15-712:
Advanced Operating Systems & Distributed Systems

Detecting Concurrency Bugs

Prof. Phillip Gibbons

Spring 2023, Lecture 6

Today's Papers

"Efficient and Scalable Thread-Safety Violation Detection"

Guangpu Li, Shan Lu, Madanlal Musuvathi, Suman Nath, Rohan Padhye 2019

Optional Further Reading:

"Eraser: A Dynamic Data Race Detector for Multi-Threaded Programs"

Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, Thomas Anderson 1997

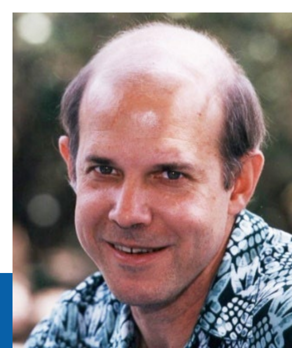
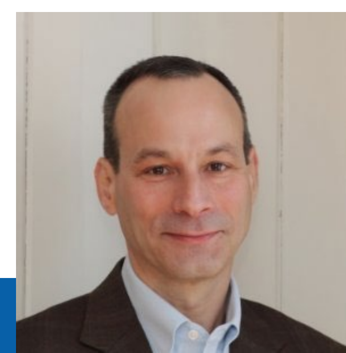
“Eraser: A Dynamic Data Race Detector for Multithreaded Programs”

Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, Thomas Anderson

- Stefan Savage (UCSD, CMU undergrad, ACM Fellow)
- Michael Burrows (Google, BWT in bzip2, FRS Fellow)
- Greg Nelson (HP, d. 2015, Herbrand Award 2013)
- Patrick Sobalvarro (Veo Robotics, many start-ups)
- Tom Anderson (U. Washington, 57000+ citations, NAE, Usenix Lifetime Achievement Award 2014)

SOSP’97 Best Paper




Data Race Detection

Data Race: Two concurrent threads access a shared variable and

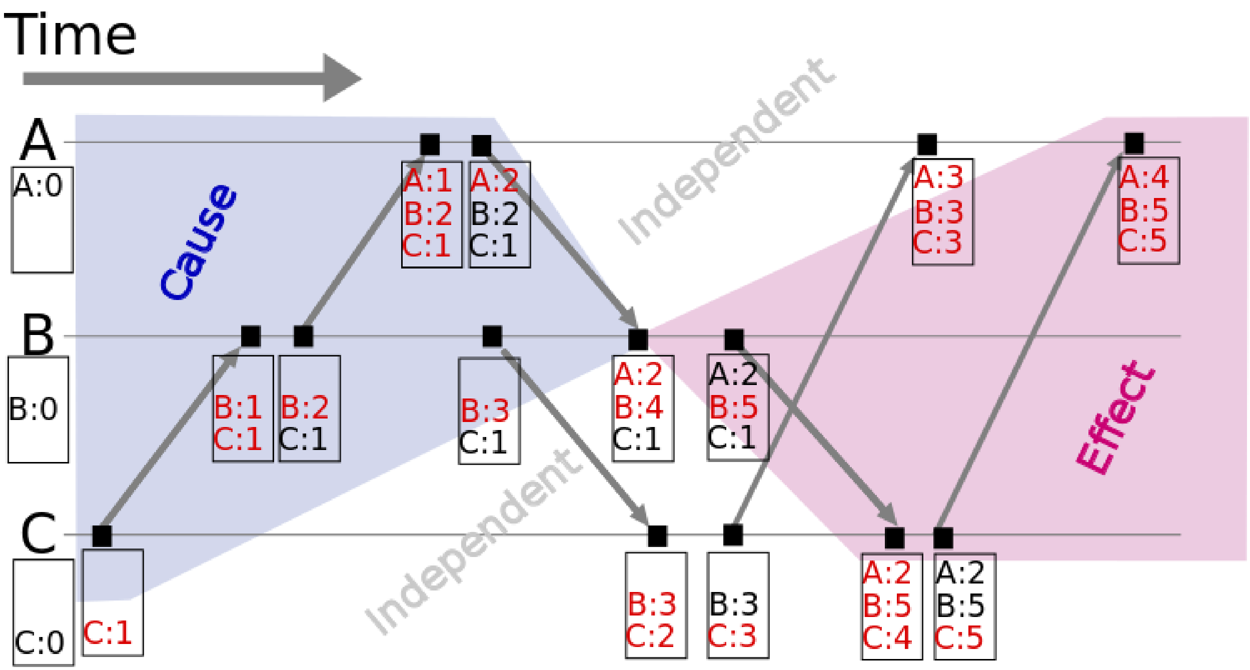
- At least one access is a write
- The threads use no explicit mechanism to prevent the accesses from being simultaneous

- **Monitors [Hoare 1974] prevent data races at compile time, but only when all shared variables are static globals**
- **Static Analysis must reason about program semantics**
- **Happens-before Analysis**
  - E.g., using vector clocks

This paper: Based on locking discipline

Vector Clocks for Race Detectors

- $a \to b$ iff $V(a) < V(b)$
- Using vector clocks
  - Inter-thread arcs are from unlock L to next lock L; otherwise, report a data race
  - Check each access for conflicting access unrelated by $\to$



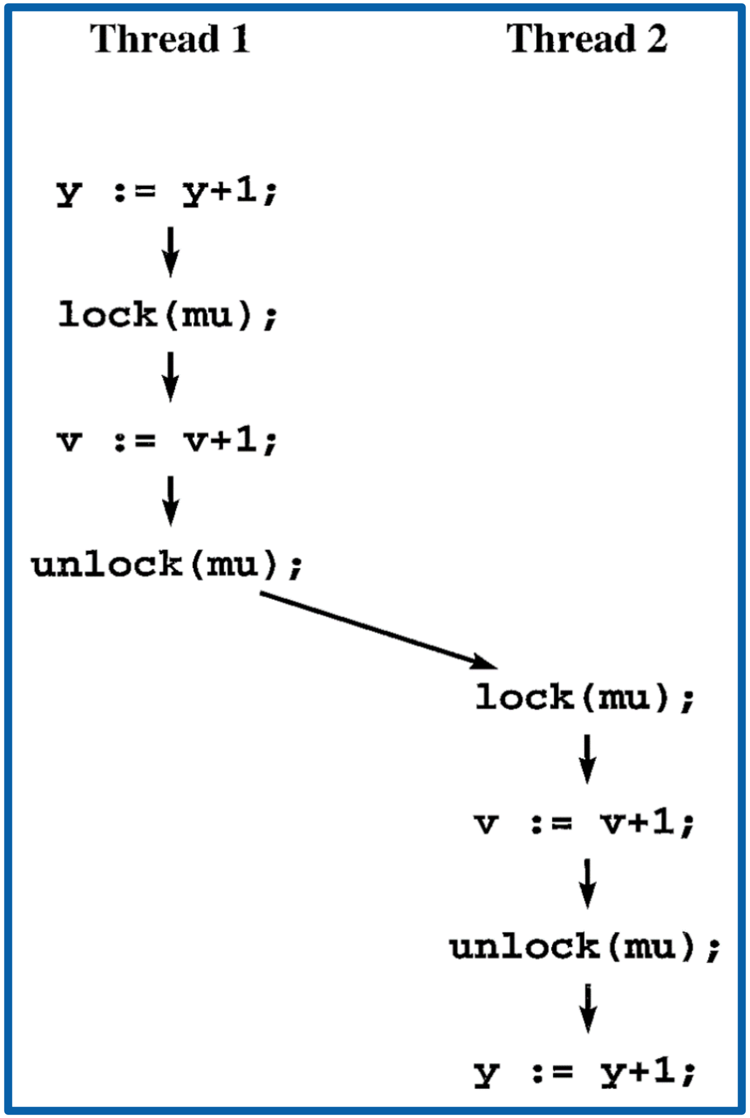
Drawbacks of Happens-Before

Difficult to implement efficiently

- Require per-thread info about concurrent accesses to each shared-memory location

