

# CollabFuzz: A Framework for Collaborative Fuzzing

Sebastian Österlund

Elia Geretto

Andrea Jemmett

Emre Güler

Philipp Görz

Thorsten Holz

Cristiano Giuffrida

Herbert Bos

*Vrije Universiteit Amsterdam*

*Ruhr-Universität Bochum*





# What's all the fuzz about?

**Answer:** run them *together*, orchestrated by CollabFuzz

- **Problem:** Many **different** fuzzers to choose from! Many scripts to write!

```
----- [ HONGGFUZZ / v1.3 ] -----
Iterations : 2354 [2.35k]
Phase : Dynamic Main (2/2)
INFO: Seed: 38870025
#0      READ  units: 201 exec/s: 0
#201    INITED cov: 2496 indir: 22 units: 148 exec/s: 0
#1891   NEW   cov: 2497 indir: 22 units: 149 exec/s: 630 L: 85 MS: 5 ChangeASC
Over- DE: "io"-"_L

american fuzzy lop 0.47b (readpng)

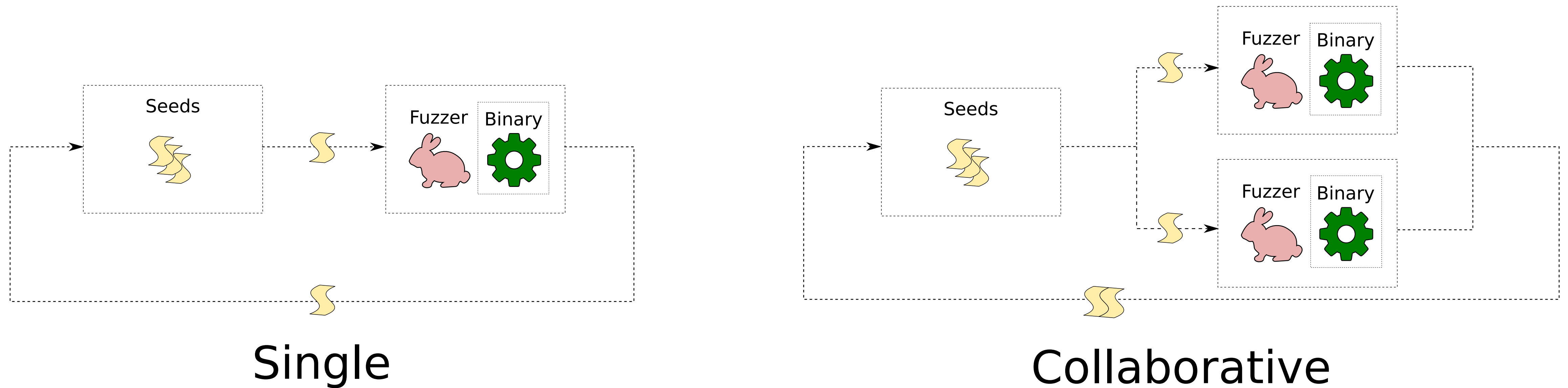
process timing
run time : 0 days, 0 hrs, 4 min, 43 sec
last new path : 0 days, 0 hrs, 0 min, 26 sec
last uniq crash : none seen yet
last uniq hang : 0 days, 0 hrs, 1 min, 51 sec
cycle progress
now processing : 38 (19.49%)
paths timed out : 0 (0.00%)
stage progress
now trying : interest 32/8
stage execs : 0/9990 (0.00%)
total execs : 654k
exec speed : 2306/sec
fuzzing strategy yields
bit flips : 88/14.4k, 6/14.4k, 6/14.4k
byte flips : 0/1804, 0/1786, 1/1750
arithmetics : 31/126k, 3/45.6k, 1/17.8k
known ints : 1/15.8k, 4/65.8k, 6/78.2k
havoc : 34/254k, 0/0
trim : 2876 B/931 (61.45% gain)

overall results
cycles done : 0
total paths : 195
uniq crashes : 0
uniq hangs : 1
map coverage
map density : 1217 (7.43%)
count coverage : 2.55 bits/tuple
findings in depth
favored paths : 128 (65.64%)
new edges on : 85 (43.59%)
total crashes : 0 (0 unique)
total hangs : 1 (1 unique)
path geometry
levels : 3
pending : 178
pend fav : 114
imported : 0
variable : 0
latent : 0

22 MS: 2 AddFromTe
297 MS: 2 ChangeBy
cker-compose run l
-exact_artifact_pa
=1 -max_total_time
```



# Collaborative fuzzing




*EnFuzz: Ensemble Fuzzing with Seed Synchronization among Diverse Fuzzers  
(USENIX Sec '19)*



# CollabFuzz Framework

- Allows for a **central manager** to orchestrate many fuzzers
- The managed fuzzers can be **different** fuzzers
- Supports: AFL, AFLFast, AFL++, FairFuzz, Honggfuzz, LibFuzzer, QSYM, Radamsa
- A framework to collect results and perform **analysis** during a collaborative fuzzing campaign



