No Harness, No Problem:

Oracle-guided Harnessing for Auto-generating C API Fuzzing Harnesses

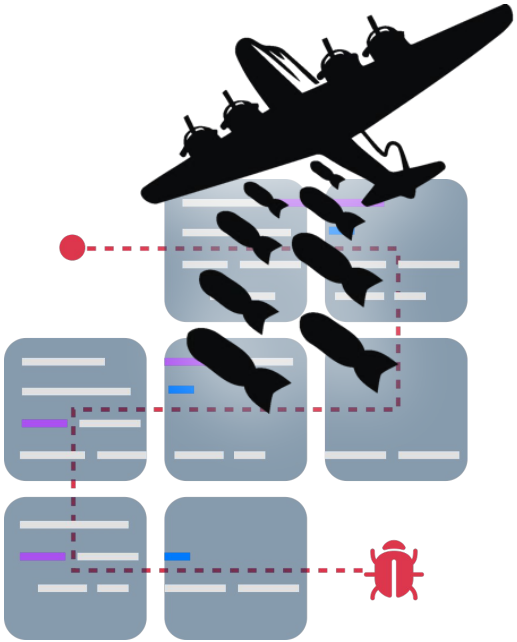
Gabriel Sherman
University of Utah

Stefan Nagy
University of Utah

FUTURES^3
LAB FUTURE TECHNOLGY FOR USABLE, RELIABLE, & EFFICIENT SECURITY OF SOFTWARE & SYSTEMS

Fuzzing: Automated Bug Discovery

- Today's leading **automated bug-finding strategy**
- Fundamental steps:
  - Generate many inputs via randomized **mutation**
  - **Execute** and check results (**crashes**, **hangs**, etc.)
  - New coverage? **Save and re-mutate** that input!



Fuzzing: Automated Bug Discovery

- Today's leading **automated bug-finding strategy**
- Fundamental steps:
  - Generate many inputs via randomized **mutation**
  - **Execute** and check results (**crashes**, **hangs**, etc.)
  - New coverage? **Save and re-mutate** that input!
- Revealed **thousands of bugs** in modern software
  - Apps and libraries
  - Operating systems
  - IoT devices and more!

Harnesses: the Foundation of Fuzzing

- Wrapper programs that feed fuzzer input to API functions

```
int harness(const uint8_t *Data, size_t Size){
  initGEOS(geos_msg_handler, geos_msg_handler);
  Reader *reader = Reader_create();
  Geometry *g1 = Reader_read(reader, Data);
    if (g1 != NULL) {
      Geometry *g2 = FromWKB_buf(Data, Size);
      if (g2 != NULL) {
        Geometry *g3 = Intersection(g1, g2);
        Geom_destroy(g3);
        Geom_destroy(g2);
     }
                        ...
```

