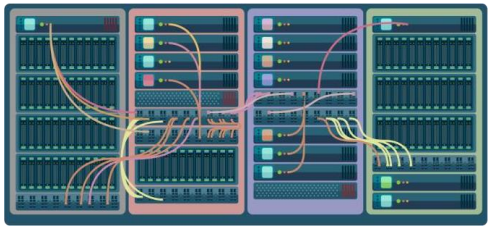
REPT: Reverse Debugging of Failures in Deployed Software

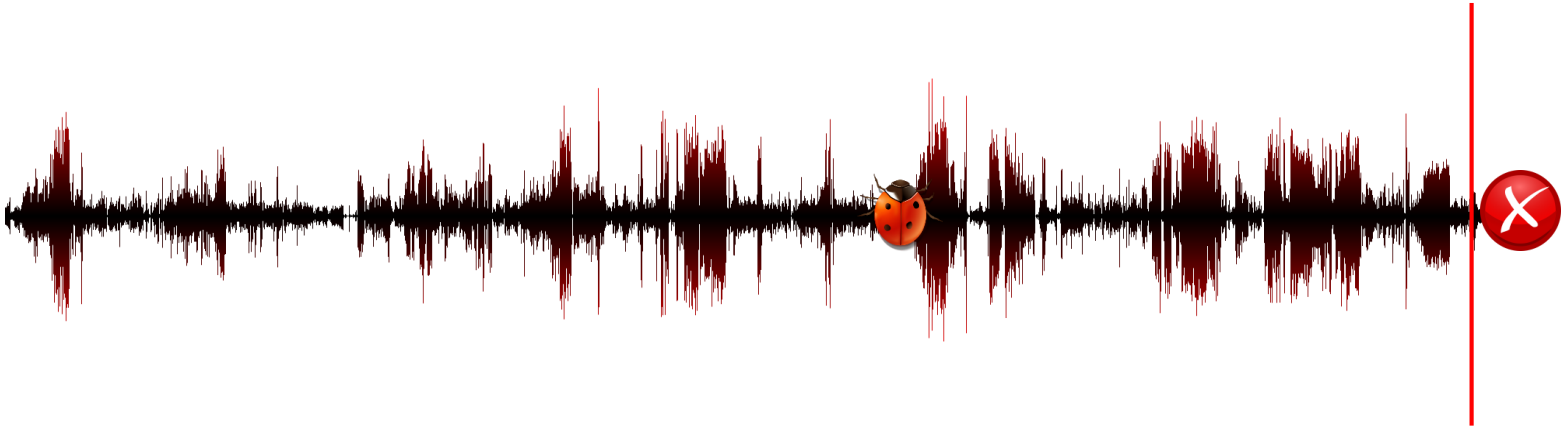
Harshitha Nagapudi - hnagapud@ttu.edu
Texas Tech University

OUTLINE

- Existing Problems
- REPT as a Solution
- How it Works
- Evaluation
- Limitations
- Future Work
- Conclusion

What happened before the crash?






A
X60

REPT: Reverse Execution with Processor Trace

REPT: Reverse Execution with Processor Trace

- Online hardware tracing (e.g., Intel Processor Trace)
  - Log the control flow with timestamps
  - Low runtime overhead (1 – 5%)
  - *No data!*

- Offline binary analysis
  - Recovers data flow from the control flow

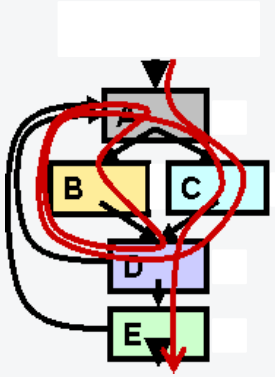
REPT Data Recovery

- Single-threaded execution reconstruction
- Multi-threaded execution reconstruction

Core Dump

```
10101011010101010101010110011010100100111101101101
11010101101101010110101010101010101011001101010010
11110110110110101011011010101101001010101010101010
01101010010011110110110101010110101010111010101010
10101010110011010100100111101101101101010110110101
10110100101010101010110011010100100111101101101010
10101101101010110101010101010101011001101010010011
10110110110101011011010101101010101010101011001101
10100100111101101101101011011010101101010101010101
01010101010110011010100100111101101101101010110101
01010110101010101010101100110101001001111011011101
01011010101010101010110011010100100111101101101010
10101101101010110101010101010101011001101010010011
10110110110101011011010101101010101010101011001101
11001101010010011110110110110101011010101010101010
01010101010110011010100100111101101101010110101011
01010101011001101010010011110110110101011011011101
01011010101010101010110011010100100111101101101010
10101101101010110101010101010101011001101010010011
01001111011011011010101101101010110101010101010101
11001101010010011110110110110101011010101010101010
01101010101010101011001101010010011110110110110101
```

Instruction Sequence



Execution History

?

