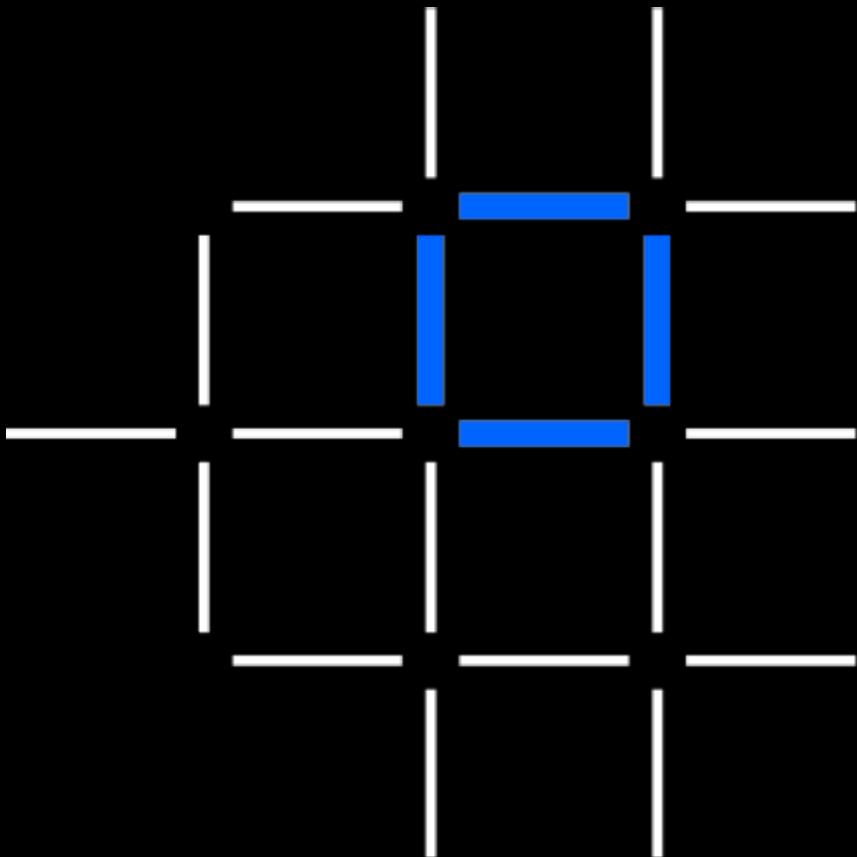


Hyperledger Fabric: Under the Covers

A Technical Deep-Dive on Hyperledger Fabric

*Jeff Tennenbaum,
IBM Blockchain Solutions Architect
Tennenjl@us.ibm.com*



V4.09, 27, Aug 2020

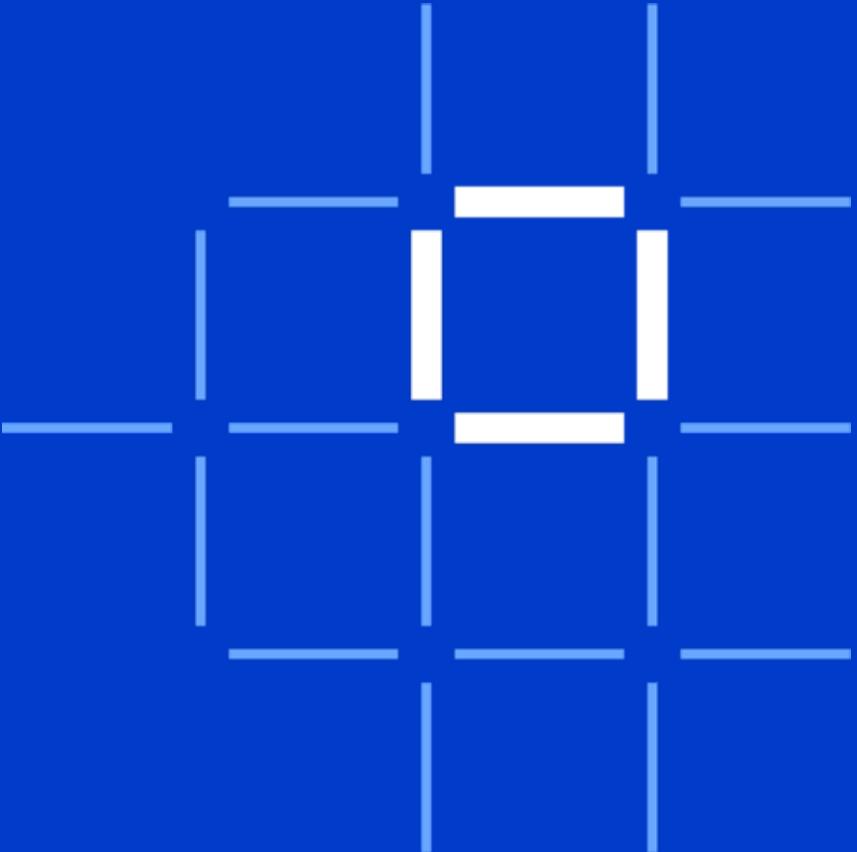
IBM Blockchain



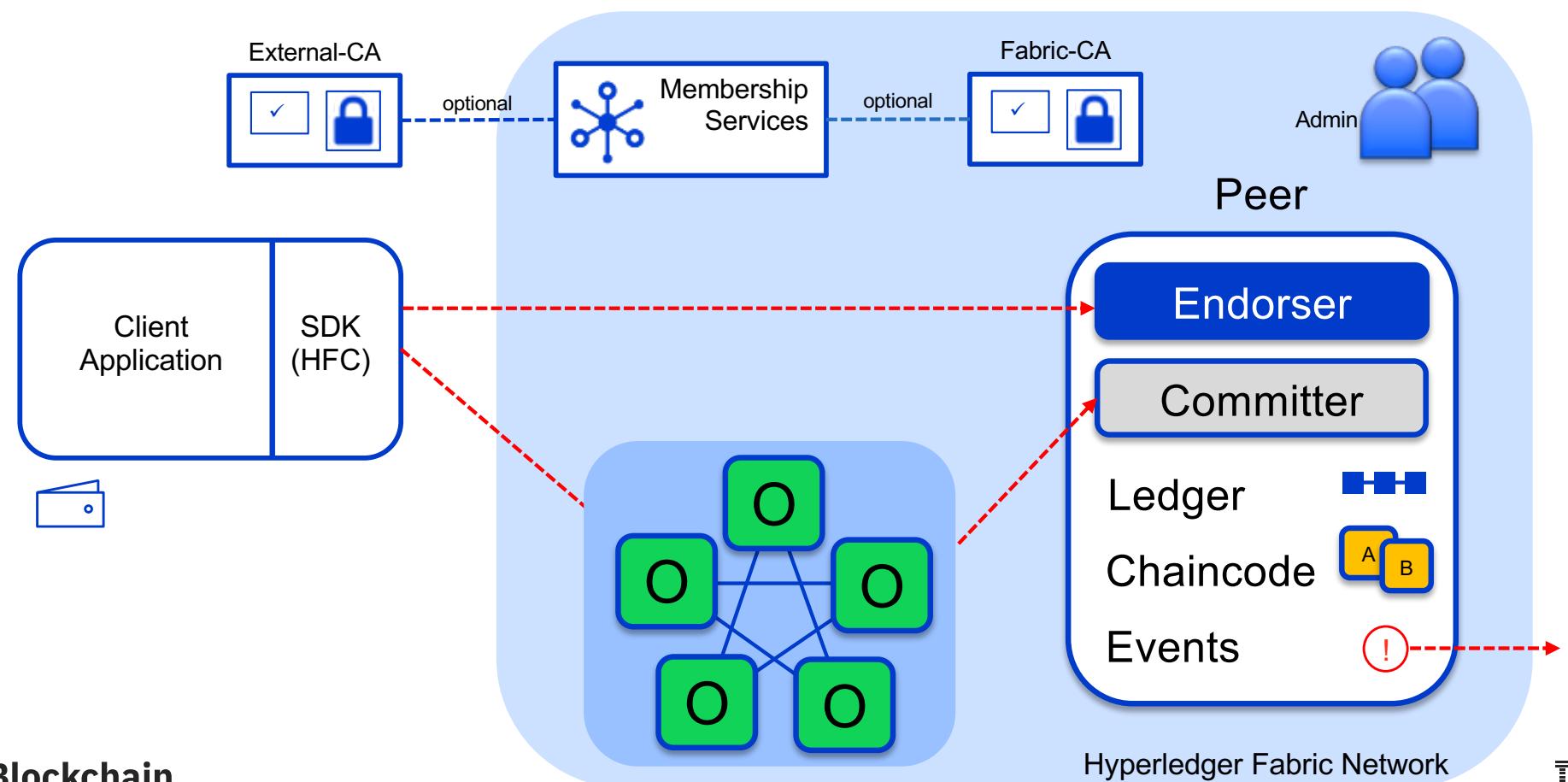
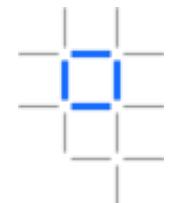


Hyperledger Fabric Technical Dive

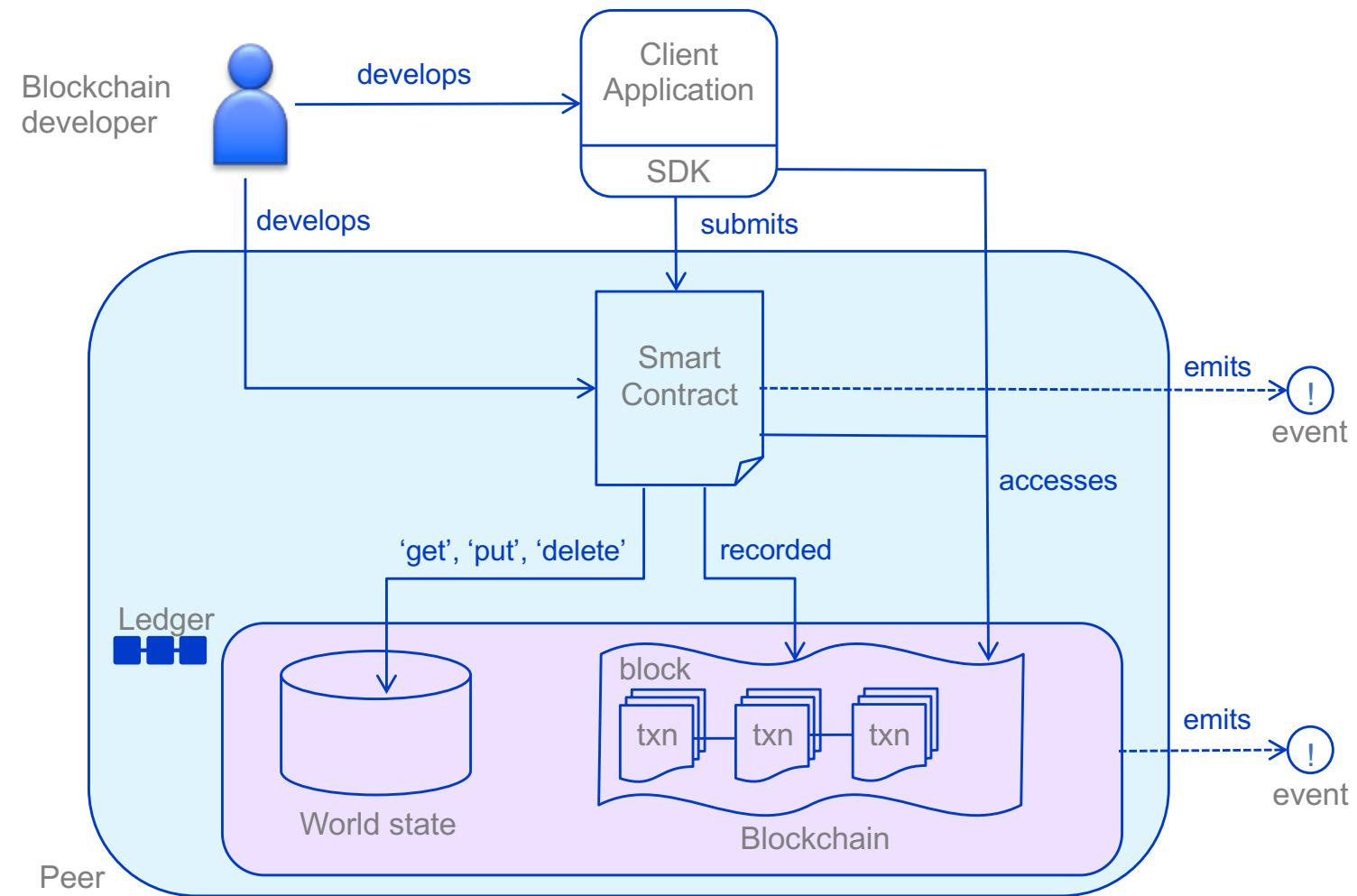
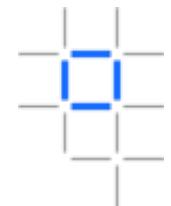
- [Architectural Overview]
- Network Consensus
- Channels and Ordering Service
- Components
- Network setup
- Endorsement Policies
- Membership Services



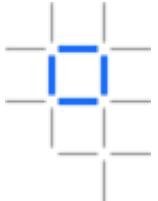
Hyperledger Fabric V1.x Architecture



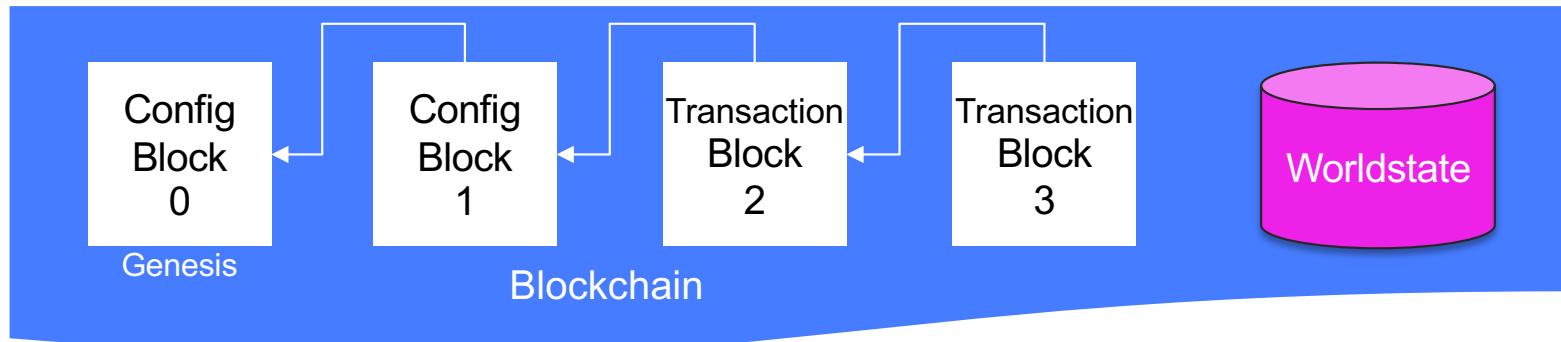
How applications interact with the ledger



Fabric Ledger



- The **Fabric ledger** is maintained by each peer and includes the **blockchain** and **worldstate**
- A separate ledger is maintained for each channel the peer joins
- Transaction **read/write sets** are written to the blockchain
- **Channel configurations** are also written to the blockchain
- The worldstate can be either LevelDB (default) or CouchDB
 - **LevelDB** is a simple key/value store
 - **CouchDB** is a document store that allows complex queries
- The smart contract decides what is written to the worldstate (If consensus is reached)



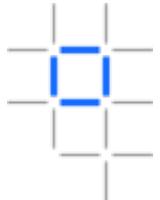


Hyperledger Fabric Technical Dive

- Architectural Overview
- [Network Consensus]
- Channels and Ordering Service
- Components
- Network setup
- Endorsement Policies
- Membership Services



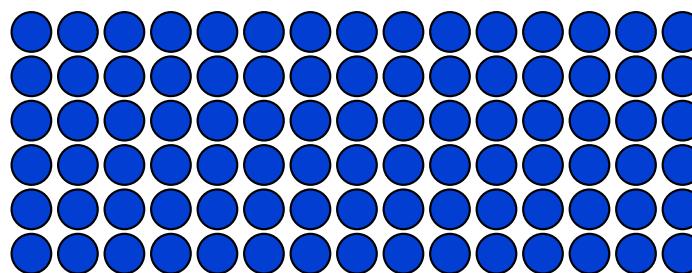
A simple **consensus** example (1/4) – a puzzle!



