Semantics of eventual consistency

Work in progress

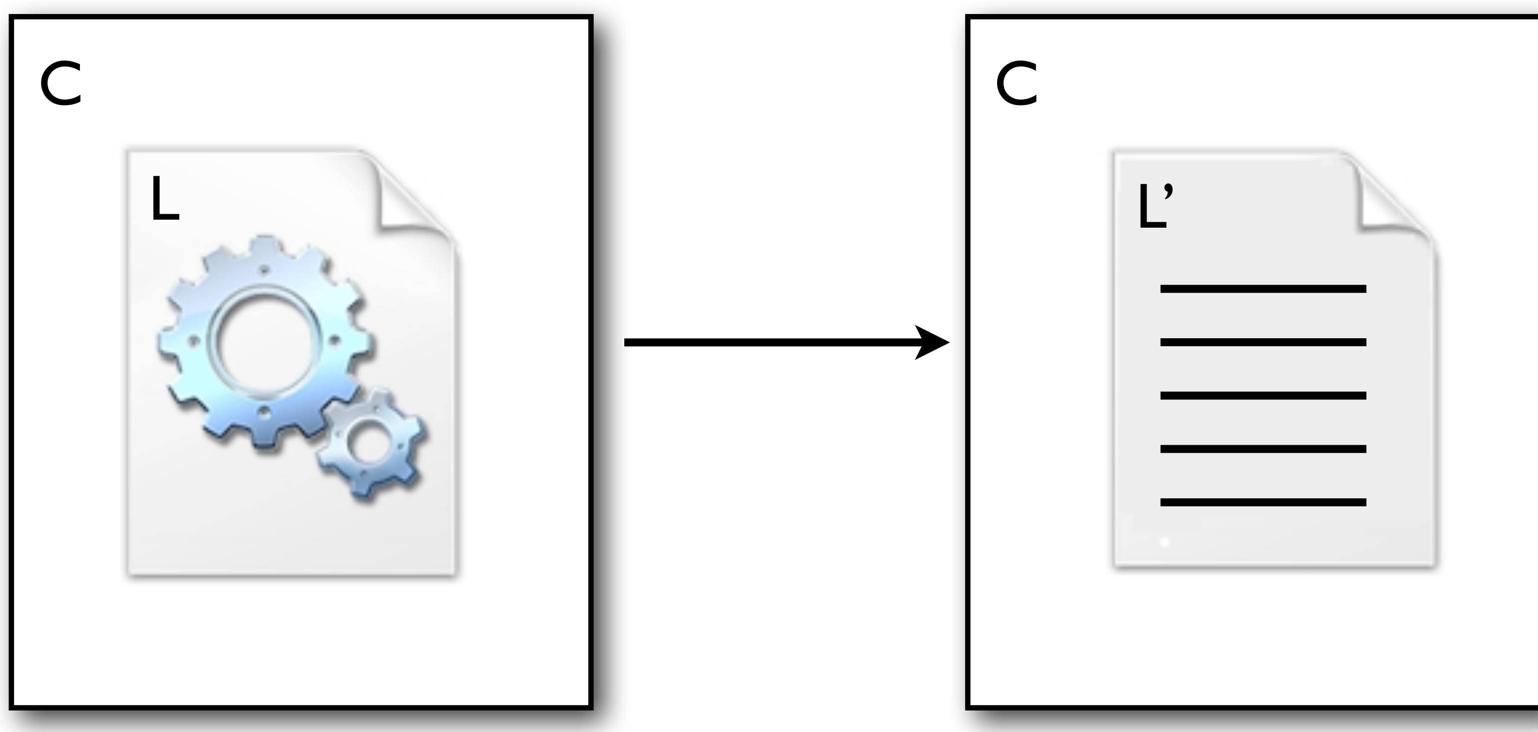
Alexey Gotsman

IMDEA Software Institute, Madrid, Spain

Joint work with Sebastian Burckhardt (MSR) and Hongseok Yang (Oxford)

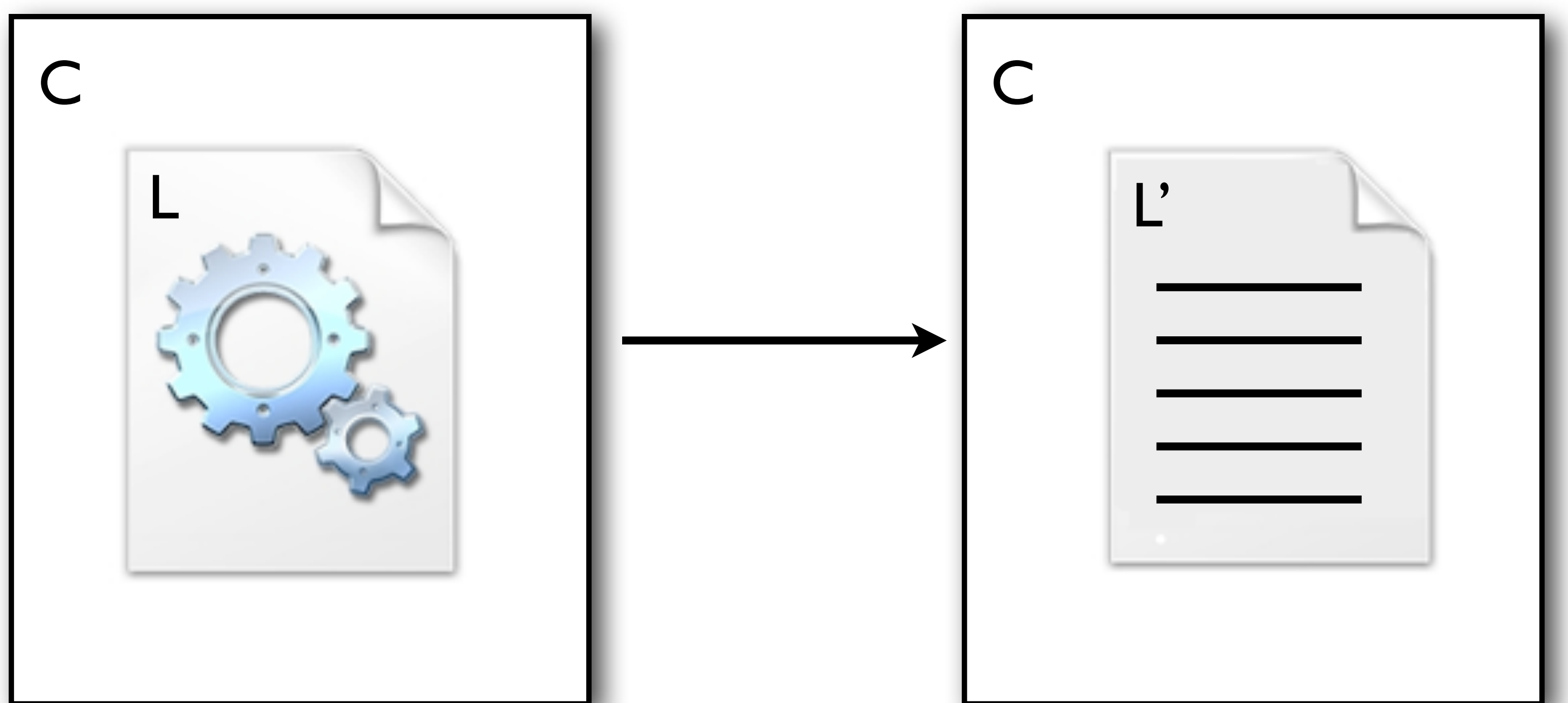
What am I doing here?

- Original interest: verification of concurrent programs
- Want to exploit program structure
- Don’t want to consider the internals of L while verifying C:



What am I doing here?

- Original interest: verification of concurrent programs
- Want to exploit program structure
- Don't want to consider the internals of L while verifying C:



Main challenge: in practice, library interfaces are complicated

Factory of correctness definitions

- Liveness properties [ICALP'11]
- Communication via data structures [CONCUR'12]
- Weak memory: x86 [ESOP'12, DISC'12]
- Weak memory: C/C++ [POPL'13]

Factory of correctness definitions

- Liveness properties [ICALP'11]
- Communication via data structures [CONCUR'12]
- Weak memory: x86 [ESOP'12, DISC'12]
- Weak memory: C/C++ [POPL'13]

Processors and languages do not provide sequential consistency

A multiprocessor is really a distributed system

"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

Rule out some anomalies

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

Preserve causality

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

Add replicated data types

Transactional storage for geo-replicated systems

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

Add transactions

50 sh

Session Gu

Douglas B. Terry, Alar

- 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?

Scalable Causa

causality

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

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

Add replicated data types

Transactional storage for geo-replicated systems

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

Add transactions

Main message

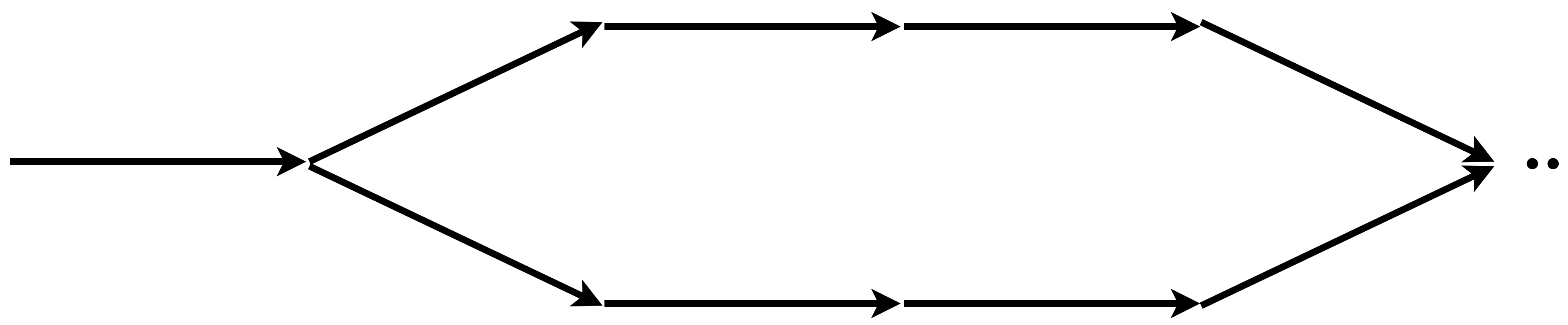
We can use lessons from shared-memory models

- A framework for declarative specification of consistency models for the whole (?) zoo:
  - different replicated data types
  - different consistency levels
  - transactions
- Opens lots of opportunities:
  - semantics of combining consistency levels
  - compositional reasoning

Axiomatic memory models

- Executions in sequential consistency: linear sequences

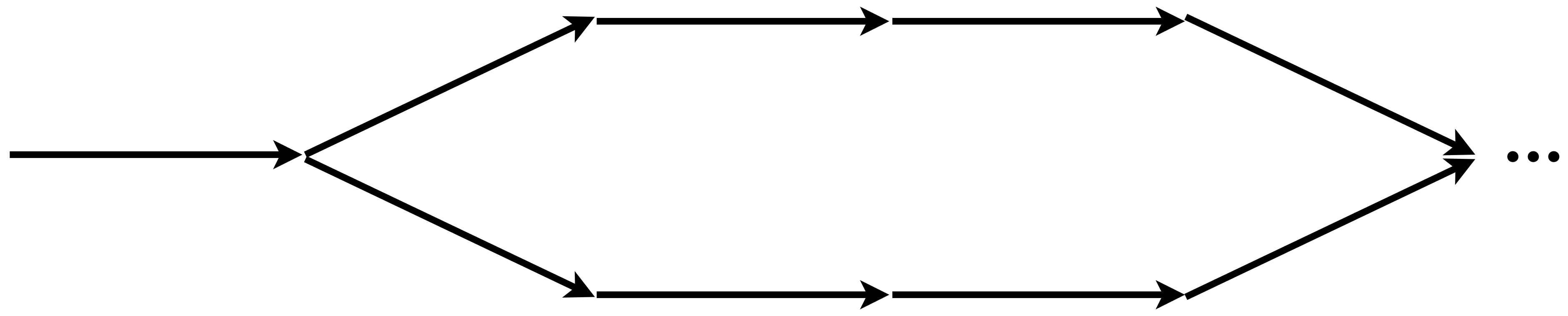
- Executions in axiomatic models: partial orders



Axiomatic memory models

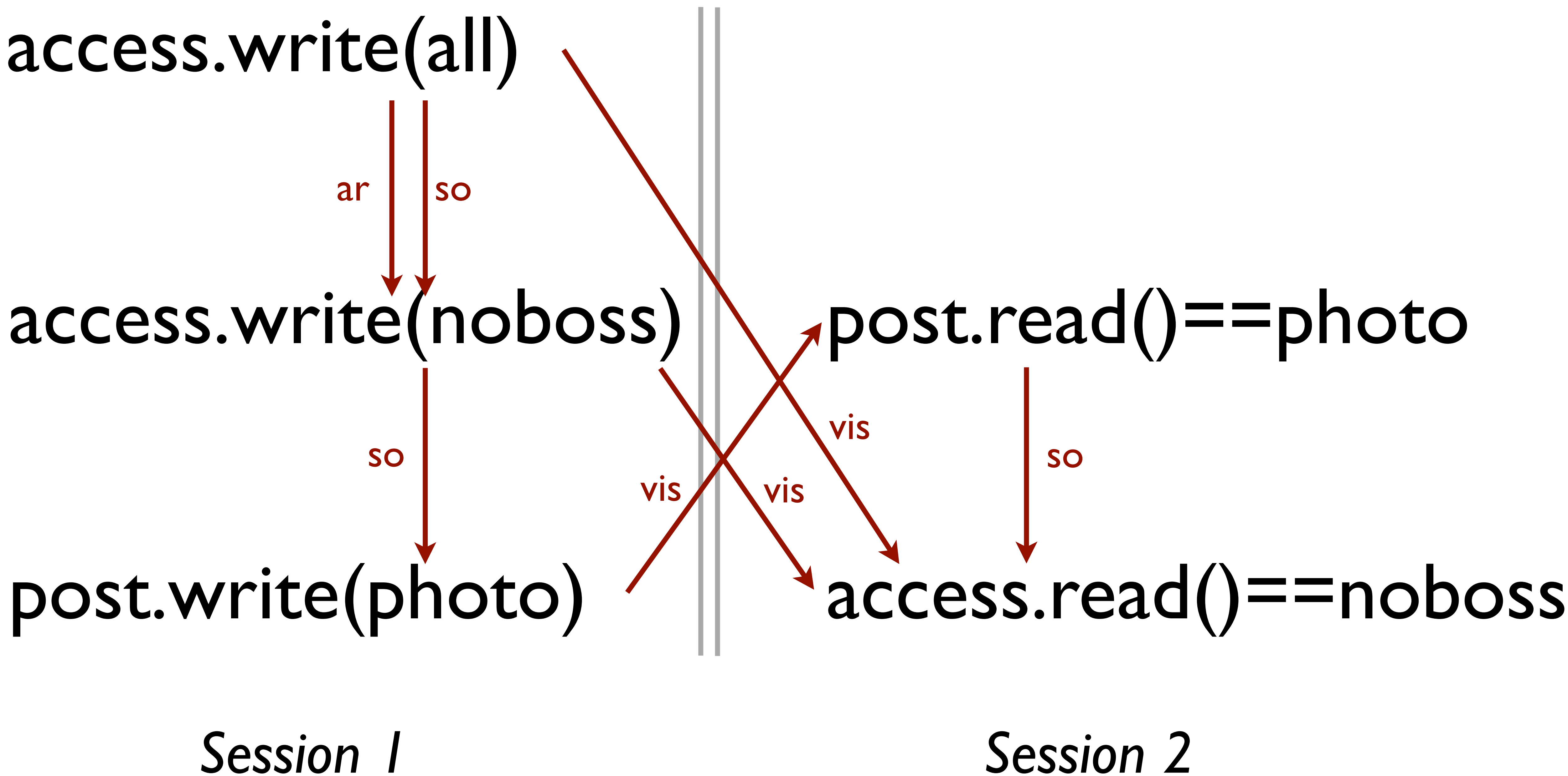
- Executions in sequential consistency: linear sequences

- Executions in axiomatic models: partial orders

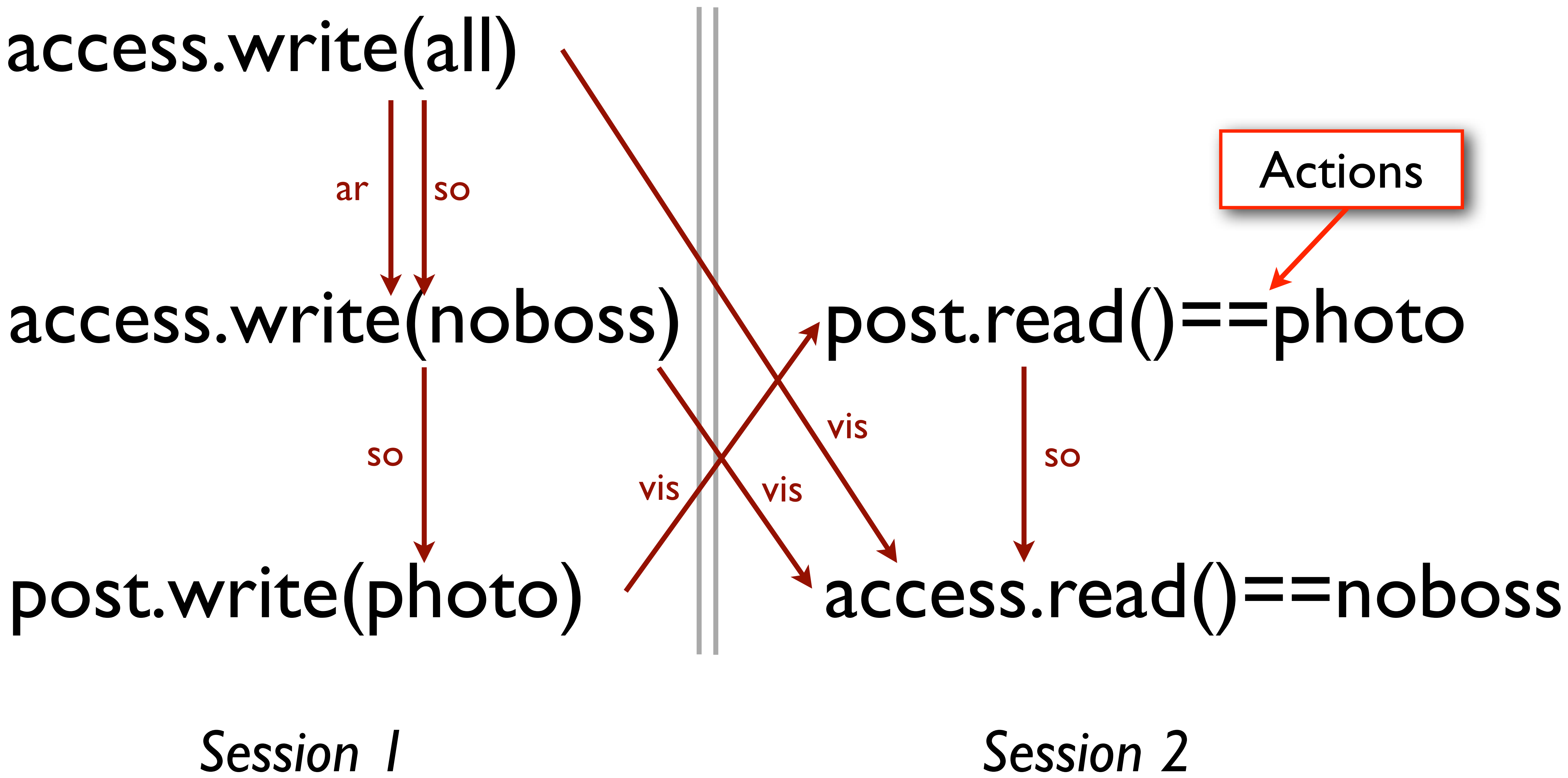


Generalise axiomatic models to replicated data types

Execution: (A, so, vis, ar)

