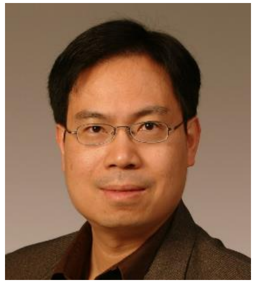
香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

RECONTEST: Effective Regression Testing of Concurrent Programs


Valerio Terragni Shing-Chi Cheung Charles Zhang

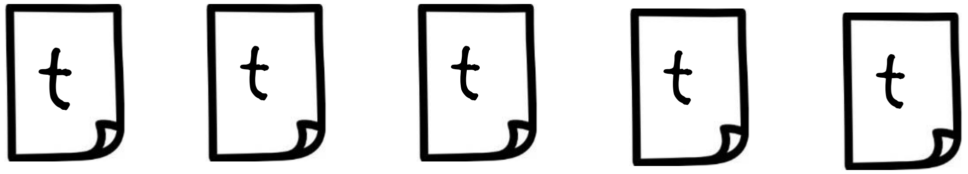
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
{vterragni, scc, charlesz}@cse.ust.hk

20 May 2015

Regression Testing is Costly

$P \longrightarrow P'$

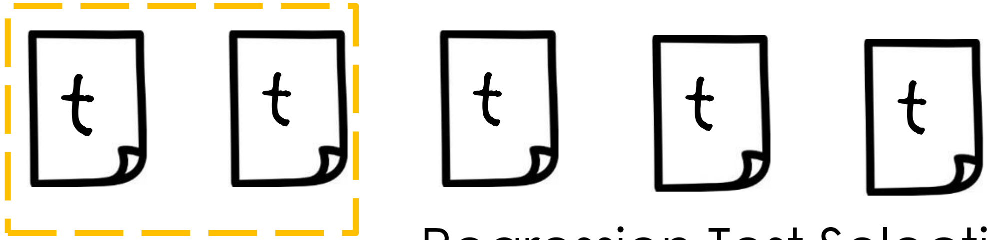
existing test suite T



Regression Testing is Costly

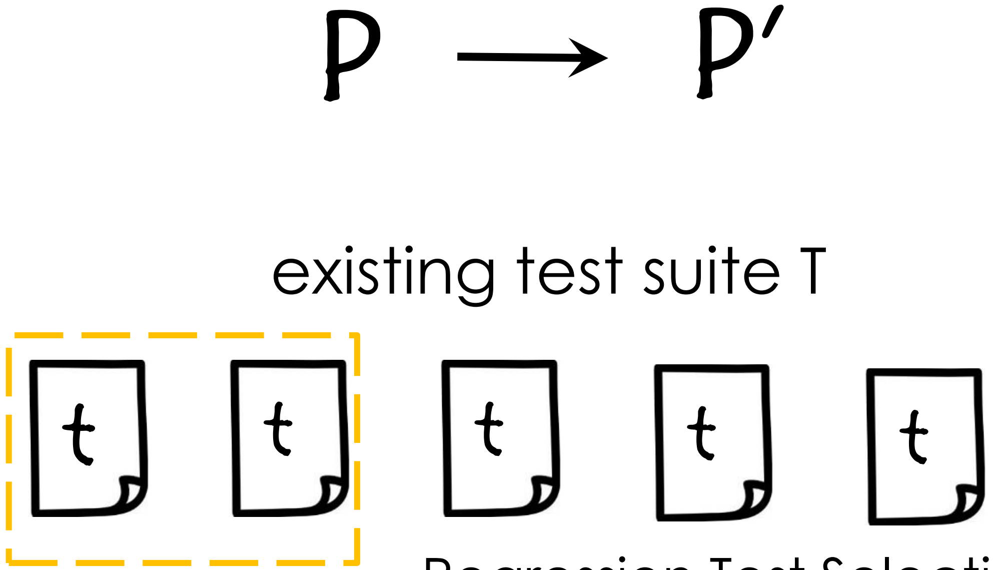
$P \longrightarrow P'$

existing test suite T



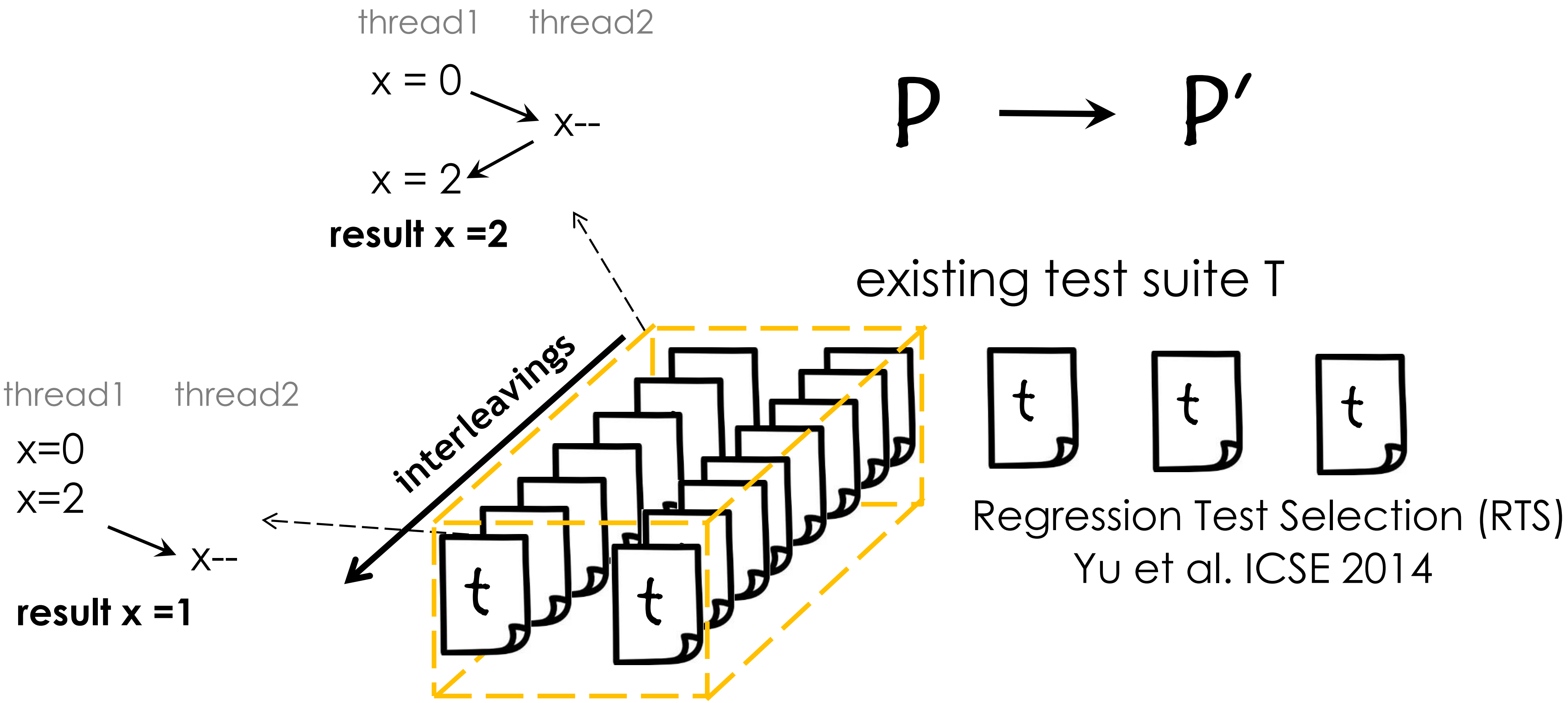
Regression Test Selection (RTS)
Yu et al. ICSE 2014

Regression Testing is Costly



Regression Test Selection (RTS)
Yu et al. ICSE 2014

Regression Testing is Costly



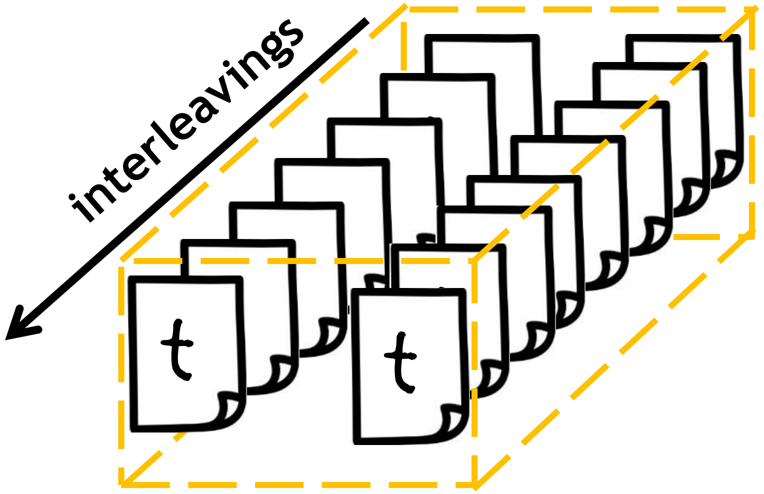
Regression Testing is Costly

size of interleaving space

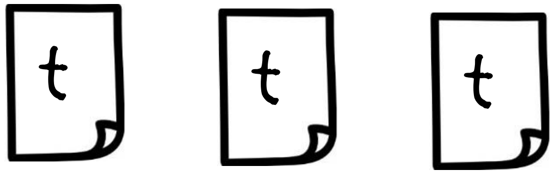
$$\frac{(n+m)!}{n!m!} \qquad n\left\{\begin{array}{l} \text{thread1} \\ W(x) \\ R(x) \\ \dots \end{array}\quad \begin{array}{r} \text{thread2} \\ R(z) \\ W(x) \\ \dots\dots \end{array}\right\} m$$

n = m = 15
155 millions
interleavings!