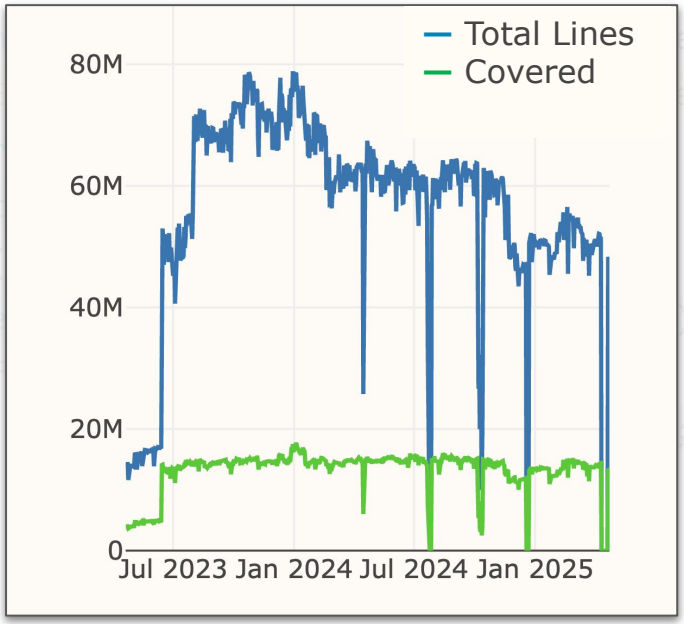
Harnesses: the Foundation of Fuzzing

- Wrapper programs that feed fuzzer input to API functions
- Each part serves a purpose:
  - **Library initialization**
  - **Data entrypoints**
  - **Data processing**

```
int harness(const uint8_t *Data, size_t Size){
  initGEOS(geos_msg_handler, geos_msg_handler);
  Reader *reader = Reader_create();
  Geometry *g1 = Reader_read(reader, Data);
    if (g1 != NULL) {
      Geometry *g2 = FromWKB_buf(Data, Size);
      if (g2 != NULL) {
        Geometry *g3 = Intersection(g1, g2);
        Geom_destroy(g3);
        Geom_destroy(g2);
     }
                        ...
```

Harnesses: the Foundation of Fuzzing

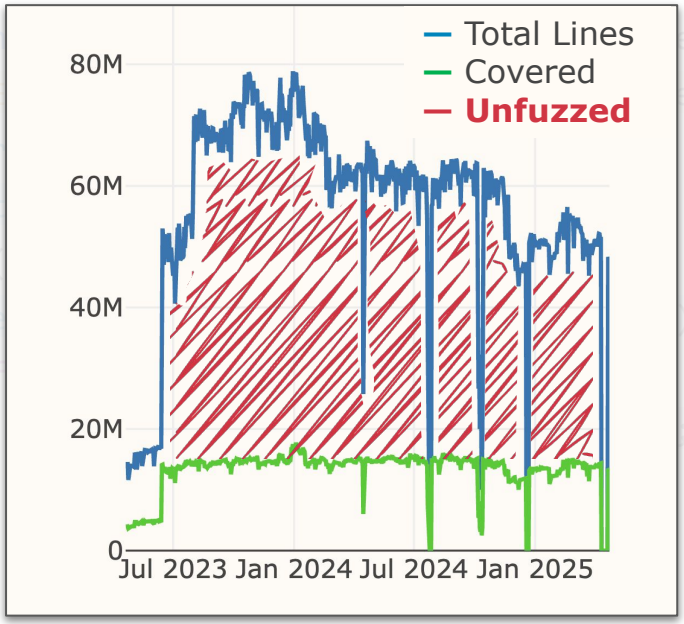
- Wrapper programs that feed fuzzer input to API functions
- Each part serves a purpose:
  - **Library initialization**
  - **Data entrypoints**
  - **Data processing**
- **Problem:** so much of today's code remains **un-harnessed**



Source: Google's OSS-Fuzz Introspector

Harnesses: the Foundation of Fuzzing

- Wrapper programs that feed fuzzer input to API functions
- Each part serves a purpose:
  - **Library initialization**
  - **Data entrypoints**
  - **Data processing**
- **Problem:** so much of today's code remains **un-harnessed**
  - Hence, it is **not being fuzzed**



Source: Google's OSS-Fuzz Introspector

Prior Work: Automatic Harness Generation

- **Idea:** create harnesses automatically exercising diverse API usage

How API Usage is Learned:	Advantages:	Disadvantages:
Available API **reference code** (Fudge, FuzzGen, Utopia, Rubick)	Generate harnesses **fast**	Reference code **covers very little**

Prior Work: Automatic Harness Generation

- **Idea:** create harnesses automatically exercising diverse API usage

How API Usage is Learned:	Advantages:	Disadvantages:
Available API **reference code** (Fudge, FuzzGen, Utopia, Rubick)	Generate harnesses **fast**	Reference code **covers very little**
Pre-written API **specification** (GraphFuzz)	Higher **semantic validity**	Specification writing is **unscalable**

Prior Work: Automatic Harness Generation

- **Idea:** create harnesses automatically exercising diverse API usage

How API Usage is Learned:	Advantages:	Disadvantages:
Available API **reference code** (Fudge, FuzzGen, Utopia, Rubick)	Generate harnesses **fast**	Reference code **covers very little**
Pre-written API **specification** (GraphFuzz)	Higher **semantic validity**	Specification writing is **unscalable**
On-the-fly **random mutation** (Hopper)	Harnesses **far more code**	Broken semantics = **false crashes**

