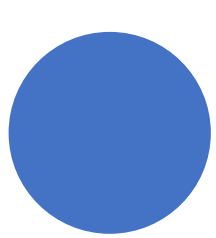
Swarnendu Biswas

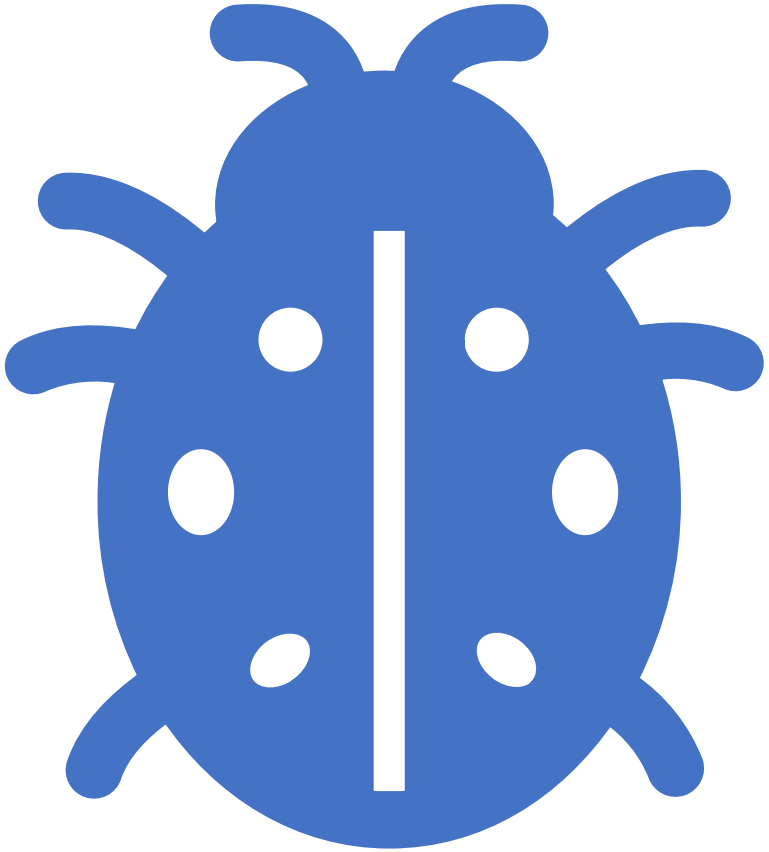
Systems Bootcamp 2018

CSE, IIT Kanpur

Program Analysis: Its Need and Usefulness



Production Software contains **BUGS**!



Production Software contains **BUGS**!

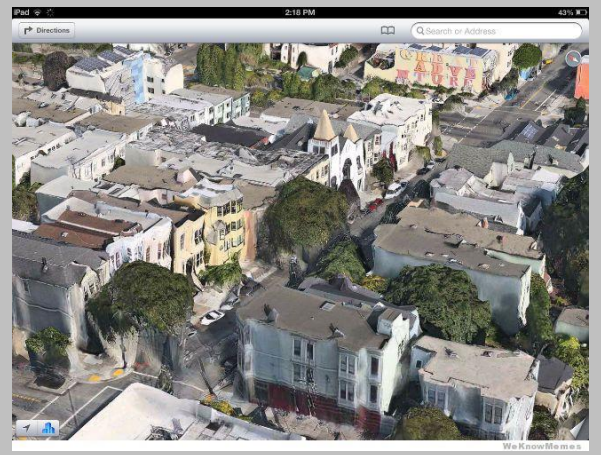
- **AT&T hangs up its long-distance service (1990)**
  - For nine hours in January 1990 no AT&T customer could make a long-distance call. The problem was the software that controlled the company's long-distance relay switches—software that had just been updated. AT&T wound up losing $60 million in charges that day—a very expensive bug.
- **The Pentium chip's math error (1993)**
- **The Mars Climate Orbiter disintegrates in space (1998)**
  - NASA's $655-million robotic space probe plowed into Mars's upper atmosphere at the wrong angle, burning up in the process. The problem? In the software that ran the ground computers the thrusters' output was calculated in the wrong units (pound–seconds instead of newton–seconds).


Getty

Can you identify this city?

