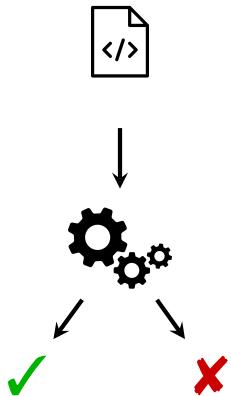
The Fine-Grained Complexity of Concurrency Testing

INFINITY 2026

Andreas Pavlogiannis

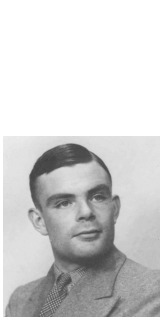
Automated Program Analysis

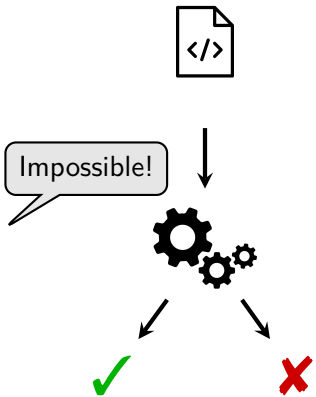
Is my program error-free?



Automated Program Analysis

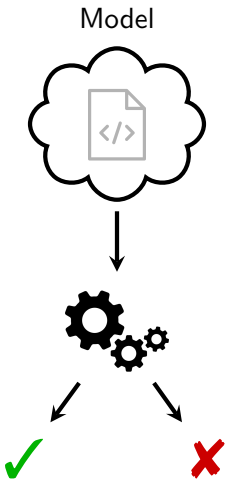
Is my program error-free?





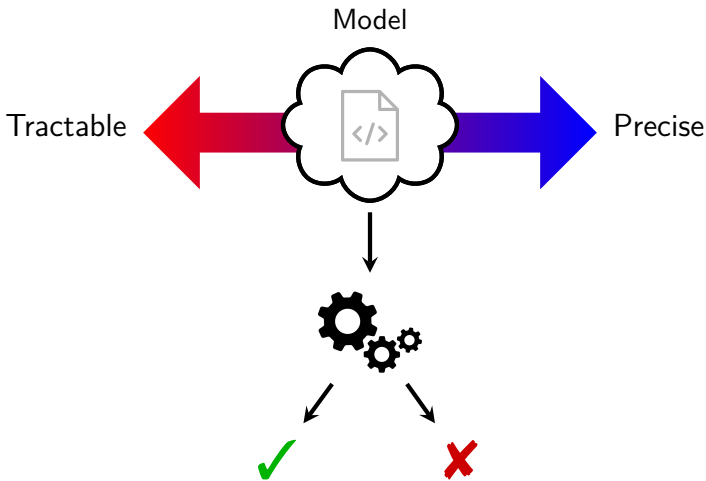
Automated Program Analysis

Is my program error-free?



Automated Program Analysis

Is my program error-free?



Good Abstractions

Do I have a good abstraction?

Good Abstractions

Do I have a good abstraction?

✓ **Algorithms**
(the abstraction is tractable)

Good Abstractions

Do I have a good abstraction?

✓ **Algorithms**
(the abstraction is tractable)

✓ **Semantics**
(the abstraction is precise)

Good Abstractions

Do I have a good abstraction?

✓ **Algorithms**
(the abstraction is tractable)

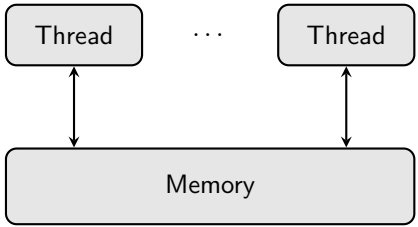
✓ **Semantics**
(the abstraction is precise)

(fine-grained) lower bounds → need for coarser abstractions

Weak Memory

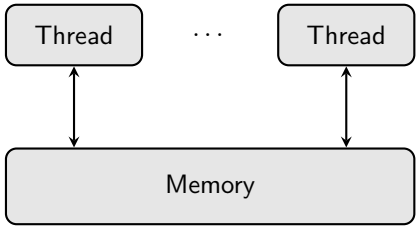
Store Buffer under (S)equential (C)onsistency

- The standard view of concurrency: communication via a *synchronous* shared memory
- Every write becomes *immediately visible* to all threads simultaneously



Store Buffer under (S)equential (C)onsistency

- The standard view of concurrency: communication via a *synchronous* shared memory
- Every write becomes *immediately visible* to all threads simultaneously



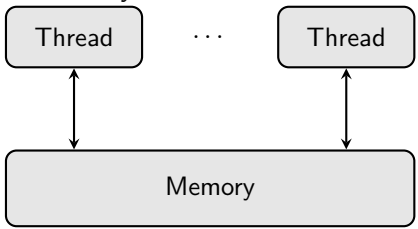
Store Buffer

$$x = 0, y = 0$$

$W(y, 1);$	$\Big\|\Big\|$	$W(x, 1);$
$a := R(x, \cdot)$		$b := R(y, \cdot)$

Store Buffer under (S)equential (C)onsistency

- The standard view of concurrency: communication via a *synchronous* shared memory
- Every write becomes *immediately visible* to all threads simultaneously



Store Buffer

$$x = 0, y = 0$$

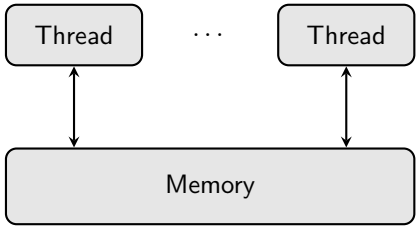
$W(y, 1);$	$\Big\|\Big\|$	$W(x, 1);$
$a := R(x, 1)$		$b := R(y, 1)$

SC Outcomes

$a = 1,\ b = 1$

Store Buffer under (S)equential (C)onsistency

- The standard view of concurrency: communication via a *synchronous* shared memory
- Every write becomes *immediately visible* to all threads simultaneously



Store Buffer

$$x = 0, y = 0$$

$W(y, 1);$	$\Big\|\Big\|$	$W(x, 1);$
$a := R(x, 1)$		$b := R(y, 0)$

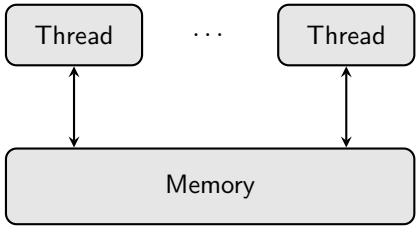
SC Outcomes

$a = 1,\ b = 1$ ✓

$a = 1,\ b = 0$

Store Buffer under (S)equential (C)onsistency

- The standard view of concurrency: communication via a *synchronous* shared memory
- Every write becomes *immediately visible* to all threads simultaneously



Store Buffer

$$x = 0, y = 0$$

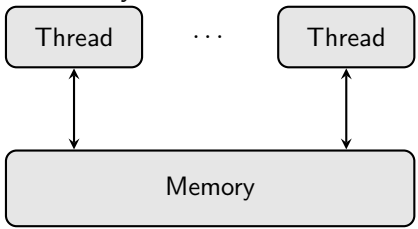
$W(y, 1);$	$\Vert$	$W(x, 1);$
$a := R(x, 0)$	$\Vert$	$b := R(y, 1)$

SC Outcomes

$a = 1,\ b = 1$ ✓
$a = 1,\ b = 0$ ✓
$a = 0,\ b = 1$

Store Buffer under (S)equential (C)onsistency

- The standard view of concurrency: communication via a *synchronous* shared memory
- Every write becomes *immediately visible* to all threads simultaneously



Store Buffer

$$x = 0, y = 0$$

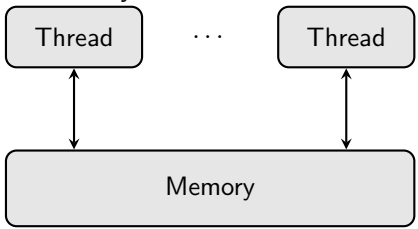
$W(y, 1);$	$\|\|$	$W(x, 1);$
$a := R(x, 0)$		$b := R(y, 0)$

SC Outcomes

$a = 1,\ b = 1$ ✓
$a = 1,\ b = 0$ ✓
$a = 0,\ b = 1$ ✓
$a = 0,\ b = 0$

Store Buffer under (S)equential (C)onsistency

- The standard view of concurrency: communication via a *synchronous* shared memory
- Every write becomes *immediately visible* to all threads simultaneously



Store Buffer

$$x = 0, y = 0$$

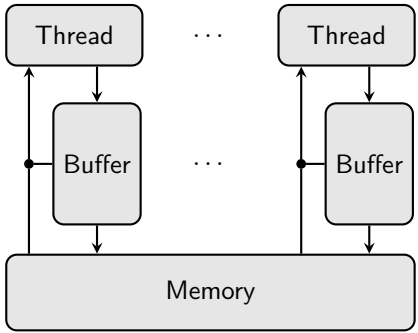
$W(y, 1);$	$\|\|$	$W(x, 1);$
$a := R(x, 0)$		$b := R(y, 0)$