Effectiveness highly dependent on interleaving that occurred

- Can miss a data race



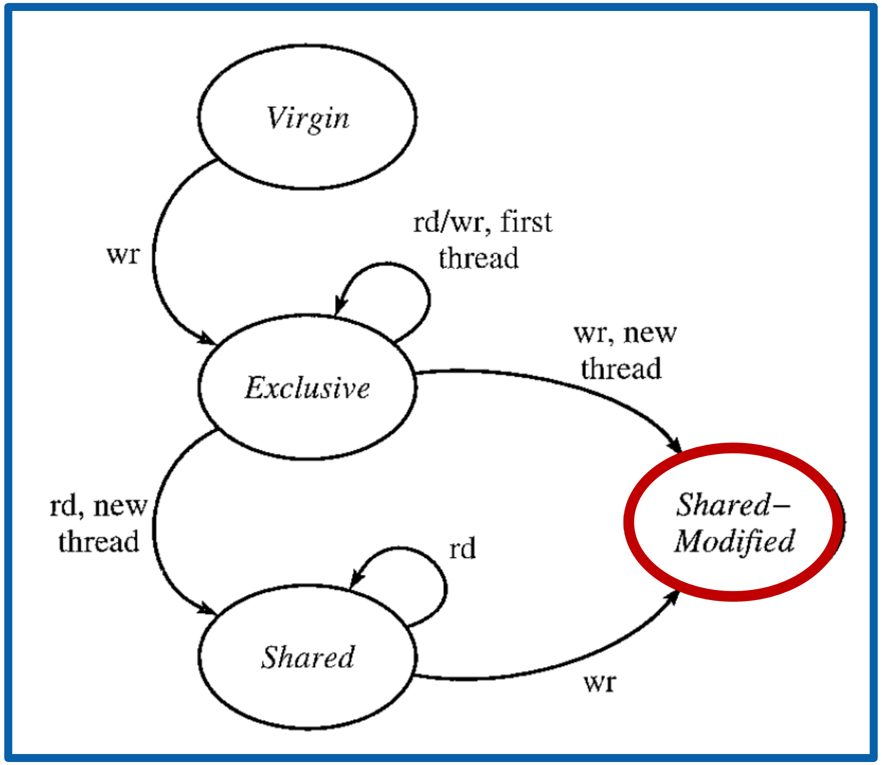
Eraser's Lockset Algorithm (1st version)

- Let **locks_held(t)** be the set of locks held by thread t
- For each v, initialize **C(v)** to the set of all locks
- On each access to v by thread t:
  - Set **C(v)** := **C(v)** ∩ **locks_held(t)**
  - If **C(v)** is empty, then issue a warning

Program	*locks_held*	*C(v)*
	`{}`	`{mu1,mu2}`
`lock(mu1);`		
	`{mu1}`	
`v := v+1;`		
		`{mu1}`
`unlock(mu1);`		
	`{}`	
`lock(mu2);`		
	`{mu2}`	
`v := v+1;`		
		`{}`
`unlock(mu2);`		
	`{}`	

Handling Initialization & Read Sharing

- State machine tracked for each variable v



- Empty lockset C(v) reported only if v is Shared-Modified

False Alarms & Annotations

- **Memory reused without resetting shadow memory**
  - When app uses private memory allocator
  - Annotation: EraserReuse(address, size) – reset to Virgin

- **Synchronization outside of instrumented channels**
  - E.g., Private lock implementations of MultiRd/SingleWr locks
  - E.g., Spin on flag
  - Annotation: EraserReadLock(lock), EraserReadUnlock(lock), EraserWriteLock(lock), EraserWriteUnlock(lock)

- **Benign races**
  - E.g., setting a one-time (e.g., finalization) flag, stats counters
  - Annotation: EraserIgnoreOn(), EraserIgnoreOff()

"We found that a handful of these annotations usually suffices to eliminate all false alarms."

Aside: Performance Comparison from FastTrack paper [Most Influential Paper from PLDI’09]

Program	Size (loc)	Thread Count	Base Time (sec)	Instrumented Time (slowdown)								Warnings					
				Empty	Eraser	MultiRace	Goldilocks RR	Goldilocks Kaffe	BasicVC	DJIT+	FastTrack	Eraser	MultiRace	Goldilocks	BasicVC	DJIT+	FastTrack
colt	111,421	11	16.1	0.9	0.9	0.9	1.8	2	0.9	0.9	0.9	3	0	0	0	0	0
crypt	1,241	7	0.2	7.6	14.7	54.8	77.4	–	84.4	54.0	14.3	0	0	0	0	0	0
lufact	1,627	4	4.5	2.6	8.1	42.5	–	8.5	95.1	36.3	13.5	4	0	–	0	0	0
moldyn	1,402	4	8.5	5.6	9.1	45.0	17.5	28.5	111.7	39.6	10.6	0	0	0	0	0	0
montecarlo	3,669	4	5.0	4.2	8.5	32.8	6.3	2.2	49.4	30.5	6.4	0	0	0	0	0	0
mtrt	11,317	5	0.5	5.7	6.5	7.1	6.7	–	8.3	7.1	6.0	1	1	1	1	1	1
raja	12,028	2	0.7	2.8	3.0	3.2	2.7	–	3.5	3.4	2.8	0	0	0	0	0	0
raytracer	1,970	4	6.8	4.6	6.7	17.9	32.8	146.8	250.2	18.1	13.1	1	1	1	1	1	1
sparse	868	4	8.5	5.4	11.3	29.8	64.1	–	57.5	27.8	14.8	0	0	0	0	0	0
series	967	4	175.1	1.0	1.0	1.0	1.0	1.1	1.0	1.0	1.0	1	0	0	0	0	0
sor	1,005	4	0.2	4.4	9.1	16.9	63.2	1.4	24.6	15.8	9.3	3	0	0	0	0	0
tsp	706	5	0.4	4.4	24.9	8.5	74.2	2.9	390.7	8.2	8.9	9	1	1	1	1	1
elevator*	1,447	5	5.0	1.1	1.1	1.1	1.1	–	1.1	1.1	1.1	0	0	0	0	0	0
philo*	86	6	7.4	1.1	1.0	1.1	7.2	1	1.1	1.1	1.1	0	0	0	0	0	0
hedc*	24,937	6	5.9	1.1	0.9	1.1	1.1	3.3	1.1	1.1	1.1	2	1	0	3	3	3
jbb*	30,491	5	72.9	1.3	1.5	1.6	2.1	–	1.6	1.6	1.4	3	1	–	2	2	2
Average				4.1	8.6	21.7	31.6	24.2	89.8	20.2	8.5	27	5	3	8	8	8

Table 1: Benchmark Results. Programs marked with ‘*’ are not compute-bound and are excluded when computing average slowdowns.

Aside: Data Race Detection in 2000s

- **Valgrind tools: Helgrind, DRD, ThreadSanitizer**
  - Use Happens-before
  - Only ThreadSanitizer also uses Lockset
  - Early versions of Helgrind used Lockset

- **Intel ThreadChecker**
  - Uses Happens-before

- **Cilk: Nondeterminator, Cilkscreen**
  - Relies on fork-join structure of Cilk programs to determine whether two conflicting accesses are ordered
  - Reports race or that <u>no race can occur with the given input</u>

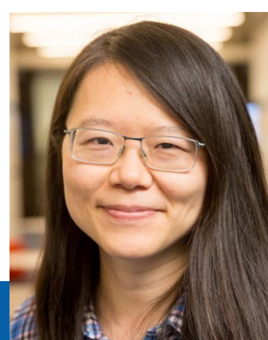
Hundreds of papers & prototype systems

"Efficient Scalable Thread-Safety-Violation Detection"

Guangpu Li, Shan Lu, Madanlal Musuvathi, Suman Nath, Rohan Padhye 2019

- **Guangpu Li (U. Chicago student, MSR intern, Citadel Securities)**
- **Shan Lu (U. Chicago, Li's advisor, SigOps Chair)**
- **Madan Musuvathi (MSR)**
- **Suman Nath (MSR, CMU PhD 2005)**
  - Who is Suman's most frequent co-author?
- **Rohan Padhye (UC Berkeley student, MSR intern, CMU asst. prof)**

SOSP'19 Best Paper


Efficient and Scalable Thread-Safety-Violation Detection

Finding thousands of concurrency bugs during testing

Guangpu Li, Shan Lu, Madanlal Musuvathi, Suman Nath, Rohan Padhye

Selected slides from Li's presentation at SOSP'19, with a few modifications

Thread-safety violation (TSV)

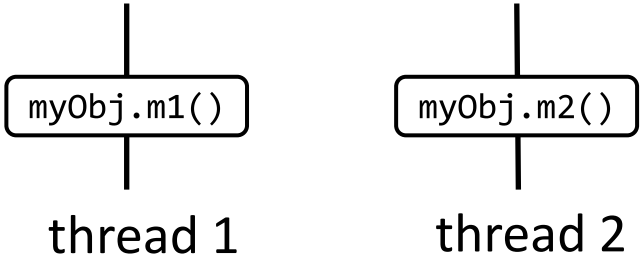
```
class MyClass{
  public:
    m1();
    m2();
  ...
}
```

