Marko Vukolić, IBM Research - Zurich

June 22, 2017

Hyperledger Fabric

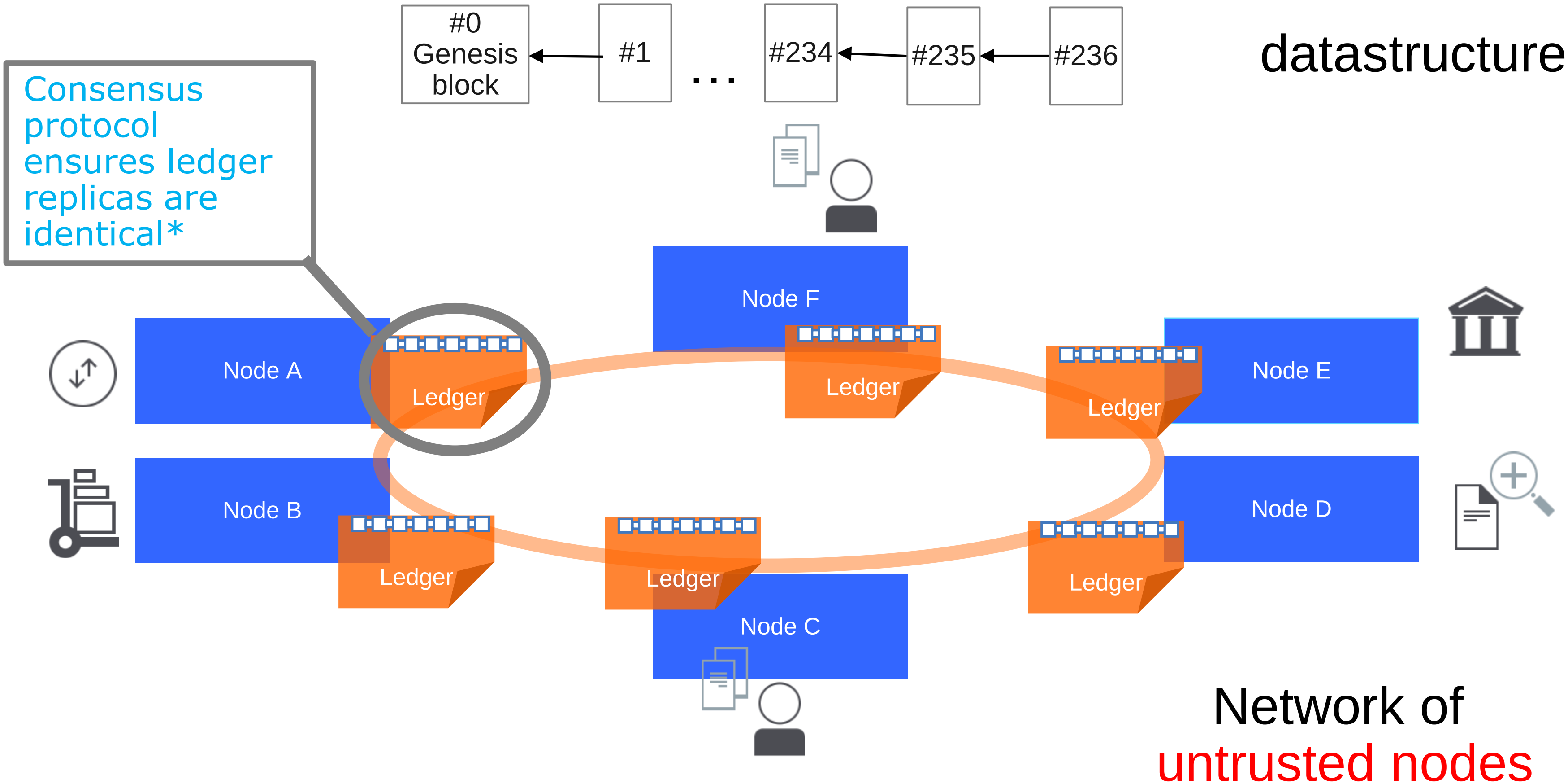
an open-source distributed operating system for permissioned blockchains

Swiss Blockchain Summer School

Lausanne, Switzerland
June 22, 2017

What is a Blockchain?

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



Blokchain transactions and distributed applications

- **Bitcoin transactions**
  - simple virtual cryptocurrency transfers

- **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 and which can take custody over assets on that ledger. [Swanson2015]

“Smart contract” → (replicated) state machine

So we just apply 40 years of research on RSM?

