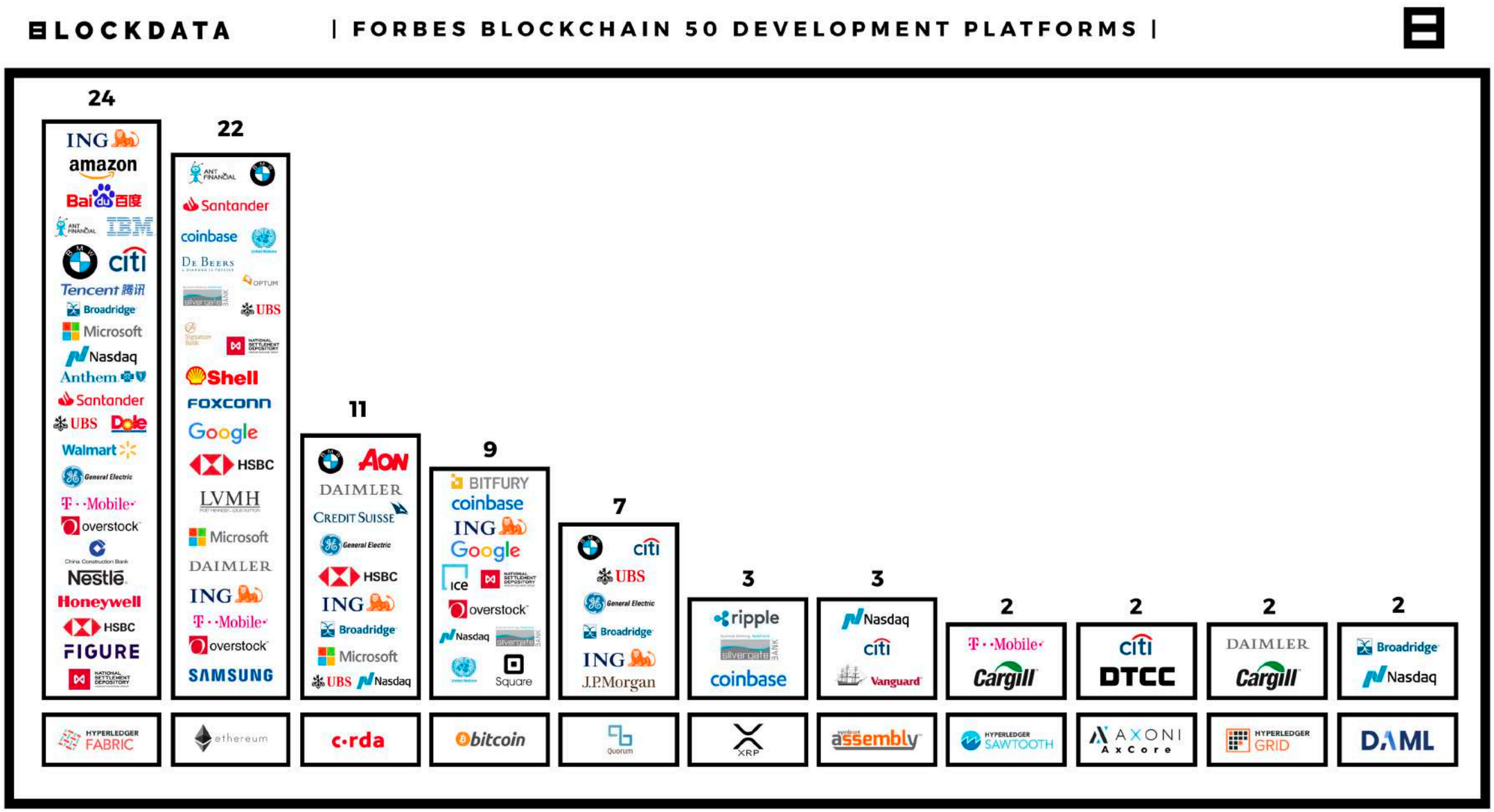
Hyperledger Fabric Reality Check

Lance Feagan

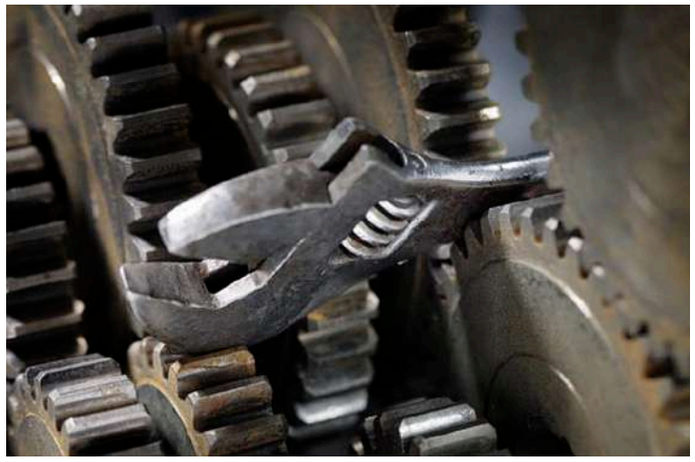
Why Fabric?



Presentation Goals

Presentation Goals

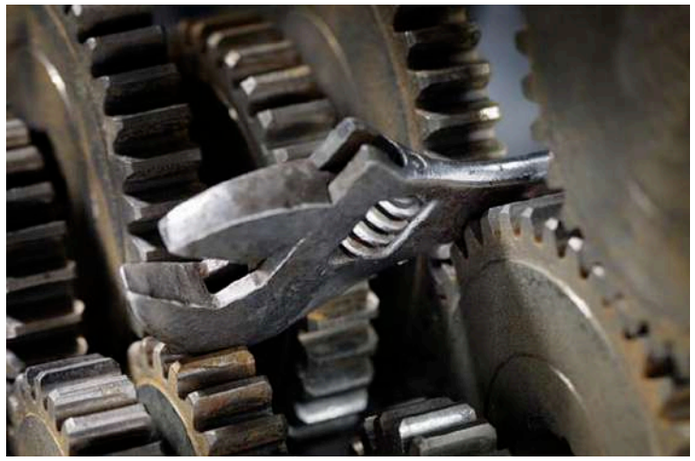
- Reveal areas where those unfamiliar with Fabric's inner-workings will run into trouble.

