ATVA 2022

Fence Synthesis under the C11 Memory Model

Sanjana Singh, Divyanjali Sharma and Subodh Sharma

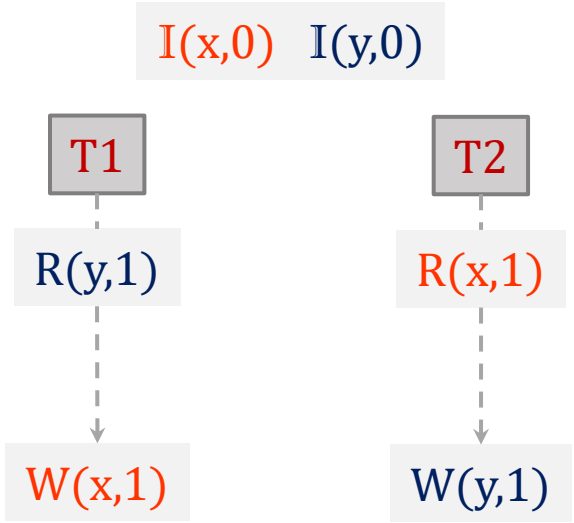
(Indian Institute of Technology Delhi, India)

Ishita Jaju

(Uppsala University, Sweden)

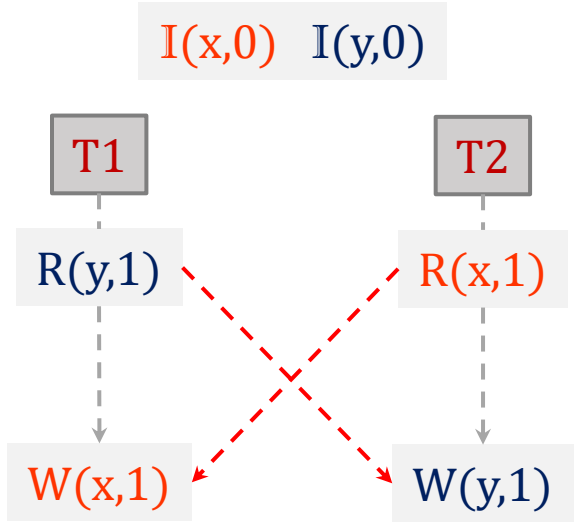
Ordering with fences

- Order might be critical for correctness



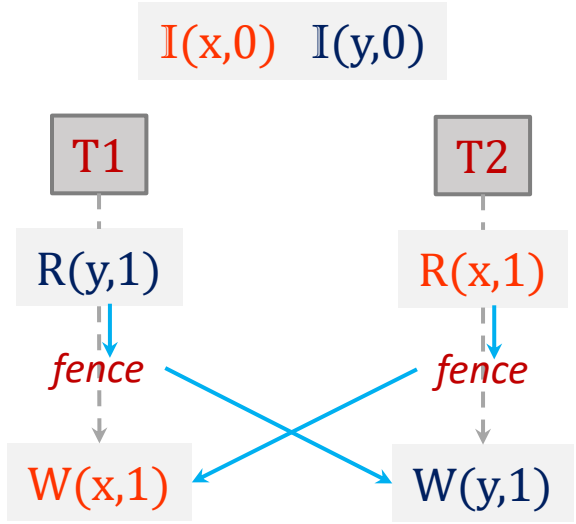
Ordering with fences

- Order might be critical for correctness

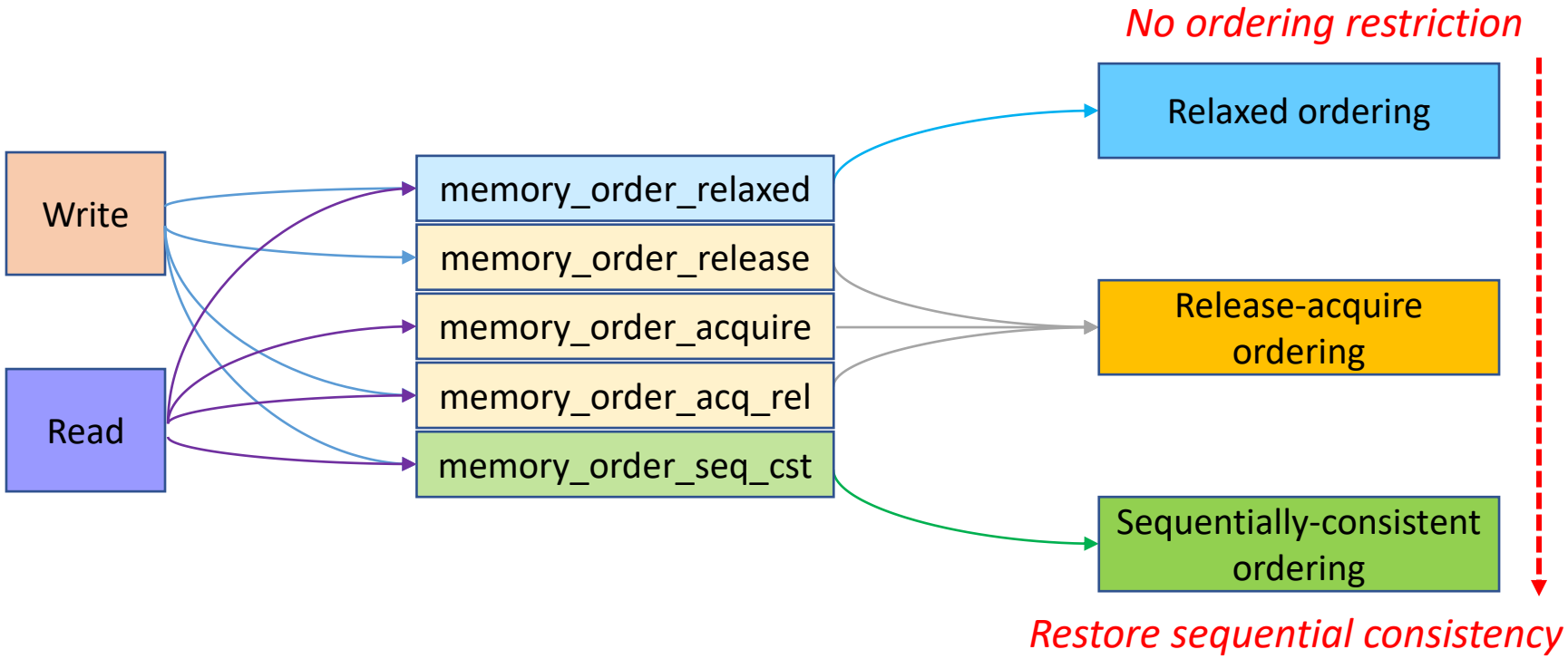


Ordering with fences

- Order might be critical for correctness
- Fences restore order



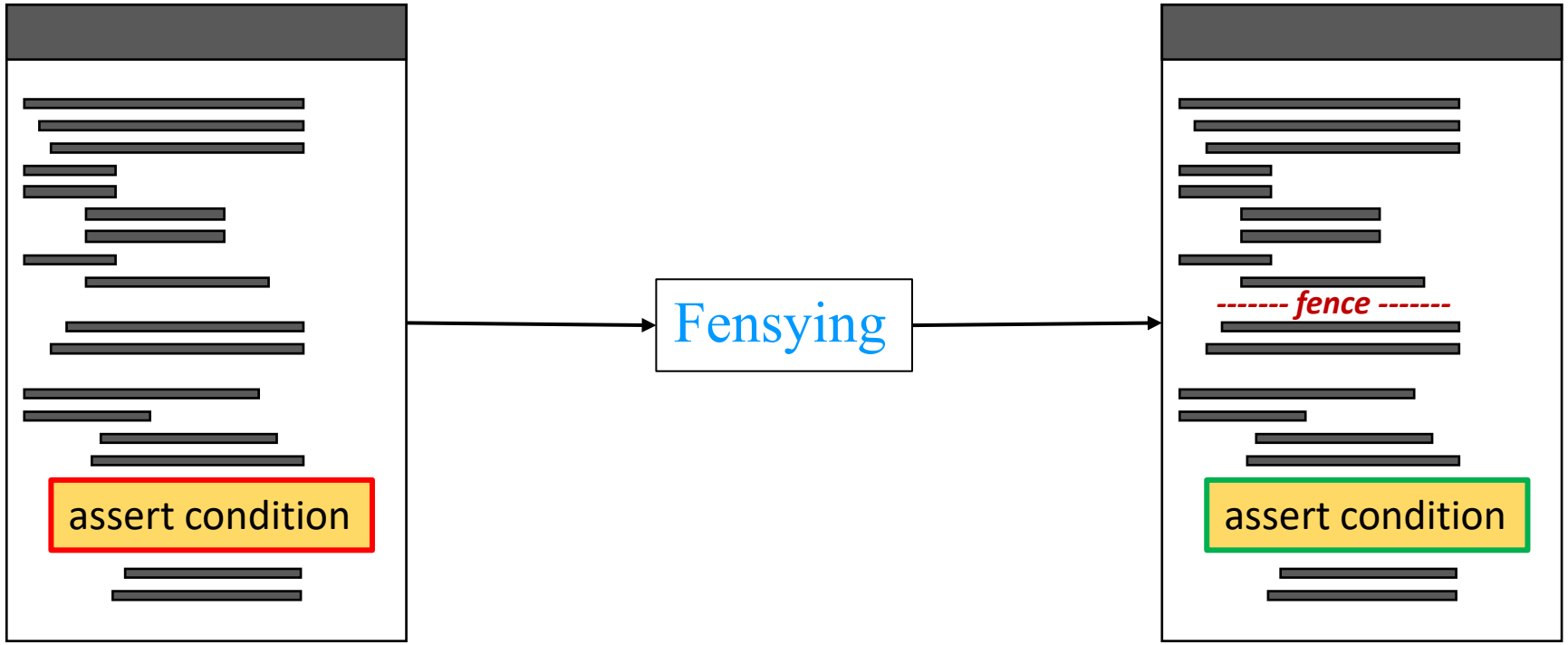
C/C++11 (C11) memory orders



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

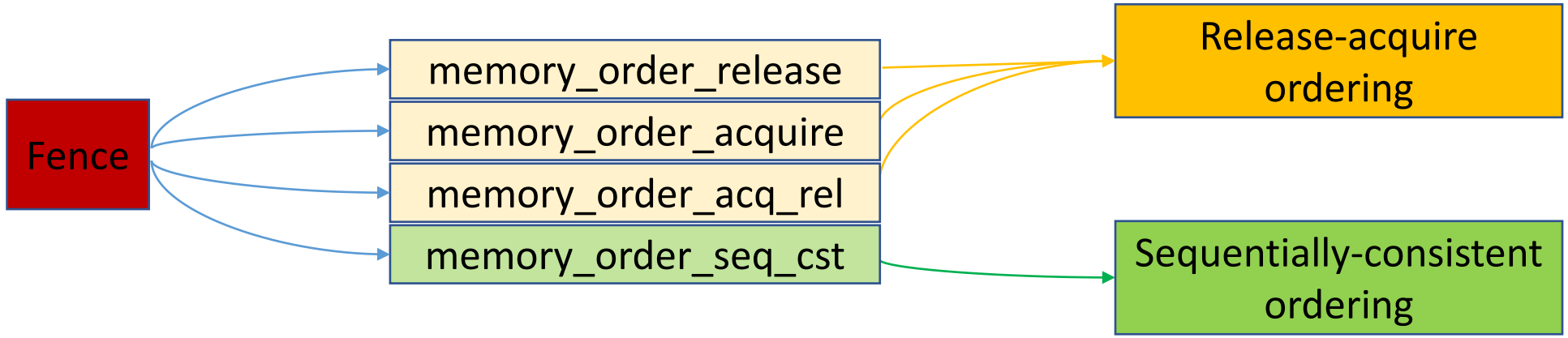
Oberhauser et al., ASPLOS'21

Fence synthesis for automated fix



