TASE
2021

MoCA: Dynamic Verification of C/C++11 Concurrency over Multi-copy Atomics

Sanjana Singh, Divyanjali Sharma, Subodh Sharma
Indian Institute of Technology Delhi

Memory models

- *Interleaving*
- varying degrees of *Reordering*

Memory models

- *Interleaving*
- varying degrees of *Reordering*

a:=y (1)	c:=x (1)
x:=1	y:=1
b:=y (0)	d:=x (0)

sequentially consistent
- interleaving
- no reordering

Memory models

- *Interleaving*
- varying degrees of *Reordering*

a:=y (1)	c:=x (1)
x:=1	y:=1
b:=y (0)	d:=x (0)

sequentially consistent
- interleaving
- no reordering

a:=y (1)	c:=x (1)
x:=1	y:=1
b:=y (0)	d:=x (0)

x86's TSO
- interleaving
- WR reordering

Memory models

- *Interleaving*
- varying degrees of *Reordering*

a:=y (1)	c:=x (1)
x:=1	y:=1
b:=y (0)	d:=x (0)

sequentially consistent
- interleaving
- no reordering

a:=y (1)	c:=x (1)
x:=1	y:=1
b:=y (0)	d:=x (0)

x86's TSO
- interleaving
- WR reordering

a:=y (1)	c:=x (1)
x:=1	y:=1
b:=y (0)	d:=x (0)

ARM
- interleaving
- all coherent reordering

C11 memory model

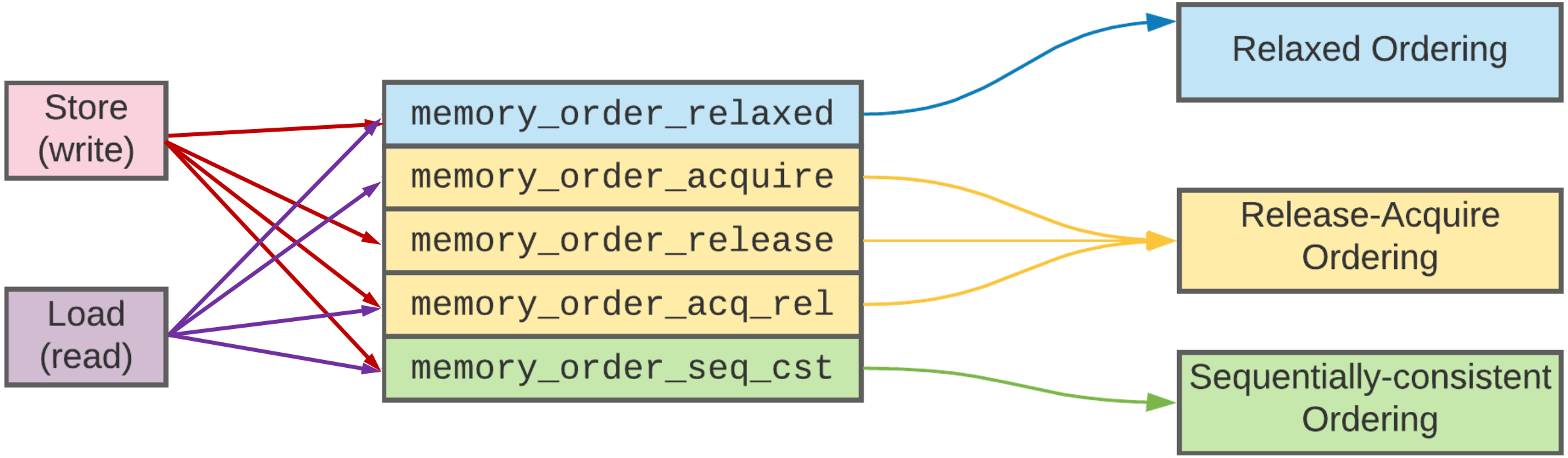
International
Organization for
Standardization

ISO

C++ 2011

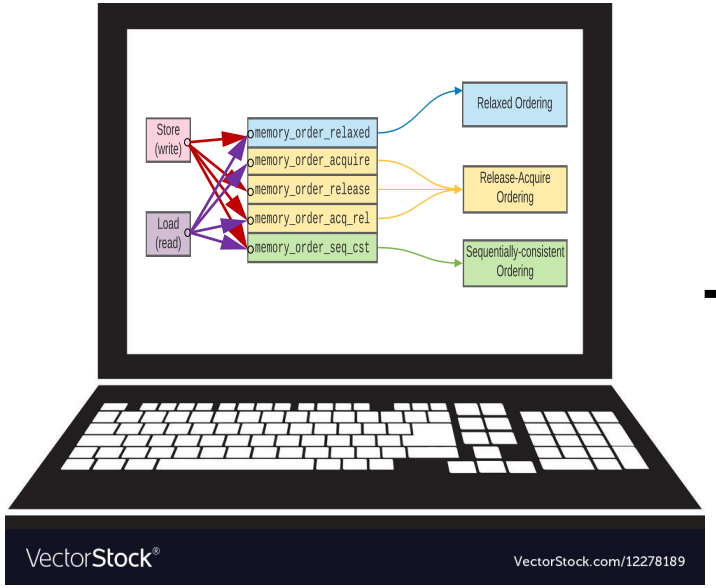
C11 memory model

International Organization for Standardization



https://www.cooldoger.com/2020/09

C11 memory model

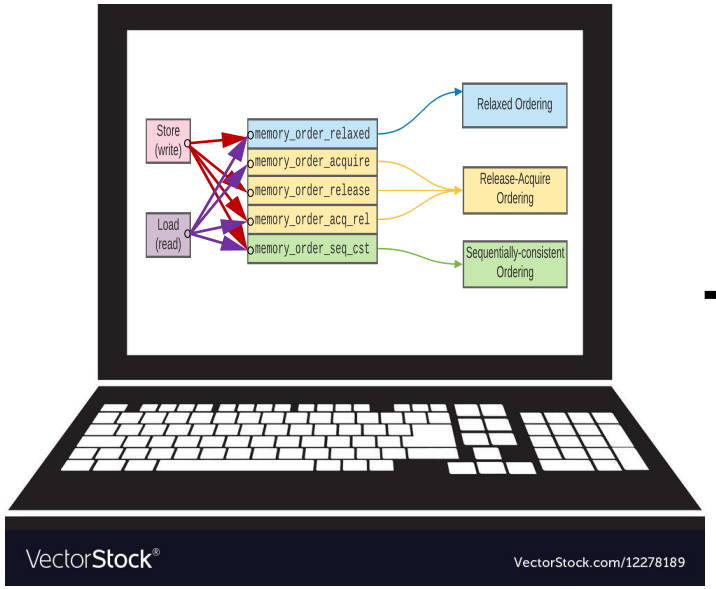


C/C++
Compiler

System
Architecture

$x{:=}1$	$y{:=}1$
barrier	*barrier*
$\mathrm{b}{:=}y$ (0)	$\mathrm{d}{:=}x$ (0)

C11 memory model



C/C++ Compiler

System Architecture

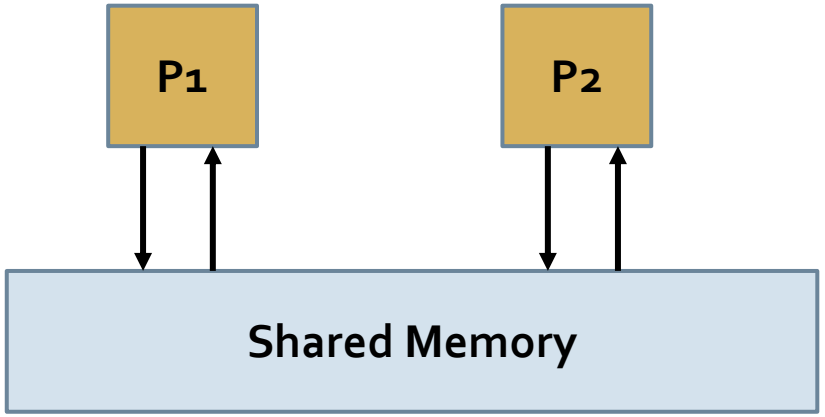
Axiomatic reasoning

reads-from relation (*rf*)
modification-order (*mo*)
Synchronizations (*sw*, *dob*)

$x{:=}1$	$y{:=}1$
barrier	*barrier*
$b{:=}y$ (0)	$d{:=}x$ (0)

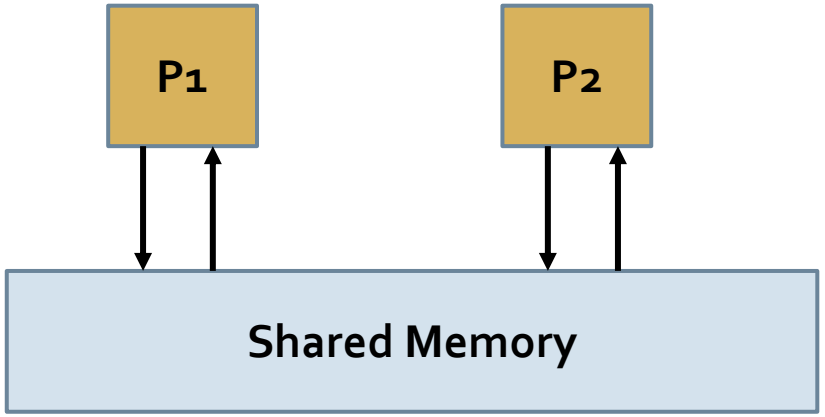
Multi-copy Atomic model

- a single abstract view of shared memory



Multi-copy Atomic model

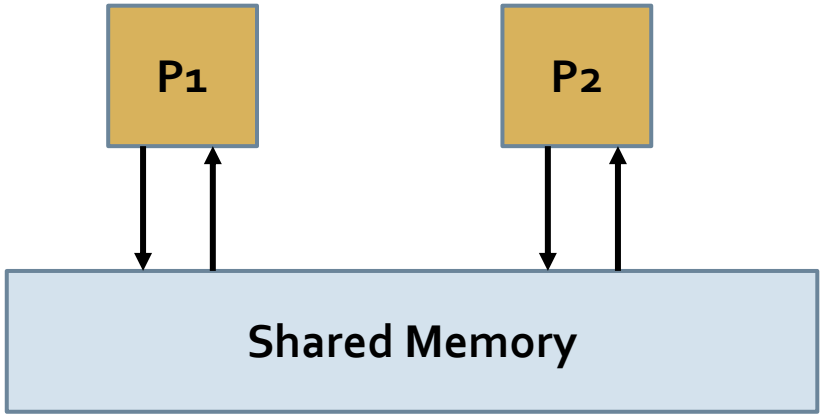
- a single abstract view of shared memory



- behaviors can be explained solely through *interleaving* and *reordering*

Multi-copy Atomic model

- a single abstract view of shared memory



- behaviors can be explained solely through *interleaving* and *reordering*

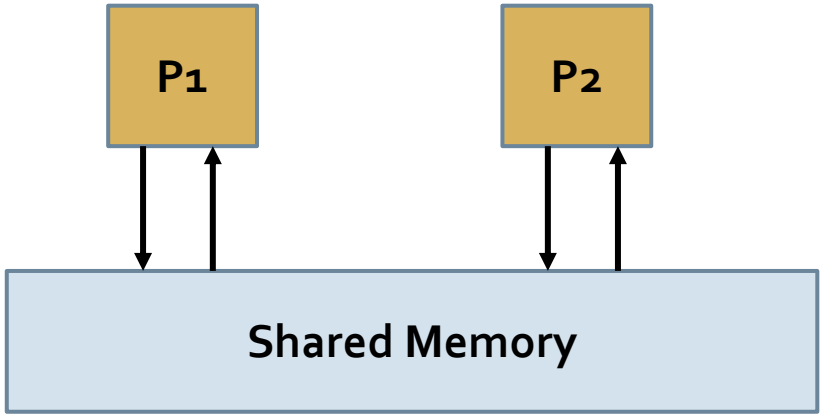
ARM®
version 8 and later

α

x86

Multi-copy Atomic model

- a single abstract view of shared memory



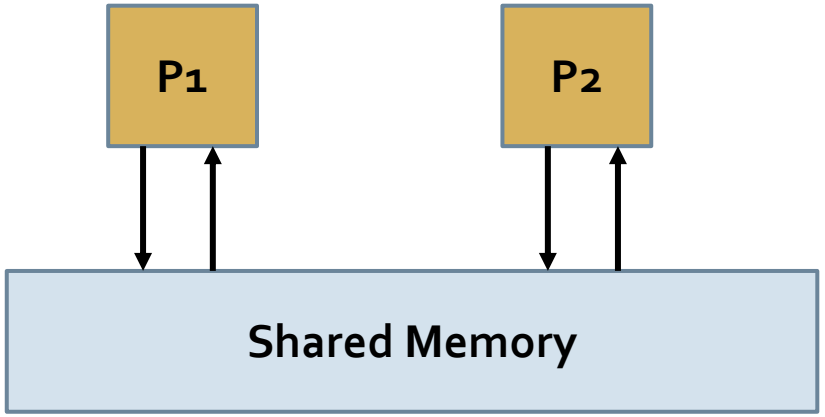
- behaviors can be explained solely through *interleaving* and *reordering*

T_1	T_2	T_3	T_4
$x:=1$	if $(x=1)$	$y:=1$	if $(y=1)$
	a:=y		b:=x

(IRIW)

Multi-copy Atomic model

- a single abstract view of shared memory



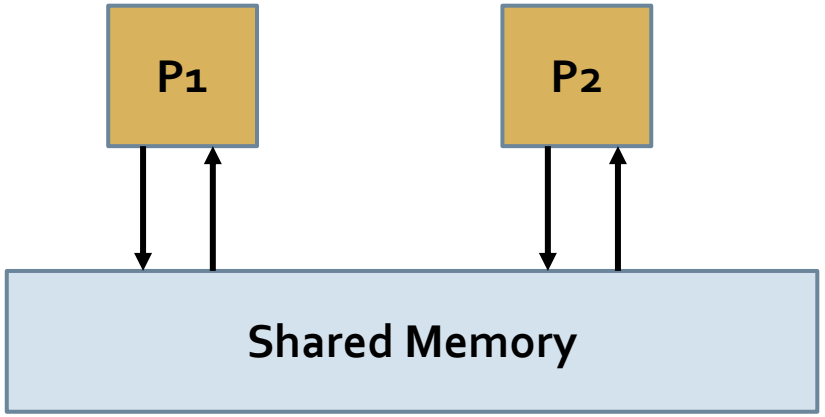
- behaviors can be explained solely through interleaving and reordering

