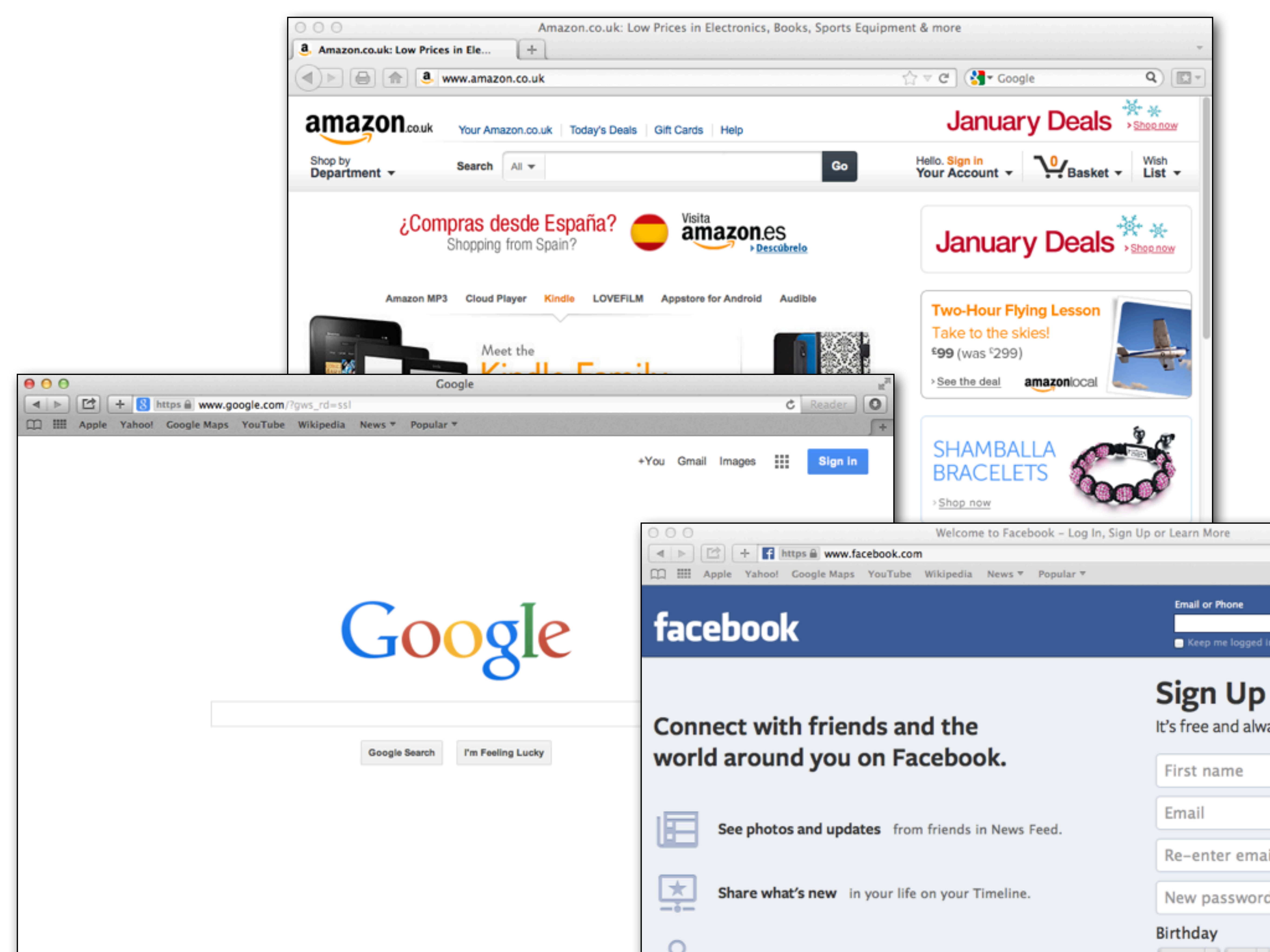
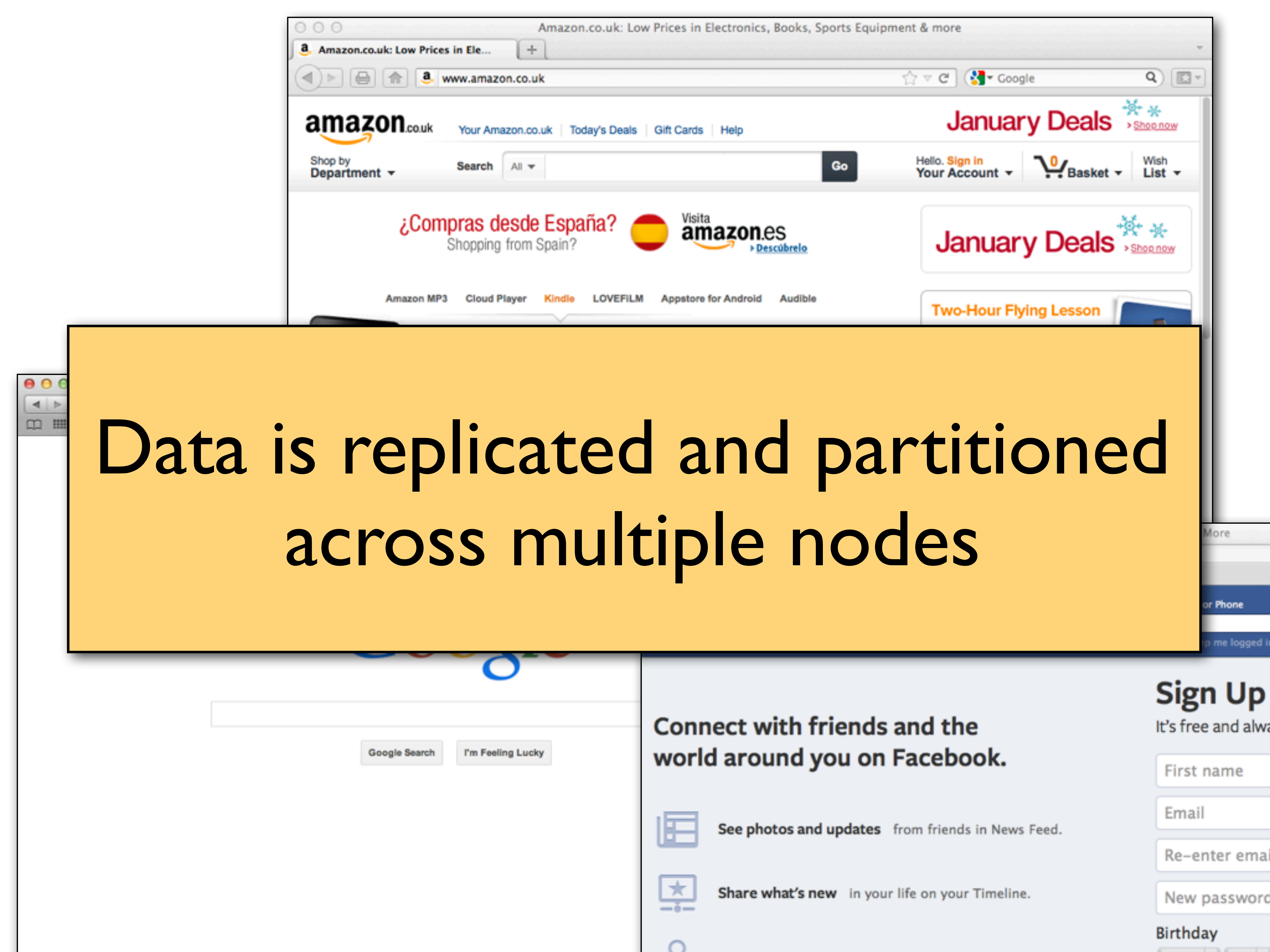
A rigorous approach to consistency in cloud databases

Alexey Gotsman

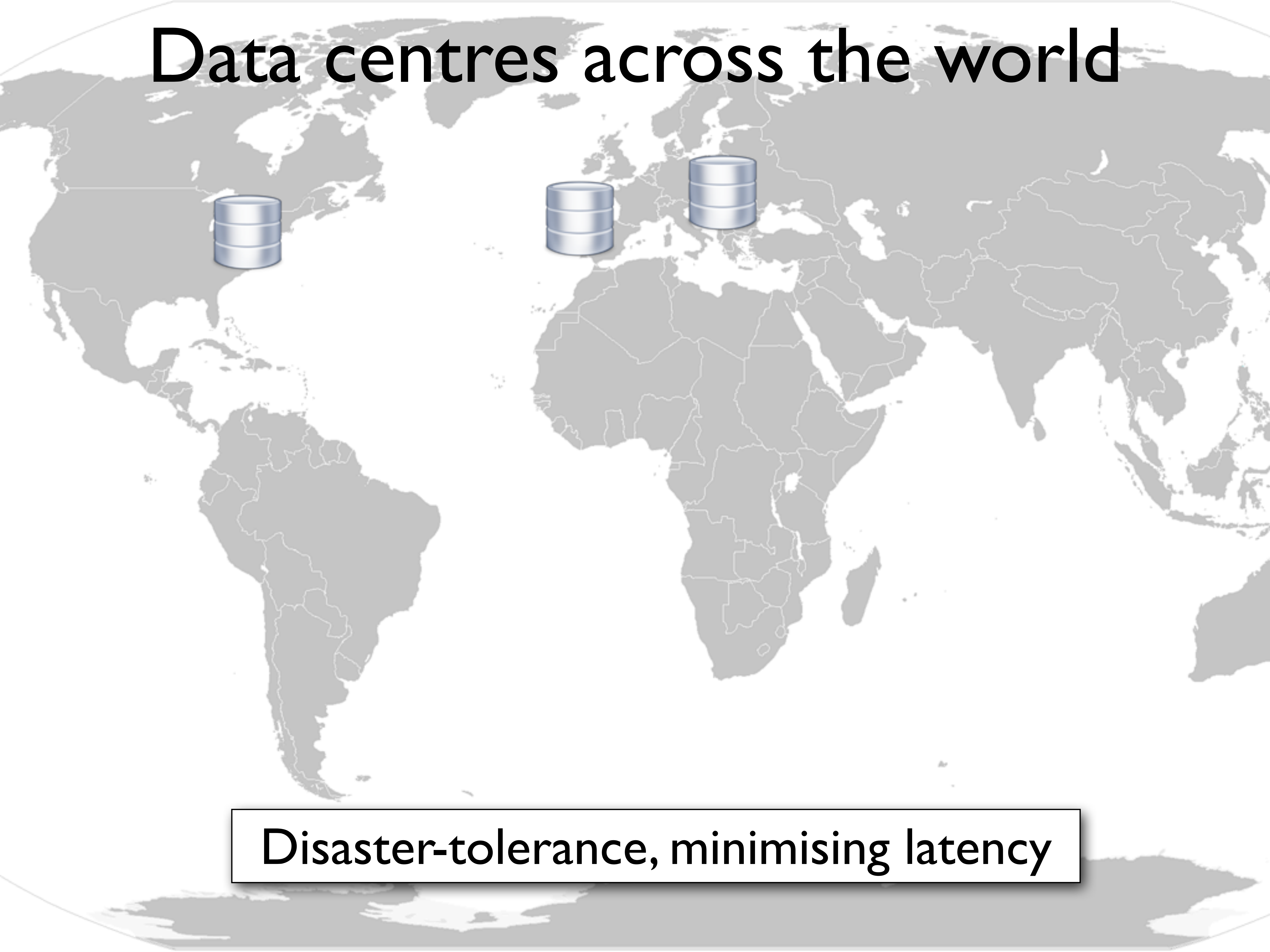
IMDEA Software Institute, Madrid, Spain


Amazon.co.uk: Low Prices in Electronics, Books, Sports Equipment & more
www.amazon.co.uk
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
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
Two-Hour Flying Lesson
Take to the skies!
£99 (was £299)
See the deal
amazonlocal
SHAMBALLA BRACELETS
Shop now
Google
www.google.com/?gws_rd=ssl
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
www.facebook.com
facebook
Email or Phone
Sign Up
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.
First name
Email
New password
Birthday



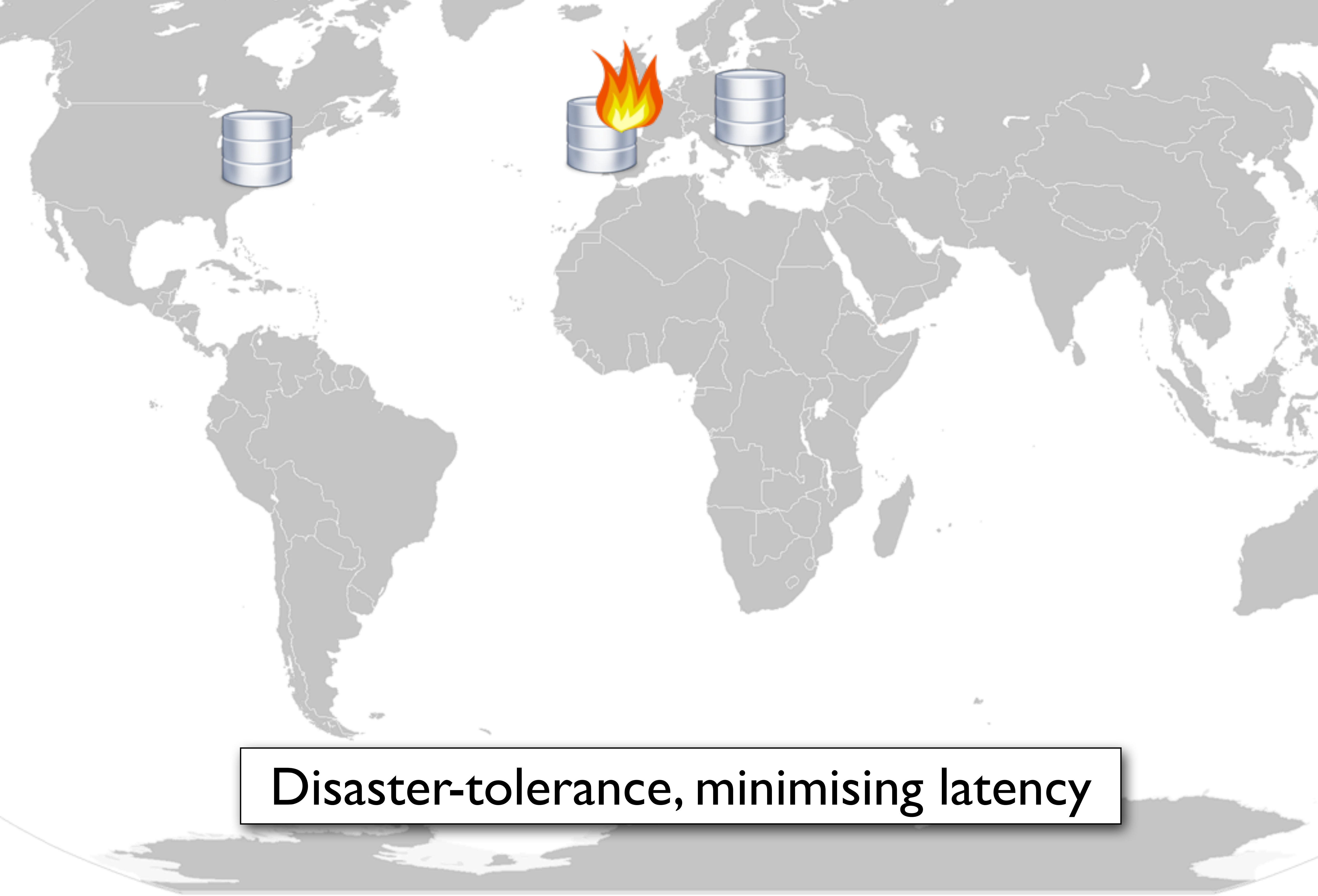
Data is replicated and partitioned across multiple nodes