Prior Work: Automatic Harness Generation

- **Idea:** create harnesses automatically exercising diverse API usage
  - Though **Hopper** eschews specifications or reference code, **it often breaks API semantics**

```
void handler(const char *fmt, ...){
  exit(EXIT_FAILURE);
}
int main(){
  ContextHandle_HS *c = GEOS_init_r();
  Context_setErrorHandler_r(c, handler);
  OrientPolygons_r(c, ...);    // Crash!
}
```

Missed Initialization

```
ContextHandle_HS *c = GEOS_init_r();
Geom_t *x = Geom_createEmptyPoint_r(c);
Geom_t *y = Geom_createCollection_r(c,-9,&x,6);
if (y == NULL) return 0;
Normalize_r(c, y);    // Crash!
```

Missed Sanity Checks

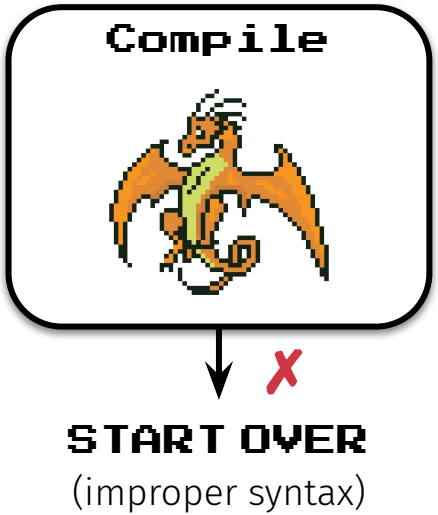
```
ContextHandle_HS *c = GEOS_init_r();
WKBReader_t *r = WKBReader_create_r(c);
WKBReader_destroy(r);
WKBReader_destroy_r(c, r);    // Crash!
```

Erroneous Call Sequences

Motivation: how can we harness **correctly**—without turning back to **specifications** or **reference code**?

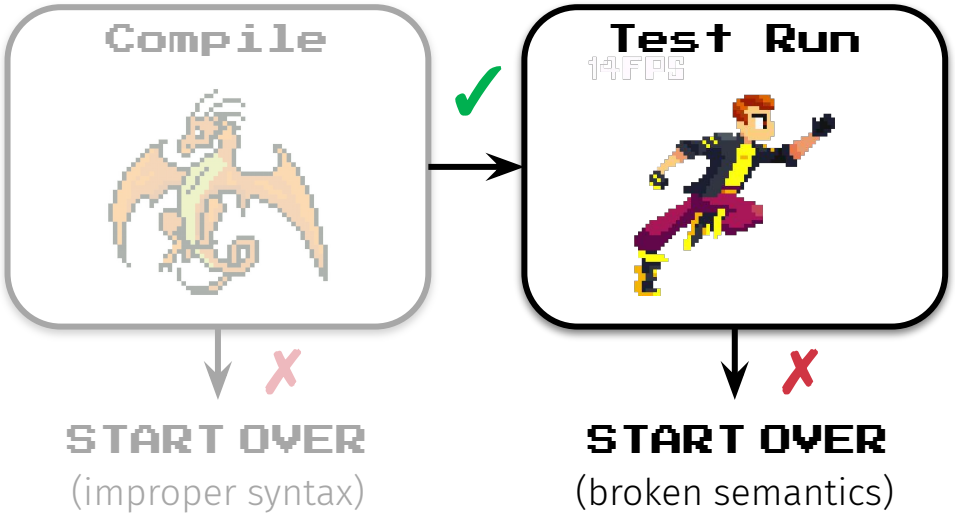
Inspiration: Harnessing by Hand

- Manual harness development spans **three sequential steps**
  - Failures at any step signal the harness is **wrong**, prompting **refinement**