$P \longrightarrow P'$

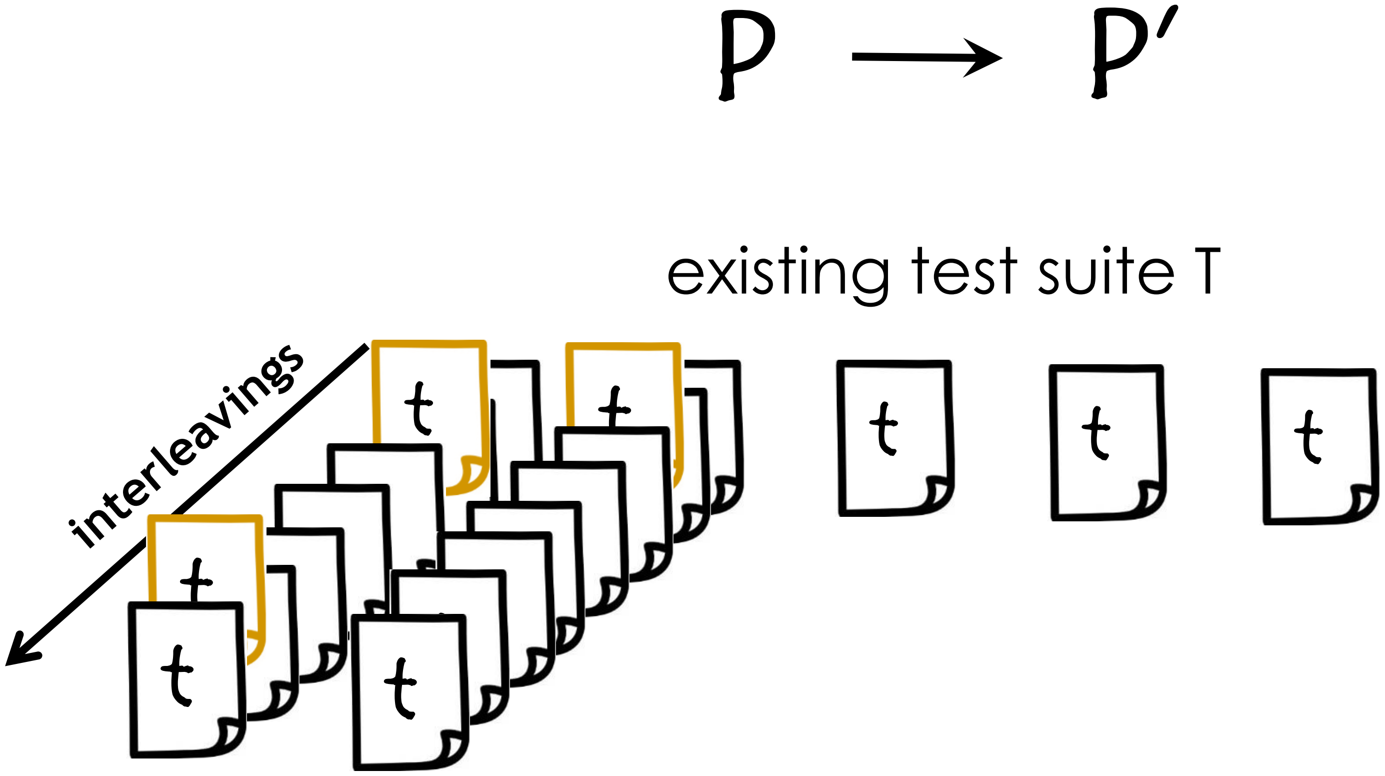


existing test suite T



Regression Test Selection (RTS)
Yu et al. ICSE 2014

Regression Testing is Costly

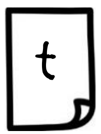


Selection of interleavings for regression testing

Problem Formulation

- Given a test **t**, select the **new** interleavings that are observable when **t** runs with the modified version P′(t)

```
public void add(int n){
synchronized(lock){      --
  x = x + n;
  }
}
```



A o = new A ();

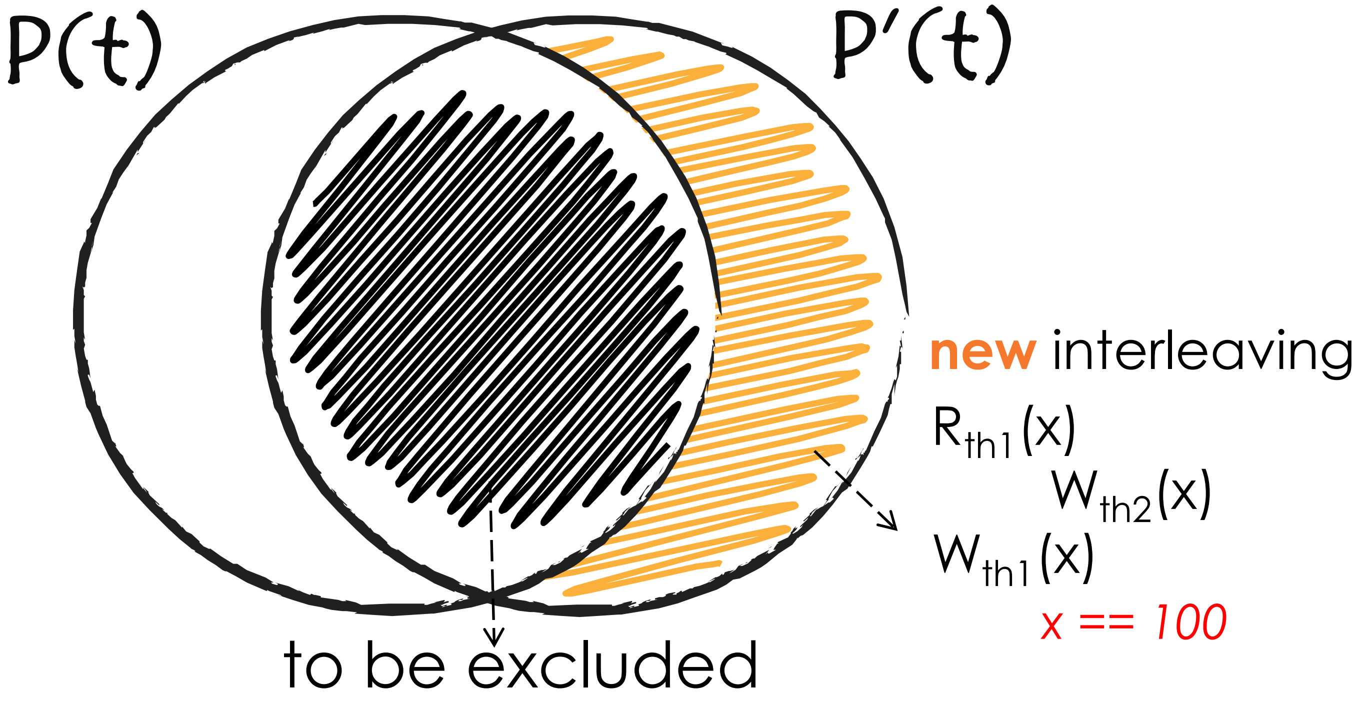
thread1 *thread2*

o.add(100); o.add(100);

R_{th1}(x)
W_{th1}(x)
R_{th2}(x)
W_{th2}(x)

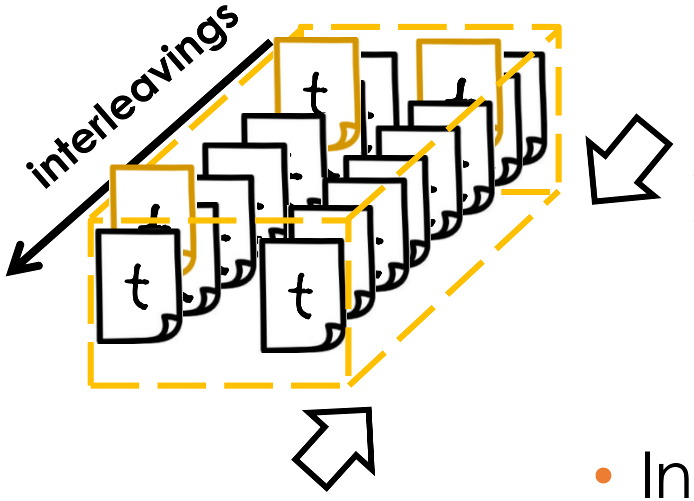
x == 200

Interleaving space



P-Correct-for-T assumption

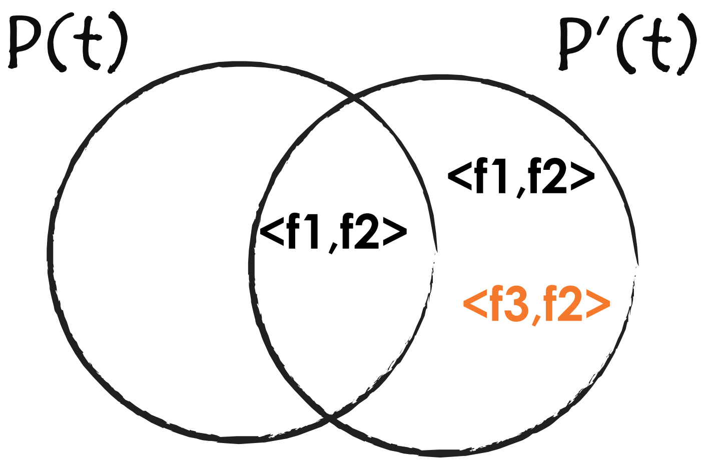
State of the Art



- Reducing the cost of **regression verification**
  - ➢ Reuse of verification results
    - Lauterburg et al. ICSE 2008
    - Yang et al. ICSM 2009
  - ➢ Prioritize the exploration of multithreaded tests
    - Jagannath et al. ISSTA 2011

- Incremental **model checking** remains too expensive
  - ➢ It systematically explore the interleaving space
    - The number of **new interleavings** also growths exponentially with execution length

State of the Art (cont.)



- Reduce cost of concurrency bug detection by characterizing and avoiding the interleaving space overlap across **inputs** and **software versions**
  - ➢Deng et al. HotPar 2012, OOPSLA 2013
  - ➢**Concurrent Function Pairs (CFP)**
    - function level is too coarse grained
      - this metric at memory access level would be too expensive

