What is the strongest consistency that could be achieved in a large scale distributed system?

Ahmed Bouajjani, Constantin Enea, Alain Girault, Gregor Goessler, Rachid Guerraoui, Jad Hamza, Viktor Kuncak, Dragos-Adrian Seredinschi

EPFL, INRIA

8 February 2017

Postdoc at EPFL/Inria

At EPFL in collaboration with:

- Alain Girault (Inria)
- Gregor Goessler (Inria)
- Rachid Guerraoui (EPFL)
- Barbara Jobstmann (EPFL)
- Viktor Kuncak (EPFL)

Postdoc at EPFL/Inria

At EPFL in collaboration with:

- Alain Girault (Inria)
- Gregor Goessler (Inria)
- Rachid Guerraoui (EPFL)
- Barbara Jobstmann (EPFL)
- Viktor Kuncak (EPFL)

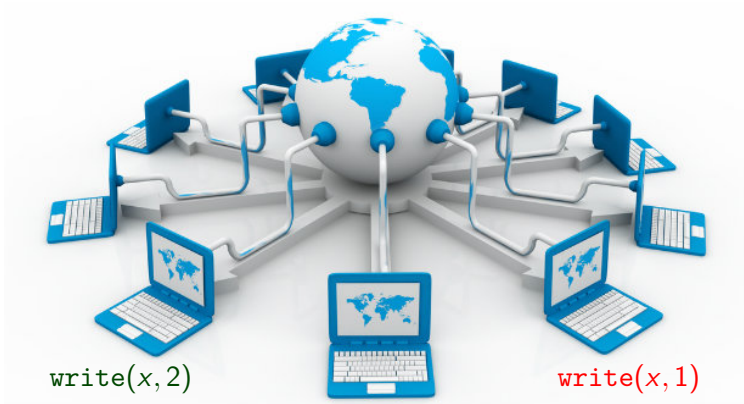
Thanks to the EPFL-Inria Postdoctoral grant

Geo-Replicated Data Structures

- **Partition** tolerance
- **Availability**

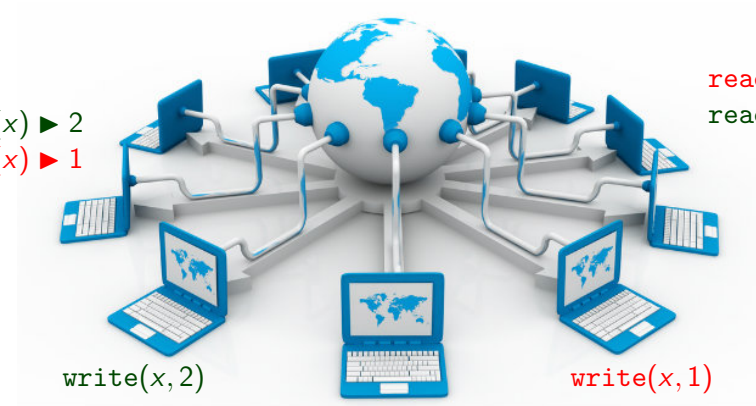
Geo-Replicated Data Structures

- **Partition** tolerance
- **Availability**



Geo-Replicated Data Structures

- **Partition** tolerance
- **Availability**



Geo-Replicated Data Structures

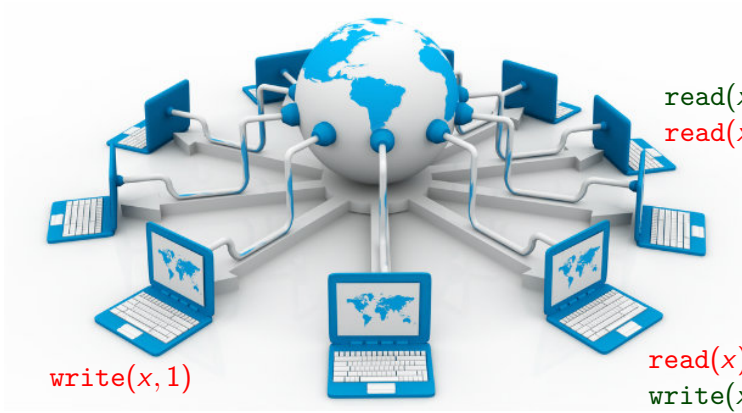
- **Partition** tolerance
- **Availability**

Geo-Replicated Data Structures

- **Partition** tolerance
- **Availability**

Geo-Replicated Data Structures

- **Partition** tolerance
- **Availability**



Geo-Replicated Data Structures

- **Partition** tolerance
- **Availability**

Consistency issues

Impossibility of Strong Consistency

Ideally: Strong consistency (**Linearizability**)
Not possible with network failures and availability[1]

[1] Brewer's Conjecture (CAP Theorem). Gilbert, Lynch.

Impossibility of Strong Consistency

Ideally: Strong consistency (**Linearizability**)
Not possible with network failures and availability[1]

$\Rightarrow$ Must use **weaker** criteria:
e.g. eventual consistency, causal consistency, prefix consistency, ...

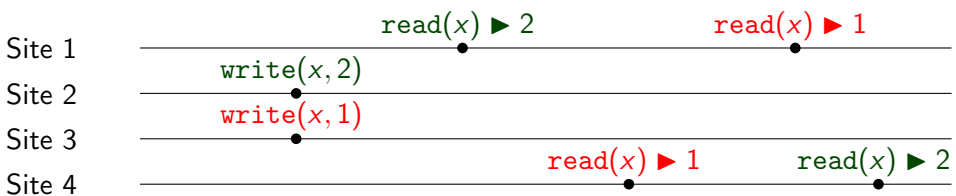
[1] Brewer's Conjecture (CAP Theorem). Gilbert, Lynch.

Impossibility of Strong Consistency

Ideally: Strong consistency (**Linearizability**)
Not possible with network failures and availability[1]

⇒ Must use **weaker** criteria:
e.g. eventual consistency, causal consistency, prefix consistency, ...

Allowed by causal consistency, but not by linearizability:



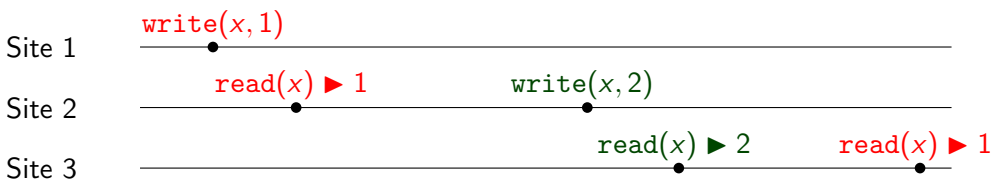
[1] Brewer's Conjecture (CAP Theorem). Gilbert, Lynch.

Impossibility of Strong Consistency

Ideally: Strong consistency (**Linearizability**)
Not possible with network failures and availability[1]

⇒ Must use **weaker** criteria:
e.g. eventual consistency, causal consistency, prefix consistency, ...

Not allowed by causal consistency (nor by linearizability)



[1] Brewer's Conjecture (CAP Theorem). Gilbert, Lynch.

The Strongest Implementation

Definition

An implementation I_1 is **stronger** than I_2 iff $I_1 \subseteq I_2$ (for traces)

The Strongest Implementation

Definition

An implementation I_1 is **stronger** than I_2 iff $I_1 \subseteq I_2$ (for traces)

Example: a **linearizable** implementation is **stronger** than a **causally consistent** implementation.

The Strongest Implementation

Definition

An implementation I_1 is **stronger** than I_2 iff $I_1 \subseteq I_2$ (for traces)

Example: a **linearizable** implementation is **stronger** than a **causally consistent** implementation.

What is the **strongest criterion** that can be implemented?

Difficulties for Implementing Distributed Data Structures

- **Communication** between **many components**
- **Network failures**

Difficulties for Implementing Distributed Data Structures

- **Communication** between **many components**
- **Network failures**
- Objects must be **highly available**
- **Optimizations**

Difficulties for Implementing Distributed Data Structures

- **Communication** between **many components**
- **Network failures**
- Objects must be **highly available**
- **Optimizations**

Easy to make **errors**. Hard to reproduce.

Difficulties for Implementing Distributed Data Structures

- **Communication** between **many components**
- **Network failures**
- Objects must be **highly available**
- **Optimizations**

Easy to make **errors**. Hard to reproduce.

$\Rightarrow$ Need for **verification** techniques

Two Verification Problems

- Check if **one trace** is consistent (Single-Trace Verification)
  - Application to testing, monitoring (by enumerating traces)

Two Verification Problems

- Check if **one trace** is consistent (Single-Trace Verification)
  - Application to testing, monitoring (by enumerating traces)
- Check if **all traces** are consistent (All-Traces Verification)
  - Requires global reasoning

Two Verification Problems

- Check if **one trace** is consistent (Single-Trace Verification)
  - Application to testing, monitoring (by enumerating traces)
- Check if **all traces** are consistent (All-Traces Verification)
  - Requires global reasoning

We study these questions for **causal consistency**.

