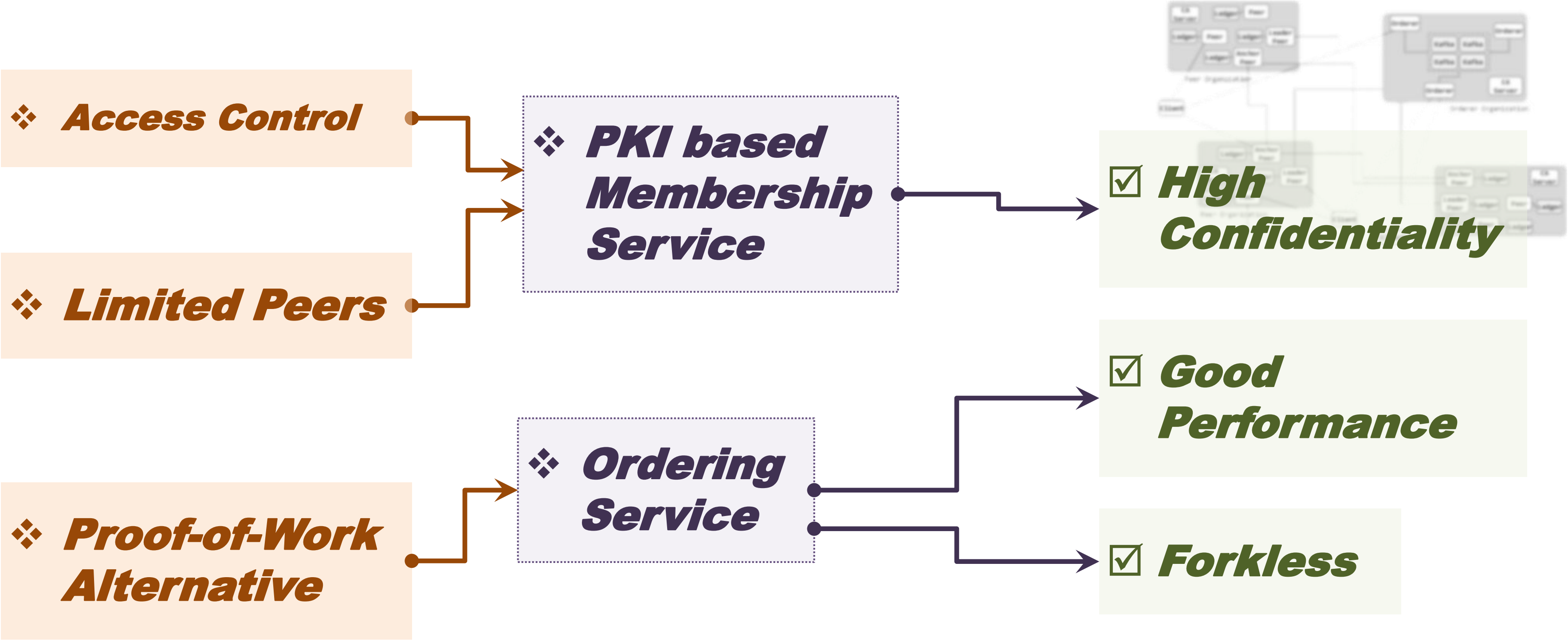
1) Propose Tx
2) Verify Tx
3) Simulate Tx
4) Submit Tx
5) Enqueue Tx
6) Dequeue Tx
7) Build Block
8) Deliver Block
9) Validate Tx
10) Commit Tx
11) Emit Ev

Fabric Network / Software Architecture / Revisited

- □ Network = Channel$^{1...n}$
- □ Channel = Peer Org.$^{1...n}$ + Orderer Cluster + Ledger + Chaincode$^{1...m}$
- □ Peer Org. = Leader Peer + Anchor Peer + Peer$^{0...n}$ + CA Server



