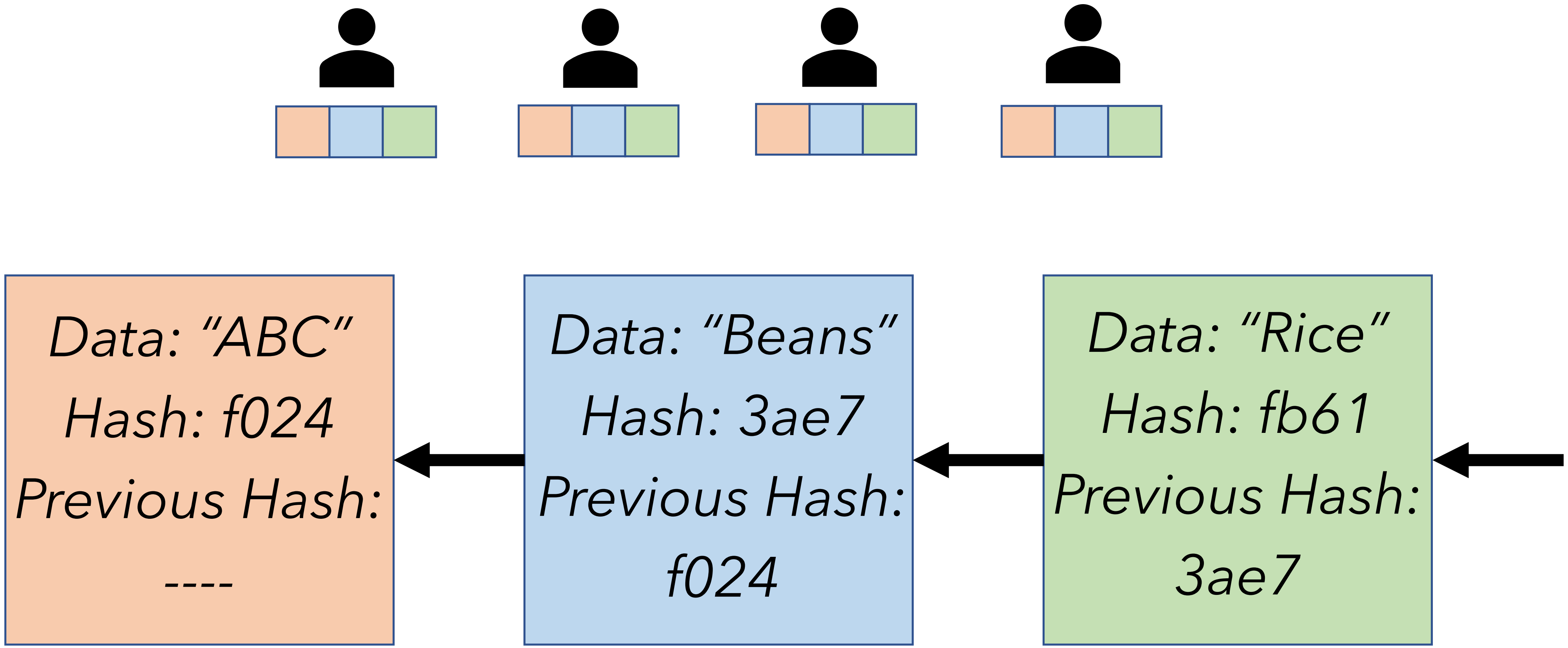
Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains

Presentation by Ben Manley

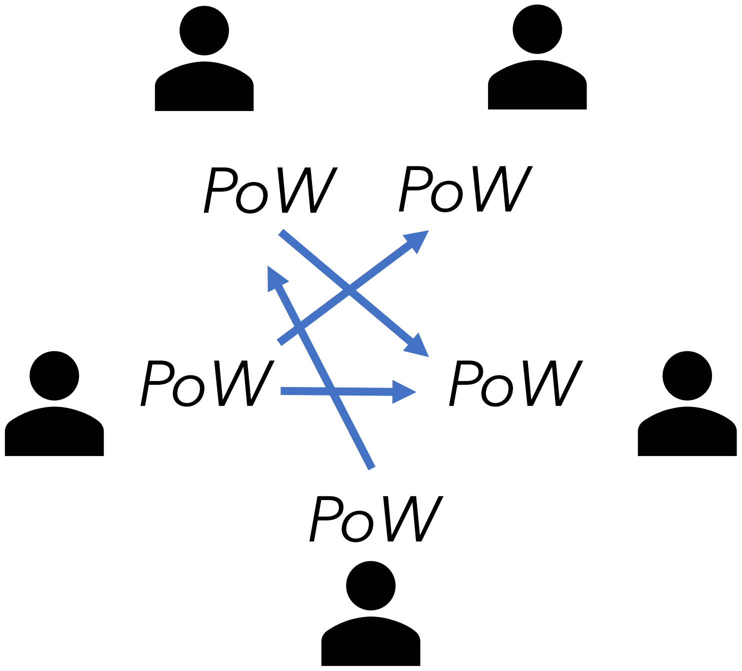
What is a Blockchain system?

Untrusting peers holding immutable ledgers of transactions



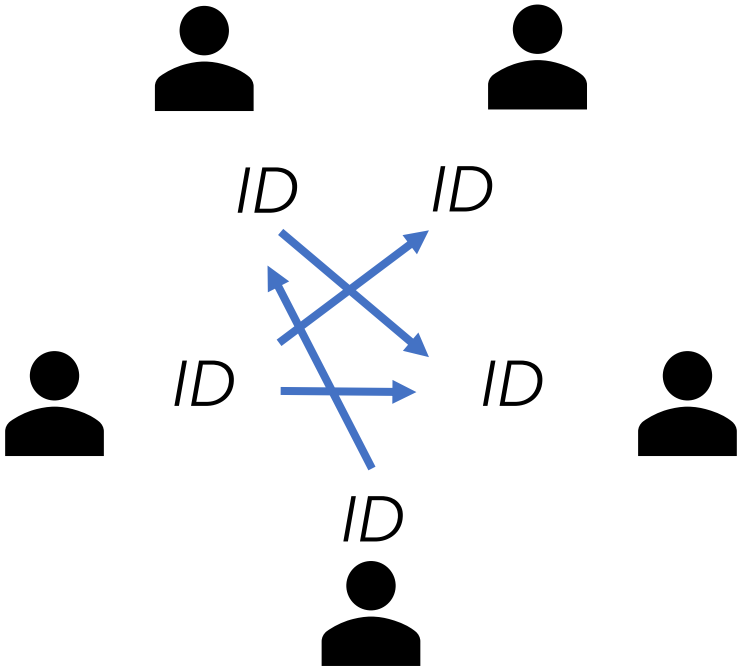
What is a Blockchain system?

Public / Permissionless



Proof-of-work (PoW) consensus

Permissioned

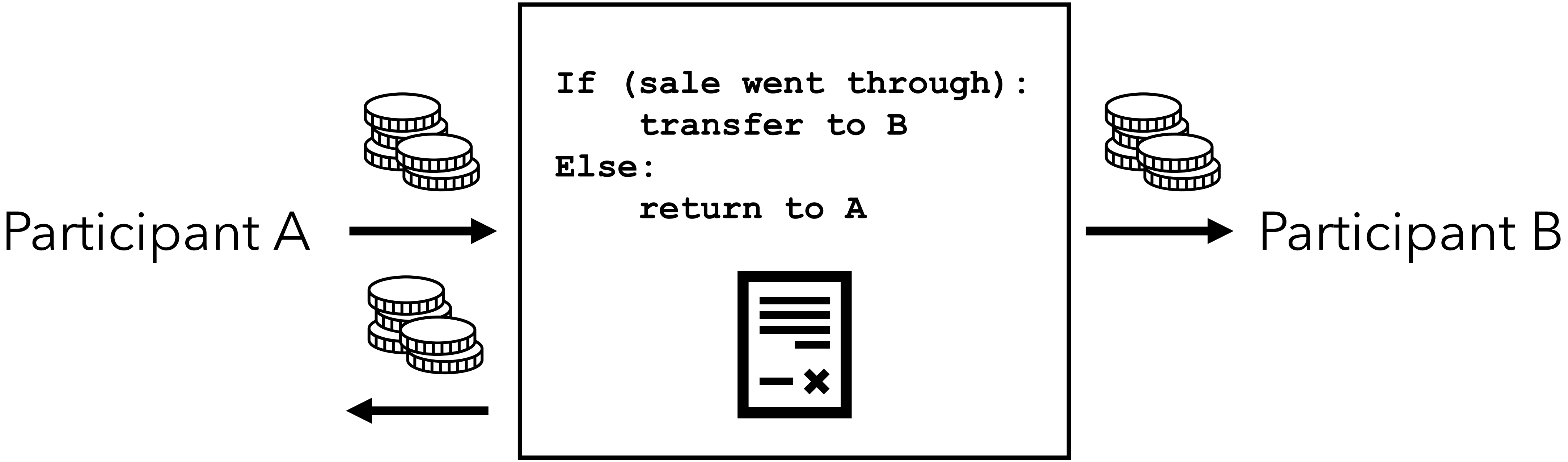


Byzantine-fault tolerant consensus

Smart Contracts

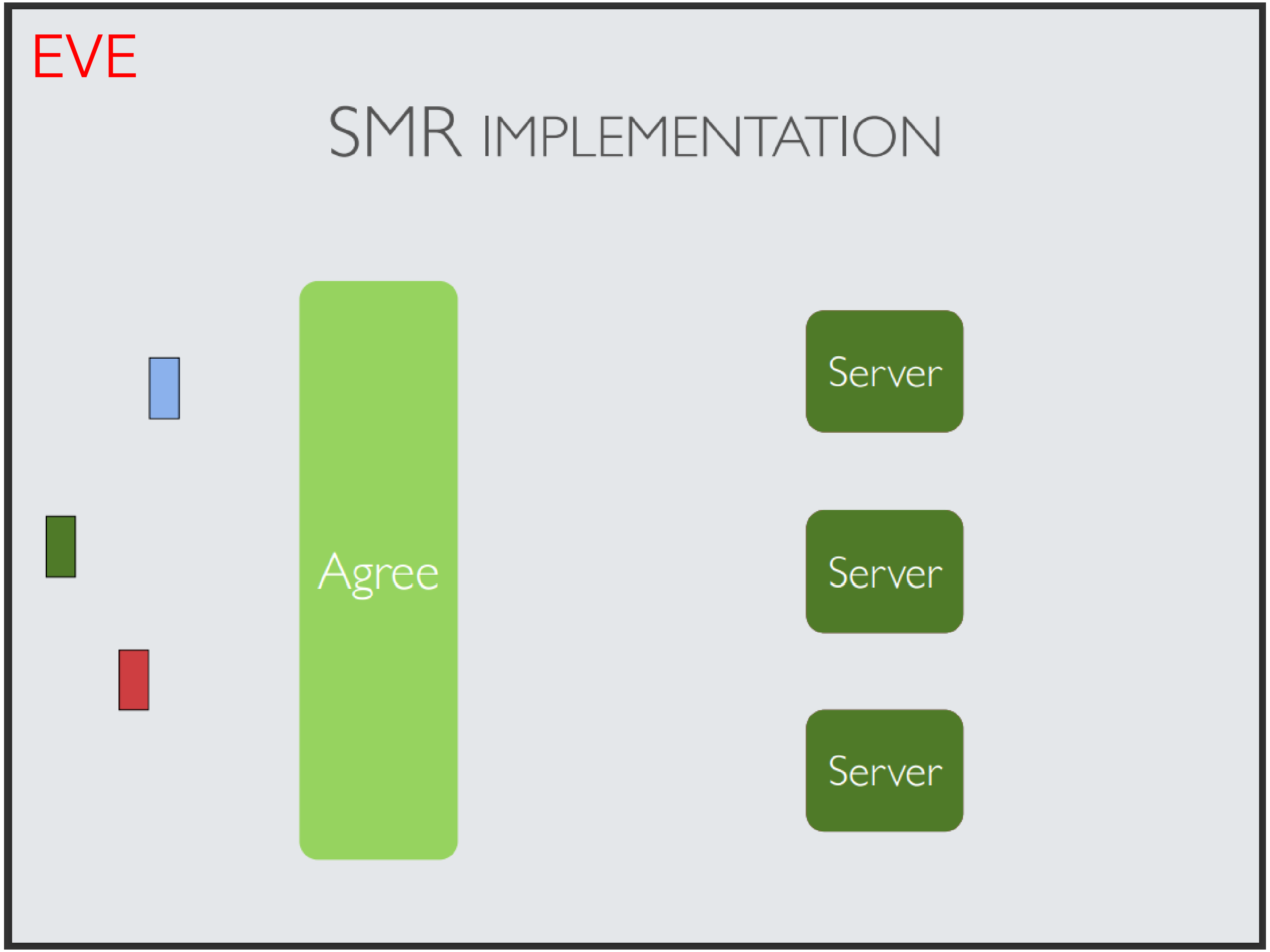
Programmable transaction logic

Cryptocurrency example:



Why reinvent the wheel?

