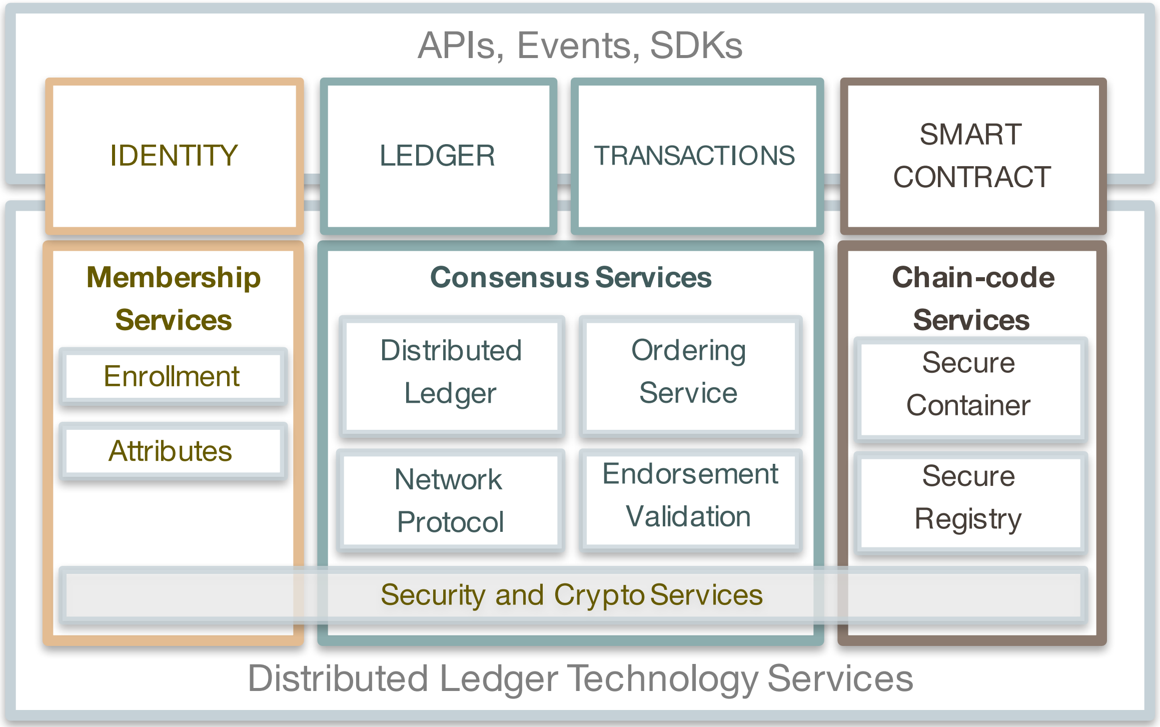
Hyperledger Fabric v1.0 Deep Dive

Binh Nguyen, IBM

Contents

Reference Architecture



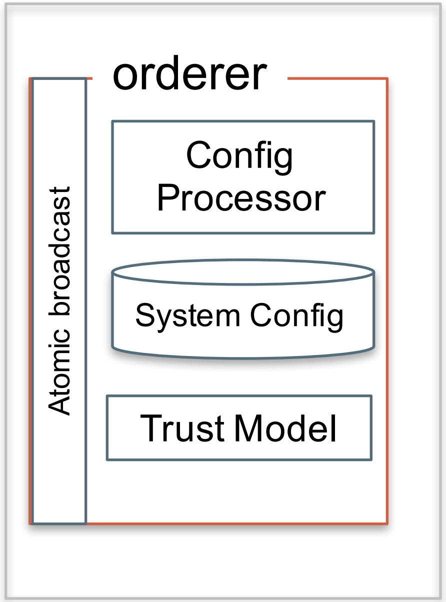
IDENTITY
Pluggable, Membership, Privacy and Auditability of transactions.

LEDGER | TRANSACTIONS
Distributed transactional ledger whose state is updated by consensus of stakeholders

SMART-CONTRACT
“Programmable Ledger”, provide ability to run business logic against the blockchain (aka smart contract)

APIs, Events, SDKs
Multi-language native SDKs allow developers to write DLT apps

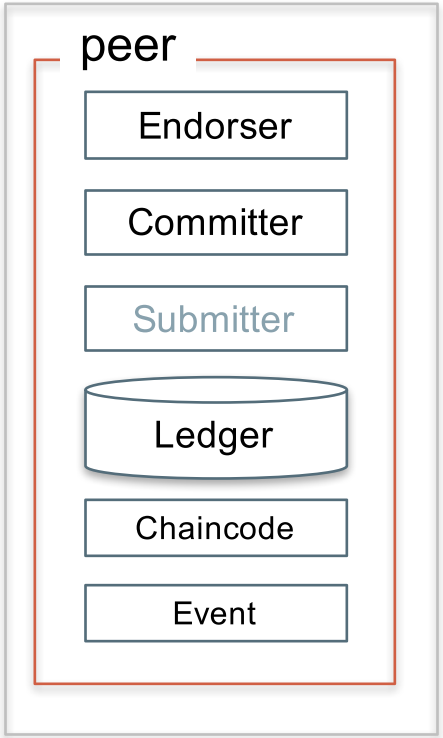
Orderer



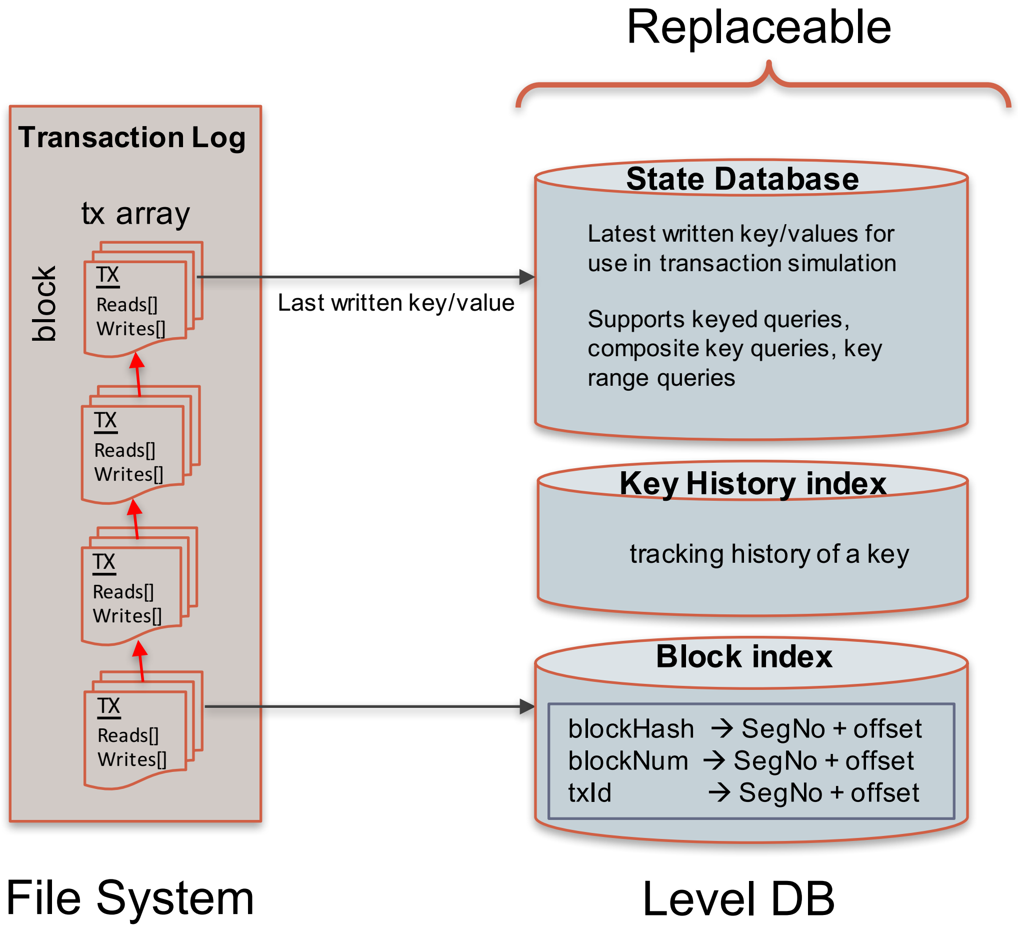
- A group of Orderers runs a communication service, called ordering service, to provide atomic broadcast
- Ordering service is the genesis of a network. Clients of ordering service are peers and applications
  - Accept transactions and deliver blocks
  - Process all configuration transactions to set up network policies (readers, writers, admins)
- Orderer manages a pluggable trust engine (eg CFT or BFT) that performs the ordering of the transactions

