Transaction Chopping for Parallel Snapshot Isolation

Andrea Cerone,
Alexey Gotsman

Hongseok Yang

institute imdea software

DEPARTMENT OF COMPUTER SCIENCE

DISC - Tokyo - October 8th, 2015

Amazon.co.uk: Low Prices in Electronics, Books, Sports Equipment & more

Amazon.co.uk: Low Prices in Ele...

www.amazon.co.uk

Google

amazon.co.uk

Your Amazon.co.uk | Today's Deals | Gift Cards | Help

January Deals › Shop now

Shop by Department

Search All Go

Hello. Sign in Your Account

0 Basket

Wish List

¿Compras desde España?
Shopping from Spain?

Visita amazon.es › Descúbrelo

Amazon MP3 | Cloud Player | Kindle | LOVEFiLM | Appstore for Android | Audible

Meet the Kindle Family

January Deals › Shop now

Two-Hour Flying Lesson
Take to the skies!
£99 (was £299)
› See the deal
amazonlocal

SHAMBALLA BRACELETS
› Shop now

Google

https www.google.com/?gws_rd=ssl

Reader

Apple Yahoo! Google Maps YouTube Wikipedia News Popular

+You Gmail Images Sign in

Google

Google Search | I'm Feeling Lucky

Welcome to Facebook – Log In, Sign Up or Learn More

https www.facebook.com

Apple Yahoo! Google Maps YouTube Wikipedia News Popular

facebook

Email or Phone

Keep me logged i

Connect with friends and the world around you on Facebook.

See photos and updates from friends in News Feed.

Share what's new in your life on your Timeline.

Sign Up

It's free and alwa

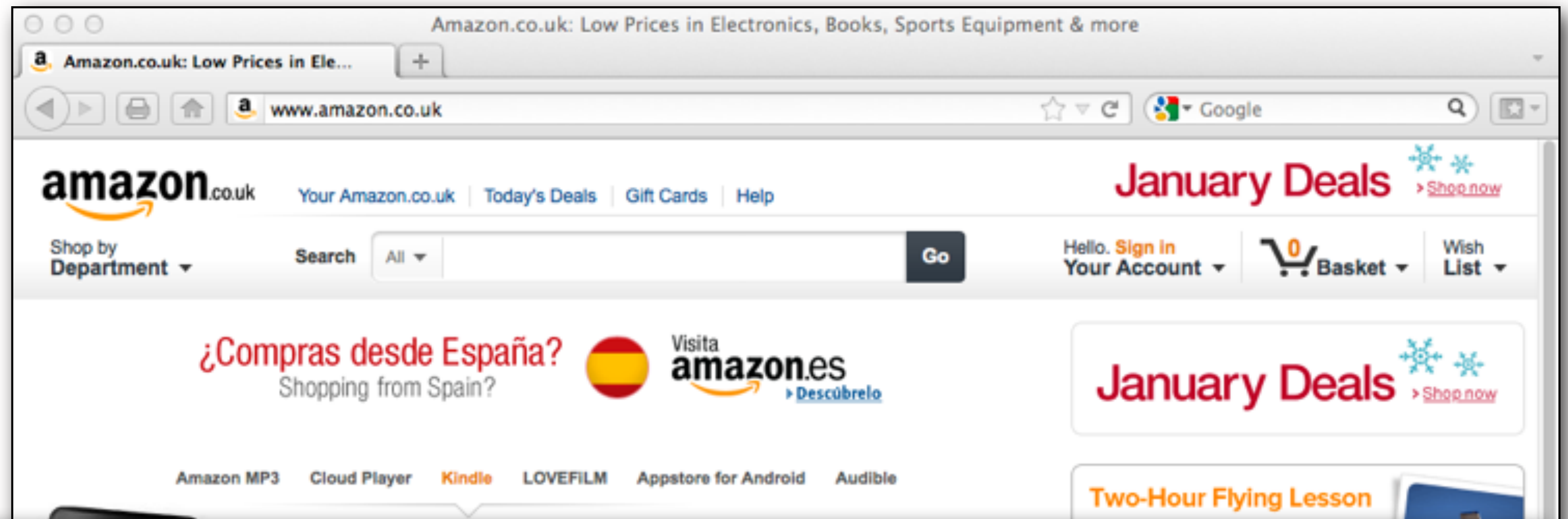
First name

Email

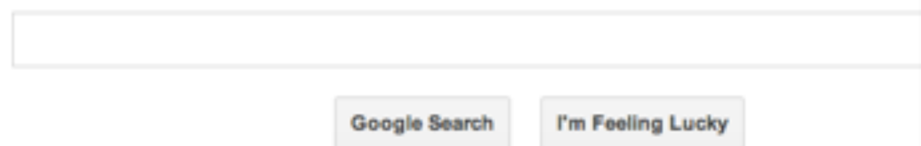
Re-enter emai

New passwor

Birthday



Data is replicated across multiple nodes



Connect with friends and the world around you on Facebook.

See photos and updates from friends in News Feed.

Share what's new in your life on your Timeline.

Sign Up

It's free and alwa

First name

Email

Re-enter emai

New passworc

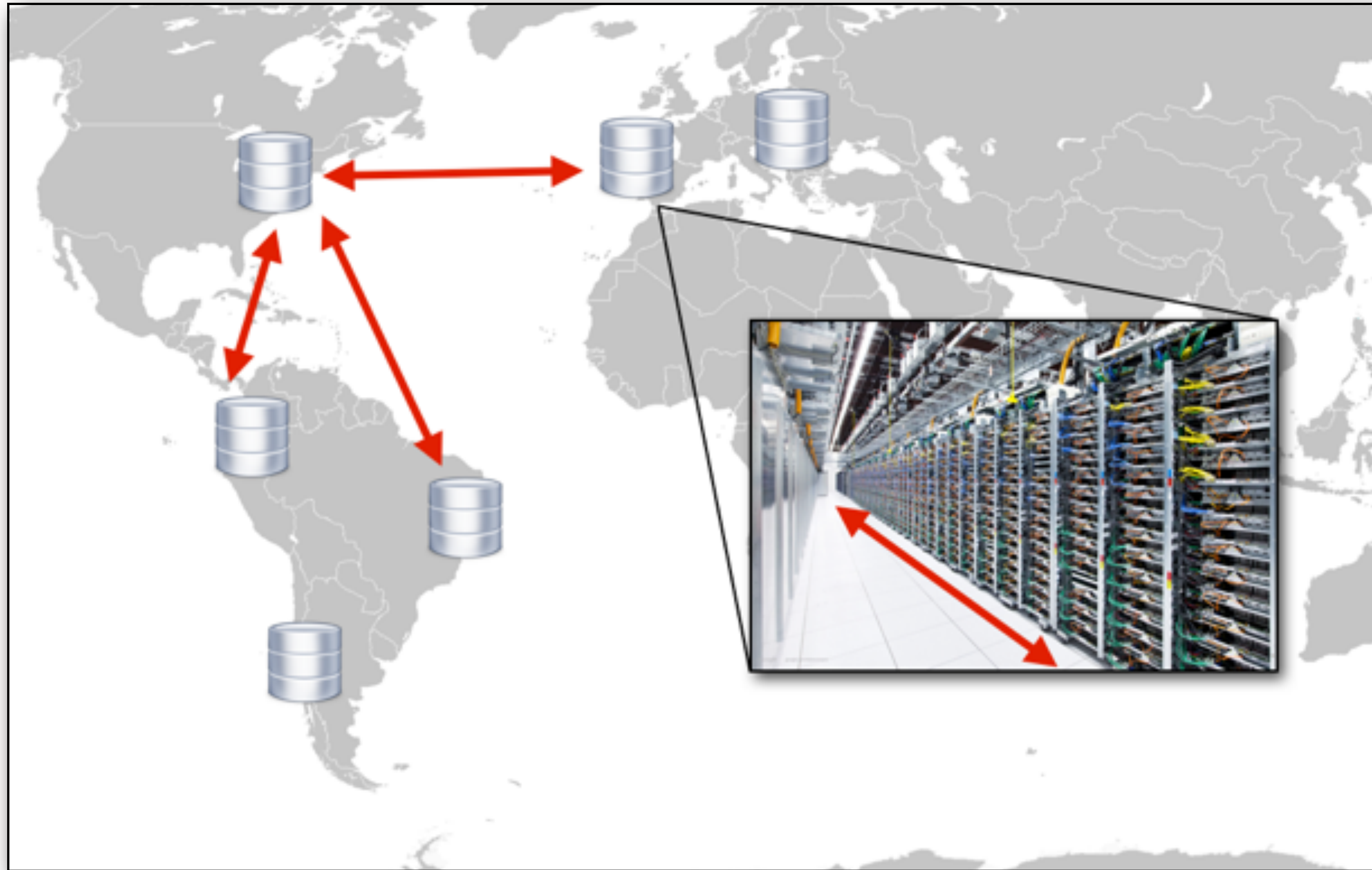
Birthday

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

Scalable Atomic Visibility with RAMP Transactions

Alan Fekete[†], Ali Ghodsi, Joseph M. Hellerstein, Ion Stoica

[†]University of Sydney

Highly Available Transactions: Virtues and Limitations

Peter Bailis, Aaron Davidson, Alan Fekete[†], Ion Stoica

Transactional storage for geo-replicated systems

Yair Sovran[*] Marcos K. Aguilera[†] Jinyang Li[*]

[*] New Silicon Valley

Eventually Consistent Transactions

Sebastian Burckhardt[1], Daan Leijen[1], Manuel Fähndrich[1], and Mooly Sagiv[2]

[1] Microsoft Research
[2] Tel-Aviv University

The database gives weaker guarantees to programmers

Weak Consistency Models

Performance boost

- require less synchronisation between replicas

Weak Consistency Models

Performance boost

- require less synchronisation between replicas

Anomalous behaviour

- executions which are not allowed by a serialisable database
- reasoning techniques for serialisable databases do not apply

Challenges

- Are applications OK with the proposed consistency models?