Everyone else uses Order-Execute



Order-Execute Sucks: Sequential Execution

Limits throughput

Denial-of-Service (DoS) from just a long/infinite smart contract

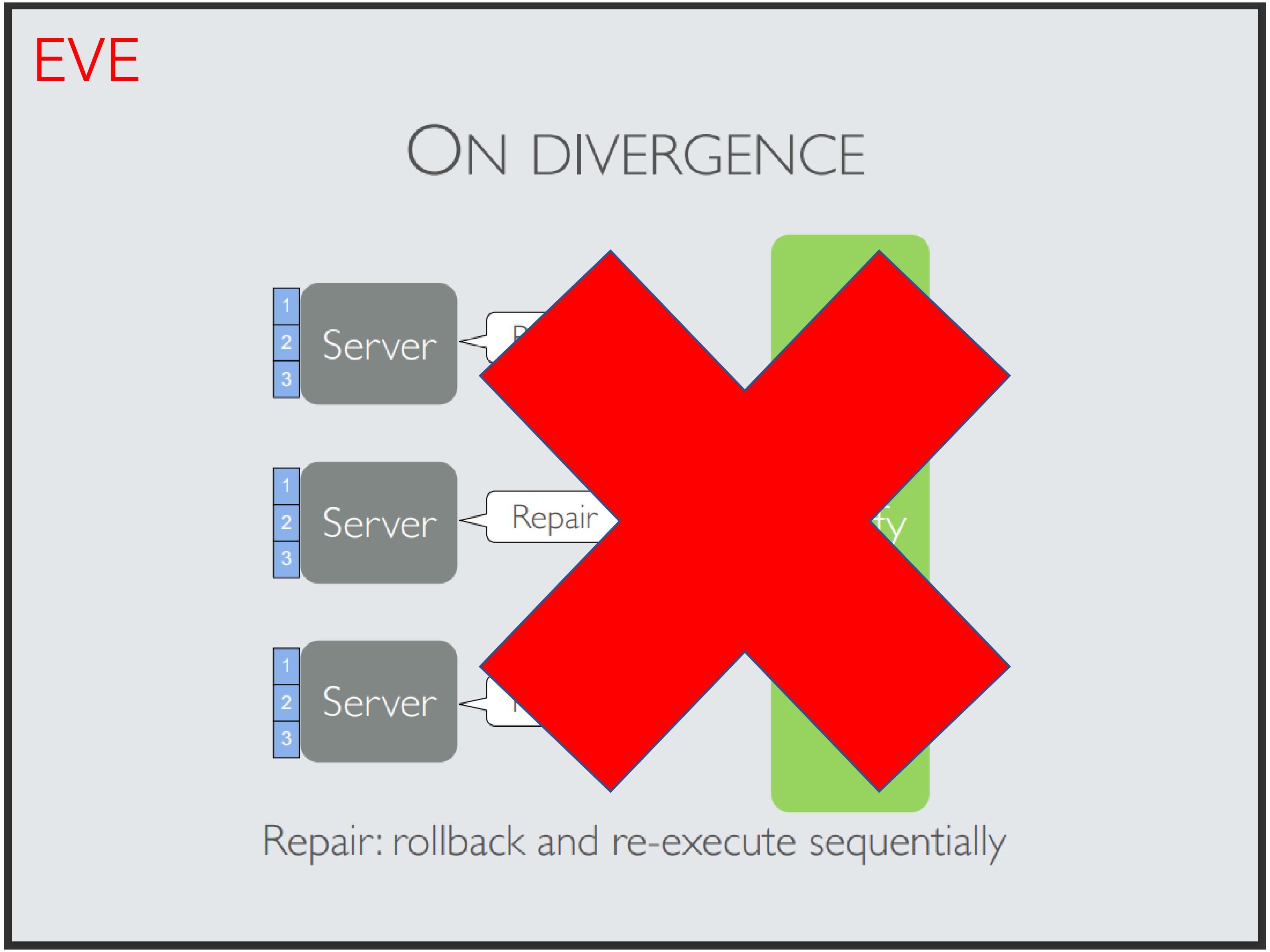
Ethereum solves with “gas”
(not helpful without a cryptocurrency)

Order-Execute Sucks: Non-Determinism

Operations after SMR must be deterministic

Could require specific languages

Can't trust programmers w/ determinism in general languages



Other Previous Limitations

Confidentiality	No secret smart contract logic, etc.
Fixed trust model	Applications stuck with BFT's *f* out of >*3f*
Hard-coded consensus	Applications stuck with whatever protocol the blockchain chose

Fabric

Components Transaction Flow Evaluation

Chaincode

Programmable transaction logic (Smart Contracts) with
endorsement policy

Fabric is the first to support standard programming languages
(non-determinism is allowed!)

Fabric is "the first *distributed operating system*"

Peers

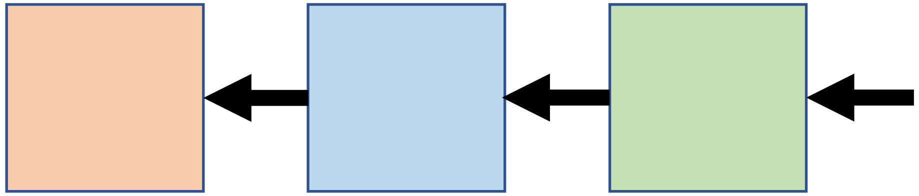
Actions

- Simulating and endorsing transactions
- Gossiping results
- Validating and committing

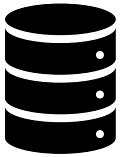
Chaincode can call
GetState(key)
DelState(key)
PutState(key, val)

Components

- Docker for chaincode "simulation"
- Ledger (hash-chained block store)



- Key-Value Store (KVS)



```
key1: (val1, ver1)
key2: (val2, ver2)
...
```

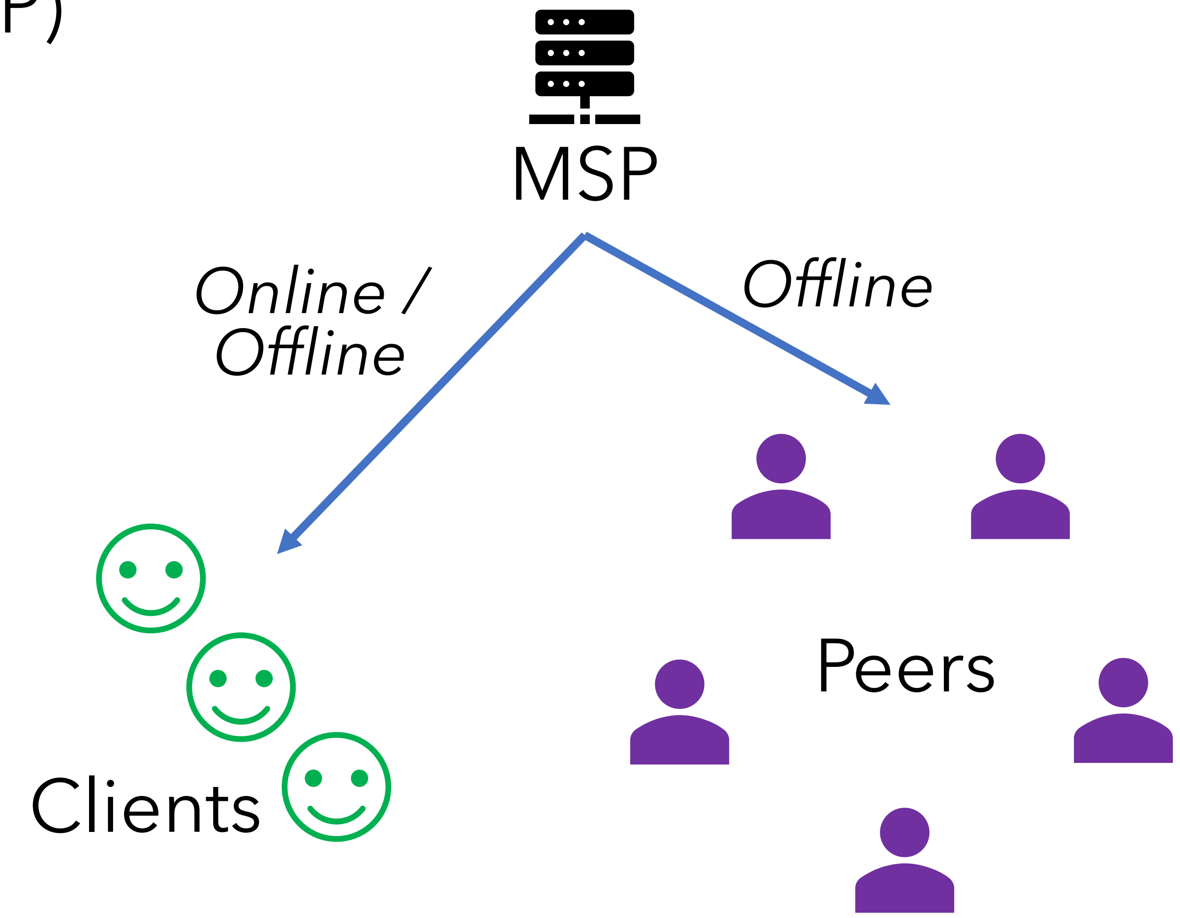
Membership Service

Membership Service Provider (MSP)

- Issues credentials
- Maintains identities
- Abstracts general auth
- Can be multiple

At each node:

- Authenticates transactions
- Signs endorsements



Ordering Service

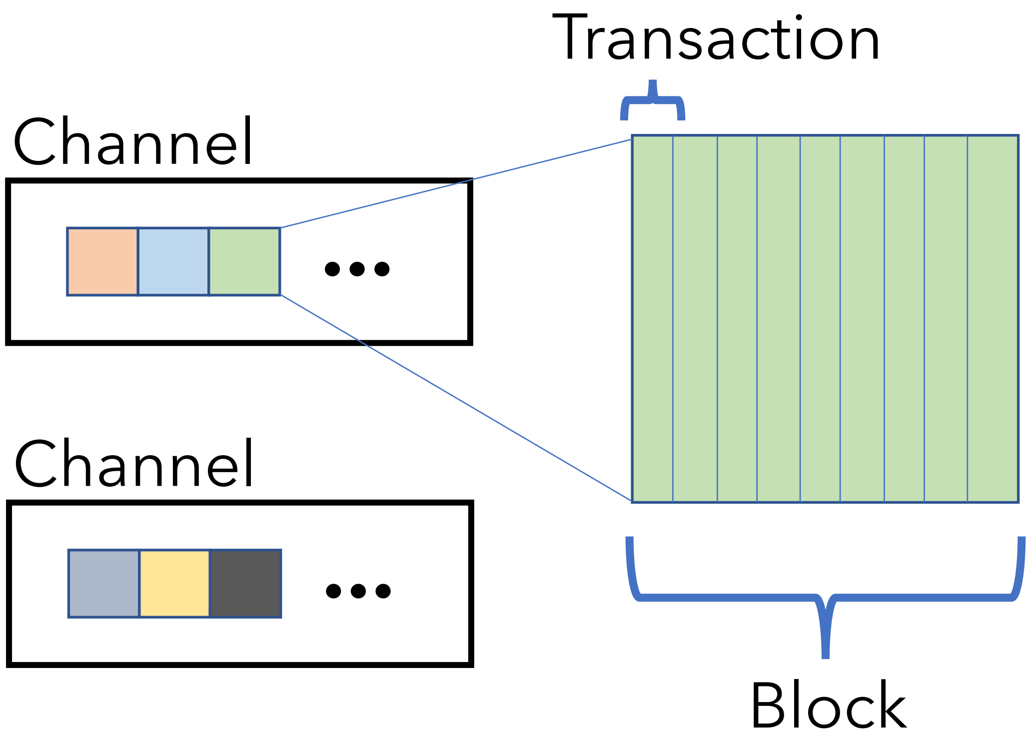
Maintains multiple *channels*

- One logical blockchain each
- Separate total order
- Reconfig and access control

Batches transactions into *blocks*

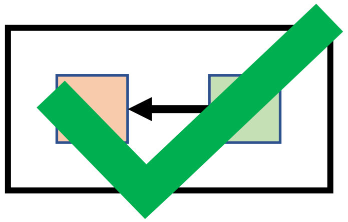
- Deterministic
- (# transactions, # bytes, timeouts)

