ETH zürich

Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains

Blockchain Security Seminar

Pirmin Schmid

Seminar presentation and discussion of this paper

Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains

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 Jason Yellick

IBM

<EURO/SYS'18>

April 23-26, 2018, Porto, Portugal

Bitcoin-like blockchains

- Distributed public anonymous ledger
- Consensus by longest chain
- PoW / PoS
- Fixed system for each variant

- Applications

picture from pixabay (CC0)

Fabric

- Open-source Framework to build blockchains
- Modular for all aspects of the system
- Permissioned
- No currency
- Go, Java, Node.js, …

- Example use cases

- New very crucial insights

picture from pixabay (CC0)

Fabric Components

Membership service provider (MSP)

Policies

Chaincode

Peer

Endorser: execute

Docker

Committer: validate

Ledger: transaction manager (PTM)

KVS: Database

Fabric Components

Membership service provider (MSP)

Policies

Chaincode

Client

Client

Client

Client

Peer

Docker

Endorser: execute

Committer: validate

Ledger: transaction manager (PTM)

KVS: Database

Client

Client

Client

Client

Gossip

Order service

Fabric Components

Membership service provider (MSP)

Policies

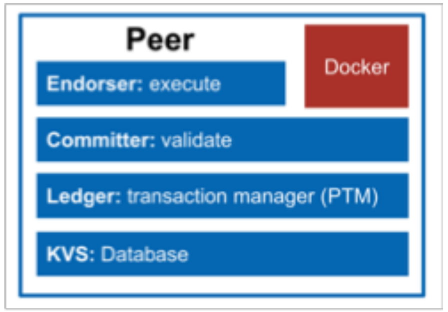
Chaincode

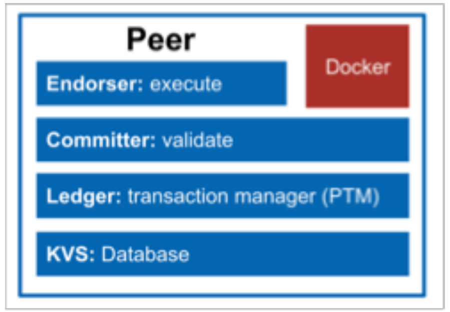
Client

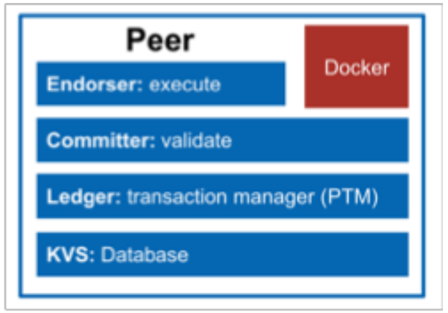
Client

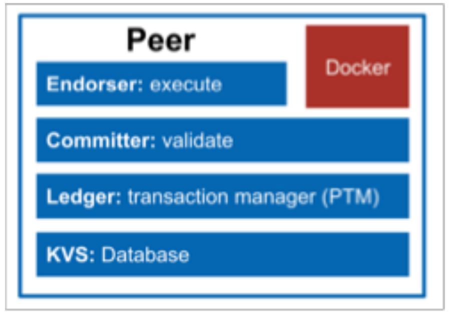
Client

Client









Client

Client

Client

Client

Gossip

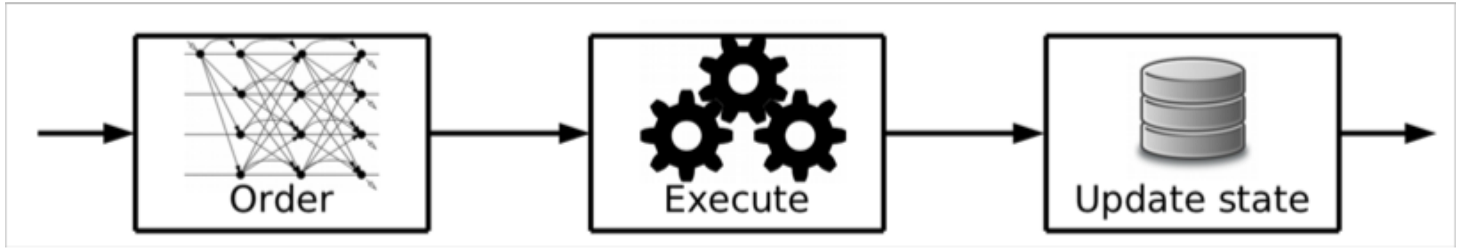
Order service

Fabric Building blocks

- Store: CouchDB / LevelDB
- Chaincode: Go, Java, Node.js, …
- Docker containers
- gRPC
- Gossip: push/pull methods

- Orderer
  - Apache Kafka (ZooKeeper)
  - Byzantine Fault Tolerant (BFT) orderer
  - Solo (centralized) for development

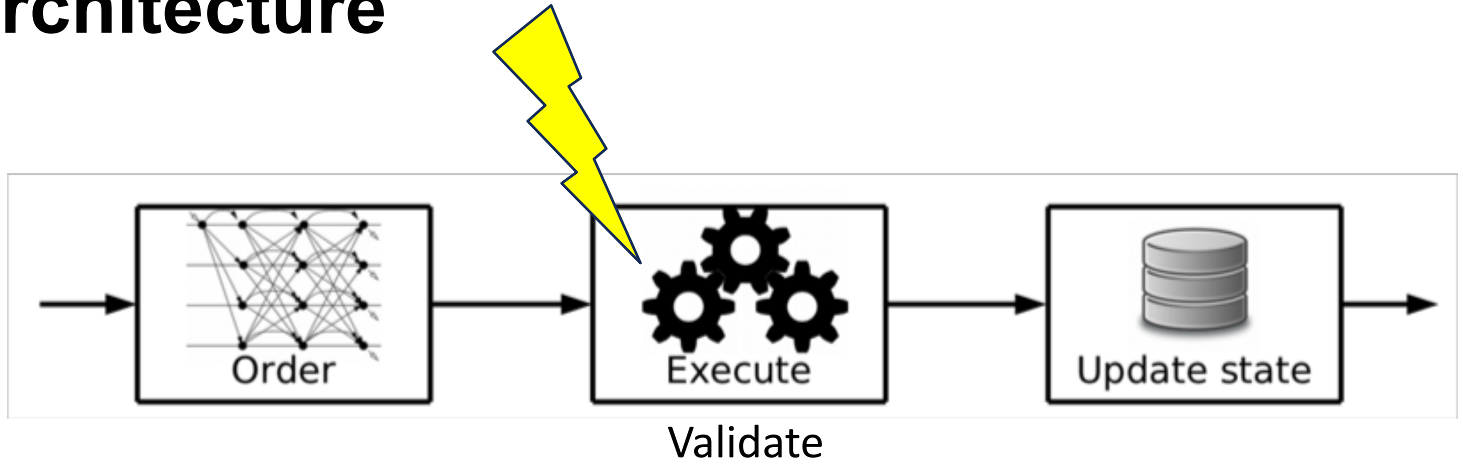
Traditional Architecture



Validate

- Order by longest chain or BFT
- Execute smart contracts on all peers
- State updates on all peers → Ledger