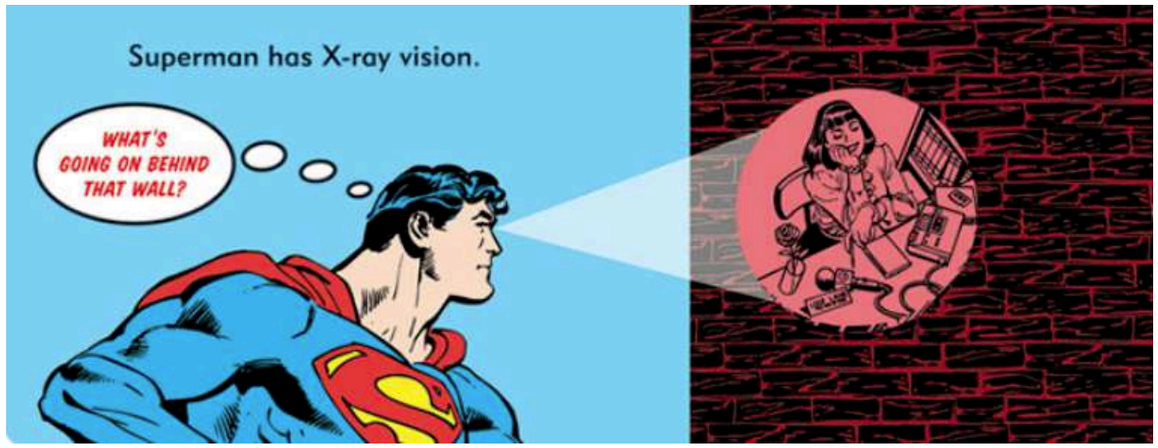


Presentation Goals

- Reveal areas where those unfamiliar with Fabric's inner-workings will run into trouble.

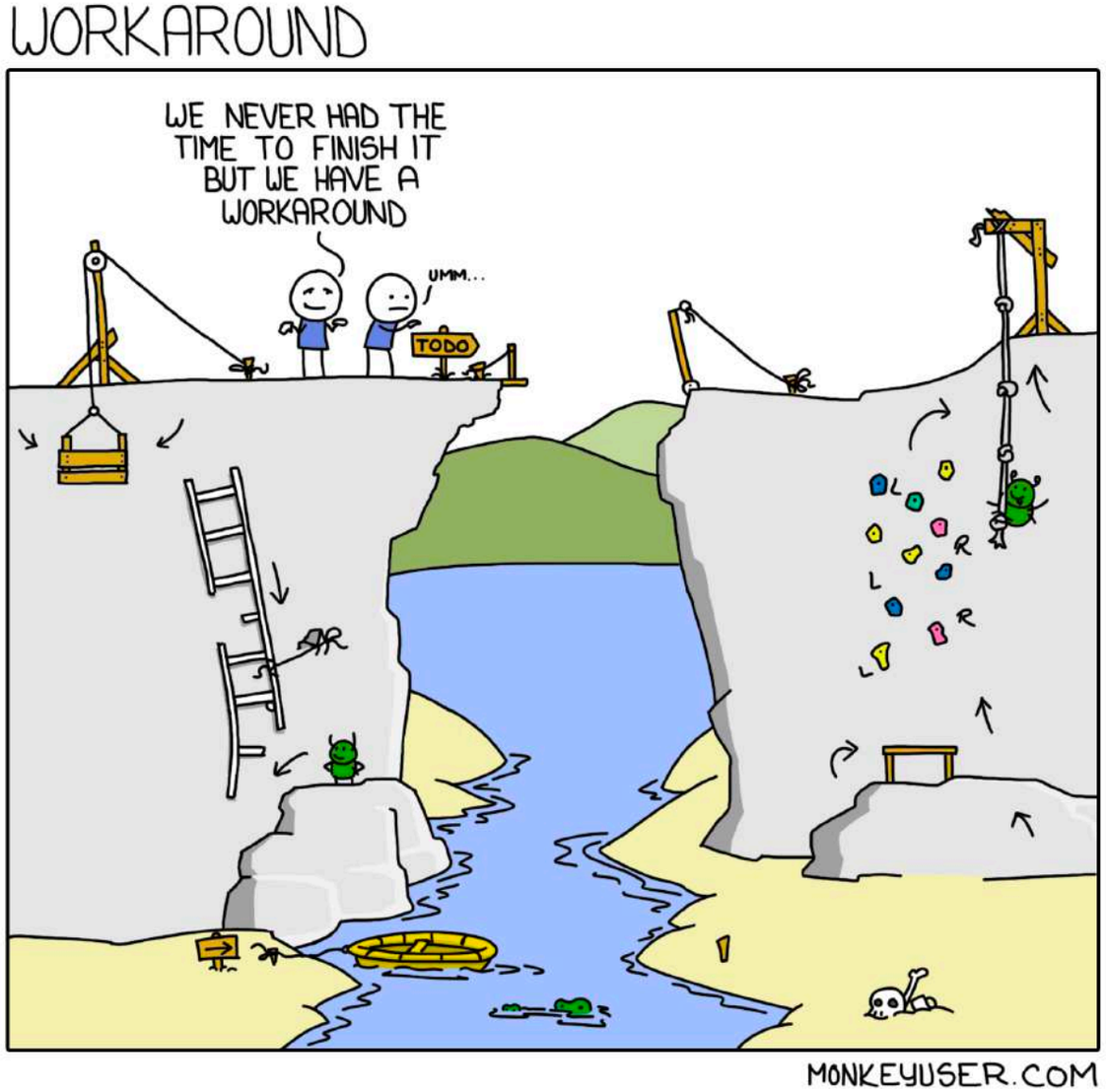
- Slice through marketing hype with X-ray vision.





Presentation Goals

- Reveal areas where those unfamiliar with Fabric's inner-workings will run into trouble.
- Slice through marketing hype with X-ray vision.
- Show how to design around the problems.



My Background

- Worked at IBM & IBM Research (USA+China) for 12 years developing database engines (Informix, Db2, Hyperledger Fabric, DataMirror, BluSpark, SolidDB)

- Chief Architect of Shanghai blockchain startup focused on using Hyperledger Fabric for Industrial IoT and supply chain management. Significant extensions of Fabric, including support for MongoDB, multiple collections, indexes, backup+restore, offline and online data verification, CloudHSM.

- Multiple customer engagements using Hyperledger, including Inter-Bank Payment System (IBPS) for the Monetary Authority of Singapore (MAS).

Programming Model

Myth: Modern Web App Dev Pattern Friendly b/c has Node.js

- Goal: Non-Repudiation — Alice signs transaction with her private key without delegating responsibility to a third party, such as an application server.

- Reality: Using Fabric as-is, this cannot be achieved in a three-tier architecture.

