UC Santa Barbara

Computer Science Department

ParBlockchain: Leveraging Transaction Parallelism in Permissioned Blockchain Systems

Mohammad Javad Amiri, Divyakant Agrawal, Amr El Abbadi

The 39th IEEE International Conference on Distributed Computing Systems (ICDCS)

China Aims to Use Blockchain

Nike, Telegram, Facebook, and Everyone Else Suddenly Love Blockchain

by P. H. Madore — 25/04/2019 in Bitcoin & Blockchain Investments, Blockchain News, Cryptocurrency Ne
3 min read

How important will blockchain be to the world's economy?

By Tim Harford
Presenter, 50 Things That Made the Modern Economy

3 July 2019

Share

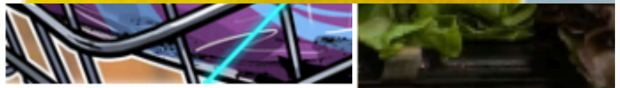
BBC

GETTY IMAGES

CNN BUSINESS   Markets   Tech   Me

3 key things to know about Facebook's Libra cryptocurrency project

By Clare Duffy, CNN Business
Updated 3:38 PM ET, Tue June 18, 2019

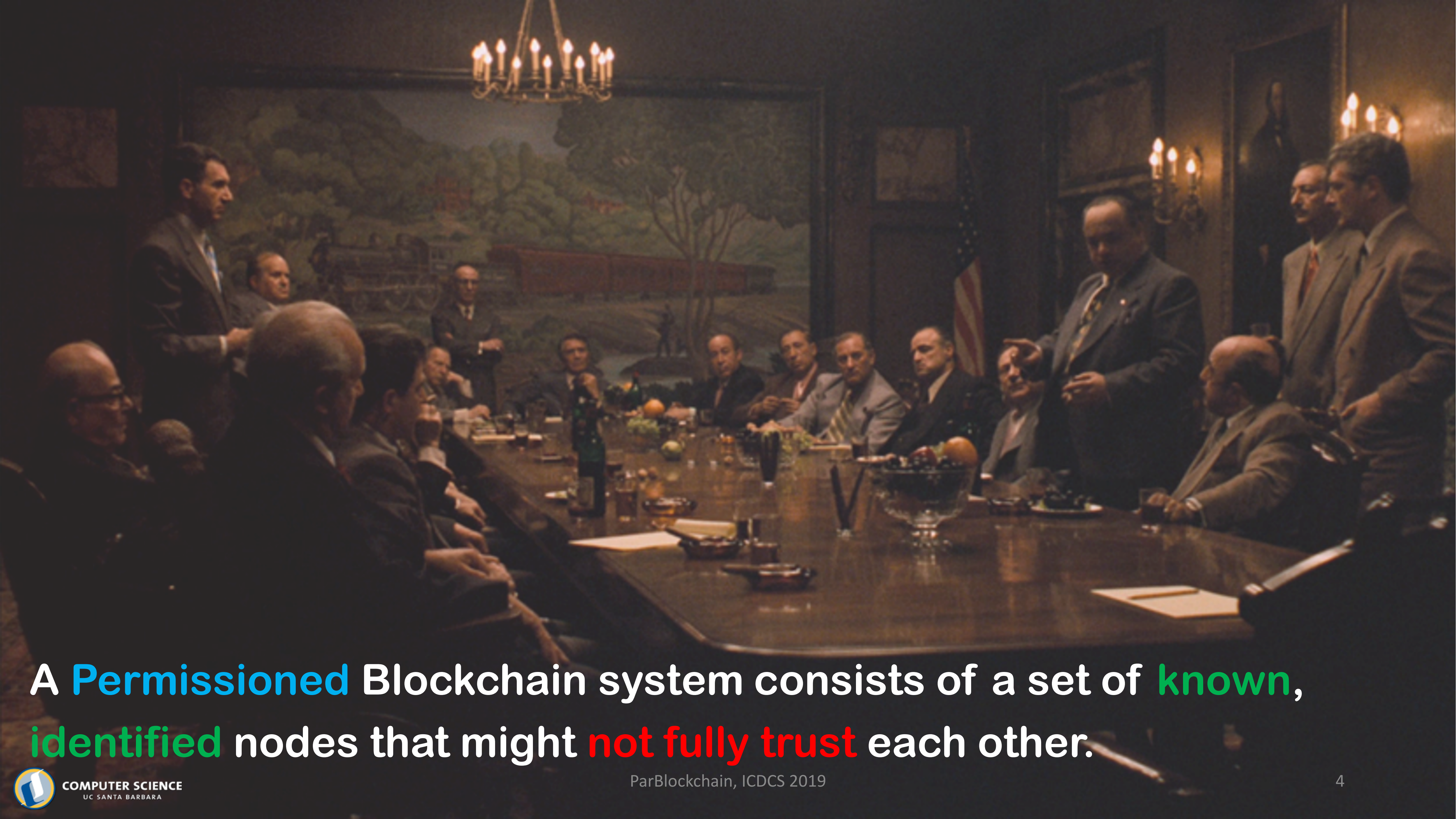


Anyone can participate without a specific identity

Participants are known and Identified

Permissionless Blockchain

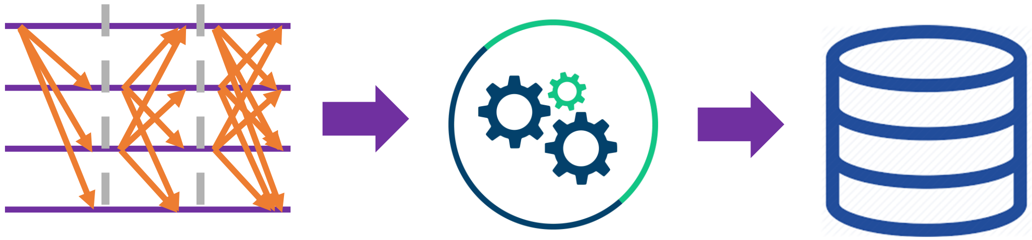
Permissioned Blockchain



A Permissioned Blockchain system consists of a set of known, identified nodes that might not fully trust each other.

