Lightweight Data Race Detection for Production Runs

Swarnendu Biswas, UT Austin
Man Cao, Ohio State University
Minjia Zhang, Microsoft Research
Michael D. Bond, Ohio State University
Benjamin P. Wood, Wellesley College

CC 2017

A Java Program With a Data Race

```
Object X = null;
boolean done= false;
```

Thread T1

```
X = new Object();
done = true;
```

Thread T2

```
while (!done) {}
X.compute();
```

```
Object X = null;
boolean done= false;
```

Thread T1

write

```
X = new Object();
done = true;
```

Thread T2

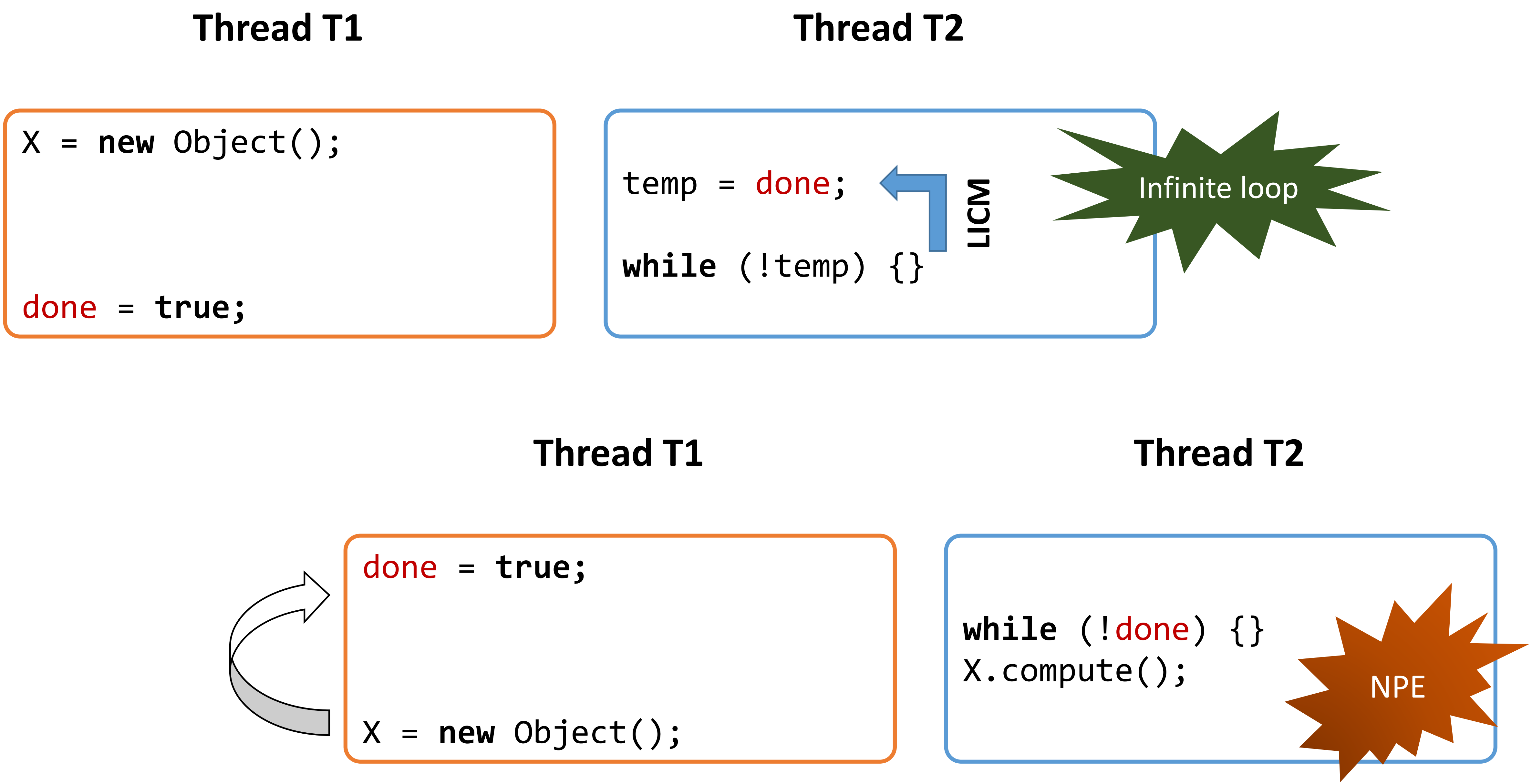
read

```
while (!done) {}
X.compute();
```

Data race

Conflicting accesses – two threads access the same shared variable where at least one access is a write

Concurrent accesses – accesses are not ordered by synchronization operations


Thread T1
Thread T2
X = new Object();
done = true;
temp = done;
while (!temp) {}
LICM
Infinite loop
Thread T1
Thread T2
done = true;
X = new Object();
while (!done) {}
X.compute();
NPE

Data Races are Evil

- Challenging to reason about the correctness of racy executions
    - May unpredictably break code
- Lack of semantic guarantees in most mainstream multithreaded languages
- Usually indicate other concurrency errors
    - Atomicity, order, or sequential consistency violations

- S. Adve and H. Boehm. Memory Models: A Case for Rethinking Parallel Languages and Hardware. CACM 2010.
- S. Adve. Data Races Are Evil with No Exceptions: Technical Perspective. CACM 2010.

Far-Reaching Impact of Data Races

Therac-25 accidents, 1985-87

BUSINESS SOFTWARE
business

BUSINESS READY

Nasdaq's Facebook Glitch Came From Race Conditions

Joab Jackson
@Joab_Jackson

May 21, 2012 12:30 PM

The Nasdaq computer system that delayed trade notices of the Facebook IPO on Friday
change announced Monday. As a result of
n, the market expects to pay out US$13

nismatched Facebook share prices. About
ed, the exchange estimated.

research highlights

Technical Perspective
Data Races are Evil with No Exceptions

By Sarita Adve

EXPLOITING PARALLELISM HAS become the primary means to higher performance.

racy code. Java's safety requiremer
preclude the use of "undefined" beh

How to miscompile programs with "benign" data races

Hans-J. Boehm
HP Laboratories

Get Rid of Data Races!!!