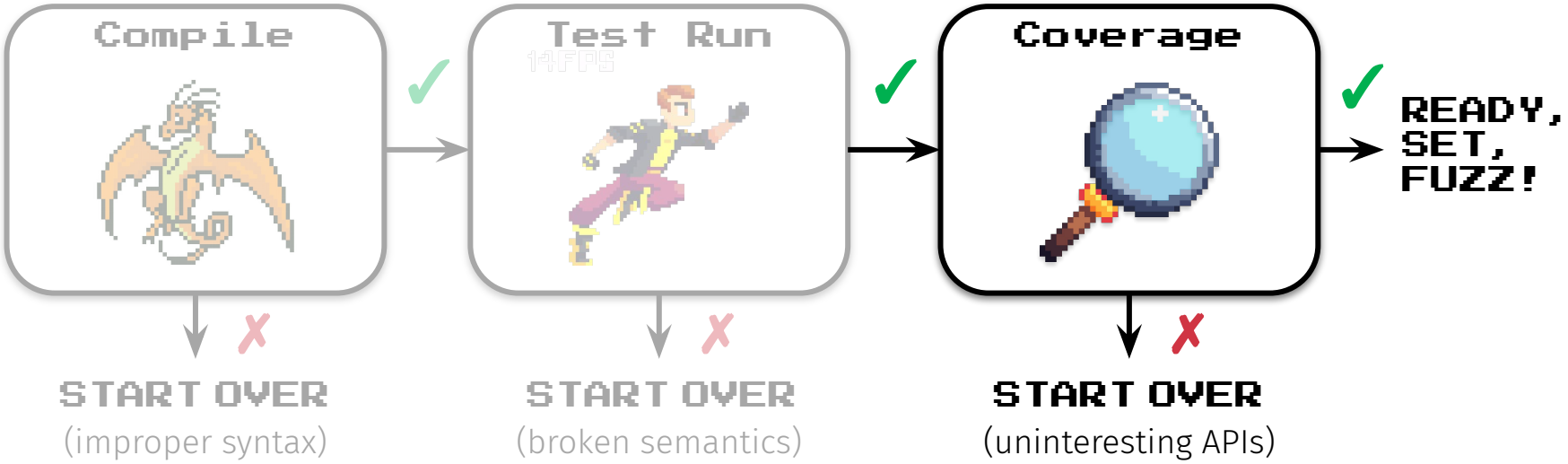
Inspiration: Harnessing by Hand

- Manual harness development spans **three sequential steps**
  - Failures at any step signal the harness is **wrong**, prompting **refinement**



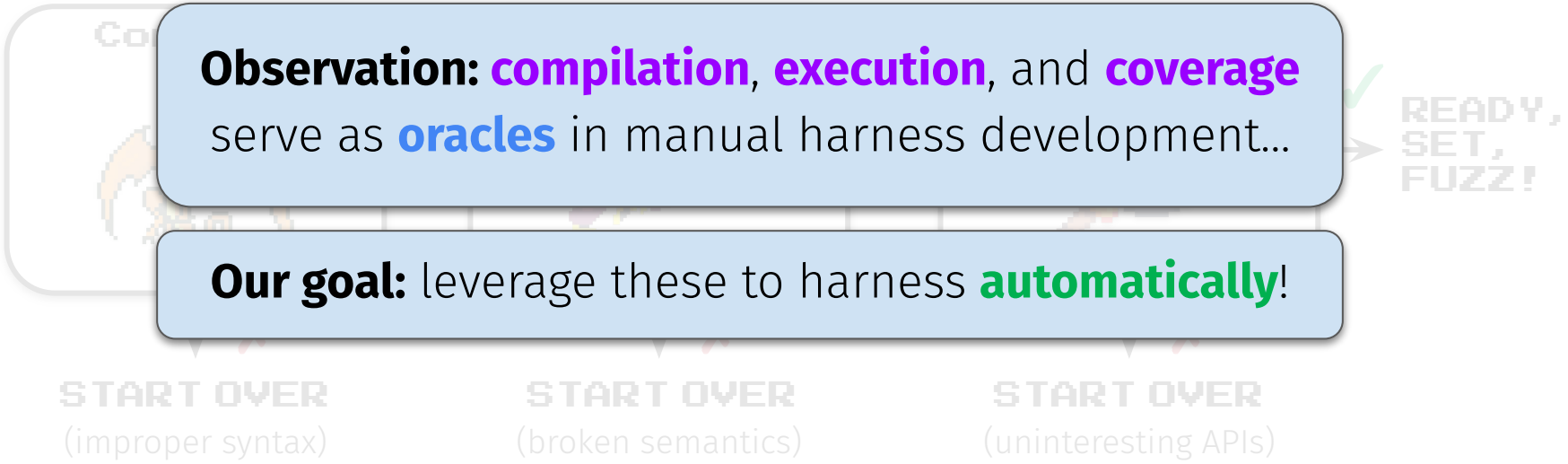
Inspiration: Harnessing by Hand

- Manual harness development spans **three sequential steps**
  - Failures at any step signal the harness is **wrong**, prompting **refinement**



