Fuzz Testing

Ao Li

Ph.D. Candidate

Software and Societal Systems (S3D)
CMU School of Computer Science

Email: aoli@cmu.edu

Ao Li

https://aoli.al

Ph.D. candidate @ S3D/SCS

Research: Make complex concurrent and distributed systems easier to debug and test. I focuses on both algorithmic efficiency and real-world practicality.

Lots of internships and lots of systems

Google

Microsoft

amazon

{:} antithesis

What is *fuzz testing*?

Puzzle: Find `x` such `p1(x)` returns `True`

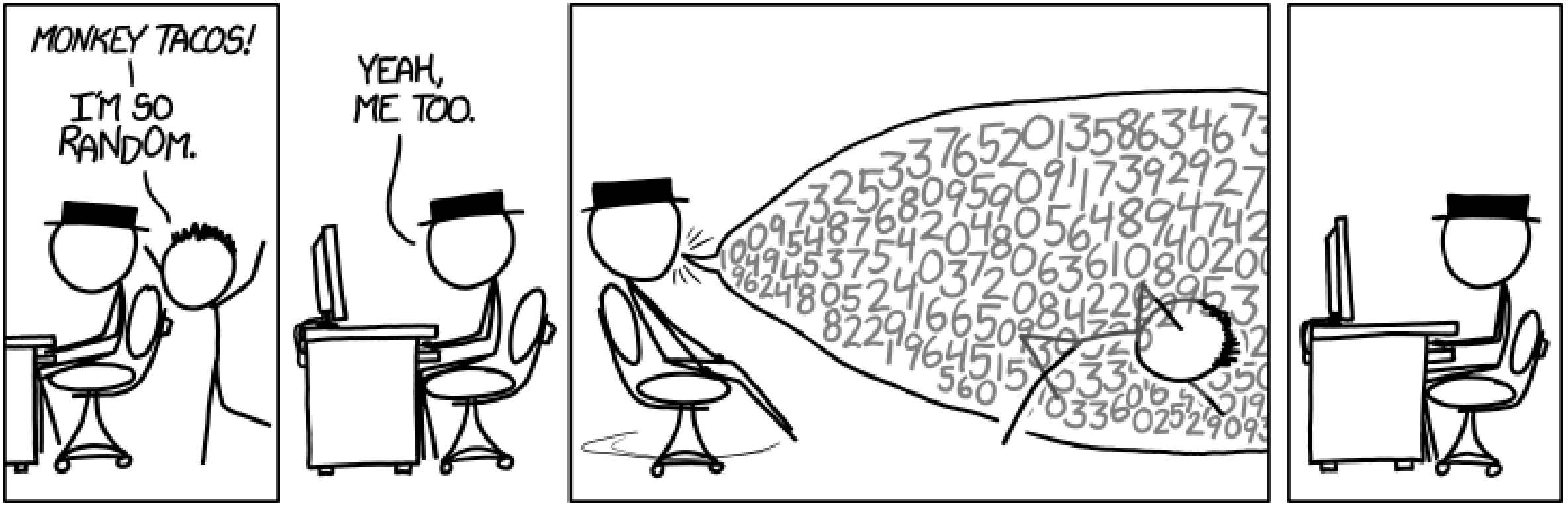
```
def p1(x):
  if x * x - 10 == 15:
    return True
  return False
```

Puzzle: Find `x` such `p2(x)` returns `True`

```
def p2(x):
  if x > 0 and x < 1000:
    if ((x - 32) * 5/9 == 100):
      return True
  return False
```

Puzzle: Find `x` such `p3(x)` returns `True`

```
def p3(x):
  if x > 3 and x < 100:
    z = x - 2
    c = 0
    while z >= 2:
      if z ** (x - 1) % x == 1:
        c = c + 1
      z = z - 1
    if c == x - 3:
      return True
  return False
```



Original: https://xkcd.com/1210 CC-BY-NC 2.5

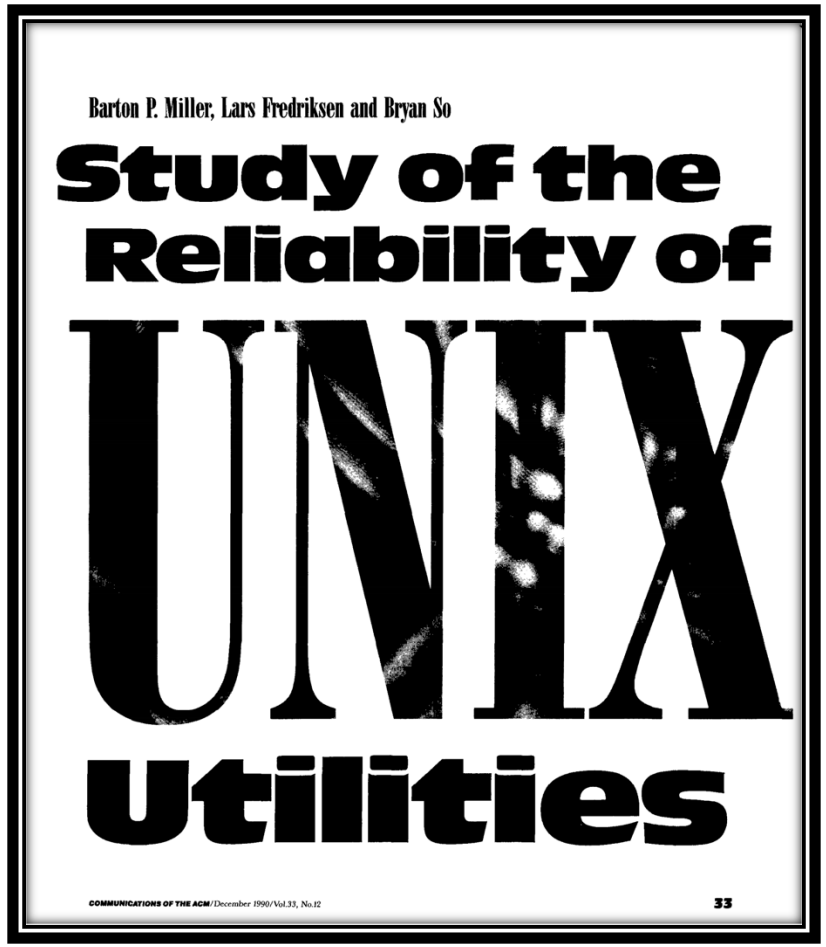
What is Fuzz Testing?

Goal:
To find test cases that reveal a bug in a program

Approach:
Generate inputs randomly until program crashes

```
$ some_program < /dev/random
```

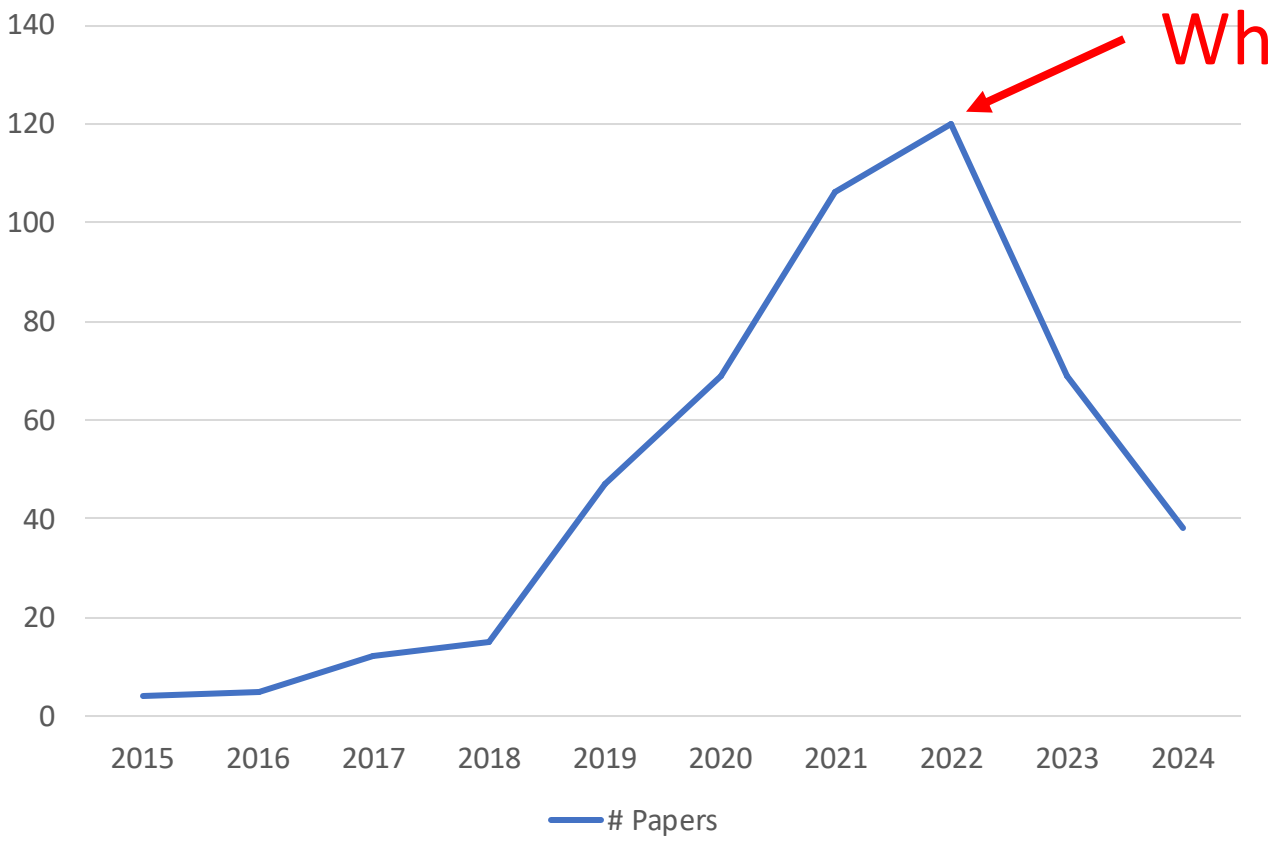
Fuzz testing was discovered quite accidentally



Communications of the ACM (1990)

> On a dark and stormy night one of the authors was logged on to his workstation on a dial-up line from home and the rain had affected the phone lines; there were frequent spurious characters on the line. The author had to race to see if he could type a sensible sequence of characters before the noise scrambled the command. This line noise was not surprising; but we were surprised that these spurious characters were causing programs to crash.

30 years on, it is still a "*hot*" research topic

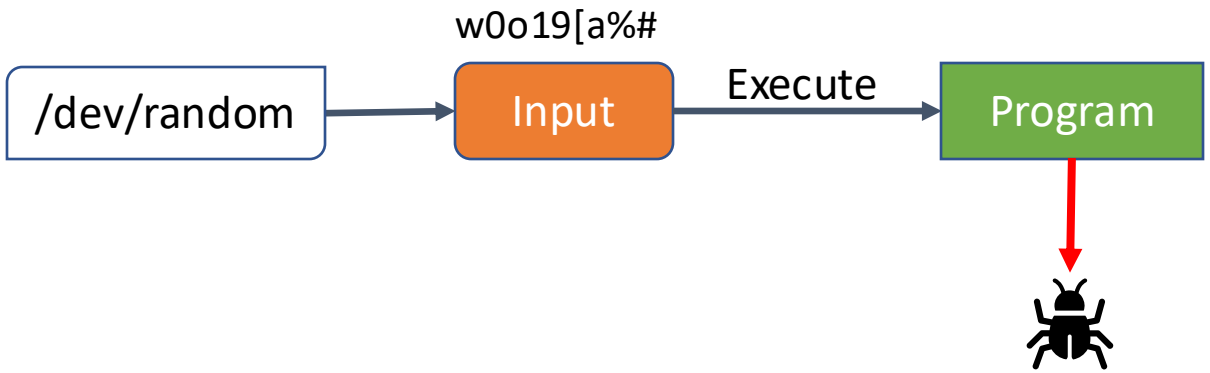


ICSE, FSE, ASE, ISSTA, OOPSLA, PLDI,
IEEE TSE, CCS, NDSS, USENIX Security,
IEEE S&P, etc.

