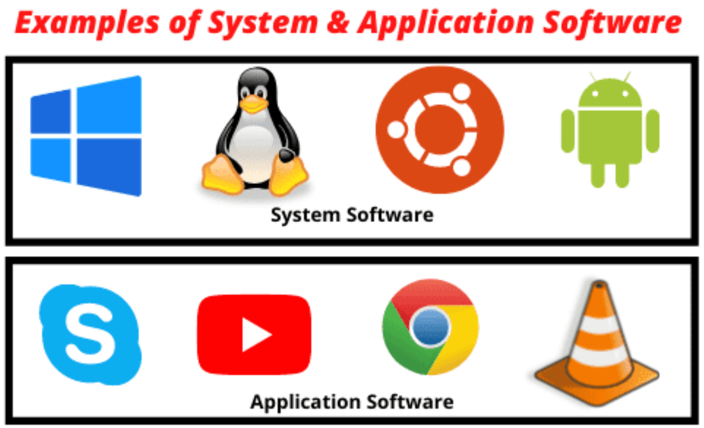
# FastKLEE: Faster Symbolic Execution via Reducing Redundant Bound Checking of Type-Safe Pointers

***ESEC/FSE 2022 Tool Demonstration***

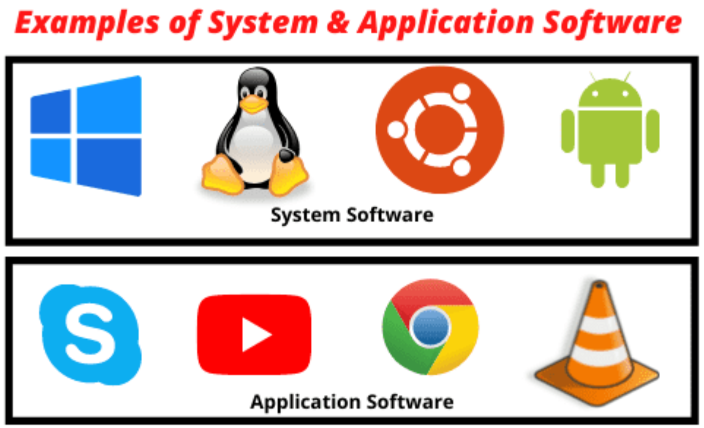
**Haoxin Tu**, Lingxiao Jiang, Xuhua Ding (Singapore Management University)
He Jiang (Dalian University of Technology)

# Background

# Background

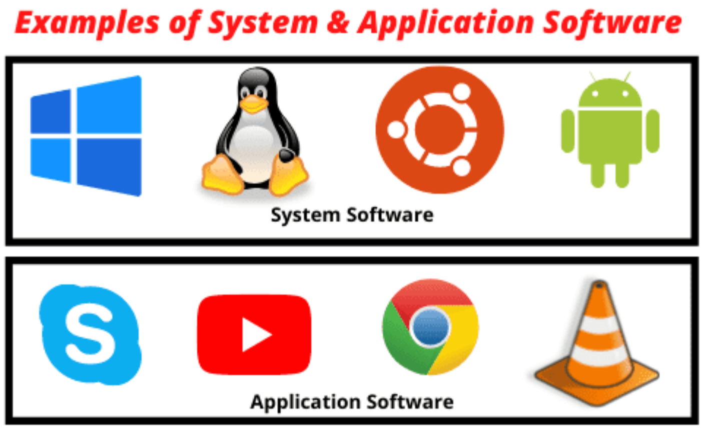

Examples of System & Application Software
System Software
Application Software

# Background



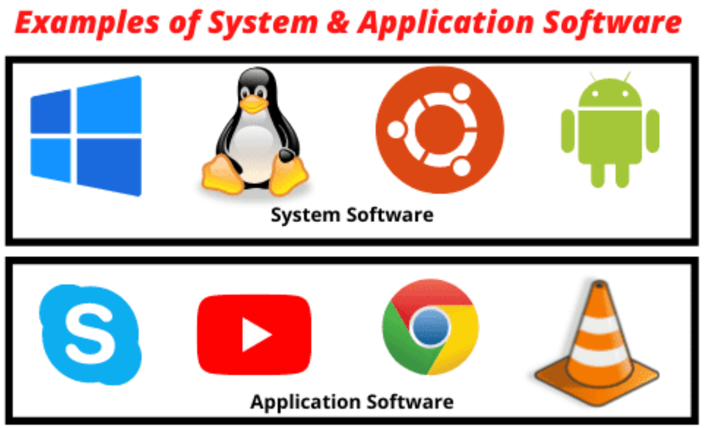

- **Q: How to improve the quality of software?**

# Background




- **Q: How to improve the quality of software?**
- **A: Software testing**

# Background



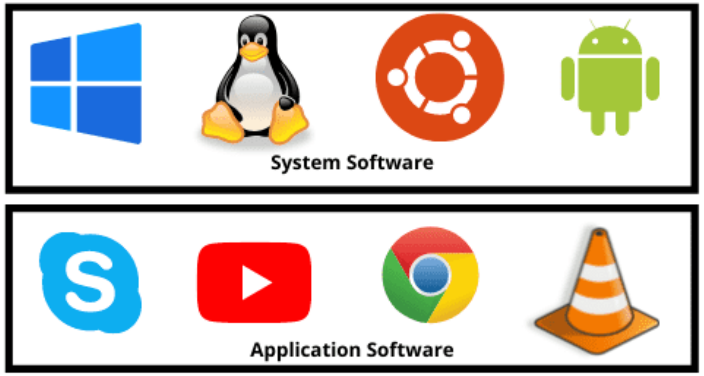

- Q: How to improve the quality of software?
- A: Software testing

↓

(Symbolic Execution)

