Verification under Weak Consistency and Recent Decidability Results in Verification

Roland Meyer, TU Braunschweig

Joint work with the best PhD students in the world :)

IFIP WG 2.3, Athens, May 2025

Verification under Weak Consistency

Introductory example

```
         x = y = 0
(a) x = 1;   |  (c) y = 1;
(b) r1 = y;  |  (d) r2 = x;
```

Introductory example

```
          x = y = 0
(a) x = 1;   |  (c) y = 1;
(b) r1 = y;  |  (d) r2 = x;
```

Q: What are the possible outcomes for the values of r1 and r2?

Introductory example

A: Check all possible interleavings!

```
         x = y = 0
(a) x = 1;    |  (c) y = 1;
(b) r1 = y;   |  (d) r2 = x;
```

Q: What are the possible outcomes for the values of r1 and r2?

Introductory example

```
         x = y = 0
(a) x = 1;    |  (c) y = 1;
(b) r1 = y;   |  (d) r2 = x;
```

Q: What are the possible outcomes for the values of r1 and r2?

A: Check all possible interleavings!

(a).(b).(c).(d)

(a).(c).(b).(d)

(a).(c).(d).(b)

(c).(d).(a).(b)

(c).(a).(d).(b)

(c).(a).(b).(d)

Introductory example

```
        x = y = 0
(a) x = 1;    |  (c) y = 1;
(b) r1 = y;   |  (d) r2 = x;
```

Q: What are the possible outcomes for the values of r1 and r2?

A: Check all possible interleavings!

(a).(b).(c).(d)	⇒	r1 = 0, r2 = 1
(a).(c).(b).(d)	⇒	r1 = 1, r2 = 1
(a).(c).(d).(b)	⇒	r1 = 1, r2 = 1
(c).(d).(a).(b)	⇒	r1 = 1, r2 = 0
(c).(a).(d).(b)	⇒	r1 = 1, r2 = 1
(c).(a).(b).(d)	⇒	r1 = 1, r2 = 1

Introductory example

```
         x = y = 0
(a) x = 1;    |  (c) y = 1;
(b) r1 = y;   |  (d) r2 = x;
```

Q: What are the possible outcomes for the values of r1 and r2?

A: Check all possible interleavings!

(a).(b).(c).(d)	r1 = 0, r2 = 1
(a).(c).(b).(d)	r1 = 1, r2 = 1
(a).(c).(d).(b)	r1 = 1, r2 = 1
(c).(d).(a).(b)	r1 = 1, r2 = 0
(c).(a).(d).(b)	r1 = 1, r2 = 1
(c).(a).(b).(d)	r1 = 1, r2 = 1

r1 = r2 = 0 seems impossible…

Introductory example

```
        x = y = 0
(a) x = 1;    |  (c) y = 1;
(b) r1 = y;   |  (d) r2 = x;
```

Q: What are the possible outcomes for the values of r1 and r2?

A: Check all possible interleavings!

(a).(b).(c).(d)	→	r1 = 0, r2 = 1
(a).(c).(b).(d)	→	r1 = 1, r2 = 1
(a).(c).(d).(b)	→	r1 = 1, r2 = 1
(c).(d).(a).(b)	→	r1 = 1, r2 = 0
(c).(a).(d).(b)	→	r1 = 1, r2 = 1
(c).(a).(b).(d)	→	r1 = 1, r2 = 1

r1 = r2 = 0 seems impossible…

…practice says otherwise!

Store buffering
x86/TSO

CPUs can buffer a store locally and only later flush it into main memory

Store buffering

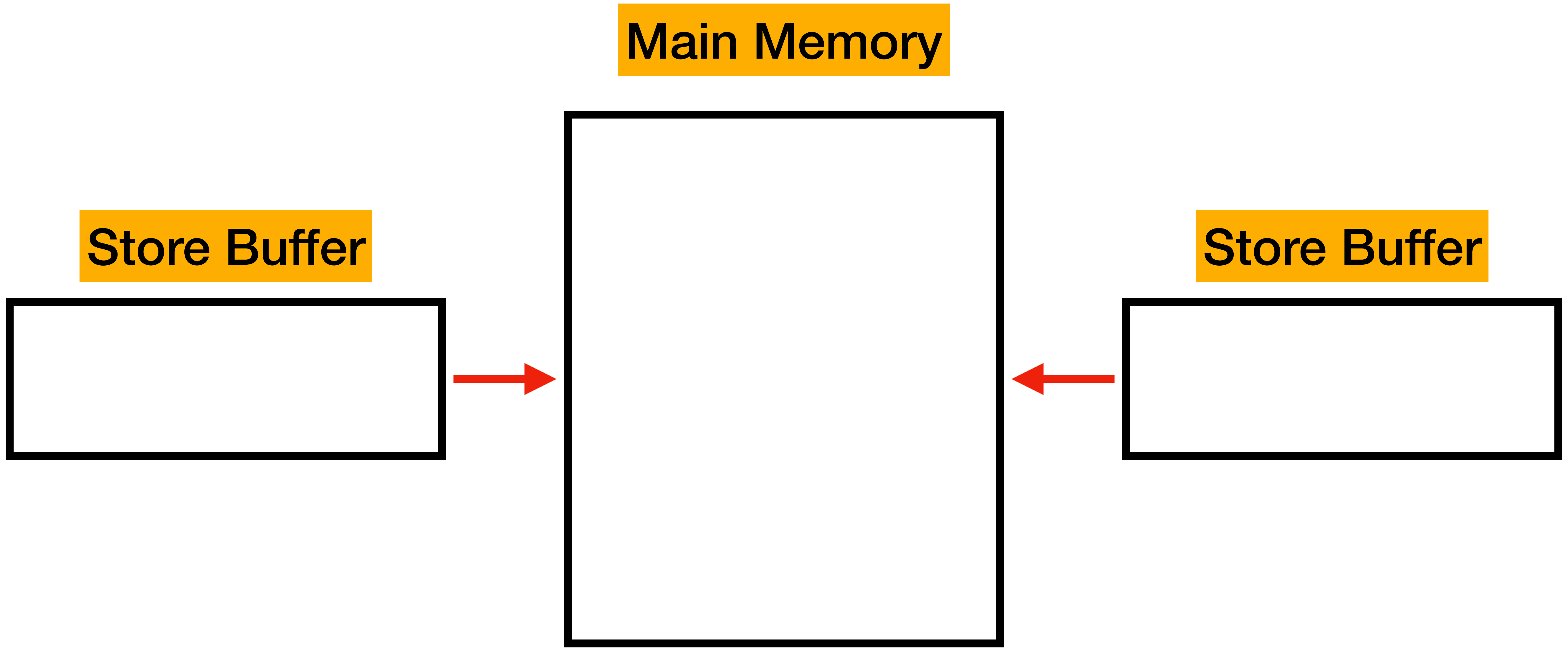
x86/TSO

CPUs can buffer a store locally and only later flush it into main memory

```
(I1) x = 0;          (I2) y = 0;
```

P0

```
(a) x = 1;

(b) r1 = y;
```



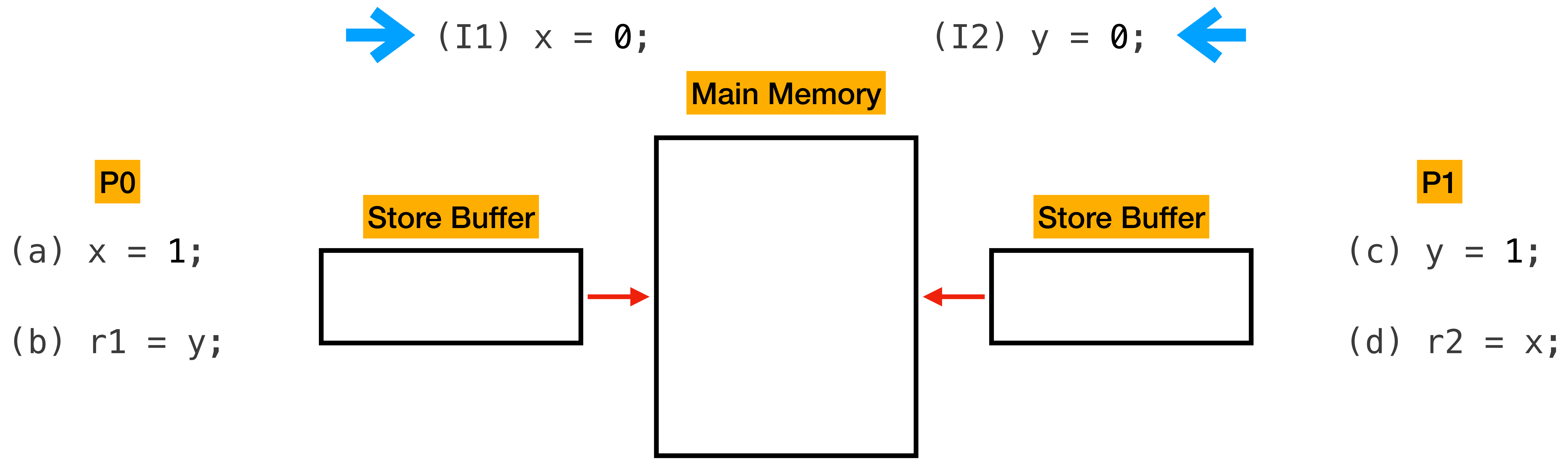
P1

```
(c) y = 1;

(d) r2 = x;
```

Store buffering

x86/TSO

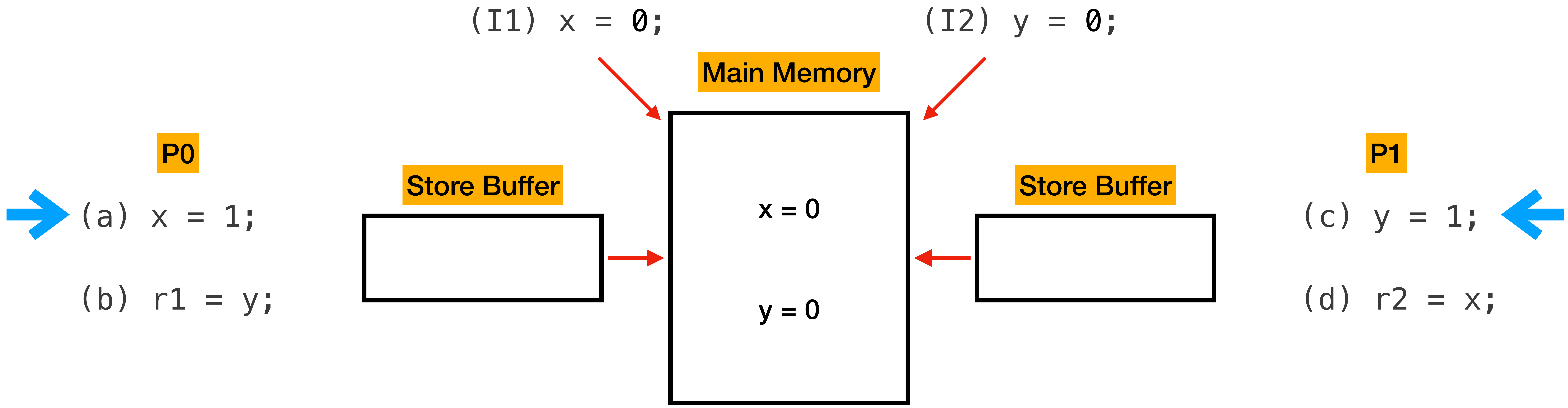
CPUs can buffer a store locally and only later flush it into main memory



Store buffering

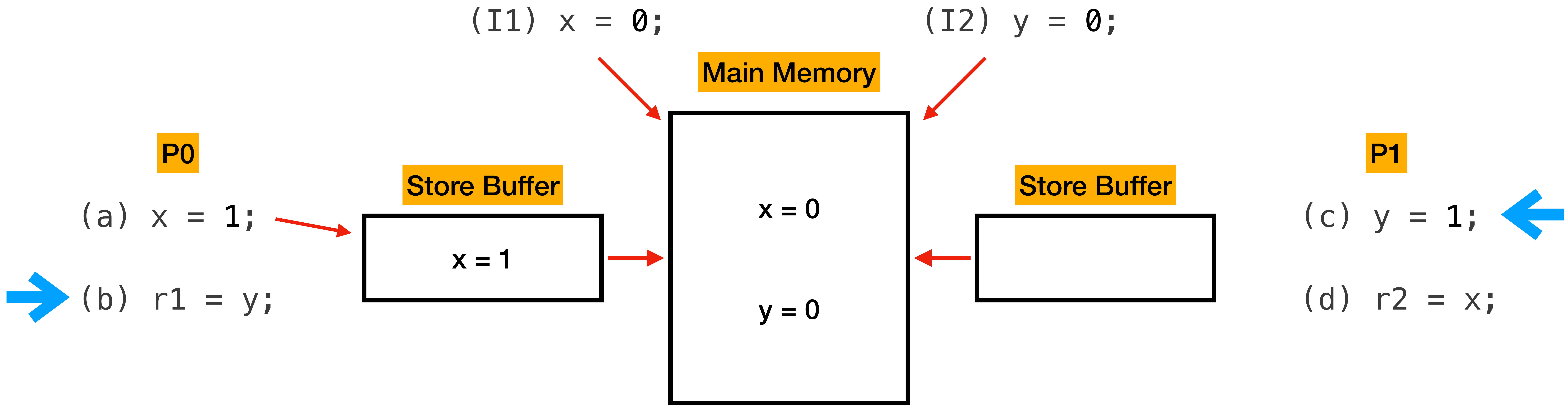
x86/TSO

CPUs can buffer a store locally and only later flush it into main memory



Store buffering
x86/TSO

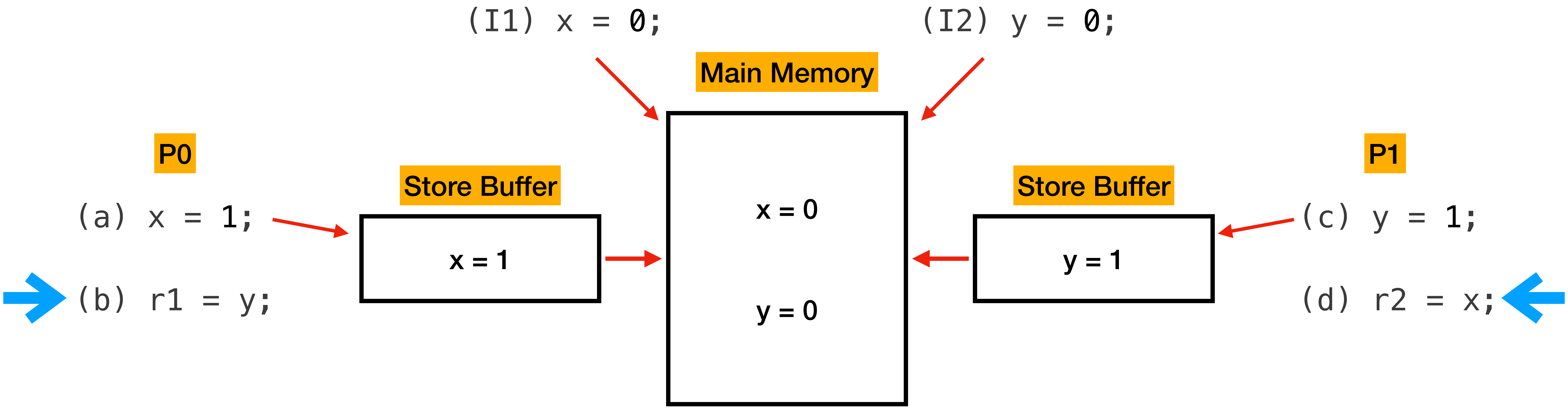
CPUs can buffer a store locally and only later flush it into main memory



Store buffering

x86/TSO

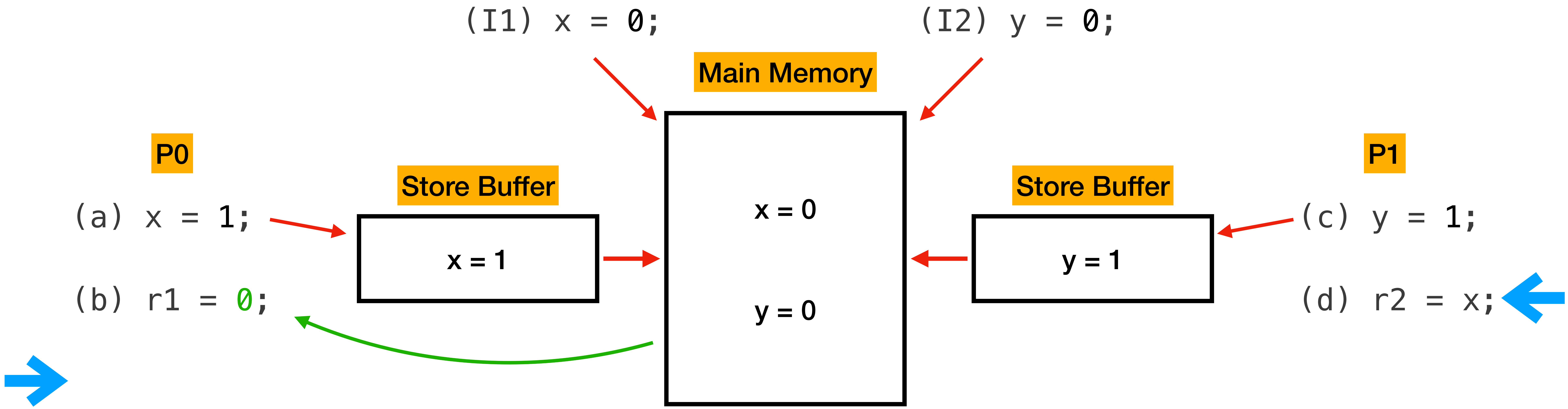
CPUs can buffer a store locally and only later flush it into main memory



Store buffering

x86/TSO

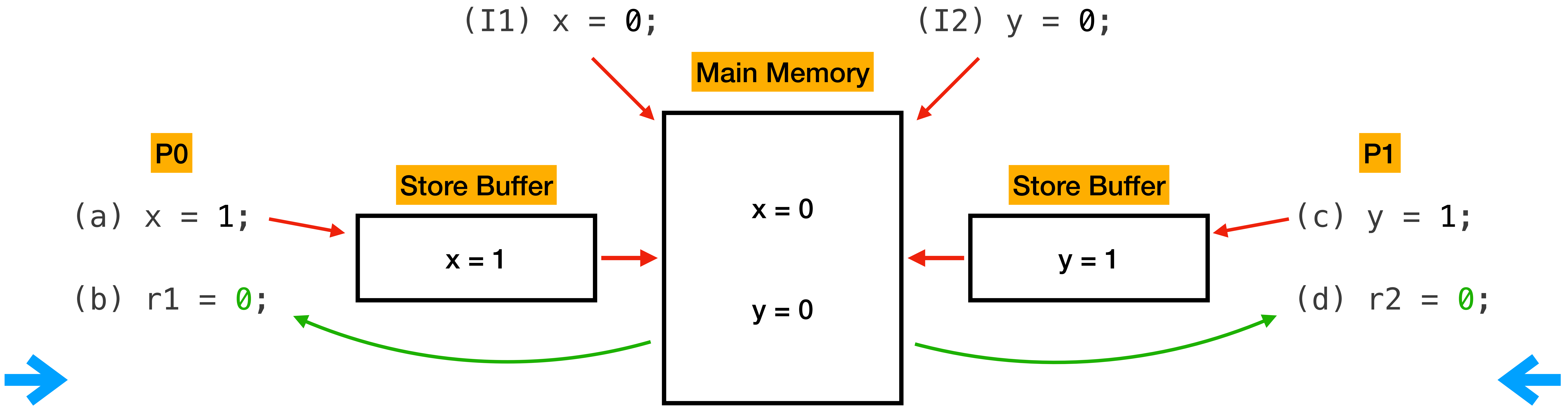
CPUs can buffer a store locally and only later flush it into main memory



Store buffering

x86/TSO

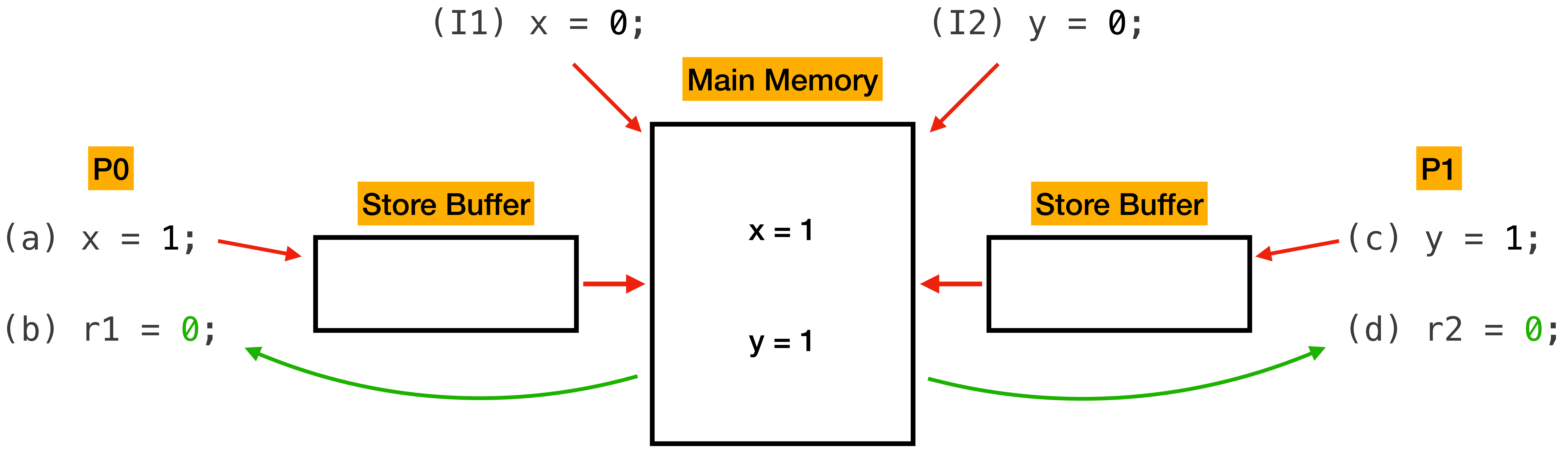
CPUs can buffer a store locally and only later flush it into main memory



Store buffering

x86/TSO

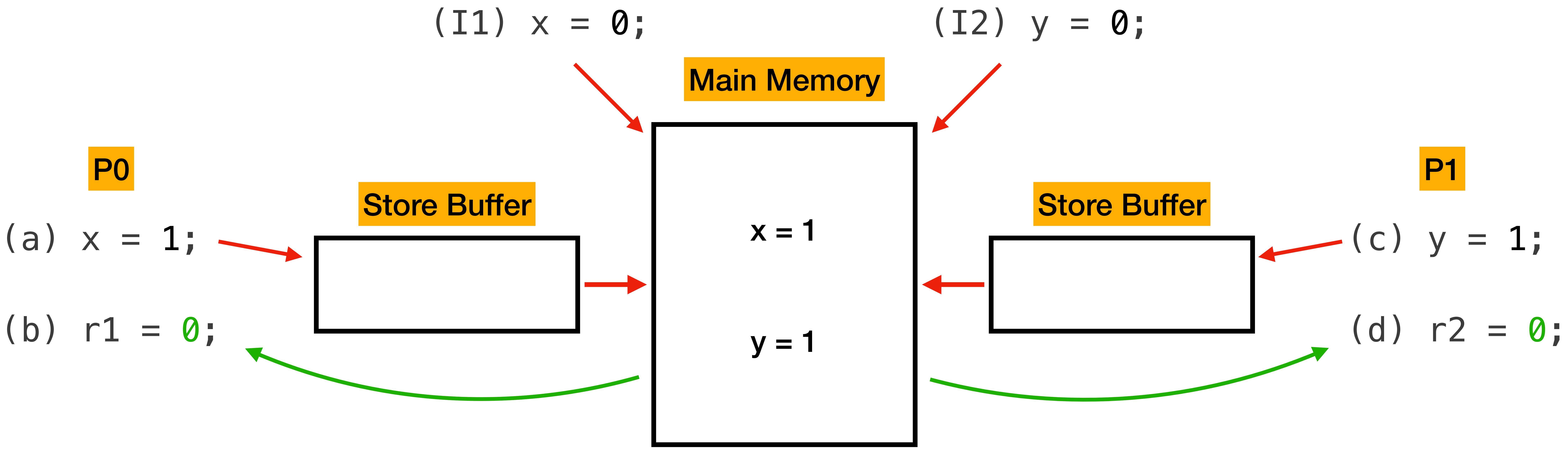
CPUs can buffer a store locally and only later flush it into main memory



Store buffering

x86/TSO

CPUs can buffer a store locally and only later flush it into main memory

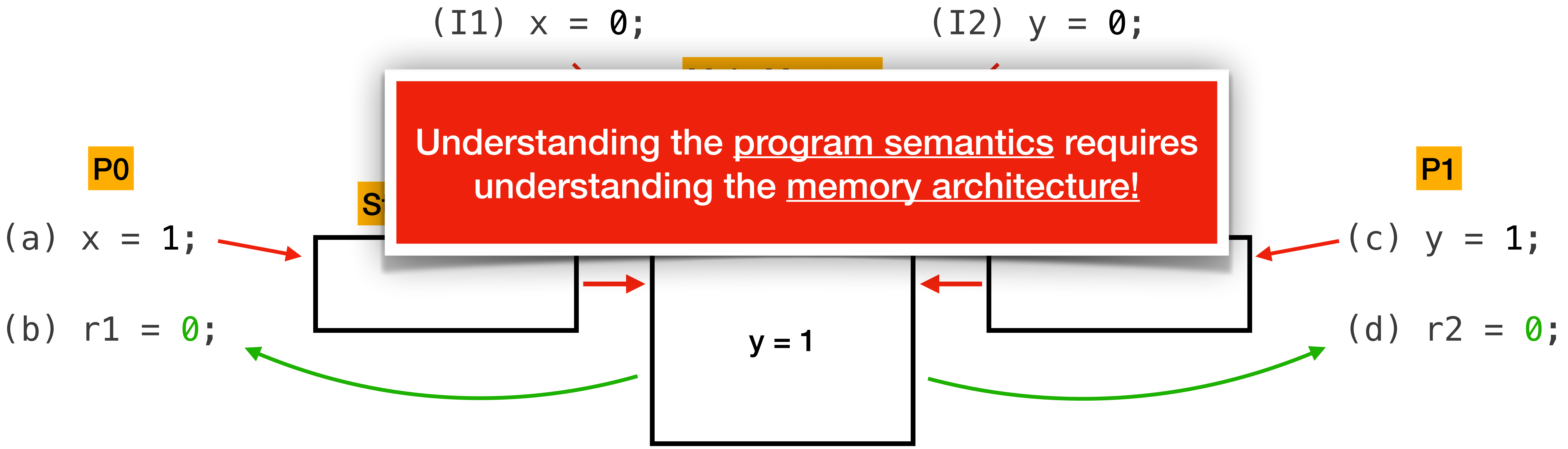


This behavior is not captured by an interleaving!

Store buffering

x86/TSO

CPUs can buffer a store locally and only later flush it into main memory

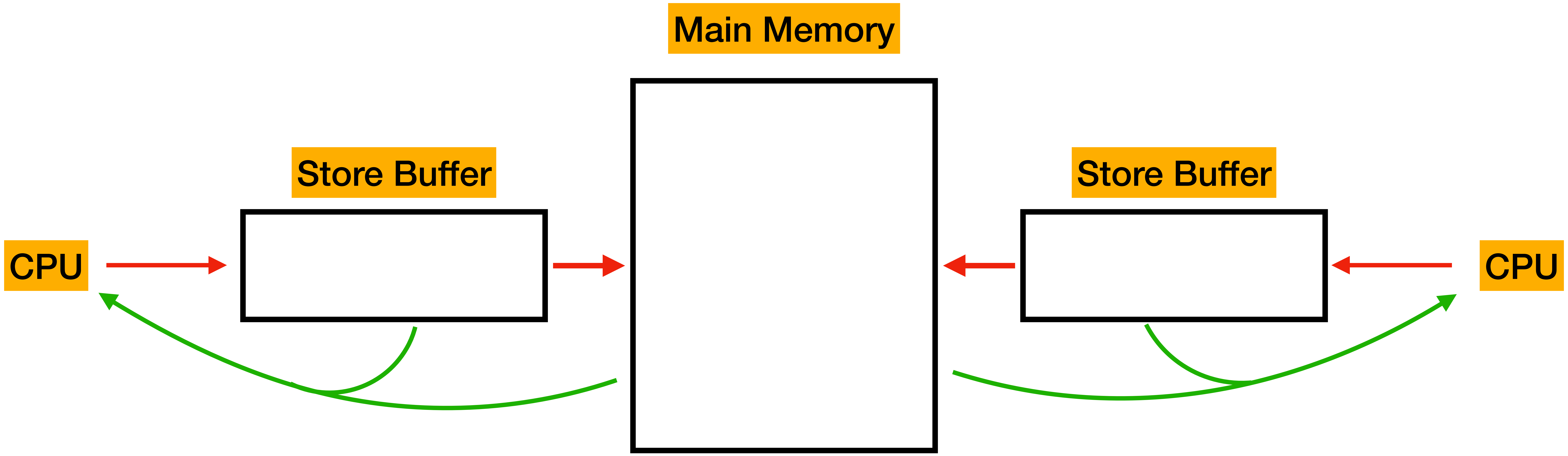


This behavior is not captured by an interleaving!

Memory architectures (sketched)

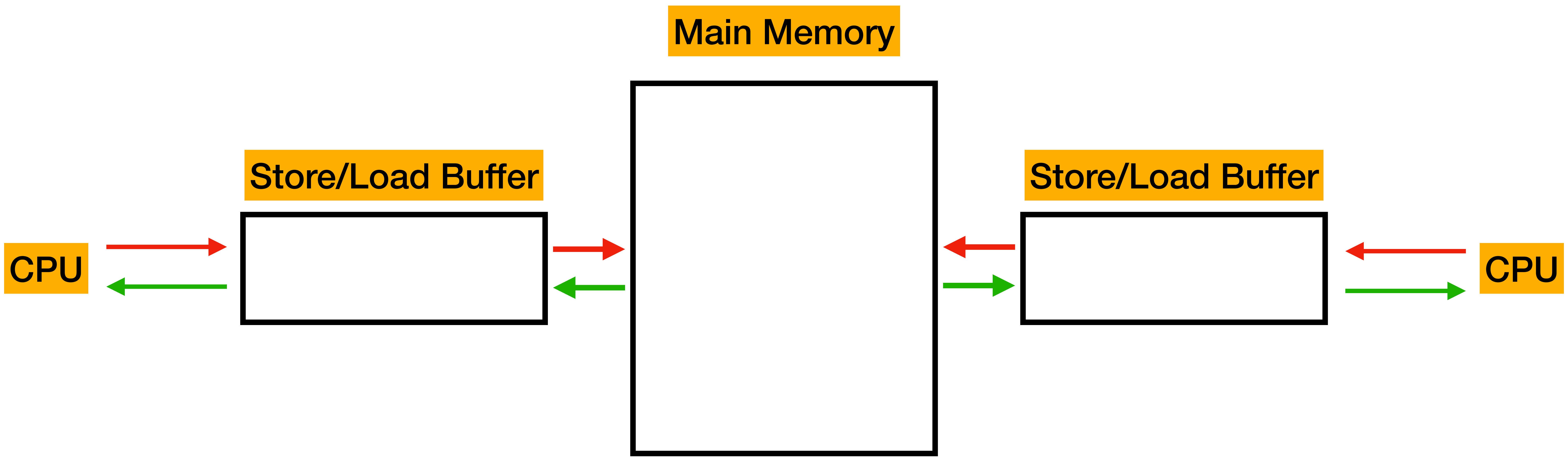
Memory architectures (sketched)

Store buffer



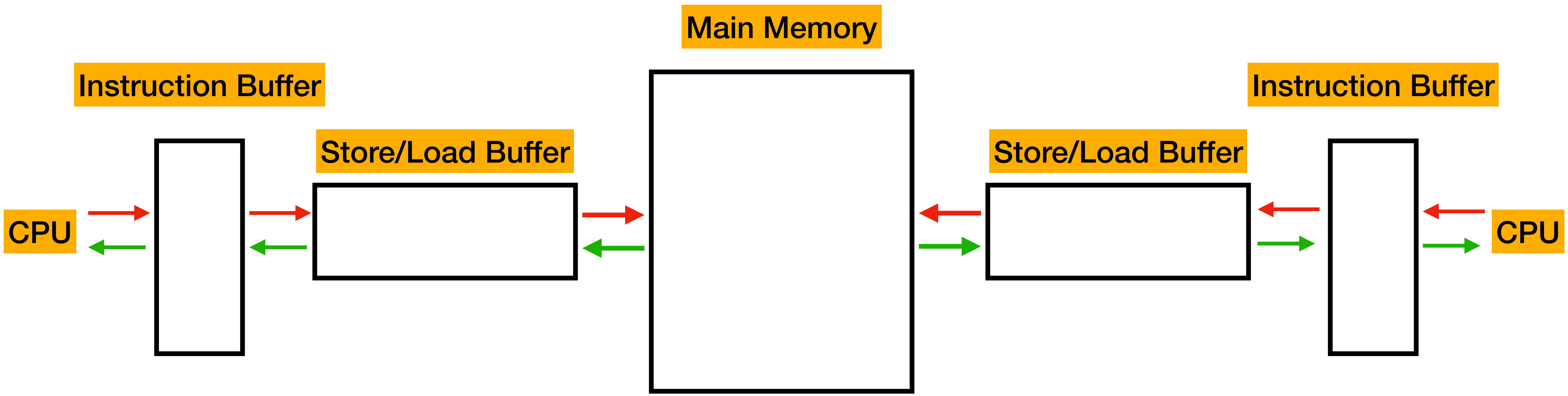
Memory architectures (sketched)

Store/Load buffer



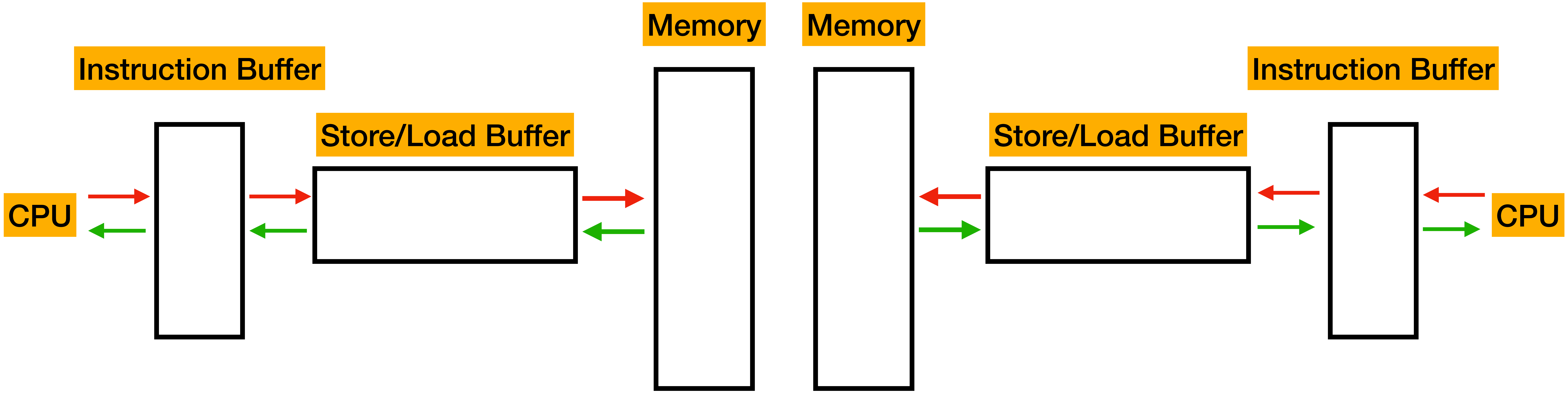
Memory architectures (sketched)

Store/Load buffer, instruction buffer



Memory architectures (sketched)

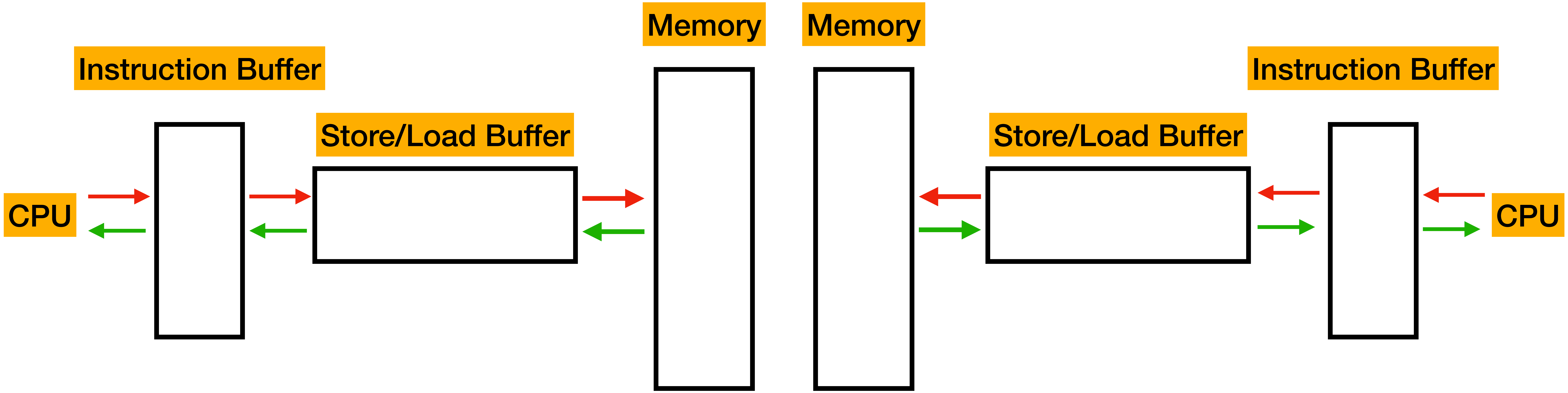
Store/Load buffer, instruction buffer, decentralised memory



Memory architectures (sketched)

Store/Load buffer, instruction buffer, decentralised memory

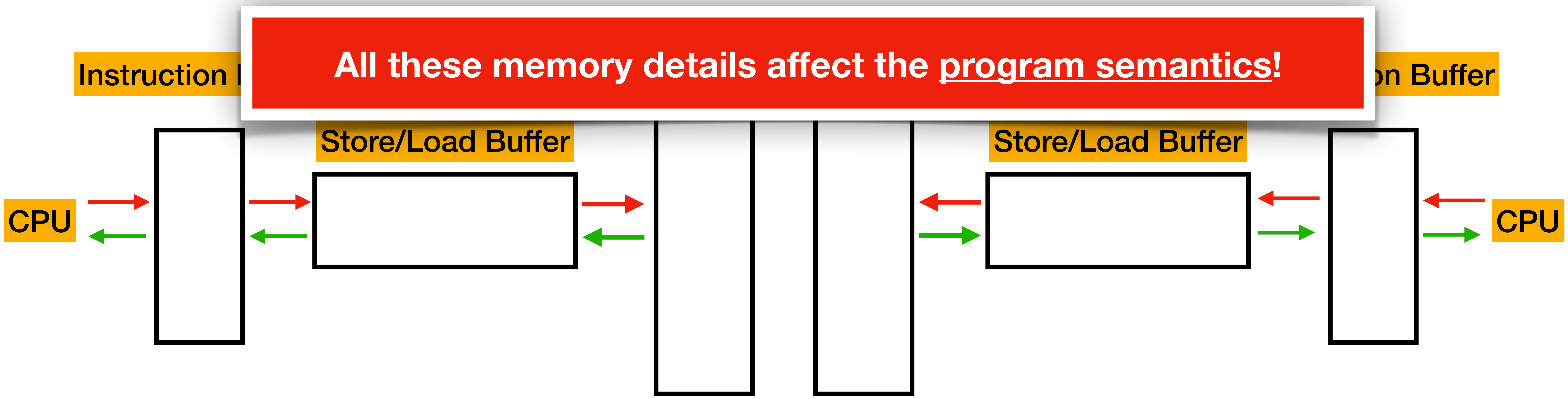
… and much more



Memory architectures (sketched)

Store/Load buffer, instruction buffer, decentralised memory

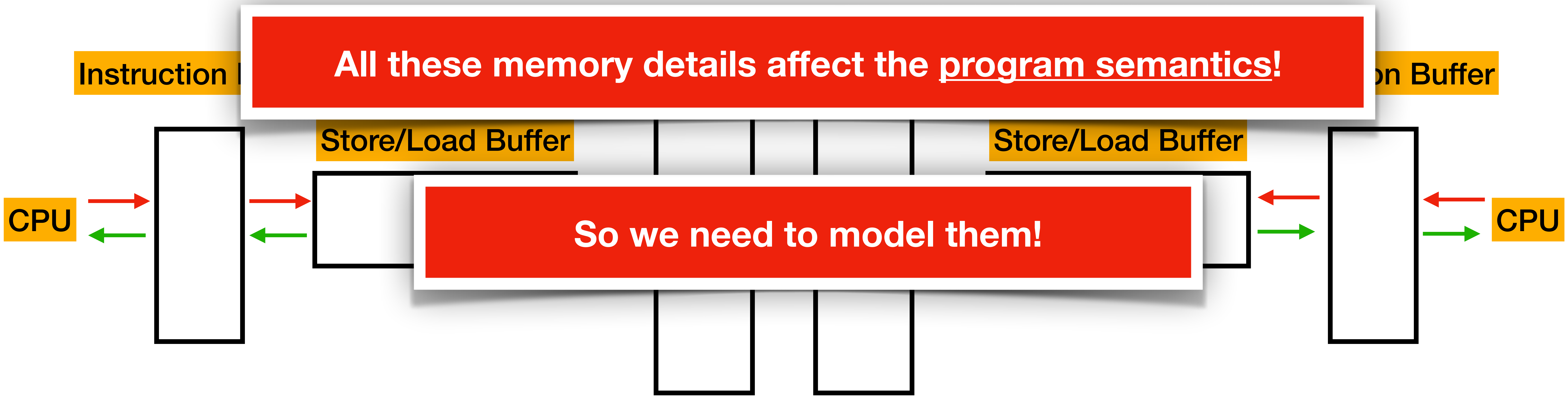
… and much more



Memory architectures (sketched)

Store/Load buffer, instruction buffer, decentralised memory

… and much more



Program semantics

Capturing program semantics

Problems so far

Interleavings are insufficient to capture program behavior

Capturing program semantics

Problems so far

Interleavings are insufficient to capture program behavior

Enrich interleavings with microarchitectural steps (e.g., fetch, execute, write-back)?

Capturing program semantics

Problems so far

Interleavings are insufficient to capture program behavior

Enrich interleavings with microarchitectural steps (e.g., fetch, execute, write-back)?

Requires all details of the memory architecture: too complex too quickly!

Capturing program semantics

Problems so far

Interleavings are insufficient to capture program behavior

- Enrich interleavings with microarchitectural steps (e.g., fetch, execute, write-back)?
  - Requires all details of the memory architecture: too complex too quickly!
    - Forget about architectural details!
      Directly model observable behavior!

Capturing program semantics

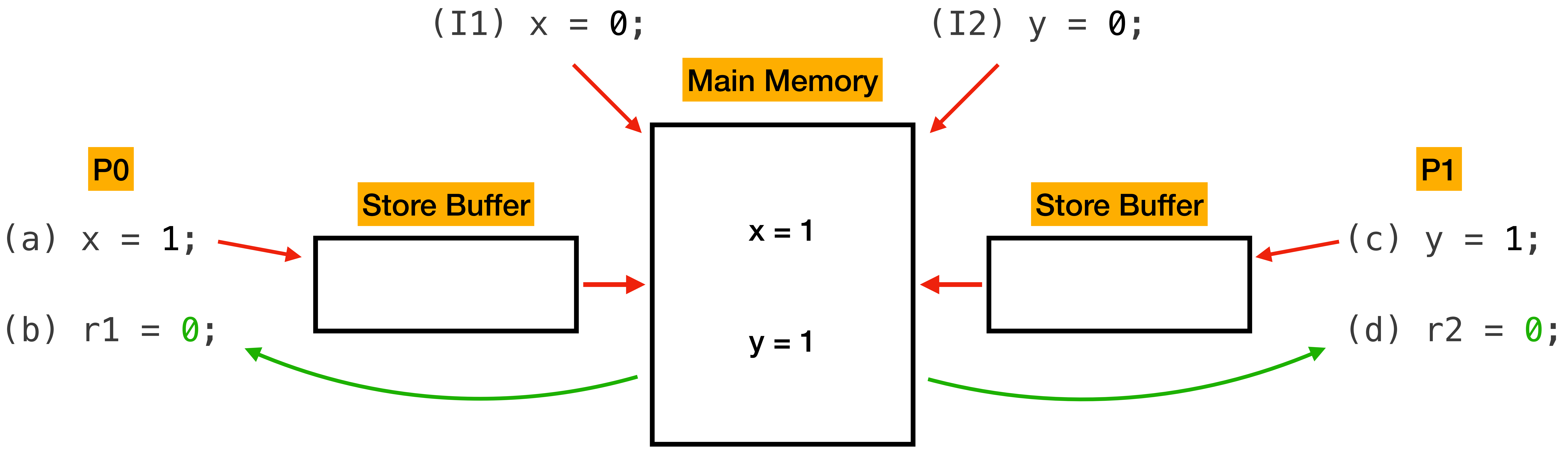
Problems so far

Interleavings are insufficient to capture program behavior

- Enrich interleavings with microarchitectural steps (e.g., fetch, execute, write-back)?
  - Requires all details of the memory architecture: too complex too quickly!
    - Forget about architectural details!
      Directly model observable behavior!
      - Axiomatic program/memory semantics!