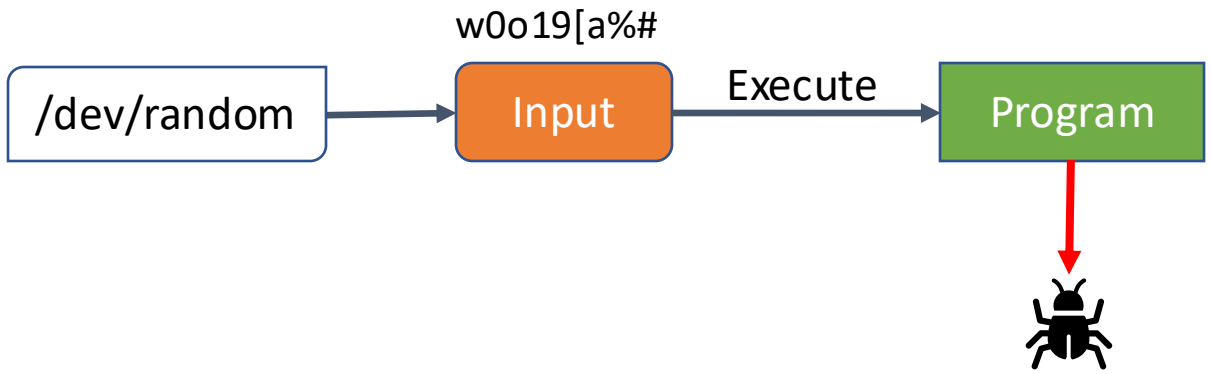
Data source: https://wcventure.github.io/FuzzingPaper

1990s

Fuzz Testing 101



1990 study found crashes in:
adb, as, bc, cb, col, diction, emacs, eqn, ftp, indent, lex, look, m4, make, nroff, plot, prolog, ptx, refer!, spell, style, tsort, uniq, vgrind, vi



What are the benefits, challenges, & limitations of this approach?

Generate inputs randomly

```
$ python foo.py
```

```
def is_even(x:int) -> bool:
    if x % 2 == 1:
        return False
    else:
        return True
```

```
$ cat /dev/random | python
```

```
lrha3wn5p0w3uz;54 p0a23
rw3i 50a20 5a2y58a2p
y3wry3p285
q@P”uer9zparu9apur9qa3802
y5o2y 392r523a90wesu
```

Purely random data is not a very interesting input!!

Generate inputs randomly via mutation

```
$ python foo.py
```

```
$ radamsa foo.py | python
```

```
def is_even(x:int) -> bool:
    if x % 2 == 1:
        return False
    else:
        return True
```

```
def is_even(x:int) -> bool:
    if x % 2 == 3412354121:
        return False
    else:
        return False
```

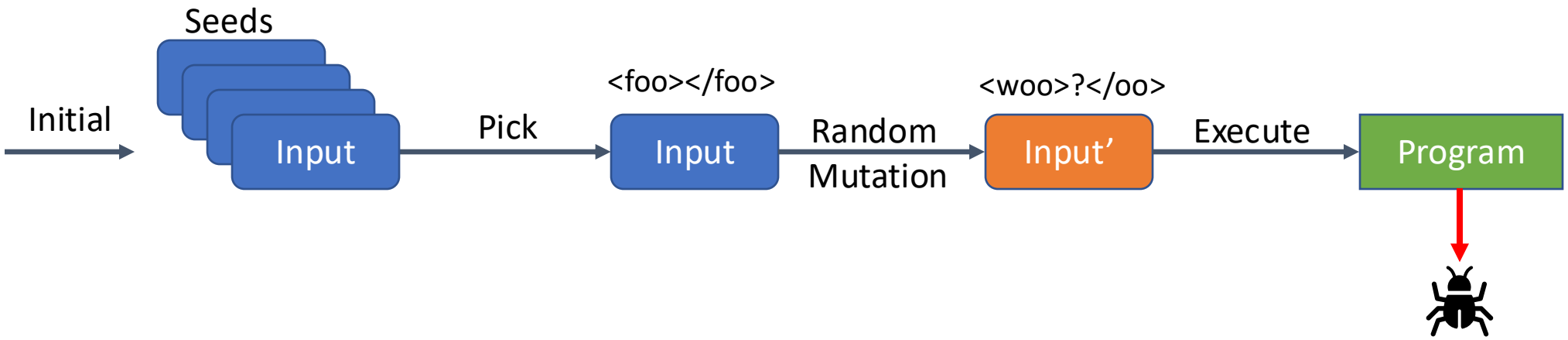
What are some good mutations?

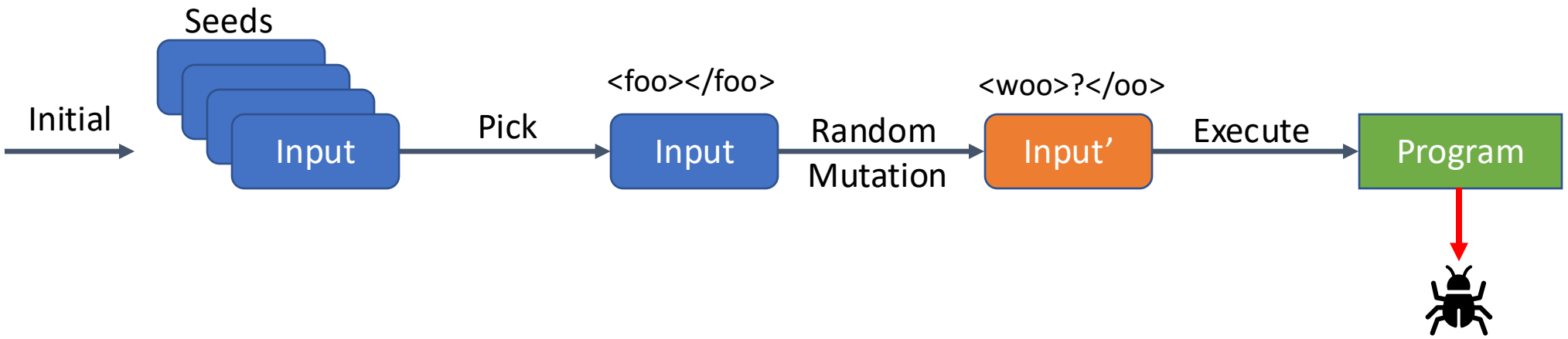
Goal: Preserve input structure as much as possible while also changing some key data values

- Binary input
  - Bit flips, byte flips
  - Change random bytes
  - Insert random byte chunks
  - Delete random byte chunks
  - Set randomly chosen byte chunks to *interesting* values e.g. INT_MAX, INT_MIN, 0, 1, -1, …
  - Other suggestions?
- Text input
  - Insert random symbols or keywords from a dictionary
  - Other suggestions?

2000s

Mutation-Based Fuzzing (e.g. Radamsa, zzuf)





What are the benefits, challenges, & limitations of this approach?

How do you know if you are making progress?

Code Coverage

LCOV - code coverage report

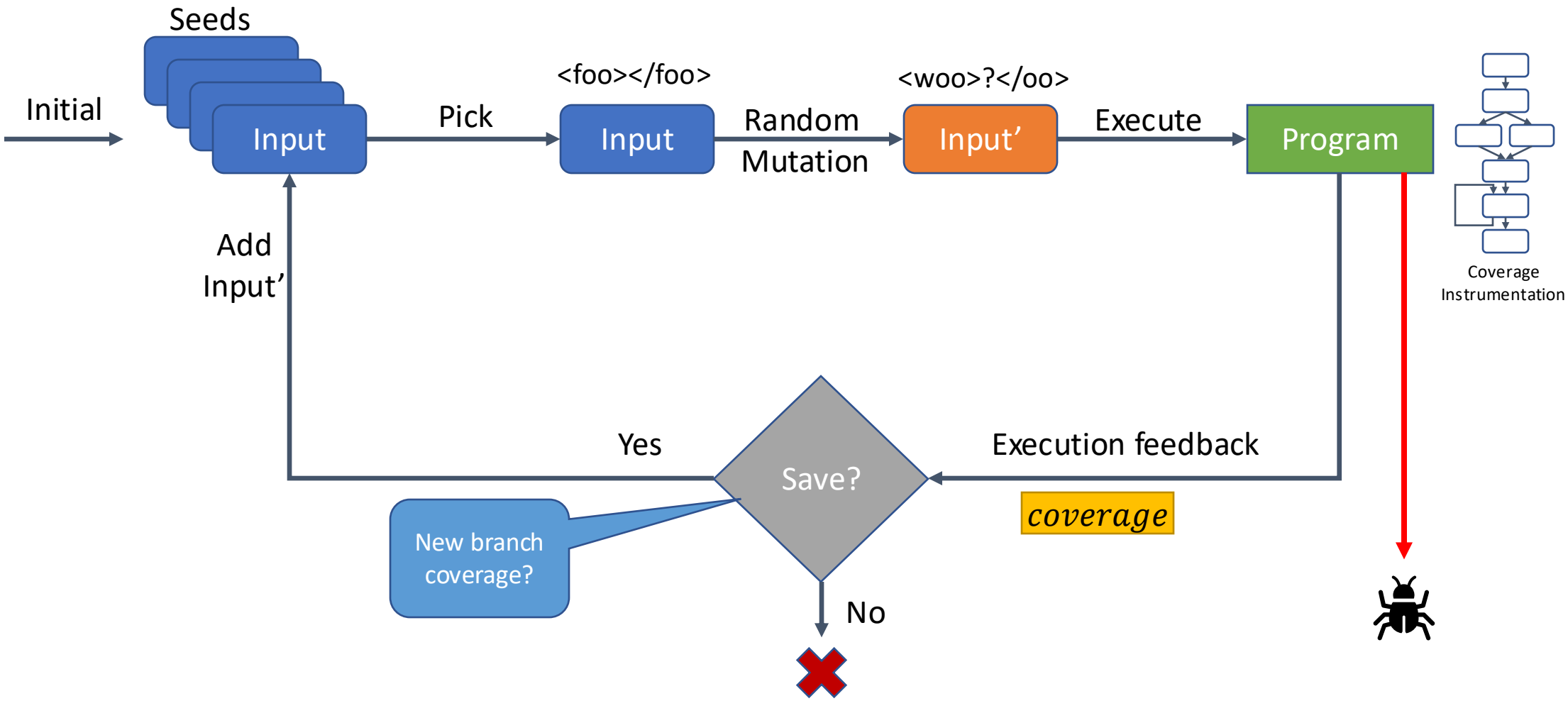
		Hit	Total	Coverage
Current view:	top level - test			
Test:	coverage.info	Lines:	6092	7293	83.5 %
Date:	2018-02-07 13:06:43	Functions:	481	518	92.9 %

