Software Testing in a Data-driven Approach

Dongdong She

Hong Kong University of Science and Technology

- 1966, MIT Computation Center
- IBM 7094, Ancient OS CTSS (before UNIX)
- A technical issue **leaked all user's passwords** in plain text

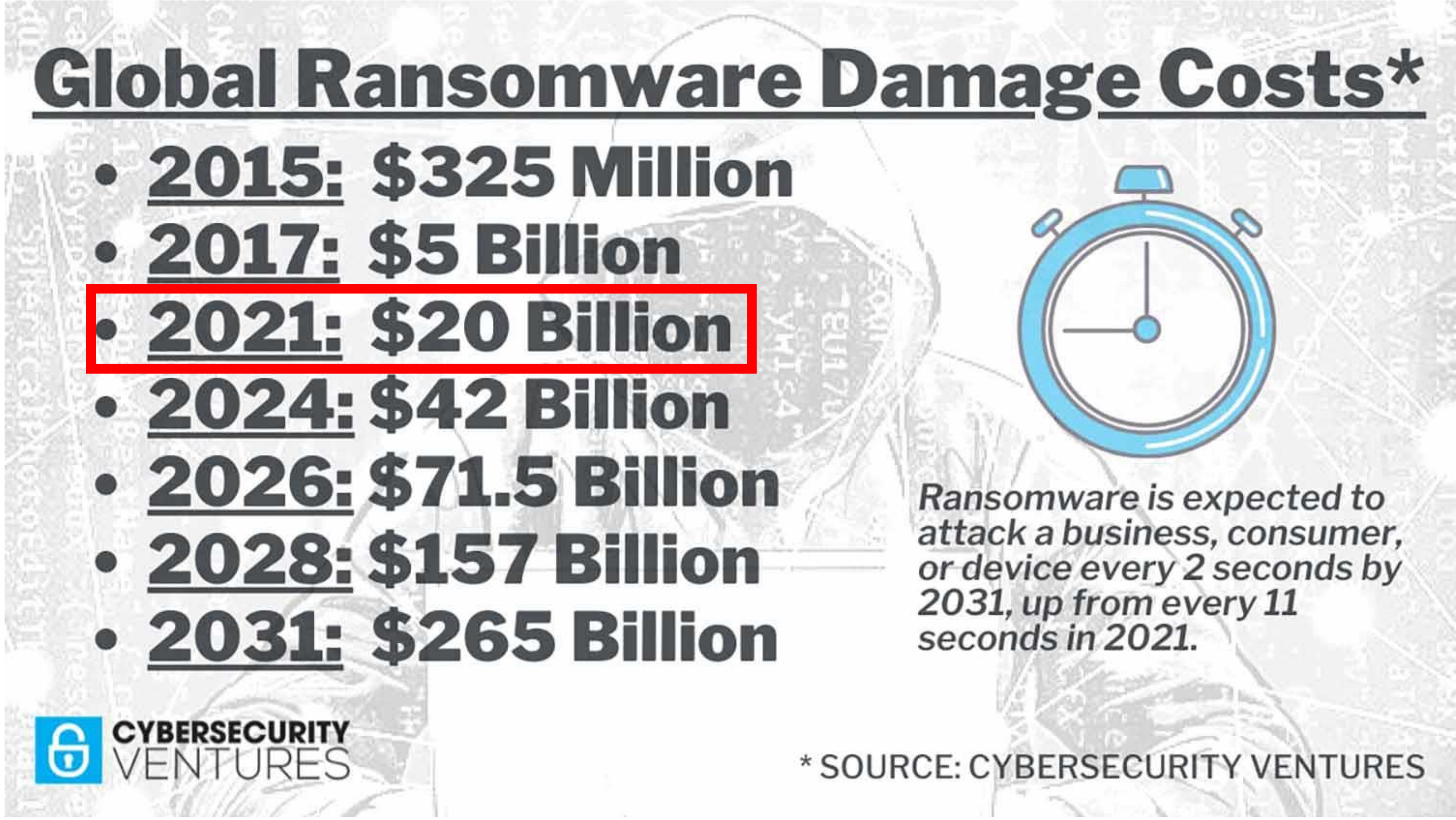
One of the earliest cybersecurity vulnerability

Source: https://multicians.org/thvv/7094.html

What is Security Vulnerability?

