# 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)

# Outlines

SMU SINGAPORE MANAGEMENT UNIVERSITY | School of Information Systems

- **Background**
- **Motivation**
- **Solution**
  - FastKLEE
- **Preliminary Evaluation**
- **Conclusion**

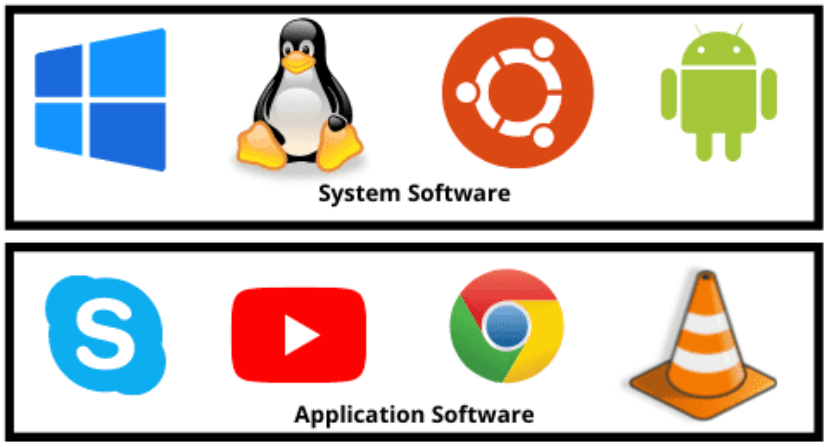
# Background

SMU SINGAPORE MANAGEMENT UNIVERSITY | School of Information Systems

# Background

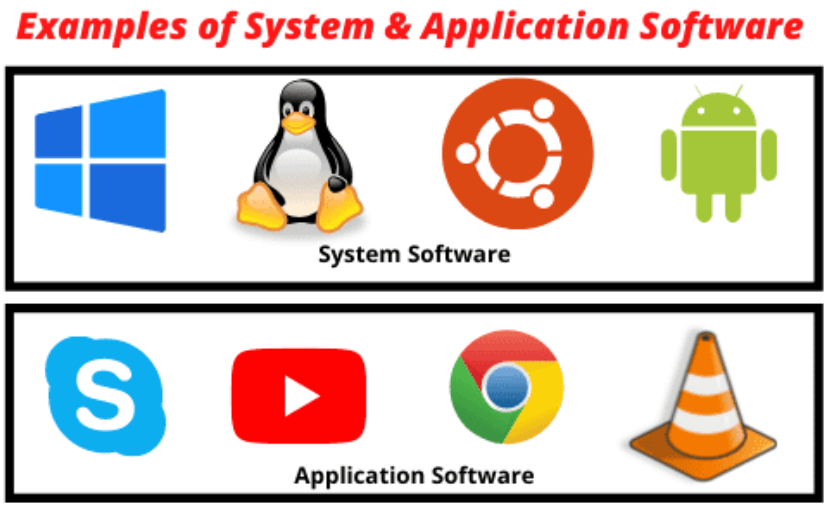


***Examples of System & Application Software***



# Background

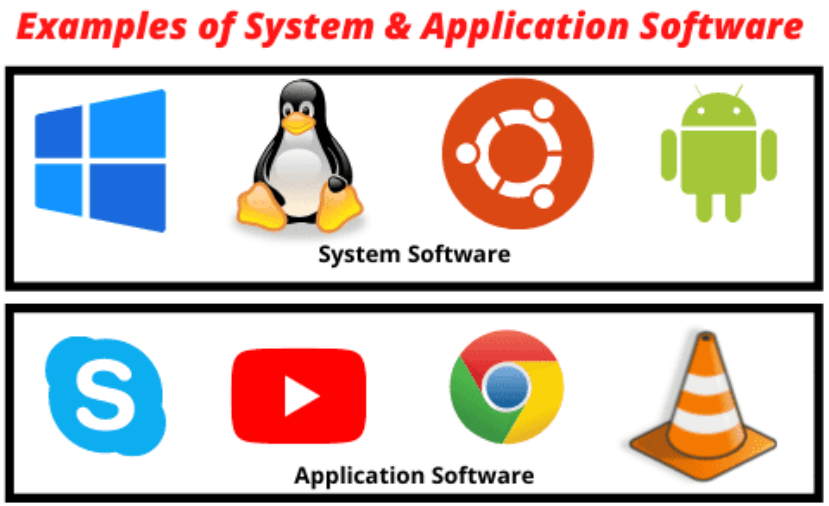





- **Q: How to improve the quality of software?**

# Background

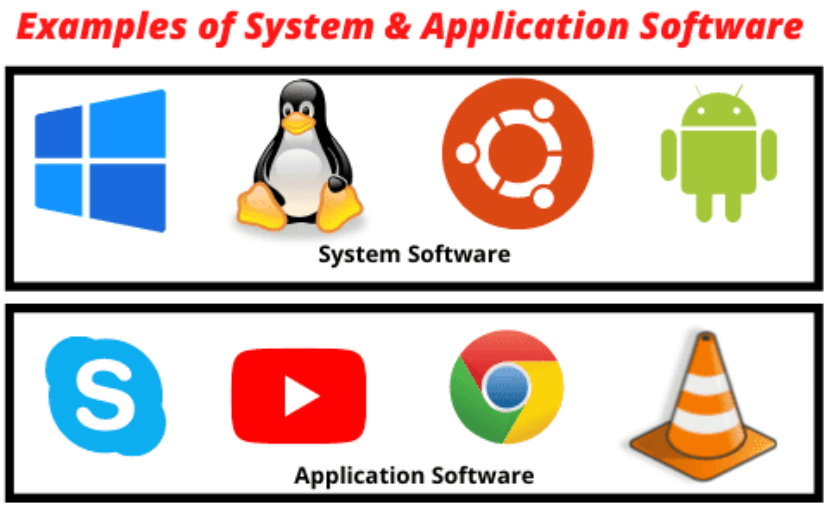





- Q: How to improve the quality of software?
- A: Software testing