Filename	Line Coverage		Functions	
asn1_string_table_test.c	58.8 %	20 / 34	100.0 %	2 / 2
asn1_time_test.c	72.0 %	72 / 100	100.0 %	7 / 7
bad_dtls_test.c	97.6 %	163 / 167	100.0 %	9 / 9
bftest.c	65.3 %	64 / 98	87.5 %	7 / 8
bio_enc_test.c	78.7 %	74 / 94	100.0 %	9 / 9
bntest.c	97.7 %	1038 / 1062	100.0 %	45 / 45
chacha_internal_test.c	83.3 %	10 / 12	100.0 %	2 / 2
ciphername_test.c	60.4 %	32 / 53	100.0 %	2 / 2
crltest.c	100.0 %	90 / 90	100.0 %	12 / 12
ct_test.c	95.5 %	212 / 222	100.0 %	20 / 20
d2i_test.c	72.9 %	35 / 48	100.0 %	2 / 2
danetest.c	75.5 %	123 / 163	100.0 %	10 / 10
dhtest.c	84.6 %	88 / 104	100.0 %	4 / 4
drbgtest.c	69.8 %	157 / 225	92.9 %	13 / 14
dtls_mtu_test.c	86.8 %	59 / 68	100.0 %	5 / 5
dtlstest.c	97.1 %	34 / 35	100.0 %	4 / 4
dtlsv1listentest.c	94.9 %	37 / 39	100.0 %	4 / 4
ecdsatest.c	94.0 %	140 / 149	100.0 %	7 / 7
enginetest.c	92.8 %	141 / 152	100.0 %	7 / 7
evp_extra_test.c	100.0 %	112 / 112	100.0 %	10 / 10
fatalerrtest.c	89.3 %	25 / 28	100.0 %	2 / 2
handshake_helper.c	84.7 %	494 / 583	97.4 %	38 / 39
hmactest.c	100.0 %	71 / 71	100.0 %	7 / 7
ideatest.c	100.0 %	30 / 30	100.0 %	4 / 4
igetest.c	87.9 %	109 / 124	100.0 %	11 / 11
lhash_test.c	78.6 %	66 / 84	100.0 %	8 / 8
mdc2_internal_test.c	81.8 %	9 / 11	100.0 %	2 / 2
mdc2test.c	100.0 %	18 / 18	100.0 %	2 / 2
ocspapitest.c	95.5 %	64 / 67	100.0 %	4 / 4
packettest.c	100.0 %	248 / 248	100.0 %	24 / 24

```
        1  /    1:                 if ((err = SSLHashMD5.final(&hashCtx, &hashOut)) != 0)
        0  /    1:                     goto fail;
                :     }
                :     else {
                :             /* DSA, ECDSA - just use the SHA1 hash */
        0  /    1:             dataToSign = &hashes[SSL_MD5_DIGEST_LEN];
        0  /    1:             dataToSignLen = SSL_SHA1_DIGEST_LEN;
                :     }
                :
        1  /    1:     hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
        1  /    1:      hashOut.length = SSL_SHA1_DIGEST_LEN;
        1  /    1:      if ((err = SSLFreeBuffer(&hashCtx)) != 0)
        0  /    1:          goto fail;
                :
        1  /    1:      if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        0  /    1:          goto fail;
        1  /    1:      if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        0  /    1:          goto fail;
        1  /    1:      if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        0  /    1:          goto fail;
        1  /    1:      if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        0  /    1:          goto fail;
        1  /    1:          goto fail;
                :      if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
                :          goto fail;
                :
                :     err = sslRawVerify(ctx,
                :                        ctx->peerPubKey,
                :                        dataToSign,                    /* plaintext *,
                :                        dataToSignLen,                 /* plaintext l
                :                        signature,
                :                        signatureLen);
                :     if(err) {
                :             sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
                :                     "returned %d\n", (int)err);
                :             goto fail;
                :     }
                :
                : fail:
        1  /    1:      SSLFreeBuffer(&signedHashes);
        1  /    1:      SSLFreeBuffer(&hashCtx);
        1  /    1:      return err;
                :
        1  /    1: }
                :
```

2010s

Coverage-Guided Fuzzing (e.g. AFL, libFuzzer)



2014+

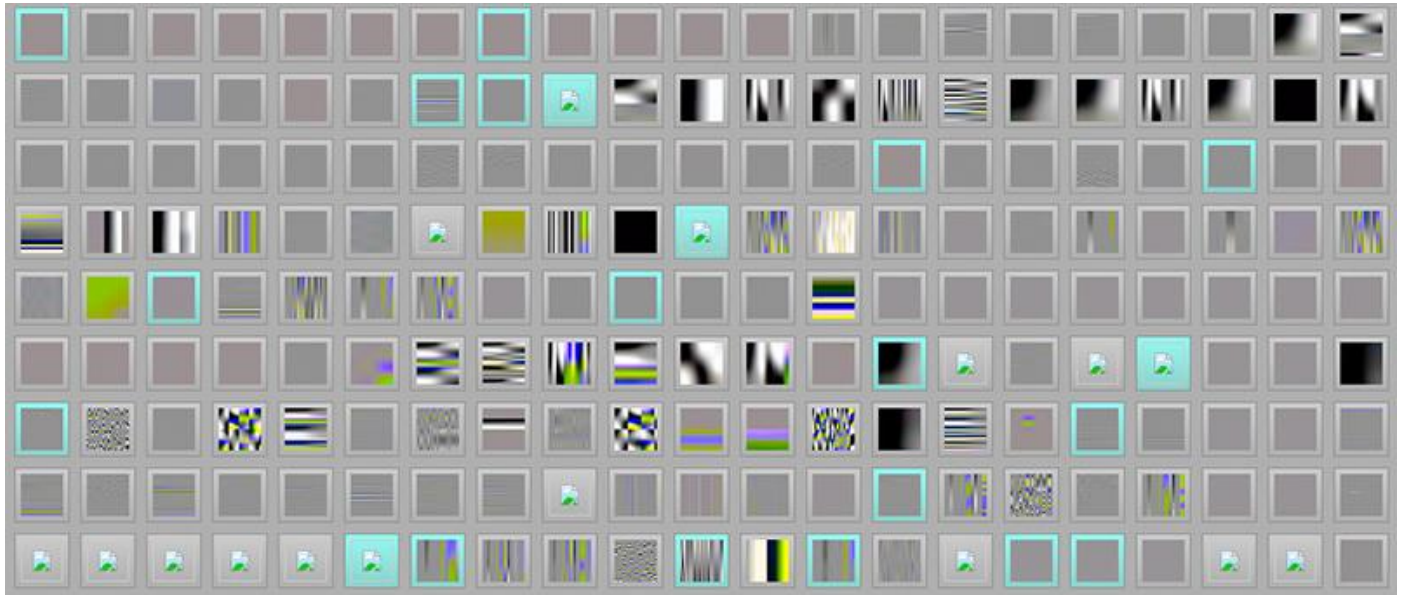
Coverage-Guided Fuzzing with AFL

November 07, 2014

Pulling JPEGs out of thin air

This is an interesting demonstration of the capabilities of afl; I was actually pretty surprised that it worked!

```
$ mkdir in_dir
$ echo 'hello' >in_dir/hello
$ ./afl-fuzz -i in_dir -o out_dir ./jpeg-9a/djpeg
```



2014+

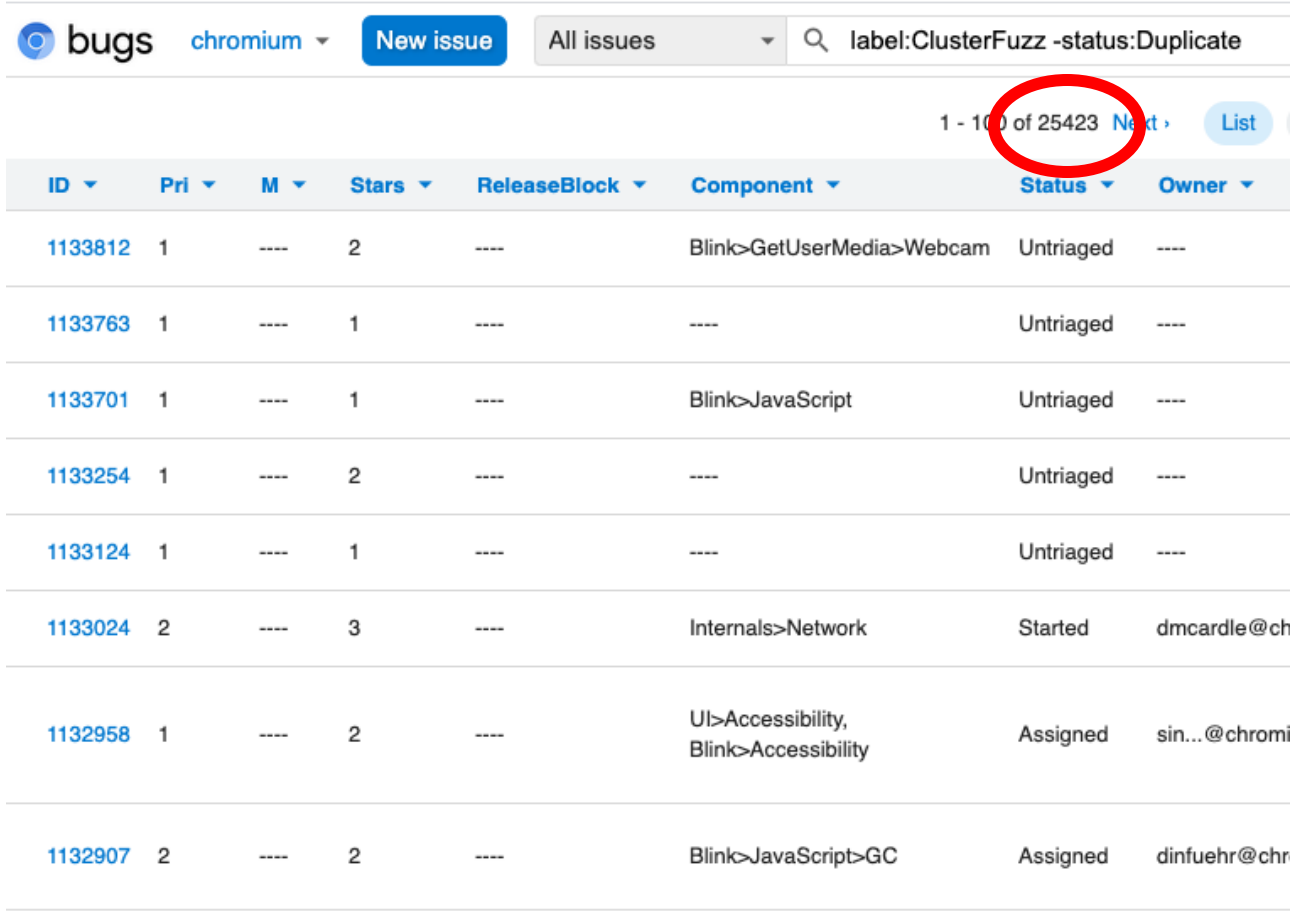
Coverage-Guided Fuzzing with AFL

The bug-o-rama trophy case

http://lcamtuf.coredump.cx/afl/

IJG jpeg [1]	libjpeg-turbo [1] [2]	libpng [1]
libtiff [1] [2] [3] [4] [5]	mozjpeg [1]	PHP [1] [2] [3] [4] [5] [6] [7] [8]
Mozilla Firefox [1] [2] [3] [4]	Internet Explorer [1] [2] [3] [4]	Apple Safari [1]
Adobe Flash / PCRE [1] [2] [3] [4] [5] [6] [7]	sqlite [1] [2] [3] [4]...	OpenSSL [1] [2] [3] [4] [5] [6] [7]
LibreOffice [1] [2] [3] [4]	poppler [1] [2]...	freetype [1] [2]
GnuTLS [1]	GnuPG [1] [2] [3] [4]	OpenSSH [1] [2] [3] [4] [5]
PuTTY [1] [2]	ntpd [1] [2]	nginx [1] [2] [3]
bash (post-Shellshock) [1] [2]	tcpdump [1] [2] [3] [4] [5] [6] [7] [8] [9]	JavaScriptCore [1] [2] [3] [4]
pdfium [1] [2]	ffmpeg [1] [2] [3] [4] [5]	libmatroska [1]
libarchive [1] [2] [3] [4] [5] [6] ...	wireshark [1] [2] [3]	ImageMagick [1] [2] [3] [4] [5] [6] [7] [8] [9] ...
BIND [1] [2] [3] ...	QEMU [1] [2]	lcms [1]