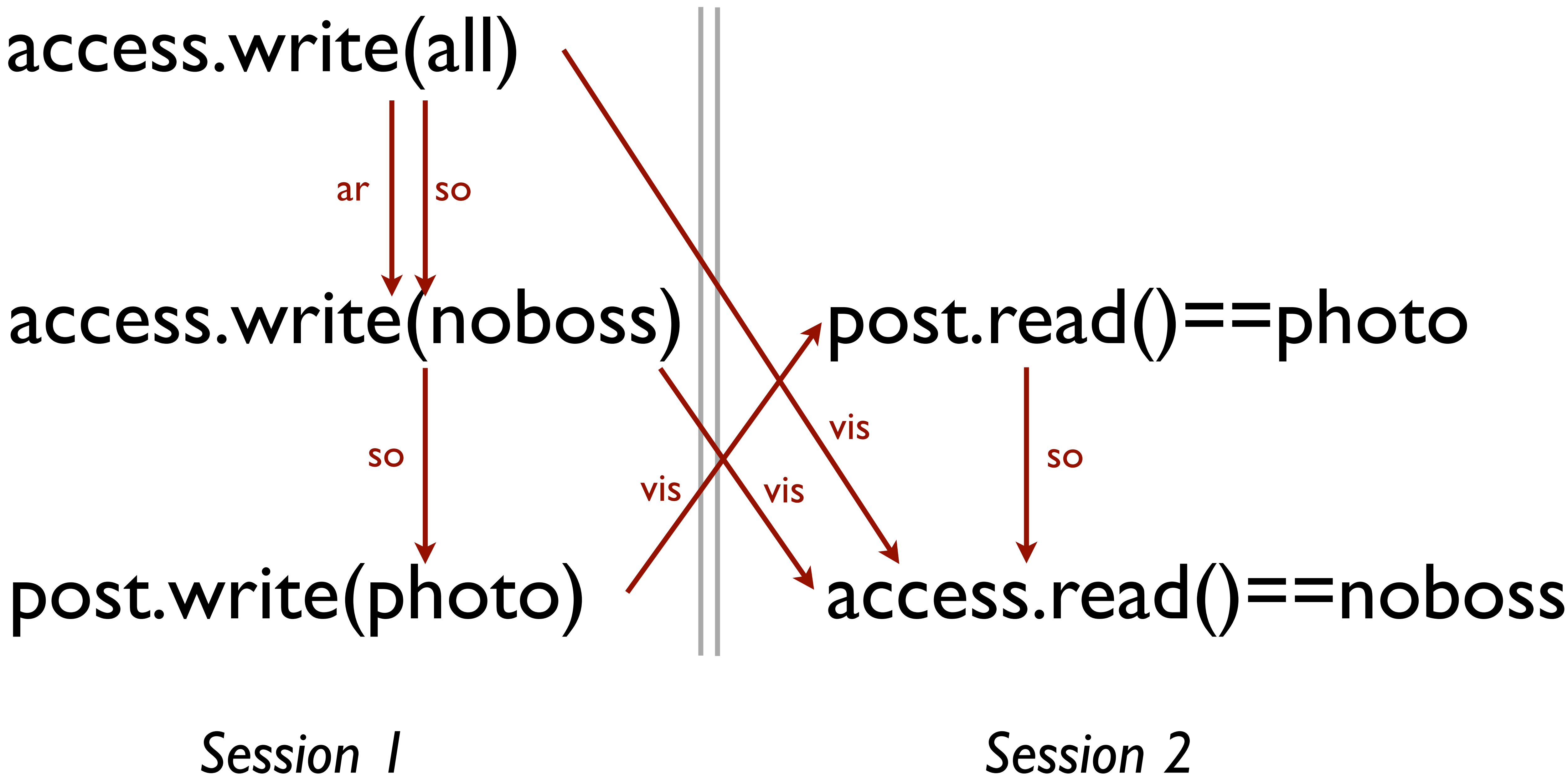


Execution: (A, so, vis, ar)

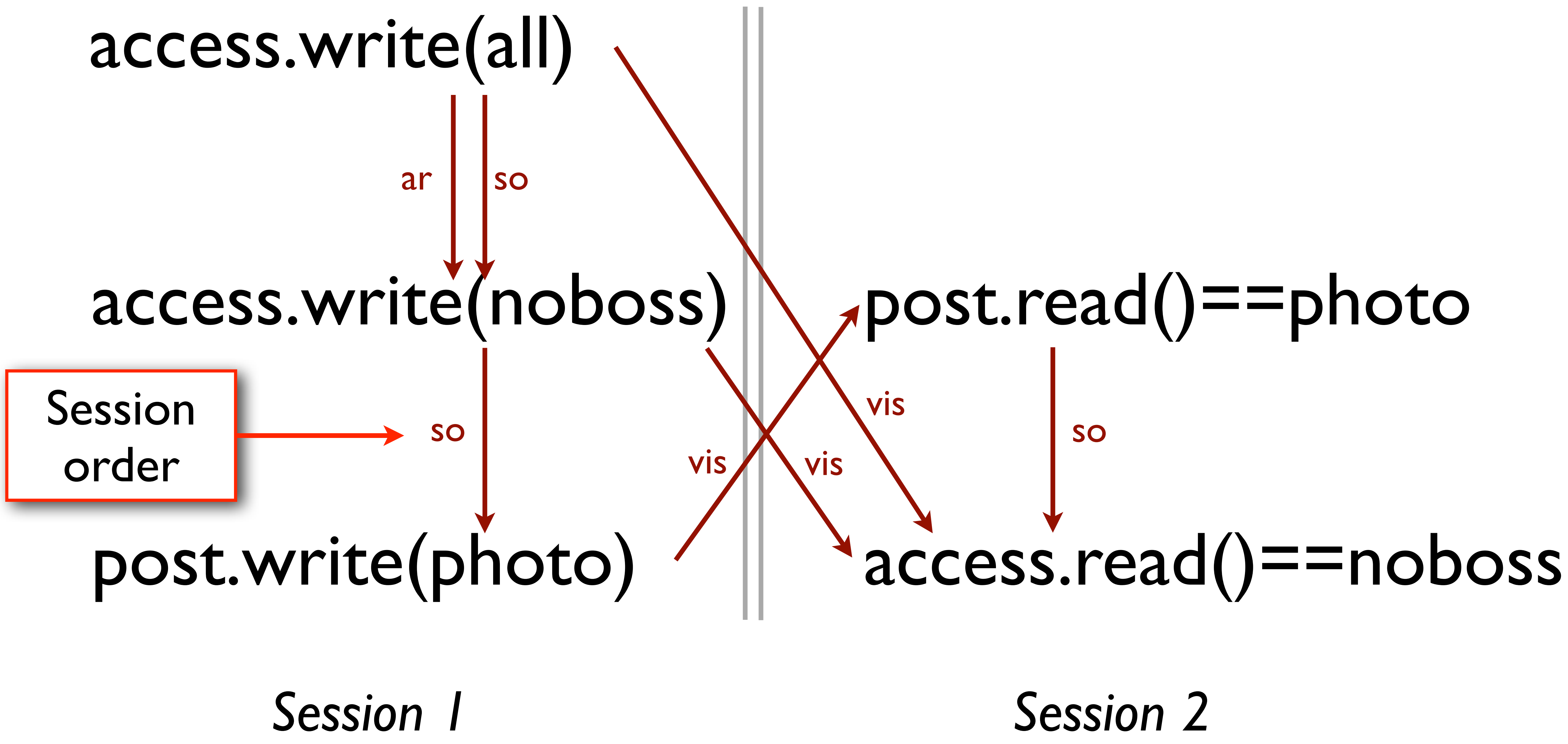


What happens on the interface client/database

Execution: (A, so, vis, ar)

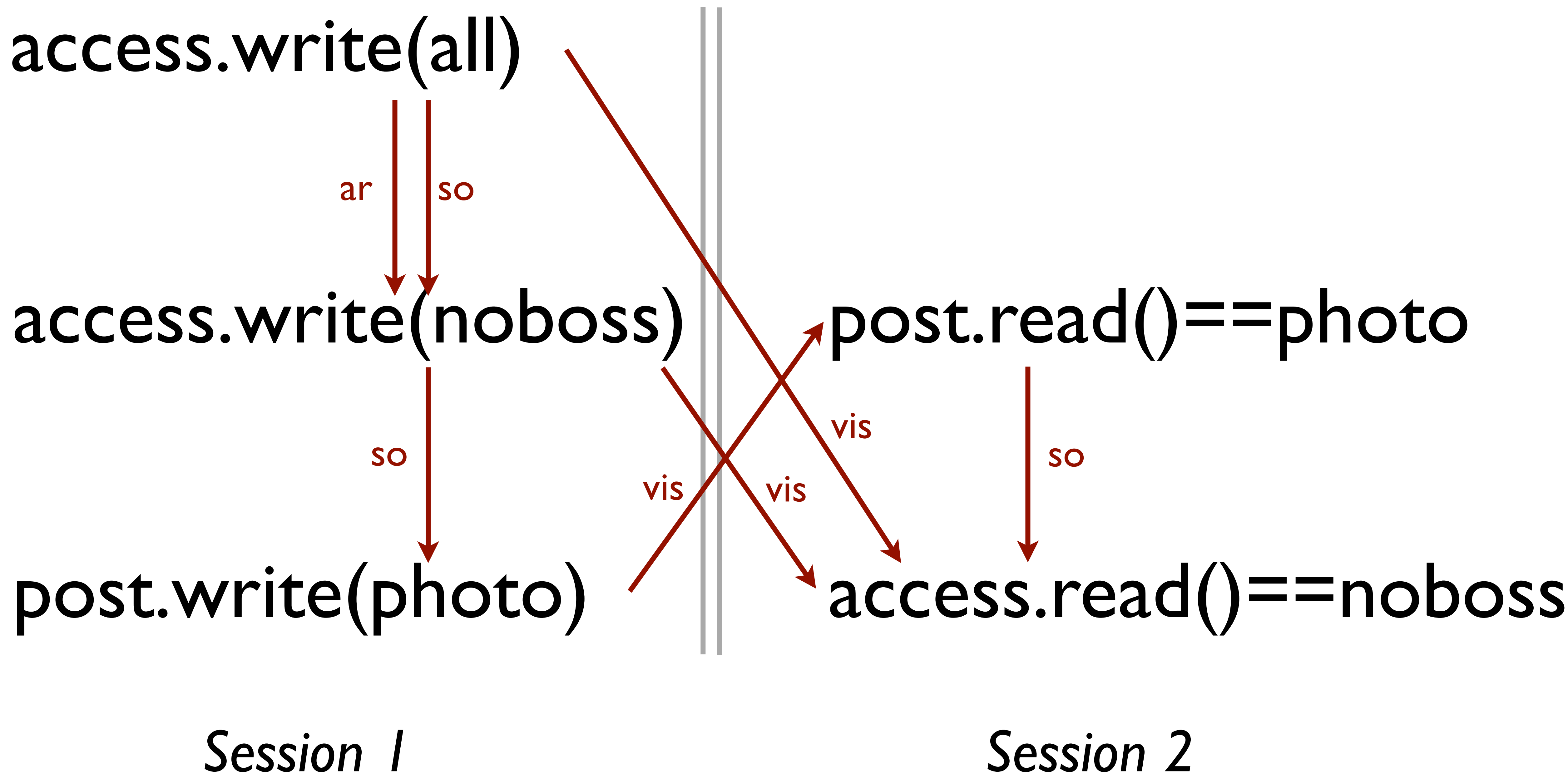


Execution: (A, so, vis, ar)

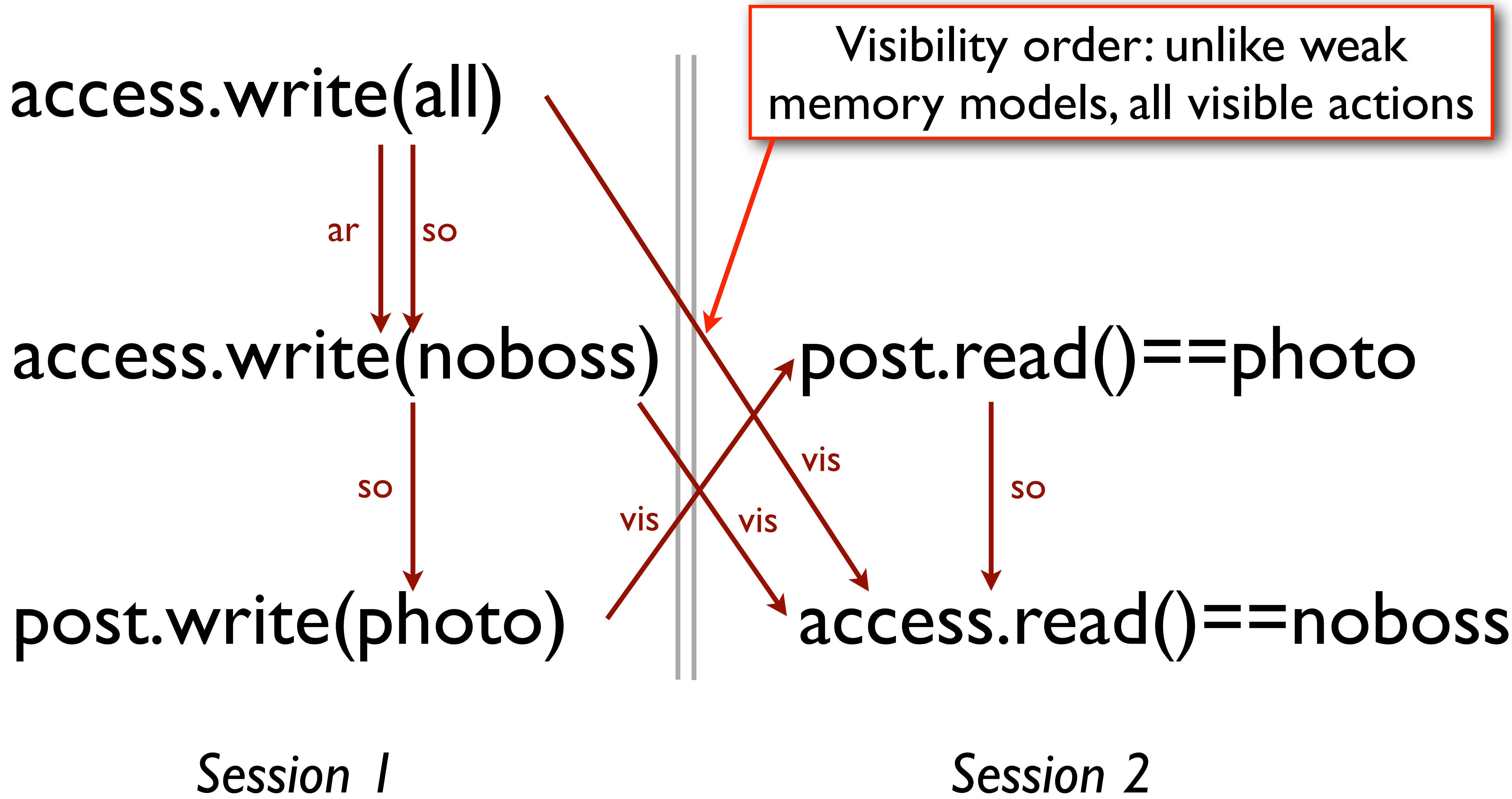


The order of submission to the database

Execution: (A, so, vis, ar)

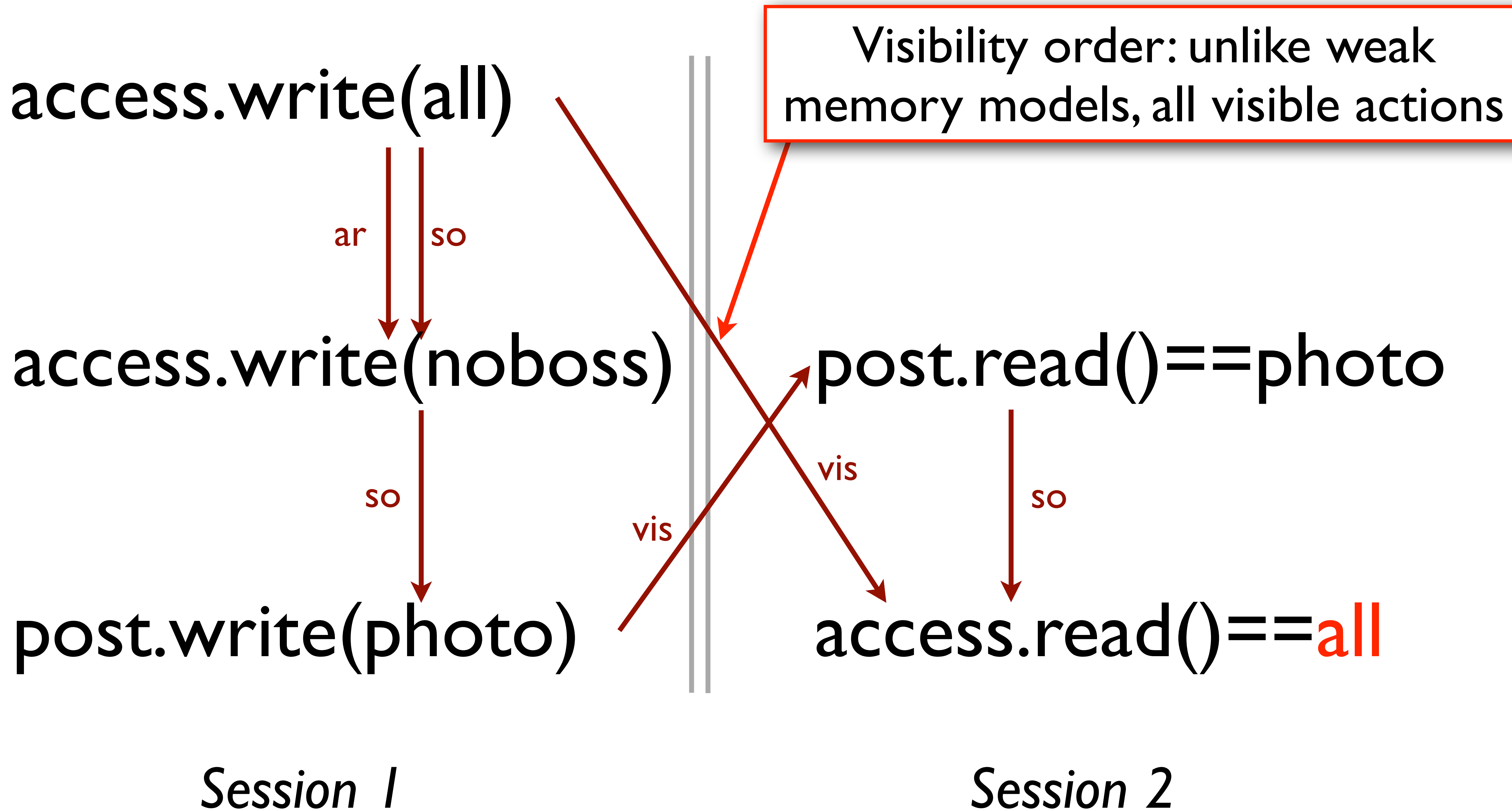


Execution: (A, so, vis, ar)



