ATVA 2023

Originally presented in ATVA 2022

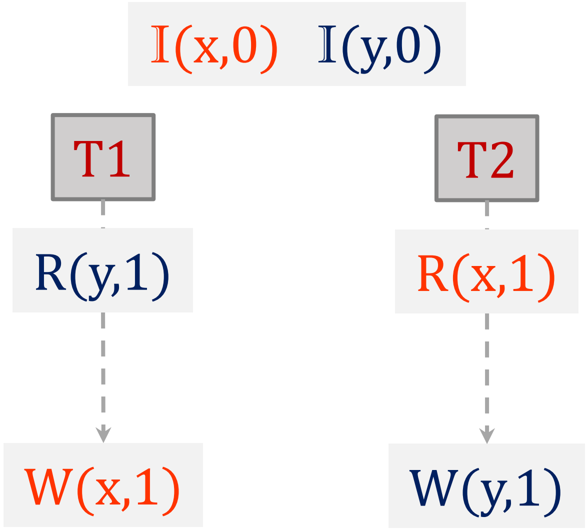
Fence Synthesis under the C11 Memory Model

Sanjana Singh[1], Divyanjali Sharma[1], Ishita Jaju[2] and Subodh Sharma[1]

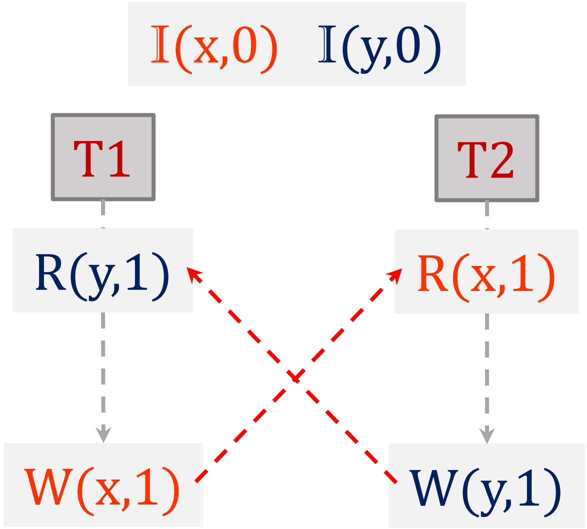
[1] Indian Institute of Technology Delhi, India

[2] Uppsala University, Sweden

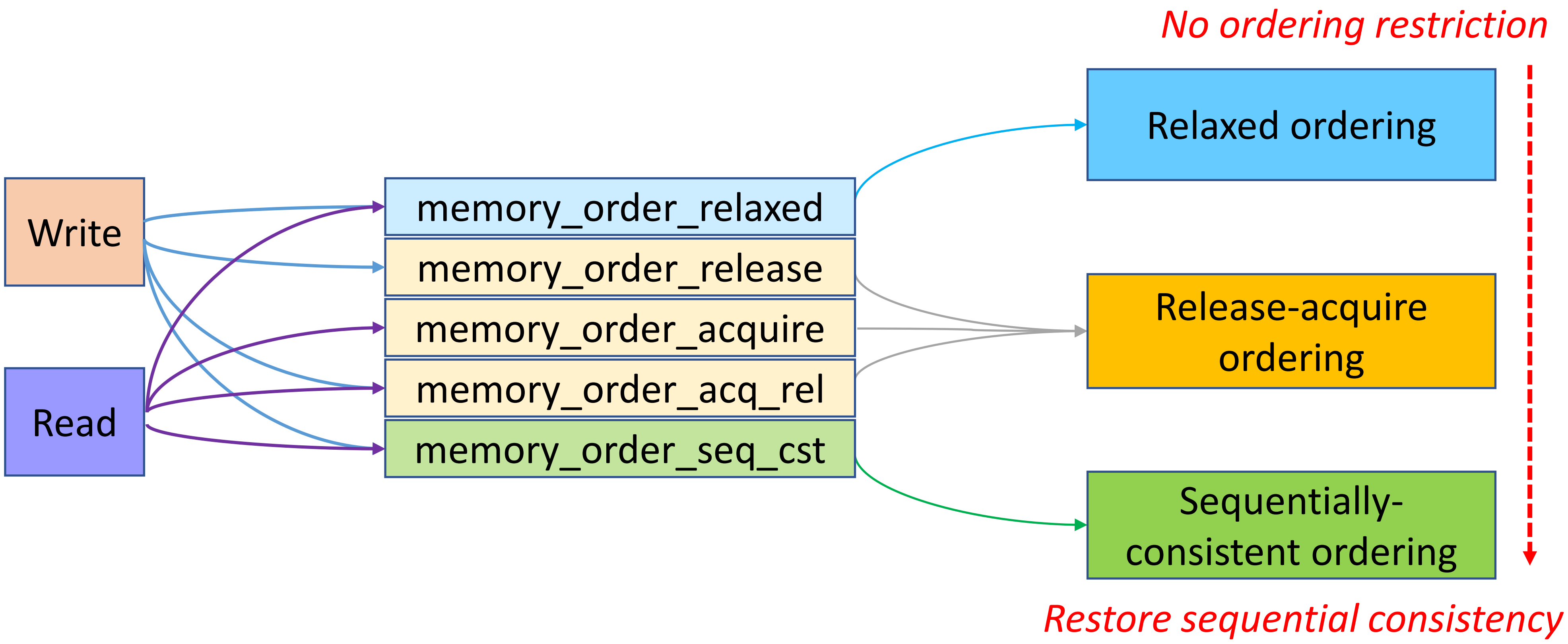
Order might be critical for correctness



Order might be critical for correctness



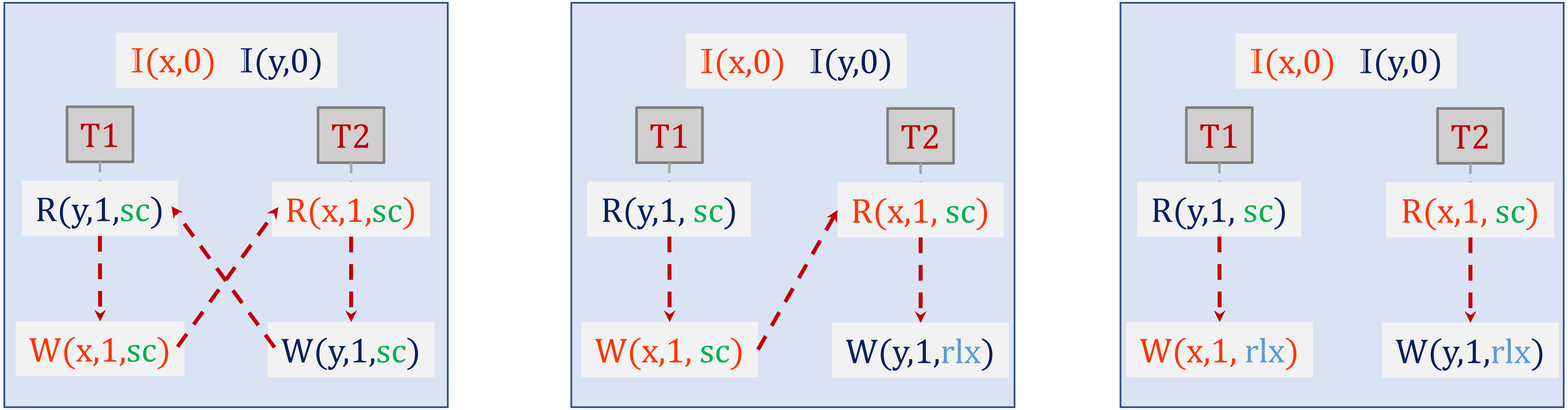
C/C++11 (C11) memory orders



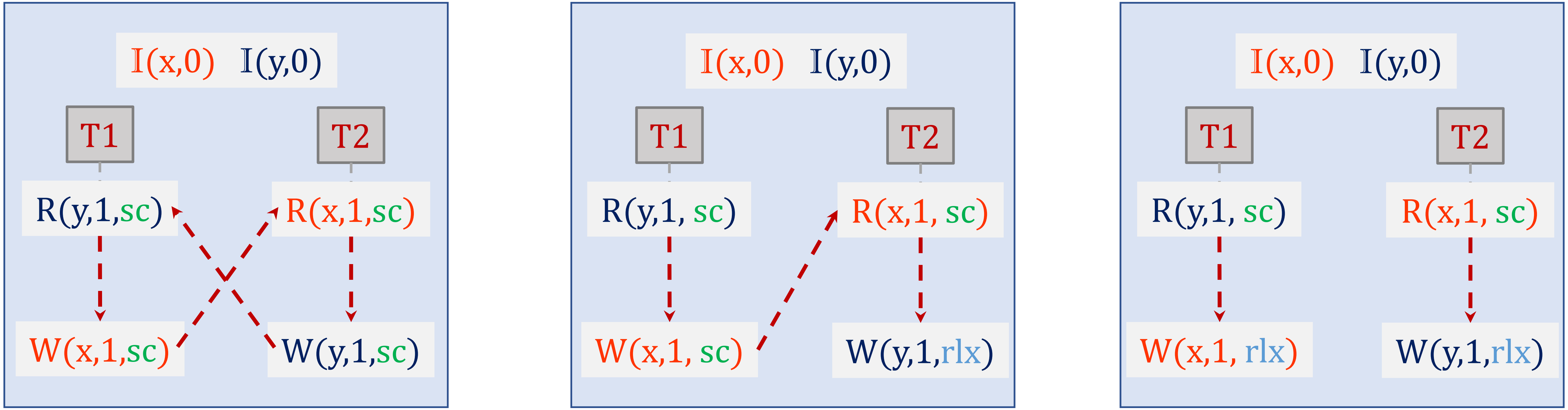

```
x.store(1, memory_order_relaxed)
x.load(memory_order_relaxed)
```

```
x.store(1, memory_order_seq_cst)
x.load(memory_order_ seq_cst)
```

Order might be critical for correctness



Order might be critical for correctness

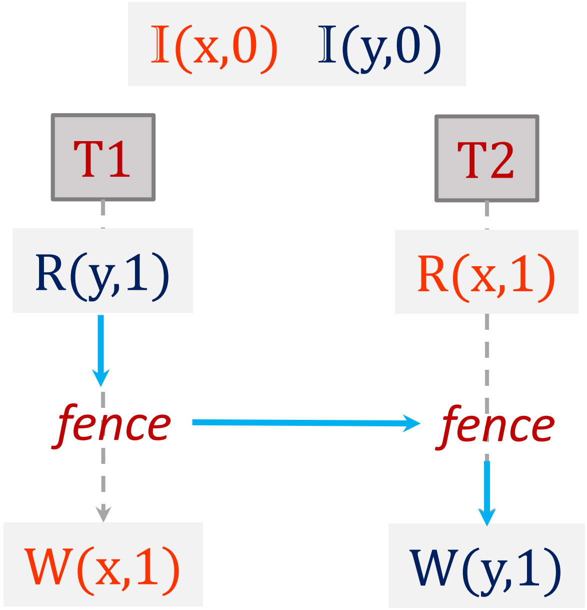


memory order specification to ensure performance and correctness should not be left to humans.

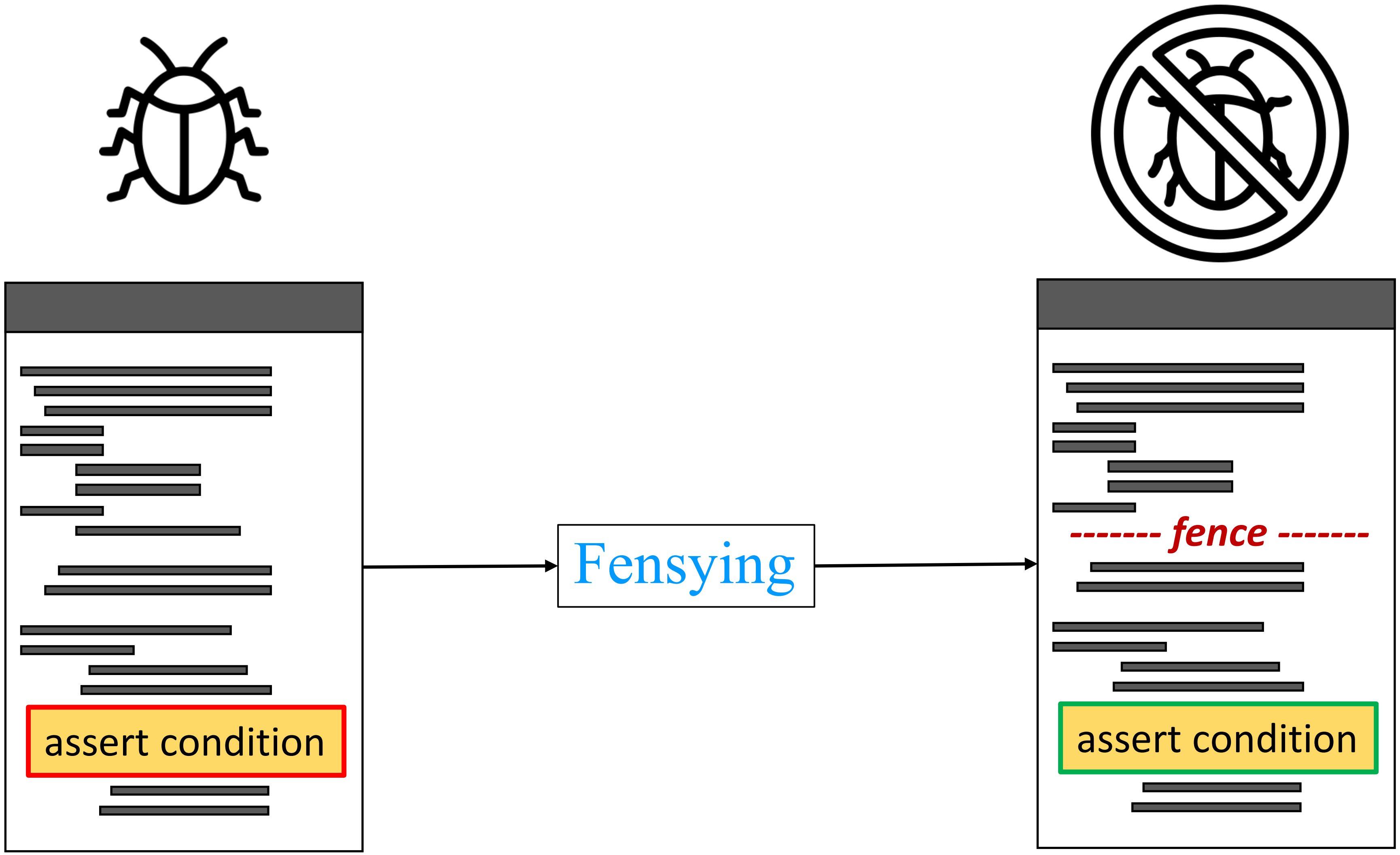
Oberhauser et al., ASPLOS'21

Ordering with fences

- Order might be critical for correctness
- Fences restore order

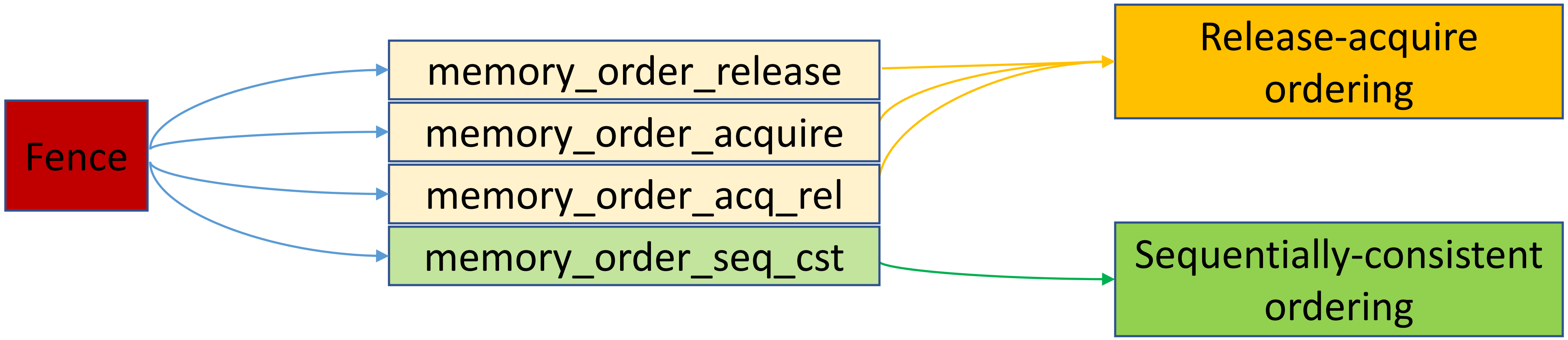


Fence synthesis for automated repair



C11 fences

- Tools for ordering restrictions.
- Support degrees of ordering guarantees



atomic_thread_fence(memory_order_acquire)