How to recover overwritten states

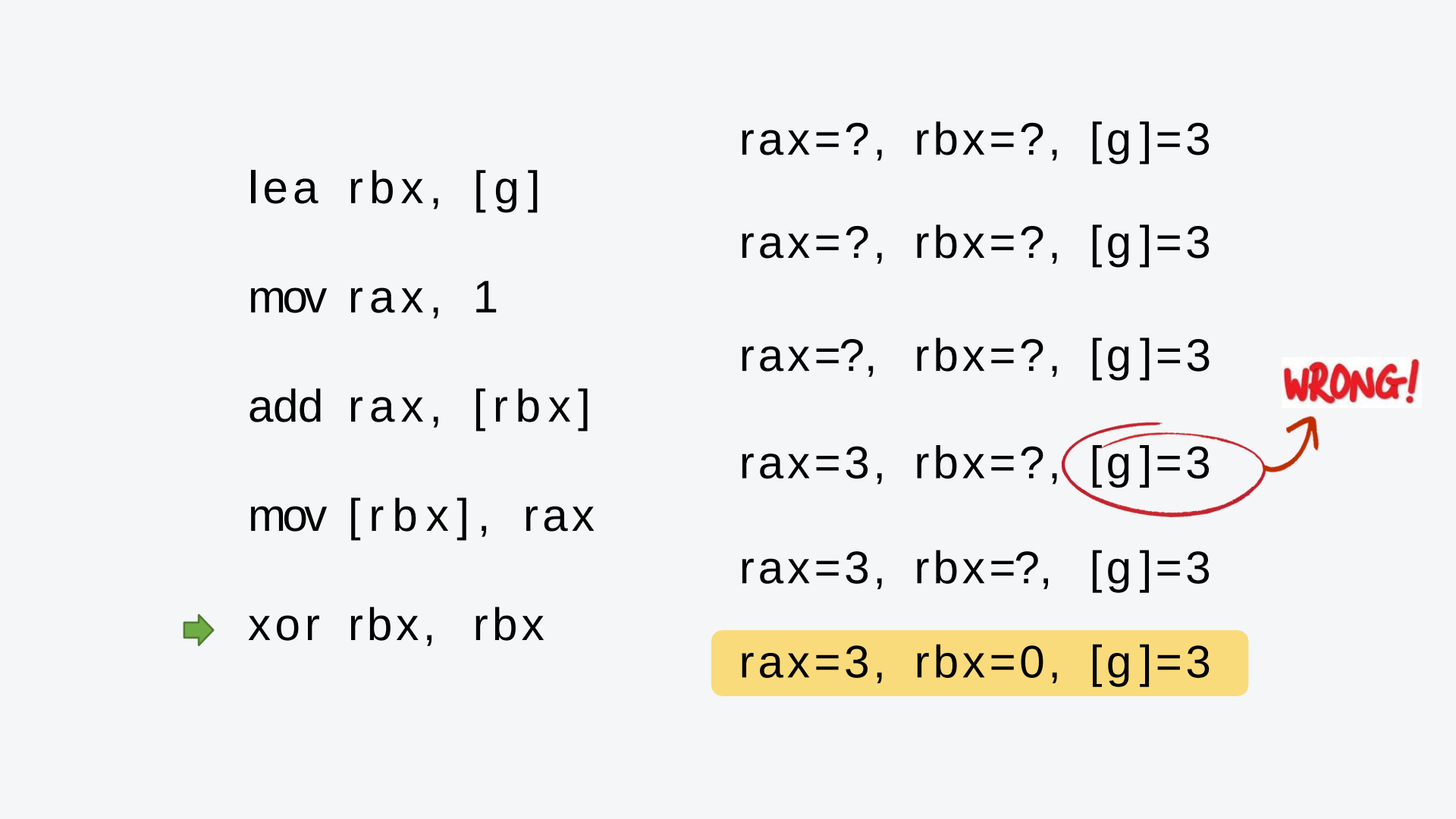
```
lea rbx, [g]

mov rax, 1

add rax, [rbx]

mov [rbx], rax

xor rbx, rbx
```


lea rbx, [g]
mov rax, 1
add rax, [rbx]
mov [rbx], rax
xor rbx, rbx
rax=?, rbx=?, [g]=3
rax=?, rbx=?, [g]=3
rax=?, rbx=?, [g]=3
rax=3, rbx=?, [g]=3
rax=3, rbx=?, [g]=3
rax=3, rbx=0, [g]=3
WRONG!

```
lea rbx, [g]
mov rax, 1
add rax, [rbx]
mov [rbx], rax
xor rbx, rbx
```