Fabric / Features Revisited



❖ *Access Control*

❖ *Limited Peers*

❖ *Proof-of-Work Alternative*

❖ *PKI based Membership Service*

❖ *Ordering Service*

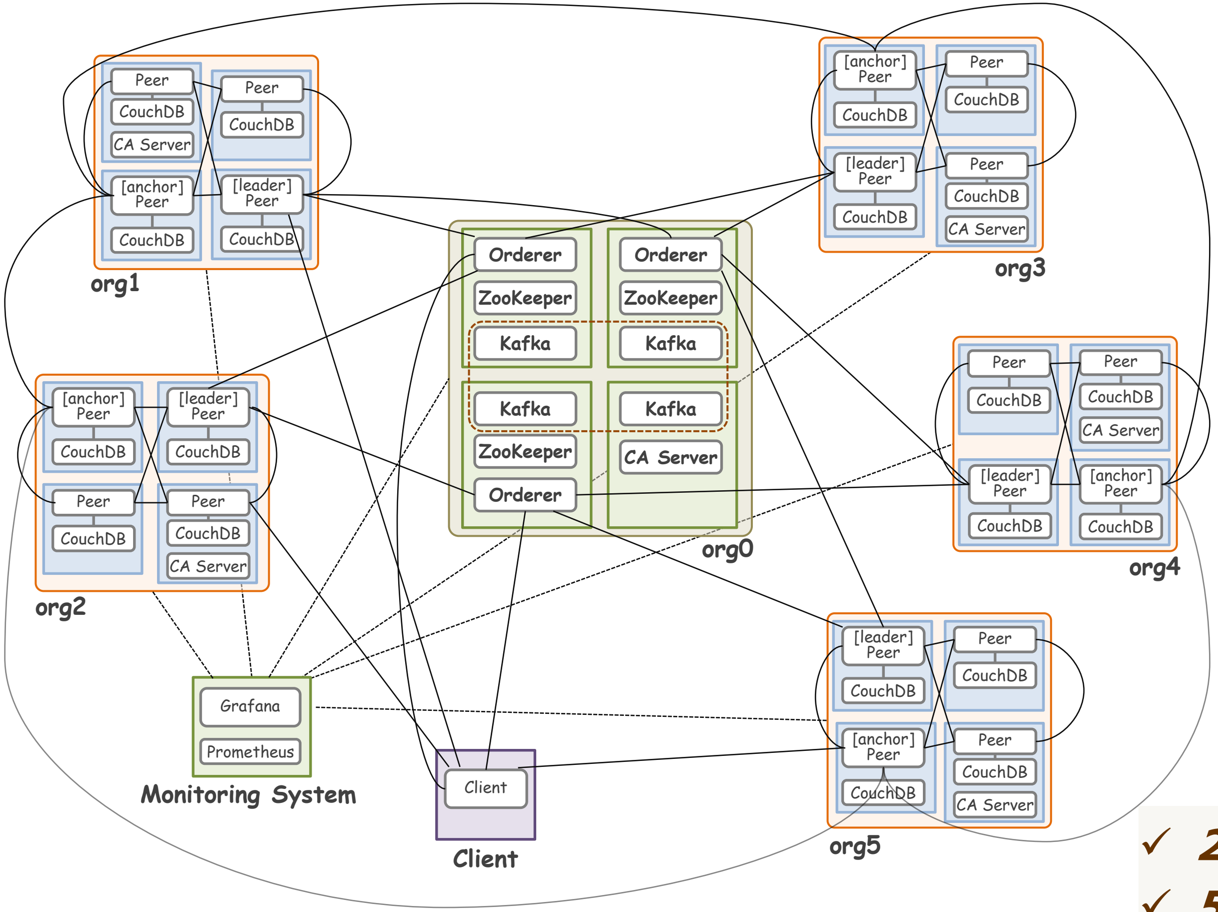
☑ *High Confidentiality*

☑ *Good Performance*

☑ *Forkless*

III. Fabric Network Provisioning

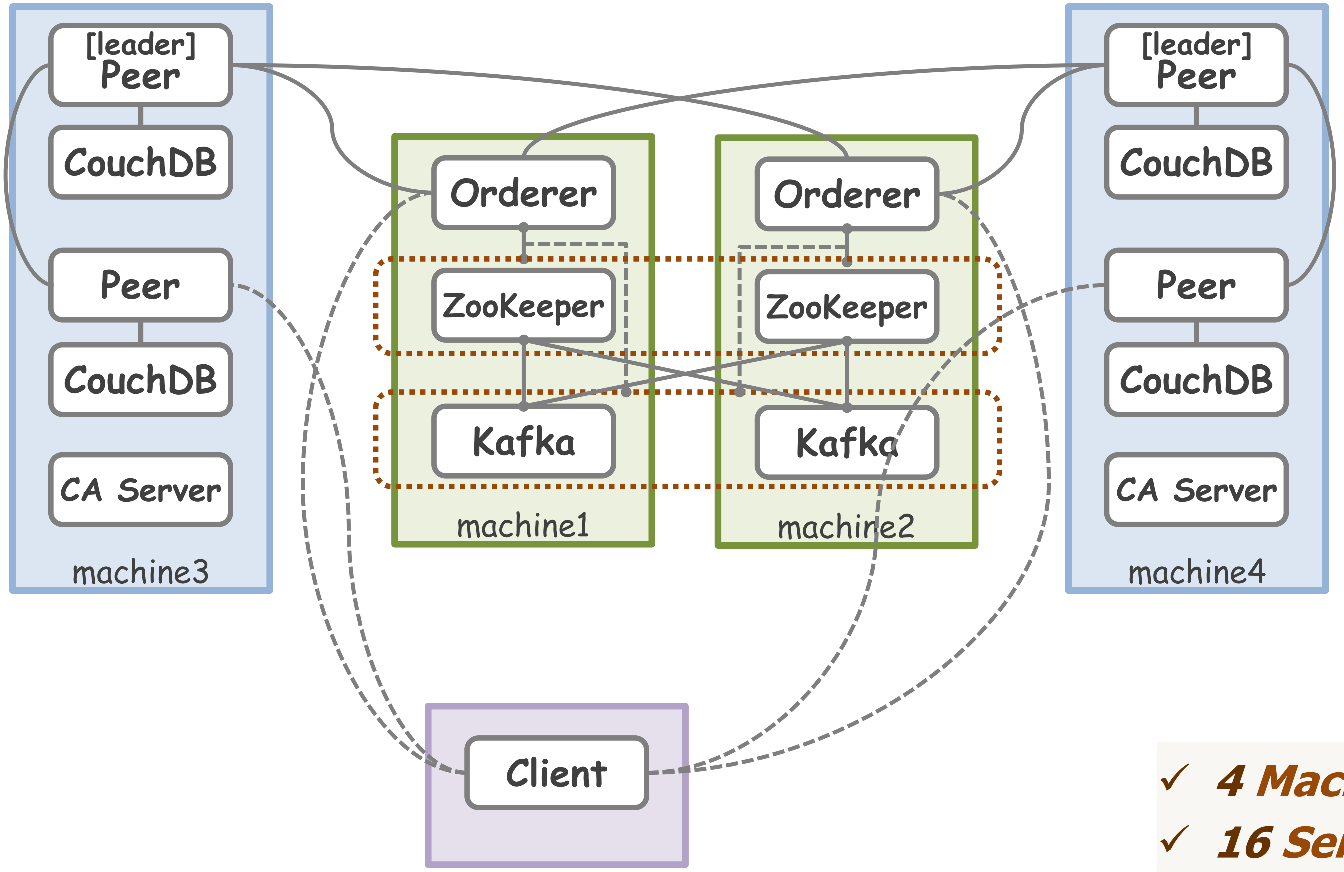
Fabric Network / Production Example



Category	#of Instance
Host	25
ZooKeeper	3
Kafka	4
Orderer	3
Peer	20
CouchDB	20
CA Server	6
Prometheus	1
Grafana	1

- ✓ ***25 Machines***
- ✓ ***58 Services of 8 Types***

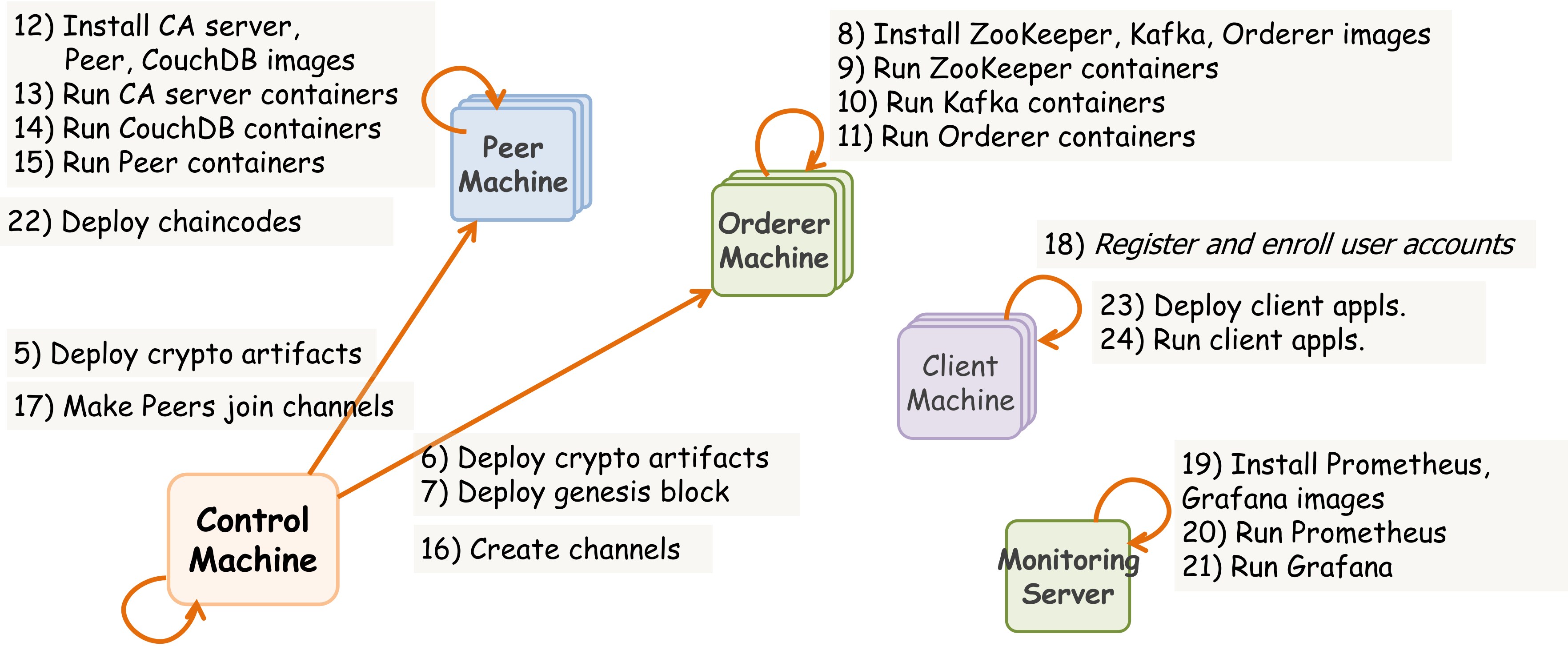
Fabric Network / PoC Example



Category	#of Instance
Host	4
ZooKeeper	2
Kafka	2
Orderer	2
Peer	4
CouchDB	4
CA Server	2

- ✓ ***4 Machines***
- ✓ ***16 Services of 6 Types***

Fabric Network / Provisioning Environment



12) Install CA server,
Peer, CouchDB images
13) Run CA server containers
14) Run CouchDB containers
15) Run Peer containers

22) Deploy chaincodes

8) Install ZooKeeper, Kafka, Orderer images
9) Run ZooKeeper containers
10) Run Kafka containers
11) Run Orderer containers

18) *Register and enroll user accounts*

23) Deploy client appls.
24) Run client appls.

5) Deploy crypto artifacts

17) Make Peers join channels

6) Deploy crypto artifacts
7) Deploy genesis block

16) Create channels

19) Install Prometheus,
Grafana images
20) Run Prometheus
21) Run Grafana

1) Install Fabric/Fabric CA tools
2) Generate crypto artifacts(keys, certs)
3) Generate genesis block
4) Generate config transactions

- Building Your First Network
- hyperledger/fabric-samples/first-network/
- hyperledger/fabric/examples/e2e_cli/

Fabric Network / Provisioning Process

1. *Install Docker, Node.js and so on.*
2. *Install Fabric and Fabric CA tools.*
3. *Write down `cryptogen` input file.*
4. *Generate crypto artifacts using `cryptogen` tool.*
5. *Write down `configtxgen` input file.*
6. *Generate genesis block.*
7. *Generate channel config transactions.*
8. *Generate anchor update transactions.*
9. *Deploy crypto artifacts and genesis block.*

1. *Install Docker images (Fabric CA server, ZooKeeper, Kafka, Orderer, CouchDB, Peer, ...)*
2. *Run Fabric CA server containers*
3. *Run ZooKeeper containers*
4. *Run Kafka containers*
5. *Run Fabric Orderer containers*
6. *Run CouchDB containers*
7. *Run Fabric Peer containers*
8. *Create Fabric channels*
9. *Make Fabric Peers join the channels*
10. *Register and enroll user accounts*
11. *Deploy test chaincode and client appls.*
12. *Run test client appls.*

Fabric Network / Provisioning / Prerequisites

Recommended Runtime Environment for Fabric 1.1

Software	Version	Remarks
Ubuntu	16.04	• For all machines
Docker	17.06.2-ce	• For all Peer and Orderer machines • Get Docker CE for Ubuntu
Docker Composer	1.14.0	• For all Peer and Orderer machines • docker/compose/1.14.0
Node.js	8.10.0	• For all client application machine • Installing Node.js 8.10 on Debian and Ubuntu based Linux distributions
npm	5.6.0	• For all client application machine
Python	2.7	• For all client application machine
Go	1.9.4	• For all Peers • Fabric 1.1 : compiled with Go 1.9.2 • Installing Golang on Ubuntu

Fabric Network / Provisioning / Fabric·Fabric CA Tools

- ✓ Fabric/Fabric CA command-line tools for provisioning Fabric network
- ✓ Download Platform-specific Binaries
- ✓ `curl -sSL https://goo.gl/6wtTN5 | bash -s 1.1.0`
- ✓ https://goo.gl/6wtTN5

Tool	Description
`cryptogen`	• Generate keys and certificates for Fabric network.
`configtxgen`	• Create and inspect configuration related artifacts. - genesis block, channel creation tx, …