Order-Execute (OX) Architecture

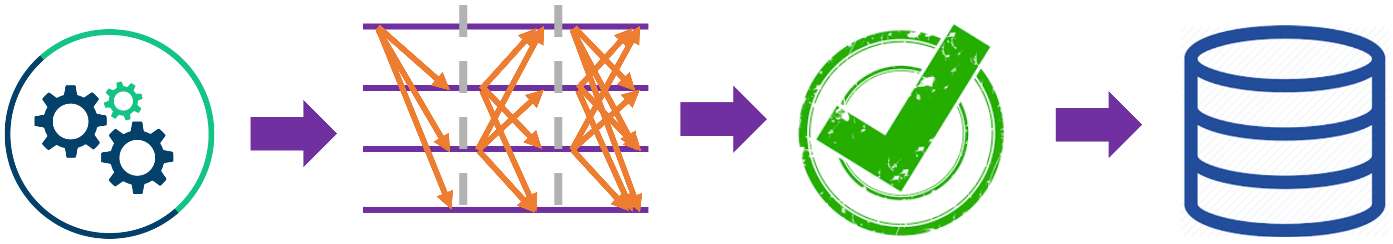
- A set of nodes (might be all of them) orders transactions (using a consensus protocol), constructs blocks, and multicasts them to all the nodes
- Each node then executes the transactions and updates the blockchain ledger



- Limitations of Order-Execute
  - Sequential execution: Transactions are sequentially executed on all peers (*performance bottleneck*)
  - Non-deterministic code: any non-deterministic execution results in "*fork*" in the distributed ledger
  - Confidentiality of execution: all smart contracts run on all peers!
    - Smart contract: A computer program that *self-executes* once it is established and deployed

Execute-Order-Validate (XOV) Architecture

- Each transaction is first executed by a subset of nodes (endorsers)
- A separate set of nodes (orderers) orders the transactions, puts them into blocks, and multicasts them to all the nodes.
- Each node validates the transactions and updates the ledger



- Supports non-deterministic execution
- Executes transactions in parallel
- Preserves confidentiality of smart contracts

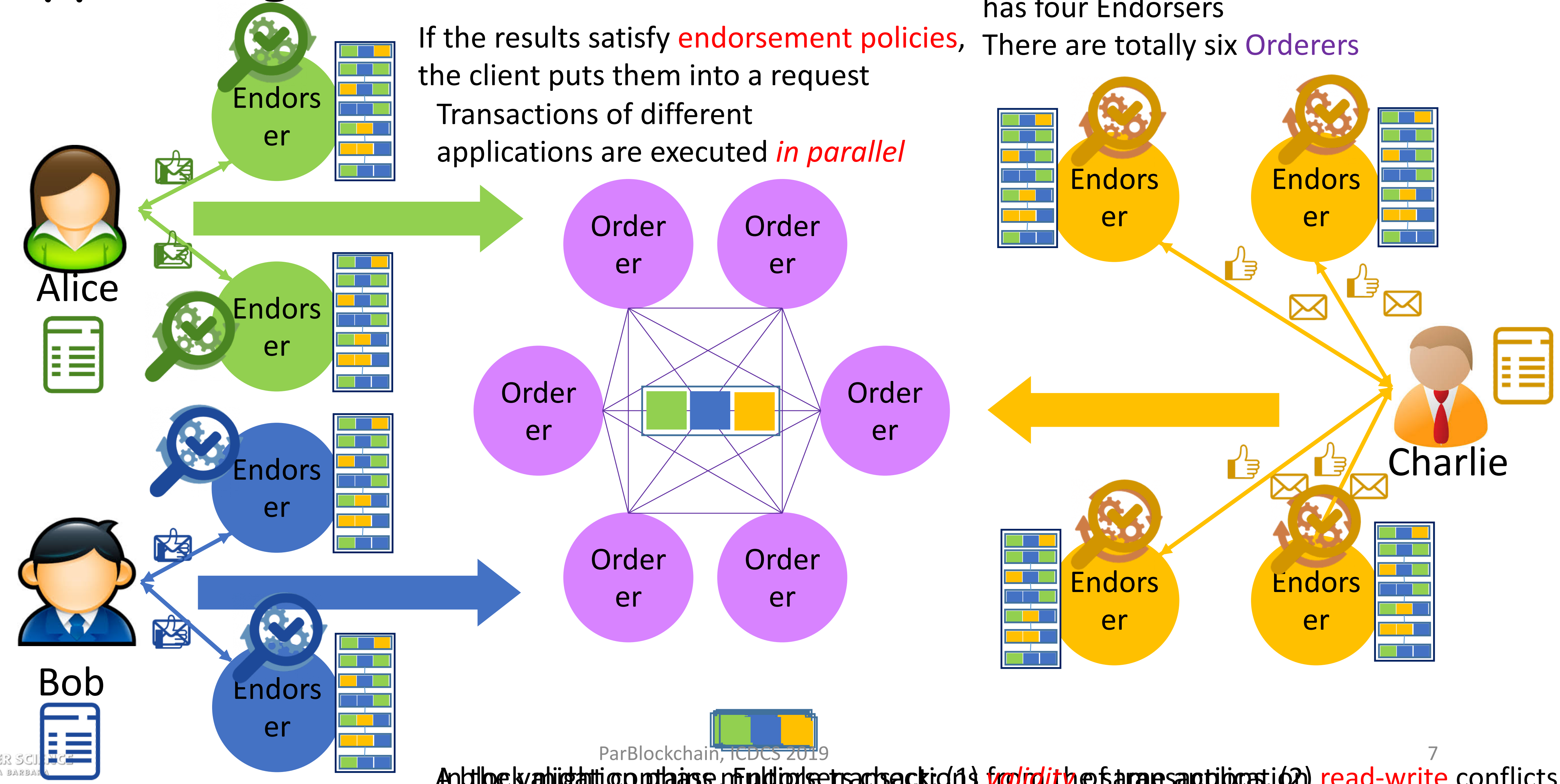
Hyperledger Fabric

IBM HYPERLEDGER

Three Applications (Green, Blue, Yellow)
Three Clients (Alice, Bob, Charlie)
Green and Blue have two Endorsers, Yellow has four Endorsers
There are totally six Orderers

