Understanding Eventual Consistency

Hongseok Yang
University of Oxford

Joint work with Sebastian Burckhardt (MSR), Alexey Gotsman (IMDEA)

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

Reader

amazon.co.uk | HONGSEOK's Amazon | Today's Deals | Gift Cards | Sell | Help

January Deals › Shop now

Shop by Department | Search | All | Go | Hello, HONGSEOK Your Account | 2 Basket | Wish List

- MP3s & Cloud Player
  20 million songs, play anywhere
- Amazon Cloud Drive
  5 GB of free storage
- Appstore for Android
  Get a paid app for free every day
- Books
- Music, Games, Film & TV
- Kindle
- Electronics
- Computers & Office
- Home, Garden & Pets
- Toys, Children & Baby
- Clothes, Shoes & Watches
- Sports & Outdoors
- Grocery, Health & Beauty
- DIY, Tools & Car

› Full Shop Directory

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

Meet the Kindle Family

Kindle › £69 | Kindle Paperwhite › from £109 | Kindle Fire HD › from £159

Protect and personalise your Kindle

› Kindle Accessories

Amazon Family | Trade-In | Clothing Store | Amazon Prime | Subscribe & Save

FASHION DEALS

› Clothing up to 60% Off
› Shoes & Bags up to 60% Off
› Jewellery up to 70% Off
› Watches up to 75% Off

New Year, New Resolutions › See more

Join amazonfamily

Save up to £50 every month and get three months' FREE One-Day Delivery

Just some of the benefits of Amazon Family
› Become a member

Advertisement

Customised Photo Calendar with Delivery £8

› See the deal | amazon local

SHOP OUR MEN'S JEWELLERY STORE

› Shop now

More Items to Consider

You viewed | Customers who viewed this also viewed

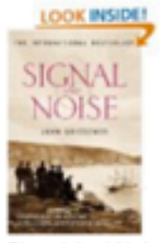
The Signal and the Noise: The Art and...
› Nate Silver
Hardcover
★★★★☆ (18)

Thinking Statistically
› Uri Bram
Paperback
★★★★★ (9)

Antifragile: How to Live in a World...
› Nassim Taleb
Hardcover
★★★★☆ (26)

Bad Pharma: How drug companies...
› Ben Goldacre
Paperback
★★★★½ (97)

The Theory That Would Not Die: How...
› Sharon Bertsch McGrayne
Paperback
★★★½☆ (9)



Signal And Noise
› John Griesemer
Paperback
★★★★☆ (1)

Particle Physics: A Very Short...

In this compelling introduction to the fundamental particles that make up the universe, Frank... Read more

£7.99 £5.03

Geo-replicated systems

- Every data centre stores a complete replica of data
- Purpose: fault tolerance, minimising latency

Geo-replicated systems

- Every data centre stores a complete replica of data
- Purpose: fault tolerance, minimising latency

Geo-replicated systems

- Every data centre stores a complete replica of data
- Purpose: fault tolerance, minimising latency

Data consistency problem

c.inc()

- Any user can access any replica
- Updating only one replica leads to weak consistency - allows strange behaviours

Data consistency problem

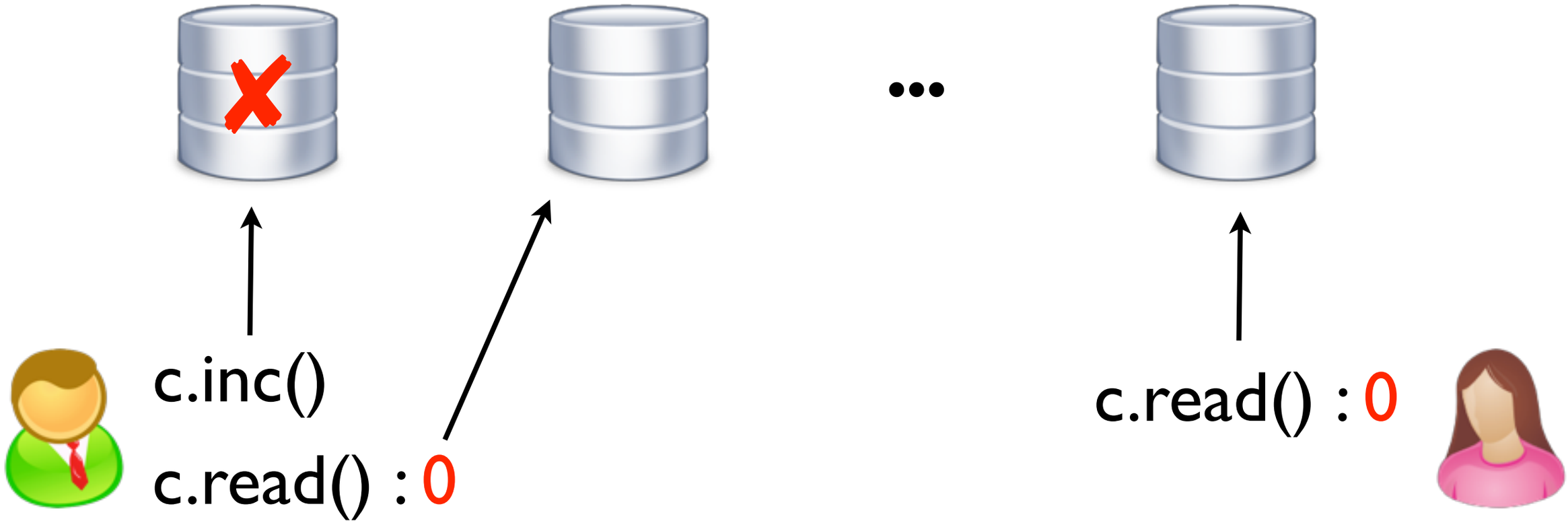
...

c.inc()

c.read() : 0

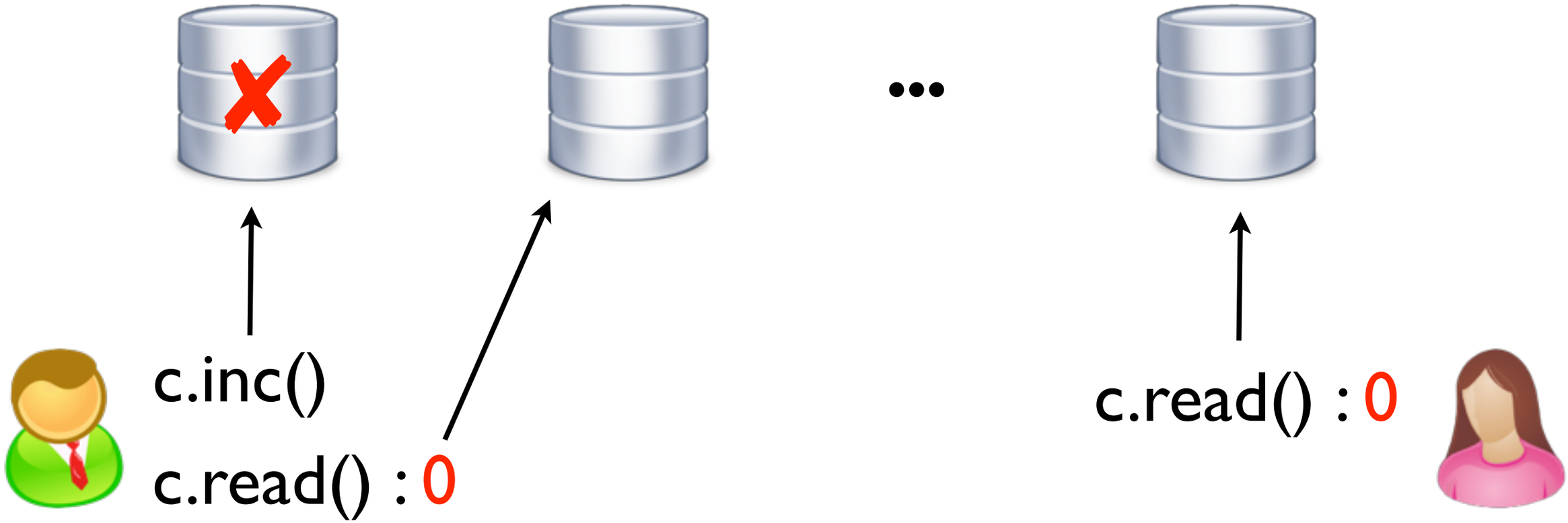
- Any user can access any replica
- Updating only one replica leads to weak consistency - allows strange behaviours

Data consistency problem



- Any user can access any replica
- Updating only one replica leads to weak consistency - allows strange behaviours

Data consistency problem



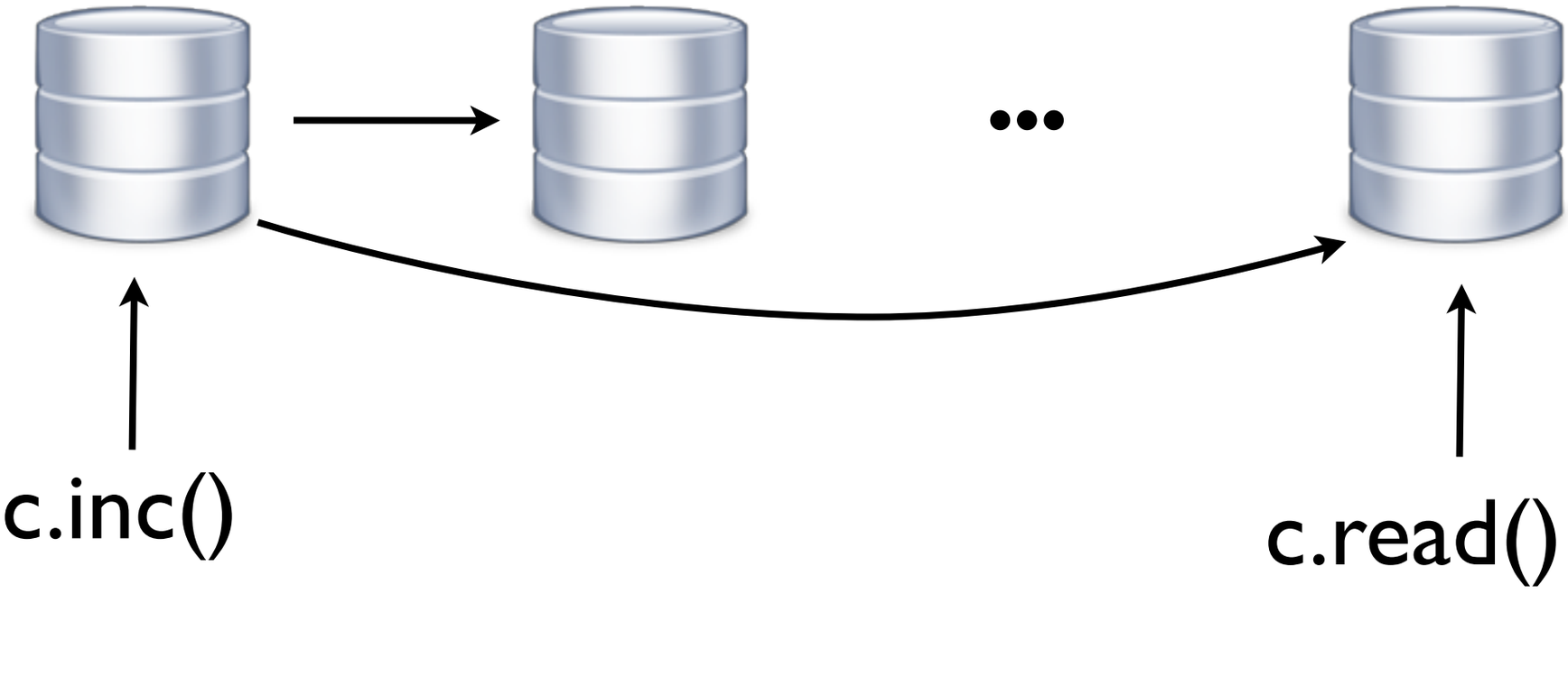
- Any user can access any replica
- Updating only one replica leads to weak consistency - allows strange behaviours

Data consistency problem

c.inc()

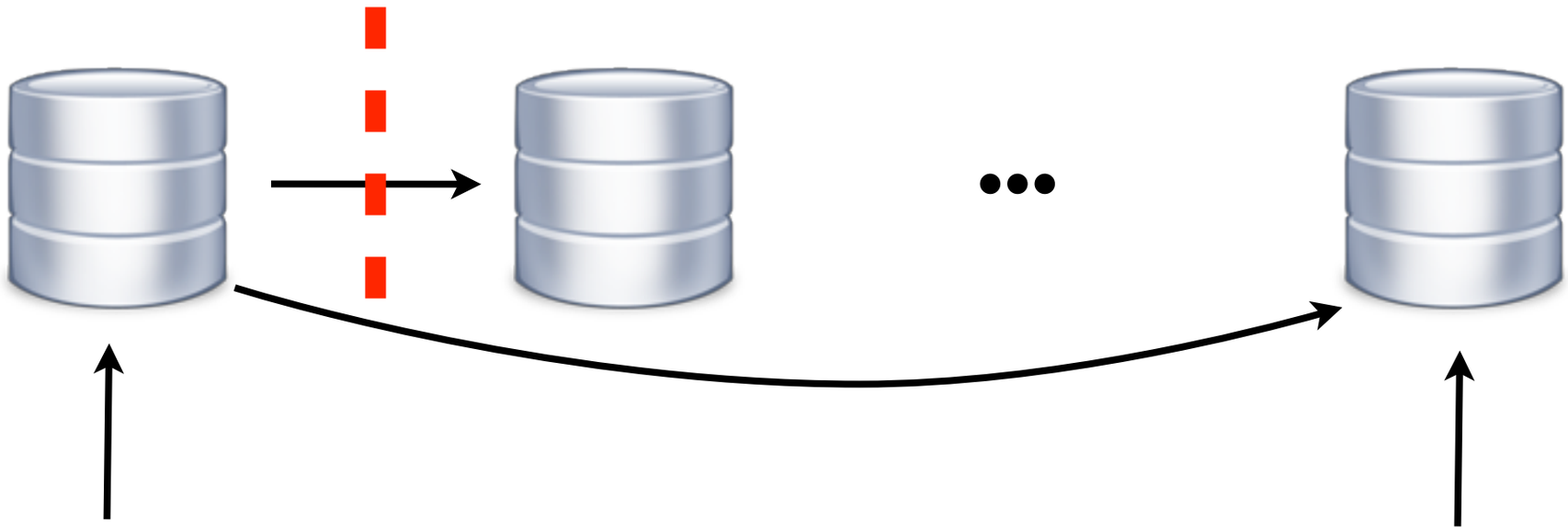
- Strong consistency requires updating multiple replicas before completing a user request
- Problematic in geo-replicated systems: links may go down ⇒ the network partitions

Data consistency problem



- Strong consistency requires updating multiple replicas before completing a user request
- Problematic in geo-replicated systems: links may go down ⇒ the network partitions

Data consistency problem



- Strong consistency requires updating multiple replicas before completing a user request
- Problematic in geo-replicated systems: links may go down ⇒ the network partitions

Data consistency problem

c.inc()

c.read() : I

Also an issue with mobile devices

Data consistency problem

- Update other replicas later, when the connectivity is restored: give up strong consistency
- Wait until the connectivity is restored: give up availability
- Assume network partitions don't happen: give up partition tolerance

Data consistency problem

- Update other replicas later, when the connectivity is restored: give up strong **C**onsistency
- Wait until the connectivity is restored: give up **A**vailability
- Assume network partitions don't happen: give up **P**artition tolerance

CAP Theorem [Lynch&Gilbert]:
Achieving all three at once is impossible

Data consistency problem

- Update other replicas later, when the connectivity is restored: give up ~~strong **C**onsistency~~ *eventual consistency*
- Wait until the connectivity is restored: give up **A**vailability
- Assume network partitions don't happen: give up **P**artition tolerance

CAP Theorem [Lynch&Gilbert]:
Achieving all three at once is impossible

"If no new updates are made to the object, eventually all accesses will return the last updated value"

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.[2]

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 object, eventually all accesses will return the last updated value"

But updates never stop!

So what does this tell to me as a client?

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.[2]

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-

50 shades of eventual consistency

Session Guarantees for Weakly Consistent Replicated Data

Douglas B. Terry, Alan J. Demers, Karin Petersen, Mike J. Spreitzer, Marvin M. Theimer, and Brent B. Welch

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

Strengthen consistency (somewhat)

Conflict-free Replicated Data Types *

Marc Shapiro[1,5], Nuno Preguiça[2,1], Carlos Baquero[3], and Marek Zawirski[1,4]

[1] INRIA, Paris, France
[2] CITI, Universidade Nova de Lisboa, Portugal
[3] Universidade do Minho, Portugal
[4] UPMC, Paris, France
[5] LIP6, Paris, France

Transactional storage for geo-replicated systems

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

Add features that make coping with weak consistency easier

Problem

- Different formalisms/levels of abstraction: how do I compare systems?
- Tied to implementation: what do I tell the programmer/verification person?
- How do I combine different features/ explore the design space?