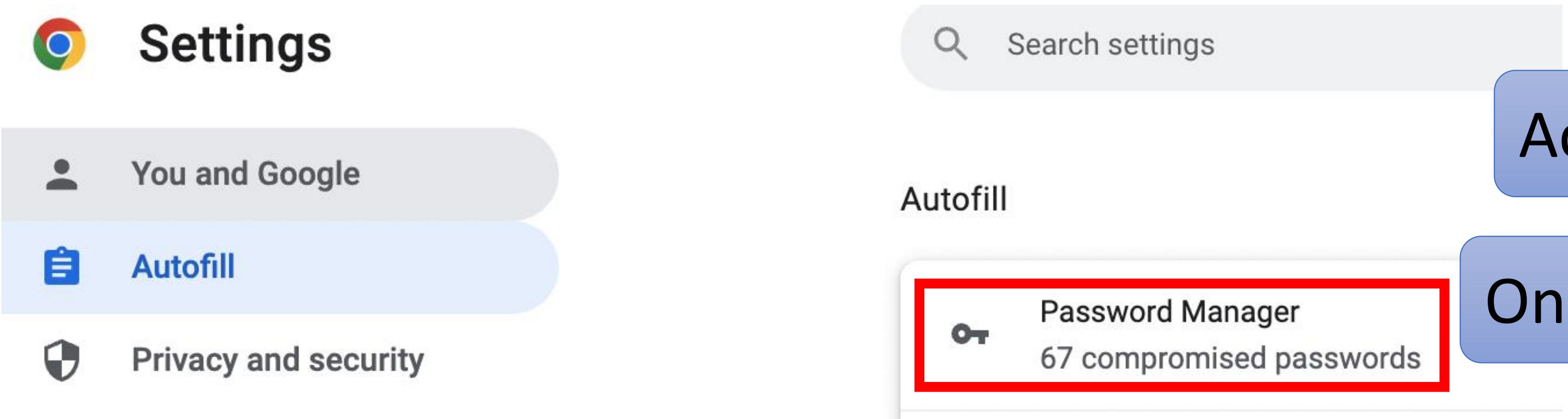
- Software code flaws or system misconfigurations
- Lead to unauthorized access/control of computer systems
- Huge real-world impact on our lives

Global Ransomware Attacks



Global ransomware attacks cost **billions of dollars** every year

Confidential Data Breach



Account leaked: **12 billion**[1]

Online population: **5 billion**[2]

Every people has an average of **2.4 accounts leaked** online

[1] https://haveibeenpwned.com/
[2] https://datareportal.com/global-digital-overview

Why Do Vulnerabilities Exist?

- Humans write code
- Humans inevitably make the mistake
- Current AI-code completion still contains vulnerabilities[1]

It is hard to eliminate all the bugs

[1] Pearce et al. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. IEEE S&P'22.

Automated Approaches to Find Vulnerabilities

- **Fuzzing**
- Static/Dynamic analysis
- Formal verification
- Symbolic execution

- Simple and effective
- Light-weight and scalable
- Widely-used in industry

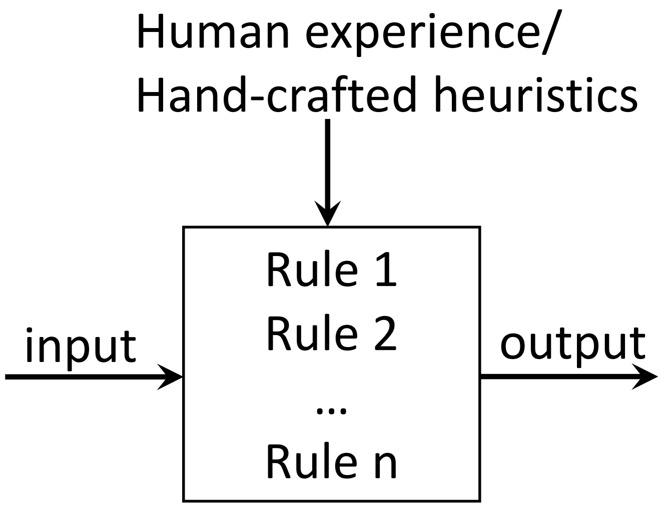
Home / Innovation / Security

Linus Torvalds says targeted fuzzing is improving Linux security

Linux 4.14 release candidate five is out. "Go out and test," says Linus Torvalds.