rax=?, rbx=?, [g]=3

rax=?, rbx=g, [g]=3

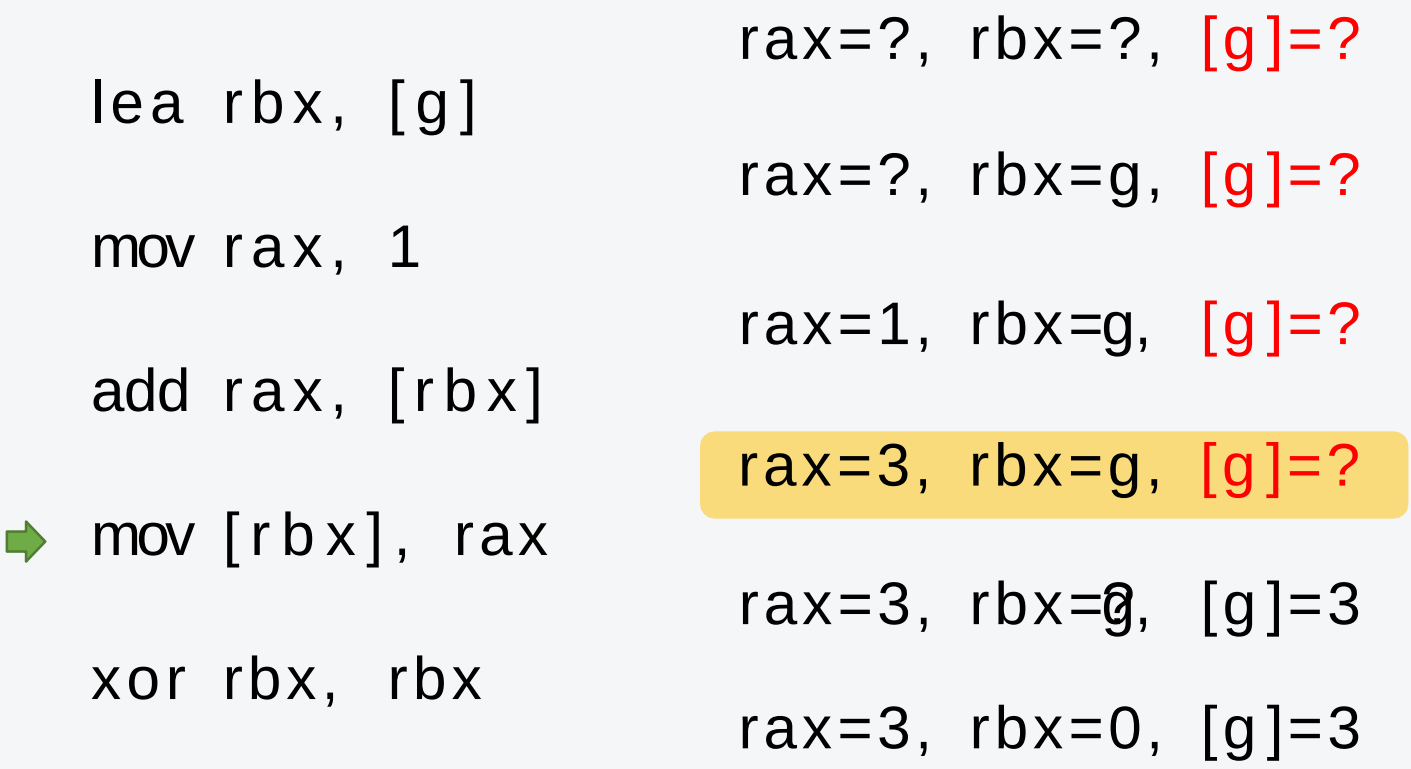
rax=1, rbx=g, [g]=3

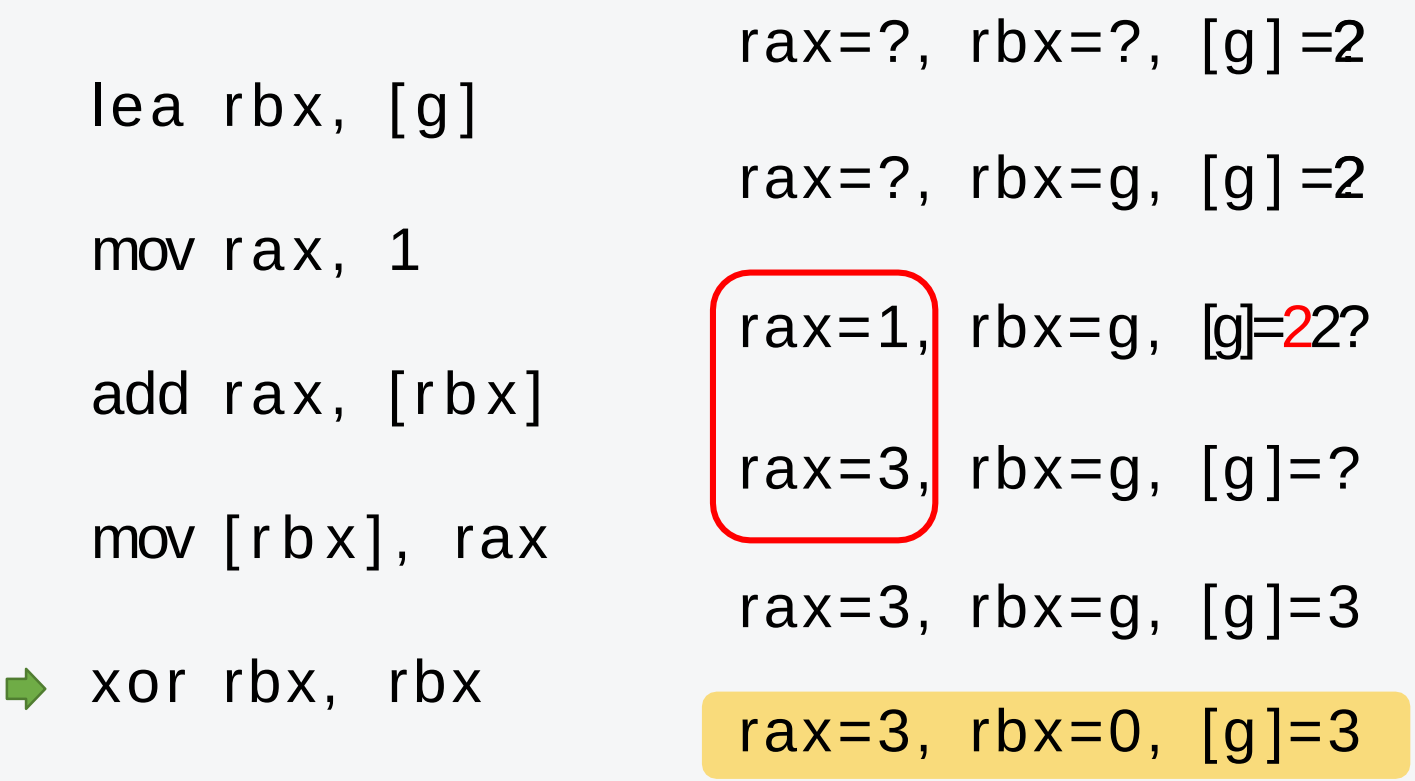
4?

rax=3, rbx=g, [g]=3

rax=3, rbx=?, [g]=3

rax=3, rbx=0, [g]=3


lea rbx, [g]
mov rax, 1
add rax, [rbx]
mov [rbx], rax
xor rbx, rbx
rax=?, rbx=?, [g]=?
rax=?, rbx=g, [g]=?
rax=1, rbx=g, [g]=?
rax=3, rbx=g, [g]=?
rax=3, rbx=g, [g]=3
rax=3, rbx=0, [g]=3


lea rbx, [g]
mov rax, 1
add rax, [rbx]
mov [rbx], rax
xor rbx, rbx
rax=?, rbx=?, [g]=2
rax=?, rbx=g, [g]=2
rax=1, rbx=g, [g]=2?
rax=3, rbx=g, [g]=?
rax=3, rbx=g, [g]=3
rax=3, rbx=0, [g]=3

```
lea rbx, [g]
mov rax, 1
add rax, [rbx]
mov [rbx], rax