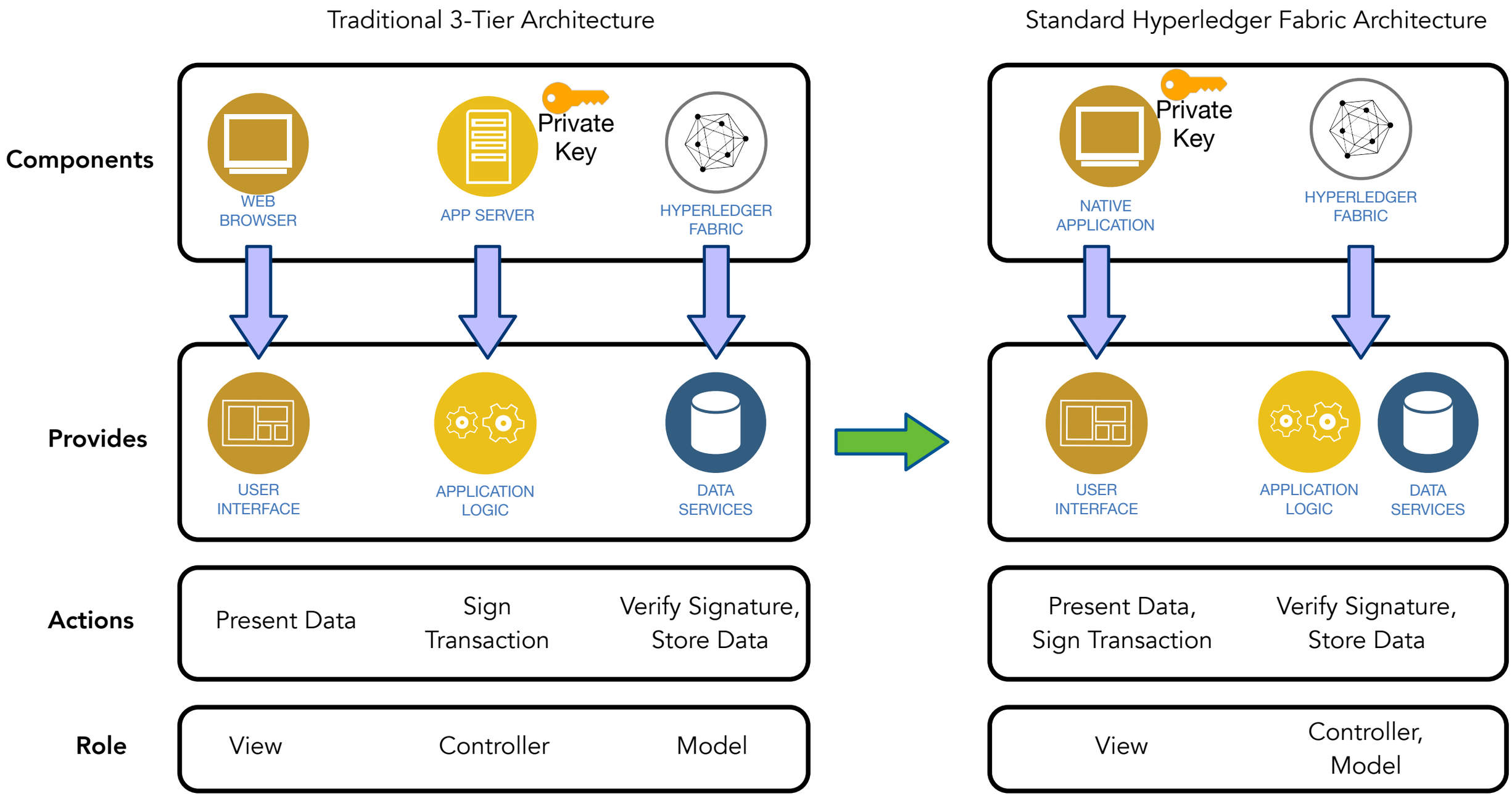
- Solutions:

  - Organization-level application server transacting on blockchain

  - USB HSM: Native mobile app w/secure enclave or USB, native desktop app

  - VirtualHSM: Browser can securely sign

Fabric ≠ 3-Tier Architecture



Myth: With support for Java & Node, any programmer can write smart contracts.

Myth: With support for Java & Node, any programmer can write smart contracts.

- Reality: There are many areas where developers will need to understand the Hyperledger Fabric transaction model to optimize performance.

Myth: With support for Java & Node, any programmer can write smart contracts.

- Reality: There are many areas where developers will need to understand the Hyperledger Fabric transaction model to optimize performance.

- Example: If multiple transactions are updating a particular key within a batch (block), all Tx subsequent to the first will be rejected, as the read-set key version no longer matches.

Myth: With support for Java & Node, any programmer can write smart contracts.

- Reality: There are many areas where developers will need to understand the Hyperledger Fabric transaction model to optimize performance.

- Example: If multiple transactions are updating a particular key within a batch (block), all Tx subsequent to the first will be rejected, as the read-set key version no longer matches.

- Solution: When storing an asset with multiple sub-concepts that can be independently manipulated, these smaller atoms should be stored in separate keys. This will make it possible for multiple Tx within a block to update different components.

Myth: With support for Java & Node, any programmer can write smart contracts.

- Reality: There are many areas where developers will need to understand the Hyperledger Fabric transaction model to optimize performance.

- Example: If multiple transactions are updating a particular key within a batch (block), all Tx subsequent to the first will be rejected, as the read-set key version no longer matches.

- Solution: When storing an asset with multiple sub-concepts that can be independently manipulated, these smaller atoms should be stored in separate keys. This will make it possible for multiple Tx within a block to update different components.

- Example: A company owns many vehicles. The original data model might have a single company asset with a nested JSON array-of-documents storing each vehicle's maintenance and usage information. If instead each vehicle is stored as a separate asset, they can all be updated simultaneously.

Read-Write Set Example

- The read-write set (RWSet) concept is integral to Fabric's disjoint transaction simulation and commit model.
- The read set ensures MVCC-like consistency in the version, and therefore value, of keys read by the smart contract. If the read set matches, the output write set can be committed without the committing peer running the smart contract.


```
{
  "collection_hashed_rwset": [],
  "namespace": "mycc",
  "rwset": {
    "metadata_writes": [],
    "range_queries_info": [],
    "reads": [
      {
        "key": "a",
        "version": {
          "block_num": "3",
          "tx_num": "0"
        }
      },
      {
        "key": "b",
        "version": {
          "block_num": "3",
          "tx_num": "0"
        }
      }
    ],
    "writes": [
      {
        "is_delete": false,
        "key": "a",
        "value": "OTA="
      },
      {
        "is_delete": false,
        "key": "b",
        "value": "MjEw"
      }
    ]
  }
}
```