T_1	T_2	T_3	T_4
$x:=1$	if $(x=1)$	$y:=1$	if $(y=1)$
	a:=y (0)		b:=x (0)

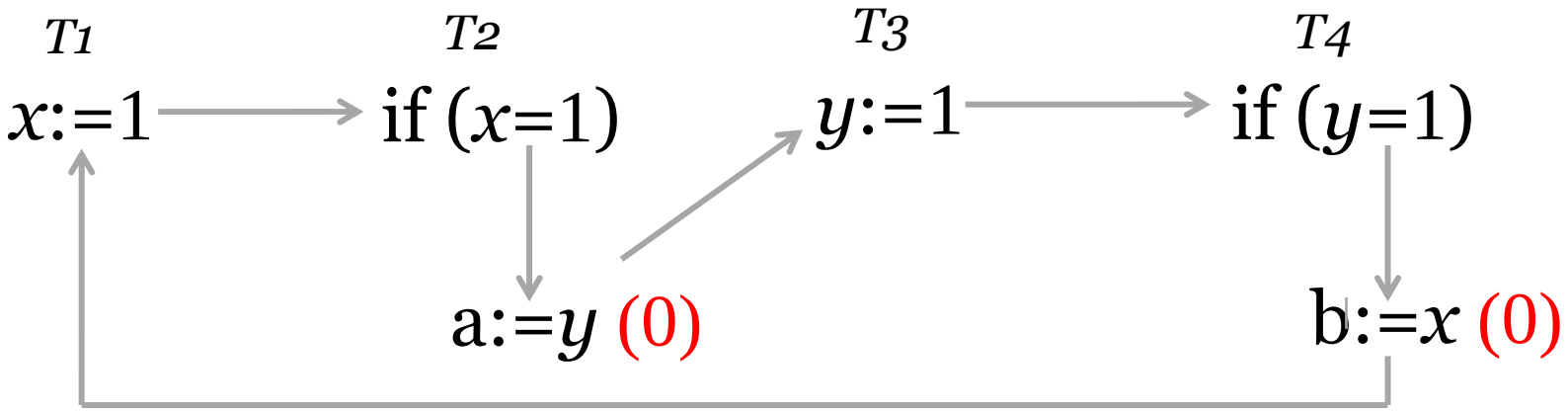
(IRIW)

Multi-copy Atomic model

- a single abstract view of shared memory



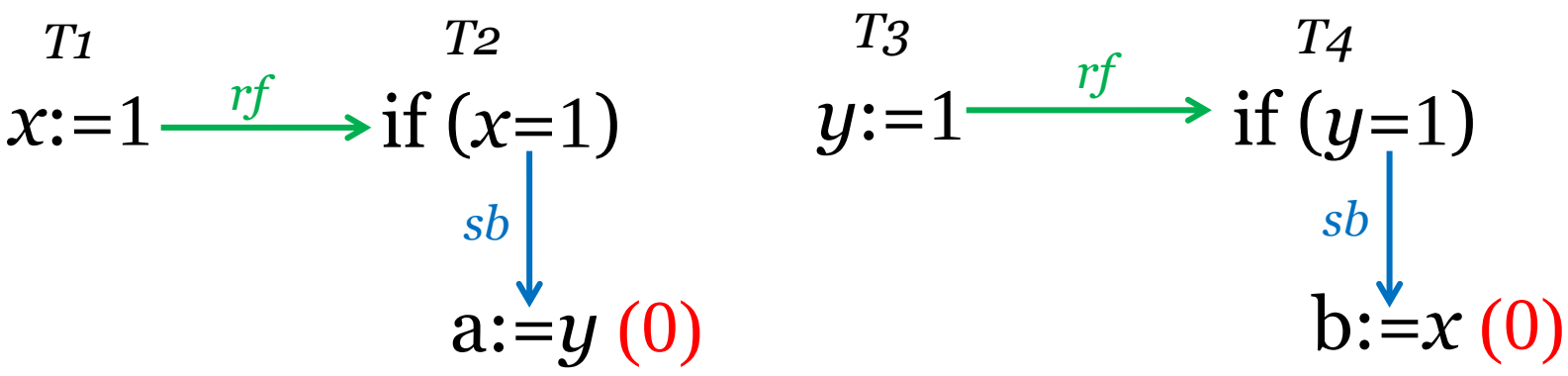
- behaviors can be explained solely through interleaving and reordering



(IRIW)