xor rbx, rbx
```

rax=?, rbx=?, [g]=2

rax=?, rbx=g, [g]=2

rax=1, rbx=g, [g]=2

rax=3, rbx=g, [g]=2

rax=3, rbx=g, [g]=3

rax=3, rbx=0, [g]=3

```
lea rbx, [g]
mov rax, 1
add rax, [rbx]
mov [rbx], rax
xor rbx, rbx
```

```
rax=?, rbx=?, [g]=2
rax=?, rbx=g, [g]=2
rax=1, rbx=g, [g]=2
rax=3, rbx=g, [g]=2
rax=3, rbx=g, [g]=3
rax=3, rbx=0, [g]=3
```

Key Techniques

- Forward Execution
  - Recovers states before irreversible instructions
- Error Correction
  - Handles errors introduced by “missing” memory writes

REPT Data Recovery

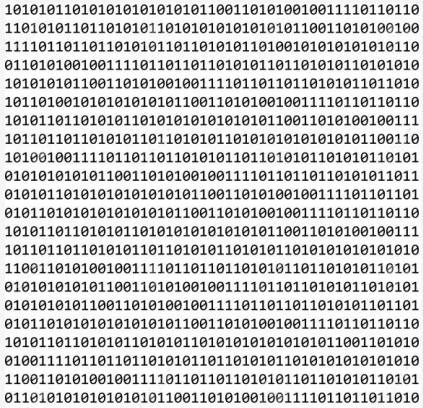
- Single-threaded execution reconstruction
- Multi-threaded execution reconstruction

