Understanding the Interleaving-Space Overlap across Inputs and Software Versions

Dongdong Deng, **Wei Zhang**, Borui Wang, Peisen Zhao, Shan Lu

University of Wisconsin, Madison



Concurrency bug detection is critical

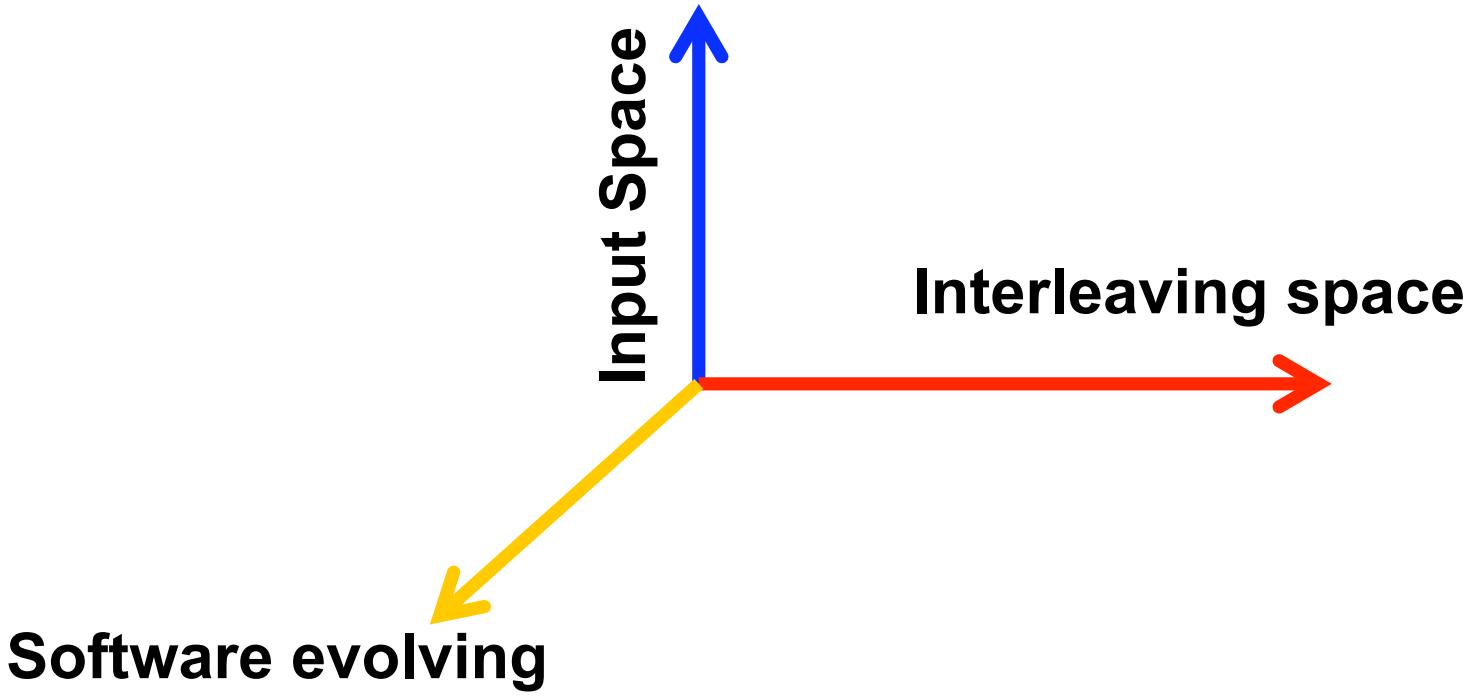
- Concurrency bugs are a big problem
  - Widely exist and cause severe damage [Northeast blackout, NasDaq]

- They only get worse
  - Multi-core machines are prevalent

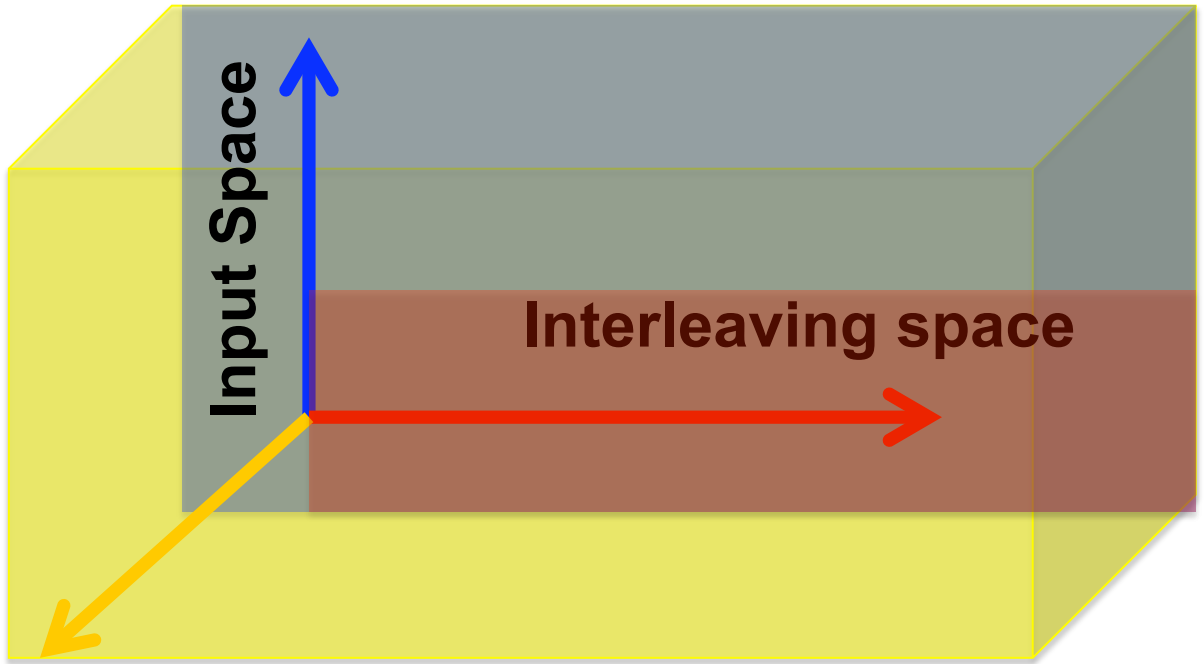
Concurrency bug detection is costly

- Data race detection incurs 10-200X slowdown
- Software many inputs to test
- Test all inputs again upon new version

