CE 440 Introduction to Operating System

Lecture 21: System Reliability
Fall 2025

Prof. Yigong Hu

BOSTON UNIVERSITY

Slides courtesy of Manuel Egele, Ryan Huang and Baris Kasikci

Administrivia

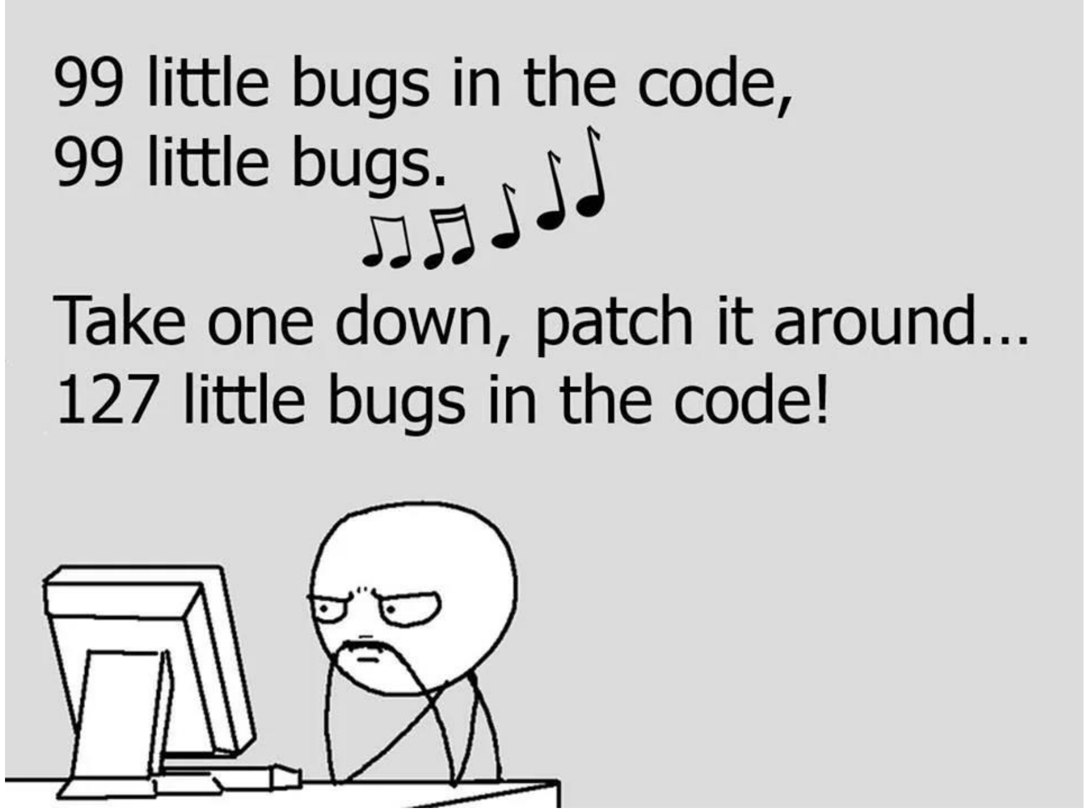
Homework 5 is released

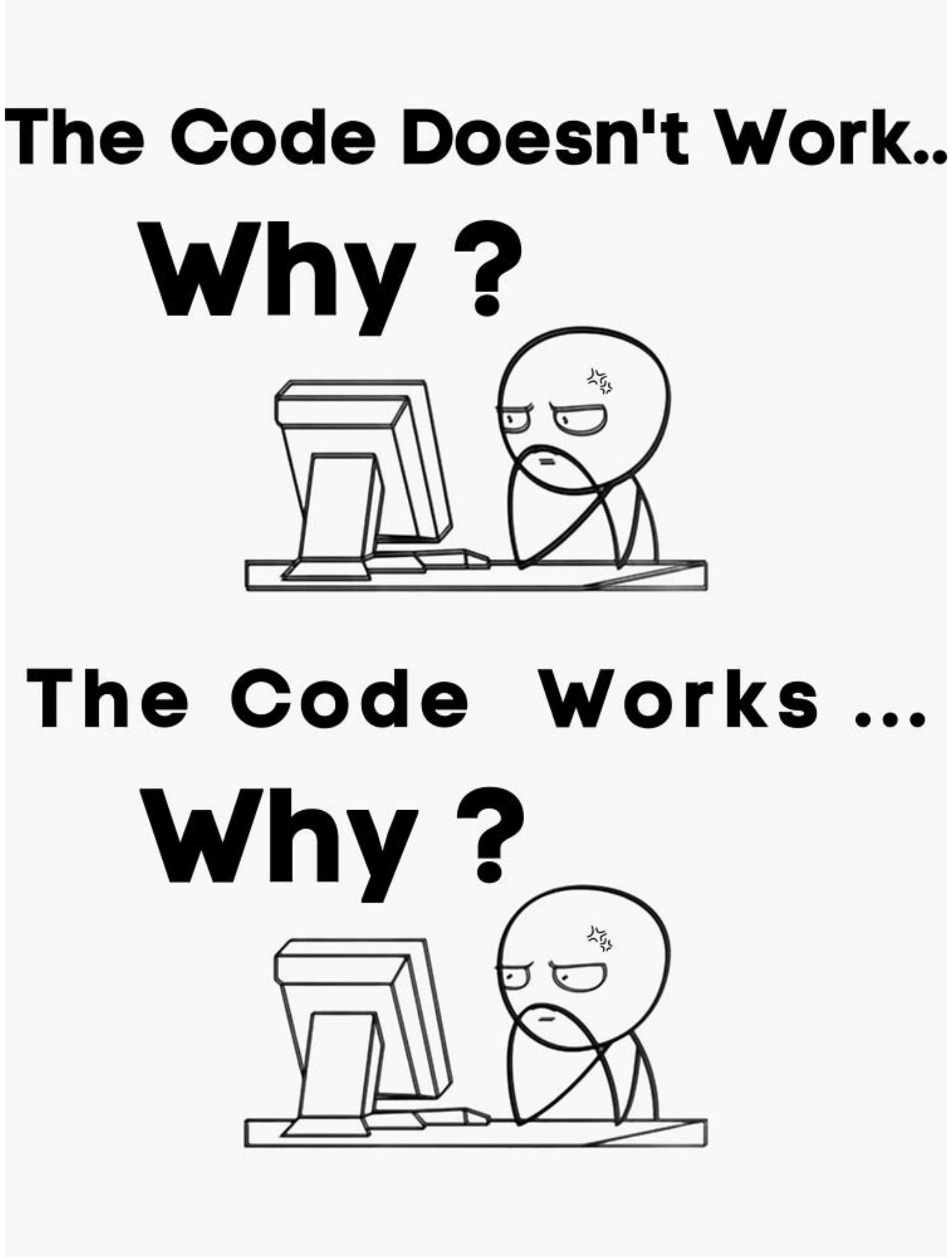
Bug Are Everywhere

- **As both programmers and users, you often struggle with buggy software**.