# Background



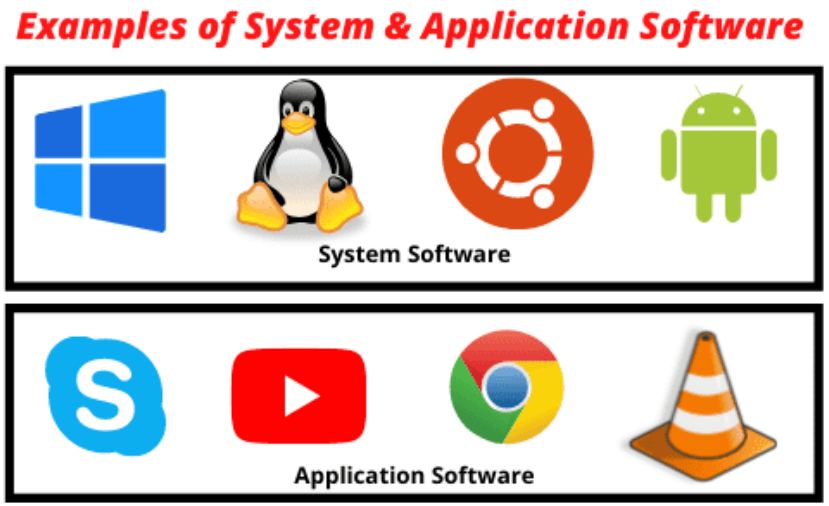



- Q: How to improve the quality of software?
- A: Software testing

↓

(Symbolic Execution)

# Background



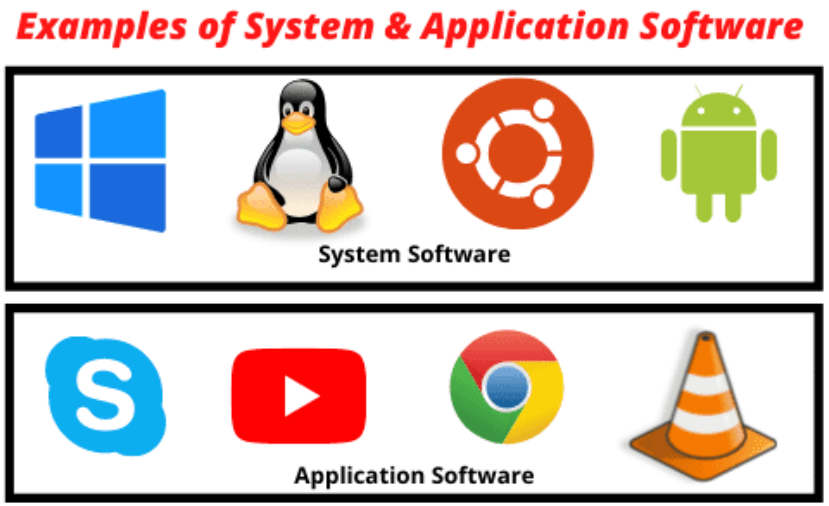

- Q: How to improve the quality of software?
- A: Software testing

↓

(Symbolic Execution)


A General Workflow of Symbolic Execution (SE)

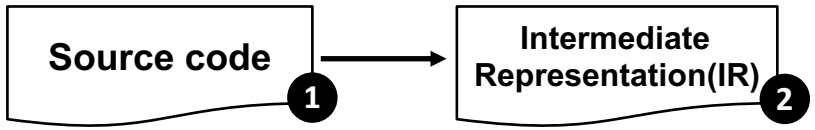
# Background




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

↓

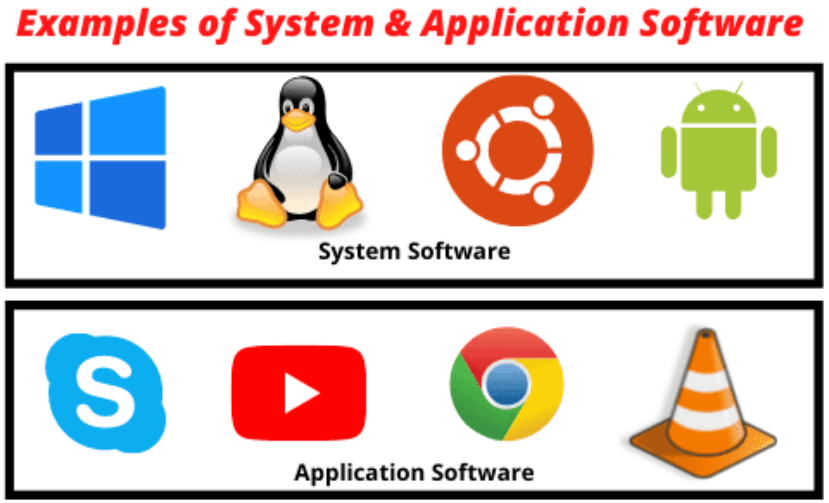
(Symbolic Execution)




A General Workflow of Symbolic Execution (SE)

# Background

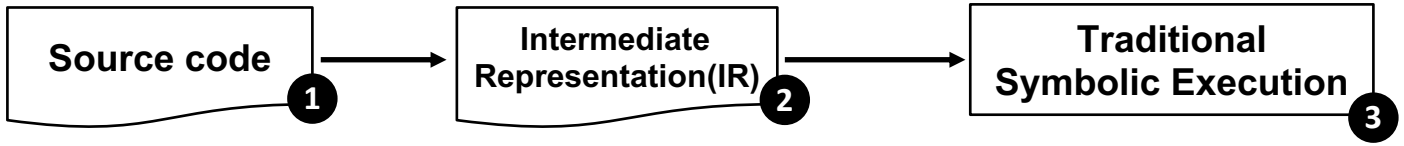





- Q: How to improve the quality of software?
- A: Software testing

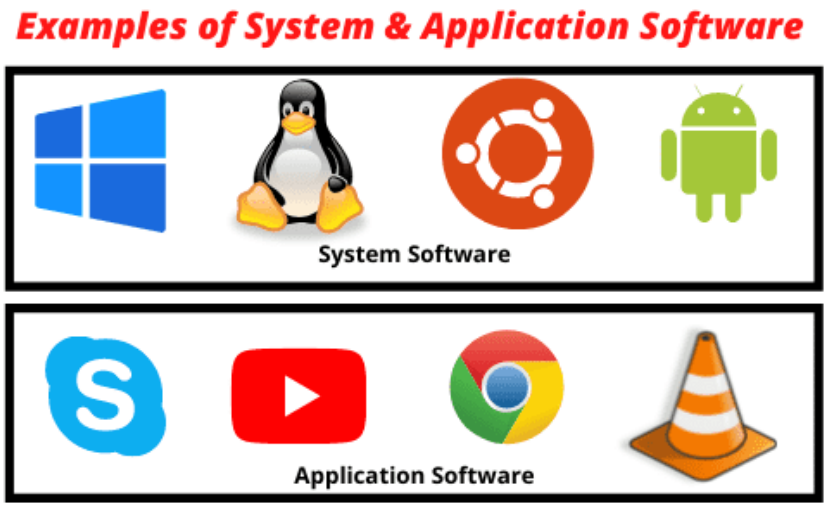
↓

(Symbolic Execution)