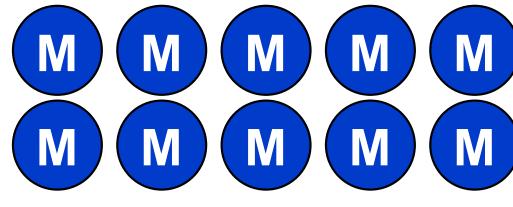
- **Situation:** A presenter in a room with 100 people, 10 of whom are mathematicians!
- **Problem:** Answer the question: **What is the square root of 2401?**
- **Question:** Can everyone agree on an answer?

a puzzle
a process
a problem
a policy

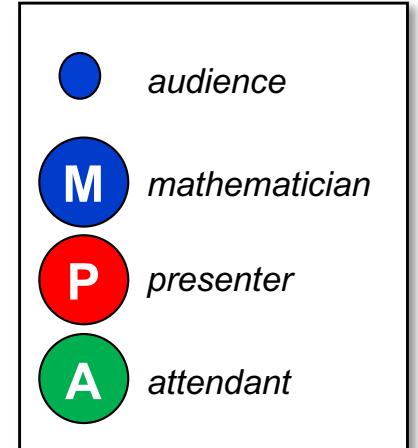
You are here! →



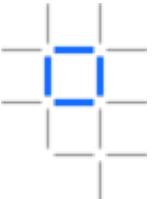
Or here! →



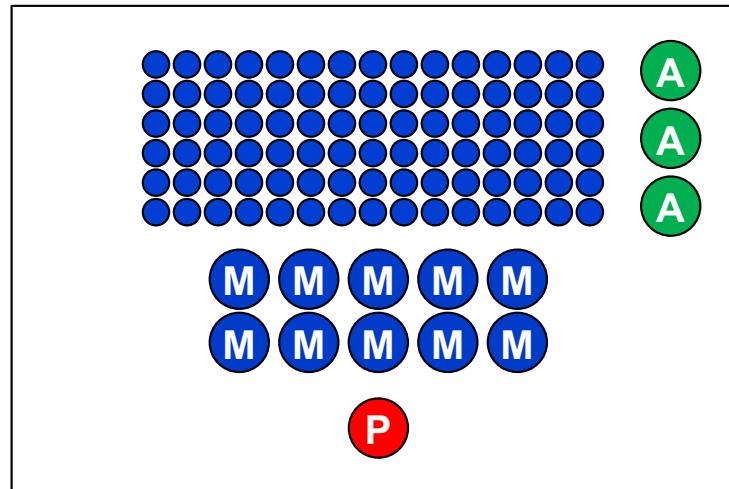
But not here ;) →



A simple **consensus** example (2/4) – a process



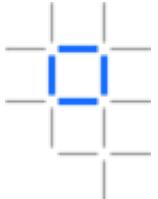
a puzzle
a process
a problem
a policy



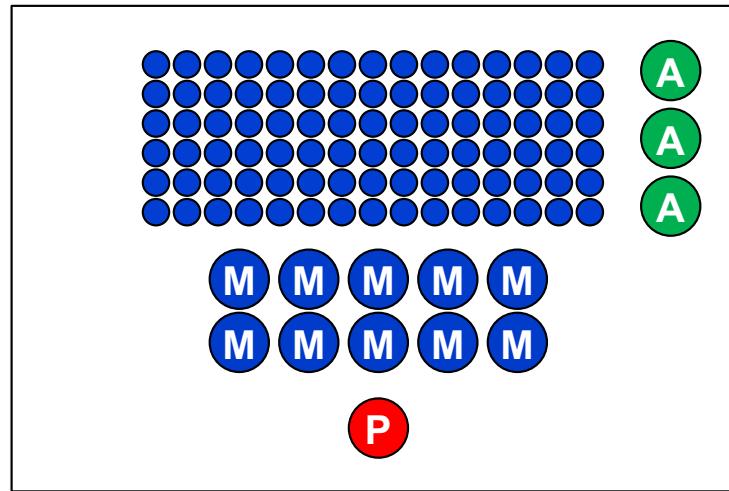
Solution approach

1. The presenter asks every mathematician to calculate $\sqrt{2401}$
2. Every mathematician writes their answer on a piece of paper, signs with their (digital) signature
3. The presenter collects every mathematician's signed response
4. The room attendants distribute copies of the signed answer to every member of the audience
5. Every audience member checks that every mathematician agrees the answer, or not!
6. Everyone is happy; there is **consensus!**

A simple **consensus** example (3/4) – a problem



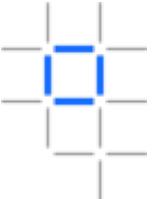
a puzzle
a process
a problem
a policy



Issues

1. How many mathematicians should the presenter ask?
 2. How many mathematicians need to agree for an answer to be correct?
 3. What happens if the mathematics disagree on the answer, or the presenter selects a bad answer?
- Everyone needs to agree what constitutes a **good** answer

A simple **consensus** example (4/4) – a policy

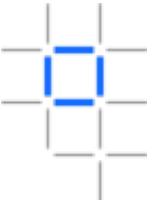


a puzzle
a process
a problem
a policy

The importance of policy

1. The audience agrees a **policy for good answers**
 2. Example policies:
 - **all mathematicians must agree**
 - **the majority of mathematicians must agree**
- Using this **policy** everyone can now agree what makes a **good** answer

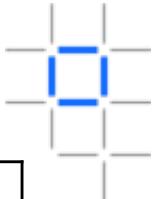
Transaction Endorsement



We, in Convention by the Unanimous Consent of the States present the Seventeenth Day of September in the Year of our Lord one thousand seven hundred and Eighty seven and of the Independance of the United States of America the Fifteenth In witness whereof We have hereunto subscribed our Names,

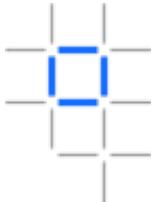
<i>Delaware</i> { G. Read Gunning Bedford jun. John Dickinson Richard Bassett T. C. Broom James M'Henry	<i>Penns.</i> Wm. of Tho. Jenifer Danl. Carroll John Blair - James Madison Jr.	<i>N. H.</i> G. Washington - Pres and Deputy from Virginia John Langdon Nicholas Gilman
<i>Maryland</i> { D. of Tho. Jenifer	<i>Mass.</i> Nathaniel Gorham Rufus King W.. Jenr. Johnson	<i>Conn.</i> Roger Sherman
<i>Virginia</i> { Wm. Blount P. D. Morris	<i>New York</i> Alexander Hamilton	<i>N. Y.</i> W. Livingston David Brearley J. W. Patryee Jona. Dayton
<i>North Carolina</i> { R. Dods Spaight. A. Williamson J. Rutledge H. L. Smith & Pinckney	<i>P. S.</i> Franklin Thomas Jefferson Robt Morris	

Nodes and roles

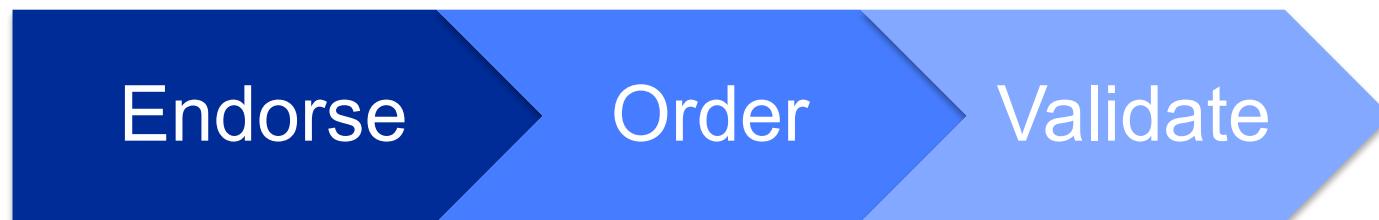


	<p>Peer: Maintains ledger and state. Commits transactions. May also endorse transactions by receiving a transaction proposal and responds by granting or denying endorsement (must hold smart contract to endorse).</p>
	<p>Ordering Node: Approves the inclusion of transaction blocks into the ledger and communicates with committing and endorsing peer nodes. Does not hold smart contract. Does not hold ledger.</p>
	<p>Certificate Authority (CA): Issues digital certificates to member organizations and their users.</p>

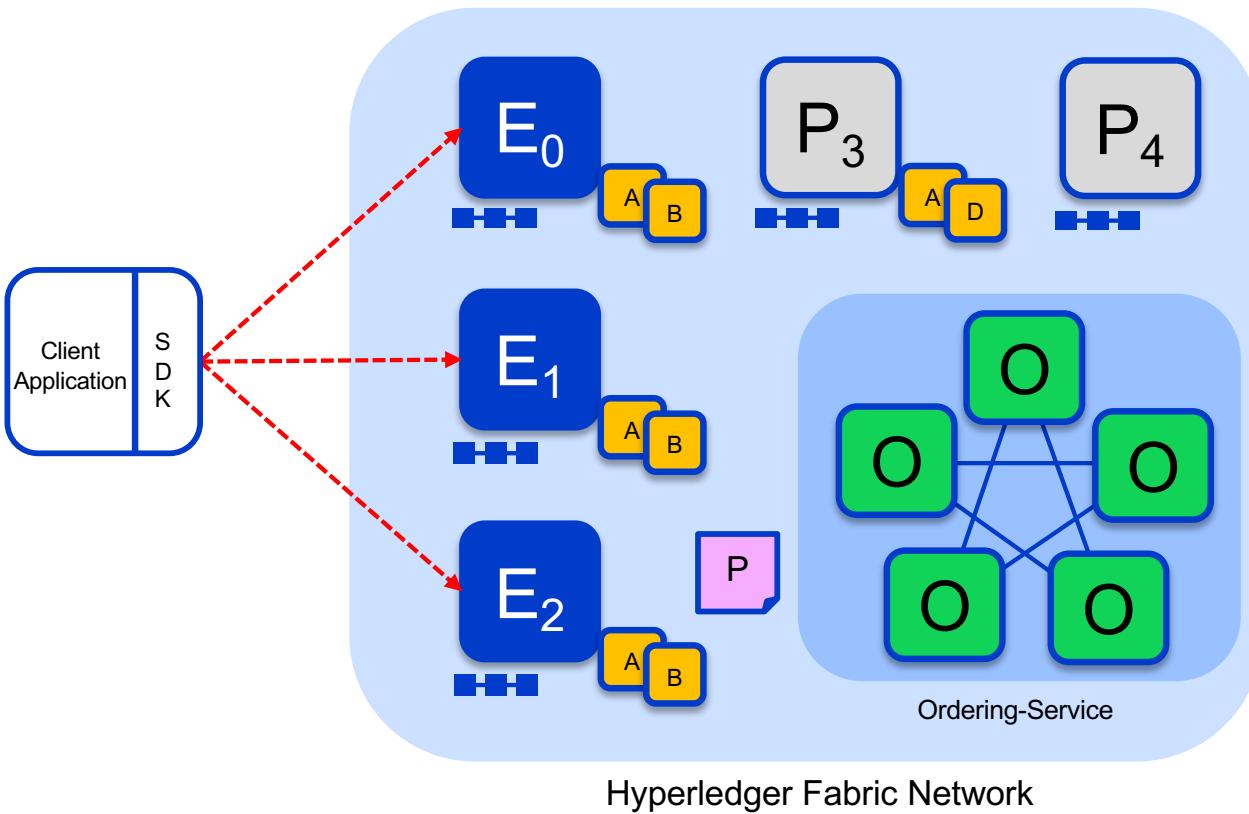
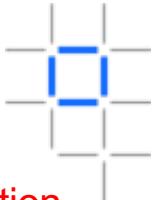
Hyperledger Fabric Consensus



Consensus is achieved using the following transaction flow:



Sample transaction: Step 1/7 – Propose transaction



Application proposes transaction

Endorsement policy:

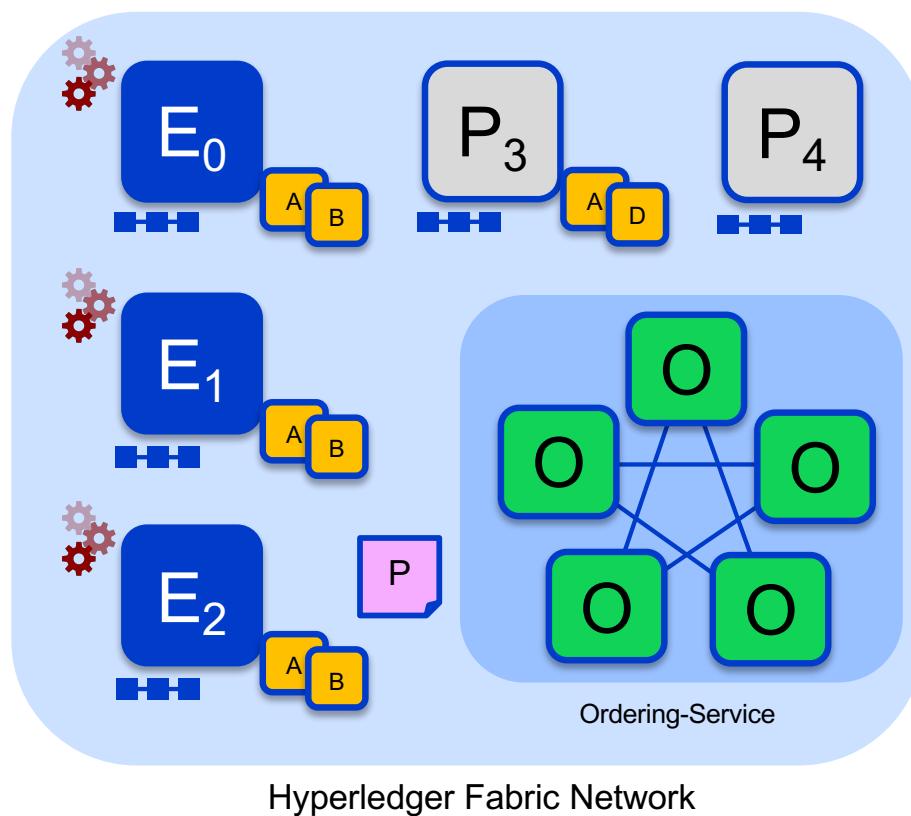
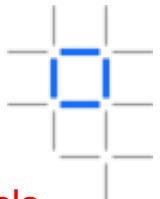
- " E_0, E_1 and E_2 must sign"
- (P_3, P_4 are not part of the policy)

Client application submits a transaction proposal for Smart Contract A. It must target the required peers $\{E_0, E_1, E_2\}$

Key:

Endorser		Ledger
Committing Peer		Application
Ordering Node		
Smart Contract (Chaincode)		Endorsement Policy

Sample transaction: Step 2/7 – Execute proposal



Endorsers Execute Proposals

E₀, E₁ & E₂ will each execute the proposed transaction. None of these executions will update the ledger

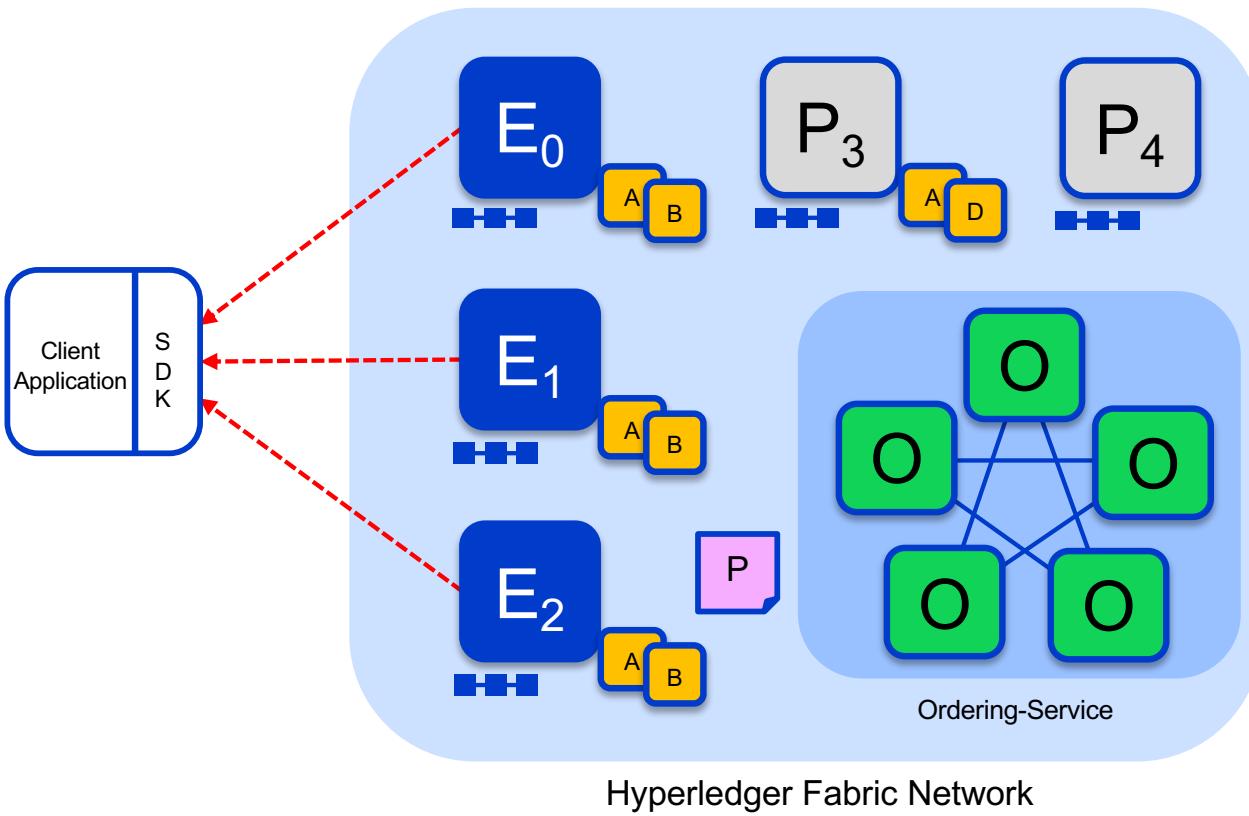
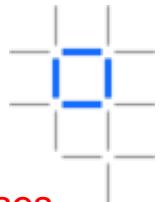
Each execution will capture the set of Read and Written data, called RW sets, which will now flow in the fabric.