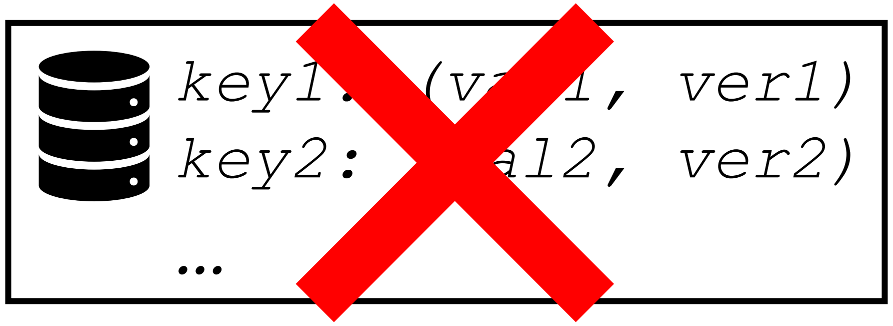
Made up of OSNs (Ordering Service Nodes), or *orderers*



Ordering Service

Provides *atomic broadcast* for ordering transactions (**stateless!**)





API (invoked by peer):

- *broadcast(tx)*
  Client calls to broadcast transaction after receiving endorsements
- *B ← deliver(s)*
  Client calls to retrieve block *B* at sequence number *s*

Ordering Service

Guarantees (informally):

- **Agreement**: All peers see same *B* delivered for a given *s*
- **Hash chain integrity**: block at *s+1* holds hash of block at *s*
- **No skipping**: If peer delivers at *s*, it has already delivered [0, *s-1*]
- **No creation**: All *tx* in a correctly-delivered block *B* was broadcast
- **Validity**: If a correct client calls *broadcast(tx)*, every correct peer eventually delivers a block *B* containing *tx*

A Day in the Life of Fabric

Transaction Flow

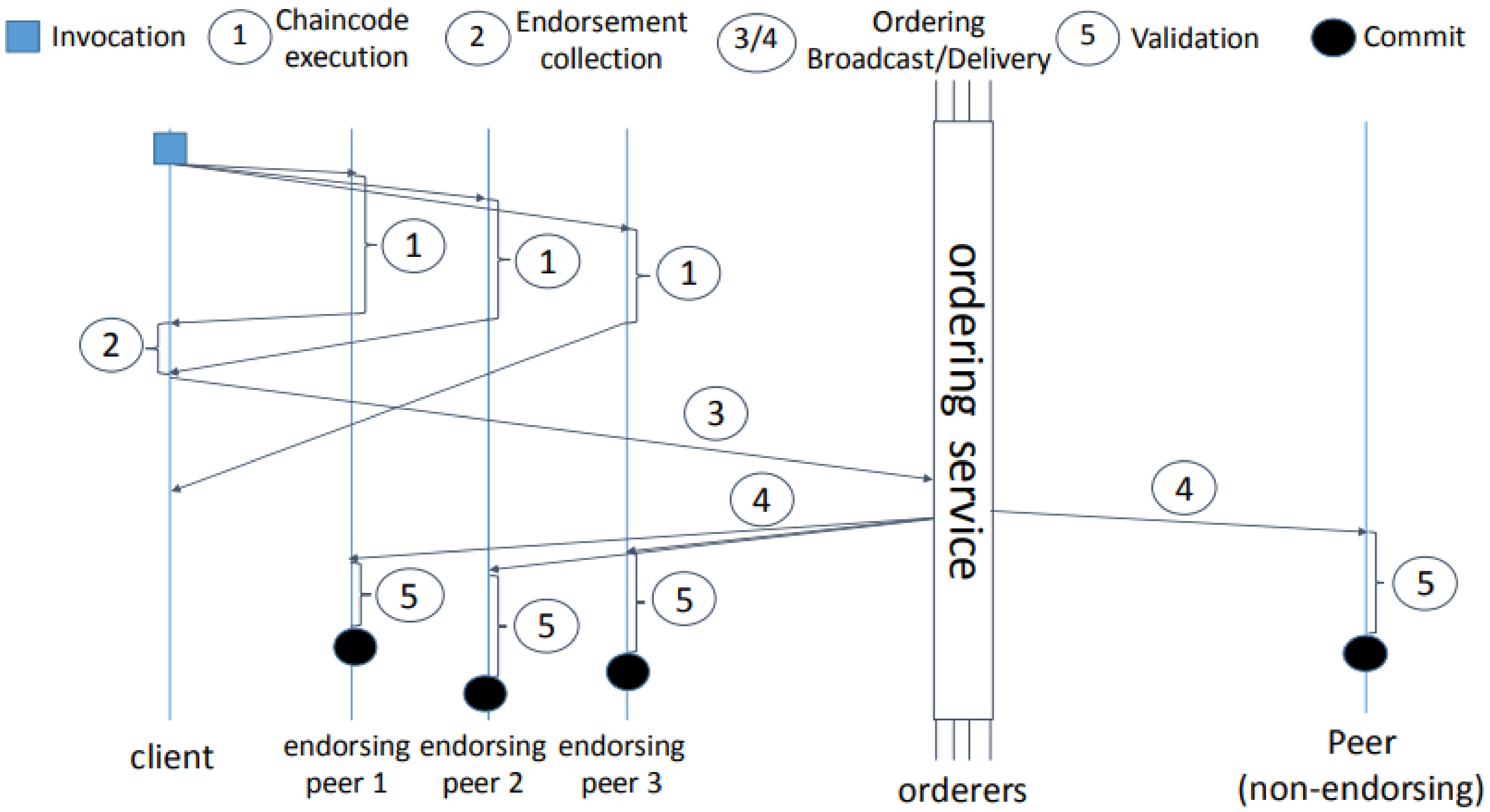
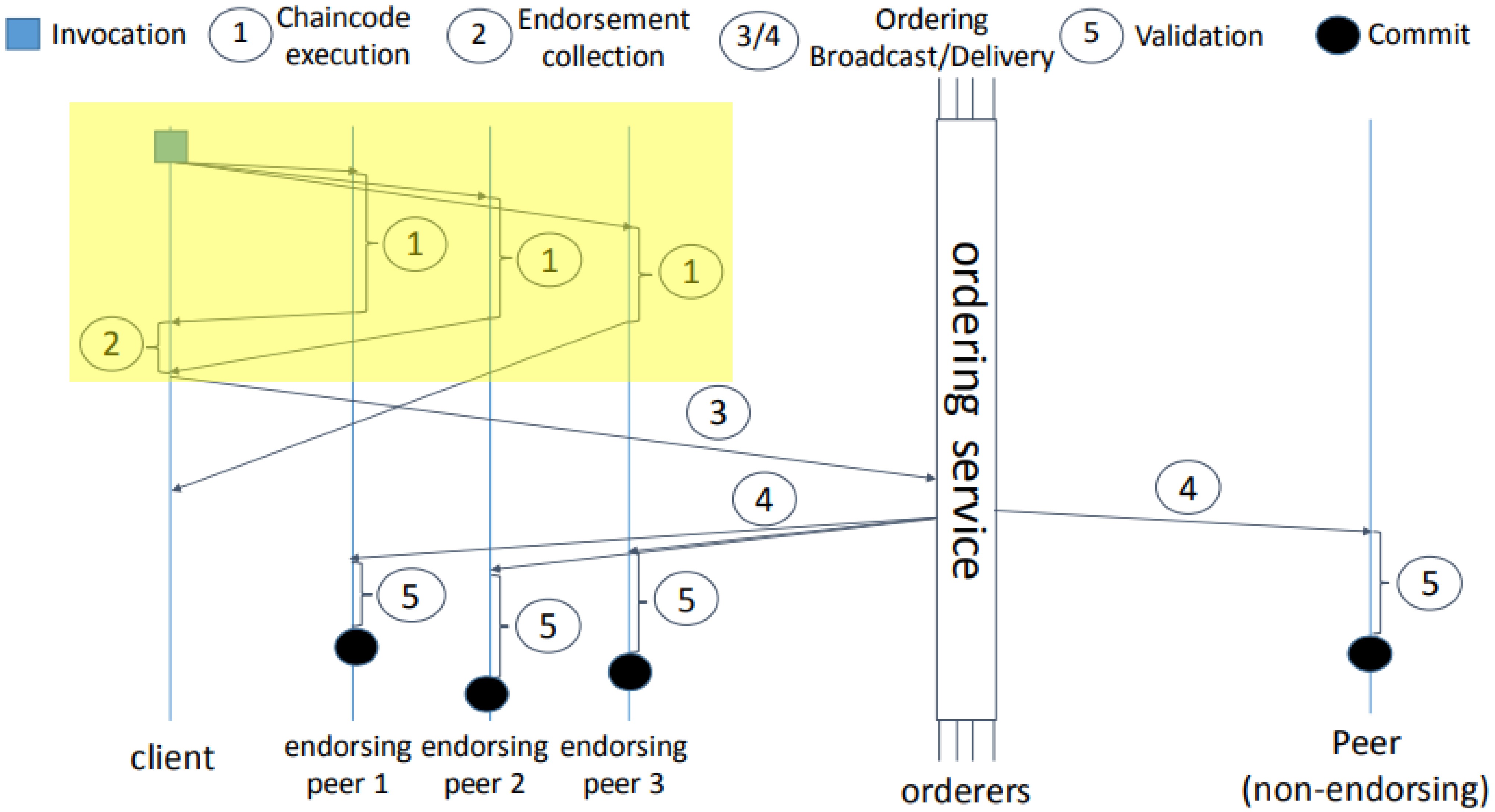


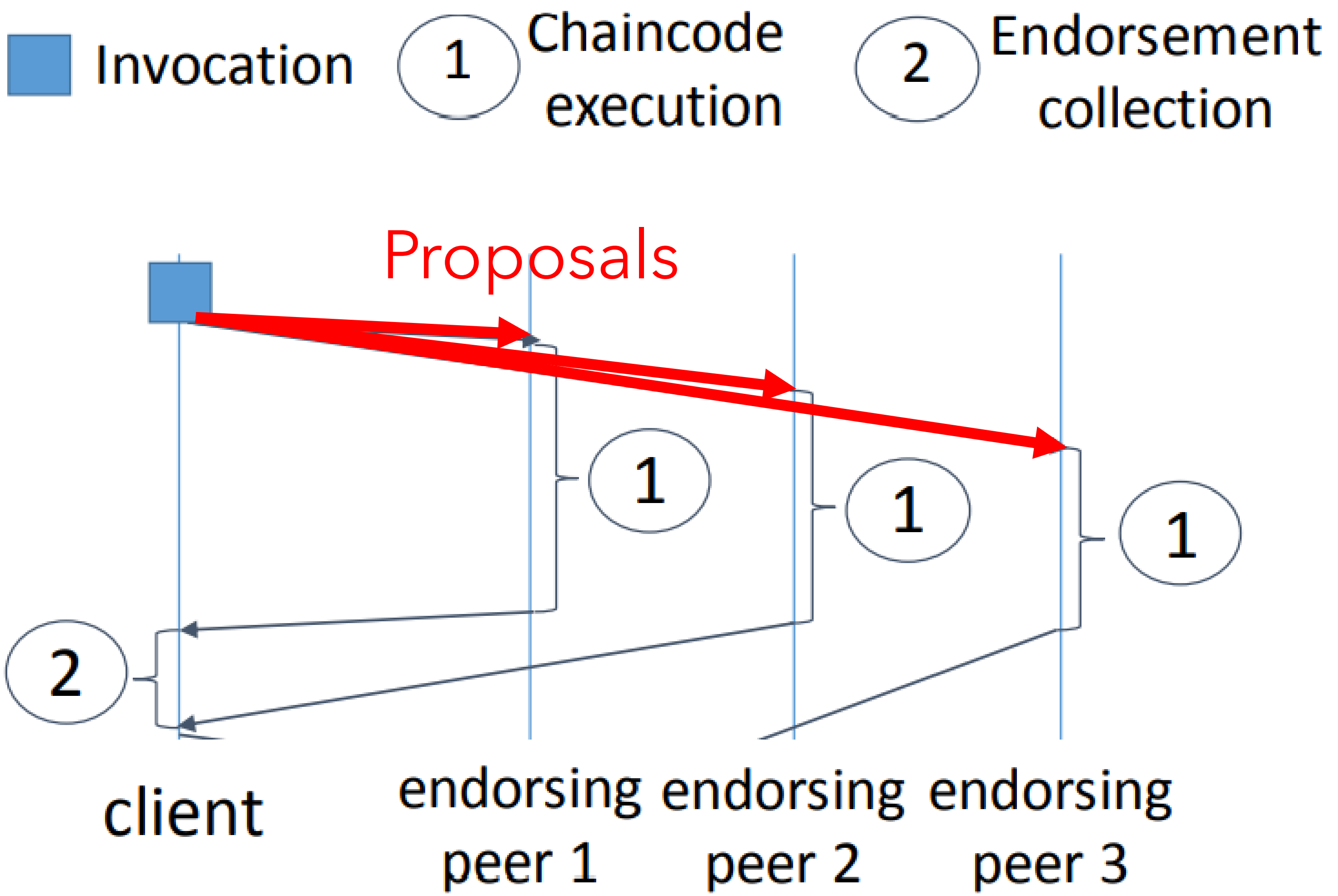
Figure 4: Fabric high level transaction flow.