If the results satisfy endorsement policies, the client puts them into a request

Transactions of different applications are executed *in parallel*



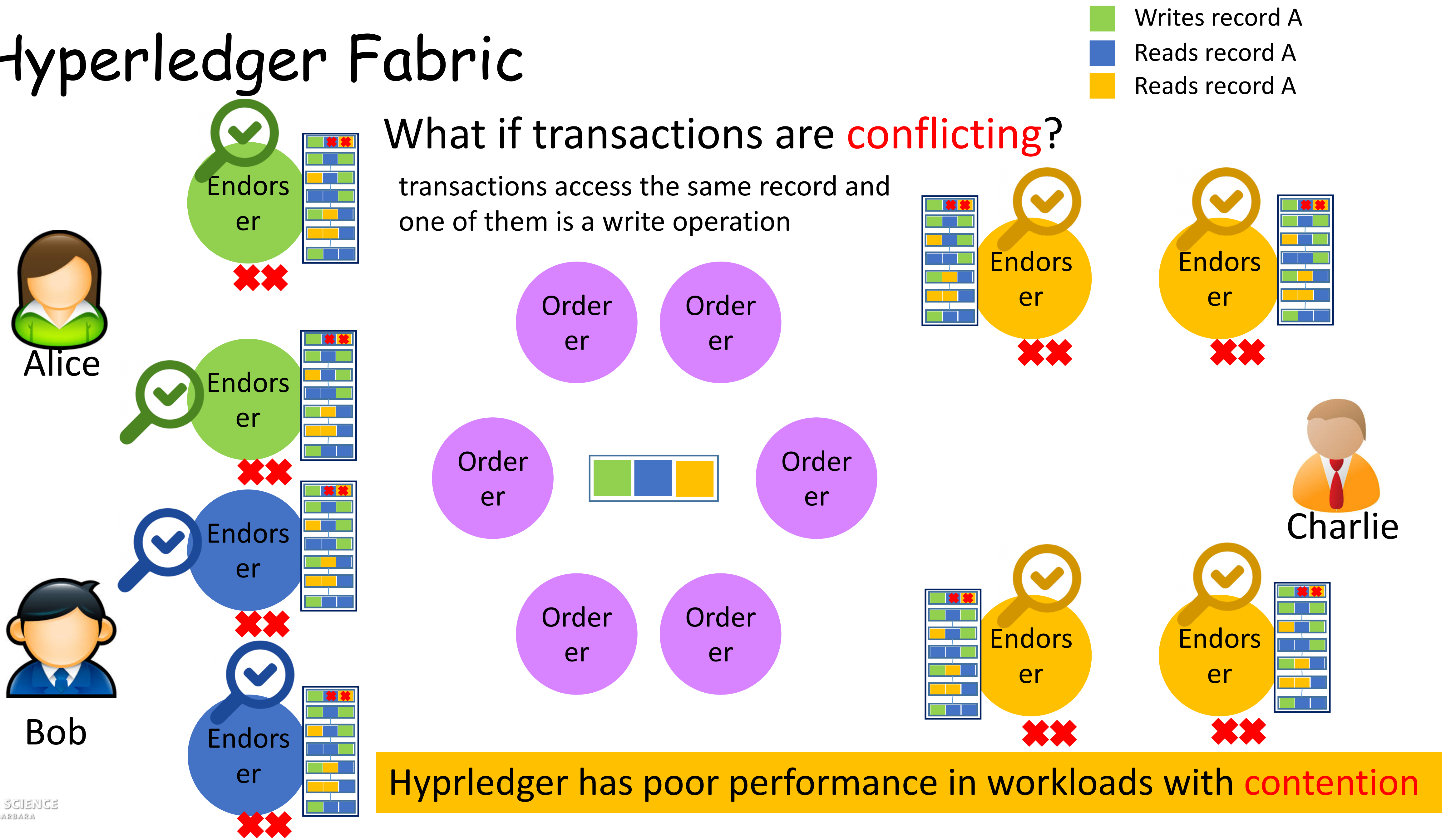
A block might contain multiple transactions from the same application

In the validation phase, Endorsers check: (1) *validity* of transactions, (2) read-write conflicts

Hyperledger Fabric

What if transactions are conflicting?

transactions access the same record and one of them is a write operation



Hyprledger has poor performance in workloads with contention

COMPUTER SCIENCE
UC SANTA BARBARA

Dependency Graph

- Ordering Dependency ($T_i \succ T_j$): ts(j) > ts(i) and one of the following:

$\rho(T_i) \cap \omega(T_j) \neq \emptyset$ $\quad\quad$ $\omega(T_i) \cap \rho(T_j) \neq \emptyset$ $\quad\quad$ $\omega(T_i) \cap \omega(T_j) \neq \emptyset$

- Dependency graph: exposes ordering dependencies (conflicts) between transactions to give a partial order of transactions.