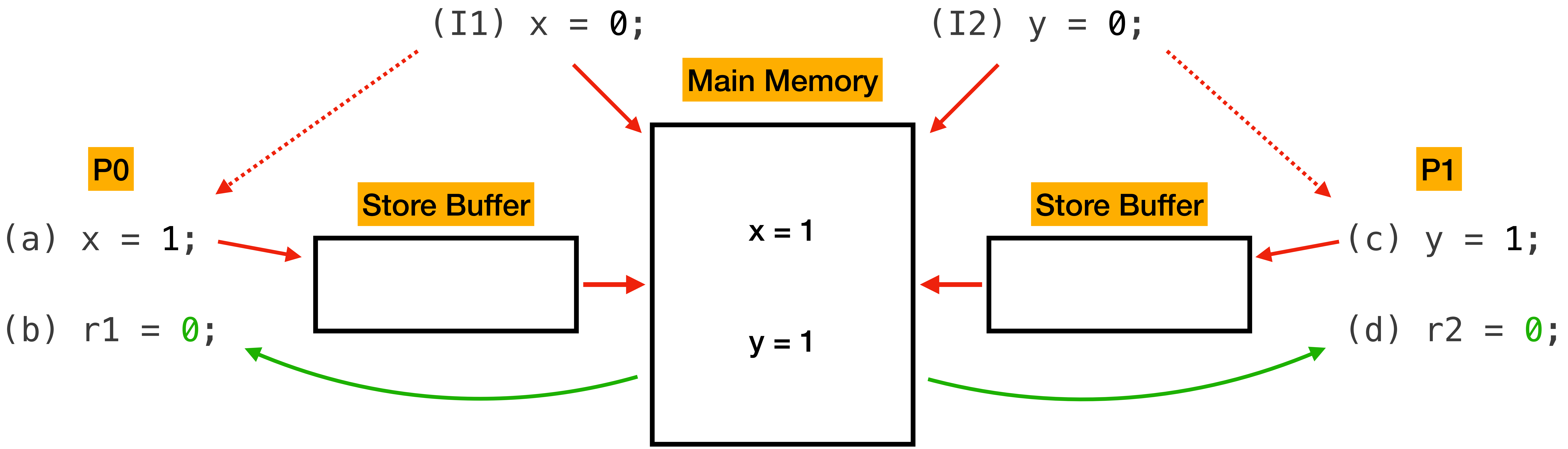
Axiomatic program semantics

Example



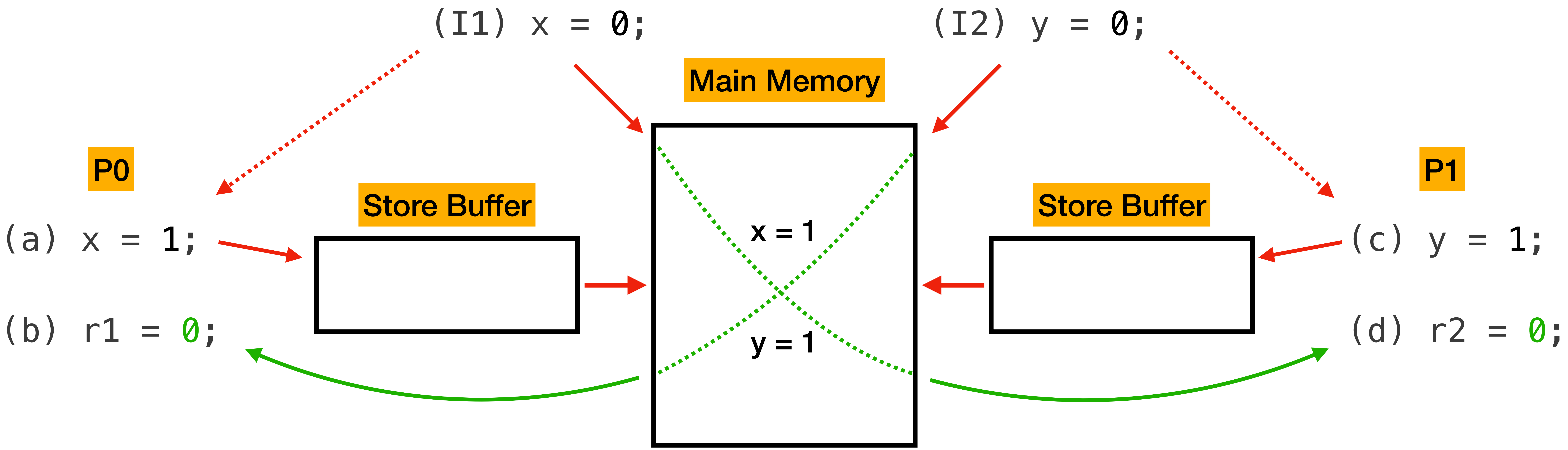
Axiomatic program semantics

Example



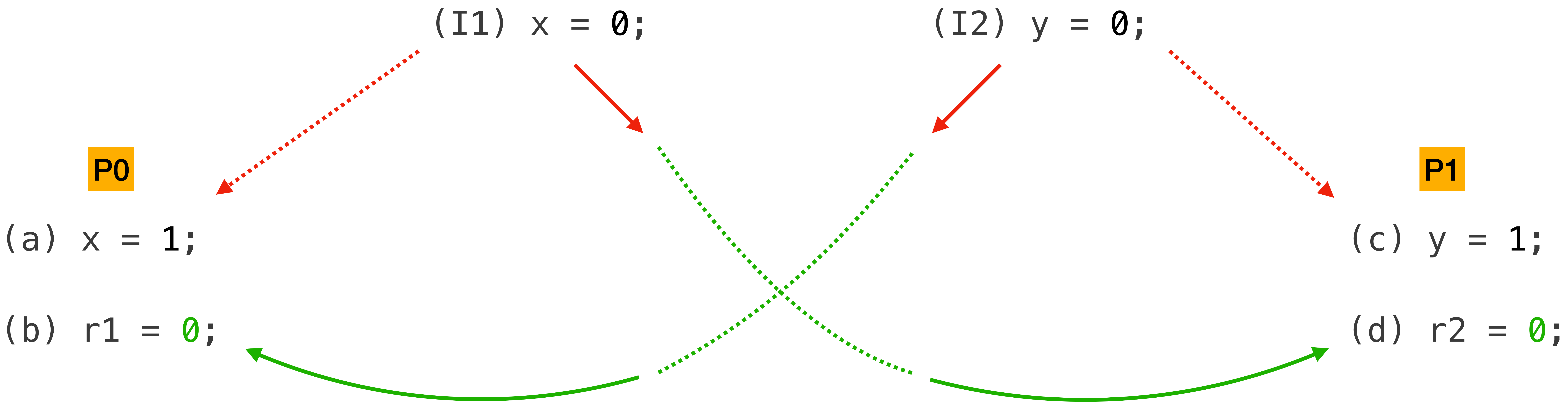
Axiomatic program semantics

Example



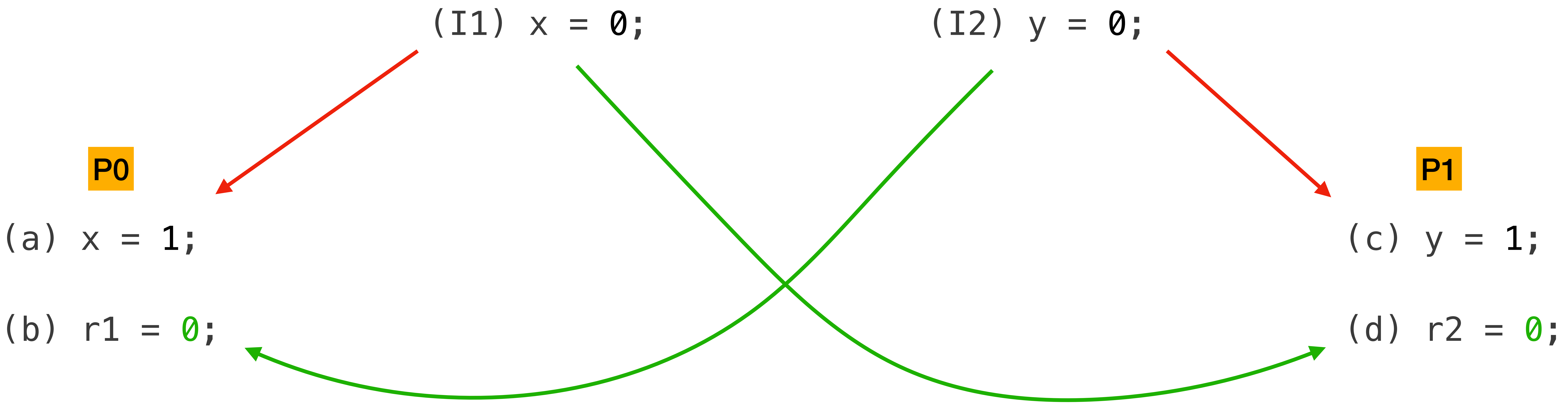
Axiomatic program semantics

Example



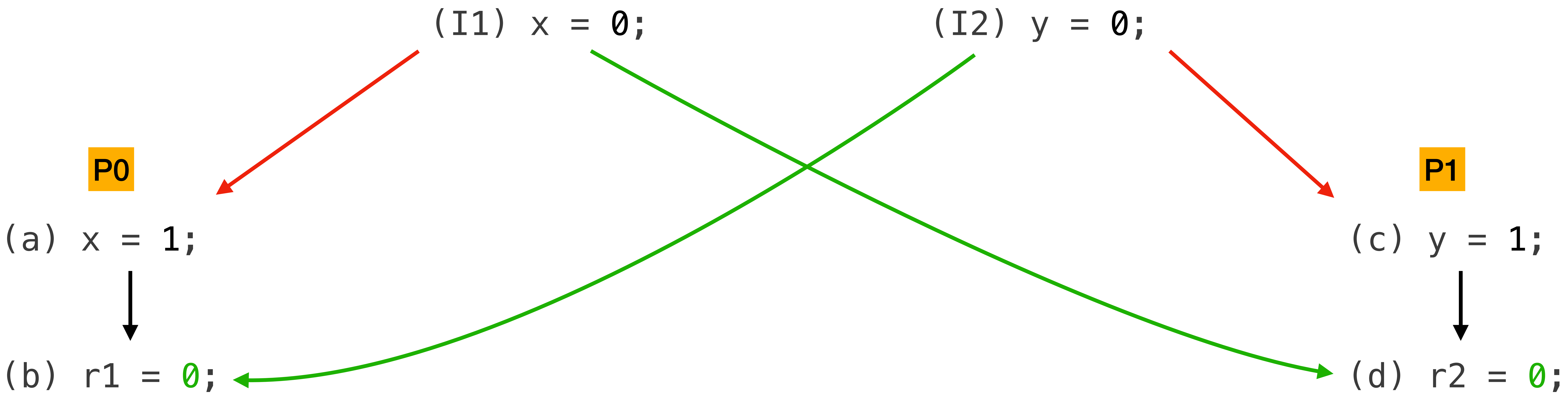
Axiomatic program semantics

Example



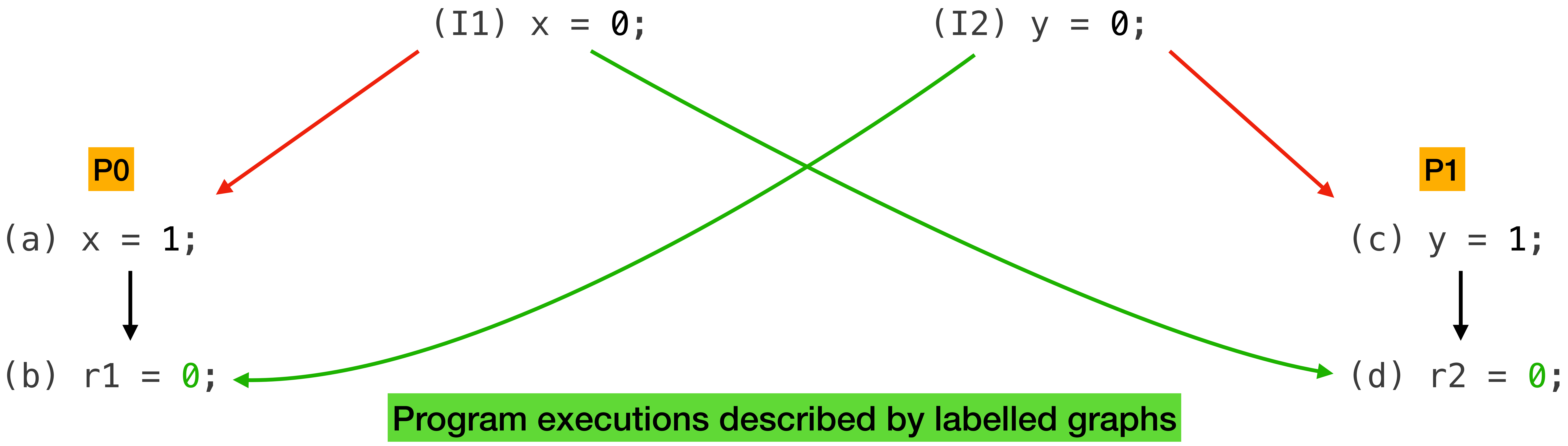
Axiomatic program semantics

Example



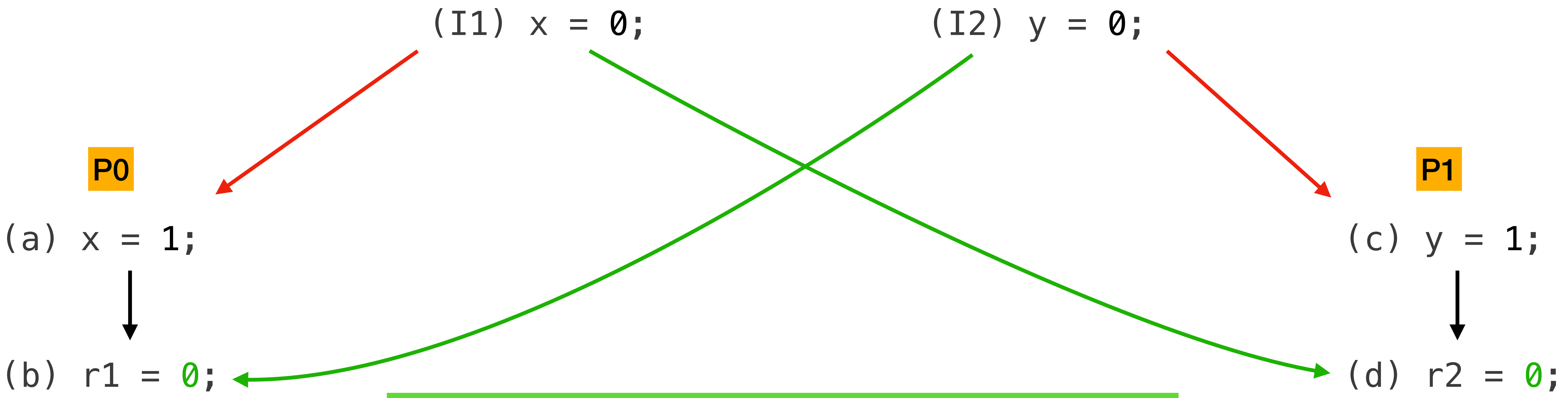
Axiomatic program semantics

Example



Axiomatic program semantics

Example

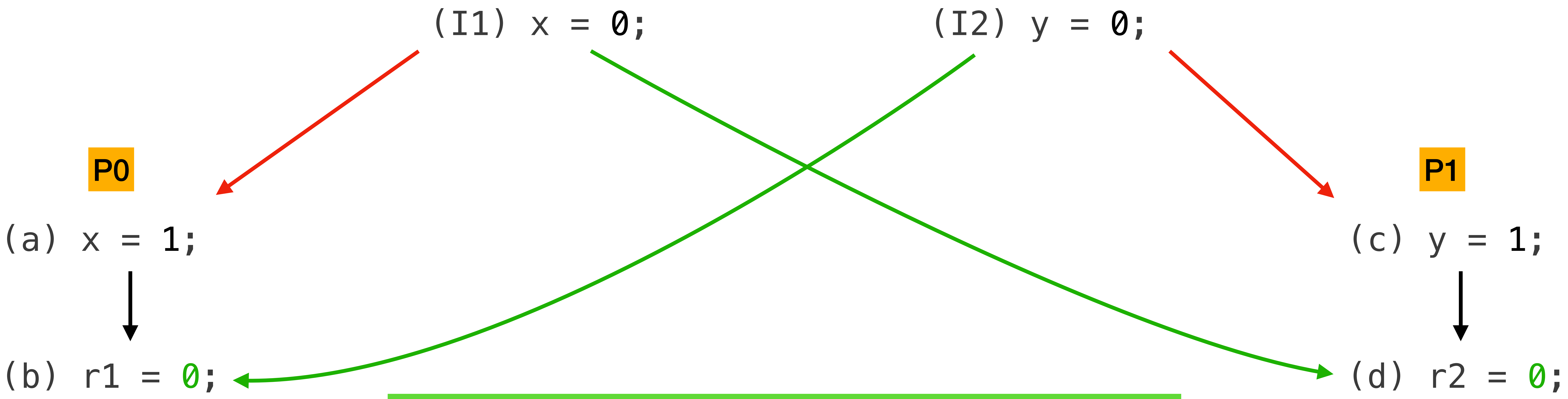


Program executions described by labelled graphs

No more interleavings!

Axiomatic program semantics

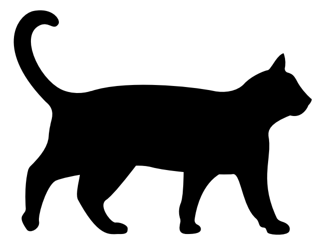
Example



Program executions described by labelled graphs

No more interleavings!

No details of the memory architecture!



Consistency models with CAT*

*
[Jade Alglave](), [Luc Maranget](), [Michael Tautschnig]():
Herding Cats: Modelling, Simulation, Testing, and Data Mining for Weak Memory. ACM Trans. Program. Lang. Syst. 36(2): 7:1-7:74 (2014)

Consistency models with CAT*

*
Jade Alglave, Luc Maranget, Michael Tautschnig:
Herding Cats: Modelling, Simulation, Testing, and Data Mining for Weak Memory. ACM Trans. Program. Lang. Syst. 36(2): 7:1-7:74 (2014)

Industry Standard

Memory consistency models

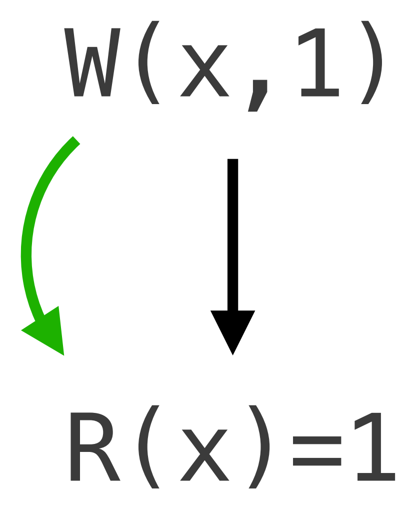
A memory consistency model answers the following question:

Memory consistency models

A memory consistency model answers the following question:
Given an anarchic execution, is it observable?

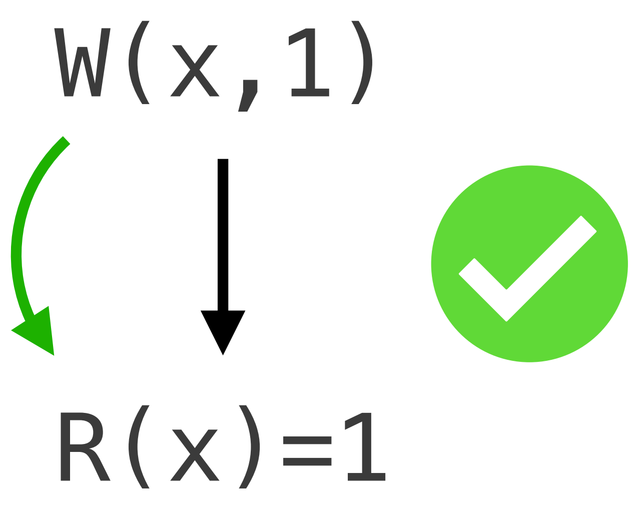
Memory consistency models

A memory consistency model answers the following question:
Given an anarchic execution, is it observable?



Memory consistency models

A memory consistency model answers the following question:
Given an anarchic execution, is it observable?

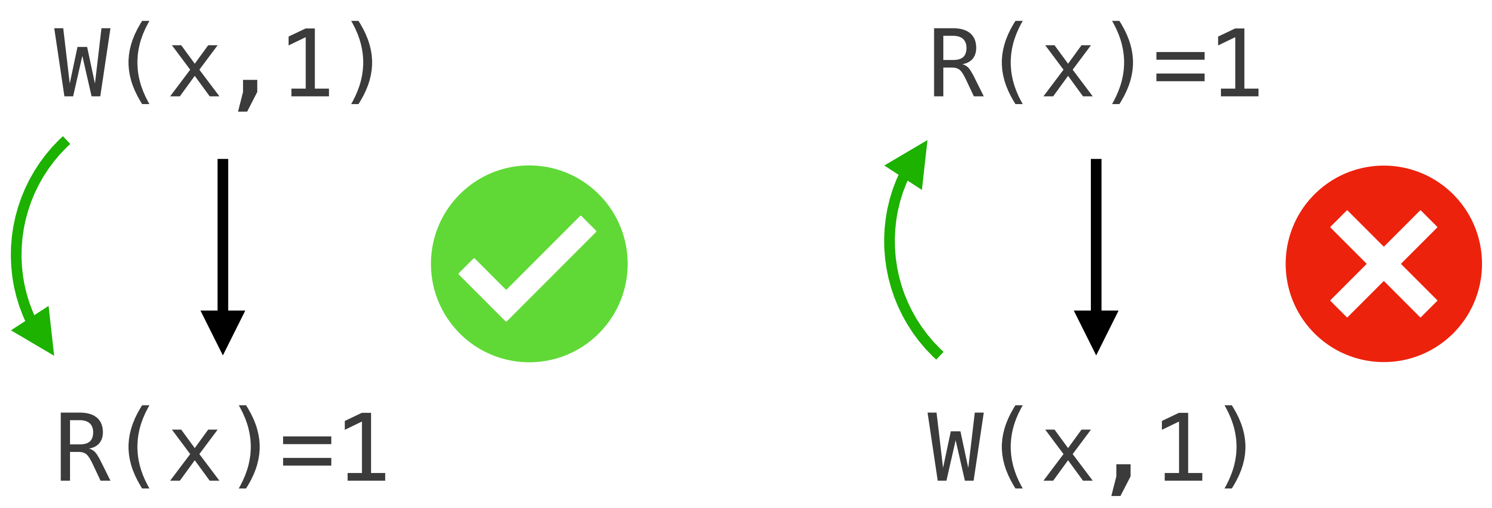


Memory consistency models

A memory consistency model answers the following question:
Given an anarchic execution, is it observable?

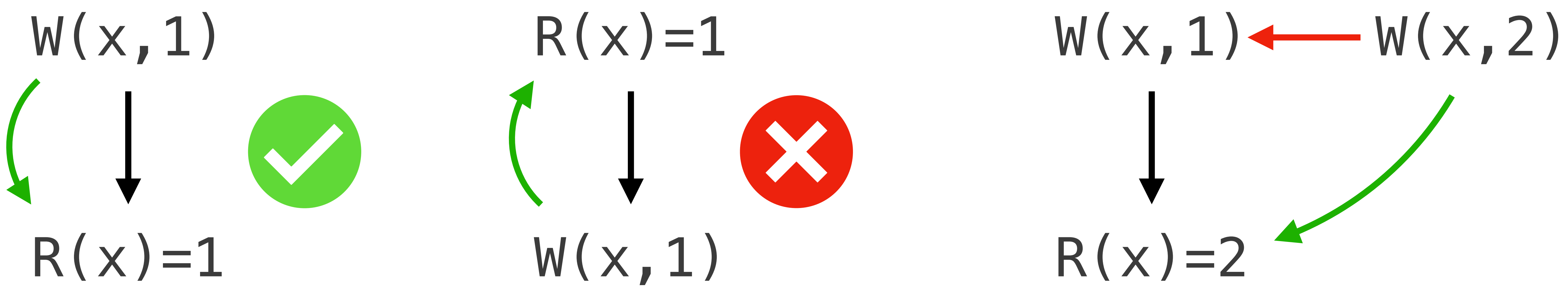
Memory consistency models

A memory consistency model answers the following question:
Given an anarchic execution, is it observable?



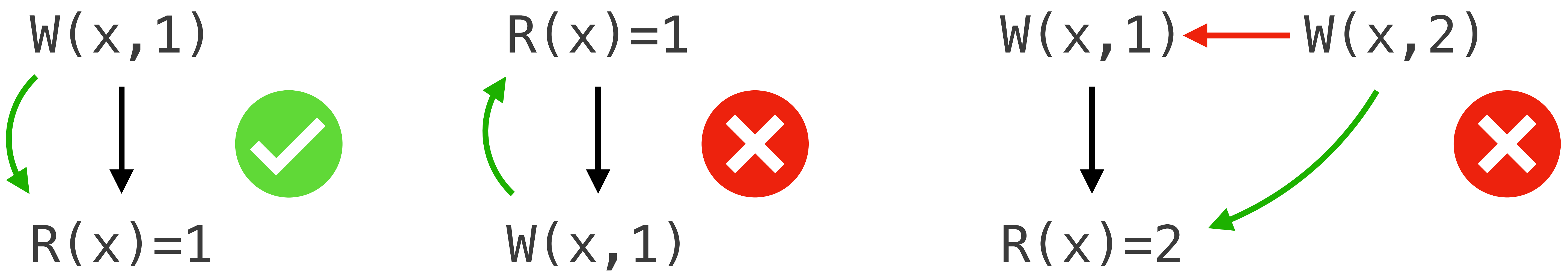
Memory consistency models

A memory consistency model answers the following question:
Given an anarchic execution, is it observable?



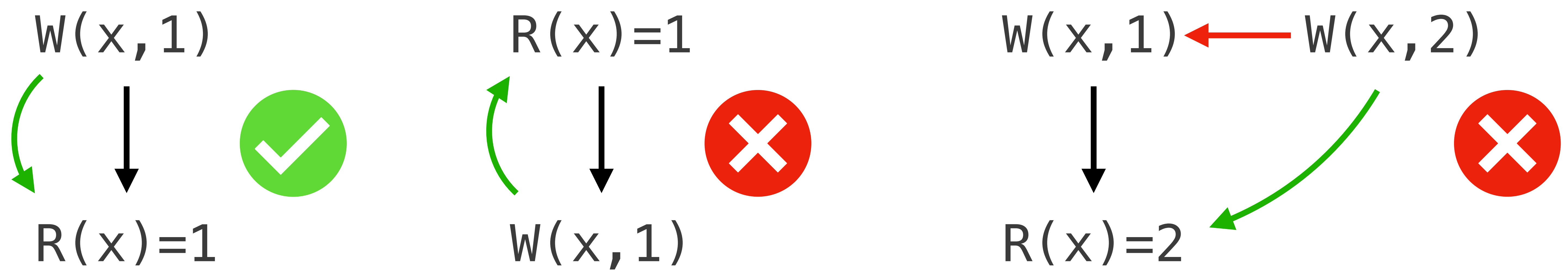
Memory consistency models

A memory consistency model answers the following question:
Given an anarchic execution, is it observable?



Memory consistency models

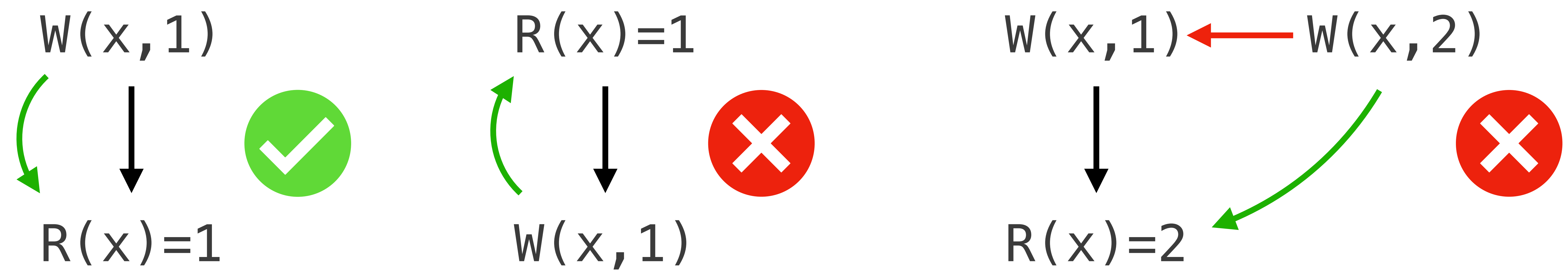
A memory consistency model answers the following question:
Given an anarchic execution, is it observable?



The CAT language is used to formulate memory consistency models

Memory consistency models

A memory consistency model answers the following question:
Given an anarchic execution, is it observable?



The CAT language is used to formulate memory consistency models

Restrict the shape (events & relations) of executions

Memory consistency models

The CAT language

CAT uses existing (base) relations

Memory consistency models

The CAT language

CAT uses existing (base) relations

Base relations

Memory consistency models

The CAT language

CAT uses existing (base) relations to define new ones (derived)

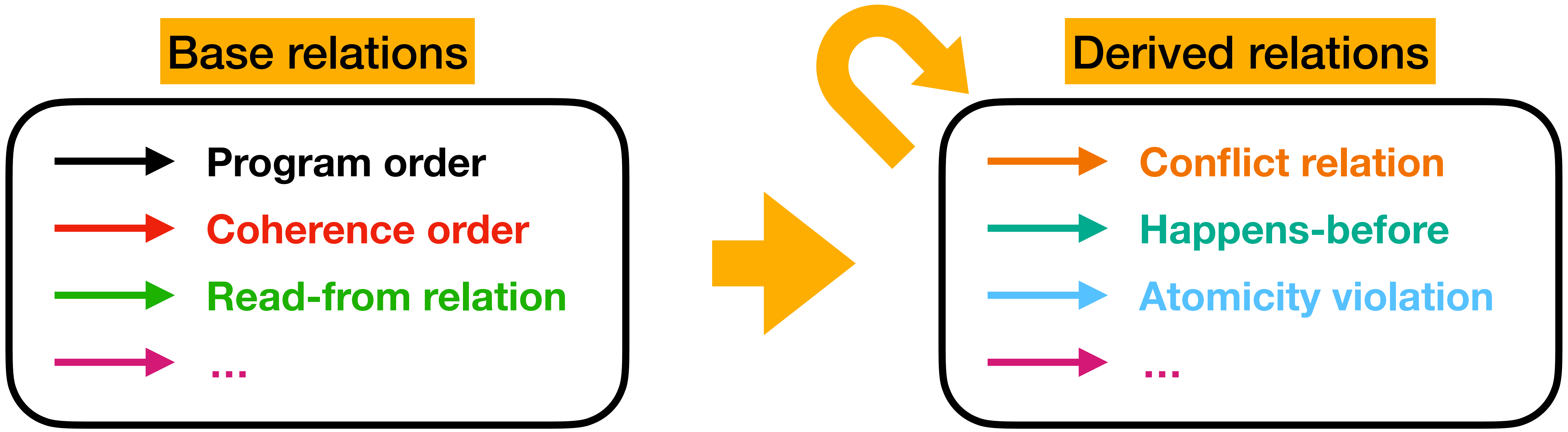
Base relations

- → Program order
- → Coherence order
- → Read-from relation
- → …

Memory consistency models

The CAT language

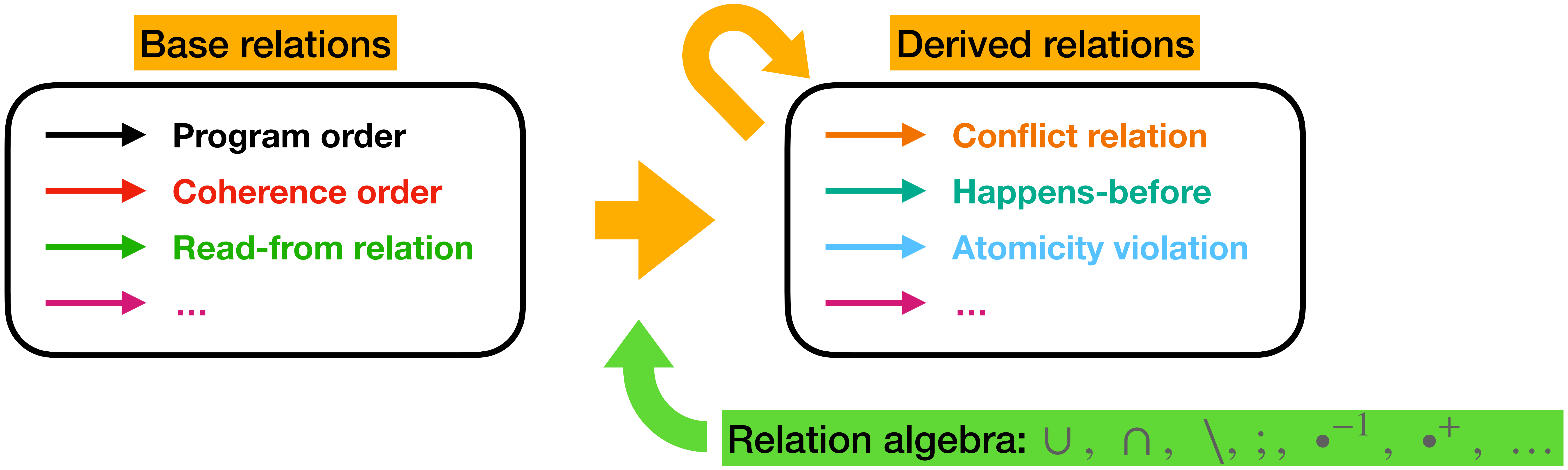
CAT uses existing (base) relations to define new ones (derived)



Memory consistency models

The CAT language

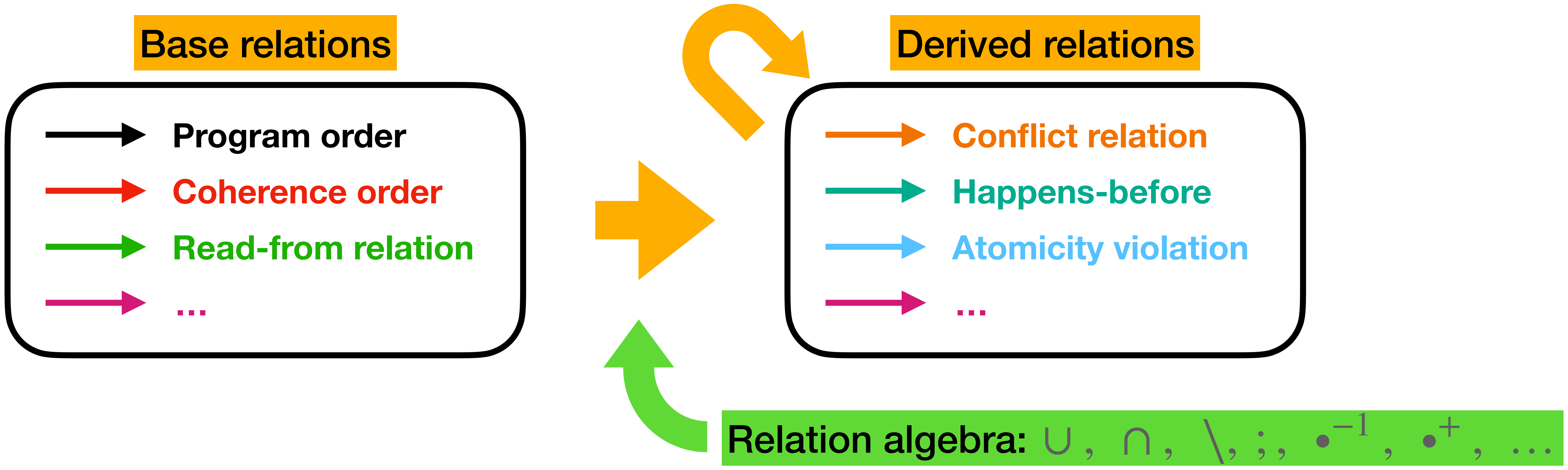
CAT uses existing (base) relations to define new ones (derived)



Memory consistency models
The CAT language

CAT uses existing (base) relations to define new ones (derived)

CAT puts constraints on relations, happens-before has to be acyclic

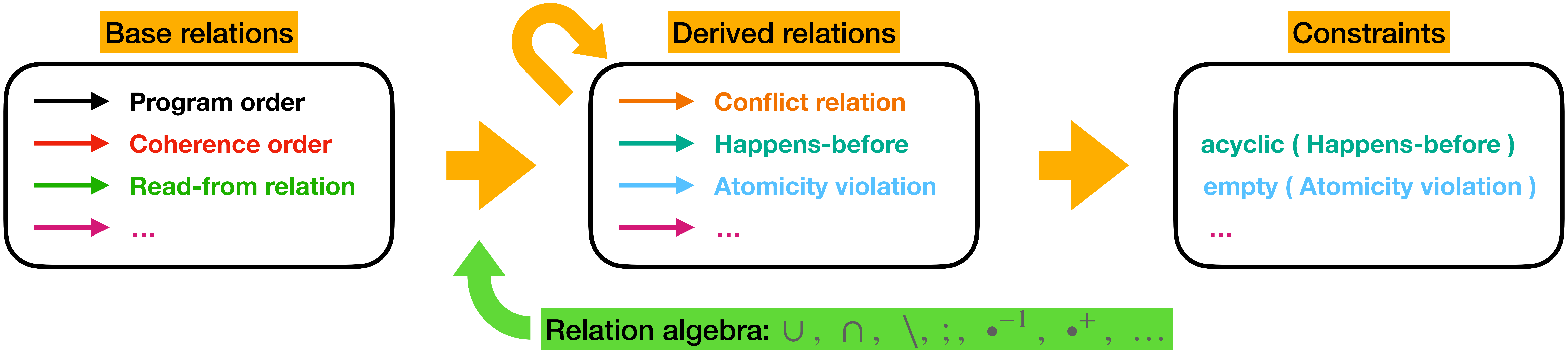


Memory consistency models

The CAT language

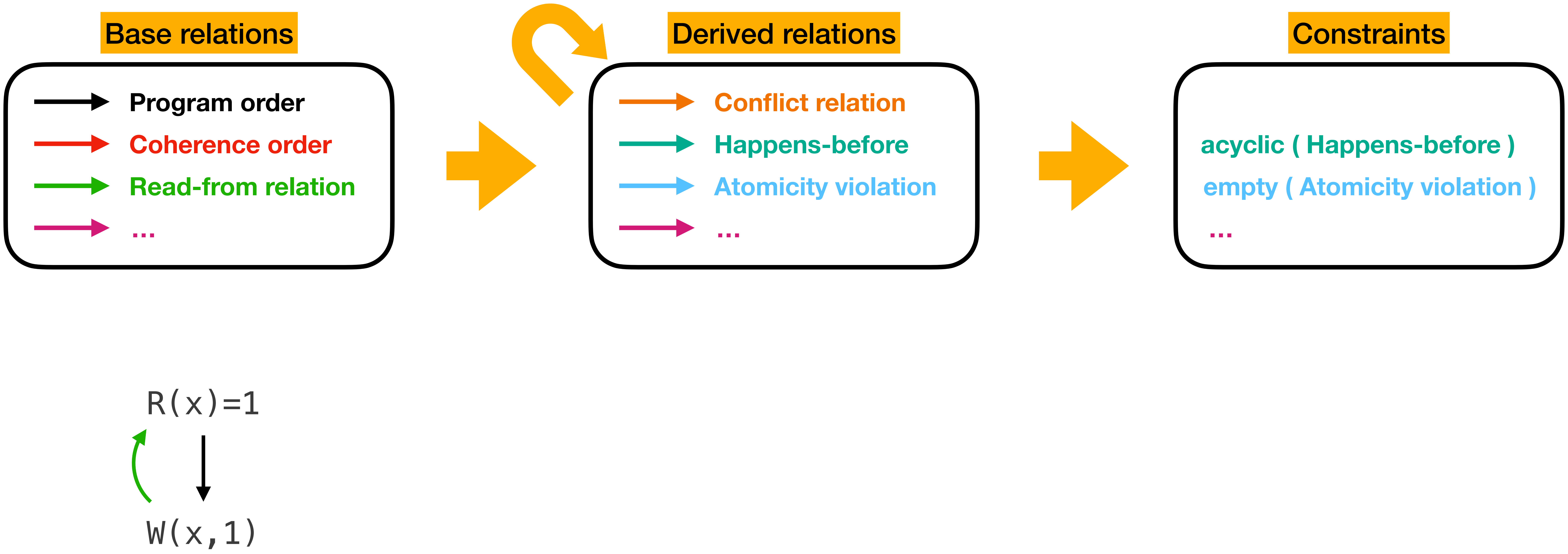
CAT uses existing (base) relations to define new ones (derived)

CAT puts constraints on relations, happens-before has to be acyclic



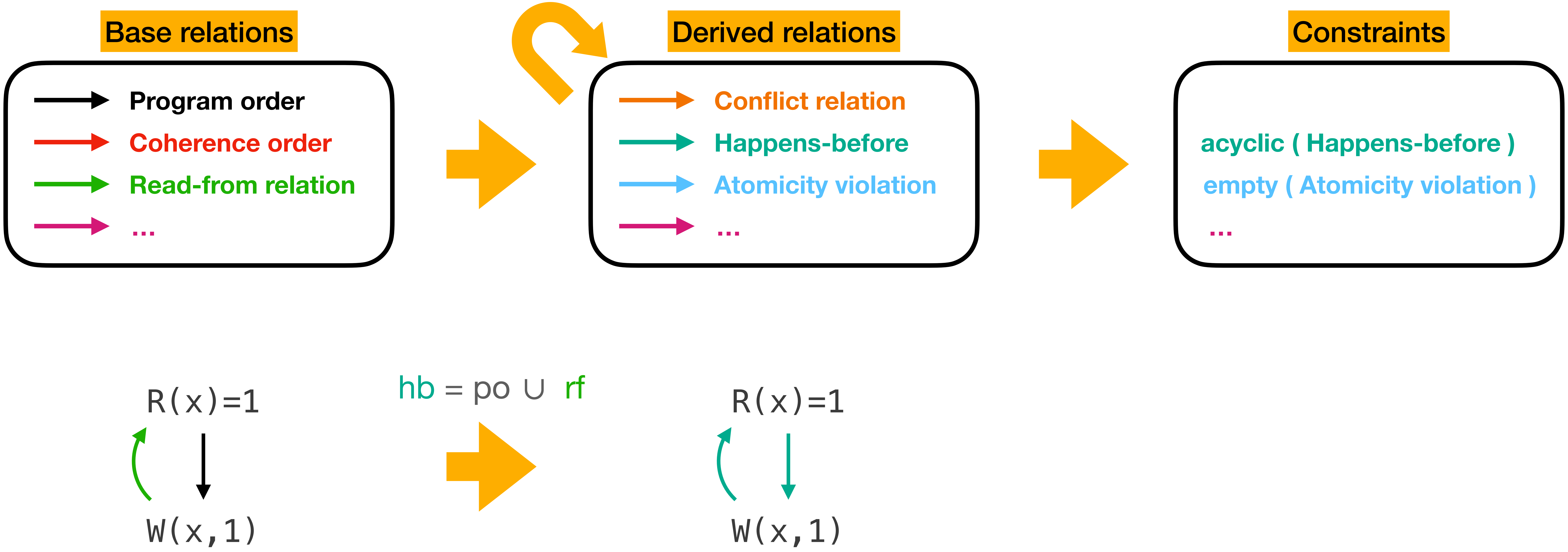
Memory consistency models

The CAT language (example)



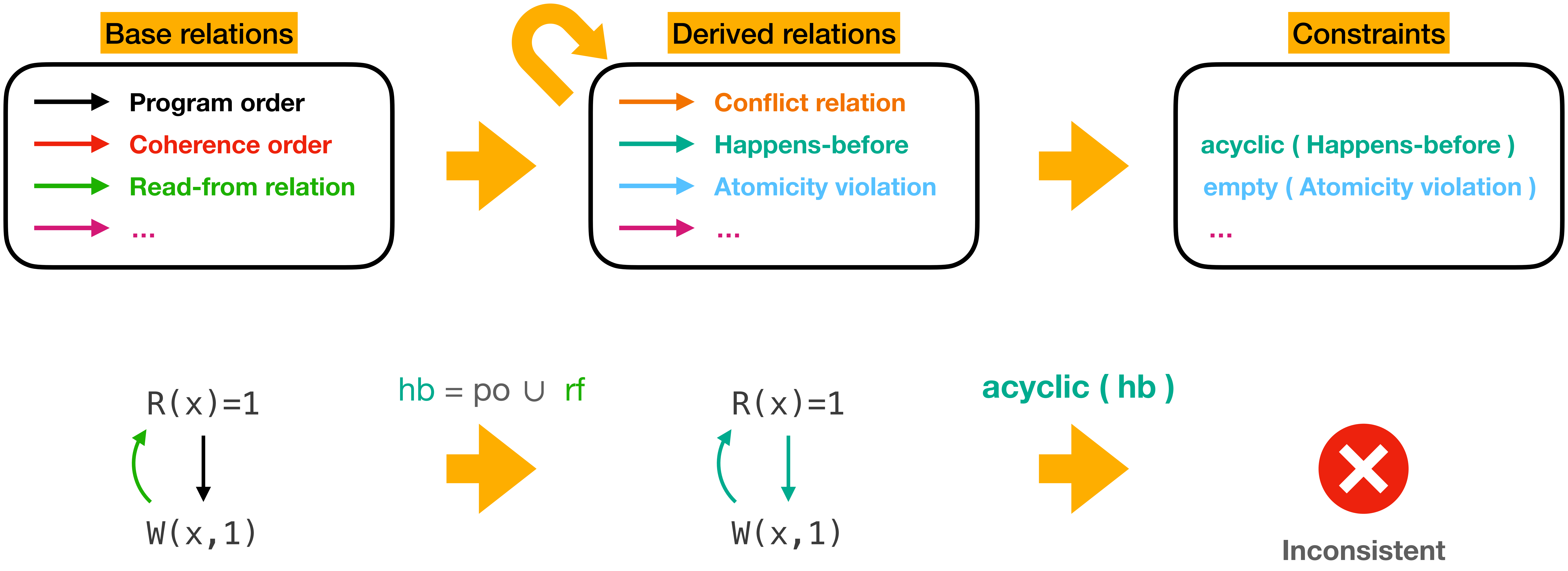
Memory consistency models

The CAT language (example)



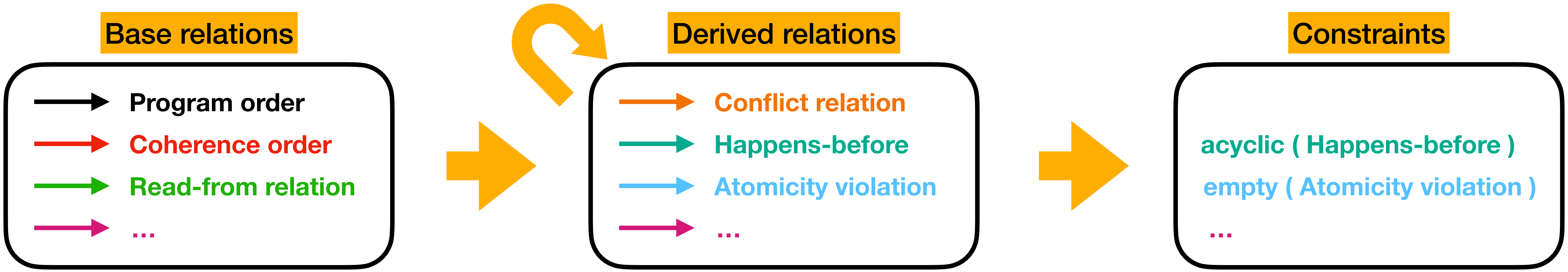
Memory consistency models

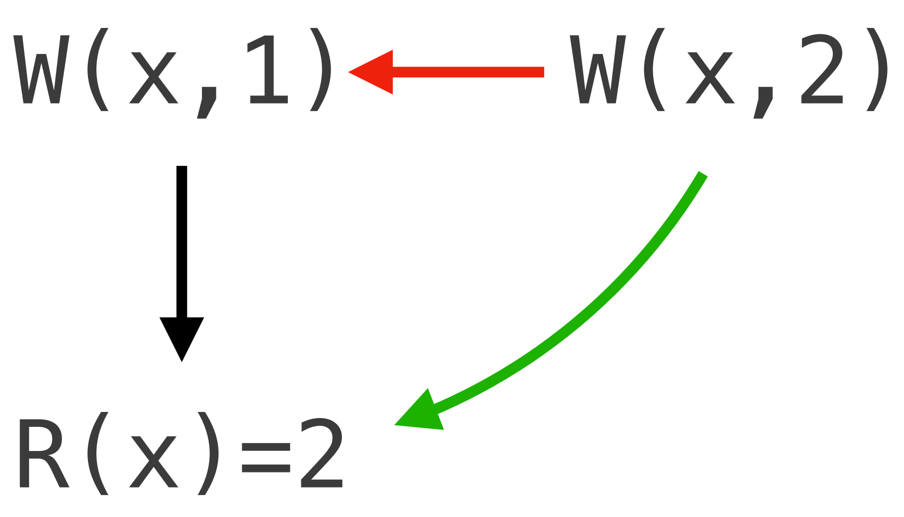
The CAT language (example)



Memory consistency models

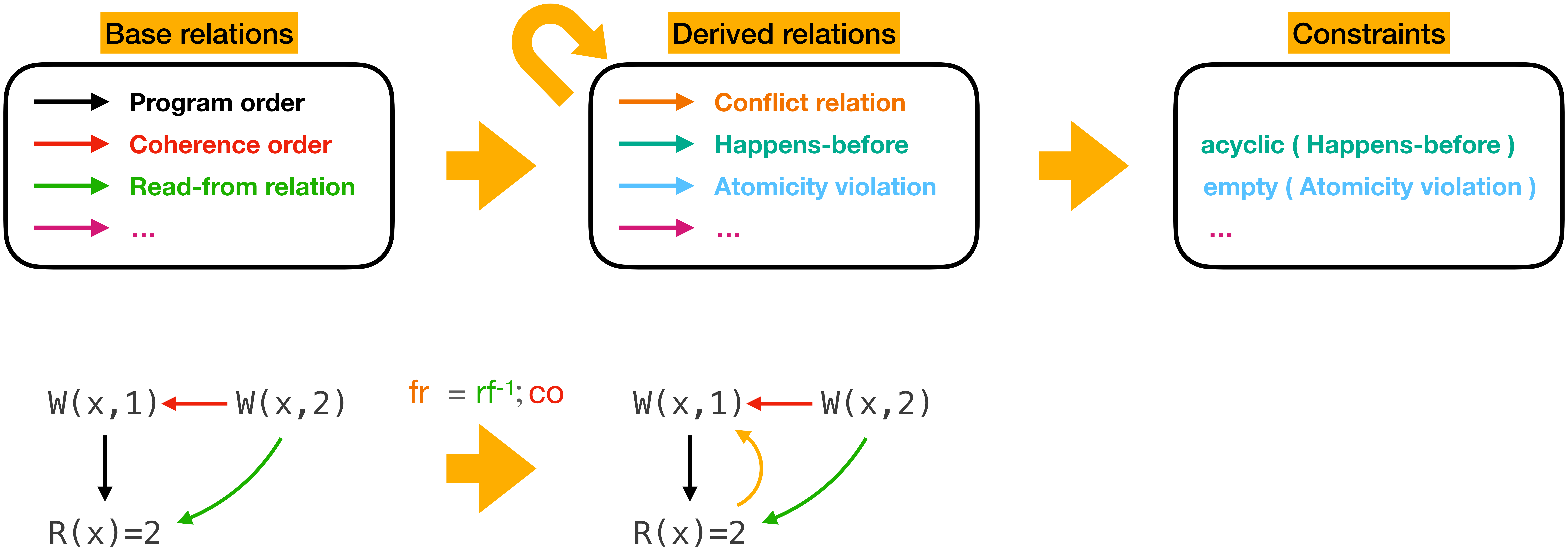
The CAT language (example)





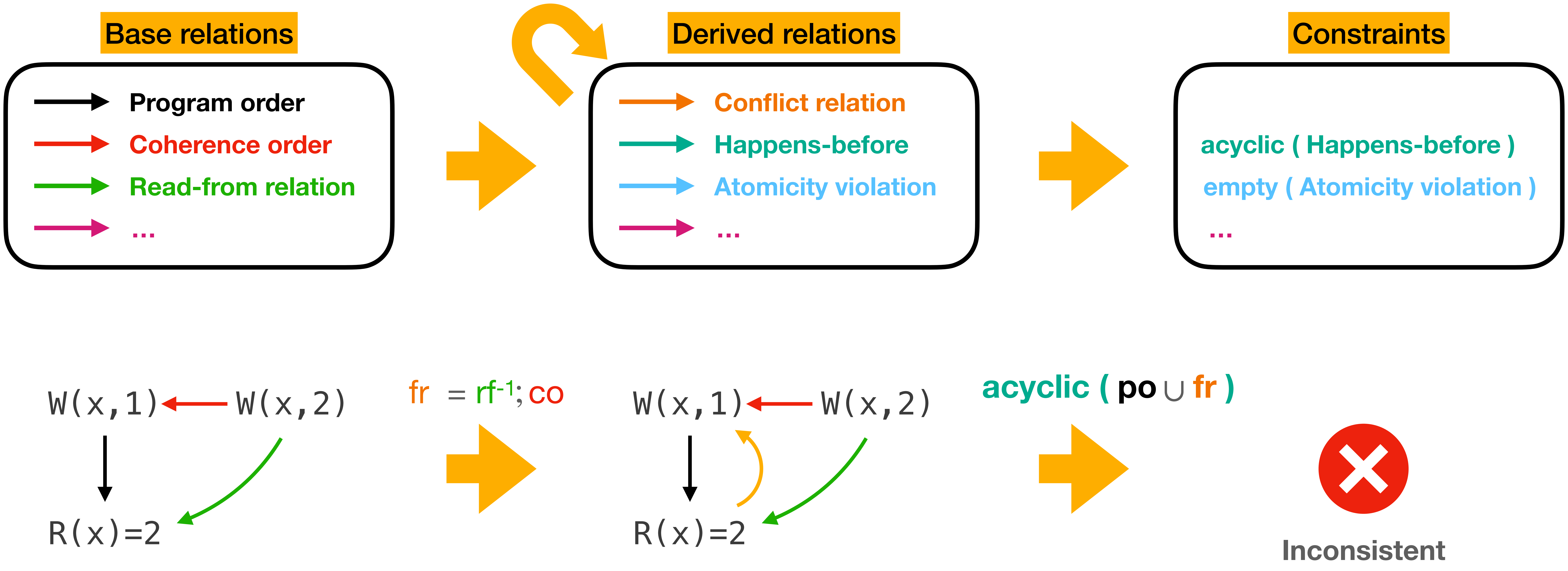
Memory consistency models

The CAT language (example)



Memory consistency models

The CAT language (example)



Memory consistency models

Outlook

Memory consistency models
Outlook

Beyond hardware memory architectures!

Memory consistency models

Outlook

Beyond hardware memory architectures!

(A) Language-level memory models (C11, LKMM, Java, …)

Memory consistency models

Outlook

Beyond hardware memory architectures!

(A) Language-level memory models (C11, LKMM, Java, …)

Compiler optimisations + compiler mappings

Memory consistency models
Outlook

Beyond hardware memory architectures!

(A) Language-level memory models (C11, LKMM, Java, …)

Compiler optimisations + compiler mappings

Library specifications: RCU, pthread, safe memory reclamation, …

Memory consistency models

Outlook

Beyond hardware memory architectures!

(A) Language-level memory models (C11, LKMM, Java, …)

Compiler optimisations + compiler mappings

Library specifications: RCU, pthread, safe memory reclamation, …

(B) Distributed systems (~ communication protocols)

Memory consistency models

Outlook

Beyond hardware memory architectures!