Avoiding and/or eliminating data races **efficiently** is a challenging and unsolved problem

Data Race Detection on Production Systems

Avoiding and/or eliminating data races **efficiently** is a challenging and unsolved problem

No satisfactory solution to date

Data Race Detection Techniques

Static and predictive analyses

- Too many false positives, do not scale

Data Race Detection Techniques

Dynamic analysis	Lockset analysis	Expensive, reports many false positives
	Happens-before analysis	Sound and precise
		Expensive, not scalable, incurs space overhead
		Coverage limited to observed executions

sound – no missed races
precise – no false races

C. Flanagan and S. Freund. FastTrack: Efficient and Precise Dynamic Data Race Detection. PLDI 2009.

Existing Approaches for Data Race Detection on Production Runs

- **Happens-before-based sampling approaches**
  - E.g., LiteRace[1], Pacer[2]
  - Overheads are still too high for a reasonable sampling rate
    - Pacer with 3% sampling rate incurs 86% overhead!!!

1. D. Marino et al. LiteRace: Effective Sampling for Lightweight Data-Race Detection. PLDI 2009.
2. M. D. Bond et al. Pacer: Proportional Detection of Data Races. PLDI 2010.

Existing Approaches for Data Race Detection on Production Runs

- **Happens-before-based sampling approaches**
  - E.g., LiteRace[1], Pacer[2]
  - Overheads are still too high for a reasonable sampling rate
    - Pacer with 3% sampling rate incurs 86% overhead!!!

- **RaceMob**[3]
  - Optimizes tracking of happens-before relations
  - Monitors only one race per run to minimize overhead
  - Cannot bound overhead, limited scalability and coverage

1. D. Marino et al. LiteRace: Effective Sampling for Lightweight Data-Race Detection. PLDI 2009.
2. M. D. Bond et al. Pacer: Proportional Detection of Data Races. PLDI 2010.
3. B. Kasikci et al. RaceMob: Crowdsourced Data Race Detection. SOSP 2013.

Existing Approaches for Data Race Detection on Production Runs

- **DataCollider**[4]
  - Tries to collide racy accesses, synchronization oblivious
  - Samples accesses, and uses hardware debug registers for performance
  - Dependence on debug registers
    - Not portable, and may not scale well
    - Few debug registers
    - Cannot bound overhead

4. J. Erickson et al. Effective Data-Race Detection for the Kernel. OSDI 2009.

Outline

- ~~Data Races~~
  - ~~Problems and Challenges~~
  - ~~Data Race Detection in Production Systems~~
  - ~~Drawbacks of existing approaches~~
- Our contribution: efficient, complementary analyses
  - RaceChaser: Precise data race detection
  - Caper: Sound data race detection

Our Insight

Decouple data race detection into two lightweight and complementary analysis

Our Contributions

Decouple data race detection into two lightweight and complementary analysis

Efficient

RaceChaser – Precise data race detector

- Under-approximates data races

Can miss true data races

Caper – Dynamically sound data race detector

- Over-approximates data races

Can report false data races

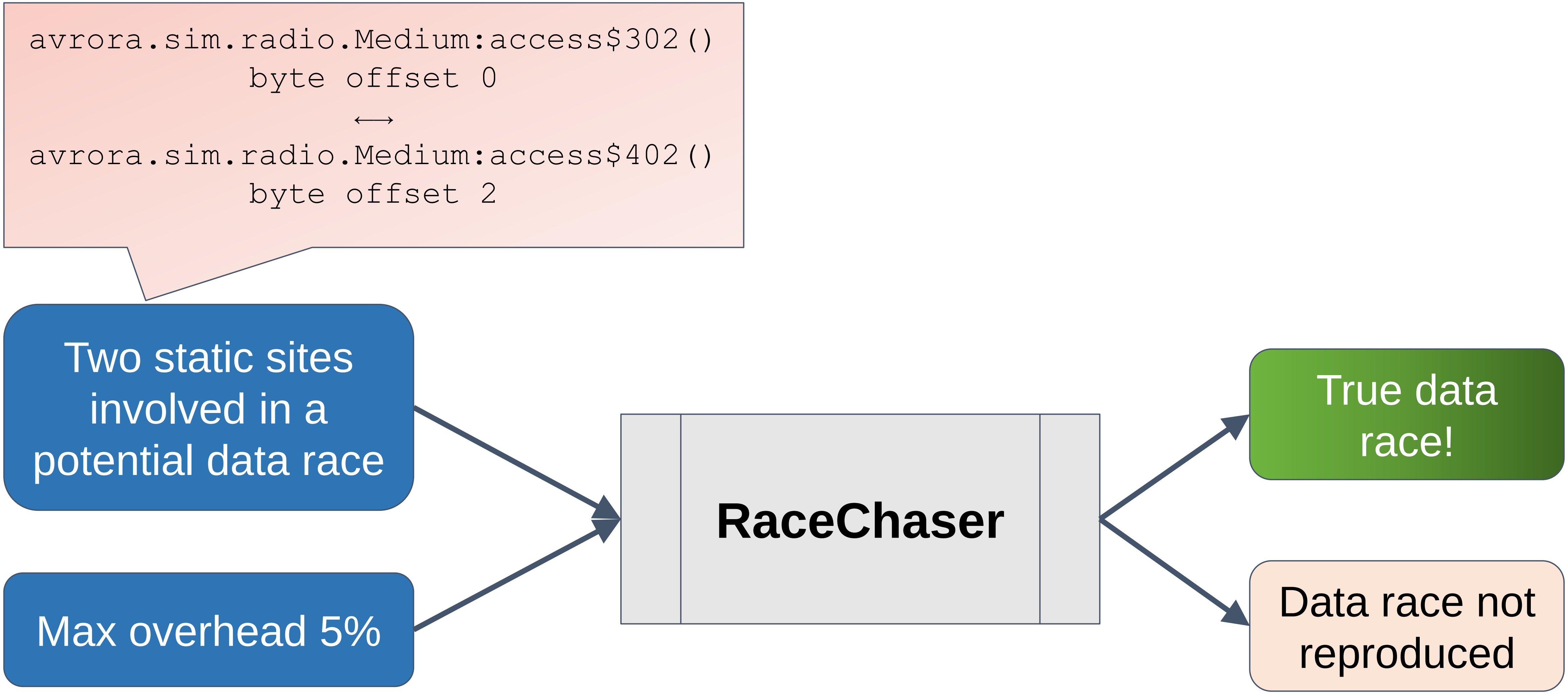
RaceChaser: Precise Data Race Detection

Desired Properties:

- Performance and Scalability **?**
- Bounded time and space overhead **?**
- Coverage and Portability **?**

Design:

- Monitor one data race (two source locations) per run
- Use collision analysis
- Bound overhead introduced



RaceChaser Algorithm

Instrumenting Racy Accesses

`avrora.sim.radio.Medium: access$302() byte offset 0`

`avrora.sim.radio.Medium: access$402() byte offset 2`

- Limited to one potential race pair

Randomly Sample Racy Accesses

- Use frequency of samples taken

  and

  Compute overhead introduced by waiting

`avrora.sim.radio.Medium: access$302() byte offset 0`

`avrora.sim.radio.Medium: access$402() byte offset 2`

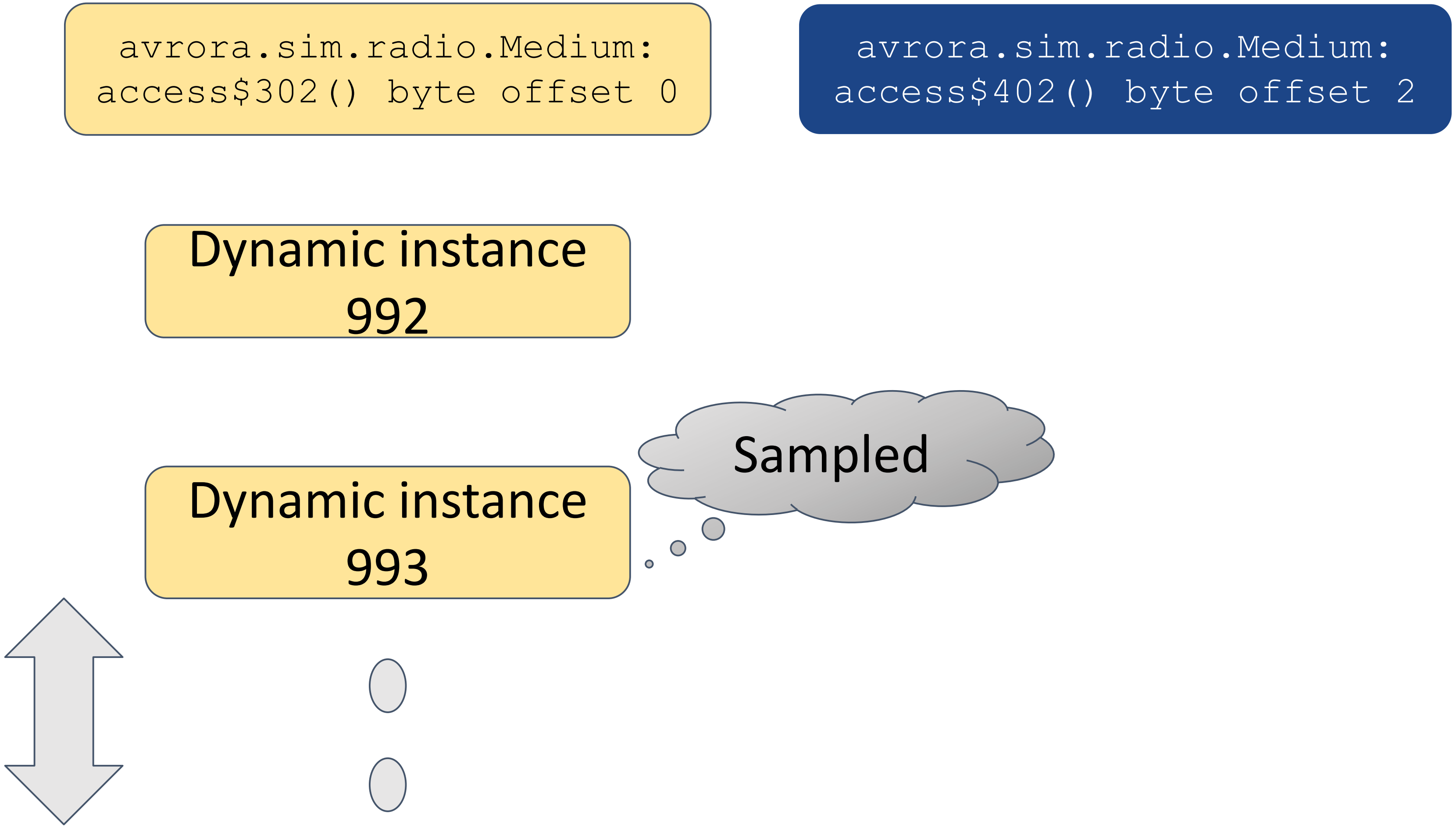
