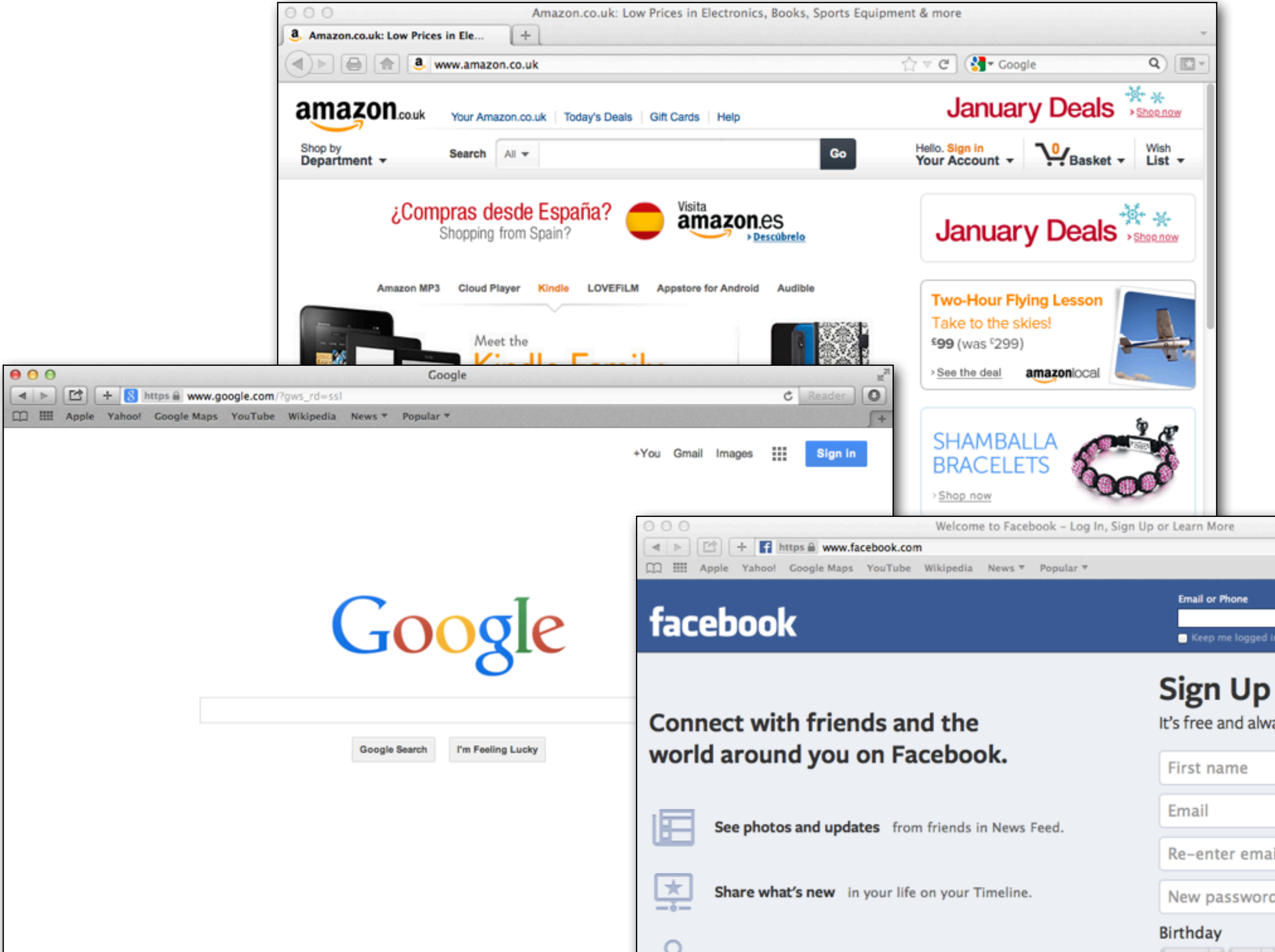
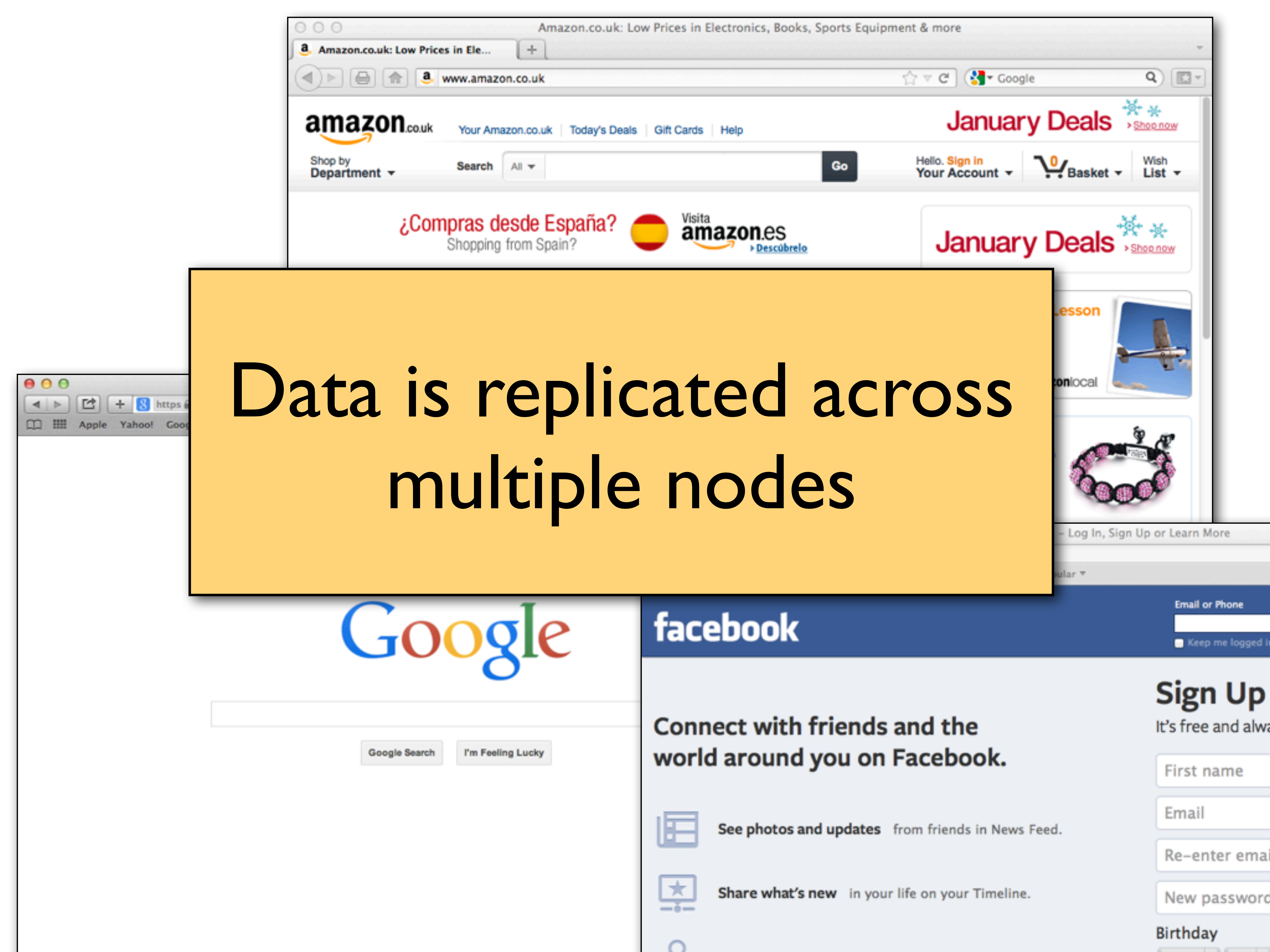
‘Cause I’m strong enough: Reasoning about consistency choices in distributed systems

Alexey Gotsman

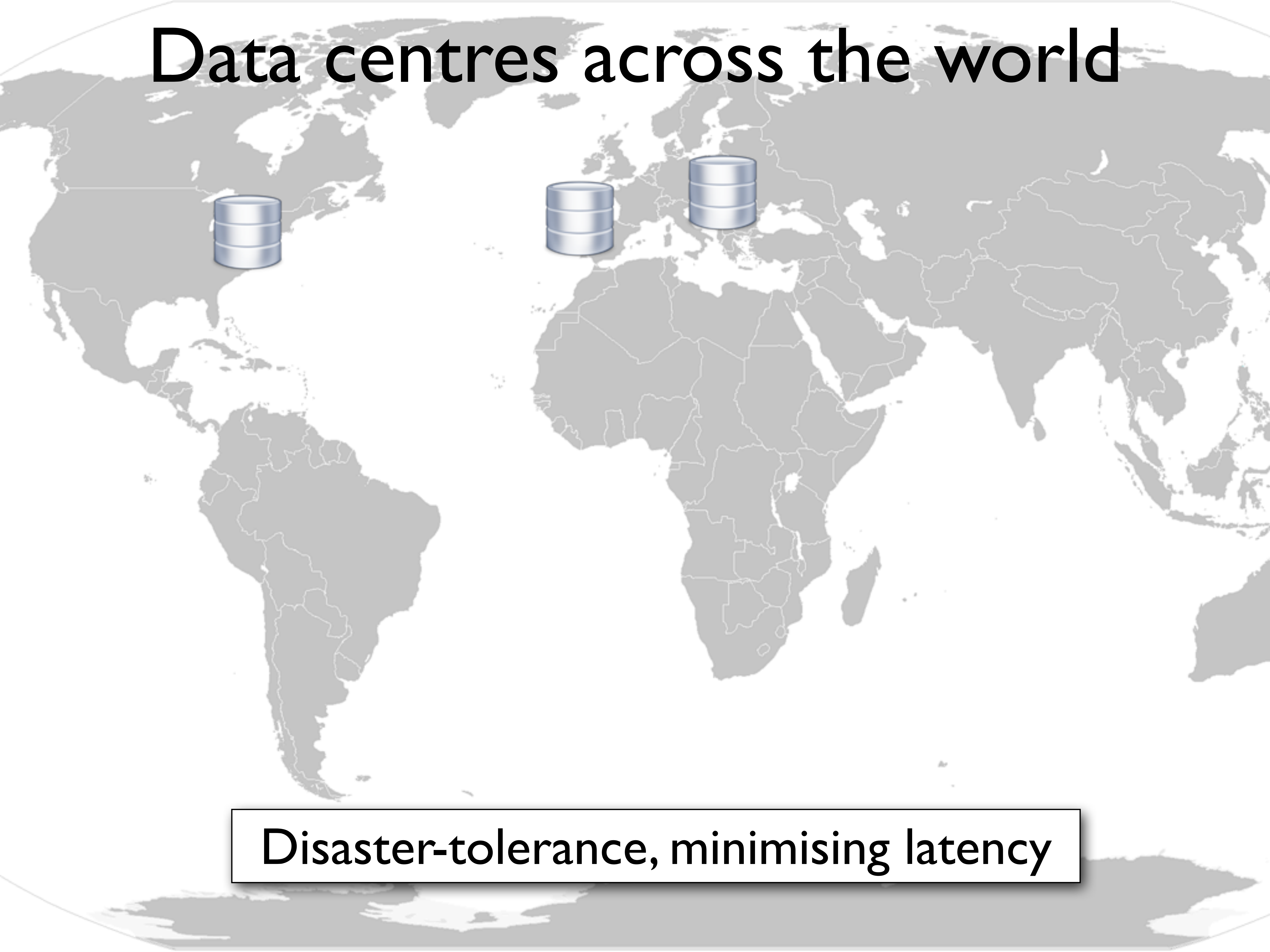
IMDEA Software Institute, Madrid, Spain

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


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
January Deals
Shop now
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
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
Sign Up
It's free and alwa
Connect with friends and the world around you on Facebook.
First name
Email
Re-enter emai
New passworc
See photos and updates from friends in News Feed.
Share what's new in your life on your Timeline.
Birthday

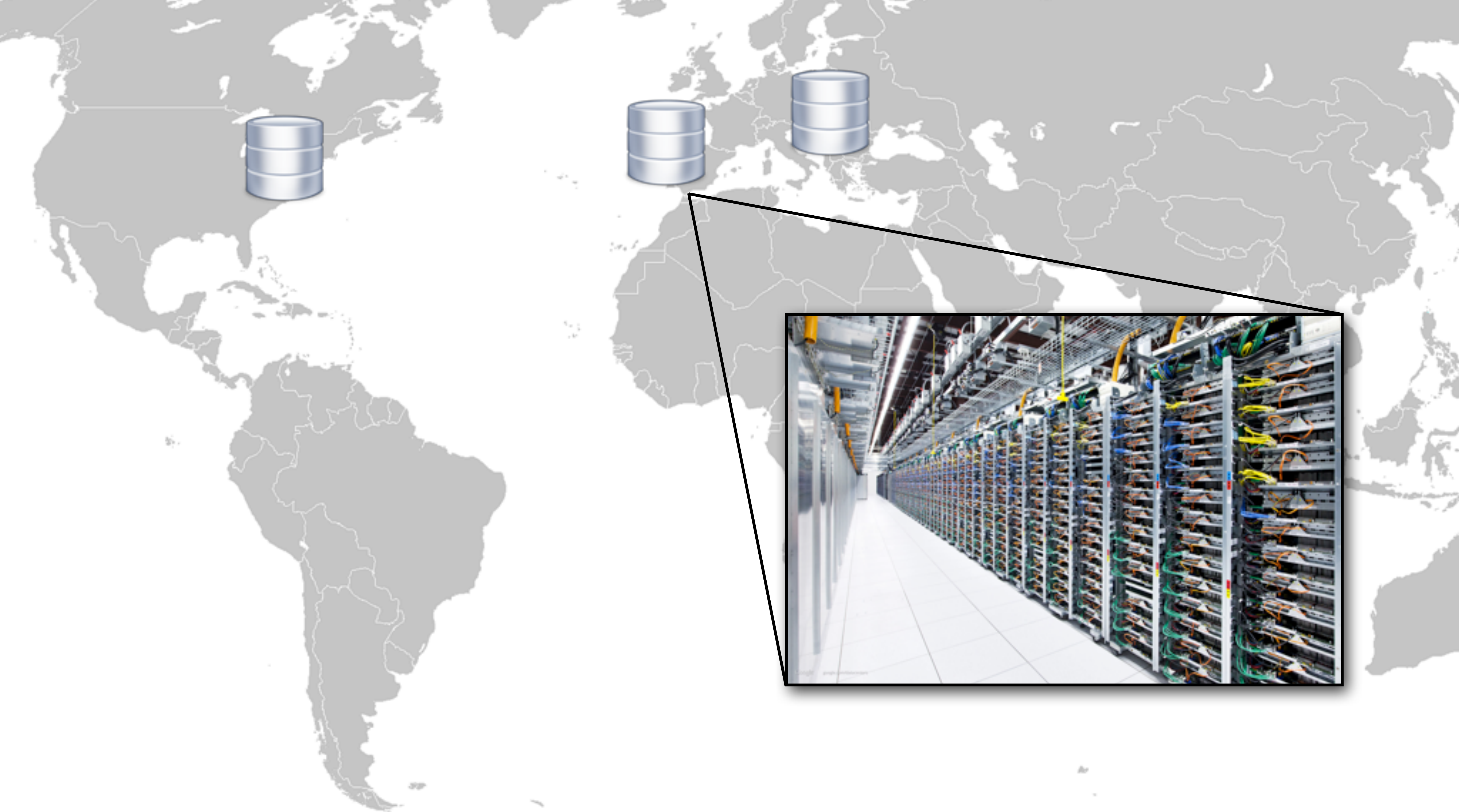

Data is replicated across multiple nodes
Amazon.co.uk: Low Prices in Electronics, Books, Sports Equipment & more
www.amazon.co.uk
amazon.co.uk
Your Amazon.co.uk
Today's Deals
Gift Cards
Help
January Deals
Shop now
Shop by Department
Search
All
Go
Hello. Sign in Your Account
0 Basket
Wish List
¿Compras desde España?
Shopping from Spain?
Visita amazon.es
Descúbrelo
January Deals
Shop now
Apple
Yahoo!
Google
Google Search
I'm Feeling Lucky
Log In, Sign Up or Learn More
facebook
Email or Phone
Keep me logged in
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 centres across the world