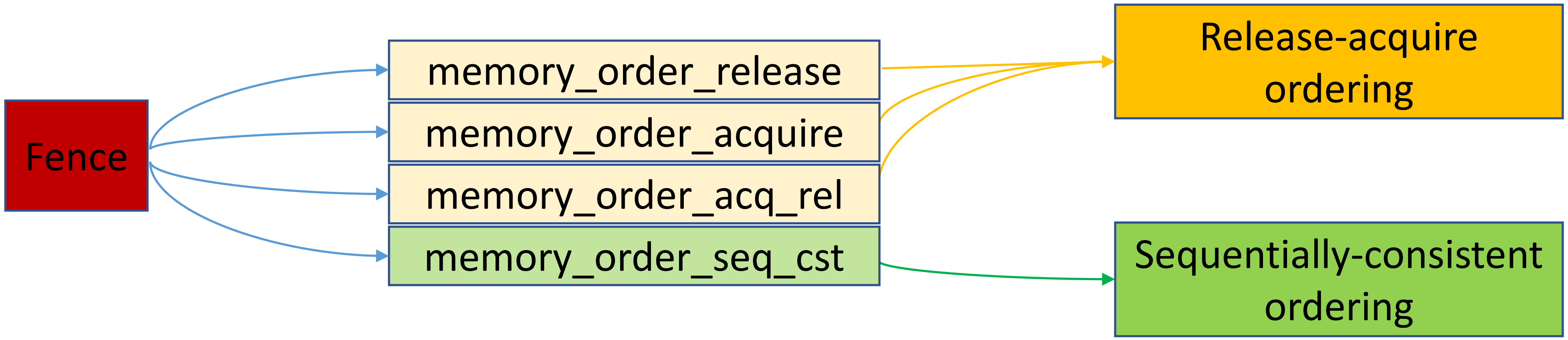
atomic_thread_fence(memory_order_seq_sct)

C11 fences

- Tools for ordering restrictions.
- Support degrees of ordering guarantees

Synthesis challenges:

How many and where?
Which memory order?



Existing fence synthesis techniques

- Imprecise (*Existing techniques assume an axiomatic definition of ordering*)
  - Strong implicit ordering ⇒ miss C11 bugs + insufficient barriers
  - Weak implicit ordering ⇒ unnecessarily strong barriers
- Reduced portability

Existing fence synthesis techniques

- Imprecise (*Existing techniques assume an axiomatic definition of ordering*)
  - Strong implicit ordering ⇒ miss C11 bugs + insufficient barriers
  - Weak implicit ordering ⇒ unnecessarily strong barriers
- Reduced portability

Fence synthesis for C11

- Precisely detect C11 traces
- Synthesize portable C11 fences

Fensying: Optimal C11 fence synthesis

Optimal fence synthesis

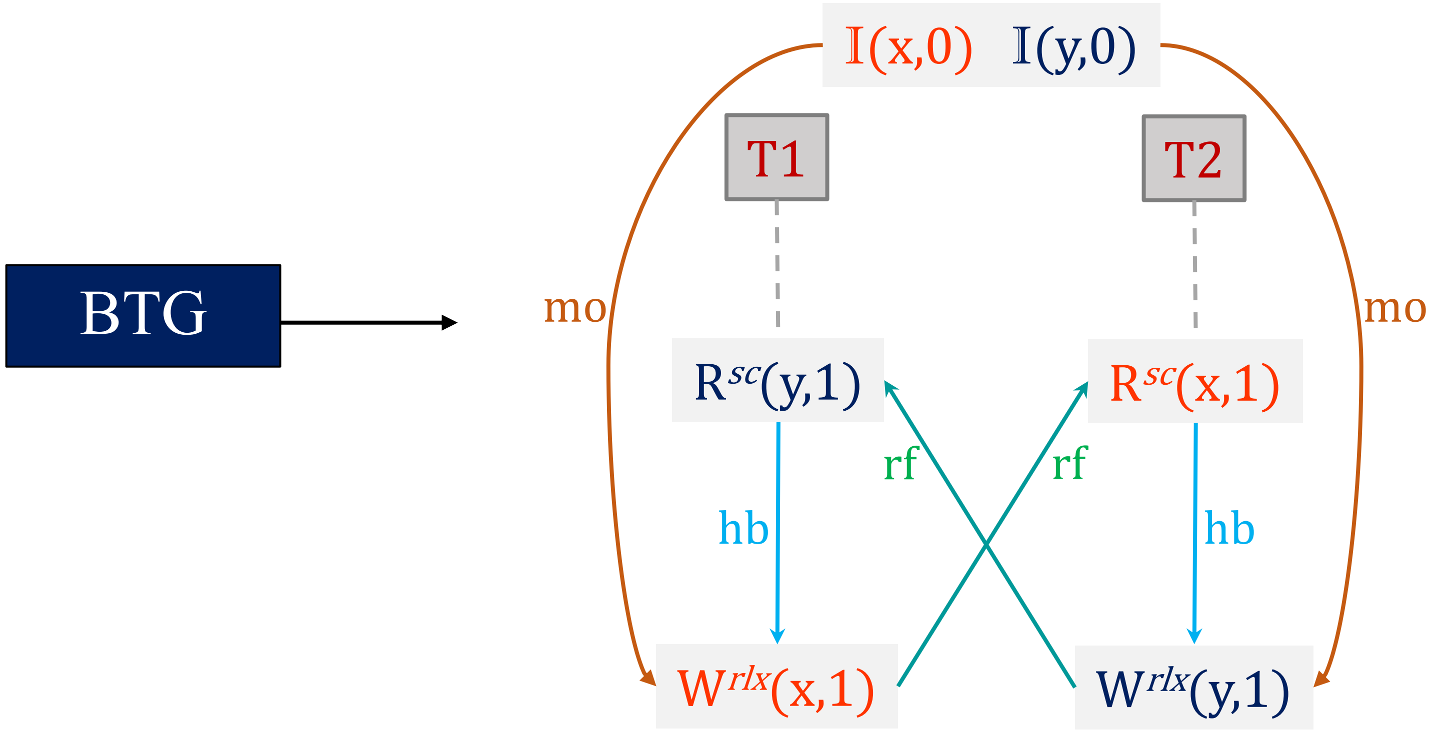
- Smallest set of fences
- Weakest type of fences

solution not unique

Fensying technique

hb	happens-before
rf	reads-from
mo	modification-order

Step 1 get the set of buggy traces

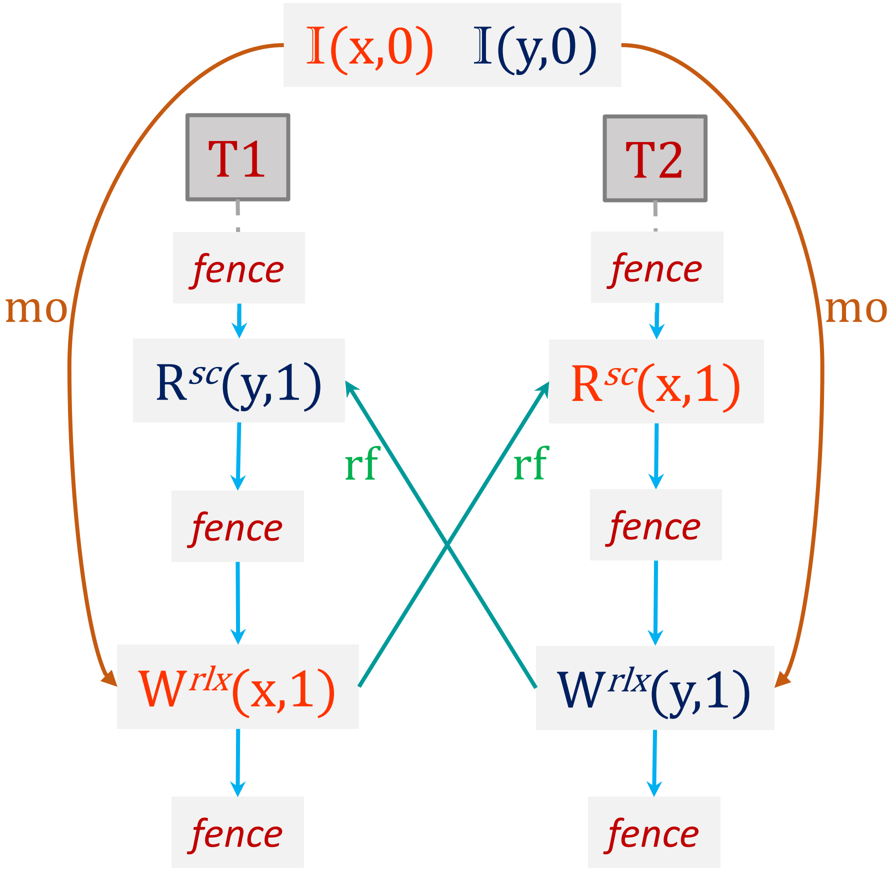


buggy trace generator (BTG): CDSChecker, open source SMC [Norris and Demsky, OOPSLA'13]

Fensying technique

hb	happens-before
rf	reads-from
mo	modification-order

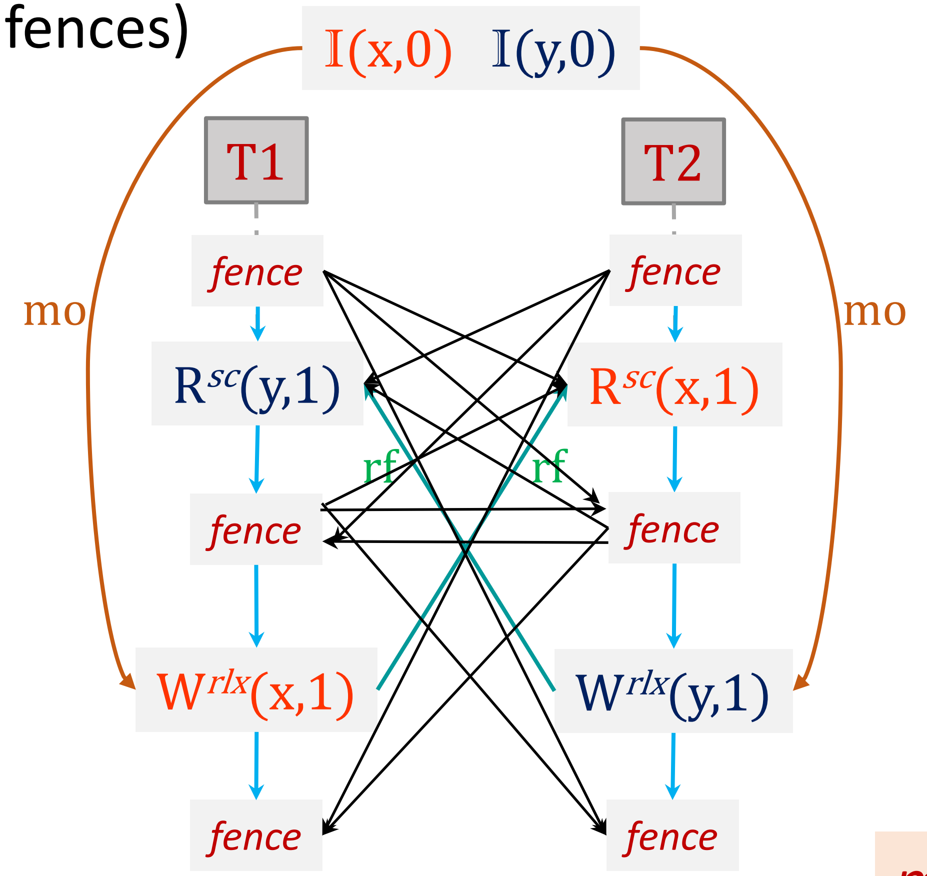
Step 2 generate *intermediate* trace



Fensying technique

hb happens-before
rf reads-from
mo modification-order

Step 2 generate *intermediate* trace
(additional ordering with fences)

