# Barking up the Wrong Tree: Correctness & Debugging

Chengcheng Wan, Lefan Zhang
{cwan, lefanz} @uchicago.edu
University of Chicago

USENIX ATC '21

# What is correctness?

## Correctness

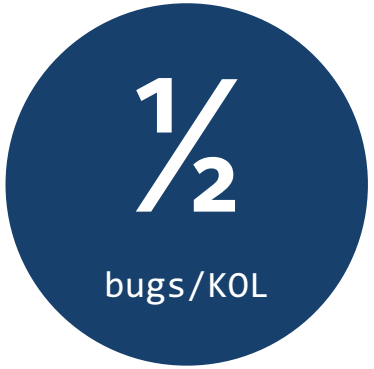
Hard to define without a comprehensive specification

## Incorrectness

- Incorrect outputs
- Software crashes
- Resource leaks
- Memory corruptions
- …

# Bugs are widespread


@ in house




@ in field


@ Linux Kernel


@ Apache

Reference:
[1] McConnell, Steve. Code complete. Pearson Education, 2004.
[2] https://arstechnica.com/information-technology/2015/02/linux-has-2000-new-developers-and-gets-10000-patches-for-each-version/

# Bugs are costly.

## Service outage

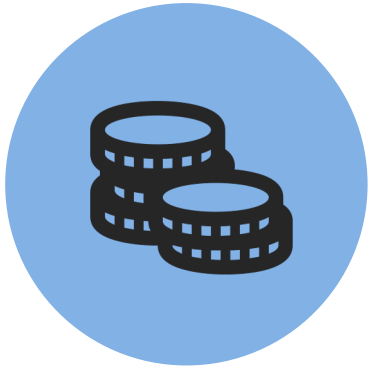
- Service outages of Facebook (2019), Google (2020), etc.
- 3-hour Nasdaq trading halt due to failure of backup system (2013)

## Security Vulnerability

- BlueKeep: potential remote code execution caused by bug in RDP (2019)
- Heartbleed: buffer over-read caused by OpenSSL bug (2014)

## Medical crisis

- Radiation treatment overdose caused by software bugs (1985-1987)



Loss of money/ information

Deaths of people


Reference:

[1] https://www.bbc.com/news/technology-47562281

[2] https://en.wikipedia.org/wiki/2020_Google_services_outages

[3] Zhivich, Michael, and Robert K. Cunningham. "The real cost of software errors." IEEE Security & Privacy 7.2 (2009): 87-90.

[4] https://en.wikipedia.org/wiki/BlueKeep

# Tackling Bugs in Different Stages


- ☐ Static checking
- ☐ Formal Verification
- ☐ …

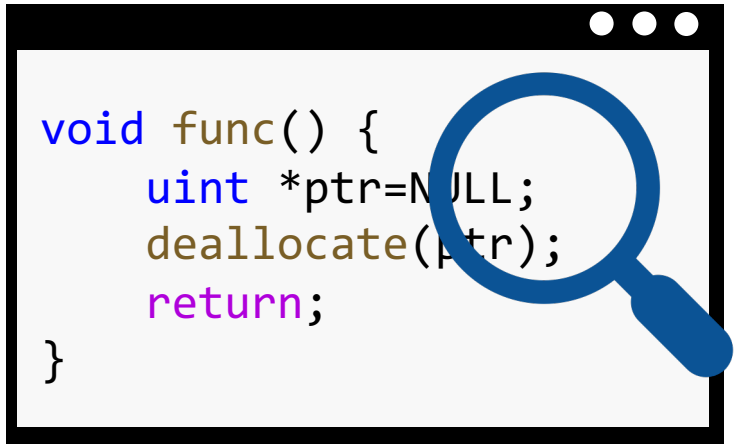
- ☐ Fuzz testing
- ☐ …

- ☐ Logging
- ☐ Dynamic checking
- ☐ Failure prevention
- ☐ …

- ☐ Failure diagnosis
- ☐ Patching
- ☐ …

# Static Checking

How does human inspect programs?




Looking for anti-patterns by code review


Looking for anti-patterns by code analysis

# Static Checking

- Analyze the code following bug patterns
  - Dereferencing null pointers (Engler, SOSP 2001)
  - Inconsistent copy-paste renaming (Li, OSDI 2004)
  - Related variables protected by different locks (Lu, SOSP 2007)