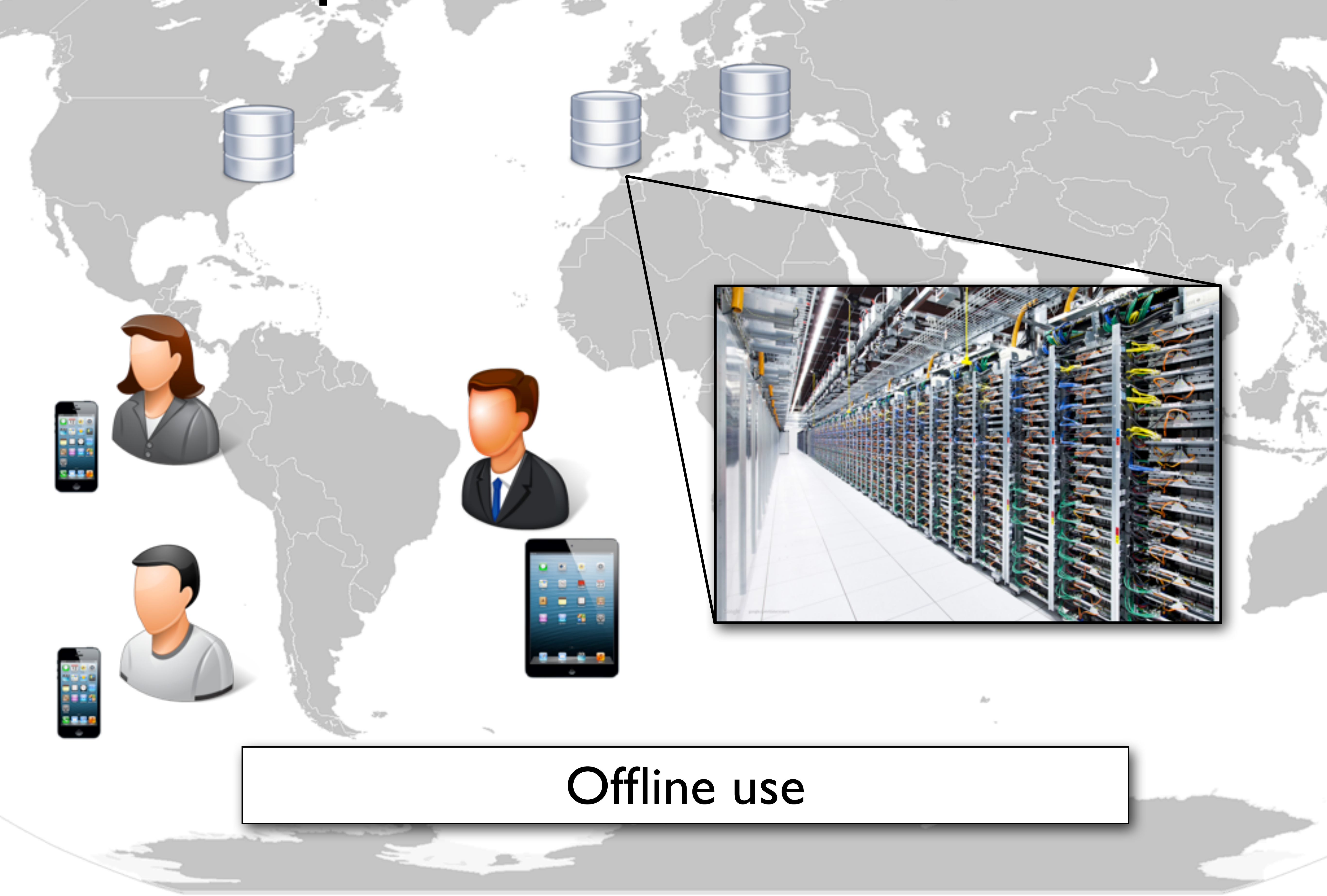
Disaster-tolerance, minimising latency

With thousands of machines inside



Load-balancing, fault-tolerance

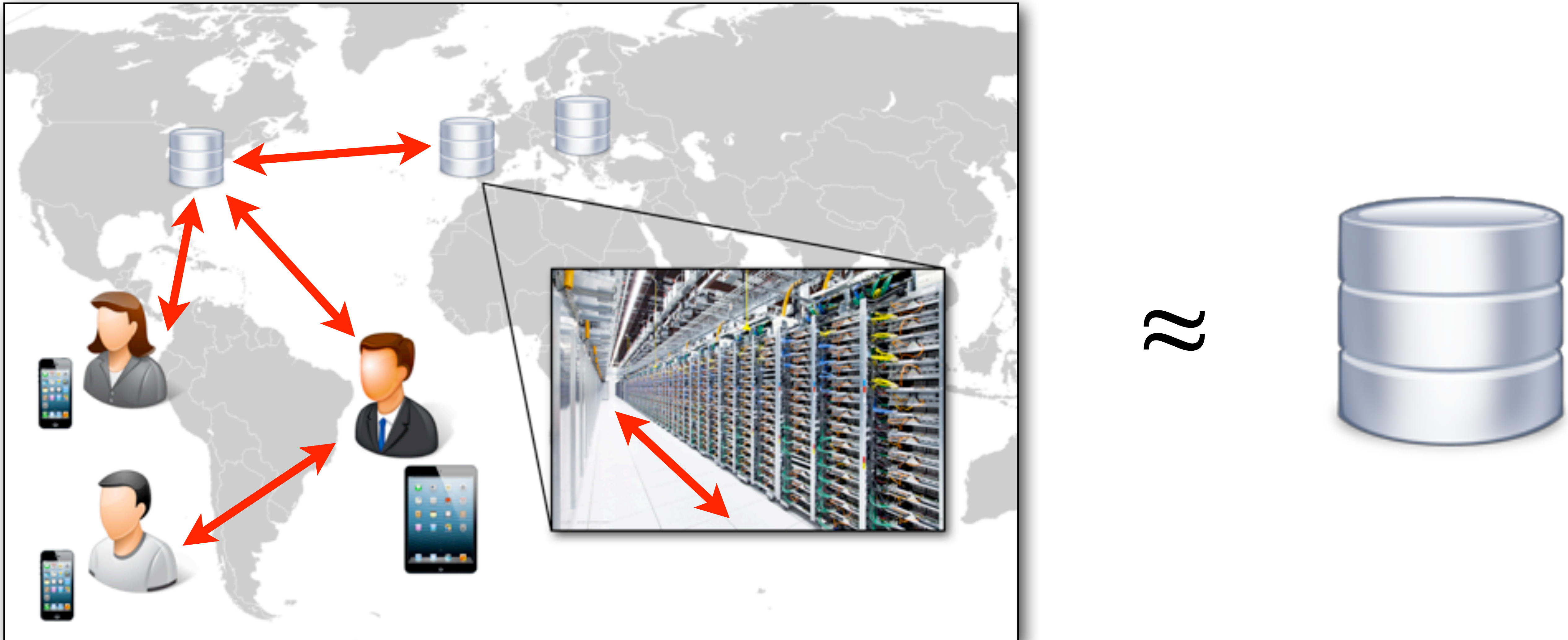
Replicas on mobile devices



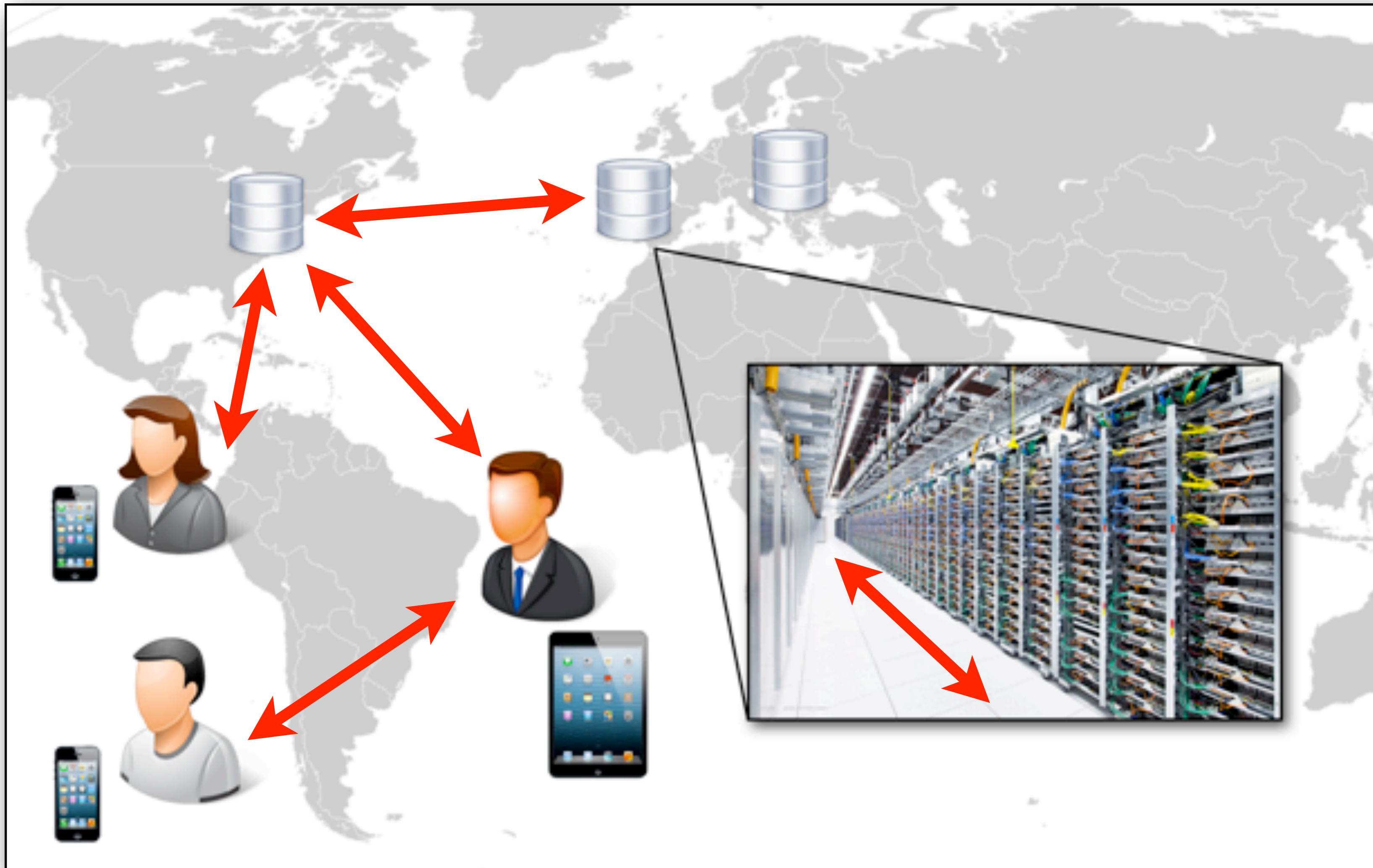
Offline use

≈

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

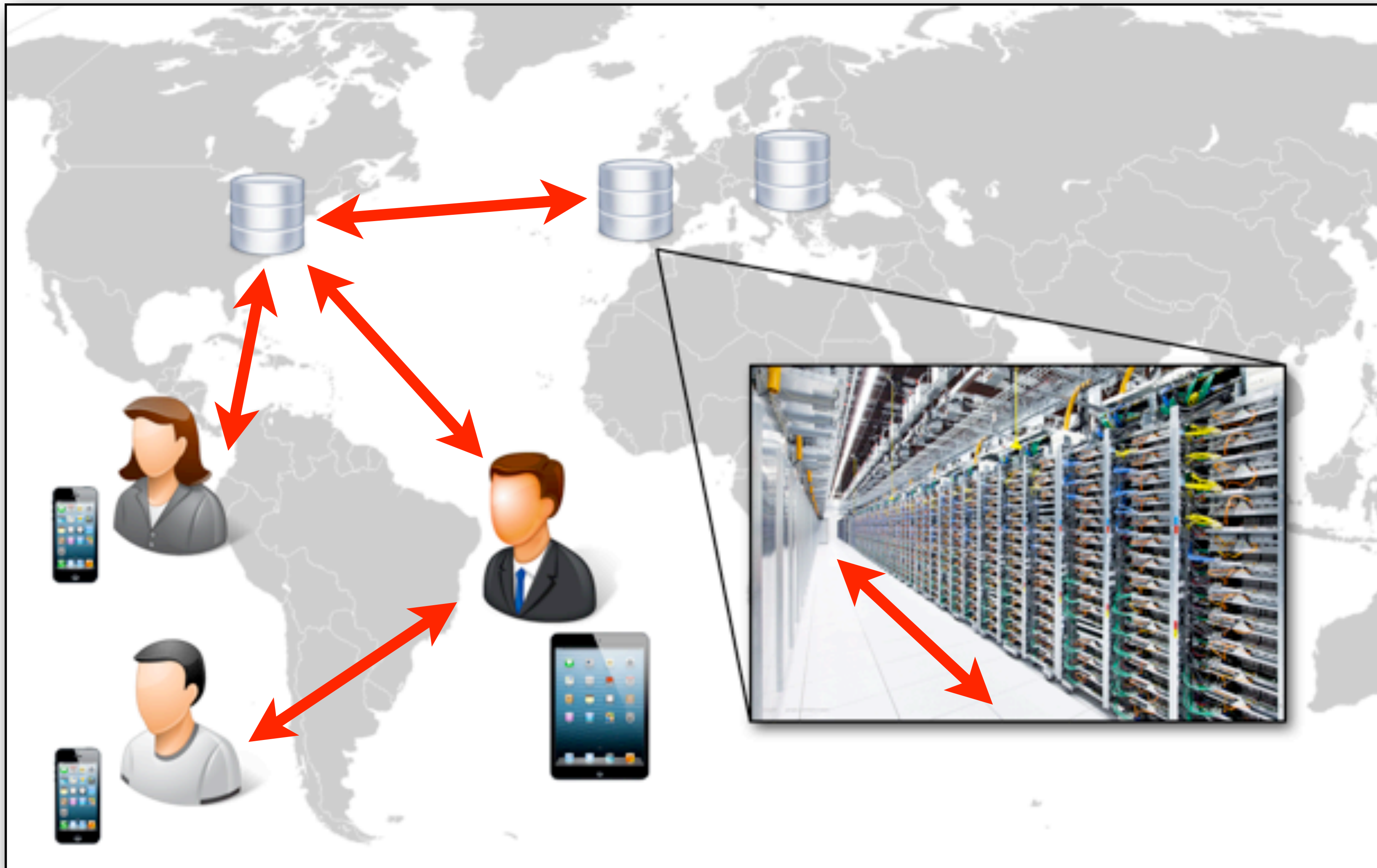


- Strong consistency model: the system behaves as if it processes requests serially on a centralised database
- Requires synchronisation: contact other replicas when processing a request



≈

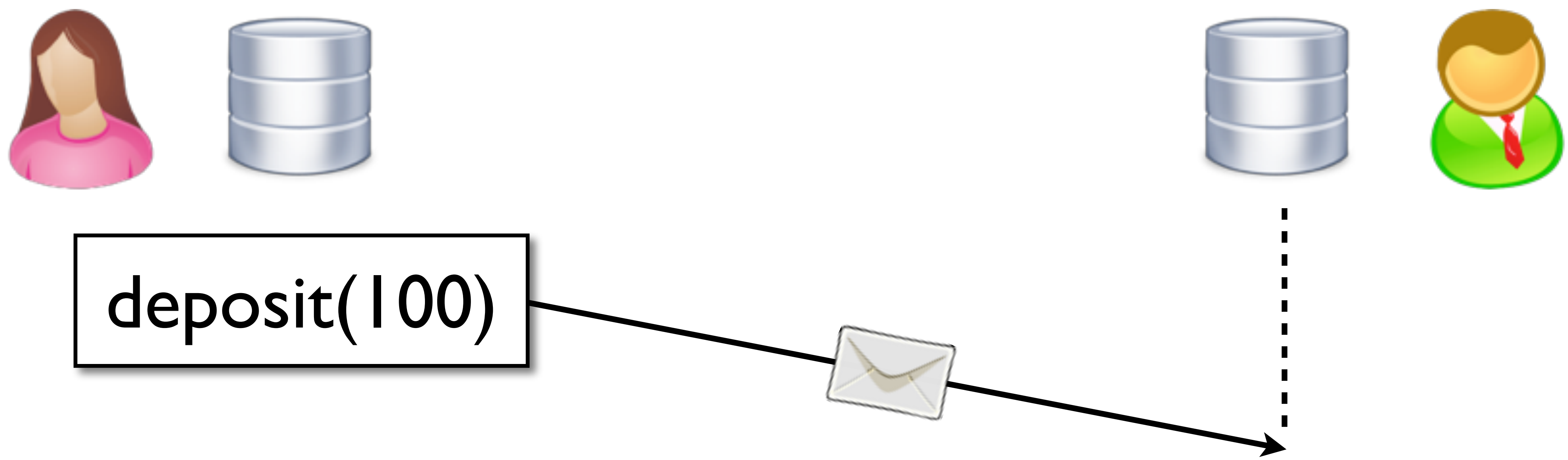
- Increased latency
- Either strong Consistency or Availability in the presence of network Partitions [CAP theorem]



≈

- Increased latency
- Either ~~strong Consistency~~ or Availability in the presence of network Partitions [CAP theorem]
- Weak consistency models

Eventually consistent databases



- No synchronisation: process an update locally, propagate effects to other replicas later
- Weakens consistency: deposit seen with a delay

Integrity invariants

- *Account balance is non-negative*
- *Only registered students are enrolled into a course*
- *The winner of an auction is the highest bidder*

Eventual consistency often too weak to preserve invariants

Integrity invariants

- *Account balance is non-negative*
- *Only registered students are enrolled into a course*
- *The winner of an auction is the highest bidder*

Eventual consistency often too weak to preserve invariants

Invariant: balance $\geq$ 0

balance = 100

balance = 100

Invariant: balance ≥ 0

balance = 100

withdraw(100) : ✔

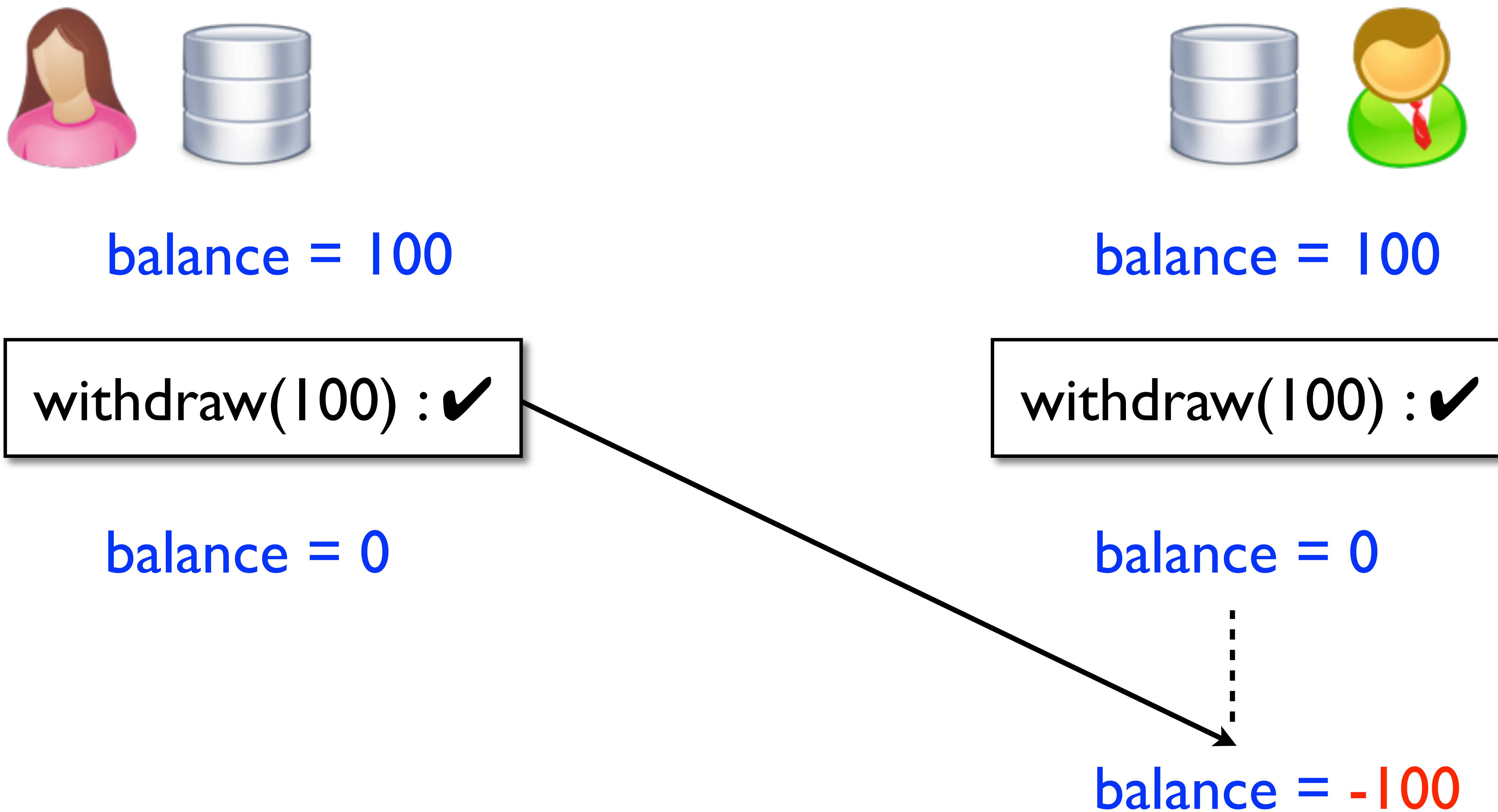
balance = 0

balance = 100

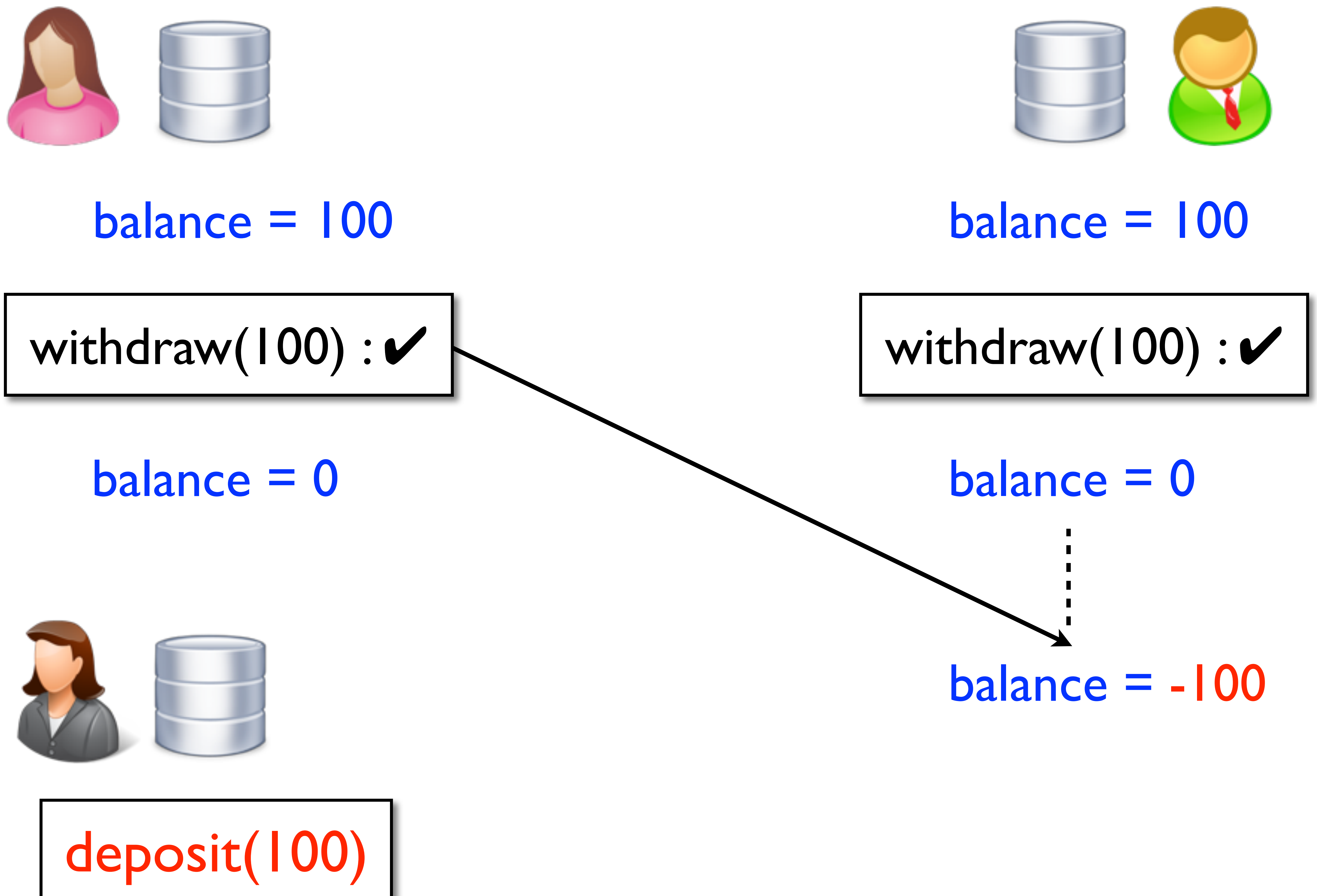
withdraw(100) : ✔

balance = 0

Invariant: balance ≥ 0



Invariant: balance ≥ 0



Consistency choices

- Choose consistency level for each operation:
  - Withdrawals strongly consistent
  - Deposits eventually consistent
- Databases:
  - Commercial: Amazon DynamoDB, Microsoft DocumentDB, Basho Riak
  - Research: Li[+] 2012, Terry[+] 2013, Balegas[+] 2015
- Pay for stronger semantics with latency, possible unavailability and money

Consistency choices

Problem: hard to figure out the minimum consistency sufficient to maintain correctness

Contribution: proof rule and tool for checking integrity invariants under given consistency choices

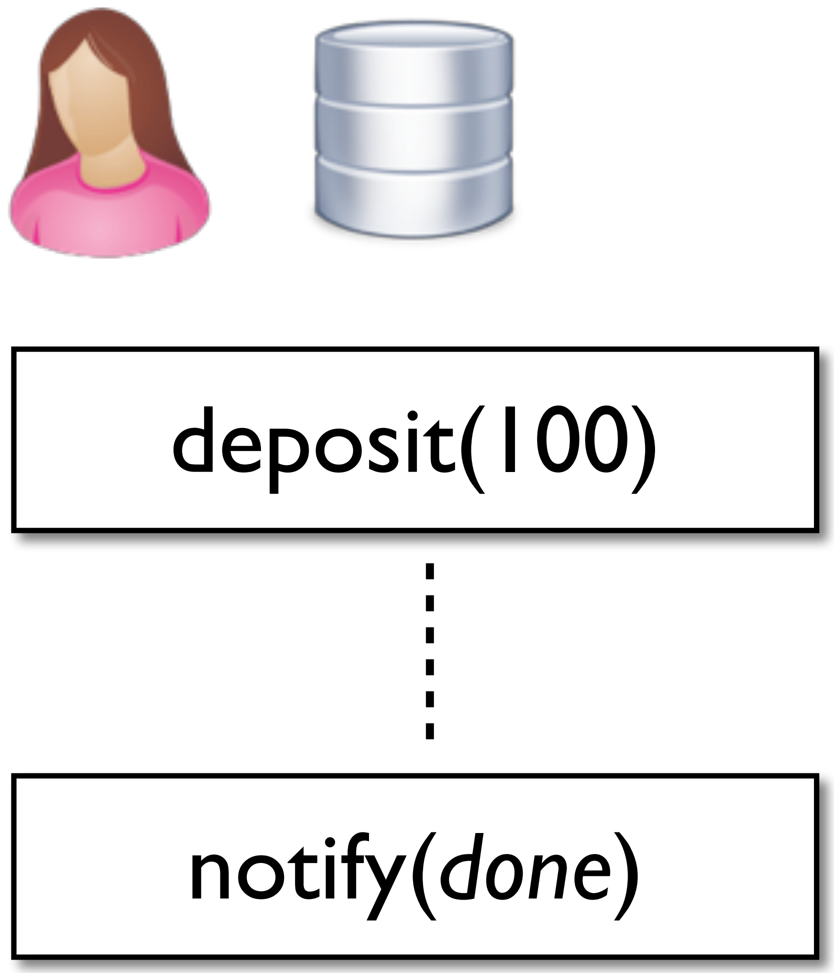
Consistency model

- Generic model with consistency choices
- Not implemented, but can encode many existing models that are:

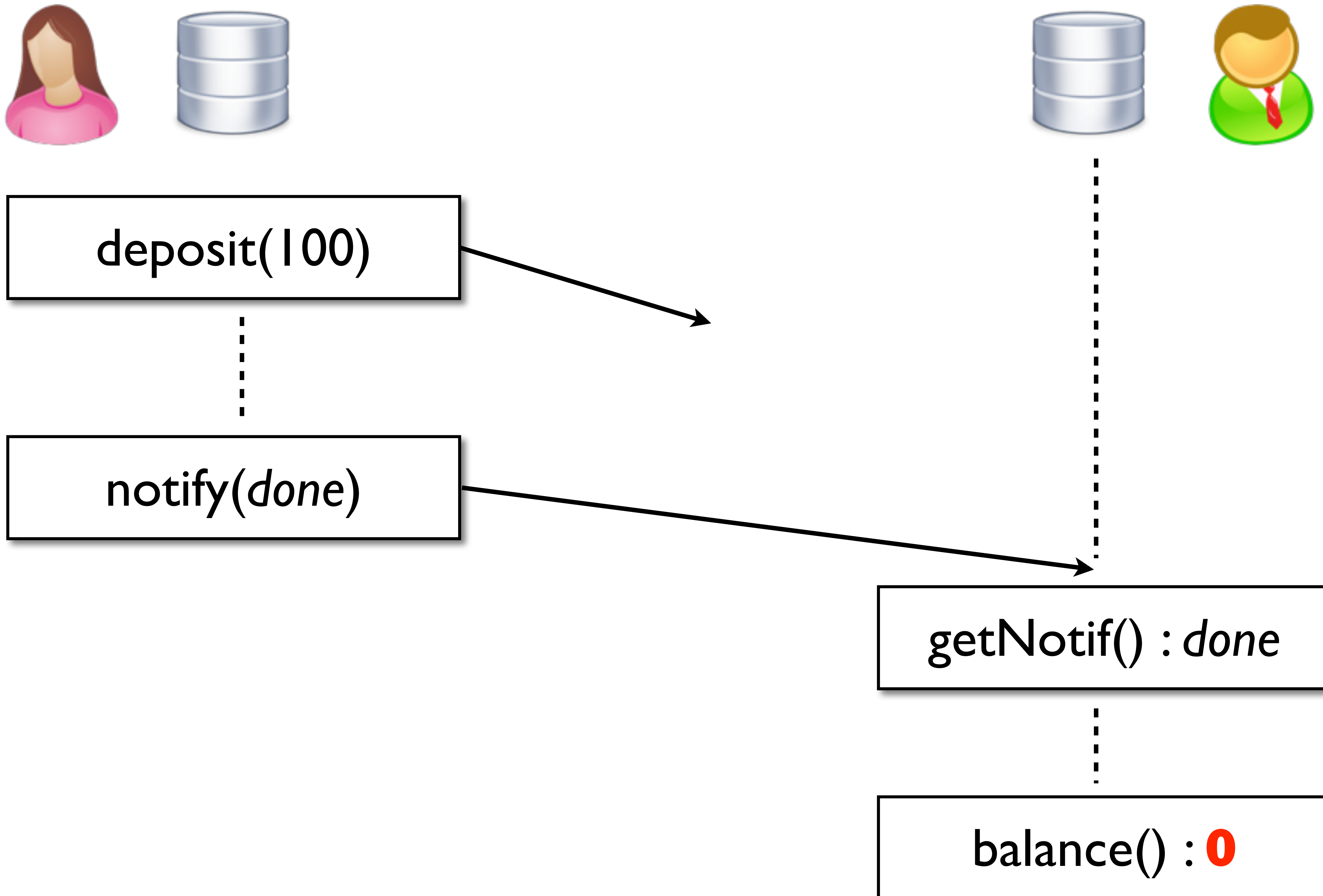
  RedBlue consistency [Li+ 2012],
  reservation locks [Balegas+ 2015],
  parallel snapshot isolation [Sovran+ 2011], ...