A General Workflow of Symbolic Execution (SE)

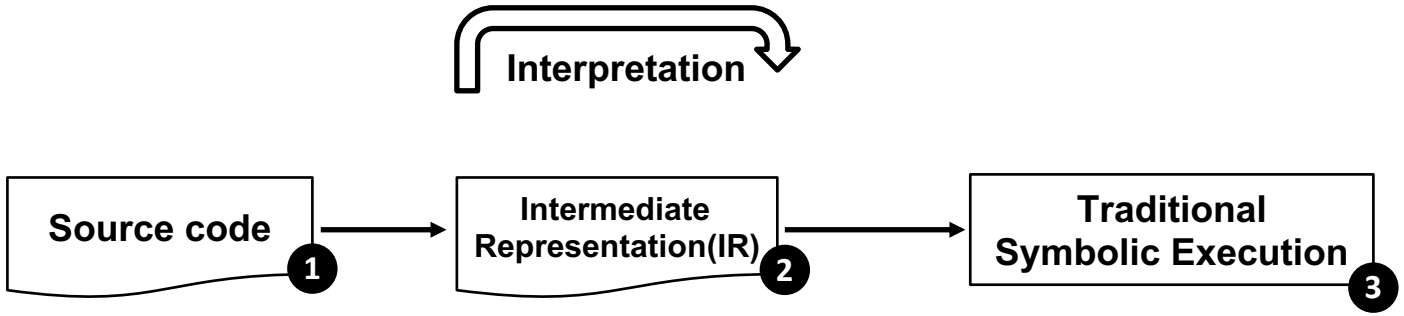
# Background




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

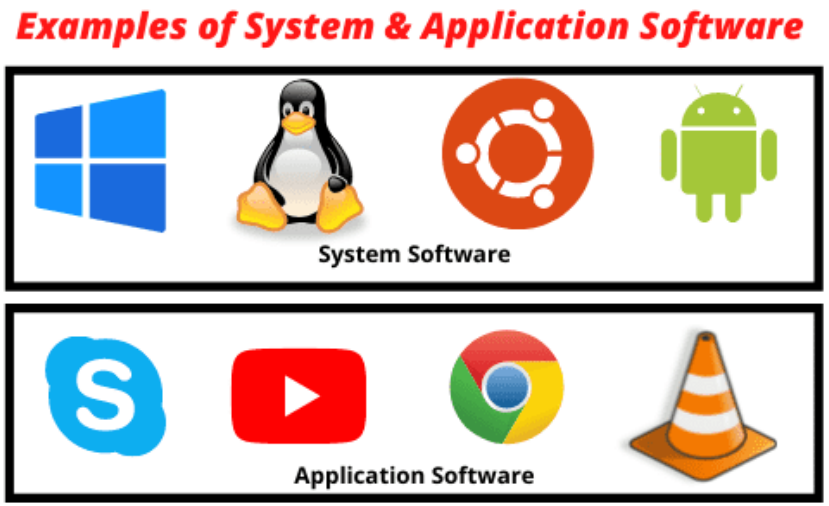
↓

(Symbolic Execution)




A General Workflow of Symbolic Execution (SE)

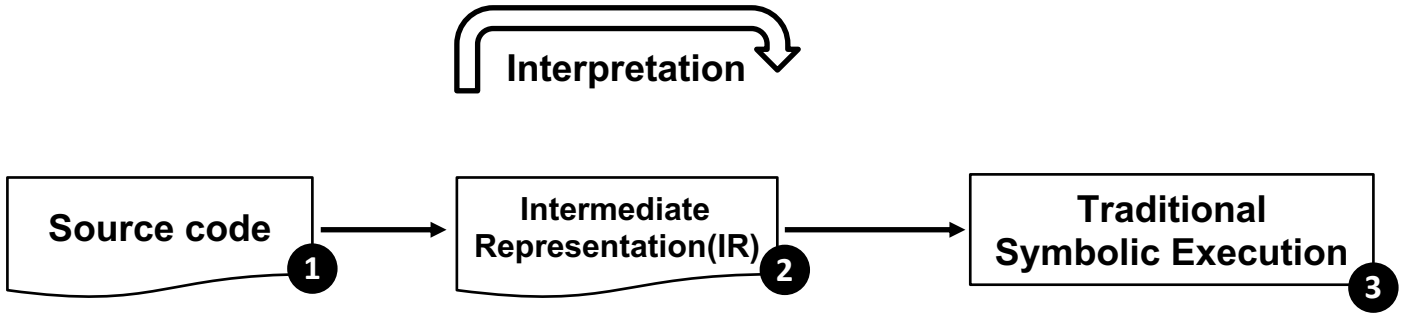
# Background




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

↓

(Symbolic Execution)




A General Workflow of Symbolic Execution (SE)

One of the major concerns: Performance

# Motivation

# Motivation

- **Related work**

# Motivation

- **Related work**
  - Compiler optimization-based
    - Existing compiler optimization for accelerating symbolic execution [6,7]
    - Customized optimization for fast verification [8]

# Motivation

SMU SINGAPORE MANAGEMENT UNIVERSITY | School of Information Systems

- **Related work**
  - Compiler optimization-based
    - Existing compiler optimization for accelerating symbolic execution [6,7]
    - Customized optimization for fast verification [8]
  - Others
    - Reducing path exploration and speeding up constraint solving