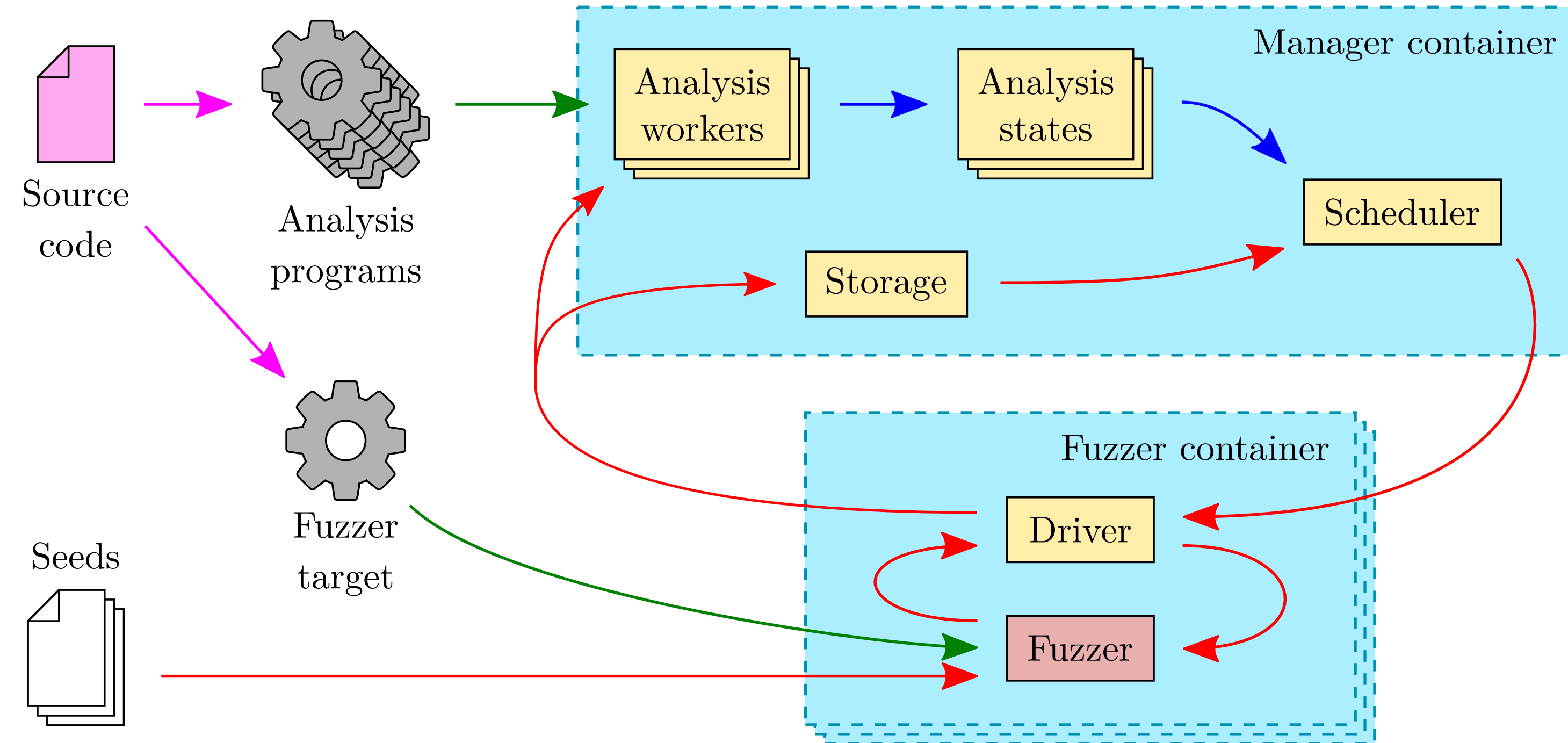
```
[ HONGGFUZZ / v1.3 ]
-----
Iterations : 2354 [2.35k]
Phase : Dynamic Main (2/2)
INFO: Seed: 38870025
#0      READ   units: 201 exec/s: 0
#201    INITED cov: 2496 indir: 22 units: 148 exec/s: 0
#1891   NEW    cov: 2497 indir: 22 units: 149 exec/s: 630 L: 85 MS: 5 ChangeASC
Over- DE: "io"- "_L

american fuzzy lop 0.47b (readpng)

process timing
  run time : 0 days, 0 hrs, 4 min, 43 sec
  last new path : 0 days, 0 hrs, 0 min, 26 sec
  last uniq crash : none seen yet
  last uniq hang : 0 days, 0 hrs, 1 min, 51 sec
cycle progress
  now processing : 38 (19.49%)
  paths timed out : 0 (0.00%)
stage progress
  now trying : interest 32/8
  stage execs : 0/9990 (0.00%)
  total execs : 654k
  exec speed : 2306/sec
fuzzing strategy yields
  bit flips : 88/14.4k, 6/14.4k, 6/14.4k
  byte flips : 0/1804, 0/1786, 1/1750
  arithmetics : 31/126k, 3/45.6k, 1/17.8k
  known ints : 1/15.8k, 4/65.8k, 6/78.2k
  havoc : 34/254k, 0/0
  trim : 2876 B/931 (61.45% gain)
overall results
  cycles done : 0
  total paths : 195
  uniq crashes : 0
  uniq hangs : 1
map coverage
  map density : 1217 (7.43%)
  count coverage : 2.55 bits/tuple
findings in depth
  favored paths : 128 (65.64%)
  new edges on : 85 (43.59%)
  total crashes : 0 (0 unique)
  total hangs : 1 (1 unique)
path geometry
  levels : 3
  pending : 178
  pend fav : 114
  imported : 0
  variable : 0
  latent : 0

root@kali:~/radamsa# radamsa --help
Usage: radamsa [arguments] [file ...]
-h --help, show this thing
-a --about, what is this thing?
-v --version, show program version
-o --output <arg>, output pattern, e.g. out.bin/tmp/fuzz-%n.Xs, -, :80 or 127.0.0.1:80 or 127.0.0.1:123/udp [-]
-n --count <arg>, how many outputs to generate (number or inf) [1]
-s --seed <arg>, random seed (number, default random)
-m --mutations <arg>, which mutations to use [ft=2,fo=2,fn,num=5,td,tr2,ts1,tr,ts2,ld,lts,lr2,li,ls,lp,lr,lis,lrs,sr,sd,bd,bf,bi,br,bp,bei,bed,ber,uw,ui=2,xp=9,ab]
-p --patterns <arg>, which mutation patterns to use [od,nd=2,bu]
-g --generators <arg>, which data generators to use [random,file=1000,jump=200,stdin=100000]
-M --meta <arg>, save metadata about generated files to this file
-r --recursive, include files in subdirectories
-S --seek <arg>, start from given testcase
-T --truncate <arg>, take only first n bytes of each output (mainly intended for UDP)
-d --delay <arg>, sleep for n milliseconds between outputs
-l --list, list mutations, patterns and generators
-c --checksums <arg>, maximum number of checksums in uniqueness filter (0 disables) [10000]
-H --hash <arg>, hash algorithm for uniqueness checks (stream or sha256) [stream]
-v --verbose, show progress during generation
root@kali:~/radamsa#
```