- Declarative formal semantics in the paper

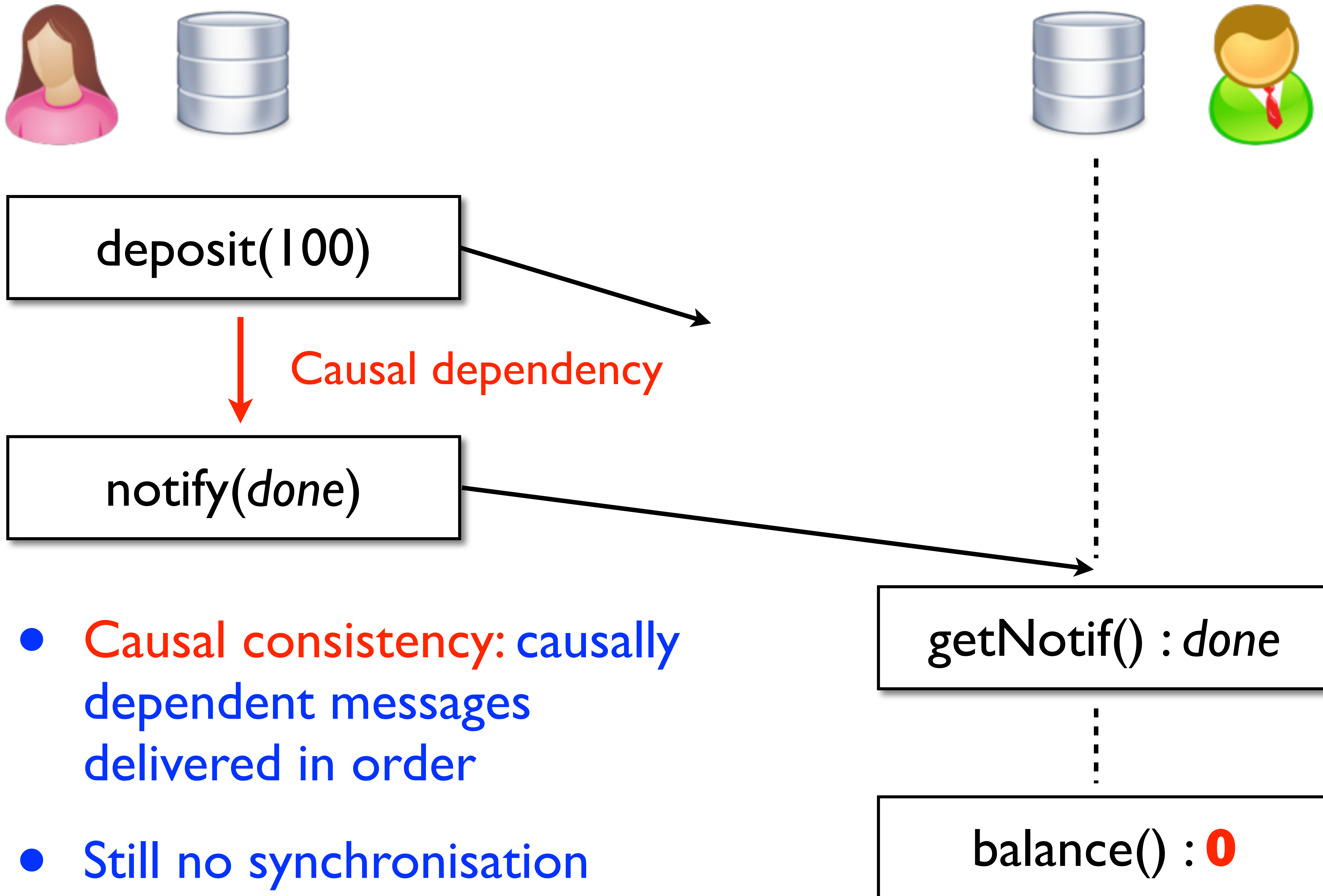
Anomalies of eventual consistency



Anomalies of eventual consistency



Anomalies of eventual consistency



- Causal consistency: causally dependent messages delivered in order
- Still no synchronisation