Outline

1. Strongest Consistency Criteria
2. Verification of Causal Consistency

Outline

Constraints on the Implementations

- **Availability**: if a client makes a request, they obtain a response directly
- **Partition Tolerance**: messages between servers may be delayed/reordered/lost
- **Eventual Consistency (EC)**: every update is eventually visible to every site

Constraints on the Implementations

- **Availability**: if a client makes a request, they obtain a response directly
- **Partition Tolerance**: messages between servers may be delayed/reordered/lost
- **Eventual Consistency (EC)**: every update is eventually visible to every site

In that setting:

- Linearizable implementation is **not possible** (CAP theorem)
- Implementation where sites don't communicate: **not allowed** (not EC)
- Causal Consistency (CC) is **possible** (e.g. vector clocks)

Nothing Stronger than Causal Consistency can be Implemented

Theorem (Mahajan et al, 2011. Attiya et al, 2015.)

It is not possible to implement a consistency criterion stronger than (variants of) causal consistency on a distributed network.

Nothing Stronger than Causal Consistency can be Implemented

Theorem (Mahajan et al, 2011. Attiya et al, 2015.)

It is not possible to implement a consistency criterion stronger than (variants of) causal consistency on a distributed network.

What about criteria which are **not comparable** to causal consistency?

Monotonic Prefix Consistency

Definition (Monotonic Prefix Consistency, MPC)

All sites see increasing prefixes of the same global order

Monotonic Prefix Consistency

Definition (Monotonic Prefix Consistency, MPC)

All sites see increasing prefixes of the same global order

Example of an MPC implementation:

- Accept any **update request**, but don't commit right away.
- **Consensus algorithm** in the background to decide on order of updates
- Answer read requests using **committed writes**

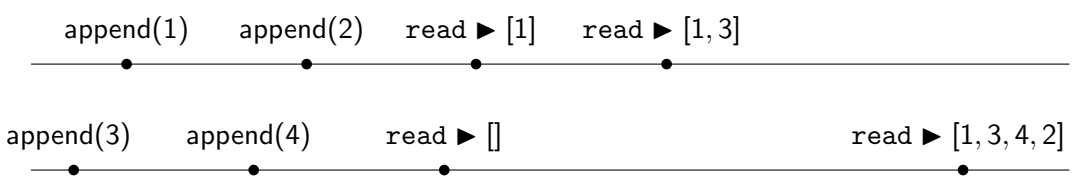
Monotonic Prefix Consistency

Definition (Monotonic Prefix Consistency, MPC)

All sites see increasing prefixes of the same global order

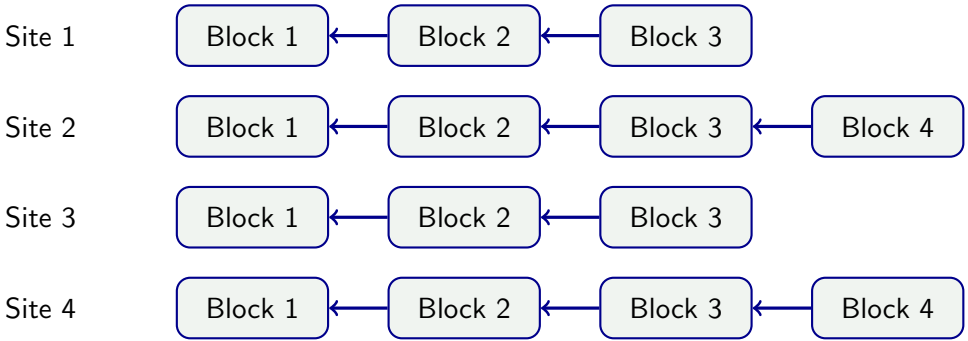
Example of an MPC implementation:

- Accept any **update request**, but don't commit right away.
- **Consensus algorithm** in the background to decide on order of updates
- Answer read requests using **committed writes**



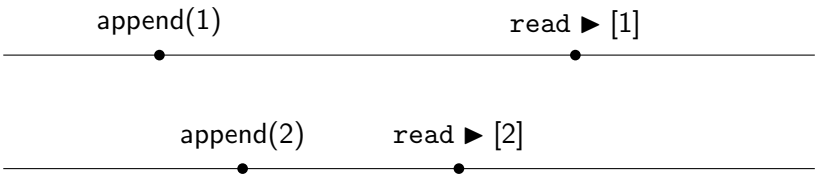
Blockchains implement Monotonic Prefix Consistency

In Bitcoin, sites maintain a **ledger** made of a chain of **blocks**



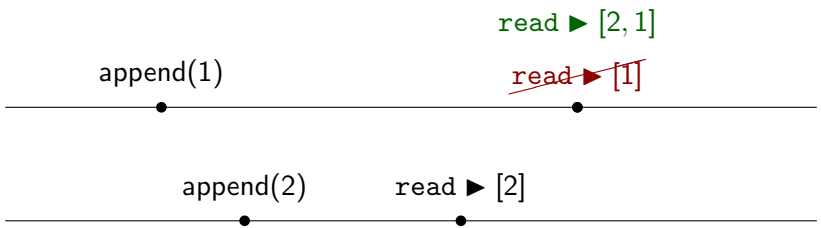
Causal Consistency and MPC are not comparable

Trace which is **causally consistent** but **not MPC**:



Causal Consistency and MPC are not comparable

Trace which is **causally consistent** but **not MPC**:



Causal Consistency and MPC are not comparable

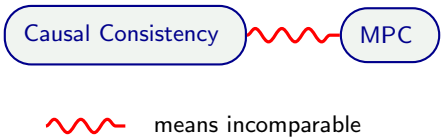
Trace which is **MPC** but **not causally consistent**:

Causal Consistency and MPC are not comparable

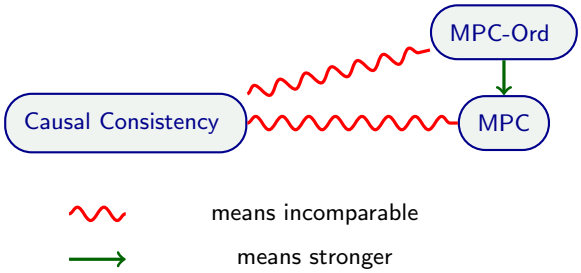
Trace which is **MPC** but **not causally consistent**:

Comparison between Consistency Criteria

Diagram showing the different implementations:



Comparison between Consistency Criteria



Monotonic Prefix Consistency With Ordered Backups (MPC-Ord)

Idea: no update is committed to **site** j before **site** $i < j$.

Monotonic Prefix Consistency With Ordered Backups (MPC-Ord)

Idea: no update is committed to **site** j before **site** $i < j$.

Implementation:

- All sites send their updates to Site 1
- Site 1 decides the order of updates, then transmits to Site 2, Site 2 commits and transmits to Site 3, and so on.

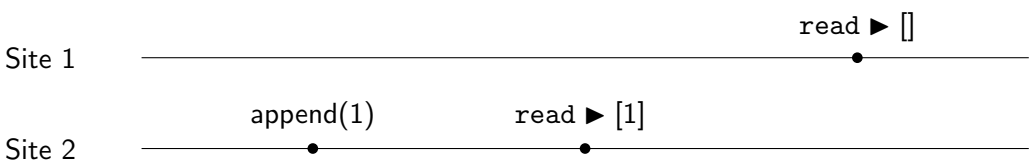
Monotonic Prefix Consistency With Ordered Backups (MPC-Ord)

Idea: no update is committed to **site** j before **site** $i < j$.

Implementation:

- All sites send their updates to Site 1
- Site 1 decides the order of updates, then transmits to Site 2, Site 2 commits and transmits to Site 3, and so on.

Trace which is **MPC** but **not MPC-Ord**.



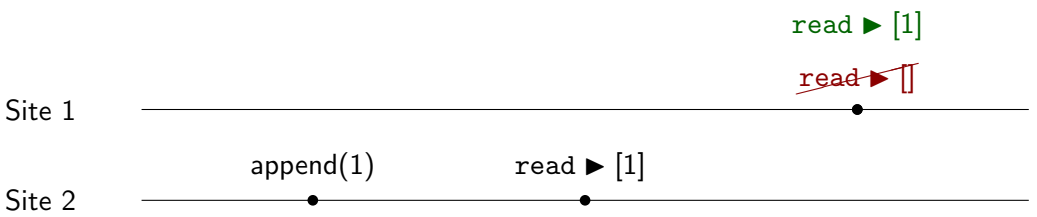
Monotonic Prefix Consistency With Ordered Backups (MPC-Ord)

Idea: no update is committed to **site** j before **site** $i < j$.

