Marko Vukolić, IBM Research - Zurich

May 4, 2017

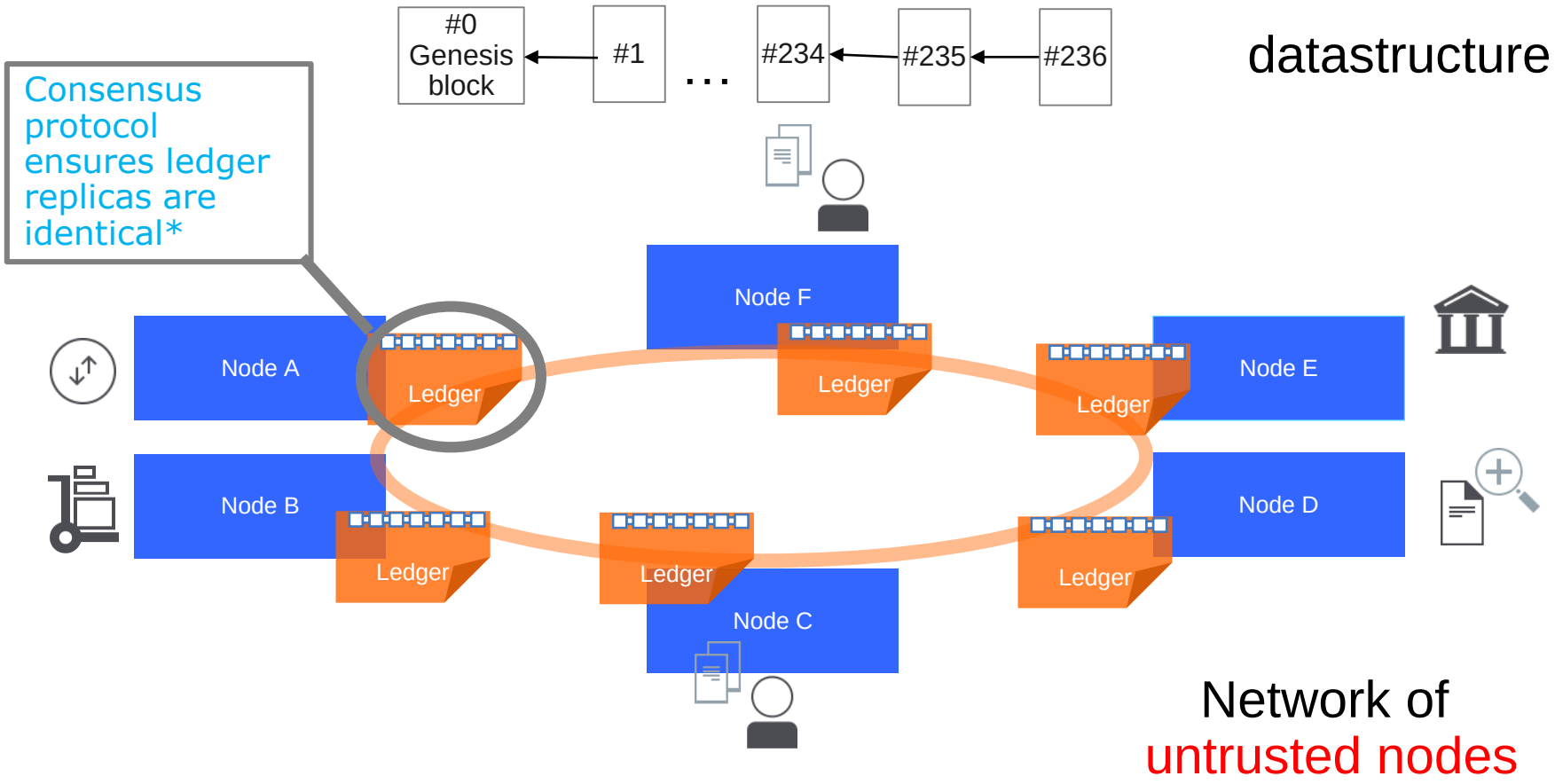
Hyperledger Fabric v1:

Rethinking Permissioned Blockchains

Blockchain: du Bitcoin au Smart Contract
4 Mai 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 transactions



This talk

How a set of seemingly simple functional requirements implied blockchain design overhaul?