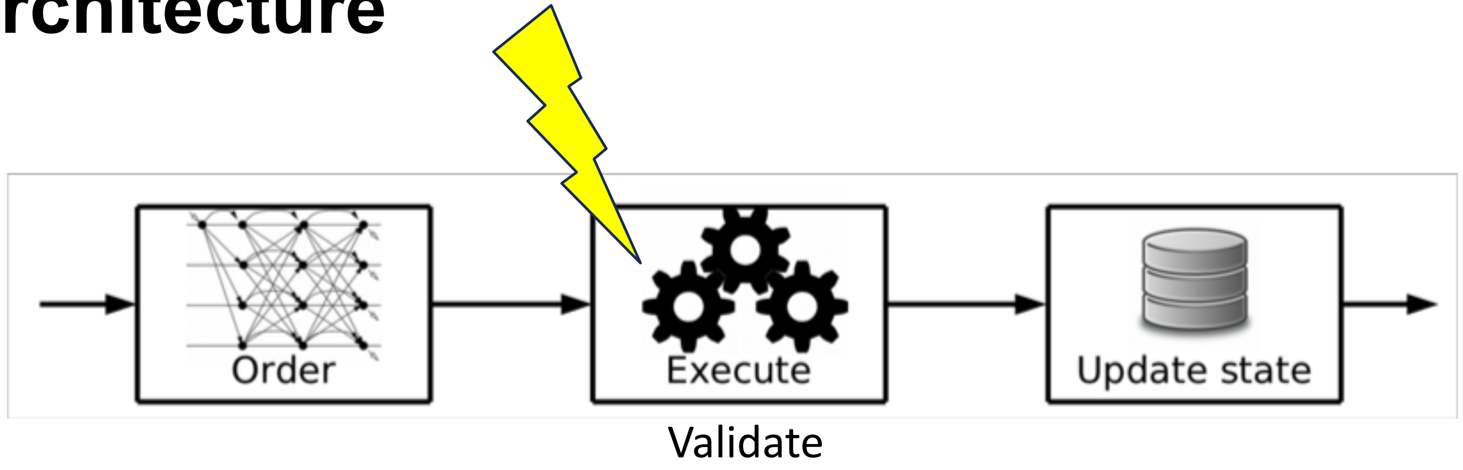
Traditional Architecture



Problem

- Sequential execution of all contracts on all peers → bottleneck

Traditional Architecture



Problems

- Sequential execution of all contracts on all peers → bottleneck
- Programs MUST be deterministic → NO general purpose languages

Deterministic?

```go
package main

import (
    "fmt"
)

func main() {
    m := []int{1, 2, 3, 4}

    for _, v := range m {
        fmt.Println( a: "Value:", v)
    }
}
```

Deterministic?

```go
package main

import (
	"fmt"
)

func main() {
	m := []int{1, 2, 3, 4}

	for _, v := range m {
		fmt.Println( a: "Value:", v)
	}
}
```

Deterministic?

```go
package main

import (
    "fmt"
)

func main() {
    m := map[int]string{1:"one", 2:"two", 3:"three", 4:"four"}

    for k, v := range m {
        fmt.Println("Key:", k, "Value:", v)
    }
}
```

Deterministic?

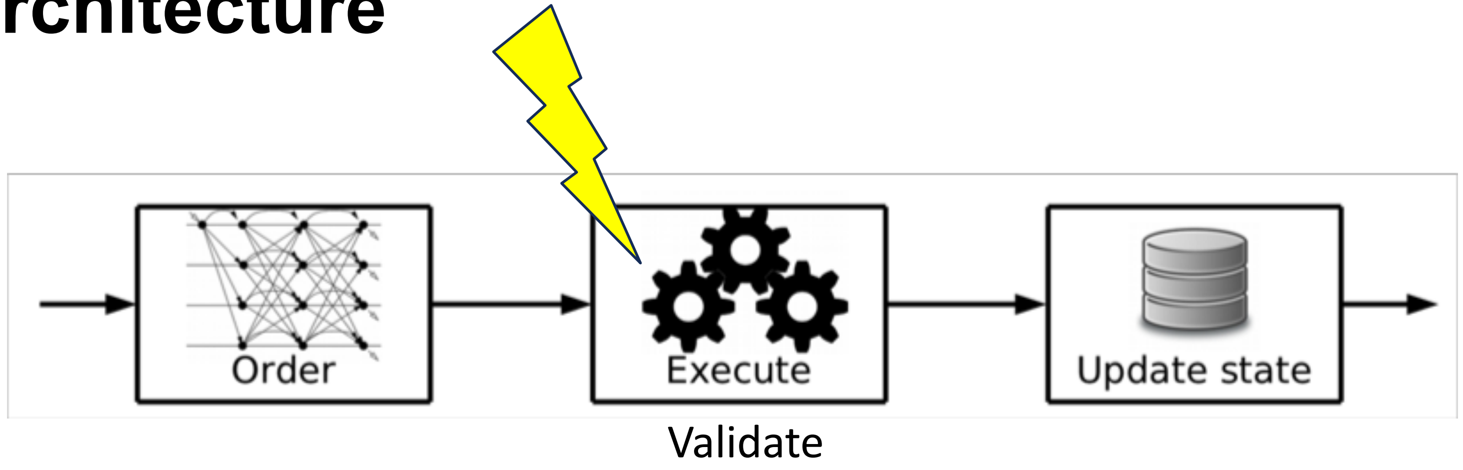
```go
package main

import (
    "fmt"
)

func main() {
    m := map[int]string{1:"one", 2:"two", 3:"three", 4:"four"}

    for k, v := range m {
        fmt.Println( a: "Key:", k, "Value:", v)
    }
}
```

```
Key: 4 Value: four
Key: 1 Value: one
Key: 2 Value: two
Key: 3 Value: three
```

```
Key: 1 Value: one
Key: 2 Value: two
Key: 3 Value: three
Key: 4 Value: four
```

```
Key: 3 Value: three
Key: 4 Value: four
Key: 1 Value: one
Key: 2 Value: two
```

