Principle Foundations of Hyperledger Fabric

Marko Vukolić, IBM Research - Zurich

UC Davis
ECS 189F Introduction to Distributed Ledger
Invited Lecture
November 3, 2020

Hyperledger: A **Linux Foundation** project

- Hyperledger is a collaborative effort created to advance cross-industry blockchain technologies for business
- Founded February 2016 and has since gathered significant cross-industry momentum
- IBM Blockchain Platform is underpinned by technology from the Hyperledger project (in particular, Fabric)
- Open source
  Open standards
  Open governance model

Premier

General

General

Associate

Academia Associate

Source: https://www.hyperledger.org/members
Updated: 24 September 2020

Hyperledger projects

HYPERLEDGER

Distributed Ledgers

HYPERLEDGER BESU
Java-based Ethereum client

HYPERLEDGER BURROW
Permissionable smart contract machine (EVM)

HYPERLEDGER FABRIC
Enterprise-grade DLT with privacy support

HYPERLEDGER INDY
Decentralized identity

HYPERLEDGER IROHA
Mobile application focus

HYPERLEDGER SAWTOOTH
Permissioned & permissionless support; EVM transaction family

Libraries

HYPERLEDGER ARIES

HYPERLEDGER QUILT

HYPERLEDGER TRANSACT

HYPERLEDGER URSA

Tools

HYPERLEDGER AVALON

HYPERLEDGER CACTUS

HYPERLEDGER CALIPER

HYPERLEDGER CELLO

HYPERLEDGER EXPLORER

Domain-Specific

HYPERLEDGER GRID

HYPERLEDGER LABS

What is Hyperledger Fabric?

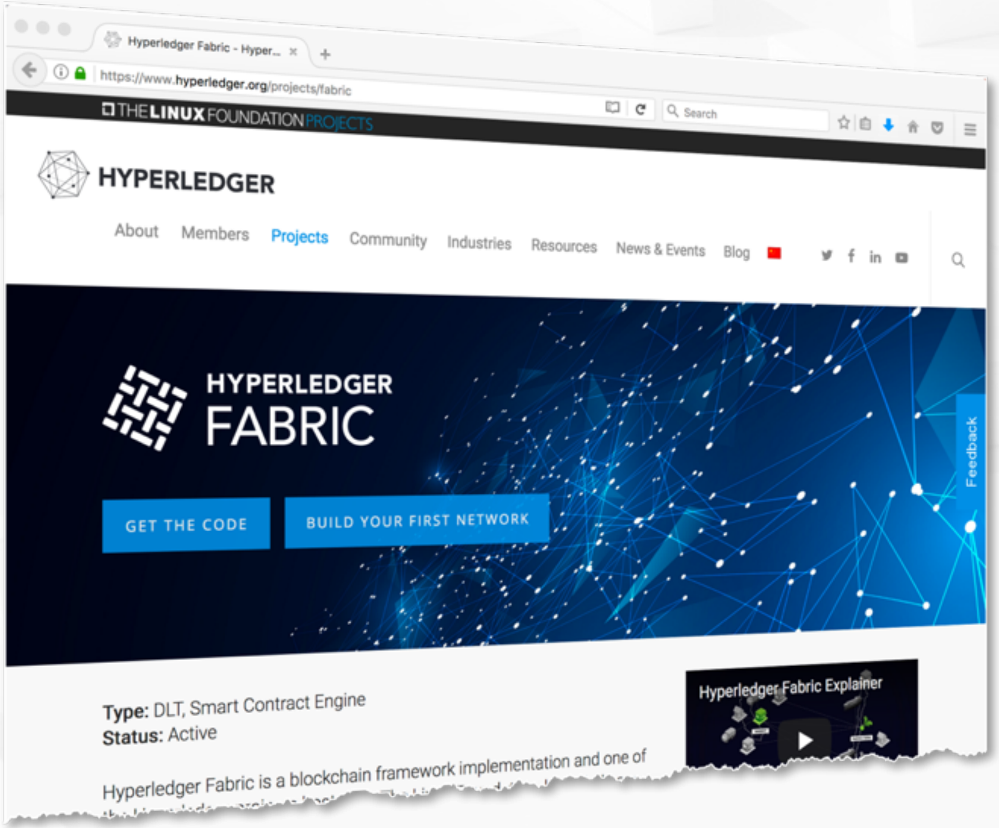
Executive Summary

- Hyperledger Fabric appears to be the platform of choice across all industries: 48% of covered projects that are used in production have chosen Hyperledger Fabric as the core protocol framework underlying the network, followed by R3's Corda platform (15%) and Coin Sciences' MultiChain framework (10%).

https://www.jbs.cam.ac.uk/fileadmin/user_upload/research/centres/alternative-finance/downloads/2019-ccaf-second-global-enterprise-blockchain-report.pdf

HYPERLEDGER
FABRIC

a Distributed Operating System for Permissioned Blockchains



- Foundation for developing general-purpose blockchain applications in general-purpose programming languages
- Emphasis on consensus modularity, confidentiality, resiliency, scalability, smart-contract programmability.
- **V1.0 released June 2017**
- V1.4 LTS released January 2019
- V2.0 was released January 2020
- **V2.2 LTS released July 2020**
- **Apache 2.0 license**
- 159 developers from 27 organizations
- IBM is one of the many contributing organizations
- https://github.com/hyperledger/fabric

Hyperledger Fabric powers IBM Blockchain

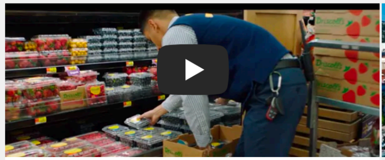
IBM Blockchain | Learn | Platform | Services | Solutions | Industries | Ecosystem

IBM Blockchain Platform: the next generation of blockchain for business

Proven, flexible and built to run on any cloud. Deploy the leading Hyperledger Fabric platform in the environment that's right for your enterprise.

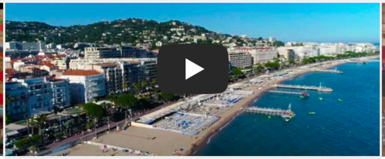
The IBM Blockchain Platform is reshaping industries

The race to reinvent the world is on. What disruption will you create?



Food supply

IBM Food Trust™ is the only blockchain network of its kind connecting growers processors, distributors, and retailers through a permissioned, permanent, and shared record of food system data.



Media and advertising

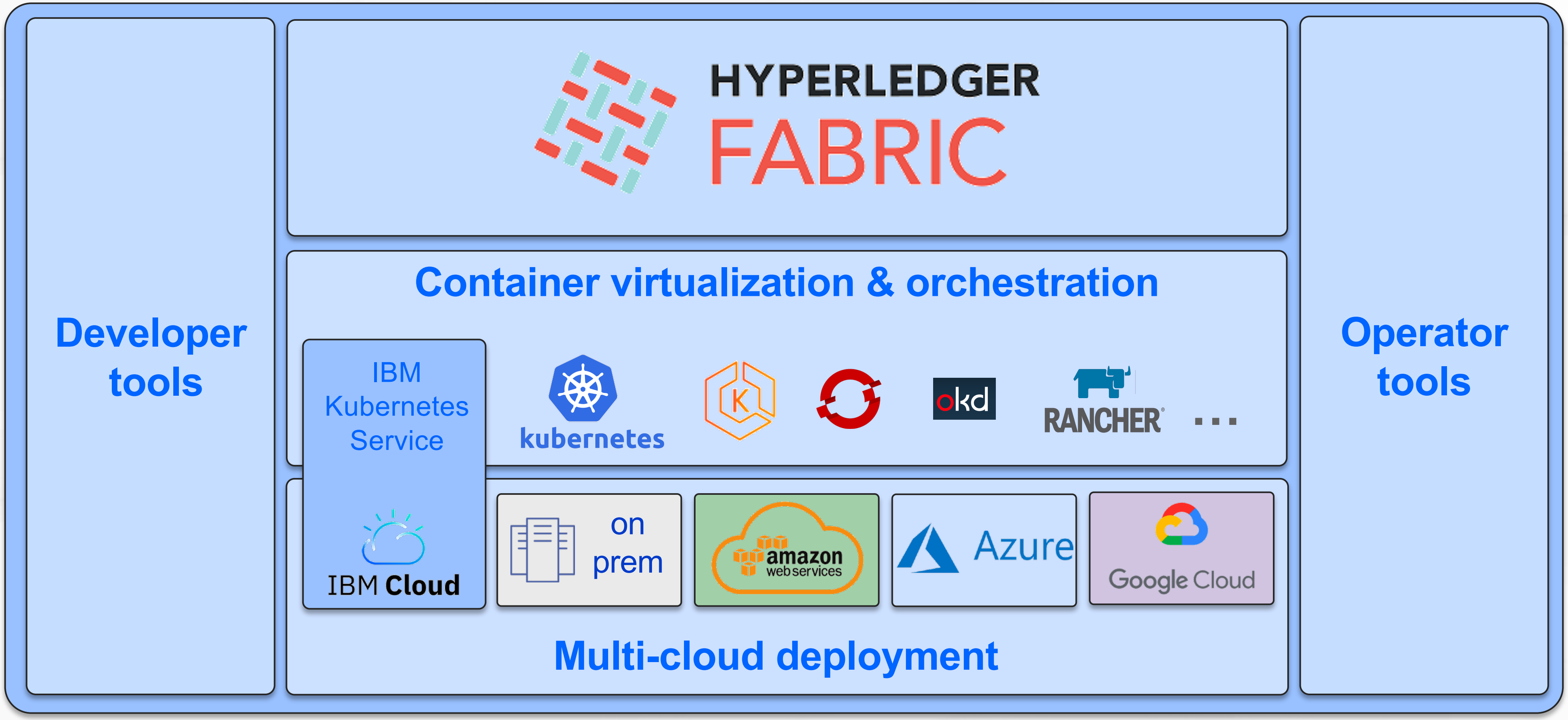
Online advertising fraud costs companies billions of dollars annually. Learn how Mediaocean is revolutionizing the media and advertising industry with the IBM Blockchain Platform and IBM Garage.