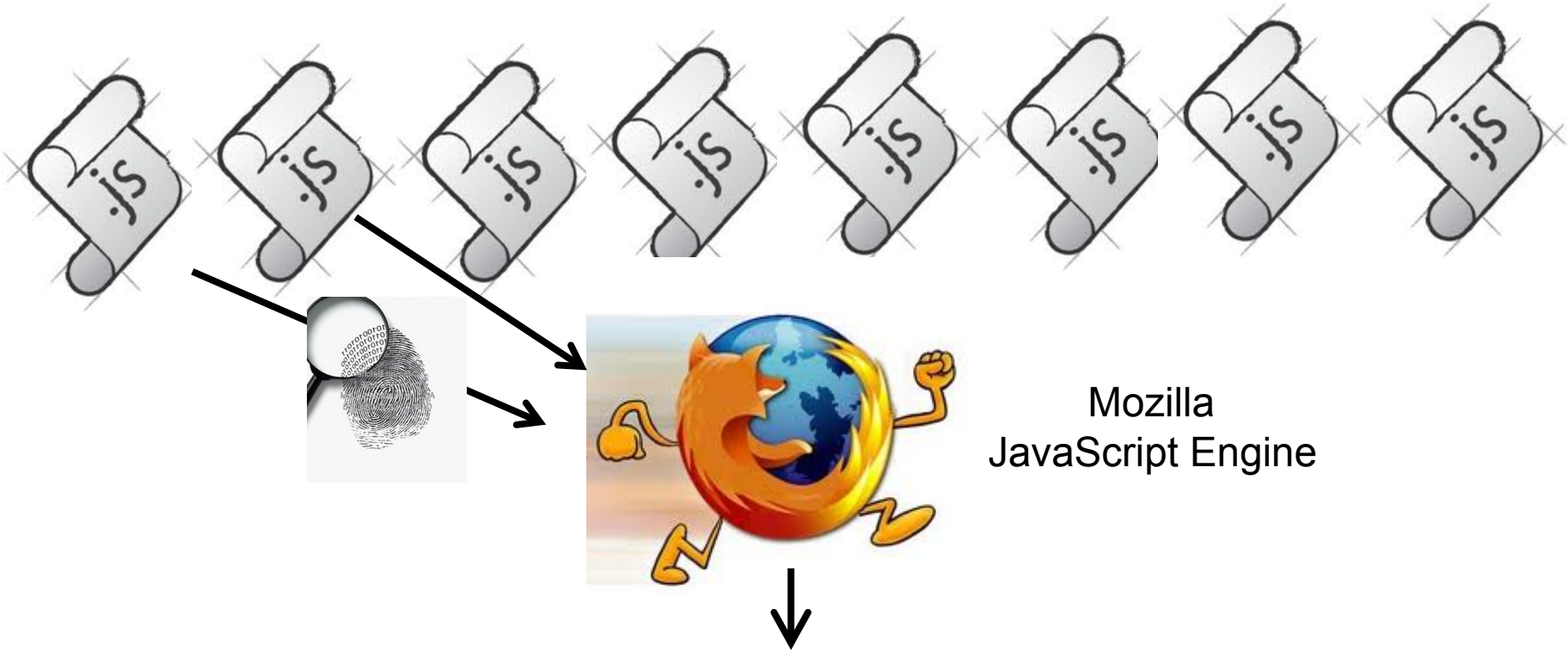
Software companies cannot afford thorough testing



How to speed up?



Redundancy across inputs (example)



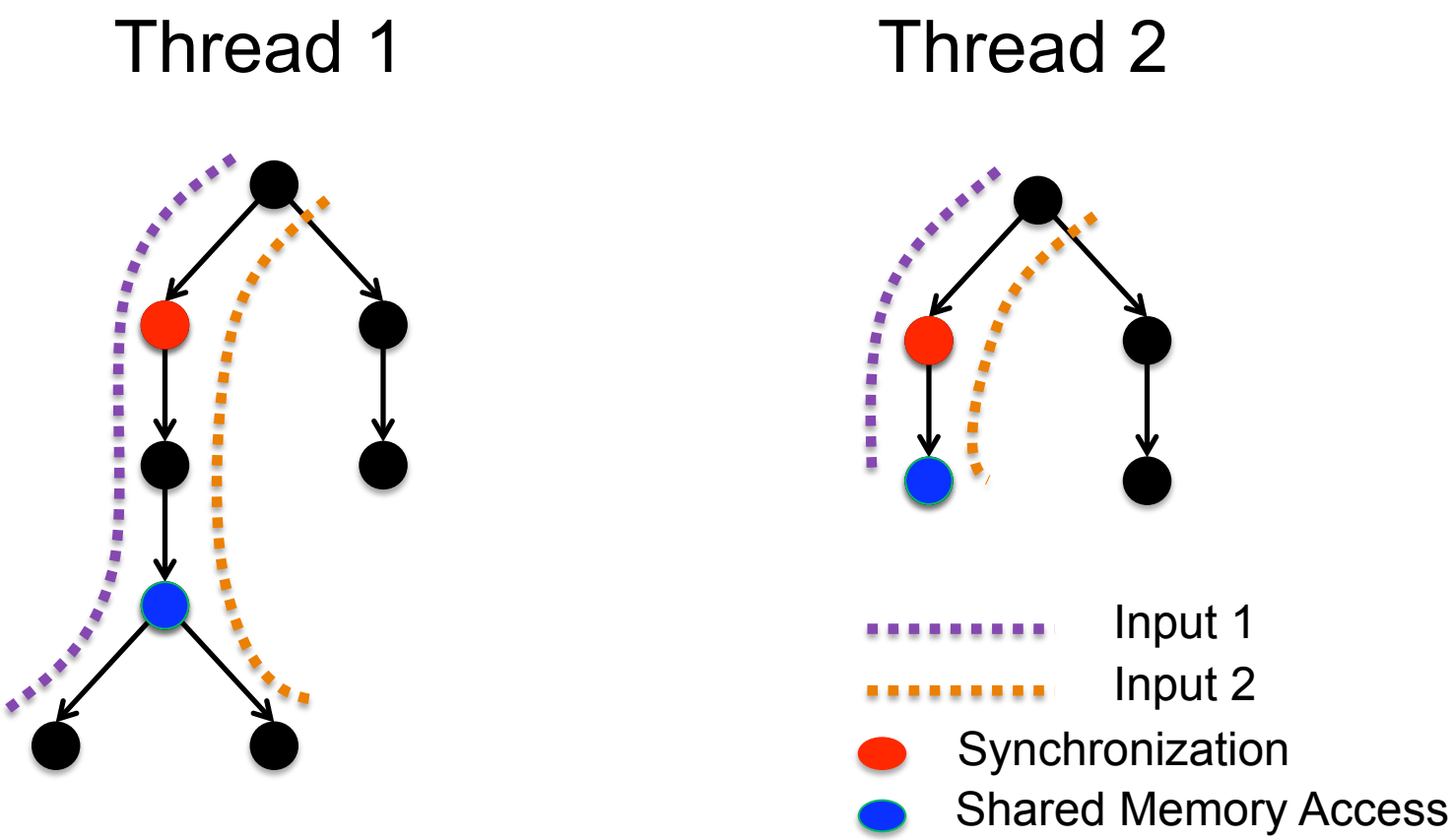
Thread 1

```
js_InitAtomState(…){
    state->table = …;
    if(state->table)
      iterate state->table;
}
```

Thread 2

```
js_FreeAtomState(..){
    …;
    state->table = NULL;
}
```

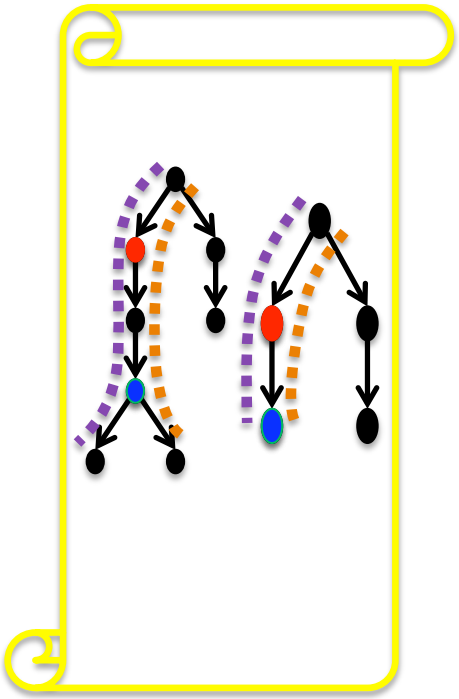
Redundancy across inputs



How to find inputs' interleaving space overlap before heavy-weight detection?