C11 fences

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



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

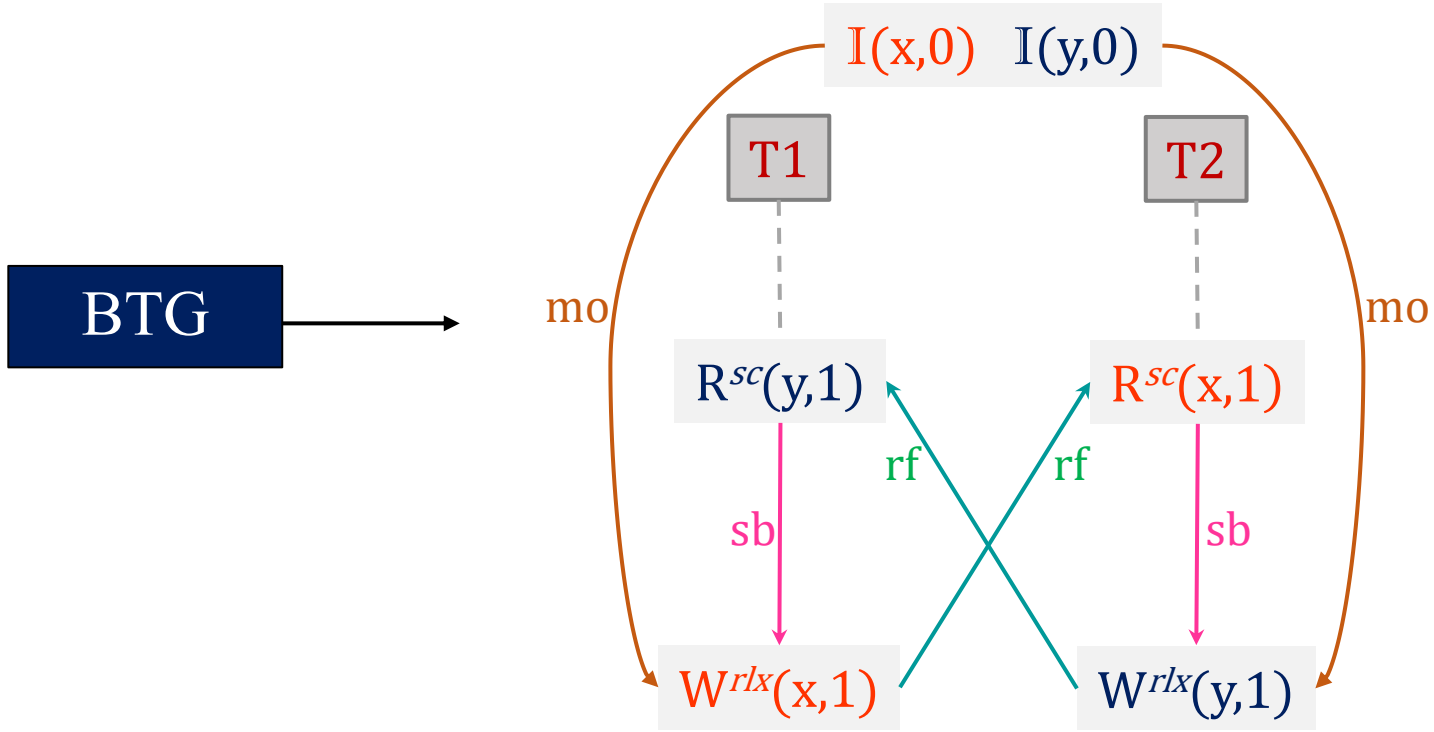
- Smallest set of fences
- Weakest type of fences

solution not unique

Fensying technique

sb sequenced-before
rf reads-from
mo modification-order

Step 1 get the set of buggy traces

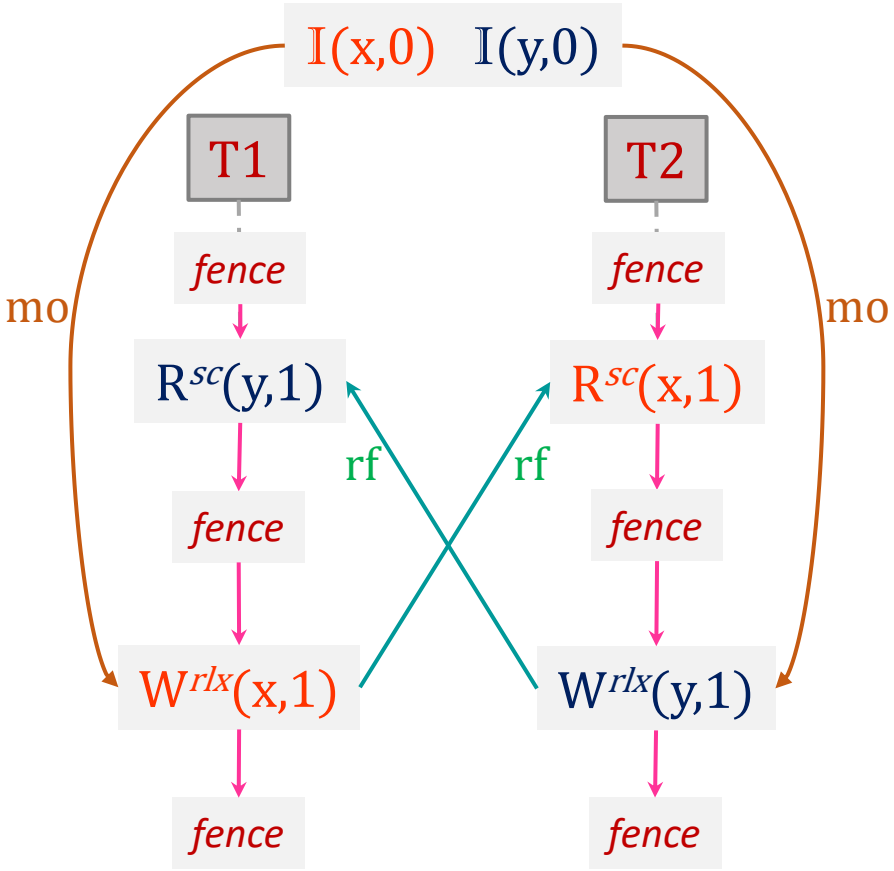


buggy trace generator (BTG): CDSChecker, open source SMC [Norris and Demsky, OOPSLA'13]

Fensying technique

sb sequenced-before
rf reads-from
mo modification-order

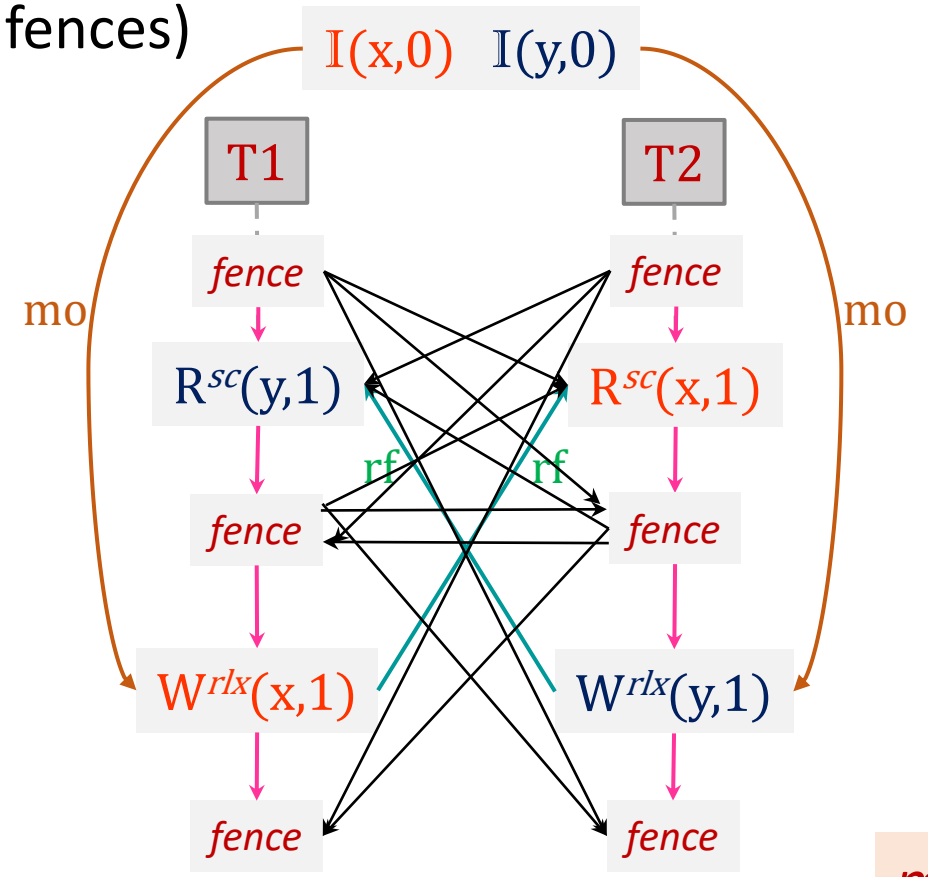
Step 2 generate *intermediate* trace



Fensying technique

sb sequenced-before
rf reads-from
mo modification-order

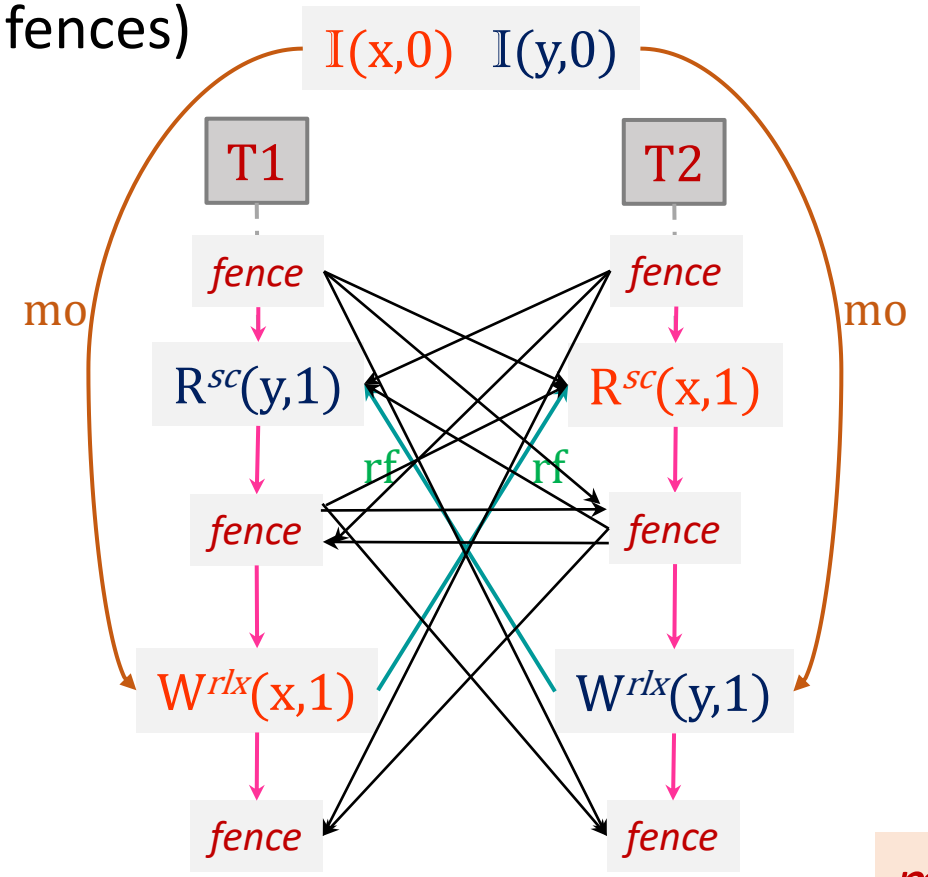
Step 2 generate *intermediate* trace (additional ordering with fences)