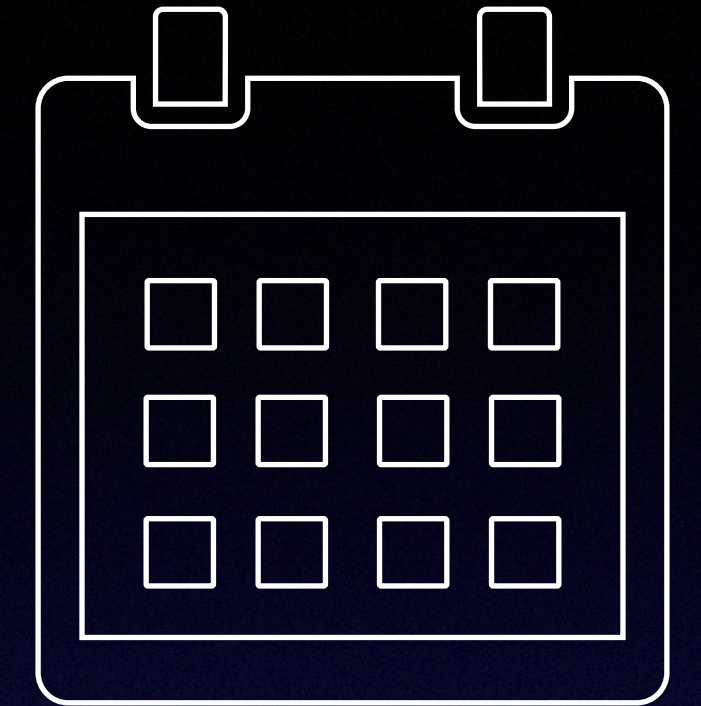




# CollabFuzz Framework



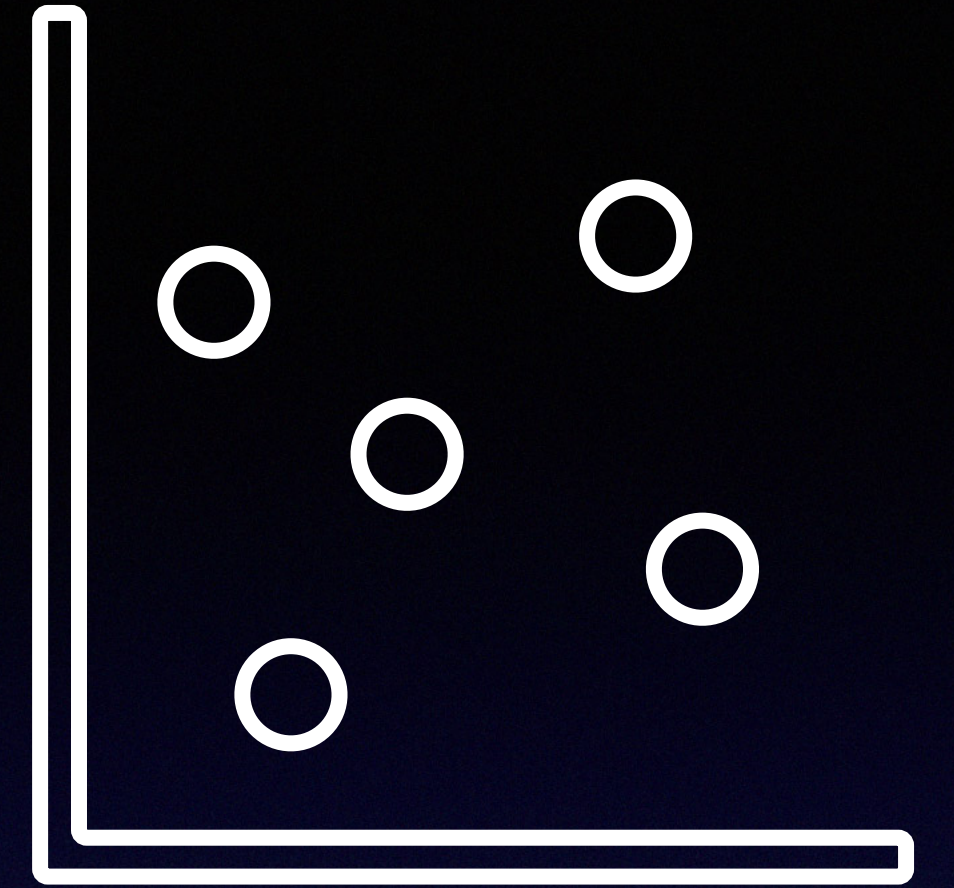
# Scheduling policies



- **Broadcast/ EnFuzz**: send all test cases to all fuzzers
- **(Cost-) Benefit**: use a “benefit” heuristics to select what seeds to send out
- In essence two possible “planes” to schedule on
  - *Temporally*: **when** to send out test case
  - *Spatial*: **which** fuzzers should get the test case



