REPT: Reverse Debugging of Failures in Deployed Software

Weidong Cui[1], **Xinyang Ge**[1], Baris Kasikci[2], Ben Niu[1], Upamanyu Sharma[2], Ruoyu Wang[3], and Insu Yun[4]

Microsoft Research[1]

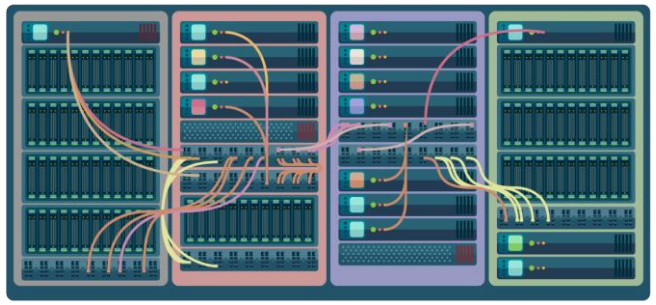
University of Michigan[2]

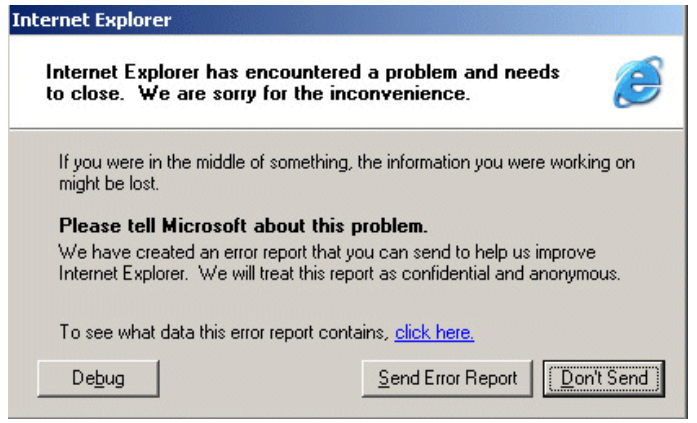
Arizona State University[3]

Georgia Institute of Technology[4]

OSDI 2018, Carlsbad, CA

What happened before the crash?