SC Outcomes

$a = 1,\ b = 1$ ✓
$a = 1,\ b = 0$ ✓
$a = 0,\ b = 1$ ✓
$a = 0,\ b = 0$ ✗

The (T)otal (S)tore (O)rder Architecture

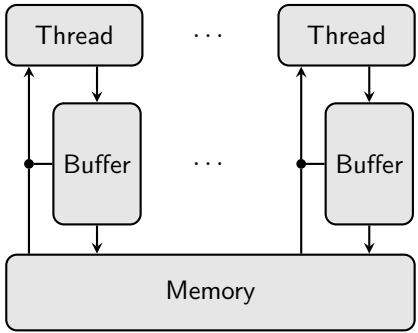
- Thread writes go to a thread-owned buffer
- Buffers non-deterministically flush to memory
- A thread always reads from its buffer, if possible



SPARC
x86

The (T)otal (S)tore (O)rder Architecture

- Thread writes go to a thread-owned buffer
- Buffers non-deterministically flush to memory
- A thread always reads from its buffer, if possible



SPARC
x86

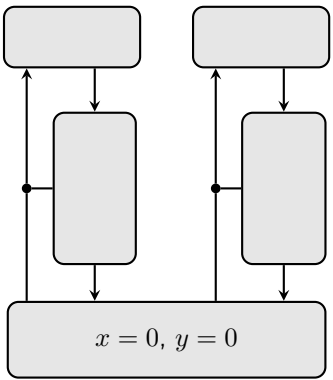
- Every SC behavior is realizable under TSO
  - Flush each buffer immediately

Store Buffer under TSO

Store Buffer

$$x = 0, y = 0$$

$W(y, 1);$	$\|\|$	$W(x, 1);$
$a := R(x, 0)$	$\|\|$	$b := R(y, 0)$

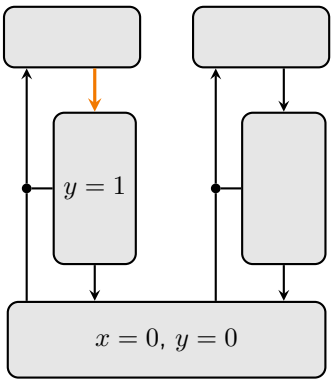


Store Buffer under TSO

Store Buffer

$$x = 0, y = 0$$

$W(y,1);$ ←	$\|\|$	$W(x,1);$
$a := R(x,0)$	$\|\|$	$b := R(y,0)$

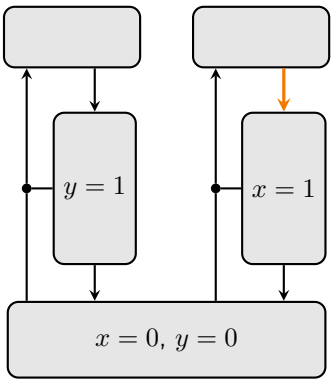


Store Buffer under TSO

Store Buffer

$$x = 0, y = 0$$

$W(y, 1);$	$\|\|$	$W(x, 1);$ ←
$a := R(x, 0)$	$\|\|$	$b := R(y, 0)$

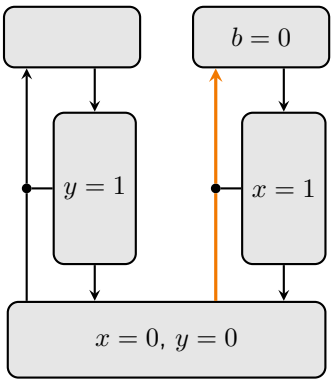


Store Buffer under TSO

Store Buffer

$$x = 0, y = 0$$

$W(y, 1);$	$\|\|$	$W(x, 1);$
$a := R(x, 0)$	$\|\|$	$b := R(y, 0)$ ←

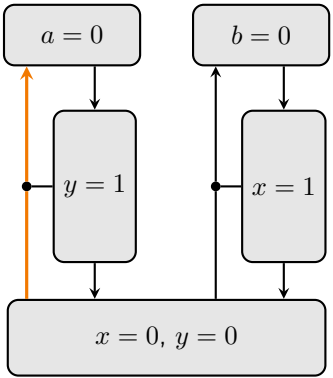


Store Buffer under TSO

Store Buffer

$$x = 0, y = 0$$

$W(y, 1);$	$\|\|$	$W(x, 1);$
$a := R(x, 0)$ ←	$\|\|$	$b := R(y, 0)$

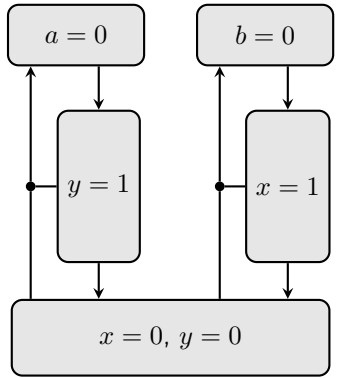


Store Buffer under TSO

Store Buffer

$$x = 0, y = 0$$

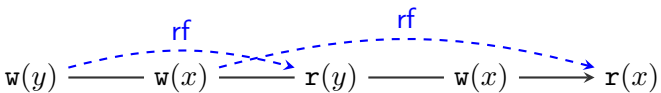
$W(y, 1);$	$\|\|$	$W(x, 1);$
$a := R(x, 0)$		$b := R(y, 0)$



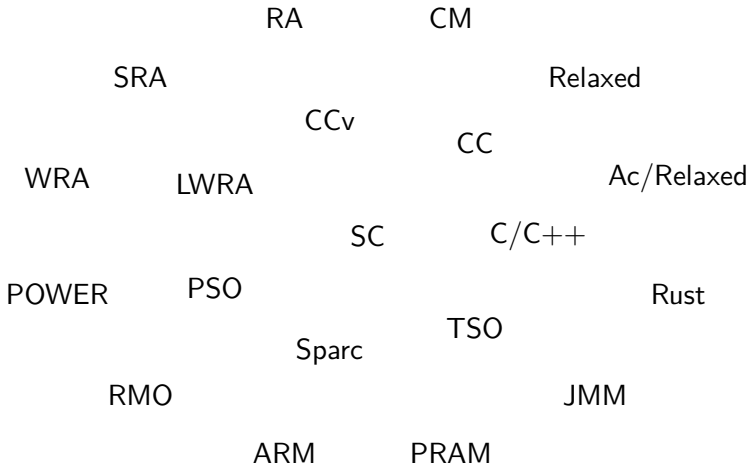
- **Reading all 0s is possible under TSO**

Sequential Consistency vs Weak Memory

- "Weak executions cannot be described in terms of (totally ordered) interleavings"
- Meaning: in every interleaving, some $\mathtt{r}(x)$ is not observing the most recent $\mathtt{w}(x)$

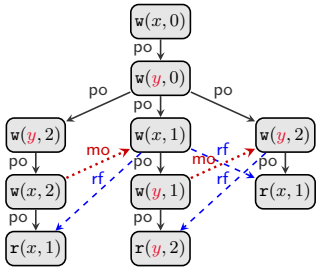


Memory Models Everywhere



Execution Graphs

Program executions are represented as execution graphs



Executions on Weak Memory

An **execution graph** is a tuple $G = (\mathsf{E}, \mathsf{po}, \mathsf{rf}, \mathsf{mo})$, where:

- E is a set of events
- po is the *program order* over E, total on each thread
- rf is a *reads-from* relation on W $\times$ R
- mo $= \bigcup_x \mathsf{mo}_x$, where mo_x is a total *modification order* over all $\mathtt{w}(x)$

The (S)equential (C)onsistency Memory Model

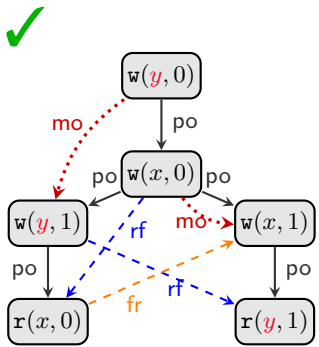
SC Consistency

$G = (\mathsf{E}, \mathsf{po}, \mathsf{rf}, \mathsf{mo})$ is **SC-consistent** if $(\mathsf{po} \cup \mathsf{rf} \cup \mathsf{mo} \cup \mathsf{fr})^+$ is irreflexive, $\mathsf{fr} \triangleq \mathsf{rf}^{-1}; \mathsf{mo}$.

The (S)equential (C)onsistency Memory Model

SC Consistency

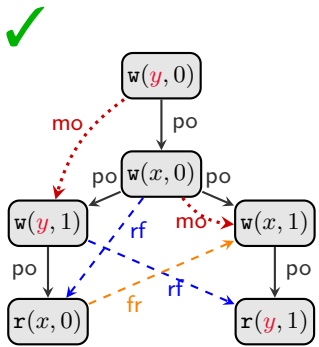
$G = (\mathsf{E}, \mathsf{po}, \mathsf{rf}, \mathsf{mo})$ is **SC-consistent** if $(\mathsf{po} \cup \mathsf{rf} \cup \mathsf{mo} \cup \mathsf{fr})^+$ is irreflexive, $\mathsf{fr} \triangleq \mathsf{rf}^{-1}; \mathsf{mo}$.