Operation semantics

σ

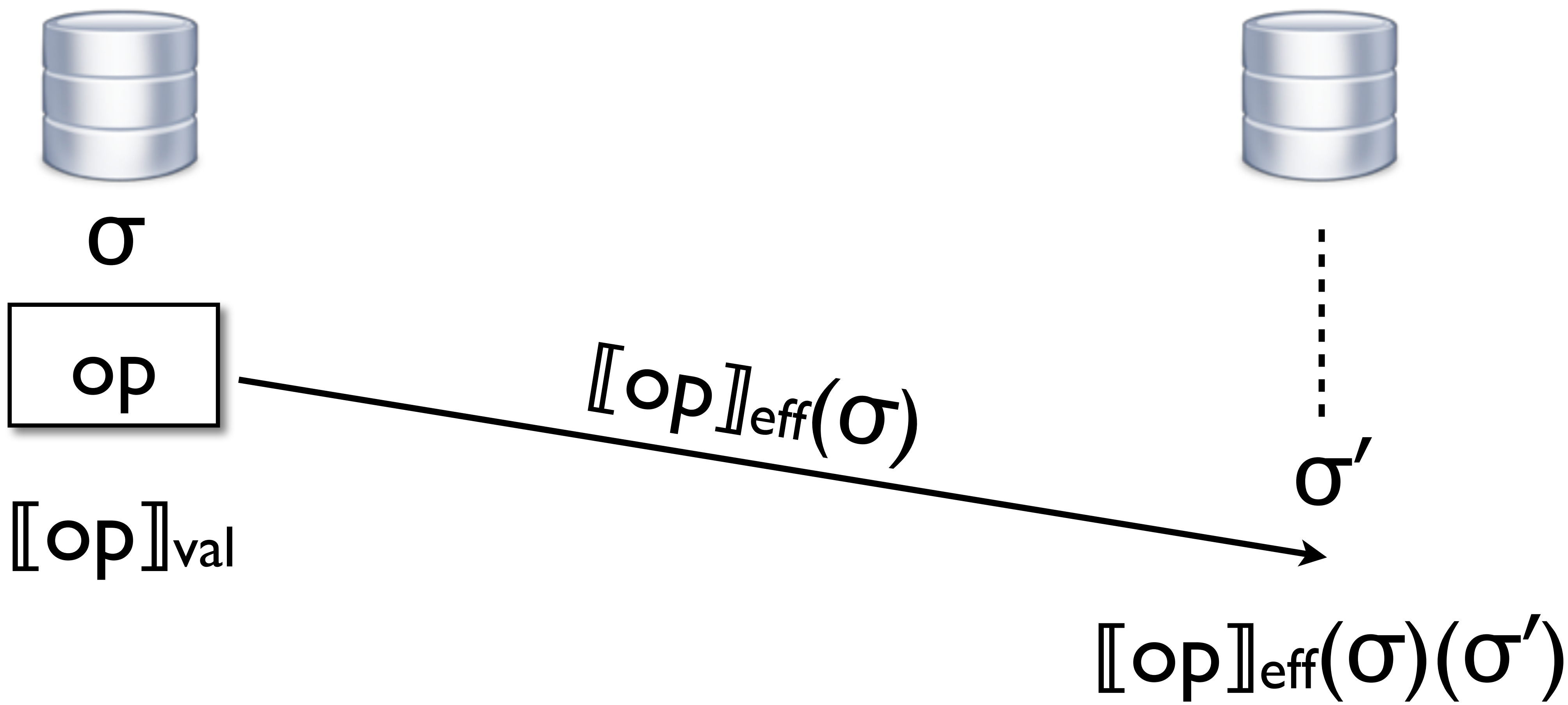
op

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

Replica states: $\sigma \in State$

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

Operation semantics

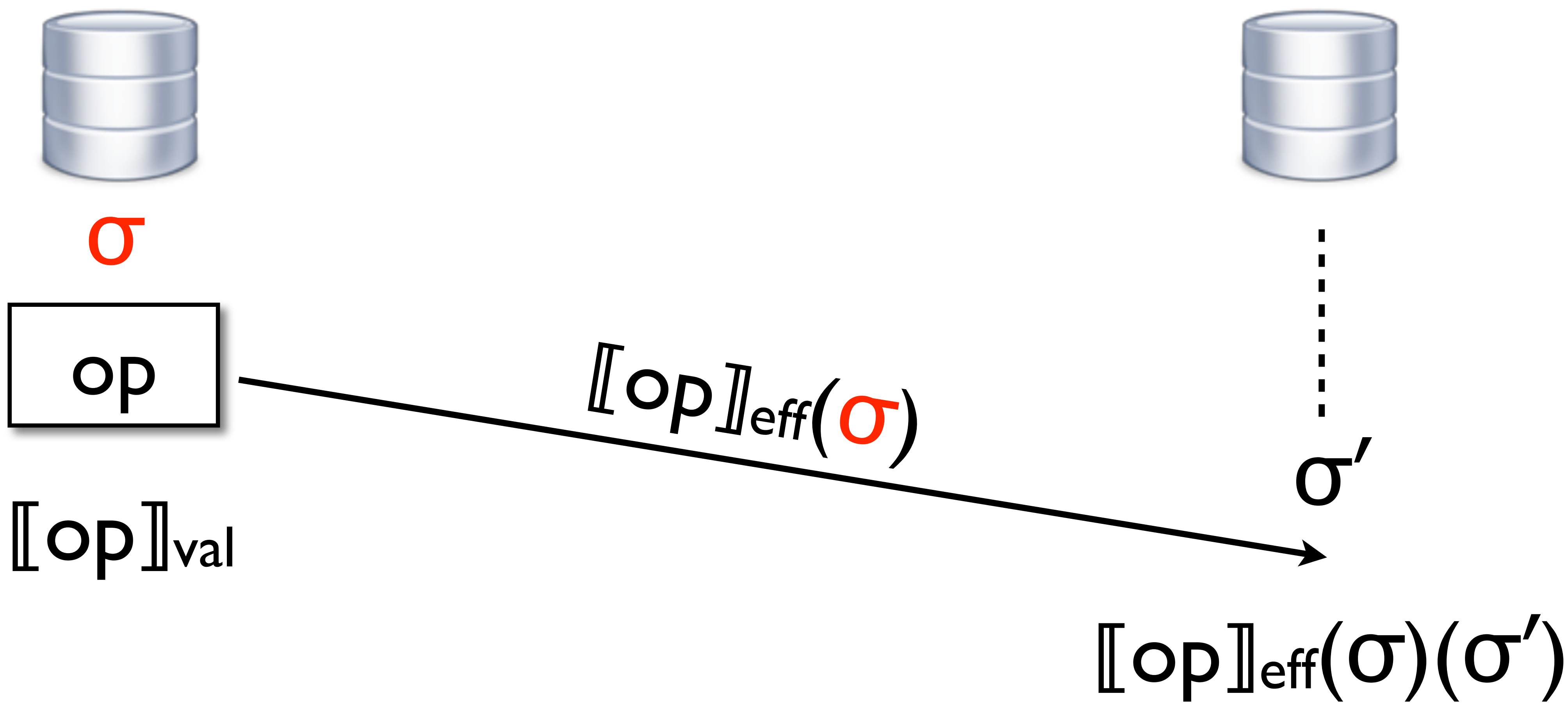


Replica states: $\sigma \in State$

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

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

Operation semantics

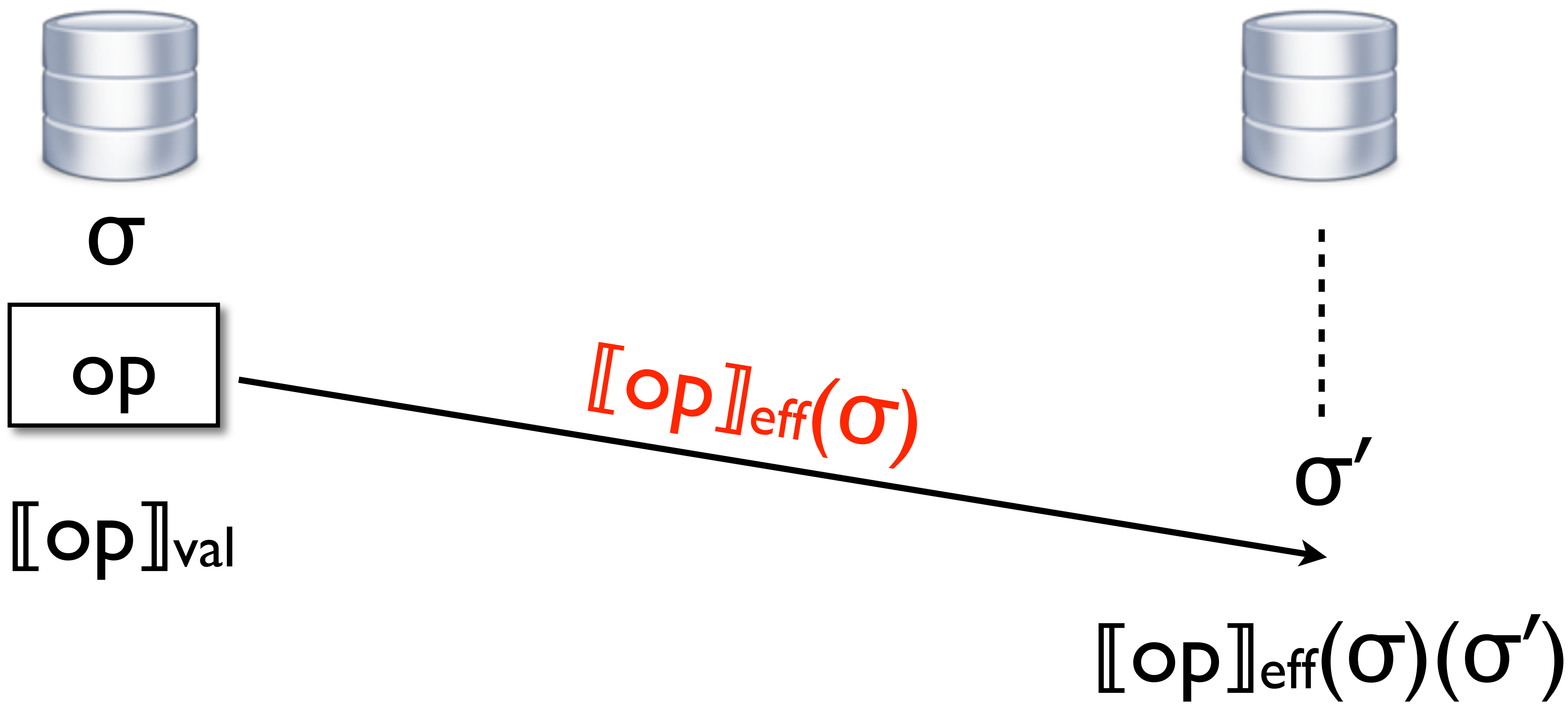


Replica states: $\sigma \in \mathrm{State}$

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

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

Operation semantics

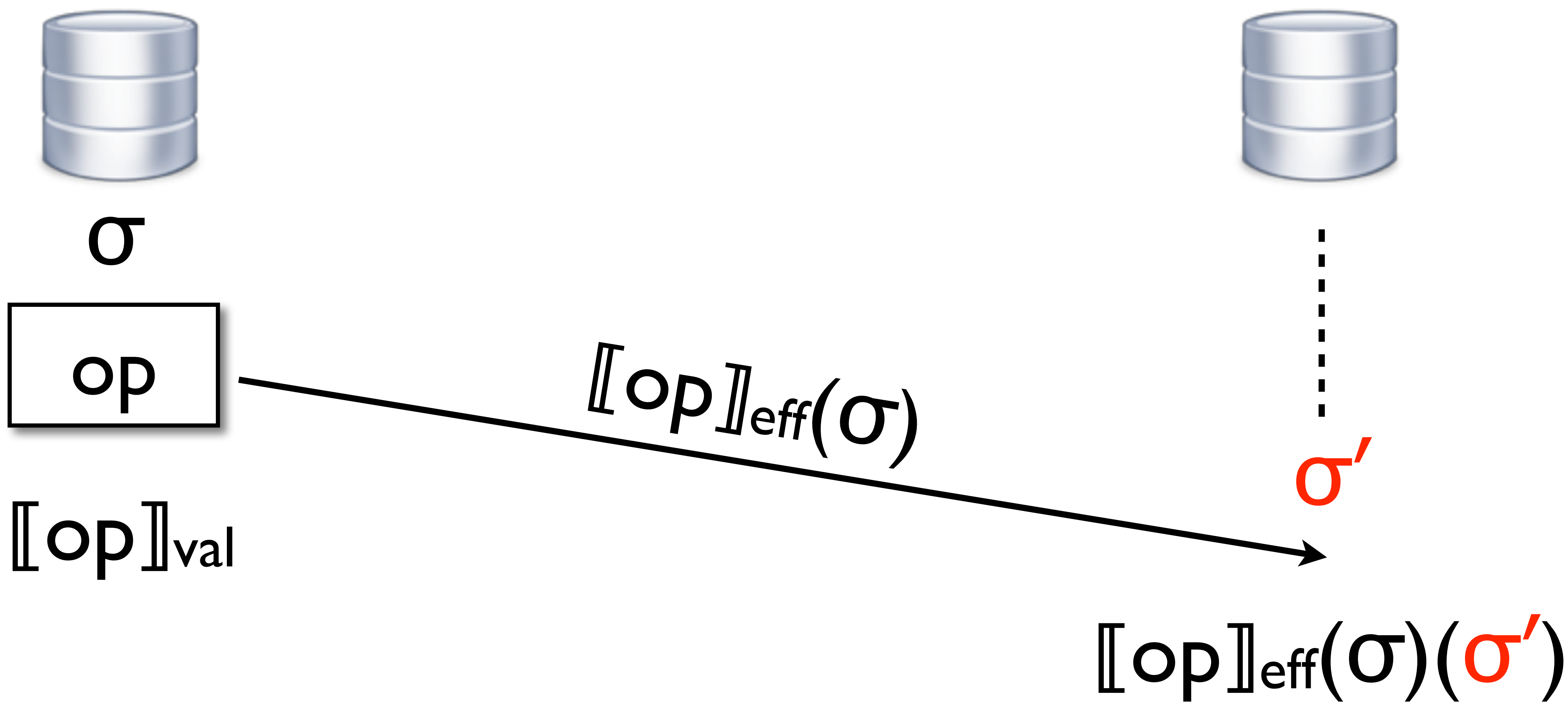


Replica states: $\sigma \in \mathrm{State}$

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

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

Operation semantics

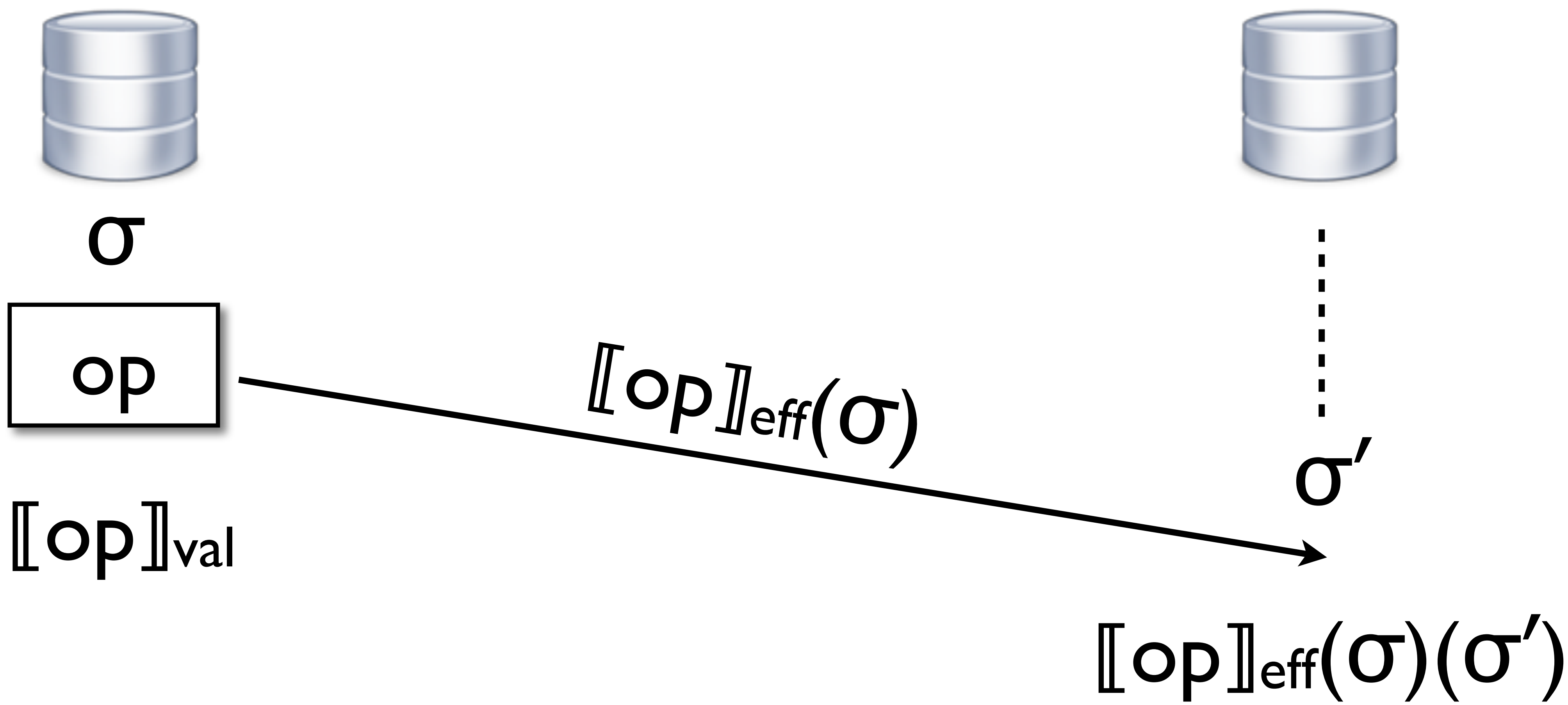


Replica states: $\sigma \in \text{State}$

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

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

Operation semantics

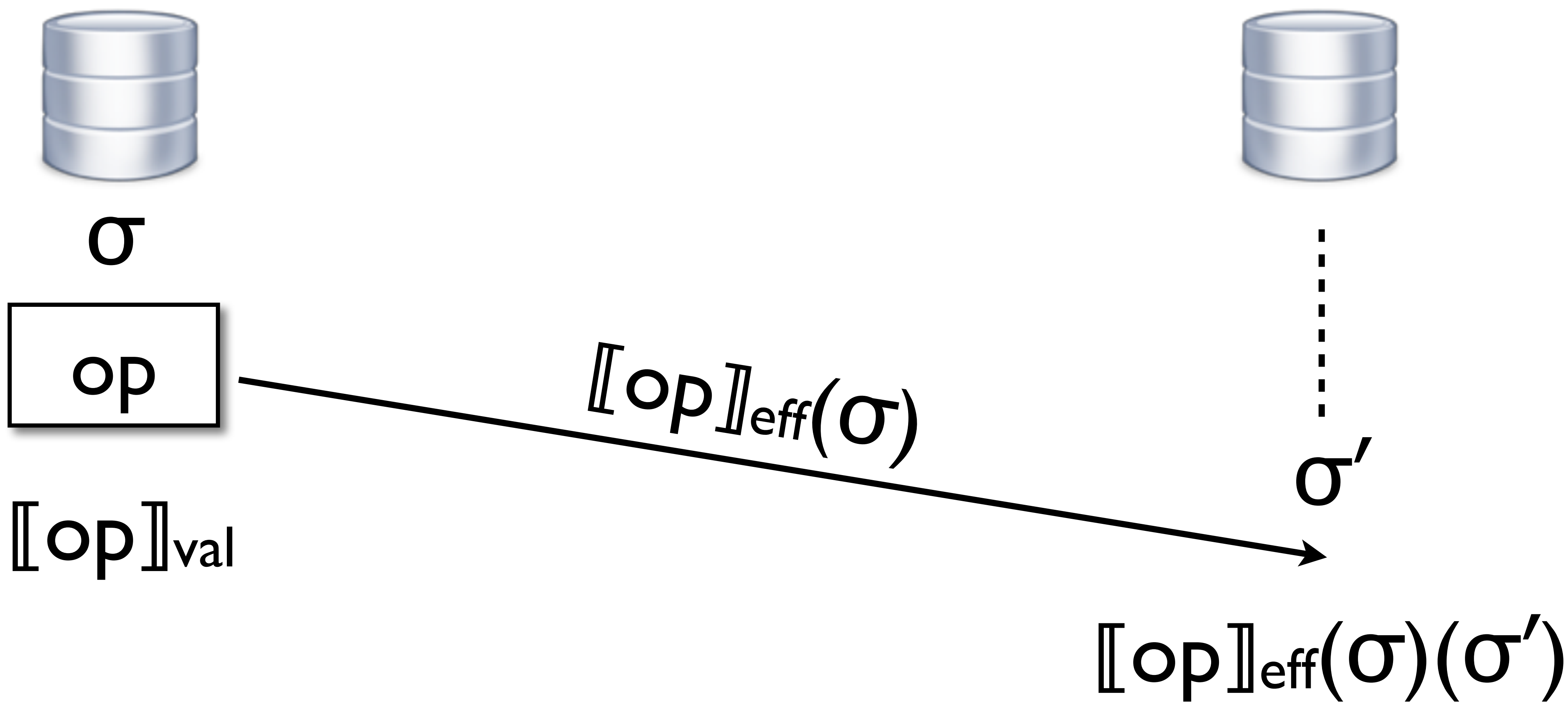


$$\mathrm{State} = Z$$

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

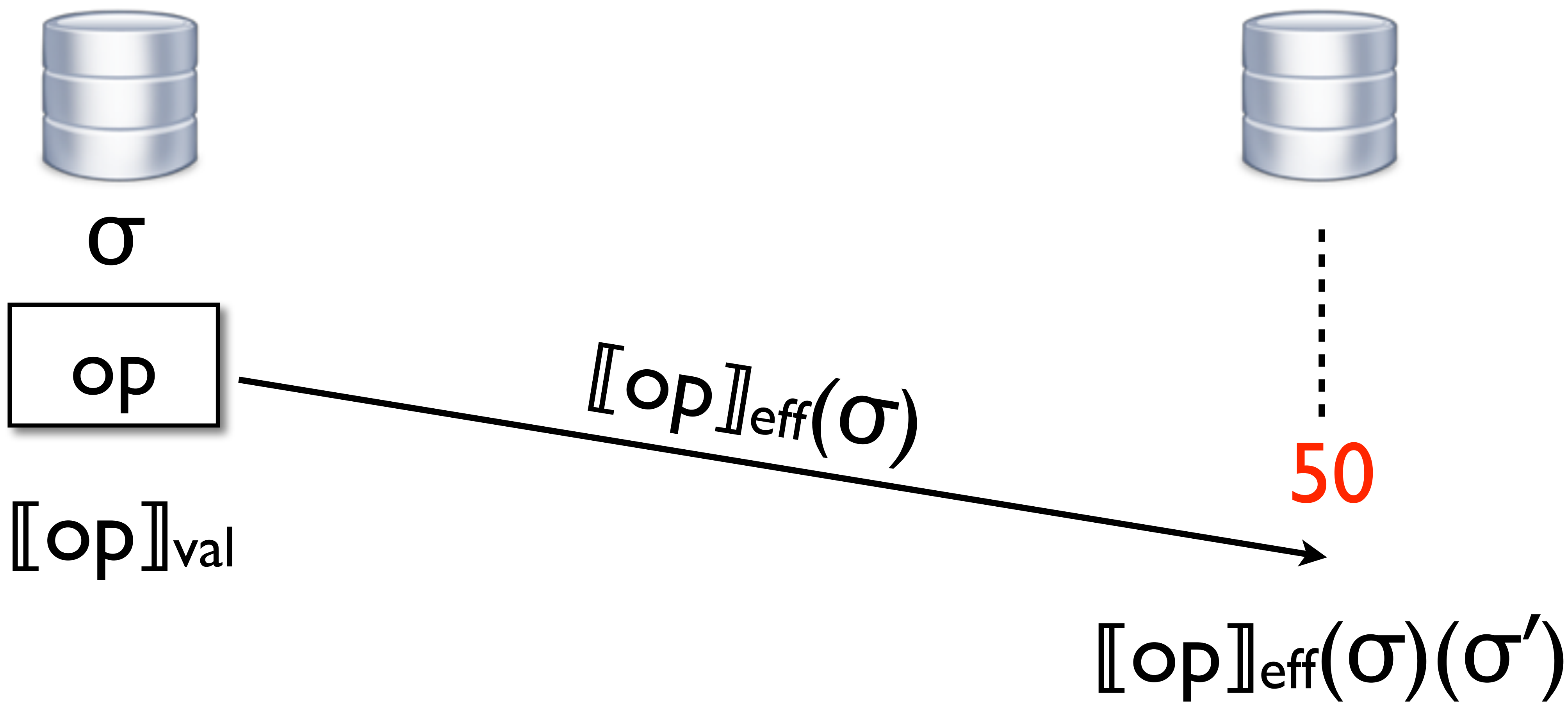
$$[\![\mathrm{balance}()]\!]_{eff}(\sigma) = \lambda\sigma.\,\sigma$$

Operation semantics



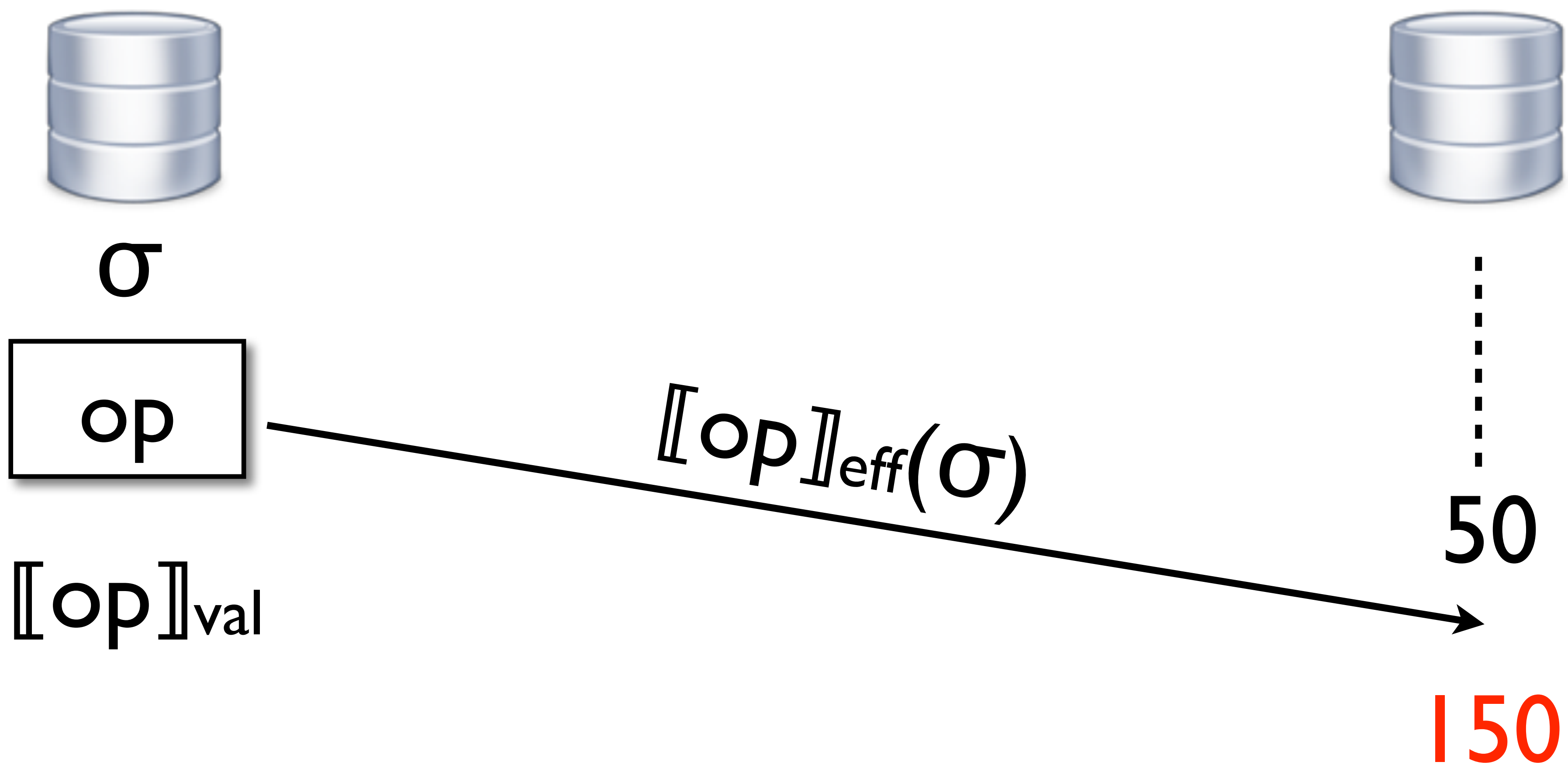
$$[\![\text{deposit}(100)]\!]_{\text{eff}}(\sigma) = \lambda\sigma'.\,(\sigma' + 100)$$

Operation semantics



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

Operation semantics



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

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

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

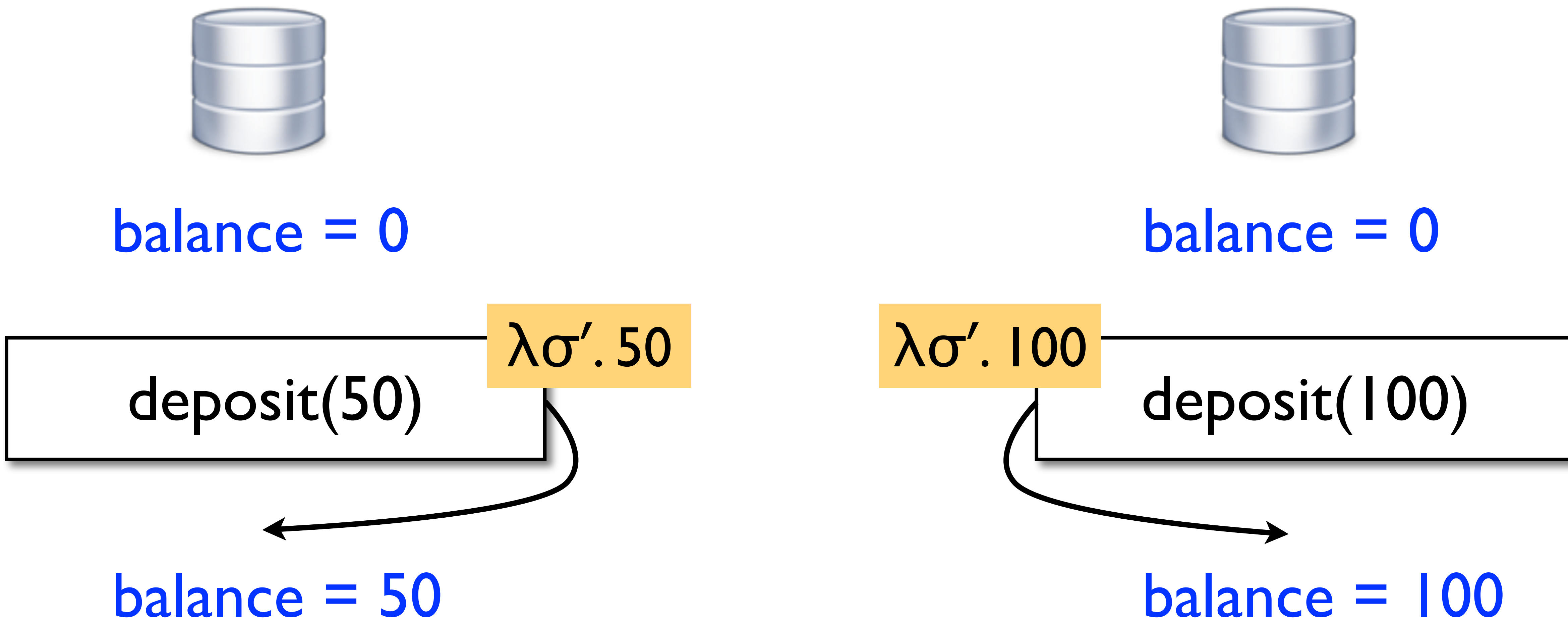
balance = 0

deposit(50)

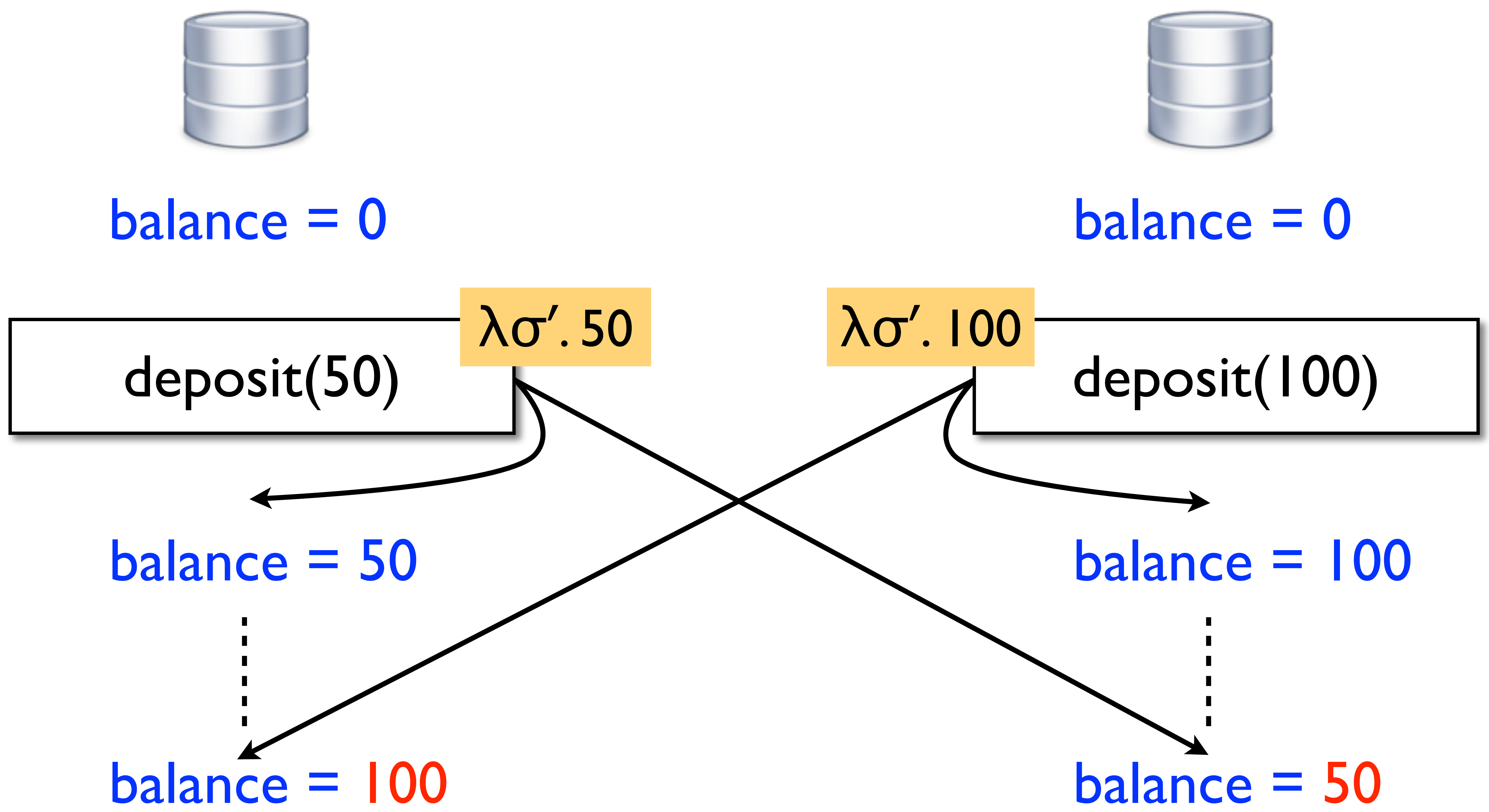
balance = 0

deposit(100)

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



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



Replicas diverge!

Ensuring convergence

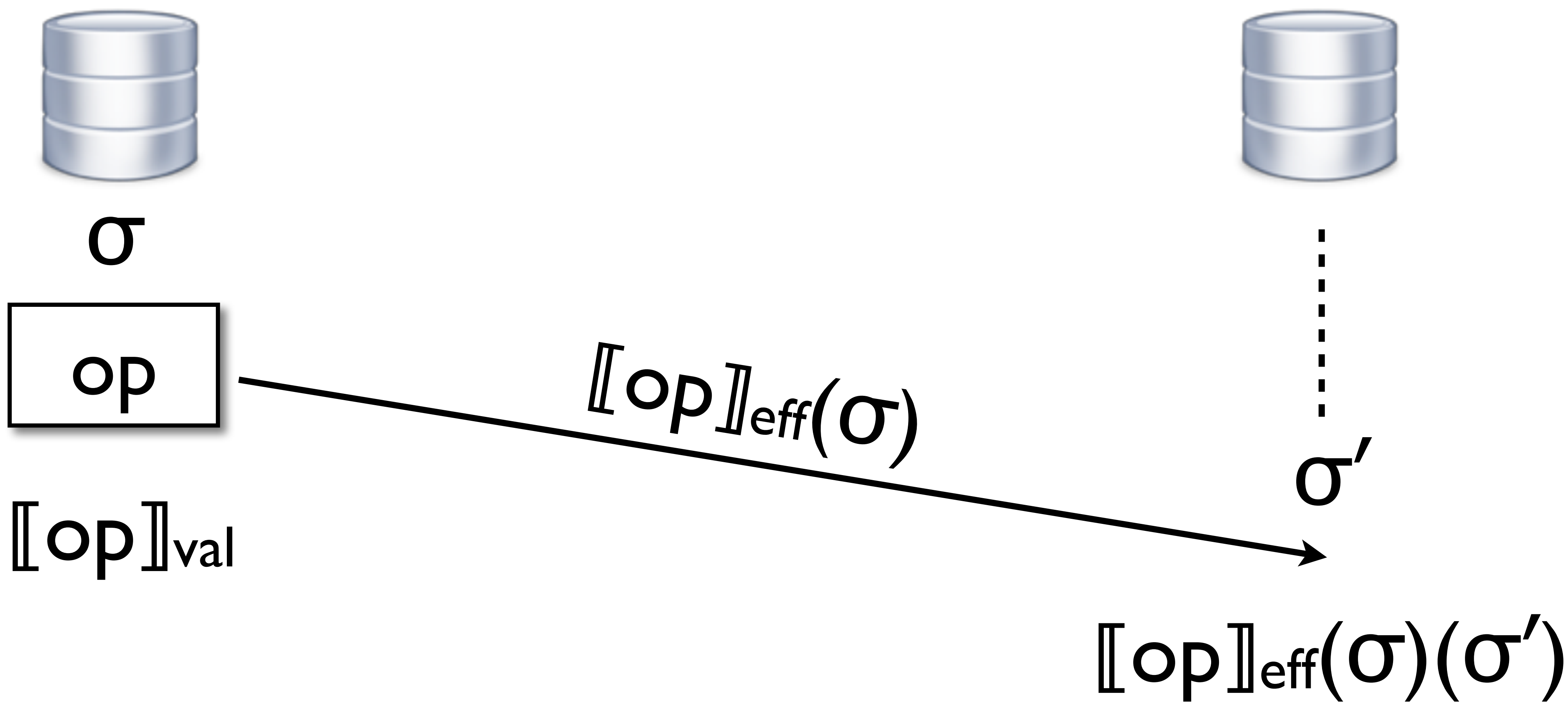
- Effects of operations have to commute:

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

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

- Replicated data types (CRDTs) [Shapiro+ 2011]: ready-made commutative implementations