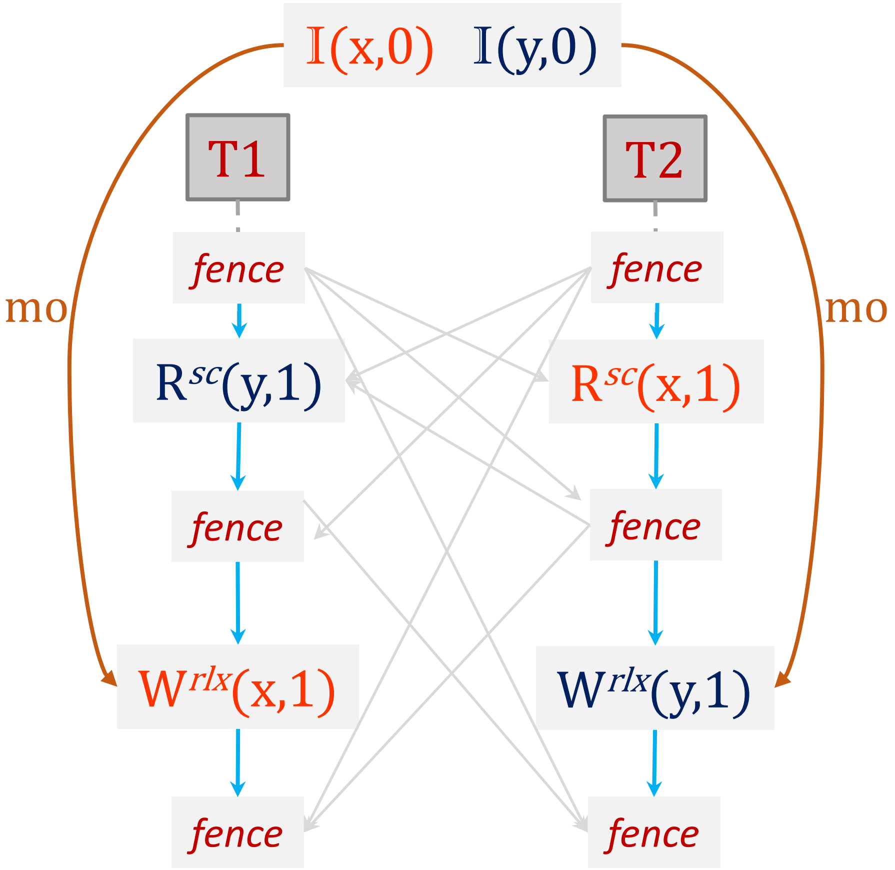


maximum possible fence ordering

Fensying technique

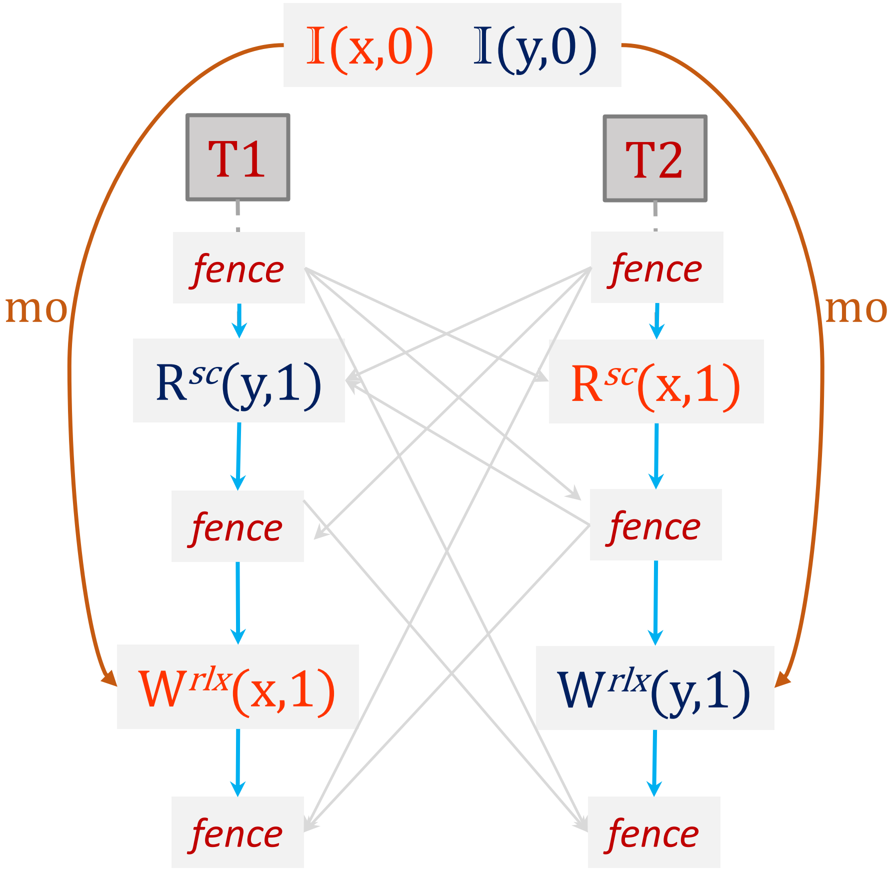
C11 coherence conditions:
hb is irreflexive
rf; hb is irreflexive
mo; hb is irreflexive
mo; rf; hb is irreflexive
mo; hb; rfinv is irreflexive
mo; rf; hb; rfinv is irreflexive
so is irreflexive

Step 3 detect violations of coherence



Fensying technique

Step 3 detect violations of coherence



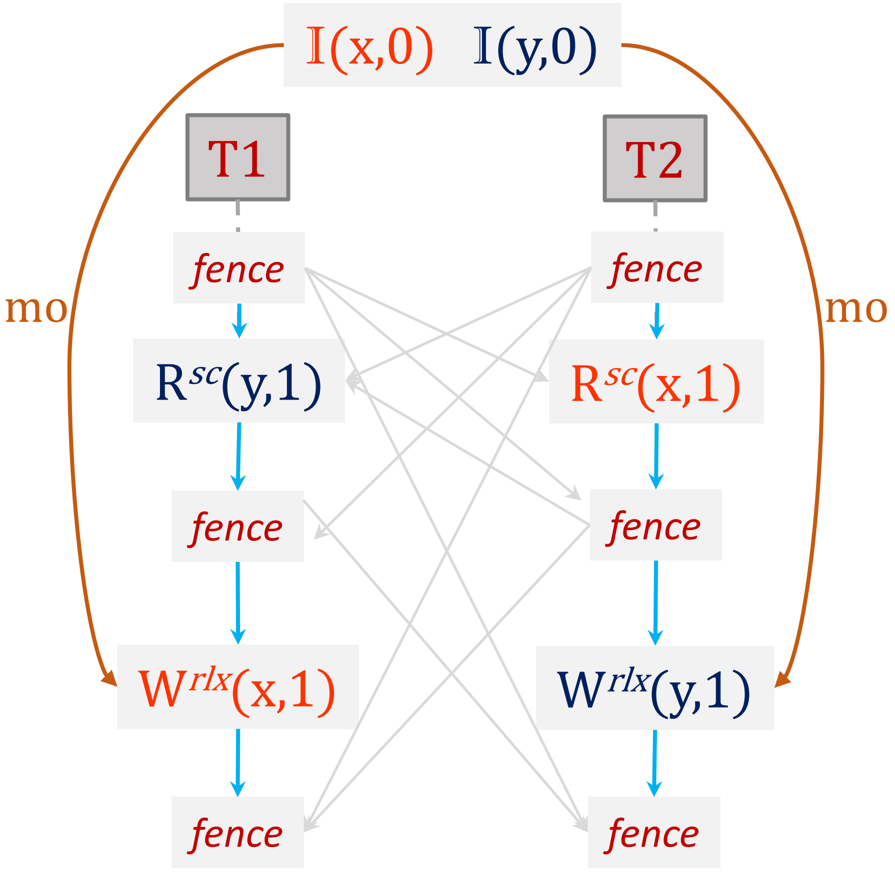
C11 coherence conditions:

hb is irreflexive
rf; hb is irreflexive
mo; hb is irreflexive
mo; rf; hb is irreflexive
mo; hb; rfinv is irreflexive
mo; rf; hb; rfinv is irreflexive
so is irreflexive

[Lahav et al. PLDI 2017]

Fensying technique

Step 3 detect violations of coherence



C11 coherence conditions:

hb is irreflexive

rf; hb is irreflexive

mo; hb is irreflexive

mo; rf; hb is irreflexive

mo; hb; rfinv is irreflexive

mo; rf; hb; rfinv is irreflexive

so is irreflexive

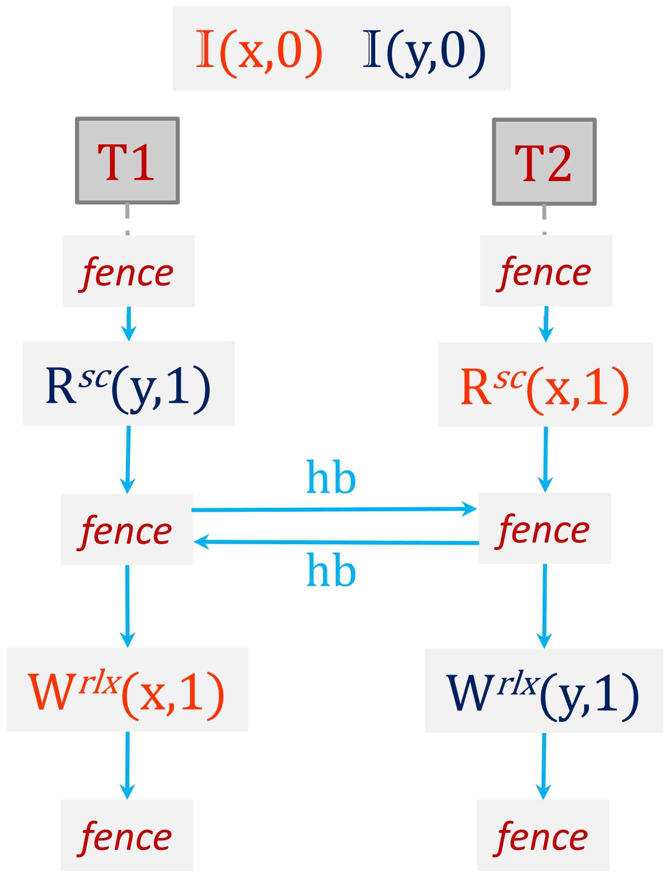
Total-order on sc events

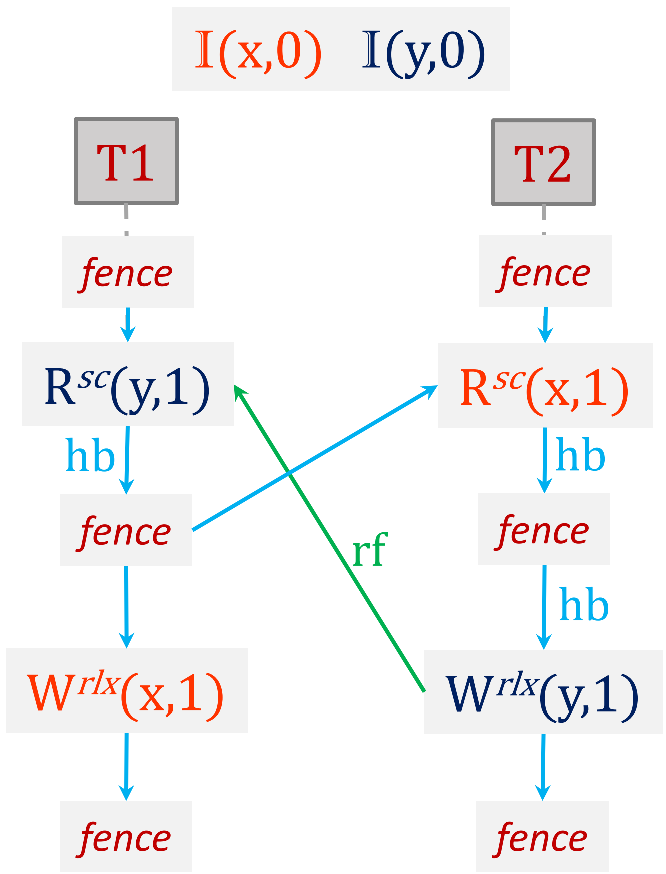
reflexive so ⇒ to not feasible

Fensying technique

C11 coherence conditions:
hb is irreflexive
rf; hb is irreflexive
mo; hb is irreflexive
mo; rf; hb is irreflexive
mo; hb; rfinv is irreflexive
mo; rf; hb; rfinv is irreflexive
so is irreflexive

Step 3 detect violations of coherence





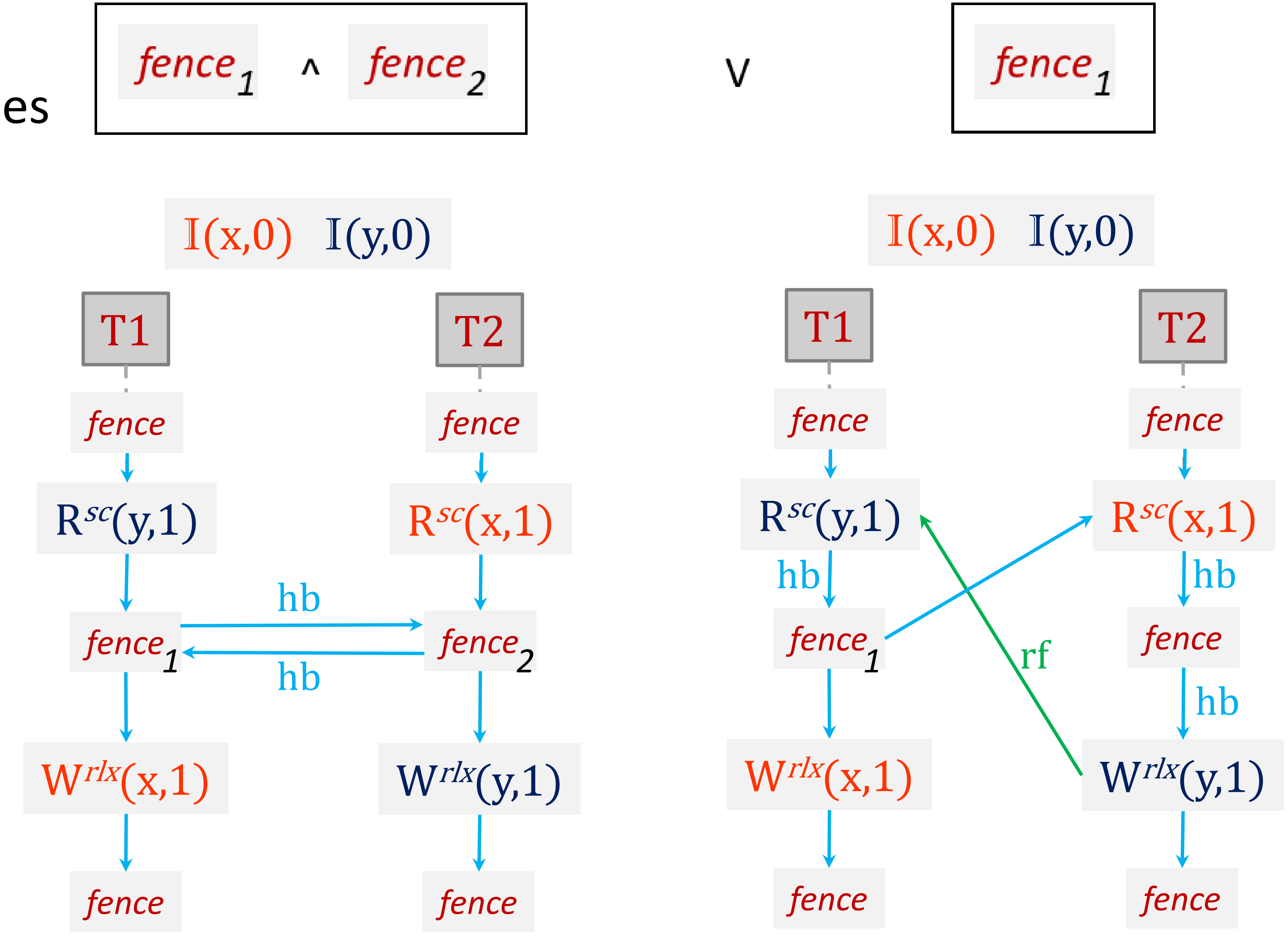
Johnson's algorithm for cycle detection

[Johnson, D.B, SICOMP'1975]

Fensying technique

Step 4 find the smallest set of fences

min-model of a SAT query



Fensying technique

Step 4 find the smallest set of fences

min-model of a SAT query

$$(\boxed{fence_1 \wedge fence_2} \vee \boxed{fence_1}) \rightarrow \text{from } i^{th} \text{ buggy traces}$$

$$\bigwedge$$

$$(\boxed{fence_3 \wedge fence_4} \vee \boxed{fence_1}) \rightarrow \text{from } j^{th} \text{ buggy traces}$$

Fensying technique

Optimal fence synthesis
- Smallest set of fences ✓
- Weakest type of fences

Step 4 find the smallest set of fences

min-model of a SAT query

$$(\boxed{fence_1 \wedge fence_2} \vee \boxed{fence_1}) \rightarrow \textit{from } i^{th} \textit{ buggy traces}$$

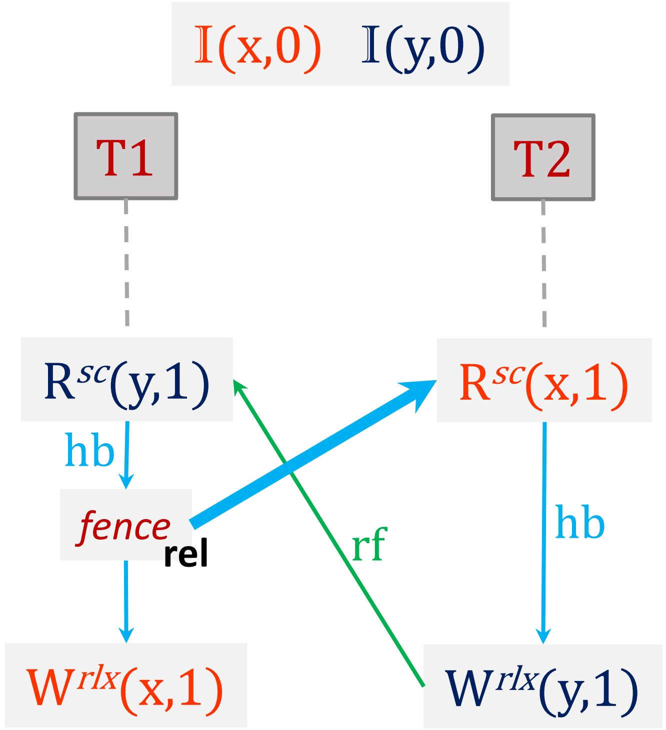
$$\bigwedge$$

$$(\boxed{fence_3 \wedge fence_4} \vee \boxed{fence_1}) \rightarrow \textit{from } j^{th} \textit{ buggy traces}$$