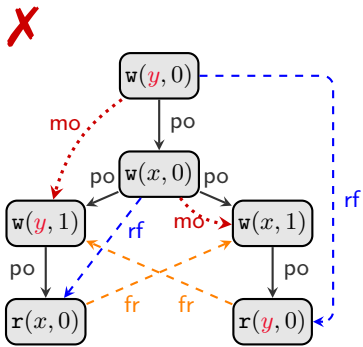


The (S)equential (C)onsistency Memory Model

SC Consistency

$G = (\mathsf{E}, \mathsf{po}, \mathsf{rf}, \mathsf{mo})$ is **SC-consistent** if $(\mathsf{po} \cup \mathsf{rf} \cup \mathsf{mo} \cup \mathsf{fr})^+$ is irreflexive, $\mathsf{fr} \triangleq \mathsf{rf}^{-1}; \mathsf{mo}$.





C/C++ Memory Access Modes

- C/C++ atomics allow the programmer to specify a **memory access mode** when accessing shared data[1]

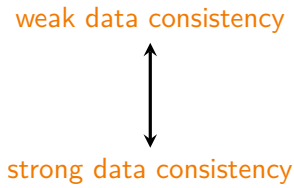
```
typedef enum memory_order {
    memory_order_relaxed,
    memory_order_acquire,
    memory_order_release,
    memory_order_acq_rel,
    memory_order_seq_cst
} memory_order;
\\...
ptr.store(v, std::memory_order_release);
```

[1] https://en.cppreference.com/w/cpp/atomic/memory_order

C/C++ Memory Access Modes

- C/C++ atomics allow the programmer to specify a **memory access mode** when accessing shared data[1]

```
typedef enum memory_order {
    memory_order_relaxed,
    memory_order_acquire,
    memory_order_release,
    memory_order_acq_rel,
    memory_order_seq_cst
} memory_order;
\\...
ptr.store(v, std::memory_order_release);
```

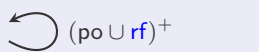


[1] https://en.cppreference.com/w/cpp/atomic/memory_order

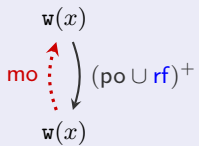
Axiomatic Semantics of Release/Acquire

An execution graph $G = (E, \mathsf{po}, \mathsf{rf}, \mathsf{mo})$ is **RA-consistent** if none of the following patterns are present

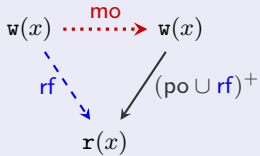
Irreflexive $(\mathsf{po} \cup \mathsf{rf})^+$



Write Coherence



Read Coherence



Examples with Release/Acquire Accesses

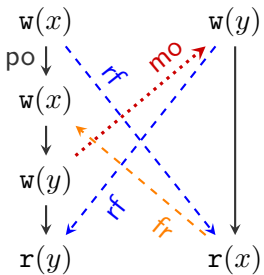
When using:

```
memory_order_acquire
memory_order_release
```

Examples with Release/Acquire Accesses

When using:

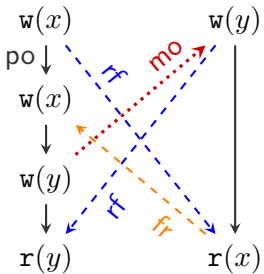
memory_order_acquire
memory_order_release

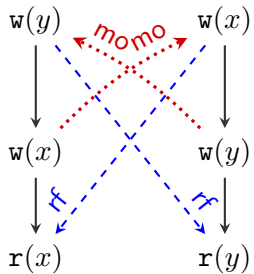


Examples with Release/Acquire Accesses

When using:

memory_order_acquire
memory_order_release



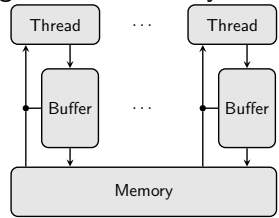


Infinite State Systems

- Programs running under weak memory are **infinite state systems**
  - Even if the program is bounded (i.e., fixed threads, locations, values)

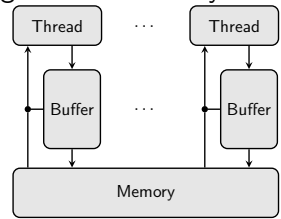
Infinite State Systems

- Programs running under weak memory are **infinite state systems**
  - Even if the program is bounded (i.e., fixed threads, locations, values)
- E.g., in TSO, the buffers can grow unboundedly



Infinite State Systems

- Programs running under weak memory are **infinite state systems**
  - Even if the program is bounded (i.e., fixed threads, locations, values)
- E.g., in TSO, the buffers can grow unboundedly

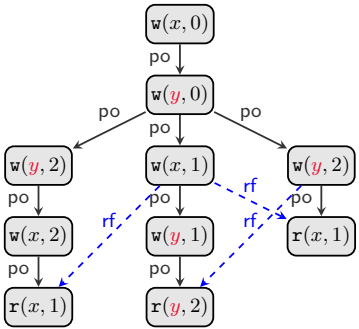


- Reachability wrt a memory model is challenging
  - Ackermann-hard for most weak memory models
  - Undecidable for some
  - **Open** for a few

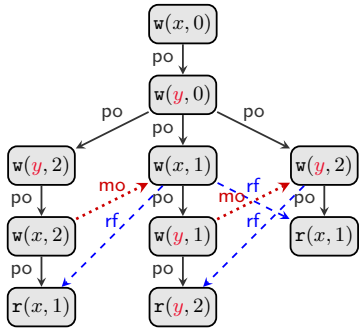
The Problem of Reads-from Consistency

Reads-from Consistency

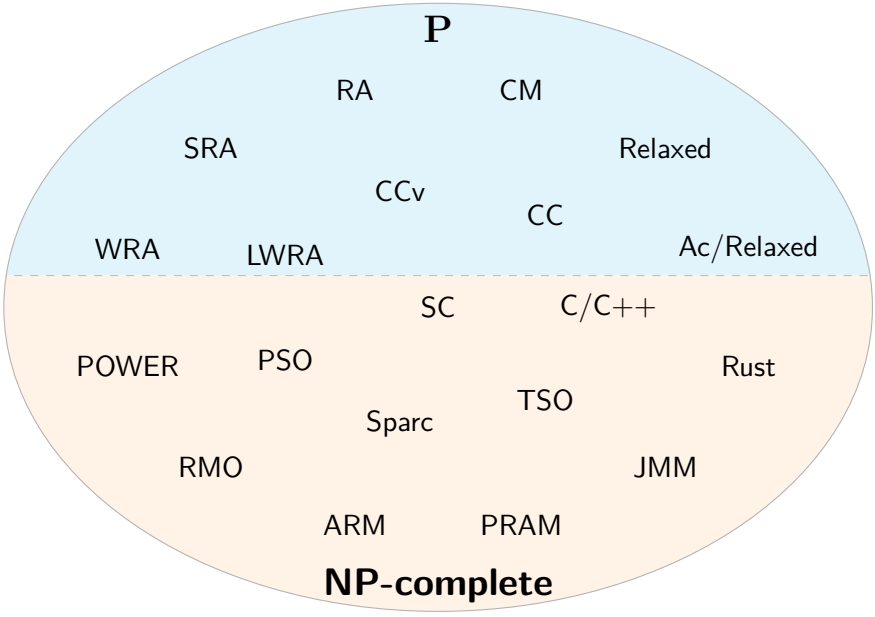
Given a rf-graph $\overline{G} = (\mathsf{E}, \mathsf{po}, \mathsf{rf})$ and a memory model $\mathcal{M}$ is there a modification order mo such that $G = (\mathsf{E}, \mathsf{po}, \mathsf{rf}, \mathsf{mo}) \vDash \mathcal{M}$?

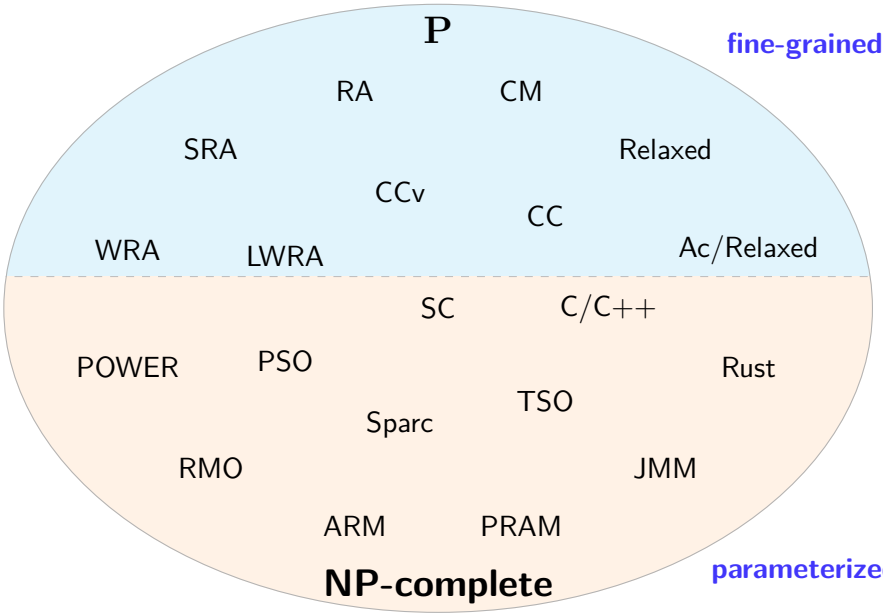


Input: rf-graph $\overline{G}$



Output: execution graph G


P
RA
CM
SRA
Relaxed
CCv
CC
WRA
LWRA
Ac/Relaxed
SC
C/C++
POWER
PSO
Rust
TSO
Sparc
RMO
JMM
ARM
PRAM
NP-complete


P
fine-grained?
RA
CM
SRA
Relaxed
CCv
CC
WRA
LWRA
Ac/Relaxed
SC
C/C++
POWER
PSO
Rust
TSO
Sparc
RMO
JMM
ARM
PRAM
NP-complete
parameterized?