Thread-safety contract

$m1 \not\parallel m1$

$m1 \not\parallel m2$

Thread-safety violation



TSVs generalize the notion of data races to coarser-grain objects

THE UNIVERSITY OF CHICAGO

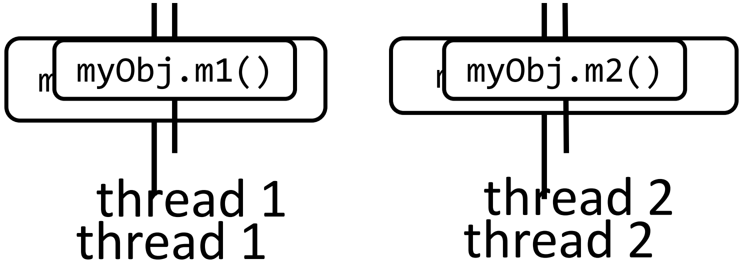
Thread-safety violation example

```
Class MyClass{
}
```

Thread-safety contract

MOTHERBOARD
TECH BY VICE

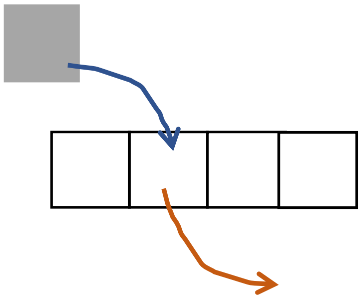
A Major Bug In Bitcoin Software Could Have Crashed the Currency



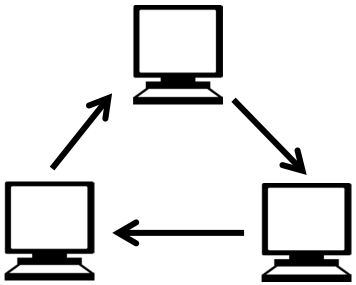
Reasoning about TSVs is difficult (I)

too many possibilities of concurrency

Multithreading



Event-driven



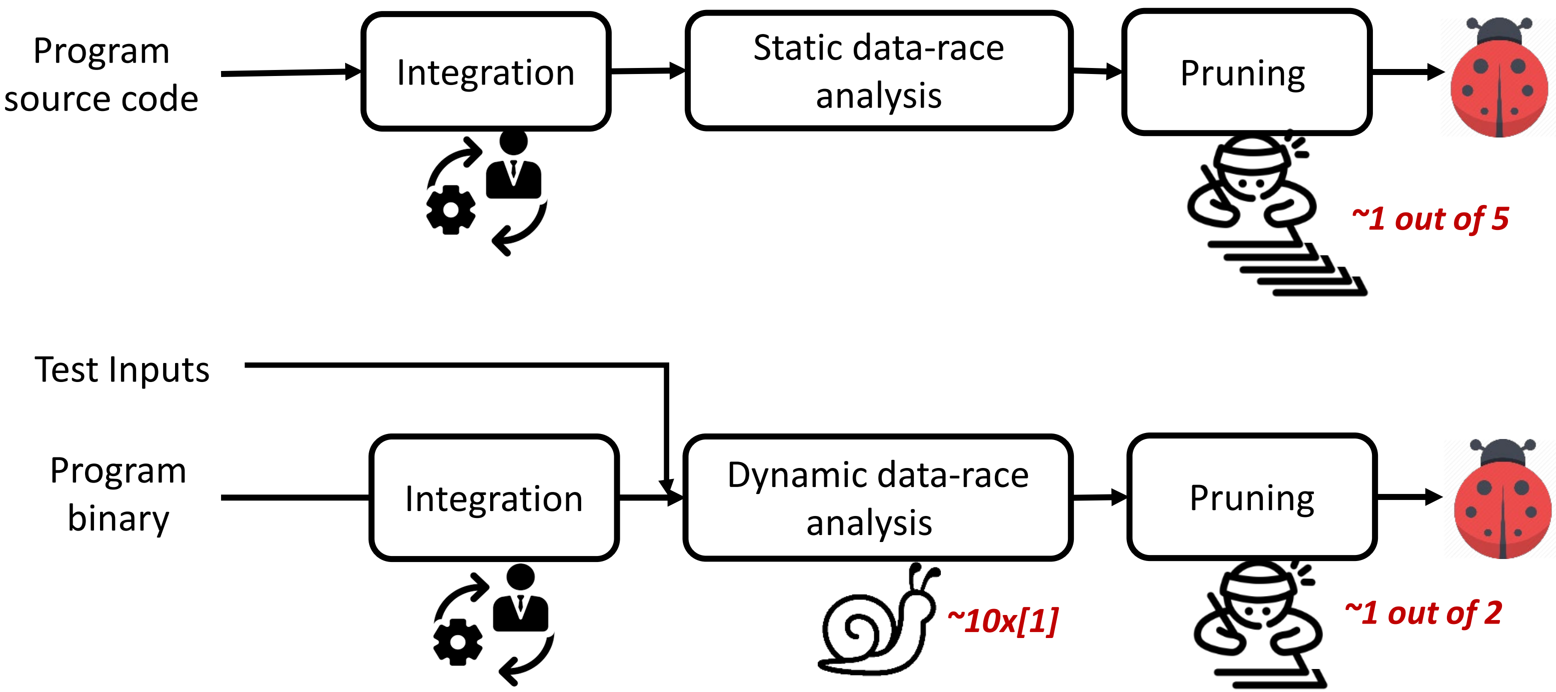
Message passing

Reasoning about TSVs is difficult (II)

too many forms of synchronizations

```
lock.Lock()
semaphore.Require()
RPC_wait()
While(flag != 1){}
Obj.notify()
pthread_mutex_lock()
Obj.wait()
Obj.signal()
Thead.join()
Event.WaitOne()
monitor.tryenter()
synchronized{…}
monitor.exit()
RPC_wait()
```

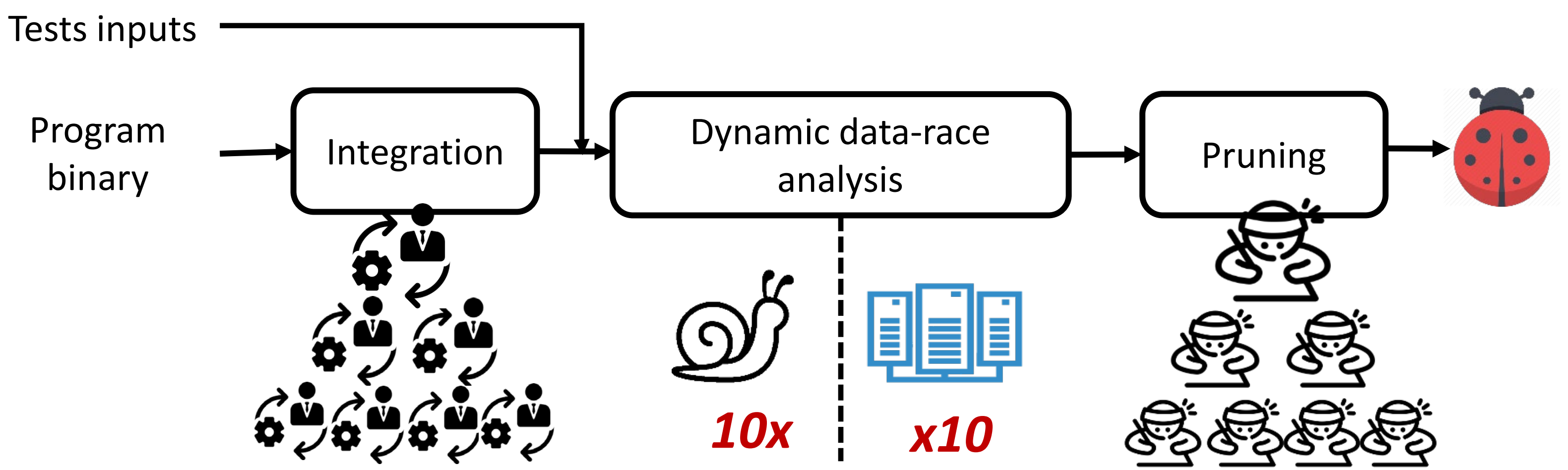
In small scale, it is fine.



[1]"FastTrack: efficient and precise dynamic race detection." *PLDI* 2009.

In large scale, it is NOT fine.