Peer

- A Peer is a node on the network maintaining state of the ledger and managing chaincodes
- Any number of Peers may participate in a network
- A Peer can be an endorser, committer and/or submitter (submitter has not been implemented). An endorser is always a committer
  - An endorser executes and endorses transactions
  - A committer verifies endorsements and validates transaction results
- A Peer manages event hub and deliver events to the subscribers
- Peers form a peer-to-peer gossip network



Ledger



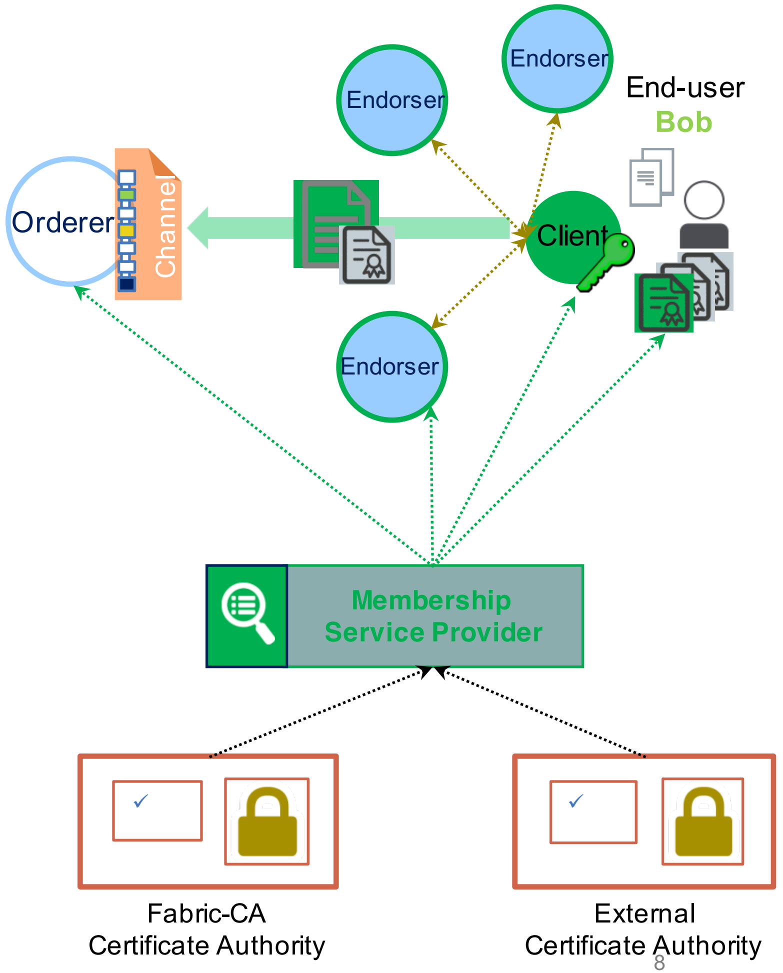
CouchDB (external option) supports keyed queries, composite key queries, key range queries, plus full data rich queries (beta in 1.0)

Channel

- A data partitioning mechanism to control transaction visibility only to stakeholders
- Consensus takes place within a channel by members of the channel
  - Other members on the network are not allowed to access the channel and will not see transactions on the channel
- A chaincode may be deployed on multiple channels, each instance is isolated within its channel
  - A chaincode may query another chaincode in other channel (ACL applied)

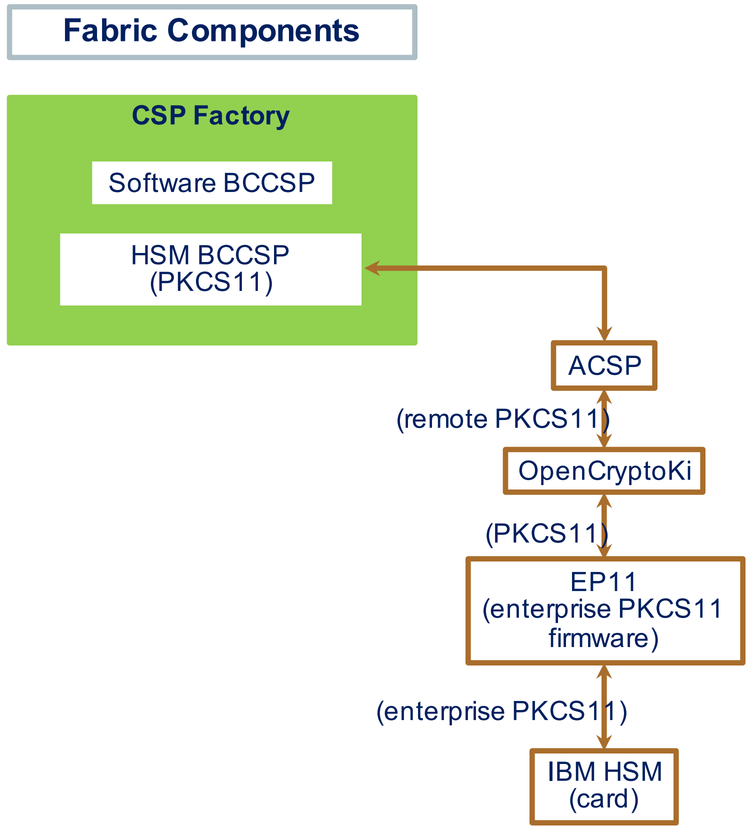
Membership Service Provider

- An abstraction of identity provider
  - <MSP.id, MSP.sign, MSP.verify, MSP.validateid, MSP.admin>
  - govern application, endorser and orderer identities
- Used as building blocks for access control frameworks
  - at the system level (read/write access on system controls, and channel creation)
  - at the channel level (read/write access),
  - at the chaincode level (invocation access)
- Represent a consortium or a member



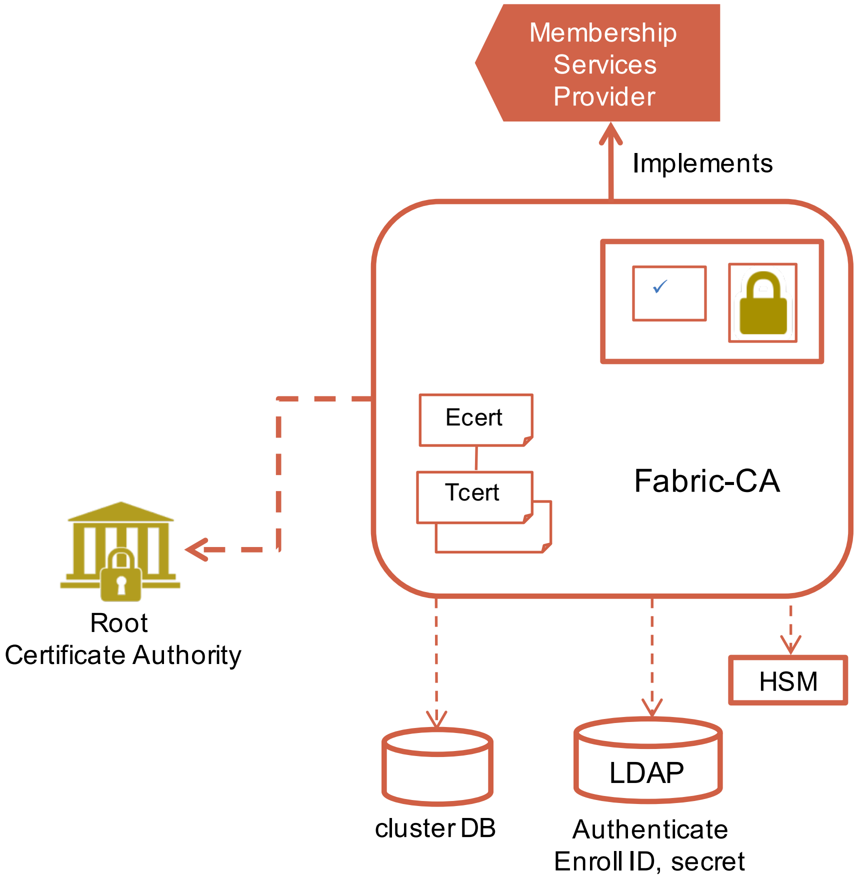
Endorser
Endorser
End-user
Bob
Orderer
Channel
Client
Endorser
Membership Service Provider
Fabric-CA Certificate Authority
External Certificate Authority