Need a formal and declarative definition for the semantics of eventually consistent systems

1. How do we resolve conflicts resulting from concurrent updates at different replicas?

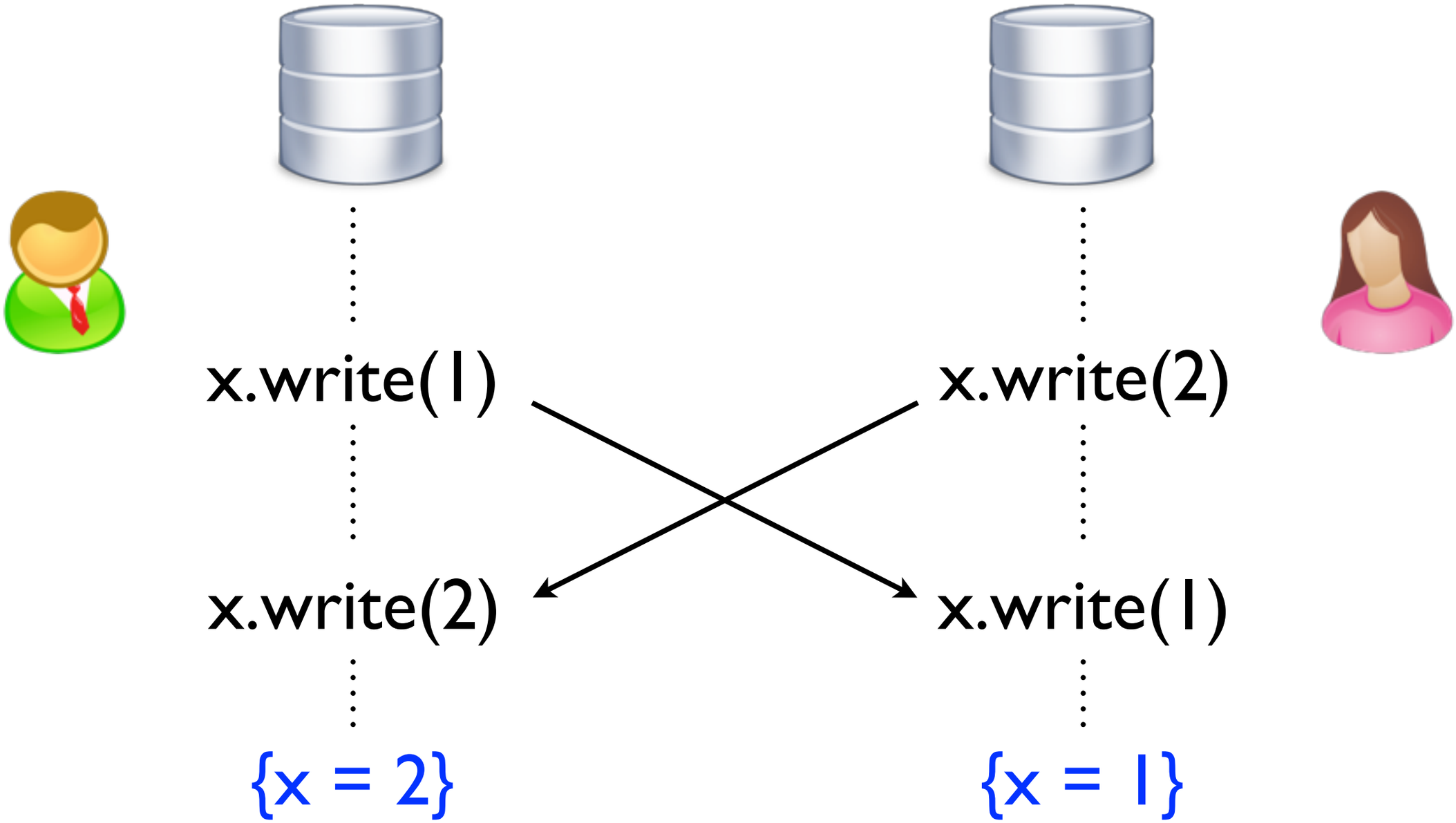
Update the local replica now,
propagate to others later

1. How do we resolve conflicts resulting from concurrent updates at different replicas?

x.write(1)

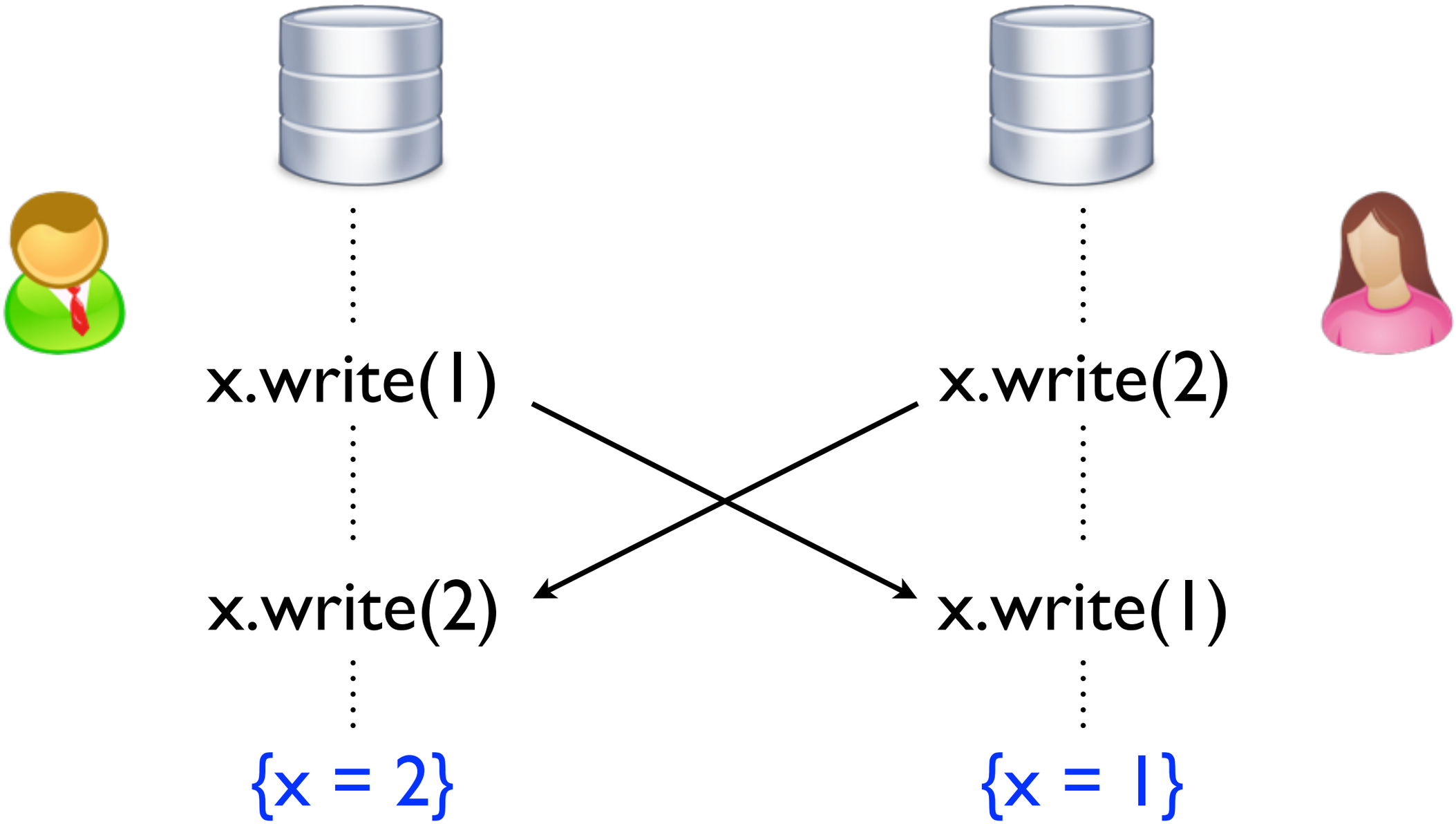
x.write(2)

1. How do we resolve conflicts resulting from concurrent updates at different replicas?



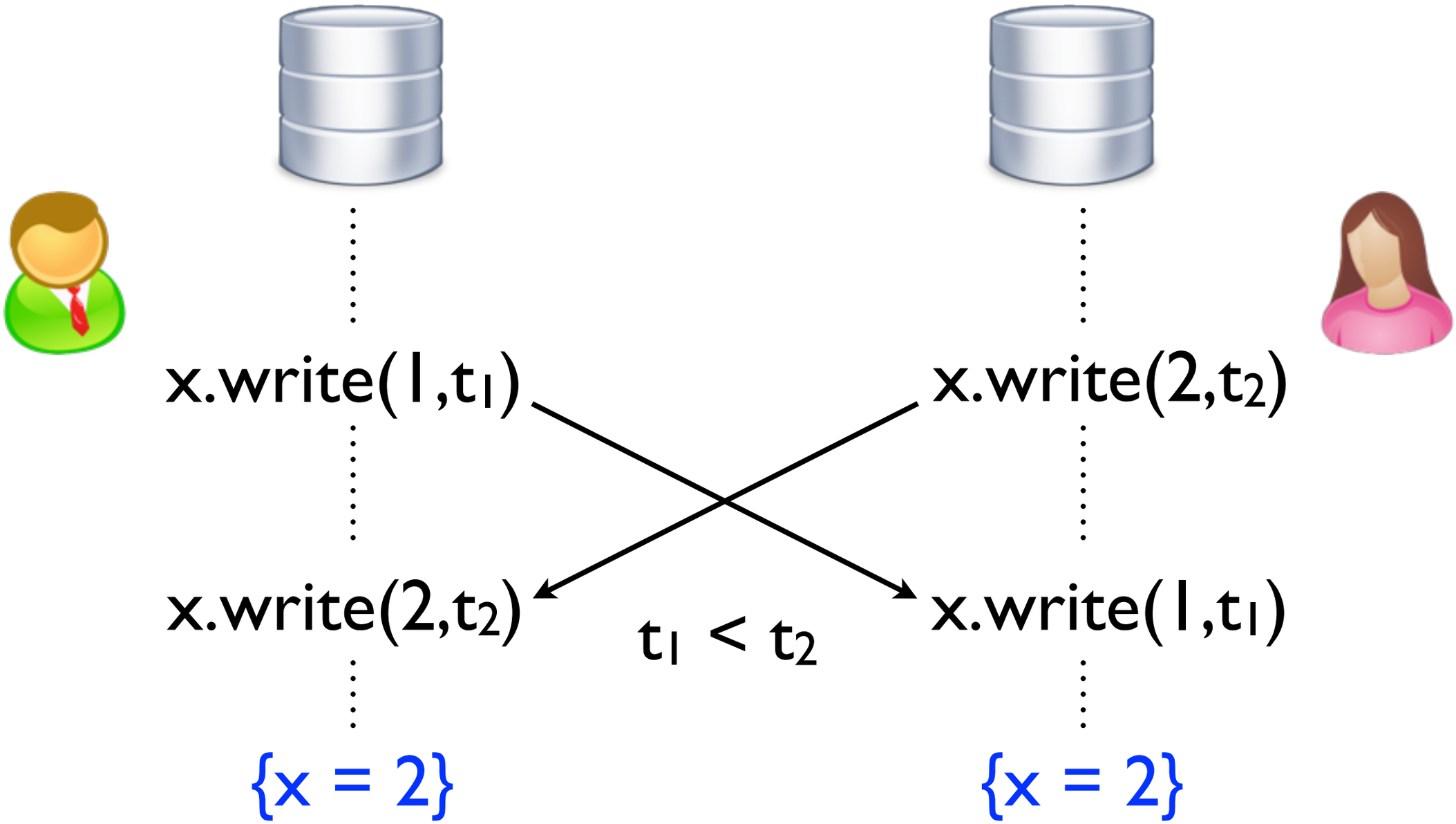
Replicas diverge ⇒ eventual consistency violated

1. How do we resolve conflicts resulting from concurrent updates at different replicas?



Updates have to commute ⇒ convergence guaranteed

1. How do we resolve conflicts resulting from concurrent updates at different replicas?



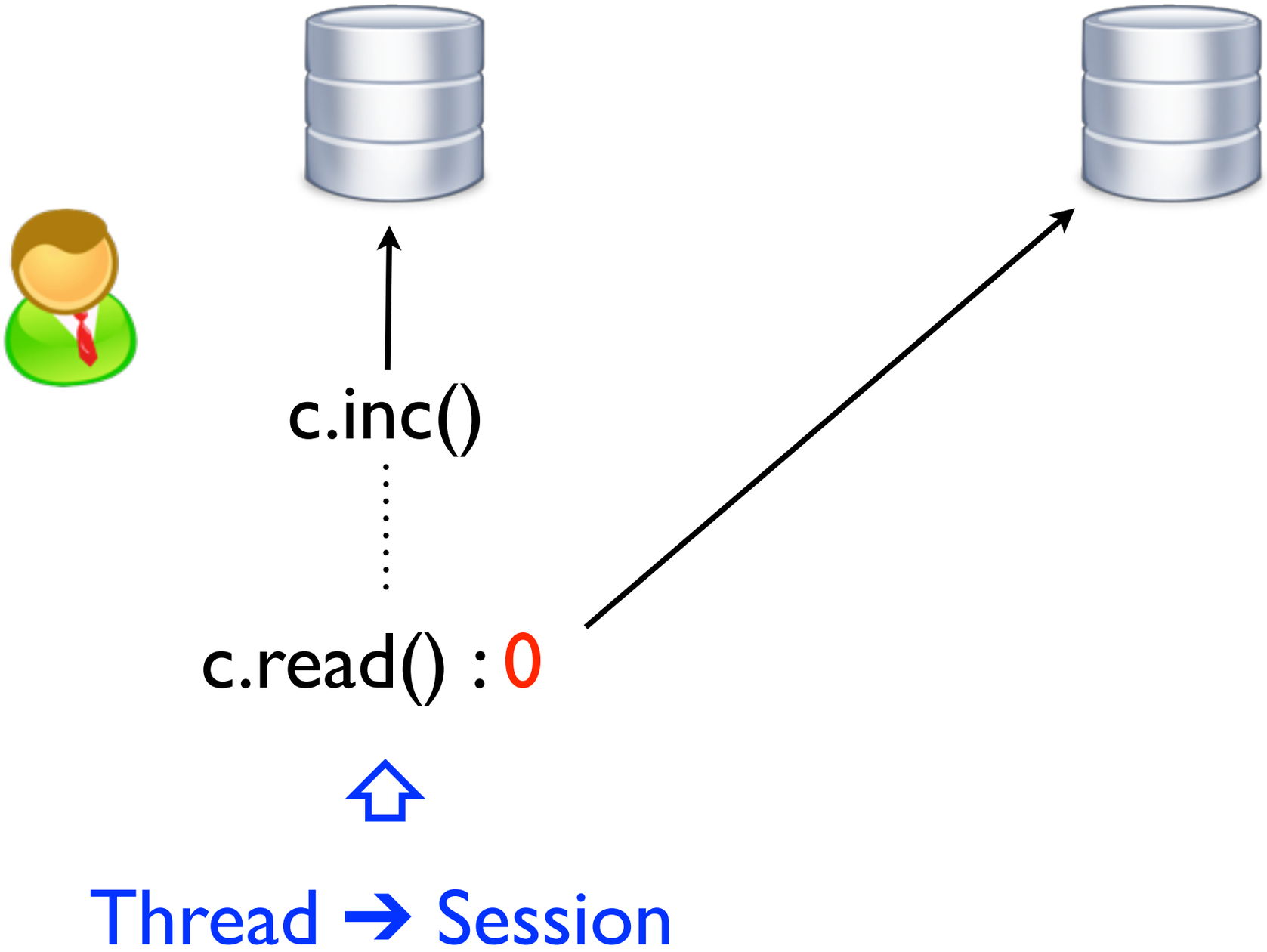
Timestamp updates if they don't commute

Not only physical time [Lamport 78]

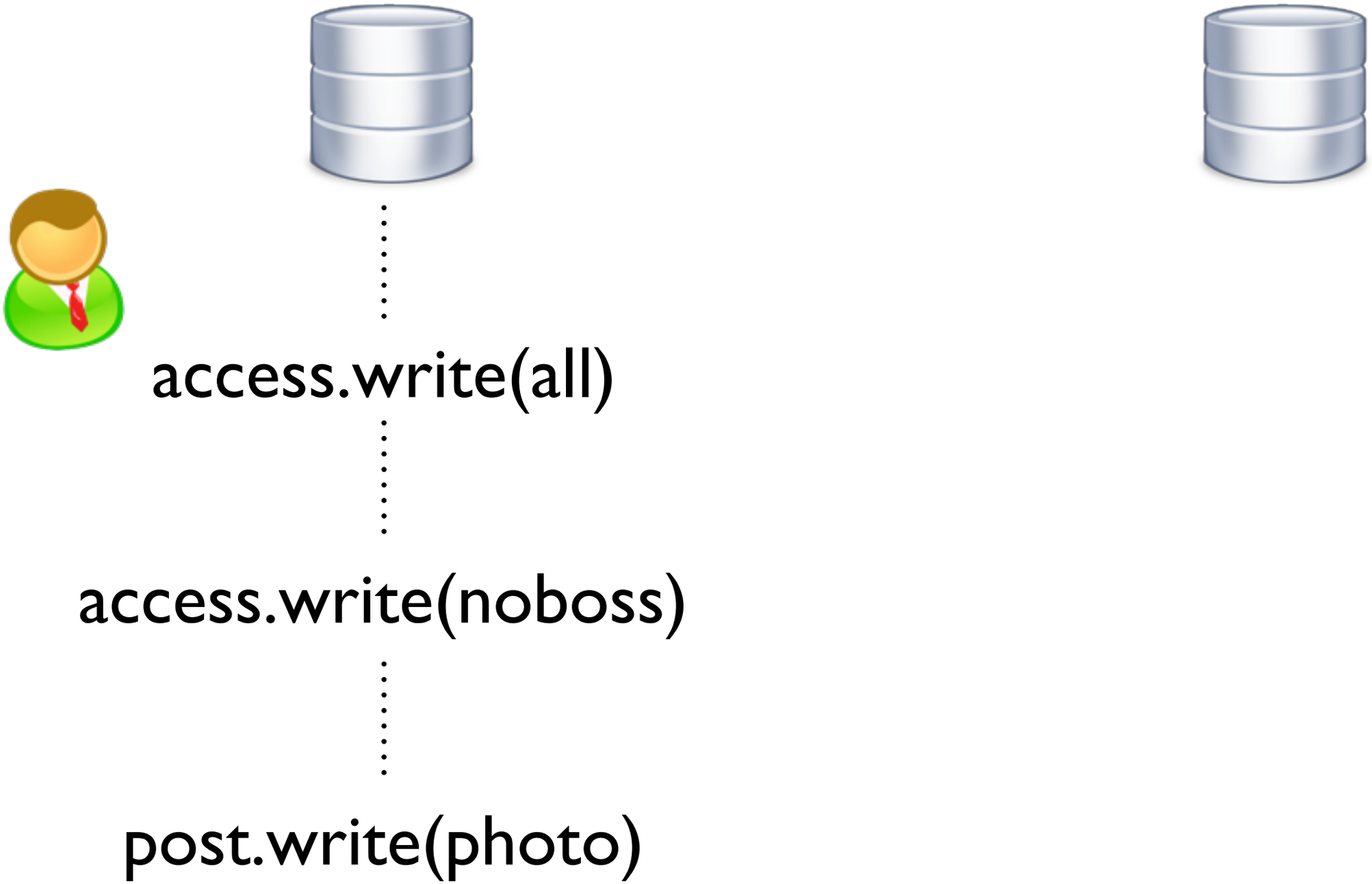
Replicated data types

- Lots more interesting ways to make operations commutative and specify a conflict resolution policy *(an example shortly)*
- More complicated data types than counters or read/write registers: sets, graphs, sequences, file systems [Shapiro+ 2011]
- Every object in the database can be assigned a replicated data type → conflict resolution policy

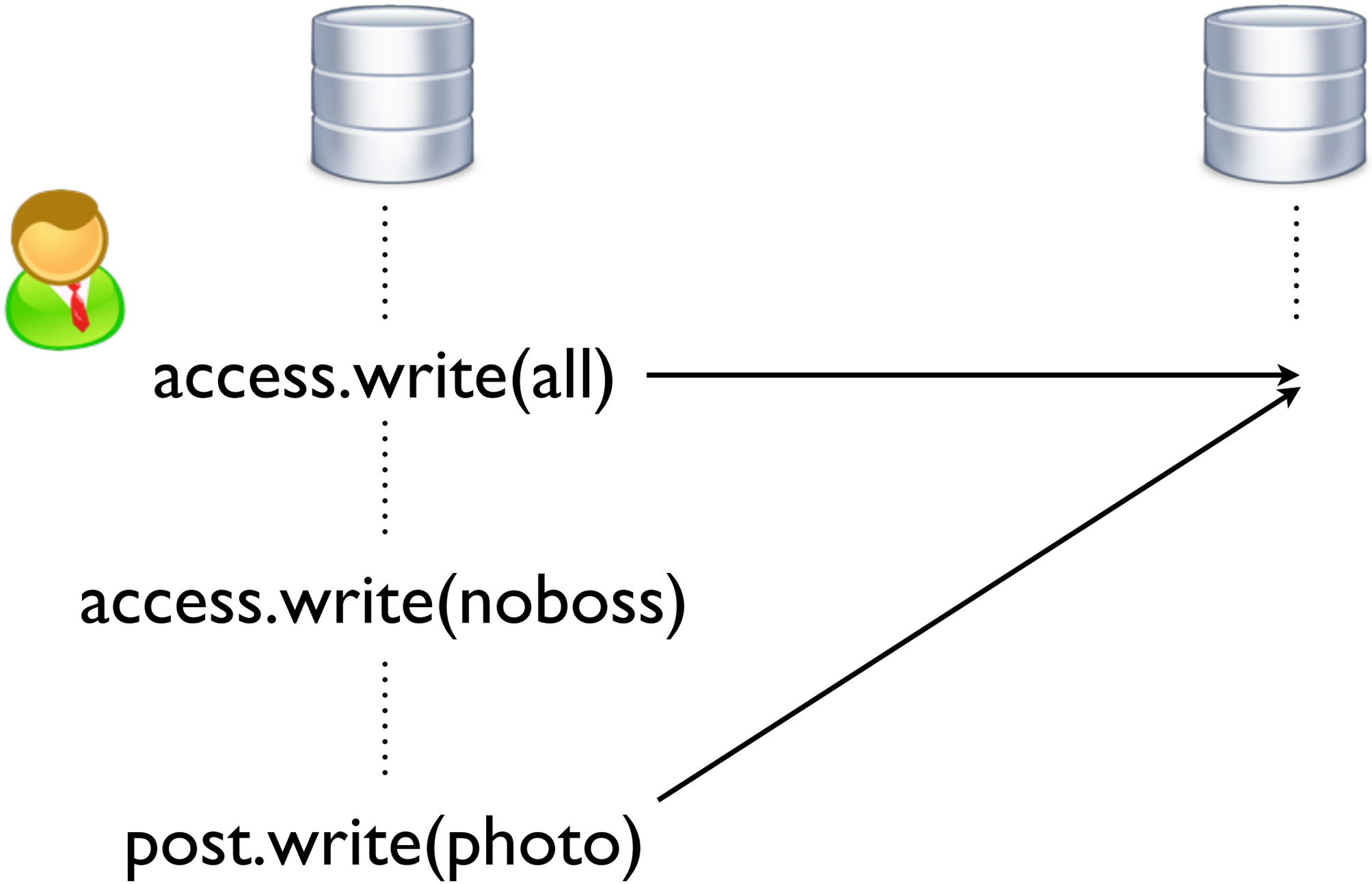
2. Does a user's knowledge change in line with the system evolution?