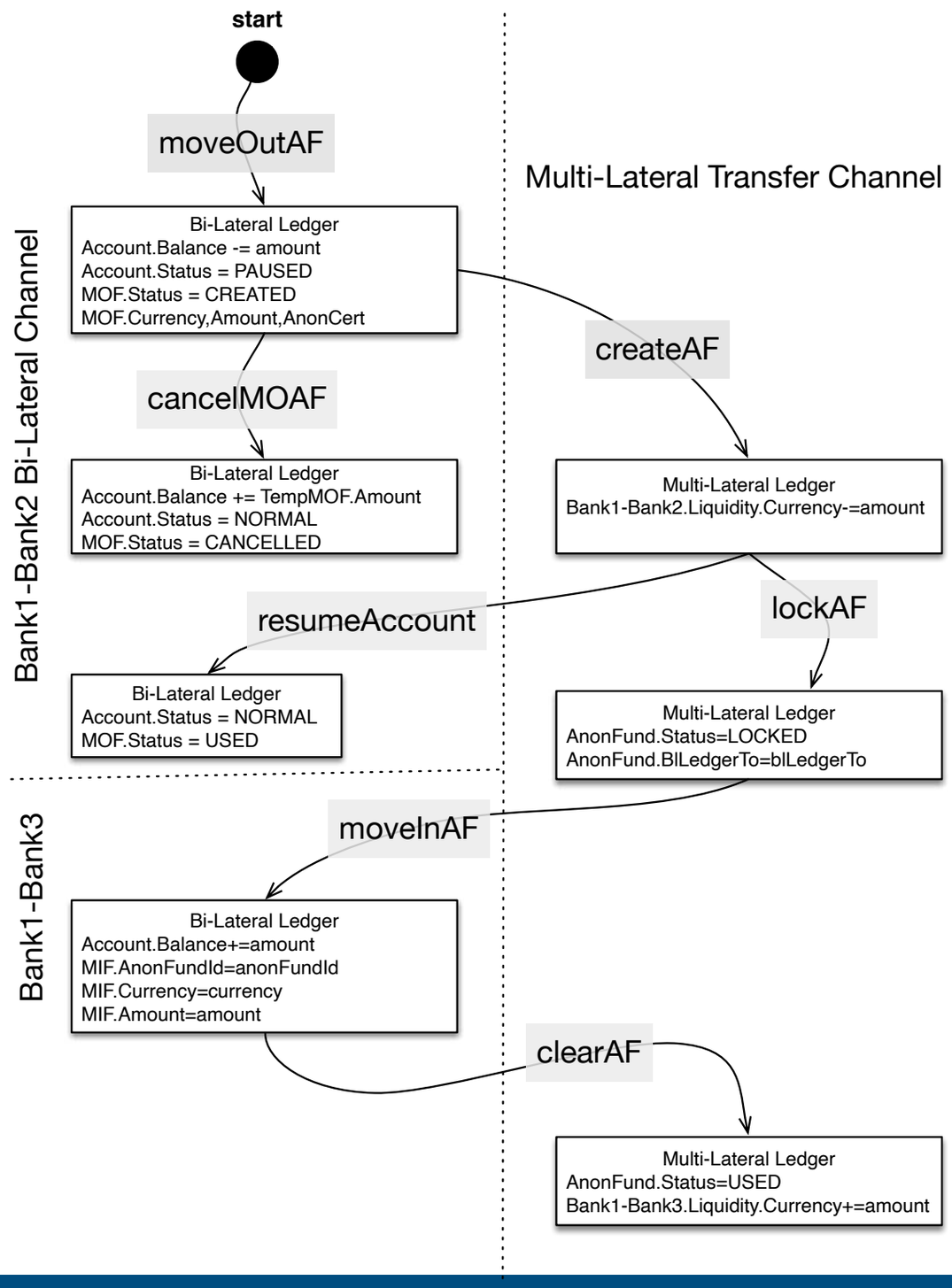
Myth: "No ledger data can pass from one channel to another"

- Reality: Cross-channel transactions can only read data. While writes can be attempted, they will not appear in the simulation read-write set returned to the client, and therefore will not appear in the ledger.

- Example Solution: When designing the IBPS mechanism to transfer a bank's liquidity between its bi-lateral channels, I developed a novel solution that only relied on cross-channel reads of previous information only "forward known" to the other party to ensure no money was created/destroyed in the process of re-balancing funds.

Quote Source: https://hyperledger-fabric.readthedocs.io/en/release-

IBPS Fund Transfer



JSON	Signature	Arguments *=transient
`{ "class": "MoveOutFund", "id": "a1b2c3", "owner": "Bank1", "currency": "SGD", "amount": "1.23", "status": "CREATED", "certificate": ".." }`	σ_{MoF}=H(mofId,currency, amount,status) Where mofId=Random#	moveOutFundId currency amount anonymous public key signature
`{ "class": "AnonymousFund", "id": "fg67hi89", "currency": "SGD", "amount": "1.23", "status": "CREATED", "certificate": ".." }`	$\sigma_{AnonFund}$=H(afId,currency, amount,status) Where afId=H(sourceChannelId,mofId)	sourceChannelId* moveOutFundId signature
`{ "class": "AnonymousFund", "id": "fg67hi89", "currency": "SGD", "amount": "1.23", "status": "LOCKED", "certificate": ".." }`	$\sigma_{AnonFund}$=H(afId,currency, amount,status)	anonymousFundId signature
`{ "class": "MoveInFund", "id": "d4e5f6", "owner": "Bank1", "currency": "SGD", "amount": "1.23", "status": "CREATED", "certificate": ".." }`	σ_{MiF}=H(mifId,currency, amount,status) Where mifId=H(targetChannelId,afId)	anonymousFundId signature
`{ "class": "AnonymousFund", "id": "fg67hi89", "currency": "SGD", "amount": "1.23", "status": "USED", "certificate": ".." }`	$\sigma_{AnonFund}$=H(afId,currency, amount,status)	targetChannelId* anonymousFundId signature

Myth: Heterogenous DB Support

- Reality: Fabric supports a pluggable database model with built-in support for LevelDB and CouchDB. If any smart contract uses the rich query feature of CouchDB in smart contract, then all peer organizations must also use CouchDB to successfully perform transaction simulation (commit-only peers could still work in theory).

- Looking at the longer term future, as multiple networks merge, just as companies merge and integrate/normalize IT systems, there will be a similar need for Fabric. (Multi-database support among peers on the network—chain code for CouchDB implementation specific.)

- Solution: Organizations need to plan for the maximal spanning feature set. Today, that would mean that even though LevelDB might be acceptable for your use cases today, perhaps it would be better to choose CouchDB. Better yet, Fabric should adopt MongoDB, which supports many useful analytics features, such as hierarchical queries, useful for supply chains.