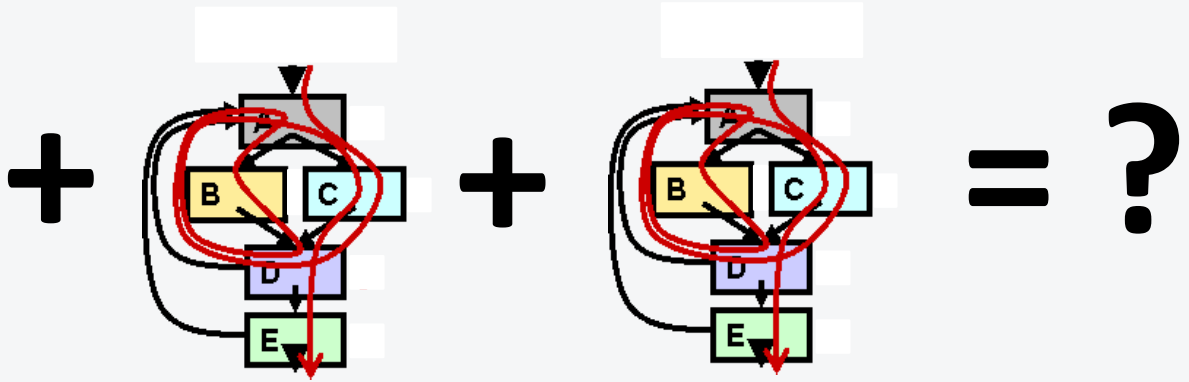
Core Dump

Instruction Sequence #1

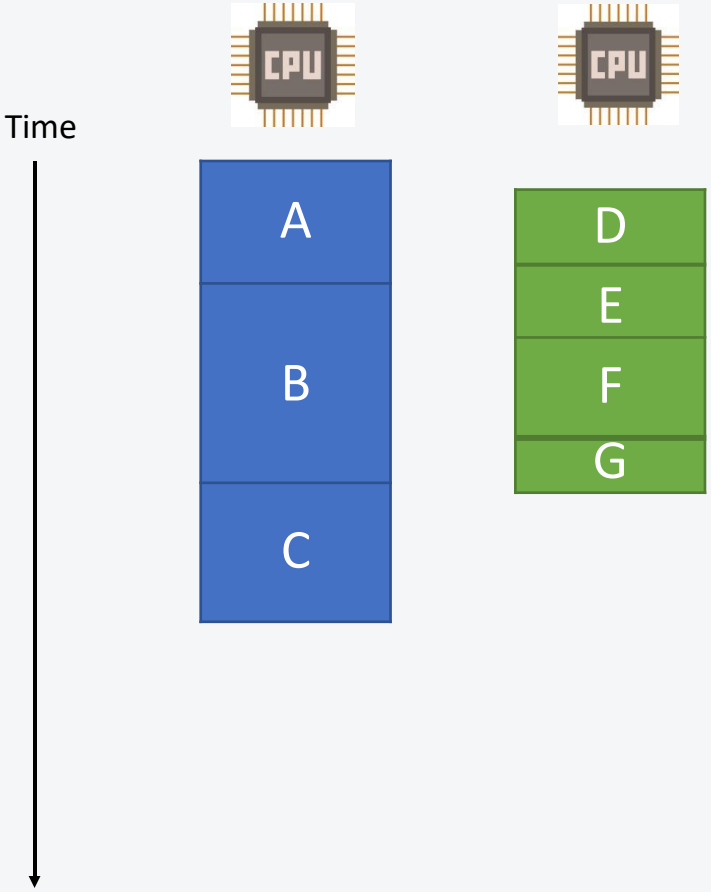