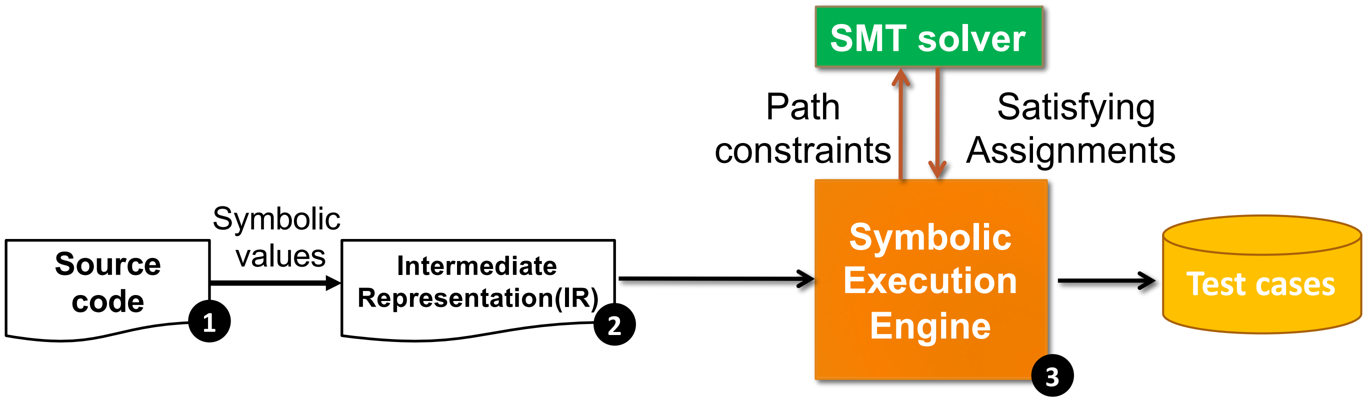
# Background

**Examples of System & Application Software**




- **Q: How to improve the quality of software?**
- **A: Software testing**

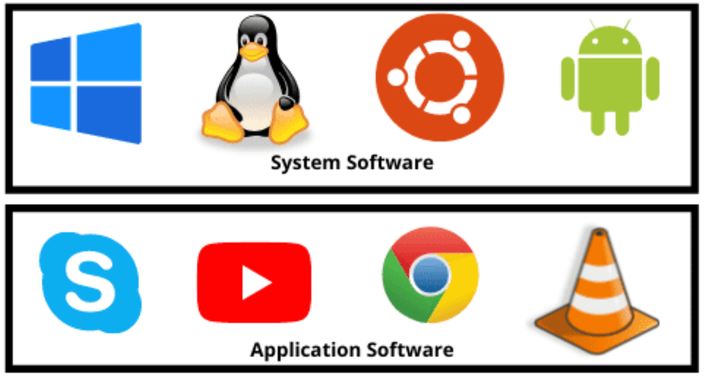
**(Symbolic Execution)**




- **General workflow of traditional symbolic execution engine (e.g., KLEE)**

# Background

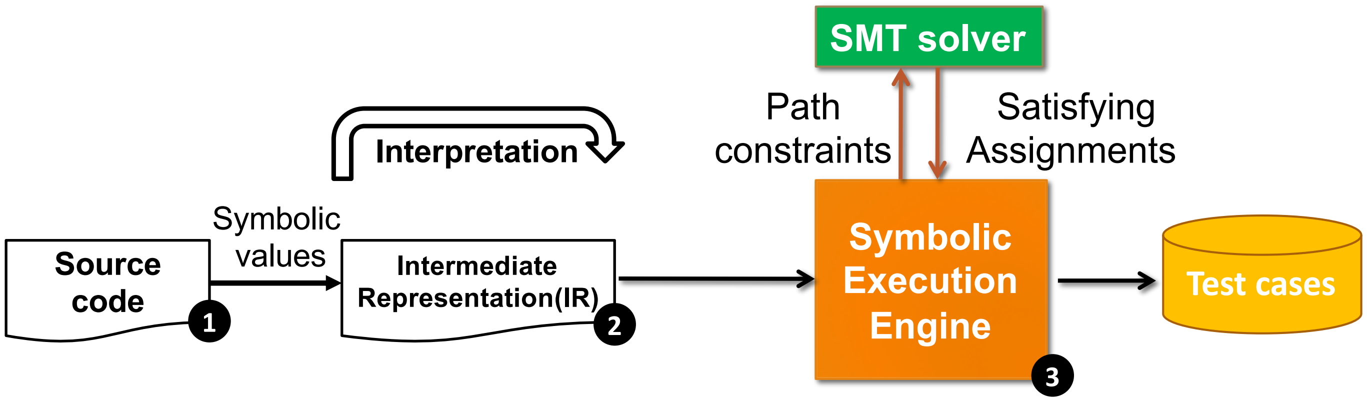
**Examples of System & Application Software**