Do You Still Need to be Convinced?

- **Windows locks out non-software pirates (2007)**
  - For 19 hours on August 24, 2007, anyone who tried to install Windows was told, by Microsoft's own antipiracy software (called Windows Genuine Advantage) that they were installing illegal copies.
- **Apple Maps gives us directions to nowhere (2012)**



NASDAQ

business

READY

Nasdaq's Facebook Glitch Came From Race Conditions

Joab Jackson
@Joab_Jackson

May 21, 2012 12:30 PM

The Nasdaq computer system that delayed trade notices of the Facebook IPO on Friday was plagued by race conditions, the stock exchange announced Monday. As a result of this technical glitch in its Nasdaq OMX system, the market expects to pay out US$13 million or even more to traders.

A number of trading firms lost money due to mismatched Facebook share prices. About 30 million shares' worth of trading were affected, the exchange estimated.

NASDAQ's Glitch Cost Facebook Investors ~$500M. It Will Pay Out Just $62M. IPO Elsewhere.

Josh Constine @joshconstine / 6 years ago

Comment

KILLED BY A MACHINE: THE THERAC-25

by: Adam Fabio

139 Comments

October 26, 2015

The Therac-25 was not a device anyone was happy to see. It was a radiation therapy machine. In layman's terms it was a "cancer zapper"; a linear accelerator with a human as its target. Using X-rays or a beam of electrons,

SEARCH

Search ... SEARCH

NEVER MISS A HACK

SUBSCRIBE

Enter Email Address SUBSCRIBE

Therac-25 Accident

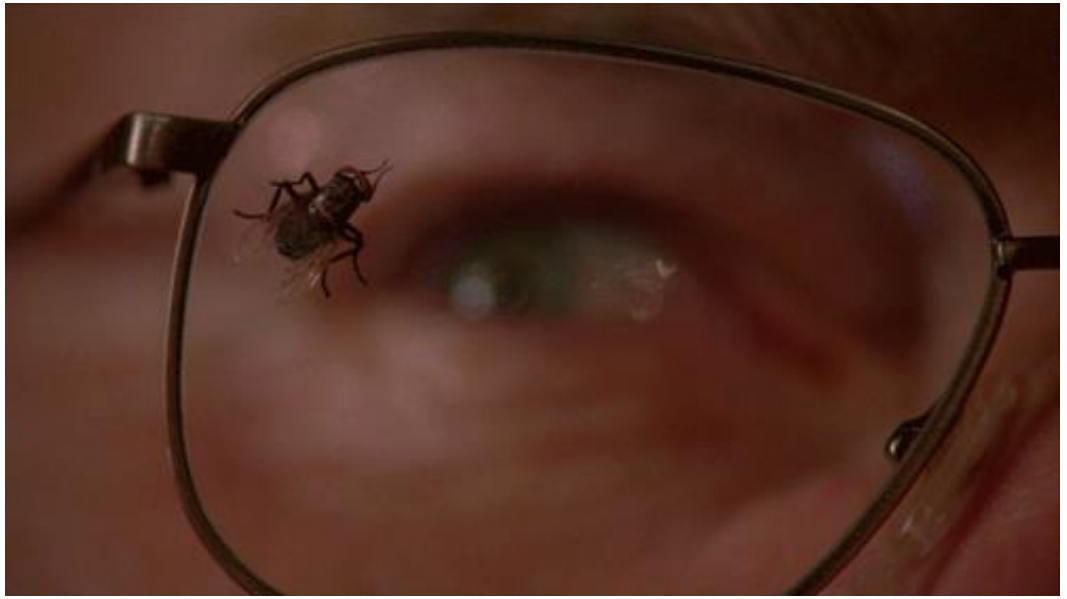
- Therac-25 was a computer-controlled radiation therapy machine
- It was involved in at least six accidents between 1985 and 1987, in which patients were given massive overdoses of radiation. Because of concurrent programming errors, it sometimes gave its patients radiation doses that were hundreds of times greater than normal, resulting in death or serious injury.