- **Students are taught programming, but not much on reliability.**

- **Industry invests significant time and resources on reliability.**

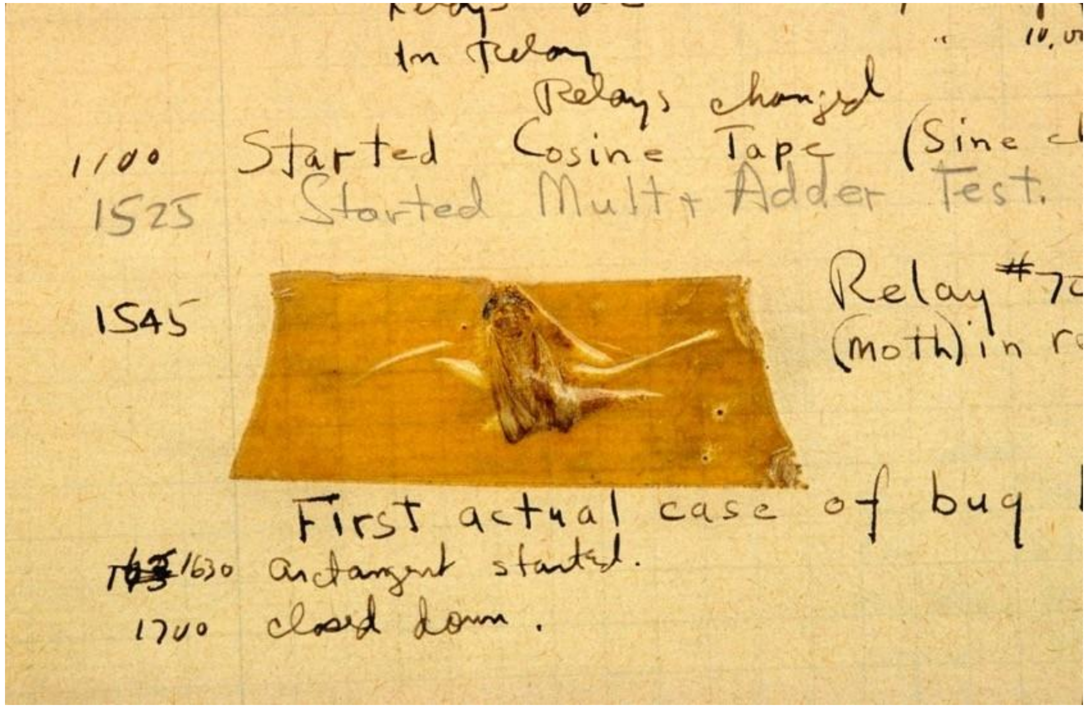

99 little bugs in the code,
99 little bugs.
♫♫♩♩♩
Take one down, patch it around...
127 little bugs in the code!


The Code Doesn't Work..
Why ?
The Code Works ...
Why ?

Bug

The origin of “bug” is literally a bug

- Coined by U.S. Navy Admiral and computer science pioneer, Grace Hopper
- A moth got into a mechanical relay of Mark II supercomputer, jamming the system.



In Relay

Relays changed

1100 Started Cosine Tape (Sine

1525 Started Mult+ Adder Test.

1545 Relay #70 (moth) in r

First actual case of bug

~~143~~1630 antangent started.

1700 closed down.

grace hopper conference

Bugs in Programmers' Eyes

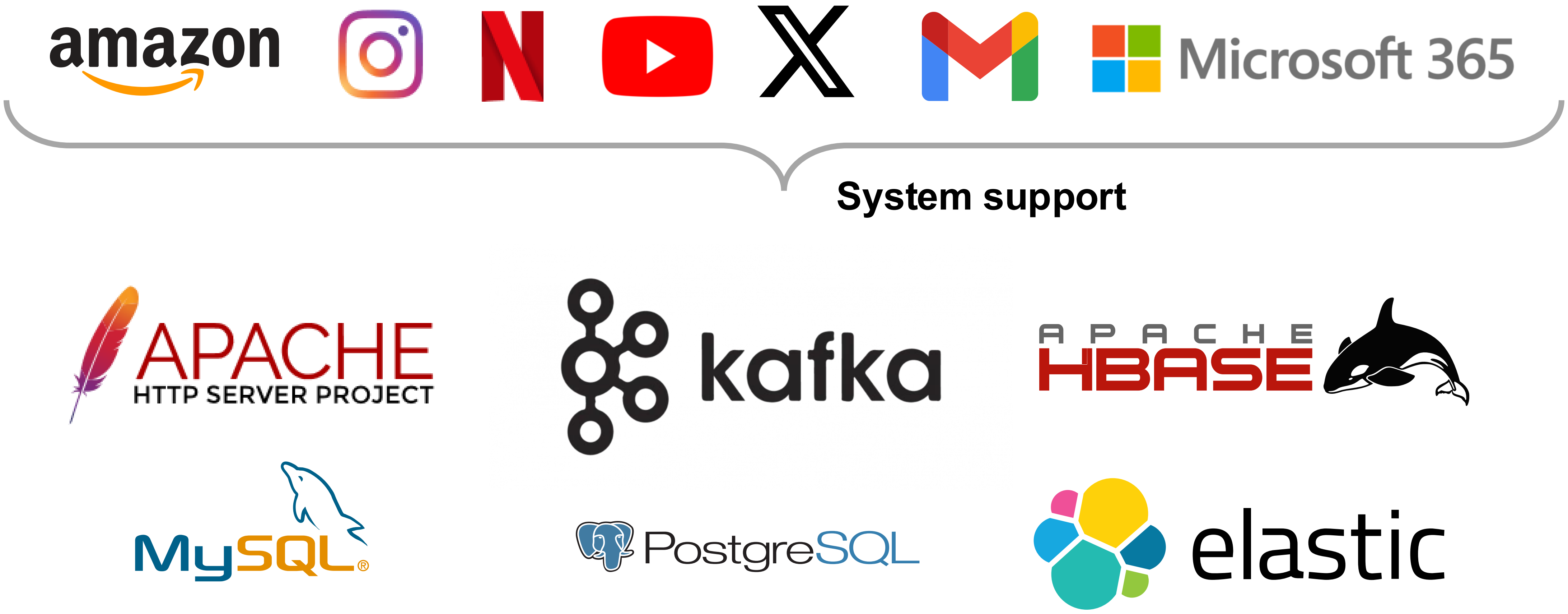
Programmer's language translation guide

What programmers say	What programmers mean
Horrible hack	Horrible hack that I didn't write
Temporary workaround	Horrible hack that I wrote
It's broken	There are bugs in your code
It has a few issues	There are bugs in my code
Obscure	Someone else's code doesn't have comments
Self-documenting	My code doesn't have comments
I can read this Perl script	I wrote this Perl script
I can't read this Perl script	I didn't write this Perl script
Bad structure	Someone else's code is badly organized
Complex structure	My code is badly organized
Bug	The absence of a feature I like
Out of scope	The absence of a feature I don't like
Clean solution	It works and I understand it

What programmers say	What programmers mean
We need to rewrite it	It works but I don't understand it
Emacs is better than Vim	It's too peaceful here, let's start a flame war
Vim is better than Emacs	It's too peaceful here, let's start a flame war
IMHO	You are wrong
Legacy code	It works but no one knows how
^X^Cquit^\[ESC][ESC]^C	I don't know how to quit Vim
That can't be done	It can be done, but it's boring and I don't want to do it
No problem, people do this all the time. It's an easy fix.	You might be the most idiotic person I've ever encountered
Put that bug in the backlog with low priority	Let's agree: nobody ever mention it again and ppl who do, will be shot
These test environments are too brittle	Works on my machine. Have you tried re-starting yours?
Proof-of-Concept	What I wrote
Perfect solution	How sales & marketing are promoting it

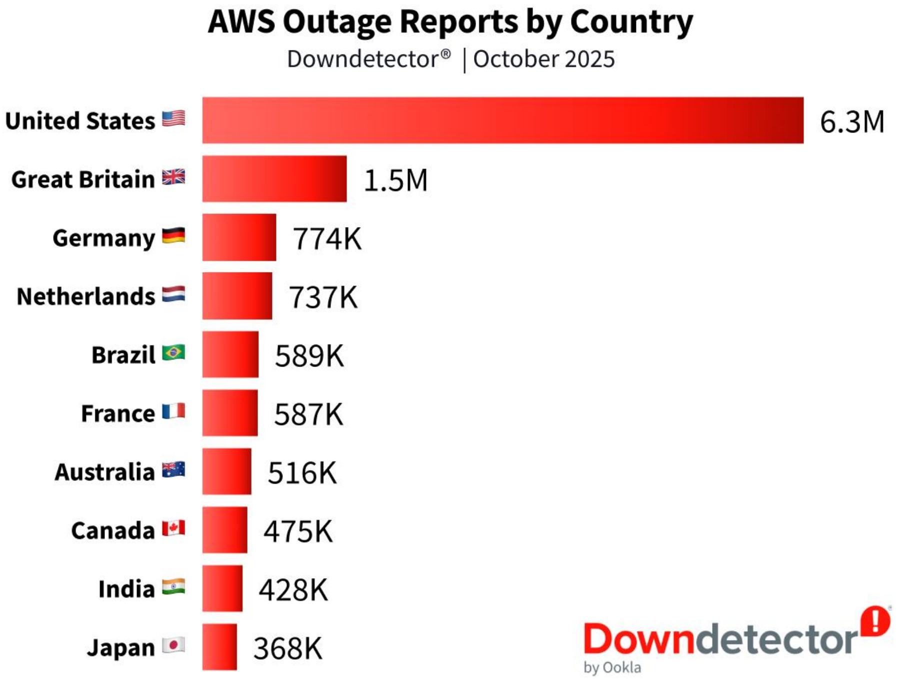
Fix a Bug in Production

Software Systems Provide Essential Services



System support

Software Failure Is Costly



158 Available
88 Unavailable
3 No Data

One race condition causes Amazon’s 15-Hour AWS Outage and broke the internet

In 2019, a disruption in Google’s network cause 2.5% dropping of YouTube views[2]