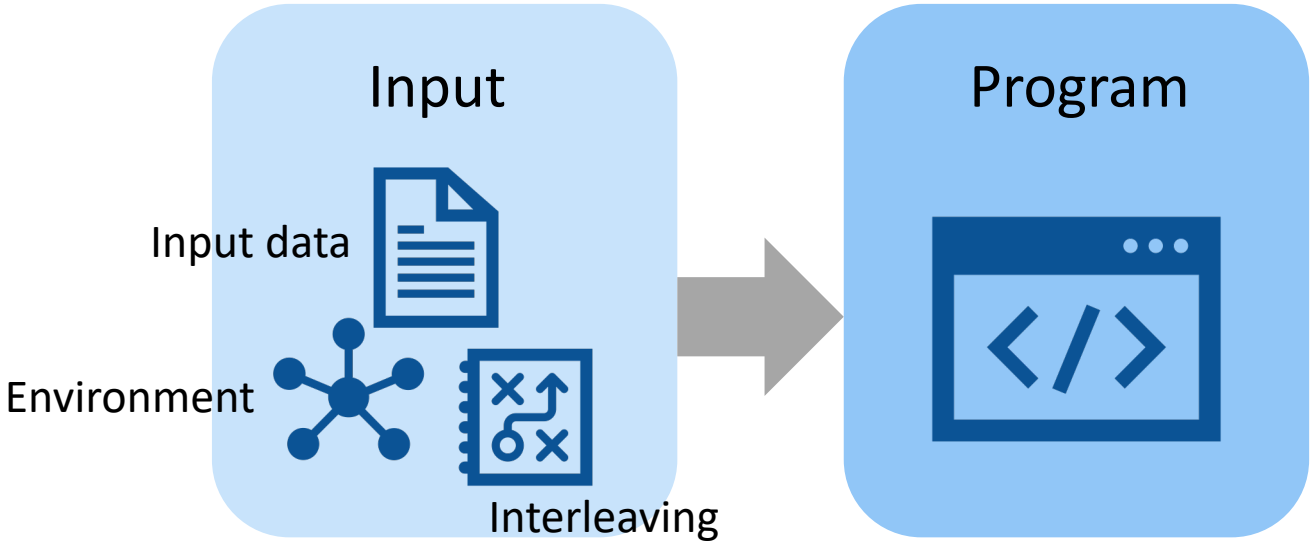
## Strength

- Good scalability
- Easy fault localization

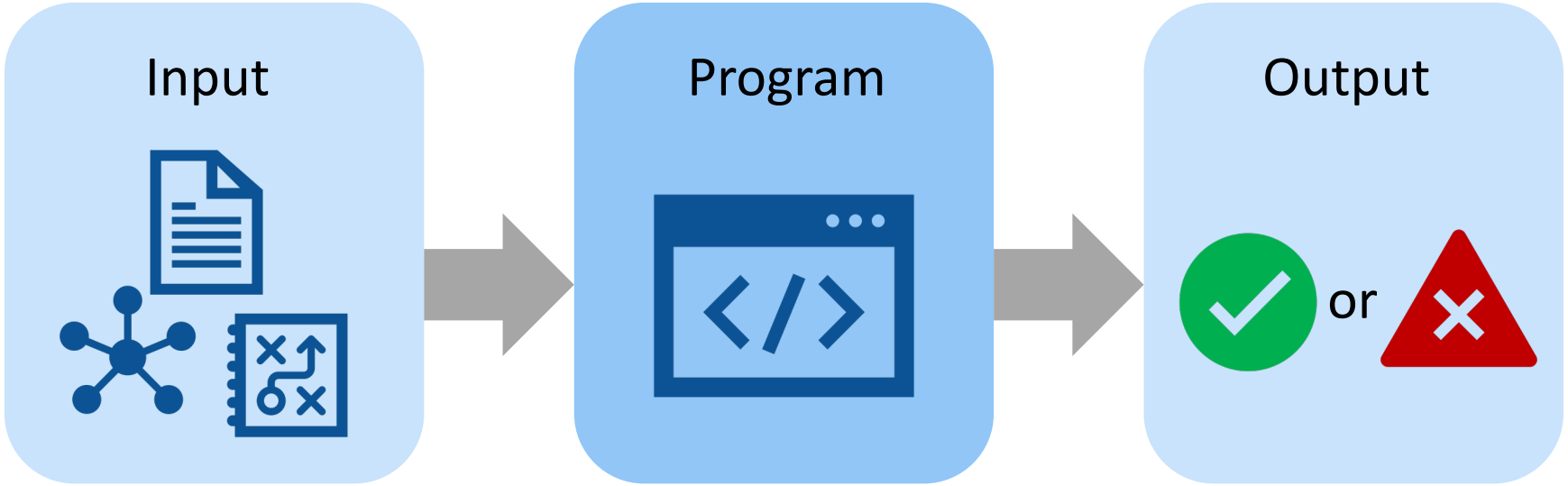
## Weakness

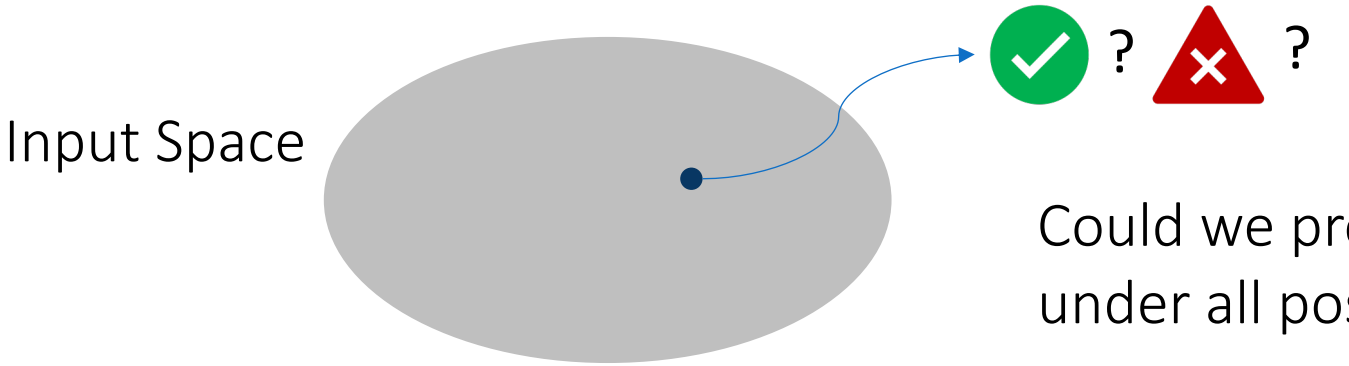
- Limited to specific bug patterns (False negatives)
- Lack of runtime information (False positives)

# Formal Verification



# Formal Verification




Could we prove the program is correct under all possible inputs?

# Formal Verification: model checking

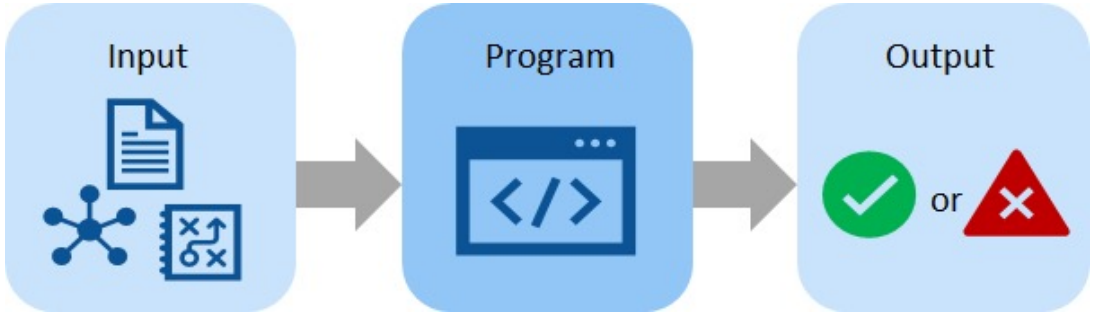
- Explicitly explore all possible states of the program
  - Example: CMC (Musuvathi, SOSP 2002)
- Symbolically executed the program + SMT/SAT solver
  - Example: CBMC (Kroening)

## Weakness

- Exponential state-explosion (poor scalability)

# Formal Verification: theorem proving

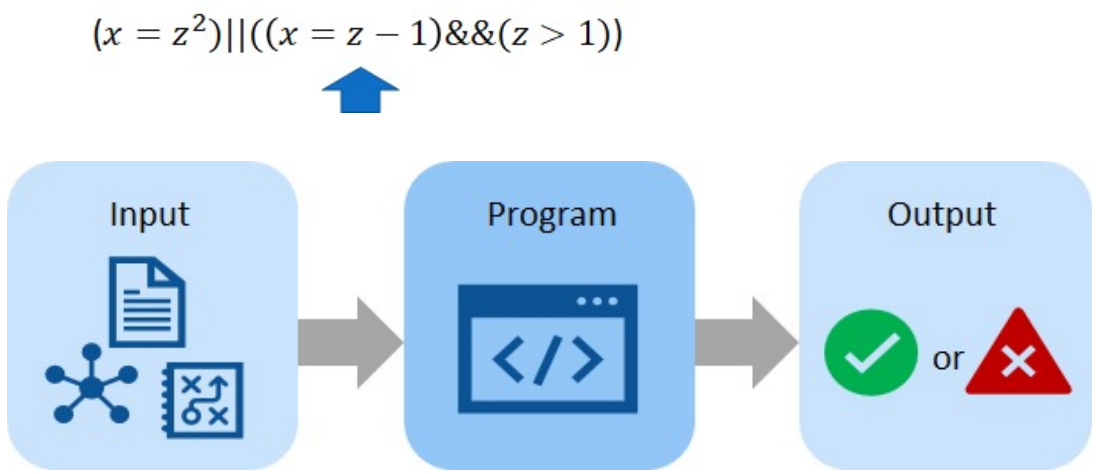
- Example: seL4 kernel (Klein, SOSP 2009), FSCQ file system (Chen, SOSP 2015), ...