# Motivation

SMU SINGAPORE MANAGEMENT UNIVERSITY | School of Information Systems

- **Related work**
  - Compiler optimization-based
    - Existing compiler optimization for accelerating symbolic execution [6,7]
    - Customized optimization for fast verification [8]
  - Others
    - Reducing path exploration and speeding up constraint solving

  - Limitations: they did not consider that interpreting alone can have performance downsides

# Motivation

- **Related work**
  - Compiler optimization-based
    - Existing compiler optimization for accelerating symbolic execution [6,7]
    - Customized optimization for fast verification [8]
  - Others
    - Reducing path exploration and speeding up constraint solving
  - Limitations: they did not consider that interpreting alone can have performance downsides
- **Summary**
  - Performance issues still remain in existing interpreting-based SE engines
  - Very few of them take the interpreting overhead alone into account

# Motivation

SMU Singapore Management University | School of Information Systems

- **Related work**
  - Compiler optimization-based
    - Existing compiler optimization for accelerating symbolic execution [6,7]
    - Customized optimization for fast verification [8]
  - Others
    - Reducing path exploration and speeding up constraint solving
  - Limitations: they did not consider that interpreting alone can have performance downsides
- **Summary**
  - Performance issues still remain in existing interpreting-based SE engines
  - Very few of them take the interpreting overhead alone into account

**How can we reduce interpretation overheads for fast symbolic execution?**

# Solution – FastKLEE (1/2)

# Solution – FastKLEE (1/2)

SMU SINGAPORE MANAGEMENT UNIVERSITY | School of Information Systems

- **Two key insights**

# Solution – FastKLEE (1/2)

SMU SINGAPORE MANAGEMENT UNIVERSITY | School of Information Systems

- **Two key insights**
  - **Insight 1:** The number of interpreted instructions tends to be stupendous

# Solution – FastKLEE (1/2)

SMU SINGAPORE MANAGEMENT UNIVERSITY | School of Information Systems

- **Two key insights**
  - **Insight 1:** The number of interpreted instructions tends to be stupendous
    - 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
```

# Solution – FastKLEE (1/2)

SMU SINGAPORE MANAGEMENT UNIVERSITY | School of Information Systems

- **Two key insights**
  - **Insight 1:** The number of interpreted instructions tends to be stupendous
    - several billion only in one hour run
  - **Insight 2:** Only a small portion of pointers need bound checking during 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
```

# Solution – FastKLEE (1/2)

SMU SINGAPORE MANAGEMENT UNIVERSITY | School of Information Systems

- **Two key insights**
  - **Insight 1:** The number of interpreted instructions tends to be stupendous
    - several billion only in one hour run
  - **Insight 2:** Only a small portion of pointers need bound checking during execution
    - Reduce interpreting overhead of most frequently interpreted ones (i.e., load/store instruction)

```
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
```

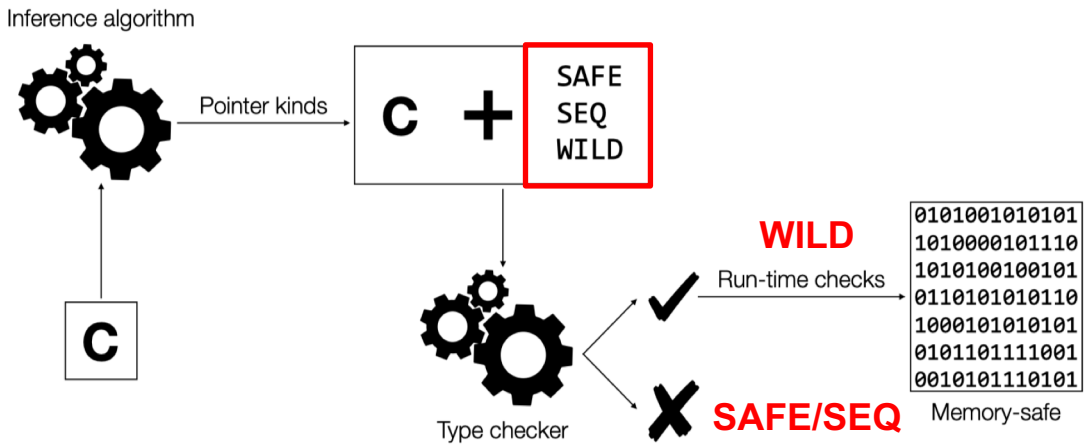
# Solution – FastKLEE (1/2)

SMU Singapore Management University | School of Information Systems

- **Two key insights**
  - **Insight 1:** The number of interpreted instructions tends to be stupendous
    - several billion only in one hour run
  - **Insight 2:** Only a small portion of pointers need bound checking during execution
    - Reduce interpreting overhead of most frequently interpreted ones (i.e., load/store instruction)
    - Inspired by ***Type Inference*** system

