TELOS Talks - Fuzzing

Yonghao Zou

2025.05.22

TELOS Talks - A Great Plan

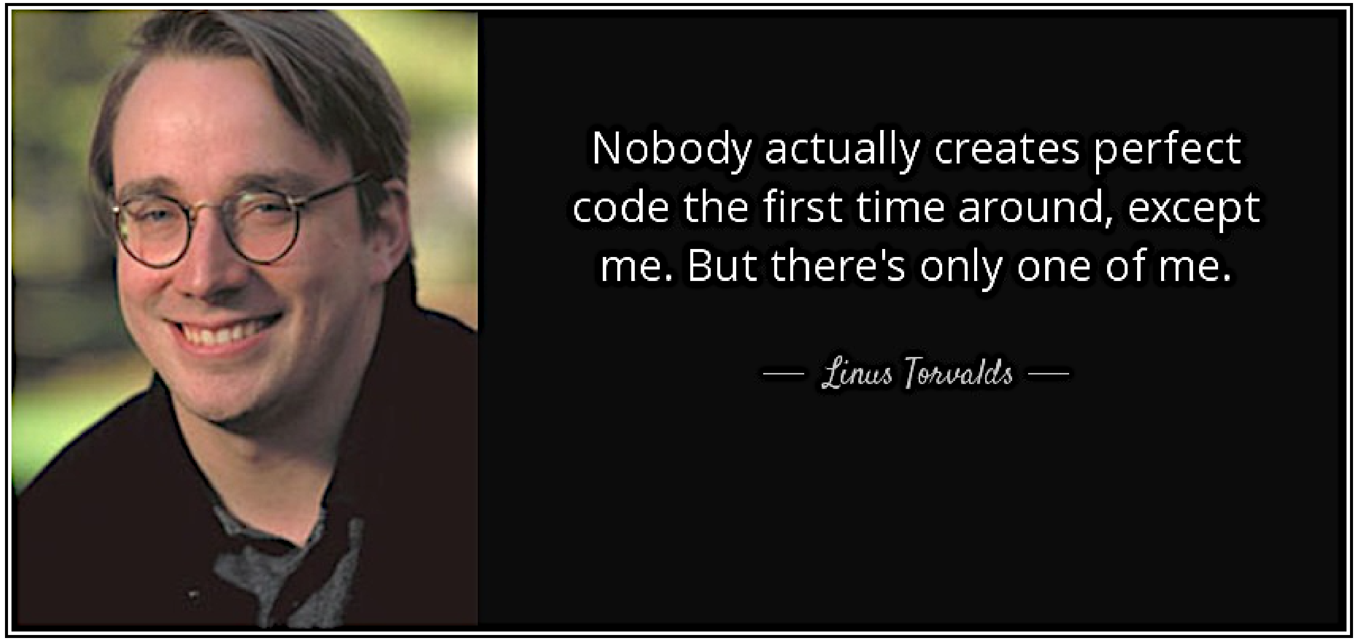
- Why?
  - Learn something new
  - Exchange ideas
  - Add “Talks” to your personal page
- How?
  - Just talk about the areas you know more (than folks in TELOS)
- When?
  - Lunch time

TELOS Talks - Call for Talks

- Current scheduled talks
  - Introduction to Fuzzing – Yonghao, now
  - Introduction to ML Sys – Zehua, in 2/3 weeks
  - Rust in depth – Junyang, in a month
  - Drawing figures for our paper – Junyang, in 2 months
- Something I can imagine
  - Intro to static analysis, LLVM, Verus…
  - Writing a symbolic execution engine/verifier, a kernel in Rust, a memory system, a user-level file system, a scheduler using eBPF, from scratch
  - File systems, user interrupts, and more system topics in depth
  - GPU, FPGA, ML, Tired Memory, …

Fuzzing - Background

- Do you write perfect code?



Fuzzing - Background

- If you are not Linus...
  - Manual
    - Code reviews
    - Unit tests
  - Automatic testing
    - ***Fuzzing***
  - Program analysis
    - Static analysis
    - Symbolic execution
  - Formal methods

Fuzzing - Background

- Pro and cons
  - Unit tests
    - White box
    - Manual effort
  - **Fuzzing**
    - Effective
    - Cannot guarantee the absence of bugs
  - Static analysis
    - Fast, easy to use, static
    - False positive/negative

Fuzzing - Background

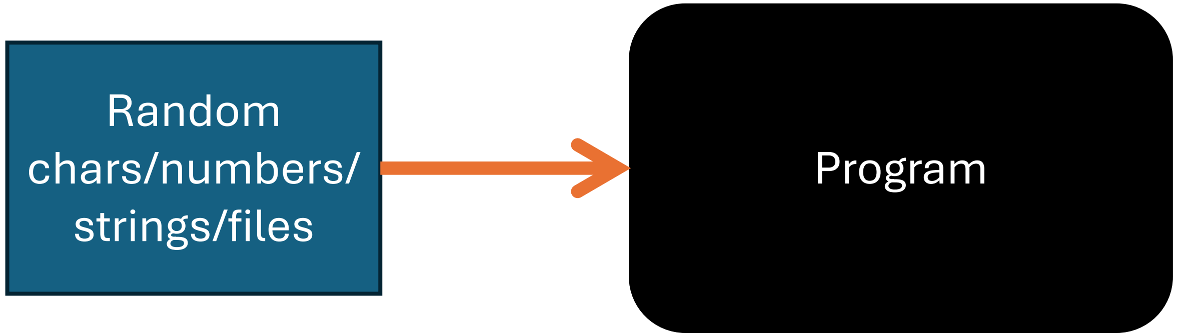
- Pro and cons
  - Symbolic execution
    - Check all possibility of the program
    - Scalability issue
  - Formal methods
    - Prove the absence of certain bugs
    - Manual effort, scalability issue