Reads-from Consistency under SC

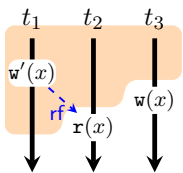
Input: reads-from graph $G = (\mathsf{E}, \mathsf{po}, \mathsf{rf})$
Create a graph H where

- verticies are the $(\mathsf{po} \cup \mathsf{rf})^+$-downwards closed subsets E

Reads-from Consistency under SC

Input: reads-from graph $G = (\mathsf{E}, \mathsf{po}, \mathsf{rf})$
Create a graph H where

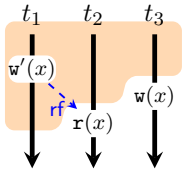
- verticies are the $(\mathsf{po} \cup \mathsf{rf})^+$-downwards closed subsets E
- $X \xrightarrow{e} X'$ iff $X' \setminus X = \{e\}$ where e is $(\mathsf{po} \cup \mathsf{rf})^+$-minimal in $\mathsf{E} \setminus X$ and
  - $e = \mathtt{r}(x)$, no conditions
  - $e = \mathtt{w}(x)$, then $\nexists \mathtt{w}'(x) \in X, \mathtt{r}(x) \notin X$ such that $\mathsf{rf}^{-1}(\mathtt{r}(x)) = \mathtt{w}'(x)$



Reads-from Consistency under SC

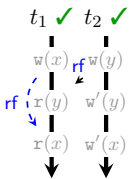
Input: reads-from graph $G = (\mathsf{E}, \mathsf{po}, \mathsf{rf})$
Create a graph H where

- verticies are the $(\mathsf{po} \cup \mathsf{rf})^+$-downwards closed subsets E
- $X \xrightarrow{e} X'$ iff $X' \setminus X = \{e\}$ where e is $(\mathsf{po} \cup \mathsf{rf})^+$-minimal in $\mathsf{E} \setminus X$ and
  - $e = \mathtt{r}(x)$, no conditions
  - $e = \mathtt{w}(x)$, then $\nexists \mathtt{w}'(x) \in X, \mathtt{r}(x) \notin X$ such that $\mathsf{rf}^{-1}(\mathtt{r}(x)) = \mathtt{w}'(x)$

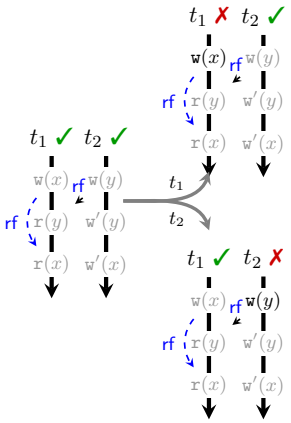


Every path $\emptyset \rightsquigarrow \mathsf{E}$ in H represents an mo such that $(\mathsf{po} \cup \mathsf{rf} \cup \mathsf{mo} \cup \mathsf{fr})^+$ is irreflexive, and vice versa

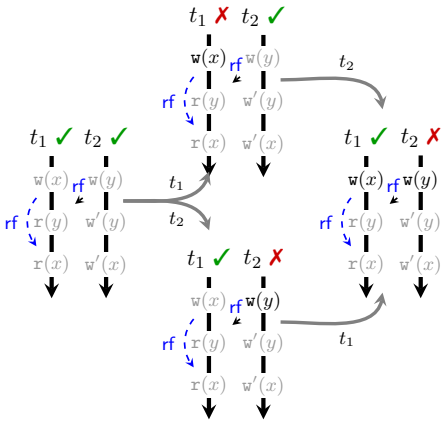
Example



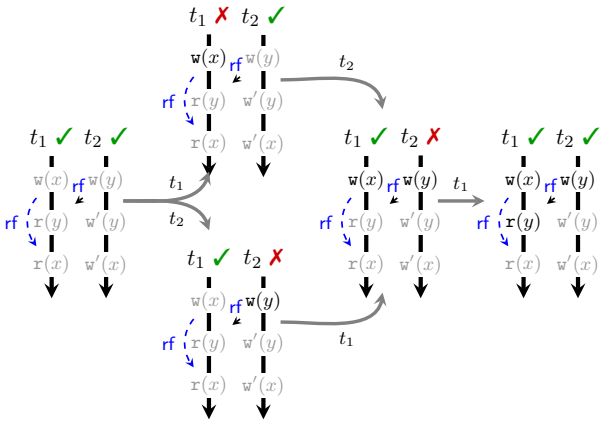
Example


t_1 ✓ t_2 ✓
w(x) rf w(y)
rf r(y) w′(y)
r(x) w′(x)
t_1
t_2
t_1 ✗ t_2 ✓
w(x) rf w(y)
rf r(y) w′(y)
r(x) w′(x)
t_1 ✓ t_2 ✗
w(x) rf w(y)
rf r(y) w′(y)
r(x) w′(x)

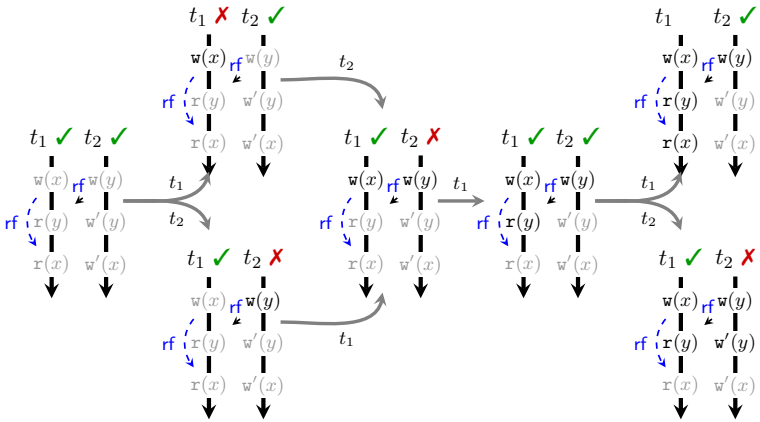
Example


t_1 ✗ t_2 ✓
w(x) rf w(y)
rf r(y) w'(y)
r(x) w'(x)
t_2
t_1 ✓ t_2 ✓
w(x) rf w(y)
rf r(y) w'(y)
r(x) w'(x)
t_1
t_2
t_1 ✓ t_2 ✗
w(x) rf w(y)
rf r(y) w'(y)
r(x) w'(x)
t_1 ✓ t_2 ✗
w(x) rf w(y)
rf r(y) w'(y)
r(x) w'(x)
t_1