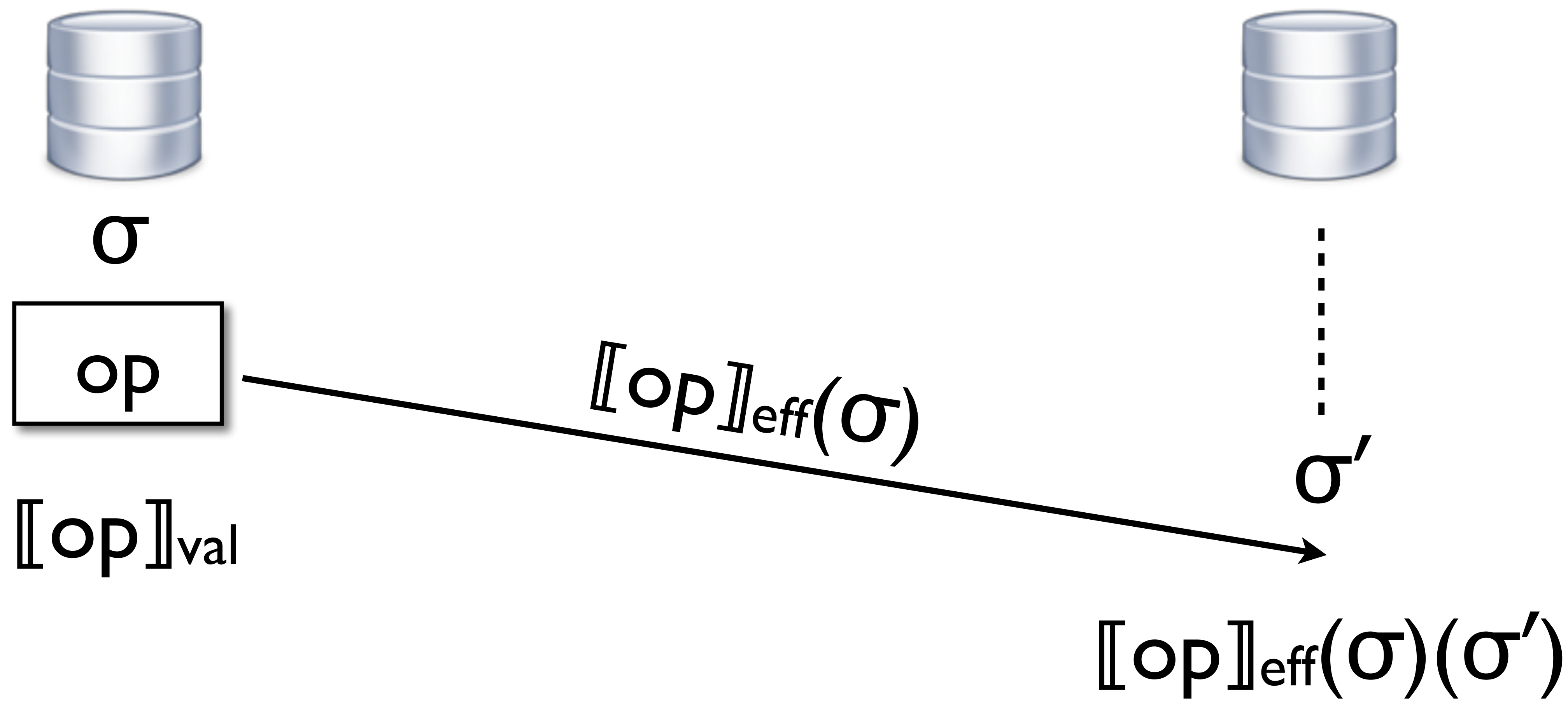
Operation semantics



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

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

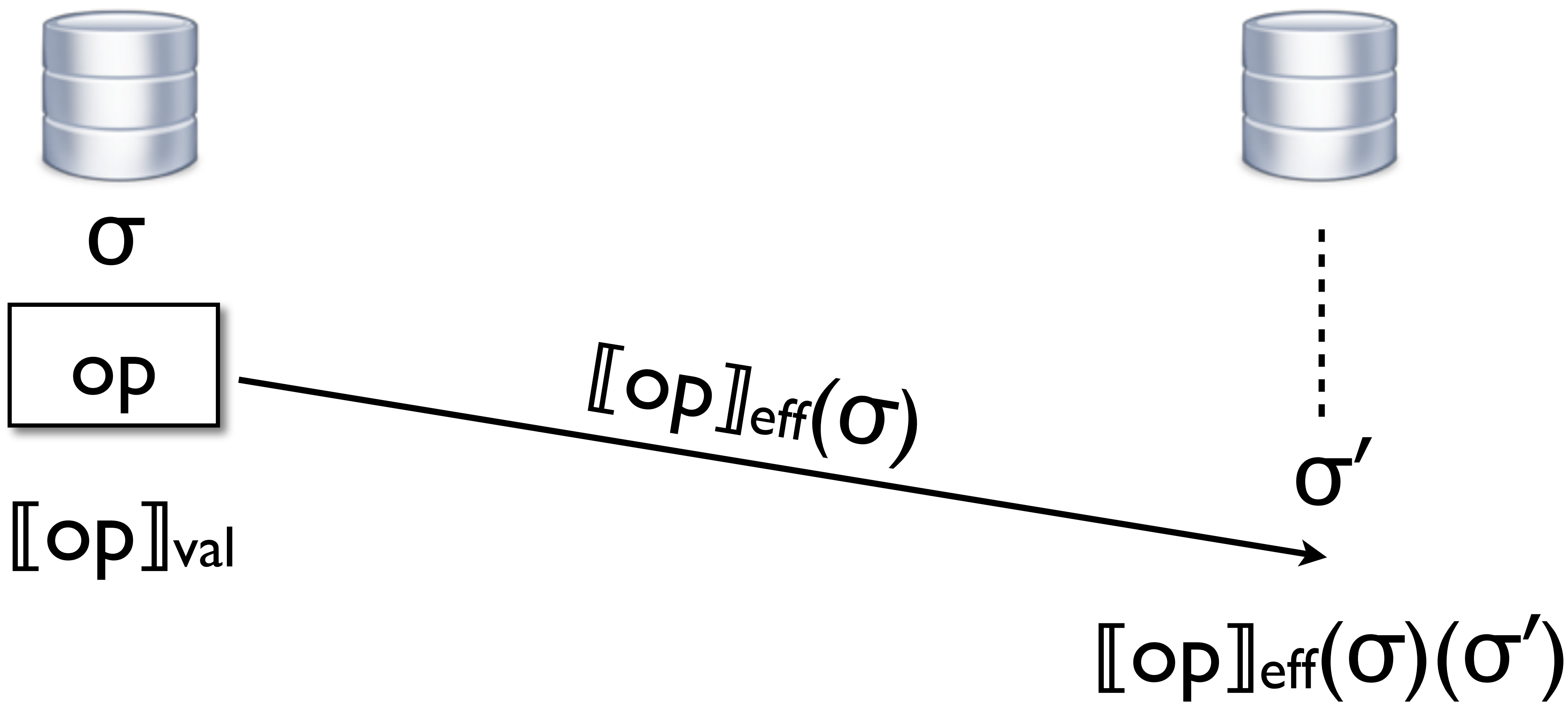
Operation semantics



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

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

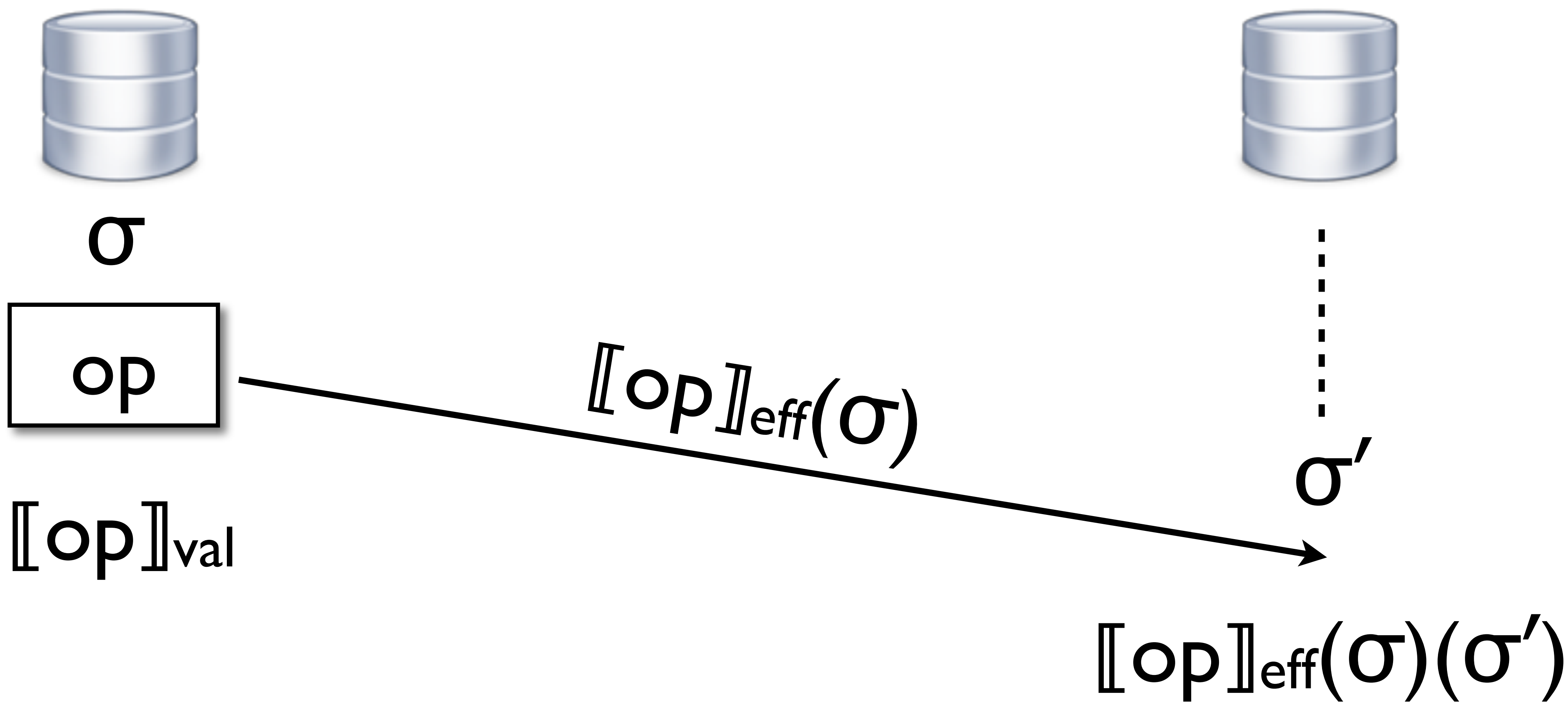
Operation semantics



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

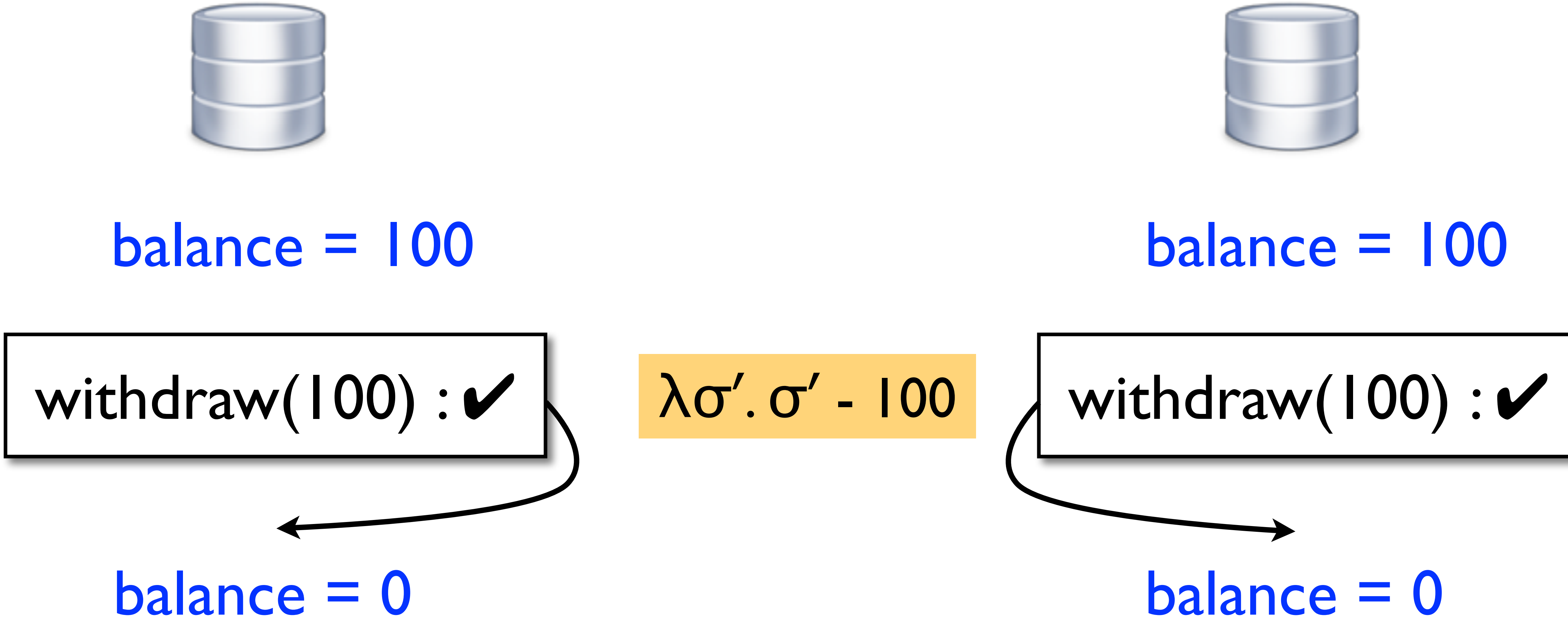
$$\quad \text{if } \sigma \geq 100 \text{ then } (\lambda\sigma'.\, \sigma' - 100) \text{ else } (\lambda\sigma'.\, \sigma')$$

Operation semantics

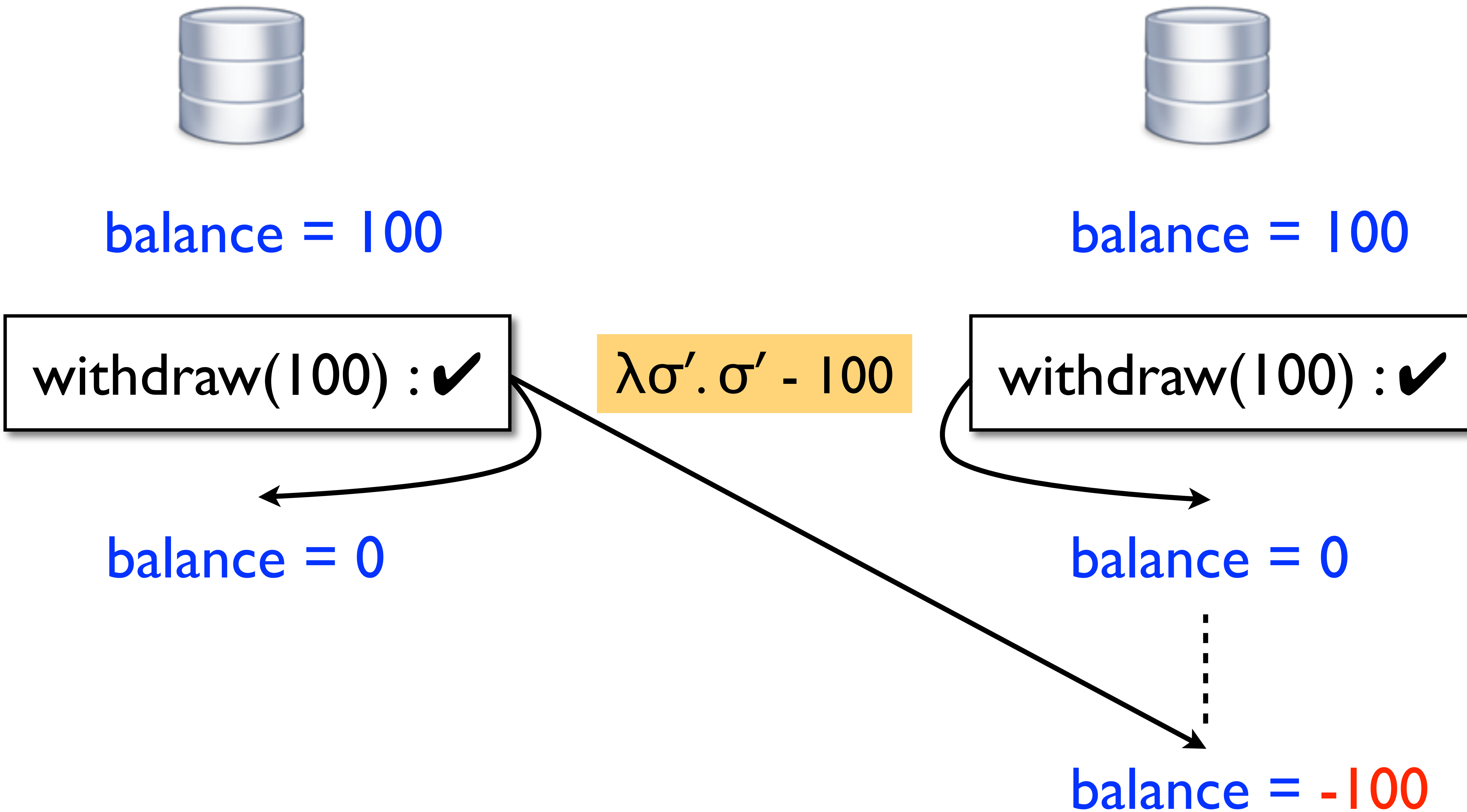


$$[\![\text{withdraw}(100)]\!]_{eff}(\sigma) =$$

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



$$[\![\text{withdraw}(100)]\!]_{\text{eff}}(\sigma) = \text{if } \sigma \geq 100 \text{ then } (\lambda\sigma'.\, \sigma' - 100) \text{ else } (\lambda\sigma'.\, \sigma')$$



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

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

Strengthening consistency

Token system ≈ locks on steroids:

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

Examples:

- Mutual exclusion lock:
  Token = $\{lock\}$; $lock \bowtie lock$
- Readers-writer lock:
  Token = $\{R, W\}$; $\_ \bowtie W$; $W \bowtie \_$

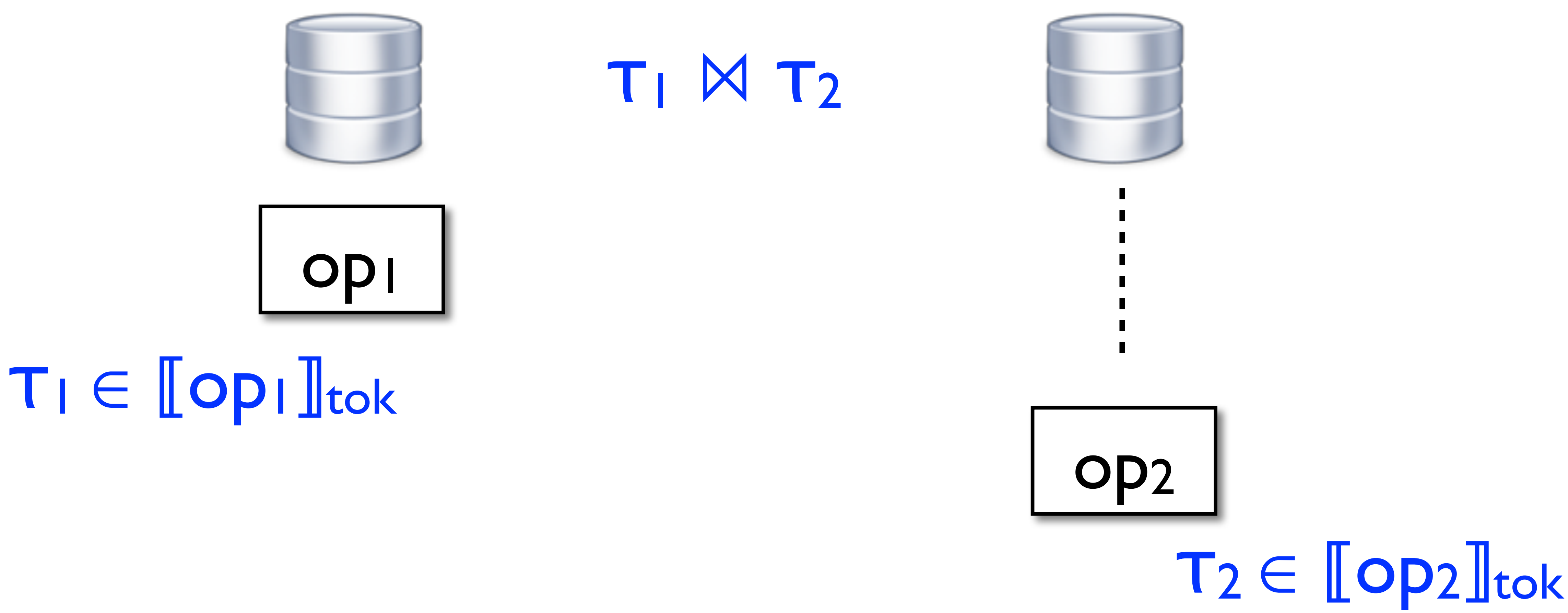
- Each operation acquires a set of tokens:
  $[\![\mathrm{op}]\!]_{\mathrm{tok}} \in \mathrm{State} \to \mathcal{P}(\mathrm{Token})$

- Operations acquiring conflicting tokens cannot be unaware of each other

- Each operation acquires a set of tokens:

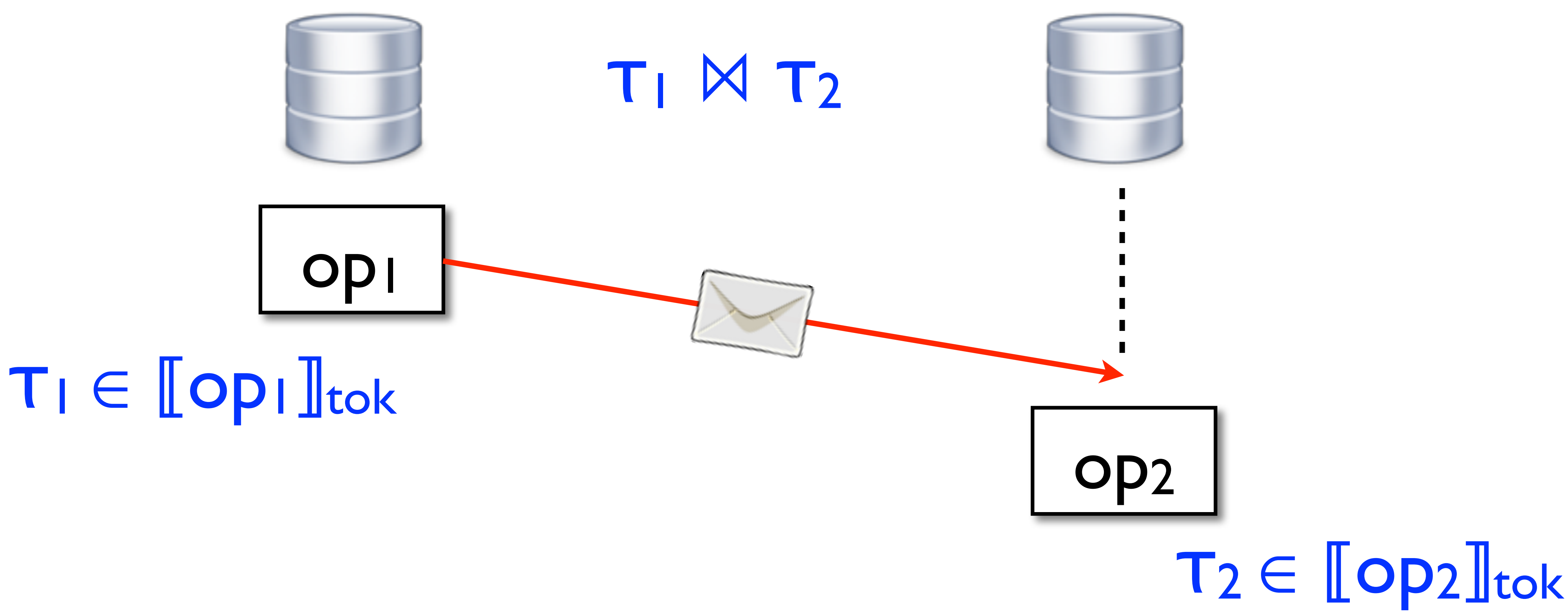
  $[\![\mathrm{op}]\!]_{\mathrm{tok}} \in \mathrm{State} \rightarrow \mathcal{P}(\mathrm{Token})$

- Operations acquiring conflicting tokens cannot be unaware of each other



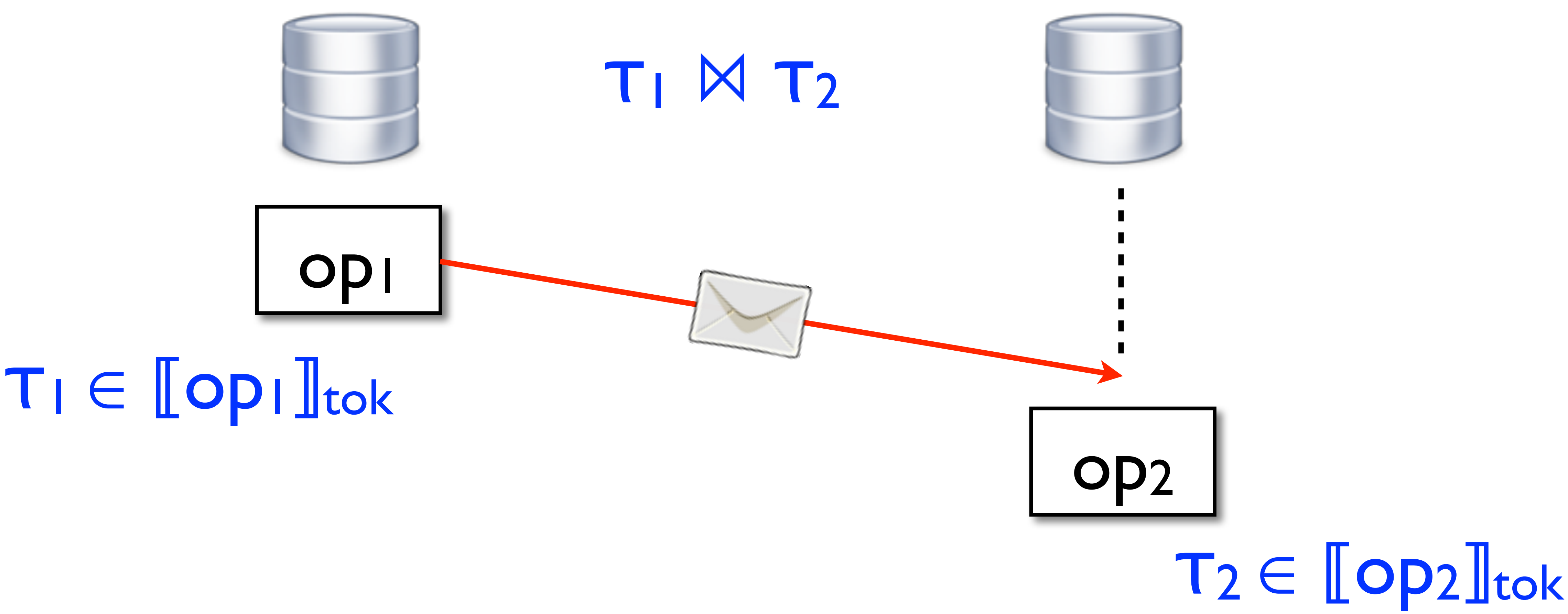
- Each operation acquires a set of tokens:
  $[\![\mathsf{op}]\!]_{\mathsf{tok}} \in \mathsf{State} \rightarrow \mathcal{P}(\mathsf{Token})$

- Operations acquiring conflicting tokens cannot be unaware of each other



- Each operation acquires a set of tokens:
  $[\![op]\!]_{tok} \in State \to \mathcal{P}(Token)$