Transactions can be signed & encrypted

Key:

Endorser		Ledger
Committing Peer		Application
Ordering Node		
Smart Contract (Chaincode)		Endorsement Policy

Sample transaction: Step 3/7 – Proposal Response



Application receives responses

RW sets are asynchronously returned to application

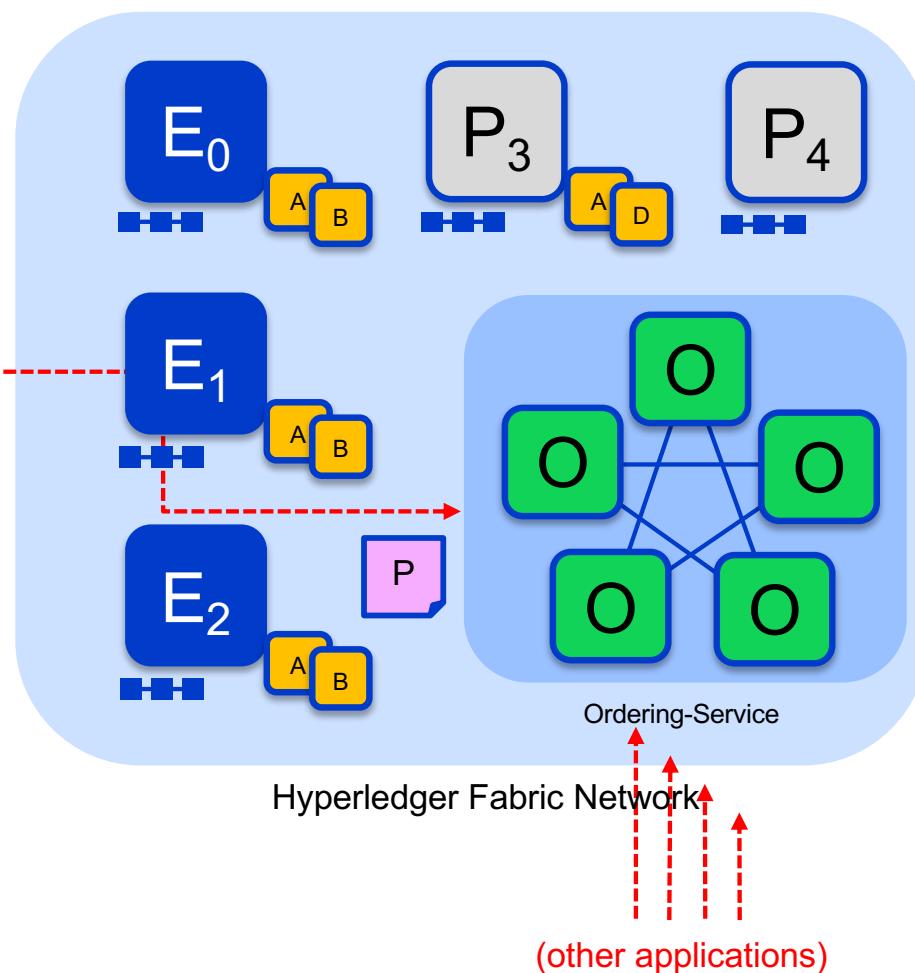
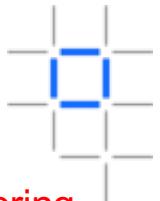
The RW sets are signed by each endorser, and also includes each record version number

(This information will be checked much later in the consensus process)

Key:

Endorser		Ledger
Committing Peer		Application
Ordering Node		
Smart Contract (Chaincode)		Endorsement Policy

Sample transaction: Step 4/7 – Order Transaction



Responses submitted for ordering

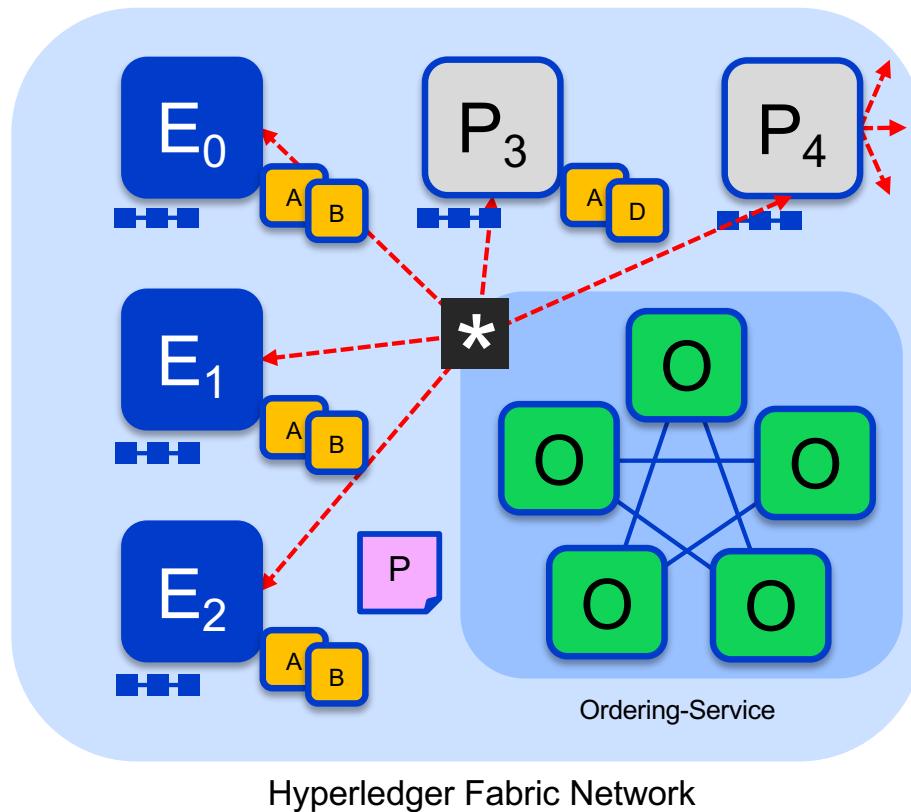
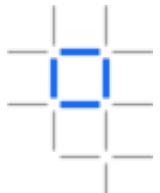
Application submits responses as a transaction to be ordered.

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

Key:

Endorser		Ledger
Committing Peer		Application
Ordering Node		
Smart Contract (Chaincode)		Endorsement Policy

Sample transaction: Step 5/7 – Deliver Transaction



Orderer delivers to committing peers

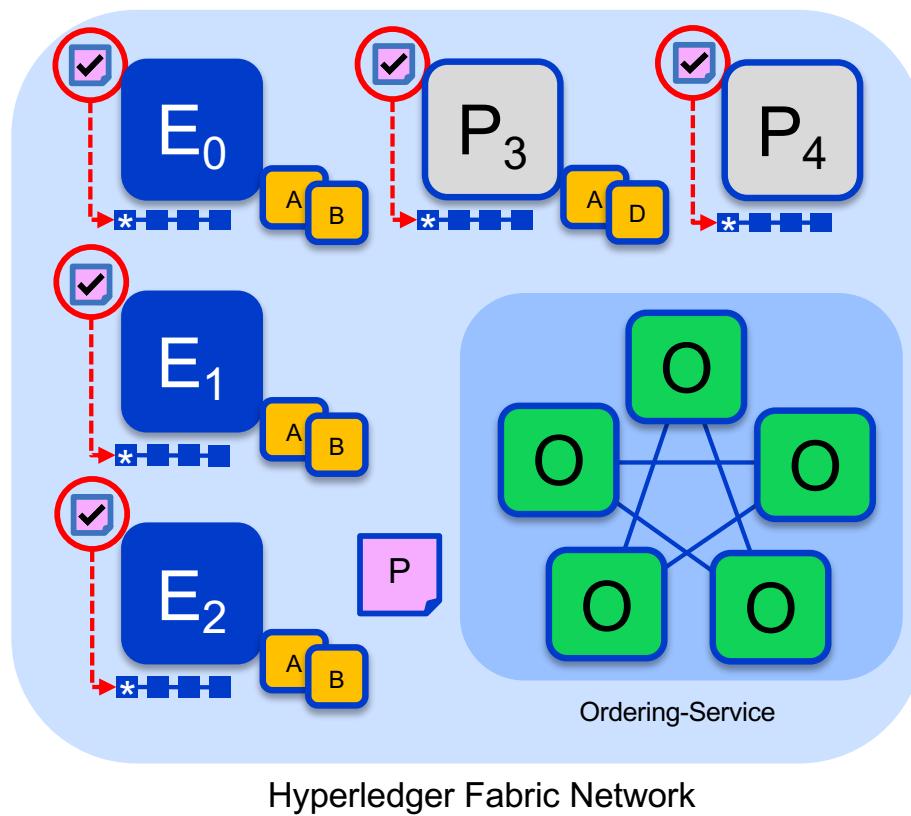
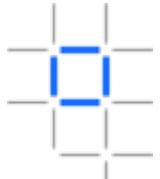
Ordering service collects transactions into proposed blocks for distribution to committing peers. Peers can deliver to other peers in a hierarchy (not shown)

RAFT consensus provides crash fault tolerance

Key:

Endorser		Ledger
Committing Peer		Application
Ordering Node		
Smart Contract (Chaincode)		Endorsement Policy

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 current world state

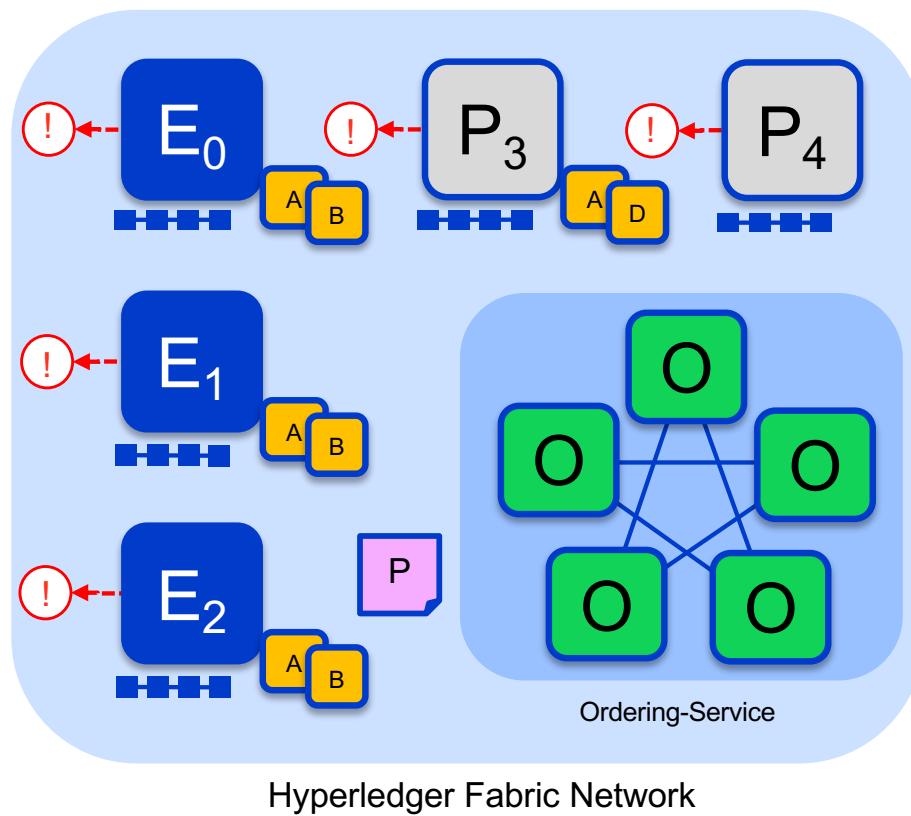
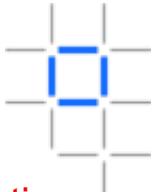
Validated transactions are applied to the world state and retained on the ledger

Invalid transactions are also retained on the ledger but do not update world state

Key:

Endorser		Ledger
Committing Peer		Application
Ordering Node		
Smart Contract (Chaincode)		Endorsement Policy

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

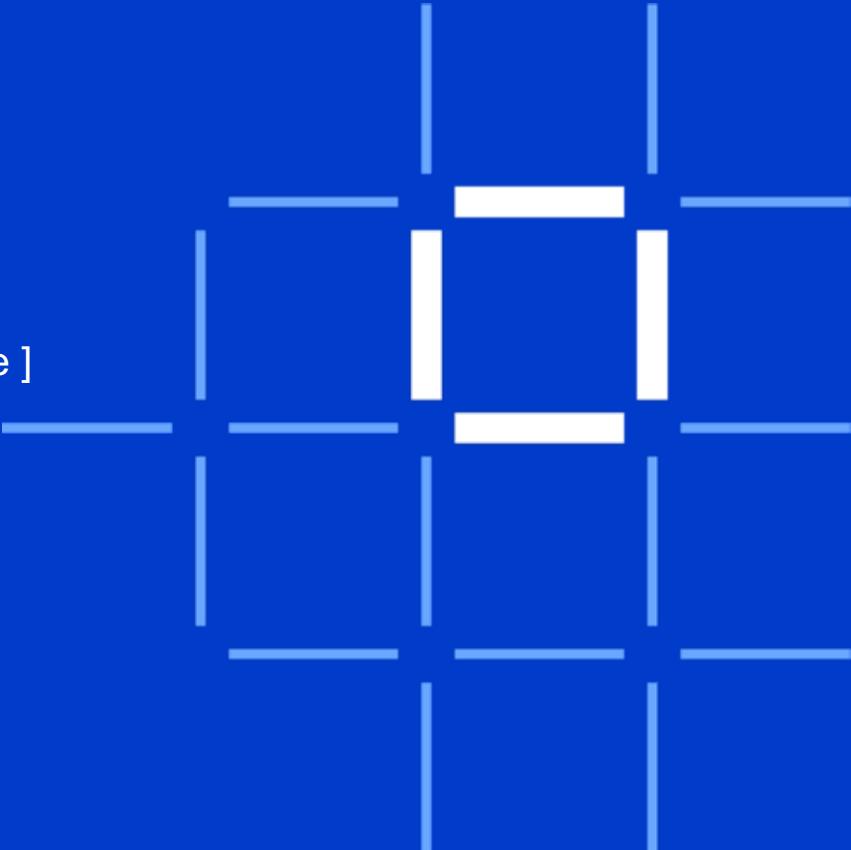
Key:

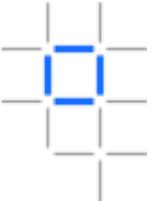
Endorser		Ledger
Committing Peer		Application
Ordering Node		
Smart Contract (Chaincode)		Endorsement Policy



Hyperledger Fabric Technical Dive

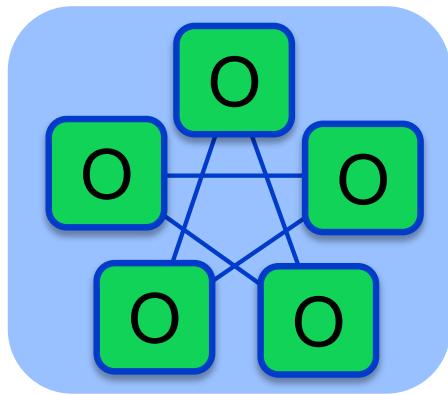
- Architectural Overview
- Network Consensus
- [Channels and Ordering Service]
- Components
- Network setup
- Endorsement Policies
- Membership Services





Ordering Service

The ordering service packages transactions into blocks to be delivered to peers. Communication with the service is via channels.



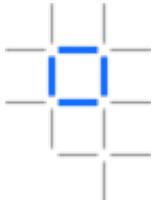
Ordering-Service

- RAFT consensus for the ordering service

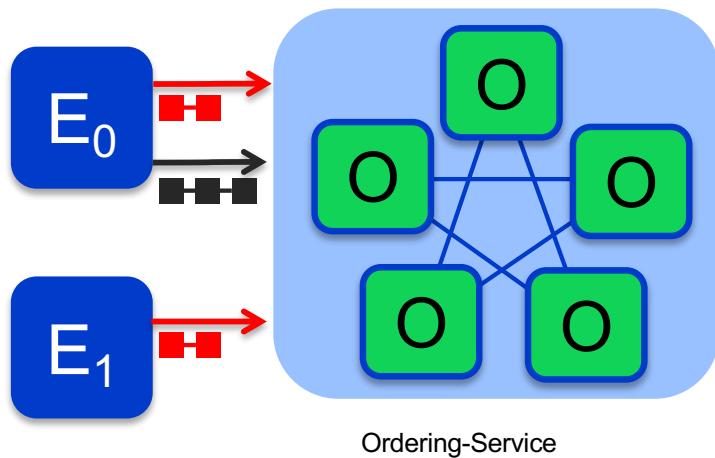
- Crash fault tolerant consensus
- 3 nodes minimum
- Odd number of nodes recommended
 - $N/2 + 1$ nodes required

<http://thesecretlivesofdata.com/raft/>

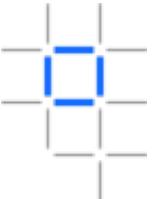
Channels



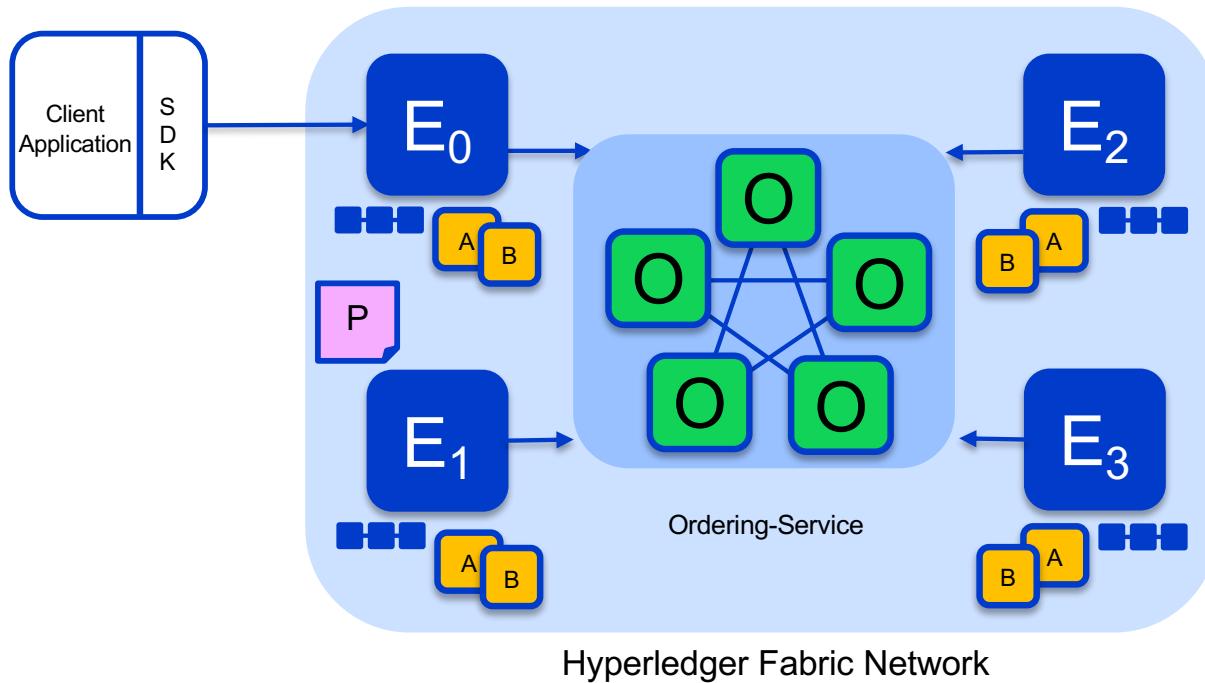
Channels provide privacy between different ledgers