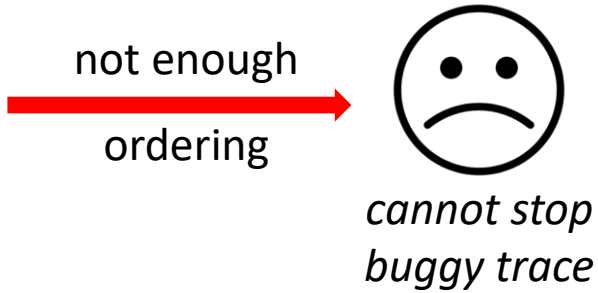


maximum possible fence ordering

Fensying technique

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

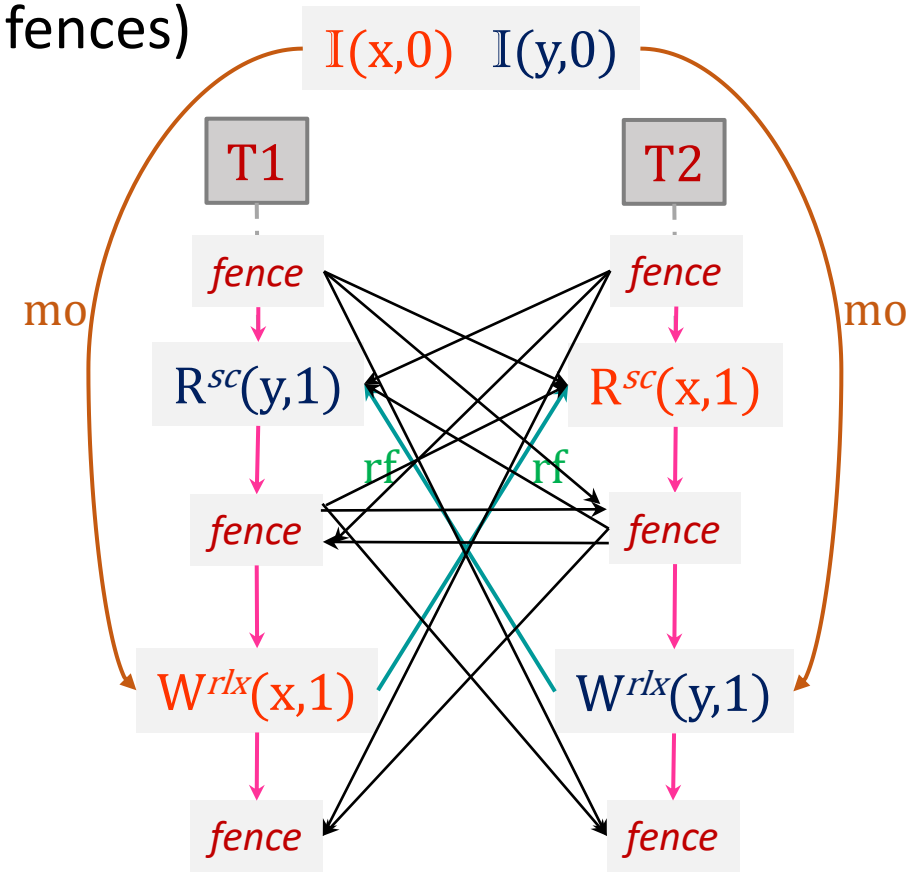




maximum possible fence ordering

Fensying technique

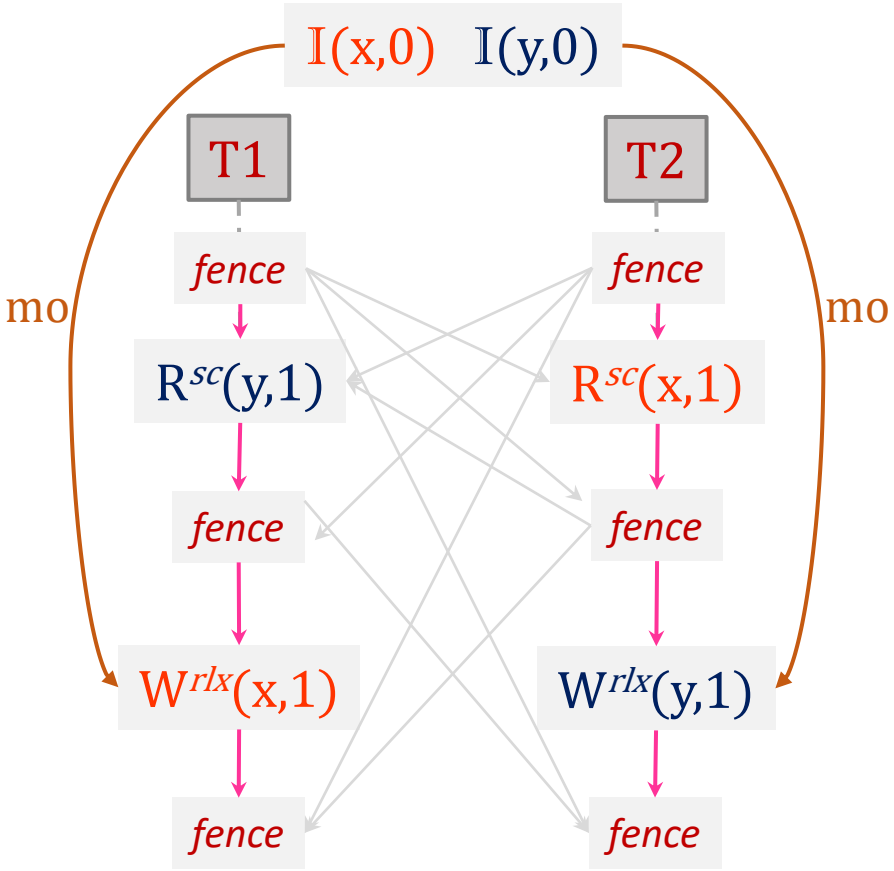
Step 2 generate *intermediate* trace (additional ordering with fences)



enough ordering ?

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

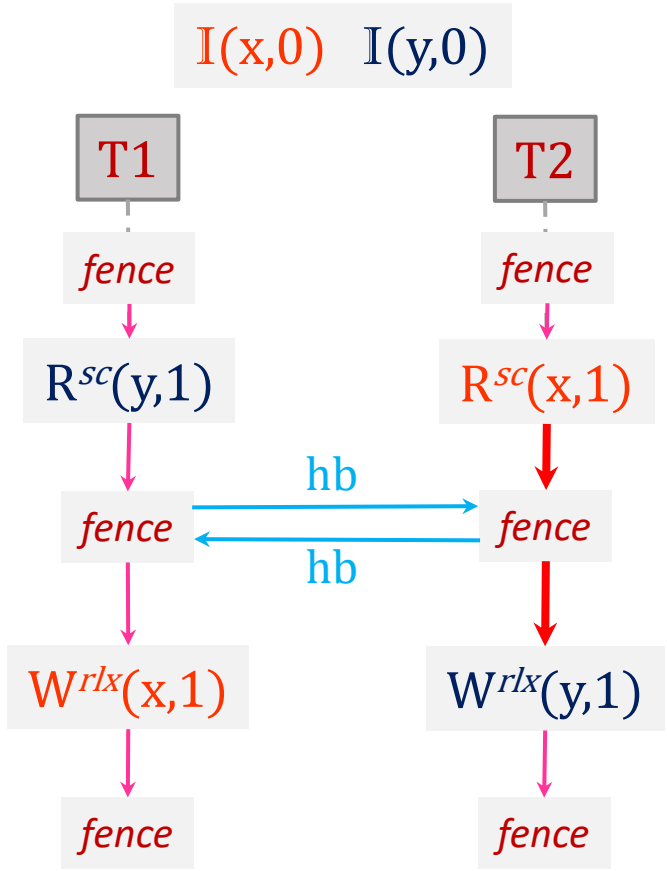
mo; hb; rfinv is irreflexive

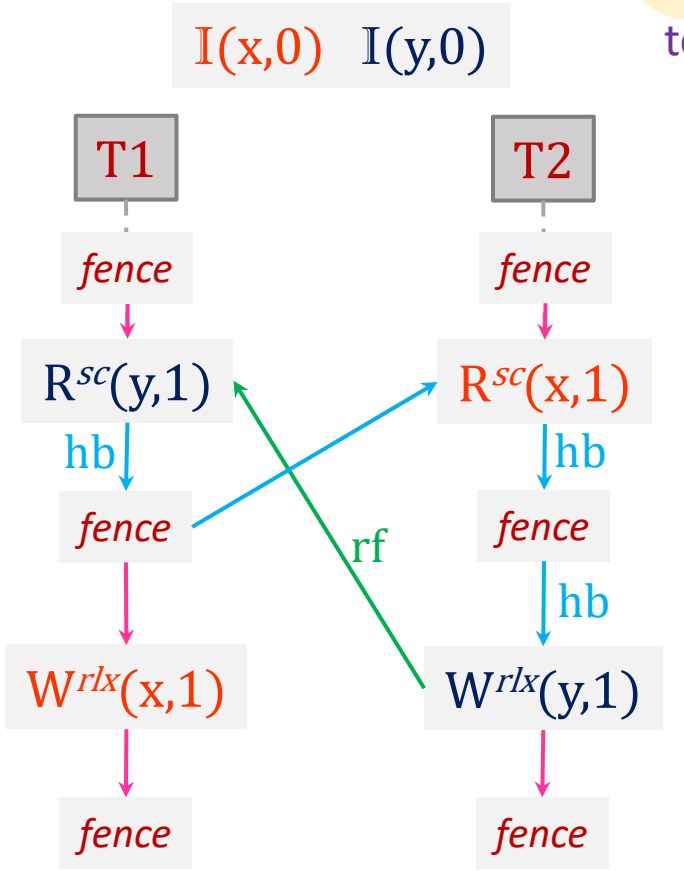
mo; rf; hb; rfinv is irreflexive

to on sc events

Fensying technique

Step 3 detect violations of coherence (*weak-fensying*)





C11 coherence conditions:

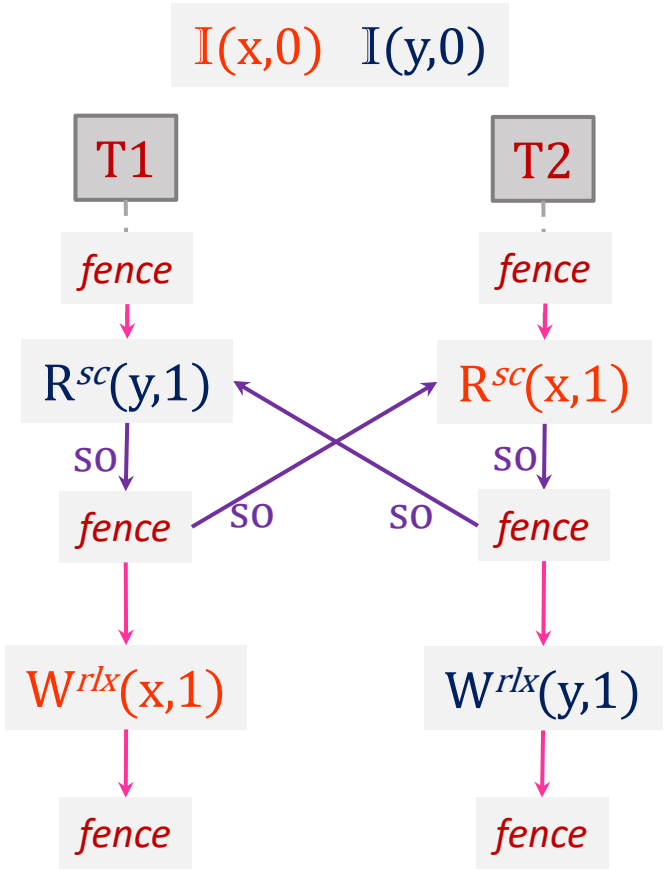
hb is irreflexive
rf; hb is irreflexive
mo;hb is irreflexive
mo; rf; hb is irreflexive
mo; hb; rfinv is irreflexive
mo; rf; hb; rfinv is irreflexive
to on sc events

Johnson's algorithm for cycle detection
[Johnson, D.B, SICOMP'1975]

assuming acq-rel ordered fences

Fensying technique

Step 3 detect violations of coherence (*strong-fensying*)



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
to on sc events

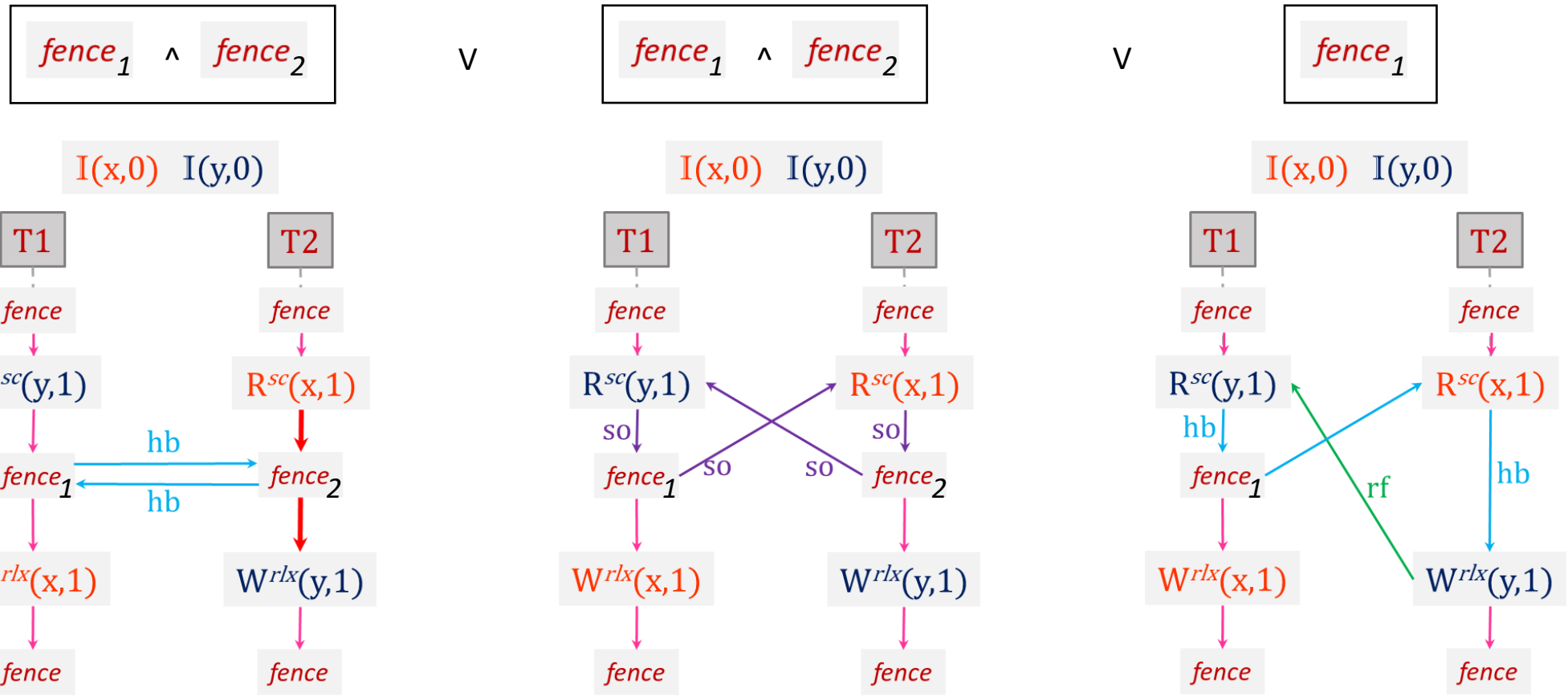
introduce ***sc-order*** (so)
cycle in so ⇒ violation of to