# Scheduling results

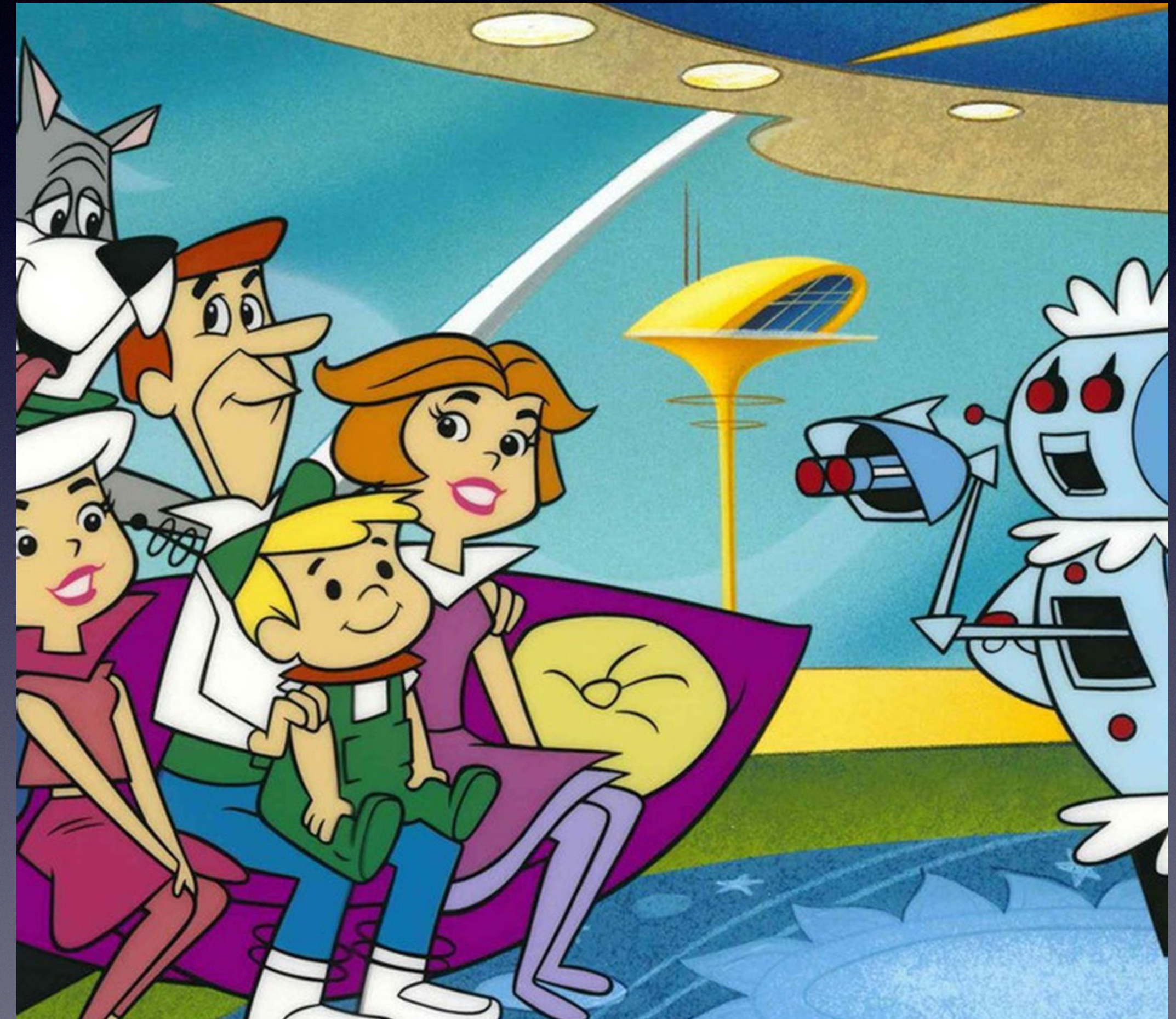


- We could not observe any **statistically significant improvements**
- Reasons:
  - No actual control over what the fuzzers work on (**interface**)
  - We might not use the right **features**



# Future ideas in scheduling

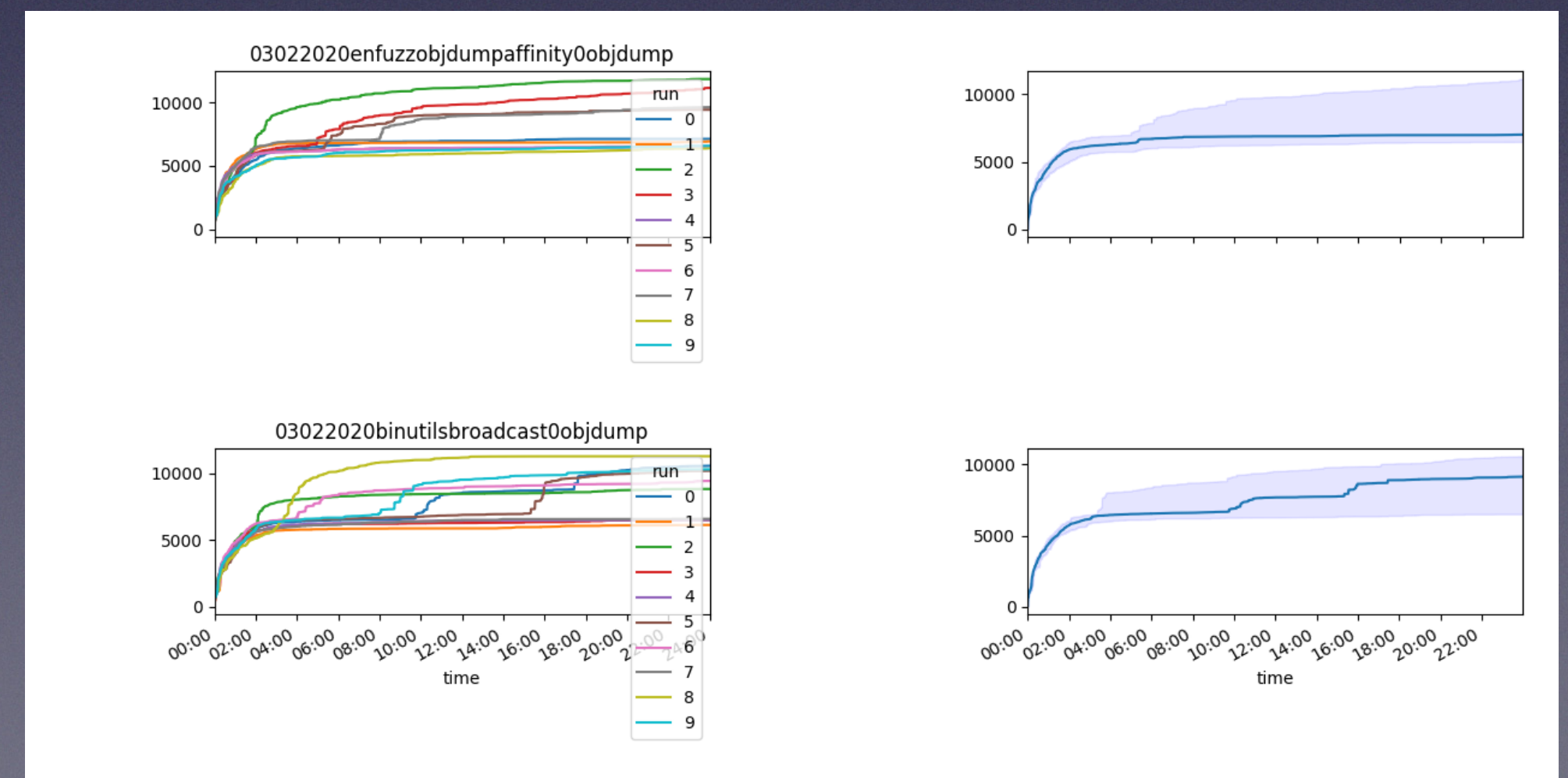
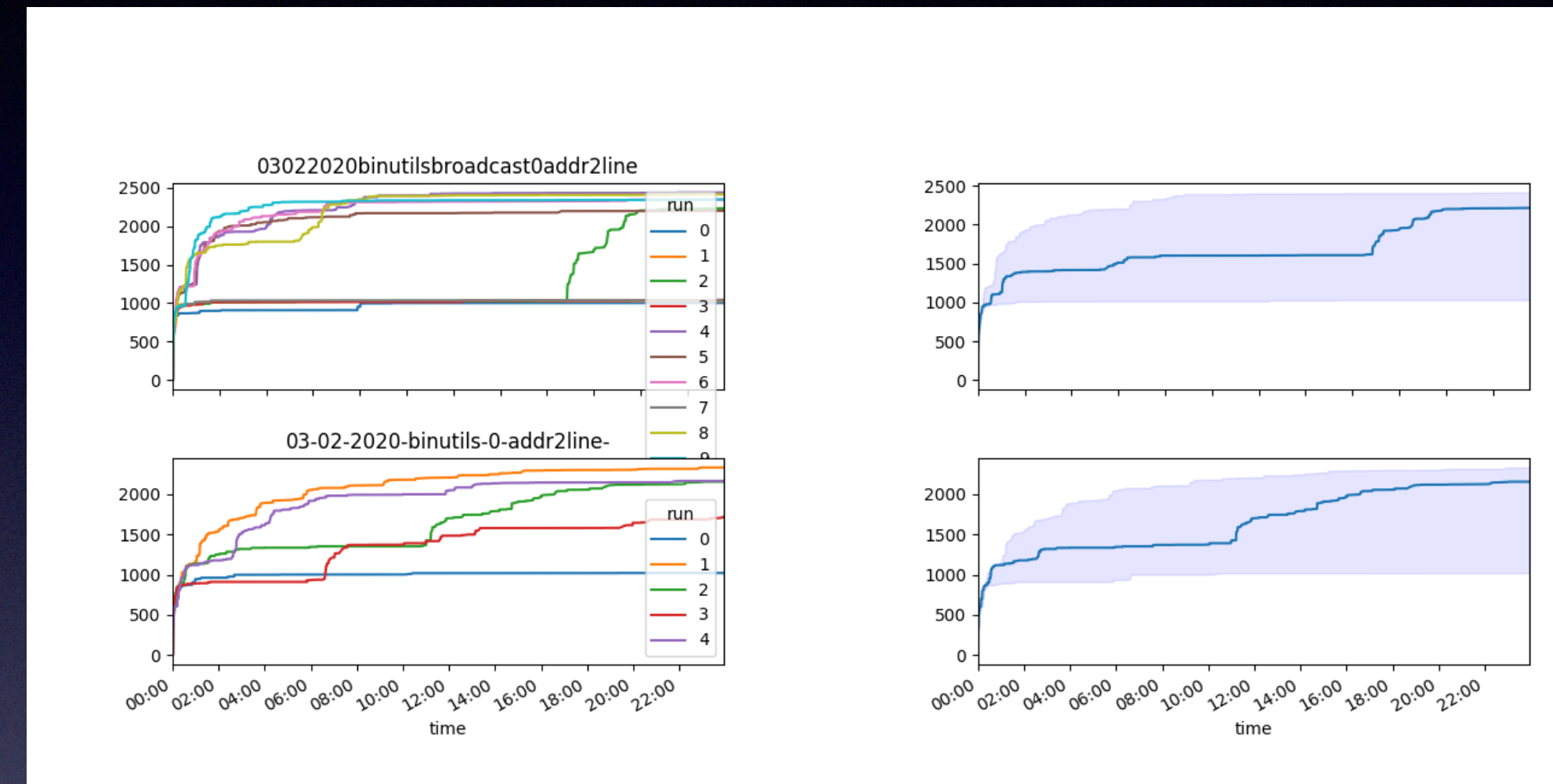
- Scheduling at a different level
  - Test case vs. resource scheduling
- More fine-grained control
  - Branch-level scheduling





