REPT: Reverse Debugging of Failures in Deployed Software

Gefei Zuo and Jiacheng Ma

How to debug?

GDB

gcc -g whatever_you_want_to_debug.c
gdb a.out

then you execute the program step-by-step
then you fix the bug
then you submit the patch
then you go back to sleep

Key Idea

If you can replay a bug, you can (hopefully) fix it.

How to debug a deployed application which fails once a month?

Pray, and do (one of) the followings:

- ~~Wait for another month~~
- ~~Rubber duck debugging [1]~~
- Record-and-replay
  - state-of-the-art
  - too expansive (usually 20% ~ 200% runtime overhead)
  - almost impossible for deployment

---

[1] Rubber Duck Debugging, https://en.wikipedia.org/wiki/Rubber_duck_debugging

How to save a programmer's life?

- Help them replay (part of) the program
  - with low overhead and good accuracy
  - because if you can replay a bug, you can fix it

Figure is from https://www.usenix.org/sites/default/files/conference/protected-files/osdi18_slides_ge.pdf

REPT:
Reverse Execution with Processor Trace

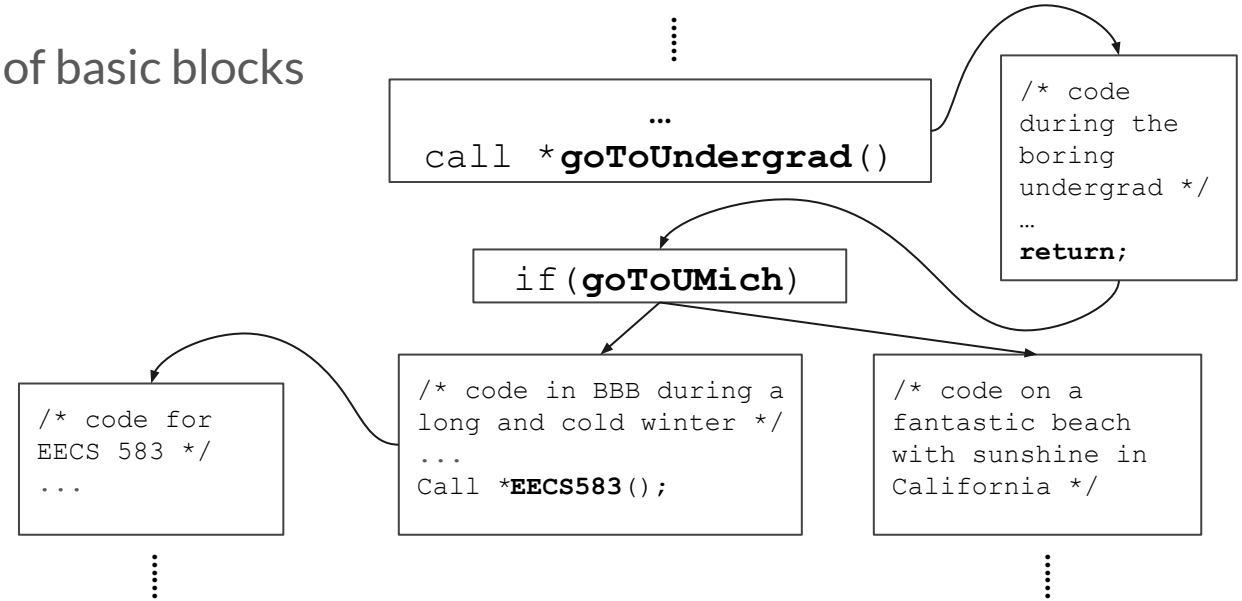
REPT = online hardware tracing + offline binary analysis