Trade finance

Thirteen European banks have collaborated on we.trade, a blockchain network that's transforming trade finance — and even trade itself — for small- and medium-sized buyers and sellers.

Introducing **IBM Blockchain Platform**

Build and operate Hyperledger Fabric networks



Advanced tooling
Create & manage smart contracts, applications & networks

Open technology
Hyperledger Fabric, Containers, Kubernetes

Deploy anywhere
Comprehensive cloud & on-premises options

IBM is making blockchain real for business with active networks spanning most industries

Unlisted Securities

Provenance

Bank Guarantees

Trade Finance

Clearing & Settlement

Healthcare

IBM Food Trust™

Identity

Insurance

Government

Distributed Energy

IBM **Blockchain**

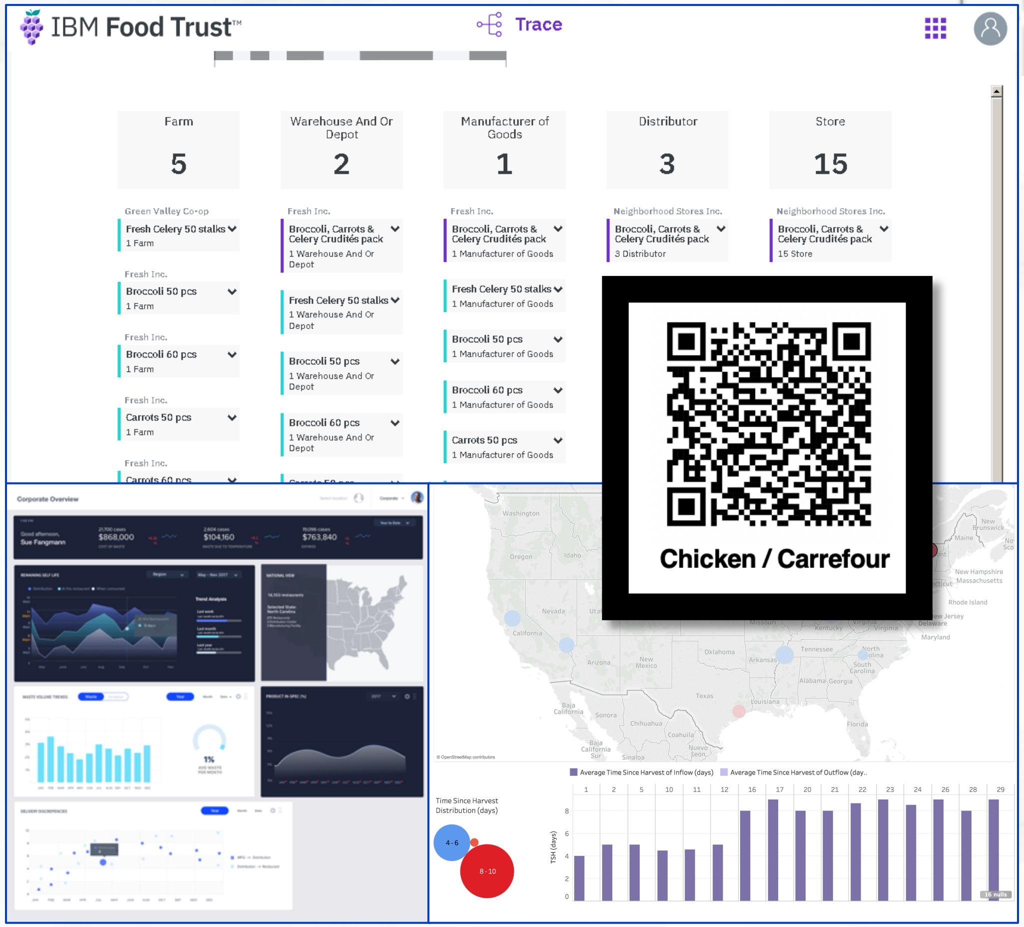
Blockchain **Transparent Supply**: IBM Food Trust

Problem

- Product information is siloed across the supply chain
- Product recalls often take weeks, and often performed manually
- Managing inventory (shelf-life, expiry date, product rotations, etc) is a challenge due to lack of pertinent information integrated with products

Solution

- Food products , including their transformation, are linked across the supply chain using GS1 data standards
- Suppliers can link or embed useful information into products
- Shoppers can trace quality and origin of products from production through distribution by a QR code scan



Benefits & Implications

- Near-instant traceback of products to their origin allow for building consumer trust in products, surgical recalls, and distribution maps
- Inventory optimization using accurate inventory positions across the supply chain with product information
- With the full trace data, there are opportunities for new analytics and insights for supply chain optimization

(Animated) Demo: **IBM Food Trust**

Further examples by (selected) industry

Financial

- Trade Finance
- Cross currency payments
- Mortgages
- Letters of Credit

Public Sector

- Asset Registration
- Citizen Identity
- Medical records
- Medicine supply chain

Retail

- Supply chain
- Loyalty programs
- Information sharing (supplier – retailer)

Insurance

- Claims processing
- Risk provenance
- Asset usage history
- Claims file

Manufacturing

- Supply chain
- Product parts
- Maintenance tracking

… and COVID-19 related use cases: IBM Digital HealthPass

ZDNet

CENTRAL EUROPE MIDDLE EAST SCANDINAVIA AFRICA UK ITALY SPAIN MORE NEWSLETTERS ALL WRITERS

MUST READ: Quantum computers are coming. Get ready for them to change everything

IBM's Watson Health launches IBM Digital Health Pass app

The app is under control of the individual and uses blockchain to verify everything from health data to COVID-19 test results.

powered by

HYPERLEDGER FABRIC

IBM Digital HealthPass balances the need to present **health status** for access **with privacy**
https://www.ibm.com/products/digital-health-pass

- Covid-19 test and overall health status is only accessible on personal devices of Users
- Users devices hold Health Passports and, therein, Health Credentials issued by approved Issuers
- Information about approved Issuers (their public keys) registered on the blockchain
- Users present Health Credentials (in the form of a QR code) to Verifiers to obtain physical access
- Verifiers can apply the appropriate policy to Users based on whether or not they are equipped with a valid and authentic Health Credentials, accessing information about Issuers stored on the blockchain
- **No Health Certificate or PII is ever stored on the blockchain (GDRP, HIPAA compliance)**

IBM

HYPERLEDGER FABRIC

IBM

The 2018 Eurosys paper described the revolutionary v1 architecture

<EURO/SYS'18>

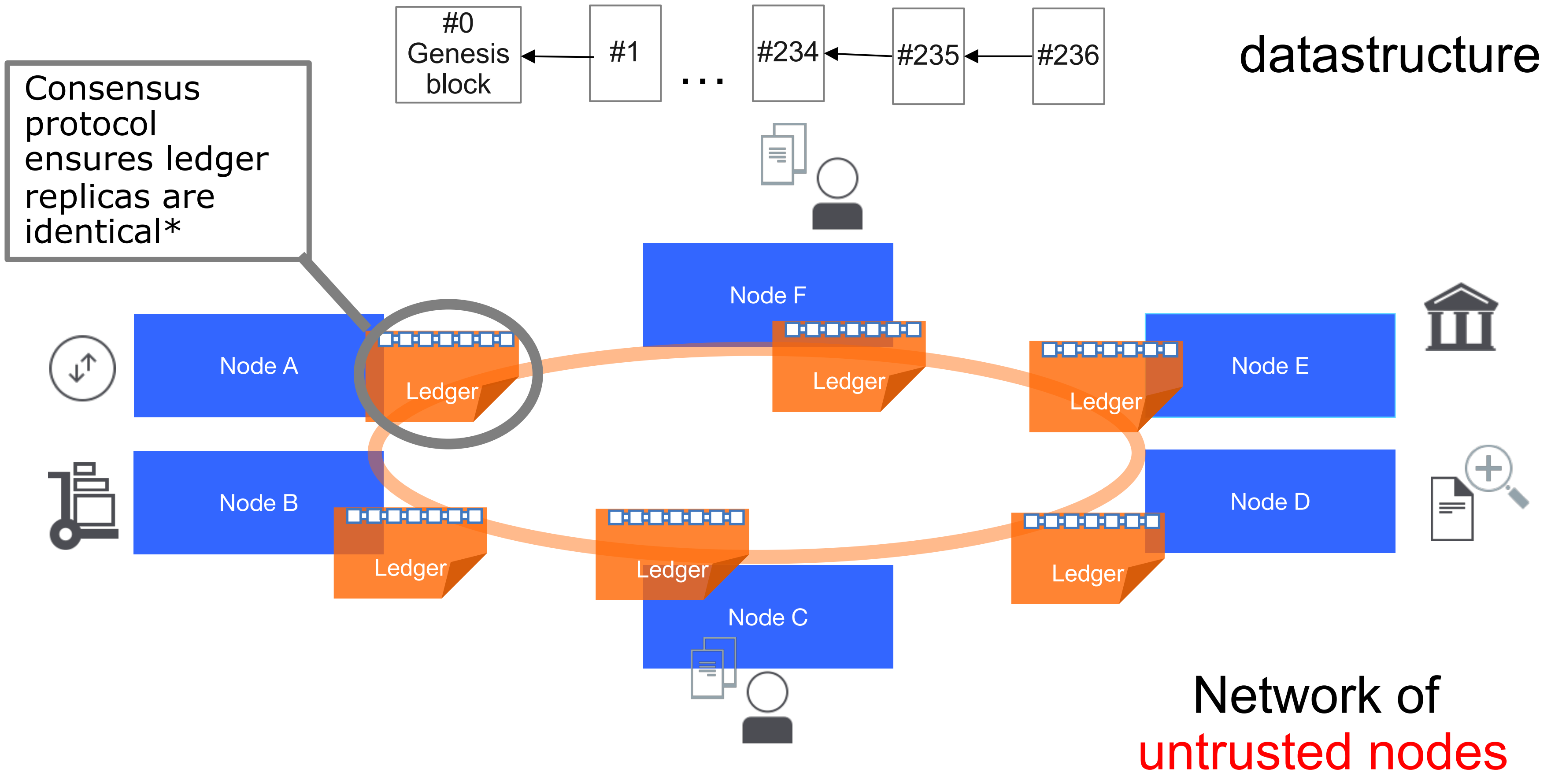
https://dl.acm.org/doi/10.1145/3190508.3190538 ~1300 citations since April 2018, university courses…