- RSM = Replicated State Machines [Lamport 78, countless follow-up papers]

Well, not really…

Among other differences:

- **RSM approach**
  - single **trusted** replicated application
- **Blockchain smart-contracts**
  - Multiple distributed applications
  - Developed by third party application developers
  - Not necessarily trusted!

Blockchain evolution (2009-present)

2009
Bitcoin

- **A hard-coded cryptocurrency application w. limited stack-based scripting language**
- **Proof-of-work-consensus**
- **Native cryptocurrency (BTC)**
- **Permissionless blockchain system**

Blockchain 1.0

2014
Ethereum

- **Distributed applications (smart contracts) in a domain-specific language (Solidity)**
- Proof-of-work-consensus
- Native cryptocurrency (ETH)
- Permissionless blockchain system

Blockchain 2.0

2017
Hyperledger
Fabric

- **Distributed applications (chaincodes) in different general-purpose languages (e.g., golang, Java)**
- **Modular/pluggable consensus**
- **No native cryptocurrency**
- **Multiple instances/deployments**
- Permissioned blockchain system

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

http://github.com/hyperledger/fabric

We will skip many details

- Membership Service Provider (and CAs)
- Chaincode details
- Gossip
- Ledger design
- Channels

http://hyperledger-fabric.readthedocs.io/en/latest/

Focus of this talk is on

system architecture and distributed systems aspects

THE LINUX FOUNDATION

HYPERLEDGER PROJECT

PREMIER MEMBERS

GENERAL MEMBERS

ASSOCIATE MEMBERS

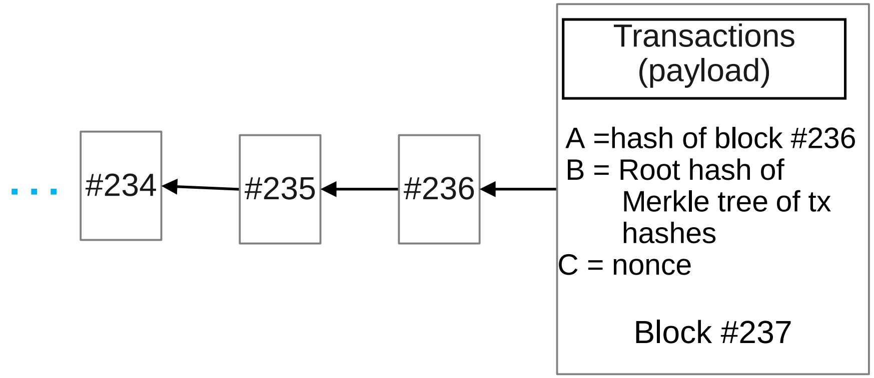
https://github.com/hyperledger

https://www.hyperledger.org/

Blockchain Architecture 101

Permissionless Blockchains

- **PoW Consensus**
  - Block “mining”



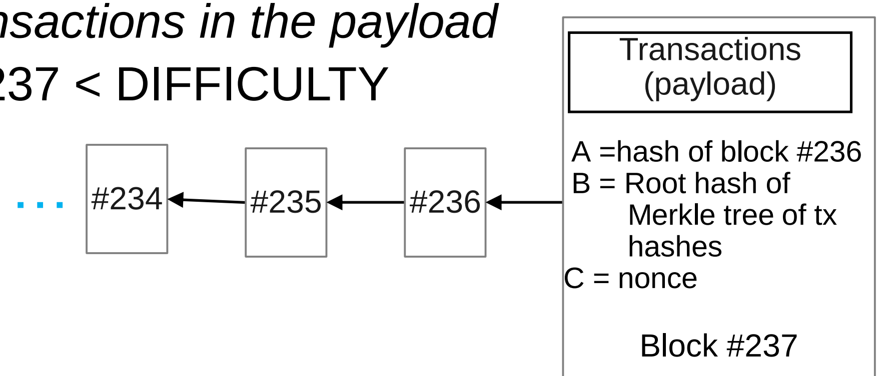
- *Validating (executing) transactions in the payload*
- *Finding nonces such that*
  h = hash of Block #237 = SHA256(A||B||C) < DIFFICULTY

- Block #237 propagation to the network

<u>ORDER→ EXECUTE architecture</u>

Nodes execute smart-contracts <u>after</u> consensus (PoW)

- **Block Validation / Smart Contract Execution (every miner)**
  - *Validating (executing) transactions in the payload*
  - *Verifying* hash of Block #237 < DIFFICULTY



Permissioned blockchains

- **Nodes (participants) need a permission to participate in the blockchain network**