- **Q: How to improve the quality of software?**
- **A: Software testing**

↓

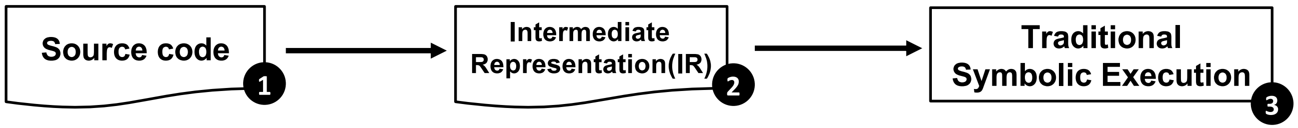
**(Symbolic Execution)**



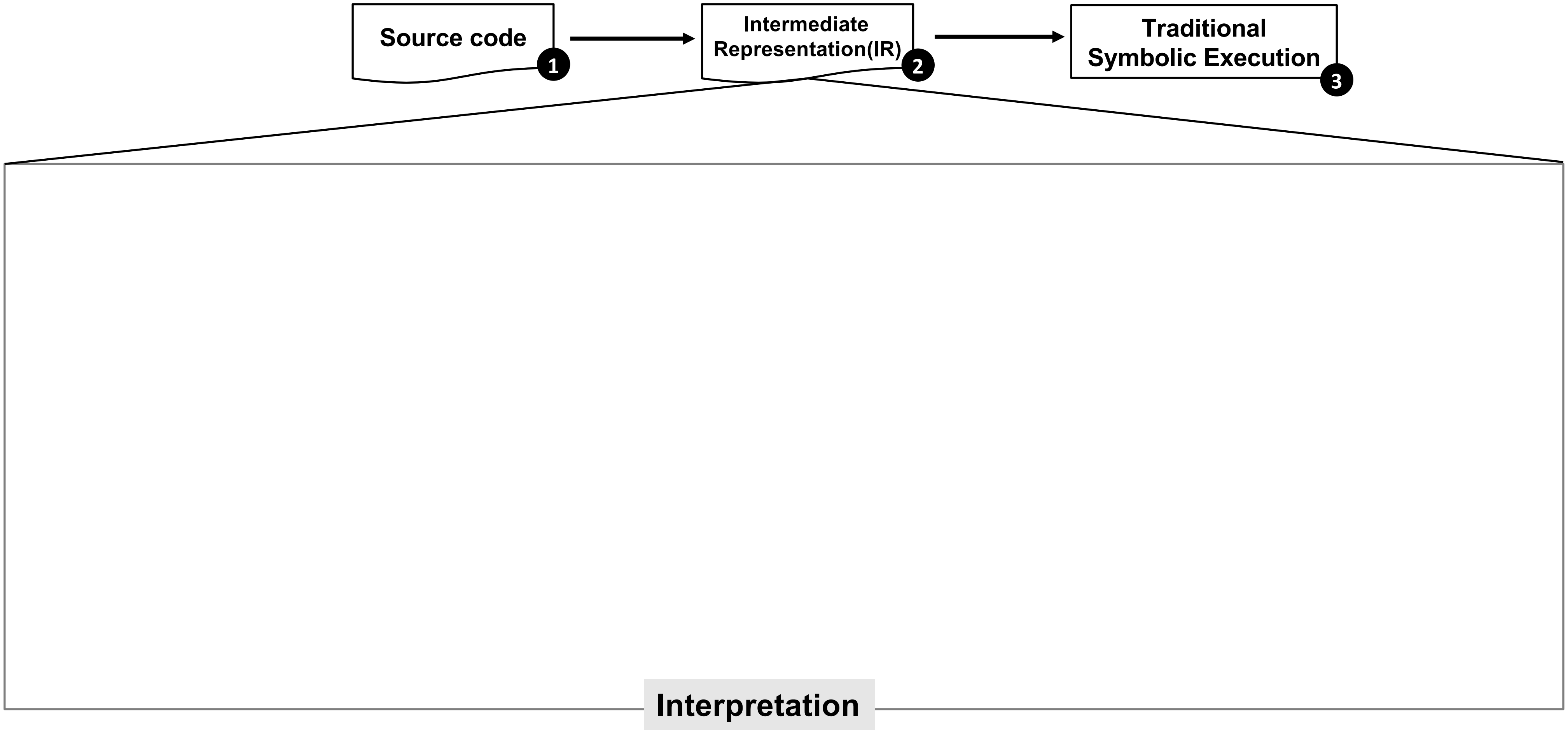

- **General workflow of traditional symbolic execution engine (e.g., KLEE)**

# Motivation

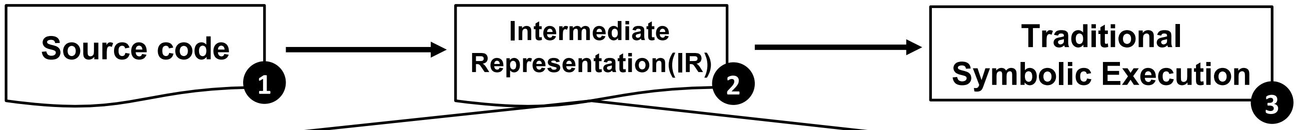
# Motivation


Source code
1
Intermediate
Representation(IR)
2
Traditional
Symbolic Execution
3

# Motivation


Source code
1
Intermediate Representation(IR)
2
Traditional Symbolic Execution
3
Interpretation

# Motivation




## (1) Observation

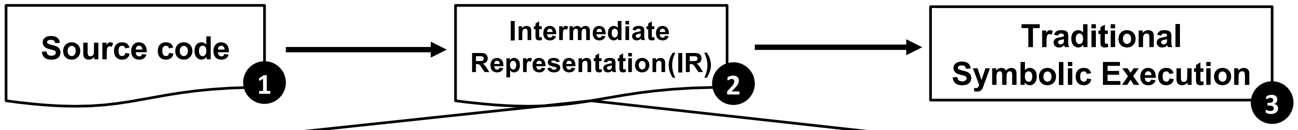
– The number of interpreted instructions tends to be **huge** (several billion only in one hour run)

```
Elapsed: 01:00:04
KLEE: done: explored paths = 125017
KLEE: done: avg. constructs per query = 74
KLEE: done: total queries = 8859
KLEE: done: valid queries = 6226
KLEE: done: invalid queries = 2633
KLEE: done: query cex = 8859

KLEE: done: total instructions = 605113213
KLEE: done: completed paths = 125017
KLEE: done: generated tests = 65
```

**Interpretation**

# Motivation




## (1) Observation

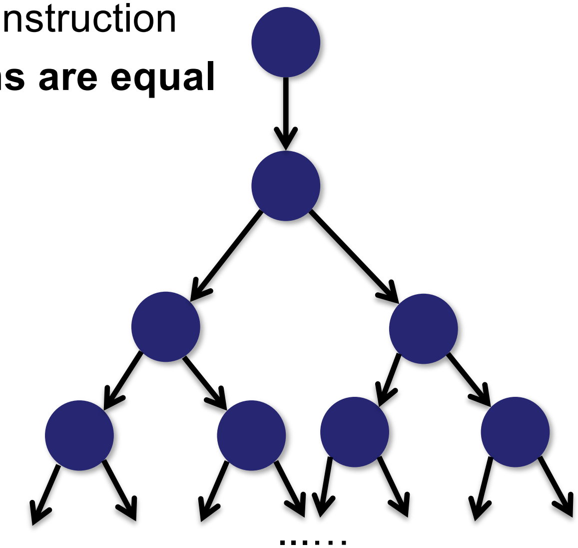
- The number of interpreted instructions tends to be **huge** (several billion only in one hour run)

```
Elapsed: 01:00:04
KLEE: done: explored paths = 125017
KLEE: done: avg. constructs per query = 74
KLEE: done: total queries = 8859
KLEE: done: valid queries = 6226
KLEE: done: invalid queries = 2633
KLEE: done: query cex = 8859

KLEE: done: total instructions = 605113213
KLEE: done: completed paths = 125017
KLEE: done: generated tests = 65
```