Fuzzing - Blackbox fuzzing

- You know nothing or very little about the program

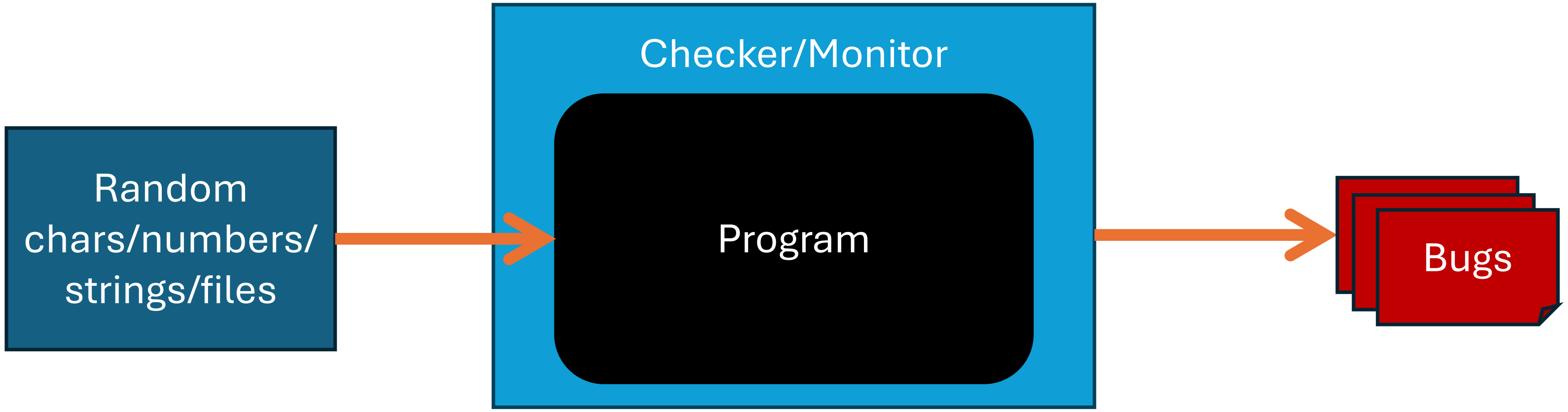
Fuzzing - Blackbox fuzzing

- Random input
  - E.g., /dev/urandom



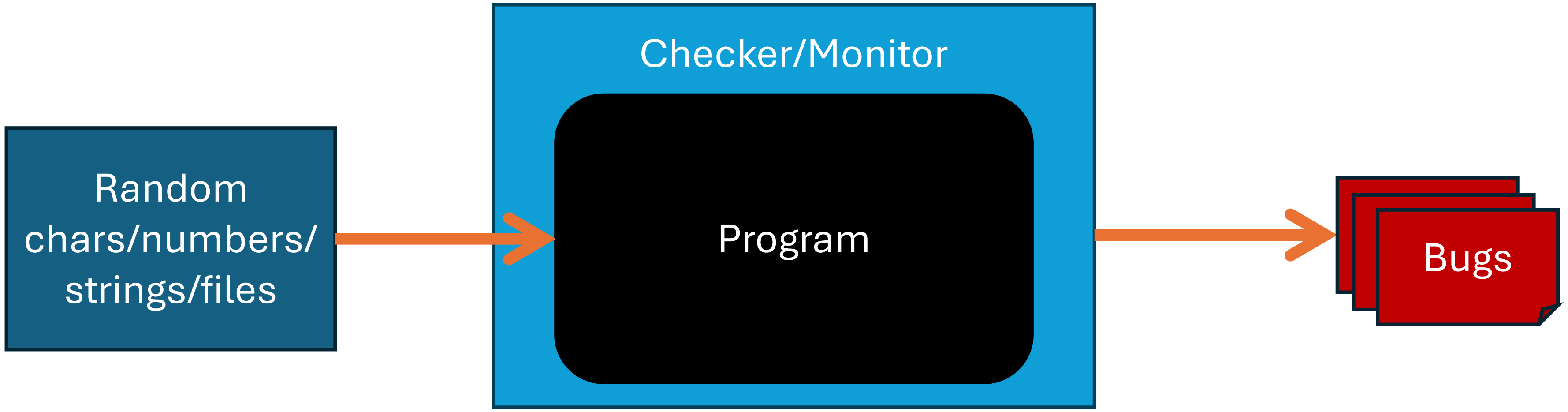
Fuzzing - Blackbox fuzzing

- Checker/Monitor



Fuzzing - Blackbox fuzzing

- Easy, but ineffective



- If we can know something about the program, what do we actually want to know?
  - We want to know whether an input is good or not

Fuzzing - Blackbox fuzzing

- Insight from practice: If an input is “good”, then a similar input is probably “good”.
  - Input 1: [99, 200]
  - Input 2: [99, 201]

```
if (input[0] < 100) {
  xxx
  if (input[1] > 200) {
    bug_on(...);
  }
}
```

Fuzzing - What is an "good" input?

- A good input can trigger bugs
- Insight: more coverage -> more bugs
- Good input: increase coverage

```
if (input[0] < 100) {
  xxx
  if (input[1] > 200) {
    bug_on(...);
  }
}
```

Fuzzing - Coverage

- Line coverage vs. Edge coverage
  - Input 1: [99, 201]
  - Input 2: [99, 200]
  - Is input 2 a good input?

```
if (input[0] < 100) {
  xxx
  if (input[1] > 200) {
    xxx(...);
  }
}
```

Fuzzing - Coverage

- Line coverage vs. Edge coverage
  - Input 1: [99, 201]
  - Input 2: [99, 200]
  - Is input 2 a good input?

```
if (input[0] < 100) {
xxx
  if (input[1] > 200) {
    xxx(...);
  }

}
```

Fuzzing - Mutation

- Mutation-based fuzzing
- E.g, [99, 201]
  - Replace -> [1, 201], [2, 201], [99, 200], [99, 300]
  - Delete -> [99], [201]
  - Duplicate -> [99, 201, 99, 201]
  - Append -> [99, 201, 100, 300]
  - Bitflip
  - ...