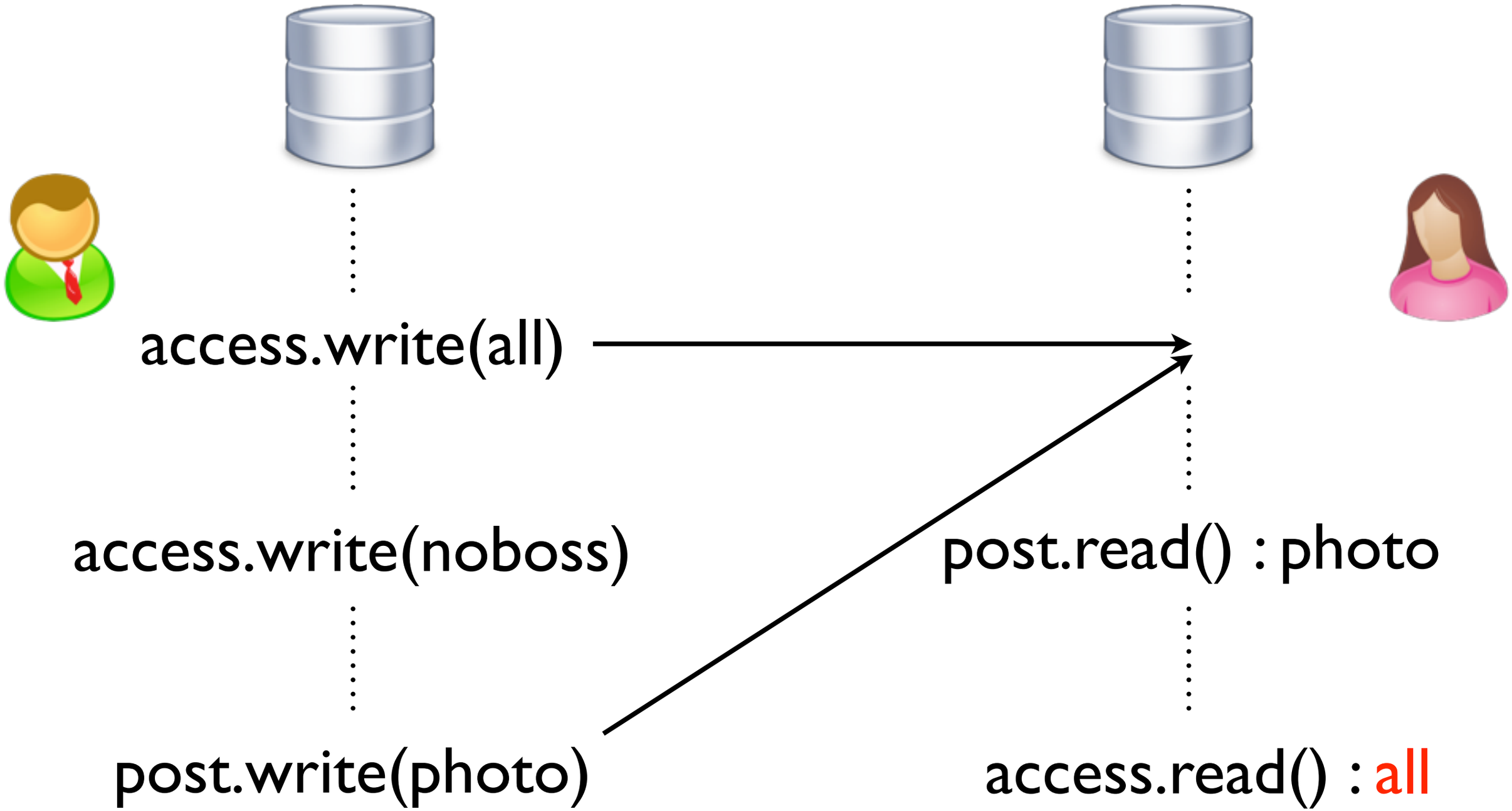
2. Does a user's knowledge change in line with the system evolution?



2. Does a user's knowledge change in line with the system evolution?



2. Does a user's knowledge change in line with the system evolution?



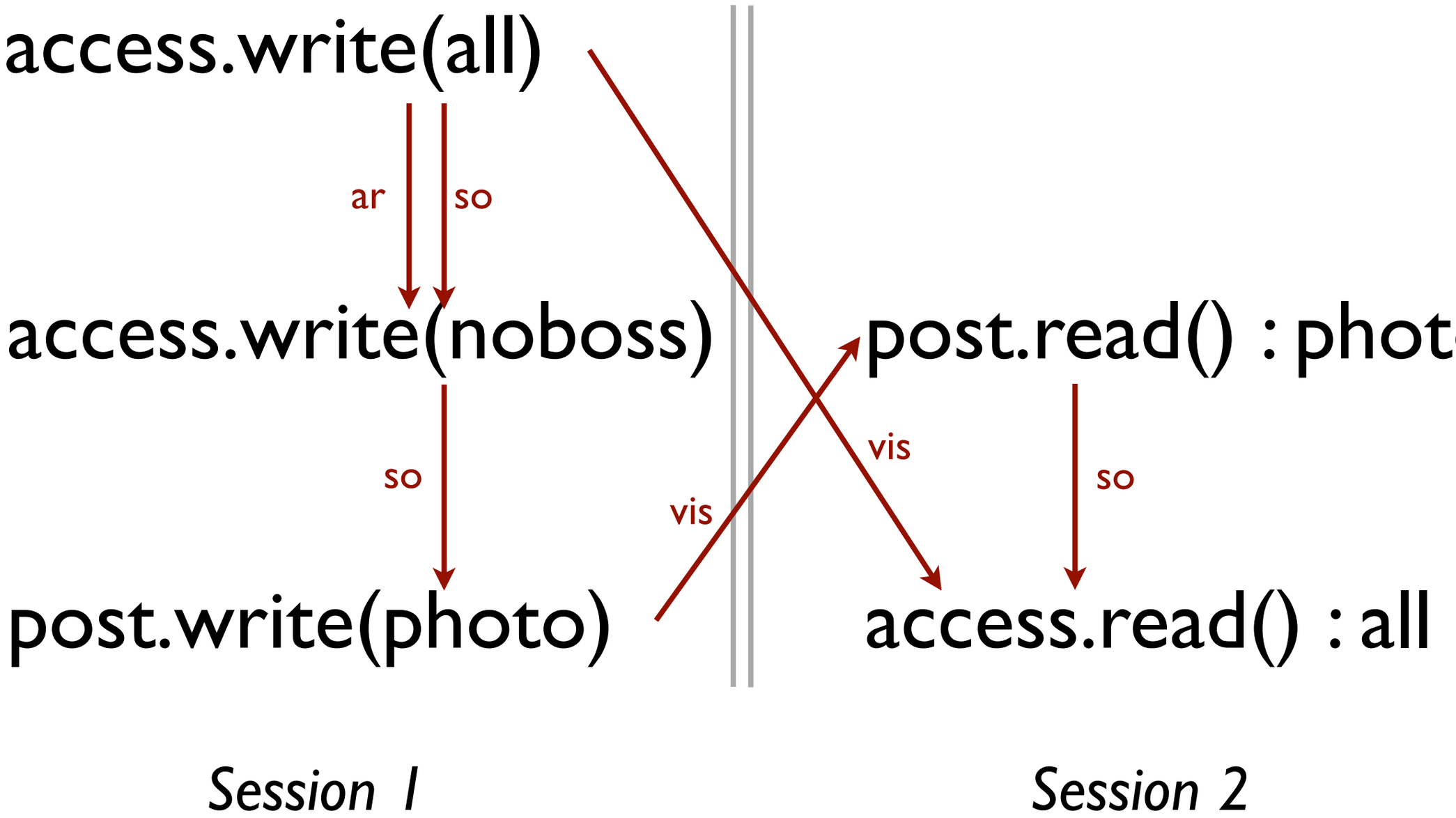
Formalising eventual consistency

1. How do we resolve conflicts resulting from concurrent updates at different replicas?

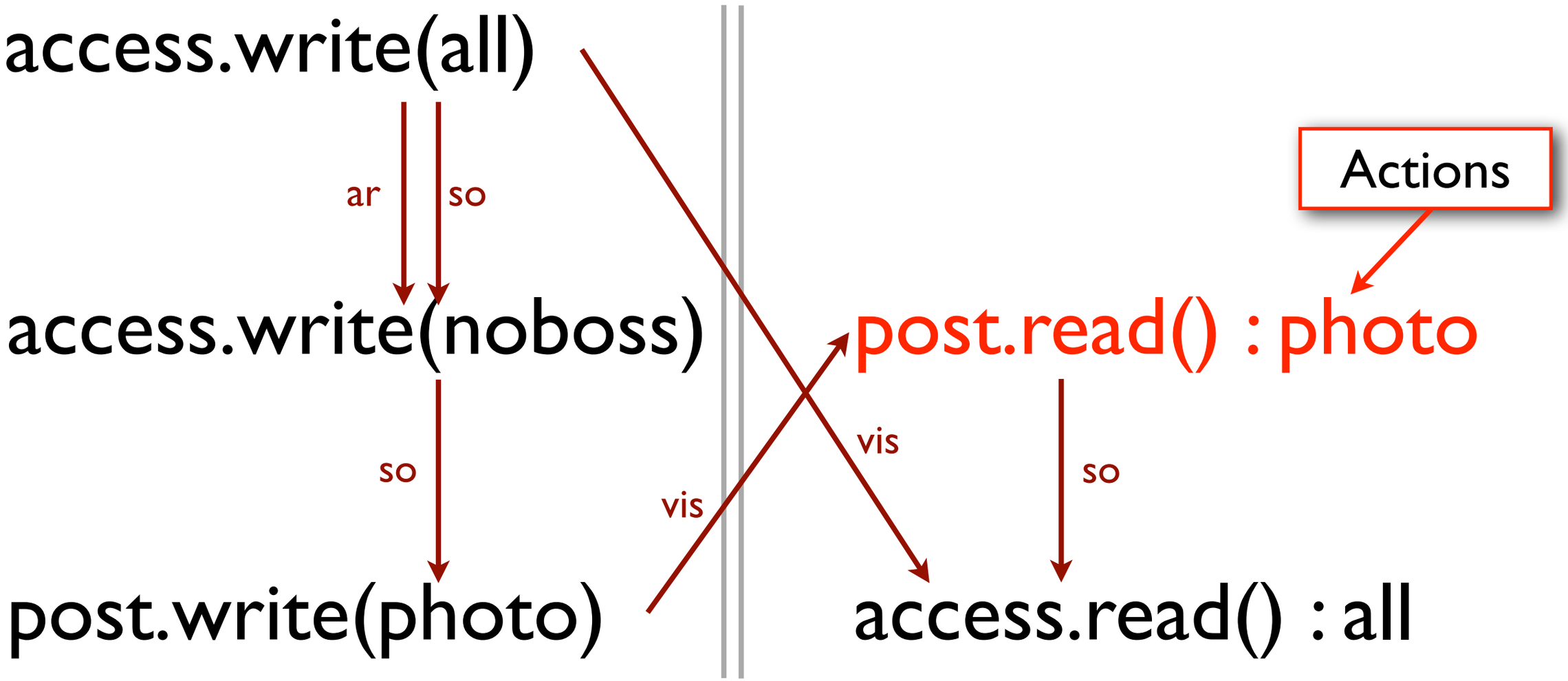
2. Does a user's knowledge change in line with the system evolution?

- First, define a notion of an execution
- Weak consistency ➜ straightforward interleaving semantics doesn't work
- Similar to axiomatic memory models

Execution: (A, so, vis, ar)

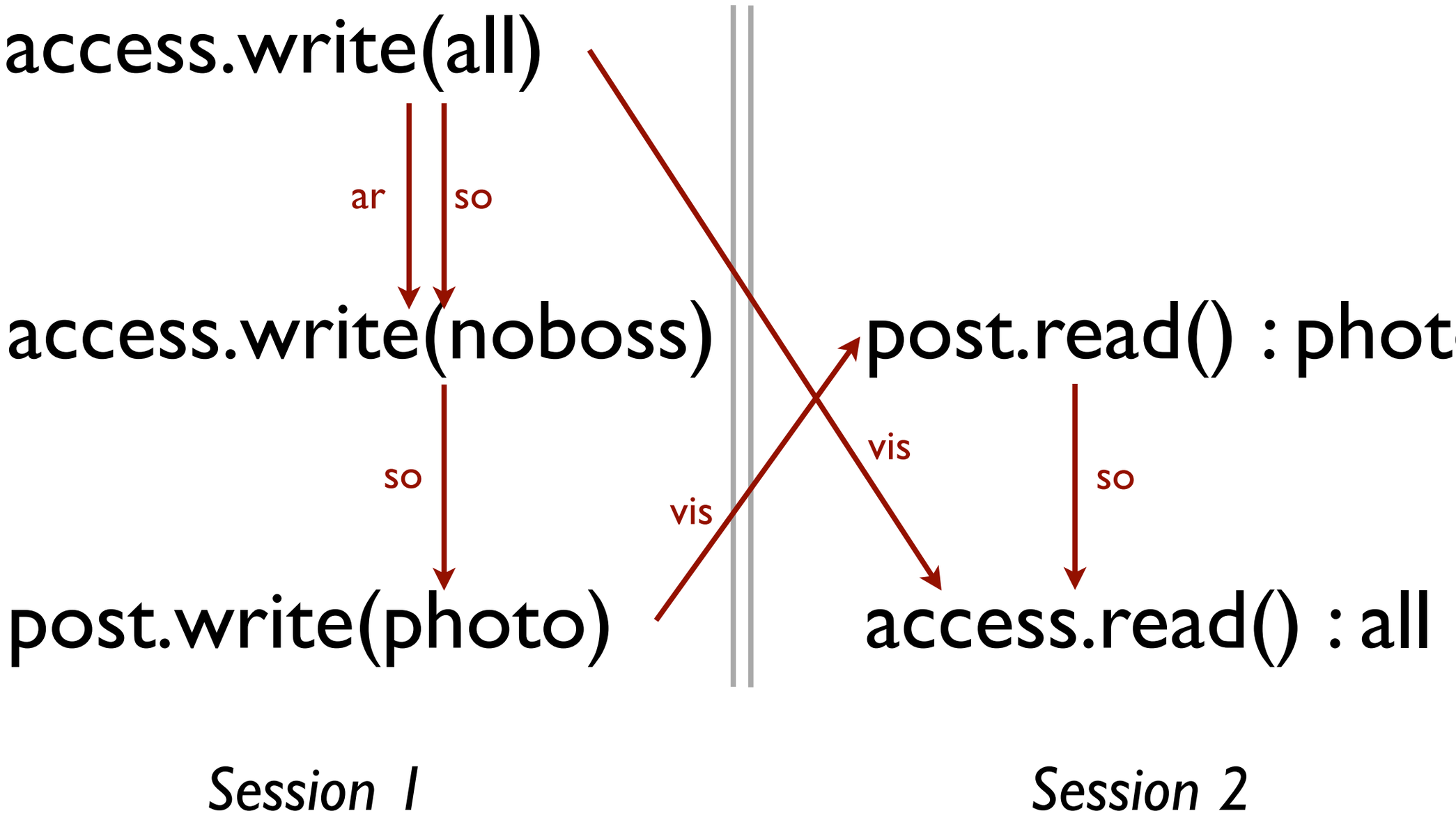


Execution: (A, so, vis, ar)

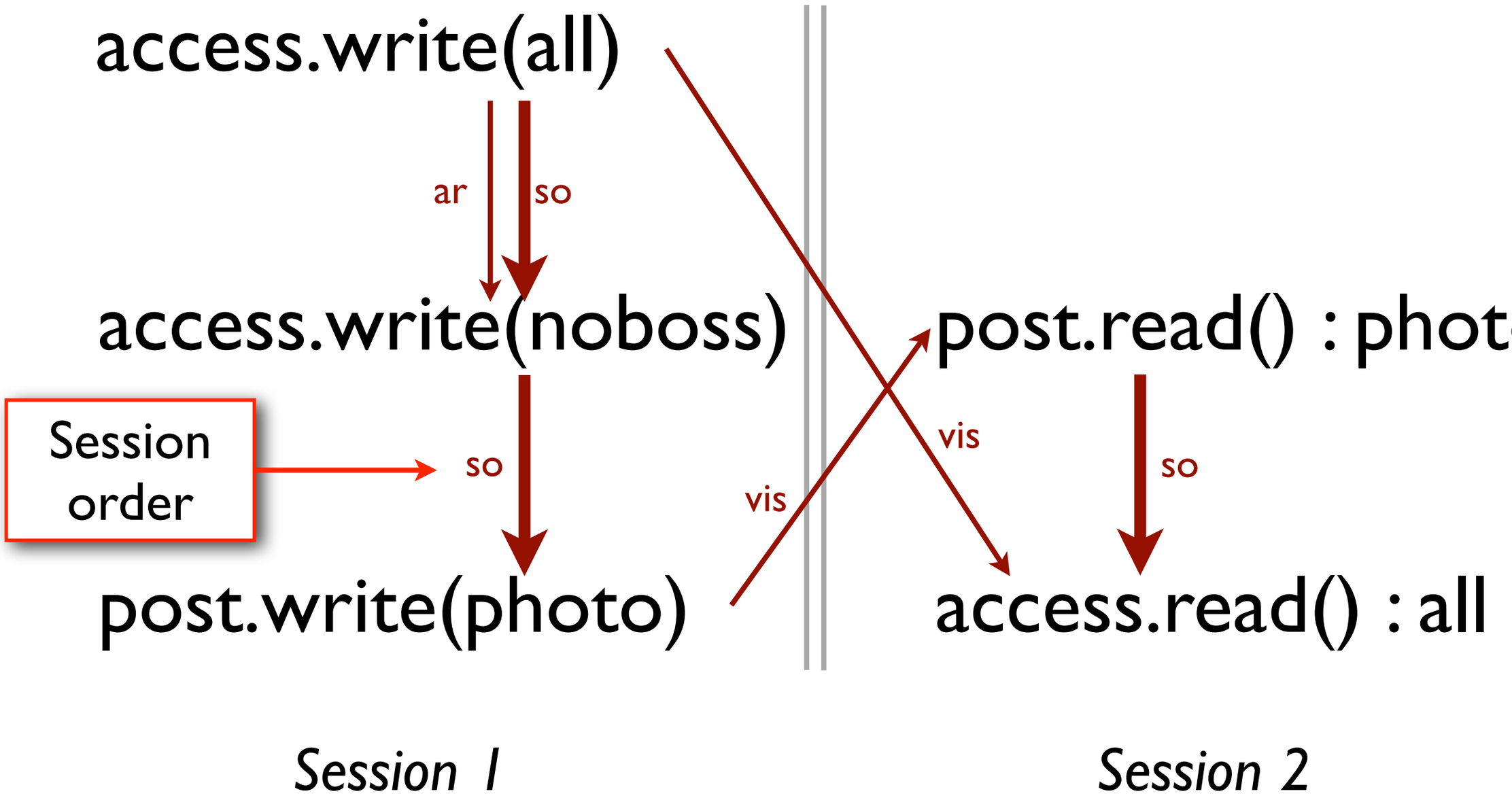


What happens on the interface client/database

Execution: (A, so, vis, ar)