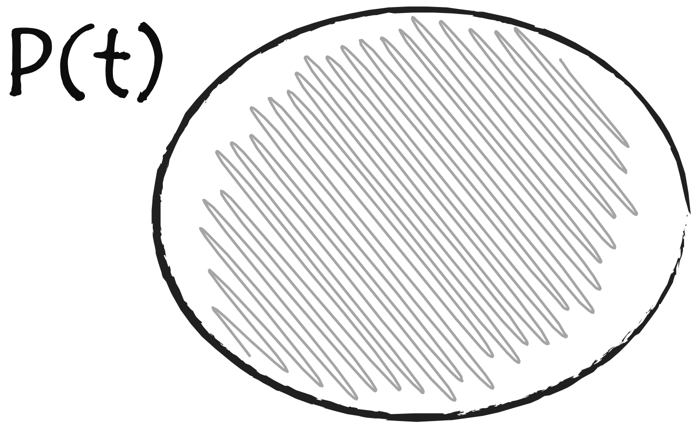
Fundamental Challenge

Fundamental Challenge

Naïve solution

1. Store the explored interleavings of P(t)

Interleaving space

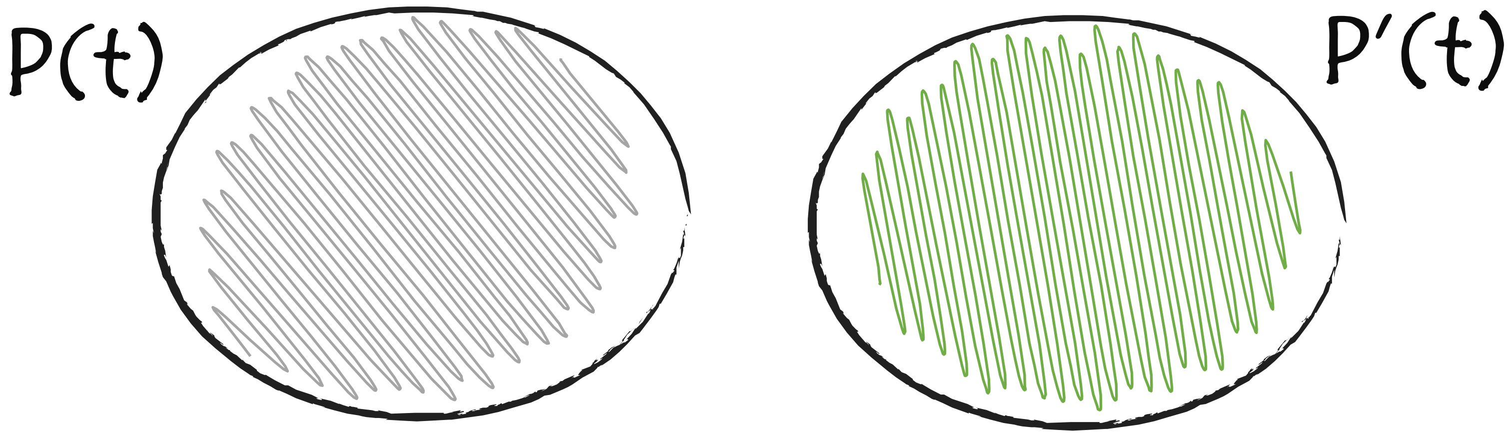


Fundamental Challenge

Naïve solution

1. Store the explored interleavings of P(t)
2. Explore the interleaving space of P'(t)

Interleaving space

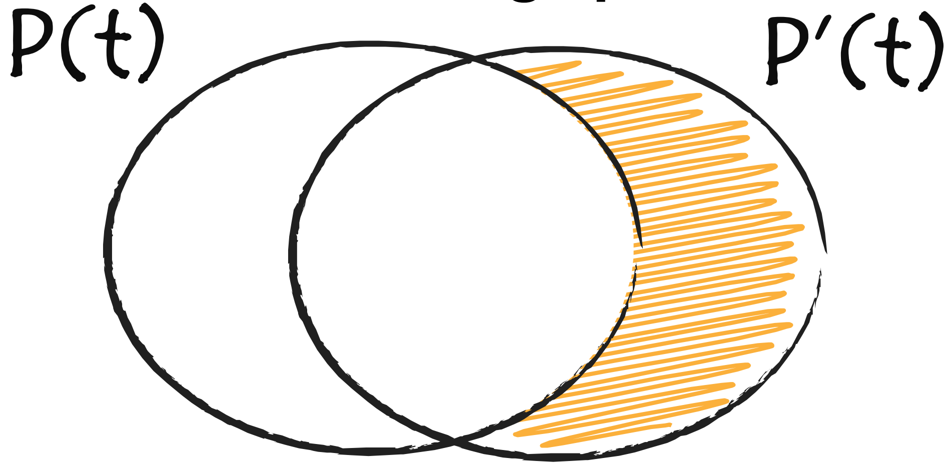


Fundamental Challenge

Naïve solution

1. Store the explored interleavings of P(t)
2. Explore the interleaving space of P'(t)
3. Identify and select **new** interleavings by computing the **difference set**

Interleaving space

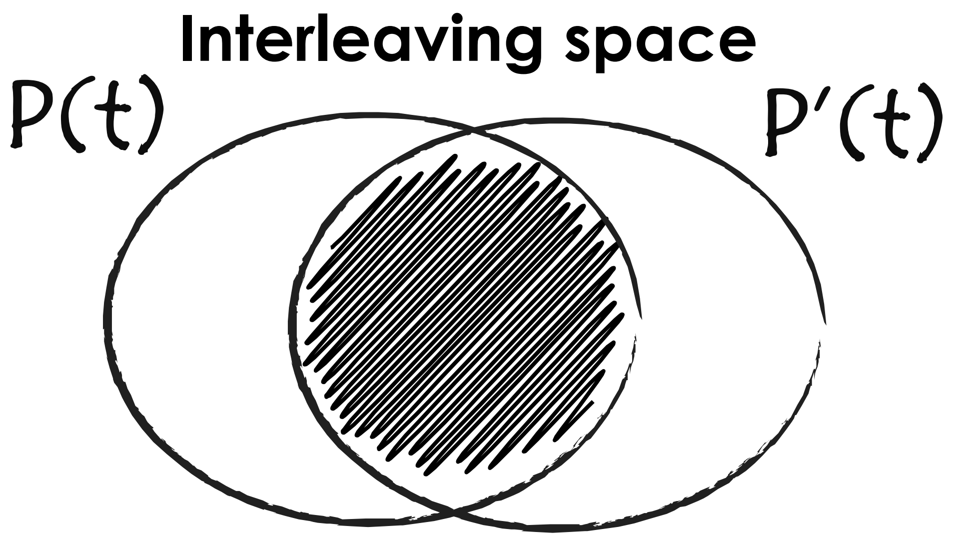


Fundamental Challenge

Naïve solution

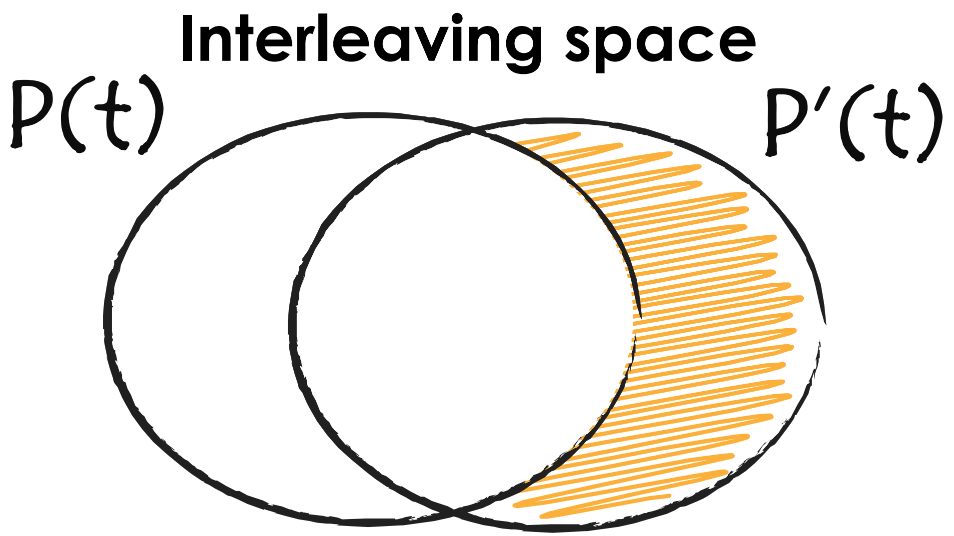
1. Store the explored interleavings of P(t)
2. Explore the interleaving space of P'(t)
3. Identify and select **new** interleavings by computing the **difference set**

- It does not eliminate the cost of re-exploring redundant interleavings in P'(t)



Fundamental Challenge

How to explore **only** the **new** interleavings and then select them for regression testing?



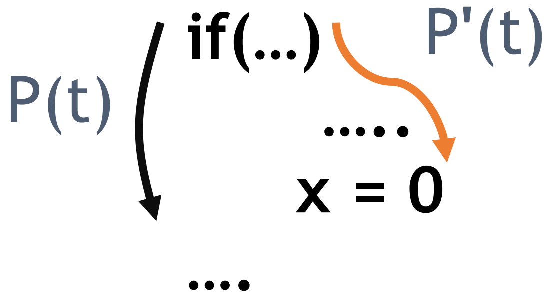
Necessary Property of a New Interleaving

- Contain at least one shared memory access impacted by the revisions

Necessary Property of a New Interleaving

- Contain at least one shared memory access impacted by the revisions
  - ➢ **Impact-set**
    1. **new** accesses triggered by new statements or new execution paths

x = 0 ++



Necessary Property of a New Interleaving

- Contain at least one shared memory access impacted by the revisions
  - **Impact-set**
    1. **new** accesses triggered by new statements or new execution paths
    2. accesses that can interleave in new ways because they have an altered **Concurrency Context (CC)**
       - lock acquire/release histories
       - happens-before relations (notify()/wait())

x = 0 ++

P(t)

if(…)

P'(t)

…..

x = 0

….

synchronized(lock){ --

LockSet={lock} P(t)

LockSet={ } P'(t)

RECONTEST

P → t → P′

PHASE I

Change-Impact Analysis (CIA)

⇨ **impact-set** ⇨

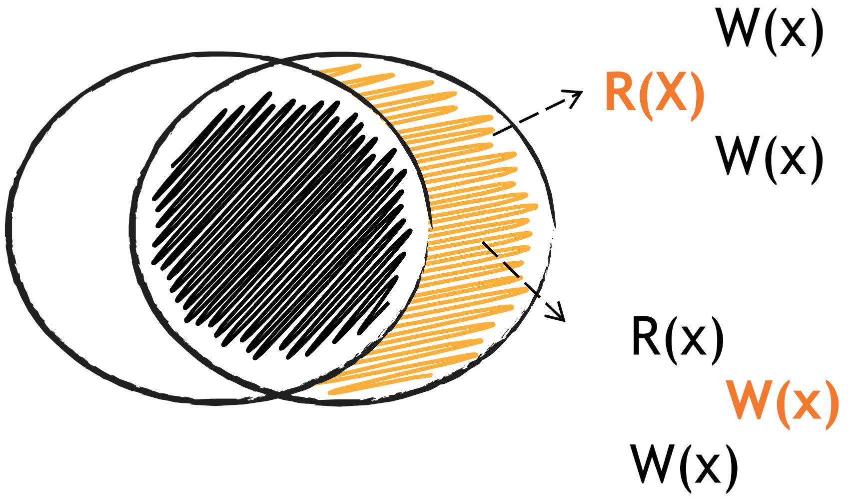
PHASE II

Selection (exploration) new interleavings

coverage criterion

```
public void add(int n){
  synchronized(lock){        --
    x = x + n;
  }
}
```

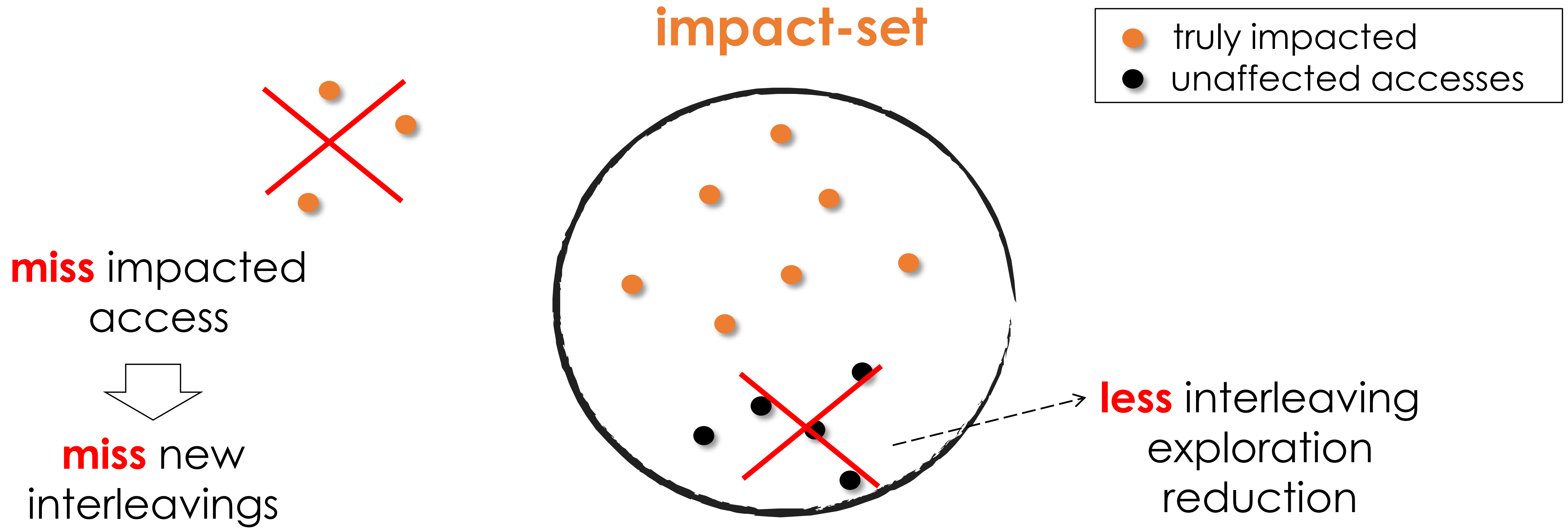
Identify **impacted** shared memory accesses



Explore only the interleavings containing at least one impacted access

How to Identify the **Truly** Impacted Accesses?