Rime Sce
REE P
do not
o not cross
rime scene


A
HIGH QUALITY SOUND
X60

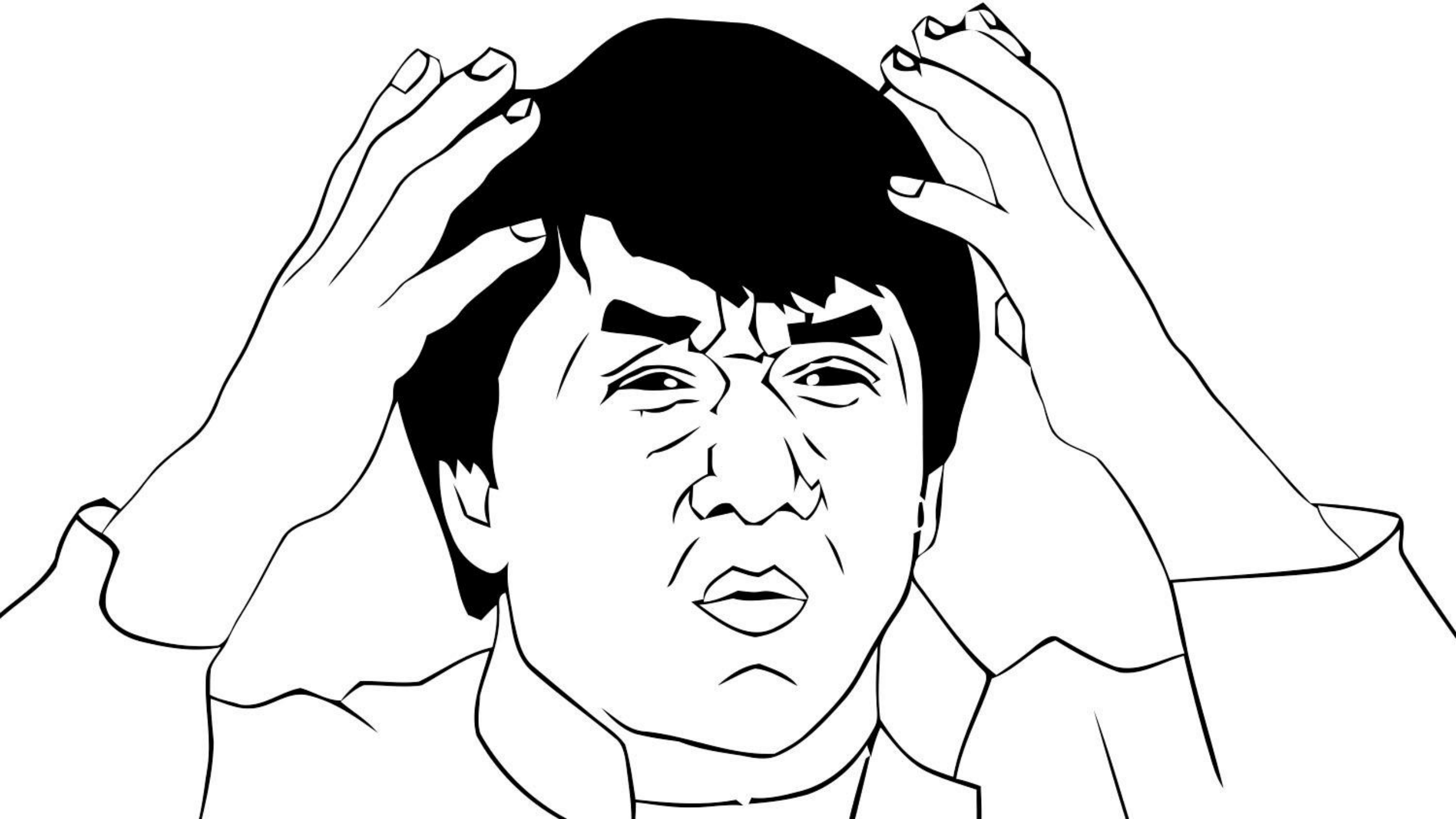
ONE TEAM
ONE MISSION

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

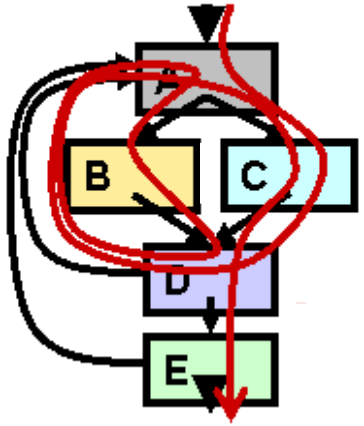
- Single-threaded execution reconstruction

- Multi-threaded execution reconstruction

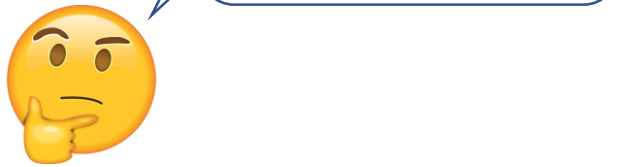
Core Dump

```
10101011010101010101011001101010010011110110110
11010101101101010110101010101010101100110101001 00
11110110110110101011011010101101001010101010110
01101010010011110110110110101011011010101101010
10101010110011010100100111101101101101011011010
10110100101010101010110011010100100111101101110
10101101101010110101010101010101100110101001001 11
10110110110101011011010101101010101010101100110
10100100111101101101101011011010101101010110101
01010101010110011010100100111101101101101011011