Fuzzing - Generation

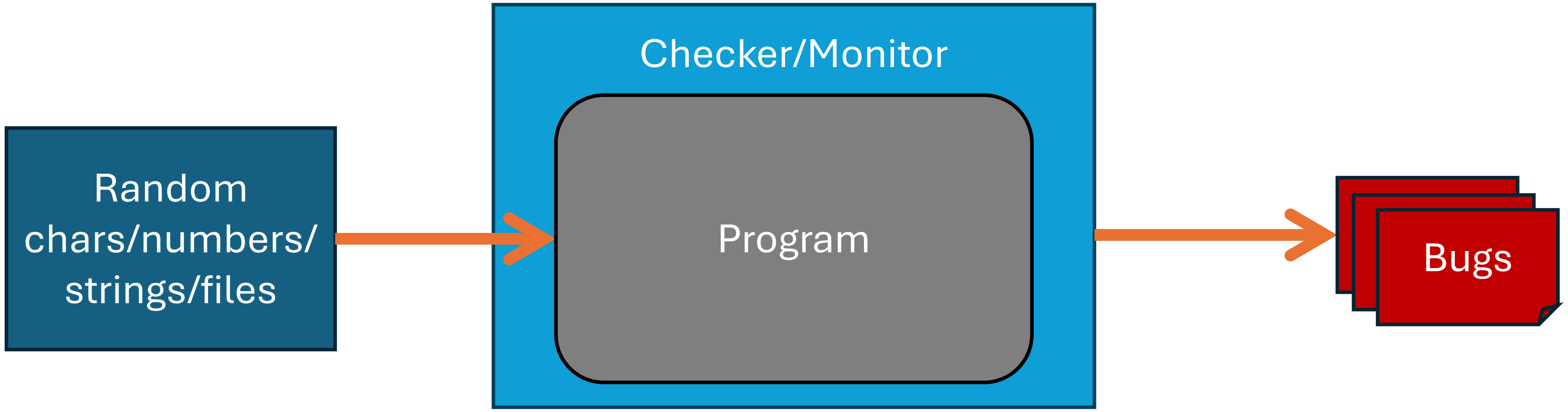
- Generation based fuzzing
- E.g., Network protocol fuzzer usually has a template
    - {IP, Port, Seq number, TCP Options, Timestamp, …}

```
<!--  A.  Local file header -->
  <Block name="LocalFileHeader">
    <String name="lfh_Signature" valueType="hex" value="504b0304" token="true" mut
    <Number name="lfh_Ver" size="16" endian="little" signed="false"/>
    ...
    [truncated for space]
    ...
    <Number name="lfh_CompSize" size="32" endian="little" signed="false">
      <Relation type="size" of="lfh_CompData"/>
    </Number>
    <Number name="lfh_DecompSize" size="32" endian="little" signed="false"/>
    <Number name="lfh_FileNameLen" size="16" endian="little" signed="false">
      <Relation type="size" of="lfh_FileName"/>
    </Number>
    <Number name="lfh_ExtraFldLen" size="16" endian="little" signed="false">
      <Relation type="size" of="lfh_FldName"/>
    </Number>
    <String name="lfh_FileName"/>
    <String name="lfh_FldName"/>
```

- In many cases, mutation-based is more effective

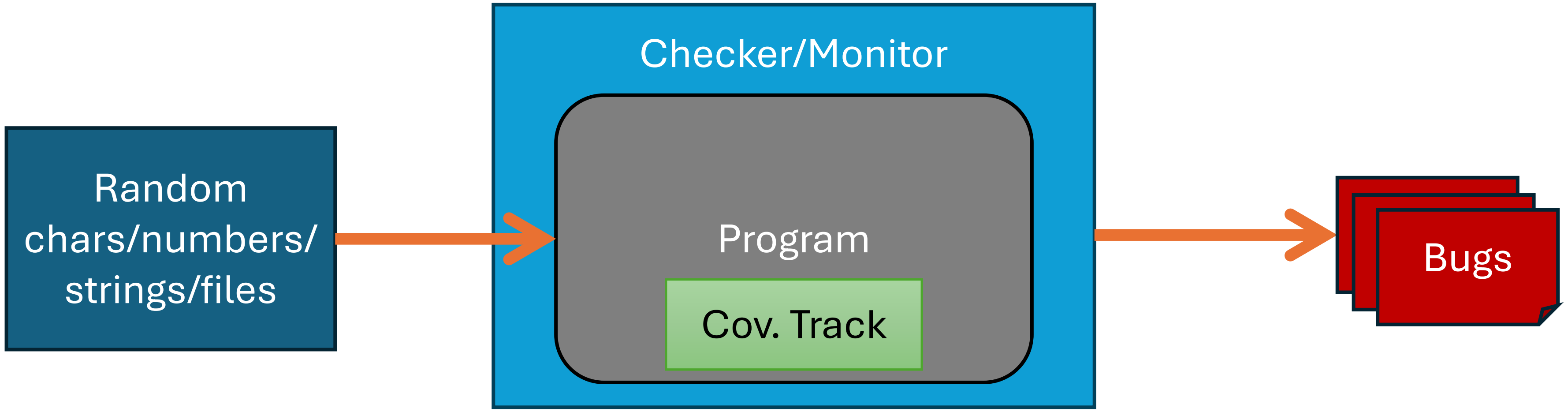
Fuzzing - Greybox fuzzing

- We do not need to know all, but we can know whether an input is "good"