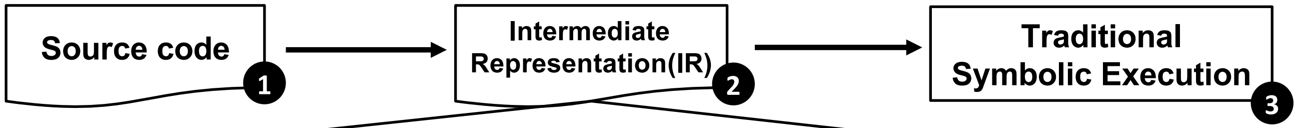
## (2) Overheads in current symbolic execution

- The color depth represents the overheads of an interpreted instruction
- **All instructions are equal**




**Interpretation**

# Motivation



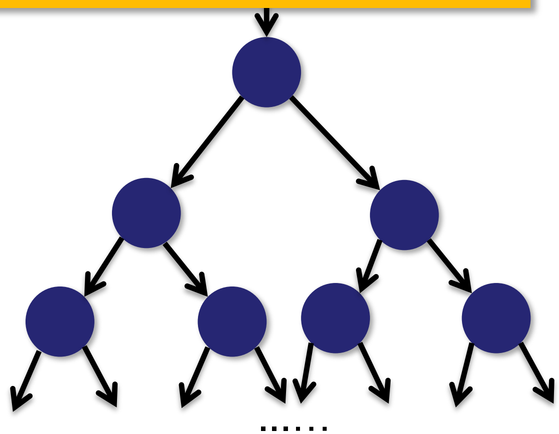

**(1) Observation**

**(2) Overheads in current symbolic execution**

**Can we reduce the overheads of interpreted instructions for faster symbolic execution?**

```
Elapsed: 01:00:04
KLEE: done: explored paths = 125017
KLEE: done: avg. constructs per query = 74
KLEE: done: total queries = 8859
KLEE: done: valid queries = 6226
KLEE: done: invalid queries = 2633
KLEE: done: query cex = 8859

KLEE: done: total instructions = 605113213
KLEE: done: completed paths = 125017
KLEE: done: generated tests = 65
```




**Interpretation**

# Solution – FastKLEE (1/2)

[1] Ccured: George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. 2005. **CCured: type-safe retrofitting of legacy software**. ACM Trans. Program. Lang. Syst. 27, 3 (May 2005), 477–526.

# Solution – FastKLEE (1/2)

- **Key insight**

[1] Ccured: George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. 2005. **CCured: type-safe retrofitting of legacy software**. ACM Trans. Program. Lang. Syst. 27, 3 (May 2005), 477–526.

# Solution – FastKLEE (1/2)

- **Key insight**
  - Only a small portion of memory-related instructions need bound checking

[1] Ccured: George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. 2005. **CCured: type-safe retrofitting of legacy software**. ACM Trans. Program. Lang. Syst. 27, 3 (May 2005), 477–526.

# Solution – FastKLEE (1/2)

- **Key insight**
  - Only a small portion of memory-related instructions need bound checking
    - Reduce interpreting overhead of most frequently interpreted ones (i.e., load/store instruction)

[1] Ccured: George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. 2005. **CCured: type-safe retrofitting of legacy software**. ACM Trans. Program. Lang. Syst. 27, 3 (May 2005), 477–526.

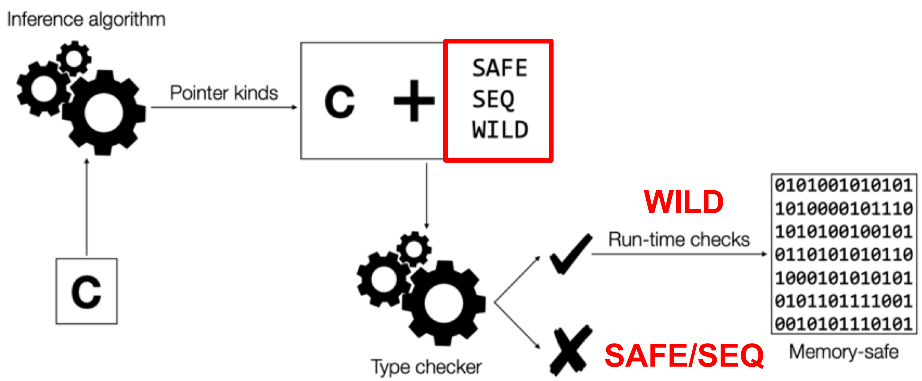
# Solution – FastKLEE (1/2)

- **Key insight**
  - Only a small portion of memory-related instructions need bound checking
    - Reduce interpreting overhead of most frequently interpreted ones (i.e., load/store instruction)
  - Inspired by ***Type Inference*** system [1]

[1] Ccured: George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. 2005. **CCured: type-safe retrofitting of legacy software**. ACM Trans. Program. Lang. Syst. 27, 3 (May 2005), 477–526.

# Solution – FastKLEE (1/2)

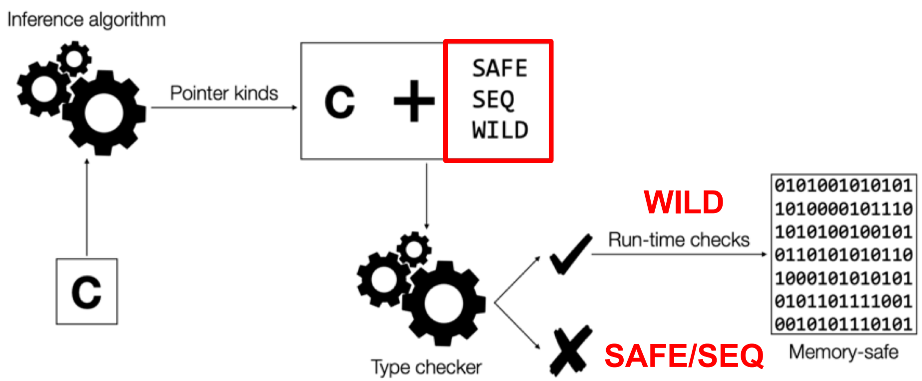
- **Key insight**
  - Only a small portion of memory-related instructions need bound checking
    - Reduce interpreting overhead of most frequently interpreted ones (i.e., load/store instruction)
  - Inspired by ***Type Inference*** system [1]