We have an opportunity

Interleaving Space Overlap across Inputs





Race Detection

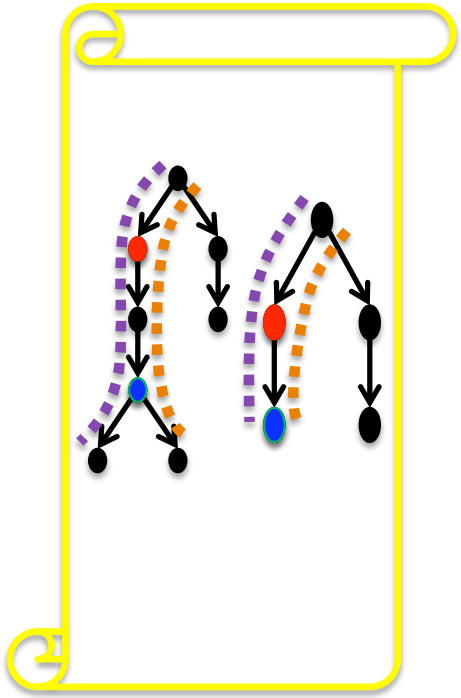


We have a problem

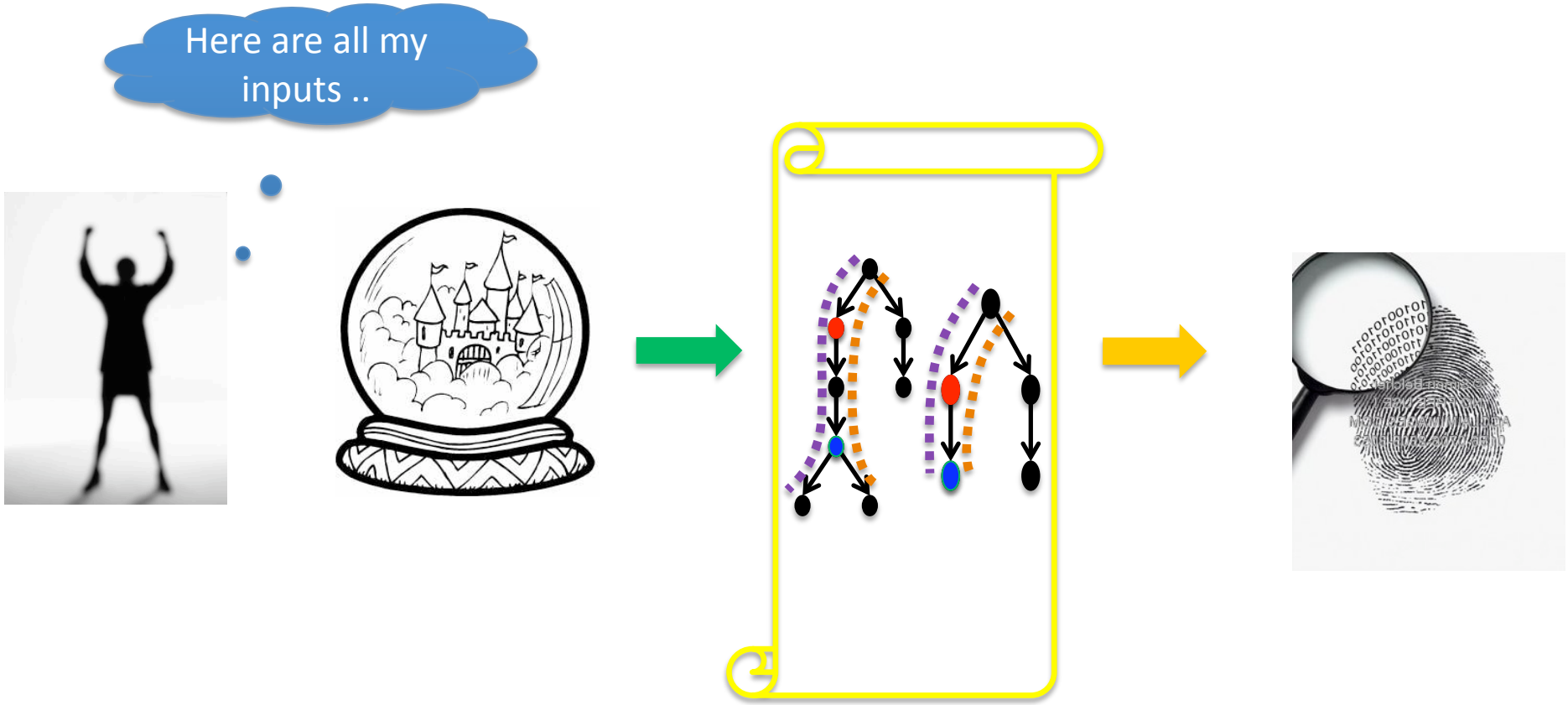
Interleaving Space Analysis



Interleaving Space Overlap across Inputs



A solution



Contributions

- Characteristics study of bug detection redundancy
  - Among different inputs
  - Among different versions
- A good metric to approximate interleaving space

  Concurrent Function Pairs
  - Reasonably accurate in depicting interleaving space
  - Lightweight in measurement
- CFP-guided race detection
- Similar ideas can be applied to regression test, etc.

Outline

- Motivation
- Challenges
- Characteristics study
- Solution in details
- Experiment Evaluation
- Conclusion

Characteristics Study Methodology

- 5 benchmarks written in C/C++
  - Aget, Click, FFT, Mozilla and Pbzip2
- One input set for each benchmark
  - Developer designed: Click and Mozilla
  - We designed: Aget, FFT and Pbzip2
- Detect data races and atomicity violations
  - Exposed by every input
  - Exposed by every version