# Formal Verification: theorem proving

- Step 1: define program behavior mathematically

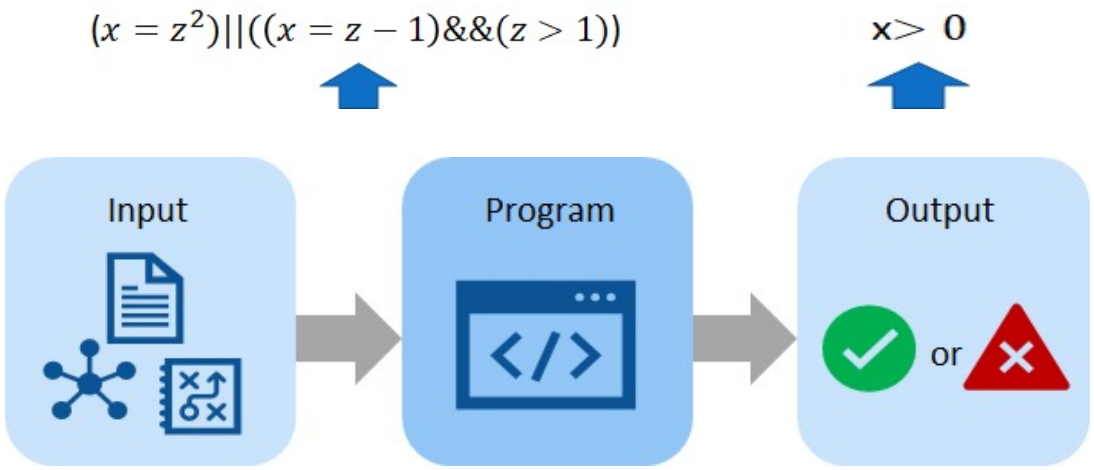
- Example: seL4 kernel (Klein, SOSP 2009), FSCQ file system (Chen, SOSP 2015), ...



# Formal Verification: theorem proving

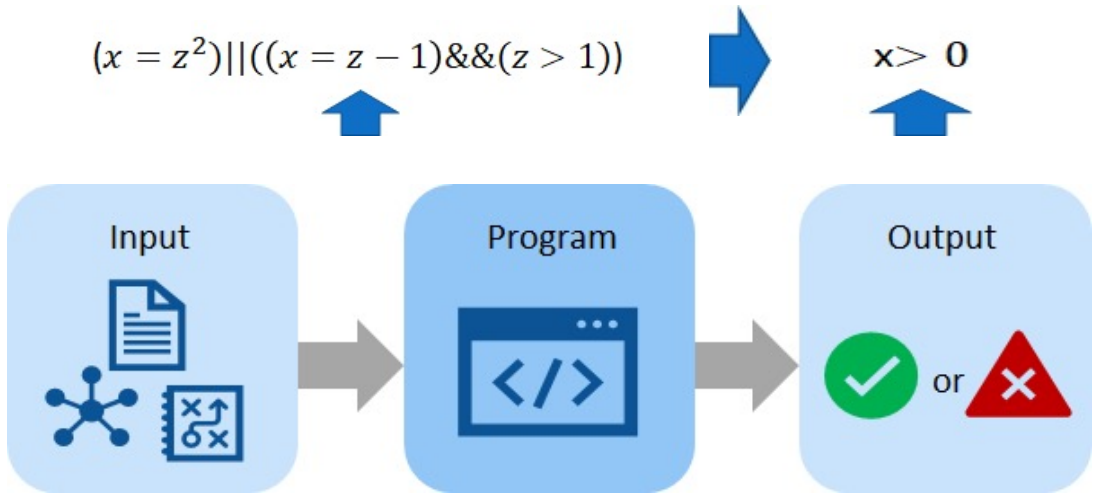
- Step 1: define program behavior mathematically
- Step 2: define correct specification mathematically

- Example: seL4 kernel (Klein, SOSP 2009), FSCQ file system (Chen, SOSP 2015), ...



# Formal Verification: theorem proving

- Step 1: define program behavior mathematically
- Step 2: define correct specification mathematically
- Step 3: prove statements with theorem provers (e.g. Coq)
- Example: seL4 kernel (Klein, SOSP 2009), FSCQ file system (Chen, SOSP 2015), ...