01010110101010101010101100110101001001111011011 01
01011010101010101010110011010100100111101101101
10101101101010110101010101010101100110101001001 11
10110110110101011011010101101010101010101100110
01101010010011111011011011010101101101010110101
01010101010110011010100100111101101101010110101
01010101011001101010010011110110110101010110101
01011010101010101010110011010100100111101101101
10101101101010110101011010101010101010011001101010
01001111011011011010101101101011010101010101010
11001101010010011110110110110101011011010110101
01101010101010101011001101010010011110110110110 10
```

Instruction Sequence



Execution History



```
lea rbx, [g]

mov rax, 1

add rax, [rbx]

mov [rbx], rax

xor rbx, rbx
```

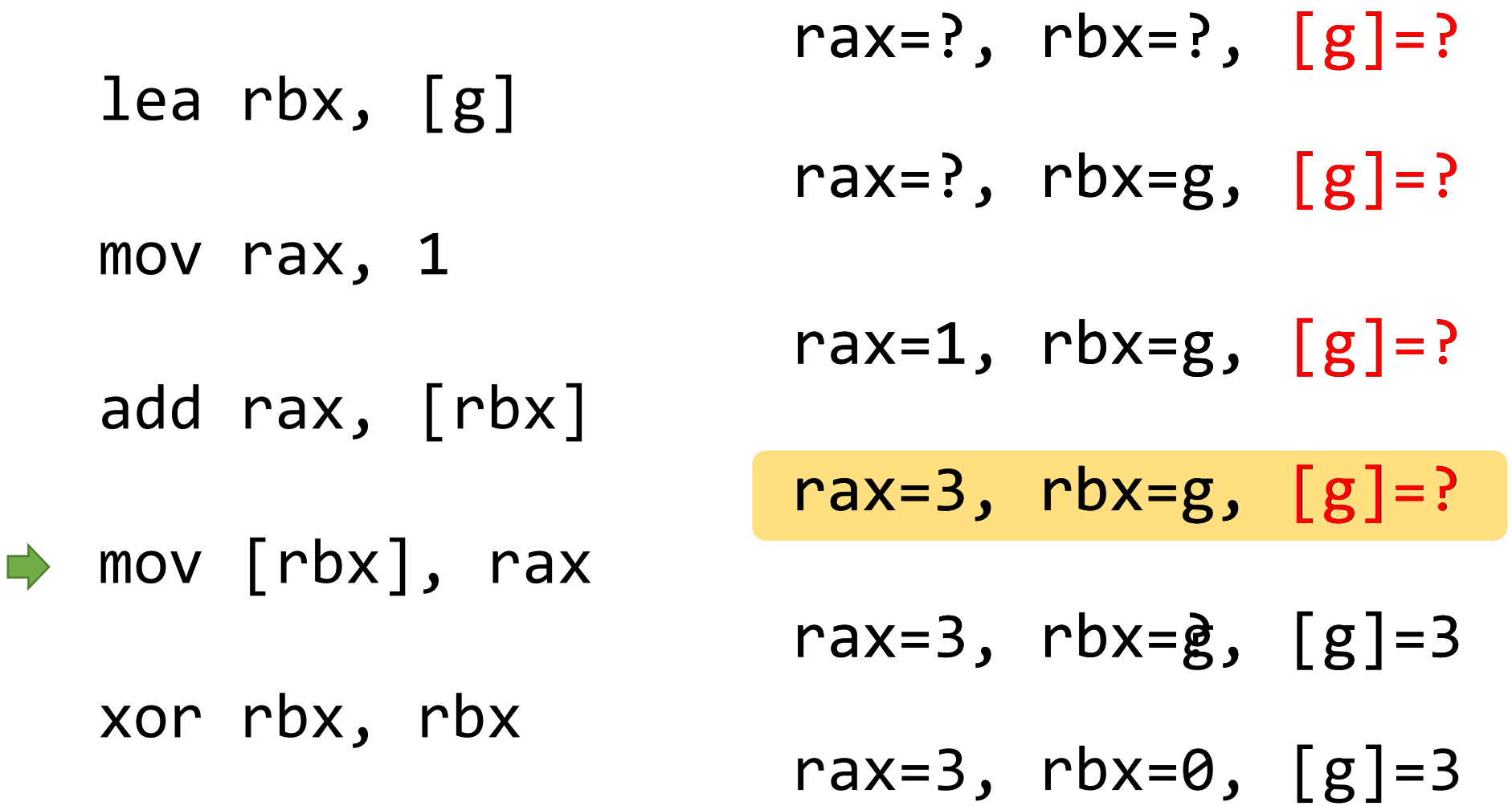
```
lea rbx, [g]
mov rax, 1
add rax, [rbx]
mov [rbx], rax
xor rbx, rbx
```

```
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
```