Traditional Architecture

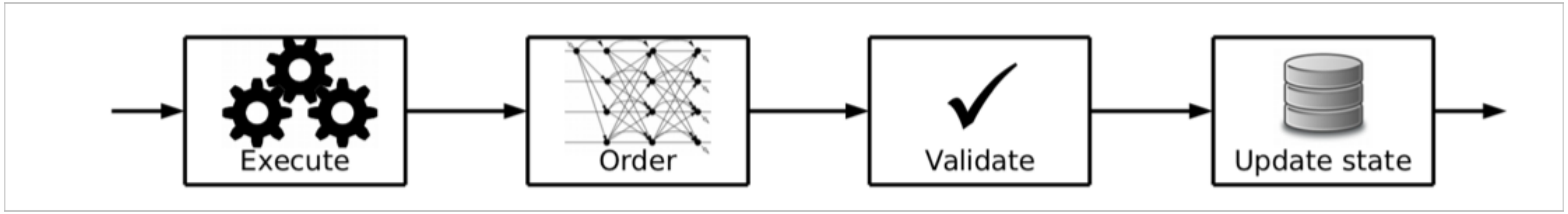


Problems

- Sequential execution of all contracts on all peers → bottleneck
- Programs MUST be deterministic → NO general purpose languages

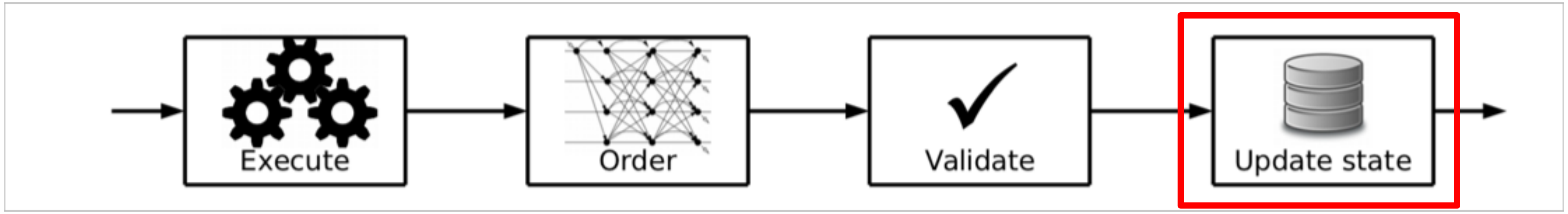
ETH zürich

Fabric Architecture



Key insight

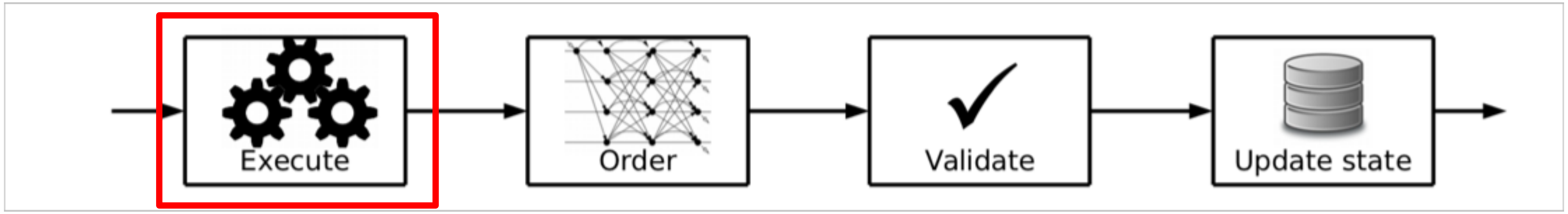
Fabric Architecture



State

- Versioned key-value store
- Maintained on all peers

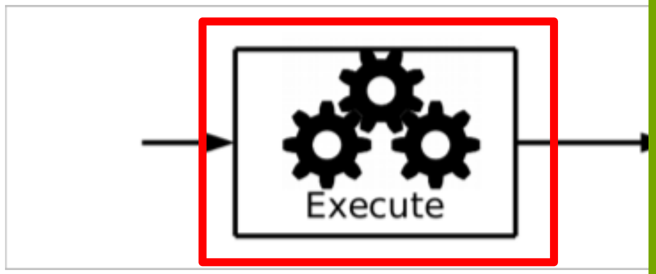
Fabric Architecture



Execute

- Only some peers are executing the chaincode (simulation)
- Use current local state
- Create read-set and write-set for access of versioned key-value store
- Create signed “endorsement”

Fabric Architecture



Key insight

State must be replicated on all peers, not execution

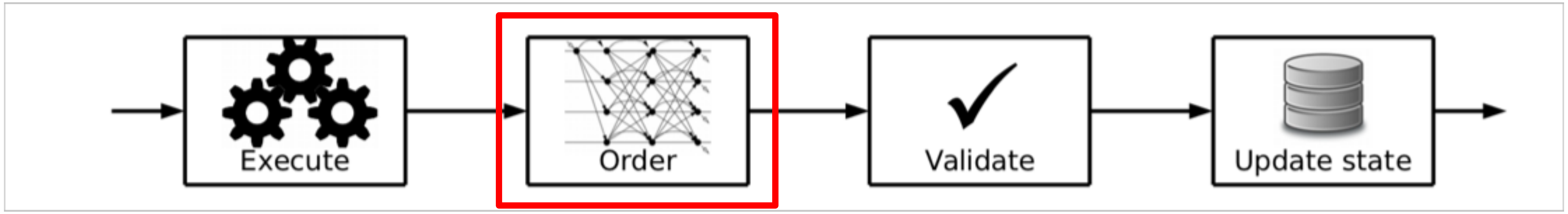
Sequential execution in O(n) instead of O(N)

n << N
N = computing steps
n = size of read and write sets

Execute

- Only some peers are executing the chaincode (simulation)
- Use current local state
- Create read-set and write-set for access of versioned key-value store
- Create signed "endorsement"

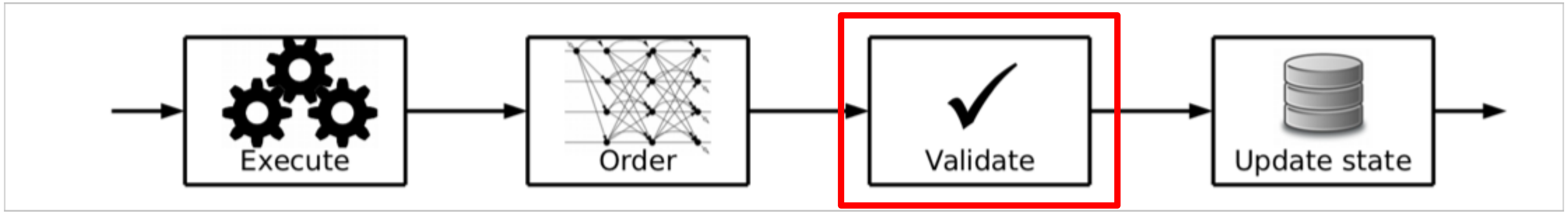
Fabric Architecture



Order

- Needs enough endorsements with identical read-/write-sets
- Uses Apache Kafka, BFT or other methods
- Peer gossip