  Do the non-serialisable behaviours exposed violate correctness?

- can we boost the performances of an application without violating its correctness?

  Are we overpaying in performance penalties?

Challenges

- Are applications OK with the proposed consistency models?

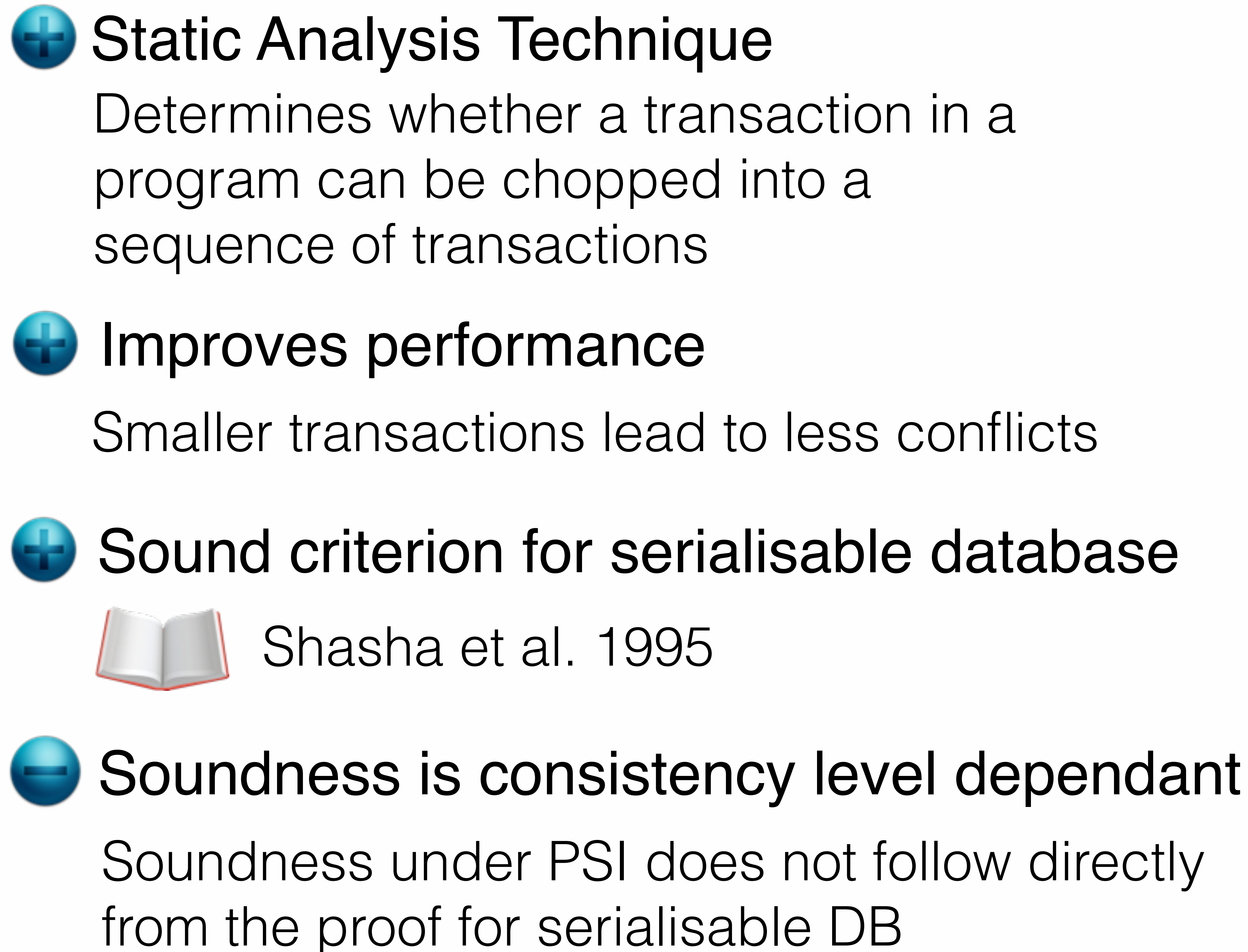
  Do the non-serialisable behaviours exposed violate correctness?

- can we boost the performances of an application without violating its correctness?

  Are we overpaying in performance penalties?

Parallel Snapshot Isolation: Specification and Transaction Chopping

Transaction chopping



PSI
Operational Model

x, y x, y x, y

- Database consists of replicas storing objects
- Every object at every replica
- Clients issue transactions to be executed at replicas

start(t_1)
x.write(*post*)
y.write(*comment*)
commit(t_1)

start(t_2)
x.write(*other post*)
abort(t_2)

- Write write conflict detection (concurrent transactions do not write to one same object)

start(t_1)
x.write(*post*)
y.write(*comment*)
commit(t_1)

start(t_2)
x.write(*other post*)
abort(t_2)

start(t_3)
y.read(*empty*)
commit(t_3)

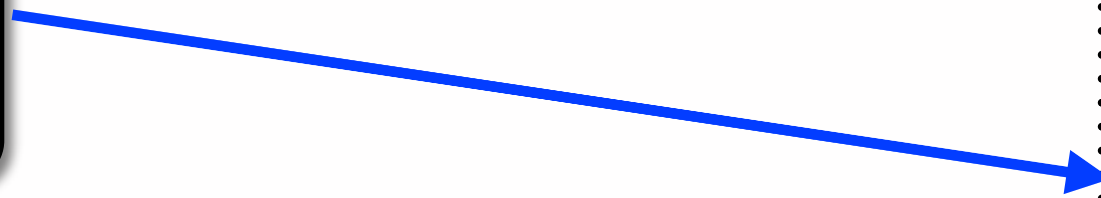
deliver
t_1: x.write(post);
y.write(comment)

start(t_4)
y.read(*comment*)
commit(t_4)

Upon commit: send all tx updates to other replicas

start(t_1)
x.write(*post*)
commit(t_1)

start(t_2)
y.write(*comment*)
commit(t_2)



start(t_3)
y.read(*comment*)
x.read( )
commit(t_3)

Message delivery:
causality is preserved

start(t_1)
x.write(*post*)
commit(t_1)

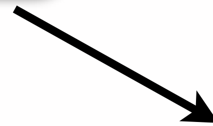
start(t_2)
y.write(*comment*)
commit(t_2)

start(t_3)
y.read(*comment*)
x.read(*post*)
commit(t_3)

Message delivery:
causality is preserved

x.write(1)

y.write(1)



Long fork

Disallowed by classical snapshot isolation

x.write(1)

y.write(1)

x.read(1)

y.read(0)

x written before y

Long fork

Disallowed by classical snapshot isolation

x.write(1)

y.write(1)

x.read(1)
y.read(0)

y.read(1)
x.read(0)

Long fork

Disallowed by classical snapshot isolation

x written before y

y written before x

Problems of the Operational Model

- Implementation Dependent
- Difficult to reason about

An alternative: Abstract Model

- Exploits the relationships between events

From operational model to abstract executions

start(t_1)
x.write(*post*)
y.write(*comment*)
commit(t_1)

start(t_2)
x.write(*other post*)
abort(t_2)