Hyperledger Fabric v1

THE LINUX FOUNDATION

HYPERLEDGER PROJECT

https://github.com/hyperledger

https://www.hyperledger.org/

Hyperledger Fabric – key requirements

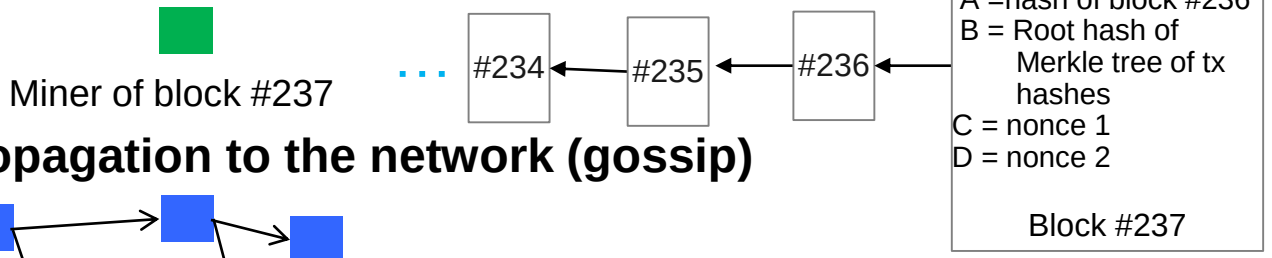
- **No native cryptocurrency**
- **Ability to code smart-contracts in general-purpose languages**
- **Modular/pluggable consensus**

Blockchain Architecture 101

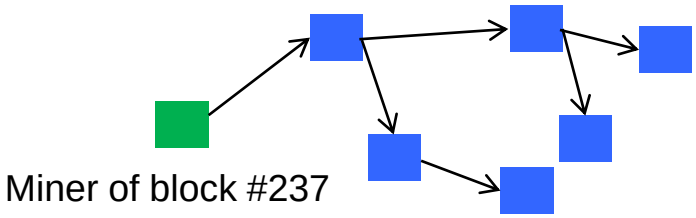
Permissionless Blockchains

find nonces such that
hash(Block#237) =SHA256(A||B||C||D) < DIFFICULTY

Step 1: Block "mining" (PoW Consensus)

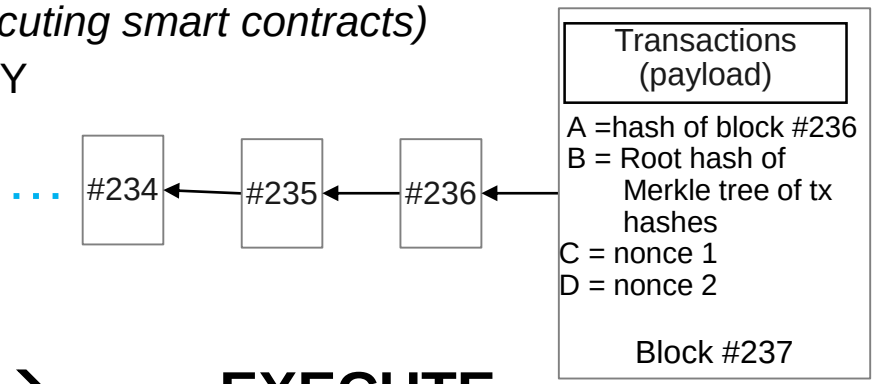


Step 2: Block #237 propagation to the network (gossip)



Step 3: Block Validation / Smart Contract Execution (every miner)

- *Validating transactions in the payload (executing smart contracts)*
- *Verifying* hash of Block #237 < DIFFICULTY