CloudBuild: 100M tests from 4K teams, up to 10K machines /day



Three challenges

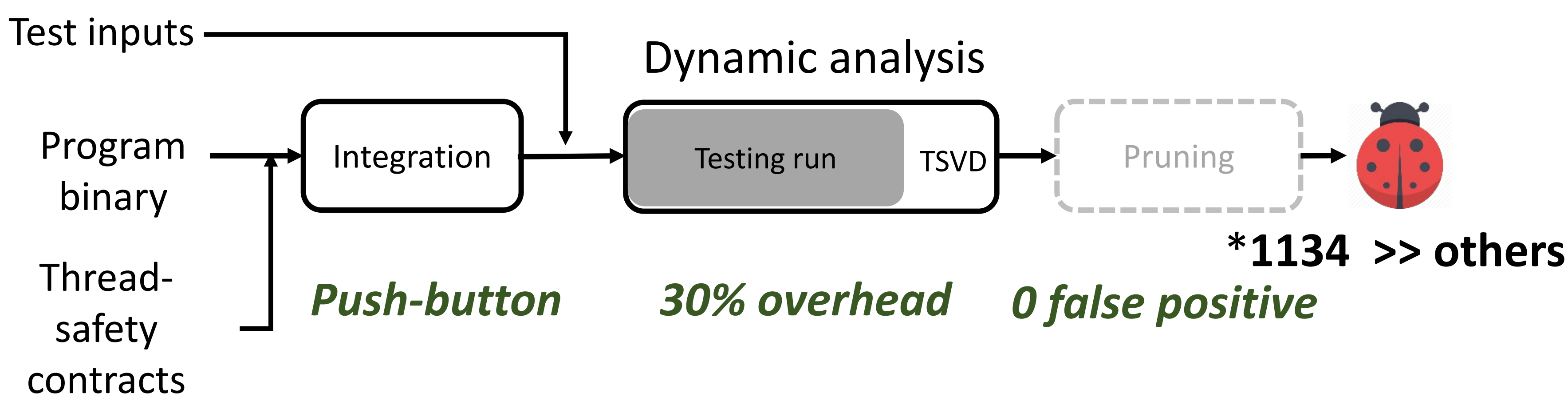
Integration

Overhead

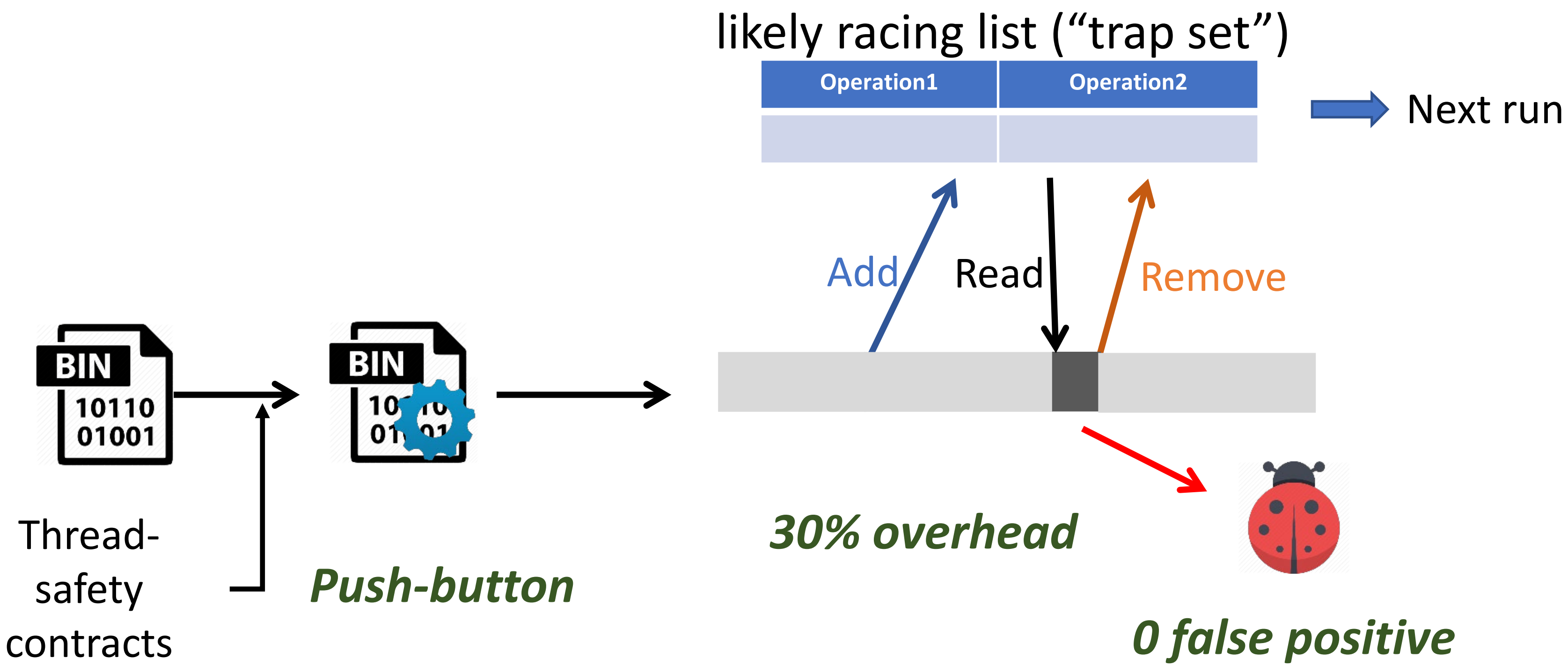
False positives

TSVD

A scalable dynamic analysis tool for TSVs:



TSVD Overview



Each entry has a prob of inserting a delay. Decays if TSV not found

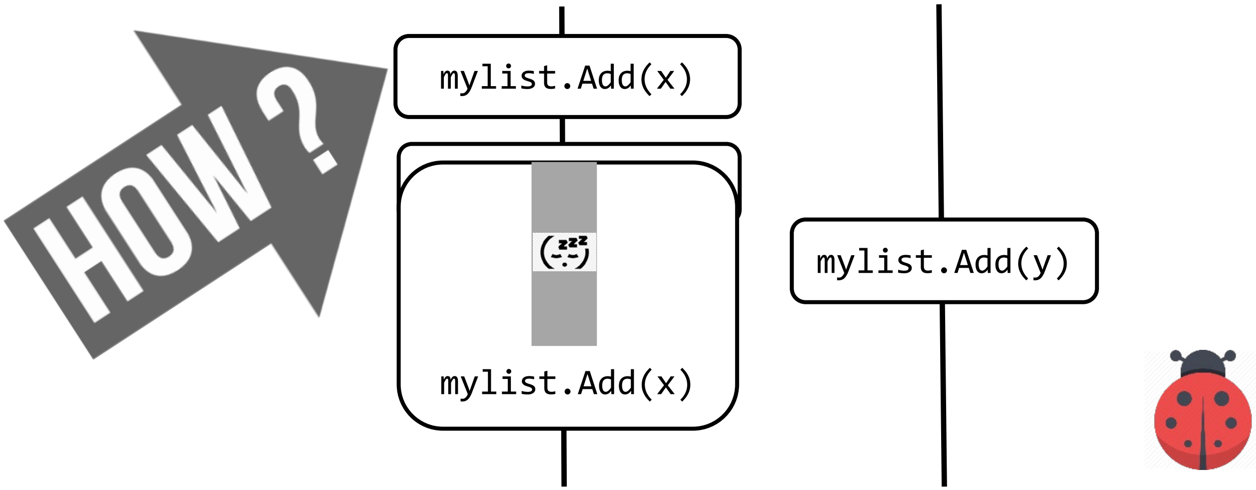
Discussion: Summary Question #1

- **State the 3 most important things the paper says.** These could be some combination of their motivations, observations, interesting parts of the design, or clever parts of their implementation.

How to achieve zero false positive?