Fabric v1 enabled, for the first time:

- A blockchain system that **allows blockchain applications (smart contracts) to be written in general-purpose programming languages (e.g., Go, Java)** without being susceptible to security vulnerabilities and code nondeterminism
- Addressed system-level challenges related to **eliminating native cryptocurrencies** from blockchains
- Enabled **modular distributed consensus and network membership services**
- Introduced, to this end, a **novel Execute – Order – Validate architecture for blockchains**
- **Excellent performance** for a variety of blockchain applications

What is a Blockchain?

- **A chain (sequence, typically a hash chain) of <u>blocks</u> of transactions**
  - Each block consists of a list of transactions
  - Blockchain establishes total order of blocks (and hence, transactions)



Blokchain transactions and distributed applications

- **Bitcoin transactions**
  - simple virtual cryptocurrency transfers
  - transfer BTC from account to account

- **Transactions do not have to be simple nor related to cryptocurrency**
  - Distributed applications
  - smart contracts (Ethereum) or chaincodes (Hyperledger Fabric)

A smart contract is an ***event driven program, with state****,*

which runs on a ***replicated, shared ledger*** *[Swanson2015]*

"Smart contract" → (replicated) state machine

Are Blockchains the same as SMR?

SMR = State-Machine Replication [Lamport 78, countless follow-up papers]

Well, not really…

The main difference

SMR approach	**Blockchain smart-contracts**
single **trusted** application	Multiple applications **Not** (necessarily) **trusted**! Developed by third party application developers

Blockchain evolution

2009

bitcoin

- **A hard-coded cryptocurrency application**
- **Limited stack-based scripting language**
- **Native cryptocurrency (BTC)**
- **Resource-intensive Proof-of-Work consensus**
- **Permissionless blockchain system**

Blockchain 1.0

2014

ethereum

- **General-purpose blockchain**
- **Distributed applications (smart contracts)**
- **Domain-specific language (Solidity)**
- Native cryptocurrency (ETH)
- Resource-intensive Proof-of-Work consensus
- Permissionless blockchain system

Blockchain 2.0

2017

HYPERLEDGER FABRIC

- General purpose blockchain
- **Distributed applications (chaincodes)**
- **Different general-purpose languages (e.g., golang, Java, Node)**
- **No native cryptocurrency**
- **Modular/pluggable consensus**
- Permissioned blockchain system (geared towards business applications)
- **Designed for multiple instances/deployments**

Blockchain 3.0

Blockchain evolution

2009

bitcoin

- **A hard-coded cryptocurrency application**
- **Limited stack-based scripting language**
- **Native cryptocurrency (BTC)**
- **Resource-intensive Proof-of-Work consensus**
- **Permissionless blockchain system**

Blockchain 1.0

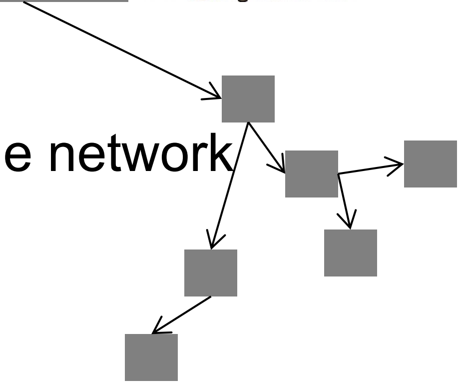
How Bitcoin works (in one slide)

- **Step 1:** PoW block **“mining”**

Miner tasks
- *Validate transactions in the block*
- *Find nonce such that*
  h: hash of Block #237
  h = SHA256(A||B||C) < DIFFICULTY

. . . #234 ← #235 ← #236 ← Block #237

Transactions (payload)
A =hash of block #236
B = Root hash of Merkle tree of tx hashes
C = nonce
Block #237



- **Step 2: Gossip** block #237 across the network

- **Step 3: Validation (at every miner)**
  - *Validating transactions in the block, sequentially*
  - *Verify* hash of Block #237 < DIFFICULTY

#234 ← #235 ← #236 ← Block #237

Transactions (payload)
A =hash of block #236
B = Root hash of Merkle tree of tx hashes
C = nonce
Block #237

bitcoin

Bitcoin energy consumption and performance

- https://digiconomist.net/bitcoin-energy-consumption
- 77 TWh/year → 8~9 GW of power
- More than Switzerland, 0.35% of world electricity consumption
- 741 kWh per transaction!
- 1 transaction can power 25 average US households for a day

- 7 transactions per second peak theoretical throughput
- Latency about 1 hour (1 block on average every 10 minutes, 6 block confirmation)

Blockchain evolution

2009

bitcoin

- **A hard-coded cryptocurrency application**
- **Limited stack-based scripting language**
- **Native cryptocurrency (BTC)**
- **Resource-intensive Proof-of-Work consensus**
- **Permissionless blockchain system**

Blockchain 1.0

2014

ethereum

- **General-purpose blockchain**
- **Distributed applications (smart contracts)**
- **Domain-specific language (Solidity)**
- Native cryptocurrency (ETH)
- Resource-intensive Proof-of-Work consensus
- Permissionless blockchain system

Blockchain 2.0

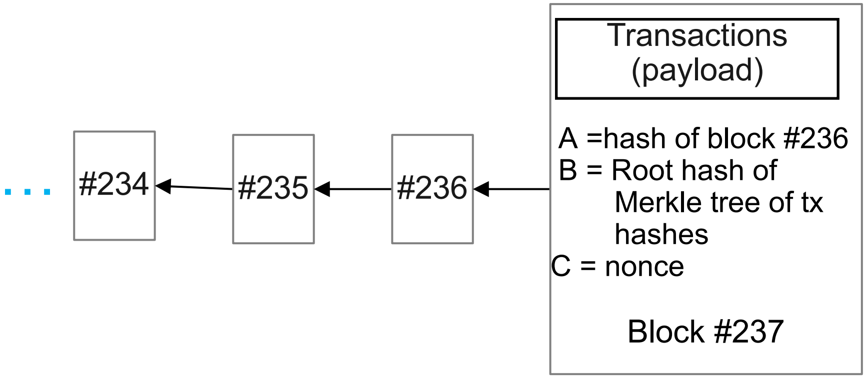
Ethereum

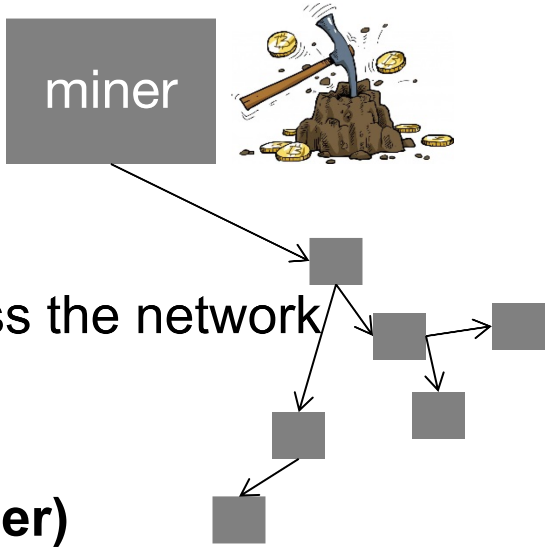
How ~~Bitcoin~~ works (in one slide)

- **Step 1:** PoW block **"mining"**

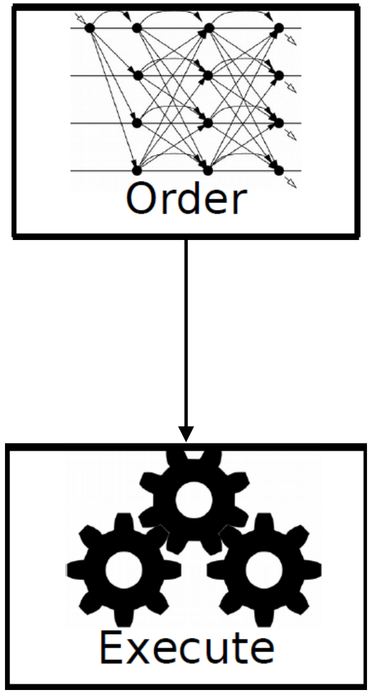
Pre-execute

Miner tasks
- *~~Validate~~ transactions in the block*
- *Find nonce such that*
  h: hash of Block #237
  h = SHA256(A||B||C) < DIFFICULTY