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

4?

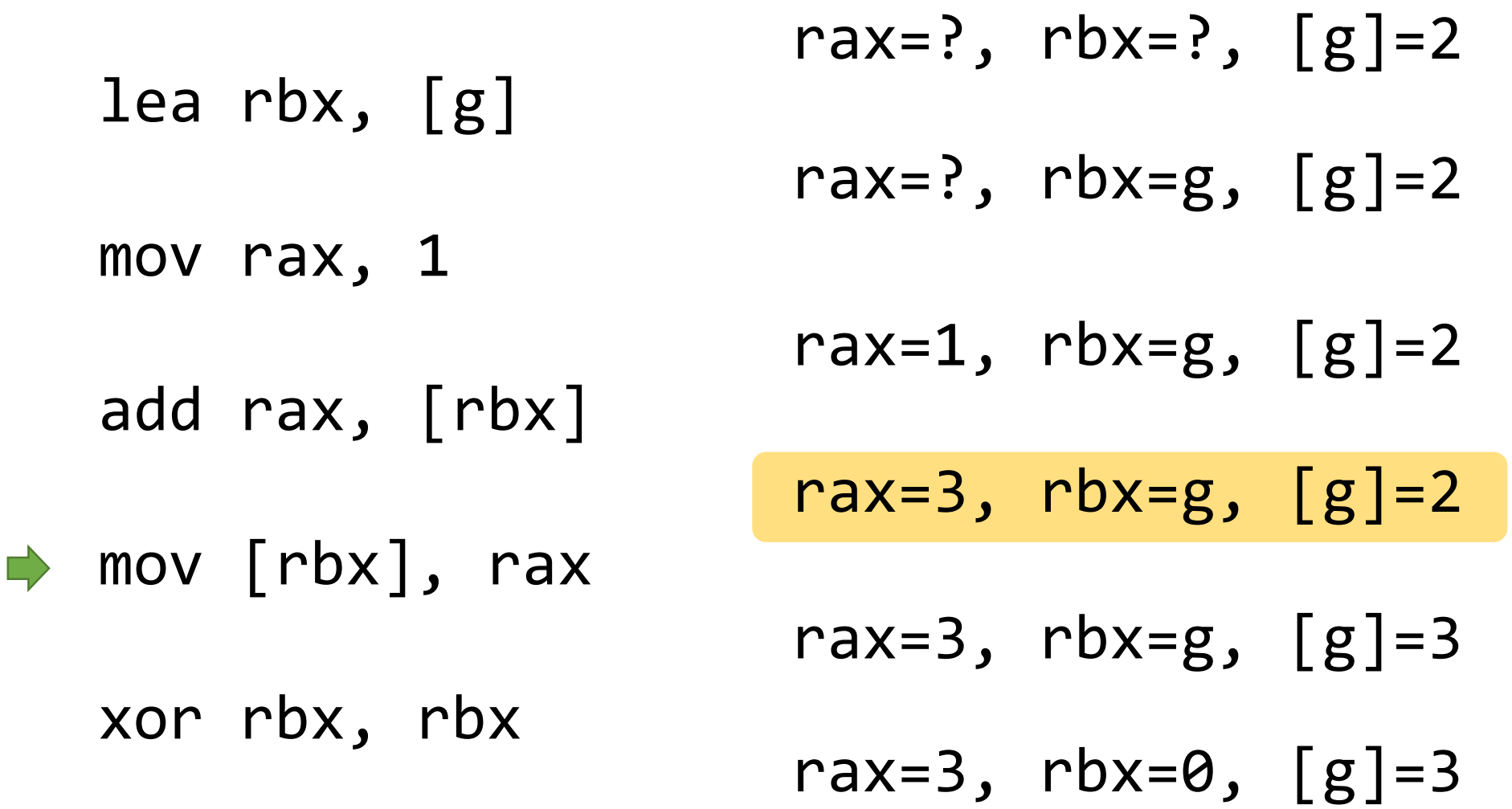

lea rbx, [g]
mov rax, 1
add rax, [rbx]
mov [rbx], rax
xor rbx, rbx
rax=?, rbx=?, [g]=?
rax=?, rbx=g, [g]=?
rax=1, rbx=g, [g]=?
rax=3, rbx=g, [g]=?
rax=3, rbx=g, [g]=3
rax=3, rbx=0, [g]=3

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

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

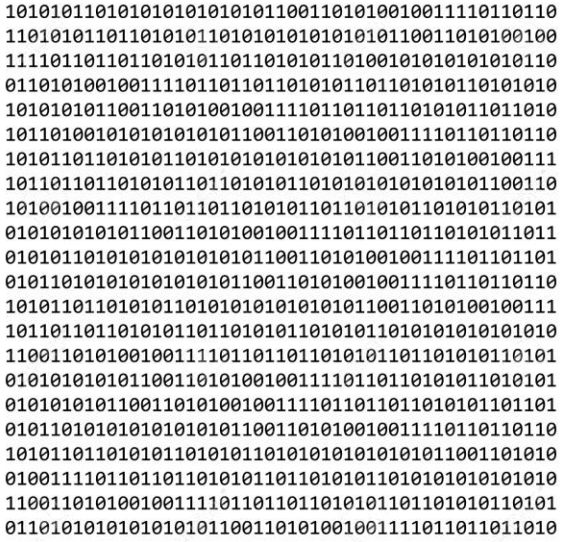
- Multi-threaded execution reconstruction