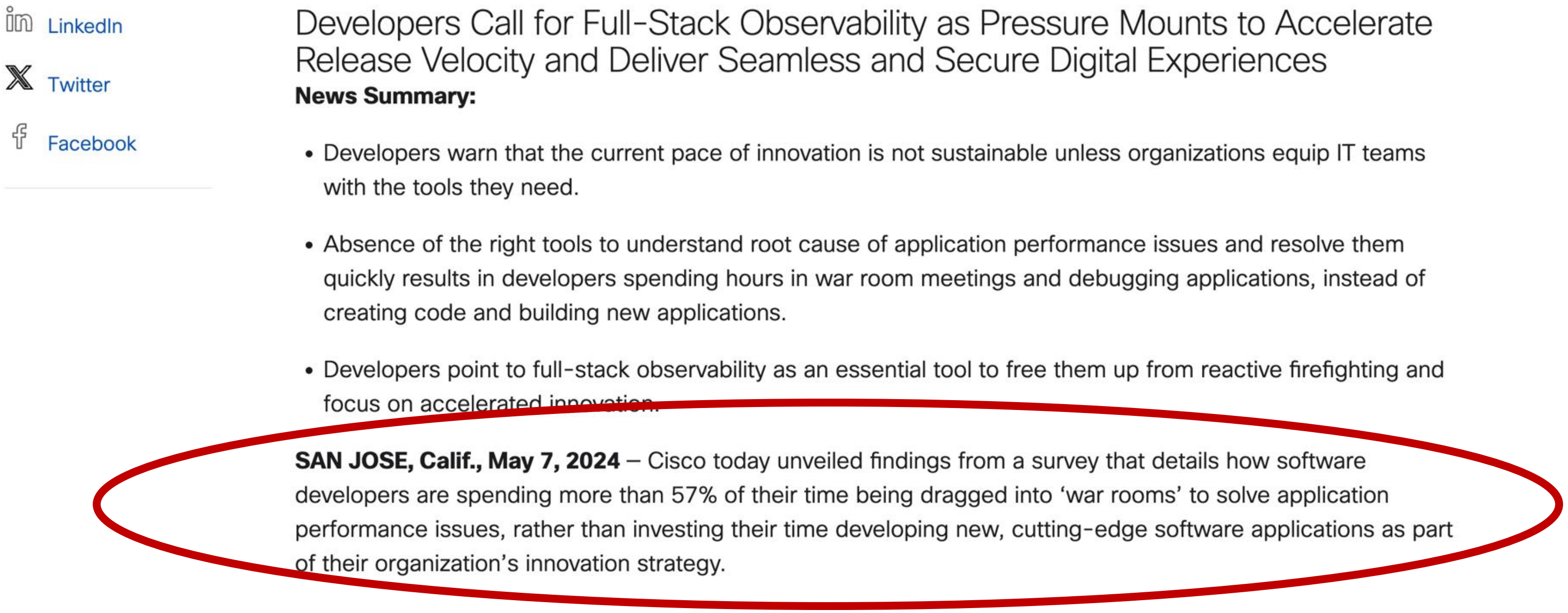
[1] Akamai Online Retail Performance Report

[2] Inside Google Cloud: An Update on Sunday’s Service Disruption

Why Is System Reliability Important?

- Ind eliability

More Time Firefighting Issues Than Delivering Innovation



Developers Call for Full-Stack Observability as Pressure Mounts to Accelerate Release Velocity and Deliver Seamless and Secure Digital Experiences

News Summary:

- Developers warn that the current pace of innovation is not sustainable unless organizations equip IT teams with the tools they need.
- Absence of the right tools to understand root cause of application performance issues and resolve them quickly results in developers spending hours in war room meetings and debugging applications, instead of creating code and building new applications.
- Developers point to full-stack observability as an essential tool to free them up from reactive firefighting and focus on accelerated innovation.

SAN JOSE, Calif., May 7, 2024 – Cisco today unveiled findings from a survey that details how software developers are spending more than 57% of their time being dragged into 'war rooms' to solve application performance issues, rather than investing their time developing new, cutting-edge software applications as part of their organization's innovation strategy.

Why Is System Reliability Important?

Industry spends significant time & resources on reliability

- Testing, finding bugs, debugging, patching, etc

"We have as many testers as we have developers. And testers spend all their time testing, and developers spend half their time testing. We're more of a testing, a quality software organization than we're a software organization."

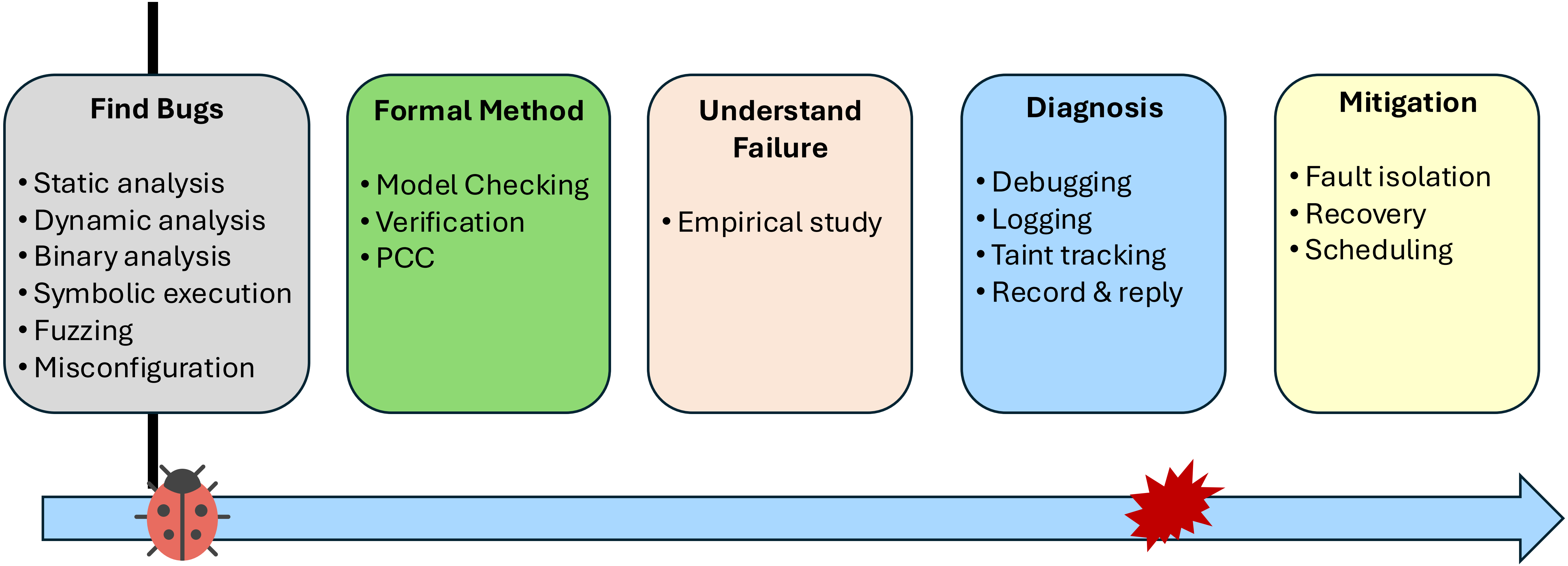
-- Bill Gates

Why Is System Reliability Important?

Industry spends significant time & resources on reliability

- Testing, finding bugs, debugging, patching, etc
- Many tech companies have dedicated teams working on it
  - e.g., Site Reliability Engineering (SRE) team

Topic in Computer System Reliability



Find Bugs

- Static analysis
- Dynamic analysis
- Binary analysis
- Symbolic execution
- Fuzzing
- Misconfiguration

Formal Method

- Model Checking
- Verification
- PCC

Understand Failure

- Empirical study

Diagnosis

- Debugging
- Logging
- Taint tracking
- Record & reply

Mitigation

- Fault isolation
- Recovery
- Scheduling

What Is Reliability?