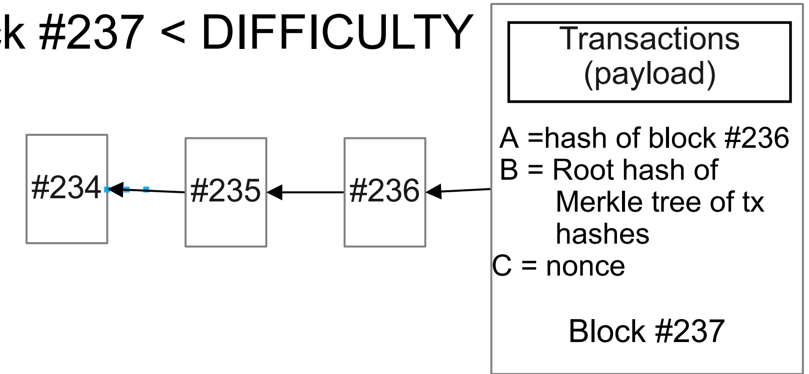
- **Step 2: Gossip** block #237 across the network



Execution

- **Step 3: ~~Validation~~ (at every miner)**

Executing
- *~~Validating~~ transactions in the block, sequentially*
- *Verify* hash of Block #237 < DIFFICULTY

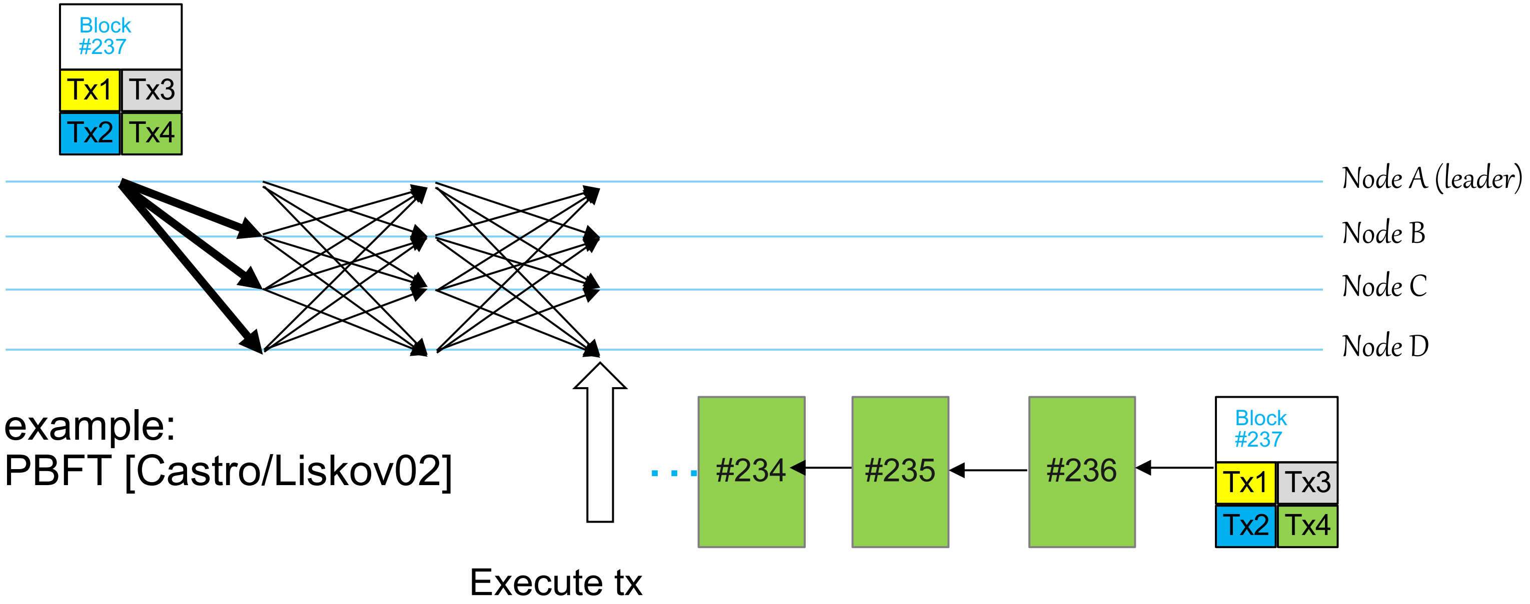


Ethereum energy consumption and performance

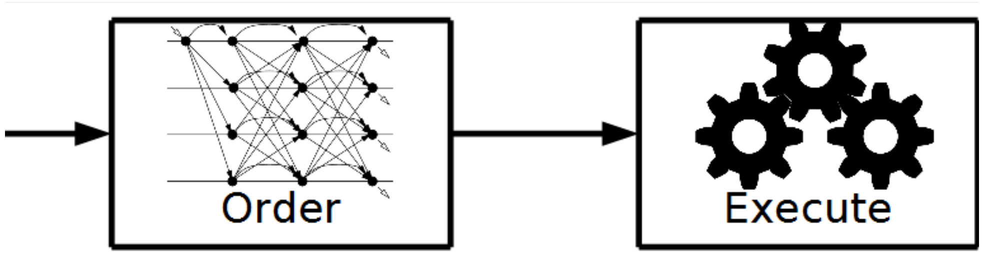
- https://digiconomist.net/ethereum-energy-consumption
- 11 TWh/year → 14% of Bitcoin
- 1 transaction can power 1 average US household for a day

- About 15 transactions per second possible peak throughput
- Latency about 7-8 minutes (1 block on average every 15 seconds, 30+ block confirmations)

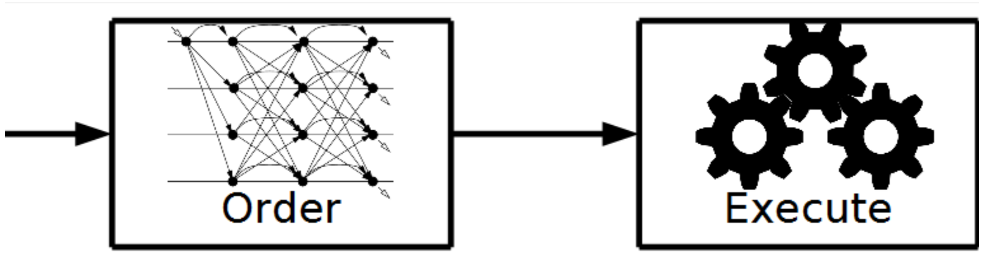
Permissioned Blockchains before Fabric v1 (also Fabric v0.5 and v0.6)



example:
PBFT [Castro/Liskov02]



Blockchain SOTA (prior to Fabric v1) follows order-execute architecture



- **Order** transactions using Proof-of-Work (PoW) or Byzantine Fault Tolerant (BFT) consensus

- **Execute** transactions at each node

-

- **Order/execute** architecture is found in many SMR systems
  - Active state machine replication [Schneider90]
  - Paxos and co., Raft
  - Vast majority of BFT

Blockchain evolution

2009

bitcoin

- **A hard-coded cryptocurrency application**
- **Limited stack-based scripting language**
- **Native cryptocurrency (BTC)**
- **Resource-intensive Proof-of-Work consensus**
- **Permissionless blockchain system**

Blockchain 1.0

2014

ethereum

- **General-purpose blockchain**
- **Distributed applications (smart contracts)**
- **Domain-specific language (Solidity)**
- Native cryptocurrency (ETH)
- Resource-intensive Proof-of-Work consensus
- Permissionless blockchain system

Blockchain 2.0

2017

HYPERLEDGER FABRIC

- General purpose blockchain
- **Distributed applications (chaincodes)**
- **Different general-purpose languages (e.g., golang, Java, Node)**
- **No native cryptocurrency**
- **Modular/pluggable consensus**
- Permissioned blockchain system (geared towards business applications)
- **Designed for multiple instances/deployments**

Blockchain 3.0

Hyperledger Fabric – key requirements

- **No native cryptocurrency** ✔
- **Ability to code distributed apps in general-purpose languages** ✔
- **Modular/pluggable consensus** ✔

Satisfying these requirements required
a complete overhaul of the (permissioned) blockchain design!

end result

Hyperledger Fabric v1

Eurosys 2018 paper

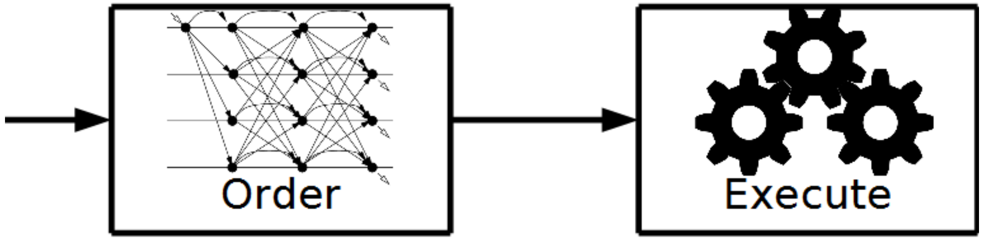
https://dl.acm.org/doi/10.1145/3190508.3190538

ORDER → EXECUTE architecture issues

- **Sequential execution of smart contracts**
  - long execution latency blocks other smart contracts, hampers performance
  - DoS smart contracts (e.g., infinite loops)
  - How Blockchain 2.0 copes with it:
    - Gas (paying for every step of computation))
    - Tied to a cryptocurrency
- **Non-determinism**
  - Smart-contracts must be deterministic (otherwise – state forks)
  - How Blockchain 2.0 copes with it:
    - Enforcing determinism: Solidity DSL, Ethereum VM
    - Cannot code smart-contracts in developers' favorite general-purpose language (Java, golang, etc)
- **Confidentiality of execution: all nodes execute all smart contracts**
- **Inflexible consensus: Consensus protocols are hard-coded**
- **Inflexible trust models: consensus trust model becomes also application trust model**

