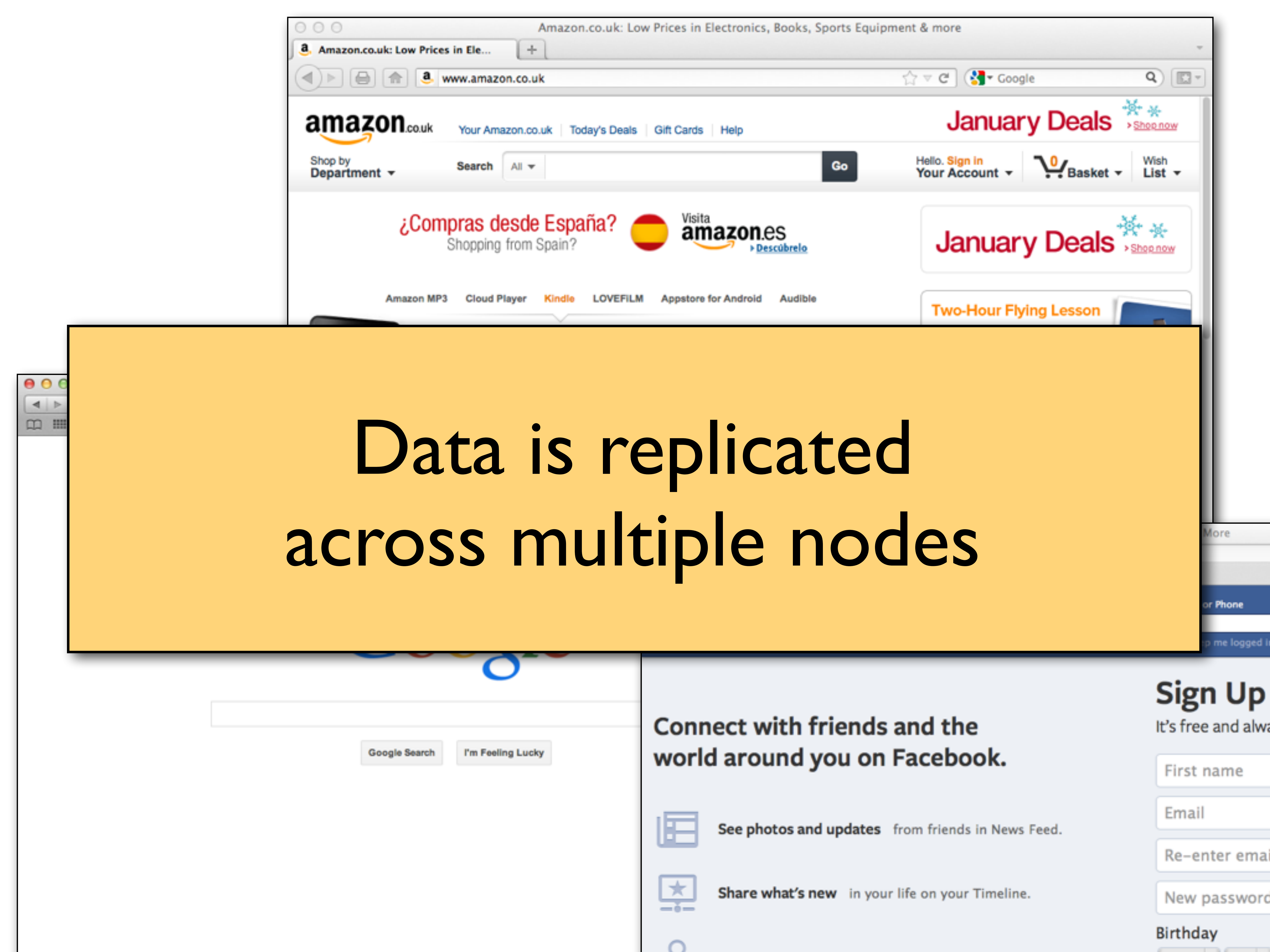
A Framework for Transactional Consistency Models with Atomic Visibility

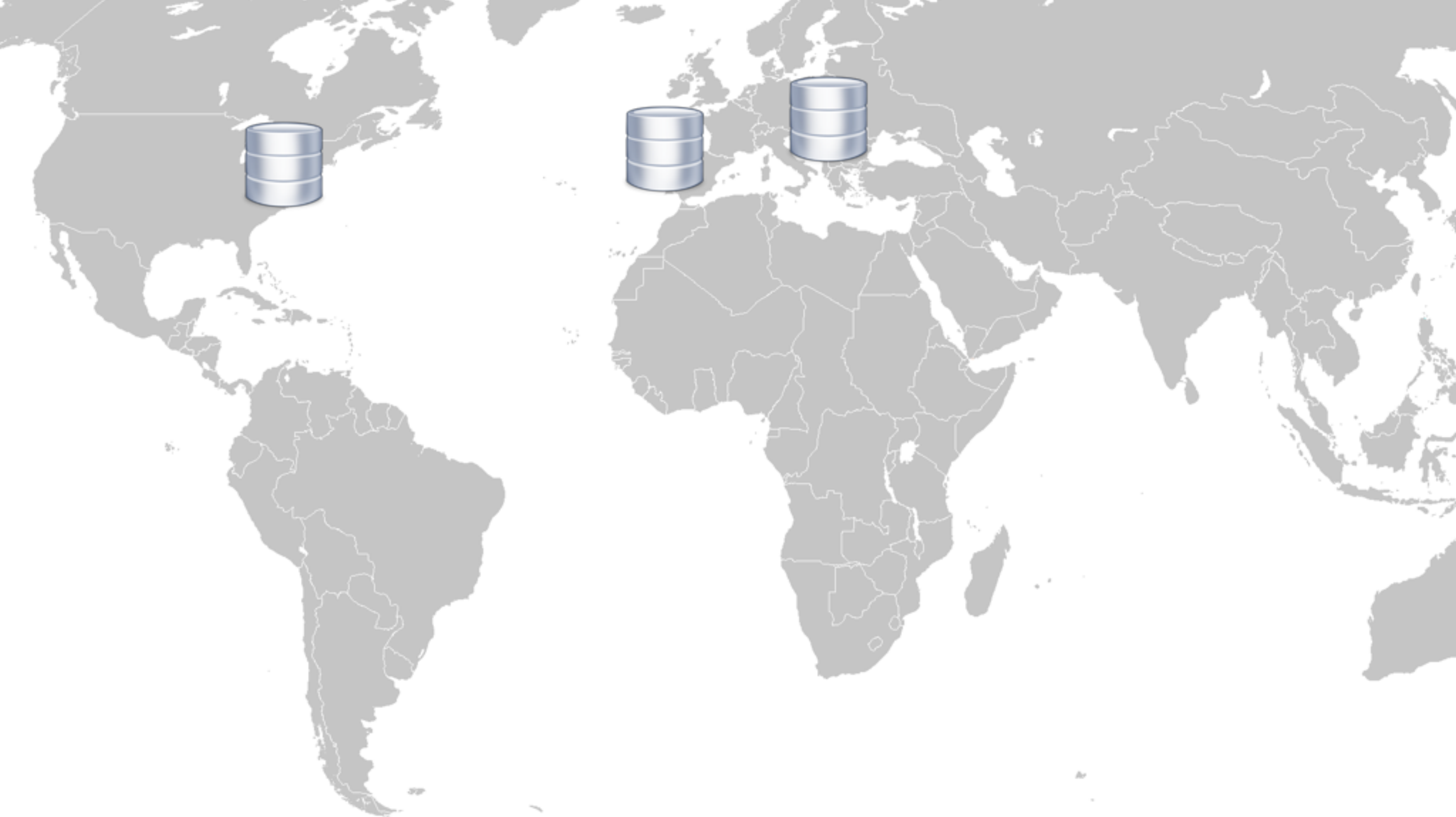
Andrea Cerone, Giovanni Bernardi, Alexey Gotsman
IMDEA Software Institute, Madrid, Spain