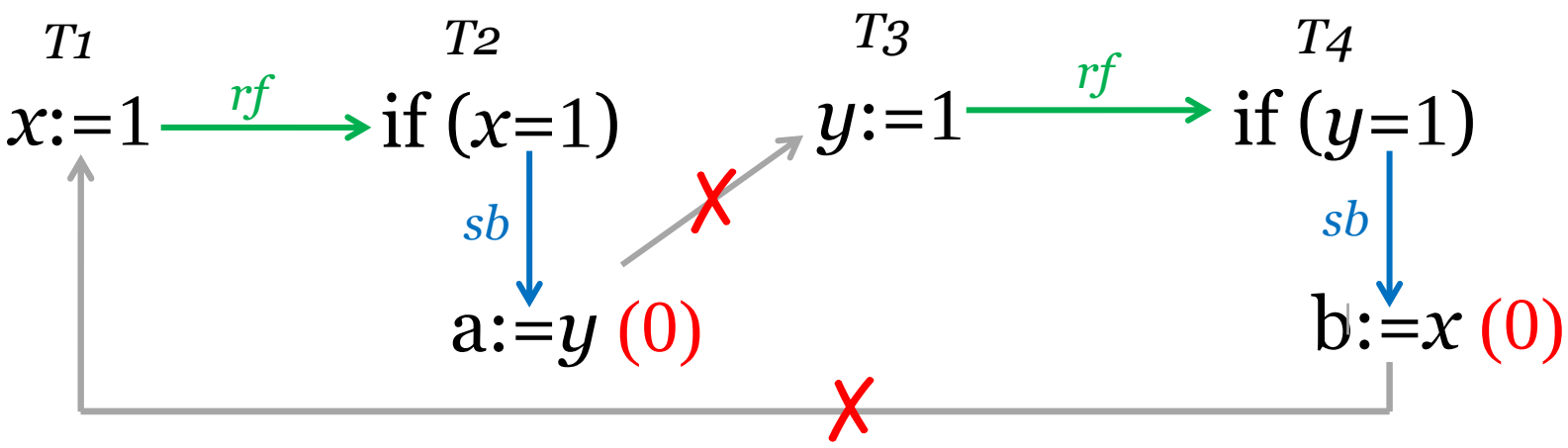
C11 allows *non*-MCA

(IRIW) allowed under C11



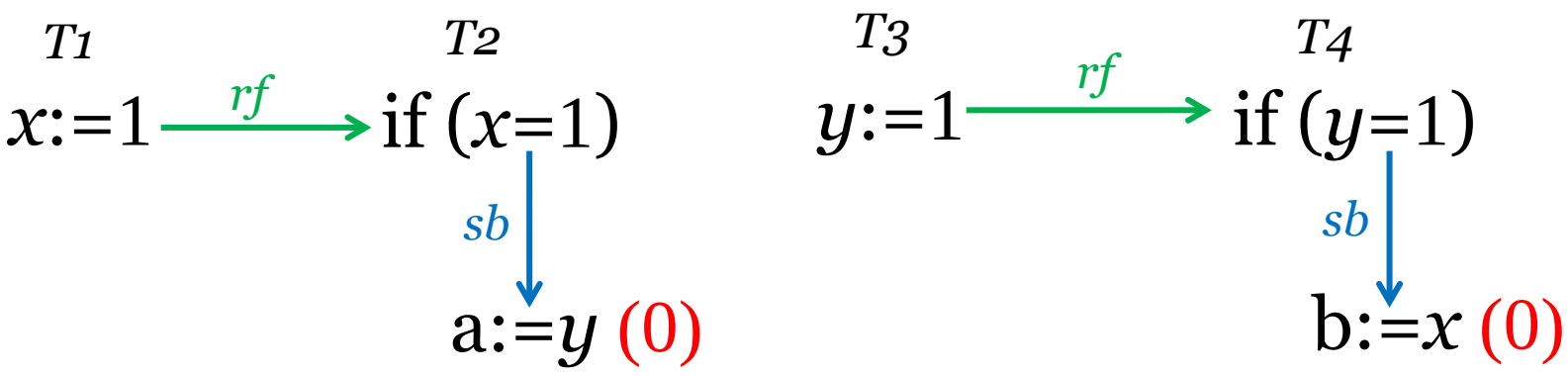
C11 allows *non*-MCA

(IRIW) allowed under C11



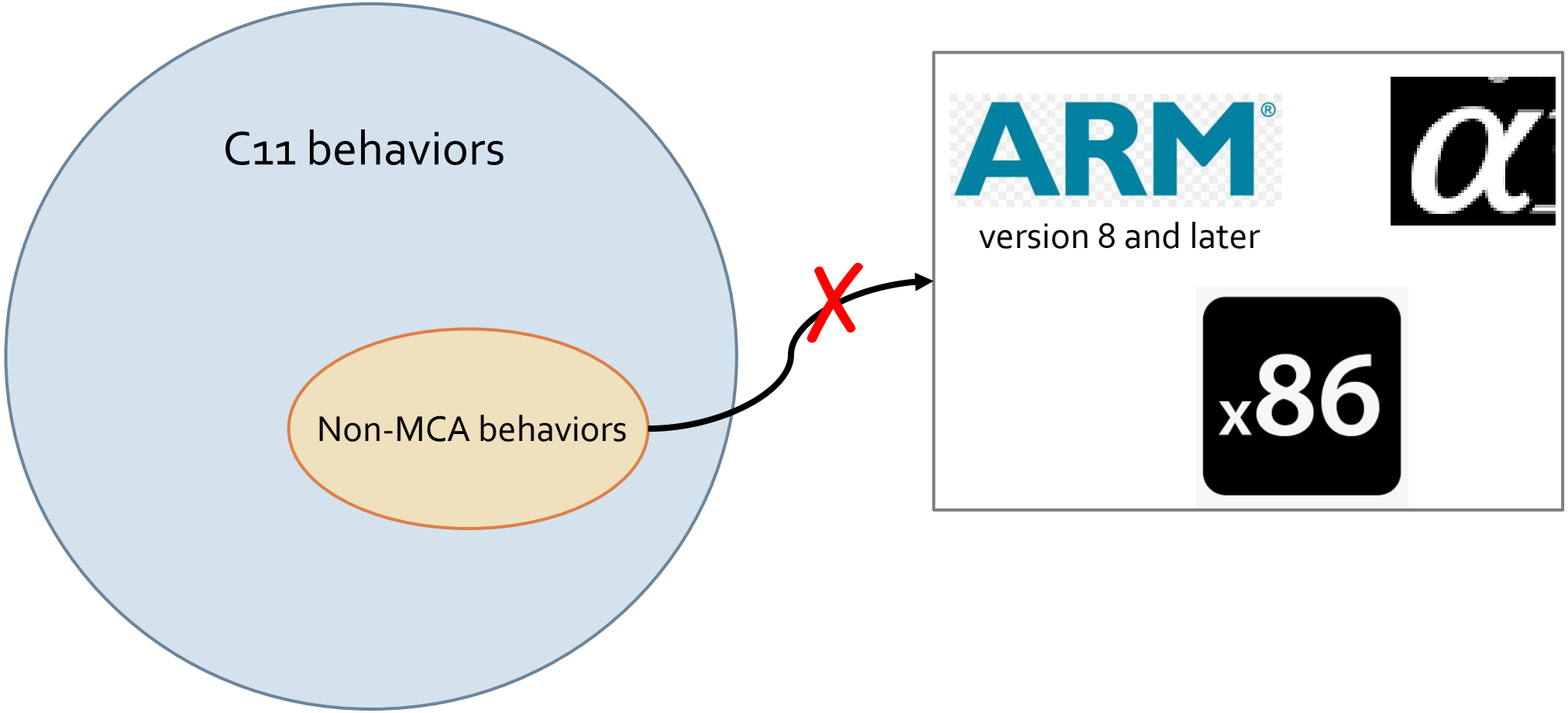
C11 allows *non*-MCA

(IRIW) allowed under C11

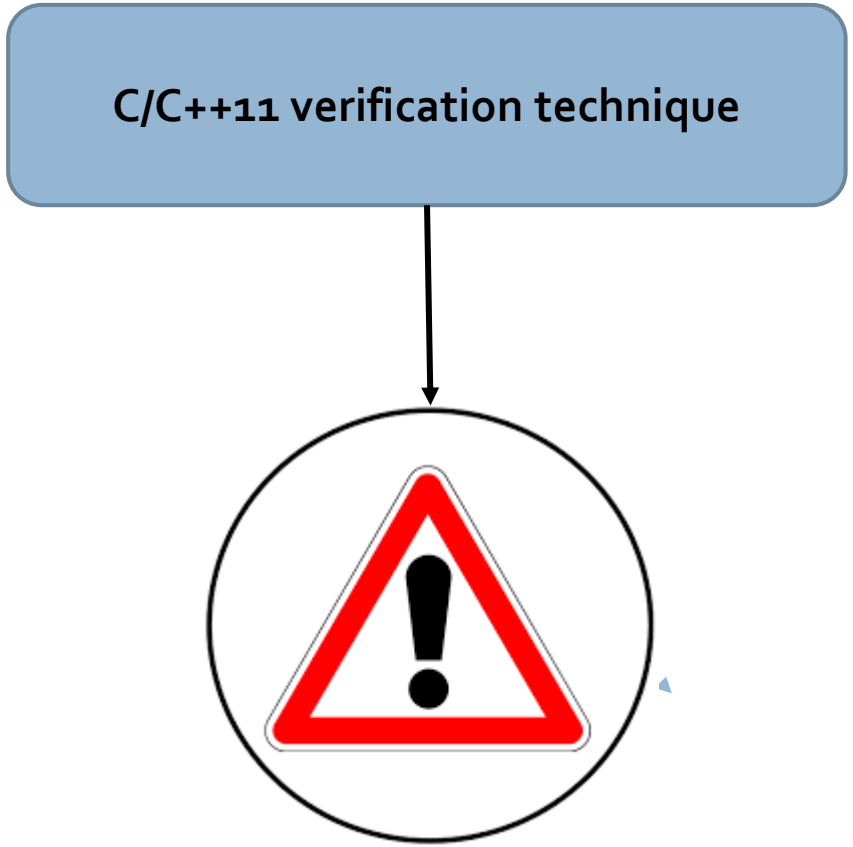


C11 over MCA

C11 over MCA -- *why?*



C11 over MCA -- *why?*



safety property violation

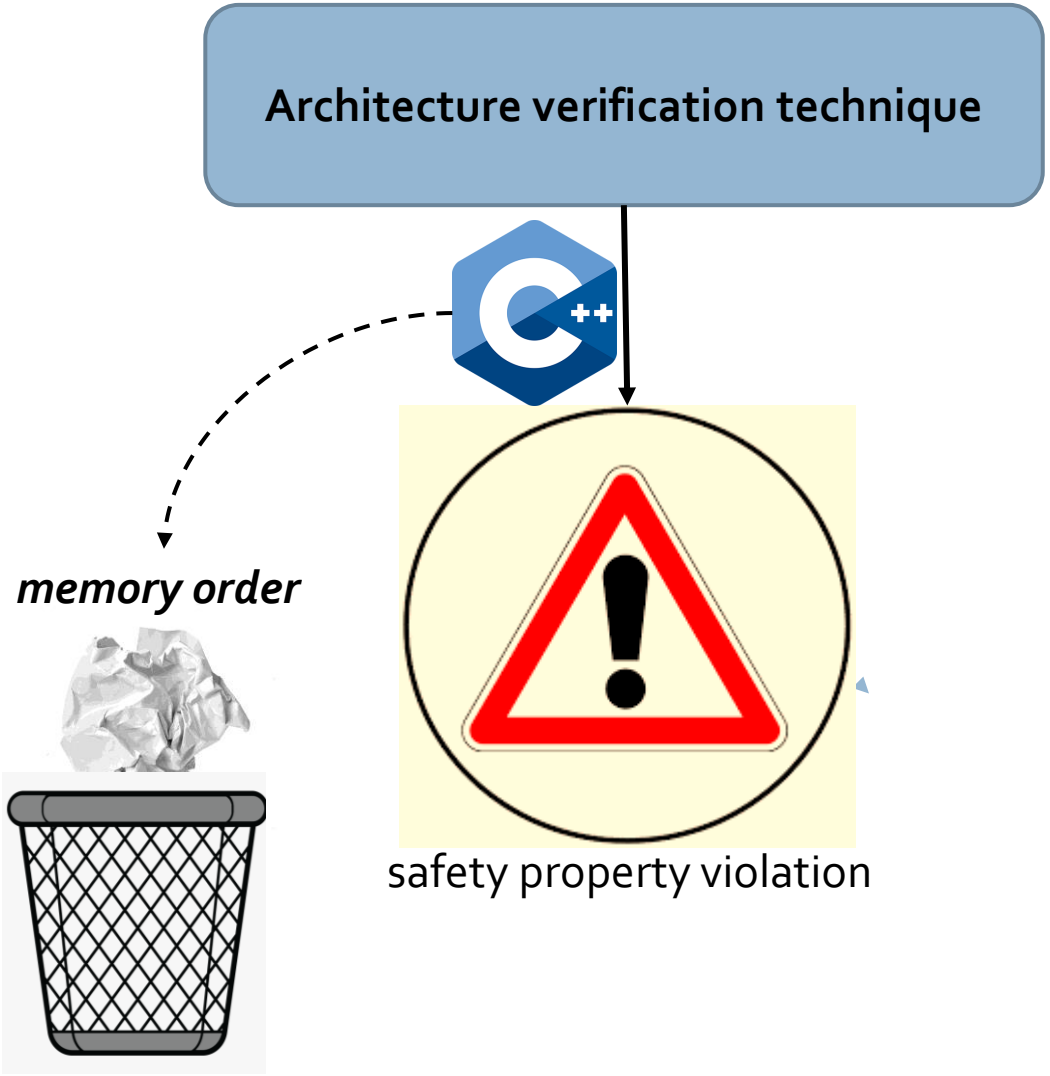
C11 over MCA -- *why?*

C/C++11 verification technique

safety property violation

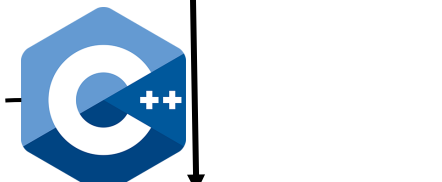
false positive?

C11 over MCA -- *why?*

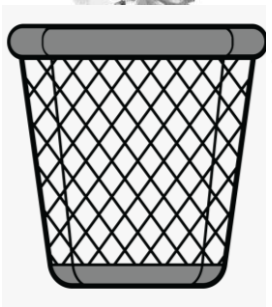


C11 over MCA -- *why?*

Architecture verification technique



memory order



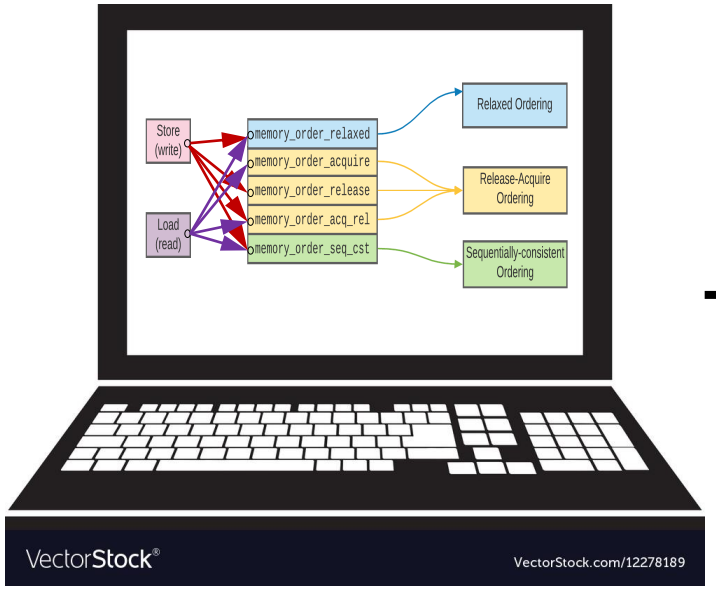
safety property violation

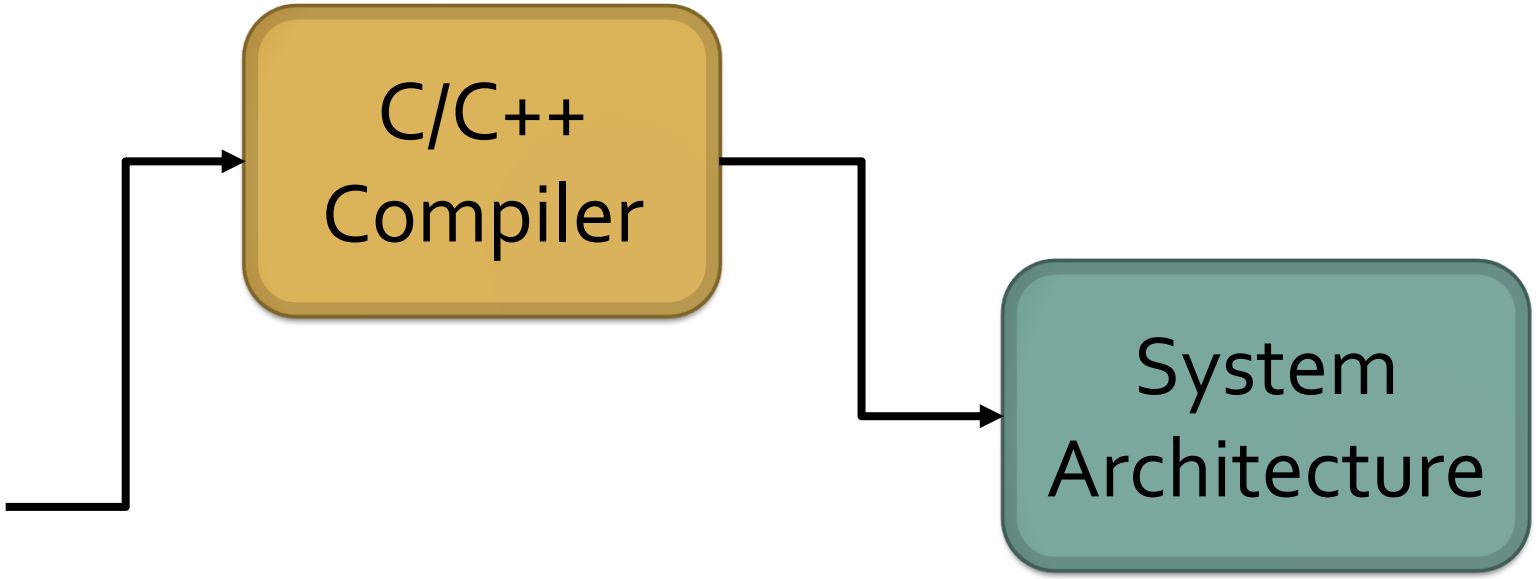
false positive?

false negative?

C11 over MCA -- *why?*

Given a reordering specification





What behaviors can manifest on the underlying architecture?

C11 over MCA -- *why?*

Architecture verification technique

C/C++11 verification technique

false negative?
false positive?

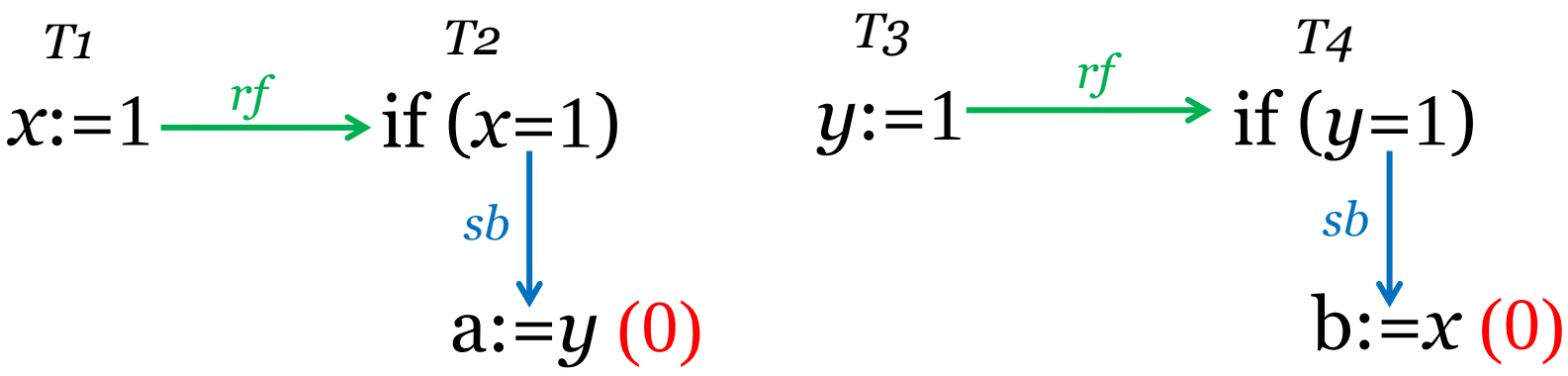
safety property violation

safety property violation

false positive?

C11 over MCA

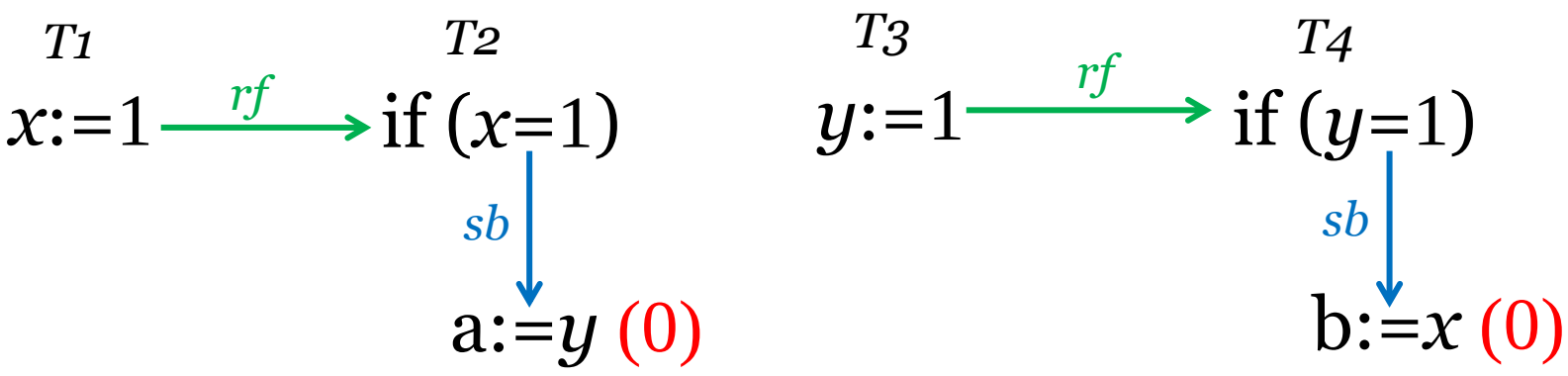
(IRIW) allowed under C11



TASK: Modify event relations to restrict C11 behaviors to MCA memory model

C11 over MCA

(IRIW) allowed under C/C++11



TASK: Modify event relations to restrict C11 behaviors to MCA memory model

[Colvin and Smith, FM 2018].