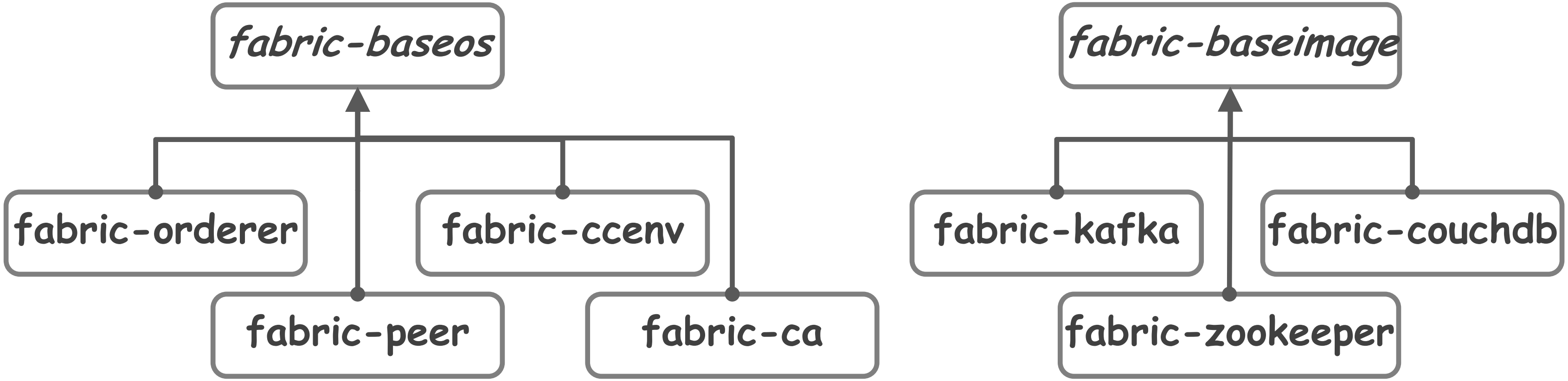
`peer`	• Operate a channel
	`(peer channel create|fetch|join|list|update|signconfigtx|getinfo)`
	• Operate a chaincode
	`(peer chaincode install|instantiate|invoke|...|upgrade|list)`
	• Log levels `(peer logging getlevel|setlevel|revertlevels)`
	• Operate a peer node `(peer node start|status)`

`fabric-ca-client`	• Register, enroll, reenroll or revoke Fabric CA identities

Fabric Network / Provisioning / Docker Containers

- Hyperledger Docker Repository
  - https://hub.docker.com/u/hyperledger/

- Hyperledger Fabric 1.1 Dockerfile
  - https://github.com/hyperledger/fabric/tree/release-1.1/images



Fabric Network / Provisioning / Prepare Artifacts

□ cryptogen

Usage	Generate keys and certificates(crypto artifacts) for Fabric network.
Input	`crypto-config.yaml`
Input Samples	▪ hyperledger/fabric-samples/first-network/crypto-config.yaml ▪ hyperledger/fabric/examples/e2e_cli/crypto-config.yaml
Output	▪ keys and certificates for Fabric CA server ▪ signing keys and certificates, TLS keys and certificates for Orderers and Peers ▪ signing key and certificate, TLS key and certificate of admin for each Org.
Output Samples	▪ hyperledger/fabric/sampleconfig/msp/

□ configtxgen

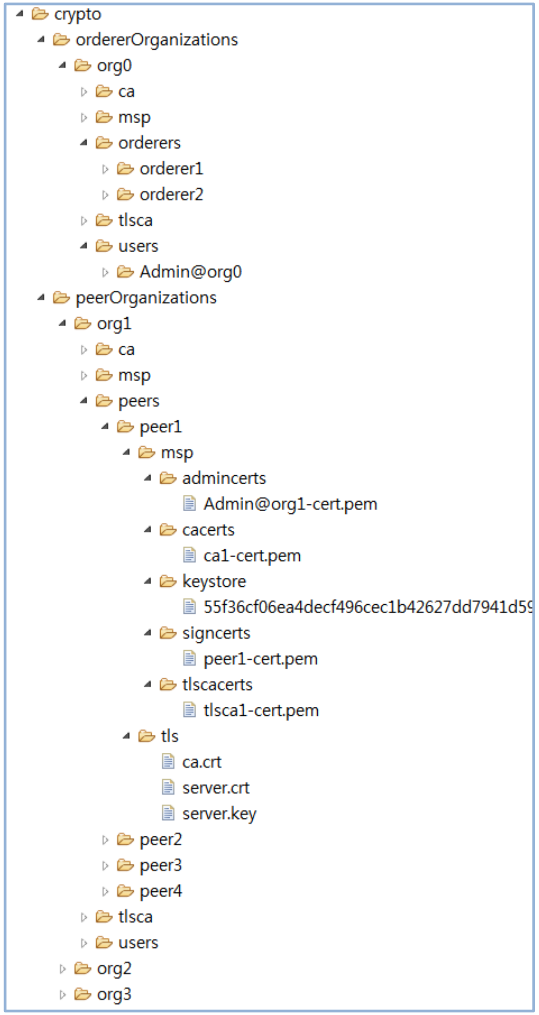
Usage	Create and inspect configuration related artifacts
Input	`configtx.yaml`
Input Samples	▪ hyperledger/fabric/sampleconfig/configtx.yaml ▪ hyperledger/fabric-samples/first-network/configtx.yaml ▪ hyperledger/fabric/e2e_cli/configtx.yaml
Output	▪ genesis block ▪ channel creation transaction for each channel ▪ anchor peer update transactions for each channel
Output Samples	▪ hyperledger/fabric-samples/basic-network/config/

Fabric Network / Provisioning / Crypto Artifacts

```
OrdererOrgs:
  - Name: org0
    Domain: org0
    CA:
      HostName : ca0
      CommonName: ca0
    Specs:
      - Hostname: orderer1
        CommonName: orderer1
      - Hostname: orderer2
        CommonName: orderer2

PeerOrgs:
  - Name: org1
    Domain: org1
    CA:
      HostName : ca1
      CommonName: ca1
    Specs:
      - Hostname: peer1
        CommonName: peer1
      - Hostname: peer2
        CommonName: peer2
      - Hostname: peer3
        CommonName: peer3
      - Hostname: peer4
        CommonName: peer4
    Users:
      Count: 1
```

```
  - Name: org2
    Domain: org2
    CA:
      HostName : ca2
      CommonName: ca2
    Specs:
      - Hostname: peer5
        CommonName: peer5
      - Hostname: peer6
        CommonName: peer6
      - Hostname: peer7
        CommonName: peer7
      - Hostname: peer8
        CommonName: peer8
    Users:
      Count: 1
  - Name: org3
    Domain: org3
    CA:
      HostName : ca3
      CommonName: ca3
    Specs:
      - Hostname: peer9
        CommonName: peer9
      - Hostname: peer10
        CommonName: peer10
      - Hostname: peer11
        CommonName: peer11
      - Hostname: peer12
        CommonName: peer12
    Users:
      Count: 1
```

cryptogen



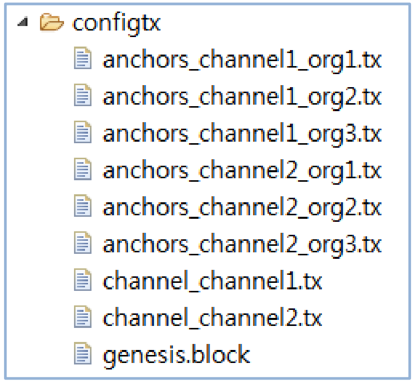
```
crypto
  ordererOrganizations
    org0
      ca
      msp
      orderers
        orderer1
        orderer2
      tlsca
      users
        Admin@org0
  peerOrganizations
    org1
      ca
      msp
      peers
        peer1
          msp
            admincerts
              Admin@org1-cert.pem
            cacerts
              ca1-cert.pem
            keystore
              55f36cf06ea4decf496cec1b42627dd7941d59
            signcerts
              peer1-cert.pem
            tlscacerts
              tlsca1-cert.pem
          tls
            ca.crt
            server.crt
            server.key
        peer2
        peer3
        peer4
      tlsca
      users
    org2
    org3
```

Fabric Network / Provisioning / Config Transactions

```
Profiles:
  GenesisProfile:
    Orderer:
      <<: *ordererDefaults
      Organizations:
        - *org0

    Consortiums:
      Channel1Consortium:
        Organizations:
          - *org1
          - *org2
          - *org3
      Channel2Consortium:
        Organizations:
          - *org1
          - *org2
          - *org3
  Channel1Profile:
    Consortium: Channel1Consortium
    Application:
      <<: *applicationDefaults
      Organizations:
        - *org1
        - *org2
        - *org3
  Channel2Profile:
    Consortium: Channel2Consortium
    Application:
      <<: *applicationDefaults
      Organizations:
        - *org1
        - *org2
        - *org3
```

```
Organizations:
  - &org0
    Name: org0
    ID: org0
    MSPDir: crypto/ordererOrganizations/org0/msp
    AdminPrincipal: Role.ADMIN
  - &org1
    Name: org1
    ID: org1
    MSPDir: crypto/peerOrganizations/org1/msp
    AdminPrincipal: Role.ADMIN
    AnchorPeers:
      - Host: *.*.*.173
        Port: 7051

...

Orderer: &ordererDefaults
  OrdererType: kafka
  Addresses:
    - *.*.*.188:7050
    - *.*.*.166:7050

  BatchTimeout: 4s
  BatchSize:
    MaxMessageCount: 400
    AbsoluteMaxBytes: 5 MB
    PreferredMaxBytes: 1024 KB

  Kafka:
    Brokers:
      - *.*.*.*:9092
      - *.*.*.*:9092
...
```