Inspiration: Harnessing by Hand

- Manual harness development spans **three sequential steps**
  - Failures at any step signal the harness is **wrong**, prompting **refinement**



Observation: compilation, **execution**, and **coverage** serve as **oracles** in manual harness development…

Our goal: leverage these to harness **automatically**!

Our Solution: Oracle-guided Harnessing

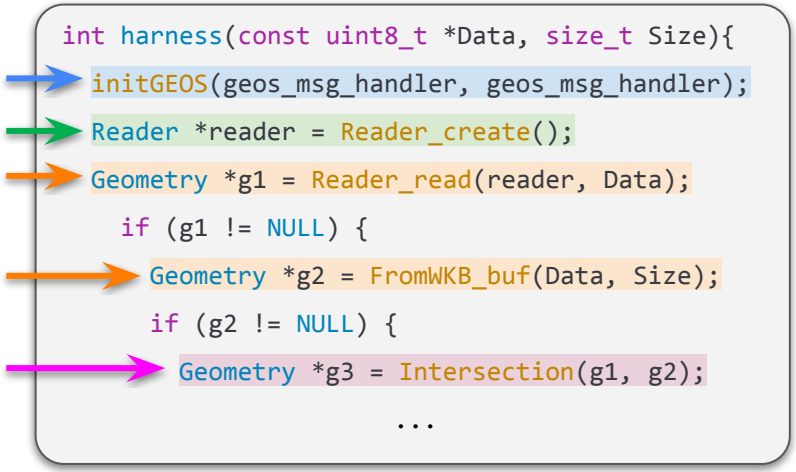
- **Idea:** mutationally build-up harnesses, validating each step via our **oracles**
  - Culling **invalid** or **uninteresting** API usage gradually converges on **valid harnesses**

Our Solution: Oracle-guided Harnessing

- **Idea:** mutationally build-up harnesses, validating each step via our **oracles**
  - Culling **invalid** or **uninteresting** API usage gradually converges on **valid harnesses**

- **Insight:** all functions serve a purpose:
  - Surveyed OSS-Fuzz's **281** C libraries:
    - Library **initialization** routines
    - Data **entrypoints**, **processors**
    - Data-resolving **auxiliary calls**

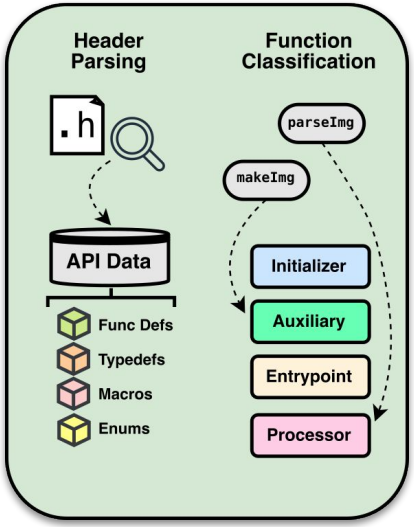
- **Approach:** build harnesses that follow these common-case API patterns!