TASK: Modify event relations and rules to restrict C11 behaviors to MCA memory model

PART I: Define an appropriate happens-before relation

PART II: Define appropriate coherence rules to ensure coherence *wrt* C11

TASK: Modify event relations and rules to restrict C11 behaviors to MCA memory model

PART I: Define an appropriate happens-before relation

For C11 behaviors that can be justified via interleaving and reordering

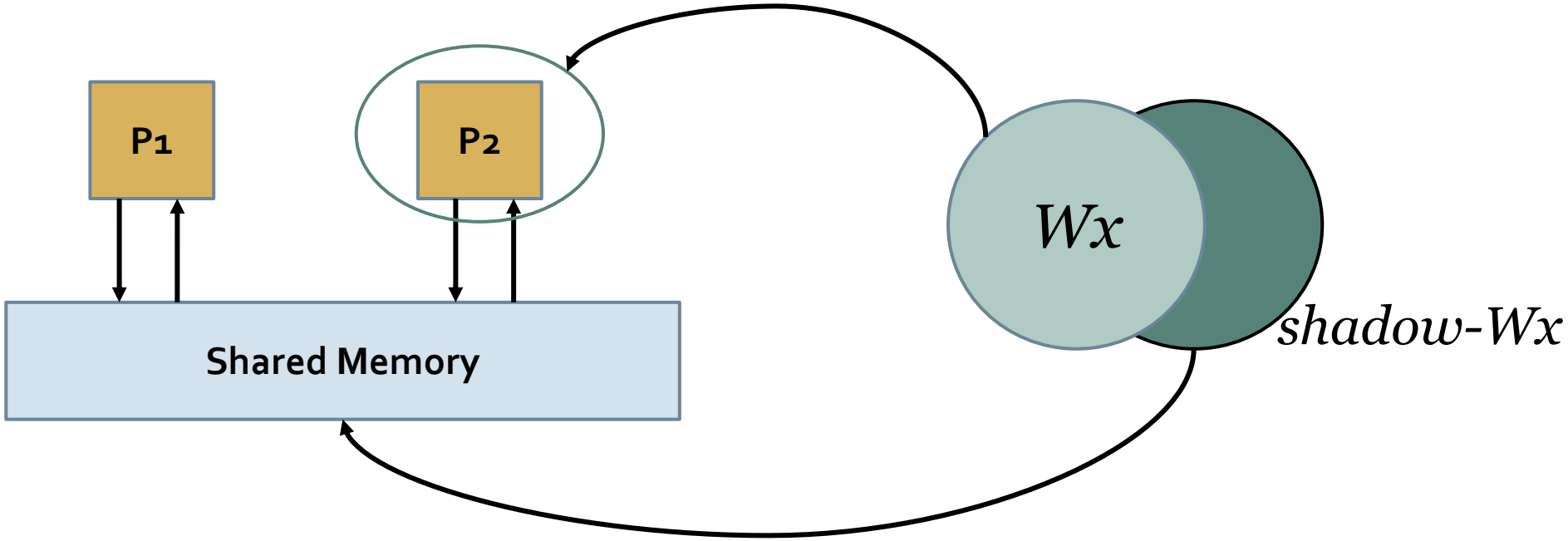
TASK: Modify event relations and rules to restrict C11 behaviors to MCA memory model

PART I: Define an appropriate happens-before relation

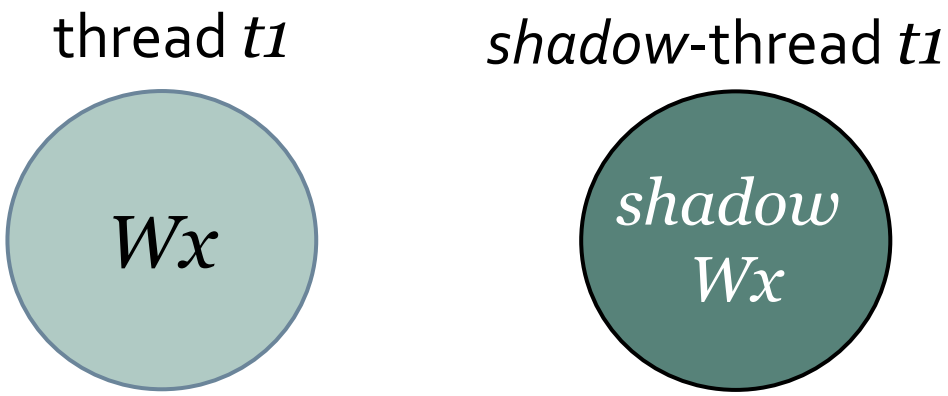
For C11 behaviors that can be justified via interleaving and reordering

through interleaving

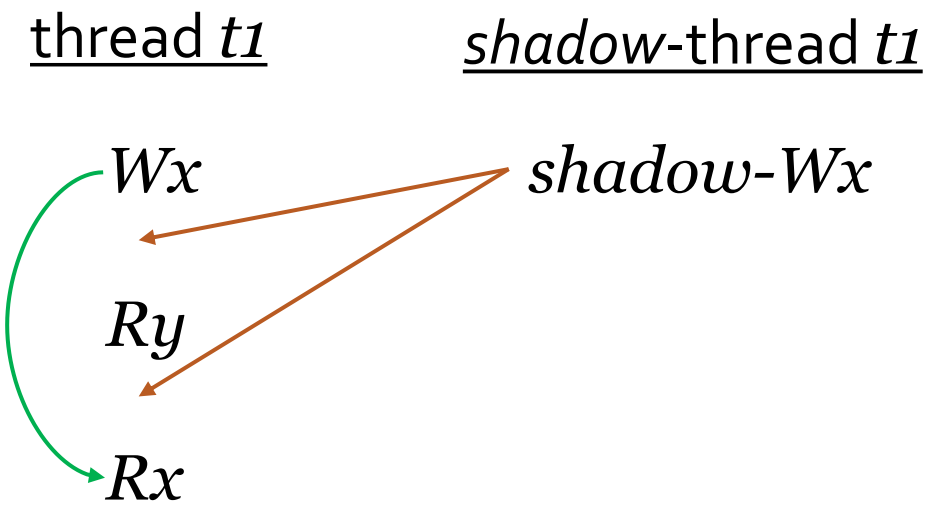
Reordering *through* Interleaving



Reordering *through* Interleaving

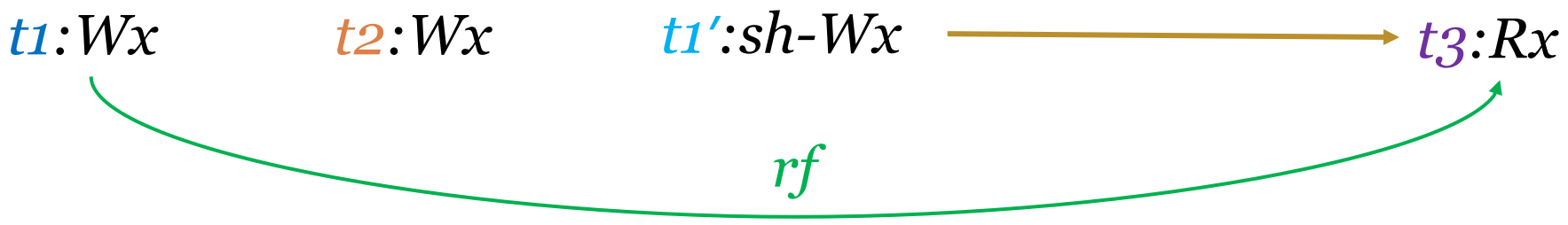


Reordering *through* Interleaving



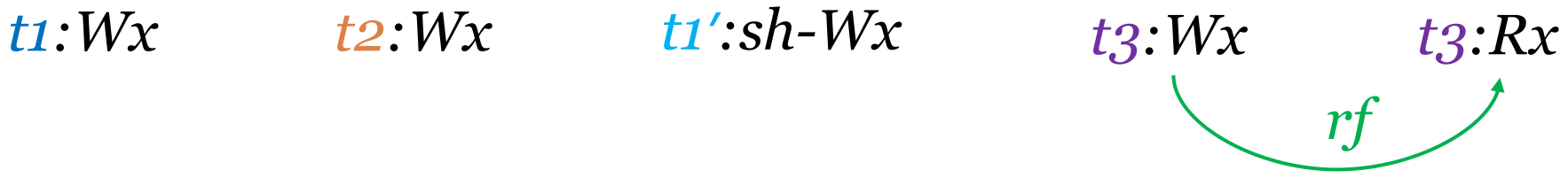
C11 over MCA: axiomatic definition

reads-from (*rf*)

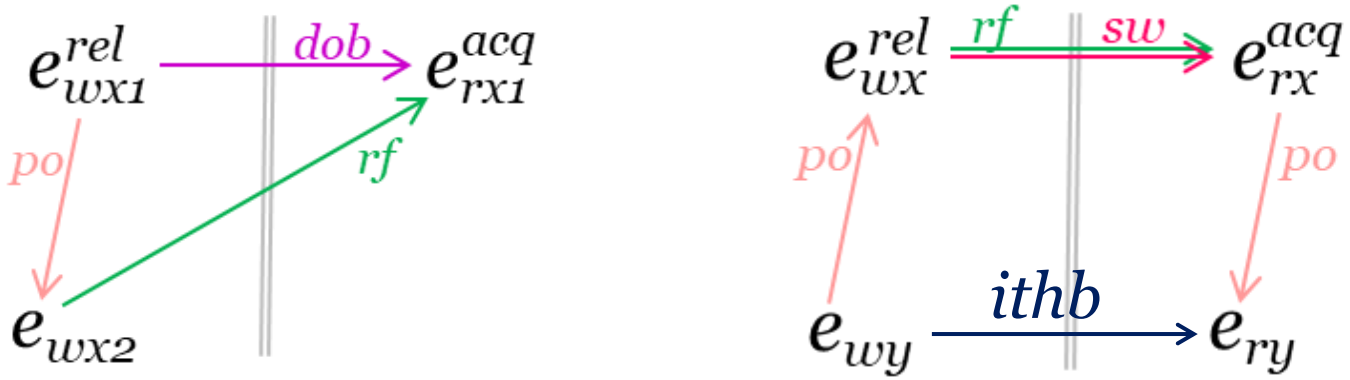


C11 over MCA: axiomatic definition

reads-from (*rf*)



C11 over MCA: axiomatic definition

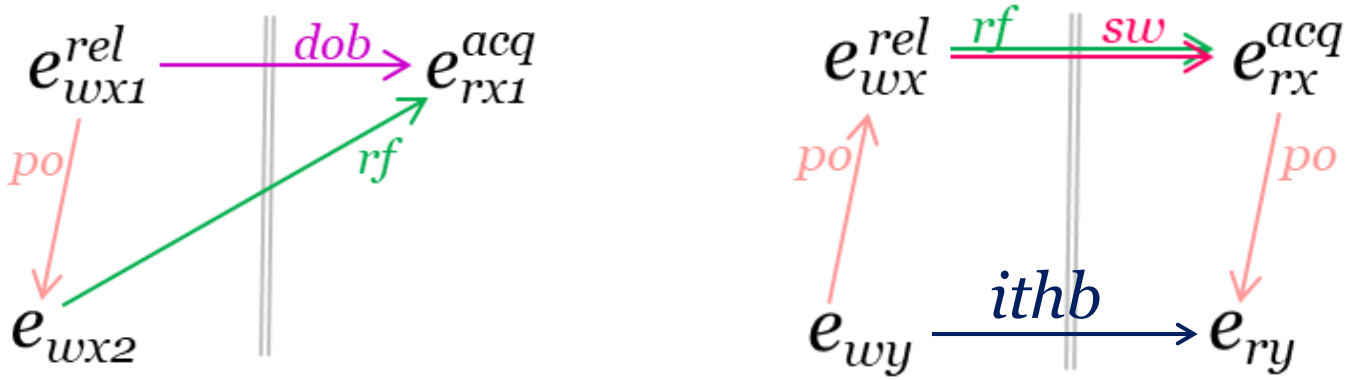


$$e' \rightarrow^{\mathbf{hb}}_{\tau} e \text{ if}(e' \rightarrow^{\mathbf{po}}_{\tau} e \vee e' \rightarrow^{\mathbf{sw}}_{\tau} e \vee e' \rightarrow^{\mathbf{dob}}_{\tau} e\ )^{+}$$

C11 over MCA: axiomatic definition

Theorem 1. $\rightarrow_{\tau}^{hb}$ *for any sequence* τ *is* <u>*valid*</u>

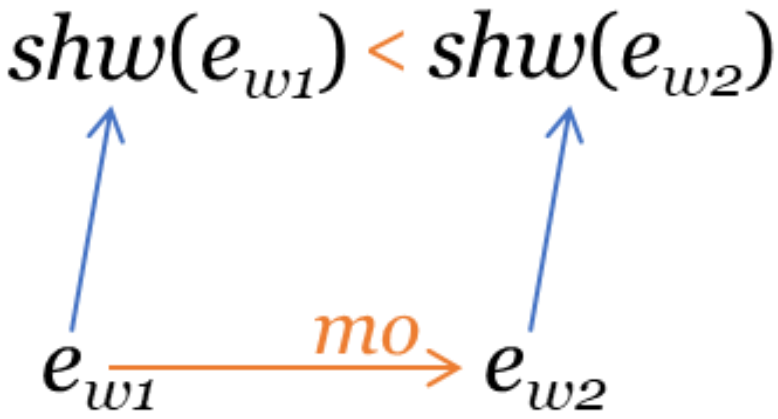
[Abdulla et al., POPL 2014]



$$e' \rightarrow_{\tau}^{\mathbf{hb}} e \text{ if}(e' \rightarrow_{\tau}^{\mathbf{po}} e \vee e' \rightarrow_{\tau}^{\mathbf{sw}} e \vee e' \rightarrow_{\tau}^{\mathbf{dob}} e\ )^{+}$$