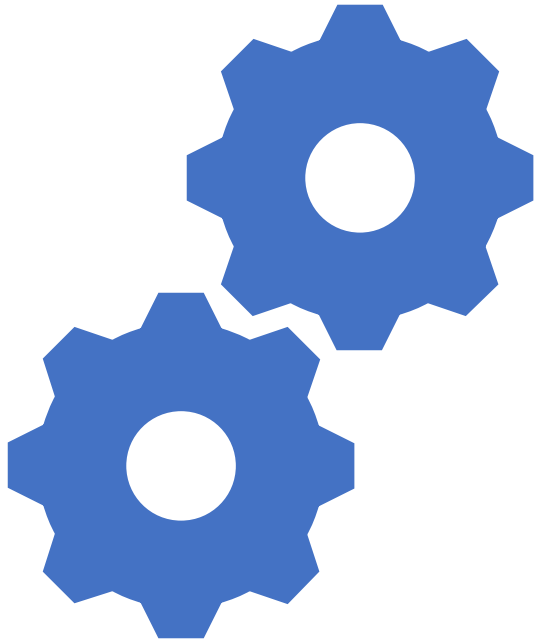
---

https://en.wikipedia.org/wiki/Therac-25

What is the Solution?

- Program testing
- Continuous error monitoring
- Fault-tolerant algorithms



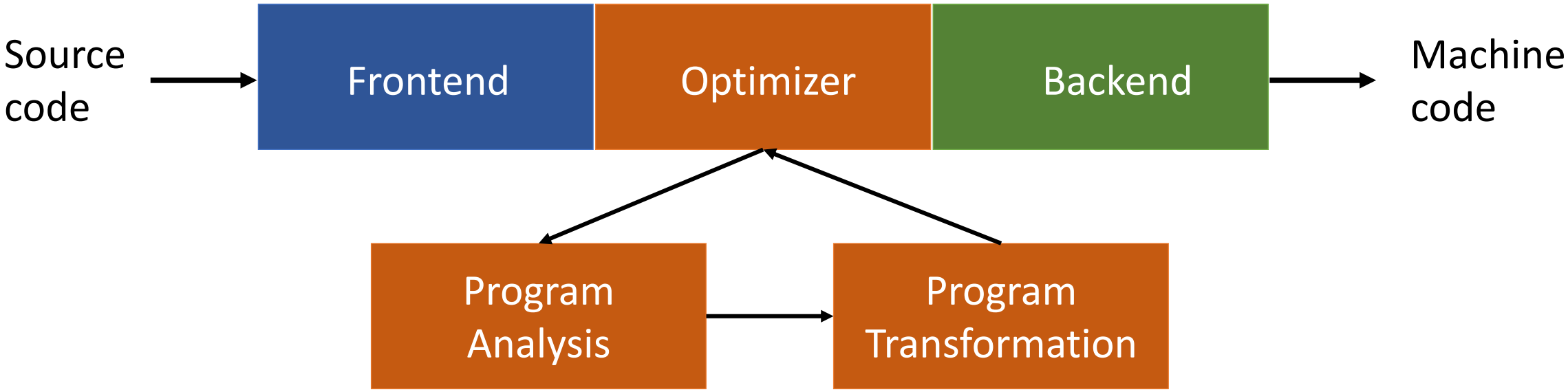


Program Analysis

- Allows monitoring program execution, predicting program behavior
- Two major directions: program optimization and correctness
- Program optimization
  - LICM, CSE, automatic parallelization
- Correctness
  - Error detection, property enforcement, verification
  - Security!

Classical Compiler Design

Classical Notion of Program Analysis



Analyses and Transformations

Analysis

- Available expressions analysis
- Detection of loop invariants
- Detection of induction variables
- Equivalent expression analysis
- Live variables analysis

Transformations

- Common subexpression elimination
- Loop invariant code motion
- Strength reduction
- Copy propagation
- Dead code elimination