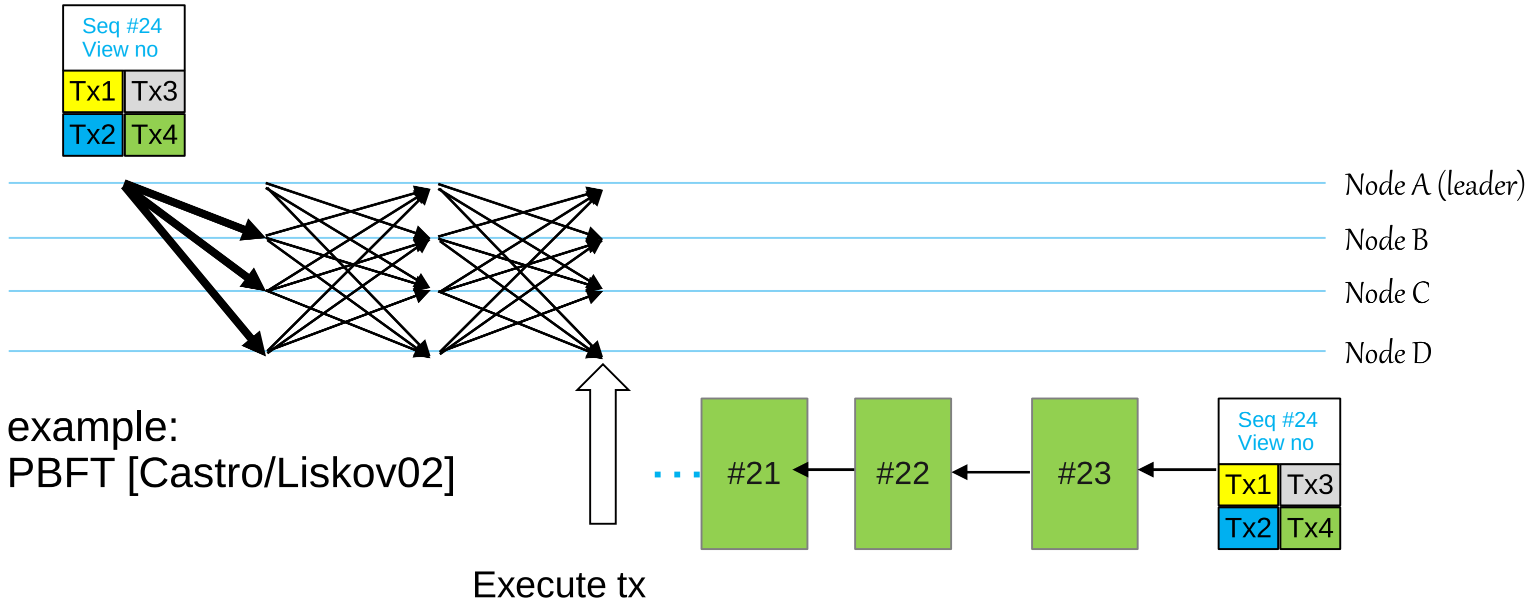
- **Motivation**
  - business applications of blockchain and distributed ledger technology (DLT)
  - Improving performance of public permissionless blockchains

- **In business applications**
  - Participant often need ability to identify other participants
  - Participants do not necessarily trust each other

- Examples: Chain, Kadena, Tendermint, Ripple, Symbiont, and…

  …Hyperledger Fabric

Permissioned Blockchain 2.0 architecture



Active state machine replication [Schneider90]

- **ORDER→ EXECUTE architecture**
- Inputs to the state machine (smart contract txs) are totally ordered
- Executed in sequence, after consensus (ordering)
- **ALL** permissioned blockchains are architected like this (incl Hyperledger Fabric v0.6), **<u>until Fabric v1</u>**

PoW vs. BFT for Blockchain (simplified overview)

	Proof of Work (Bitcoin, Ethereum,...)	BFT state machine replication (Ripple, Hyperledger, …)
Membership type	Permisionless	Permissioned
User IDs (Sybil attack)	Decentralized, Anonymous (Decentralized protection by PoW compute/hash power)	Centralized, all Nodes know all other Nodes (Centralized identity management protects against Sybil attacks)
Scalability (no. of Nodes)	Excellent, >100k Nodes	~~Verified up to few tens (or so) Nodes~~ Can scale to 100 nodes with certain performance degradation

Open research problem:

Given the use case, network, no. of nodes

What is the most suitable and scalable Blockchain technology/protocol?