Characteristics study

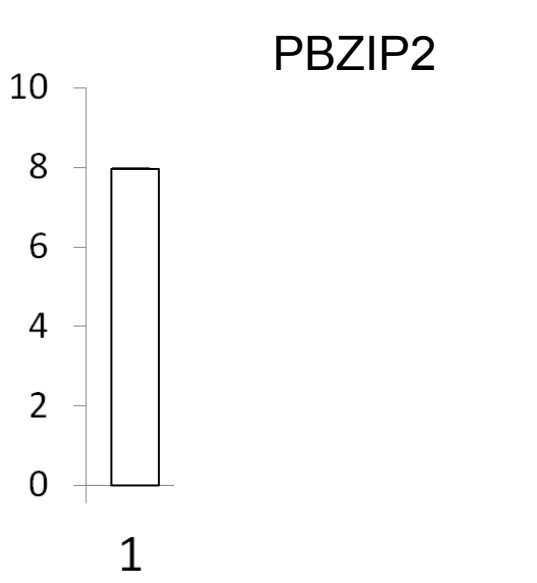
Y-axis # of bugs

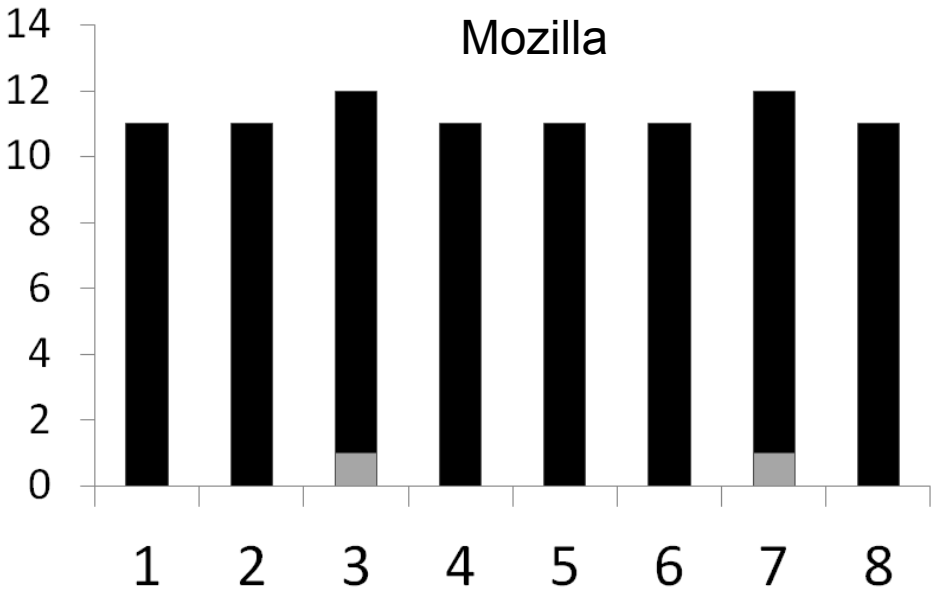
X-axis Input id

Bug detected by 2-4 inputs

Bug detected by 8 inputs

Harmful Data Race





Please refer to our paper for more data

Characteristics study

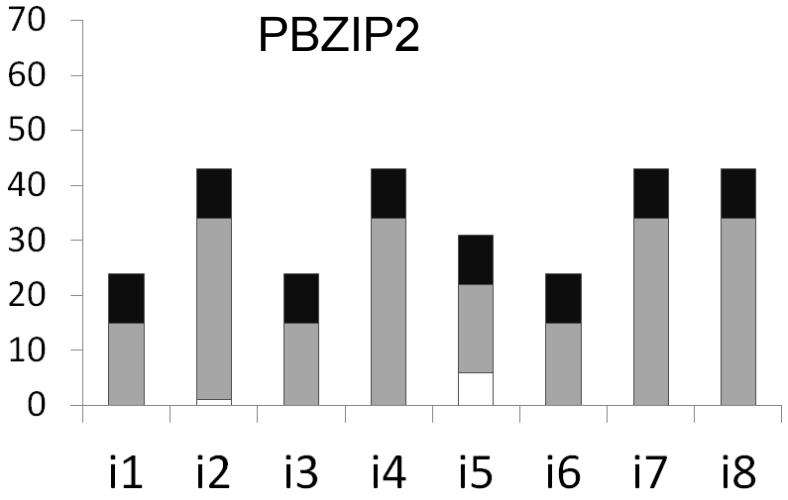
Y-axis # of bugs

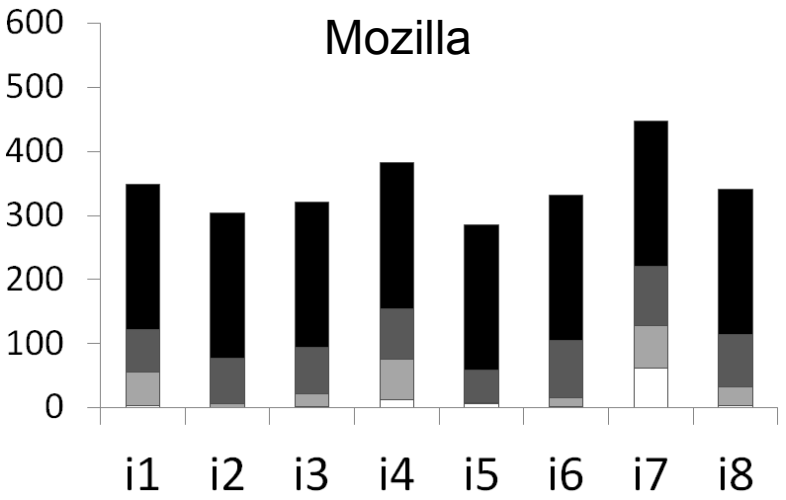
X-axis Input id

Bug detected by 2-4 inputs

Bug detected by 8 inputs

Data Race (Benign & Harmful)