assuming seq-cst ordered fences

Fensying technique

Step 4 find the smallest set of fences

min-model of a SAT query

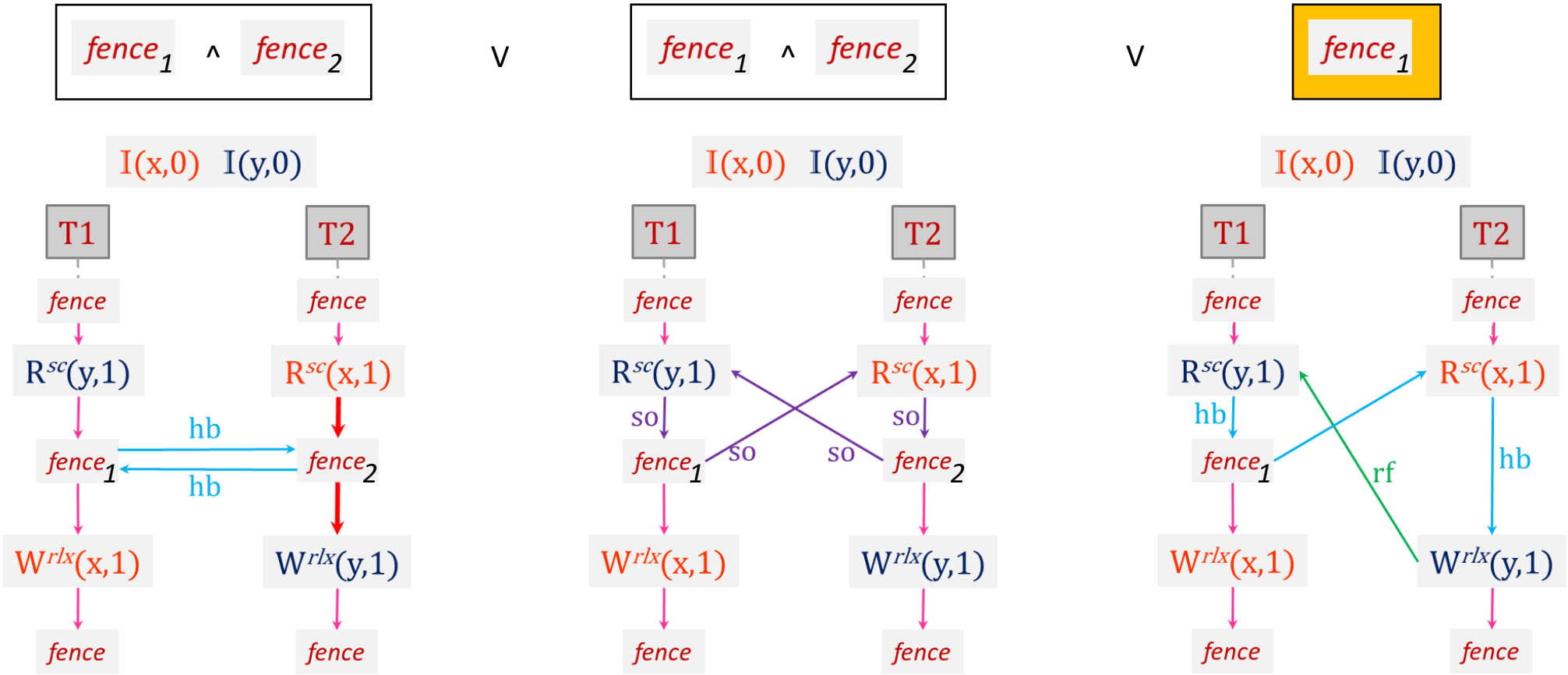


Fensying technique

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

Step 4 find the smallest set of fences

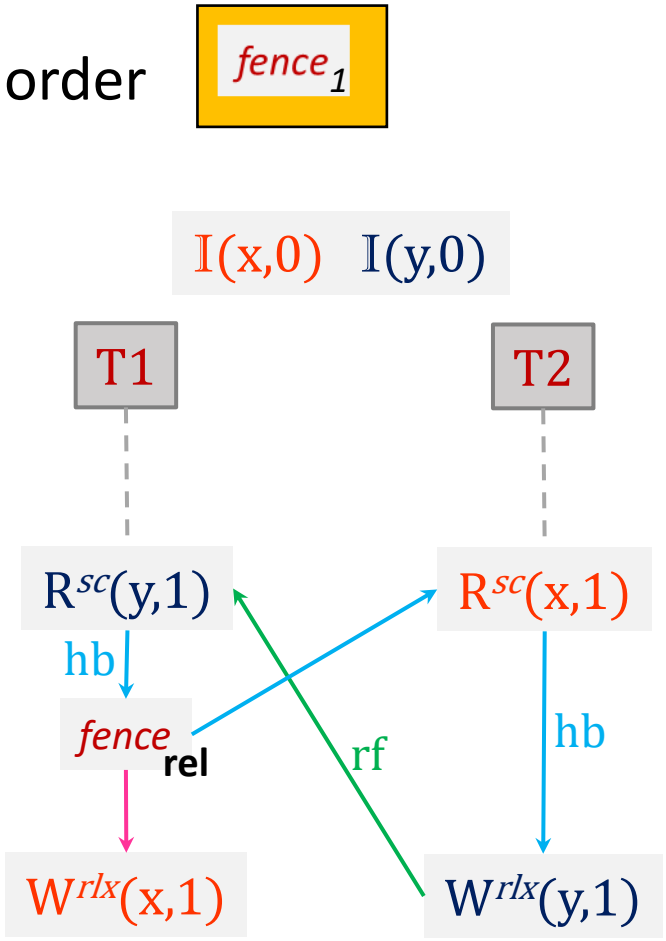
min-model of a SAT query



Fensying technique

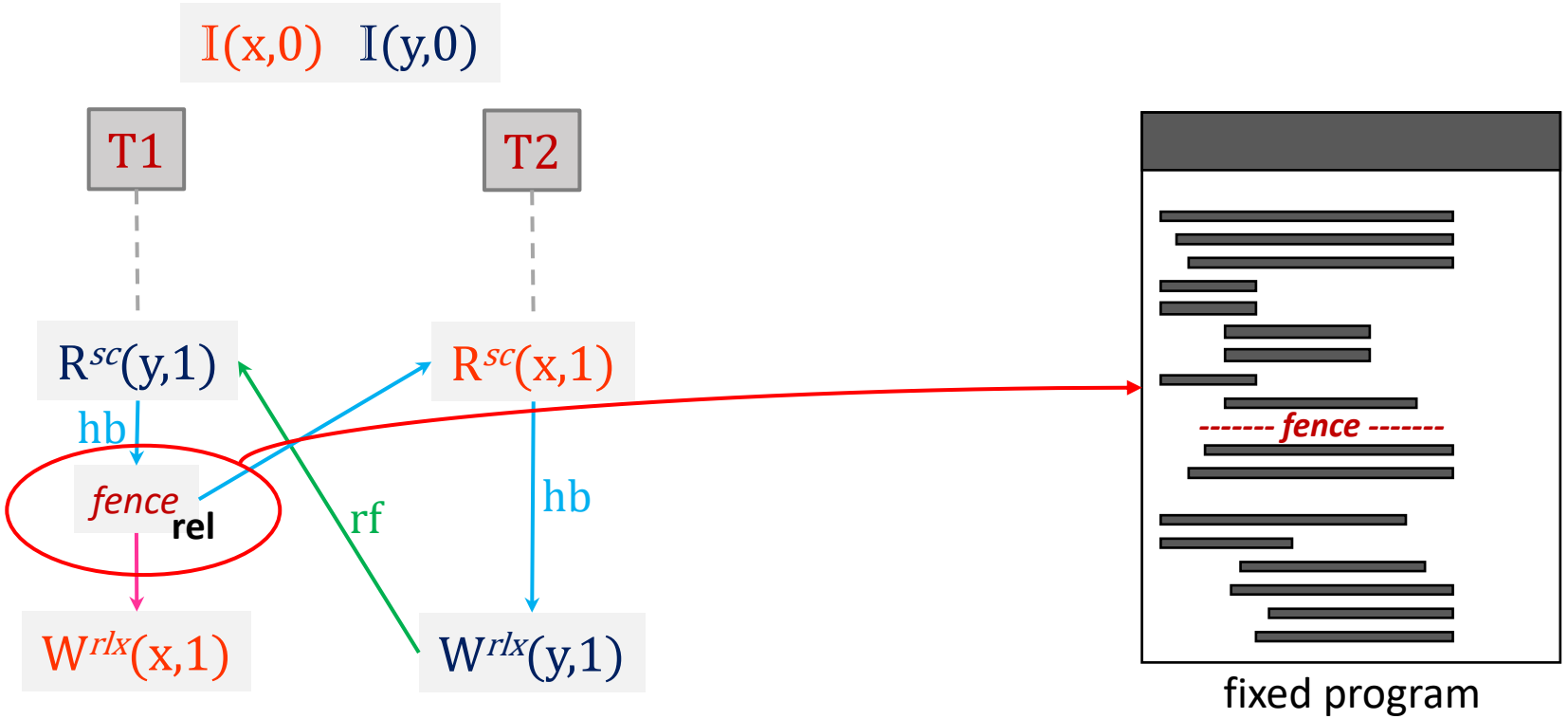
Optimal fence synthesis
- Smallest set of fences ✓
- Weakest type of fences ✓

Step 5 find weakest order



Fensying technique

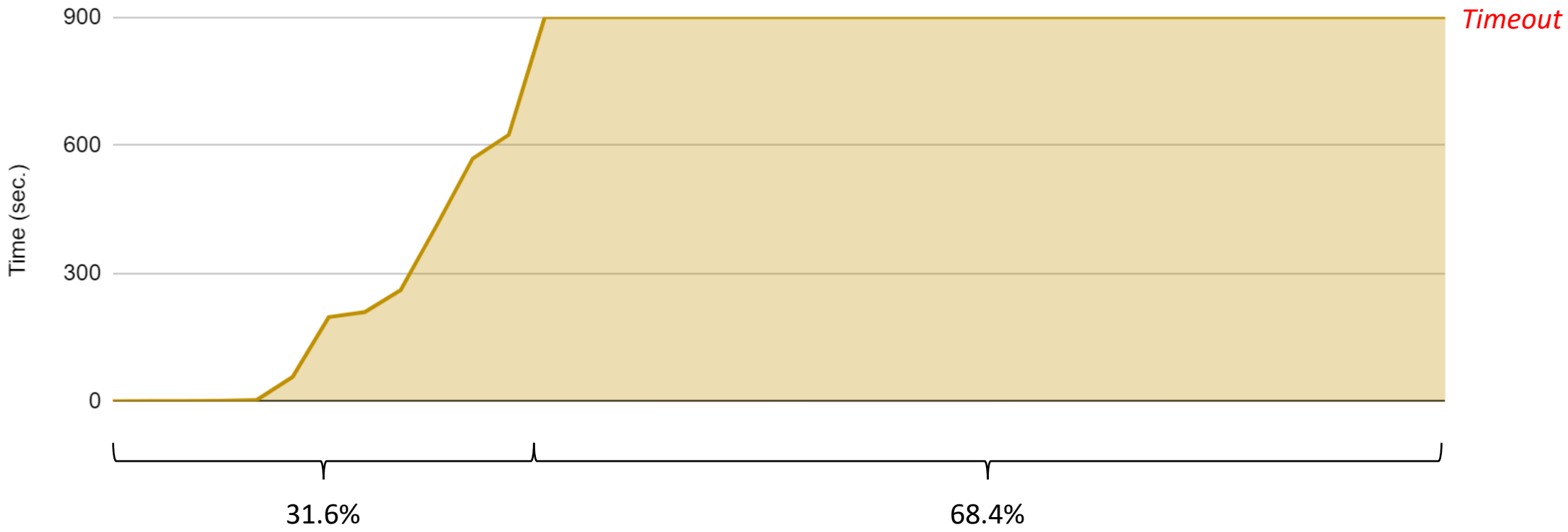
Step 5 find weakest order $fence_1$



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

fFensying: *near*-Optimal C11 fence synthesis

(fast-Fensying)