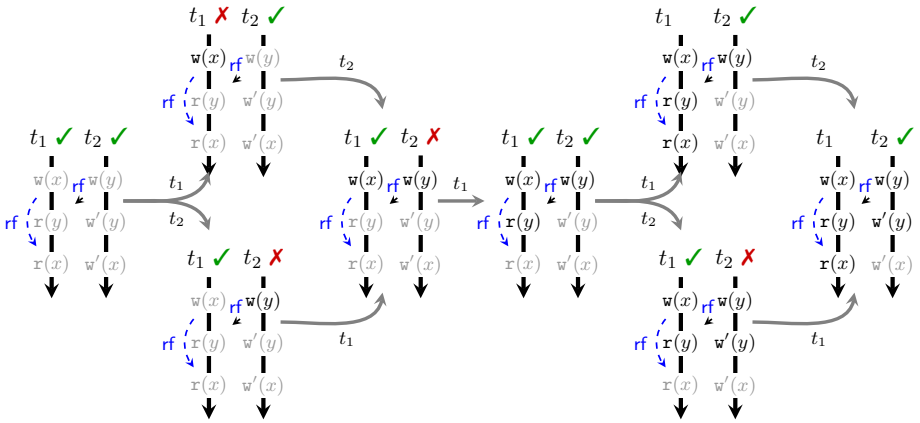
Example



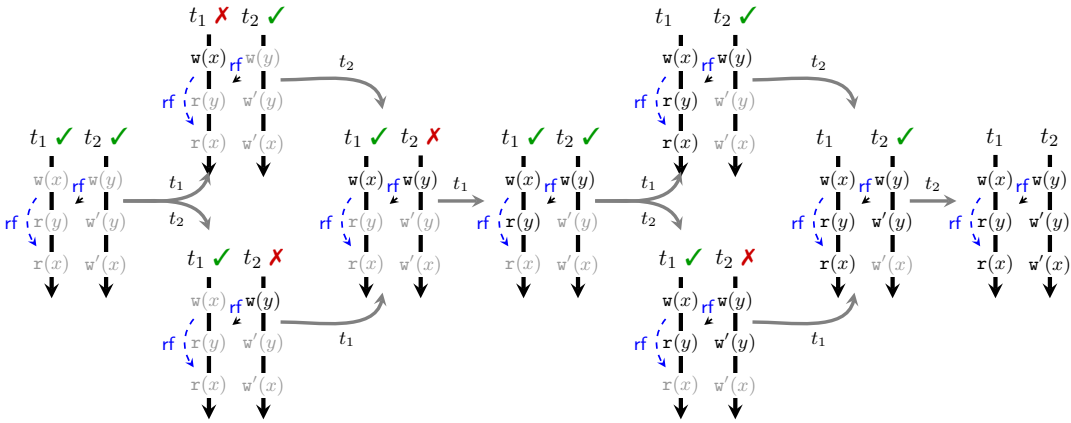
Example


t1 ✗ t2 ✓
w(x) rf w(y)
rf r(y) w′(y)
r(x) w′(x)
t2
t1 ✓ t2 ✓
w(x) rf w(y)
rf r(y) w′(y)
r(x) w′(x)
t1
t2
t1 ✓ t2 ✗
w(x) rf w(y)
rf r(y) w′(y)
r(x) w′(x)
t1
t1 ✓ t2 ✓
w(x) rf w(y)
rf r(y) w′(y)
r(x) w′(x)
t1
t2
t1 t2 ✓
w(x) rf w(y)
rf r(y) w′(y)
r(x) w′(x)
t1 ✓ t2 ✗
w(x) rf w(y)
rf r(y) w′(y)
r(x) w′(x)

Example



Example



Reads-from Consistency Parameterized

Theorem

Reads-from consistency under SC is solvable in $O(n^k)$ time,

- n *is the number of events, and*
- k *is the number of threads*

Idea: $(\mathsf{po} \cup \mathsf{rf})^+$ *has width* $\leq k \rightarrow H$ *has size* $O(n^k)$

Reads-from Consistency Parameterized

Theorem

Reads-from consistency under SC is solvable in $O(n^k)$ time,

- n *is the number of events, and*
- k *is the number of threads*

Idea: $(\mathsf{po} \cup \mathsf{rf})^+$ *has width* $\leq k \rightarrow H$ *has size* $O(n^k)$

Theorem

Reads-from consistency under SC is

- W[1]*-hard parameterized by the number of threads* k
- *not solvable in* $f(k)n^{o(k)}$ *under ETH*

Reads-from Consistency Parameterized

Theorem

Reads-from consistency under SC is solvable in $O(n^k)$ time,

- n *is the number of events, and*
- k *is the number of threads*

Idea: $(\mathsf{po} \cup \mathsf{rf})^+$ *has width* $\leq k \rightarrow H$ *has size* $O(n^k)$

Theorem

Reads-from consistency under SC is

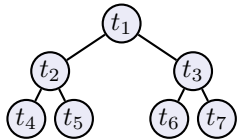
- W[1]*-hard parameterized by the number of threads* k
- *not solvable in* $f(k)n^{o(k)}$ *under ETH*

- W[1]-hardness is over $\Theta(n)$ memory locations
- What about bounded threads?

Tree Topologies and Fine-grained Hardness

Communication topology

- t_i-t_j iff t_i and t_j touch common memory



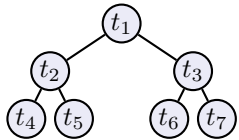
Theorem

Reads-from consistency under SC is solvable in $O(n^2)$ time over tree topologies.

Tree Topologies and Fine-grained Hardness

Communication topology

- t_i-t_j iff t_i and t_j touch common memory



Theorem

Reads-from consistency under SC is solvable in $O(n^2)$ time over tree topologies.

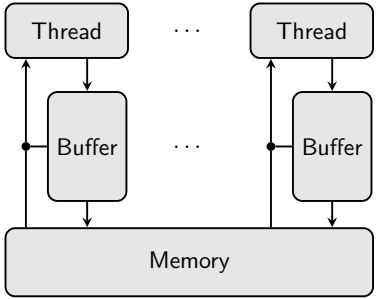
Theorem

There is linear-time reduction from OV to Reads-from consistency under SC over rf-graphs $G = (\mathsf{E}, \mathsf{po}, \mathsf{rf})$ with only

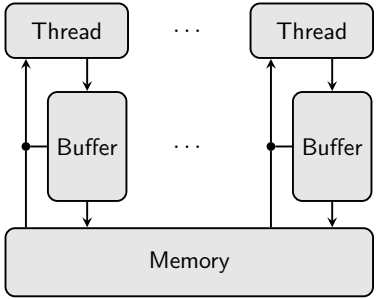
- $k = 2$ *threads, and*
- $d = 6$ *memory locations*

$\implies$ RF-consistency over this family takes $\Omega(n^{2-\epsilon})$ under OVH

Reads-from Consistency under TSO

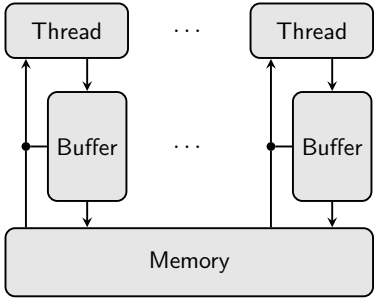


Reads-from Consistency under TSO



- One buffer per thread $\rightarrow$ consistency checking in $O(n^{2k})$

Reads-from Consistency under TSO



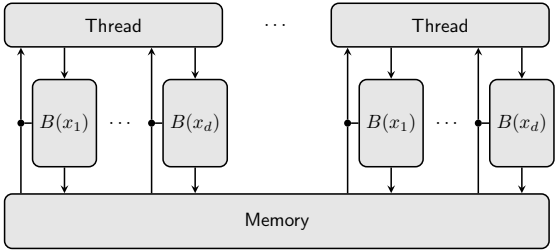
- One buffer per thread $\rightarrow$ consistency checking in $O(n^{2k})$

Theorem

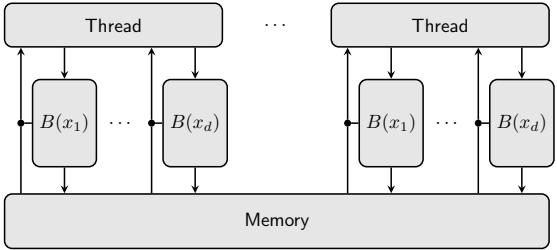
Reads-from consistency under TSO *is solvable in* $O(n^{k+1})$ *time,*

- n *is the number of events and*
- k *is the number of threads.*

Reads-from Consistency under PSO

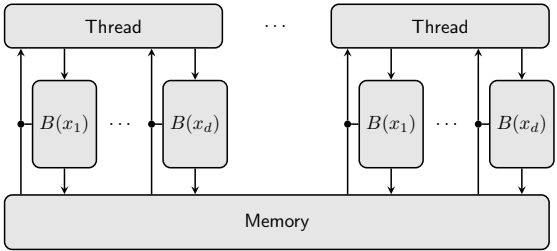


Reads-from Consistency under PSO



- One buffer per thread per location $\rightarrow$ consistency checking in $O(n^{k \cdot d})$

Reads-from Consistency under PSO

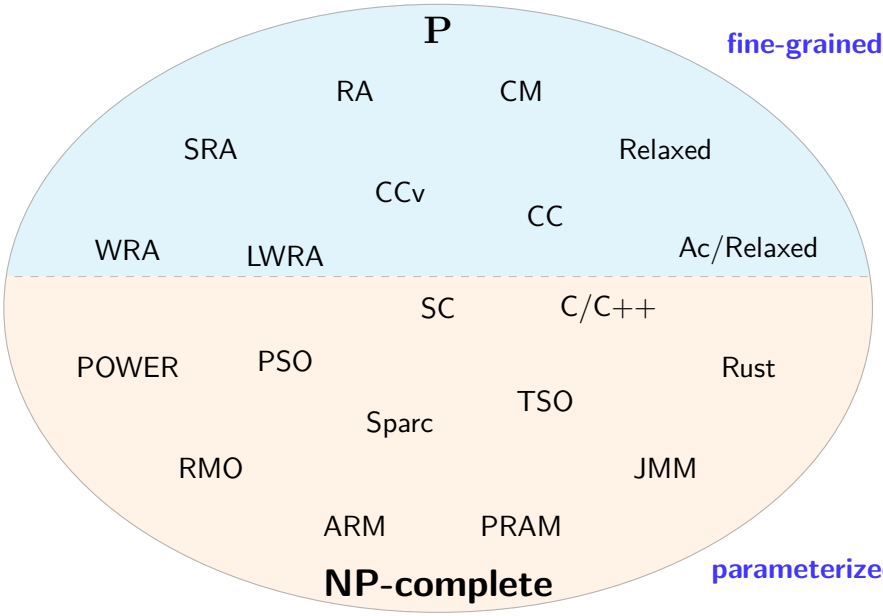


- One buffer per thread per location $\rightarrow$ consistency checking in $O(n^{k \cdot d})$

Theorem

Reads-from consistency under PSO *is solvable in* $O(n^{k+1} \cdot \min(n^{k^2}, 2^{k \cdot d}))$ *time,*

- n *is the number of events*
- k *is the number of threads, and*
- d *is the number of memory locations.*


P
fine-grained?
RA
CM
SRA
Relaxed
CCv
CC
WRA
LWRA
Ac/Relaxed
SC
C/C++
POWER
PSO
Rust
TSO
Sparc
RMO
JMM
ARM
PRAM
NP-complete
parameterized?