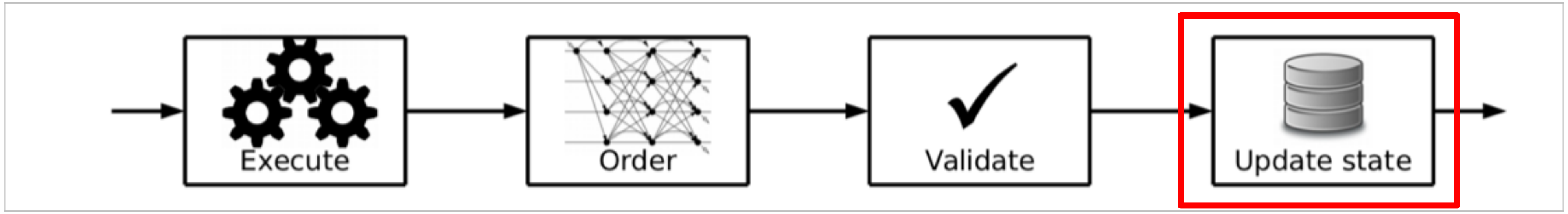
Fabric Architecture



Validate

- Parallel
- All peers validate correctness of transaction based on policy
- NO execution of the chaincode

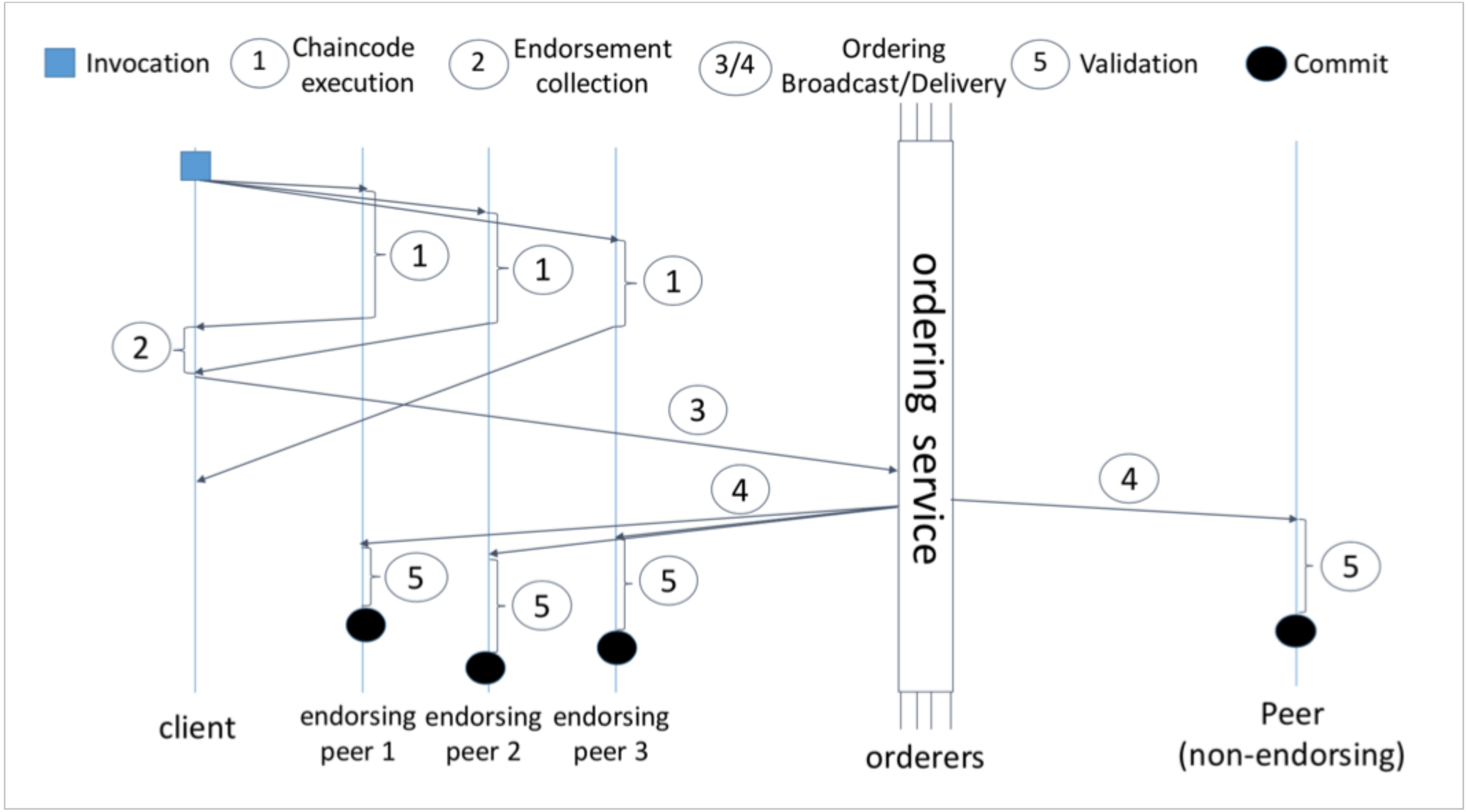
Fabric Architecture



Update state

- sequential
- Peer transaction manager (PTM)
- Checks again versions of the keys in readset
  mismatch → invalidate transaction

Transaction flow



Fabric Components

Membership service provider (MSP)