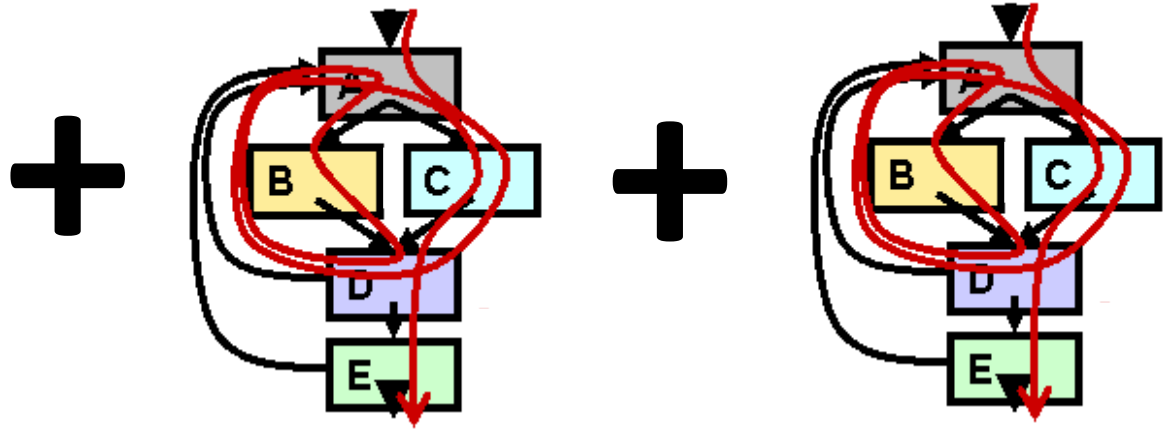
Core Dump

Instruction Sequence #1

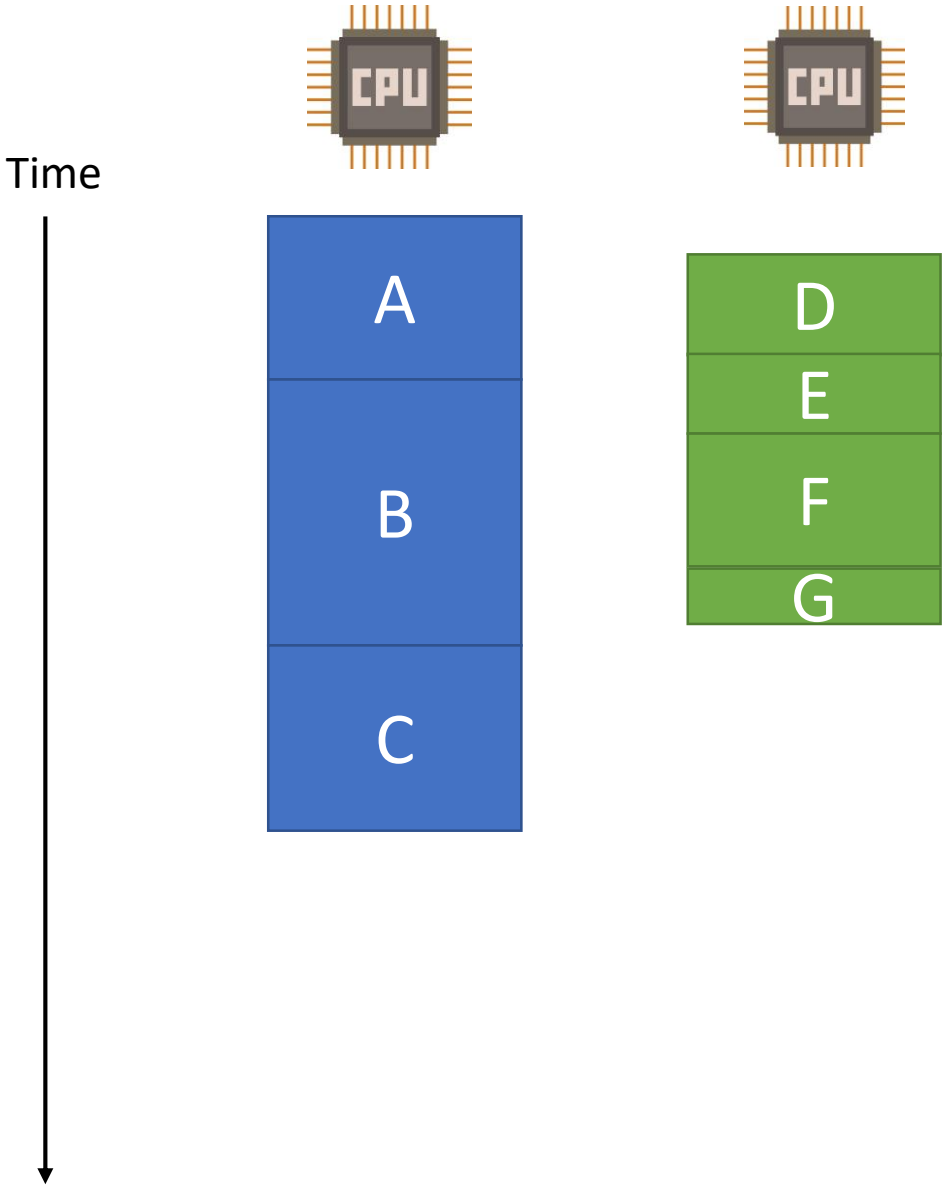
