UniStore: A fault-tolerant marriage of causal and strong consistency

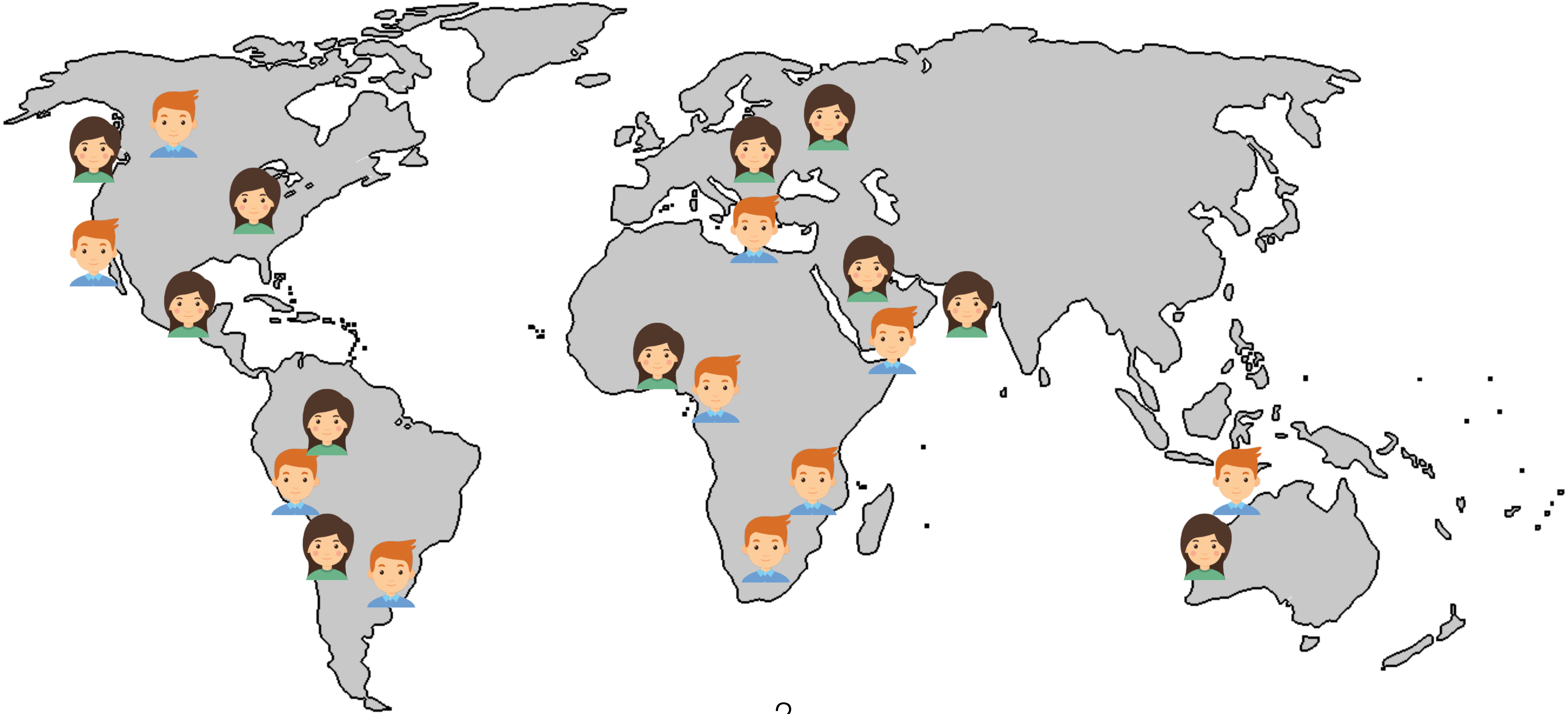
Manuel Bravo

IMDEA Software Institute, Madrid, Spain

Joint work with Alexey Gotsman, Borja de Régil (IMDEA) and Hengfeng Wei (Nanjing University)

Geo-replication

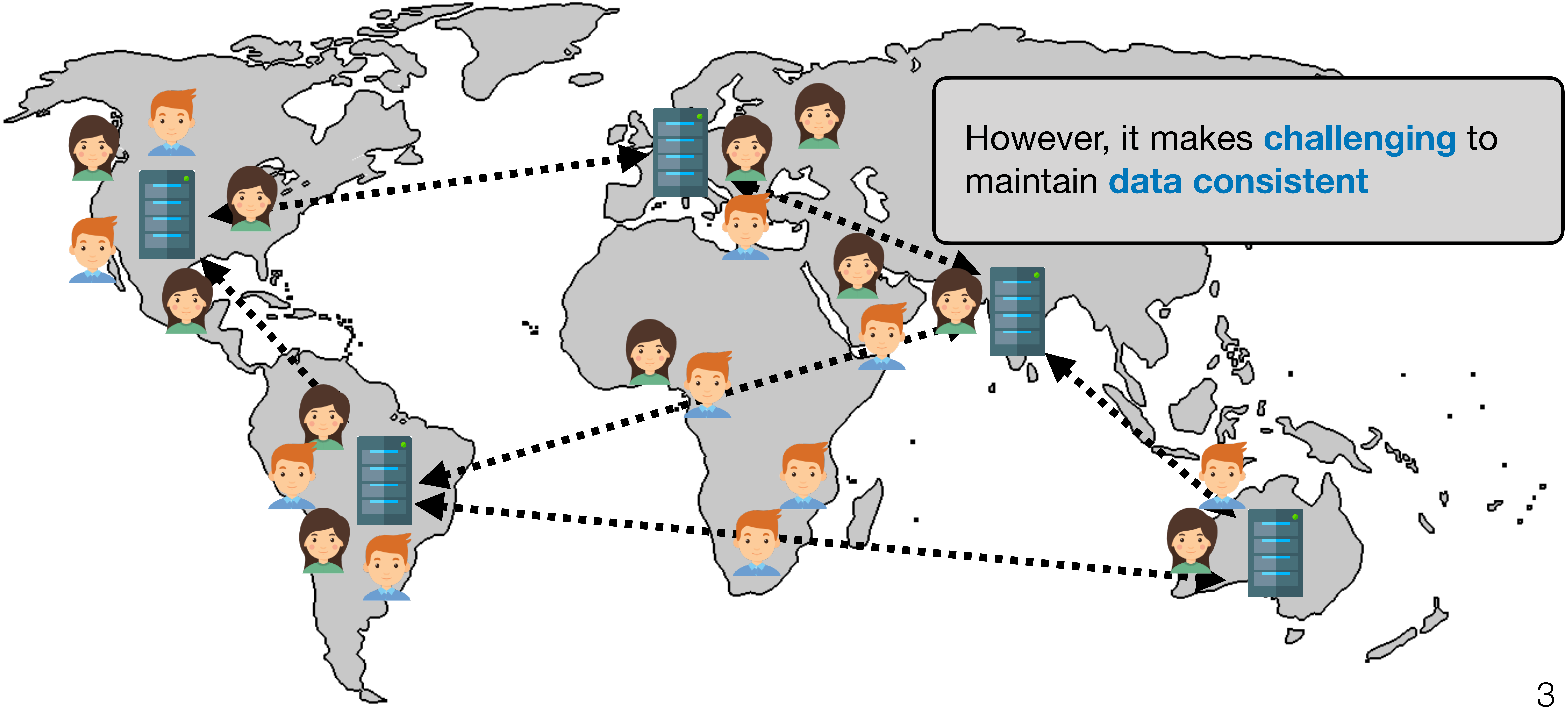
Geo-replication



Geo-replication

This **improves user experience** by allowing accesses to the closest site and ensures disaster-tolerance

Geo-replication



However, it makes **challenging** to maintain **data consistent**

Fundamental trade-offs

Fundamental trade-offs

strong consistency

Fundamental trade-offs

strong consistency

✓ makes replication transparent

Fundamental trade-offs

strong consistency

✓ makes replication transparent

✗ high response time:
synchronization critical path

Fundamental trade-offs

strong consistency

- ✓ makes replication transparent
- ✗ high response time:
  synchronization critical path
- ✗ unavailable during network partitions

Fundamental trade-offs

strong consistency

weak consistency

- ✓ makes replication transparent
- ✗ high response time:
  synchronization critical path
- ✗ unavailable during network
  partitions

Fundamental trade-offs

strong consistency

weak consistency

✓ makes replication transparent

✗ high response time: synchronization critical path

✗ unavailable during network partitions

low response time ✓

Fundamental trade-offs

strong consistency

weak consistency

- ✓ makes replication transparent
- ✗ high response time: synchronization critical path
- ✗ unavailable during network partitions

- low response time ✓
- highly-available ✓

Fundamental trade-offs

strong consistency

- ✓ makes replication transparent
- ✗ high response time: synchronization critical path
- ✗ unavailable during network partitions

weak consistency

- low response time ✓
- highly-available ✓
- unable to preserve critical application invariants ✗

The hybrid alternative

The hybrid alternative

- To allow **multiple consistency levels** to coexist

The hybrid alternative

- To allow **multiple consistency levels** to coexist

- **Programmers can choose** whether to execute a particular operation under strong or weak consistency

The hybrid alternative

- To allow **multiple consistency levels** to coexist

- **Programmers can choose** whether to execute a particular operation under strong or weak consistency

- E.g., if the execution of an operation may violate an application invariant, then the programmer should execute it under strong consistency

Partial order-restrictions (PoR)

Partial order-restrictions (PoR)

- The PoR model is a hybrid consistency model that allows programmers to classify operations as either **causal** or **strong**

Partial order-restrictions (PoR)

- The PoR model is a hybrid consistency model that allows programmers to classify operations as either **causal** or **strong**

- **Causal operations** satisfy causal consistency: clients observe operations in an order that respects potential causality

Partial order-restrictions (PoR)

- The PoR model is a hybrid consistency model that allows programmers to classify operations as either **causal** or **strong**

- **Causal operations** satisfy causal consistency: clients observe operations in an order that respects potential causality

- **Strong operations** give the programmer more control over causally independent operations

A banking application

A banking application

- **Deposits** to the same account can be executed under **weak consistency:** deposit is marked a **causal**

A banking application

- **Deposits** to the same account can be executed under **weak consistency:** deposit is marked a **causal**

A banking application

- **Deposits** to the same account can be executed under **weak consistency:** deposit is marked a **causal**

US

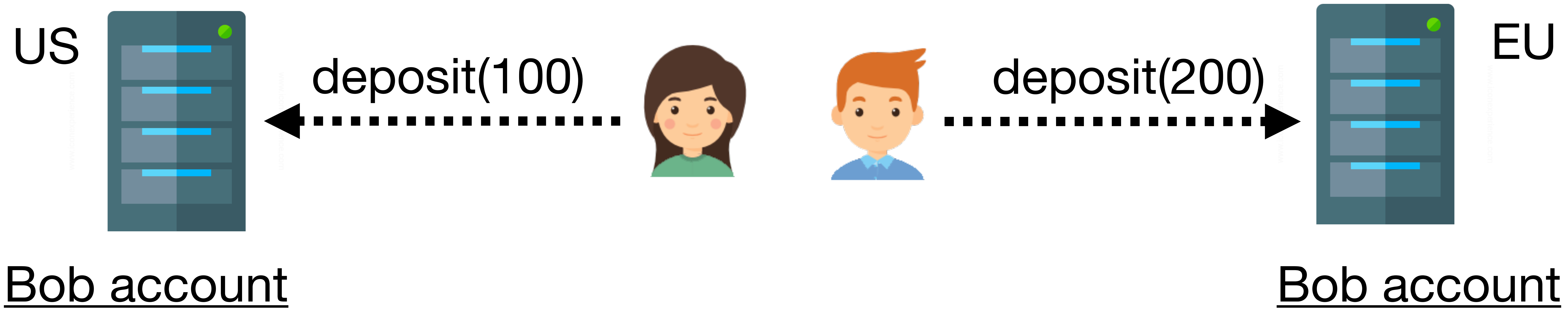
EU

Bob account

Bob account

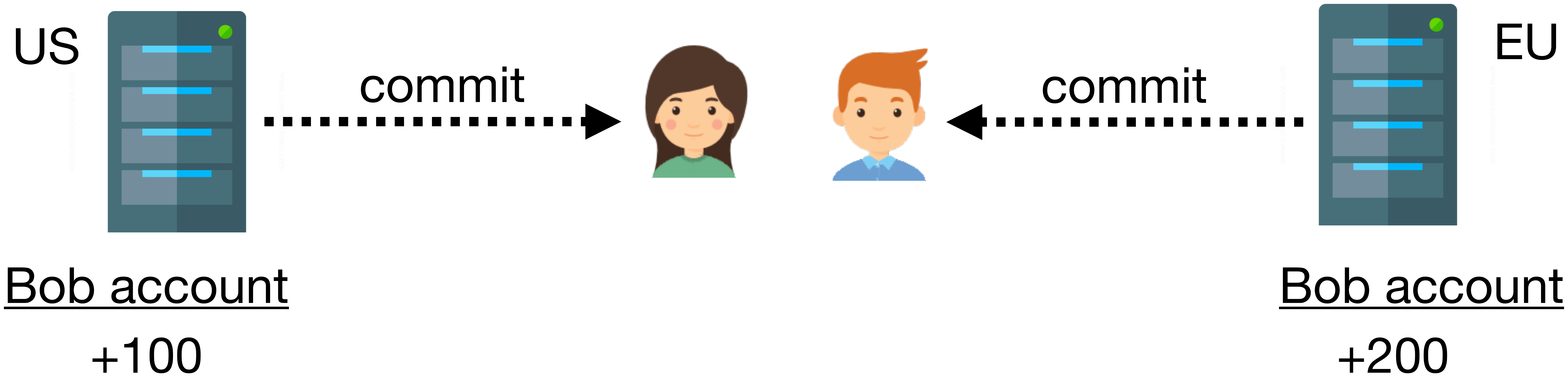
A banking application

- **Deposits** to the same account can be executed under **weak consistency:** deposit is marked a **causal**



A banking application

- **Deposits** to the same account can be executed under **weak consistency:** deposit is marked a **causal**



US

commit

commit

EU

Bob account

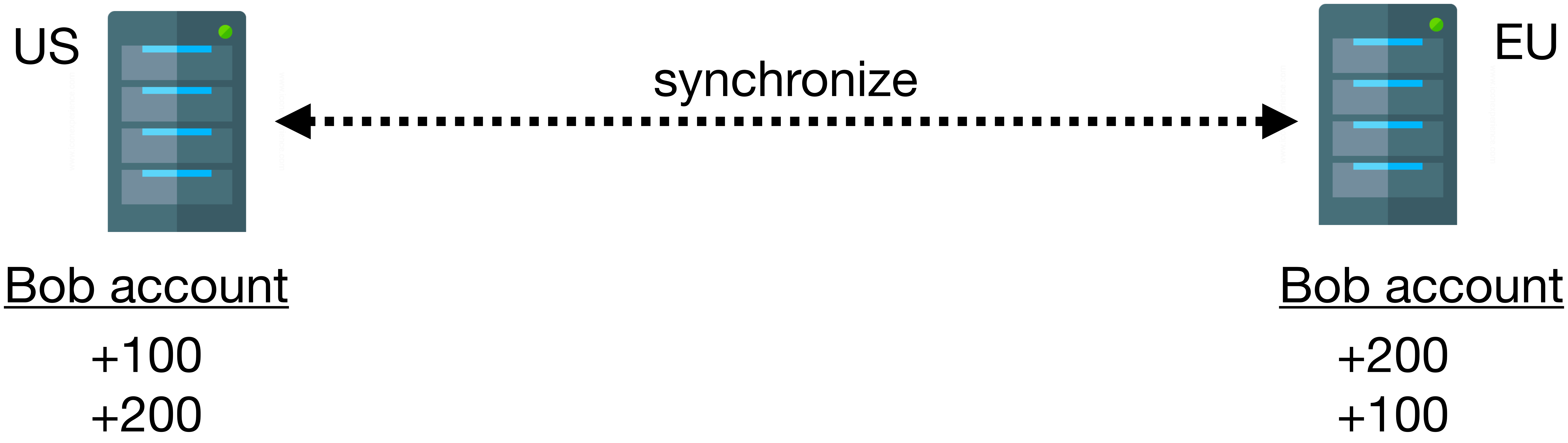
+100

Bob account

+200

A banking application

- **Deposits** to the same account can be executed under **weak consistency:** deposit is marked a **causal**



US

EU

synchronize

Bob account

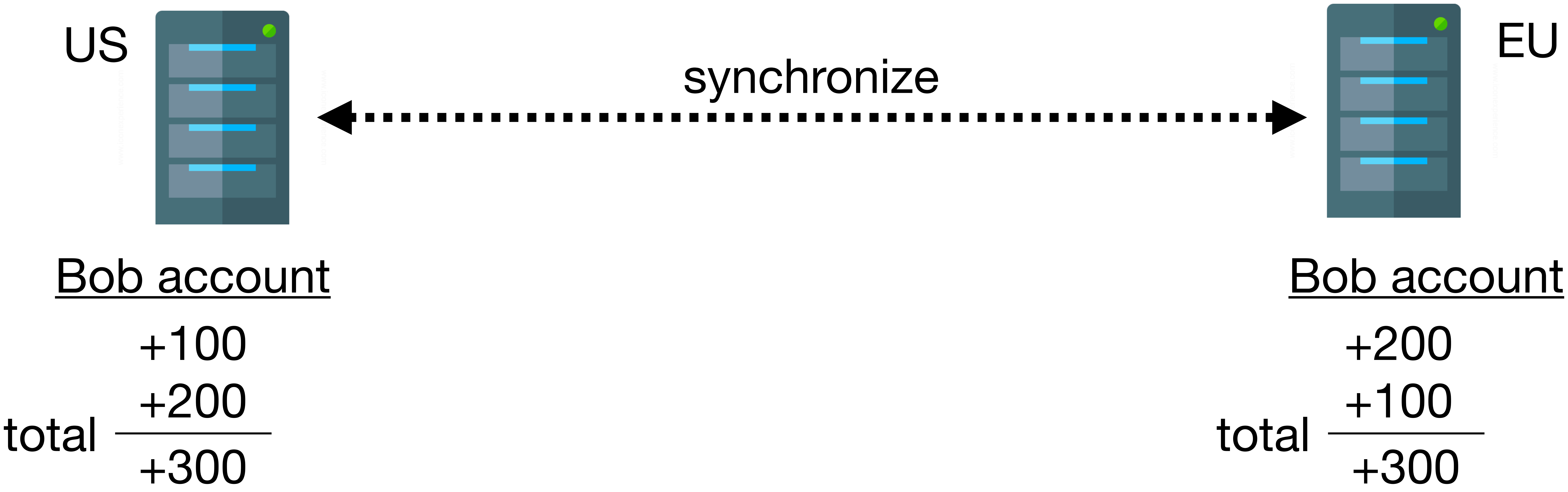
+100
+200

Bob account

+200
+100

A banking application

- **Deposits** to the same account can be executed under **weak consistency:** deposit is marked a **causal**