Dynamic instance 992

Dynamic instance 993

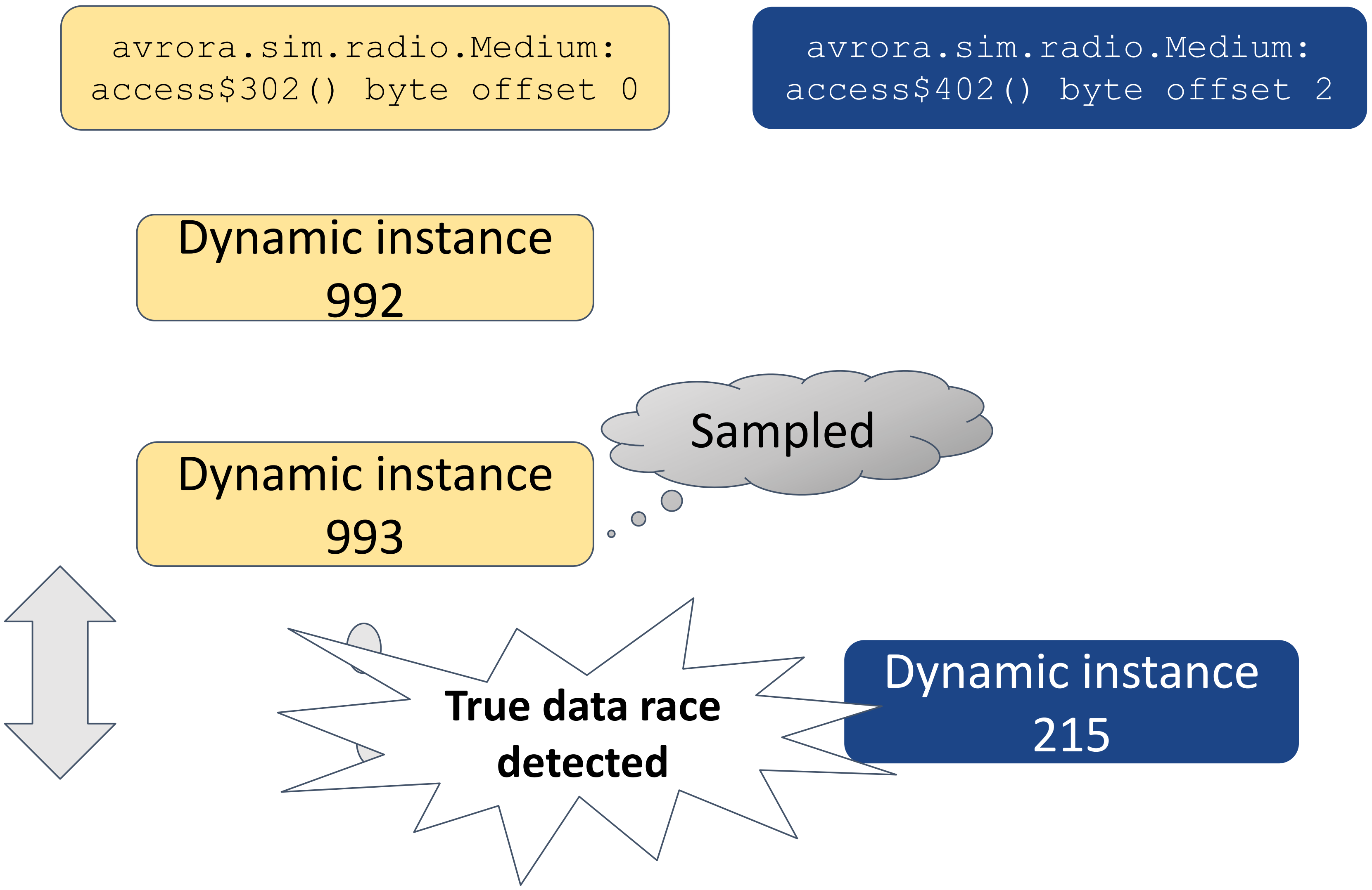
Sampled

Try to Collide Racy Accesses

- Block thread for some time

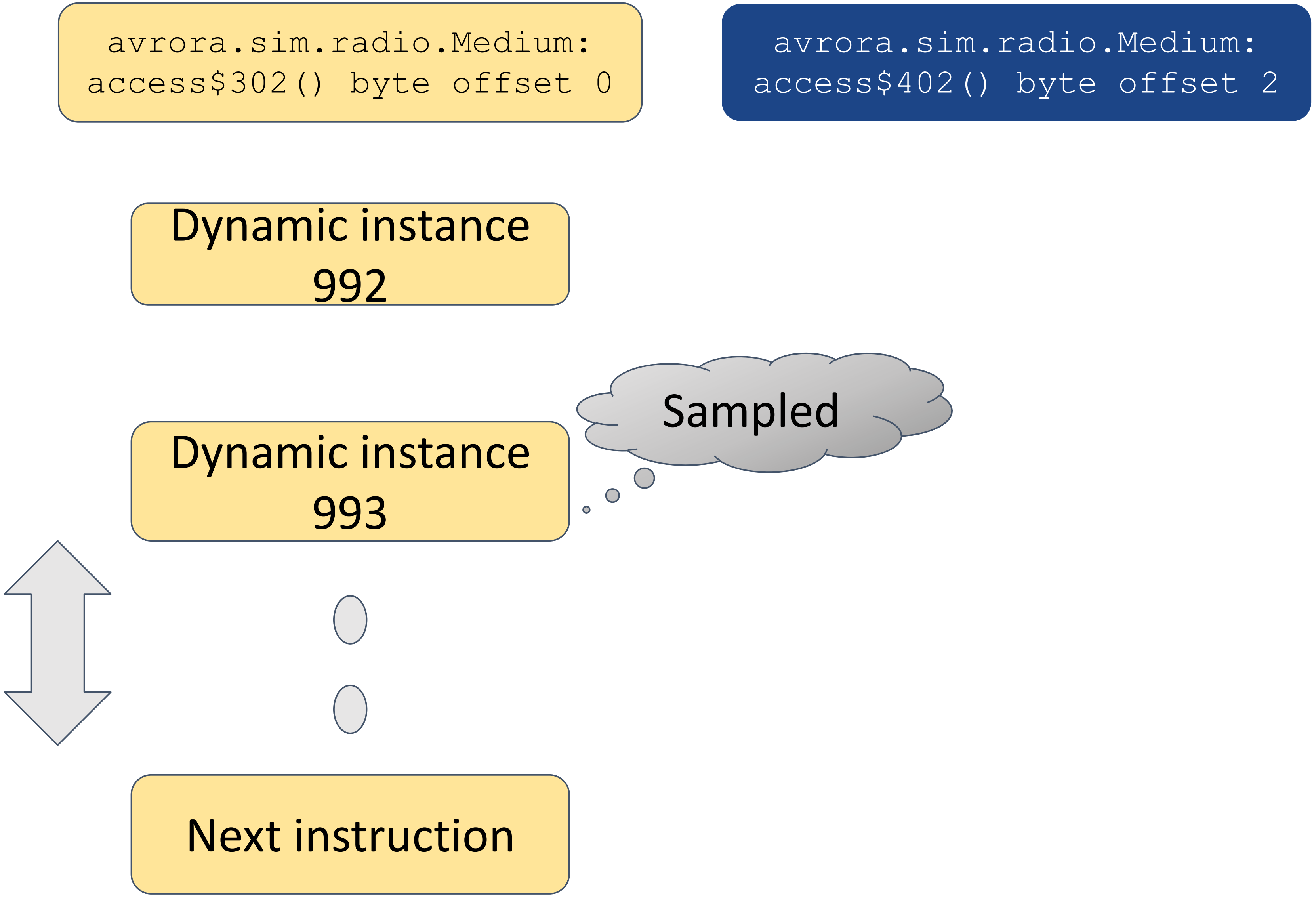


Collision is Successful



Collision is Unsuccessful

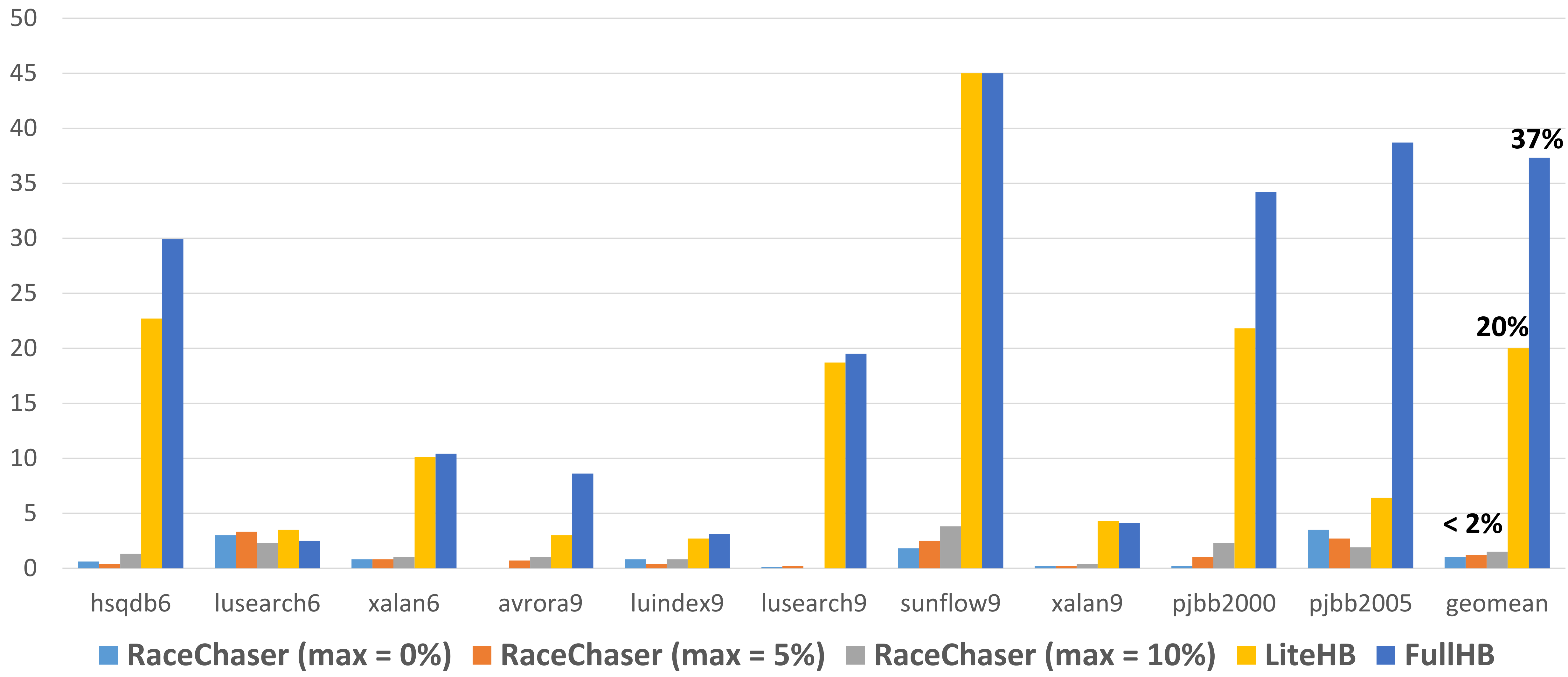
- Thread unblocks, resets the analysis state, and continues execution



Evaluation of RaceChaser

- Implementation is publicly available
  - Jikes RVM 3.1.3
- Benchmarks
  - Large workload sizes of DaCapo 2006 and 9.12-bach suite
  - Fixed-workload versions of SPECjbb2000 and SPECjbb2005
- Platform
  - 64-core AMD Opteron 6272

Run-time Overhead (%) of RaceChaser



adapted from

1. B. Kasikci et al. RaceMob: Crowdsourced Data Race Detection. SOSP 2013.

Effectiveness of RaceChaser

- Collision analysis can potentially detect data races that are hidden by spurious happens-before relations

- Data race coverage of collision analysis depends on the perturbation and the delay
    - Prior studies seem to indicate that data races often occur close in time

- RaceChaser did as well as RaceMob/LiteHB over a number of runs

Outline

- ~~Data Races~~
  - ~~Problems and Challenges~~
  - ~~Data Race Detection in Production Systems~~
  - ~~Drawbacks of existing approaches~~
- Our contribution: efficient, complementary analyses
  - ~~RaceChaser: Precise data race detection~~
  - Caper: Sound data race detection