Peak Throughput	from 7 tx/sec (Bitcoin)	>10k tx/sec with existing implem. in software [<10 nodes]
Power efficiency	>1 GW (Bitcoin)	Good (commodity hardware)
Temporary forks in blockchain	Possible (leads to double-spending attacks)	Not possible
Consensus Finality	No	Yes

What are the issues with ORDER → EXECUTE architecture?

ORDER → EXECUTE architecture issues (Blockchain 2.0)

- **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-model: Exposing low-level consensus assumptions**

Hyperledger Fabric v1 Architecture

http://github.com/hyperledger/fabric

HYPERLEDGER PROJECT

HLF v1 architecture in one slide

- **Existing blockchains' architecture**

ORDER → EXECUTE

(input tx) (tx against smart contracts)

- **Hyperledger Fabric v1 architecture**

EXECUTE → ORDER → VALIDATE

(tx against smart contracts) (versioned state updates) (versions, execution attestations)

Application developers specify two application components:

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

Challenge #1: Non-Determinism

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

- **Hyperledger Fabric v1 approach**
  – Execute smart contracts **before** consensus
  – Use consensus to agree on propagation of versioned state-updates

Hyperledger Fabric v1 Transaction flow

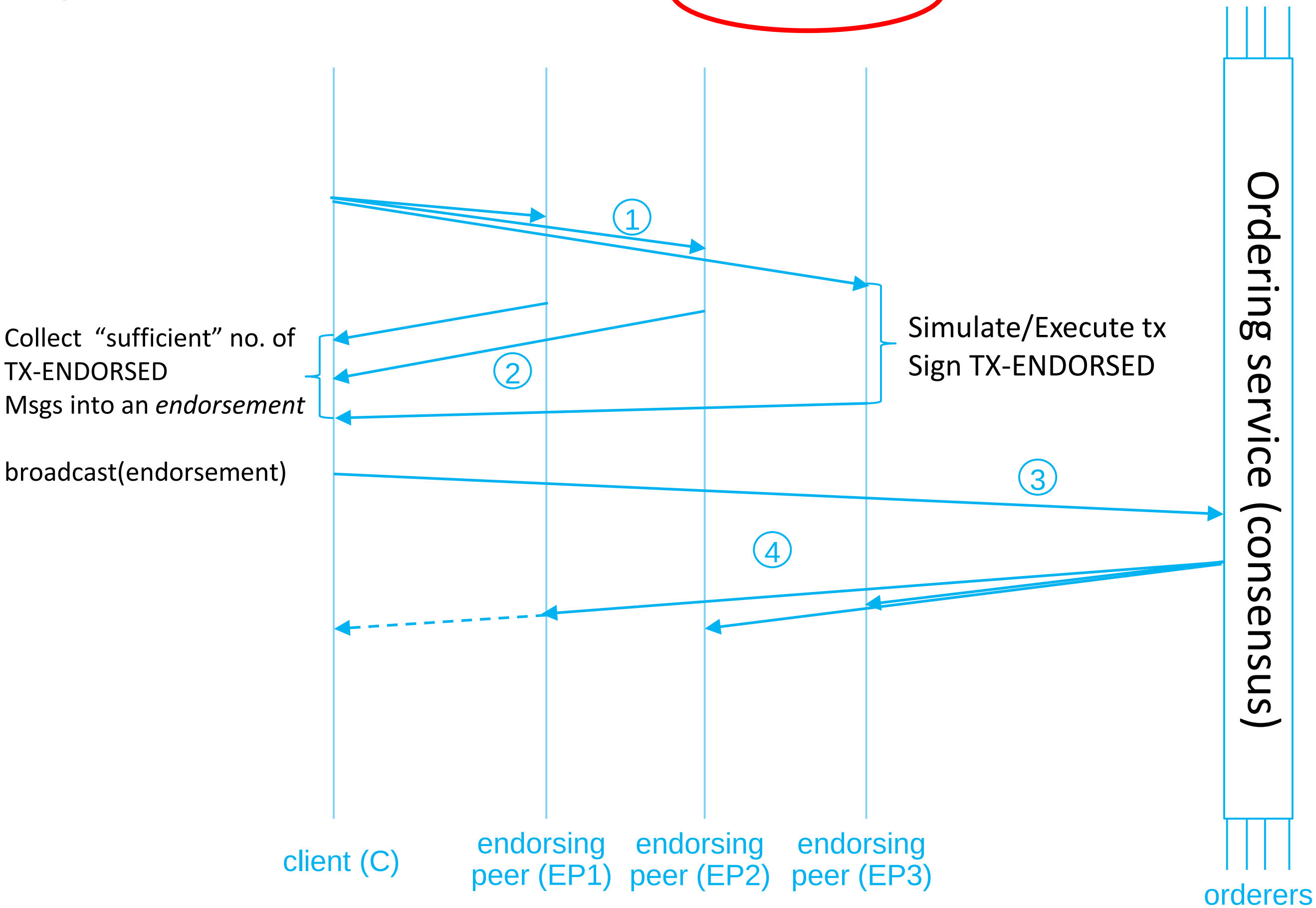
1 <PROPOSE, clientID, chaincodeID, txPayload, timestamp, clientSig>