Bob account

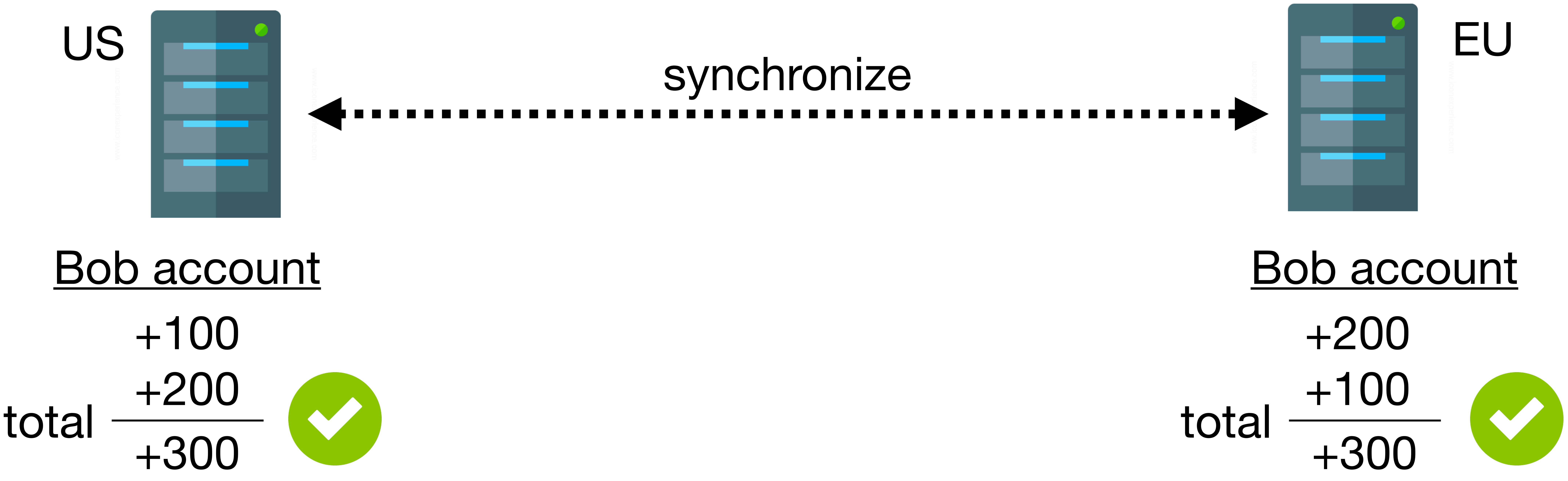
+100
+200
total +300

Bob account

+200
+100
total +300

A banking application

- **Deposits** to the same account can be executed under **weak consistency:** deposit is marked a **causal**



US

synchronize

EU

Bob account

+100
+200
total +300 ✓

Bob account

+200
+100
total +300 ✓

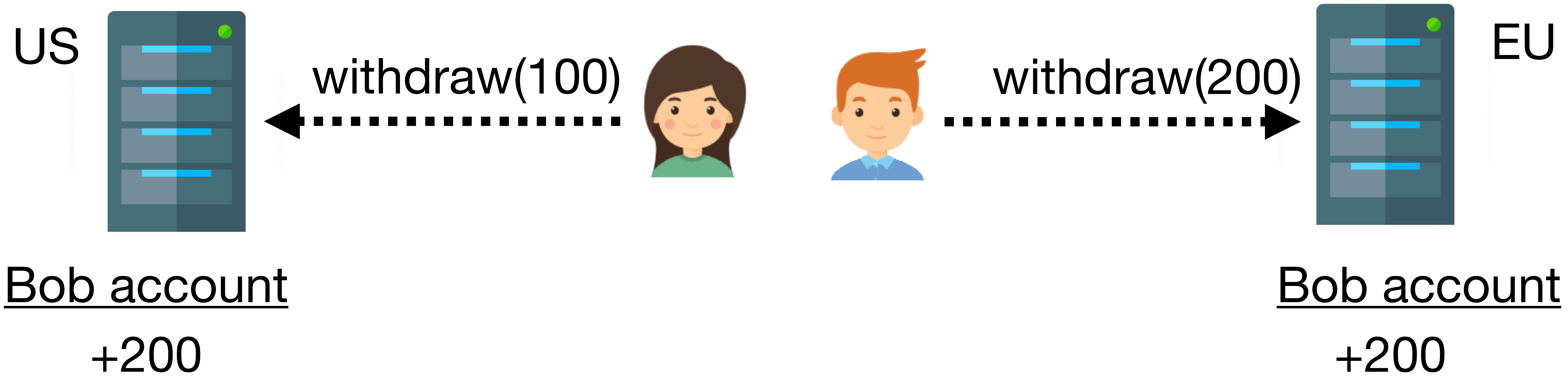
A banking application

- **Withdrawals** to the same account **cannot** be executed under weak consistency

US
Bob account
+200
EU
Bob account
+200

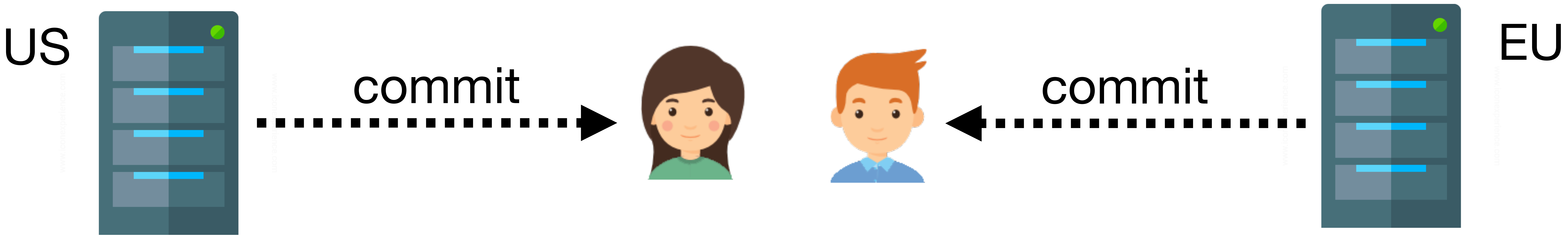
A banking application

- **Withdrawals** to the same account **cannot** be executed under weak consistency



A banking application

- **Withdrawals** to the same account **cannot** be executed under weak consistency



US

commit

commit

EU

Bob account

+200
-100

≥0

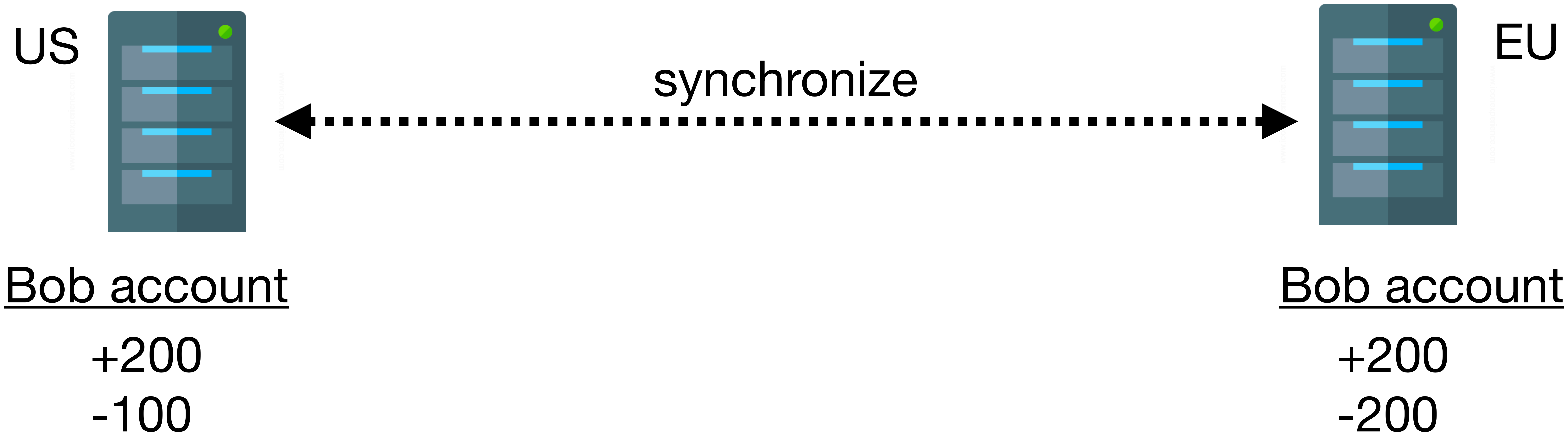
Bob account

+200
-200

≥0

A banking application

- **Withdrawals** to the same account **cannot** be executed under weak consistency



US

EU

synchronize

Bob account

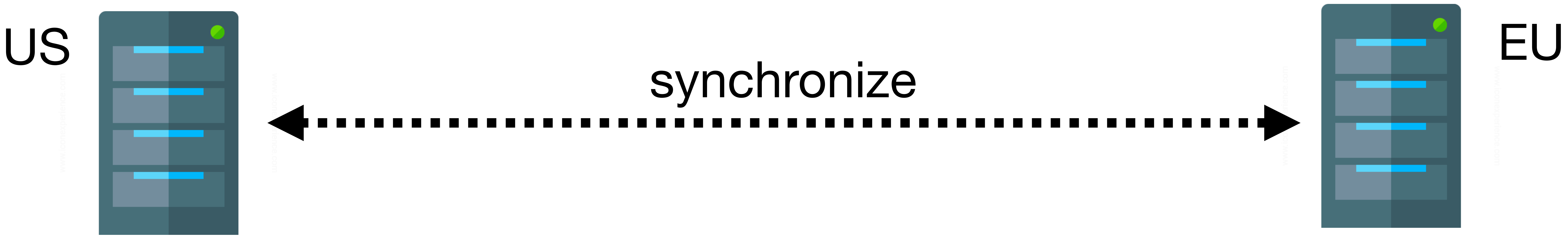
+200
-100

Bob account

+200
-200

A banking application

- **Withdrawals** to the same account **cannot** be executed under weak consistency



US

synchronize

EU

Bob account

+200
-100
-200

<0

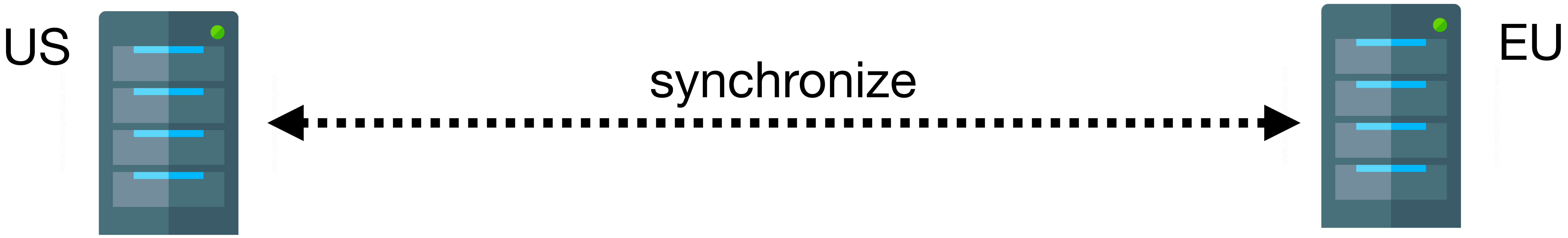
Bob account

+200
-200
-100

<0

A banking application

- **Withdrawals** to the same account **cannot** be executed under weak consistency



US

synchronize

EU

Bob account

+200
-100
-200

<0

too late, money’s gone!

Bob account

+200
-200
-100

<0

Partial order-restrictions (PoR)

Partial order-restrictions (PoR)

- The programmer provides a **symmetric conflict relation** $\bowtie$ on operations

Partial order-restrictions (PoR)

- The programmer provides a **symmetric conflict relation** $\bowtie$ on operations

- **Any operation involved** in the conflict relation is marked as **strong**

Partial order-restrictions (PoR)

- The programmer provides a **symmetric conflict relation** $\bowtie$ on operations

- **Any operation involved** in the conflict relation is marked as **strong**

- PoR guarantees that, **out of two conflicting** strong transactions, **one has to observe the other**

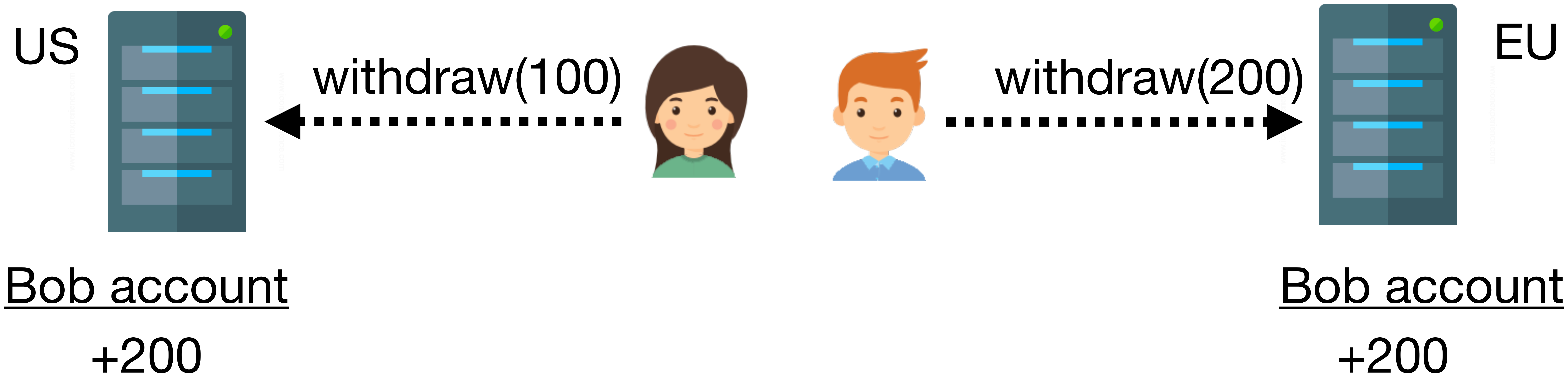
A banking application

- **Withdrawals** to the same account **are** executed under **strong consistency:** withdrawals to the same account conflict

US
EU
Bob account
+200
Bob account
+200

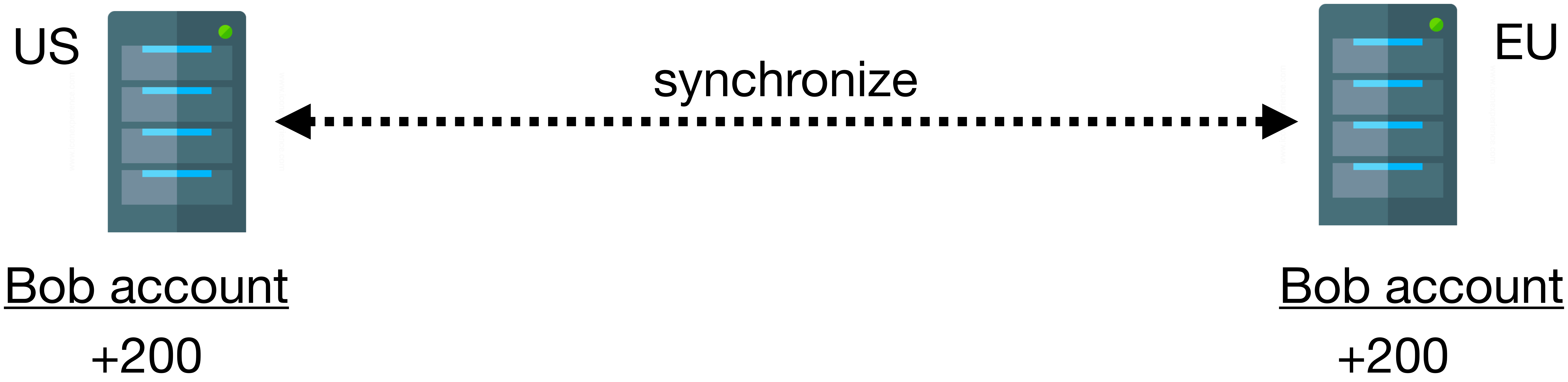
A banking application

- **Withdrawals** to the same account **are** executed under **strong consistency:** withdrawals to the same account conflict



A banking application

- **Withdrawals** to the same account **are** executed under **strong consistency:** withdrawals to the same account conflict



US

EU

synchronize

Bob account

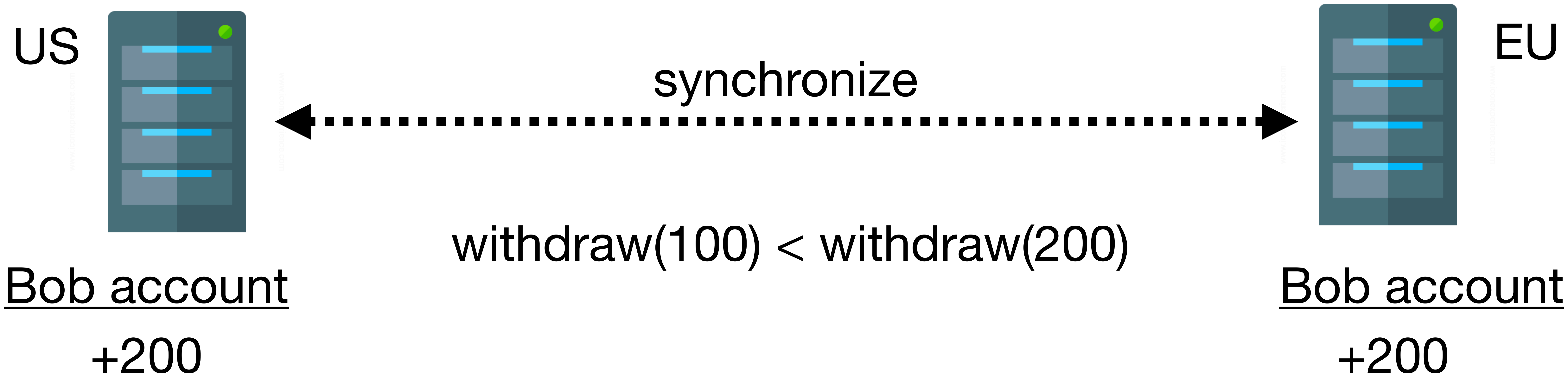
+200

Bob account

+200

A banking application

- **Withdrawals** to the same account **are** executed under **strong consistency:** withdrawals to the same account conflict