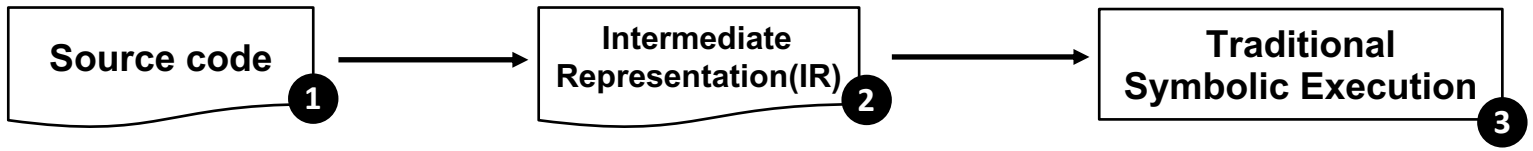
```
Elapsed: 01:00:04
KLEE: done: explored paths = 125017
KLEE: done: avg. constructs per query = 74
KLEE: done: total queries = 8859
KLEE: done: valid queries = 6226
KLEE: done: invalid queries = 2633
KLEE: done: query cex = 8859

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

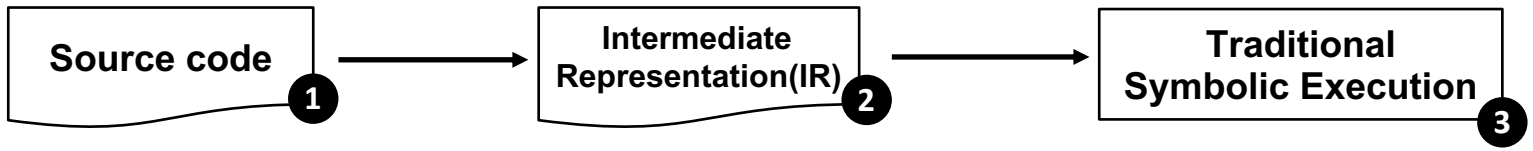


# Solution – FastKLEE (2/2)

# Solution – FastKLEE (2/2)

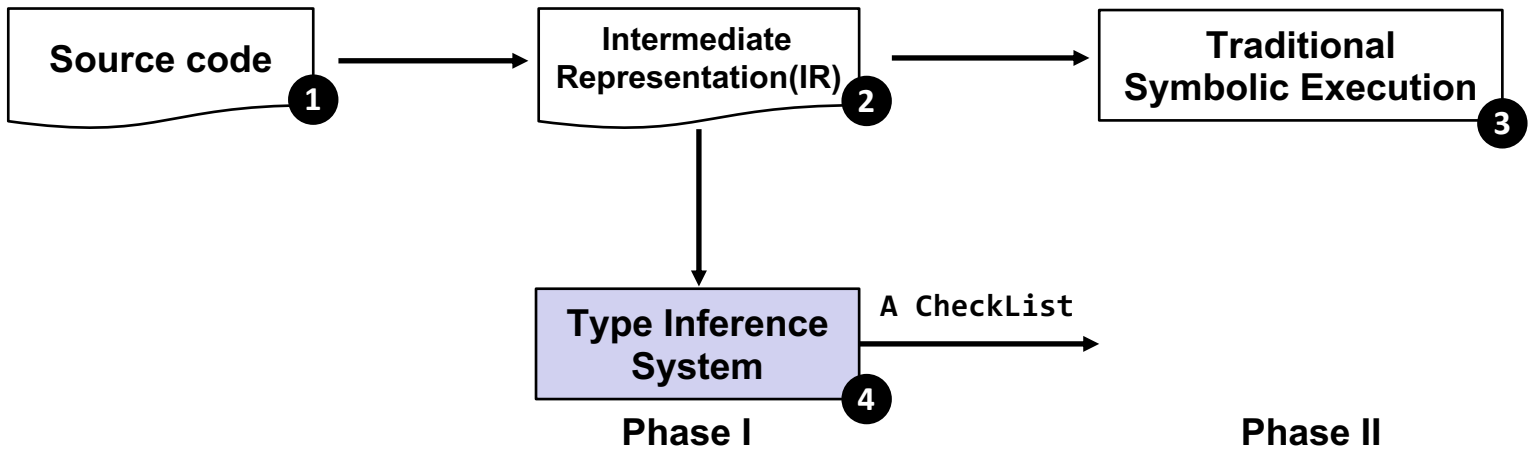


# Solution – FastKLEE (2/2)



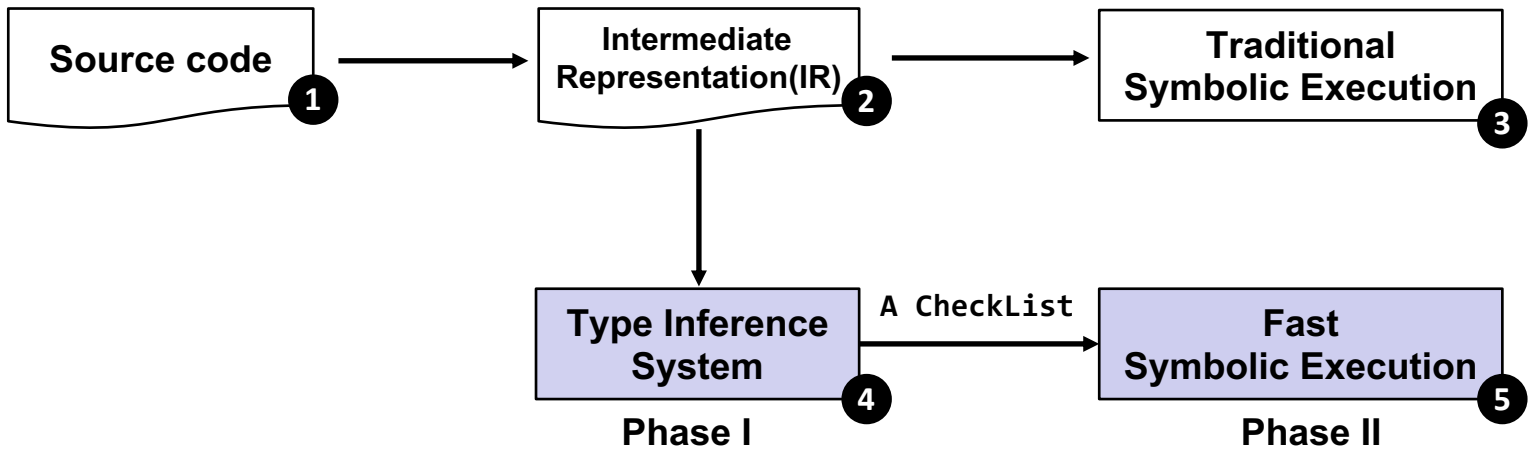

**Phase I** **Phase II**

# Solution – FastKLEE (2/2)




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

# Solution – FastKLEE (2/2)




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

❺ • **Phase II**: Conduct **Customized Instruction Operation** in Fast SE
  – Only perform checking for ***Unsafe*** instructions during interpretation

# Preliminary Evaluation

# Preliminary Evaluation

SMU SINGAPORE MANAGEMENT UNIVERSITY | School of Information Systems

- **Impementation**
  - KLEE [1] and Ccured [4]
- **Benchmark**
  - GNU Coreutils

# Preliminary Evaluation

- **Impementation**
  - KLEE [1] and Ccured [4]
- **Benchmark**
  - GNU Coreutils
- **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

- **Impementation**
  - KLEE [1] and Ccured [4]
- **Benchmark**
  - GNU Coreutils