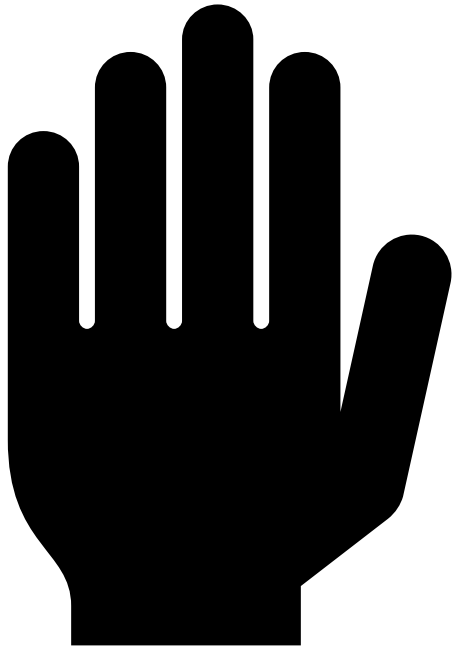
But there is more to program analysis…

Static Initialization Order Issue in C++

```
$ cat a.cc

#include <stdio.h>

extern int extern_global;

int __attribute__((noinline))
read_extern_global() {
  return extern_global;
}

int x = read_extern_global() + 1;

int main() {
  printf("%d\n", x);
  return 0;
}
```

```
$ cat b.cc

int foo() {
  return 42;
}

int extern_global = foo();
```

https://github.com/google/sanitizers/wiki/AddressSanitizerInitializationOrderFiasco



What is the value of x?

What is the value of x?

```
$ clang++ a.cc b.cc && ./a.out
```

```
$ clang++ b.cc a.cc && ./a.out
```

More Program Analysis Examples

Find program hotspots

- Optimize and test hotspots more thoroughly
- **80-20 rule**
  - 20% of the program responsible for 80% of execution time

Memory reference errors

- Uninitialized memory, memory leaks, double free, array bounds checks

Initialization order bugs

Type checking

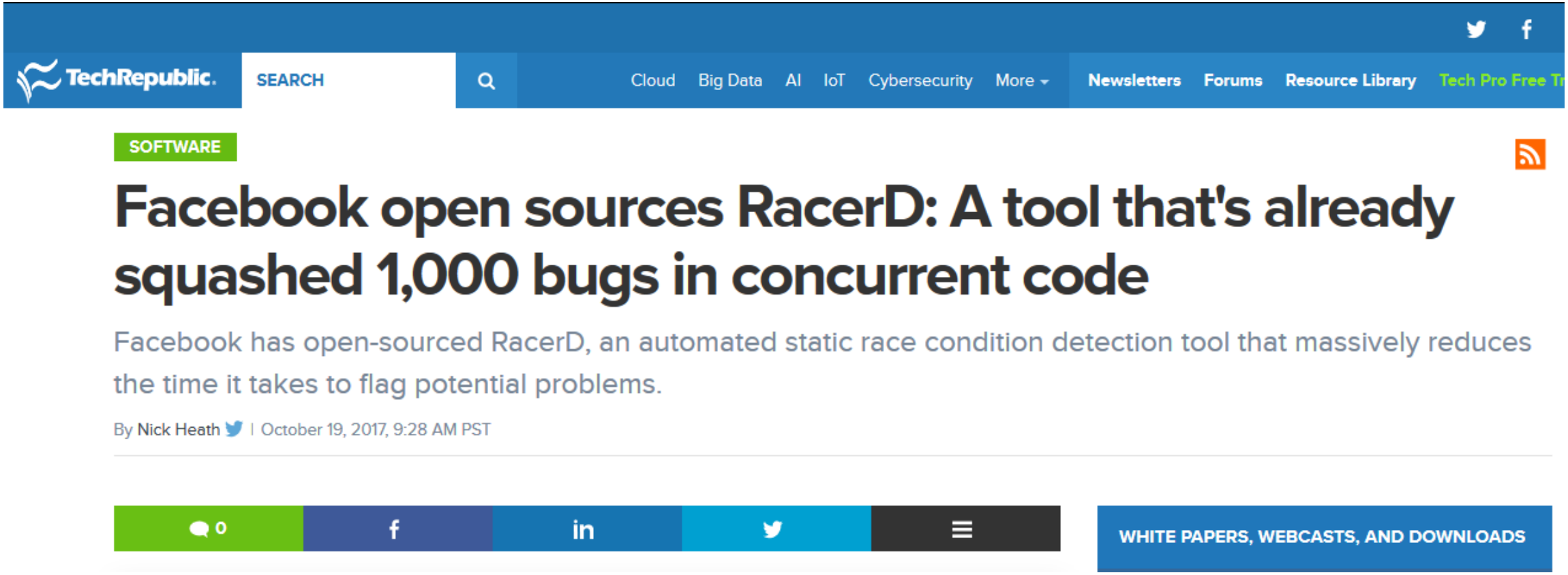
many more….