Fabric v1 architecture in one slide

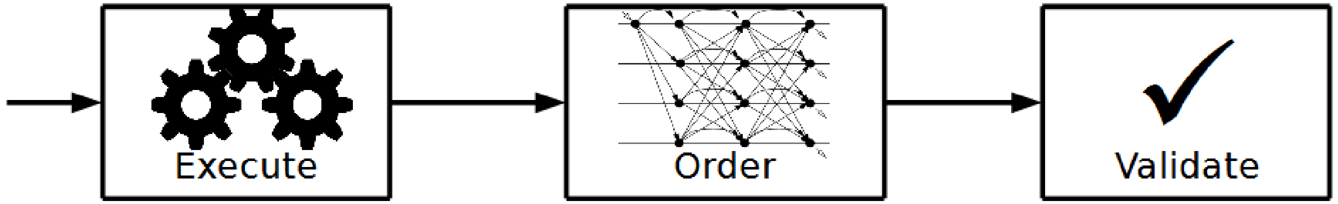
Existing blockchains' architecture



input tx

tx against smart contracts

Hyperledger Fabric v1 architecture



Tx against smart-contracts
Create versioned state updates
Collect endorsements

Versioned state updates/endorsements
Stateless ordering

Versioned updates&endorsements
Flag invalid and conflicting tx
Persist valid tx

Application consists of two components:

1) Chaincode (execution code)
2) Endorsement policy (validation code)

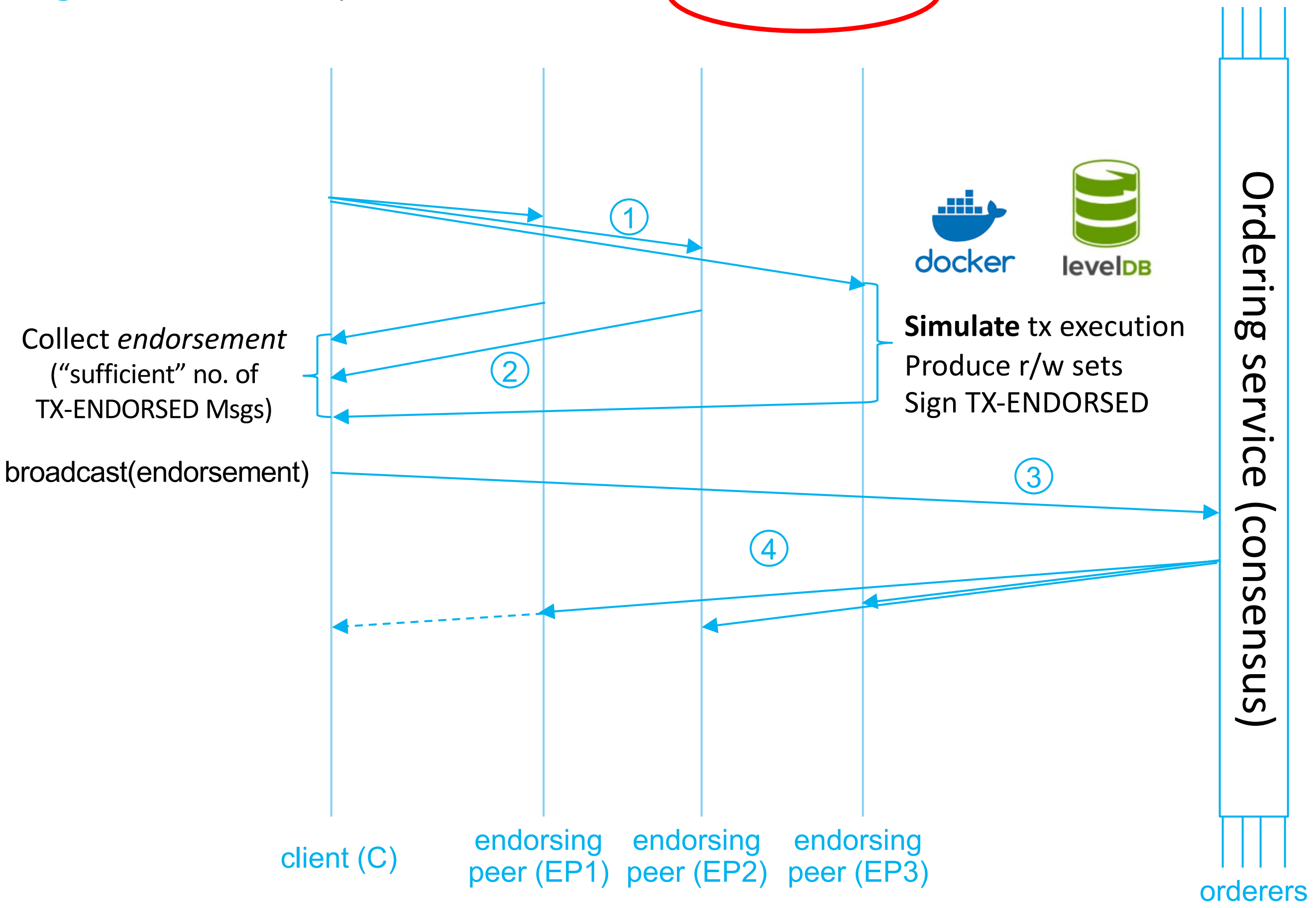
Node roles in Fabric

- **Fabric splits the roles of the nodes**
- **Peers**
  - Hold the application state
  - Execute and validate transactions
- **Ordering service**
  - Composed of ordering service nodes (OSNs or orderers)
  - Build the blockchain data structure
  - Impose total order across transactions, grouped in blocks
- **Clients**
  - Submit transactions to the system

Hyperledger Fabric v1 Transaction flow

1 <PROPOSE, clientID, chaincodeID, txPayload, timestamp, clientSig>

2 <TX-ENDORSED, peerID, txID, chaincodeID, readset, writeset>



Total order semantics
(ordering service)

3 BROADCAST(blob)

4 DELIVER(seqno,prevhash,block)

Hyperledger Fabric v1 Transaction flow

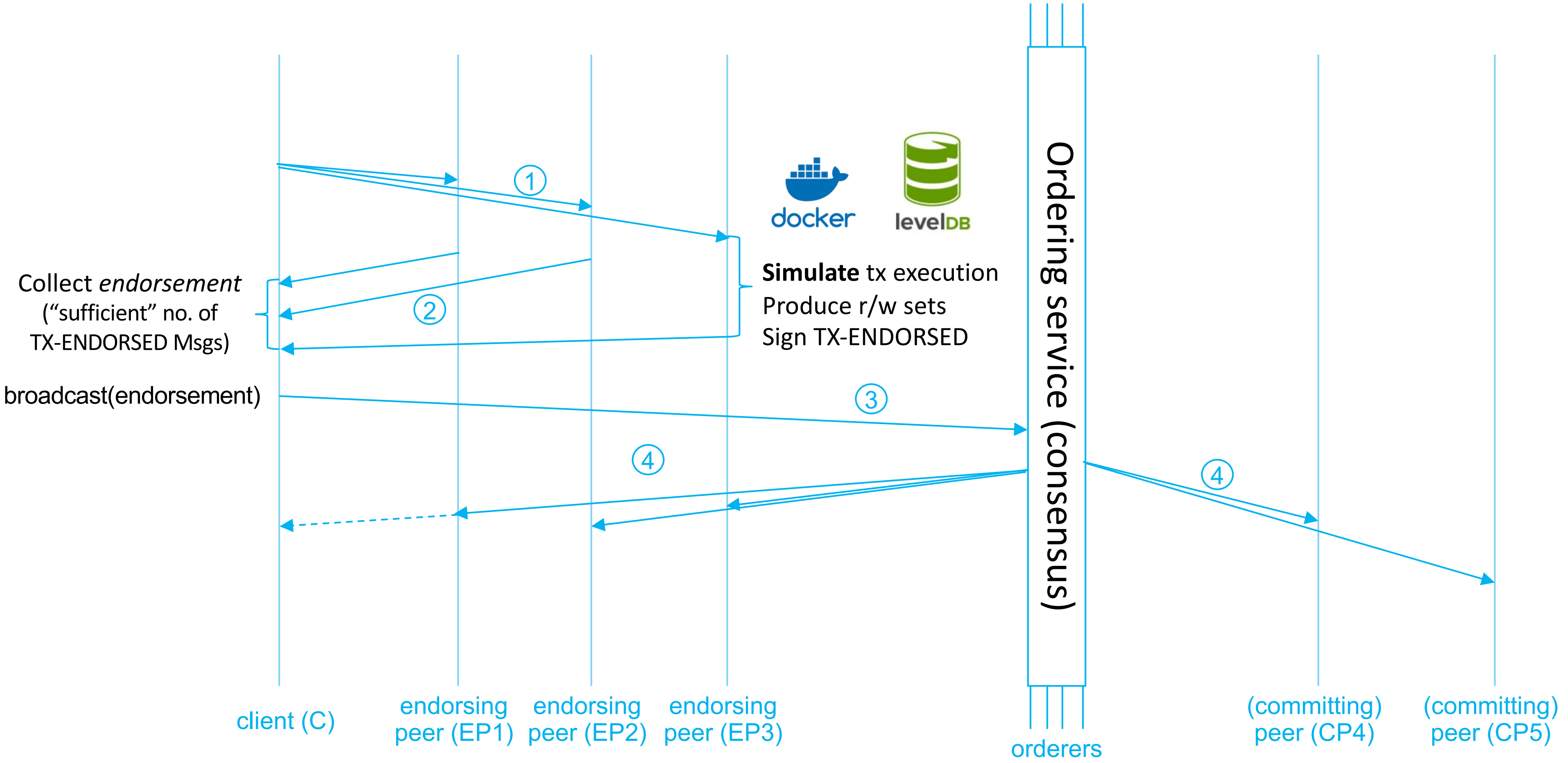
1 <PROPOSE, clientID, chaincodeID, txPayload, timestamp, clientSig>