Please refer to our paper for more data

Characteristics study

Y-axis # of bugs

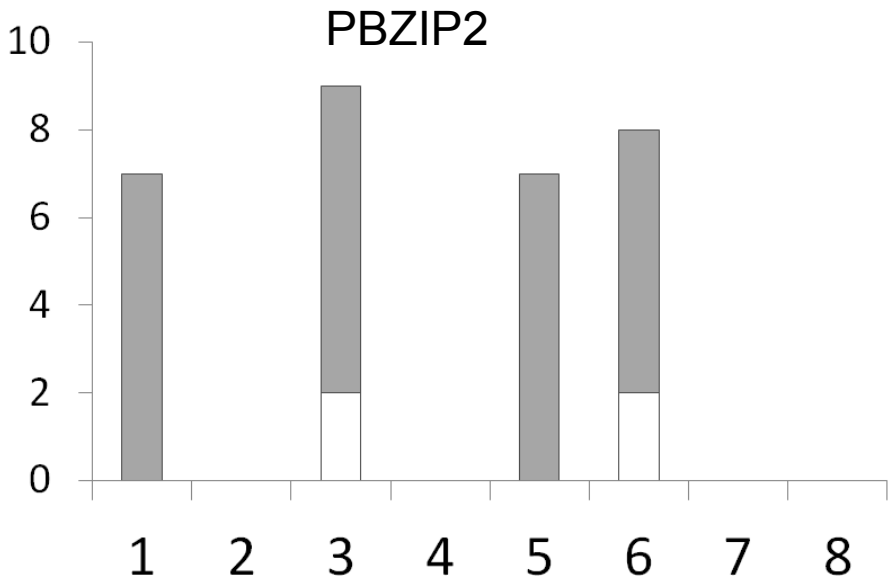
X-axis Input id

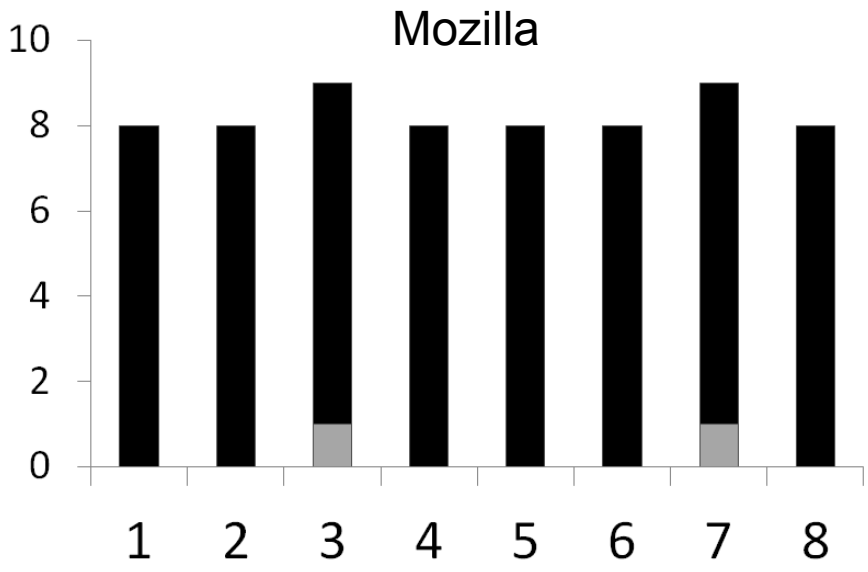
Bug detected by 2-4 inputs

Bug detected by 8 inputs

Bug detected by 1 input

Harmful Atomicity Violation





Please refer to our paper for more data

Characteristics study

Y-axis # of bugs

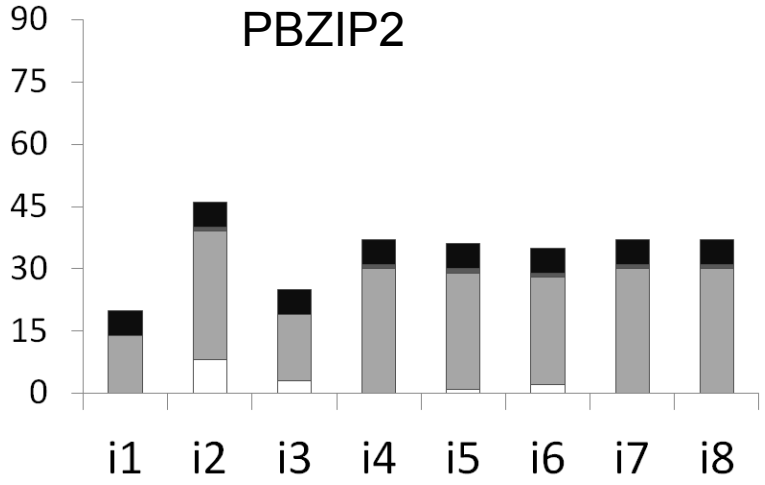
X-axis Input id

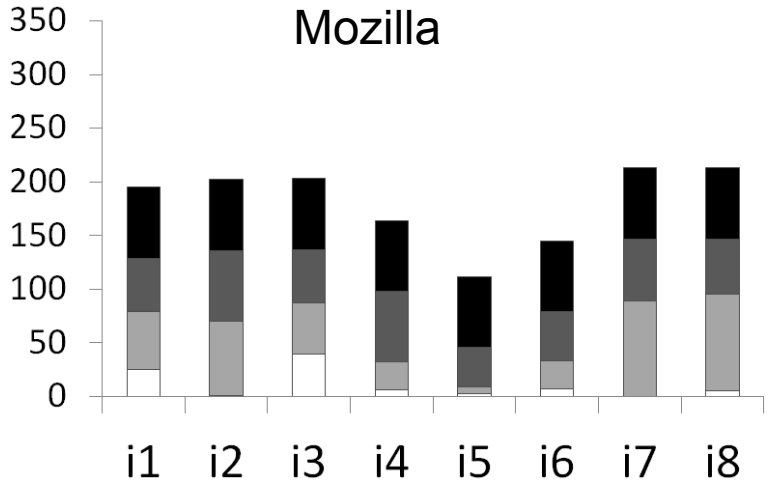
Bug detected by 2-4 inputs

Bug detected by 8 inputs

Bug detected by 1 input

Atomicity Violation (Benign & Harmful)





Please refer to our paper for more data

Summary of characteristics study

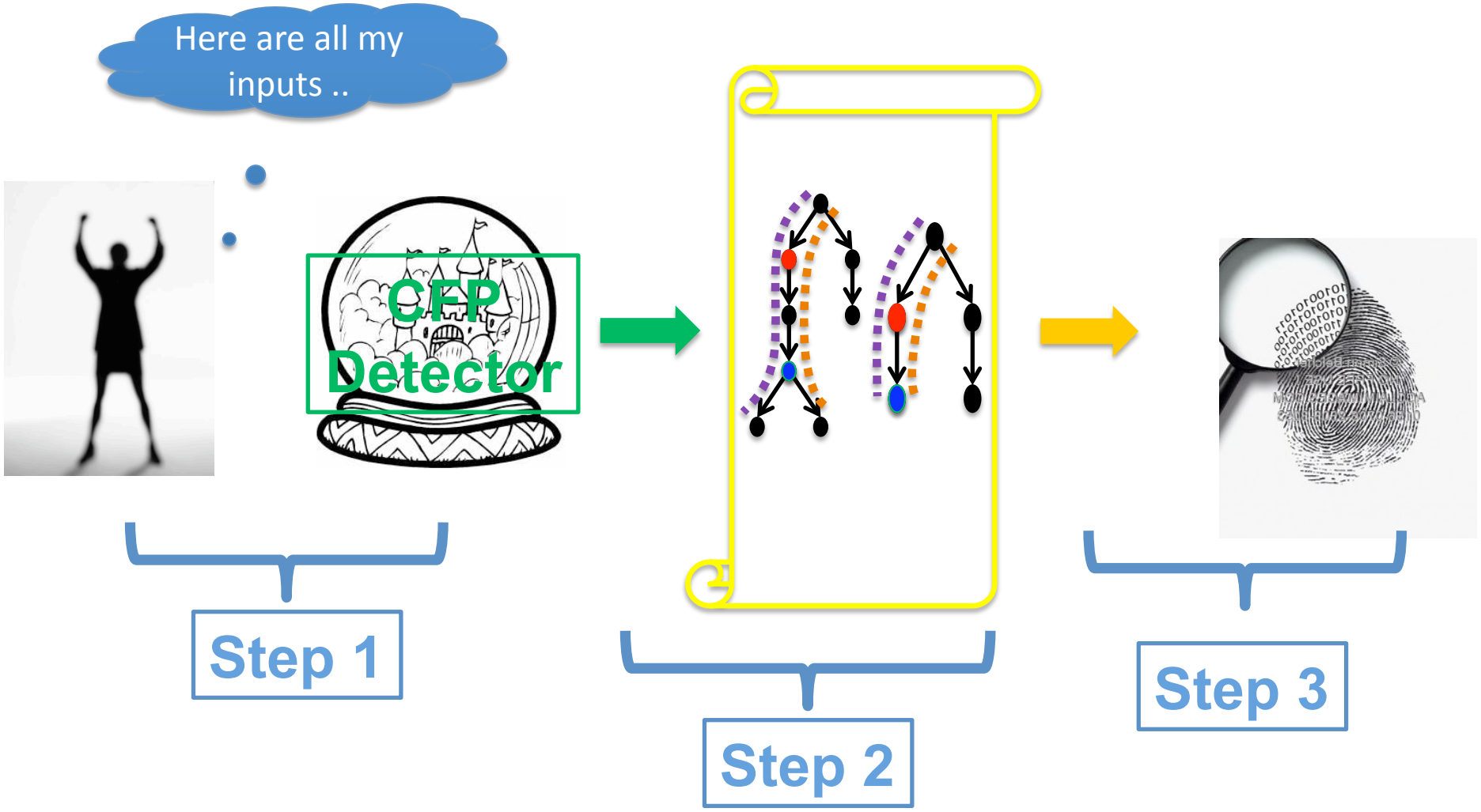
Application	# Inputs that expose the same bug
Aget	1
Click	1
FFT	1
Mozilla	1
Pbzip2	1



