Directed Greybox Fuzzing

Marcel Böhme

Thuan Pham

M.-D. Nguyen

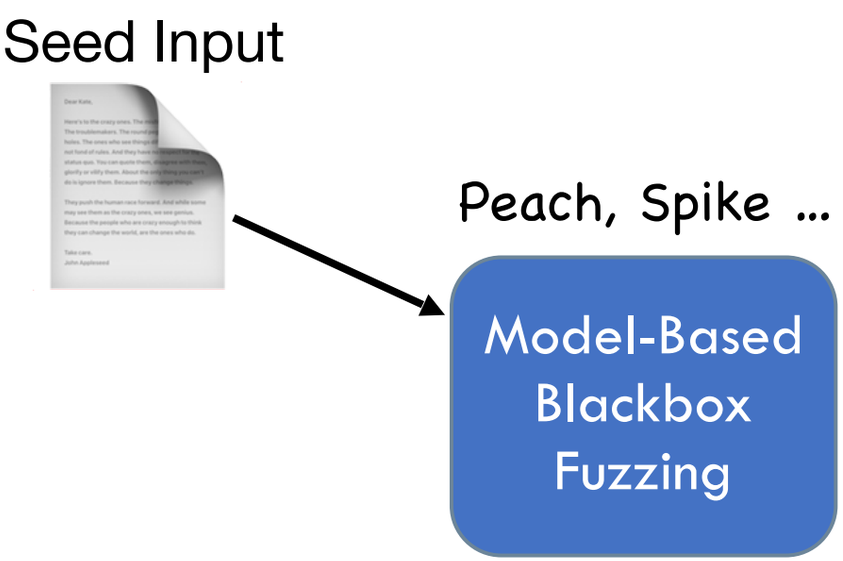
Abhik Roychoudhury

Motivation

- Automated vulnerability detection techniques
  1. **Blackbox Fuzzing** (no program analysis, no feedback)
  2. **Whitebox Fuzzing** (mostly program analysis)
  3. **Greybox Fuzzing** (no program analysis, but coverage feedback)

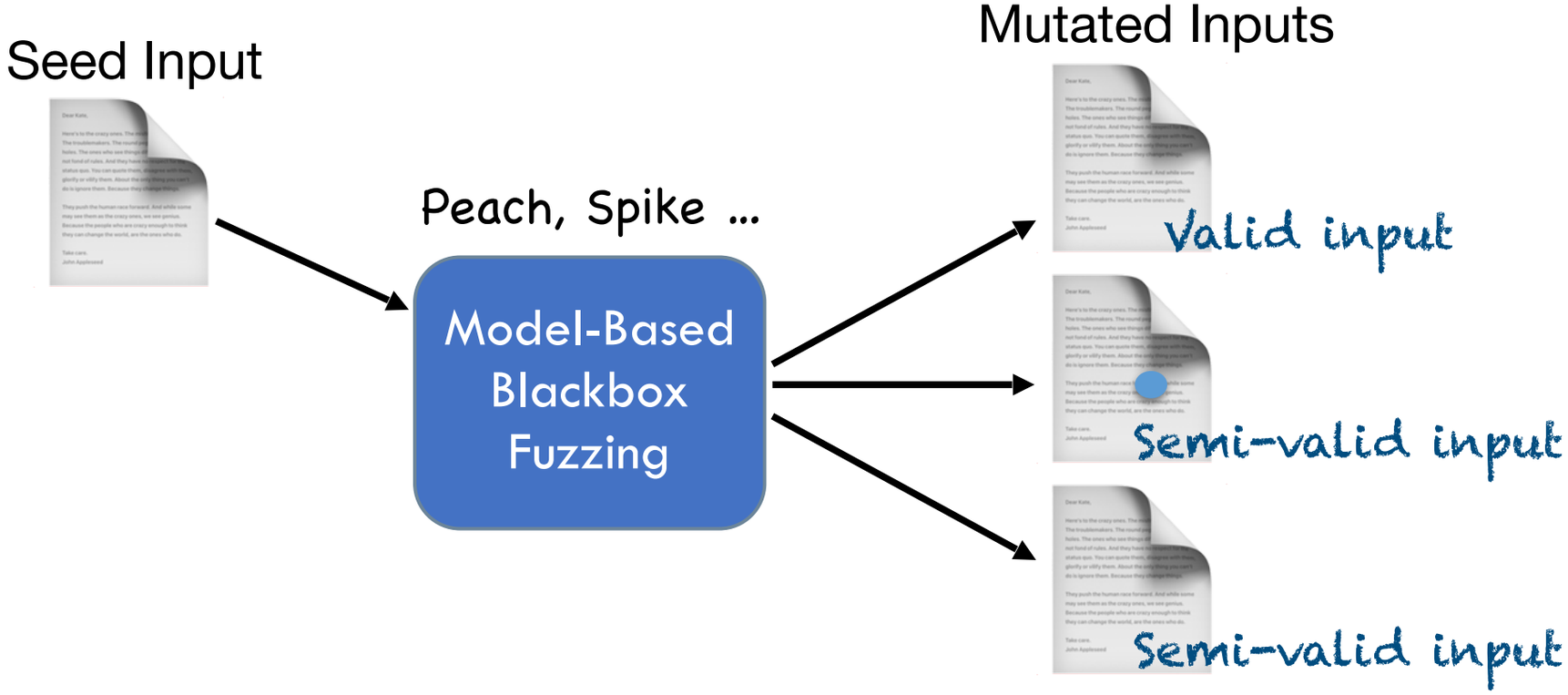
Motivation

- Automated vulnerability detection techniques
  1. **Blackbox Fuzzing** (no program analysis, no feedback)



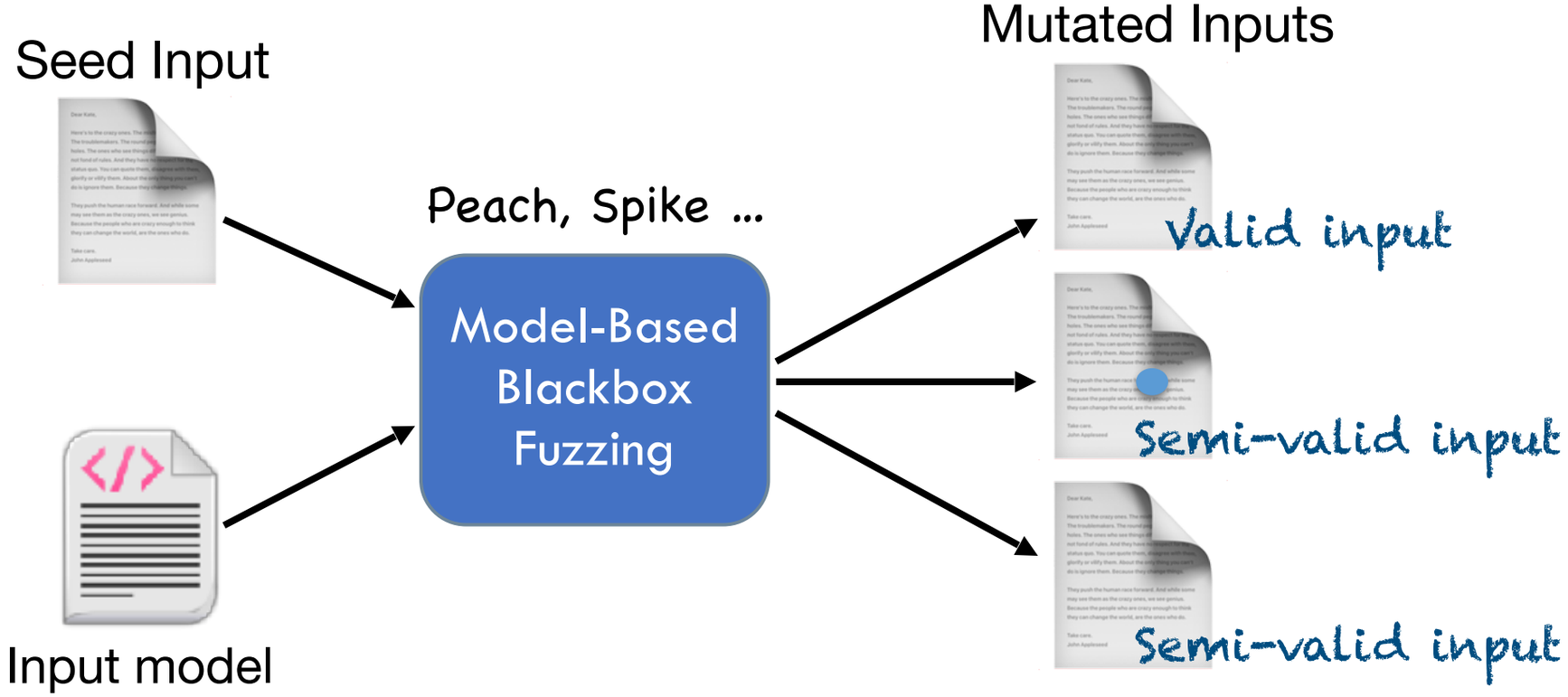
Motivation

- Automated vulnerability detection techniques
  1. **Blackbox Fuzzing** (no program analysis, no feedback)



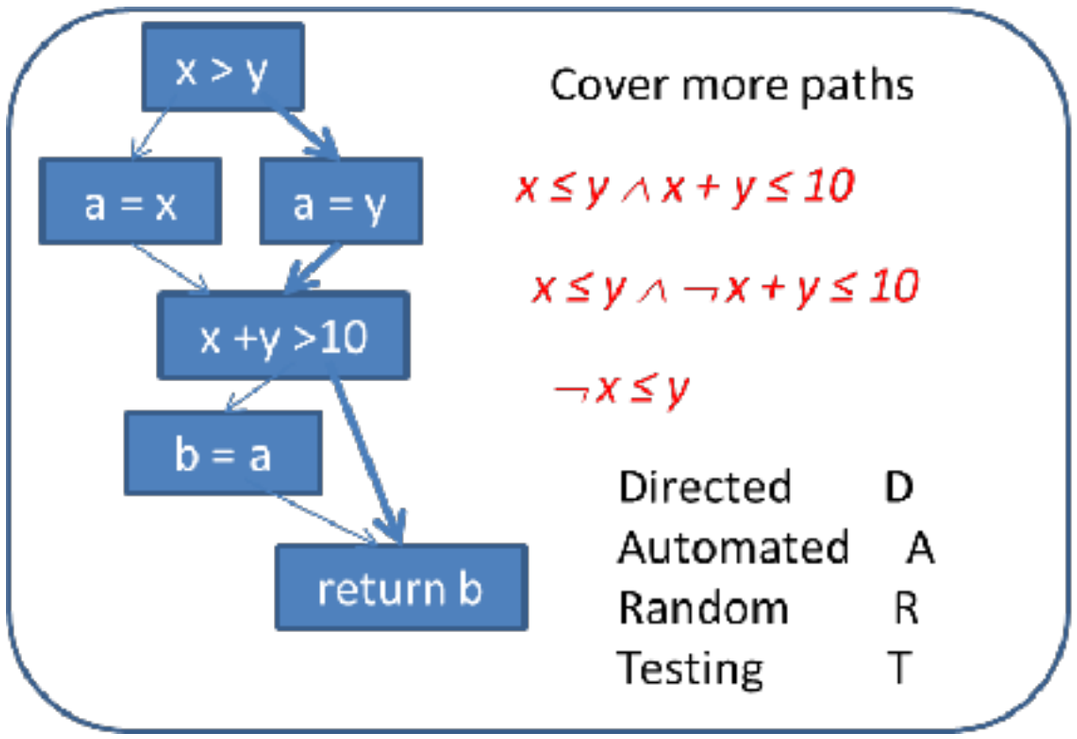
Motivation

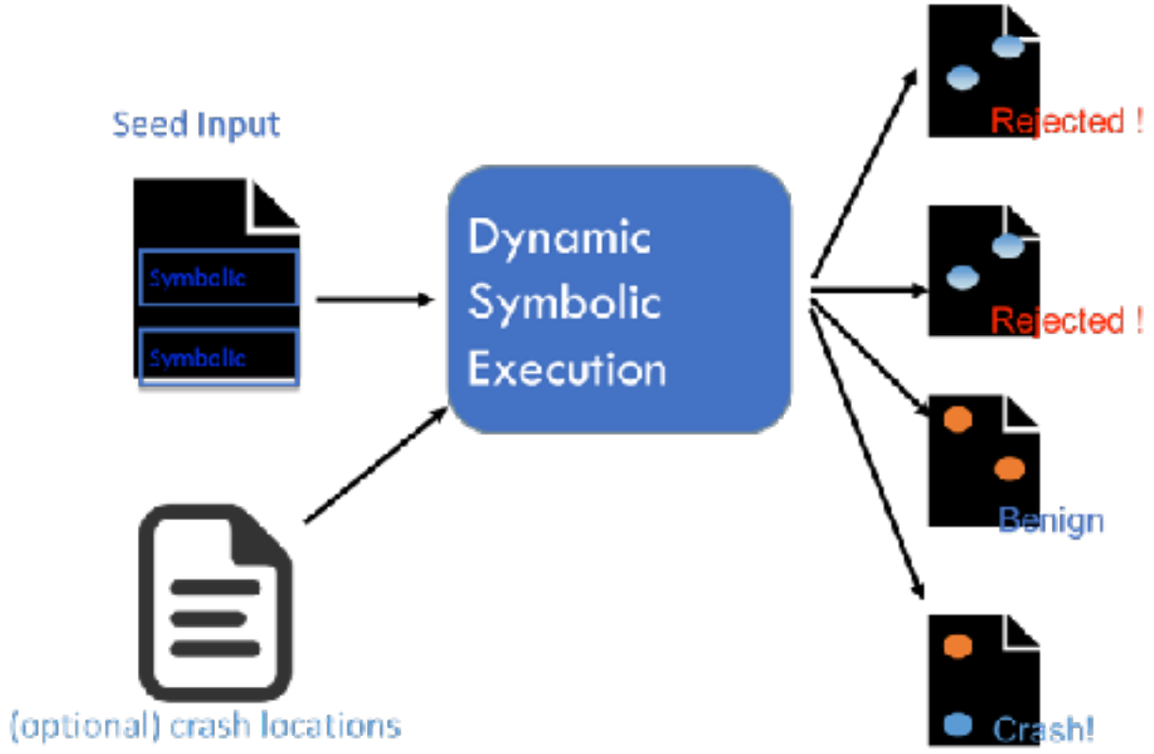
- Automated vulnerability detection techniques
  1. **Blackbox Fuzzing** (no program analysis, no feedback)