Instruction Sequence #2

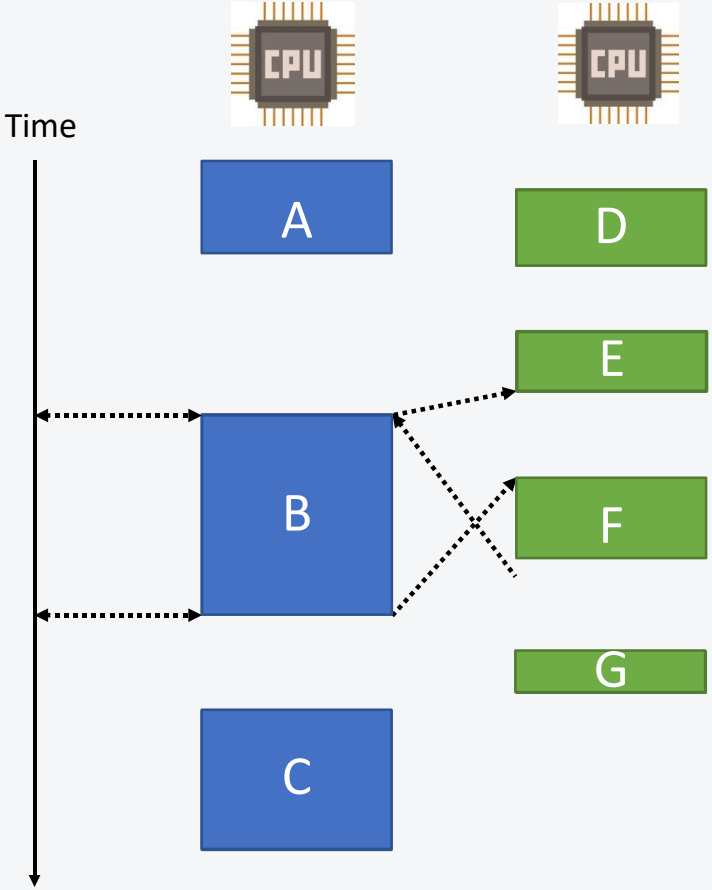
Execution History

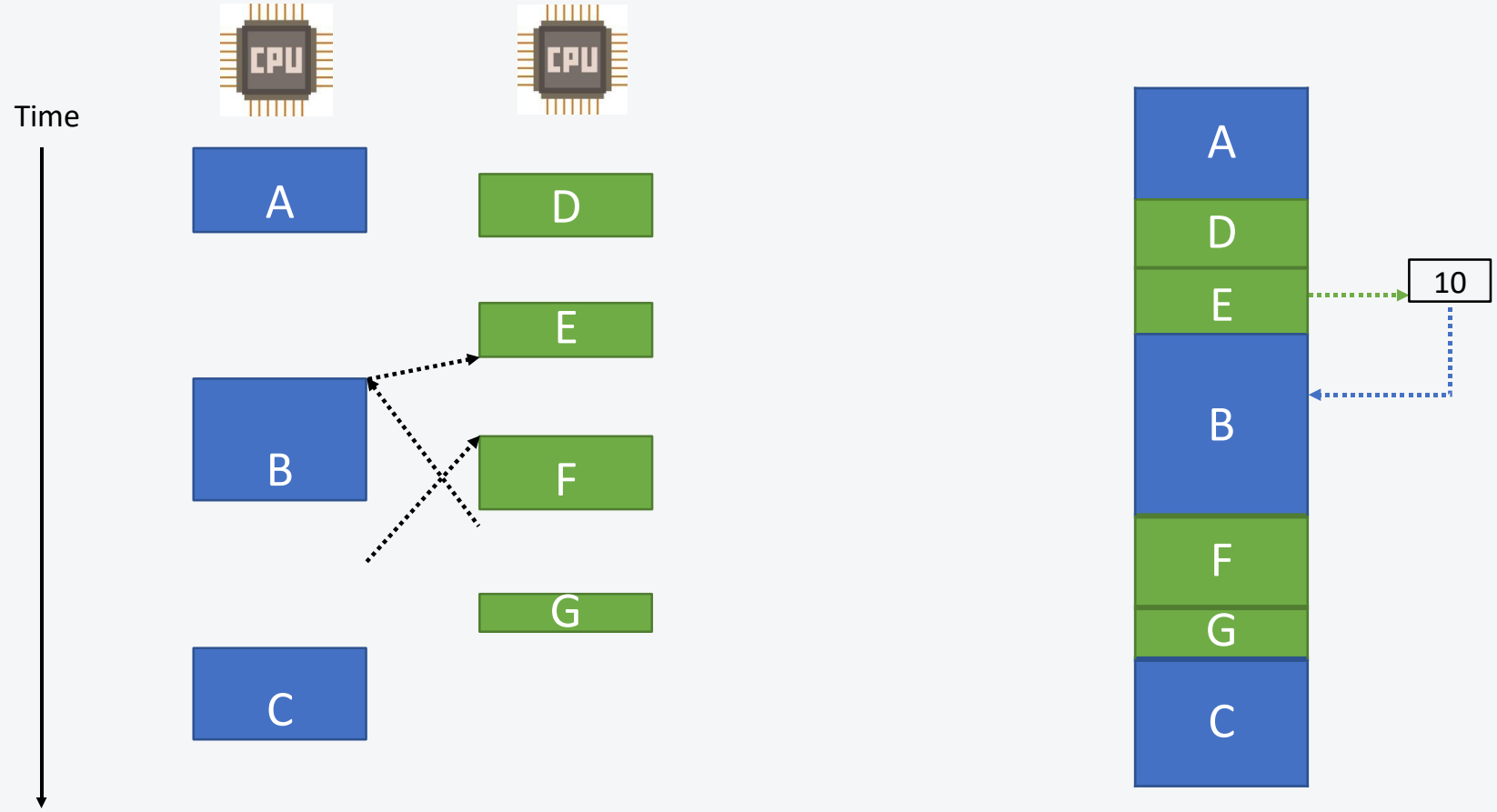