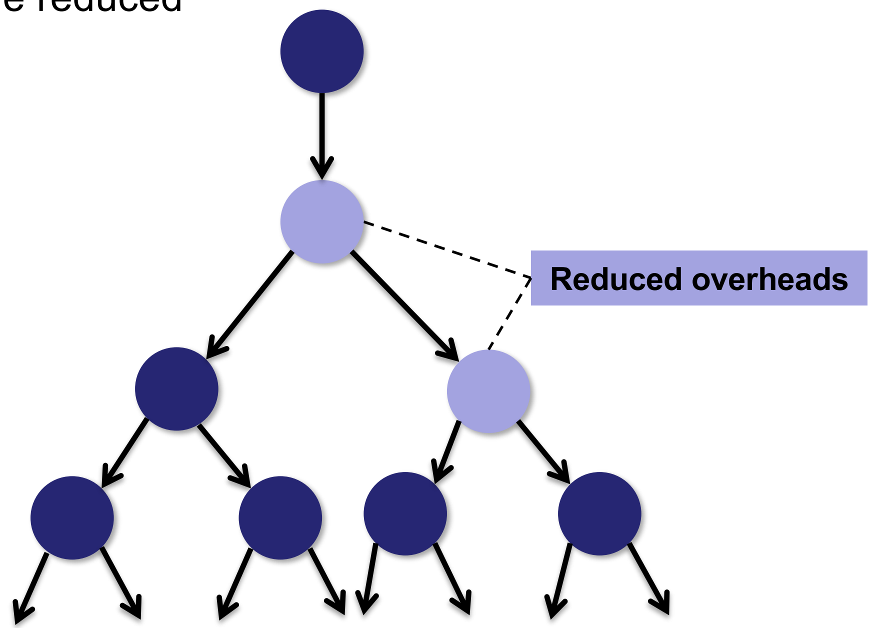

[1] Ccured: George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. 2005. **CCured: type-safe retrofitting of legacy software**. ACM Trans. Program. Lang. Syst. 27, 3 (May 2005), 477–526.

# Solution – FastKLEE (1/2)

- **Key insight**
  – Only a small portion of memory-related instructions need bound checking
    – Reduce interpreting overhead of most frequently interpreted ones (i.e., load/store instruction)
  – Inspired by ***Type Inference*** system [1]




- **Advantage: overheads in FastKLEE**
  – Interpretation overheads for some instructions are reduced

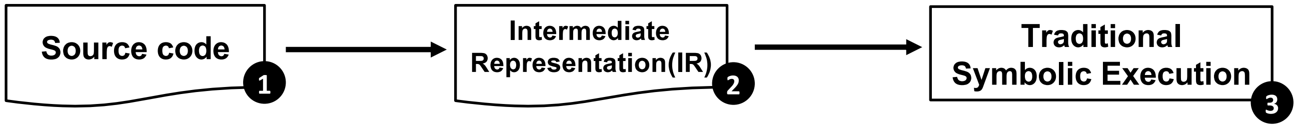



[1] Ccured: George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. 2005. **CCured: type-safe retrofitting of legacy software**. ACM Trans. Program. Lang. Syst. 27, 3 (May 2005), 477–526.

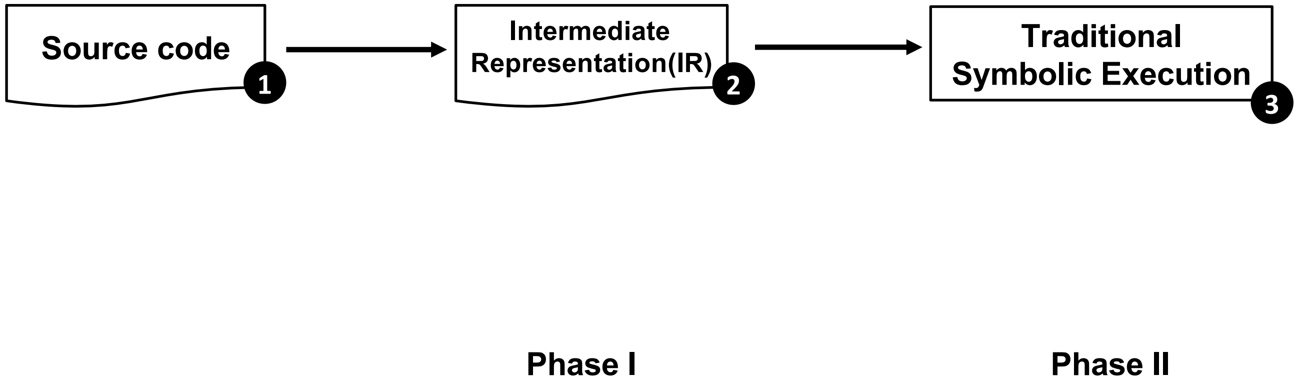
# Solution – FastKLEE (2/2)