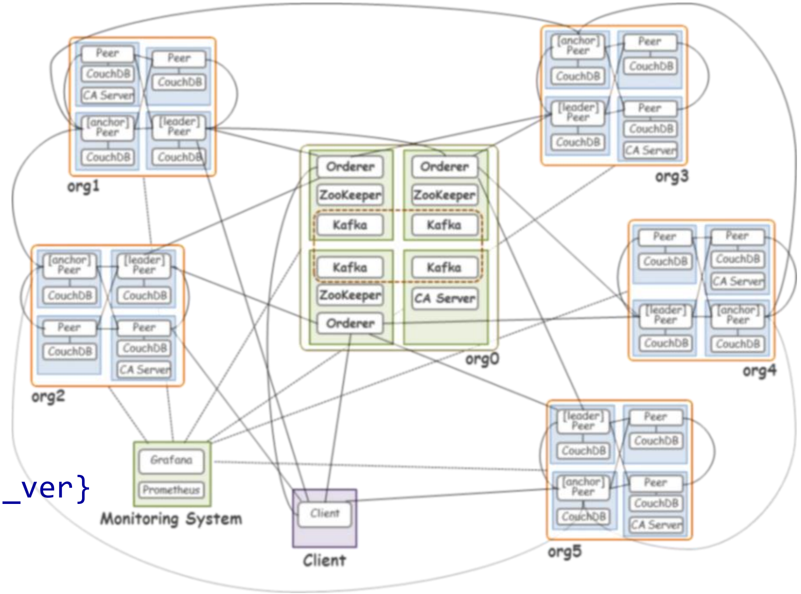
configtxgen



```
configtx
    anchors_channel1_org1.tx
    anchors_channel1_org2.tx
    anchors_channel1_org3.tx
    anchors_channel2_org1.tx
    anchors_channel2_org2.tx
    anchors_channel2_org3.tx
    channel_channel1.tx
    channel_channel2.tx
    genesis.block
```

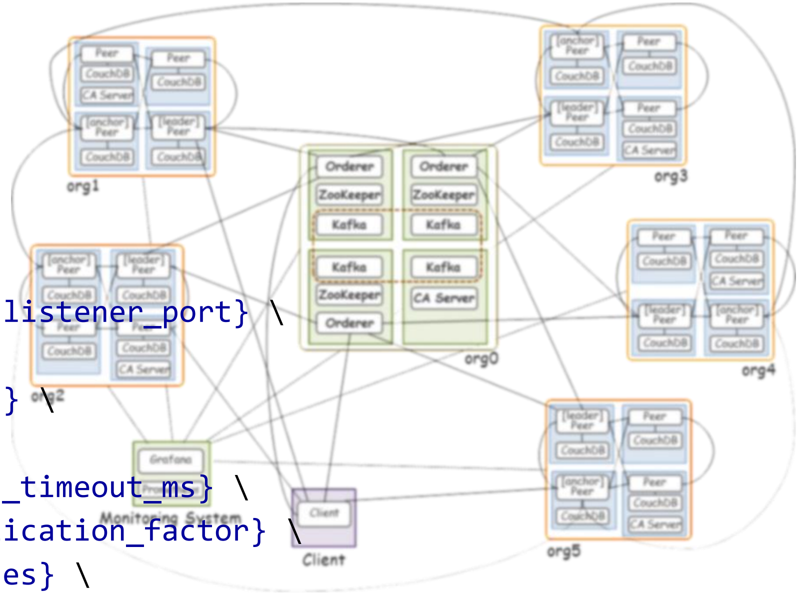
Fabric Network / Provisioning / Run ZooKeeper

```
#! /bin/bash

docker run -d \
--name ${zk_name} \
--network host \
-e ZOO_MY_ID=${zk_id} \
-e ZOO_PORT=${zk_port} \
-e ZOO_SERVERS=${zk_servers} \
-e ZOO_TICK_TIME ${zk_tick_time} \
-e ZOO_INIT_LIMIT ${zk_init_limit} \
-e ZOO_SYNC_LIMIT ${zk_sync_limit} \
hyperledger/fabric-zookeeper:${host_arch}-${fabric_ver}
```



Fabric Network / Provisioning / Run Kafka

```
#! /bin/bash
docker run -d \
--name ${name} \
--network host \
-e KAFKA_ZOOKEEPER_CONNECT=${zk_connect_str} \
-e KAFKA_ADVERTISED_HOST_NAME=${name} \
-e KAFKA_BROKER_ID=${broker_id} \
-e KAFKA_LISTENERS=PLAINTEXT://${listener_addr}:${listener_port} \
-e KAFKA_MESSAGE_MAX_BYTES=${message_max_bytes} \
-e KAFKA_MIN_INSYNC_REPLICAS=${min_insync_replicas} \
-e KAFKA_UNCLEAN_LEADER_ELECTION_ENABLE=false
-e KAFKA_ZOOKEEPER_CONNECTION_TIMEOUT.MS=${zk_conn_timeout_ms} \
-e KAFKA_DEFAULT_REPLICATION_FACTOR=${default_replication_factor} \
-e KAFKA_REPLICA_FETCH_MAX_BYTES=${message_max_bytes} \
-e KAFKA_METRICS_RECORDING_LEVEL=${metrics_recording_level} \
-e KAFKA_HEAP_OPTS=${jvm_heap_opts} \
-e KAFKA_JVM_PERFORMANCE_OPTS=${jvm_perf_opts} \
-e KAFKA_GC_LOG_OPTS=${jvm_gc_log_opts} \
-e KAFKA_JMX_OPTS=${jvm_jmx_opts} \
hyperledger/fabric-kafka:${host_arch}-${fabric_ver}
```



Fabric Network / Provisioning / Run Orderers

```
- name: Run Fabric Orderer containers
  docker_container:
    image: "{{ docker.images.orderer.repository }}:{{ docker.images.orderer.tag }}"
    name: "{{ item.name }}"
    network_mode: host
    env:
      ORDERER_GENERAL_LISTENADDRESS: "{{ hostvars[item.host].ansible_host }}"
      ORDERER_GENERAL_LISTENPORT: "{{ item.port }}"
      ORDERER_GENERAL_TLS_PRIVATEKEY: /var/hyperledger/orderer/tls/server.key
      ORDERER_GENERAL_TLS_CERTIFICATE: /var/hyperledger/orderer/tls/server.crt
      ORDERER_GENERAL_TLS_ROOTCAS: [/var/hyperledger/orderer/tls/ca.crt]
      ORDERER_GENERAL_LOGLEVEL: "{{ item.config.General.LogLevel }}"
      ORDERER_GENERAL_GENESISMETHOD: file  # provisional | file
      ORDERER_GENERAL_GENESISFILE: /var/hyperledger/orderer/orderer.genesis.block
      ORDERER_GENERAL_LOCALMSPDIR: /var/hyperledger/orderer/msp
      ...
    volumes:
        - "~/fabric/configtx/genesis.block:/var/hyperledger/orderer/orderer.genesis.block"
        - "~/fabric/crypto/ordererOrganizations/{{ item.org }}/orderers/{{ item.name }}/msp
              :/var/hyperledger/orderer/msp"
        - "~/fabric/crypto/ordererOrganizations/{{ item.org }}/orderers/{{ item.name }}/tls/
              :/var/hyperledger/orderer/tls"
        - "~/fabric/volumes/{{ item.name }}/var/hyperledger/production/orderer
              :/var/hyperledger/production/orderer"
```

※ hyperledger/fabric/fabric/sampleconfig/orderer.yaml

Fabric Network / Provisioning / Run Peers

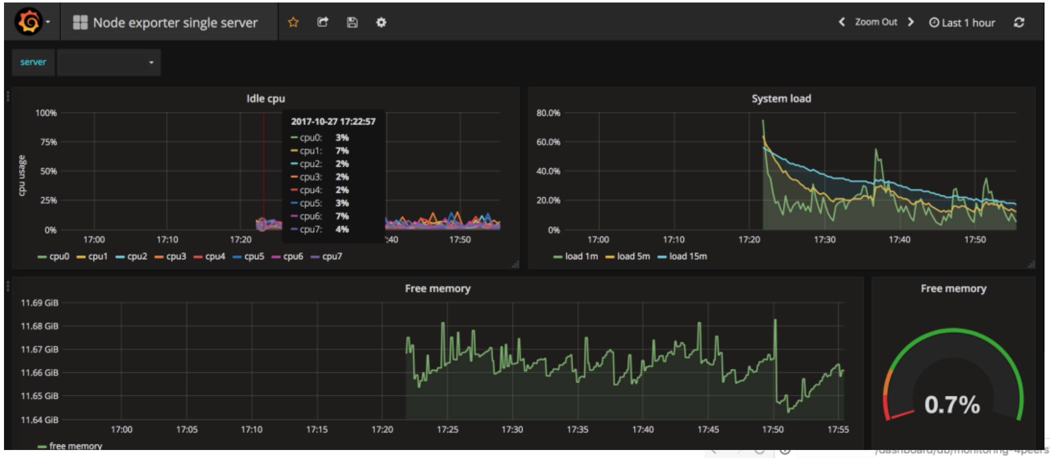
```
- name: Run Fabric Peer Containers
  docker_container:
    image: "{{ docker.images.peer.repository }}:{{ docker.images.peer.tag }}"
    name: "{{ item.name }}"
    network_mode: host
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
    env:
      CORE_LOGGING_LEVEL: "{{ item.config.logging.level }}"
      CORE_PEER_ID: "{{ item.name }}"
      CORE_PEER_LISTENADDRESS: "{{ address }}:{{ item.config.peer.listenPort }}"
      CORE_PEER_CHAINCODELISTENADDRESS: "{{ address }}:{{ item.config.peer.chaincodeListenPort }}"
      CORE_PEER_ADDRESS: "{{ address }}:{{ item.config.peer.listenPort }}"
      CORE_PEER_GOSSIP_USELEADERELECTION: false
      CORE_PEER_GOSSIP_ORGLEADER: "{{ item.config.peer.gossip.orgLeader }}"
      CORE_PEER_GOSSIP_ENDPOINT: "{{ address }}:{{ item.config.peer.listenPort }}"
      CORE_PEER_TLS_CERT_FILE: /etc/hyperledger/fabric/tls/server.crt
      CORE_PEER_TLS_KEY_FILE: /etc/hyperledger/fabric/tls/server.key
      CORE_PEER_TLS_ROOTCERT_FILE: /etc/hyperledger/fabric/tls/ca.crt
      CORE_PEER_FILESYSTEMPATH: /var/hyperledger/production
      CORE_PEER_MSPCONFIGPATH: msp
      volumes:
        - /var/run/:/host/var/run/
        - "~/fabric/volumes/{{ item.name }}/var/hyperledger/production:/var/hyperledger/production"
        - "~/fabric/crypto/peerOrganizations/{{ item.org }}/peers/{{ item.name }}/msp
            :/etc/hyperledger/fabric/msp"
```