ClusterFuzz @ Chromium



ID	Pri	M	Stars	ReleaseBlock	Component	Status	Owner
1133812	1	----	2	----	Blink>GetUserMedia>Webcam	Untriaged	----
1133763	1	----	1	----	----	Untriaged	----
1133701	1	----	1	----	Blink>JavaScript	Untriaged	----
1133254	1	----	2	----	----	Untriaged	----
1133124	1	----	1	----	----	Untriaged	----
1133024	2	----	3	----	Internals>Network	Started	dmcardle@ch
1132958	1	----	2	----	UI>Accessibility, Blink>Accessibility	Assigned	sin...@chromi
1132907	2	----	2	----	Blink>JavaScript>GC	Assigned	dinfuehr@chr

Many challenges to address before deploying a fuzzing tool

- Fuzzing heuristics
  - Mutation: Which input to mutate? How many times? Which mutations?
  - Feedback: What to instrument? How to keep overhead low?
- Oracles
  - What is a bug? Crash? Silent overflow? Infinite loop? Race condition? Undefined behavior? How do we know when we have found a bug?
- Debugging
  - Reproducibility
  - Crash triaging
  - Input minimization
- Fuzzing roadblocks
  - Magic bytes, checksums (see PNG, SSL)
  - Dependencies in binary inputs (e.g. length of chunks, indexes into tables – see PNG)
  - Inputs with complex syntax and semantics (e.g. XML, JSON, C++)
  - Stateful applications

What kind of “crashes” do fuzzers find?

Causes: incorrect arg validation, incorrect type casting, executing untrusted code, etc.

Effects: buffer-overflows, memory leak, division-by-zero, use-after-free, assertion violation, etc. (“crash”)

Impact: security, reliability, performance, correctness

Oracles: Sanitizers

- Address Sanitizer (ASAN) ***
- LeakSanitizer (comes with ASAN)
- Thread Sanitizer (TSAN)
- Undefined-behavior Sanitizer (UBSAN)

https://github.com/google/sanitizers

AddressSanitizer

```
int get_element(int* a, int i) {
  return a[i];
}
```

```
int get_element(int* a, int i) {
  if (a == NULL) abort();
  return a[i];
}
```

```
int get_element(int* a, int i) {
  if (a == NULL) abort();
  region = get_allocation(a);
  if (in_stack(region)) {
    if (popped(region)) abort();
    …
  }
  if (in_heap(region)) { ... }
  return a[i];
}
```

```
int get_element(int* a, int i) {
  if (a == NULL) abort();
  region = get_allocation(a);
  if (in_heap(region)) {
    low, high = get_bounds(region);
    if ((a + i) < low || (a +i) > high) {
      abort();
    }
  }
  return a[i];
}
```

AddressSanitizer

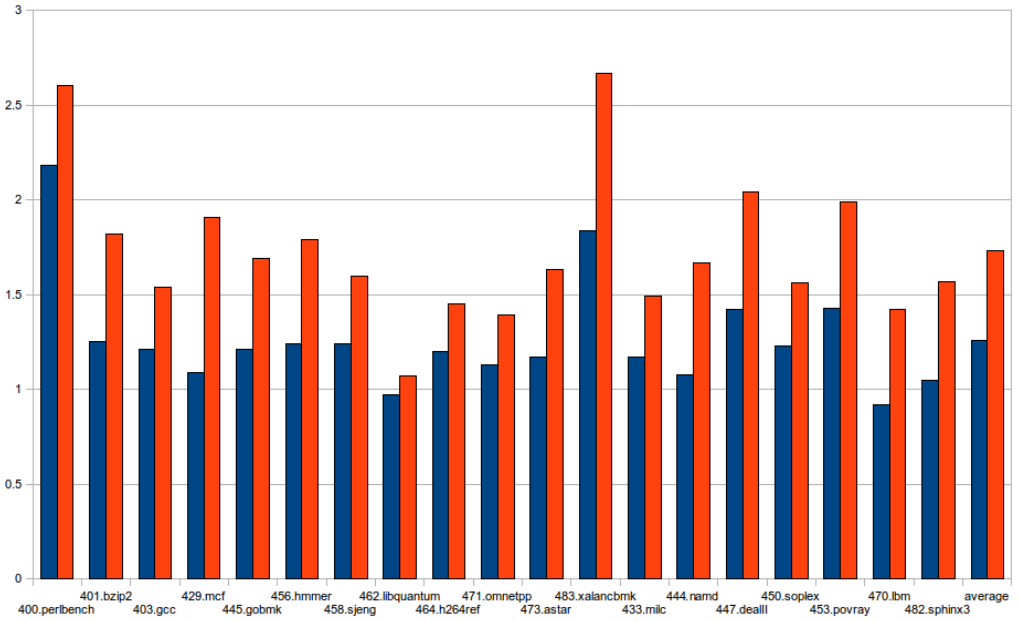
https://github.com/google/sanitizers/wiki/AddressSanitizer

Compile with `clang –fsanitize=address`

Asan is a memory error detector for C/C++. It finds:

- Use after free (dangling pointer dereference)
- Heap buffer overflow
- Stack buffer overflow
- Global buffer overflow
- Use after return
- Use after scope
- Initialization order bugs
- Memory leaks

Slowdown on SPEC CPU 2006



What is Fuzz Testing?

Approach:

Generate inputs randomly until program crashes

What is Fuzz Testing?

Approach:

Generate network packets randomly until program crashes

EvoMaster: Evolutionary Multi-context Automated System Test Generation. [ICST 18’]

RESTler: Stateful REST API Fuzzing [ICSE 19’]

SPIDER: Fuzzing for Stateful Performance Issues in the ONOS Software-Defined Network Controller [ICST 25’]

PASTALAB
Program Analysis, Software Testing, and Applications Laboratory

What is Fuzz Testing?

Approach:

Generate thread schedulings randomly until program crashes

Lincheck: A Practical Framework for Testing Concurrent Data Structures on JVM [CAV 23’]

Fray: An Efficient General-Purpose Concurrency Testing Platform for the JVM [OOPSLA 25’]

PASTALAB

Program Analysis, Software Testing, and Applications Laboratory

What is Fuzz Testing?

Approach:

Generate program enviroment randomly until program crashes

Program Environment Fuzzing [CCS 24’]

What is Fuzz Testing?

Approach:

Generate inputs randomly until program runs slow

SlowFuzz: Automated Domain-Independent Detection of Algorithmic Complexity Vulnerabilities [CCS 17’]

PerfFuzz: Automatically Generating Pathological Inputs [ISSTA 18’]

PASTALAB

Program Analysis, Software Testing, and Applications Laboratory

What is Fuzz Testing?

Approach:

Generate inputs randomly until mutants crashes

Guiding Greybox Fuzzing with Mutation Testing [ISSTA 23’]

How do you fuzz concurrent software?

To identify and eliminate race conditions

```
1  class Foo extends Thread {
2    static Object o = new Object();
3    static AtomicInteger a = AtomicInteger();
4    static volatile int b;
5    public void run() {
6      int x = a.getAndIncrement();
7      synchronized(o) {
8        if (x == 0) {
9          o.wait();
10       } else {
11         o.notify();
12       }
13     }
14     b = x;
15   }
16   public static void main(...) {
17     Foo[] threads = {new Foo(), new Foo()};
18     for (var thread : threads) thread.start();
19     for (var thread : threads) thread.join();
20     assert (b == 1);
21   }
22 }
```

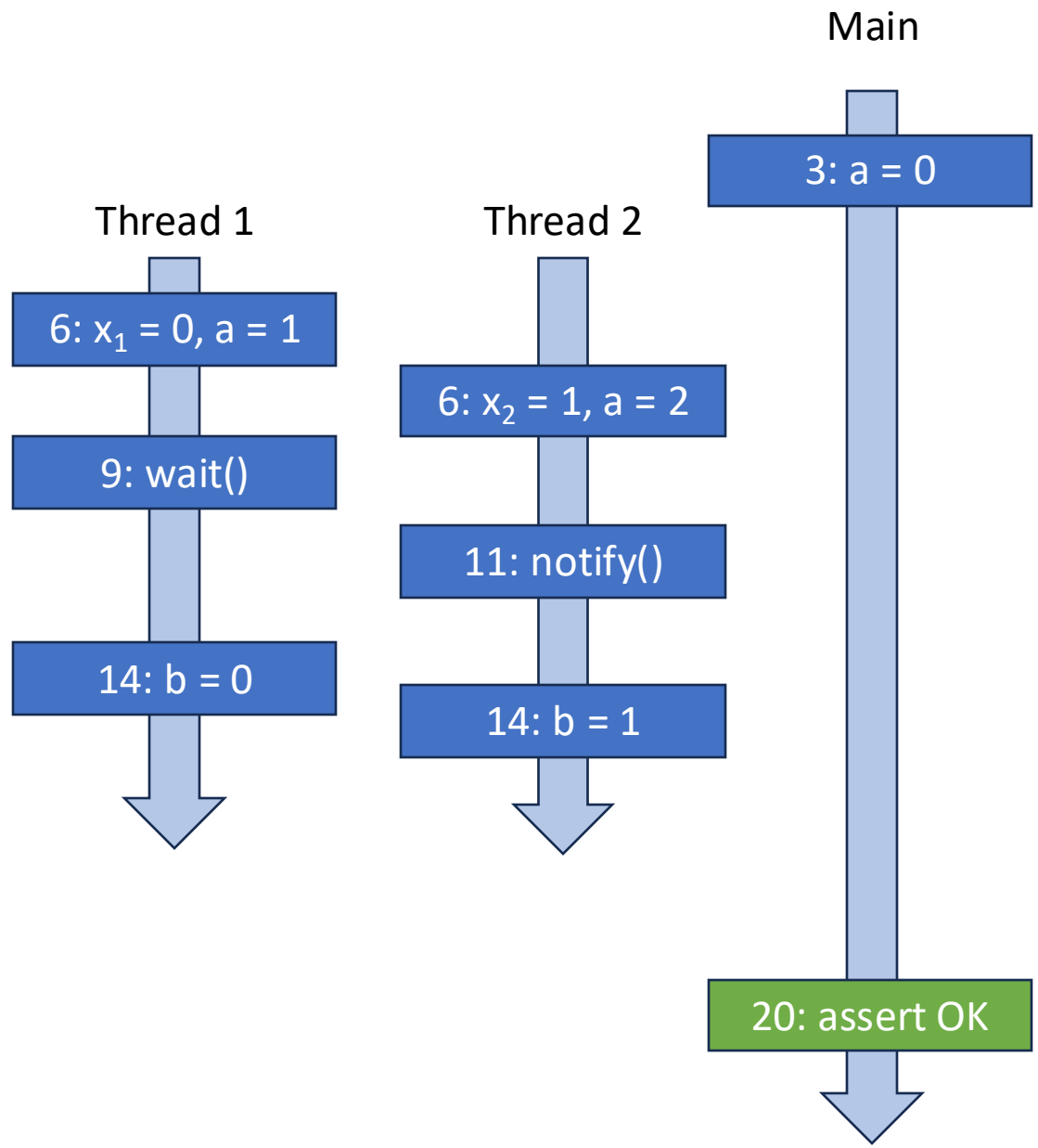
```
1  class Foo extends Thread {
2    static Object o = new Object();
3    static AtomicInteger a = AtomicInteger();
4    static volatile int b;
5    public void run() {
6      int x = a.getAndIncrement();
7      synchronized(o) {
8        if (x == 0) {
9          o.wait();
10       } else {
11         o.notify();
12       }
13     }
14     b = x;
15   }
16   public static void main(...) {
17     Foo[] threads = {new Foo(), new Foo()};
18     for (var thread : threads) thread.start();
19     for (var thread : threads) thread.join();
20     assert (b == 1);
21   }
22 }
```

```
1  class Foo extends Thread {
2    static Object o = new Object();
3    static AtomicInteger a = AtomicInteger();
4    static volatile int b;
5    public void run() {
6      int x = a.getAndIncrement();
7      synchronized(o) {
8        if (x == 0) {
9          o.wait();
10       } else {
11         o.notify();
12       }
13     }
14     b = x;
15   }
16   public static void main(...) {
17     Foo[] threads = {new Foo(), new Foo()};
18     for (var thread : threads) thread.start();
19     for (var thread : threads) thread.join();
20     assert (b == 1);
21   }
22 }
```