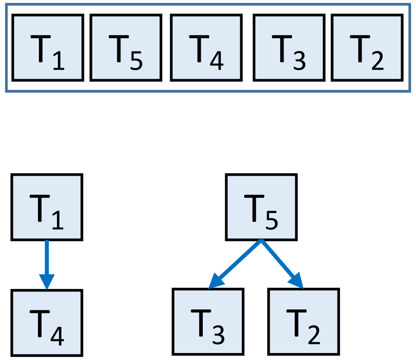
T_1: $\rho = \{a\}$, $\omega = \{a,b\}$

T_2: $\rho = \{f\}$, $\omega = \{d\}$

T_3: $\rho = \{f\}$, $\omega = \{e\}$

T_4: $\rho = \{b\}$, $\omega = \{c\}$

T_5: $\rho = \{e\}$, $\omega = \{d\}$



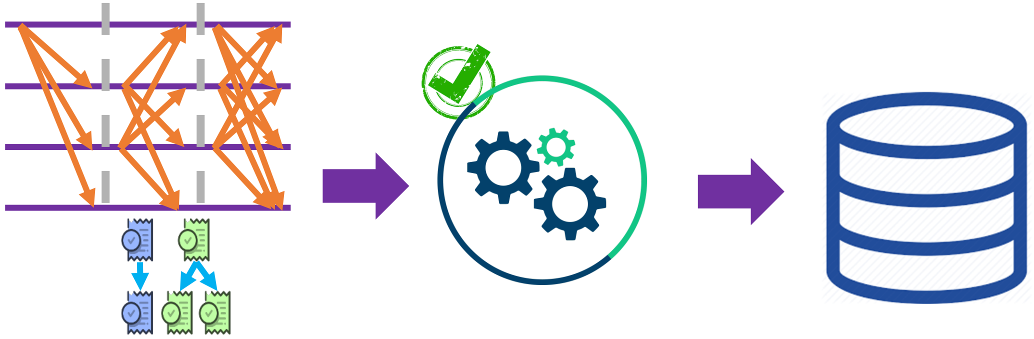
T_4 reads b that is written by T_1

T_3 writes e that is read by T_5

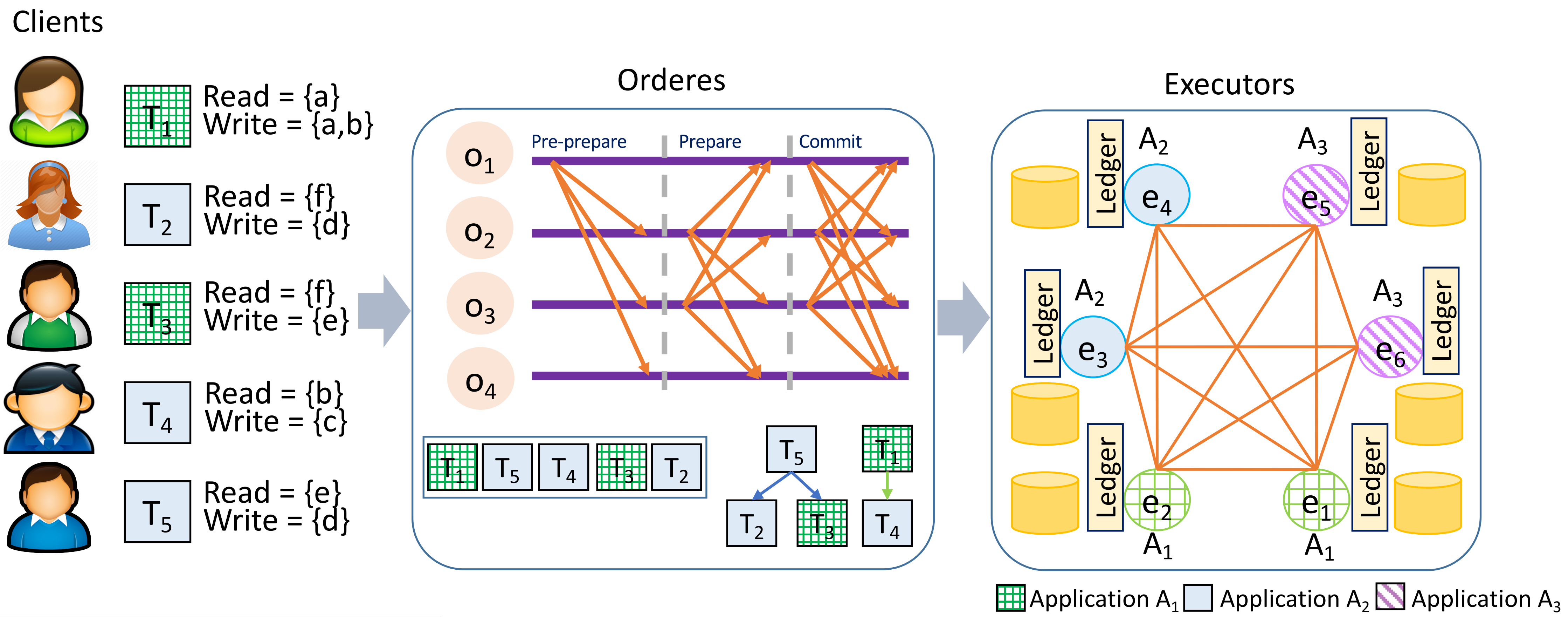
T_2 writes d that is written by T_5

Order-Parallel Execute (OXII) Architecture

- A separate set of nodes (orderers) orders the transactions, puts them into blocks, generates a *dependency graph* for the block, and multicasts it to all the nodes
- Each transaction (of an application) is then validated and executed by a subset of nodes (agents of the application) following the dependency graph
- The nodes multicast the results of execution and append the block