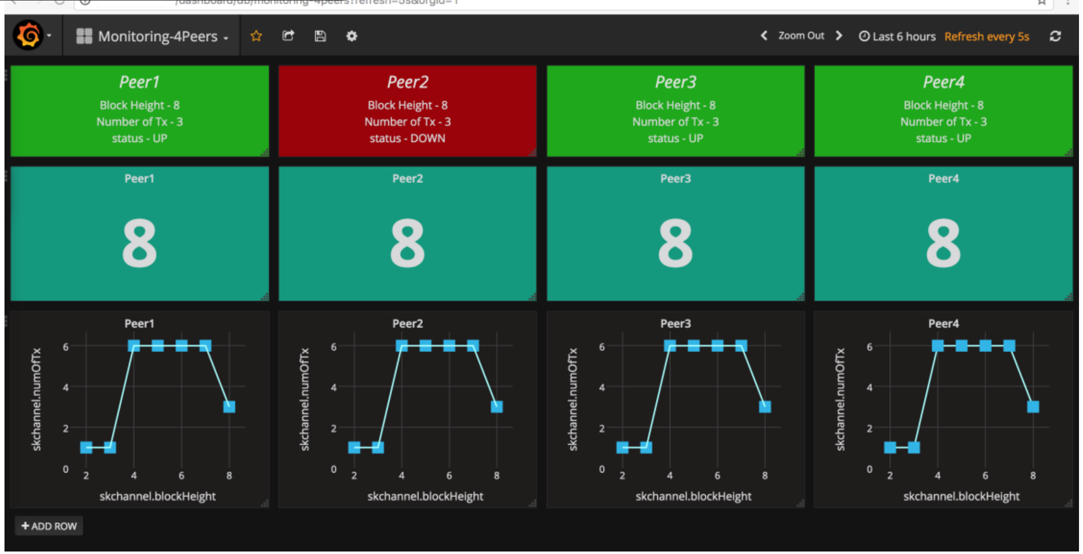
※ hyperledger/fabric/fabric/sampleconfig/core.yaml

Fabric Network / Power Tools

Tool	Software	Remarks
Software Provisioning Automation	Ansible	• Using SSH connection: No agent • Systematic but flexible inventory management • Provides YAML based DSL • Provides templating • Parallel task processing
System Monitoring Software	Prometheus	• Time-series database • Able to define complex (multi-dimensional) data • Provides query language
Dashboard Software	Grafana	• Built-in supports various data source - Graphite, Prometheus, Elasticsearch, InfluxDB, MySQL, PostgreSQL • Provides highly customizable graph, chart, and dashboard • Provides alerting
Log Integration	ELK Stack	

Fabric Network / System Monitoring





Peer1
Block Height - 8
Number of Tx - 3
status - UP

Peer2
Block Height - 8
Number of Tx - 3
status - DOWN

Peer3
Block Height - 8
Number of Tx - 3
status - UP

Peer4
Block Height - 8
Number of Tx - 3
status - UP

IV. Architectural Issues of Hyperledger Fabric

Fabric / Performance

- ***Fabric Network Scale : # of Peers***

- ***Optimize Disk and Network over CPU and Memory***

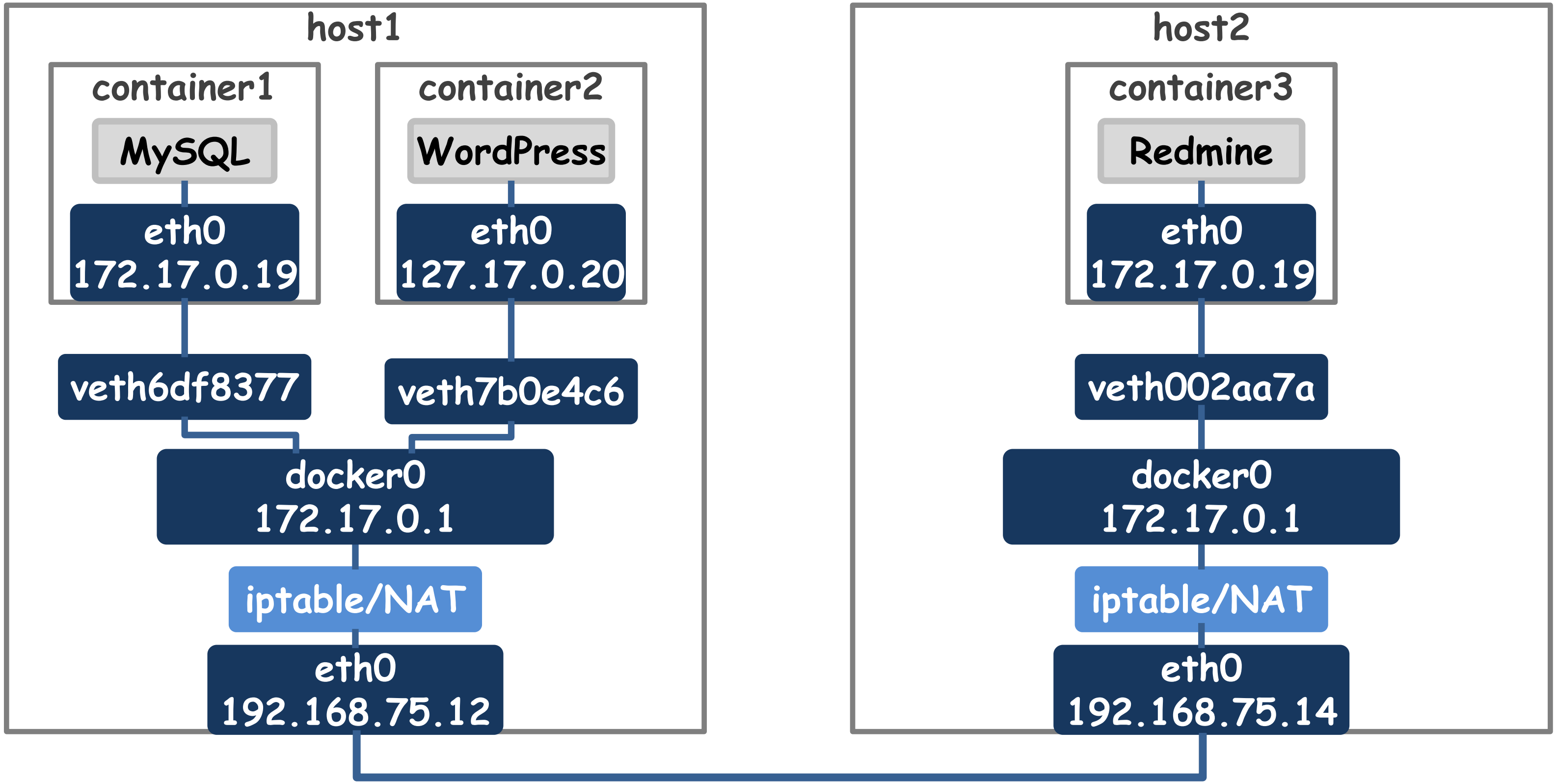
- ***Minimize Virtualization***

- ***Bare-metal based Cloud over Virtual Machine based Cloud***

- ***Docker free***

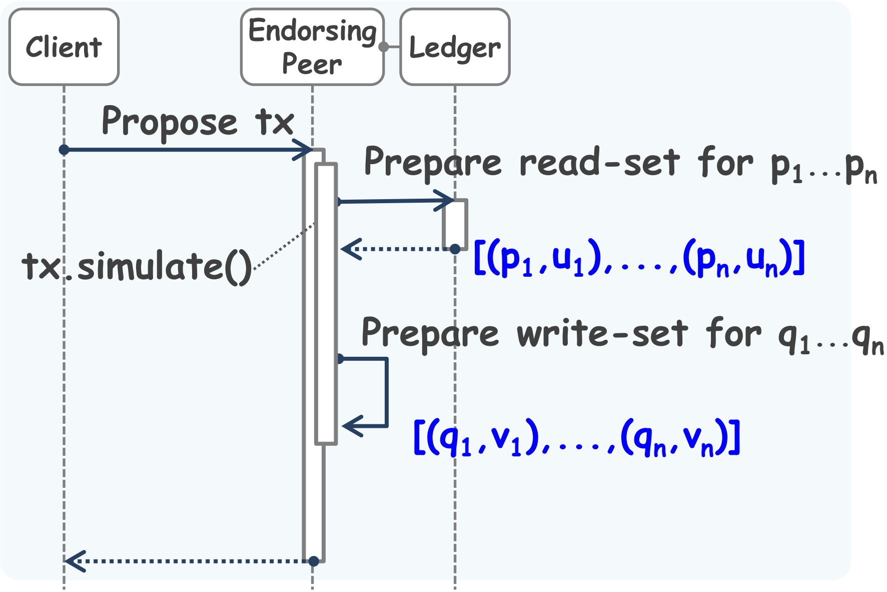
Fabric / Performance

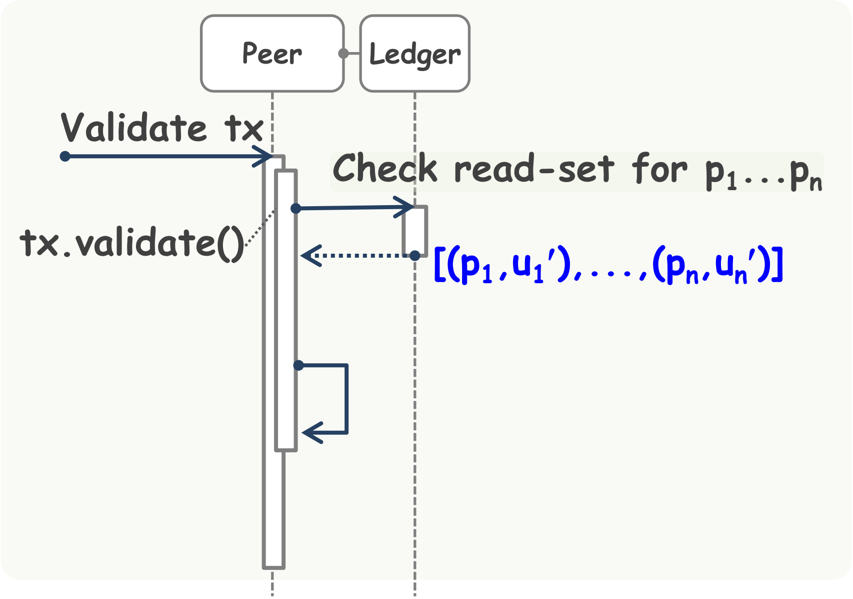
How to use "loopback interface" with Docker containers in a same host ?