Complete & Minimal



Limitations of Existing Change Impact Analysis

- CIAs specific to concurrent semantic
  - Jagannath et al. ISSTA 2011
  - Yu et al. ICSE 2014

~~Complete~~ or ~~Minimal~~

```
public void add(int n){
synchronized(lock){ --
  x = x + n;
  }
}
```

- Dependency-based impact analysis
  - **Static** analysis is imprecise
  - They require to **enumerate a priori** all the changes that affect Concurrency Contexts
    - Do not consider changes that affect happens before relations (wait()/notify())

RECONTEST's Phase I

- Dynamic
- No dependency based

$P(t)$	$P'(t)$
$R_{th1}(x)$	$R_{th1}(x)$
....	....
....	....
....	....
$W_{th1}(x)$	$W_{th1}(x)$
....	....
....	$W_{th1}(x)$
.....	
$R_{th1}(x)$	$R_{th1}(x)$
	....
	$W_{th1}(x)$

no interleaving exploration!

RECONTEST's Phase I

- ➢**Dynamic**
- ➢**No dependency based**

```
public void add(int n){
synchronized(lock){  --
  x = x + n;
  }
}
```

Concurrency Context (CC)

Lockset

{lock} ≠ { }

$P(t)$		$P'(t)$
$R_{th1}(x)$	⟷ equivalent	$R_{th1}(x)$ ⇦
….		….
….		….
….		….
$W_{th1}(x)$		$W_{th1}(x)$
….		….
….		$W_{th1}(x)$
…..		
$R_{th1}(x)$		$R_{th1}(x)$
		….
		$W_{th1}(x)$

RECONTEST's Phase I

- **Dynamic**
- **No dependency based**

Concurrency Context (CC)

obj.wait();
....
x = 0;

P(t)		P'(t)
$R_{th1}(x)$		$R_{th1}(x)$
....	**Happens Before**	....
....	$\boxed{\{obj.wait()\} = \{obj.wait()\}}$	....
....		....
$W_{th1}(x)$		$W_{th1}(x)$
....		....
....		$W_{th1}(x)$
.....		
$R_{th1}(x)$		$R_{th1}(x)$
		....
		$W_{th1}(x)$

RECONTEST's Phase I

- **Dynamic**
- **No dependency based**

Concurrency Context (CC)

$P(t)$		$P'(t)$
$R_{th1}(x)$		$R_{th1}(x)$
....		....
....		....
....		....
$W_{th1}(x)$		$W_{th1}(x)$
....		....
....		$W_{th1}(x)$
.....	**Lockset**	
$R_{th1}(x)$	{lock} ≠ {lock}	$R_{th1}(x)$
		....
		$W_{th1}(x)$

```
this.lock = new Object(); ++
this.lock = lock   --
```

RECONTEST's Phase I

- **Dynamic**
- **No dependency based**

Concurrency Context (CC)

P(t)	P′(t)
$R_{th1}(x)$	$R_{th1}(x)$
….	….
….	….
….	….
$W_{th1}(x)$	$W_{th1}(x)$
….	….
….	? $W_{th1}(x)$
…..	$R_{th1}(x)$
$R_{th1}(x)$	….
	? $W_{th1}(x)$

P(t) if(…) P'(t)

….. x = 0

….

x = 0 ++

Phase I's Technical Challenges

1. How to **align** (precisely and efficiently) execution points ?
   - ➢ Our CIA's algorithm
     - cluster accesses triggered by equivalent statements
     - linear time complexity w.r.t execution trace lengths

$R_{th1}(x) \longleftrightarrow R_{th1}(x)$

alignment

2. How to match dynamic objects across executions?
   - ➢ We present two **change-resilient** heap abstractions
     - Object identify abstraction
     - Change resilient k-CFA [Shivers PLDI 1988]

{lock} ≠ {lock}

$R_{th1}(x)$ $R_{th1}(x)$

Phase I's Guarantees

- Given an execution trace RECONTEST computes its **complete and minimal** impact-set under the following assumptions
  - Unmodified source code lines are perfectly aligned

P	P′
	y =1; ++
z = 3;	z = 3; --
x = 0; ⟷	x = 0;

  - Change resilient k-CFA is precise

RECONTEST's Phase II

- Given a coverage criterion explore interleavings that
  1. Are a coverage requirement (match a problematic access pattern)
  2. Contain at least one **impacted** access

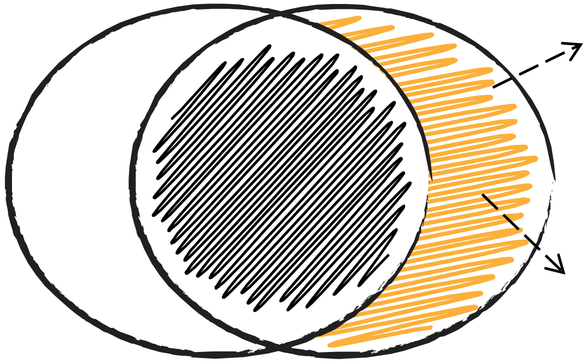
atomic-set serializability
Vaziri et al. POPL 2006

$W_{u1}(x)$
$R_{u2}(X)$
$W_{u2}(x)$

11 problematic access patterns

Predictive Trace Analysis (PTA)
[Sen et al. FMOODS 2005]

off-line interleaving exploration



Validation

W(x)
R(X)
W(x)

feasible

R(x)
W(x)
W(x)

infeasible

RECONTEST's Phase II (cont.)

- **Naïve solution** does not reduce interleaving exploration costs
  1. Detect all potential problematic interleavings in P'(t)
  2. Pruning those that do not contain **impacted** accesses

- RECONTEST explores **only** problematic interleavings containing impacted accesses
  - It starts the off-line interleaving exploration from the impact-set

<?>

R(x)

<?>

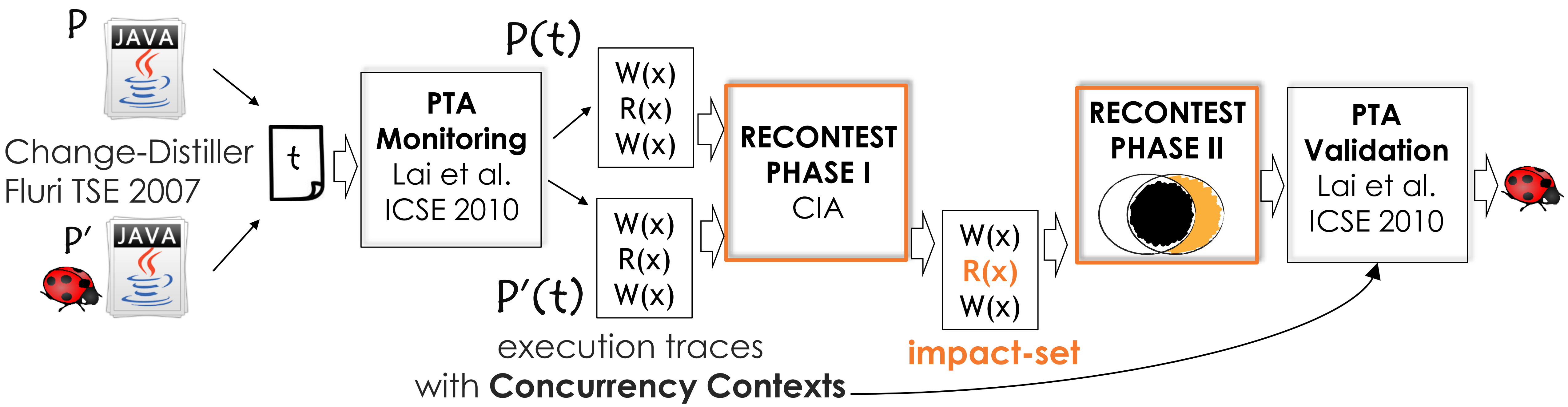
problematic access pattern

$W_{u1}(x)$

$\mathbf{R_{u2}(X)}$

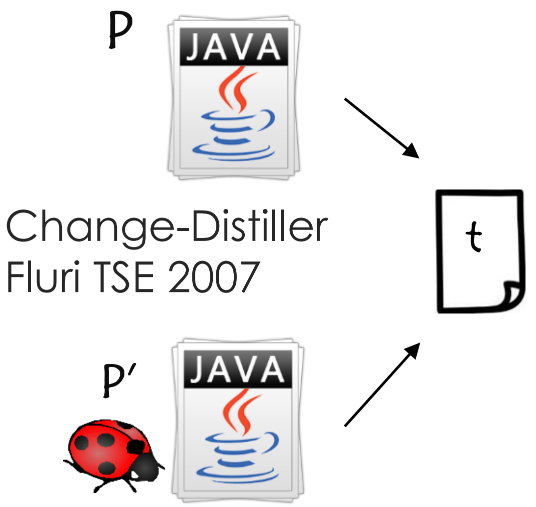
$W_{u2}(x)$

RECONTEST



http://sccpu2.cse.ust.hk/recontest/index.html

Subjects



ID	Name	SLOC	REF.	|change-set|
1	Groovy	361	GROOVY-1890	89
2	Airline	136	[1]	10
3	Log4j	2,598	CONF-1603	23
4	Pool	745	POOL-120	970
5	Lang	486	LANG-481	9
6	Vector1	835	JDK-4420686	7,836
7	SBuffer1	1,265	JDK-4810210	15
8	SBuffer2	1,249	JDK-4790882	69
9	Vector2	292	JDK-4334376	2
10	Garage	554	[1]	20
11	Logger	39 K	JDK-4779253	1,661
12	Xtango	2,097	[2]	26
13	Cache4j	3,897	[3]	128

[1] Farchi IPDPS 2003
[2] https://www.cs.drexel.edu/~umpeysak/Xtango/
[3] http://cache4j.sourceforge.net/