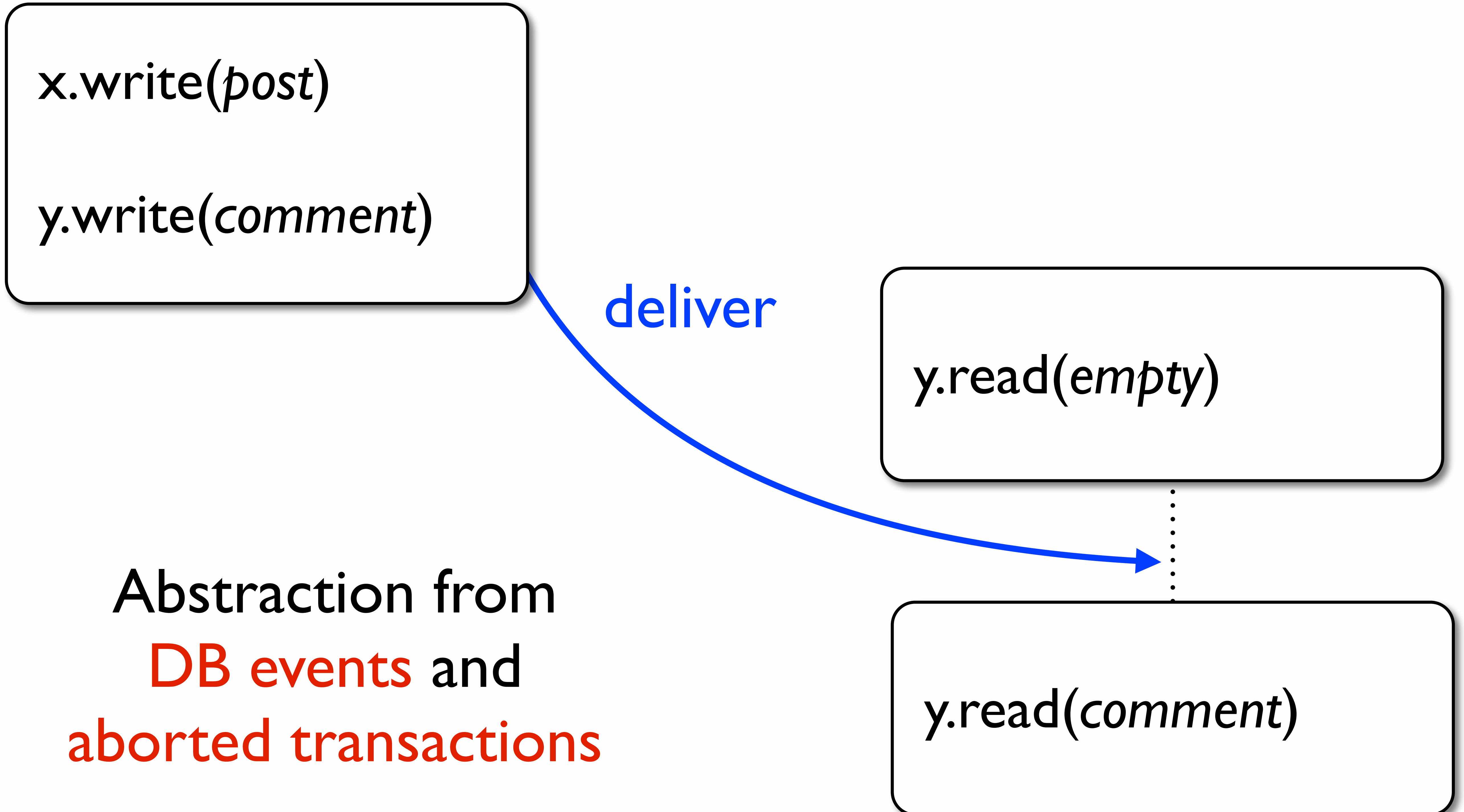
deliver

start(t_3)
y.read(*empty*)
commit(t_3)

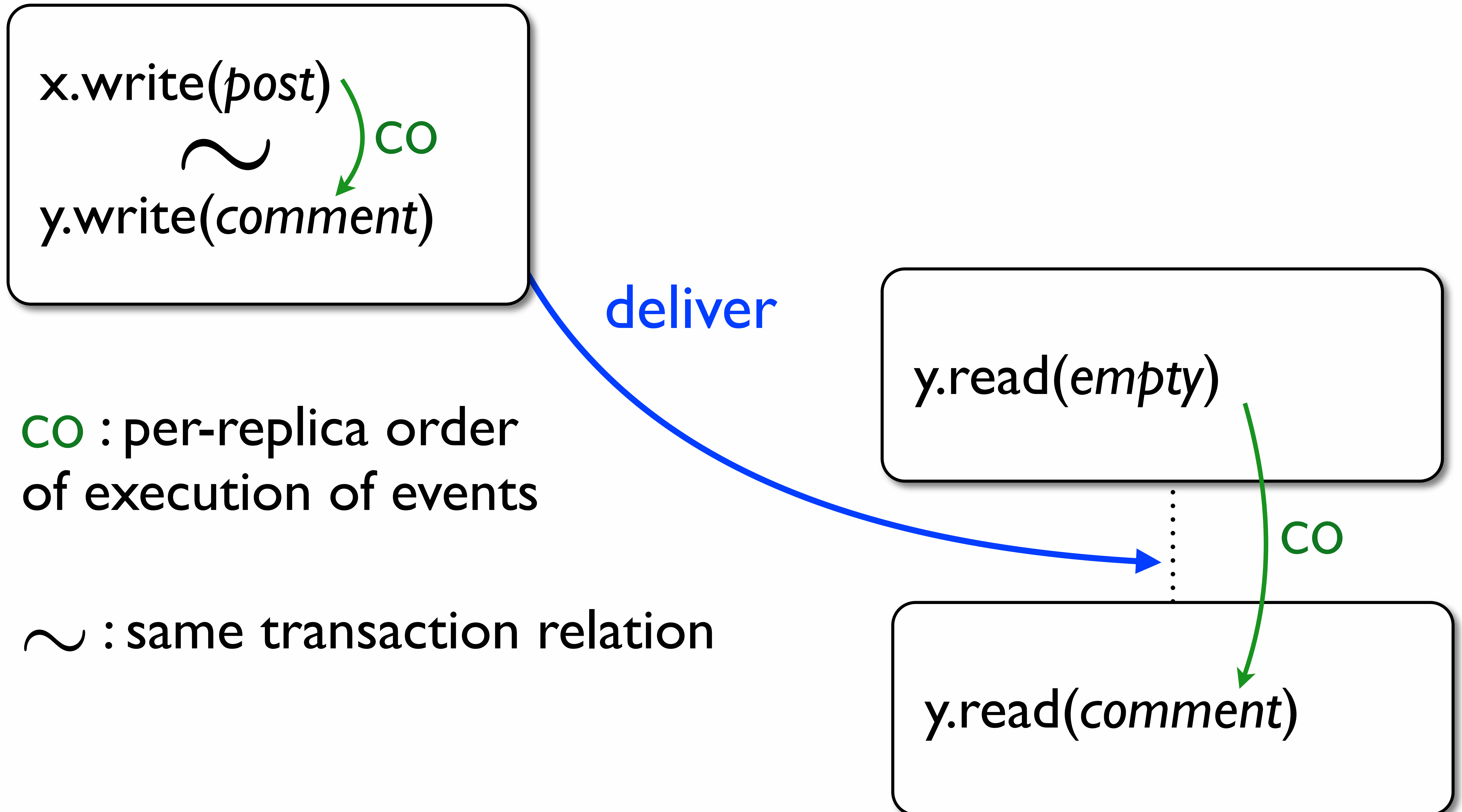
start(t_4)
y.read(*comment*)
commit(t_4)

Abstraction from DB events and aborted transactions

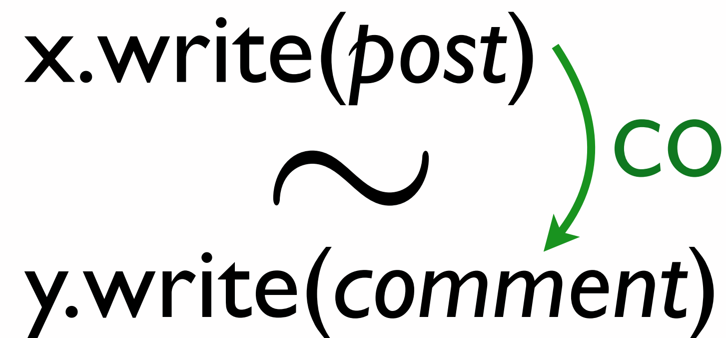
From operational model to abstract executions