Our Solution: Oracle-guided Harnessing

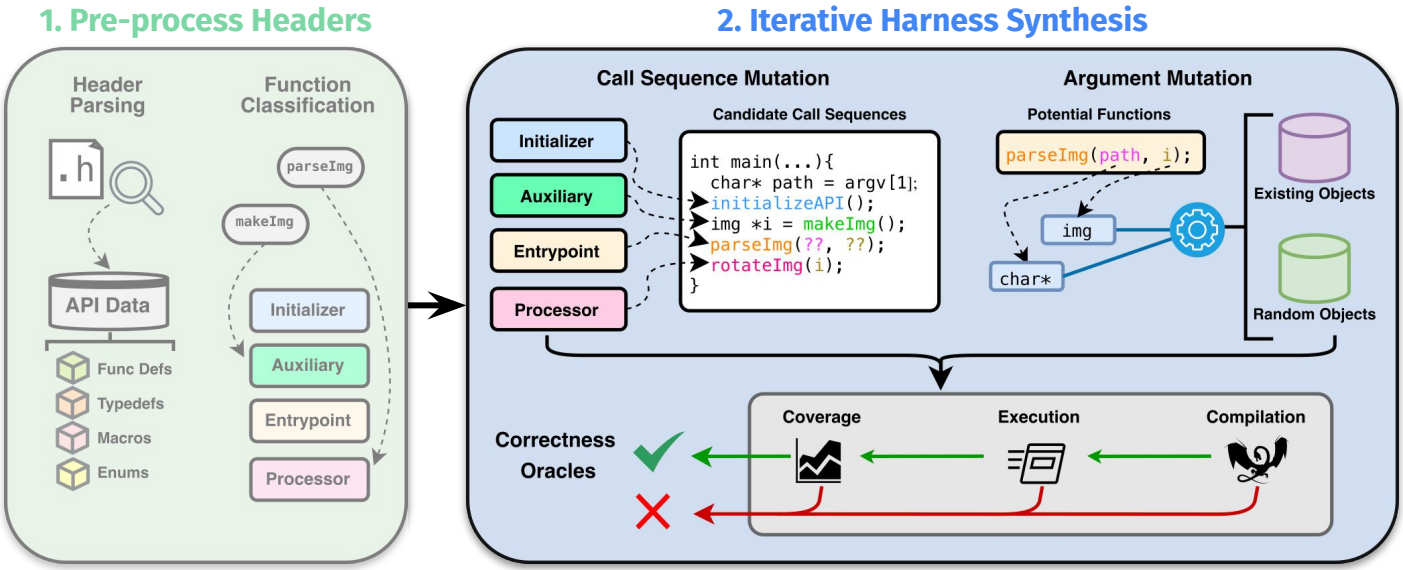
- **Idea:** mutationally build-up harnesses, validating each step via our **oracles**
  - Culling **invalid** or **uninteresting** API usage gradually converges on **valid harnesses**