※ https://docs.docker.com/engine/tutorials/networkingcontainers/
https://developer.ibm.com/recipes/tutorials/networking-your-docker-containers-using-docker0-bridge/

Fabric / Transaction Concurrency /Simulation and Validation

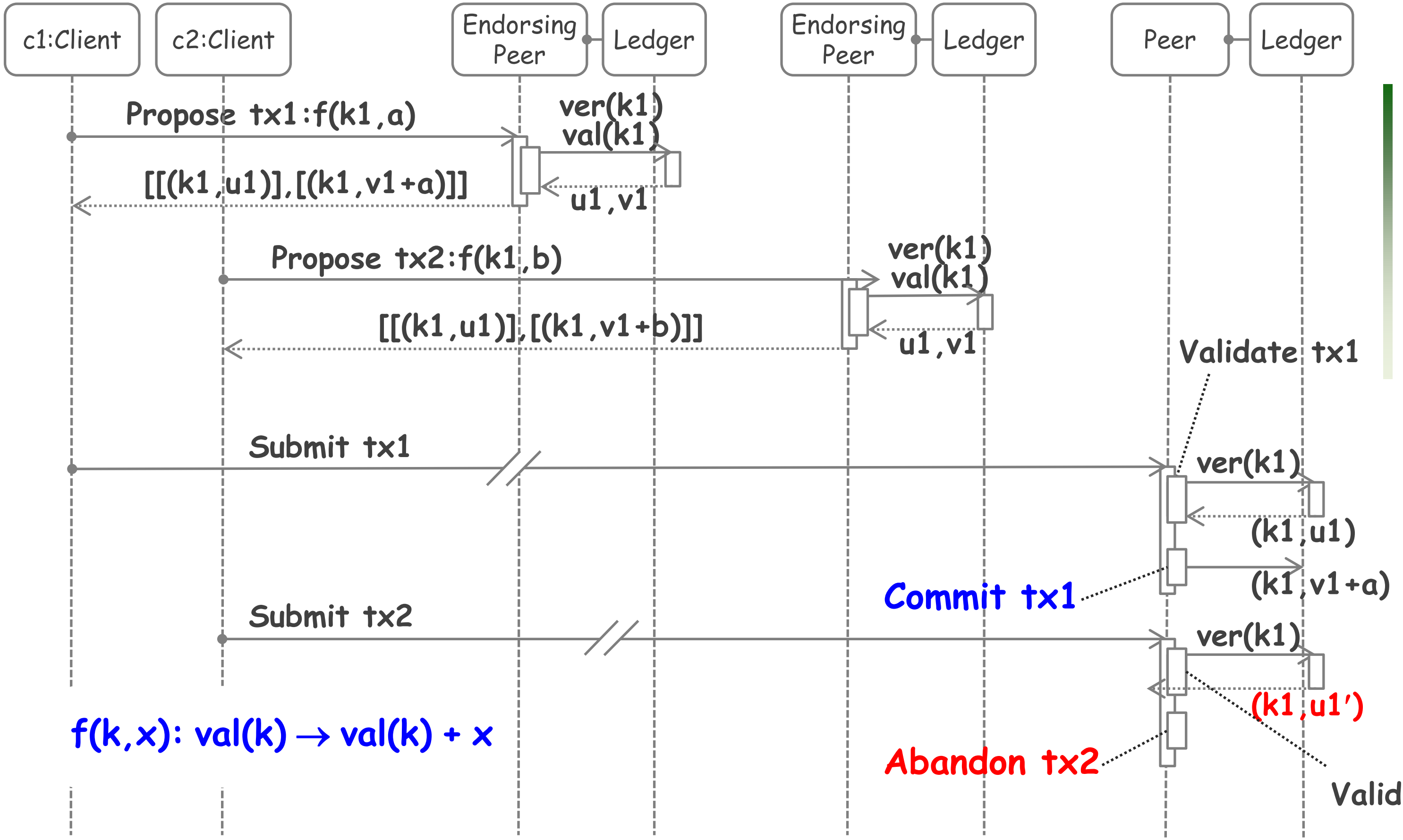

Client
Endorsing Peer
Ledger
Propose tx
Prepare read-set for $p_1 \ldots p_n$
tx.simulate()
$[(p_1,u_1),\ldots,(p_n,u_n)]$
Prepare write-set for $q_1 \ldots q_n$
$[(q_1,v_1),\ldots,(q_n,v_n)]$


Peer
Ledger
Validate tx
Check read-set for $p_1 \ldots p_n$
tx.validate()
$[(p_1,u_1'),\ldots,(p_n,u_n')]$

If $u_i = u_i'$ for all, tx is Valid

Else tx is Invalid

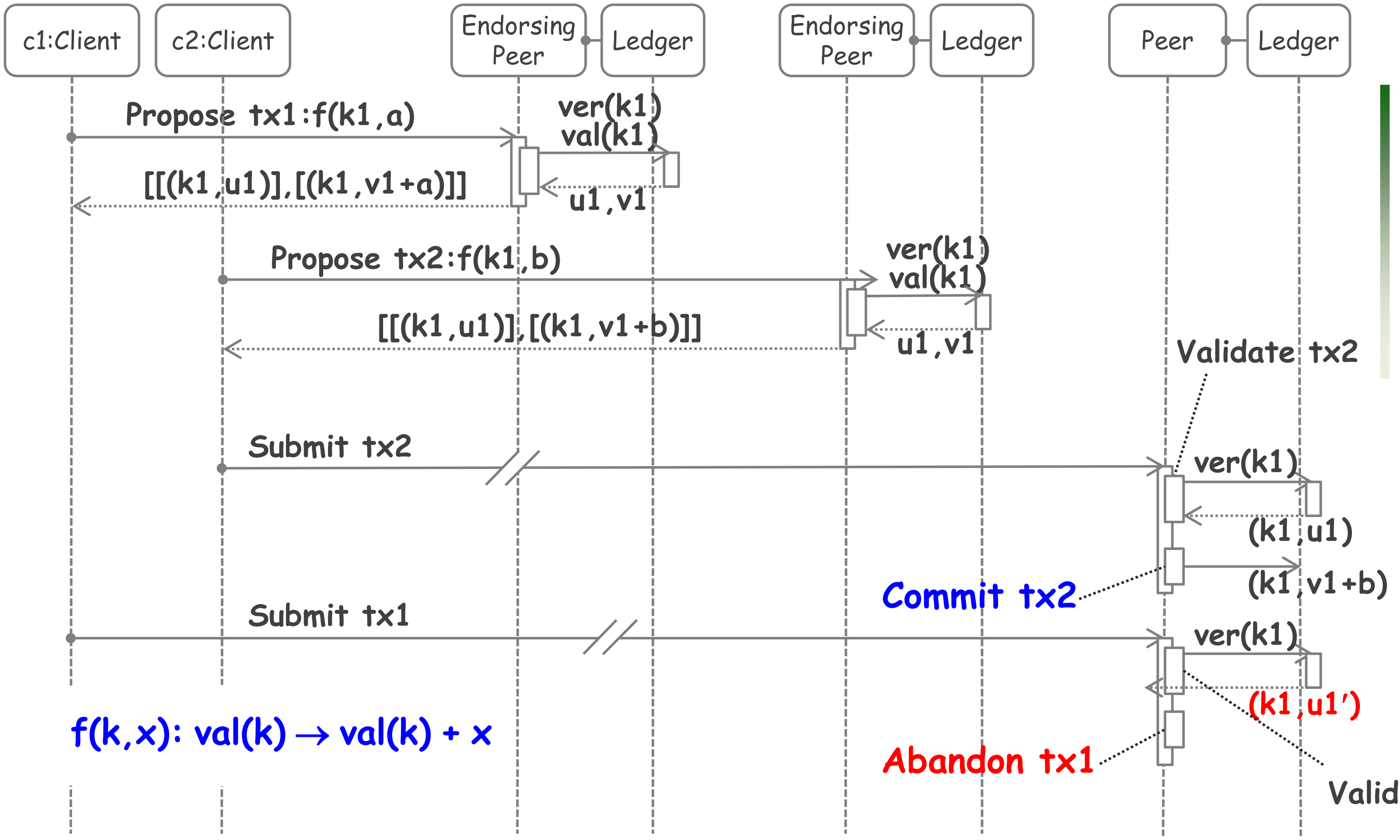
Fabric / Transaction Concurrency / Case 1



f(k,x): val(k) → val(k) + x

① tx1.simulate()
② tx2.simulate()
③ tx1.validate()
④ tx1.commit()
⑤ tx2.validate()
⑥ tx2.failed()