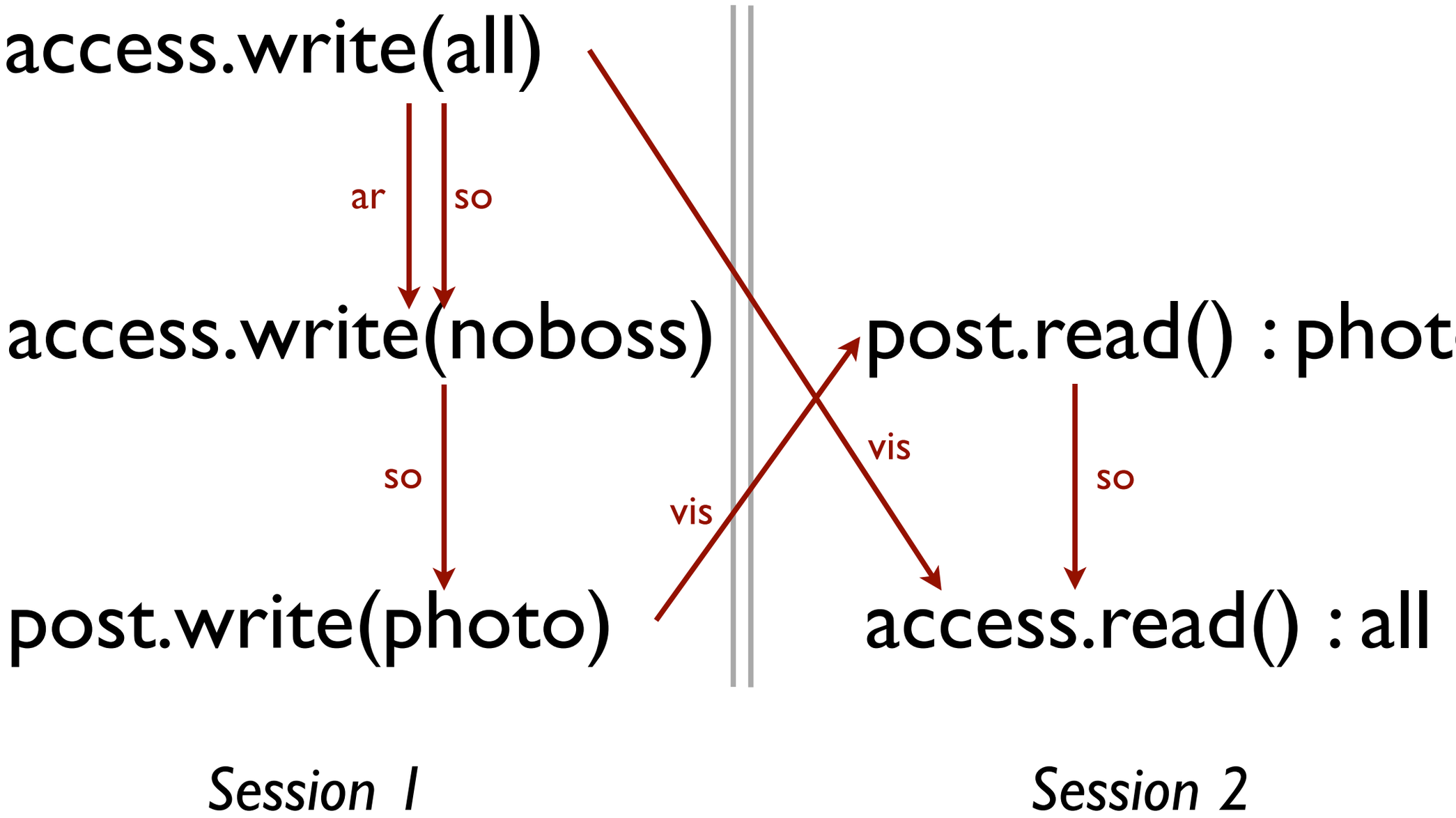


Execution: (A, so, vis, ar)

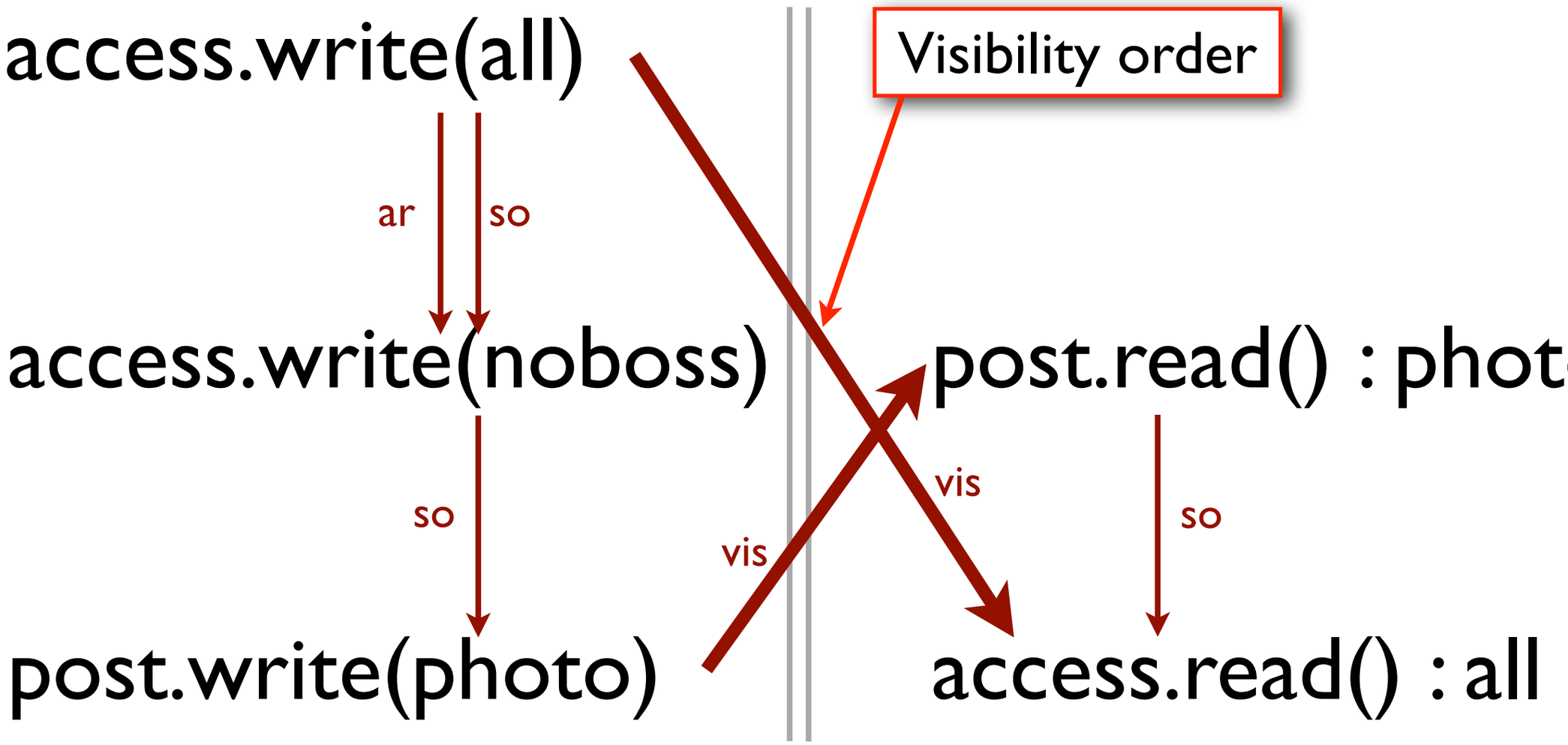


The order of submission to the database by a session

Execution: (A, so, vis, ar)

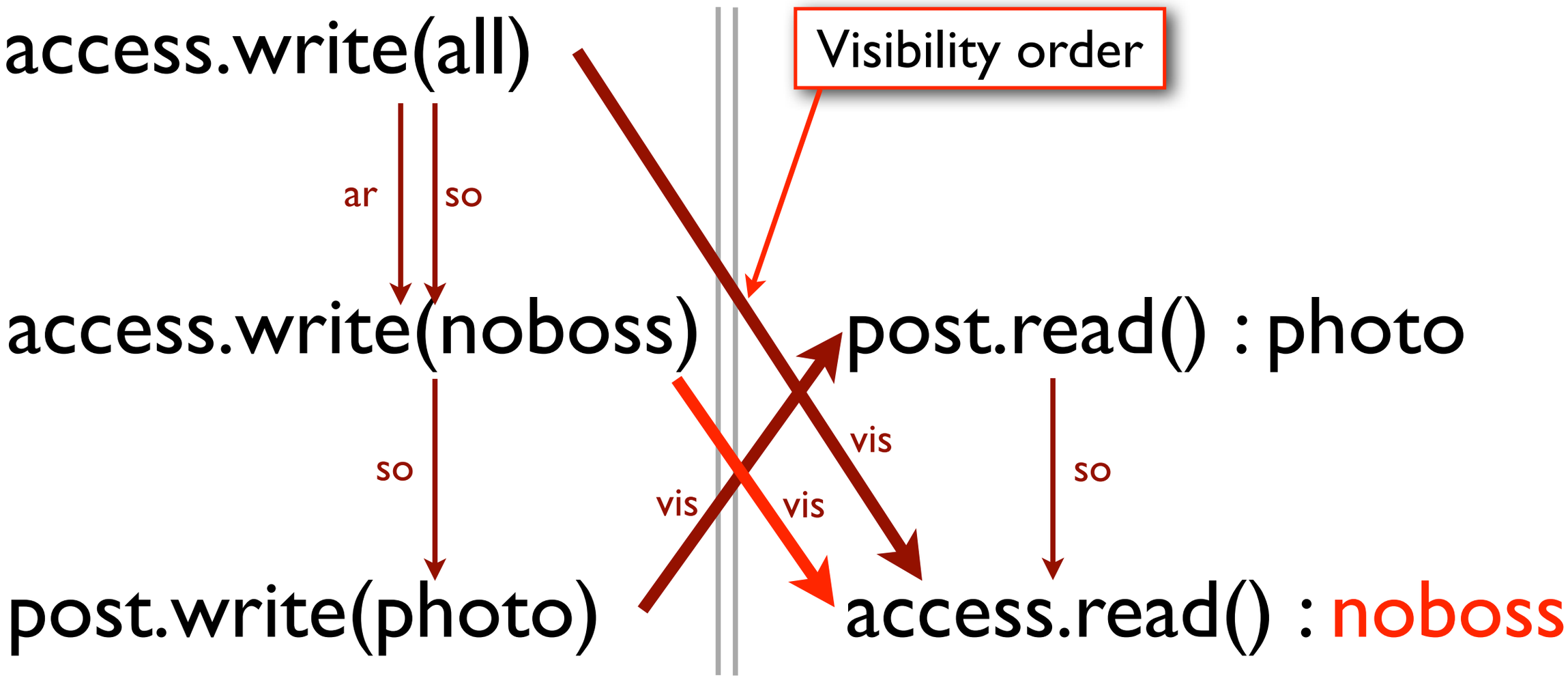


Execution: (A, so, vis, ar)



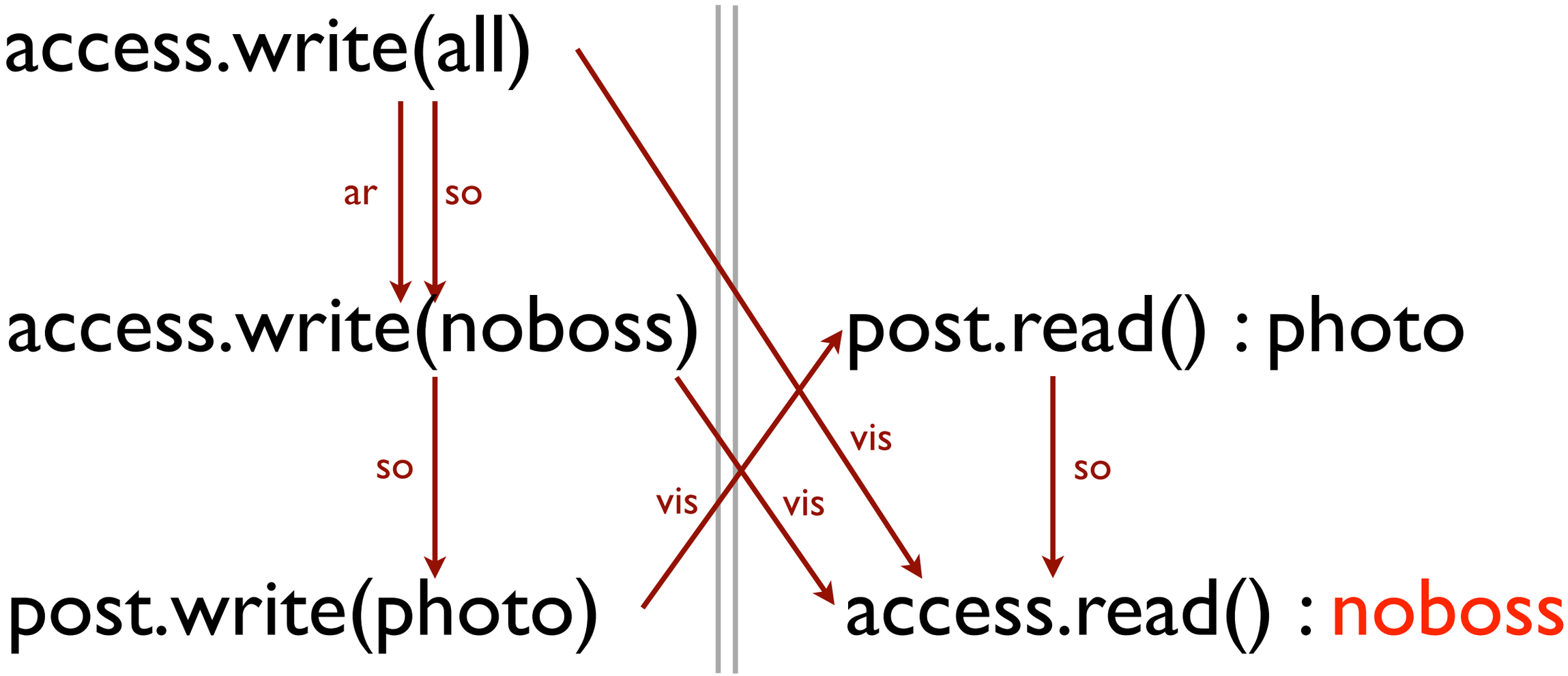
The update is visible to the session: has been delivered to the replica it is connected to

Execution: (A, so, vis, ar)

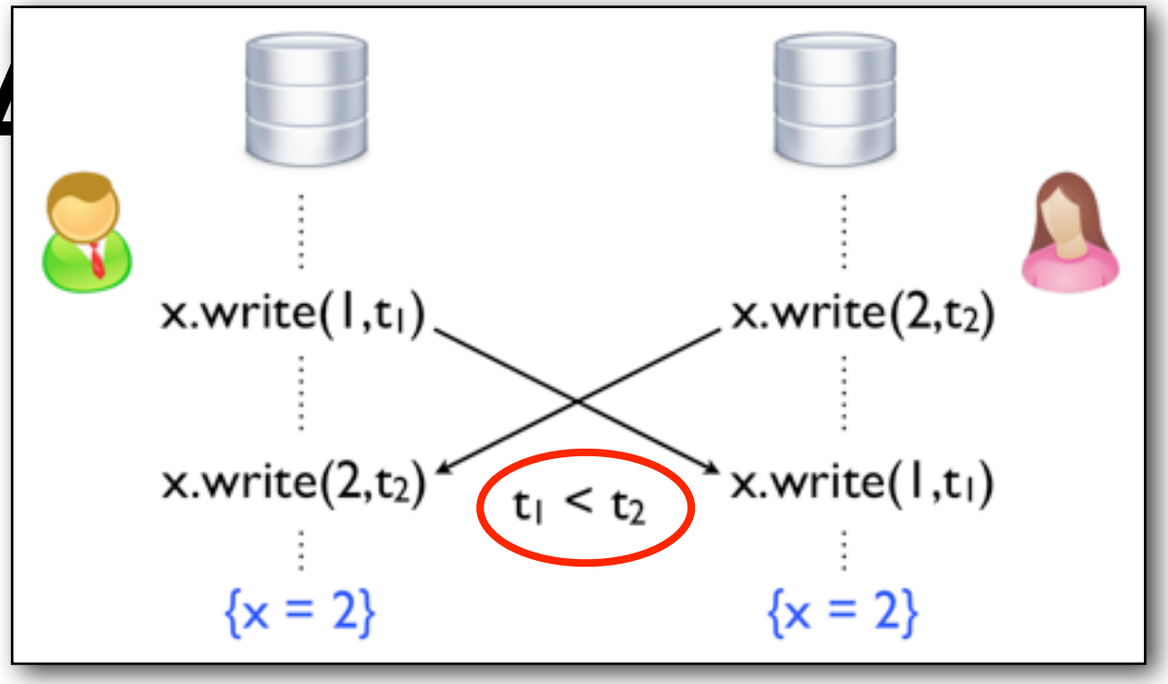


Affects the results of operations

Execution: (A, so, vis, ar)



Execution: (A



access.write(all)