RECONTEST's Phase I

RECONTEST
PHASE I
CIA

W(x)
R(x)
W(x)

impact-set

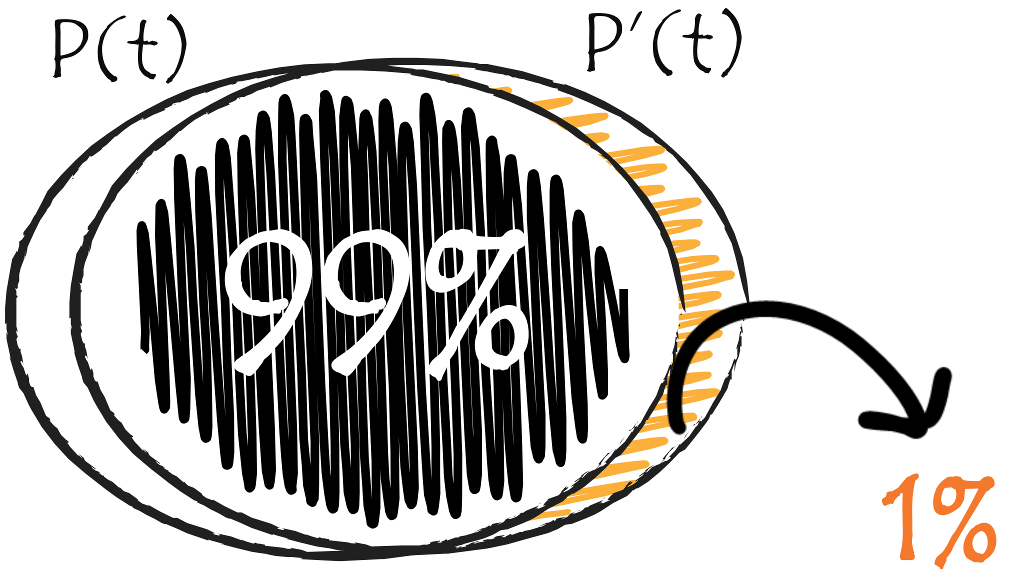
- small impact-set
- efficient algorithm

ID	#shared memory accesses P'(t)	% impacted	Time Phase I
1	49	53.06%	**9 ms**
2	55	16.36%	10 ms
3	130	6.15%	16 ms
4	274	73.57%	42 ms
5	310	10.97%	27 ms
6	381	1.31%	19 ms
7	1,909	0.20%	104 ms
8	2,527	1.50%	106 ms
9	3,447	0.10%	122 ms
10	17 K	0.03%	409 ms
11	39 K	0.19%	**15.10 s**
12	150 K	0.14%	3.07 s
13	570 K	0.02%	7.99 s

RQ1: Effectivness

How much interleaving space reduction can be achieved by RECONTEST?

ID	# problematic interleavings	RECONTEST # problematic interleavings	RQ1
1	295	162	**1.82x**
2	325	157	2.07x
3	224	12	18.67x
4	3,040	328	9.27x
5	23,038	5,322	4.33x
6	45,656	74	617x
7	396,256	1,102	360x
8	1,437,972	151,214	9.51x
9	144,354,356	1,053	**137,088x**
10	16,086,708	32,846	490x
11	110,337	7,430	14.85x
12	>236,340,709	326,603	>724x
13	>116,616,808	64,670	>1,803x



problematic interleavings containing at least **one impacted access** (all subjects as a whole)

RQ2: Efficiency

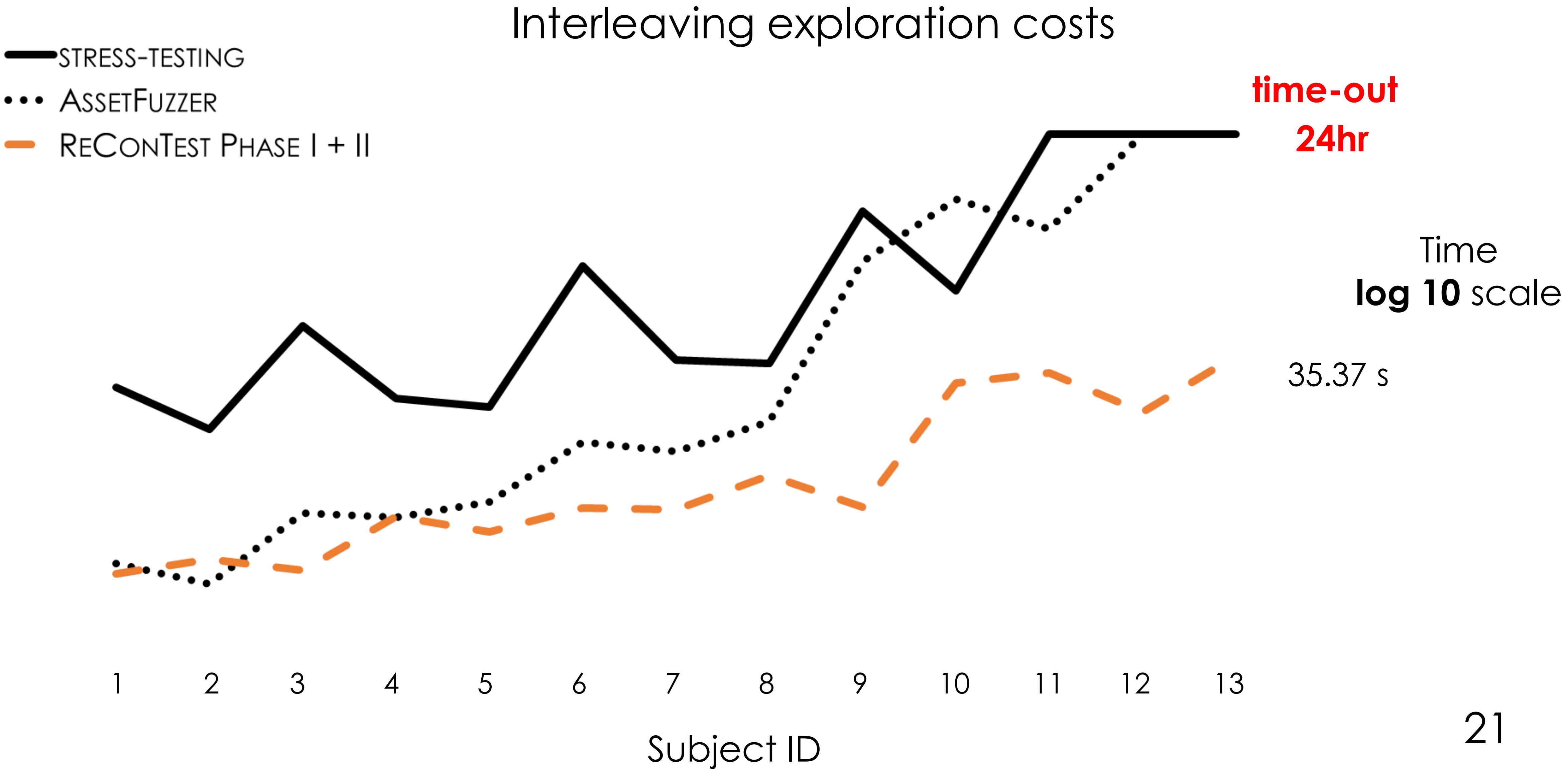
Does the overhead of pre-computing the impact-set out-weight the reduction in test effort?

CIA
+
Exploring problematic interleavings that contain at least one impacted access
+
pruning infeasible ones with the Lockset & HB analysis

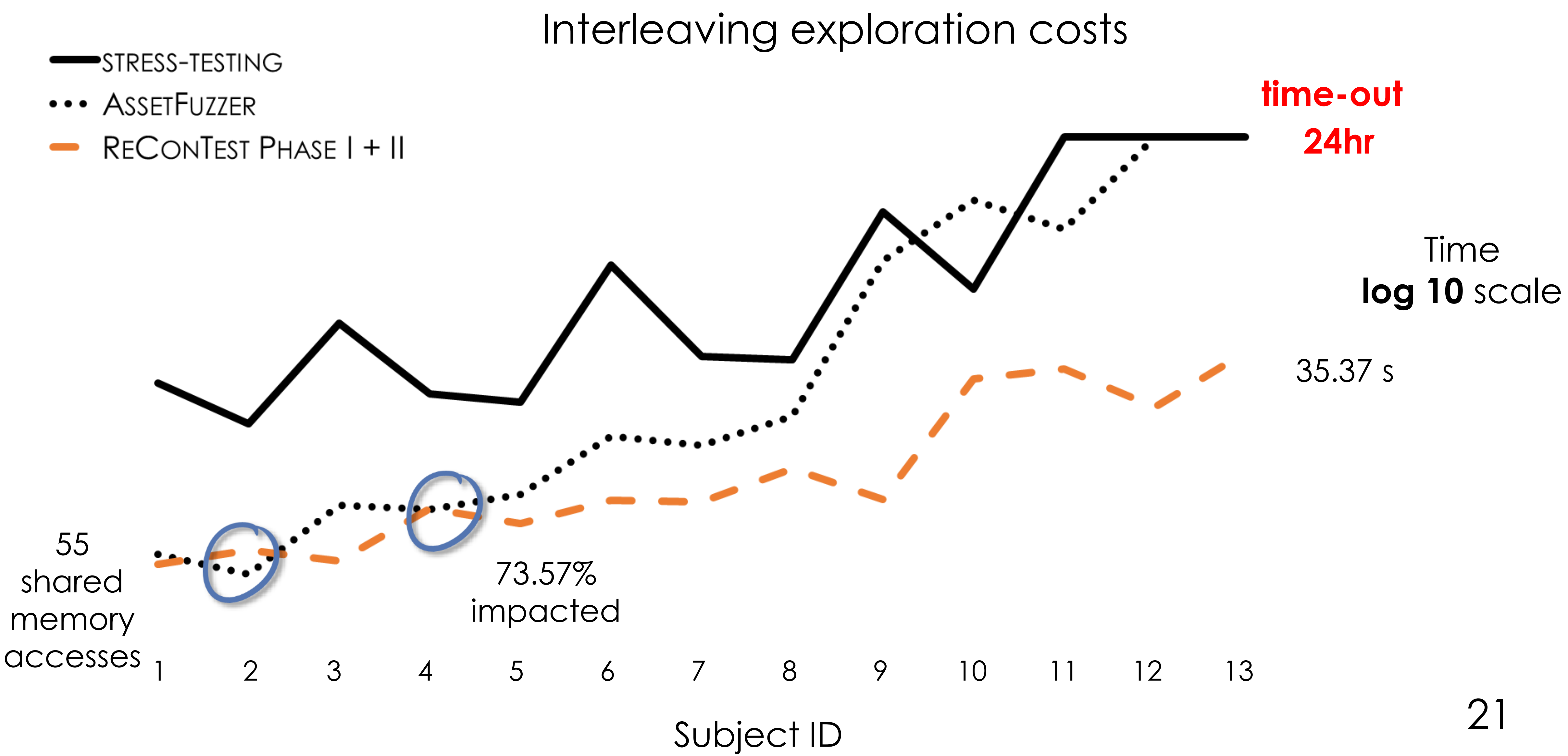
ID	RECONTEST Phase I + II	AssetFuzzer Lai et al. ICSE 2010	Stress testing
1	17 ms	24 ms	11.90 s
2	28 ms	12 ms	2.70 s
3	19 ms	145 ms	10.20 s
4	127 ms	122 ms	8.15 s
5	74 ms	214 ms	5.83 s
6	169 ms	1.72 s	840 s
7	159 ms	1.24 s	31 s
8	540 ms	3.53 s	27 s
9	177 ms	961 s	1.5 hr
10	13.5 s	2.39 hr	352s
11	19.75 s	0.83 hr	**time-out**
12	5 s	**time-out**	**time-out**
13	35.37 s	**time-out**	**time-out**

24 hours

RQ2: Efficiency (cont.)