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

<u>Total order semantics (HLF v1)</u>

3 BROADCAST(blob)

4 DELIVER(seqno,prevhash,block)



On readset and writeset

- **HLF v1 models state as a key-value store (KVS)**
  – This is only a model (KVS does not have to be used)

- **Readset**
  – Contains all keys, read by the chaincode, during execution
  – Along with their monotonically increasing version numbers

- **Writeset (state updates)**
  – Contains all keys written by the chaincode
  – Along with their new values

- Eventual application of writeset is conditional on readset being (still) valid in the Validation phase (like MVCC in DBs)

Challenge #2: Sequential execution of smart-contracts

- **Goal**
  - Preventing that slow smart-contracts delay the system

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

Hyperledger Fabric v1 Transaction flow

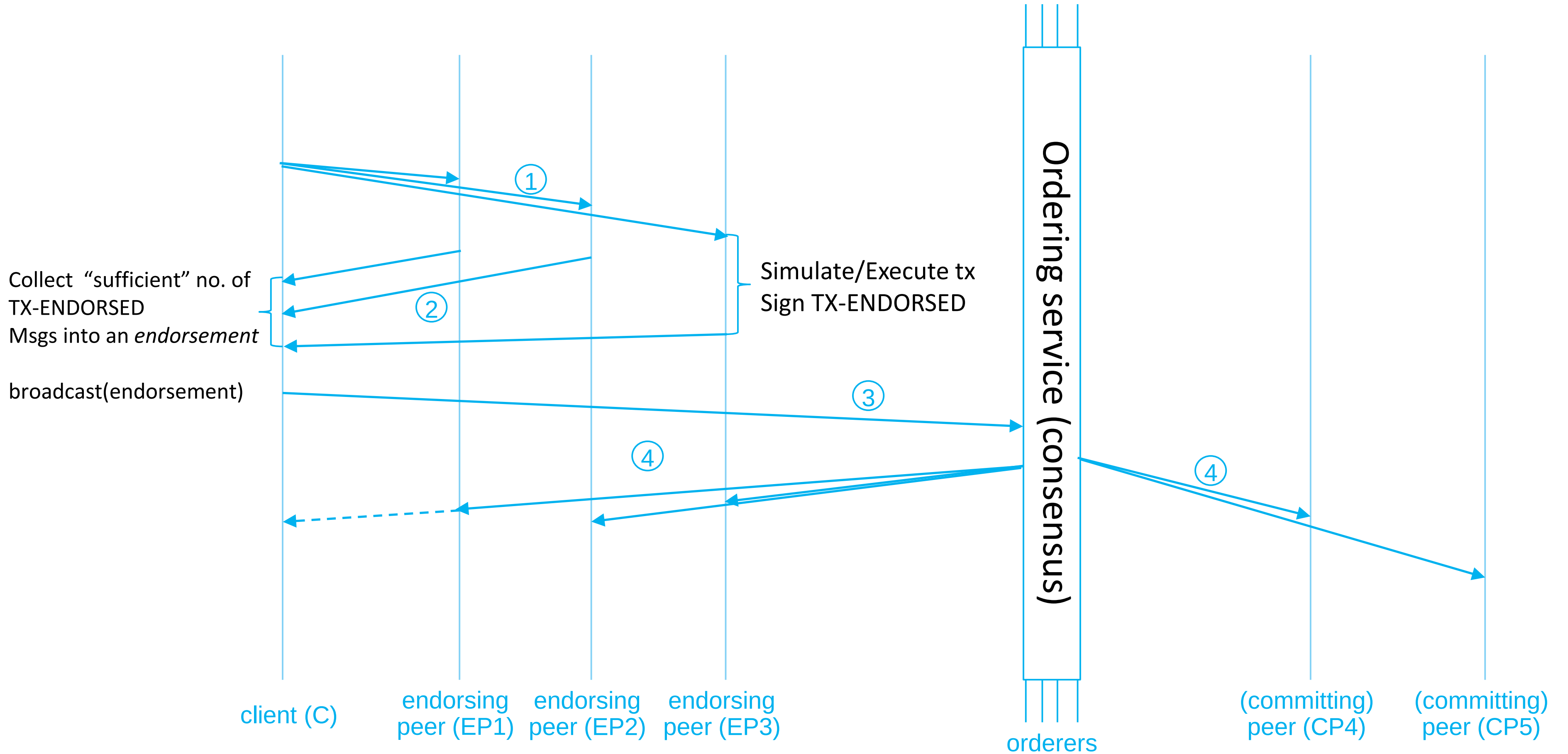
1 <PROPOSE, clientID, chaincodeID, txPayload, timestamp, clientSig>