- **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**

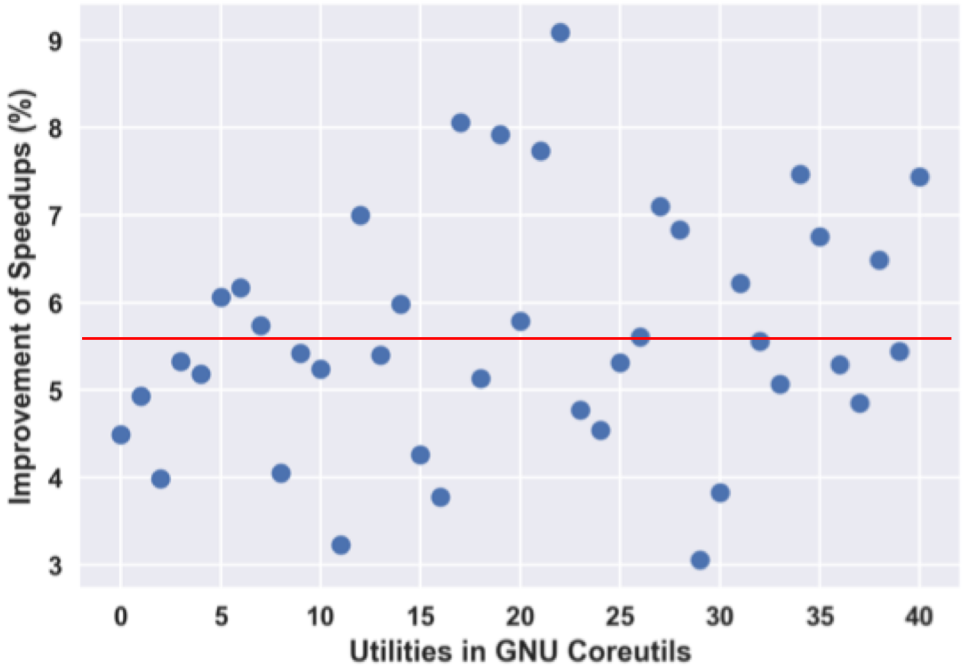



**Figure 2: Scatter plot of the improvement in speedups**

# Preliminary Evaluation

- **Impementation**
  - KLEE [1] and Ccured [4]
- **Benchmark**
  - GNU Coreutils
- **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**

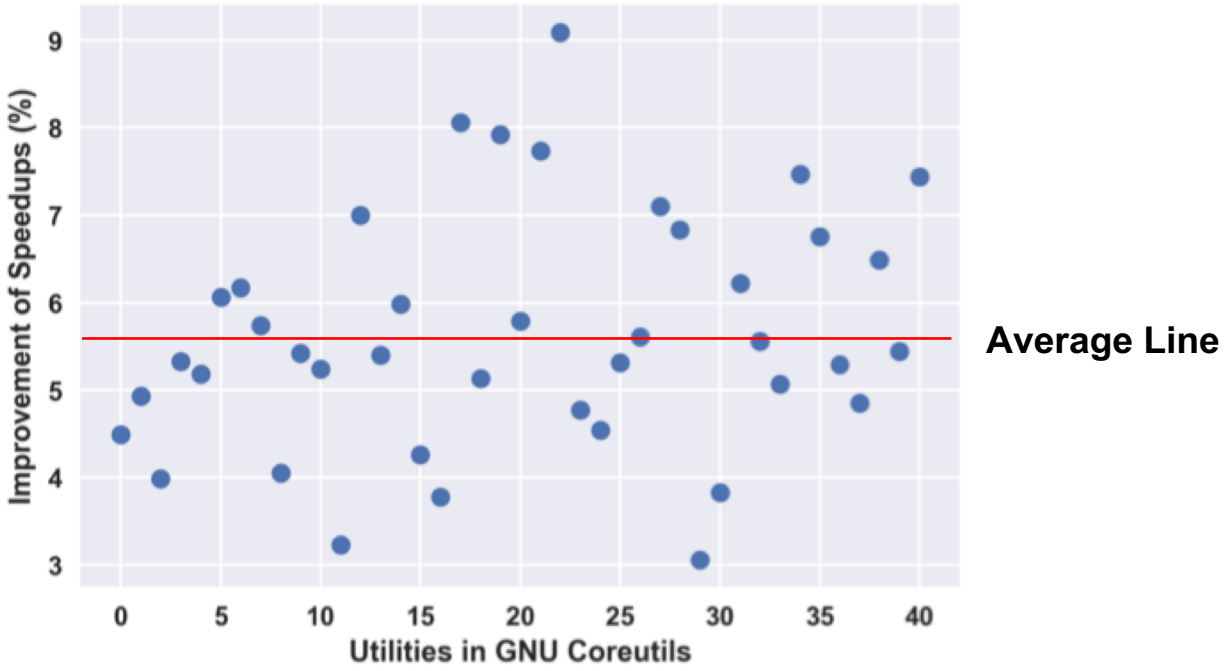
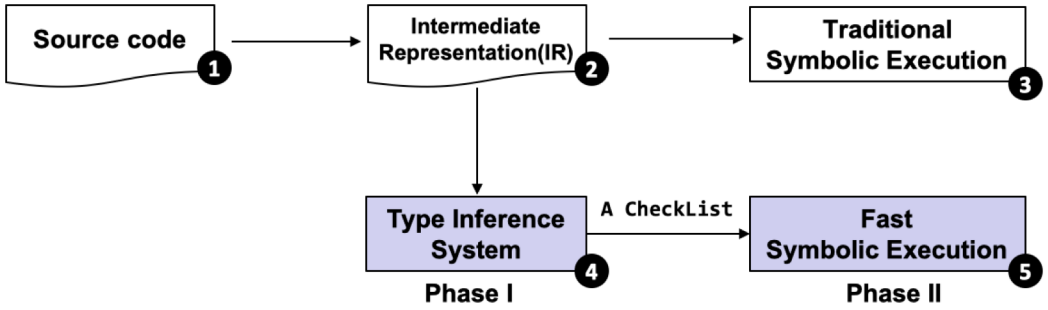



**Figure 2: Scatter plot of the improvement in speedups**

  - FastKLEE is able to reduce by up to **9.1% (5.6% on average)** time as the state-of-the-art approach

# Conclusion

- **Contribution**
  - We present **FastKLEE**, which aims to reduce the interpretation overheads for fast symbolic execution



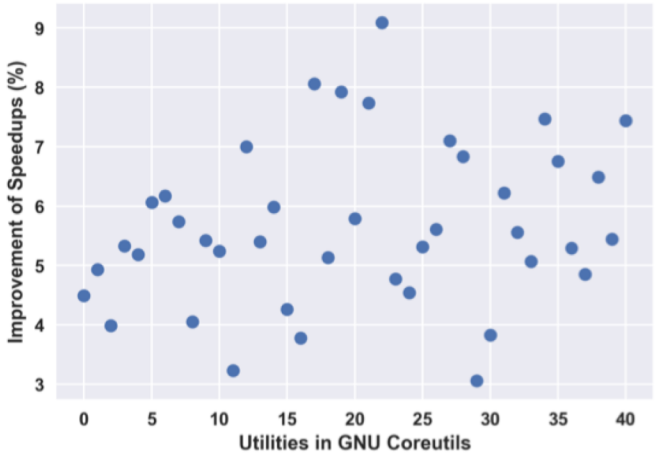




Figure 2: Scatter plot of the improvement in speedups

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

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

# References


[1] Cristian Cadar, Daniel Dunbar, and Dawson Engler. 2008. **KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs.** In Proceedings of the 8th USENIX conference on Operating systems design and implementation (OSDI'08). USENIX Association, USA, 209–224.

[2] V. Chipounov, V. Kuznetsov, and G. Candea, **S2E: a platform for in-vivo multi-path analysis of software systems**, in Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems, New York, NY, USA, Mar. 2011, pp. 265–278.