Limitation of Existing Approaches

Rule-based design: rely on a set of **static rules or heuristics**.

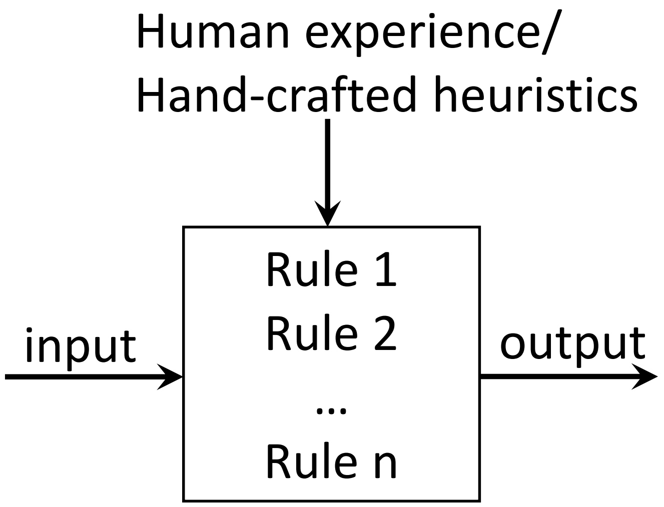


Rule-based system

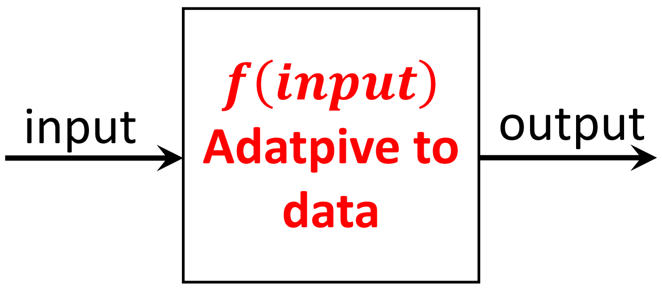
Rule 1: Schedule the seed by file size
Rule 2: Schedule the seed by execution time
Rule 3: Randomly mutate the first byte of the seed
Rule N: …

- Good heuristics are expensive
- Often **fail to generalize** on diverse programs