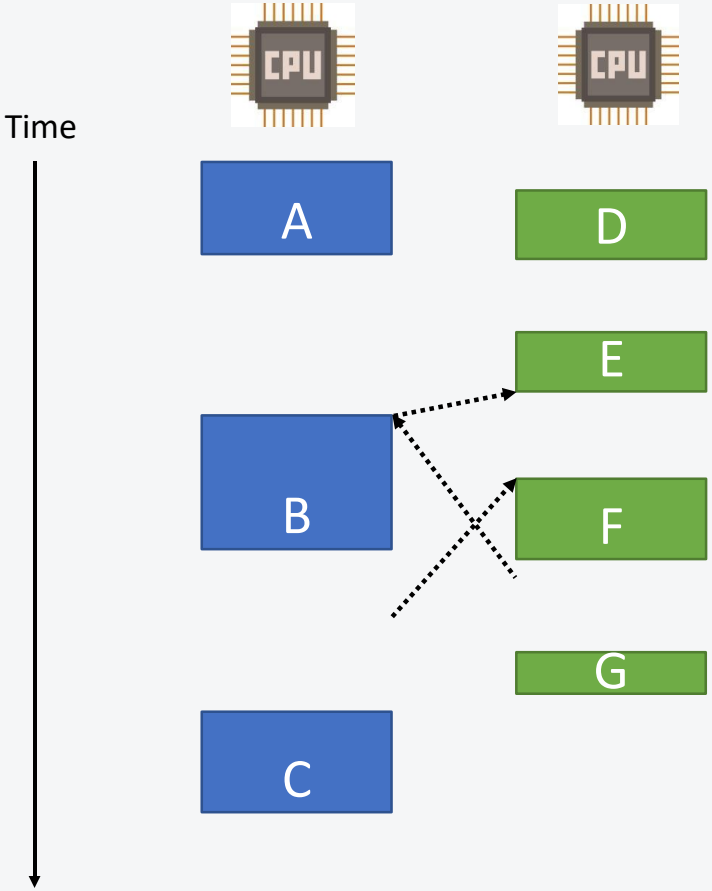


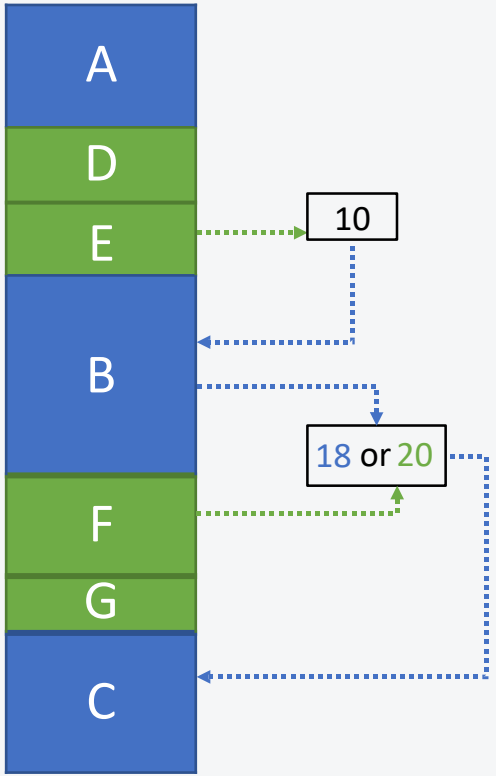
How to determine the thread interleavings?