Fabric / Transaction Concurrency / Case 2



① tx1.simulate()

② tx2.simulate()

③ tx2.validate()

④ tx2.commit()

⑤ tx1.validate()

⑥ tx1.failed()

f(k,x): val(k) → val(k) + x

Fabric / Transaction Concurrency / Review 1

```
tx1.simulate()
tx2.simulate()
tx1.validate()
tx1.commit()
tx2.validate()
tx2.failed()
```

Case 1

```
tx1.simulate()
tx2.simulate()
tx2.validate()
tx2.commit()
tx1.validate()
tx1.failed()
```

Case 2

```
tx1.simulate()
tx2.simulate()
tx1.validate()
tx2.validate()
tx1.commit()
tx2.commit()
```

Case 3

☑ ***Case 1 for All Peers or Case 2 for All Peers***

☑ ***Never Case 1 for Some and Case 2 for the Others***

☑ ***Never Case 3***

✓ ***Only 1 transaction among concurrent can succeed. (with some conditions)***

✓ ***`tx.validate()` and `tx.commit()` are atomic.***

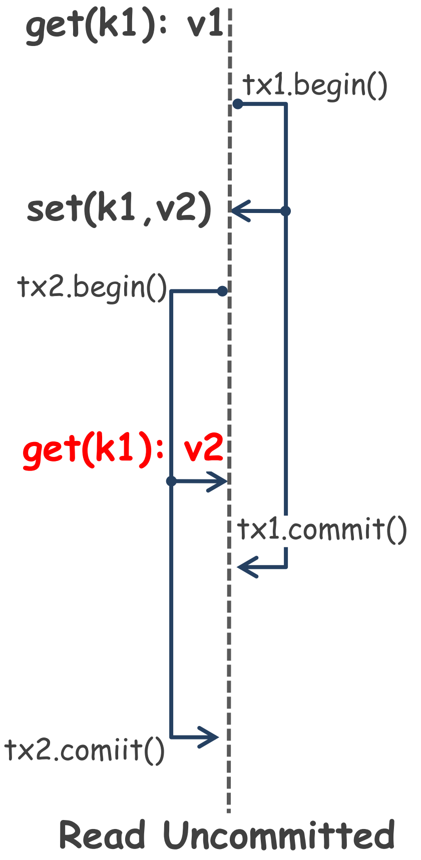
Why ?

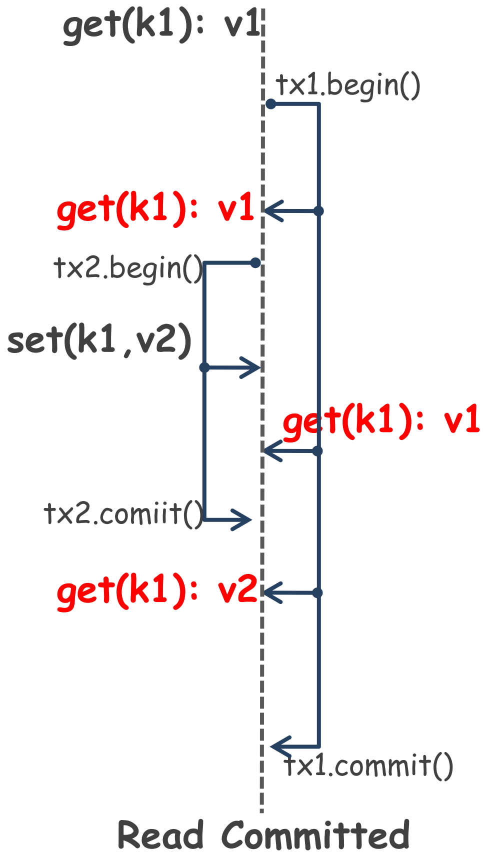
Fabric / Transaction Concurrency / Summary

☑ ***No Dirty Read***
No Non-repeatable Read

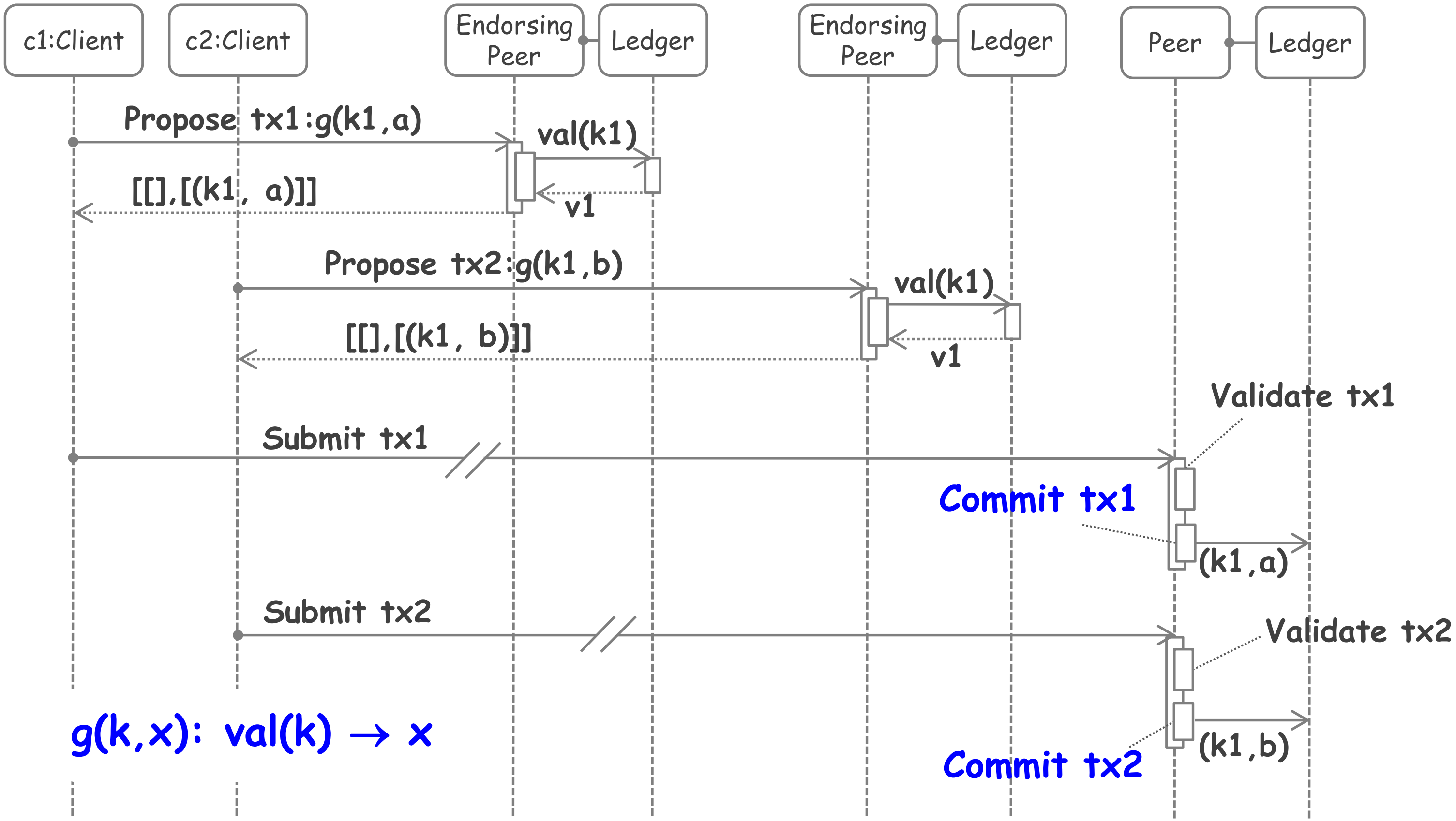
☑ ***Transaction Isolation Level : ?***

☑ ***Multi-version Concurrency Control***
Optimistic Locking





Fabric / Transaction Concurrency / Write-set only case



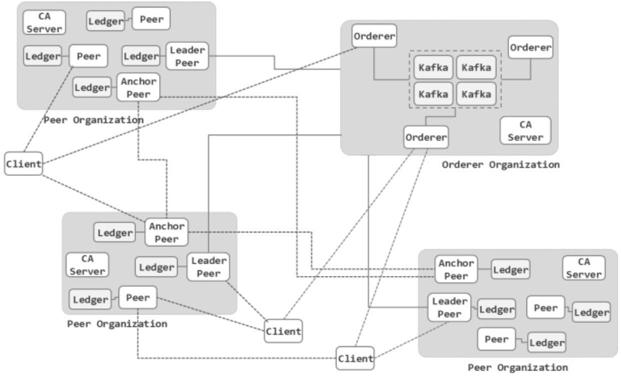
$g(k,x)$: $val(k) \rightarrow x$

FAQ : How many peer organizations?