Arbitration order

ar

so

access.write(noboss)

so

post.write(photo)

post.read() : photo

vis

vis

vis

so

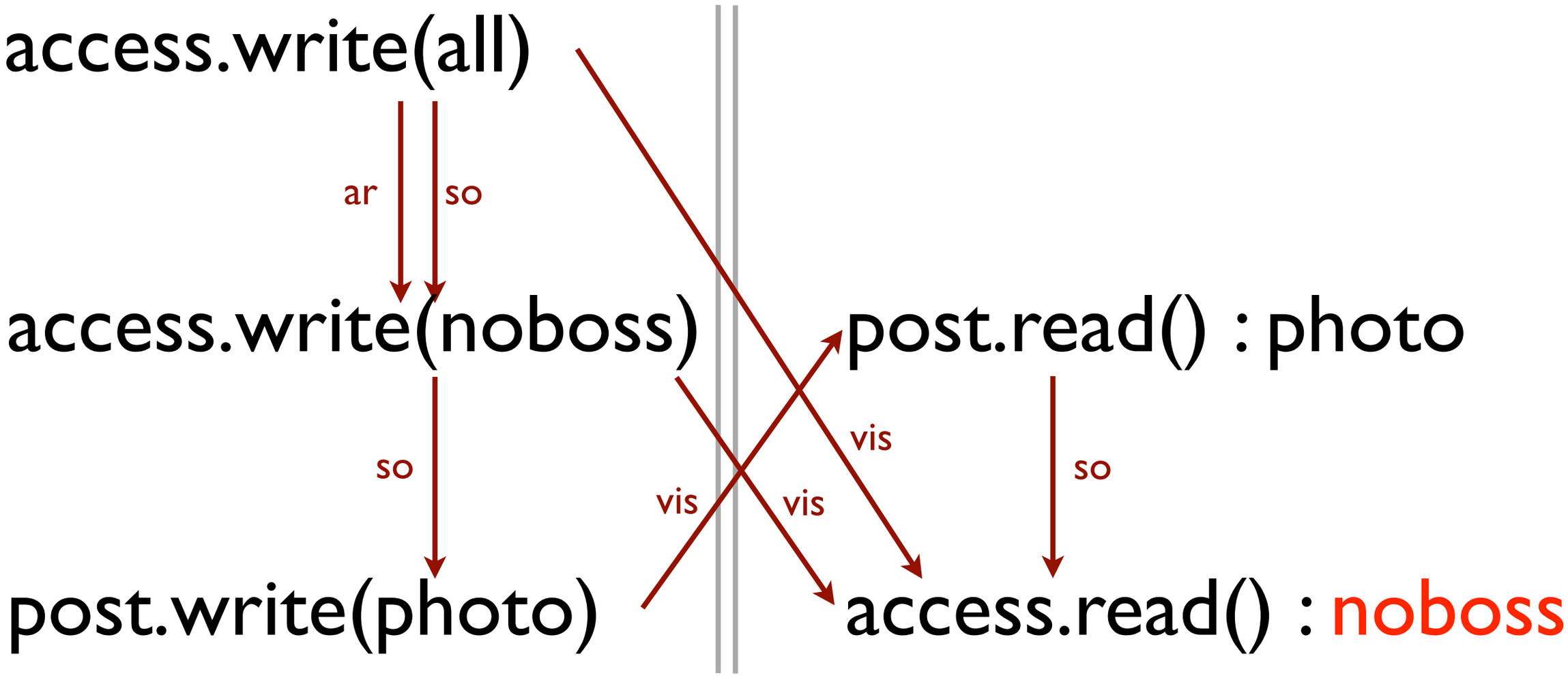
access.read() : noboss

Session 1

Session 2

The order of timestamps, used for conflict resolution

Execution: (A, so, vis, ar)



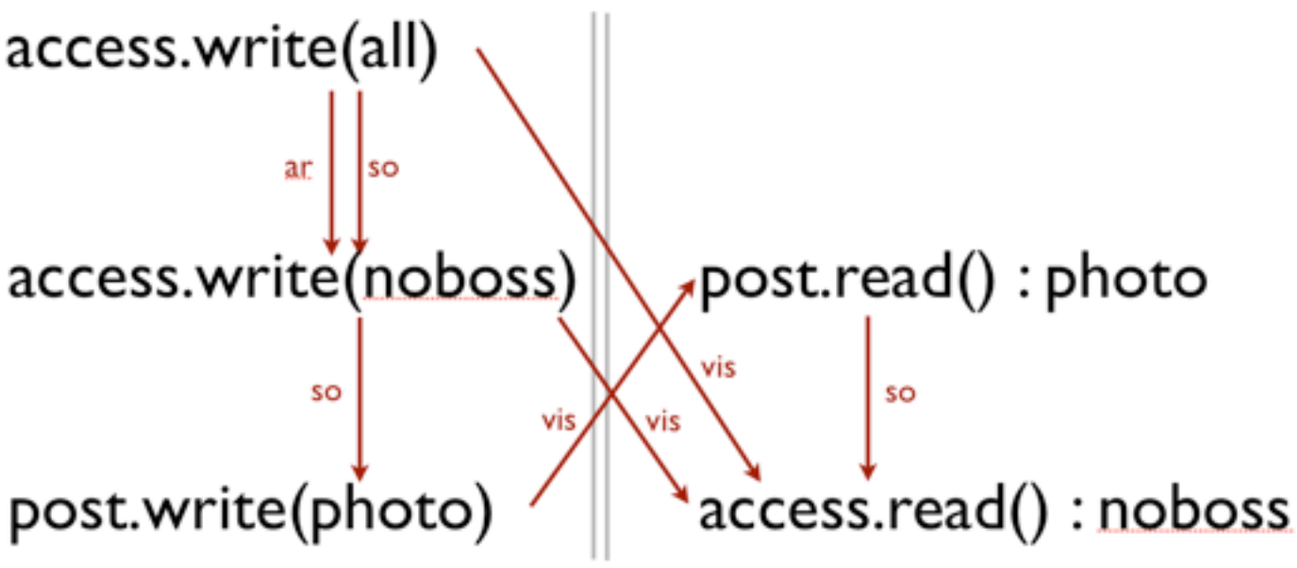
System specification = set of valid executions

1. How do we resolve conflicts resulting from concurrent updates at different replicas?

   Data type specification

2. Does a user's knowledge change in line with the system evolution?

   Consistency axioms



System specification = set

1. How do we resolve conflicts resul
   concurrent updates at different re

   Data type specification

2. Does a user's knowledge change ir
   the system evolution?

   Consistency axioms

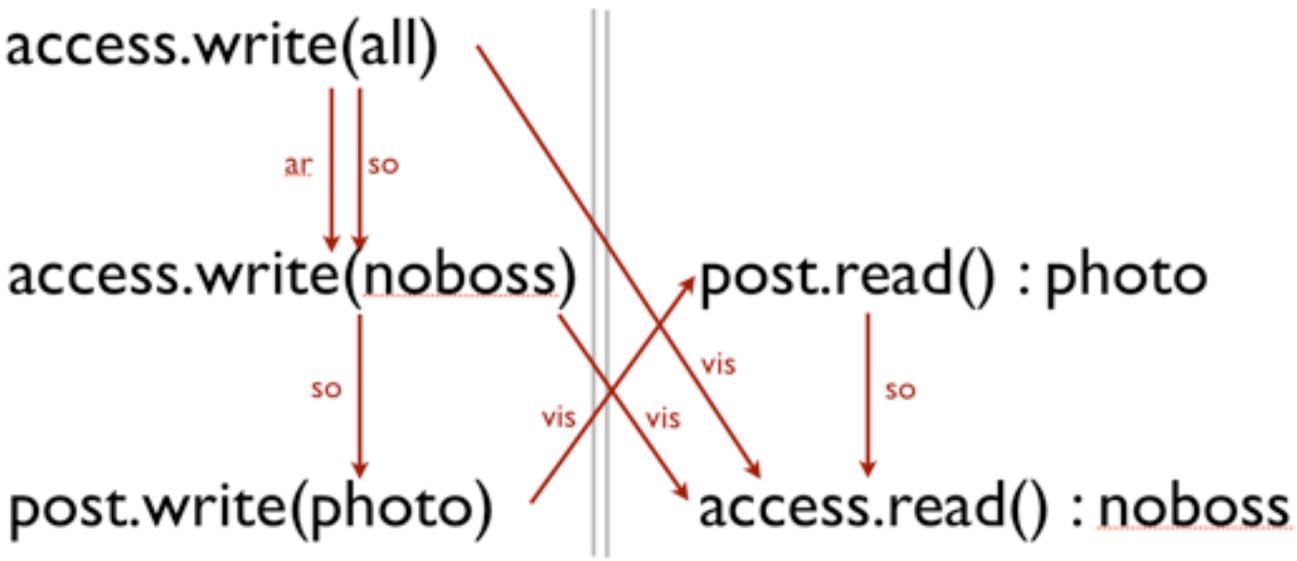


Figure 1. Axioms of eventual consistency

WELL-FORMEDNESS AXIOMS

SOWF: so is the union of transitive, irreflexive and total orders on actions by each session

VISWF: $\forall a, b.\, a \xrightarrow{\mathsf{vis}} b \implies \mathsf{obj}(a) = \mathsf{obj}(b)$

ARWF: $\forall a, b.\, a \xrightarrow{\mathsf{ar}} b \implies \mathsf{obj}(a) = \mathsf{obj}(b)$, ar is transitive and irreflexive, and $\mathsf{ar}|_{\mathsf{vis}^{-1}(a)}$ is a total order for all $a \in A$

AUXILIARY RELATIONS

Per-object session order: $\mathsf{soo} = (\mathsf{so} \cap \mathsf{sameobj})$

Per-object causality order: $\mathsf{hbo} = (\mathsf{soo} \cup \mathsf{vis})^+$

Causality order: $\mathsf{hb} = (\mathsf{so} \cup \mathsf{vis})^+$

BASIC EVENTUAL CONSISTENCY AXIOMS

RVAL: $\forall a \in A.\, \mathsf{rval}(a) = F_{\mathsf{type}(a)}(\mathsf{cone}(a))$

EVENTUAL: $\forall a \in A. \neg(\exists \text{ infinitely many } b \in A. \mathsf{sameobj}(a, b) \wedge \neg(a \xrightarrow{\mathsf{vis}} b))$

THINAIR: so ∪ vis is acyclic

SESSION GUARANTEES

RYW (Read Your Writes): $\mathsf{soo} \subseteq \mathsf{vis}$

MR (Monotonic Reads): $(\mathsf{vis}; \mathsf{soo}) \subseteq \mathsf{vis}$

WFRV (Writes Follow Reads in Visibility): $(\mathsf{vis}; \mathsf{soo}^*; \mathsf{vis}) \subseteq \mathsf{vis}$

WFRA (Writes Follow Reads in Arbitration): $(\mathsf{vis}; \mathsf{soo}^*) \subseteq \mathsf{ar}$

MWV (Monotonic Writes in Visibility): $(\mathsf{soo}; \mathsf{vis}) \subseteq \mathsf{vis}$

MWA (Monotonic Writes in Arbitration): $\mathsf{soo} \subseteq \mathsf{ar}$

CAUSALITY AXIOMS

POCV (Per-Object Causal Visibility): $\mathsf{hbo} \subseteq \mathsf{vis}$

POCA (Per-Object Causal Arbitration): $\mathsf{hbo} \subseteq \mathsf{ar}$

COCV (Cross-Object Causal Visibility): $(\mathsf{hb} \cap \mathsf{sameobj}) \subseteq \mathsf{vis}$

COCA (Cross-Object Causal Arbitration): hb ∪ ar is acyclic

System specification = set

1. How do we resolve conflicts resul
   concurrent updates at different re

   Data type specification

2. Does a user's knowledge change ir
   the system evolution?

   Consistency axioms

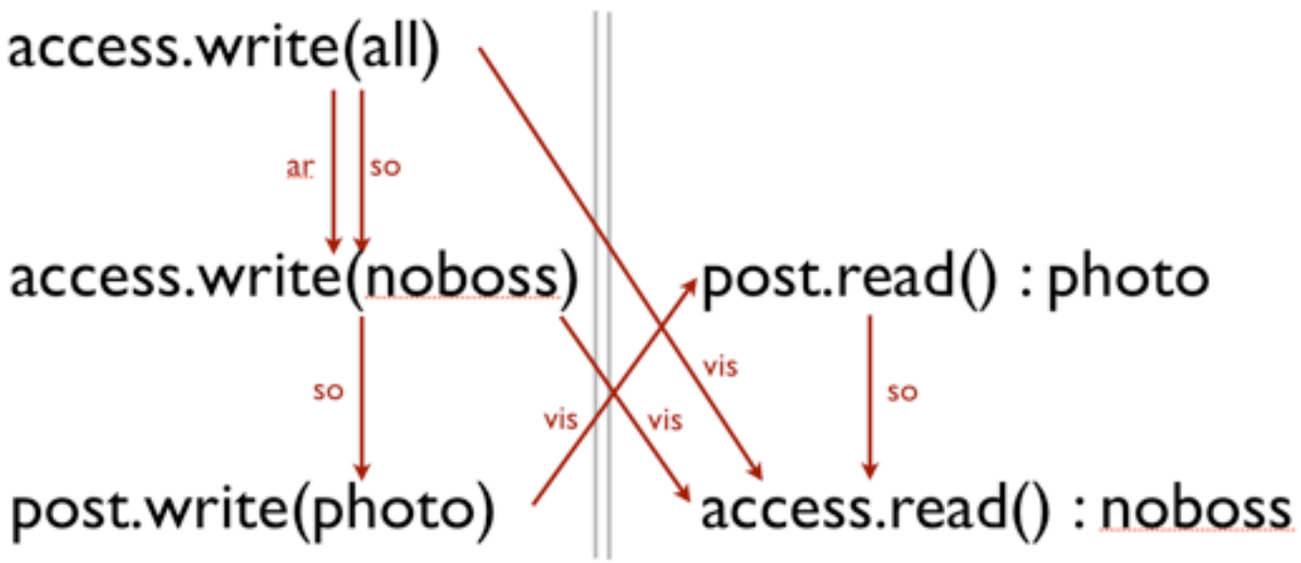


Figure 1. Axioms of eventual consistency

WELL-FORMEDNESS AXIOMS

SOWF: so is the union of transitive, irreflexive and total orders on actions by each session

VISWF: $\forall a, b.\, a \xrightarrow{\mathsf{vis}} b \implies \mathsf{obj}(a) = \mathsf{obj}(b)$

ARWF: $\forall a, b.\, a \xrightarrow{\mathsf{ar}} b \implies \mathsf{obj}(a) = \mathsf{obj}(b)$, ar is transitive and irreflexive, and $\mathsf{ar}|_{\mathsf{vis}^{-1}(a)}$ is a total order for all $a \in A$

AUXILIARY RELATIONS

Per-object session order: $\mathsf{soo} = (\mathsf{so} \cap \mathsf{sameobj})$

Per-object causality order: $\mathsf{hbo} = (\mathsf{soo} \cup \mathsf{vis})^+$

Causality order: $\mathsf{hb} = (\mathsf{so} \cup \mathsf{vis})^+$

BASIC EVENTUAL CONSISTENCY AXIOMS

RVAL: $\forall a \in A.\, \mathsf{rval}(a) = F_{\mathsf{type}(a)}(\mathsf{cone}(a))$

EVENTUAL: $\forall a \in A. \neg(\exists \text{ infinitely many } b \in A.\, \mathsf{sameobj}(a, b) \wedge \neg(a \xrightarrow{\mathsf{vis}} b))$

THINAIR: so ∪ vis is acyclic

SESSION GUARANTEES

RYW (Read Your Writes): $\mathsf{soo} \subseteq \mathsf{vis}$

MR (Monotonic Reads): $(\mathsf{vis}; \mathsf{soo}) \subseteq \mathsf{vis}$

WFRV (Writes Follow Reads in Visibility): $(\mathsf{vis}; \mathsf{soo}^*; \mathsf{vis}) \subseteq \mathsf{vis}$

WFRA (Writes Follow Reads in Arbitration): $(\mathsf{vis}; \mathsf{soo}^*) \subseteq \mathsf{ar}$

MWV (Monotonic Writes in Visibility): $(\mathsf{soo}; \mathsf{vis}) \subseteq \mathsf{vis}$

MWA (Monotonic Writes in Arbitration): $\mathsf{soo} \subseteq \mathsf{ar}$

CAUSALITY AXIOMS

POCV (Per-Object Causal Visibility): $\mathsf{hbo} \subseteq \mathsf{vis}$

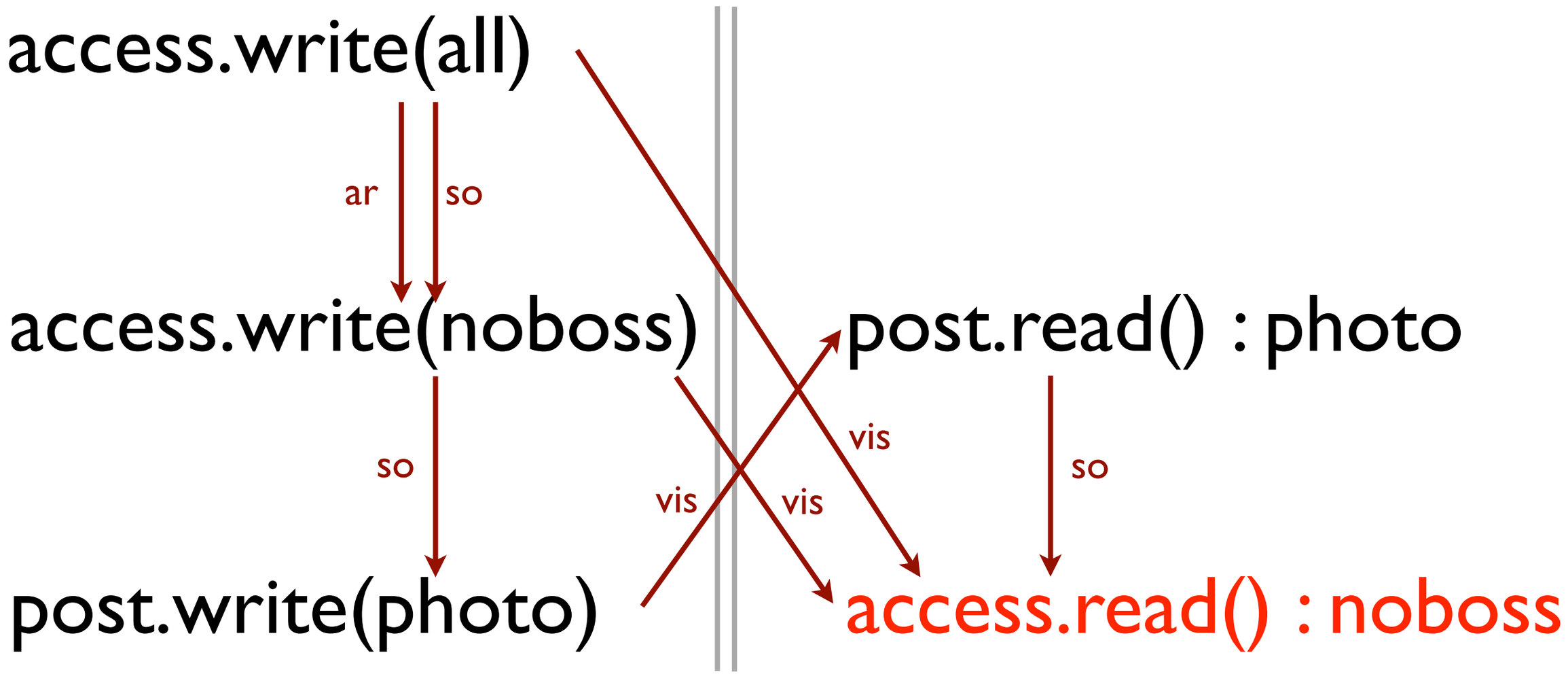
POCA (Per-Object Causal Arbitration): $\mathsf{hbo} \subseteq \mathsf{ar}$

COCV (Cross-Object Causal Visibility): $(\mathsf{hb} \cap \mathsf{sameobj}) \subseteq \mathsf{vis}$