US

EU

synchronize

withdraw(100) < withdraw(200)

Bob account

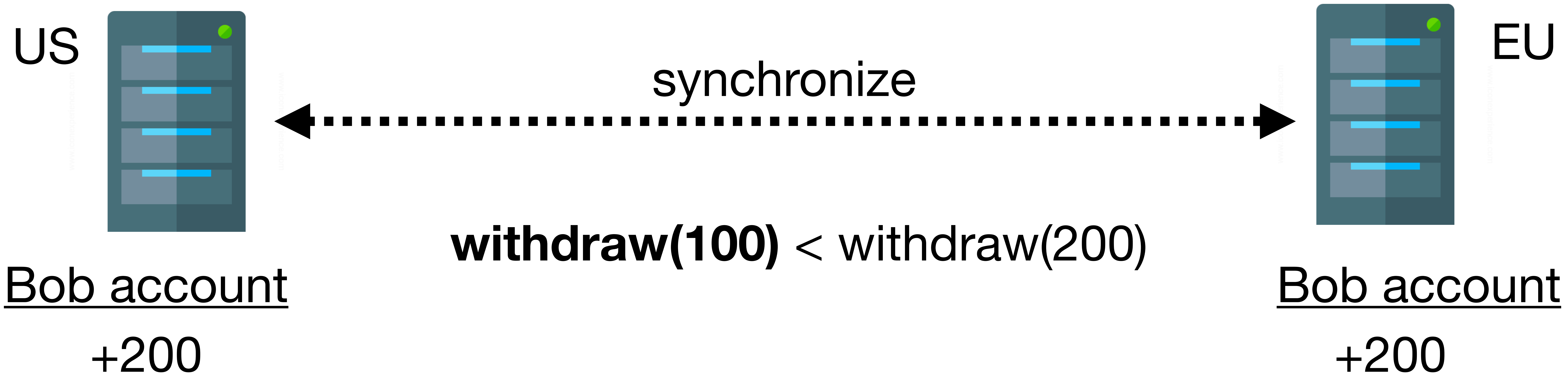
+200

Bob account

+200

A banking application

- **Withdrawals** to the same account **are** executed under **strong consistency:** withdrawals to the same account conflict



US

EU

synchronize

withdraw(100) < withdraw(200)

Bob account

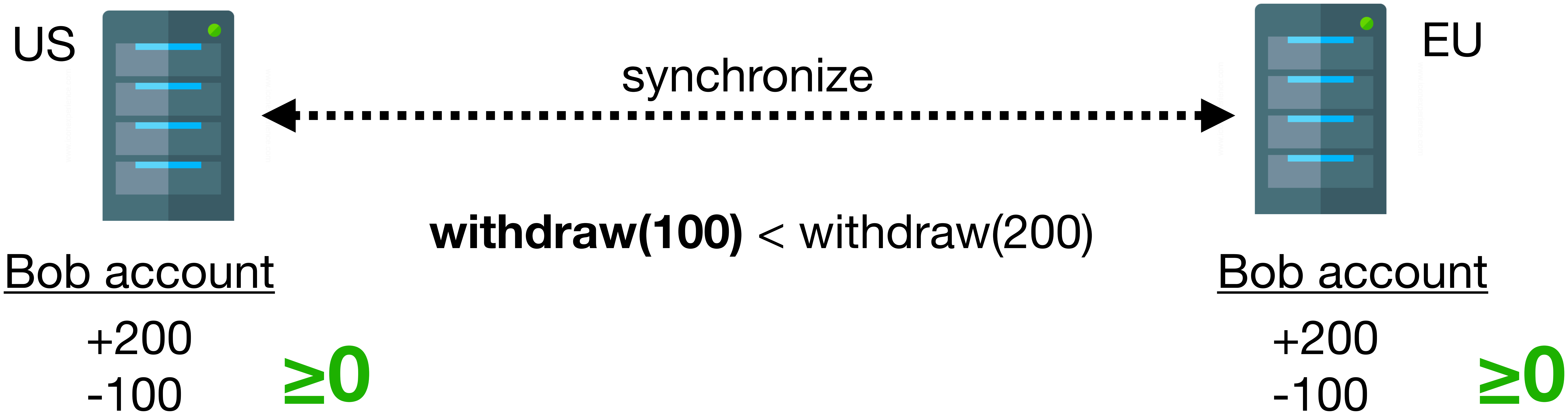
+200

Bob account

+200

A banking application

- **Withdrawals** to the same account **are** executed under **strong consistency:** withdrawals to the same account conflict



US

EU

synchronize

withdraw(100) < withdraw(200)

Bob account

+200
-100

≥0

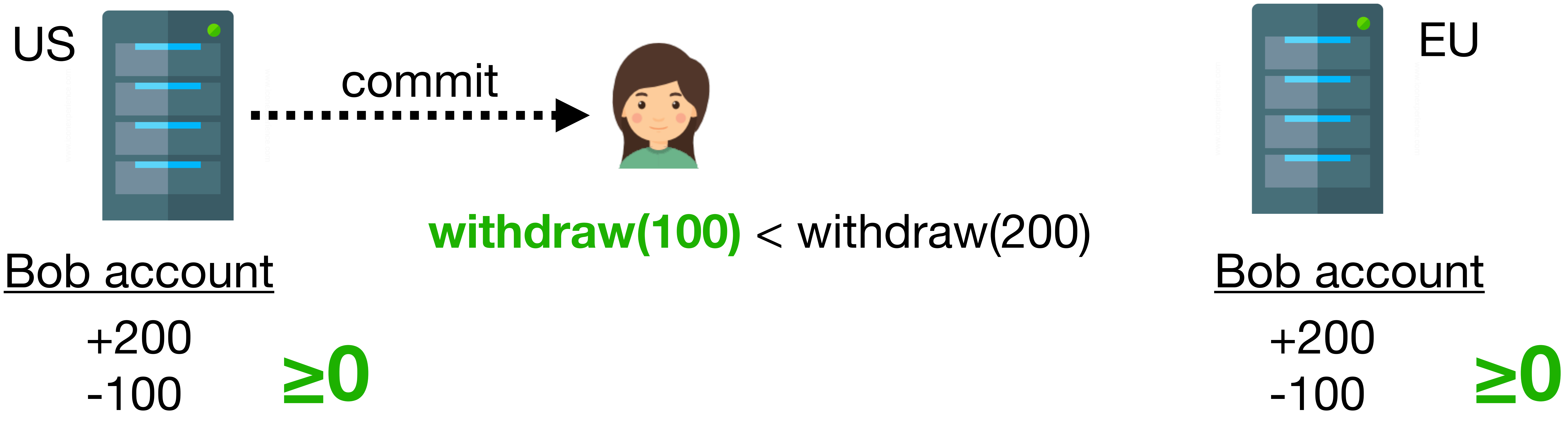
Bob account

+200
-100

≥0

A banking application

- **Withdrawals** to the same account **are** executed under **strong consistency:** withdrawals to the same account conflict



US

commit

EU

withdraw(100) < withdraw(200)

Bob account

+200
-100

≥0

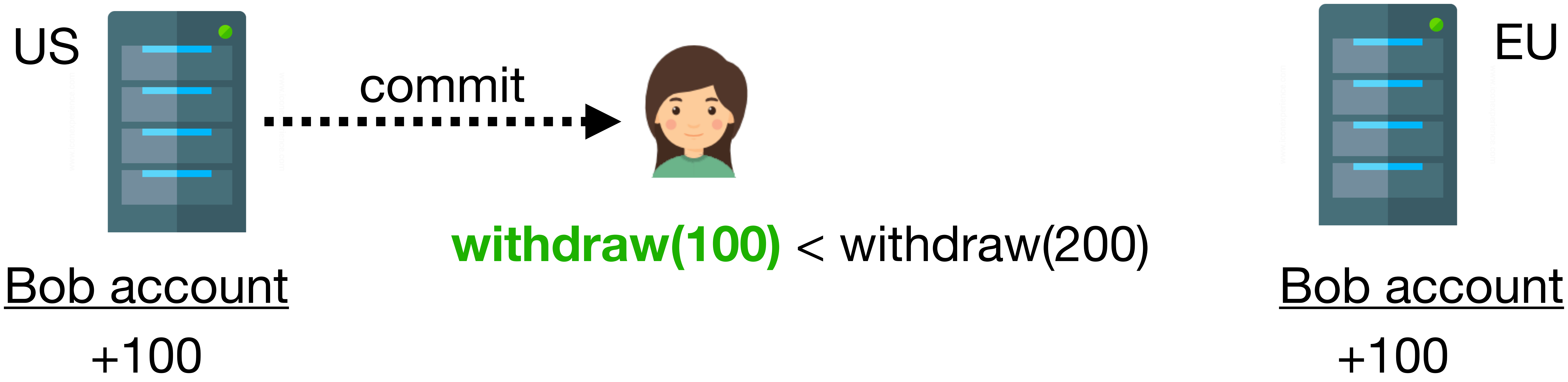
Bob account

+200
-100

≥0

A banking application

- **Withdrawals** to the same account **are** executed under **strong consistency:** withdrawals to the same account conflict



US

commit

EU

withdraw(100) < withdraw(200)

Bob account

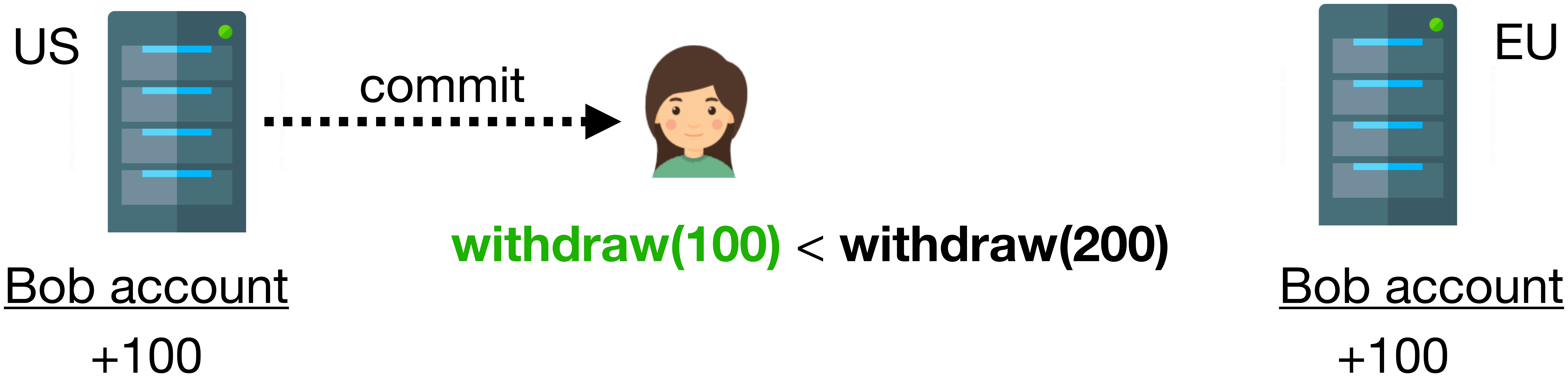
+100

Bob account

+100

A banking application

- **Withdrawals** to the same account **are** executed under **strong consistency:** withdrawals to the same account conflict



US

commit

EU

withdraw(100) < **withdraw(200)**

Bob account

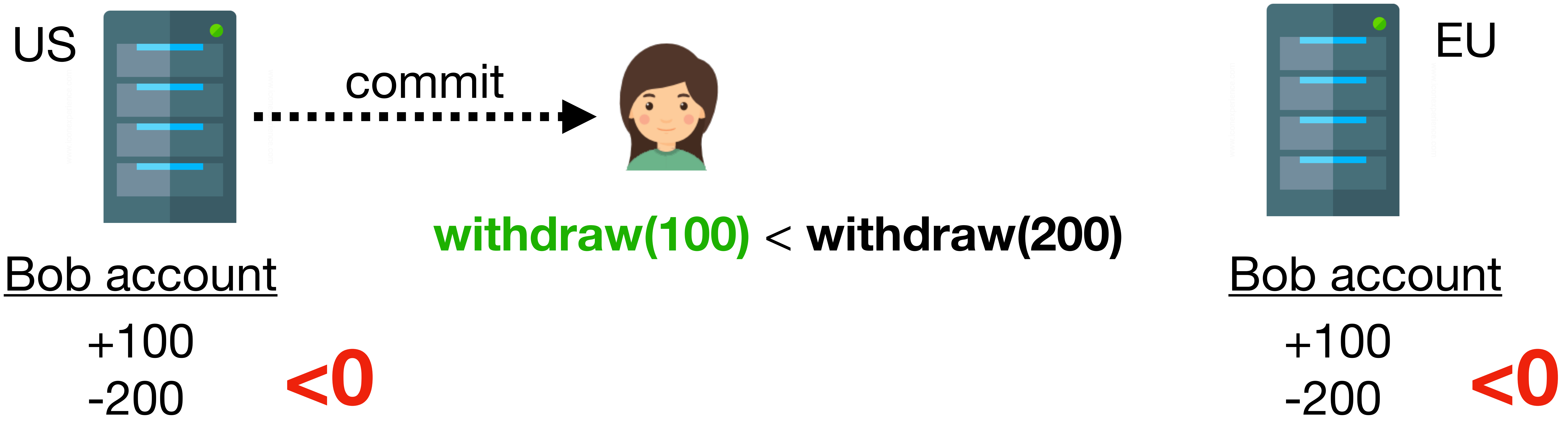
+100

Bob account

+100

A banking application

- **Withdrawals** to the same account **are** executed under **strong consistency:** withdrawals to the same account conflict



US

commit

EU

withdraw(100) < **withdraw(200)**

Bob account

+100
-200

<0

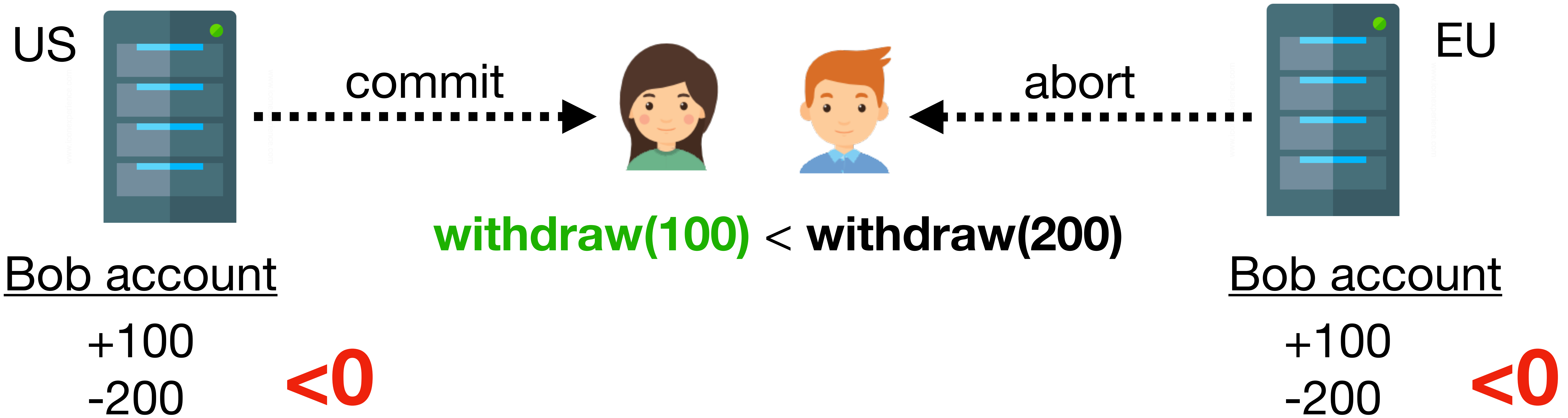
Bob account

+100
-200

<0

A banking application

- **Withdrawals** to the same account **are** executed under **strong consistency:** withdrawals to the same account conflict



US

commit

abort

EU

withdraw(100) < **withdraw(200)**

Bob account

+100
-200

<0

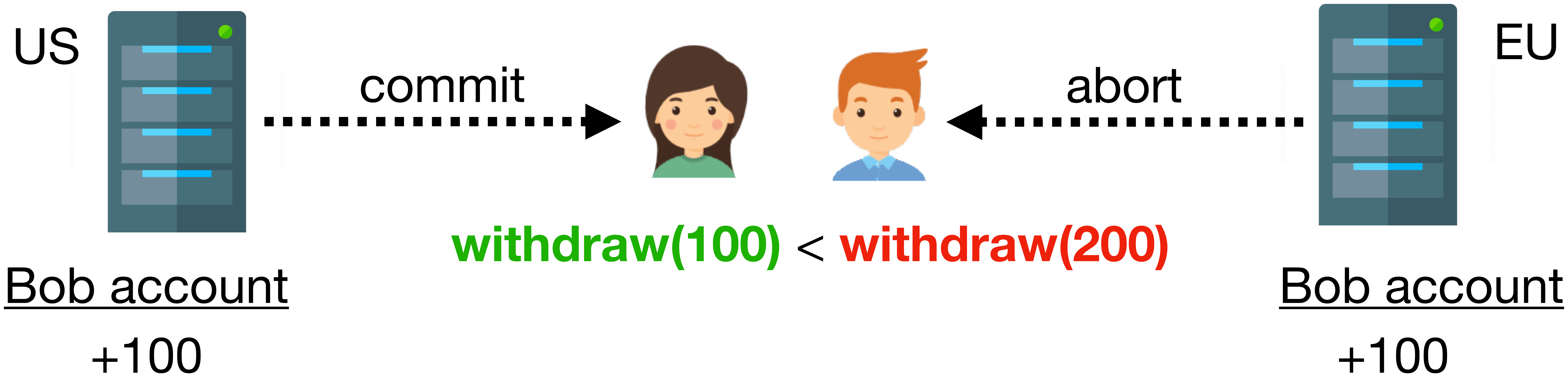
Bob account

+100
-200

<0

A banking application

- **Withdrawals** to the same account **are** executed under **strong consistency:** withdrawals to the same account conflict



US

commit

abort

EU

withdraw(100) < withdraw(200)