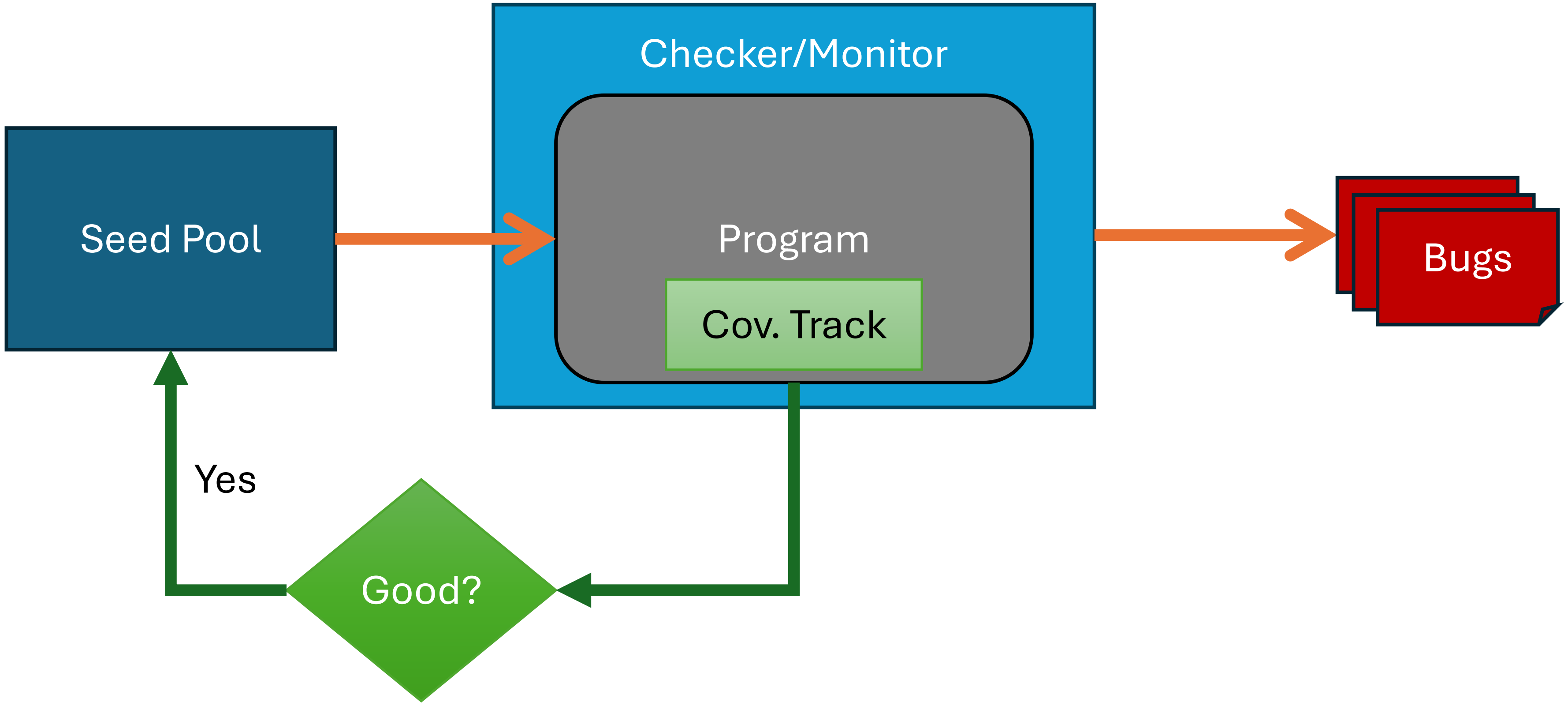
Fuzzing - Greybox fuzzing

- Coverage tracking



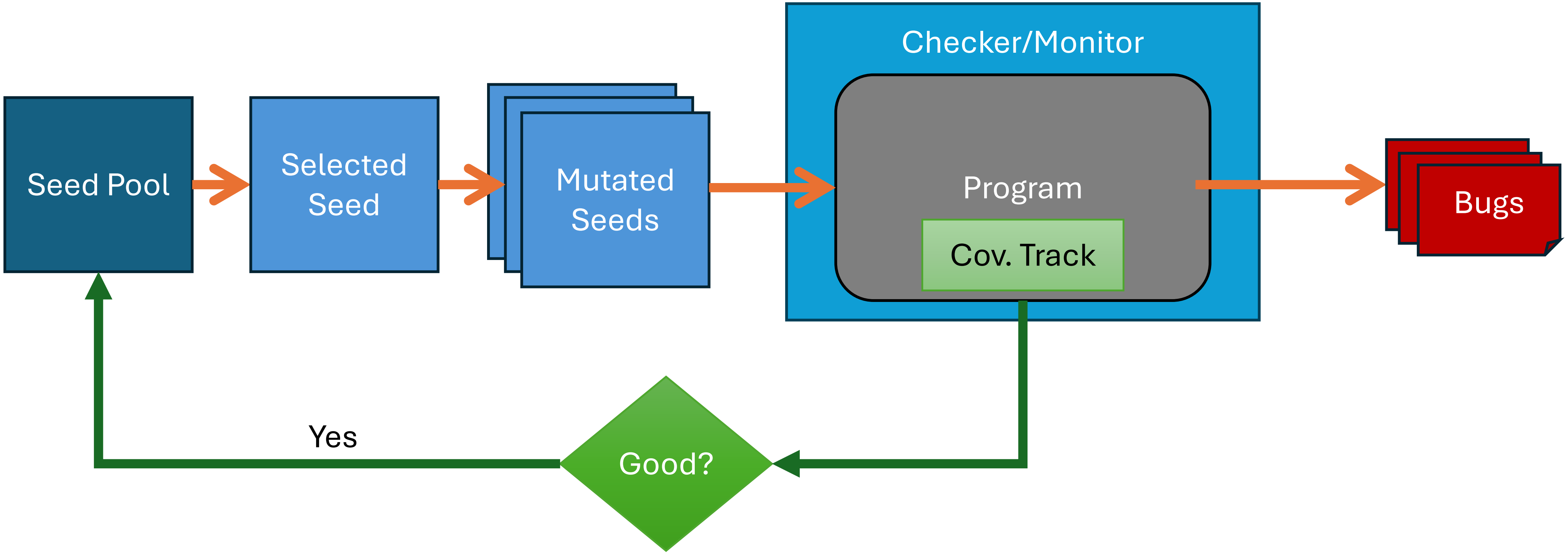
Fuzzing - Greybox fuzzing

- Input management
  - In fuzzing, we call them “seed”