COCA (Cross-Object Causal Arbitration): hb ∪ ar is acyclic

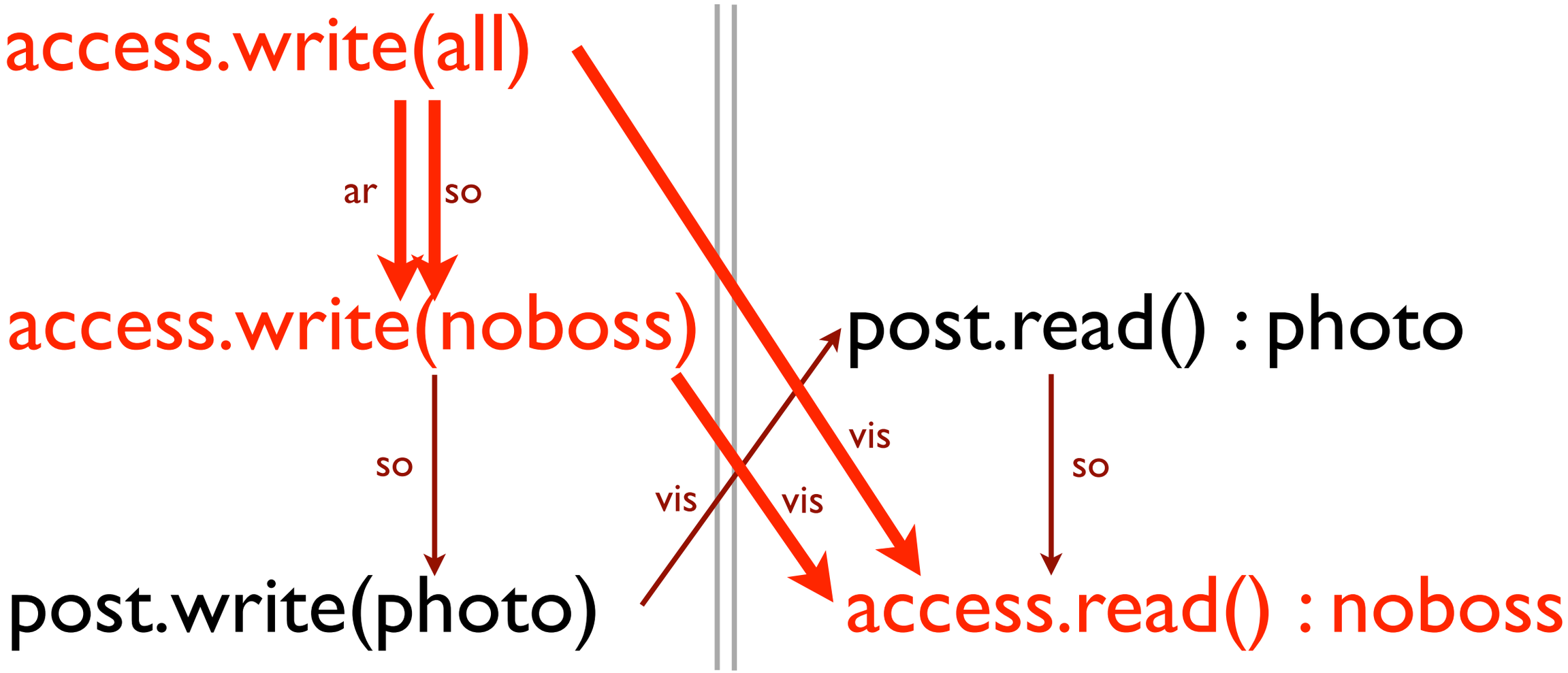
Data type specification

- How do I compute the return value of an action a?
- Only actions visible to a are important: have been delivered to the replica performing a



Data type specification

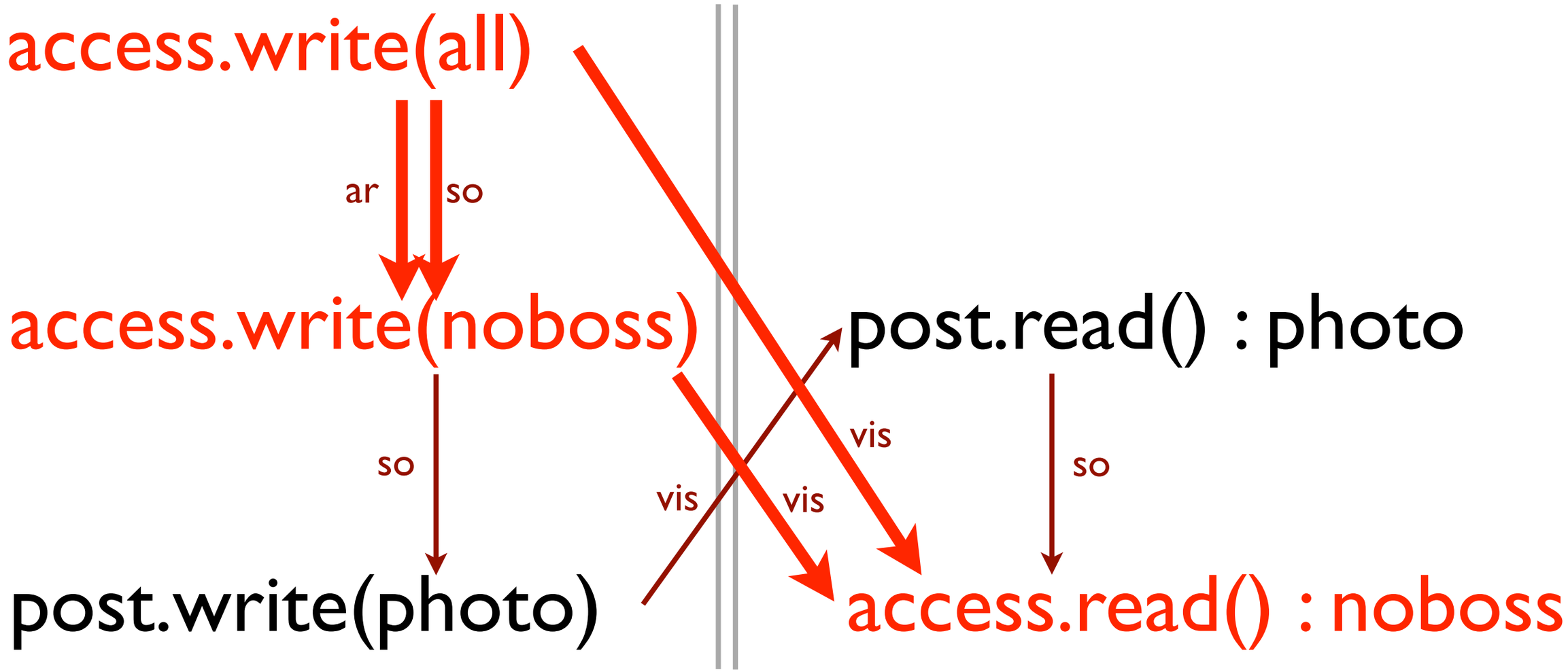
- How do I compute the return value of an action a?
- Only actions visible to a are important: have been delivered to the replica performing a
- Context of a - projection of the execution onto such actions



Data type specification

F: context of a → return value of a

$$\textsc{Query}.\ \forall a \in A.\ \mathsf{retval}(a) = F_{\mathsf{type}(a)}(\mathsf{ctxt}(a))$$



Data type specification

F: context of a → return value of a

$$\textsc{Query}.\ \forall a \in A.\ \mathsf{retval}(a) = F_{\mathsf{type}(a)}(\mathsf{ctxt}(a))$$

access.write(all)

ar so

access.write(noboss)

so

post.write(photo)

vis vis vis

F for read-write registers: sort all actions according to ar and return the last value written

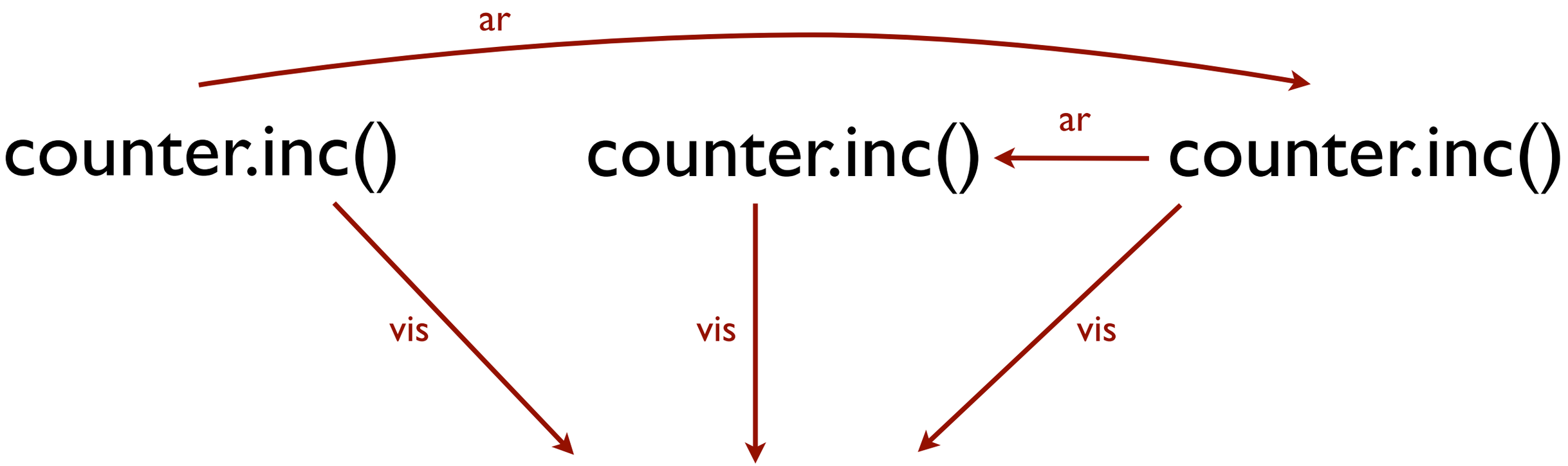
post.read() : photo

so

access.read() : noboss

Counter data type

F: context of a → return value of a



F: apply standard counter operations in any order

Commutative ➜ ar is not used

A set data type

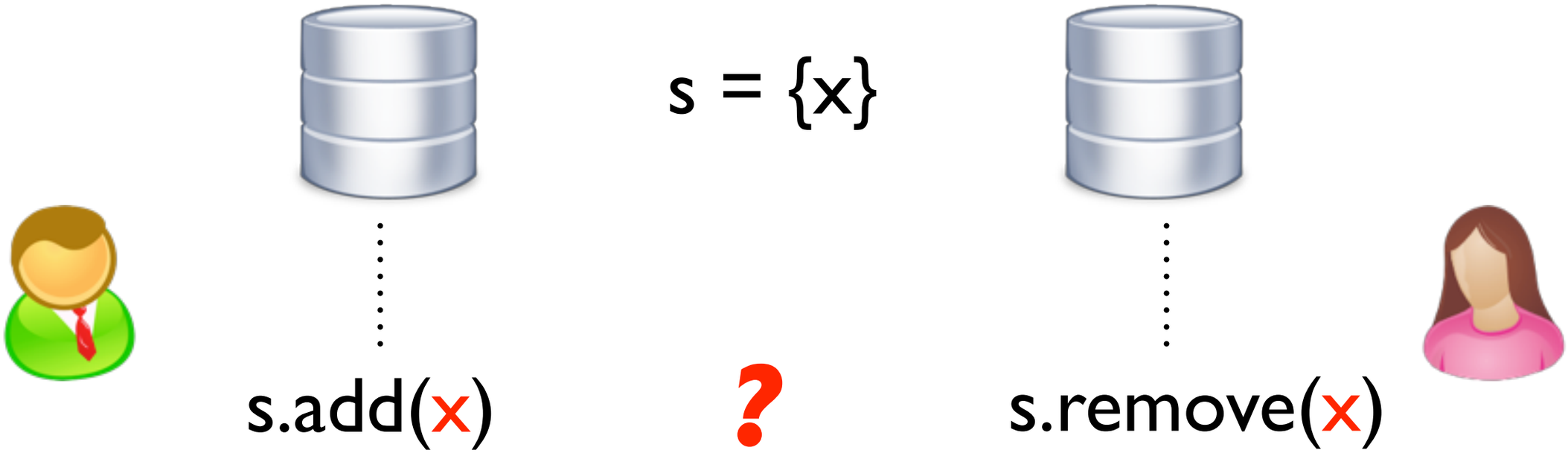
s = {x}

s.add(x)

?

s.remove(x)

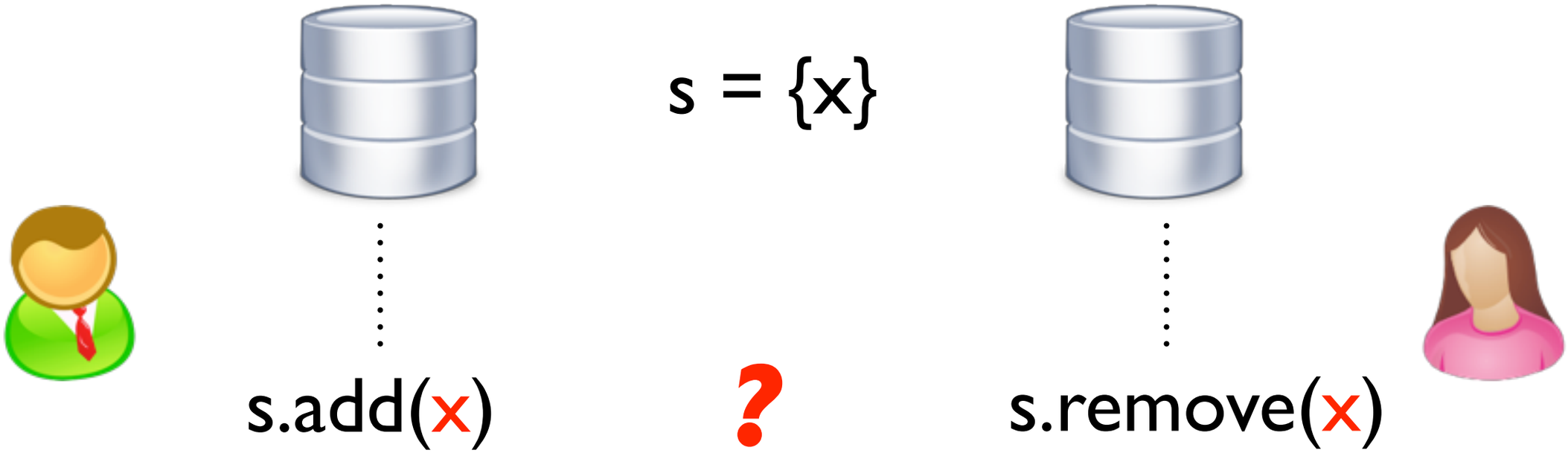
A set data type



Remove wins: s.add(x); s.remove(x) ➜ s = {}

Add wins: s.remove(x); s.add(x) ➜ s = {x}

A set data type



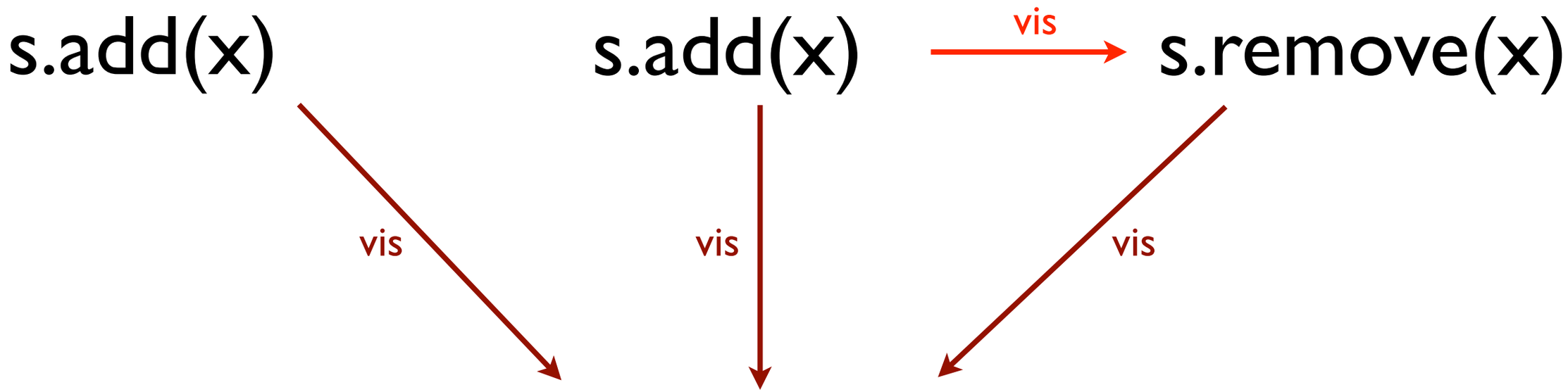
Remove wins: s.add(x); s.remove(x) ➜ s = {}

Add wins: s.remove(x); s.add(x) ➜ s = {x}

- Time-stamping is not always the best solution
- Application semantics might prefer a given outcome: add wins for a shopping cart

Add-wins set

F: context of a → return value of a

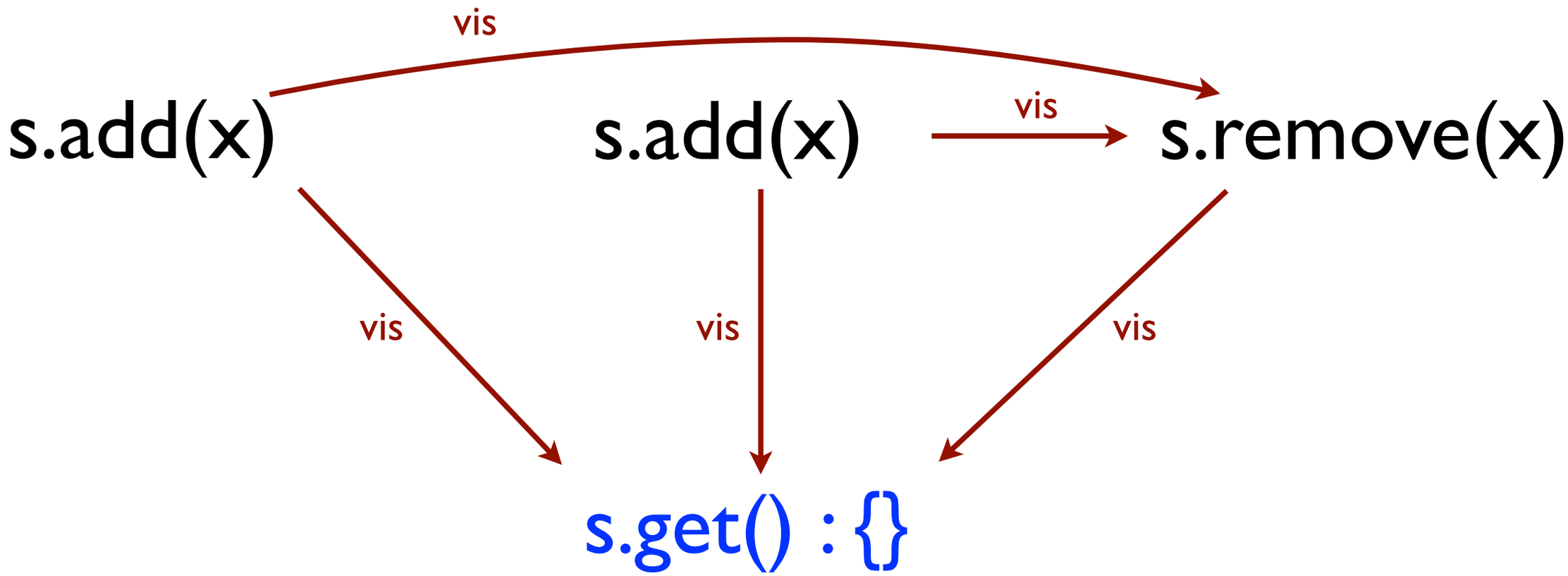


F: remove cancels out only vis-preceding adds

If you saw it, it's not a conflict

Add-wins set

F: context of a → return value of a



F: remove cancels out only vis-preceding adds

If you saw it, it's not a conflict

System specification = set

1. How do we resolve conflicts resul
   concurrent updates at different re

   Data type specification

2. Does a user's knowledge change i
   the system evolution?

   Consistency axioms

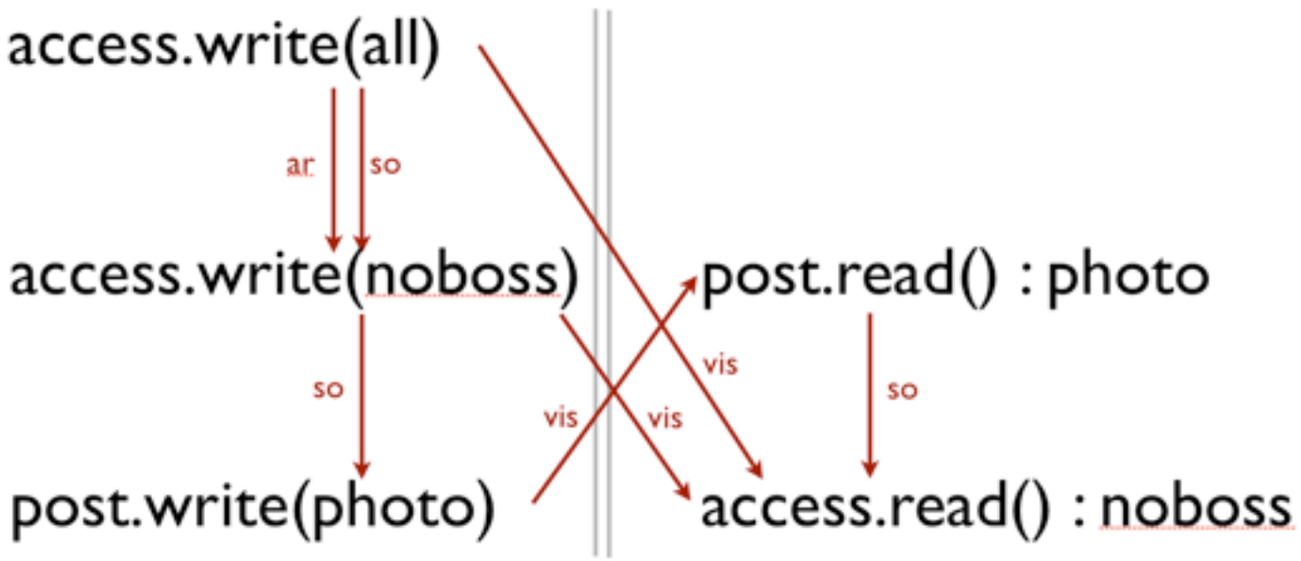


Figure 1. Axioms of eventual consistency

WELL-FORMEDNESS AXIOMS

SOWF: so is the union of transitive, irreflexive and total orders on actions by each session