(A) Language-level memory models (C11, LKMM, Java, …)

Compiler optimisations + compiler mappings

Library specifications: RCU, pthread, safe memory reclamation, …

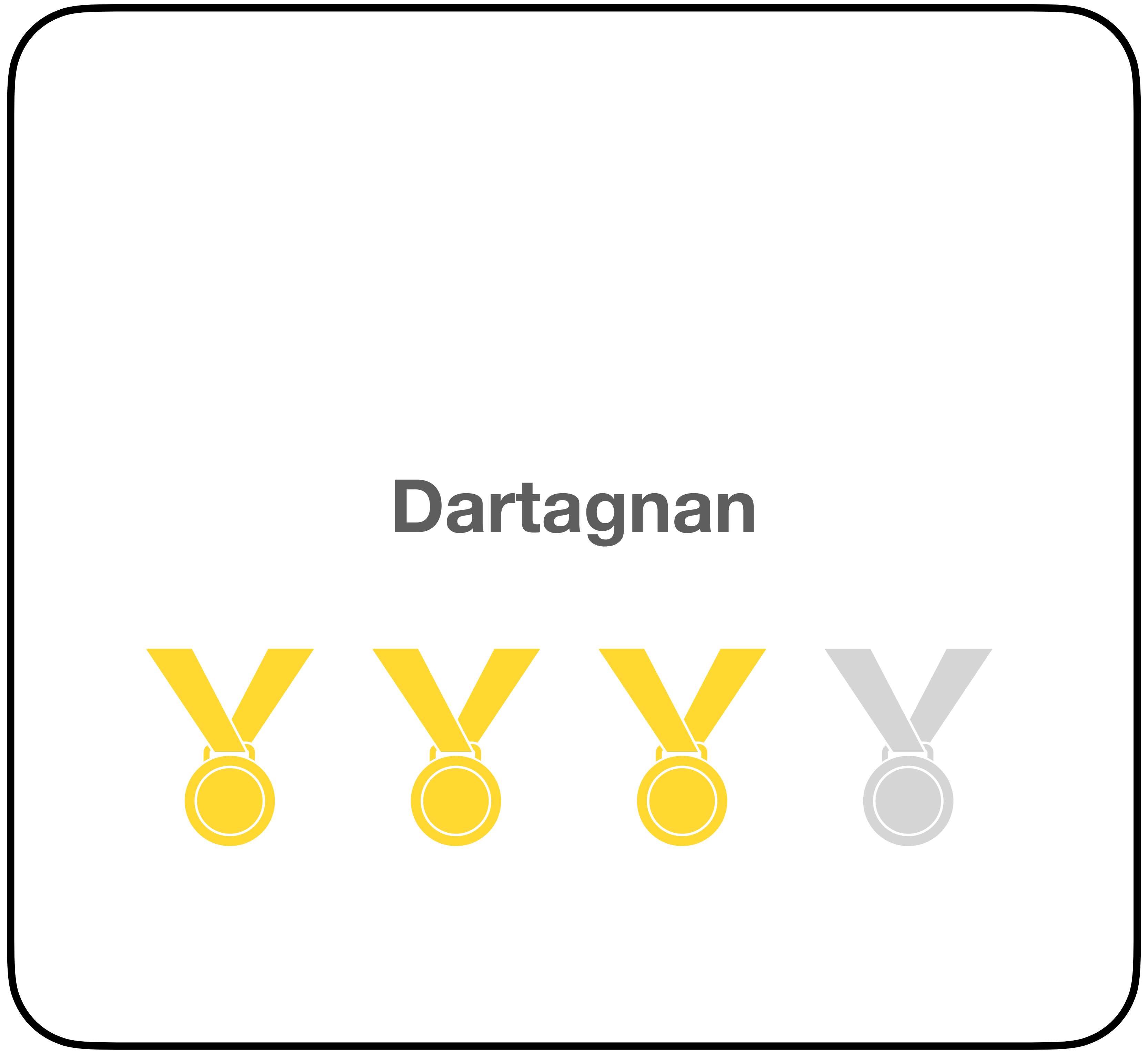
(B) Distributed systems (~ communication protocols)

(C) Databases (~ database isolation levels)

Dartagnan

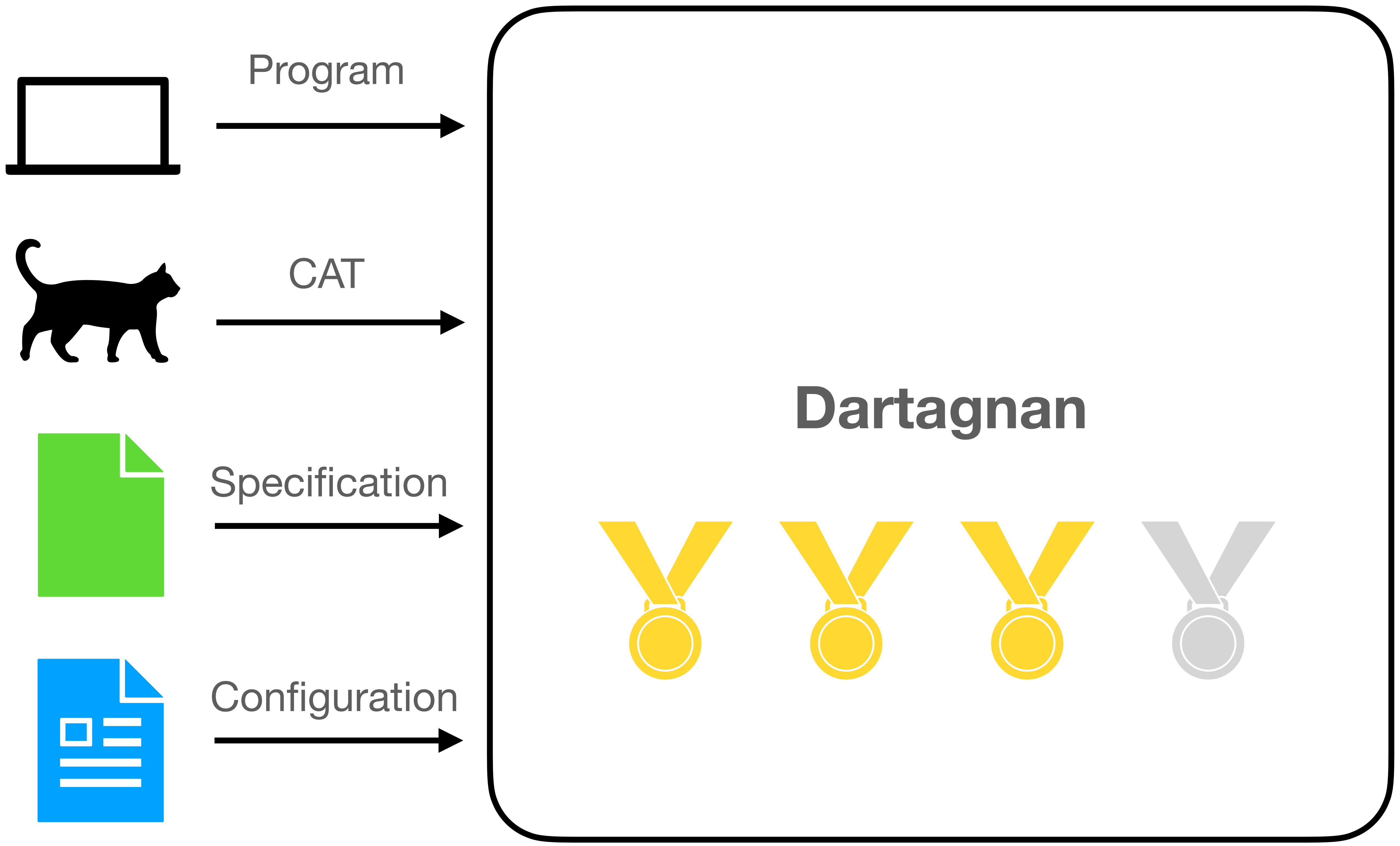
Dartagnan

Model checking real code



Dartagnan

Model checking real code

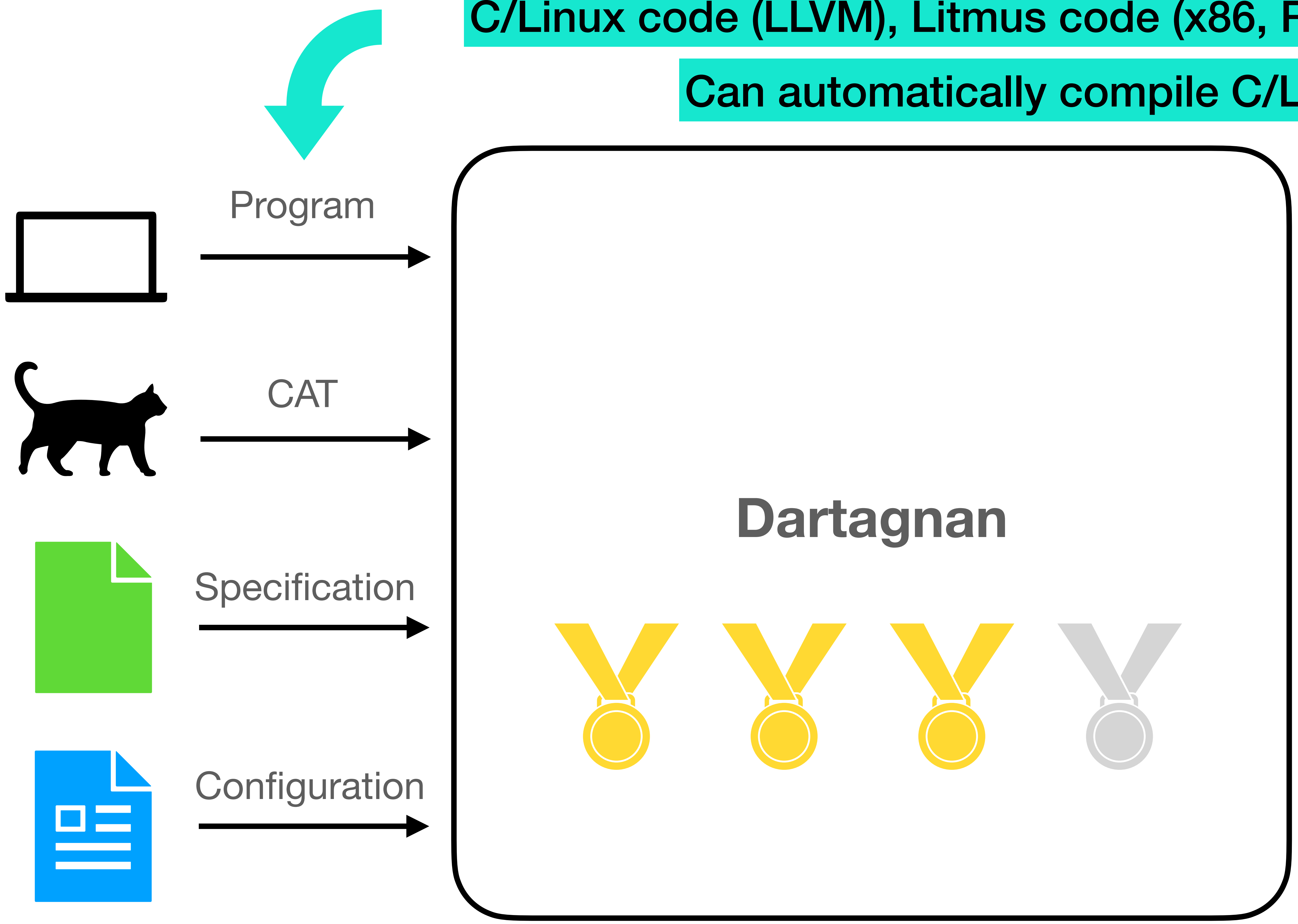


Dartagnan

Model checking real code

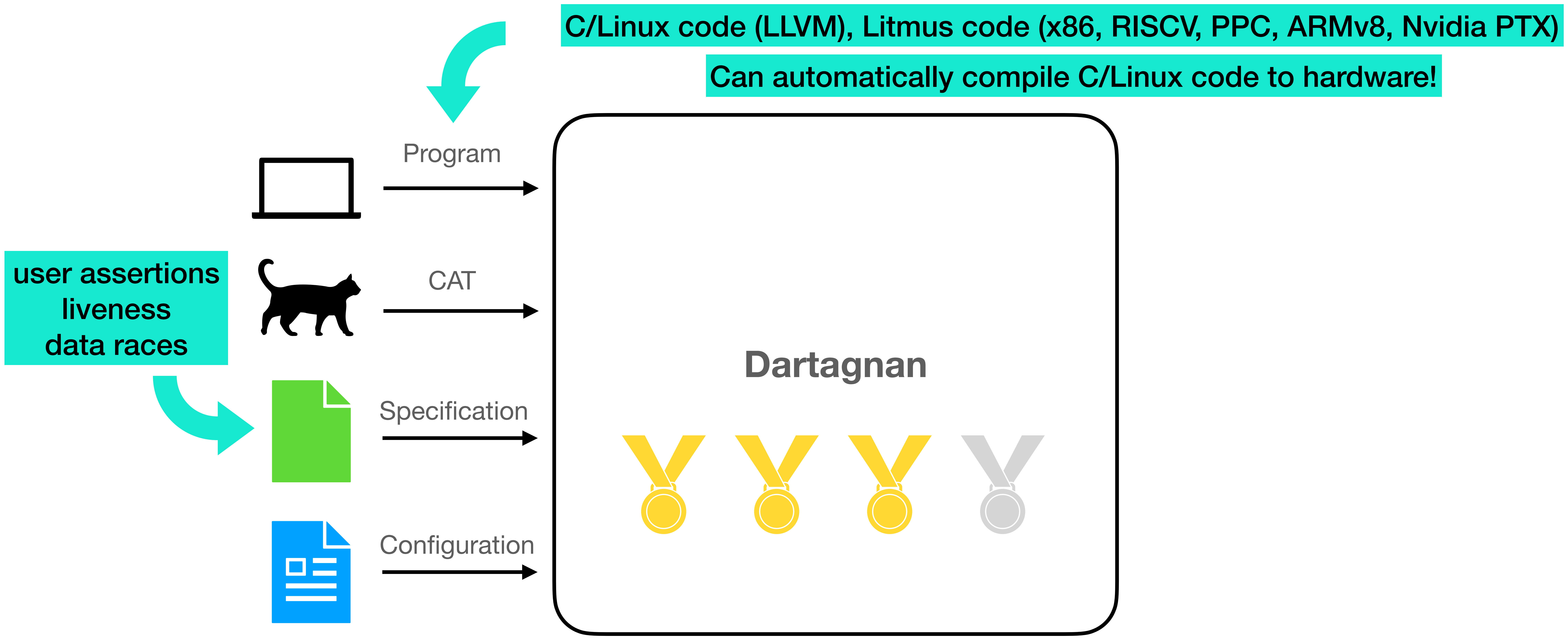
C/Linux code (LLVM), Litmus code (x86, RISCV, PPC, ARMv8, Nvidia PTX)

Can automatically compile C/Linux code to hardware!



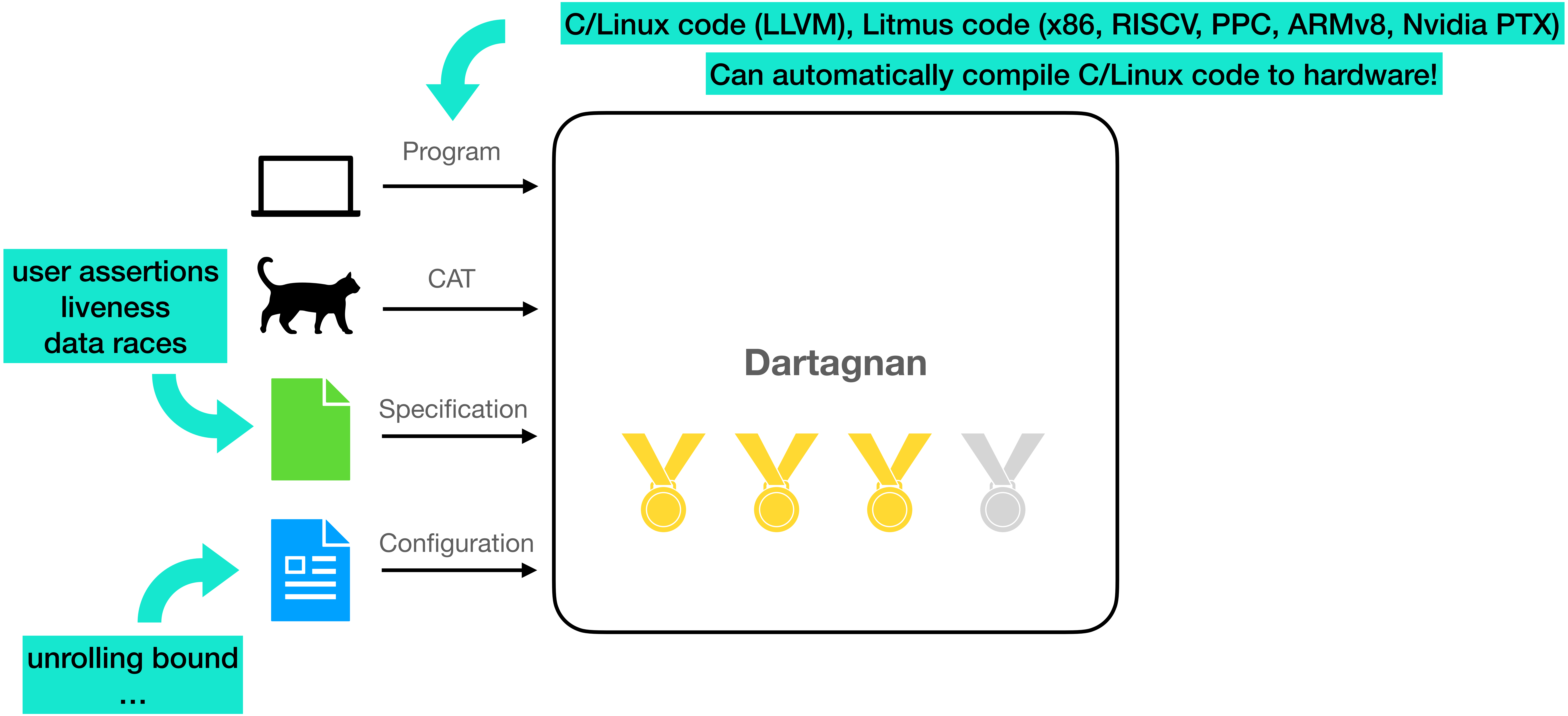
Dartagnan

Model checking real code



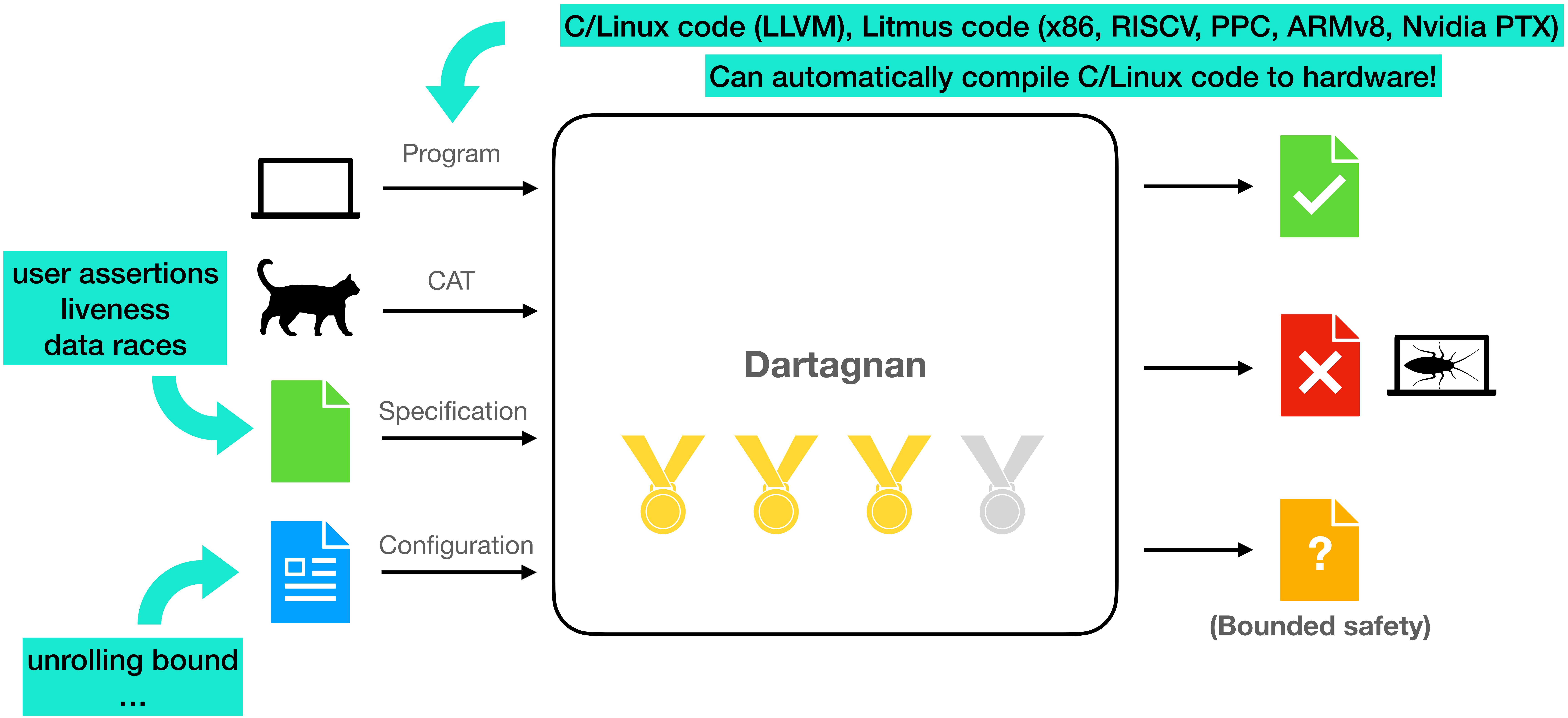
Dartagnan

Model checking real code



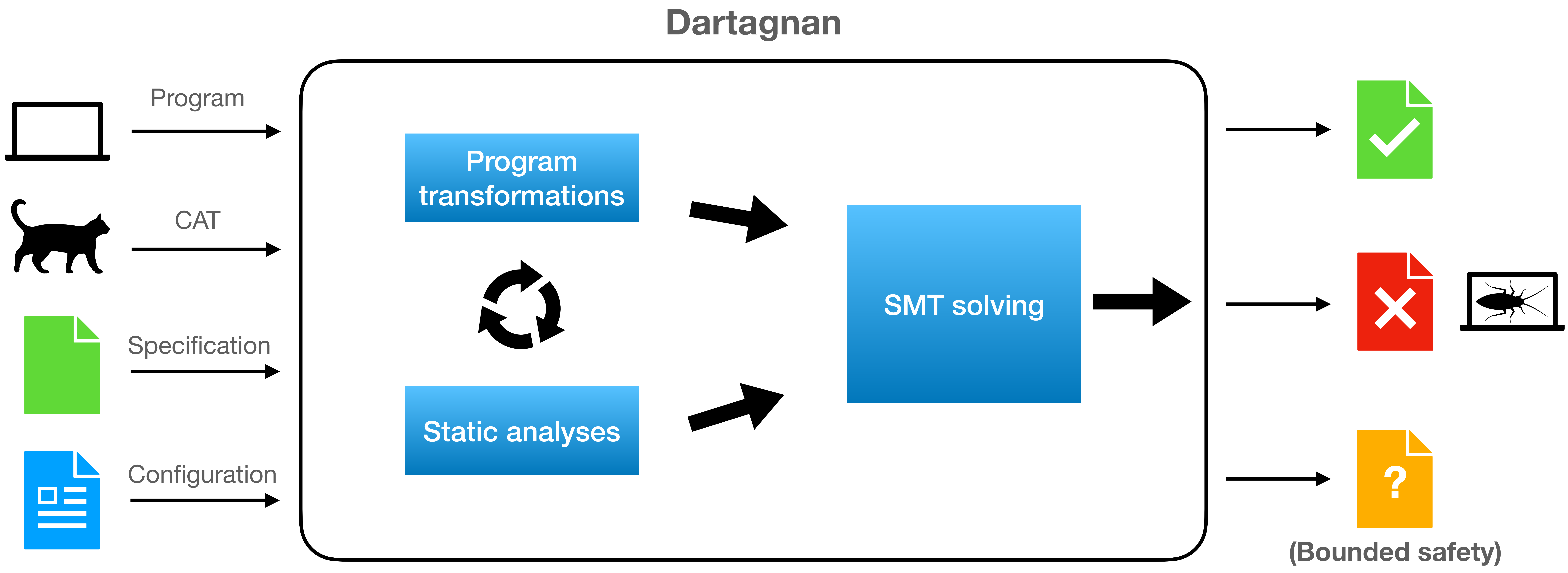
Dartagnan

Model checking real code

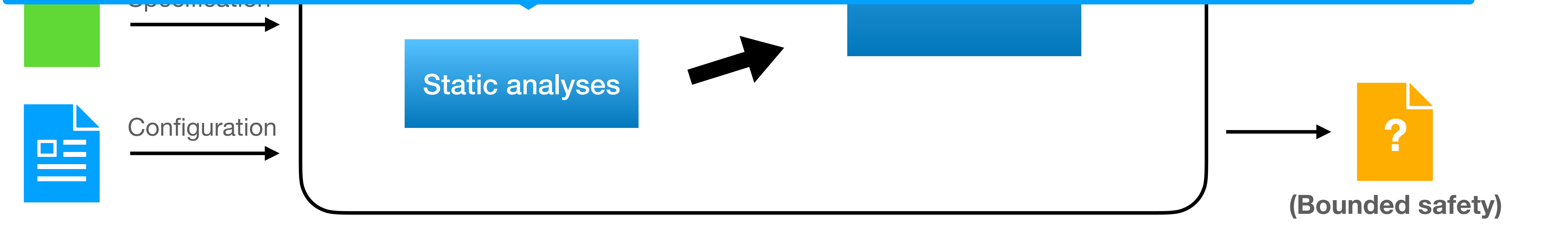


Dartagnan

Internals

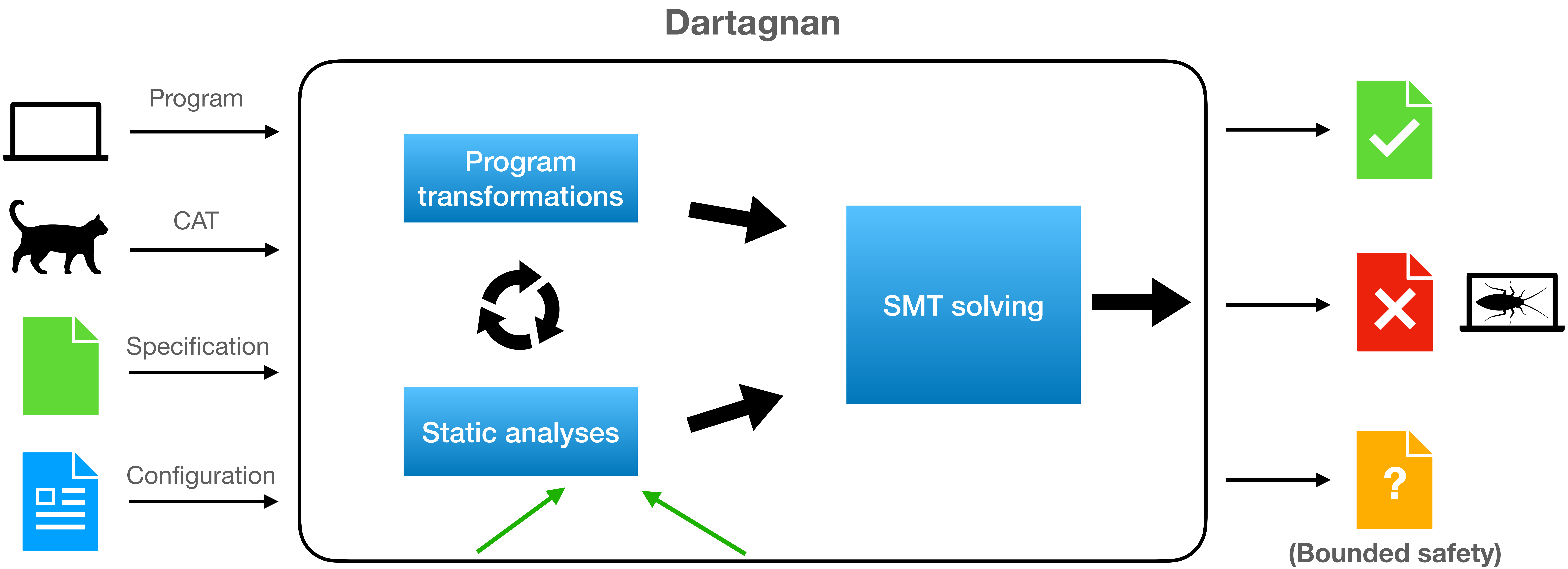


Alias Analysis
Control-Flow Analysis (find basic blocks of instructions executed together, control-flow variables)
Constant Propagation
Def-Use-Analysis
Dominator Analysis
Expression Simplification
Function Call Devirtualization (resolve call targets)
Function Inlining
Live Variables
Loop Unrolling
Mem2Reg (treat stack as registers)
Normalize Loops (single backjump, single entry)
Reaching Definitions
Sparse Conditional Constant Propagation (constant propagation + dead code elimination)
Spin Loop Detection and Instrumentation for Dynamic Detection
Symmetry Breaking



Dartagnan

Internals



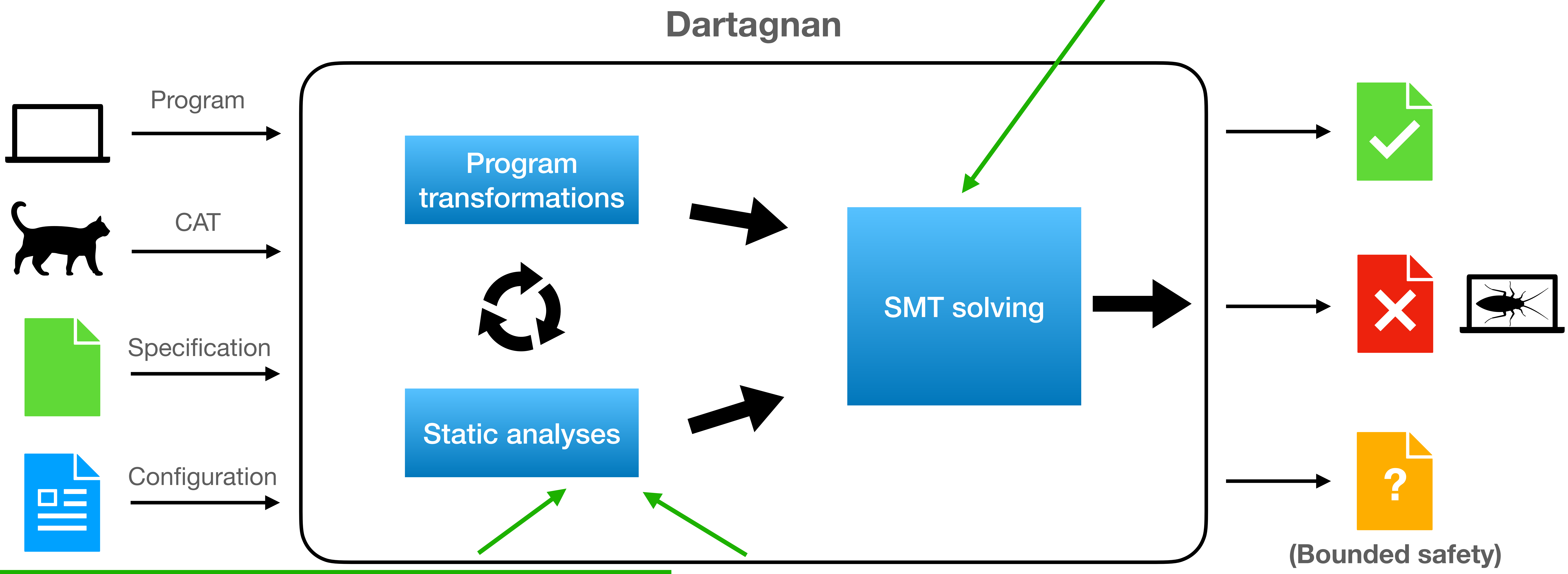
Natalia Gavrilenko, Hernán Ponce de León, Florian Furbach, Keijo Heljanko, Roland Meyer: *BMC for Weak Memory Models: Relation Analysis for Compact SMT Encodings @ CAV19*

Thomas Haas, René Maseli, Roland Meyer, Hernán Ponce de León: *Static Analysis of Memory Models for SMT Encodings @ OOPSLA23*

Dartagnan

Internals

Thomas Haas, Roland Meyer, Hernán Ponce de León: *CAAT: consistency as a theory @ OOPSLA22*



Natalia Gavrilenko, Hernán Ponce de León, Florian Furbach, Keijo Heljanko, Roland Meyer: *BMC for Weak Memory Models: Relation Analysis for Compact SMT Encodings @ CAV19*

Thomas Haas, René Maseli, Roland Meyer, Hernán Ponce de León: *Static Analysis of Memory Models for SMT Encodings @ OOPSLA23*

Encoding CAT into logical theories

Thomas Haas, Roland Meyer, Hernán Ponce de León:
CAAT: consistency as a theory @ OOPSLA22

CAT in logical theories

CAT in logical theories

- CAT has simple operations over relations: $;\,,\cup\,,\cap\,,\setminus,\bullet^{-1}$

  $\rightarrow$ Easily encodable into plain SAT (over finite domain)

CAT in logical theories

- CAT has simple operations over relations: $;\,,\cup\,,\cap\,,\backslash,\bullet^{-1}$

  → Easily encodable into plain SAT (over finite domain)

- CAT has axioms on relations: **empty**, **irreflexive**, **acyclic**

  → Emptiness and irreflexivity encodable into plain SAT;
  Acyclicity encodable into integer difference logic (SMT)

CAT in logical theories

- CAT has

→ Easily

Problem: CAT allows for (non-linear) recursive definitions with (stratified) least fixed point semantics!

- CAT has axioms on relations: **empty**, **irreflexive**, **acyclic**

→ Emptiness and irreflexivity encodable into plain SAT; Acyclicity encodable into integer difference logic (SMT)

CAT in logical theories

- CAT has

→ Easily

Problem: CAT allows for (non-linear) recursive definitions with (stratified) least fixed point semantics!

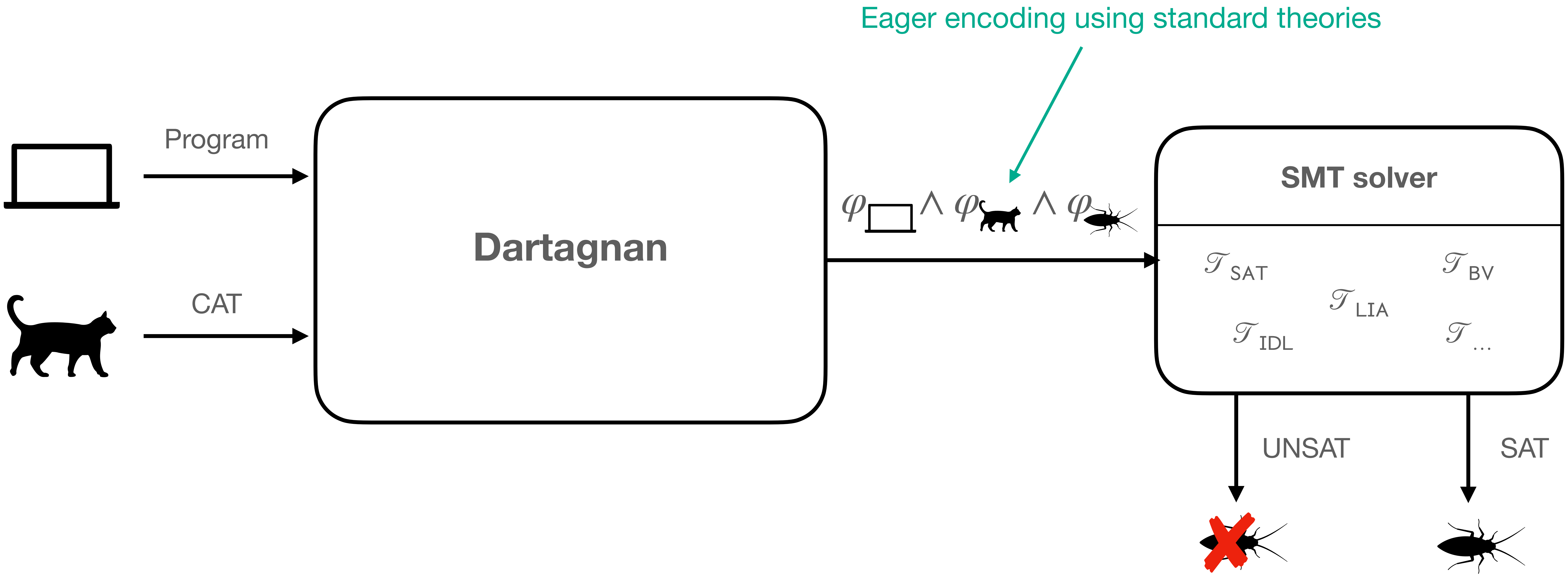
- C

Existing theories have a hard time capturing least fixed point semantics!

→ Emptiness and irreflexivity encodable into plain SAT; Acyclicity encodable into integer difference logic (SMT)

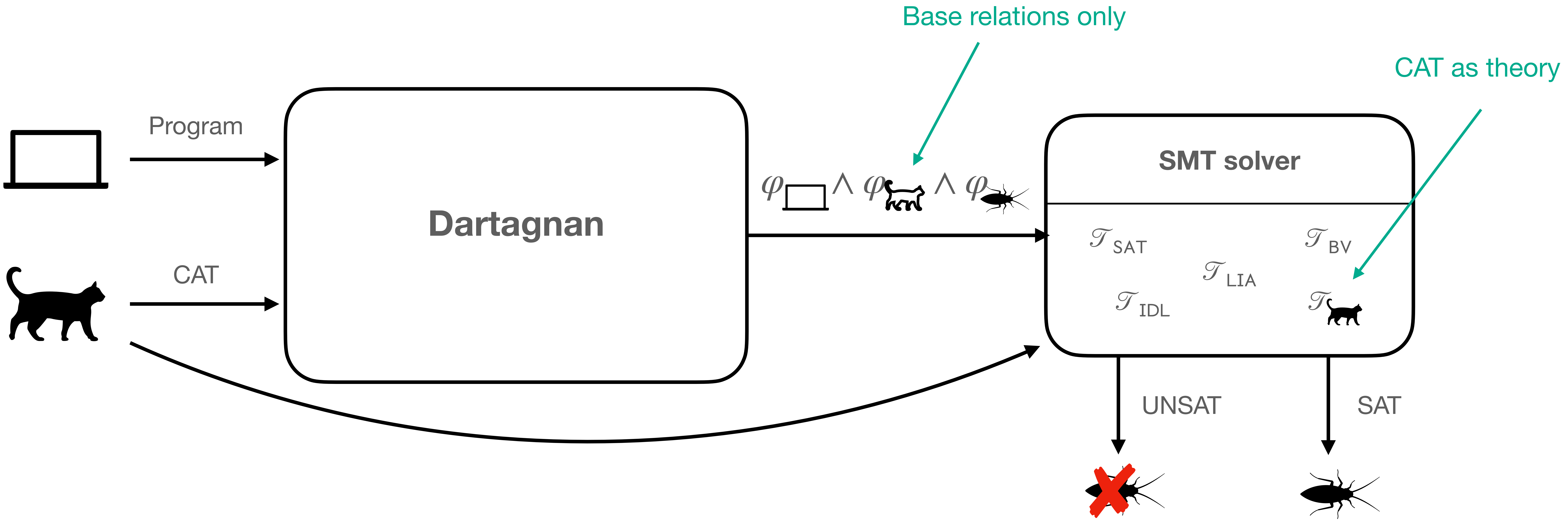
Dartagnan

Memory-model-parametric BMC



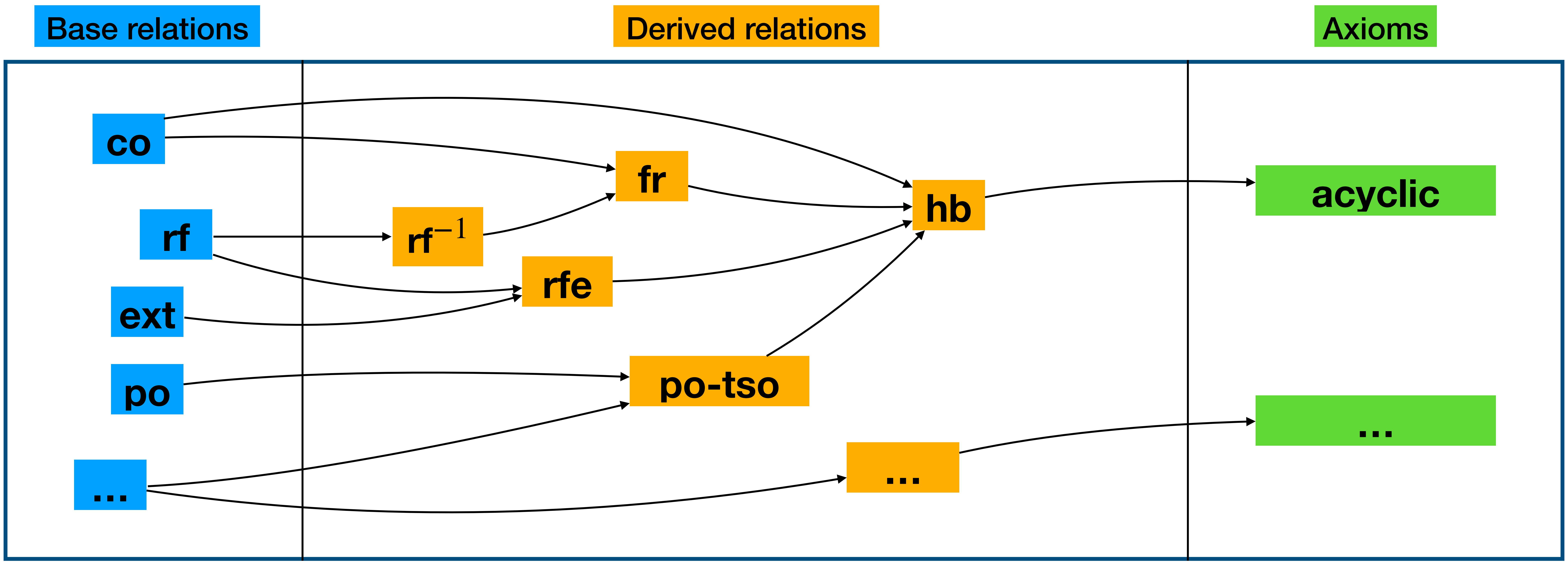
Dartagnan + CAAT

Memory-model-parametric BMC with CAAT

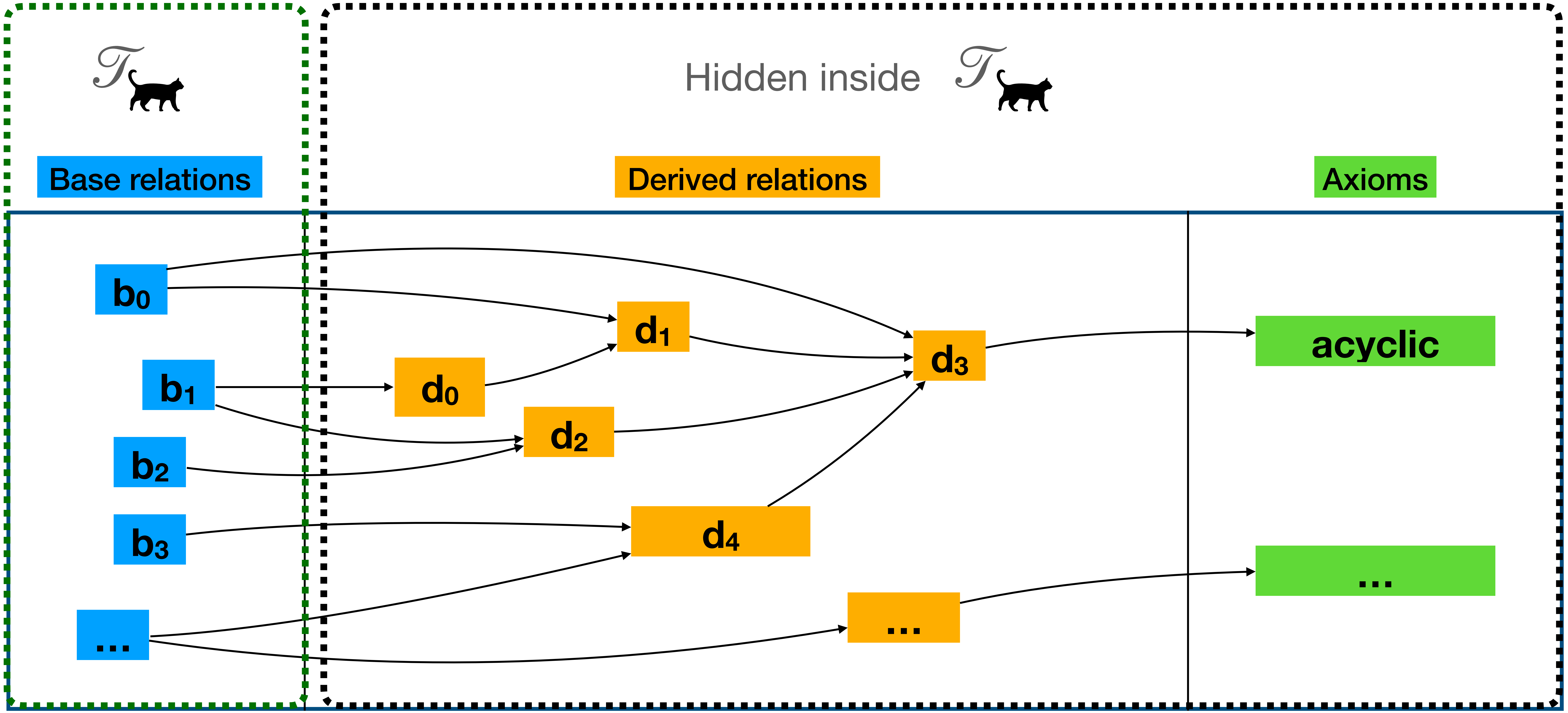


CAT as logical theory

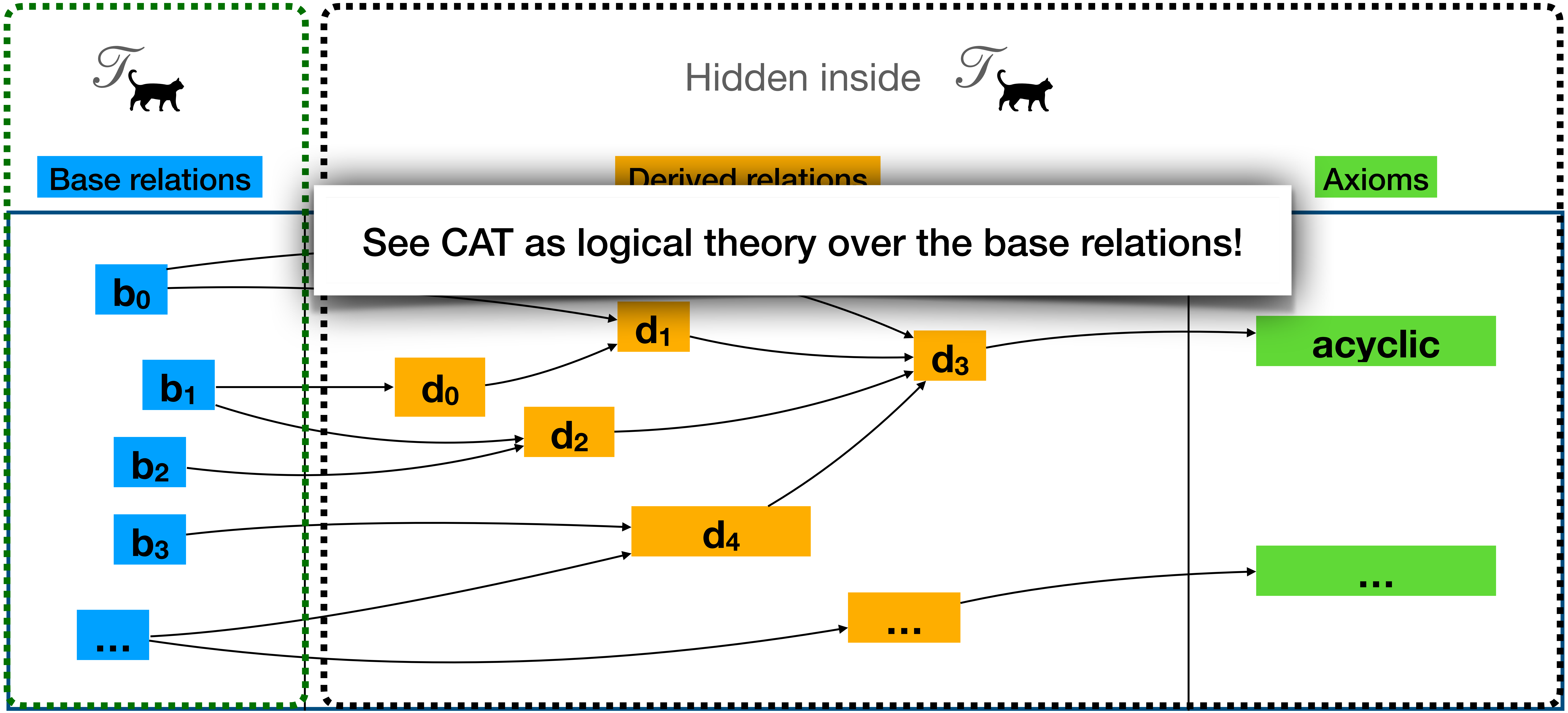
```
let fr = rf^-1;co
let po-tso = (po \ WxR) | mfence
let hb = po-tso | (rf & ext) | fr | co
acyclic hb
// more relations & axioms
```



CAAT: Consistency as a Theory



CAAT: Consistency as a Theory



See CAT as logical theory over the base relations!

How does it work?