Fensying technique

Optimal fence synthesis
- Smallest set of fences ✔
- Weakest type of fences

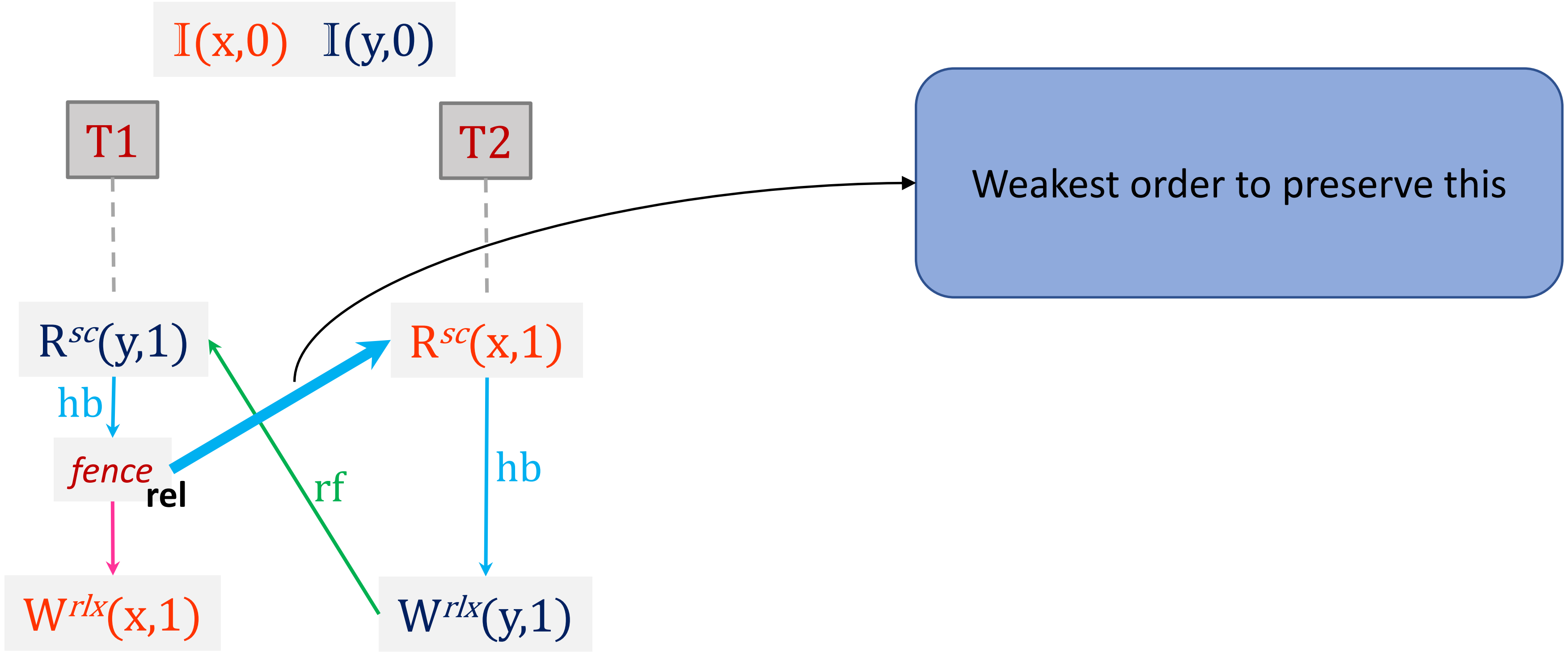
Step 5 find weakest order $\boxed{fence_1}$



Fensying technique

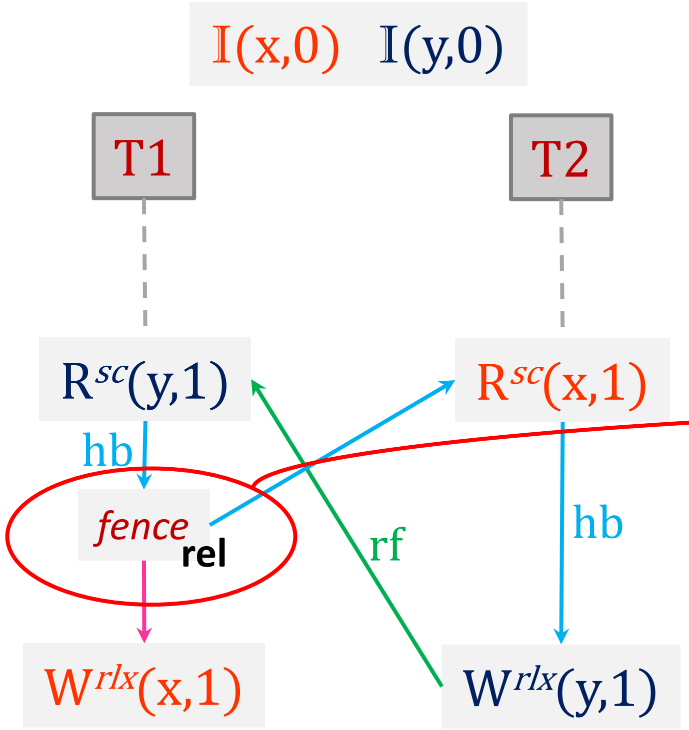
Optimal fence synthesis
- Smallest set of fences ✓
- Weakest type of fences ✓

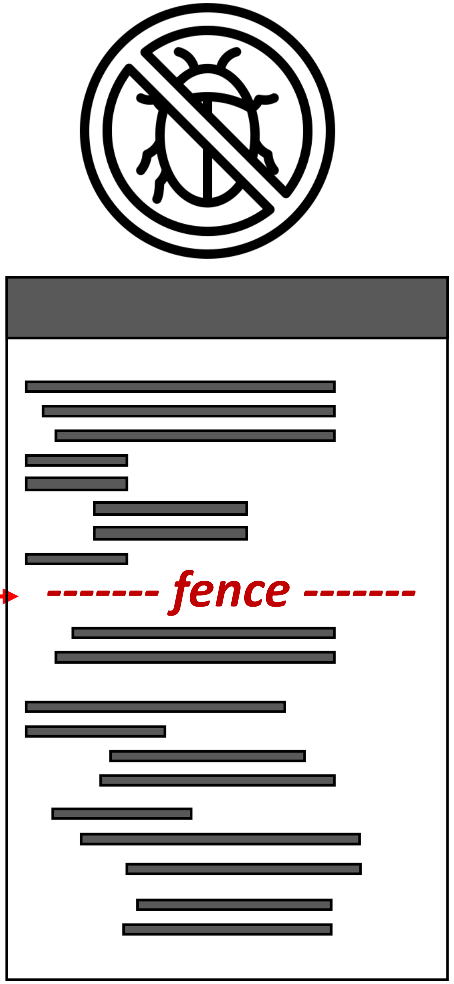
Step 5 find weakest order *fence*$_1$



Fensying technique

Step 5 find weakest order *fence*$_1$



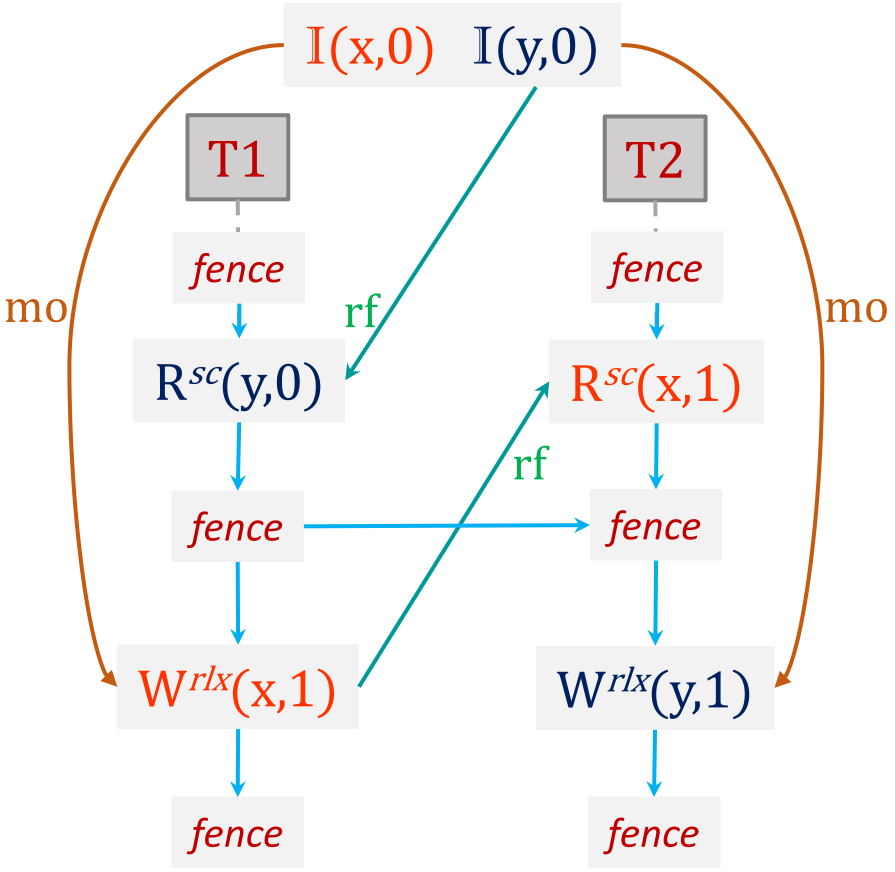


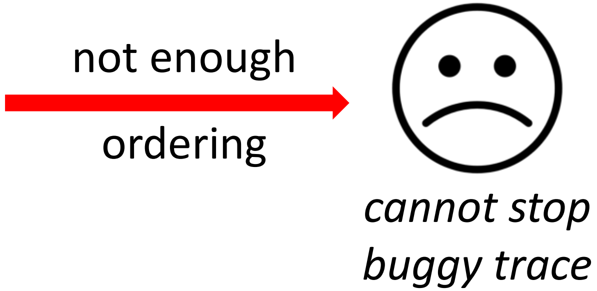
fixed program

Fensying technique

C11 coherence conditions:
hb is irreflexive
rf; hb is irreflexive
mo; hb is irreflexive
mo; rf; hb is irreflexive
mo; hb; rfinv is irreflexive
mo; rf; hb; rfinv is irreflexive
so is irreflexive

Step 3 detect violations of coherence

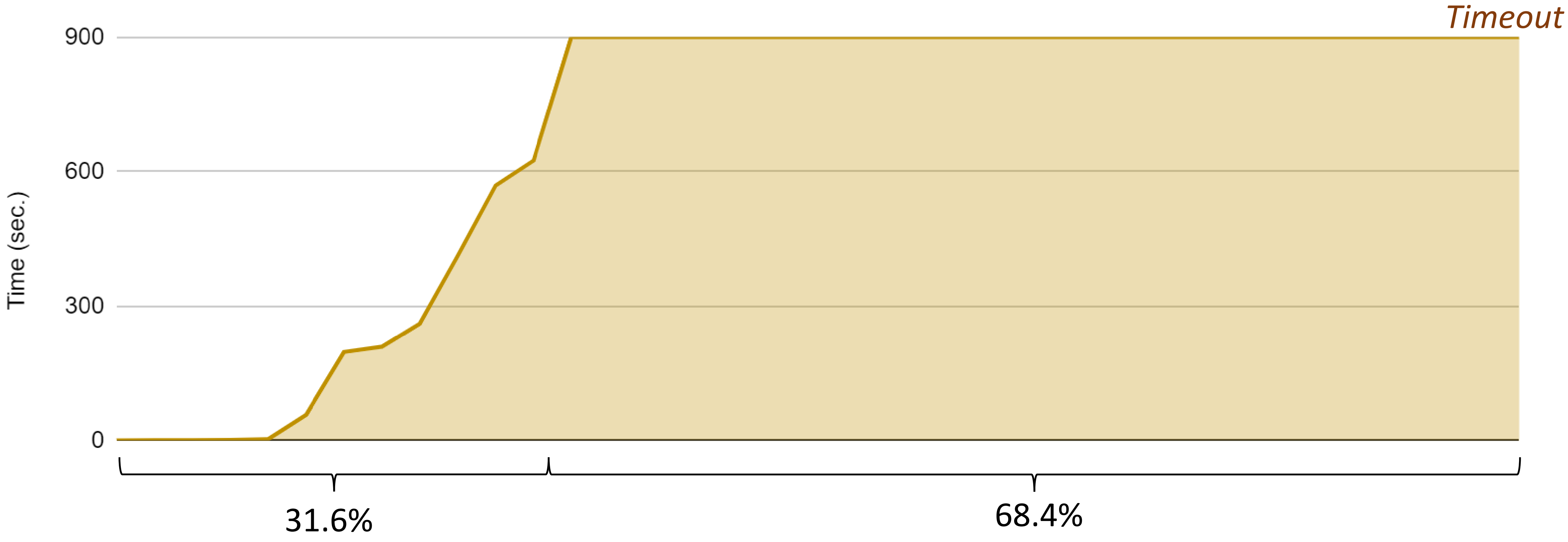




Fensying: Optimal C11 fence synthesis

- Smallest set of fences
- Weakest type of fences

NP-hard [Taheri et al., DISC'19]