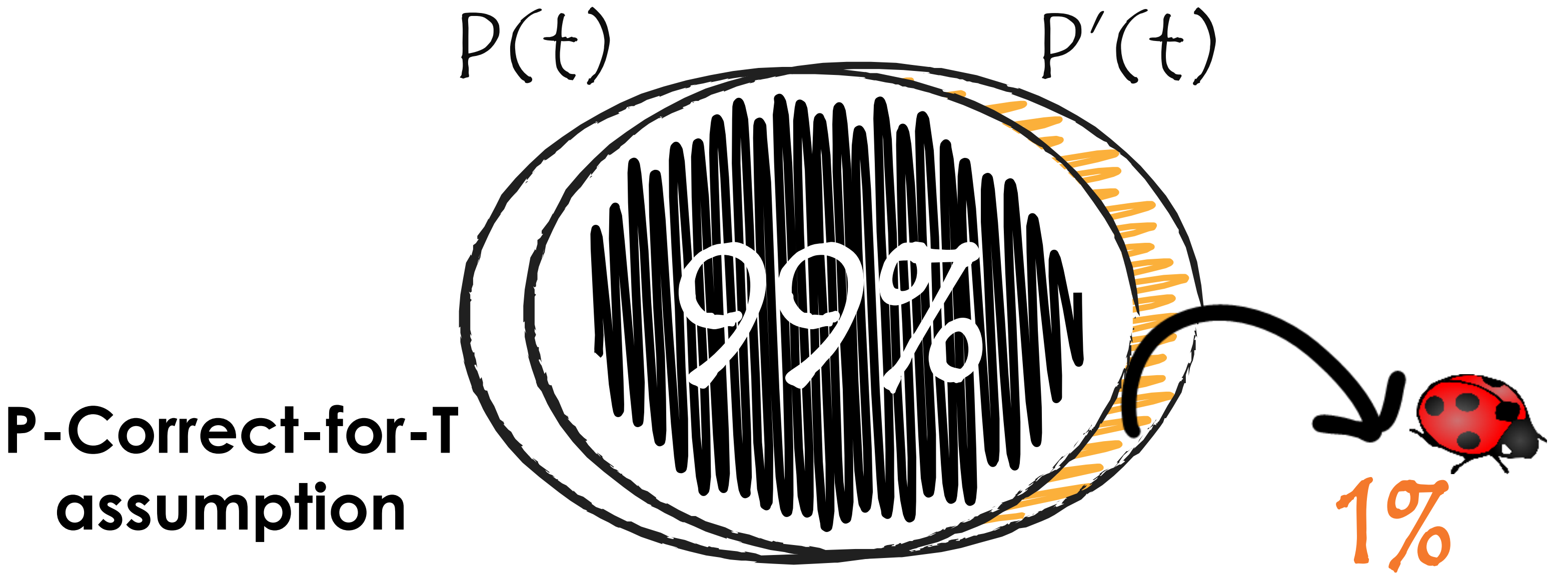
RQ2: Efficiency (cont.)



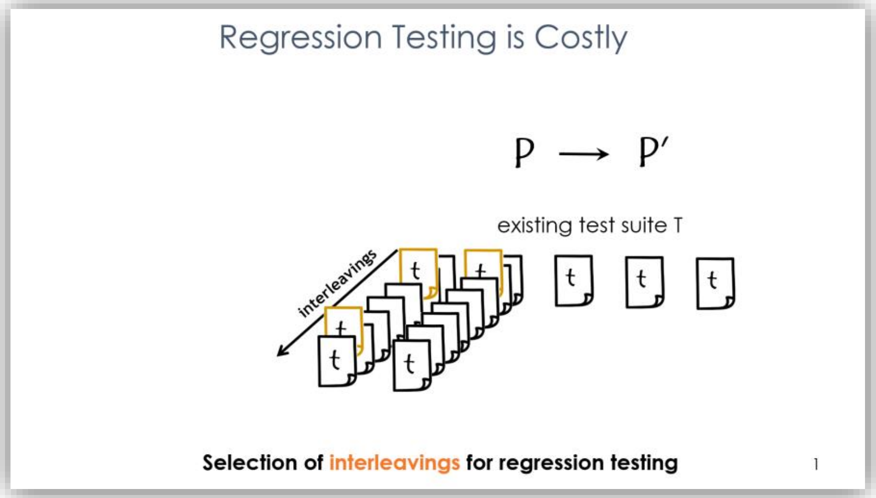
RQ3: Correctness

Does our regression technique practically achieve safety when selecting interleavings?

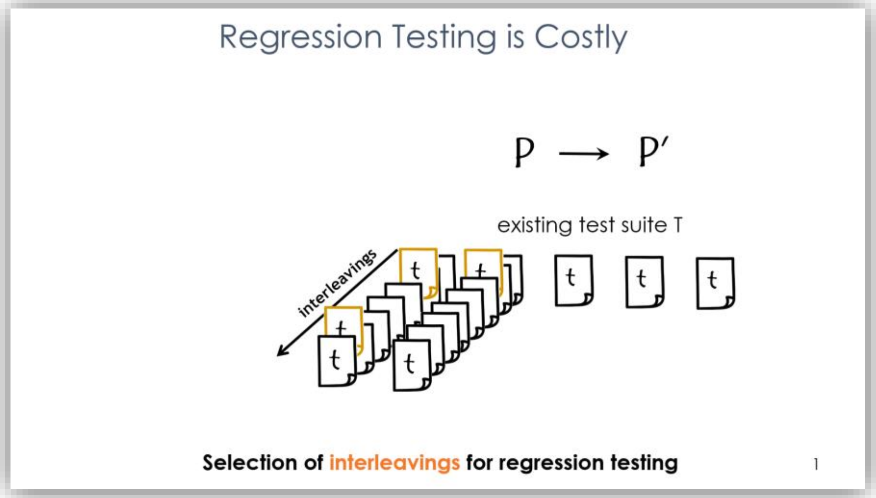
- After pruning the infeasible problematic interleavings, RECONTEST did not miss any regression concurrency bugs.