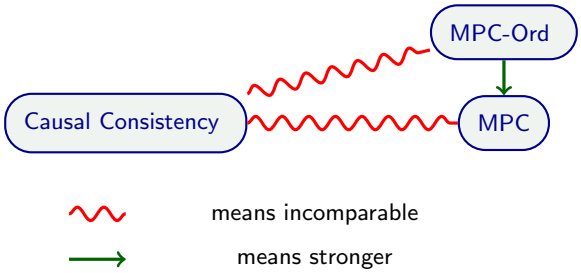
Implementation:

- All sites send their updates to Site 1
- Site 1 decides the order of updates, then transmits to Site 2, Site 2 commits and transmits to Site 3, and so on.

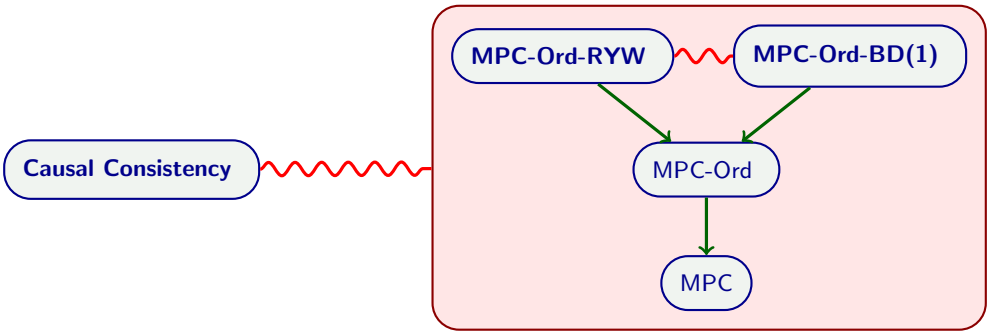
Trace which is **MPC** but **not MPC-Ord**.



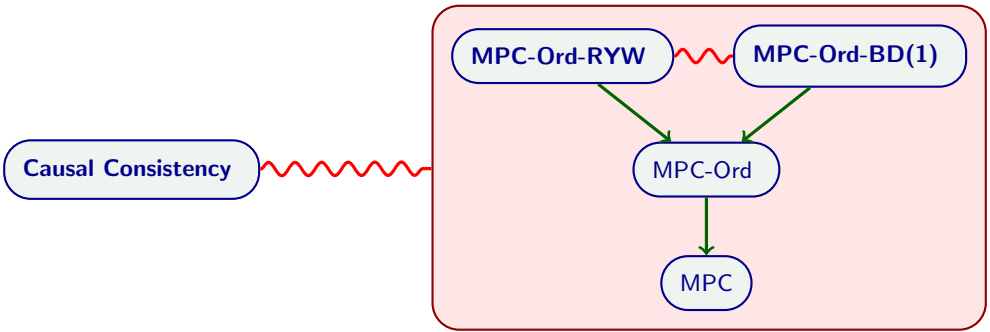
Comparison between Consistency Criteria



Comparison between Consistency Criteria



Comparison between Consistency Criteria



Questions:

- Can we prove nothing stronger than these criteria can be implemented?
- Are there criteria incomparable to these that can be implemented?

Outline

Existing Complexity Results for the Verification Problems

Single-Trace Verification:

- NP-**complete** for most consistency criteria[2]

[1] Memory Model-aware Testing. Furbach et al. 2014.
[2] Model-Checking of Correctness Conditions. Alur et al. 1996.
[3] On the complexity of linearizability. H. 2015.
[4] Verifying Eventual Consistency of ORS. Bouajjani et al. 2014.

Existing Complexity Results for the Verification Problems

Single-Trace Verification:

- NP-**complete** for most consistency criteria[2]
- **Contribution**: NP-**complete** for causal consistency as well

[1] Memory Model-aware Testing. Furbach et al. 2014.
[2] Model-Checking of Correctness Conditions. Alur et al. 1996.
[3] On the complexity of linearizability. H. 2015.
[4] Verifying Eventual Consistency of ORS. Bouajjani et al. 2014.

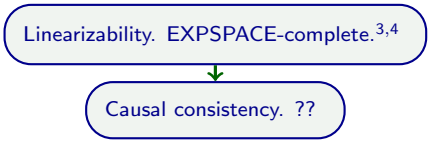
Existing Complexity Results for the Verification Problems

Single-Trace Verification:

- NP-**complete** for most consistency criteria[2]
- **Contribution**: NP-**complete** for causal consistency as well

All-Traces Verification:

- EXPSPACE-**complete** for **linearizability**[3,4]



[1]Memory Model-aware Testing. Furbach et al. 2014.
[2]Model-Checking of Correctness Conditions. Alur et al. 1996.
[3]On the complexity of linearizability. H. 2015.
[4]Verifying Eventual Consistency of ORS. Bouajjani et al. 2014.

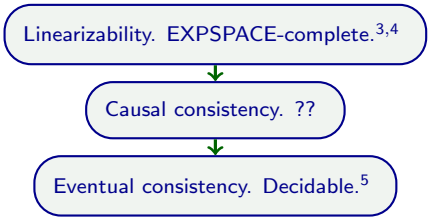
Existing Complexity Results for the Verification Problems

Single-Trace Verification:

- NP-**complete** for most consistency criteria[2]
- **Contribution**: NP-**complete** for causal consistency as well

All-Traces Verification:

- EXPSPACE-**complete** for **linearizability**[3,4]
- **Decidable** for eventual consistency[5]



[1] Memory Model-aware Testing. Furbach et al. 2014.
[2] Model-Checking of Correctness Conditions. Alur et al. 1996.
[3] On the complexity of linearizability. H. 2015.
[4] Verifying Eventual Consistency of ORS. Bouajjani et al. 2014.

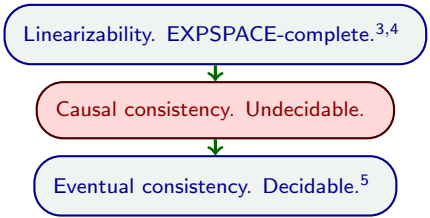
Existing Complexity Results for the Verification Problems

Single-Trace Verification:

- NP-**complete** for most consistency criteria[2]
- **Contribution**: NP-**complete** for causal consistency as well

All-Traces Verification:

- EXPSPACE-**complete** for **linearizability**[3,4]
- **Decidable** for eventual consistency[5]
- **Contribution: Undecidable** for causal consistency



[1] Memory Model-aware Testing. Furbach et al. 2014.
[2] Model-Checking of Correctness Conditions. Alur et al. 1996.
[3] On the complexity of linearizability. H. 2015.
[4] Verifying Eventual Consistency of ORS. Bouajjani et al. 2014.

What About Usual Data Structures?

Key-value store (read/write operations):
one of the **simplest** and most **widely used** data structures.

What About Usual Data Structures?

Key-value store (read/write operations):
one of the **simplest** and most **widely used** data structures.

Theorem (All-Traces Verification)

Checking if **all traces** *of an implementation are causally consistent is* **undecidable**.

What About Usual Data Structures?

Key-value store (read/write operations):
one of the **simplest** and most **widely used** data structures.

Theorem (All-Traces Verification)

Checking if **all traces** *of an implementation are causally consistent is* **undecidable**.

Even with the following restrictions:

- For **key-value stores**
- For a bounded number of **sites**
- For **finite-state** implementations
- For a bounded number of **variables**
- For a bounded variables' **domain**

What About Usual Data Structures?

Key-value store (read/write operations):
one of the **simplest** and most **widely used** data structures.

Theorem (All-Traces Verification)

Checking if **all traces** *of an implementation are causally consistent is* **undecidable**.

Even with the following restrictions:

- For **key-value stores**
- For a bounded number of **sites**
- For **finite-state** implementations
- For a bounded number of **variables**
- For a bounded variables' **domain**

(Input: **finite-state automaton** representing all traces)

Key: Implementations are Data Independent

Key-value store implementations are **data independent**

The **behaviors** do not depend on the particular values stored in the KVS.

Key: Implementations are Data Independent

Key-value store implementations are **data independent**

The **behaviors** do not depend on the particular values stored in the KVS.

($\Rightarrow$) **Writes can be assumed to be unique**

Results: Causal Consistency Violations Using Bad Patterns

Bad Pattern: A set of operations ordered in a particular way

Results: Causal Consistency Violations Using Bad Patterns

Bad Pattern: A set of operations ordered in a particular way

Theorem (Bad Patterns)

A trace is **not causally consistent** *iff it contains a* **bad pattern**.

Results: Causal Consistency Violations Using Bad Patterns

Bad Pattern: A set of operations ordered in a particular way

Theorem (Bad Patterns)

A trace is **not causally consistent** *iff it contains a* **bad pattern**.

Only need to look for **4** bad patterns.