Data centres across the world



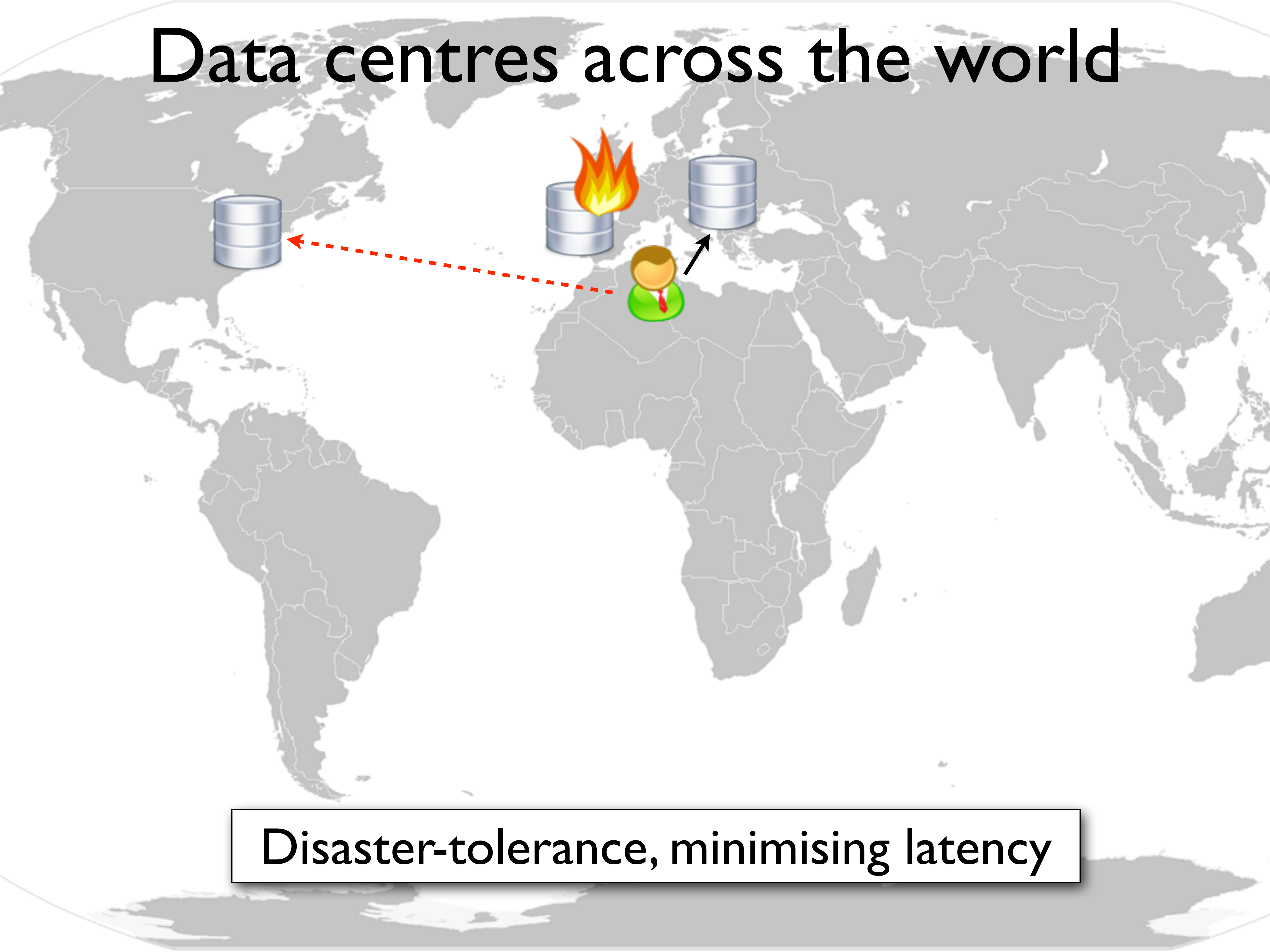
Disaster-tolerance, minimising latency

Data centres across the world



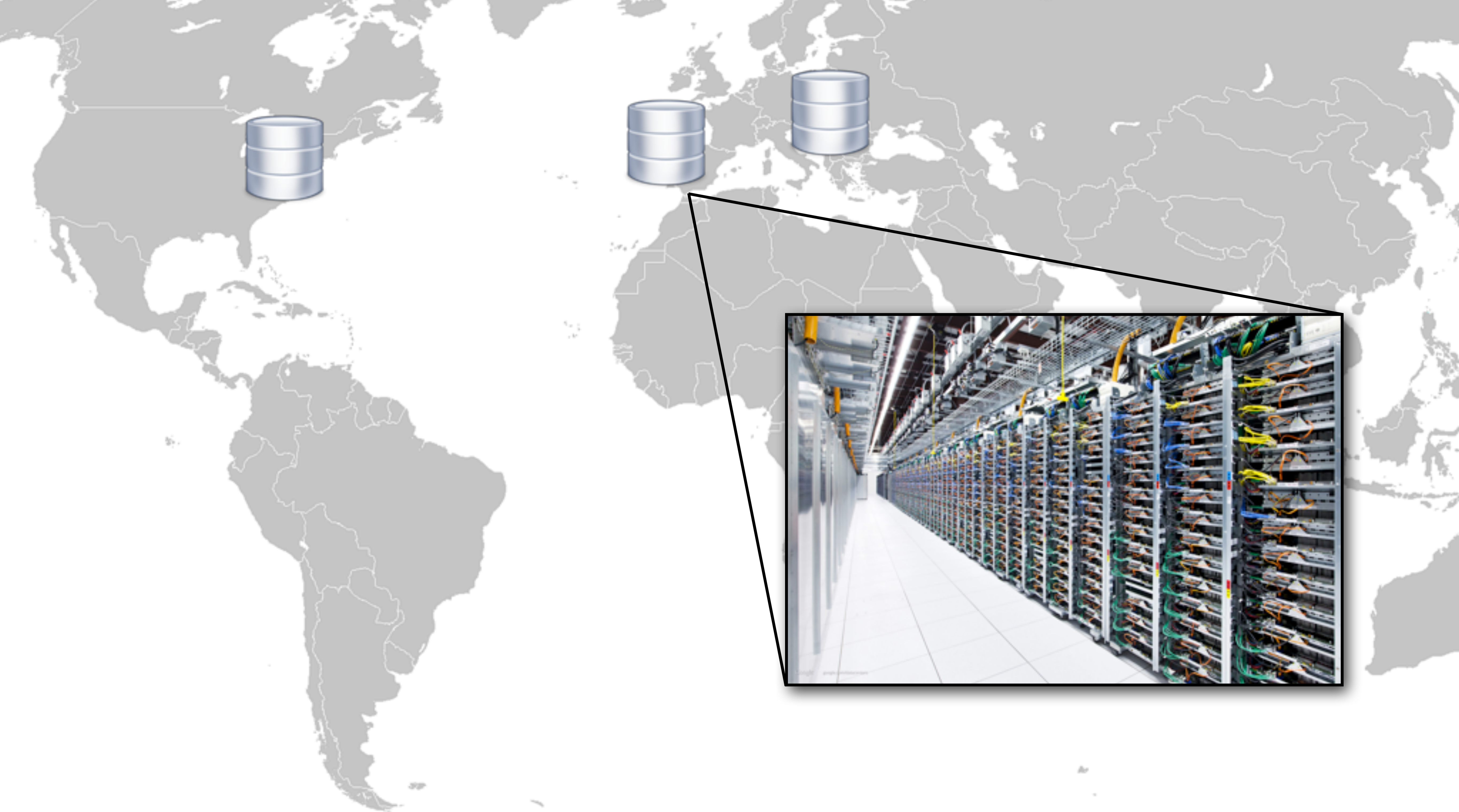
Disaster-tolerance, minimising latency

Data centres across the world



Disaster-tolerance, minimising latency

With thousands of machines inside



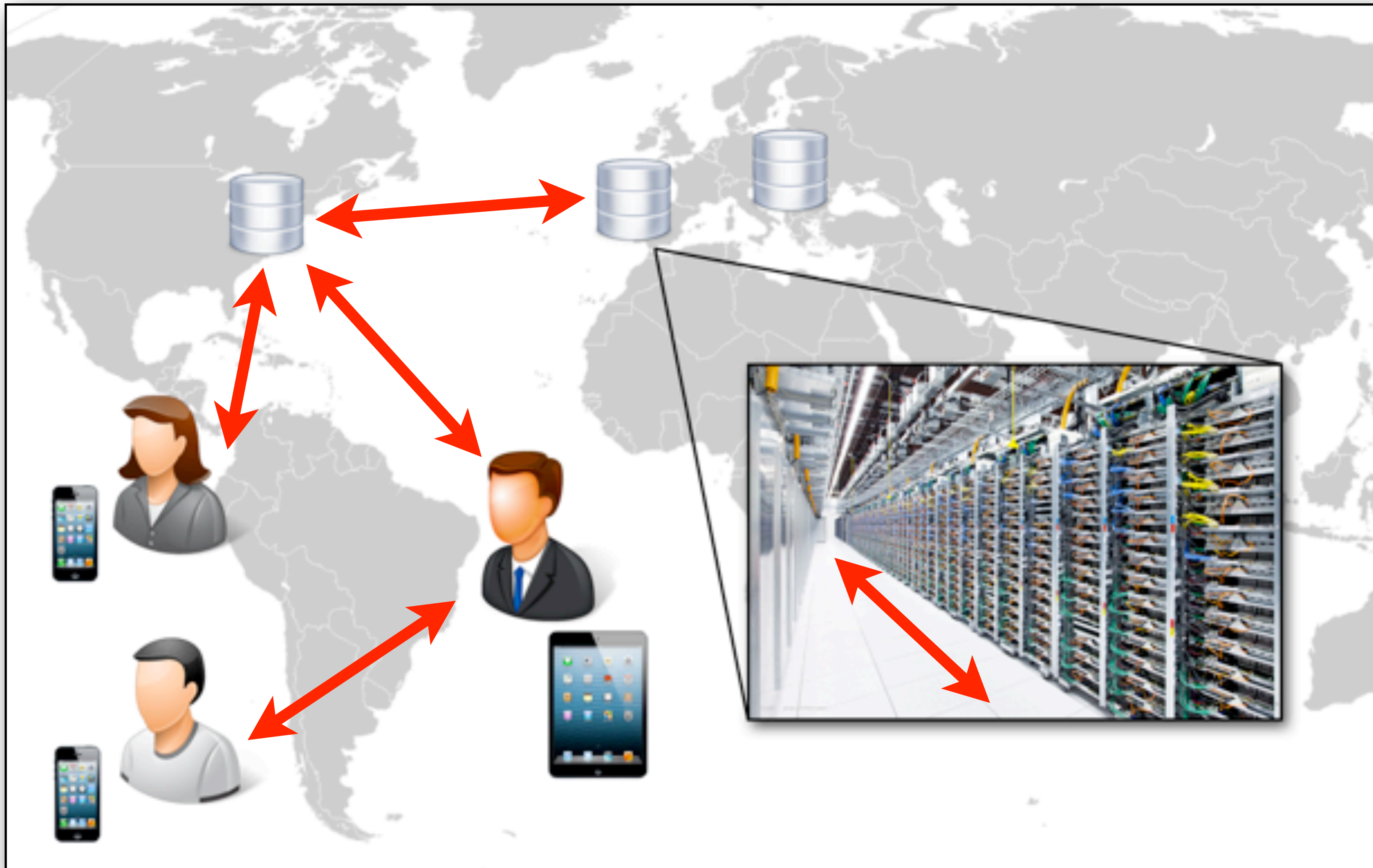
Load-balancing, fault-tolerance

Replicas on mobile devices

Offline use

$\approx$

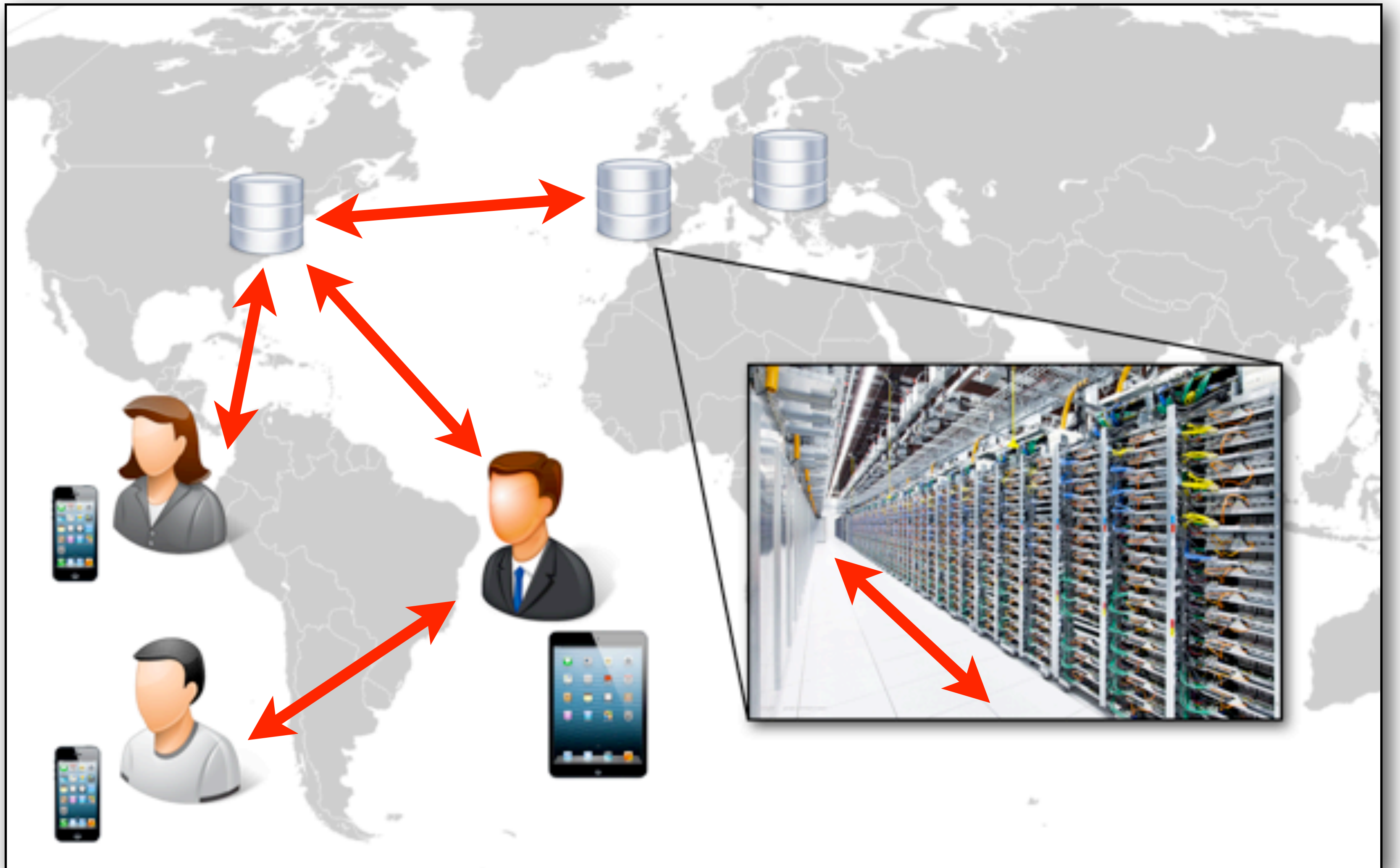
- Strong consistency model: the system behaves as if it processes requests serially on a centralised database - linearizability, serializability



≈

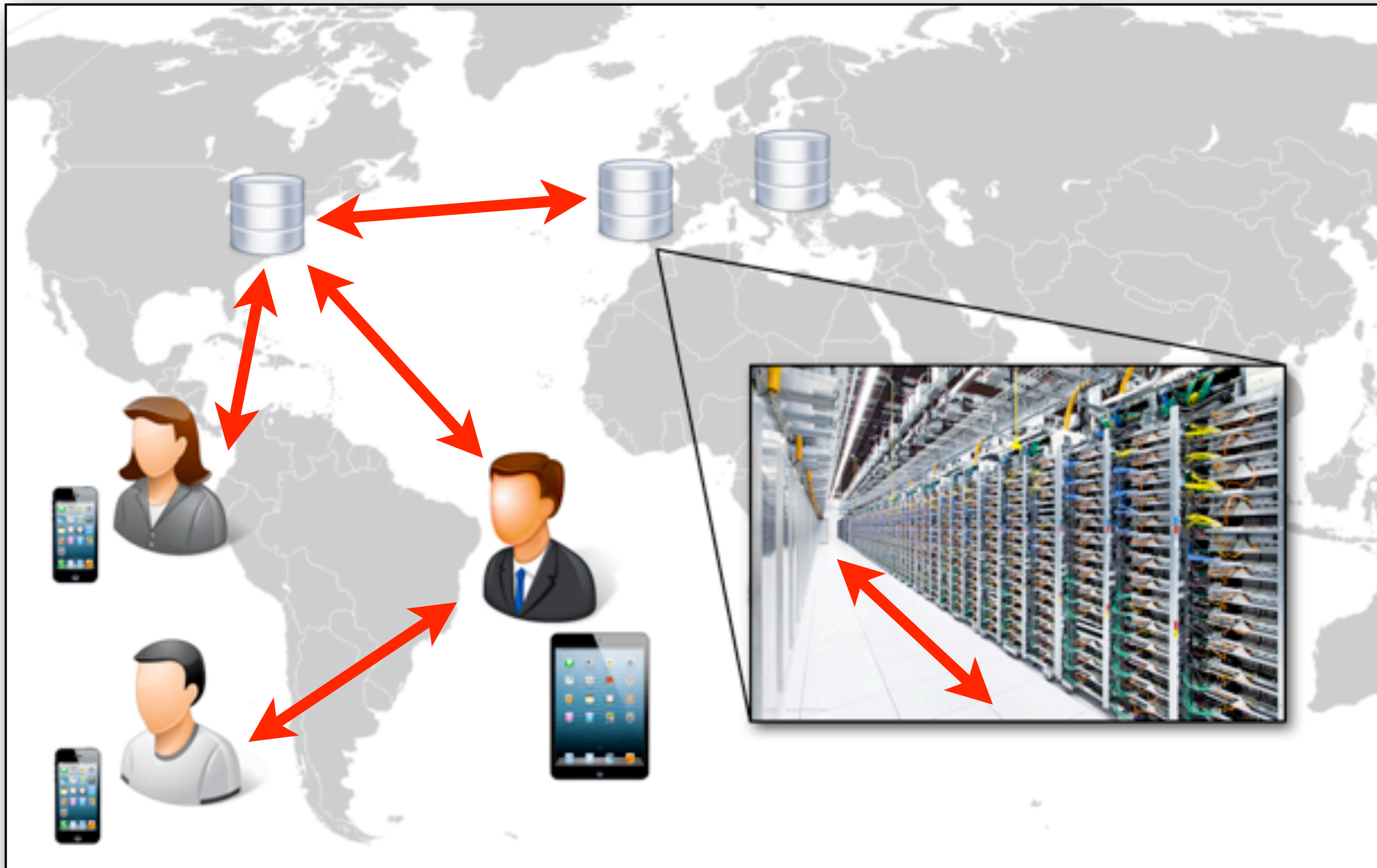
- Strong consistency model: the system behaves as if it processes requests serially on a centralised database - linearizability, serializability

- Requires synchronisation: contact other replicas when processing a request



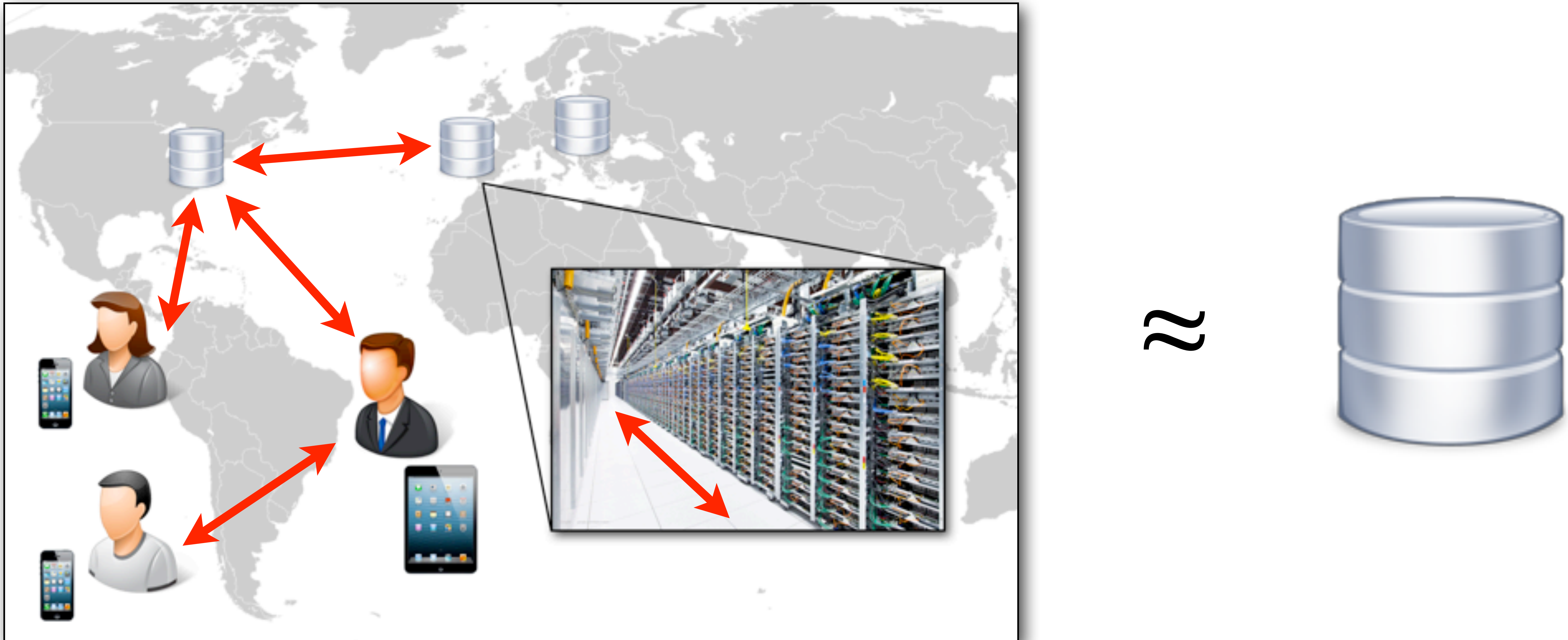
$\approx$

- Either strong Consistency or Availability in the presence of network Partitions [CAP theorem]



$\approx$

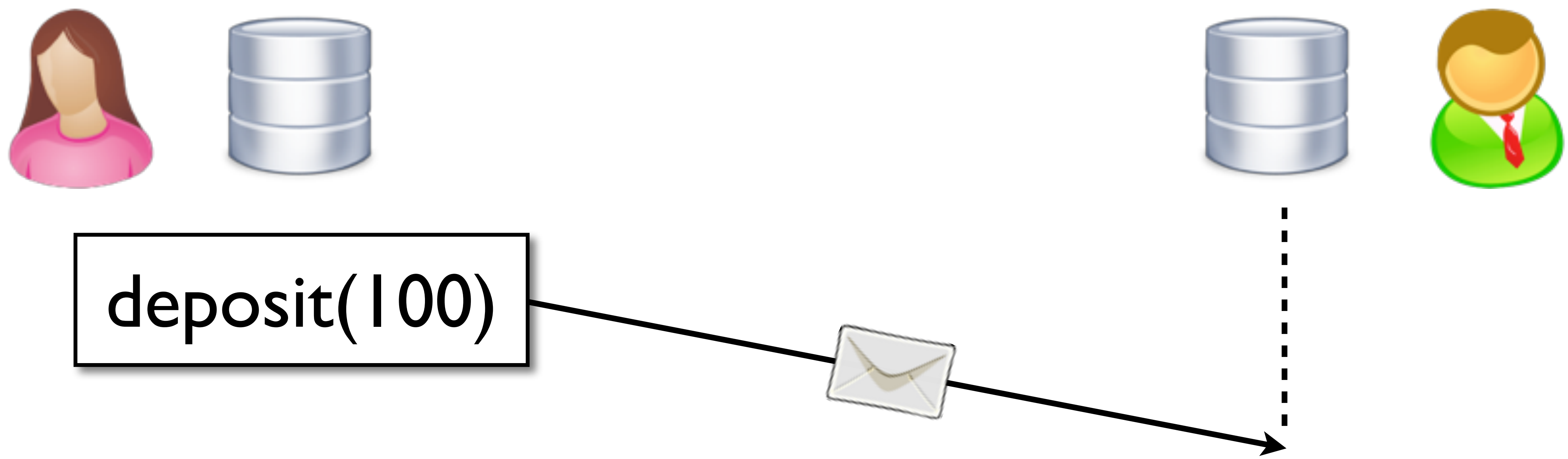
- Either ~~strong Consistency~~ or Availability in the presence of network Partitions [CAP theorem]



- Either ~~strong Consistency~~ or Availability in the presence of network Partitions [CAP theorem]

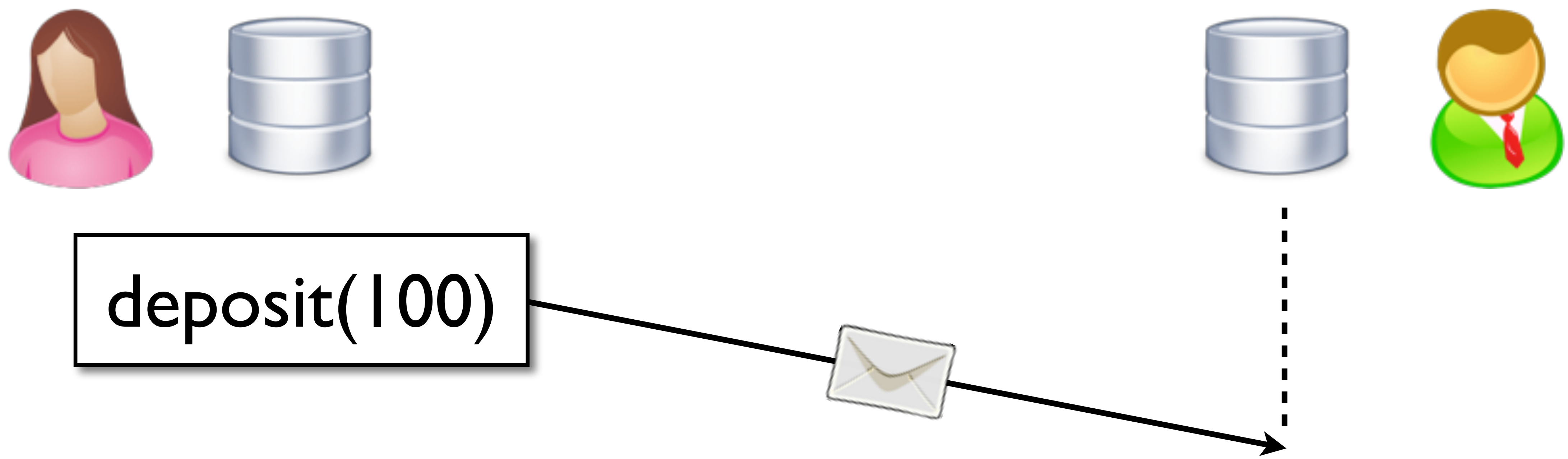
- Increased latency and resource consumption

Relaxing synchronisation



Process an update locally, propagate effects to other replicas later

Relaxing synchronisation



Process an update locally, propagate effects to other replicas later

\+ Better scalability & availability

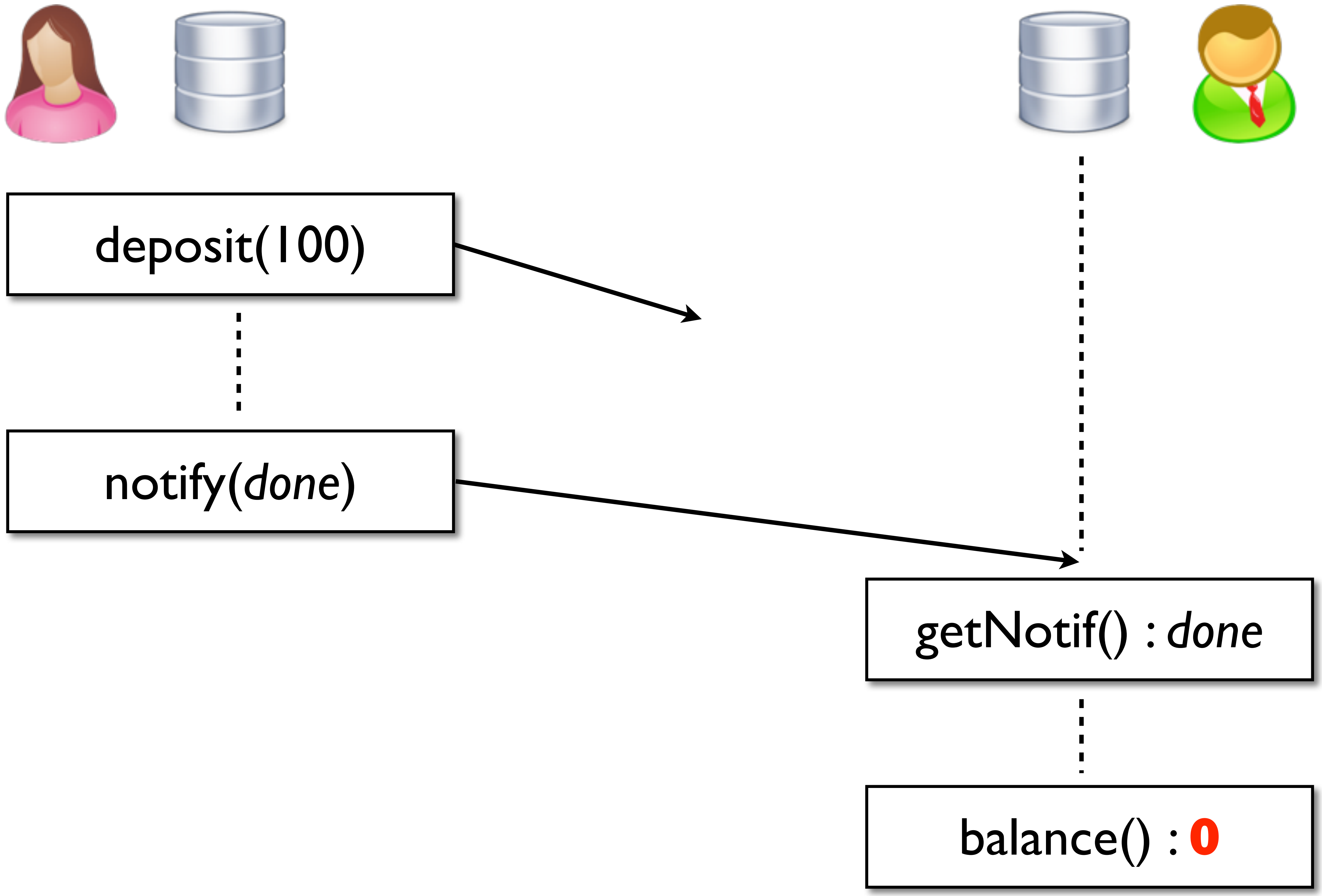
\- Weakens consistency: deposit seen with a delay

Anomalies ~ relaxed memory

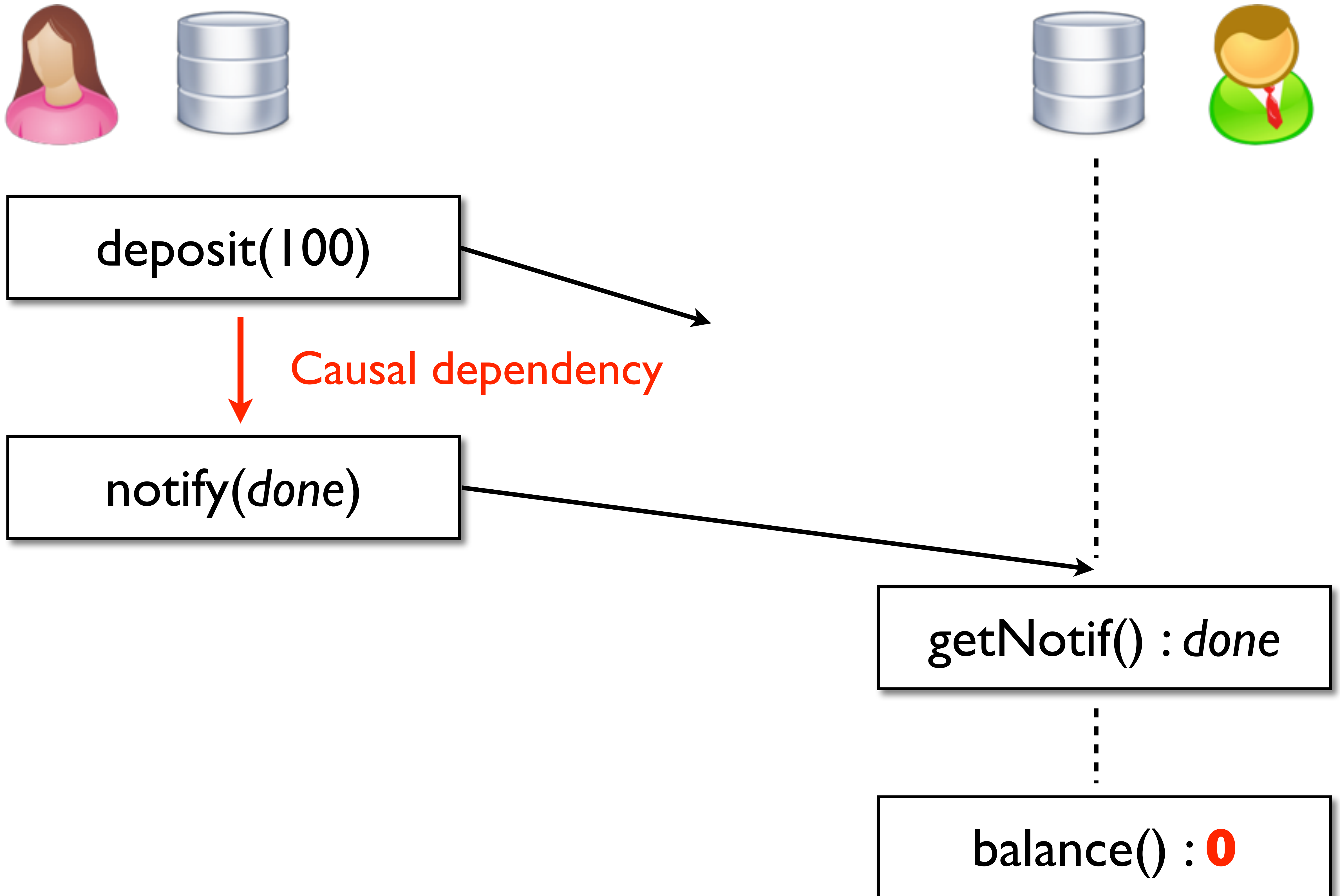
deposit(100)

notify(*done*)

Anomalies ~ relaxed memory



Anomalies ~ relaxed memory



Cassandra

Microsoft Azure DocumentDB

amazon DynamoDB

riak

Don't Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS

Wyatt Lloyd[*], Michael J. Freedman[*], Michael Kaminsky[†], and David G. Andersen[‡]
[*]Princeton University, [†]Intel Labs, [‡]Carnegie Mellon University

Highly Available Transactions: Virtues and Limitations

Peter Bailis, Aaron Davidson, Alan Fekete[†], Ali Ghodsi, Joseph M. Hellerstein, Ion Stoica
UC Berkeley and [†]University of Sydney

Transactional storage for geo-replicated systems

Yair Sovran[*] Russell Power[*] Marcos K. Aguilera[†] Jinyang Li[*]
[*]New York University [†]Microsoft Research 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

Cassandra

Microsoft Azure DocumentDB

amazon DynamoDB

Don't Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS

Wyatt Lloyd*, Michael J. Freedman*, Michael Kaminsky†, and David G. Andersen‡
*Princeton University, †Intel Labs, ‡Carnegie Mellon University

Highly Available Transactions: Virtues and Limitations

Peter Bailis, Aaron Davidson, Alan Fekete†, Ali Ghodsi, Joseph M. Hellerstein, Ion Stoica
UC Berkeley and †University of Sydney

Transactional storage for geo-replicated systems

Yair Sovran* Russell Power* Marcos K. Aguilera† Jinyang Li*
*New York University †Microsoft Research Silicon Valley

Eventually Consistent Transactions

- A spectrum of consistency models:
  eventual consistency, causal consistency, ...
- Programming constructs for mitigating weakness

Problem

- Is a given consistency model good for maintaining correctness in a given application?