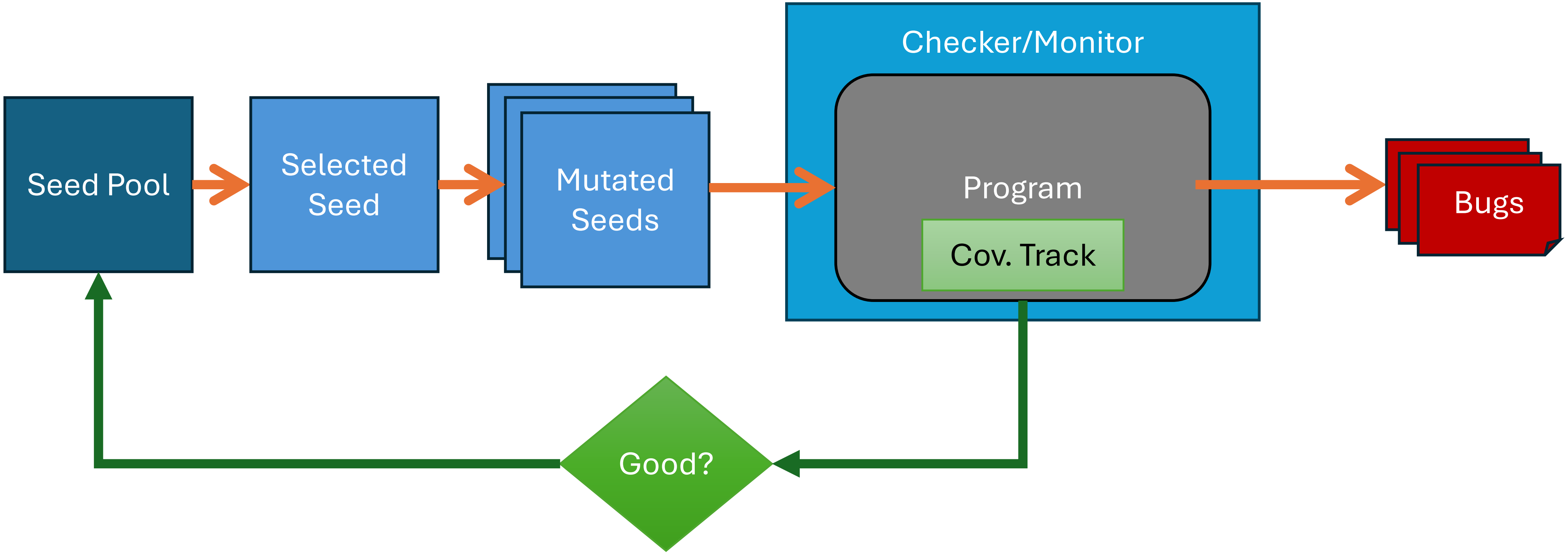
Fuzzing - Greybox fuzzing

- Seed selection and mutation



Fuzzing - Greybox fuzzing

- The tool – American Fuzzy Lop (AFL)



Fuzzing - Greybox fuzzing

- The tool – American Fuzzy Lop (AFL)
  - Filter input with code coverage
  - Mutation-based, few knowledge required
  - Fast: fork-server
  - Checkers
    - ASan
    - TSan
    - …

Fuzzing - Greybox fuzzing

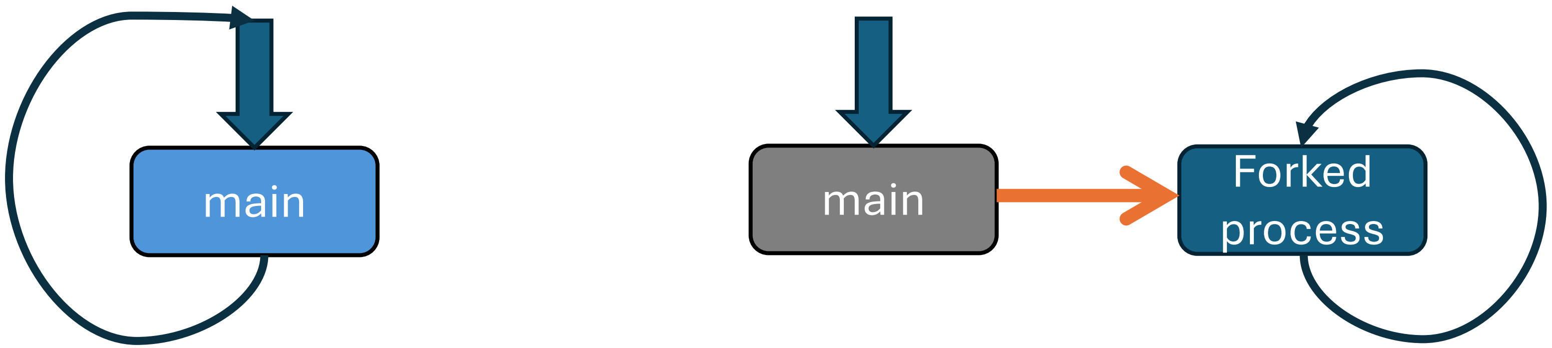
- How to get the coverage? Instrumentation
  - Insert code for each block
  - Must be short
  - Update the shared memory

```
if (input[0] < 100) {
  xxx
  if (input[1] > 200) {
    xxx(...);
  }
}
```