Theory solving for SC

```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```

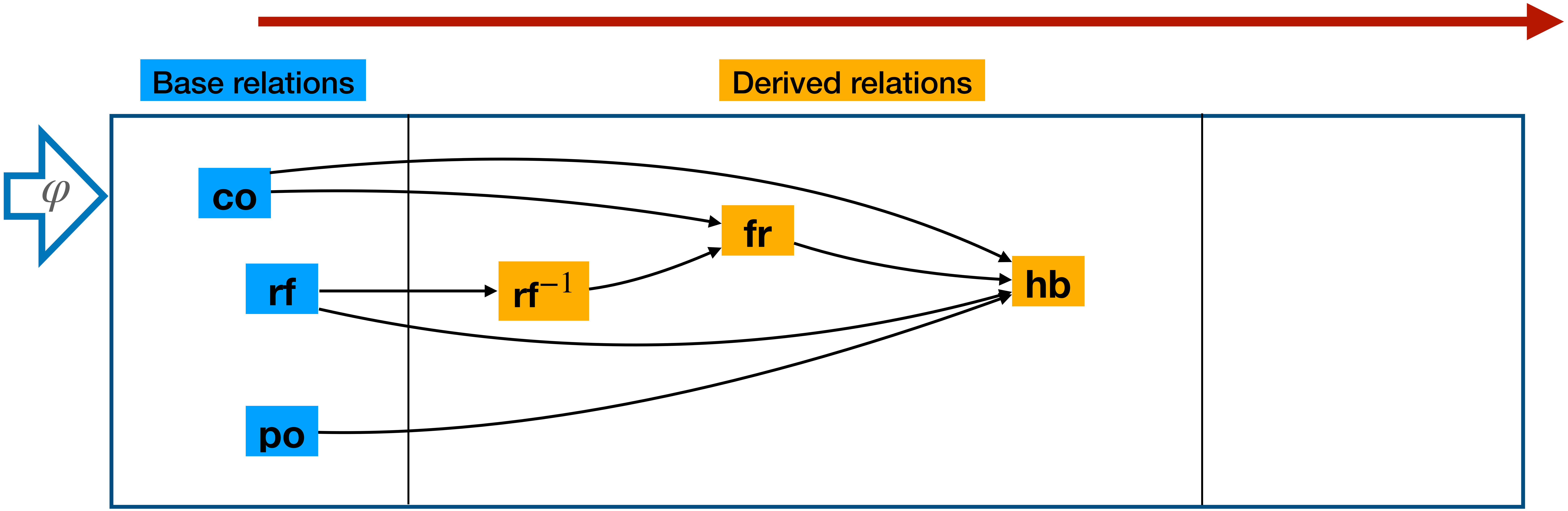
Theory solving for SC

```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```

Theory solving for SC

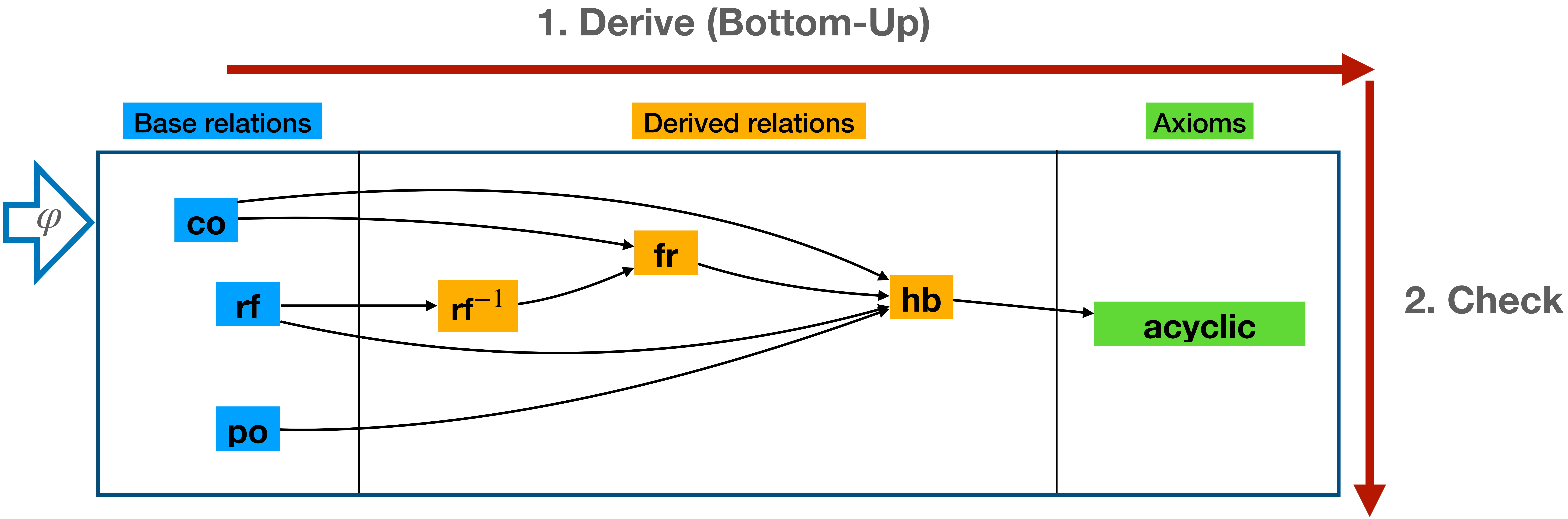
```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```

1. Derive (Bottom-Up)

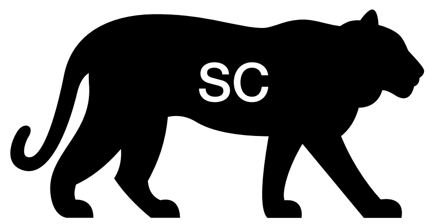


Theory solving for SC

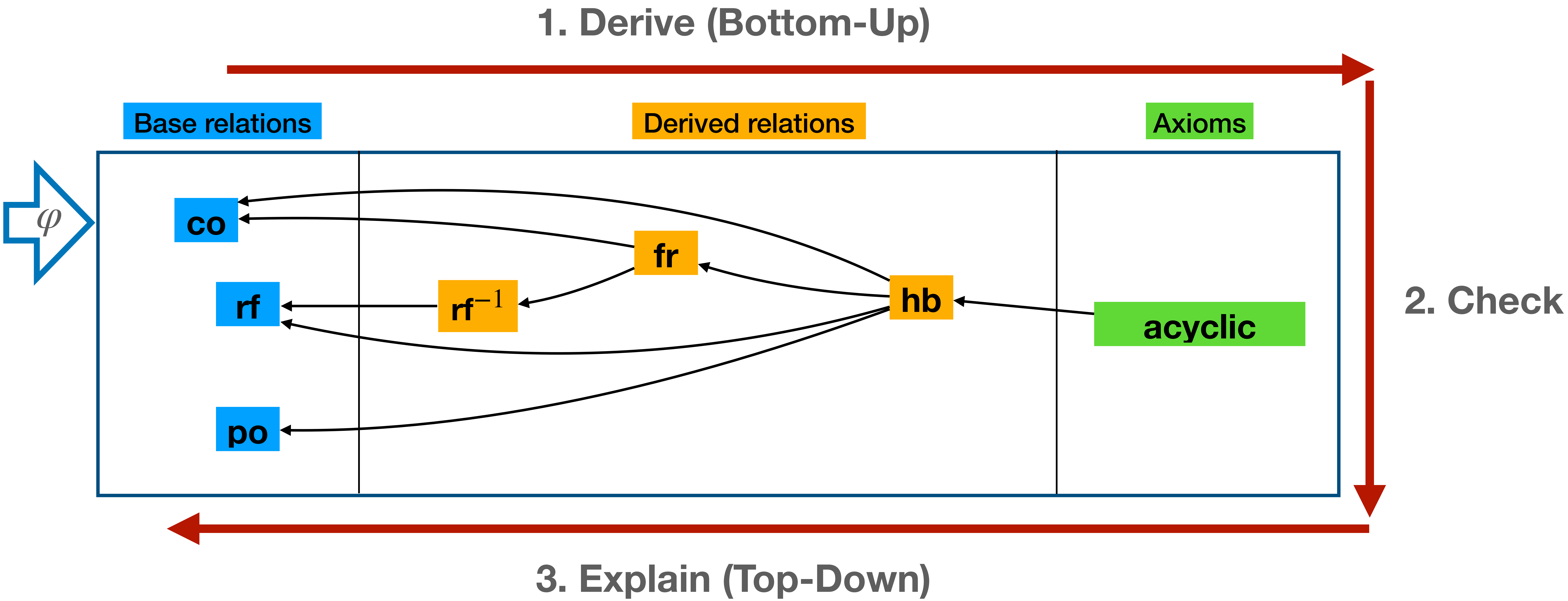
```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```



Theory solving for SC

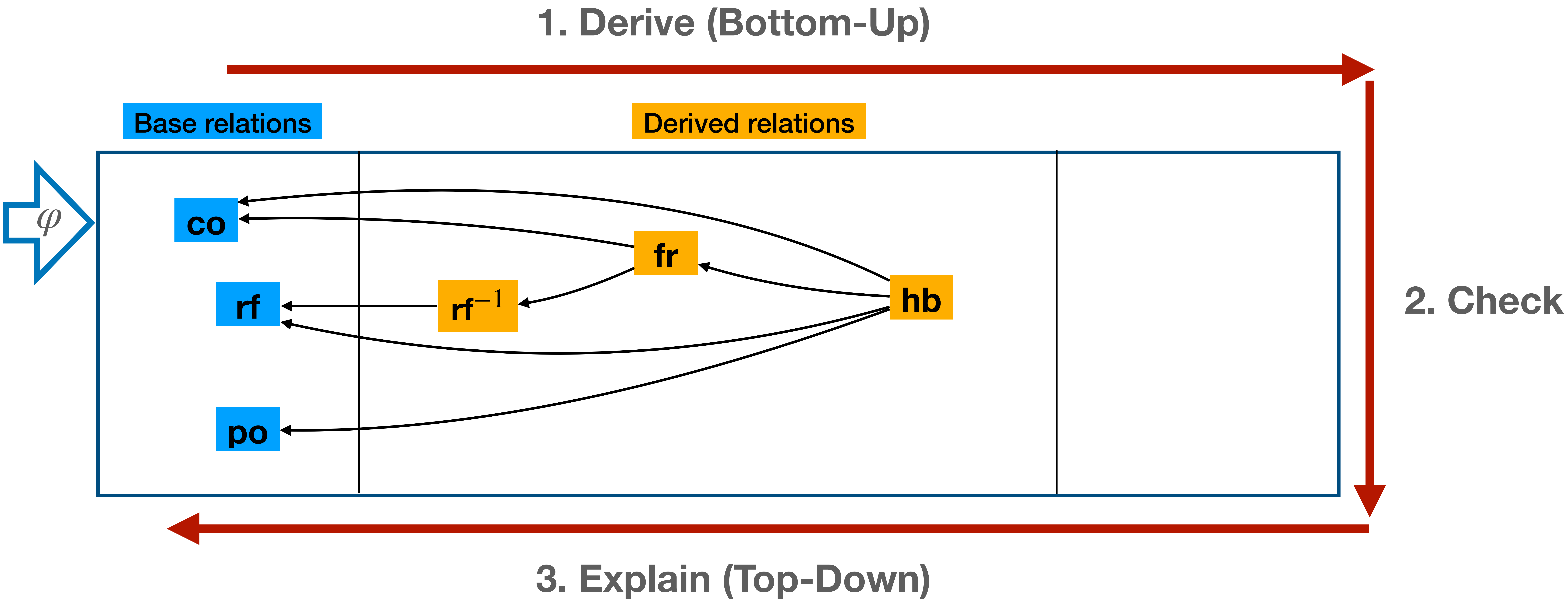



```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```



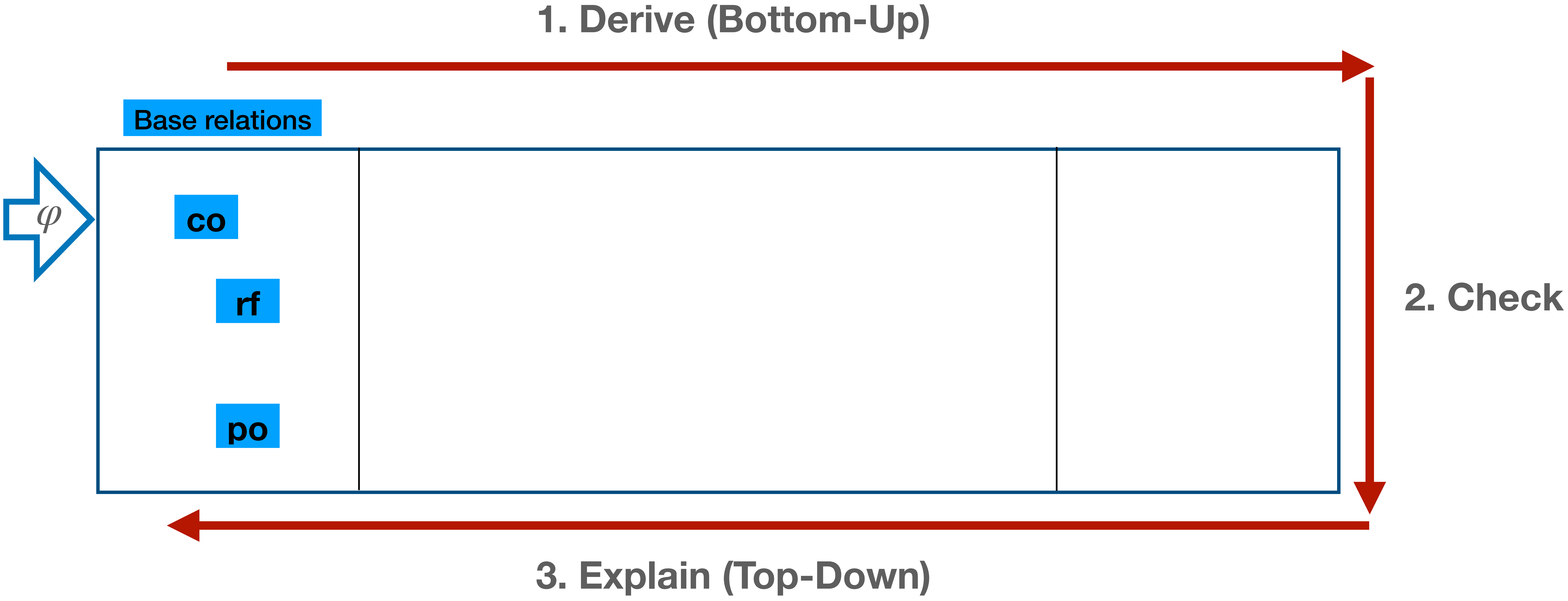
Theory solving for SC

```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```



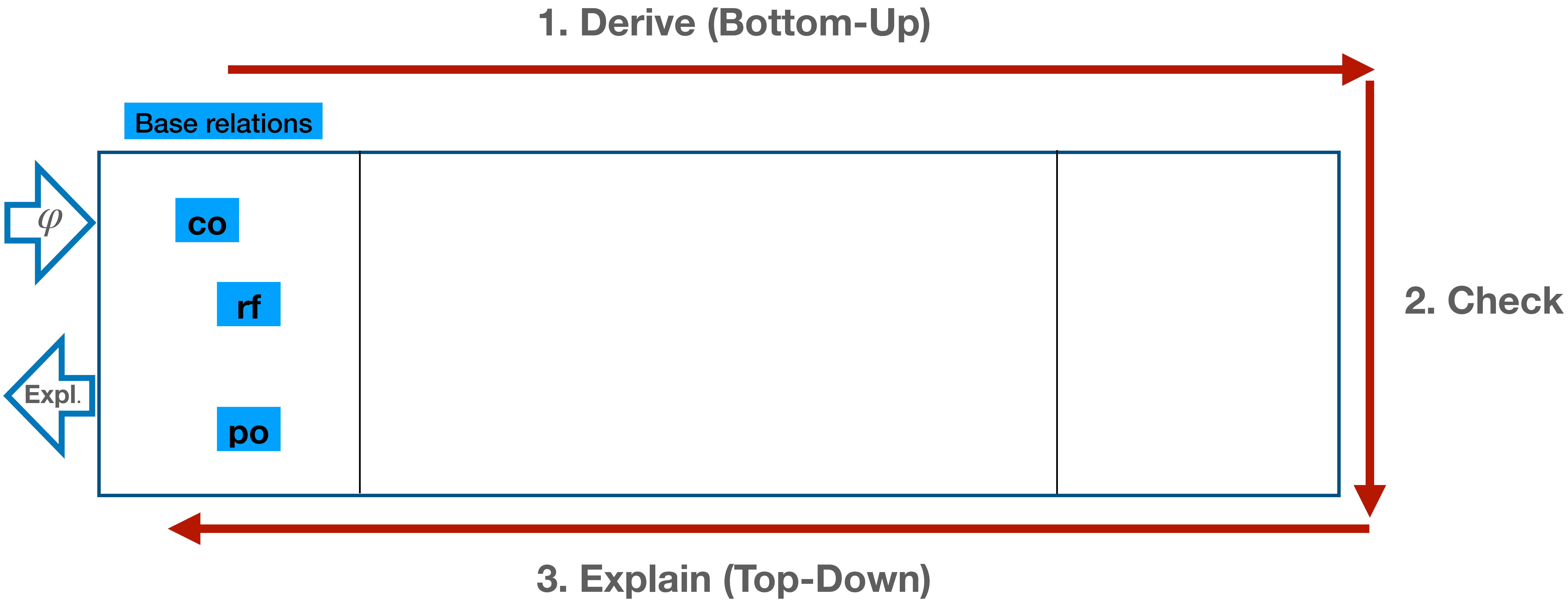
Theory solving for SC

```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```



Theory solving for SC

```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```



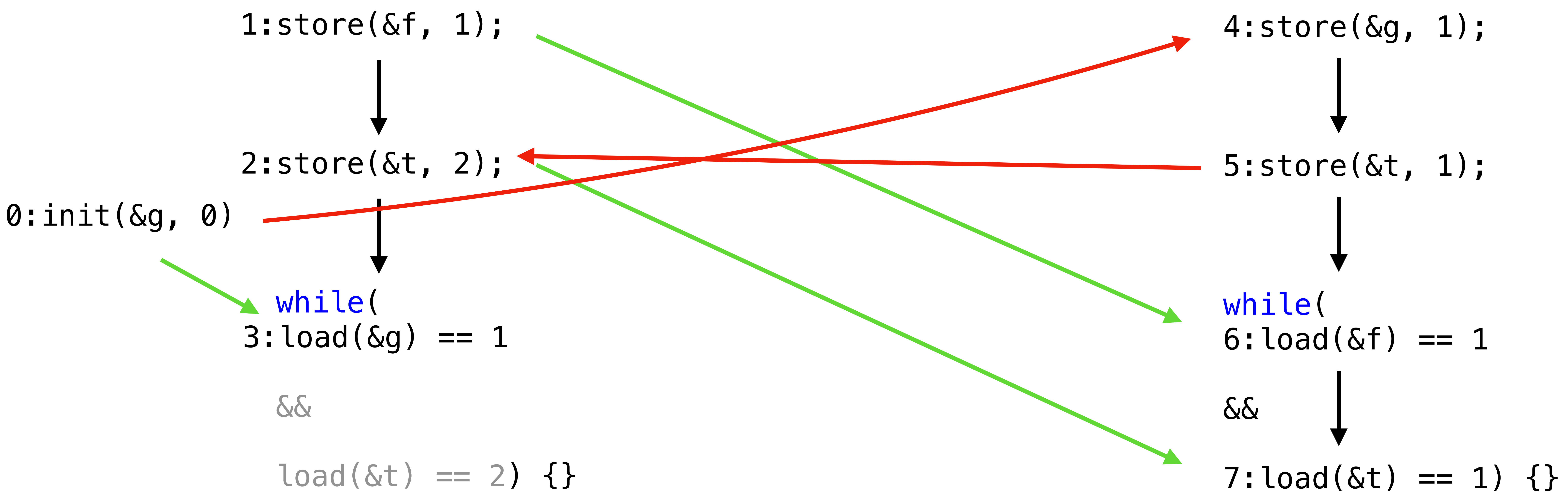
SC

```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```

Program order:

Read-from:

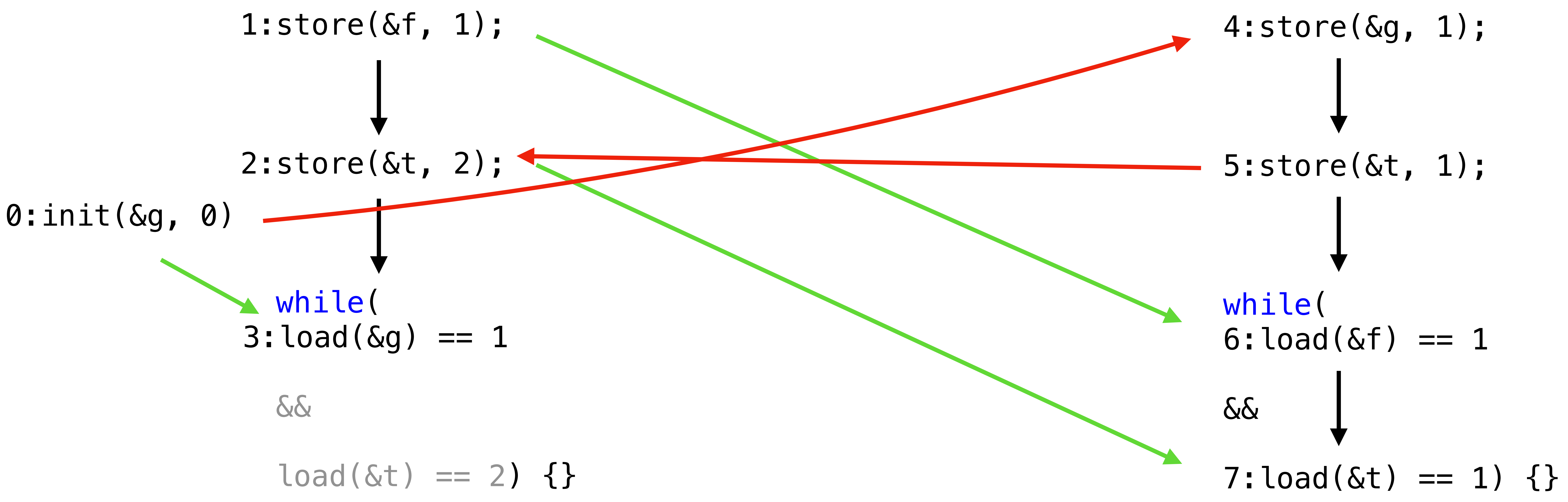
Coherence:



SC

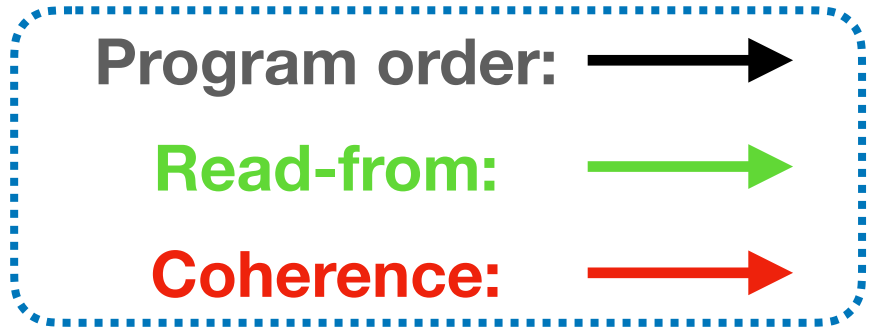
```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```

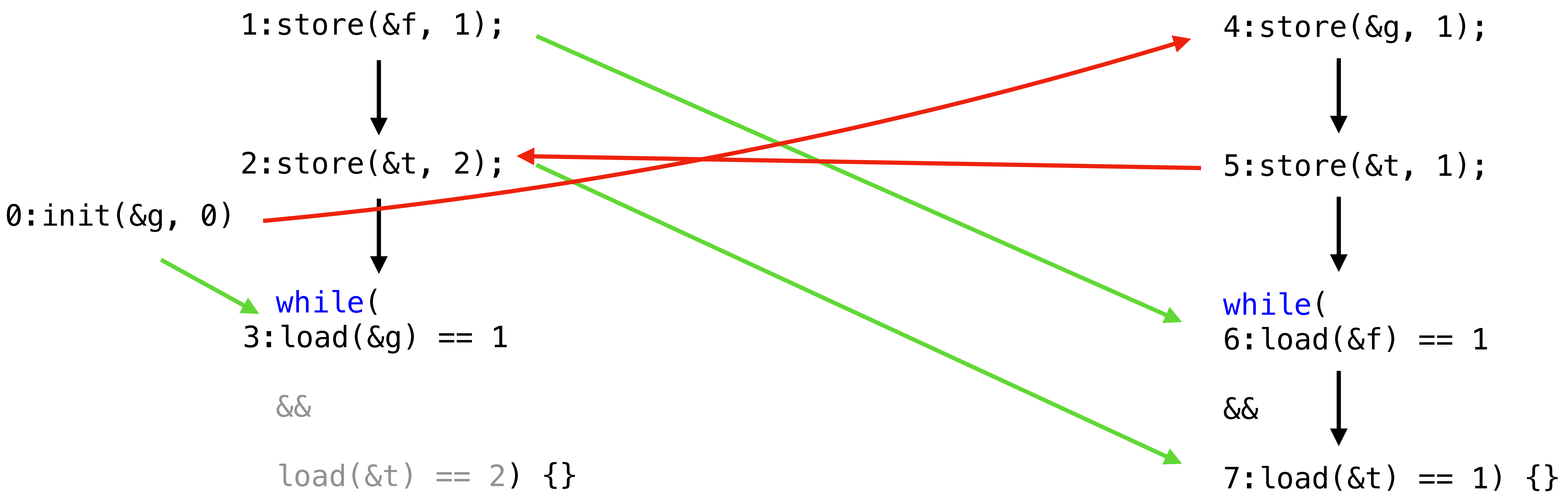
Program order:
Read-from:
Coherence:



$$\varphi = \begin{array}{c} po(1,2) \wedge po(2,3) \wedge po(4,5) \wedge po(5,6) \wedge po(6,7) \\ \wedge\, rf(0,3) \wedge rf(1,6) \wedge rf(2,7) \\ \wedge\, co(0,4) \wedge co(5,2) \end{array}$$

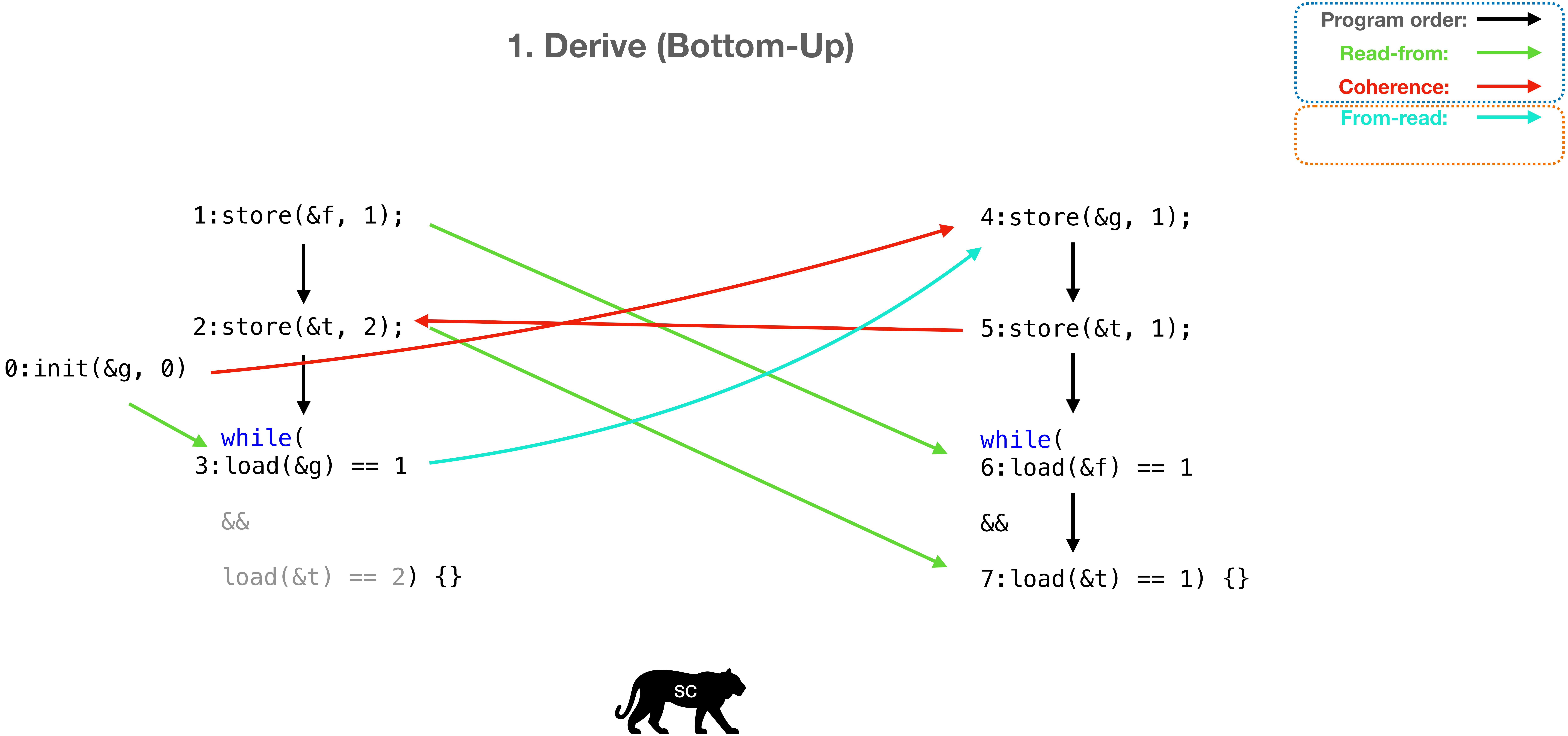
1. Derive (Bottom-Up)






```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```

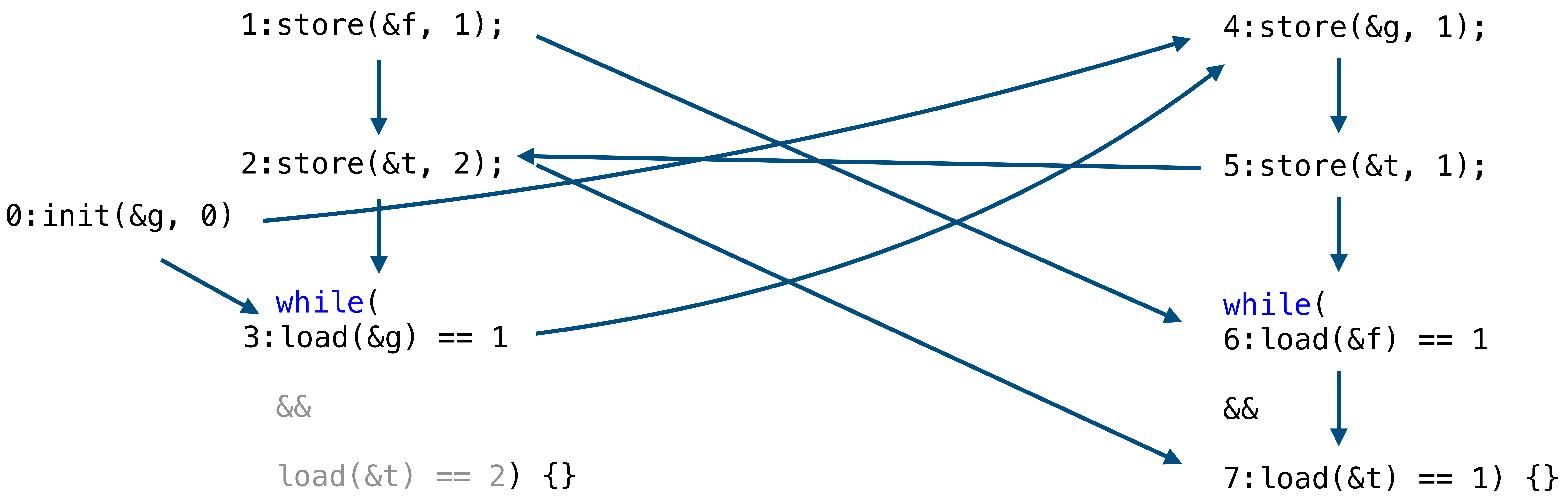
1. Derive (Bottom-Up)

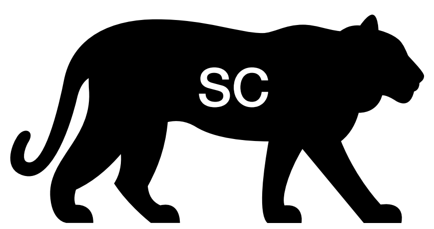



```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```

1. Derive (Bottom-Up)

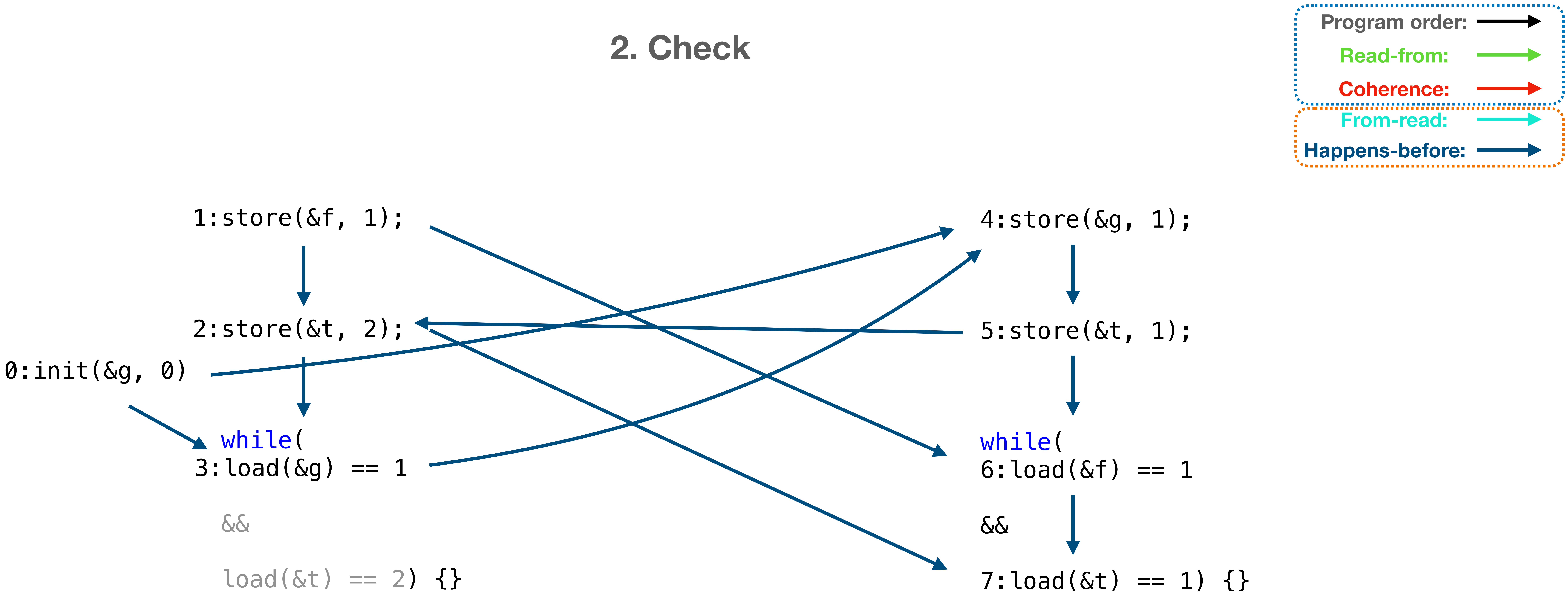
Program order:
Read-from:
Coherence:
From-read:
Happens-before:

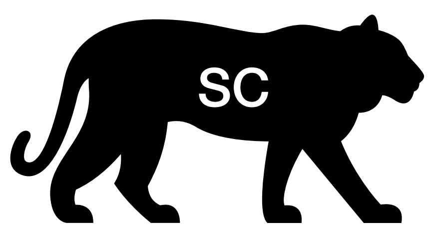





```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```

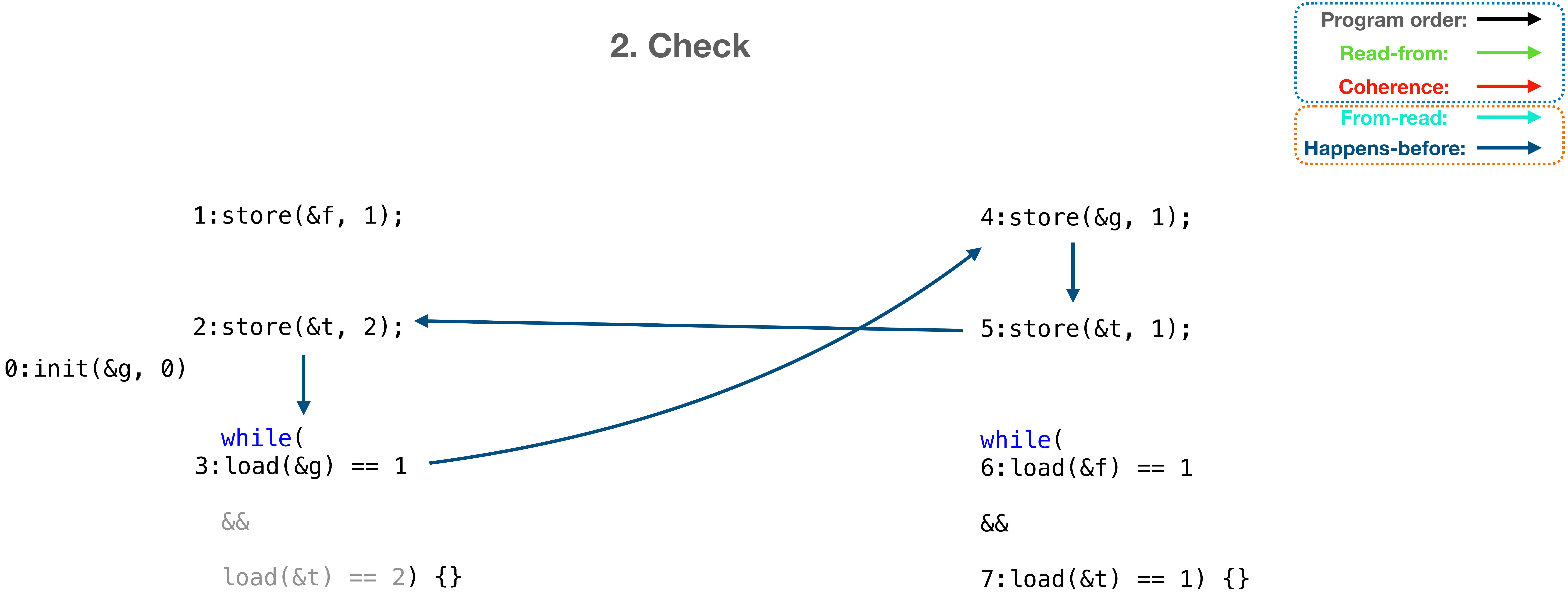
2. Check

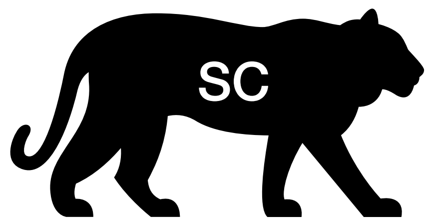





```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```

2. Check





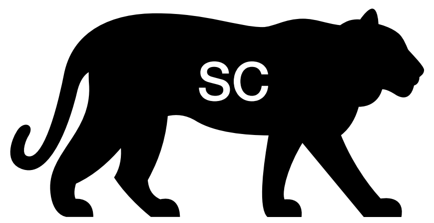

```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```

3. Explain (Top-Down)

Program order:
Read-from:
Coherence:
From-read:
Happens-before:

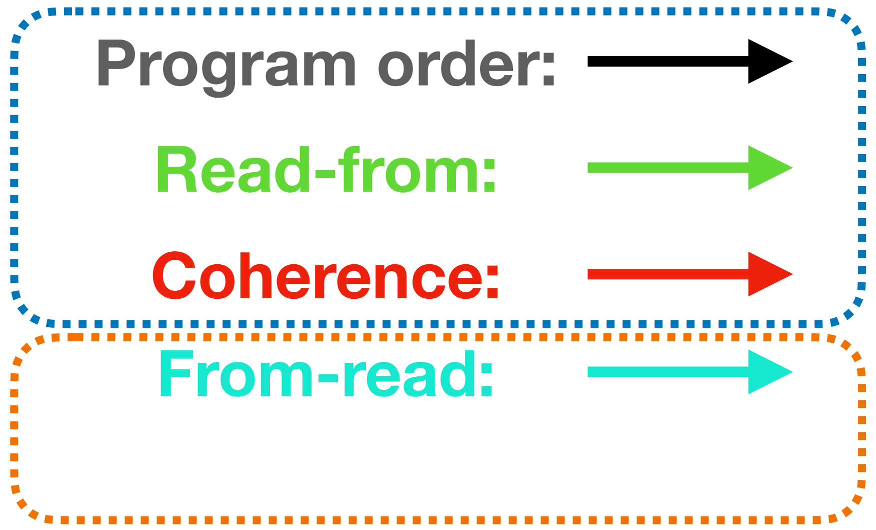
```
1:store(&f, 1);

2:store(&t, 2);

0:init(&g, 0)

  while(
3:load(&g) == 1

  &&

  load(&t) == 2) {}
```

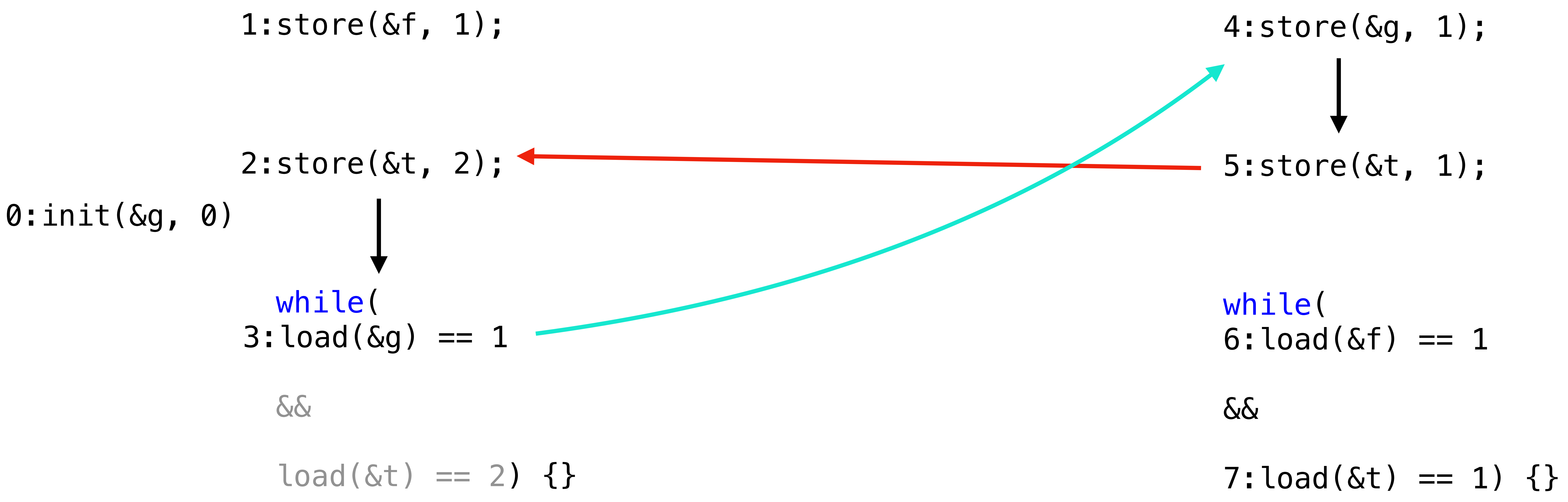
```
4:store(&g, 1);

5:store(&t, 1);

while(
6:load(&f) == 1

&&

7:load(&t) == 1) {}
```

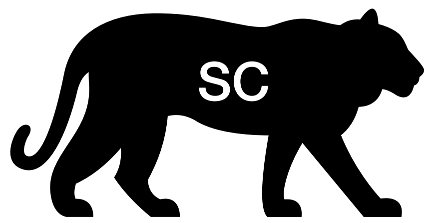



```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```

3. Explain (Top-Down)



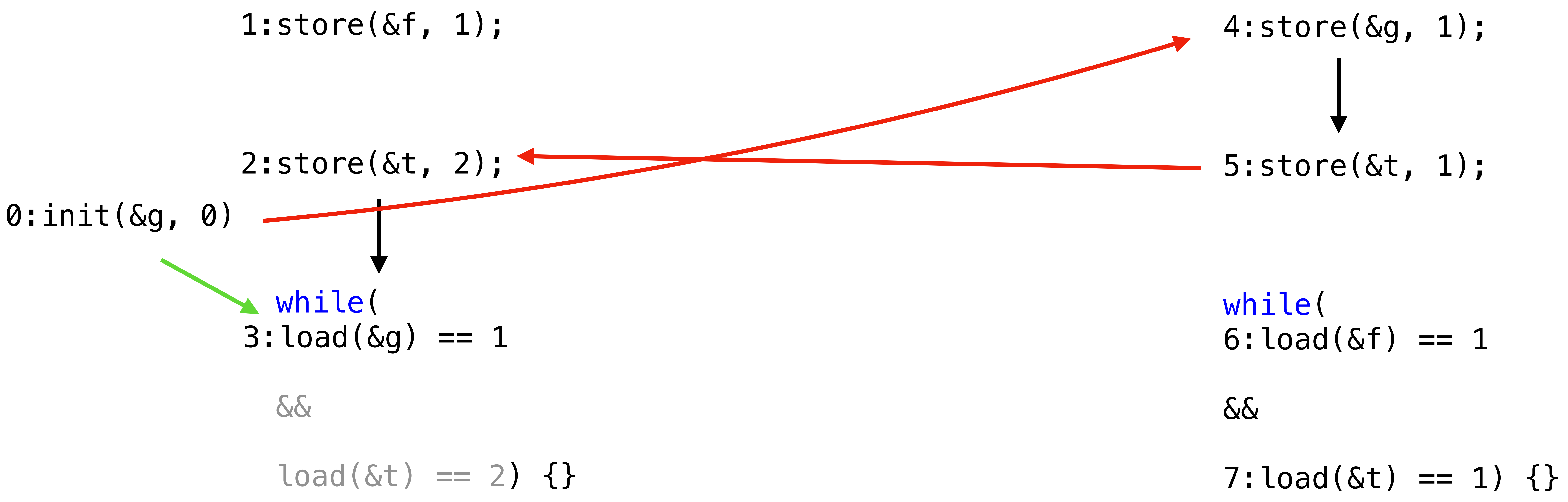


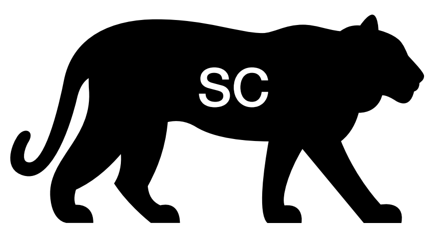



```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```

3. Explain (Top-Down)

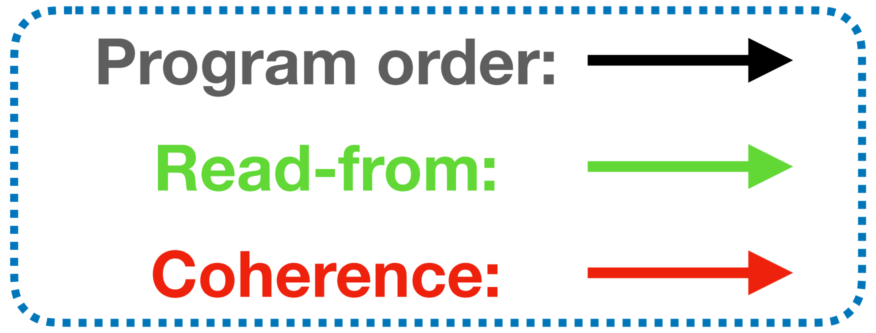
Program order: →
Read-from: →
Coherence: →

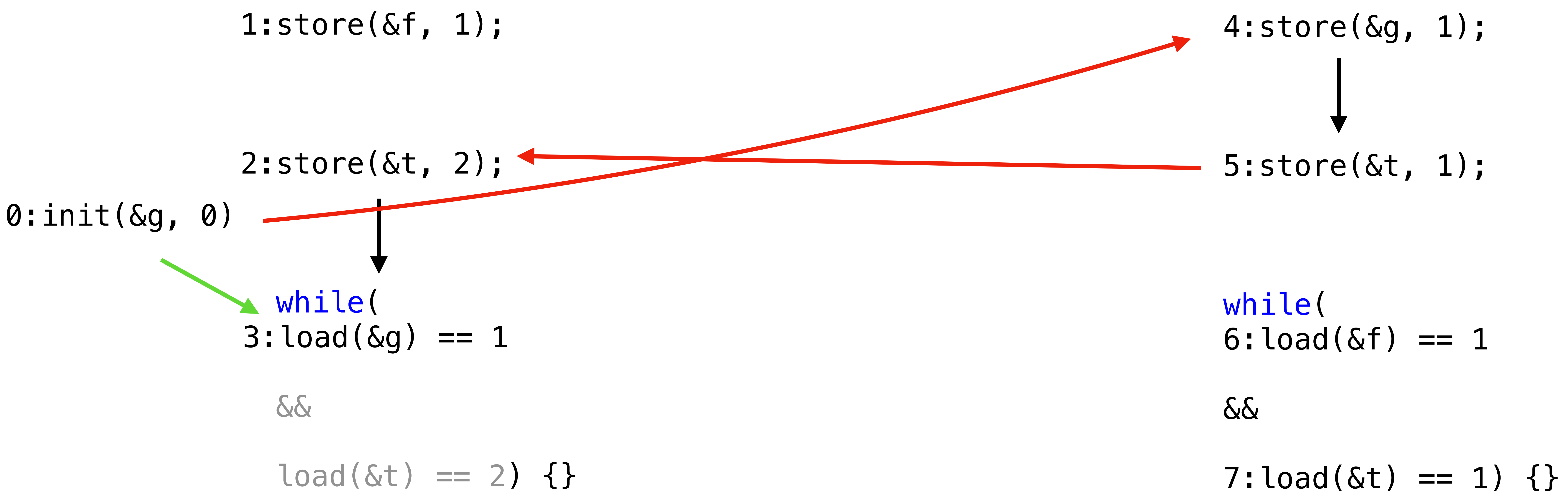