(for a single channel)

Phase 1: Execution



Phase 1: Execution



Endorsement policy specified by chaincode

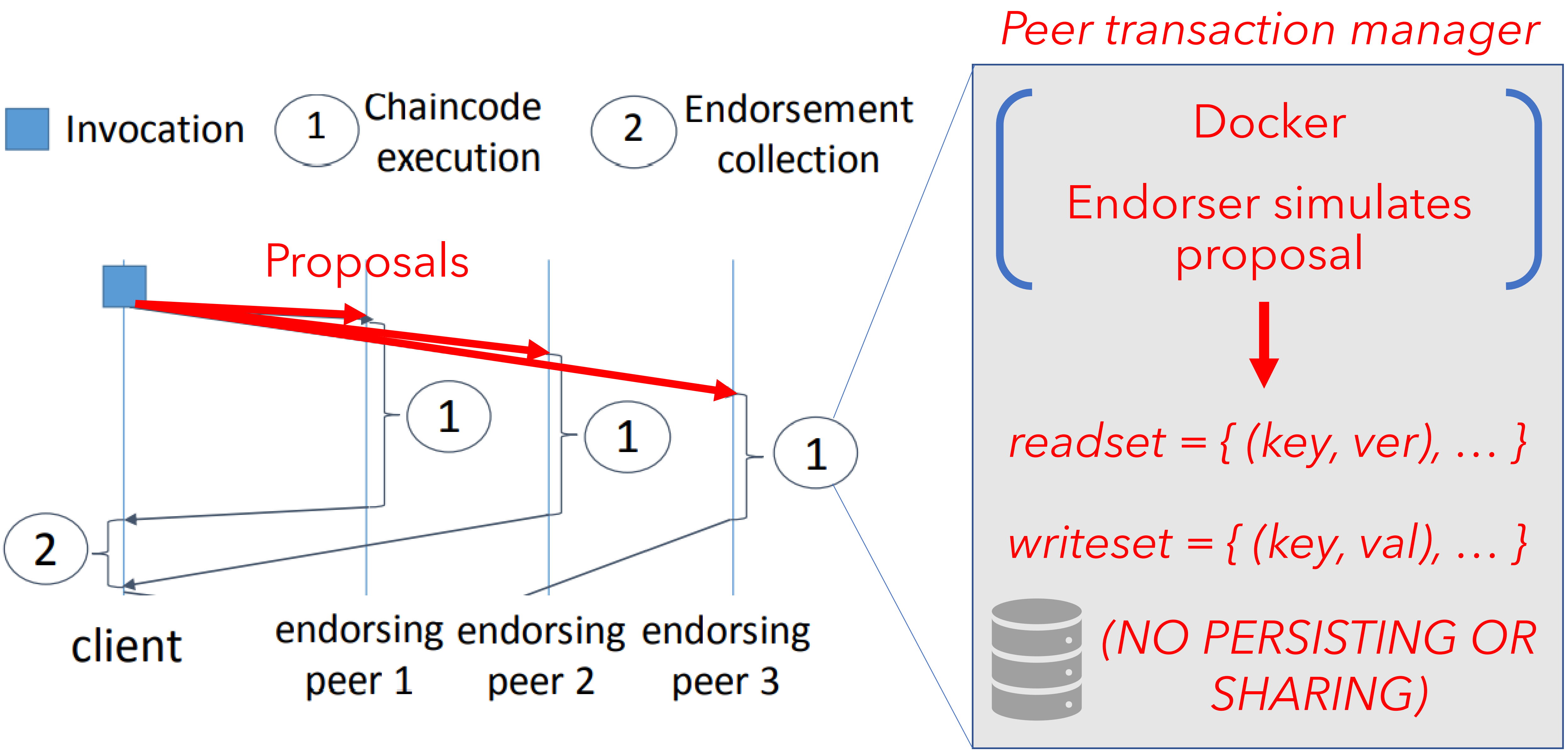
Example:

Send to P1-P3.

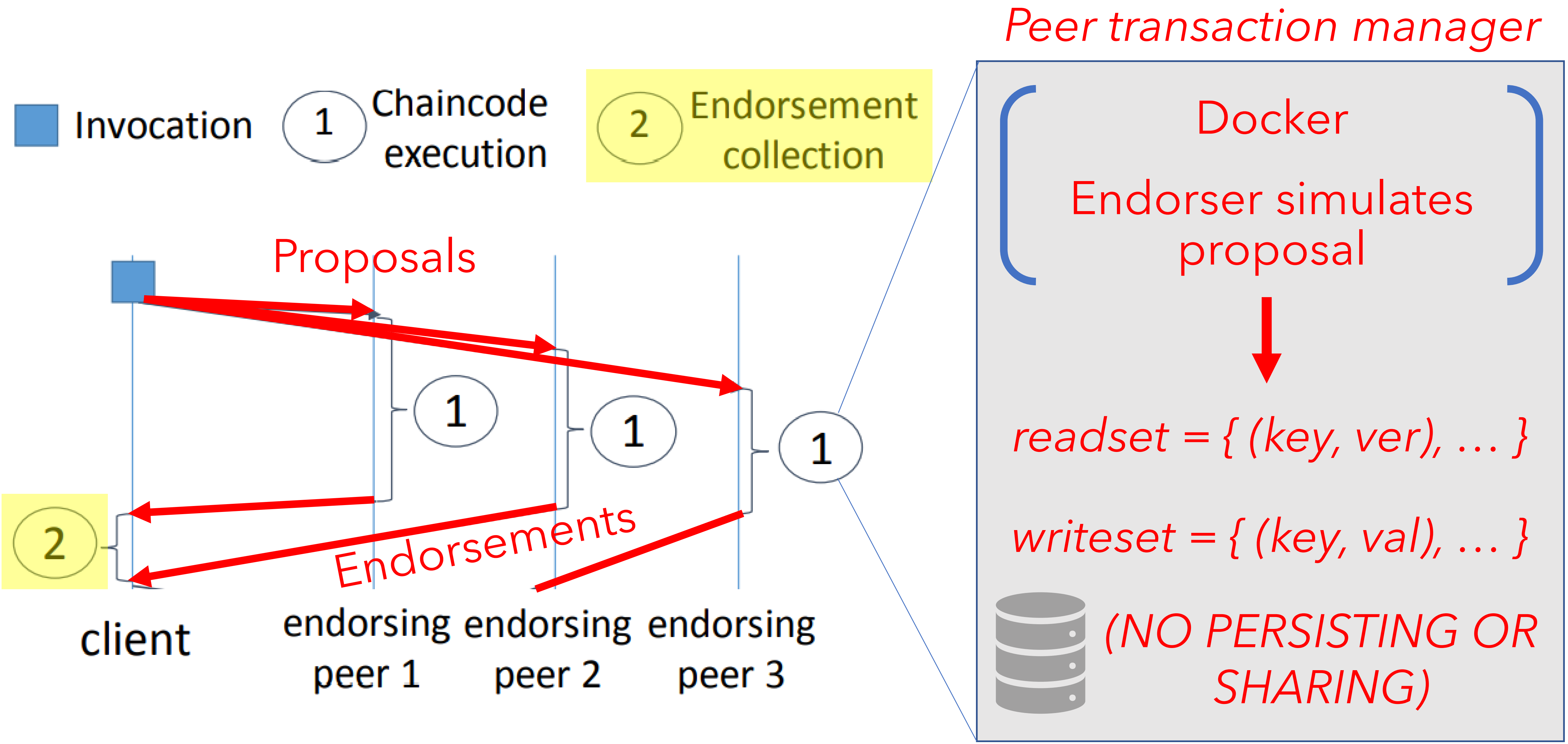
Valid if endorsed by

(P1 AND P2) OR P3.