CONCUR - Madrid, September 1st 2015

Data is replicated across multiple nodes

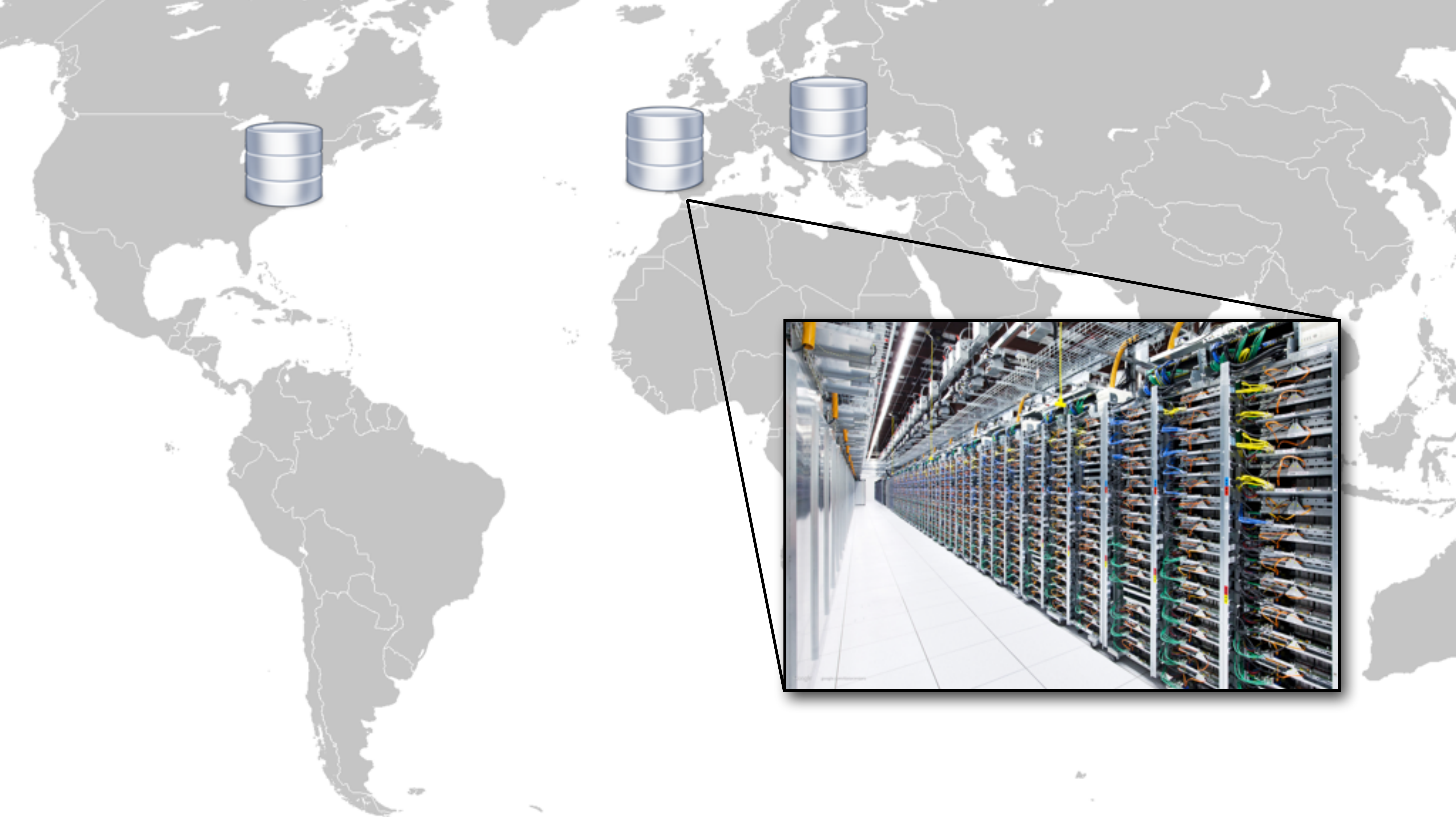


Data centres across the world



Disaster-tolerance, minimising latency

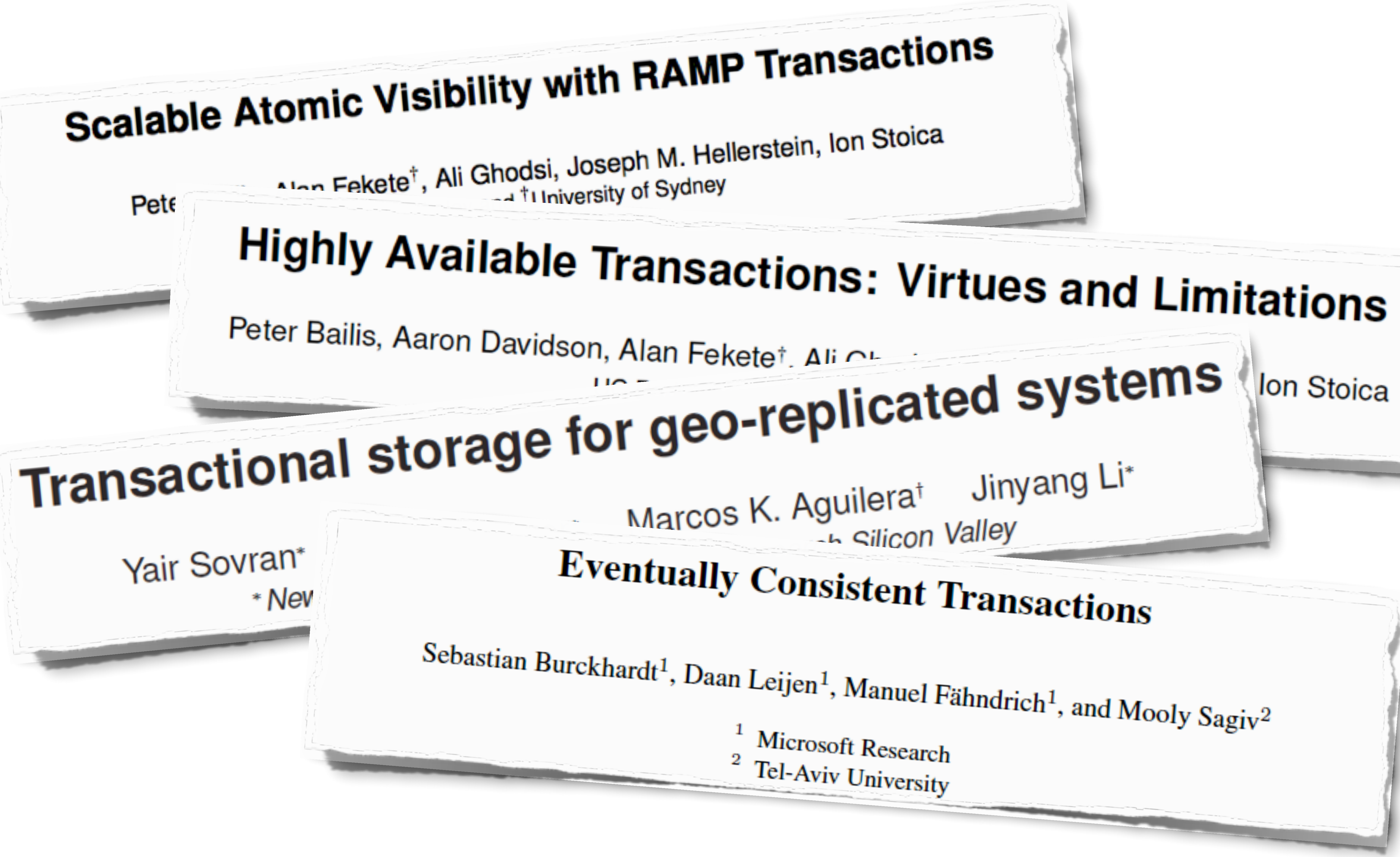
With thousands of machines inside



Fault-tolerance, load-balancing

- Serialisability: the system behaves like a serial processor of transactions on a centralised database
- Requires synchronisation: expensive

Rethinking consistency in large-scale



The database gives weaker guarantees to programmers

Weak Consistency Models

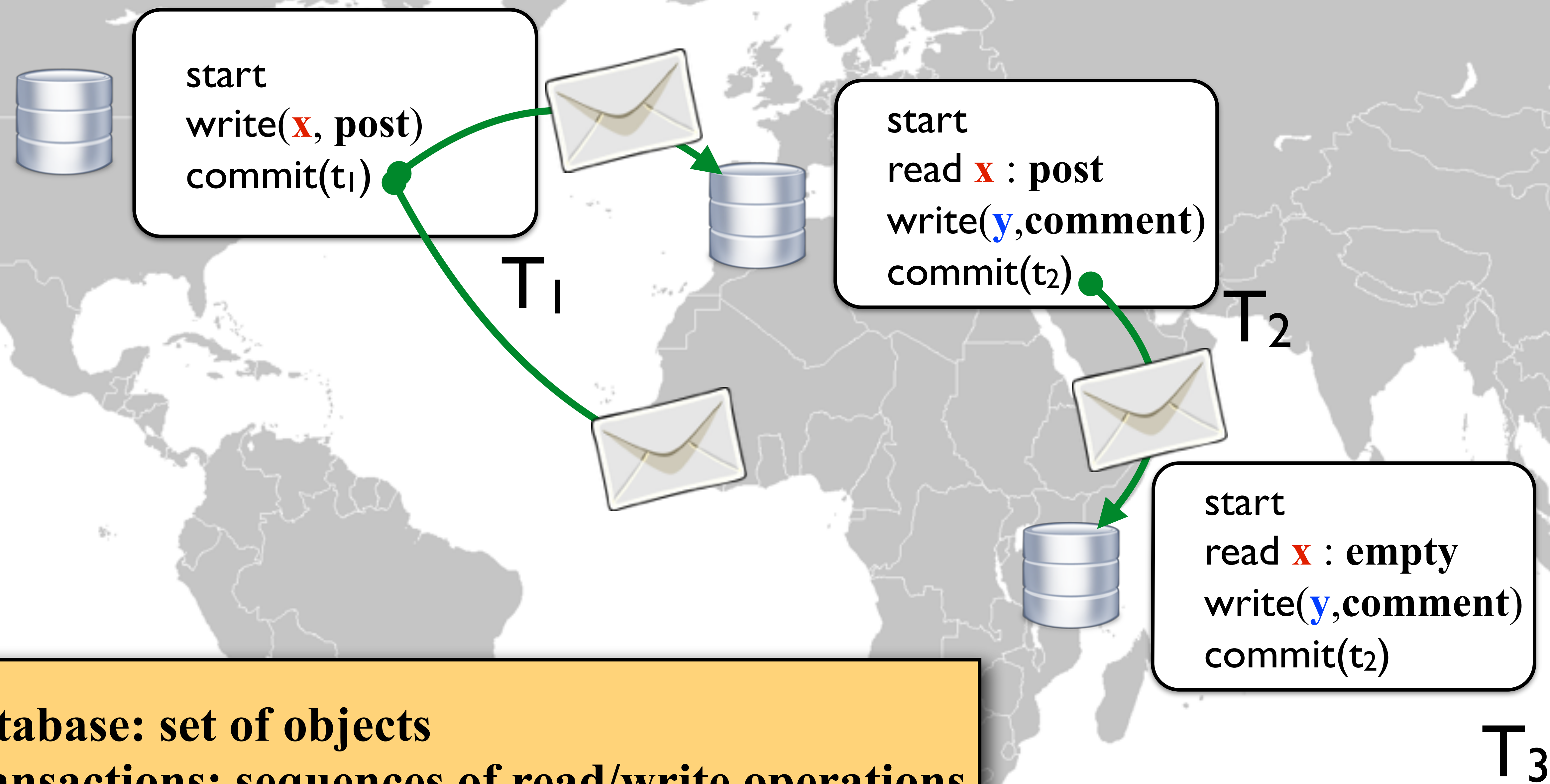
✓ Performance boost

- require less synchronisation between replicas

✗ Anomalous behaviour

- executions which are not allowed by a serialisable database

Anomalies



Database: set of objects
Transactions: sequences of read/write operations