Motivation

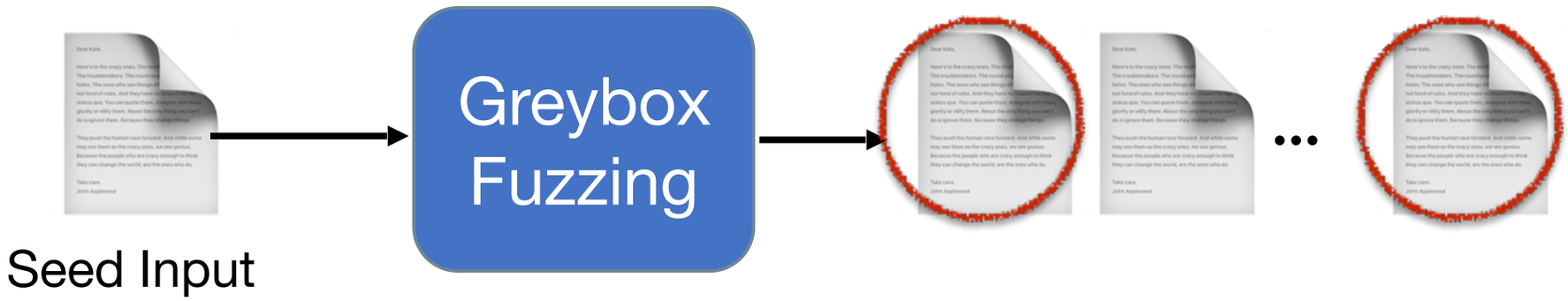
- Automated vulnerability detection techniques
  1. **Blackbox Fuzzing** (no program analysis, no feedback)
  2. **Whitebox Fuzzing** (mostly program analysis)





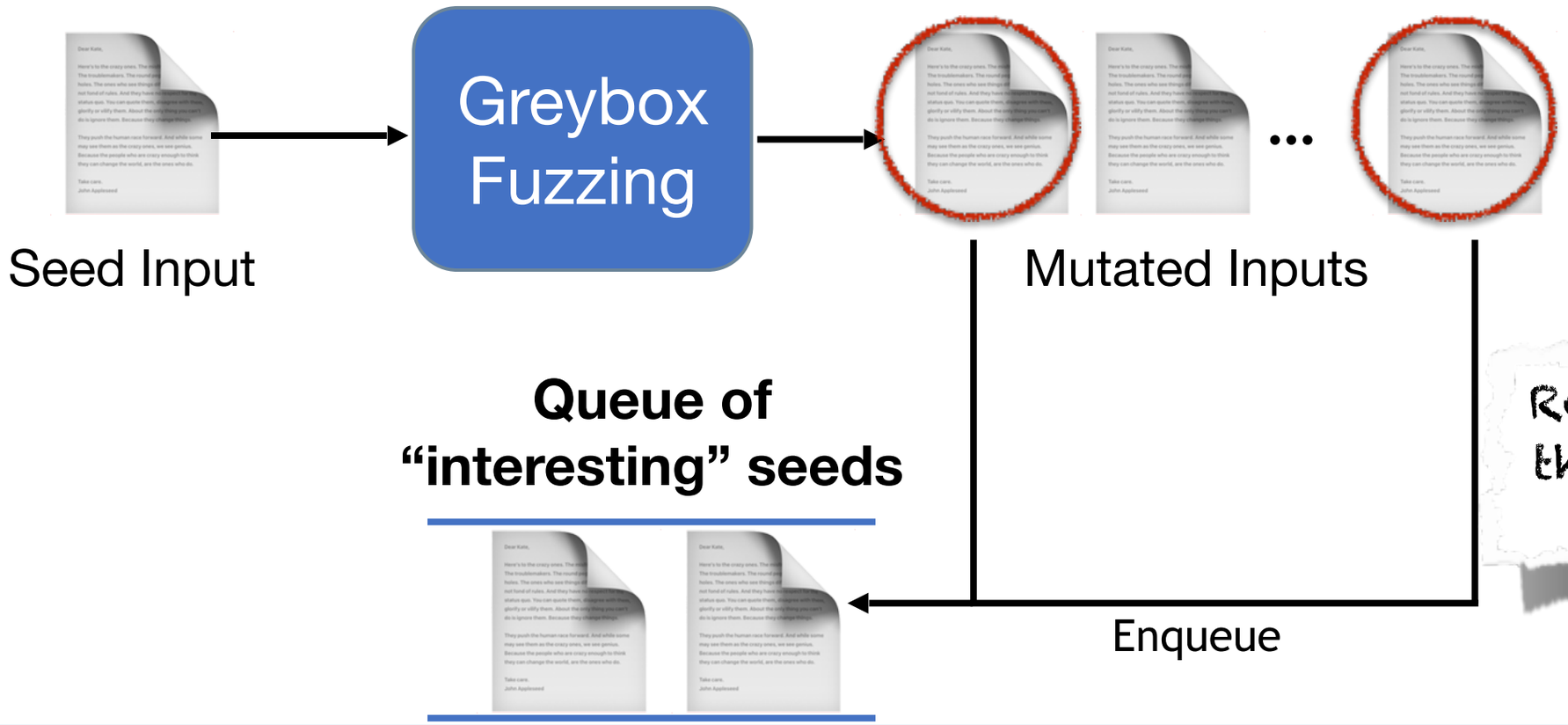
Motivation

- Automated vulnerability detection techniques
  1. **Blackbox Fuzzing** (no program analysis, no feedback)
  2. **Whitebox Fuzzing** (mostly program analysis)
  3. **Greybox Fuzzing** (no program analysis, but coverage feedback)



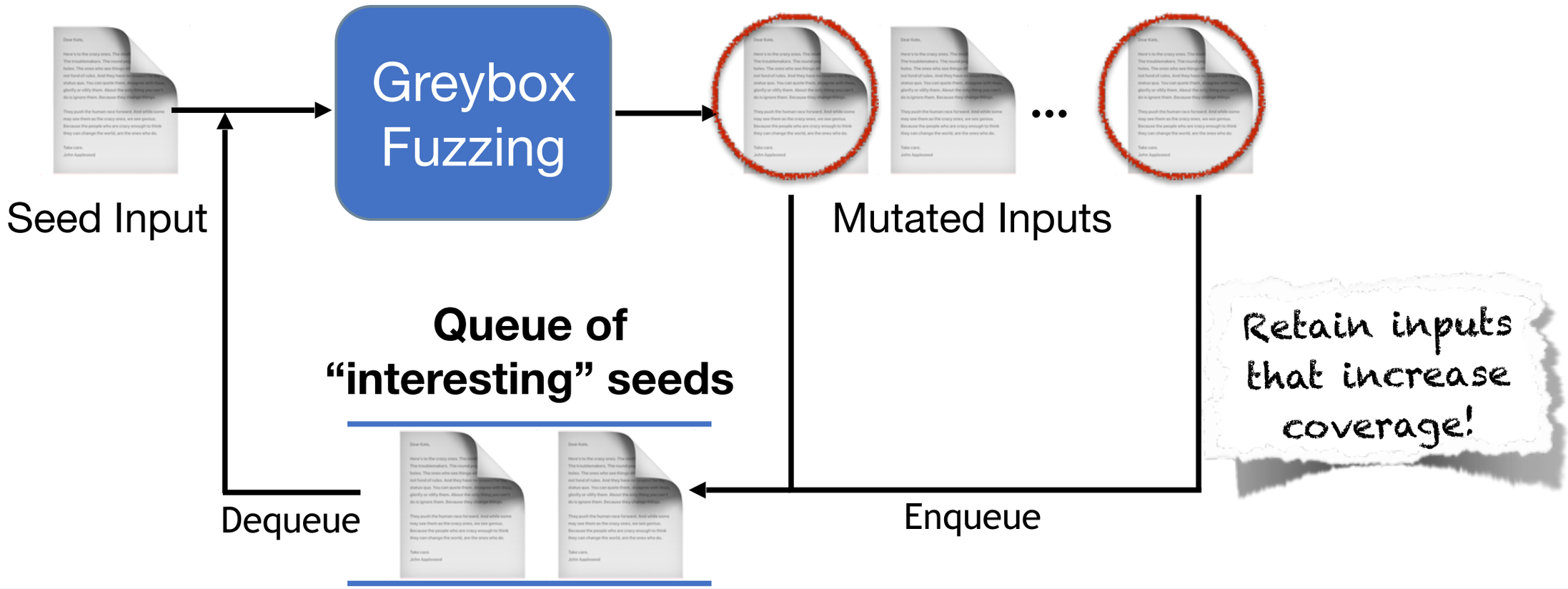
Motivation

- Automated vulnerability detection techniques
  1. **Blackbox Fuzzing** (no program analysis, no feedback)
  2. **Whitebox Fuzzing** (mostly program analysis)
  3. **Greybox Fuzzing** (no program analysis, but coverage feedback)



Motivation

- Automated vulnerability detection techniques
  1. **Blackbox Fuzzing** (no program analysis, no feedback)
  2. **Whitebox Fuzzing** (mostly program analysis)
  3. **Greybox Fuzzing** (no program analysis, but coverage feedback)



Motivation

- **Greybox Fuzzing** is frequently used
  - **State-of-the-art** in automated vulnerability detection
  - **Extremely efficient** coverage-based input generation
    - All program analysis before/at **instrumentation time**.
    - Start with a seed corpus, choose a seed file, **fuzz it**.
    - Add to corpus **only if new input increases coverage**.

Motivation

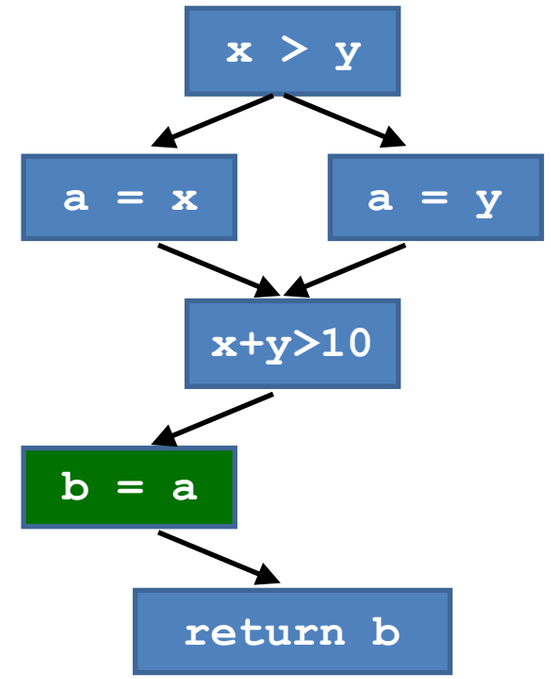
- **Greybox Fuzzing** is frequently used
  - **State-of-the-art** in automated vulnerability detection
  - **Extremely efficient** coverage-based input generation
    - All program analysis before/at **instrumentation time**.
    - Start with a seed corpus, choose a seed file, **fuzz it**.
    - Add to corpus **only if new input increases coverage**.
- **Cannot be directed!**

Motivation

- **Directed Fuzzing** has many applications
  - **Patch Testing**: reach changed statements
  - **Crash Reproduction**: exercise stack trace
  - **SA Report Verification**: reach “dangerous” location
  - **Information Flow Detection**: exercise source-sink pairs

Motivation

- **Directed Fuzzing**: classical **constraint satisfaction prob**.
  - **Program analysis** to identify **program paths** that reach given program locations.
  - **Symbolic Execution** to derive **path conditions** for any of the identified paths.
  - **Constraint Solving** to find an **input** that
    - **satisfies the path condition** and thus
    - **reaches a program location** that was given.



$\varphi_1 = (x>y) \wedge (x+y>10)$

$\varphi_2 = \neg(x>y) \wedge (x+y>10)$

Motivation

- **Directed Fuzzing**: classical **constraint satisfaction prob**.
  - **Program analysis** to identify **program paths** that reach given program locations.
  - **Symbolic Execution** to derive **path conditions** for any of the identified paths.
  - **Constraint Solving** to find an **input** that
    - **satisfies the path condition** and thus
    - **reaches a program location** that was given.

Requires heavy-weight machinery!