  Does it weaken consistency too much, too little, just right?

- How should programmers use the models and programming features correctly?

  No guidelines, patterns, static analysis tools, informal or inadequate specifications

Problem

- Is a given consistency model good for maintaining correctness in a given application?

  Does it weaken consistency too r

  just right?

- How should programmers use

practice

DOI:10.1145/1435417.1435432

Building reliable distributed systems at a worldwide scale demands trade-offs between consistency and availability.

BY WERNER VOGELS

Eventually Consistent

AT THE FOUNDATION of Amazon's cloud computing are infrastructure services such as Amazon's S3 (Simple Storage Service), SimpleDB, and EC2 (Elastic Compute Cloud) that provide the resources for constructing Internet-scale computing platforms and a great variety of applications. The requirements placed on these infrastructure services are very strict; they need to score high marks in the areas of security, scalability, availability, performance, and cost-effectiveness, and they need to meet these requirements while serving millions of customers around the globe, continuously.

Under the covers these services are massive distributed systems that operate on a worldwide scale. This scale creates additional challenges, because when a system processes trillions and trillions of requests, events that normally have a low probability of occurrence are now guaranteed to happen and must be accounted for upfront in the design and architecture of the system. Given the worldwide scope of these systems, we use replication techniques ubiquitously to guarantee consistent performance and high availability. Although replication brings us closer to our goals, it cannot achieve them in a perfectly transparent manner; under a number of conditions the customers of these services will be confronted with the consequences of using replication techniques inside the services.

One of the ways in which this manifests itself is in the type of data consistency that is provided, particularly when many widespread distributed systems provide an *eventual consistency* model in the context of data replication. When designing these large-scale systems at Amazon, we use a set of guiding principles and abstractions related to large-scale data replication and focus on the trade-offs between high availability and data consistency. Here, I present some of the relevant background that has informed our approach to delivering reliable distributed systems that must operate on a global scale. (An earlier version of this article appeared as a posting on the "All Things Distributed" Weblog and was greatly improved with the help of its readers.)

Historical Perspective

In an ideal world there would be only one consistency model: when an update is made all observers would see that update. The first time this surfaced as difficult to achieve was in the database systems of the late 1970s. The best "period piece" on this topic is "Notes on Distributed Databases" by Bruce Lindsay et al.[5] It lays out the fundamental principles for database replication and discusses a number of techniques that deal with achieving consistency. Many of these techniques try to achieve *distribution transparency*—that is, to the user of the system it appears as if there is only one system instead of a number of collaborating systems. Many systems during this time took the approach that it was better to fail the complete system than to break this transparency.[3]

In the mid-1990s, with the rise of larger Internet systems, these practices were revisited. At that time people began to consider the idea that availability was perhaps the most impor-

"If no new updates are made to the database, then replicas will eventually reach a consistent state"

Problem

- Is a given consistency model good for maintaining correctness in a given application?

  Does it weaken consistency too m

  just right?

- How should programmers use

practice

DOI:10.1145/1435417.1435432

Building reliable distributed systems at a worldwide scale demands trade-offs between consistency and availability.

BY WERNER VOGELS

Eventually Consistent

AT THE FOUNDATION of Amazon's cloud computing are infrastructure services such as Amazon's S3 (Simple Storage Service), SimpleDB, and EC2 (Elastic Compute Cloud) that provide the resources for constructing Internet-scale computing platforms and a great variety of applications. The requirements placed on these infrastructure services are very strict; they need to score high marks in the areas of security, scalability, availability, performance, and cost-effectiveness, and they need to meet these requirements while serving millions of customers around the globe, continuously.

Under the covers these services are massive distributed systems that operate on a worldwide scale. This scale creates additional challenges, because when a system processes trillions and trillions of requests, events that normally have a low probability of occurrence are now guaranteed to happen and must be accounted for upfront in the design and architecture of the system. Given the worldwide scope of these systems, we use replication techniques ubiquitously to guarantee consistent performance and high availability. Although replication brings us closer to our goals, it cannot achieve them in a perfectly transparent manner; under a number of conditions the customers of these services will be confronted with the consequences of using replication techniques inside the services.

One of the ways in which this manifests itself is in the type of data consistency that is provided, particularly when many widespread distributed systems provide an *eventual consistency* model in the context of data replication. When designing these large-scale systems at Amazon, we use a set of guiding principles and abstractions related to large-scale data replication and focus on the trade-offs between high availability and data consistency. Here, I present some of the relevant background that has informed our approach to delivering reliable distributed systems that must operate on a global scale. (An earlier version of this article appeared as a posting on the "All Things Distributed" Weblog and was greatly improved with the help of its readers.)

Historical Perspective

In an ideal world there would be only one consistency model: when an update is made all observers would see that update. The first time this surfaced as difficult to achieve was in the database systems of the late 1970s. The best "period piece" on this topic is "Notes on Distributed Databases" by Bruce Lindsay et al.[5] It lays out the fundamental principles for database replication and discusses a number of techniques that deal with achieving consistency. Many of these techniques try to achieve *distribution transparency*—that is, to the user of the system it appears as if there is only one system instead of a number of collaborating systems. Many systems during this time took the approach that it was better to fail the complete system than to break this transparency.[3]

In the mid-1990s, with the rise of larger Internet systems, these practices were revisited. At that time people began to consider the idea that availability was perhaps the most impor-

"*If no new updates are made* to the *database, then replicas will eventually reach a consistent state*"

Problem

- Is a given consistency model good for maintaining correctness in a given application?

  Does it weaken consistency too m

  just right?

- How should programmers use

practice

DOI:10.1145/1435417.1435432

Building reliable distributed systems at a worldwide scale demands trade-offs between consistency and availability.

BY WERNER VOGELS

Eventually Consistent

AT THE FOUNDATION of Amazon's cloud computing are infrastructure services such as Amazon's S3 (Simple Storage Service), SimpleDB, and EC2 (Elastic Compute Cloud) that provide the resources for constructing Internet-scale computing platforms and a great variety of applications. The requirements placed on these infrastructure services are very strict; they need to score high marks in the areas of security, scalability, availability, performance, and cost-effectiveness, and they need to meet these requirements while serving millions of customers around the globe, continuously.

Under the covers these services are massive distributed systems that operate on a worldwide scale. This scale creates additional challenges, because when a system processes trillions and trillions of requests, events that normally have a low probability of occurrence are now guaranteed to happen and must be accounted for upfront in the design and architecture of the system. Given the worldwide scope of these systems, we use replication techniques ubiquitously to guarantee consistent performance and high availability. Although replication brings us closer to our goals, it cannot achieve them in a perfectly transparent manner; under a number of conditions the customers of these services will be confronted with the consequences of using replication techniques inside the services.

One of the ways in which this manifests itself is in the type of data consistency that is provided, particularly when many widespread distributed systems provide an *eventual consistency* model in the context of data replication. When designing these large-scale systems at Amazon, we use a set of guiding principles and abstractions related to large-scale data replication and focus on the trade-offs between high availability and data consistency. Here, I present some of the relevant background that has informed our approach to delivering reliable distributed systems that must operate on a global scale. (An earlier version of this article appeared as a posting on the "All Things Distributed" Weblog and was greatly improved with the help of its readers.)

Historical Perspective

In an ideal world there would be only one consistency model: when an update is made all observers would see that update. The first time this surfaced as difficult to achieve was in the database systems of the late 1970s. The best "period piece" on this topic is "Notes on Distributed Databases" by Bruce Lindsay et al.[5] It lays out the fundamental principles for database replication and discusses a number of techniques that deal with achieving consistency. Many of these techniques try to achieve *distribution transparency*—that is, to the user of the system it appears as if there is only one system instead of a number of collaborating systems. Many systems during this time took the approach that it was better to fail the complete system than to break this transparency.[3]

In the mid-1990s, with the rise of larger Internet systems, these practices were revisited. At that time people began to consider the idea that availability was perhaps the most impor-

"If no new updates are made to the database, then replicas will eventually reach a consistent state"

TOWARDS A CLOUD COMPUTING RESEARCH AGENDA

Ken Birman, Gregory Chockler, Robbert van Renesse

This particular example is a good one because, as we'll see shortly, if there was a single overarching theme within the keynote talks, it turns out to be that strong synchronization of the sort provided by a locking service must be avoided like the plague. This doesn't diminish the need for a tool like Chubby; when locking actually can't be avoided, one wants a reliable, standard, provably correct

TOWARDS A CLOUD COMPUTING RESEARCH AGENDA

Ken Birman, Gregory Chockler, Robbert van Renesse

This particular example is a good one because, as we'll see shortly, if there was a single overarching theme within the keynote talks, it turns out to be that strong synchronization of the sort provided by a locking service must be avoided like the plague. This doesn't diminish the need for a tool like Chubby; when locking actually can't be avoided, one wants a reliable, standard, provably correct

F1: A Distributed SQL Database That Scales

Jeff Shute, Chad Whipkey, David Menestrina, Radek Vingralek, Eric Rollins, Stephan Ellner, Traian Stancescu, Bart Samwel, Mircea Oancea, John Cieslewicz, Himani Apte, Ben Handy, Kyle Littlefield, Ian Rae*

Google, Inc.
*University of Wisconsin-Madison

ABSTRACT

F1 is a distributed relational database system built at Google to support the AdWords business. F1 is a hybrid database that combines high availability, the scalability of NoSQL systems like Bigtable, and the consistency and us-

consistent and correct data.

Designing applications to cope with concurrency anomalies in their data is very error-prone, time-consuming, and ultimately not worth the performance gains.

TOWARDS A CLOUD COMPUTING RESEARCH AGENDA

Ken Birman, Gregory Chockler, Robbert van Renesse

This particular example is a good one because, as we'll see shortly, if there was a single overarching theme within the keynote talks, it turns out to be that strong synchronization of the sort provided by a locking service must be avoided like the plague. This doesn't diminish the need for a tool like

Need a rigorous approach: programming models and static analysis tools that allow relaxing synchronisation without compromising correctness

David Menestrina Stephan Ellner John Cieslewicz Ian Rae*
Traian Stancescu Himani Apte

Google, Inc.
*University of Wisconsin-Madison

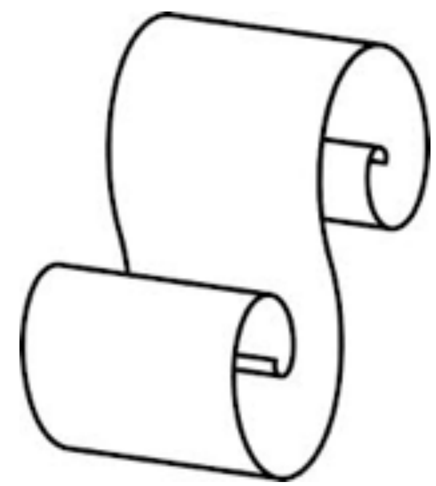
ABSTRACT

F1 is a distributed relational database system built at Google to support the AdWords business. F1 is a hybrid database that combines high availability, the scalability of NoSQL systems like Bigtable, and the consistency and us-

consistent and correct data.
Designing applications to cope with concurrency anomalies in their data is very error-prone, time-consuming, and ultimately not worth the performance gains.

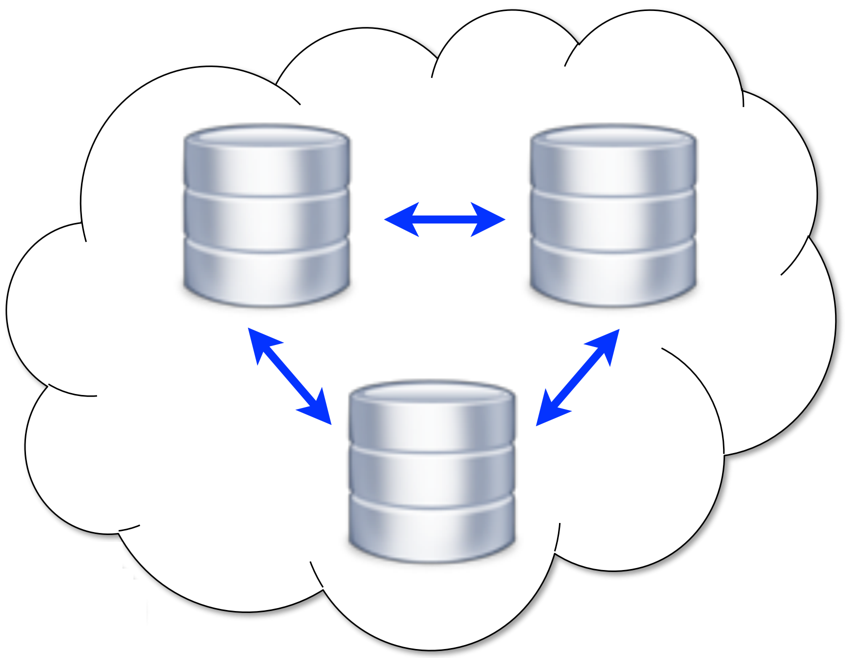
Research agenda

Research agenda

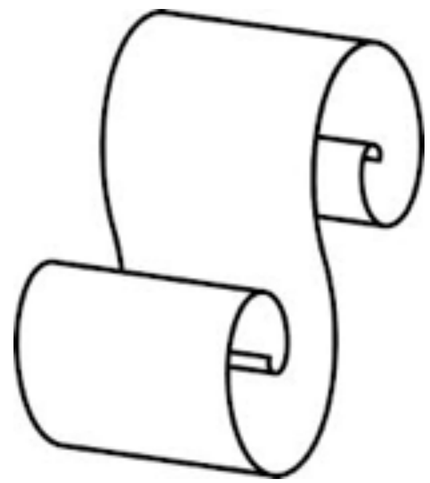


Formal semantics
specification

Research agenda

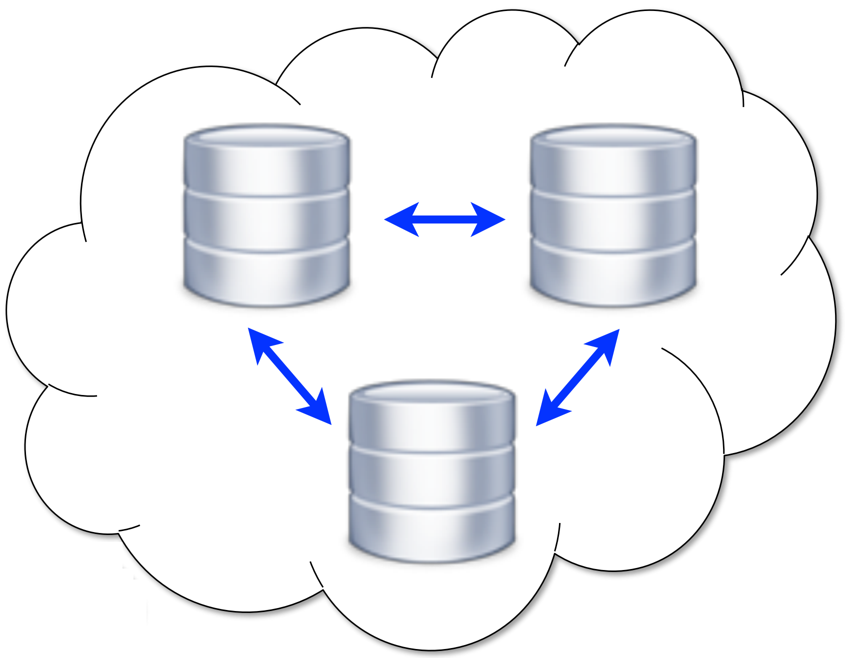


Reasoning about DB implementations

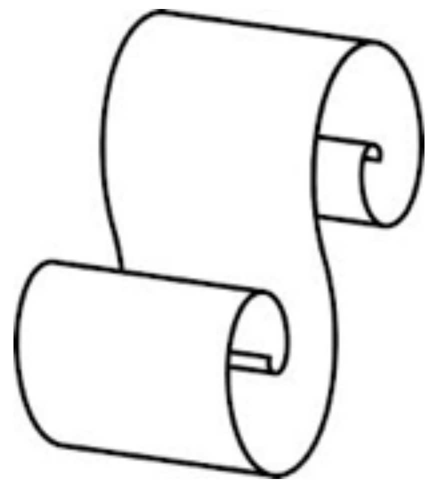


Formal semantics specification

Research agenda



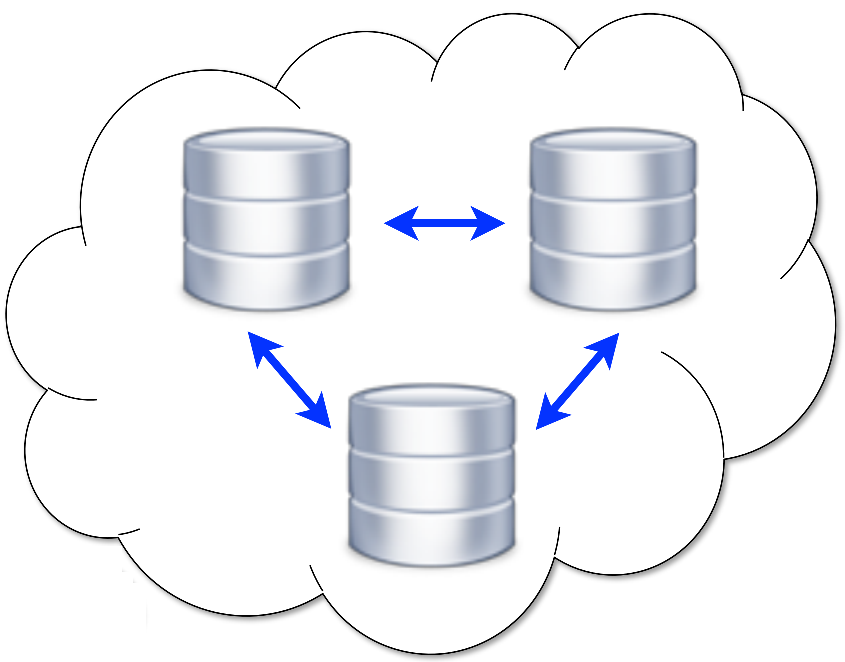
Reasoning about DB implementations



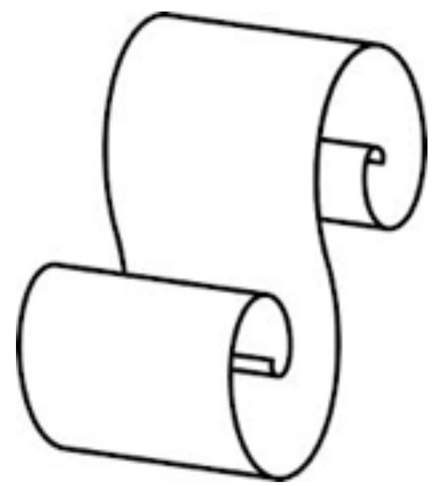
Formal semantics specification

Reasoning about applications

Research agenda



Reasoning about DB implementations

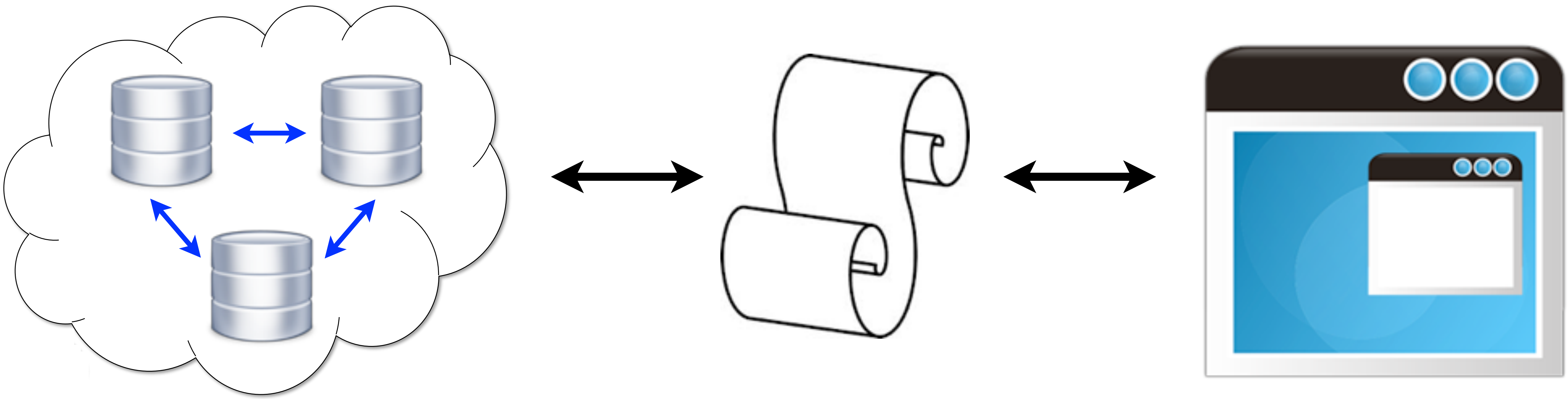


Formal semantics specification

Reasoning about applications

Improve DB programming models and implementations

Research agenda



Reasoning about DB implementations

Formal semantics specification

Reasoning about applications

Joint work with Hongseok Yang (Oxford), Carla Ferreira (U Nova Lisboa), Mahsa Najafzadeh, Marc Shapiro (INRIA)