2 <TX-ENDORSED, peerID, txID, chaincodeID, readset, writeset>

Total order semantics (ordering service)

3 BROADCAST(blob)

4 DELIVER(seqno,prevhash,block)



Hyperledger Fabric v1 Transaction flow

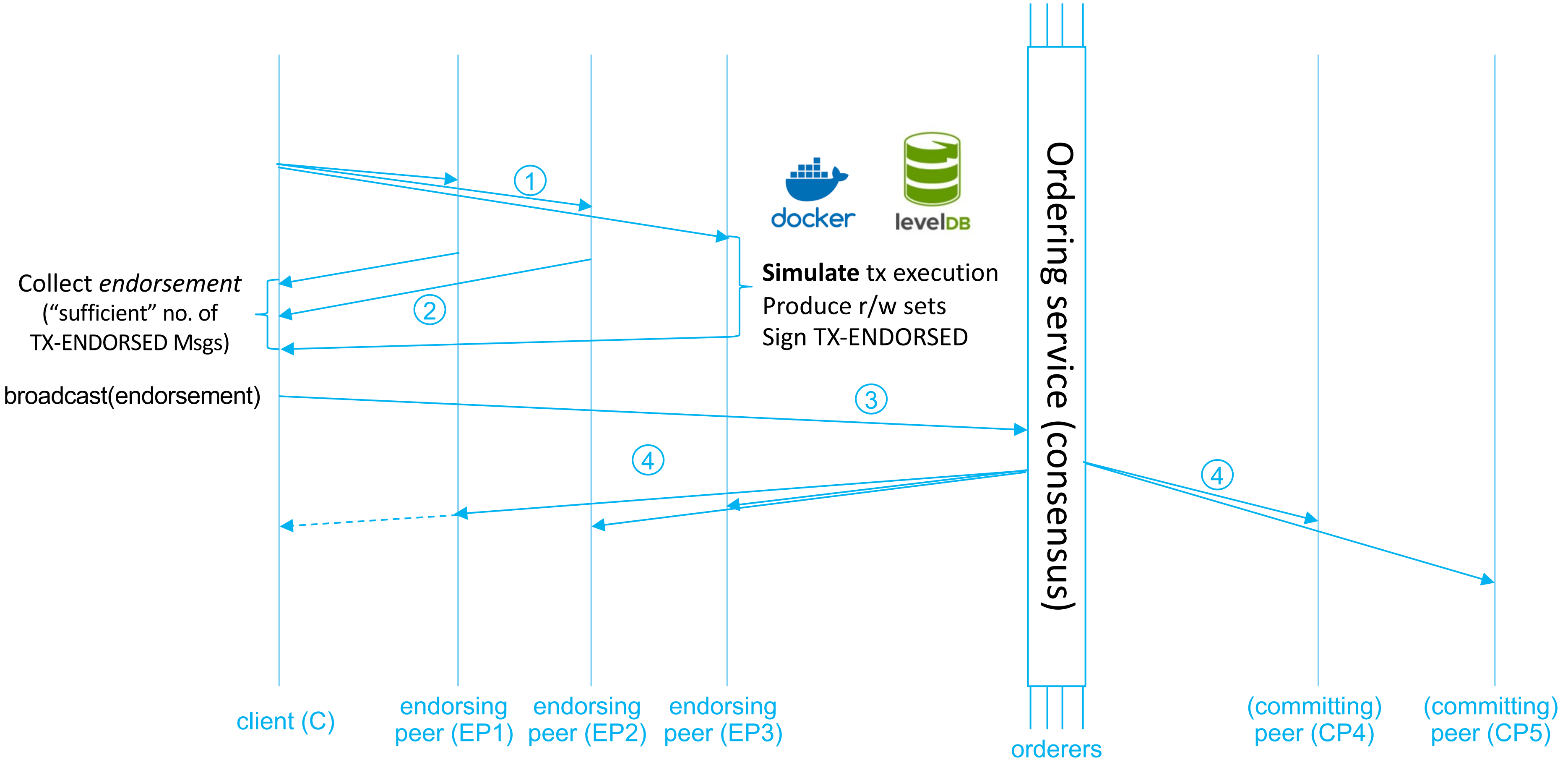
1 <PROPOSE, clientID, chaincodeID, txPayload, timestamp, clientSig>

2 <TX-ENDORSED, peerID, txID, chaincodeID, readset, writeset>

Total order semantics (ordering service)

3 BROADCAST(blob)

4 DELIVER(seqno,prevhash,block)



Hyperledger Fabric v1 Transaction flow

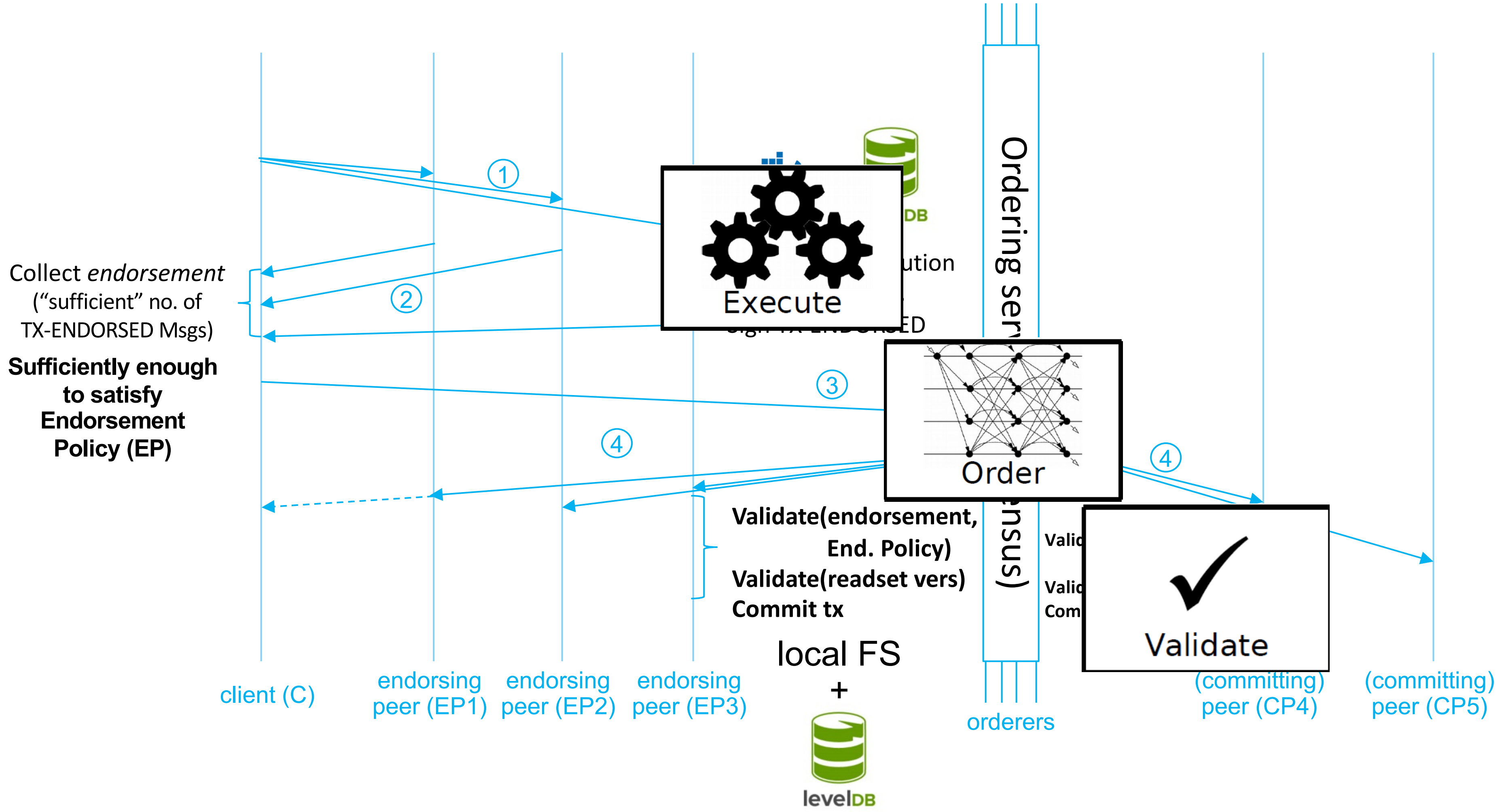
1 <PROPOSE, clientID, chaincodeID, txPayload, timestamp, clientSig>

2 <TX-ENDORSED, peerID, txID, chaincodeID, readset, writeset>

Total order semantics (ordering service)

3 BROADCAST(blob)

4 DELIVER(seqno,prevhash,block)



Challenge #1: Non-Determinism

- **Goals**
  - Enabling chaincodes in golang, Java, … (can be non-deterministic)
  - While preventing state-forks due to non-determinism

- **Hyperledger Fabric v1 approach**
  - Execute chaincode **before** consensus
  - Non-deterministic chaincode execution is tolerated
  - Use consensus to agree on propagation of versioned state-updates

EXECUTE→ORDER→VALIDATE:

non-deterministic tx are not guaranteed to be live

(e.g., cannot collect endorsement due to non-determinism)

ORDER→EXECUTE

non-deterministic tx are not guaranteed to be safe (forks can occur)

Challenge #2: Sequential execution of smart-contracts

- **Goals**
  - Prevent slow smart-contracts from delaying the system
  - Address DoS without native cryptocurrency

- **Hyperledger Fabric v1 approach**
  - Partition execution of smart-contracts
  - Only a subset of peers are endorsers for a given smart-contract (chaincode)

- **DoS, resource exhaustion?**
  - Fabric v1 transaction flow is resilient to non-determinism
  - Endorsers can apply local policies (non-deterministically) to decide when to abandon the execution of a smart-contract
  - No need for gas/cryptocurrency!