record edge

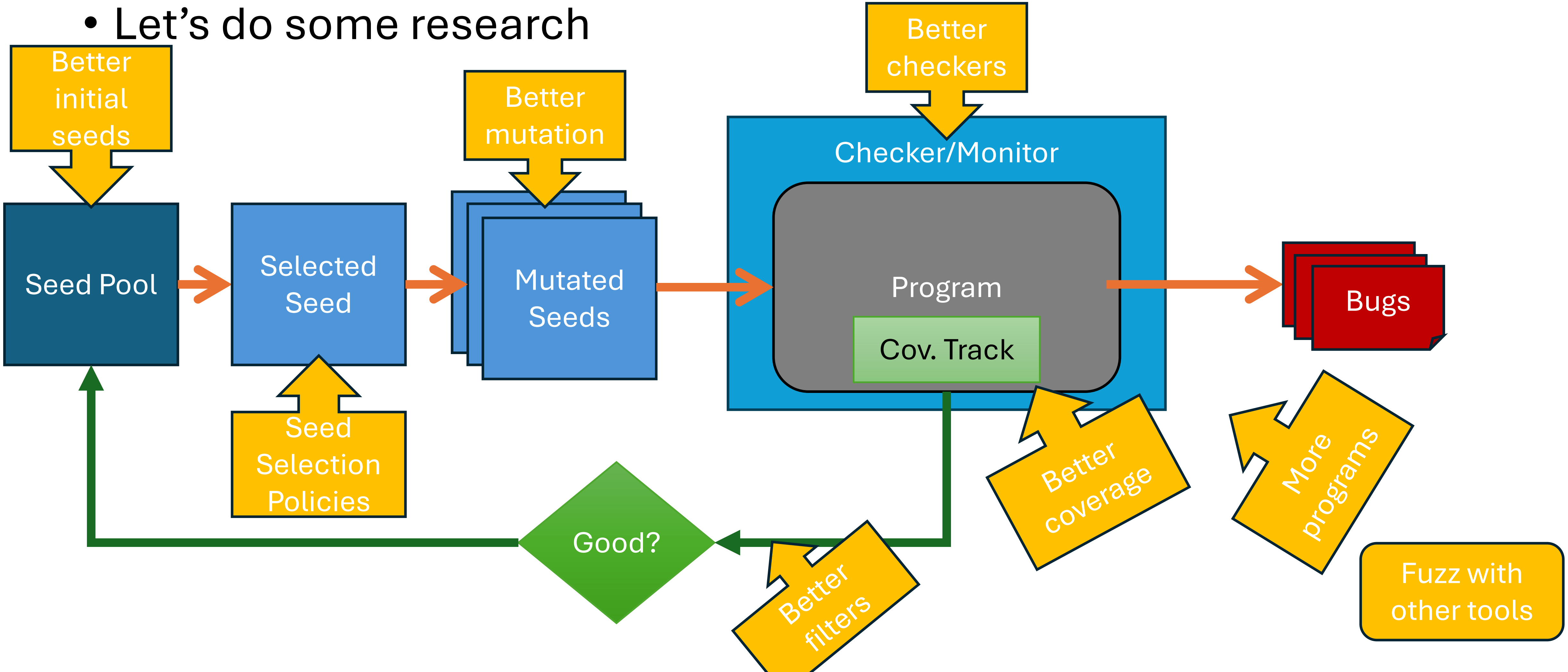
Fuzzing - Greybox fuzzing

- Fork server
  - To make fuzzing effective, you better make the program run fast (>1000 run/sec)
  - There are some loading/functions before “main”
  - We can pause at main, and fork the process to skip them



Fuzzing - Research

- Let’s do some research



Fuzzing – Ideas/Cases

- Better initial seeds
  - Use seeds made by experts
  - Generate seeds by symbolic execution -> concolic fuzzing (NDSS 16, ISSTA 18, CCS19, CCS 21)
    - What kind of programs is fuzzing ineffective for?

```
if (config->magic != MAGIC NUMBER)
    puts("Bad magic number");
```

  - Any other ideas?
    - 2-Phase fuzz, data analysis, log analysis…

Fuzzing – Ideas/Cases

- Seed Selection Policies (scheduling)
  - Coverage order
  - Augment order
  - Random
  - Seed minimization (ISSTA 21)
  - CFG analysis (S&P 22)

  - Performance varies for different policies (USENIX Sec 21)

Fuzzing – Ideas/Cases

- Better mutation

```
if (config->magic != MAGIC_NUMBER)
    puts("Bad magic number");
```

  - We can know that parts of the numbers match
    - MAGIC_NUMBER 0xFFFF_FFFE
    - Current input 0xFFFF_FFFF
    - Mutation based on branch distance (GreyOne, USENIX Sec 20)

Fuzzing – Ideas/Cases

- Better mutation
  - You can make it complex! (S&P 15)

$$\frac{v_c = \text{a concrete value from } src \quad v_a = \{\text{set of input bit positions}\}}{m_c, r_c, m_a, r_a \vdash \texttt{get\_input(src)} \Downarrow \langle v_c, v_a \rangle}\ \textsc{Input} \qquad \frac{}{m_c, r_c, m_a, r_a \vdash \texttt{var} \Downarrow \langle r_c[\texttt{var}], r_a[\texttt{var}] \rangle}\ \textsc{Var}$$

$$\frac{m_c, r_c, m_a, r_a \vdash e \Downarrow \langle v_c, v_a \rangle \quad v_c' = m_c[v_c] \quad v_a' = m_a[v_c] \Diamond_b v_a}{m_c, r_c, m_a, r_a \vdash \text{load } e \Downarrow \langle v_c', v_a' \rangle}\ \textsc{Load} \qquad \frac{m_c, r_c, m_a, r_a \vdash e \Downarrow \langle v_c, v_a \rangle \quad v_c' = \Diamond_u v_c}{m_c, r_c, m_a, r_a \vdash \Diamond_u e \Downarrow \langle v_c', v_a \rangle}\ \textsc{Unary-Op}$$

$$\frac{m_c, r_c, m_a, r_a \vdash e_1 \Downarrow \langle v_{c1}, v_{a1} \rangle \quad m_c, r_c, m_a, r_a \vdash e_2 \Downarrow \langle v_{c2}, v_{a2} \rangle}{m_c, r_c, m_a, r_a \vdash e_1 \Diamond_b e_2 \Downarrow \langle v_{c1} \Diamond_b v_{c2}, v_{a1} \Diamond_b v_{a2} \rangle}\ \textsc{Binary-Op} \qquad \frac{}{m_c, r_c, m_a, r_a \vdash v \Downarrow \langle v, \{\} \rangle}\ \textsc{Const}$$

$$\frac{m_c, r_c, m_a, r_a \vdash e \Downarrow \langle v_c, v_a \rangle \quad \begin{array}{l} r_c' = r_c[var \leftarrow v_c] \\ r_a' = r_a[var \leftarrow v_a] \end{array} \quad c' = \text{checkIDS}(c, pc) \quad \boxed{l' = l.\text{add}(\langle v_a, c'.\text{top}() \rangle)} \quad i = s[pc+1]}{s, m_c, r_c, m_a, r_a, \Delta, c, l, pc, \texttt{var} := e \rightsquigarrow s, m_c, r_c', m_a, r_a', \Delta, c', l', pc+1, i}\ \textsc{Assign}$$