Synchronisation can be necessary

balance ≥ 0

balance = 100

balance = 100

Synchronisation can be necessary

balance ≥ 0

balance = 100

withdraw(100) : ✔

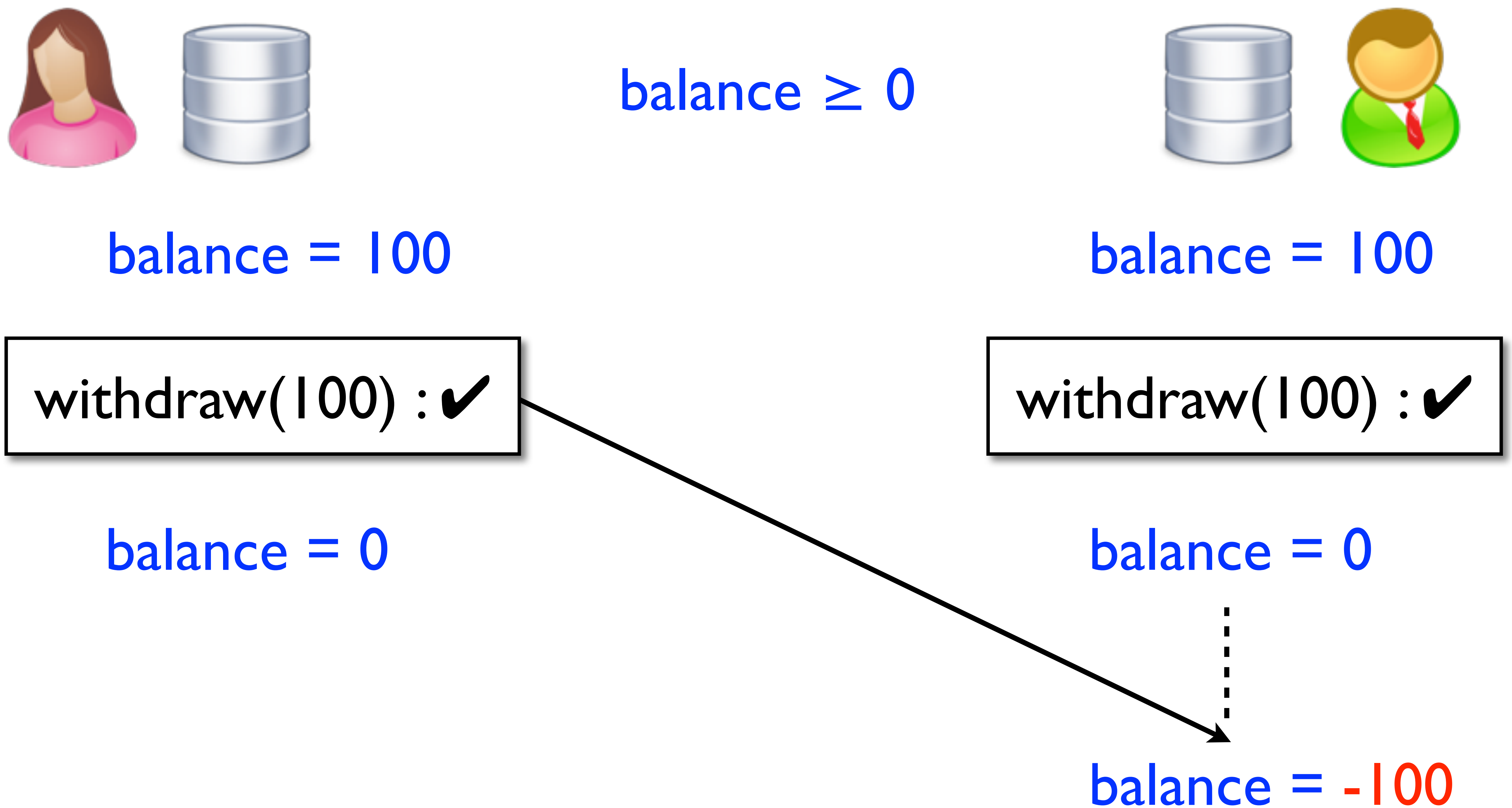
balance = 0

balance = 100

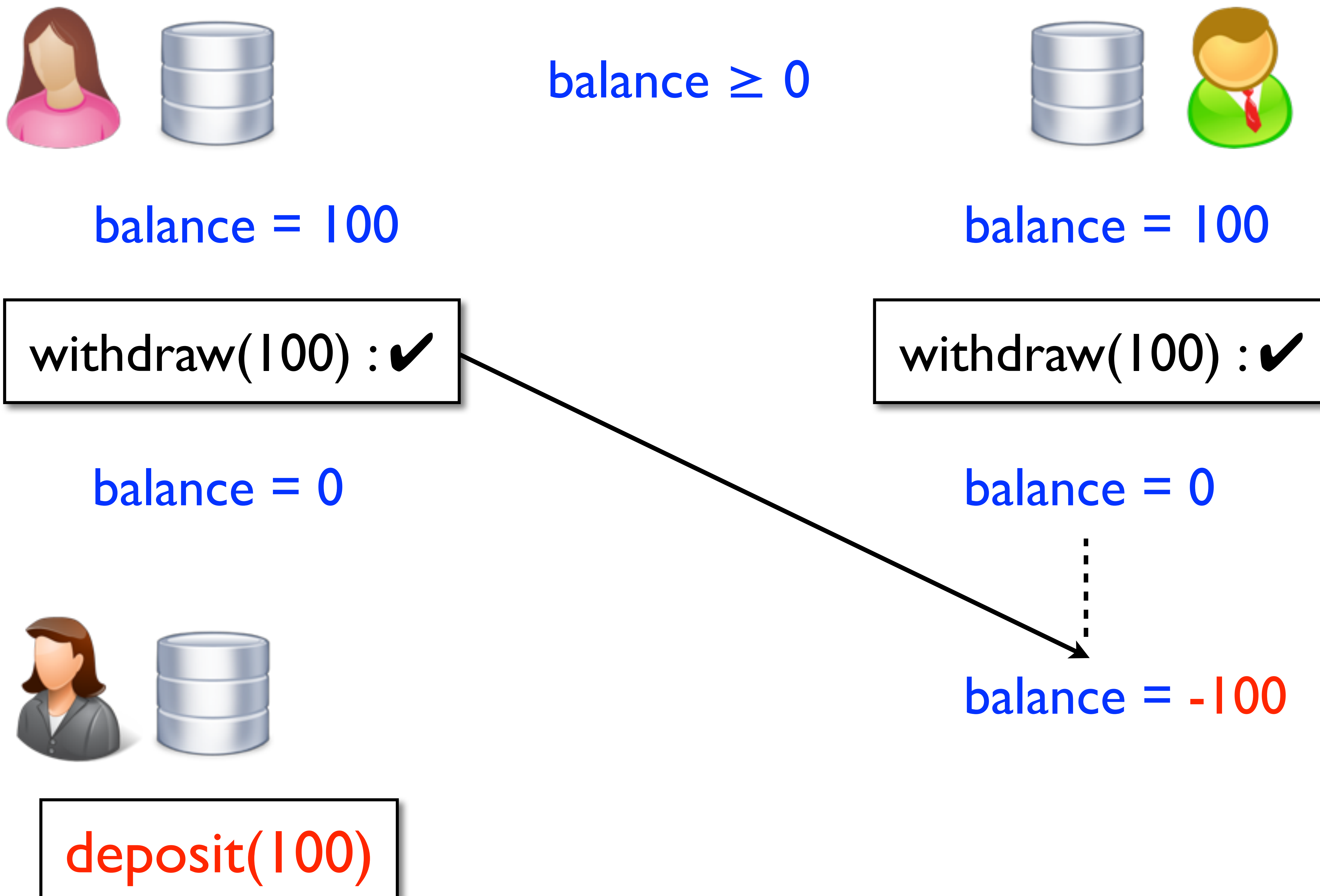
withdraw(100) : ✔

balance = 0

Synchronisation can be necessary



Synchronisation can be necessary



Consistency choices

- Choose consistency level for each operation:
  - Withdrawals strongly consistent
  - Deposits eventually consistent
- Pay for stronger semantics with latency, possible unavailability and money
- Hard to figure out the minimum consistency necessary to maintain correctness - proof rule and tool

Consistency model

Generic model - not implemented, but can encode many existing models that are

- Causal consistency as a baseline: observe an update ➜ observe the updates it depends on
- A construct for strengthening consistency on demand

Operation semantics

σ

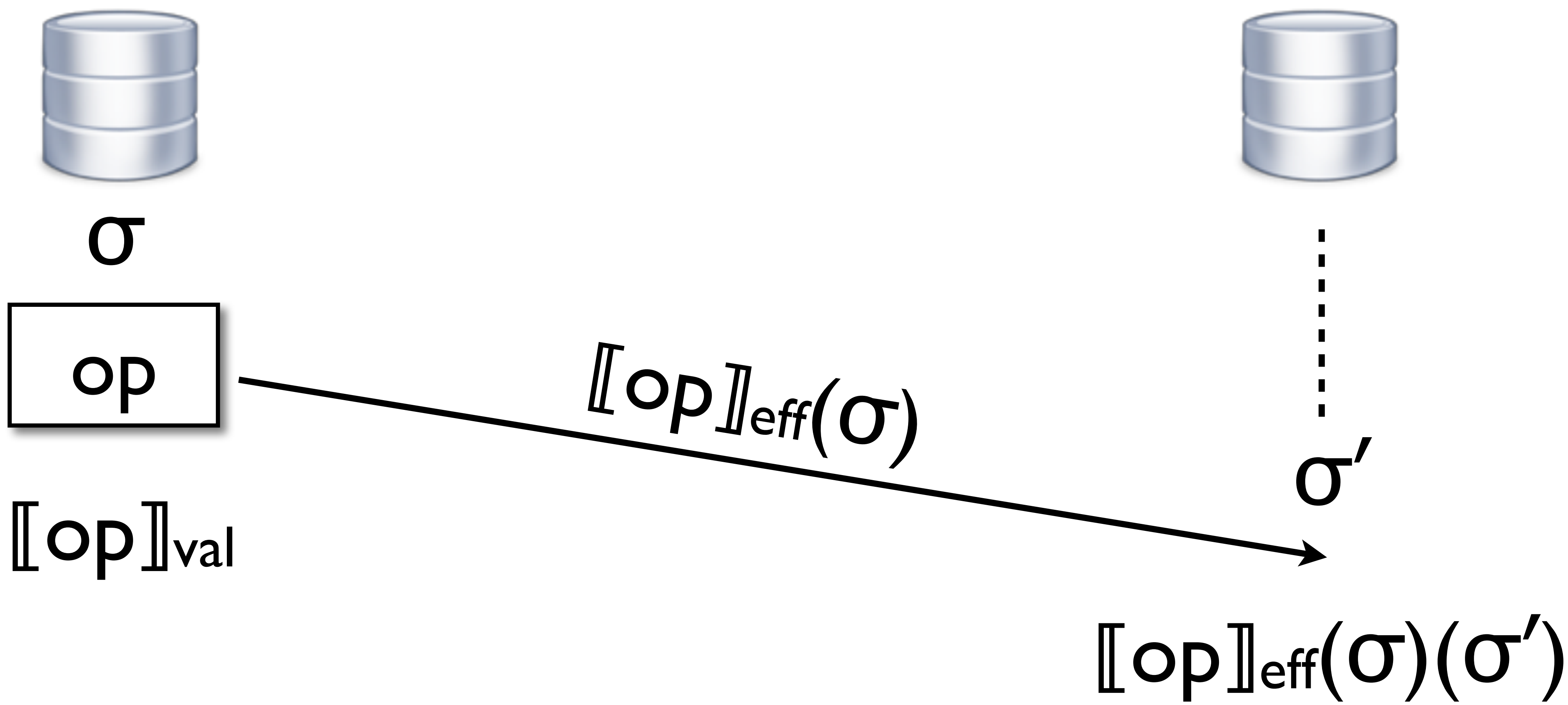
$\boxed{\text{op}}$

$[\![\text{op}]\!]_{\text{val}}$

Replica states: $\sigma \in \text{State}$

Return value: $[\![\text{op}]\!]_{\text{val}} \in \text{State} \rightarrow \text{Value}$

Operation semantics

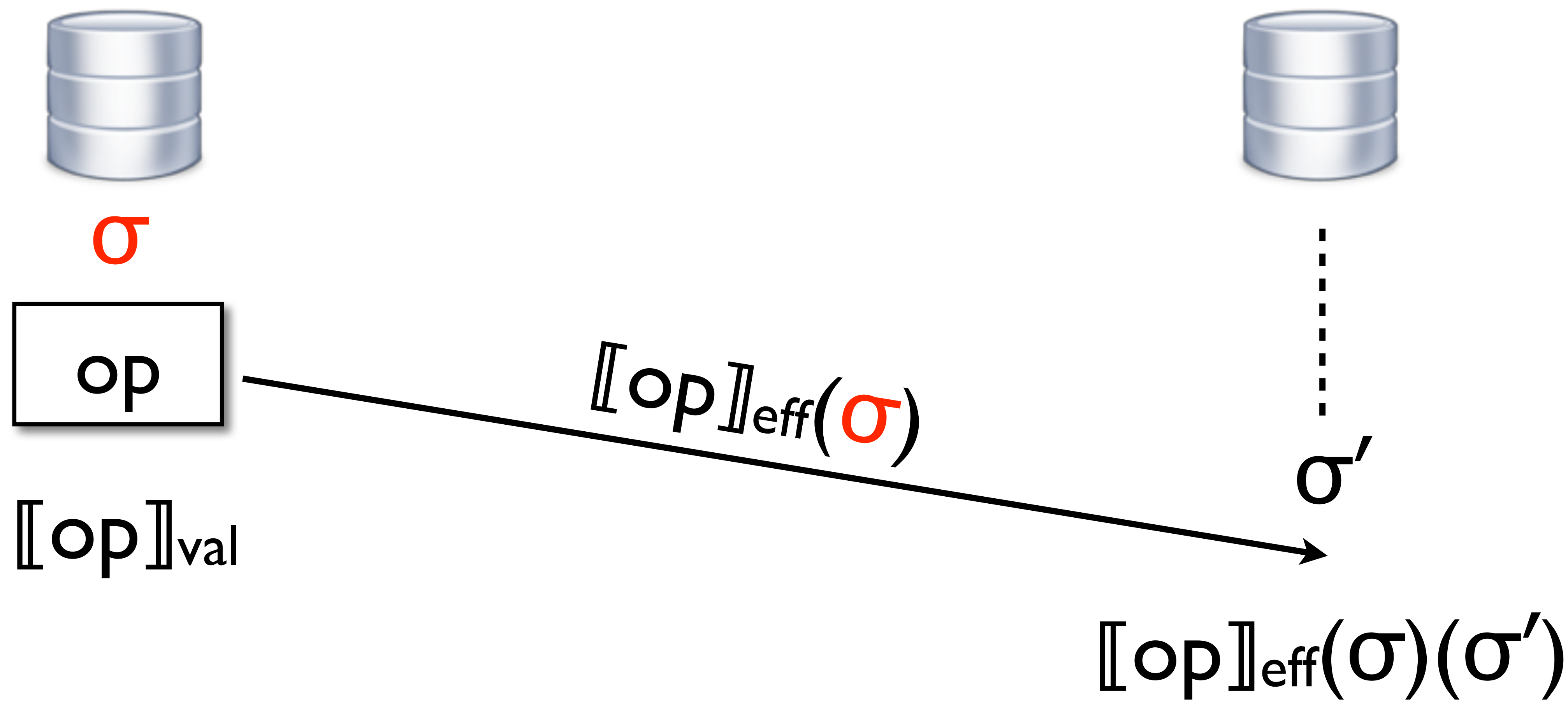


Replica states: $\sigma \in \mathrm{State}$

Return value: $[\![op]\!]_{val} \in \mathrm{State} \rightarrow \mathrm{Value}$

Effector: $[\![op]\!]_{eff} \in \mathrm{State} \rightarrow (\mathrm{State} \rightarrow \mathrm{State})$

Operation semantics

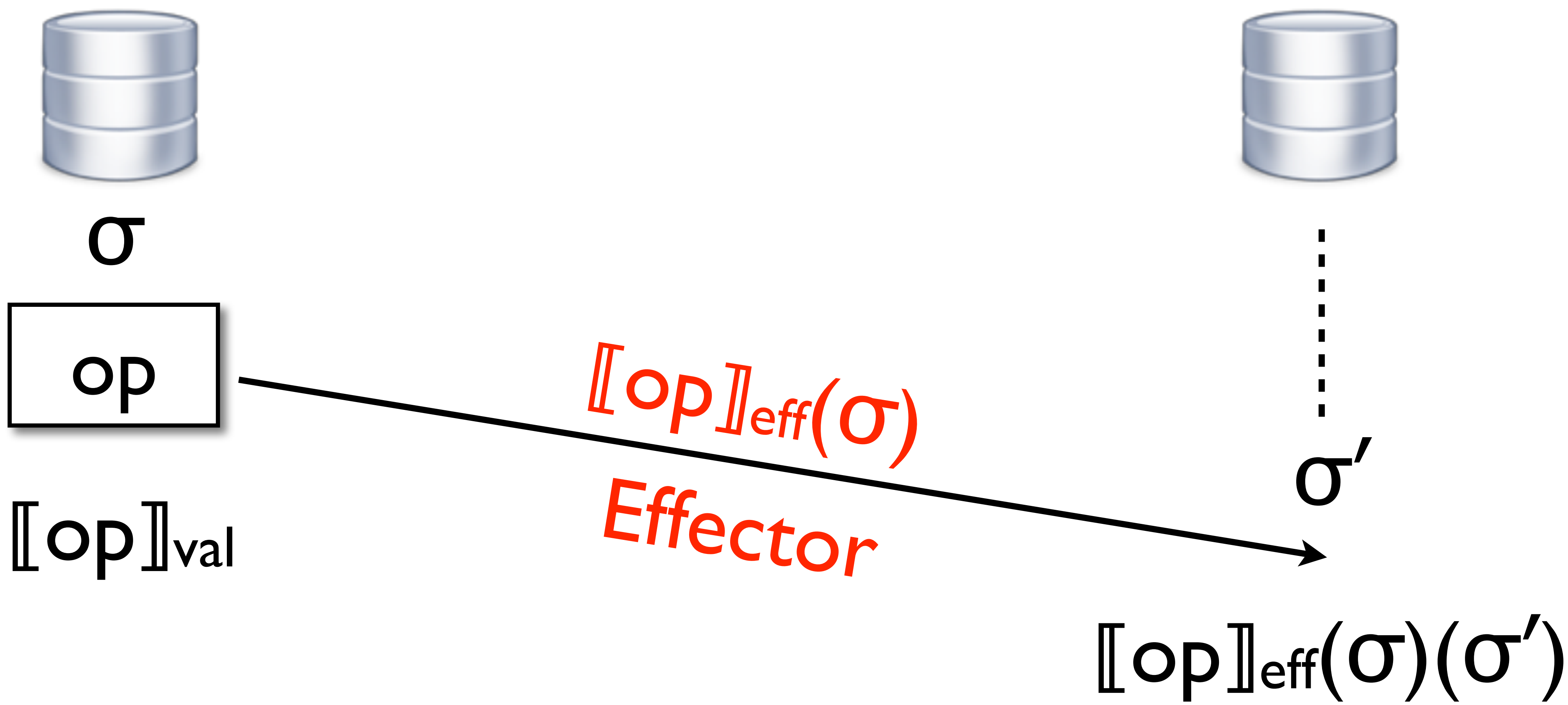


Replica states: $\sigma \in \text{State}$

Return value: $[\![op]\!]_{val} \in \text{State} \rightarrow \text{Value}$

Effector: $[\![op]\!]_{eff} \in \text{State} \rightarrow (\text{State} \rightarrow \text{State})$

Operation semantics

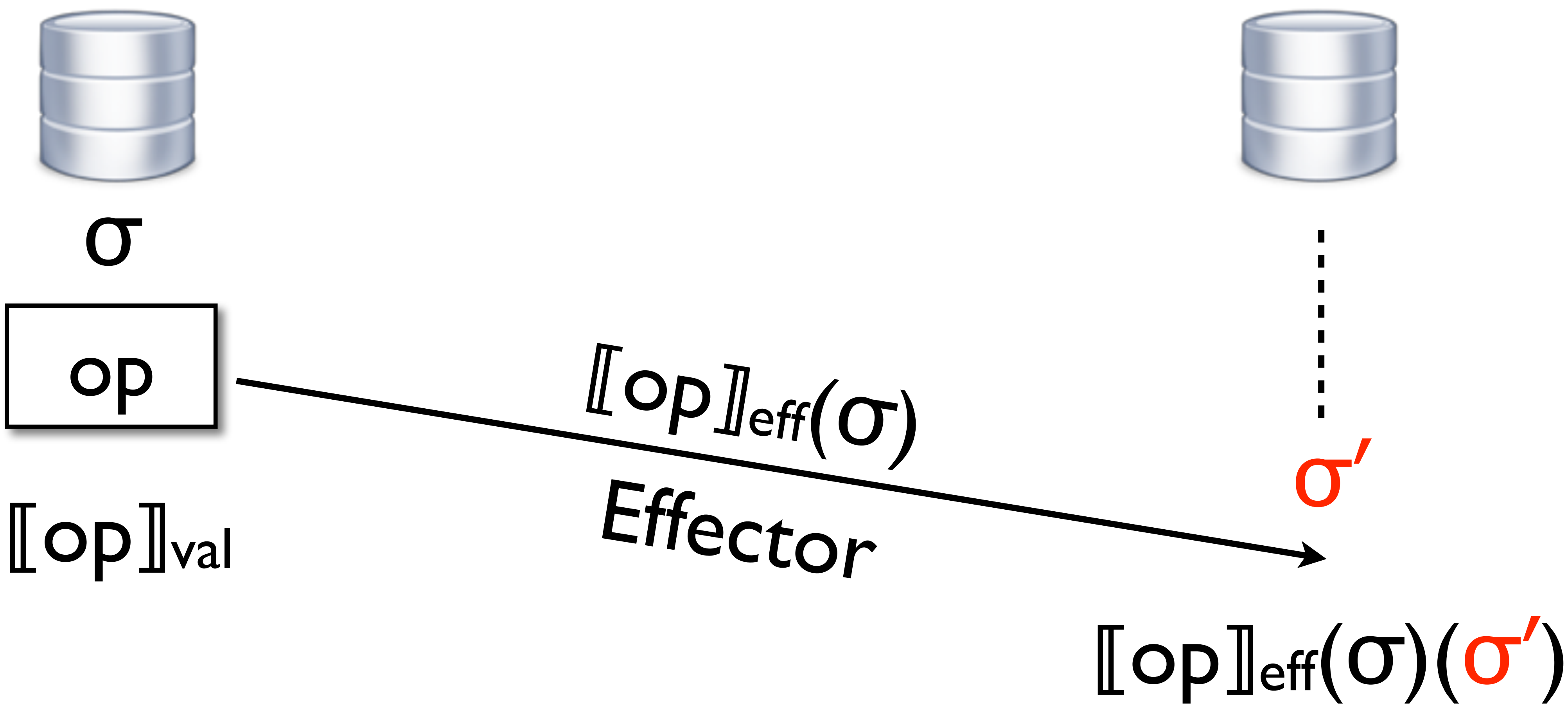


Replica states: $\sigma \in \mathrm{State}$

Return value: $[\![op]\!]_{val} \in \mathrm{State} \rightarrow \mathrm{Value}$

Effector: $[\![op]\!]_{eff} \in \mathrm{State} \rightarrow (\mathrm{State} \rightarrow \mathrm{State})$

Operation semantics

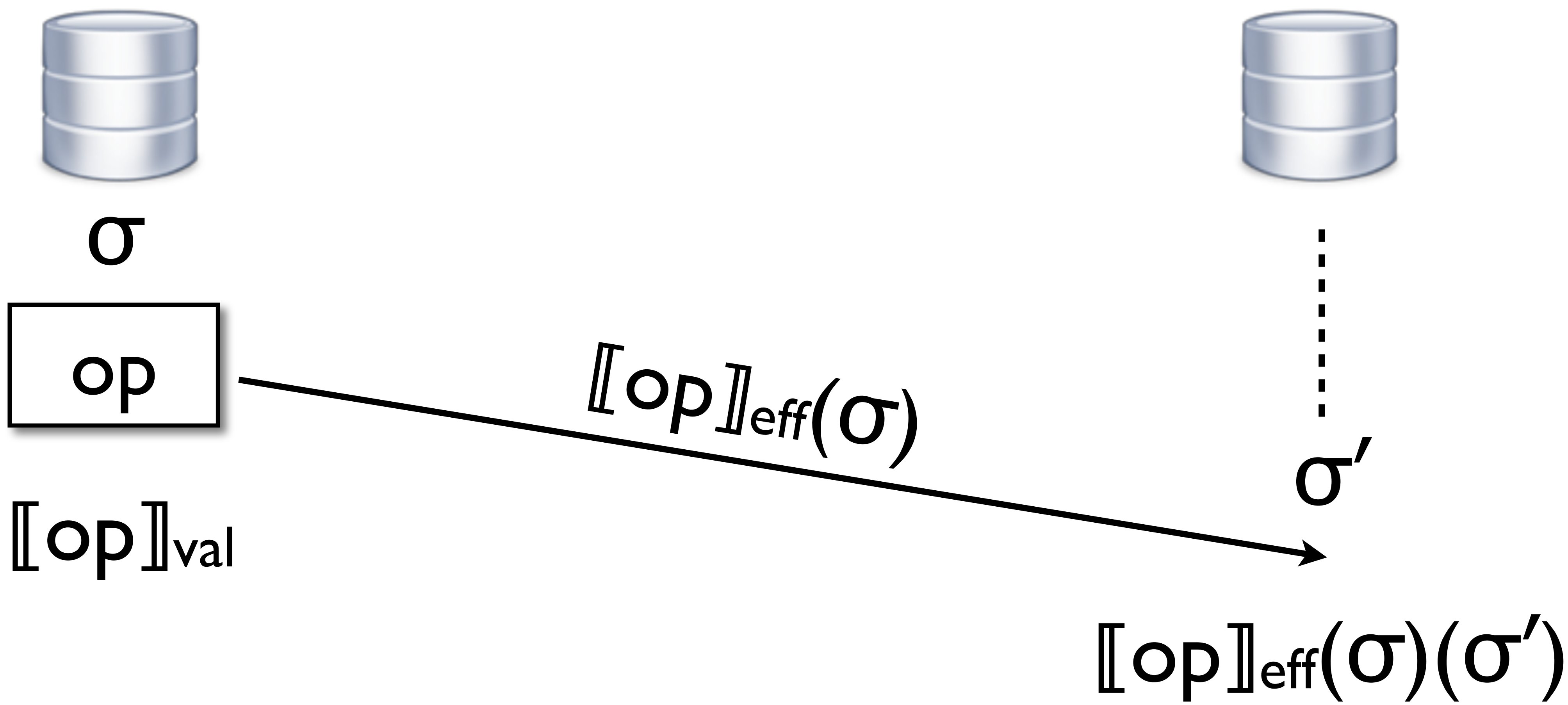


Replica states: $\sigma \in \mathrm{State}$

Return value: $[\![op]\!]_{val} \in \mathrm{State} \to \mathrm{Value}$

Effector: $[\![op]\!]_{eff} \in \mathrm{State} \to (\mathrm{State} \to \mathrm{State})$