$\varphi_1 = (x>y) \wedge (x+y>10)$

$\varphi_2 = \neg(x>y) \wedge (x+y>10)$

Overview

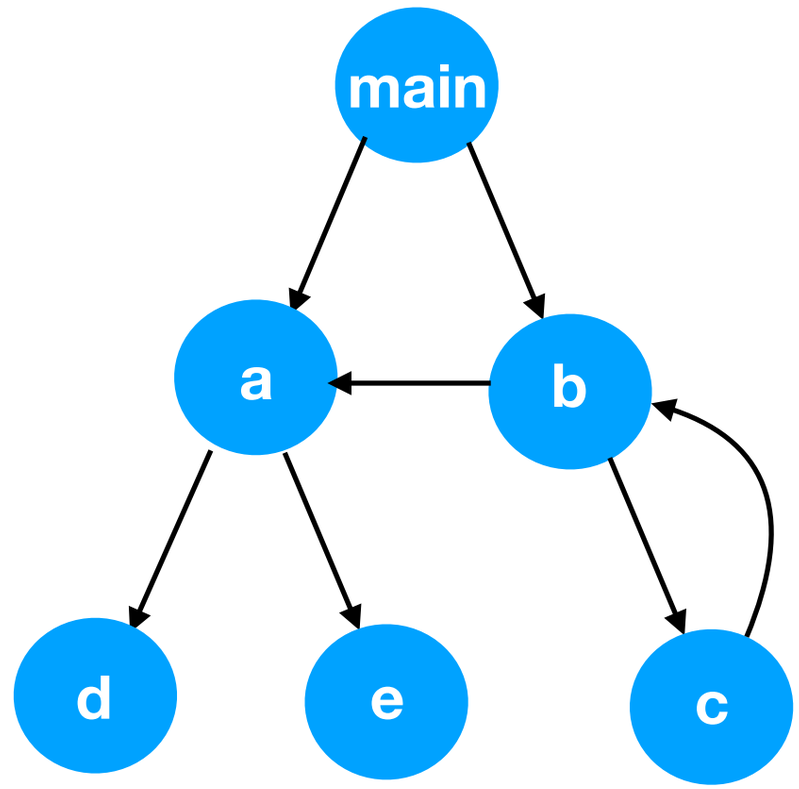
- **Directed Fuzzing** as **optimisation problem**!
    1. Instrumentation Time:
        1. Extract **call graph** (CG) and **control-flow graphs** (CFGs).
        2. For each **BB**, compute **distance** to target locations.
        3. Instrument program to **aggregate distance values**.

Overview

- **Directed Fuzzing** as **optimisation problem**!
  1. Instrumentation Time:
     1. Extract **call graph** (CG) and **control-flow graphs** (CFGs).
     2. For each **BB**, compute **distance** to target locations.
     3. Instrument program to **aggregate distance values**.
  2. Runtime, **for each input**
     1. collect coverage and distance **information**, and
     2. decide **how long to be fuzzed** based on distance.
        - If input is **closer** to the targets, it is fuzzed for **longer**.
        - If input is **further away** from the targets, it is fuzzed for **shorter**.

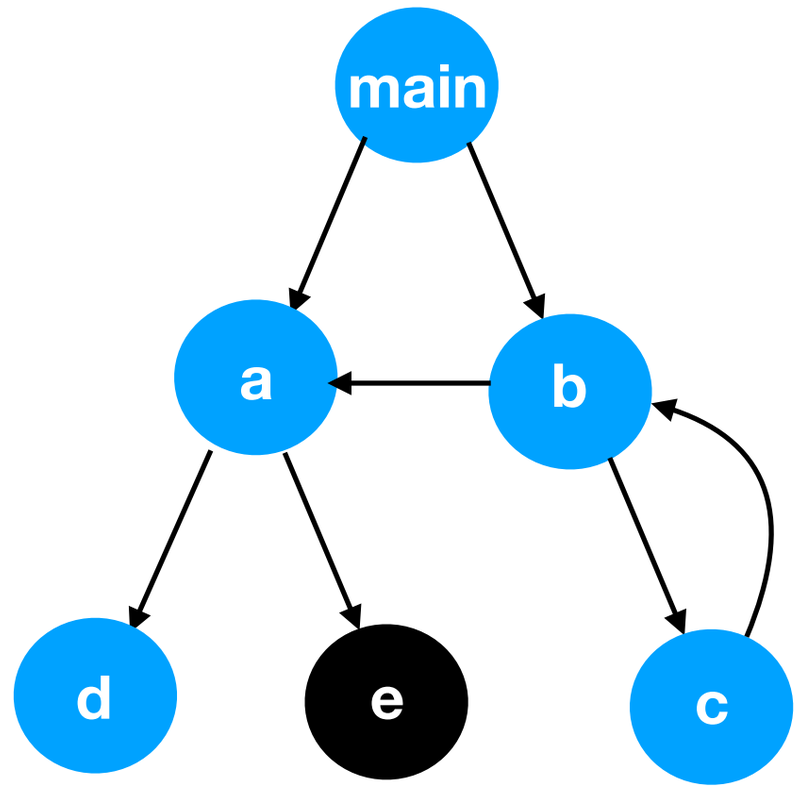
Instrumentation

- **Function-level target distance** using call graph (CG)



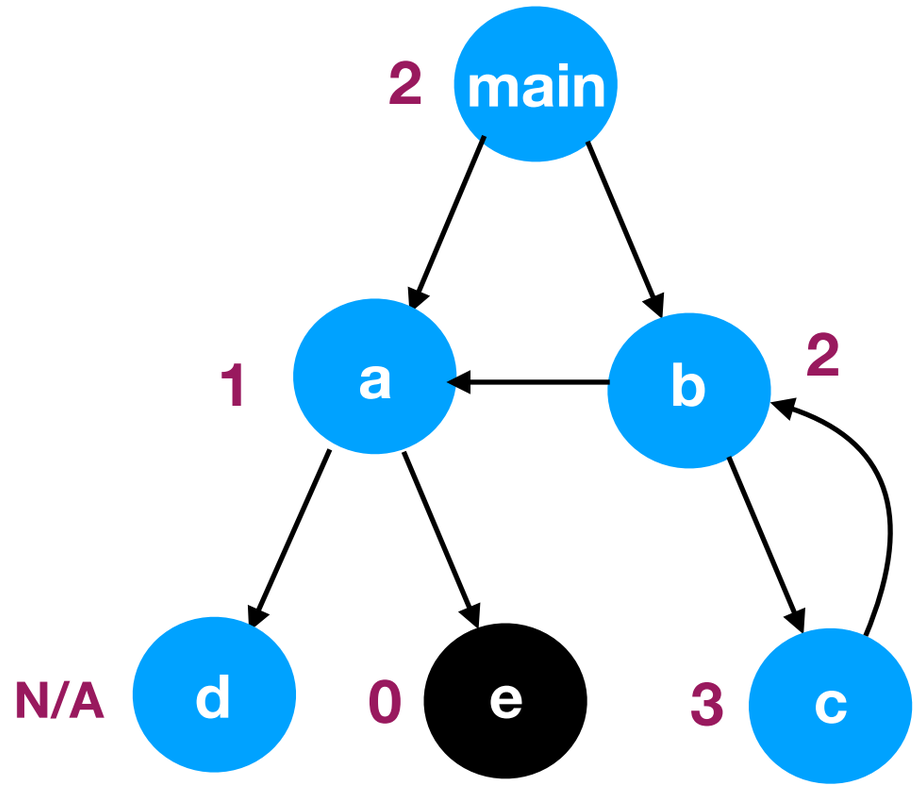
Instrumentation

- **Function-level target distance** using call graph (CG)
  1. Identify **target functions** in CG



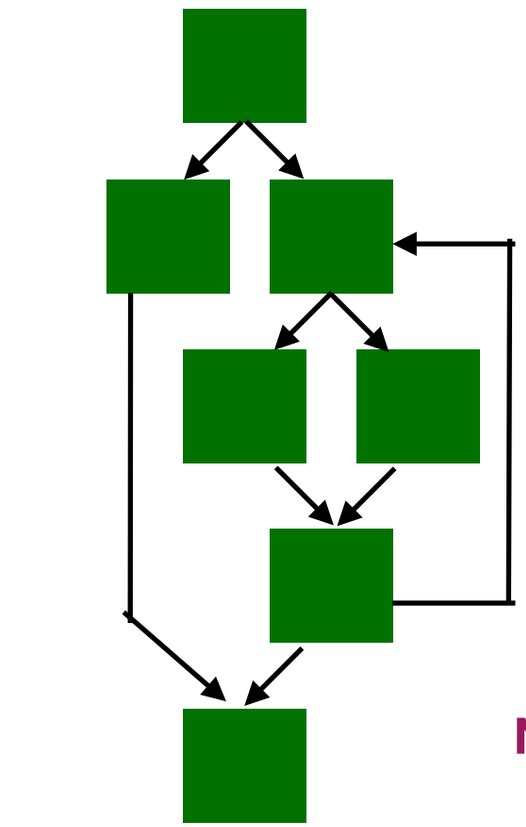
Instrumentation

- **Function-level target distance** using call graph (CG)
  1. Identify **target functions** in CG
  2. For each **function**, compute the harmonic mean of the **length** of the shortest path to targets

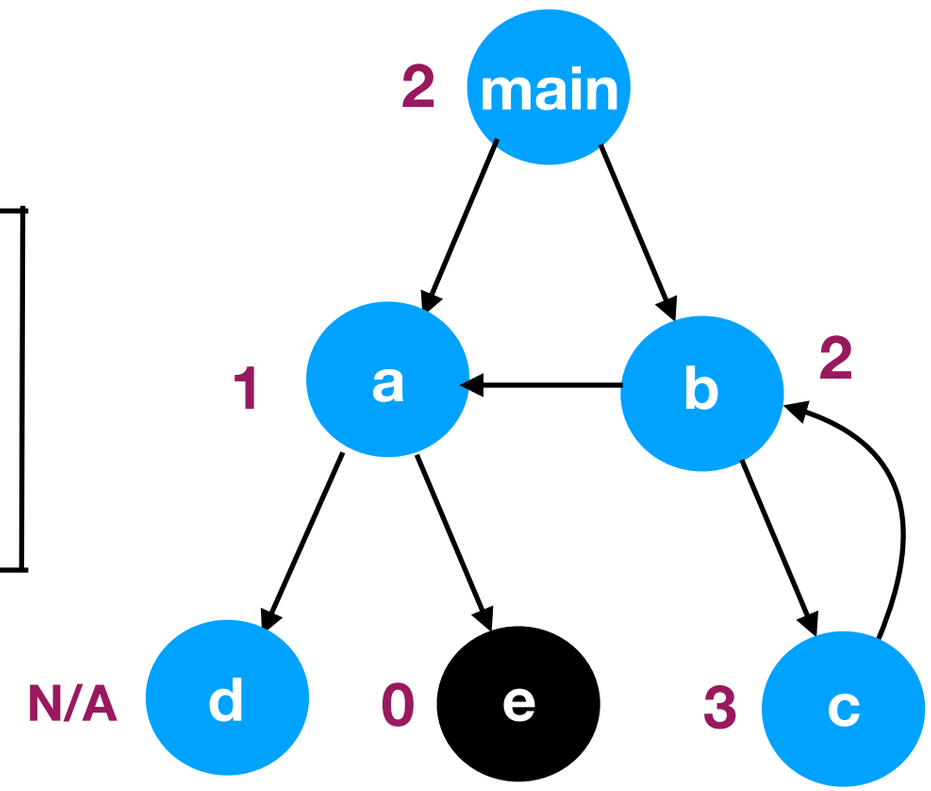


Instrumentation

- **Function-level target distance** using call graph (CG)
- **BB-level target distance** using CFG

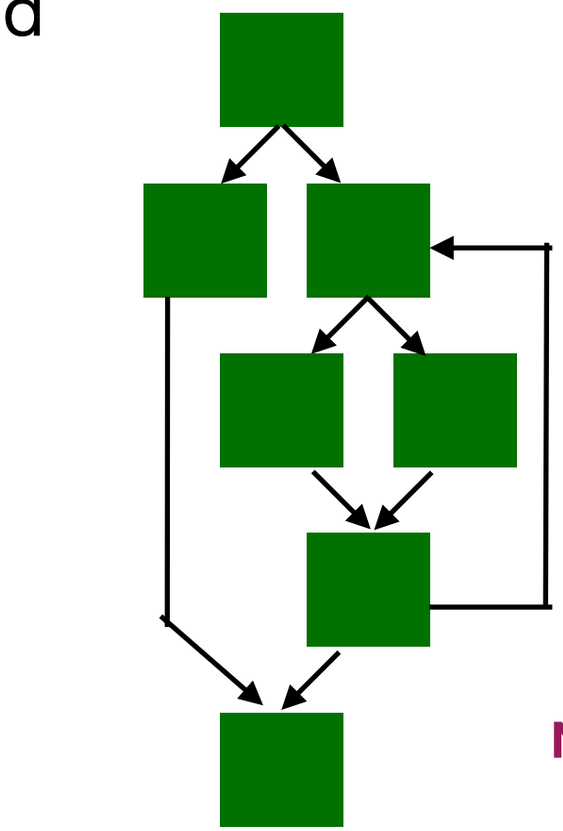


CFG for function b

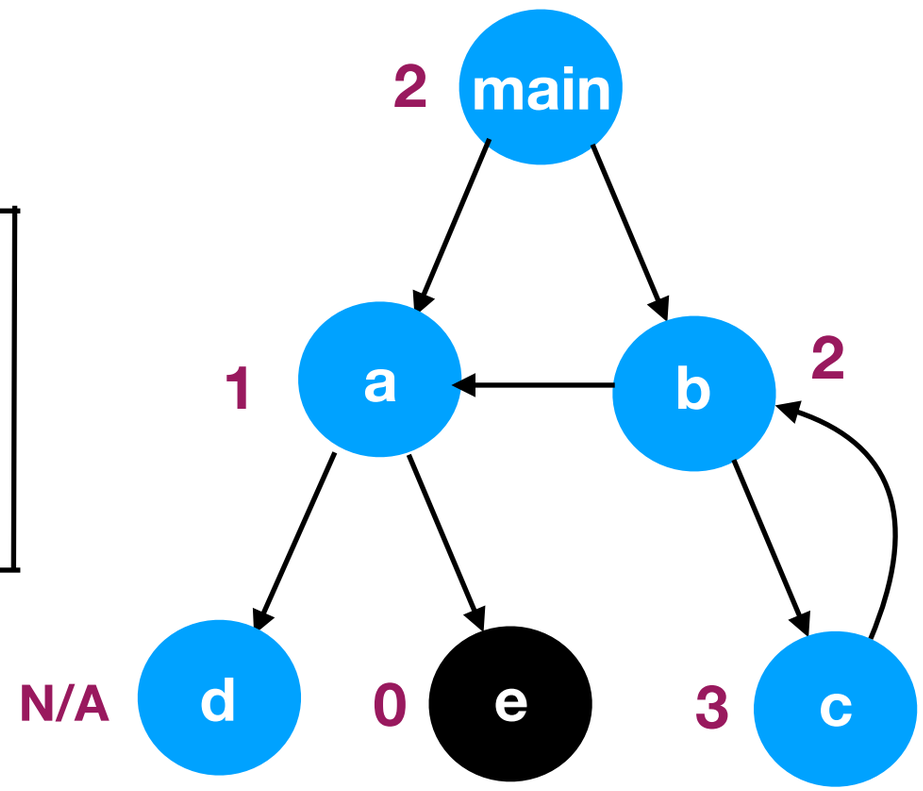


Instrumentation

- **Function-level target distance** using call graph (CG)
- **BB-level target distance** using control-flow graph (CFG)
  1. Identify **target BBs** and assign **distance 0** (none in function b)