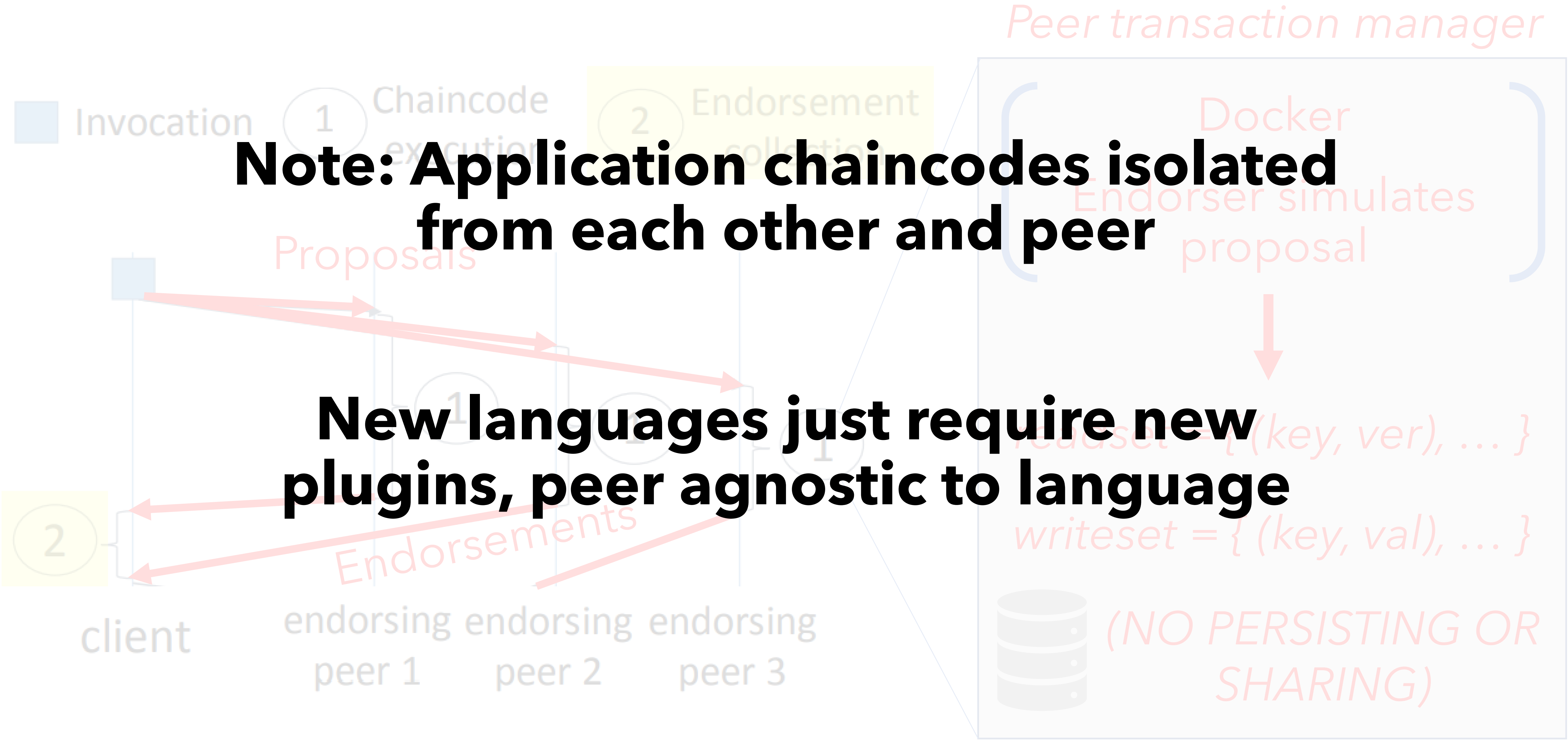
Phase 1: Execution



Phase 1: Execution



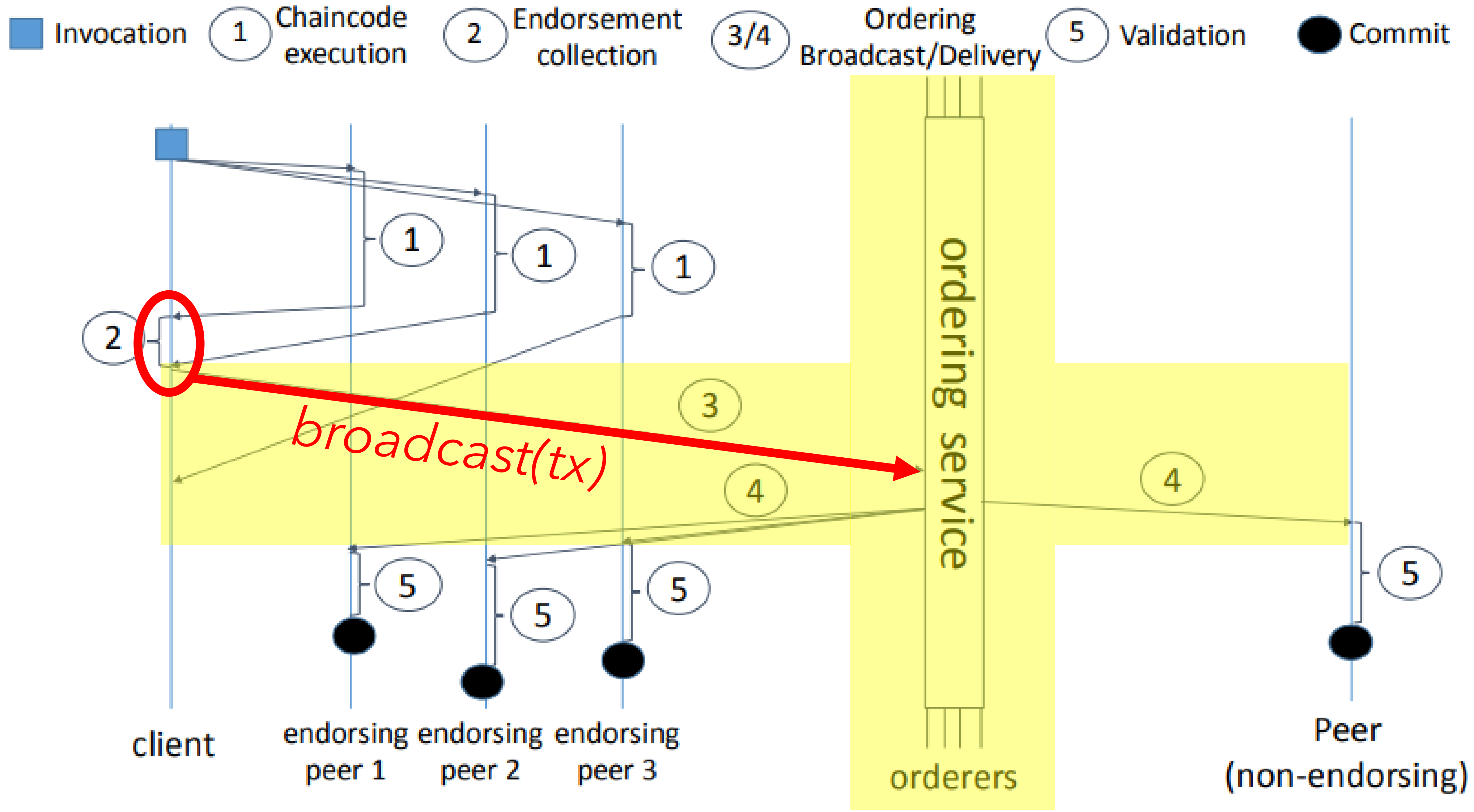
Phase 1: Execution



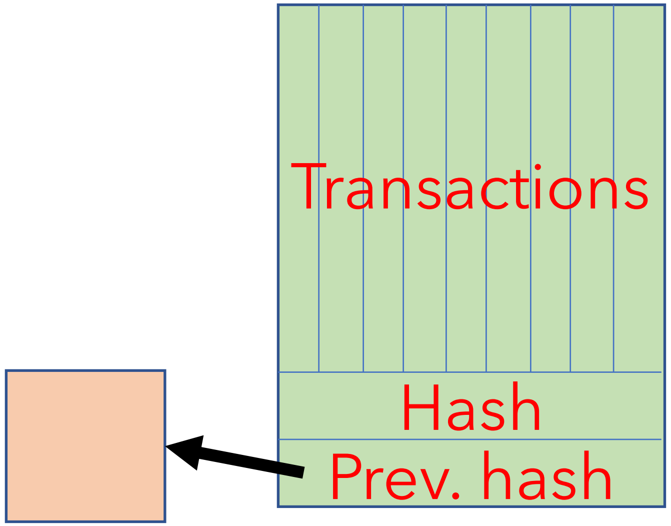
Note: Application chaincodes isolated from each other and peer

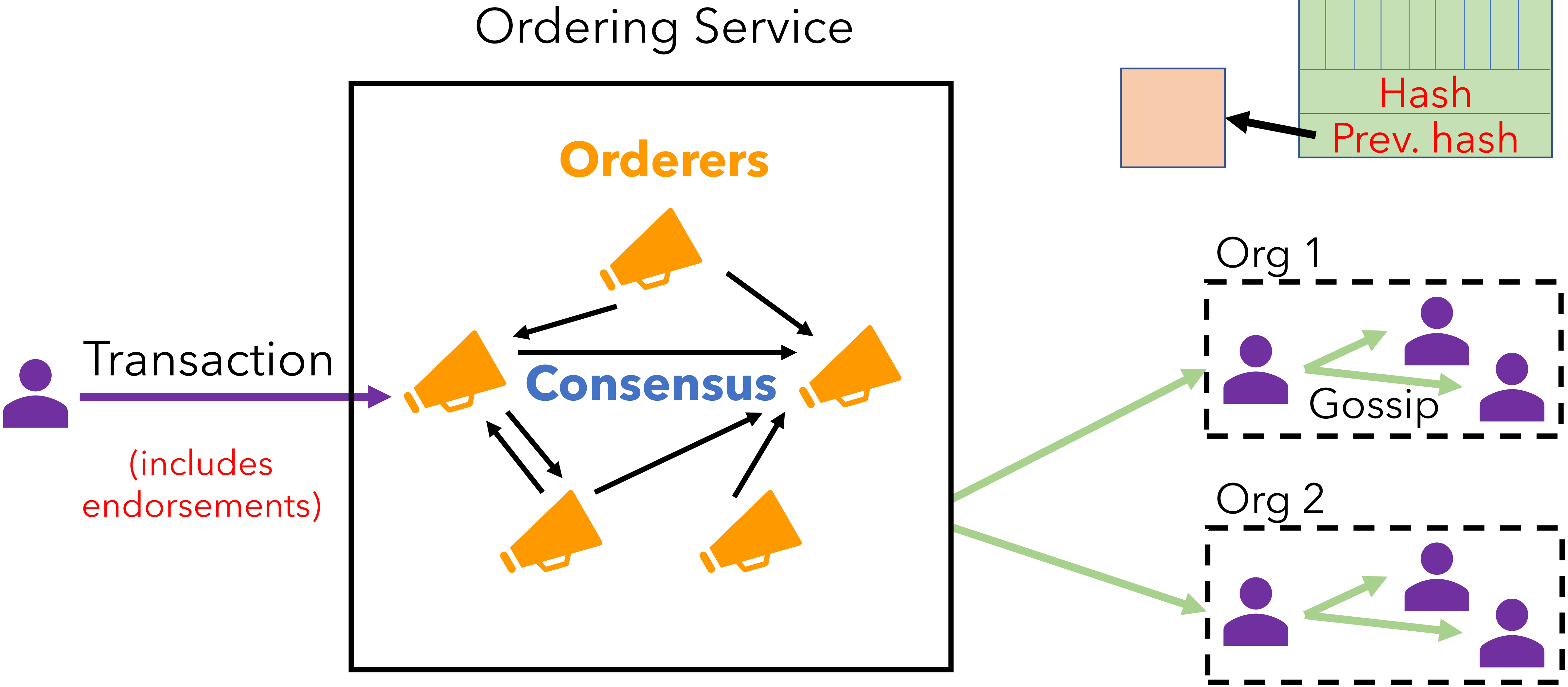
New languages just require new plugins, peer agnostic to language