- Ledgers exist in the scope of a channel
 - Channels can be shared across an entire network of peers
 - Channels can be permissioned for a specific set of participants
- Chaincode is **installed** on peers to access the worldstate
- Chaincode is **instantiated** on specific channels
- Peers can participate in multiple channels
- Concurrent execution for performance and scalability



Single Channel Network

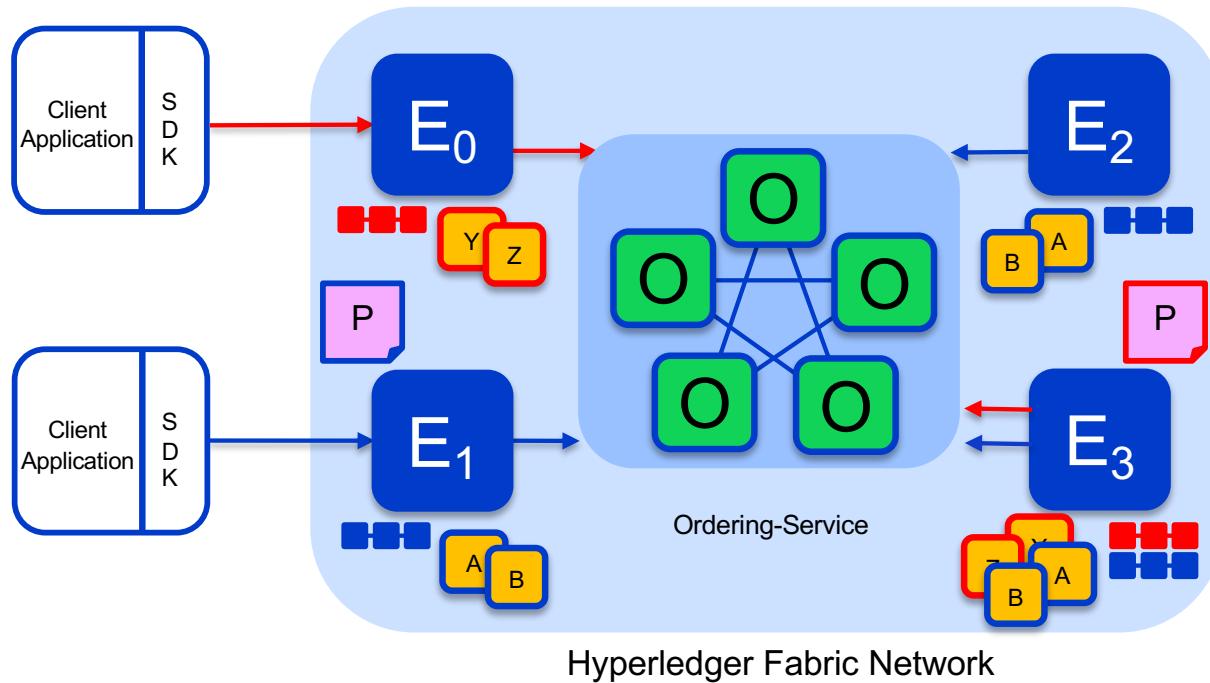
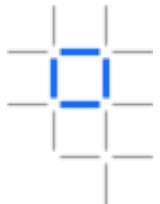


- All peers connect to the same system channel (blue).
- All peers have the same chaincode and maintain the same ledger
- Endorsement by peers **E₀**, **E₁**, **E₂** and **E₃**

Key:

Endorser		Ledger
Committing Peer		Application
Ordering Node		
Smart Contract (Chaincode)		Endorsement Policy

Multi Channel Network

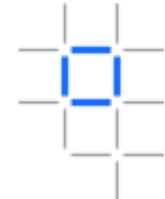


- Peers E₀ and E₃ connect to the red channel for chaincodes Y and Z
- E₁, E₂ and E₃ connect to the blue channel for chaincodes A and B

Key:

Endorser		Ledger
Committing Peer		Application
Ordering Node		
Smart Contract (Chaincode)		Endorsement Policy

Private Data Collections – Marble Scenario



Privacy Requirements:

- No marble data should go through ordering service as part of a transaction
- All peers have access to general marble information
 - *Name, Size, Color, Owner*
- Only a subset of peers have access to marble *pricing* information

Transaction

- Primary read/write set (if exists)
- Hashed private read/write set (hashed keys/values)

Transaction

- Public channel data
- Goes to all orderers/peers

Collection: Marbles

- Private Write Set
 - Name, Size, Color, Owner
- Policy: Org1, Org2**
- ```
"requiredPeerCount": 1,
"maxPeerCount":2,
"blockToLive":1000000
```

### Collection: Marbles

- Private data for channel peers
- Goes to all peers but not orderers

### Collection: Marble Private Details

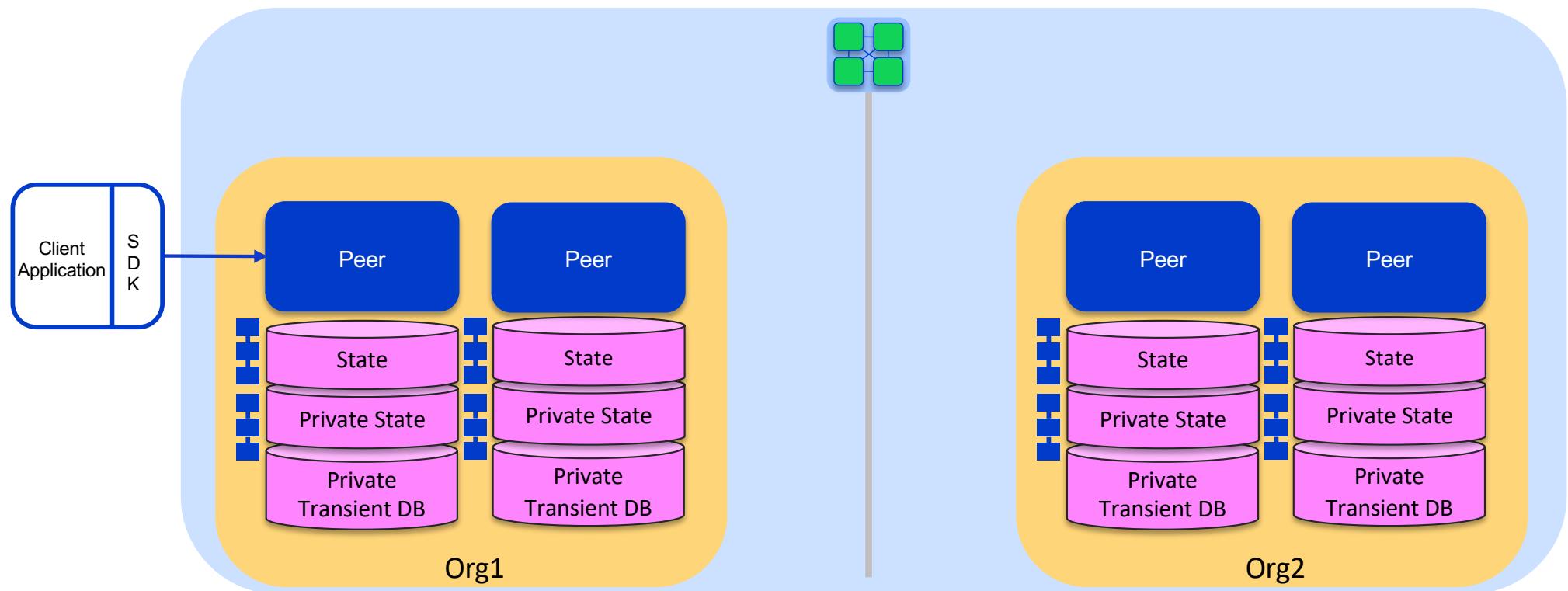
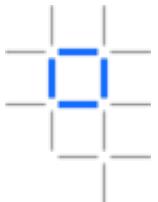
- Private Write Set
  - Price
- Policy: Org1**
- ```
"requiredPeerCount": 1,  
"maxPeerCount": 1,  
"blockToLive":3
```

Collection: Marbles Private Details

- Private data for subset of channel peers
- Goes to subset of peers only

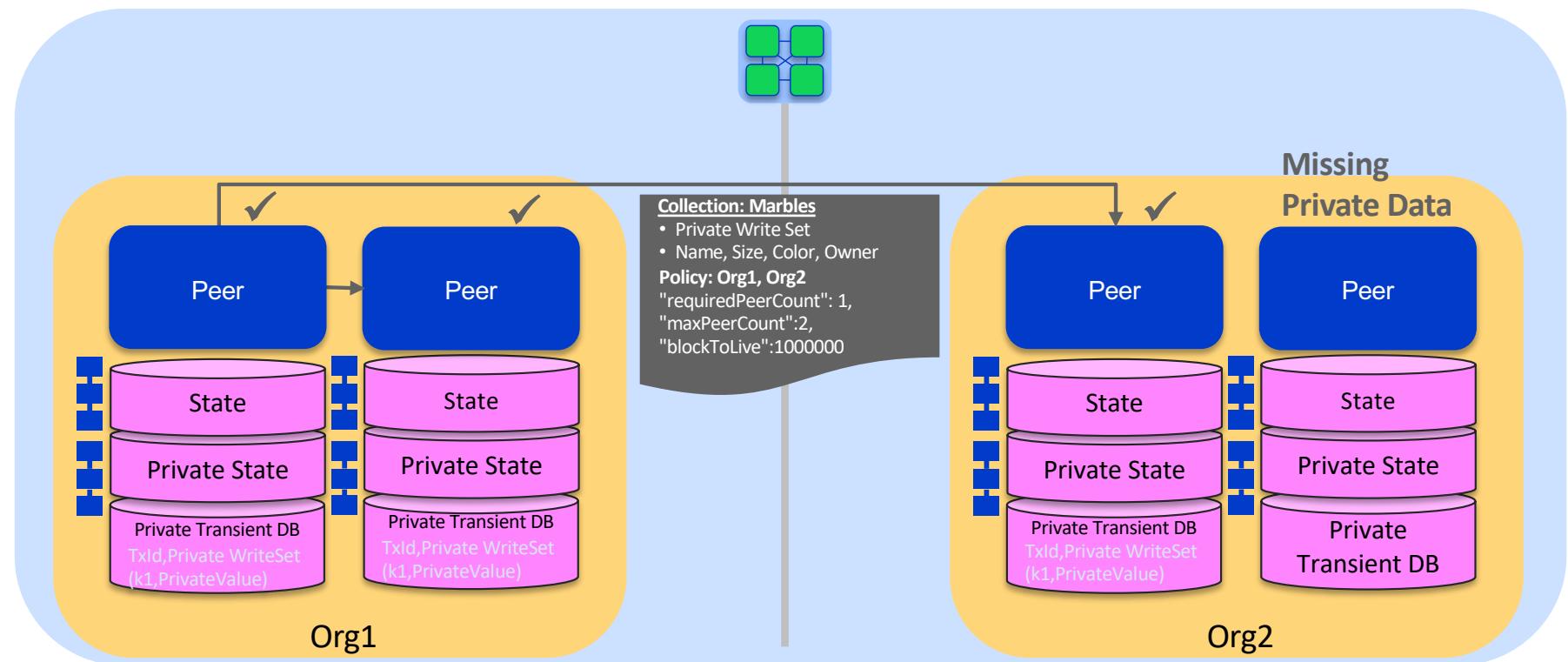
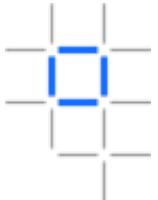
Step 1: Propose Transaction

Client sends proposal to endorsing peer(s)



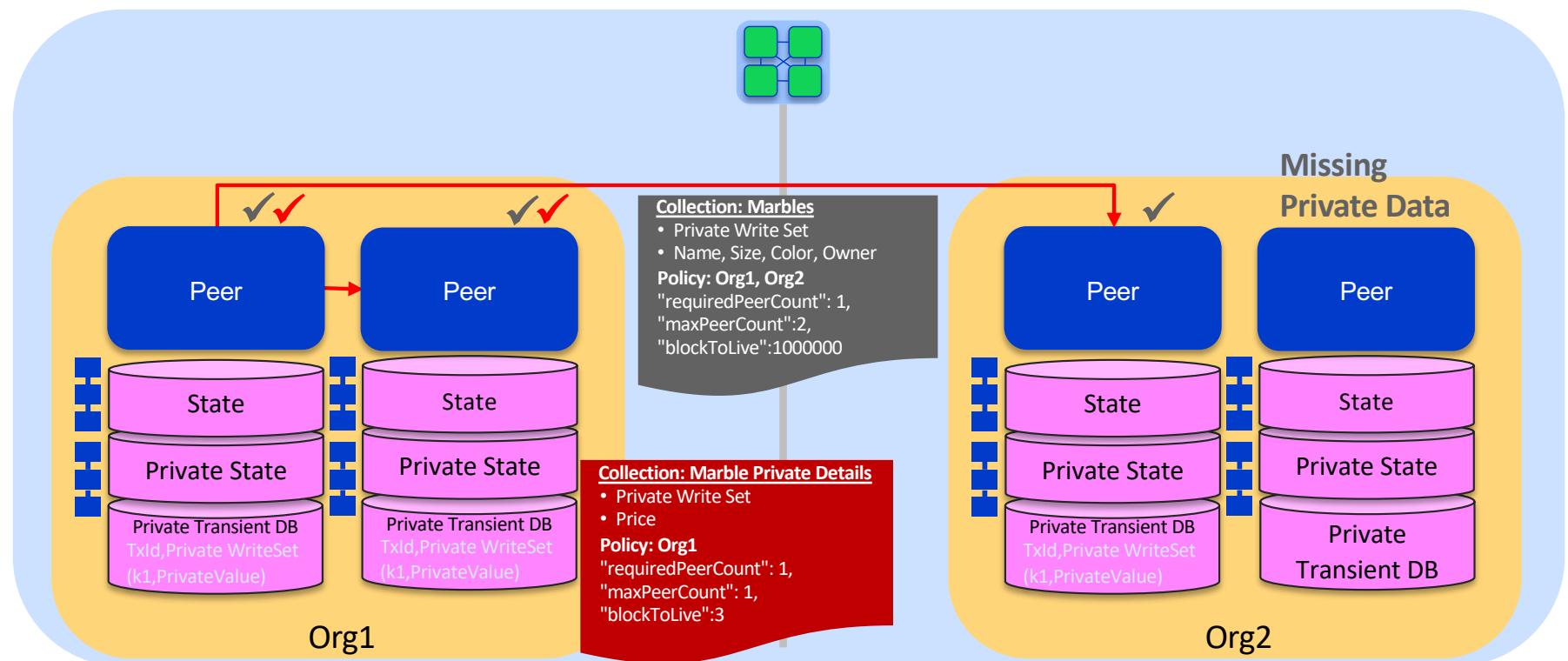
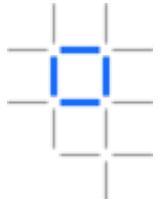
Step 2a: Execute Proposal and Distribute 1st Collection

Endorsing peer simulates transaction and distributes marbles collection data based on policy

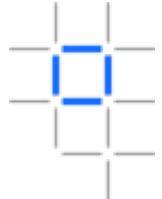


Step 2b: Distribute 2nd Collection

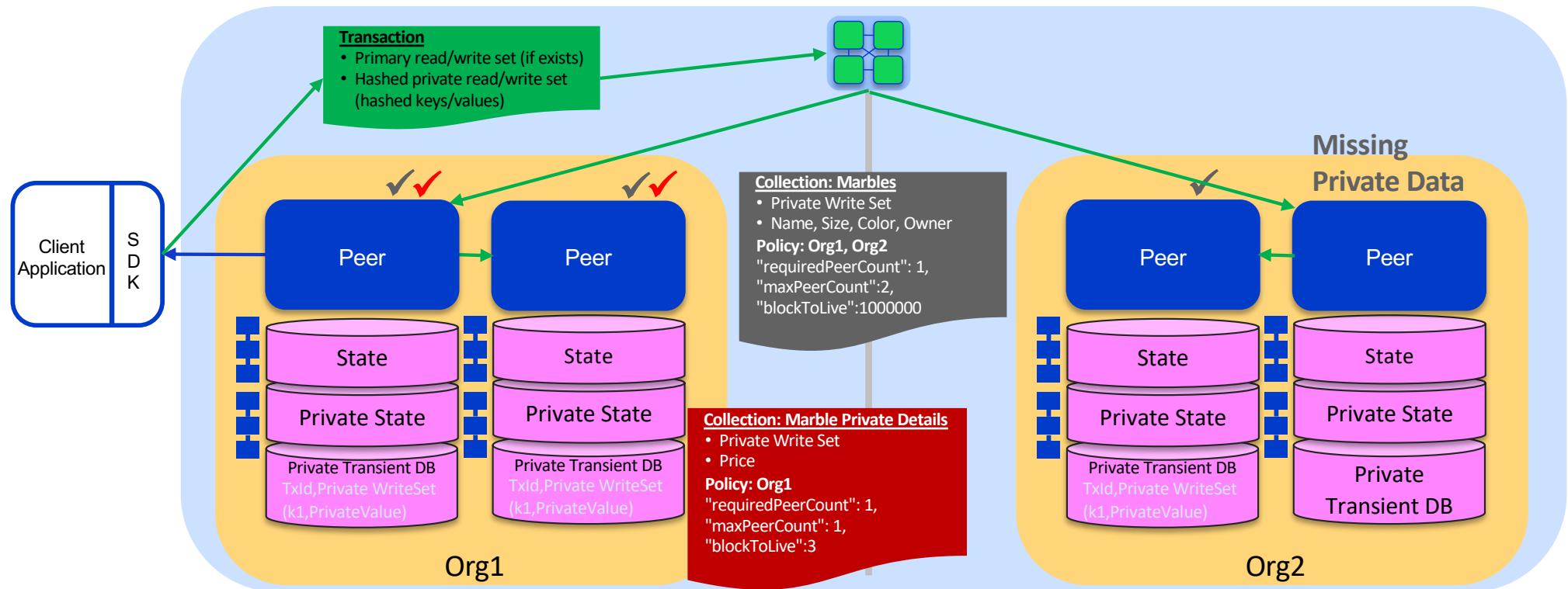
Endorsing peer distributes **marbles private details collection** data based on policy