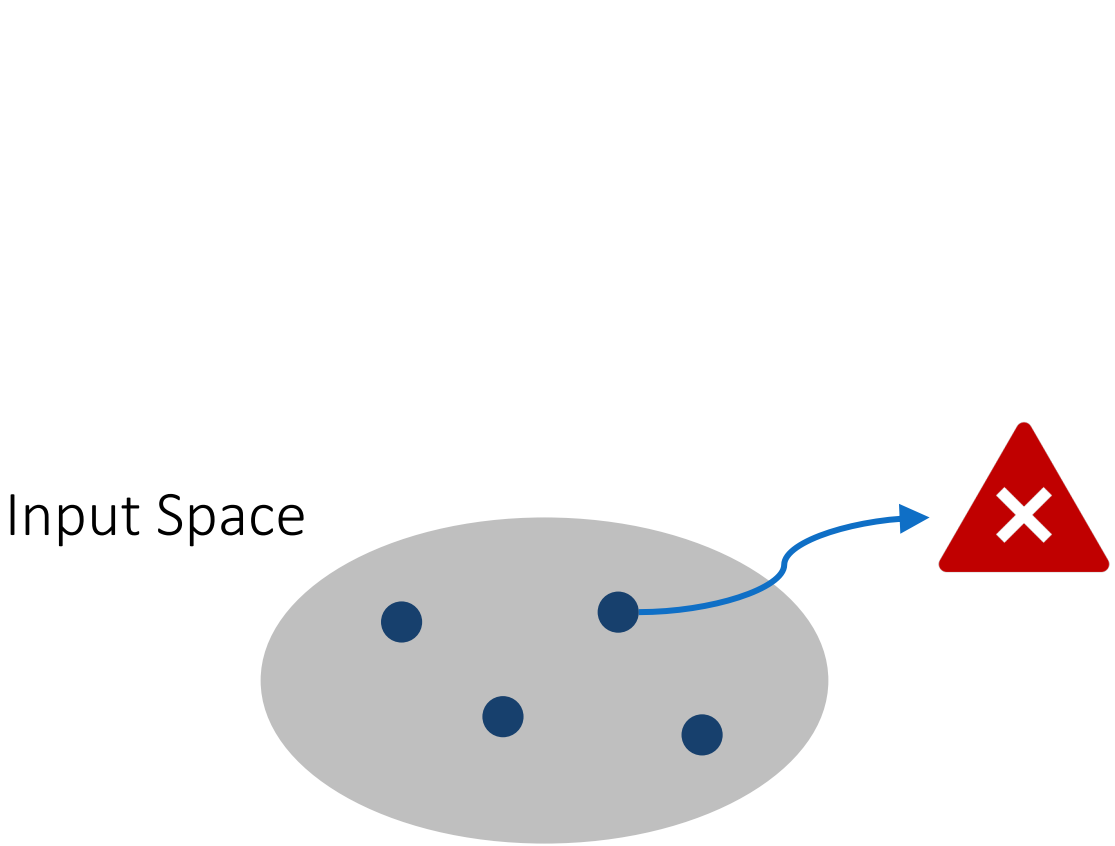
# Formal Verification: theorem proving

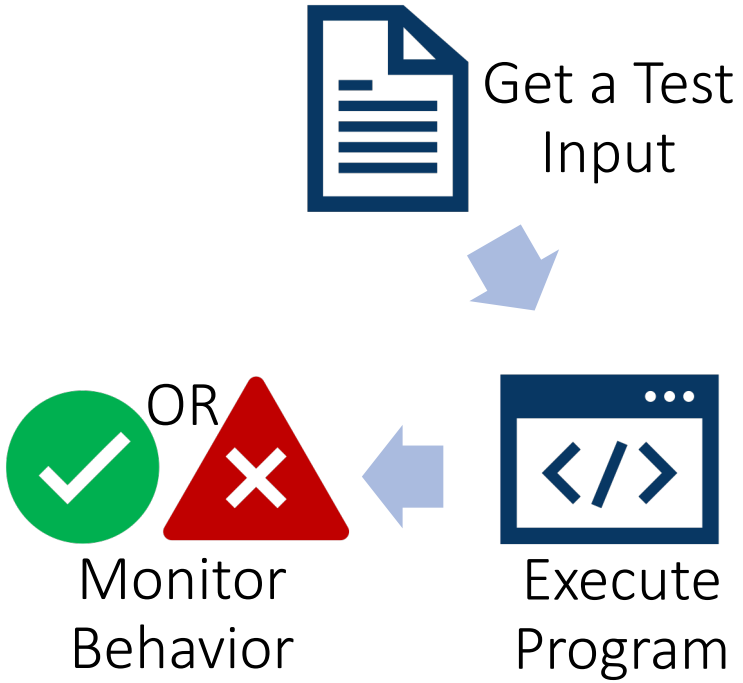
- Step 1: define program behavior mathematically
- Step 2: define correct specification mathematically
- Step 3: prove statements with theorem provers (e.g. Coq)
- Example: seL4 kernel (Klein, SOSP 2009), FSCQ file system (Chen, SOSP 2015), ...

## Weakness

- Huge manual effort to write proofs (proof can be bigger than program!)
- Behaviors/spec. may not directly correspond to math statements

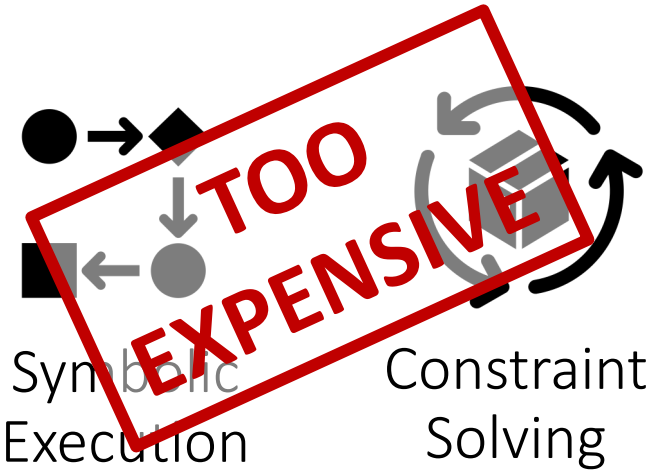
# Testing Overview



# Symbolic Execution

```
int main(int x, int y){
  if (x > 10){
    if (y == x+1){
      assert(false);
  }}
  return x/(y-1);
}
```




**TEST INPUT**

(x=11, y=12)

Get a Test Input

**PATH CONDITION**

[1] x>10 [2] y==x+1

OR

Monitor Behavior

Execute Program

Reference:

[1] Ciortea, Liviu, Cristian Zamfir, Stefan Bucur, Vitaly Chipounov, and George Candea. "Cloud9: A software testing service." ACM SIGOPS Operating Systems Review 43, no. 4 (2010): 5-10..

[2] Yang, Junfeng, Can Sar, Paul Twohey, Cristian Cadar, and Dawson Engler. "Automatically generating malicious disks using symbolic execution." In 2006 IEEE Symposium on Security and Privacy (S&P'06), pp. 15-pp. IEEE, 2006.

[3] Baldoni, Roberto, Emilio Coppa, Daniele Cono D'elia, Camil Demetrescu, and Irene Finocchi. "A survey of symbolic execution techniques." ACM Computing Surveys (CSUR) 51, no. 3 (2018): 1-39.

# Fuzz Testing

```
int main(int x, int y){
  if (x > 10){
    if (y == x+1){
      assert(false);
  }}
  return x/(y-1);
}
```

? How to mutate structured / domain-specific data?

Mutation

**TEST INPUT**

(x=11, y=1) Pass

...

(x=11, y=12) Fail

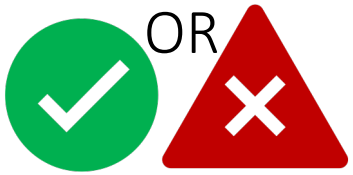
Get a Test Input

Collect & Process Feedback

**INPUT SEED**

(x=0, y=0) (x=1, y=10)

(x=0, x=12) (x=1, y=11)



OR

Monitor Behavior

Execute Program

Reference:

[1] Yang, Youngseok, Taesoo Kim, and Byung-Gon Chun. "Finding Consensus Bugs in Ethereum via Multi-transaction Differential Fuzzing." In OSDI, pp. 349-365. 2021.