From operational model to abstract executions



From operational model to abstract executions



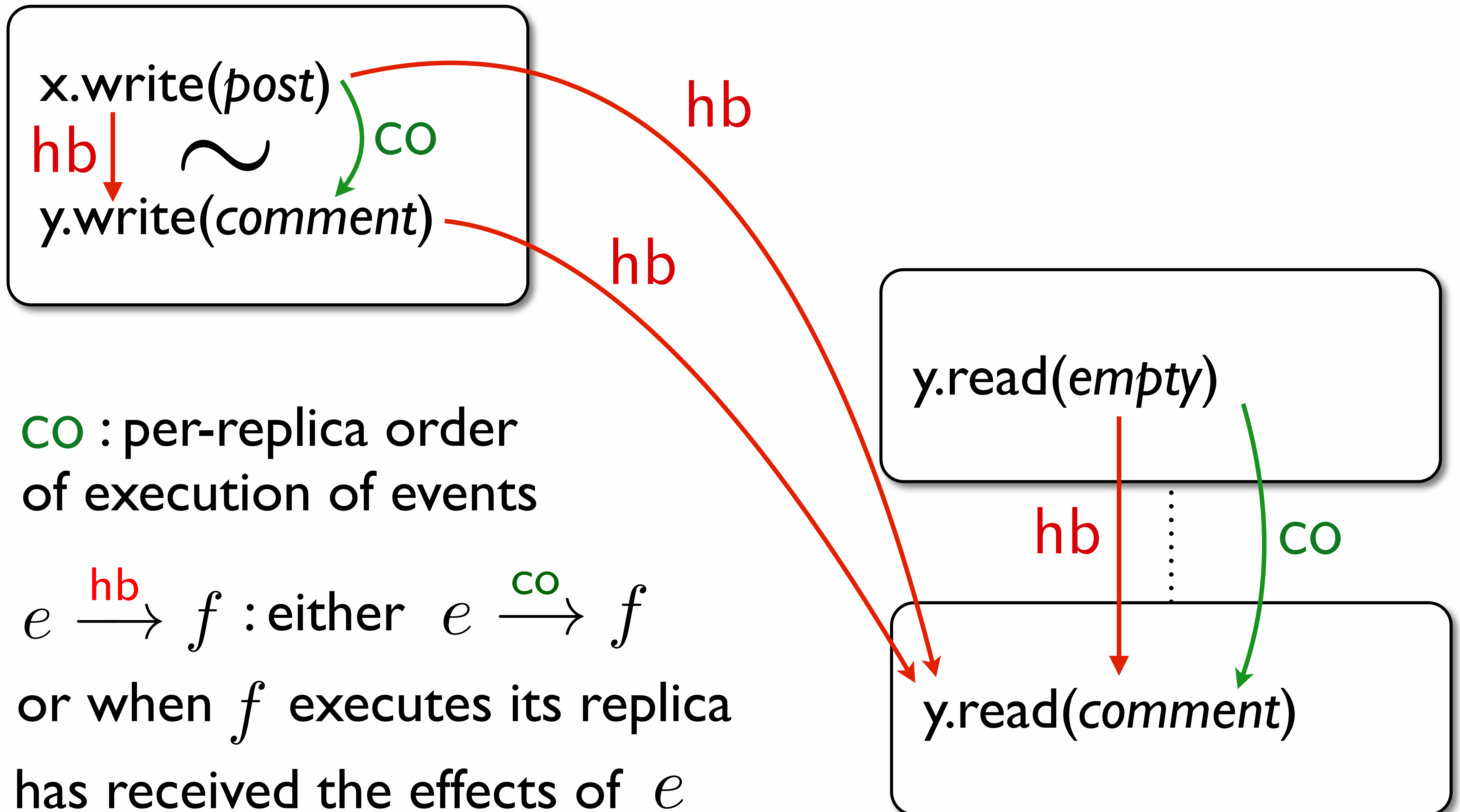
co

co

co : per-replica order of execution of events

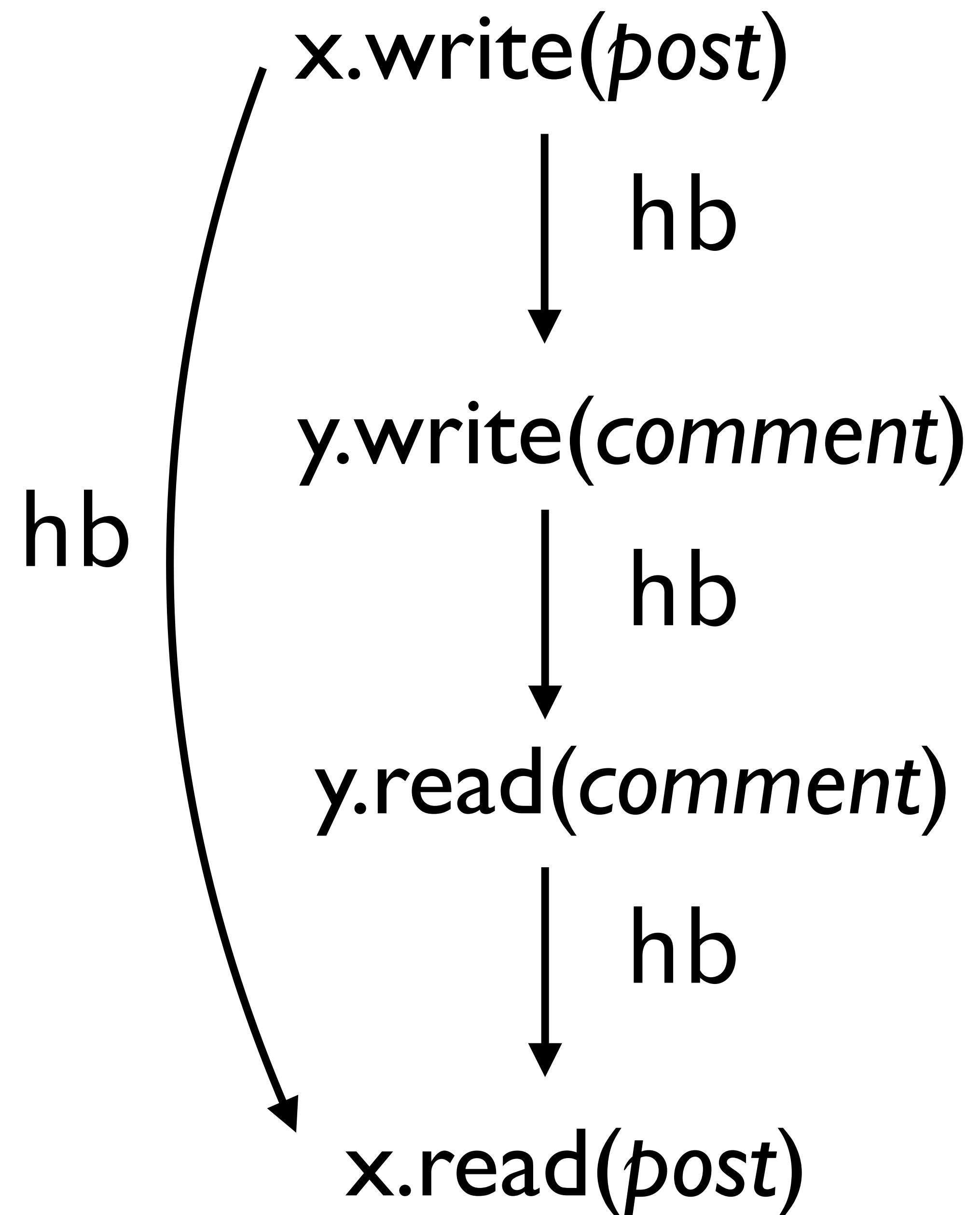
$e \xrightarrow{\text{hb}} f$: either $e \xrightarrow{\text{co}} f$ or when f executes its replica has received the effects of e

From operational model to abstract executions



Abstract model of PSI: obtained by constraining hb

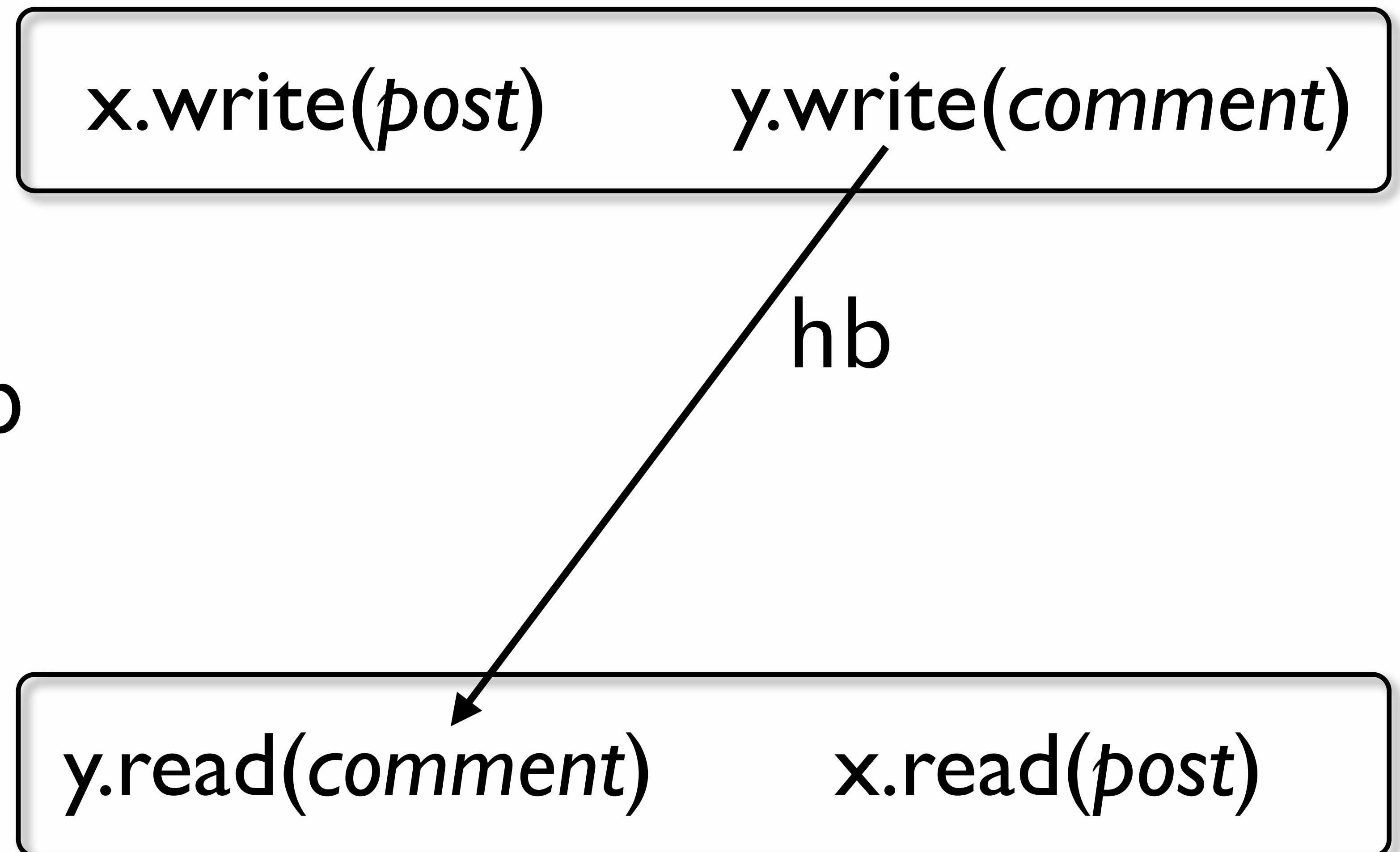
- $\mathsf{co} \subseteq \mathsf{hb}$
- $\mathsf{hb}^+ \subseteq \mathsf{hb}$