Static Analyses

- Does not execute the program - no runtime overhead
- Control flow analysis, reaching definitions, dominance computation, type checking
- Can prove absence of bugs (sometimes)

Features of Static Analyses

- Can potentially reason about all program behaviors
  - Suffers from state space explosion
- Suffers from false negatives and false positives
  - Many language features complicate static analysis – dynamic class loading and reflection in Java


TechRepublic.
SEARCH
Cloud
Big Data
AI
IoT
Cybersecurity
More
Newsletters
Forums
Resource Library
SOFTWARE
Facebook open sources RacerD: A tool that's already squashed 1,000 bugs in concurrent code
Facebook has open-sourced RacerD, an automated static race condition detection tool that massively reduces the time it takes to flag potential problems.
By Nick Heath | October 19, 2017, 9:28 AM PST
0
WHITE PAPERS, WEBCASTS, AND DOWNLOADS

Dynamic Analyses

- Monitors program properties by running the program over one or more executions
- Testing, data race detection, software transactional memory

Dynamic Analyses Features

- Monitors only the **current** program execution
  - Improved precision (can avoid false positives)
    - E.g., dynamic program slicing → computes accurate flow dependences
  - Incurs runtime overhead
  - Scales better than static analyses
- Requires engineering effort
  - Insight → build prototype → evaluate → rethink → iterate

Implementing Dynamic Program Analyses

- Instrument the program
  - Add analysis code to the program
  - Possibly does not affect the semantics of the program
  - Overhead is important
    - More returns from optimizing the fast path
- Online vs offline analysis

Popular Program Analyses Tools

Static analyses	Chord, Language linters (cpplint, pylint), checkstyle, FindBugs, Facebook Infer, Clang, Soot
Dynamic analyses	Intel Pin, Valgrind, Profilers (gprof), gcov, AddressSanitizer/ThreadSanitizer
Compilers/Runtimes	LLVM, GCC, Jikes RVM, OpenJDK

TRY TO UTILIZE THESE in YOUR WORK!

- https://github.com/google/sanitizers
  - **AddressSanitizer** and LeakSanitizer
  - **ThreadSanitizer**
  - **MemorySanitizer**
- **Valgrind** http://www.valgrind.org/
- **Clang** https://clang.llvm.org/
- …

Static vs Dynamic Analysis

Program + Input = Behavior

Thomas Ball. The Essence of Dynamic Analysis.

Static Analysis

Program + ~~Input~~ = Behavior

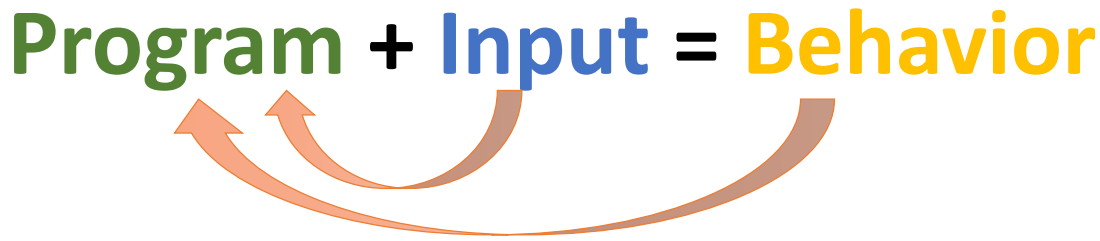
No notion of input, there has to be conservative

- Input insensitive

Program guides the behavior

Thomas Ball. The Essence of Dynamic Analysis.