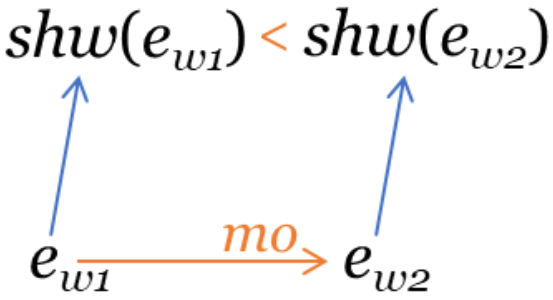
C11 over MCA: axiomatic definition

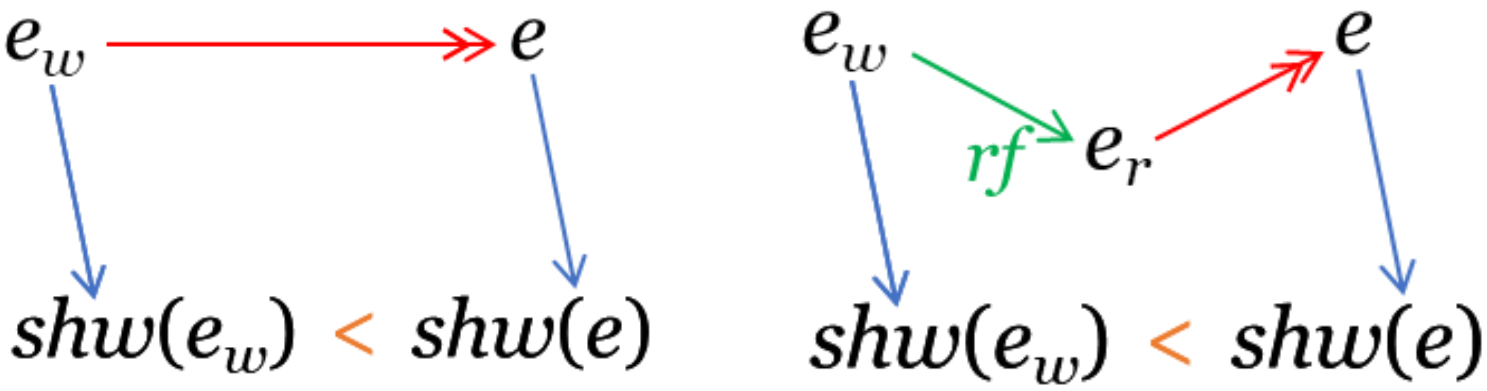
PART I: Define an appropriate happens-before relation
PART II: Define appropriate coherence rules to ensure coherence *wrt* C11



C11 over MCA: axiomatic definition

PART I: Define an appropriate happens-before relation
PART II: Define appropriate coherence rules to ensure coherence *wrt* C11

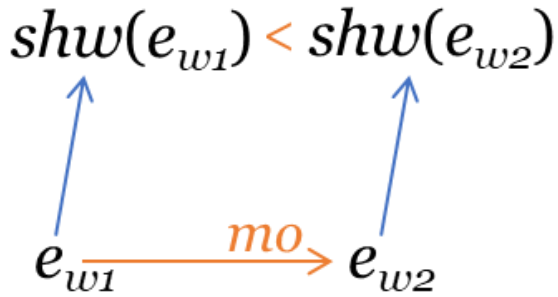


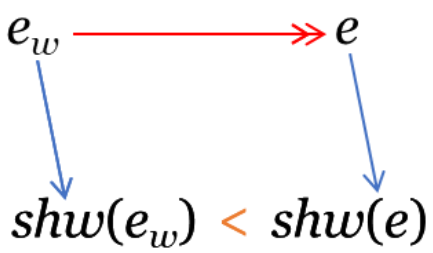


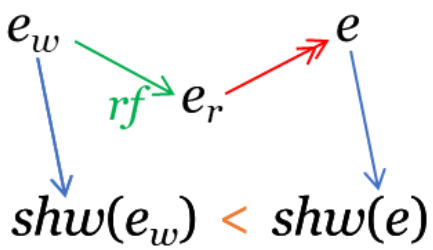
(shmo1)

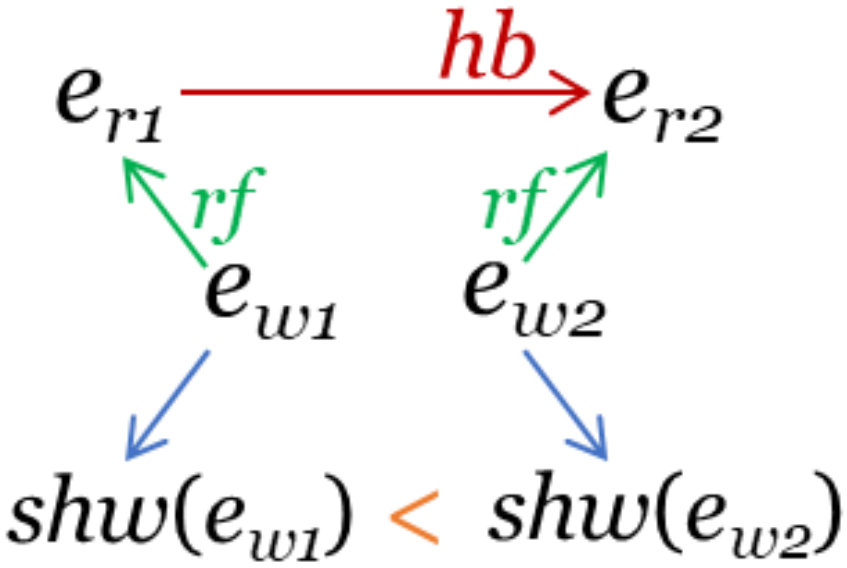
C11 over MCA: axiomatic definition

PART I: Define an appropriate happens-before relation
PART II: Define appropriate coherence rules to ensure coherence *wrt* C11





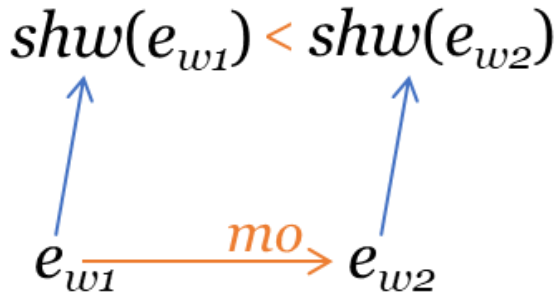


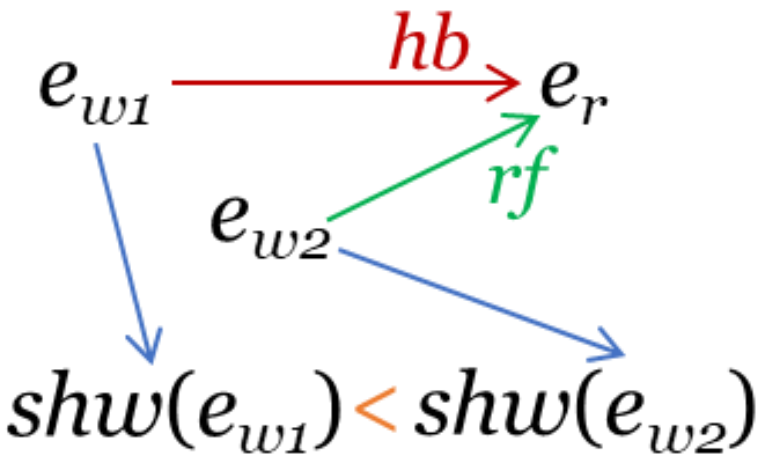


(shmo2)

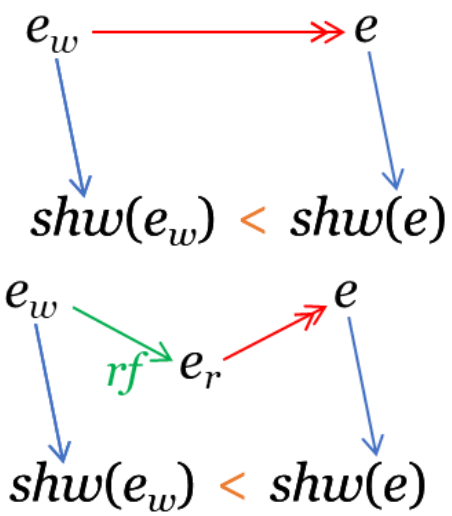
C11 over MCA: axiomatic definition

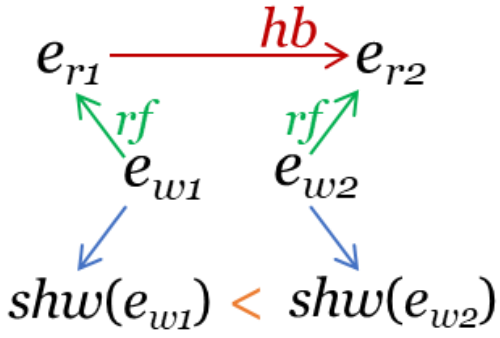
PART I: Define an appropriate happens-before relation
PART II: Define appropriate coherence rules to ensure coherence *wrt* C11





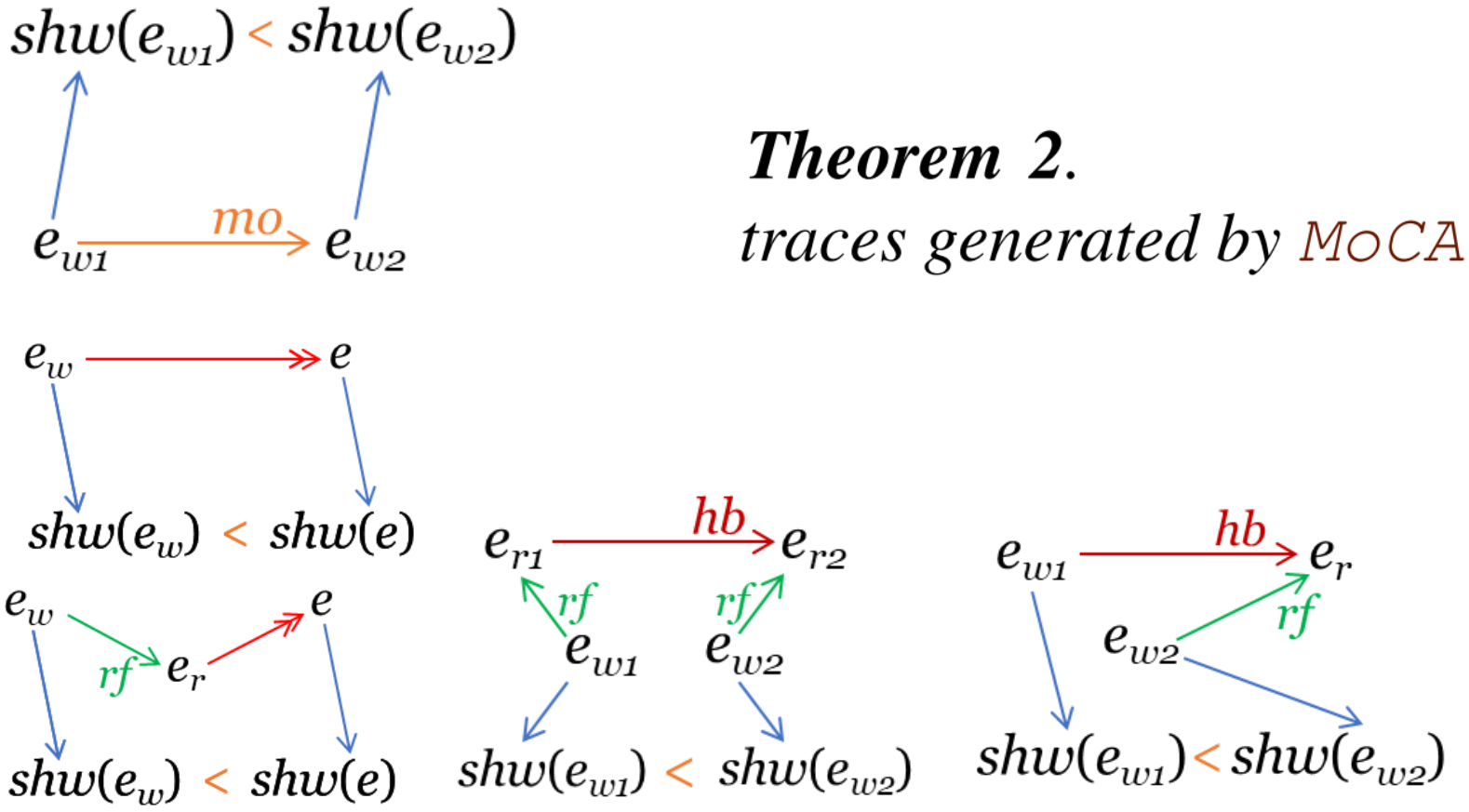
(shmo3)





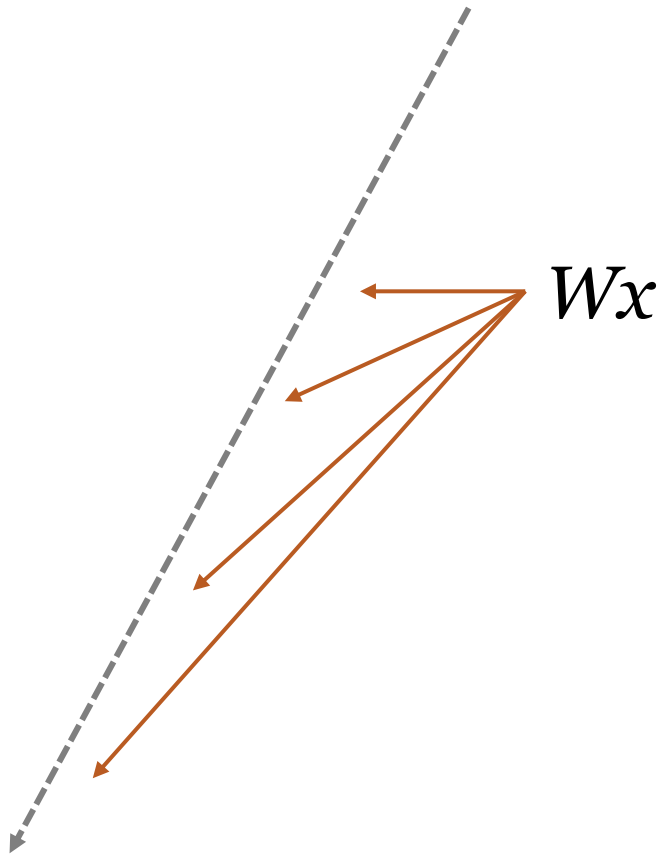
C11 over MCA: axiomatic definition

PART I: Define an appropriate happens-before relation
PART II: Define appropriate coherence rules to ensure coherence *wrt* C11