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

Total order semantics (HLF v1)

3 BROADCAST(blob)

4 DELIVER(seqno,prevhash,block)



What about DoS, resource exhaustion?

- **HLF v1 transaction flow is resilient* to non-determinism**
- **Hence, endorsers can apply local policies (non-deterministically) to decide when to abandon the execution of a smart-contract**
  - No need for gas/cryptocurrency!

*** EXECUTE→ORDER→VALIDATE:**

non-deterministic tx are not guaranteed to be live

ORDER→EXECUTE

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

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)

Hyperledger Fabric v1 Transaction flow

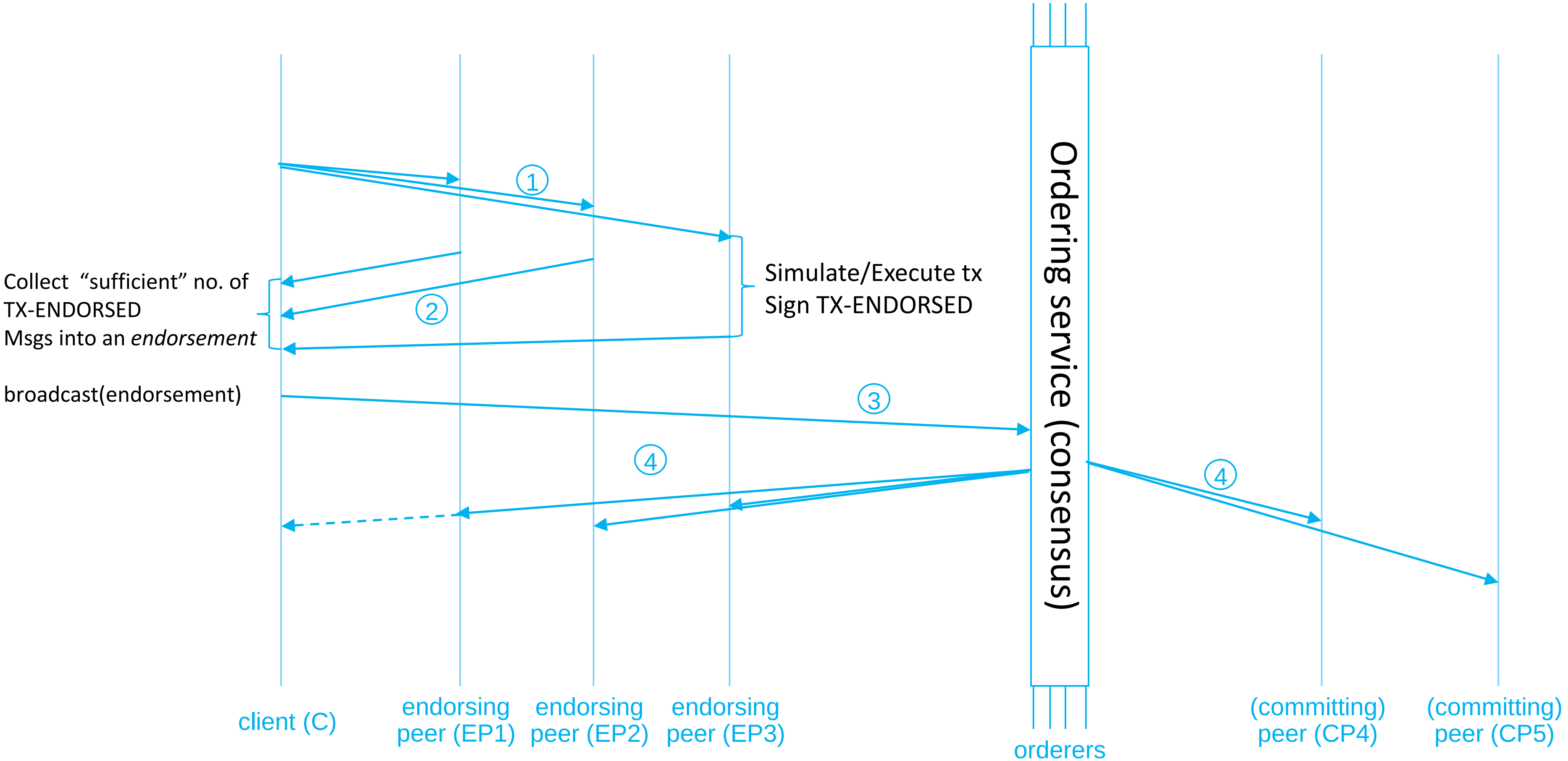
1 <PROPOSE, clientID, chaincodeID, txPayload, timestamp, clientSig>