Results: Complexity/Decidability and Reduction to Reachability

Bad patterns implications for **data-independent** implementations:

Theorem (Single-Trace Verification)

Singe-Trace Verification of causal consistency is **polynomial** *when* **writes are unique**.

Results: Complexity/Decidability and Reduction to Reachability

Bad patterns implications for **data-independent** implementations:

Theorem (Single-Trace Verification)

Singe-Trace Verification of causal consistency is **polynomial** *when* **writes are unique**.

Theorem (Reduction to Reachability)

All-Traces Verification can be reduced to **reachability** *or* **invariant checking**.
(by building a monitor (state machine) M that tracks bad patterns)

Results: Complexity/Decidability and Reduction to Reachability

Bad patterns implications for **data-independent** implementations:

Theorem (Single-Trace Verification)

Singe-Trace Verification of causal consistency is **polynomial** *when* **writes are unique**.

Theorem (Reduction to Reachability)

All-Traces Verification can be reduced to **reachability** *or* **invariant checking**.
(by building a monitor (state machine) M that tracks bad patterns)

Theorem (All-Traces Verification)

Checking whether **all traces** *of a* **data-independent** *finite-state implementation are causally consistent is* **decidable**.

Outline

- Observation: Implementation are data-independent (or parametric)
  - **Bad patterns**: operations ordered in a particular way
  - Characterize **all causal consistency violations** using bad patterns

Outline

- Observation: Implementation are data-independent (or parametric)
  - **Bad patterns**: operations ordered in a particular way
  - Characterize **all causal consistency violations** using bad patterns
- Application of bad patterns for data-independent implementations
  - Single-Trace Verification: **polynomial time**
  - **Bad patterns** can be recognized with **state machines**
  - **Generic reduction** from causal consistency to **reachability**
  - All-Traces Verification: **decidable**

Outline

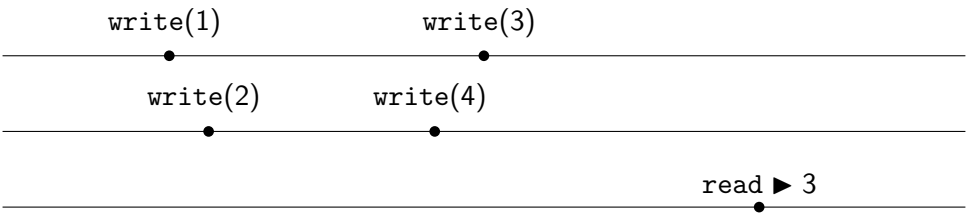
- Observation: Implementation are data-independent (or parametric)
  - **Bad patterns**: operations ordered in a particular way
  - Characterize **all causal consistency violations** using bad patterns
- Application of bad patterns for data-independent implementations
  - Single-Trace Verification: **polynomial time**
  - **Bad patterns** can be recognized with **state machines**
  - **Generic reduction** from causal consistency to **reachability**
  - All-Traces Verification: **decidable**

Data Independent Implementations

(**Observation**: Written values do not influence behaviors.
$\Rightarrow$ We can assume written values are **unique**.)

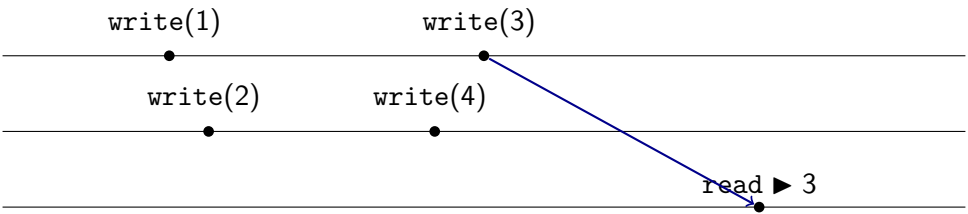
Data Independent Implementations

(**Observation**: Written values do not influence behaviors.
$\Rightarrow$ We can assume written values are **unique**.)



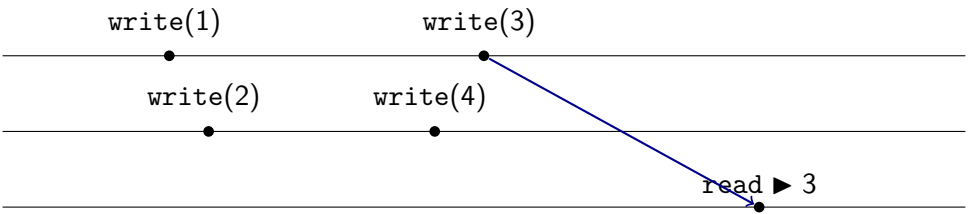
Data Independent Implementations

(**Observation**: Written values do not influence behaviors.
$\Rightarrow$ We can assume written values are **unique**.)



Data Independent Implementations

(**Observation**: Written values do not influence behaviors.
$\Rightarrow$ We can assume written values are **unique**.)



Unicity of writes automatically gives the **causality** relation.

Bad Patterns to Characterize Violations

Bad pattern: set of operations ordered is a particular way

Bad Patterns to Characterize Violations

Bad pattern: set of operations ordered is a particular way

Defined using the following orders:

- *PO* (program order): connects operations from the same site

Bad Patterns to Characterize Violations

Bad pattern: set of operations ordered is a particular way

Defined using the following orders:

- *PO* (program order): connects operations from the same site
- *RF* (reads-from relation): connects `write` to `read`

Bad Patterns to Characterize Violations

Bad pattern: set of operations ordered is a particular way

Defined using the following orders:

- PO (program order): connects operations from the same site
- RF (reads-from relation): connects `write` to `read`
- CO (causal order): defined as $(PO \cup RF)^+$

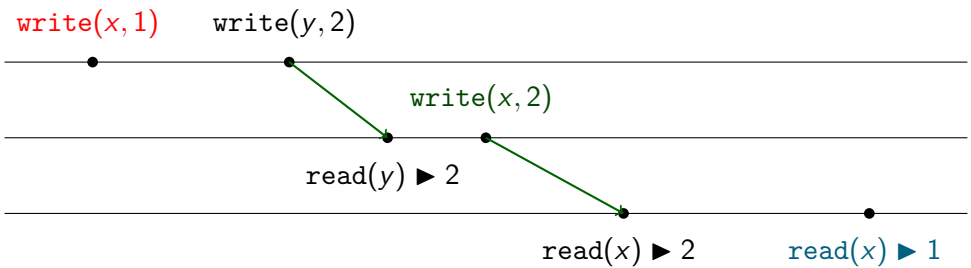
Bad Pattern Example for Causal Consistency

- Two `write` operations w_1 and w_2 on the same variable
- One `read` operation r_1 that **reads-from** w_1
- **Causal** relations $w_1 <_{CO} w_2 <_{CO} r_1$

Bad Pattern Example for Causal Consistency

- Two `write` operations w_1 and w_2 on the same variable
- One `read` operation r_1 that **reads-from** w_1
- **Causal** relations $w_1 <_{CO} w_2 <_{CO} r_1$

Example:



Bad Patterns for Causal Consistency Variants

Causal Consistency	Causal Memory	Causal Convergence
CyclicCO	CyclicCO	CyclicCO
WriteCOInitRead	WriteCOInitRead	WriteCOInitRead
ThinAirRead	ThinAirRead	ThinAirRead
WriteCORead	WriteCORead	WriteCORead
	WriteHBInitRead	CyclicCF
	CyclicHB	

Bad Patterns for Causal Consistency Variants

Causal Consistency	Causal Memory	Causal Convergence
CyclicCO	CyclicCO	CyclicCO
WriteCOInitRead	WriteCOInitRead	WriteCOInitRead
ThinAirRead	ThinAirRead	ThinAirRead
WriteCORead	WriteCORead	WriteCORead
	WriteHBInitRead	CyclicCF
	CyclicHB	

Theorem (Bad Patterns)

A trace is **not consistent** *iff it contains a* **bad pattern**.

Outline

- Observation: Implementation are data-independent (or parametric)
  - **Bad patterns**: operations ordered in a particular way
  - Characterize **all causal consistency violations** using bad patterns
- Application of bad patterns for data-independent implementations
  - Single-Trace Verification: **polynomial time**
  - **Bad patterns** can be recognized with **state machines**
  - **Generic reduction** from causal consistency to **reachability**
  - All-Traces Verification: **decidable**

Polynomial-Time Single-Trace Verification

Theorem (Single-Trace Verification)

Singe-Trace Verification of causal consistency is NP-**complete**.

Polynomial-Time Single-Trace Verification

Theorem (Single-Trace Verification)

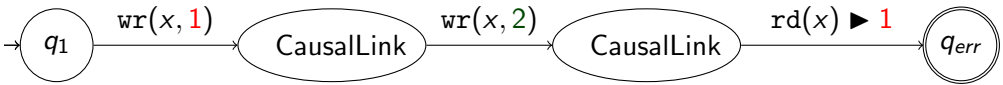
Singe-Trace Verification of causal consistency is NP-**complete**.