```
let fr = rf^-1;co
let hb = po | rf | fr | co
acyclic hb
```

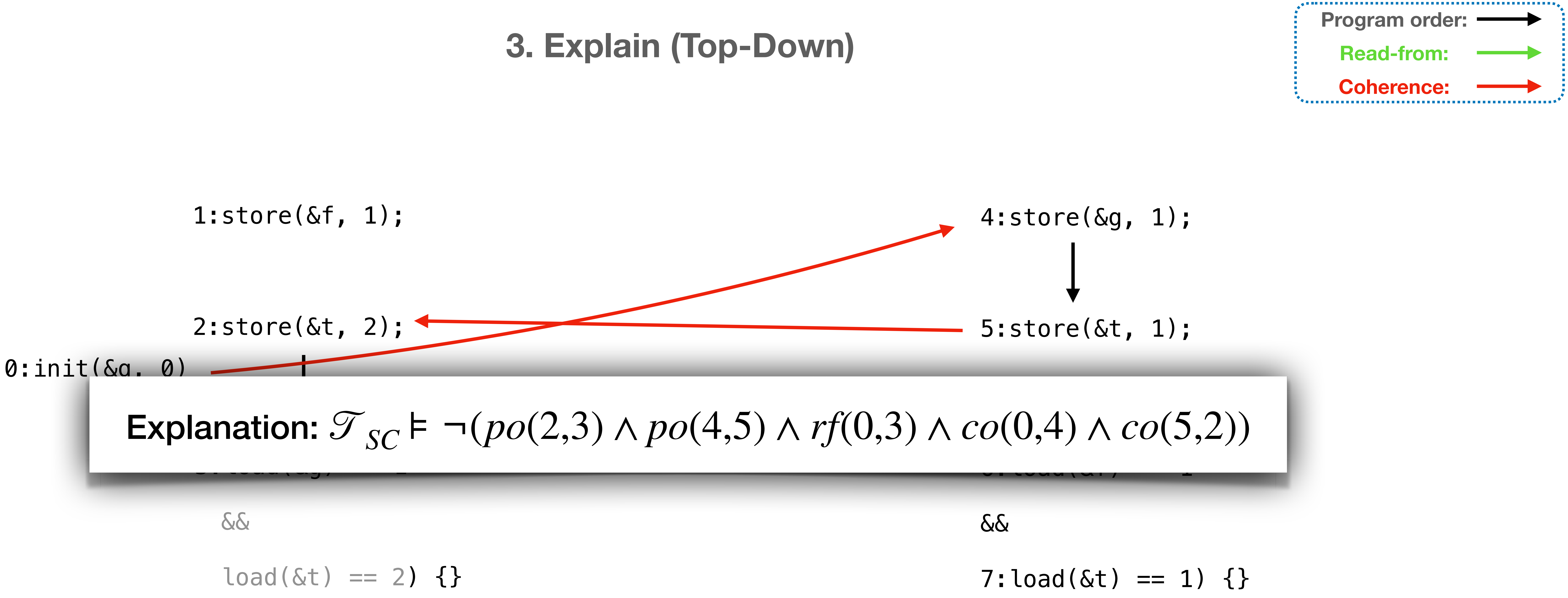
3. Explain (Top-Down)





$$\left[\begin{array}{c} po(2,3) \wedge po(4,5) \\ \wedge\, rf(0,3) \\ \wedge\, co(0,4) \wedge co(5,2) \end{array}\right] \sqsubseteq \left[\begin{array}{c} po(1,2) \wedge po(2,3) \wedge po(4,5) \wedge po(5,6) \wedge po(6,7) \\ \wedge\, rf(0,3) \wedge rf(1,6) \wedge rf(2,7) \\ \wedge\, co(0,4) \wedge co(5,2) \end{array}\right] = \varphi$$

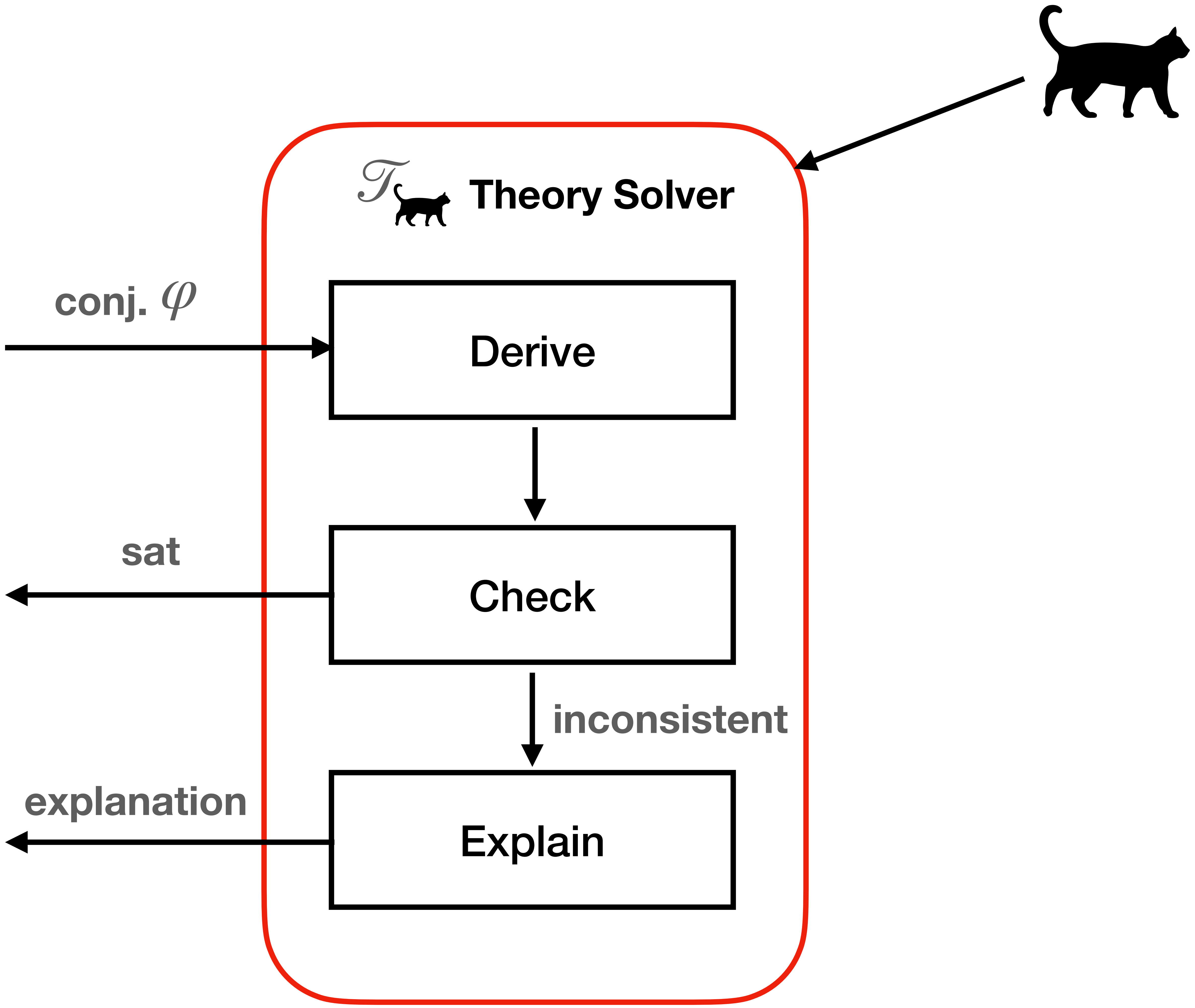
3. Explain (Top-Down)



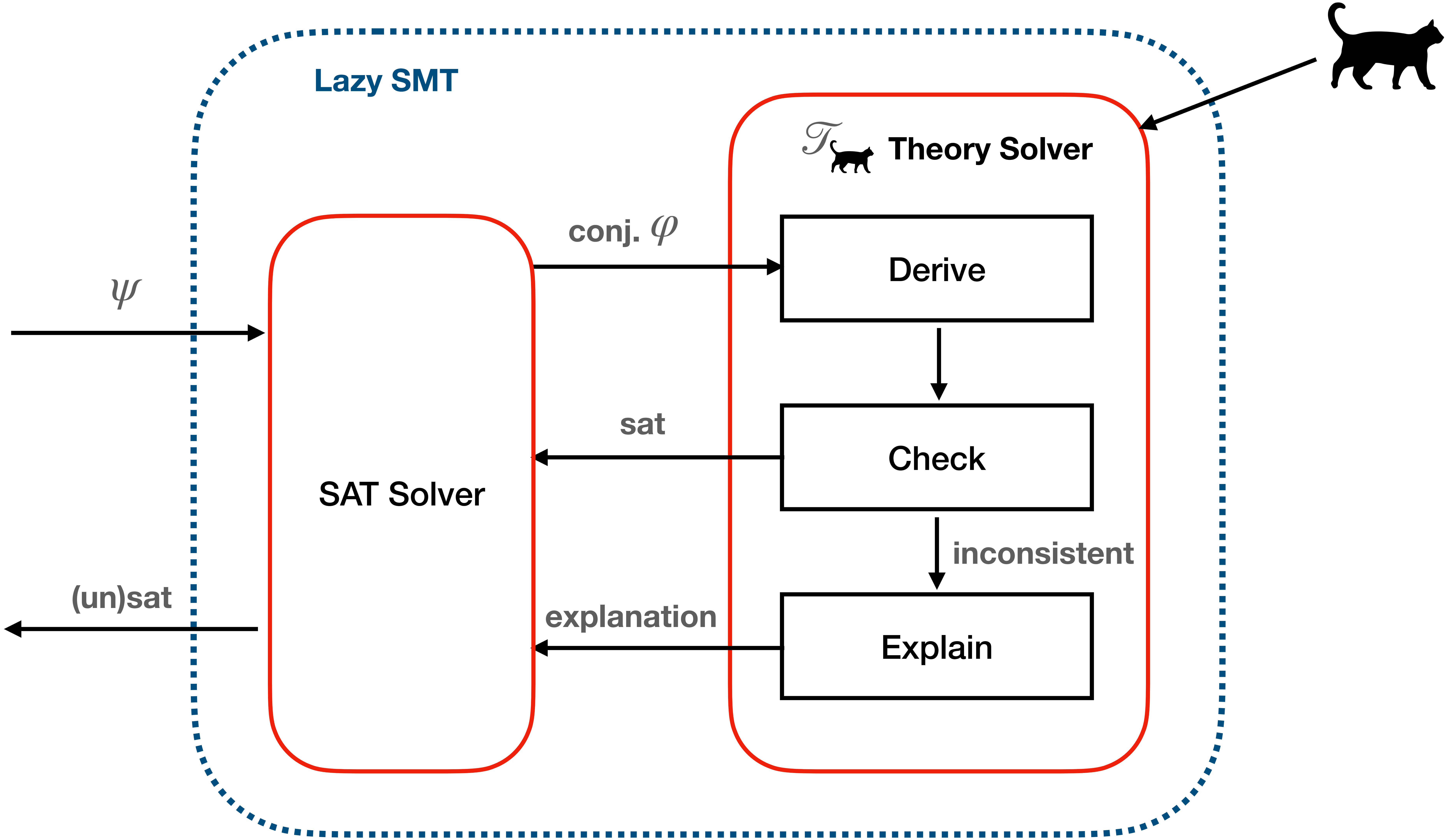
Explanation: $\mathcal{T}_{SC} \vDash \neg(po(2,3) \wedge po(4,5) \wedge rf(0,3) \wedge co(0,4) \wedge co(5,2))$

$$\begin{array}{c} po(2,3) \wedge po(4,5) \\ \wedge\, rf(0,3) \\ \wedge\, co(0,4) \wedge co(5,2) \end{array} \sqsubseteq \begin{array}{c} po(1,2) \wedge po(2,3) \wedge po(4,5) \wedge po(5,6) \wedge po(6,7) \\ \wedge\, rf(0,3) \wedge rf(1,6) \wedge rf(2,7) \\ \wedge\, co(0,4) \wedge co(5,2) \end{array} = \varphi$$

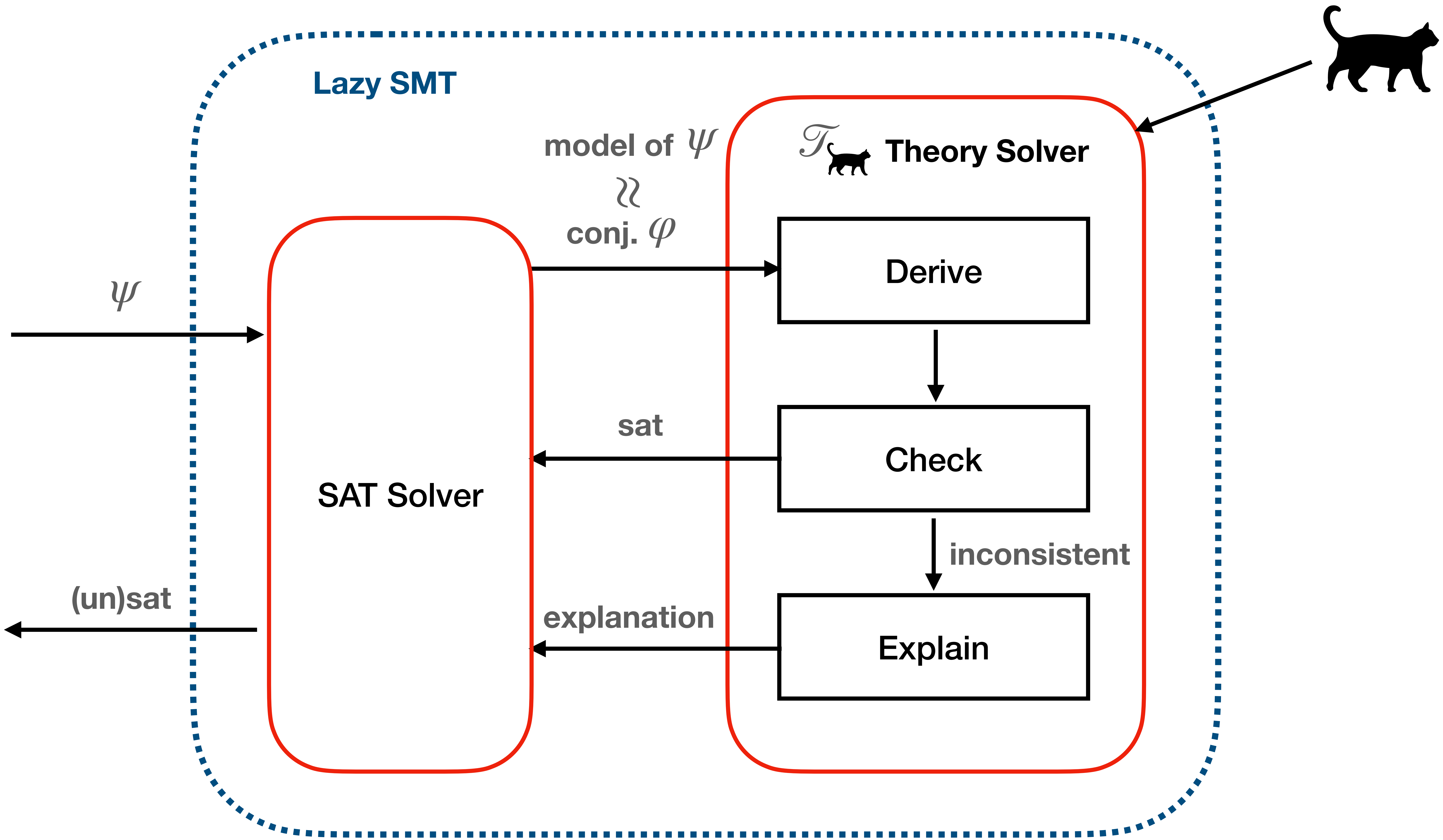
A Theory Solver for Consistency



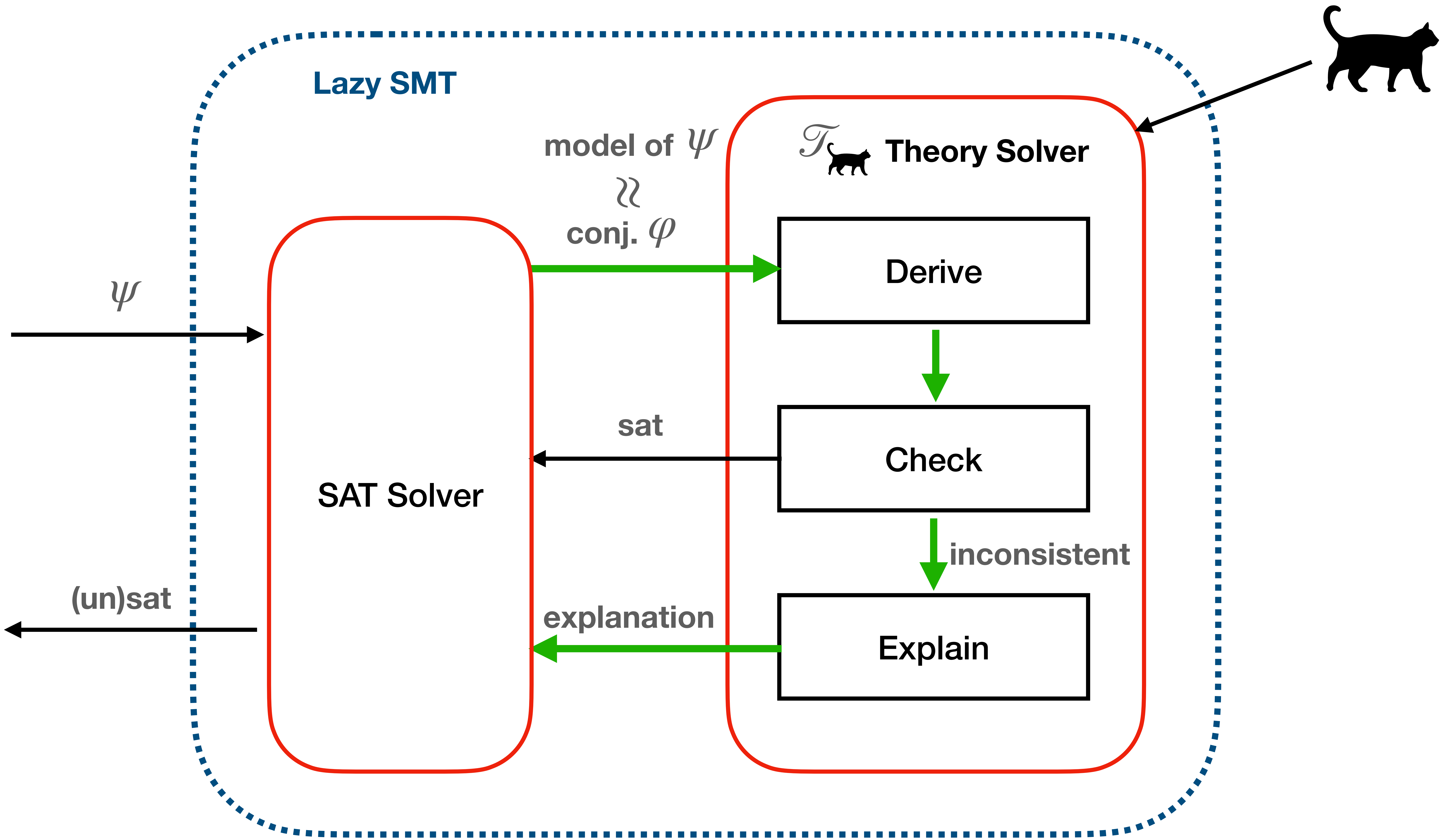
A Decision Procedure for Consistency



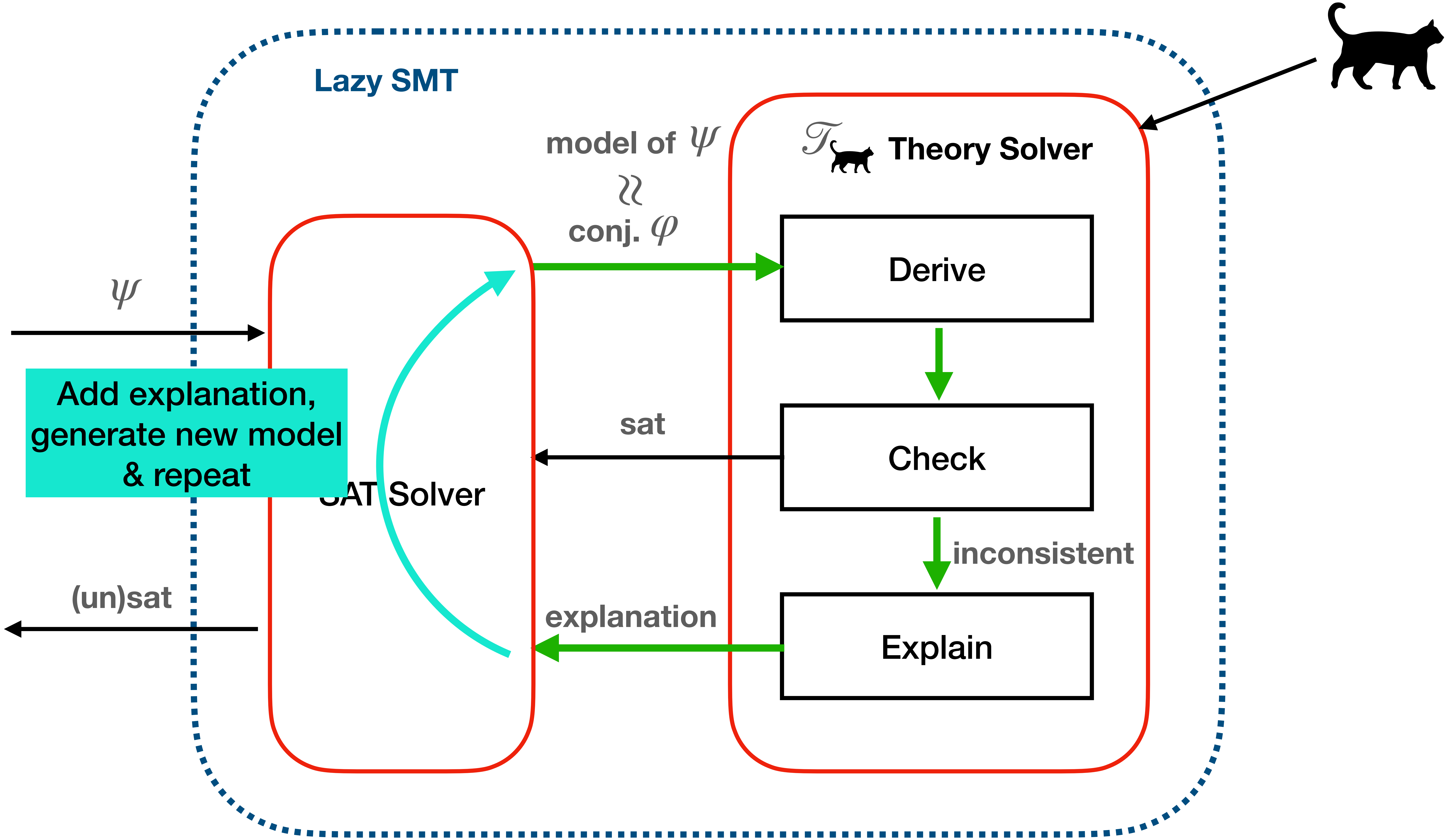
A Decision Procedure for Consistency



A Decision Procedure for Consistency



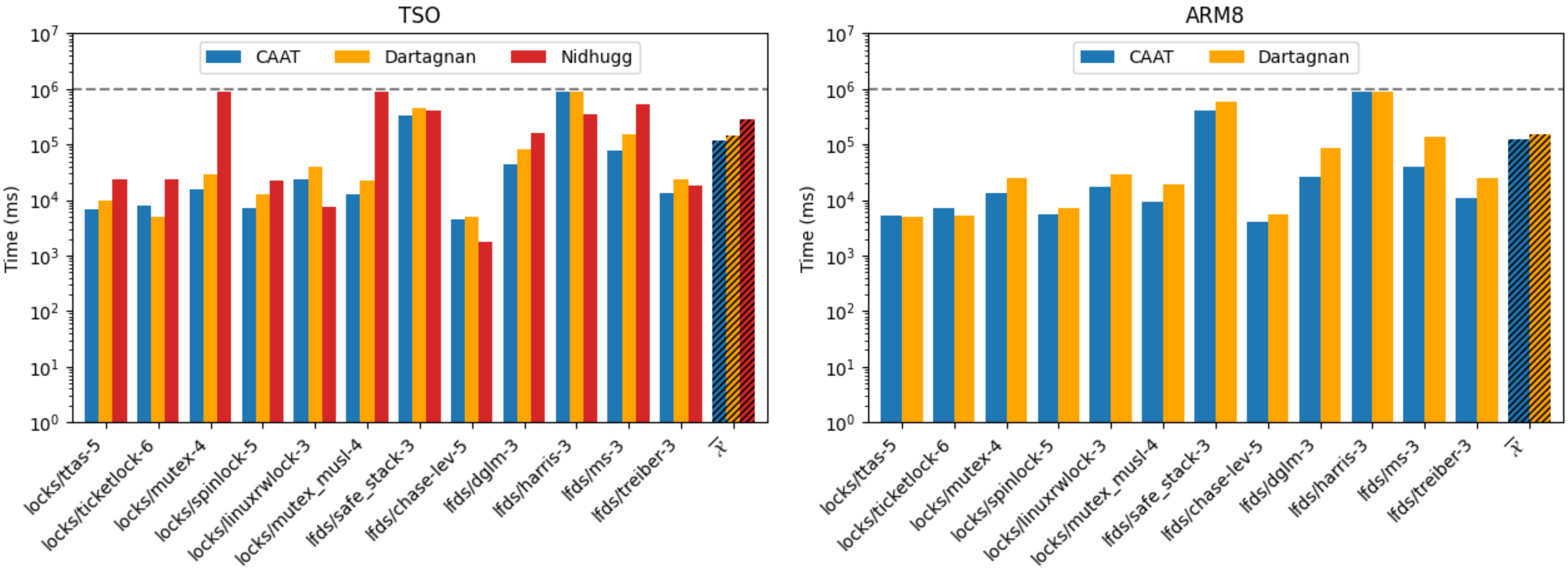
A Decision Procedure for Consistency



Evaluation

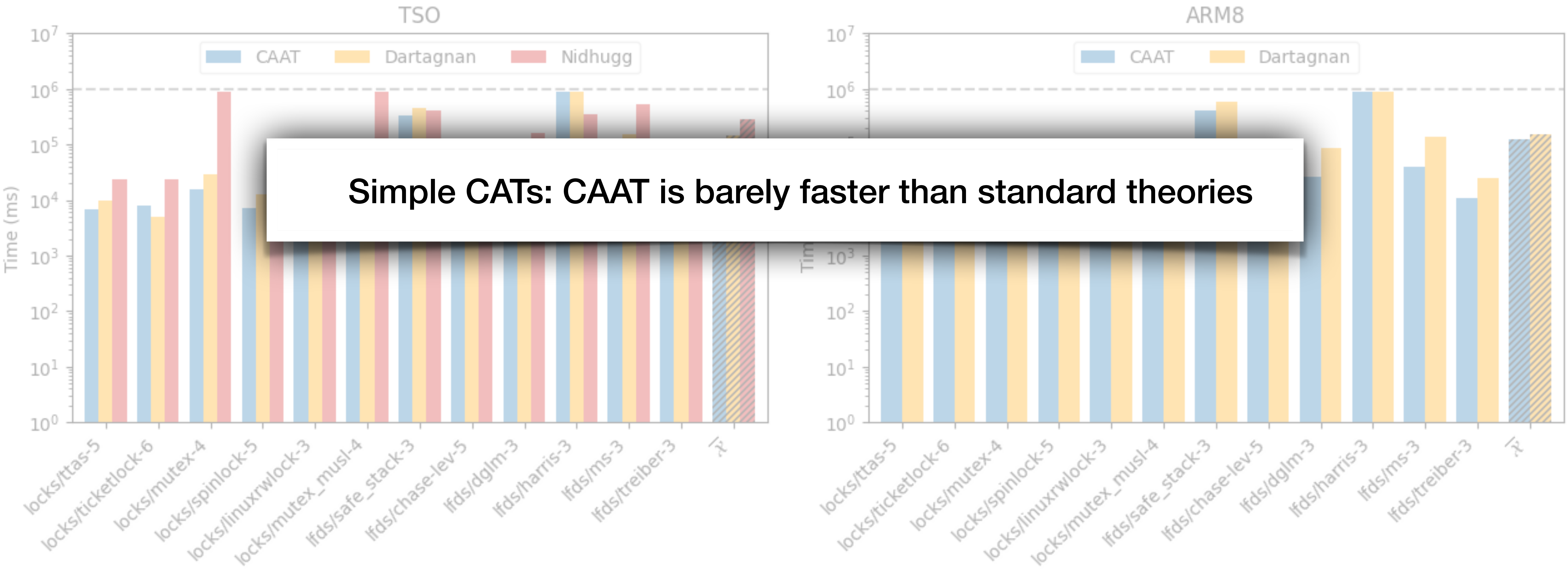
Evaluation: simple CATs 🐈‍⬛

No recursion

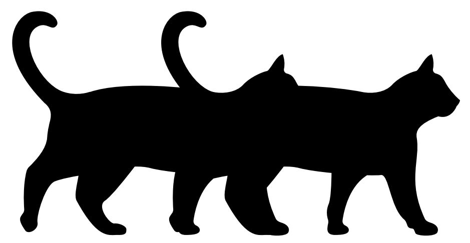


Evaluation: simple CATs

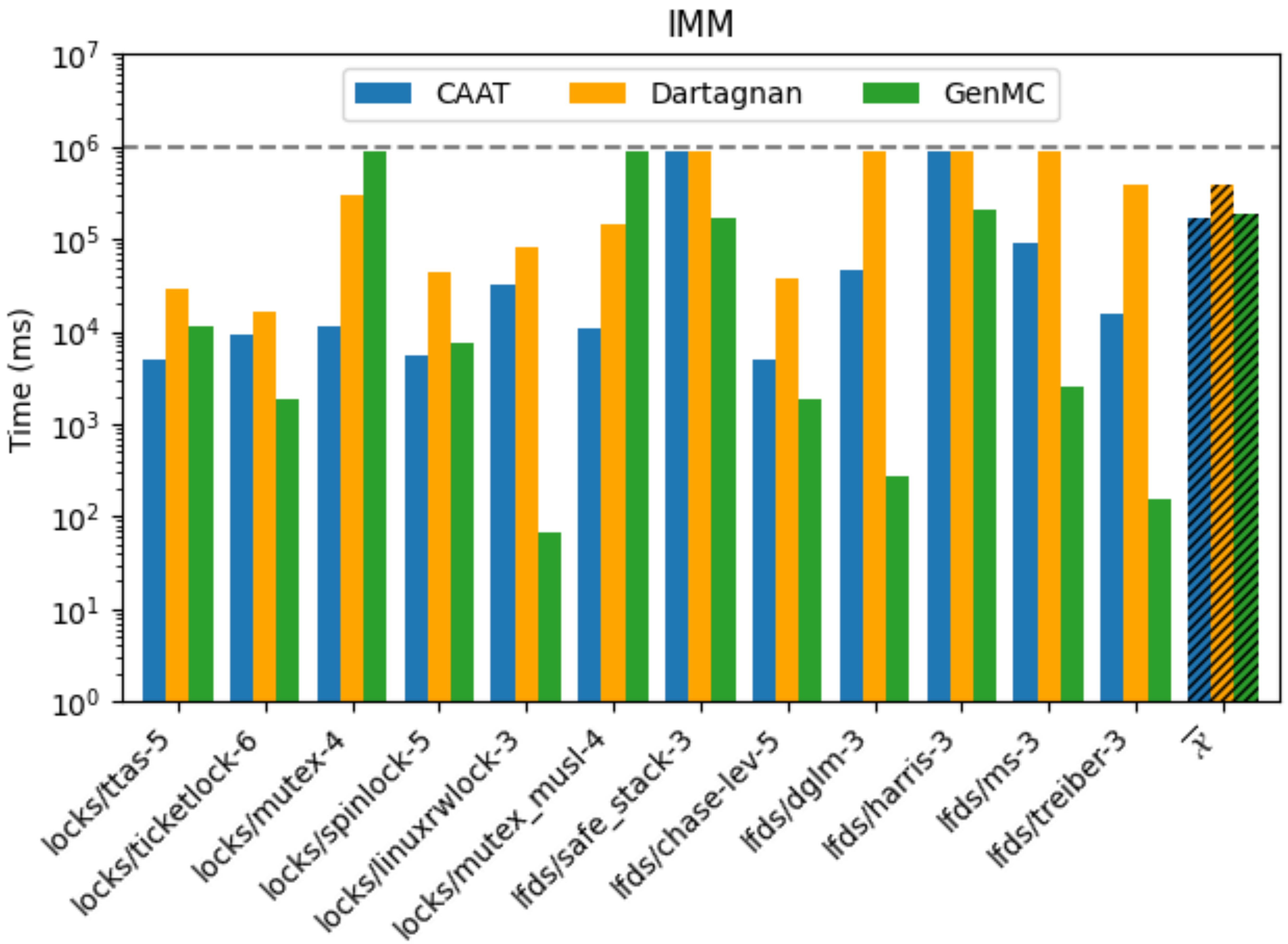
No recursion

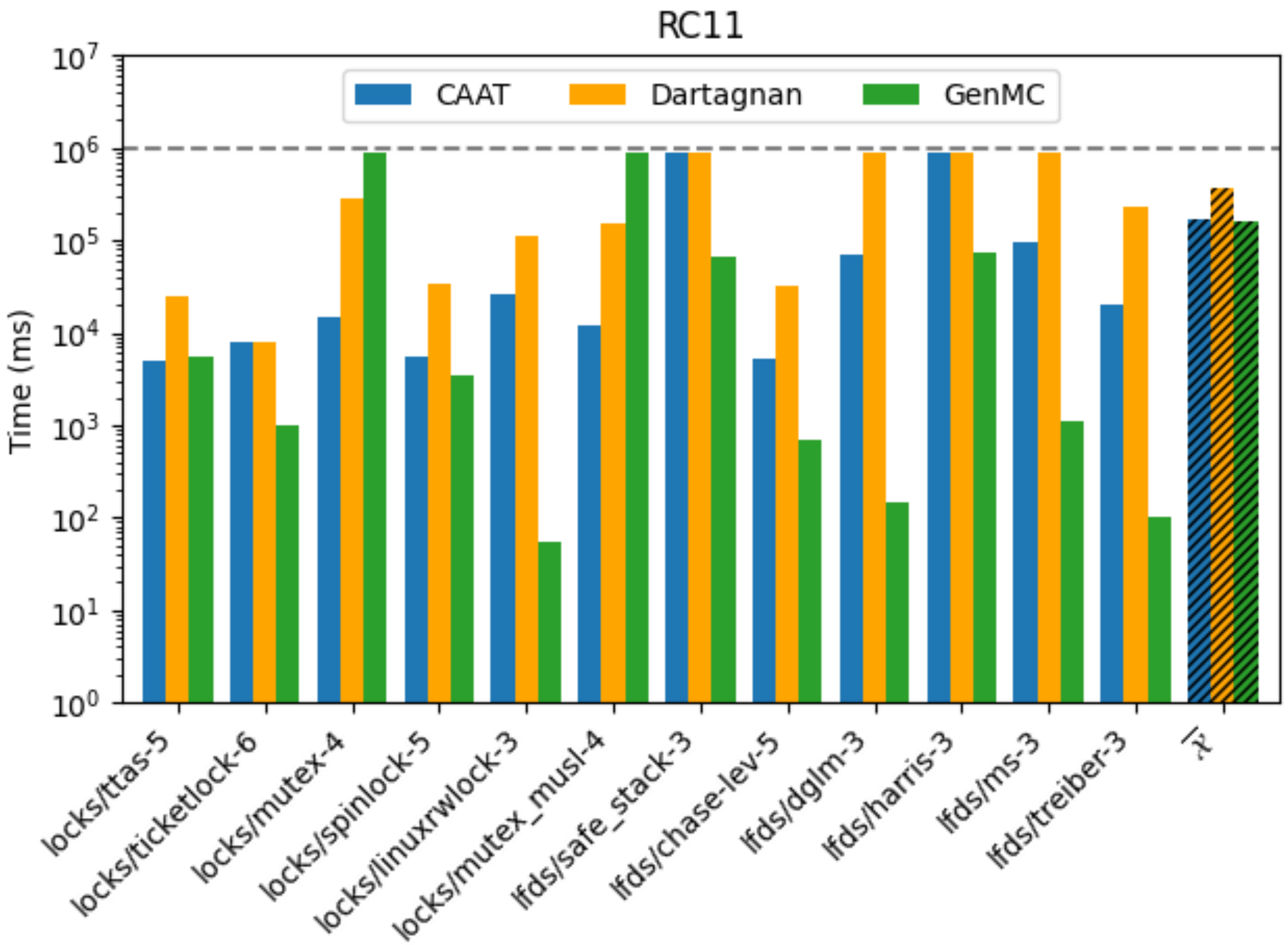


Simple CATs: CAAT is barely faster than standard theories

Evaluation: complex CATs 

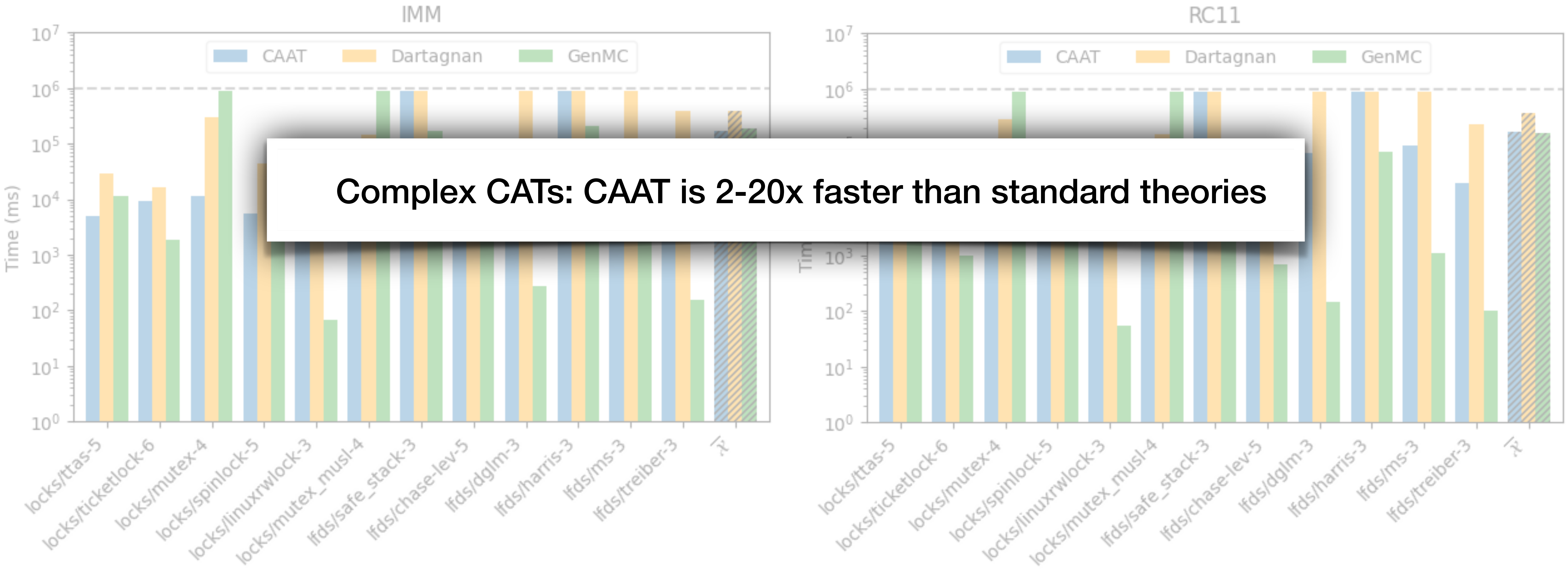
Linear recursion





Evaluation: complex CATs

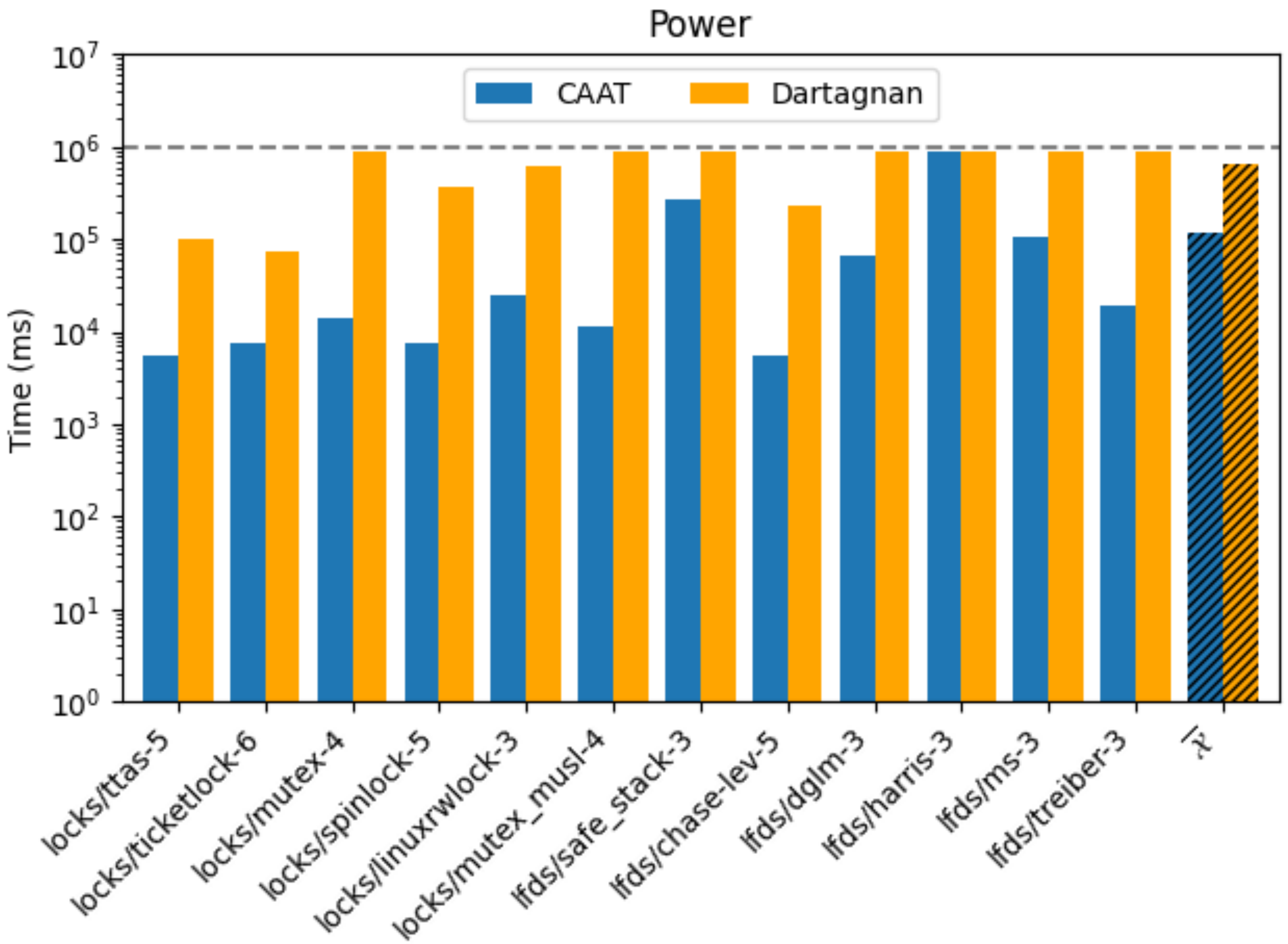
Linear recursion



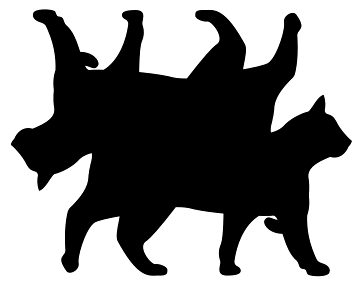
Complex CATs: CAAT is 2-20x faster than standard theories

Evaluation: very complex CATs

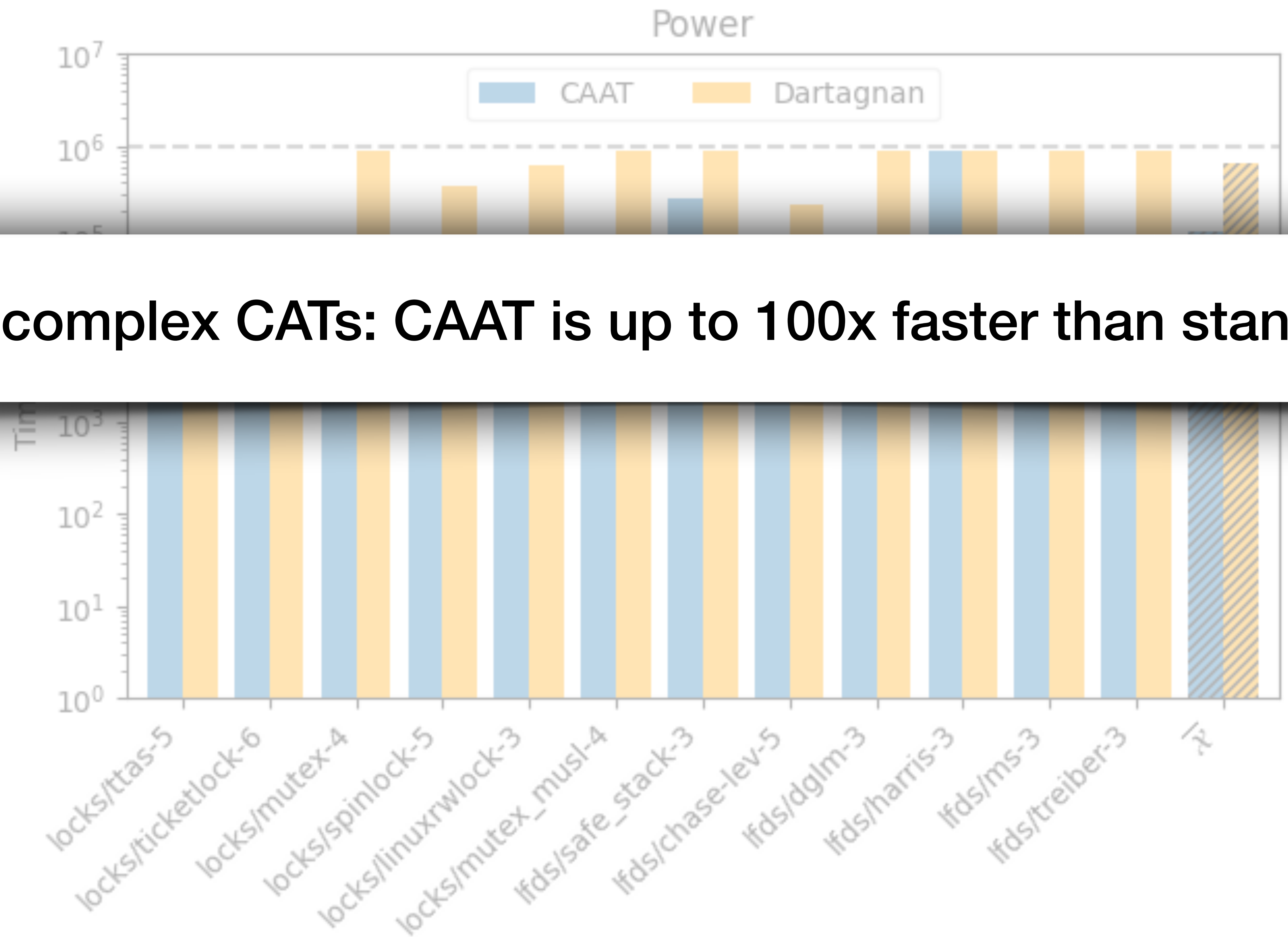
Non-linear recursion



Evaluation: very complex CATs

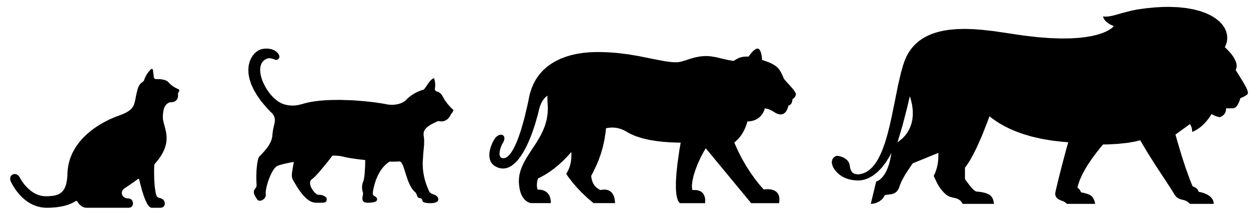


Non-linear recursion



Very complex CATs: CAAT is up to 100x faster than standard theories

Conclusion

- We see consistency models as a family of theories 
- Consistency theories handle least fixed points, unlike existing theories
- We give a general theory solver for consistency theories
- Using CAAT in BMC gives substantial performance improvement

Ongoing Work

- Online integration with the SMT solver
- Incrementality is a problem — the partial models are often largely different, because other theories make the solver backtrack!
- Use matching instead!

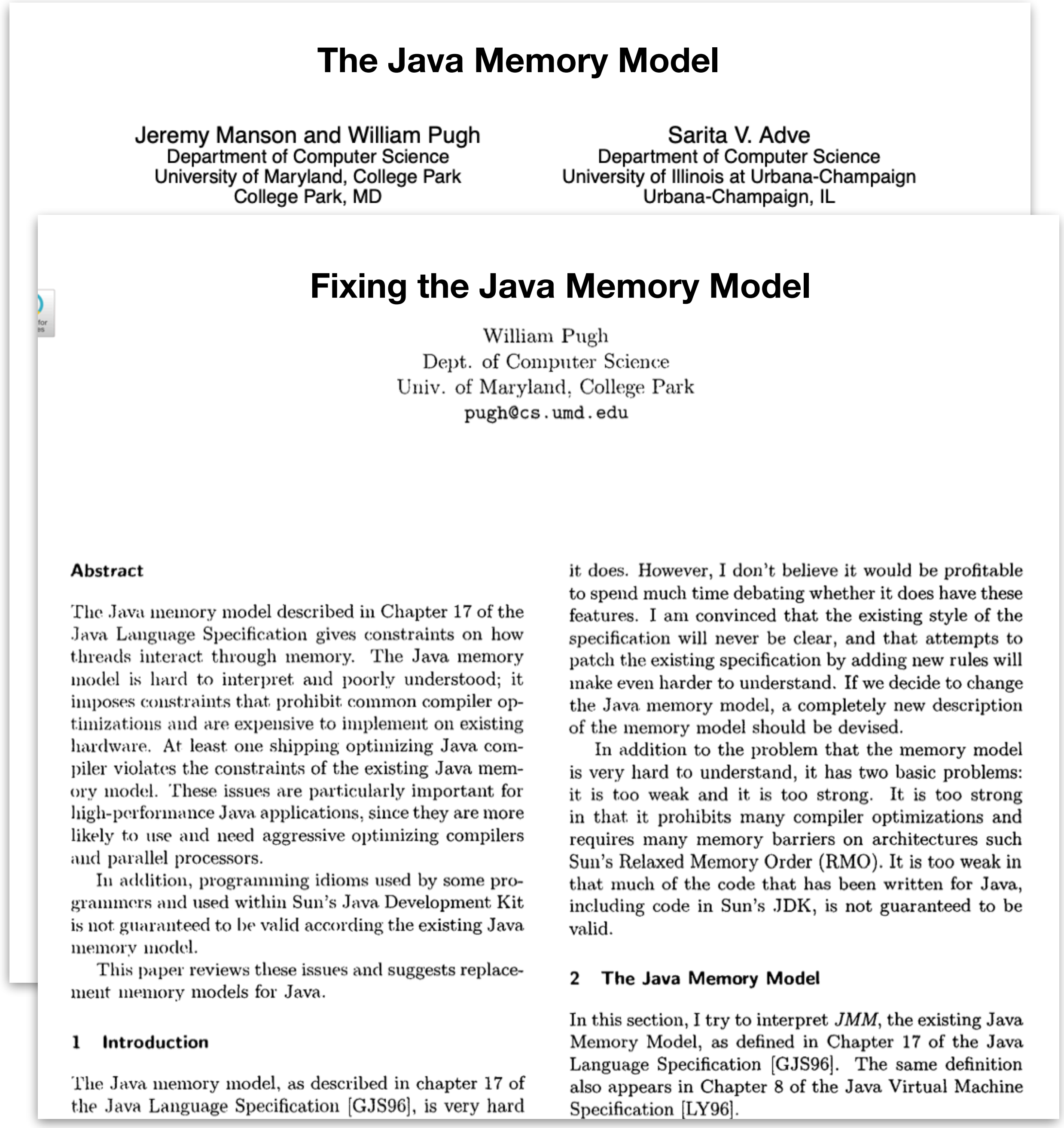
Cyclic Proofs for Axiomatic Memory Models

ongoing work with Jan Grünke and Thomas Haas

Memory Models have Bugs

Memory Models have Bugs

- Java MM *[JLS 1996]*
  - too weak to build new synchronization primitives *[Pugh]*
  - too strong for common compiler optimizations (i.e. CSE) *[Pugh]*



The Java Memory Model

Jeremy Manson and William Pugh
Department of Computer Science
University of Maryland, College Park
College Park, MD

Sarita V. Adve
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana-Champaign, IL

Fixing the Java Memory Model

William Pugh
Dept. of Computer Science
Univ. of Maryland, College Park
pugh@cs.umd.edu

Abstract

The Java memory model described in Chapter 17 of the Java Language Specification gives constraints on how threads interact through memory. The Java memory model is hard to interpret and poorly understood; it imposes constraints that prohibit common compiler optimizations and are expensive to implement on existing hardware. At least one shipping optimizing Java compiler violates the constraints of the existing Java memory model. These issues are particularly important for high-performance Java applications, since they are more likely to use and need aggressive optimizing compilers and parallel processors.

In addition, programming idioms used by some programmers and used within Sun's Java Development Kit is not guaranteed to be valid according the existing Java memory model.

This paper reviews these issues and suggests replacement memory models for Java.

1 Introduction

The Java memory model, as described in chapter 17 of the Java Language Specification [GJS96], is very hard

it does. However, I don't believe it would be profitable to spend much time debating whether it does have these features. I am convinced that the existing style of the specification will never be clear, and that attempts to patch the existing specification by adding new rules will make even harder to understand. If we decide to change the Java memory model, a completely new description of the memory model should be devised.

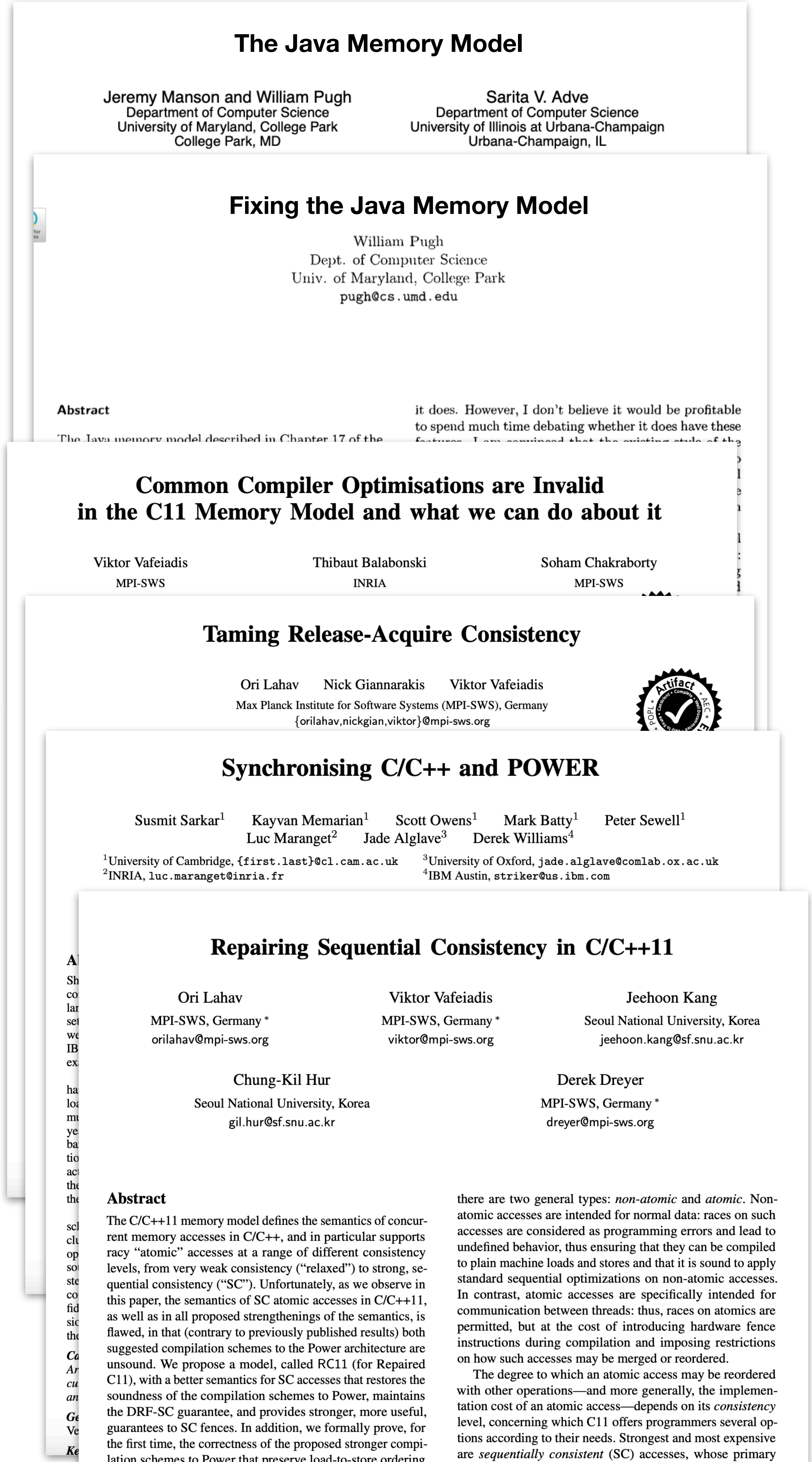
In addition to the problem that the memory model is very hard to understand, it has two basic problems: it is too weak and it is too strong. It is too strong in that it prohibits many compiler optimizations and requires many memory barriers on architectures such Sun's Relaxed Memory Order (RMO). It is too weak in that much of the code that has been written for Java, including code in Sun's JDK, is not guaranteed to be valid.

2 The Java Memory Model

In this section, I try to interpret *JMM*, the existing Java Memory Model, as defined in Chapter 17 of the Java Language Specification [GJS96]. The same definition also appears in Chapter 8 of the Java Virtual Machine Specification [LY96].

Memory Models have Bugs

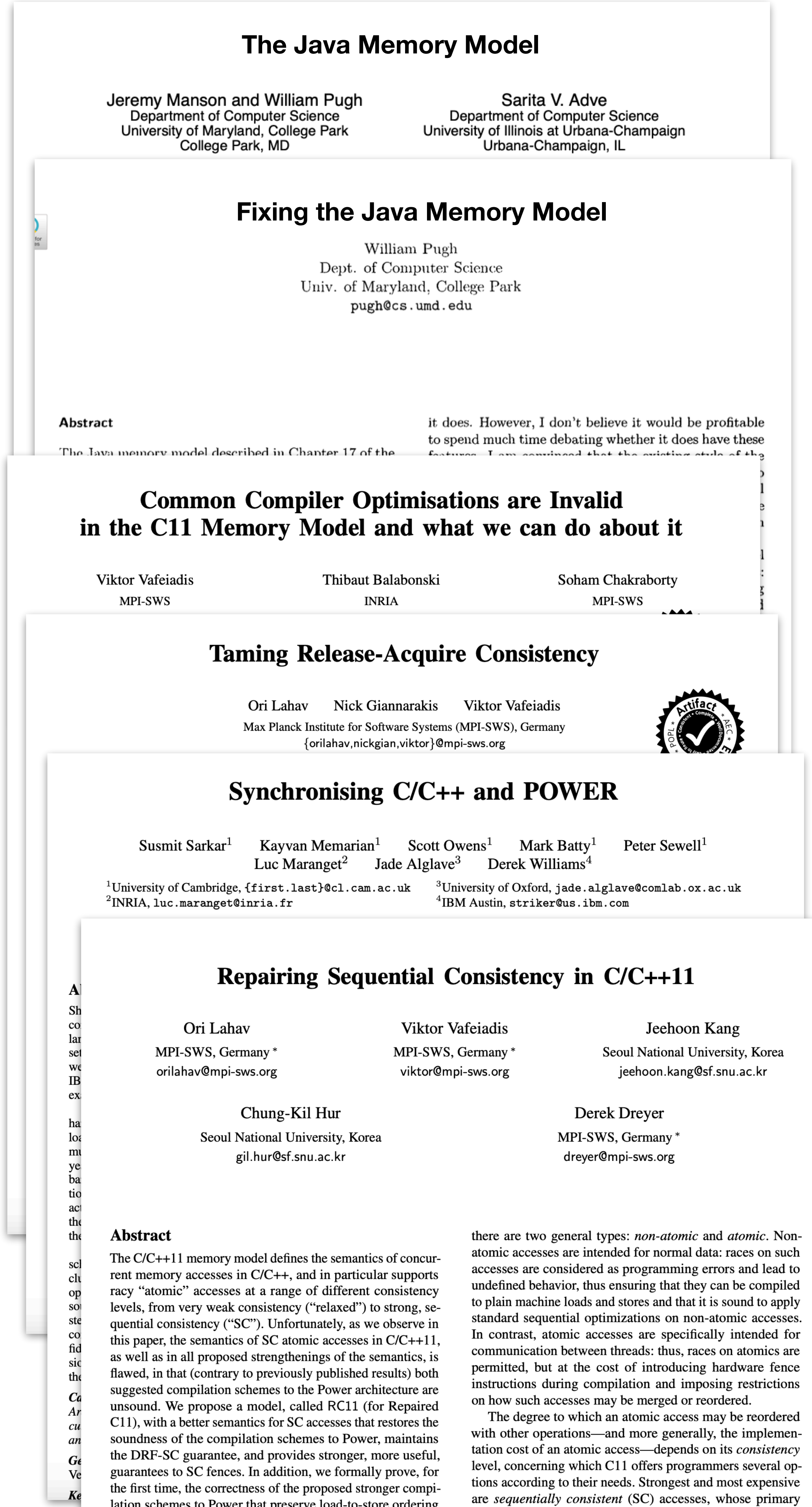
- Java MM *[JLS 1996]*
  - too weak to build new synchronization primitives *[Pugh]*
  - too strong for common compiler optimizations (i.e. CSE) *[Pugh]*
- C/C++11 MM
  - common compiler optimizations are invalid *[Vafeiadis et. al]*
  - allows strange behavior (i.e. OOTA) *[Vafeiadis et. al]*
  - SC fences are too weak *[Sarkar et. al]*
  - unsound compilation schemes to POWER *[Lahav et. al]*



Memory Models have Bugs

- Java MM *[JLS 1996]*
  - too weak to build new synchronization primitives *[Pugh]*
  - too strong for common compiler optimizations (i.e. CSE) *[Pugh]*
- C/C++11 MM
  - common compiler optimizations are invalid *[Vafeiadis et. al]*
  - allows strange behavior (i.e. OOTA) *[Vafeiadis et. al]*
  - SC fences are too weak *[Sarkar et. al]*
  - unsound compilation schemes to POWER *[Lahav et. al]*

➡ **Need for automatic Memory Model verification!**



Dartagnan

Checking the Linux Kernel

Dartagnan
Checking the Linux Kernel

Dartagnan found qspinlock to be broken (according to LKMM):
it was racy, failed to provide mutual exclusion, and could deadlock

Dartagnan

Checking the Linux Kernel

Dartagnan found qspinlock to be broken (according to LKMM):
it was racy, failed to provide mutual exclusion, and could deadlock

Antonio Paolillo, Hernán Ponce de León, Thomas Haas, Diogo Behrens, Rafael Lourenco de Lima Chehab, Ming Fu, Roland Meyer:
Verifying and Optimizing Compact NUMA-Aware Locks on Weak Memory Models @ arXiv

Dartagnan

Checking the Linux Kernel

Dartagnan found qspinlock to be broken (according to LKMM):
it was racy, failed to provide mutual exclusion, and could deadlock

Antonio Paolillo, Hernán Ponce de León, Thomas Haas, Diogo Behrens, Rafael Lourenco de Lima Chehab, Ming Fu, Roland Meyer: Verifying and Optimizing Compact NUMA-Aware Locks on Weak Memory Models @ arXiv

However: qspinlock runs fine on hardware (TSO, Power, ARMv8, RISCV)

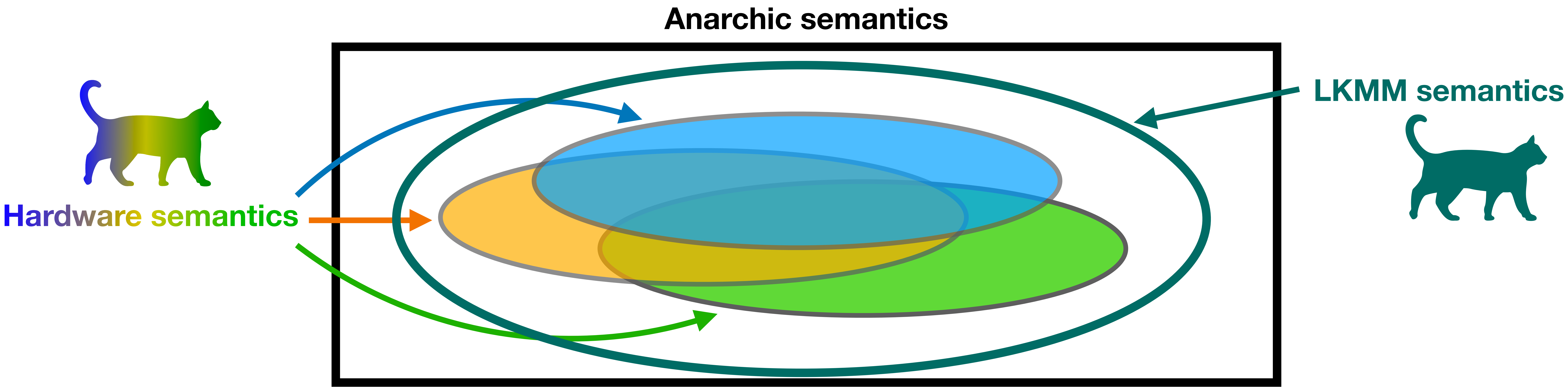
Dartagnan

Checking the Linux Kernel

Dartagnan found qspinlock to be broken (according to LKMM):
it was racy, failed to provide mutual exclusion, and could deadlock

Antonio Paolillo, Hernán Ponce de León, Thomas Haas, Diogo Behrens, Rafael Lourenco de Lima Chehab, Ming Fu, Roland Meyer: Verifying and Optimizing Compact NUMA-Aware Locks on Weak Memory Models @ arXiv

However: qspinlock runs fine on hardware (TSO, Power, ARMv8, RISCV)



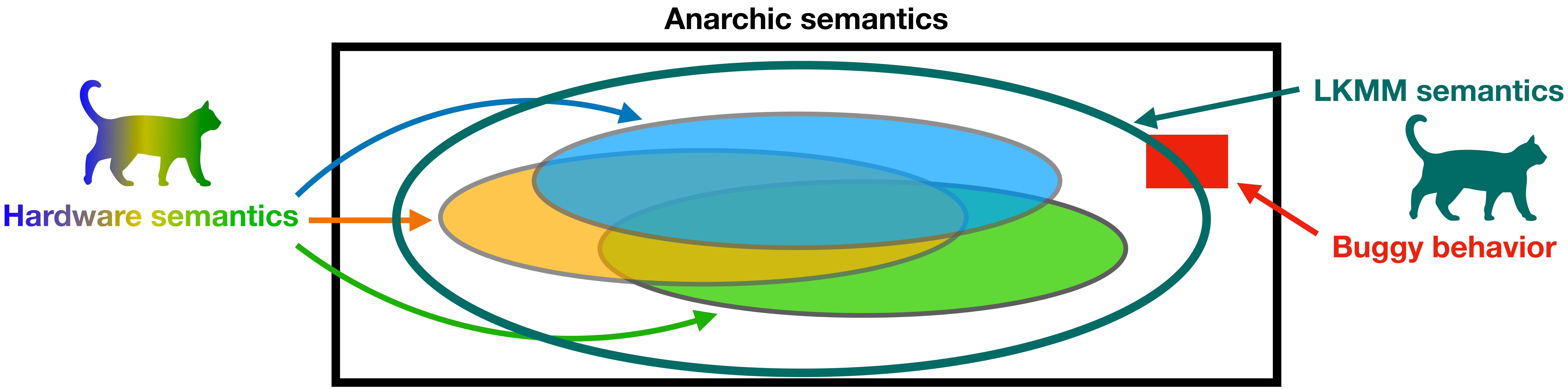
Dartagnan

Checking the Linux Kernel

Dartagnan found qspinlock to be broken (according to LKMM):
it was racy, failed to provide mutual exclusion, and could deadlock

Antonio Paolillo, Hernán Ponce de León, Thomas Haas, Diogo Behrens, Rafael Lourenco de Lima Chehab, Ming Fu, Roland Meyer: Verifying and Optimizing Compact NUMA-Aware Locks on Weak Memory Models @ arXiv

However: qspinlock runs fine on hardware (TSO, Power, ARMv8, RISCV)



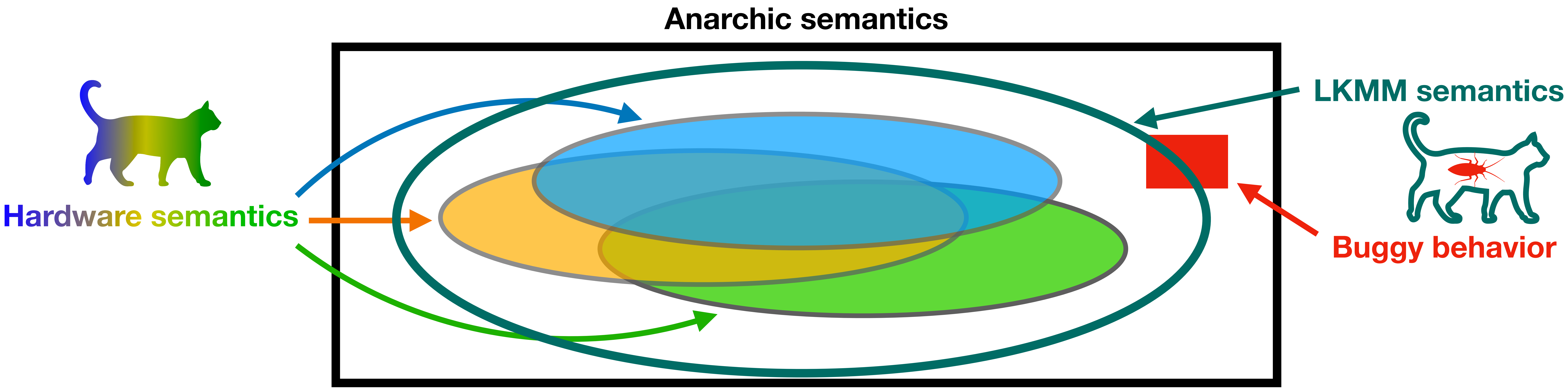
Dartagnan

Checking the Linux Kernel

Dartagnan found qspinlock to be broken (according to LKMM):
it was racy, failed to provide mutual exclusion, and could deadlock

Antonio Paolillo, Hernán Ponce de León, Thomas Haas, Diogo Behrens, Rafael Lourenco de Lima Chehab, Ming Fu, Roland Meyer:
Verifying and Optimizing Compact NUMA-Aware Locks on Weak Memory Models @ arXiv

However: qspinlock runs fine on hardware (TSO, Power, ARMv8, RISCV)



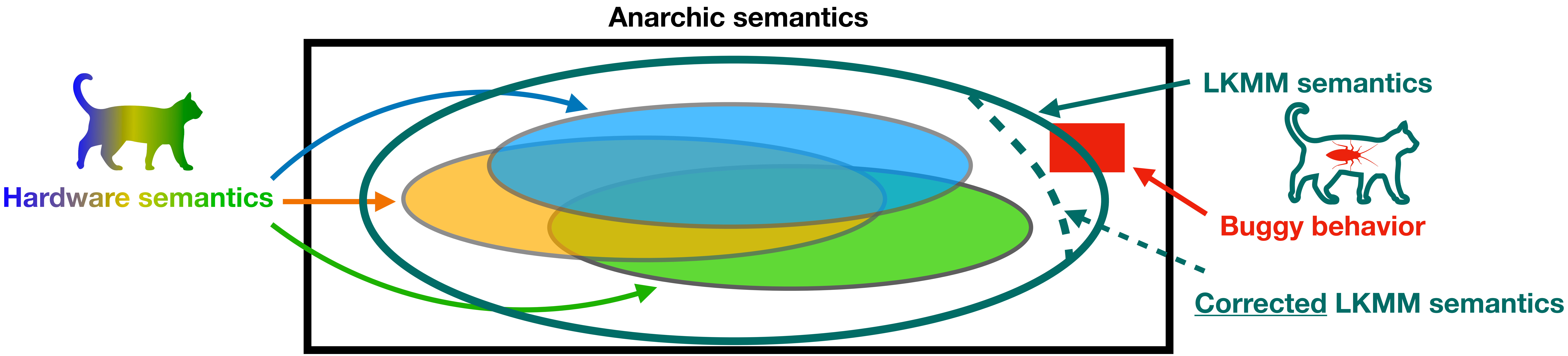
Dartagnan

Checking the Linux Kernel

Dartagnan found qspinlock to be broken (according to LKMM):
it was racy, failed to provide mutual exclusion, and could deadlock

Antonio Paolillo, Hernán Ponce de León, Thomas Haas, Diogo Behrens, Rafael Lourenco de Lima Chehab, Ming Fu, Roland Meyer: Verifying and Optimizing Compact NUMA-Aware Locks on Weak Memory Models @ arXiv

However: qspinlock runs fine on hardware (TSO, Power, ARMv8, RISCV)



Model Checking Memory Models

Given: Memory Models M_1, M_2

Question: Is M_1 weaker than M_2?

Model Checking Memory Models

Given: Memory Models M_1, M_2

Question: Is M_1 weaker than M_2?

Approach: Check inclusion between relational algebra expressions

Example: TSO is weaker than SC (acyclic hb$_{SC}$ $\implies$ acyclic hb$_{TSO}$)

Model Checking Memory Models

Given: Memory Models M_1, M_2

Question: Is M_1 weaker than M_2?

Approach: Check inclusion between relational algebra expressions

Example: TSO is weaker than SC (acyclic hb_{SC} $\implies$ acyclic hb_{TSO})

$\mathsf{hb}^+_{TSO} \cap \mathsf{id} \subseteq \top; (\mathsf{hb}^+_{SC} \cap \mathsf{id}); \top$

Model Checking Memory Models

Given: Memory Models M_1, M_2

Question: Is M_1 weaker than M_2?

Approach: Check inclusion between relational algebra expressions

Example: TSO is weaker than SC (acyclic $\text{hb}_{SC} \implies$ acyclic hb_{TSO})

$\text{hb}^+_{TSO} \cap \text{id} \subseteq \top; (\text{hb}^+_{SC} \cap \text{id}); \top$

- KATER tool for a **restricted fragment** (regular language inclusion) *[Kokologiannakis, 2023]*

$a \quad | \quad r_1 \cup r_2 \quad | \quad r_1 . r_2 \quad | \quad r*$

Model Checking Memory Models

Given: Memory Models M_1, M_2

Question: Is M_1 weaker than M_2?

Approach: Check inclusion between relational algebra expressions