Sound, Efficient Data Race Detection

Options

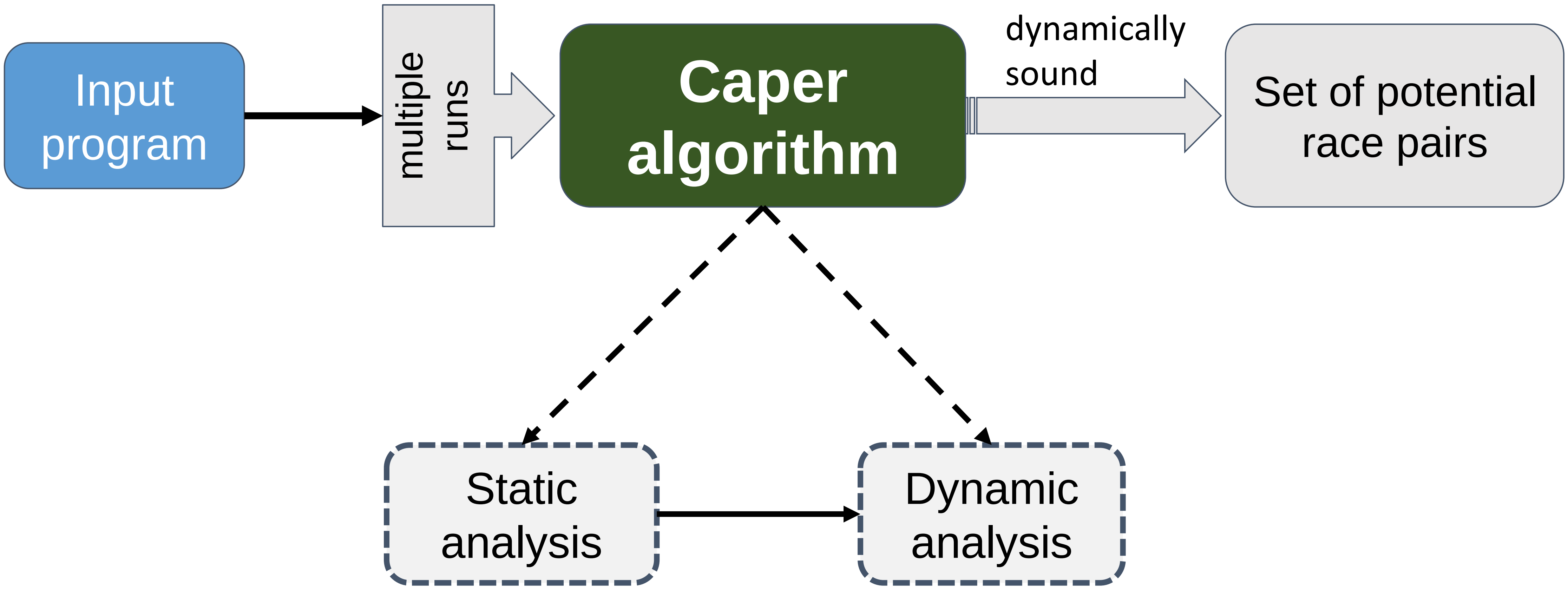
Use static analysis offline

Too many false positives

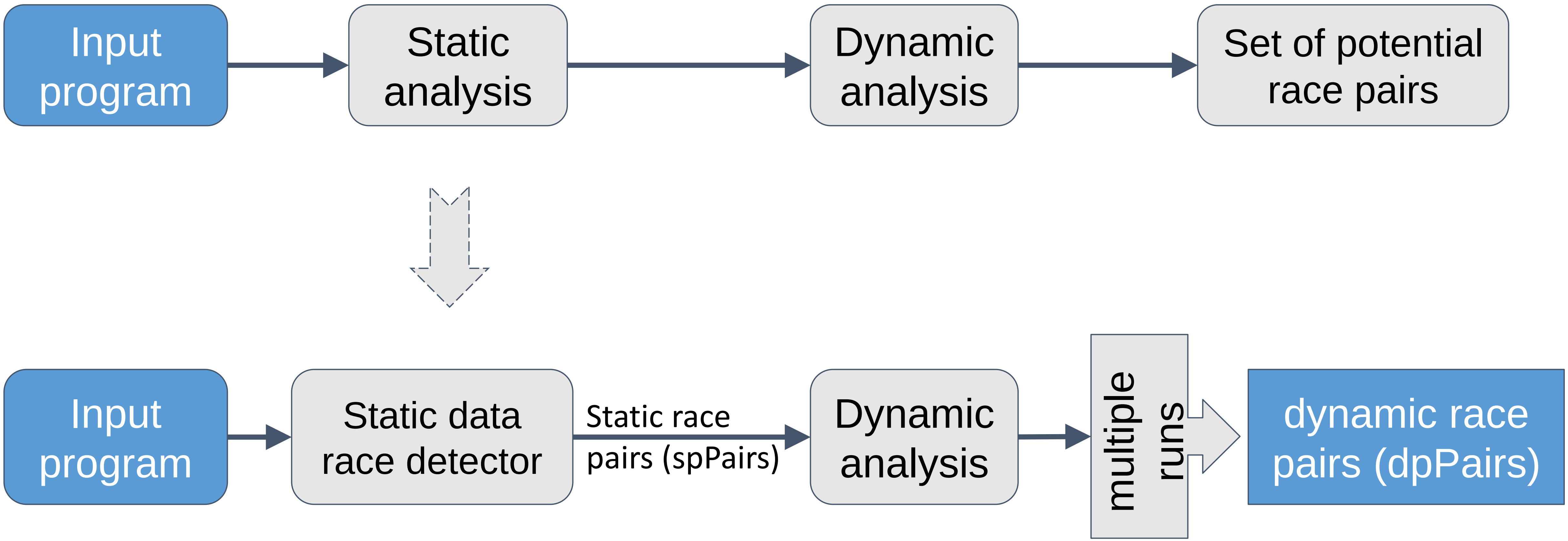
Use dynamic analysis online

Efficient enough for production runs?