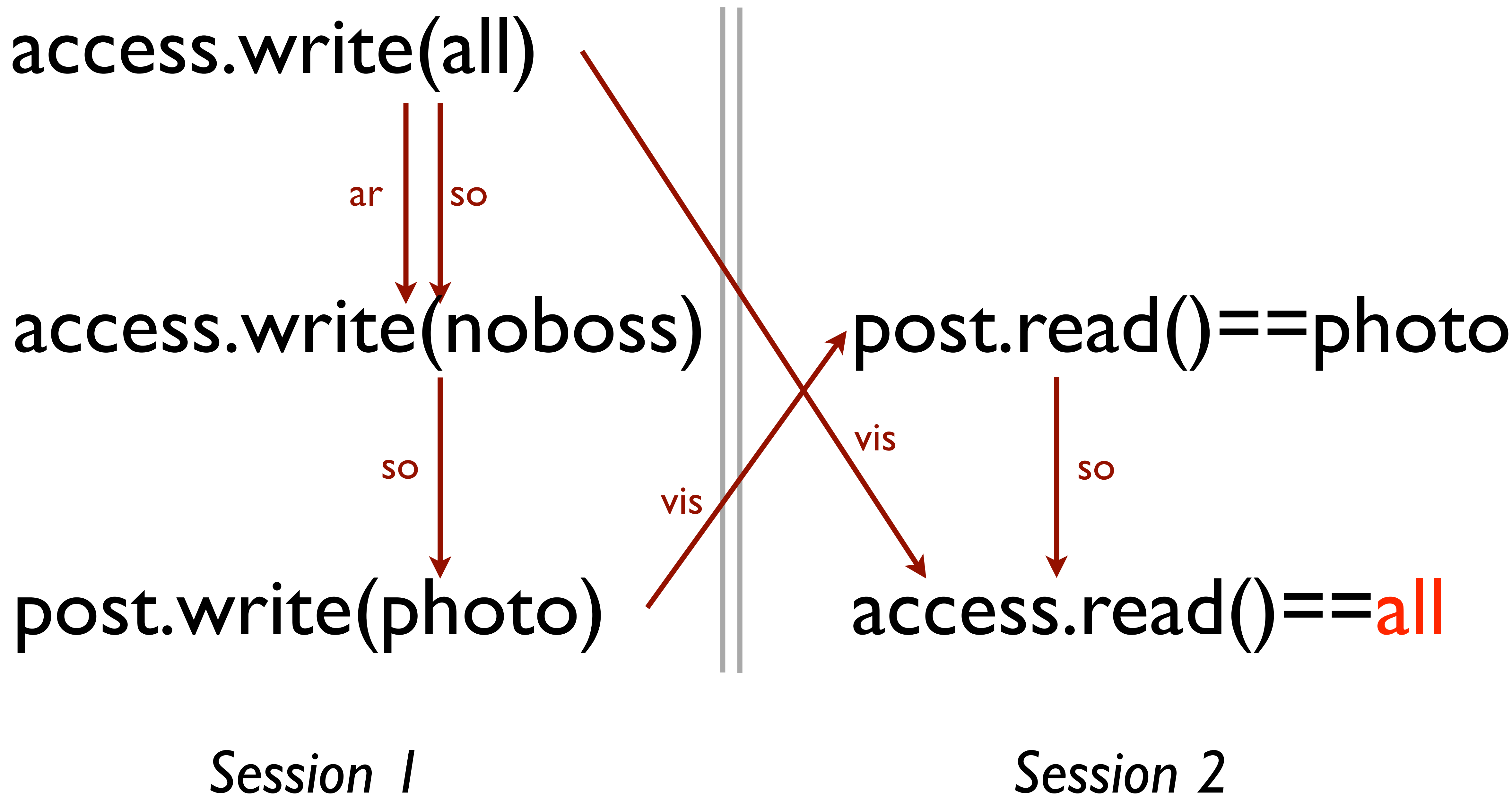
Update delivery (same object)

Execution: (A, so, vis, ar)

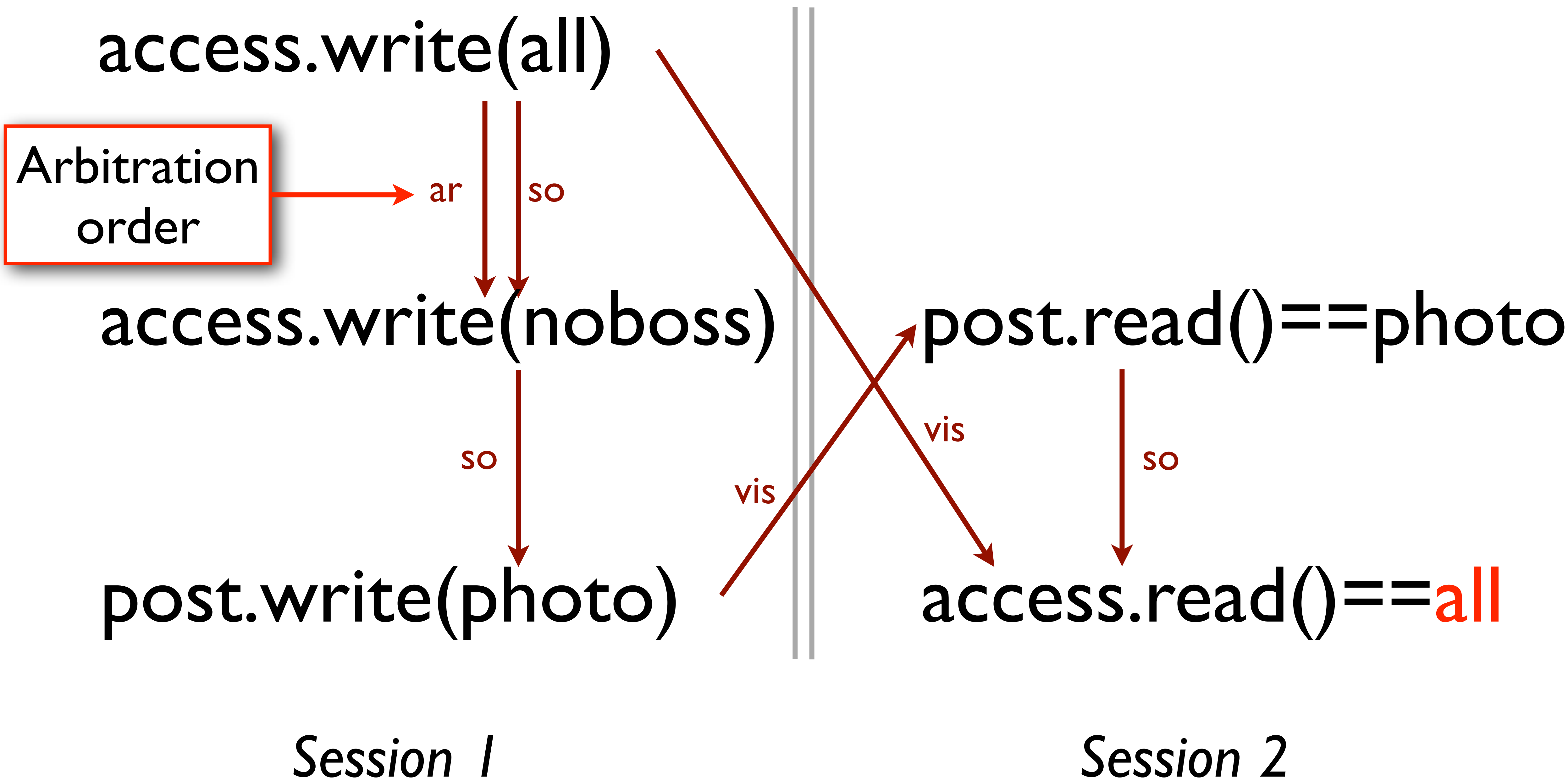


Update delivery (same object)

Execution: (A, so, vis, ar)

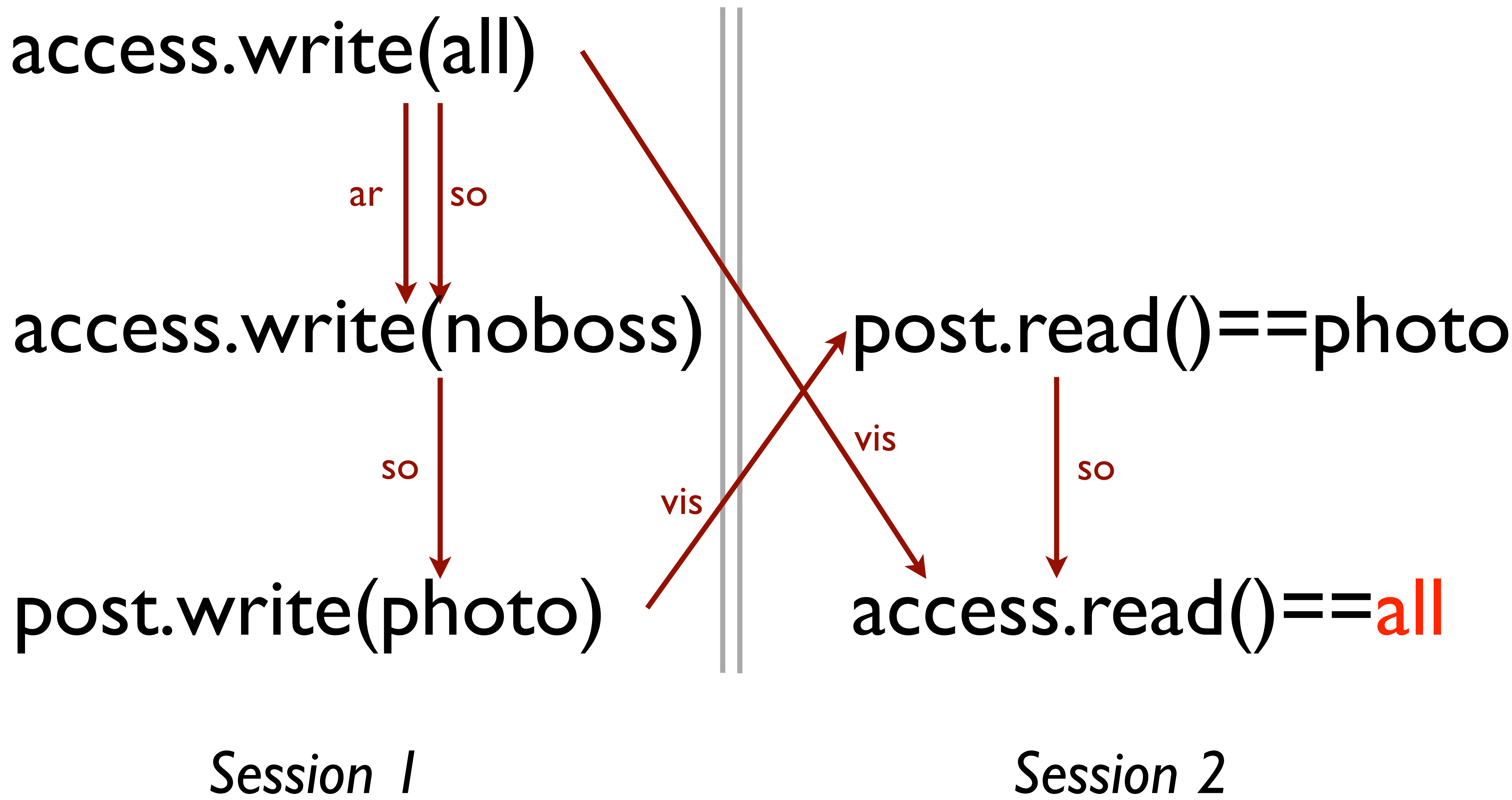


Execution: (A, so, vis, ar)



Lamport time, used in conflict resolution

Execution: (A, so, vis, ar)



System specification = set of executions satisfying axioms:

- Data type specifications
- Consistency constraints

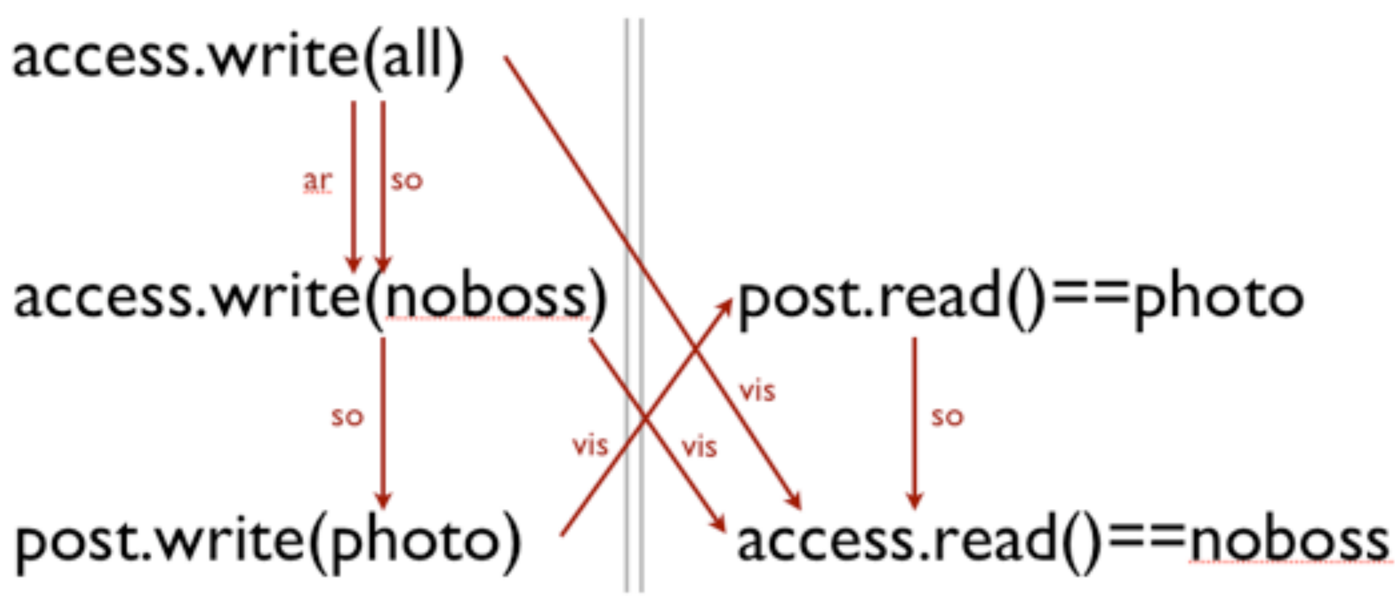


Figure 1. Axioms of eventual consistency

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: $\mathsf{so} \cup \mathsf{vis}$ 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): $\mathsf{hbo} \subseteq \mathsf{ar}$

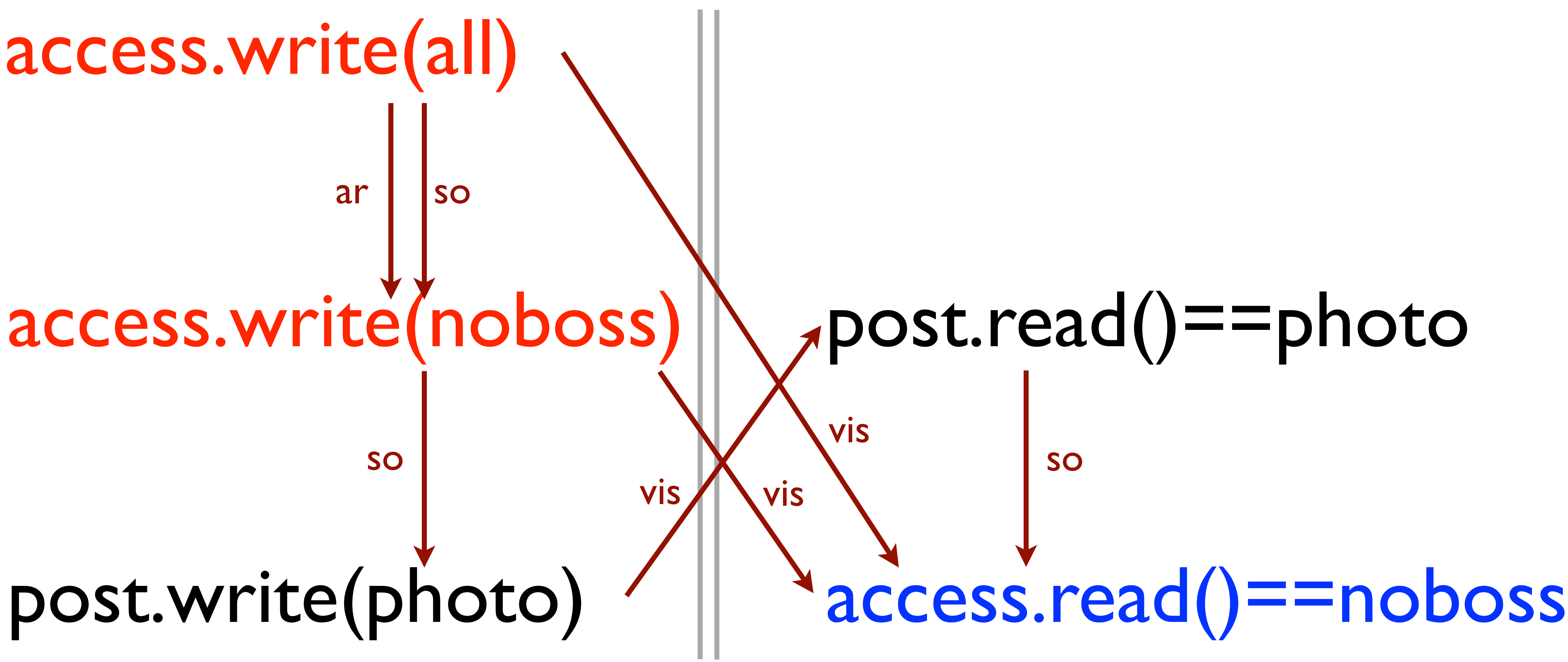
COCV (Cross-Object Causal Visibility): $(\mathsf{hb} \cap \mathsf{sameobj}) \subseteq \mathsf{vis}$

COCA (Cross-Object Causal Arbitration): $\mathsf{hb} \cup \mathsf{ar}$ is acyclic

Data type specification

F: cone of influence → return value

Projection of the execution onto visible actions: (A’, vis’, ar’)