ethereum

ORDER using Consensus → **EXECUTE**

(input tx) **(tx against smart contracts)**

Permissioned blockchains

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

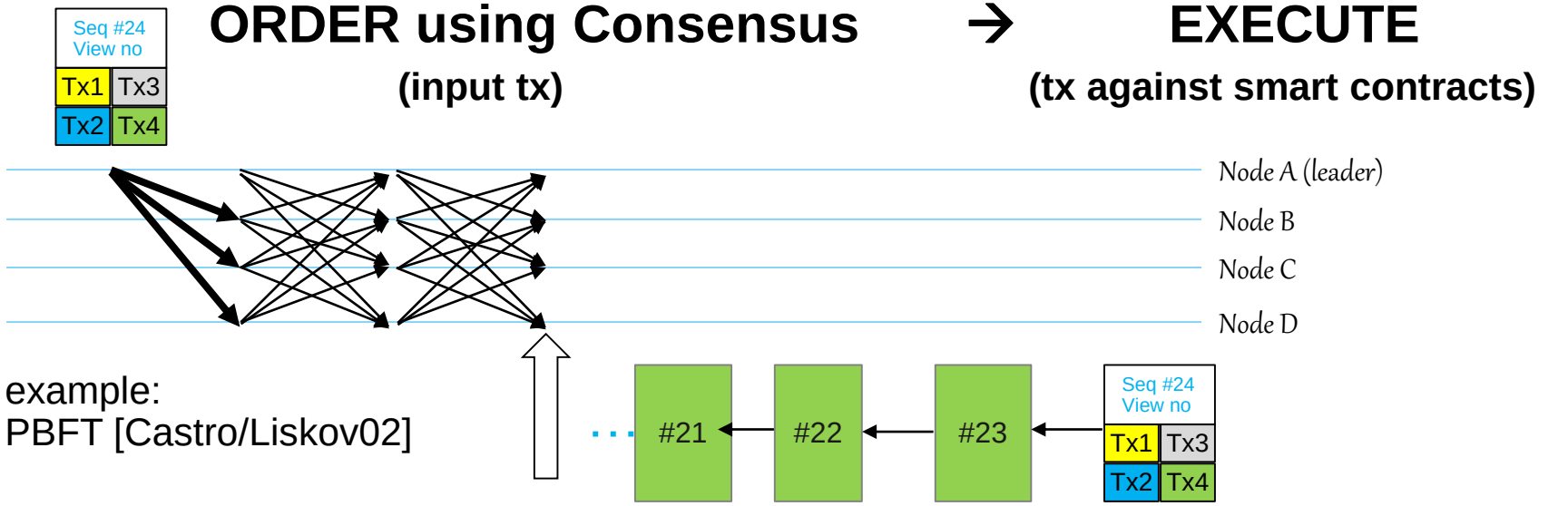
- **Motivation: business applications of blockchain and distributed ledger technology (DLT)**
  - Participant often need ability to identify other participants
  - Participants do not necessarily trust each other

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

  Hyperledger Fabric

Permissioned vs permissionless blockchains

- **Membership management**
  - Pemissioneless: none
  - Permissioned: node identities and membership need to be managed
- **Consensus (system) performance**
  - Permissionless (PoW consensus): high latency, low throughput
  - Permissioned (BFT consensus protocols): low latency, high throughput



What are the issues with ORDER → EXECUTE architecture (with HLF requirements in mind)?

Permissioned blockchain architecture issues

- **Sequential execution of smart contracts**
  - long execution latency blocks other smart contracts, hampers performance
  - DoS smart contracts (e.g., `while true { }`)
  - How permissioneless blockchains cope with it: 
    - Gas (paying for every step of computation)
    - Tied to a cryptocurrency
- **Non-determinism**
  - Smart-contracts must be deterministic (otherwise – state forks)
  - How permissioneless blockchains cope 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**

Hyperledger Fabric – key requirements

- **No native cryptocurrency** 
- **Ability to code smart-contracts in general-purpose languages** 
- **Modular/pluggable consensus** 

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

end result

Hyperledger Fabric v1

Hyperledger Fabric v1 Architecture

http://github.com/hyperledger/fabric

HLF v1 architecture in one slide

- **Existing blockchains' architecture**

ORDER using Consensus → EXECUTE

(input tx) (tx against smart contracts)

- **Hyperledger Fabric v1 architecture**

EXECUTE → ORDER using Consensus → VALIDATE

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

Step #1: Execute first

- **Goals**
  - Paralelize execution (addresses sequential execution bottleneck)
  - Partition execution (addresses confidentiality of execution)
  - Remove non-determinism (prevent state forks due to non-determinism)