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

Total order semantics (HLF v1)

3 BROADCAST(blob)

4 DELIVER(seqno,prevhash,block)



Challenge #4: Consensus modularity/pluggability

- **Goal**
  - With no-one-size-fits-all consensus:
  - Consensus protocol must be modular and pluggable

- **Hyperledger Fabric v1 approach**
  - Fully pluggable consensus (was present in v0.6 design as well)

- **HLF v1 consensus (ordering service) implementations, June 2017**
  - Centralized! (**SOLO**, mostly for development and testing)
  - Crash FT (**KAFKA**, thin wrapper around Kafka/Zookeeper)

- **Many more to come**
  - BFT-SMaRt library (University of Lisbon, expected July 2017, https://github.com/bft-smart/library)
  - Others? SGX Consensus (TU Braunschweig, Eurosys 2017), Honeybadger BFT (UIUC, CCS'16), XFT (IBM, OSDI'16)

Perhaps also your new, great blockchain consensus?

Challenge #5: Smart-contract trust flexibility

- **Goal**
  - Preventing low level consensus trust assumptions (e.g., “f out of 3f+1”) propagate to the application

- **Hyperledger Fabric v1 approach**
  - Let smart-contract developers specify application trust assumption
  - Trust assumption captured within **endorsement policy**

Hyperledger Fabric v1 Transaction flow

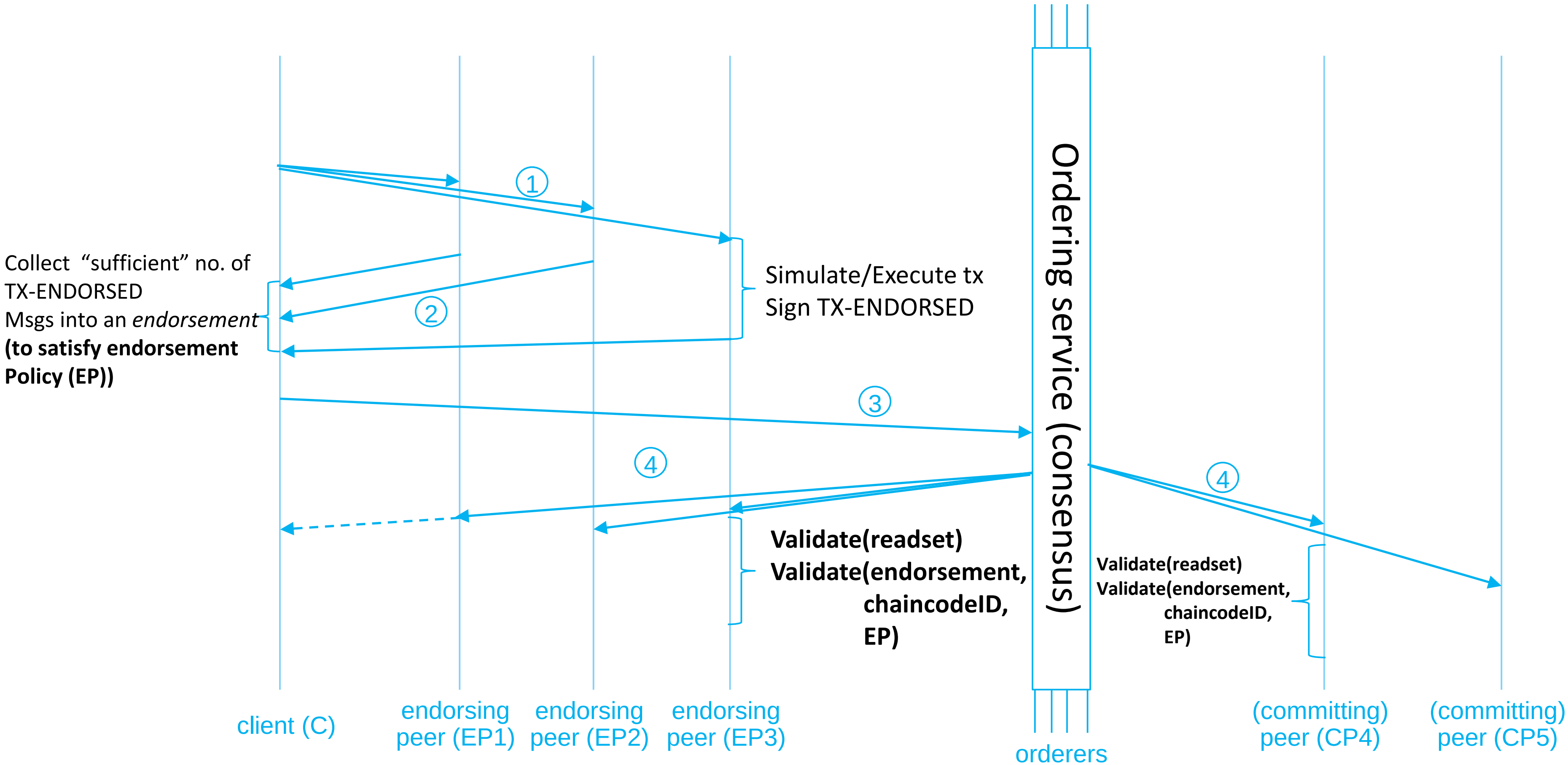
1 <PROPOSE, clientID, chaincodeID, txPayload, timestamp, clientSig>