Crypto Service Provider



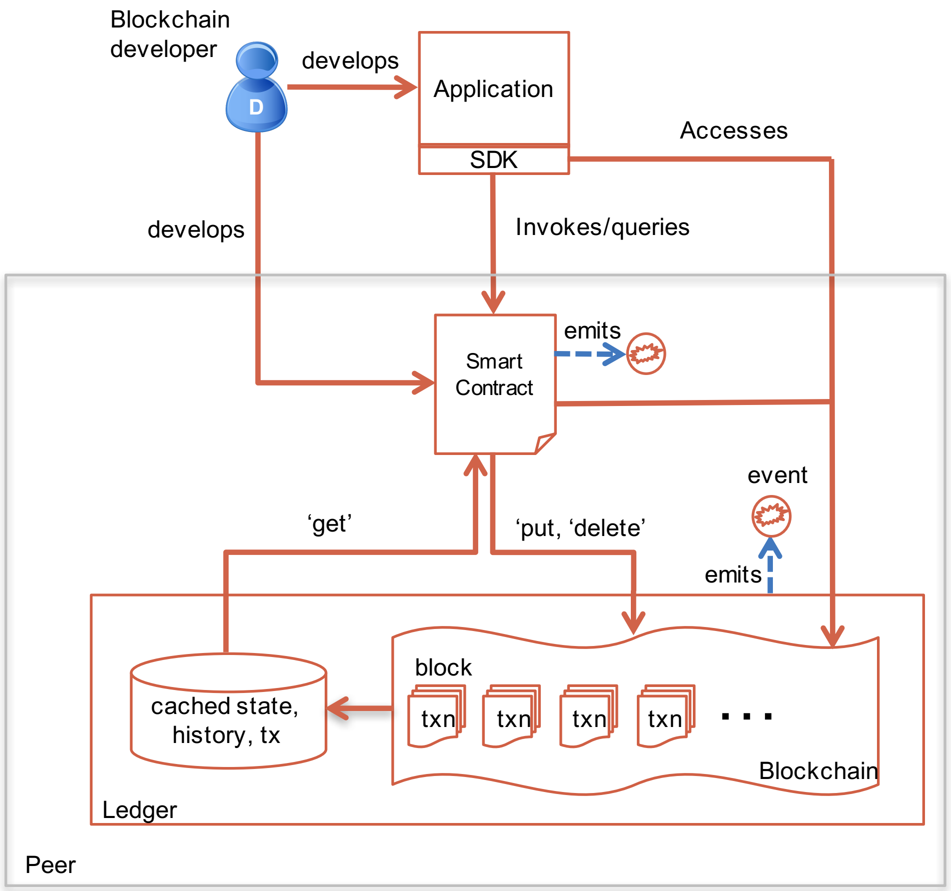
- CSP abstracts crypto standards (software and hardware) to enable plugging in different implementation
  - Alternate implementations of crypto interface can be used within the Fabric code, without modifying the core
- Support for Multiple CSPs
  - Easy addition of more types of CSPs, e.g., of different HSM types
  - Enable the use of different CSP on different system components transparently

Fabric-CA


Membership Services Provider
Implements
Ecert
Tcert
Fabric-CA
Root Certificate Authority
HSM
LDAP
cluster DB
Authenticate Enroll ID, secret

- Default implementation of the Membership Services Provider Interface.
- Issues Ecerts (long-term identity) and Tcerts (disposable certificate)
- Supports clustering for HA characteristics
- Supports LDAP for user authentication
- Supports HSM

Overview of Application Flow



- Developers create application and smart contracts (chaincodes)
  - Chaincodes are deployed on the network and control the state of the ledger
  - Application handles user interface and submits transactions to the network which call chaincodes
- Network emits events on block of transactions allowing applications to integrate with other systems

Contents

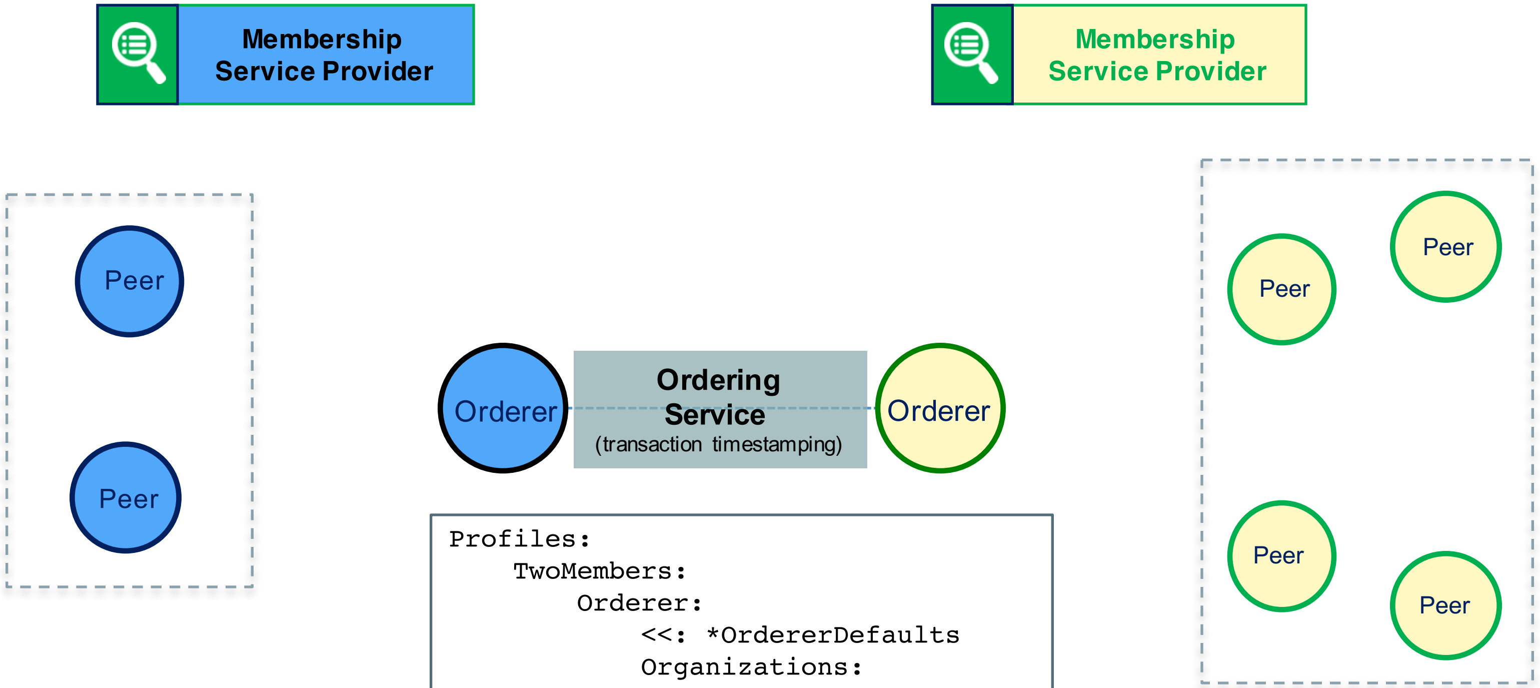
- What
- > How
- When

Bootstrapping a Network

- Decide on members (MSPs) controlling the ordering service
  - Set up MSP configuration for each member (root certs, signing certs, key, admins)
  - Set up policies governing the network (who has privilege to modify config and create channels)
  - Start up orderers with the configuration
- Each member decides on the number of peers to participate
  - For each peer, issue peer identity (local MSP configuration) and start it up