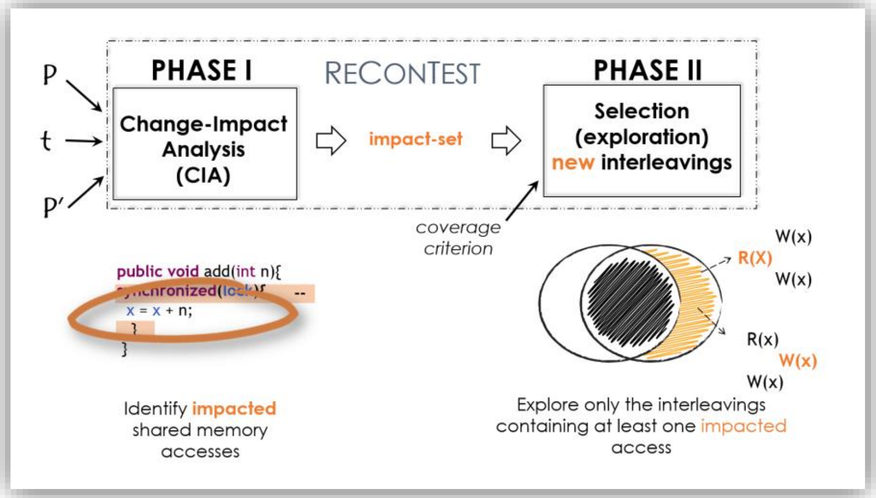


Conclusions

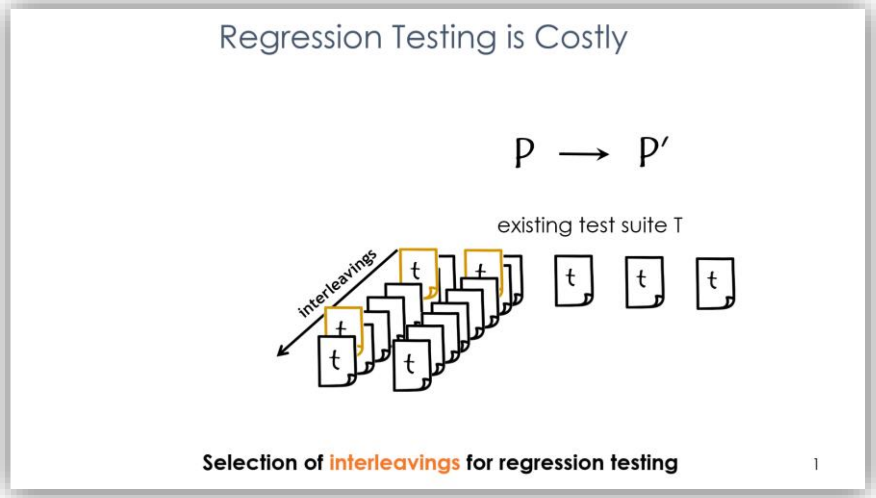


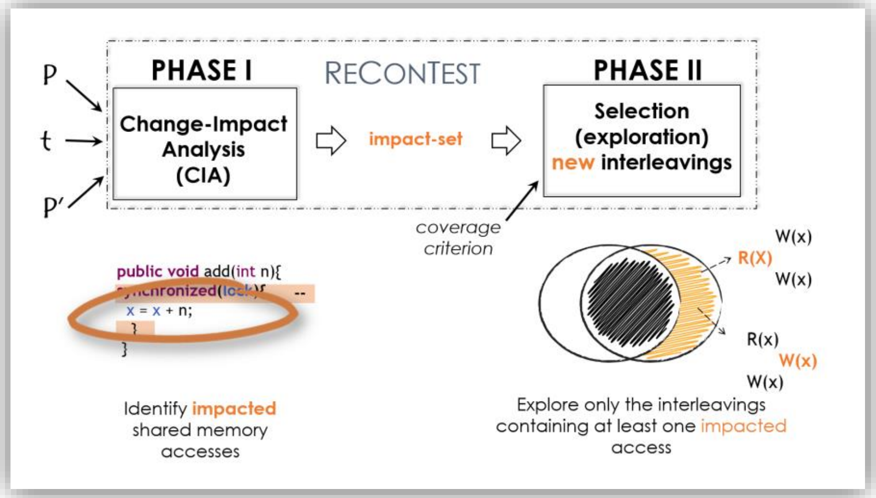
Conclusions

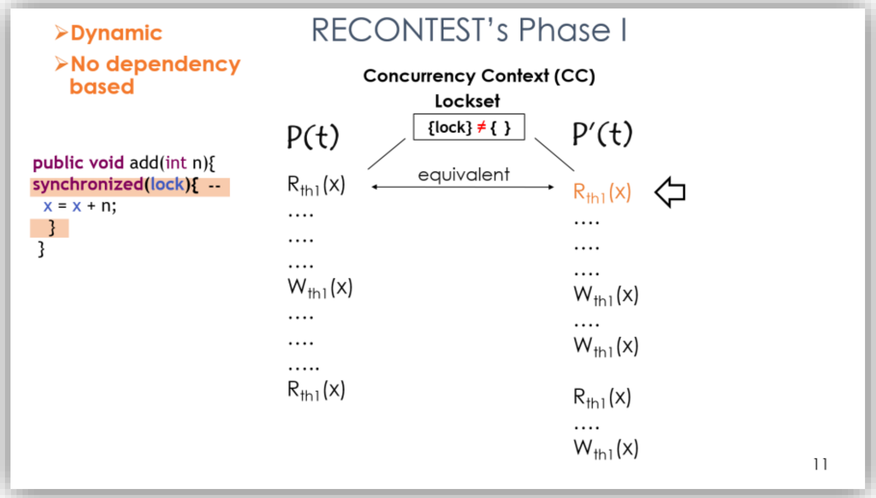




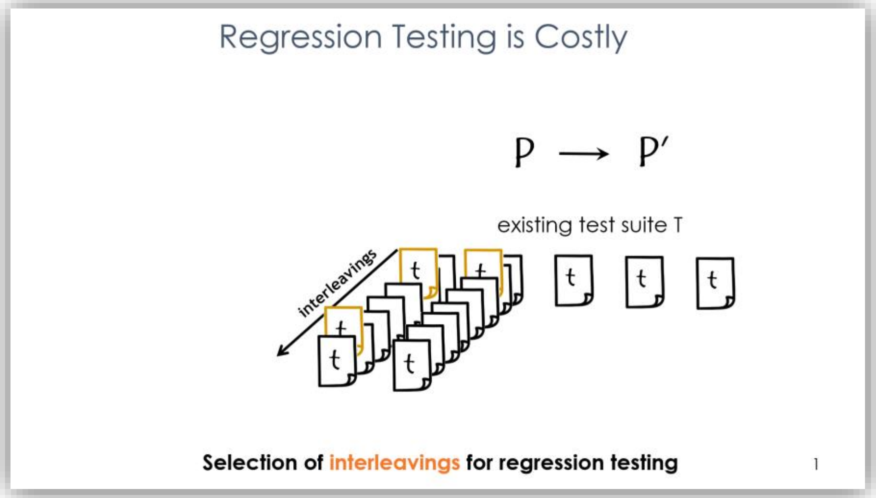
Conclusions

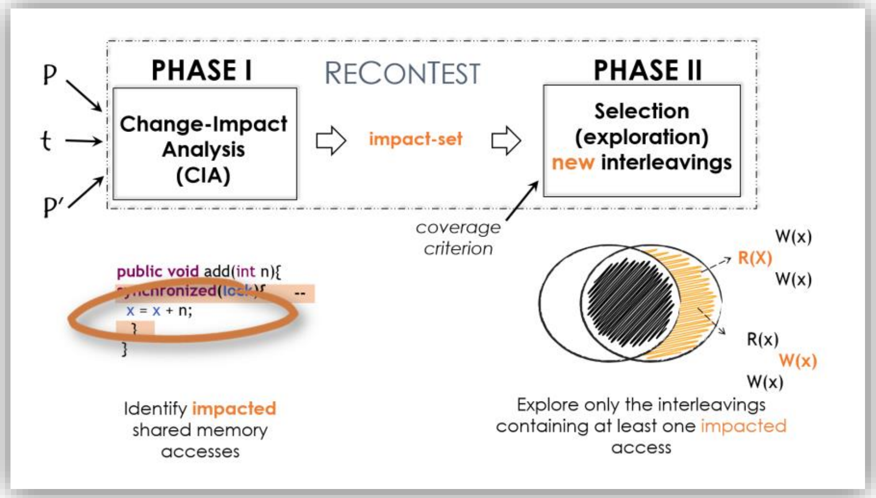


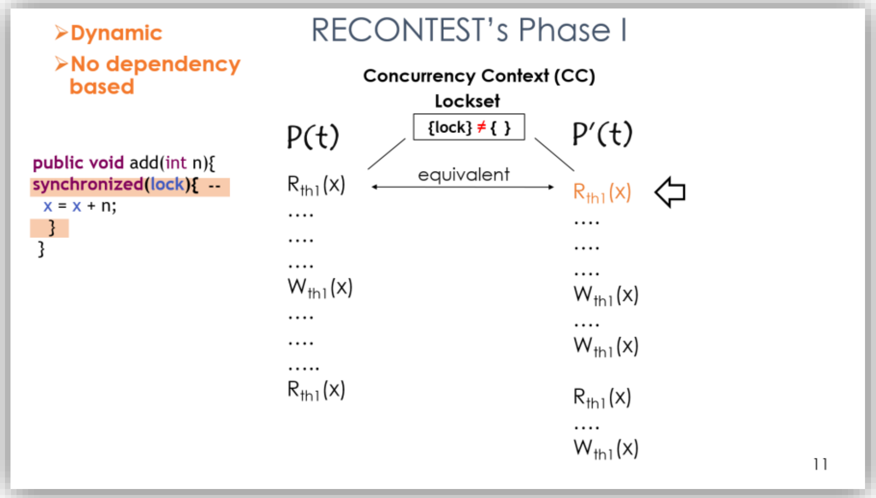


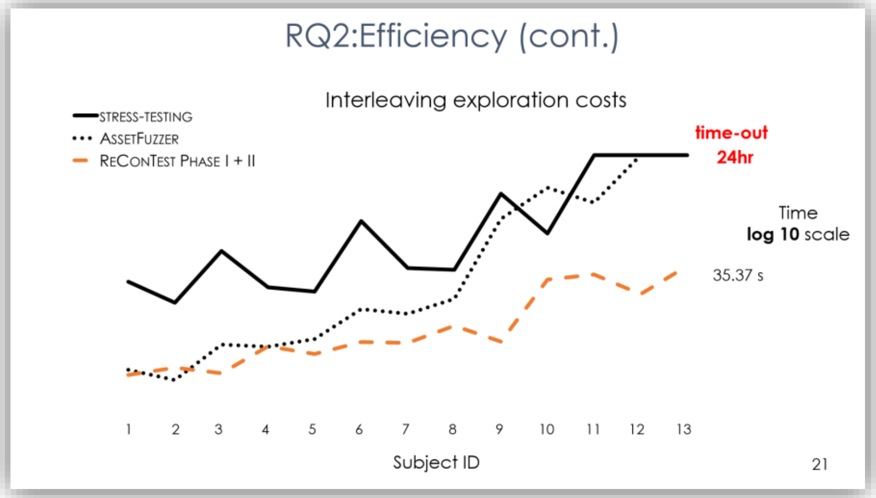


Conclusions

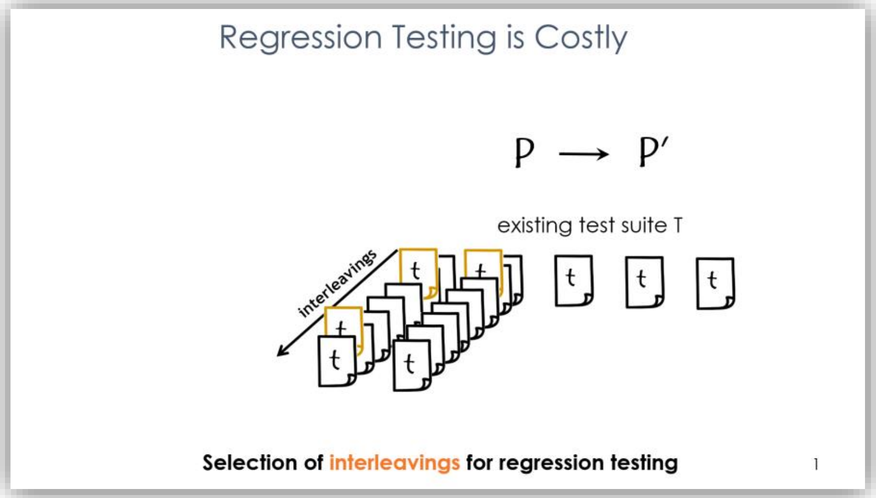


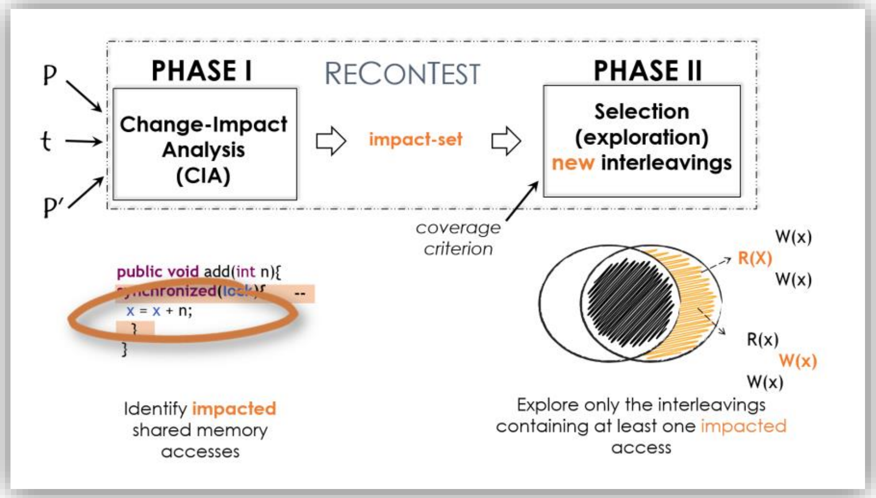


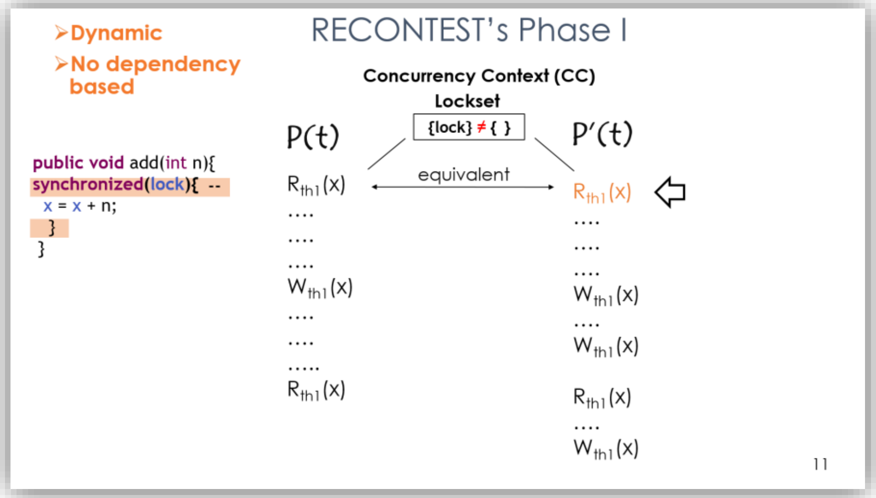


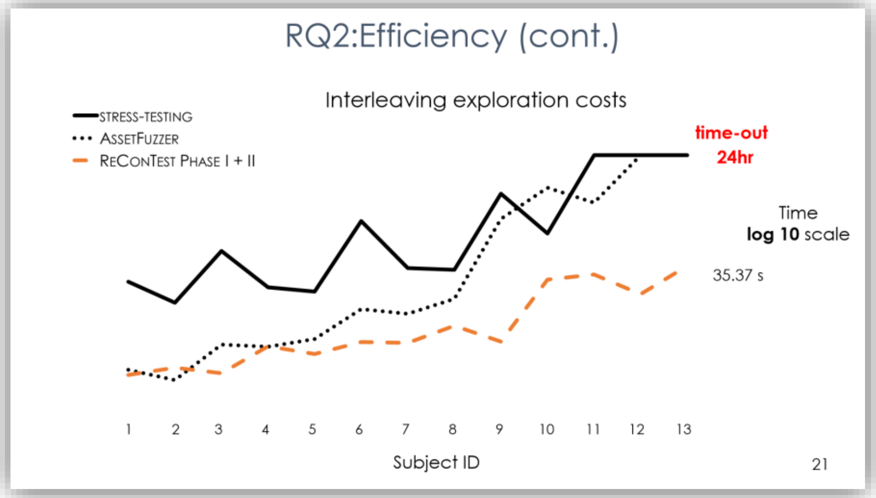


Conclusions



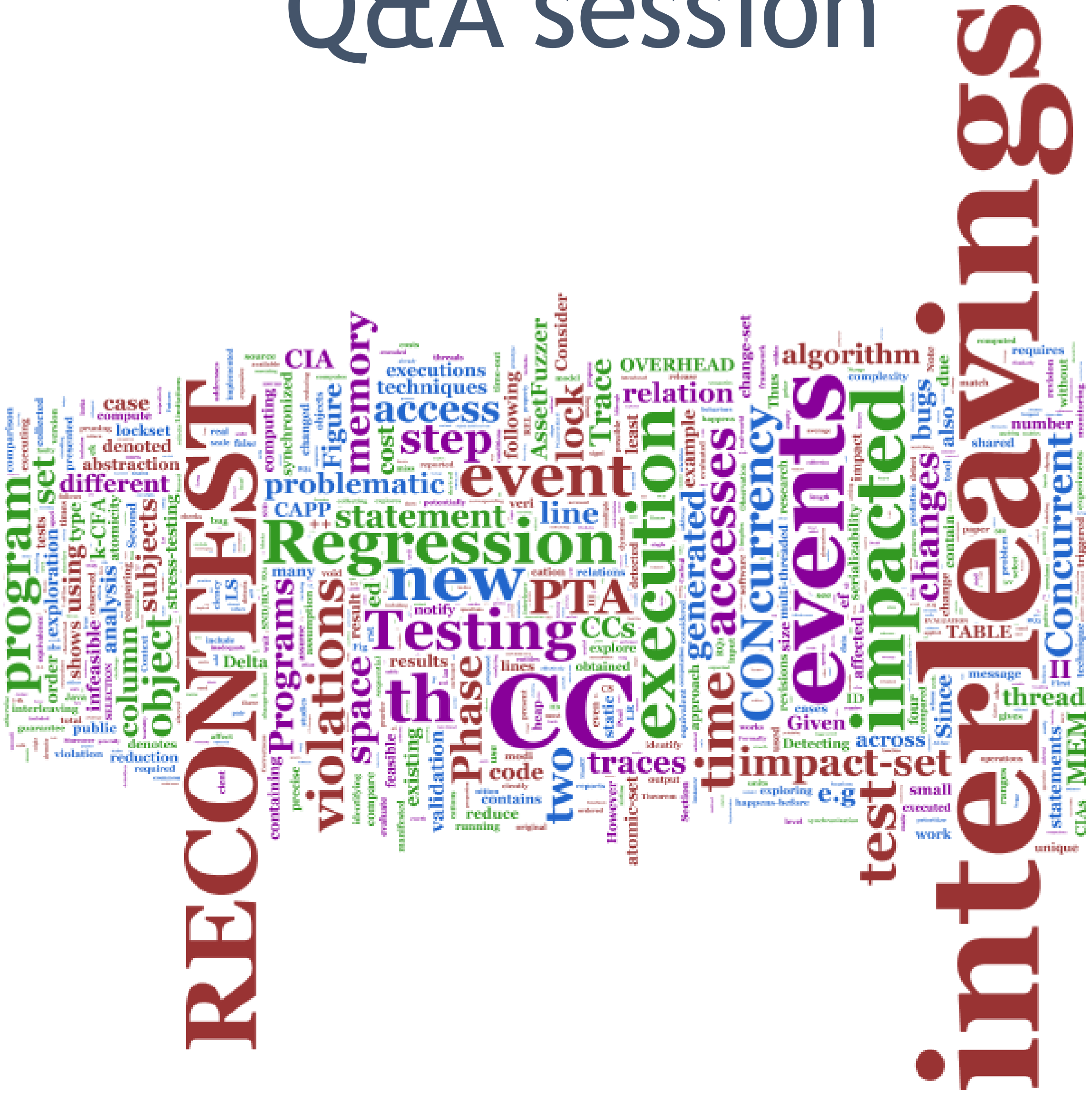






➢**Future work:** Test suite augmentation for concurrent programs

Q&A session



interleavings
RECONTEST
CC
events
impacted
Regression
Testing
execution
event
new
PTA
program
violations
Programs
accesses
concurrency
impact-set
test
time
memory
access
step
problematic
statement
Phase
traces
two
space
object
subjects
Concurrent
changes
algorithm
relation
OVERHEAD
AssetFuzzer
Consider
techniques
executions
CIA
different
abstraction
denoted
lockset
case
compute
CCs
generated
lock
Trace
line
thread
Given
across
TABLE
MEM
CIAs

Backup Slides

Sensitivity of the trace

- Executing P′(t) multiple times could yield to different execution traces
  - Fluctuation in atomicity violation detection among multiple runs with the same input is at most 0.4% [Deng at al. OOPSLA 2013]

P′(t)

thread1	thread2
W(x)	R(z)
R(x)	W(x)
….	…..

thread1	thread2
W(x)	R(z)
R(x)	W(x)
….	…..

thread1	thread2
W(x)	R(z)
W(x)	W(x)
R(x)	…..
…..	

RECONTEST's Phase II (cont.)

- Explore **only** problematic interleavings containing impacted accesses
  - **Predictive Trace Analysis (PTA)** (e.g, Sen et al. FMOODS 2005)
    - RECONTEST starts the exploration from the impact-set and then it finds accesses to complete a partially matched problematic pattern.

<?>

R(x)

<?>

worst time complexity

$$O(n^3)$$

original PTA

$$O(I \cdot n^2)$$

RECONTEST

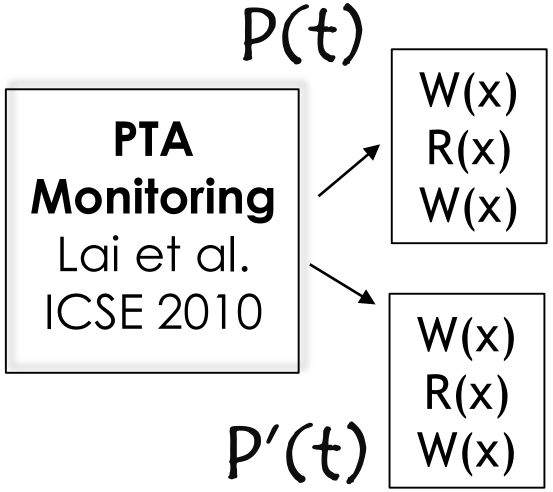
I = impact set

n = execution trace length P'(t)

Naïve solution does not reduce interleaving exploration costs

1. Detect all potential problematic interleavings in P'(t)
2. Pruning those that do not contain **impacted** accesses

Execution Traces



execution traces
with Concurrency contexts

P′(t)

ID	#shared memory accesses	time	overhead
1	49	2.93s	1.88x
2	55	2.44s	1.91x
3	130	2.79s	1.86x
4	274	3.95s	1.76x
5	310	4.82s	1.41x
6	381	2.50s	1.83x