Security

Hype: Channel Policy Mechanism Can Enforce Related Party Endorsement

- The built-in channel policy mechanism (VSCC) does not support per-method and per-transaction specific behavior.
- For example, a channel contains three participants: Alice, Bob, and Charles. The smart contract manages the ownership of assets.
    - To transfer an asset, both the buyer and seller organization must agree to, and therefore endorse, the transfer.
    - To cover all possible cases, every pair of participants must be listed in the policy, resulting in the following:
      (A&B)||(B&C)||(A&C)
    - And, while, this clearly covers situations where Alice and Bob are interacting, it also means that Bob could get Charles to endorse transfer of an asset owned by Alice to Bob, without Alice's endorsement being required.
    - The problem is that the endorsement policy needs to be responsive to the method, transfer, and to the buyer and seller arguments corresponding with the endorsing organization's peers.
- Solution: To be useful, a customer will need to create a custom golang Verification System ChainCode (VSCC) that inspects the method and arguments to enforce meaningful business-level verification.

Myth: Channels Solve All Your Data Privacy Problems

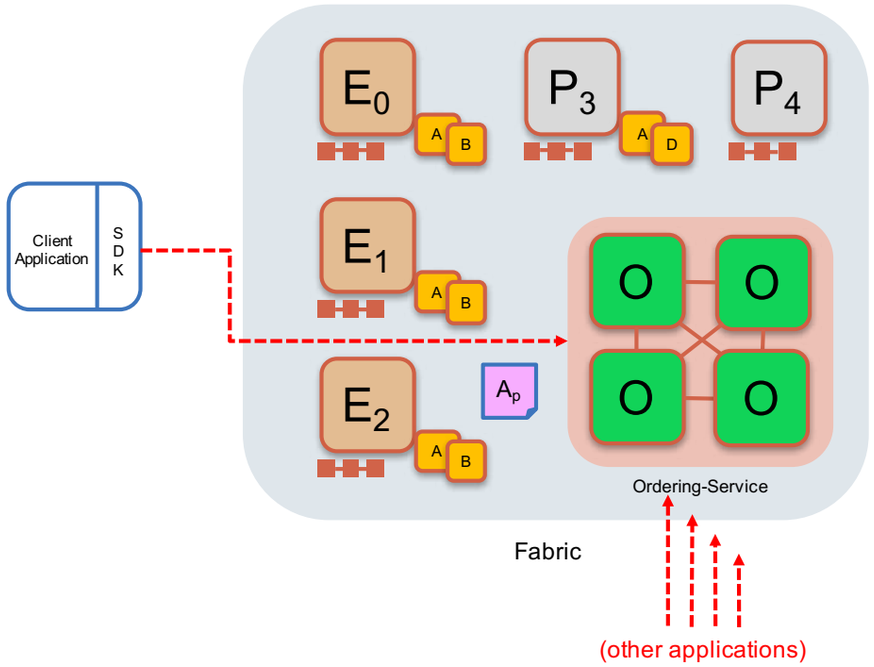
- Reality: The ordering service is a single point of trust. All transactions on all channels flow through it.

- As a central banker, this is a good thing as it represents a single point of control and audibility.

- As an organization interested in a decentralized solution, this is a bad thing.

- Solution: It depends on your objectives. A consortium might prefer use of a centralized ordering service to regulate members. Most organizations will prefer to use private collections.

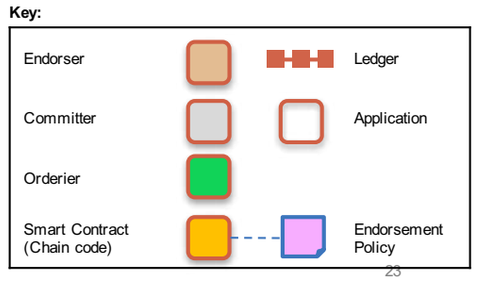
Ordering Service SPoT



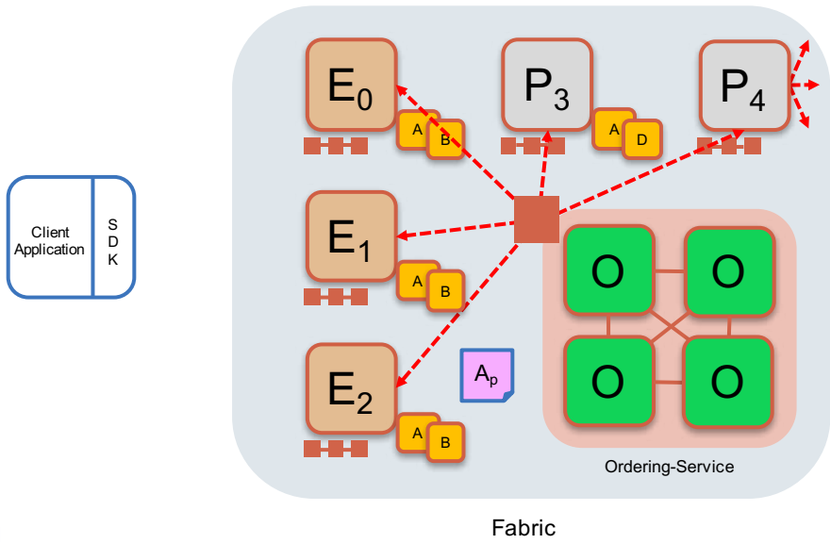
Application submits responses for ordering

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

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



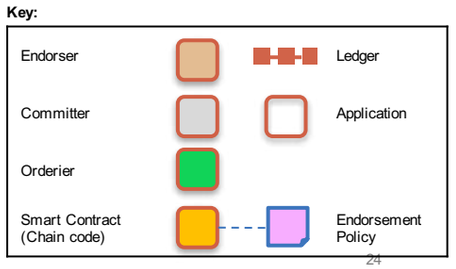
23



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)



24

DevOps

Myth: Channels Are Free

- Reality: Channels multiply like crazy in many real-world scenarios.

- When creating the Inter-Bank Payment System for Singapore's Central bank, a little over 200 banks needed to privately communicate with each other through bi-lateral transfer channels to ensure no information would appear in other participants ledgers.

    - This means each bank has approximately 200 connections to other banks. Each of those other banks similarly has ~200 connections.

    - The net product of all of these private channels is an N^2 explosion of over 20,000 channels.

    - The ordering service wasn't designed to handle so many channels and becomes quite slow.

- Solution: Private Collections

Myth: Private Collections Solve the Channel Explosion Problem

- Continuing from the previous slide on channels being free, private collections are often touted as a holy grail solution. And, to be fair, in some ways they are, but…
  - You still need to define all 20k private collections in a single JSON file for the channel definition.
  - Although the ordering service is no longer a performance or trust problem, now we need to establish a P2P network between all organization's peers.
- Solution: Think carefully about your network architecture.