ParBlockchain Overview

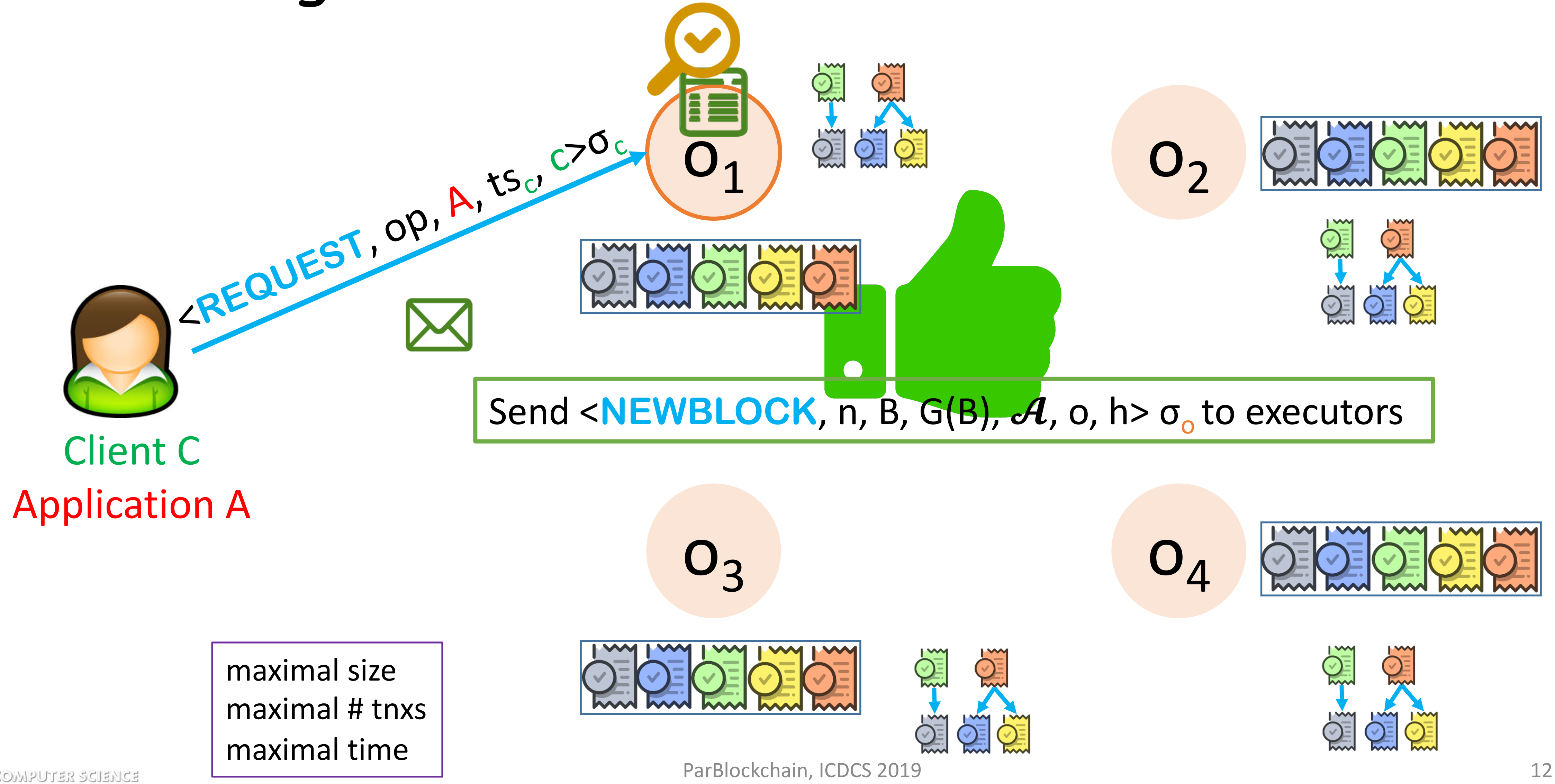


Each transaction of an applicatio
include records to be read and w

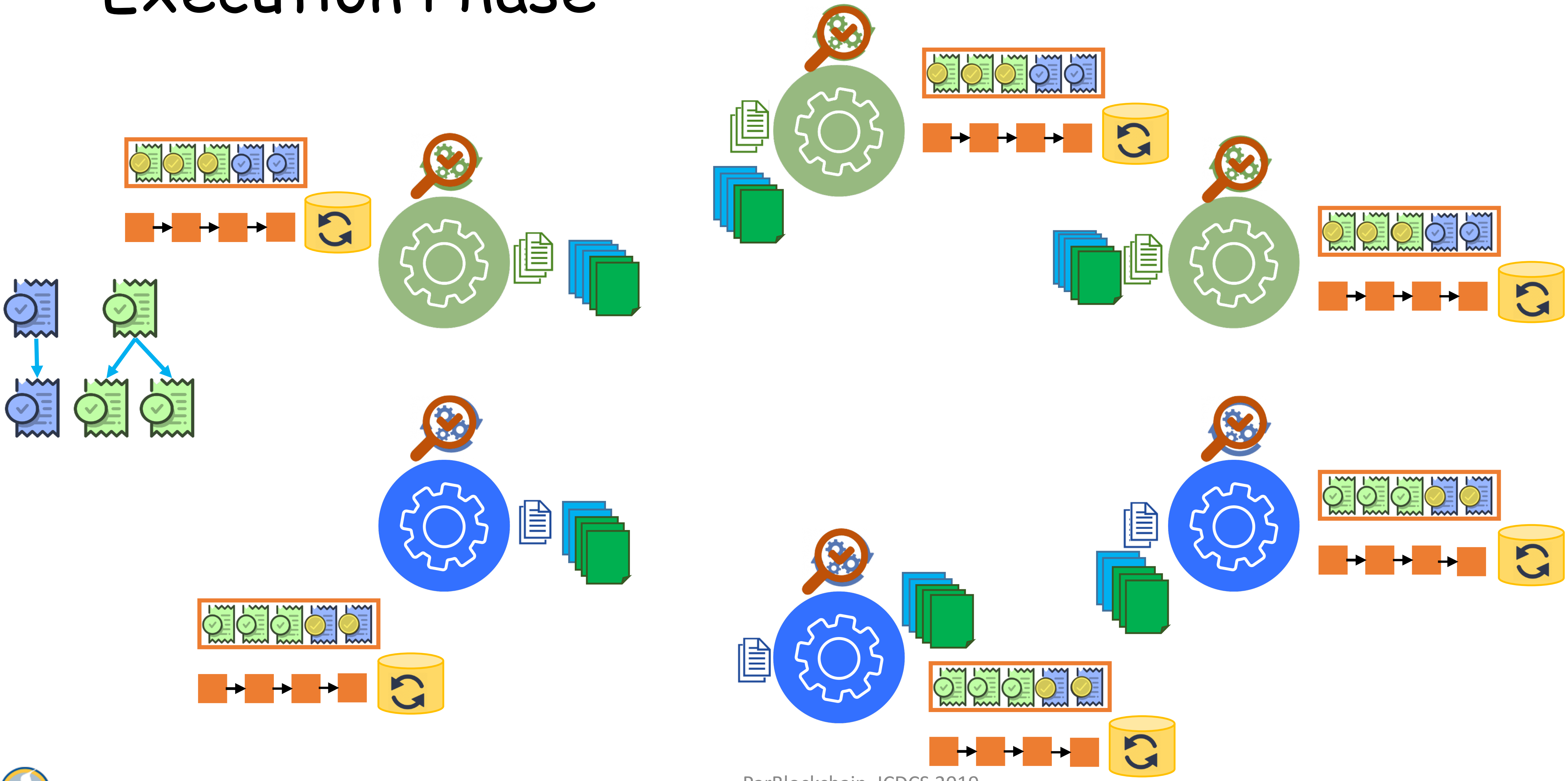
Each orderer generates a dependency graph
for the block and multicasts it to all Executors