# Using CollabFuzz for Evaluations

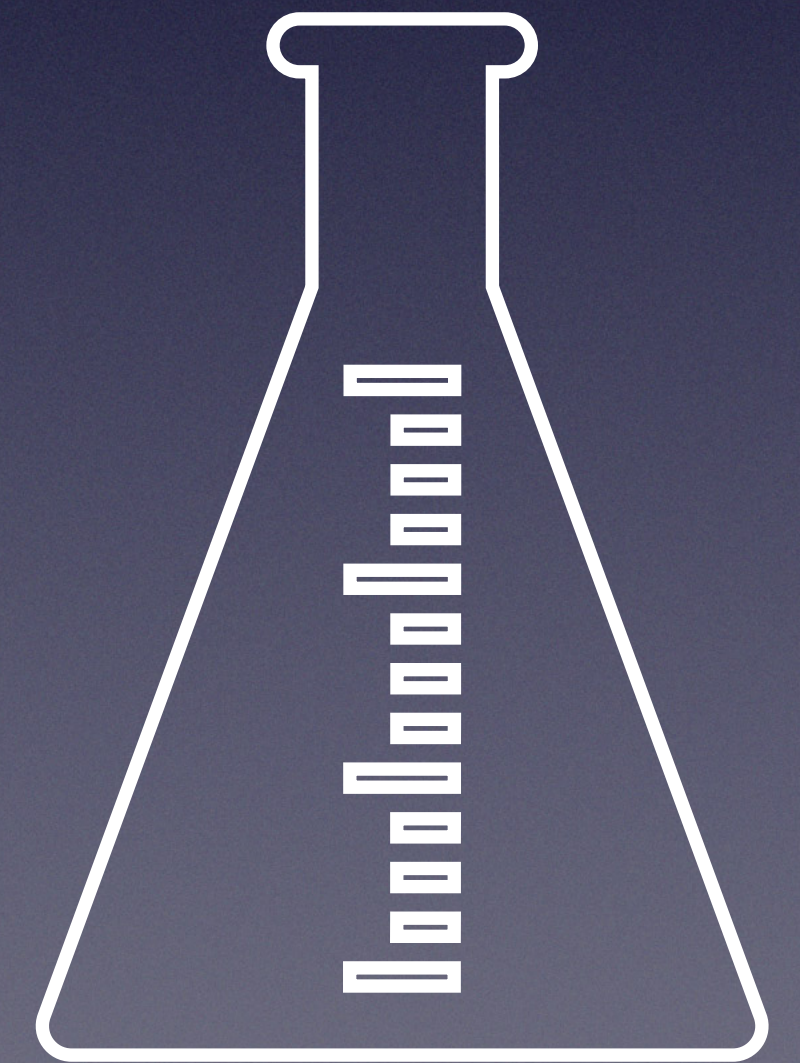
- **Repeatability** of experiments
- Large-scale experiments on a cluster
- Real-time analysis of campaign





# Analysis

- **Analysis passes** can run when a new test case is discovered by a fuzzer
- E.g., get the coverage of the test case
- Allows pipelining (i.e., do taint track analysis if there is new coverage)
- Quite flexible