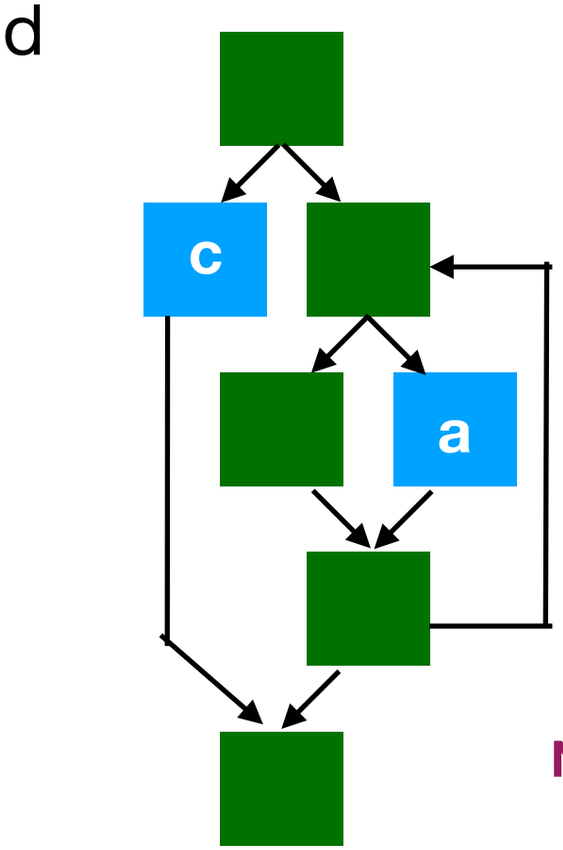


CFG for function b

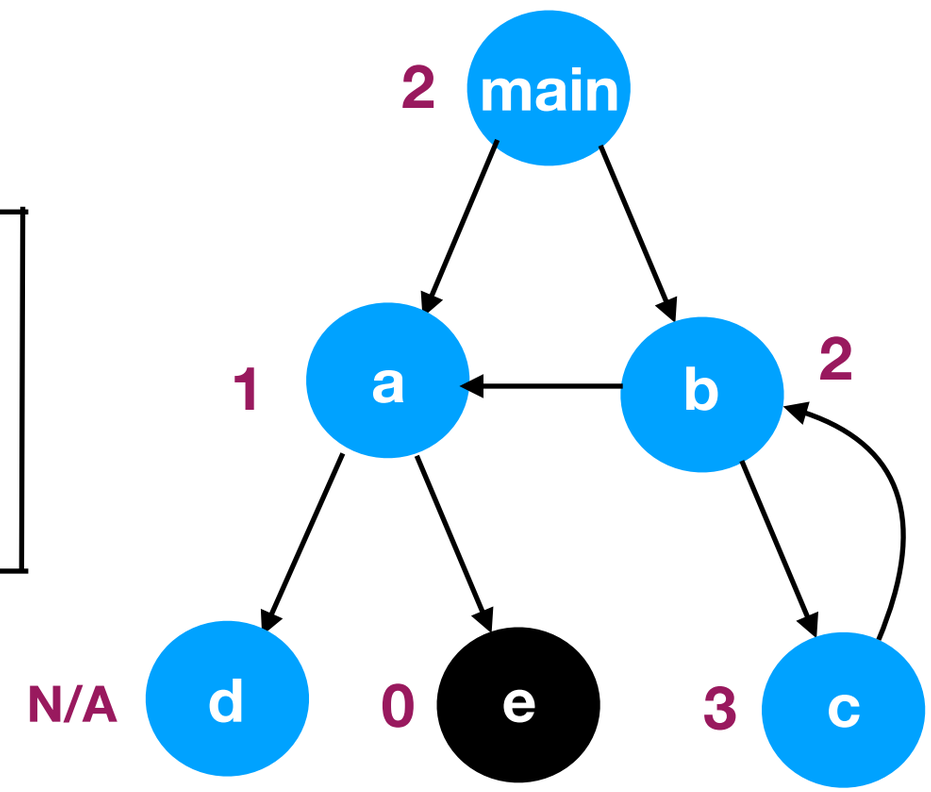


Instrumentation

- **Function-level target distance** using call graph (CG)
- **BB-level target distance** using control-flow graph (CFG)
  1. Identify **target BBs** and assign **distance 0**
  2. Identify BBs that call **functions**

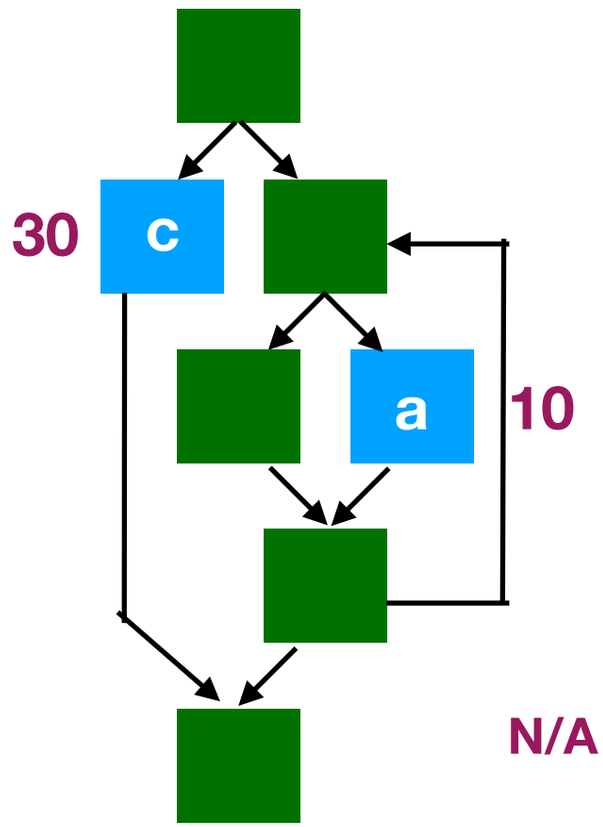


CFG for function b

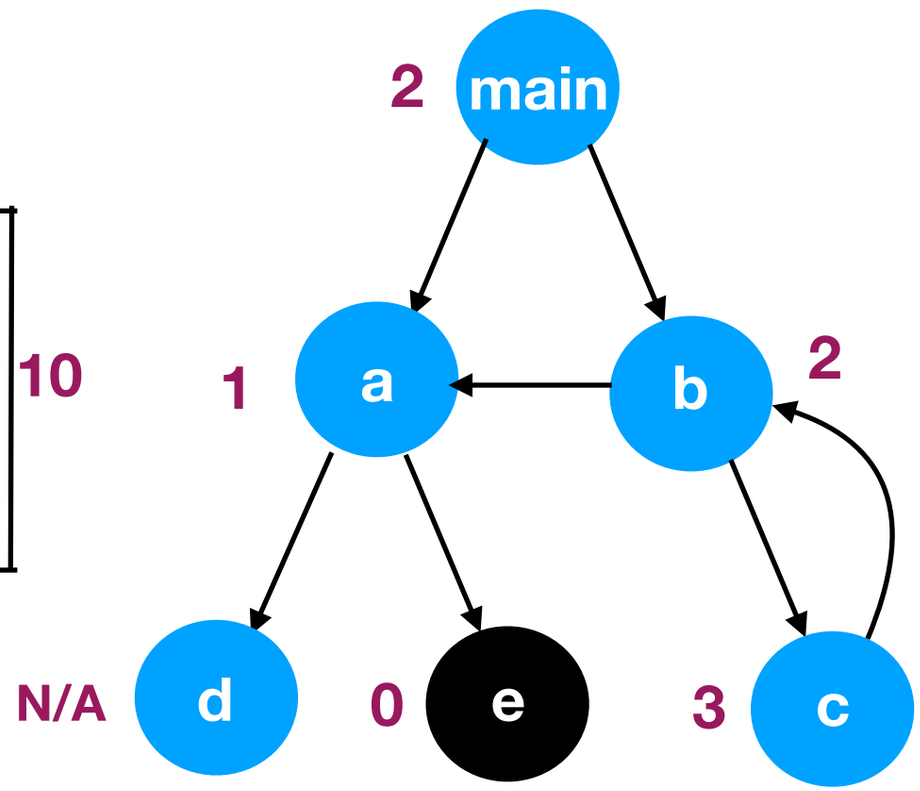


Instrumentation

- **Function-level target distance** using call graph (CG)
- **BB-level target distance** using control-flow graph (CFG)
  1. Identify **target BBs** and assign **distance 0**
  2. Identify BBs that call **functions** and assign **10*FLTD**

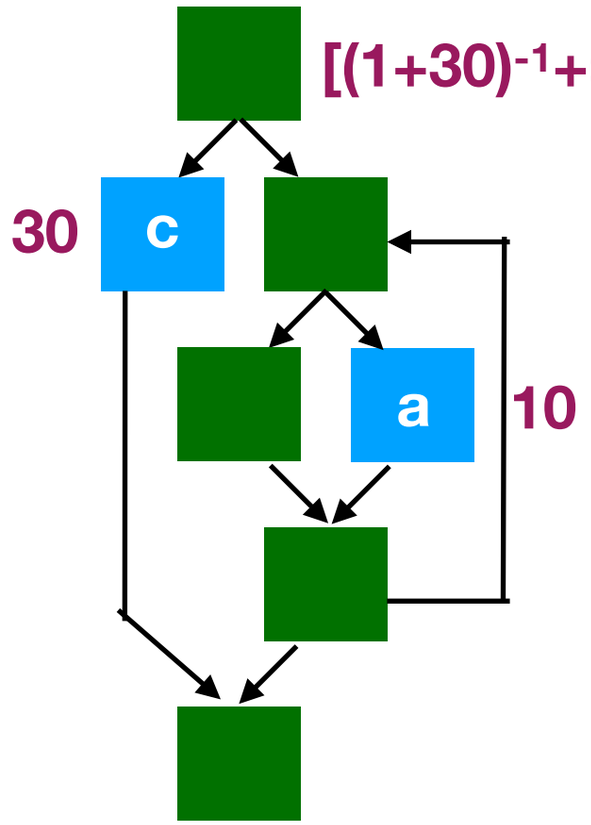


CFG for function b



Instrumentation

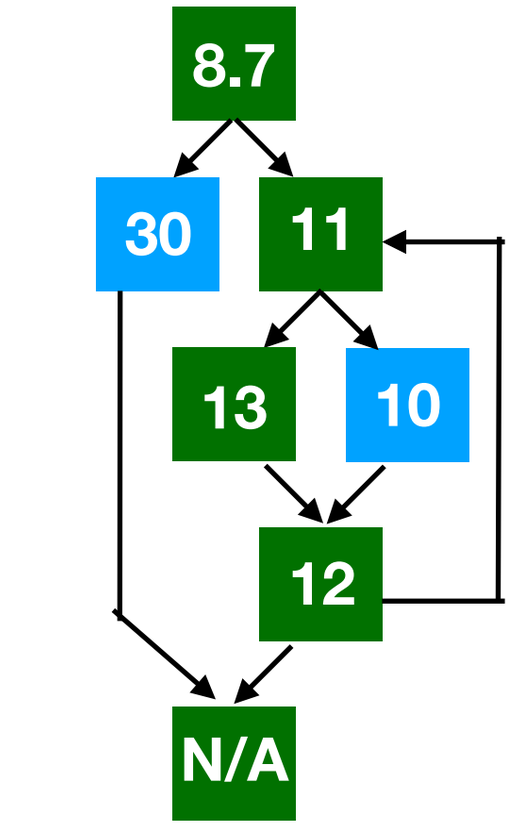
- **Function-level target distance** using call graph (CG)
- **BB-level target distance** using control-flow graph (CFG)
  1. Identify **target BBs** and assign **distance 0**
  2. Identify BBs that call **functions** and assign **10*FLTD**
  3. For **each BB**, compute harmonic mean of (length of shortest path to any function-calling BB + 10*FLTD).



CFG for function b

Instrumentation

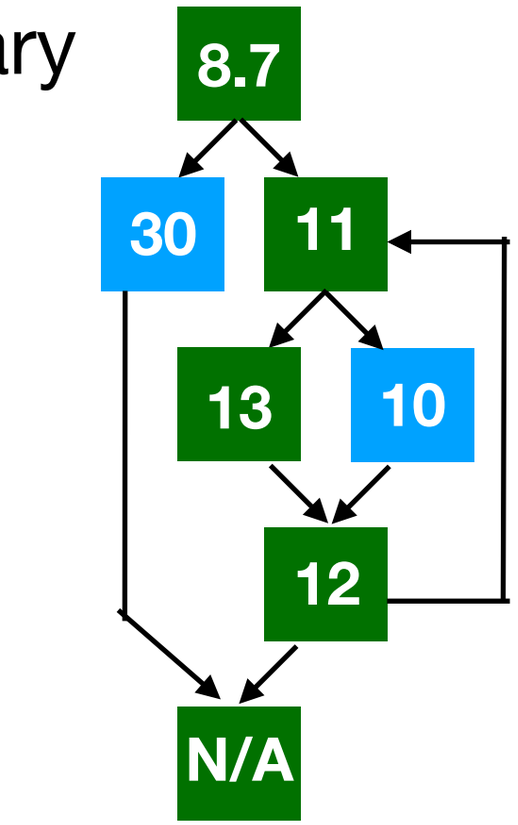
- **Function-level target distance** using call graph (CG)
- **BB-level target distance** using control-flow graph (CFG)
  1. Identify **target BBs** and assign **distance 0**
  2. Identify BBs that call **functions** and assign **10*FLTD**
  3. For **each BB**, compute harmonic mean of (length of shortest path to any function-calling BB + 10*FLTD).