Outline

- Motivation
- Challenges
- Characteristics study
- Solution in details
- Experiment Evaluation
- Conclusion

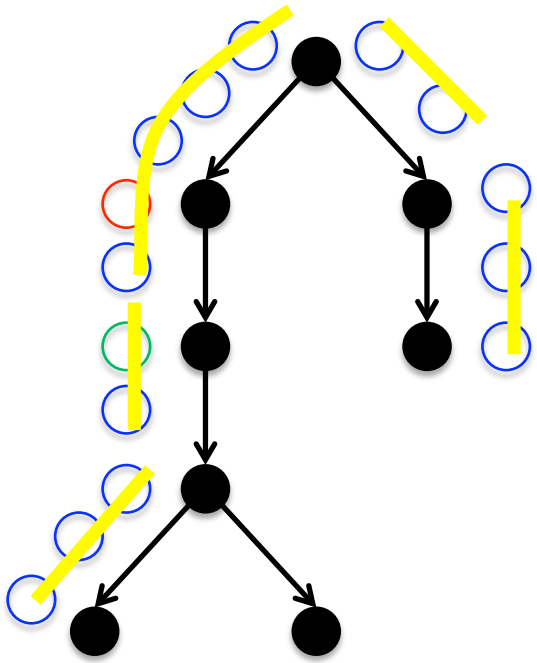
Overview



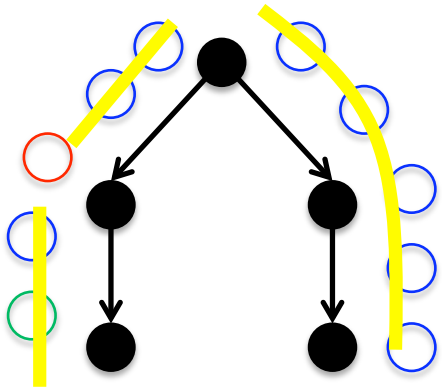


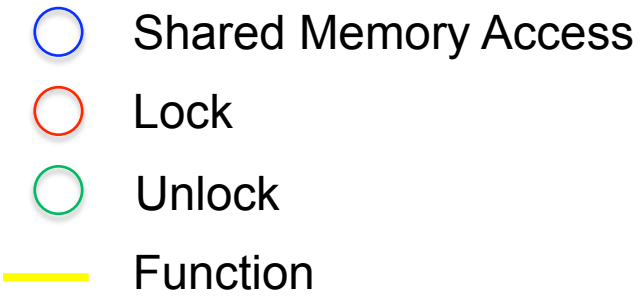
Concurrent Function Pair

Thread 1



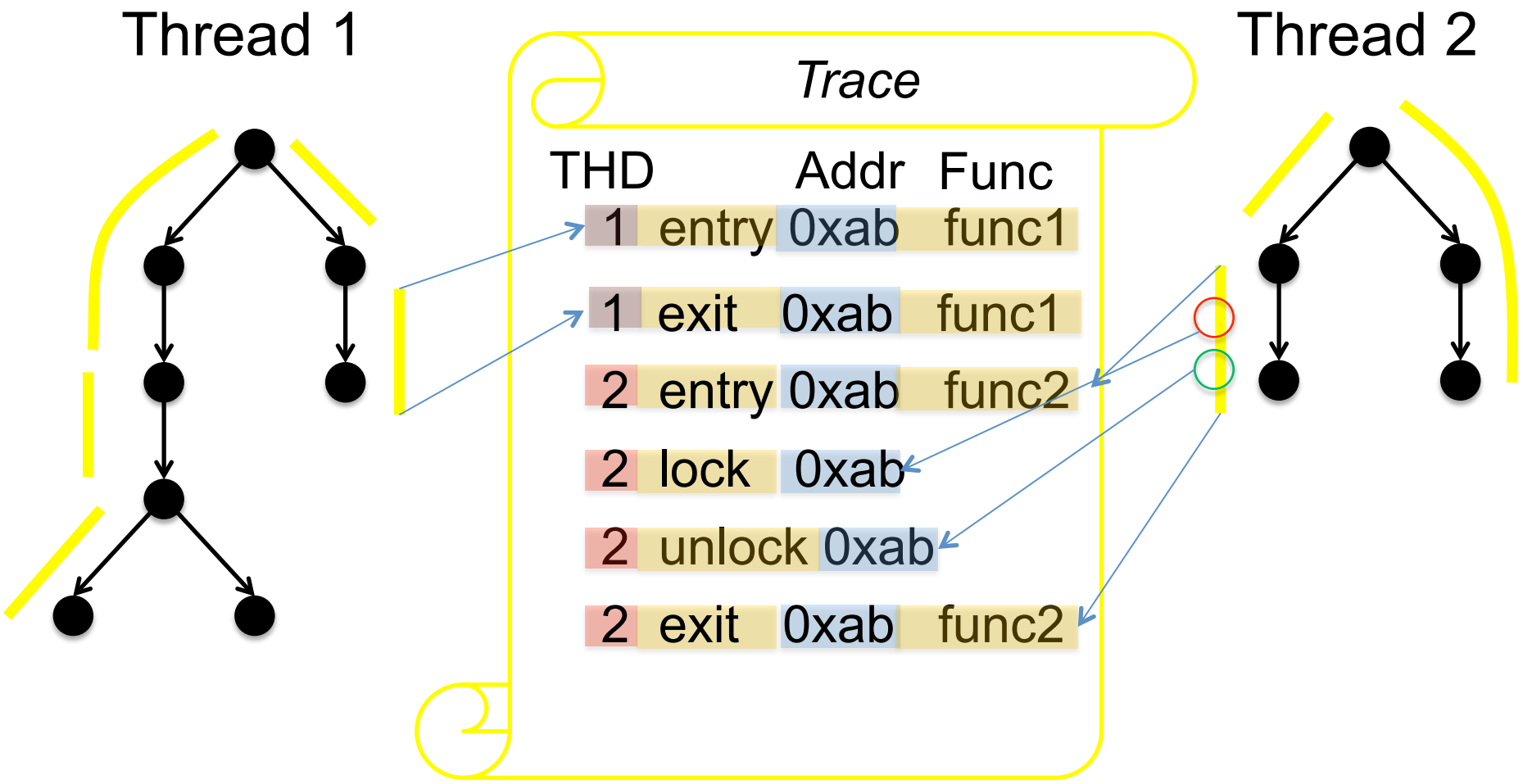
Thread 2





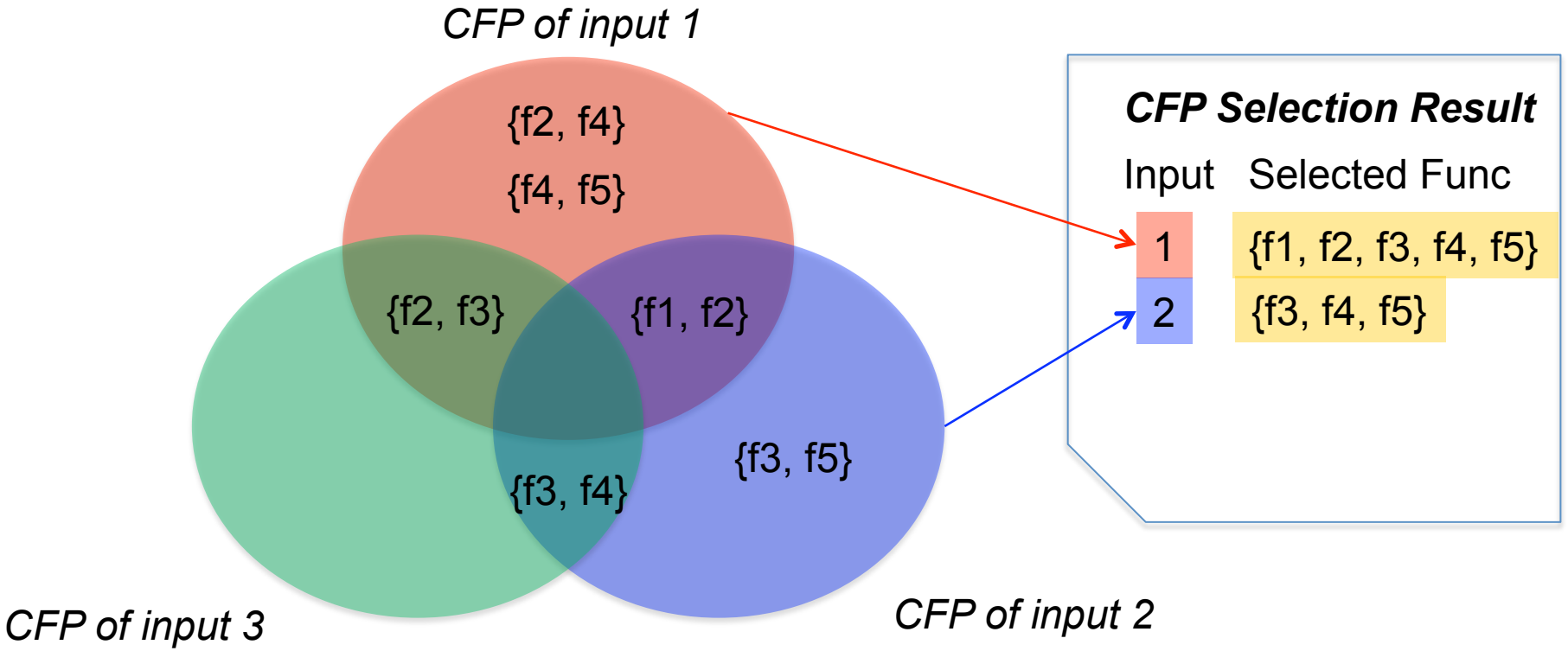