- Operations acquiring conflicting tokens cannot be unaware of each other



- Requires synchronisation in implementations

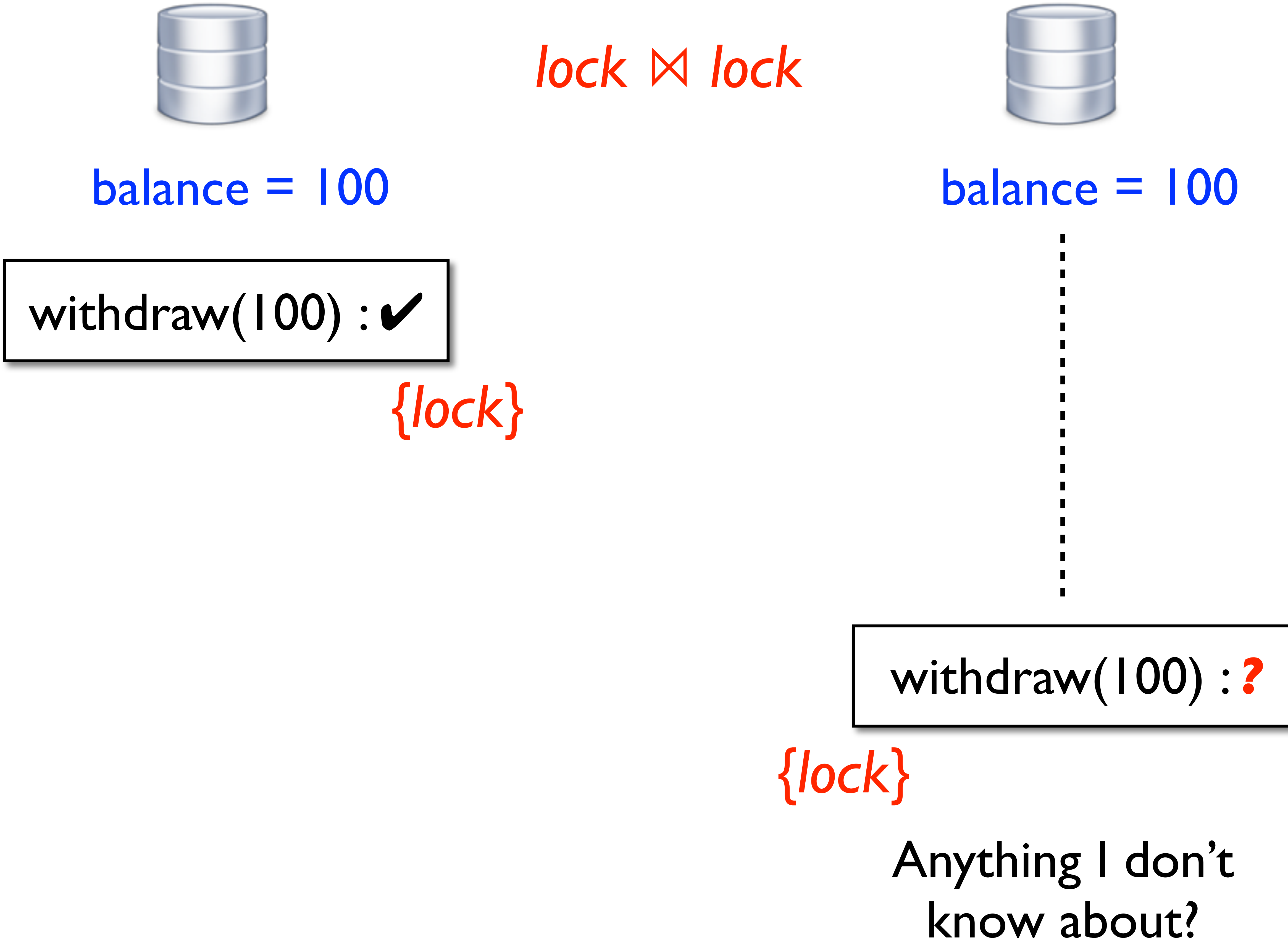
lock ⋈ *lock*

balance = 100

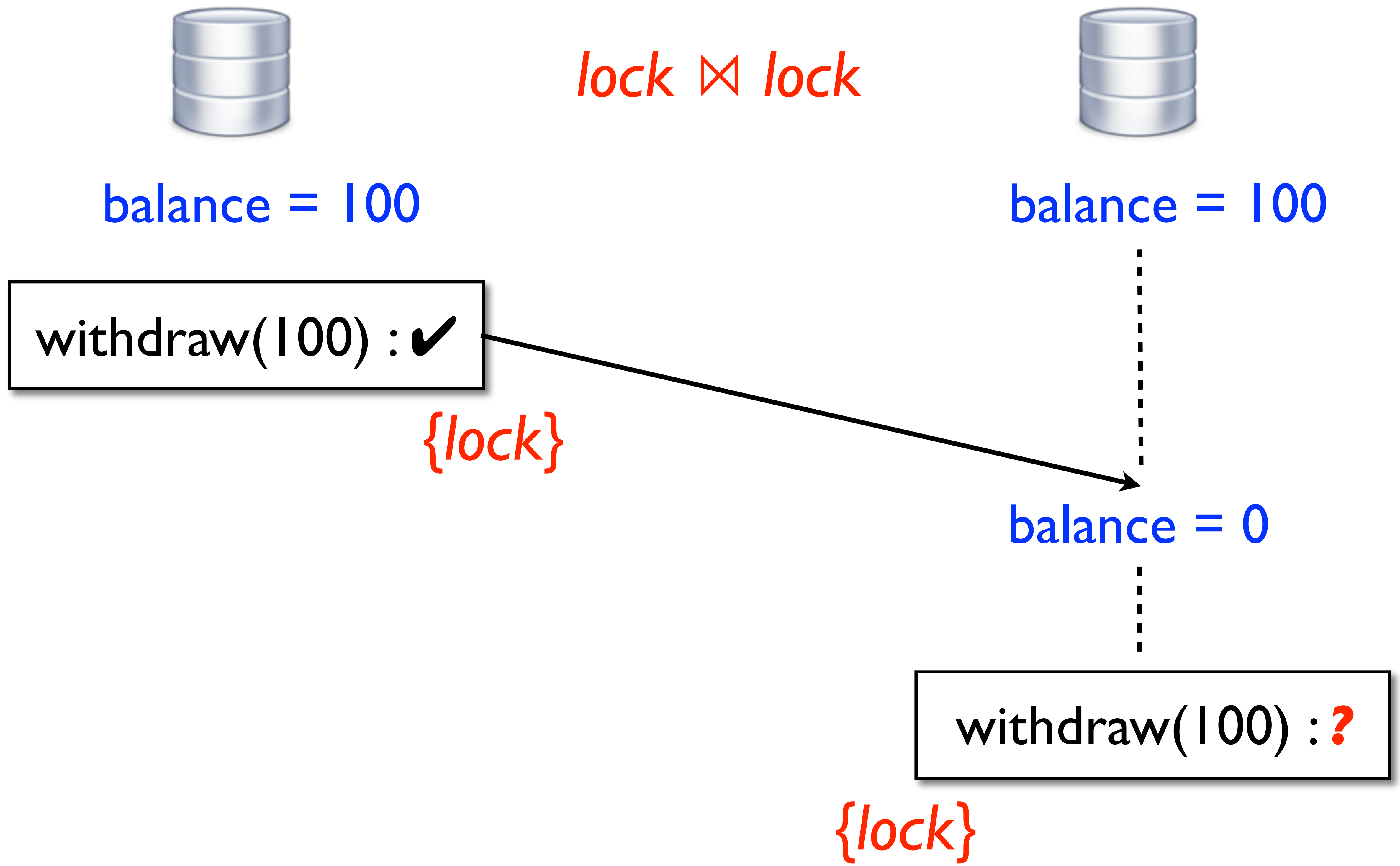
balance = 100

withdraw(100) : ✔

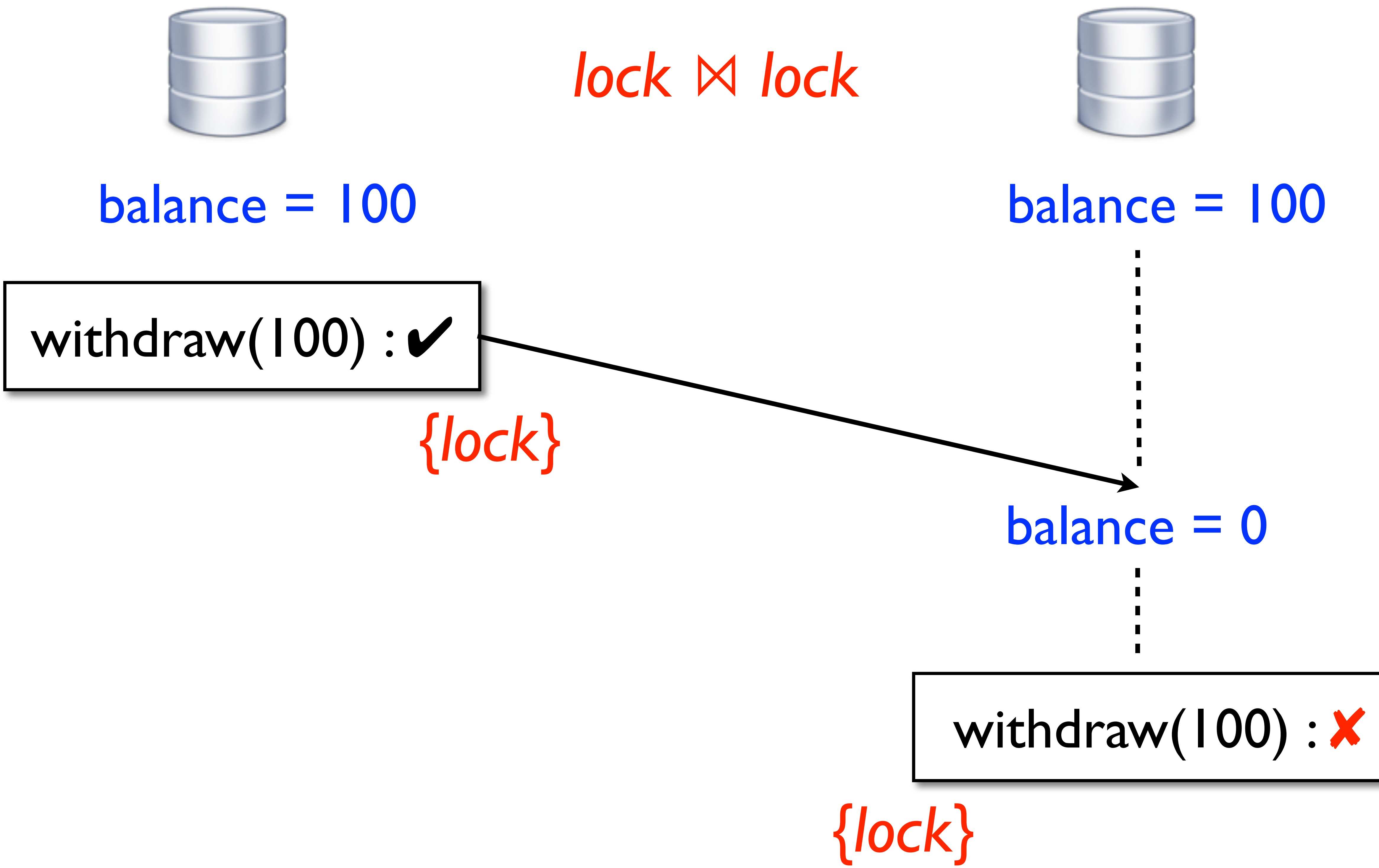
{*lock*}

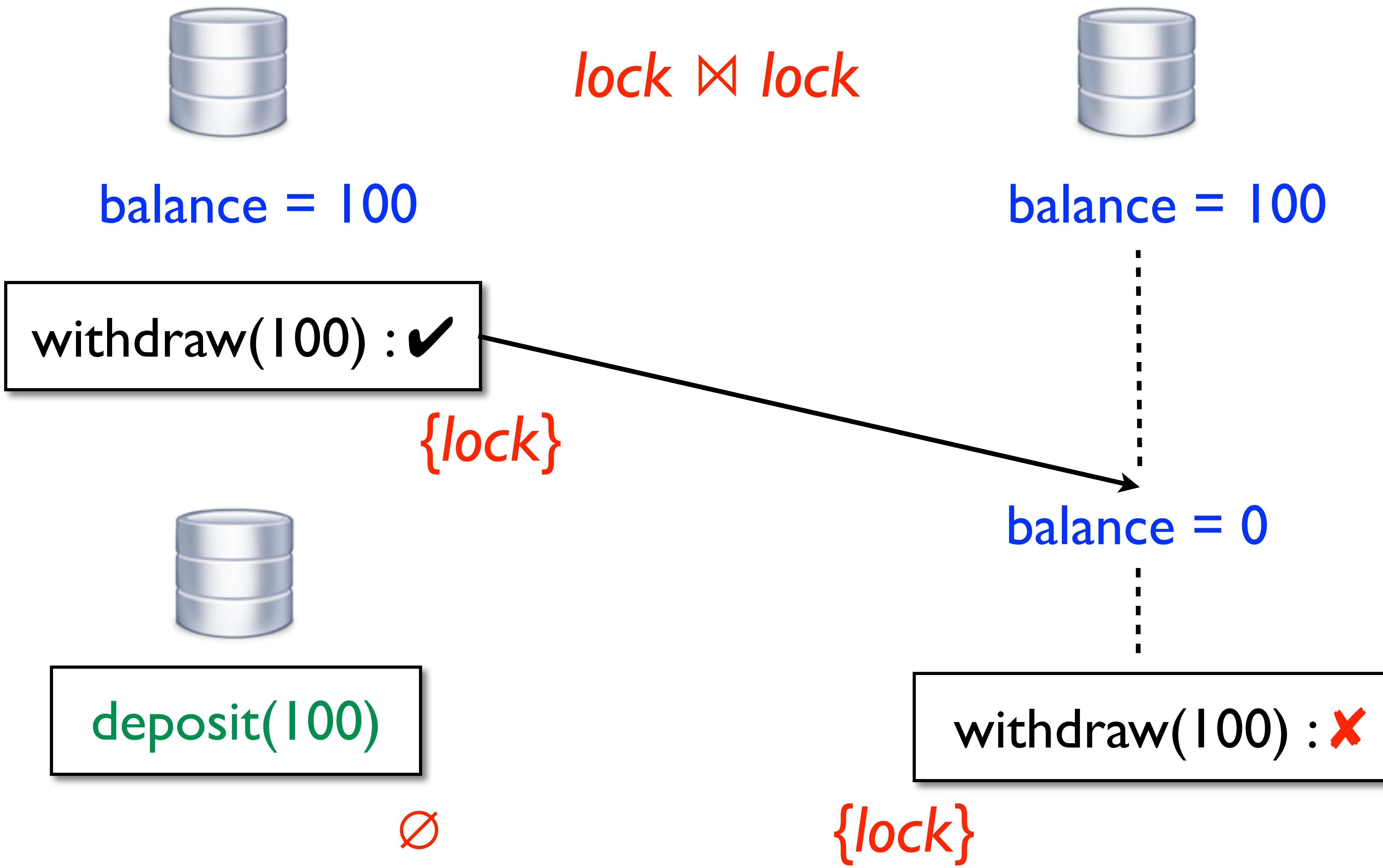


lock ⋈ *lock*
balance = 100
balance = 100
withdraw(100) : ✔
{*lock*}
withdraw(100) : ?
{*lock*}
Anything I don't know about?



lock ⋈ lock
balance = 100
balance = 100
withdraw(100) : ✔
{lock}
balance = 0
withdraw(100) : ?
{lock}


lock ⋈ lock
balance = 100
balance = 100
withdraw(100) : ✔
{lock}
balance = 0
withdraw(100) : ✘
{lock}



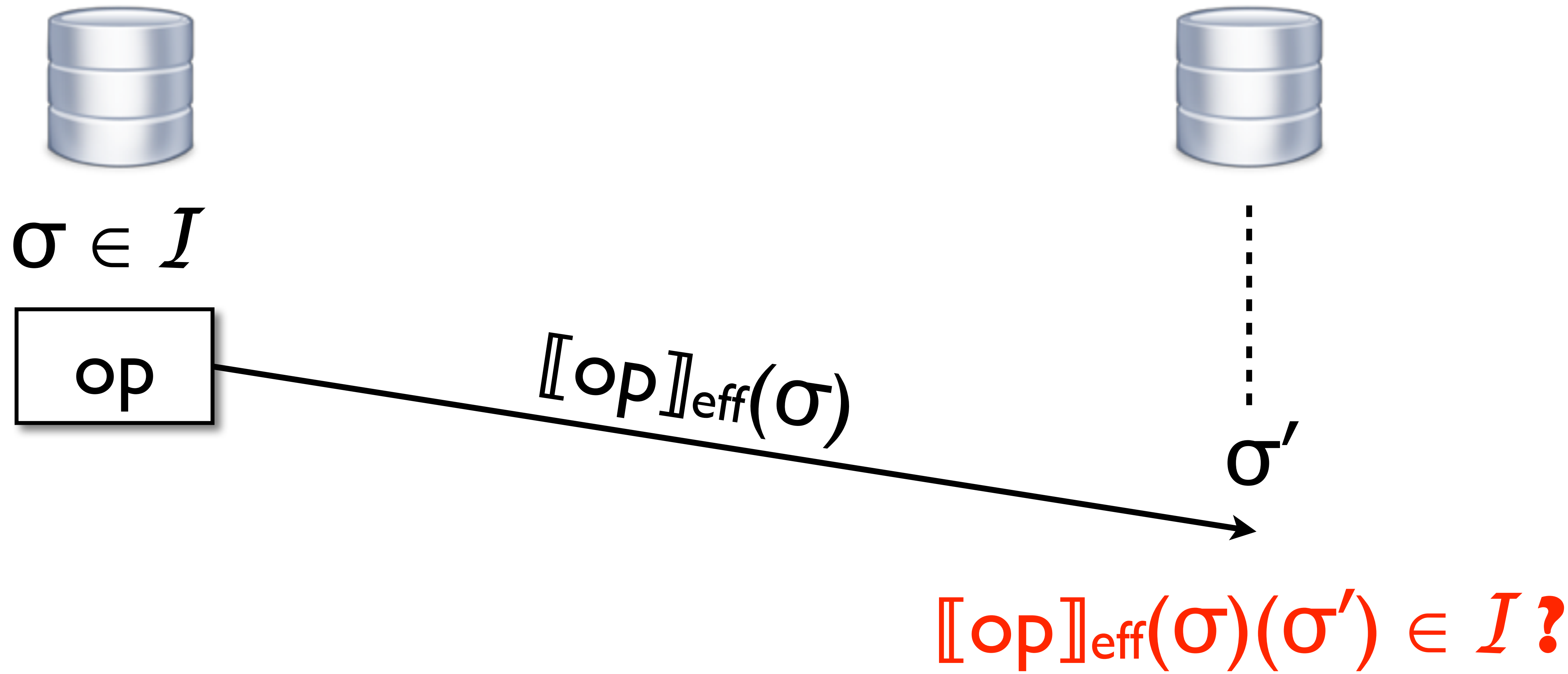
Deposits proceed without synchronisation

Is the invariant preserved?

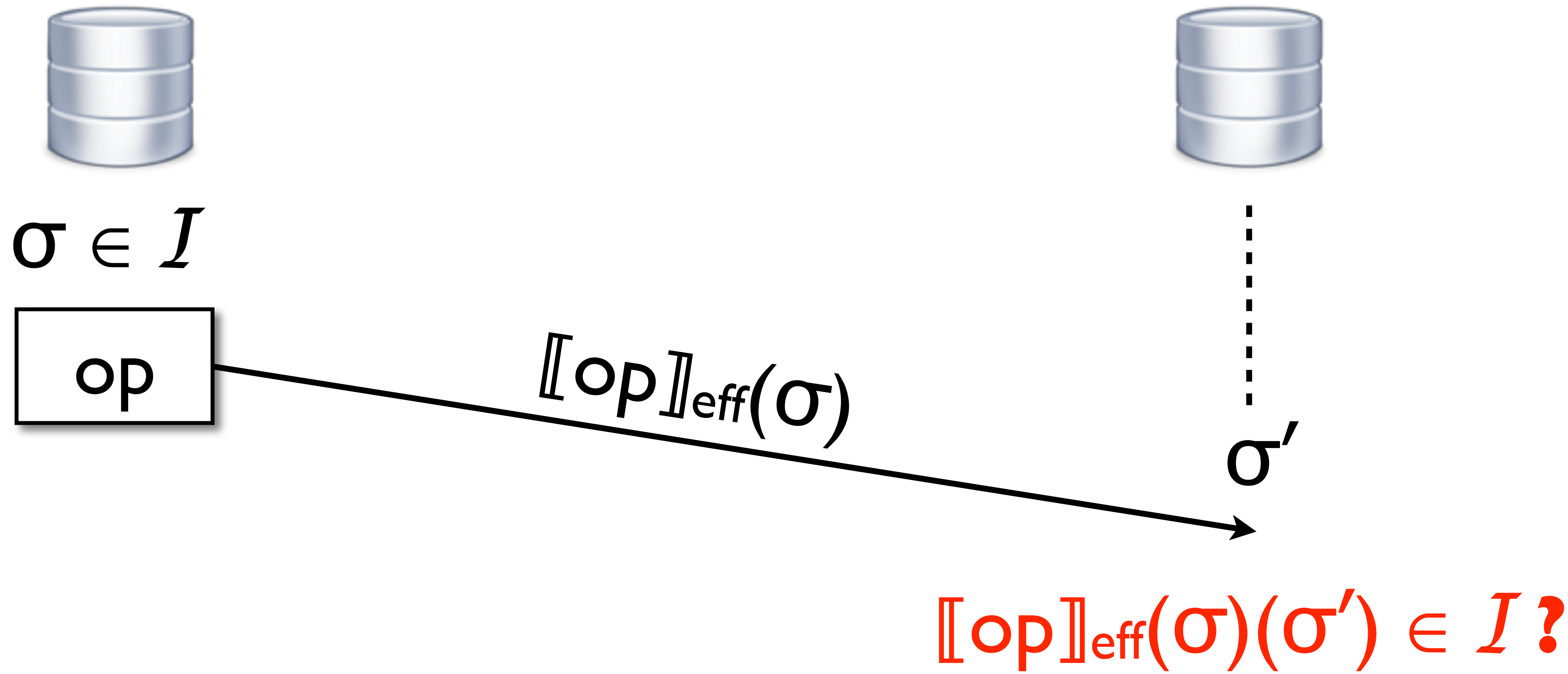
$\sigma \in \mathcal{I}$ ← Assume invariant holds

op

← Check it's preserved after executing op



- Effect applied in a different state!
- Need to constrain possible σ' given σ



- Effect applied in a different state!
- Need to constrain possible σ′ given σ
- Rely-guarantee reasoning: make assumptions about how states of other replicas can change

Guarantee relations

Acquire a token ➜ acquire a permission to change states in a particular way

- $\forall \tau.\ G(\tau) \subseteq State \times State$: changes allowed if acquiring τ

- $G_0 \subseteq State \times State$: changes allowed always

Guarantee relations

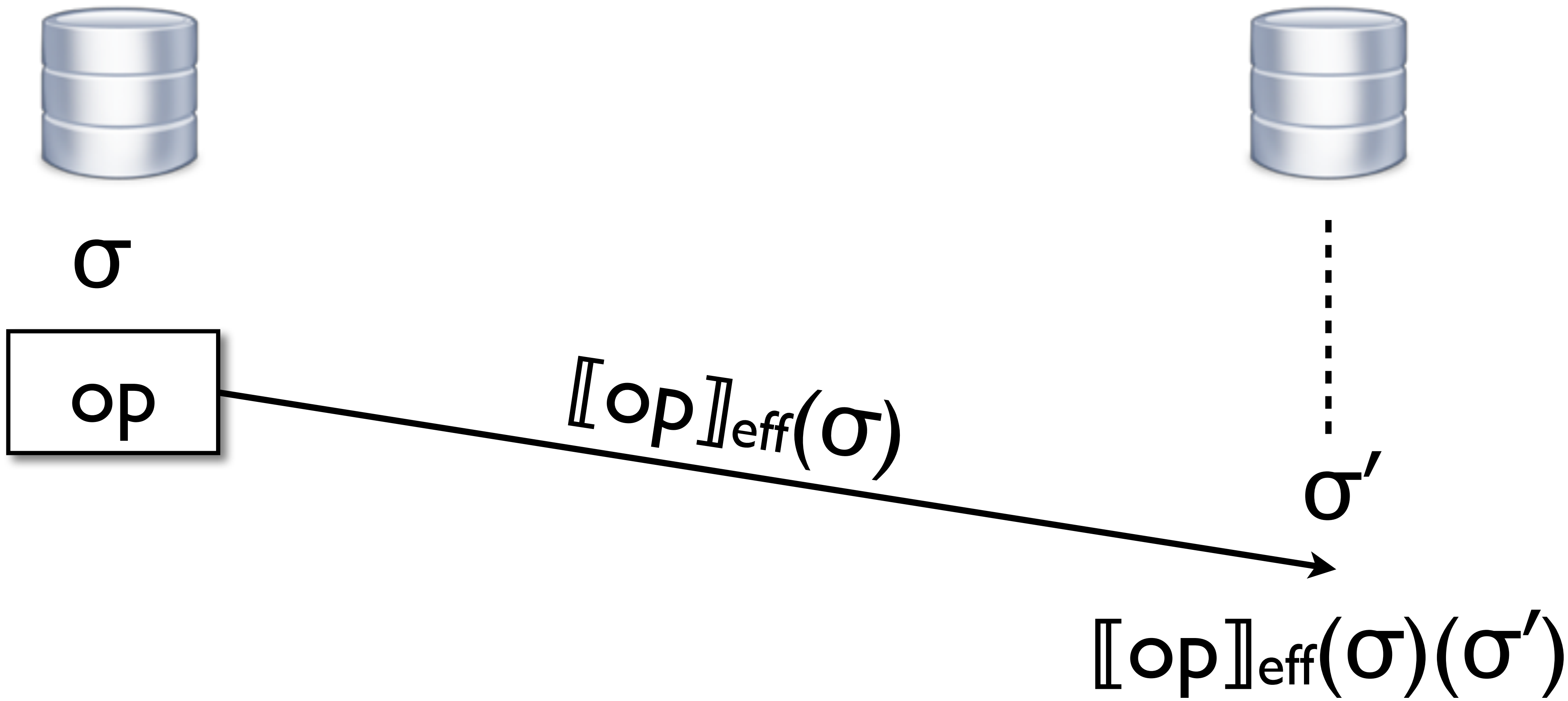
Acquire a token ➜ acquire a permission to change states in a particular way