HB is transitive: causality is preserved

Abstract model of PSI: obtained by constraining hb

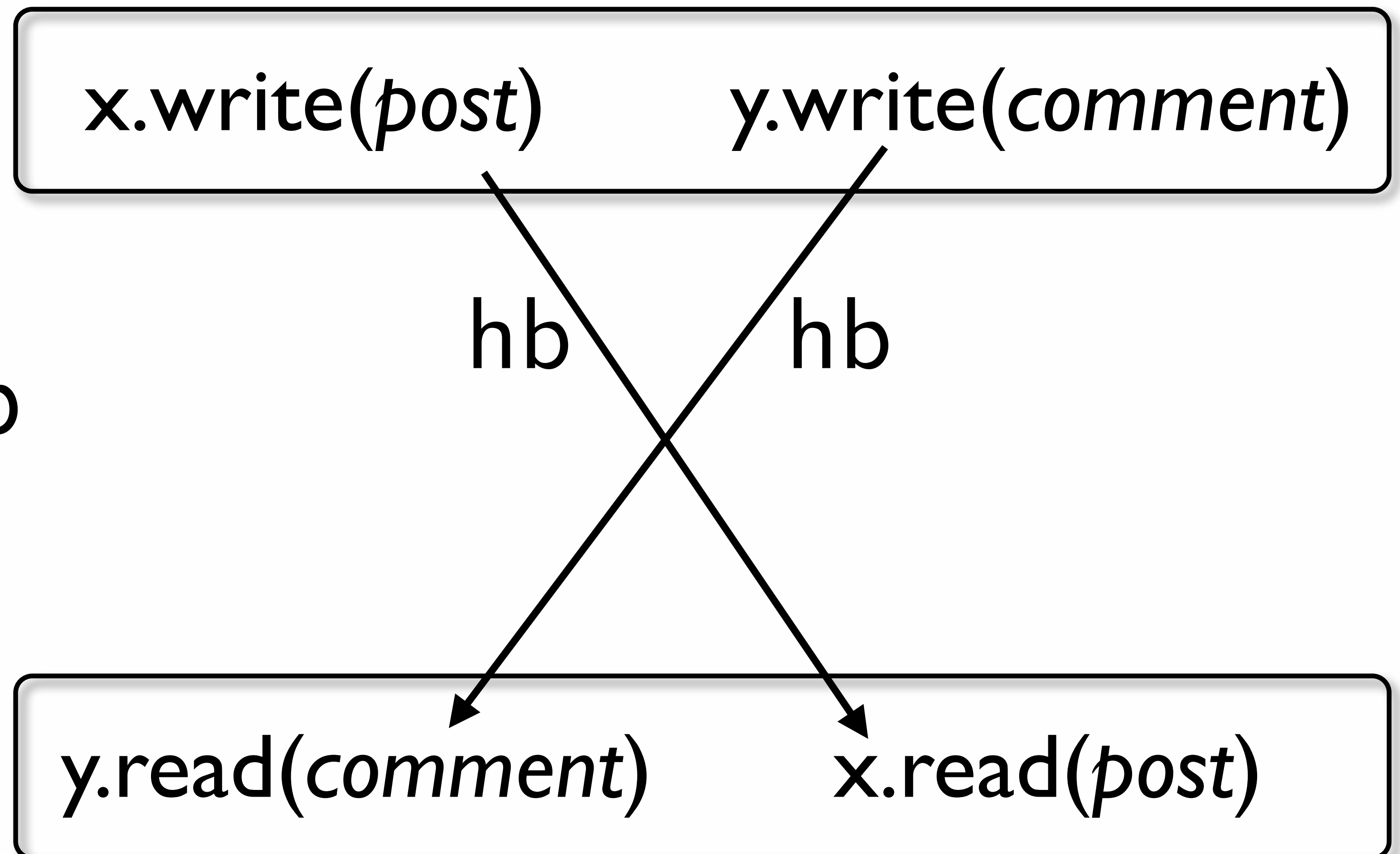
- $\mathsf{co} \subseteq \mathsf{hb}$
- $\mathsf{hb}^+ \subseteq \mathsf{hb}$
- $\sim; (\mathsf{hb} \setminus \sim); \sim \subseteq \mathsf{hb}$



Atomicity: either none or all the events of a transaction are observed by another one

Abstract model of PSI: obtained by constraining hb

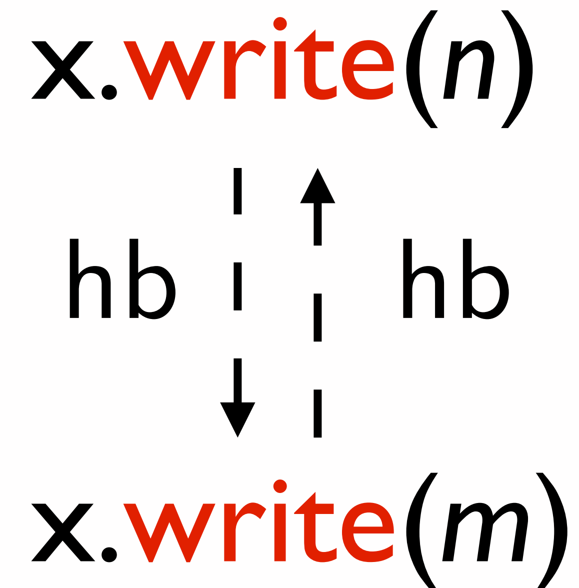
- $\mathsf{co} \subseteq \mathsf{hb}$
- $\mathsf{hb}^+ \subseteq \mathsf{hb}$
- $\sim; (\mathsf{hb} \setminus \sim); \sim \subseteq \mathsf{hb}$



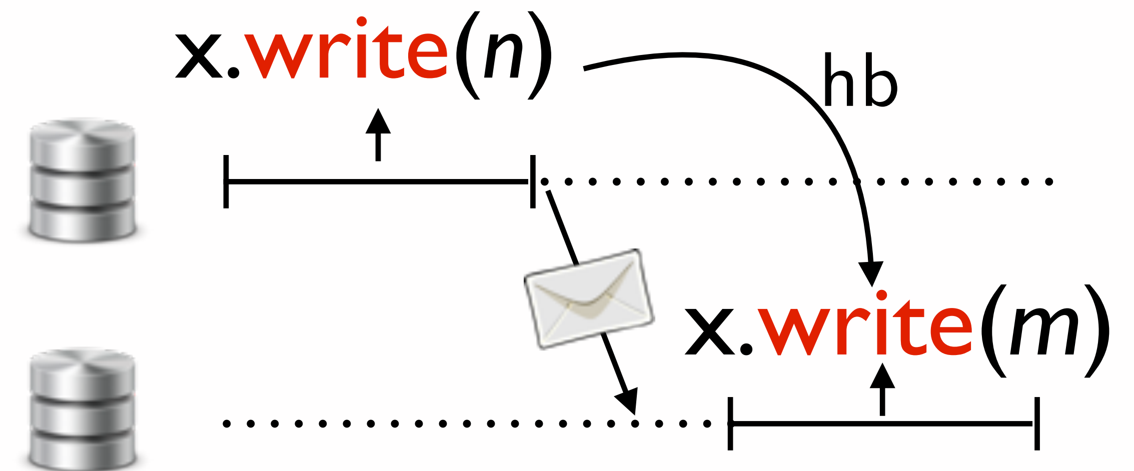
Atomicity: either none or all the events of a transaction are observed by another one