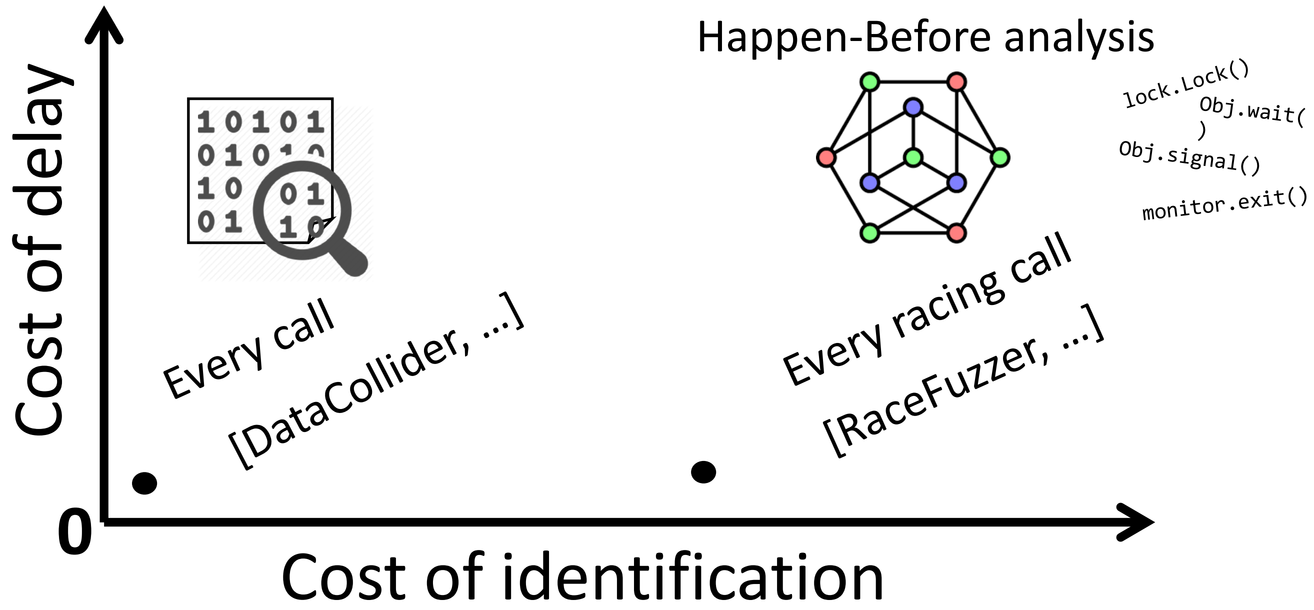
Report after violation

+ inject delays to trigger violations



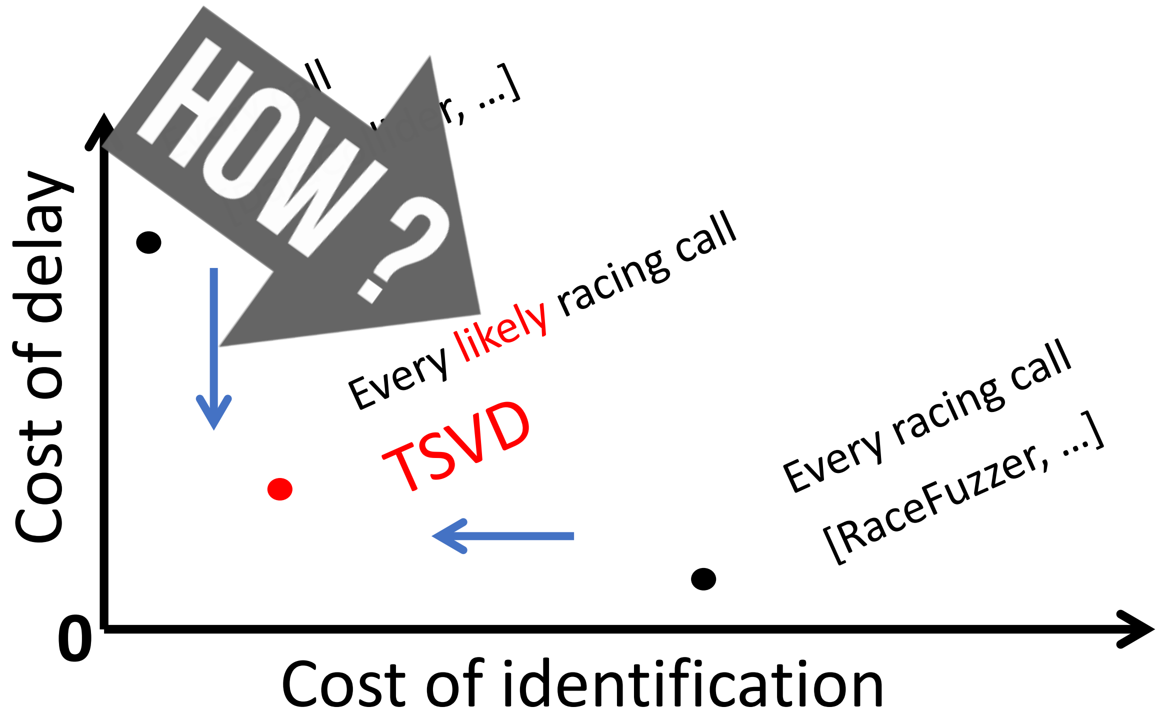
thread 1 thread 2

What are the potential unsafe calls?



THE UNIVERSITY OF CHICAGO

What are the potential unsafe calls?



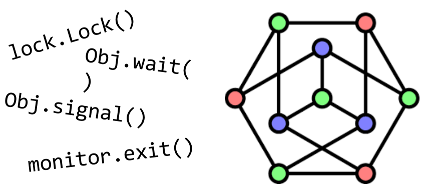
likely Racing Calls

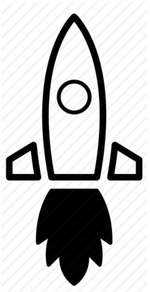
- Two conflict methods
- Called from different threads
- Accessing the same object
- Having ~~concurrent logical~~ timestamps

close-by physical

`mylist.Add(y)` — thread 1

`mylist.Add(z)` — thread 2

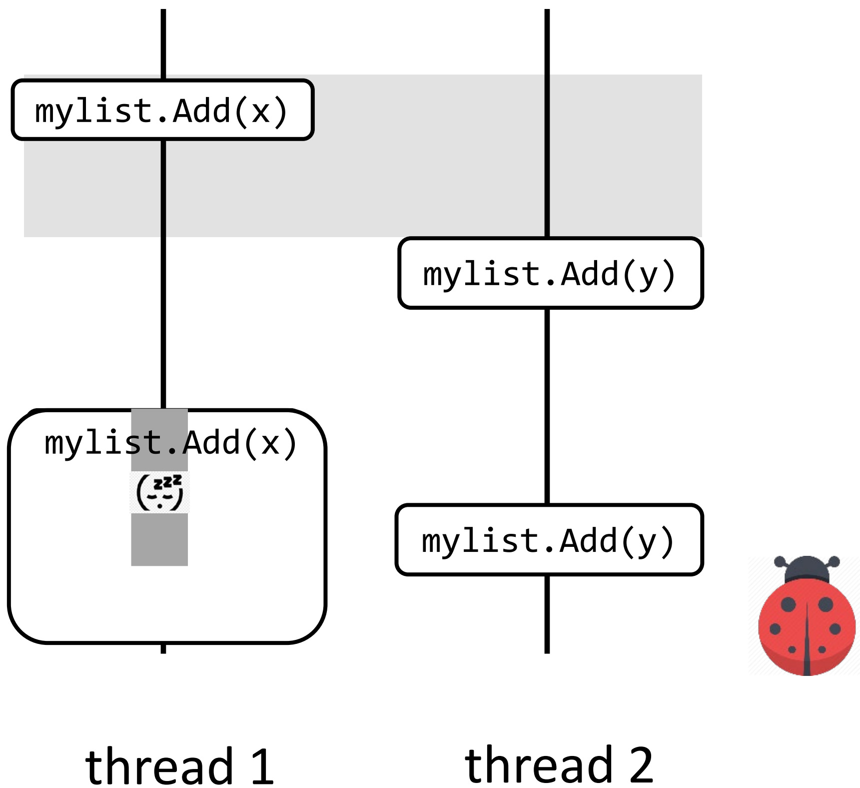




Identify likely race calls

likely racing list

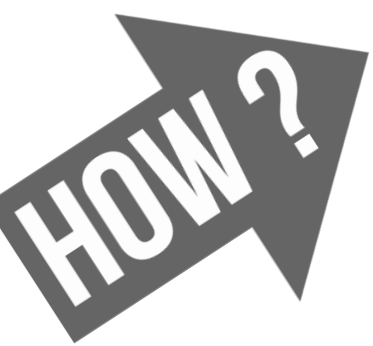
Operation1	Operation2
mylist.add(x)	mylist.add(y)