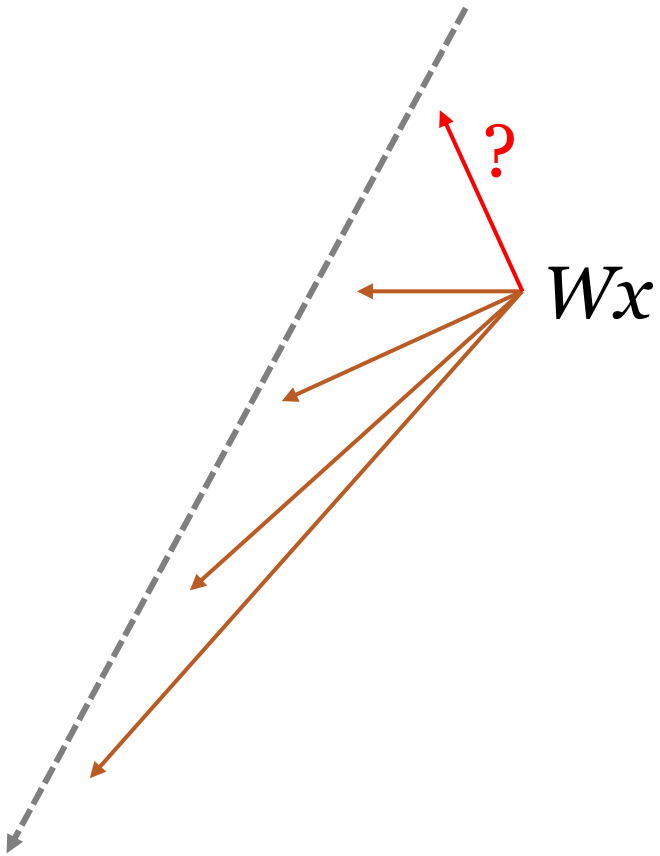


Theorem 2.
traces generated by MoCA *are coherent under C11*

Reordering *through* Interleaving

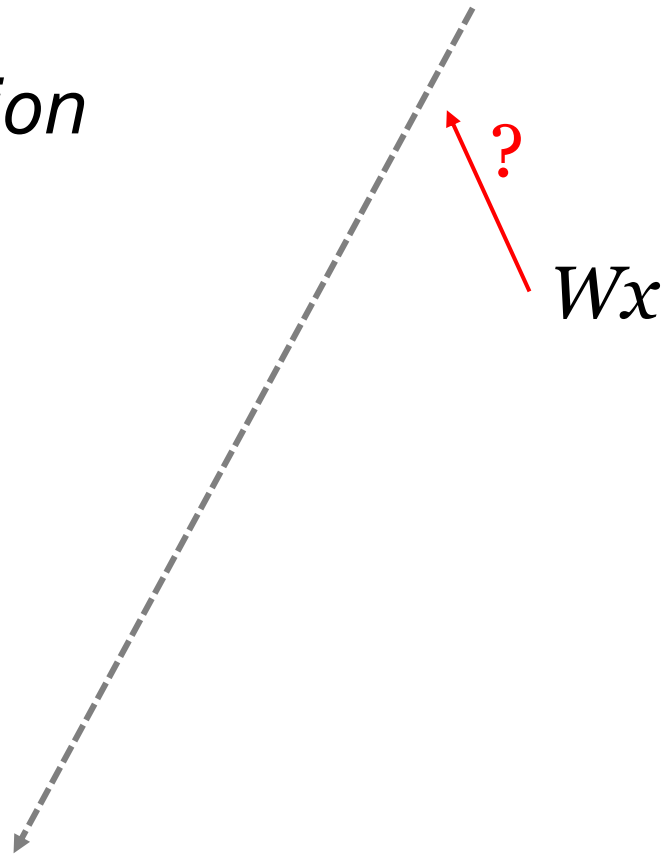


Reordering *through* Interleaving



Reordering *through* Interleaving

Early-write Transformation



Reordering *through* Interleaving

Early-write Transformation

<u>thread *ti*</u>

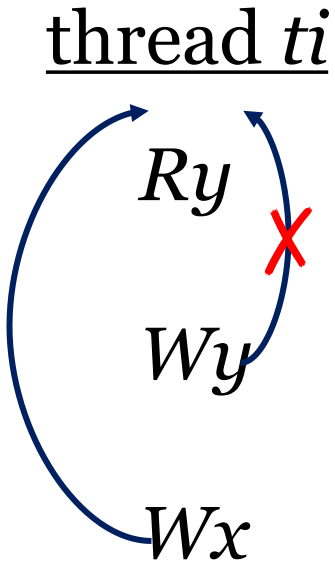
Ry

Wy

Wx

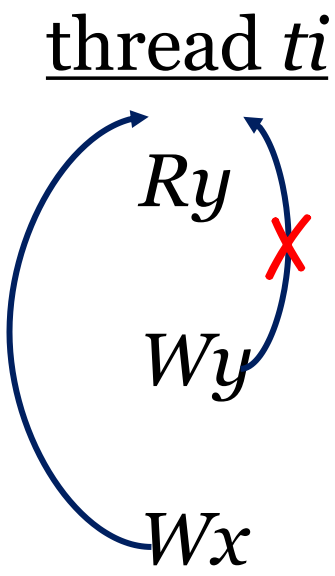
Reordering *through* Interleaving

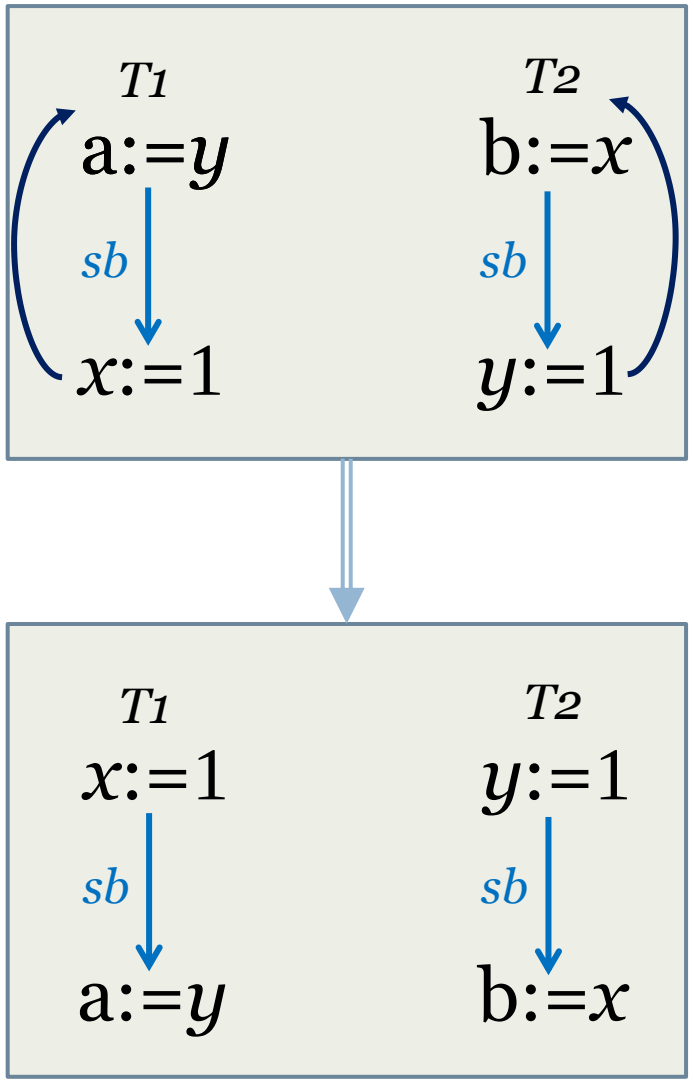
Early-write Transformation



Reordering *through* Interleaving

Early-write Transformation



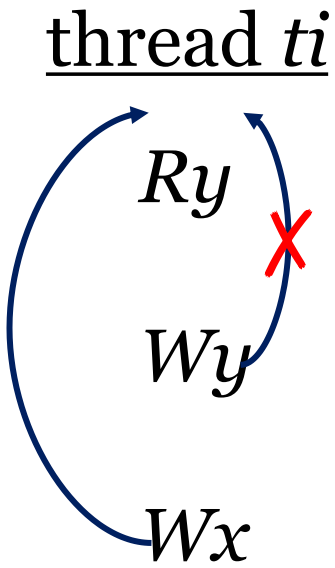


Reordering *through* Interleaving

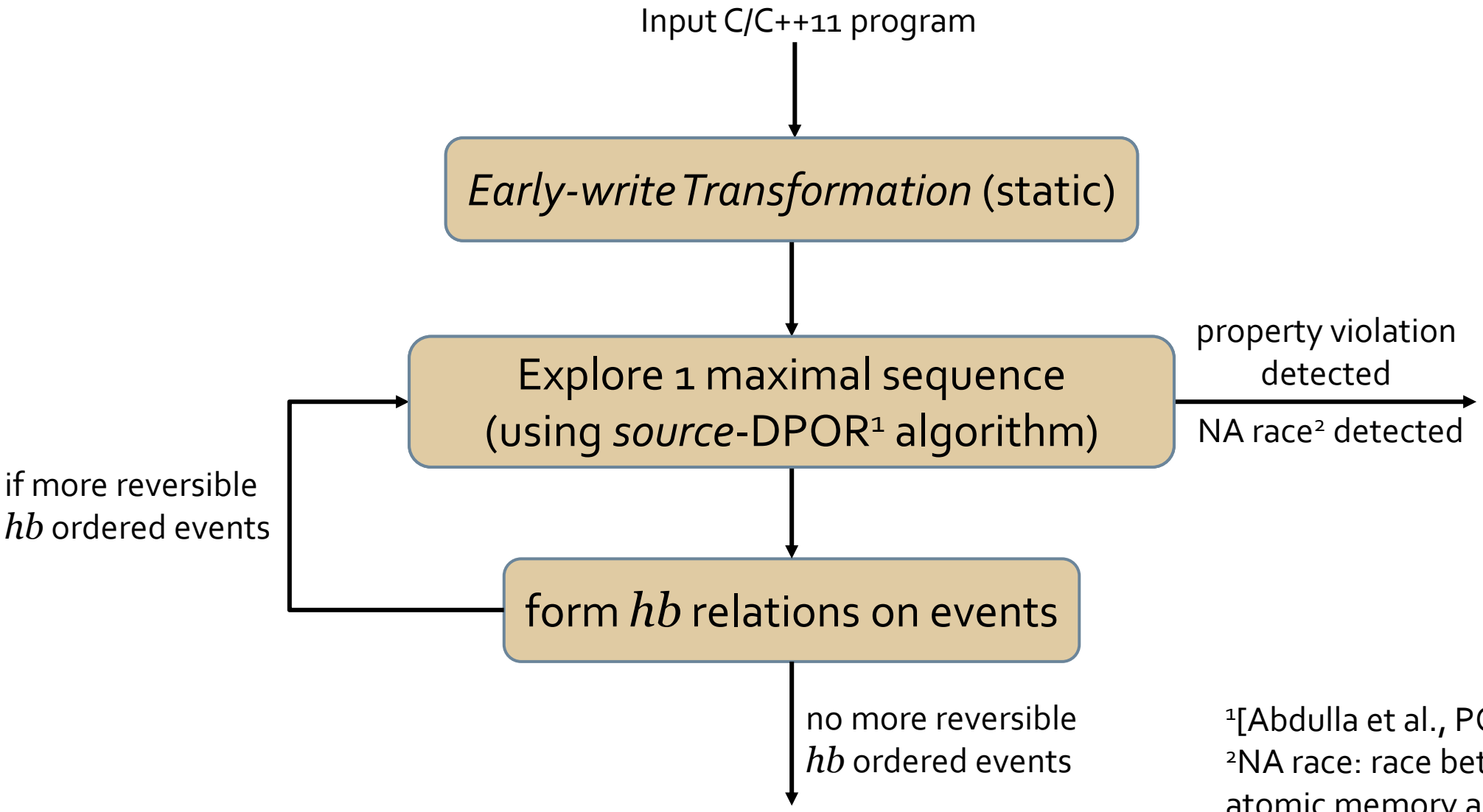
Early-write Transformation

Theorem 3. *Early-write (P to $\widehat{P}$) is semantic preserving.*

MCA model [Colvin and Smith, FM 2018]