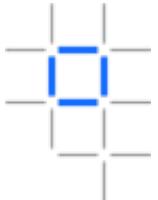
Step 3: Proposal Response / Order / Deliver



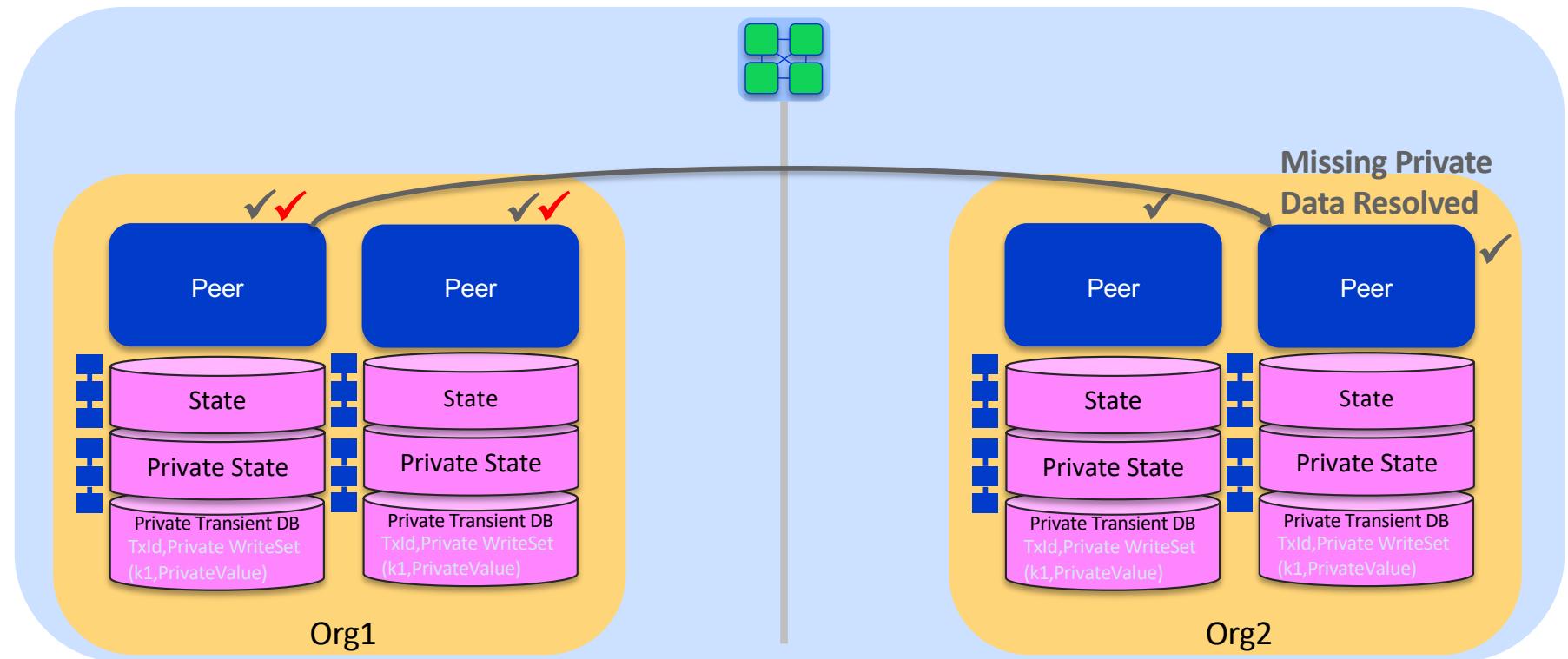
Proposal response sent back to client, which then sends the proposal to the ordering service for delivery to all peers



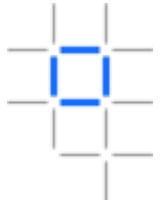
Step 4: Validate Transaction



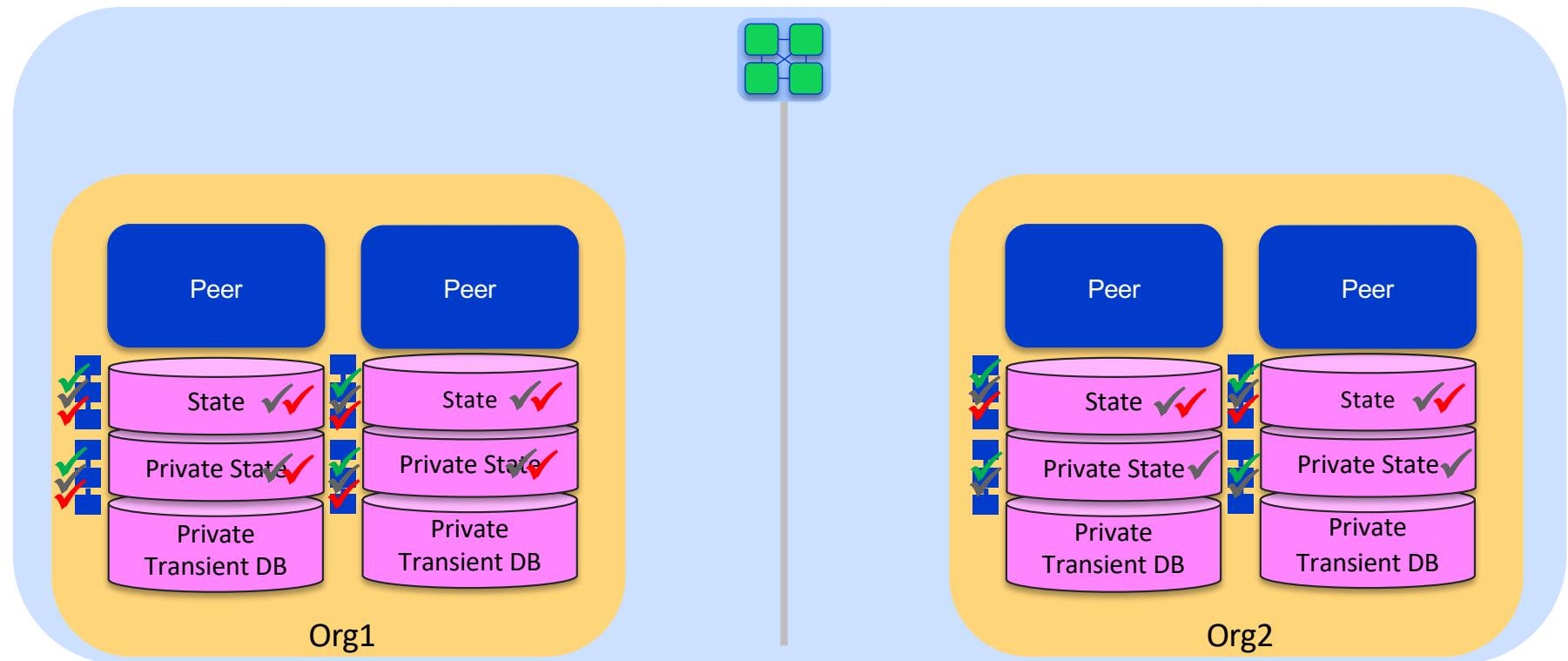
Peers validate transactions. Private data validated against hashes. Missing private data resolved with pull requests from other peers.



Step 5: Commit



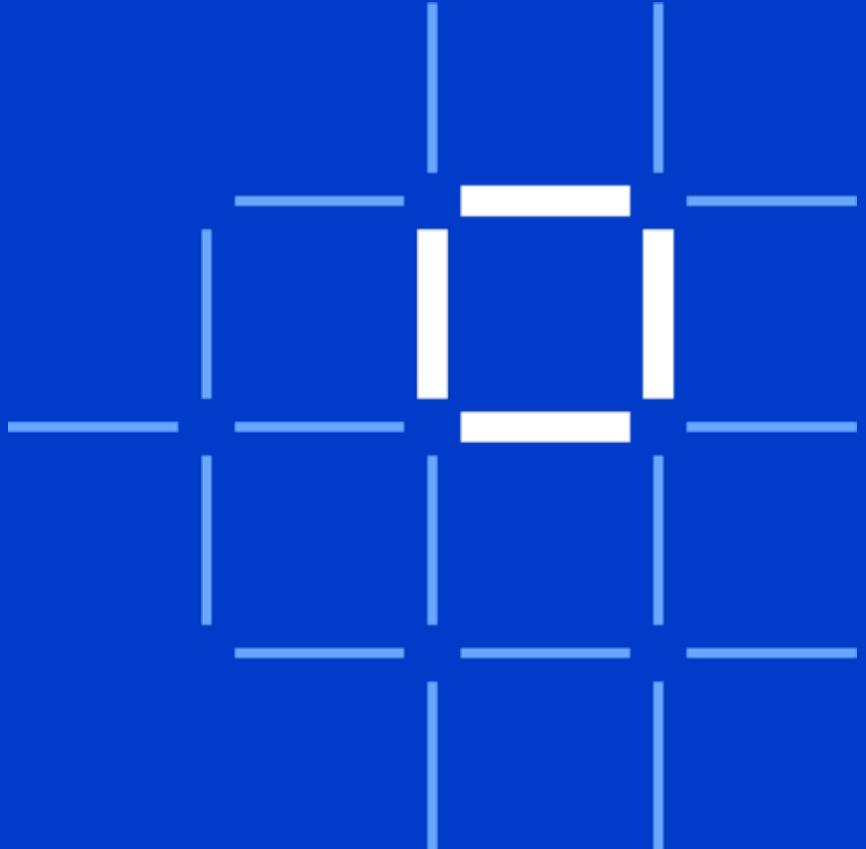
- 1) Commit private data to private state db. 2) Commit hashes to public state db.
- 3) Commit public block and private write set storage. 4) Delete transient data.





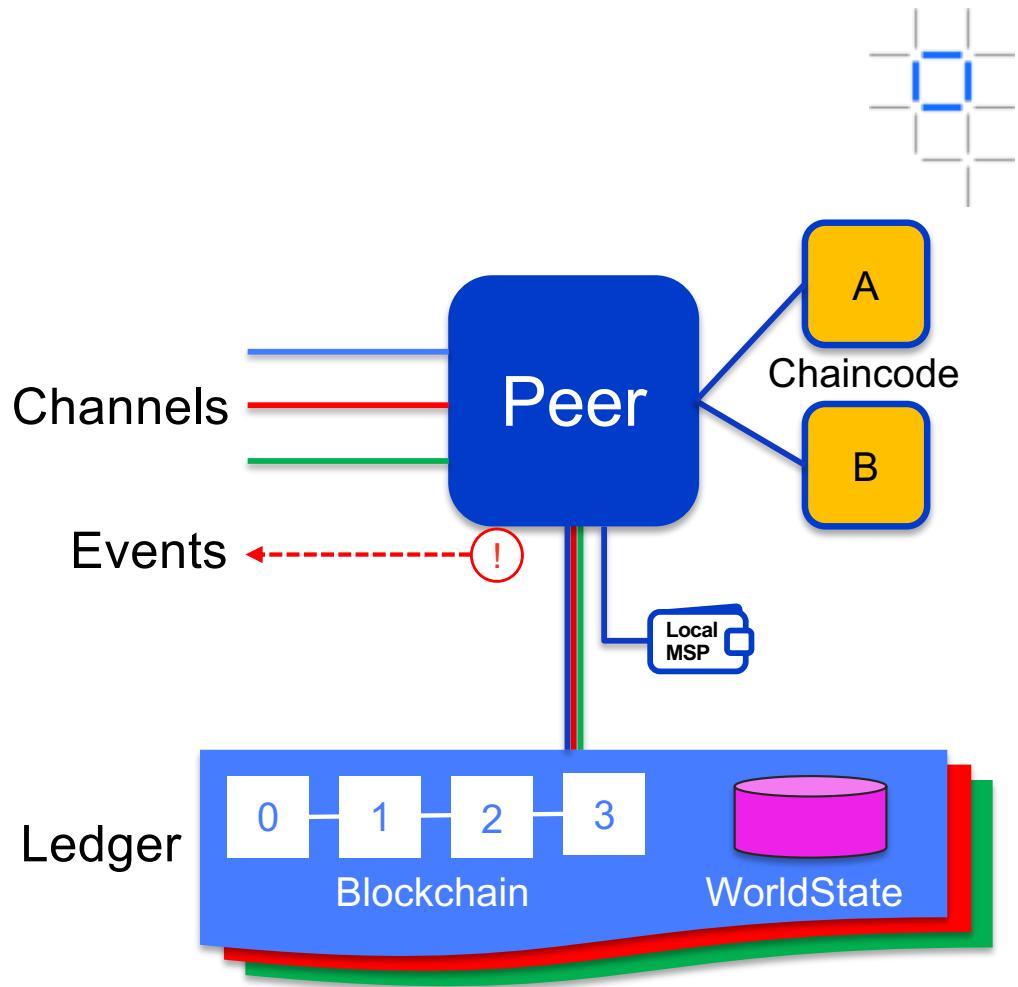
Hyperledger Fabric Technical Dive

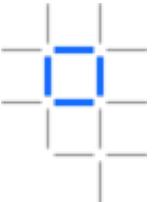
- Architectural Overview
- Network Consensus
- Channels and Ordering Service
- [Components]
- Endorsement Policies
- Membership Services



Fabric Peer

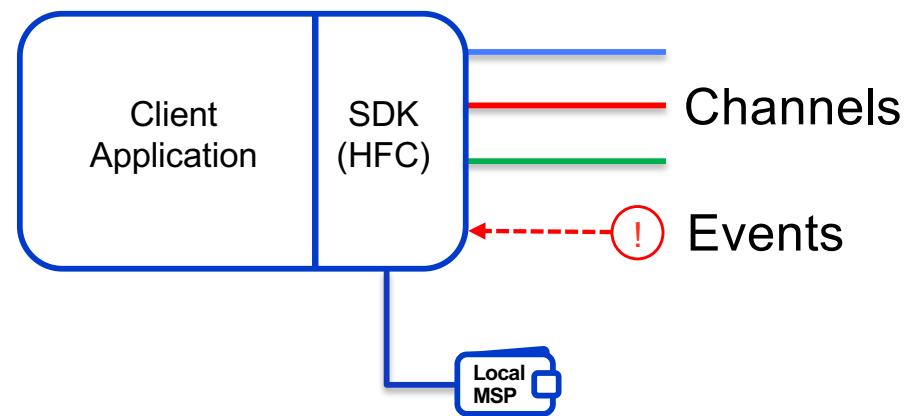
- Each peer:
 - Connects to one or more **channels**
 - Maintains one **ledger** per channel (plus possibly additional ledgers to hold private data)
 - Maintains **installed chaincode**
 - Manages **runtime docker containers** for **instantiated chaincode**
 - Chaincode is instantiated on a channel
 - Runtime docker container shared by channels with same chaincode instantiated (no state stored in container)
 - Has a local MSP (Membership Services Provider) that provides **crypto material**
 - **Emits events** to the client application

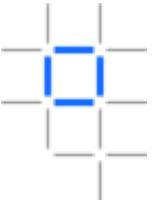




Client Application

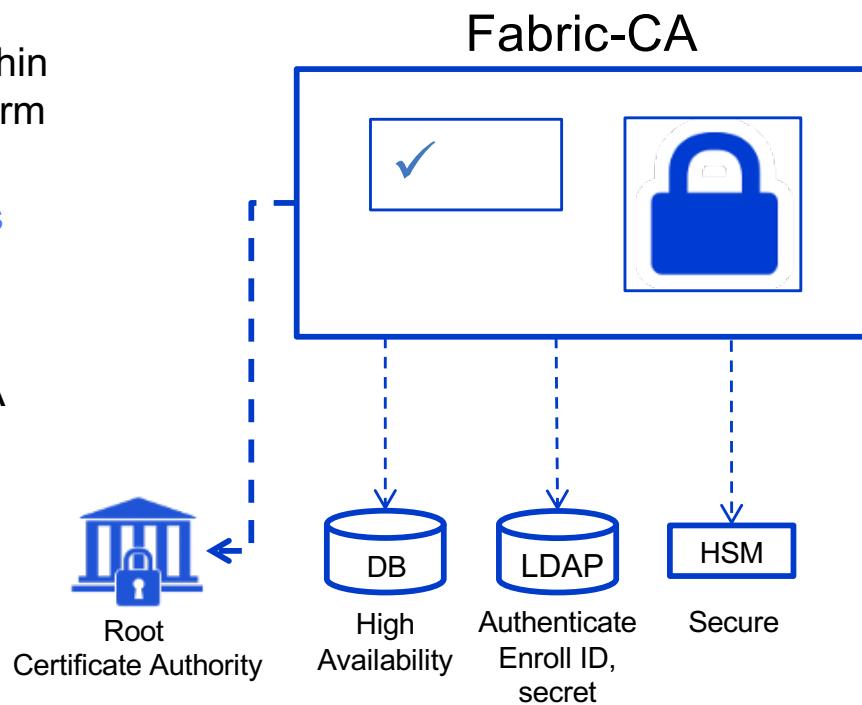
- Each client application uses Fabric SDK to:
 - Connects over channels to one or more peers
 - Connects over channels to one or more orderer nodes
 - Receives events from peers
 - Local MSP provides client [crypto material](#)
- Client can be written in different languages (Node.js, Go, Java, Python?)