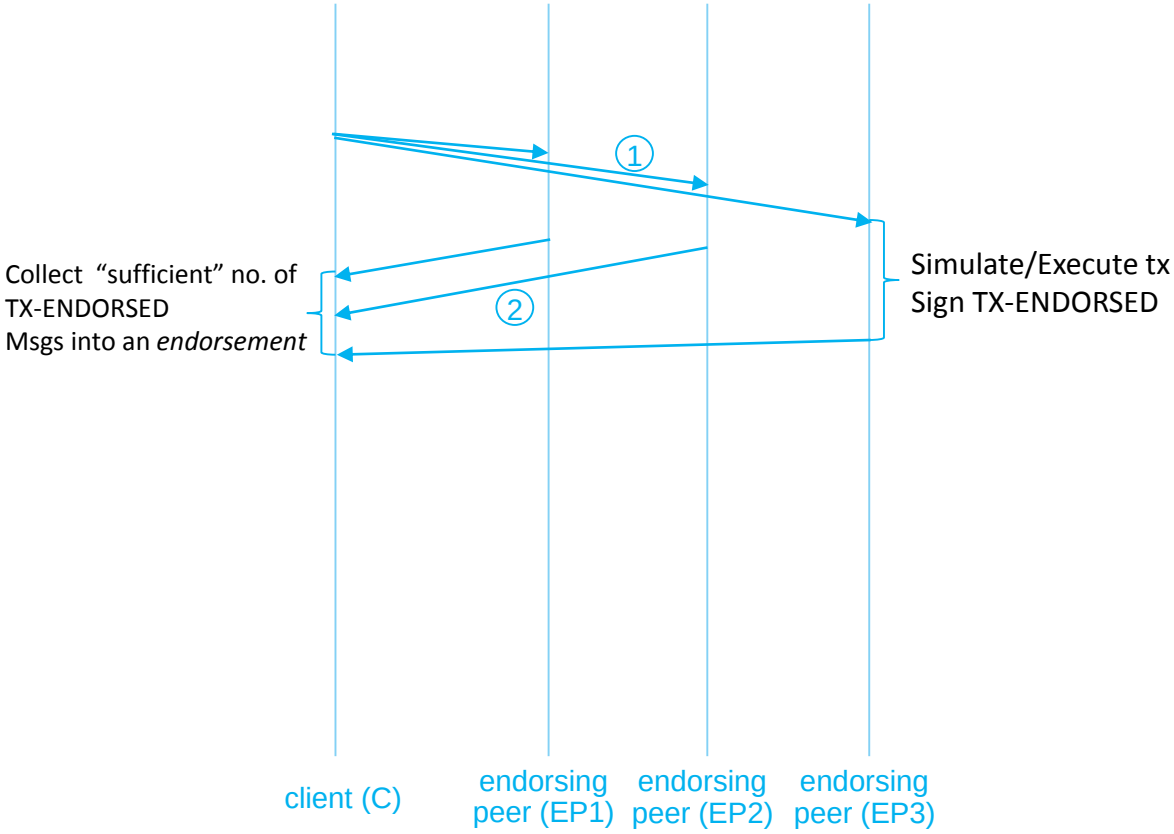
- **Hyperledger Fabric v1 approach**
  - A subset of nodes called ***endorsers*** executes chaincode**
    - Endorsers produce and sign versioned state updates
  - ***Client*** library orchestrates collection of execution results