- At this point, we have a network of peers and orderers
  - Peers are not yet connected to orderers nor to each other

Two-Member Network


Membership Service Provider
Membership Service Provider
Peer
Peer
Peer
Peer
Peer
Peer
Orderer
Ordering Service
(transaction timestamping)
Orderer

```
Profiles:
    TwoMembers:
        Orderer:
            <<: *OrdererDefaults
            Organizations:
                - *Member1
                - *Member2
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *Member1
                - *Member2
```

Setting up Channels, Policies, and Chaincodes

- Depending on the business network, 1 or more channels may be required
- To create a channel, send a configuration transaction to the ordering service specifying members of the channel, ACL policies, anchor peers
  - The configuration becomes part of the genesis block of the channel
  - Then notify members to join the channel (a peer may join multiple channels)
- Deploy chaincodes on the channel with appropriate endorsement policy
- Now the network is ready for transacting

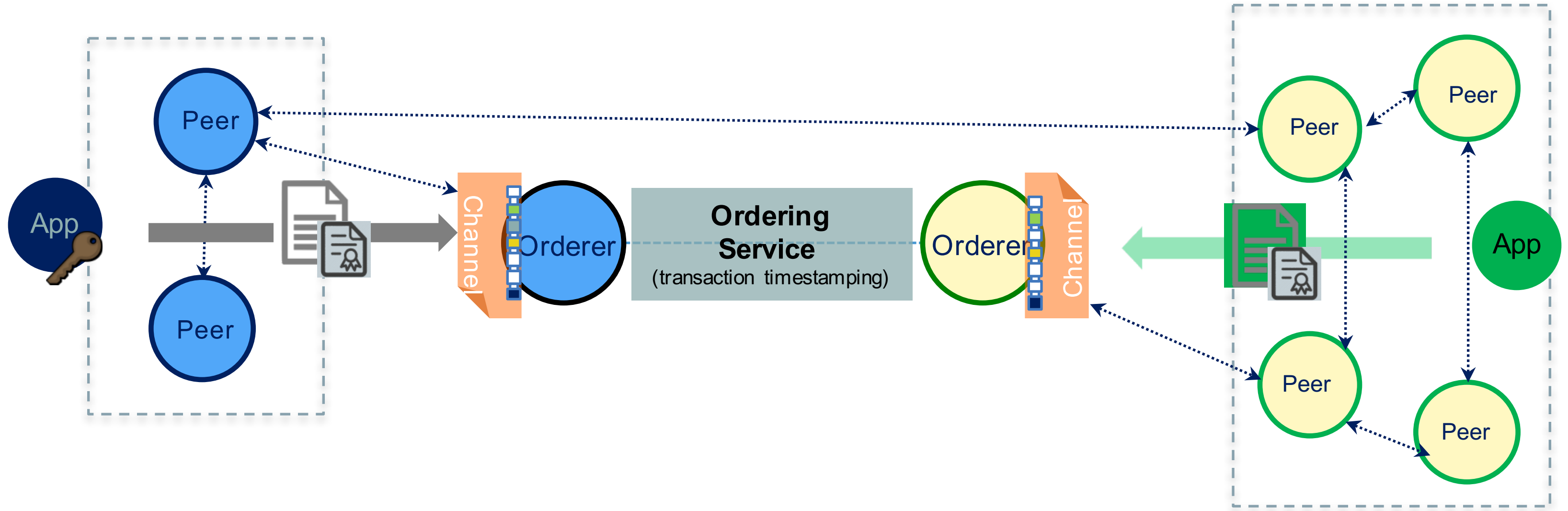
Consensus Redefined

- Consensus = Transaction Endorsement + Ordering + Validation
- Endorsement: Each stakeholder decides whether to accept or reject a transaction
- Ordering: Sort all transactions within a period into a block to be committed in that order
- Validation: Verify transaction endorsement satisfied the policy and transaction transformation is valid according to multiversion concurrency control (MVCC)

Transaction Endorsement

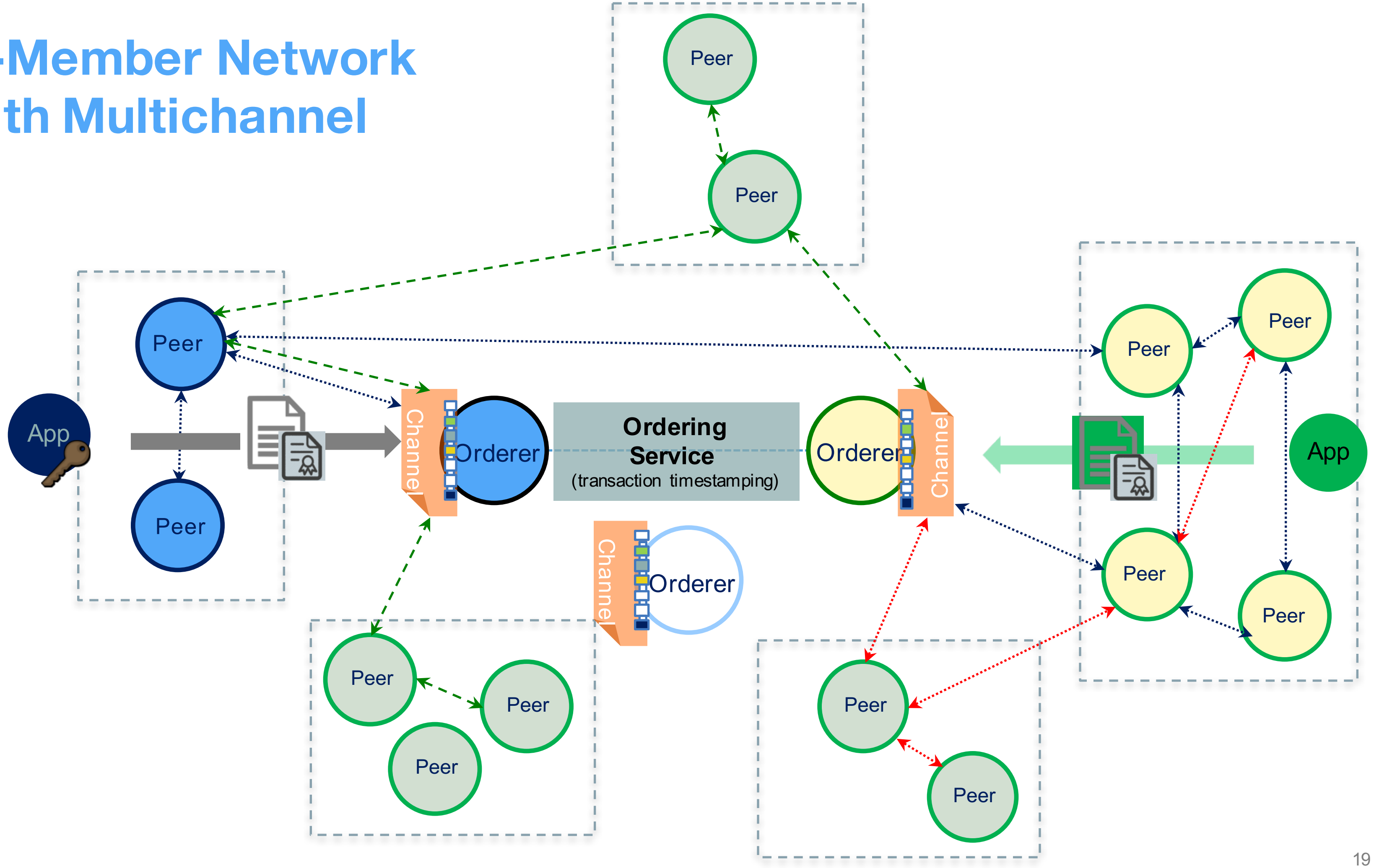
- An endorsement is a signed response of the result of a transaction execution
- An endorsement policy encapsulates the requirement for a transaction to be accepted by the stakeholders, either explicit or implicit
  - A signature from both member1 and member2
  - Either a signature from both member1 and member2 or a signature from member3
  - A signature from John Doe
- The endorsement policy is specified during a chaincode instantiation on a channel; each channel-chaincode may have different endorsement policy