- $\forall \tau.\ G(\tau) \subseteq \text{State} \times \text{State}$: changes allowed if acquiring τ

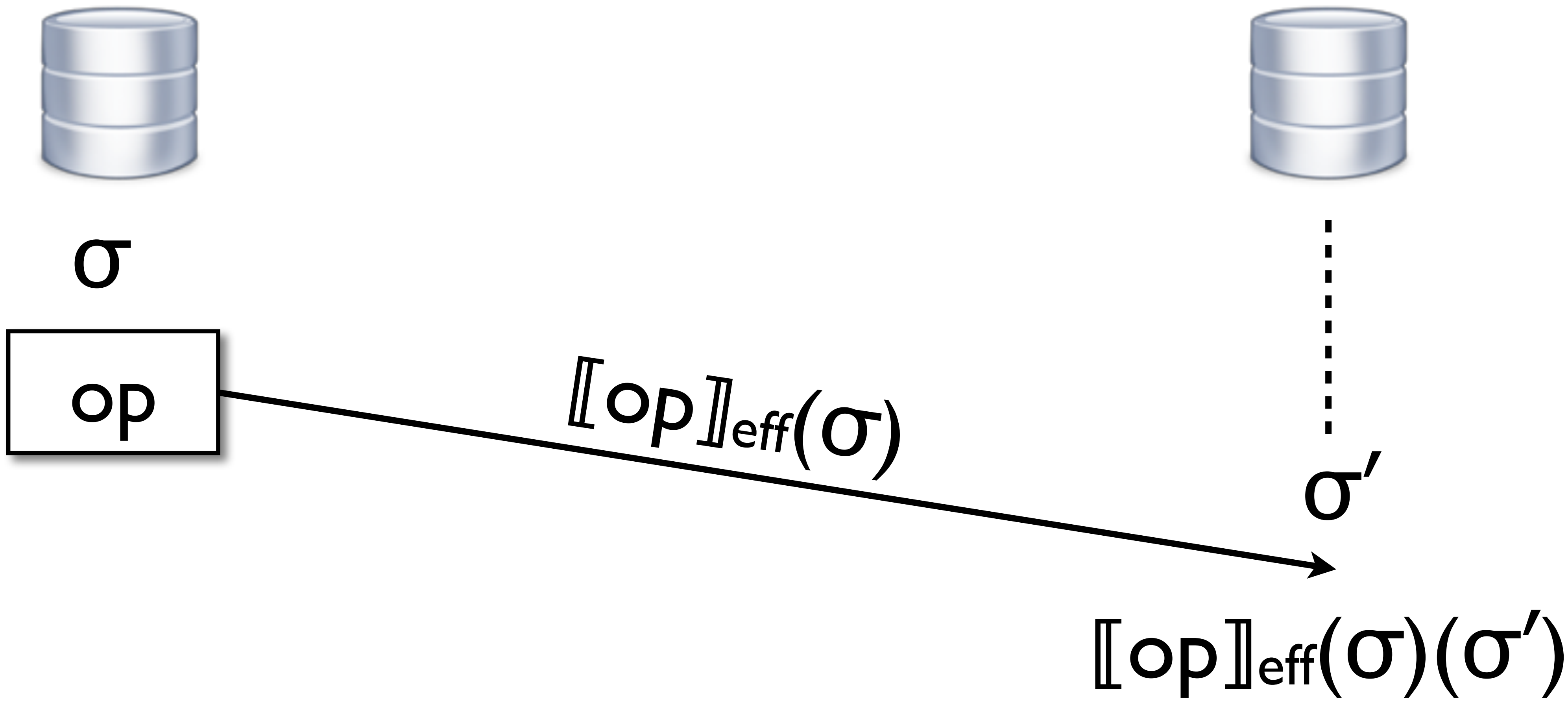
  $G(\textit{lock}) = \{(\sigma_1, \sigma_2) \mid \sigma_2 < \sigma_1\}$

- $G_0 \subseteq \text{State} \times \text{State}$: changes allowed always

  $G_0 = \{(\sigma_1, \sigma_2) \mid \sigma_2 \geq \sigma_1\}$



$$
\begin{aligned}
&\exists G, G_0.\ \forall op.\\
&\forall \sigma, \sigma'.\ \ \sigma \in \mathcal{I}\ \wedge\ (\sigma, \sigma') \in (G_0 \cup G((\llbracket op \rrbracket_{tok}(\sigma))^{\perp}))^{*}\\
&\qquad\qquad \Longrightarrow\ \llbracket op \rrbracket_{eff}(\sigma)(\sigma') \in \mathcal{I}\ \wedge\\
&\qquad\qquad\quad\ (\sigma', \llbracket op \rrbracket_{eff}(\sigma)(\sigma')) \in G_0 \cup G(\llbracket op \rrbracket_{tok}(\sigma))
\end{aligned}
$$



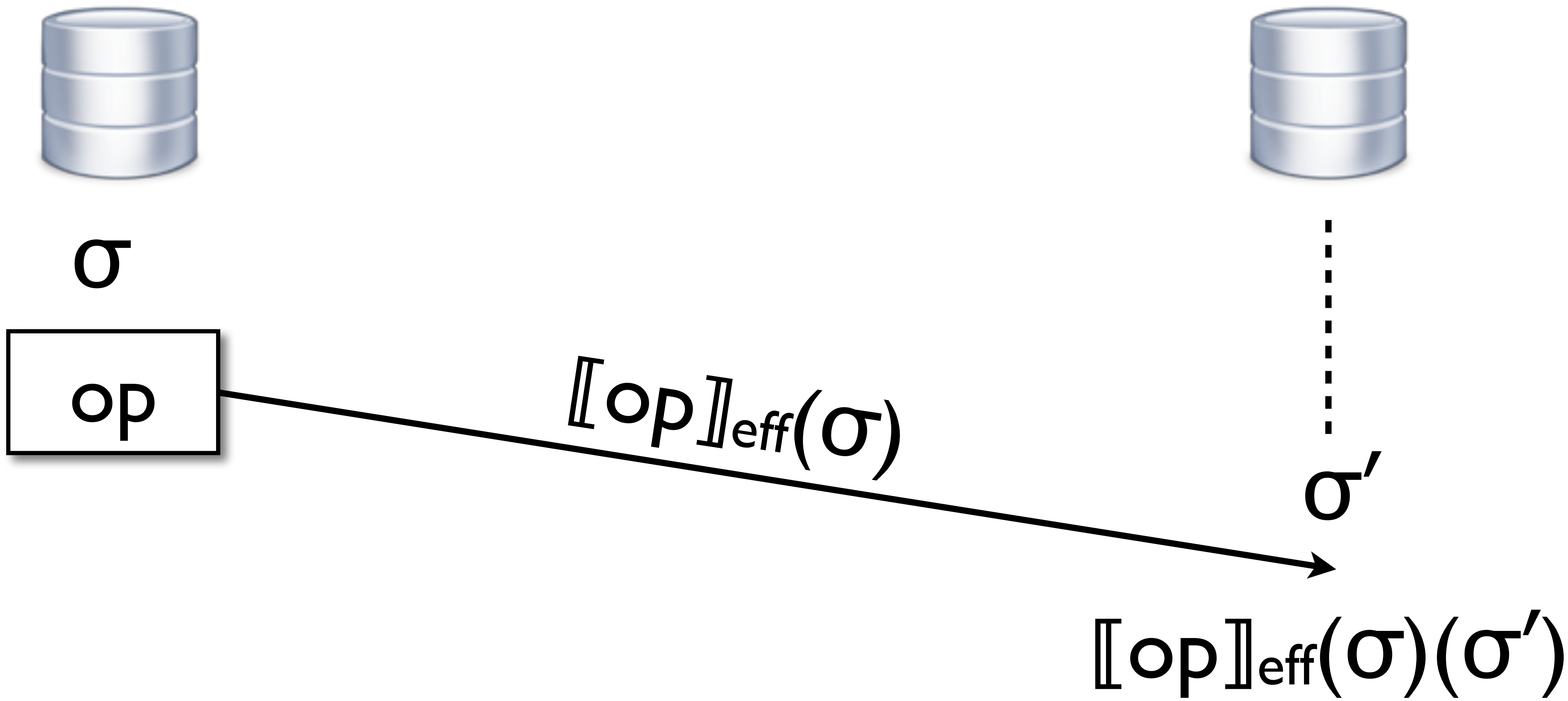
$\exists G, G_0.\ \forall op.$

$$\forall \sigma, \sigma'.\ \sigma \in \mathcal{I} \ \wedge\ (\sigma, \sigma') \in (G_0 \cup G((\llbracket op \rrbracket_{tok}(\sigma))^{\perp}))^*$$

$$\implies \llbracket op \rrbracket_{eff}(\sigma)(\sigma') \in \mathcal{I} \ \wedge$$

$$(\sigma', \llbracket op \rrbracket_{eff}(\sigma)(\sigma')) \in G_0 \cup G(\llbracket op \rrbracket_{tok}(\sigma))$$

Need to find guarantees as part of the proof



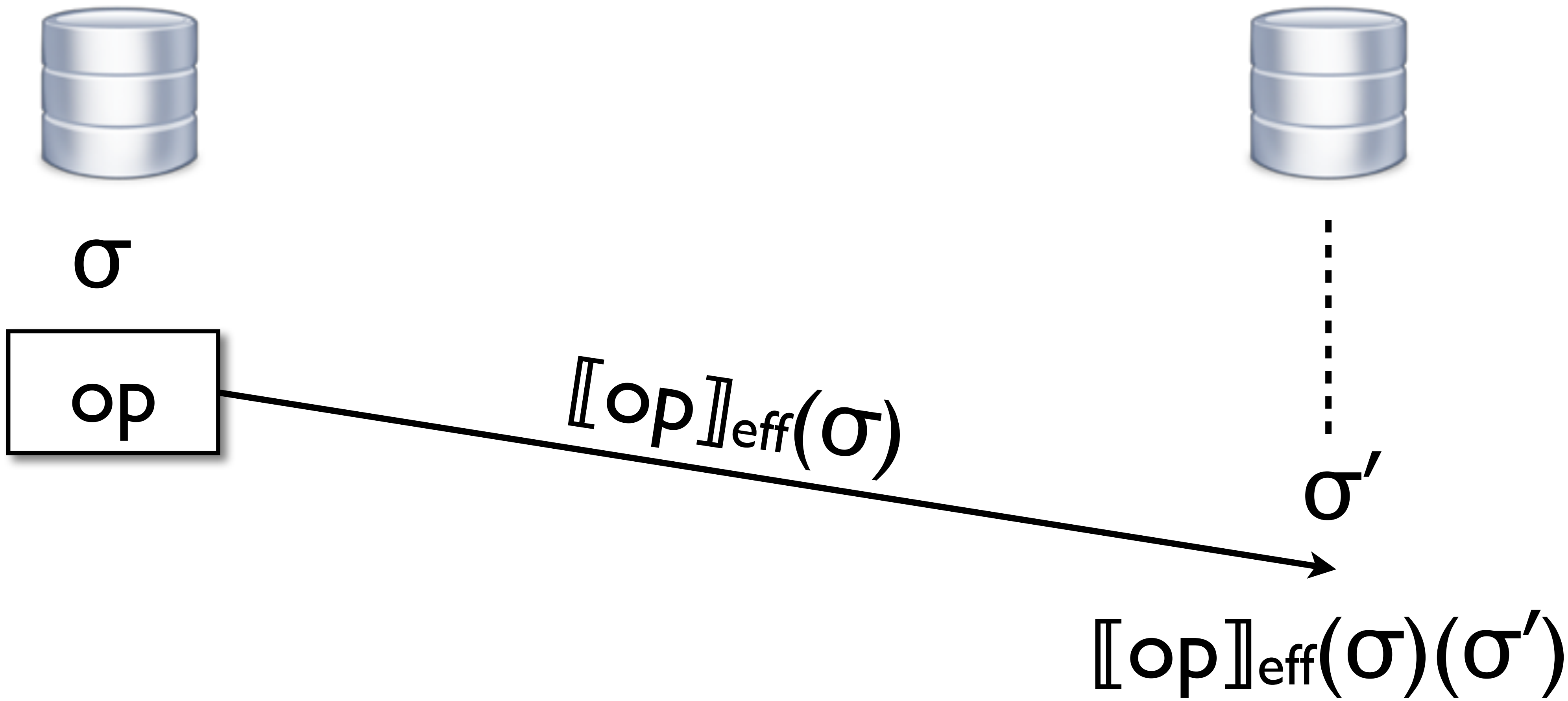
$$\exists G, G_0.\ \forall op.$$

$$\forall \sigma, \sigma'.\ \ \sigma \in \mathcal{I}\ \wedge\ (\sigma, \sigma') \in (G_0 \cup G((\llbracket op \rrbracket_{tok}(\sigma))^{\perp}))^{*}$$

$$\Longrightarrow\ \llbracket op \rrbracket_{eff}(\sigma)(\sigma') \in \mathcal{I}\ \wedge$$

$$(\sigma', \llbracket op \rrbracket_{eff}(\sigma)(\sigma')) \in G_0 \cup G(\llbracket op \rrbracket_{tok}(\sigma))$$

Check $\mathcal{I}$ is preserved after applying op's effect



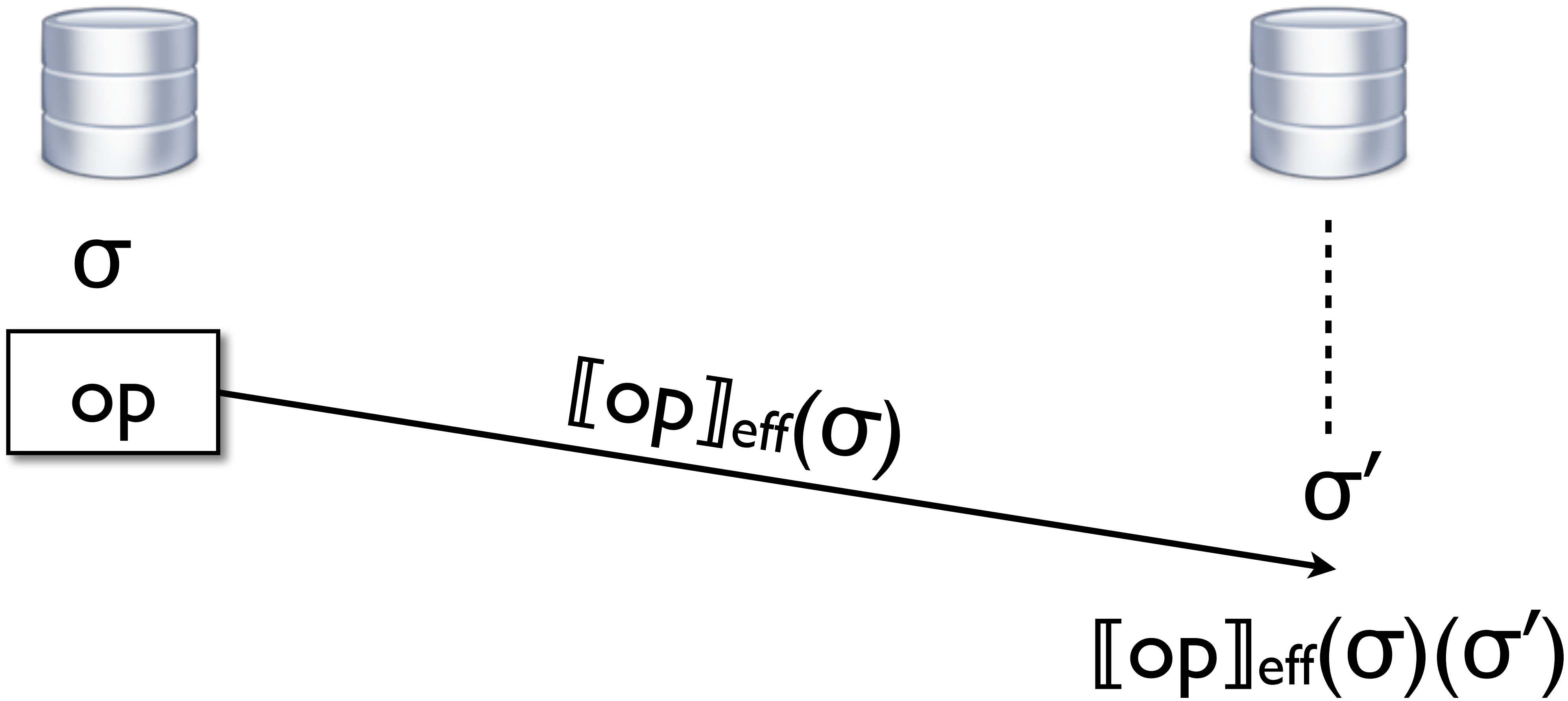
$$\exists G, G_0.\ \forall op.$$

$$\forall \sigma, \sigma'.\ \ \sigma \in \mathcal{I}\ \wedge\ (\sigma, \sigma') \in (G_0 \cup G((\llbracket op \rrbracket_{tok}(\sigma))^{\perp}))^*$$

$$\implies\ \llbracket op \rrbracket_{eff}(\sigma)(\sigma') \in \mathcal{I}\ \wedge$$

$$(\sigma', \llbracket op \rrbracket_{eff}(\sigma)(\sigma')) \in G_0 \cup G(\llbracket op \rrbracket_{tok}(\sigma))$$

Guarantee that op’s effect conforms to G and G_0



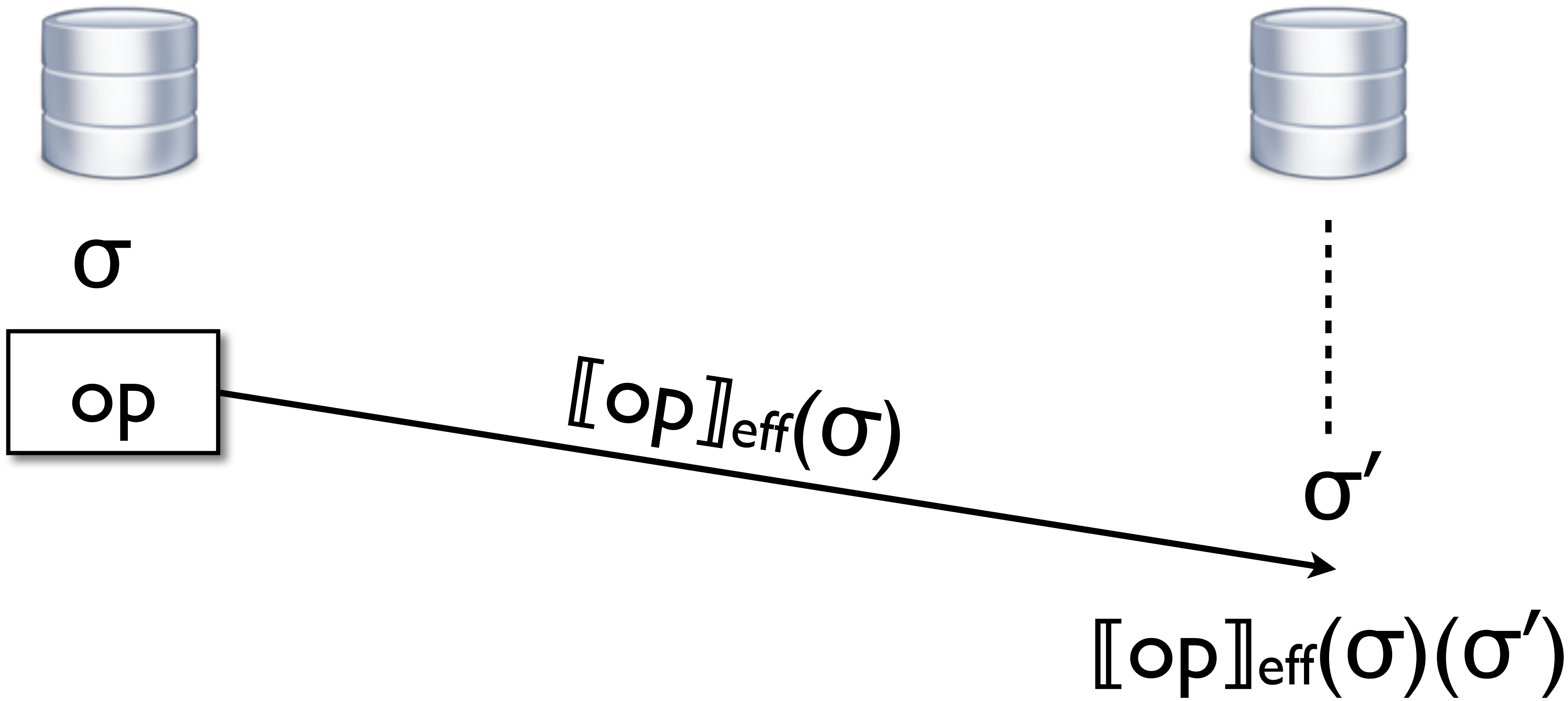
$$\exists G, G_0.\ \forall op.$$

$$\forall \sigma, \sigma'.\ \ \sigma \in \mathcal{I}\ \wedge\ (\sigma, \sigma') \in (G_0 \cup G((\llbracket op \rrbracket_{tok}(\sigma))^{\perp}))^*$$

$$\implies\ \llbracket op \rrbracket_{eff}(\sigma)(\sigma') \in \mathcal{I}\ \wedge$$

$$(\sigma', \llbracket op \rrbracket_{eff}(\sigma)(\sigma')) \in G_0 \cup G(\llbracket op \rrbracket_{tok}(\sigma))$$

op's effect does state changes allowed always or ...