Reliability is an important metric about a system's quality

- Other metrics: efficiency, security, usability, maintainability, etc.

Definition

The probability that a system operates without failure in a given period of time.

$$Reliability = 1 - Probability(Failure)$$

Can be expressed as failure rate λ

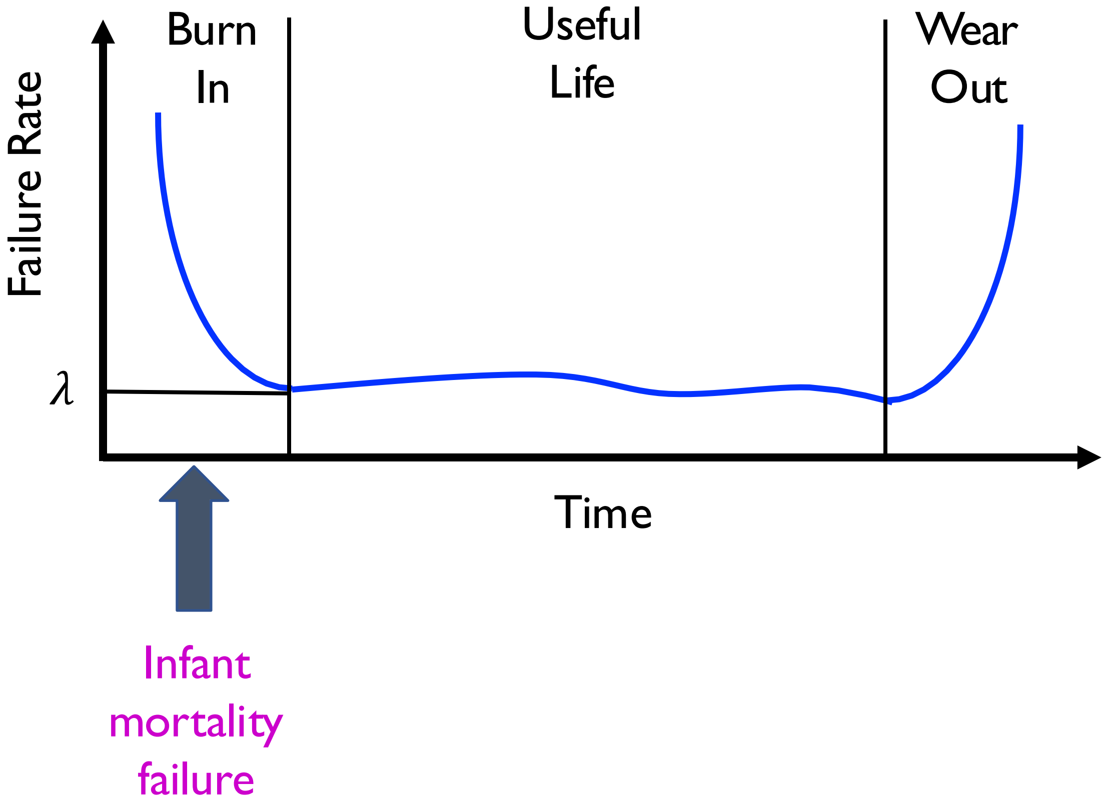
Mean Time Between Failure (MTBF, $1/\lambda$) is often reported

- MTBF = 2000 hours => λ = 0.0005/hour

Reliability Curve

The failure rate of a system usually depends on time

- Hard disk’s failure rate in its fifth year > the rate in the first year



The bathtub curve

Software Reliability Vs. Hardware Reliability

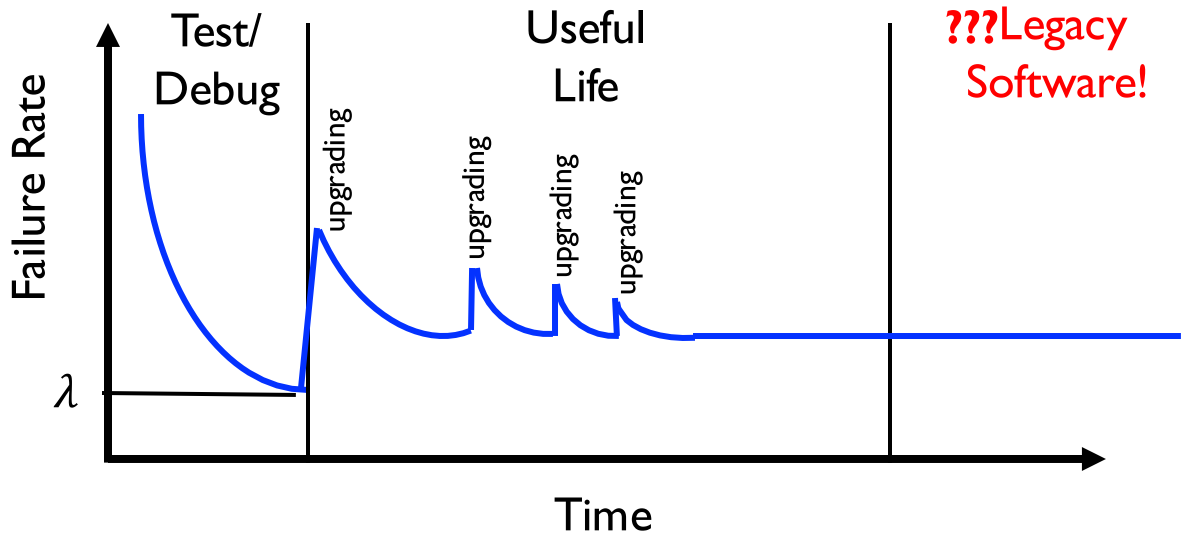
Hardware typically exhibit the bathtub curve, but software don’t

- Why?
- Hardware faults are mostly *physical faults*
- Software faults are *design/implementation faults*
    - Hard to visualize, classify, detect, and correct
    - Related to human factors, which we often don’t understand well
- Software does not need “manufacturing”
    - Its quality does not change much once it’s deployed

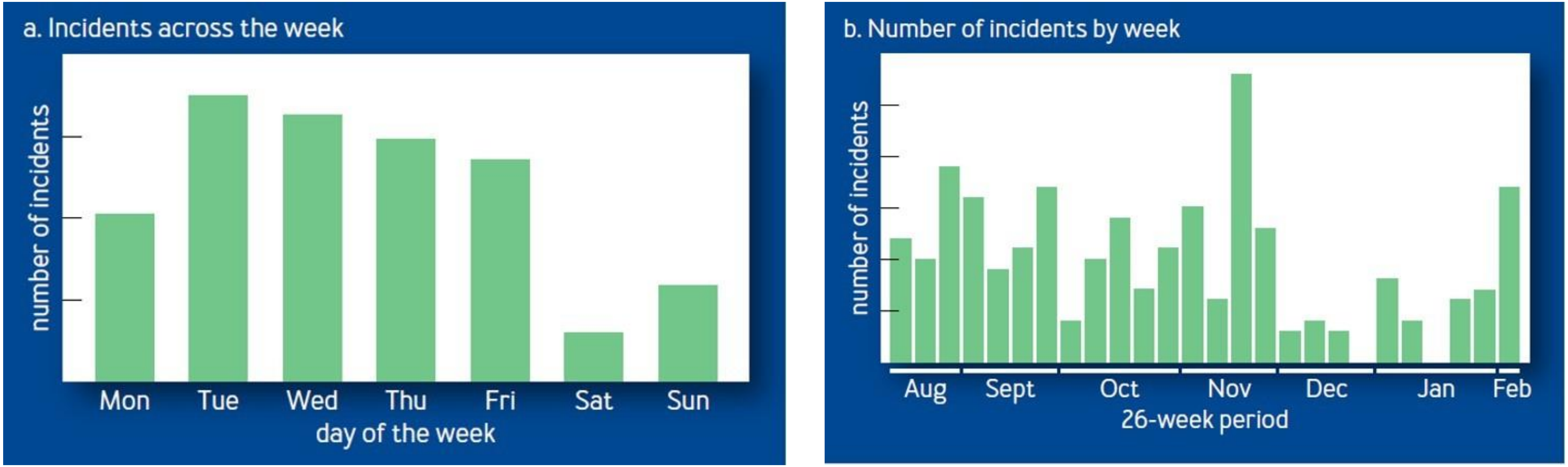
What’s The “Bathtub Curve” For Software?