[3] Y. Shoshitaishvili et al., **SOK: (State of) The Art of War: Offensive Techniques in Binary Analysis**, in 2016 IEEE Symposium on Security and Privacy (SP), May 2016, pp. 138–157.

[4] 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.

[5] S. Poeplau and A. Francillon, **Symbolic execution with SymCC: Don't interpret, compile!,** in 29th USENIX Security Symposium, 2020, pp. 181–198.

[6] Dong, Shiyu, Oswaldo Olivo, Lingming Zhang, and Sarfraz Khurshid. **Studying the influence of standard compiler optimizations on symbolic execution**, In 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2015.

[7] Chen, Junjie, Wenxiang Hu, Lingming Zhang, Dan Hao, Sarfraz Khurshid, and Lu Zhang. **Learning to accelerate symbolic execution via code transformation**, In 32nd European Conference on Object-Oriented Programming (ECOOP 2018).

[8] Wagner, Jonas, Volodymyr Kuznetsov, and George Candea. **{-OVERIFY}: Optimizing Programs for Fast {Verification},** In 14th Workshop on Hot Topics in Operating Systems (HotOS XIV). 2013.

# Thanks you!

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

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

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

# Backup: other evaluations

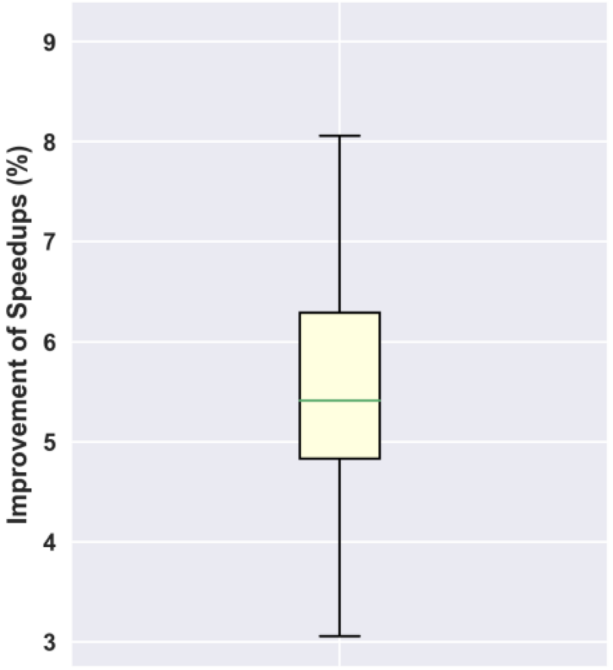



**Figure 4: Box plot of the improvement in speedups**

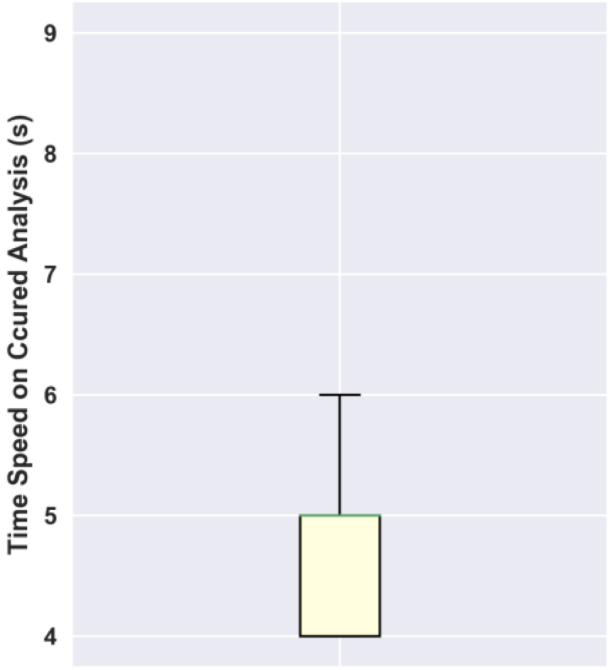