Bob account

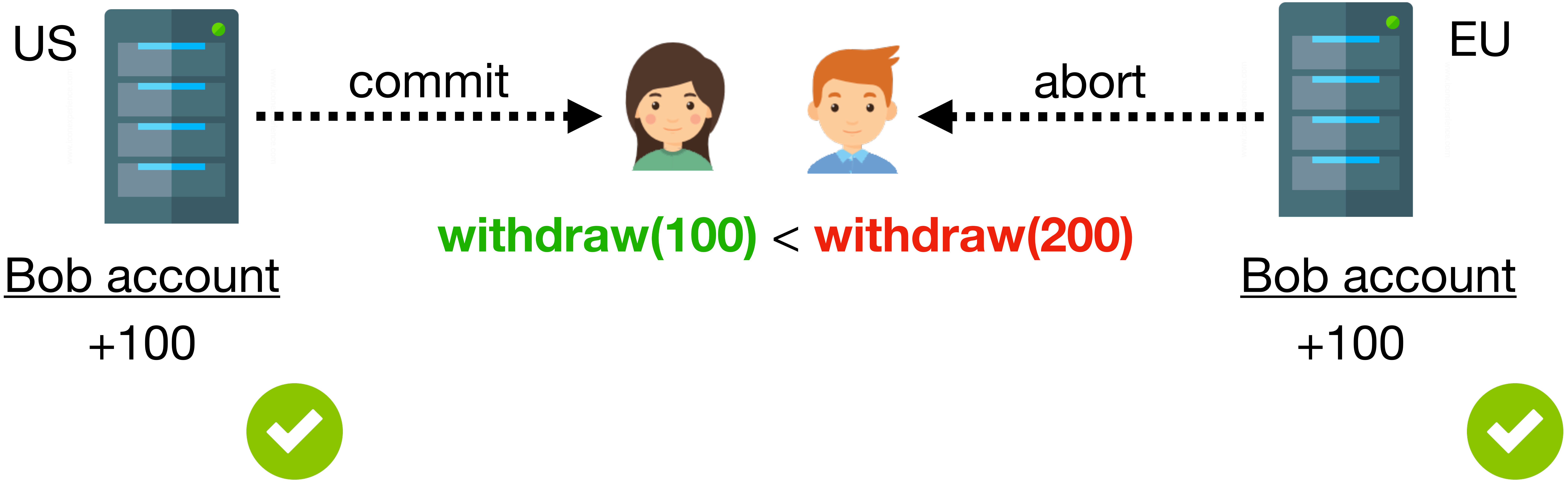
+100

Bob account

+100

A banking application

- **Withdrawals** to the same account **are** executed under **strong consistency:** withdrawals to the same account conflict



US

commit

abort

EU

withdraw(100) < **withdraw(200)**

Bob account

+100

Bob account

+100

UniStore

UniStore

- The first fault-tolerant and scalable data store that combines causal and strong consistency

UniStore

- The first fault-tolerant and scalable data store that combines causal and strong consistency

- Implements a transactional variant of PoR consistency

UniStore

- The first fault-tolerant and scalable data store that combines causal and strong consistency

- Implements a transactional variant of PoR consistency

- It guarantees **transactional causal consistency by default** and allows the programmer to **additionally specify which pairs of transactions *conflict***, i.e., have to synchronize

UniStore: causal baseline

UniStore: causal baseline

- **UniStore builds on Cure** [ICDCS’ 16], a scalable implementation of transactional causal consistency

UniStore: causal baseline

- **UniStore builds on Cure** [ICDCS’ 16], a scalable implementation of transactional causal consistency

- A causal transaction first **executes at a single data center** on a causally consistent snapshot

UniStore: causal baseline

- **UniStore builds on Cure** [ICDCS' 16], a scalable implementation of transactional causal consistency

- A causal transaction first **executes at a single data center** on a causally consistent snapshot

- After this it immediately commits, and its updates are **replicated** to all other data centers **in the background**

UniStore: strong txs

UniStore: strong txs

- UniStore uses **optimistic concurrency control** to execute strong transactions

UniStore: strong txs

- UniStore uses **optimistic concurrency control** to execute strong transactions

- **first executed speculatively** and the results are **then *certified*** to determine whether the transaction can commit

UniStore: strong txs

- UniStore uses **optimistic concurrency control** to execute strong transactions

- **first executed speculatively** and the results are **then *certified*** to determine whether the transaction can commit

- Certification **requires synchronization between the replicas** of partitions it accessed, located in different data centers

UniStore: strong txs

- UniStore uses **optimistic concurrency control** to execute strong transactions

- **first executed speculatively** and the results are **then *certified*** to determine whether the transaction can commit

- Certification **requires synchronization between the replicas** of partitions it accessed, located in different data centers

- Uses an existing fault-tolerant protocol that combines **two-phase commit and Paxos** while minimizing commit latency

UniStore: key challenge

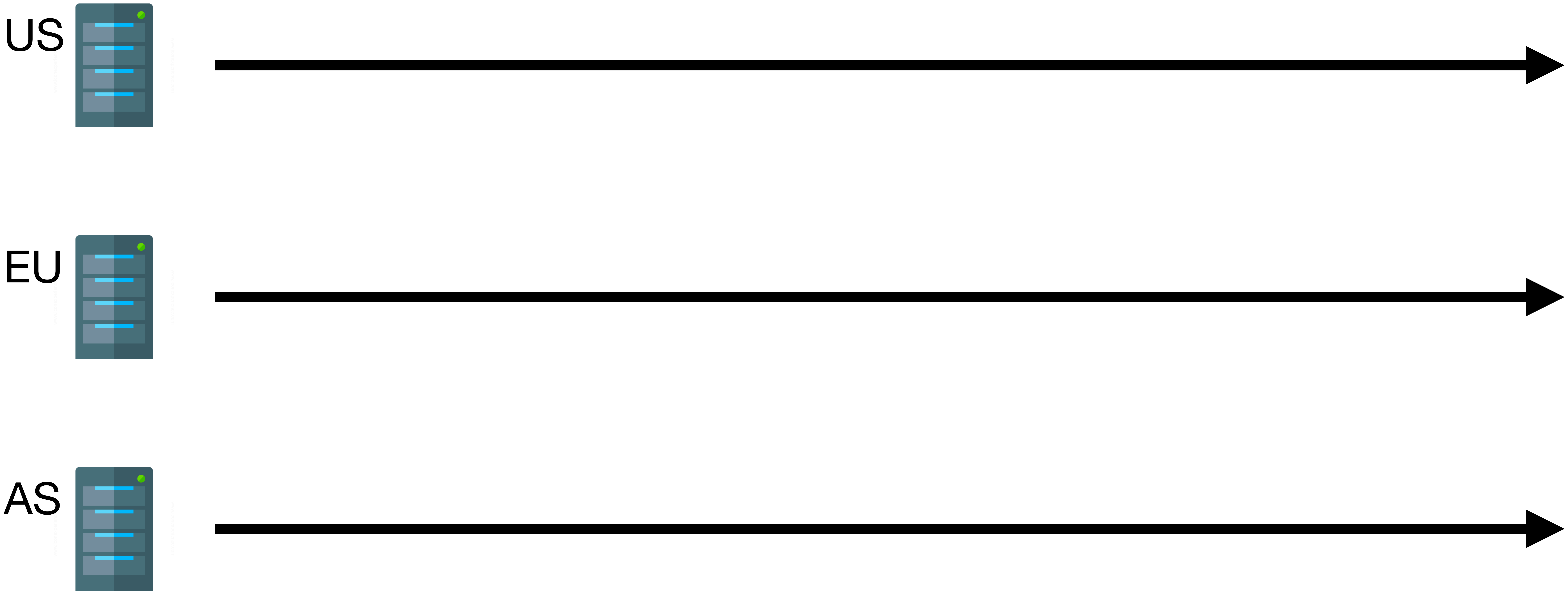
UniStore: key challenge

- Maintain **liveness despite data center failures**

UniStore: key challenge

- Maintain **liveness despite data center failures**
- Simply **adding a Paxos-based commit protocol** for strong transactions to an existing causally consistent protocol **does not yield a fault-tolerant data store**