# Solution – FastKLEE (2/2)

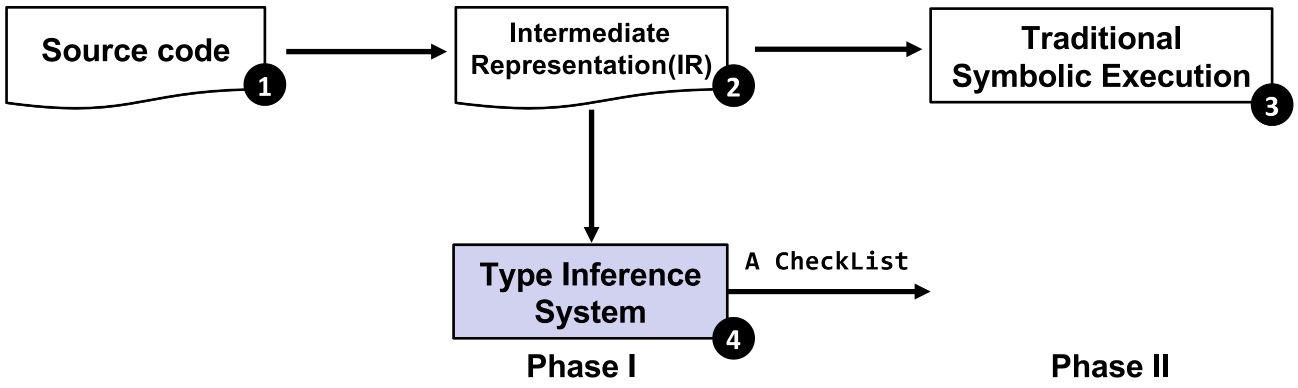




# Solution – FastKLEE (2/2)



# Solution – FastKLEE (2/2)

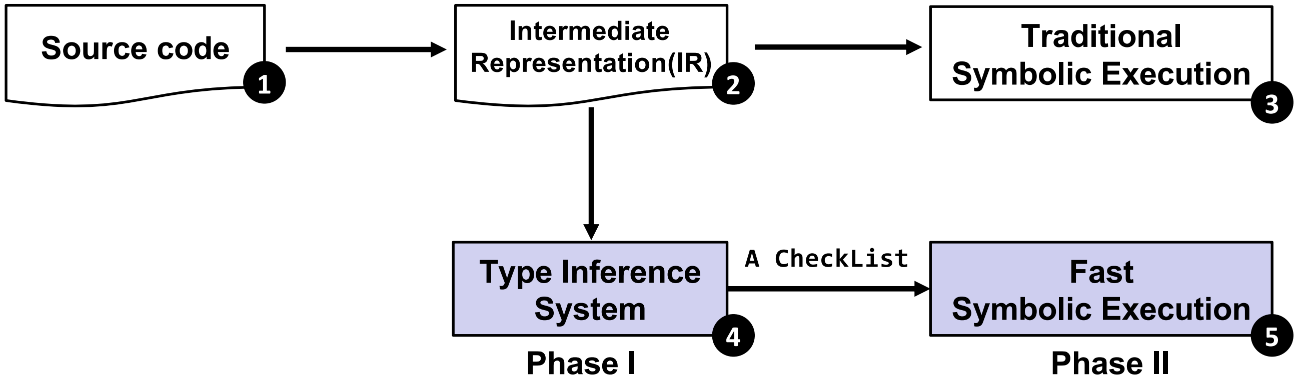



❹ • **Phase I**: Introduce a **Type Inference System** to classify memory-related instruction types

– ***Unsafe*** memory instructions will be stored in **CheckList**

# Solution – FastKLEE (2/2)

SMU Singapore Management University | School of Information Systems

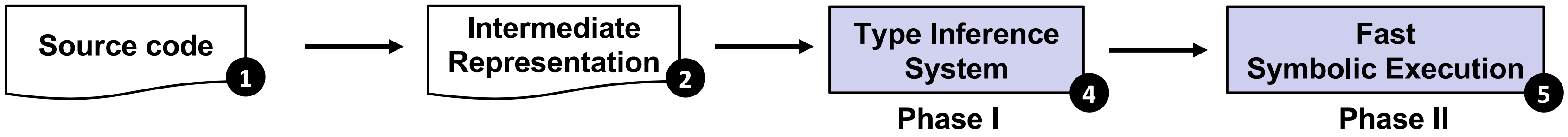



❹ • **Phase I**: Introduce a **Type Inference System** to classify memory-related instruction types
  – ***Unsafe*** memory instructions will be stored in **CheckList**

❺ • **Phase II**: Conduct **Customized Memory Operation** in Fast symbolic execution
  – Only perform checking for ***Unsafe*** memory instructions during interpretation

# Tool demonstration

School of
Information Systems




- **Impementation**
  - KLEE [1] and Ccured [2]

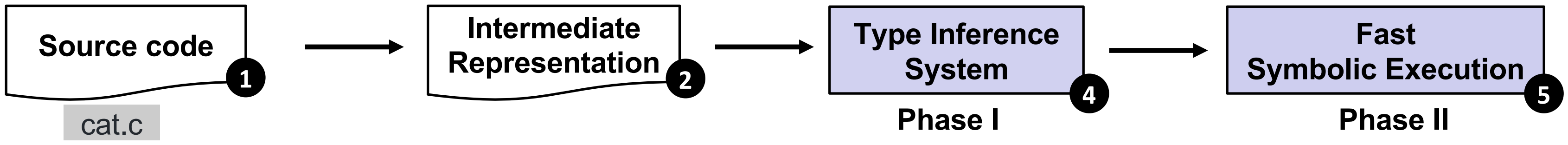
Installation of FastKLEE

Full video demo: https://youtu.be/fjV_a3kt-mo

[1] Cristian Cadar, Daniel Dunbar, and Dawson Engler. 2008. **KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs.** In OSDI'08. USENIX Association, USA, 209–224.
[2] Ccured: George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. 2005. **CCured: type-safe retrofitting of legacy software**. ACM Trans. Program. Lang. Syst. 27, 3 (May 2005), 477–526.

# Tool demonstration

SMU SINGAPORE MANAGEMENT UNIVERSITY | School of Information Systems



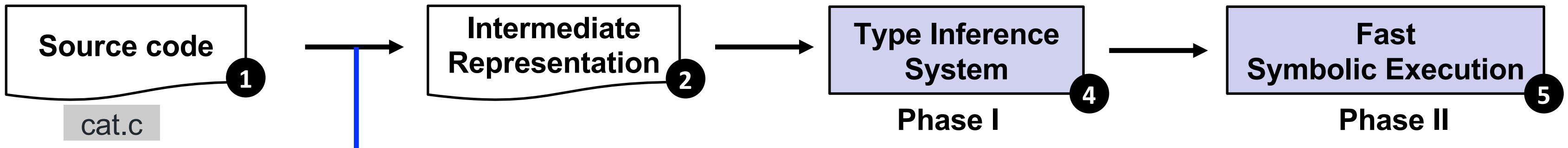

- **Impementation**
  - KLEE [1] and Ccured [2]

Installation of FastKLEE

Full video demo: [https://youtu.be/fjV_a3kt-mo](https://youtu.be/fjV_a3kt-mo)


[1] Cristian Cadar, Daniel Dunbar, and Dawson Engler. 2008. **KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs.** In OSDI'08. USENIX Association, USA, 209–224.
[2] Ccured: George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. 2005. **CCured: type-safe retrofitting of legacy software**. ACM Trans. Program. Lang. Syst. 27, 3 (May 2005), 477–526.