Fabric-CA

- Default (optional) Certificate Authority within Fabric network for issuing [Ecerts](#) (long-term identity)
- Supports clustering for [HA characteristics](#)
- Supports LDAP for [user authentication](#)
- Supports HSM for [security](#)
- Can be configured as an intermediate CA

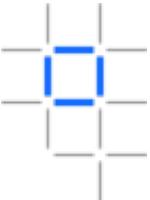




Hyperledger Fabric Technical Dive

- Architectural Overview
- Network Consensus
- Channels and Ordering Service
- Components
- [Endorsement Policies]
- Membership Services

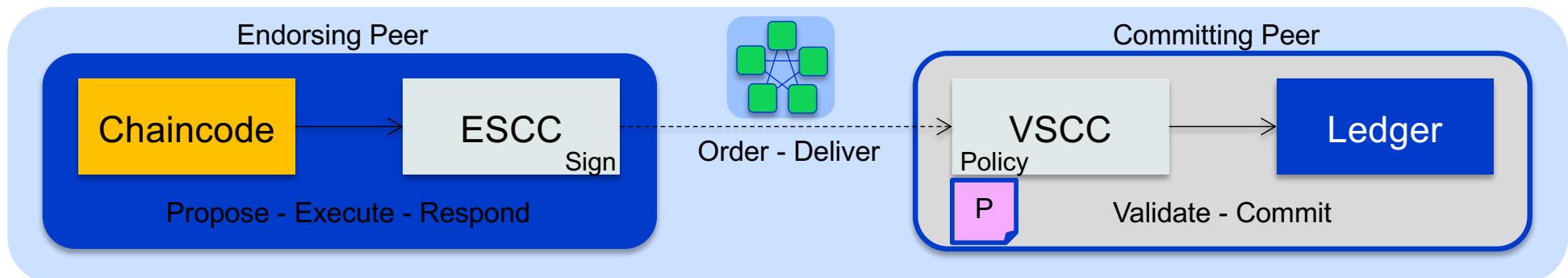


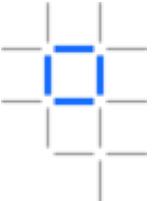


Endorsement Policies

An endorsement policy describes the conditions by which a transaction can be endorsed. A transaction can only be considered valid if it has been endorsed according to its policy.

- Each chaincode is deployed with an Endorsement Policy
- **ESCC** (Endorsement System ChainCode) signs the proposal response on the endorsing peer
- **VSCC** (Validation System ChainCode) validates the endorsements





Endorsement Policy Syntax

```
$ peer chaincode instantiate  
-C mychannel  
-n mycc  
-v 1.0  
-p chaincode_example02  
-c '{"Args":["init","a", "100", "b", "200"] }'  
-P "AND('Org1MSP.member')"
```

Instantiate the chaincode `mycc` on channel `mychannel` with the policy `AND('Org1MSP.member')`

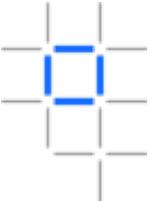
Policy Syntax: `EXPR(E[, E...])`

Where `EXPR` is either AND or OR and `E` is either a principal or nested EXPR

Principal Syntax: `MSP.ROLE`

Supported roles are: member and admin

Where `MSP` is the MSP ID, and `ROLE` is either “member” or “admin”



Endorsement Policy Examples

Examples of policies:

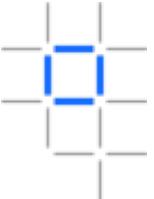
- Request 1 signature from all three principals
 - `AND('Org1.member', 'Org2.member', 'Org3.member')`
- Request 1 signature from either one of the two principals
 - `OR('Org1.member', 'Org2.member')`
- Request either one signature from a member of the Org1 MSP or (1 signature from a member of the Org2 MSP and 1 signature from a member of the Org3 MSP)
 - `OR('Org1.member', AND('Org2.member', 'Org3.member'))`



Hyperledger Fabric Technical Dive

- Architectural Overview
- Network Consensus
- Channels and Ordering Service
- Components
- Network setup
- Endorsement Policies
- [Membership Services]

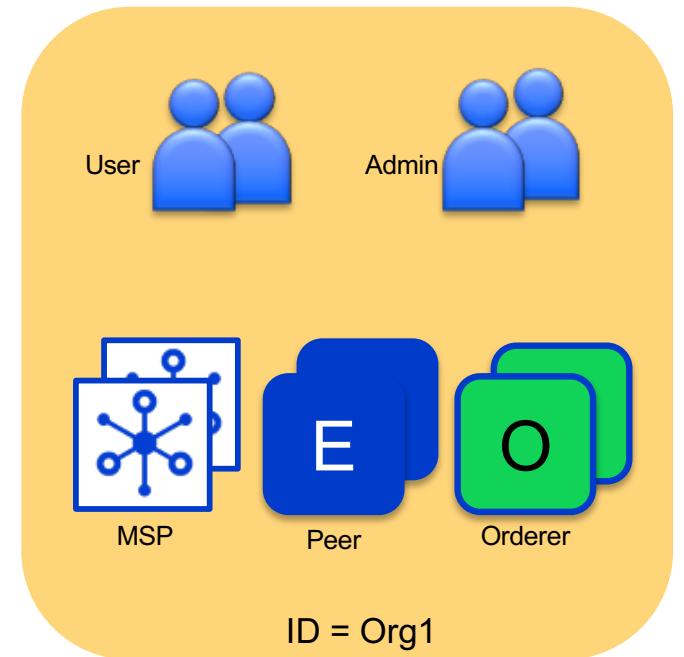


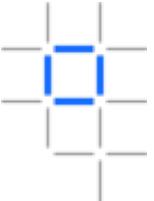


Organizations

Organizations define boundaries within a Fabric Blockchain Network

- Each Organization defines:
 - Membership Services Provider (MSP) for identities
 - Administrator(s)
 - Users
 - Peers
 - Orderers (optional)
- A network can include many Organizations representing a consortium
- Each Organization has an ID

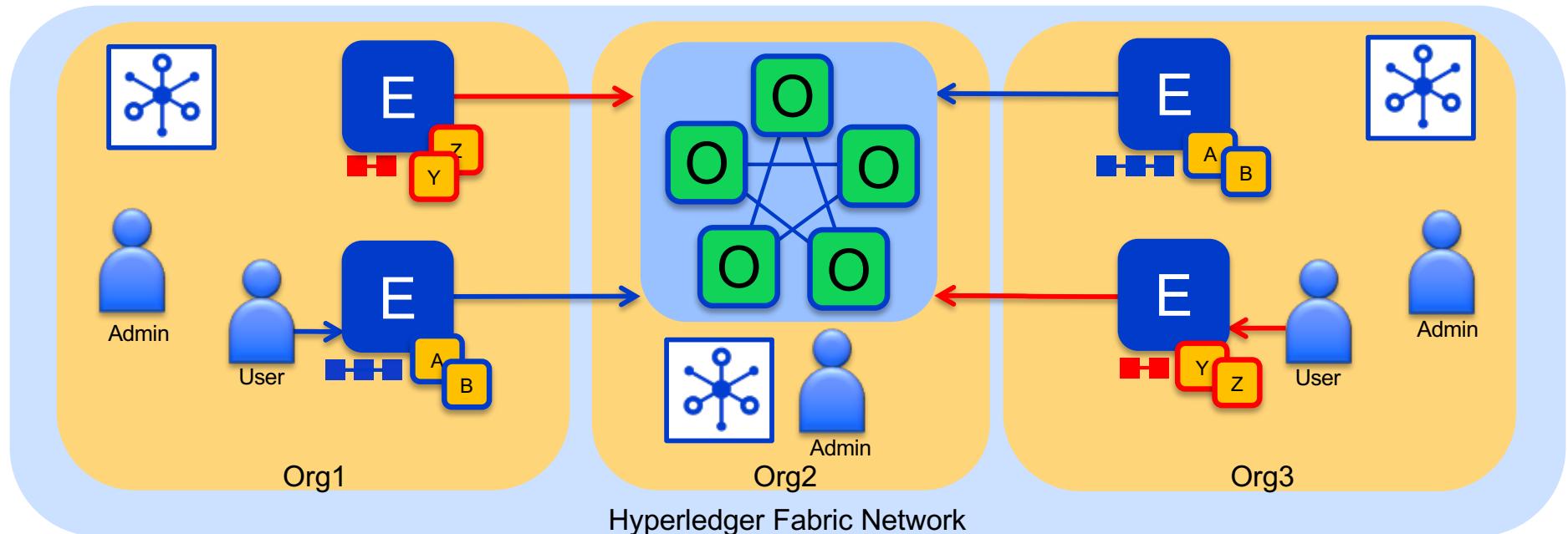




Consortium Network

An example consortium network of 3 organizations

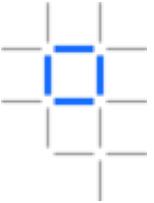
- Orgs 1 and 3 run peers
- Org 2 provides the ordering service only



* Ordering service may be provided by a single organization, or a decentralized group of organizations

IBM Blockchain

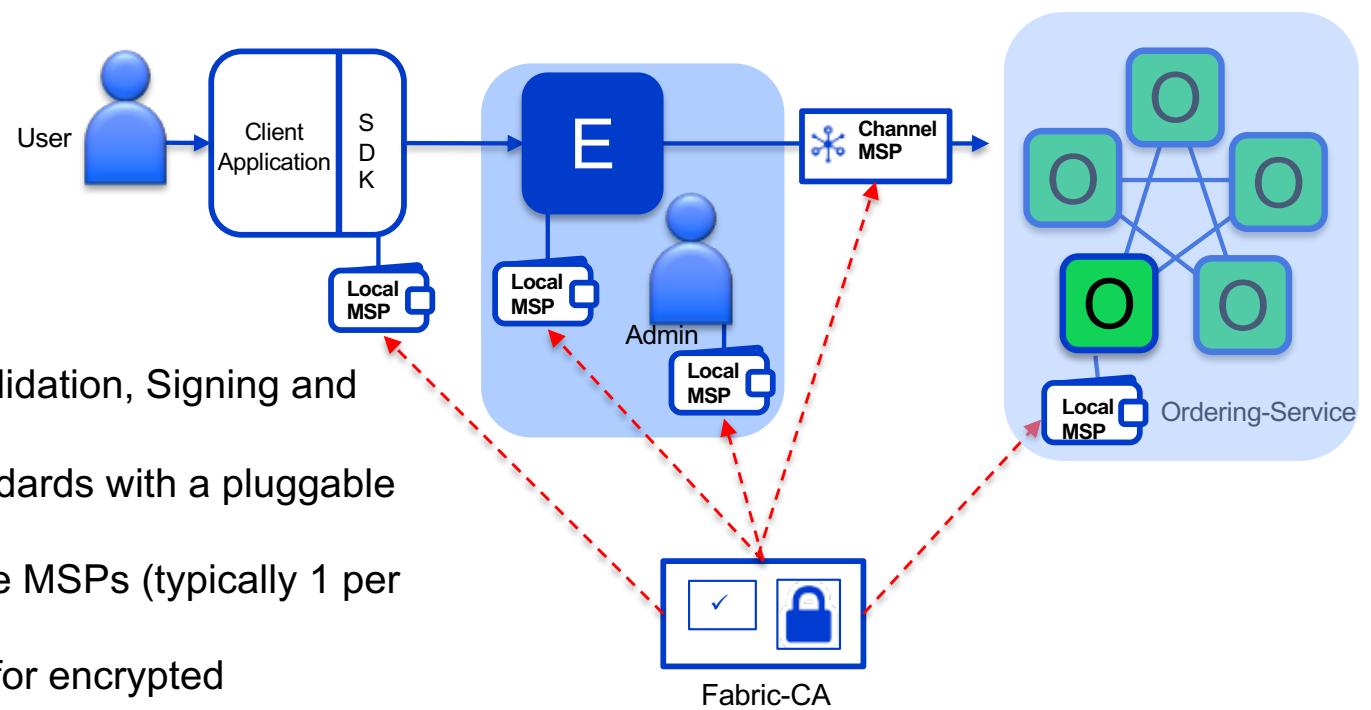
IBM

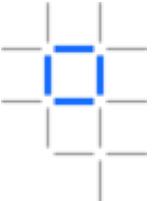


Membership Services Provider - Overview

A MSP manages a set of identities within a distributed Fabric network

- Provides identity for:
 - Peers and Orderers
 - Client Applications
 - Administrators
- Identities can be issued by:
 - Fabric-CA
 - An external CA
- Provides: Authentication, Validation, Signing and Issuance
- Supports different crypto standards with a pluggable interface
- A network can include multiple MSPs (typically 1 per org)
- Includes TLS crypto material for encrypted communications

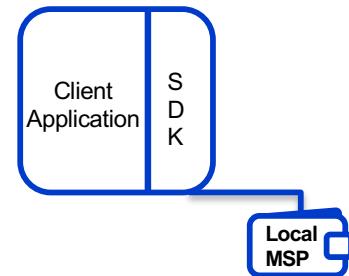




User Identities

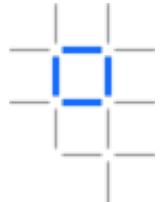
Each client application has a local MSP to store user identities

- Each local MSP includes:
 - **Keystore**
 - **Private key** for signing transactions
 - **Signcert**
 - **Public x.509 certificate**
- May also include TLS credentials
- Can be backed by a Hardware Security Module (HSM)



user@org1.example.com	
keystore	<private key>
signcert	user@org1.example.com-cert.pem

Admin Identities

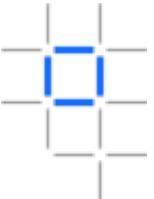


Each Administrator has a local MSP to store their identity

- Each local MSP includes:
 - **Keystore**
 - **Private key** for signing transactions
 - **Signcert**
 - **Public x.509 certificate**
- May also include TLS credentials
- Can be backed by a Hardware Security Module (HSM)



admin@org1.example.com	
keystore	<private key>
signcert	admin@org1.example.com-cert.pem



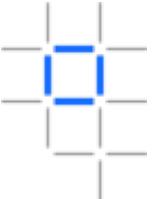
Peer and Orderer Identities

Each peer and orderer has a local MSP

- Each local MSP includes:
 - **keystore**
 - **Private key** for signing transactions
 - **signcert**
 - **Public x.509 certificate**
- In addition Peer/Orderer MSPs identify authorized administrators:
 - **admincerts**
 - List of **administrator certificates**
 - **cacerts**
 - The **CA public cert** for verification
 - **crls**
 - List of **revoked certificates**
- Peers and Orderers also receive channel MSP info
- Can be backed by a Hardware Security Module (HSM)



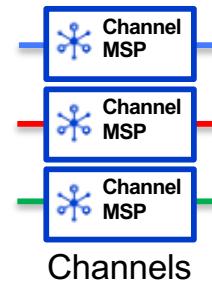
peer@org1.example.com	
admincerts	admin@org1.example.com-cert.pem
cacerts	ca.org1.example.com-cert.pem
keystore	<private key>
signcert	peer@org1.example.com-cert.pem
crls	<list of revoked admin certificates>



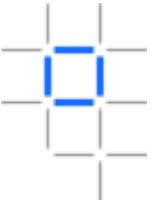
Channel MSP information

Channels include additional Organizational MSP information

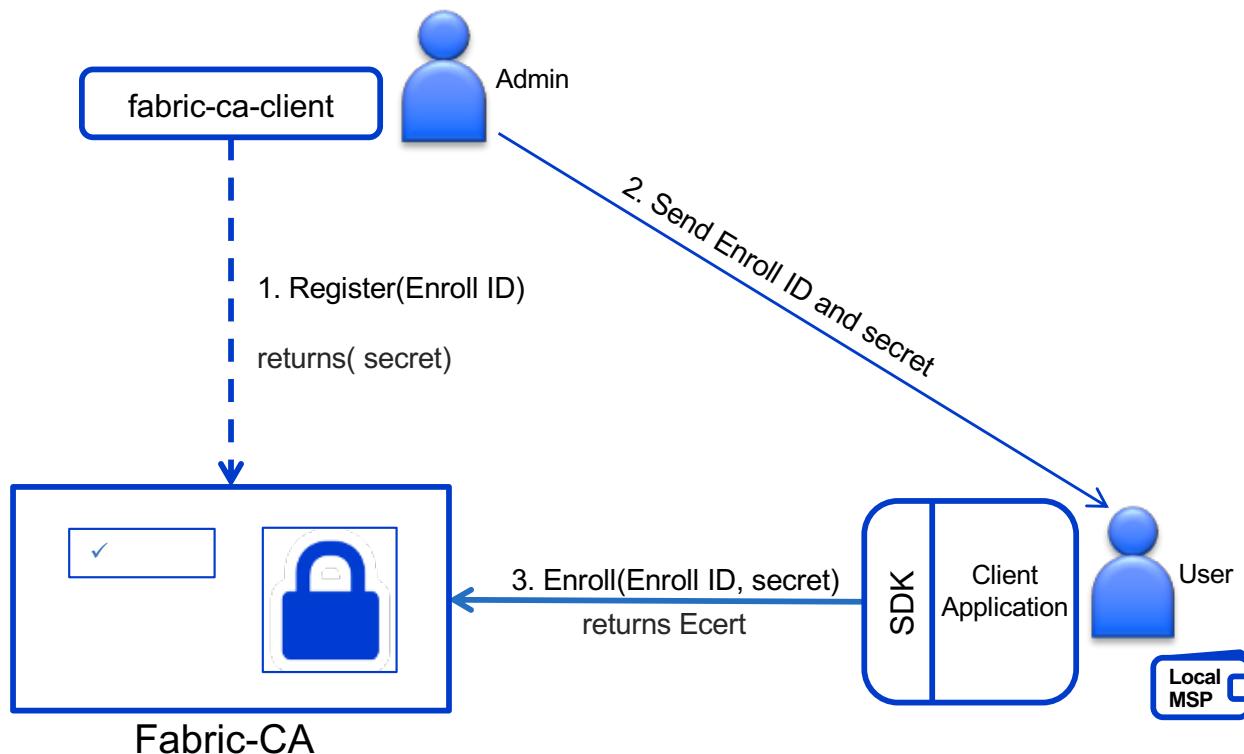
- Determines which orderers or peers can join the channel
- Determines client applications read or write access to the channel
- Stored in configuration blocks in the ledger
- Each channel MSP includes:
 - **admincerts**
 - Any public certificates for administrators
 - **cacerts**
 - The CA public certificate for this MSP
 - **crls**
 - List of revoked certificates
- **Does not include any private keys for identity**