Reads-from Consistency for Causal Models

Fine-grained Upper Bounds

Memory model	CC	LWRA	RA	CCv	Ac/Relaxed	Relaxed
Running time	$O(nk)$	$O(nk)$	$O(nk)$	$O(nk)$	$O(n)$	$O(n)$

for n events and k threads

Reads-from Consistency for Causal Models

Fine-grained Upper Bounds

Memory model	CC	LWRA	RA	CCv	Ac/Relaxed	Relaxed
Running time	$O(nk)$	$O(nk)$	$O(nk)$	$O(nk)$	$O(n)$	$O(n)$

for n events and k threads

Lower bounds? $O(n)$ for all?

Fine-grained Range Reductions

Comparing Memory Models

Given memory models $\mathcal{M}_1$, $\mathcal{M}_2$, call $\mathcal{M}_1$ **stronger** than $\mathcal{M}_2$ if for all execution graphs G, we have $G \vDash \mathcal{M}_1 \implies G \vDash \mathcal{M}_2$.

Fine-grained Range Reductions

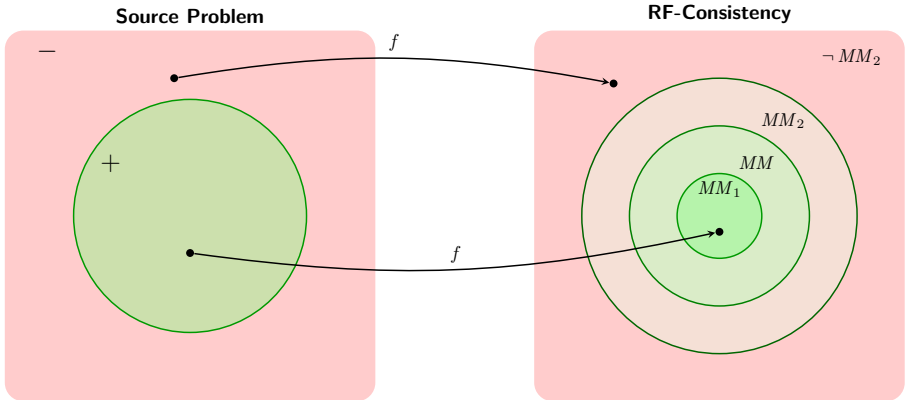
Comparing Memory Models

Given memory models $\mathcal{M}_1$, $\mathcal{M}_2$, call $\mathcal{M}_1$ **stronger** than $\mathcal{M}_2$ if for all execution graphs G, we have $G \vDash \mathcal{M}_1 \implies G \vDash \mathcal{M}_2$. E.g., SC $\sqsubseteq$ TSO $\sqsubseteq$ CCv $\sqsubseteq$ RA $\sqsubseteq \{\mathsf{CC}, \mathsf{Ac/Relaxed}\}$

Fine-grained Range Reductions

Comparing Memory Models

Given memory models $\mathcal{M}_1$, $\mathcal{M}_2$, call $\mathcal{M}_1$ **stronger** than $\mathcal{M}_2$ if for all execution graphs G, we have $G \vDash \mathcal{M}_1 \implies G \vDash \mathcal{M}_2$. E.g., SC $\sqsubseteq$ TSO $\sqsubseteq$ CCv $\sqsubseteq$ RA $\sqsubseteq$ {CC, Ac/Relaxed}



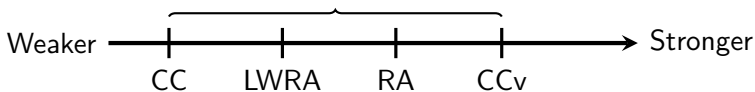
Reads-from Consistency for Causal Models

Fine-grained upper bounds

Memory model	CC	LWRA	RA	CCv	Ac/Relaxed
Running time	$O(nk)$	$O(nk)$	$O(nk)$	$O(nk)$	$O(n)$

for n events and k threads

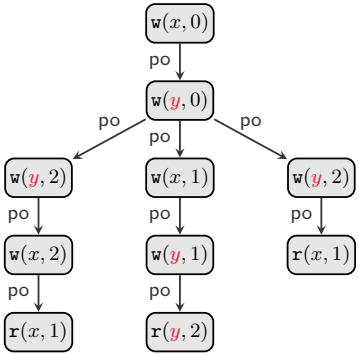
lower bounds: $n\sqrt{n}$ under comb. BMM, $n^{\omega/2}$ in general



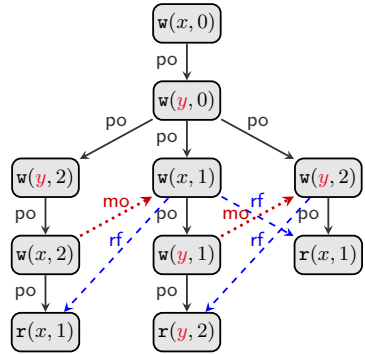
Value Consistency under Weak Memory

Value Consistency

Given a po-graph $\overline{G} = (\mathsf{E}, \mathsf{po})$ and a memory model $\mathcal{M}$ is there a reads from relation rf and a modification order mo such that $G = (\mathsf{E}, \mathsf{po}, \mathsf{rf}, \mathsf{mo}) \vDash \mathcal{M}$?



Input: po-graph $\overline{G}$

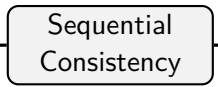


Output: execution graph G

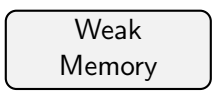
The (Parameterized) Complexity of Value Consistency?

n events, k threads, d memory locations

NP-complete for $k = 3$
NP-complete for $d = 1$

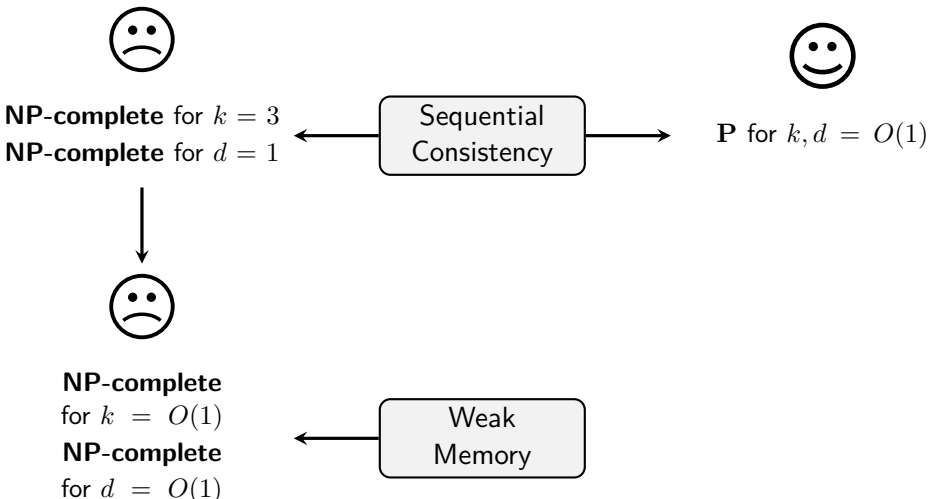


P for $k, d = O(1)$

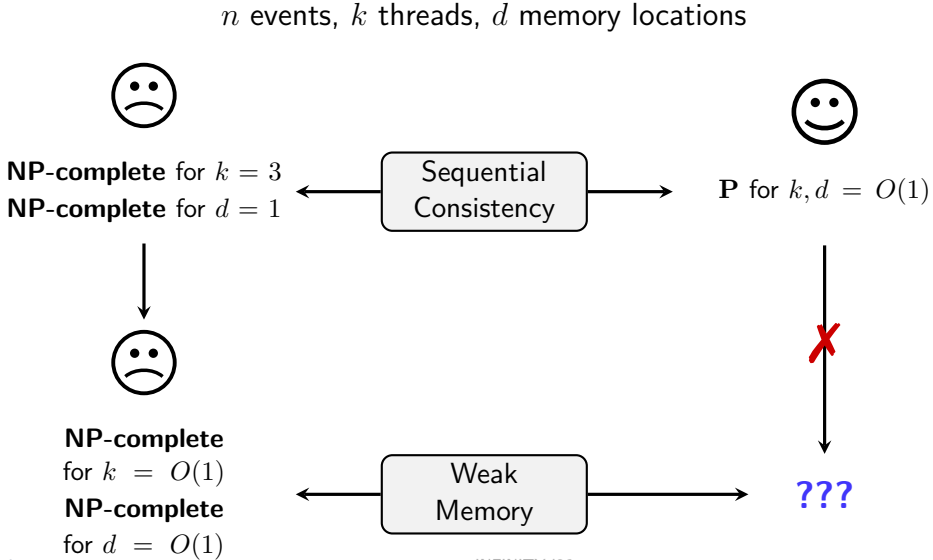


The (Parameterized) Complexity of Value Consistency?

n events, k threads, d memory locations



The (Parameterized) Complexity of Value Consistency?