1-5% overhead

92% accuracy

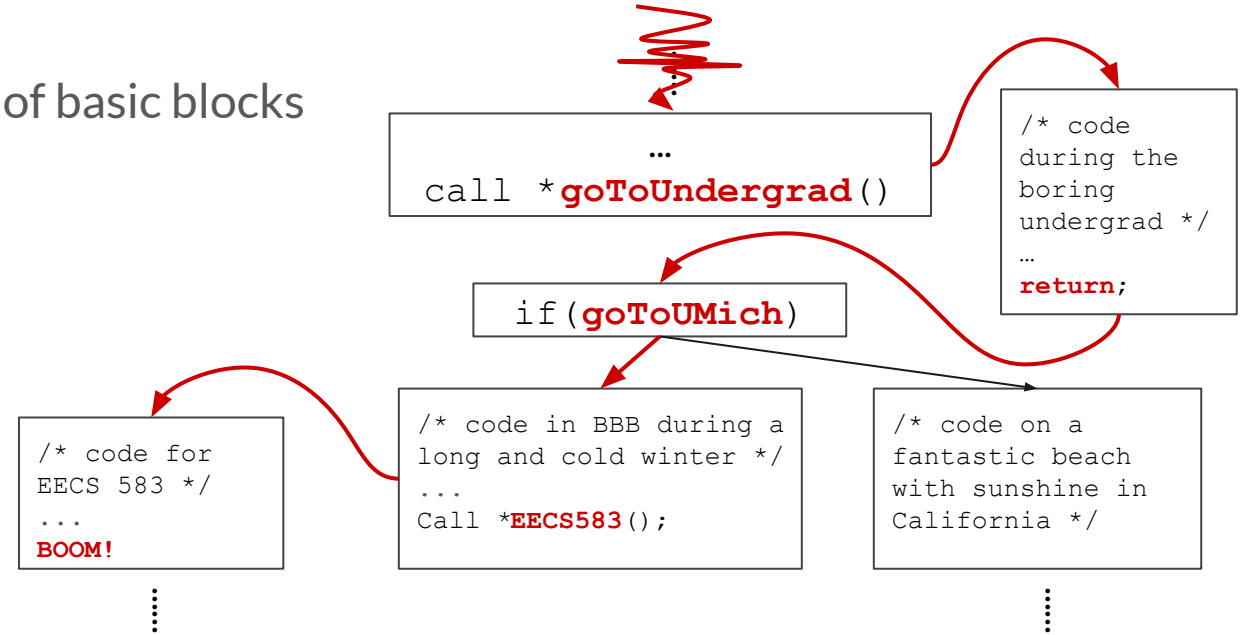
Background: Intel Processor Tracer

- ➔ Trace control flow of basic blocks
  - conditional jump
  - call and ret



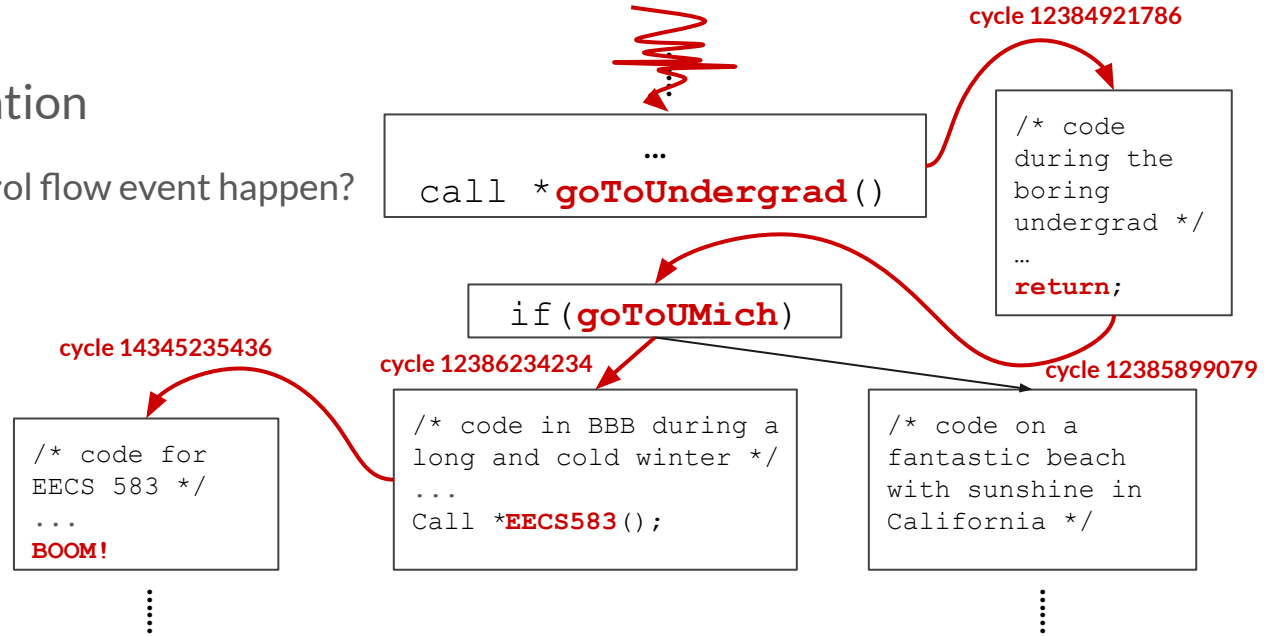