MOCA's Technique

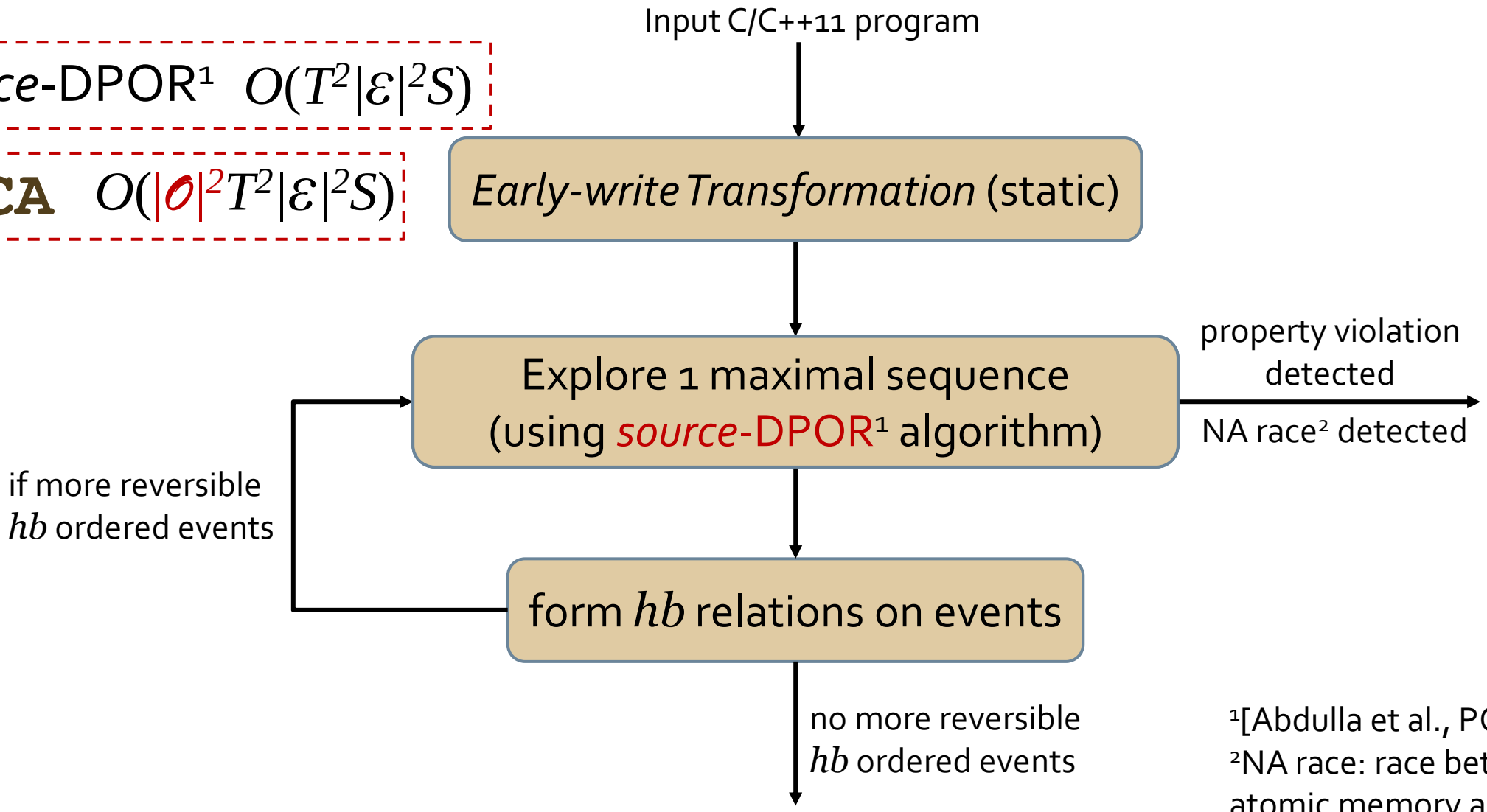


[1][Abdulla et al., POPL 2014]
[2]NA race: race between non-atomic memory accesses

MOCA's Technique

source-DPOR[1] $O(T^2|\mathcal{E}|^2S)$

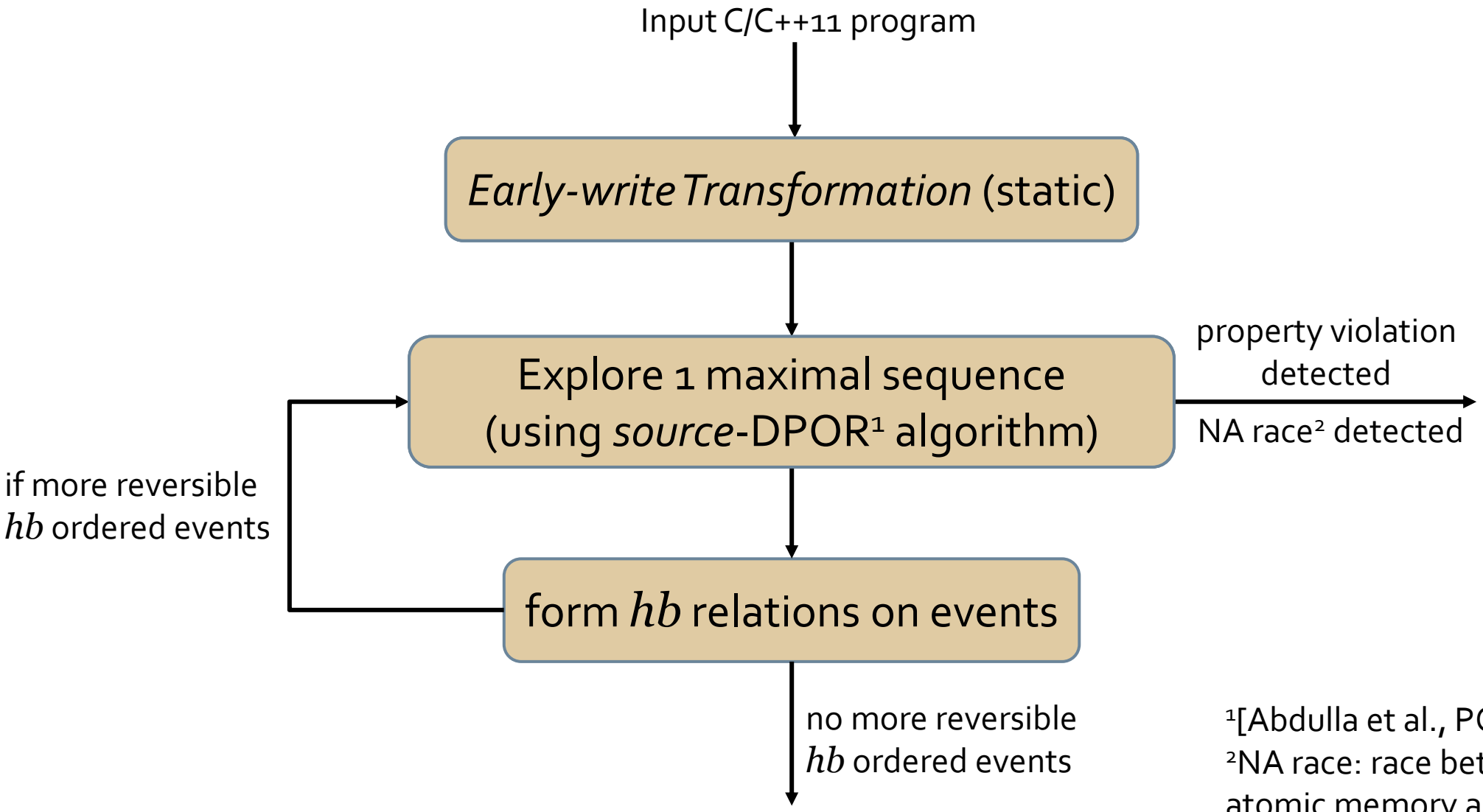
MOCA $O(|\mathcal{O}|^2T^2|\mathcal{E}|^2S)$



[1][Abdulla et al., POPL 2014]
[2]NA race: race between non-atomic memory accesses

MOCA's Technique

Theorem 4. *MoCA traces are equivalent to C11 traces valid over MCA*



[1][Abdulla et al., POPL 2014]
[2]NA race: race between non-atomic memory accesses

Comparative Results on litmus tests

Test	MoCA			CDSChecker			GenMC			HMC		
	M	N	Time	M	N	Time	M	N	Time	M	N	Time
WRC+addrs(7)	7	0	0.03s	7	1	0.01s	7	1	0.02s	7	1	0.03s
WR-ctrl(4)	7	0	0.03s	4	2	0.01s	4	2	0.02s	4	2	0.02s
Z6+poxxs(4)	18	0	0.12s	14	4	0.01s	4	4	0.03s	4	4	0.03s
IRIW+addrs(15)	15	0	0.07s	15	1	0.01s	15	1	0.02s	15	1	0.02s
WW+RR(15)	96	0	0.53s	15	66	0.02s	15	66	0.02s	15	66	0.02s

M: #*MCA* sequences, N: #non-*MCA* sequences

CDSChecker [Norris and Demsky, OOPSLA 2013]
GenMC [Kokologiannakis et al., POPL 2017]
HMC [Kokologiannakis and Vafeiadis , ASPLOS 2020]

Comparative Results on litmus tests

Test	MoCA			CDSChecker			GenMC			HMC		
	M	N	Time	M	N	Time	M	N	Time	M	N	Time
WRC+addrs(7)	7	0	0.03s	7	1	0.01s	7	1	0.02s	7	1	0.03s
WR-ctrl(4)	7	0	0.03s	4	2	0.01s	4	2	0.02s	4	2	0.02s
Z6+poxxs(4)	18	0	0.12s	14	4	0.01s	4	4	0.03s	4	4	0.03s
IRIW+addrs(15)	15	0	0.07s	15	1	0.01s	15	1	0.02s	15	1	0.02s
WW+RR(15)	96	0	0.53s	15	66	0.02s	15	66	0.02s	15	66	0.02s

M: #*MCA* sequences, N: #non-*MCA* sequences

CDSChecker [Norris and Demsky, OOPSLA 2013]
GenMC [Kokologiannakis et al., POPL 2017]
HMC [Kokologiannakis and Vafeiadis , ASPLOS 2020]

Comparative Results on benchmarks

Test	MoCA		CDSChecker		GenMC		HMC	
	#Seq	Time	#Seq	Time	#Seq	Time	#Seq	Time
mutex	5	0.02s	2-NVs	0.01s	NV	0.03s	NV	0.02s
peterson	13	0.15s	666-NVs	2.73s	NV	0.02s	NV	0.02s
RW-lock	246	0.52s	193-NVs	0.38s	NV	0.02s	NV	0.04s
spinlock	506	16.98s	TO	-	NV	0.08s	NV	0.15s
fibonacci-2	667	5.57s	TO	-	NV	0.04s	NV	0.03s
fibonacci-3	10628	2m14s	TO	-	NV	0.06s	NV	0.07s
fibonacci-4	92421	56m21s	TO	-	NV	0.13s	NV	0.31s
counter-5	3599	39.78s	25-NVs	0.31s	NV	0.06s	NV	0.03s
counter-10	55927	12m53s	100-NVs	9.21s	NV	0.05s	NV	0.07s
counter-15	TO	-	225-NVs	50.31s	NV	0.11s	NV	0.16s
flipper-5	2489	20.19s	201-NVs	3.26s	NV	0.03s	NV	0.04s
flipper-10	96737	6m12s	TO	-	NV	0.04s	NV	0.02s
flipper-15	TO	-	TO	-	NV	0.03s	NV	0.03s
prod-cons-10	9373	1m23s	TO	-	NV	0.04s	NV	0.04s
prod-cons-15	38593	6m46s	TO	-	NV	0.02s	NV	0.02s
prod-cons-20	109838	20m28s	TO	-	NV	0.02s	NV	0.02s

Comparative Results on benchmarks

Test	MoCA		CDSChecker		GenMC		HMC	
	#Seq	Time	#Seq	Time	#Seq	Time	#Seq	Time
mutex	5	0.02s	2-NVs	0.01s	NV	0.03s	NV	0.02s
peterson	13	0.15s	666-NVs	2.73s	NV	0.02s	NV	0.02s
RW-lock	246	0.52s	193-NVs	0.38s	NV	0.02s	NV	0.04s
spinlock	506	16.98s	TO	-	NV	0.08s	NV	0.15s
fibonacci-2	667	5.57s	TO	-	NV	0.04s	NV	0.03s
fibonacci-3	10628	2m14s	TO	-	NV	0.06s	NV	0.07s
fibonacci-4	92421	56m21s	TO	-	NV	0.13s	NV	0.31s
counter-5	3599	39.78s	25-NVs	0.31s	NV	0.06s	NV	0.03s
counter-10	55927	12m53s	100-NVs	9.21s	NV	0.05s	NV	0.07s
counter-15	TO	-	225-NVs	50.31s	NV	0.11s	NV	0.16s
flipper-5	2489	20.19s	201-NVs	3.26s	NV	0.03s	NV	0.04s
flipper-10	96737	6m12s	TO	-	NV	0.04s	NV	0.02s
flipper-15	TO	-	TO	-	NV	0.03s	NV	0.03s
prod-cons-10	9373	1m23s	TO	-	NV	0.04s	NV	0.04s
prod-cons-15	38593	6m46s	TO	-	NV	0.02s	NV	0.02s
prod-cons-20	109838	20m28s	TO	-	NV	0.02s	NV	0.02s

Comparative Results on benchmarks

Test	MoCA		CDSChecker		GenMC		HMC	
	#Seq	Time	#Seq	Time	#Seq	Time	#Seq	Time
mutex	5	0.02s	2-NVs	0.01s	NV	0.03s	NV	0.02s
peterson	13	0.15s	666-NVs	2.73s	NV	0.02s	NV	0.02s
RW-lock	246	0.52s	193-NVs	0.38s	NV	0.02s	NV	0.04s
spinlock	506	16.98s	TO	-	NV	0.08s	NV	0.15s
fibonacci-2	667	5.57s	TO	-	NV	0.04s	NV	0.03s
fibonacci-3	10628	2m14s	TO	-	NV	0.06s	NV	0.07s
fibonacci-4	92421	56m21s	TO	-	NV	0.13s	NV	0.31s
counter-5	3599	39.78s	25-NVs	0.31s	NV	0.06s	NV	0.03s
counter-10	55927	12m53s	100-NVs	9.21s	NV	0.05s	NV	0.07s
counter-15	TO	-	225-NVs	50.31s	NV	0.11s	NV	0.16s
flipper-5	2489	20.19s	201-NVs	3.26s	NV	0.03s	NV	0.04s
flipper-10	96737	6m12s	TO	-	NV	0.04s	NV	0.02s
flipper-15	TO	-	TO	-	NV	0.03s	NV	0.03s
prod-cons-10	9373	1m23s	TO	-	NV	0.04s	NV	0.04s
prod-cons-15	38593	6m46s	TO	-	NV	0.02s	NV	0.02s
prod-cons-20	109838	20m28s	TO	-	NV	0.02s	NV	0.02s

Comparative Results on benchmarks

Test	MoCA		CDSChecker		GenMC		HMC	
	#Seq	Time	#Seq	Time	#Seq	Time	#Seq	Time
mutex	5	0.02s	2-NVs	0.01s	NV		NV	
peterson	13	0.15s	666-NVs	2.73s	NV		NV	
RW-lock	246	0.52s	193-NVs	0.38s	NV		NV	
spinlock	506	16.98s	TO	-	NV		NV	
fibonacci-2	667	5.57s	TO	-	NV		NV	
fibonacci-3	10628	2m14s	TO	-	NV		NV	
fibonacci-4	92421	56m21s	TO	-	NV		NV	
counter-5	3599	39.78s	25-NVs	0.31s	NV		NV	
counter-10	55927	12m53s	100-NVs	9.21s	NV		NV	
counter-15	TO	-	225-NVs	50.31s	NV		NV	
flipper-5	2489	20.19s	201-NVs	3.26s	NV		NV	
flipper-10	96737	6m12s	TO	-	NV		NV	
flipper-15	TO	-	TO	-	NV		NV	
prod-cons-10	9373	1m23s	TO	-	NV		NV	
prod-cons-15	38593	6m46s	TO	-	NV		NV	
prod-cons-20	109838	20m28s	TO	-	NV		NV	

Thank You

[1] ISO/IEC, "Programming languages – c. international standard," www.open-std.org/jtc1/sc22/wg14/www/docs/n1548.pdf, ISO/IEC, 2011.