Non-serialisable execution
Causality is violated

- Consistency models: specified informally or using disparate formalism

This talk:

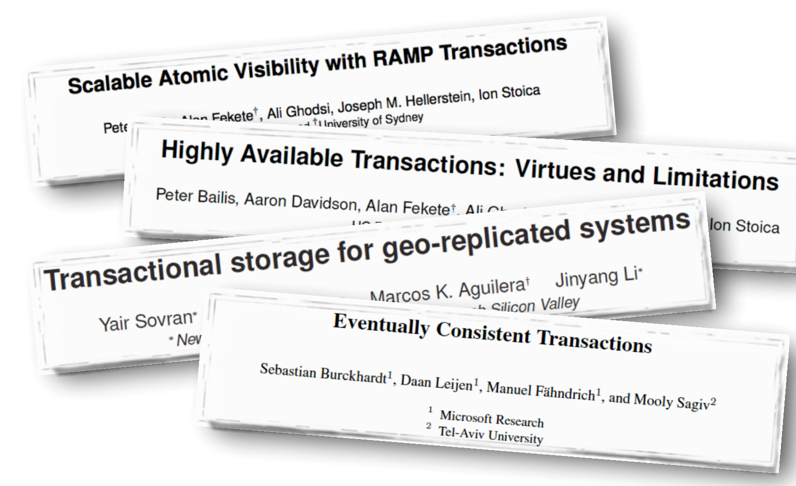
- A framework for specifying transactional consistency models
- A pseudo-implementation of such consistency models
- Correctness of the implementation with respect to the specification (for any consistency model)

Abstract Framework

Desired features:

- Abstract from implementation dependent details
  (replicas, synchronisation events, …)

- Expressive enough to formalise practical consistency models



- Concise specifications

Abstract Framework

- An execution models the dependencies between transactions in a run of the system
  ~ weak memory models

- A consistency model is specified as the set of executions it allows

Abstract Framework

Transactions:

$T = (E, po)$

T: $\text{read } x{:}\,0 \xrightarrow{po} \text{write}(y, 1)$

No DB events (start, abort, commit)
We record only committed transactions

$\text{read } x{:}\,0 \xrightarrow{po} \text{read } x{:}\,1$ ✖

value of read operations coincide with the value of the last operation on the same object

value of x changed by external entity

Abstract Framework

Transactions:

T = (E, po)

No DB events (start, abort, commit)
We record only committed transactions

T: read **x**: 0 $\xrightarrow{po}$ write(**y**, 1)

S: read **x**: 0 $\xrightarrow{po}$ write(**x**, 1) $\xrightarrow{po}$ read **x**: 1 $\xrightarrow{po}$ write(**x**, 2)

Reads an external value

S ⊢ Read **x**: 0

Reads a value written by the same transaction

Abstract Framework

Transactions:

$T = (E, po)$

T: read **x**: 0 $\xrightarrow{po}$ write(**y**, 1)

No DB events (start, abort, commit)
We record only committed transactions

S: read **x**: 0 $\xrightarrow{po}$ write(**x**, 1) $\xrightarrow{po}$ read **x**: 1 $\xrightarrow{po}$ write(**x**,2)

Gets overwritten later: not observable by other transactions

Last write to object: can be observed by other transactions

$S \vdash$ Write **x**: 2

Abstract Framework

Transactions:

$T = (E, po)$

No DB events (start, abort, commit)
We record only committed transactions

T: read x: 0 $\xrightarrow{po}$ write(y, 1)

Atomic Visibility:

S: read x: 0 $\xrightarrow{po}$ write(x, 1) $\xrightarrow{po}$ read x: 1 $\xrightarrow{po}$ write(x,2)

$S \vdash$ Read x: 0 $\qquad$ $S \vdash$ Write x: 2

Abstract Framework

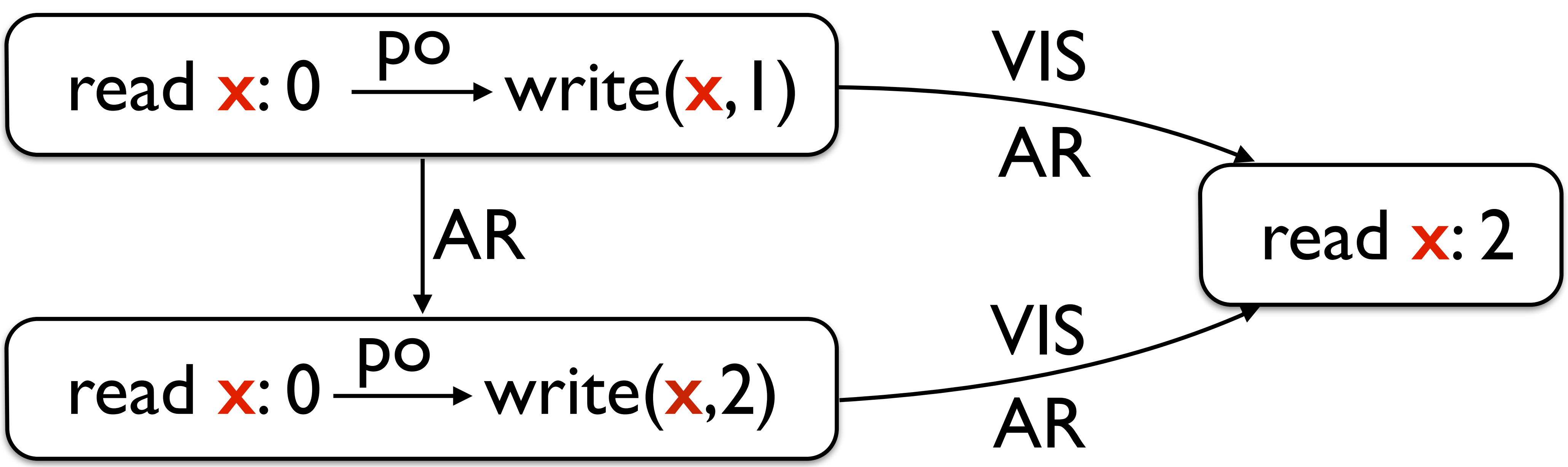
Executions: (H, VIS, AR)

H: Set of transactions {S, T, . . .}

$S \xrightarrow{VIS} T$: T sees the updates of S

$S \xrightarrow{AR} T$: keeps track of version order

VIS $\subseteq$ AR   AR is total



Abstract Framework

Executions: (H, VIS, AR)

H: Set of transactions {S, T, …}

$S \xrightarrow{\text{VIS}} T$: T sees the updates of S

$S \xrightarrow{\text{AR}} T$: keeps track of version order

VIS $\subseteq$ AR   AR is total

$$\forall T \in \mathcal{H}. \forall x, n. \, T \vdash \text{Read } x : n \implies$$
$$((\text{VIS}^{-1}(T) \cap \{S \mid S \vdash \text{Write } x : \_\} = \emptyset \wedge n = 0) \vee$$
$$\max_{\text{AR}}(\text{VIS}^{-1}(T) \cap \{S \mid S \vdash \text{Write } x : \_\}) \vdash \text{Write } x : n)$$

Read Atomic (RA): Baseline Consistency Model

Consistency Models

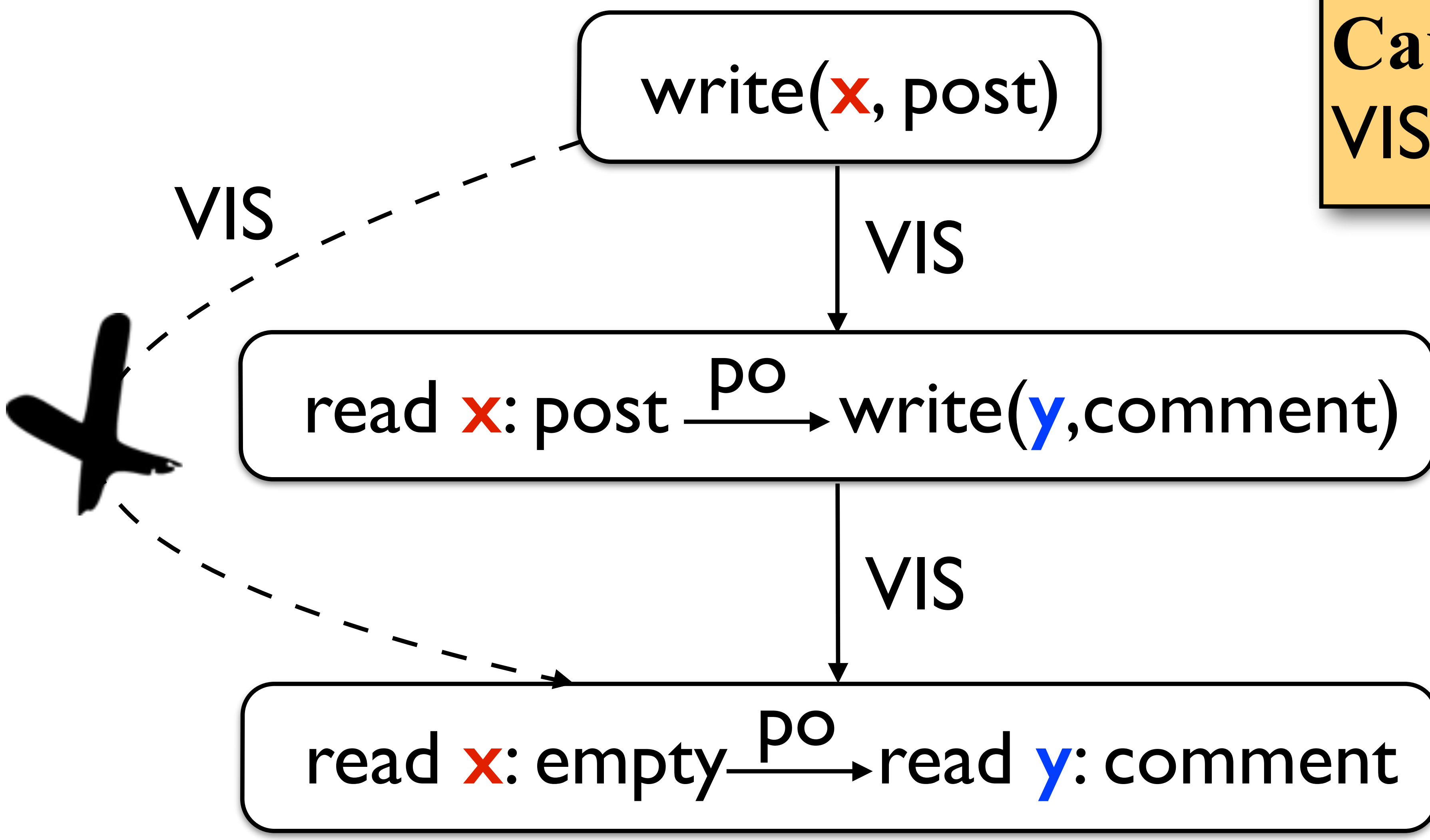
- Specification given by restraining VIS and AR