$$\frac{m_c, r_c, m_a, r_a \vdash e \Downarrow \langle v_c, v_a \rangle \quad c' = \text{checkIDS}(c, pc) \quad i = s[v_c]}{s, m_c, r_c, m_a, r_a, \Delta, c, l, pc, \text{goto } e \rightsquigarrow s, m_c, r_c, m_a, r_a, \Delta, c', l, v_c, i}\ \textsc{Goto}$$

$$\frac{\begin{array}{l} m_c, r_c, m_a, r_a \vdash e \Downarrow \langle 1, v_a \rangle \\ m_c, r_c, m_a, r_a \vdash e_1 \Downarrow \langle v_{c1}, v_{a1} \rangle \end{array} \quad \begin{array}{l} c = \text{checkIDS}(c, pc) \\ c' = \text{updateIDS}(c, pc, v_a) \end{array} \quad l' = l.\text{add}(\langle v_{a1} \cup v_a, c'.\text{top}() \rangle) \quad i = s[v_{c1}]}{s, m_c, r_c, m_a, r_a, \Delta, c, l, pc, \text{if } e \text{ then goto } e_1 \text{ else goto } e_2 \rightsquigarrow s, m_c, r_c, m_a, r_a, \Delta, c', l', v_{c1}, i}\ \textsc{True-Cond}$$

$$\frac{\begin{array}{l} m_c, r_c, m_a, r_a \vdash e \Downarrow \langle 0, v_a \rangle \\ m_c, r_c, m_a, r_a \vdash e_2 \Downarrow \langle v_{c2}, v_{a2} \rangle \end{array} \quad \begin{array}{l} c = \text{checkIDS}(c, pc) \\ c' = \text{updateIDS}(c, pc, v_a) \end{array} \quad l' = l.\text{add}(\langle v_{a2} \cup v_a, c'.\text{top}() \rangle) \quad i = s[v_{c2}]}{s, m_c, r_c, m_a, r_a, \Delta, c, l, pc, \text{if } e \text{ then goto } e_1 \text{ else goto } e_2 \rightsquigarrow s, m_c, r_c, m_a, r_a, \Delta, c', l', v_{c2}, i}\ \textsc{False-Cond}$$

$$\frac{m_c, r_c, m_a, r_a \vdash e \Downarrow \langle v_c, v_a \rangle \quad c = \text{checkIDS}(c, pc) \quad c' = c.\text{push}(\langle pc+1, c.\text{pop}() \rangle) \quad i = s[v_c]}{s, m_c, r_c, m_a, r_a, \Delta, c, l, pc, \text{call } e \rightsquigarrow s, m_c, r_c, m_a, r_a, \Delta, c', l, v_c, i}\ \textsc{Call}$$

$$\frac{m_c, r_c, m_a, r_a \vdash e \Downarrow \langle v_c, v_a \rangle \quad c' = \text{returnIDS}(c, pc) \quad \langle \Delta', l' \rangle = \text{delayedUpdate}(\Delta, l) \quad i = s[v_c]}{s, m_c, r_c, m_a, r_a, \Delta, c, l, pc, \text{ret } e \rightsquigarrow s, m_c, r_c, m_a, r_a, \Delta', c', l', v_c, i}\ \textsc{Ret}$$

$$\frac{\begin{array}{l} m_c, r_c, m_a, r_a \vdash e_1 \Downarrow \langle v_{c1}, v_{a1} \rangle \\ m_c, r_c, m_a, r_a \vdash e_2 \Downarrow \langle v_{c2}, v_{a2} \rangle \end{array} \quad c' = \text{checkIDS}(c, pc) \quad \begin{array}{l} m_c' = m_c[v_{c1} \leftarrow v_{c2}] \\ m_a' = m_a[v_{c1} \leftarrow v_{a1} \Diamond_b v_{a2}] \end{array} \quad i = s[pc+1]}{s, m_c, r_c, m_a, r_a, \Delta, c, l, pc, \text{store}(e_1,\ e_2) \rightsquigarrow s, m_c', r_c, m_a', r_a, \Delta, c', l, pc+1, i}\ \textsc{Store}$$

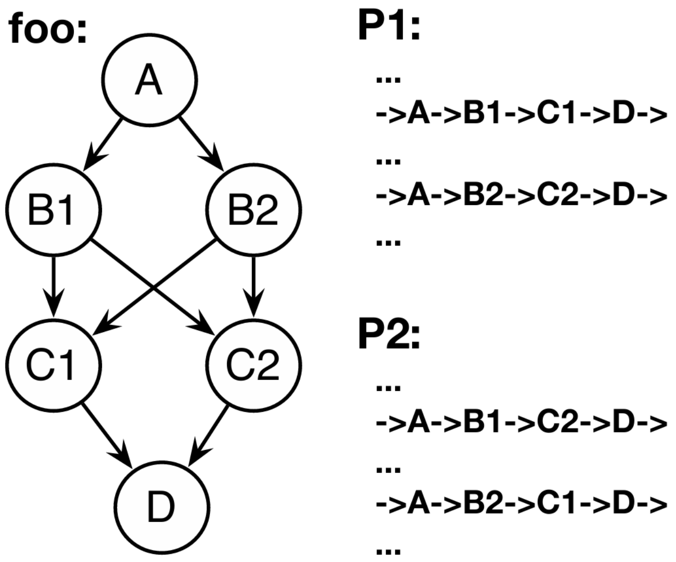
Fuzzing – Ideas/Cases

- Better coverage
  - Coverage calculation

$$cur \oplus (prev \gg 1)$$

  - Path sensitive coverage (CollAFL, S&P 18)

$$(cur \gg x) \oplus (prev \gg y) + z$$



Fuzzing – Ideas/Cases

- Better coverage
  - Coverage is often co-designed with seed selection
  - Coverage is often related to the target scenario

  - If you want to find memory usage bugs
    - MemLock: Memory Usage Guided Fuzzing
    - The more memory usage, the better