Instruction Sequence #2

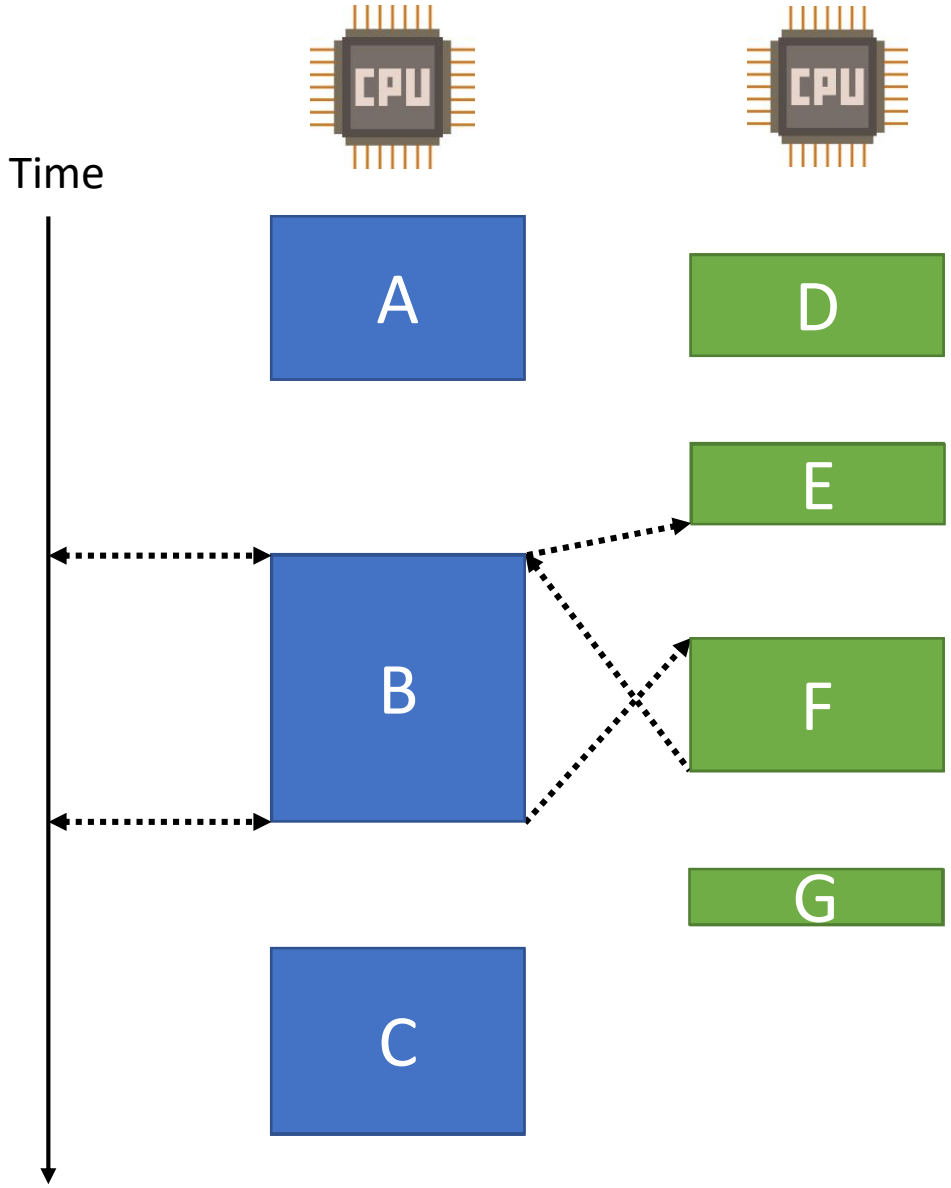
Execution History

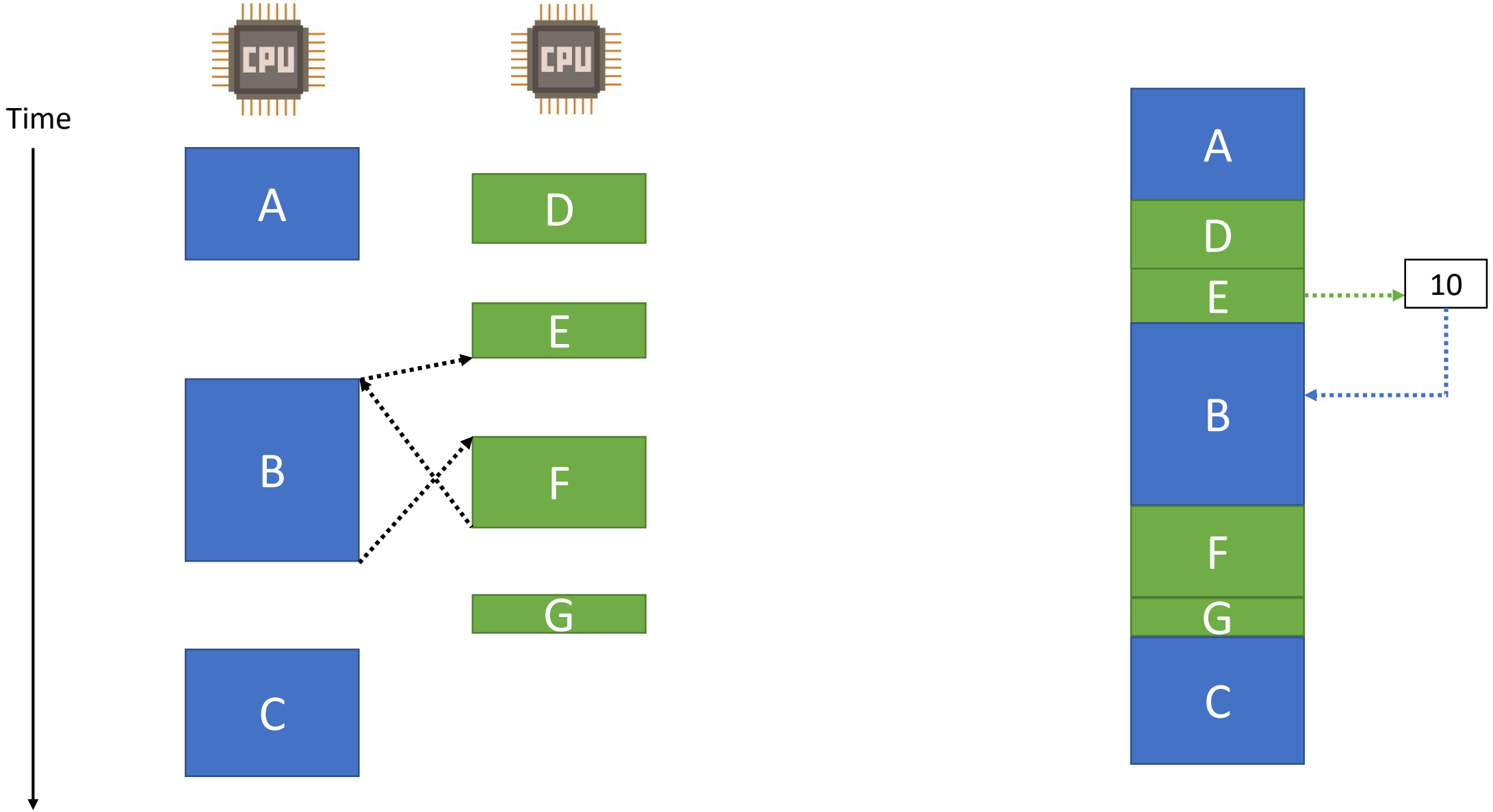