CFP, *C*oncurrent *F*unction *P*air, set of function pairs that can be potentially executed with each other

Step1 How to calculate CFP?



Step2 Design testing plan

Goal: select inputs and functions to avoid redundant CFP coverage

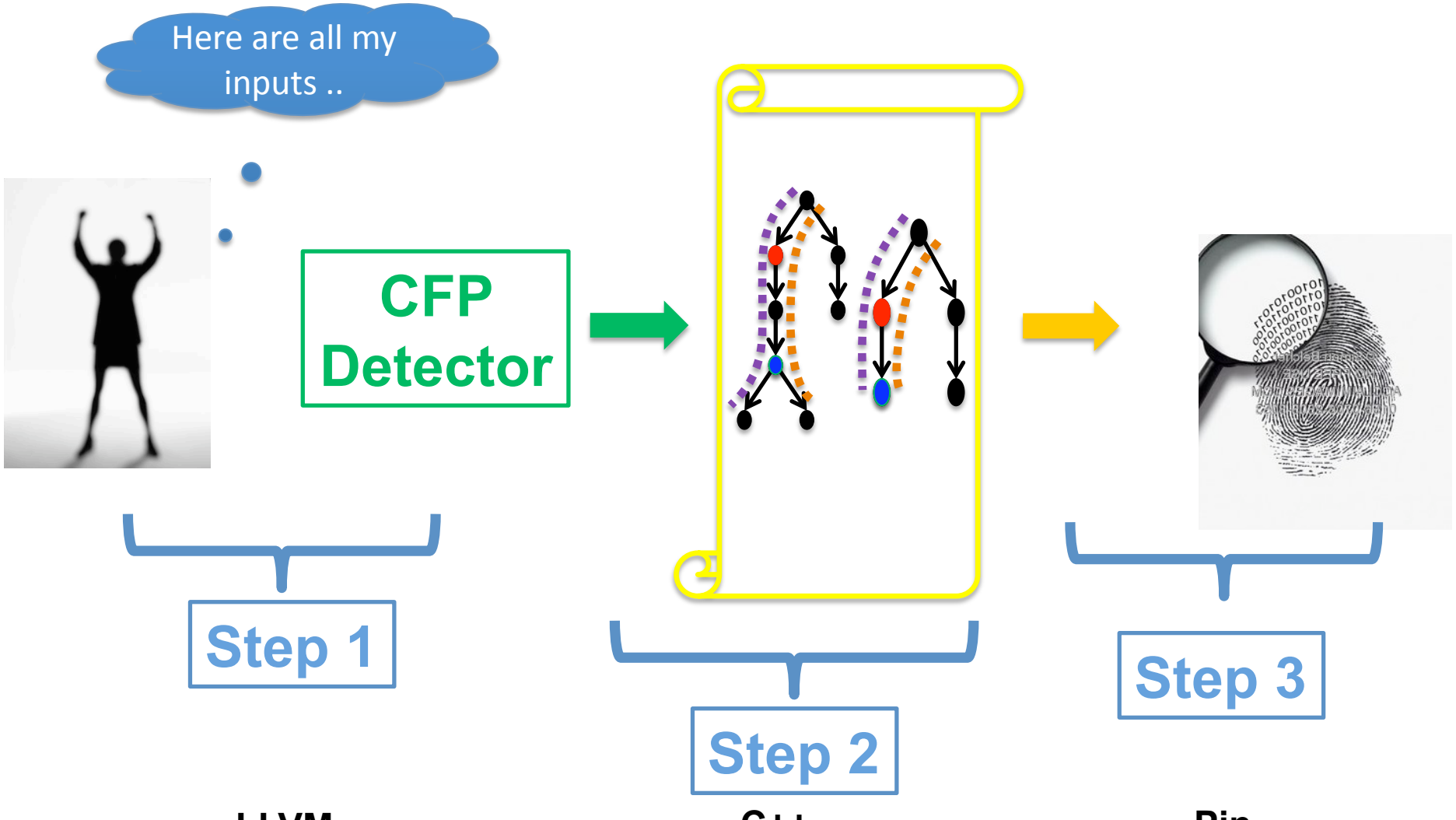


Step3 CFP-based Race Detection

- What do we instrument?
  - Only instrument *selected* functions for *selected* inputs.