Operation semantics

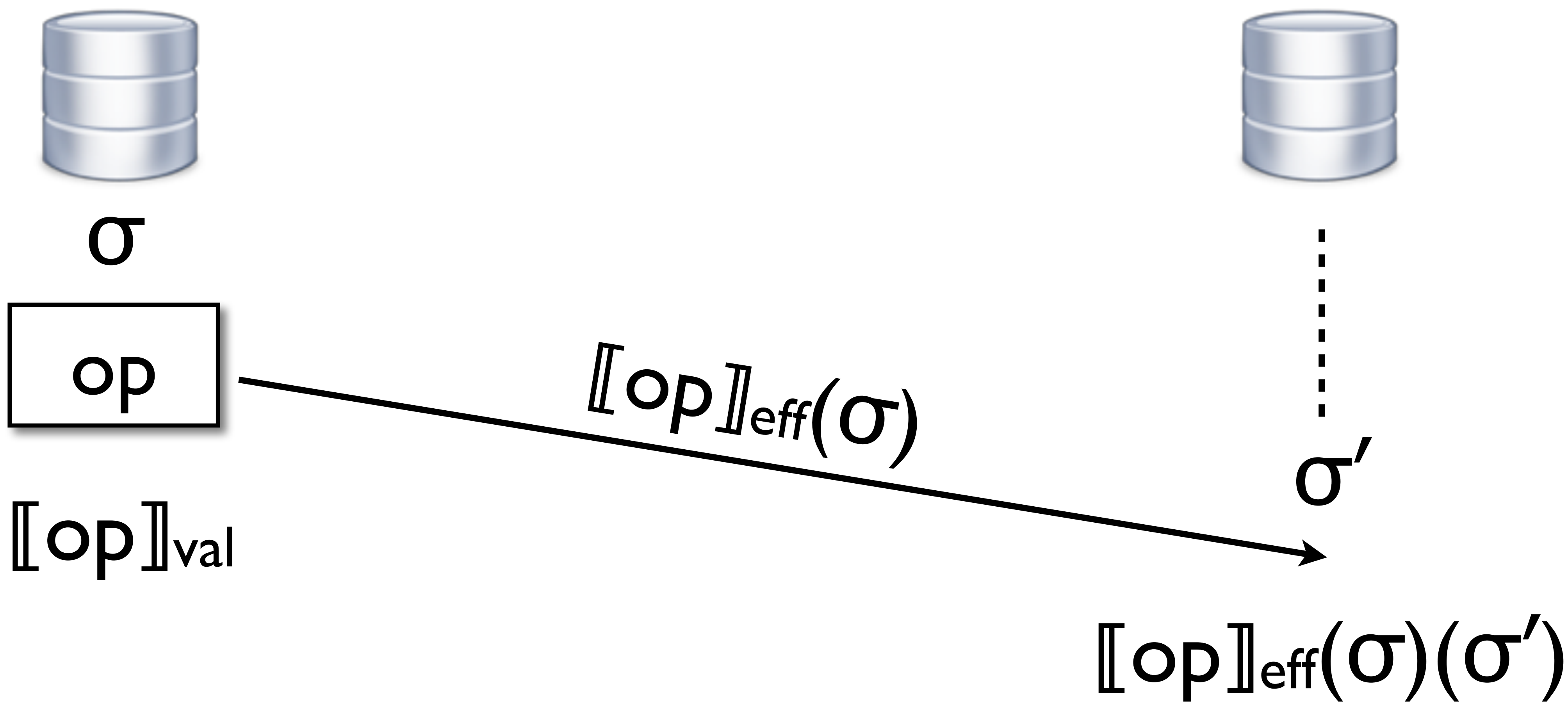


$\mathrm{State} \;=\; \mathbb{Z}$

$[\![\mathrm{balance}()]\!]_{\mathrm{val}}(\sigma) \;=\; \sigma$

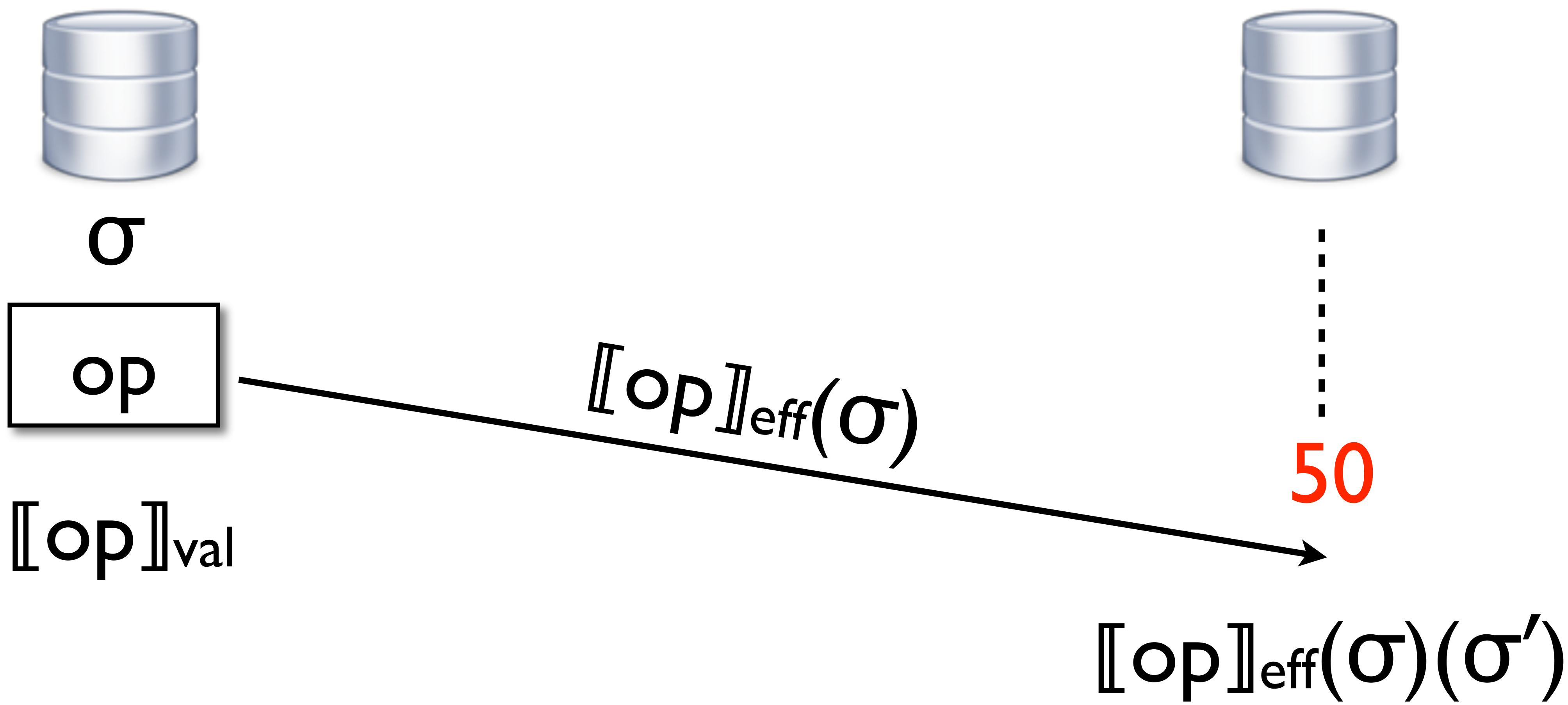
$[\![\mathrm{balance}()]\!]_{\mathrm{eff}}(\sigma) \;=\; \lambda\sigma.\,\sigma$

Operation semantics



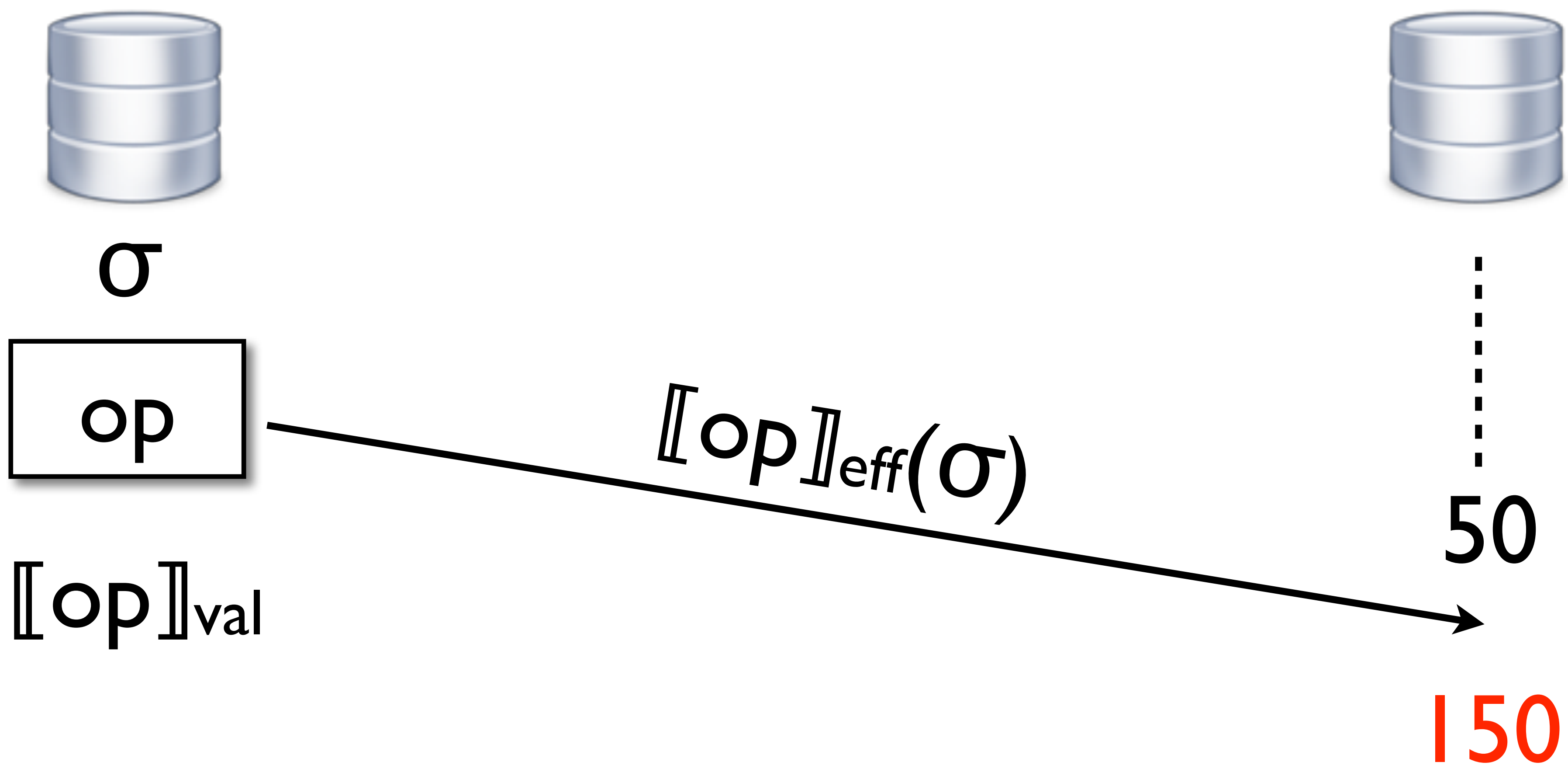
$$[\![deposit(100)]\!]_{eff}(\sigma) \;=\; \lambda\sigma'.\,(\sigma' + 100)$$

Operation semantics



$$[\![\mathrm{deposit}(100)]\!]_{\mathrm{eff}}(\sigma) \;=\; \lambda\sigma'.\,(\sigma' + 100)$$

Operation semantics



$$[\![\mathrm{deposit}(100)]\!]_{\mathrm{eff}}(\sigma) \;=\; \lambda\sigma'.\,(\sigma' + 100)$$

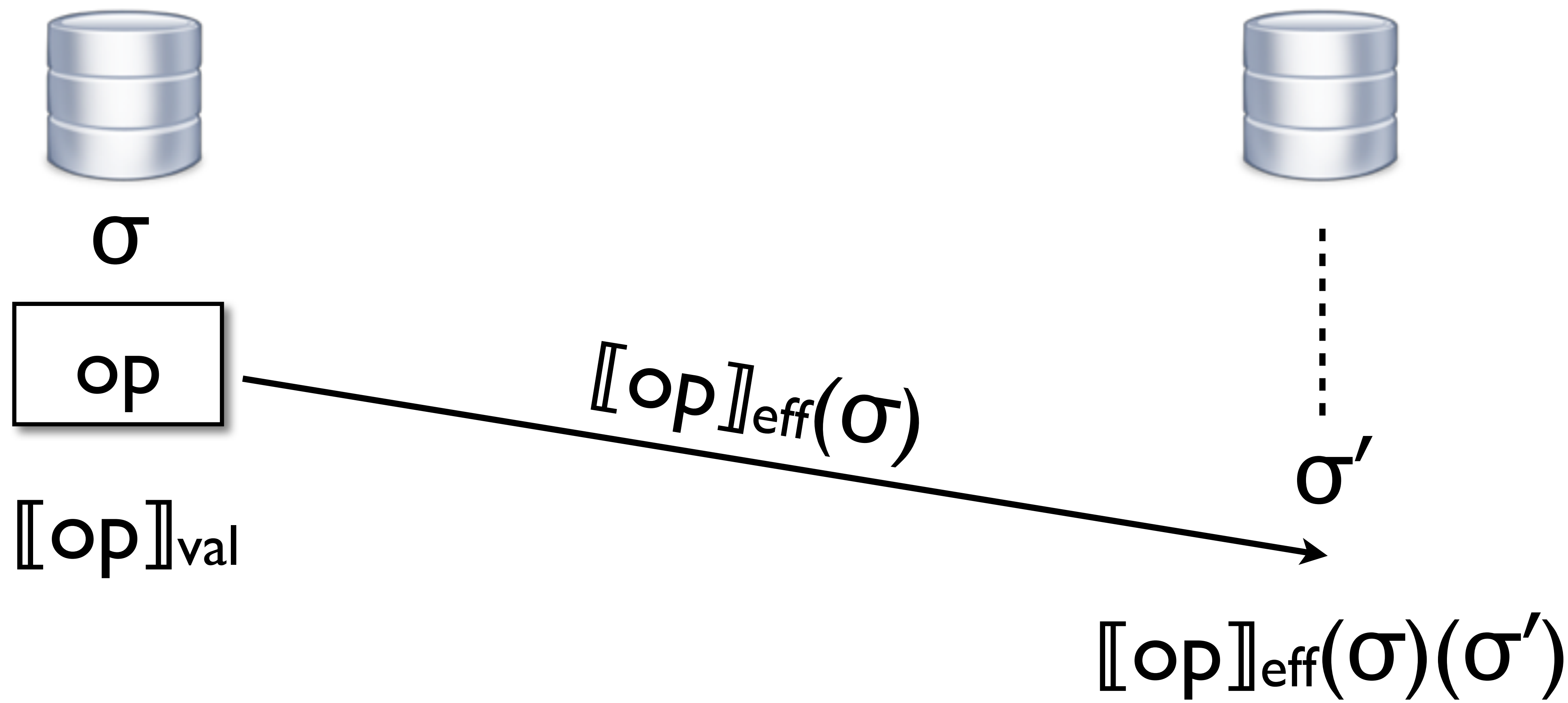
Ensuring eventual consistency

- Effectors have to commute:

$$\forall op_1, op_2, \sigma_1, \sigma_2.\ [\![op_1]\!]_{eff}(\sigma_1) \,;\, [\![op_2]\!]_{eff}(\sigma_2) = [\![op_2]\!]_{eff}(\sigma_2) \,;\, [\![op_1]\!]_{eff}(\sigma_1)$$

- Eventual consistency: replicas receiving the same messages in different orders end up in the same state
- Replicated data types [Shapiro+ 2011]: ready-made commutative implementations

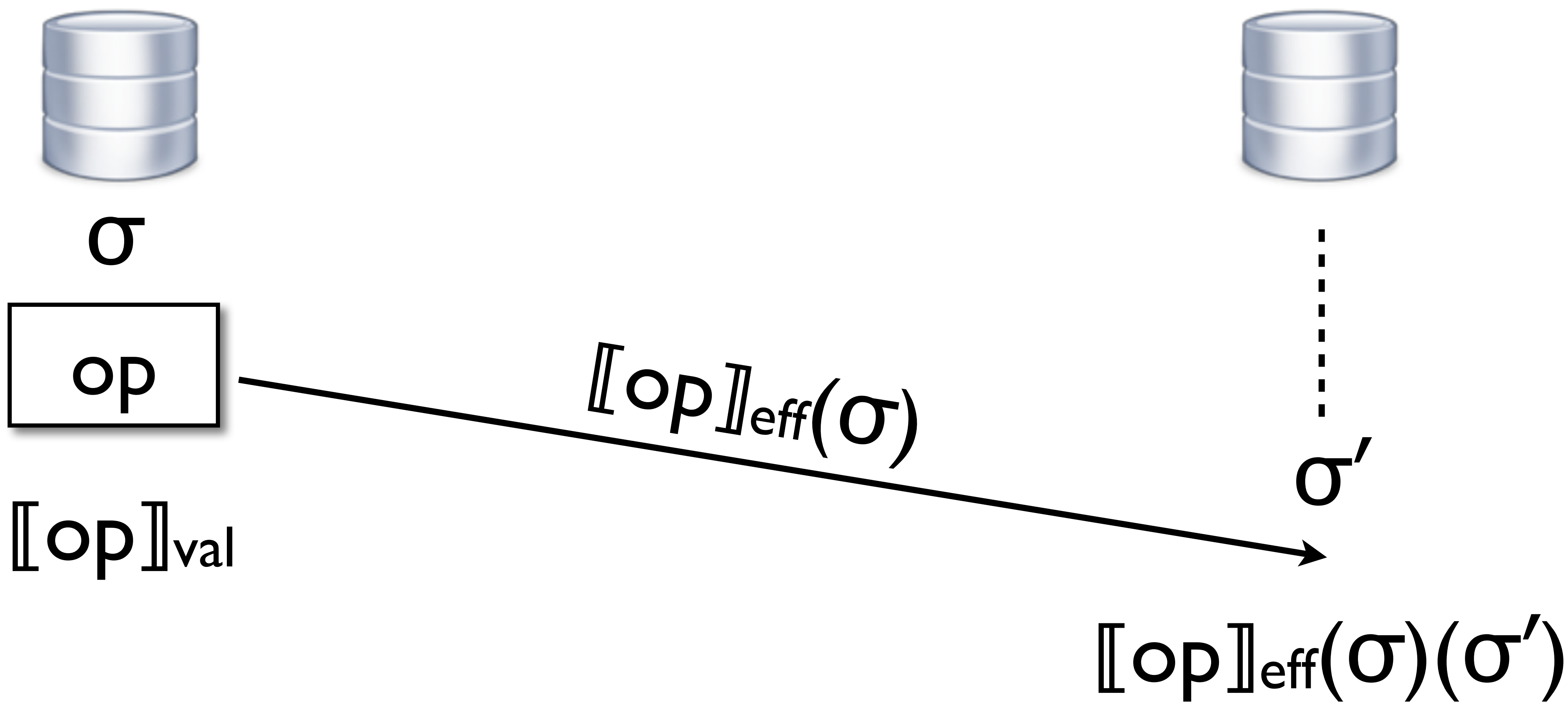
Operation semantics



$[\![\text{withdraw}(100)]\!]_{\text{eff}}(\sigma)\ =$

$\text{if } \sigma \geq 100 \text{ then } (\lambda\sigma'.\,\sigma' - 100) \text{ else } (\lambda\sigma'.\,\sigma')$

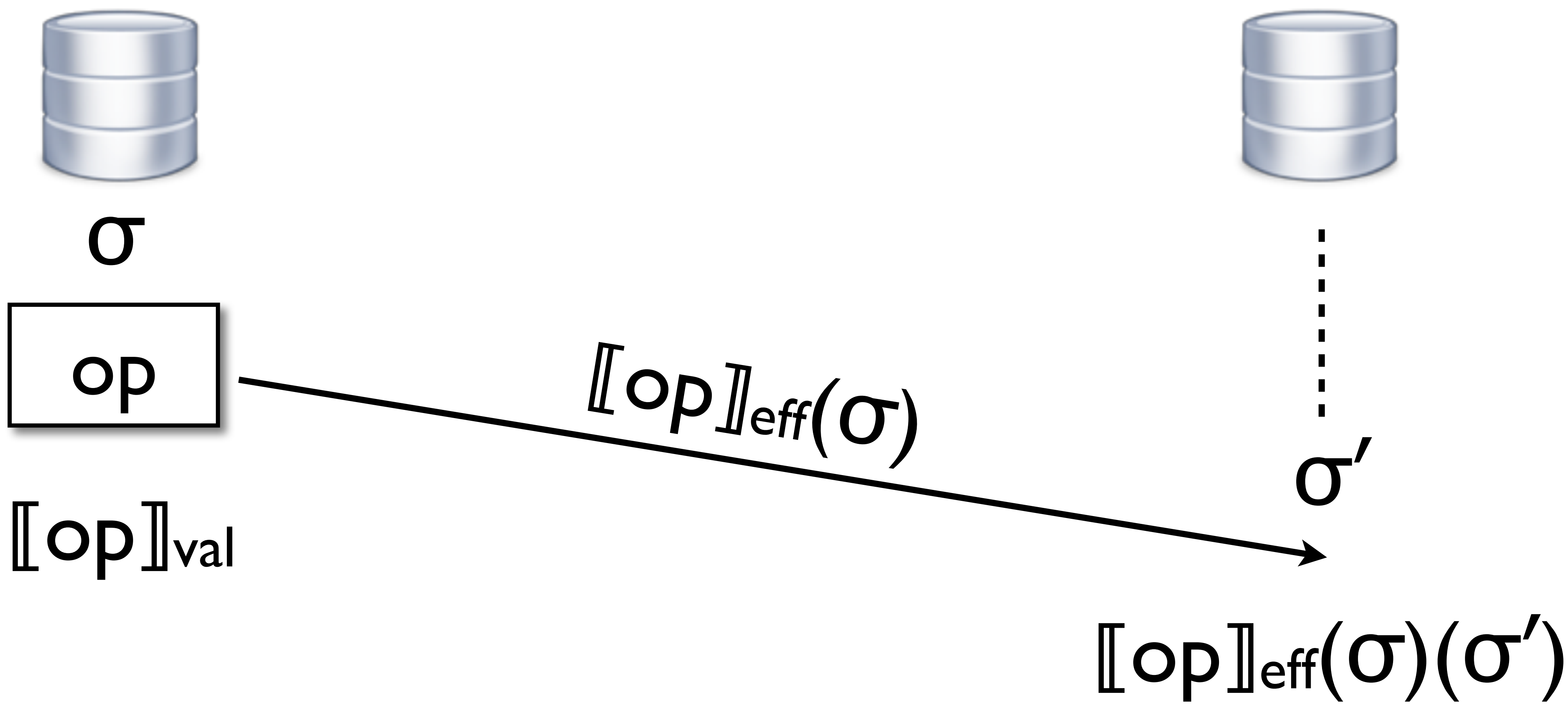
Operation semantics



$[\![\text{withdraw}(100)]\!]_{eff}(\sigma) =$

$\quad \text{if } \sigma \geq 100 \text{ then } (\lambda\sigma'.\, \sigma' - 100) \text{ else } (\lambda\sigma'.\, \sigma')$