Dynamic Analysis



Input and the behavior is a guide the program

- Input sensitive

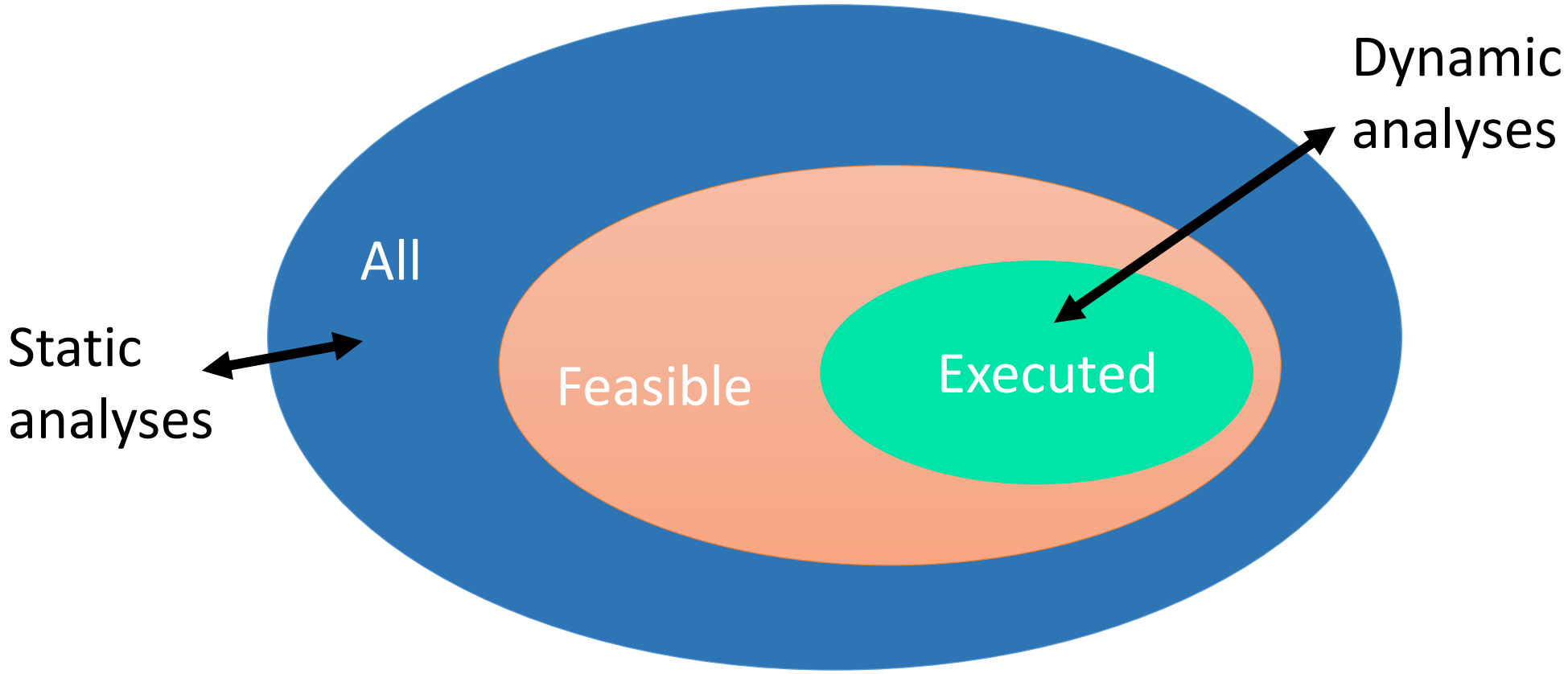
Thomas Ball. The Essence of Dynamic Analysis.

Static vs Dynamic Analysis

- Static analysis is complete, dynamic analysis is incomplete
- Dynamic analysis leaves feasible paths unexplored
  - May conclude that a property holds while it really doesn’t (precise)
- Static analysis explores unfeasible paths
  - May conclude a property holds while it really doesn’t (safe but imprecise)

Thomas Ball. The Essence of Dynamic Analysis.

Control Flow Paths



Thomas Ball. The Essence of Dynamic Analysis.