Fensying

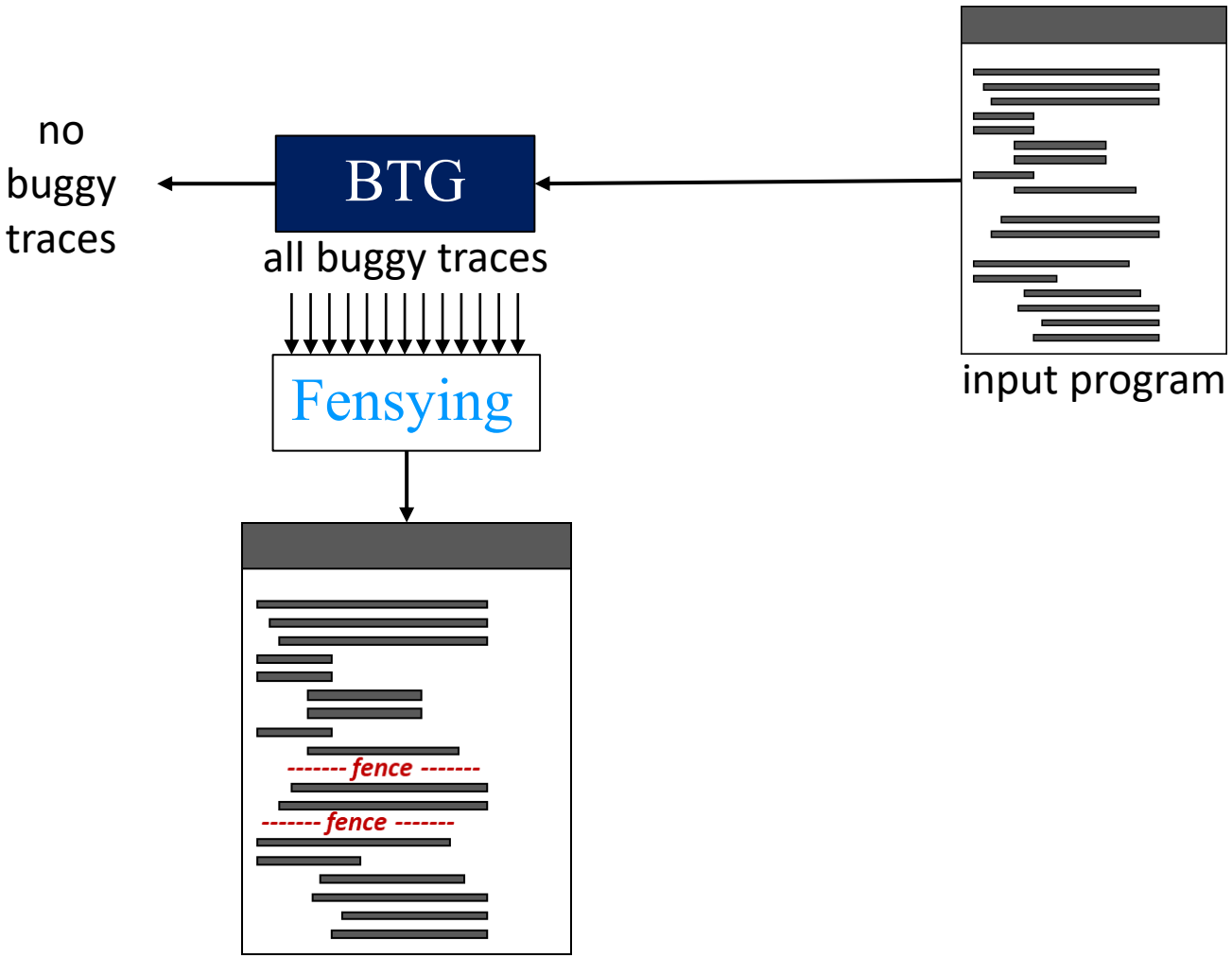
- Sound
- Optimal
- Slow
- Doesn't scale

fFensying

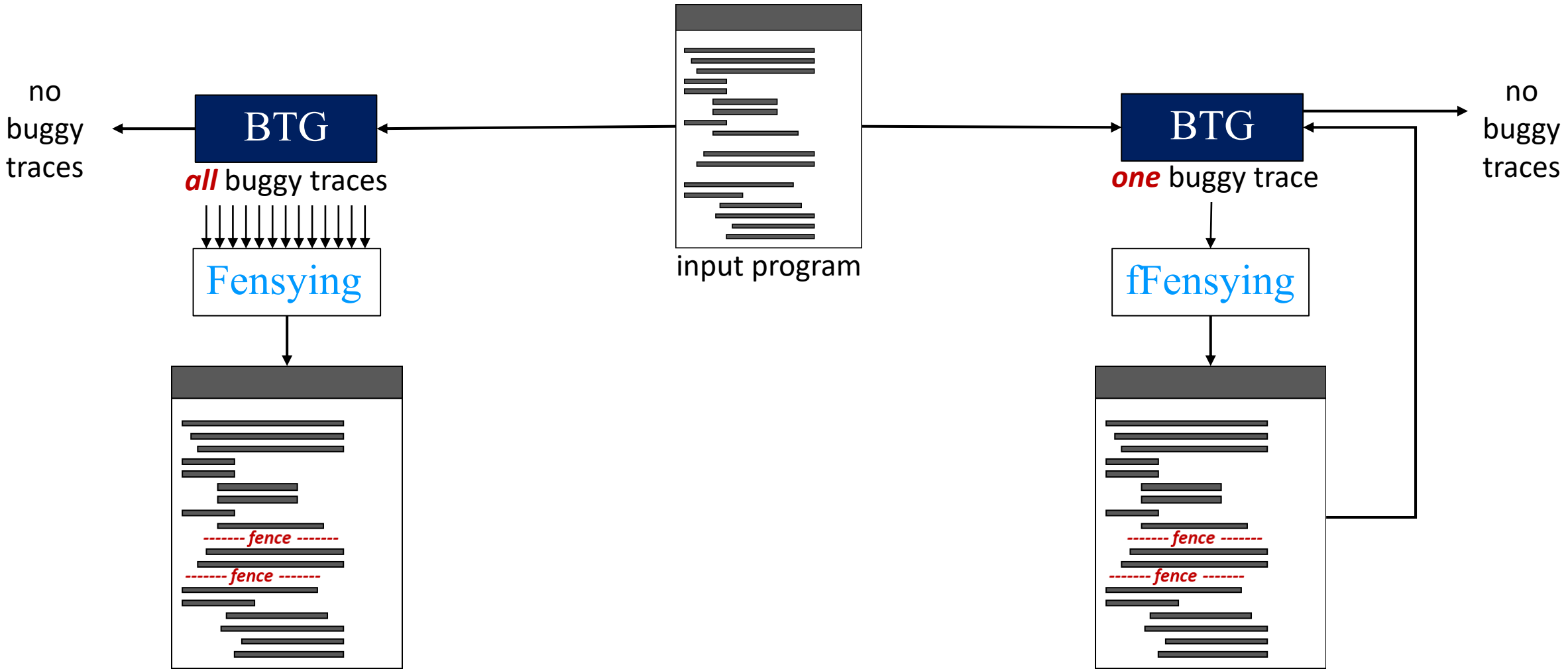
- Sound
- *near*-Optimal
- Fast
- Scales

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

Fensying *vs* fFensying



Fensying vs fFensying



Fensying *vs* fFensying

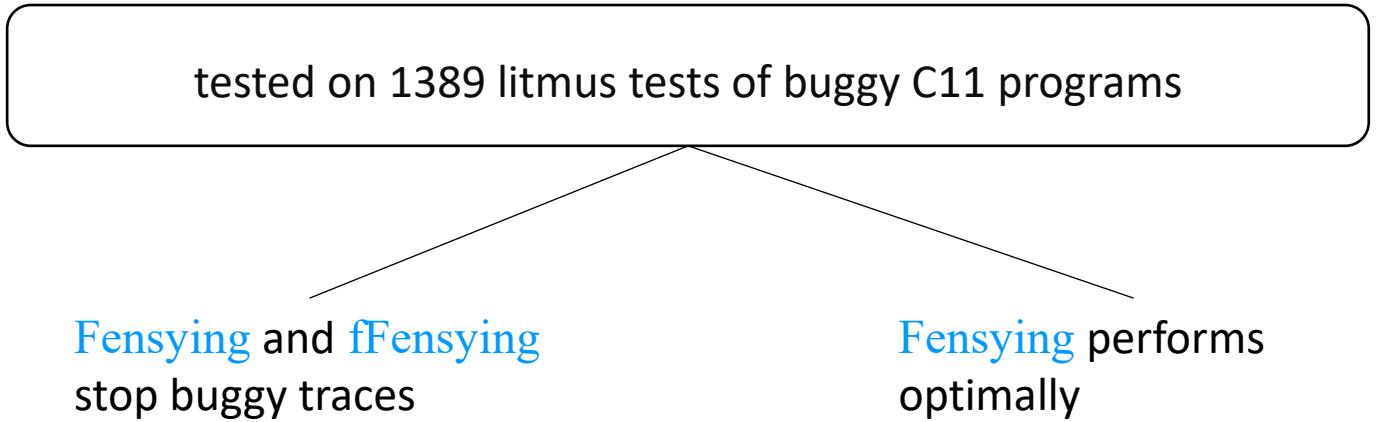
Theorem: Fensying is sound.

Theorem: Fensying is optimal.

Theorem: fFensying is sound.

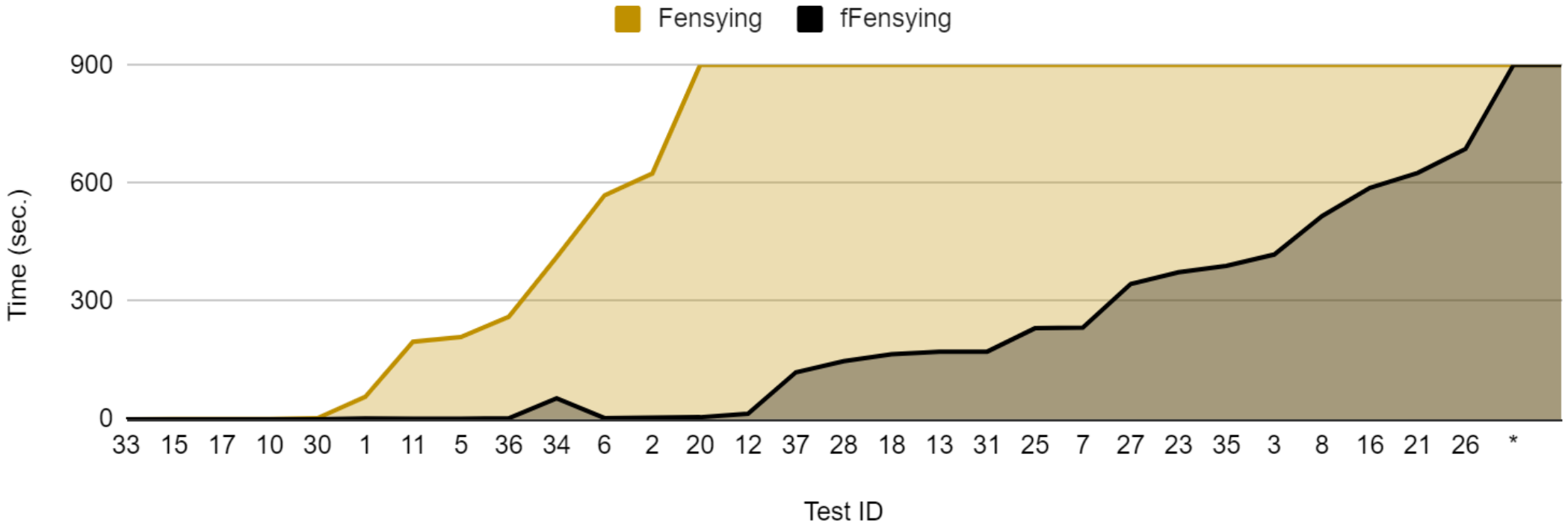
Sound: stops a buggy trace that can be stopped.
Optimal: synthesizes precise fences.