**** HLF:chaincode ~ Ethereum:smart contract**

Hyperledger Fabric v1 Transaction flow

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

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



Step 2: Order using Consensus

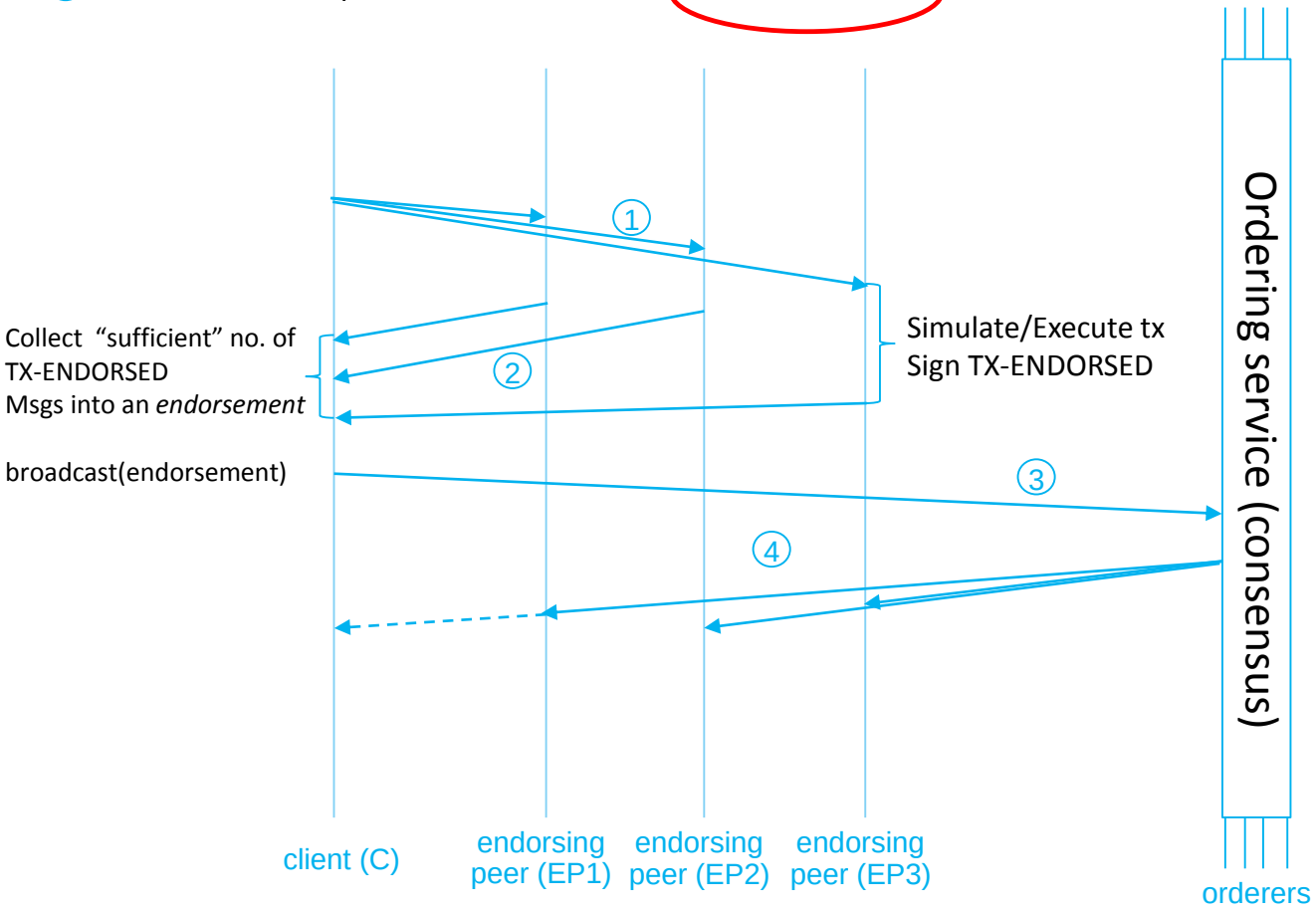
- **Goal**
  - Order versioned state-updates to prevent inconsistencies/double spending
  - Enforce consensus modularity

- **Hyperledger Fabric v1 approach**
  - Make consensus modular
  - Introduce ordering nodes (**orderers**)
  - Order **after** Execute → prevents inconsistencies due to non-determinism

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)