Fuzzing – Ideas/Cases

- More programs/bugs – Hot!
  - You already know OZZ: kernel concurrent bugs

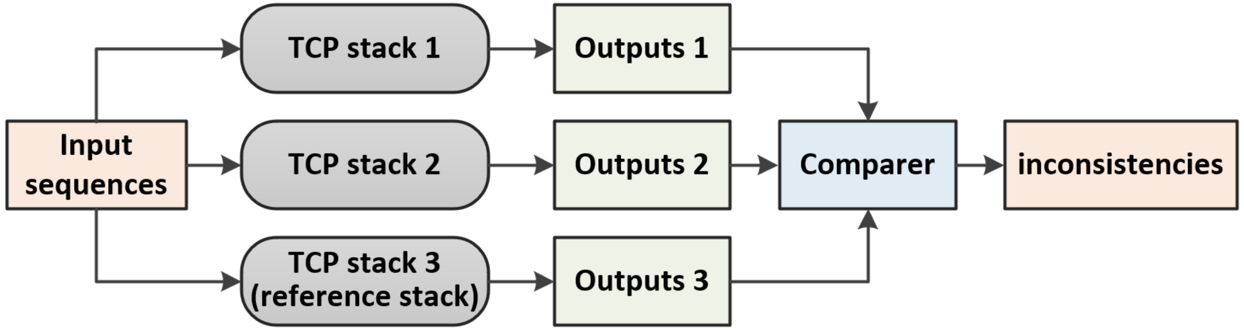
  - There's more:
    - Kernel
    - Network protocol (TCP-Fuzz)
    - Database
    - Mobile systems (Android, iOS)
    - Embedded systems
    - File systems
    - Distributed systems (DistFuzz)
    - Robotic systems (ROZZ)

Fuzzing – Ideas/Cases

- TCP-Fuzz: Detecting Memory and Semantic Bugs in TCP Stacks with Fuzzing (ATC 21)
  - Two-dimensional input: packets and system calls
  - Coverage: state transition coverage

$$\boldsymbol{BrTran_n = < BrCov_n, BrCov_n - BrCov_{n-1} >}$$

  - Checker: Differential checker



Fuzzing – Ideas/Cases

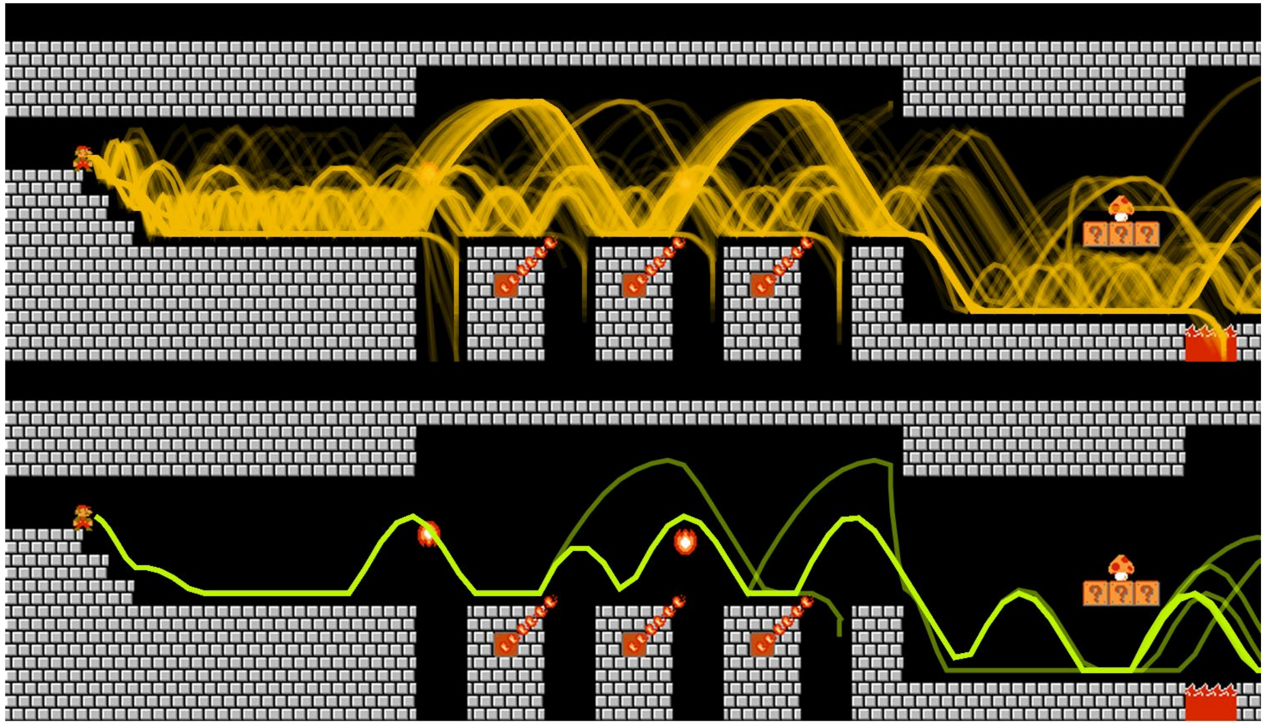
- Database Fuzzing
  - You can find how easy the solution is and how vulnerable the system is

  - Insight: The result should be identical if two statements are semantically equal
    - select * from db
    - select * from db where 1=1
    - select * from db where 1=1 or 1=0
    - …

  - EET (OSDI 24) finds 66 bugs in 5 DBMS
  - Yinyang (PLDI 20) finds 1500+ bugs in Z3 and CVC4
  - Try to use similar insights!

Fuzzing – Ideas/Cases

- Fun papers
  - IJON (S&P 20): manual annotations
  - Play Super Mario better than AFL



Fuzzing – Ideas/Cases

- Useful tools
  - AFL++: a better AFL that integrates recent research ideas
    - The process of fuzzing libpng

Fuzzing – Ideas/Cases

- Much more
  - How to fuzz more programs, or things?
    - Compilers
    - Firmware, Drivers
    - Hardware: CPU (S&P 21)
    - Machine learning systems
  - How to make the target program run faster?
    - Checkpoint/restore (Nyx-net, EuroSys 22; DistFuzz, NDSS 25)
  - More types of bugs
  - Integrate with other techniques
    - Static analysis, symbolic execution, formal methods
  - Checkers

Fuzzing

- Thanks for listening!

- Q&A

Don’t forget

- Add your profile to the lab website!