Rule-Based vs. Data-Driven



Rule-based system

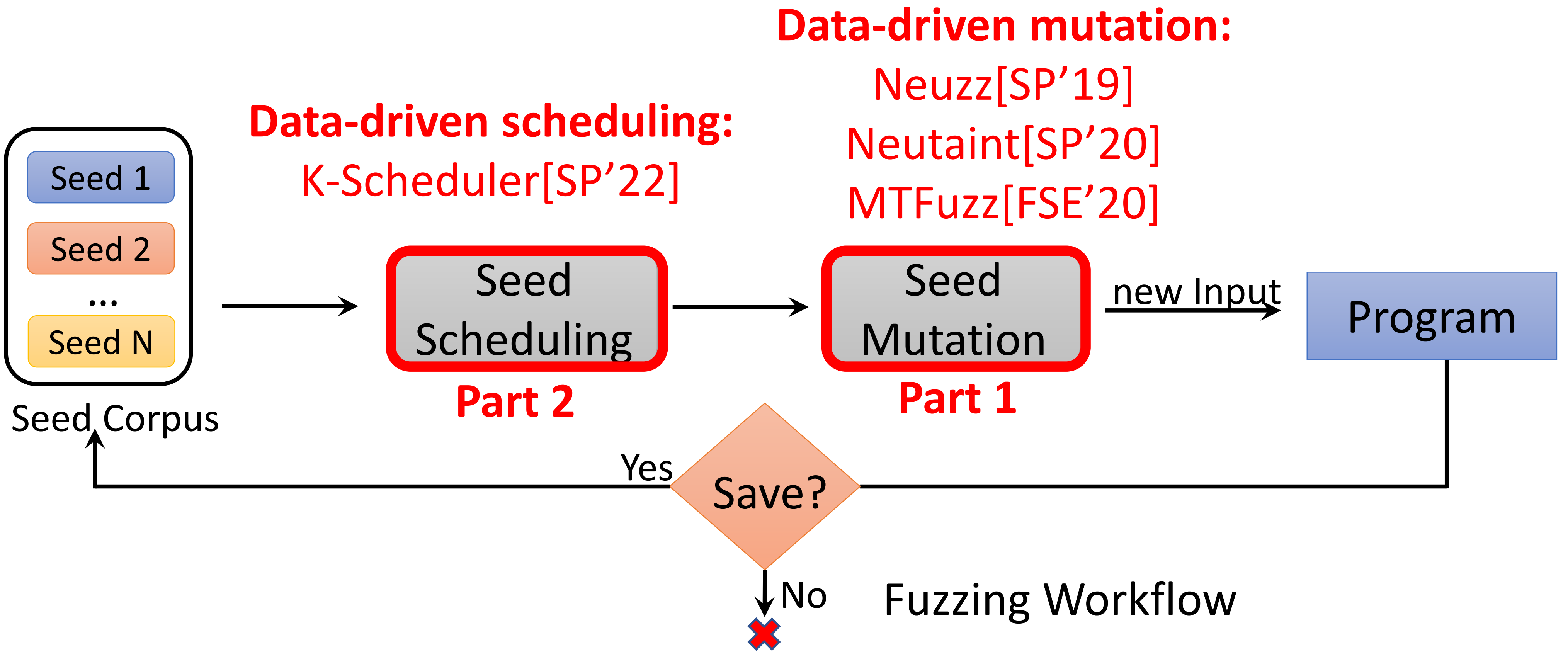


Data-driven system

Data-driven approach is adaptive and effective

My Research

- Part 1: Data-driven mutation
- Part 2: Data-driven scheduling



Fuzzing Workflow

NEUZZ: Data-Driven Mutation

Background: Fuzzing is a **search problem** aimed at discovering testcases that can trigger vulnerabilities

Problem: How to **effectively search** for interesting testcases

Existing works: rule-based mutation
Our solution: data-driven mutation
- Fuzzing as an **optimization problem** => Gradient-guided mutation