Value-Consistency for Causal Models

Theorem

Value-consistency is **NP-*hard*** *for any memory model among*

- CCv
- RA
- CM
- CC
- Ac/Relaxed

even for po*-graphs with a bounded number of threads, memory locations, and values.*

Value-Consistency for Causal Models

Theorem

Value-consistency is **NP-*hard*** *for any memory model among*

- CCv
- RA
- CM
- CC
- Ac/Relaxed

Non parameterizable!

(wrt standard syntactic parameters)

even for po*-graphs with a bounded number of threads, memory locations, and values.*

The Proof is a Bit Involved ...

Reduction from monotone 1-in-3 SAT $\phi = \mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \cdots \wedge \mathcal{C}_m$

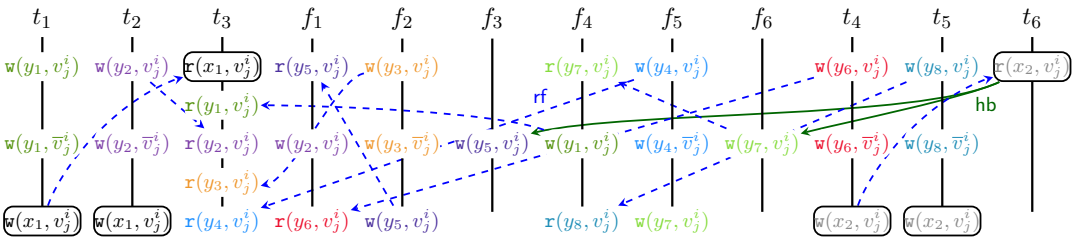
- No negations: $\mathcal{C}_i = x_i^1 \vee x_i^2 \vee x_i^3$
- For each clause, exactly one variable must be true

The Proof is a Bit Involved ...

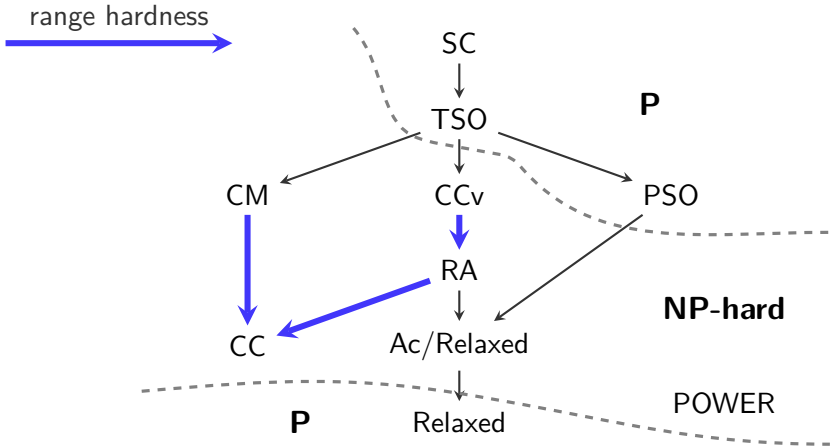
Reduction from monotone 1-in-3 SAT $\phi = \mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \cdots \wedge \mathcal{C}_m$

- No negations: $\mathcal{C}_i = x_i^1 \vee x_i^2 \vee x_i^3$
- For each clause, exactly one variable must be true

A Copy Gadget:



Fixed-Parameter Value Consistency



Takeaways

Takeaways

- Concurrency (memory) models are ubiquitous, wild, and evolving

Takeaways

- Concurrency (memory) models are ubiquitous, wild, and evolving
- Consistency checking is a fundamental problem in all these models
  - Many other problems are also intricate (reachability, robustness, . . .)

Takeaways

- Concurrency (memory) models are ubiquitous, wild, and evolving
- Consistency checking is a fundamental problem in all these models
  - Many other problems are also intricate (reachability, robustness, . . .)
- Fine-grained/parameterized complexity is
  - natural (so many parameters)
  - necessary (most problems are intractable in theory, even when poly-time)
  - fruitful (complexity highly varies wrt memory models, parameters)
  - challenging (combinatorially involved)
  - **fun!**

Takeaways

- Concurrency (memory) models are ubiquitous, wild, and evolving
- Consistency checking is a fundamental problem in all these models
  - Many other problems are also intricate (reachability, robustness, ...)
- Fine-grained/parameterized complexity is
  - natural (so many parameters)
  - necessary (most problems are intractable in theory, even when poly-time)
  - fruitful (complexity highly varies wrt memory models, parameters)
  - challenging (combinatorially involved)
  - **fun!**

THANK YOU!

Appendix

References

U. Mathur, A. Pavlogiannis, and M. Viswanathan. "The Complexity of Dynamic Data Race Prediction". In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. ACM, 2020.

R. Kulkarni, U. Mathur, and A. Pavlogiannis. "Dynamic Data-Race Detection Through the Fine-Grained Lens". In: *32nd International Conference on Concurrency Theory (CONCUR 2021)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

S. Chakraborty et al. "How Hard Is Weak-Memory Testing?" In: *Proceedings of the ACM on Programming Languages* POPL (Jan. 2024).

H. C. Tunç et al. "Optimal Reads-From Consistency Checking for C11-Style Memory Models". In: PLDI. June 2023.

L. Møldrup and A. Pavlogiannis. "AWDIT: An Optimal Weak Database Isolation Tester". In: *Proceedings of the ACM on Programming Languages* PLDI (June 2025).

T. L. Bui et al. "The Reads-from Equivalence for the TSO and PSO Memory Models". In: *Proc. ACM Program. Lang.* OOPSLA (Oct. 2021).

Axiomatic Semantics of TSO

External Relations

$$\mathsf{rfe} \triangleq \mathsf{rf} \setminus \mathsf{po}; \qquad \mathsf{moe} \triangleq \mathsf{mo} \setminus \mathsf{po}; \qquad \mathsf{fre} \triangleq \mathsf{fr} \setminus \mathsf{po}$$

Preserved Program Order

$$\mathsf{ppo} \triangleq \mathsf{po} \cap ((\mathit{Reads} \times (\mathit{Reads} \cup \mathit{Writes})) \cup (\mathit{Writes} \times \mathit{Writes})$$

TSO-Consistency

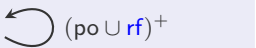
An execution graph G is **TSO-consistent** if the following conditions are met

- $(G.\mathsf{po}_x \cup G.\mathsf{rf}_x \cup G.\mathsf{mo}_x \cup G.\mathsf{fr}_x)^+$ is irreflexive (SC-per-location)
- $(G.\mathsf{ppo} \cup G.\mathsf{rfe} \cup G.\mathsf{moe} \cup G.\mathsf{fre})^+$ is irreflexive (external happens-before)

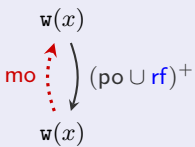
Axiomatic Semantics of Release/Acquire

An execution graph $G = (E, \mathsf{po}, \mathsf{rf}, \mathsf{mo})$ is **RA-consistent** if none of the following patterns are present

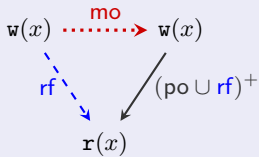
Irreflexive $(\mathsf{po} \cup \mathsf{rf})^+$



Write Coherence



Read Coherence

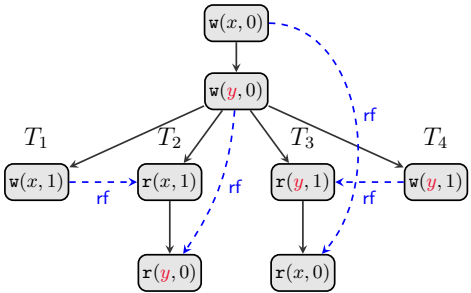


Multy-copy Atomicity

- **MCA:** “Every two writes become visible to the system in the same order”
- All threads are “equally close”

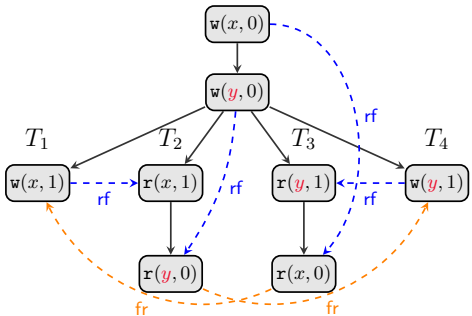
Multy-copy Atomicity

- **MCA:** "Every two writes become visible to the system in the same order"
- All threads are "equally close"
- Forbids behaviors like the following