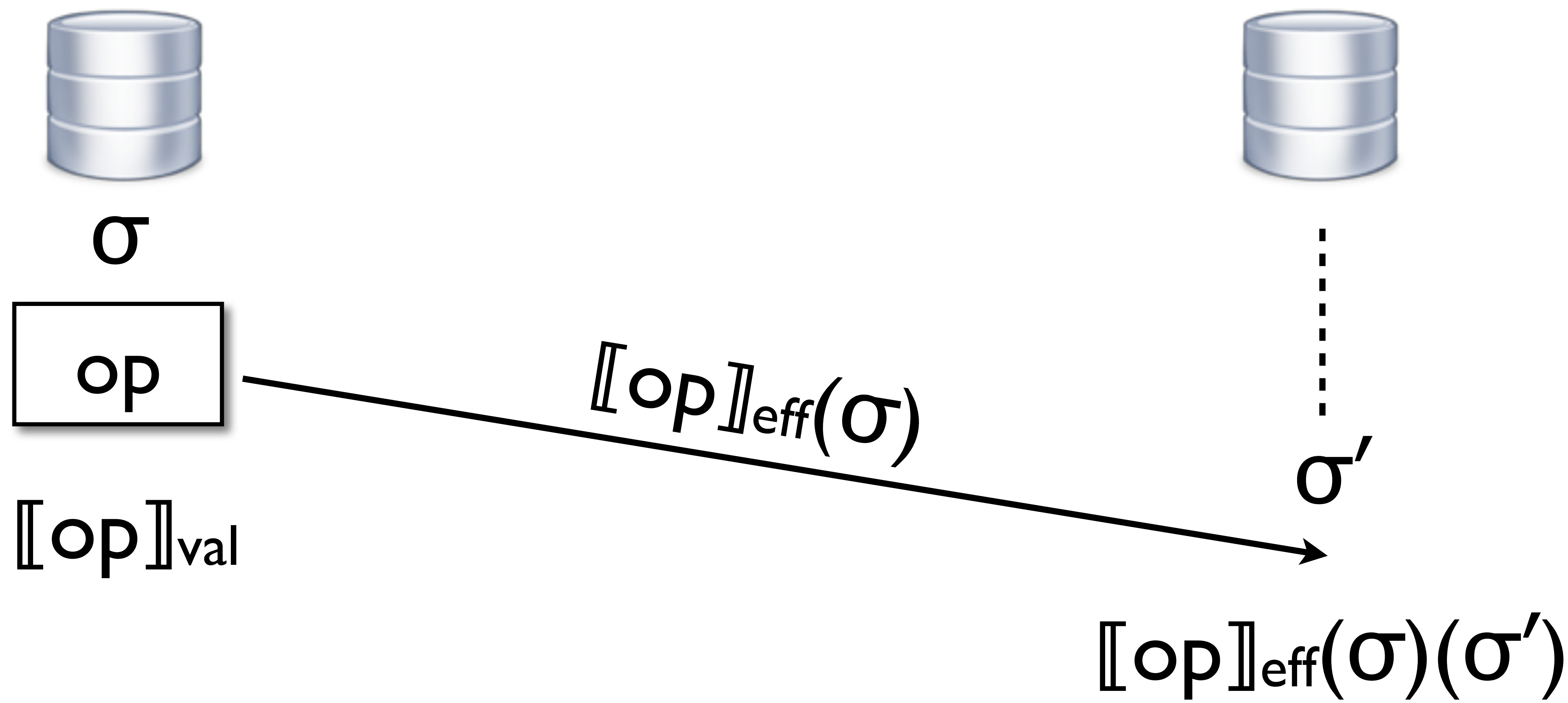
Operation semantics



$[\![withdraw(100)]\!]_{eff}(\sigma)$ =

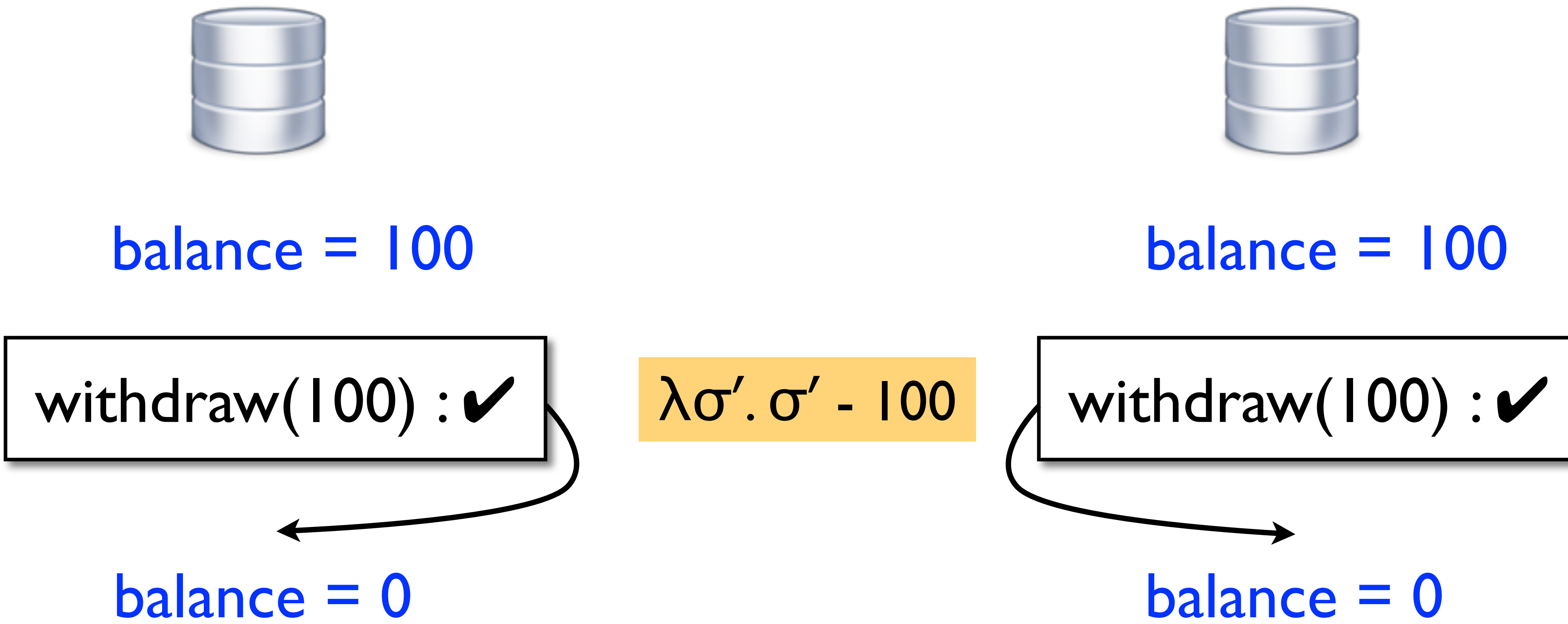
if $\sigma \geq 100$ then $(\lambda\sigma'.\ \sigma' - 100)$ else $(\lambda\sigma'.\ \sigma')$

Operation semantics



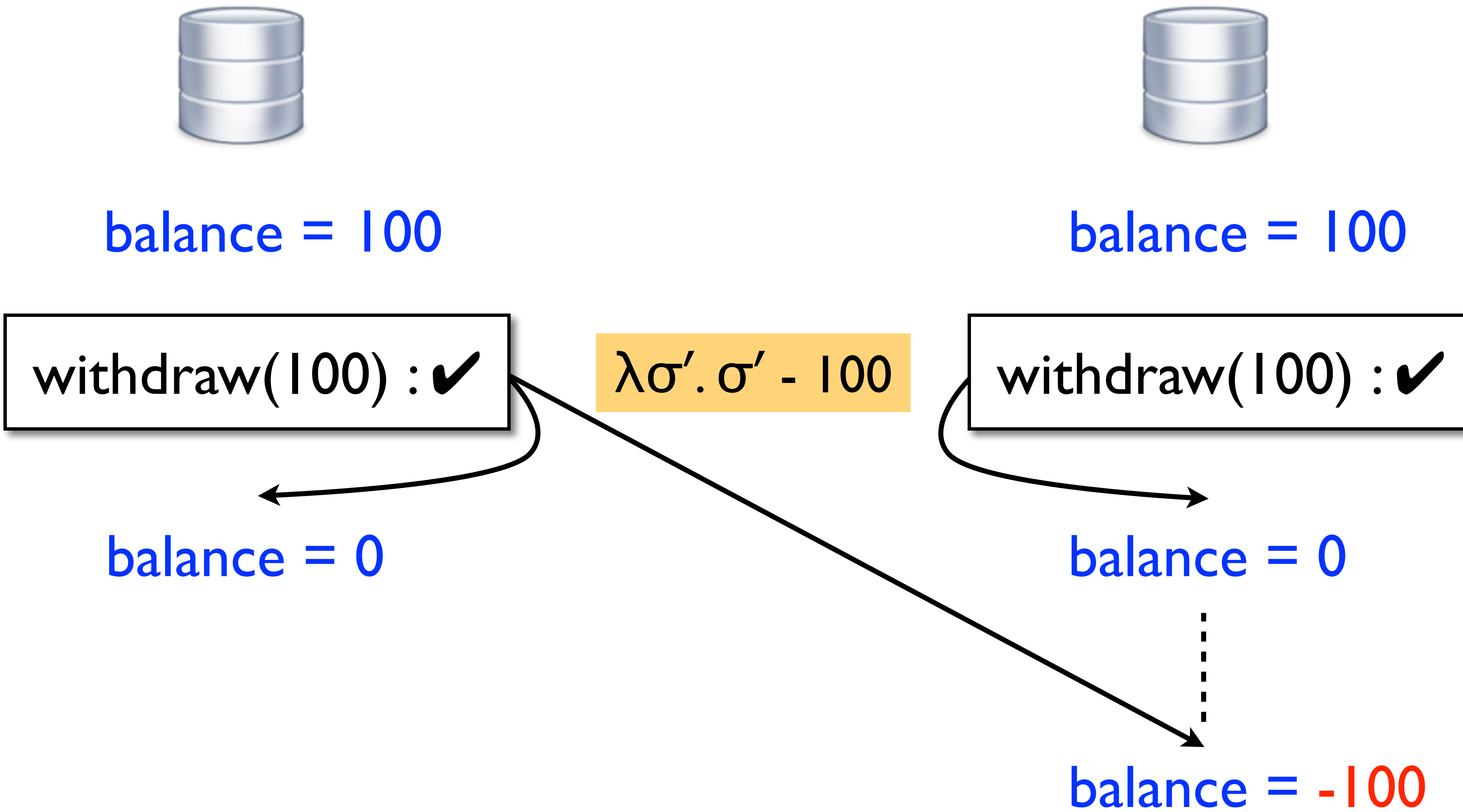
$$[\![\text{withdraw}(100)]\!]_{eff}(\sigma) \;=$$

$$\text{if } \sigma \geq 100 \text{ then } (\lambda\sigma'.\,\sigma' - 100) \text{ else } (\lambda\sigma'.\,\sigma')$$



$$[\![\text{withdraw}(100)]\!]_{\text{eff}}(\sigma) \;=$$

$$\text{if } \sigma \geq 100 \text{ then } (\lambda\sigma'.\, \sigma' - 100) \text{ else } (\lambda\sigma'.\, \sigma')$$



$$[\![\text{withdraw}(100)]\!]_{\text{eff}}(\sigma) \;=\;$$

$$\text{if } \sigma \geq 100 \text{ then } (\lambda\sigma'.\,\sigma' - 100) \text{ else } (\lambda\sigma'.\,\sigma')$$

Strengthening consistency

Token system ≈ locks on steroids:

- Token = $\{\tau_1, \tau_2, \ldots\}$
- Symmetric conflict relation $\bowtie \subseteq \text{Token} \times \text{Token}$

Strengthening consistency

Token system ≈ locks on steroids:

- Token = $\{\tau_1, \tau_2, \ldots\}$
- Symmetric conflict relation $\bowtie \subseteq$ Token × Token

Example - mutual exclusion lock:

Token = $\{\tau\}$; $\tau \bowtie \tau$

Strengthening consistency

Token system ≈ locks on steroids:

- $\mathsf{Token} = \{\tau_1, \tau_2, \ldots\}$
- Symmetric conflict relation $\bowtie \subseteq \mathsf{Token} \times \mathsf{Token}$

Example - mutual exclusion lock:

$\mathsf{Token} = \{\tau\};\ \tau \bowtie \tau$

Each operation associated with a set of tokens:

$[\![\mathsf{op}]\!]_{\mathsf{tok}} \in \mathsf{State} \rightarrow \mathcal{P}(\mathsf{Token})$

Operations associated with conflicting tokens cannot be unaware of each other

$\tau \bowtie \tau$

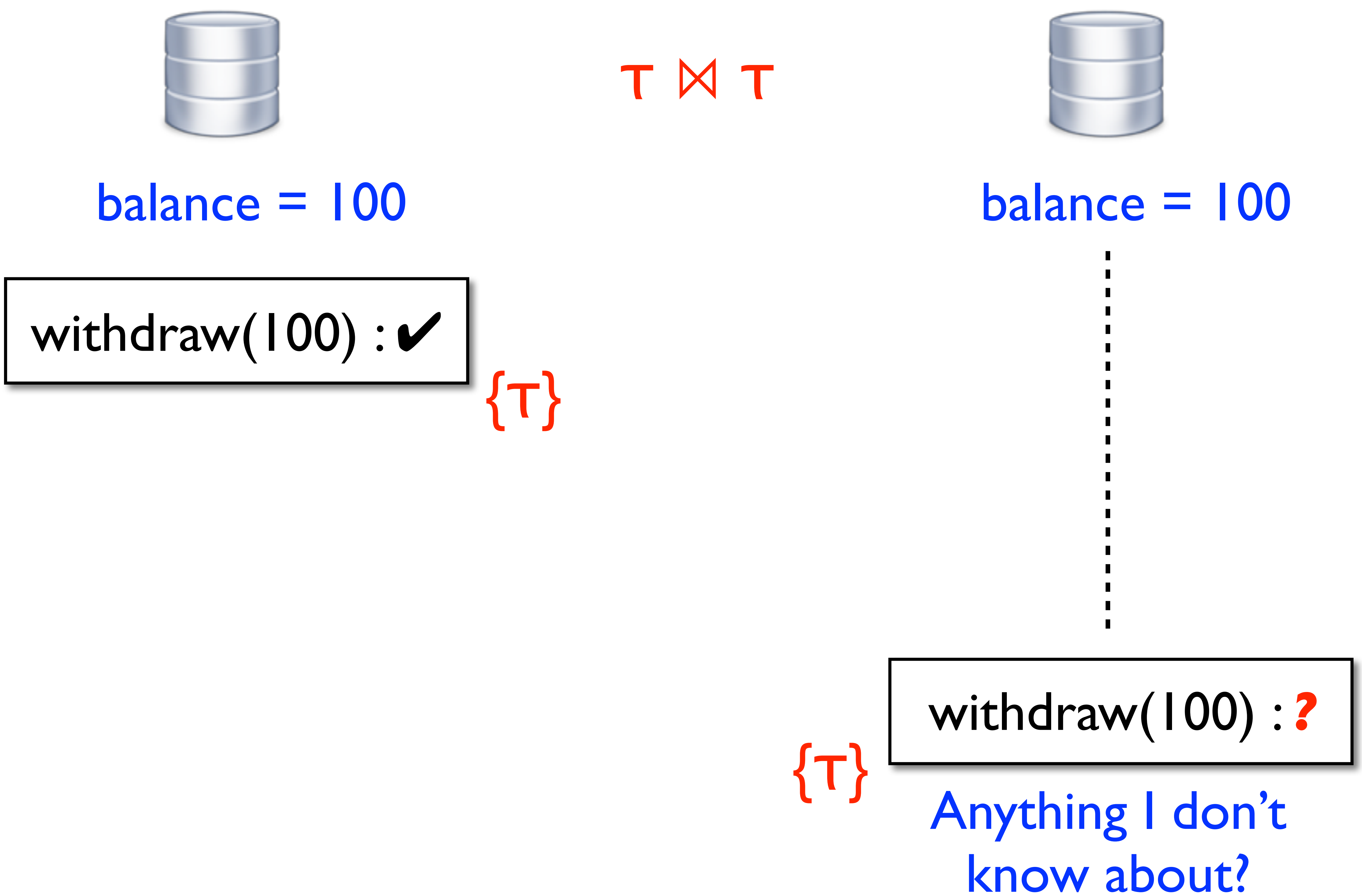
balance = 100

balance = 100

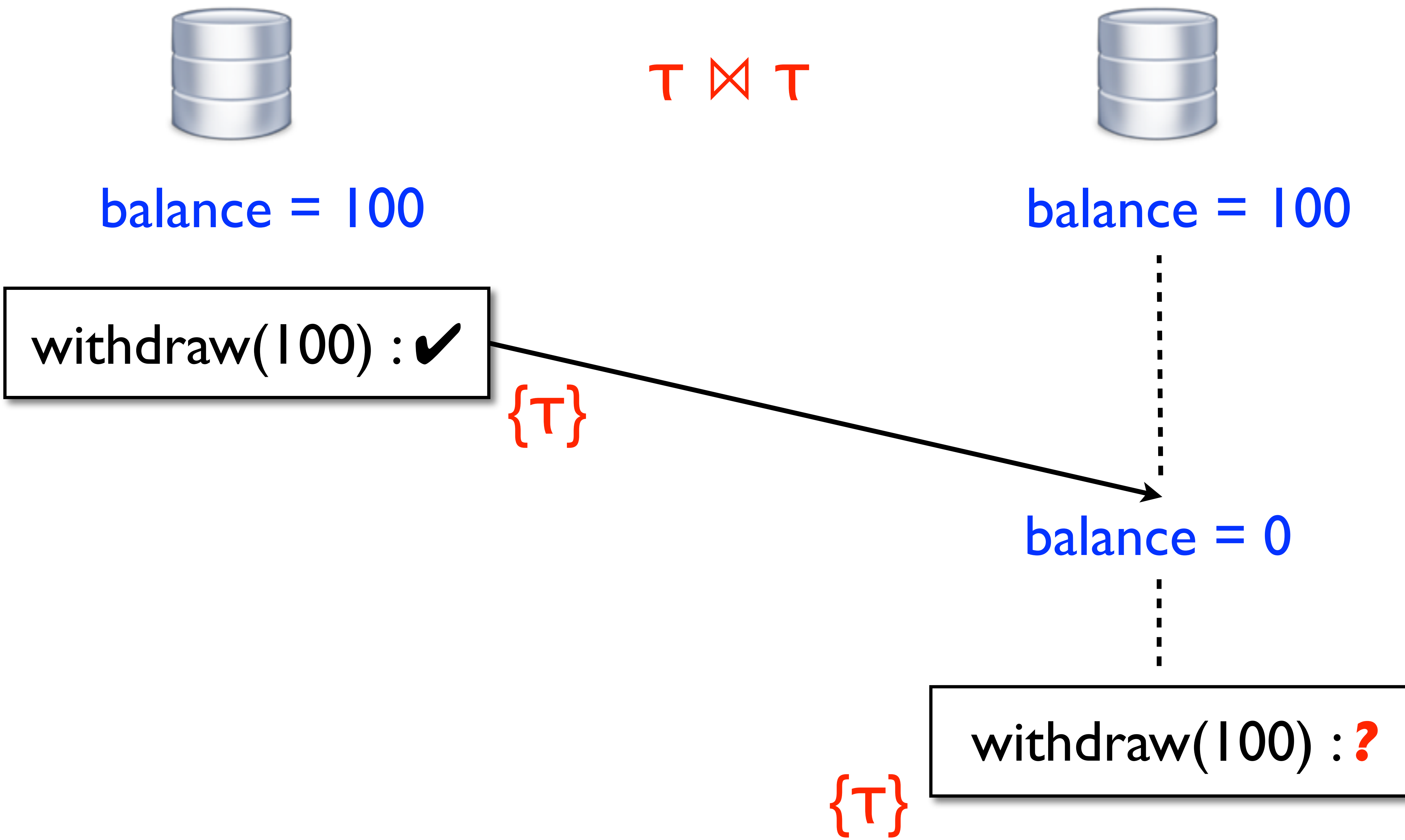
withdraw(100) : ✔

$\{\tau\}$

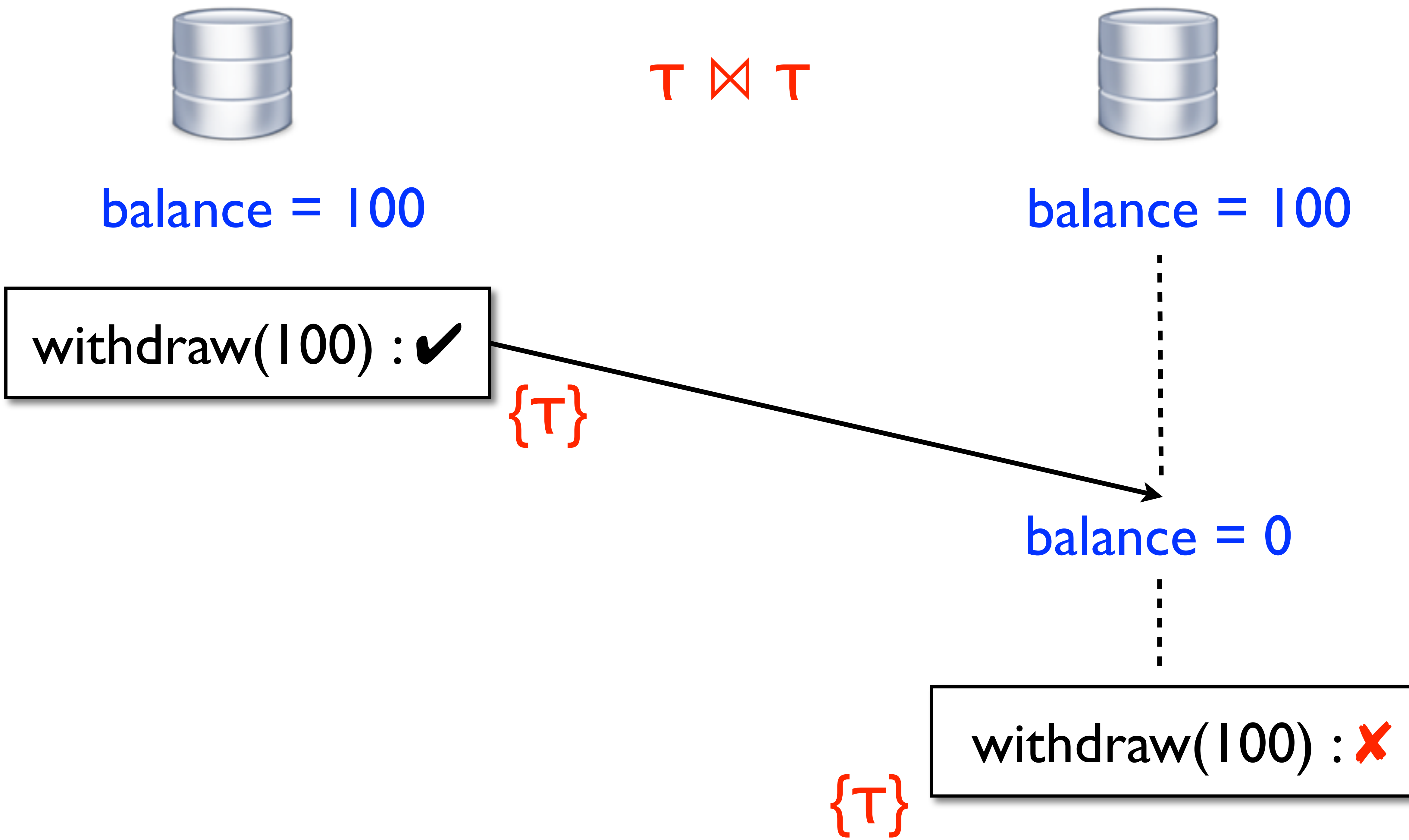
Operations associated with conflicting tokens cannot be unaware of each other