Phase 2: Ordering

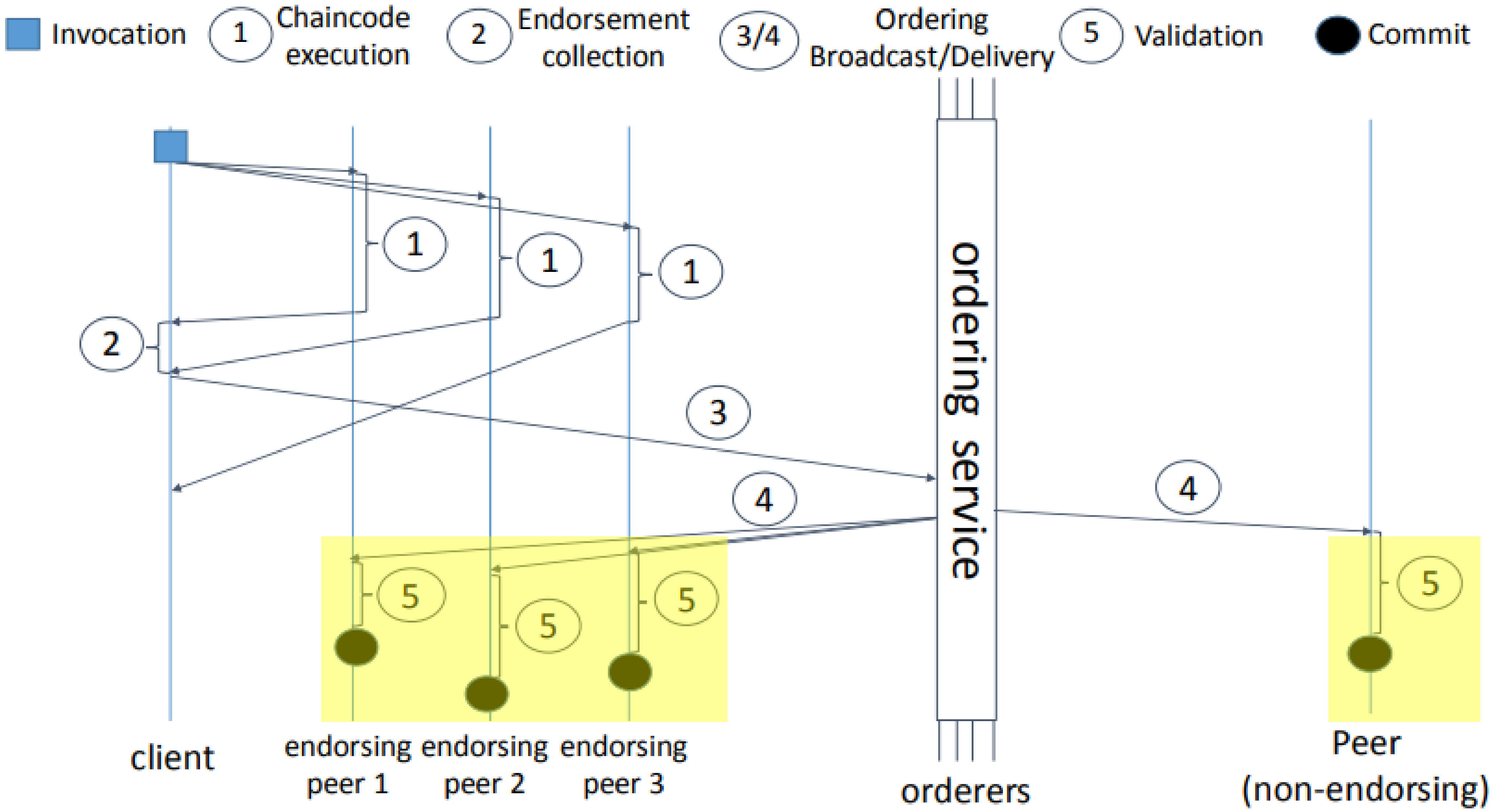


Phase 2: Ordering

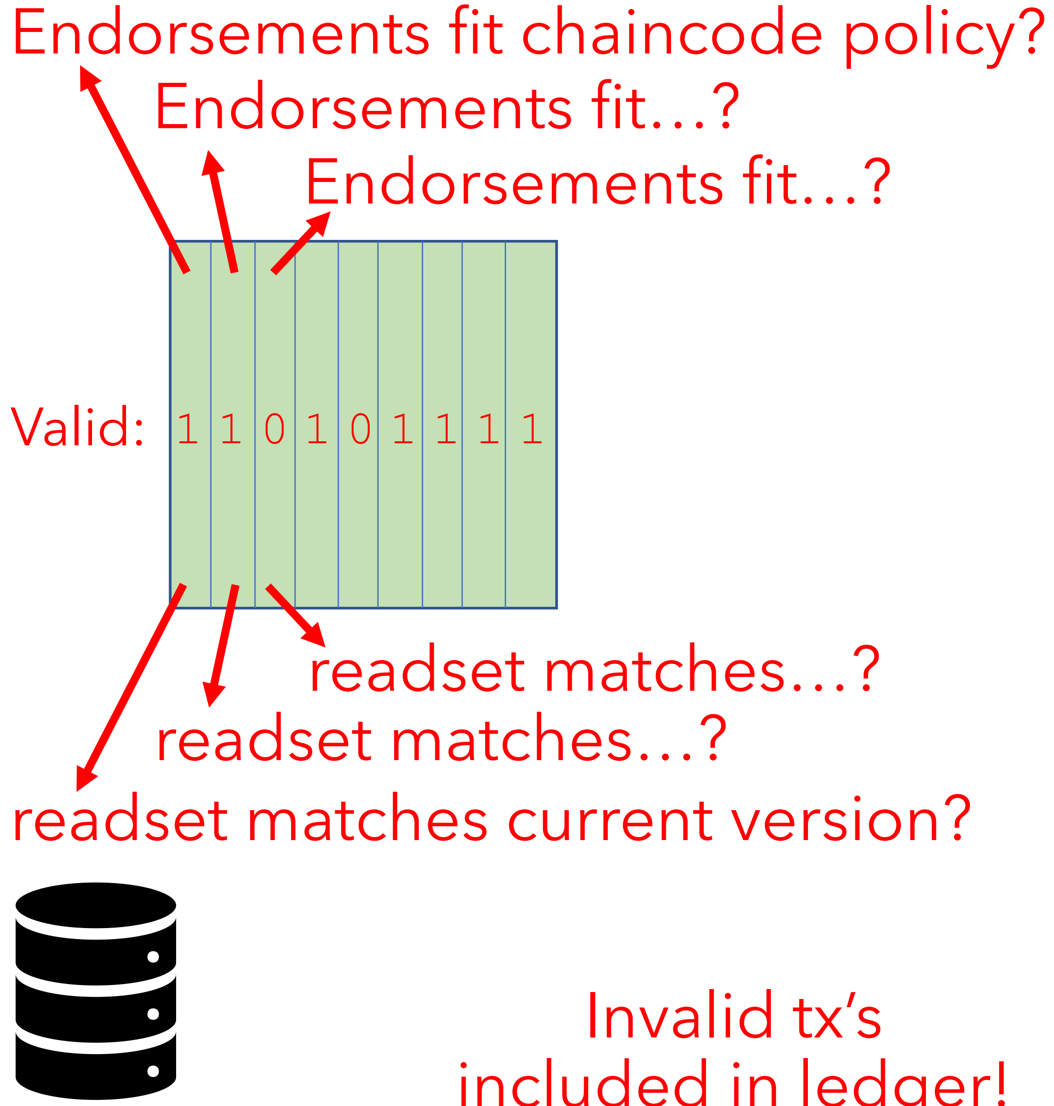




Phase 3: Validation



Phase 3: Validation



1. Endorsement policy evaluation

   Validation system chaincode (VSCC)

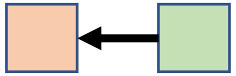
2. R/W conflict check

   Sequentially for all tx's in block, compares readset with KVS

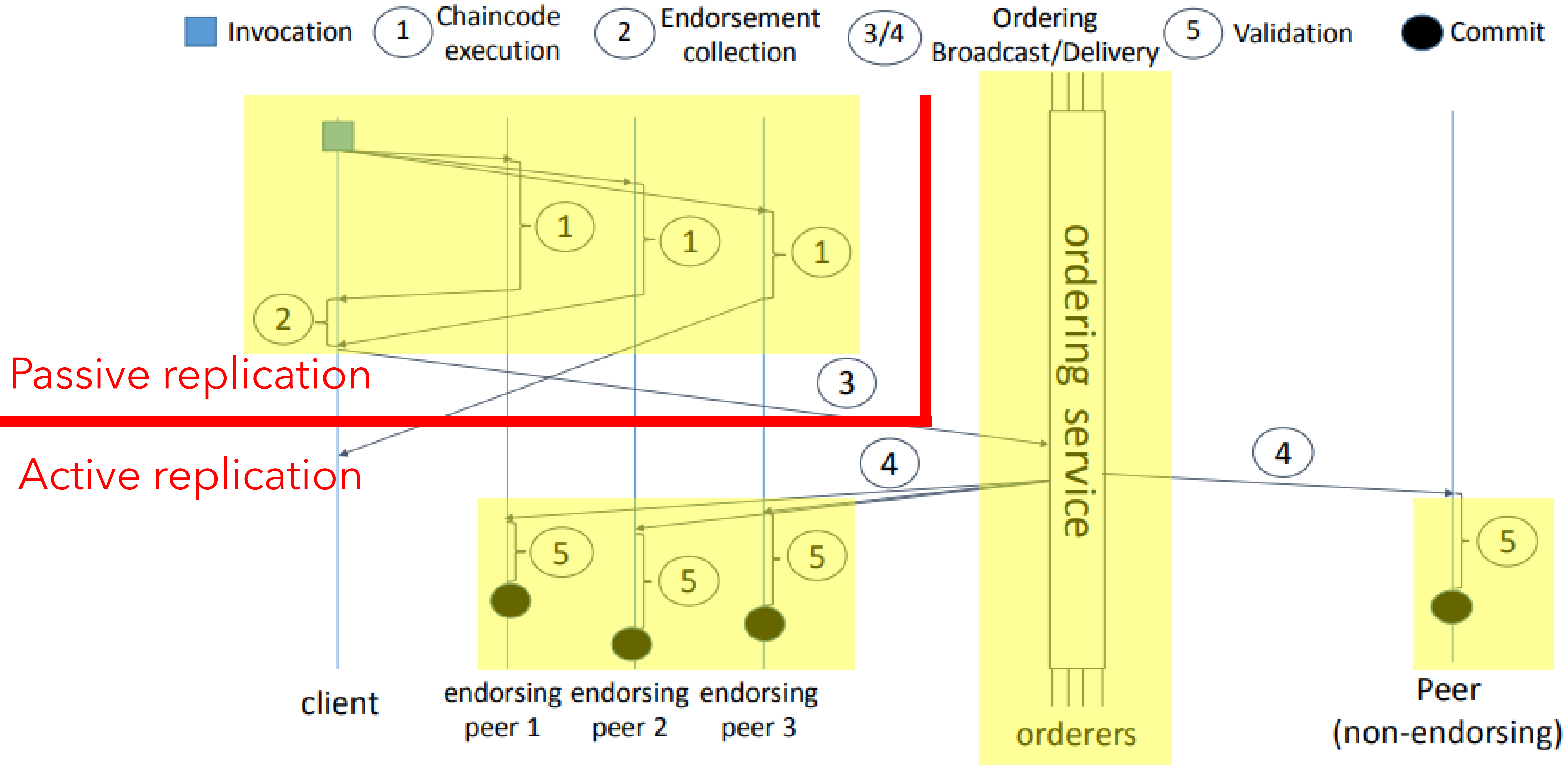
3. Ledger update

   Append block, apply writeset to KVS for valid tx's, store Steps 1-2

Invalid tx's included in ledger!

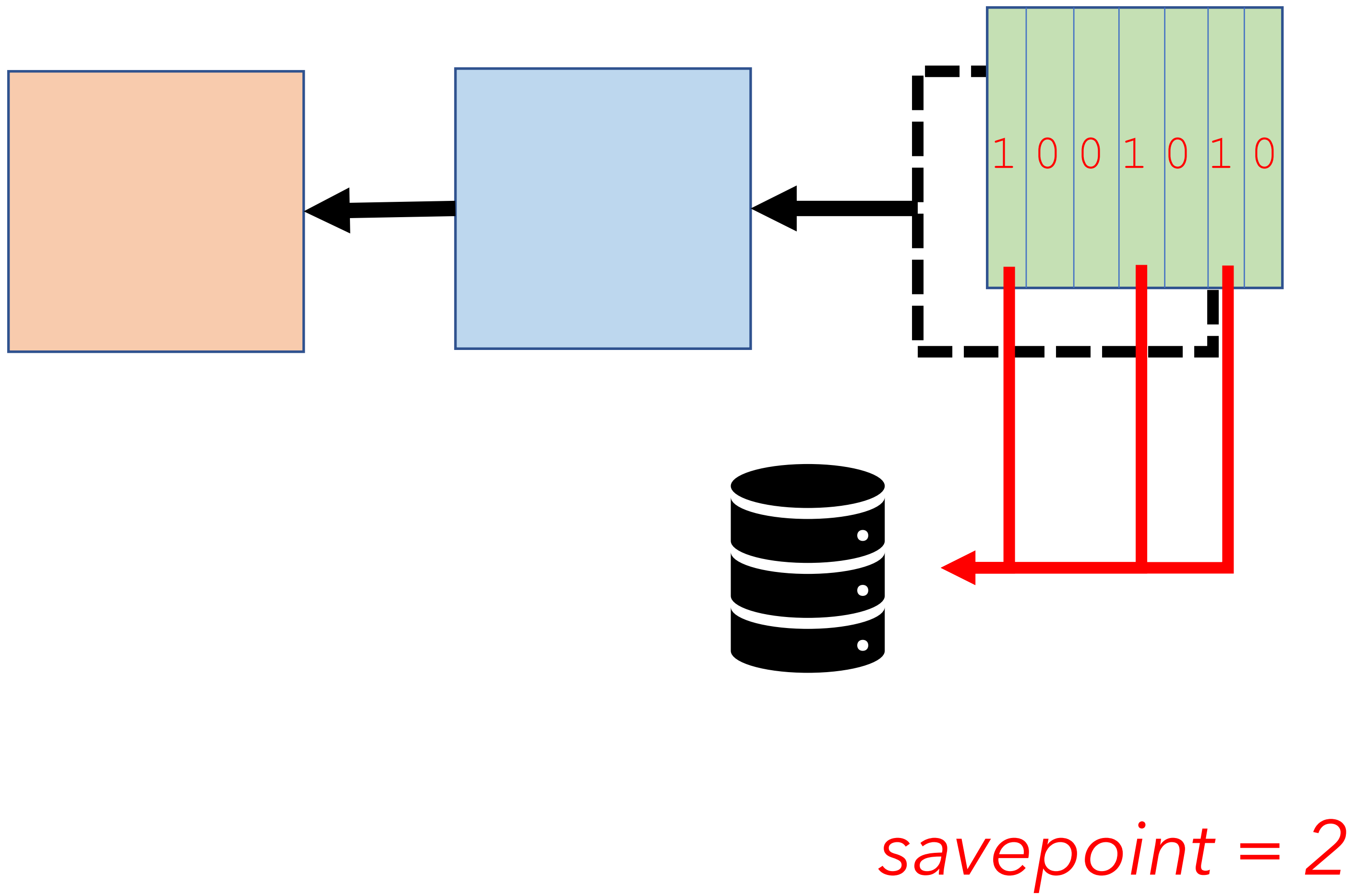


Execute-Order-Validate



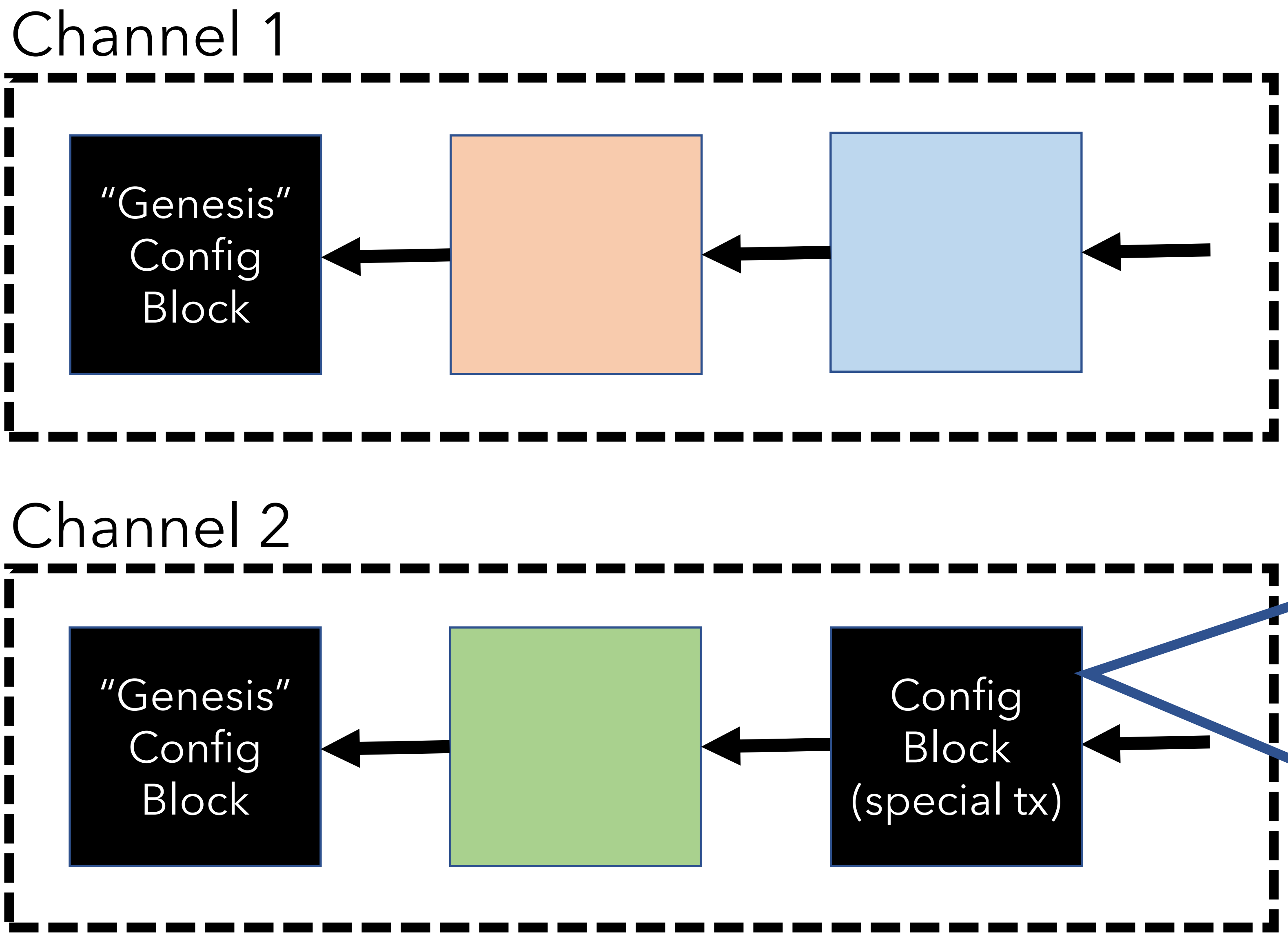
Filling In Some Details

Ledgers tolerate peer crashes.