Example: TSO is weaker than SC (acyclic hb_{SC} $\implies$ acyclic hb_{TSO})

$\text{hb}^+_{TSO} \cap \text{id} \subseteq \top; (\text{hb}^+_{SC} \cap \text{id}); \top$

- KATER tool for a **restricted fragment** (regular language inclusion) *[Kokologiannakis, 2023]*

  $a \quad | \quad r_1 \cup r_2 \quad | \quad r_1 \,.\, r_2 \quad | \quad r*$

- Our tool supports the **regular fragment** (based on cyclic proofs)

  $\ldots \quad | \quad r_1 \cap r_2 \quad | \quad r^{-1} \quad | \quad s_1 \times s_2$

Model Checking Memory Models

Given: Memory Models M_1, M_2

Question: Is M_1 weaker than M_2?

Approach: Check inclusion between relational algebra expressions

Example: TSO is weaker than SC (acyclic hb_{SC} $\implies$ acyclic hb_{TSO})

$$\mathsf{hb}_{TSO}^{+} \cap \mathsf{id} \subseteq \top; (\mathsf{hb}_{SC}^{+} \cap \mathsf{id}); \top$$

- KATER tool for a **restricted fragment** (regular language inclusion) *[Kokologiannakis, 2023]*

$$a \quad | \quad r_1 \cup r_2 \quad | \quad r_1 . r_2 \quad | \quad r*$$

- Our tool supports the **regular fragment** (based on cyclic proofs)

$$\ldots \quad | \quad r_1 \cap r_2 \quad | \quad r^{-1} \quad | \quad s_1 \times s_2$$

MM like LKMM are in this fragment!

Model Checking Memory Models

Given: Memory Models M_1, M_2

Question: Is M_1 weaker than M_2?

Relational $\top$,
cycle somewhere!

Approach: Check inclusion between relational algebra expressions

Example: TSO is weaker than SC (acyclic $\text{hb}_{SC} \implies$ acyclic hb_{TSO})

$\text{hb}^+_{TSO} \cap \text{id} \subseteq \top; (\text{hb}^+_{SC} \cap \text{id}); \top$

- KATER tool for a **restricted fragment** (regular language inclusion) *[Kokologiannakis, 2023]*

$a \quad | \quad r_1 \cup r_2 \quad | \quad r_1 . r_2 \quad | \quad r^*$

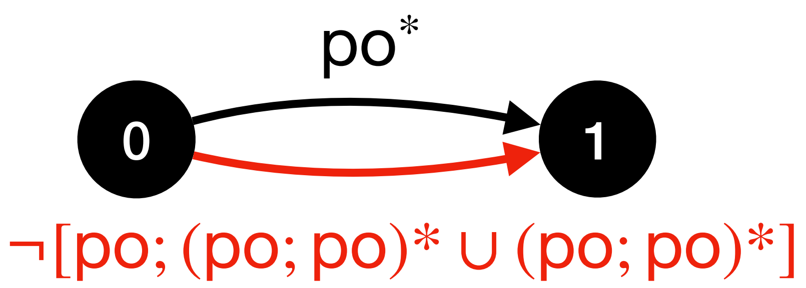
- Our tool supports the **regular fragment** (based on cyclic proofs)

$\ldots \quad | \quad r_1 \cap r_2 \quad | \quad r^{-1} \quad | \quad s_1 \times s_2$

MM like LKMM are in this fragment!

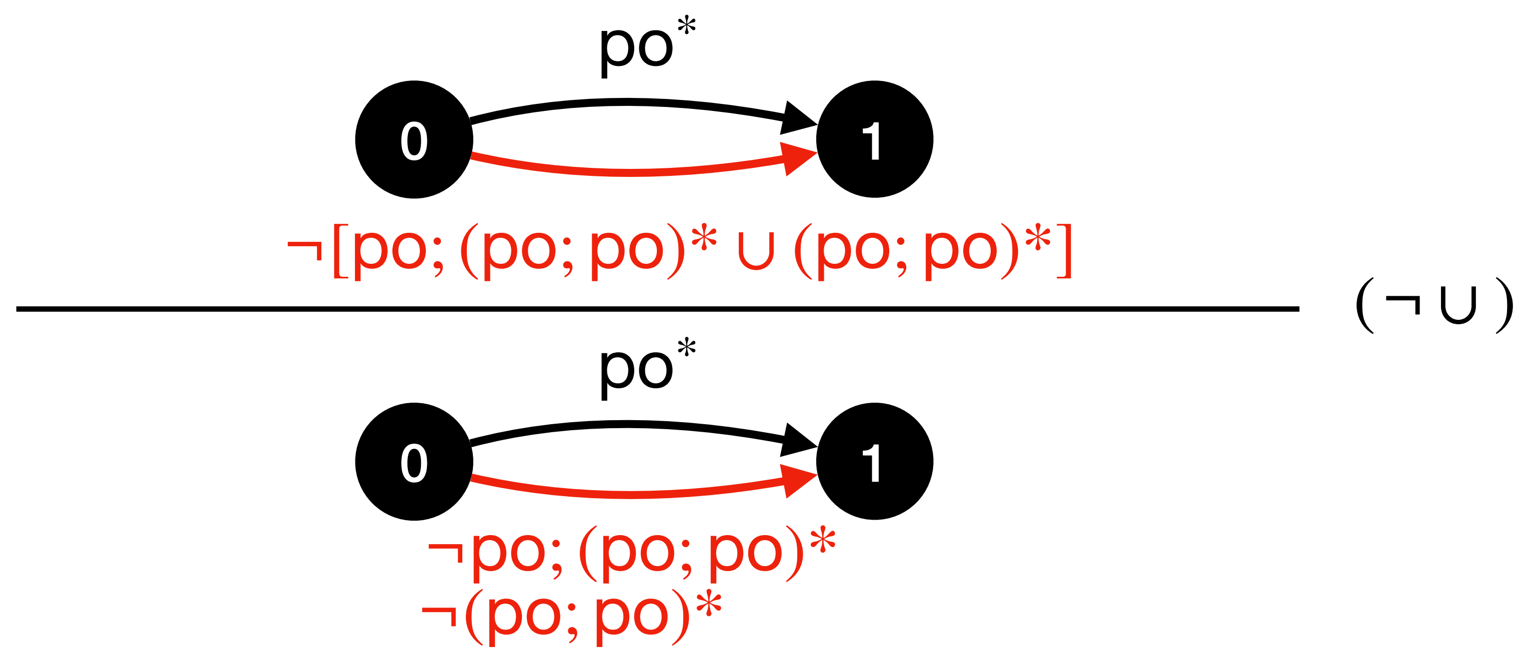
Cyclic Proof System

To prove $\mathrm{po}^* \subseteq \mathrm{po};(\mathrm{po};\mathrm{po})^* \cup (\mathrm{po};\mathrm{po})^*$ a proof tries to find a counterexample



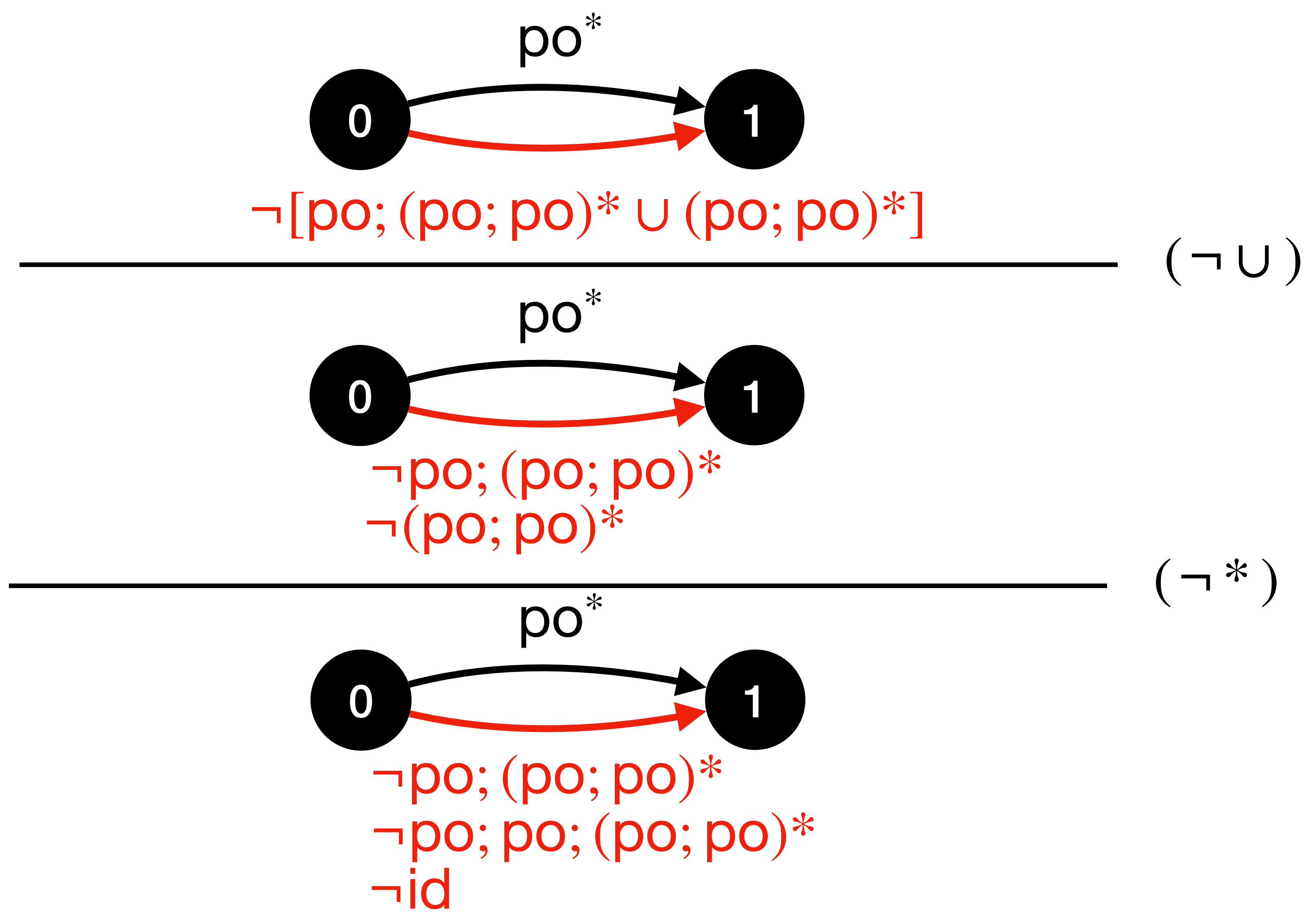
Cyclic Proof System

To prove $po^* \subseteq po;(po;po)^* \cup (po;po)^*$ a proof tries to find a counterexample



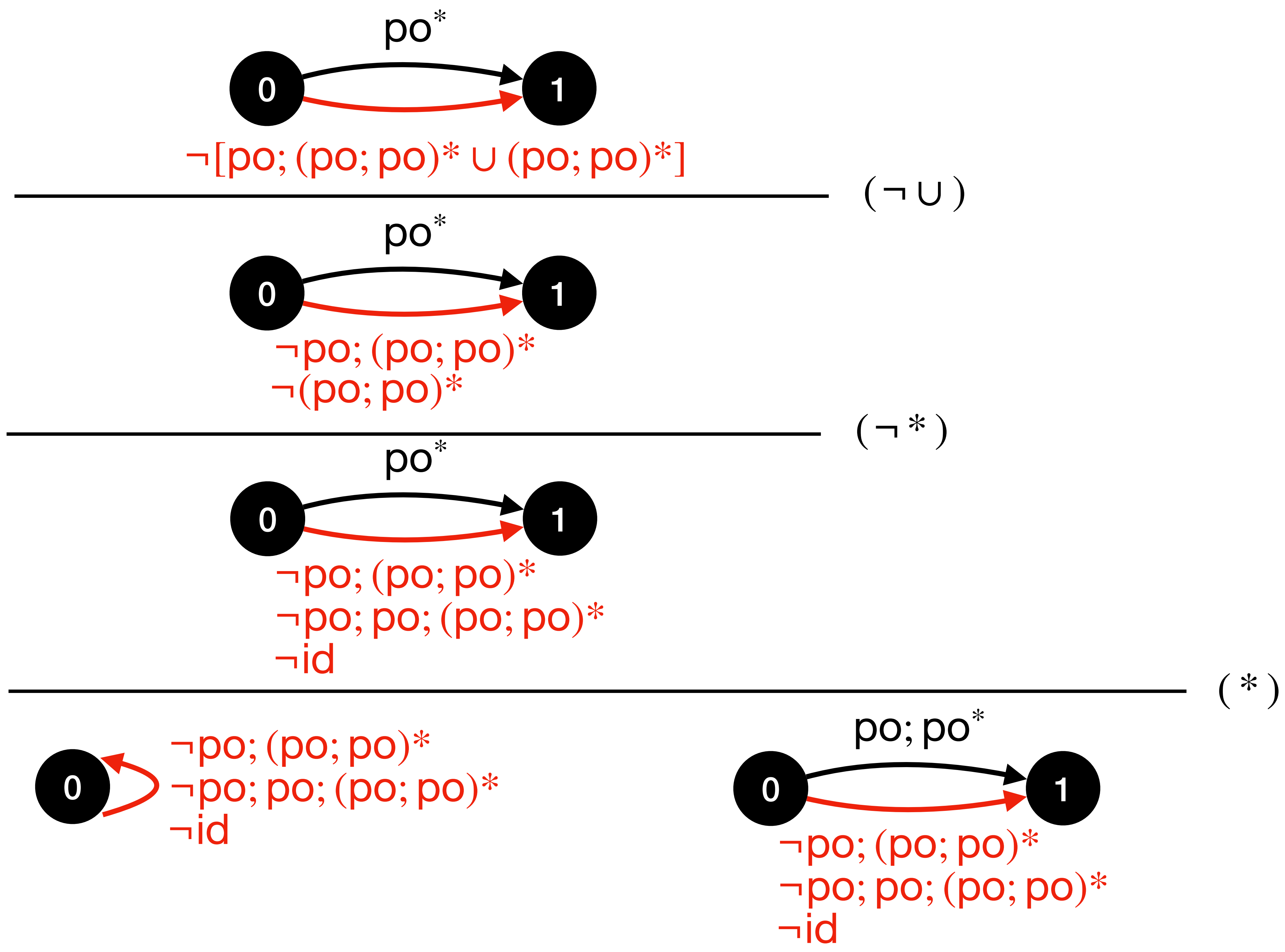
Cyclic Proof System

To prove $po^* \subseteq po; (po; po)^* \cup (po; po)^*$ a proof tries to find a counterexample



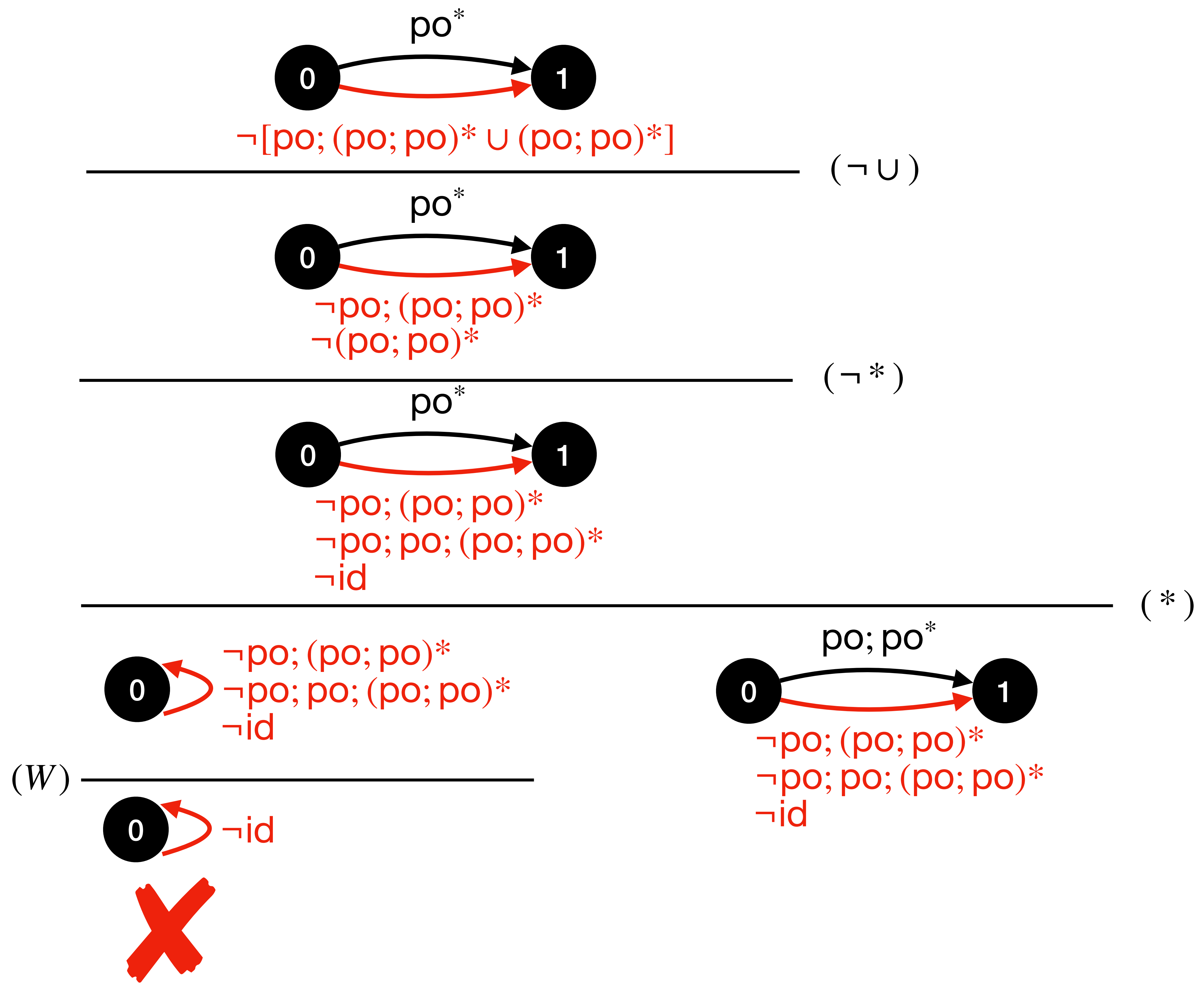
Cyclic Proof System

To prove $po^* \subseteq po; (po; po)^* \cup (po; po)^*$ a proof tries to find a counterexample



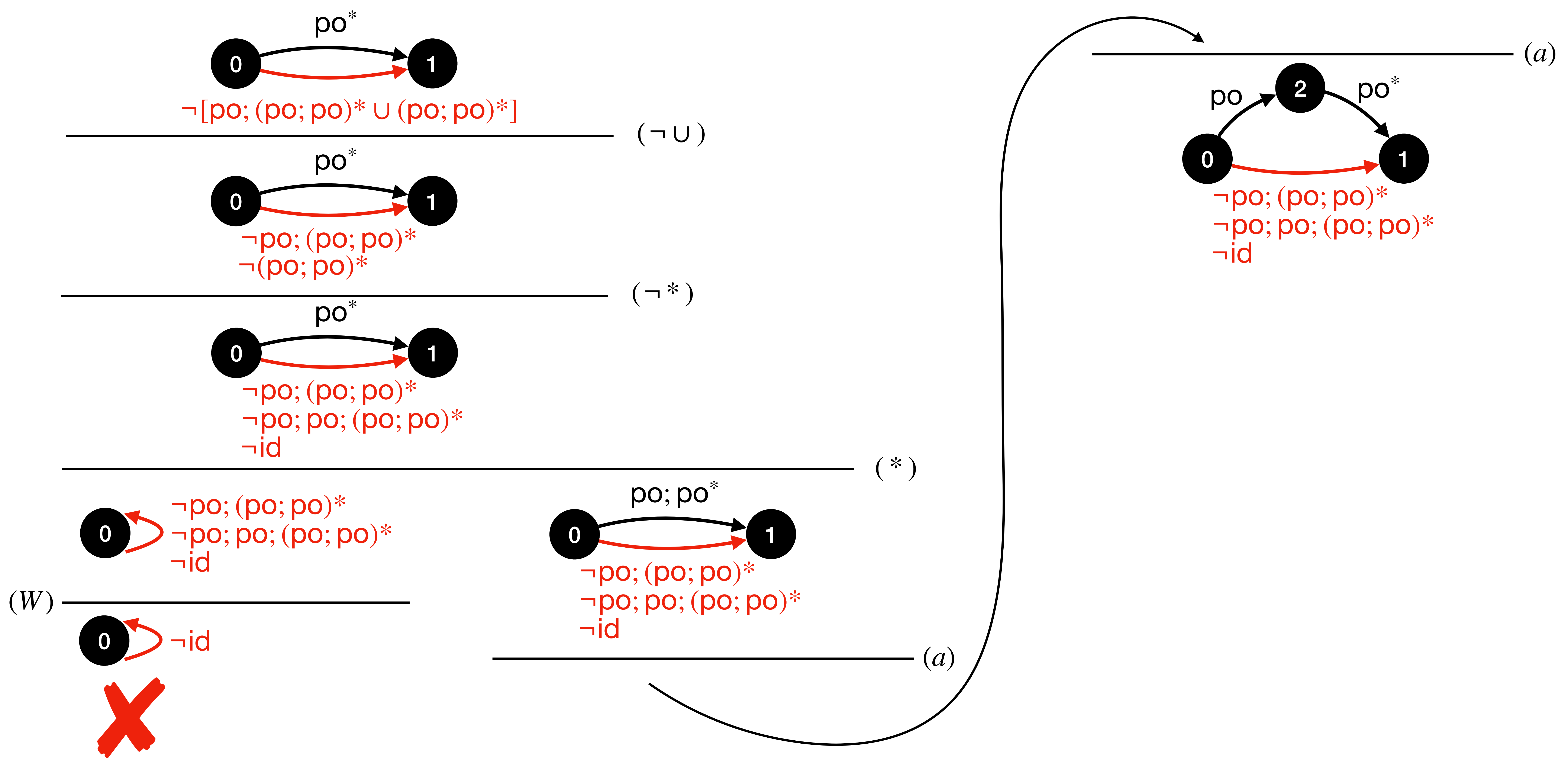
Cyclic Proof System

To prove $\mathsf{po}^* \subseteq \mathsf{po};(\mathsf{po};\mathsf{po})^* \cup (\mathsf{po};\mathsf{po})^*$ a proof tries to find a counterexample



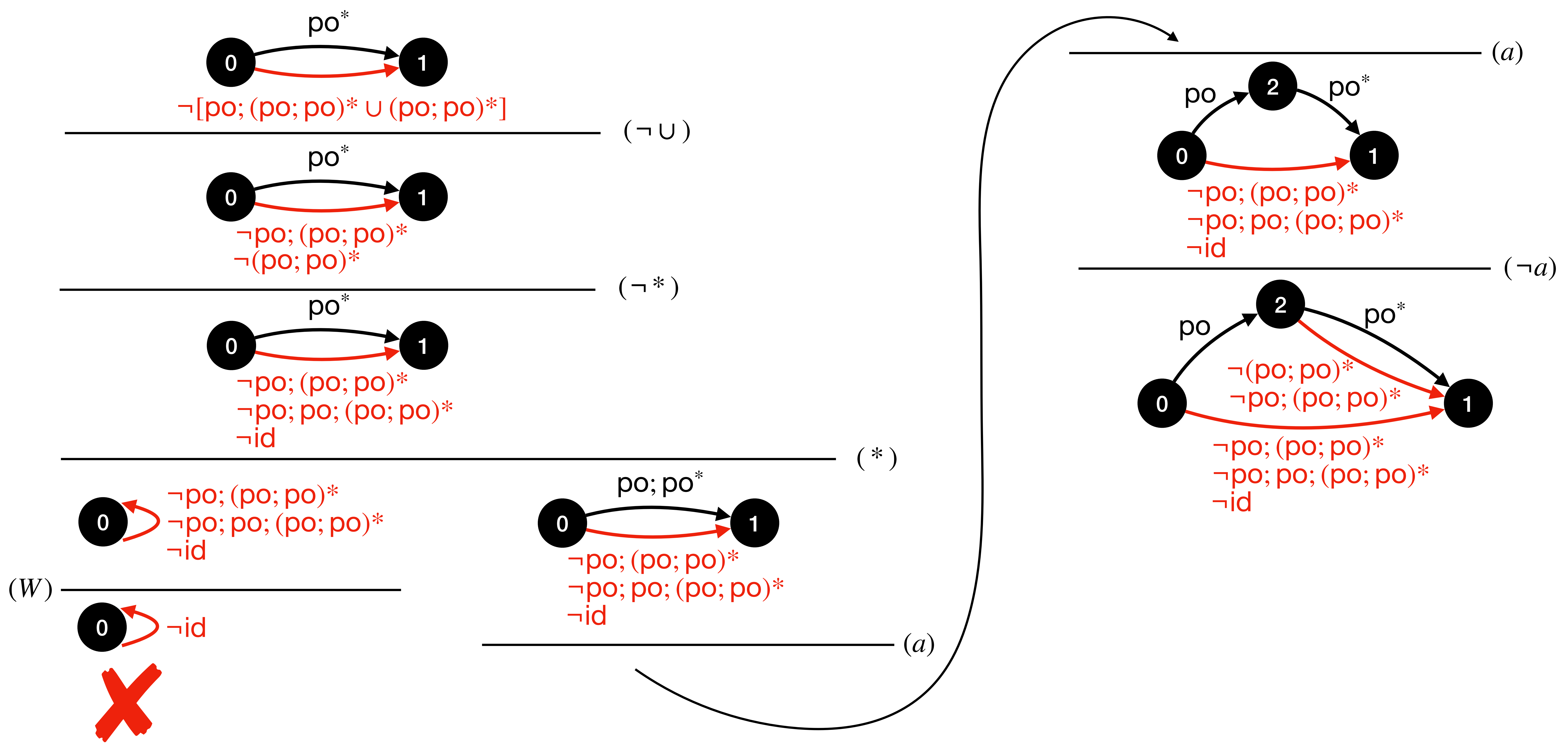
Cyclic Proof System

To prove $\mathsf{po}^* \subseteq \mathsf{po}; (\mathsf{po}; \mathsf{po})^* \cup (\mathsf{po}; \mathsf{po})^*$ a proof tries to find a counterexample



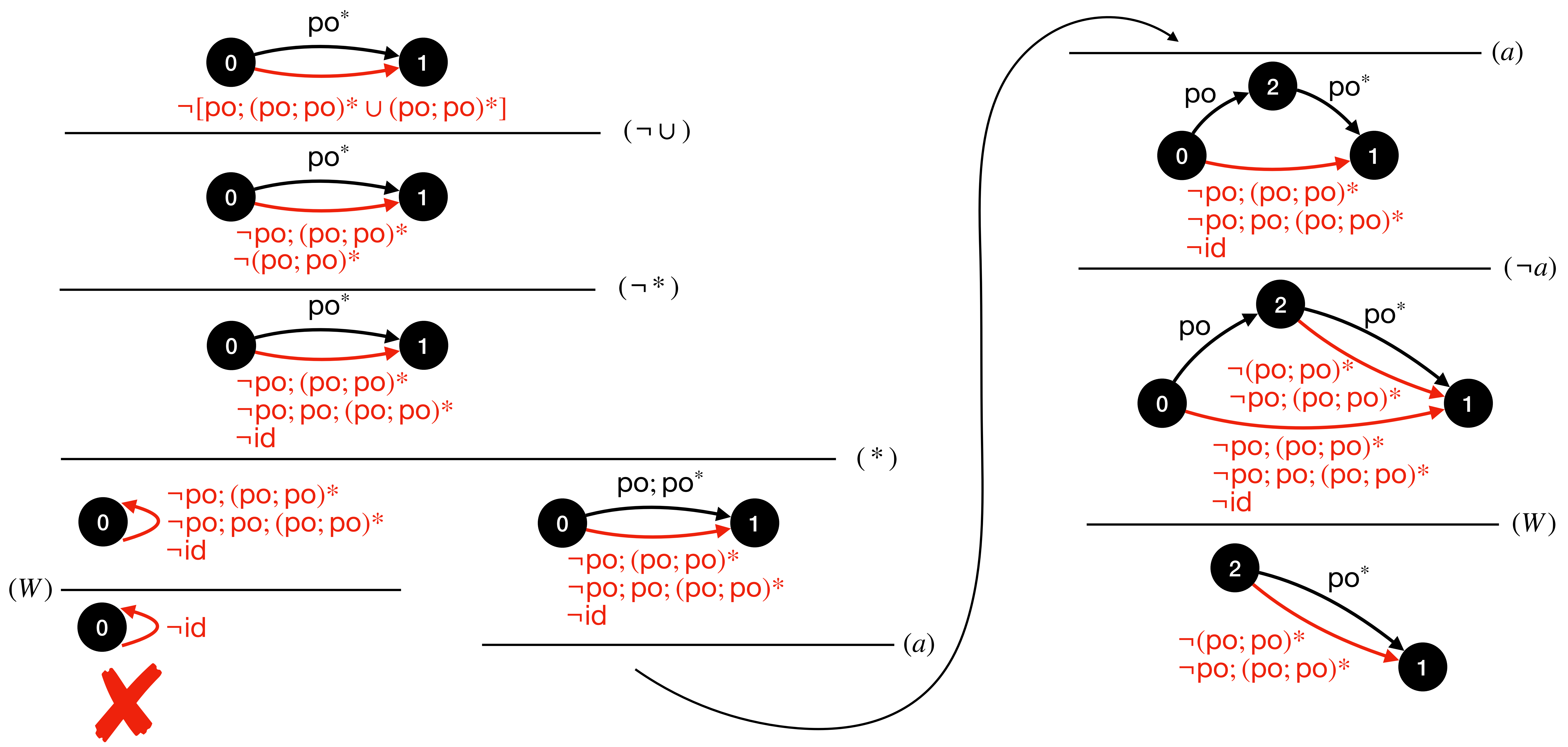
Cyclic Proof System

To prove $\text{po}^* \subseteq \text{po}; (\text{po}; \text{po})^* \cup (\text{po}; \text{po})^*$ a proof tries to find a counterexample



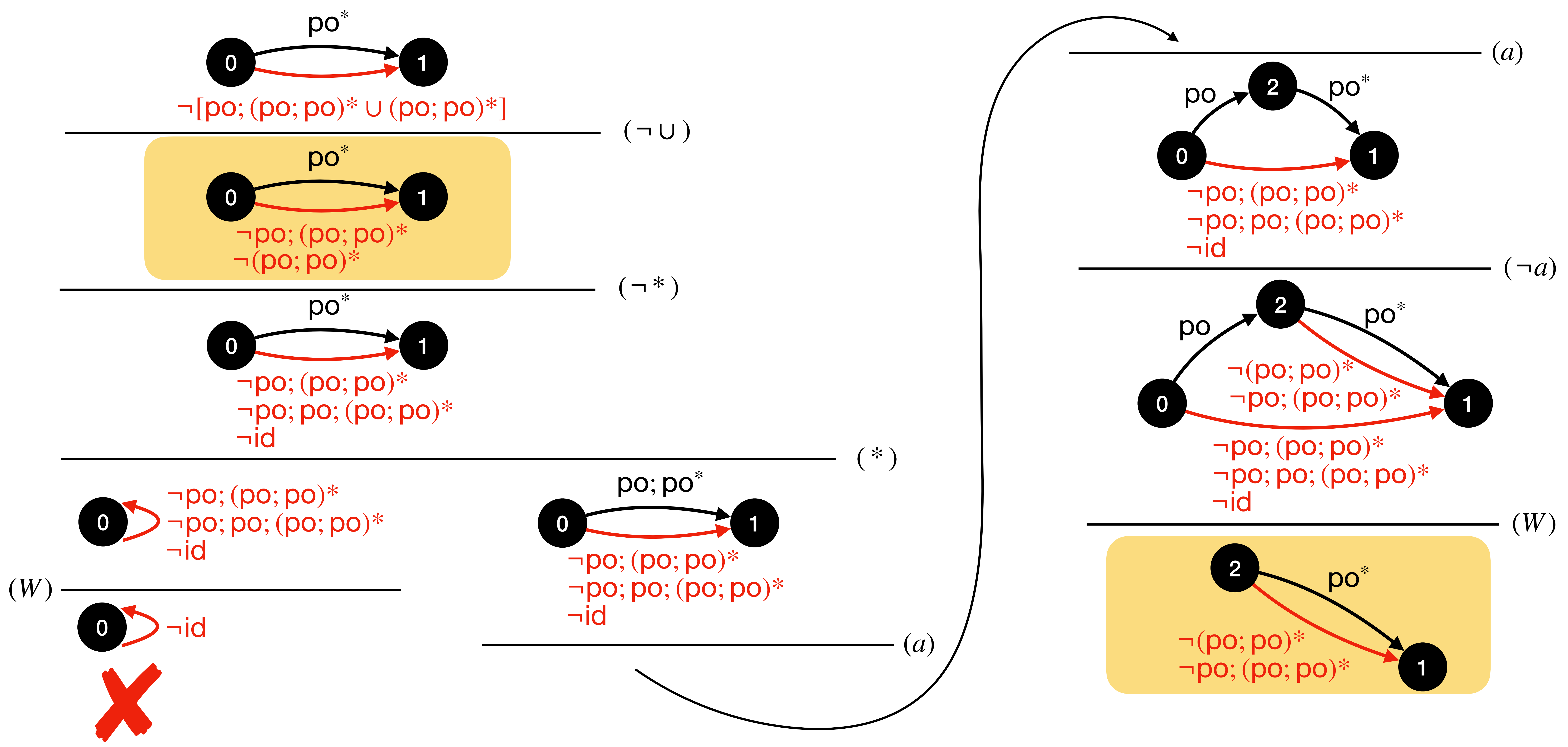
Cyclic Proof System

To prove $po^* \subseteq po; (po; po)^* \cup (po; po)^*$ a proof tries to find a counterexample



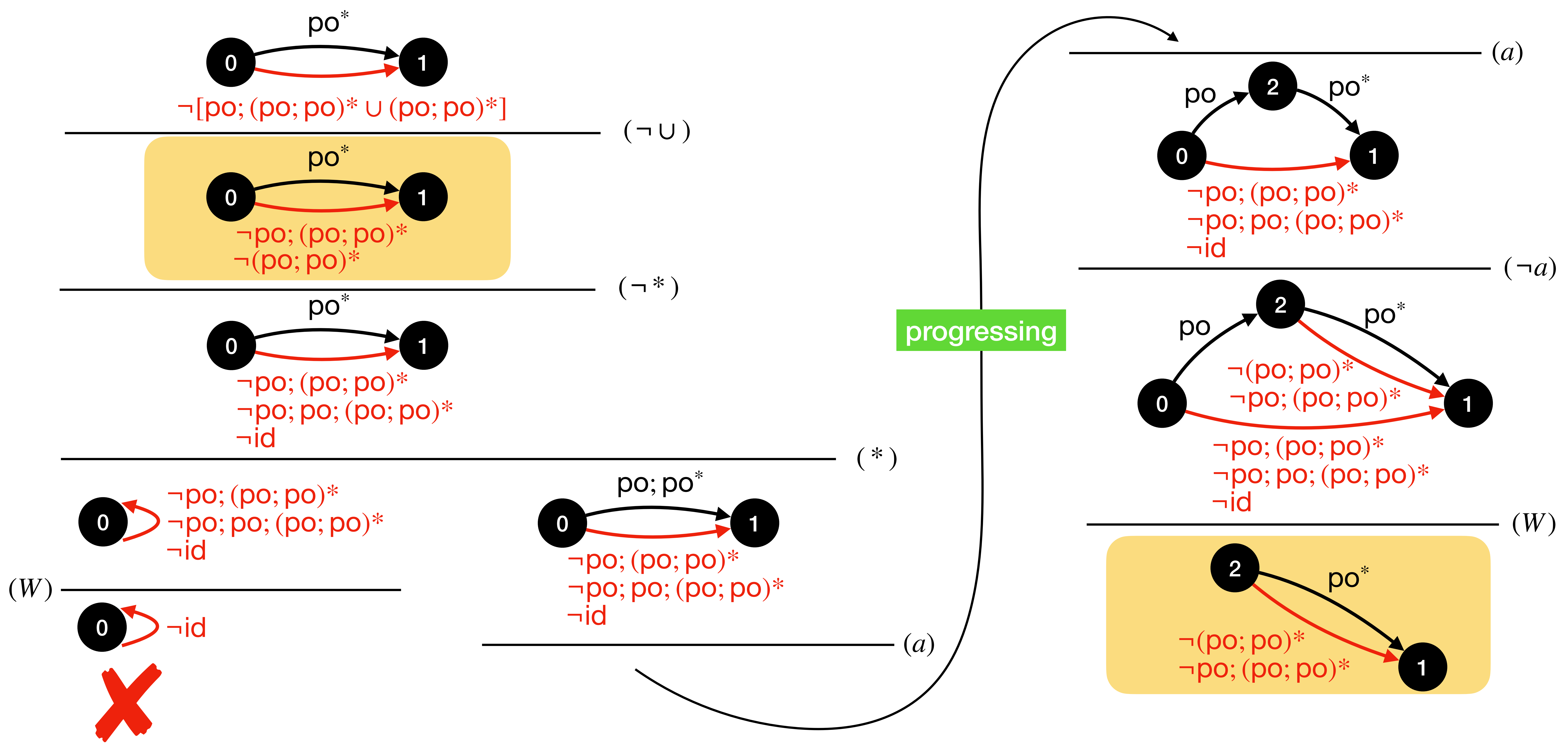
Cyclic Proof System

To prove $\mathsf{po}^* \subseteq \mathsf{po};(\mathsf{po};\mathsf{po})^* \cup (\mathsf{po};\mathsf{po})^*$ a proof tries to find a counterexample



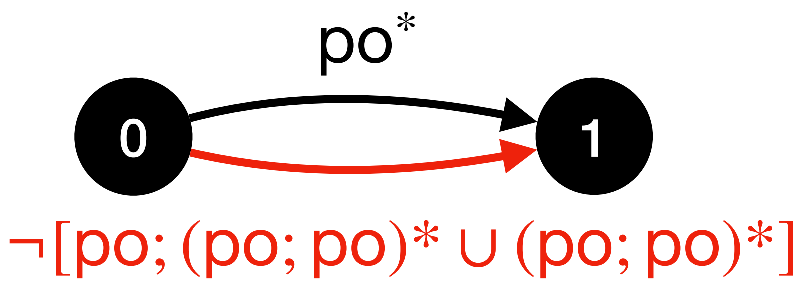
Cyclic Proof System

To prove $\mathrm{po}^* \subseteq \mathrm{po}; (\mathrm{po}; \mathrm{po})^* \cup (\mathrm{po}; \mathrm{po})^*$ a proof tries to find a counterexample



Cyclic Proof System

- Graphs = represented by relational algebra expressions + **event symbols**

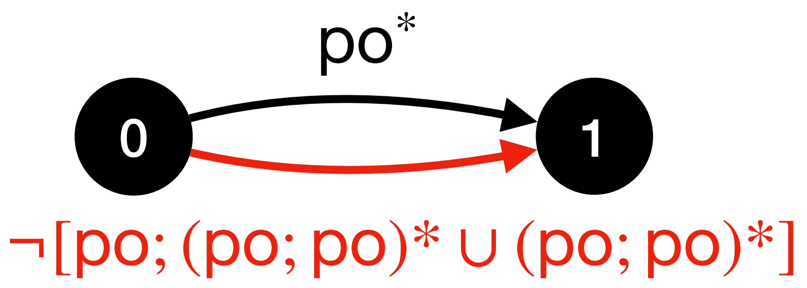


=

$$0; \mathrm{po}^* \cap 1$$
$$\neg[0; [\mathrm{po}; (\mathrm{po}; \mathrm{po})^* \cup (\mathrm{po}; \mathrm{po})^*] \cap 1]$$

Cyclic Proof System

- Graphs = represented by relational algebra expressions + **event symbols**

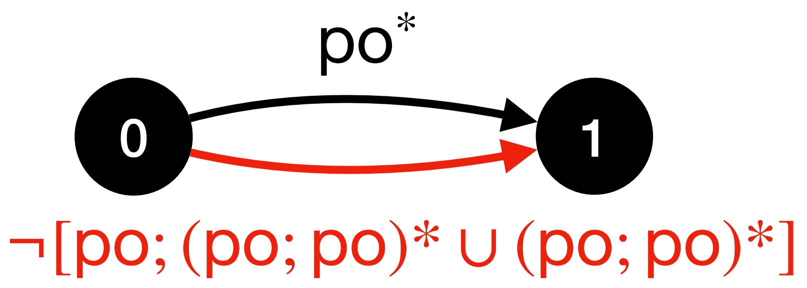


$$= \quad \begin{array}{l} 0; \mathrm{po}^* \cap 1 \\ \neg[0; [\mathrm{po}; (\mathrm{po}; \mathrm{po})^* \cup (\mathrm{po}; \mathrm{po})^*] \cap 1] \end{array}$$

- Proof system is sound + complete for relational algebra inclusions

Cyclic Proof System

- Graphs = represented by relational algebra expressions + **event symbols**



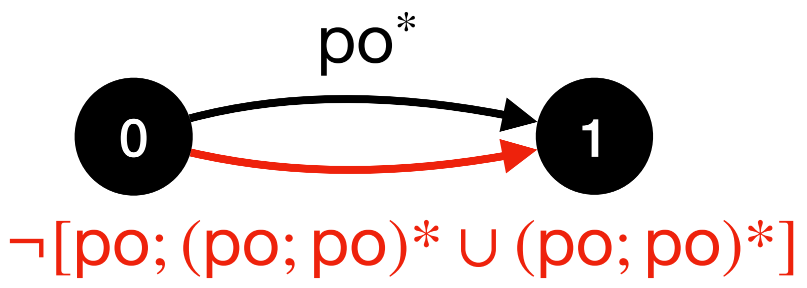
$$= \quad 0; \mathrm{po}^* \cap 1 \quad \neg[0; [\mathrm{po}; (\mathrm{po}; \mathrm{po})^* \cup (\mathrm{po}; \mathrm{po})^*] \cap 1]$$

- Proof system is sound + complete for relational algebra inclusions

Bound number of events symbols

Cyclic Proof System

- Graphs = represented by relational algebra expressions + **event symbols**



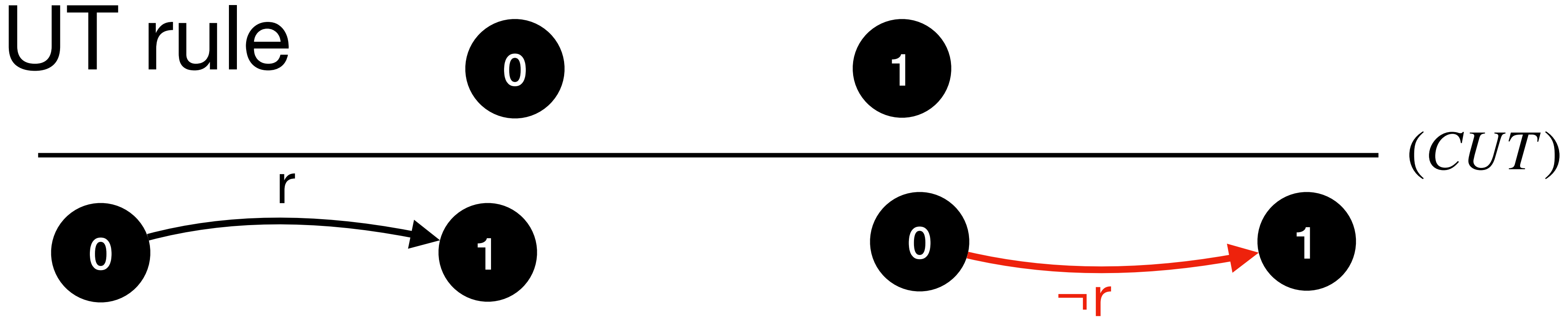
=

$$0; \mathrm{po}^* \cap 1$$
$$\neg[0; [\mathrm{po}; (\mathrm{po}; \mathrm{po})^* \cup (\mathrm{po}; \mathrm{po})^*] \cap 1]$$

- Proof system is sound + complete for relational algebra inclusions

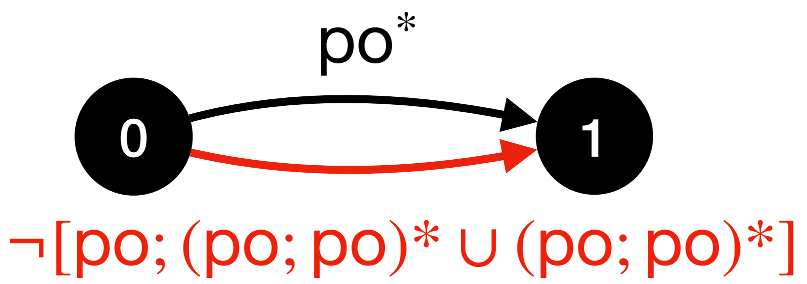
Bound number of events symbols

CUT rule



Cyclic Proof System

- Graphs = represented by relational algebra expressions + **event symbols**



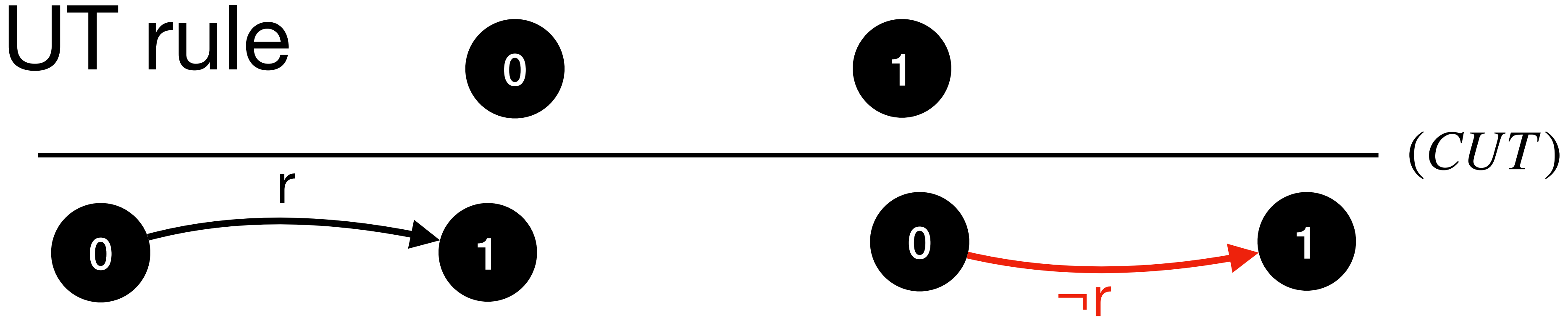
=

$$0; \mathrm{po}^* \cap 1$$
$$\neg[0; [\mathrm{po}; (\mathrm{po}; \mathrm{po})^* \cup (\mathrm{po}; \mathrm{po})^*] \cap 1]$$

- Proof system is sound + complete for relational algebra inclusions

Bound number of events symbols

CUT rule



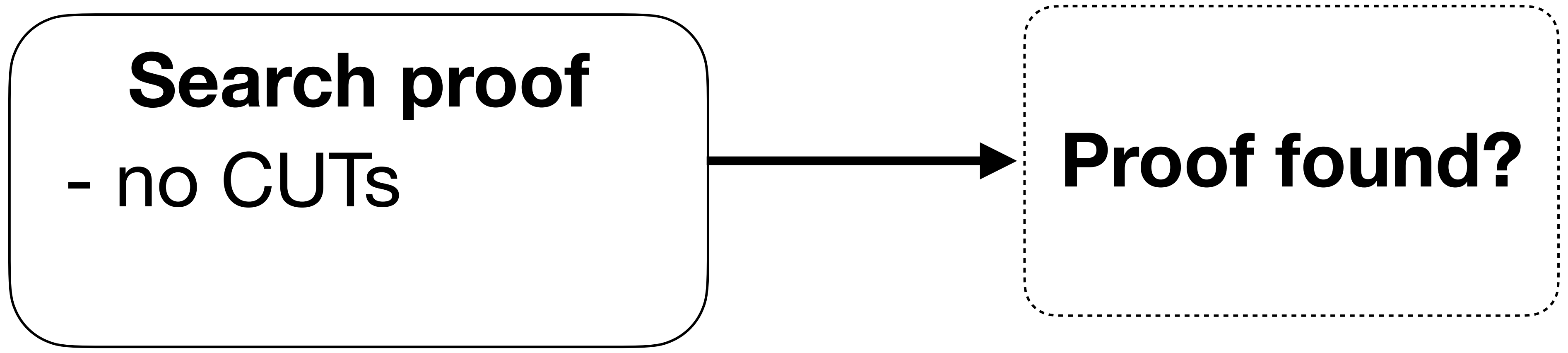
➡ **Naive proof search is inefficient (EXPSPACE-complete)**

CEGAR Proof Search

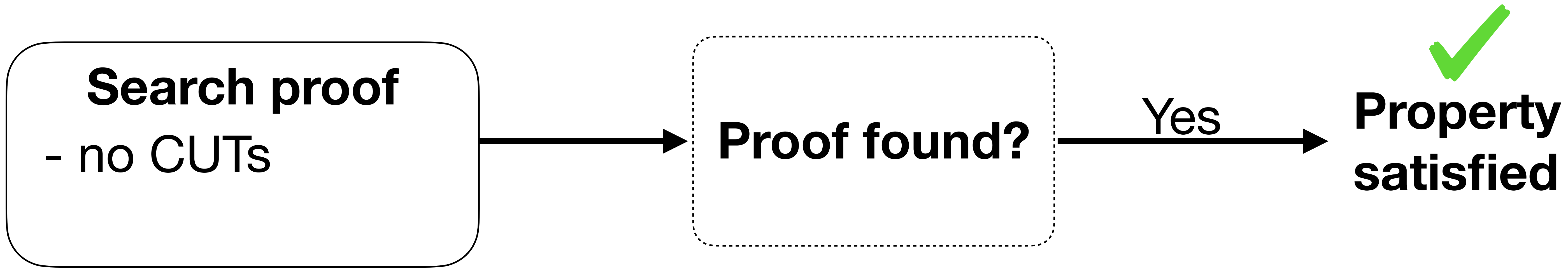
Search proof

- no CUTs

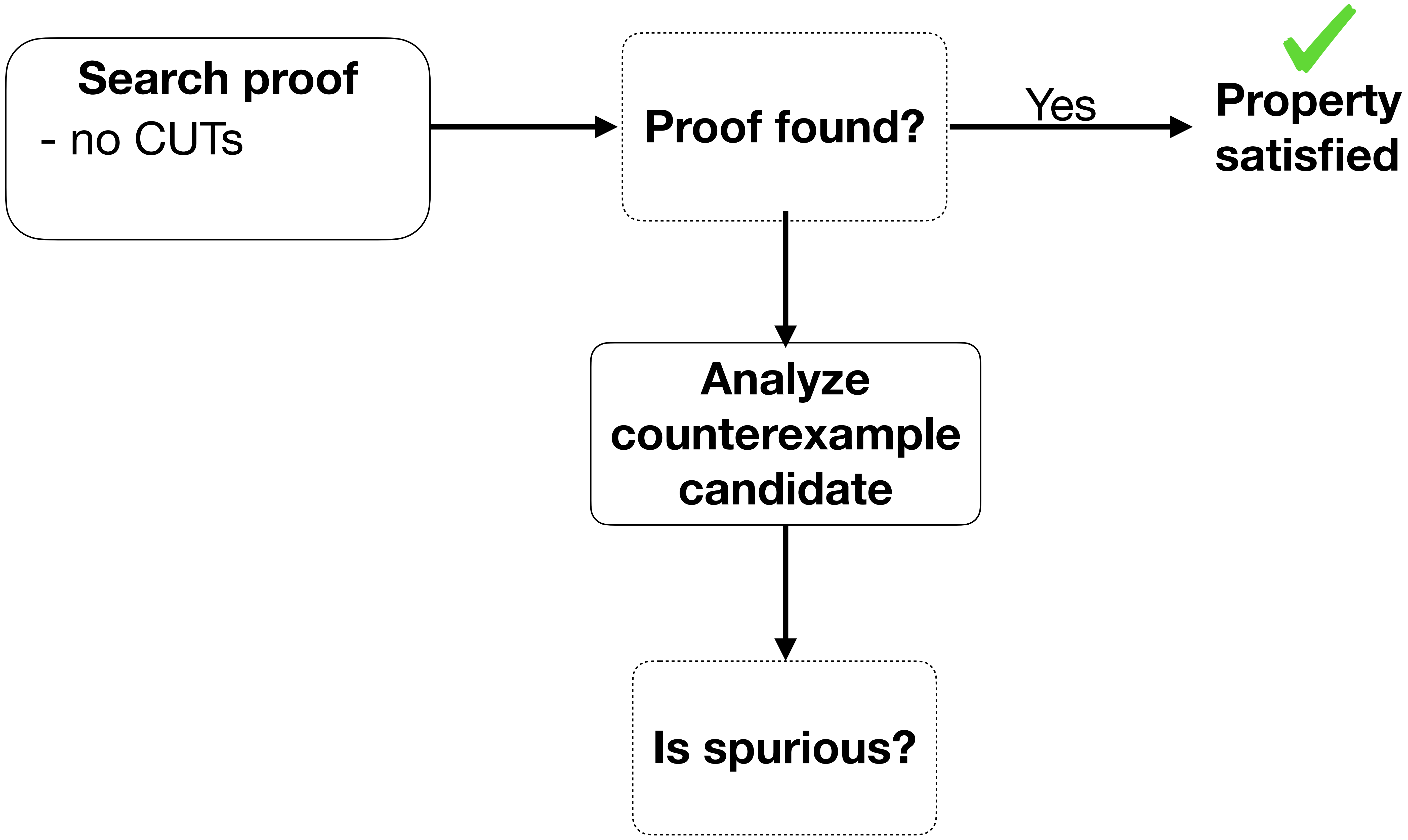
CEGAR Proof Search



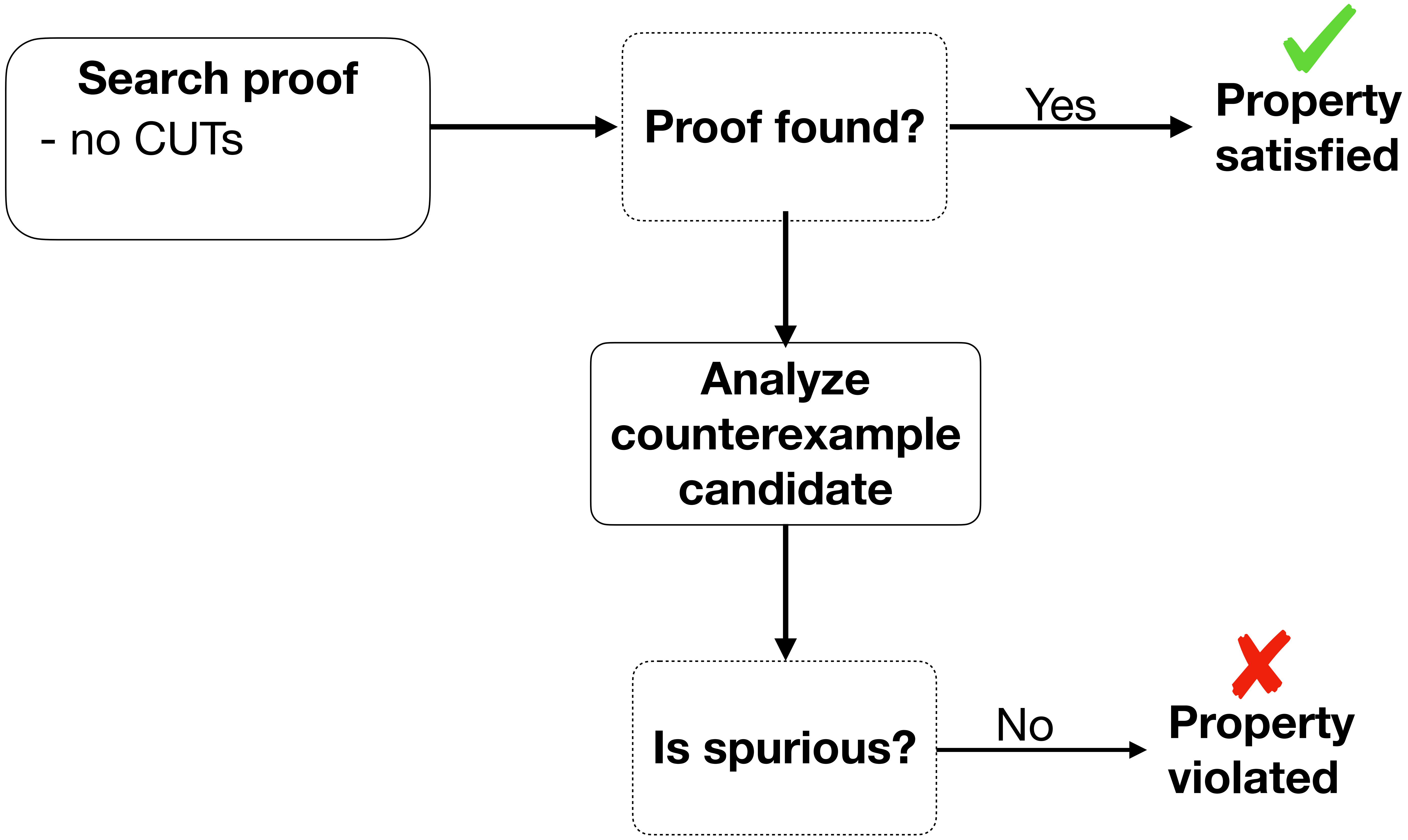
CEGAR Proof Search



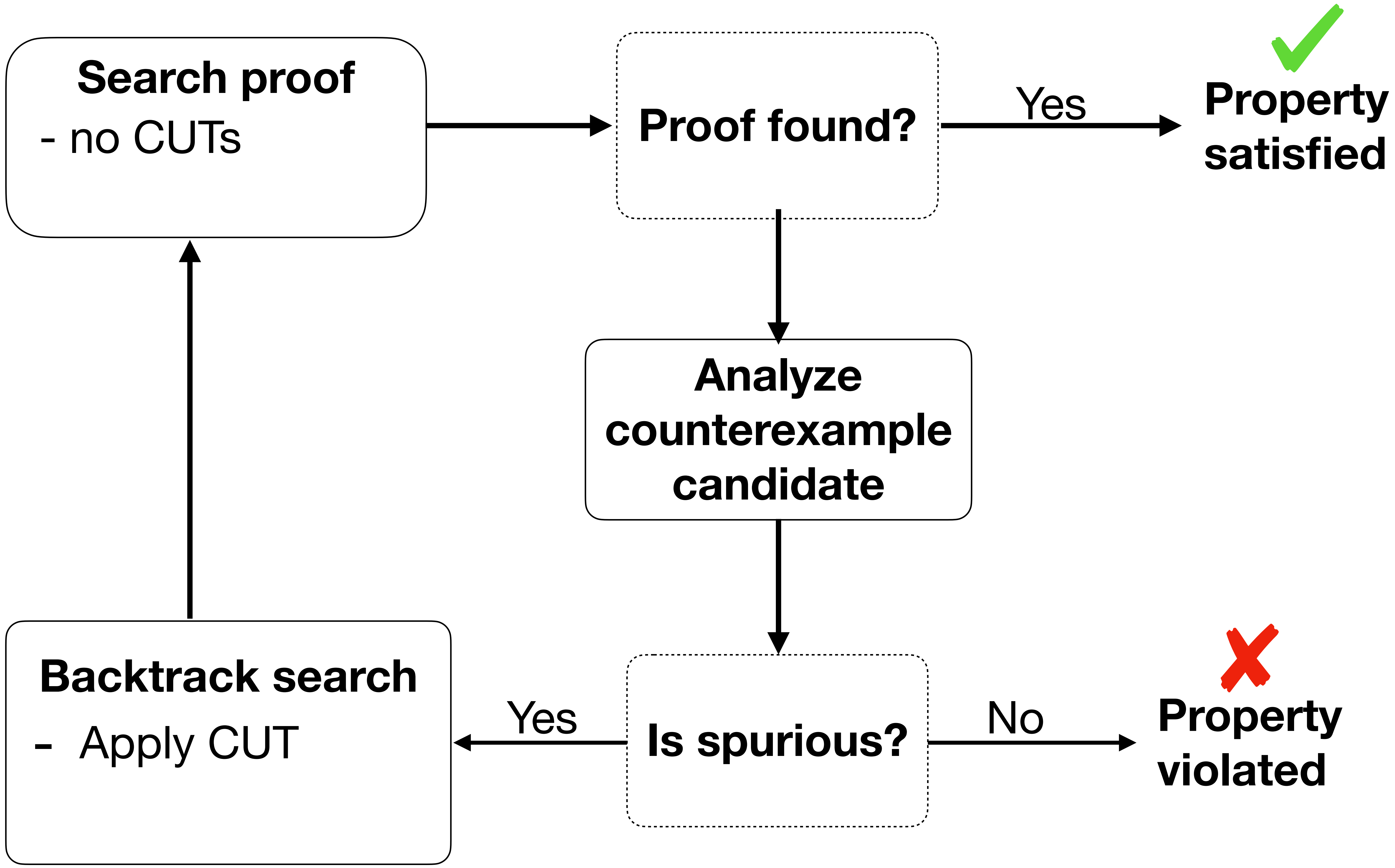
CEGAR Proof Search



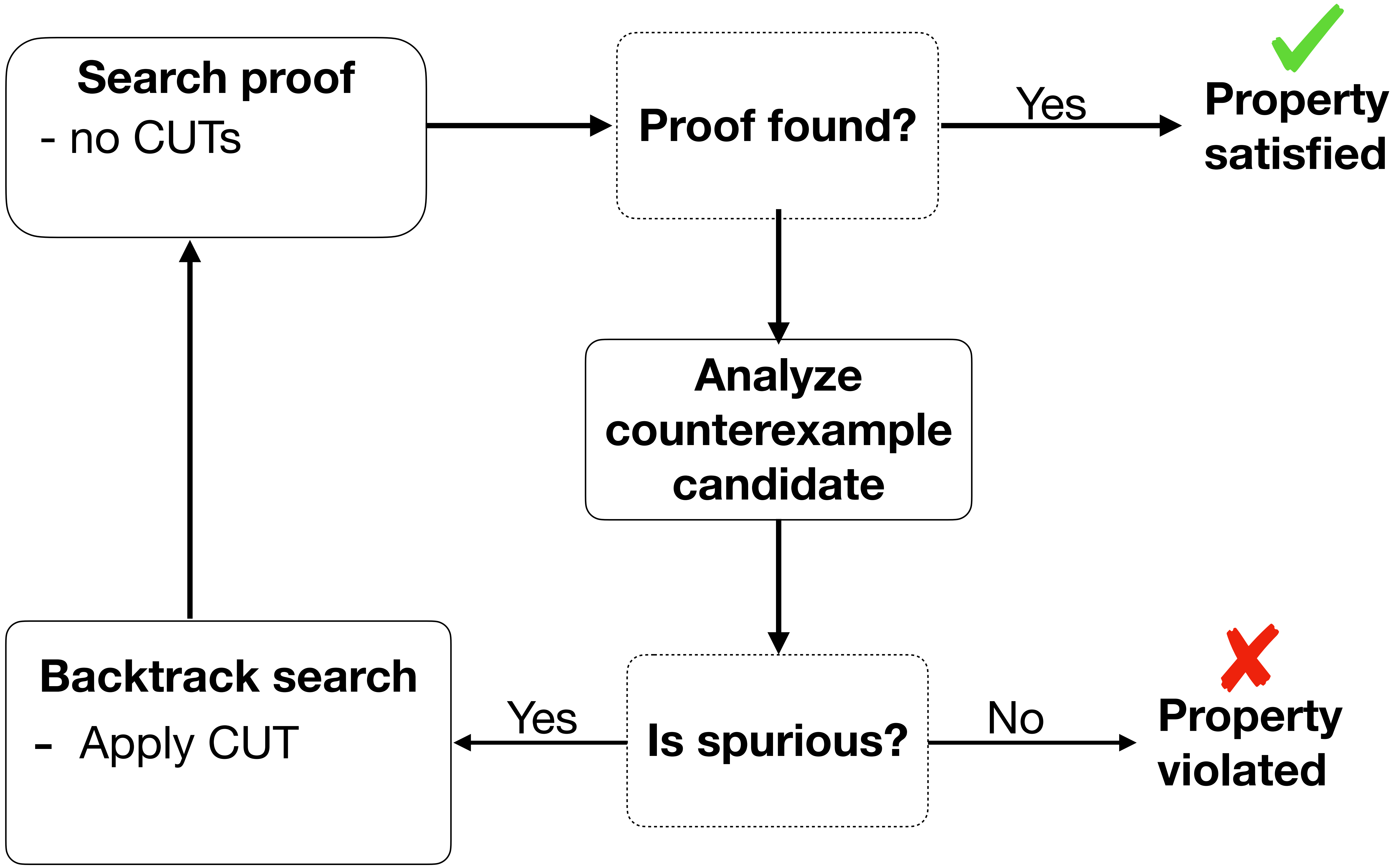
CEGAR Proof Search



CEGAR Proof Search

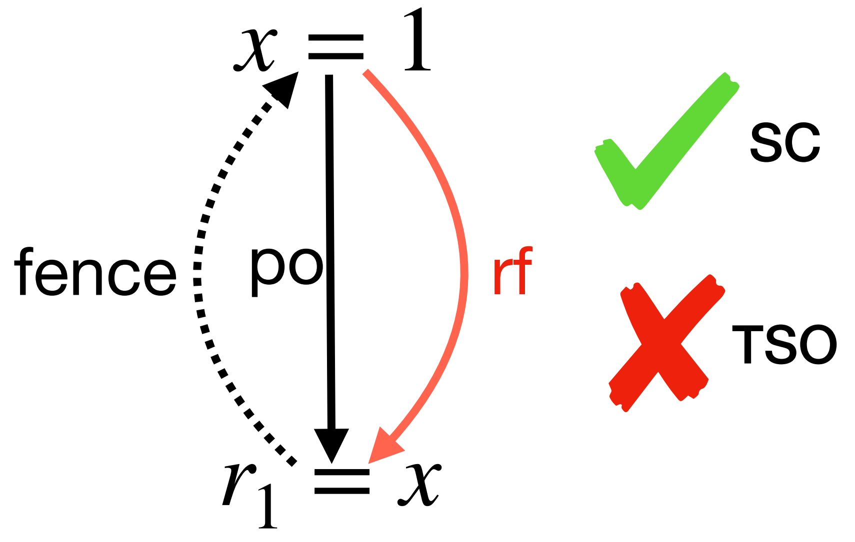


CEGAR Proof Search

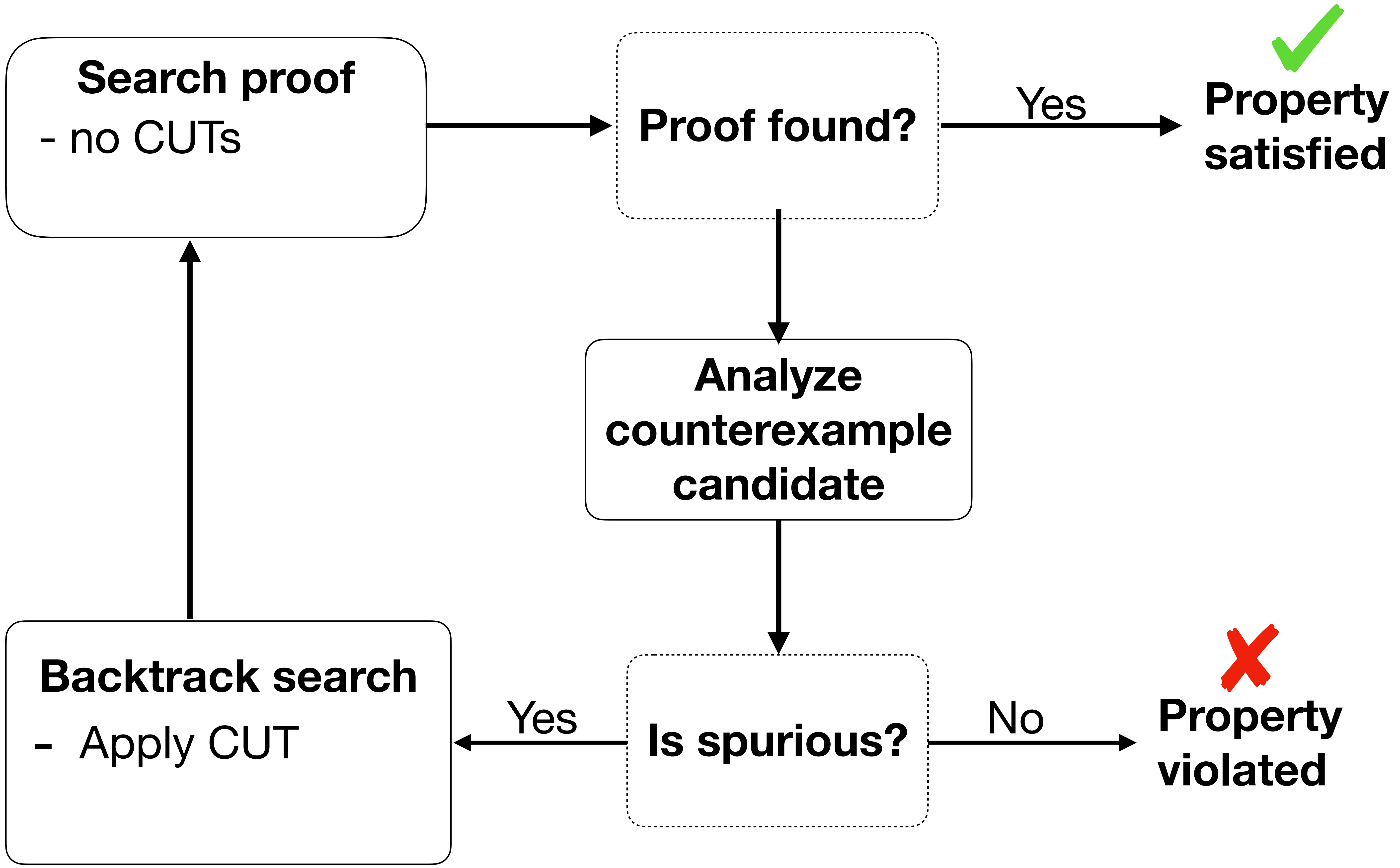


Problem: checking inclusions in relational algebra is not sufficient

$\mathsf{hb}^+_{TSO} \cap \mathsf{id} \subseteq \mathsf{T};(\mathsf{hb}^+_{SC} \cap \mathsf{id});\mathsf{T}$

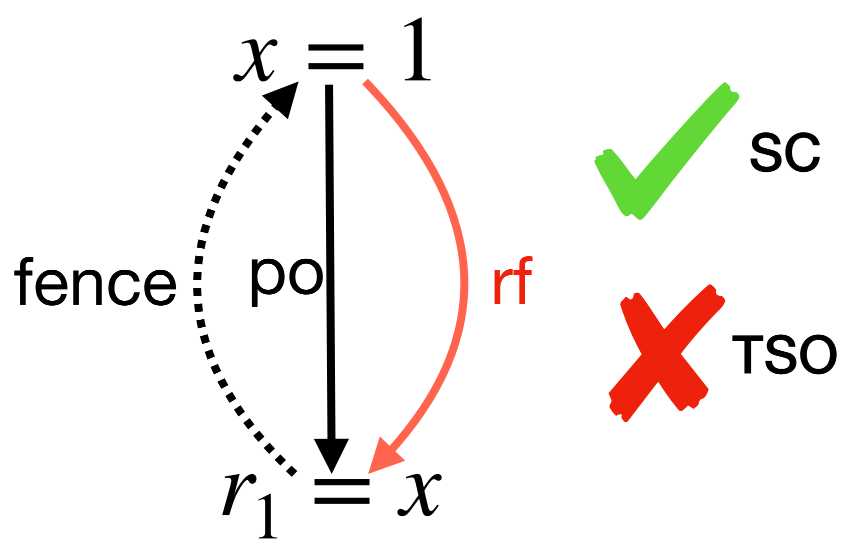


CEGAR Proof Search



Problem: checking inclusions in relational algebra is not sufficient

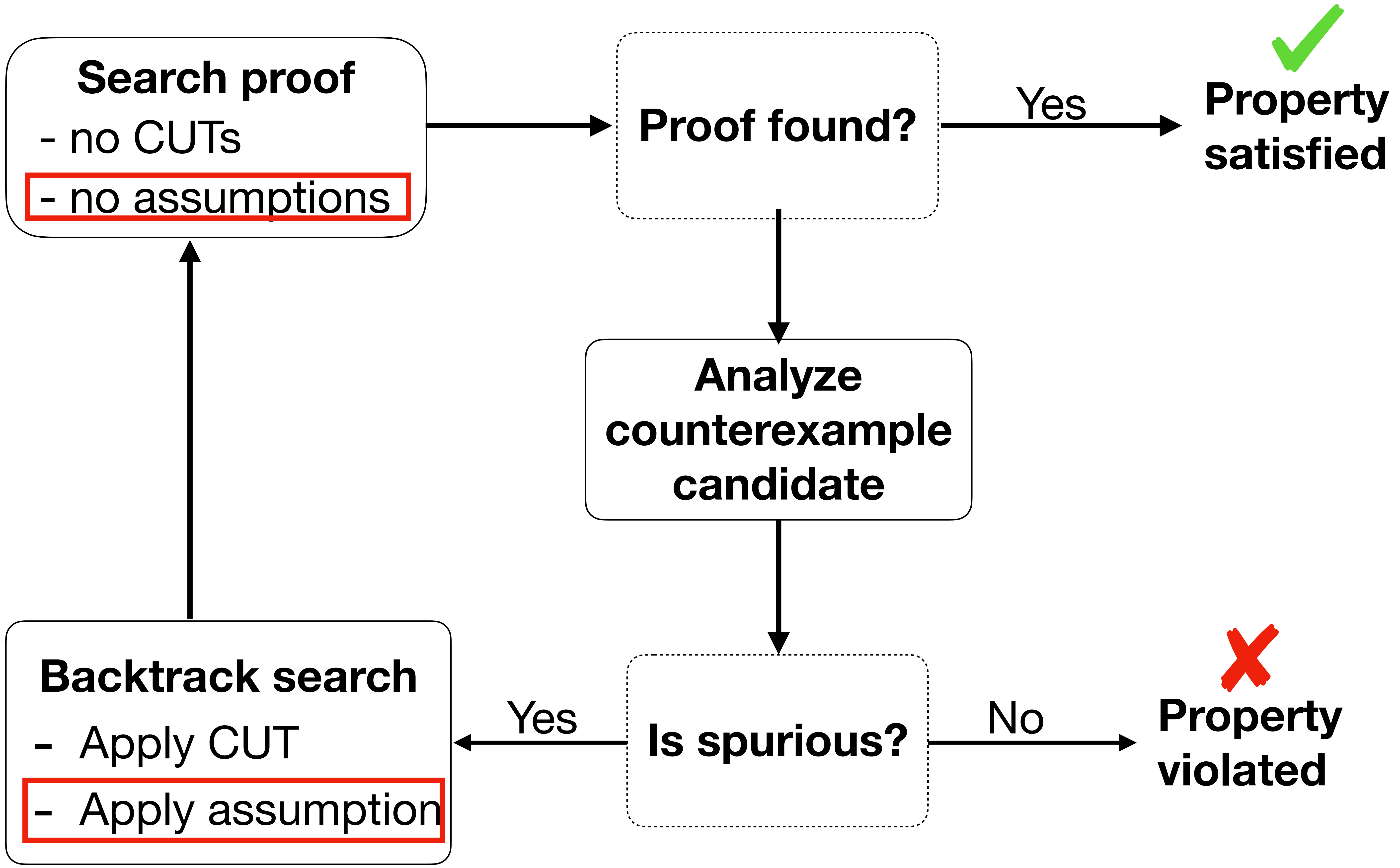
$$hb^{+}_{TSO} \cap id \subseteq \top; (hb^{+}_{SC} \cap id); \top$$



Solution: use assumptions to restrict graphs to executions

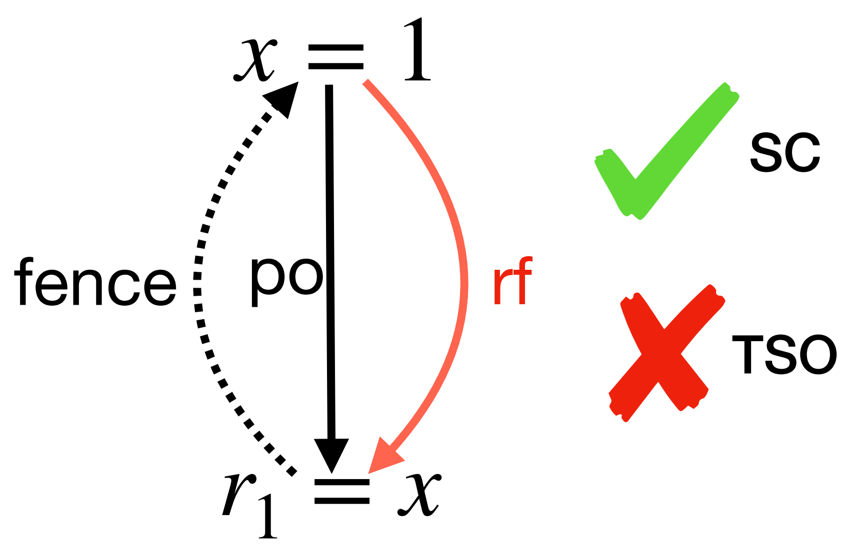
$$fence \subseteq po$$
$$po^{+} \subseteq po$$
$$rf; rf^{-1} \subseteq id$$

CEGAR Proof Search



Problem: checking inclusions in relational algebra is not sufficient

$\mathsf{hb}^+_{TSO} \cap \mathsf{id} \subseteq \mathsf{T}; (\mathsf{hb}^+_{SC} \cap \mathsf{id}); \mathsf{T}$



Solution: use assumptions to restrict graphs to executions

$\mathsf{fence} \subseteq \mathsf{po}$

$\mathsf{po}^+ \subseteq \mathsf{po}$

$\mathsf{rf}; \mathsf{rf}^{-1} \subseteq \mathsf{id}$

Evaluation

Comparison with KATER

Table 1: TOOL vs. KATER.

	TOOL		KATER	
Benchmark	T(s)	Res.	T(s)	Res.
COH1↔COH2	2.36	✓	—	ERR
ECO1↔ECO2	0.01	✓	0.00	✓
RA1↔RA2	1.72	✓	0.00	✓
RA1↔RA3	—	ERR	0.00	✓
RC11↔RC12	43.56	✓	0.12	✓
SC↔SCFM	6.04	✓	0.02	✓
TSO↔TSOFM	TO	—	0.06	✓
IMM→ARM8	0.02	✓	0.09	✓
IMM→TSO	TO	—	0.01	✓
PPC→PPC-S	TO	—	1.13	✓
RC11→ARM8	1.20	✓	0.11	✓
RC11→IMM	4.36	✓	0.01	✓
RC11→PPC-W	TO	—	1.00	✓
RC11F→PPC-SF	14.39	✓	5.27	✓
RC11RW→PPC-SF	TO	—	6.36	✓
RC11S→ARM8	3.06	✓	0.12	✓
C11→PPC-S	TO	—	1.60	✖
C11→PPC-W	8.78	✖	0.66	✖

Evaluation

Comparison with KATER

Table 1: TOOL vs. KATER.

	TOOL		KATER	
Benchmark	T(s)	Res.	T(s)	Res.
COH1↔COH2	2.36	✓	—	ERR
ECO1↔ECO2	0.01	✓	0.00	✓
RA1↔RA2	1.72	✓	0.00	✓
RA1↔RA3	—	ERR	0.00	✓
RC11↔RC12	43.56	✓	0.12	✓
SC↔SCFM	6.04	✓	0.02	✓
TSO↔TSOFM	TO	—	0.06	✓
IMM→ARM8	0.02	✓	0.09	✓
IMM→TSO	TO	—	0.01	✓
PPC→PPC-S	TO	—	1.13	✓
RC11→ARM8	1.20	✓	0.11	✓
RC11→IMM	4.36	✓	0.01	✓
RC11→PPC-W	TO	—	1.00	✓
RC11F→PPC-SF	14.39	✓	5.27	✓
RC11RW→PPC-SF	TO	—	6.36	✓
RC11S→ARM8	3.06	✓	0.12	✓
C11→PPC-S	TO	—	1.60	✖
C11→PPC-W	8.78	✖	0.66	✖

- Kater outperforms our tool

Evaluation

Comparison with KATER

Table 1: Tool vs. Kater.

	Tool		Kater	
Benchmark	T(s)	Res.	T(s)	Res.
coh1↔coh2	2.36	✓	—	ERR
eco1↔eco2	0.01	✓	0.00	✓
ra1↔ra2	1.72	✓	0.00	✓
ra1↔ra3	—	ERR	0.00	✓
rc11↔rc12	43.56	✓	0.12	✓
sc↔scfm	6.04	✓	0.02	✓
tso↔tsofm	TO	—	0.06	✓
imm→arm8	0.02	✓	0.09	✓
imm→tso	TO	—	0.01	✓
ppc→ppc-s	TO	—	1.13	✓
rc11→arm8	1.20	✓	0.11	✓
rc11→imm	4.36	✓	0.01	✓
rc11→ppc-w	TO	—	1.00	✓
rc11f→ppc-sf	14.39	✓	5.27	✓
rc11rw→ppc-sf	TO	—	6.36	✓
rc11s→arm8	3.06	✓	0.12	✓
c11→ppc-s	TO	—	1.60	✖
c11→ppc-w	8.78	✖	0.66	✖

- Kater outperforms our tool
- Our tool supports complex CAT features (intersections, converses)

Evaluation

Comparison with KATER

Table 1: TOOL vs. KATER.

	TOOL		KATER	
Benchmark	T(s)	Res.	T(s)	Res.
COH1↔COH2	2.36	✓	—	ERR
ECO1↔ECO2	0.01	✓	0.00	✓
RA1↔RA2	1.72	✓	0.00	✓
RA1↔RA3	—	ERR	0.00	✓
RC11↔RC12	43.56	✓	0.12	✓
SC↔SCFM	6.04	✓	0.02	✓
TSO↔TSOFM	TO	—	0.06	✓
IMM→ARM8	0.02	✓	0.09	✓
IMM→TSO	TO	—	0.01	✓
PPC→PPC-S	TO	—	1.13	✓
RC11→ARM8	1.20	✓	0.11	✓
RC11→IMM	4.36	✓	0.01	✓
RC11→PPC-W	TO	—	1.00	✓
RC11F→PPC-SF	14.39	✓	5.27	✓
RC11RW→PPC-SF	TO	—	6.36	✓
RC11S→ARM8	3.06	✓	0.12	✓
C11→PPC-S	TO	—	1.60	✖
C11→PPC-W	8.78	✖	0.66	✖

- Kater outperforms our tool
- Our tool supports complex CAT features (intersections, converses)

Successful applications

Table 2: MCA, OOTA and UNIPROC.

Benchmark	T(s)	Res.
UNIPROC	0.20	✓
ARM-MCA	13.21	✓
ARM-NO-OOTA	15.84	✓
IMM-MCA	0.21	✖
IMM-NO-OOTA	28.80	✓
TSO-MCA	0.11	✓
TSO-NO-OOTA	0.38	✓

Table 3: LKMM tests.

Benchmark	T(s)	Res.
NO-OOTA-SEM	0.04	✓
NO-OOTA-SYN	1.26	✖
MCA	7.75	✖
V00-ONCE2ACQ	0.07	✓
V00-ONCE2REL	0.01	✓
V00-ACQREL2MB	0.21	✖
V04-ONCE2ACQ	0.26	✓
V04-ONCE2MB	0.21	✓
V04-ONCE2REL	0.20	✓
V04-ACQREL2MB	5.57	✓
$v00 \subseteq v01$	21.54	✖
$v01 \subseteq v00$	536.36	✓
$v01 \subseteq v02$	0.05	✖
$v02 \subseteq v01$	0.05	✓
$v02 \subseteq v03$	0.14	✓
$v03 \subseteq v02$	0.07	✖(?)
PPO $\subseteq$ PO (v02)	0.05	✖
PPO $\subseteq$ PO (v03)	0.05	✓

Evaluation

Comparison with KATER

Table 1: TOOL vs. KATER.

Benchmark	TOOL T(s)	TOOL Res.	KATER T(s)	KATER Res.
COH1↔COH2	2.36	✓	—	ERR
ECO1↔ECO2	0.01	✓	0.00	✓
RA1↔RA2	1.72	✓	0.00	✓
RA1↔RA3	—	ERR	0.00	✓
RC11↔RC12	43.56	✓	0.12	✓
SC↔SCFM	6.04	✓	0.02	✓
TSO↔TSOFM	TO	—	0.06	✓
IMM→ARM8	0.02	✓	0.09	✓
IMM→TSO	TO	—	0.01	✓
PPC→PPC-S	TO	—	1.13	✓
RC11→ARM8	1.20	✓	0.11	✓
RC11→IMM	4.36	✓	0.01	✓
RC11→PPC-W	TO	—	1.00	✓