Two-Member Network with A Channel

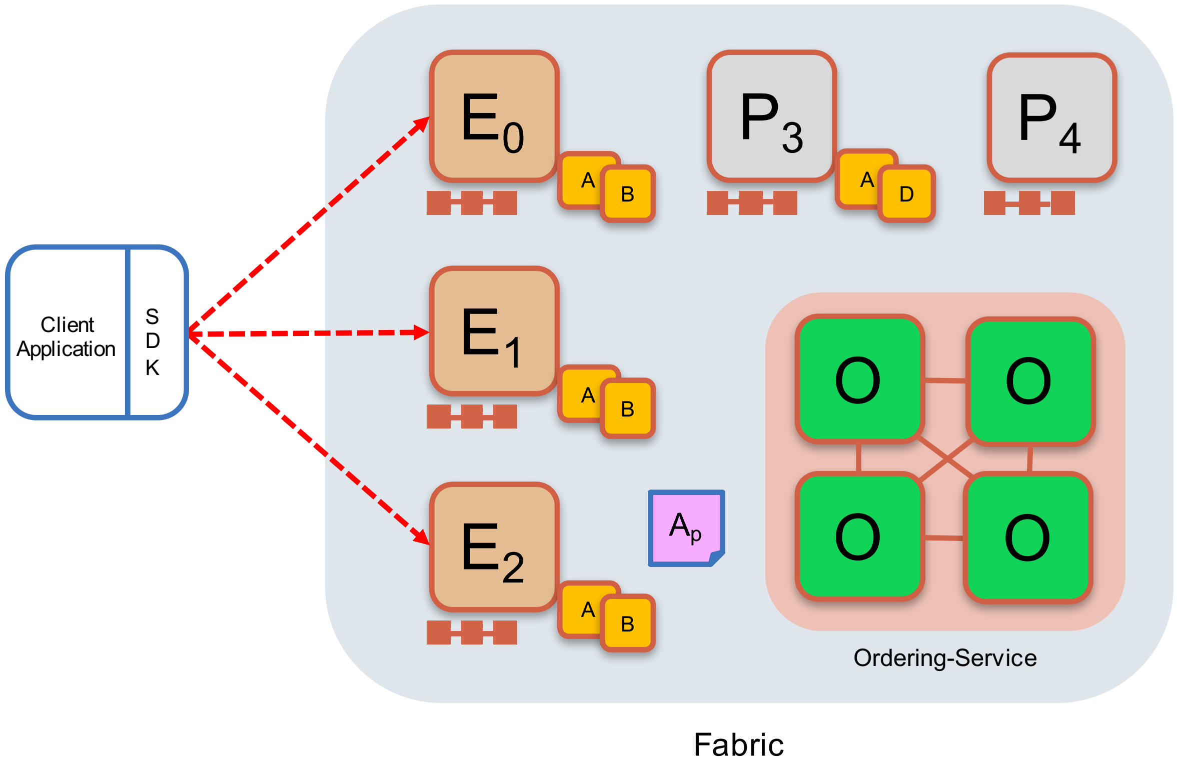


What if we want to add more members ?

N-Member Network with Multichannel



Sample transaction: Step 1/7 – Propose transaction



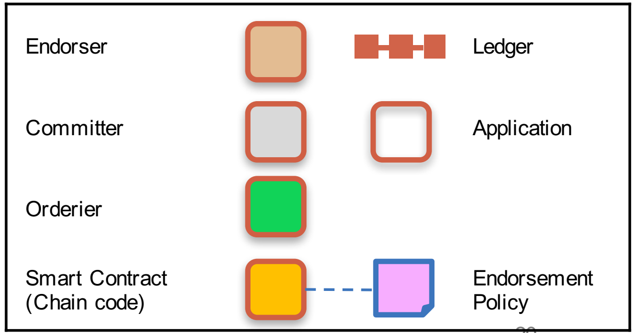
Application proposes transaction

Endorsement policy:

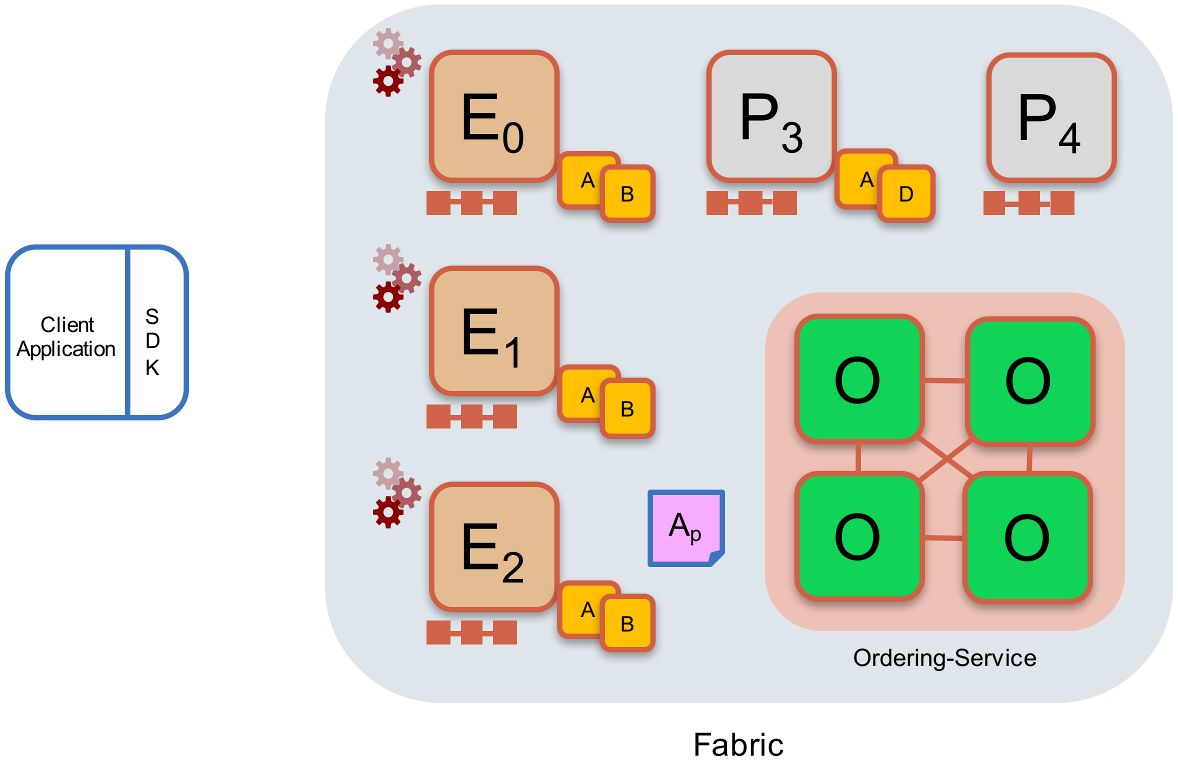
- "E_0, E_1 and E_2 must sign"
- (P_3, P_4 are not part of the policy)

Client application submits a transaction proposal for **chaincode A.** It must target the required peers **{E_0, E_1, E_2}**

Key:



Sample transaction: Step 2/7 – Execute proposal



Endorsers Execute Proposals

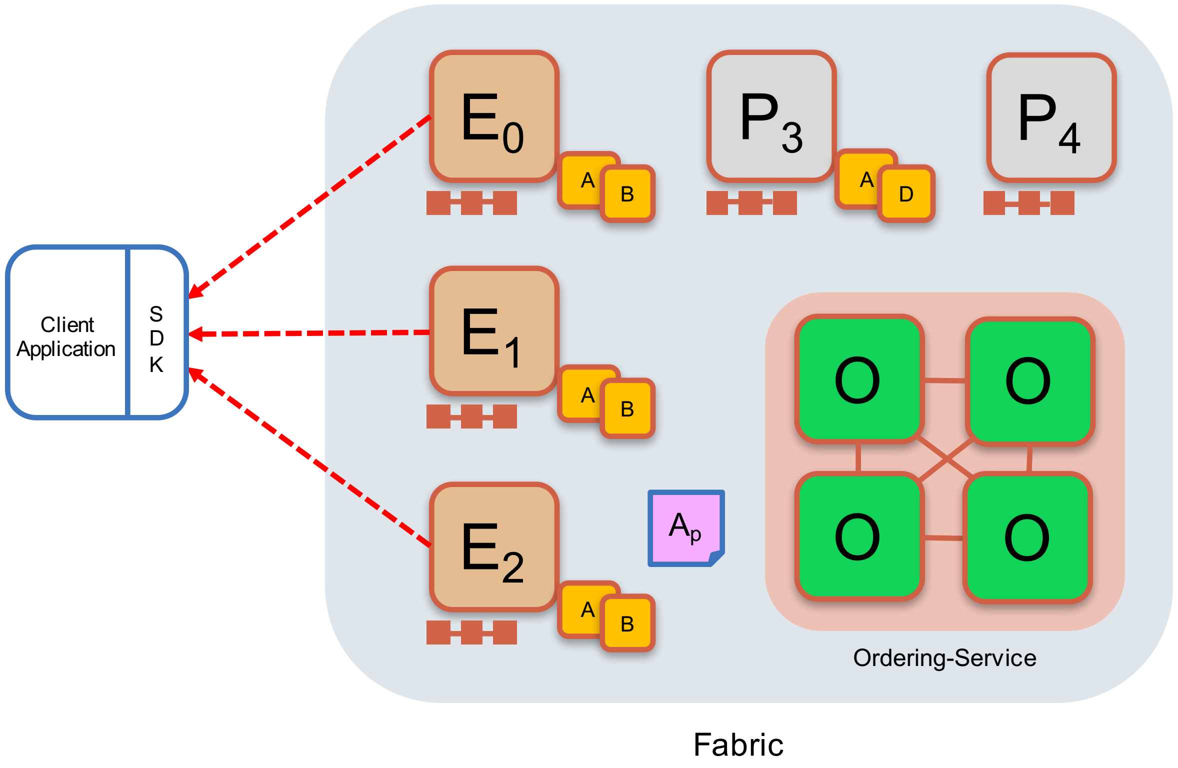
E_0, E_1 & E_2 will each execute the *proposed* transaction. None of these executions will update the ledger

Each execution will capture the set of **R**ead and **W**ritten data, called **RW sets,** which will now flow in the fabric.

Key:

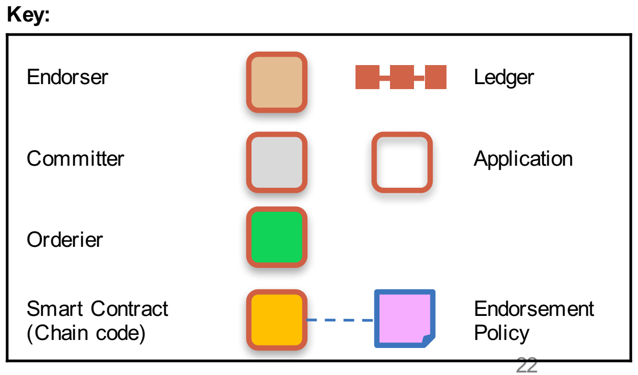
Endorser	🟧	▬	Ledger
Committer	⬜	□	Application
Orderier	🟩		
Smart Contract (Chain code)	🟨	🟪	Endorsement Policy