Abstract model of PSI: obtained by constraining hb

- $\mathsf{co} \subseteq \mathsf{hb}$
- $\mathsf{hb}^+ \subseteq \mathsf{hb}$
- $\sim; (\mathsf{hb} \setminus \sim); \sim \subseteq \mathsf{hb}$
- $\forall x. \forall e, f \in \mathsf{Writes}_x.$
  $e = f \vee e \xrightarrow{\mathsf{hb}} f \vee f \xrightarrow{\mathsf{hb}} e$



Writes on the same object are related by hb
Write write conflict detection:



Abstract model of PSI: obtained by constraining hb

- $\mathsf{co} \subseteq \mathsf{hb}$
- $\mathsf{hb}^+ \subseteq \mathsf{hb}$
- $\sim; (\mathsf{hb} \setminus \sim); \sim \subseteq \mathsf{hb}$
- $\forall x. \forall e, f \in \mathsf{Writes}_x.$
  $e = f \vee e \xrightarrow{\mathsf{hb}} f \vee f \xrightarrow{\mathsf{hb}} e$
- $\mathsf{op}(e) = \mathsf{x.read}(n) \implies \mathsf{op}(f) = \mathsf{x.write}(n)$
  where f is the last write on xhappening before e

x.write(1)

hb

x.write(2)

hb

x.write(3)

hb

x.read(3)

read operations fetch their value from the most recent write

Correctness of the specification

Soundness:

every concrete execution is encoded in an abstract one that satisfies the given properties

Correctness of the specification

Soundness:

every concrete execution is encoded in an abstract one that satisfies the given properties

Completeness:

any abstract execution that satisfies the given properties can be obtained from the encoding of a concrete PSI one

Chopping: an example

```
Transaction transfer (int n) {
 TMP1 := read(acct1);
 TMP2 := read(acct2);
 write(acct2, TMP1 + TMP2);
 tryCommit;
}
```

```
Transaction lookup1() {
 TMP := read(acct1);
 tryCommit;
}
```

transfer can be **chopped** in two transactions without introducing new behaviour

```
Transaction withdraw (int n) {
 TMP := read(acct1);
 write(acct1, TMP - n);
 tryCommit;
}
```

```
Transaction deposit (int n) {
 TMP := read(acct2);
 write(acct1, TMP + n);
 tryCommit;
}
```

```
Chain transfer' (int n) { withdraw(n); deposit(n); }
```

Chopping: an example

Chopping is not always possible:

```
Transaction transfer (int n) {
 TMP1 := read(acct1);
 TMP2 := read(acct2);
 write(acct2, TMP1 + TMP2);
 tryCommit;
}
```

```
Transaction Mlookup() {
 TMP1 := read(acct1);
 TMP2 := read(acct2);
 tryCommit;
}
```

```
Chain transfer' (int n) { withdraw(n); deposit(n); }
```

Chopping: an example

Chopping is not always possible:

```
Transaction transfer (int n) {
 TMP1 := read(acct1);
 TMP2 := read(acct2);
 write(acct2, TMP1 + TMP2);
 tryCommit;
}
```

```
Transaction Mlookup() {
 TMP1 := read(acct1);
 TMP2 := read(acct2);
 tryCommit;
}
```

```
Chain transfer' (int n) { withdraw(n); deposit(n); }
```

`Mlookup` can be used to observe an intermediate state of the database

$\texttt{acct1} = 50$, $\texttt{acct2} = 0$ — `transfer(20);` → $\texttt{acct1} = 30$, $\texttt{acct2} = 20$

Chopping: an example

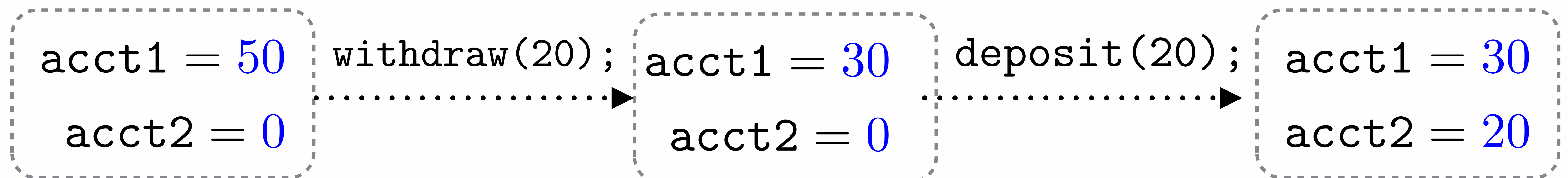
Chopping is not always possible:

```
Transaction transfer (int n) {
 TMP1 := read(acct1);
 TMP2 := read(acct2);
 write(acct2, TMP1 + TMP2);
 tryCommit;
}
```

```
Transaction Mlookup() {
 TMP1 := read(acct1);
 TMP2 := read(acct2);
 tryCommit;
}
```

```
Chain transfer' (int n) { withdraw(n); deposit(n); }
```

`Mlookup` can be used to observe an intermediate state of the database

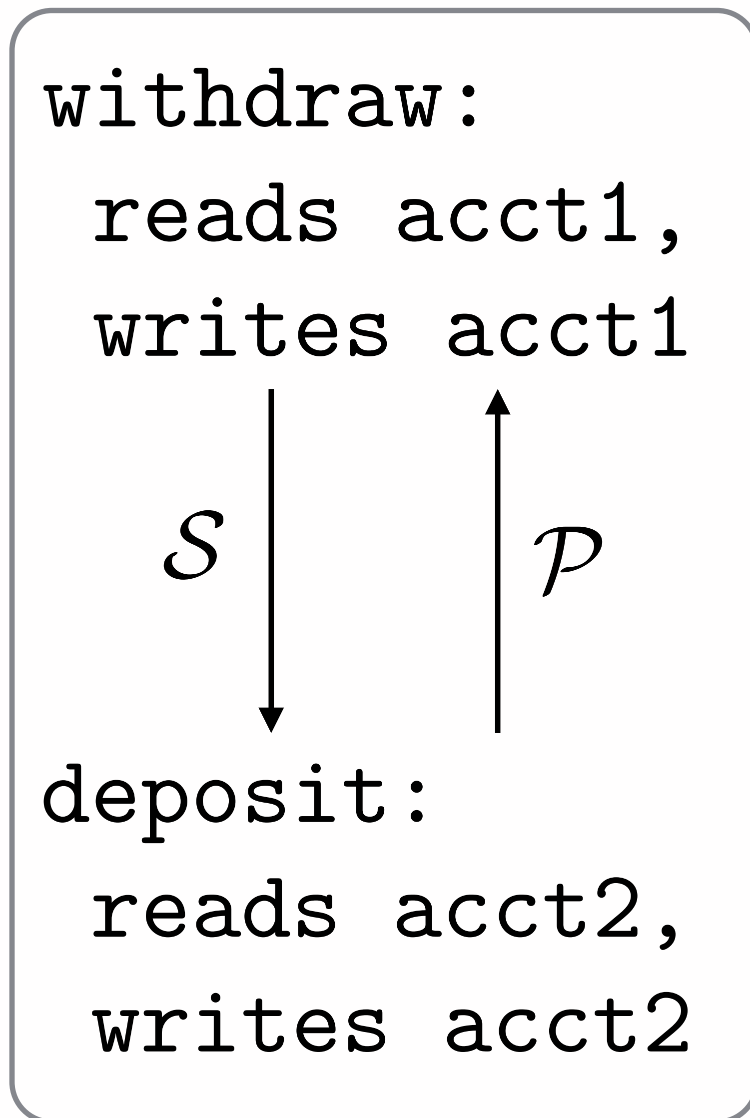


Chopping graphs

```
transfer':

withdraw:
 reads acct1,
 writes acct1

deposit:
 reads acct2,
 writes acct2

Mlookup:
 reads acct1,
 reads acct2
```