Benchmarks source: Singh et al., TASE'21, Abdulla at al., PLDI'19, Abdulla at al., OOPSLA'18, Norris & Demsky, OOPSLA'13

fastFenSying : *near*-Optimal C11 fence synthesis

Fensying

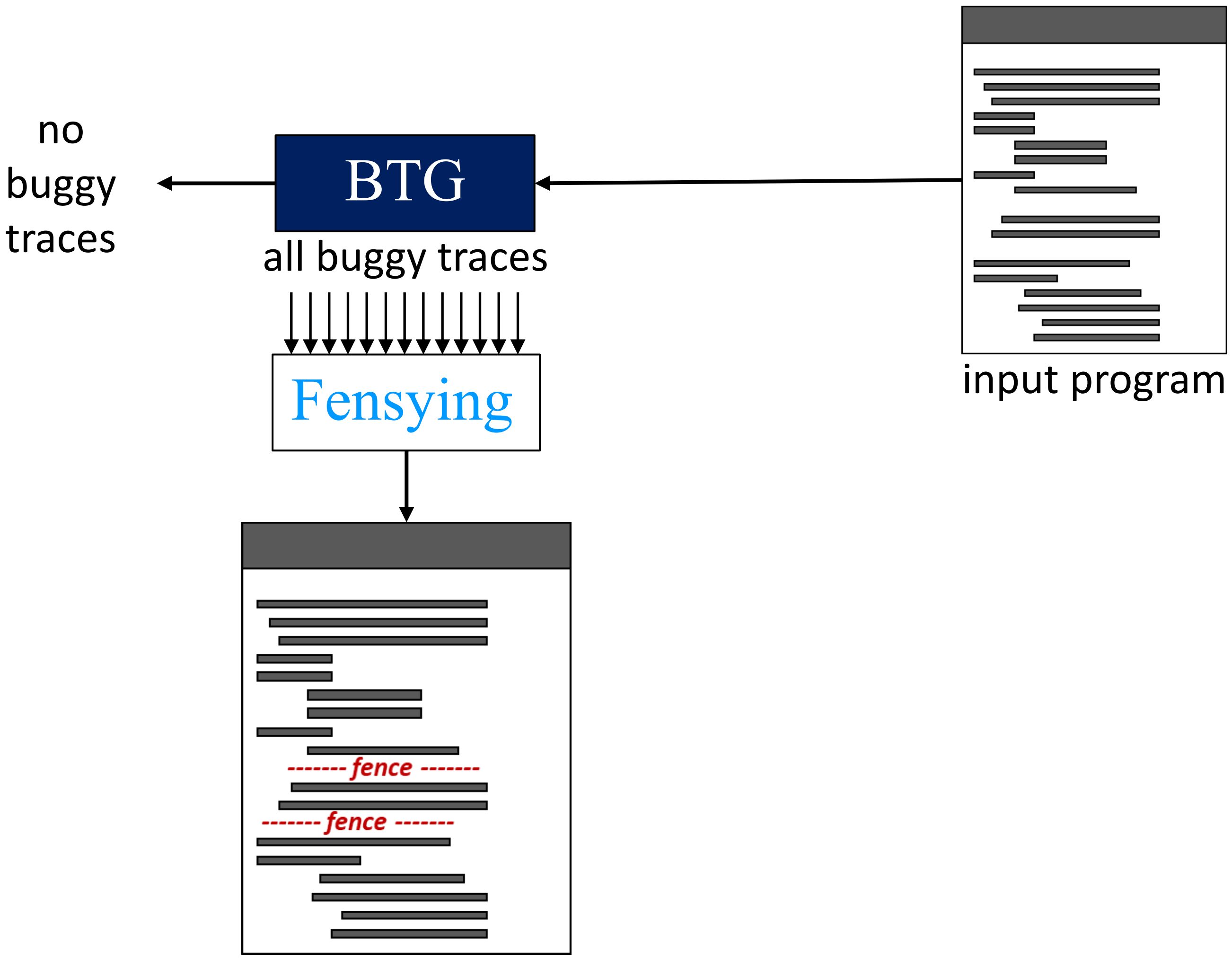
- Sound
- Optimal
- Slow
- Doesn't scale

fastFenSying

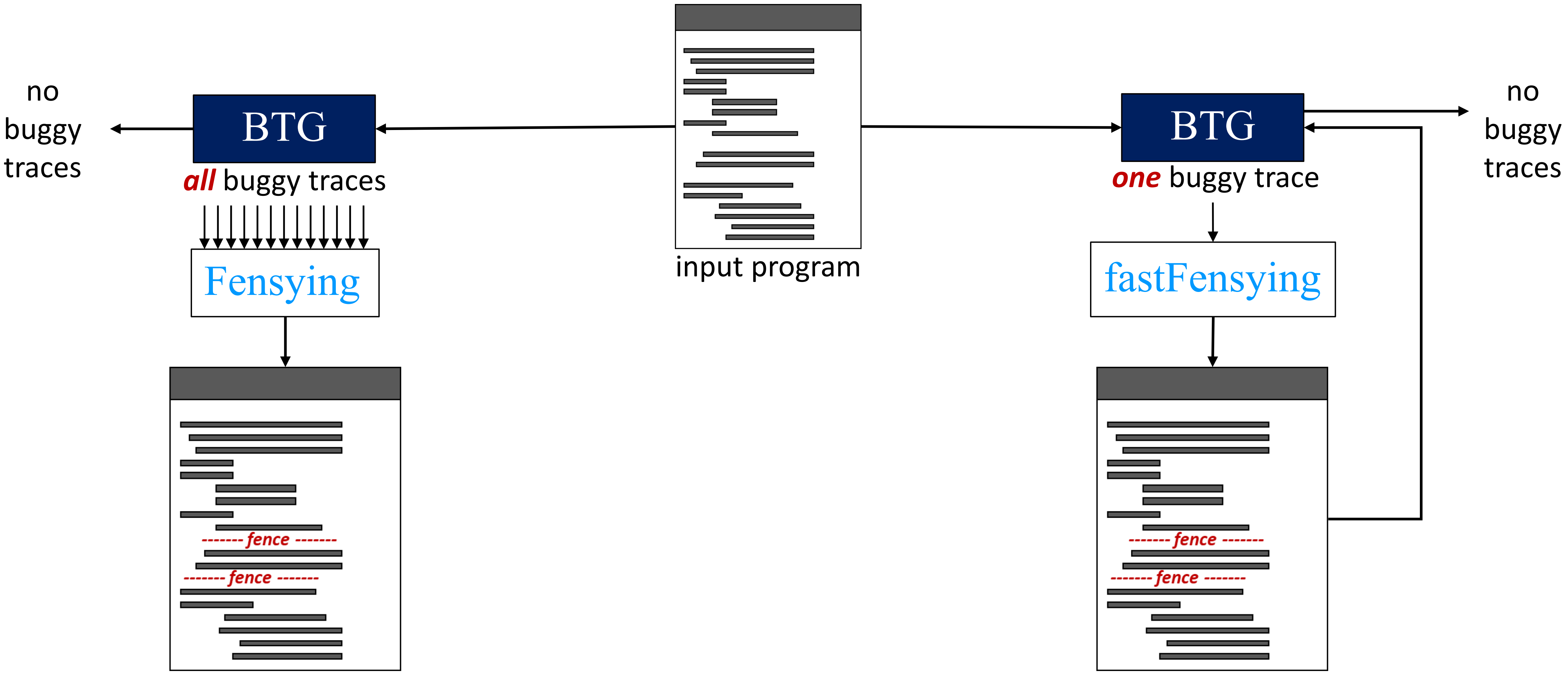
- Sound
- *near*-Optimal
- Fast
- Scales

Sound: stops every buggy trace that can be stopped.
Optimal: synthesizes minimal and weakest fences.
Near-optimal: provably optimal for one trace, and empirically optimal for all traces in 99.5% tests

Fensying *vs* fastFenSying



Fensying *vs* fastFenSying



Fensying *vs* fastFenSying

Theorem: Fensying is sound.

Theorem: Fensying is optimal.

Theorem: fastFensying is sound.

Sound: stops a buggy trace that can be stopped.
Optimal: synthesizes precise fences.

Experiments

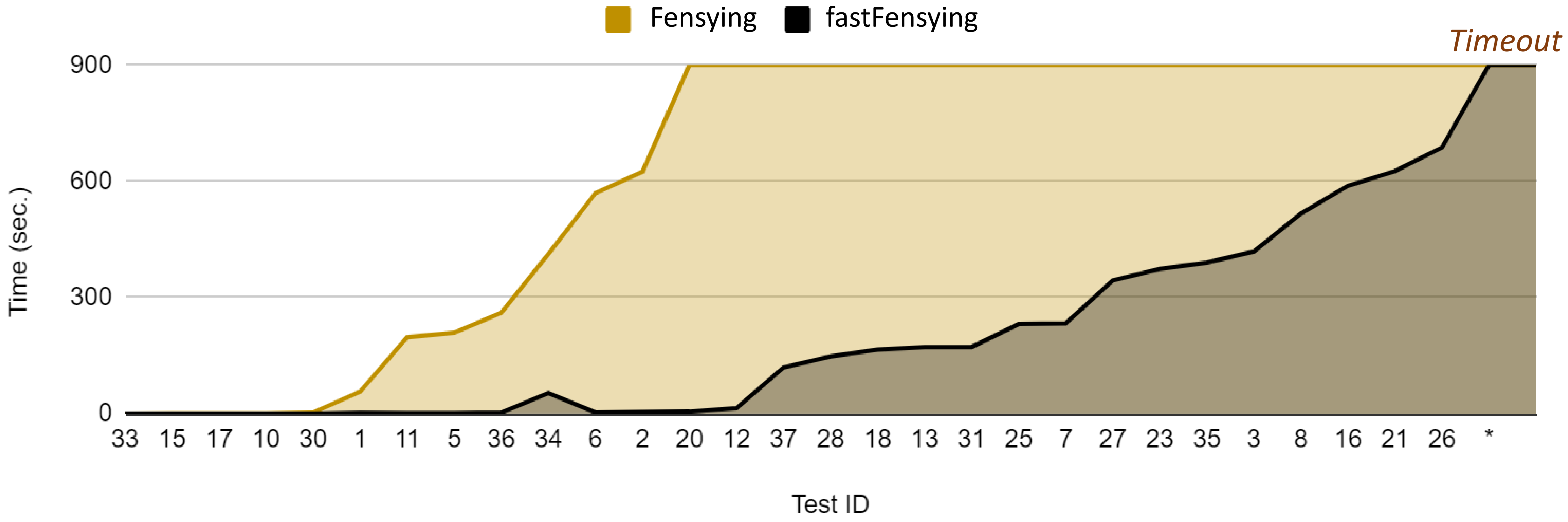
tested on 1389 litmus tests of buggy C11 programs

Fensying and fastFensying stop buggy traces

Fensying performs optimally

Litmus tests source: Abdulla at al., OOPSLA'18

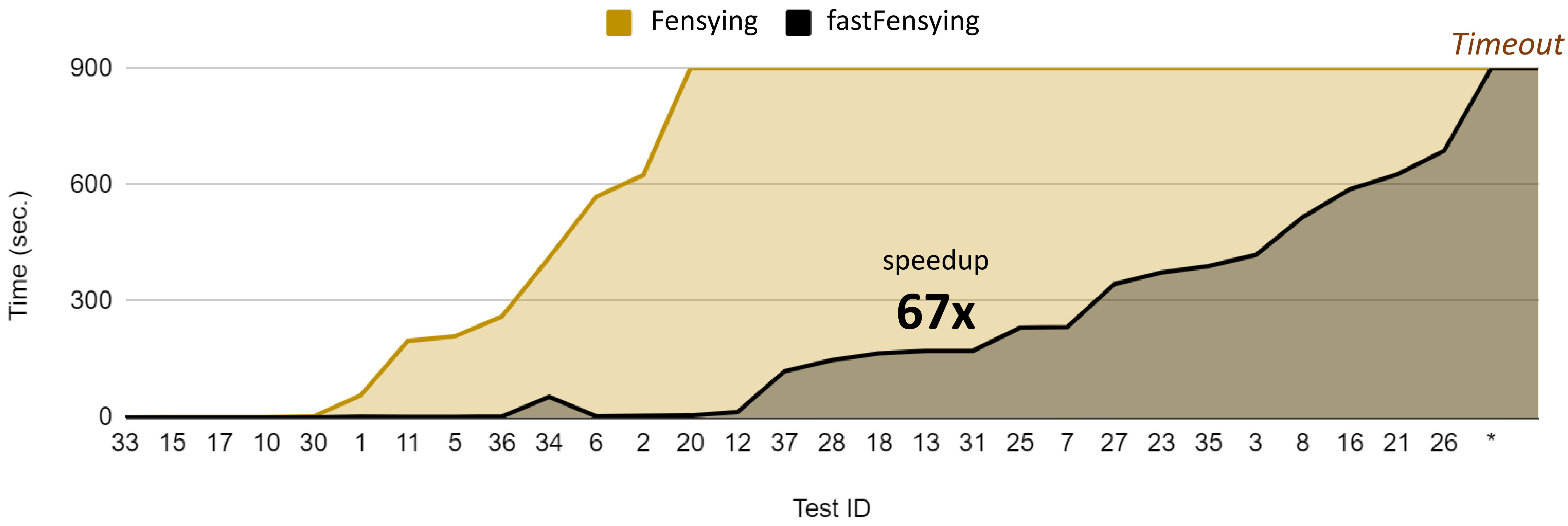
Experiments



* tests that timeout for both Fensying and fastFensying

Benchmarks source: Singh et al., TASE'21, Abdulla at al., PLDI'19, Abdulla at al., OOPSLA'18, Norris & Demsky, OOPSLA'13

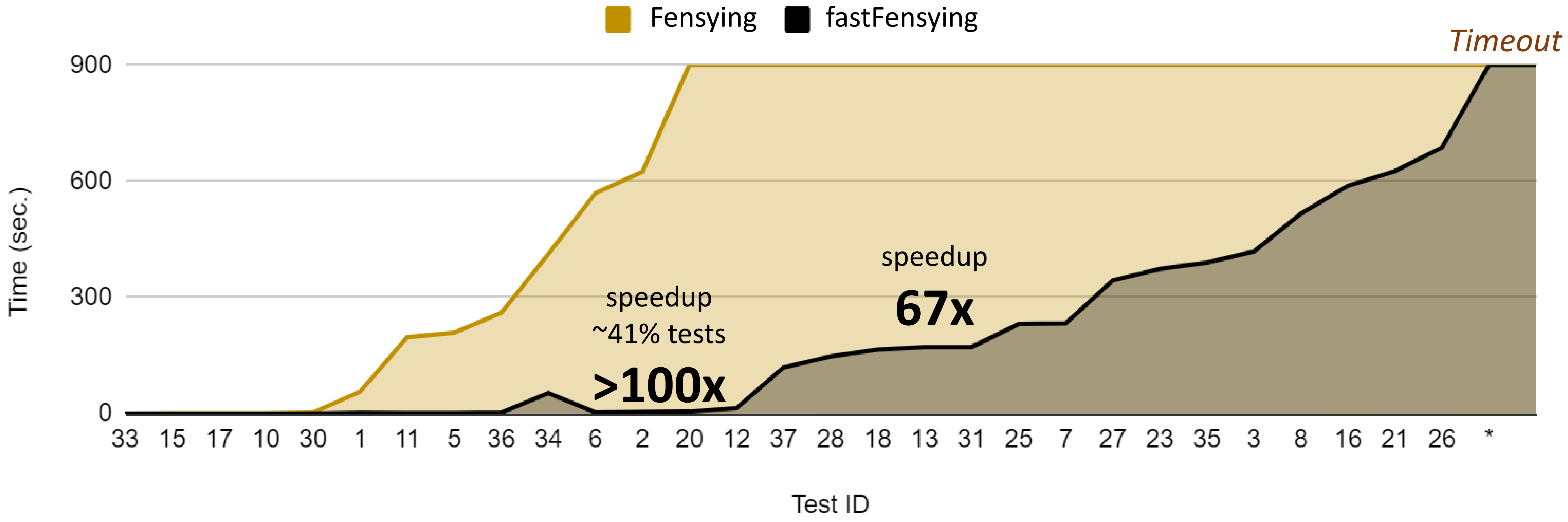
Experiments



* tests that timeout for both Fensying and fastFensying

Benchmarks source: Singh et al., TASE'21, Abdulla at al., PLDI'19, Abdulla at al., OOPSLA'18, Norris & Demsky, OOPSLA'13