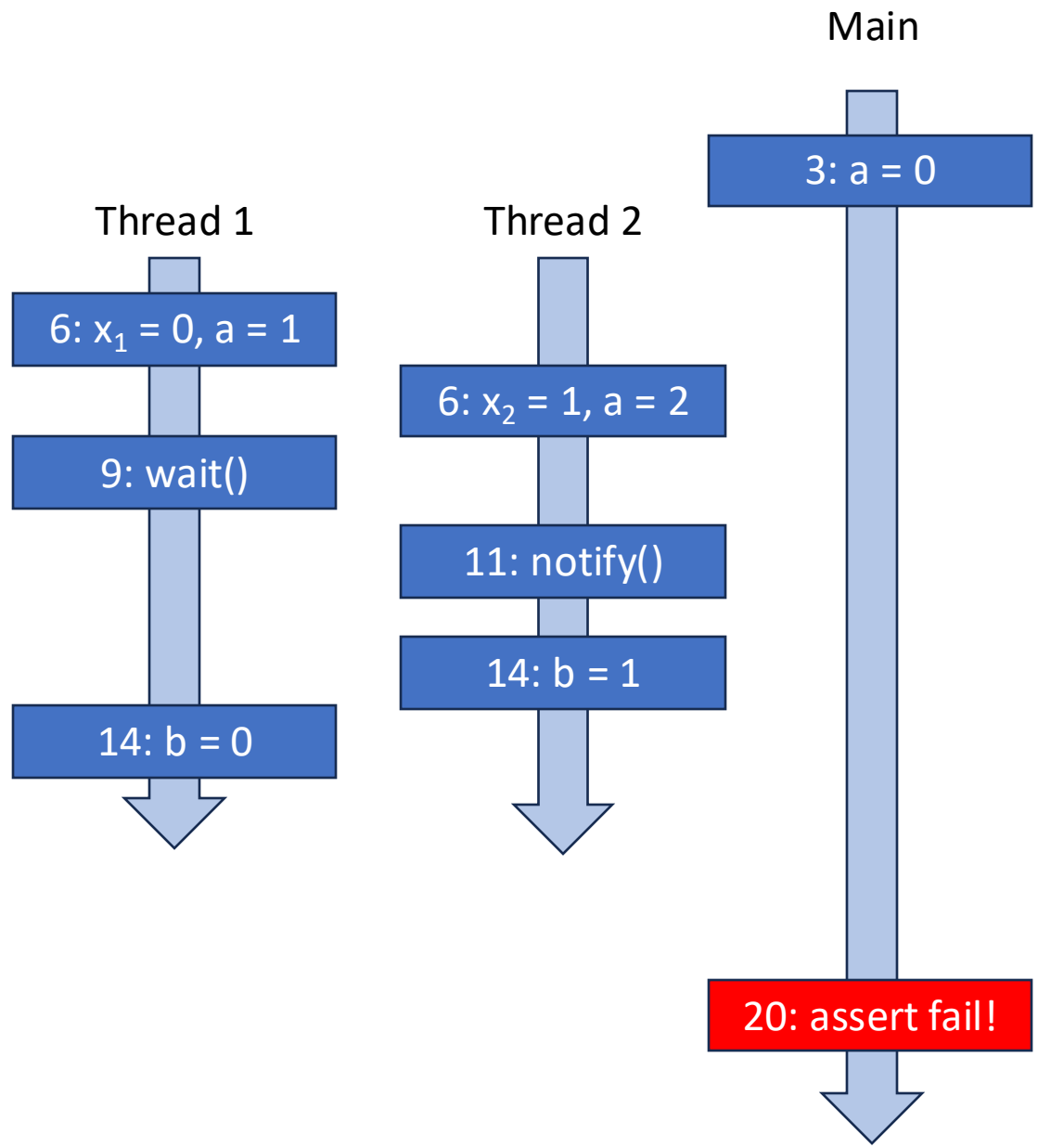
```
1  class Foo extends Thread {
2    static Object o = new Object();
3    static AtomicInteger a = AtomicInteger();
4    static volatile int b;
5    public void run() {
6      int x = a.getAndIncrement();
7      synchronized(o) {
8        if (x == 0) {
9          o.wait();
10       } else {
11         o.notify();
12       }
13     }
14     b = x;
15   }
16   public static void main(...) {
17     Foo[] threads = {new Foo(), new Foo()};
18     for (var thread : threads) thread.start();
19     for (var thread : threads) thread.join();
20     assert (b == 1);
21   }
22 }
```

```
1  class Foo extends Thread {
2    static Object o = new Object();
3    static AtomicInteger a = AtomicInteger();
4    static volatile int b;
5    public void run() {
6      int x = a.getAndIncrement();
7      synchronized(o) {
8        if (x == 0) {
9          o.wait();
10       } else {
11         o.notify();
12       }
13     }
14     b = x;
15   }
16   public static void main(...) {
17     Foo[] threads = {new Foo(), new Foo()};
18     for (var thread : threads) thread.start();
19     for (var thread : threads) thread.join();
20     assert (b == 1);
21   }
22 }
```

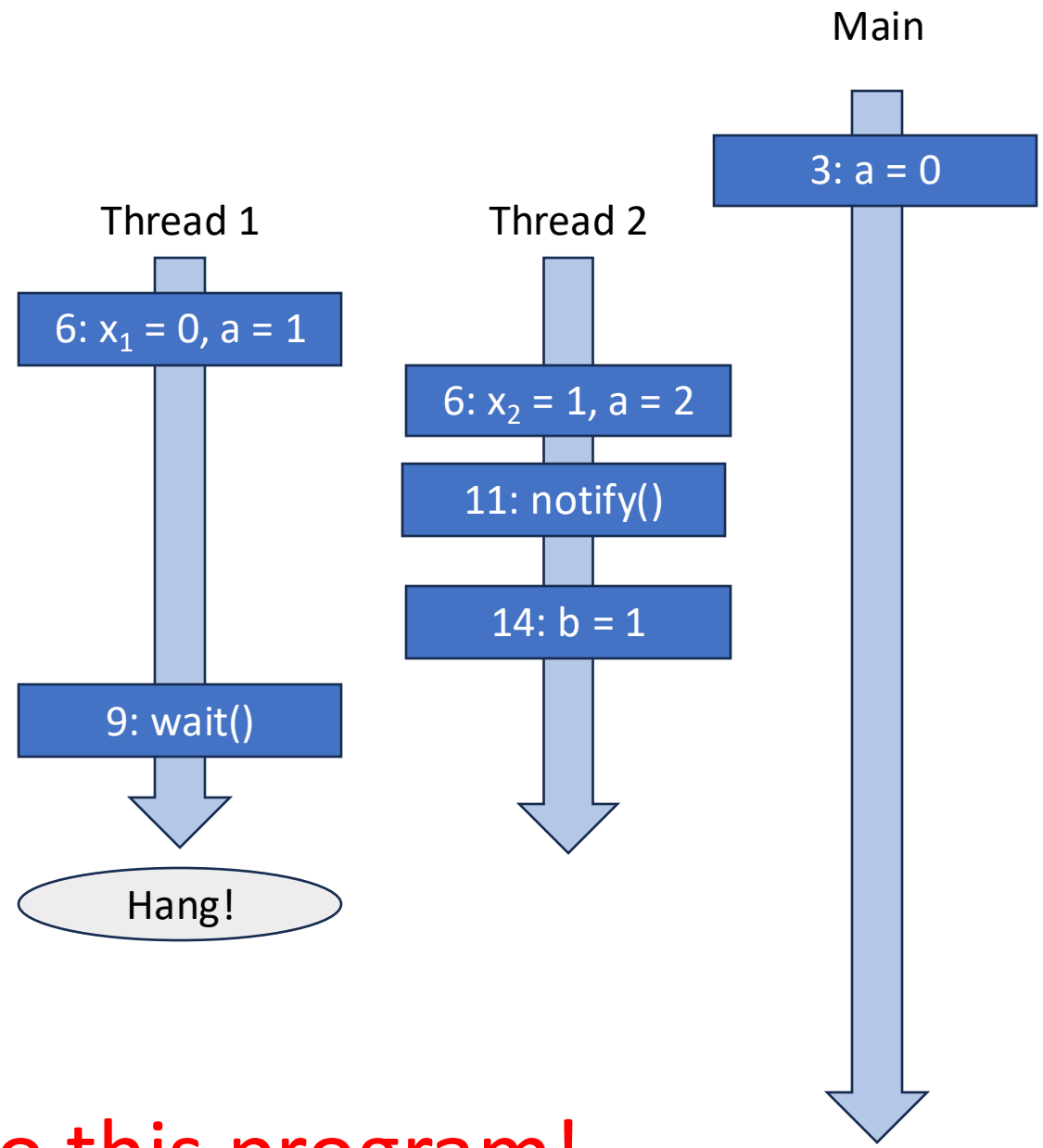
```
1  class Foo extends Thread {
2    static Object o = new Object();
3    static AtomicInteger a = AtomicInteger();
4    static volatile int b;
5    public void run() {
6      int x = a.getAndIncrement();
7      synchronized(o) {
8        if (x == 0) {
9          o.wait();
10       } else {
11         o.notify();
12       }
13     }
14     b = x;
15   }
16   public static void main(...) {
17     Foo[] threads = {new Foo(), new Foo()};
18     for (var thread : threads) thread.start();
19     for (var thread : threads) thread.join();
20     assert (b == 1);
21   }
22 }
```



```
1  class Foo extends Thread {
2    static Object o = new Object();
3    static AtomicInteger a = AtomicInteger();
4    static volatile int b;
5    public void run() {
6      int x = a.getAndIncrement();
7      synchronized(o) {
8        if (x == 0) {
9          o.wait();
10       } else {
11         o.notify();
12       }
13     }
14     b = x;
15   }
16   public static void main(...) {
17     Foo[] threads = {new Foo(), new Foo()};
18     for (var thread : threads) thread.start();
19     for (var thread : threads) thread.join();
20     assert (b == 1);
21   }
22 }
```



```
1  class Foo extends Thread {
2    static Object o = new Object();
3    static AtomicInteger a = AtomicInteger();
4    static volatile int b;
5    public void run() {
6      int x = a.getAndIncrement();
7      synchronized(o) {
8        if (x == 0) {
9          o.wait();
10       } else {
11         o.notify();
12       }
13     }
14     b = x;
15   }
16   public static void main(...) {
17     Foo[] threads = {new Foo(), new Foo()};
18     for (var thread : threads) thread.start();
19     for (var thread : threads) thread.join();
20     assert (b == 1);
21   }
22 }
```



No input provided to this program!