1. Pre-process Headers



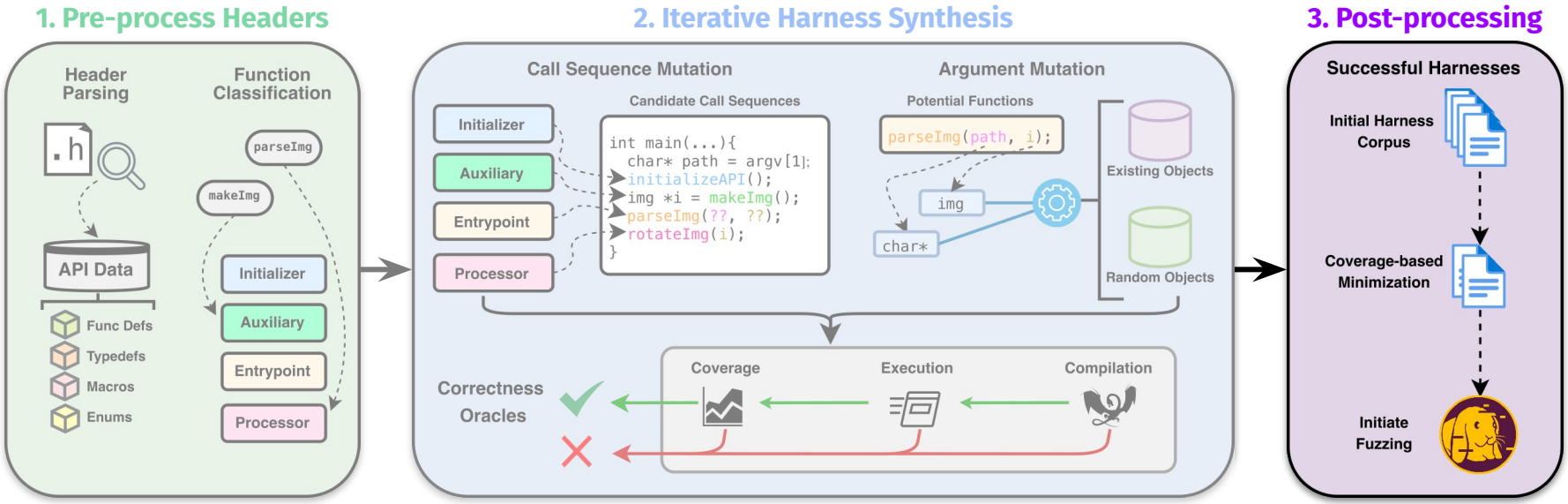
Our Solution: Oracle-guided Harnessing

- **Idea:** mutationally build-up harnesses, validating each step via our **oracles**
  - Culling **invalid** or **uninteresting** API usage gradually converges on **valid harnesses**



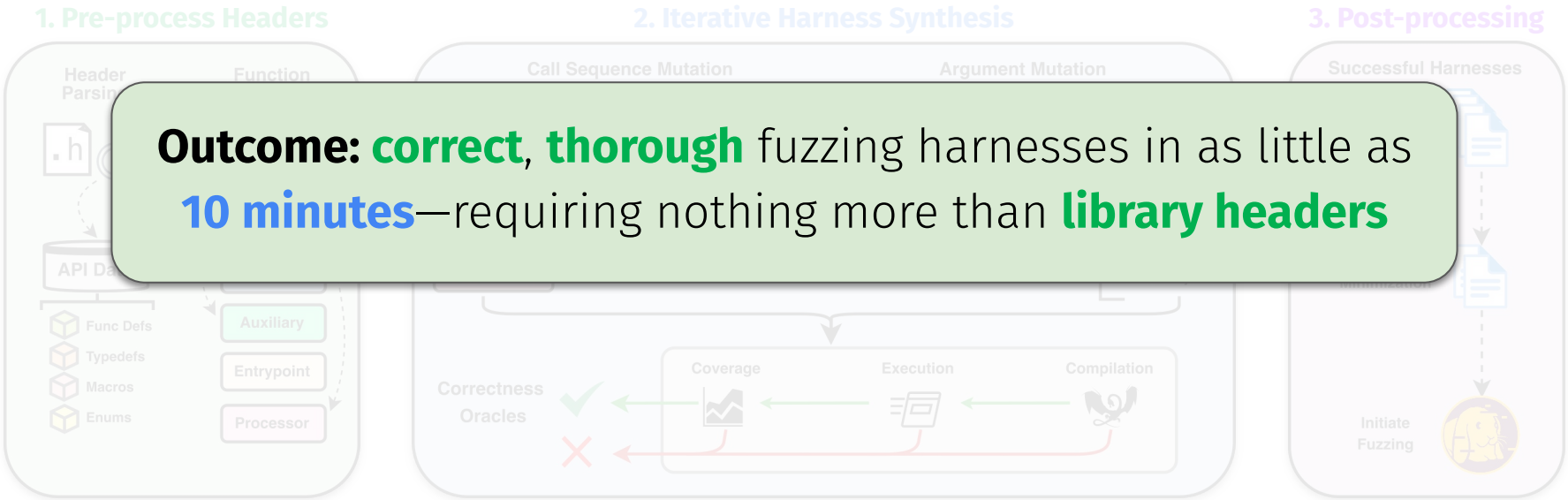
Our Solution: Oracle-guided Harnessing

- **Idea:** mutationally build-up harnesses, validating each step via our **oracles**
  - Culling **invalid** or **uninteresting** API usage gradually converges on **valid harnesses**



Our Solution: Oracle-guided Harnessing

- **Idea:** mutationally build-up harnesses, validating each step via our **oracles**
  - Culling **invalid** or **uninteresting** API usage gradually converges on **valid harnesses**



Outcome: correct, **thorough** fuzzing harnesses in as little as **10 minutes**—requiring nothing more than **library headers**