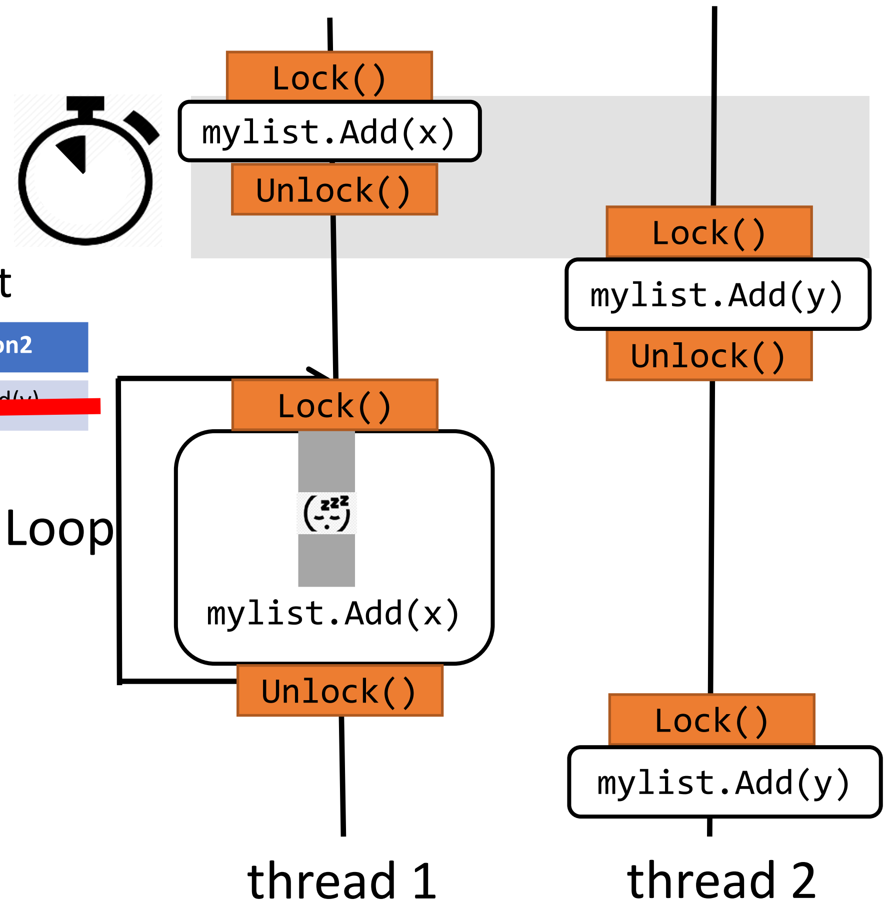


Remove unlikely race calls

likely racing list

Operation1	Operation2
~~mylist.add(x)~~	~~mylist.add(y)~~





Insights for synchronization ~~analysis~~ inference

Many ways to implement synchronization:

```
lock.Lock()
Obj.wait()
RPC_wait()
Thead.join()
While(flag != 1){}
```

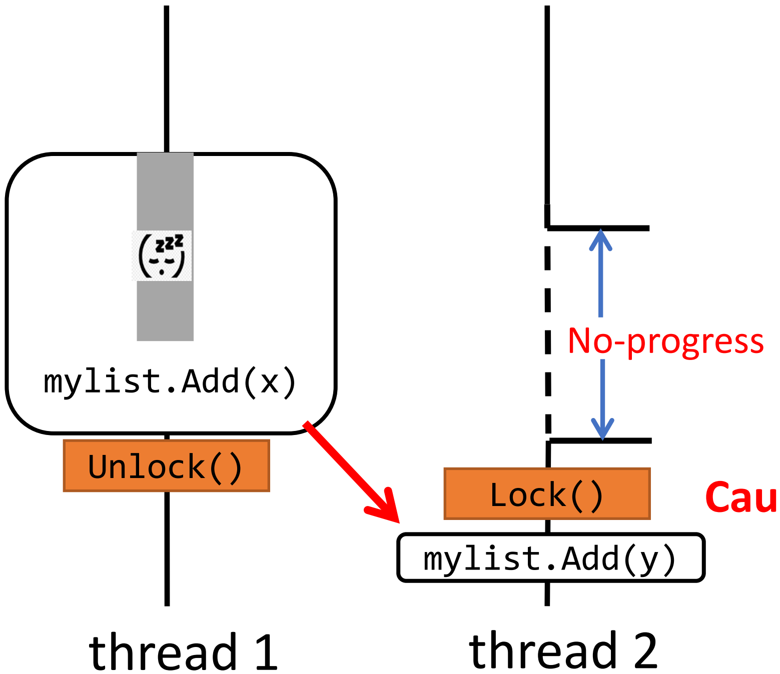
One common effect of all synchronizations:

If m1 synchronized before m2 and m1—m2 are nearby

→ *delay to m1 will cause delay to m2*

THE UNIVERSITY OF CHICAGO

Synchronization inference: transitive delay



likely racing list

Operation1	Operation2
~~mylist.add(x)~~	~~mylist.add(y)~~

Caused by any synchronization

"Happens-before interference" technique

Discussion: Summary Question #2

- **Describe the paper's single most glaring deficiency.** Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.

Limitations

- **Finds TSVs but not other data races or timing bugs**
  - Good news: Need not monitor every shared-memory access
- **Assumes for each data structure:**
  - Methods can be grouped into a read set and a write set
  - Two concurrent methods are TSVs iff at least one in the write set
- **Its parallel delay injection can muddy the waters**
- **Two TSV stack-trace pairs may correspond to the same bug**
- **Near-miss false negatives (only close in time under rare schedules)**
  - Other false negatives: HB inference, too short injected-delay
- **Implemented only for in-memory data structures and .NET apps**
  - Can't detect TSVs to persistent data

Overall Results

Thread safety contract:

14 system classes (e.g., List, Dictionary)

Test in Microsoft:

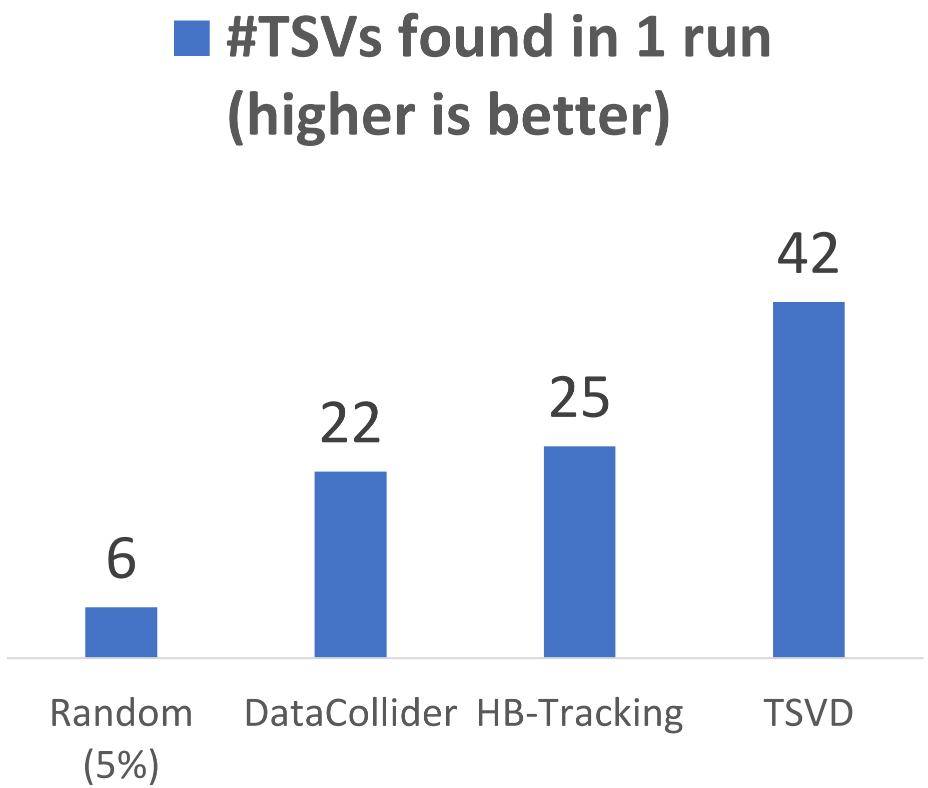
1.6K projects, run 1 or 2 times, 1134 TSVs

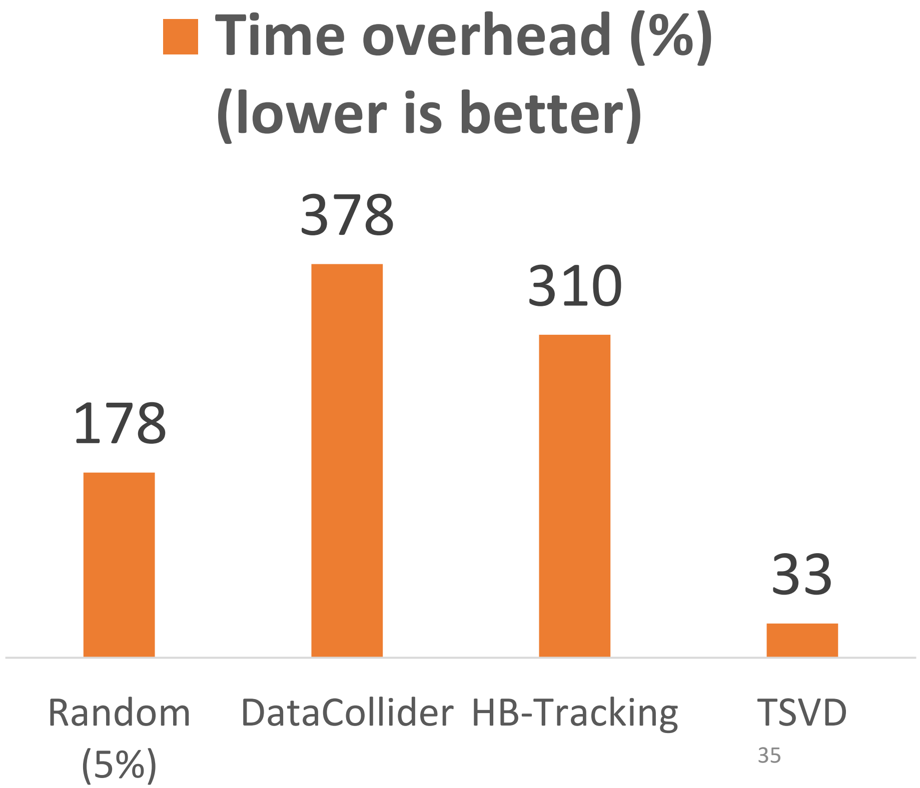
Validation of 80 bugs by 4 product teams:

96% unknown, 47% causing severe customer facing issues

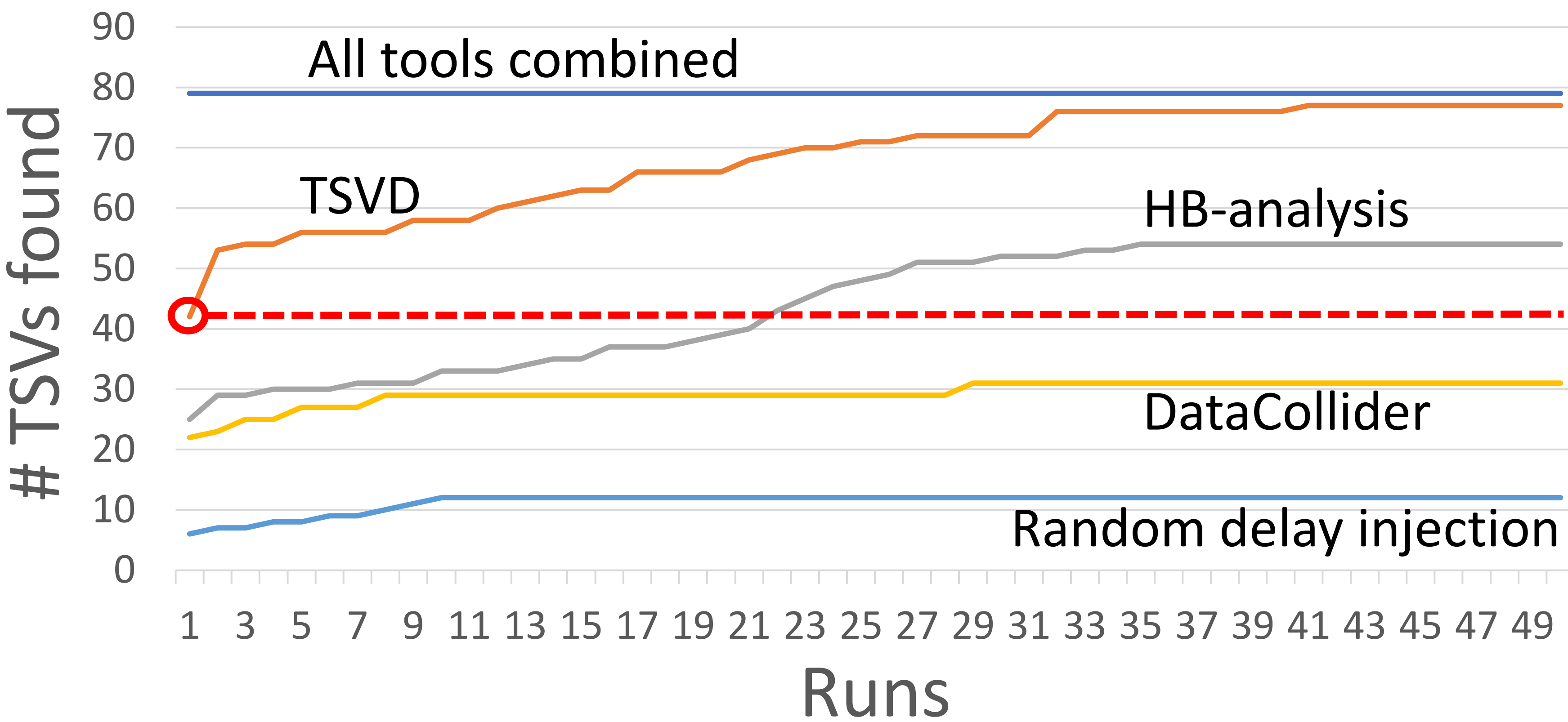
Comparison with other techniques

1K software components, 3K+ unit tests





Comparison with other techniques



Technique Sensitivity

	# bug			
	Total	Run1	Run2	overhead
TSVD	53	42	11	33%
No HB-inference	45	36	9	84%
No windowing in near-miss	46	35	11	143%
No concurrent phase detection	54	42	12	61%

Table 3. Removing one technique at a time from TSVD

Discussion: Summary Question #3

- **Describe what conclusion you draw from the paper as to how to build systems in the future.** Most of the assigned papers are significant to the systems community and have had some lasting impact on the area.

Friday's Paper

Starting a new topic:

File Systems and Disks

"A Fast File System for UNIX"

Marshall K. McKusick, William N. Joy, Samuel J. Leffler, Robert S. Fabry 1984

BACKUP SLIDES

TSVD Parameter Sensitivity Analysis

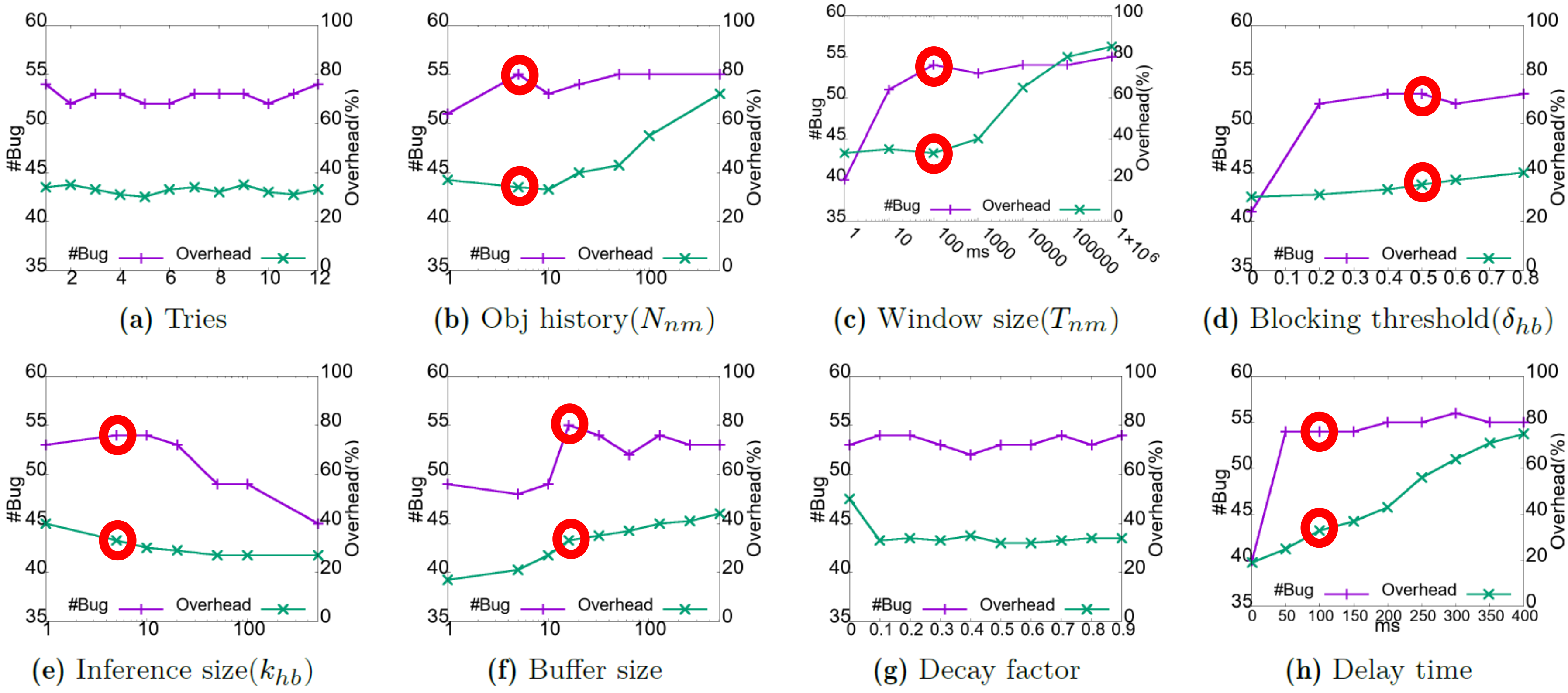


Figure 9. Sensitivity analysis of various parameters of TSVD.