Private Collection Policy

```
// collections_config.json

[
  {
        "name": "collectionMarbles",
        "policy": "OR('Org1MSP.member', 'Org2MSP.member')",
        "requiredPeerCount": 0,
        "maxPeerCount": 3,
        "blockToLive":1000000,
        "memberOnlyRead": true
  },

  {
        "name": "collectionMarblePrivateDetails",
        "policy": "OR('Org1MSP.member')",
        "requiredPeerCount": 0,
        "maxPeerCount": 3,
        "blockToLive":3,
        "memberOnlyRead": true
  }
]
```

Myth: Data Privacy is Easy

- Reality: Continuing with the IBPS example, even with channels or private collections, you still have money (assets) divided up into hundreds of partitions to avoid other network participants having a view of your total liquidity made public.
  - This results in seemingly simple problems becoming more complicated. For example, if a specific private collection/ channel lacks sufficient liquidity to fund a transfer between two banks, money must be transferred from another private collection in a way that ensures no funds are created/ destroyed in the process while preserving privacy.

- Solution: Hire smart people experienced with developing multi-party algorithms and distributed systems.

Myth: A Hyperledger Fabric network is easy to administer

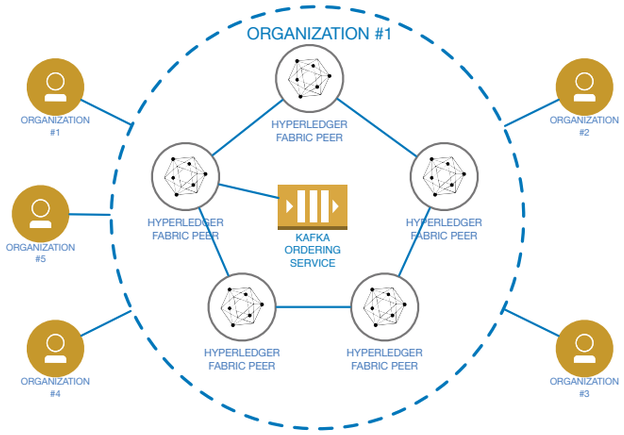
- Reality: Running a production Hyperledger Fabric network requires specialized routing and firewall rules to establish secure links between organizations peers, MSPs, and the ordering service. Additional administrators will need to be involved with management of a cross-organizational CA.

- IT network engineers need to work as part of a consortium to determine the appropriate way to expose their peer to other organizations to receive Tx endorsement proposal/simulation requests while minimizing an attackers ability to gain access to sensitive information stored in the peer database.

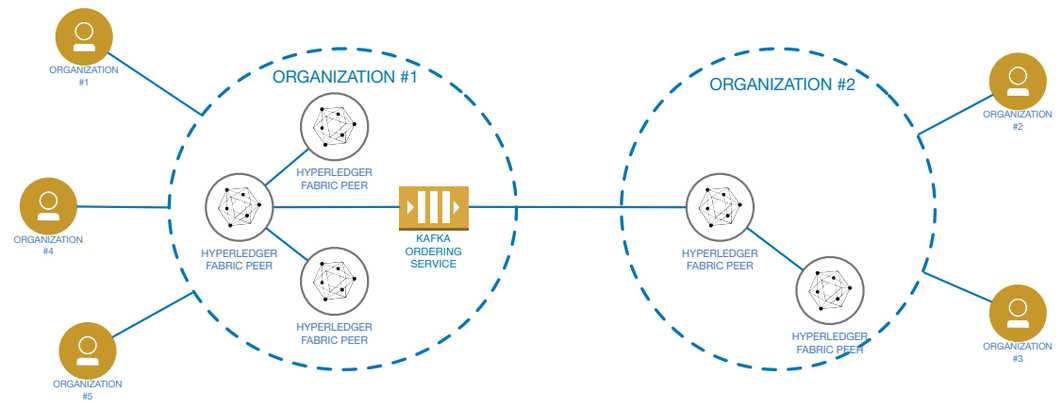
- Solution: Co-locate all organizations' peers within a single data center or provider, such as AWS/Azure/IBM. Of course, being in a single AZ is a risk unto itself. Ultimately, you need highly skilled network engineers on-board.