CFG for function b

Runtime

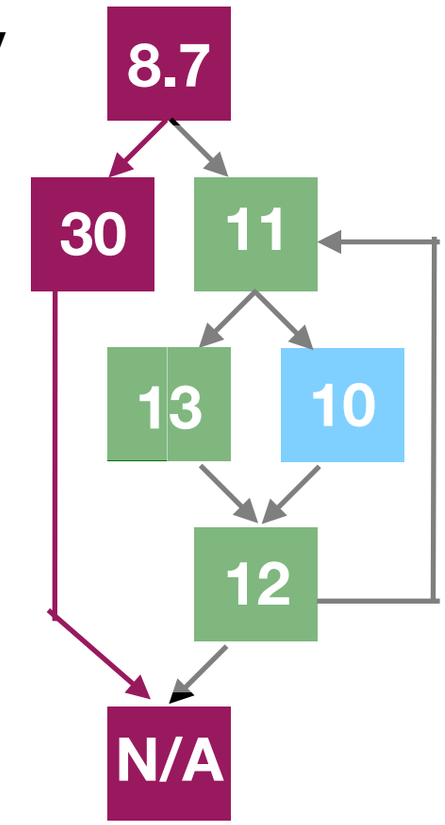
- **Function-level target distance** using call graph (CG)
- **BB-level target distance** using control-flow graph (CFG)
- **Seed distance** from instrumented binary



CFG for function b

Runtime

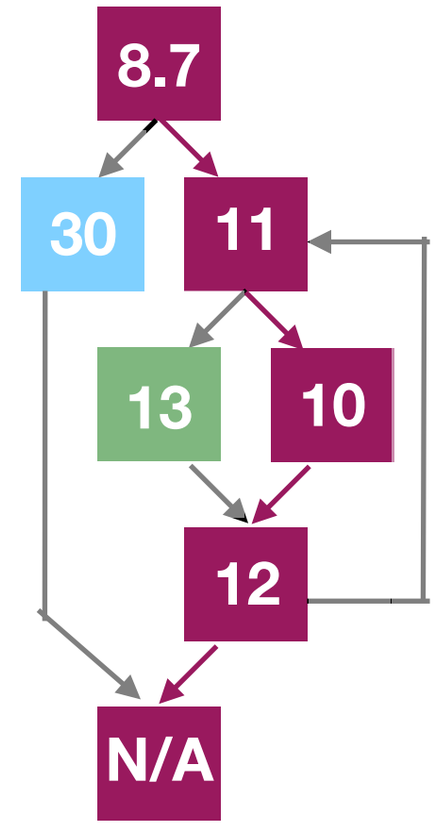
- **Function-level target distance** using call graph (CG)
- **BB-level target distance** using control-flow graph (CFG)
- **Seed distance** from instrumented binary
  - Two 64-bit shared memory entries
    - Aggregated BB-level distance values
    - Number of executed BBs



Seed Distance: 19.4
= (8.7+30)/2

Runtime

- **Function-level target distance** using call graph (CG)
- **BB-level target distance** using control-flow graph (CFG)
- **Seed distance** from instrumented binary
  - Two 64-bit shared memory entries
    - Aggregated BB-level distance values
    - Number of executed BBs

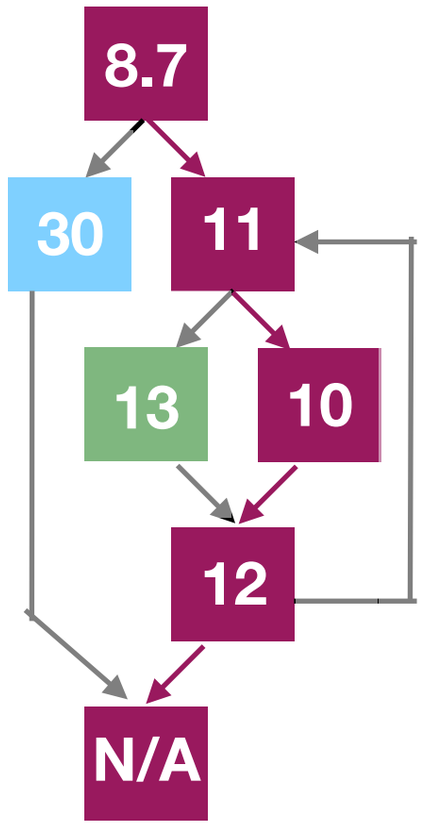


Seed Distance: 10.4
= (8.7+11+10+12)/4

Runtime

- **Function-level target distance** using call graph (CG)
- **BB-level target distance** using control-flow graph (CFG)
- **Seed distance** from instrumented binary
  - Two 64-bit shared memory entries
    - Aggregated BB-level distance values
    - Number of executed BBs

Now that we know how to compute seed distance,
– let's minimise it! –



Seed Distance: 10.4
= (8.7+11+10+12)/4

Directed Fuzzing as Optimisation Problem

NUS National University of Singapore

- **Background**: Coverage-based Greybox Fuzzing

Seed File

Directed Fuzzing as Optimisation Problem

- **Background**: Coverage-based Greybox Fuzzing

Mutation Operators:

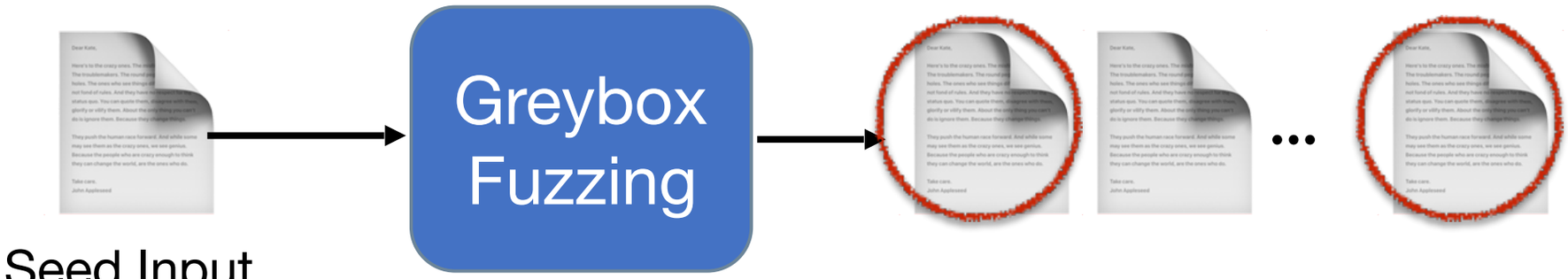
- Bitflips
- Boundary Values
  `(0,1,-1,INT_MAX,INT_MIN)`
- Simple arithmetics
  `(add/subtract 1)`
- Block deletion
- Block insertion

Seed File

Directed Fuzzing as Optimisation Problem

NUS National University of Singapore

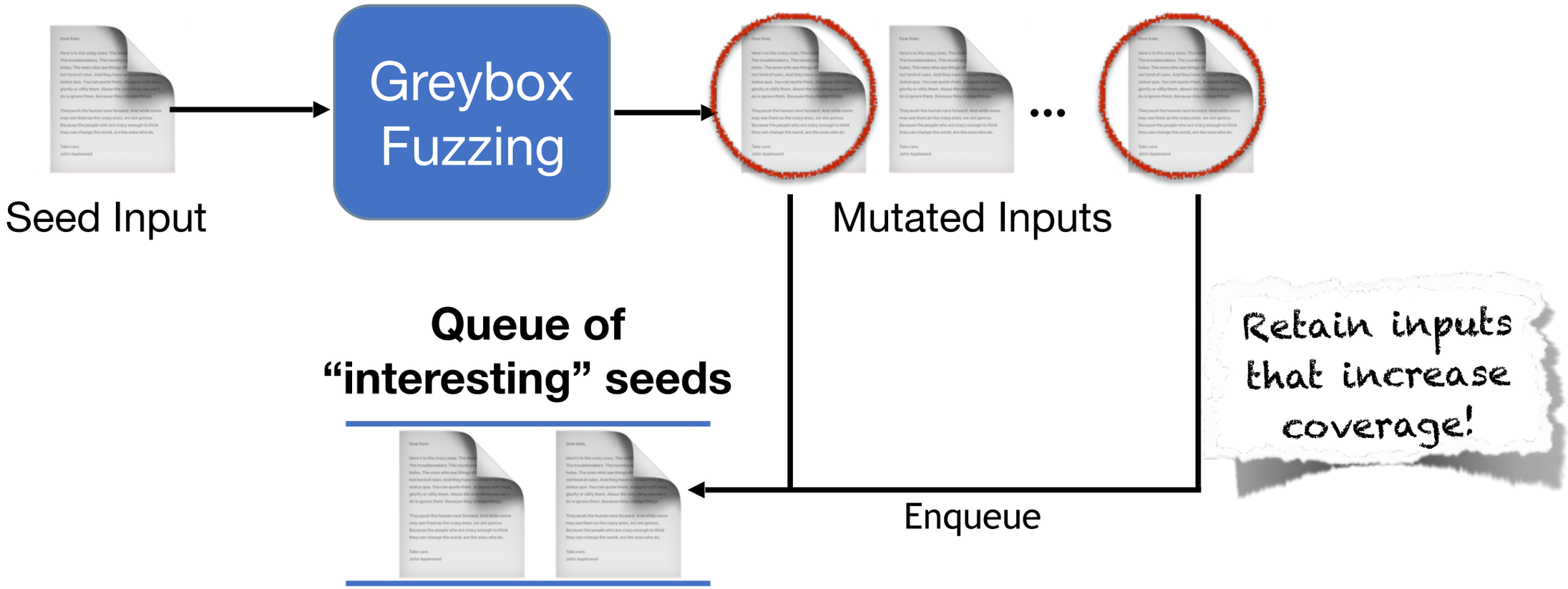
- **Background**: Coverage-based Greybox Fuzzing



Directed Fuzzing as Optimisation Problem

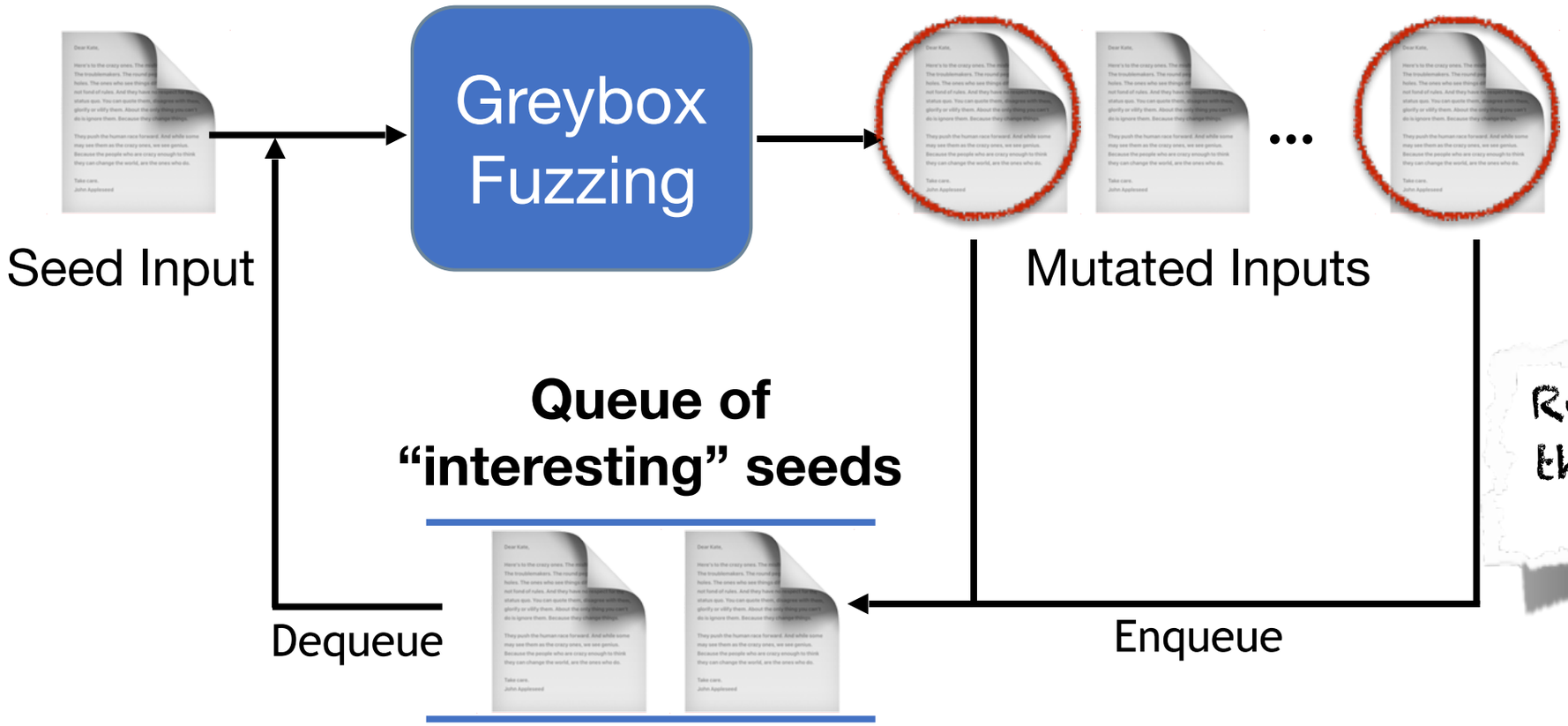
NUS National University of Singapore

- **Background**: Coverage-based Greybox Fuzzing



Directed Fuzzing as Optimisation Problem

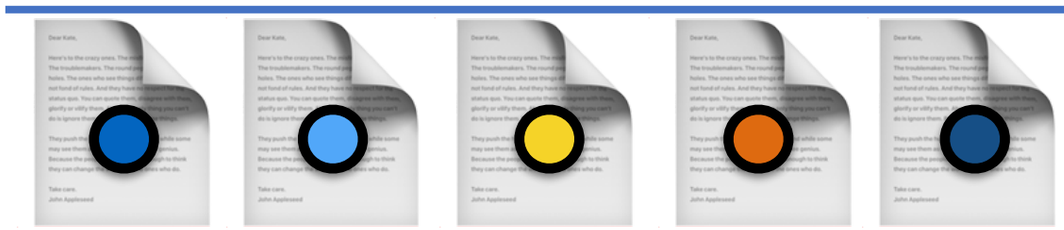
- **Background**: Coverage-based Greybox Fuzzing