VISWF: $\forall a, b.\ a \xrightarrow{\mathsf{vis}} b \implies \mathsf{obj}(a) = \mathsf{obj}(b)$

ARWF: $\forall a, b.\ a \xrightarrow{\mathsf{ar}} b \implies \mathsf{obj}(a) = \mathsf{obj}(b)$, ar is transitive and irreflexive, and $\mathsf{ar}|_{\mathsf{vis}^{-1}(a)}$ is a total order for all $a \in A$

AUXILIARY RELATIONS

Per-object session order: $\mathsf{soo} = (\mathsf{so} \cap \mathsf{sameobj})$

Per-object causality order: $\mathsf{hbo} = (\mathsf{soo} \cup \mathsf{vis})^+$

Causality order: $\mathsf{hb} = (\mathsf{so} \cup \mathsf{vis})^+$

BASIC EVENTUAL CONSISTENCY AXIOMS

RVAL: $\forall a \in A.\ \mathsf{rval}(a) = F_{\mathsf{type}(a)}(\mathsf{cone}(a))$

EVENTUAL: $\forall a \in A. \neg(\exists \text{ infinitely many } b \in A. \mathsf{sameobj}(a, b) \wedge \neg(a \xrightarrow{\mathsf{vis}} b))$

THINAIR: so ∪ vis is acyclic

SESSION GUARANTEES

RYW (Read Your Writes): $\mathsf{soo} \subseteq \mathsf{vis}$

MR (Monotonic Reads): $(\mathsf{vis}; \mathsf{soo}) \subseteq \mathsf{vis}$

WFRV (Writes Follow Reads in Visibility): $(\mathsf{vis}; \mathsf{soo}^*; \mathsf{vis}) \subseteq \mathsf{vis}$

WFRA (Writes Follow Reads in Arbitration): $(\mathsf{vis}; \mathsf{soo}^*) \subseteq \mathsf{ar}$

MWV (Monotonic Writes in Visibility): $(\mathsf{soo}; \mathsf{vis}) \subseteq \mathsf{vis}$

MWA (Monotonic Writes in Arbitration): $\mathsf{soo} \subseteq \mathsf{ar}$

CAUSALITY AXIOMS

POCV (Per-Object Causal Visibility): $\mathsf{hbo} \subseteq \mathsf{vis}$

POCA (Per-Object Causal Arbitration): $\mathsf{hbo} \subseteq \mathsf{ar}$

COCV (Cross-Object Causal Visibility): $(\mathsf{hb} \cap \mathsf{sameobj}) \subseteq \mathsf{vis}$

COCA (Cross-Object Causal Arbitration): hb ∪ ar is acyclic

Axioms

Basic eventual consistency

Session guarantees

Per-object causal consistency

Causal consistency

Strong consistency

Figure 1. Axioms of eventual consistency

WELL-FORMEDNESS AXIOMS

SOWF: so is the union of transitive, irreflexive and total orders on actions by each session

VISWF: $\forall a, b.\, a \xrightarrow{\mathsf{vis}} b \implies \mathsf{obj}(a) = \mathsf{obj}(b)$

ARWF: $\forall a, b.\, a \xrightarrow{\mathsf{ar}} b \implies \mathsf{obj}(a) = \mathsf{obj}(b)$, ar is transitive and irreflexive, and $\mathsf{ar}|_{\mathsf{vis}^{-1}(a)}$ is a total order for all $a \in A$

AUXILIARY RELATIONS

Per-object session order: $\mathsf{soo} = (\mathsf{so} \cap \mathsf{sameobj})$

Per-object causality order: $\mathsf{hbo} = (\mathsf{soo} \cup \mathsf{vis})^+$

Causality order: $\mathsf{hb} = (\mathsf{so} \cup \mathsf{vis})^+$

BASIC EVENTUAL CONSISTENCY AXIOMS

RVAL: $\forall a \in A.\, \mathsf{rval}(a) = F_{\mathsf{type}(a)}(\mathsf{cone}(a))$

EVENTUAL: $\forall a \in A. \neg(\exists \text{ infinitely many } b \in A.\, \mathsf{sameobj}(a, b) \wedge \neg(a \xrightarrow{\mathsf{vis}} b))$

THINAIR: so ∪ vis is acyclic

SESSION GUARANTEES

RYW (Read Your Writes): $\mathsf{soo} \subseteq \mathsf{vis}$

MR (Monotonic Reads): $(\mathsf{vis}; \mathsf{soo}) \subseteq \mathsf{vis}$

WFRV (Writes Follow Reads in Visibility): $(\mathsf{vis}; \mathsf{soo}^*; \mathsf{vis}) \subseteq \mathsf{vis}$

WFRA (Writes Follow Reads in Arbitration): $(\mathsf{vis}; \mathsf{soo}^*) \subseteq \mathsf{ar}$

MWV (Monotonic Writes in Visibility): $(\mathsf{soo}; \mathsf{vis}) \subseteq \mathsf{vis}$

MWA (Monotonic Writes in Arbitration): $\mathsf{soo} \subseteq \mathsf{ar}$

CAUSALITY AXIOMS

POCV (Per-Object Causal Visibility): $\mathsf{hbo} \subseteq \mathsf{vis}$

POCA (Per-Object Causal Arbitration): $\mathsf{hbo} \subseteq \mathsf{ar}$

COCV (Cross-Object Causal Visibility): $(\mathsf{hb} \cap \mathsf{sameobj}) \subseteq \mathsf{vis}$

COCA (Cross-Object Causal Arbitration): hb ∪ ar is acyclic

Formalising "eventual"

EVENTUAL.

$\forall a \in A. \neg(\exists \text{ infinitely many } b \in A. \mathsf{sameobj}(a, b) \wedge \neg(a \xrightarrow{\mathsf{vis}} b))$

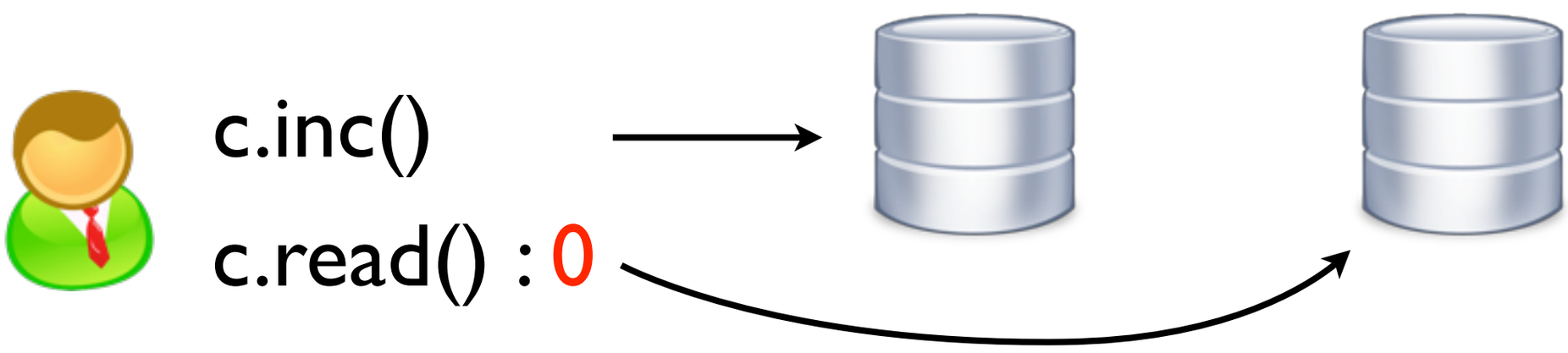
An action cannot be invisible to infinitely many actions on the same object

Formalising "eventual"

EVENTUAL.

$\forall a \in A. \neg(\exists \text{ infinitely many } b \in A. \mathsf{sameobj}(a, b) \wedge \neg(a \xrightarrow{\mathsf{vis}} b))$

An action cannot be invisible to infinitely many actions on the same object

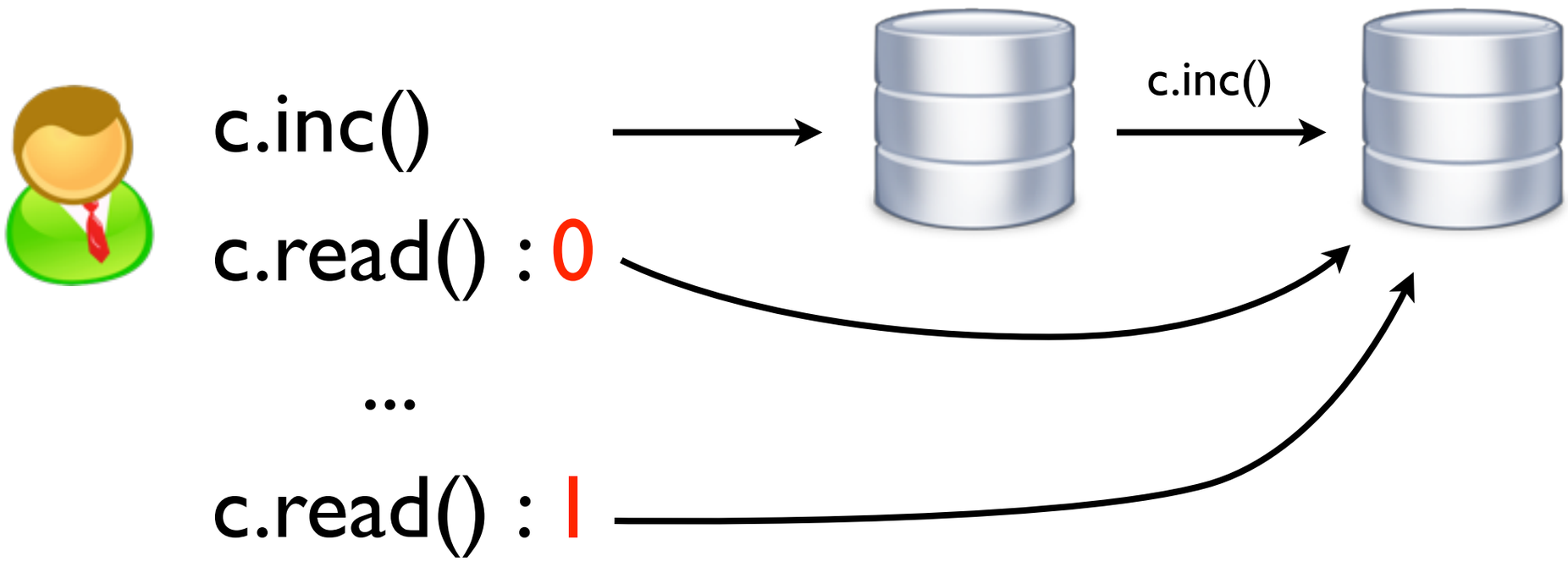


Formalising "eventual"

EVENTUAL.

$\forall a \in A. \neg(\exists \text{ infinitely many } b \in A. \mathsf{sameobj}(a, b) \wedge \neg(a \xrightarrow{\mathsf{vis}} b))$

An action cannot be invisible to infinitely many actions on the same object



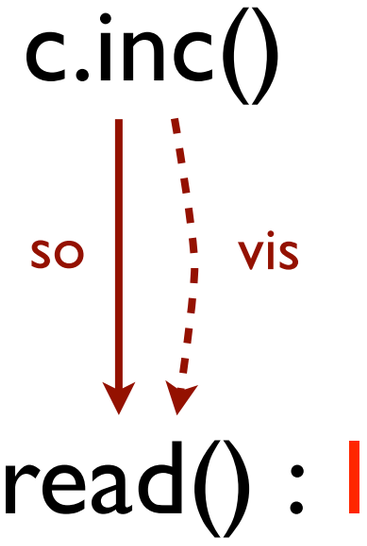
Strengthening consistency

General principle: mandate that certain actions be visible by including additional edges into vis

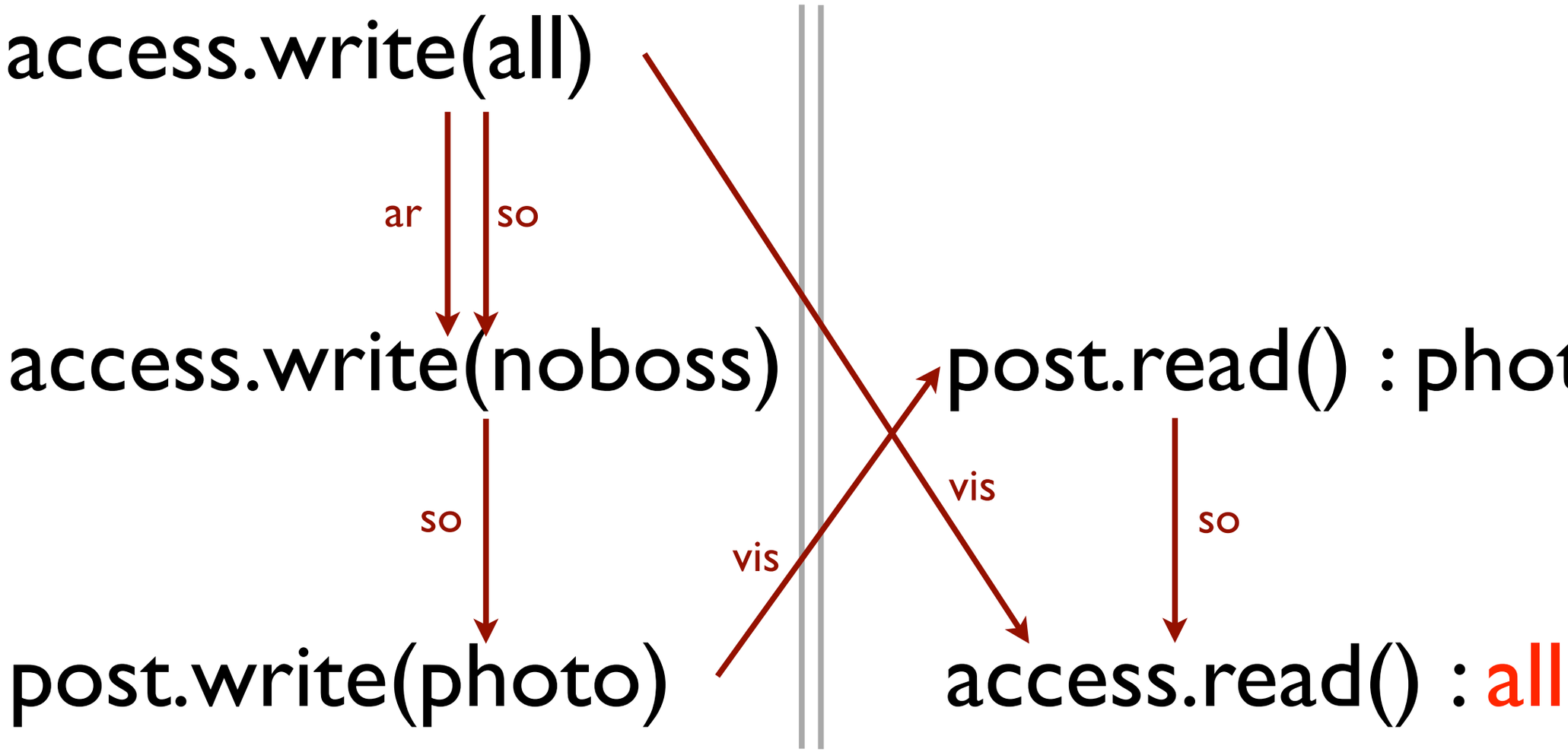
READ YOUR WRITES: so ∩ sameobj ⊆ vis

c.inc()

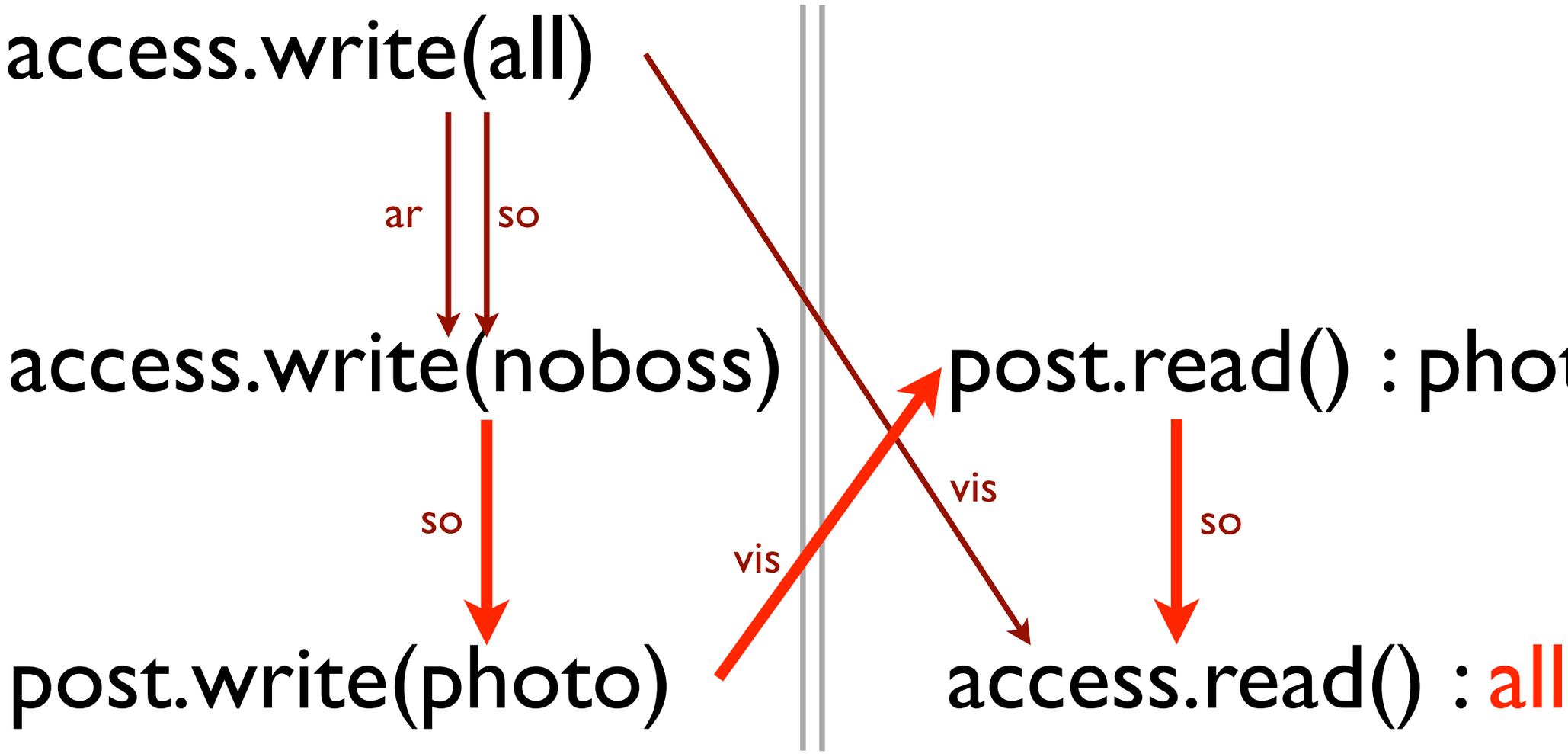
c.read() : 0



How do we disallow this?