Peer Topology

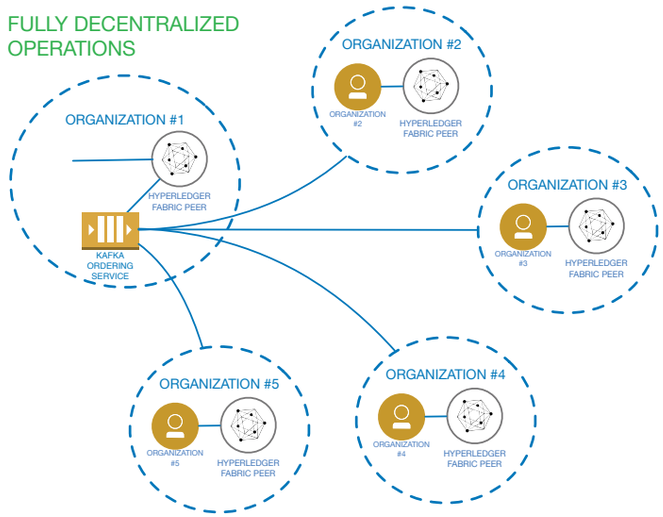
SINGLE TENANT
OWNER-OPERATOR



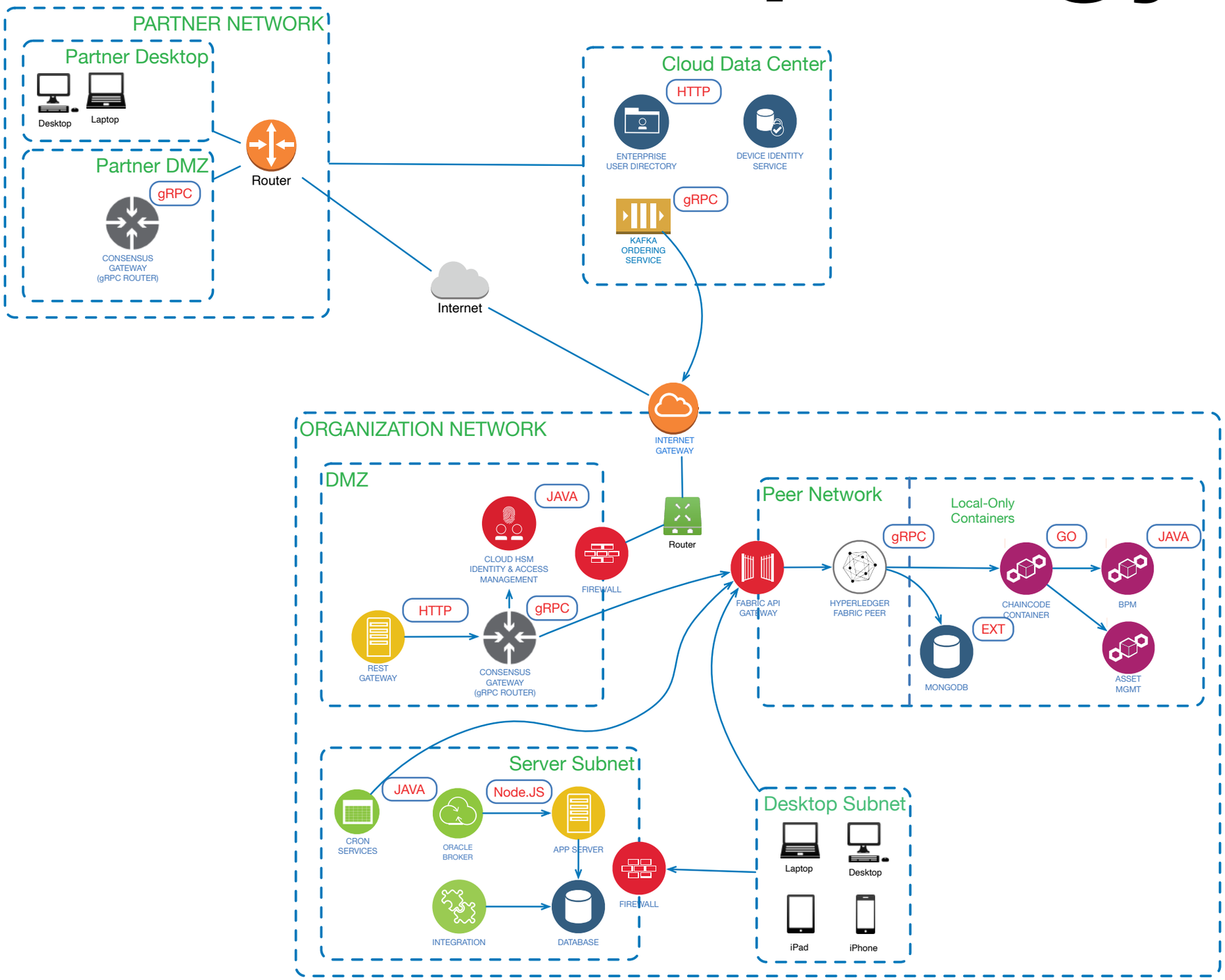
MULTI-PARTY
OPERATIONS



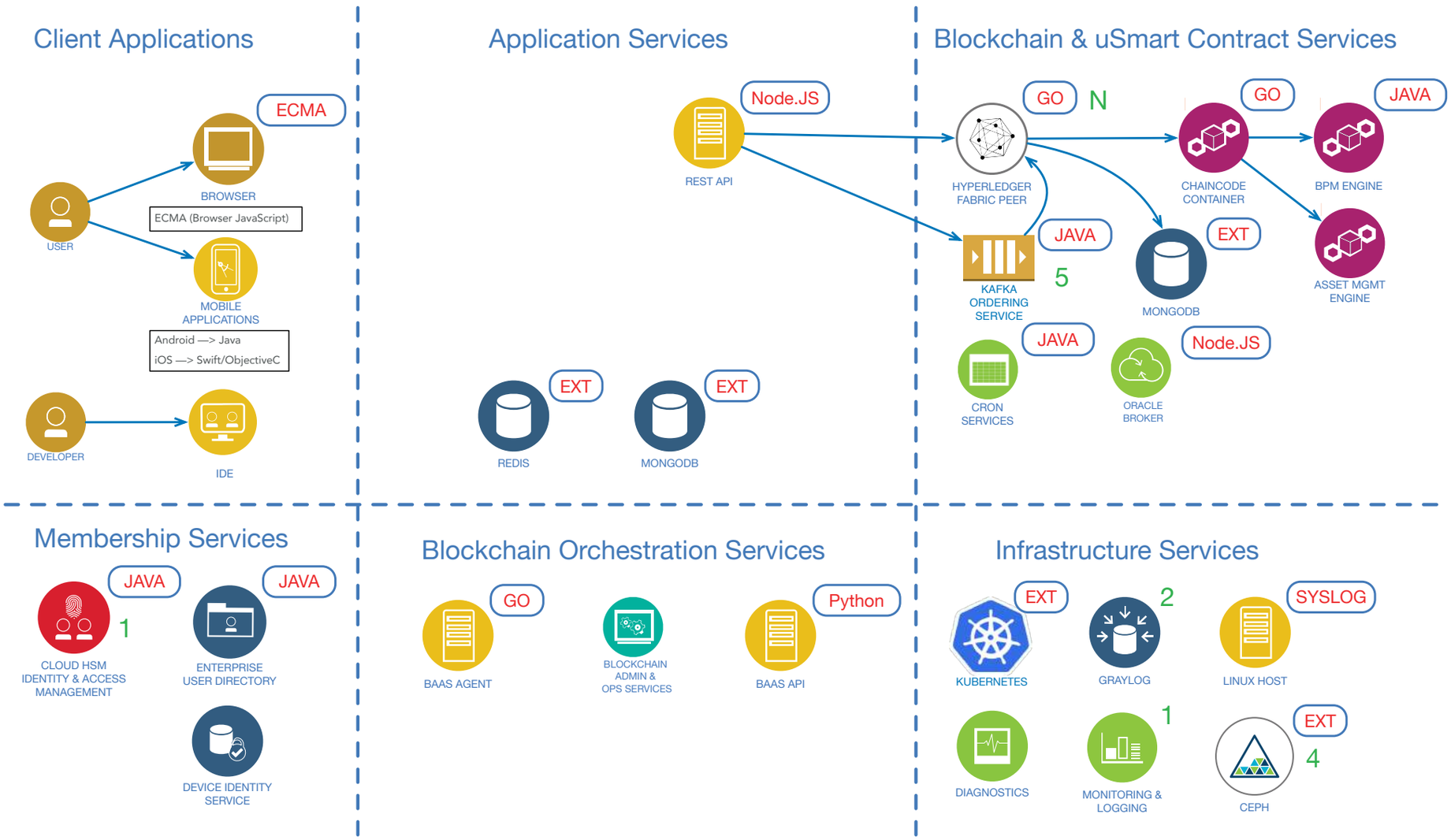
FULLY DECENTRALIZED
OPERATIONS



Network Topology



Fabric Components



Myth: Clusters Don’t Need Backups

- Reality: Disasters happen.
  - If all Fabric peers are in the same data center/locale and suffer a direct lightning strike, fire, earthquake, etc… only a remote backup will be sufficient for recovery.
  - If crypto ransomware encrypts all online volumes, having an offline backup prior to the attack provides the option to reject paying the attacker’s ransom without complete data loss.
- Recovery by replaying the logs from the genesis block over the network will be painfully slow.
- Private collections are not part of ledger.
- Solution: Ok: Use LVM and volume snapshots. Better: Customize Fabric to pause processing and use native backup tool for database.