RC11F→PPC-SF	14.39	✓	5.27	✓
RC11RW→PPC-SF	TO	—	6.36	✓
RC11S→ARM8	3.06	✓	0.12	✓
C11→PPC-S	TO	—	1.60	✖
C11→PPC-W	8.78	✖	0.66	✖

- Kater outperforms our tool
- Our tool supports complex CAT features (intersections, converses)

Successful applications

Table 2: MCA, OOTA and UNIPROC.

Benchmark	T(s)	Res.
UNIPROC	0.20	✓
ARM-MCA	13.21	✓
ARM-NO-OOTA	15.84	✓
IMM-MCA	0.21	✖
IMM-NO-OOTA	28.80	✓
TSO-MCA	0.11	✓
TSO-NO-OOTA	0.38	✓

Table 3: LKMM tests.

Benchmark	T(s)	Res.
NO-OOTA-SEM	0.04	✓
NO-OOTA-SYN	1.26	✖
MCA	7.75	✖
v00-ONCE2ACQ	0.07	✓
v00-ONCE2REL	0.01	✓
v00-ACQREL2MB	0.21	✖
v04-ONCE2ACQ	0.26	✓
v04-ONCE2MB	0.21	✓
v04-ONCE2REL	0.20	✓
v04-ACQREL2MB	5.57	✓
v00 ⊆ v01	21.54	✖
v01 ⊆ v00	536.36	✓
v01 ⊆ v02	0.05	✖
v02 ⊆ v01	0.05	✓
v02 ⊆ v03	0.14	✓
v03 ⊆ v02	0.07	✖(?)
PPO ⊆ PO (v02)	0.05	✖
PPO ⊆ PO (v03)	0.05	✓

- Analyzed MCA and OOTA for different MM

Evaluation

Comparison with KATER

Table 1: Tool vs. Kater.

Benchmark	Tool T(s)	Tool Res.	Kater T(s)	Kater Res.
coh1↔coh2	2.36	✓	—	ERR
eco1↔eco2	0.01	✓	0.00	✓
ra1↔ra2	1.72	✓	0.00	✓
ra1↔ra3	—	ERR	0.00	✓
rc11↔rc12	43.56	✓	0.12	✓
sc↔scfm	6.04	✓	0.02	✓
tso↔tsofm	TO	—	0.06	✓
imm→arm8	0.02	✓	0.09	✓
imm→tso	TO	—	0.01	✓
ppc→ppc-s	TO	—	1.13	✓
rc11→arm8	1.20	✓	0.11	✓
rc11→imm	4.36	✓	0.01	✓
rc11→ppc-w	TO	—	1.00	✓
rc11f→ppc-sf	14.39	✓	5.27	✓
rc11rw→ppc-sf	TO	—	6.36	✓
rc11s→arm8	3.06	✓	0.12	✓
c11→ppc-s	TO	—	1.60	✖
c11→ppc-w	8.78	✖	0.66	✖

- Kater outperforms our tool
- Our tool supports complex CAT features (intersections, converses)

Successful applications

Table 2: MCA, OOTA and UNIPROC.

Benchmark	T(s)	Res.
uniproc	0.20	✓
arm-mca	13.21	✓
arm-no-oota	15.84	✓
imm-mca	0.21	✖
imm-no-oota	28.80	✓
tso-mca	0.11	✓
tso-no-oota	0.38	✓

Table 3: LKMM tests.

Benchmark	T(s)	Res.
no-oota-sem	0.04	✓
no-oota-syn	1.26	✖
mca	7.75	✖
v00-once2acq	0.07	✓
v00-once2rel	0.01	✓
v00-acqrel2mb	0.21	✖
v04-once2acq	0.26	✓
v04-once2mb	0.21	✓
v04-once2rel	0.20	✓
v04-acqrel2mb	5.57	✓
v00 ⊆ v01	21.54	✖
v01 ⊆ v00	536.36	✓
v01 ⊆ v02	0.05	✖
v02 ⊆ v01	0.05	✓
v02 ⊆ v03	0.14	✓
v03 ⊆ v02	0.07	✖(?)
ppo ⊆ po (v02)	0.05	✖
ppo ⊆ po (v03)	0.05	✓

- Analyzed MCA and OOTA for different MM
- Identification of (known) LKMM bugs

Evaluation

Comparison with KATER

Table 1: Tool vs. Kater.

	Tool		Kater	
Benchmark	T(s)	Res.	T(s)	Res.
coh1↔coh2	2.36	✓	—	ERR
eco1↔eco2	0.01	✓	0.00	✓
ra1↔ra2	1.72	✓	0.00	✓
ra1↔ra3	—	ERR	0.00	✓
rc11↔rc12	43.56	✓	0.12	✓
sc↔scfm	6.04	✓	0.02	✓
tso↔tsofm	TO	—	0.06	✓
imm→arm8	0.02	✓	0.09	✓
imm→tso	TO	—	0.01	✓
ppc→ppc-s	TO	—	1.13	✓
rc11→arm8	1.20	✓	0.11	✓
rc11→imm	4.36	✓	0.01	✓
rc11→ppc-w	TO	—	1.00	✓
rc11f→ppc-sf	14.39	✓	5.27	✓
rc11rw→ppc-sf	TO	—	6.36	✓
rc11s→arm8	3.06	✓	0.12	✓
c11→ppc-s	TO	—	1.60	✖
c11→ppc-w	8.78	✖	0.66	✖

- Kater outperforms our tool
- Our tool supports complex CAT features (intersections, converses)

Successful applications

Table 2: MCA, OOTA and UNIPROC.

Benchmark	T(s)	Res.
uniproc	0.20	✓
arm-mca	13.21	✓
arm-no-oota	15.84	✓
imm-mca	0.21	✖
imm-no-oota	28.80	✓
tso-mca	0.11	✓
tso-no-oota	0.38	✓

Table 3: LKMM tests.

Benchmark	T(s)	Res.
no-oota-sem	0.04	✓
no-oota-syn	1.26	✖
mca	7.75	✖
v00-once2acq	0.07	✓
v00-once2rel	0.01	✓
v00-acqrel2mb	0.21	✖
v04-once2acq	0.26	✓
v04-once2mb	0.21	✓
v04-once2rel	0.20	✓
v04-acqrel2mb	5.57	✓
v00 ⊆ v01	21.54	✖
v01 ⊆ v00	536.36	✓
v01 ⊆ v02	0.05	✖
v02 ⊆ v01	0.05	✓
v02 ⊆ v03	0.14	✓
v03 ⊆ v02	0.07	✖(?)
ppo ⊆ po (v02)	0.05	✖
ppo ⊆ po (v03)	0.05	✓

- Analyzed MCA and OOTA for different MM
- Identification of (known) LKMM bugs
- Generation of useful counterexamples

Conclusion

Conclusion

- Memory Models need verification

Conclusion

- Memory Models need verification
- This can be achieved by checking inclusions in relational algebra

Conclusion

- Memory Models need verification
- This can be achieved by checking inclusions in relational algebra
- We provided a sound & complete proof system for relational algebra inclusions

Conclusion

- Memory Models need verification
- This can be achieved by checking inclusions in relational algebra
- We provided a sound & complete proof system for relational algebra inclusions
- We presented a CEGAR approach for an efficient proof search

Recent Decidability Results in Verification

Recent Decidability Results in Verification

Hard Problems in Verification

Complexity of VASS Reachability

Decidability of Regular Separability for VASS Reachability Languages

Decidability of PVASS Reachability

Decidability of BVASS Reachability

Decidability of DataVASS Reachability

Complexity of Parity Games

Hard Problems in Verification

~~Complexity of VASS Reachability~~ [solved by Czerwinski, Leroux, and Schmitz in 2019 (upper bound) and 2021 (lower bound)]
[LICS’19, FOCS’21 2x]

Decidability of Regular Separability for VASS

Decidability of PVASS Reachability

Decidability of BVASS Reachability

Decidability of DataVASS Reachability

Complexity of Parity Games

Hard Problems in Verification

~~Complexity of VASS Reachability~~ [solved by Czerwinski, Leroux, and Schmitz in 2019 (upper bound) and 2021 (lower bound)] [LICS'19, FOCS'21 2x]

~~Decidability of Regular Separability for VASS~~ [solved by us, with E. Keskin, LICS'24]

Decidability of PVASS Reachability

Decidability of BVASS Reachability

Decidability of DataVASS Reachability

Complexity of Parity Games

Hard Problems in Verification

~~Complexity of VASS Reachability~~ [solved by Czerwinski, Leroux, and Schmitz in 2019 (upper bound) and 2021 (lower bound)] [LICS'19, FOCS'21 2x]

~~Decidability of Regular Separability for VASS~~ [solved by us, with E. Keskin, LICS'24]

~~Decidability of PVASS Reachability~~ [solved by us, with E. Keskin and R. Guttenberg, under submission]

Decidability of BVASS Reachability

Decidability of DataVASS Reachability

Complexity of Parity Games

Hard Problems in Verification

~~Complexity of VASS Reachability~~ [solved by Czerwinski, Leroux, and Schmitz in 2019 (upper bound) and 2021 (lower bound)] [LICS'19, FOCS'21 2x]

~~Decidability of Regular Separability for VASS~~ [solved by us, with E. Keskin, LICS'24]

~~Decidability of PVASS Reachability~~ [solved by us, with E. Keskin and R. Guttenberg, under submission]

~~Decidability of BVA~~SS Reachability [working on it, with J. Grünke]

Decidability of DataVASS Reachability

Complexity of Parity Games

Hard Problems in Verification

~~Complexity of VASS Reachability~~ [solved by Czerwinski, Leroux, and Schmitz in 2019 (upper bound) and 2021 (lower bound)] [LICS'19, FOCS'21 2x]

~~Decidability of Regular Separability for VASS~~ [solved by us, with E. Keskin, LICS'24]

~~Decidability of PVASS Reachability~~ [solved by us, with E. Keskin and R. Guttenberg, under submission]

~~Decidability of BVA~~SS Reachability [working on it, with J. Grünke]

Decidability of DataVASS Reachability

~~Comp~~lexity of Parity Games [working on it, with E. Keskin]

Hard Problems in Verification

~~Complexity of VASS Reachability~~ [solved by Czerwinski, Leroux, and Schmitz in 2019 (upper bound) and 2021 (lower bound)] [LICS'19, FOCS'21 2x]

~~Decidability of Regular Separability for VASS~~ [solved by us, with E. Keskin, LICS'24]

~~Decidability of PVASS Reachability~~ [solved by us, with E. Keskin and R. Guttenberg, under submission]

~~Decidability of BVA~~SS Reachability [working on it, with J. Grünke]

Decidability of DataVASS Reachability

~~Comp~~lexity of Parity Games [working on it, with E. Keskin]

Regular Separability of VASS Reachability Languages

Eren Keskin, Roland Meyer: On the separability problem of VASS reachability languages @ LICS24

Regular Separability

Regular Separability

$\mathbb{X} \in \{\mathbb{Z}, \mathbb{N}\}$.

Regular Separability

$\mathbb{X} \in \{\mathbb{Z}, \mathbb{N}\}$.

$\mathbb{X}$-REGSEP:

Given: Initialized VASS V_1 and V_2 over Σ .

Question: Does $L_{\mathbb{X}}(V_1) \mid L_{\mathbb{X}}(V_2)$ hold?

Regular Separability

$\mathbb{X} \in \{\mathbb{Z}, \mathbb{N}\}$.

Reachability languages.

$\mathbb{X}$-REGSEP:

Given: Initialized VASS V_1 and V_2 over Σ .

Question: Does $L_{\mathbb{X}}(V_1) \mid L_{\mathbb{X}}(V_2)$ hold?

Regular Separability

Reachability languages.

$\mathbb{X} \in \{\mathbb{Z}, \mathbb{N}\}$.

$\mathbb{X}$-REGSEP:

Given: Initialized VASS V_1 and V_2 over Σ .

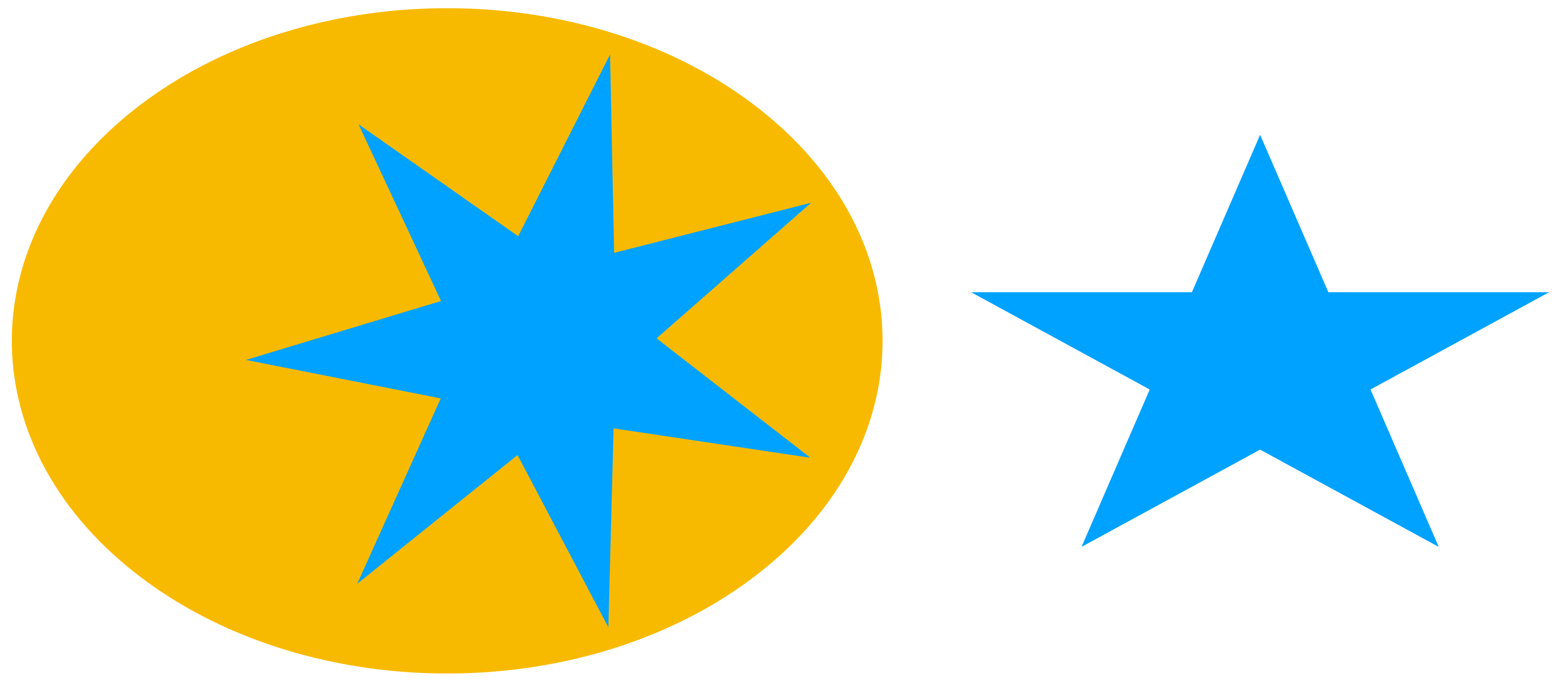
Question: Does $L_{\mathbb{X}}(V_1) \mid L_{\mathbb{X}}(V_2)$ hold?

$L_1 \mid L_2$:

$\exists R \subseteq \Sigma^*$ regular. $L_1 \subseteq R \wedge R \cap L_2 = \emptyset$.

Regular Separability

Reachability languages.



$\mathbb{X} \in \{\mathbb{Z}, \mathbb{N}\}$.

$\mathbb{X}$-REGSEP:

Given: Initialized VASS V_1 and V_2 over Σ .

Question: Does $L_{\mathbb{X}}(V_1) \mid L_{\mathbb{X}}(V_2)$ hold?

$L_1 \mid L_2$:

$\exists R \subseteq \Sigma^*$ regular. $L_1 \subseteq R \wedge R \cap L_2 = \emptyset$.

Regular Separability

Reachability languages.

$\mathbb{X} \in \{\mathbb{Z}, \mathbb{N}\}$.

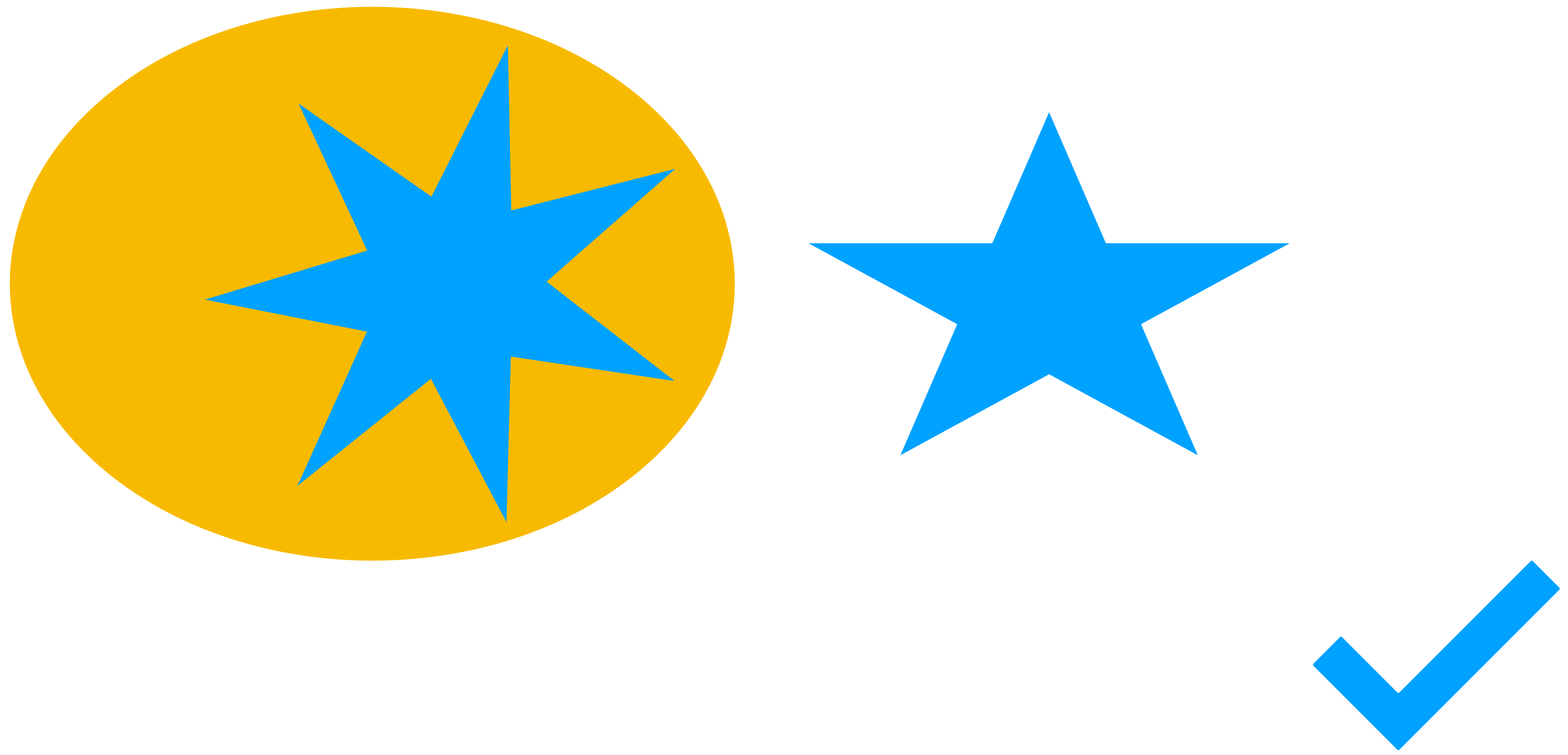
$\mathbb{X}$-REGSEP:

Given: Initialized VASS V_1 and V_2 over Σ .

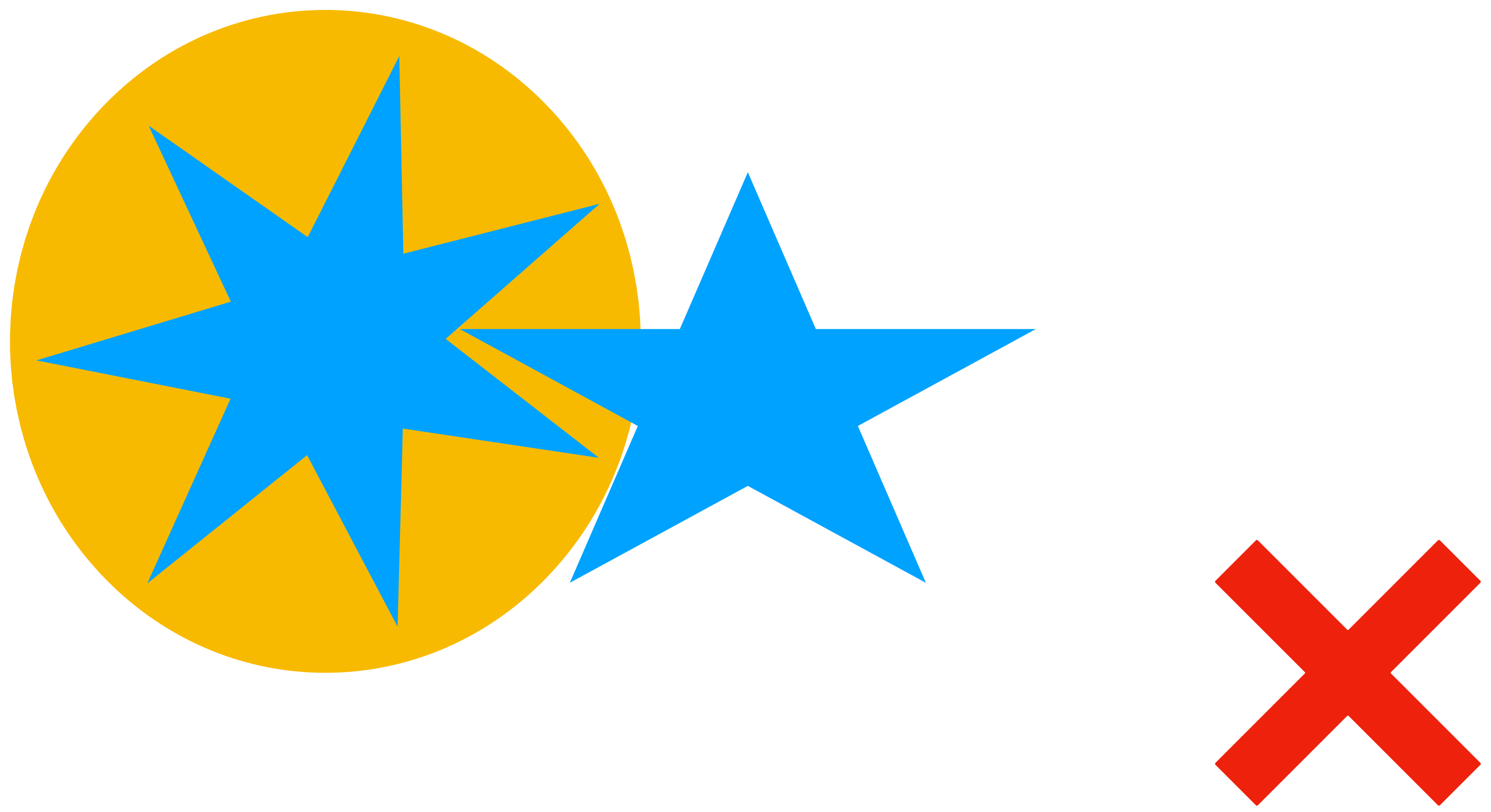
Question: Does $L_{\mathbb{X}}(V_1) \mid L_{\mathbb{X}}(V_2)$ hold?

$L_1 \mid L_2$:

$\exists R \subseteq \Sigma^*$ regular. $L_1 \subseteq R \wedge R \cap L_2 = \emptyset$.



vs.



Regular Separability

Regular Separability

Example:

1. $\{a^n . b^n \mid n \in \mathbb{N}\} \mid \{a^n . b^{n+1} \mid n \in \mathbb{N}\}$.

Regular Separability

Example:

1. $\{a^n . b^n \mid n \in \mathbb{N}\} \mid \{a^n . b^{n+1} \mid n \in \mathbb{N}\}$.

Yes! Separator: Even.Even ∪ Odd.Odd.

Regular Separability

Example:

1. $\{a^n . b^n \mid n \in \mathbb{N}\} \mid \{a^n . b^{n+1} \mid n \in \mathbb{N}\}$.

Yes! Separator: Even.Even ∪ Odd.Odd.

Regular Separability

Example:

1. $\{a^n . b^n \mid n \in \mathbb{N}\} \mid \{a^n . b^{n+1} \mid n \in \mathbb{N}\}$.

Yes! Separator: Even.Even ∪ Odd.Odd.

2. $\{a^n . b^{\leq n} \mid n \in \mathbb{N}\} \nmid \{a^n . b^{>n} \mid n \in \mathbb{N}\}$.

Regular Separability

Example:

1. $\{a^n . b^n \mid n \in \mathbb{N}\} \mid \{a^n . b^{n+1} \mid n \in \mathbb{N}\}$. ✓

Yes! Separator: Even.Even ∪ Odd.Odd.

2. $\{a^n . b^{\leq n} \mid n \in \mathbb{N}\} \nmid \{a^n . b^{>n} \mid n \in \mathbb{N}\}$.

No! Assume $A : L_1 \mid L_2$ and A has m states.
Consider $a^{m+1} . b^{m+1} \in L_1 \subseteq L(A)$.

Regular Separability

Example:

1. $\{a^n . b^n \mid n \in \mathbb{N}\} \mid \{a^n . b^{n+1} \mid n \in \mathbb{N}\}$.

Yes! Separator: Even.Even ∪ Odd.Odd.

2. $\{a^n . b^{\leq n} \mid n \in \mathbb{N}\} \nmid \{a^n . b^{>n} \mid n \in \mathbb{N}\}$.

No! Assume $A : L_1 \mid L_2$ and A has m states.
Consider $a^{m+1} . b^{m+1} \in L_1 \subseteq L(A)$. ↯

Regular Separability

Example:

1. $\{a^n \,.\, b^n \mid n \in \mathbb{N}\} \mid \{a^n \,.\, b^{n+1} \mid n \in \mathbb{N}\}$.

Yes! Separator: Even.Even ∪ Odd.Odd.