Directed Fuzzing as Optimisation Problem

- **Background**: Coverage-based Greybox Fuzzing

Queue of "interesting" seeds

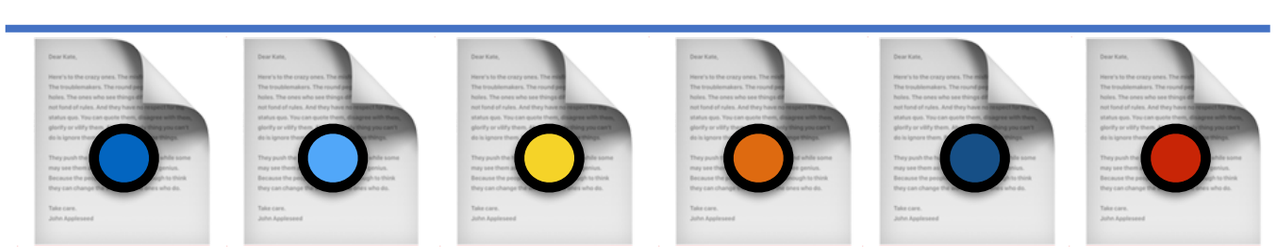


low energy
- generated less test inputs!

Directed Fuzzing as Optimisation Problem

- **Background**: Coverage-based Greybox Fuzzing

Queue of "interesting" seeds



high energy
- generated more test inputs!

Directed Fuzzing as Optimisation Problem

- **Background**: Coverage-based Greybox Fuzzing
  - Seed's energy:
    - Number of inputs generated when chosen for fuzzing
    - Local property: each seed has its own energy
  - Power schedule:
    - Assigns energy to seeds according to a pre-defined formula

★ **Boosted** Greybox Fuzzing (**AFLFast** CCS'16)

- Assign **more energy** to seeds exercising **low-frequency paths.**

★ **Directed** Greybox Fuzzing (**AFLGo** CCS'17)

- Assign **more energy** to seeds that a **closer to the given targets**!

Directed Fuzzing as Optimisation Problem

- **Background**: Coverage-based Greybox Fuzzing
  - Seed's energy:
    - Number of inputs generated when chosen for fuzzing
    - Local property: each seed has its own energy
  - Power schedule:
    - Assigns energy to seeds according to a pre-defined formula

★ **Boosted** Greybox Fuzzing (**AFLFast** CCS'16)

- Assign **more energy** to seeds exercising **low-frequency paths.**

★ **Directed** Greybox Fuzzing (**AFLGo** CCS'17)

- Assign **more energy** to seeds that a **closer to the given targets**!

Directed Fuzzing as Optimisation Problem

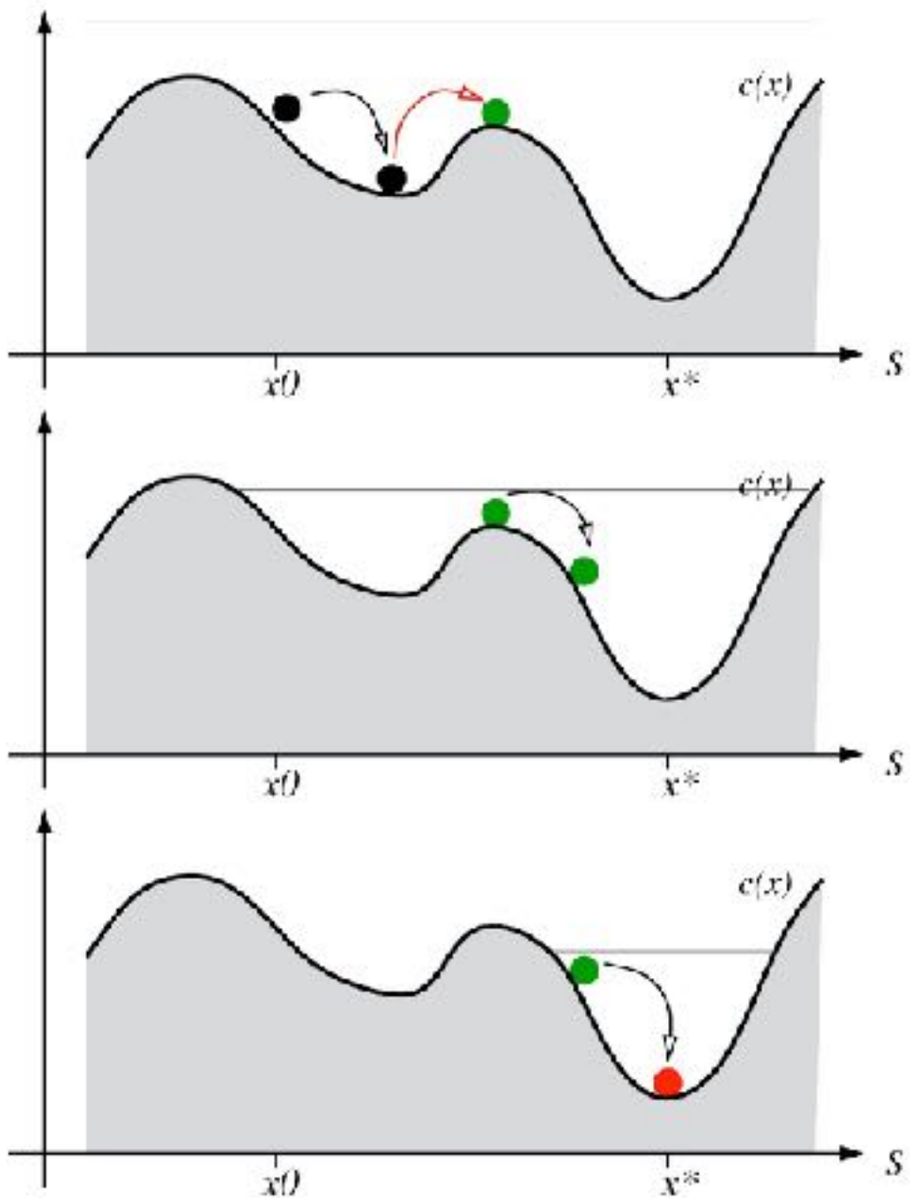
- **Directed** Greybox Fuzzing
  - Assign **more energy** to seeds that a **closer** to the given targets!
  - **Problem** (Stochastic Gradient Descent)
    - If we **always** assign **more energy** to **closer** seeds,
      we **typically** reach only a **local** minimum,
      but **never** a **global** minimum distance!
- **Solution** (Simulated Annealing)
  - **Sometimes** assign **more energy** to **further-away** seeds!
  - Approaches **global** minimum distance.

Directed Fuzzing as Optimisation Problem

- **Directed** Greybox Fuzzing
  - Assign **more energy** to seeds that a **closer** to the given targets!
  - **Problem** (Stochastic Gradient Descent)
    - If we **always** assign **more energy** to **closer** seeds,
      we **typically** reach only a **local** minimum,
      but **never** a **global** minimum distance!
- **Solution** (Simulated Annealing)
  - **Sometimes** assign **more energy** to **further-away** seeds!
  - Approaches **global** minimum distance.

Directed Fuzzing as Optimisation Problem

- Simulated Annealing (SA)
  - **Exploration** phase:
    - Energy of **closer** seeds similar to energy of **further-away** seeds
  - **Exploitation** phase:
    - Energy of **closer** seeds is assigned to be **higher** and higher
    - Energy of **further-away** seeds is assigned to be **lower** and lower
- We are increasing the “importance” of seed distance over time.

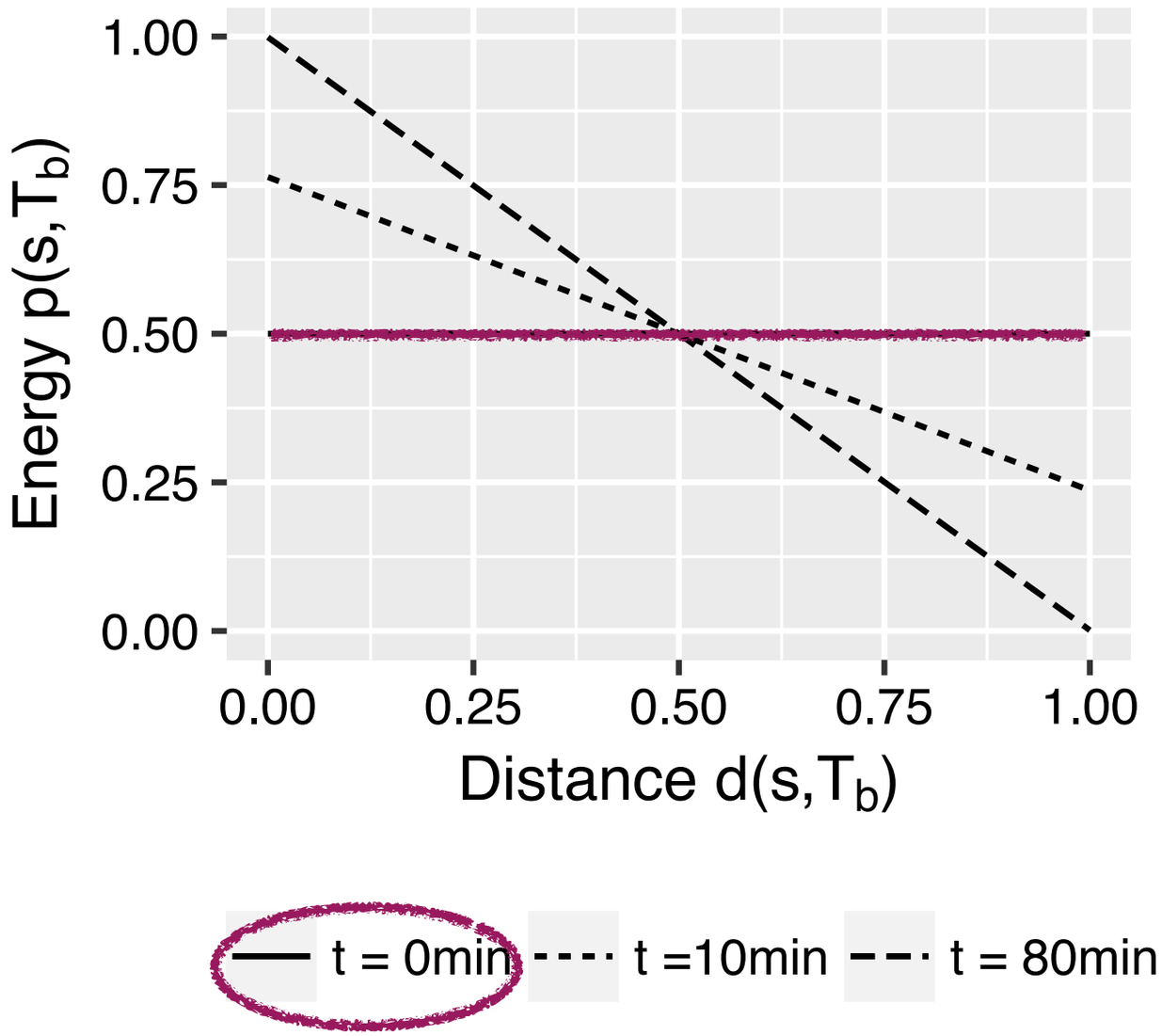


Directed Fuzzing as Optimisation Problem

- Simulated Annealing (SA)
  - Annealing from **metallurgy**: control the cooling of material to reduce defects (e.g., cracks or bubbles) in the material.
  - **Temperature** T ∈ [0,1] specifies "importance" of distance.
    - At T=1, **exploration** (normal AFL)
    - At T=0, **exploitation** (gradient descent)
- **Cooling schedule** controls (global) temperature
  - Classically, exponential cooling.