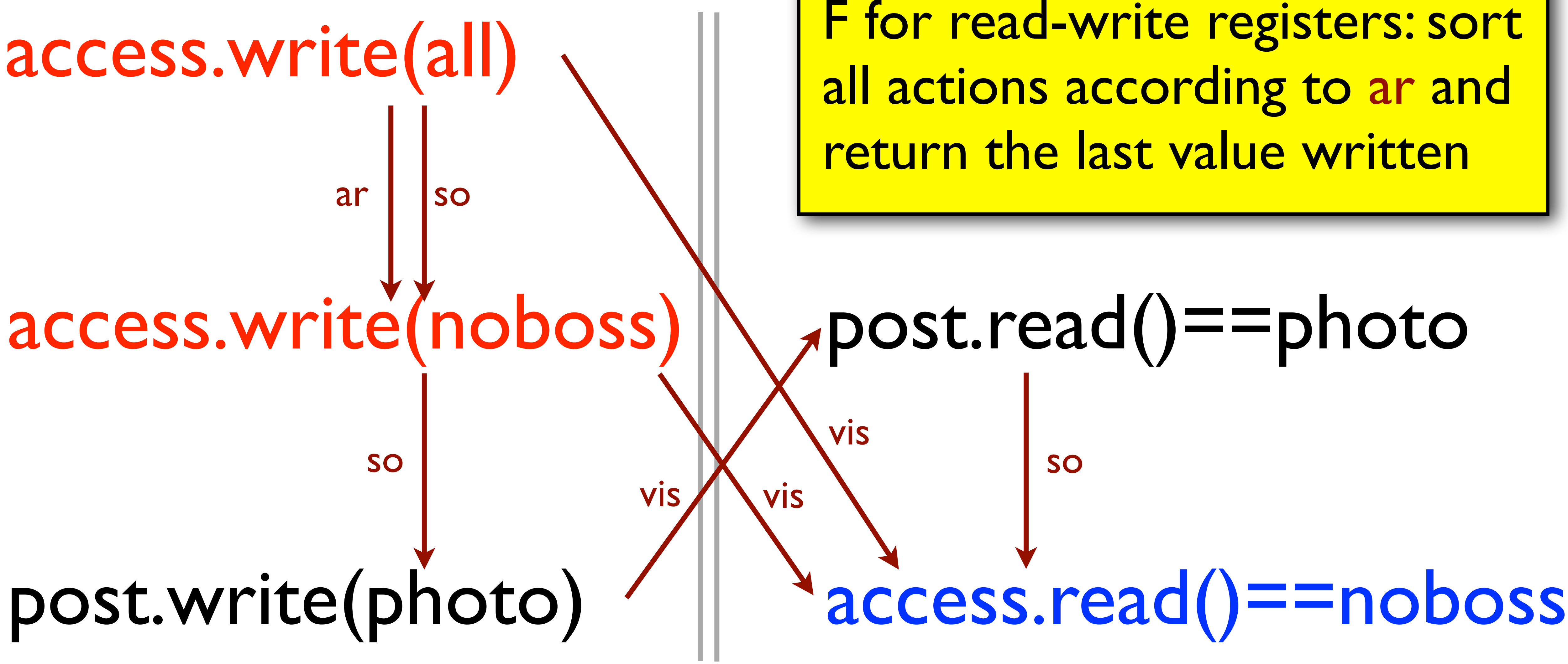


Data type specification

F: cone of influence → return value

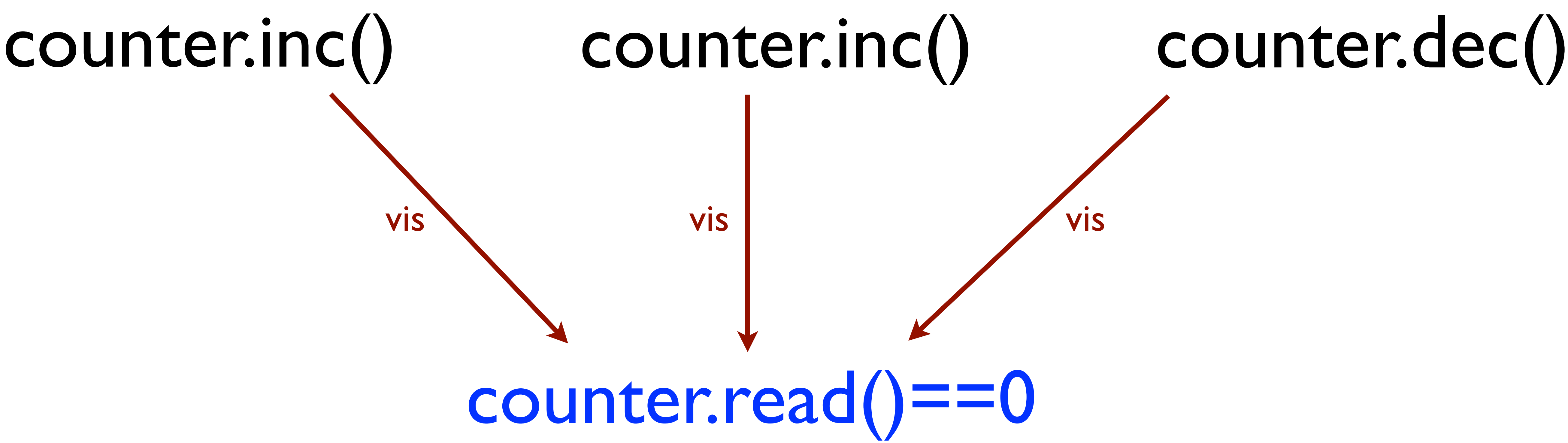
Projection of the execution onto visible actions: (A', vis', ar')

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



Counter data type

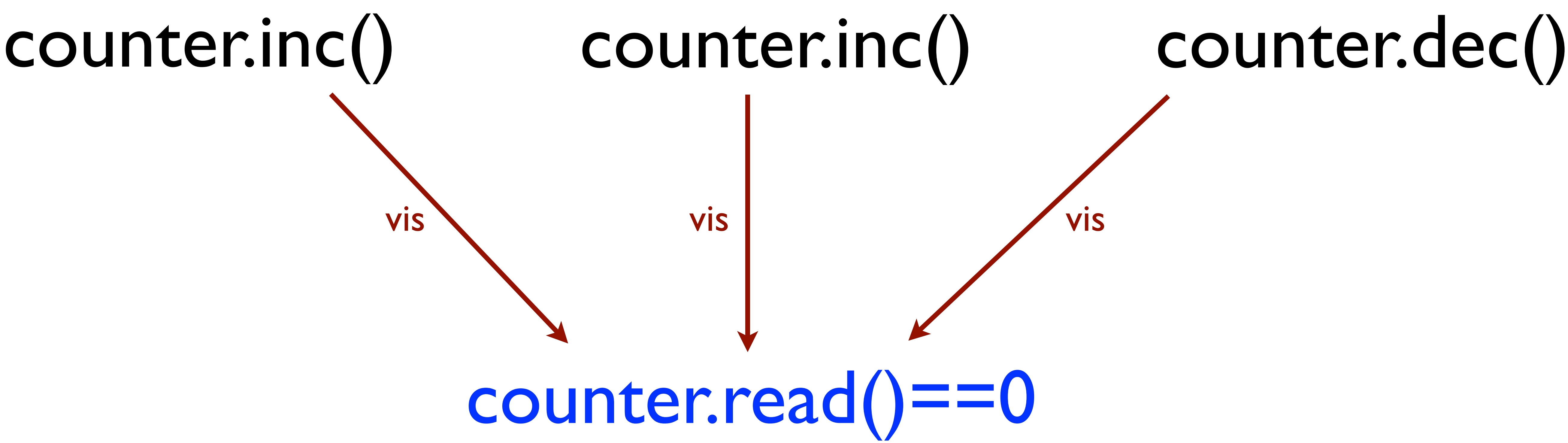
F: cone of influence → return value



Apply standard counter ADT operations in any order, without using ar

Counter data type

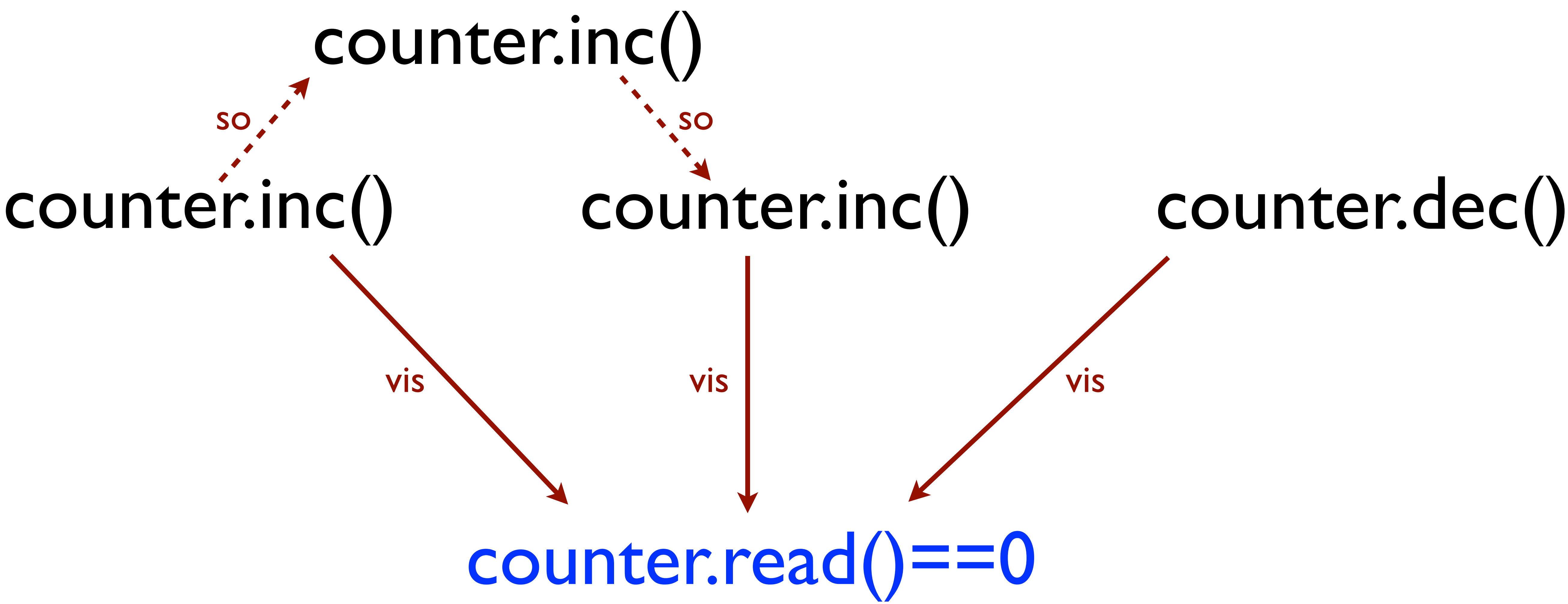
F: cone of influence → return value



Abstracts from internal counter representation: no vector clocks, etc.

Counter data type

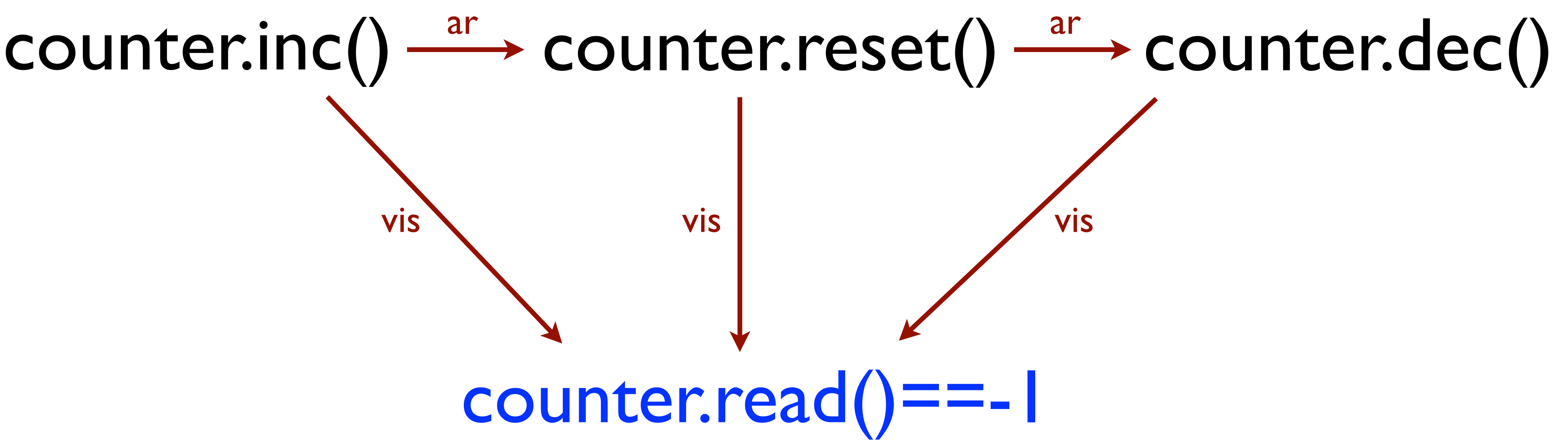
F: cone of influence → return value



What gets taken into account depends only on vis

Counter with a reset

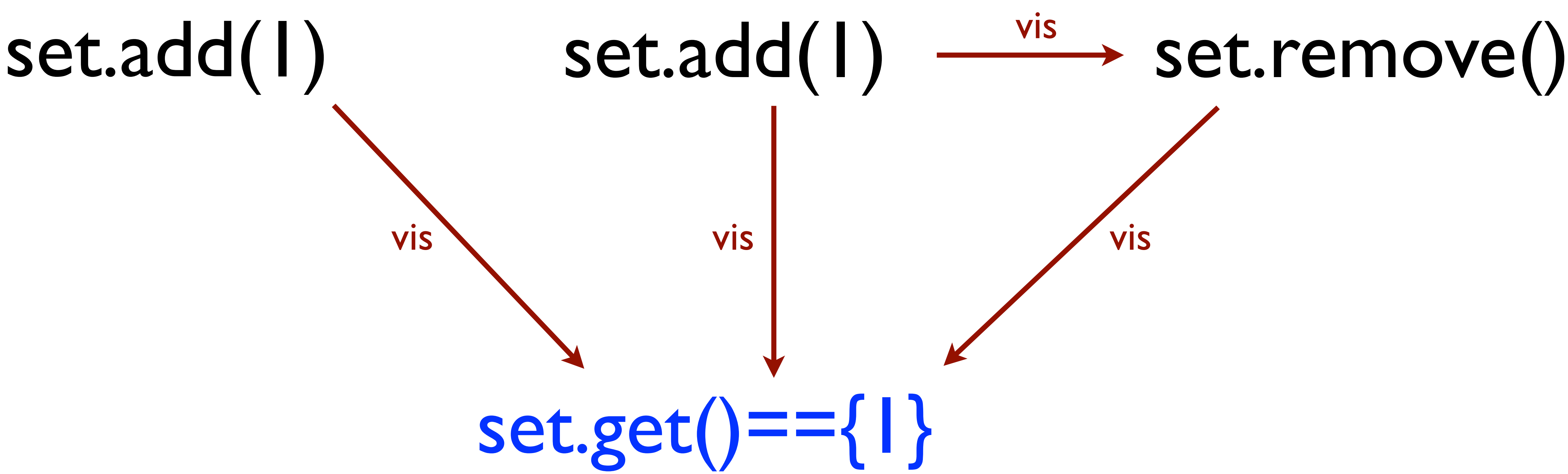
F: cone of influence → return value



Sort by ar, and then apply standard operations

Observed-remove set

F: cone of influence → return value



- F: remove cancels out vis-preceding adds
- OR-set with a reset: defined using ar

Data type specifications

- Agnostic to the internal data type representation: abstract semantics based on relations in the execution

- All (?) of data types in “A comprehensive study of CRDTs”

Axioms: Levels of consistency

Basic eventual consistency [Dynamo]

Session guarantees [Bayou crowd]

Per-object causal consistency [CRDT crowd?]

Causal consistency [COPS, Walter - SOSP'11]

Strong consistency

Axioms: Levels of consistency

Basic eventual consistency [Dynamo]

Session guarantees [Bayou crowd]

Per-object causal consistency [CRDT crowd?]

Causal consistency [COPS, Walter - SOSP'11]

Strong consistency

Axioms ⟺ Operational semantics (in progress). **Riak?**

Axioms: Levels of consistency

Basic eventual consistency [Dynamo]

Session guarantees [Bayou crowd]

Per-object causal consistency [CRDT crowd?]

Causal consistency [COPS, Walter - SOSP'11]

Strong consistency

Axioms: Levels of consistency

Basic eventual consistency [Dynamo]

Session guarantees [Bayou crowd]

Per-object causal consistency [CRDT crowd?]

≈ C/C++ relaxed operations ≈ ARM/Power

Causal consistency [COPS, Walter - SOSP'11]

≈ C/C++ release/acquire operations ≈ ARM/Power

Strong consistency

Specialisation to read-write registers = C/C++ model

Basic eventual consistency

QUERY. Return values computed using data type specifications:
$\forall a \in A.\, \mathsf{rval}(a) = F_{\mathsf{type}(a)}(\mathsf{cone}(a))$

EVENTUAL. An operation cannot be invisible forever:
$\forall a \in A.\, \neg(\exists \text{ infinitely many } b \in A.\, \mathsf{sameobj}(a, b) \wedge \neg(a \xrightarrow{\mathsf{vis}} b))$

THINAIR. No out-of-thin-air values:
$\mathsf{so} \cup \mathsf{vis}$ is acyclic

Basic eventual consistency

QUERY. Return values computed using data type specifications:
$\forall a \in A.\, \mathsf{rval}(a) = F_{\mathsf{type}(a)}(\mathsf{cone}(a))$

EVENTUAL. An operation cannot be invisible forever:
$\forall a \in A.\, \neg(\exists \text{ infinitely many } b \in A.\, \mathsf{sameobj}(a, b) \wedge \neg(a \xrightarrow{\mathsf{vis}} b))$

THINAIR. No out-of-thin-air values:
$\mathsf{so} \cup \mathsf{vis}$ is acyclic

Basic eventual consistency

QUERY. Return values computed using data type specifications:
$\forall a \in A.\, \mathsf{rval}(a) = F_{\mathsf{type}(a)}(\mathsf{cone}(a))$

EVENTUAL. An operation cannot be invisible forever:
$\forall a \in A.\, \neg(\exists \text{ infinitely many } b \in A.\, \mathsf{sameobj}(a, b) \wedge \neg(a \xrightarrow{\mathsf{vis}} b))$

THINAIR. No out-of-thin-air values:
$\mathsf{so} \cup \mathsf{vis}$ is acyclic

Basic eventual consistency

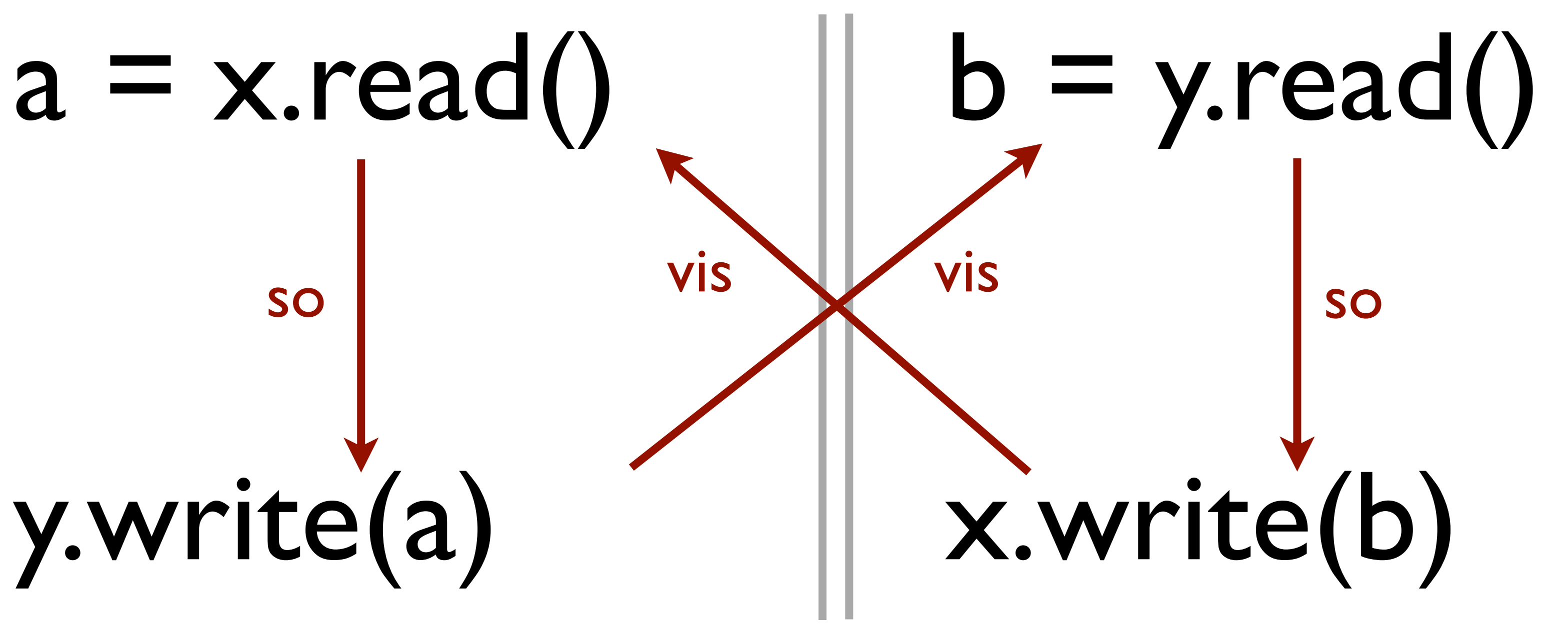
QUERY. Return values computed using data type specifications:
$\forall a \in A.\, \mathsf{rval}(a) = F_{\mathsf{type}(a)}(\mathsf{cone}(a))$