2. $\{a^n \,.\, b^{\leq n} \mid n \in \mathbb{N}\} \nmid \{a^n \,.\, b^{>n} \mid n \in \mathbb{N}\}$.

No! Assume $A : L_1 \mid L_2$ and A has m states.
Consider $a^{m+1} \,.\, b^{m+1} \in L_1 \subseteq L(A)$. $\lightning$

Regular Separability

Example:

1. $\{a^n . b^n \mid n \in \mathbb{N}\} \mid \{a^n . b^{n+1} \mid n \in \mathbb{N}\}$. ✓

Yes! Separator: Even.Even ∪ Odd.Odd.

2. $\{a^n . b^{\leq n} \mid n \in \mathbb{N}\} \nmid \{a^n . b^{>n} \mid n \in \mathbb{N}\}$. ✗

No! Assume $A : L_1 \mid L_2$ and A has m states.
Consider $a^{m+1} . b^{m+1} \in L_1 \subseteq L(A)$. ↯

Discussion:
Separability tries to understand the gap between languages.

Regular Separability

Example:

1. $\{a^n . b^n \mid n \in \mathbb{N}\} \mid \{a^n . b^{n+1} \mid n \in \mathbb{N}\}$.

Yes! Separator: Even.Even ∪ Odd.Odd.

2. $\{a^n . b^{\leq n} \mid n \in \mathbb{N}\} \nmid \{a^n . b^{>n} \mid n \in \mathbb{N}\}$.

No! Assume $A : L_1 \mid L_2$ and A has m states.
Consider $a^{m+1} . b^{m+1} \in L_1 \subseteq L(A)$. ↯

Discussion:
Separability tries to understand the gap between languages.

Insight:
Modulo seems to play an important role!

Regular Separability

Regular Separability

Theorem [Lorenzo, Wojtek, Slawek, Charles, ICALP'17]:
$\mathbb{Z}$-REGSEP is decidable.

Regular Separability

Theorem [Lorenzo, Wojtek, Slawek, Charles, ICALP'17]:
$\mathbb{Z}$-REGSEP is decidable.

Regular Separability

Theorem [Lorenzo, Wojtek, Slawek, Charles, ICALP'17]:
$\mathbb{Z}$-REGSEP is decidable.

Theorem [LICS'24]:
$\mathbb{N}$-REGSEP is decidable.

VASS Reachability

Deciding Reachability

Deciding Reachability

Approximations:

Deciding Reachability

Approximations:

Coverability graphs:
Good: Can keep counters non-negative.
Bad: Cannot guarantee precise counter values.

Deciding Reachability

Approximations:

Coverability graphs:
Good: Can keep counters non-negative.
Bad: Cannot guarantee precise counter values.

Marking Equation:
Good: Can guarantee precise counter values.
Bad: Cannot keep counters non-negative.

Deciding Reachability

Approximations:

Coverability graphs:
Good: Can keep counters non-negative.
Bad: Cannot guarantee precise counter values.

Marking Equation:
Good: Can guarantee precise counter values.
Bad: Cannot keep counters non-negative.

Solution:
Combine the two.

Deciding Reachability

Deciding Reachability

Challenge:
Coverability graphs need pumping to guarantee non-negativity.
Pumping has to respect the marking equation.

Deciding Reachability

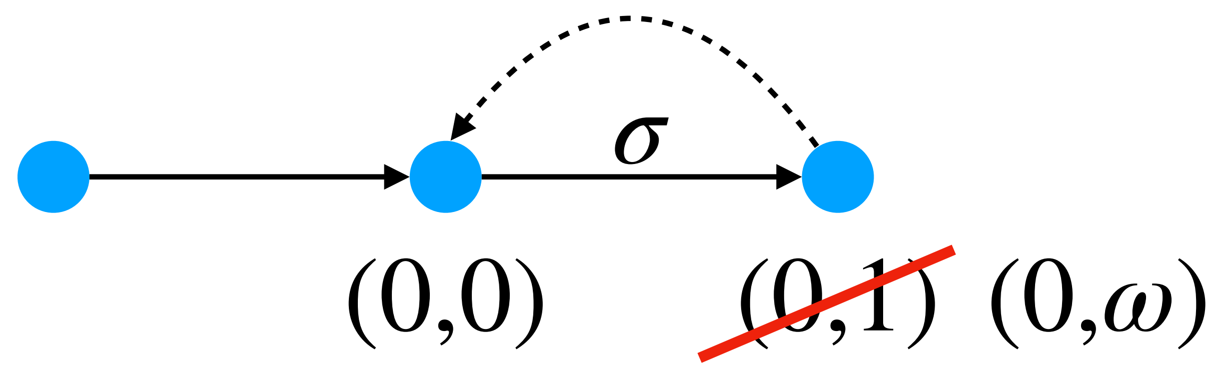
Challenge:
Coverability graphs need pumping to guarantee non-negativity.
Pumping has to respect the marking equation.

Solution:
Only pump where the solution space is unbounded.

Deciding Reachability

Challenge:
Coverability graphs need pumping to guarantee non-negativity.
Pumping has to respect the marking equation.

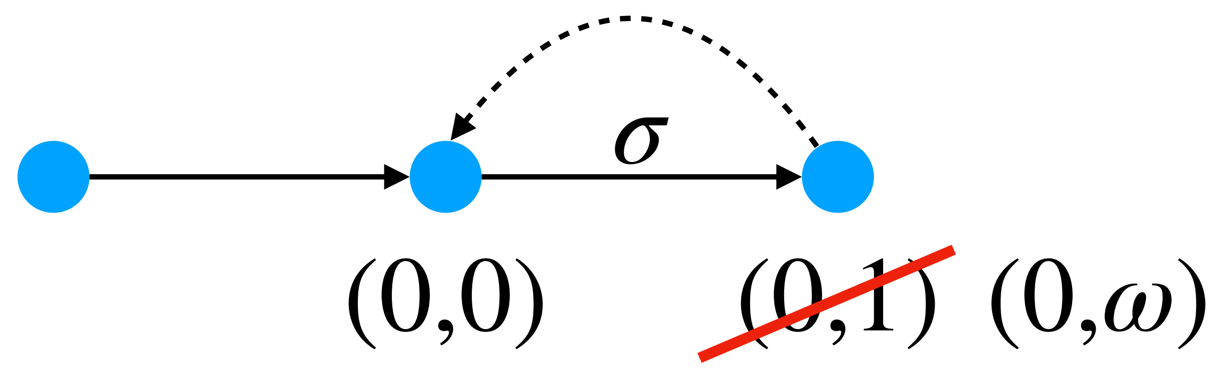
Solution:
Only pump where the solution space is unbounded.



Deciding Reachability

Challenge:
Coverability graphs need pumping to guarantee non-negativity.
Pumping has to respect the marking equation.

Solution:
Only pump where the solution space is unbounded.



$\Rightarrow$

$x[e]$ with $e \in \sigma$
$x[j]$ with j = 2

have to be unbounded in the solution space.

Deciding Reachability

Deciding Reachability

Lemma:

Consider $A \cdot x = b$ over $\mathbb{N}^k$ and variable x[i].

Deciding Reachability

Lemma:

Consider $A \cdot x = b$ over $\mathbb{N}^k$ and variable x[i].

x[i] is unbounded in $sol(A \cdot x = b)$

$$\Leftrightarrow \quad \exists s \in sol(A \cdot x = 0)\,.\ s(x[i]) > 0.$$

Deciding Reachability

Lemma:

Consider $A \cdot x = b$ over $\mathbb{N}^k$ and variable x[i].

x[i] is unbounded in $sol(A \cdot x = b)$

$$\Leftrightarrow \quad \exists s \in sol(A \cdot x = 0)\,.\ s(x[i]) > 0.$$

Support = the set of unbounded variables.

Deciding Reachability

Lemma:

Consider $A \cdot x = b$ over $\mathbb{N}^k$ and variable x[i].

x[i] is unbounded in $sol(A \cdot x = b)$

$$\Leftrightarrow \quad \exists s \in sol(A \cdot x = 0)\,.\; s(x[i]) > 0.$$

Support = the set of unbounded variables.

Support solution =

$s \in sol(A \cdot x = 0)$ giving a positive value to all variables in the support.

Deciding Reachability

Lemma:
Consider $A \cdot x = b$ over $\mathbb{N}^k$ and variable x[i].

x[i] is unbounded in $sol(A \cdot x = b)$

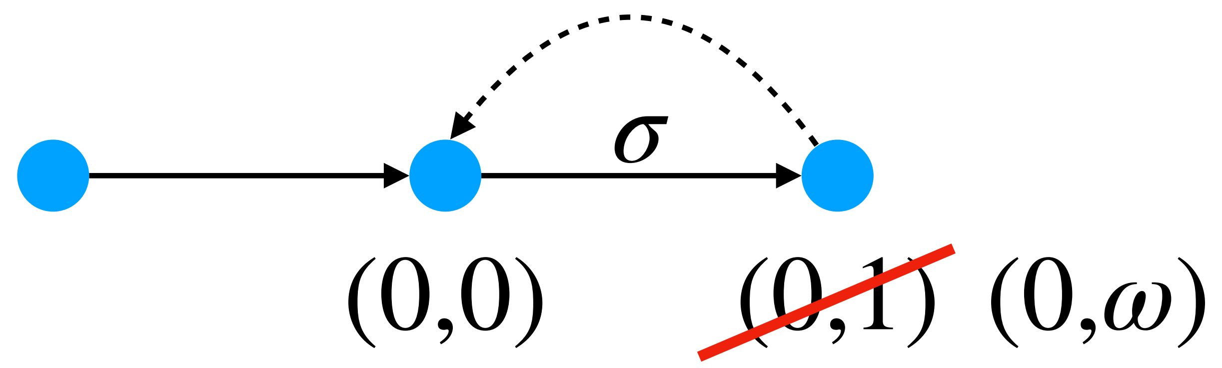
$$\Leftrightarrow \quad \exists s \in sol(A \cdot x = 0) \,.\, s(x[i]) > 0.$$

Support = the set of unbounded variables.

Support solution =
$s \in sol(A \cdot x = 0)$ giving a positive value to all variables in the support.

Note: Homogeneous solutions are stable under addition.

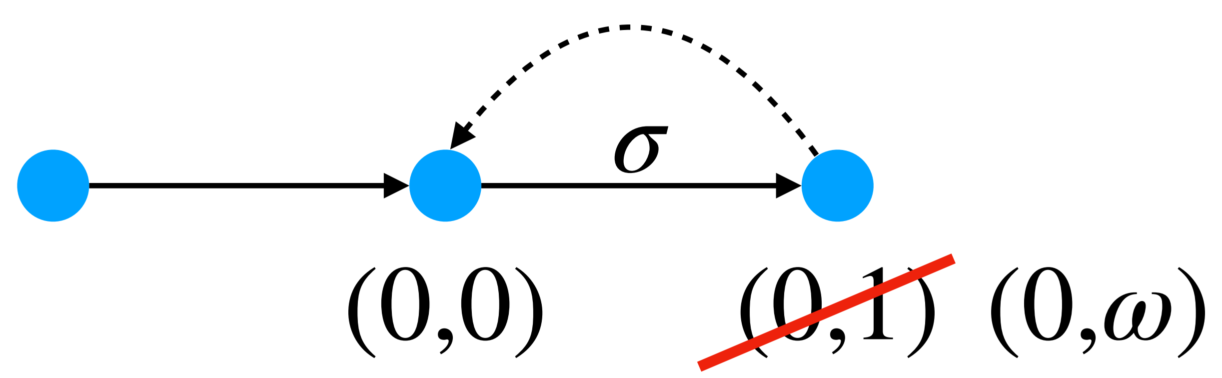
Deciding Reachability



⇒ $x[e]$ with $e \in \sigma$
$x[j]$ with j = 2

have to be unbounded in the solution space.

Deciding Reachability



⇒

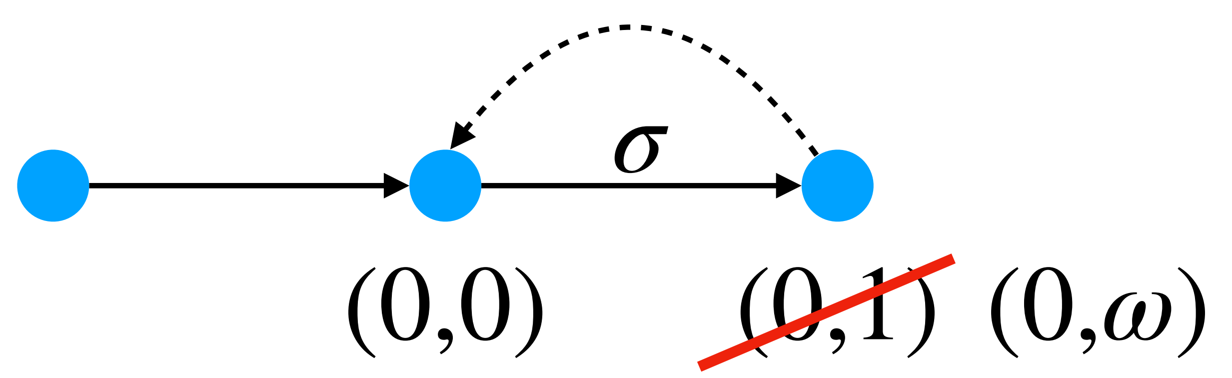
$x[e]$ with $e \in \sigma$

$x[j]$ with j = 2

have to be unbounded in the solution space.

So far:
Pumping where the solution space is unbounded
= pumping should yield a support solution.

Deciding Reachability



$\Rightarrow$ $x[e]$ with $e \in \sigma$
$x[j]$ with j = 2

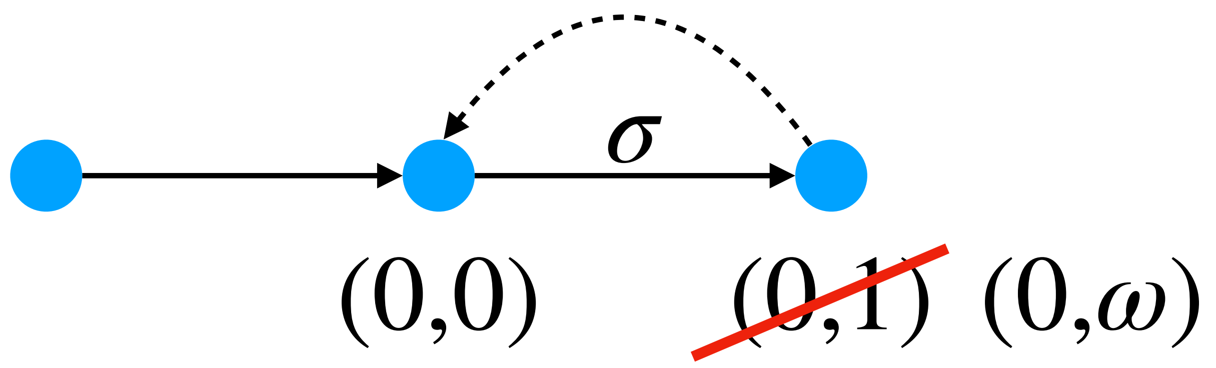
have to be unbounded in the solution space.

So far:
Pumping where the solution space is unbounded
= pumping should yield a support solution.

Problem:

σ may not match a support solution s.

Deciding Reachability



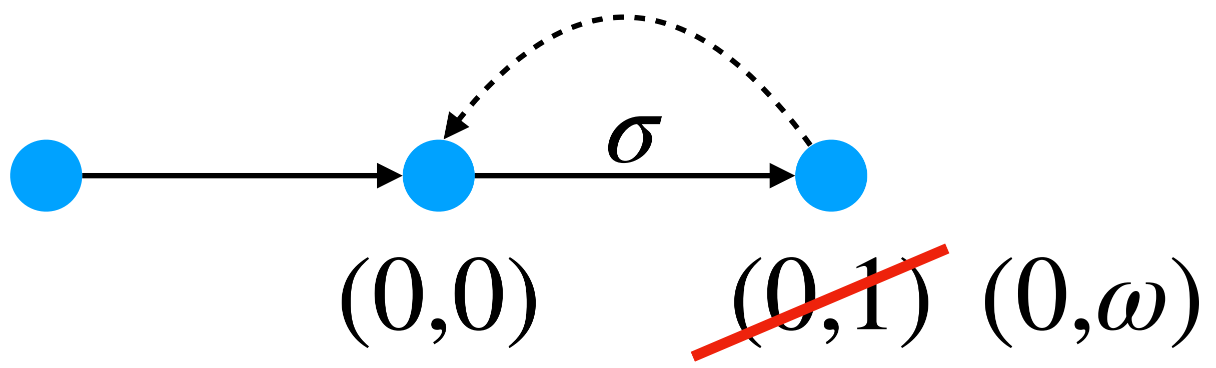
$\Rightarrow$ $x[e]$ with $e \in \sigma$ and $x[j]$ with j = 2 have to be unbounded in the solution space.

So far:
Pumping where the solution space is unbounded
= pumping should yield a support solution.

Problem:
σ may not match a support solution s.

Idea:
Turn $s - \psi(\sigma)$ into a path.

Deciding Reachability



$\Rightarrow$ $x[e]$ with $e \in \sigma$
$x[j]$ with j = 2

have to be unbounded in the solution space.

So far:

Pumping where the solution space is unbounded

= pumping should yield a support solution.

Problem:

σ may not match a support solution s.

Parikh image.

Idea:

Turn $s - \psi(\sigma)$ into a path.

Deciding Reachability

Deciding Reachability

Lemma (Euler-Kirchhoff):

Let $G = (V, E)$ be a strongly connected directed graph.

Let $x : \mathbb{N}^E$ satisfy

Deciding Reachability

Lemma (Euler-Kirchhoff):
Let $G = (V, E)$ be a strongly connected directed graph.
Let $x : \mathbb{N}^E$ satisfy

$$\sum_{e=(-,v)} x[e] = \sum_{e=(v,-)} x[e] \qquad \forall v \in V$$

$$x \geq 1$$

Deciding Reachability

Lemma (Euler-Kirchhoff):
Let $G = (V, E)$ be a strongly connected directed graph.
Let $x : \mathbb{N}^E$ satisfy

$$\sum_{e=(-,v)} x[e] = \sum_{e=(v,-)} x[e] \qquad \forall v \in V$$

$$x \geq 1$$

Then there is a cycle c in G with $\psi(c) = x$.
Also write $c = \langle x \rangle$.

Deciding Reachability

Lemma (Euler-Kirchhoff):
Let $G = (V, E)$ be a strongly connected directed graph.
Let $x : \mathbb{N}^E$ satisfy

$$\sum_{e=(-,v)} x[e] = \sum_{e=(v,-)} x[e] \qquad \forall v \in V$$

$$x \geq 1$$

Then there is a cycle c in G with $\psi(c) = x$.
Also write $c = \langle x \rangle$.

Realization.

Deciding Reachability

Deciding Reachability

Pumping should yield a support solution:

Deciding Reachability

Pumping should yield a support solution:

Let s be a support solution with

Deciding Reachability

Pumping should yield a support solution:

Let s be a support solution with

$$d := s - \psi(up) - \psi(dn) \geq 1 \ .$$

Deciding Reachability

Pumping should yield a support solution:

Let s be a support solution with

$$d := s - \psi(up) - \psi(dn) \geq 1 \ .$$

By the Euler-Kirchhoff Lemma, the difference can be realized by a cycle

Deciding Reachability

Pumping should yield a support solution:

Let s be a support solution with

$$d := s - \psi(up) - \psi(dn) \geq 1 \ .$$

By the Euler-Kirchhoff Lemma, the difference can be realized by a cycle

$$w = \langle d \rangle \ .$$

Deciding Reachability

Pumping should yield a support solution:

Let s be a support solution with

$$d := s - \psi(up) - \psi(dn) \geq 1 \ .$$

By the Euler-Kirchhoff Lemma, the difference can be realized by a cycle

$$w = \langle d \rangle \ .$$

Now $\psi(up) + \psi(w) + \psi(dn) = s$ and we say they match.

Deciding Reachability

Deciding Reachability

Lambert's Iteration Lemma [TCS'92]:

For c large enough, one can even fit in a $\mathbb{Z}$-cycle that reaches the exit from the entry marking:

Deciding Reachability

Lambert's Iteration Lemma [TCS'92]:

For c large enough, one can even fit in a $\mathbb{Z}$-cycle that reaches the exit from the entry marking:

$$up^c \,.\, \rho \,.\, w^c \,.\, dn^c \,.$$

Deciding Reachability

Lambert's Iteration Lemma [TCS'92]:

For c large enough, one can even fit in a $\mathbb{Z}$-cycle
that reaches the exit from the entry marking:

$$up^c \,.\, \rho \,.\, w^c \,.\, dn^c \,.$$

Since pumping happens in a support solution, this still solves reachability.
Notably, it stays non-negative.

Deciding Reachability

Deciding Reachability

Problem: Precovering graphs may not be perfect.

Deciding Reachability

Problem: Precovering graphs may not be perfect.

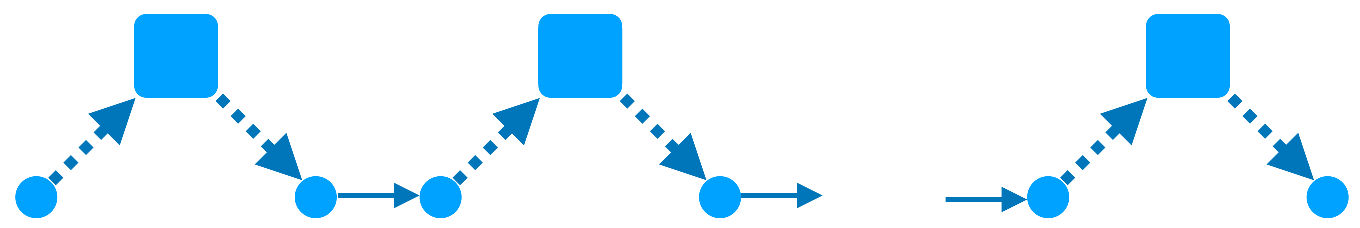
Solution: Decompose them into sequences of precovering graphs, MGTS:

Deciding Reachability

Problem: Precovering graphs may not be perfect.

Solution: Decompose them into sequences of precovering graphs, MGTS:

MGTS



Deciding Reachability

Deciding Reachability

Deciding Reachability:

Deciding Reachability

Deciding Reachability:

As long as perfectness fails, decomposition is guaranteed to succeed.

Deciding Reachability

Deciding Reachability:

As long as perfectness fails, decomposition is guaranteed to succeed.

It yields finite sets of MGTS that are smaller in a well-founded order.

Deciding Reachability

Deciding Reachability:

As long as perfectness fails, decomposition is guaranteed to succeed.

It yields finite sets of MGTS that are smaller in a well-founded order.

Hence, perfectness will eventually hold.

Deciding Reachability

Deciding Reachability:

As long as perfectness fails, decomposition is guaranteed to succeed.

It yields finite sets of MGTS that are smaller in a well-founded order.

Hence, perfectness will eventually hold.

For perfect MGTS,

Deciding Reachability

Deciding Reachability:

As long as perfectness fails, decomposition is guaranteed to succeed.

It yields finite sets of MGTS that are smaller in a well-founded order.

Hence, perfectness will eventually hold.

For perfect MGTS,

$$\mathbb{N}\text{-reachability holds} \quad \Leftrightarrow \quad \mathbb{Z}\text{-reachability holds.}$$

We are hiring!

Associate Professorship in Verification (tenured, W2)

Please inform your postdocs and colleagues who may be interested!
Please contact me for questions!