What is the one major reason software fails?

- Upgrades!



Real-World System Failure Rates: Meta



“Fail at Scale” [ACM Queue]

Why Do Systems Fail?

Hardware factors

- Power loss
- Disk wears out
- CPU random bit flip
- Memory corruption
- Room temperature too hot

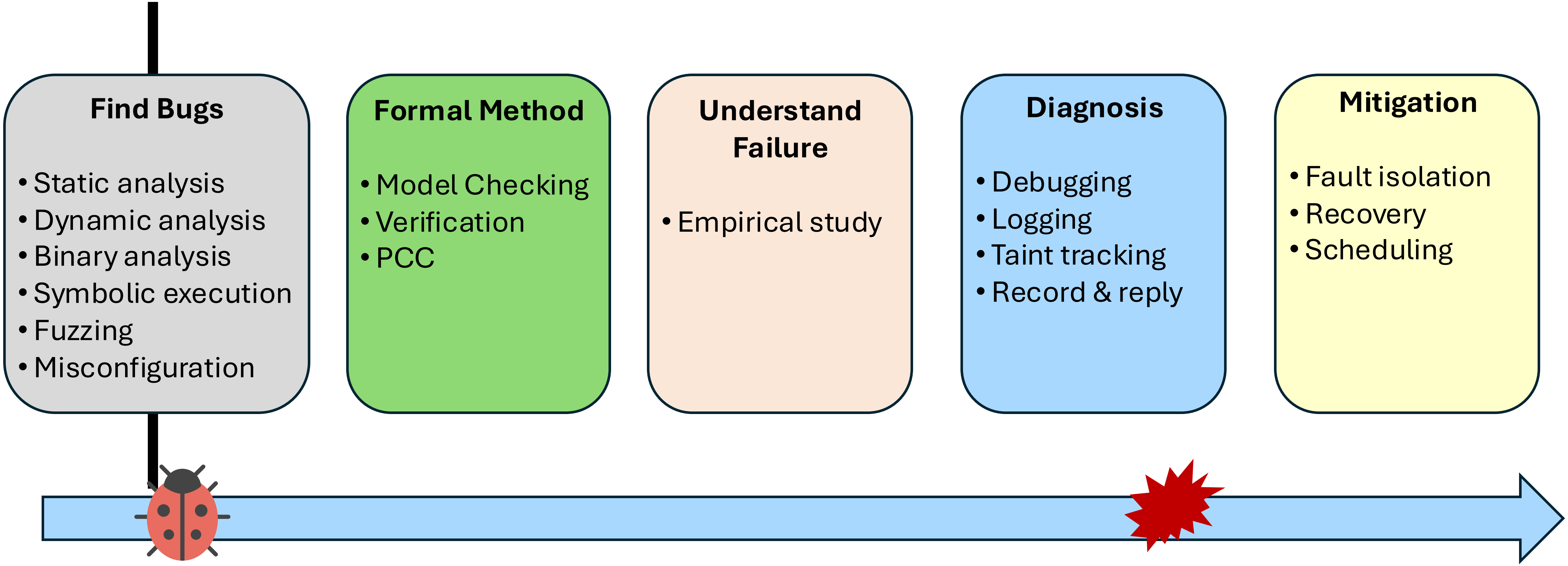
Software factors

- Bugs
- Configuration errors

Human factors

- Human errors (e.g., rm –rf /)

Topic in Computer System Reliability



Find Bugs

- Static analysis
- Dynamic analysis
- Binary analysis
- Symbolic execution
- Fuzzing
- Misconfiguration

Formal Method

- Model Checking
- Verification
- PCC

Understand Failure

- Empirical study

Diagnosis

- Debugging
- Logging
- Taint tracking
- Record & reply

Mitigation

- Fault isolation
- Recovery
- Scheduling

How to find a bug

Testing

- Generate random input, run and see if it triggers a bug

Static Analysis

- Analyze all possible behavior of a program without running it

What is a Static Analysis

A static analysis tool S analyzes the source code of a program P to determine whether it satisfies a property φ, such as:

- “P never deference a null pointer”
- “P does not leak file handles”
- “P does not divide by Zero”

Bad News

It is impossible to write such a tool!

- Why?
- **Rice Theorem:** Any nontrivial semantic property φ is undecidable, meaning it is impossible to construct a general automated method to determine whether P satisfies φ

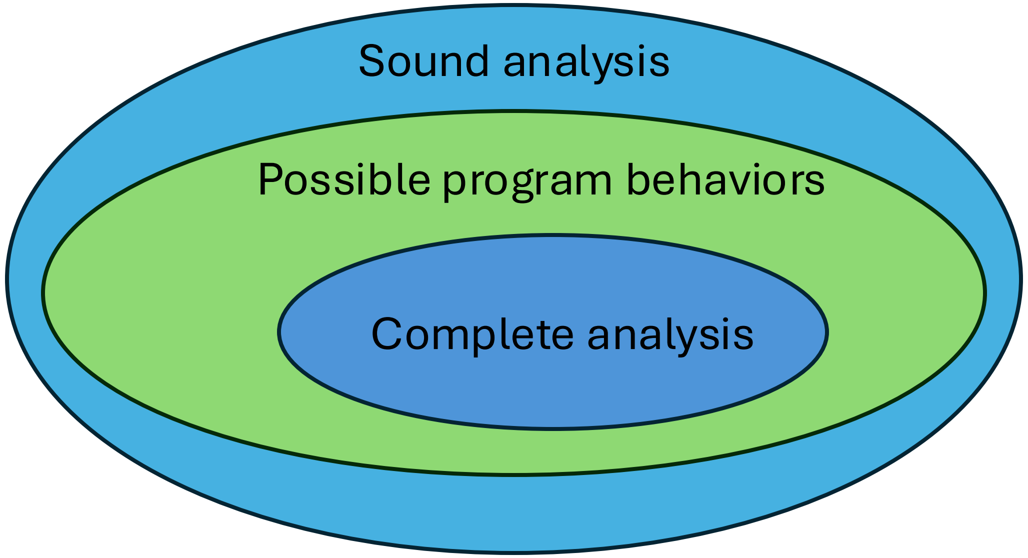
How to design a practical static analysis?

- Trade-off between soundness and completeness

Soundness VS Completeness

A practical static analysis tool analyzes whether a program satisfies a property ϕ, but can be wrong in one of two way:

- A sound static analysis never miss any violations but may **report false positives**
- A complete static analysis never report false positives, but it does not guarantee all violations will be reported (**false negative**)



Applications of Static Analysis

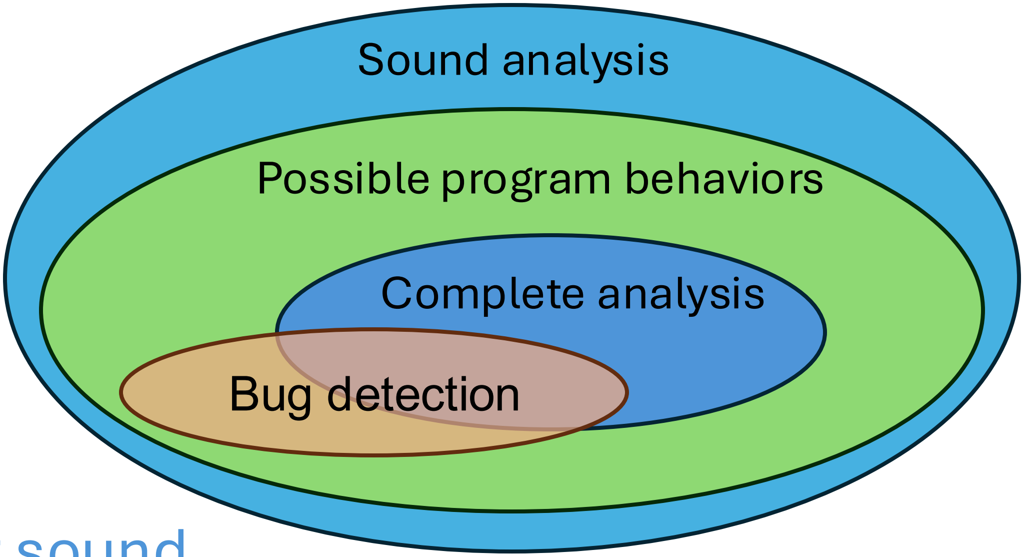
Compiler (sound)