Myth: Fabric needs few network & hardware resources to operate

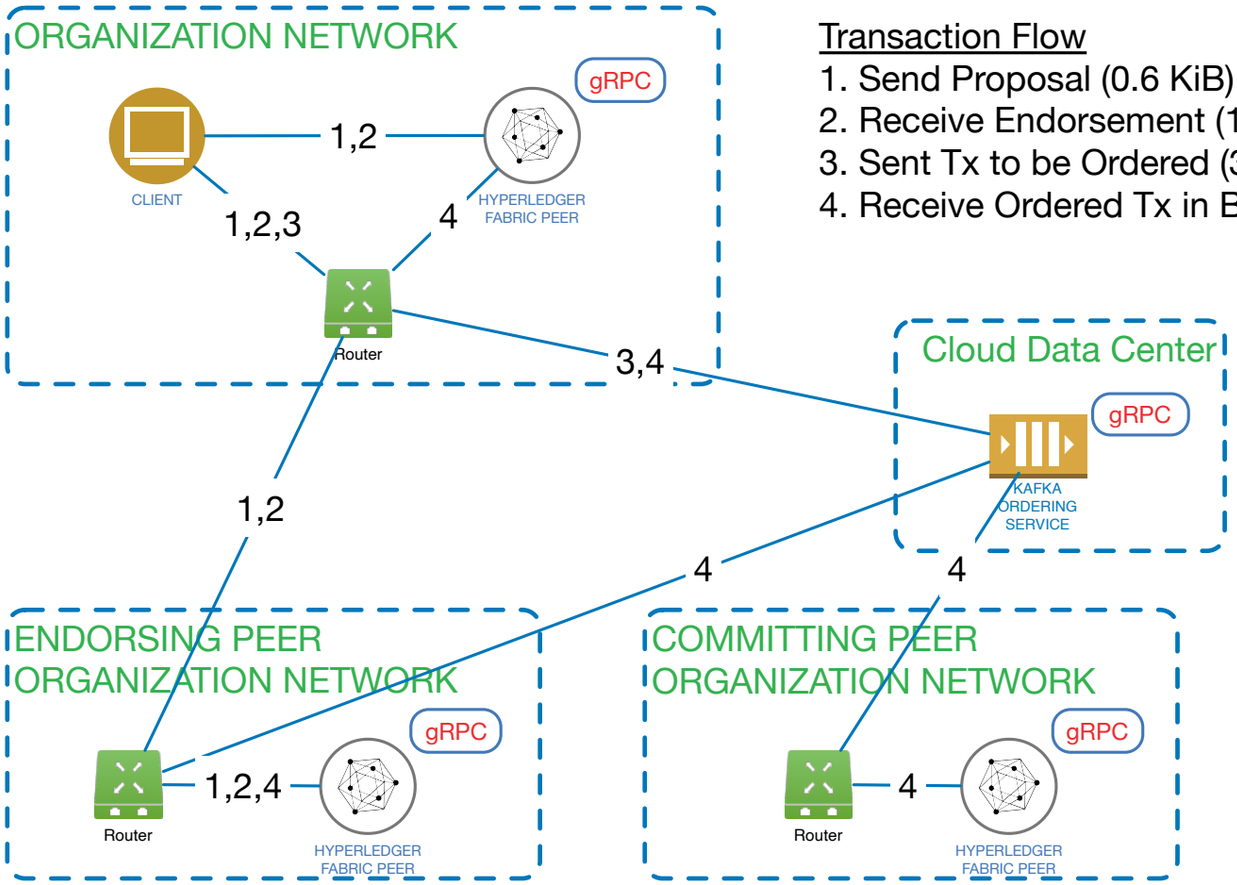
- Reality: All blockchain systems are, in essence, unbounded append-only logical logs. Storing all data from the genesis block onward and maintaining it has a non-trivial cost.
  - Solution: Increase robustness by including information from older blocks to incentivize keeping them available. Use hierarchical storage to reduce cost of storing old blocks.
- Reality: The ordering service is a significant source of network congestion. All transactions must flow into it one time and are replicated out to “N” organizations. This problem grows quadratically with the number of organizations.
  - Thankfully organizations use gossip between replica peers internally to reduce inter-network traffic.
  - Solution: Use a gossip-based Tx distribution to eliminate need for ordering service to directly talk with all organizations.

Bandwidth Required

- Minimal transaction proposal size is 0.6KiB
- Proposal endorsement is 1.4 KiB
- 3.4 KiB to/from ordering service for each commit peer

Transactions/Second on Gigabit Network

Endorsers	Commiters (Orgs)	Bandwidth (KiB/sec)	1GbE Tx/sec
1	1	8.8	14204.55
1	2	12.2	10245.90
1	4	19	6578.95
1	8	32.6	3834.36
2	2	14.2	8802.82
2	4	21	5952.38
2	8	34.6	3612.72
2	16	61.8	2022.65
2	32	116.2	1075.73
2	64	225	555.56
2	128	442.6	282.42
3	3	19.6	6377.55
3	128	444.6	281.15
3	256	879.8	142.08
3	512	1750.2	71.42



Transaction Flow
1. Send Proposal (0.6 KiB)
2. Receive Endorsement (1.4 KiB)
3. Sent Tx to be Ordered (3.4 KiB)
4. Receive Ordered Tx in Block (3.4 KiB)

1 Tx Bandwidth = 2.0 KiB x P_E + 3.4 KiB + 3.4 KiB x P_C

Where
P_E = Number of Endorsing Peers
P_C = Number of Committing Peers

Final Words

Conclusion

- Experience matters.

- Blockchain adoption is a marathon. Start to build in-house experience, especially for DevOps and networking.

- Adopting Fabric, or any blockchain system, as an integral part of your business will require thoughtful analysis of the business network and technology to maximize ROI.

- Digitalization and a concrete understanding of your existing businesses process are critical to success.

Final Thoughts

- When Fabric was designed, it was not clear what the emerging business model changes would be. The hope was to be sufficiently flexible to provide a good starting platform given broad configurability and modularity.

- As the most enterprise-ready of the first wave of blockchain products, there are few situations for which Fabric cannot be adapted.

- Adoption requires significant expertise in network security & routing, cryptography, databases, containerization, and distributed computation.

Contact

- Twitter: @lfeagan
- LinkedIn