Theorem (Single-Trace Verification)

Singe-Trace Verification of causal consistency is **polynomial** *when* **writes are unique**. *(By checking the absence of bad patterns.)*

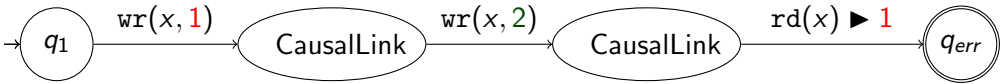
Recognizing Bad Patterns with State Machines

E.g. For one bad pattern

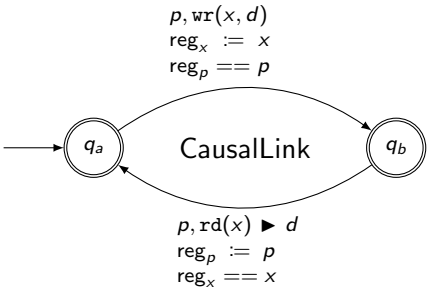


Recognizing Bad Patterns with State Machines

E.g. For one bad pattern



CausalLink tracks causality by alternating between the **program-order** and **read-from** relations



(PTime) Reduction to Reachability/Invariant Checking

Machine M tracking all bad patterns.

Theorem (Reduction to Reachability)

An implementation I is **causally consistent** *iff $I \times M$* **cannot** *reach an* **error state**.

(PTime) Reduction to Reachability/Invariant Checking

Machine M tracking all bad patterns.

Theorem (Reduction to Reachability)

An implementation I is **causally consistent** *iff $I \times M$* **cannot** *reach an* **error state**.

- Holds for **any** data-independent implementation

(PTime) Reduction to Reachability/Invariant Checking

Machine M tracking all bad patterns.

Theorem (Reduction to Reachability)

An implementation I is **causally consistent** *iff $I \times M$* **cannot** *reach an* **error state**.

- Holds for **any** data-independent implementation
- Reuse of existing tools that solve **reachability**

(PTime) Reduction to Reachability/Invariant Checking

Machine M tracking all bad patterns.

Theorem (Reduction to Reachability)

An implementation I is **causally consistent** *iff $I \times M$* **cannot** *reach an* **error state**.

- Holds for **any** data-independent implementation
- Reuse of existing tools that solve **reachability**
- Manual or semi-automated proofs

All-Traces Verification

Setting: **Finite** number of **finite-state sites**.
(All traces are modelled by a finite-state automaton)

All-Traces Verification

Setting: **Finite** number of **finite-state sites**.
(All traces are modelled by a finite-state automaton)

Theorem (All-Traces Verification)

Checking whether **all traces** *of a finite-state implementation are causally consistent is* **undecidable**.

All-Traces Verification

Setting: **Finite** number of **finite-state sites**.
(All traces are modelled by a finite-state automaton)

Theorem (All-Traces Verification)

Checking whether **all traces** *of a finite-state implementation are causally consistent is* **undecidable**.

Theorem (All-Traces Verification)

Checking whether **all traces** *of a* **data-independent** *finite-state implementation are causally consistent is* **decidable**.

Related Work

- **Single-Trace Verif.** is **NP-complete** for causal memory, and other criteria[2]
- **Data Independence**, **bad patterns** and reduction to **reachability** for linearizability[5,6,7]
- **Proof techniques** for manually proving causal consistency[8]

[5] Aspect-Oriented Linearizability Proofs. Henzinger et al. 2013.
[6] An Integrated Specification and Verification Technique for [...] Abdulla et al. 2013.
[7] On Reducing Linearizability to State Reachability. Bouajjani et al. 2015.
[8] Chapar: Certified Causally Consistent Distributed KVS. Lesani et al. 2016

Summary and Future Work

Summary:

- Difficult to verify causal consistency in general
  (Single-Trace: NP-complete, All-Traces: Undecidable)

Summary and Future Work

Summary:

- Difficult to verify causal consistency in general
  (Single-Trace: NP-complete, All-Traces: Undecidable)
- **Bad patterns** for data-independent implementations

Summary and Future Work

Summary:

- Difficult to verify causal consistency in general
  (Single-Trace: NP-complete, All-Traces: Undecidable)
- **Bad patterns** for data-independent implementations
  - Single-Trace: PTime, All-Traces: Decidable

Summary and Future Work

Summary:

- Difficult to verify causal consistency in general
  (Single-Trace: NP-complete, All-Traces: Undecidable)
- **Bad patterns** for data-independent implementations
  - Single-Trace: PTime, All-Traces: Decidable
  - Polynomial-time reduction to **reachability**: approach for verifying causal consistency

Summary and Future Work

Summary:

- Difficult to verify causal consistency in general
  (Single-Trace: NP-complete, All-Traces: Undecidable)
- **Bad patterns** for data-independent implementations
  - Single-Trace: PTime, All-Traces: Decidable
  - Polynomial-time reduction to **reachability**: approach for verifying causal consistency

Future work:

- Bad patterns for **other criteria** (Prefix Consistency, . . .)
- for **other specifications** (Multi-Value Register, CRDTs, . . .)

Summary and Future Work

Summary:

- Difficult to verify causal consistency in general
  (Single-Trace: NP-complete, All-Traces: Undecidable)
- **Bad patterns** for data-independent implementations
  - Single-Trace: PTime, All-Traces: Decidable
  - Polynomial-time reduction to **reachability**: approach for verifying causal consistency

Future work:

- Bad patterns for **other criteria** (Prefix Consistency, . . .)
- for **other specifications** (Multi-Value Register, CRDTs, . . .)
- Application to existing **causally consistent systems** to prove their correctness
  (or find bugs)

Summary and Future Work

Summary:

- Difficult to verify causal consistency in general
  (Single-Trace: NP-complete, All-Traces: Undecidable)
- **Bad patterns** for data-independent implementations
  - Single-Trace: PTime, All-Traces: Decidable
  - Polynomial-time reduction to **reachability**: approach for verifying causal consistency

Future work:

- Bad patterns for **other criteria** (Prefix Consistency, . . .)
- for **other specifications** (Multi-Value Register, CRDTs, . . .)
- Application to existing **causally consistent systems** to prove their correctness
  (or find bugs)

Thank you

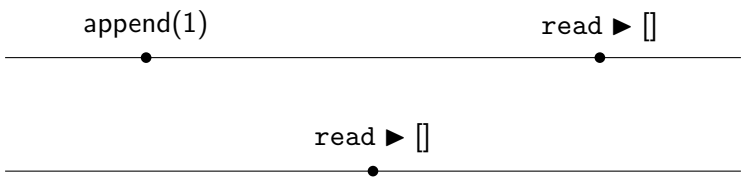
MPC-Ord Where Site 1 Commits Directly: MPC-Ord-RYW

Idea: Site 1 can commit his own updates right away: **Read-Your-Writes**

MPC-Ord Where Site 1 Commits Directly: MPC-Ord-RYW

Idea: Site 1 can commit his own updates right away: **Read-Your-Writes**

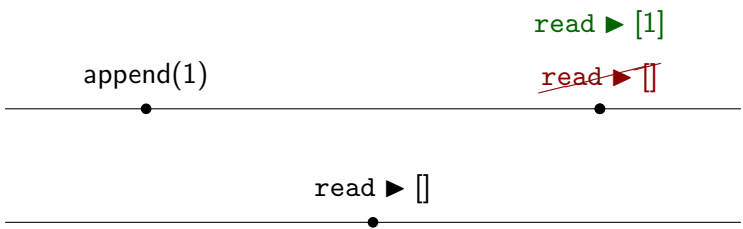
Trace which is **MPC-Ord** but **not MPC-Ord-RYW**.



MPC-Ord Where Site 1 Commits Directly: MPC-Ord-RYW

Idea: Site 1 can commit his own updates right away: **Read-Your-Writes**

Trace which is **MPC-Ord** but **not MPC-Ord-RYW**.