[2] Lemieux, Caroline, and Koushik Sen. "Fairfuzz: A targeted mutation strategy for increasing greybox fuzz testing coverage." In ASE, pp. 475-485. 2018.

[3] Jeon, Yuseok, WookHyun Han, Nathan Burow, and Mathias Payer. “FuZZan: Efficient sanitizer metadata design for fuzzing.” In USENIX ATC, pp. 249-263. 2020.

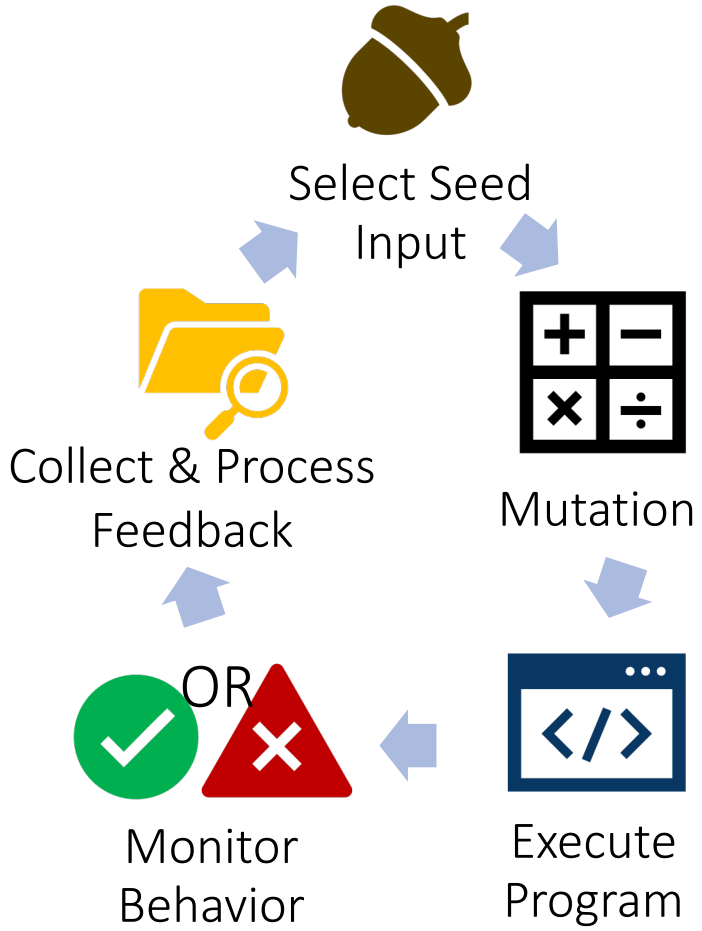
# Fuzz Testing

## Strength

- Lower cost than symbolic execution
- Less implementation effort

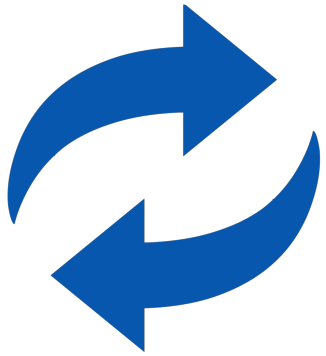
## Weakness

- Need to design efficient mutation algorithm
- Hard to manage structured/domain-specific inputs
- Highly rely on randomness



# Logging

- Log runtime information to help ...



Reproduce a particular bug

Monitor program status

Find optimizing opportunities

# Dynamic Checking

- Analyze the execution following bug patterns
  - Locking problems (Engler, SOSP 2001; Kasikci, SOSP 2013; Li, SOSP 2019; …)
  - Security problems (Costa, SOSP 2007; Yip, SOSP 2009; Hicks, ASPLOS 2015; …)
  - File system consistency (Gunawi, OSDI 2008; Fryer, TOS 2012; …)