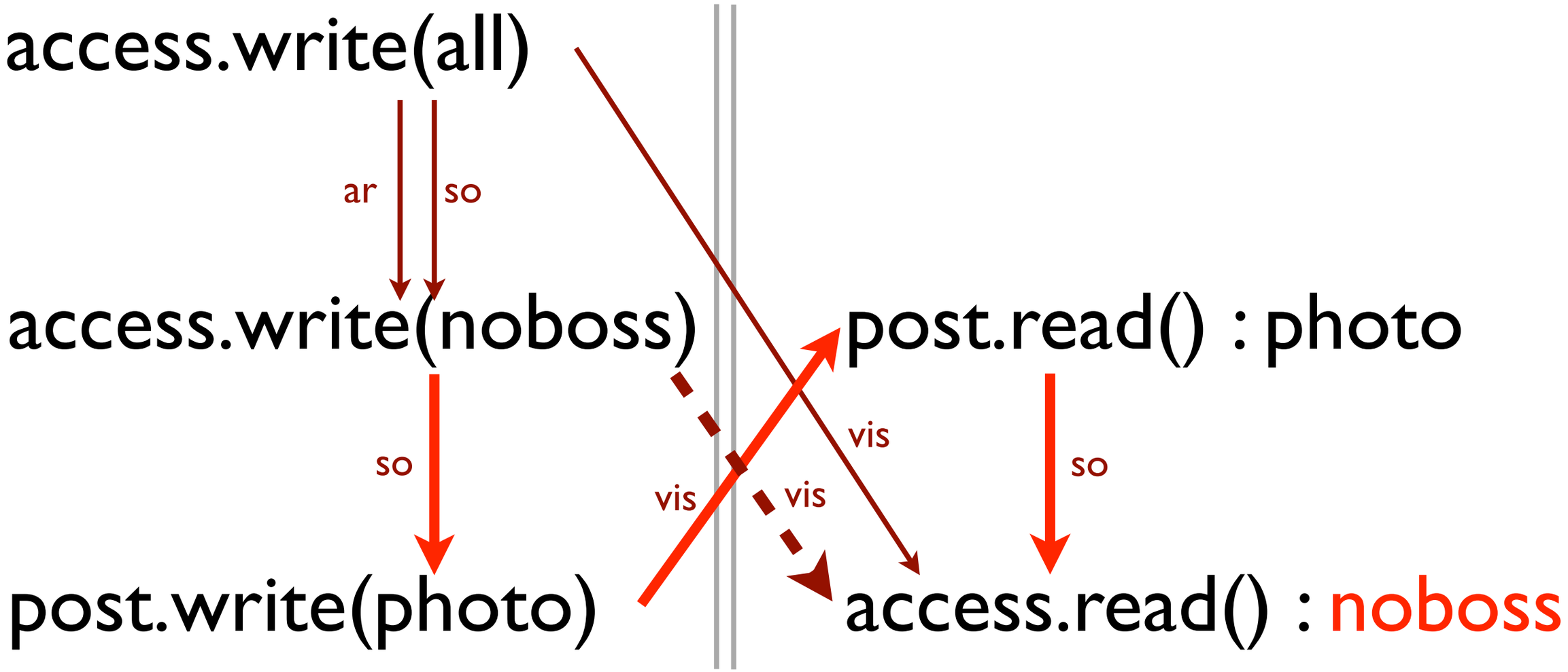


How do we disallow this?



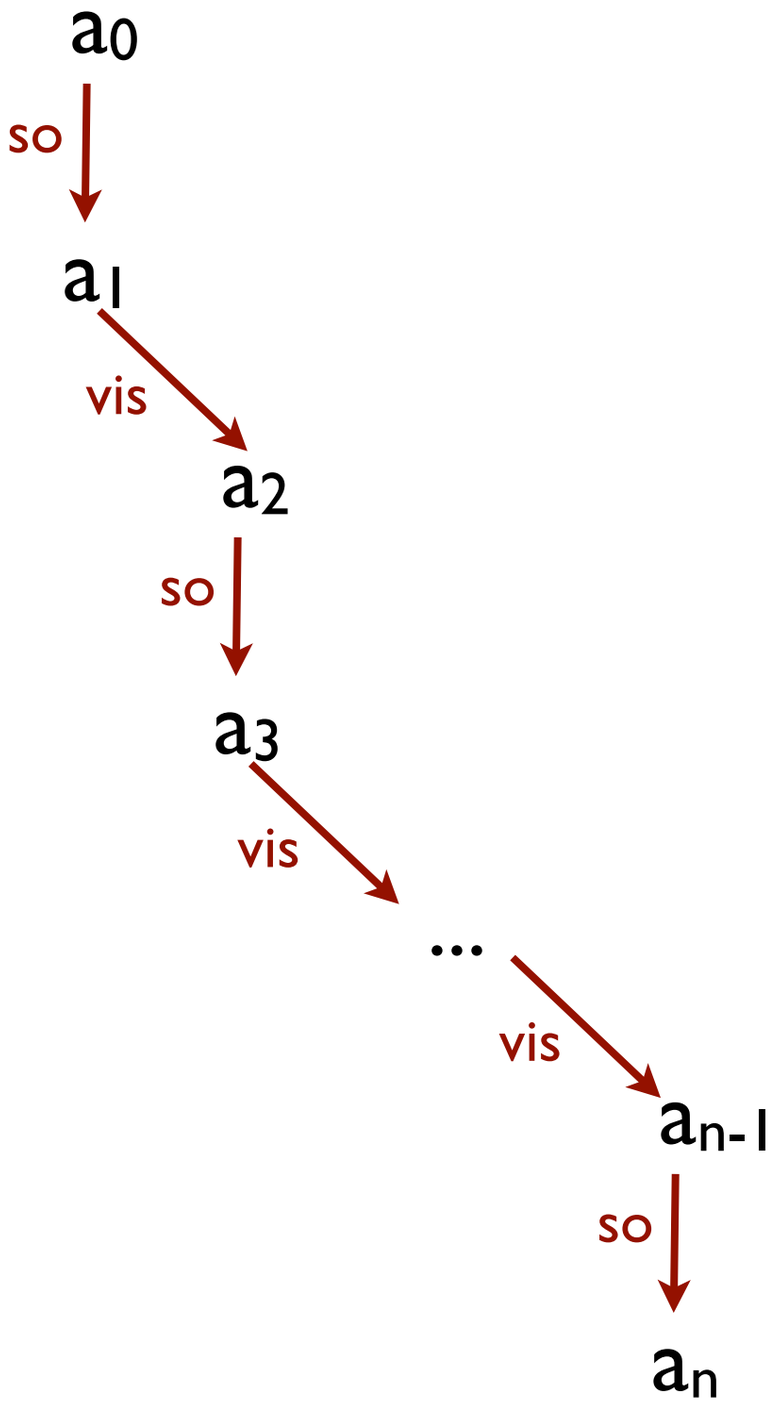
write(noboss) happened-before the read,
but the read didn't see it

How do we disallow this?



Mandate that all actions that
happened before have to be visible

Causal consistency



Happens-before relation:

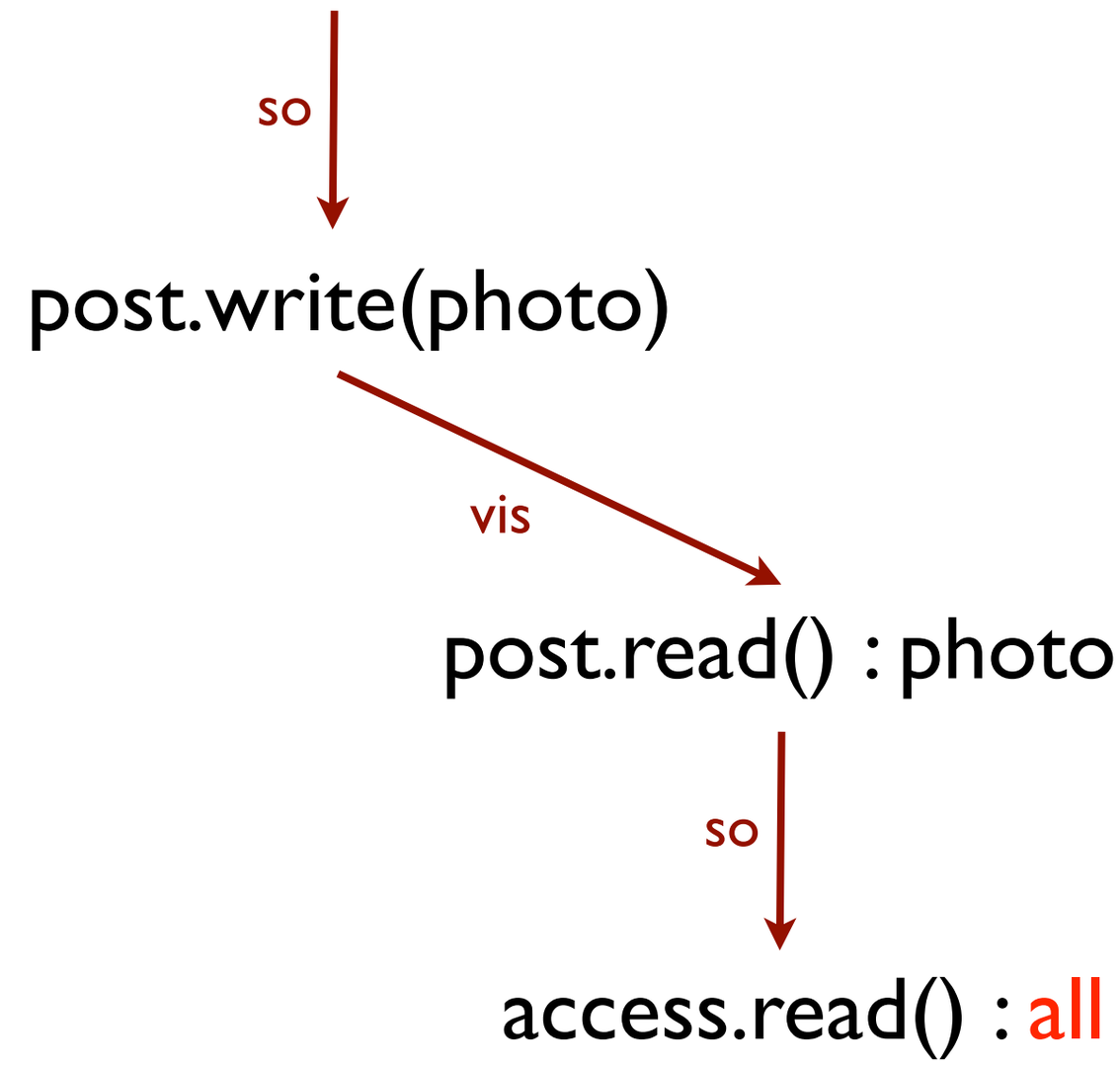
$$\mathsf{hb} = (\mathsf{so} \cup \mathsf{vis})^{+}$$

CAUSAL CONSISTENCY:

$$(\mathsf{hb} \cap \mathsf{sameobj}) \subseteq \mathsf{vis}$$

$$(\mathsf{hb} \cap \mathsf{sameobj}) \subseteq \mathsf{ar}$$

Causal consistency



Happens-before relation:

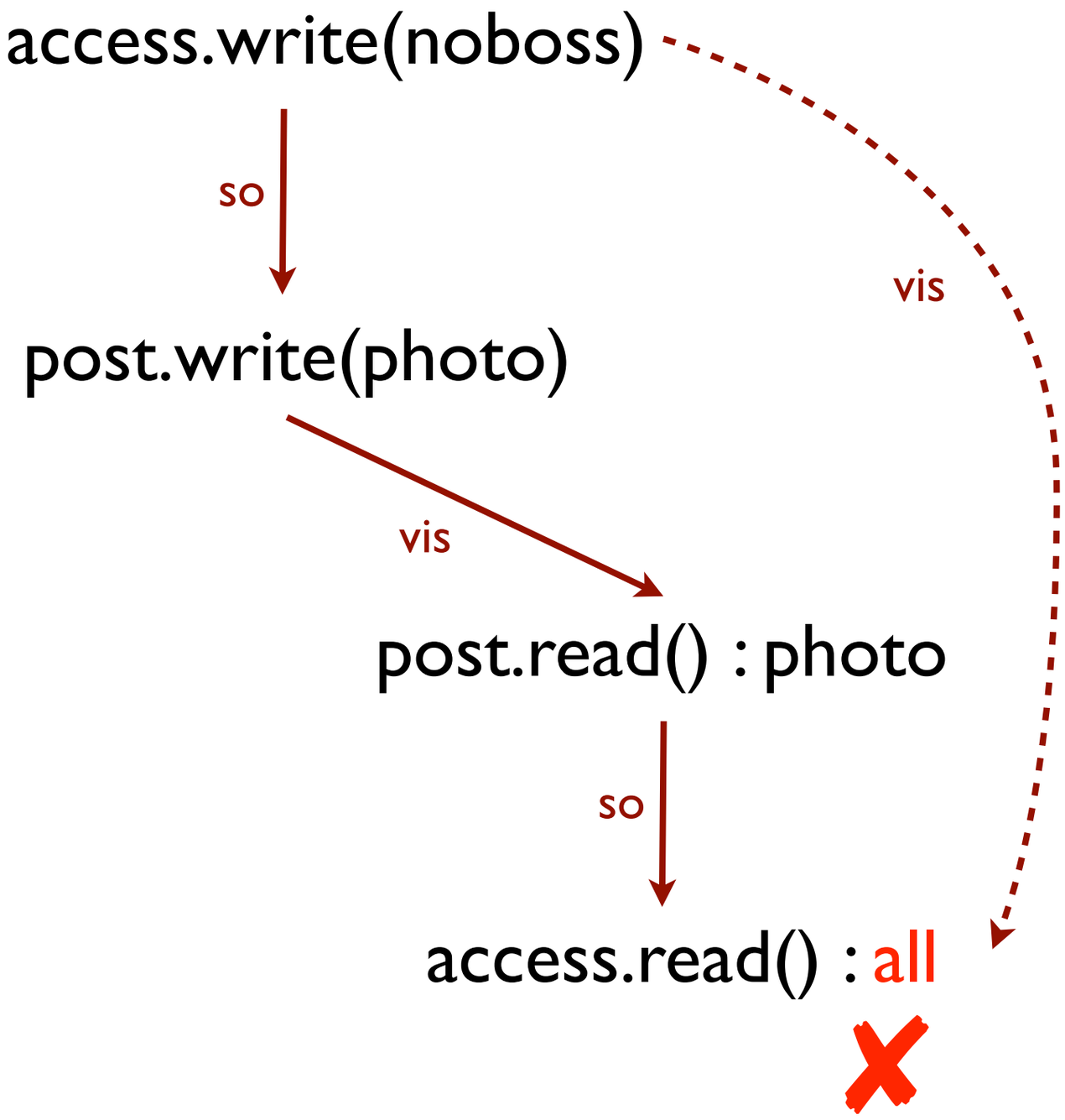
$$hb = (so \cup vis)^+$$

CAUSAL CONSISTENCY:

$$(hb \cap sameobj) \subseteq vis$$

$$(hb \cap sameobj) \subseteq ar$$

Causal consistency



Happens-before relation:

$$hb = (so \cup vis)^{+}$$

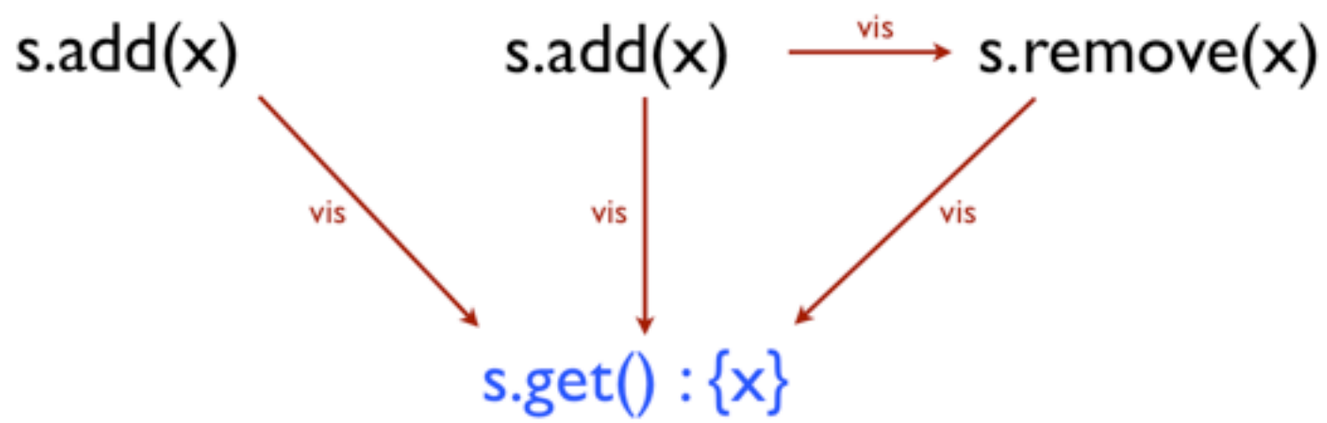
CAUSAL CONSISTENCY:

$$(hb \cap sameobj) \subseteq vis$$

$$(hb \cap sameobj) \subseteq ar$$

Technical take-aways

- Ways in which the database processes requests specified declaratively by vis and ar
- Conflict resolution policies specified by functions of vis and ar



- Consistency strengthened by including additional edges into vis or ar

READ YOUR WRITES:

so ∩ sameobj ⊆ vis

CAUSAL CONSISTENCY:

(hb ∩ sameobj) ⊆ vis

(hb ∩ sameobj) ⊆ ar

What does this buy us?

- Exploiting testing and verification technology developed for weak memory models

  Throw the axiomatic model at a SAT solver to do bounded model checking

- Letting the programmer switch between different types of eventual consistency within the same system implementation

  E.g., using fences, like in weak memory

- Compositional reasoning about eventually consistent systems

The End

Draft paper available at

http://www.cs.ox.ac.uk/people/hongseok.yang/Public/Publications.html