1. Write block to persistent ledger
2. Apply *writeset* of valid transactions to versioned KVS
3. Compute and persist *savepoint* = largest successfully-applied block #

Configuration is baked into the ledger.



- MSP definitions
- Orderer addresses
- Ordering service / consensus config (batch size, timeouts, etc.)
- Ordering API access rules
- Config modification rules

Fabric has its own special chaincodes.

System Chaincodes (both customizable)

- Endorsement system chaincode (ESCC)

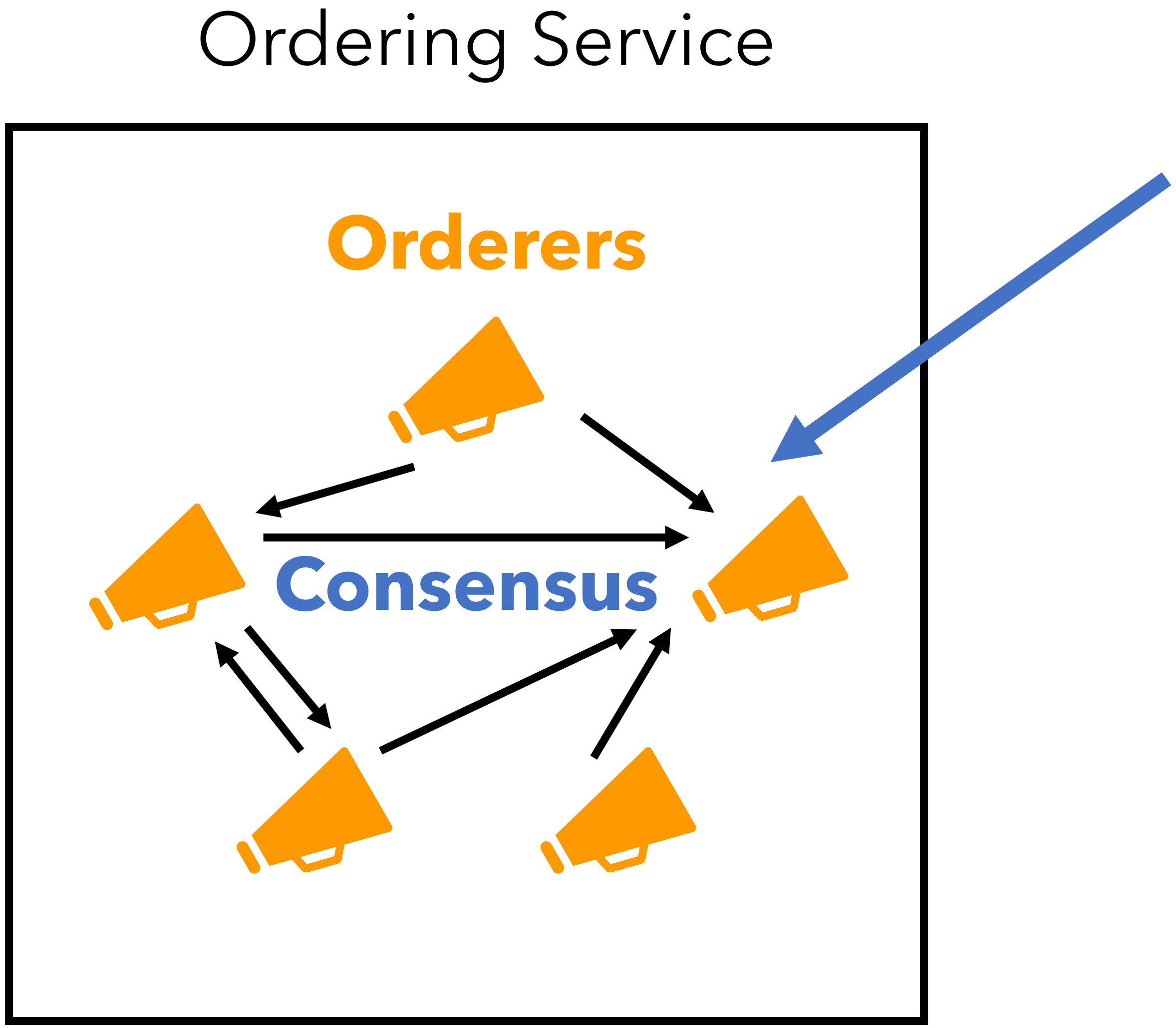
  ESCC(proposal, simulation results) → results, endorsement

- Validation system chaincode (VSCC)

  VSCC(tx) → validity bool

Run directly on peer outside of Docker

Applications have independent trust/fault models.



Single-node, CFT cluster, BFT cluster…

Application models are independent: chaincode endorsement policy

Evaluating Fabric is difficult.

Performance depends on…

choice of distributed application and transaction size

network parameters and topology

network dynamics

ordering service and consensus implementation and parameters

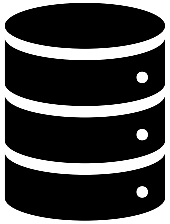
node hardware

number of nodes and channels

… and more config parameters

Fabcoin: Bitcoin-Inspired Fabric Coin

"coin states"



```
(txid_#: (amt, owner))
tx0_0: ($100, Manos)
tx5_2: ($20, Manos)
tx5_3: ($50, Leslie)
```

Existence = unspent

Delete when spent

UTXO

Unspent Transaction Output

Transactions:

- MINT: *request* = (*centralBankID, outputs, sigs*)
  outputs = coin states to create

- SPEND: *request* = (*inputs, outputs, sigs*)
  inputs = list of coin states to spend (delete)
  outputs = coin states to create

Fabcoin: Bitcoin-Inspired Fabric Coin

Chaincode:

```
SPEND_request(inputs, outputs, sigs):
     verify sigs;
     for (input in inputs):
          GetState(in)   // add to readset
          DelState(in)   // add to writeset
     for (int i = 0; i < outputs.size; ++i):
          PutState(txid_i, outputs[i])   // add to writeset
```

Verification: Check sum(inputs) = sum(outputs), etc.
No need to check double-spending!

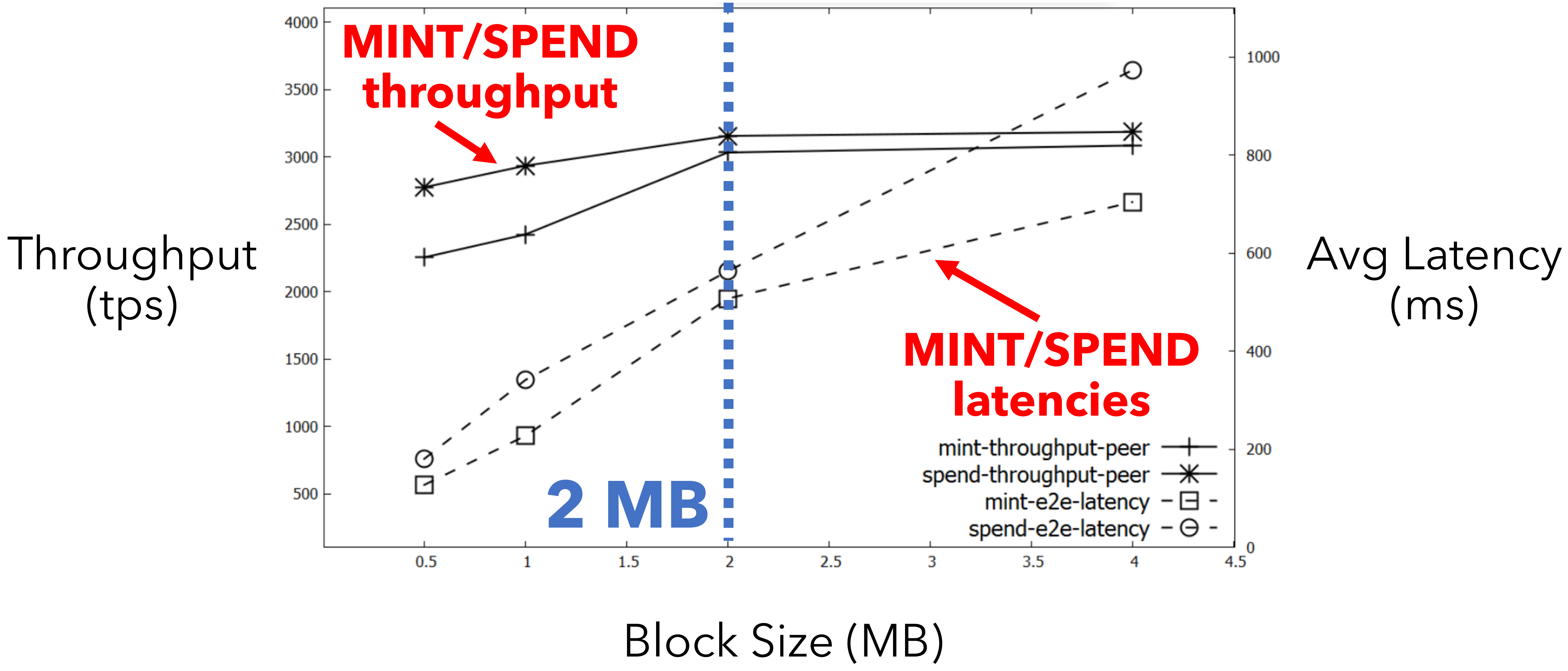
Default Experimental Setup

- Fabric v1.1.0-preview
- IBM Cloud (SoftLayer) Data Center

Nodes:

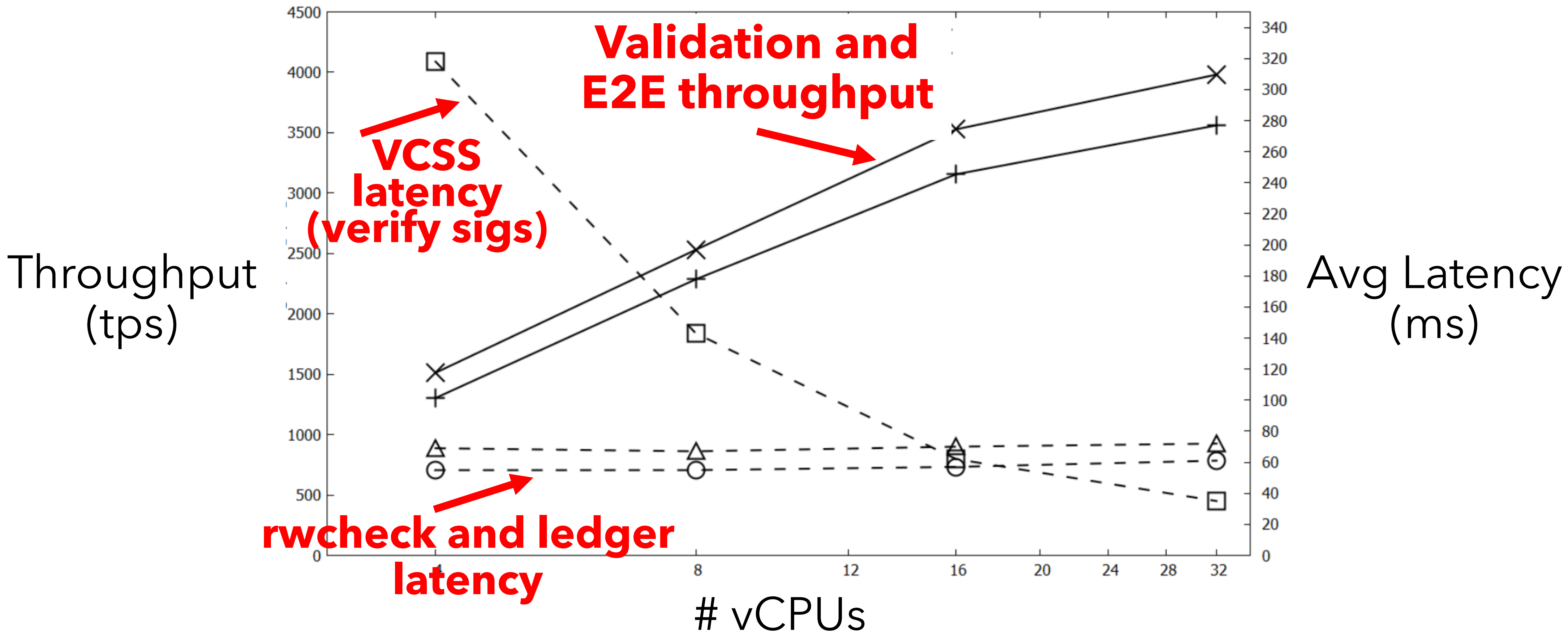
- Dedicated VMs, 1Gbps networking
- 16-vCPU 2GHz dedicated VMs
- Ubuntu, 8GB RAM, SSD local disks
- 3 orderers (all distinct VMs)
- 5 peers (all different orgs, all endorsers)
- 256-bit ECDSA signatures