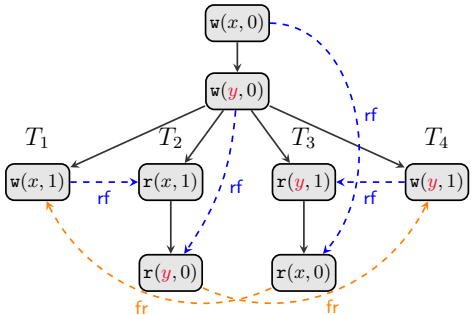
Multy-copy Atomicity

- **MCA:** "Every two writes become visible to the system in the same order"
- All threads are "equally close"
- Forbids behaviors like the following



Multy-copy Atomicity

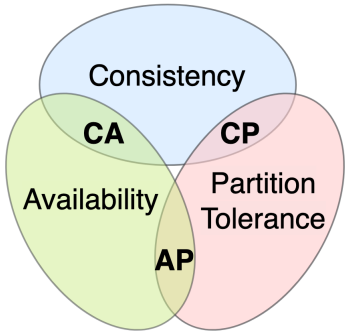
- **MCA:** "Every two writes become visible to the system in the same order"
- All threads are "equally close"
- Forbids behaviors like the following



- T_2 sees $\mathtt{w}(x,1) \to \mathtt{w}(y,1)$
- T_3 sees $\mathtt{w}(y,1) \to \mathtt{w}(x,1)$

Database Isolation

Databases are Highly Distributed Objects



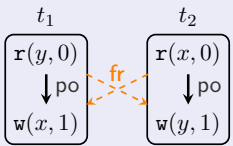
Many database implementations forego strong data consistency (**AP**)

- Also for efficiency reasons (distributed locks are expensive → OCC)

Consistency (Isolation) Anomalies

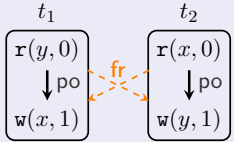
Consistency (Isolation) Anomalies

Write Skew

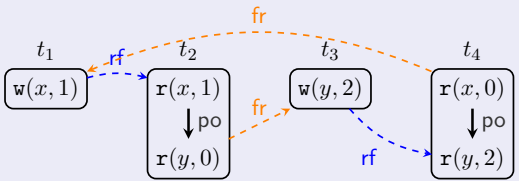


Consistency (Isolation) Anomalies

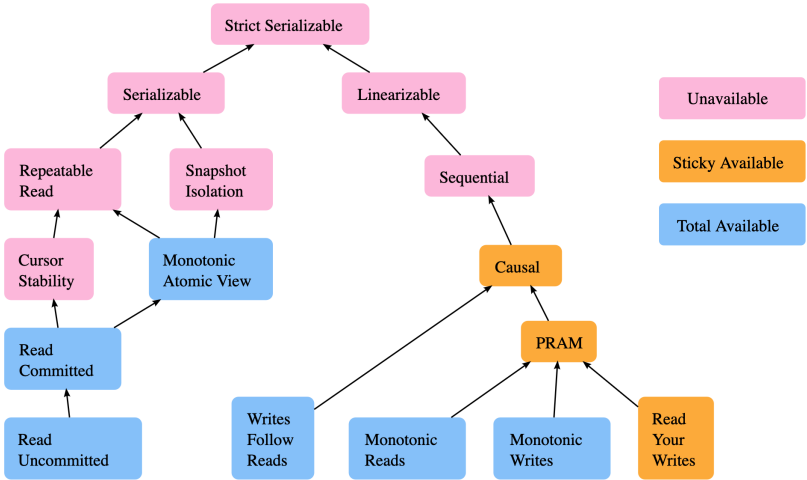
Write Skew



Long Fork

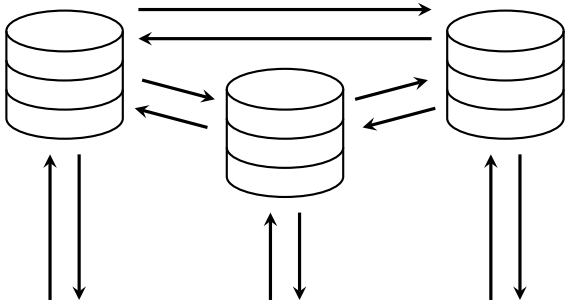


A Zoo of Isolation Levels



Source: `http://jepsen.io`

Black-Box Testing



1. Send transactions to database cluster
2. Collect history H
3. Does H satisfy claimed isolation level $\mathcal{I}$?

Testing tool

Isolation Levels as Graph Predicates

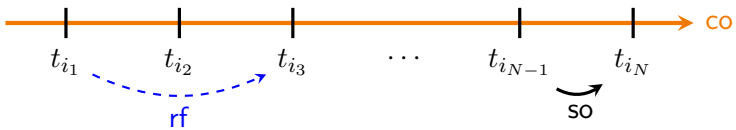
Definition of isolation level $\mathcal{I}$:

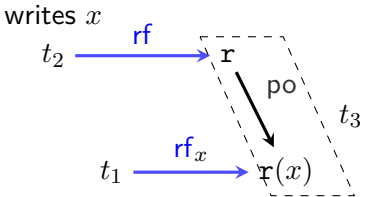
Goal Find total *commit order* co of all transactions extending $(\mathsf{so} \cup \mathsf{rf})^+$ that satisfies the respective isolation-level predicate

Isolation Levels as Graph Predicates

Definition of isolation level $\mathcal{I}$:

Goal Find total *commit order* co of all transactions extending $(\mathsf{so} \cup \mathsf{rf})^+$ that satisfies the respective isolation-level predicate



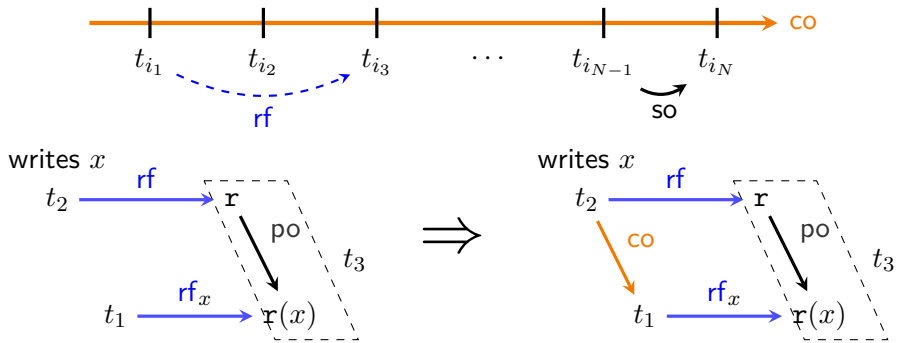


Read Committed (RC)

Isolation Levels as Graph Predicates

Definition of isolation level $\mathcal{I}$:

Goal Find total *commit order* co of all transactions extending $(\text{so} \cup \text{rf})^+$ that satisfies the respective isolation-level predicate



Read Committed (RC)

How Fast Can We Test for Weak Isolation?

How Fast Can We Test for Weak Isolation?

AWDIT (A Weak Database Isolation Tester)

Isolation level	RC	RA	CC
Running time	$O(n\sqrt{n})$	$O(n\sqrt{n})$	$O(nk)$

for n operations and k sessions

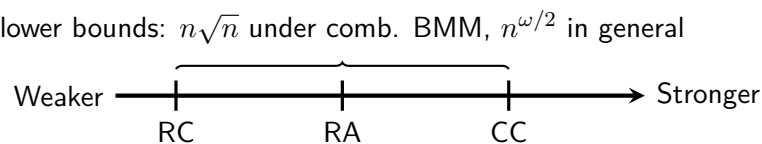
How Fast Can We Test for Weak Isolation?

AWDIT (A Weak Database Isolation Tester)

Isolation level	RC	RA	CC
Running time	$O(n\sqrt{n})$	$O(n\sqrt{n})$	$O(nk)$

for n operations and k sessions

AWDIT is (essentially) *optimal*:



AWDIT Experiments

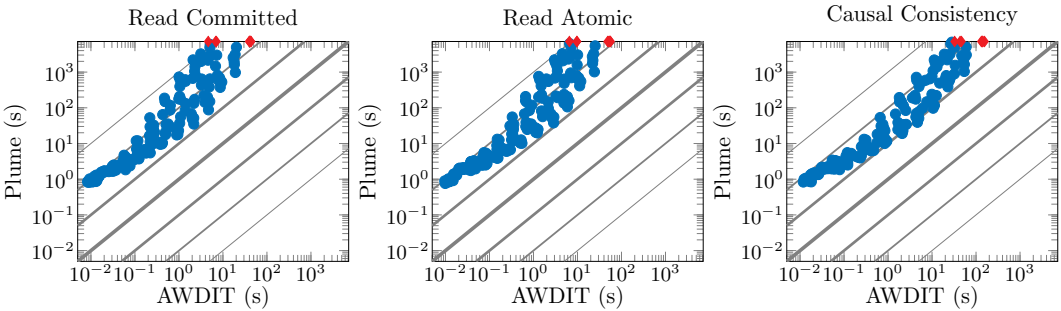


Table: Avg. speedup on largest ∼20%

Isolation level	RC	RA	CC
Avg. speedup	245×	193×	62×