UniStore: key challenge

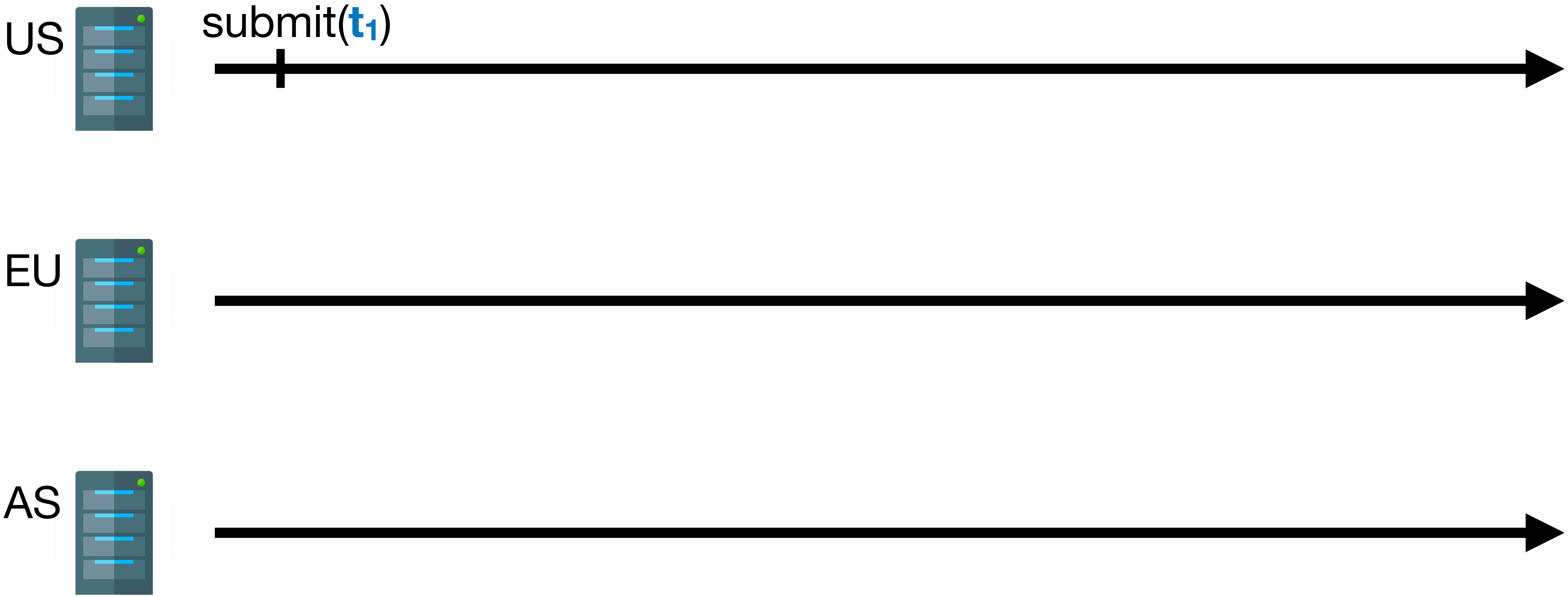


US

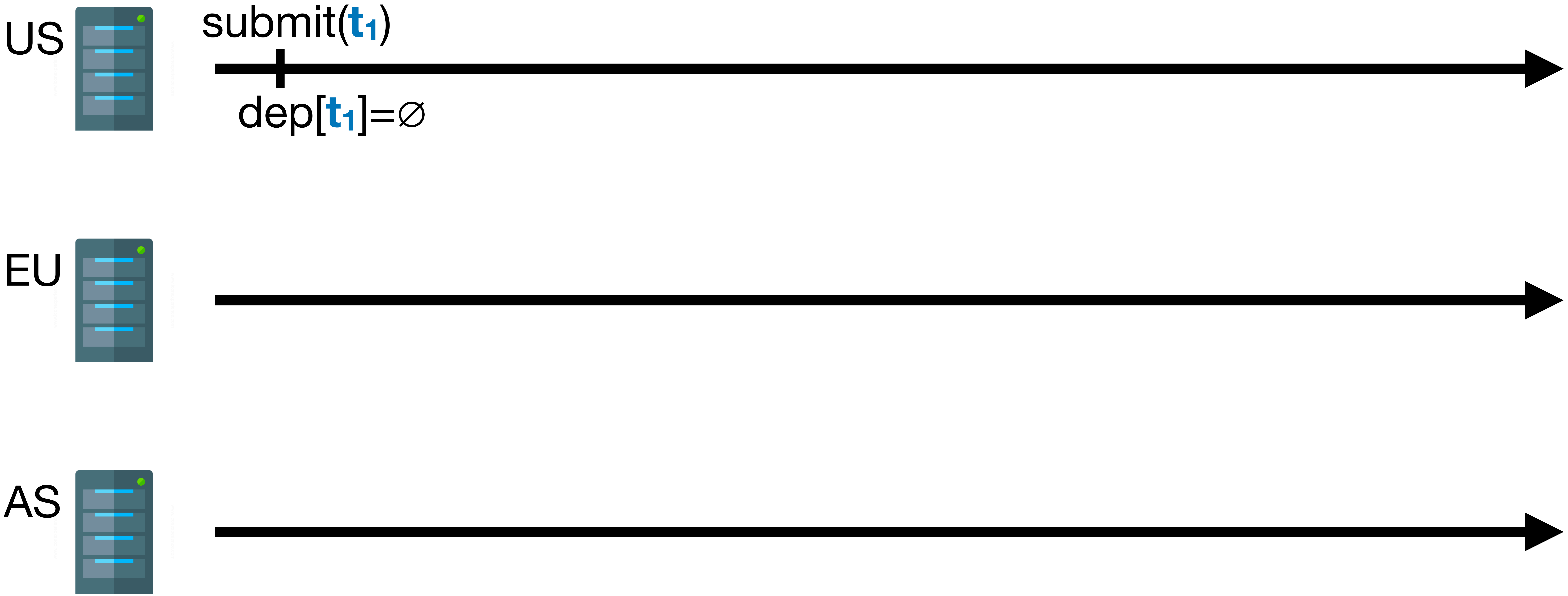
EU

AS

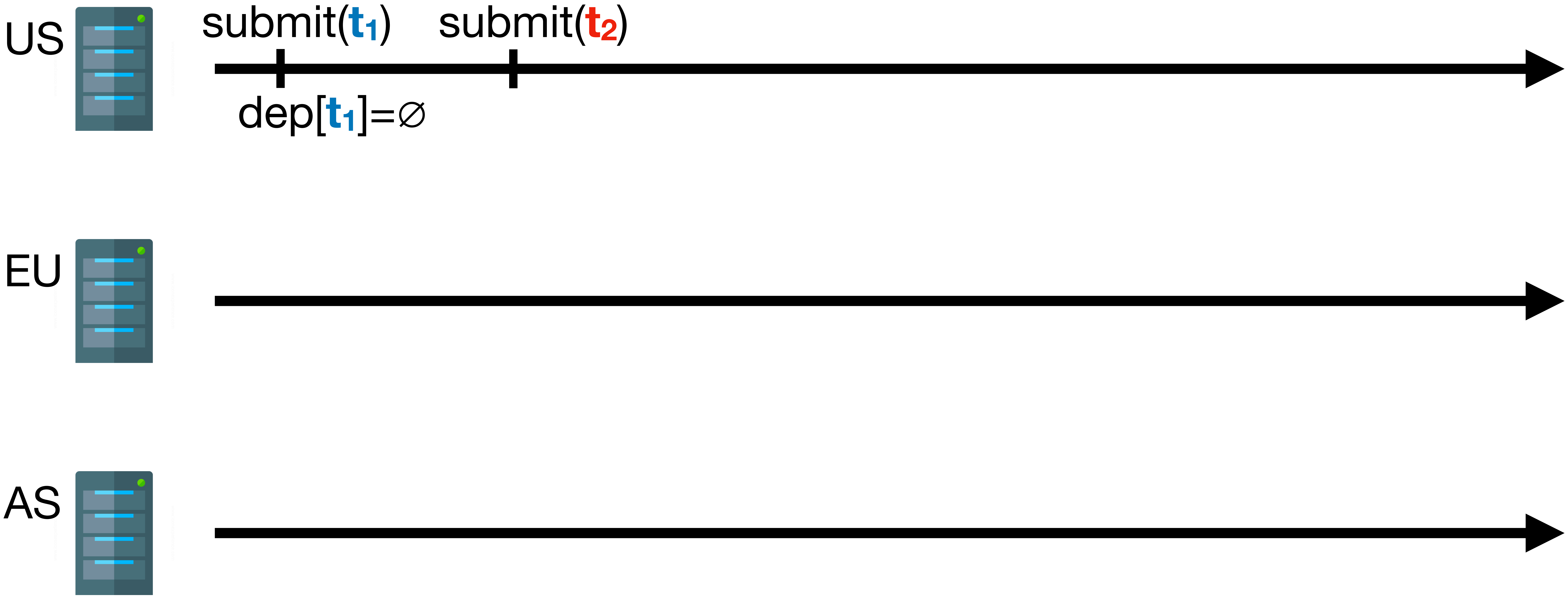
UniStore: key challenge



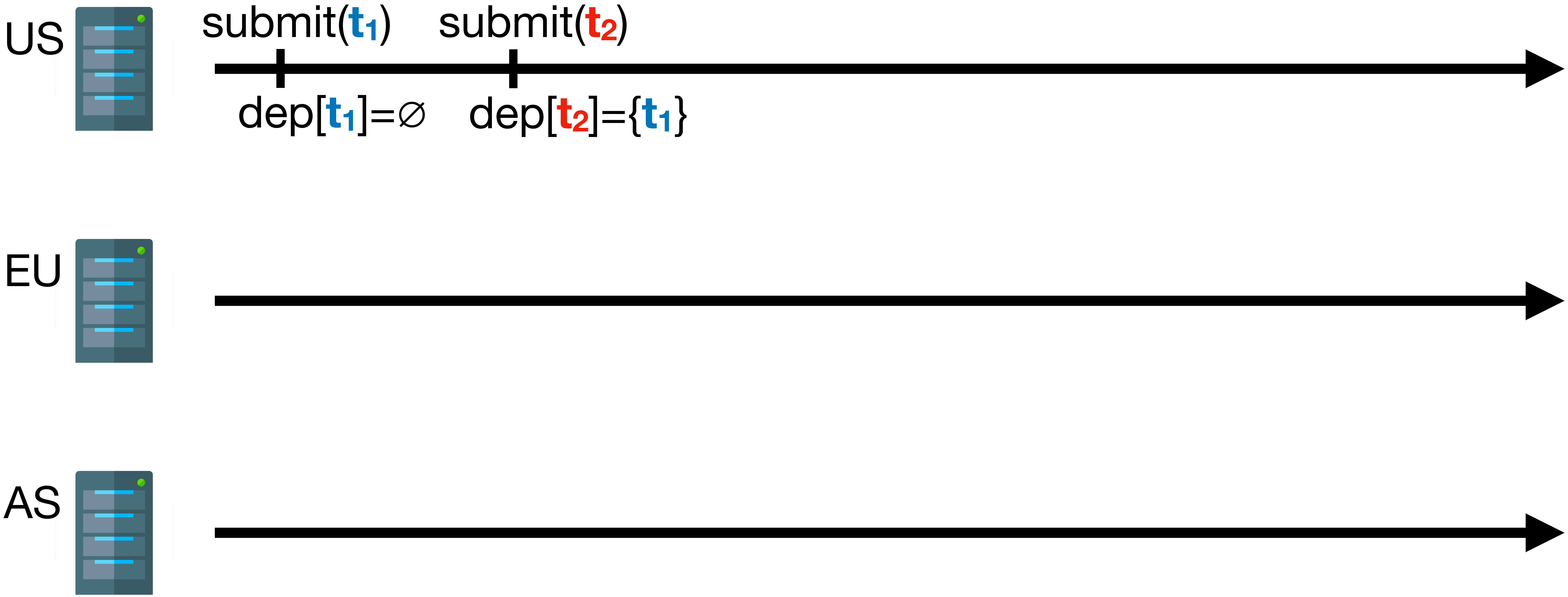
UniStore: key challenge



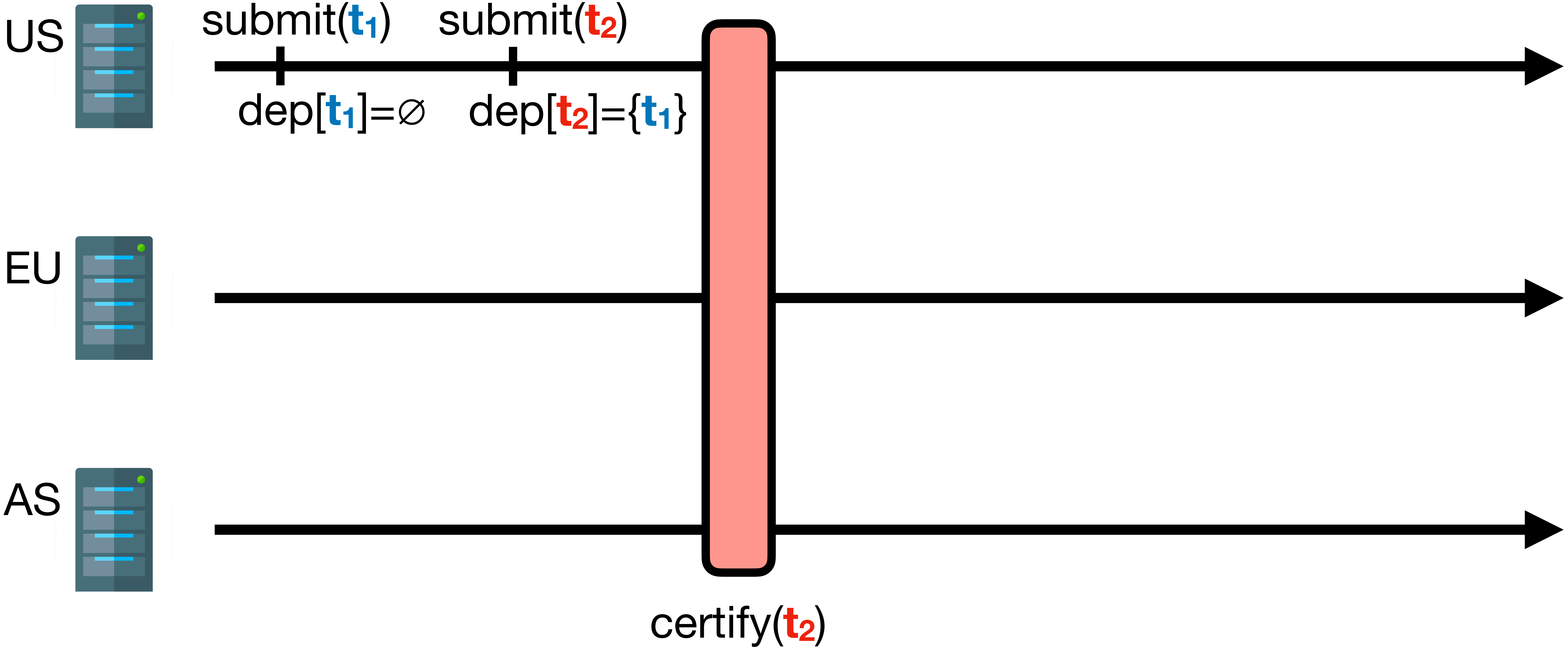
UniStore: key challenge



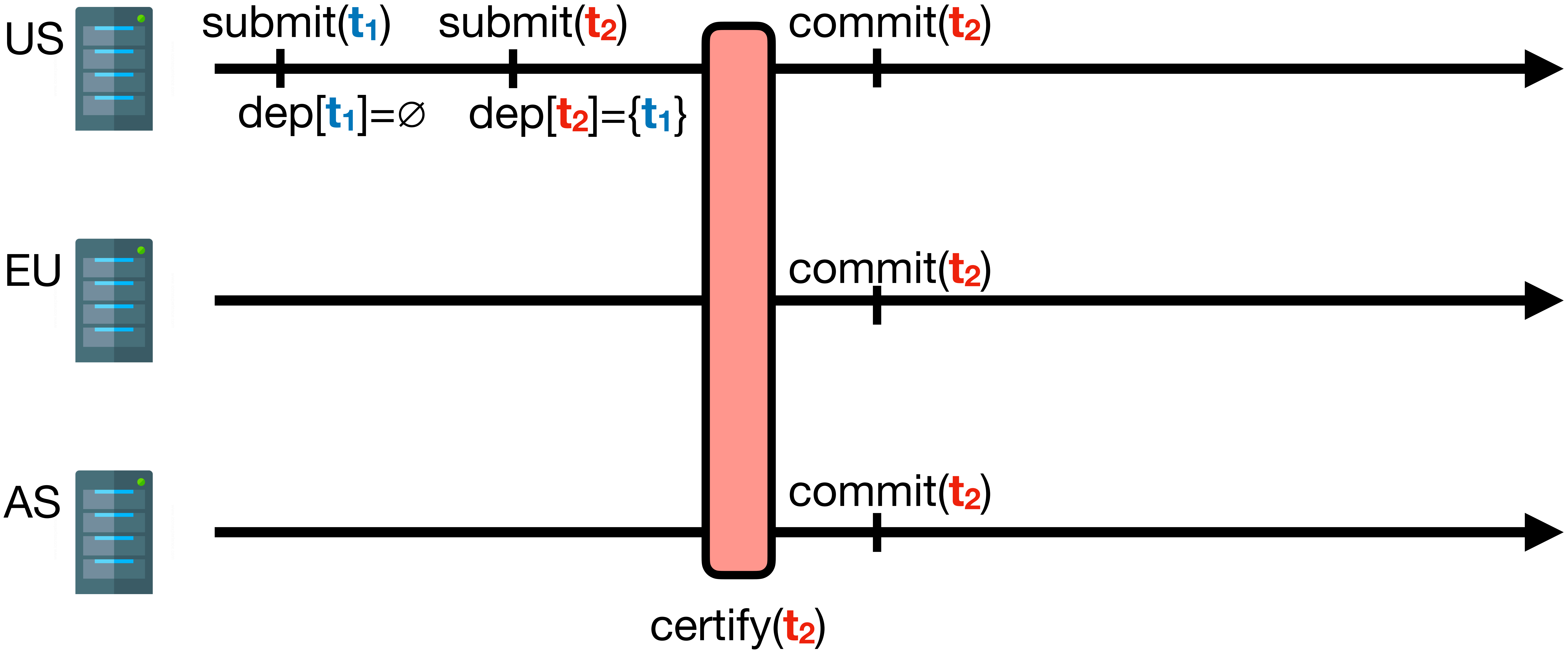
UniStore: key challenge



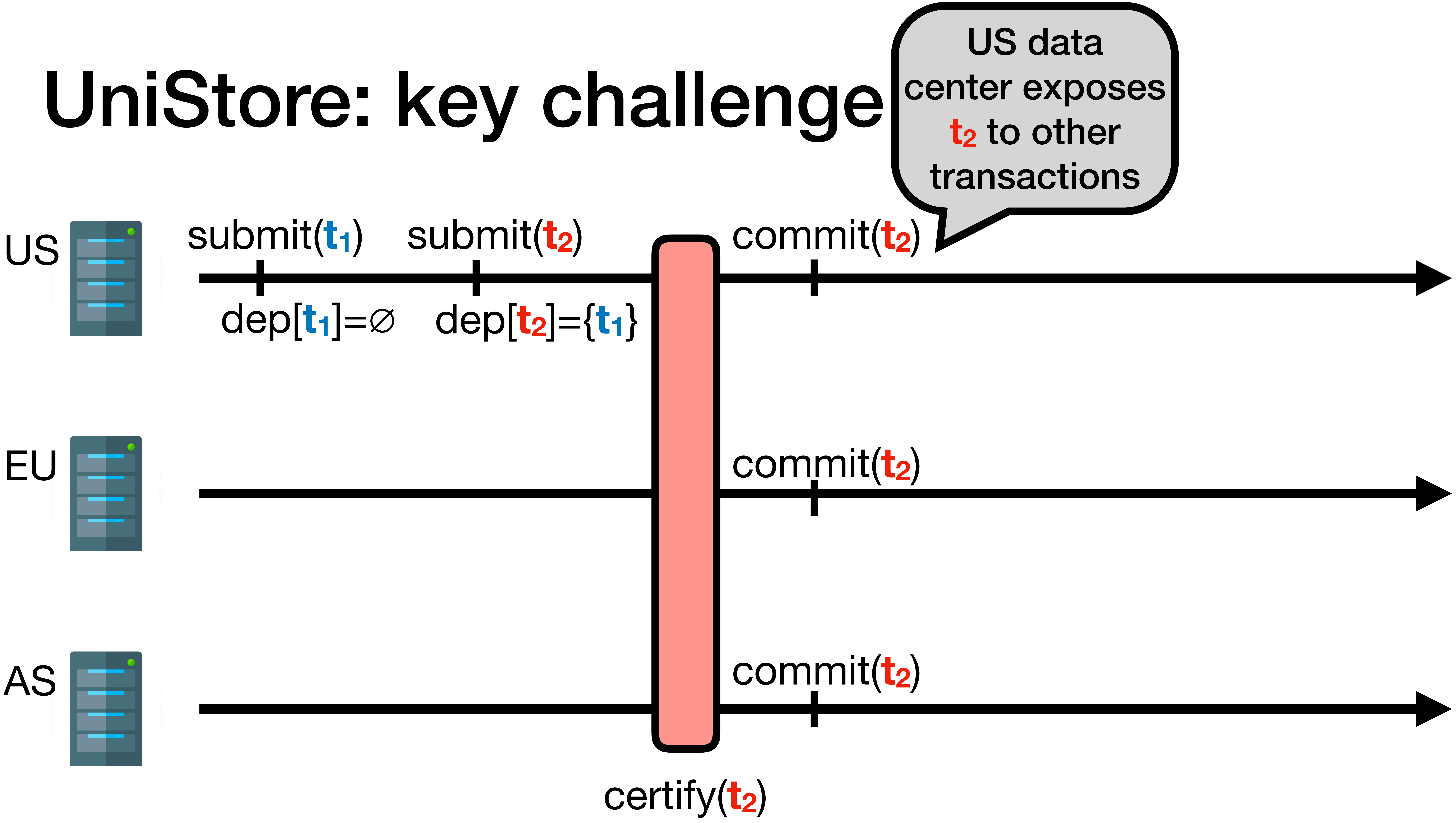
UniStore: key challenge



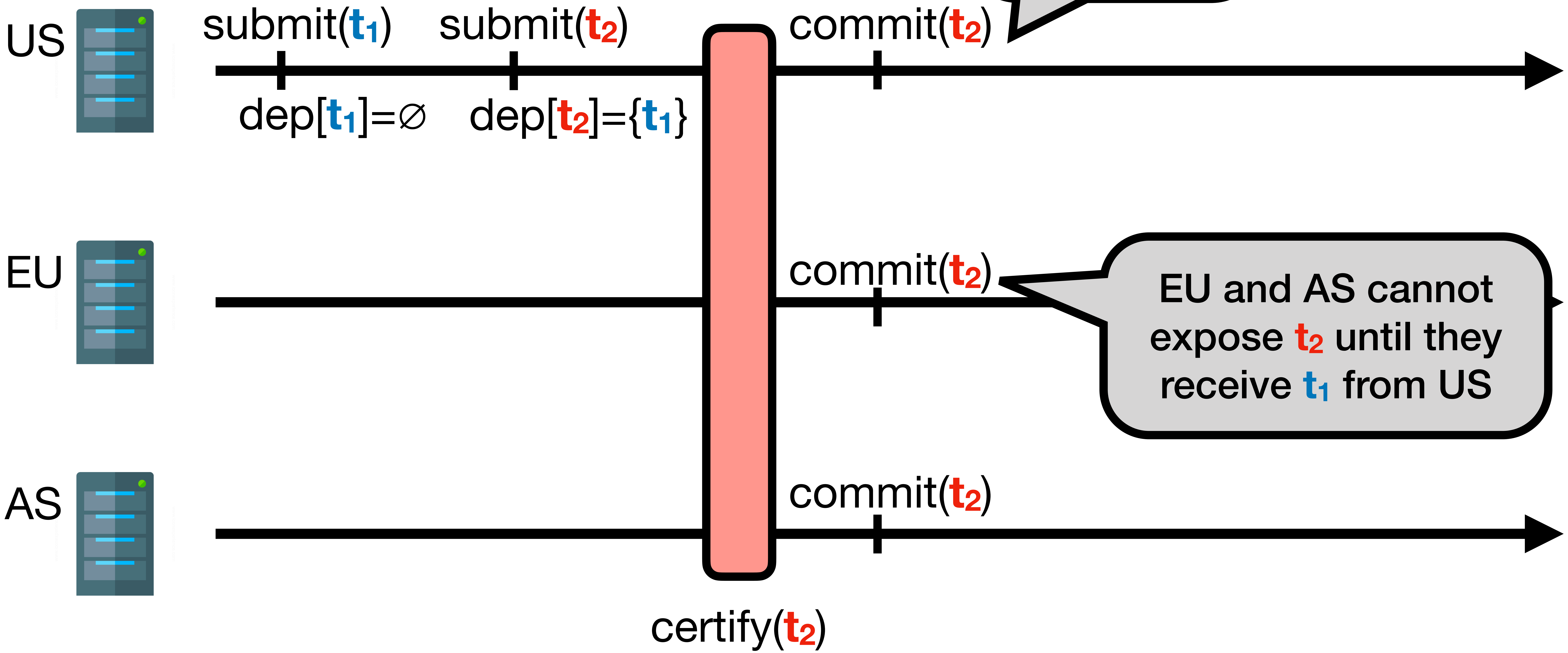
UniStore: key challenge



UniStore: key challenge



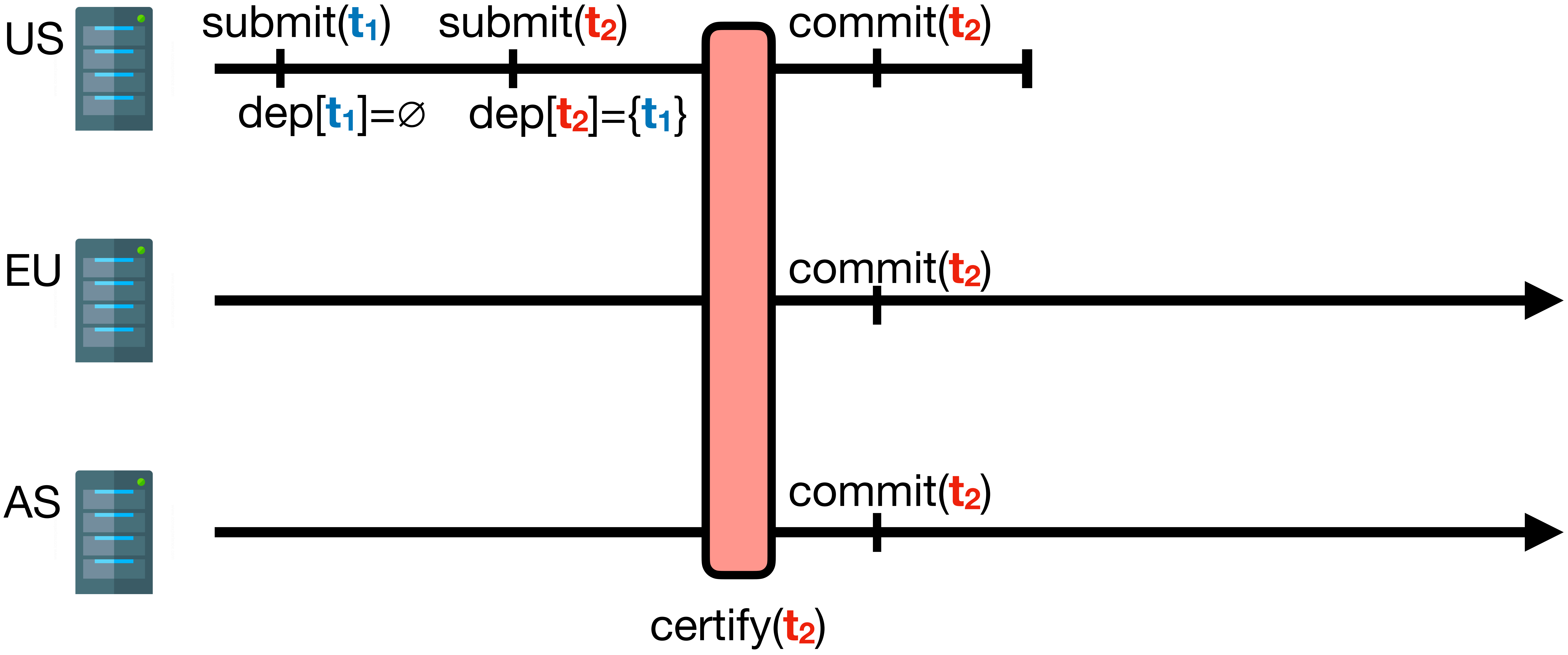