7	1,909	6.81s	1.34x
8	2,527	6.08s	1.15x
9	3,447	7.26s	3.14x
10	17 K	39.08s	15.88x
11	39 K	34.22s	14.77x
12	150 K	20.22s	3.17x
13	570 K	90.55s	50.72x

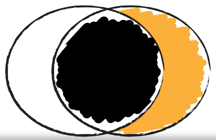
RECONTEST

RECONTEST PHASE I
CIA

W(x)
R(x)
W(x)

impact-set

RECONTEST PHASE II



PTA Validation
Lai et al.
ICSE 2010

ID	#shared memory accesses P'(t)	% impacted	Time Phase I	interleavings tested	Time Phase II
1	49	53.06%	9 ms	162	8 ms
2	55	16.36%	10 ms	157	18 ms
3	130	6.15%	16 ms	12	3 ms
4	274	73.57%	42 ms	328	85 ms
5	310	10.97%	27 ms	5,322	47 ms
6	381	1.31%	19 ms	74	150 ms
7	1,909	0.20%	104 ms	1,102	55 ms
8	2,527	1.50%	106 ms	151,214	434 ms
9	3,447	0.10%	122 ms	1,053	55 ms
10	17 K	0.03%	409 ms	32,846	13.10 s
11	39 K	0.19%	15.10 s	7,430	4.65 s
12	150 K	0.14%	3.07 s	326,603	1.93 s
13	570 K	0.02%	7.99 s	64,670	27.38 s

RQ2:Efficiency

Does the overhead of pre-computing the impact-set out-weight the reduction in test effort?

CIA
+
Exploring problematic interleavings that contain at least one impacted access
+
pruning infeasible ones with the Lockset & HB analysis

RECONTEST Phase I + II	Reduction AssetFuzzer Lai et al. ICSE 2010	Reduction Stress testing
17 ms	1.41x	4.06x
28 ms	**0.43x**	1.09x
19 ms	7.63x	36.31x
127 ms	**0.96x**	2.28x
74 ms	2.89x	1.19x
169 ms	10.20x	314.72x
159 ms	7.81x	4.45x
540 ms	6.54x	4.08x
177 ms	5,429x	760.24x
13.5 s	638x	6.69x
19.75 s	152x	**>1,601x**
5 s	**>17,245x**	**>3,425x**
35.37 s	**>2,442x**	**>686x**

Includes overhead collecting the trace P'(t) which **is not** required by stress testing

Trace collection
2.93s
2.44s
2.79s
3.95s
4.82s
2.50s
6.81s
6.08s
7.26s
39.08s
34.22s
20.22s
90.55s

time-out 24 hours

RQ2:Efficiency

Does the overhead of pre-computing the impact-set out-weights the reduction in test effort?

time-out 24 hours

RECONTEST Phase I + II	**Lai et al. ICSE 2010**	**reduction**
17 ms	24 ms	1.41x
28 ms	12 ms	0.43x
19 ms	145 ms	7.63x
127 ms	122 ms	0.96 x
74 ms	214 ms	2.89x
169 ms	1.72 s	10.20x
159 ms	1.24 s	7.81x
540 ms	3.53 s	6.54x
177 ms	961 s	5,429x
13.5 s	2.39 hr	638x
19.75 s	0.83 hr	152x
5 s	**time-out**	>17,245x
35.37 s	**time-out**	>2,442x

RECONTEST Phase I + II + overhead P'(t)	**Stress-Testing**	**reduction**
2.95s	11.90s	4.06x
2.47s	2.70s	1.09x
2.81s	10.20s	36.31x
3.51s	8.15s	2.28x
4.90s	5.83s	1.19x
2.67s	840s	314.72x
6.97s	31s	4.45x
6.62s	27s	4.08x
7.44s	1.5hr	760.24x
52.59s	352s	6.69x
53.97s	**time-out**	>1,601x
25.23s	**time-out**	>3,425x
126s	**time-out**	>686x