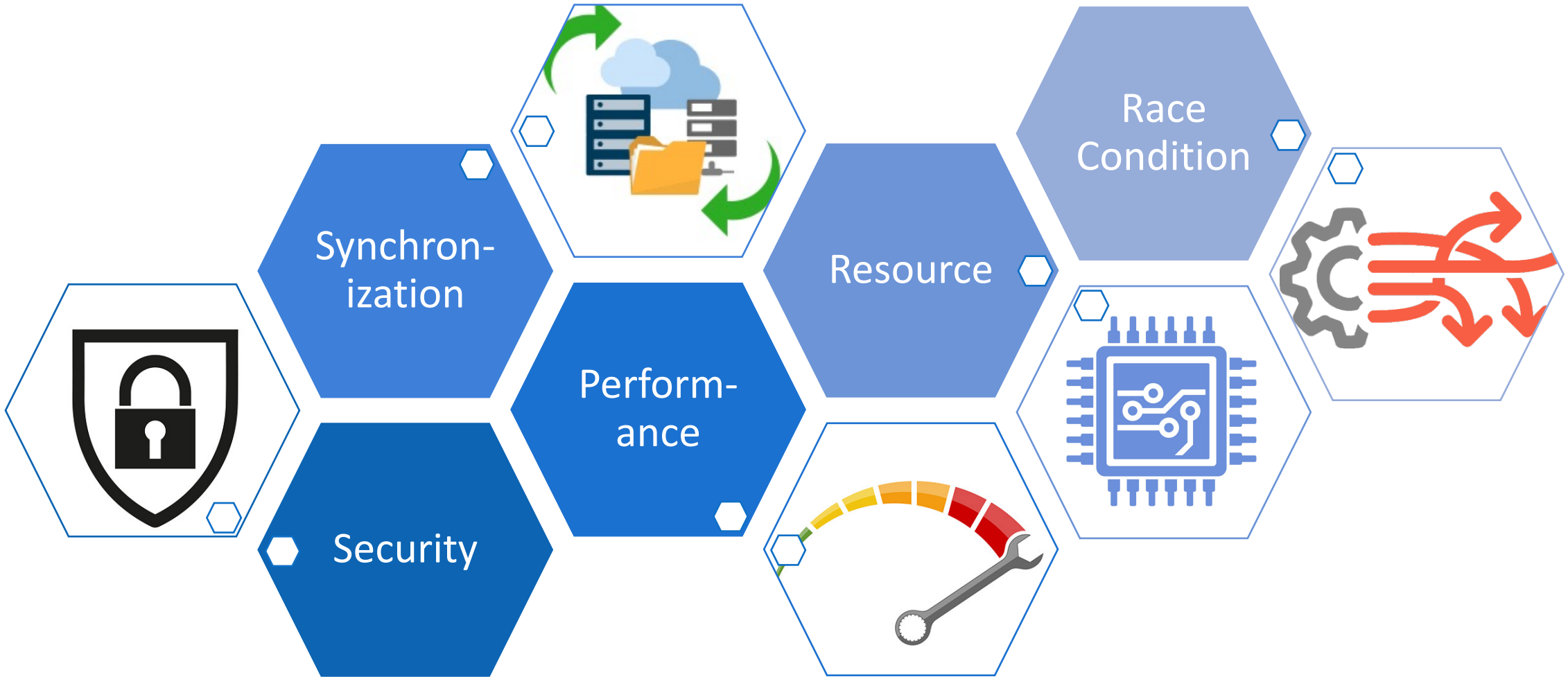
## Strength

- More accurate runtime information
- Allowing failure prevention

## Weakness

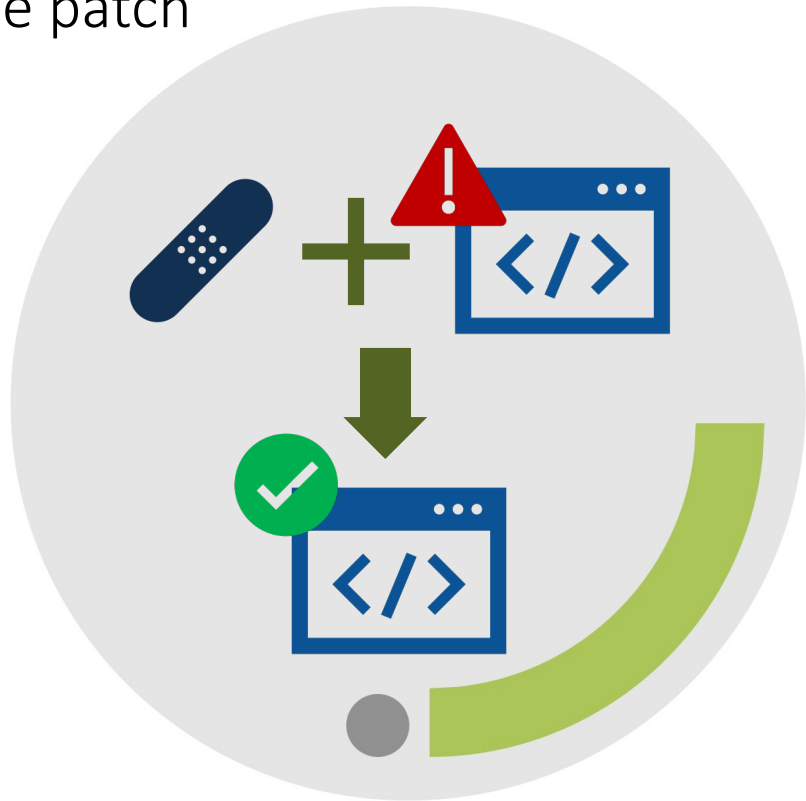
- Limited to specific bug patterns (False negatives)
- Runtime overhead

# Failure Prevention



# Patching

- Software patching is expensive
  - Design cost: thorough testing to avoid introducing new bugs
  - Deployment cost: Restarting the software to adopt the patch
- Solution
  - Patch management tool
  - Automated patching
  - On-the-fly patching
  - Etc.



Reference:
[1] Durieux, Thomas, Youssef Hamadi, and Martin Monperrus. "Production-driven patch generation." In 2017 ICSE-NIER, pp. 23-26. IEEE, 2017.
[2] Baumann, Andrew, Gernot Heiser, Jonathan Appavoo, Dilma Da Silva, Orran Krieger, Robert W. Wisniewski, and Jeremy Kerr. "Providing Dynamic Update in an Operating System." In USENIX ATC, pp. 279-291. 2005.

# PYLIVE: On-the-Fly Code Change for Python-based Online Services

Haochen Huang, Chengcheng Xiang, Li Zhong, and Yuanyuan Zhou,
University of California, San Diego

# PYLIVE

Haochen Huang, Chengcheng Xiang, Li Zhong, and Yuanyuan Zhou, University of California, San Diego


- ☐ Static checking
- ☐ Formal Verification
- ☐ …

- ☐ Fuzz testing
- ☐ …

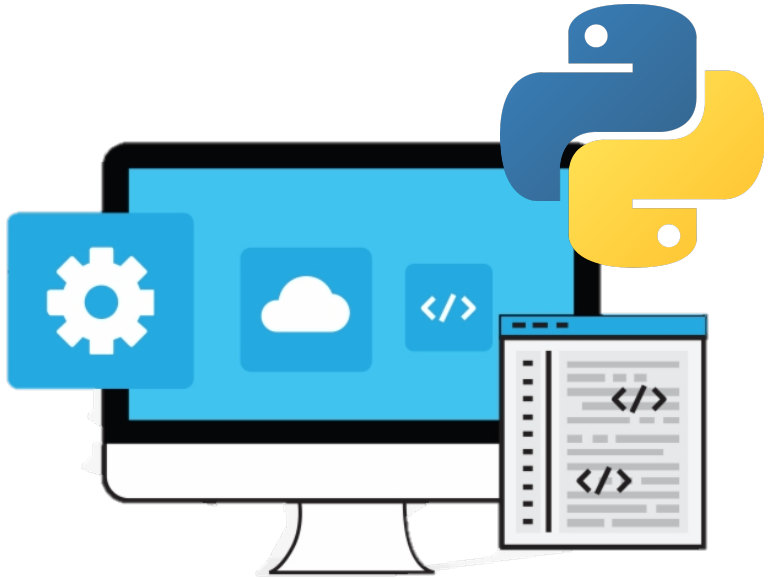
- ☐ Logging
- ☐ Dynamic checking
- ☐ Failure prevention
- ☐ …

- ☐ Failure diagnosis
- ☑ Patching
- ☐ …

# PYLIVE

Haochen Huang, Chengcheng Xiang, Li Zhong, and Yuanyuan Zhou, University of California, San Diego

- **WHAT:** Online patching without service down time
  - Support dynamic logging, profiling and bug-fixing
- **WHY:** Restarting server instances is expensive
- **HOW:** Utilizing Python language features
  - Meta-object protocol
  - Dynamic typing
  - Python byte-code



# Two Fuzz Testing Papers

RIFF: Reduced Instruction Footprint for Coverage-Guided Fuzzing;
TCP-Fuzz: Detecting Memory and Semantic Bugs in TCP Stacks with Fuzzing