Hyperledger Fabric v1 Transaction flow

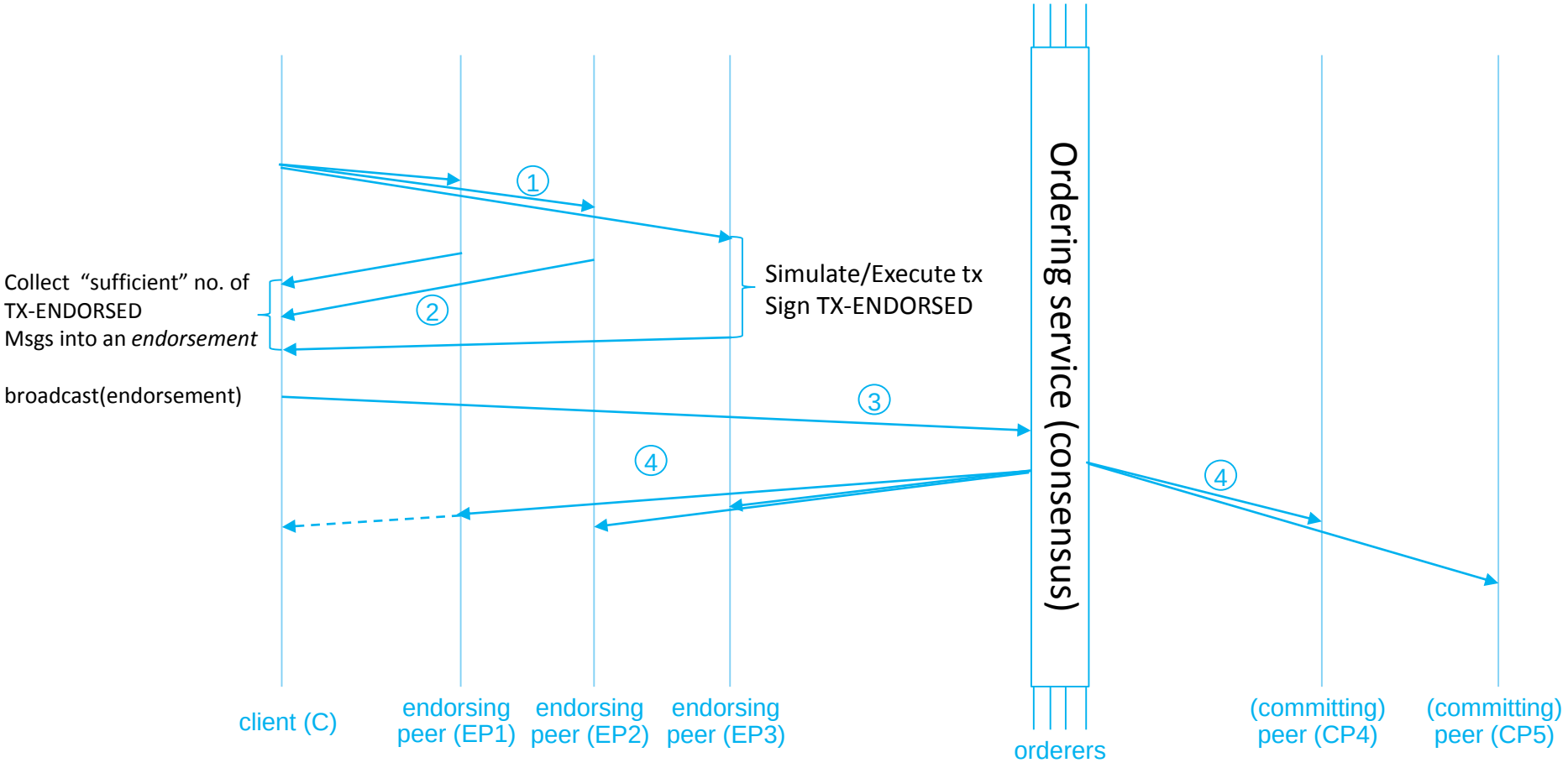
① <PROPOSE, clientID, chaincodeID, txPayload, timestamp, clientSig>

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

Total order semantics (HLF v1)

③ BROADCAST(blob)

④ DELIVER(seqno,prevhash,block)



HLF Consensus

- **HLF v1 consensus (ordering service) implementations**
  - Byzantine FT (**SimpleBFT**, variant of v0.6 PBFT, **development in progress**)
  - Crash FT (**KAFKA**, thin wrapper around Kafka/Zookeeper)
  - Centralized! (**SOLO**, mostly for development and testing)

- **Many more to come**
  - BFT-SMaRt (University of Lisbon), Honeybadger BFT (UIUC), XFT (IBM)

Perhaps also your favorite blockchain consensus?