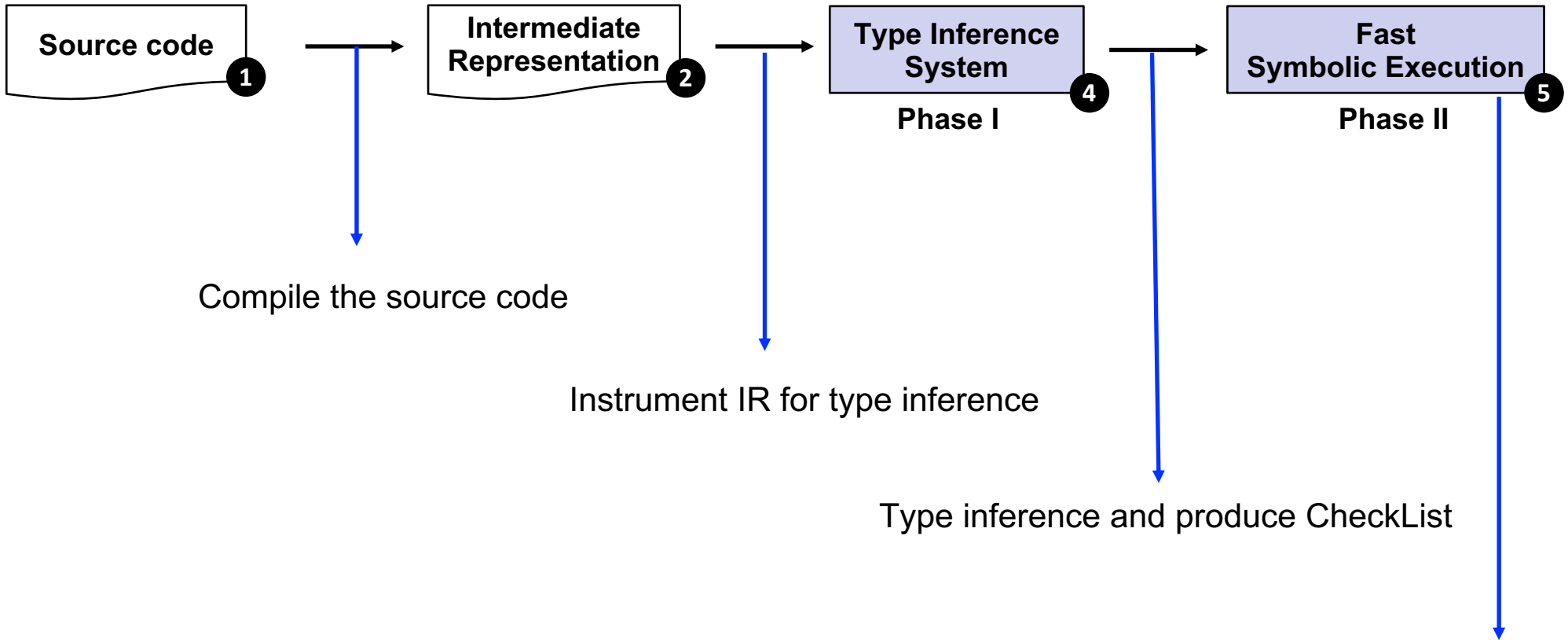



**Figure 5: Box plot of the time spent on type inference**

# Backup: Tool Usage

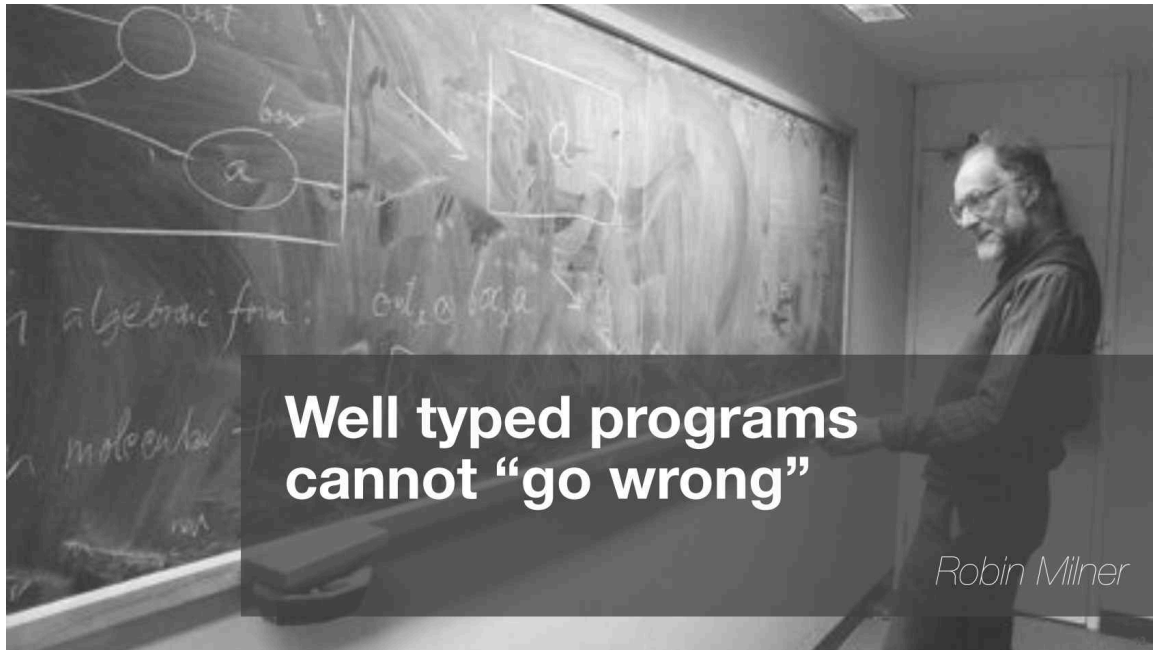



Installation of FastKLEE

Code: https://github.com/haoxintu/FastKLEE
(Please feel free to pull requests and raise any questions if you have!)

# Background: type safety (1)

- What is type safety?
  - Type safety means that the compiler will validate types while compiling, and throw an error if assign the wrong type to a variable.




Source: What is Type-safety?

Java

```
int x = 100;
String y = "hello";
int z = x + y;
```

**Error: incompatible types:**
**String cannot be converted to int**

Type ***safe*** programming language

```
int x = 100;
char* y = "hello";
int z = x + y;
```

**No errors**

Type ***unsafe*** programming language

**Note: type safety was intentionally sacrificed for flexibility/performance in C**

# Backup: type inference (2)

- **Type inference**
  - General Definition: automatic detection of the type of an expression in a programming language
  - Type inference in C
    - Try to keep type safety (specifically for pointer types)

- **Ccured** [4]
  - CCured transforms C programs to achieve memory safety guarantees
    - A large portion of pointers can be statically verified to be safe, and only small need run-time check