Experiment 1: Choosing Block Size



Experiment 2: Impact of Peer CPU

SPEND only, Validation Phase only (Ordering wasn't bottleneck)



Experiment 2: Impact of Peer CPU

Conclusion: VSCC is very parallel. Pipeline validation stages, optimize stable-storage access, parallelize dependency checks.

At peak throughput (3560+ tps SPEND) with 32-vCPU, 2MB blocks:

	avg	st.dev	99%	99.9%
(1) endorsement	5.6 / 7.5	2.4 / 4.2	15 / 21	19 / 26
(2) ordering	248 / 365	60.0 / 92.0	[illegible] / 624	523 / 636
(3) VSCC val.	31.0 / 35.3	10.2 / [illegible]	[illegible] / 57.0	113 / 108.4
(4) R/W check	34.8 / 61.5	3.9 / 9.3	[illegible] / 88.5	59.0 / 93.3
(5) ledger	50.6 / 72.2	6.2 / 8.8	70.1 / 97.5	72.5 / 105
(6) validation (3+4+5)	116 / 169	12.8 / 17.8	156 / 216	199 / 230
(7) end-to-end (1+2+6)	371 / 542	63 / 94	612 / 805	646 / 813

Ordering dominates time

MINT/SPEND (in ms)

Sub-second tail E2E (tails from initial load / first blocks)

Experiment 3: SSD vs. RAM Disk

RAM disk (tmpfs) on all peers instead of SSD

(only helps ledger phase of validation)

32-vCPU peer sustained ~3870 SPEND tps (+9% vs. SSD)

Experiment 4: Scalability on LAN

20-100 16-vCPU peers in one data center. 10 endorsers, no gossip

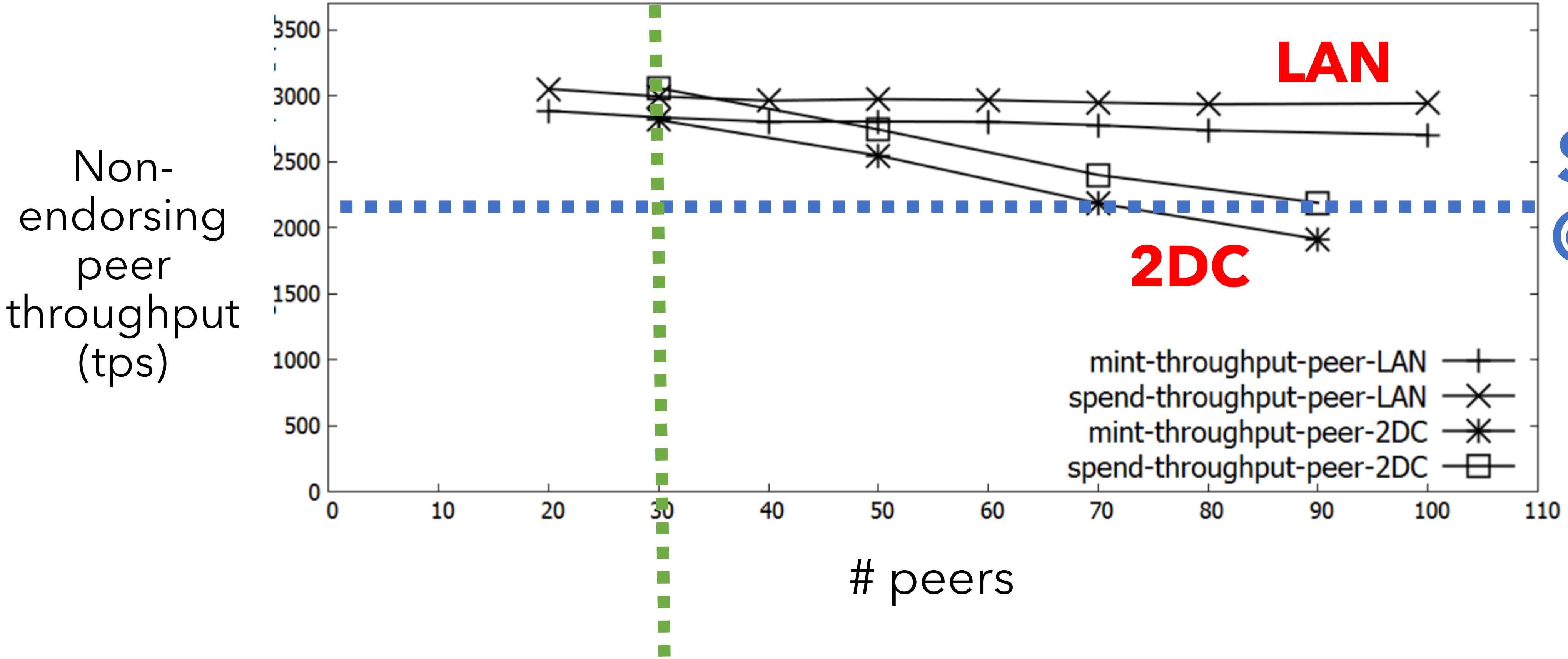
Experiment 5: Scalability Over 2 Data Centers

20-90 16-vCPU peers in 2 data centers (Hong Kong & Tokyo)

Ordering service, all 10 endorsers, and clients in Tokyo.

Non-endorsers in HK

Experiments 4/5: Scalability



2190 SPEND tps @ 90 peers over 2DC

Past 30 peers, orderers' network saturated

Expected LAN drop from orderer network saturation, but IBM Cloud had provisioned higher bandwidth

Experiment 6: Multiple Data Centers

5 data centers (Tokyo, HK, Melbourne, Sydney, Oslo)
20 peers each. Ordering service, 10 endorsers, and clients in Tokyo

Without gossip: 1 peer/org
With gossip: 10 orgs of 10 peers, 2 orgs per data center

Experiment 6: Multiple Data Centers

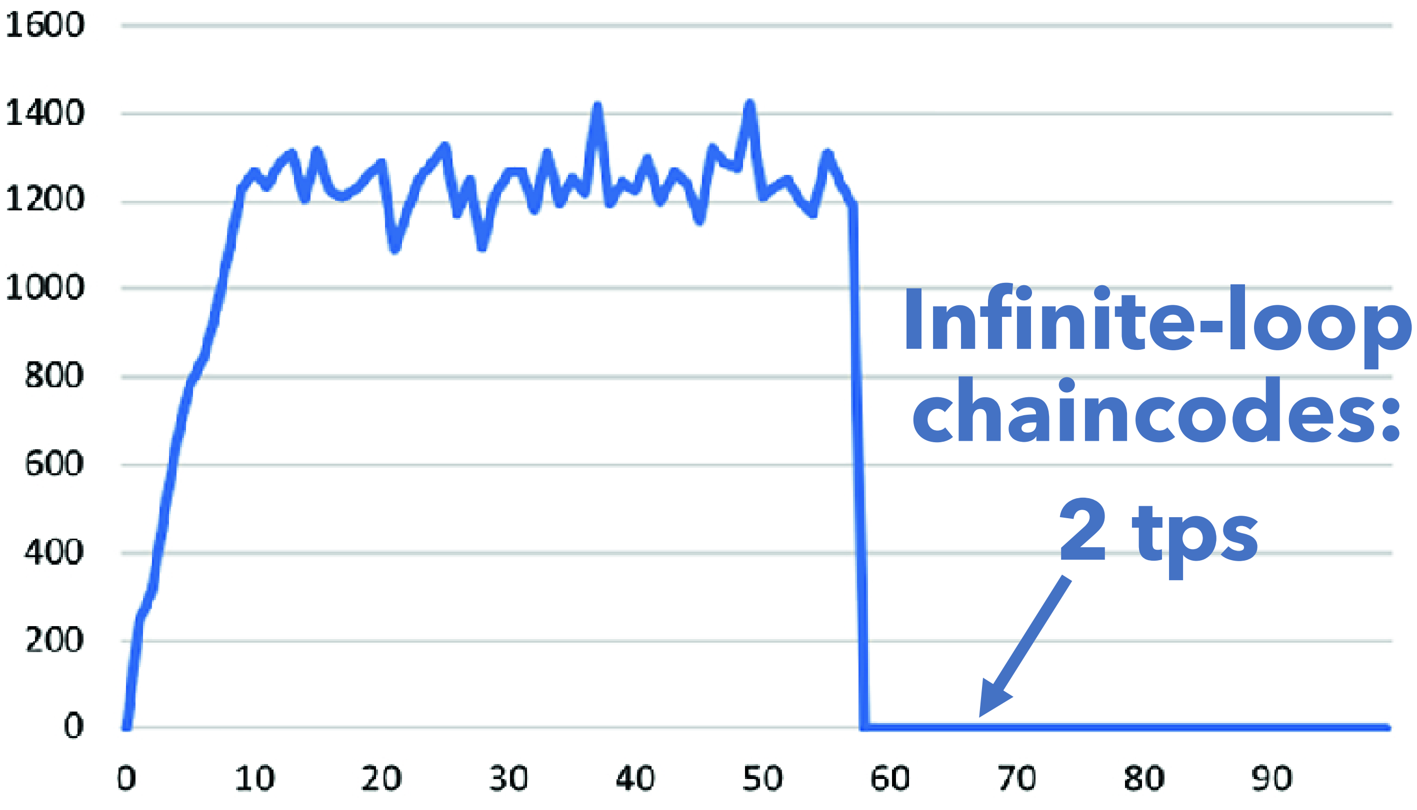
	HK	ML	SD	OS
netperf to TK [Mbps]	240	98	108	54
peak MINT / SPEND throughput [tps] (without gossip)	1914 / 2048	1914 / 2048	1914 / 2048	[illegible]89 / 18[illegible]
peak MINT / SPEND throughput [tps] (with gossip)	2553 / 2762	2558 / 2763	2271 / 2409	[illegible]84 / 20[illegible]

Gossiping helps recover some of the tps lost in transition to more peers / data centers!

Sydney had CPU limitations

Heeere's Mallory!

Transactions committed per second



Time (s)

This graph is Fabric v0.6

Execute-Order-Validate in v1 fixes this ("performs well and… immune")

Wang S. (2019) Performance Evaluation of Hyperledger Fabric with Malicious Behavior. In: Joshi J., Nepal S., Zhang Q., Zhang LJ. (eds) Blockchain – ICBC 2019. ICBC 2019. Lecture Notes in Computer Science, vol 11521. Springer, Cham. https://doi.org/10.1007/978-3-030-23404-1_15

Applications and Use Cases

Foreign exchange netting	Private Fabric channel for each pair of institutions; blockchain resolves non-settling trades, data available in ledger
Enterprise asset management	Track hardware asset life-cycle (mfg., shipping, receiving, customers)
Global cross-currency payment	Process int'l transactions; blockchain records payments + conditions endorsed by participants. Fabric decides settlement method

Conclusion

Fabric is a distributed operating system for permissioned blockchains.

Key features:

Execute-Order-Verify

Transaction execution separated from consensus

Policy-based endorsement

Thank You!

Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains . In EuroSys '18: Thirteenth EuroSys Conference 2018, April 23–26, 2018, Porto, Portugal. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3190508.319053