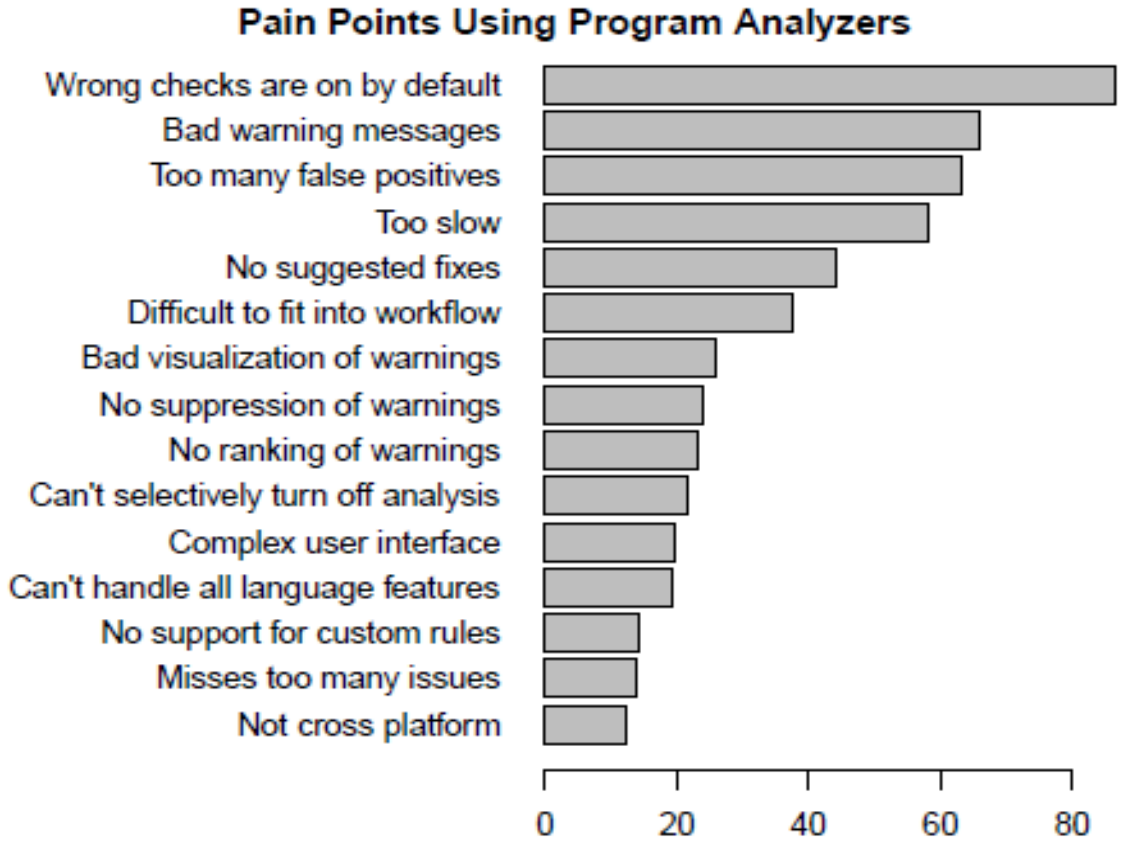


Figure 1: Pain points reported by developers when using program analyzers.

M. Christakis and C. Bird. What Developers Want and Need From Program Analysis: An Empirical Study. ASE 2016.

New Research Directions

- Hybrid analysis
  - Leverage the strengths of static analysis to improve the runtime overheads of dynamic analysis

- Predictive analysis
  - Improve the coverage of dynamic analyses by trying to reason about **other** feasible paths
  - Schedule sensitive branches

B. Kasikci et al. RaceMob: Crowdsourced Data Race Detection. SOSP'13.
J. Huang and L. Rauchwerger.. Finding Schedule-Sensitive Branches.ESEC/FSE'15.

References

- https://web.stanford.edu/class/cs240/old/sp2014/readings/therac-25.pdf
- https://en.wikipedia.org/wiki/Therac-25
- https://hackaday.com/2015/10/26/killed-by-a-machine-the-therac-25/
- https://techcrunch.com/2013/03/25/ip-oh-my-gosh-all-that-money-just-disappeared/
- https://www.computerworld.com/article/2504676/financial-it/nasdaq-s-facebook-glitch-came-from--race-conditions-.html
- https://www.scientificamerican.com/article/pogue-5-most-embarrassing-software-bugs-in-history/
- https://www.cs.odu.edu/~tkennedy/cs350/sum16/Public/analysis/index.html

Swarnendu Biswas

Systems Bootcamp 2018

CSE, IIT Kanpur

Program Analysis: Its Need and Usefulness