ID = MSP1	
admincerts	admin.org1.example.com-cert.pem
cacerts	ca.org1.example.com-cert.pem
crls	<list of revoked admin certificates>



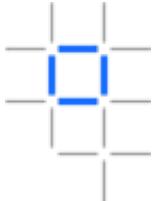
New User Registration and Enrollment



Registration and Enrollment

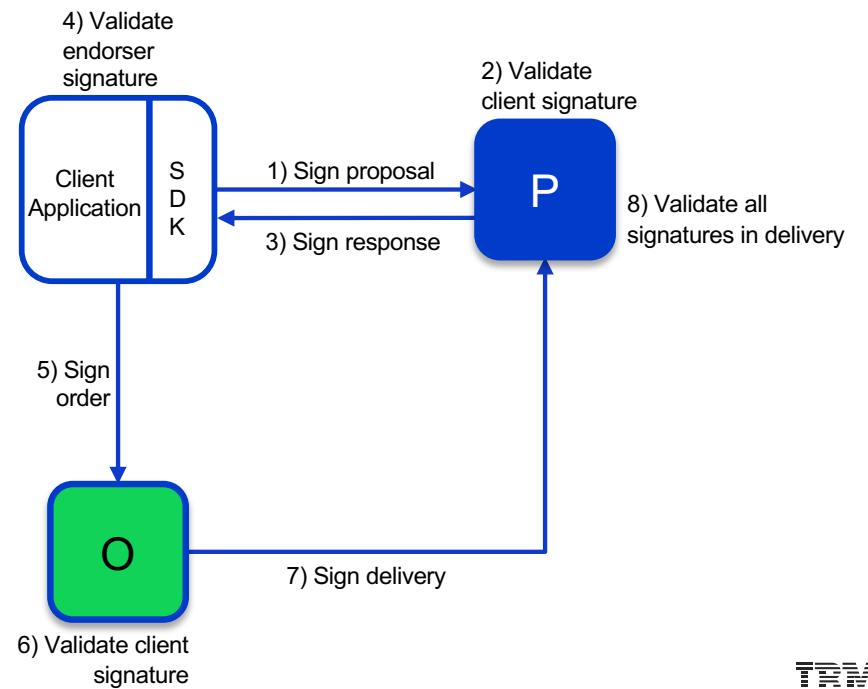
- Admin registers new user with `Enroll ID`
- User enrolls and receives credentials
- Additional offline registration and enrollment options available

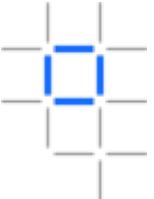
Transaction Signing



All transactions within a Hyperledger Fabric network are signed by permissioned actors, and those signatures validated

- Actors sign transactions with their enrolment private key
 - Stored in their local MSP
- Components validate transactions and certificates
 - Root CA certificates and CRLs stored in local MSP
 - Root CA certificates and CRLs stored in Org MSP in channel





Further Hyperledger Fabric Information

- Project Home: <https://www.hyperledger.org/projects/fabric>
- GitHub Repo: <https://github.com/hyperledger/fabric>
- Latest Docs: <https://hyperledger-fabric.readthedocs.io/en/latest/>
- Community Chat: <https://chat.hyperledger.org/channel/fabric>
- Project Wiki: <https://wiki.hyperledger.org/projects/fabric>
- Design Docs: <https://wiki.hyperledger.org/community/fabric-design-docs>

Thank you

Jeff Tennenbaum

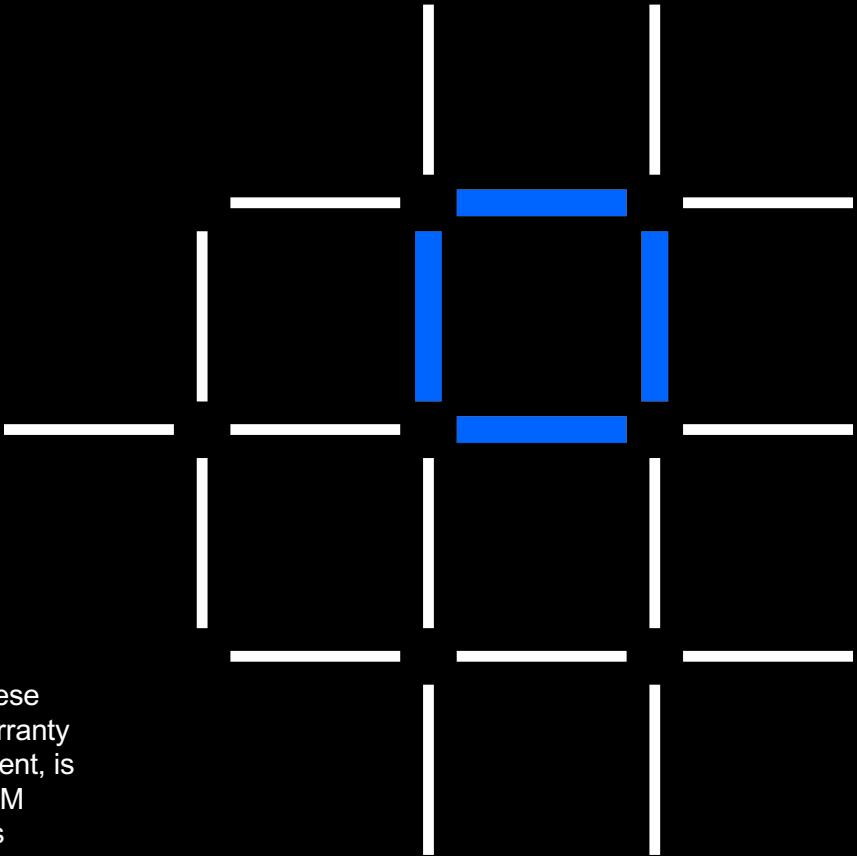
Tennenjl@us.ibm.com

IBM Blockchain

www.ibm.com/blockchain

developer.ibm.com/blockchain

www.hyperledger.org

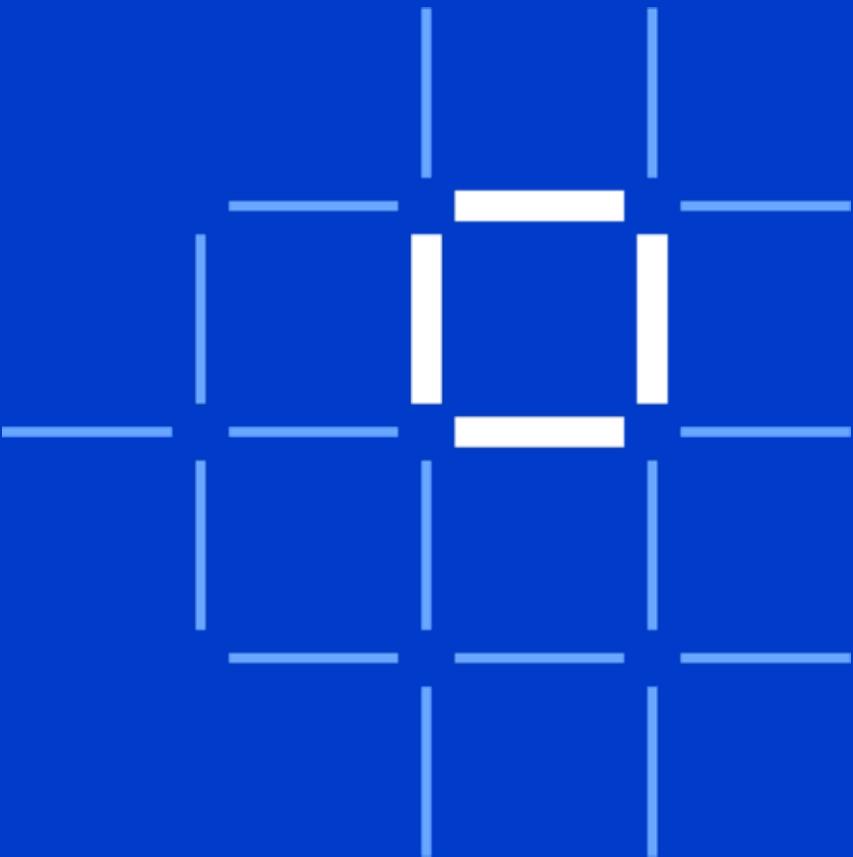


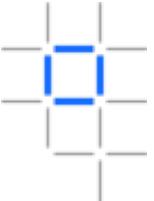
© Copyright IBM Corporation 2019. All rights reserved. The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. Any statement of direction represents IBM's current intent, is subject to change or withdrawal, and represents only goals and objectives. IBM, the IBM logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.

IBM

...

Appendix A: Hyperledger Fabric Commands

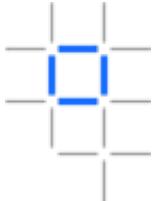




Fabric Commands

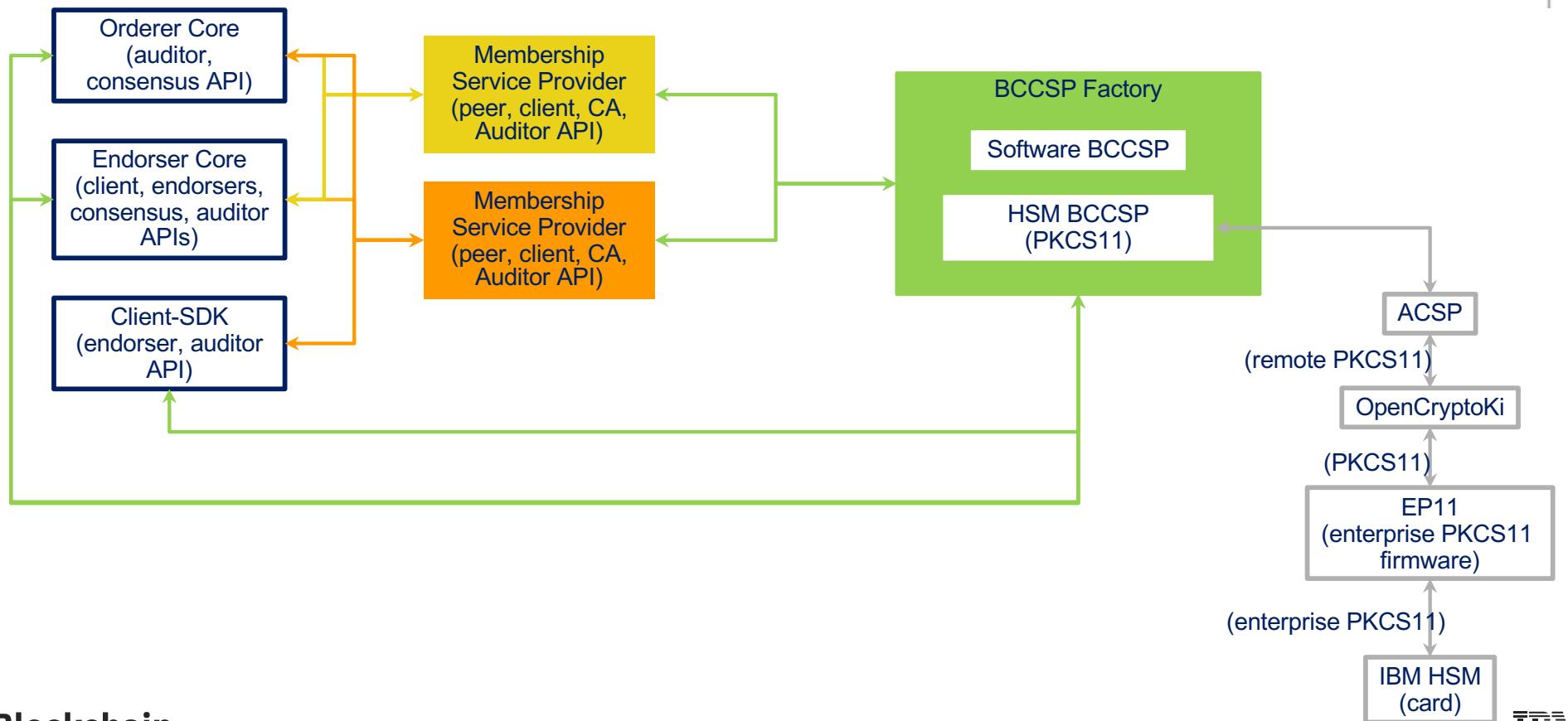
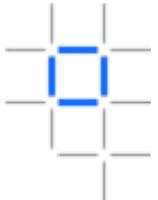
- Fabric has the following commands:
 - **peer** ... (For operating and configuring a peer)
 - **peer chaincode** ... (Manages chaincode on the peer)
 - **peer channel** ... (Manages channels on the peer)
 - **peer node** ... (Manages the peer)
 - **peer version** (Returns the peer version)
 - **cryptogen** ... (Utility for generating crypto material)
 - **configtxgen** ... (Creates configuration data such as the genesis block)
 - **configtxlator** ... (Utility for generating channel configurations)
 - **fabric-ca-client** ... (Manages identities)
 - **fabric-ca-server** ... (Manages the Fabric-CA server)

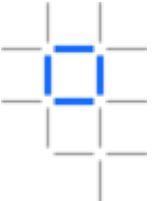
Configuration Detail



Path	MSP ID	Attributes	Attributes
config -> channel_group -> groups -> application -> groups	Org1MSP	mod_policy	Admins
		policies -> Admins	mod_policy
			Admins
		policy -> value -> identities	Org1MSP, Admin
		Policy -> value -> rule	n_out_of
	Policies -> Readers	Mod_policy	Admins
		Policy -> value -> identities	Org1MSP
		Policy -> value -> rule	N_out_of
	Policies -> Writers	Mod_policy	Admins
		Policy -> value -> identities	Org1MSP
		Policy -> value -> rule	N_out_of

MSP and BCCSP (Modularity and Decentralisation)



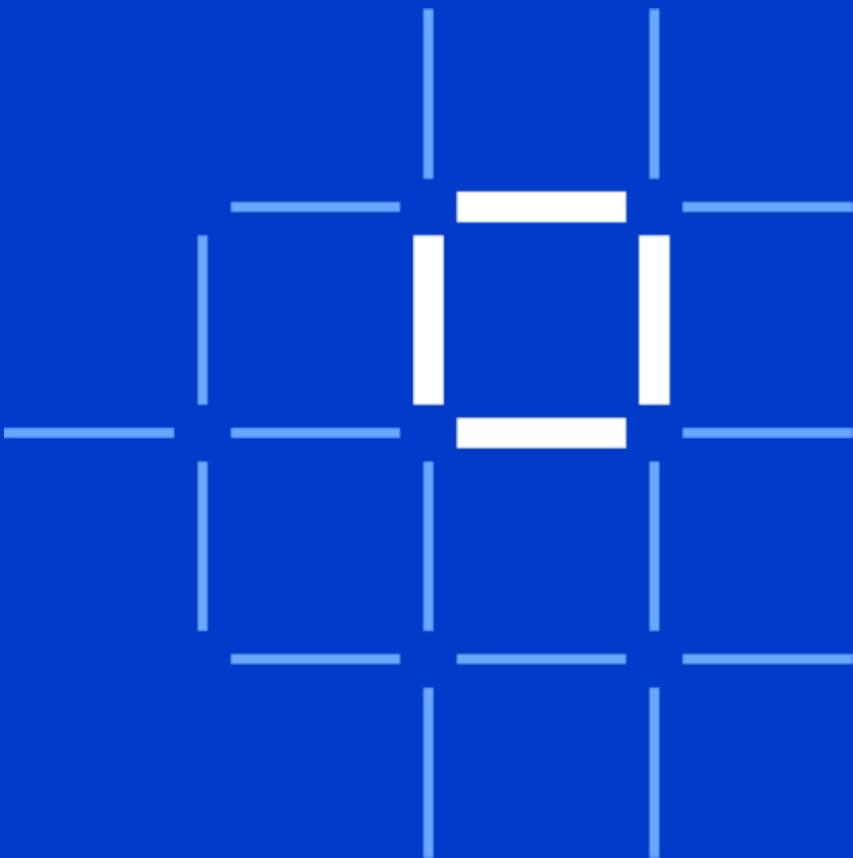


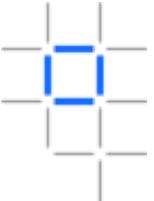
Blockchain Crypto Service Provider (BCCSP)

- Pluggable implementation of cryptographic standards and algorithms.
- **Pluggability**
 - alternate implementations of crypto interface can be used within the Hyperledger 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
- **International Standards Support**
 - E.g., via a new/separate CSP
 - Interoperability among standards is not necessarily guaranteed

...