Input Space of Program

High-dimensional and **discrete** input space

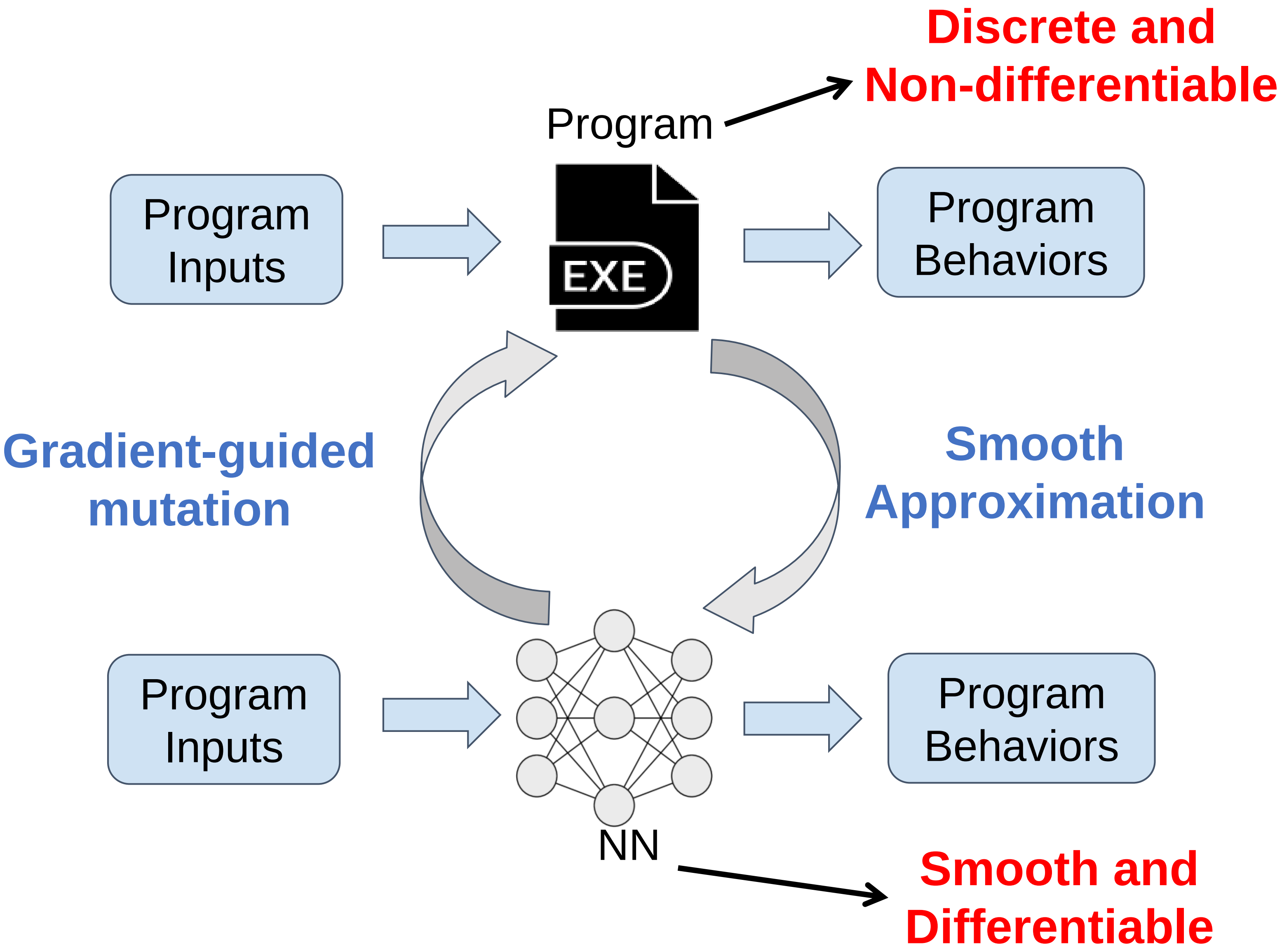
$$X = [x_1, x_2, x_3, \ldots, x_n]$$

x_1	0	1	2	3	4	.	.	.	.	.	.	255
x_2	0	1	2	3	4	.	.	.	.	.	.	255
$\vdots$						$\vdots$						
x_n	0	1	2	3	4	.	.	.	.	.	.	255