Experiments



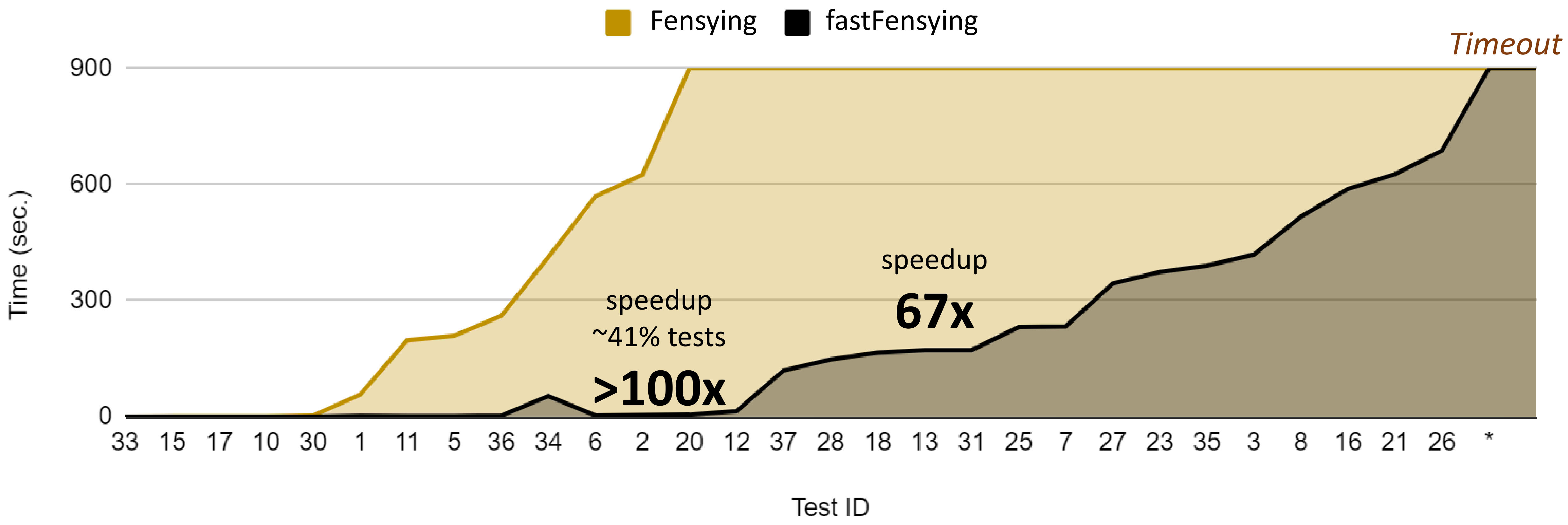
* tests that timeout for both Fensying and fastFensying

Benchmarks source: Singh et al., TASE'21, Abdulla at al., PLDI'19, Abdulla at al., OOPSLA'18, Norris & Demsky, OOPSLA'13

Experiments

fastFensying analysis

≤2 traces for ~85% of tests



* tests that timeout for both Fensying and fastFensying

Benchmarks source: Singh et al., TASE'21, Abdulla at al., PLDI'19, Abdulla at al., OOPSLA'18, Norris & Demsky, OOPSLA'13

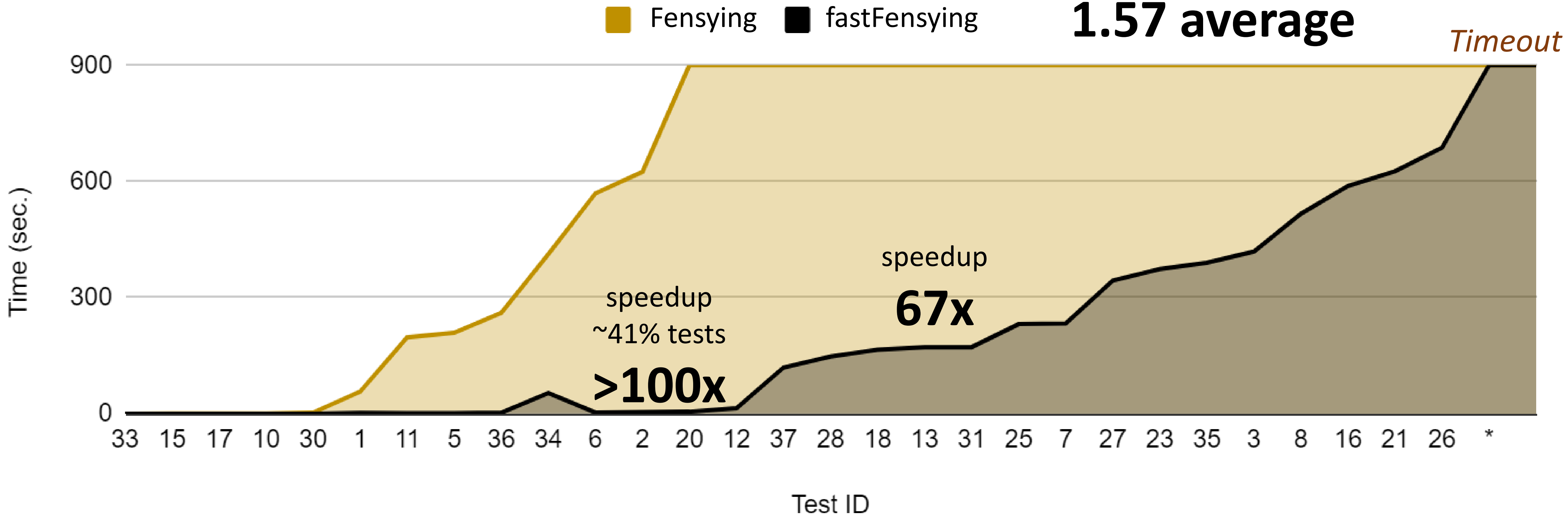
Experiments

non-optimal (fastFensying)

0.005% tests

extra fences (fastFensying)

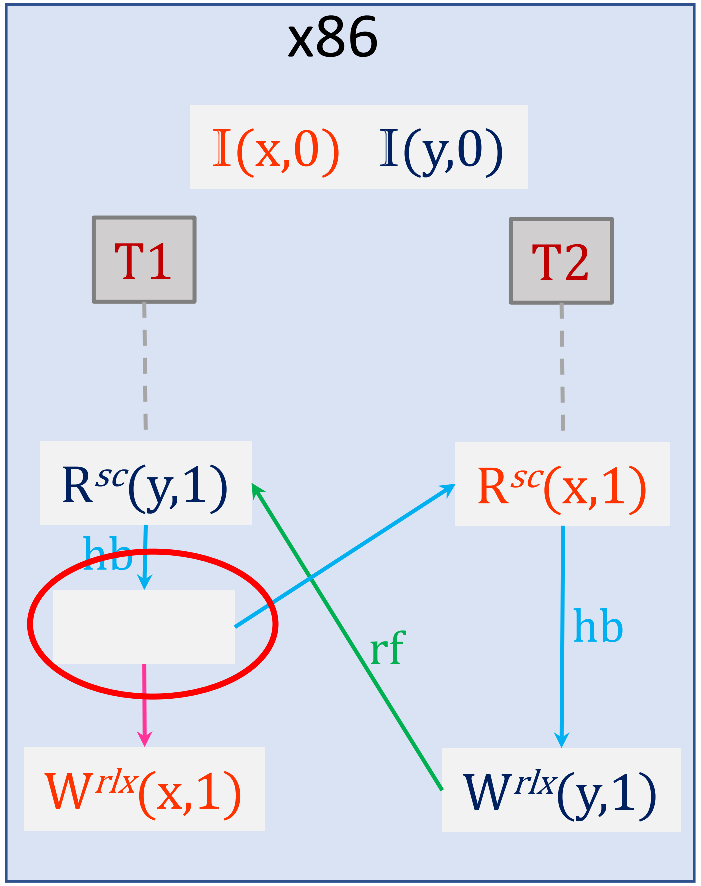
1.57 average

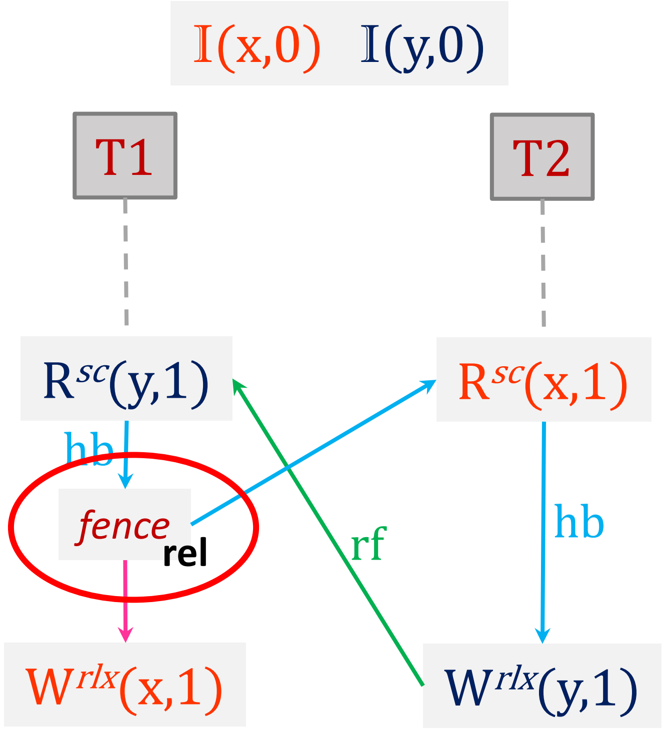


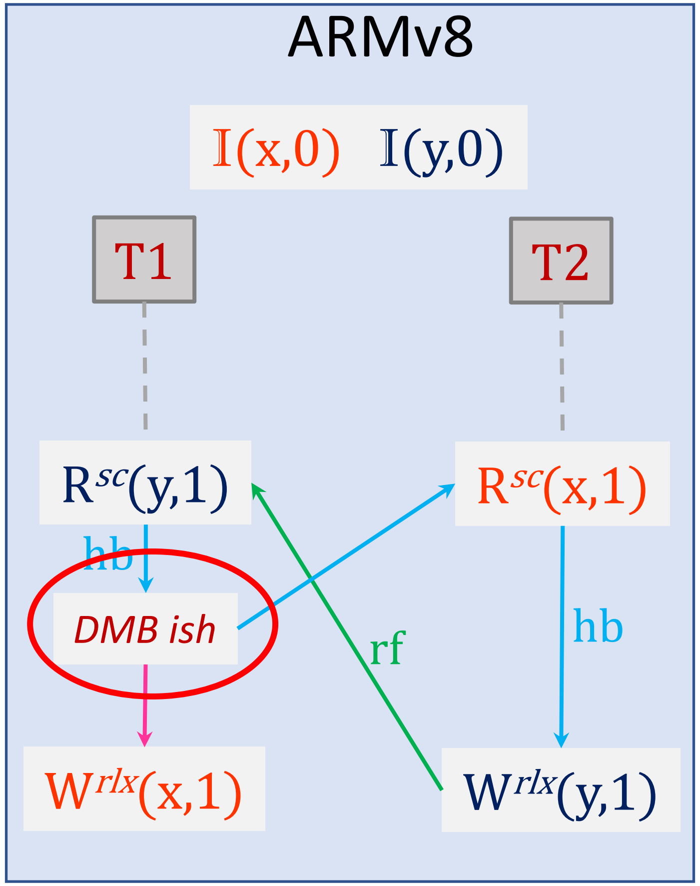
* tests that timeout for both Fensying and fastFensying

Benchmarks source: Singh et al., TASE'21, Abdulla at al., PLDI'19, Abdulla at al., OOPSLA'18, Norris & Demsky, OOPSLA'13

Benefit of portability







(fast)Fensying tool

open source

https://github.com/singhsanjana/fensying

Future Directions

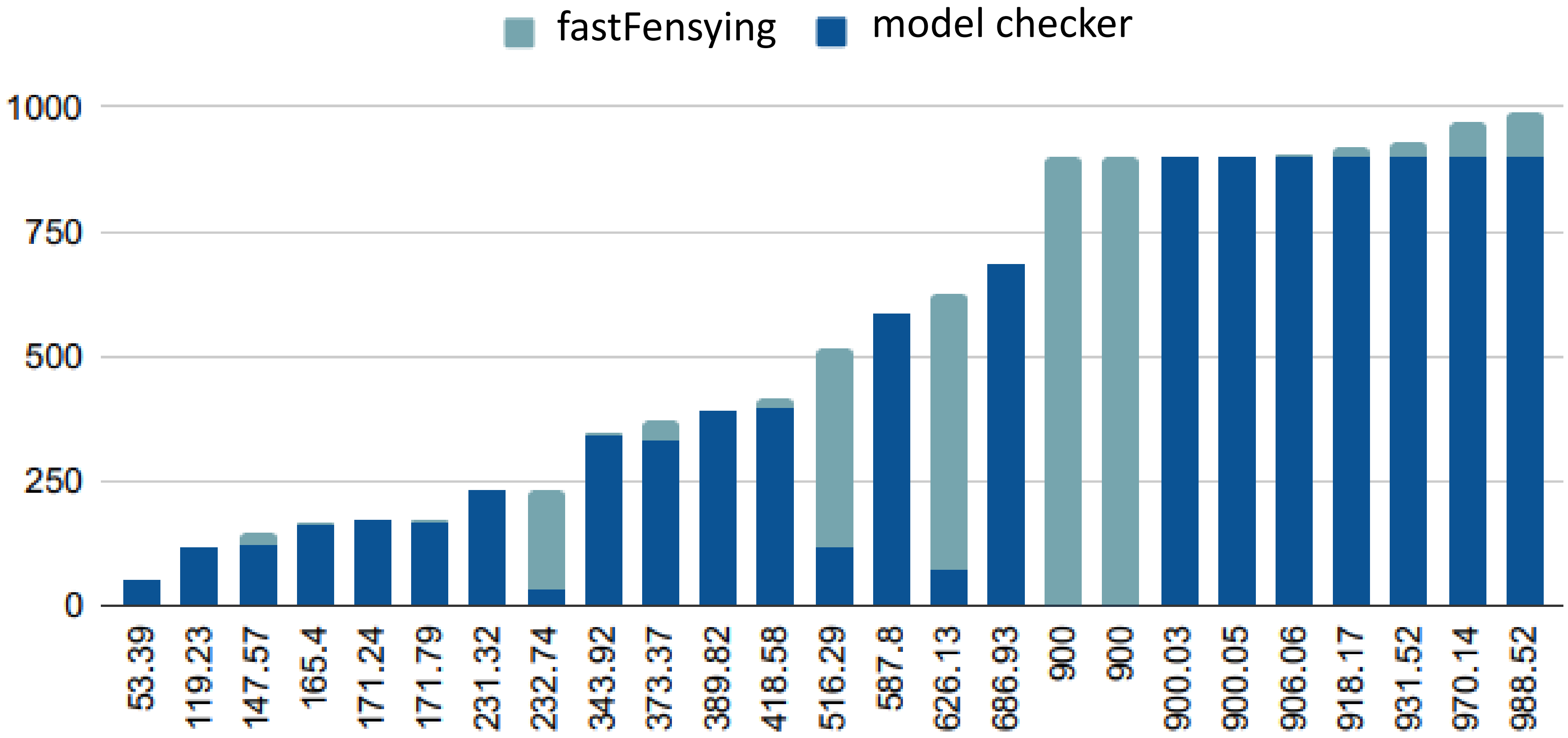
Improve BTG time

Improve fence synthesis time

Future Directions

Improve BTG time

Improve fence synthesis time



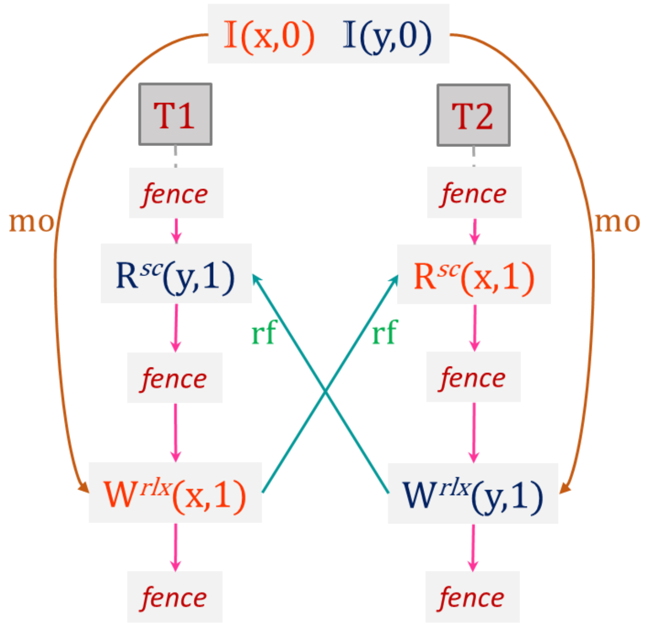
Total Time (fastFensying + model checker)

Future Directions

Improve BTG time

Improve fence synthesis time

Intermediate trace generation → Cycle detection → Min-model finding



Future Directions

Improve BTG time

Improve fence synthesis time

Intermediate trace generation → Cycle detection → Min-model finding

$\mathcal{O}((|\varepsilon|+\mathrm{E}).(\mathrm{C}+1))$

ε: set of events of buggy trace

E: #pairs of events in ε, in $\mathcal{O}(|\varepsilon|^2)$

C: #cycles of buggy trace, in $\mathcal{O}(|\varepsilon|!)$

Thank You

Questions?