What do we want from a concurrency testing system?

1. Support *controlled concurrency*
   (i.e., run program with specified thread interleaving *schedule*)

2. Enable *search* methods
   (i.e., systematically or randomly vary thread schedules)

3. Push-button general-purpose
   (i.e., not require testing DSLs, clocks, mocks, or specialized oracles)

4. Work everywhere and efficiently
   (i.e., can target arbitrary real-world JVM programs)

Fray: An Efficient General-Purpose Concurrency Testing Platform for the JVM

Push-button wrapper that performs bytecode instrumentation

e.g. Replace `` `java <cmd>` `` with `` `fray <cmd>` ``

(or annotate Junit tests with `` `@ConcurrencyTest` ``)

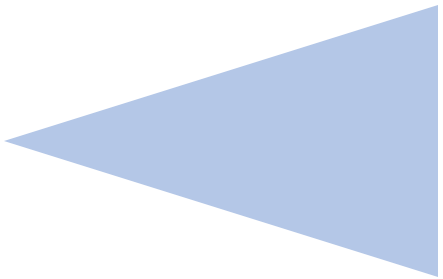
Scheduler thread randomly samples app-thread interleavings

(or `` `--scheduler <fancy_alg>` ``)

Saves crash files for repro

(`` `fray --replay=<file> <cmd>` ``)

```
1  class Foo extends Thread {
2    static Object o = new Object();
3    static AtomicInteger a = AtomicInteger();
4    static volatile int b;
5    public void run() {
6      int x = a.getAndIncrement();
7      synchronized(o) {
8        if (x == 0) {
9          o.wait();
10       } else {
11         o.notify();
12       }
13     }
14     b = x;
15   }
16   public static void main(...) {
17     Foo[] threads = {new Foo(), new Foo()};
18     for (var thread : threads) thread.start();
19     for (var thread : threads) thread.join();
20     assert (b == 1);
21   }
22 }
```



```
synchronized(o) {
  if (x == o) {
    o.wait();
  } else {
    o.notify();
  }
}
```

Thread 1:

```
synchronized(o) {
  o.wait();
}
```

Thread 2:

```
synchronized(o) {
  o.notify();
}
```

Challenges:

Wait(native method) implicitly unlocks the corresponding monitor lock.

Threads can be waken up in random order.

Original:

Thread 1:

```
synchronized(o) {
  o.wait();
}
```

Fray Instrumented:

Thread 1:

```
🔒[1, o].lock()
synchronized(o) {
  🔒[1, o].unlock();
  do {
    o.wait();
  } while(!🔒[1, o].tryLock());
}
🔒[1, o].unlock();
```

Thread 1:

🔒[1, o].*lock*()
synchronized(o) {
  🔒[1, o].*unlock*();
  do {
    o.*wait*();
  } while(!🔒[1, o].*tryLock*());
}
🔒[1, o].*unlock*()

Thread 2:

🔒[2, o].*lock*()
synchronized(o) {
  o.*notifyAll*();
}
🔒[2, o].*unlock*()

A **shadow lock** 🔒[t, r] is associated with thread *t* and resource *r* (e.g., 🔒[1, o]).

Access invariant: App thread *t* can access resource *r* only if it owns shadow lock 🔒[t, r].

Mutual exclusion invariant: At most one app thread *t* can own shadow lock 🔒[*, r] for any given resource *r.*

Control policy: Only one thread runs at a time, until next shadow-lock acquire/release; Fray scheduler chooses next 🔒[t, r] to release only if thread *t* can immediately access resource *r*.

A **shadow lock** 🔒[t, r] is associated with thread *t* and resource *r* (e.g., 🔒[1, o]).

acquire/release; Fray scheduler chooses next 🔒[t, r] to release only if thread *t* can immediately access resource *r*.

A **shadow lock** 🔒[t, r] is associated with thread *t* and resource *r* (e.g., 🔒[1, o]).

Goal: Providing full control of thread scheduling without replacing any existing concurrency primitive.

acquire/release; Fray scheduler chooses next 🔒[t, r] to release only if thread *t* can immediately access resource *r*.

Thread 1:

```
[1, o].lock()
synchronized(o) {
  [1, o].unlock();
  do {
    o.wait();
  } while(![1, o].tryLock());
}
[1, o].unlock()
```

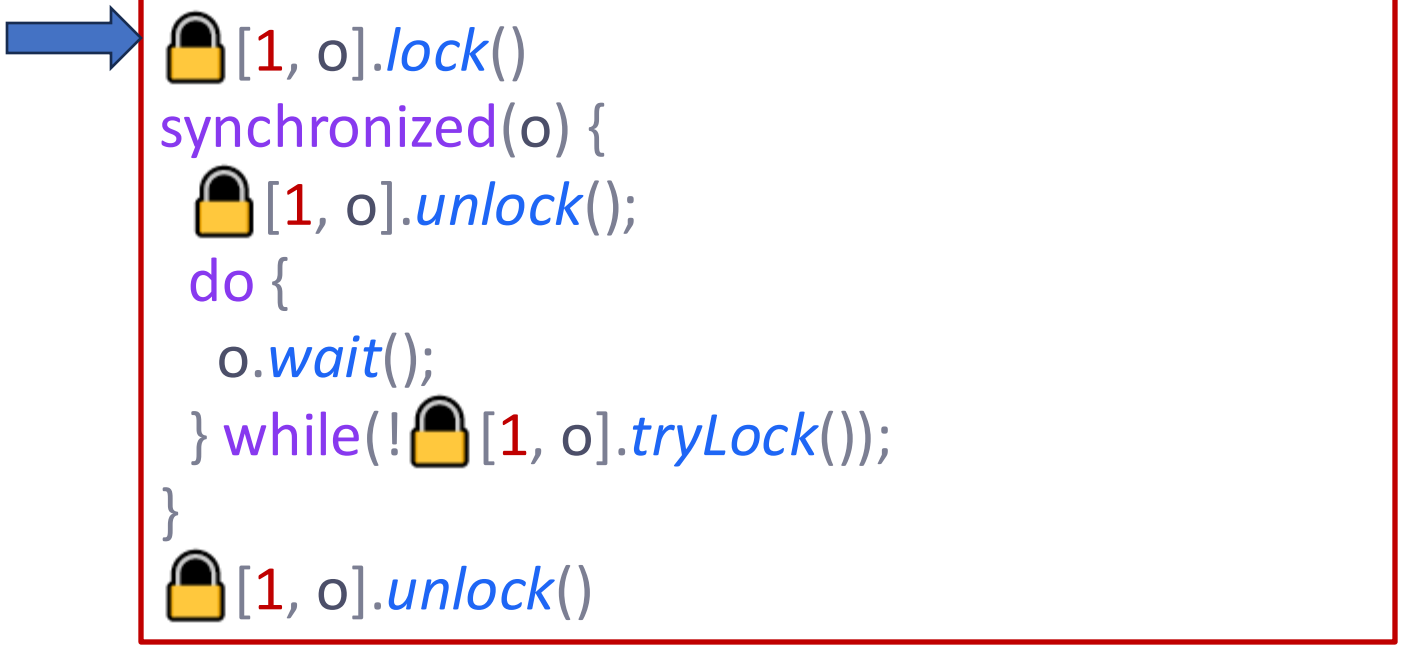
Thread 2:

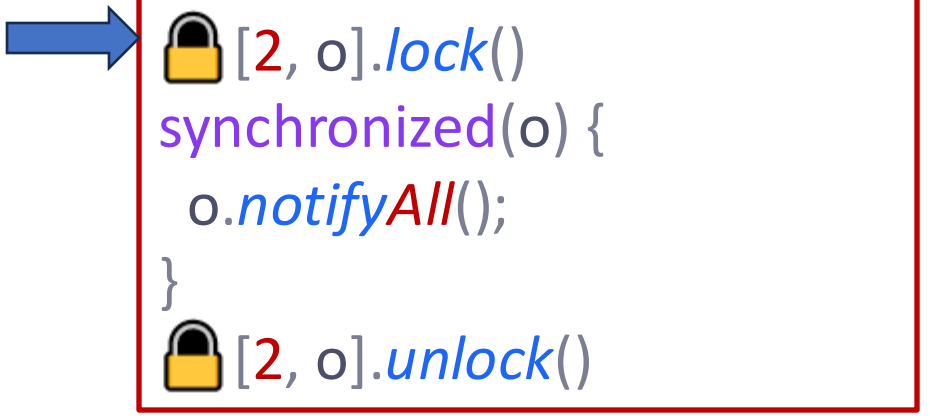
```
[2, o].lock()
synchronized(o) {
  o.notifyAll();
}
[2, o].unlock()
```

Lock Status:

[1, o]: Fray

[2, o]: Fray

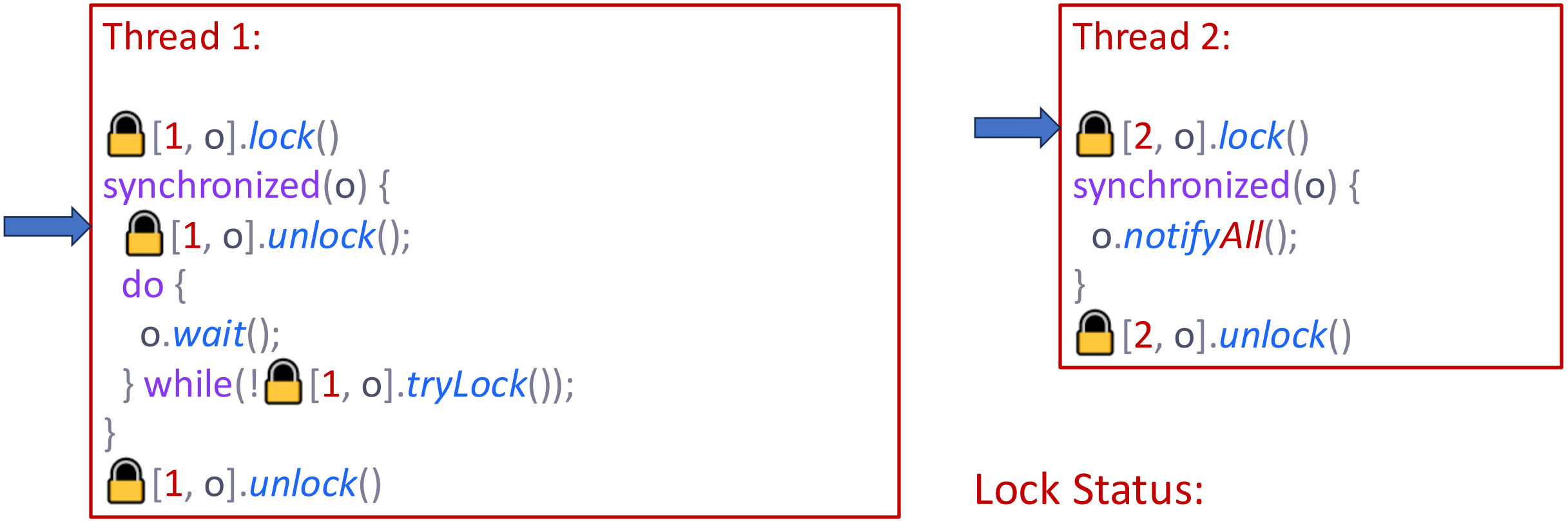




Lock Status:

[1, o]:

[2, o]: Fray



Mutual exclusion invariant: At most one app thread *t* can own shadow lock [*, r] for any given resource *r.*

Thread 1:

```
[1, o].lock()
synchronized(o) {
  [1, o].unlock();
  do {
    o.wait();
  } while(![1, o].tryLock());
}
[1, o].unlock()
```

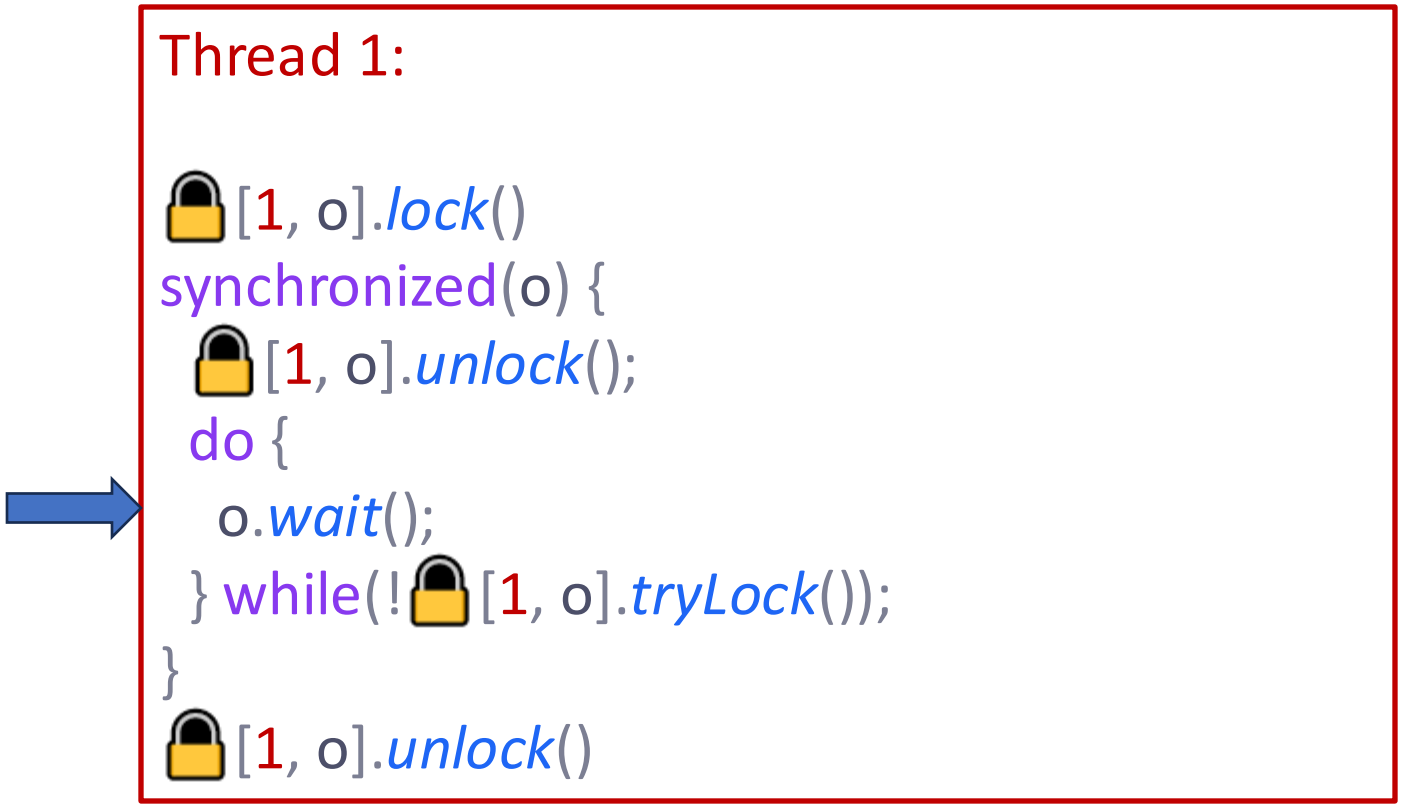
Thread 2:

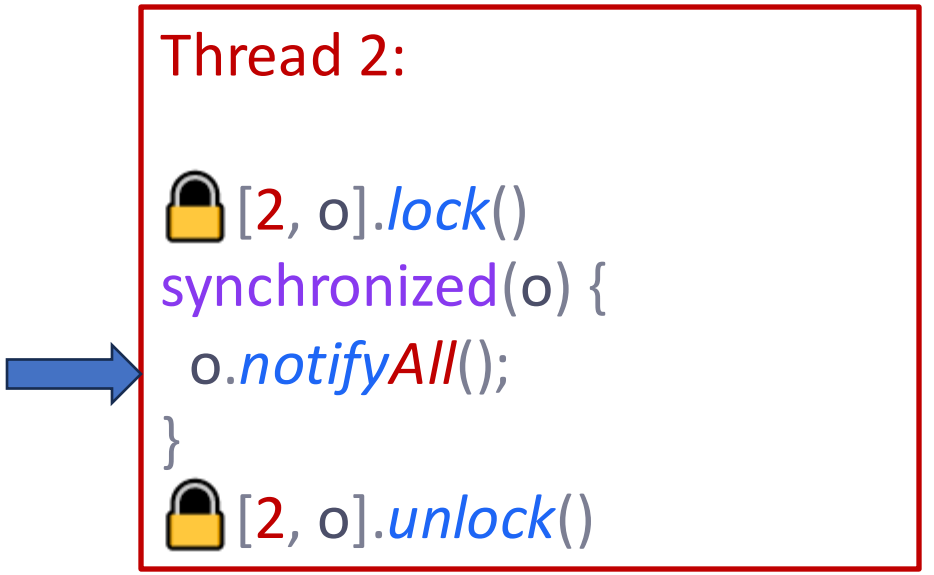
```
[2, o].lock()
synchronized(o) {
  o.notifyAll();
}
[2, o].unlock()
```

Lock Status:

[1, o]: Fray

[2, o]: Fray

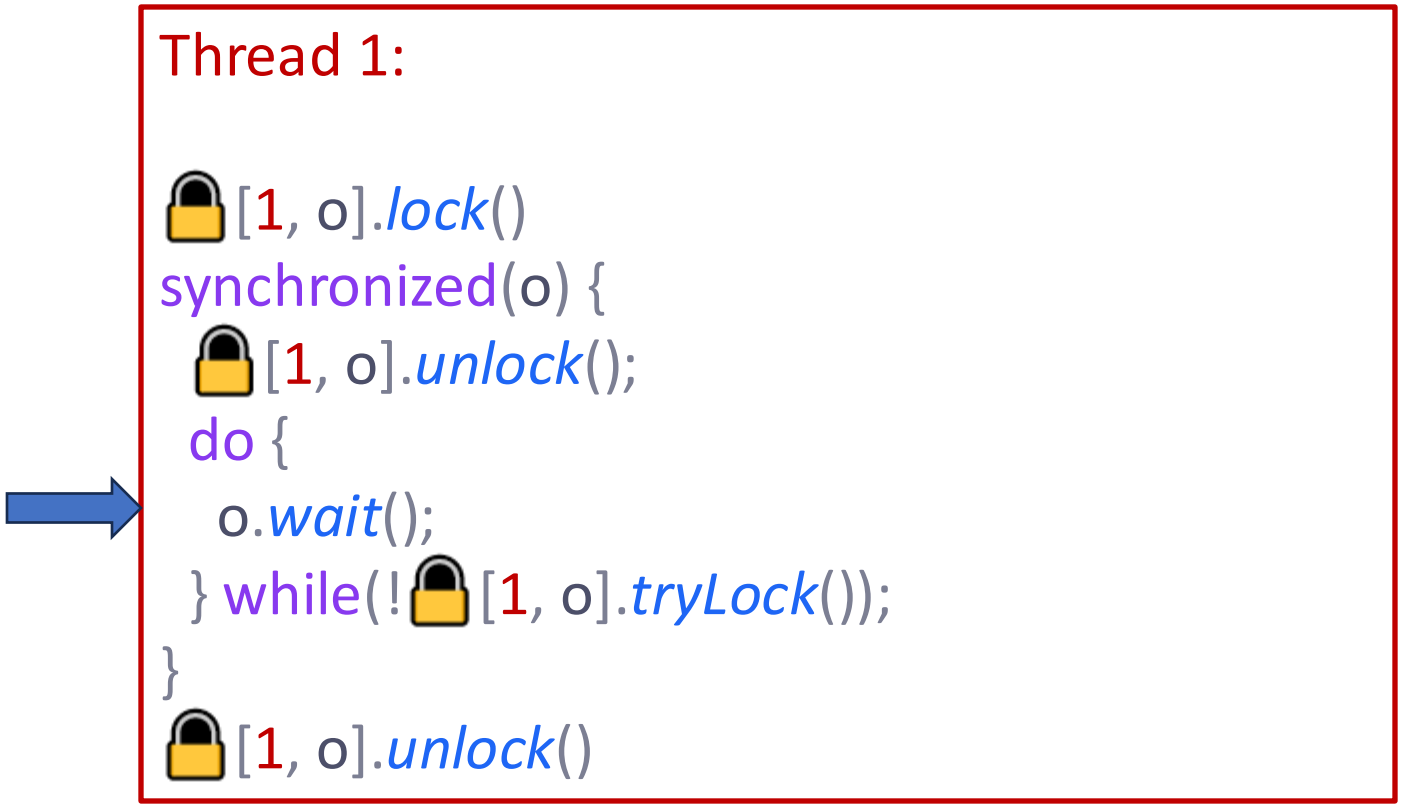


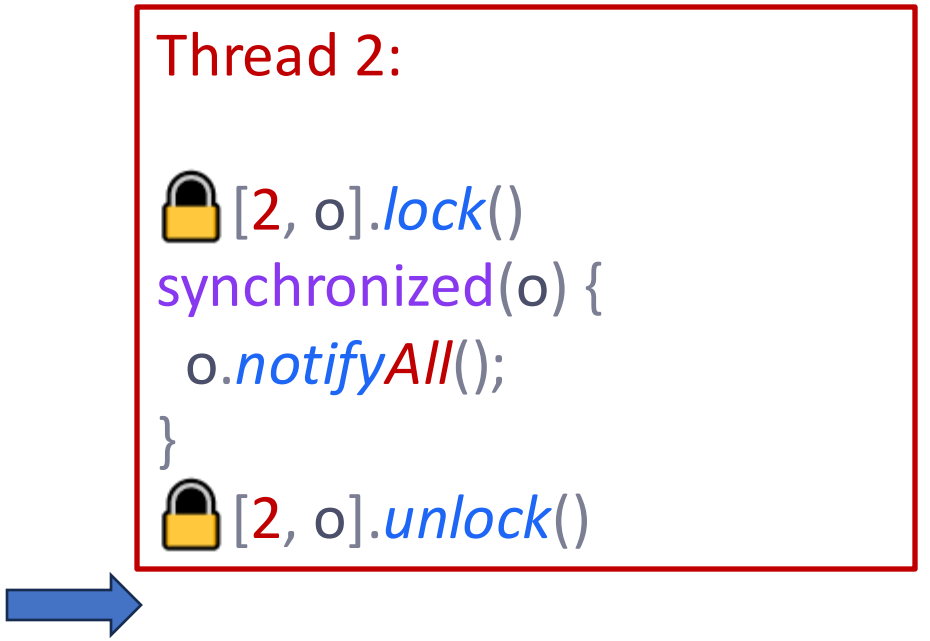


Lock Status:

[1, o]: Fray

[2, o]: Thread 2





Lock Status:

[1, o]: Fray

[2, o]: Fray

Thread 1:

```
[1, o].lock()
synchronized(o) {
  [1, o].unlock();
  do {
    o.wait();
  } while(![1, o].tryLock());
}
[1, o].unlock()
```

Thread 2:

```
[2, o].lock()
synchronized(o) {
  o.notifyAll();
}
[2, o].unlock()
```

Lock Status:

[1, o]: Fray

[2, o]: Fray

Thread 1:

```
[1, o].lock()
synchronized(o) {
  [1, o].unlock();
  do {
    o.wait();
  } while(![1, o].tryLock());
}
[1, o].unlock()
```

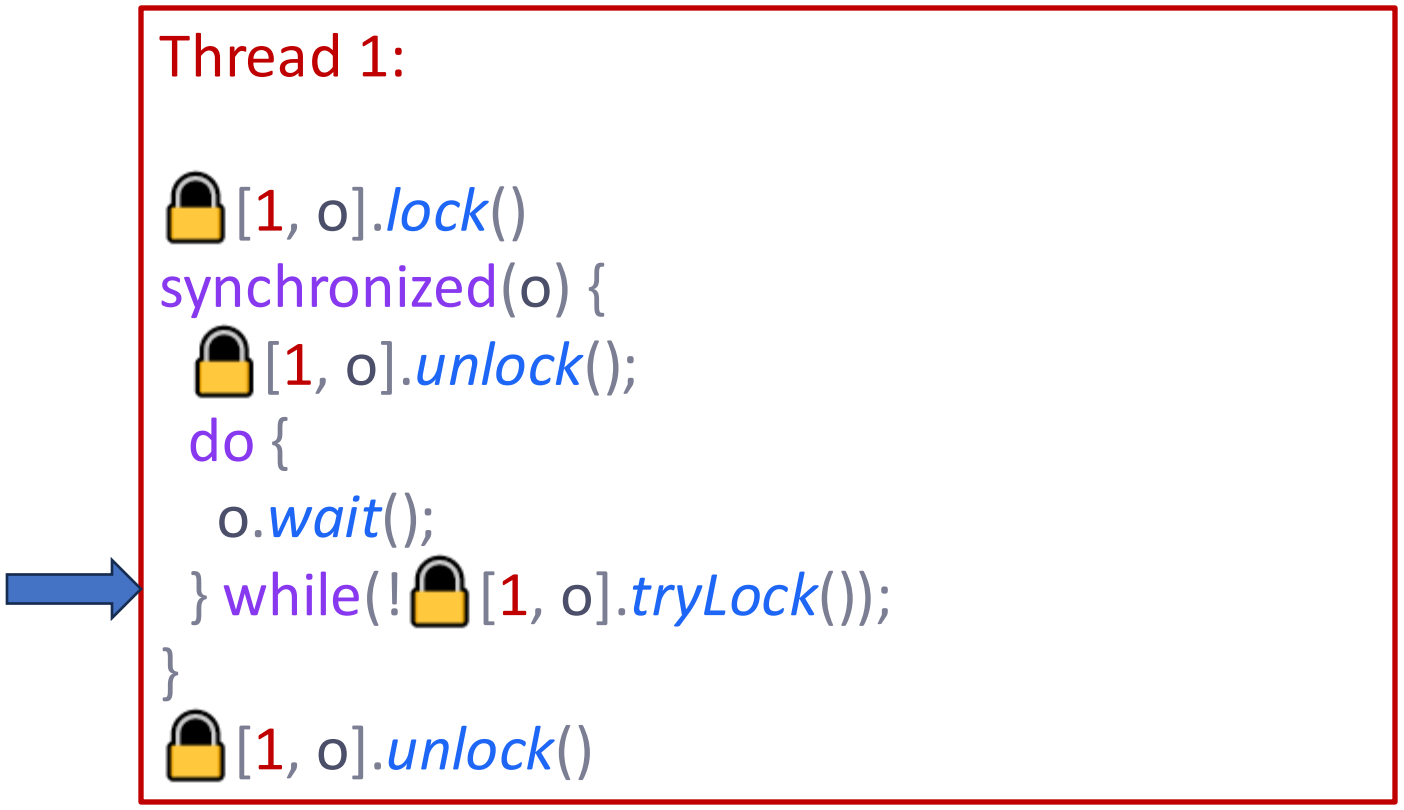
Thread 2:

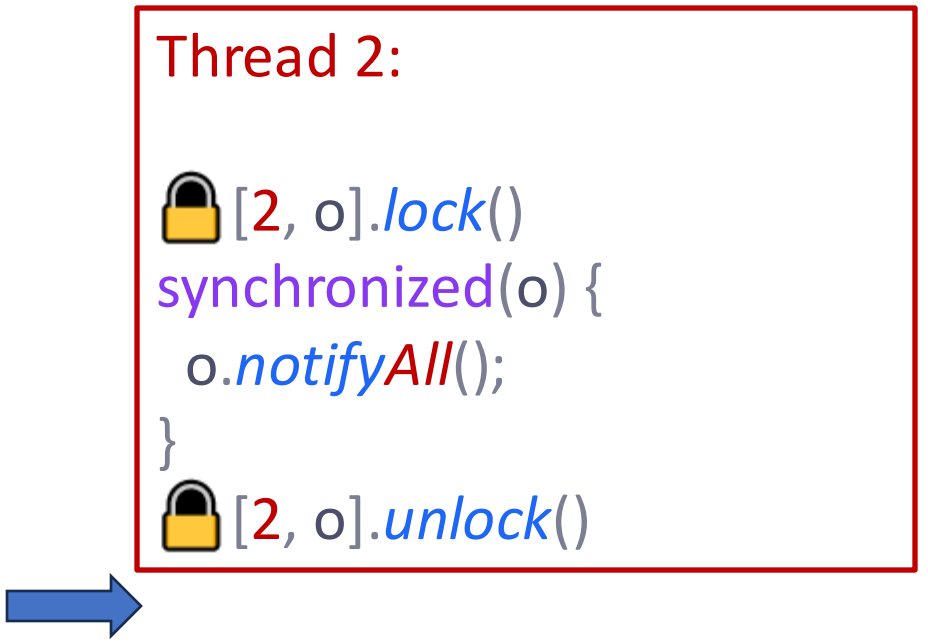
```
[2, o].lock()
synchronized(o) {
  o.notifyAll();
}
[2, o].unlock()
```

Lock Status:

[1, o]: Fray

[2, o]: Fray





Fray calls o.*notifyAll*() to unblock Thread 1.

Lock Status:

[1, o]:
[2, o]: Fray

Thread 1:

```
[1, o].lock()
synchronized(o) {
  [1, o].unlock();
  do {
    o.wait();
  } while(![1, o].tryLock());
}
[1, o].unlock()
```

Thread 2:

```
[2, o].lock()
synchronized(o) {
  o.notifyAll();
}
[2, o].unlock()
```

Lock Status:

[1, o]: Thread 1

[2, o]: Fray

Thread 1:

```
🔒[1, o].lock()
synchronized(o) {
  🔒[1, o].unlock();
  do {
    o.wait();
  } while(!🔒[1, o].tryLock());
}
🔒[1, o].unlock()
```

Thread 2:

```
🔒[2, o].lock()
synchronized(o) {
  o.notifyAll();
}
🔒[2, o].unlock()
```

Lock Status:

🔒[1, o]: Fray

🔒[2, o]: Fray

Fray Provides Strong Correctness Guarantees

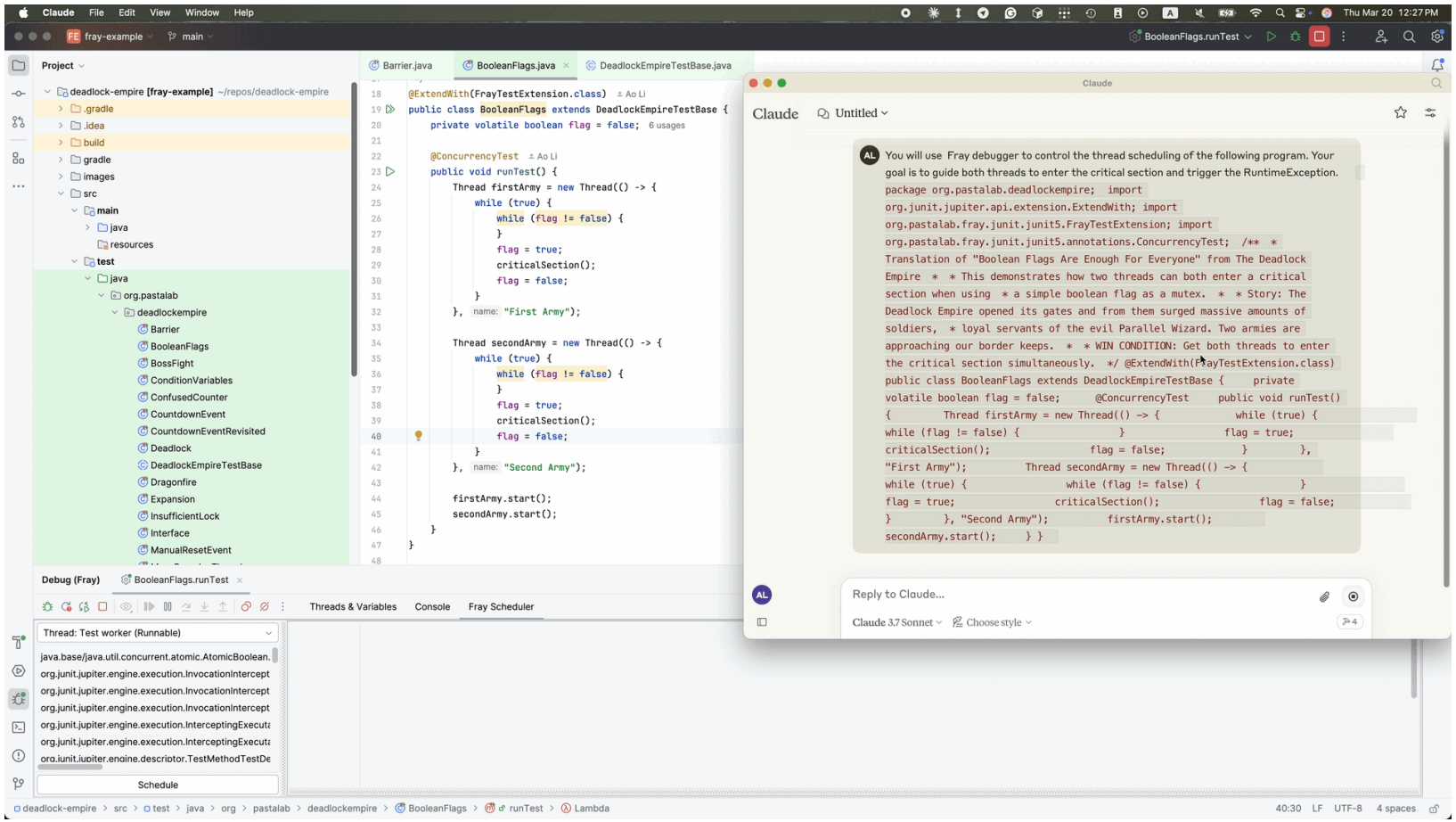
Soundness:

Any concurrency bug (e.g., racy exception, deadlock) observed via Fray could also manifest in the original program.

Completeness:

Any concurrency bug (e.g., racy exception, deadlock) that could manifest in the original program can be exercised via Fray with some schedule.

Fray AI Debugging Preview



https://github.com/cmu-pasta/fray

Fuzz Testing

Ao Li

Ph.D. Candidate

Software and Societal Systems (S3D)
CMU School of Computer Science

Email: aoli@cmu.edu