UniStore: key challenge



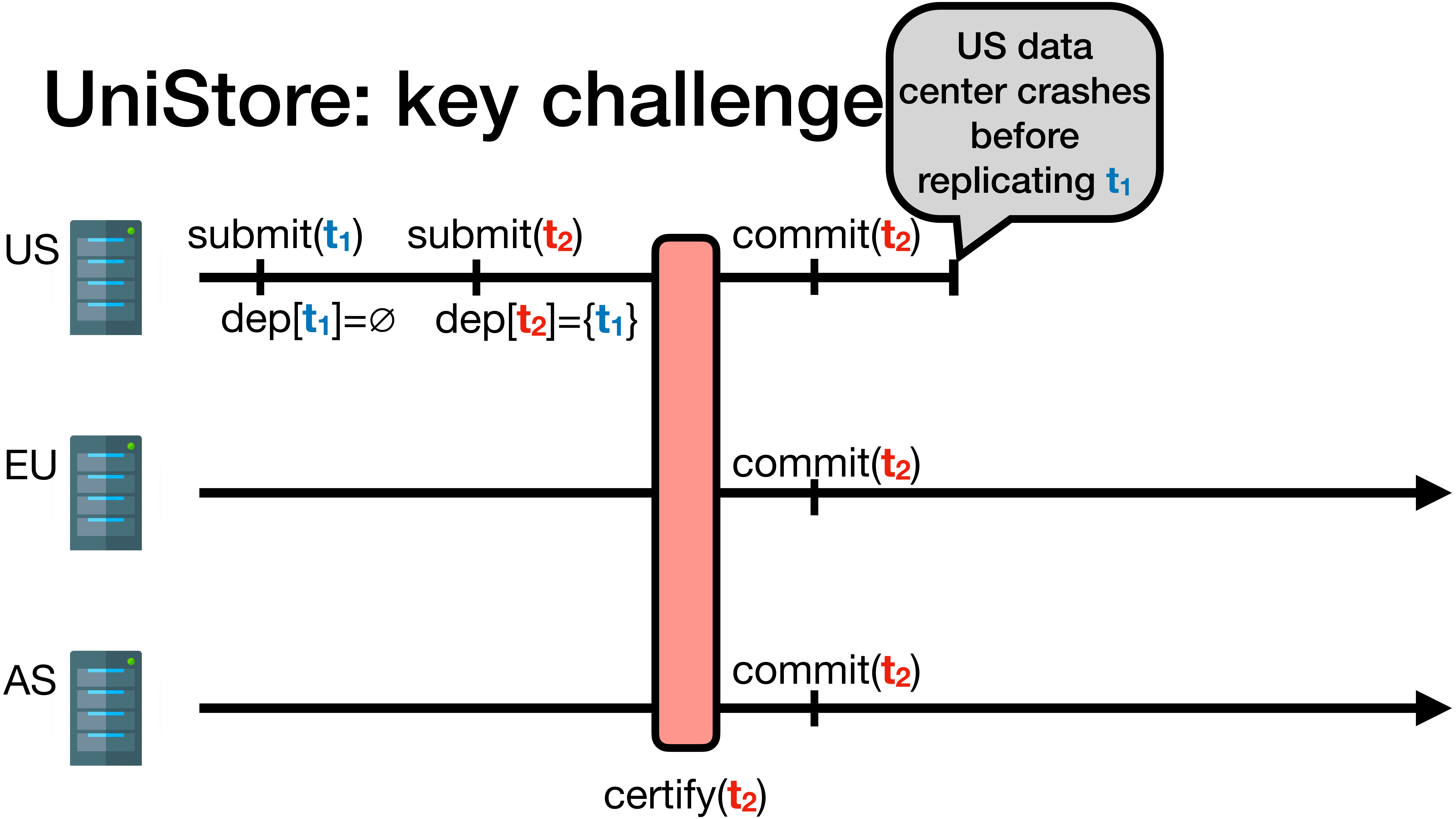
US data center exposes t_2 to other transactions

EU and AS cannot expose t_2 until they receive t_1 from US

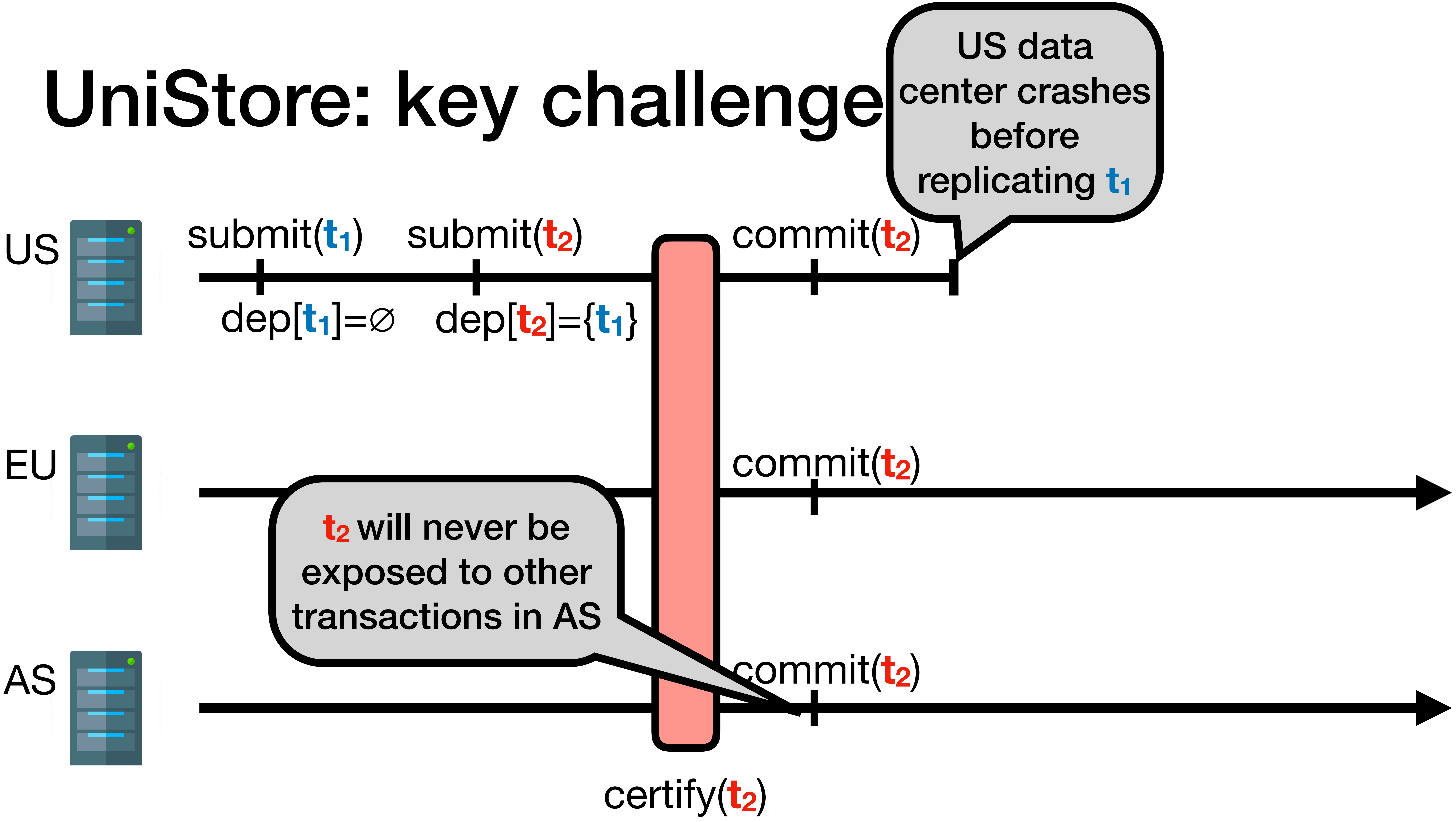
UniStore: key challenge



UniStore: key challenge



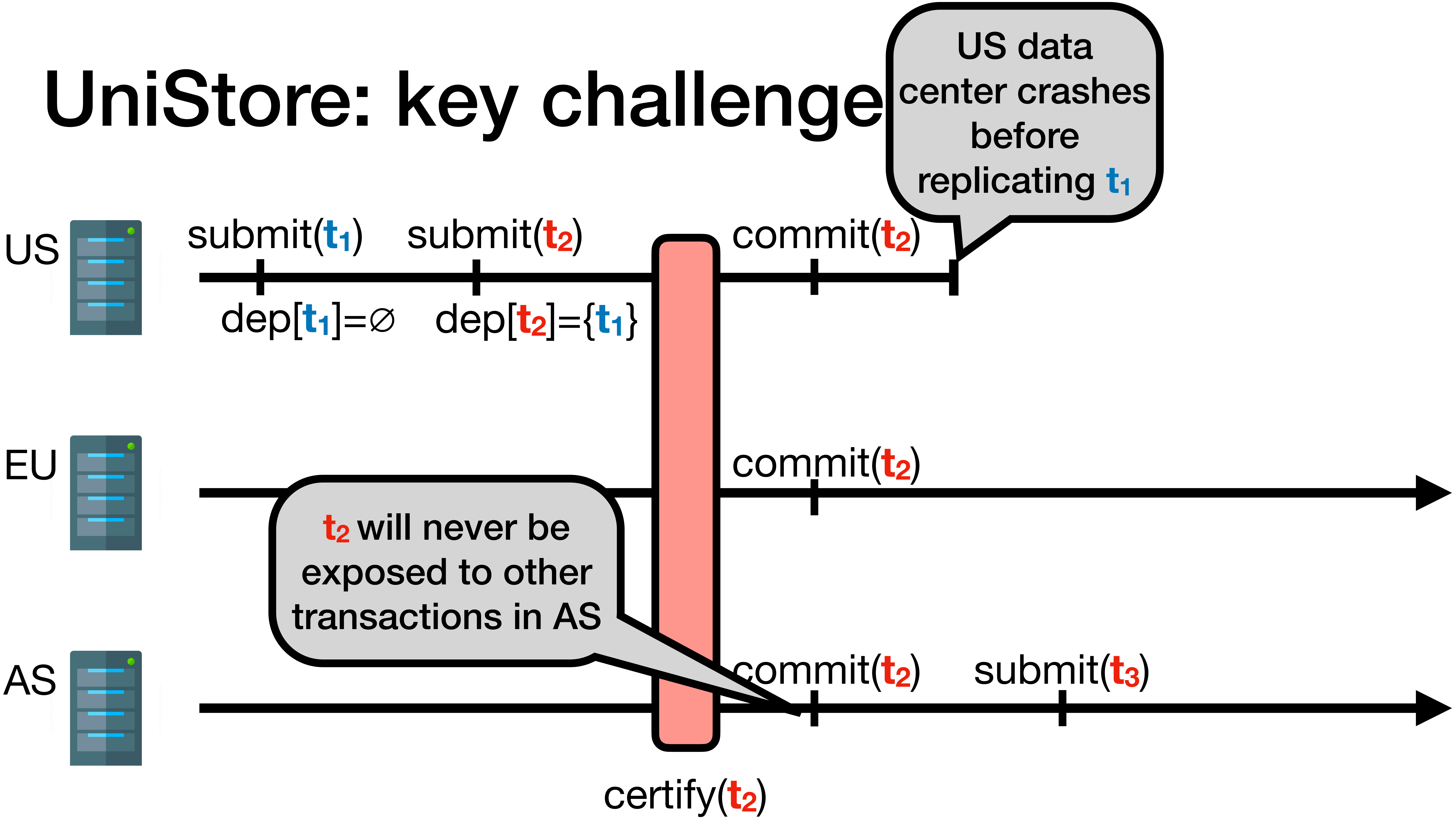
UniStore: key challenge



US data center crashes before replicating t_1

t_2 will never be exposed to other transactions in AS

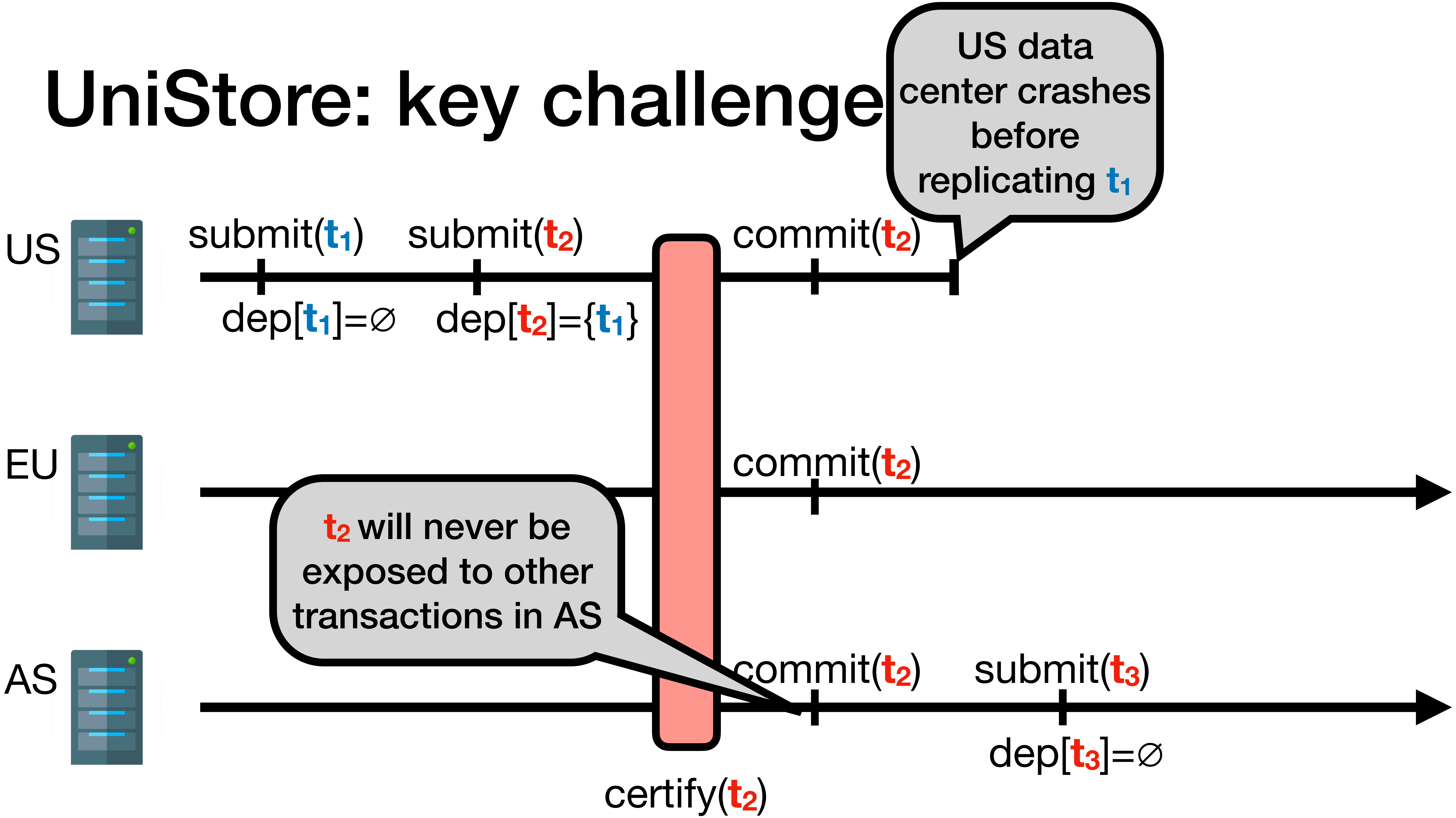
UniStore: key challenge



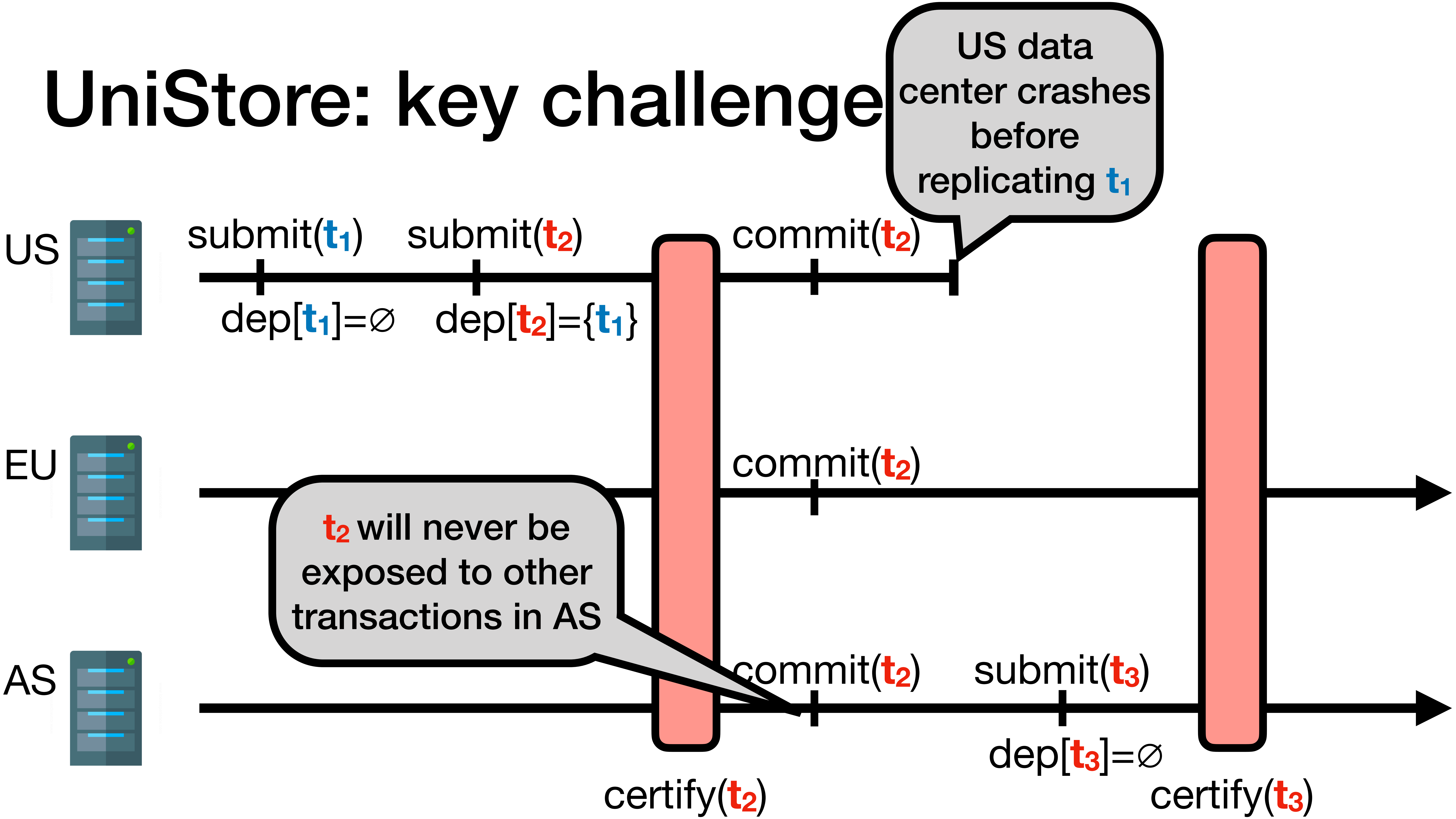
US data center crashes before replicating t_1