- Type checking, liveness analysis …

Verification (sound)

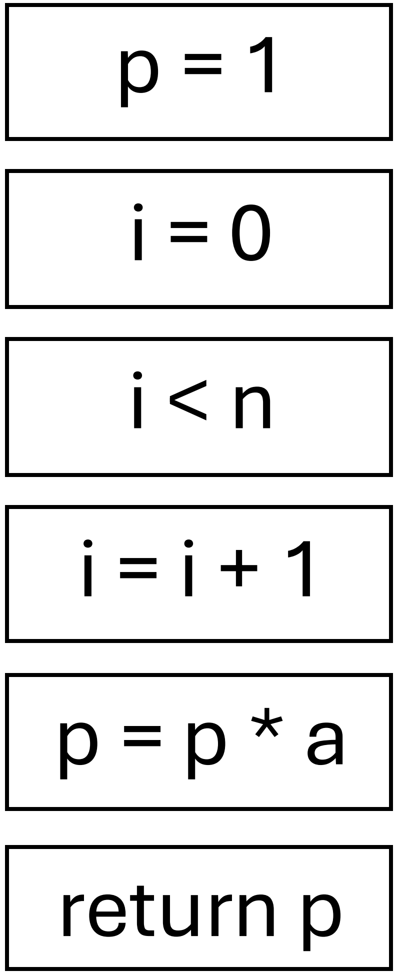
Bug finding?

- Usually complete
- But many bug detection tool for are neither sound nor complete



A Toy Static Analysis Example

```
int foo (int a, int n) {
    int p = 1;
    for (int i = 0; i < n; i++)
        p *= a;
    return p;
}
```



p = 1

i = 0

i < n

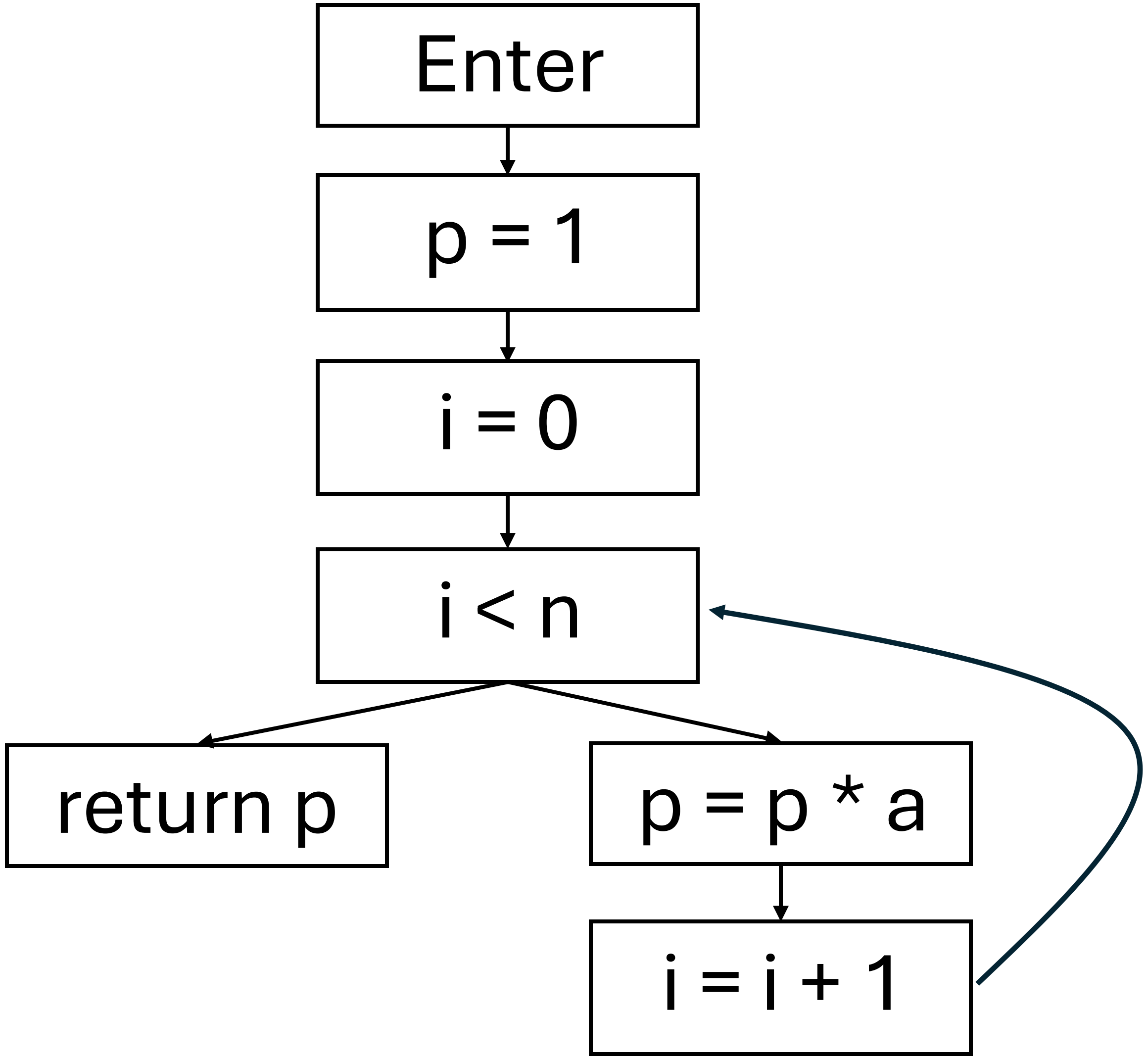
i = i + 1

p = p * a

return p

A Toy Example: Control-Flow Graph

```
int foo (int a, int n) {
    int p = 1;
    for (int i = 0; i < n; i++)
        p *= a;
    return p;
}
```