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

Total order semantics (HLF v1)

3 BROADCAST(blob)

4 DELIVER(seqno,prevhash,block)



HLF v1 Endorsement Policies

- **Deterministic (!) programs used for validation**
- **Executed by <u>all peers</u> post-consensus**

- **Examples**
  – K out of N chaincode endorsers need to endorse a tx
  – Alice OR (Bob AND Charlie) need to endorse a tx

- **Cannot be specified by smart-contract developers**
- **Can be parametrized by smart-contract developers**

HLF v1 Endorsement Policies and Execution Flow

- **Endorsement Policy can, in principle, implement arbitrary program**

Hybrid execution model

EXECUTE → ORDER → VALIDATE approach of HLF v1

Can be used to split execution in two

EXECUTE (smart-contracts) → can be non-deterministic

VALIDATE(endorsement policy) → must be deterministic

HLF v1 mixes

***passive* and *active* replication**

into <u>*hybrid*</u> replication

Summary

Why we re-architected Hyperledger Fabric?

- **Hyperledger Fabric requirements**
  - Run smart contracts (chaincodes) in general-purpose language(s)
  - No native cryptocurrency
  - Modular consensus (unlike other permissioned blockchains)

All permissioned blockchains (incl. HLF up to v0.6)

order → execute

pattern

- **This is problematic:**
  - Sequential execution (limits throughput, DoS transactions)
  - All nodes execute all smart contracts (at odds with confidentiality)
  - Non-deterministic execution (hinders consistency, may create ``forks'')

Why we re-architected Hyperledger Fabric?

- **HLF v1 approach in one line:**

execute → order → validate

Permissioned blockchain architecture – overhauled

- **Modular/pluggable consensus**
  – There is no one-size-fits-all consensus (performance, trust flexibility)

- **Execute (chaincode) → Order (state updates) → Validate (endorsement policies)**
  – Chaincodes are no longer executed sequentially (performance, scalability)
  – Not all peers execute all chaincodes (helps with confidentiality, scalability)
  – Chaincode non-deterministic execution is not an issue (consistency)

- **Hybrid execution model (combining passive and active replication)**

Further reading

Why we re-architected HLF v1?

M. Vukolic. “Rethinking Permissioned Blockchains”, BCC 2017.

On non-determinism

C. Cachin, S. Schubert, M. Vukolic. “Non-determinism in Byzantine fault-tolerant replication”, OPODIS 2016.

PoW vs BFT consensus

M. Vukolic. “The Quest for Scalable Blockchain Fabric: PoW vs BFT replication”, iNetSec 2015.

New consensus protocols (and fault models)

S. Liu, P. Viotti, C. Cachin, V. Quema, M. Vukolic. “XFT: Practical Fault-Tolerance Beyond Crashes”, OSDI 2016.

Fault/trust model	CFT	XFT	BFT
Number of Nodes	2f+1	2f+1	3f+1
Tolerating Byzantine Nodes	no	yes	yes
Performance	Good	Practically as good as CFT	Poor (compared to CFT)

Thank You!