Executors of each application execute the
corresponding transactions following the
dependency graph and multicast the results

Ordering Phase



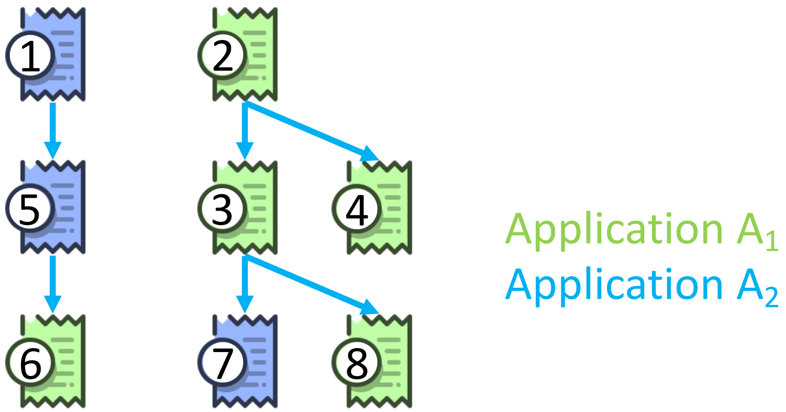
Execution Phase



Execution Phase

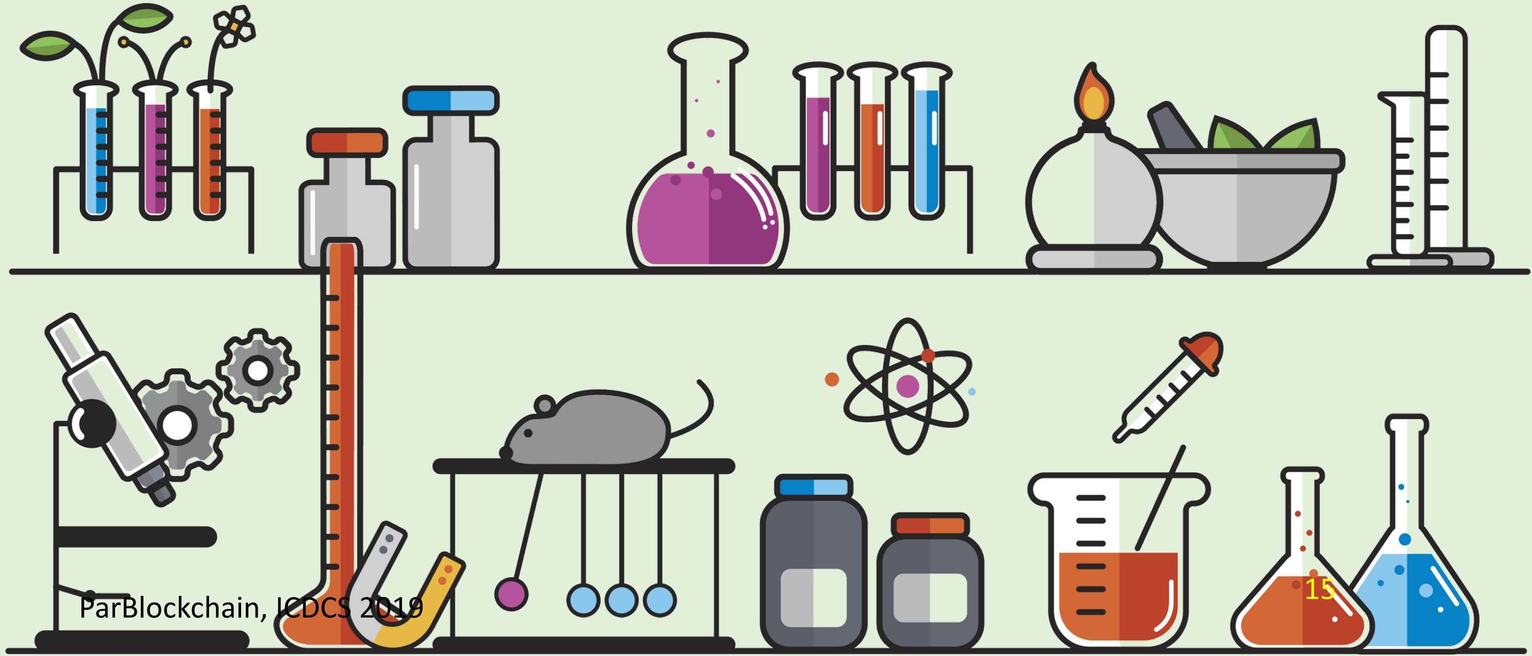
- What if there is some dependency between transactions of different applications?
  - The agents of those applications cannot execute the transactions independently
- Solutions:

1. Send messages as soon as the execution of each transaction is completed
   - The number of exchanged messages will be large
2. Send messages when the execution results are needed by some other application



Experimental Settings

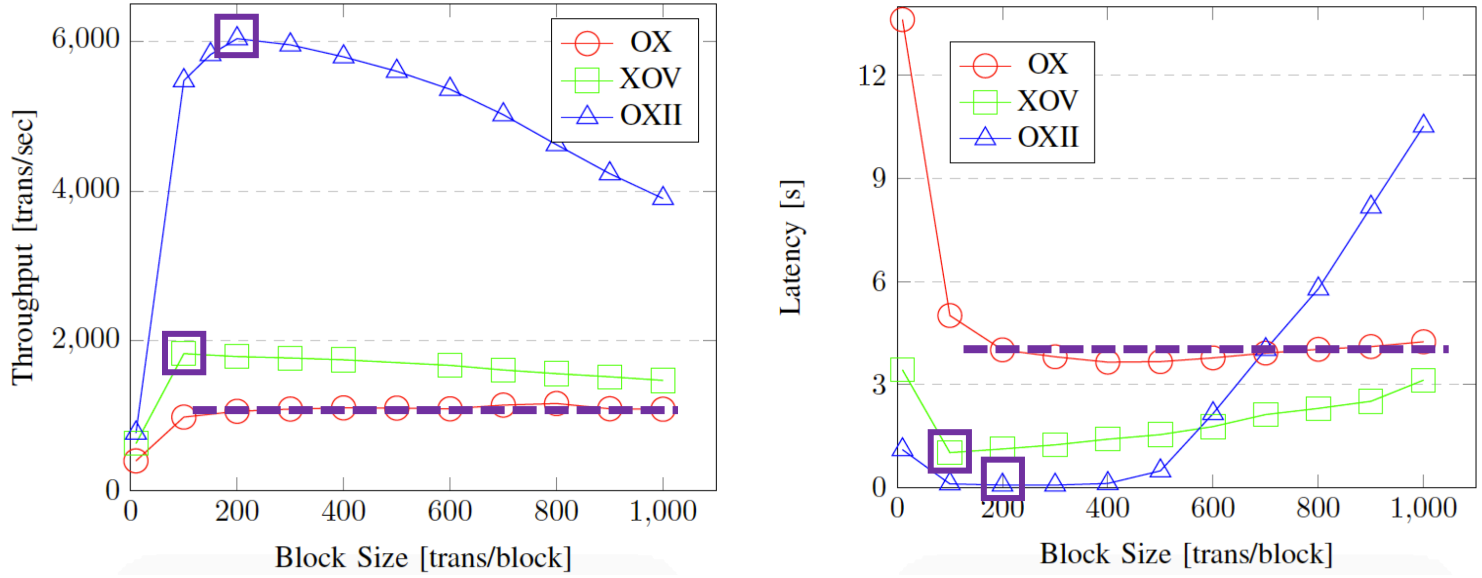
- Systems:
  - OX: Sequential Order-Execute
  - XOV: Execute-Order-Validate (XOV) [HyperLedger Fabric]
  - OXII: Order-Parallel Execute [ParBlockchain]
- Applications: Accounting
- Platform: Amazon EC2
- Measuring performance
  - Throughput
  - Latency



Choosing the Block Size

6000 transactions
78 ms latency

1800 transactions
1024 ms latency

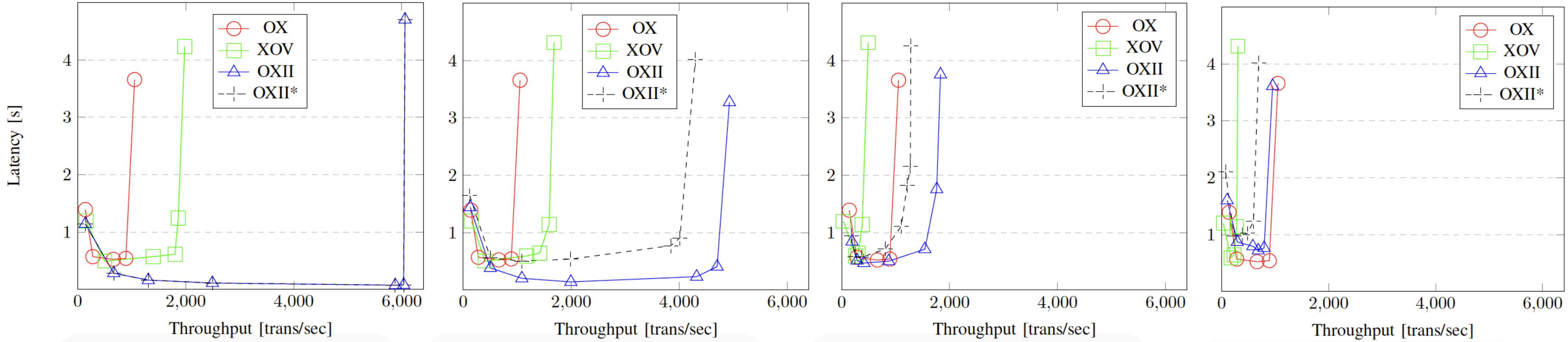


Any further increasing of the block size reduce the performance due to the large number of required computations for the dependency graph generation

Since nodes execute transactions sequentially, the block creation time is negligible in comparison to the execution time

Performance in Workloads with Contention

No Contention (0%) | Low-contention (20%) | High-contention (80%) | Full-contention (100%)



The performance of OX remains unchanged in different workloads (sequential execution)