Policies

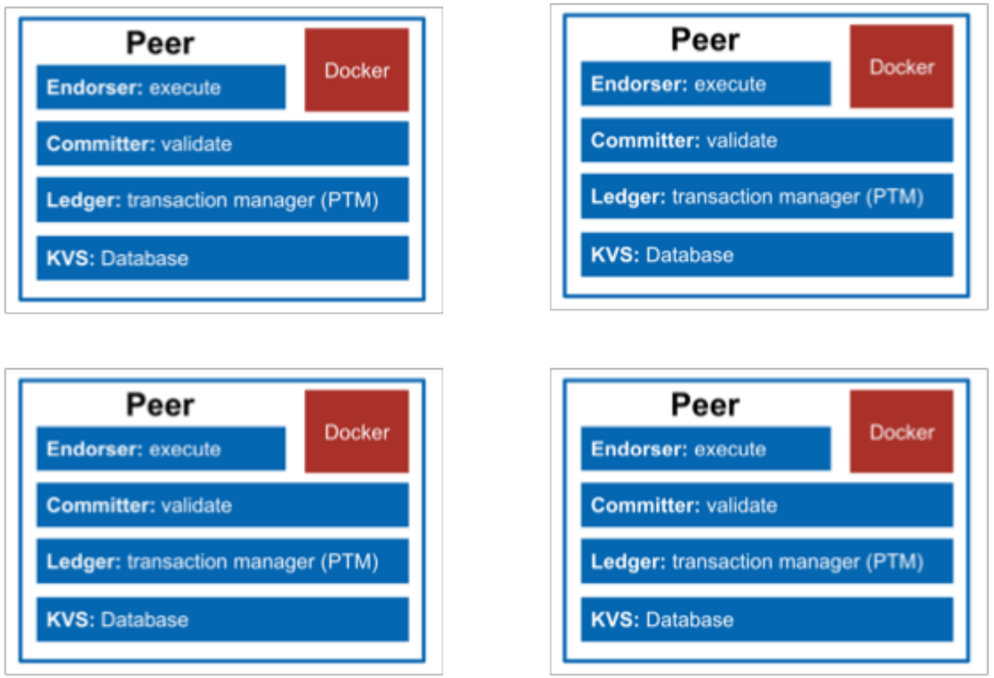
Chaincode

Client

Client

Client

Client



Client

Client

Client

Client

Gossip

Order service

Policy

- Number of endorsements
- Which endorser shall be used
- Execution limitations
- Validation rules

- Parallel chaincode execution
- Confidential chaincode

Security

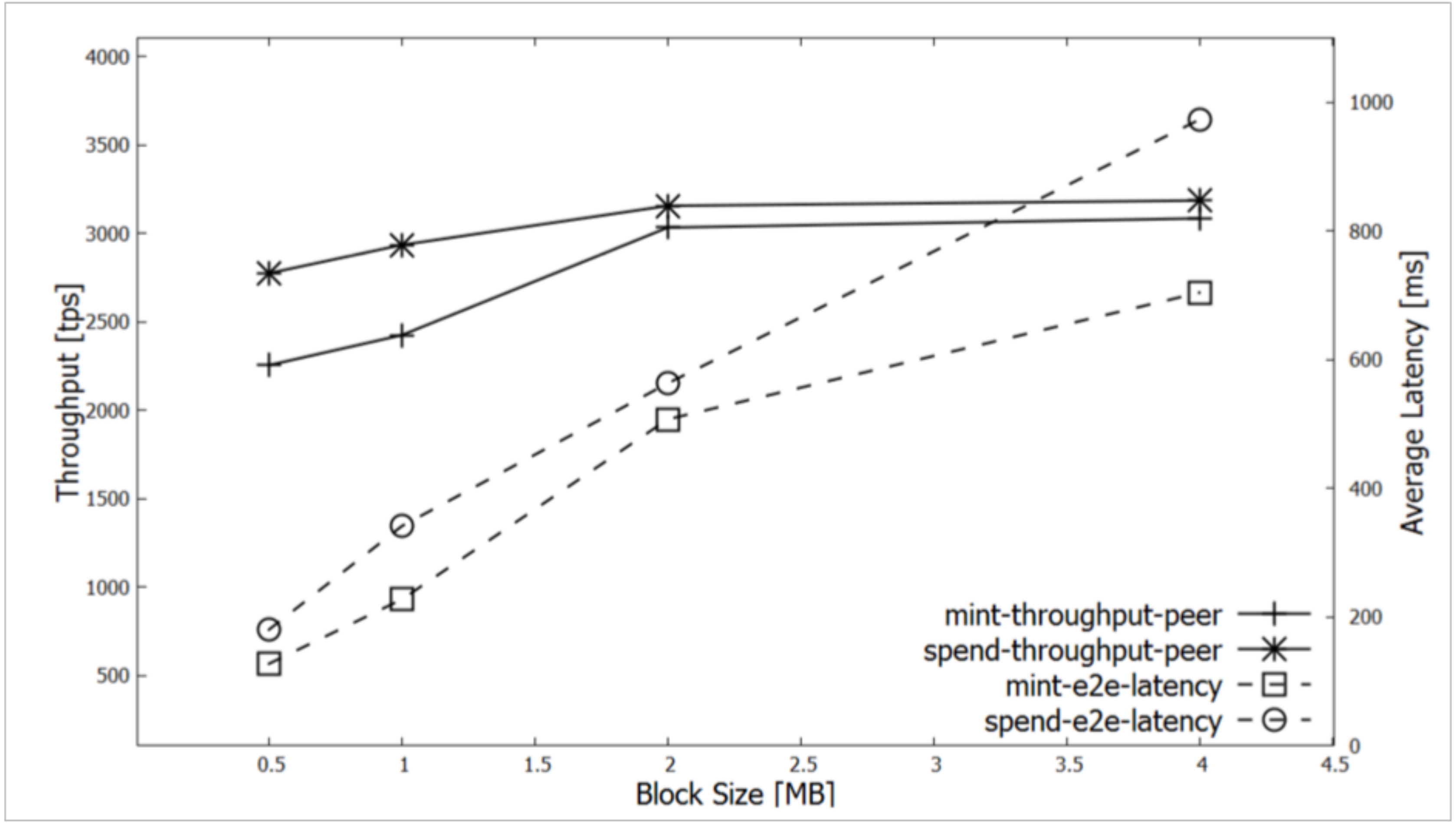
- TLS for communication
- Classic membership service
- Signatures
- Docker for sandboxing