# Tool demonstration




**1. Compile the source code**

```
$ clang -emit-llvm -c cat.c -o cat.bc
```

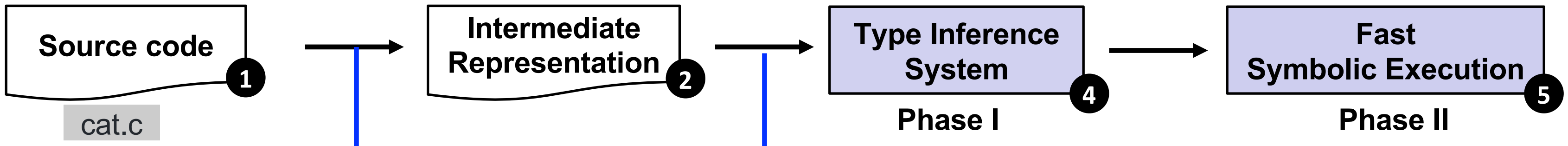
- **Impementation**
  - KLEE [1] and Ccured [2]

Installation of FastKLEE

Full video demo: [https://youtu.be/fjV_a3kt-mo](https://youtu.be/fjV_a3kt-mo)

[1] Cristian Cadar, Daniel Dunbar, and Dawson Engler. 2008. **KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs.** In OSDI'08. USENIX Association, USA, 209–224.
[2] Ccured: George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. 2005. **CCured: type-safe retrofitting of legacy software**. ACM Trans. Program. Lang. Syst. 27, 3 (May 2005), 477–526.

# Tool demonstration




**1. Compile the source code**

```
$ clang -emit-llvm -c cat.c -o cat.bc
```

**2. Instrument IR for type inference**

```
$ llvm-link cat.bcneschecklib.bc -o cat-linked.bc
```

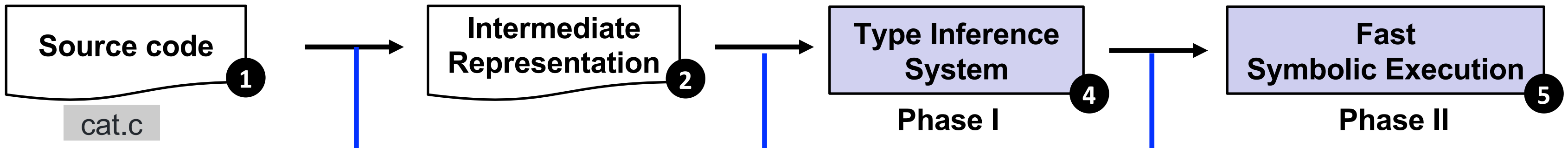
- **Impementation**
  - KLEE [1] and Ccured [2]

Installation of FastKLEE

Full video demo: [https://youtu.be/fjV_a3kt-mo](https://youtu.be/fjV_a3kt-mo)


[1] Cristian Cadar, Daniel Dunbar, and Dawson Engler. 2008. **KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs.** In OSDI'08. USENIX Association, USA, 209–224.
[2] Ccured: George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. 2005. **CCured: type-safe retrofitting of legacy software**. ACM Trans. Program. Lang. Syst. 27, 3 (May 2005), 477–526.

# Tool demonstration




**1. Compile the source code**

`$ clang -emit-llvm -c cat.c -o cat.bc`

**2. Instrument IR for type inference**

`$ llvm-link cat.bc neschecklib.bc -o cat-linked.bc`

**3. Type inference and produce CheckList**

```
$ opt -load libccured.so -nescheck -stats
-time-passes < cat-linked.bc >& /dev/null
```

- **Impementation**
  - KLEE [1] and Ccured [2]

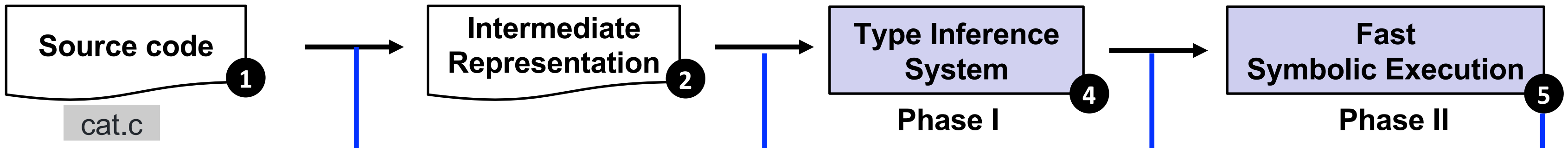
Installation of FastKLEE

Full video demo: https://youtu.be/fjV_a3kt-mo

[1] Cristian Cadar, Daniel Dunbar, and Dawson Engler. 2008. **KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs.** In OSDI'08. USENIX Association, USA, 209–224.
[2] Ccured: George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. 2005. **CCured: type-safe retrofitting of legacy software**. ACM Trans. Program. Lang. Syst. 27, 3 (May 2005), 477–526.

# Tool demonstration

SMU Singapore Management University | School of Information Systems




**1. Compile the source code**

```
$ clang -emit-llvm -c cat.c -o cat.bc
```

**2. Instrument IR for type inference**

```
$ llvm-link cat.bc neschecklib.bc -o cat-linked.bc
```

**3. Type inference and produce CheckList**

```
$ opt -load libccured.so -nescheck -stats
-time-passes < cat-linked.bc >& /dev/null
```

**4. Conduct faster symbolic execution**

```
$ fastklee [options] ./cat.bc --sym-args 0 1
10  --sym-args 0 2 2  --sym-files 1 8
```

- **Impementation**
  - KLEE [1] and Ccured [2]

Installation of FastKLEE

Full video demo: https://youtu.be/fjV_a3kt-mo

[1] Cristian Cadar, Daniel Dunbar, and Dawson Engler. 2008. **KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs.** In OSDI'08. USENIX Association, USA, 209–224.
[2] Ccured: George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. 2005. **CCured: type-safe retrofitting of legacy software**. ACM Trans. Program. Lang. Syst. 27, 3 (May 2005), 477–526.