EVENTUAL. An operation cannot be invisible forever:
$\forall a \in A.\, \neg(\exists \text{ infinitely many } b \in A.\, \mathsf{sameobj}(a, b) \wedge \neg(a \xrightarrow{\mathsf{vis}} b))$

THINAIR. No out-of-thin-air values:
$\mathsf{so} \cup \mathsf{vis}$ is acyclic

Out-of-thin-air values

initially x = y = 0



{ a = b = 42 }

In weak memory happens as a result of speculation

Session guarantees [Terry+ 94]

$\mathsf{soo} = \mathsf{so} \cap \mathsf{sameobj}$

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): $\mathsf{soo} \subseteq \mathsf{ar}$

Session guarantees [Terry+ 94]

$\mathsf{soo} = \mathsf{so} \cap \mathsf{sameobj}$

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

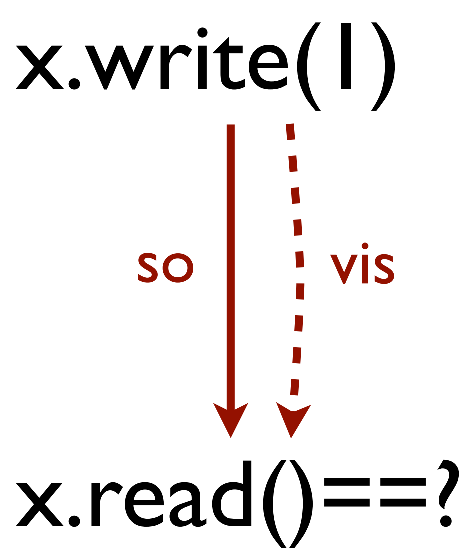
MWA (Monotonic Writes in Arbitration): $\mathsf{soo} \subseteq \mathsf{ar}$

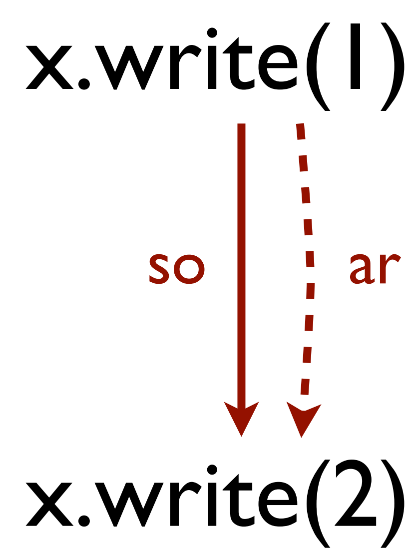
≈ specialise to C++ coherence axioms

Session guarantees [Terry$^+$ 94]

$\mathsf{soo} = \mathsf{so} \cap \mathsf{sameobj}$

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

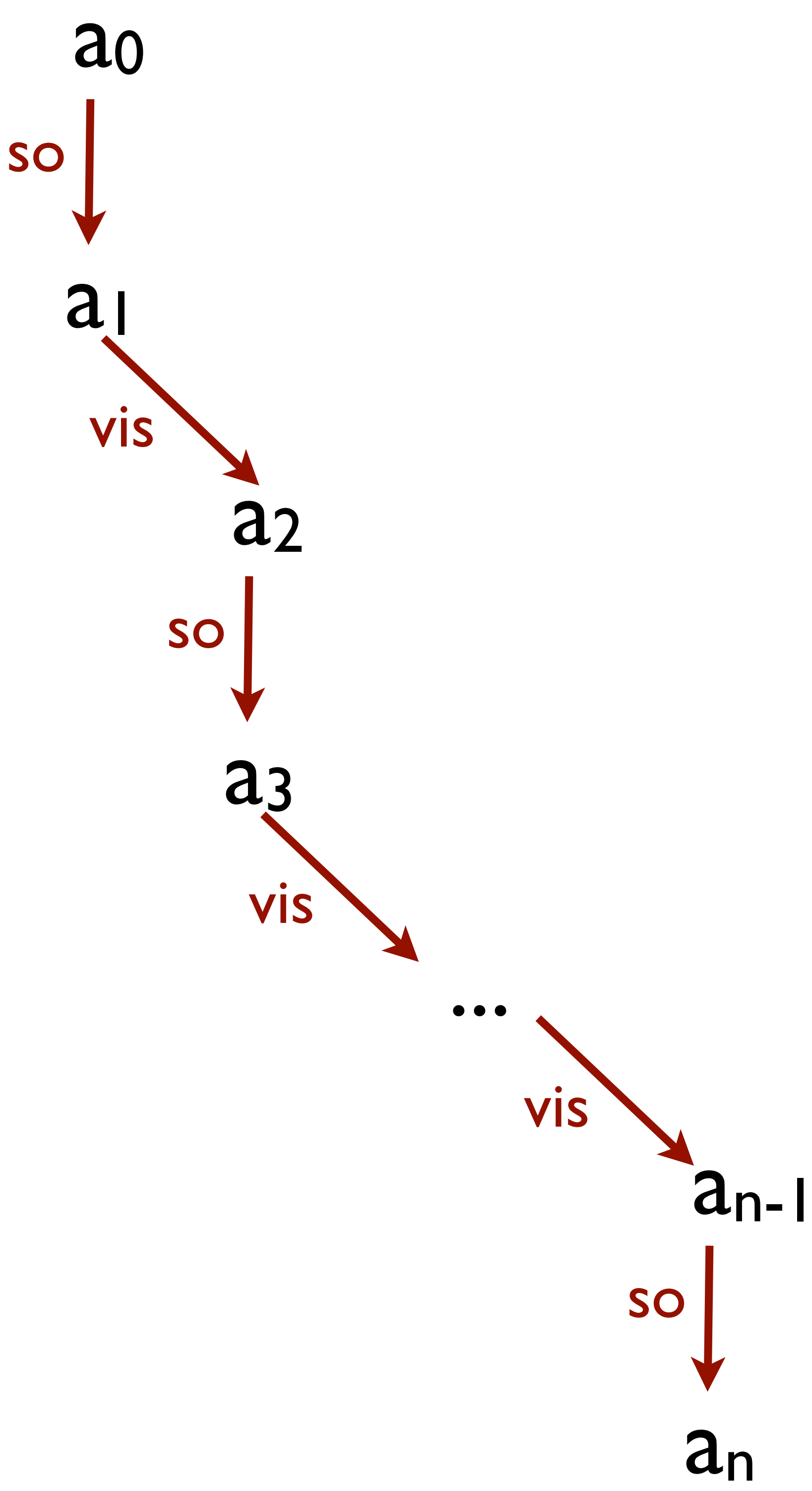




MWA (Monotonic Writes in Arbitration): $\mathsf{soo} \subseteq \mathsf{ar}$

≈ specialise to C++ coherence axioms

Per-object causal consistency



Preserve per-object happens-before:

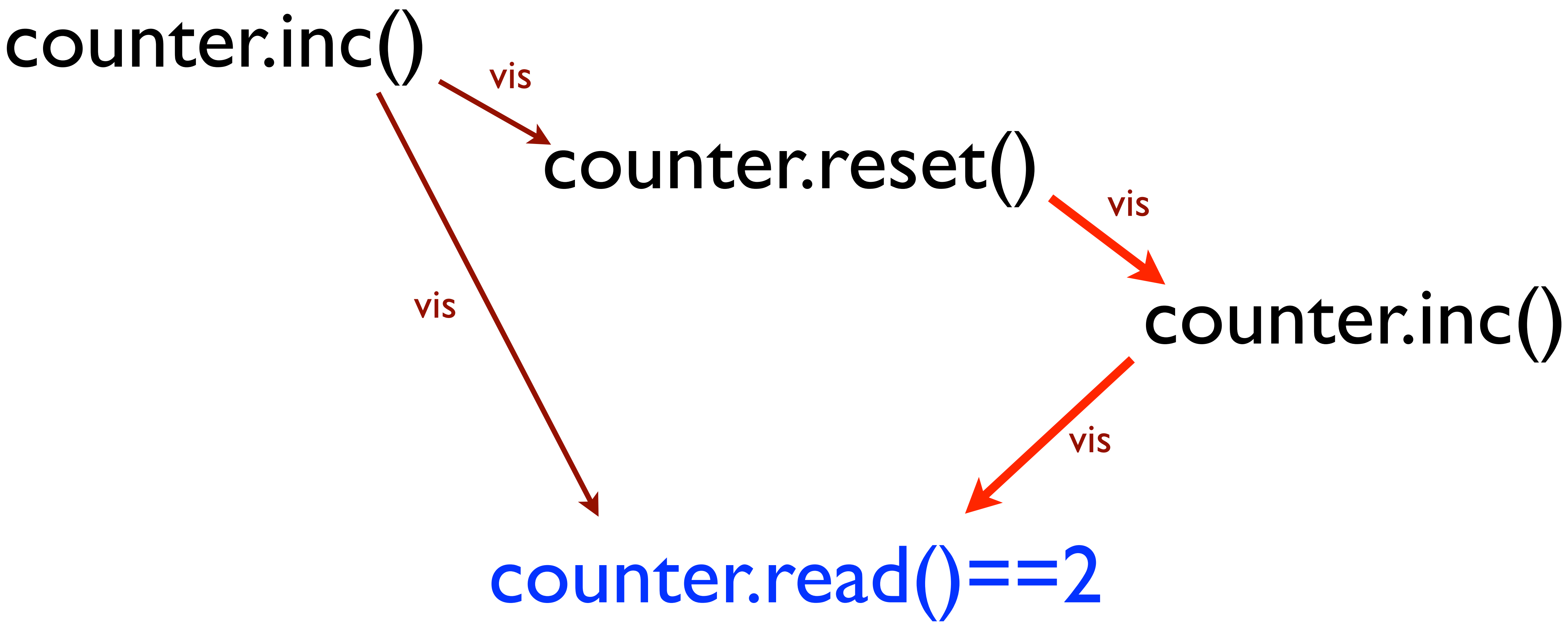
$$hbo = ((so \cap sameobj) \cup vis)^+$$

Per-object causal consistency

Per-object happens-before: $\mathsf{hbo} = ((\mathsf{so} \cap \mathsf{sameobj}) \cup \mathsf{vis})^+$

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

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

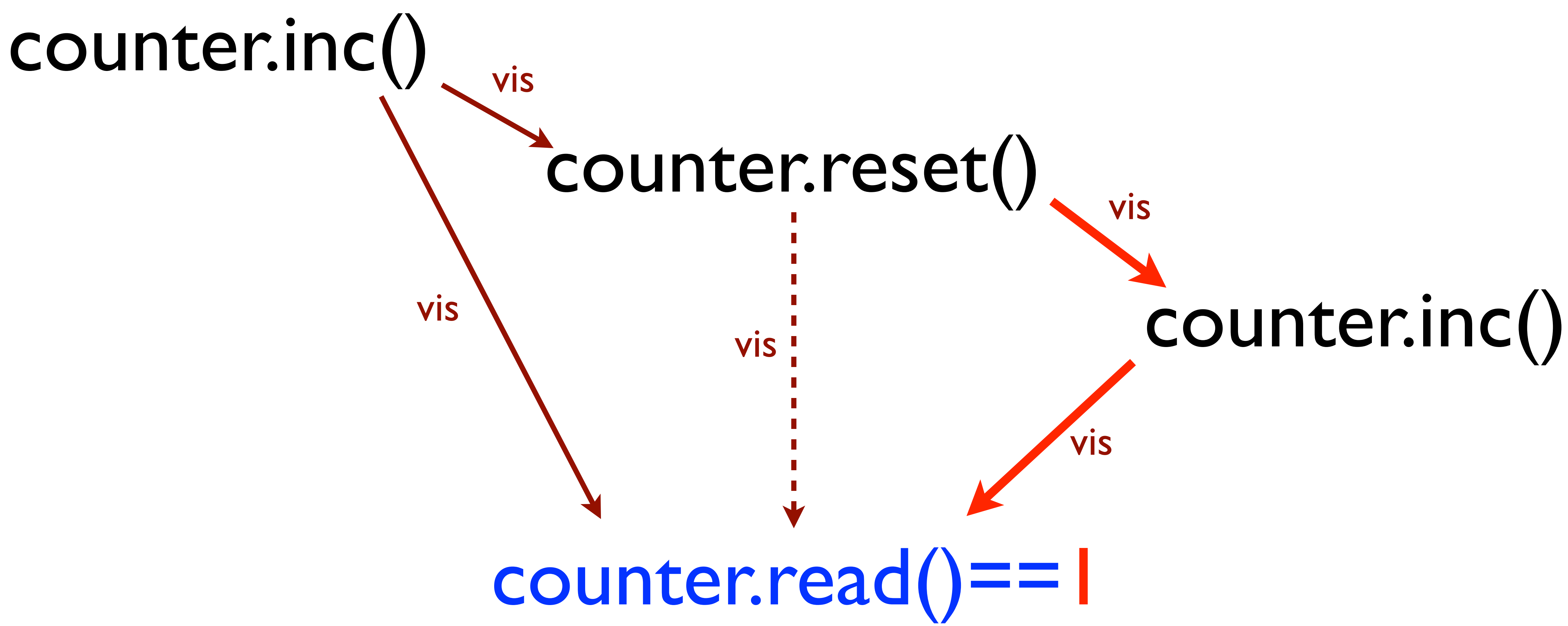


Per-object causal consistency

Per-object happens-before: $\mathsf{hbo} = ((\mathsf{so} \cap \mathsf{sameobj}) \cup \mathsf{vis})^+$

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

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



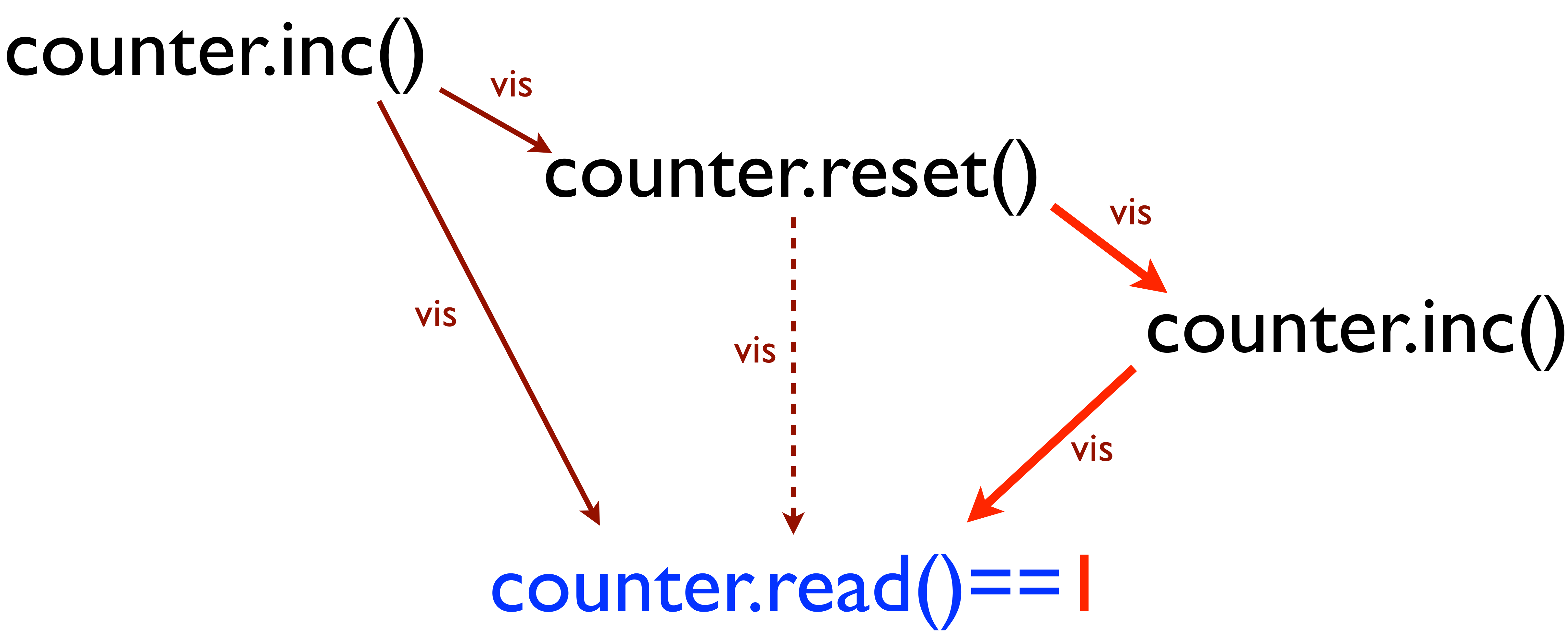
Per-object causal co

All of Terry's session guarantees = per-object causal consistency

Per-object happens-before: $\mathsf{hbo} = ((\mathsf{so} \cap \mathsf{sameobj}) \cup \mathsf{vis})^+$