- Complex system
- Dependency on many 3rd party codes

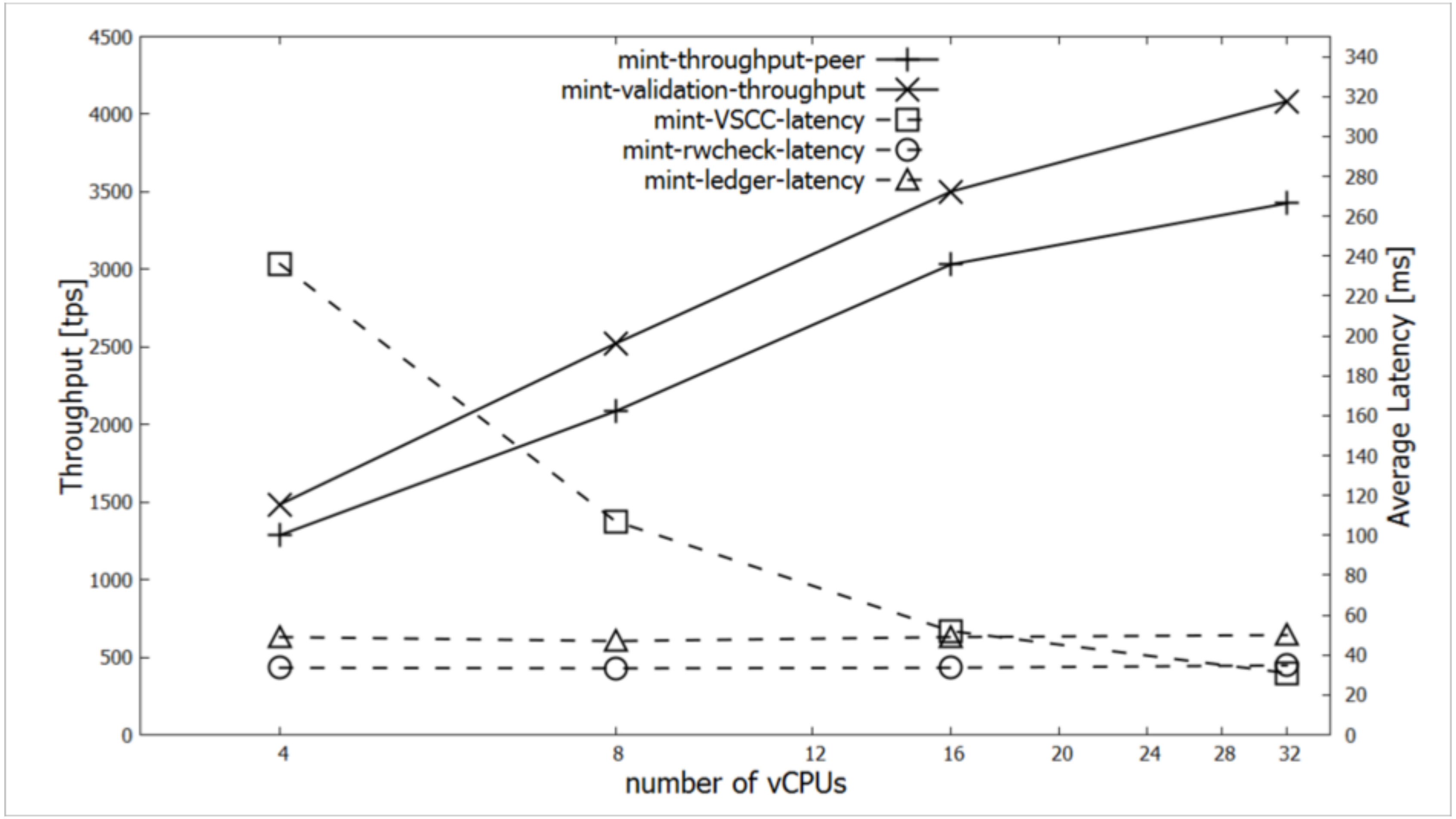
Evaluation

- Fabcoin: UTXO
- VMs in one data center
- 2.0 GHz 16 vCPU VMs running Ubuntu with 8 GiB RAM and SSDs
- 1Gbps networking connections
- Orderer: Kafka with 3 ZooKeeper nodes, 4 Kafka brokers, 3 Fabric orderers
- 5 peers, all Fabcoin endorsers
- TLS for all connections
- Signatures with 256-bit ECDA scheme
- Node clocks synchronized by NTP
- MINT phase / SPEND phase

Block size


Throughput [tps]
Average Latency [ms]
Block Size [MB]
4000
3500
3000
2500
2000
1500
1000
500
1000
800
600
400
200
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
mint-throughput-peer
spend-throughput-peer
mint-e2e-latency
spend-e2e-latency

Scales with number of vCPUs



Latency in detail

	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	484 / 624	523 / 636
(3) VSCC val.	31.0 / 35.3	10.2 / 9.0	72.7 / 57.0	113 / 108.4
(4) R/W check	34.8 / 61.5	3.9 / 9.3	47.0 / 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

Latency in detail

	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	484 / 624	523 / 636
(3) VSCC val.	31.0 / 35.3	10.2 / 9.0	72.7 / 57.0	113 / 108.4
(4) R/W check	34.8 / 61.5	3.9 / 9.3	47.0 / 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

Latency in detail

	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	484 / 624	523 / 636
(3) VSCC val.	31.0 / 35.3	10.2 / 9.0	72.7 / 57.0	113 / 108.4
(4) R/W check	34.8 / 61.5	3.9 / 9.3	47.0 / 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

Latency in detail

	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	484 / 624	523 / 636
(3) VSCC val.	31.0 / 35.3	10.2 / 9.0	72.7 / 57.0	113 / 108.4
(4) R/W check	34.8 / 61.5	3.9 / 9.3	47.0 / 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

Conclusion

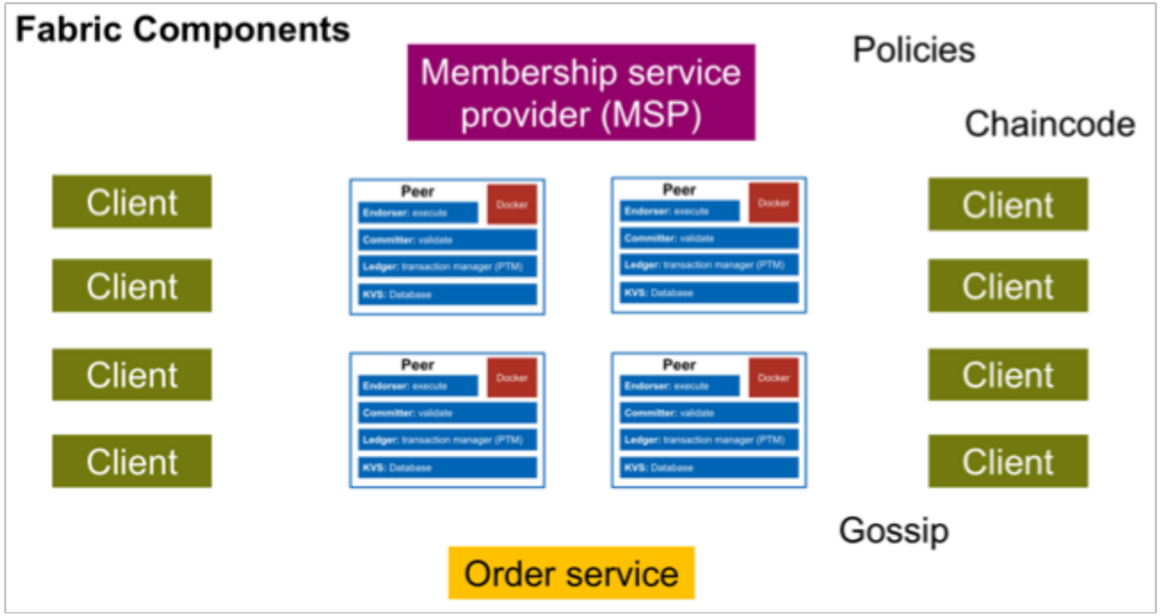
Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains

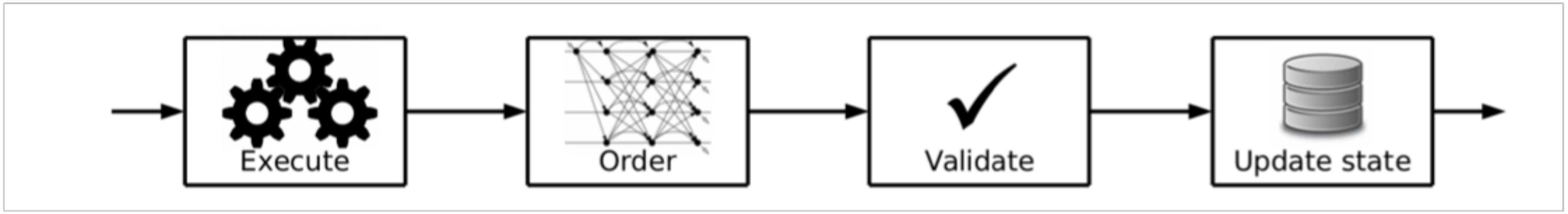
Elli Androulaki   Artem Barger   Vita Bortnikov   Christian Cachin   Konstantinos Christidis   Angelo De Caro   David Enyeart   Christopher Ferris   Gennady Laventman   Yacov Manevich   Srinivasan Muralidharan *   Chet Murthy †   Binh Nguyen *   Manish Sethi   Gari Singh   Keith Smith   Alessandro Sorniotti   Chrysoula Stathakopoulou   Marko Vukolić   Sharon Weed Cocco   Jason Yellick

IBM

<EURO/SYS'18>

April 23-26, 2018, Porto, Portugal





Reserve slides for questions

Blockchain use cases

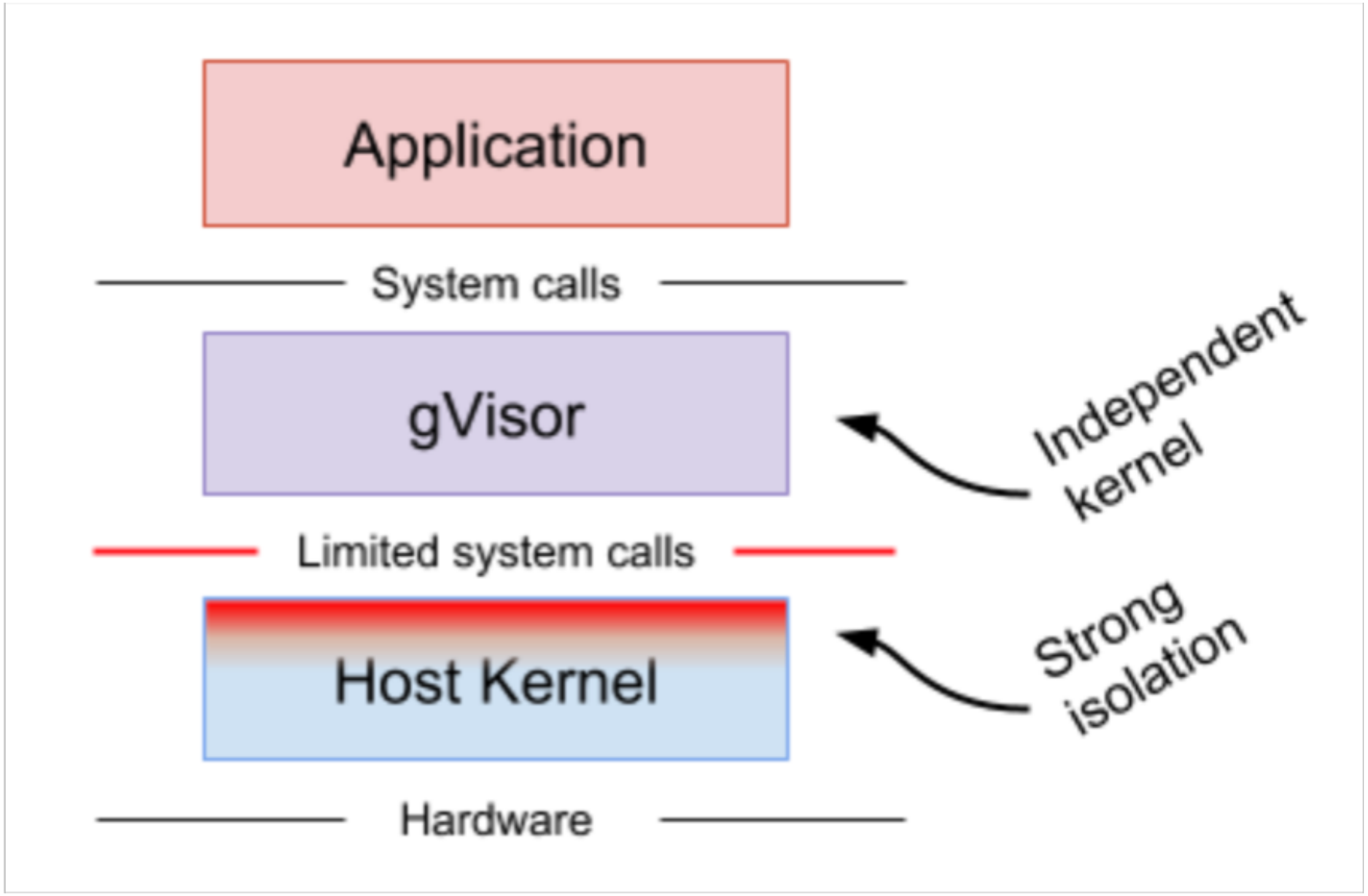
- Food-safety network
- Global shipping trade
- Enterprise asset management
- Foreign exchange netting
- Global cross-currency payments