Caper: Sound Data Race Detection



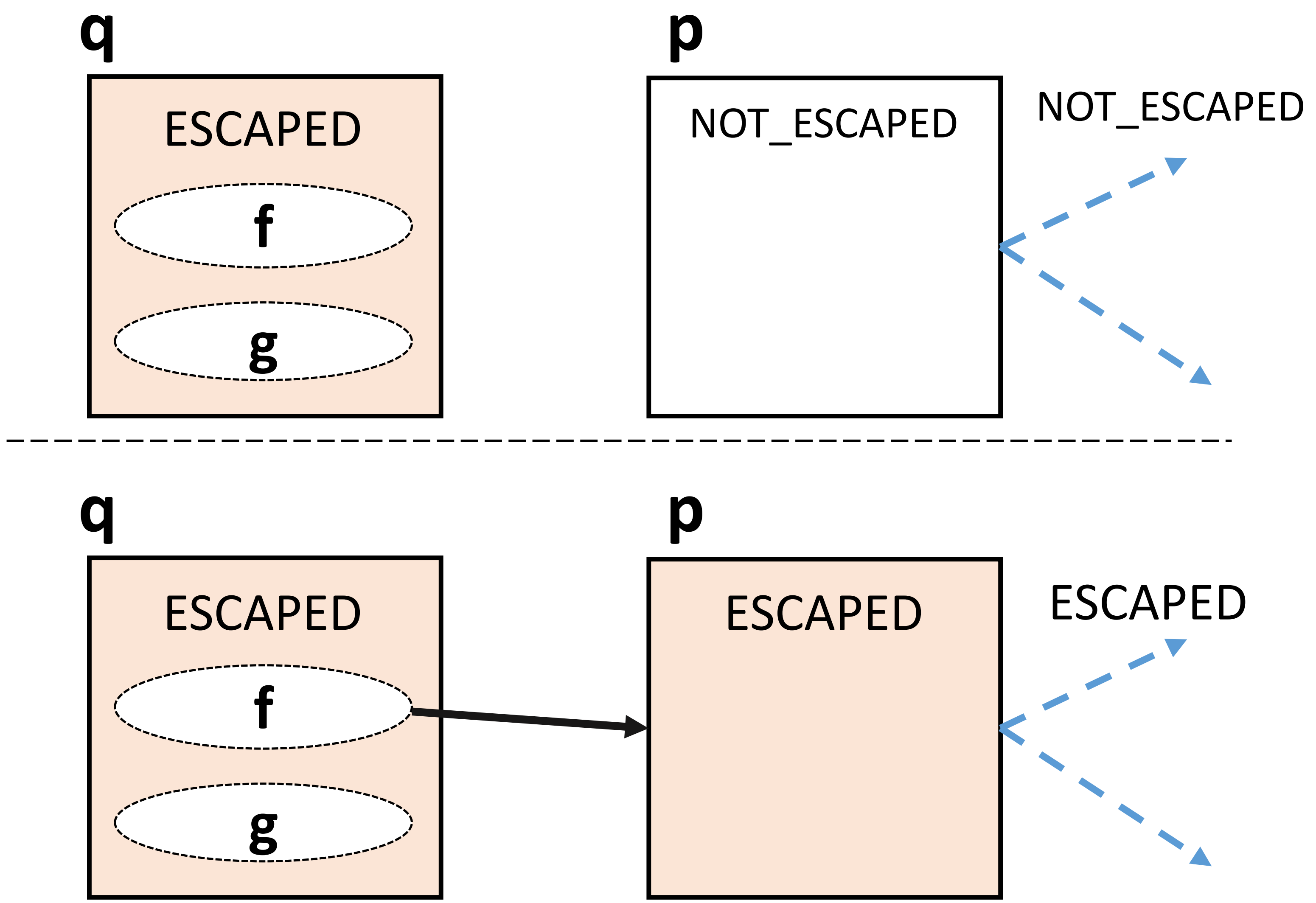
Caper Algorithm



Sound Dynamic Escape Analysis for Data Race Detection

Reachability-based analysis

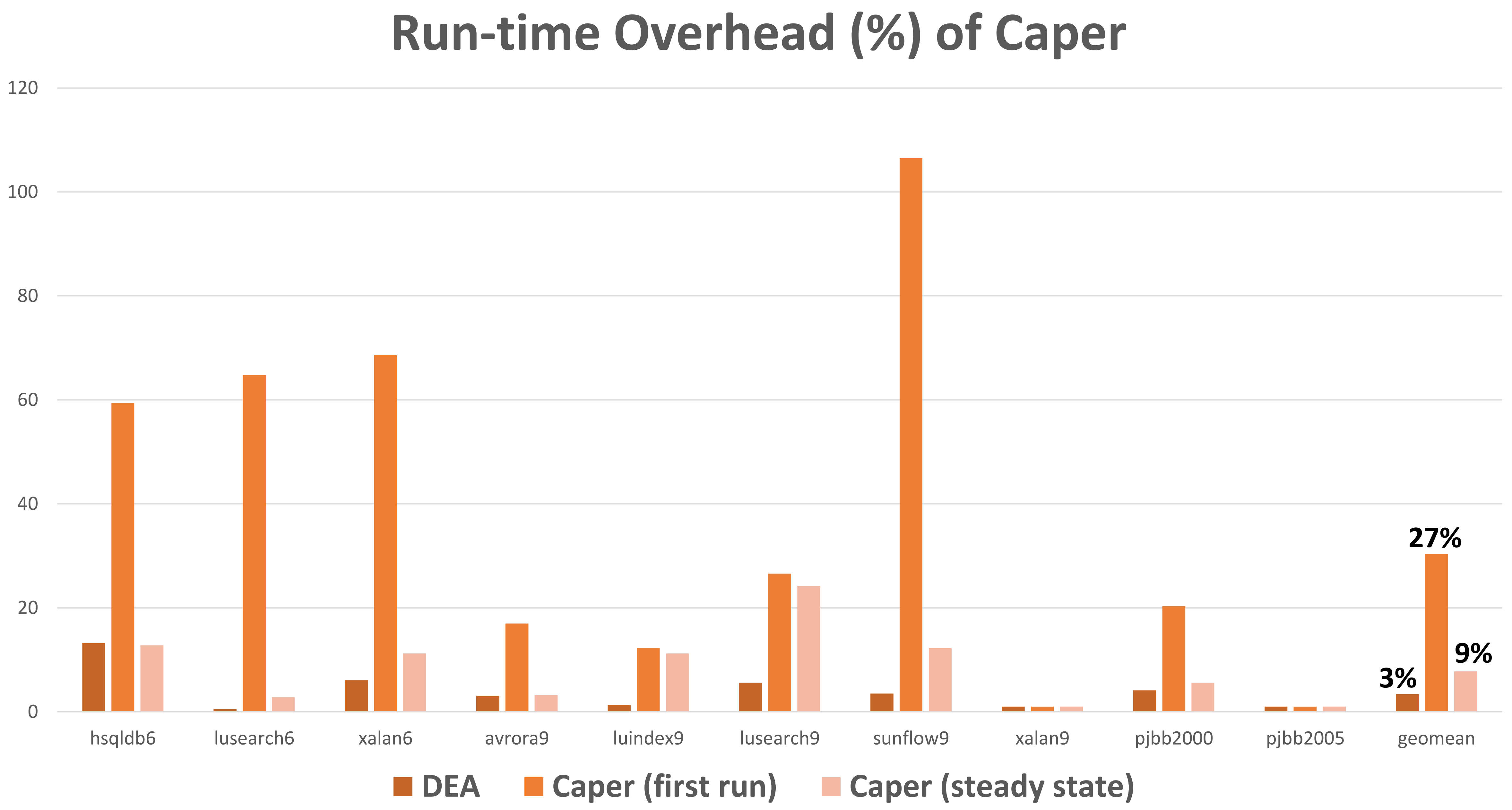
`q.f = p`



Caper's Dynamic Analysis

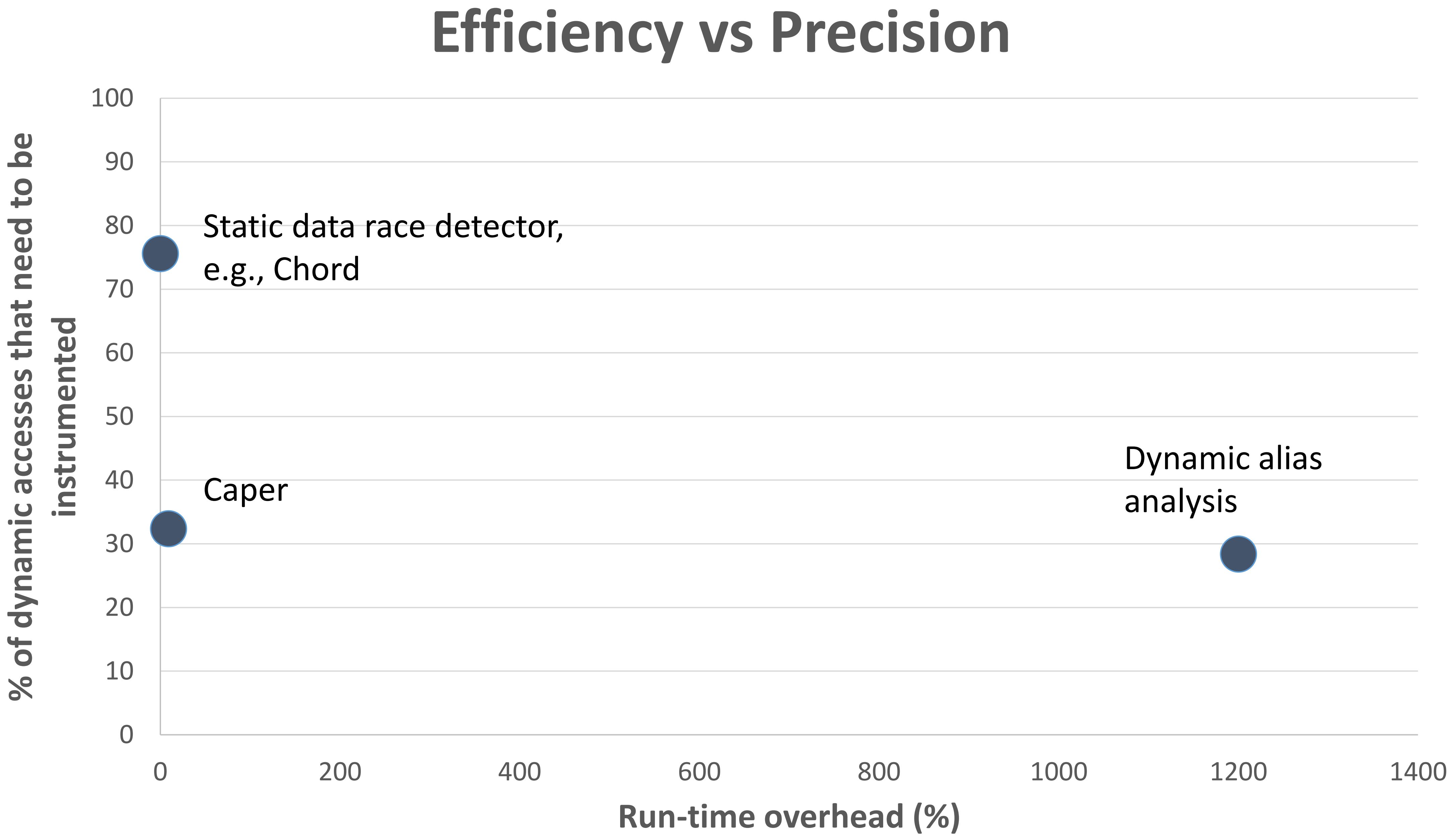
$$\text{deSites} = \{\, s \mid (\exists\, s' \mid \langle s, s' \rangle \in \text{spPairs} \cup \text{dpPairs}) \wedge s \text{ escaped in an analyzed execution} \,\}$$

$$\text{dpPairs} = \{\, \langle s_1, s_2 \rangle \mid s_1 \in \text{deSites} \wedge s_2 \in \text{deSites} \,\}$$


Run-time Overhead (%) of Caper
120
100
80
60
40
20
0
27%
9%
3%
hsqldb6
lusearch6
xalan6
avrora9
luindex9
lusearch9
sunflow9
xalan9
pjbb2000
pjbb2005
geomean
DEA
Caper (first run)
Caper (steady state)

Effectiveness of Caper

	Sound static data race detector	Caper	Dynamic alias analysis
hsqldb6	**212,205**	**1,612**	**757**
lusearch6	4,692	302	292
xalan6	**83,488**	**1,241**	**581**
avrora9	**61,193**	**19,941**	**570**
luindex9	10,257	192	193
lusearch9	7,303	34	39
sunflow9	28,587	200	1,086
xalan9	20,036	1,861	600
pjbb2000	29,604	11,243	1,679
pjbb2005	2,552	984	447



Efficiency vs Precision
% of dynamic accesses that need to be instrumented
Run-time overhead (%)
Static data race detector, e.g., Chord
Caper
Dynamic alias analysis
100
90
80
70
60
50
40
30
20
10
0
0
200
400
600
800
1000
1200
1400

Usefulness of Caper

- Improve performance of analyses whose correctness relies on knowing all data races
  - Record and replay systems
  - Atomicity checking
  - Software transactional memory

- Generate potential data races for analyses like RaceChaser/RaceMob/DataCollider

Lightweight Data Race Detection for Production Runs

Swarnendu Biswas, UT Austin
Man Cao, Ohio State University
Minjia Zhang, Microsoft Research
Michael D. Bond, Ohio State University
Benjamin P. Wood, Wellesley College

CC 2017