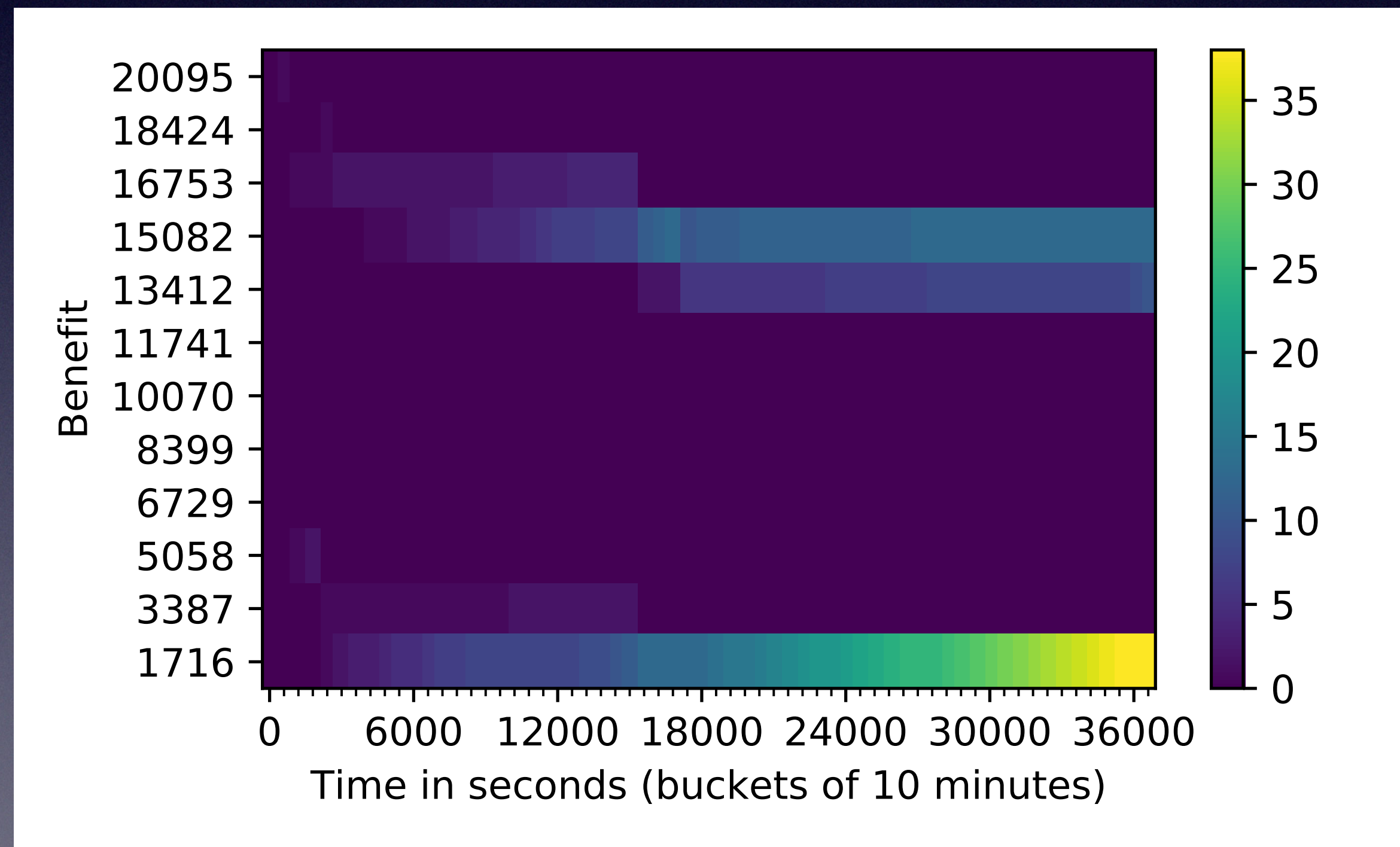
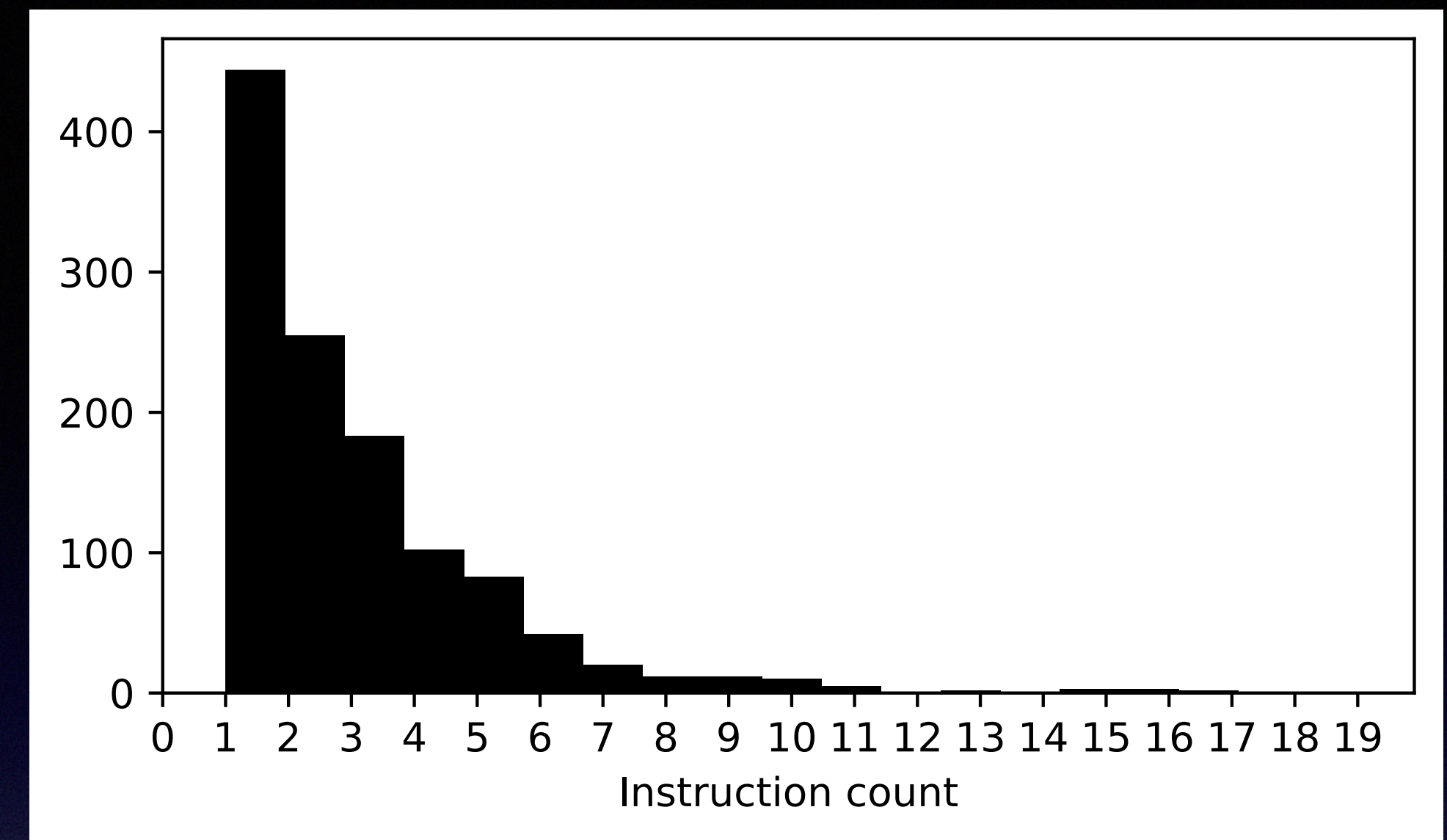
# Analysis Passes