n is the **length of the input**
Total possible inputs = **256^n**

Random mutation in huge search space is **inefficient**

Overview of NEUZZ



K-Scheduler: Data-Driven Scheduling

Background: fuzzing needs to **choose a seed** during the search

Problem: How to choose **a promising seed** from seed corpus

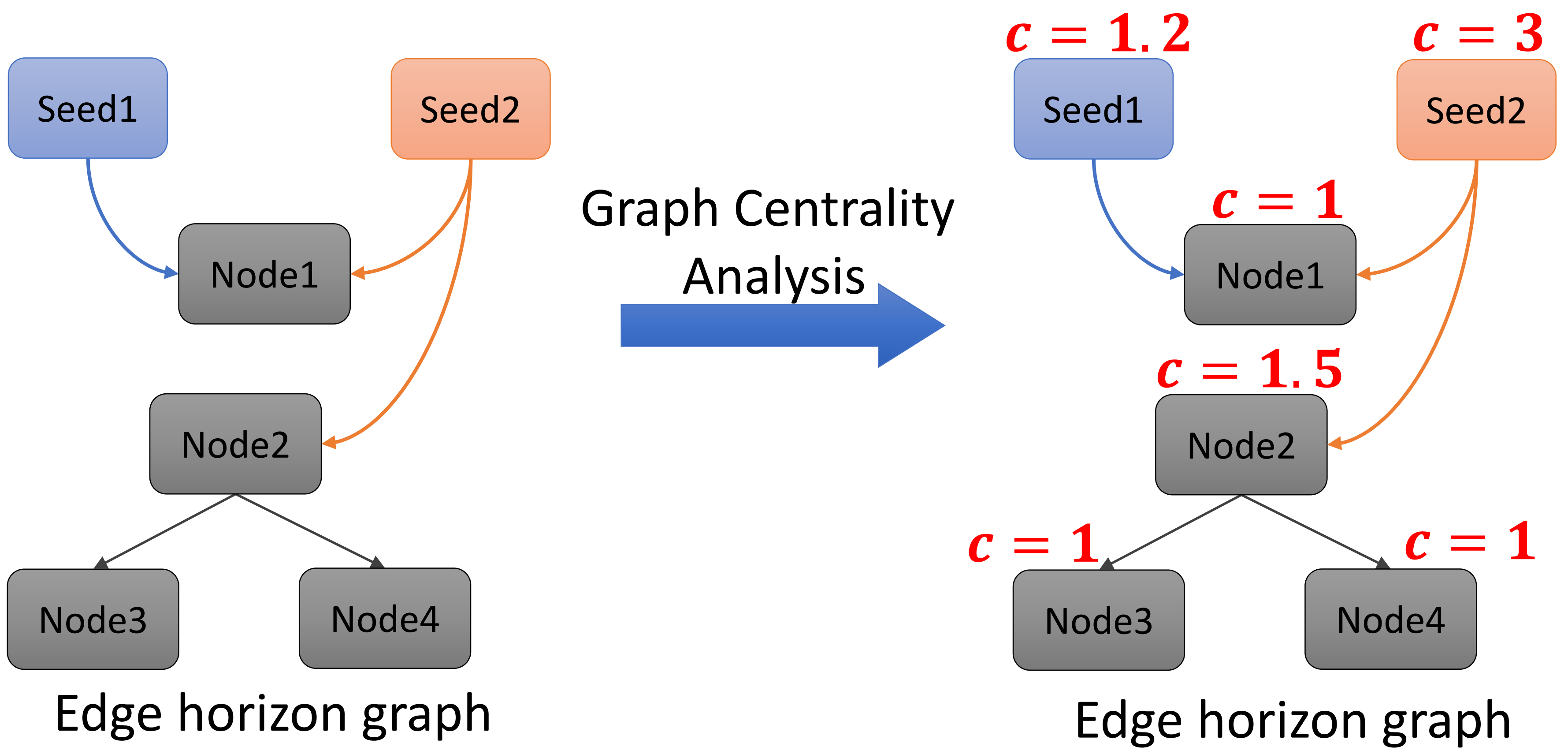
Existing work: rule-based selection

Our approach: Data-driven scheduling

- Fuzzing as an **influence analysis** problem => Graph centrality analysis

Overview of K-Scheduler

We use graph centrality score to **estimate the search gain** of each seed



Future Directions

- Neural-symbolic software testing
- LLM-assisted program analysis

Neural-Symbolic Software Testing

Software testing with **domain knowledge**

- Smart contract, Network protocol, Autonomous driving, Deep Learning API

Temporal Logic
Fuzzy Logic
Solver-based techniques

Symbolic Module
Domain knowledge

Robustness
Interpretability

+

NN approximation
Optimizations
Sampling-based techniques

Neural Module
Learn from data

Expressiveness
Scalability

=

Neural-Symbolic Software Testing

Explore the domain-specific software testing in a **neural-symbolic** way

LLM-Assisted Program Analysis

Leverage LLM's capability of **code comprehension** and **code summary** to boost traditional program analysis tasks

- Dataflow analysis, vulnerability detection (e.g., race condition, memory corruption, integer overflow), software testing (e.g., fuzzing)

- Task decomposition
- Automatic prompt generation
- Retrieval augmented generation