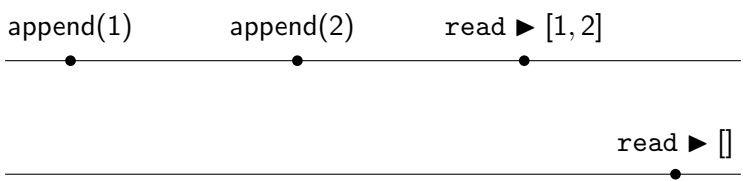
MPC-Ord With k-Bounded Delay: MPC-Ord-BD(k)

Idea: avoid too much lag between sites by waiting between consensus phases.
A site can lag at most by k updates (with respect to the other sites)

MPC-Ord With k-Bounded Delay: MPC-Ord-BD(k)

Idea: avoid too much lag between sites by waiting between consensus phases.
A site can lag at most by k updates (with respect to the other sites)

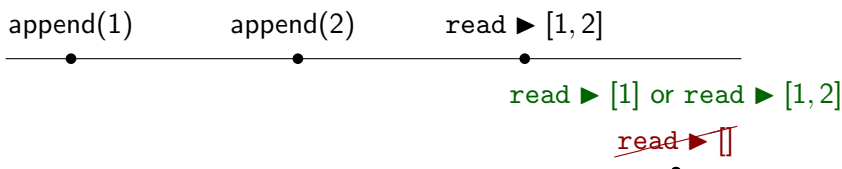
Trace which is **MPC-Ord** but **not MPC-Ord-BD(1)**.



MPC-Ord With k-Bounded Delay: MPC-Ord-BD(k)

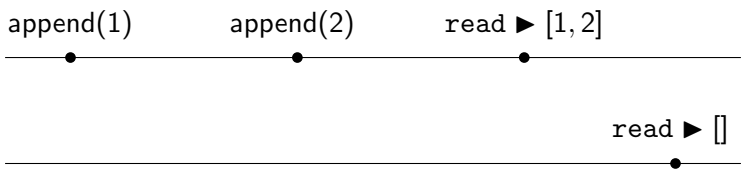
Idea: avoid too much lag between sites by waiting between consensus phases.
A site can lag at most by k updates (with respect to the other sites)

Trace which is **MPC-Ord** but **not MPC-Ord-BD(1)**.



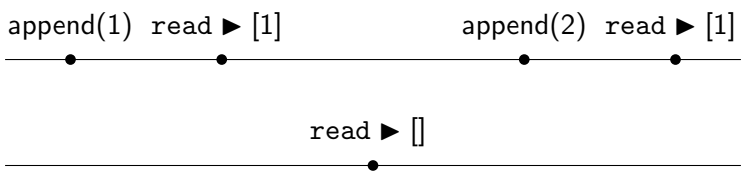
MPC-Ord-RYW and MPC-Ord-BD(1) are not Comparable

Trace which is **MPC-Ord-RYW** but **not MPC-Ord-BD(1)**.

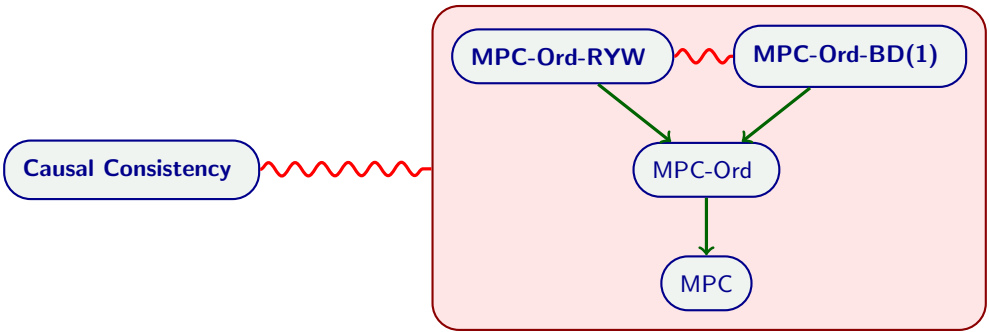


MPC-Ord-RYW and MPC-Ord-BD(1) are not Comparable

Trace which is **MPC-Ord-BD(1)** but **not MPC-Ord-RYW**.



Comparison between Consistency Criteria



Blockchain Implementation: Monotonic Prefix Consistency

Using blockchains, we can implement a data-structure that ensures **Monotonic Prefix Consistency** (with good probability).

Implementation of MPC using blockchains:

- **Update request**: broadcast it and try to add it to a block
- **Query request**: ignore the last X blocks, and return the last written value in the blockchain.

Undecidability: Post Correspondence Problem

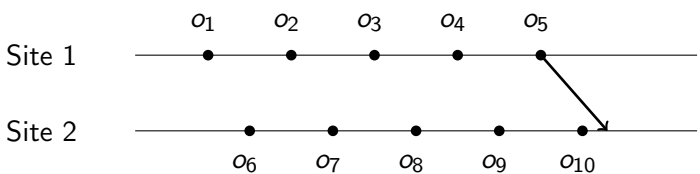
Input: a finite number of domino models $\frac{u_1}{v_1}, \frac{u_2}{v_2}, \cdots, \frac{u_n}{v_n}$
where the u_i's and v_i's are finite words.

Output: yes if there exists a (non-empty) sequence of dominoes where top = bottom
(each domino model can be used an unbounded number of times)

Undecidability: Post Correspondence Problem

Input: a finite number of domino models $\frac{u_1}{v_1}, \frac{u_2}{v_2}, \cdots, \frac{u_n}{v_n}$
where the u_i's and v_i's are finite words.

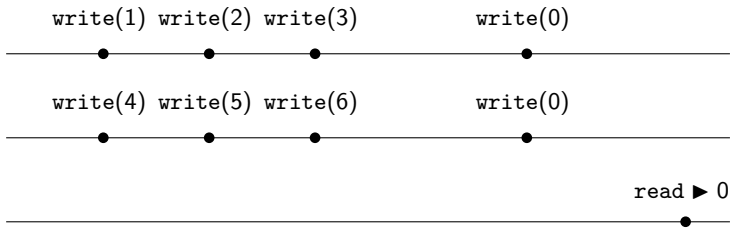
Output: yes if there exists a (non-empty) sequence of dominoes where top = bottom
(each domino model can be used an unbounded number of times)



Intuition for undecidability

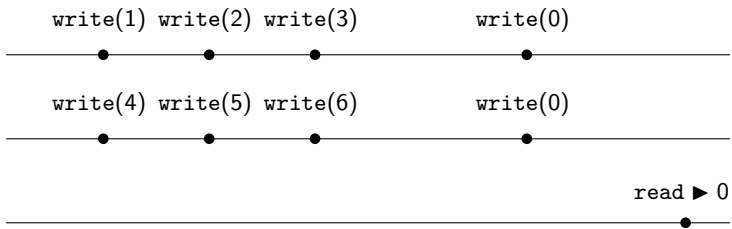
A verification algorithm needs to check **all possible traces**.

E.g. Traces where sites are disconnected for **arbitrarily long**.



Intuition for undecidability

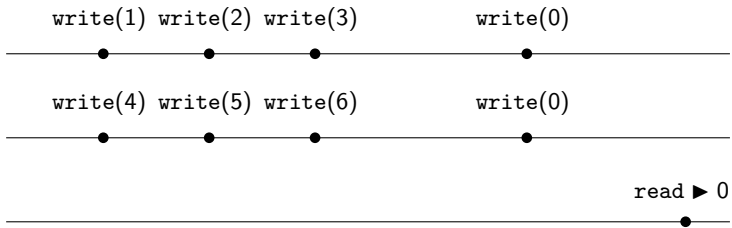
A verification algorithm needs to check **all possible traces**.
E.g. Traces where sites are disconnected for **arbitrarily long**.



- From which `write(0)` does `read` ▶ 0 read?
  ⇒ Need to remember all the writes that happened: **unbounded memory**

Intuition for undecidability

A verification algorithm needs to check **all possible traces**.
E.g. Traces where sites are disconnected for **arbitrarily long**.



- From which `write(0)` does `read` ▶ 0 read?
  ⇒ Need to remember all the writes that happened: **unbounded memory**
- Can encode the **Post correspondence problem** (undecidable)

Causal Convergence[10]

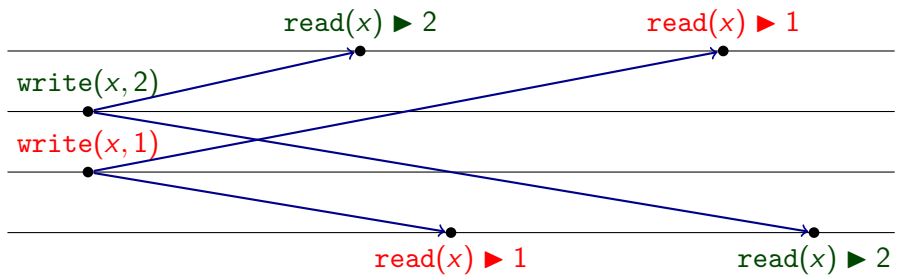
- Conflicts are resolved using a global **arbitration order**
- **Strong eventual consistency**:
  If two sites see the same writes, they are in the same state[9]

[9] A comprehensive study of CRDTs. 2011. Shapiro et al.
[10] Understanding Eventual Consistency. Burckhardt et al. 2013.