t_2 will never be exposed to other transactions in AS

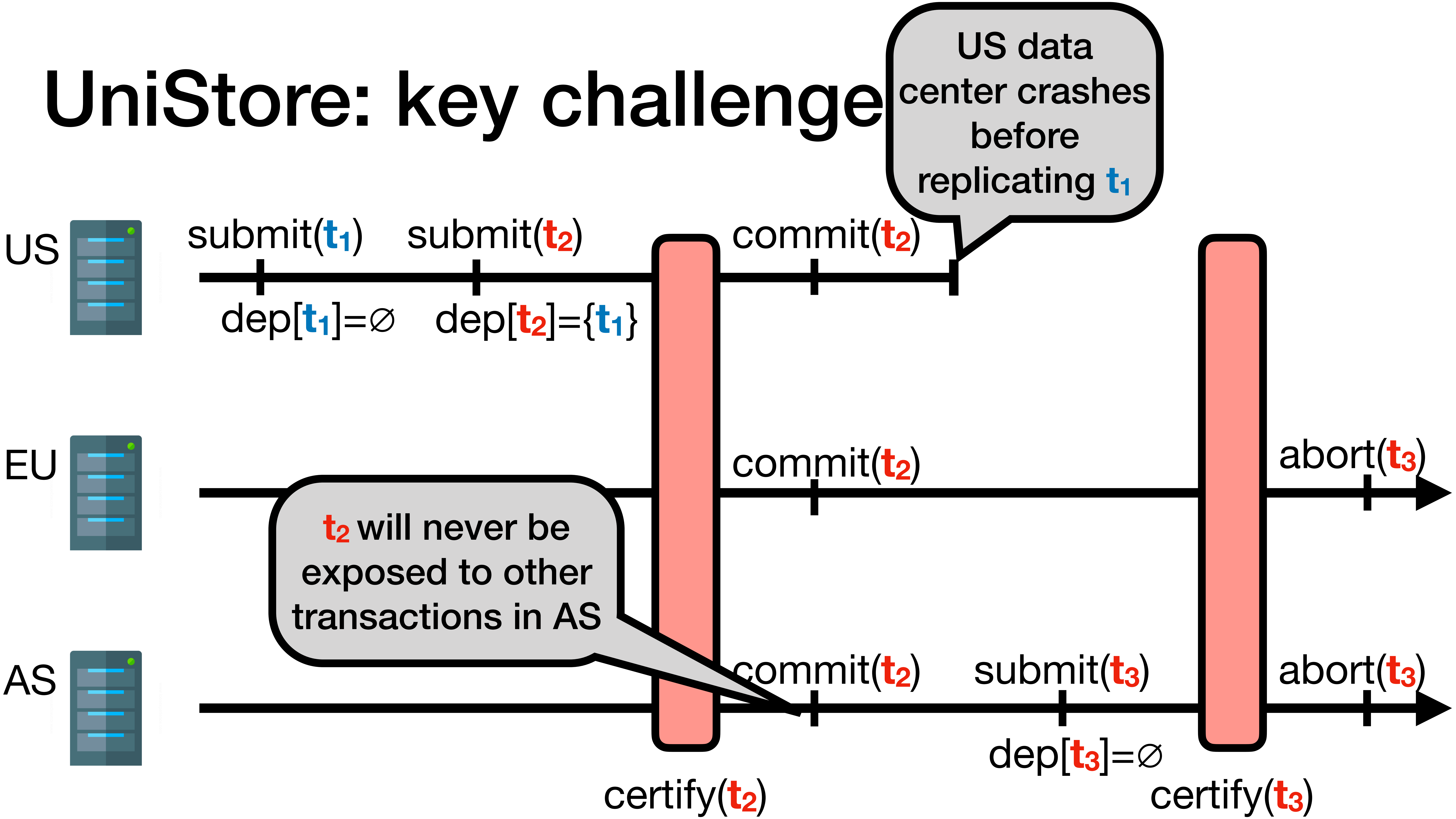
UniStore: key challenge



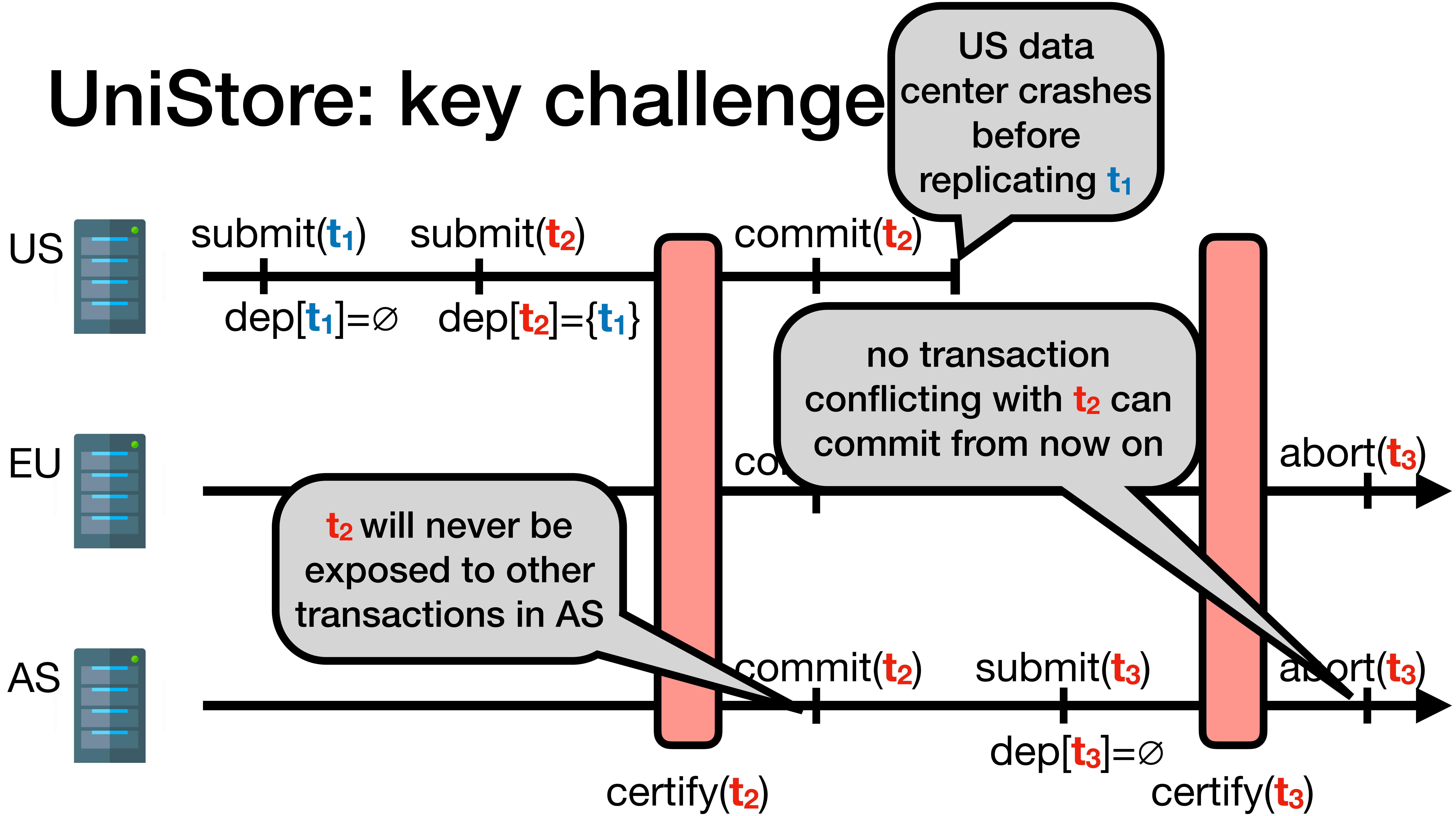
UniStore: key challenge



UniStore: key challenge



UniStore: key challenge



Related work

Related work

- Solutions that are **fault-tolerant** do **not** support **highly available causal** operations, and viceversa.

Related work

- Solutions that are **fault-tolerant** do **not** support **highly available causal** operations, and viceversa.

- **Previous solutions aren’t scalable**: do not include mechanisms for partitioning the key space among different machines in a data center or include per-data center centralized services

UniStore: uniformity

UniStore: uniformity

- UniStore ensures that all **causal dependencies** of a strong transaction are ***uniform*** **before certification**

UniStore: uniformity

- UniStore ensures that all **causal dependencies** of a strong transaction are ***uniform* before certification**

- A transaction is uniform if both **the transaction and its causal dependencies are guaranteed** to be **eventually replicated at all** correct data centers

UniStore: uniformity

- UniStore ensures that all **causal dependencies** of a strong transaction are ***uniform* before certification**

- A transaction is uniform if both **the transaction and its causal dependencies are guaranteed** to be **eventually replicated at all** correct data centers

- UniStore considers a transaction to be uniform **once it is visible at f + 1 data centers**, because at least one of these must be correct, and data centers can forward causal transactions to others

UniStore: other features

UniStore: other features

- **Causal transactions** execute in a **snapshot** that it is slightly **in the past** to minimise the latency of strong transactions

UniStore: other features

- **Causal transactions** execute in a **snapshot** that it is slightly **in the past** to minimise the latency of strong transactions

- UniStore uses a **fully-decentralized** and **lightweight** background stabilisation protocol to track uniformity

UniStore: other features

- **Causal transactions** execute in a **snapshot** that it is slightly **in the past** to minimise the latency of strong transactions

- UniStore uses a **fully-decentralized** and **lightweight** background stabilisation protocol to track uniformity

- It reuses the mechanism for tracking uniformity to let clients make **causal transaction durable on demand** and enable **consistent client migration**

Evaluation

Evaluation

- **Amazon EC2** using m4.2xlarge VMs from **3 different regions**: Virginia (US-East), California (US-West) and Frankfurt (EU-FRA)

Evaluation

- **Amazon EC2** using m4.2xlarge VMs from **3 different regions**: Virginia (US-East), California (US-West) and Frankfurt (EU-FRA)

- We use **RUBiS**, a popular benchmark that emulates an online auction website such as Ebay

Evaluation

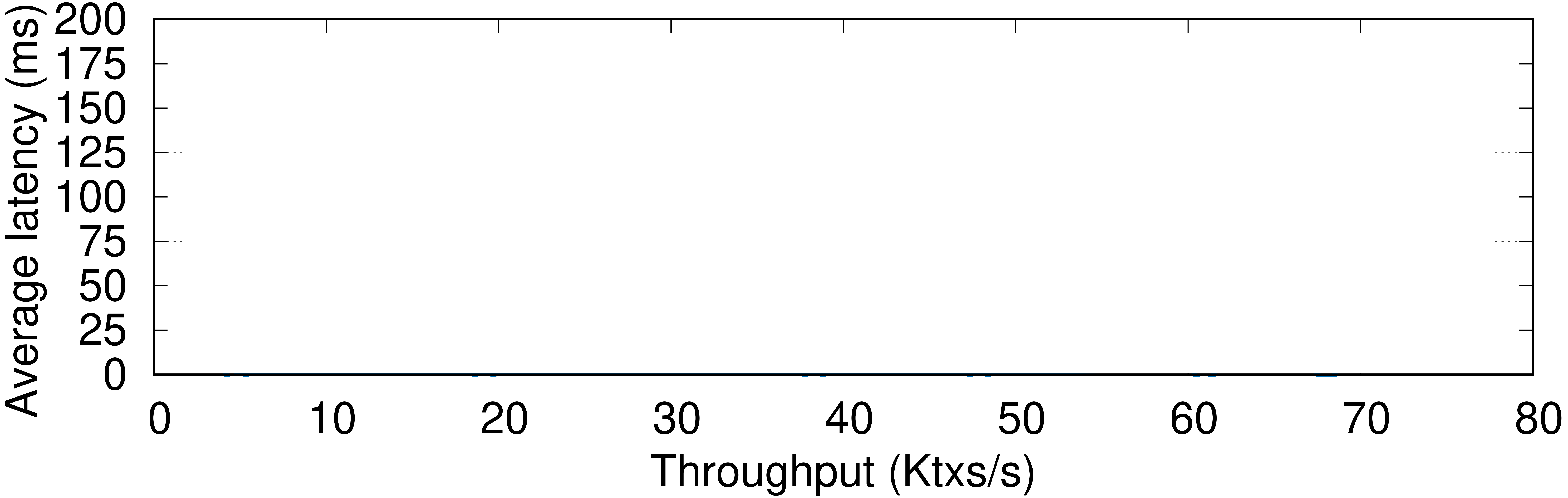
- **Amazon EC2** using m4.2xlarge VMs from **3 different regions**: Virginia (US-East), California (US-West) and Frankfurt (EU-FRA)

- We use **RUBiS**, a popular benchmark that emulates an online auction website such as Ebay

- Out of 15 transactions, **four transactions are strong** and declares **three conflicts** between them

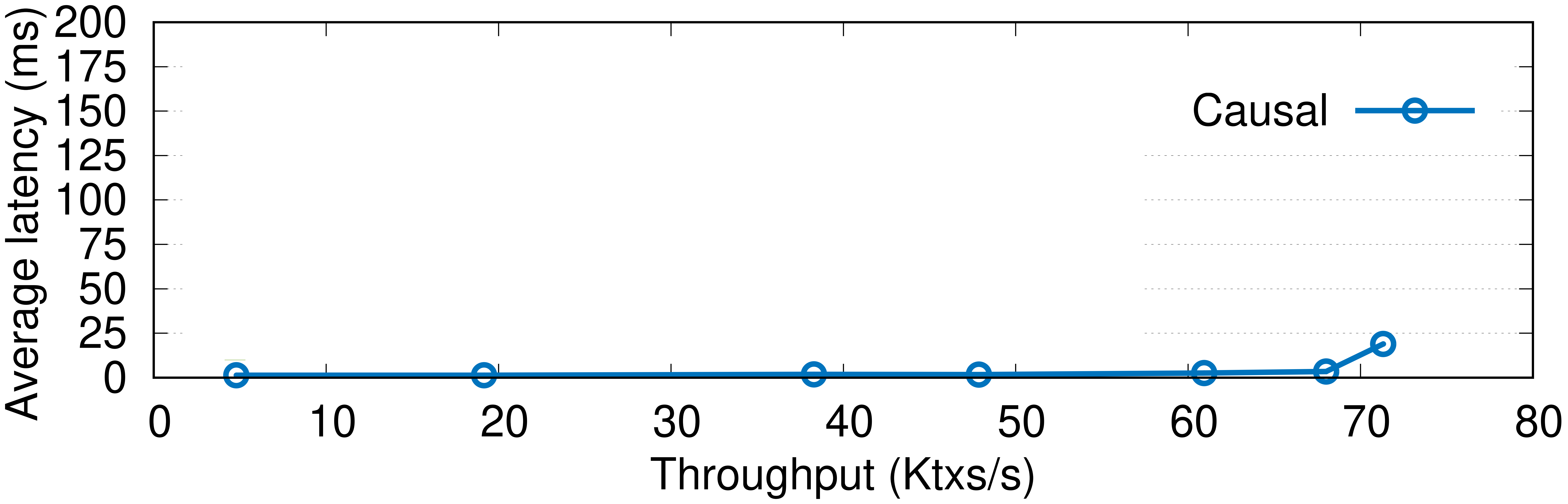
Evaluation: results

Mix workload with 15% of update transactions, which yields a **10% of strong transactions**



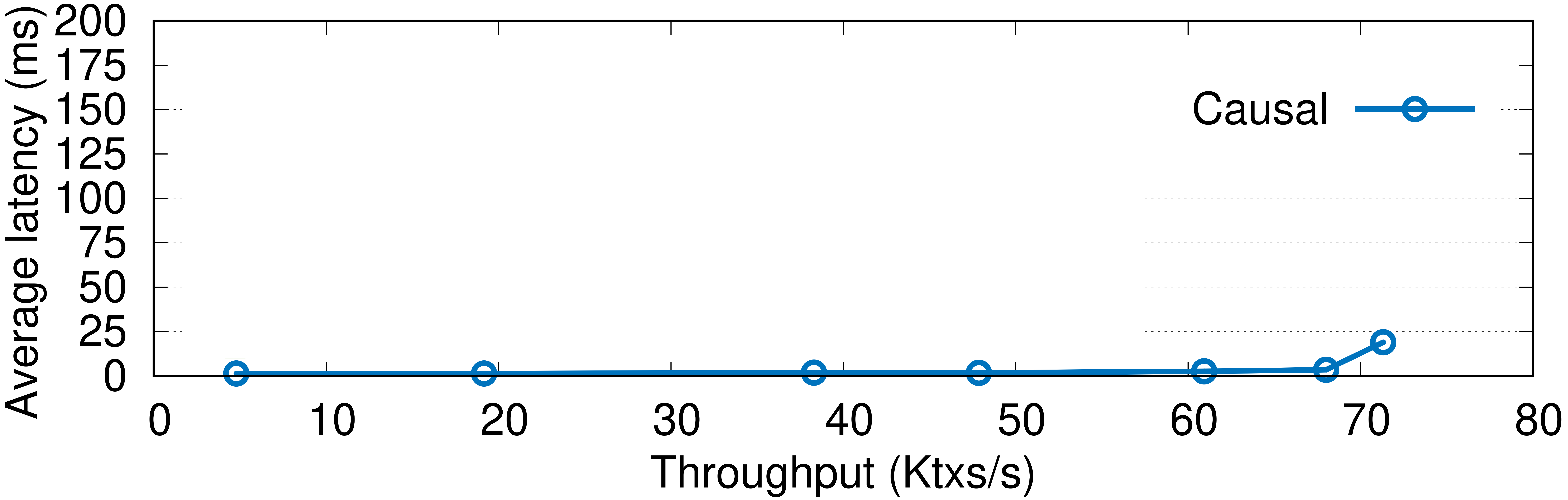
Evaluation: results

Causal implements causal consistency as a special case of UniStore where all transactions are causal