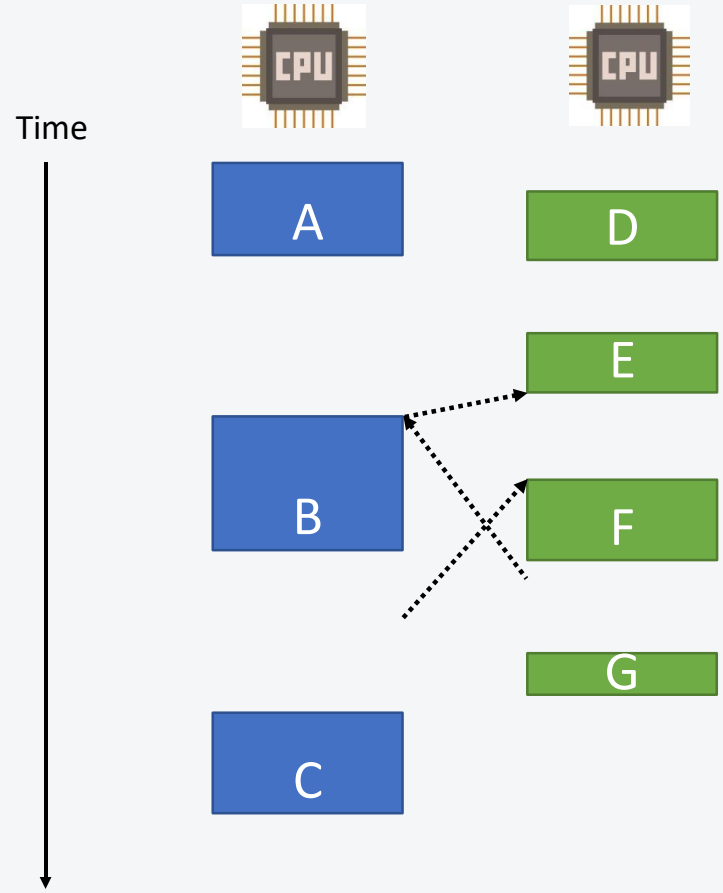

Time
CPU
CPU
A
B
C
D
E
F
G

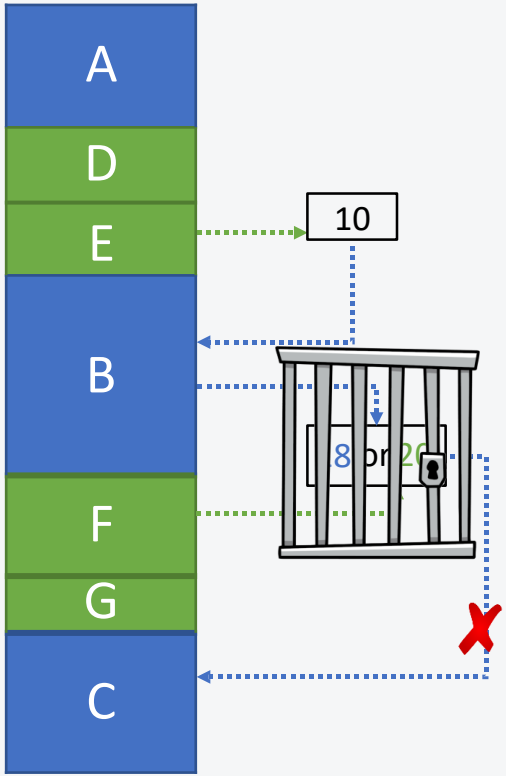

CPU
CPU
Time
A
D
E
B
F
G
C


Time
CPU
CPU
A
D
E
B
F
G
C
A
D
E
10
B
F
G
C


Time
CPU
CPU
A
D
E
B
F
G
C


A
D
E
10
B
18 or 20
F
G
C


Time
CPU
CPU
A
D
E
B
F
G
C


A
D
E
B
F
G
C
10

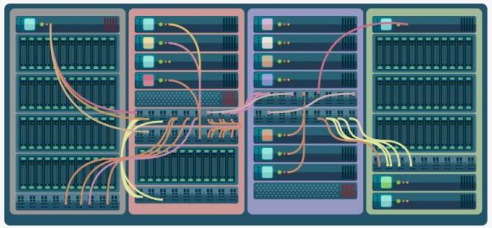
Key Techniques

- Hardware Timestamps
  - Constructs a partial order
- Concurrent memory write detection
  - Constrains their usage to avoid propagating a wrong value

With REPT

I want history information!

Hey, client, turn on tracing next time.



Evaluation

16 bugs

1-5% overhead

14 bugs

92% accuracy

Limitations

- The control flow trace may not be long enough to capture the defect e.g., the free call is not in the trace for a use-after-free bug.

- Data values that are necessary for debugging the failure are not recovered e.g., the heap address passed to the free call is not recovered for a use-after-free bug.

Future Work

- REPT currently does not capture any data during a program's execution. To fundamentally solve these two limitations, we will need to log more data than just the memory dump.

- While REPT's core analysis is on machine instructions and thus independent of the privilege mode, we need to properly handle kernel-specific artifacts such as interrupts to support reverse debugging of kernel-mode executions.

- Can study how program tracing can be combined with event logging to help developers debug bugs in distributed systems.

Conclusion

- Debugging production failures is important but hard
- REPT is a practical reverse debugging solution for production failures
  - Online hardware tracing to log the control flow with timestamps
  - Offline binary analysis to recover the data flow with high accuracy
- REPT has been deployed on Microsoft Windows

REFERENCES

- Cui, Weidong, et al. "{REPT}: Reverse debugging of failures in deployed software." *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 2018.
- Cui, Weidong, et al. "Retracer: Triaging crashes by reverse execution from partial memory dumps." *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 2016.


THANKYOU