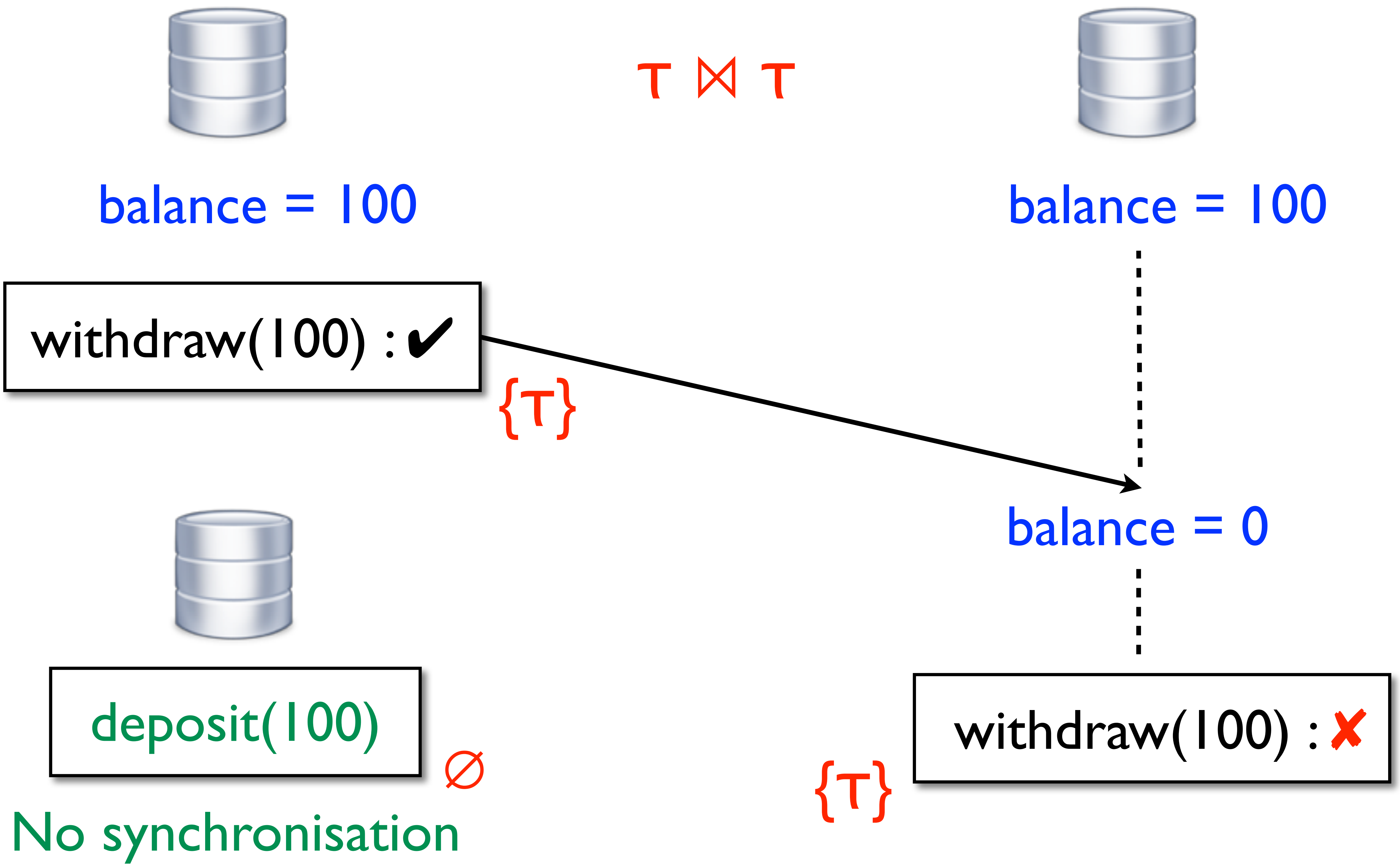
Operations associated with conflicting tokens cannot be unaware of each other



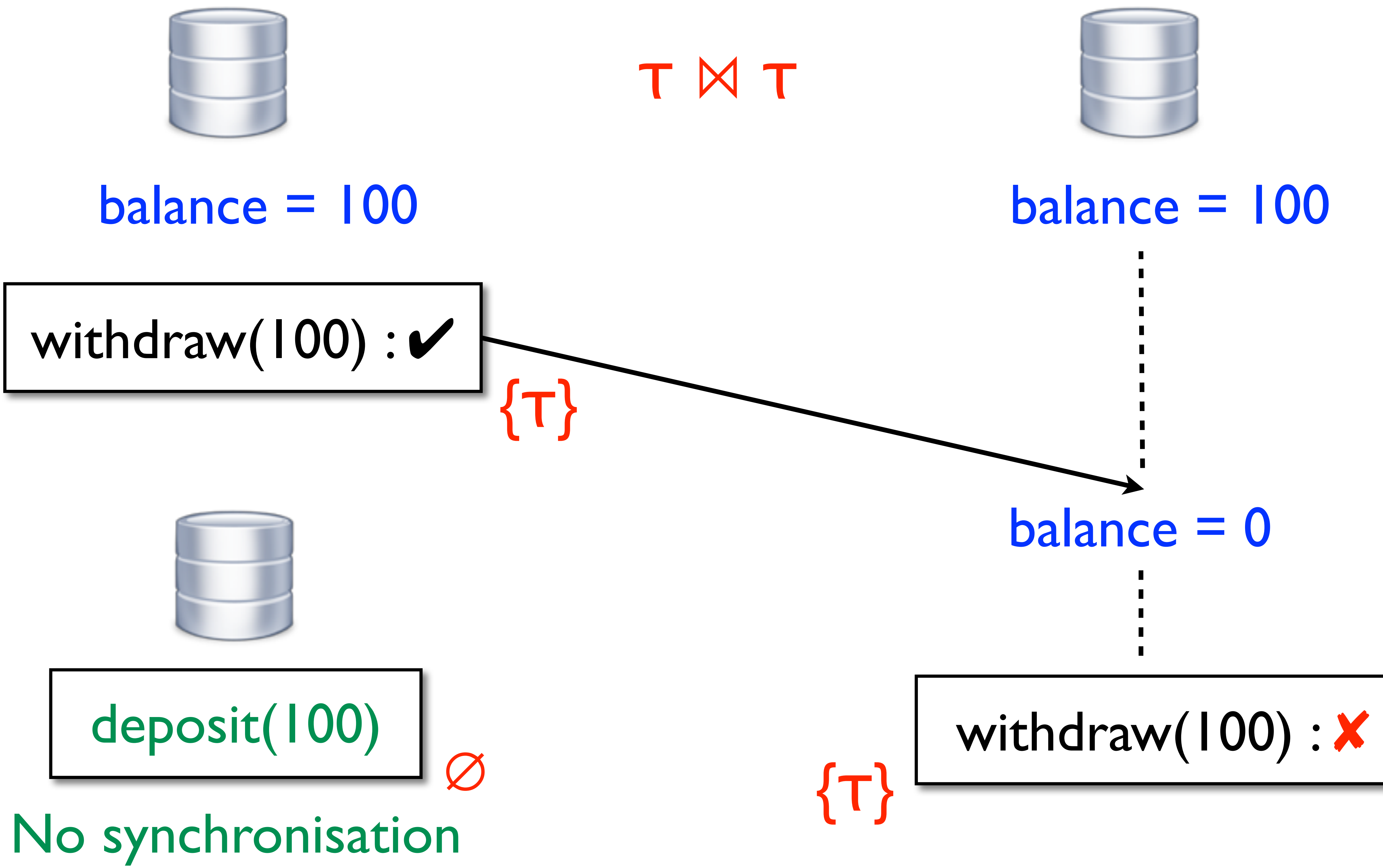
Operations associated with conflicting tokens cannot be unaware of each other



Operations associated with conflicting tokens cannot be unaware of each other



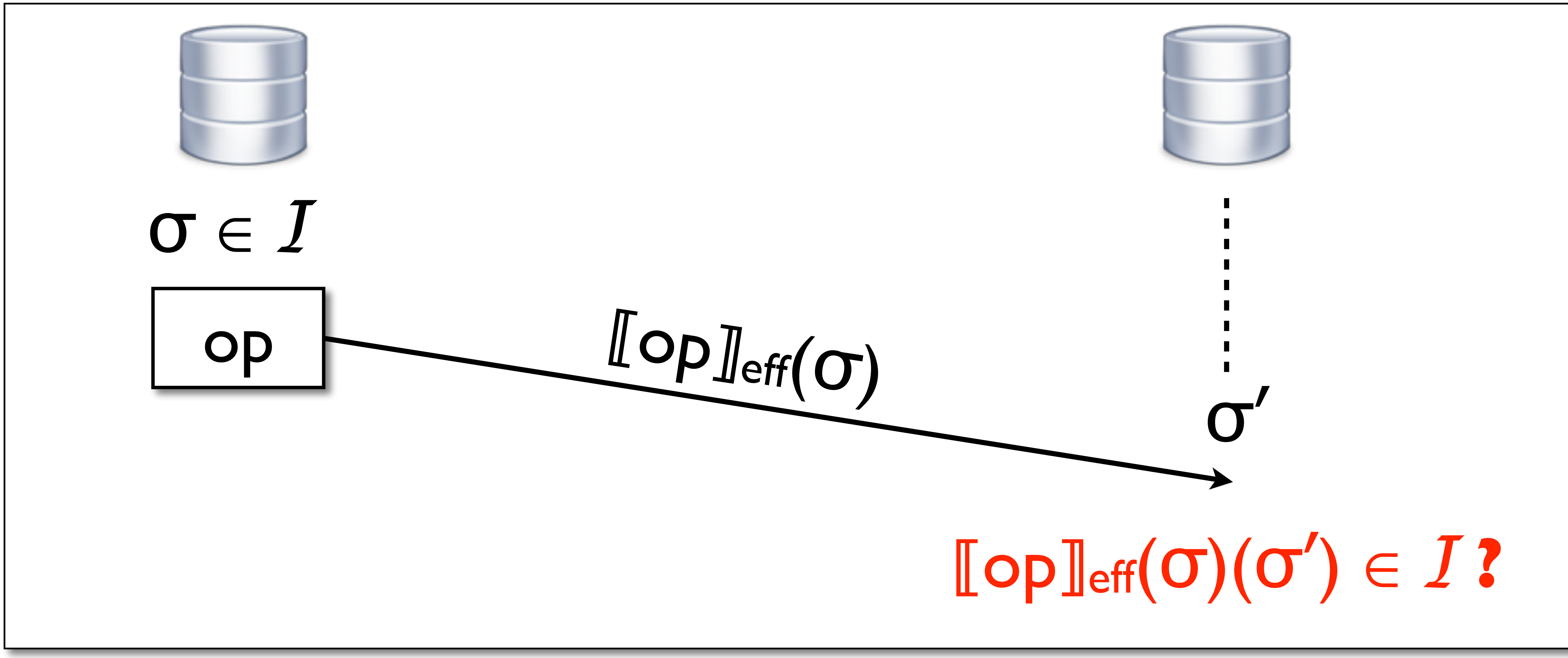
Do we always have I = (balance ≥ 0)?
Rely-guarantee-based proof rule



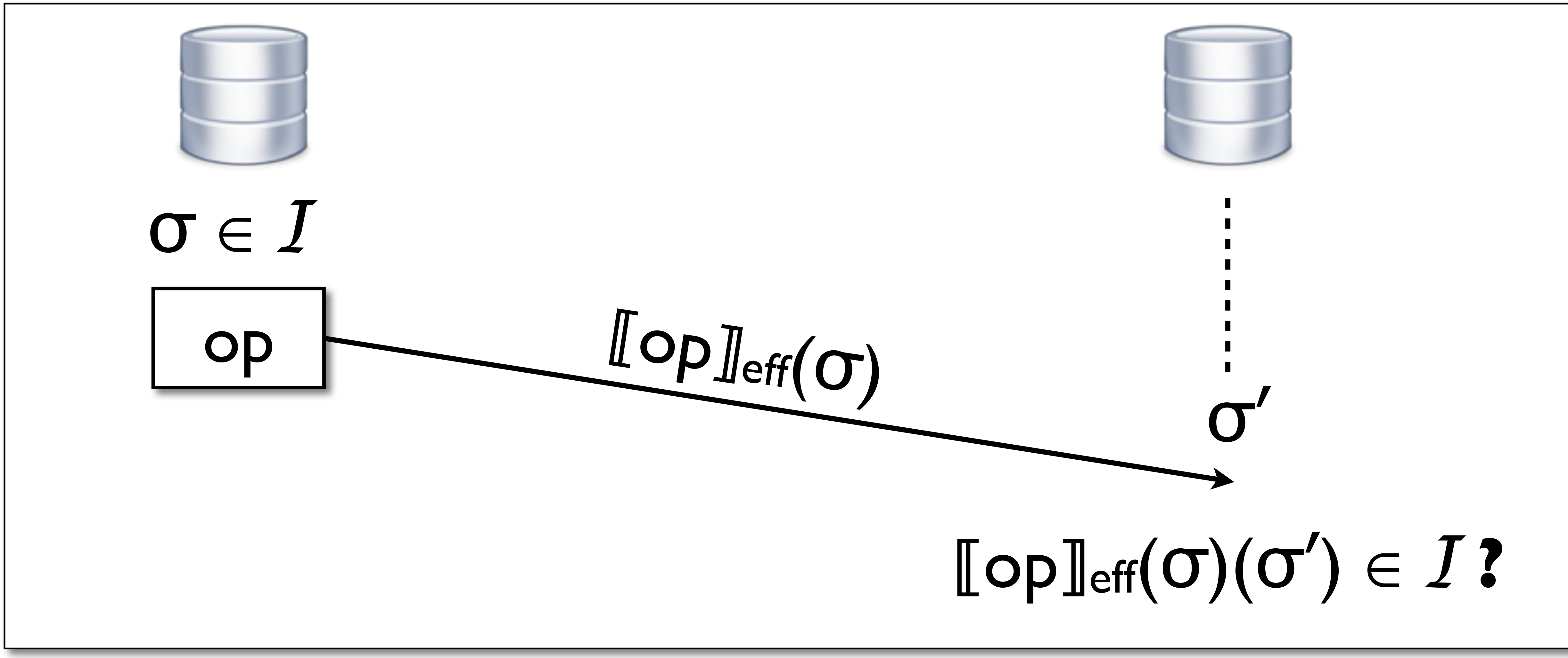
$\sigma \in \mathcal{I}$ ← Assume invariant holds

$\boxed{\text{op}}$

← Check it's preserved after executing op



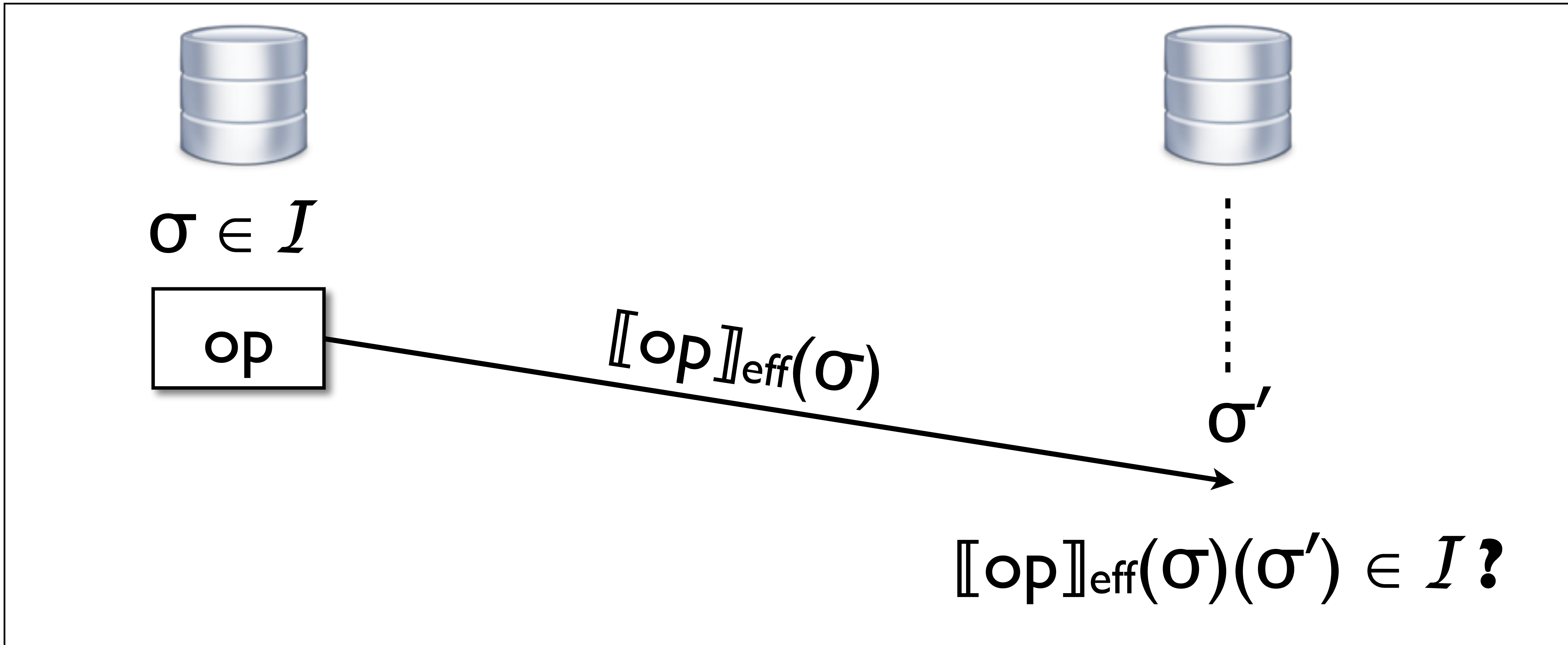
Effect applied in a different state!



$[\![\mathrm{op}]\!]_{\mathrm{eff}}(\sigma)$ = if $P(\sigma)$ then $f(\sigma)$ else if...

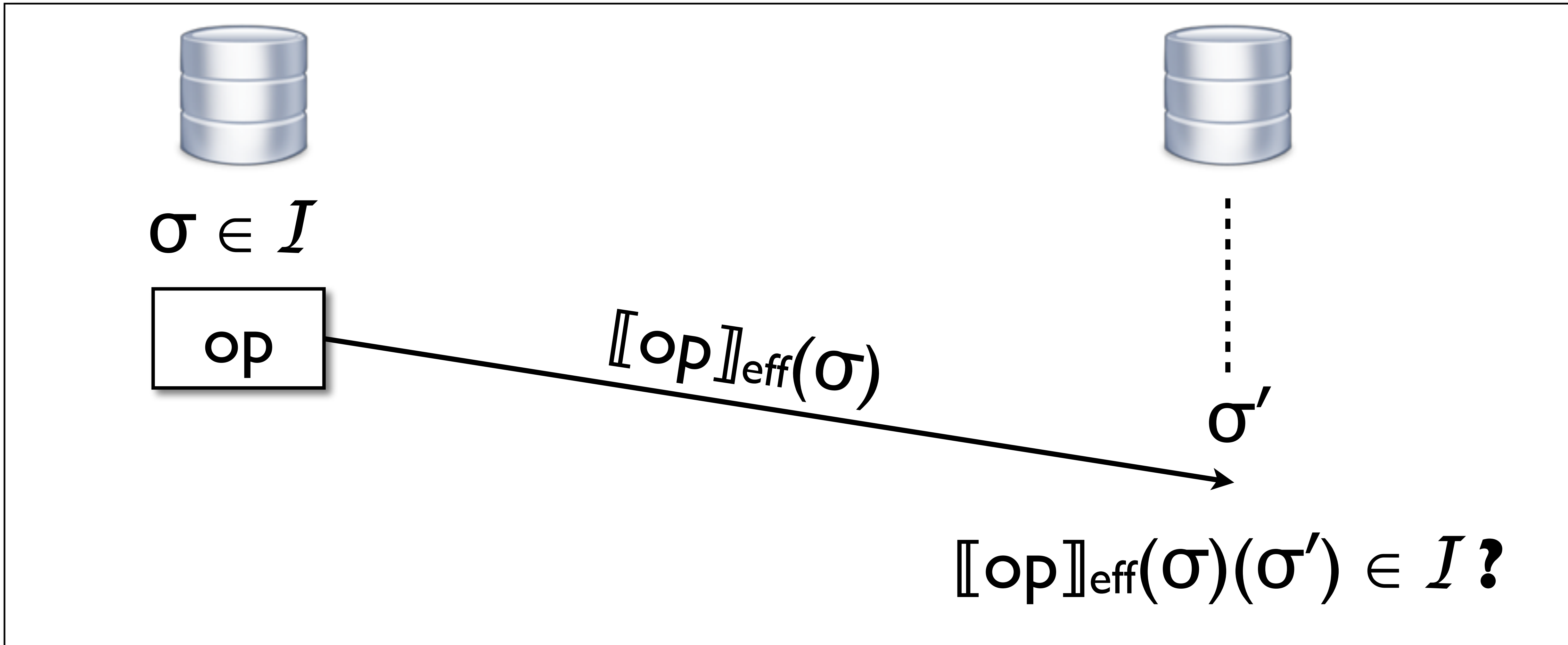
$[\![\mathrm{withdraw}(100)]\!]_{\mathrm{eff}}(\sigma)$ =

if $\sigma \geq 100$ then $(\lambda\sigma'.\,\sigma' - 100)$ else $(\lambda\sigma'.\,\sigma')$



$[\![\mathrm{op}]\!]_{\mathrm{eff}}(\sigma)$ = if $P(\sigma)$ then $f(\sigma)$ else if…

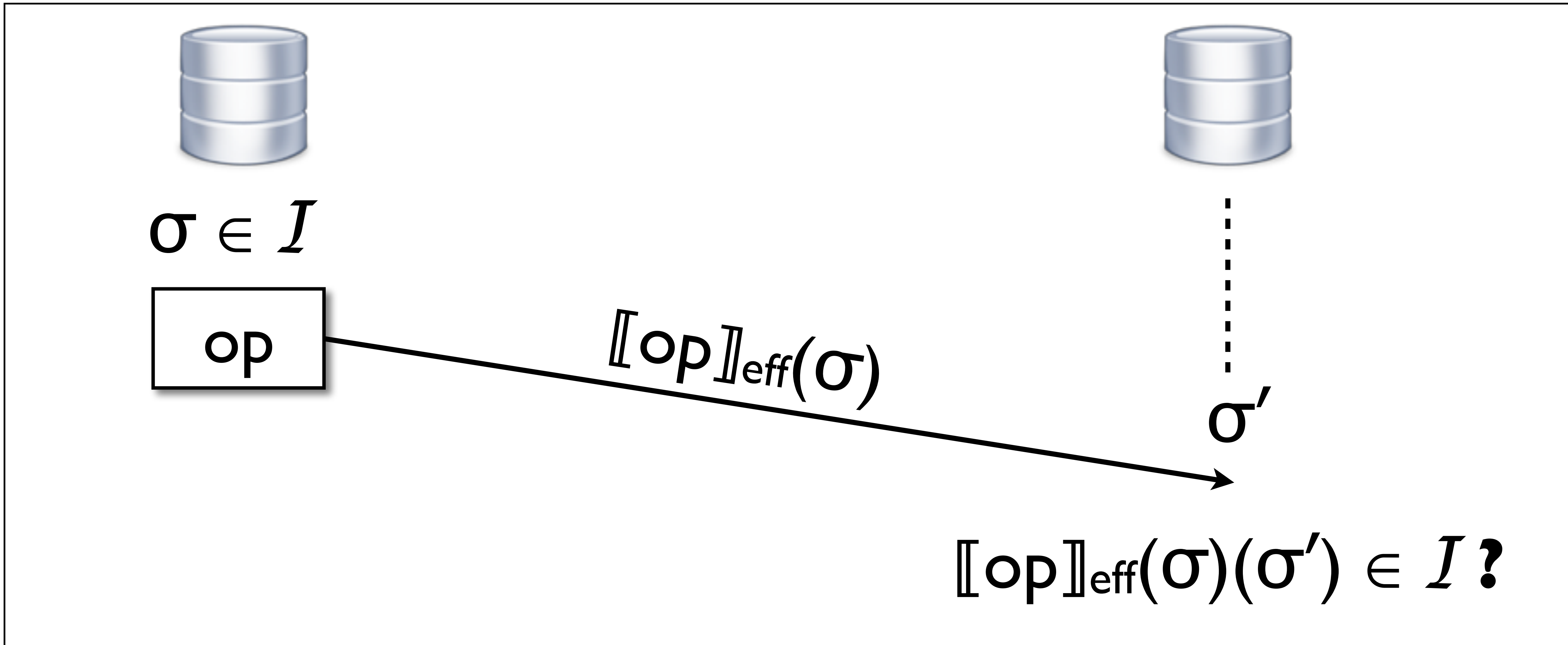
1. Effector safety: f preserves $\mathcal{I}$ when executed in any state satisfying P: $\{\mathcal{I} \wedge P\}\ f\ \{\mathcal{I}\}$



$[\![\mathrm{op}]\!]_{\mathrm{eff}}(\sigma)$ = if $P(\sigma)$ then $f(\sigma)$ else if...