- One size does not fit all

Modules: allow step-wise improvements

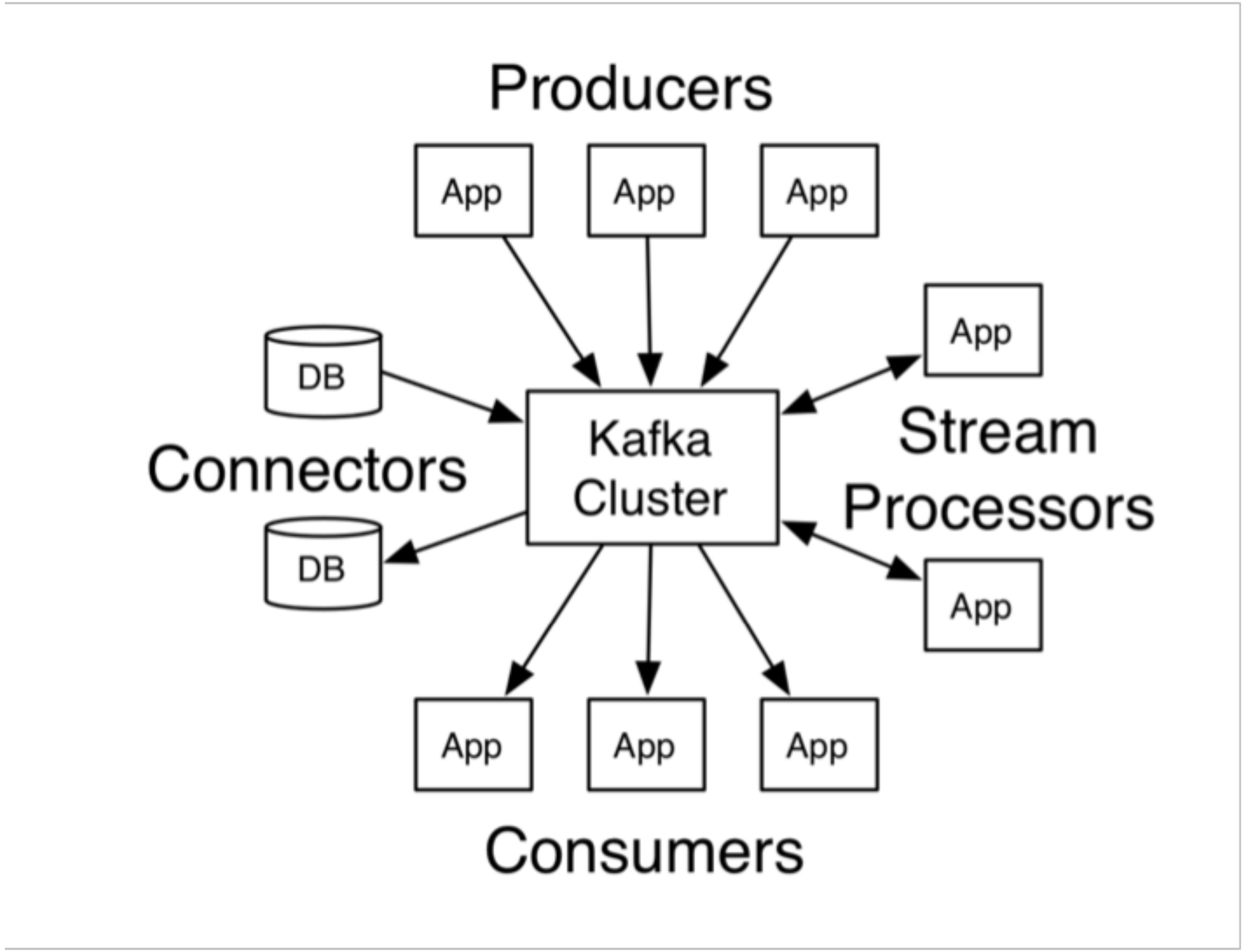
- **Docker:** container but not actually sandbox
  Google just presented gVisor these days
  → improved security
- **Orderer:** Currently weak part of the system
  → improved distributed BFT based order is being built
- **Execution / Validation:** Can be extended to various policies and advancements in research
- **Storage:** Improved DBs / KVS if available

Google gVisor: available for docker



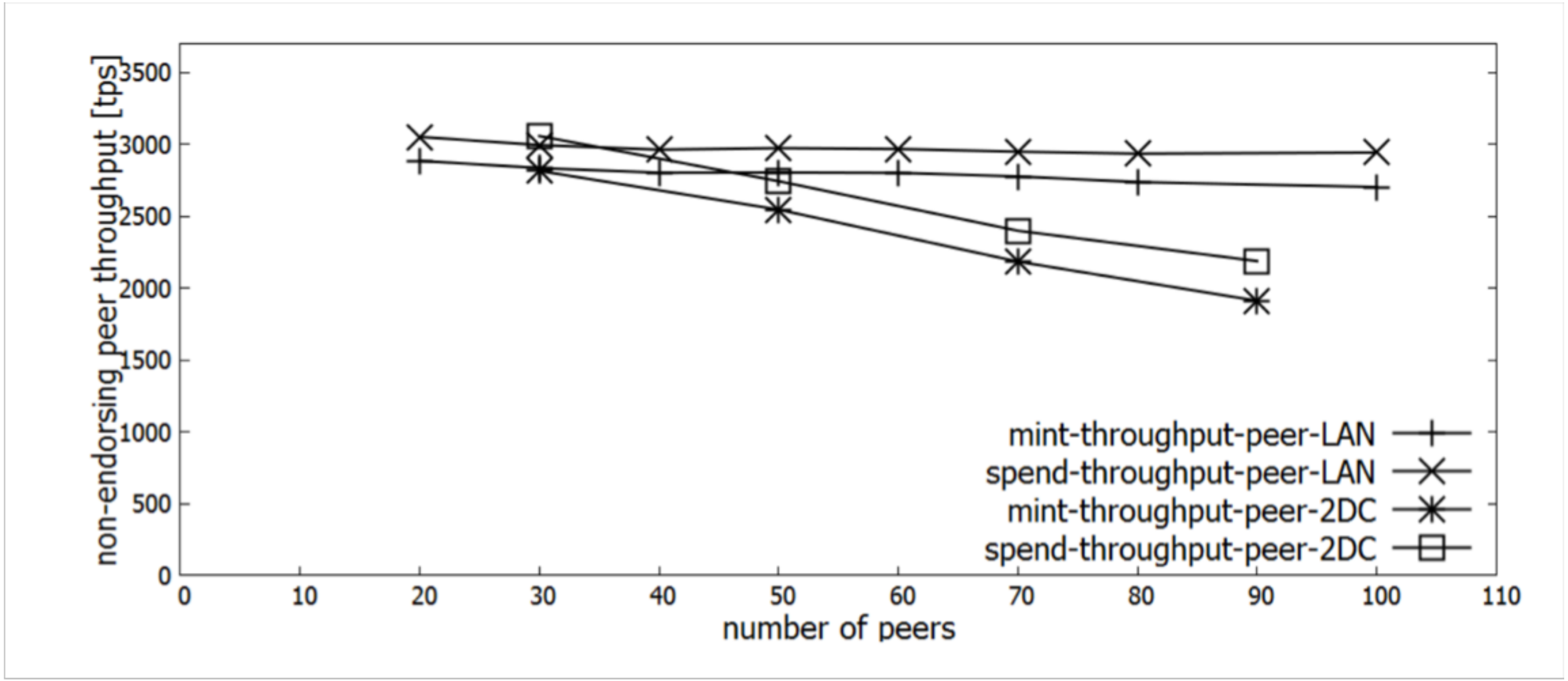
picture from github.com/google/gvisor

Apache Kafka: a distributed streaming platform



picture from kafka.apache.org

Number of peers



Distance between 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	1389 / 1838
peak MINT / SPEND throughput [tps] (with gossip)	2553 / 2762	2558 / 2763	2271 / 2409	1484 / 2013

- 100 peers across 5 data centers