Causal Convergence[10]

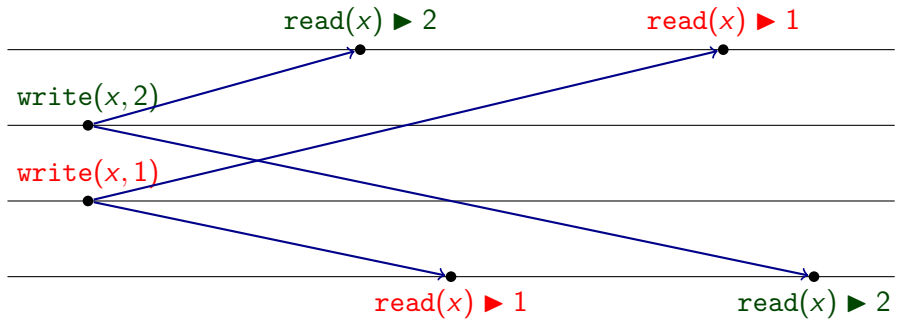
- Conflicts are resolved using a global **arbitration order**
- **Strong eventual consistency**:
  If two sites see the same writes, they are in the same state[9]

Not allowed by causal convergence:



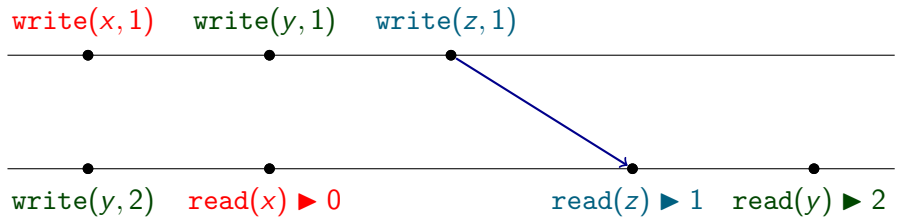
[9] A comprehensive study of CRDTs. 2011. Shapiro et al.
[10] Understanding Eventual Consistency. Burckhardt et al. 2013.

Satisfying Causal Memory but not Causal Convergence[11]



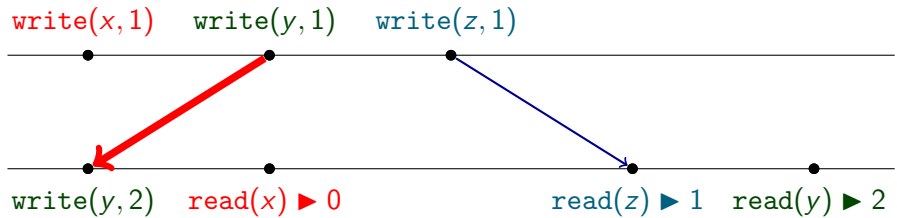
[11] Causal Consistency: Beyond Memory. Perrin et al. 2016.

Satisfying Causal Convergence but not Causal Memory[12]



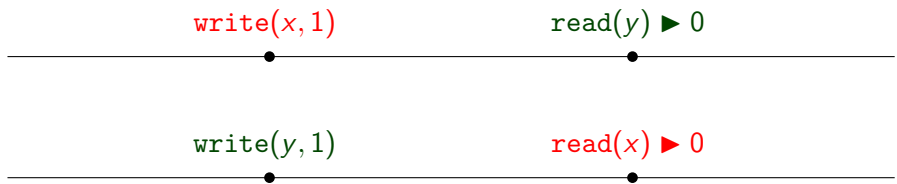
[12] Causal Consistency: Beyond Memory. Perrin et al. 2016.

Satisfying Causal Convergence but not Causal Memory[12]



[12] Causal Consistency: Beyond Memory. Perrin et al. 2016.

Satisfying Causal Consistency Variants but not Sequential Consistency



Bad Pattern Example for Causal Convergence

New relation CF: **conflict**

- if $w_1 <_{CO} r_2$, then define $w_1 <_{CF} w_2$

Bad Pattern Example for Causal Convergence

New relation CF: **conflict**

- if $w_1 <_{CO} r_2$, then define $w_1 <_{CF} w_2$

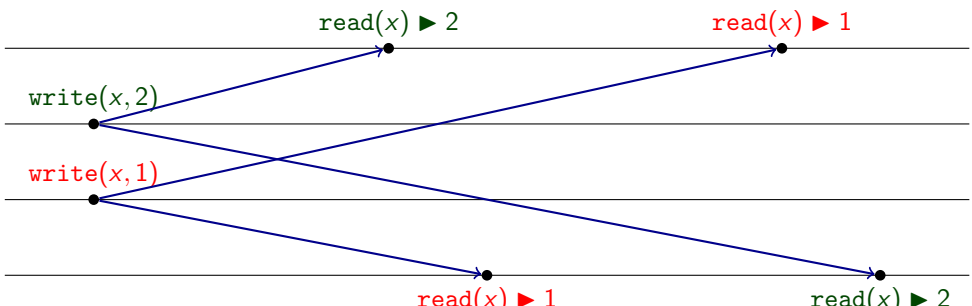
Bad pattern: cycle in the **conflict** relation

Bad Pattern Example for Causal Convergence

New relation CF: **conflict**

- if $w_1 <_{CO} r_2$, then define $w_1 <_{CF} w_2$

Bad pattern: cycle in the **conflict** relation

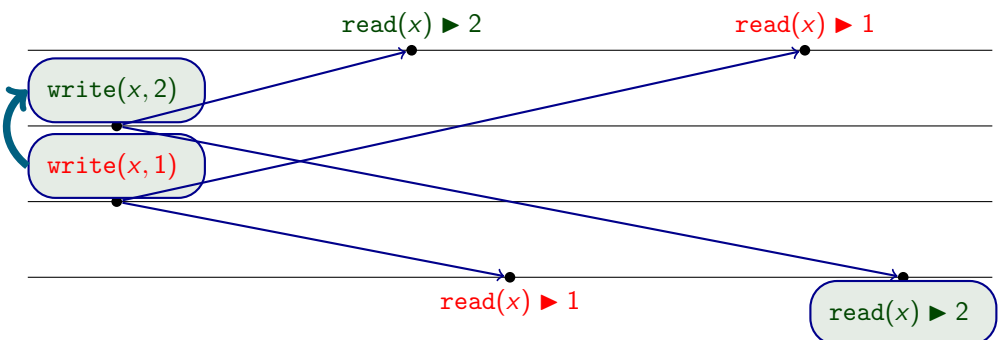


Bad Pattern Example for Causal Convergence

New relation CF: **conflict**

- if $w_1 <_{CO} r_2$, then define $w_1 <_{CF} w_2$

Bad pattern: cycle in the **conflict** relation

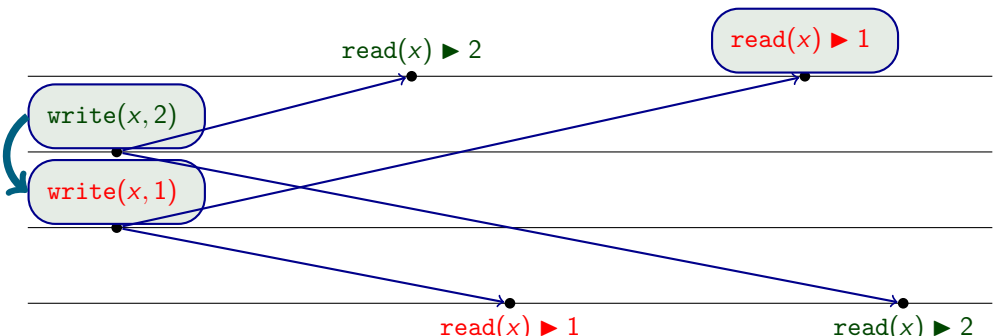


Bad Pattern Example for Causal Convergence

New relation CF: **conflict**

- if $w_1 <_{CO} r_2$, then define $w_1 <_{CF} w_2$

Bad pattern: cycle in the **conflict** relation

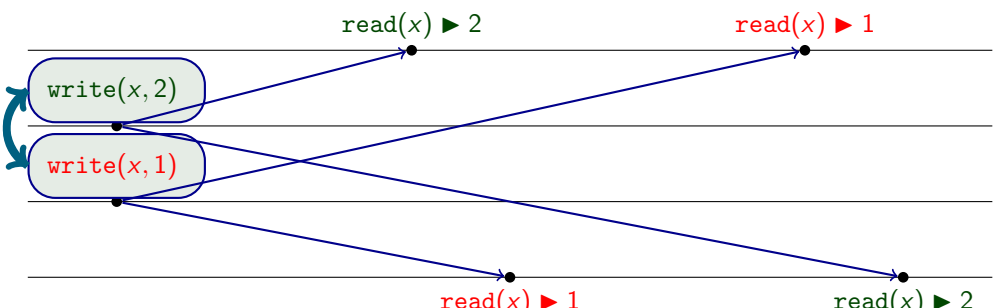


Bad Pattern Example for Causal Convergence

New relation CF: **conflict**

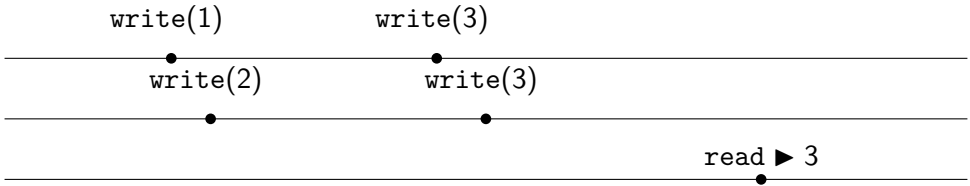
- if $w_1 <_{CO} r_2$, then define $w_1 <_{CF} w_2$