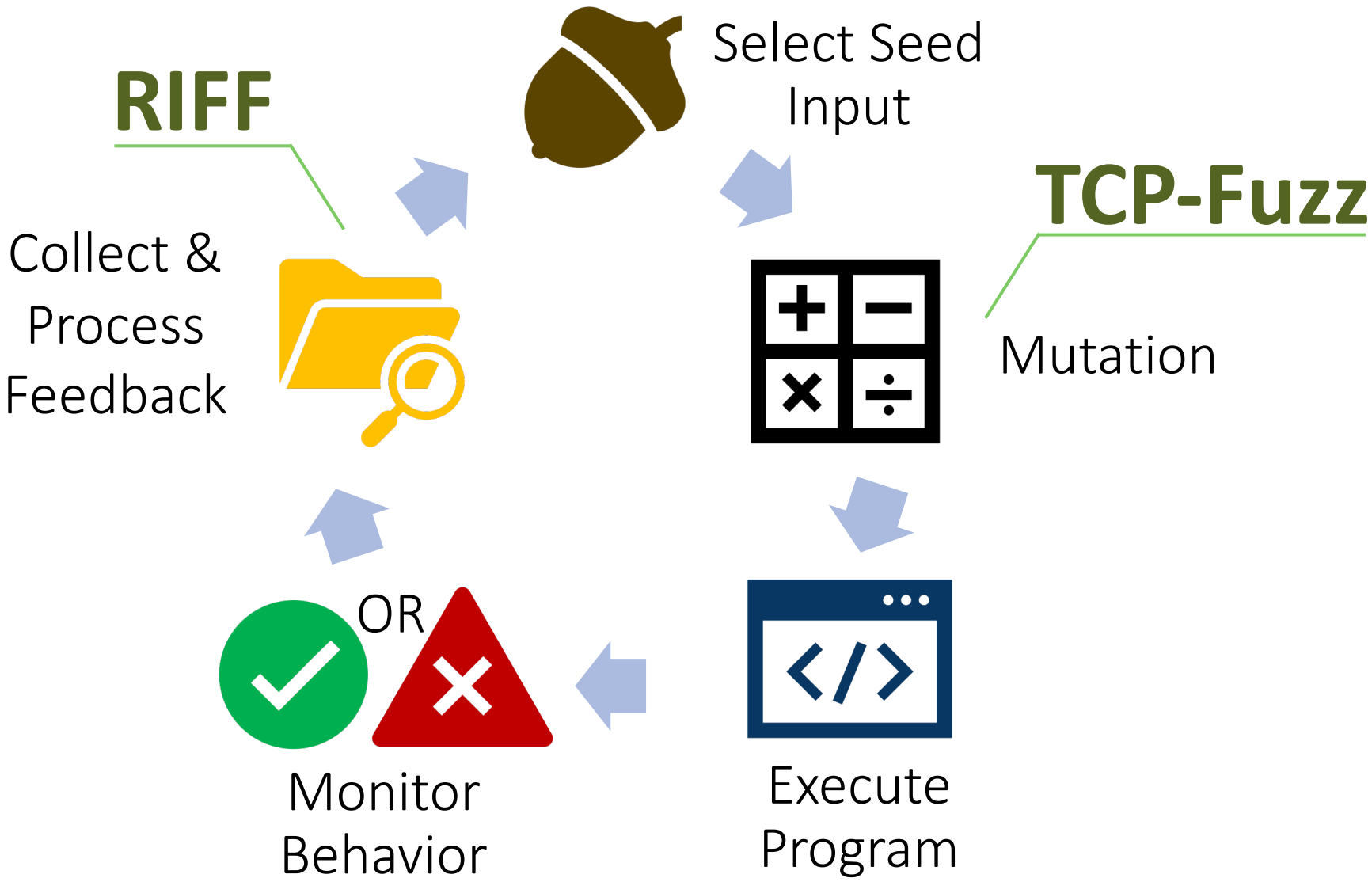

- ☐ Static checking
- ☐ Formal Verification
- ☐ …

- ☑ Fuzz testing
- ☐ …

- ☐ Logging
- ☐ Dynamic checking
- ☐ Failure prevention
- ☐ …

- ☐ Failure diagnosis
- ☐ Patching
- ☐ …

# Fuzz Testing Workflow



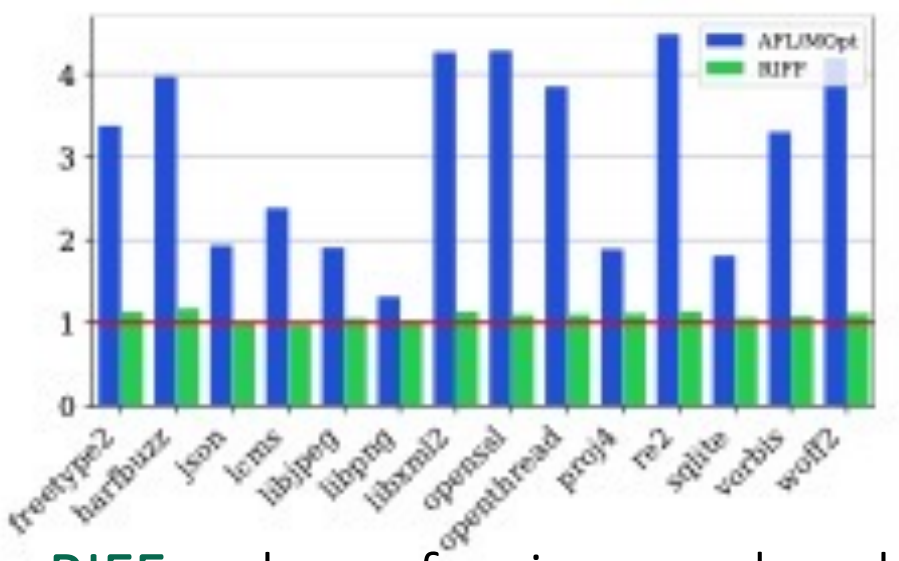
# RIFF: Reduced Instruction Footprint for Coverage-Guided Fuzzing

Mingzhe Wang, Jie Liang, Chijin Zhou, and Yu Jiang, Tsinghua University; Rui Wang, Capital Normal University; Chengnian Sun, Waterloo University; Jiaguang Sun, Tsinghua University

# RIFF

Mingzhe Wang, Jie Liang, Chijin Zhou, and Yu Jiang, Tsinghua University; Rui Wang, Capital Normal University; Chengnian Sun, Waterloo University; Jiaguang Sun, Tsinghua University

- **WHAT:** Reduce fuzzing feedback overhead
- **WHY:** Feedback collection is expensive
- HOW:
  - Feedback collection
    - Optimize indexing with static code analysis
  - Feedback processing
    - Compute coverage in 3 levels of granularity