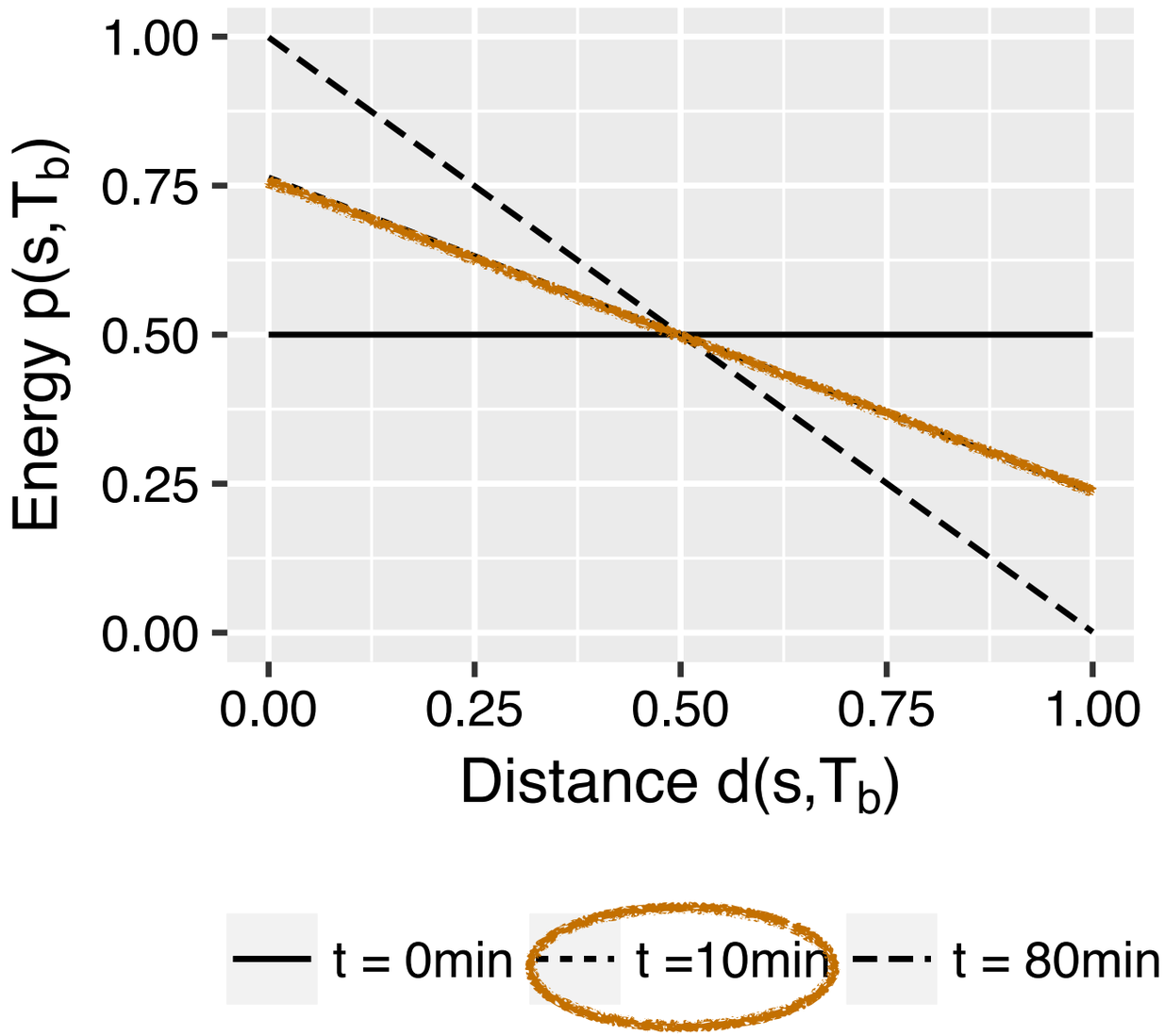
Directed Fuzzing as Optimisation Problem

- Integrating Simulated Annealing as power schedule
  - In the beginning (t = 0min), assign the **same energy to all seeds**.



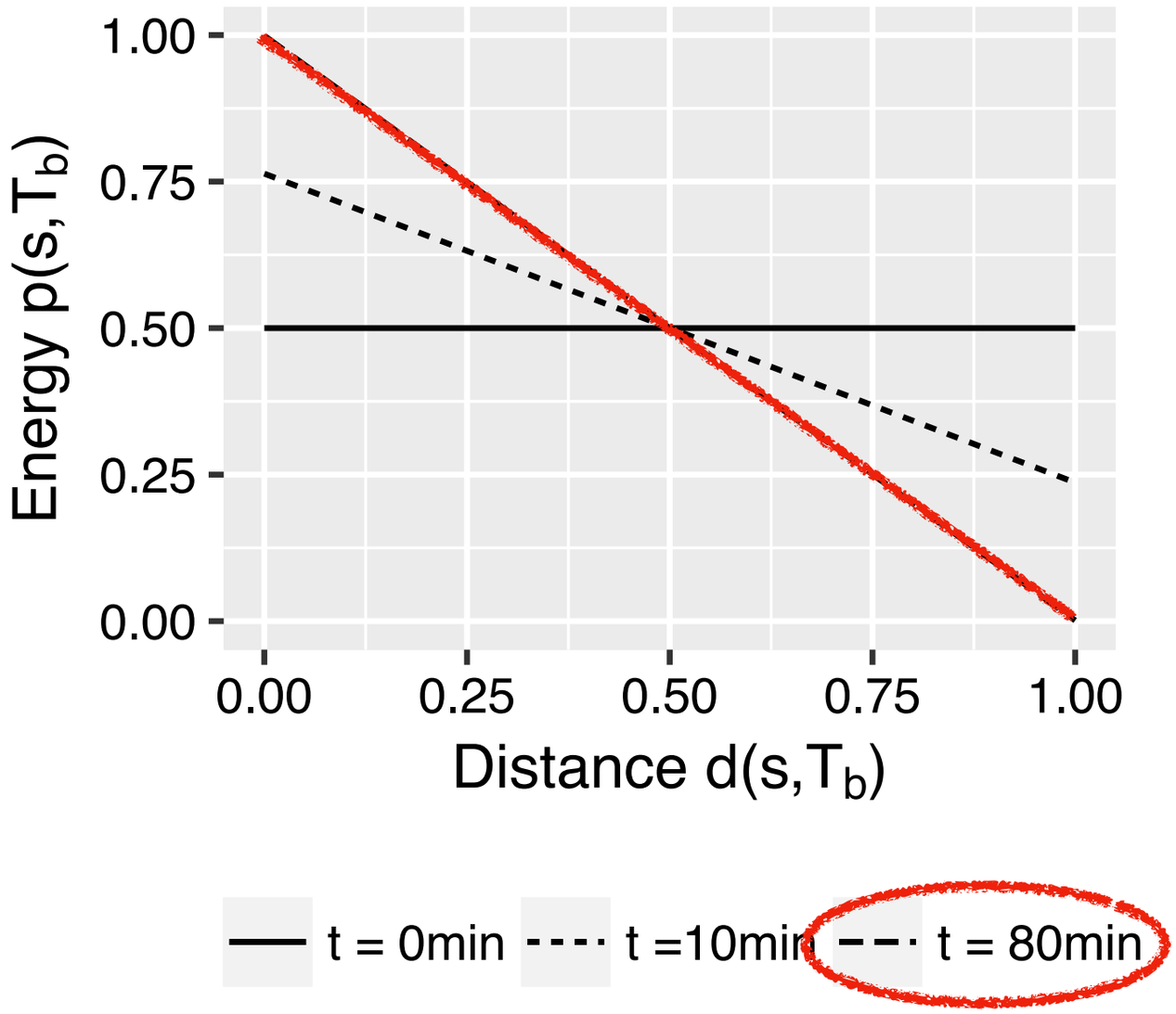
Directed Fuzzing as Optimisation Problem

- Integrating Simulated Annealing as power schedule
  - In the beginning (t = 0min), assign the **same energy to all seeds**.
  - Later (t=10min), assign a **bit more energy** to seeds that are **closer**.



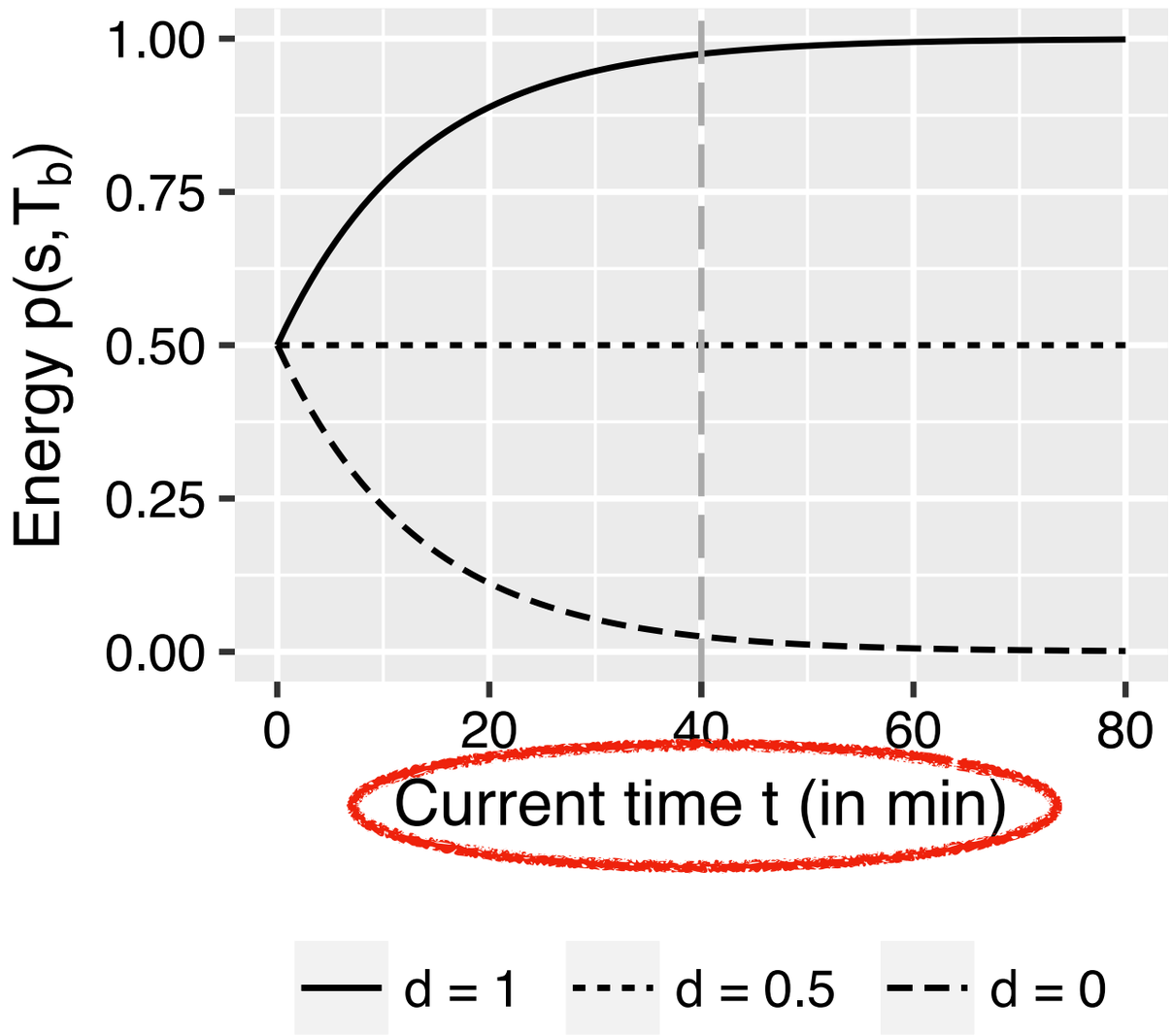
Directed Fuzzing as Optimisation Problem

- Integrating Simulated Annealing as power schedule
  - In the beginning (t = 0min), assign the **same energy to all seeds**.
  - Later (t=10min), assign a **bit more energy** to seeds that are **closer**.
  - At exploitation (t=80min), assign **maximal energy** to seeds that are **closest**.



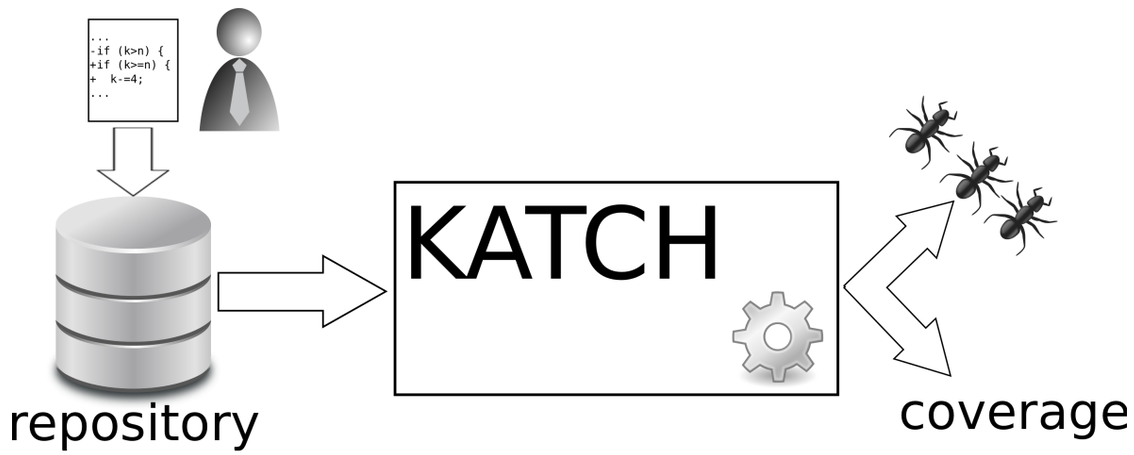
Directed Fuzzing as Optimisation Problem

- Integrating Simulated Annealing as power schedule
  - In the beginning (t = 0min), assign the **same energy to all seeds**.
  - Later (t=10min), assign a **bit more energy** to seeds that are **closer**.
  - At exploitation (t=80min), assign **maximal energy** to seeds that are **closest**.