Evaluation: Overview

- Prototype: **OGHarn**
  - Supports harnessing for C libraries
- Evaluated on **20 real-world APIs**
  - **16** with prior harnesses, **4** without
  - Coverage, correctness bug-finding
  - All campaigns run for 5x24-hr trials
- Competitors:
  - Hopper (current state-of-the-art)
  - Existing OSS-Fuzz harnesses (**33** total)

Libraries **with** existing harnesses:	C-Ares, cglTF, cJSON, GPAC, HDF5, LCMS, Lexbor, libGEOS, libICAL, libMagic, libPCAP, libUCL, OpenEXR, PCRE2, SQLite3, Zlib
Libraries **without** prior harnesses:	Faup, libFYAML, RayLib, StormLib

Evaluation: Correctness

- Measured **false-positive bugs** found by all competitors:
  - Initial evaluation on OGHarn's **top-10 highest-coverage** harnesses

Manual Harnesses
11 true,
0 false

Evaluation: Correctness

- Measured **false-positive bugs** found by all competitors:
  - Initial evaluation on OGHarn's **top-10 highest-coverage** harnesses

Manual Harnesses	Hopper
11 true, **0** false	**10** true, **54** false

libGEOS devs:
"This is a **usage error.**"

StormLib devs:
"**Wrong usage** of input and output parameters."

libUCL devs:
"Clearly **misuse** of API."

Evaluation: Correctness

- Measured **false-positive bugs** found by all competitors:
  - Initial evaluation on OGHarn's **top-10 highest-coverage** harnesses
  - Scaled-up to include **all harnesses**; resulted in one **new bug found**

Manual Harnesses	Hopper	Oracle-guided Harnessing
11 true, **0** false	**10** true, **54** false	**41** true, **0** false

- **Takeaways:** Oracle-guided Harnessing correctly **upholds API semantics**

Evaluation: Code Coverage

- Measured OGHarn's relative **total and unique code coverage**
  - Median **653% more functions harnessed** over developer-written harnesses

Versus Manual Harnesses
+14% total,
+364% unique

Evaluation: Code Coverage

- Measured OGHarn's relative **total and unique code coverage**
  - Median **653% more functions harnessed** over developer-written harnesses

Versus Manual Harnesses	Versus Hopper
+14% total, **+364%** unique	**-31%** total, **-72%** unique

- **Takeaways:** Hopper covers more, but at steep expense of **correctness**

Evaluation: New Bugs Found

- Quantified all competitors' **total and uniquely-found bugs**
  - Following our own triage, we reported newly-found bugs to developers

Manual Harnesses	Hopper
11 total, **4** unique	**10** total, **8** unique

Evaluation: New Bugs Found

- Quantified all competitors' **total and uniquely-found bugs**
  - Following our own triage, we reported newly-found bugs to developers

Manual Harnesses	Hopper	Oracle-guided Harnessing
11 total, **4** unique	**10** total, **8** unique	**41** total, **32** unique

- **Takeaways:** Oracle-guided Harnessing **expands fuzzing bug discovery**
  - All **41 are confirmed**, with **40 patched** post-reporting

Conclusion: Why Oracle-guided Harnessing?

- Prior harnessing approaches **restricted** by need for **reference code** or **specifications**
- Recent workarounds fail to preserve **valid API usage**, causing **false-positive crashes**

Conclusion: Why Oracle-guided Harnessing?

- Prior harnessing approaches **restricted** by need for **reference code** or **specifications**
- Recent workarounds fail to preserve **valid API usage**, causing **false-positive crashes**
- Our solution: **Oracle-guided Harnessing**
  - Mutational harness creation **using only headers**
  - Structure generation by **common-case patterns**
  - Leverage oracles—**compilation, execution, and code coverage**—to validate harness mutations
  - **Outcome:** valid, thorough API fuzzing harnesses

Key Results:

41 new bugs,

0 false bugs,

14% coverage over OSS-Fuzz

Thank you!

Try Oracle-guided Harnessing at:
github.com/FuturesLab/OGHarn

Contact: gabe.sherman@utah.edu
gabe-sherman.github.io
futures.cs.utah.edu

FUTURES^3
LAB FUTURE TECHNOLGY FOR USABLE, RELIABLE, & EFFICIENT SECURITY OF SOFTWARE & SYSTEMS