Background: Intel Processor Tracer

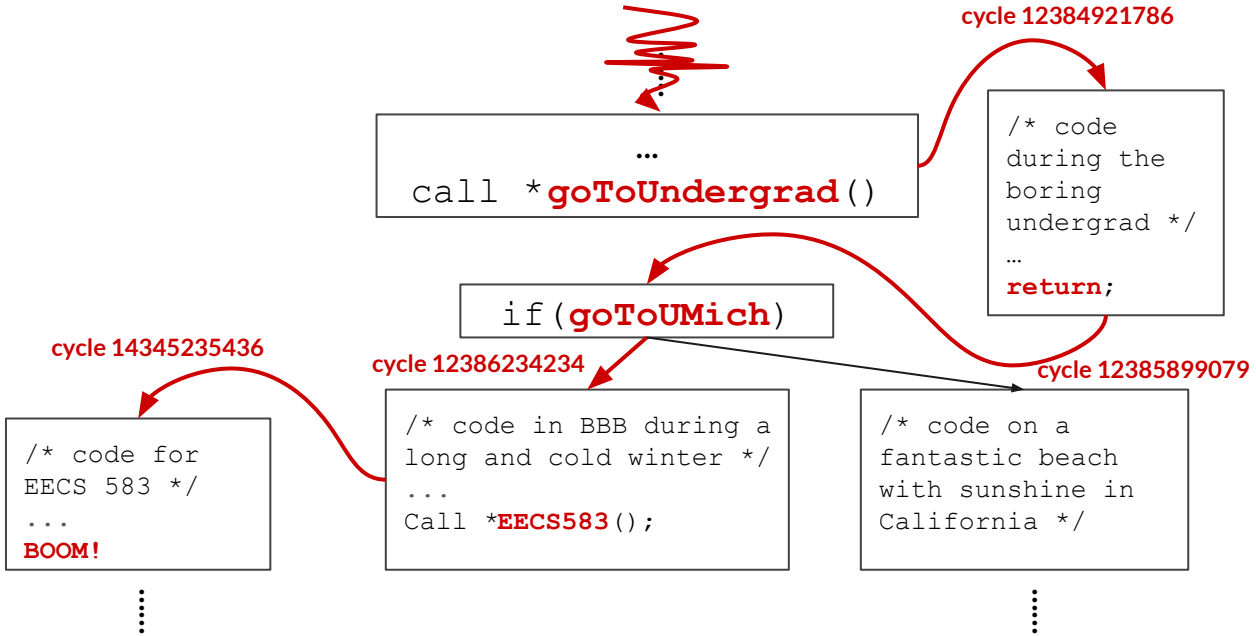
- Trace control flow of basic blocks
  - conditional jump
  - call and ret



Background: Intel Processor Tracer

- Trace time information
  - when does a control flow event happen?