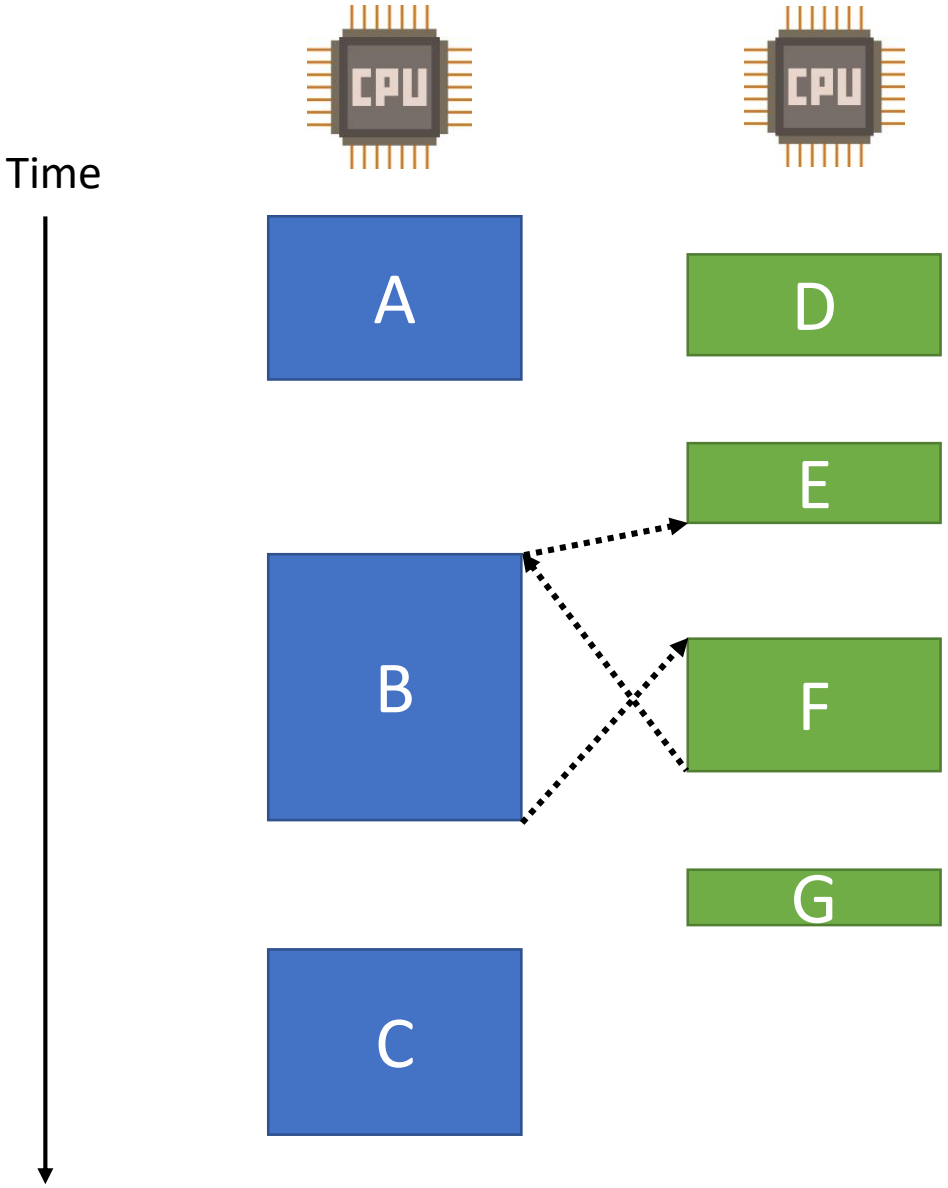


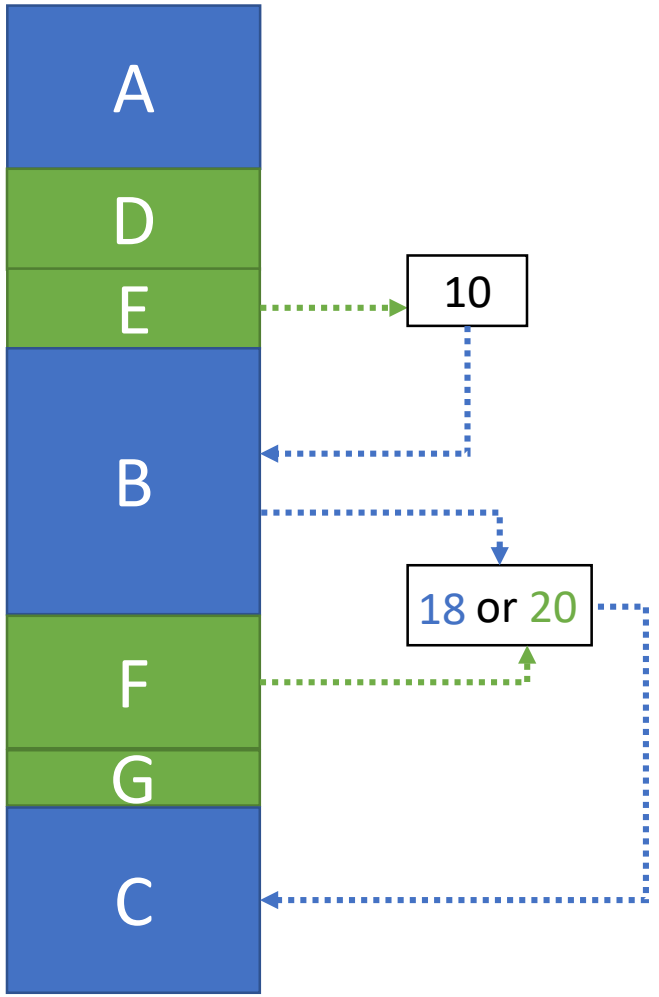
How to determine the thread interleavings?