- Coverage
- Taint information
- Tainted instruction count pass
- Branch analysis
- Flexible: global aggregate data, per-fuzzer data, per test-case data

```
.
├── analyses
│   ├── benefit_pass.bak
│   ├── coverage_utils.rs
│   ├── fuzzer_coverage_analysis.rs
│   ├── generation_graph_analysis.rs
│   ├── global_coverage_analysis.rs
│   ├── instruction_count_analysis.rs
│   ├── mod.rs
│   ├── observed_conditions_analysis.rs
│   ├── test_analysis.rs
│   └── test_case_benefit_analysis.rs
├── analysis_interface.rs
├── config.rs
├── mod.rs
├── passes
│   ├── bb_taint_tracer_pass.rs
│   ├── generic_instr_pass.rs
│   ├── instruction_counter_pass.rs
│   ├── mod.rs
│   └── test_pass.rs
├── pass_interface.rs
└── utils.rs
```

2 directories, 20 files







# Technical Info

- Each fuzzer needs a “**driver**”
  - Program to monitor the fuzzer (we have a generic one for AFL-like fuzzers)
- Drivers communicate with the framework over ZeroMQ
  - Allows large-scale **distributed** fuzzing
- Framework: **Rust**, driver: Python



# Configuration

- Sets up experiments using YAML files
- Easy to queue long-running experiments
- All fuzzer-targets are contained within Docker containers
- It's even possible to run experiments on a cluster using Docker Swarm/ Kubernetes

```
---
targets:
  -
    binary: guetzli
    name: guetzli
    input: ./inputs/guetzli
  -
    binary: json
    name: json
    input: ./inputs/json
experiment:
  name: "my-cool-experiment"
  timeout: 36000
  repeat: 10
  use_collab: true
  enable_afl_affinity: true
  scheduler: "test_case_benefit"
fuzzers:
  -
    name: qsym
    type: qsym
    parallel: 1
  -
    name: libfuzzer
    type: libfuzzer
    parallel: 1
  -
    name: honggfuzz
    type: honggfuzz
    parallel: 2
```