[2] M. Batty, S. Owens, S. Sarkar, P. Sewell, and T. Weber, "Mathematizing c++ concurrency," *ACM SIGPLAN Notices*, vol. 46, no. 1, pp. 55–66, 2011.

[3] P. A. Abdulla, J. Arora, M. F. Atig, and S. Krishna, "Verification of programs under the release-acquire semantics," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 1117–1132.

[4] P. Abdulla, S. Aronis, B. Jonsson, and K. Sagonas, "Optimal dynamic partial order reduction," *ACM SIGPLAN Notices*, vol. 49, no. 1, pp. 373–384, 2014.

[5] C. Flanagan and P. Godefroid, "Dynamic partial-order reduction for model checking software," *ACM Sigplan Notices*, vol. 40, no. 1, pp. 110–121, 2005.

[6] P. A. Abdulla, S. Aronis, M. F. Atig, B. Jonsson, C. Leonardsson, and K. Sagonas, "Stateless model checking for tso and pso," in *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems - Volume 9035*, 2015.

[7] N. Zhang, M. Kusano, and C. Wang, "Dynamic partial order reduction for relaxed memory models," in *ACM SIGPLAN Notices*, vol. 50, no. 6. ACM, 2015, pp. 250–259.

[8] A. Ltd., "Arm architecture reference manual (armv8, for armv8-a architecture profile)," https://developer.arm.com/documentation/ddi0487/aa, ARM Ltd., 2017.

[9] C. Pulte, S. Flur, W. Deacon, J. French, S. Sarkar, and P. Sewell, "Simplifying arm concurrency: multicopy-atomic axiomatic and operational models for armv8," *Proceedings of the ACM on Programming Languages*, vol. 2, no. POPL, pp. 1–29, 2017.

[10] B. Norris and B. Demsky, "Cdschecker: checking concurrent data structures written with c/c++ atomics," in *ACM SIGPLAN Notices*, vol. 48, no. 10. ACM, 2013, pp. 131–150.

[11] M. Kokologiannakis, O. Lahav, K. Sagonas, and V. Vafeiadis, "Effective stateless model checking for c/c++ concurrency," *Proceedings of the ACM on Programming Languages*, vol. 2, no. POPL, p. 17, 2017.

[12] M. Kokologiannakis, A. Raad, and V. Vafeiadis, "Model checking for weakly consistent libraries," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 96–110.

[13] A. Podkopaev, O. Lahav, and V. Vafeiadis, "Bridging the gap between programming languages and hardware weak memory models," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–31, 2019.

[14] M. Kokologiannakis and V. Vafeiadis, "Hmc: Model checking for hardware memory models," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 1157–1171.

[15] R. J. Colvin and G. Smith, "A wide-spectrum language for verification of programs on weak memory models," in *International Symposium on Formal Methods*. Springer, 2018, pp. 240–257.

[16] A. Jade and M. Luc, "diy7 tool suite," http://diy.inria.fr/doc/index.html, 2017, accessed: 2020-02-12.

[17] "Competition on software verification," https://sv-comp.sosy-lab.org/2018/benchmarks.php, SV-COMP, 2018, accessed: 2019-10-04.

[18] M. Kokologiannakis, A. Raad, and V. Vafeiadis, "Model checking for weakly consistent libraries," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 96–110.

[19] Y. Yang, X. Chen, G. Gopalakrishnan, and R. M. Kirby, "Efficient stateful dynamic partial order reduction," in *Model Checking Software, 15th International SPIN Workshop, Los Angeles, CA, USA, August 10-12, 2008, Proceedings*, 2008, pp. 288–305.

[20] J. Simsa, R. Bryant, G. A. Gibson, and J. Hickey, "Scalable dynamic partial order reduction," in *Runtime Verification, Third International Conference, RV 2012, Istanbul, Turkey, September 25-28, 2012, Revised Selected Papers*, 2012, pp. 19–34.

[21] E. Albert, P. Arenas, M. G. de la Banda, M. Gómez-Zamalloa, and P. J. Stuckey, "Context-sensitive dynamic partial order reduction," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 526–543.

[22] M. Chalupa, K. Chatterjee, A. Pavlogiannis, N. Sinha, and K. Vaidya, "Data-centric dynamic partial order reduction," *Proc. ACM Program. Lang.*, vol. 2, no. POPL, pp. 31:1–31:30, 2017.

[23] P. A. Abdulla, M. F. Atig, B. Jonsson, and C. Leonardsson, "Stateless model checking for power," in *International Conference on Computer Aided Verification*. Springer, 2016, pp. 134–156.

[24] P. A. Abdulla, M. F. Atig, B. Jonsson, and T. P. Ngo, "Optimal stateless model checking under the release-acquire semantics," *Proceedings of the ACM on Programming Languages*, vol. 2, no. OOPSLA, p. 135, 2018.

[25] C. Rodríguez, M. Sousa, S. Sharma, and D. Kroening, "Unfolding-based partial order reduction," in *CONCUR*, 2015.

[26] H. T. T. Nguyen, C. Rodríguez, M. Sousa, C. Coti, and L. Petrucci, "Quasi-optimal partial order reduction," in *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, 2018, pp. 354–371.

[27] V. Forejt, D. Kroening, G. Narayanaswamy, and S. Sharma, "Precise predictive analysis for discovering communication deadlocks in mpi programs," in *International Symposium on Formal Methods*. Springer, 2014, pp. 263–278.

[28] A. Gupta, T. A. Henzinger, A. Radhakrishna, R. Samanta, and T. Tarrach, "Succinct representation of concurrent trace sets," in *ACM SIGPLAN Notices*, vol. 50, no. 1. ACM, 2015, pp. 433–444.

[29] C. Wang, S. Kundu, M. K. Ganai, and A. Gupta, "Symbolic predictive analysis for concurrent programs," in *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*, 2009, pp. 256–272.

[30] D. Khanna, S. Sharma, C. Rodríguez, and R. Purandare, "Dynamic symbolic verification of MPI programs," in *Formal Methods - 22nd International Symposium, FM 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 15-17, 2018, Proceedings*, 2018, pp. 466–484.

[31] C. Lidbury and A. F. Donaldson, "Dynamic race detection for c++ 11," in *ACM SIGPLAN Notices*, vol. 52, no. 1. ACM, 2017, pp. 443–457.

[32] J. Huang, P. O. Meredith, and G. Rosu, "Maximal sound predictive race detection with control flow abstraction," in *Proceedings of the 35th ACM SIGPLAN conference on programming language design and implementation*, 2014, pp. 337–348.

[33] C. Flanagan, S. N. Freund, and S. Qadeer, "Thread-modular verification for shared-memory programs," in *European Symposium on Programming*. Springer, 2002, pp. 262–277.

[34] M. Kusano and C. Wang, "Flow-sensitive composition of thread-modular abstract interpretation," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 799–809.

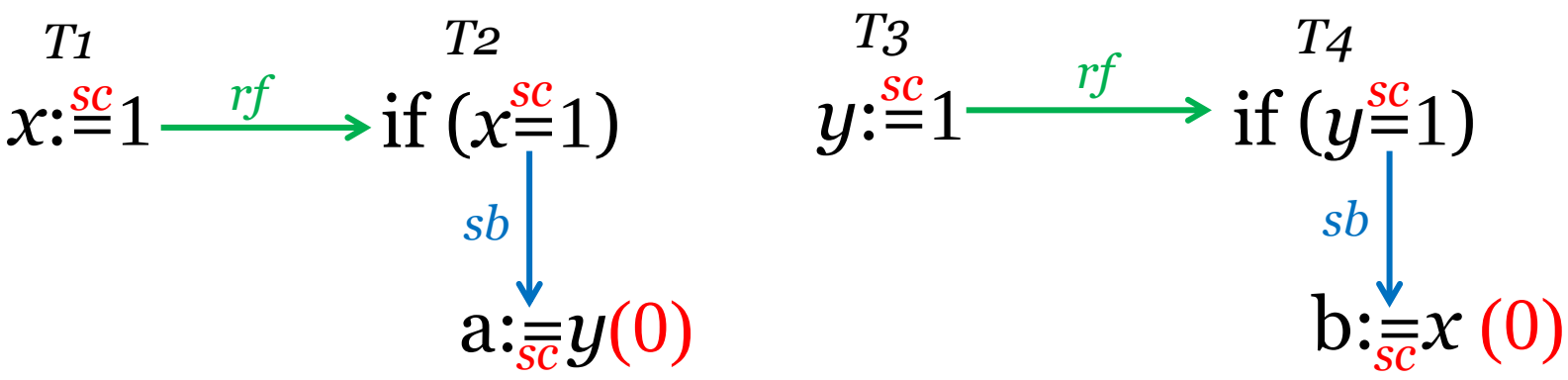
[35] ——, "Thread-modular static analysis for relaxed memory models," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017, 2017.

[36] T. Suzanne and A. Miné, "Relational thread-modular abstract interpretation under relaxed memory models," in *Asian Symposium on Programming Languages and Systems*. Springer, 2018, pp. 109–128.

[37] H. Ponce-de León, F. Furbach, K. Heljanko, and R. Meyer, "Dartagnan: Bounded model checking for weak memory models (competition contribution)," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2020, pp. 378–382.

C/C++11 over MCA

(IRIW) allowed under C/C++11 *unless all events are sc ordered*



specified as:

```
x.store(1, memory_order_seq_cst);
x.load(memory_order_seq_cst);
```

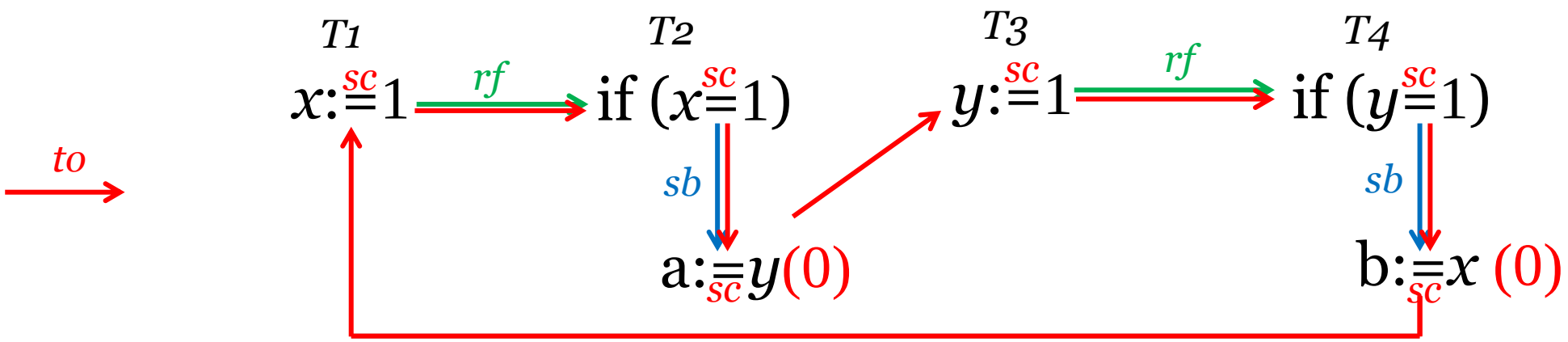
other memory orders:

rlx ⊏ {*rel*, *acq*} ⊏ *acq-rel* ⊏ *sc*

back

C/C++11 over MCA

(IRIW) allowed under C/C++11 *unless all events are sc ordered*



specified as:

```
x.store(1, memory_order_seq_cst);
x.load(memory_order_seq_cst);
```

other memory orders:

rlx ⊏ {*rel*, *acq*} ⊏ *acq-rel* ⊏ *sc*

back

SUB-TASK: Identify MCA model semantics [Colvin and Smith, FM 2018]

Due to the ***dynamic*** nature of technique

- ~~local operations~~
- ~~branches and paths~~

$$\frac{c \xrightarrow{\beta} c' \quad \alpha \overset{\mathrm{R}}{\Leftarrow} \beta_{[\alpha]}}{(\alpha\,;\ c) \xrightarrow{\beta_{[\alpha]}} (\alpha\,;\ c')}$$

(reordering)

$$\frac{c \xrightarrow{x\,:=\,r} c' \quad \sigma(r) = v}{(\mathbf{lcl}\ \sigma \bullet c) \xrightarrow{x\,:=\,v} (\mathbf{lcl}\ \sigma \bullet c')}$$

(write-issue)

$$\frac{c \xrightarrow{r\,:=\,x} c'}{(\mathbf{lcl}\ \sigma \bullet c) \xrightarrow{[x=v]} (\mathbf{lcl}\ \sigma_{[r\,:=\,v]} \bullet c')}$$

(read-shared)

$$\frac{p \xrightarrow{\mathrm{N}:x\,:=\,e} p'}{(\mathbf{glb}\ \sigma \bullet p) \xrightarrow{\tau} (\mathbf{glb}\ \sigma_{[x\,:=\,e_\sigma]} \bullet p')}$$

(write-update)

$$\frac{p_1 \xrightarrow{\alpha} p_1'}{p_1 \parallel p_2 \xrightarrow{\alpha} p_1' \parallel p_2} \qquad \frac{p_2 \xrightarrow{\alpha} p_2'}{p_1 \parallel p_2 \xrightarrow{\alpha} p_1 \parallel p_2'}$$

(parallel-composition)

back