Example: Serialisability

VIS is a total order

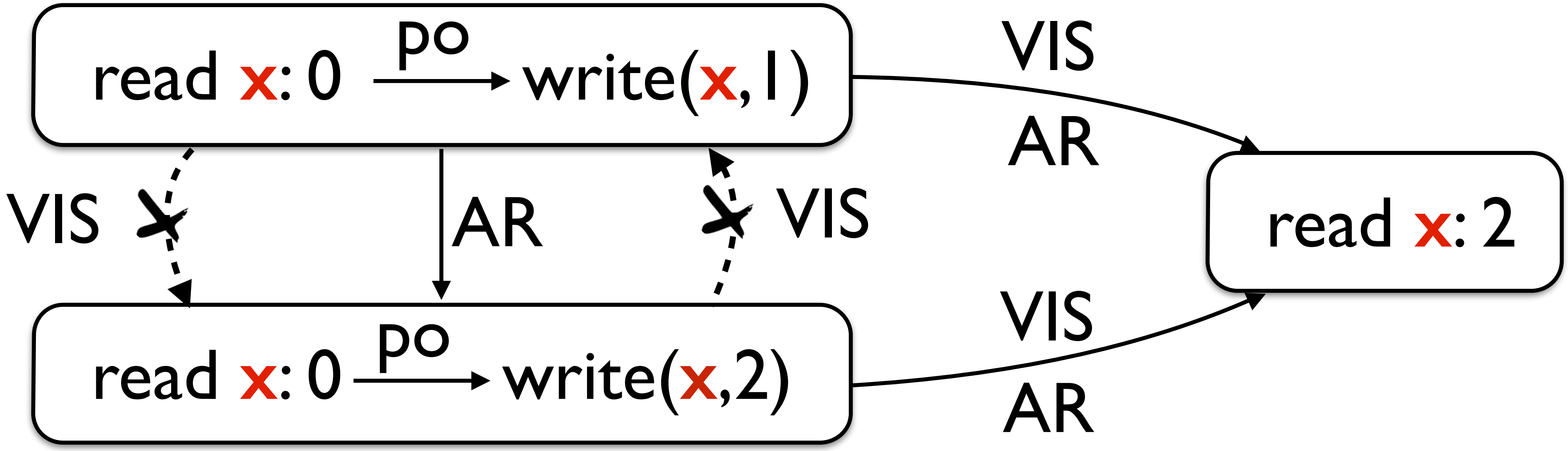
- Different consistency models allow different anomalies

Violation of Causality



Causal Consistency:
VIS is transitive

Lost Update



Parallel Snapshot Isolation:

VIS is transitive + Write-write conflict detection:
if $S \vdash$ Write x:_ , $T \vdash$ Write x:_ **and** $S \neq T$
then either $S \xrightarrow{VIS} T$, **or** $T \xrightarrow{VIS} S$

Consistency Models

- Specification given by restraining VIS and AR

Consistency Model	Constraint
• Read Atomic	None
• Causal Consistency	VIS is transitive
• Parallel Snapshot Isolation	VIS is transitive Write-write conflict detection

Consistency Models

- Specification given by restraining VIS and AR

Co

$\forall (E, \mathsf{po}) \in \mathcal{H}.\, \forall e \in E.\, \forall x, n.\, (\mathsf{op}(e) = \mathtt{read}(x, n) \wedge (\mathsf{po}^{-1}(e) \cap \mathsf{HEvent}_x \neq \emptyset)) \implies \mathsf{op}(\max_{\mathsf{po}}(\mathsf{po}^{-1}(e) \cap \mathsf{HEvent}_x)) = \_(x, n)$ (INT)		
$\forall T \in \mathcal{H}.\, \forall x, n.\, T \vdash \mathsf{Read}\ x : n \implies ((\mathsf{VIS}^{-1}(T) \cap \{S \mid S \vdash \mathsf{Write}\ x : \_\} = \emptyset \wedge n = 0) \vee \max_{\mathsf{AR}}(\mathsf{VIS}^{-1}(T) \cap \{S \mid S \vdash \mathsf{Write}\ x : \_\}) \vdash \mathsf{Write}\ x : n)$ (EXT)		
VIS is transitive (TRANSVIS)	$\mathsf{AR}; \mathsf{VIS} \subseteq \mathsf{VIS}$ (PREFIX)	VIS is total (TOTALVIS)
$\forall T, S \in \mathcal{H}.\, (T \neq S \wedge T \vdash \mathsf{Write}\ x : \_ \wedge S \vdash \mathsf{Write}\ x : \_) \implies (T \xrightarrow{\mathsf{VIS}} S \vee S \xrightarrow{\mathsf{VIS}} T)$ (NOCONFLICT)		

-
-

- Parallel Snapshot Isolation — VIS is transitive; Write-write conflict detection

Why should you trust me?