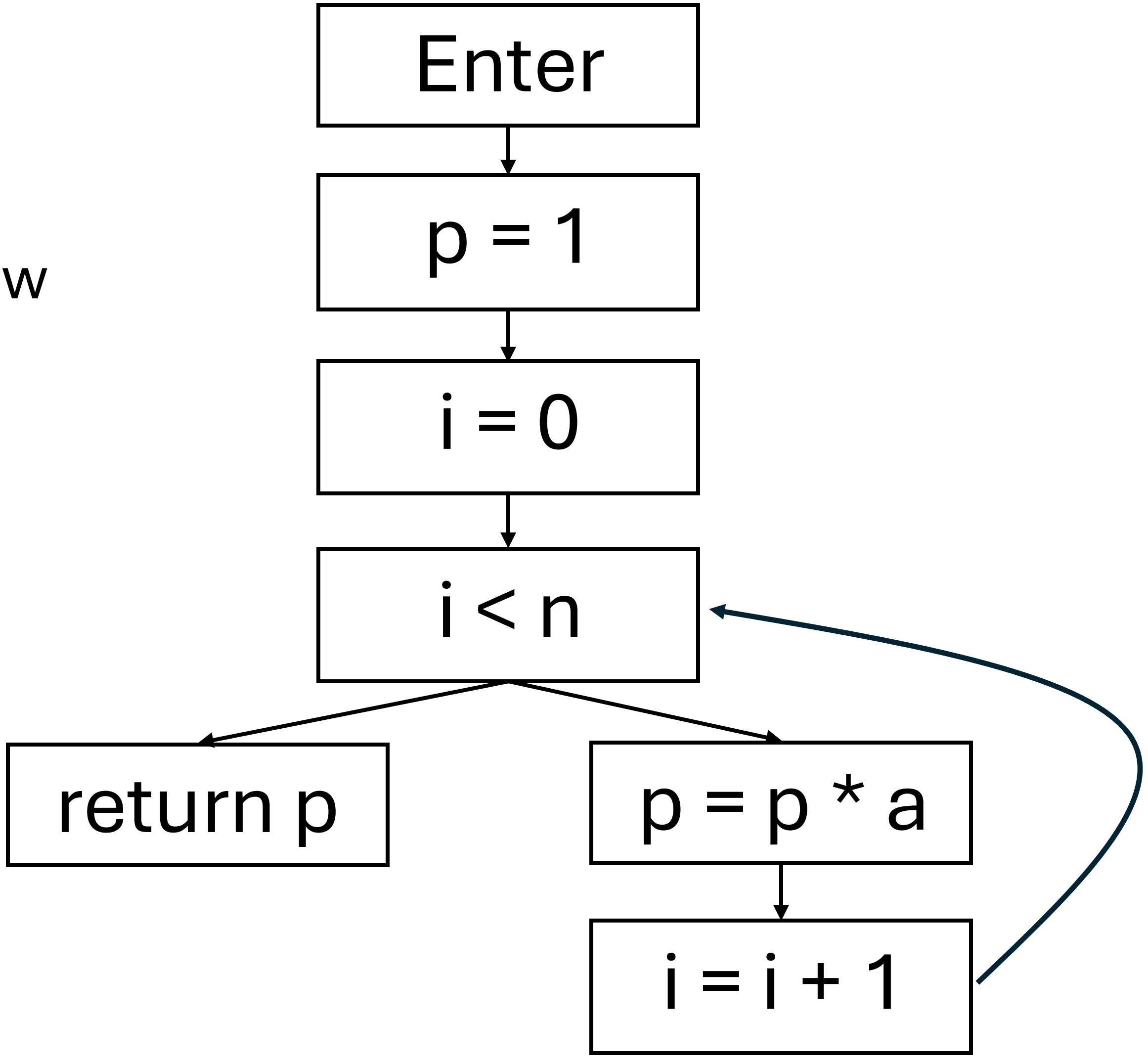
A Toy Example: Control-Flow Graph

Directed graph

- Each node is a statement
- Each edges represent possible control flow

Statement may be

- Assignments
- Branch
- Enter/return
- …



Why Not Static Analysis

- **Sound static analysis is great**
  - Can prove many important errors(leak, divided by zero)

- **But their false positive can be difficult to eliminate**
  - Static analysis can produce many false positives on large or unusual code bases
  - Unless you are an expert, telling a false positive from a real bug can be hard

Symbolic Execution

Goal: a bug finding technique that is easy to use!

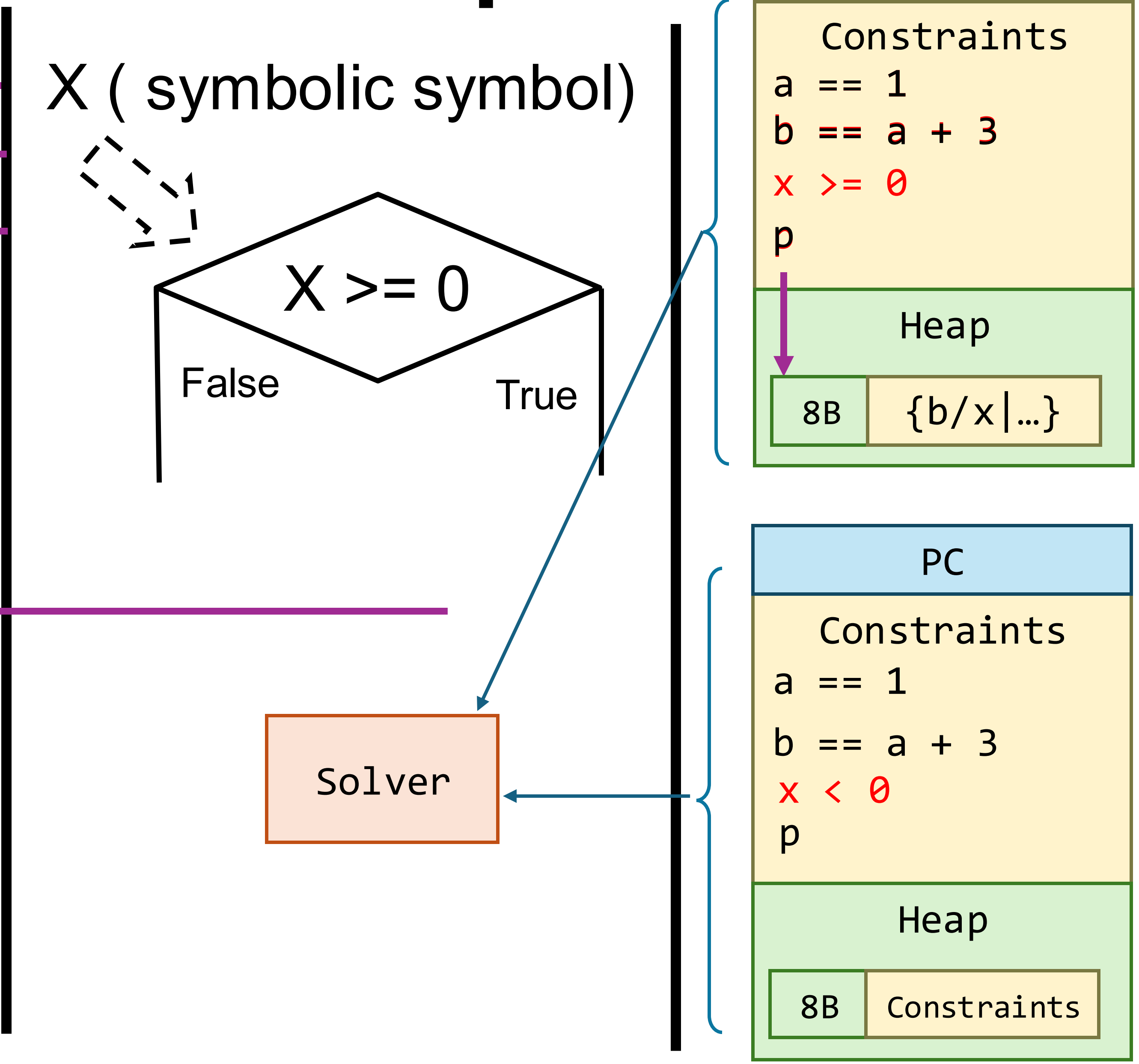
- No false positives
- Produces a concrete input on which the program will fail to meet the specification
- But can not prove the absence of errors

Key insight of Symbolic Execution

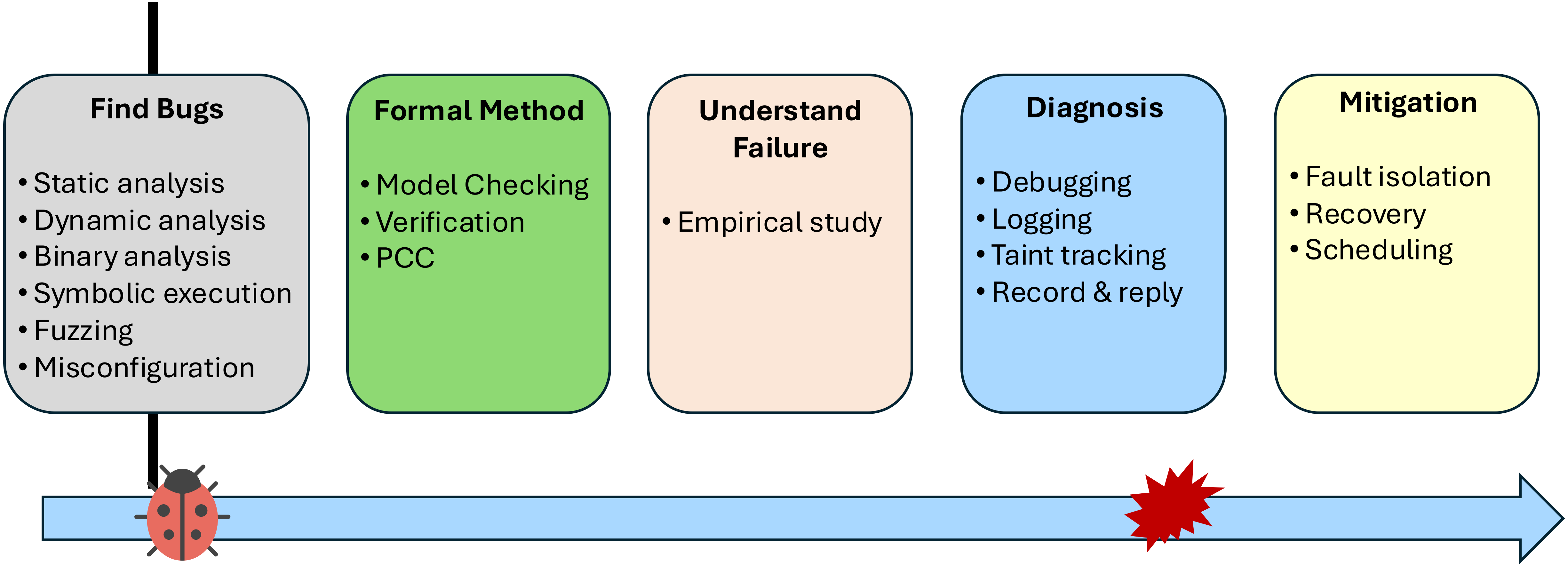
- Evaluate the program on symbolic input values
- Use an automated theorem prover to check whether there are corresponding concrete input values that make the program fail.