Open Source Results

Project	LoC	# tests	# run	# TSV	overhead
ApplicationInsights [3]	67.5K	934	2	1	15.31%
DataTimeExtention [12]	3.2K	169	1	3	18.51%
FluentAssertion [20]	78.3K	3076	1	2	8.89%
K8s-client [33]	332.3K	76	2	1	11.79%
Radical [50]	96.9K	965	1	3	1552.13%
Sequolocity [59]	6.6K	209	1	3	2.97%
Stastd [62]	2.5K	34	2	1	9.72%
System.Linq.Dynamic [63]	1.2K	7	1	1	41.39%
Thunderstruck [64]	1.1K	52	1	2	3.33%

Table 4. TSVD results on open source projects.

Lockset Algorithm in Shared-Modified State

- **Let locks_held(t) be the set of locks held by thread t;**
  Let write_locks_held(t) be set of locks held in write mode by t

- **When enter Shared-Modified state:**
  For each v, initialize C(v) to the set of all locks

  On each read of v by thread t:
  - Set **C(v)** := **C(v)** ∩ **locks_held(t)**
  - If **C(v)** is empty, then issue a warning

  On each write of v by thread t:
  - Set **C(v)** := **C(v)** ∩ **write_locks_held(t)**
  - If **C(v)** is empty, then issue a warning

Correct: Locks held purely in read mode do not protect against a data race between the writer & some other reader thread

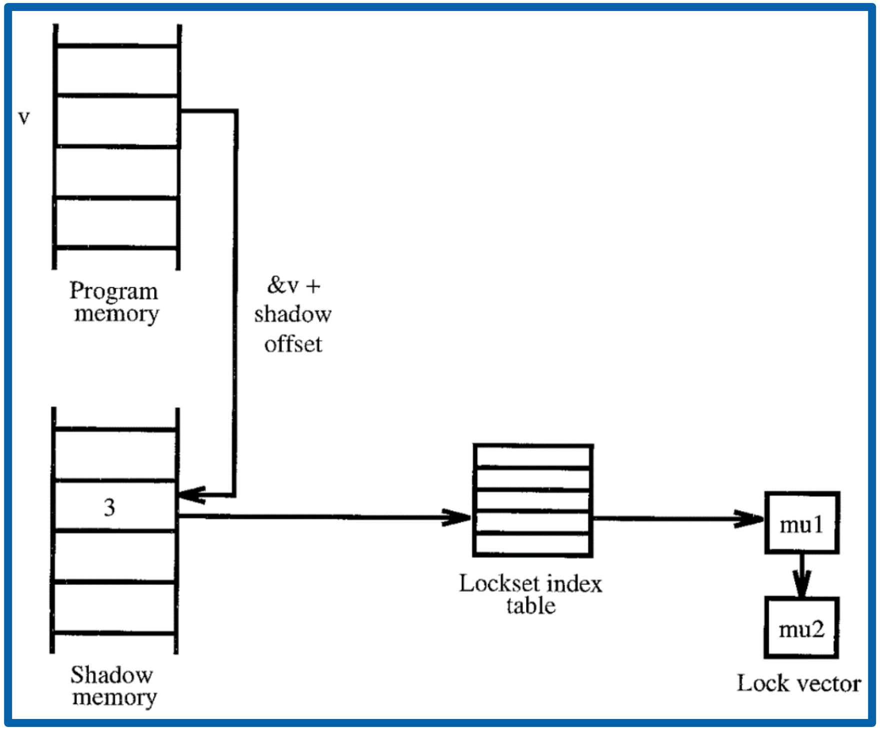
LockSet Implementation

- **Binary instrumentation**
- **Instruments lock/unlock calls, thread init/finalize to maintain lock_held(t)**
- **Instruments each load/store, malloc to maintain C(v)**
  - 32-bit (aligned) words
  - But not stack-based accesses (stack is assumed private)
  - 32-bits in "shadow memory" for each word
    (holds 2-bit state + thread ID or "lockset index")
- **Warnings report file, line number, active stack frames, thread ID, memory access address & type, PC, SP**
  - Option: Log all accesses to v that modify **C(v)**

Representing C(v)s

- Represent by small integer "lockset index" into table
  - Never observed > 10K distinct lock sets

- Append-only table
- Lock vectors sorted
- Cache results of set intersections
- Shadow word: 30-bit index, 2-bit state



Issue: Shadow memory doubles size of memory!
(Aside: Can fix with 2-level shadow memory)

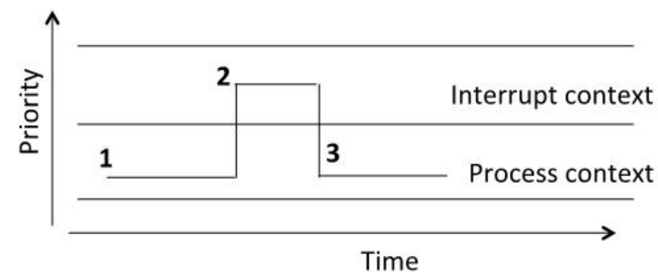
Race Detection in OS Kernel

- **OS often raises the processor interrupt level to provide mutual exclusion**
  - Particular interrupt level inclusively protects all data protected by lower interrupt levels
  - Solution: Have a virtual lock for each level; when raise level to *n*, treat this as first *n* per-level locks acquired
- **OS makes greater use of POST/WAIT style synch, e.g., semaphores to signal when a device op is done**
  - Problem: Hard to infer which data a semaphore is protecting

Race Detection in Kernels

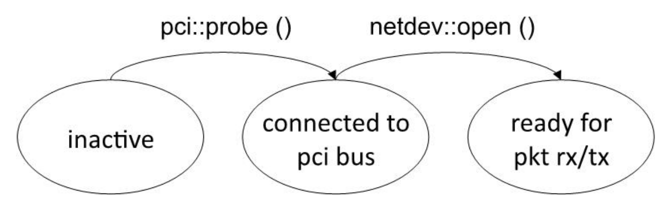
- **OS raises the processor interrupt level to provide mutual exclusion & uses POST/WAIT style synch to signal when a device op is done**

- **DataCollider [OSDI'10] – uses *active delay-injection***
  - Randomly delays a kernel thread to see if racy access occurs while stalled (but can't use for time-critical interrupts)



1. In ***process*** context (e.g. packet transmission)
2. Preempted to ***interrupt context*** to service NIC interrupt
3. Resume ***process*** context

- **Guardrail [ASPLOS'14] for kernel-mode drivers addresses the challenges:**
  - Single thread can race itself (!)
  - Synchronization invariants based on context of device state
  - Mutual exclusion via HW test-and-set or disabling interrupts & preemption



Experience

- **Ten iterations of races/false alarms to resolve all reported races**

- **Worked well on servers**
  - Experienced programmers obey the simple locking discipline

- **AltaVista Web indexing service: mhttpd & Ni2**
  - Some good examples of benign races in production codes
  - 24 annotations reduced false positives from 100+ to 0
  - Test: Reintroduced 2 old Ni2 bugs & found/corrected in 30 mins

- **Vesta Cache Server**
  - Found data race on “valid” bit—serious on weak memory model
  - Benign: Main thread passes RPC request to worker thread;
    Head-of-log lock makes entire log private
  - 10 annotations & 1 bug fix reduced alarms from 100s to 0

Experience

- **Petal distributed storage system**
  - Implements distributed consensus, failure detector/recovery
  - False alarms: private RW-lock implementation; statistics; stack frame reuse (could not annotate away)
  - Found one real race

- **Undergraduate coursework**
  - 100 runnable assignments
  - Found data races in 10% of them

- **Sensitivity to thread interleavings**
  - Reran Ni2 & Vesta on 2 threads instead of 10
  - Same race reports, in different order

Performance

“Performance was not a major goal in our implementation”

- **Typical app slowdown: 10x-30x**
  - Estimate half due to procedure call at every load/store
  - Today: dynamic binary instrumentation (DBI) using inlining for short code segments

“Eraser is fast enough to debug most programs and therefore meets the most essential performance criterion.”

Protection by Multiple Locks

- **Every writer must hold all locks**
  Every reader must hold at least 1 lock
  - Used to avoid deadlock in program that contains upcalls

- **Causes false alarms**
  - Not worth cost of handling this

Deadlock

“If the data race is Scylla, the deadlock is Charybdis.”

(Sea monsters in Homer’s Odyssey)

- **Discipline: Acquire locks in ascending order**
- **Found cycle of locks in formsedit application**
- **Would be useful addition to Eraser…**
  - Aside: In today’s tools such as Valgrind