Experiments



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

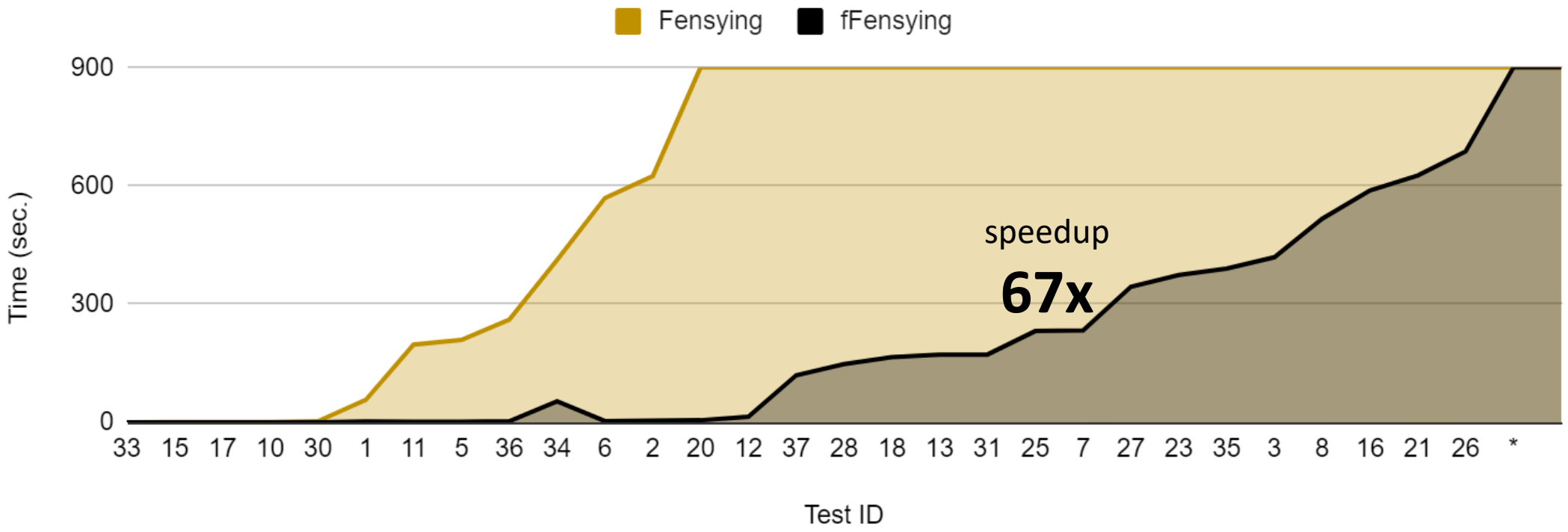
Experiments



* tests that timeout for both Fensying and fFensying

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

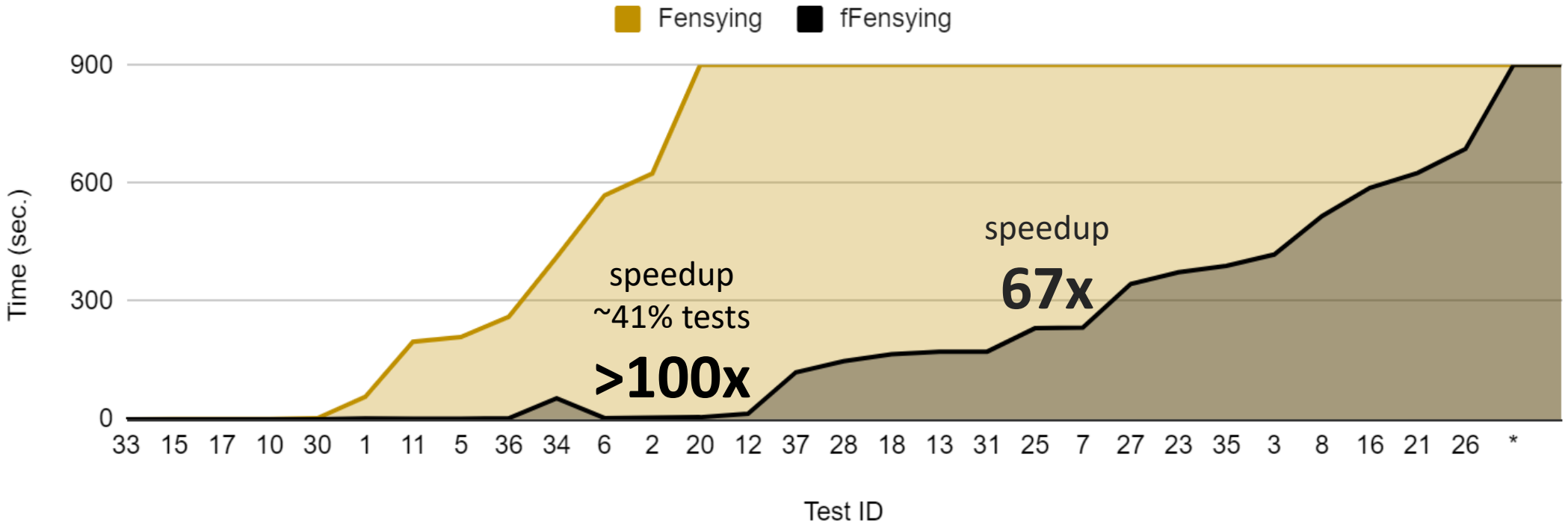
Experiments



* tests that timeout for both Fensying and fFensying

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

Experiments



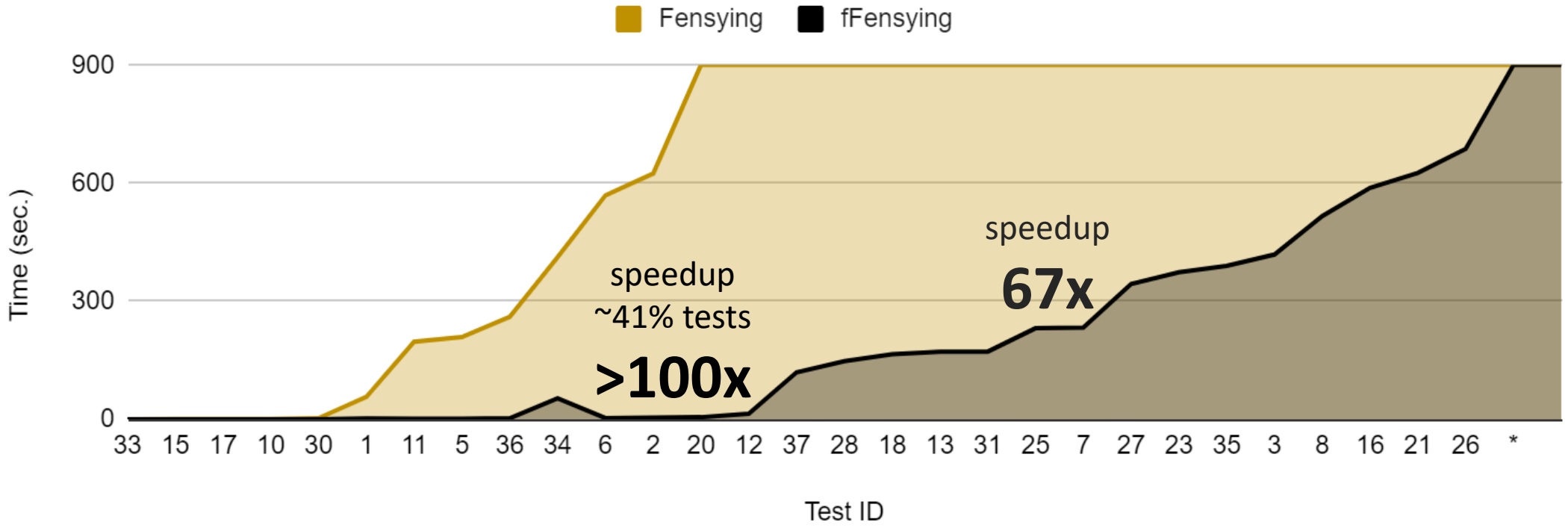
* tests that timeout for both Fensying and fFensying

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

Experiments

fFensying analysis

≤2 traces for ~85% of tests



* tests that timeout for both Fensying and fFensying

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

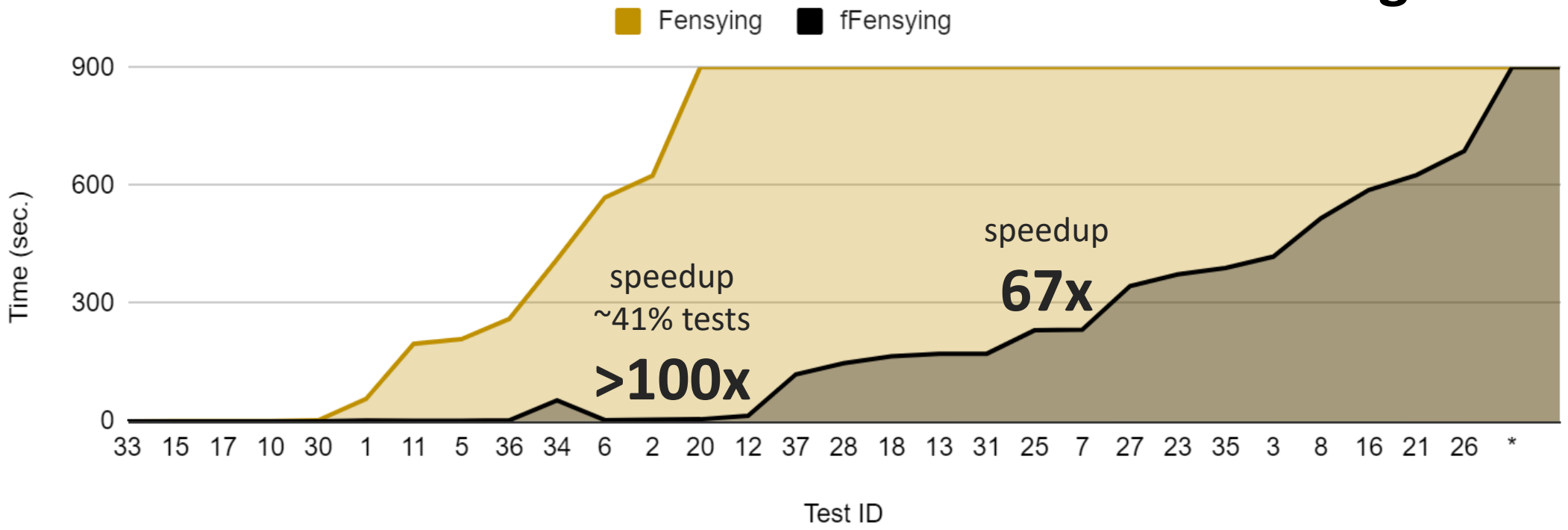
Experiments

non-optimal (fFensying)

0.005% tests

extra fences (fFensying)