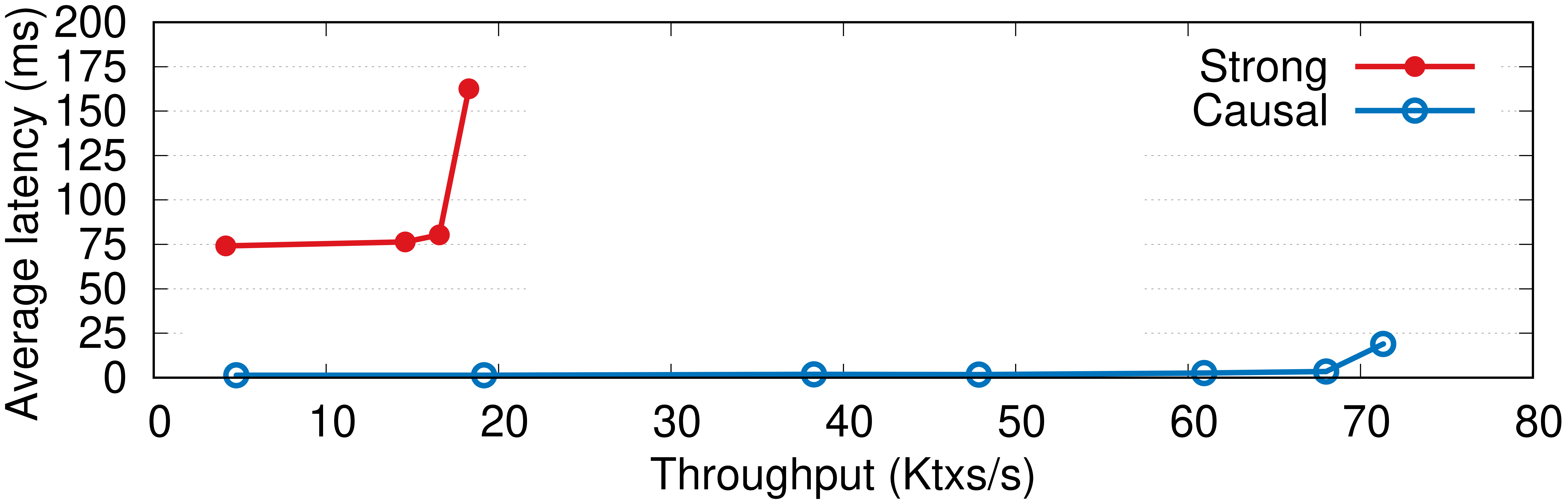
Evaluation: results

Causal cannot preserve the integrity invariants of RUBiS, but gives an upper bound on the expected performance.



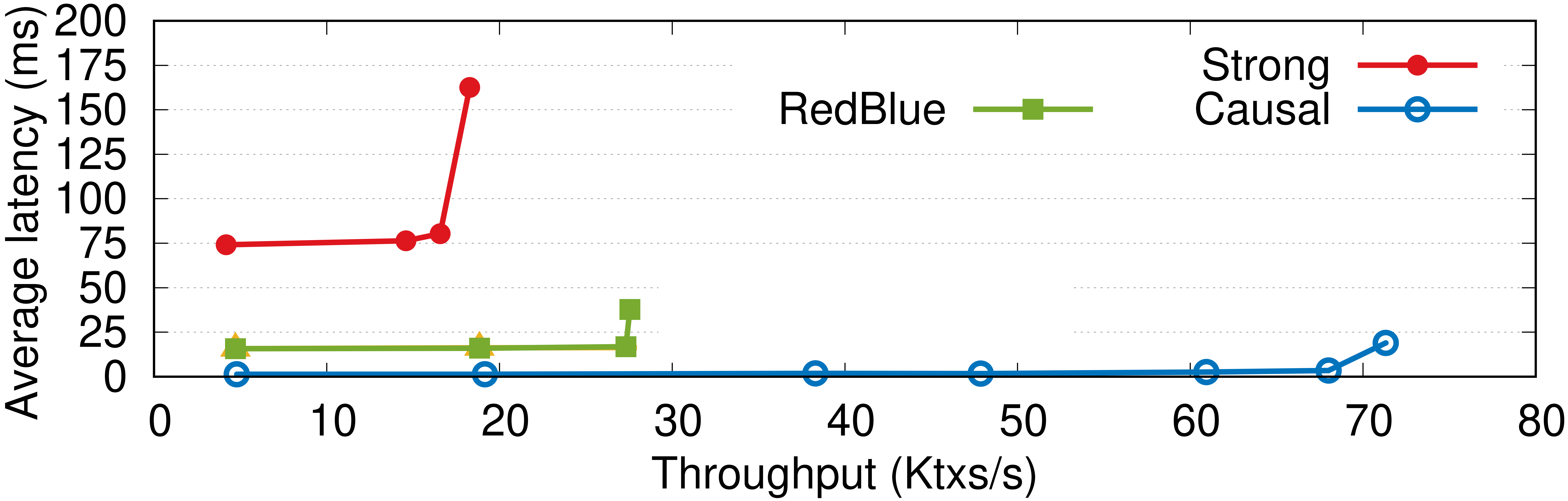
Evaluation: results

Strong implements serializability as a special case of UniStore where all transactions are strong



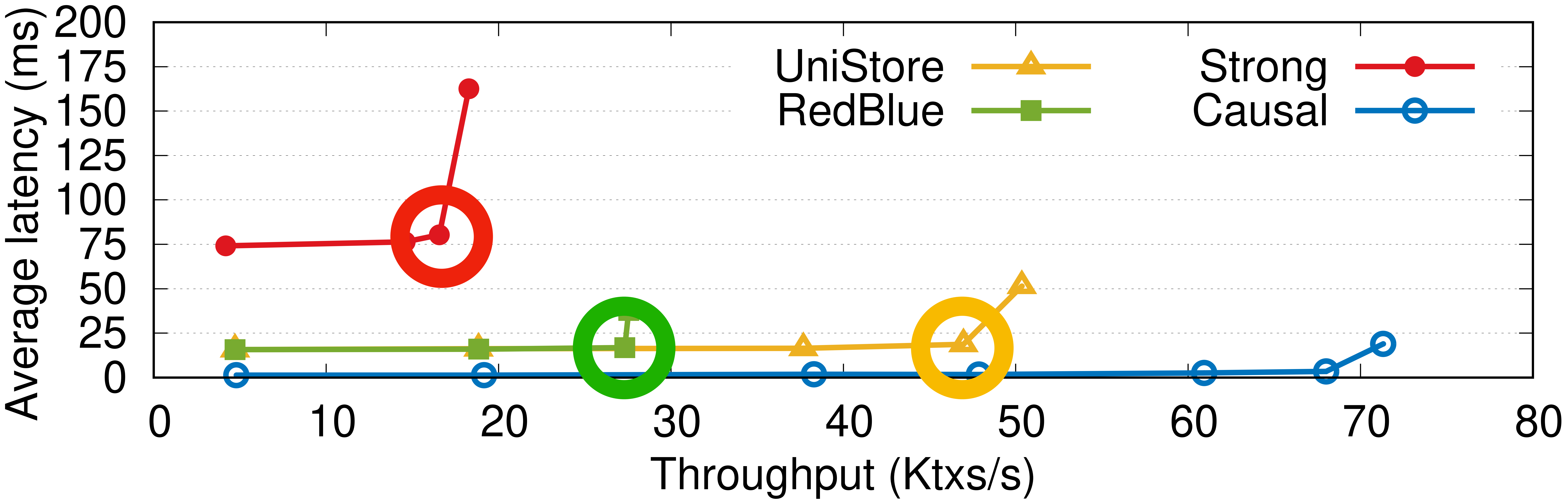
Evaluation: results

RedBlue implements redblue consistency,
which like PoR, combines causal and strong consistency.
However, it declares conflicts between all strong transactions.



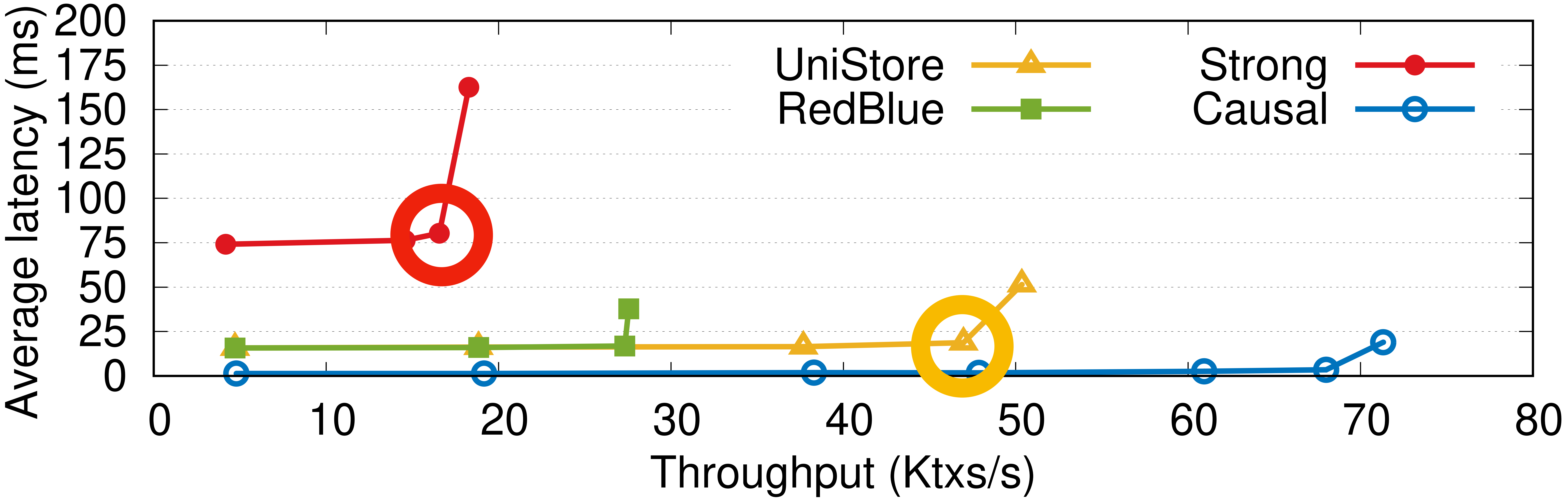
Evaluation: results

UniStore exhibits a high throughput:
72% and 183% higher than RedBlue and Strong
respectively at their saturation point.



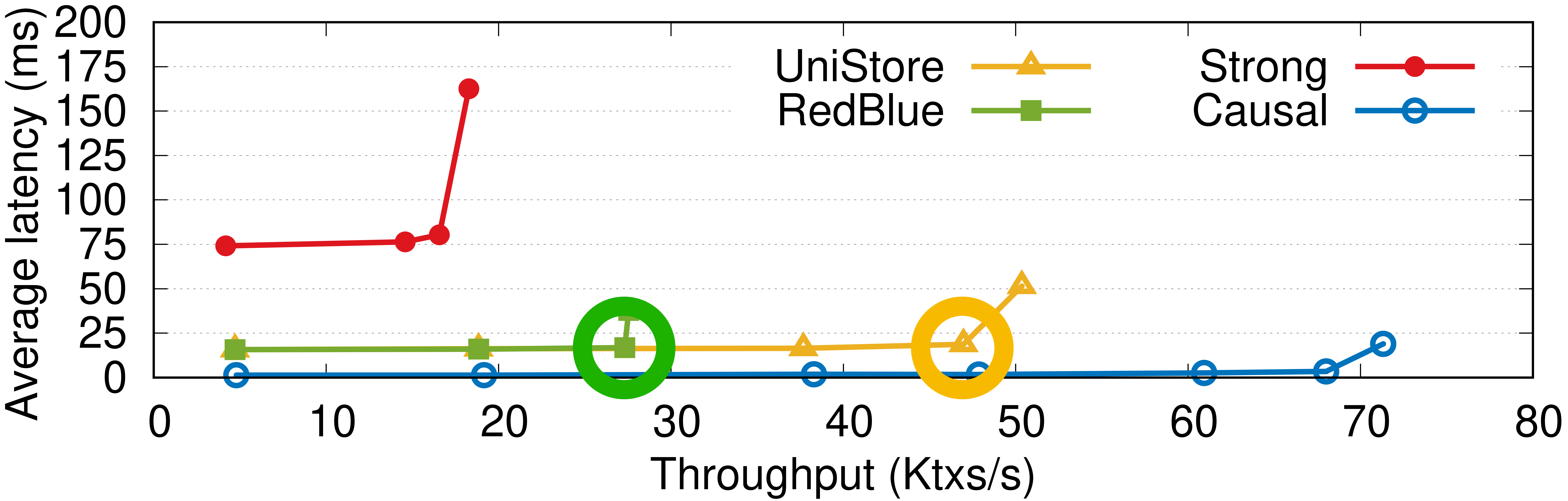
Evaluation: results

UniStore exhibits an average latency of 16.5ms, lower than 80.4ms of **Strong**



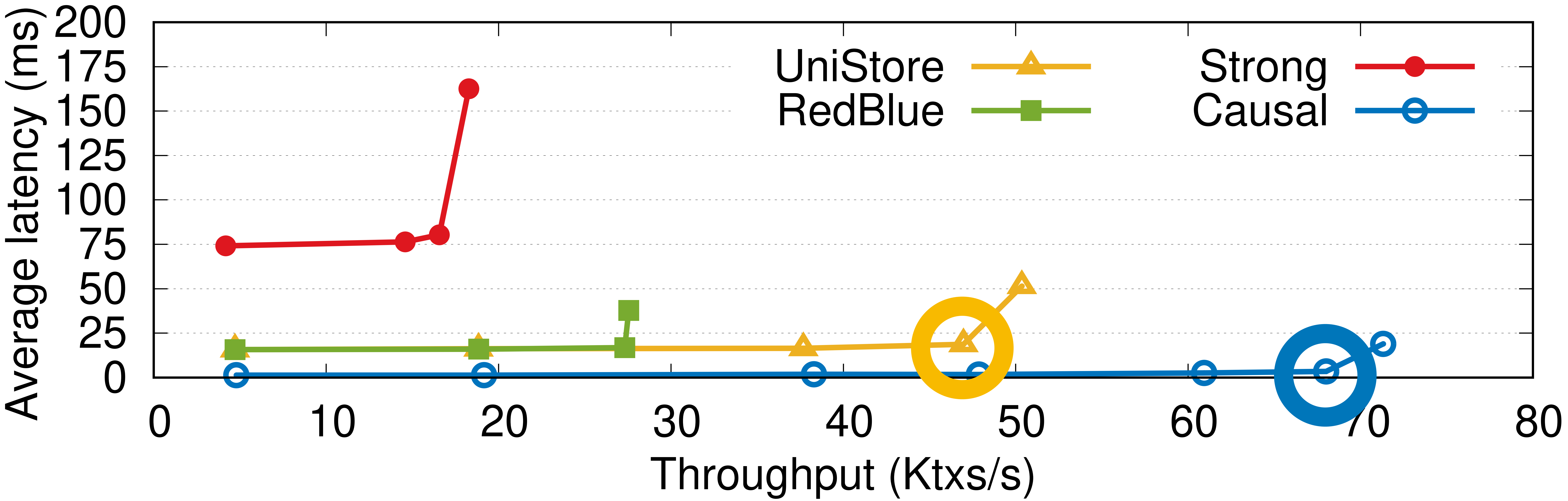
Evaluation: results

The latency of **RedBlue** is comparable to that of **UniStore**. This is because both systems mark the same set of transactions as strong



Evaluation: results

In comparison to **Causal**, **UniStore** penalizes throughput by 45%. This is the unavoidable price to pay to preserve application-specific invariants.



Evaluation: results

Evaluation: results

- In **UniStore**, strong transactions exhibit a latency of 73.9ms on average, which is dominated by the RTT between Virginia (the leader's region) and California (Virginia's closest data center) – 61ms

Evaluation: results

- In **UniStore**, strong transactions exhibit a latency of 73.9ms on average, which is dominated by the RTT between Virginia (the leader's region) and California (Virginia's closest data center) – 61ms
- Causal transactions exhibit a very low latency – 1.2ms on average, which is comparable to that of **Causal**

Evaluation: results

- In **UniStore**, strong transactions exhibit a latency of 73.9ms on average, which is dominated by the RTT between Virginia (the leader's region) and California (Virginia's closest data center) – 61ms

- Causal transactions exhibit a very low latency – 1.2ms on average, which is comparable to that of **Causal**

- This demonstrates that **UniStore** is able to mix causal and strong consistency effectively

Evaluation: results

- In UniStore, strong transactions exhibit a latency of 73.9ms on average, which is dominated by the RTT between Virginia (the leader's region) and California (Virginia's closest data center) – 61ms

- Causal transactions exhibit a very low latency – 1.2ms on average, which is comparable to that of **Causal**

- This demonstrates that UniStore is able to mix causal and strong consistency effectively

Conclusion

Conclusion

- UniStore is the first fault-tolerant and scalable data store that combines causal and strong consistency

Conclusion

- UniStore is the first fault-tolerant and scalable data store that combines causal and strong consistency

- It combines causal and strong consistency effectively: **3.7× lower latency** on average **than a strongly** consistent system with **1.2ms latency on average for causal** transactions

Conclusion

- UniStore is the first fault-tolerant and scalable data store that combines causal and strong consistency
- It combines causal and strong consistency effectively: **3.7× lower latency** on average **than a strongly** consistent system with **1.2ms latency on average for causal** transactions
- We expect that the key ideas in UniStore will **pave the way for practical systems that combine causal and strong consistency**

Thank you

Follow up questions to manuel.bravo@imdea.org

- UniStore is the first fault-tolerant and scalable data store that combines causal and strong consistency

- It combines causal and strong consistency effectively: **3.7× lower latency** on average **than a strongly** consistent system with **1.2ms latency on average for causal** transactions

- We expect that the key ideas in UniStore will **pave the way for practical systems that combine causal and strong consistency**