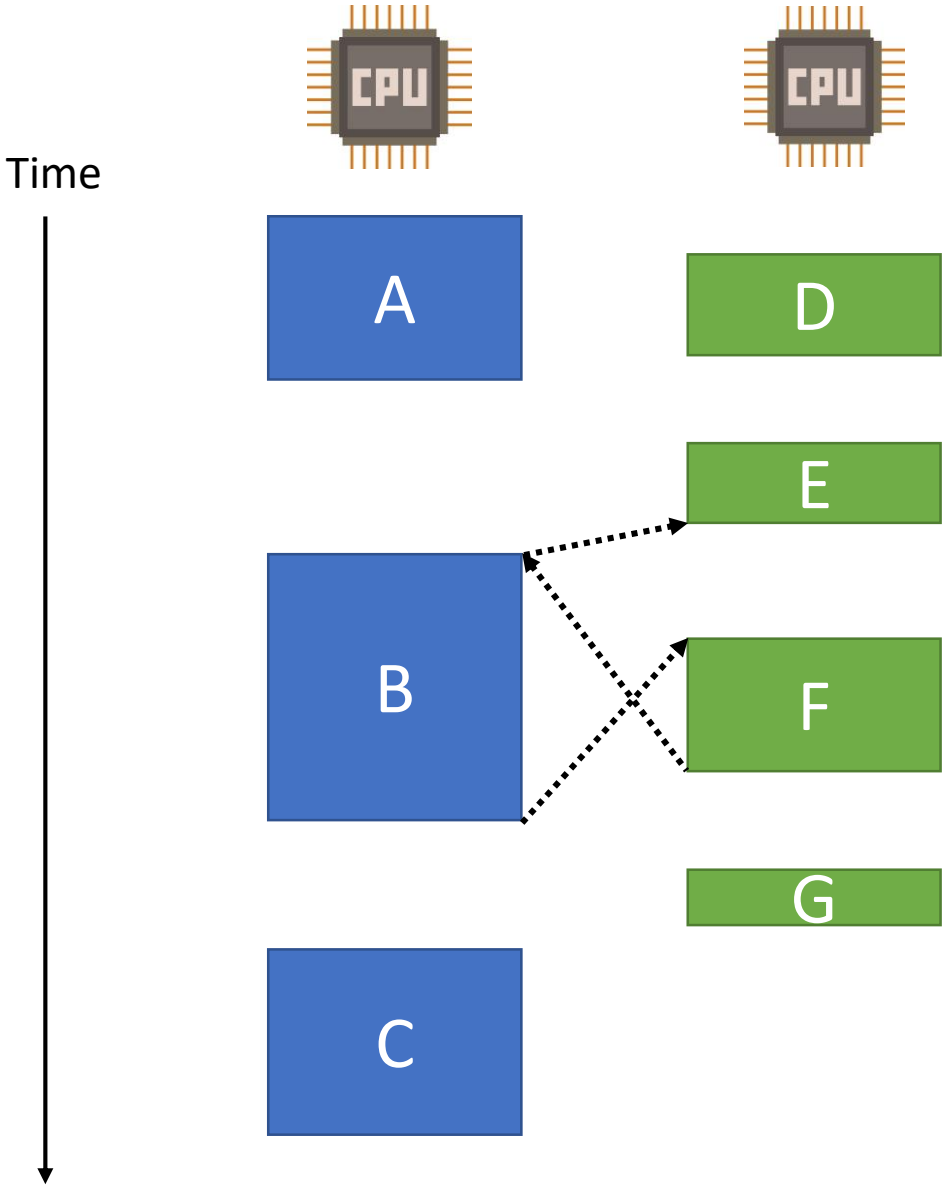

CPU
CPU
Time
A
B
C
D
E
F
G

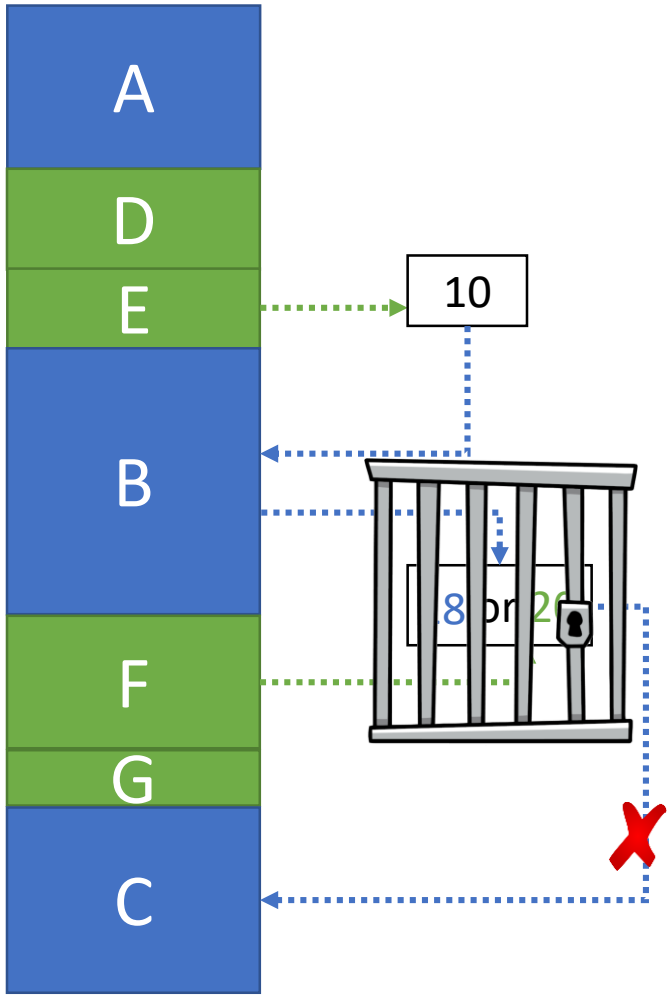

CPU
CPU
Time
A
D
E
B
F
G
C


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


CPU
CPU
Time
A
D
E
B
F
G
C


A
D
E
10
B
18 or 20
F
G
C


CPU
CPU
Time
A
D
E
B
F
G
C


A
D
E
B
F
G
C
10

Key Techniques

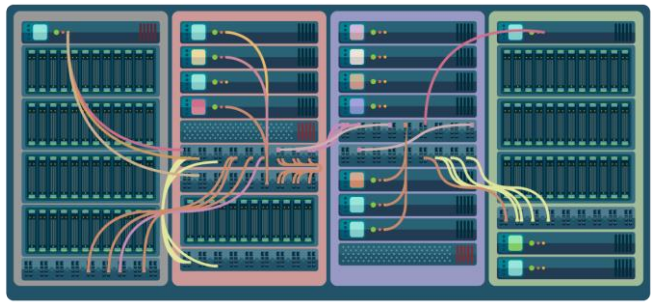
- Hardware Timestamps
  - Constructs a partial order

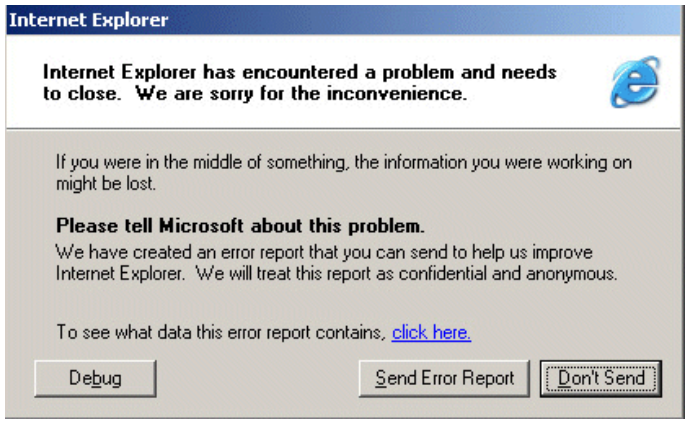
- Concurrent memory write detection
  - Constrains their usage to avoid propagating a wrong value

With REPT, …

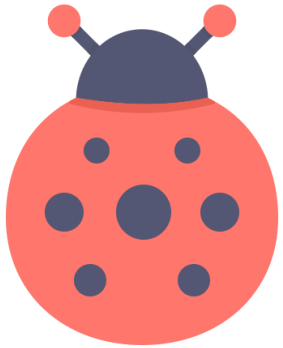
I want history information!

Hey, client, turn on tracing next time.

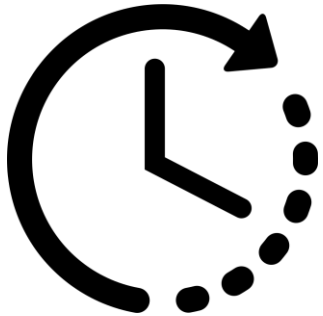




Demo

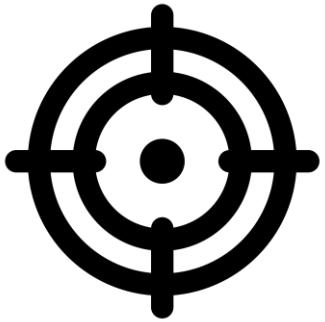


16 bugs



1-5% overhead

14 bugs



92% accuracy

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