- What data race detector do we use?
  - Hybrid of Happen-before & lock-set

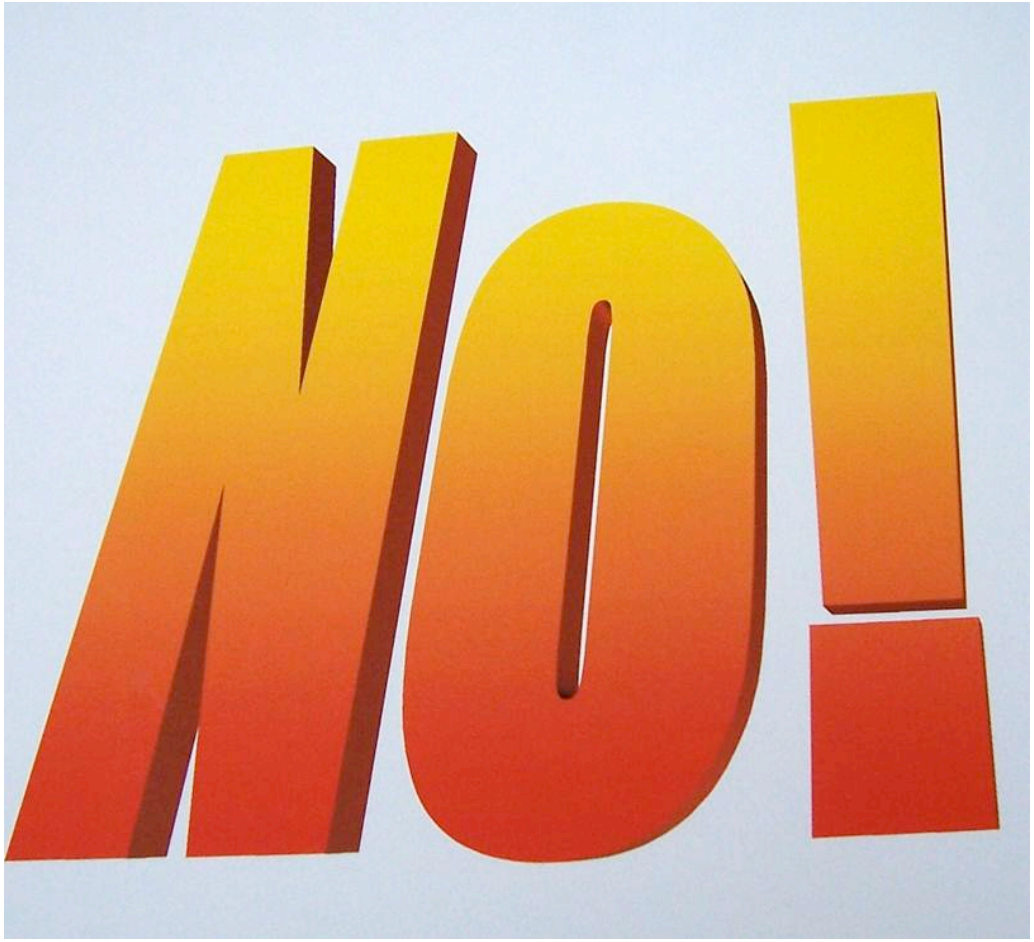
CFP-based Bug Detection Methodology



Input and function selection results

Application	Input Section	Log-size Reduction (%)
Aget	3 ouf of 8	94%
Click	6 out of 6	64%
FFT	1 out of 8	83%
Mozilla	7 out of 8	33%
PBZIP2	2 out of 8	44%

Miss any failure inducing data race?



How much redundancy we reduced?

Application	# Inputs that expose the same bug
Aget	6.67
Click	2.19
FFT	4.67
Mozilla	5.47
Pbzip2	4.17

VS

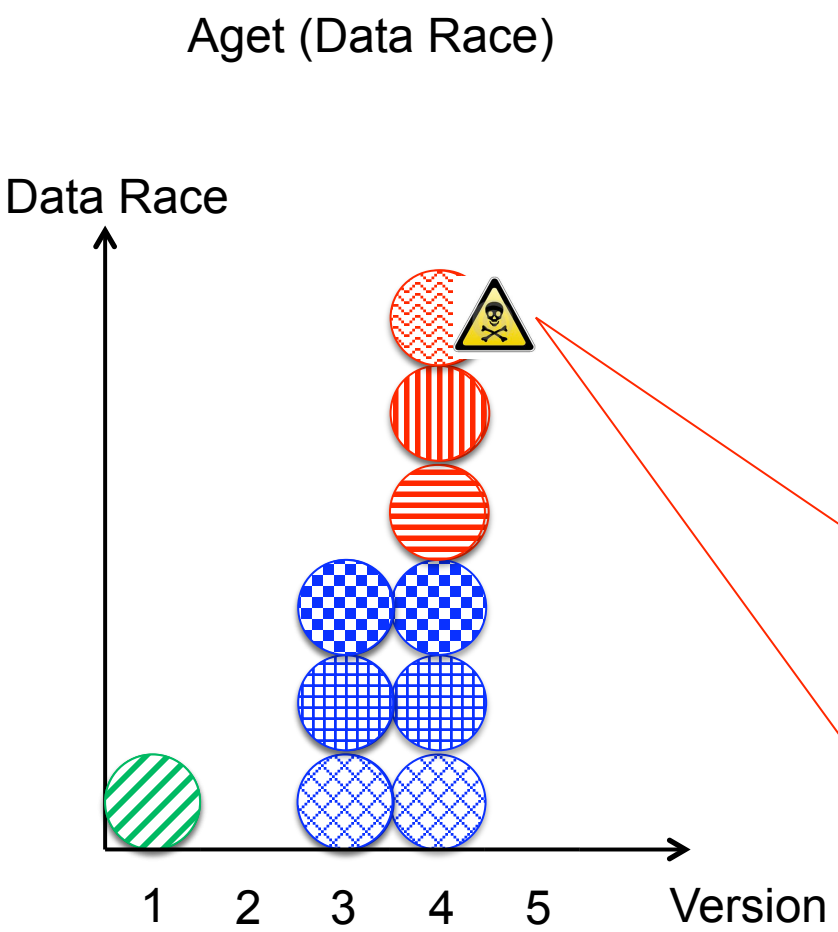
Application	# Inputs that expose the same bug
Aget	1
Click	1.63
FFT	1
Mozilla	2.73
Pbzip2	1.16

How much time we saved?

Application	Speedup (X)
Aget	0.76
Click	2.14
FFT	**7.13**
Mozilla	1.27
PBZIP2	1.72

*The time in our approach is calculated as the sum of three steps introduced before

Extension to Regression Test




```
typedef struct hist_data
{
    ...
    int bwritten;
    ...
} hist_data;
```

```
while(td->offset < foffset)
{
    ...
    dw = pwrite(…);
    td->offset += dw;
    ...
    bwritten += dw;
}
```

```
void save_log()
{
    ...
    h.bwritten += bwritten;
    ...
}
```

Conclusions

- Concurrency bug detection results are redundant across inputs and software versions
- Concurrent function pair (CFP) is good coverage metric for interleaving space
  - Lightweight
  - Reasonably accurate
- CFP can be used to guide concurrency-bug detection across inputs and software versions

Future Work

- Applying CFP to help detect other types of bugs
- Preprocessing to get more accurate CFP
  - Remove functions that do not access shared variables
- Trying different heuristics for different testing budgets
- Exploring other coverage metrics

Thanks!

Wei Zhang

http://pages.cs.wisc.edu/~wzh