Looking for post-doc positions

"*We still do not have an acceptable way to make our informal (since C++14) prohibition of out-of-thin-air results precise. The primary practical effect of that is that formal verification of C++ programs using relaxed atomics remains unfeasible.*

The paper [Lahav et al. PLDI'17] suggests a solution similar to
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3710.html .
We continue to ignore the problem here, but try to stay out of the way of such a solution."

source: https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0668r5.html
(Bullet 4. under 'Revising the C++ memory model')

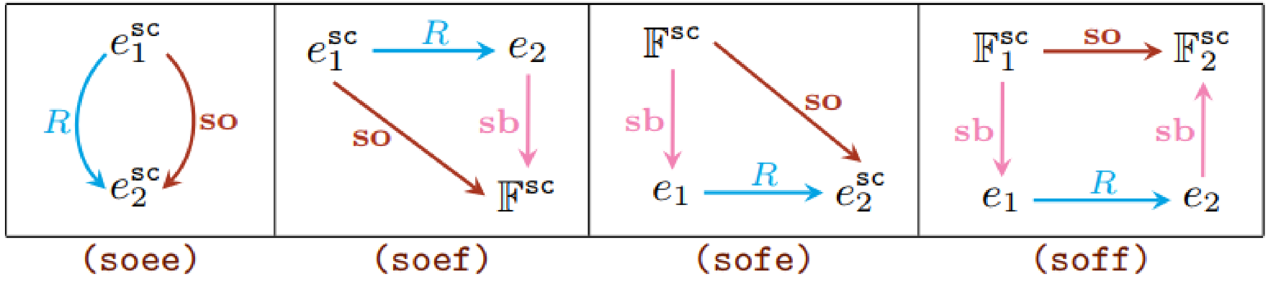
back-intro Back-working

sc-order (so)

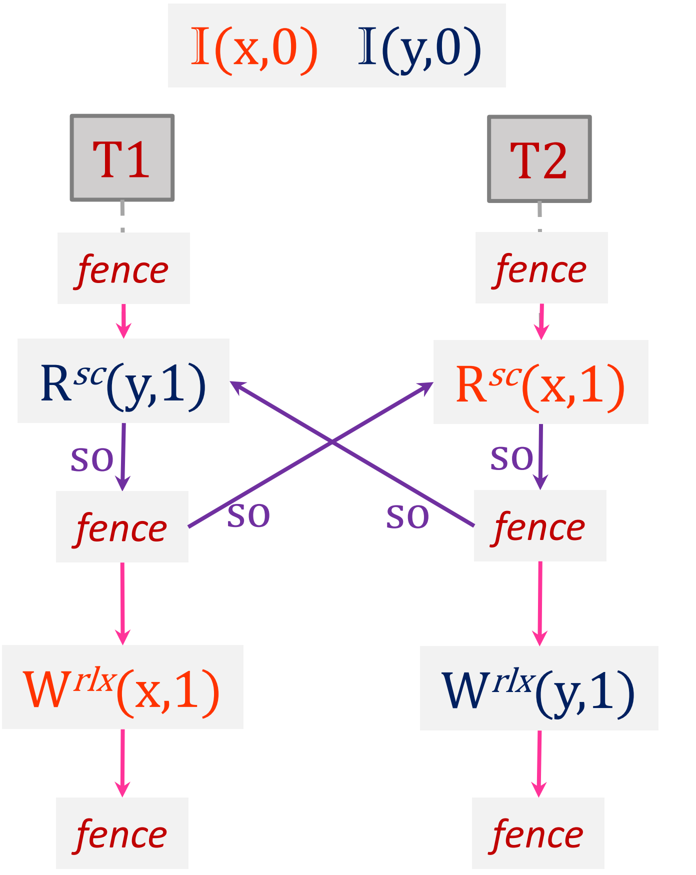
inability to create a total-order

Step 3 detect violations of coherence
(strong-fensying)

introduce ***sc-order*** (so)
cycle in so ⇒ to cannot be formed



$$R = \rightarrow_{\tau}^{\mathrm{hb}} \cup \rightarrow_{\tau}^{\mathrm{mo}} \cup \rightarrow_{\tau}^{\mathrm{rf}} \cup \rightarrow_{\tau}^{\mathrm{fr}}$$



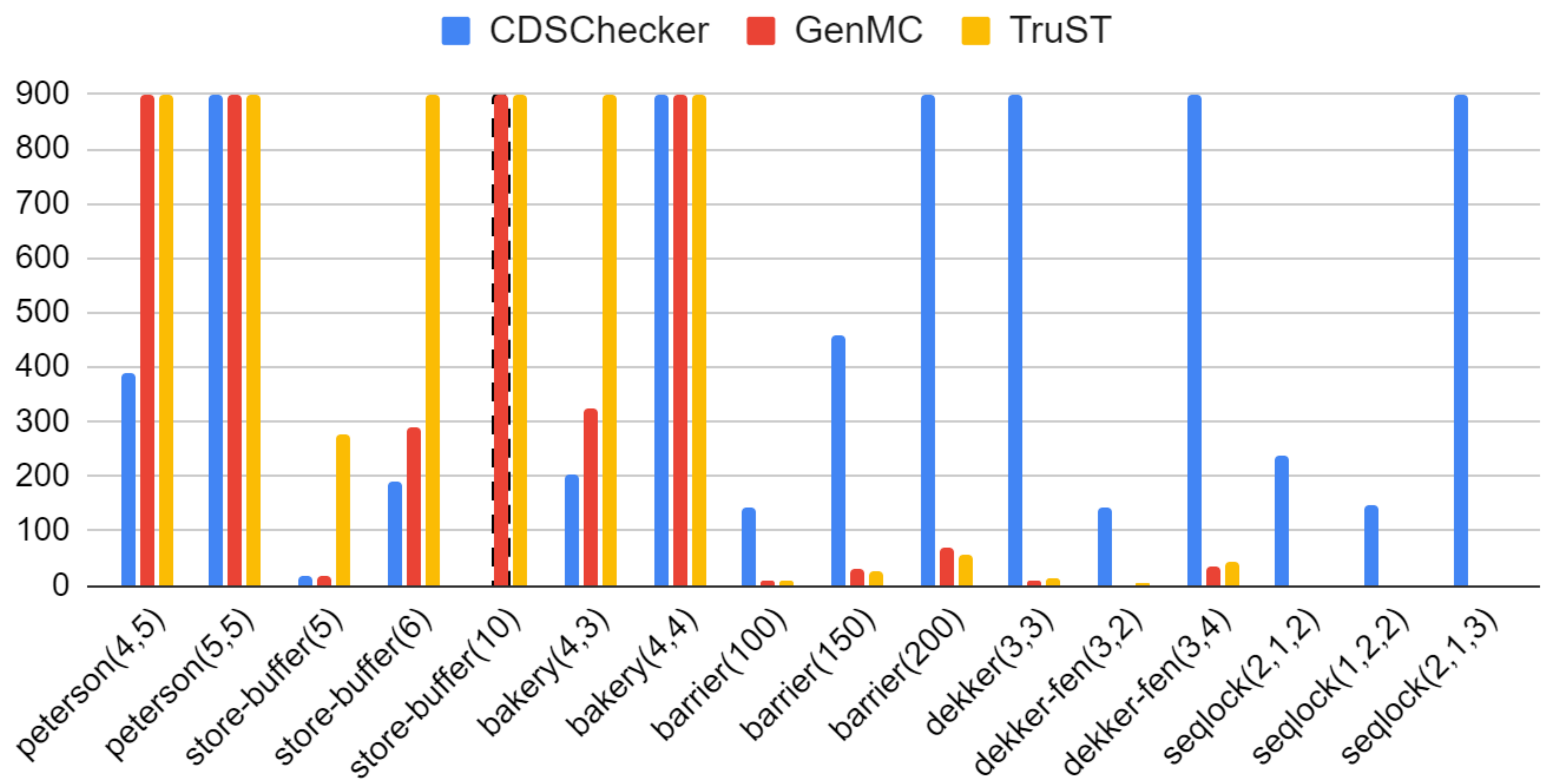
back

assuming seq-cst ordered fences

Alternate BTG

Improve BTG time

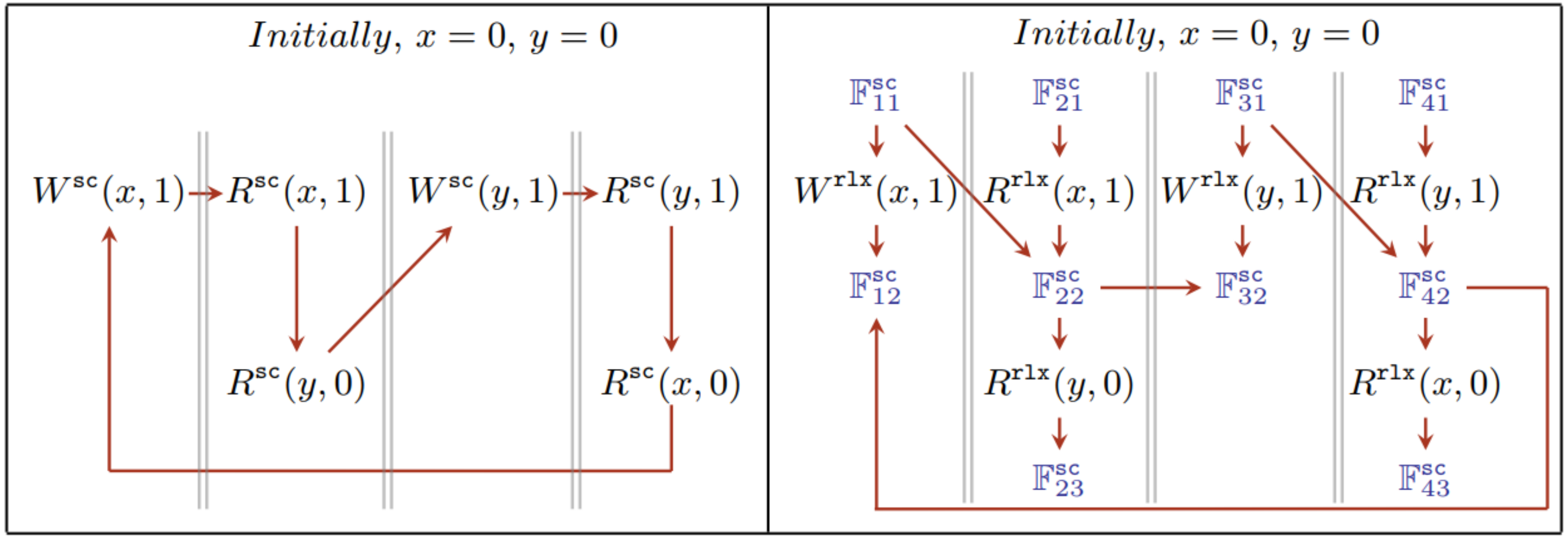
Improve fence synthesis time



back

Fence synthesis vs event strengthening

C11 fences do not restore sequential consistency

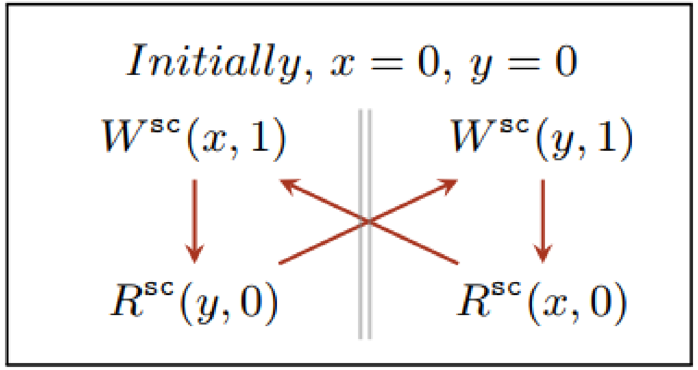


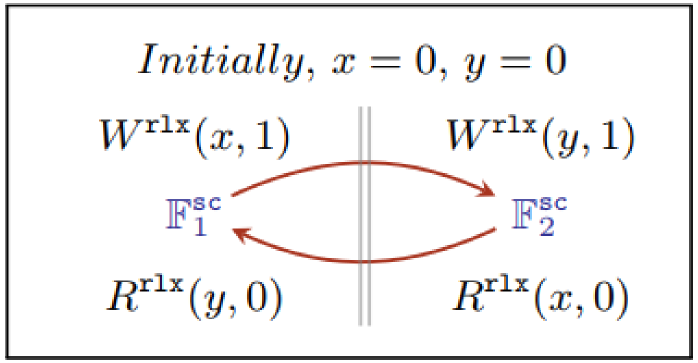
iriw-invalid iriw-valid

top end

Fence synthesis vs event strengthening

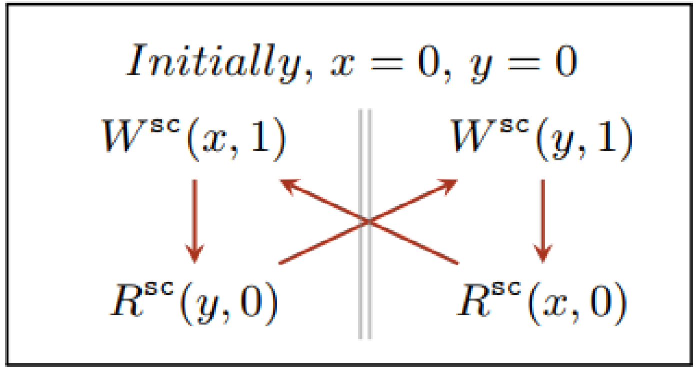
Interpreting barriers from memory orders is not precise

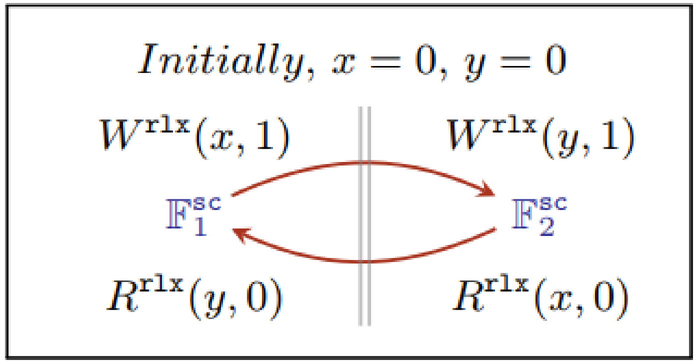




Fence synthesis vs event strengthening

Interpreting barriers from memory orders is not precise





$Initially,\ x = 0,\ y = 0$

DMB ish	DMB ish
$str(x,1)$	$str(y,1)$
DMB ish	DMB ish
$ld(y)$	$ld(x)$
DMB ish	DMB ish

barriers on ARM

$Initially,\ x = 0,\ y = 0$

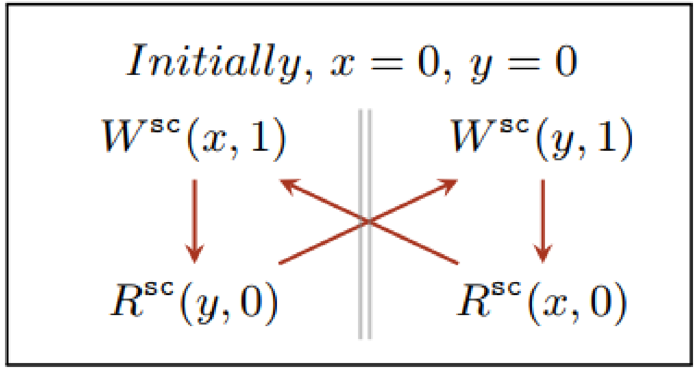
$str(x,1)$	$str(y,1)$
DMB ish	DMB ish
$ld(y)$	$ld(x)$