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

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

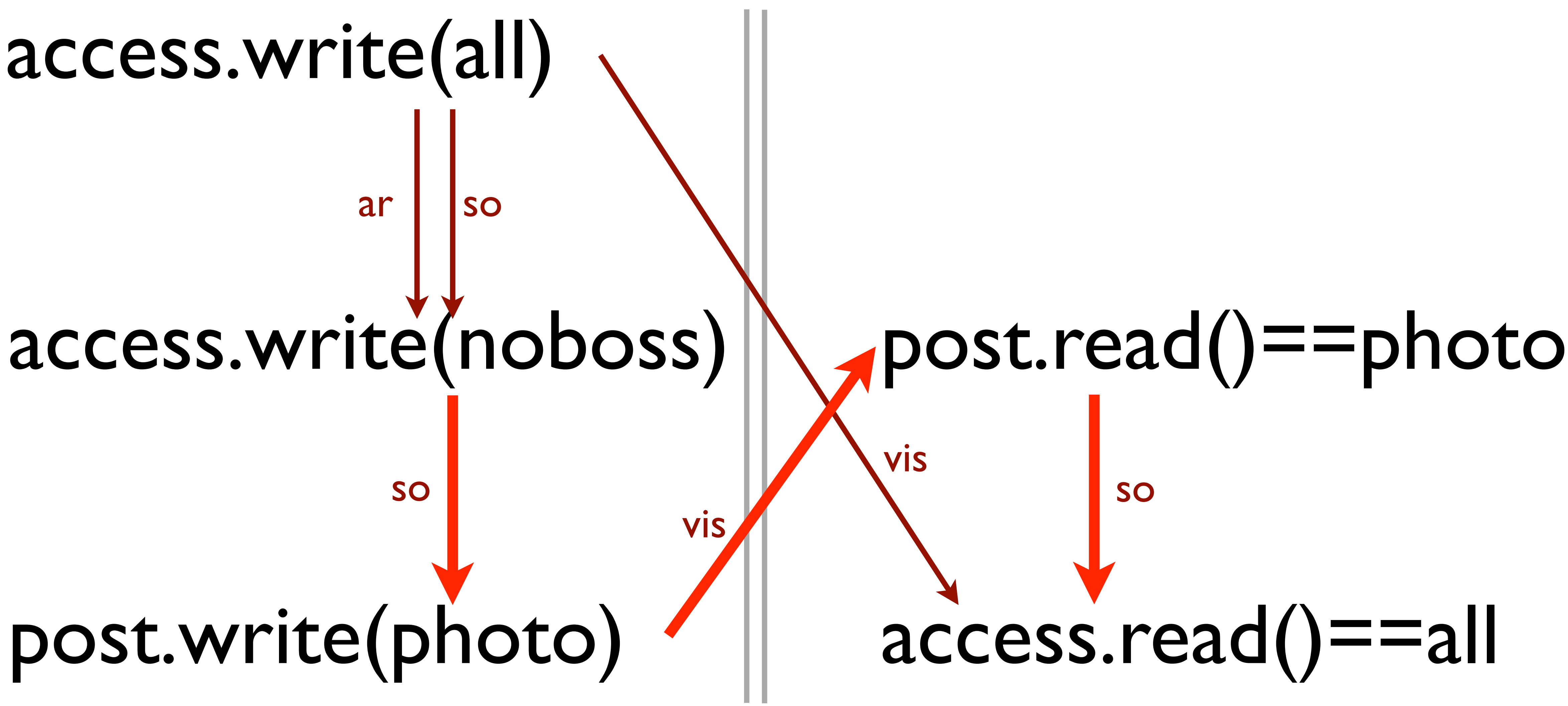


(Cross-object) causal consistency

Happens-before: $\mathsf{hb} = (\mathsf{so} \cup \mathsf{vis})^+$

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

COCA (Cross-Object Causal Arbitration): $\mathsf{hb} \cup \mathsf{ar}$ is acyclic

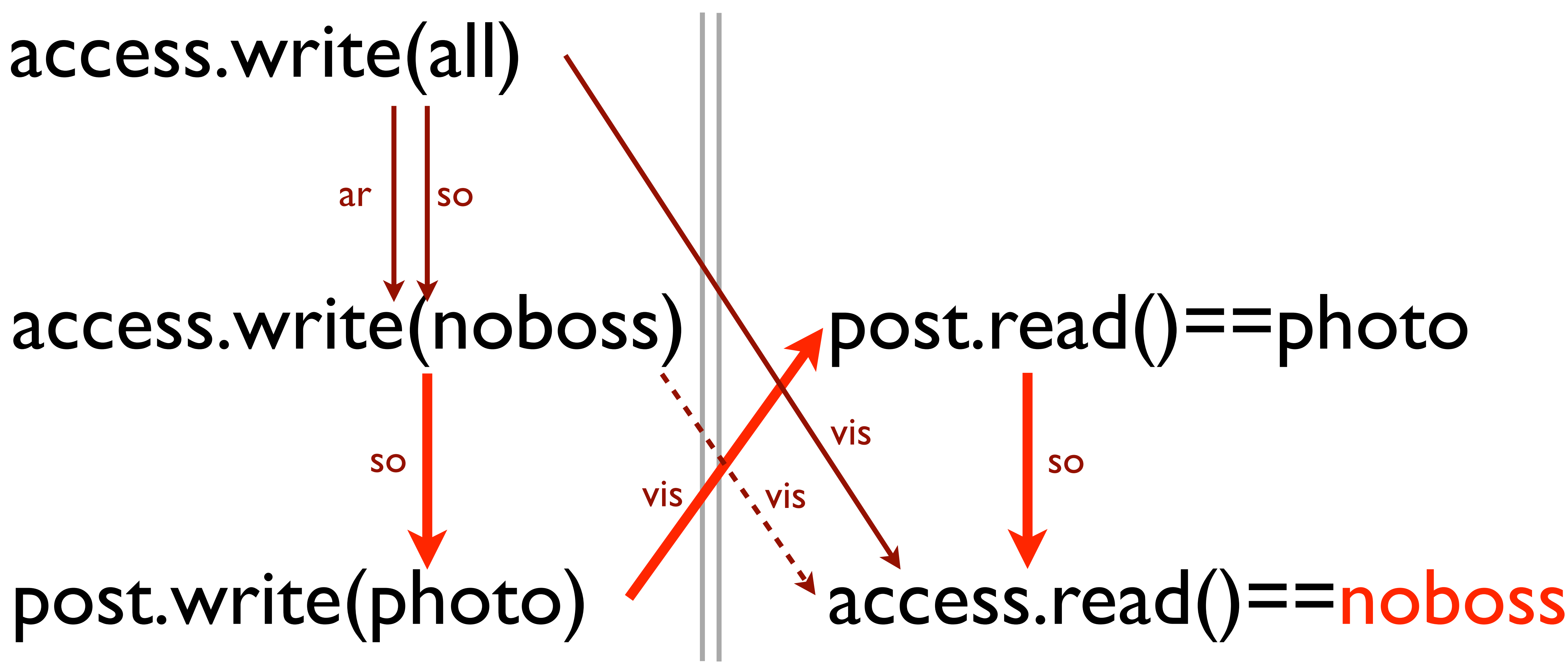


(Cross-object) causal consistency

Happens-before: $\mathsf{hb} = (\mathsf{so} \cup \mathsf{vis})^+$

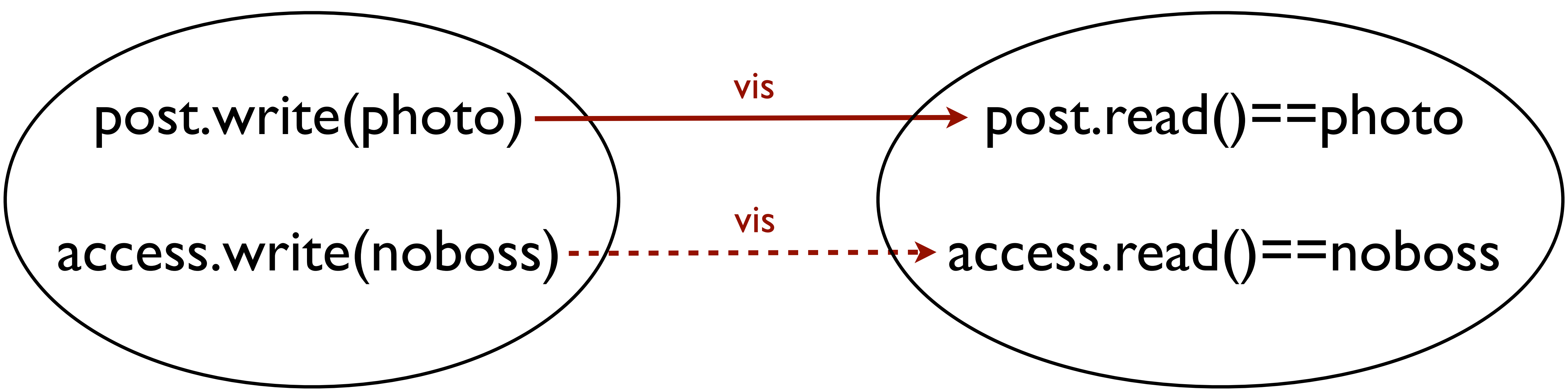
COCV (Cross-Object Causal Visibility): $(\mathsf{hb} \cap \mathsf{sameobj}) \subseteq \mathsf{vis}$

COCA (Cross-Object Causal Arbitration): $\mathsf{hb} \cup \mathsf{ar}$ is acyclic



Transactions

- ~: relates actions in the same transaction
- Main idea: factor key relations over ~
- Similar to snapshot isolation without write/write conflict detection
- For causal consistency equivalent to Parallel Snapshot Isolation (Walter)



The need for combining consistency levels

- Causal consistency is desirable, okay with availability and partition-tolerance, but still expensive [Bailis+, SOCC'12]:

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

  - Track dependencies and wait until they are satisfied
  - Consistency vs latency trade-off
  - In real-world situations, including all of sb and vis makes the number of dependencies prohibitive

- Strong consistency sometimes needed by application semantics

The need for combining consistency levels

- Causal consistency is desirable, okay with availability and partition-tolerance, but still expensive [Bailis⁺, SOCC'12]:

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

  - Track dependencies and wait until they are satisfied
  - Consistency vs latency trade-off
  - In real-world situations, including all of sb and vis makes the number of dependencies prohibitive

- Strong consistency sometimes needed by application semantics

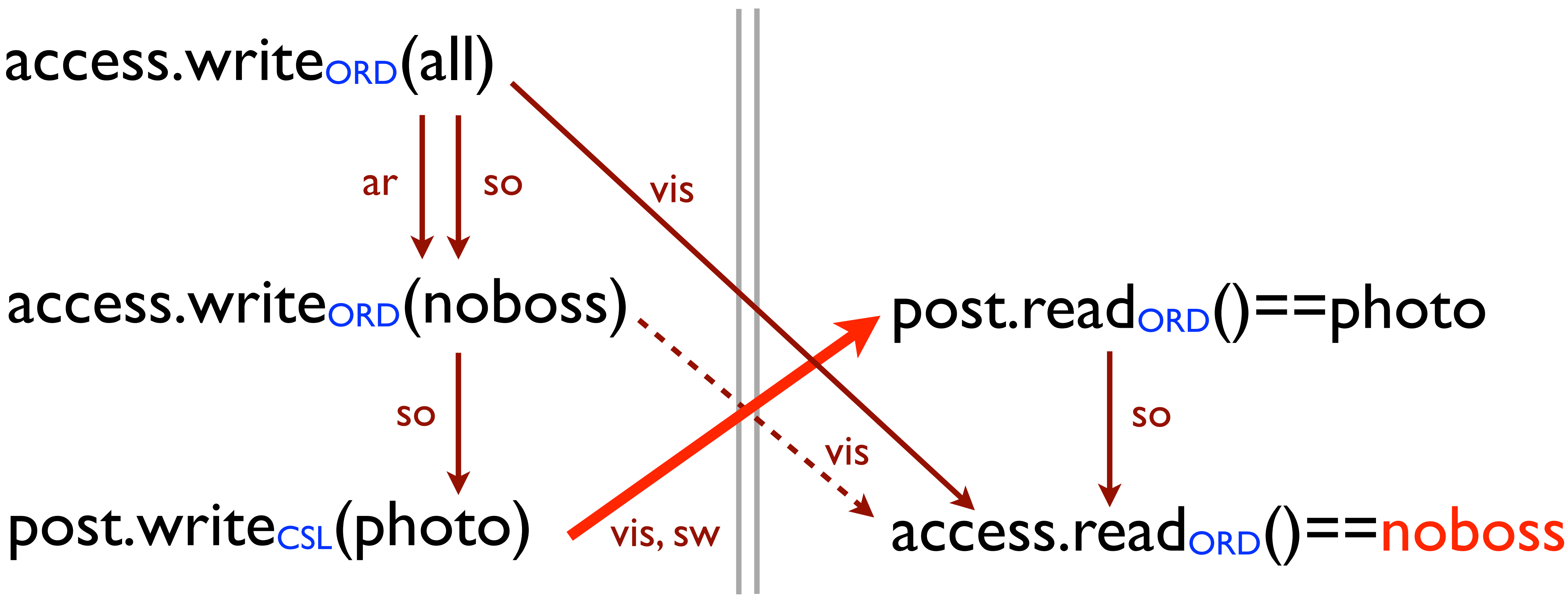
Solution from weak memory models: let the programmer specify which actions need which level consistency

- Assume per-object consistency as default
- Request cross-object consistency using consistency annotations:

$$a \xrightarrow{\mathsf{sw}} b \iff a \xrightarrow{\mathsf{vis}} b \land \mathsf{level}(a) = \mathsf{CSL}$$

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

- Selects vis edges that should be causal:



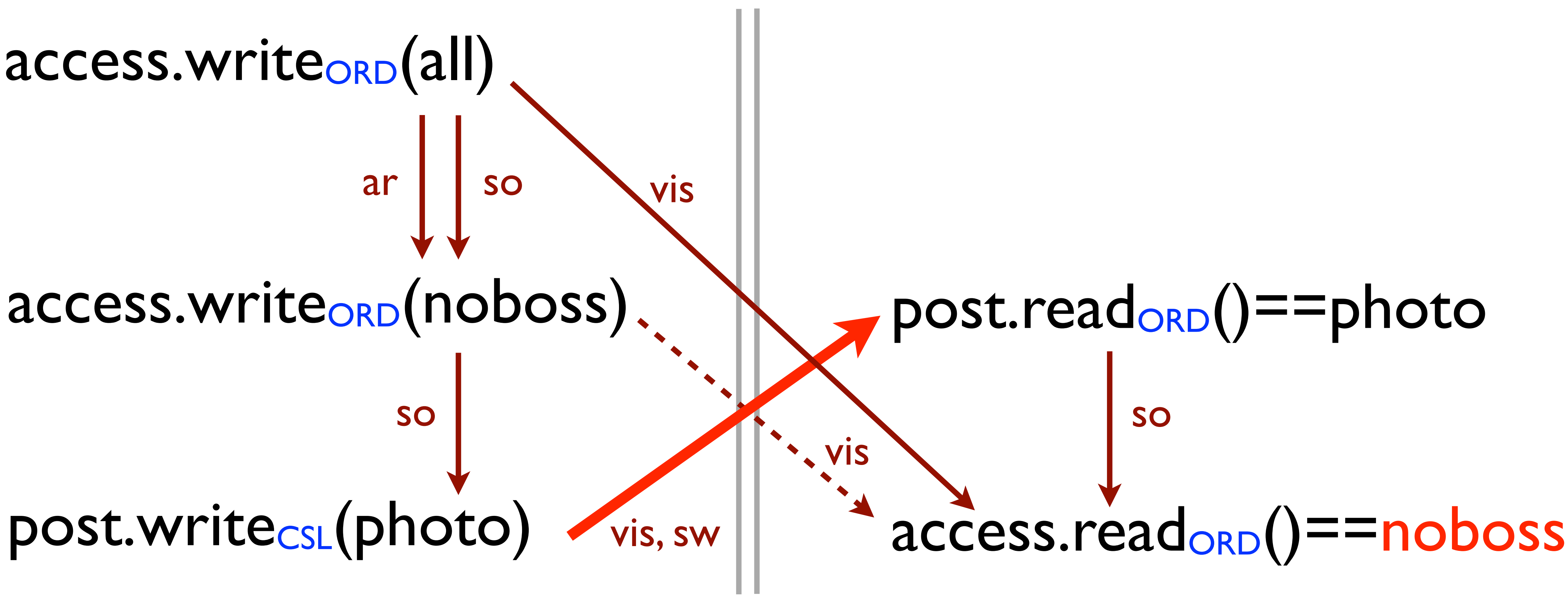
Strong consistency added similarly

- Assume per-object consistency as defau
- Request cross-object consistency using consistency annotations:

$$a \xrightarrow{\mathsf{sw}} b \iff a \xrightarrow{\mathsf{vis}} b \wedge \mathsf{level}(a) = \mathsf{CSL}$$

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

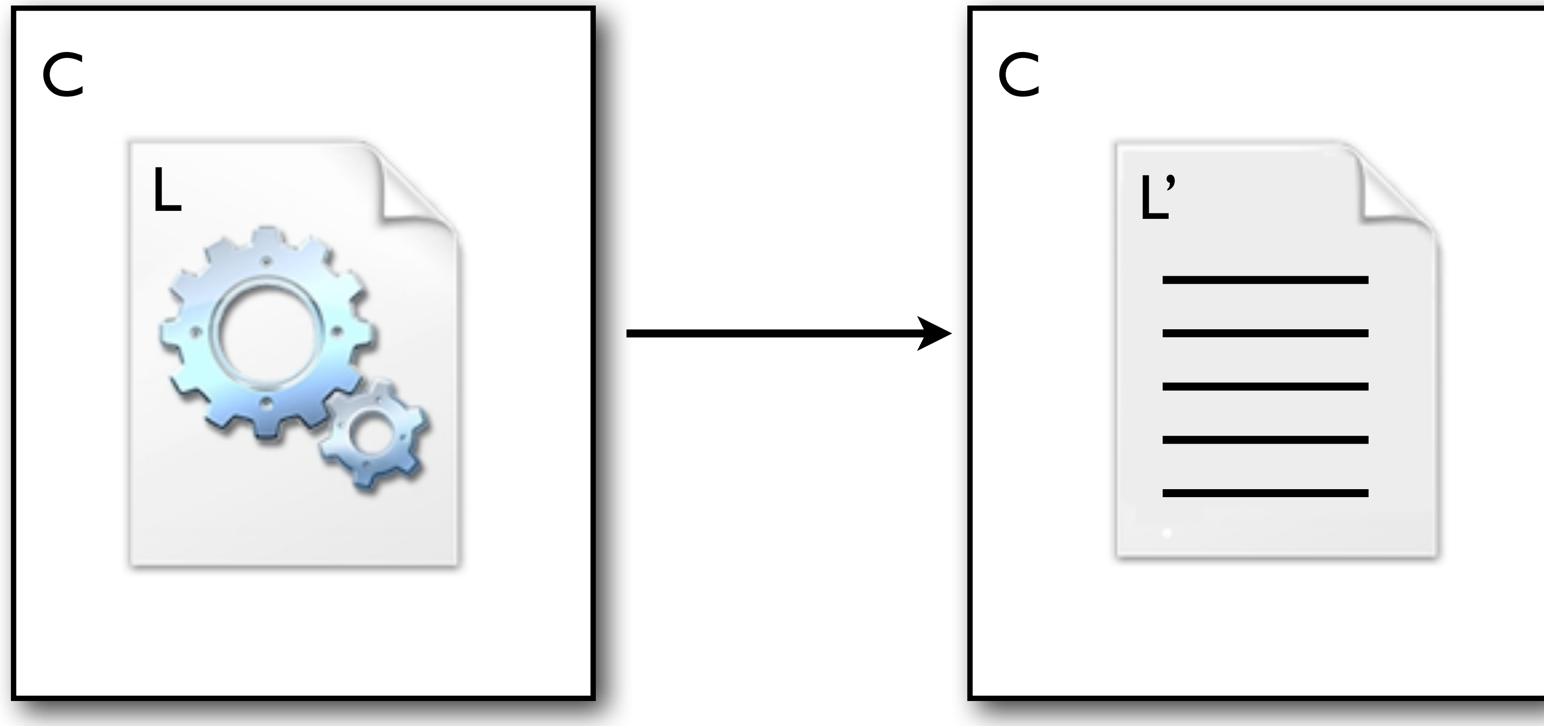
- Selects vis edges that should be causal:



Formulating combinations is tricky

- Choosing axioms: depends on how the implementation works
- Choosing a mechanism for specifying consistency:
  - Operation annotations vs fences
    (fences affect multiple operations)
  - The choice affects the implementation
- C/C++ model offers some guidance
- Formal specification good for exploring the design space and evaluating programmability

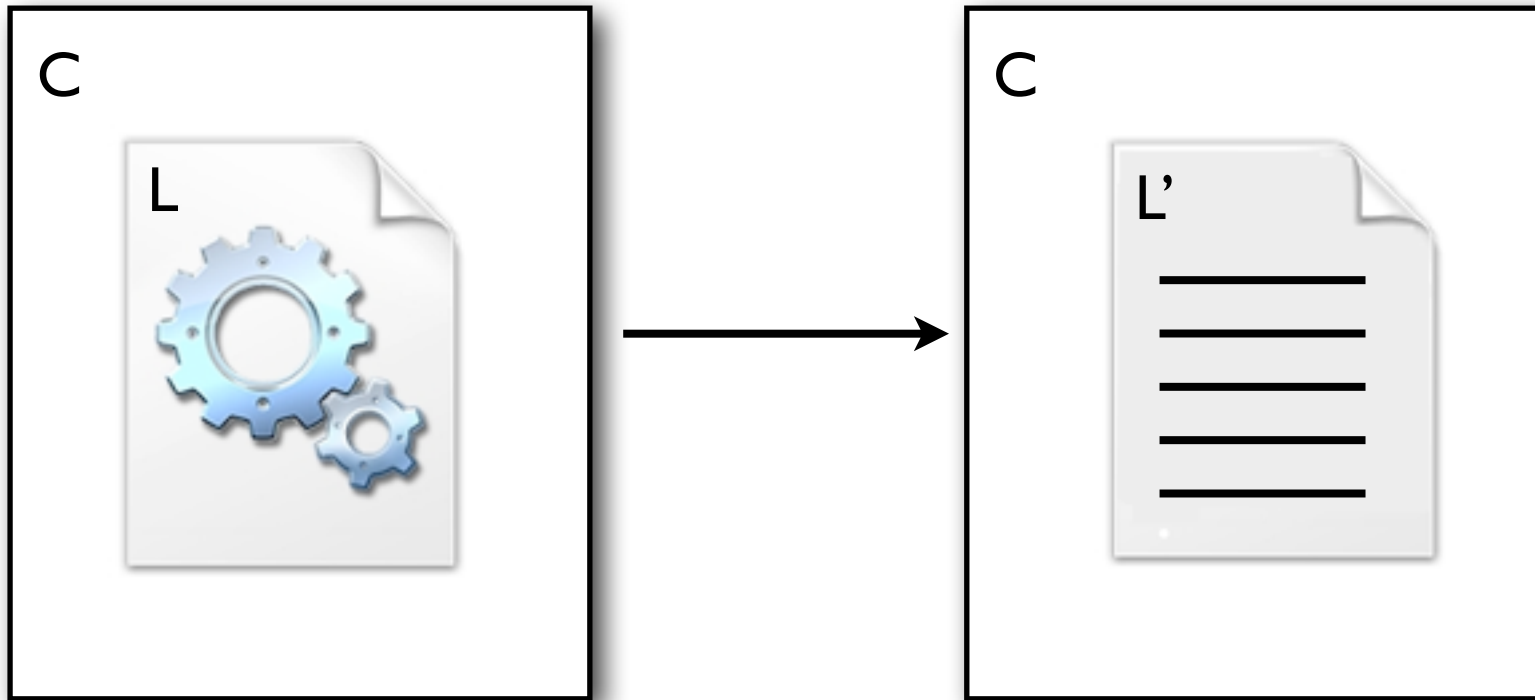
I have a dream...



- Can we reason about eventually consistent systems compositionally?
- Example: a cloud storage system on top of a key-value store
- Package a library as a built-in data type

I have a dream...

A set of methods with a dedicated set of objects in the database:
... m() ...



Data type specification F

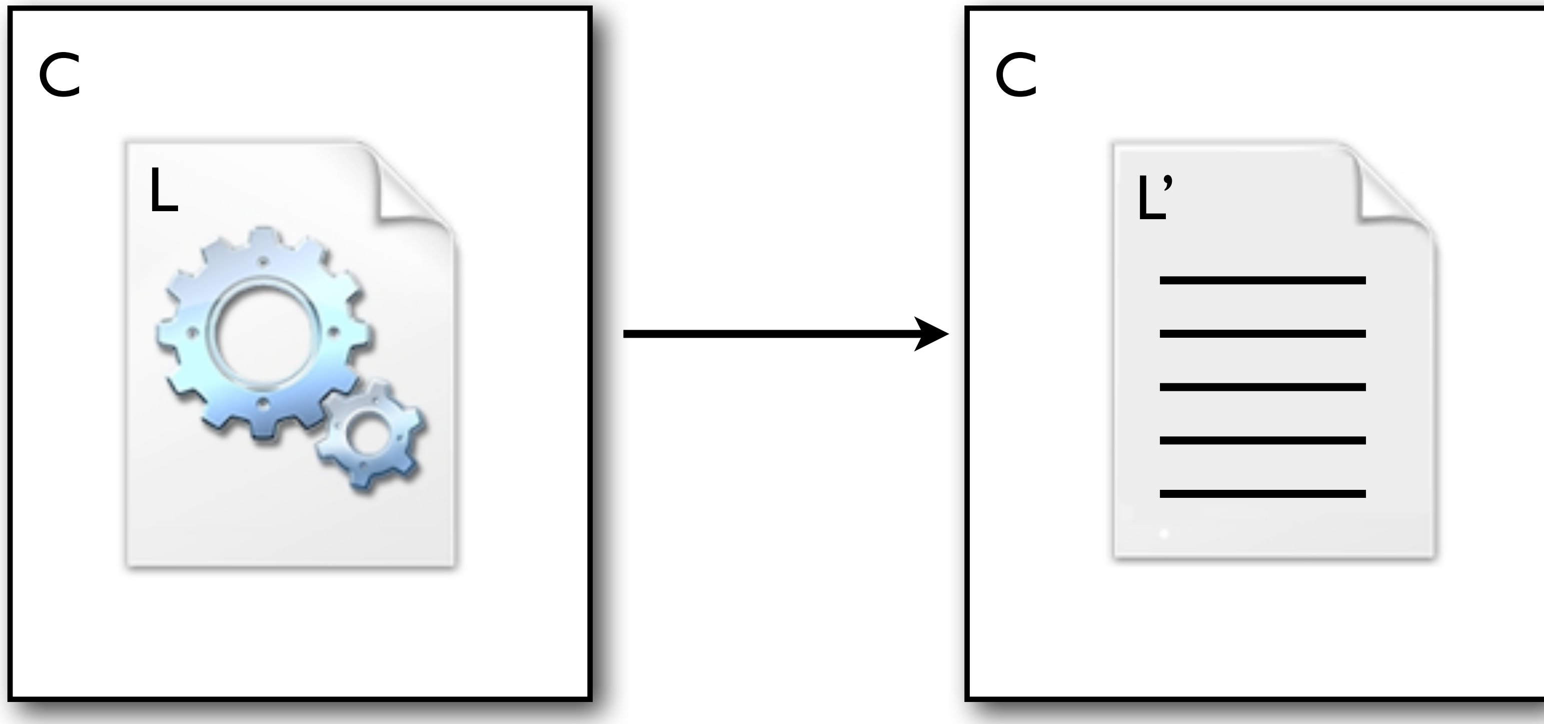
Abstraction Theorem:

$$L \sqsubseteq F \Rightarrow \text{client}([\![C(L)]\!]) \subseteq \text{client}([\![C(F)]\!])$$

Corollary: $C(F) \models P \Rightarrow C(L) \models P$

I have a dream...

A set of methods with a dedicated set of objects in the database:
... m() ...



Data type specification F

Abstraction Theorem:

$$L \sqsubseteq F \Rightarrow \text{client}([\![C(L)]\!]) \subseteq \text{client}([\![C(F)]\!])$$

Corollary: $C(F) \models P \Rightarrow C(L) \models P$

Current solution: rip-off of C/C++ library correctness [POPL'13]

Comparing libraries

- Take the most general client:

$$\mathop{\Big\|}_{k=1}^{n} \left(\begin{array}{ll} \texttt{while (true) \{} & \\ \quad \texttt{if (nondet())} & m_1\texttt{(nondet());} \\ \quad \texttt{else if (nondet())} & m_2\texttt{(nondet());} \\ \quad \texttt{...} & \\ \quad \texttt{else} & m_l\texttt{(nondet());} \\ \texttt{\}} & \end{array} \right)$$

- Get all possible library histories $[\![L]\!]$: describe library behaviour relevant to the client
- $L \sqsubseteq F \iff \forall H \in [\![L]\!]. \exists H' \in [\![F]\!]. H \sqsubseteq H'$

Comparing libraries

- Take the most general client:

Any number of sessions

$$\bigg\|_{k=1}^{n} \left(\begin{array}{ll} \texttt{while (true) \{} & \\ \quad \texttt{if (nondet())} & m_1\texttt{(nondet());} \\ \quad \texttt{else if (nondet())} & m_2\texttt{(nondet());} \\ \quad \texttt{...} & \\ \quad \texttt{else} & m_l\texttt{(nondet());} \\ \texttt{\}} & \end{array} \right)$$

- Get all possible library histories $[\![L]\!]$: describe library behaviour relevant to the client
- $L \sqsubseteq F \iff \forall H \in [\![L]\!]. \exists H' \in [\![F]\!]. H \sqsubseteq H'$

Comparing libraries

- Take the most general client:

Any number of sessions

Any methods, in any order, with any parameters

$$\big\|_{k=1}^{n} \left(\begin{array}{ll} \texttt{while (true) \{} & \\ \quad \texttt{if (nondet())} & m_1\texttt{(nondet());} \\ \quad \texttt{else if (nondet())} & m_2\texttt{(nondet());} \\ \quad \texttt{...} & \\ \quad \texttt{else} & m_l\texttt{(nondet());} \\ \texttt{\}} & \end{array} \right)$$

- Get all possible library histories $[\![L]\!]$: describe library behaviour relevant to the client

- $L \sqsubseteq F \iff \forall H \in [\![L]\!].\, \exists H' \in [\![F]\!].\, H \sqsubseteq H'$

Comparing libraries

- Take the most general client:

Any number of sessions

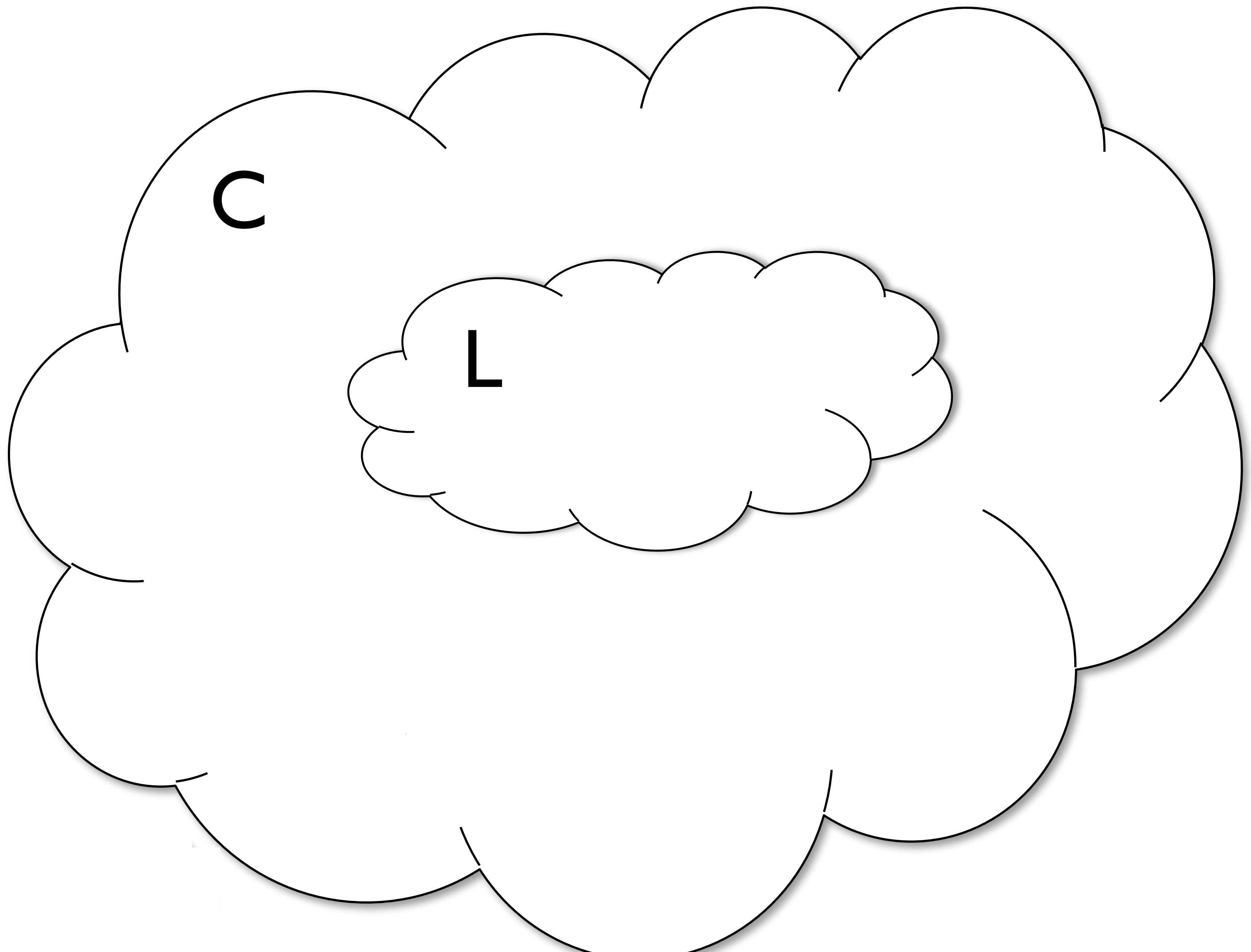
Any methods, in any order, with any parameters

$$\mathop{\Big\|}_{k=1}^{n} \left(\begin{array}{ll} \texttt{while (true) \{} & \\ \quad \texttt{if (nondet())} & m_1\texttt{(nondet());} \\ \quad \texttt{else if (nondet())} & m_2\texttt{(nondet());} \\ \quad \texttt{...} & \\ \quad \texttt{else} & m_l\texttt{(nondet());} \\ \texttt{\}} & \end{array} \right)$$

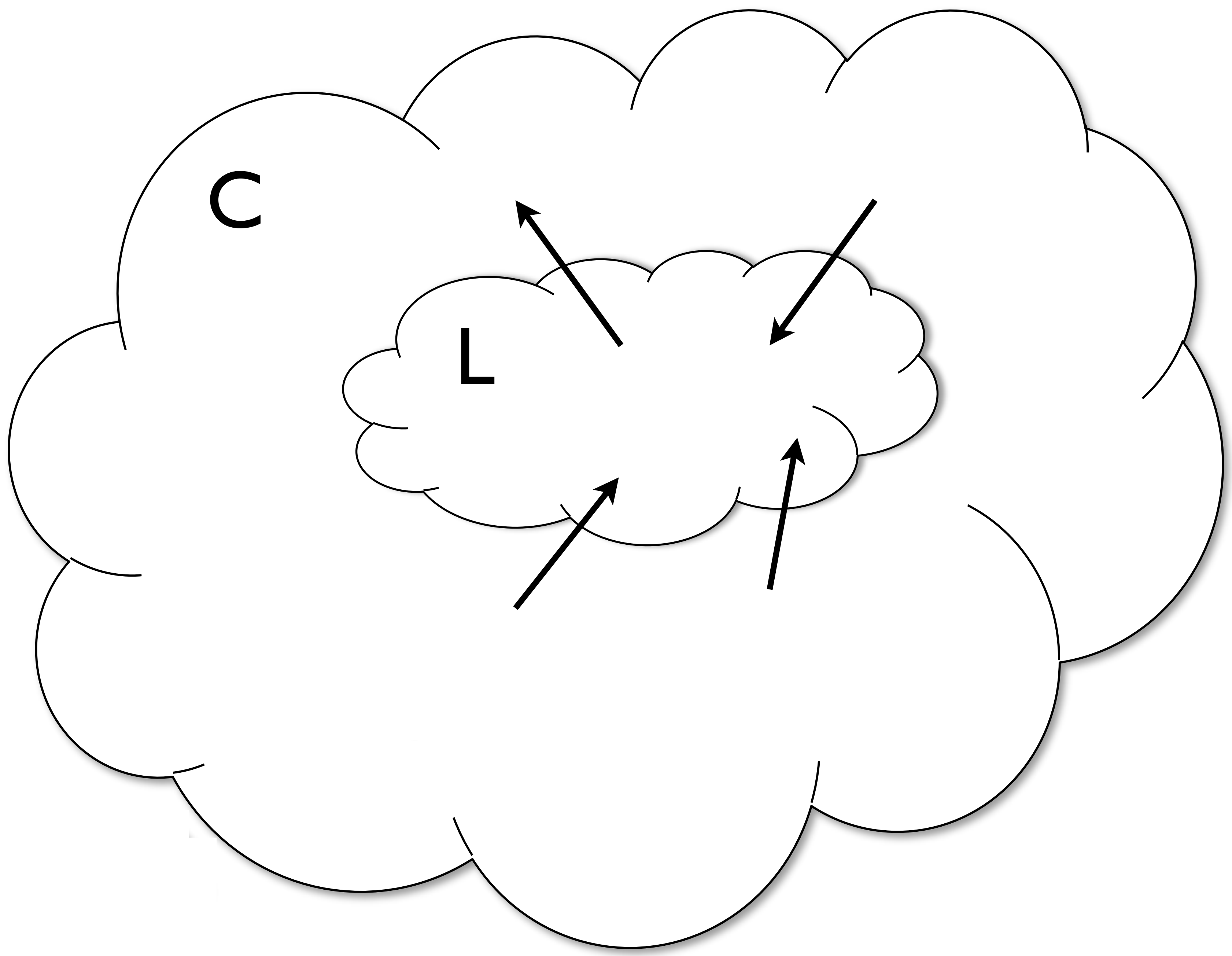
- Get all possible library histories $[\![L]\!]$: describe library behaviour relevant to the client

- $L \sqsubseteq F \iff \forall H \in [\![L]\!]. \exists H' \in [\![F]\!]. H \sqsubseteq H'$