Symbolic Execution - an Example

```
int foo (int x) {
    int a = 1;
    int b = a + 3;
    int *p = malloc(sizeof(int));
    if (x >= 0) {
        p *= b / x;
    } else {
        FILE *fp = fopen(“a.txt”, “w”)
        fputc ('?’,fp);
}
```



Topic in Computer System Reliability



Find Bugs

- Static analysis
- Dynamic analysis
- Binary analysis
- Symbolic execution
- Fuzzing
- Misconfiguration

Formal Method

- Model Checking
- Verification
- PCC

Understand Failure

- Empirical study

Diagnosis

- Debugging
- Logging
- Taint tracking
- Record & reply

Mitigation

- Fault isolation
- Recovery
- Scheduling

Debugging

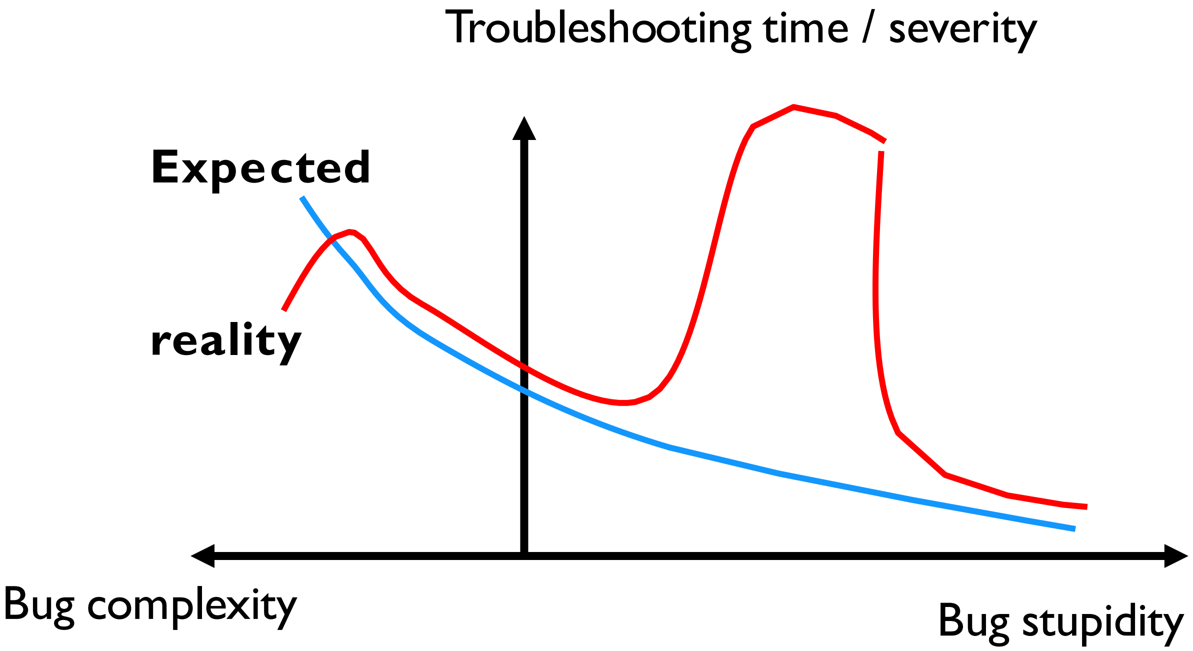
“Debugging is like being the detective in a crime movie…

…where you are also the murderer”

My advisors' Rule of Thumb on Debugging

All bugs are obvious, *after* you debug them

Some bugs are "stupid", but stupid bugs ≠ easy bugs



Huang' Rule of Thumb on Debugging

All bugs are obvious, *after* you debug them

Some bugs are "stupid", but stupid bugs ≠ easy bugs

- After some point, the more time you spend on troubleshooting an issue, the more stupid the bug turns out to be
  - one-off error, intvs. unsigned int, > vs. >=
- Example:
  - https://azure.microsoft.com/en-us/blog/summary-of-windows-azure-service-disruption-on-feb-29th-2012/

The more bugs you debug in a system, the deeper you understand about that system

- Also why companies' new engineer training task is often debugging

Debugging

Ad-hoc: printf**, systematic tool:** gdb

- examine program state, e.g., if a branch is taken, value of a variable
- compare the state with expected behavior
- if it deviates from the expected, how does it become like this

Challenge 1: debugger may not be available

- e.g., distributed system

Challenge 2: hard to reproduce an issue in production

- e.g., no core dump generated

Challenge 3: root cause is far away from the failure site

- e.g., why is this pointer becoming a null pointer?

Logging: Source of Clues in Debugging

Logging is an instrumental aid for debugging

- Often the only clues left in the crime scene (production environment)

That's why the quality of logs is important

- Trade-offs among information, overhead, importance
- *Log20: Fully Automated Optimal Placement of Log Printing Statements under Specified Overhead Threshold [SOSP '17]*
- *Be Conservative: Enhancing Failure Diagnosis with Proactive Logging* [OSDI '12]
- *Improving Software Diagnosability via Log Enhancement* [ASPLOS '11]

Deducing information from logs is an art

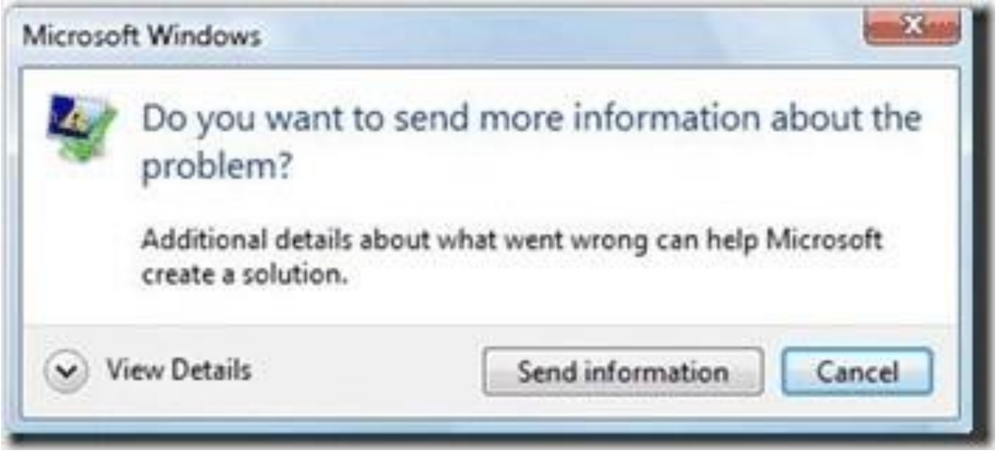
- "The Science of Deduction"
- SherLog: Error Diagnosis by Connecting Clues from Run-time Logs [ASPLOS '10]

Debugging In the Large

How would Microsoft developers debug a Windows problem?

- OS is already deployed to customer computer
- Debug symbols not enabled at customer site
- Hard to convince customer to run a debugger

Windows Error Reporting (WER) [Paper]



Microsoft Windows

Do you want to send more information about the problem?

Additional details about what went wrong can help Microsoft create a solution.

Other Interesting Topics

Bug fixing

- Bug fixes can become bug again
  - The fixes are only workaround or the other parts of software changes

System verification

- Passing testing and static analysis tools does not mean the software is bug-free
- How can we *prove* that a software is correct *under all circumstances*

Configuration errors

- Not just code bug or hardware issue, human error!

Failure detection

- Production software does not always simply crash, often exhibit gray failure

Failure isolation

Fault tolerance

If You Are Interested In Knowing More…

EC 528 Cloud Computing

https://yigonghu.github.io/#publications

We can talk!

Next Time…

Midterm 2 review