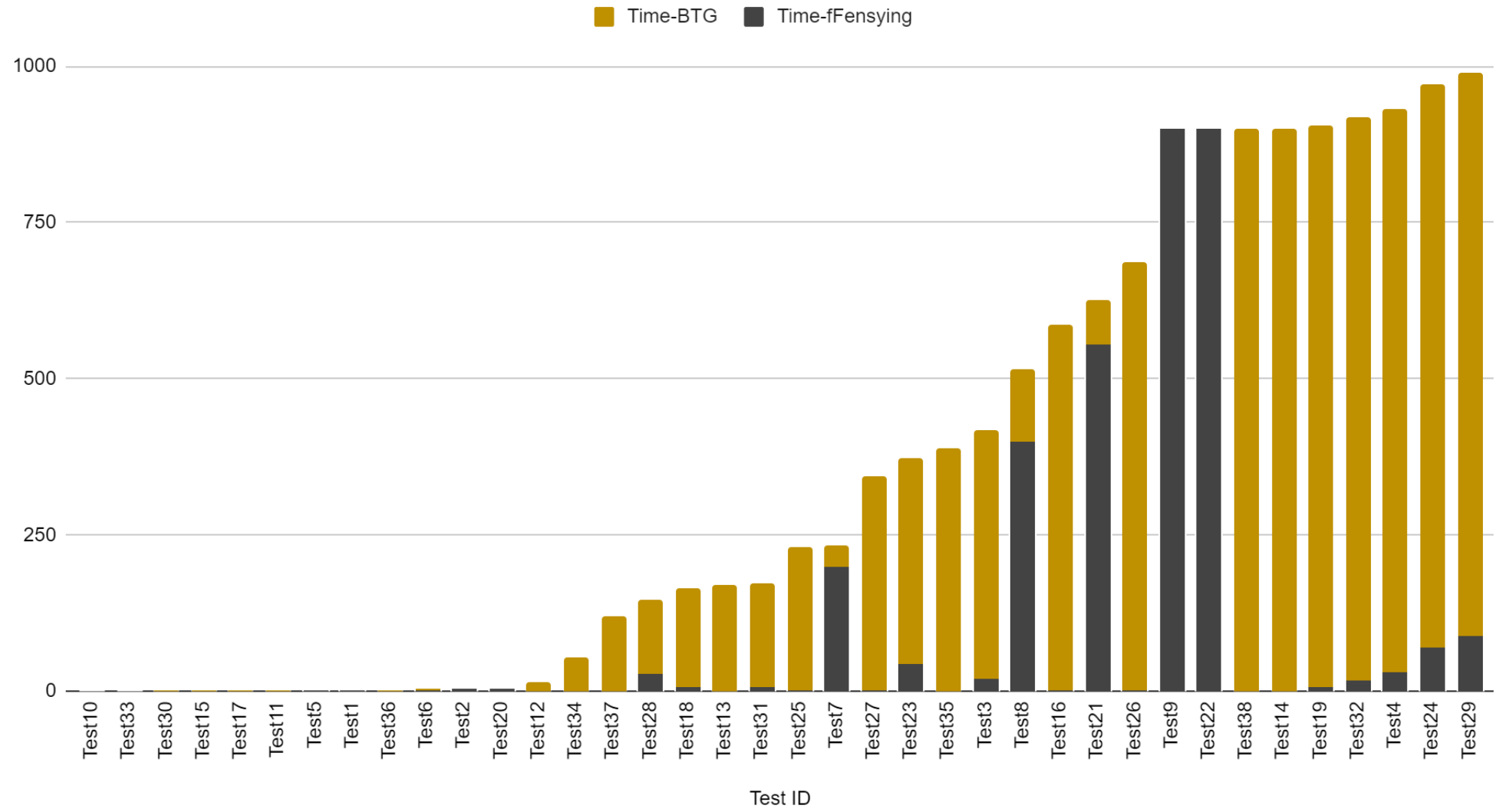
1.57 average



* tests that timeout for both Fensying and fFensying

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

Experiments (breakup of fFensying analysis time)



Future Directions

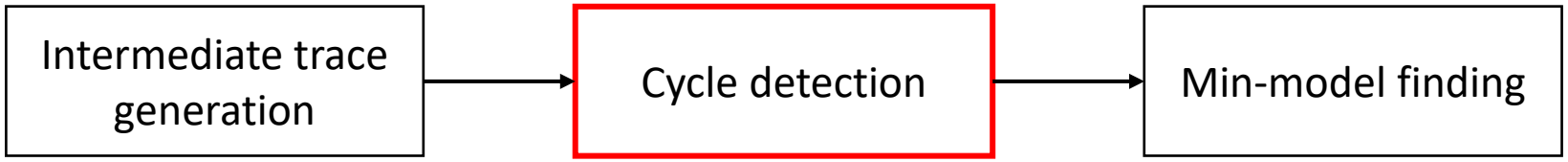
Improve BTG time

Improve fence synthesis time

Future Directions

Improve BTG time

Improve fence synthesis time



$O(|\varepsilon|+E).(C+1)$

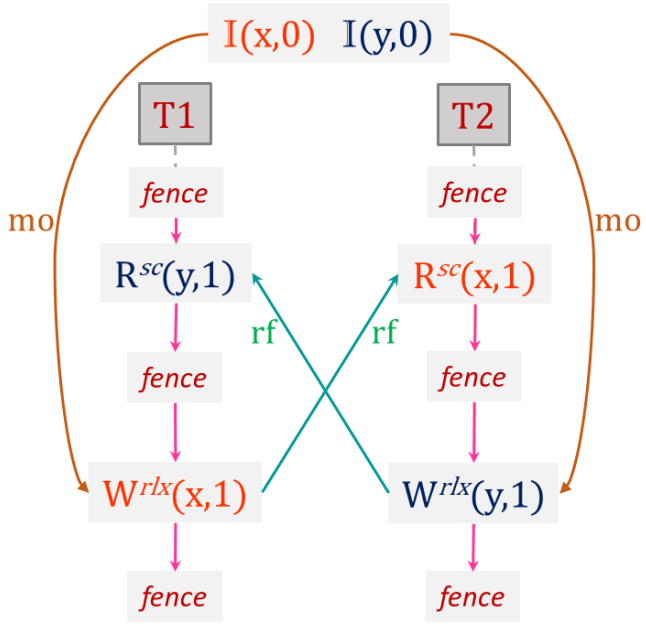
ε: set of events of buggy trace
E: #pairs of events in ε, in $O(|\varepsilon|^2)$
C: #cycles of buggy trace, in $O(|\varepsilon|!)$

Future Directions

Improve BTG time

Improve fence synthesis time

Intermediate trace generation → Cycle detection → Min-model finding



(f)Fensying tool

open source

https://github.com/singhsanjana/fensying

Thank You

Questions?

Looking for post-doc positions

Thank You

Questions?

introduction technique experiments future work

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."

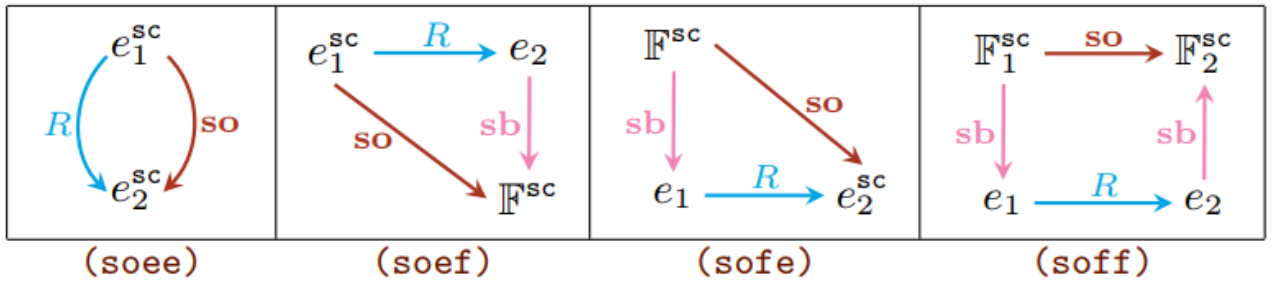
source: https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0668r5.html
(Bullet 4. under 'Revising the C++ memory model')

back

Fensying technique

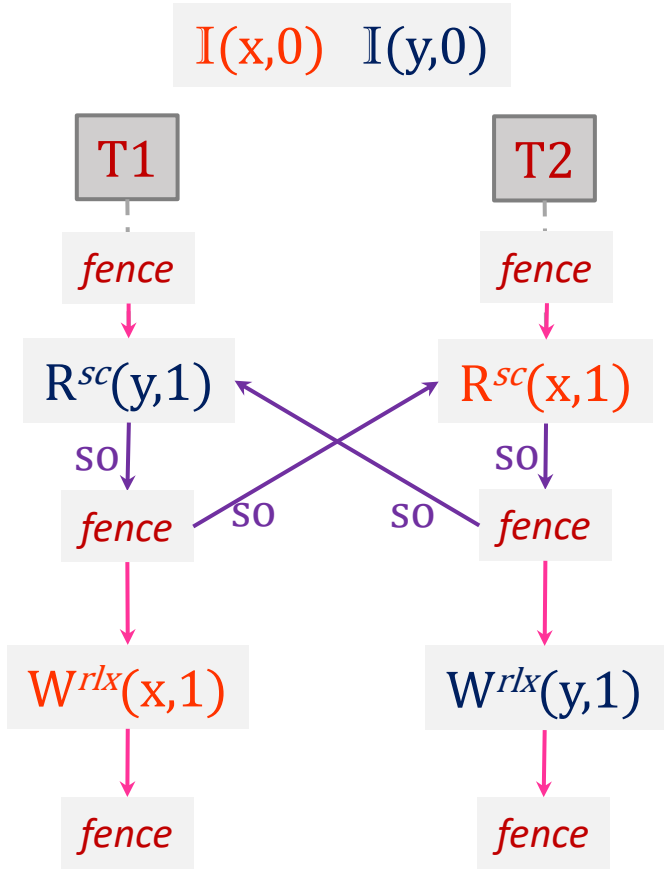
Step 3 detect violations of coherence
(*strong-fensying*)

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



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

inability to create a total-order

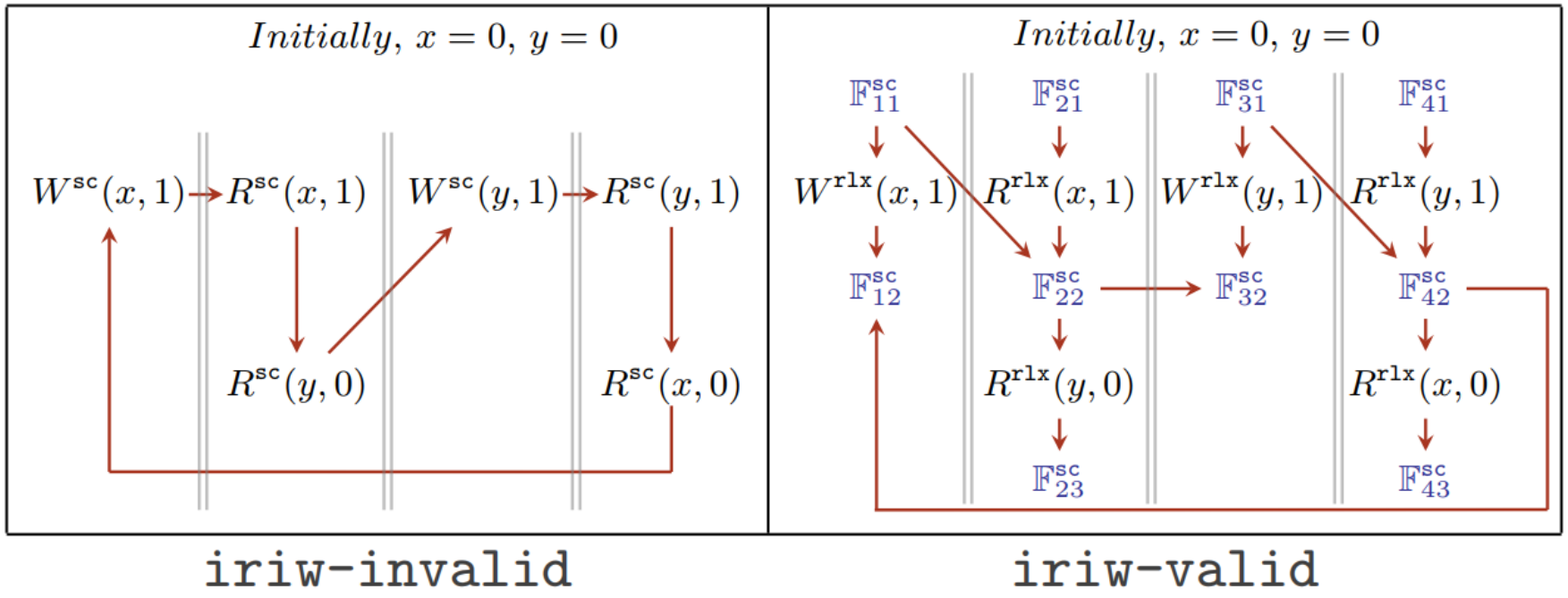


back

assuming seq-cst ordered fences

Fence synthesis vs event strengthening

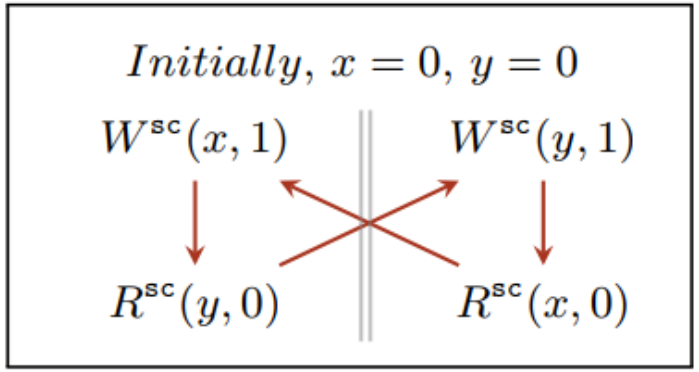
C11 fences do not restore sequential consistency