1. Effector safety: f preserves $\mathcal{I}$ when executed in any state satisfying P: $\{\mathcal{I} \wedge P\}\ f\ \{\mathcal{I}\}$

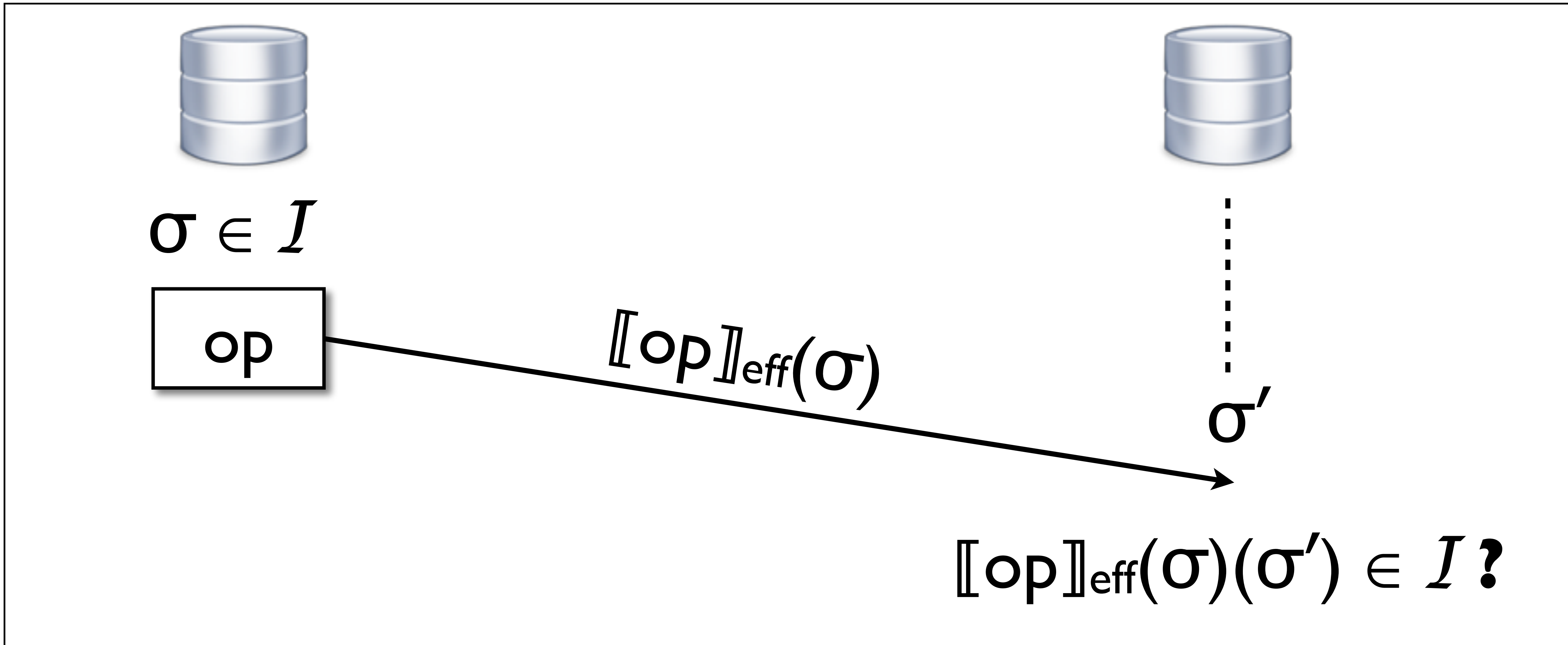
$\{bal \geq 0 \wedge bal \geq 100\}$ bal := bal-100 $\{bal \geq 0\}$



$[\![\text{op}]\!]_{\text{eff}}(\sigma)$ = if $P(\sigma)$ then $f(\sigma)$ else if...

1. Effector safety: f preserves $\mathcal{I}$ when executed in any state satisfying P: $\{\mathcal{I} \wedge P\}\ f\ \{\mathcal{I}\}$

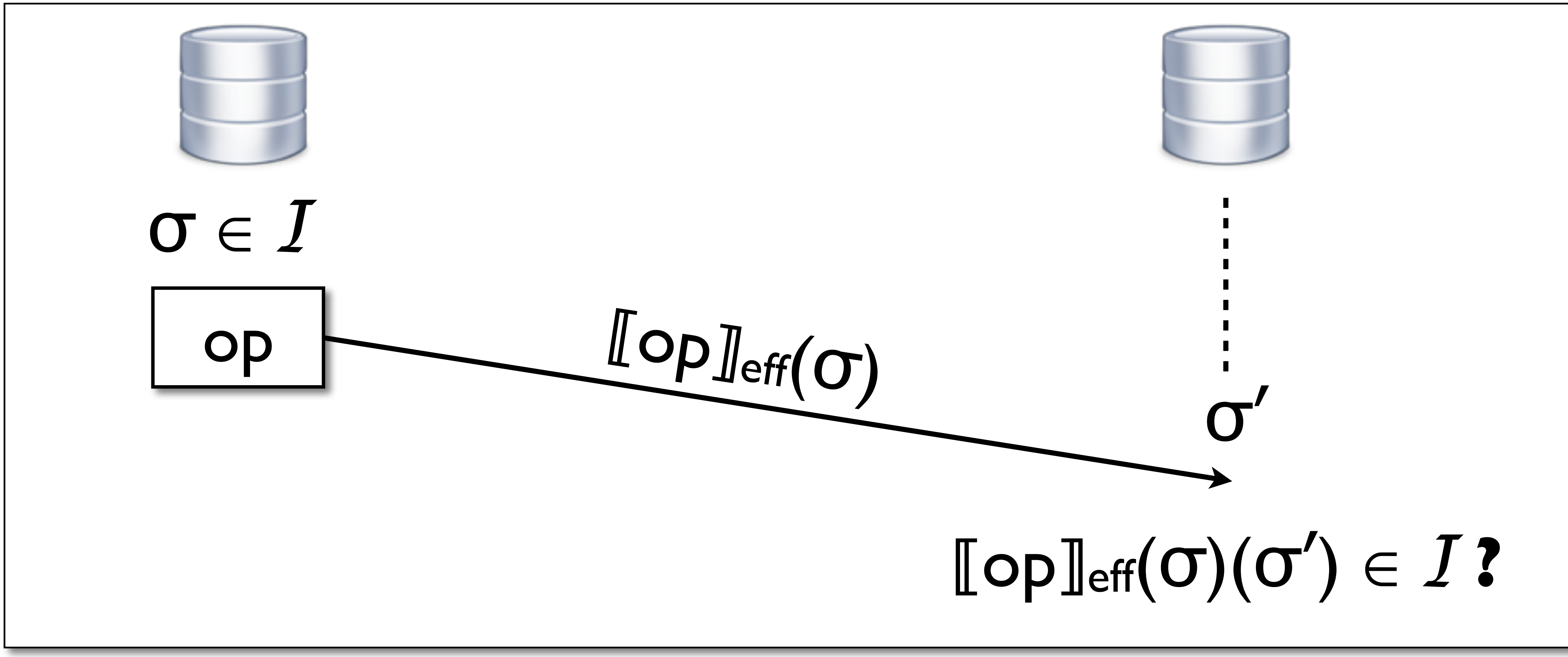
$\{\text{bal} \geq 0 \wedge \text{bal} \geq 100\}$ bal := bal-100 $\{\text{bal} \geq 0\}$



$[\![\mathrm{op}]\!]_{\mathrm{eff}}(\sigma)$ = if $P(\sigma)$ then $f(\sigma)$ else if…

1. Effector safety: f preserves $\mathcal{I}$ when executed in any state satisfying P: $\{\mathcal{I} \wedge P\}\ f\ \{\mathcal{I}\}$

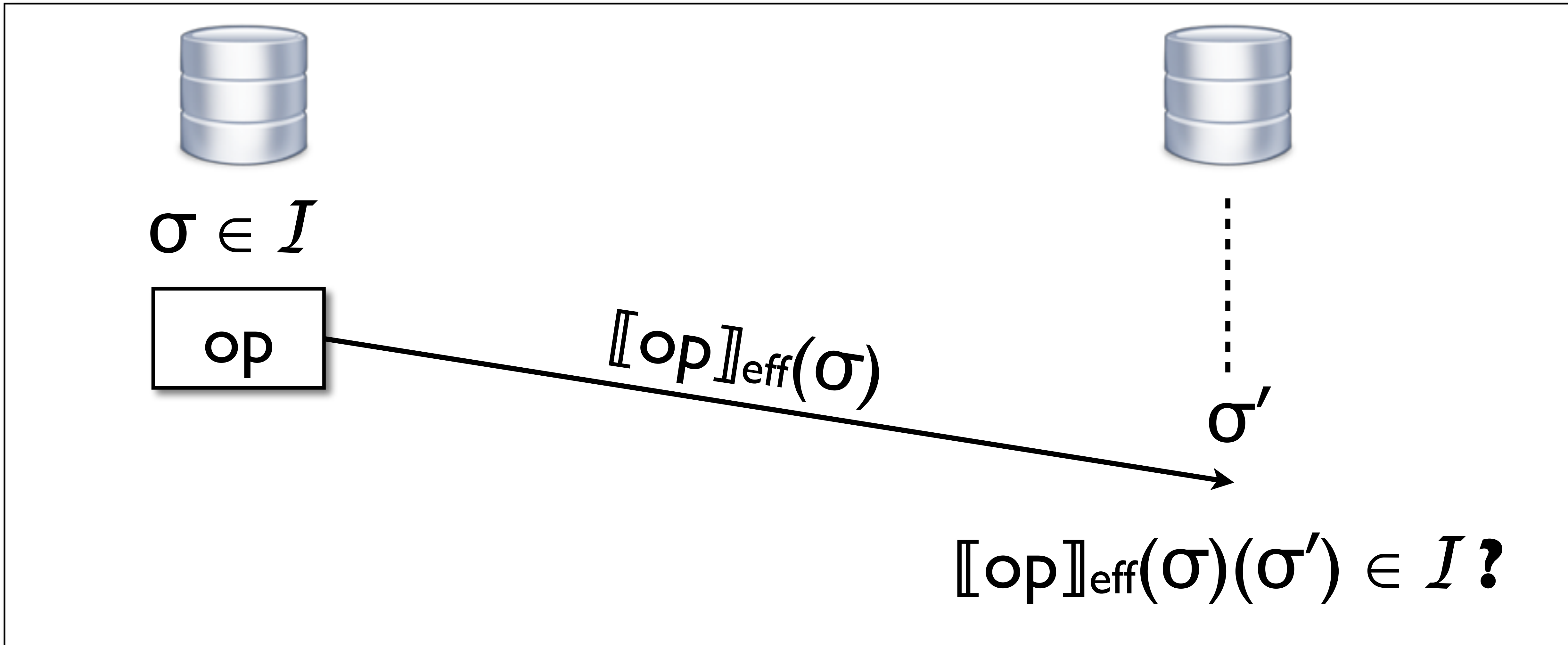
$$\{bal \geq 0 \wedge bal \geq 100\}\ \ bal := bal\text{-}100\ \ \{bal \geq 0\}$$



$[\![\mathrm{op}]\!]_{\mathrm{eff}}(\sigma)$ = if $P(\sigma)$ then $f(\sigma)$ else if...

1. Effector safety: f preserves $\mathcal{I}$ when executed in any state satisfying P: $\{\mathcal{I} \wedge P\}\ f\ \{\mathcal{I}\}$

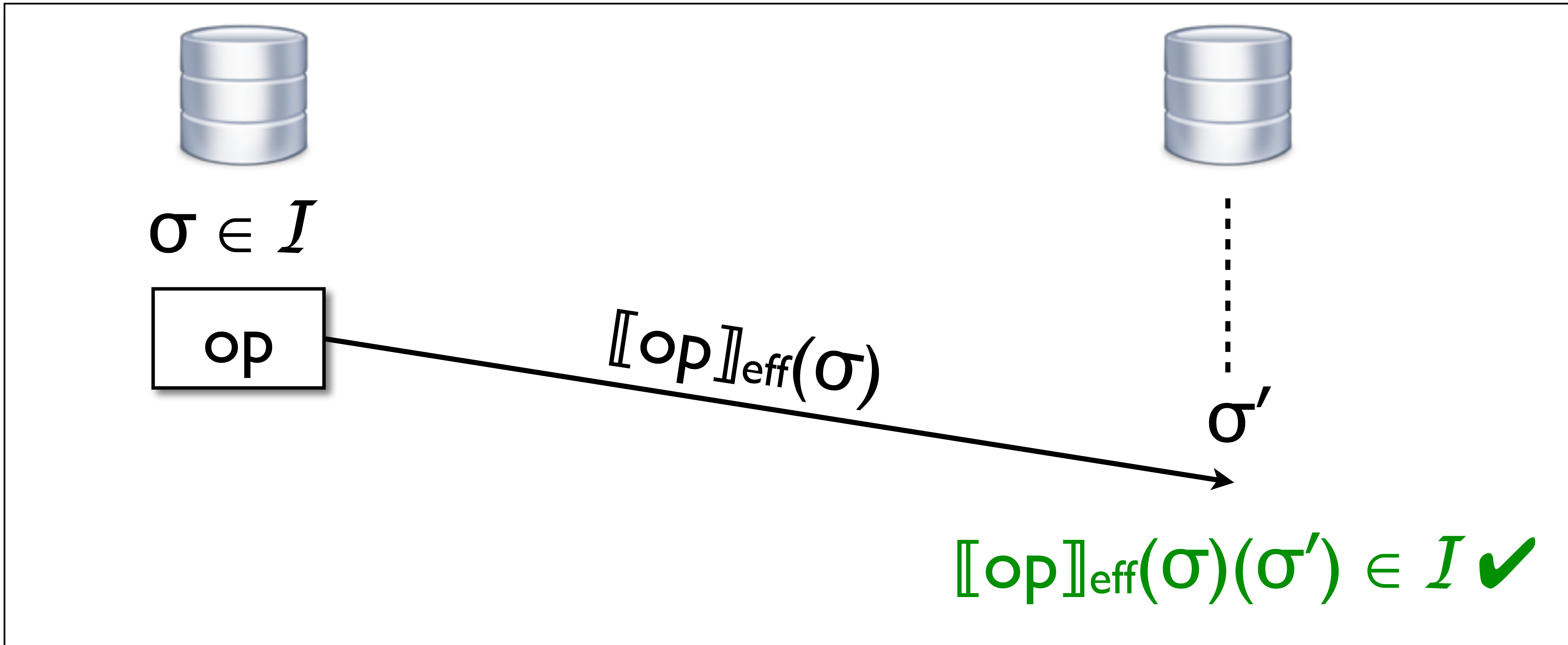
$\{bal \geq 0 \wedge bal \geq 100\}$ bal := bal-100 $\{bal \geq 0\}$



$[\![op]\!]_{eff}(\sigma)$ = if $P(\sigma)$ then $f(\sigma)$ else if...

1. Effector safety: f preserves $\mathcal{I}$ when executed in any state satisfying P: $\{\mathcal{I} \wedge P\}\ f\ \{\mathcal{I}\}$

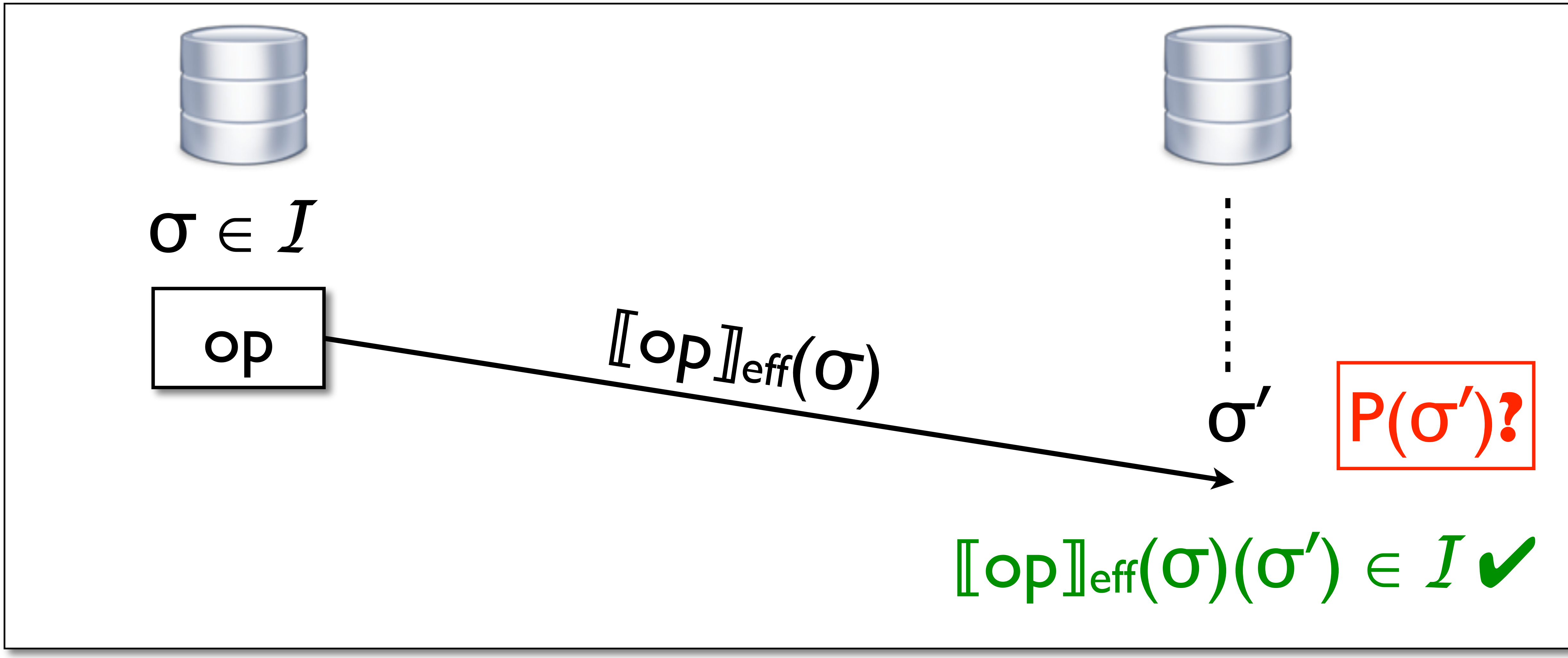
$$\{bal \geq 0 \wedge bal \geq 100\}\ \ bal := bal\text{-}100\ \ \{bal \geq 0\}$$



$[\![op]\!]_{eff}(\sigma)$ = if $P(\sigma)$ then $f(\sigma)$ else if...

1. Effector safety: f preserves $\mathcal{I}$ when executed in any state satisfying P: $\{\mathcal{I} \wedge P\}\ f\ \{\mathcal{I}\}$

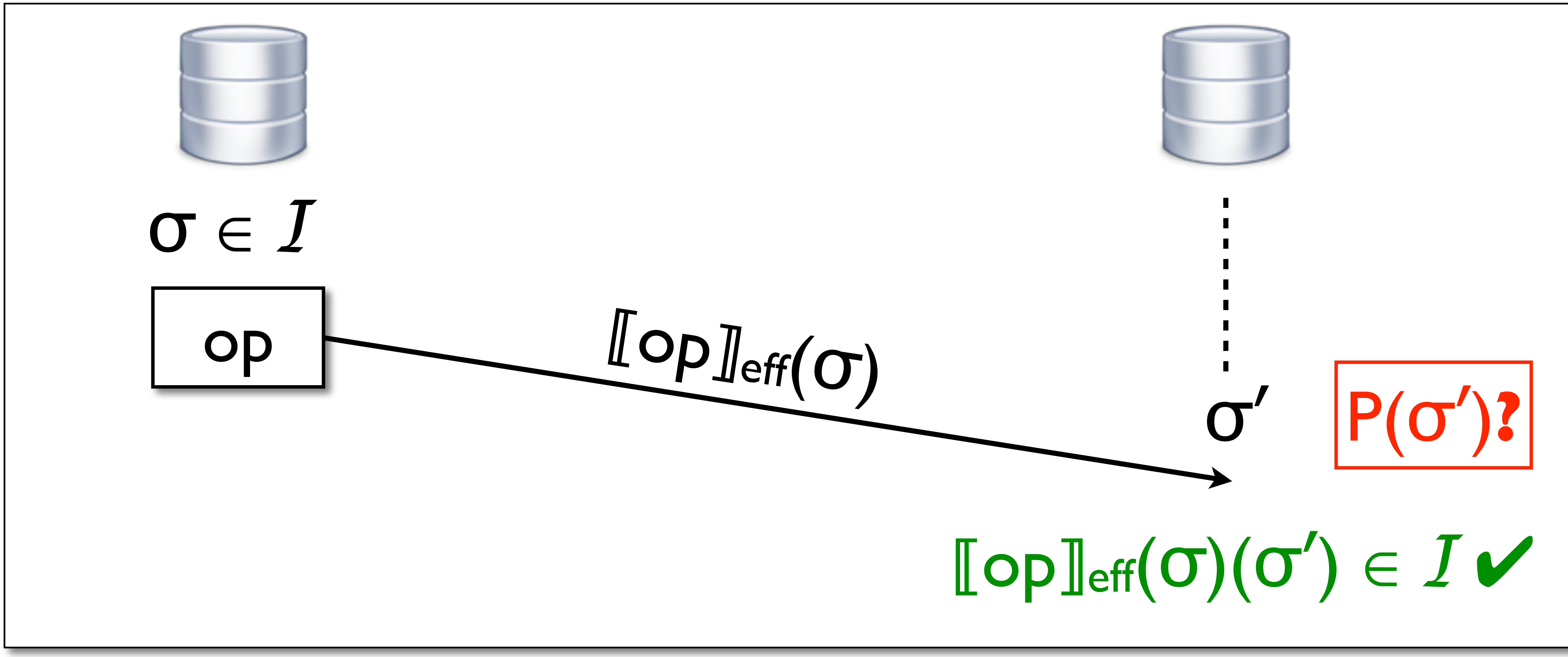
$$\{bal \geq 0 \wedge bal \geq 100\}\ \ bal := bal\text{-}100\ \ \{bal \geq 0\}$$



$[\![op]\!]_{eff}(\sigma)$ = if P(σ) then f(σ) else if…

1. Effector safety: f preserves $\mathcal{I}$ when executed in any state satisfying P: $\{\mathcal{I} \wedge P\}\ f\ \{\mathcal{I}\}$

$$\{bal \geq 0 \wedge bal \geq 100\}\ \ bal := bal\text{-}100\ \ \{bal \geq 0\}$$



$[\![op]\!]_{eff}(\sigma)$ = if $P(\sigma)$ then $f(\sigma)$ else if...

1. **Effector safety:** f preserves $\mathcal{I}$ when executed in any state satisfying P: $\{\mathcal{I} \wedge P\}\ f\ \{\mathcal{I}\}$

2. **Precondition stability:** P will hold when f is applied at any replica

$[\![op]\!]_{eff}(\sigma)$ = if $P(\sigma)$ then $f(\sigma)$ else if…

$[\![op]\!]_{tok}(\sigma)$ = if $P(\sigma)$ then T else if…

P is preserved by any effector f' of any operation that is not associated with a token conflicting with T: $\{P\}\ f'\ \{P\}$

$[\![op]\!]_{eff}(\sigma)$ = if $P(\sigma)$ then $f(\sigma)$ else if...

$[\![op]\!]_{tok}(\sigma)$ = if $P(\sigma)$ then T else if...

P is preserved by any effector f' of any operation that is not associated with a token conflicting with T: $\{P\}\ f'\ \{P\}$

$[\![withdraw(100)]\!]_{tok}(\sigma) = \{\tau\}$ $\tau \bowtie \tau$

$[\![deposit(100)]\!]_{tok}(\sigma) = \varnothing$

Check stability of withdraw's precondition against deposits:

$\{bal \geq 100\}\ bal := bal+100\ \{bal \geq 100\}$

$[\![op]\!]_{eff}(\sigma)$ = if $P(\sigma)$ then $f(\sigma)$ else if…

$[\![op]\!]_{tok}(\sigma)$ = if $P(\sigma)$ then T else if…

P is preserved by any effector f' of any operation that is not associated with a token conflicting with T: $\{P\}\ f'\ \{P\}$

$[\![withdraw(100)]\!]_{tok}(\sigma) = \{\tau\}$ $\qquad \tau \bowtie \tau$

$[\![deposit(100)]\!]_{tok}(\sigma) = \varnothing$

Check stability of withdraw's precondition against deposits:

$\{bal \geq 100\}$ $bal := bal+100$ $\{bal \geq 100\}$

$[\![op]\!]_{eff}(\sigma)$ = if $P(\sigma)$ then $f(\sigma)$ else if...

$[\![op]\!]_{tok}(\sigma)$ = if $P(\sigma)$ then T else if...

P is preserved by any effector f' of any operation that is not associated with a token conflicting with T: $\{P\}\ f'\ \{P\}$

$[\![withdraw(100)]\!]_{tok}(\sigma) = \{\tau\}$ $\qquad \tau \bowtie \tau$

$[\![deposit(100)]\!]_{tok}(\sigma) = \varnothing$

Check stability of withdraw's precondition against deposits:

$\{bal \geq 100\}\ bal := bal+100\ \{bal \geq 100\}$

$[\![op]\!]_{eff}(\sigma)$ = if $P(\sigma)$ then $f(\sigma)$ else if…

$[\![op]\!]_{tok}(\sigma)$ = if $P(\sigma)$ then T else if…

P is preserved by any effector f' of any operation that is not associated with a token conflicting with T: $\{P\}\ f'\ \{P\}$

$[\![\text{withdraw}(100)]\!]_{tok}(\sigma) = \{\tau\}$ $\qquad\qquad \tau \bowtie \tau$

$[\![\text{deposit}(100)]\!]_{tok}(\sigma) = \varnothing$

Check stability of withdraw's precondition against deposits:

$\{bal \geq 100\}\ \ bal := bal+100\ \ \{bal \geq 100\}$

Prototype tool

- Automates the proof rule
- Discharges verification conditions using SMT
- Case studies:
  - fragments of web applications
  - currently applying to a distributed file system

Conclusion

- Weak consistency poses challenges for programmability
- But pay-off often worth it: availability, cost-effectiveness
- Verification methods enable weakening consistency without compromising correctness