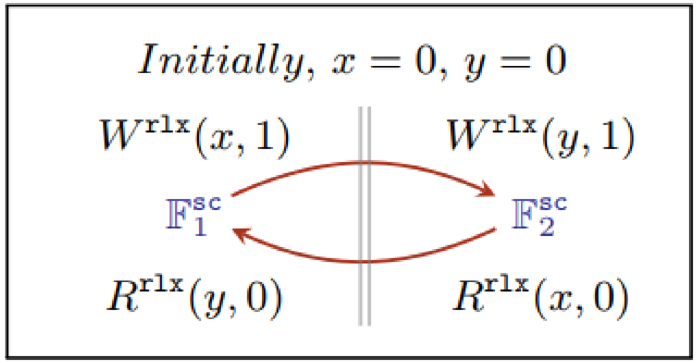
barriers on ARM

top end

Fence synthesis vs event strengthening

Interpreting barriers from memory orders is not precise





Initially, $x = 0,\ y = 0$

`DMB ish`	`DMB ish`
$str(x, 1)$	$str(y, 1)$
`DMB ish`	`DMB ish`
$ld(y)$	$ld(x)$
`DMB ish`	`DMB ish`

barriers on ARM

Initially, $x = 0,\ y = 0$

`hwsync`	`hwsync`
$str(x, 1)$	$str(y, 1)$
`hwsync`	`hwsync`
$ld(y)$	$ld(x)$
`isync+`	`isync+`

barriers on power

Initially, $x = 0,\ y = 0$

$str(x, 1)$	$str(y, 1)$
`DMB ish`	`DMB ish`
$ld(y)$	$ld(x)$

barriers on ARM

Initially, $x = 0,\ y = 0$

$str(x, 1)$	$str(y, 1)$
`hwsync`	`hwsync`
$ld(y)$	$ld(x)$

barriers on power

top end

Soundness

Assuming soundness of SAT solver and cycle detection

Weak-fensying: since all cycles are detected it is sound

Strong-fensying: violation of each of the following is caught as a cycle in so

$$order(P, R) \triangleq (\nexists a\ R(a,a)) \wedge (R^+ \subseteq R) \wedge (R \subseteq P \times P); \text{ and,}$$
$$total(P, R) \triangleq \forall a, b \in P \implies a = b \vee R(a,b) \vee R(b,a).$$

All $\mathtt{sc}$ ordered events must form a total order $\rightarrow_\tau^{\mathbf{to}}$ s.t. the following conditions are satisfied:

1. $order(\mathcal{E}^{(\mathtt{sc})}, \rightarrow_\tau^{\mathbf{to}}) \wedge total(\mathcal{E}^{(\mathtt{sc})}, \rightarrow_\tau^{\mathbf{to}}) \wedge \rightarrow_\tau^{\mathbf{hb}}|_{\mathtt{sc}} \cup \rightarrow_\tau^{\mathbf{mo}}|_{\mathtt{sc}} \subseteq \rightarrow_\tau^{\mathbf{to}}$ (coto)
2. $\forall e_w \rightarrow_\tau^{\mathbf{rf}} e_r$ s.t. $e_r \in \mathcal{E}_\tau^{(\mathtt{sc})}$
   - either, $e_w \in \mathcal{E}_\tau^{(\mathtt{sc})} \wedge \mathtt{imm\text{-}scr}(\tau, e_w, e_r)$. (rfto1)
   - or, $e_w \notin \mathcal{E}_\tau^{(\mathtt{sc})} \wedge \nexists e'_w \in \mathcal{E}_\tau^{\mathbb{W}(\mathtt{sc})}$ s.t. $e_w \rightarrow_\tau^{\mathbf{hb}} e'_w \wedge \mathtt{imm\text{-}scr}(\tau, e'_w, e_r)$. (rfto2)

   where, $\mathtt{imm\text{-}scr}(\tau, a, b) \triangleq a \in \mathcal{E}_\tau^{\mathbb{W}(\mathtt{sc})}, b \in \mathcal{E}_\tau^{\mathbb{R}(\mathtt{sc})}, a \rightarrow_\tau^{\mathbf{to}} b$ and $obj(a) = obj(b)$ $\wedge \nexists c \in \mathcal{E}_\tau^{\mathbb{W}(\mathtt{sc})}$ s.t. $obj(c) = obj(a) \wedge a \rightarrow_\tau^{\mathbf{to}} c \rightarrow_\tau^{\mathbf{to}} b$.
3. $\forall e_w \rightarrow_\tau^{\mathbf{rf}} e_r$ s.t. $e_w \in \mathcal{E}_\tau^{(\mathtt{sc})}, \exists \mathbb{F} \in \mathcal{E}_\tau^{\mathbb{F}(\mathtt{sc})}$ s.t. $\mathbb{F} \rightarrow_\tau^{\mathbf{sb}} e_r \wedge e_w \rightarrow_\tau^{\mathbf{to}} \mathbb{F} \wedge \nexists e'_w \in \mathcal{E}^{\mathbb{W}(\mathtt{sc})}$ where $e_w \rightarrow_\tau^{\mathbf{to}} e'_w \rightarrow_\tau^{\mathbf{to}} \mathbb{F}$. (frfto)

[Vafeiadis et al., POPL 2015]

back

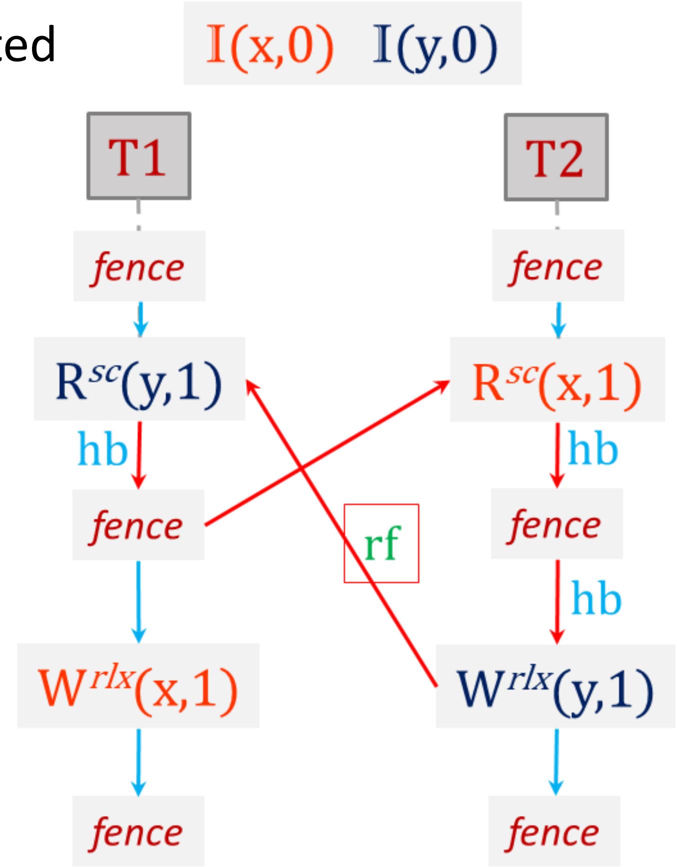
Optimality

Assuming soundness of SAT solver and cycle detection

Optimal solution is not in SAT Query

$$\boxed{fence_1 \wedge fence_2} \quad \vee \quad \boxed{\cancel{fence_1}}$$

⇒ Cycle not detected

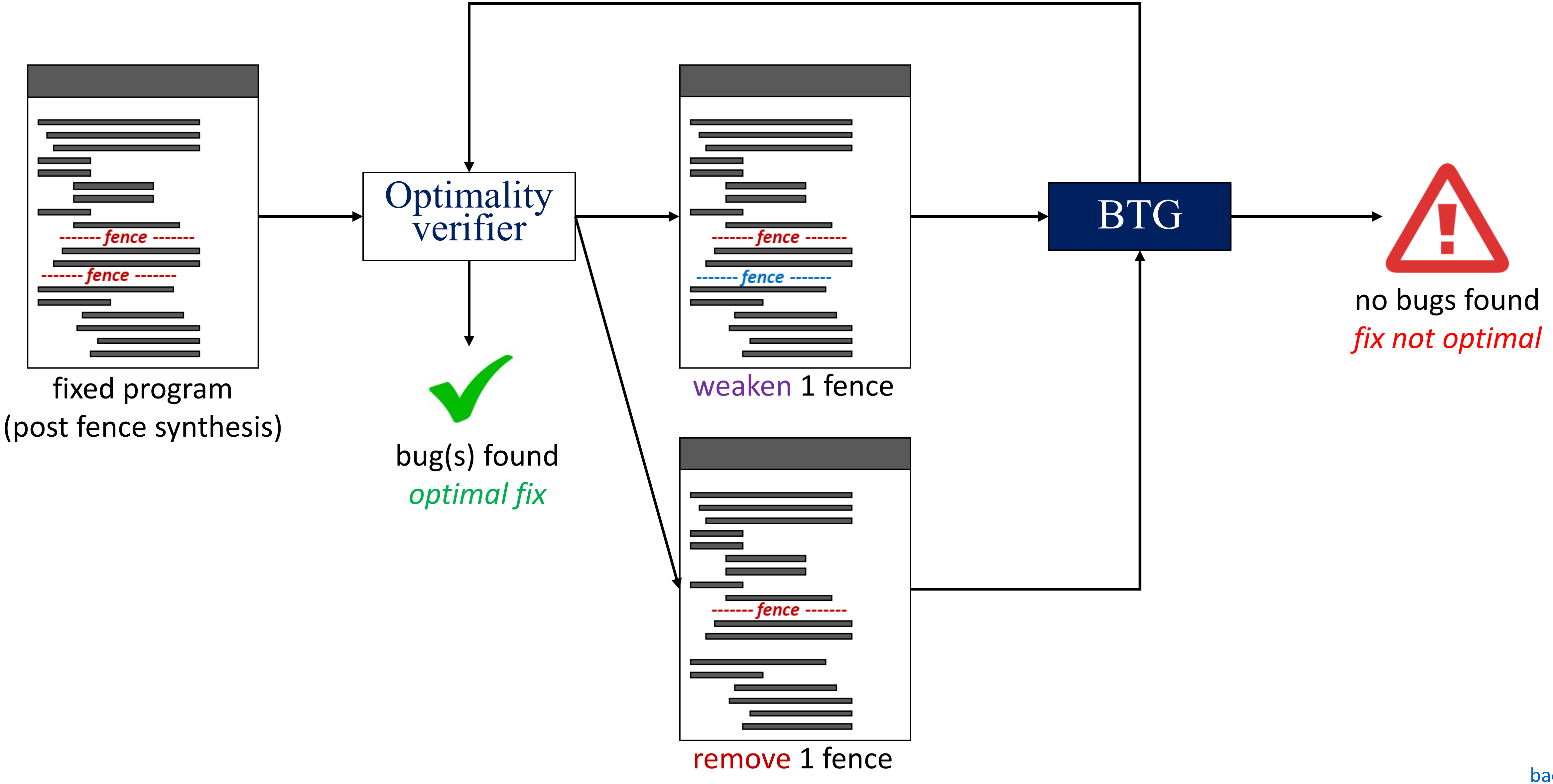


But, fences at all locations

⇒ all cycles formed

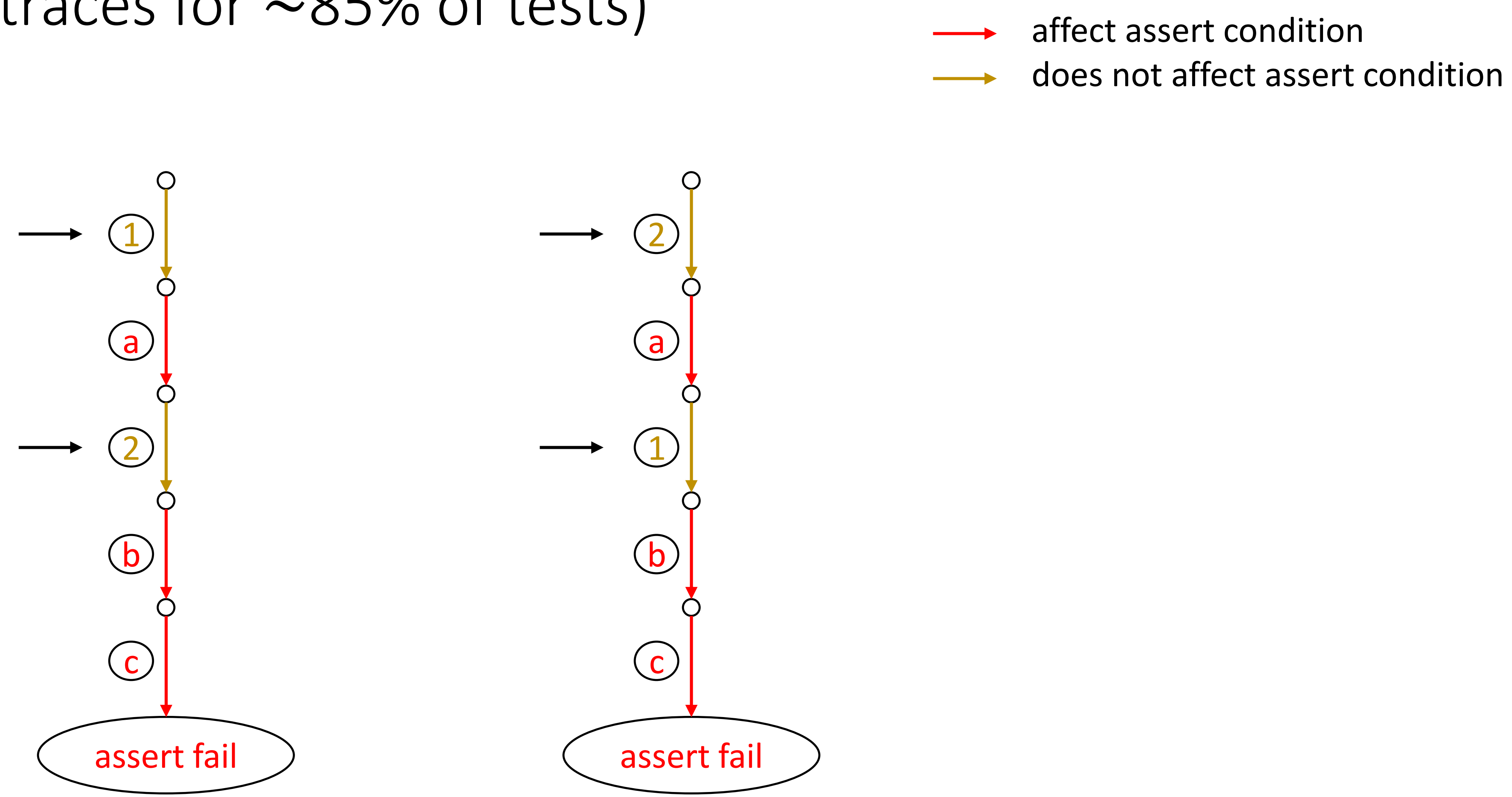
back

Verifying optimality



back

Reason (≤2 traces for ~85% of tests)



buggy trace 1

buggy trace 2

back