Challenge #3: Confidentiality of execution

- **Goal**
  - Not all nodes should execute all smart contracts

- **Hyperledger Fabric v1 approach**
  - Partition execution of smart-contracts
  - Only a subset of peers are endorsers for a given smart-contract (chaincode)

- **Later extended to Private chaincode execution leveraging Intel SGX**
  - Fabric Private Chaincode, SRDS 2019, https://arxiv.org/abs/1805.08541 (IBM Research + Intel collaboration)
  - Available in v1.4

- **Confidentiality of data (versioned updates) was later added for certain token applications**
  - Support for Zero Knowledge Asset Transfer (ZKAT) in Fabric v2-alpha
  - https://eprint.iacr.org/2019/1058

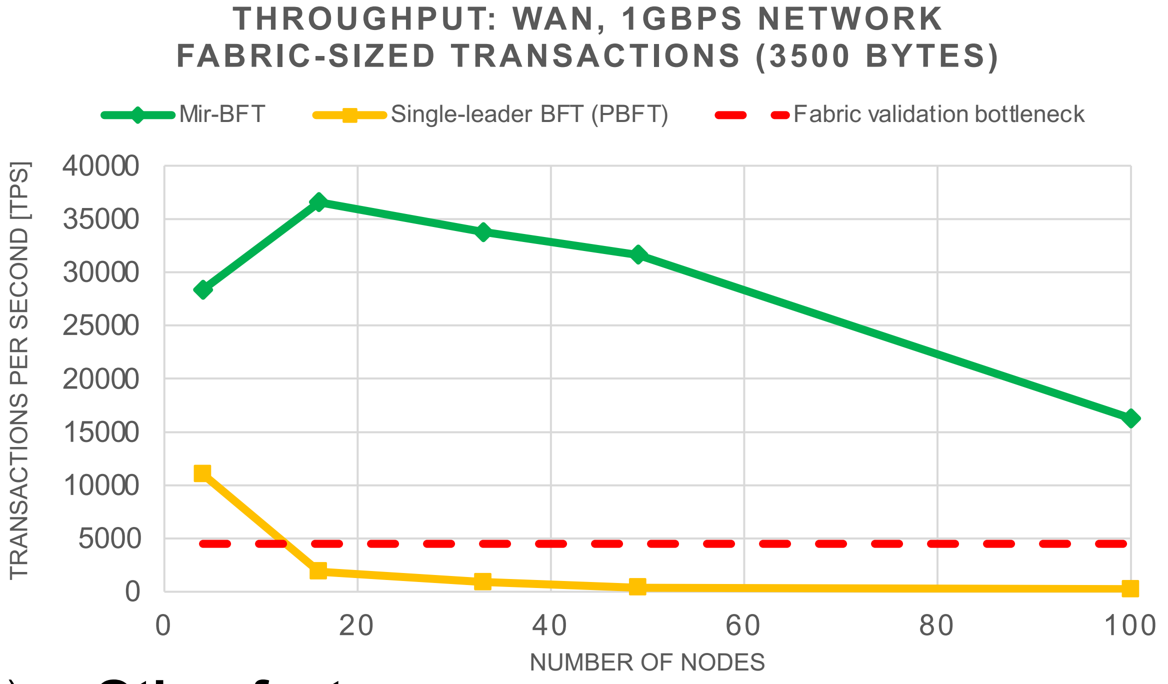
Challenge #4: Consensus modularity/pluggability

- **Goal**
  - No-one-size-fits-all consensus → Consensus protocol must be modular and pluggable
- **Hyperledger Fabric v1 approach**
  - Fully pluggable consensus (was present in order-execute v0.6 design as well)
- **HLF v1 consensus (ordering service) implementations**
  - Centralized! (**SOLO**, mostly for development and testing)
  - Crash FT (**KAFKA**, thin wrapper around Kafka/Zookeeper)
  - Both deprecated since v2.0
  - **Crash FT (RAFT, wrapper around etcd/raft) since v1.4.1**
- **BFT Consensus**
  - BFT-SMaRt Java library (Research collaboration with University of Lisbon) as PoC
    - Code: https://github.com/jcs47/hyperledger-bftsmart
    - Paper: https://arxiv.org/abs/1709.06921, later appeared in DSN 2018
  - Ported also to Go in 2019: https://github.com/SmartBFT-Go/
  - «Native» BFT implementation targeting about 100 orderers – in progress, expected in 2021
    - **Based on Mir-BFT, https://arxiv.org/abs/1906.05552**

Mir-BFT: Scalable and High-Throughput BFT consensus for Blockchains

(paper is available at https://arxiv.org/pdf/1906.05552.pdf)

	Proof of Work	Byzantine Fault Tolerance	Mir-BFT
Scalability	🙂	☹️	🙂
Fairness	🙂	☹️ ★	🙂
Energy Sustainability	☹️	🙂	🙂
Consensus Finality	☹️	🙂	🙂
Performance	☹️	🙂	🙂



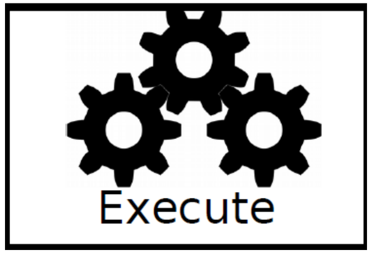
Main Design Principles

- **Multiple leaders**
  - Multiple leaders propose requests in parallel (vs PBFT single leader)
- **Prevents duplication that may arise with multiple leaders**
  - Request hashspace divided in buckets and sharded across a set of leaders (this deals with request duplication)
  - Bucket assignment to leaders periodically rotates (this eliminates censoring attacks)
- **Incrementally built on proven protocol (PBFT)**
  - Critical for easier reasoning about correctness

Other features

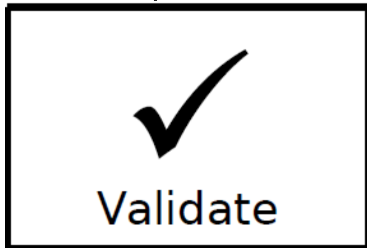
- WAN latencies at the order of 1-2s (finality)
- High performance in clusters (LAN) as well
- Robust to performance attacks
- Configurable as crash-fault tolerant (replacing Raft)

Challenge #5: Distributed applications with configurable trust assumptions



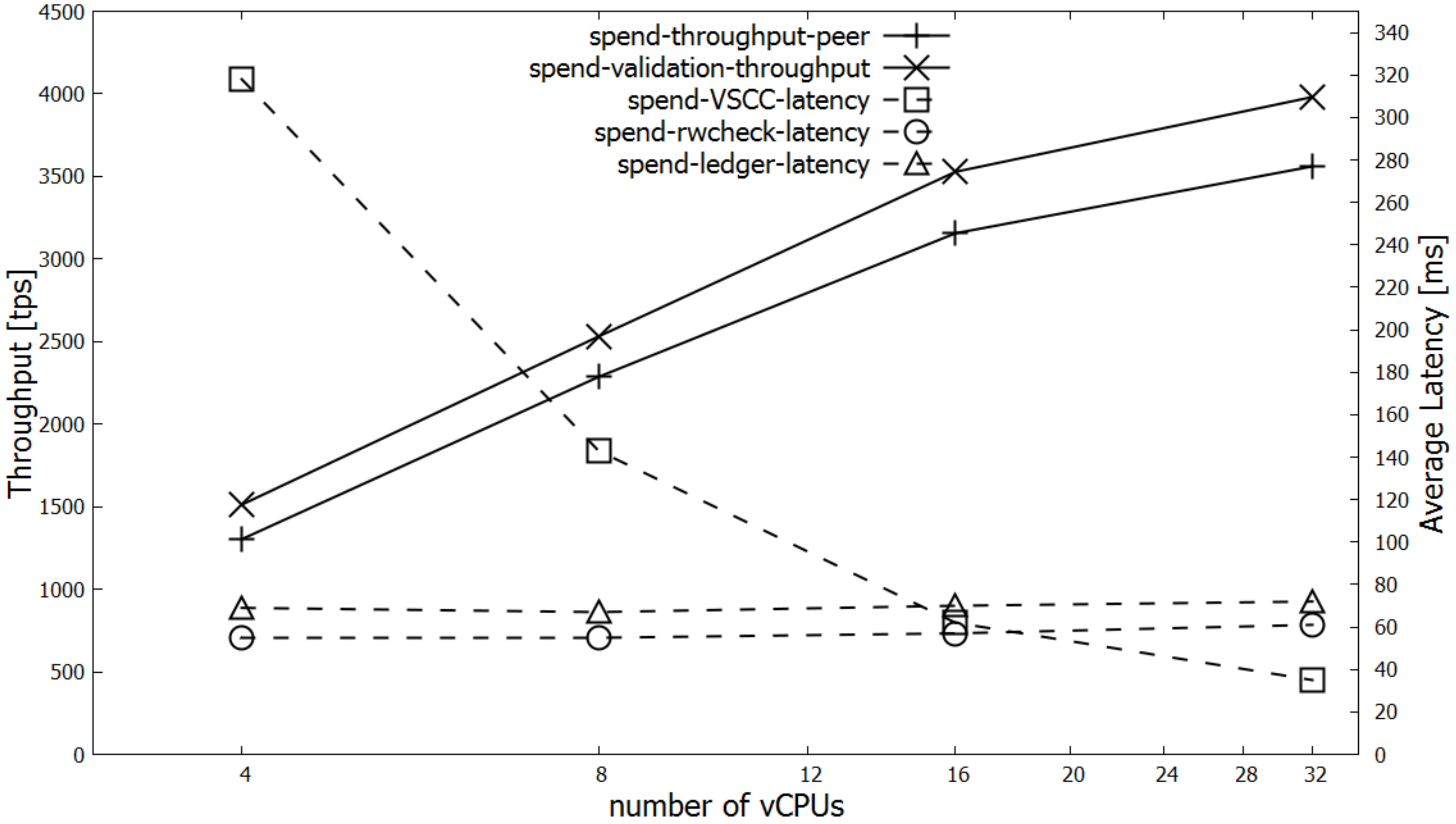
- **Execution code (a.k.a. chaincode)**
  - Execute untrusted chaincode **<u>before</u>** consensus
  - Non-deterministic chaincode tolerated
  - **EXECUTE→ORDER→VALIDATE: non-deterministic tx are not guaranteed to be live**
  - **ORDER→EXECUTE: non-deterministic tx are not guaranteed to be safe (forks)**