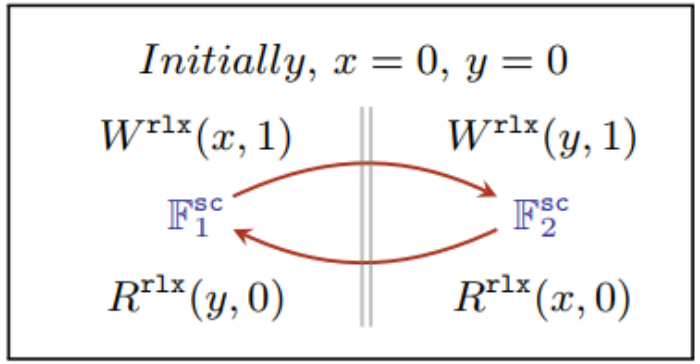


top end

Fence synthesis vs event strengthening

Interpreting barriers from memory orders is not precise

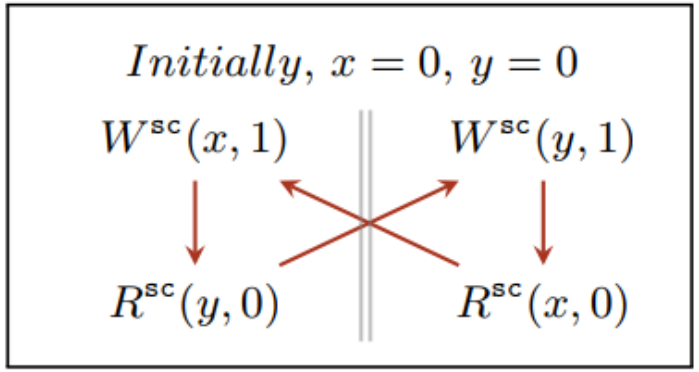


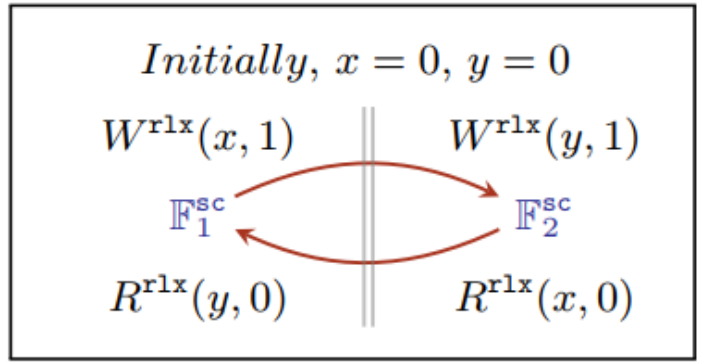


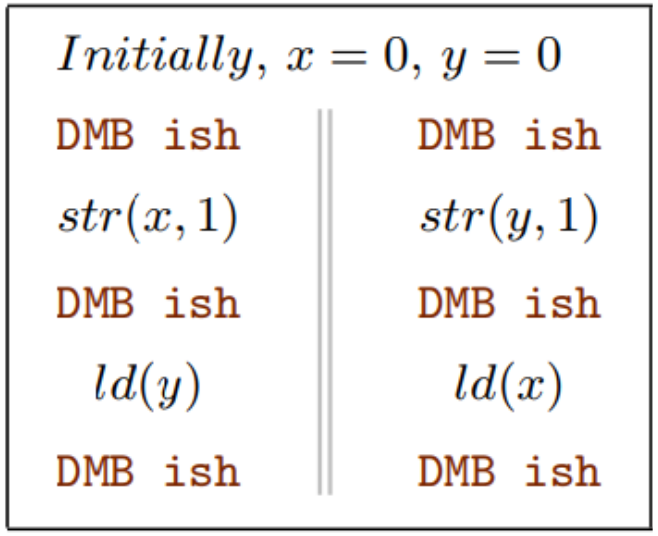
top end

Fence synthesis vs event strengthening

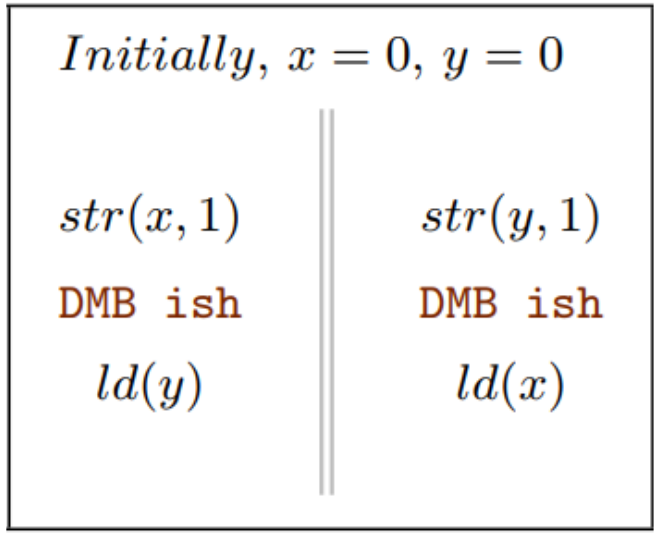
Interpreting barriers from memory orders is not precise







barriers on ARM

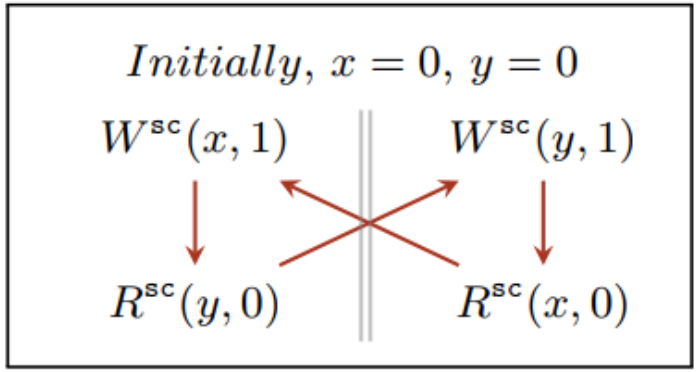


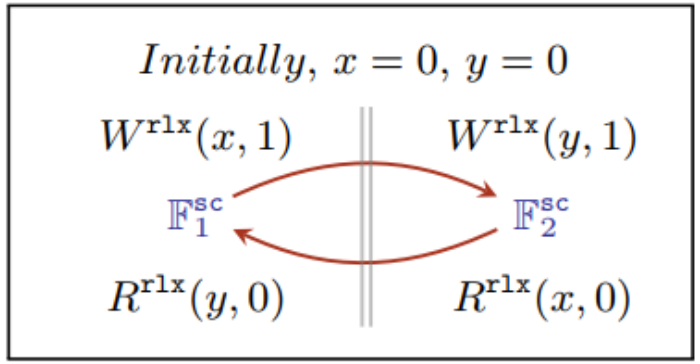
barriers on ARM

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
$ld(y)$	$ld(x)$

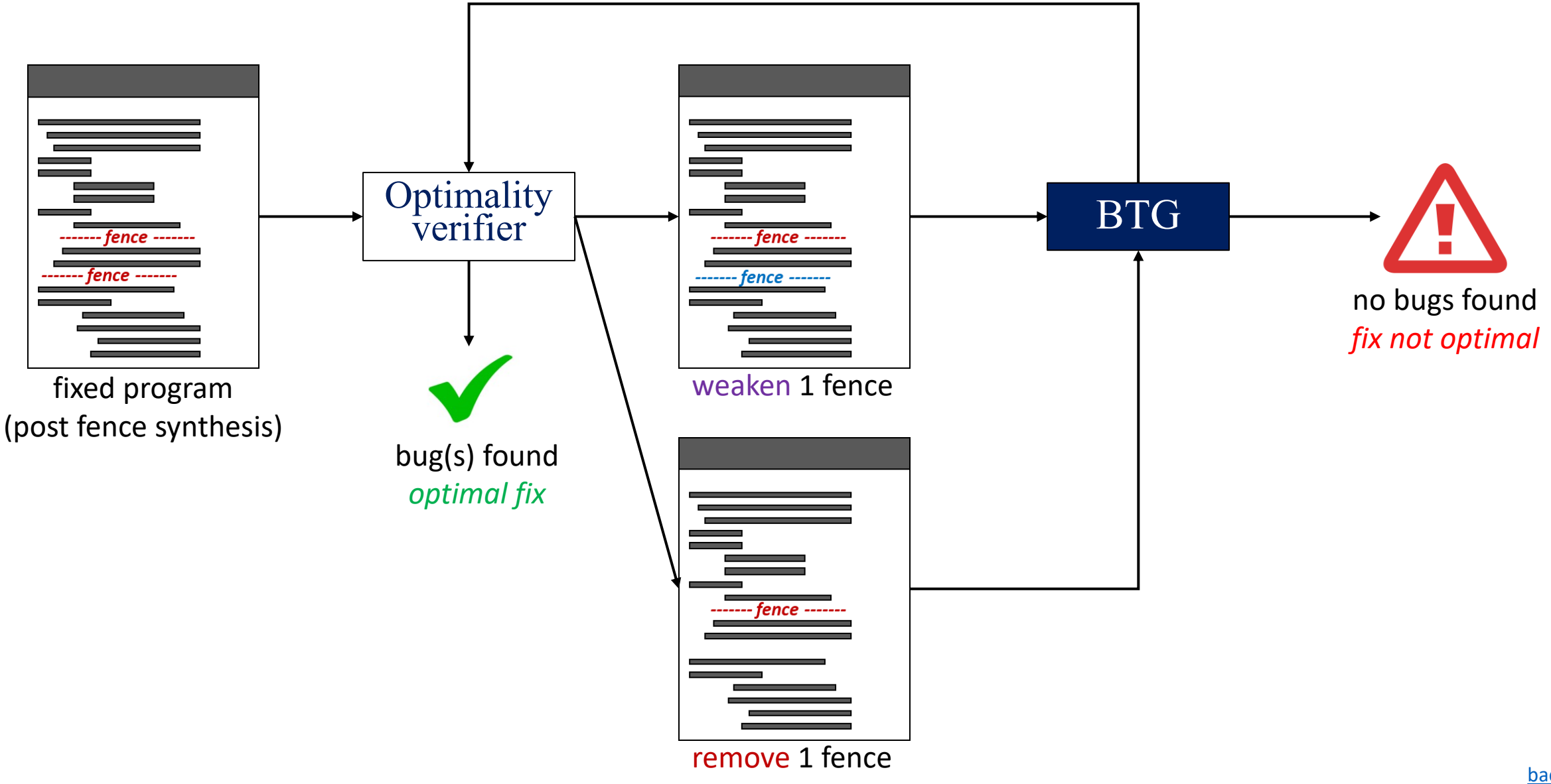
barriers on ARM

Initially, $x = 0$, $y = 0$	
$str(x, 1)$	$str(y, 1)$
hwsync	hwsync
$ld(y)$	$ld(x)$

barriers on power

top end

Verifying optimality

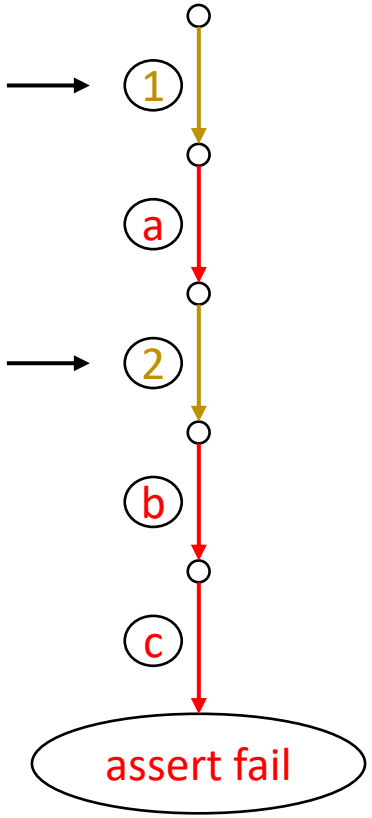


back

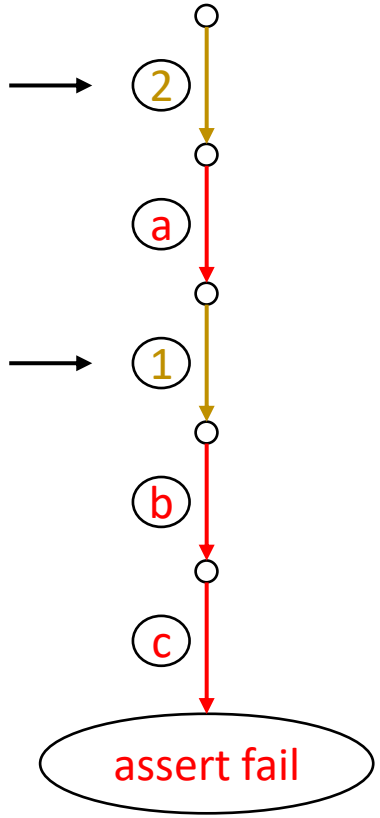
Reason (≤2 traces for ~85% of tests)

affect assert condition

does not affect assert condition



buggy trace 1



buggy trace 2

back