Results

- **Patch Testing**: Reach changed statements
  - State-of-the-art in patch testing
    - **KATCH** (based on Klee symbolic exec. tool)
  - Experimental Setup
    - Reuse original **KATCH**-benchmark
    - Measure patch coverage (#changed BBs reached)
    - Measure vuln. detection (#errors discovered)



Results

- **Patch Testing**: Reach changed statements
  - State-of-the-art in patch testing
    - **KATCH** (based on Klee symbolic exec. tool)
  - Patch Coverage (#changed BBs reached)

	#Changed Basic Blocks	#Uncovered Changed BBs	KATCH	AFLGo
Binutils	852	702	135	159
Diffutils	166	108	63	64
Sum	1018	810	198	223

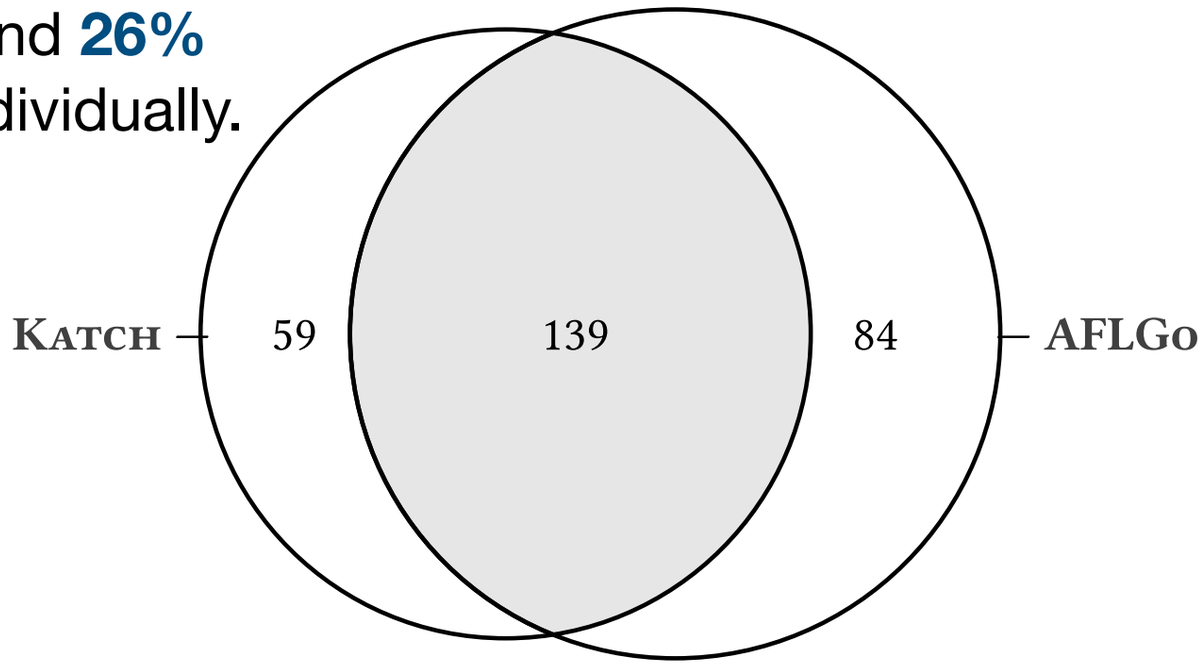
Results

- **Patch Testing**: Reach changed statements
  - State-of-the-art in patch testing
    - **KATCH** (based on Klee symbolic exec. tool)
  - Patch Coverage (#changed BBs reached)
    - While we would expect Klee to take a substantial lead, **AFLGo outperforms KATCH** in terms of patch coverage.

	#Changed Basic Blocks	#Uncovered Changed BBs	KATCH	AFLGo
Binutils	852	702	135	159
Diffutils	166	108	63	64
Sum	1018	810	198	223

Results

- **Patch Testing**: Reach changed statements
  - State-of-the-art in patch testing
    - **KATCH** (based on Klee symbolic exec. tool)
  - Patch Coverage (#changed BBs reached)
    - While we would expect Klee to take a substantial lead, **AFLGo outperforms KATCH** in terms of patch coverage.
    - **BUT**: **Together** they cover **42%** and **26%** more than **AFLGo** and **KATCH** individually. **They complement each other!**



Results

- **Patch Testing**: Reach changed statements
  - State-of-the-art in patch testing
    - **KATCH** (based on Klee symbolic exec. tool)
  - Patch Coverage (#changed BBs reached)
  - Vulnerability Detection (#errors discovered)

	KATCH	AFLGO		
	#Reports	**#Reports**[14]	**#New Reports**	**#CVEs**
Binutils	7	4	12	7
Diffutils	0	*N/A*	1	0
Sum	7	4	13	7

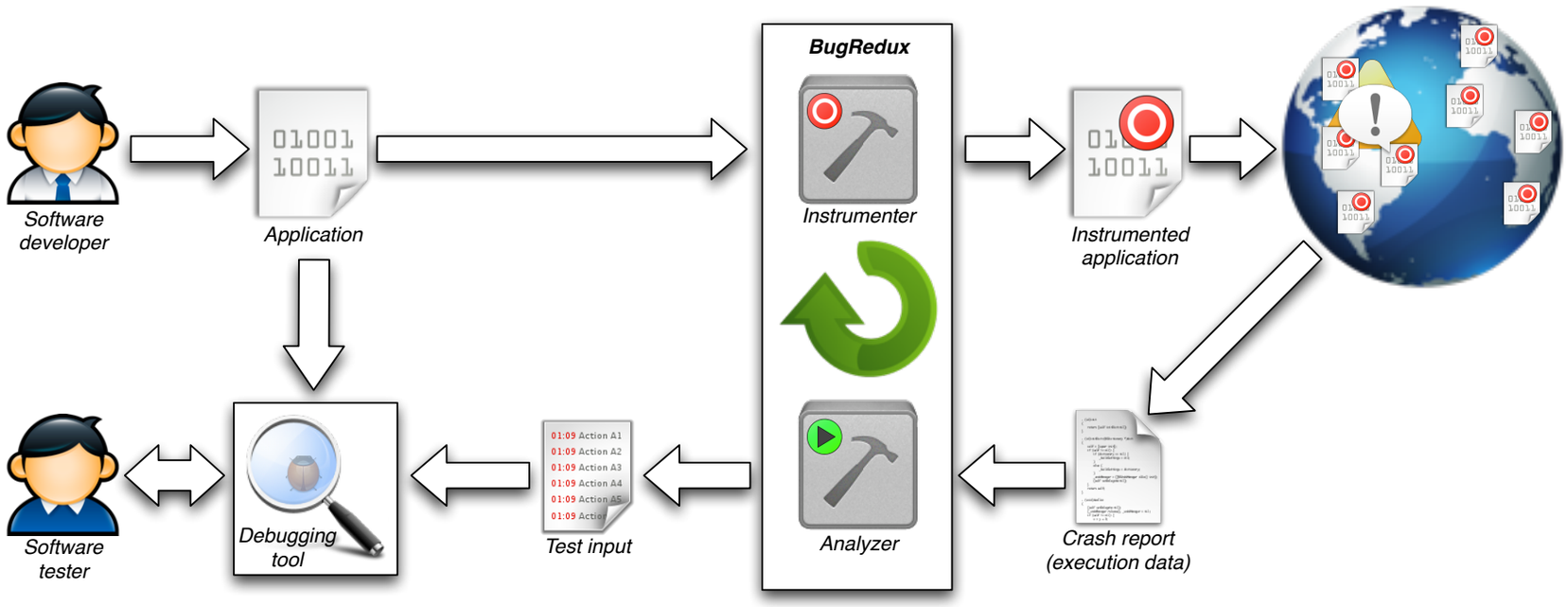
Results

- **Patch Testing**: Reach changed statements
  - State-of-the-art in patch testing
    - **KATCH** (based on Klee symbolic exec. tool)
  - Patch Coverage (#changed BBs reached)
  - Vulnerability Detection (#errors discovered)
    - **AFLGo found 13 previously unreported bugs (7 CVEs)** in addition to 4 of the 7 bugs that were found by **KATCH**.

	KATCH	AFLGO		
	#Reports	#Reports[14]	#New Reports	#CVEs
Binutils	7	4	12	7
Diffutils	0	*N/A*	1	0
Sum	7	4	13	7

Results

- **Crash Reproduction**: Exercise stack trace
  - State-of-the-art in crash reproduction
    - **BugRedux** (based on Klee symbolic exec. tool)
  - Experimental Setup
    - Reuse original **BugRedux**-benchmark
    - Determine whether or not crash can be reproduced



Results

- **Crash Reproduction**: Exercise stack trace
  - State-of-the-art in crash reproduction
    - **BugRedux** (based on Klee symbolic exec. tool)
  - Experimental Setup
    - Reuse original **BugRedux**-benchmark
    - Determine whether or not crash can be reproduced

Subjects	BugRedux	AFLGo	Comments
sed.fault1	✗	✗	Takes two files as input
sed.fault2	✗	✓	
grep	✗	✓	
gzip.fault1	✗	✓	
gzip.fault2	✗	✓	
ncompress	✓	✓	
polymorph	✓	✓	

AFLGo reproduces 3 times more crashes!

Result Summary

- Our **directed greybox fuzzer** (AFLGo) **outperforms symbolic execution-based directed fuzzers** (KATCH & BugRedux)
  - in terms of **reaching more target locations** and
  - in terms of **detecting more vulnerabilities**,
  - on their own, original benchmark sets.

Motivation

- **Directed Fuzzing**: classical **constraint satisfaction prob**.
  - **Program analysis** to identify **program paths** that reach given program locations.
  - **Symbolic Execution** to derive **path conditions** for any of the identified paths.
  - **Constraint Solving** to find an **input** that
    - **satisfies the path condition** and thus
    - **reaches a program location** that was given.

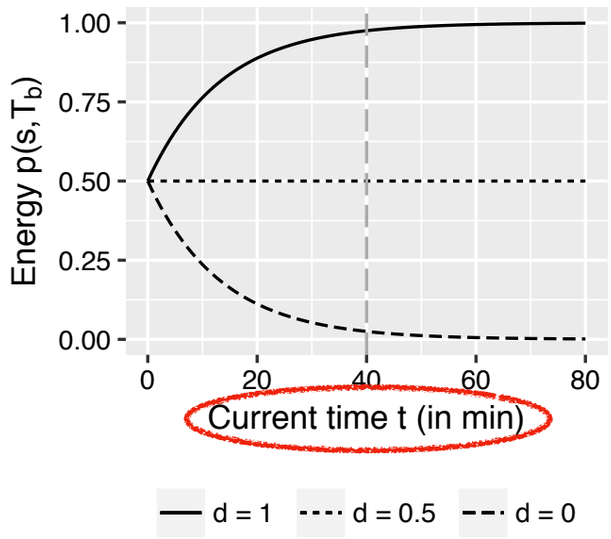
Requires heavy-weight machinery!

Overview

- **Directed Fuzzing** as **optimisation problem**!
  1. Instrumentation Time:
     1. Extract **call graph** (CG) and **control-flow graphs** (CFGs).
     2. For each **BB**, compute **distance** to target locations.
     3. Instrument program to **aggregate distance values**.
  2. Runtime, **for each input**
     1. collect coverage and distance **information**, and
     2. decide **how long to be fuzzed** based on distance.
        - If input is **closer** to the targets, it is fuzzed for **longer**.
        - If input is **further away** from the targets, it is fuzzed for **shorter**.

Directed Fuzzing as Optimisation Problem

- Integrating Simulated Annealing as power schedule
  - In the beginning (t = 0min), assign the **same energy to all seeds**.
  - Later (t=10min), assign a **bit more energy** to seeds that are **closer**.
  - At exploitation (t=80min), assign **maximal energy** to seeds that are **closest**.



Result Summary

- Our **directed greybox fuzzer** (AFLGo) **outperforms*** **symbolic execution-based directed fuzzers** (KATCH & BugRedux)
  - in terms of **reaching more target locations** and
  - in terms of **detecting more vulnerabilities**,
  - on their own, original benchmark sets.

Motivation

- **Directed Fuzzing**: classical **constraint satisfaction prob**.
  - **Program analysis** to identify **program paths** that reach given program locations.
  - **Symbolic Execution** to derive **path conditions** for any of the identified paths.
  - **Constraint Solving** to find an **input** that
    - **satisfies the path condition** and thus
    - **reaches a program location** that was given.

Requires heavy-weight machinery!

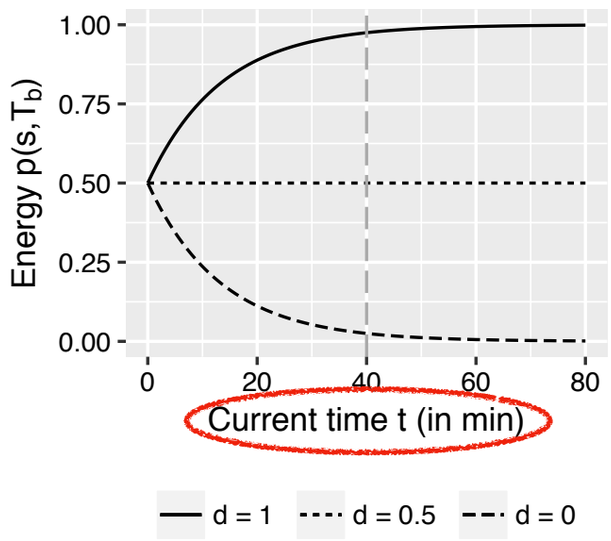
Overview

- **Directed Fuzzing** as **optimisation problem**!
  1. Instrumentation Time:
     1. Extract **call graph** (CG) and **control-flow graphs** (CFGs).
     2. For each **BB**, compute **distance** to target locations.
     3. Instrument program to **aggregate distance values**.
  2. Runtime, **for each input**
     1. collect coverage and distance **information**, and
     2. decide **how long to be fuzzed** based on distance.
        - If input is **closer** to the targets, it is fuzzed for **longer**.
        - If input is **further away** from the targets, it is fuzzed for **shorter**.

Questions?

Directed Fuzzing as Optimisation Problem

- Integrating Simulated Annealing as power schedule
  - In the beginning (t = 0min), assign the **same energy to all seeds**.
  - Later (t=10min), assign a **bit more energy** to seeds that are **closer**.
  - At exploitation (t=80min), assign **maximal energy** to seeds that are **closest**.



Result Summary

- Our **directed greybox fuzzer** (AFLGo) **outperforms*** **symbolic execution-based directed fuzzers** (KATCH & BugRedux)
  - in terms of **reaching more target locations** and
  - in terms of **detecting more vulnerabilities**,
  - on their own, original benchmark sets.