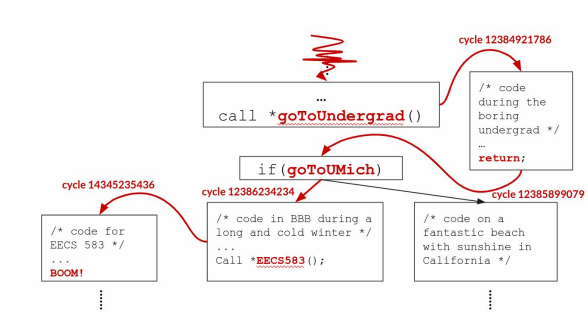

cycle 12384921786
…
call *goToUndergrad()
/* code during the boring undergrad */
…
return;
if(goToUMich)
cycle 14345235436
cycle 12386234234
cycle 12385899079
/* code for EECS 583 */
...
BOOM!
/* code in BBB during a long and cold winter */
...
Call *EECS583();
/* code on a fantastic beach with sunshine in California */

Background: Intel Processor Tracer

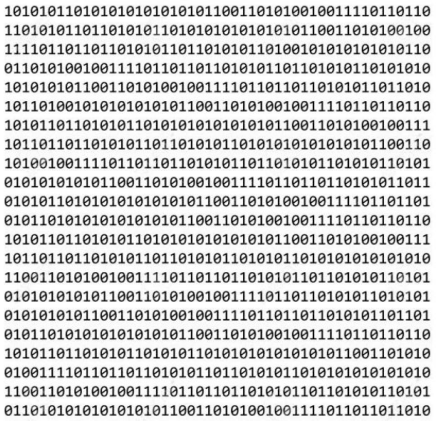
- ➔ Use an overwritable ring buffer
  - ◆ maintain a window about “important life decisions”
- ➔ Low runtime overhead (1-5%)
  - ◆ depends on workloads

REPT: when there's a crash

- We have:
  - Core Dump, i.e. memory layout and registers
  - Control flow trace from Intel PT



Core Dump



Figures are from https://www.usenix.org/sites/default/files/conference/protected-files/osdi18_slides_ge.pdf

REPT Data Recovery

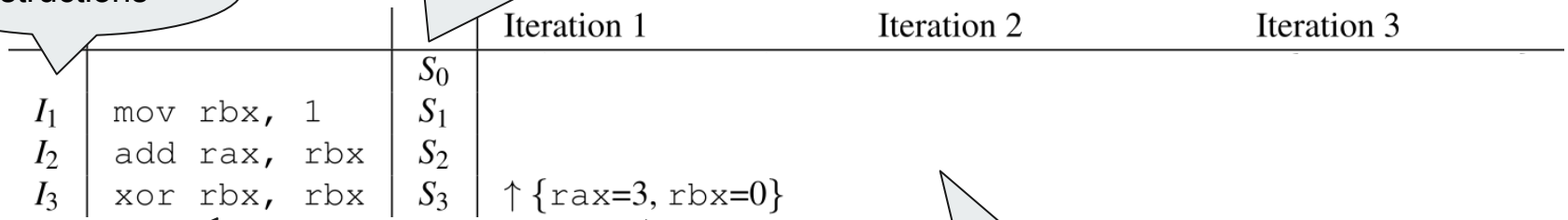
- ➔ Single-threaded execution reconstruction
    a. Execute the program reversely
- ➔ Multi-threaded execution reconstruction
    a. Take advantage of timestamps in PT Trace

Reversible Instructions: I2
Irreversible Instructions: I1, I3

A simp

Execution States
S[x-1] --I[x]--> S[x]

Instructions



Op dst src

Coredump

From coredump, rax = 3
From I2, rax = 1 + ? = ?