Distributed Ledger :

replicated, shared,
and synchronized digital data
<u>geographically spread</u> across multiple sites …

from Wikipedia

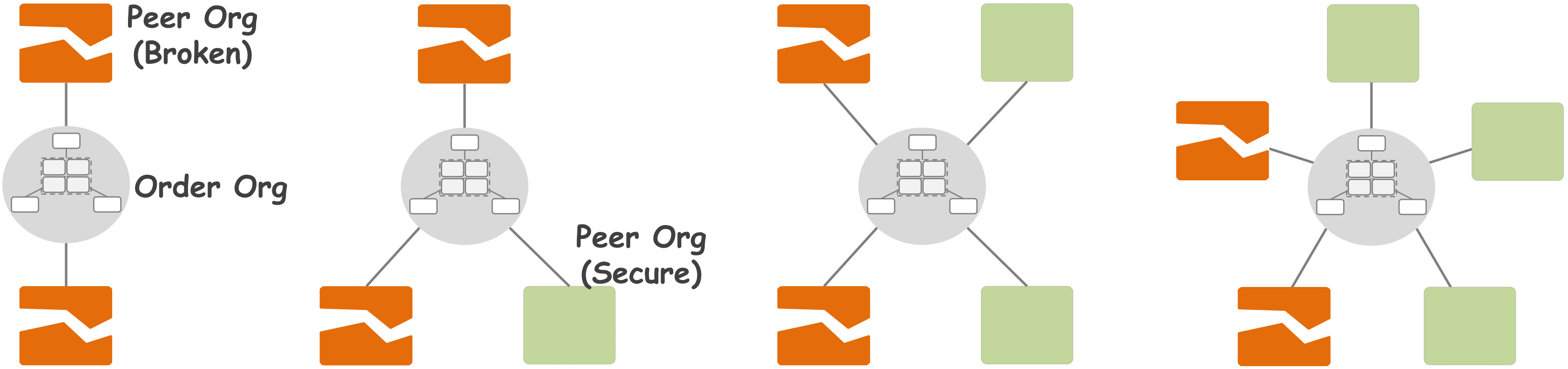


What is node ?

Independent Location/Access Control/Ownership

Public blockchain	**Peer**
Private blockchain	**Peer ?** **Peer Organization ?**

FAQ : How many peer organizations?



FAQ : How many peers?

▪ Application Server Clustering	**Distributes Requests**
▪ Hadoop ▪ No SQL Sharding	**Splits Huge Data**

Scale Out

Availability ↑

Performance ↑

▪ Private Blockchain	**Replicated and Synchronized Data**

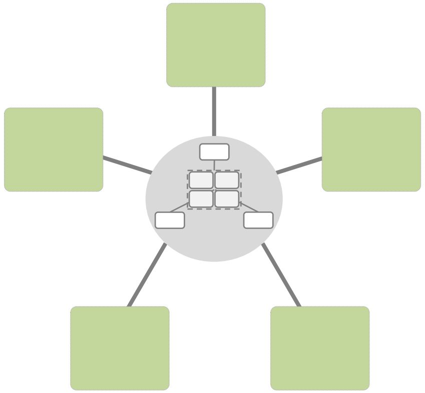
Scale Out

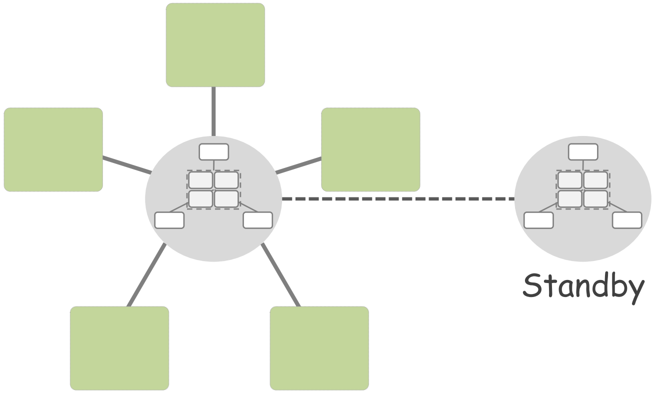
Availability ↑

Performance ↓

For Availability	***Scale Out***
For Performance	***Scale Up***

FAQ : Where is SPoF (Single Point Of Failure)?





☑ ***Peer***

- ***multiple cluster***
- ***multiple organization***

☑ ***Orderer***

- ***single cluster***
- ***single organization***

☑ ***Peer***

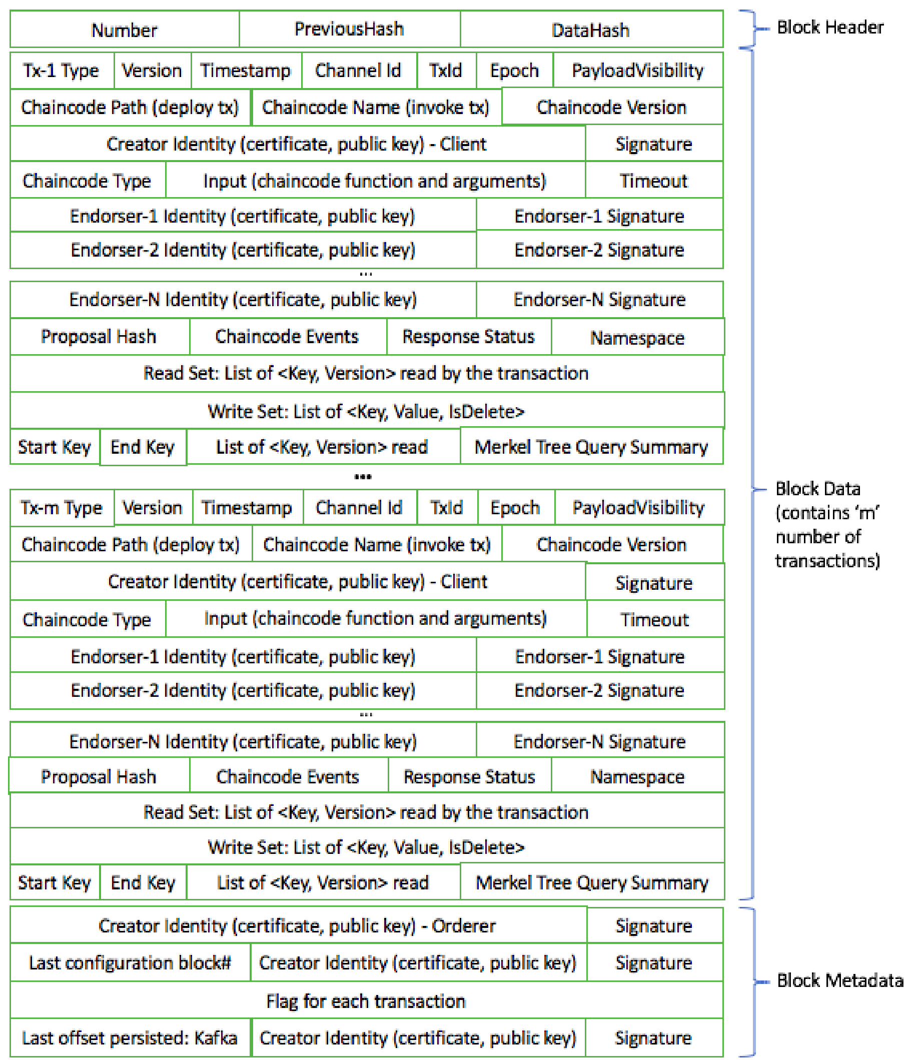
- ***Disaster recoverable by nature***

☑ ***Orderer Cluster***

- ***Single Point of Failure***
- ***More robust access control and security***
- ***Standby cluster***

A. Fabric Block Structure

Appendix



Number	PreviousHash	DataHash				
Tx-1 Type	Version	Timestamp	Channel Id	TxId	Epoch	PayloadVisibility
Chaincode Path (deploy tx)	Chaincode Name (invoke tx)	Chaincode Version				
Creator Identity (certificate, public key) - Client	Signature					
Chaincode Type	Input (chaincode function and arguments)	Timeout				
Endorser-1 Identity (certificate, public key)	Endorser-1 Signature					
Endorser-2 Identity (certificate, public key)	Endorser-2 Signature					
...						
Endorser-N Identity (certificate, public key)	Endorser-N Signature					
Proposal Hash	Chaincode Events	Response Status	Namespace			
Read Set: List of <Key, Version> read by the transaction						
Write Set: List of <Key, Value, IsDelete>						
Start Key	End Key	List of <Key, Version> read	Merkel Tree Query Summary			
•••						
Tx-m Type	Version	Timestamp	Channel Id	TxId	Epoch	PayloadVisibility
Chaincode Path (deploy tx)	Chaincode Name (invoke tx)	Chaincode Version				
Creator Identity (certificate, public key) - Client	Signature					
Chaincode Type	Input (chaincode function and arguments)	Timeout				
Endorser-1 Identity (certificate, public key)	Endorser-1 Signature					
Endorser-2 Identity (certificate, public key)	Endorser-2 Signature					
...						
Endorser-N Identity (certificate, public key)	Endorser-N Signature					
Proposal Hash	Chaincode Events	Response Status	Namespace			
Read Set: List of <Key, Version> read by the transaction						
Write Set: List of <Key, Value, IsDelete>						
Start Key	End Key	List of <Key, Version> read	Merkel Tree Query Summary			
Creator Identity (certificate, public key) - Orderer	Signature					
Last configuration block#	Creator Identity (certificate, public key)	Signature				
Flag for each transaction						
Last offset persisted: Kafka	Creator Identity (certificate, public key)	Signature				

https://blockchain-fabric.blogspot.kr/2017/04/hyperledger-fabric-v10-block-structure.html

B. Resources

Title	URL	Remarks
Hyperledger Homepage	https://www.hyperledger.org/	
Fabric Sources	https://github.com/hyperledger/fabric	
Fabric Documentation	http://hyperledger-fabric.readthedocs.io/	• Getting started • Tutorials • References
Fabric CA Documentation	http://hyperledger-fabric-ca.readthedocs.io/	
Fabric Wiki	https://wiki.hyperledger.org/projects/fabric	
Fabric JIRA	https://jira.hyperledger.org/projects/FAB/	• Issue management
Hyperledger Docker Repository	https://hub.docker.com/u/hyperledger/	
Fabric 1.1 Commands Reference	http://hyperledger-fabric.readthedocs.io/en/release-1.1/command_ref.html	
Fabric Chaincode API (Go)	https://godoc.org/github.com/hyperledger/fabric/core/chaincode/shim	
Fabric SDK for Node.js Documentation	https://fabric-sdk-node.github.io/	• API documentation • Tutorials