Chopping graphs

`transfer':`

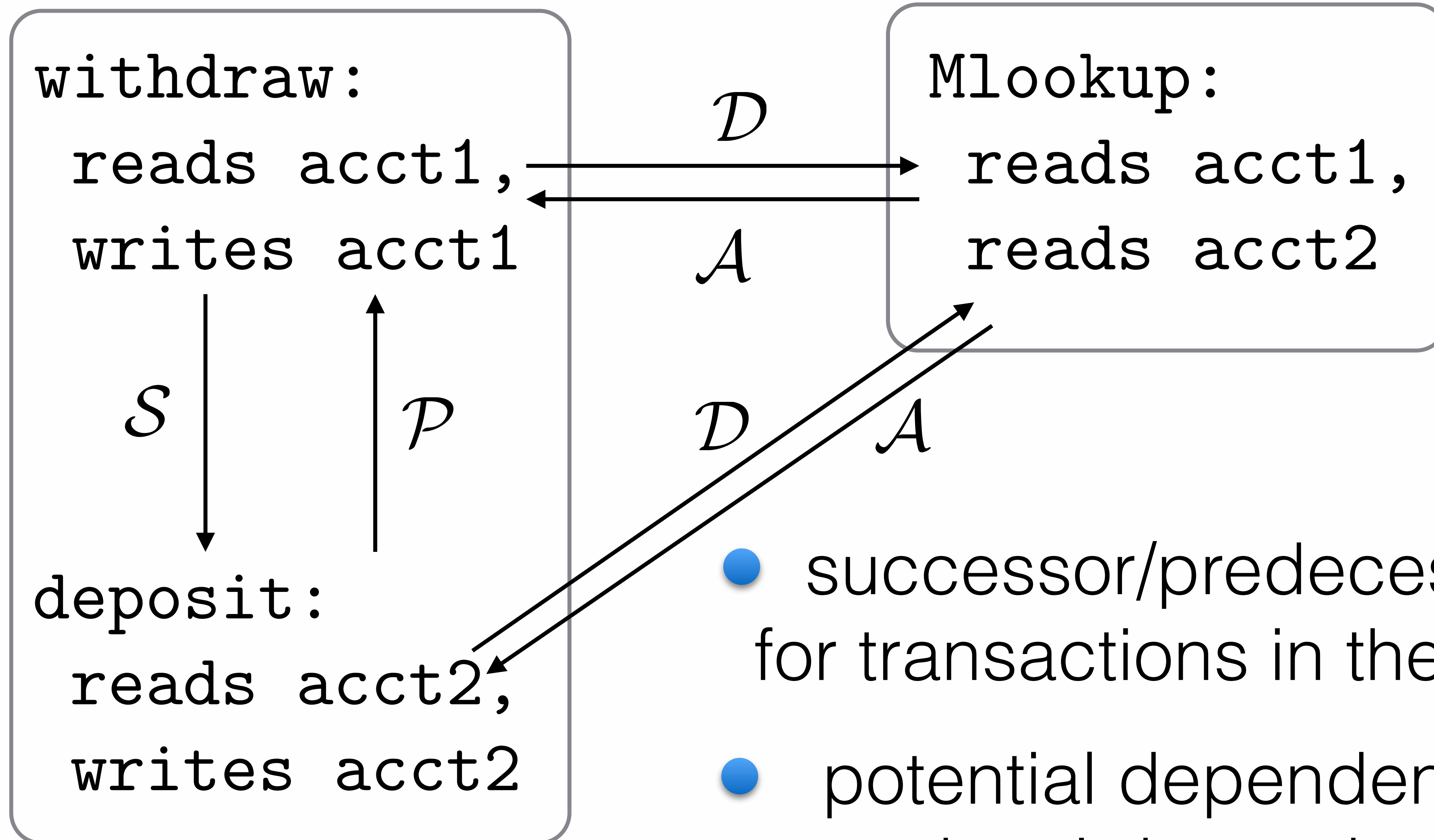



```
Mlookup:
  reads acct1,
  reads acct2
```

- successor/predecessor edges for transactions in the same chain

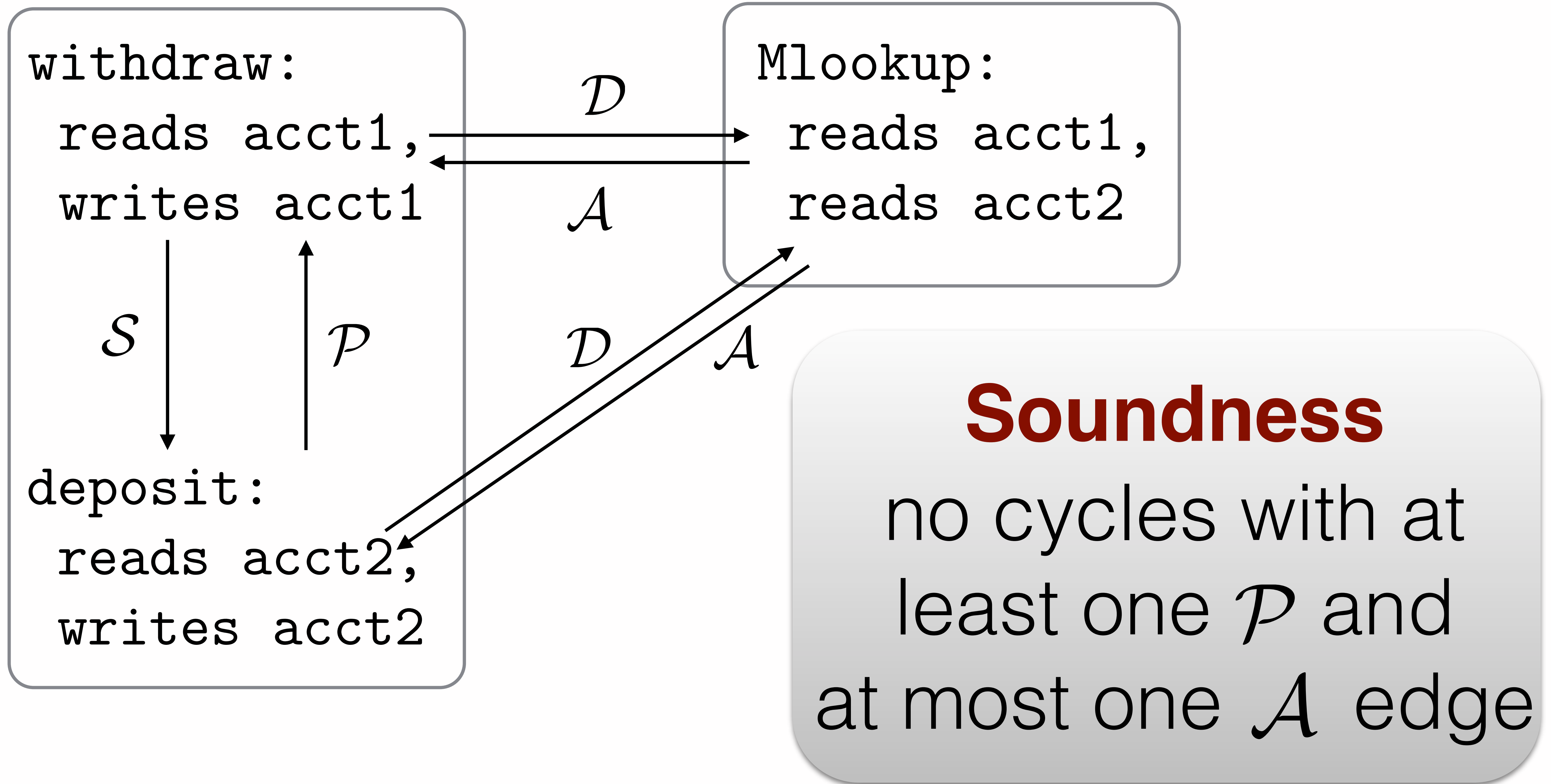
Chopping graphs

`transfer':`



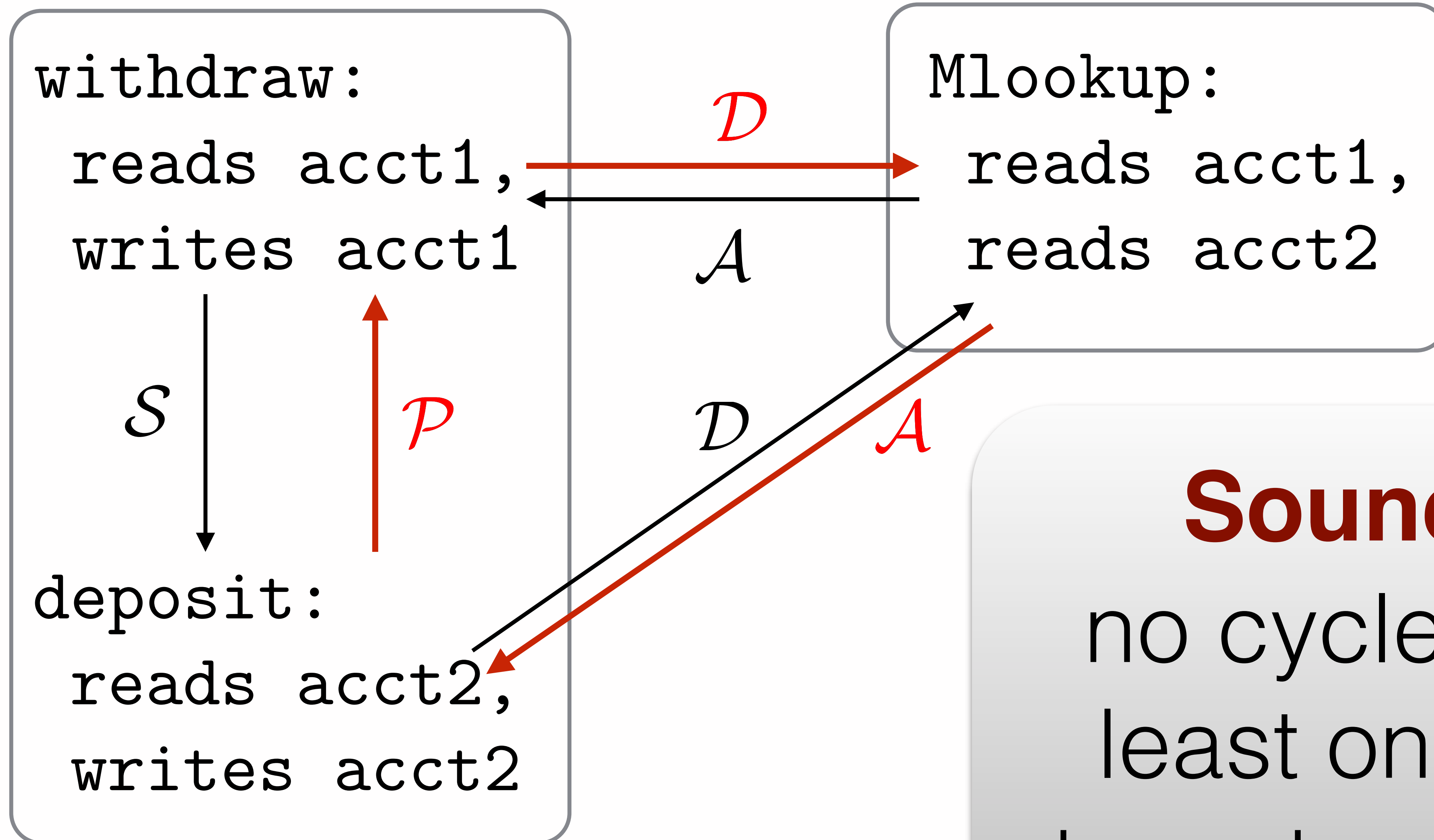
- successor/predecessor edges for transactions in the same chain
- potential dependencies and anti dependencies between transactions in different chains