Bad pattern: cycle in the **conflict** relation



Testing is NP-Complete

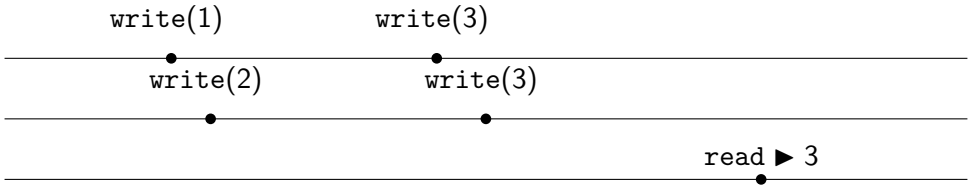
From **which** `write(3)` does `read ▶ 3` read?



[13] Memory Model-aware Testing. Furbach et al. 2014.

Testing is NP-Complete

From **which** `write(3)` does `read ▶ 3` read?

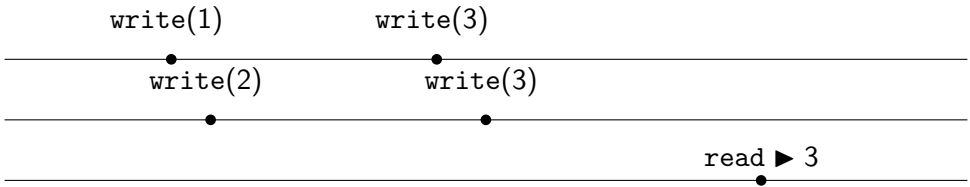


All possibilities need to be considered: exponential blow-up

[13] Memory Model-aware Testing. Furbach et al. 2014.

Testing is NP-Complete

From **which** `write`(3) does `read` ▶ 3 read?



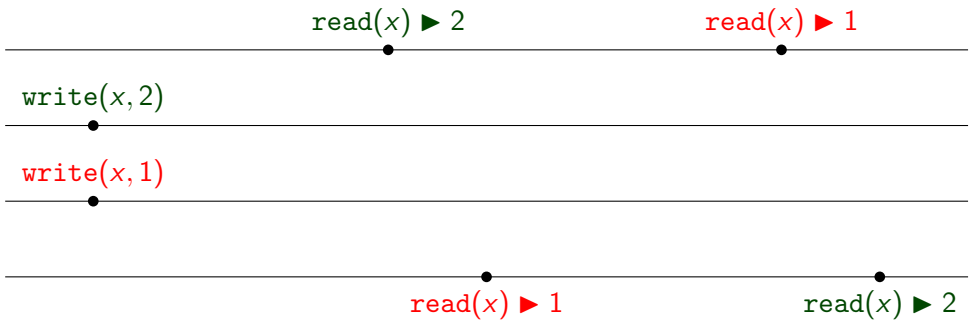
All possibilities need to be considered: exponential blow-up

Theorem (Testing)

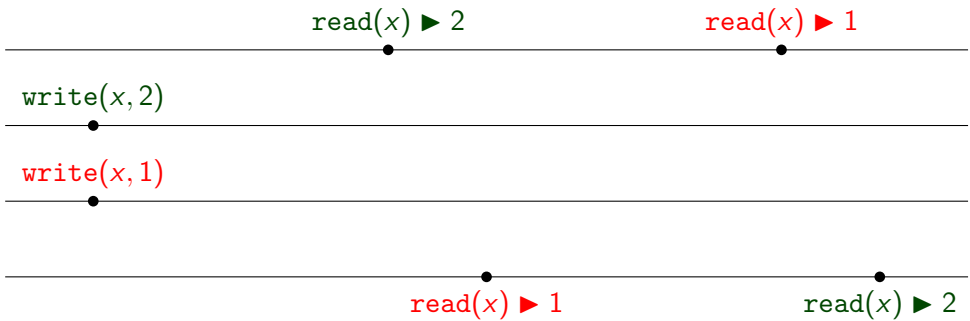
Testing causal memory[13]*, causal convergence, or causal consistency is* NP-**complete**.

[13] Memory Model-aware Testing. Furbach et al. 2014.

Definition of Causal Consistency

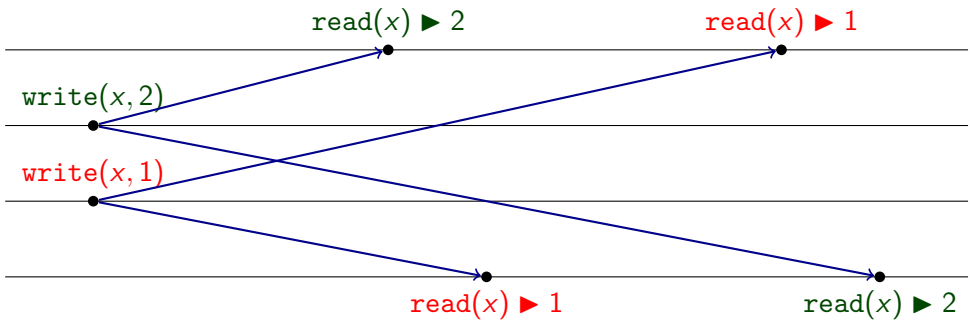


Definition of Causal Consistency



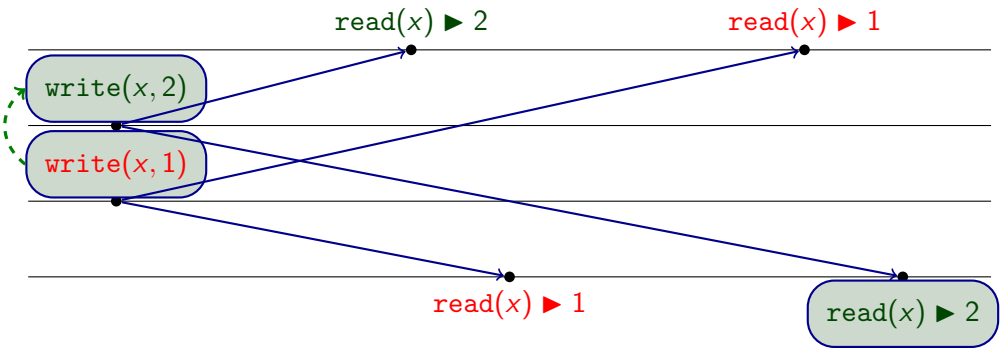
There exists a **causality order** *CO* such that for every **read**, we can order its **causal past** to explain its **return value**

Definition of Causal Consistency



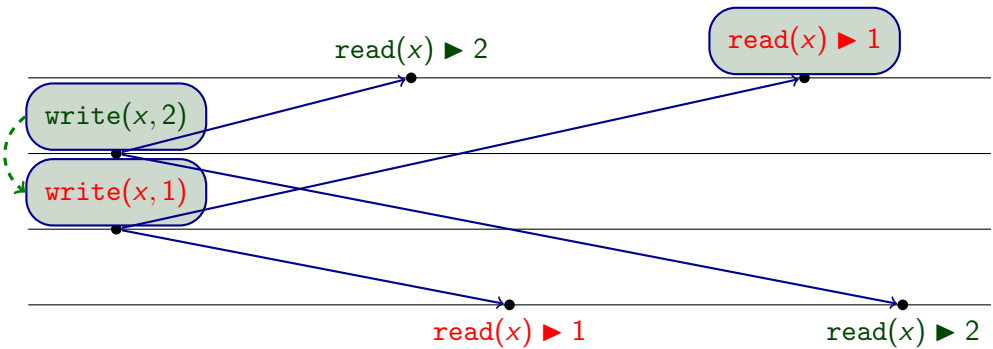
There exists a **causality order** *CO* such that for every **read**, we can order its **causal past** to explain its **return value**

Definition of Causal Consistency



There exists a **causality order** *CO* such that for every **read**, we can order its **causal past** to explain its **return value**

Definition of Causal Consistency



There exists a **causality order** *CO* such that for every **read**, we can order its **causal past** to explain its **return value**

Example of (non)-Causal Consistency

Causally related *writes must be seen by all sites in the same order.*