RIFF reduces fuzzing overhead to one instruction per site

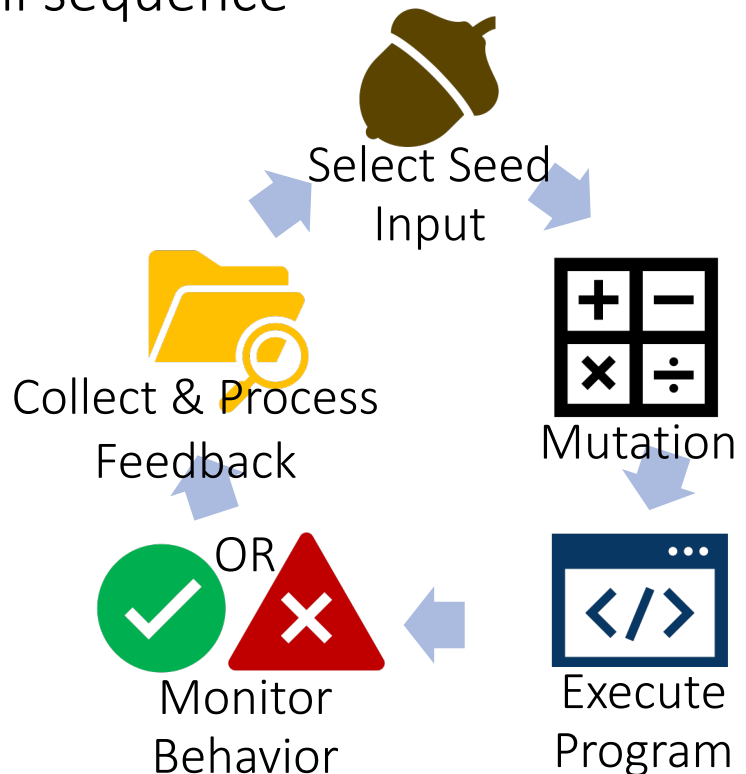
# TCP-Fuzz: Detecting Memory and Semantic Bugs in TCP Stacks with Fuzzing

Yong-Hao Zou and Jia-Ju Bai, Tsinghua University; Jielong Zhou, Jianfeng Tan, and Chenggang Qin, Ant Group; Shi-Min Hu, Tsinghua University

# TCP-Fuzz

Yong-Hao Zou and Jia-Ju Bai, Tsinghua University; Jielong Zhou, Jianfeng Tan, and Chenggang Qin, Ant Group; Shi-Min Hu, Tsinghua University

- **WHAT:** domain specific fuzzer for TCP stack
  - Automatic generate TCP packets and network system call sequence
- **WHY:** Testing TCP stack is difficult
  - Semantic constraints on inputs
  - Complex state transitions in TCP stacks
- **HOW:**
  - Domain-specific seed mutation
  - Feedback collection on state transition
  - Monitor software behavior with a differential checker



# MLEE: Effective Detection of Memory Leaks on Early-Exit Paths in OS Kernels

Wenwen Wang, University of Georgia

# MLEE

Wenwen Wang, University of Georgia


- ☑ Static checking
- ☐ Formal Verification
- ☐ …

- ☐ Fuzz testing
- ☐ …

- ☐ Logging
- ☐ Dynamic checking
- ☐ Failure prevention
- ☐ …

- ☐ Failure diagnosis
- ☐ Patching
- ☐ …

# MLEE

Wenwen Wang, University of Georgia

An early-exit path in Linux kernel

```
1  /* mm/mempool.c */
2  int mempool_resize(mempool_t *pool, int new_min_nr) {
3    ...
4    spin_lock_irqsave(&pool->lock, flags);
5    if (new_min_nr <= pool->min_nr) {
6      spin_unlock_irqrestore(&pool->lock, flags);
7      kfree(new_elements); // A memory deallocation.
8      return 0;
9    }
10   ...
11   return 0;
12 }
```

- WHAT: detect memory leaks @ early exits
- WHY: leaks are common @ early exits
- HOW:
  - An empirical study of early exits in kernels
  - Looking for objects de-allocated @ regular exits but not @ early exits

*120 new memory leak bugs found in Linux Kernel!*

# Argus: Debugging Performance Issues in Modern Desktop Applications with Annotated Causal Tracing

Lingmei Weng, Columbia University; Peng Huang, Johns Hopkins University; Junfeng Yang and Jason Nieh, Columbia University

# Argus

Lingmei Weng, Columbia University; Peng Huang, Johns Hopkins University;
Junfeng Yang and Jason Nieh, Columbia University


- ☐ Static checking
- ☐ Formal Verification
- ☐ …

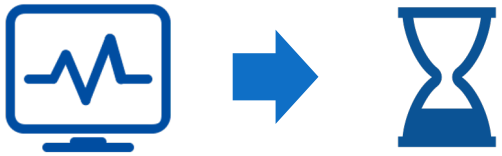
- ☐ Fuzz testing
- ☐ …

- ☑ Logging
- ☐ Dynamic checking
- ☐ Failure prevention
- ☐ …

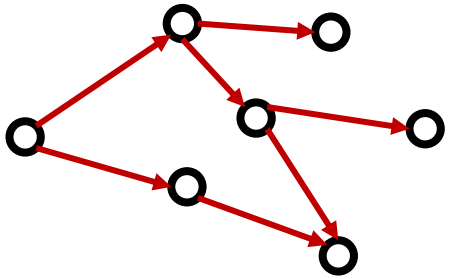
- ☐ Failure diagnosis
- ☐ Patching
- ☐ …

# Argus

Lingmei Weng, Columbia University; Peng Huang, Johns Hopkins University;
Junfeng Yang and Jason Nieh, Columbia University



Performance bug at runtime



Log

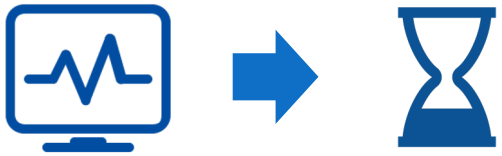
: Program segment

: Causal relation

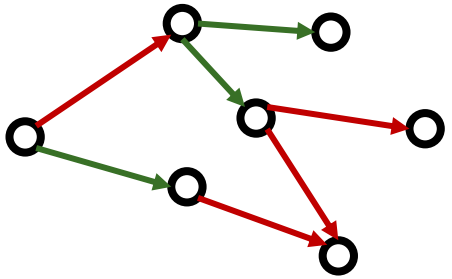
- WHAT: Performance diagnosis through causal analysis
- WHY challenging:
  - A wide variety of causal relations
  - Different causal relation has different confidence

# Argus

Lingmei Weng, Columbia University; Peng Huang, Johns Hopkins University;
Junfeng Yang and Jason Nieh, Columbia University



Performance bug at runtime



Log

: Program segment

: Strong Causal relation

: Weak Causal relation

- WHAT: Performance diagnosis through causal analysis
- WHY challenging:
  - A wide variety of causal relations
  - Different causal relation has different confidence
- HOW:
  - Divide causal relations into
    - Strong edges: causal relations that we are confident with
    - Weak edges: causal relations we are not exactly sure
  - Beam search to find the path of root cause

*Identified root causes of 12 spinning pinwheel bugs in MacOS!*