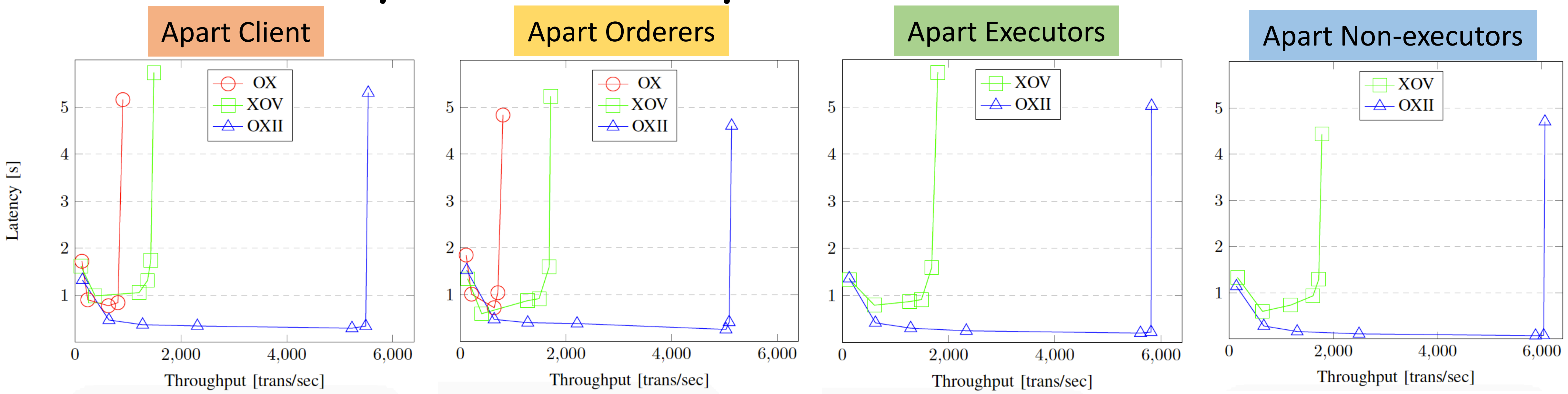
OXII processes more than 6000 transactions with latency less than 80 ms (220% more than XOV with 16% latency)

Processing the workloads with contention across the applications decreases the performance of OXII

The peak throughput of XOV in a high-contention workload is 25% of its peak throughput in a no-contention workload

The performance of OXII is a bit worse than OX because of the dependency graph generation overhead

Scalability over Multiple Data Centers



Each time we move one group of nodes to AWS Asia Pacific (Tokyo) Region data center, leaving the other nodes in the AWS US West Region (California) data center (the RTT between these two data centers is 113 ms).

Moving the clients has the most impact on the XOV paradigm because in XOV clients participate in the first two phases

Orderers are the core part of all three blockchains

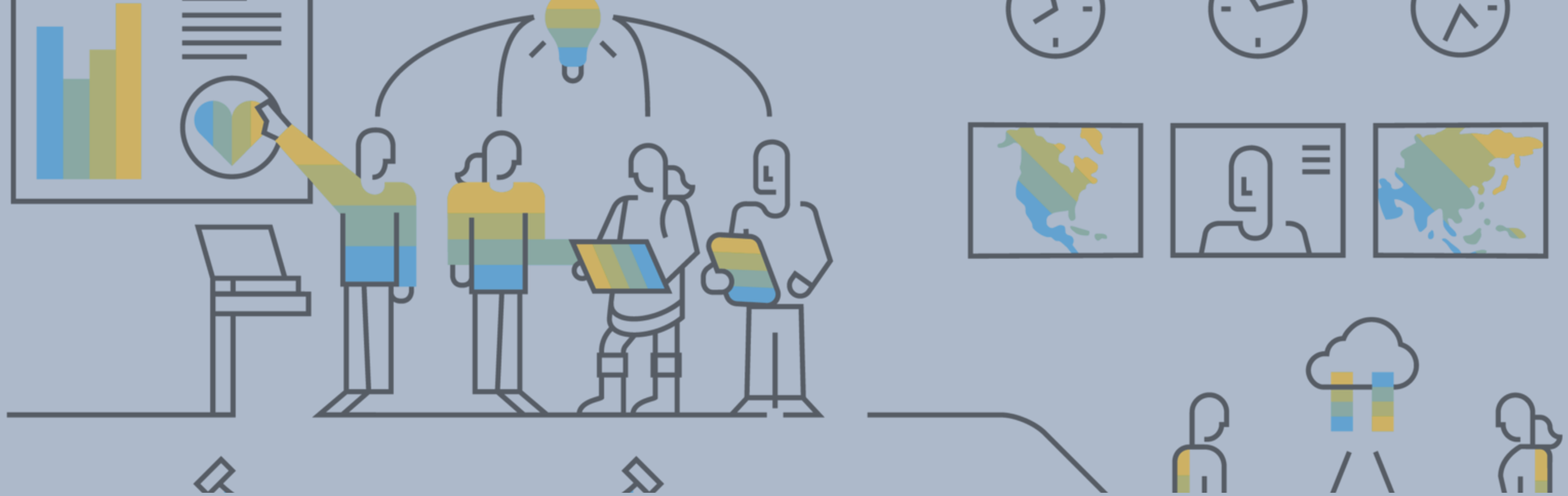
Moving executor nodes adds latency to the two phases of communication in XOV and one phase of communication in OXII

Non-executor nodes are only informed about the blockchain state in OXII but validate the blocks in XOV

Optimistic vs. Pessimistic Execution

Two ways to look at the problem!

	Supporting non-deterministic execution	Supporting High Contention Workloads
Hyperledger Fabric	Executes first (does not submit transactions with inconsistent results)	Validates read-write conflicts last (aborts conflicting transactions)
ParBlcockchain	Validates non-determinist execution last (aborts transactions with inconsistent results)	Checks conflicts first (generates a dependency graph)



Future Work

- Supporting both non-deterministic execution and High Contention Workloads
- Preserving the confidentiality of the blockchain ledger
- Enhancing the scalability of Blockchains using the sharding techniques

THANK YOU!

Questions?!