Step #3: Validate after Ordering

- **Goal**
  - Efficiently validate execution results from (potentially untrusted) endorsers
  - Validate “freshness” of state updates (prevents asset double-spending)

- **Hyperledger Fabric v1 approach**
  - All peers verify versions of state updates coming out of consensus
  - All peers validate endorsers’ signatures against **endorsement policy**

Hyperledger Fabric v1 Transaction flow

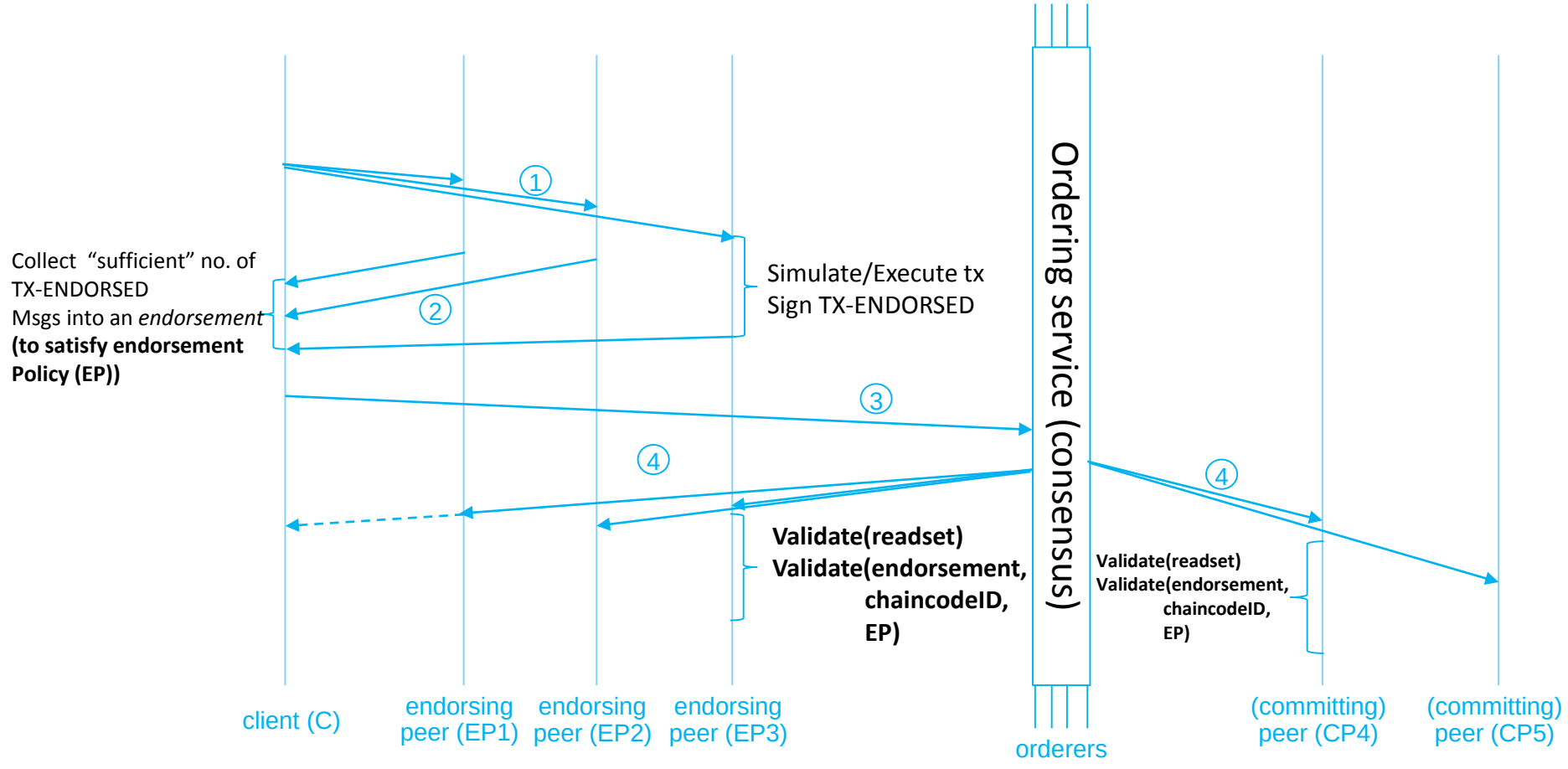
① <PROPOSE, clientID, chaincodeID, txPayload, timestamp, clientSig>

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 chaincode developers**
- **Can be parametrized by chaincode 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 (chaincode) → can be non-deterministic

VALIDATE(endorsement policy) → must be deterministic

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 chaincode**
  - 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)

Thank You!