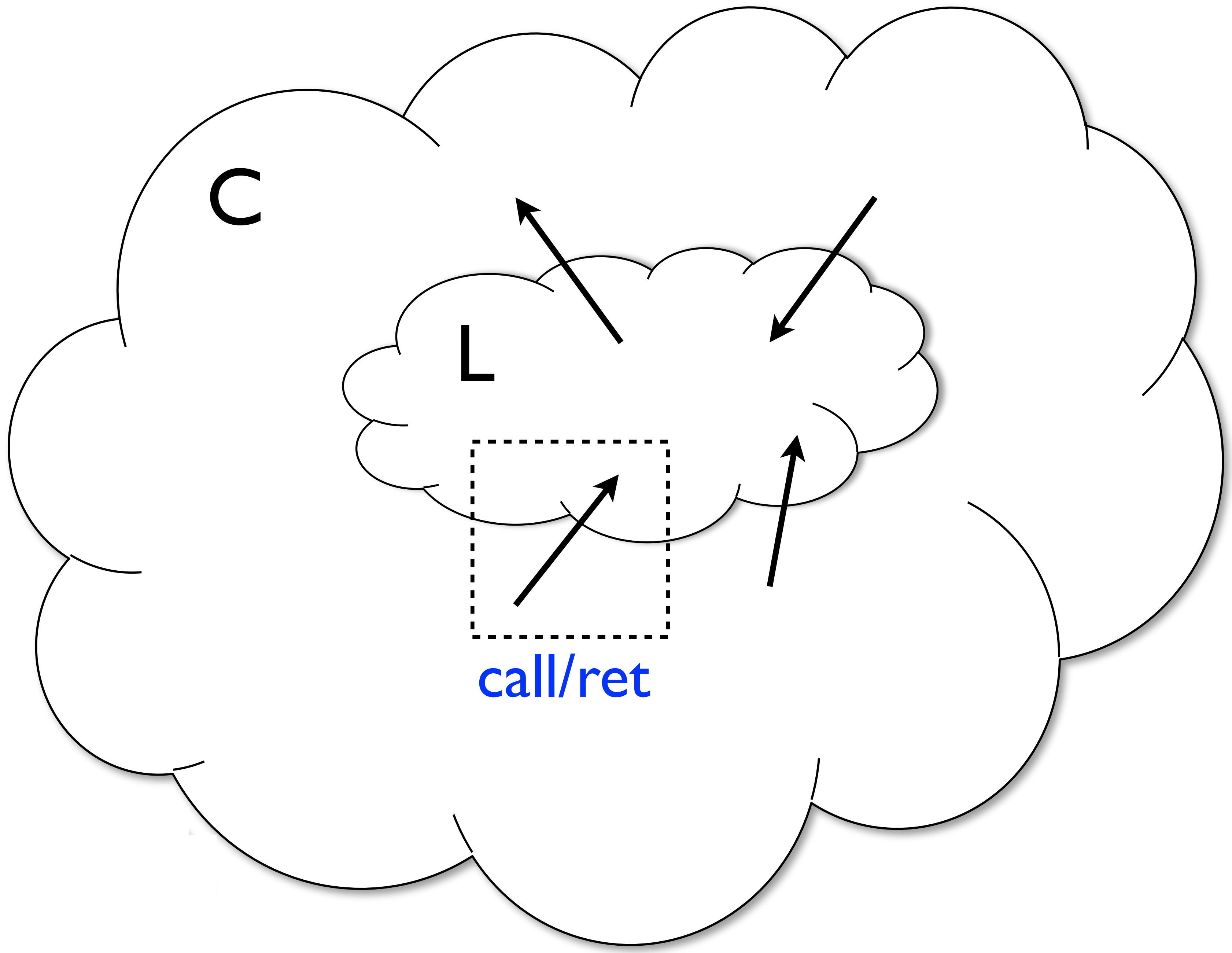
Subgraph replacement



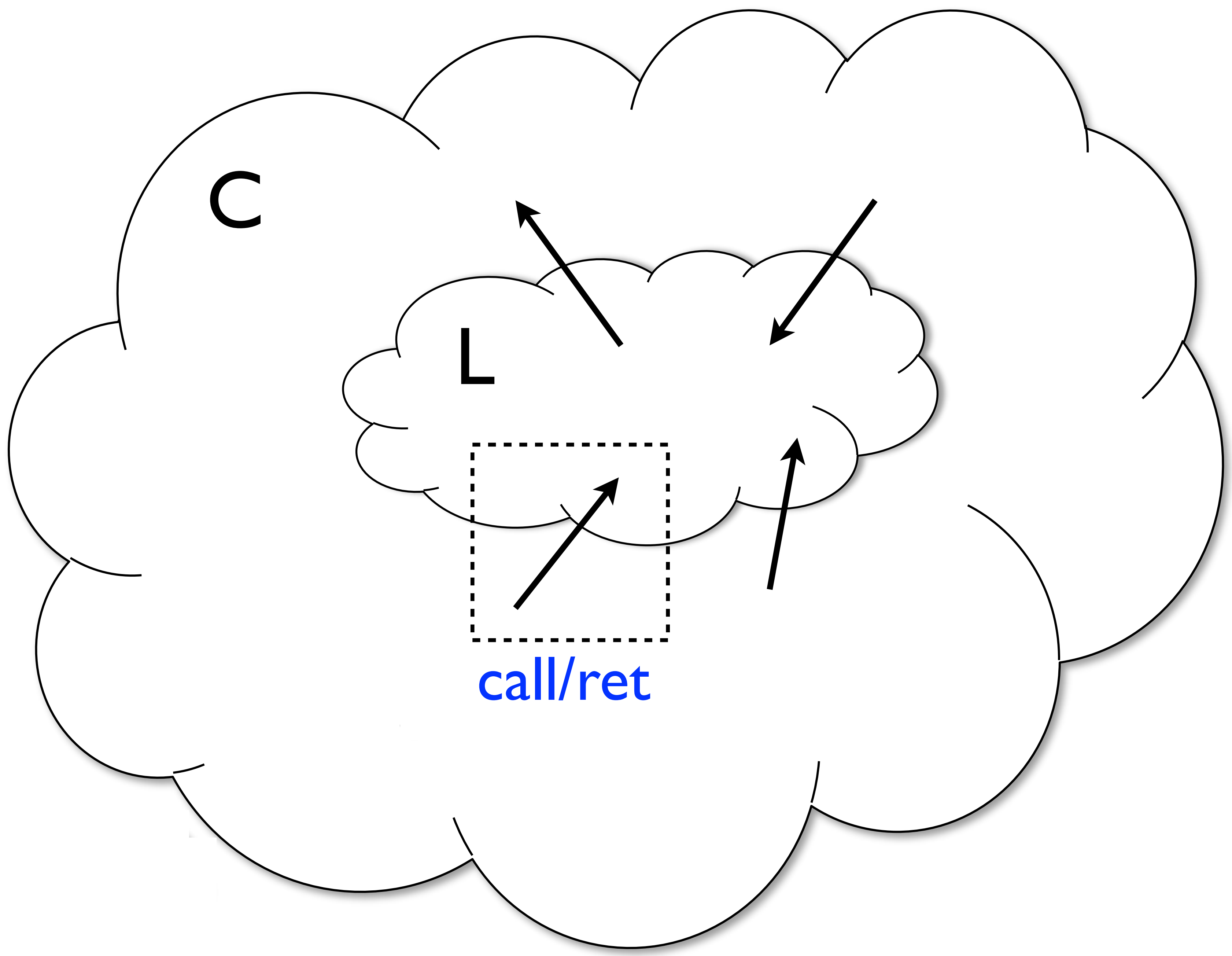
Subgraph replacement



Subgraph replacement



Subgraph replacement



History: several relations on call/return actions

A history component

Projection of hb to calls and returns:

call produce	‖	call consume;
...		...
ready.write$_{CSL}$(1)		ready.read$_{ORD}$()==1;
...		...
return produce		return consume;

A history component

Projection of hb to calls and returns:

call produce
...
ready.write$_{\mathrm{CSL}}$(1)
...
return produce

vis

call consume;
...
ready.read$_{\mathrm{ORD}}$()==1;
...
return consume;

A history component

Projection of hb to calls and returns:

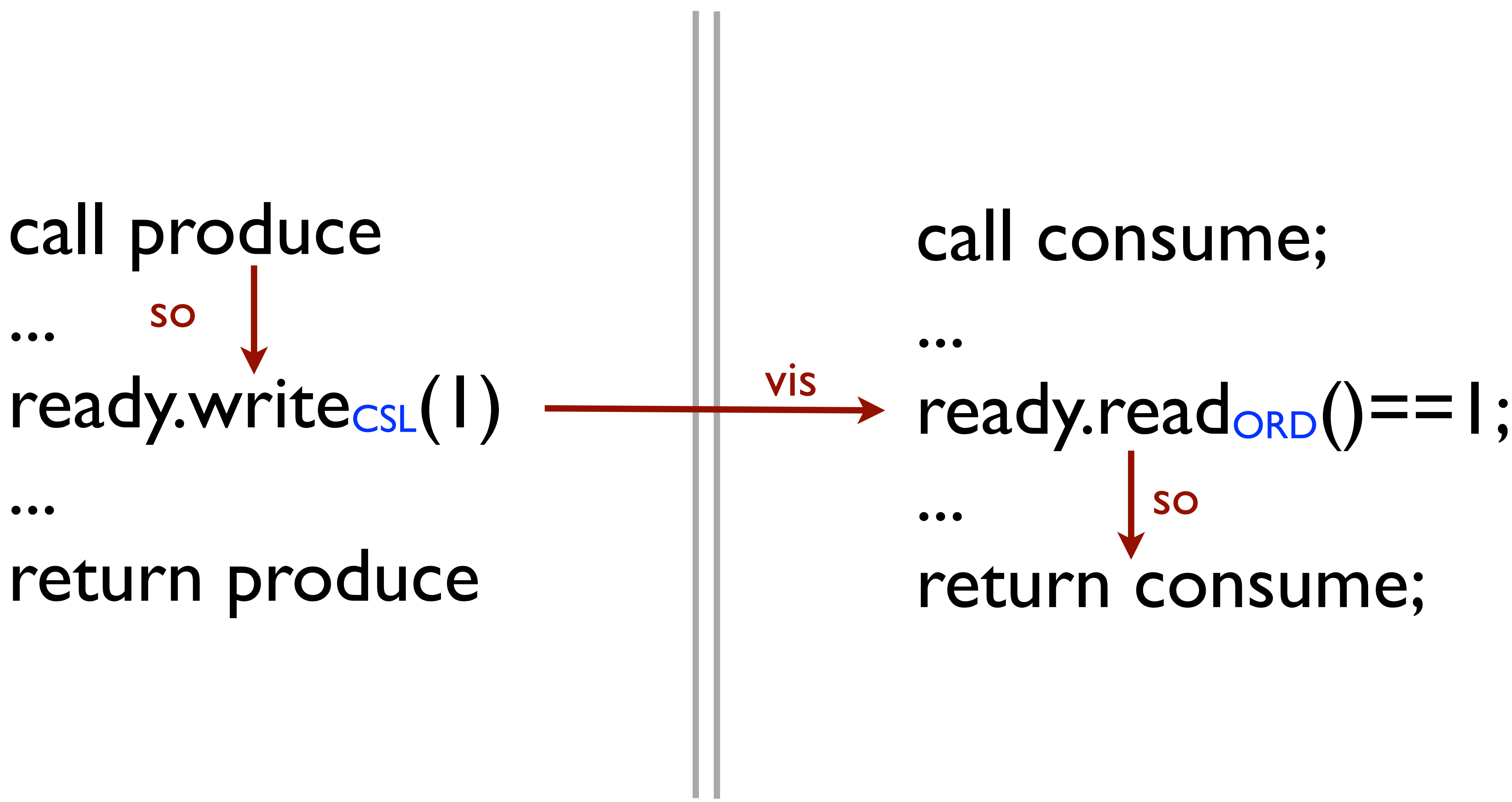
call produce
... so
ready.write$_{CSL}$(1)
...
return produce

vis

call consume;
...
ready.read$_{ORD}$()==1;
...
return consume;

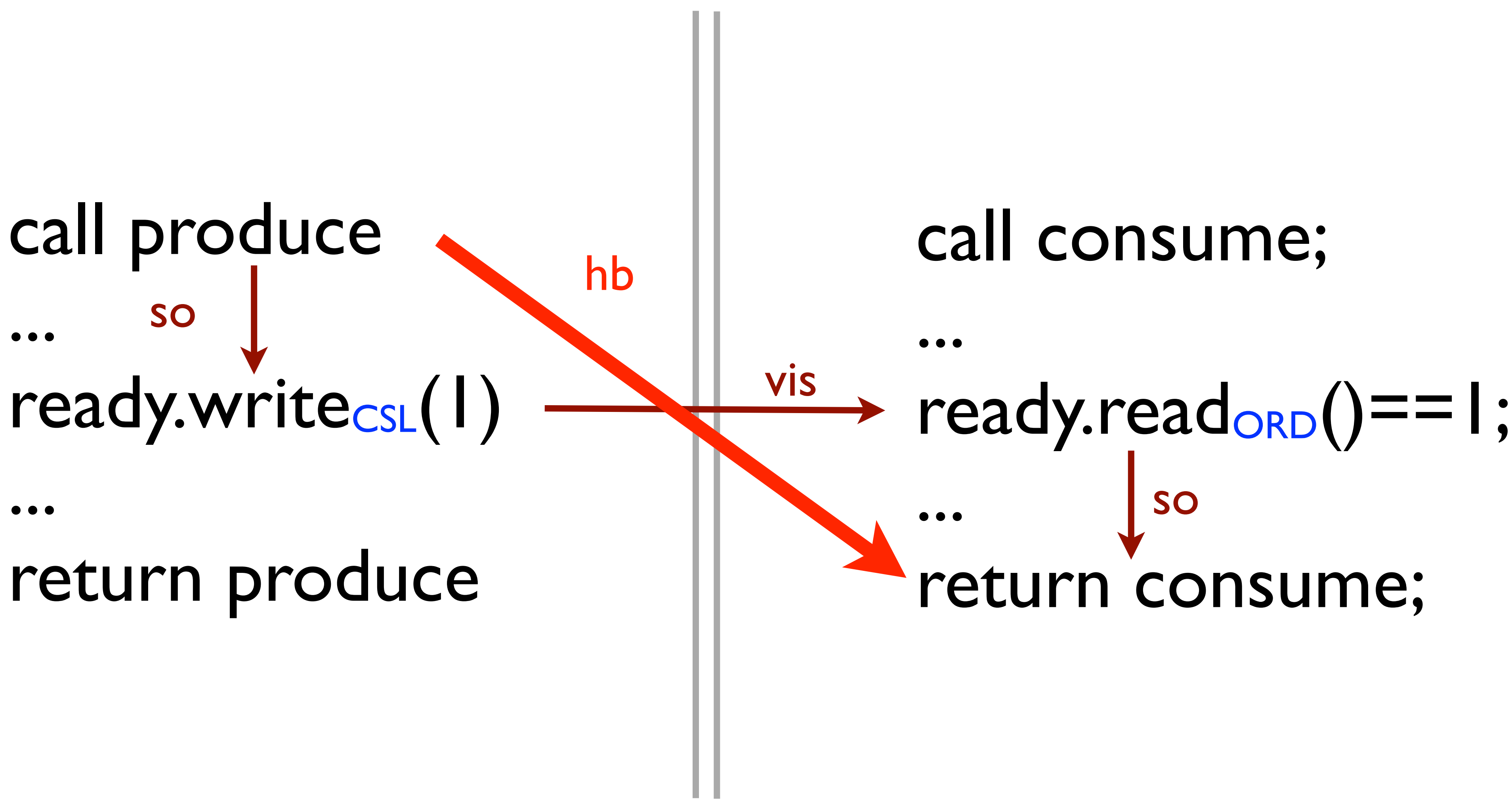
A history component

Projection of hb to calls and returns:



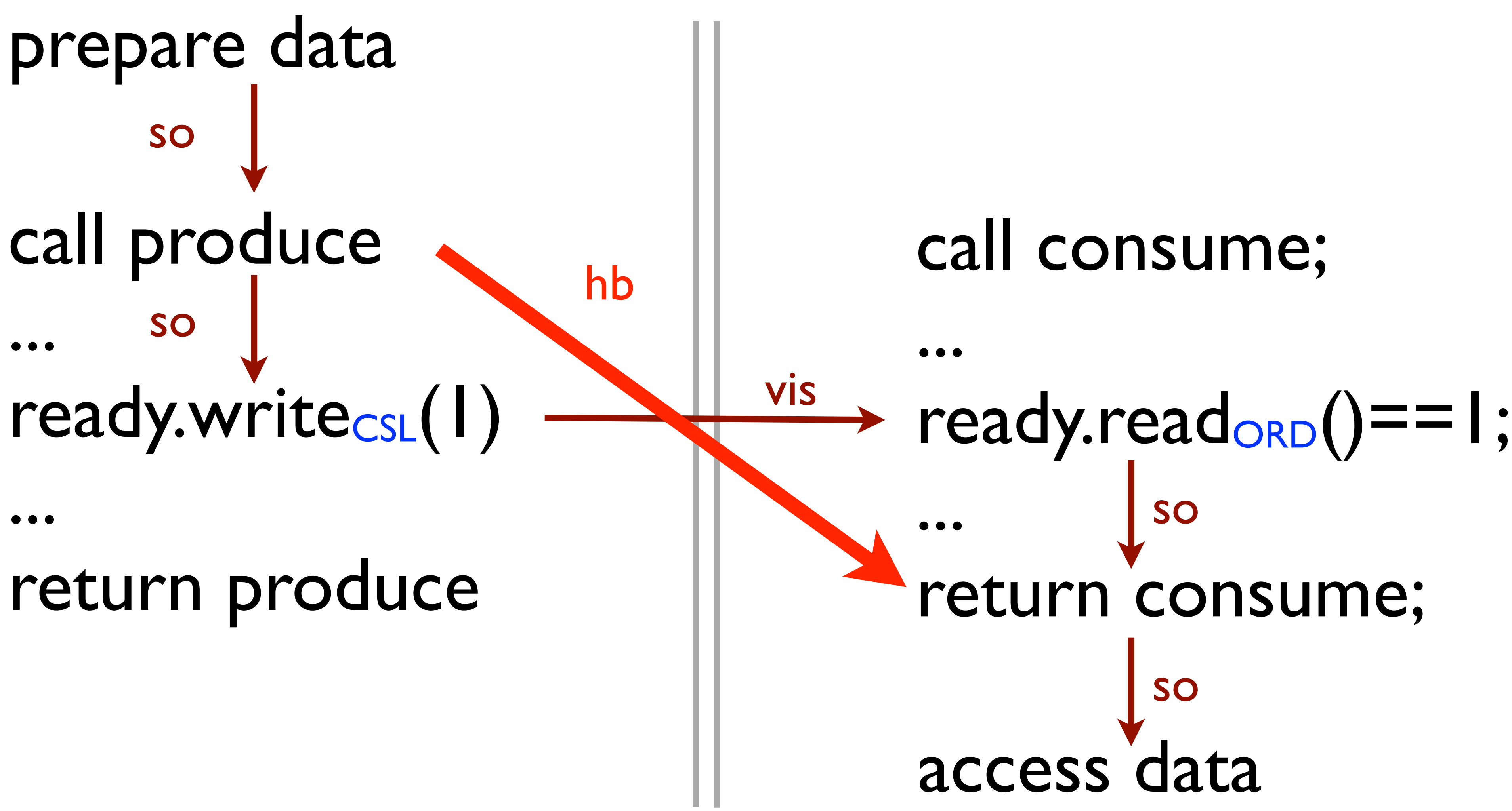
A history component

Projection of hb to calls and returns:



A history component

Projection of hb to calls and returns:



The access sees the prepared data

Conclusion

- Formal and declarative specification of forms of eventual consistency
- Unifies different systems, different data types, different levels of consistency and their combinations
- Connections to shared-memory models
- Interesting applications

Conclusion

- Formal and declarative specification of forms of eventual consistency
- Unifies different systems, different data types, different levels of consistency and their combinations
- Connections to shared-memory models
- Interesting applications

Draft paper in 2 weeks: alexey.gotsman@imdea.org

Opportunities

- Exploiting testing and verification technology developed for weak memory models
- Push compositionality further: low-level RDT implementations, practical case studies, testing
- Basis for theoretical investigation of RDTs
- Letting the programmer switch between different types of eventual consistency within the same system implementation