- **Validation code (a.k.a. endorsement policy)**
  - Deterministic(!), executed post-consensus
  - Deployed by a set of administrators (e.g., majority of nodes on the network)
  - Instantiated by chaincode
  - **Examples**
    - K out of N chaincode endorsers need to endorse a tx
    - Alice OR (Bob AND Charlie) need to endorse a tx
    - **Fabcoin** – Bitcoin-inspired UTXO authority-minted cryptocurrency for Fabric
    - Customized validation code

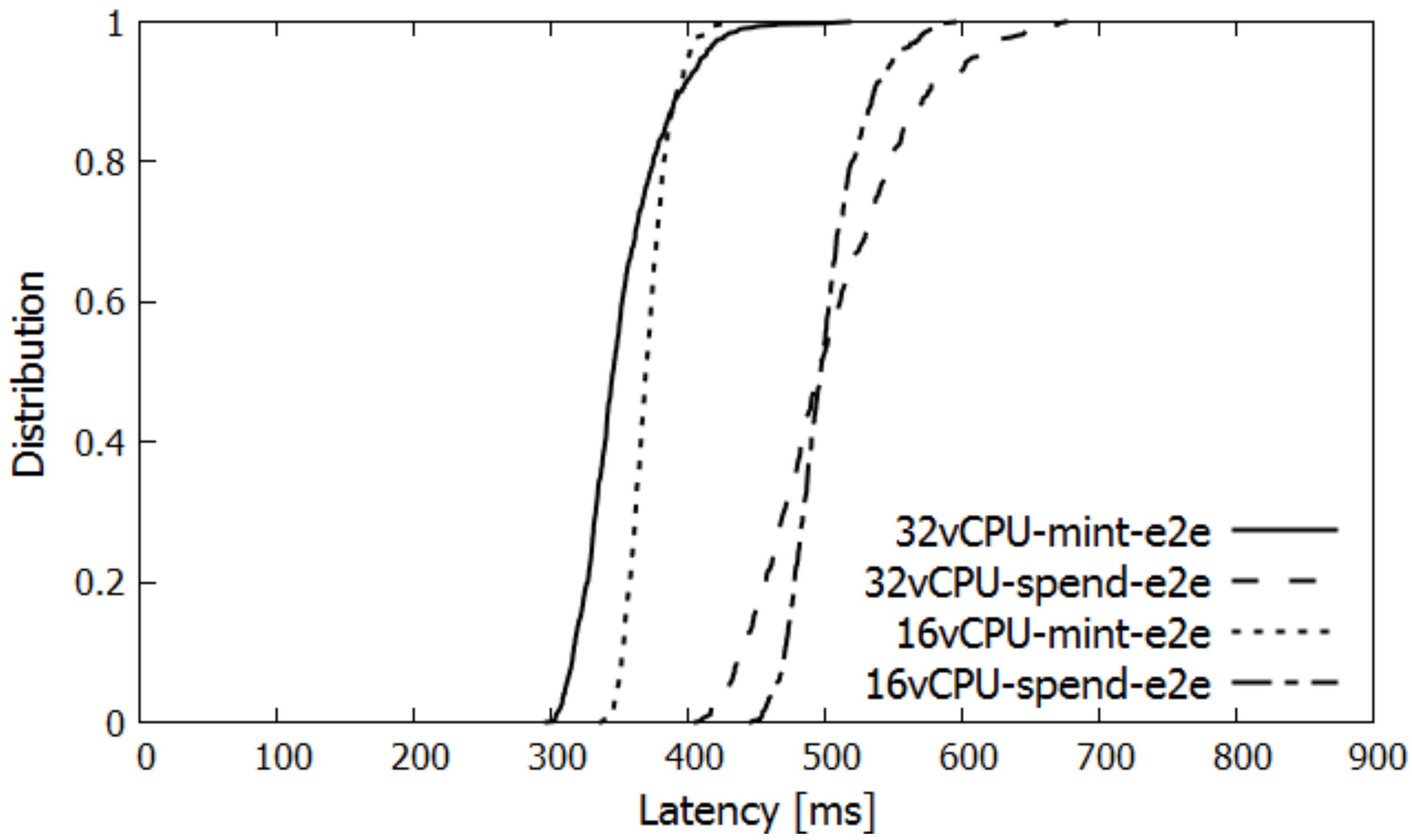


Fabric mixes
passive **and** ***active*** **replication**
into ***<u>hybrid</u>*** **replication**

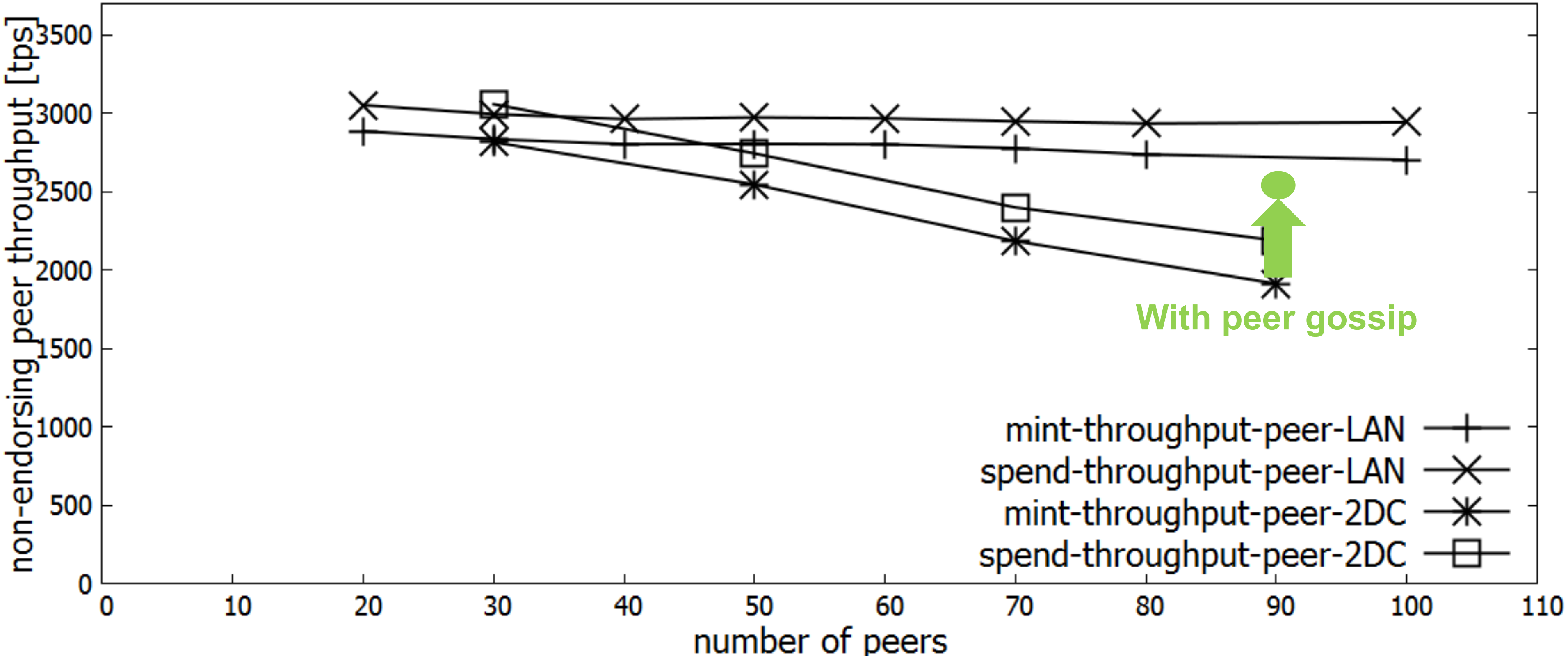
Fabric performance (Fabcoin)



Fabric performance (Fabcoin)



Fabric performance (Fabcoin)



Thank You!