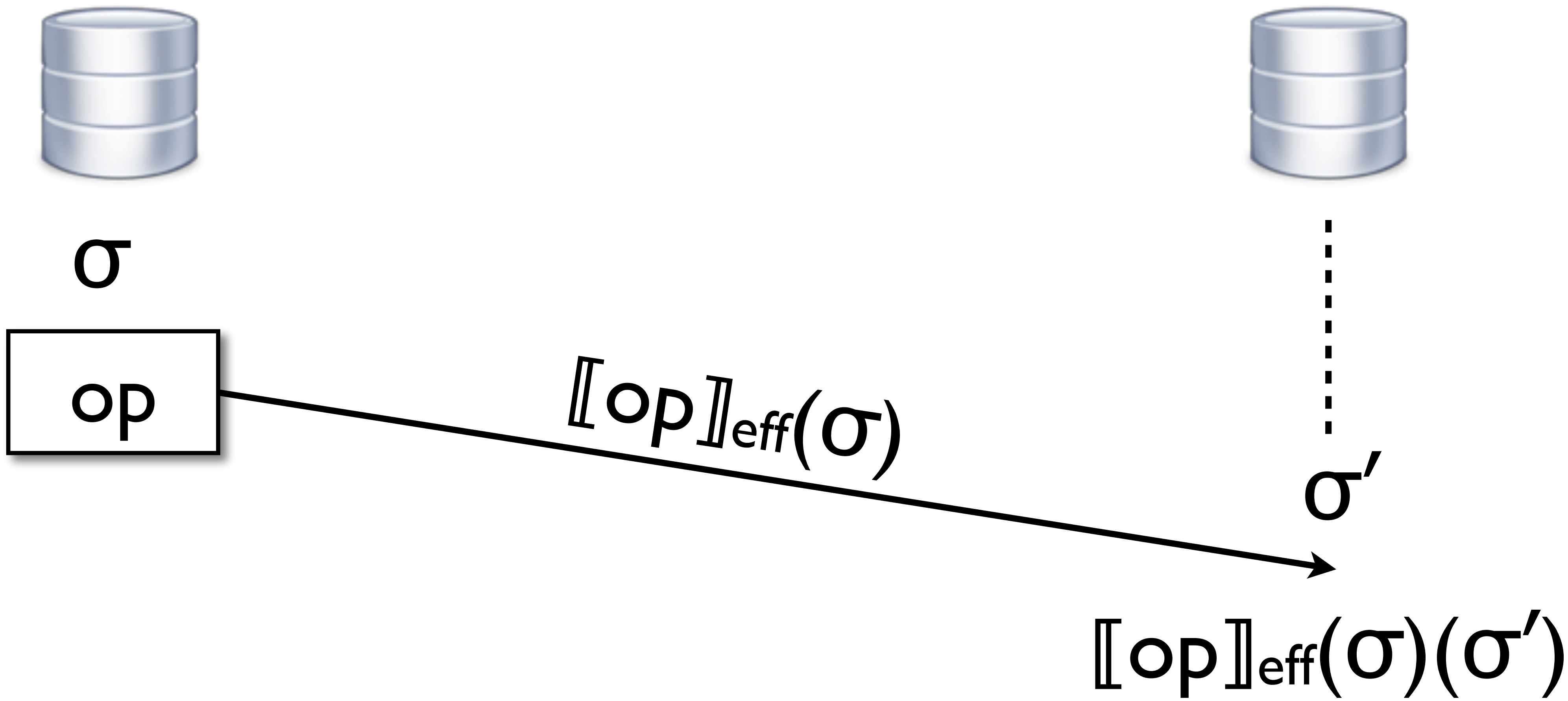
$$\exists G, G_0.\ \forall op.$$

$$\forall \sigma, \sigma'.\ \ \sigma \in \mathcal{I}\ \wedge\ (\sigma, \sigma') \in (G_0 \cup G(([\![op]\!]_{tok}(\sigma))^{\perp}))^{*}$$

$$\implies\ [\![op]\!]_{eff}(\sigma)(\sigma') \in \mathcal{I}\ \wedge$$

$$(\sigma', [\![op]\!]_{eff}(\sigma)(\sigma')) \in G_0 \cup G([\![op]\!]_{tok}(\sigma))$$

...changes allowed by the tokens op acquires



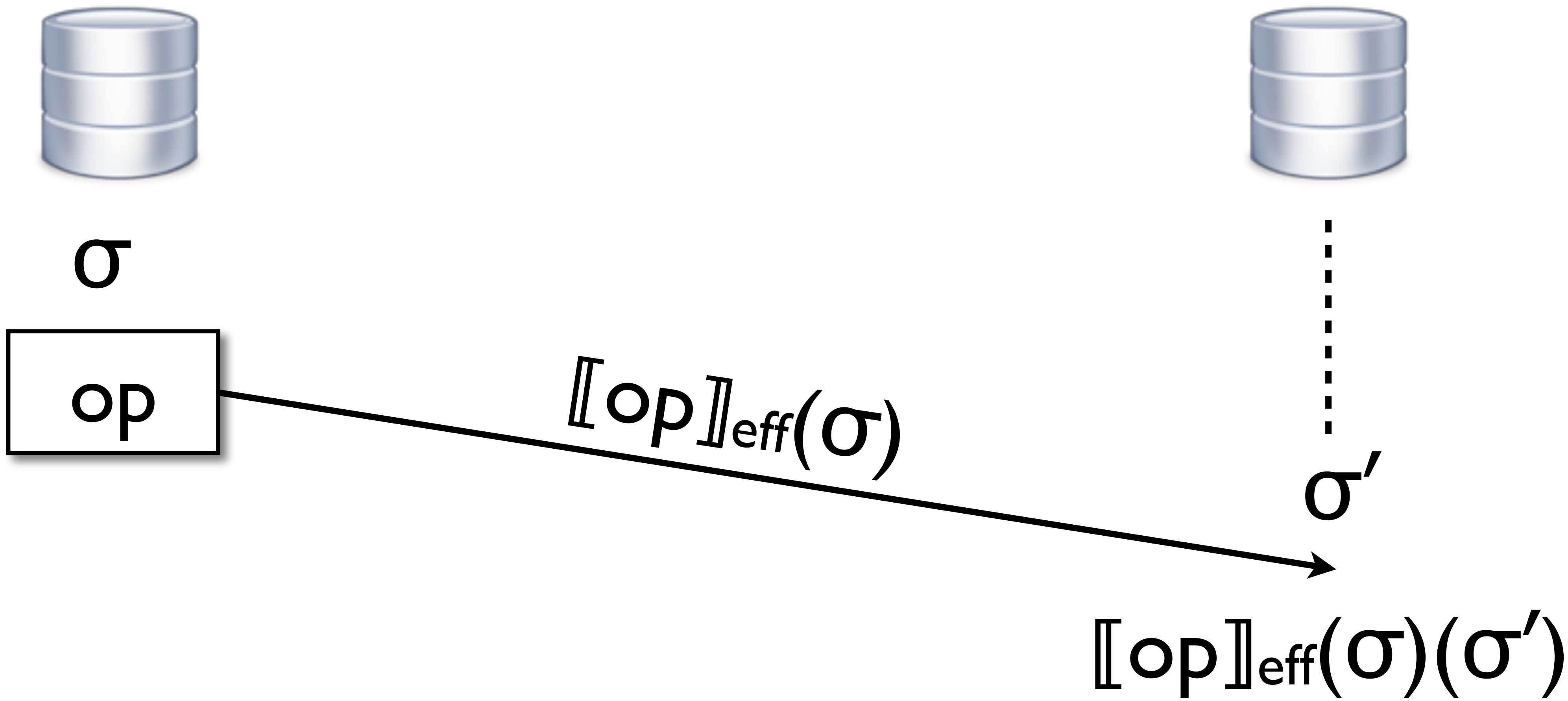
$$\exists G, G_0.\ \forall op.$$

$$\forall \sigma, \sigma'.\ \ \sigma \in \mathcal{I}\ \wedge\ (\sigma, \sigma') \in (G_0 \cup G((⟦op⟧_{tok}(\sigma))^{\perp}))^{*}$$

$$\implies ⟦op⟧_{eff}(\sigma)(\sigma') \in \mathcal{I}\ \wedge$$

$$(\sigma', ⟦op⟧_{eff}(\sigma)(\sigma')) \in G_0 \cup G(⟦op⟧_{tok}(\sigma))$$

Rely on σ and σ′ correlated using G and G_0



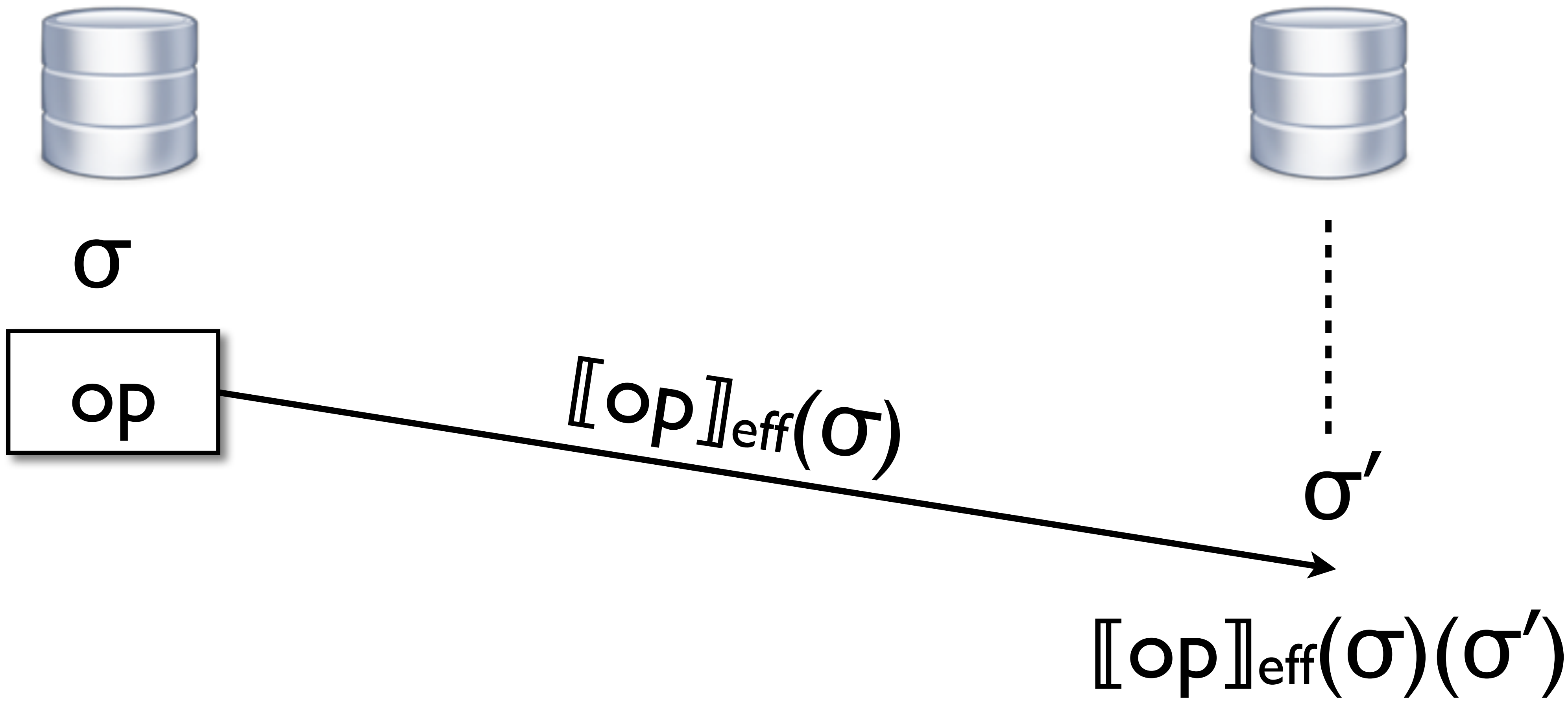
$$\exists G, G_0.\ \forall op.$$

$$\forall \sigma, \sigma'.\ \ \sigma \in \mathcal{I}\ \wedge\ (\sigma, \sigma') \in (G_0 \cup G((\llbracket op \rrbracket_{tok}(\sigma))^{\perp}))^{*}$$

$$\implies \llbracket op \rrbracket_{eff}(\sigma)(\sigma') \in \mathcal{I}\ \wedge$$

$$(\sigma', \llbracket op \rrbracket_{eff}(\sigma)(\sigma')) \in G_0 \cup G(\llbracket op \rrbracket_{tok}(\sigma))$$

Multiple operations may change state



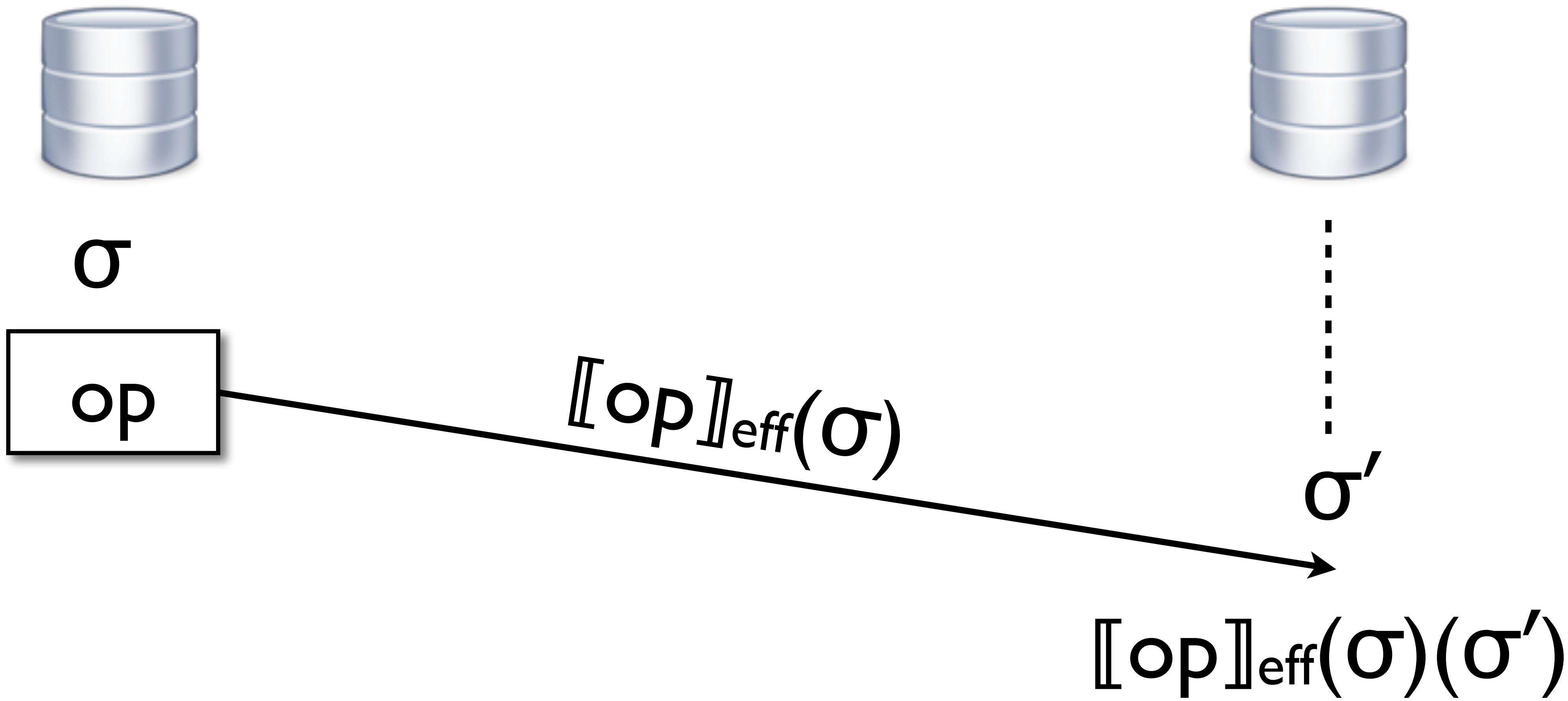
$$\exists G, G_0.\ \forall op.$$

$$\forall \sigma, \sigma'.\ \ \sigma \in \mathcal{I} \ \wedge\ (\sigma, \sigma') \in (G_0 \cup G((\llbracket op \rrbracket_{tok}(\sigma))^{\perp}))^*$$

$$\Longrightarrow\ \llbracket op \rrbracket_{eff}(\sigma)(\sigma') \in \mathcal{I} \ \wedge$$

$$(\sigma', \llbracket op \rrbracket_{eff}(\sigma)(\sigma')) \in G_0 \cup G(\llbracket op \rrbracket_{tok}(\sigma))$$

Concurrent operations make changes allowed always or...



$$\exists G, G_0.\ \forall op.$$

$$\forall \sigma, \sigma'.\ \ \sigma \in \mathcal{I}\ \wedge\ (\sigma, \sigma') \in (G_0 \cup G((\llbracket op \rrbracket_{tok}(\sigma))^{\perp}))^*$$

$$\Longrightarrow\ \llbracket op \rrbracket_{eff}(\sigma)(\sigma') \in \mathcal{I}\ \wedge$$

$$(\sigma', \llbracket op \rrbracket_{eff}(\sigma)(\sigma')) \in G_0 \cup G(\llbracket op \rrbracket_{tok}(\sigma))$$

... changes allowed by guarantees for tokens that don’t conflict with those of op as per ⋈

$$\mathcal{I} = \{\sigma \mid \sigma \geq 0\}$$

$$G(\mathit{lock}) = \{(\sigma_1, \sigma_2) \mid \sigma_2 < \sigma_1\}$$

$$G_0 = \{(\sigma_1, \sigma_2) \mid \sigma_2 \geq \sigma_1\}$$

$$\mathrm{op} = \mathrm{withdraw}(100)$$

$$\forall \sigma, \sigma'.\ \sigma \in \mathcal{I} \ \wedge\ (\sigma, \sigma') \in (G_0 \cup G((\llbracket \mathrm{op} \rrbracket_{\mathrm{tok}}(\sigma))^{\perp}))^{*}$$
$$\implies\ \llbracket \mathrm{op} \rrbracket_{\mathrm{eff}}(\sigma)(\sigma') \in \mathcal{I} \ \wedge$$
$$(\sigma', \llbracket \mathrm{op} \rrbracket_{\mathrm{eff}}(\sigma)(\sigma')) \in G_0 \cup G(\llbracket \mathrm{op} \rrbracket_{\mathrm{tok}}(\sigma))$$

$$\mathcal{I} = \{\sigma \mid \sigma \geq 0\}$$

$$G(\mathit{lock}) = \{(\sigma_1, \sigma_2) \mid \sigma_2 < \sigma_1\}$$

$$G_0 = \{(\sigma_1, \sigma_2) \mid \sigma_2 \geq \sigma_1\}$$

$$\mathrm{op} = \mathrm{withdraw}(100)$$

$$\forall \sigma, \sigma'.\ \sigma \in \mathcal{I} \wedge (\sigma, \sigma') \in (G_0 \cup G((\llbracket \mathrm{op} \rrbracket_{\mathrm{tok}}(\sigma))^{\perp}))^{*}$$

$$\implies \llbracket \mathrm{op} \rrbracket_{\mathrm{eff}}(\sigma)(\sigma') \in \mathcal{I}$$

$$\mathcal{I} = \{\sigma \mid \sigma \geq 0\}$$

$$G(\mathit{lock}) = \{(\sigma_1, \sigma_2) \mid \sigma_2 < \sigma_1\}$$

$$G_0 = \{(\sigma_1, \sigma_2) \mid \sigma_2 \geq \sigma_1\}$$

$$\mathsf{op} = \mathsf{withdraw}(100)$$

$$\forall \sigma, \sigma'.\ \sigma \in \mathcal{I} \ \wedge\ (\sigma, \sigma') \in (G_0 \cup G((\llbracket \mathsf{op} \rrbracket_{\mathsf{tok}}(\sigma))^{\perp}))^{*}$$

$$\implies \llbracket \mathsf{op} \rrbracket_{\mathsf{eff}}(\sigma)(\sigma') \in \mathcal{I}$$

$$\mathcal{I} = \{\sigma \mid \sigma \geq 0\}$$

$$G(\mathit{lock}) = \{(\sigma_1, \sigma_2) \mid \sigma_2 < \sigma_1\}$$

$$G_0 = \{(\sigma_1, \sigma_2) \mid \sigma_2 \geq \sigma_1\}$$

$$\mathrm{op} = \mathrm{withdraw}(100)$$

$$\forall \sigma, \sigma'.\ \sigma \in \mathcal{I} \wedge (\sigma, \sigma') \in (G_0 \cup G(\overbrace{(\llbracket \mathrm{op} \rrbracket_{\mathrm{tok}}(\sigma))^{\perp}}^{\emptyset}))^{*}$$

$$\Longrightarrow \llbracket \mathrm{op} \rrbracket_{\mathrm{eff}}(\sigma)(\sigma') \in \mathcal{I}$$

$$\mathit{lock} \bowtie \mathit{lock}$$

$$\mathcal{I} = \{\sigma \mid \sigma \geq 0\}$$

$$G(\mathit{lock}) = \{(\sigma_1, \sigma_2) \mid \sigma_2 < \sigma_1\}$$

$$G_0 = \{(\sigma_1, \sigma_2) \mid \sigma_2 \geq \sigma_1\}$$

$$\mathrm{op} = \mathrm{withdraw}(100)$$

G_0^*

$\varnothing$

$$\forall \sigma, \sigma'.\ \sigma \in \mathcal{I} \wedge (\sigma, \sigma') \in (G_0 \cup G((\llbracket \mathrm{op} \rrbracket_{\mathrm{tok}}(\sigma))^{\perp}))^* \implies \llbracket \mathrm{op} \rrbracket_{\mathrm{eff}}(\sigma)(\sigma') \in \mathcal{I}$$

$\mathit{lock} \bowtie \mathit{lock}$

$$\mathcal{I} = \{\sigma \mid \sigma \geq 0\}$$

$$G(\textit{lock}) = \{(\sigma_1, \sigma_2) \mid \sigma_2 < \sigma_1\}$$

$$G_0 = \{(\sigma_1, \sigma_2) \mid \sigma_2 \geq \sigma_1\}$$

$$\text{op} = \text{withdraw}(100)$$

$$\forall \sigma, \sigma'.\ \sigma \in \mathcal{I} \wedge (\sigma, \sigma') \in G_0^* \implies [\![\text{op}]\!]_{\text{eff}}(\sigma)(\sigma') \in \mathcal{I}$$

$$I = \{\sigma \mid \sigma \geq 0\}$$

$$G(lock) = \{(\sigma_1, \sigma_2) \mid \sigma_2 < \sigma_1\}$$

$$G_0 = \{(\sigma_1, \sigma_2) \mid \sigma_2 \geq \sigma_1\}$$

$$op = \text{withdraw}(100)$$

$$\sigma' \geq \sigma$$

$$\forall \sigma, \sigma'.\ \sigma \in I \wedge (\sigma, \sigma') \in G_0^*$$

$$\Longrightarrow [\![op]\!]_{eff}(\sigma)(\sigma') \in I$$

Balance at a destination replica as high as balance at the origin replica

$$\mathcal{I} = \{\sigma \mid \sigma \geq 0\}$$

$$G(\mathit{lock}) = \{(\sigma_1, \sigma_2) \mid \sigma_2 < \sigma_1\}$$

$$G_0 = \{(\sigma_1, \sigma_2) \mid \sigma_2 \geq \sigma_1\}$$

$$\mathrm{op} = \mathrm{withdraw}(100)$$

$\sigma \geq 0$ $\sigma' \geq \sigma$

$$\forall \sigma, \sigma'.\ \sigma \in \mathcal{I} \ \wedge \ (\sigma, \sigma') \in G_0^*$$

$$\implies [\![\mathrm{op}]\!]_{\mathrm{eff}}(\sigma)(\sigma') \in \mathcal{I}$$

$$\mathcal{I} = \{\sigma \mid \sigma \geq 0\}$$

$$G(\mathit{lock}) = \{(\sigma_1, \sigma_2) \mid \sigma_2 < \sigma_1\}$$

$$G_0 = \{(\sigma_1, \sigma_2) \mid \sigma_2 \geq \sigma_1\}$$

$$op = \text{withdraw}(100)$$

$\sigma \geq 0$ $\qquad$ $\sigma' \geq \sigma$

$$\forall \sigma, \sigma'.\ \sigma \in \mathcal{I} \wedge (\sigma, \sigma') \in G_0^* \implies [\![op]\!]_{\text{eff}}(\sigma)(\sigma') \in \mathcal{I}$$

$(\text{if } \sigma \geq 100 \text{ then } \sigma' - 100 \text{ else } \sigma') \geq 0$

$$\mathcal{I} = \{\sigma \mid \sigma \geq 0\}$$

$$G(\mathit{lock}) = \{(\sigma_1, \sigma_2) \mid \sigma_2 < \sigma_1\}$$

$$G_0 = \{(\sigma_1, \sigma_2) \mid \sigma_2 \geq \sigma_1\}$$

$$\mathrm{op} = \mathrm{withdraw}(100)$$

$\sigma \geq 0$

$\sigma' \geq \sigma$

$$\forall \sigma, \sigma'.\ \sigma \in \mathcal{I} \wedge (\sigma, \sigma') \in G_0^* \implies [\![\mathrm{op}]\!]_{\mathrm{eff}}(\sigma)(\sigma') \in \mathcal{I}$$

✔

(if $\sigma \geq 100$ then $\sigma' - 100$ else σ') ≥ 0

If there was enough money at the origin replica,
there will be enough money at a destination replica

Soundness

- Proved soundness of the proof rule
- Nontrivial: depends on causal consistency and effect commutativity
- Soundness by compilation into an event-based proof rule: uses structures for specifying eventual consistency [POPL'14]

Prototype tool

- Automates the proof rule
- Discharges verification conditions using SMT
- Case studies: fragments of several web applications

Conclusion

- Lots of logics for shared-memory concurrency
  - Owicki-Gries [1976]
  - Rely-guarantee [Jones 1983, Pnueli 1985]
  - Concurrent separation logic [O'Hearn 2004]
  - RGSep/SAGL [Vafeiadis+ 2007, Feng+ 2007]
  - Concurrent abstract predicates [Dinsdale-Young+ 2010]
  - Higher-order CAP [Svendsen+ 2013]
  - CaReSL [Turon+ 2013]
  - Fine-grained concurrent separation logic [Nanevski+ 2014]
  - Iris [Jung+ 2015]
  - ...

Conclusion

- Lots of logics for shared-memory concurrency
- Almost none for distributed systems

Conclusion

- Lots of logics for shared-memory concurrency
- Almost none for distributed systems
- Clean, modular reasoning principles still applicable: rely-guarantee reasoning
- Starting point for research in distributed systems verification