Example with Mem Write

			Iteration 1	Iteration 2	Iteration 3
		S_0	↑{rax=?, rbx=?, [g]=3} →		↑{rax=?, =?, [g]=2}
I_1	lea rbx, [g]	S_1	↑{rax=?, rbx=?, [g]=3}	↓{rax=?, rbx=g, [g]=3}	↑{rax=?, rb =g, [g]=2}
I_2	mov rax, 1	S_2	↑{rax=?, rbx=?, [g]=3}	↓{rax=1, rbx=g, [g]=3}	↑{rax=1, rbx=g, **[g]=2**}
I_3	add rax, [rbx]	S_3	↑{rax=3, rbx=?, **[g]=3**}	↓{**rax=3**, rbx=g, [g]=3}	↑{rax=3, rbx=g, **[g]=?**}
I_4	mov [rbx], rax	S_4	↑{rax=3, rbx=?, [g 3}	↓{rax 3, rbx=g, [g]=3}	↑{rax=3, rbx= [g]=3}
I_5	xor rbx, rbx	S_5	↑{rax=3, rbx=0, }	↓{rax= x=0, [g]=3} →	

[g] in S2 can be inferred by reversing I3

I4 writes to unknow
Ignore this mem

From core
From I3:

Iteration 4 is omitted here, which can recover this [g]

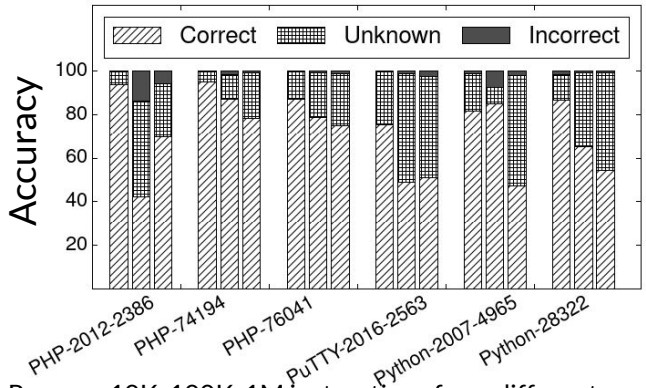
Key Techniques

1. Combine reverse execution and forward execution
    a. Reverse execution recovers states before reversible instructions
    b. Forward execution recovers states before irreversible instructions
2. Ignore unknown memory writes
    a. Use dereference level to correct missing memory writes
    b. The more dereference, the less confidence

Results Overview

1. 14 out of 16 bugs can be fixed by REPT
2. Only 1~5% overhead is introduced
3. On average memory states can be recovered with 92% accuracy

Program-BugId	# Iters	REPT (s)
Apache-24483	4	5.8
Apache-39722	5	3.0
Apache-60324	2	5.5
Chrome-784183	6	8.2
Nasm-2004-1287	10	18.6
Pbzip2	7	8.2
PHP-2007-1001	5	2.0
PHP-2012-2386	6	3.8
PHP-74194	7	6.3
PHP-76041	6	14.5
PuTTY-2016-2563	5	5.2
Python-2007-4965	12	10.5
Python-28322	18	17.5
Python-31530	6	10.6



Recover 10K, 100K, 1M instructions from different applications

Program-BugId	# Insts	Cor	Unk	Inc
Apache-24483	49	96.72%	1.64%	1.64%
Apache-39722	1,644	99.30%	0.70%	0.00%
Apache-60324	672	96.47%	1.83%	1.70%
Nasm-2004-1287	67,726	95.95%	3.70%	0.35%
PHP-2007-1001	54,475	99.08%	0.90%	0.02%
PHP-2012-2386	43,813	71.55%	25.40%	3.05%
PHP-74194	78,103	90.88%	7.82%	1.30%
PHP-76041	115	94.96%	3.60%	1.44%
PuTTY-2016-2563	677	99.55%	0.45%	0.00%
Python-2007-4965	1,043	95.04%	4.09%	0.87%
Python-28322	1,062	90.85%	8.60%	0.55%

What can MS do with PEDT

Questions?