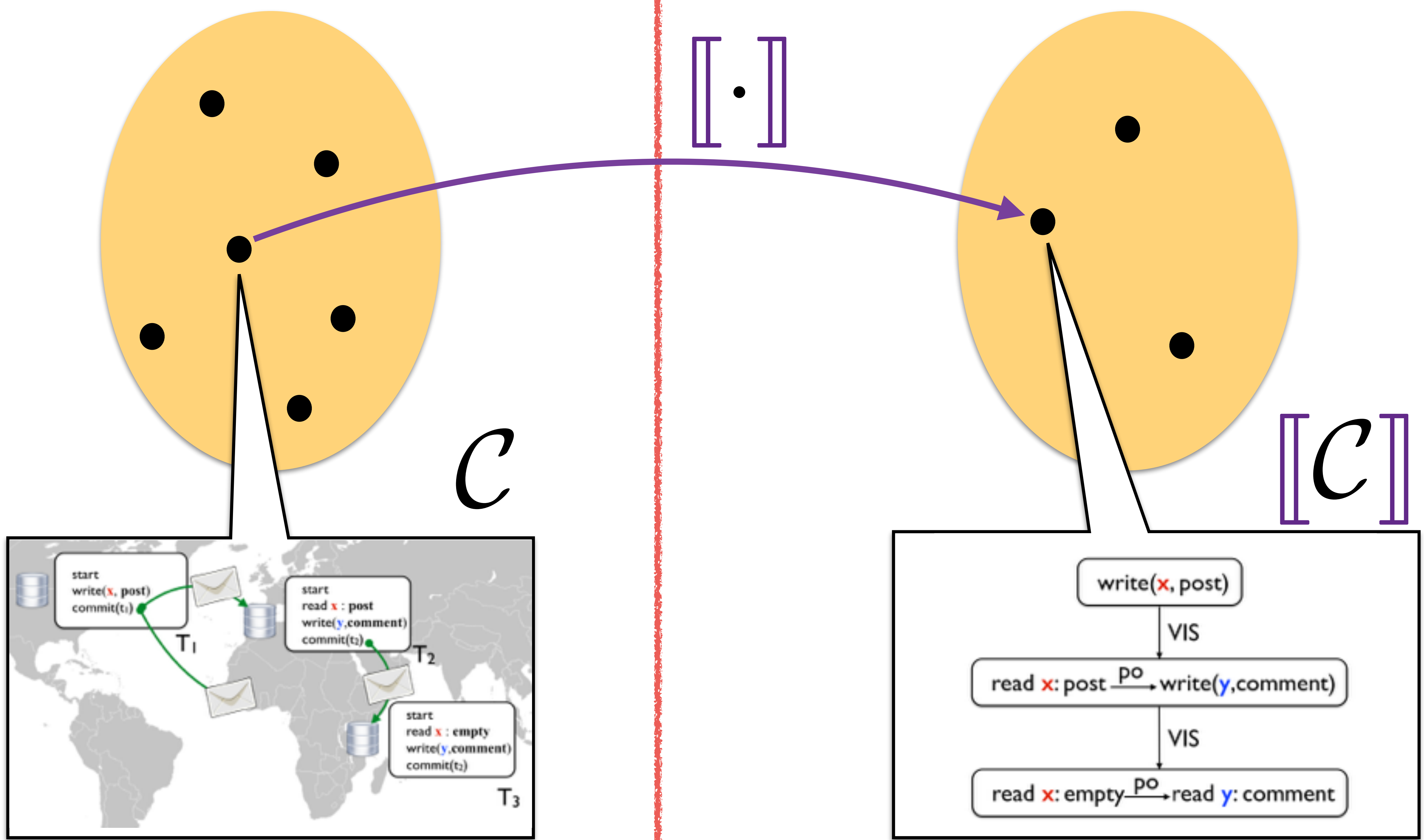
- Do the formal specifications really correspond to the informal ones?

Operational Model

- Used to define a pseudo-implementation of consistency models
- Modelled after real implementations

Implementation

Abstract Verification



Operational Model - Replicas

start
write(x,1)
read x :1
write(y,2)

x = 0, ts: 0   y = 0, ts: 0

```
write(x, 1) · write(y,2)
```

- Replicas store a copy of the database
  each object has a value and a timestamp

- Transactions: issued by clients and processed sequentially by a replica
  (a replica can be either idle or executing a transaction)

- Transaction log: keeps track of operations performed by pending transaction

- Read from transaction log first

Operational Model - Replicas

start
write(x,1)
read x :1
write(y,2)
commit(1)

x = 1, ts: 1   y = 2, ts: 1

______

upon commit:

```
generate timestamp
update state of the DB
clean transaction log
broadcast(timestamp: transaction log)
```

generate timestamp → monotonically increasing

Effects sent in a single message: ensures Atomic Visibility

Operational Model - Replicas

start
write(x,1)
read x :1
write(y,2)
abort

x = 0, ts: 0   y = 0, ts: 0

______

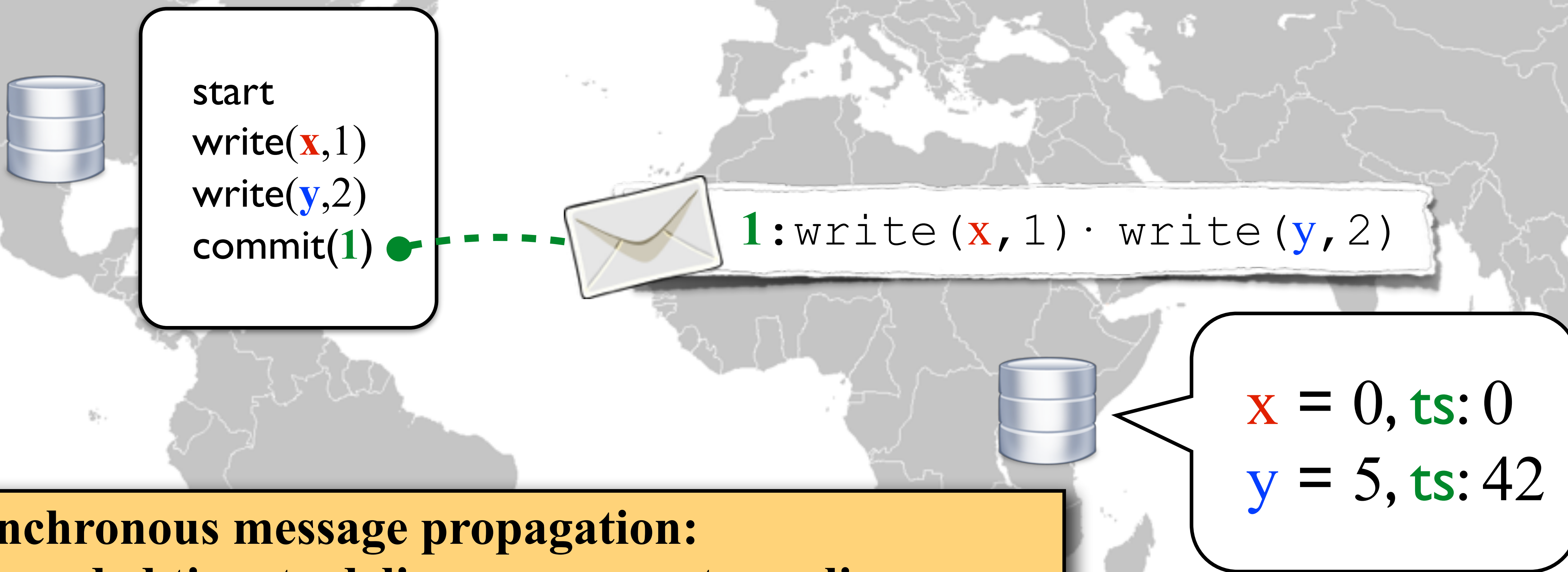
upon commit:

```
generate timestamp
update state of the DB
clean transaction log
broadcast(timestamp: transaction log)
```

upon abort:

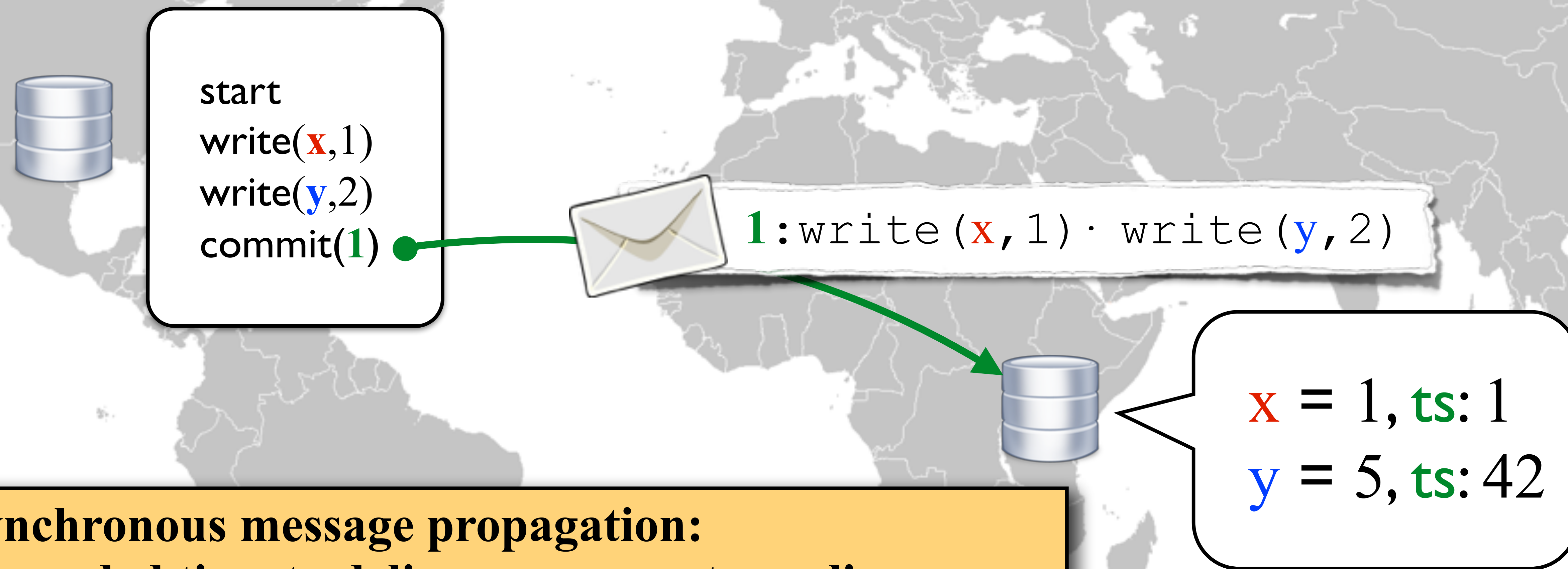
~~generate timestamp~~
~~update state of the DB~~
clean transaction log
~~broadcast(timestamp: transaction log)~~

Operational Model - Message Delivery



Asynchronous message propagation:
unbounded time to deliver messages to replicas

Operational Model - Message Delivery

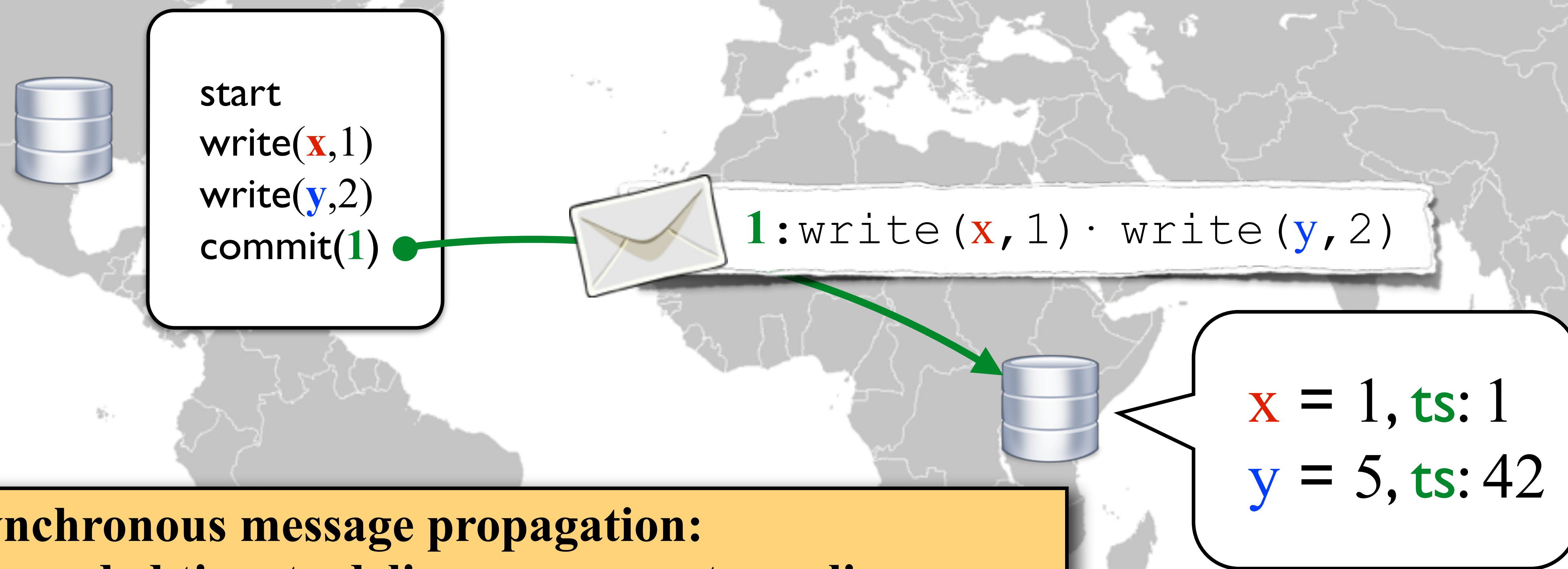


Asynchronous message propagation:
unbounded time to deliver messages to replicas

upon receive(ts:log)

```
for each write(obj,val) in log
 if (timestamp(obj) > ts)
  obj := val
  timestamp(obj) := ts
```