Targets

Parameters

Fuzzer config

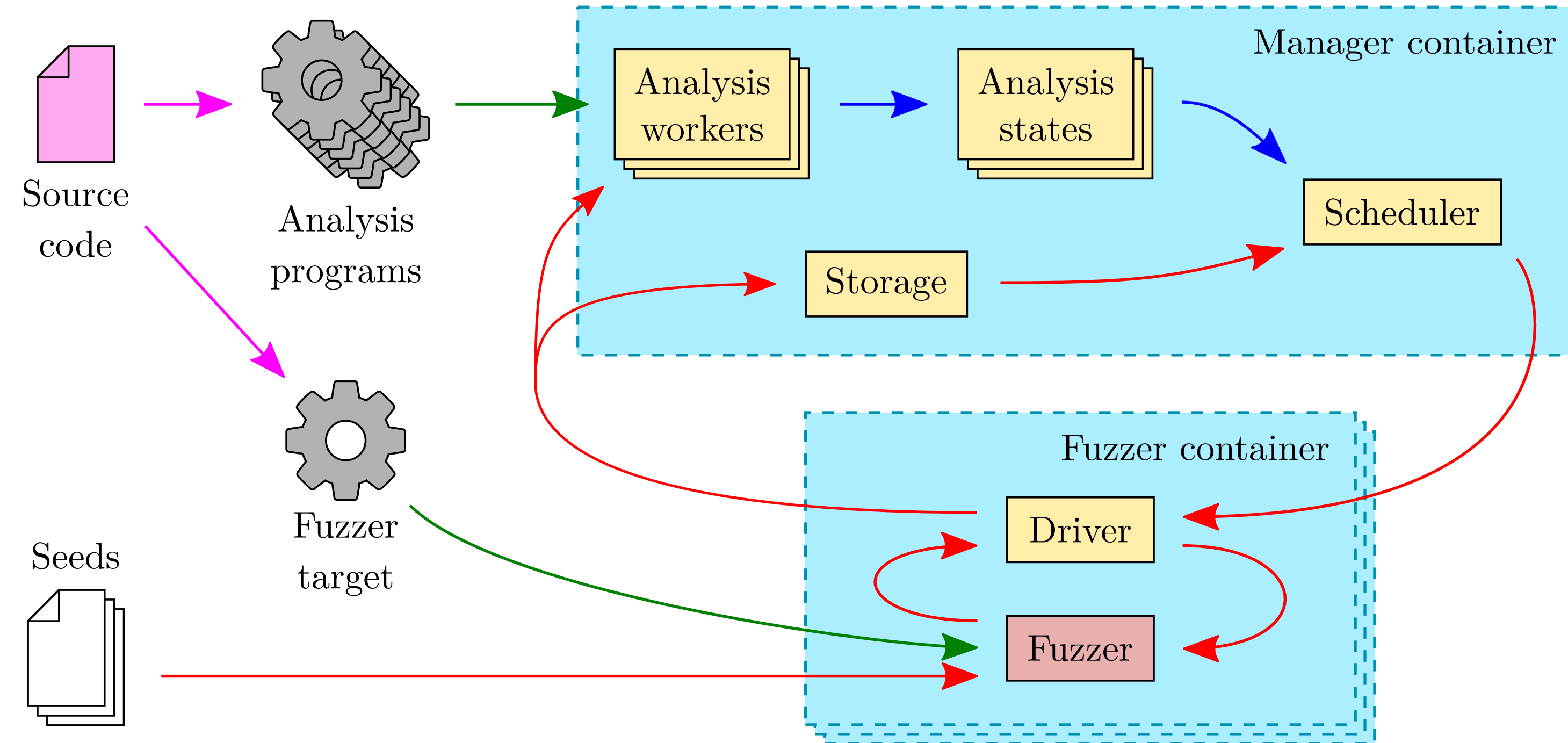


# Logging

- The framework dumps all data in a sqlite database
- E.g., all the analysis pass results, scheduling decisions, fuzzer events

```
sqlite> .schema
CREATE TABLE fuzzer_types (
    id          INTEGER PRIMARY KEY,
    description TEXT
);
CREATE TABLE test_case_types (
    id          INTEGER PRIMARY KEY,
    description TEXT
);
CREATE TABLE fuzzer_event_types (
    id          INTEGER PRIMARY KEY,
    description TEXT
);
CREATE TABLE analysis_types (
    id          INTEGER PRIMARY KEY,
    description TEXT
);
CREATE TABLE fuzzers (
    fuzzer_id    INTEGER PRIMARY KEY,
    fuzzer_type_id INTEGER REFERENCES fuzzer_types
);
CREATE TABLE test_cases (
    hash          TEXT PRIMARY KEY,
    test_case_type_id INTEGER REFERENCES test_case_types,
    discovery_fuzzer INTEGER REFERENCES fuzzers,
    discovery_time INTEGER
);
CREATE TABLE dispatch (
    test_case_hash TEXT REFERENCES test_cases,
    fuzzer_id      INTEGER REFERENCES fuzzers,
    dispatch_time  INTEGER
);
CREATE TABLE fuzzer_events (
    fuzzer_id    INTEGER REFERENCES fuzzers,
    event_type_id INTEGER REFERENCES fuzzer_event_types,
    event_time   INTEGER
);
CREATE TABLE analysis_states (
    test_case_hash INTEGER REFERENCES test_cases,
    analysis_id    INTEGER REFERENCES analysis_types,
    analysis_dump  BLOB,
    PRIMARY KEY(test_case_hash, analysis_id)
);
```





# CollabFuzz Framework



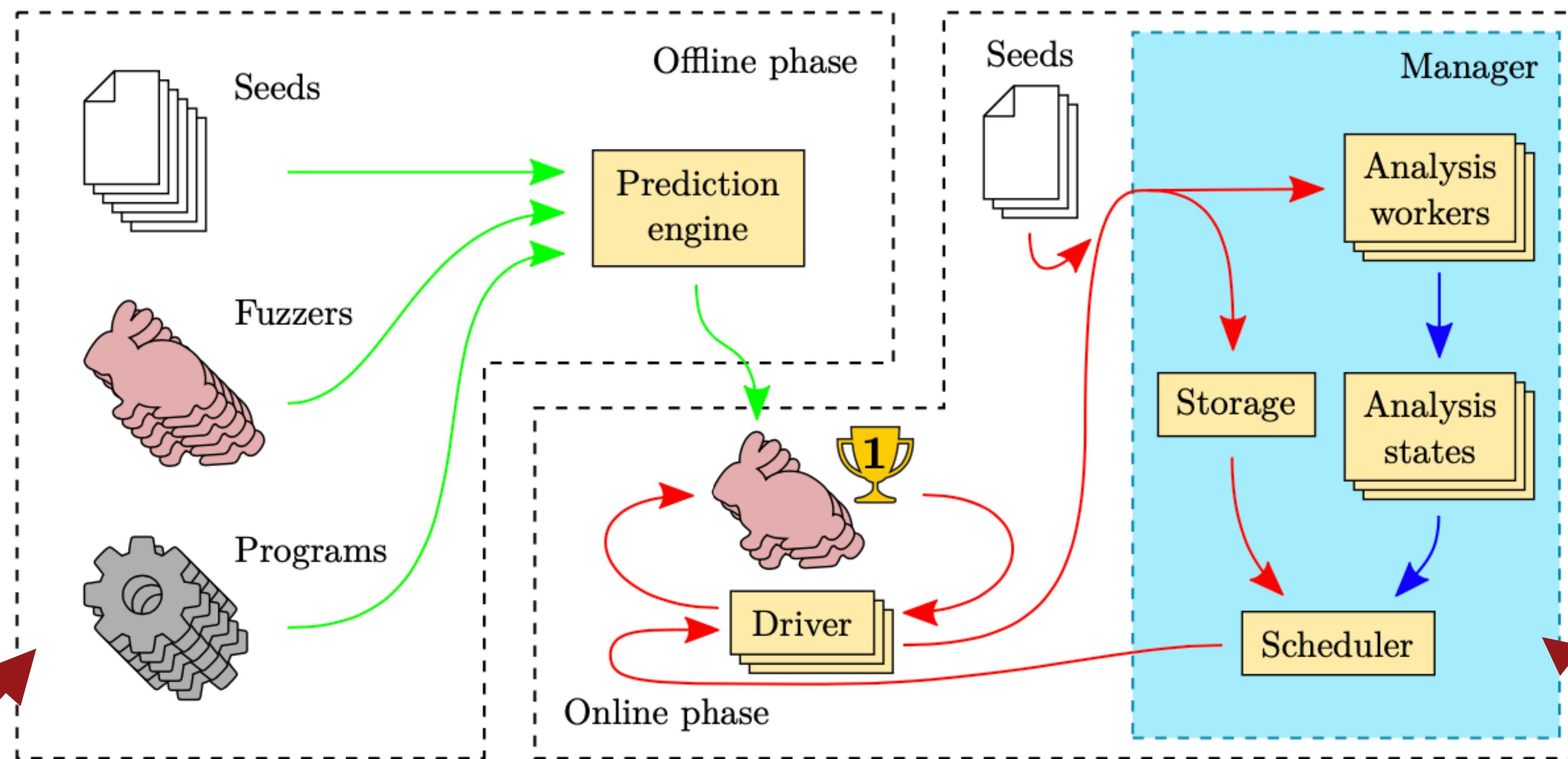


Fig. 2: An overview of COLLABFUZZ and its different components.

CUPID (ACSAC '20)

CollabFuzz (EuroSec '21)



# DEMO



# Conclusion

- CollabFuzz: orchestrate collaborative fuzzing campaigns, also in a distributed setting
- Allows expression of scheduling policies
- Enables real-time analysis of large-scale campaigns
- Available at: [github.com/vusec/collabfuzz](https://github.com/vusec/collabfuzz)





# CollabFuzz: A Framework for Collaborative Fuzzing

Sebastian Österlund

Elia Geretto

Andrea Jemmett

Emre Güler

Philipp Görz

Thorsten Holz

Cristiano Giuffrida

Herbert Bos

*Vrije Universiteit Amsterdam*

*Ruhr-Universität Bochum*