Chopping criterion for PSI



Proof: relies heavily on the specification

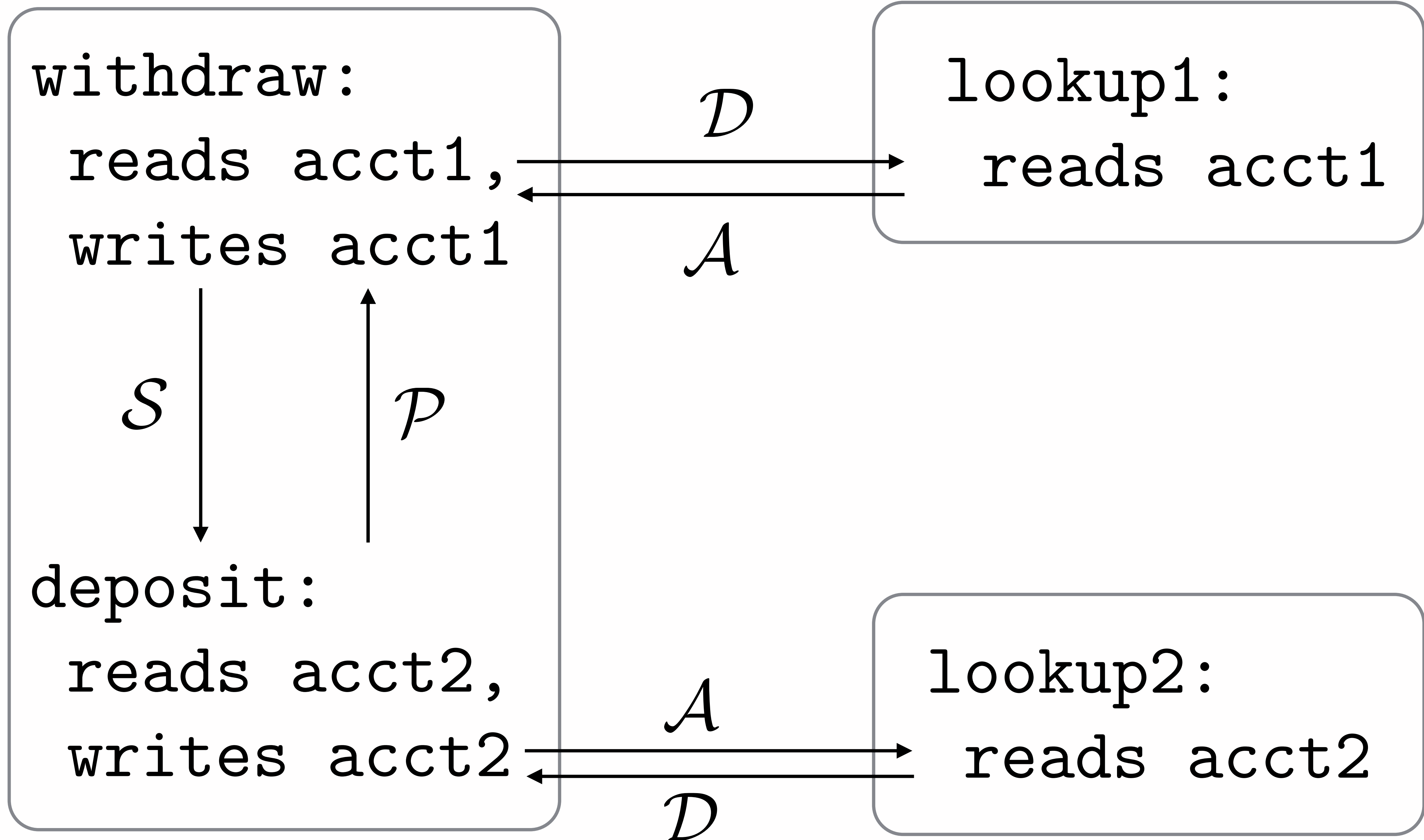
Chopping criterion for PSI



Soundness

no cycles with at least one $\mathcal{P}$ and at most one $\mathcal{A}$ edge

A positive example



Chopping: Serialisability VS. PSI

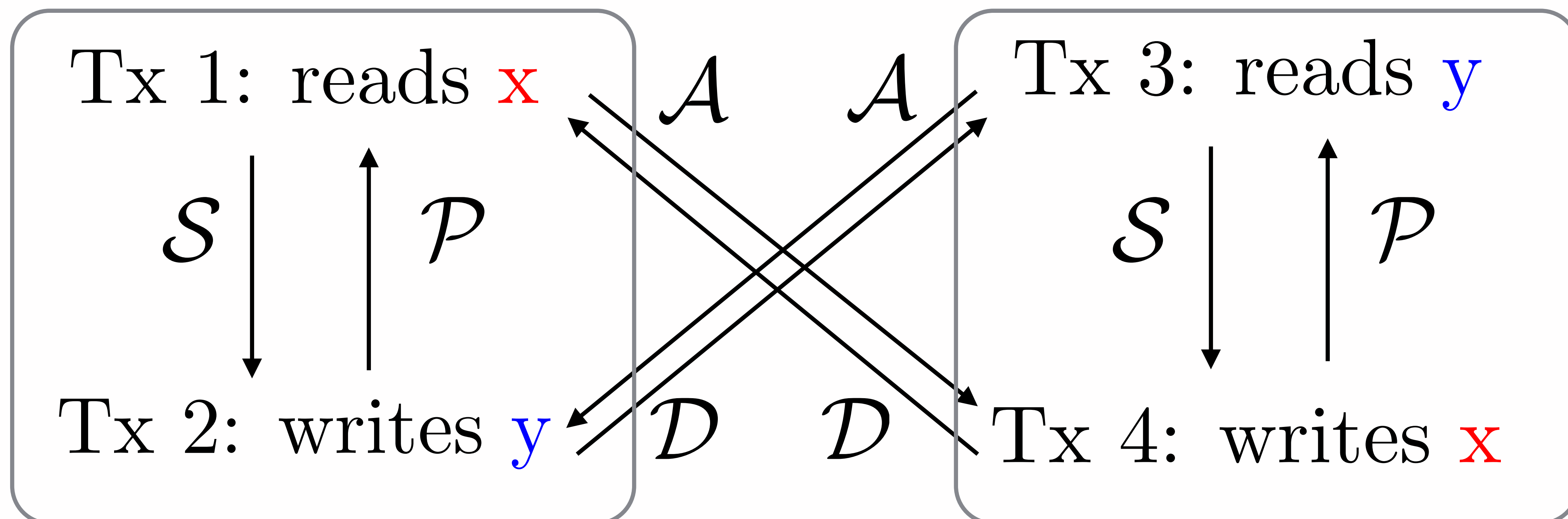
- The existing criterion for serialisability can be applied in PSI databases
  proof: show that the criterion for serialisability implies the one for PSI

- But the converse is not true

Chopping: Serialisability VS. PSI

- The existing criterion for serialisability can be applied in PSI databases
  proof: show that the criterion for serialisability implies the one for PSI

- But the converse is not true



What to do next
(and what we have already done)

- Abstract Specification of Different Consistency Models (CONCUR 2015)

- Robustness (Giovanni Bernardi, work in progress)
  ensure that the behaviour of a program is preserved when the consistency model is weakened

- Chopping for other consistency models
  We already have a proposal for SI

Thank you!