Sample transaction: Step 3/7 – Proposal Response

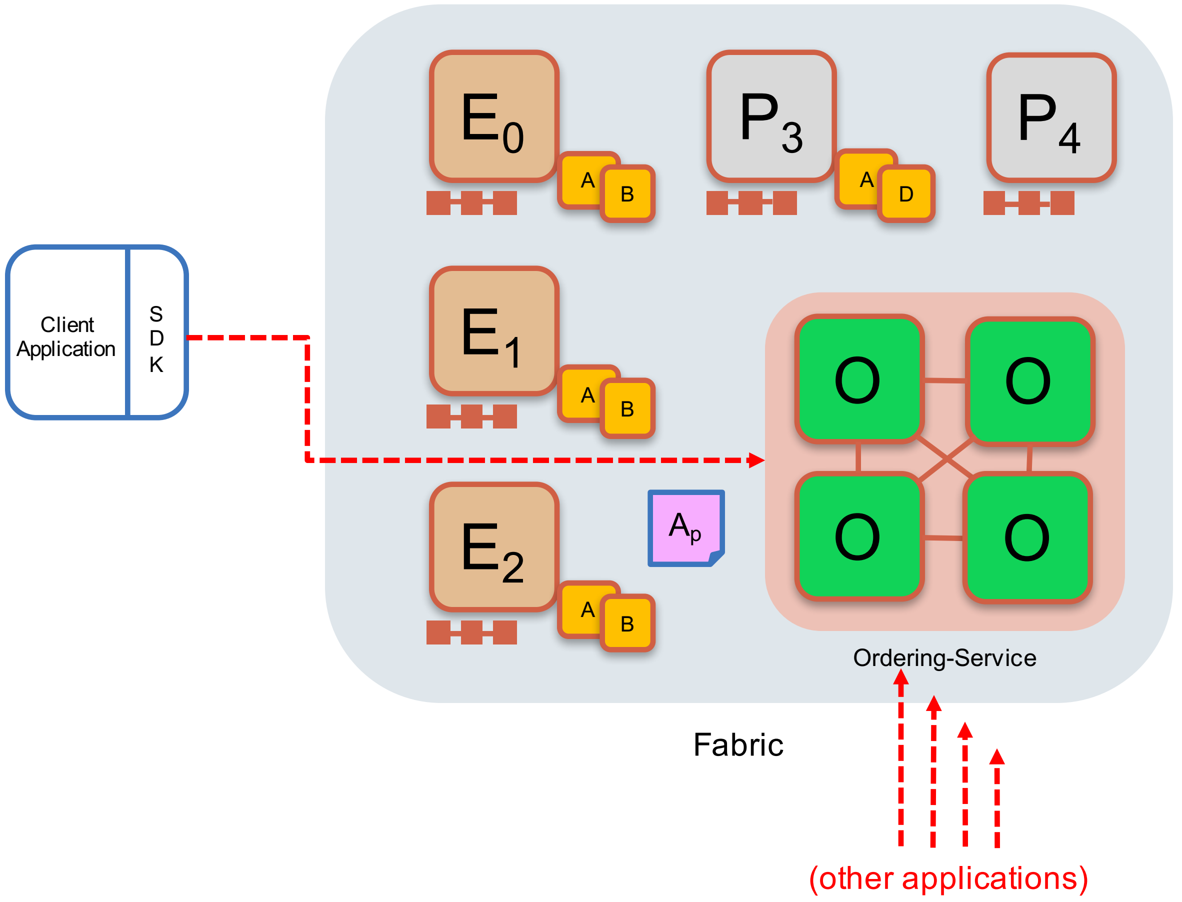


Application receives responses

The RW sets are signed by each endorser and returned to the application



Sample transaction: Step 4/7 – Order Transaction

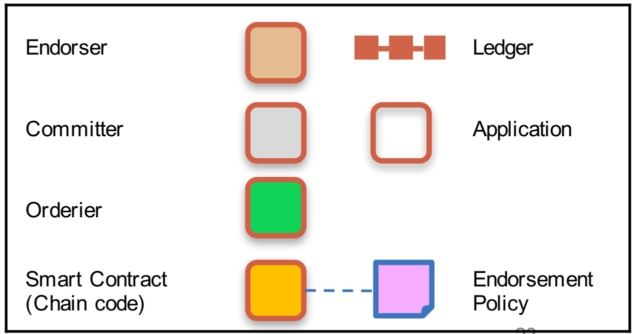


Application submits responses for ordering

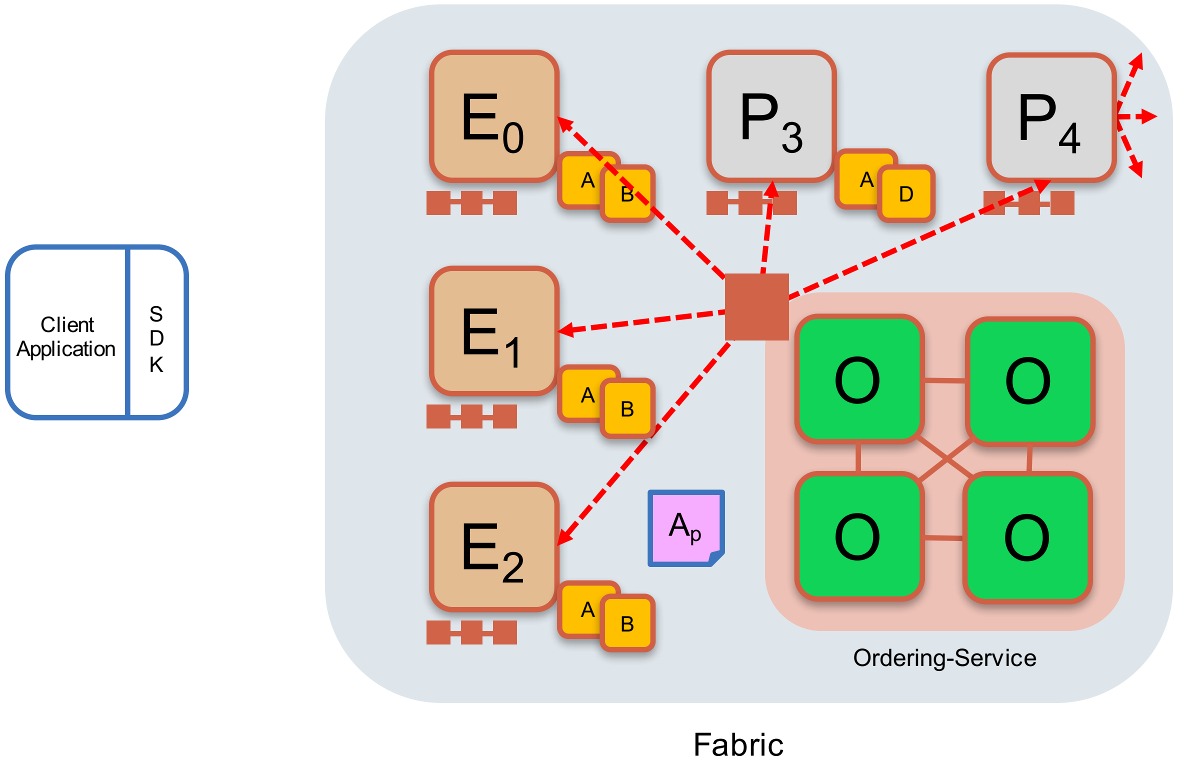
Application submits responses as a **transaction** to be ordered.

Ordering happens across the fabric in parallel with transactions submitted by other applications

Key:



Sample transaction: Step 5/7 – Deliver Transaction



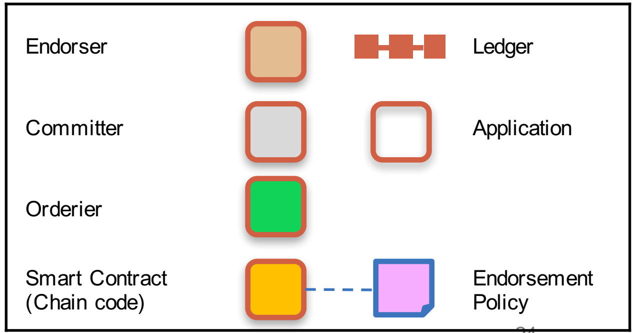
Orderer delivers to all committing peers

Ordering service collects transactions into blocks for distribution to committing peers. Peers can deliver to other peers using gossip (not shown)

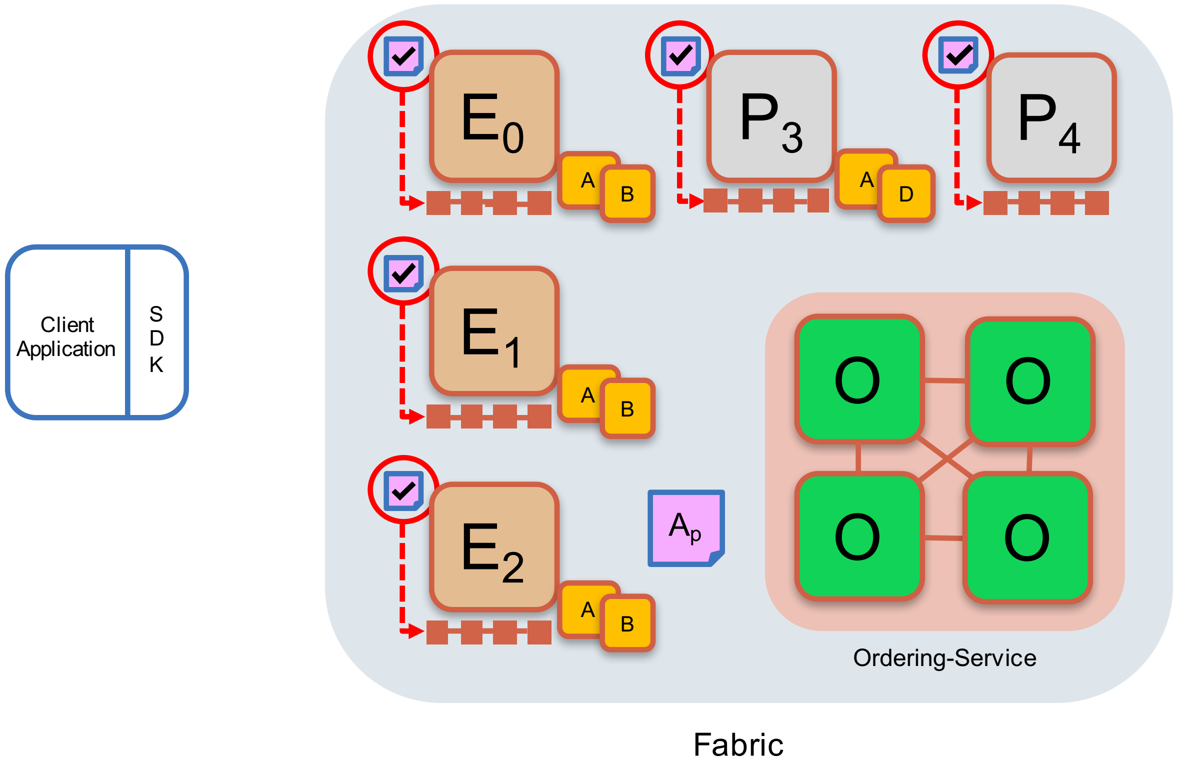
Different ordering algorithms available:

- SOLO (single node, development)
- Kafka (blocks map to topics)
- SBFT (tolerates faulty peers, future)

Key:



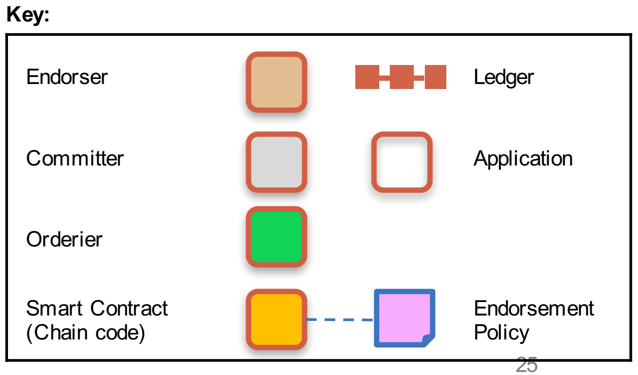
Sample transaction: Step 6/7 – Validate Transaction



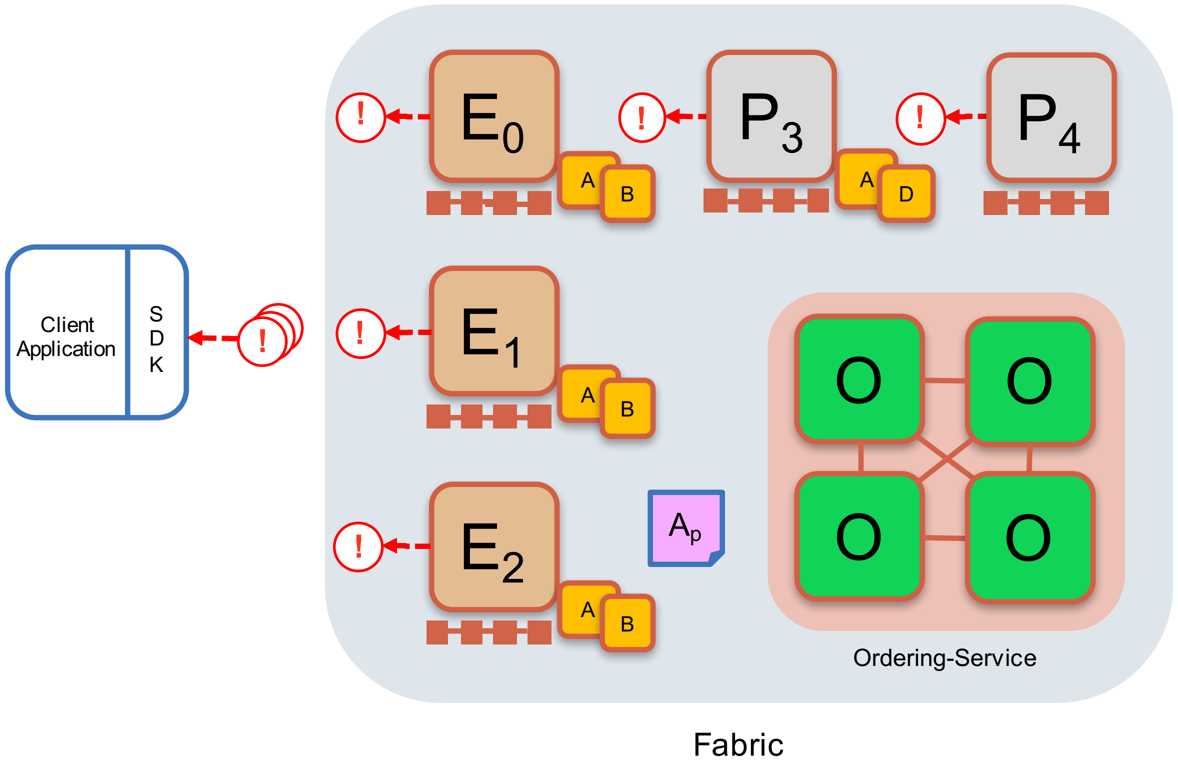
Committing peers validate transactions

Every committing peer validates against the endorsement policy. Also check RW sets are still valid for the current state

Transactions are written to the ledger and update caching DBs with validated transactions



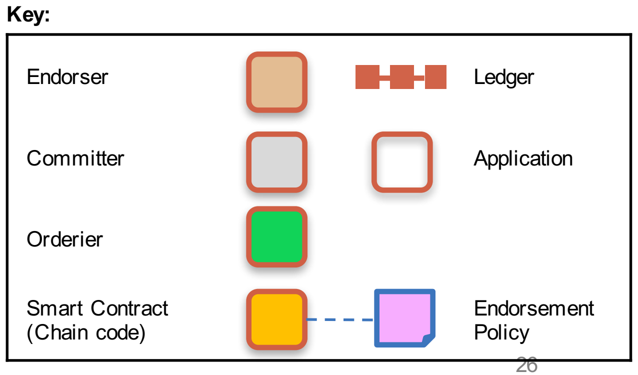
Sample transaction: Step 7/7 – Notify Transaction



Committing peers notify applications

Applications can register to be notified when transactions succeed or fail, and when blocks are added to the ledger

Applications will be notified by each peer to which they are connected



Contents

Hyperledger Fabric Roadmap

Hack Fest docker images

- **60 participates tested**
- Basic v1 architecture in place
- Add / Remove Peers
- Channels
- Node SDK
- Go Chaincode
- Ordering Solo
- Fabric CA

V1 Alpha *

- Docker images
- Tooling to bootstrap network
- Fabric CA or bring your own
- Java and Node SDKs
- Ordering Services - Solo and Kafka
- Endorsement policy
- Level DB and Couch DB
- Block dissemination across peers via Gossip

V1 GA *

- Hardening, usability, serviceability, load, operability and stress test
- Java Chaincode
- Chaincode ACL
- Chaincode packaging & LCI
- Pluggable crypto
- HSM support
- Consumability of configuration
- Next gen bootstrap tool (config update)
- Config transaction lifecycle
- Eventing security
- Cross Channel Query
- Peer management APIs
- Documentation

V Next *

- SBFT
- Archive and pruning
- System Chaincode extensions
- Side DB for private data
- Application crypto library
- Dynamic service discovery
- REST wrapper
- Python SDK
- Identity Mixer (Stretch)
- Tcerts

2016/17 December March June Future

Connect-a-thon

- 11 companies in Australia, Hungary, UK, US East Coast, US West Coast, Canada dynamically added peers and traded assets

Connect-a-cloud

- Dynamically connecting OEM hosted cloud environments to trade assets

HYPERLEDGER
BLOCKCHAIN TECHNOLOGIES FOR BUSINESS

*** Dates for Alpha, Beta, and GA are determined by Hyperledger community and are currently proposals.**

Proposed Alpha detailed content:
https://wiki.hyperledger.org/projects/proposedv1alphacontent

Where to Ask Questions

Hyperledger Community has moved off Slack to RocketChat.

Go to chat.hyperledger.org and register.

You will be required to have a linux foundation ID however. If you aren't registered with the Linux Foundation, get an ID from https://identity.linuxfoundation.org/

For questions on Version 1.0, go to the fabric-questions channel.

Also every day, the docker build status is posted when passing the continuous integration tests will be posted on fabric-ci (only posted when tests pass)

Useful Information To Get You Started

- **Documentation actively getting updated as we progress:** http://hyperledger-fabric.readthedocs.io/en/latest/

- **Support for Docker images for easy deployment for Hyperledger-fabric 1.0.** Docker images will be available for all major components to run a network (peers, solo orderer, CLI, CA, Kafka, CouchDB). A "Getting started" section will be available in the Hyperledger-fabric publications. Getting started will help a developer or user to start the network, run a simple application , and learn the basics of running fabric 1.0. See: http://hyperledger-fabric.readthedocs.io/en/latest/

- **Support for a tool that helps bootstrap a network.**
  - Configuration Transaction Generator (configtxgen): a tool designed to generate the genesis block to bootstrap the orderers, and the configuration transaction artifacts used for channel creation
  - Crypto Materials Generator (cryptogen): a tool to generate a multi-root tree of certificates and signing identities (private keys) to use to bootstrap orderers, peers, fabric-cas and apps