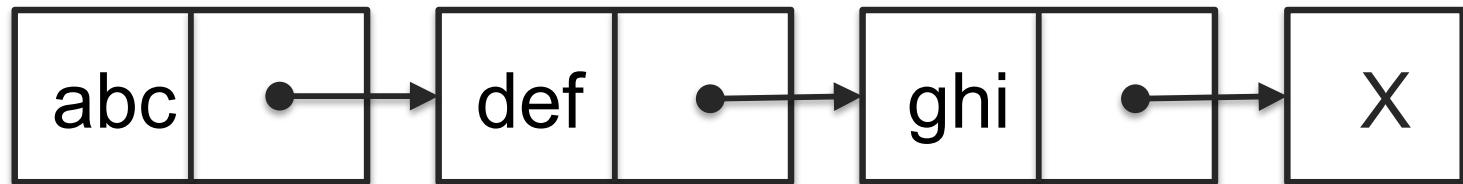
Appendix B: **Blockchain Data Structures Overview**



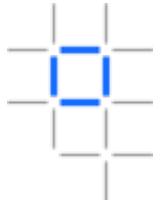


The Linked List

- Linear collection of data elements
- Each element is linked to the next
- Concept dates from 1955

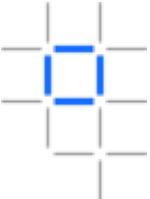


One-Way Hash Functions



- Any function that can be applied to a set of data that is guaranteed to produce the same output for the same input
- One-way means that you can't derive the input from the output
- Often, outputs are *unlikely* to repeat for different inputs
- Forms the basis of much cryptography

$$\begin{aligned} h(abc) &= 7859 \\ h(def) &= 8693 \\ h'(7859) &= ? \\ h(abc) &= 7859 \end{aligned}$$



The Hash Chain

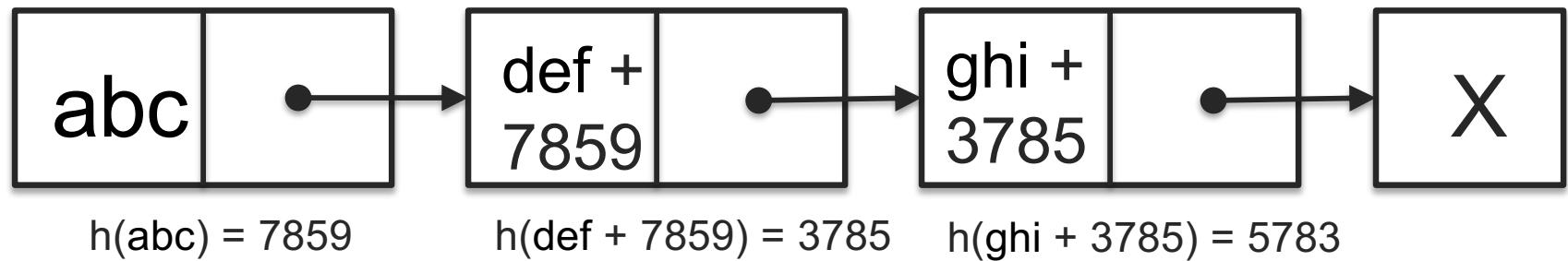
Hash chain: A successive application of a hash function

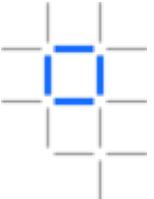
$$h(h(h(abc))) = 1859$$

Can combine new data with each successive hash:

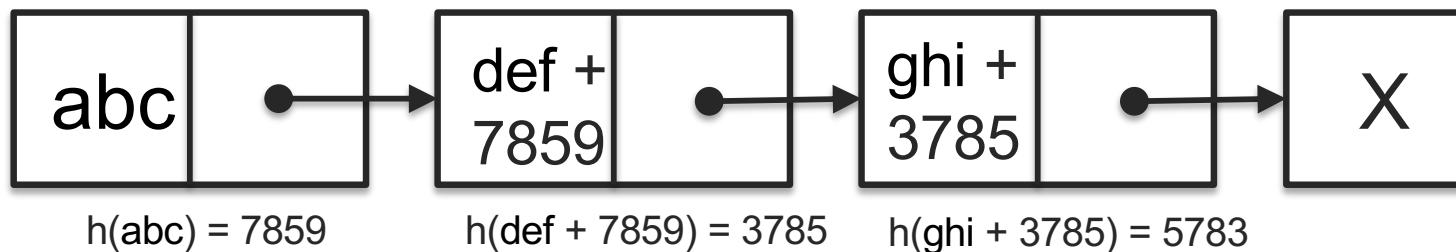
$$h(ghi + h(def + h(abc))) = 5783$$

Using this concept you can produce a tamper resistant linked list:

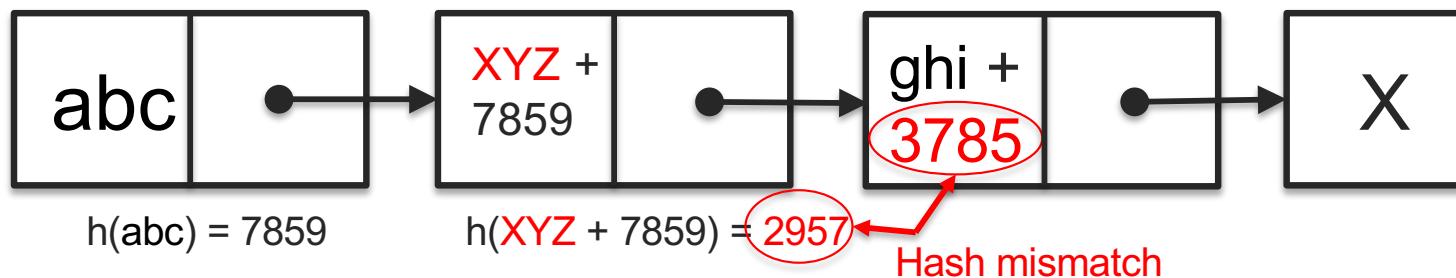


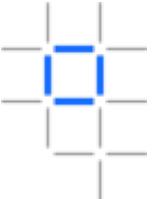


How is This Tamper Resistant?



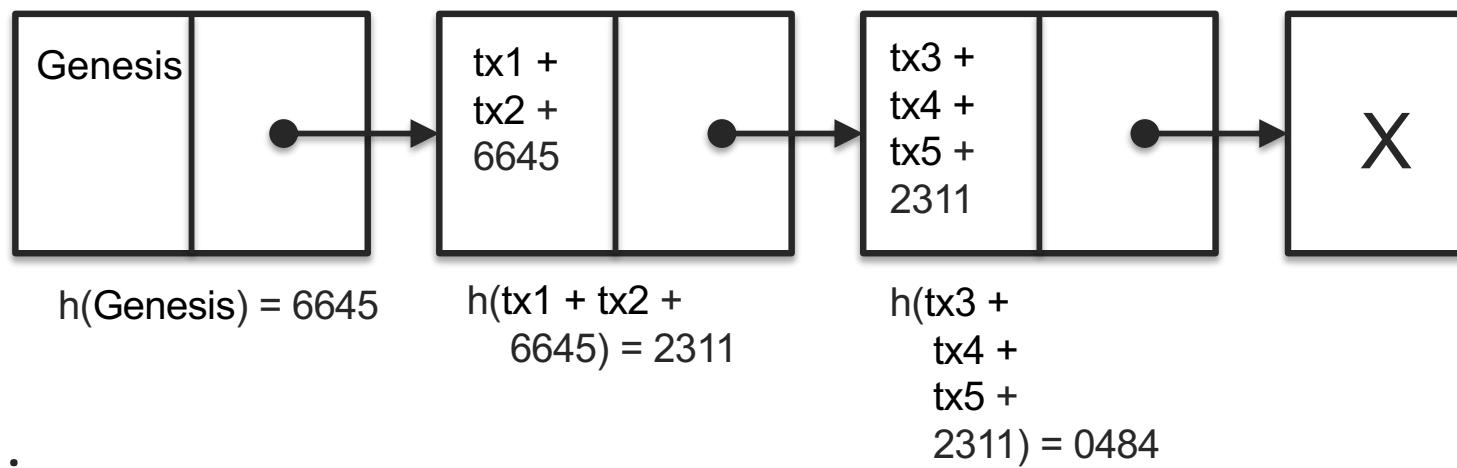
- Any modification to a data element means that the hashes will not match up
 - You would need to recreate the downstream chain



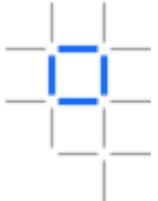


Applied to Blockchain

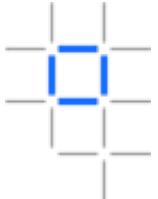
- A blockchain is a hash chain (*with optimizations* that we'll cover shortly)
- Each element (block) in the linked list is a set of zero or more transactions
 - Transactions are an implementation-dependent data object
- First block known as a genesis block
 - May contain some identifying string or other configuration metadata



Some Problems with This Approach

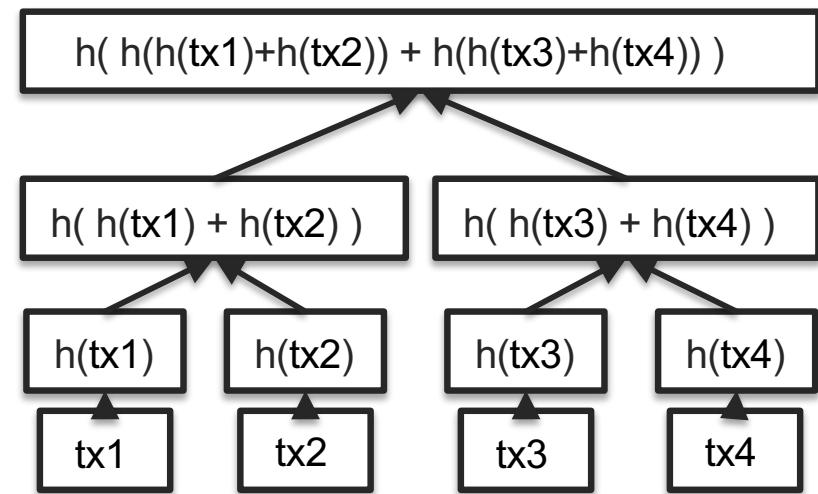


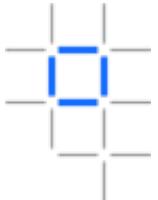
- In the event of tampering, it can be difficult to identify which transaction was modified (particularly when there are many transactions in a block)
 - It is not feasible to have one transaction per block
- It requires all transaction data in order to retain integrity of chain
- Searching transactions is linear (time consuming)



Merkle Tree

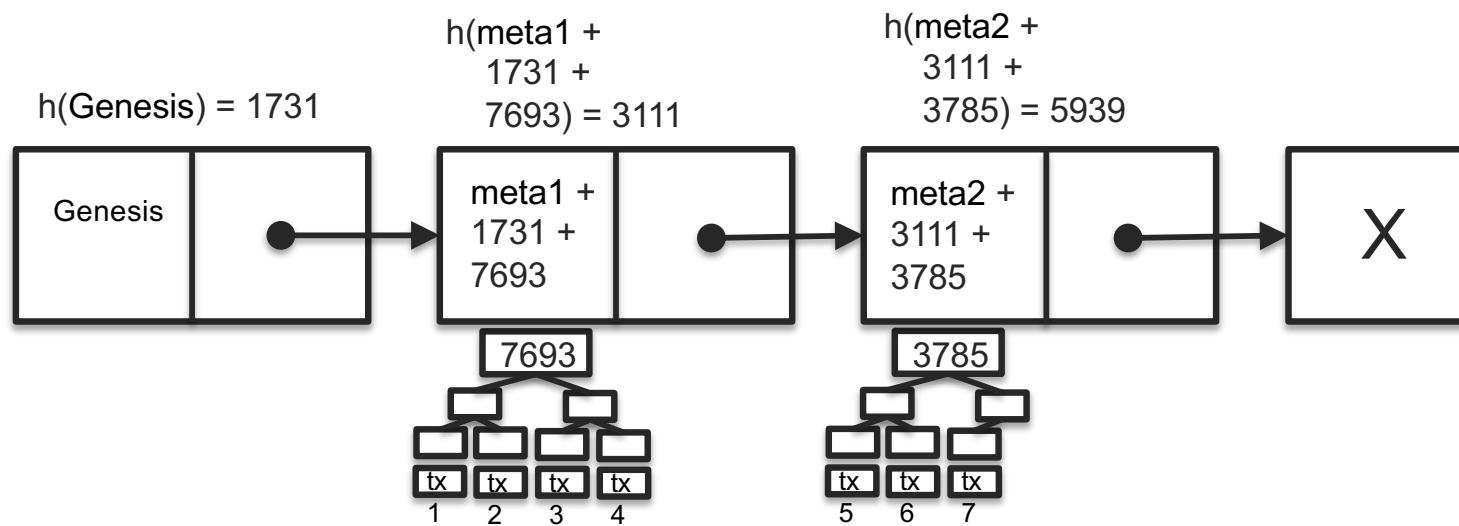
- It is possible to optimise the chain data structure if we arrange it as a tree
 - Makes it easier to identify tampering without sacrificing stability
 - Makes it quicker to traverse
- However, this makes it impossible to add new transactions without re-hashing root nodes



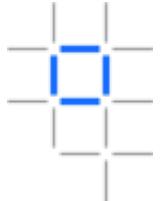


Combining Tree and Chain

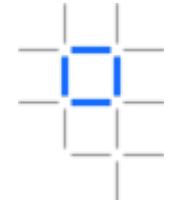
- Each element in the chain contains:
 - A pointer (“Merkle root”) to the tree of transactions
 - Other metadata (e.g. timestamp)
 - A hash of the previous block’s data (i.e. Merkle root, metadata and hash)



Benefits of This Approach



- It allows tampered transactions to be identified easily
- More [efficient search](#) within a block
 - $O(\log N)$ rather than $O(N)$
- Allows transaction detail to be stubbed
 - Bitcoin has a [Simplified Payment Verifier](#) (SPV) concept: a type of user that doesn't have the entire tree available, just the Merkel roots
 - Note it is also possible to checkpoint and archive old blocks, creating a new Genesis block mid-way through the chain

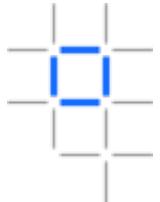


Common Transactions

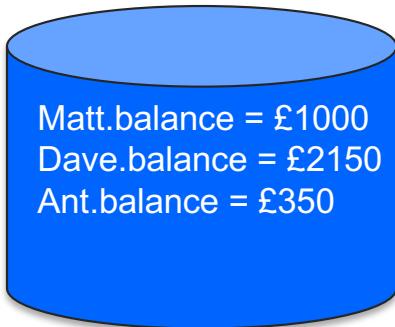
- What's Dave's balance?
- Does Matt have funds to clear a £1000 transaction? (Assuming no overdraft)

#	Transaction	Initiator	Receiver	Amount
1	Create a/c	Cash	Matt	£1000
2	Create a/c	Cash	Dave	£2000
3	Transfer	Matt	Dave	£100
4	Create a/c	Cash	Ant	£500
5	Transfer	Ant	Matt	£50
6	Transfer	Ant	Dave	£200
7	Transfer	Dave	Matt	£100
8	Transfer	Dave	Ant	£50
9	Transfer	Matt	Ant	£50

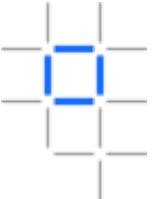
World State



- It is clearly not feasible to reparse the entire transaction log to complete a new transaction
- Blockchains often include an associated database (world state) – e.g. Hyperledger Fabric
- Transactions become a set of creates, reads, updates and deletes of records in this data store

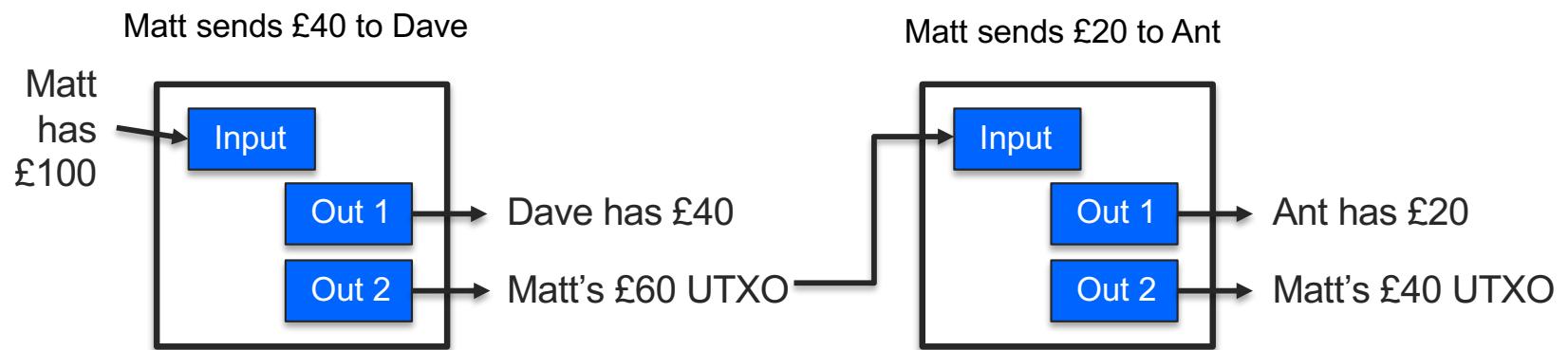


#	Transaction	Initiator	Receiver	Amount
1	Create a/c	Cash	Matt	£1000
2	Create a/c	Cash	Dave	£2000
3	Transfer	Matt	Dave	£100
4	Create a/c	Cash	Ant	£500
5	Transfer	Ant	Matt	£50
6	Transfer	Ant	Dave	£200
7	Transfer	Dave	Matt	£100
8	Transfer	Dave	Ant	£50
9	Transfer	Matt	Ant	£50



Unspent Transaction Outputs

- Some blockchains (e.g. Bitcoin) don't maintain balances
 - Transactions are linked to earlier transactions using an ID (TXID)
 - Outputs always equal inputs
 - Unspent funds are marked as an “[Unspent Transaction Output](#)” (UTXO)
 - Only UTXOs can be used as inputs (to prevent double spending)
 - Your “balance” is the aggregation of all of your UTXOs
- In Bitcoin, if your application doesn't specify the UTXO output then the miner gets the excess!



IBM