Operational Model - Message Delivery

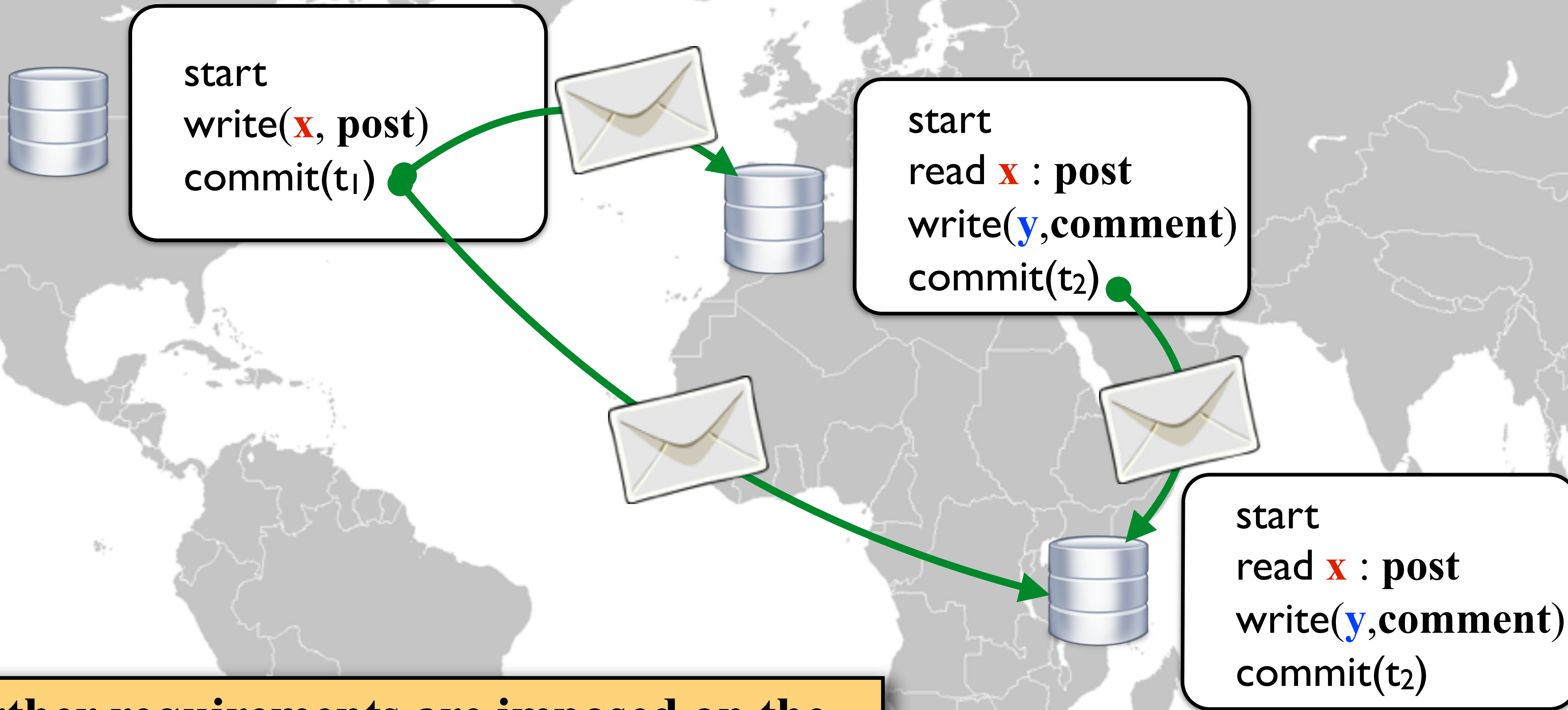


Asynchronous message propagation:
unbounded time to deliver messages to replicas

CONSTRAINT:
no messages are delivered while transactions are executing

Operational Model for Read Atomic

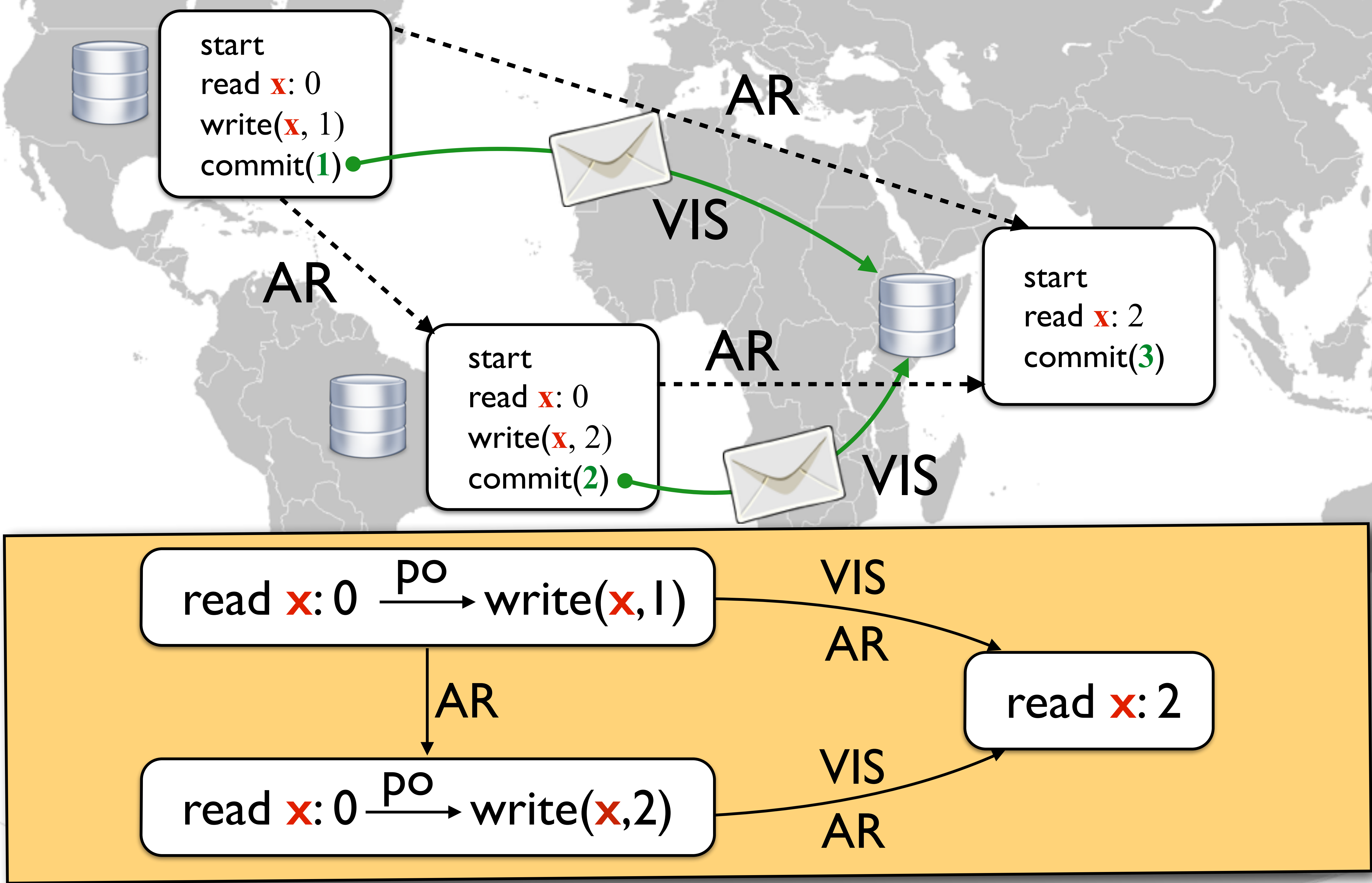
Consistency Models



Further requirements are imposed on the communication protocol to capture other consistency models

Causal Consistency:
Message delivery is Causal

From operational to abstract



Theorem

For any consistency model Φ

- $\mathcal{C}$ is an execution in the operational model for Φ

  implies that $[\![\mathcal{C}]\!]$ is an abstract execution for Φ

- $\mathcal{A}$ is an abstract execution for Φ

  implies that $\exists \mathcal{C}. [\![\mathcal{C}]\!] = \mathcal{A}$ and

  $\mathcal{C}$ is an execution in the operational model for Φ

Why should I care?

- Reasoning techniques for programs running on weak consistency models
- In the paper: A simple application aimed at optimising transaction executions
- Robustness: Applications run on a given consistency model without anomalies (Giovanni Bernardi's talk at YR-Concur)
- Optimising transactional applications via transaction chopping (A. Cerone, A. Gotsman and H. Yang, DISC 2015)

THANK YOU!