# Preliminary Evaluation

School of Information Systems

# Preliminary Evaluation

- **Benchmark**
  - GNU Coreutils
  - ~ 1-5k SLOC for each test program

# Preliminary Evaluation

- **Benchmark**
  - GNU Coreutils
  - ~ 1-5k SLOC for each test program

- **Metric**
  - ***Speedups***: the **time** spent on exploring the same number of instructions

$$Speedups : \frac{T_{baseline} - T_{our}}{T_{baseline}} \times 100$$

- $T_{baseline}$ **: existing approach**
- $T_{our}$ **: our approach**

# Preliminary Evaluation

- **Benchmark**
  - GNU Coreutils
  - ~ 1-5k SLOC for each test program

- **Metric**
  - ***Speedups***: the **time** spent on exploring the same number of instructions

$$Speedups : \frac{T_{baseline} - T_{our}}{T_{baseline}} \times 100$$

- $T_{baseline}$ **: existing approach**
- $T_{our}$ **: our approach**

- **Results**

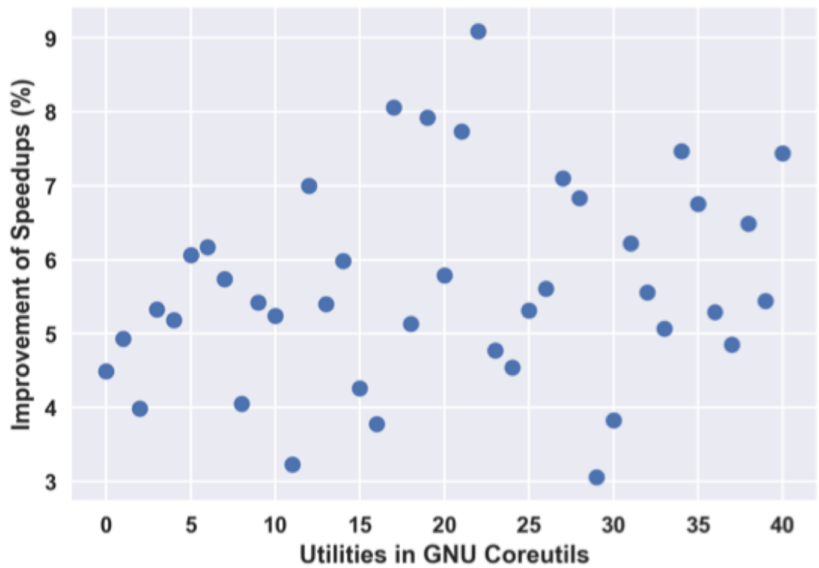



**Figure 2: Scatter plot of the improvement in speedups**

# Preliminary Evaluation

- **Benchmark**
  - GNU Coreutils
  - ~ 1-5k SLOC for each test program

- **Metric**
  - ***Speedups***: the **time** spent on exploring the same number of instructions

$$Speedups : \frac{T_{baseline} - T_{our}}{T_{baseline}} \times 100$$

  - $T_{baseline}$ **: existing approach**
  - $T_{our}$ **: our approach**

- **Results**

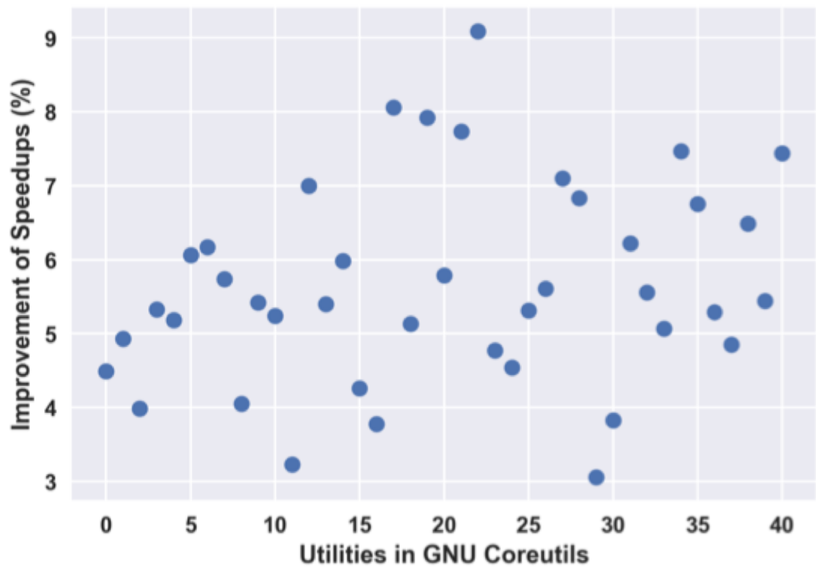
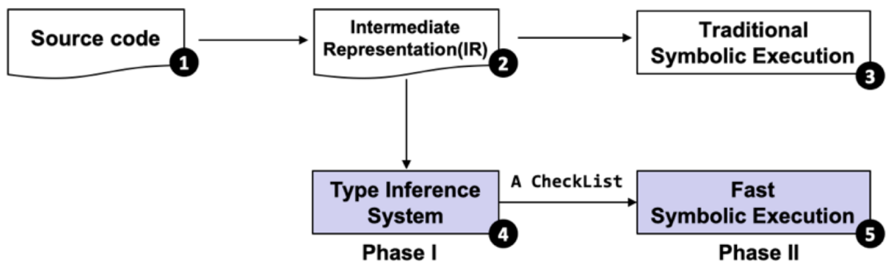



**Figure 2: Scatter plot of the improvement in speedups**

  - FastKLEE can reduce by up to **9.1%** time as the state-of-the-art approach (i.e., KLEE)

# Conclusion

- **Contribution**
  - We present **FastKLEE**, which reduces the interpretation overheads for faster symbolic execution



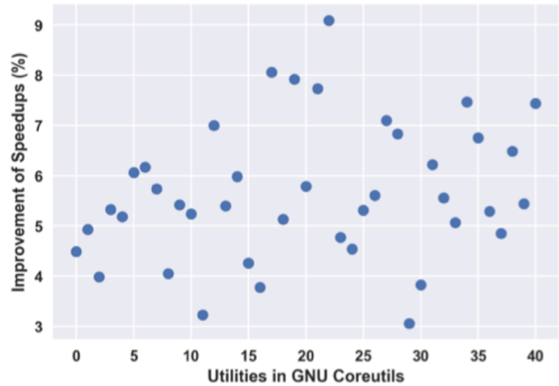




**Figure 2: Scatter plot of the improvement in speedups**

- **Future work**
  - Use FastKLEE to explore more **valuable** execution paths in software systems
    - valuable: vulnerable and exploitable

Code: https://github.com/haoxintu/FastKLEE
Video demo: https://youtu.be/fjV_a3kt-mo
Email: haoxintu.2020@phdcs.smu.edu.sg
(Please feel free to pull requests or raise any questions if you have!)