QA: Dynamic Analysis

17-313 Fall 2025

Foundations of Software Engineering

https://cmu-17313q.github.io

Eduardo Feo Flushing

Administrivia

- Teamwork:
  - *“Every member of the team must contribute to the implementation”.*
- Evidence of contribution
  - GitHub commits/PRs
- Individual grade penalty may apply in case of insufficient contribution
- Contact course staff to request regrades

Learning Goals

- Describe random test-input generation strategies such as fuzz testing
- Identify and discuss the key challenges associated with performance testing in software development.
- Understand the ideas behind chaos engineering and how it is used to test resiliency of cloud-based applications
- Describe A/B testing for usability
- Recommend appropriate dynamic analysis techniques for specific software quality issues.

Automated Analysis for Functional and Non-Functional Properties

- Static Analysis and Testing
  - Correctness, Maintainability, Security, …
  - Pattern-based checks

Automated Analysis for Functional and Non-Functional Properties

- Static Analysis and Testing
  - Correctness, Maintainability, Security, …
  - Pattern-based checks
- **Dynamic Analysis**
  - **Robustness – Fuzzing**
  - **Performance – Profiling**
  - **Scalability – Stress testing**
  - **Resilience – Soak testing**
  - **Reliability – Chaos Engineering**
  - **Usability – A/B testing**

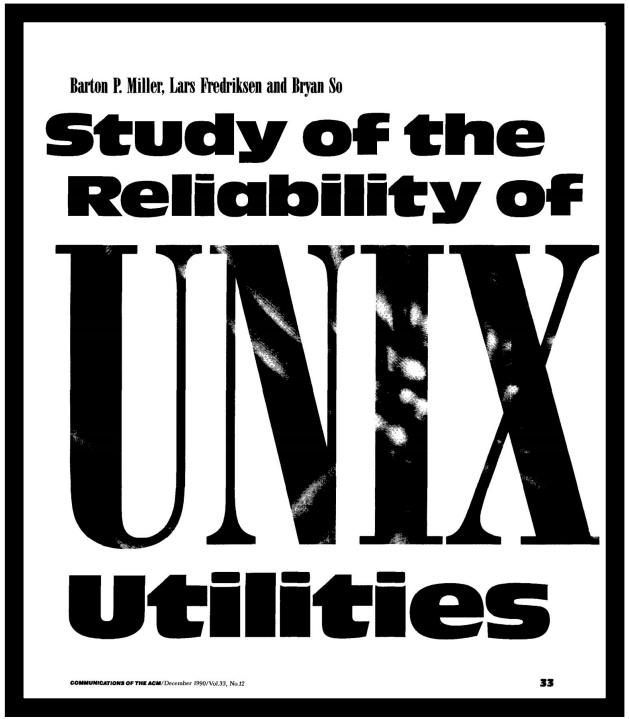
Outline

- Fuzz Testing
- Performance Testing and Debugging
- Testing in Production
  - Reliability: Chaos Engineering
  - GUI and Usability: A/B Testing

Disclaimer

Many (unintentional) references to animals ahead

Security and Robustness



Communications of the ACM (1990)

> On a dark and stormy night one of the authors was logged on to his workstation on a dial-up line from home and the rain had affected the phone lines; there were frequent spurious characters on the line. The author had to race to see if he could type a sensible sequence of characters before the noise scrambled the command. This line noise was not surprising; but we were surprised that these spurious characters were causing programs to crash.

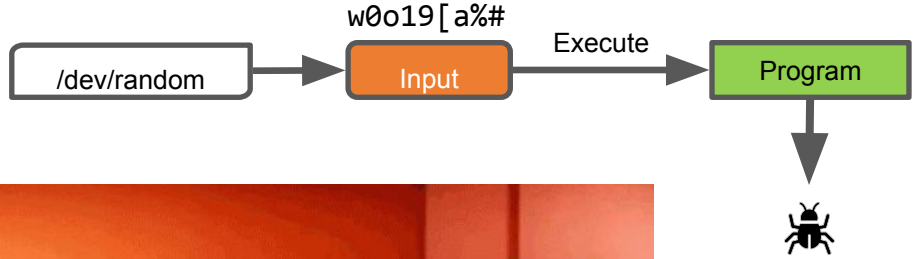
How to identify these bugs?

Infinite monkey theorem

*"a **monkey** hitting keys **at random** on a typewriter **keyboard** for an **infinite amount of time** will almost surely type any given text, including the complete works of **William Shakespeare.** "*

https://en.wikipedia.org/wiki/Infinite_monkey_theorem

Fuzz Testing



A 1990 study found crashes in:
adb, as, bc, cb, col, diction, emacs, eqn, ftp, indent, lex, look, m4, make, nroff, plot, prolog, ptx, refer!, spell, style, tsort, uniq, vgrind, vi

Cicada

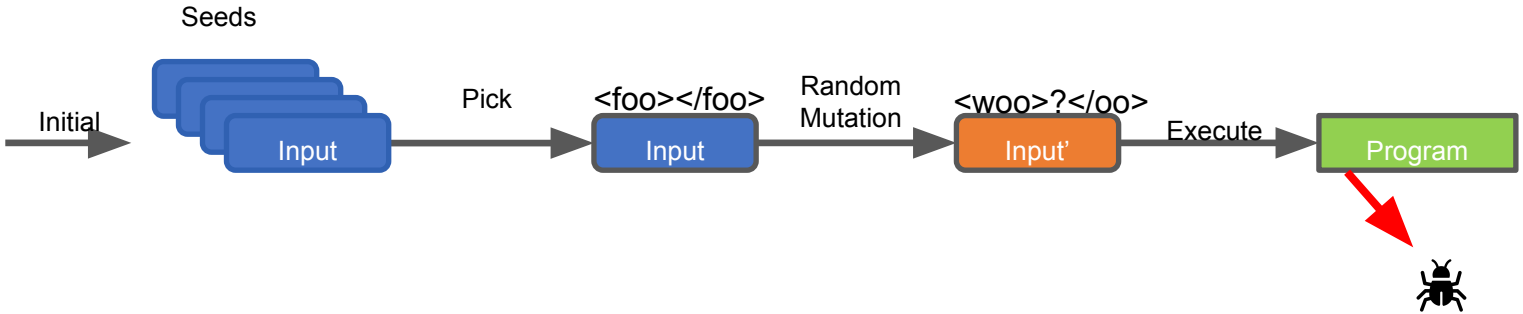
Common Fuzzer-Found Bugs in C/C++

Causes: incorrect arg validation, incorrect type casting, executing untrusted code, etc.

Effects: buffer-overflows, memory leak, division-by-zero, use-after-free, assertion violation, etc. (“crash”)

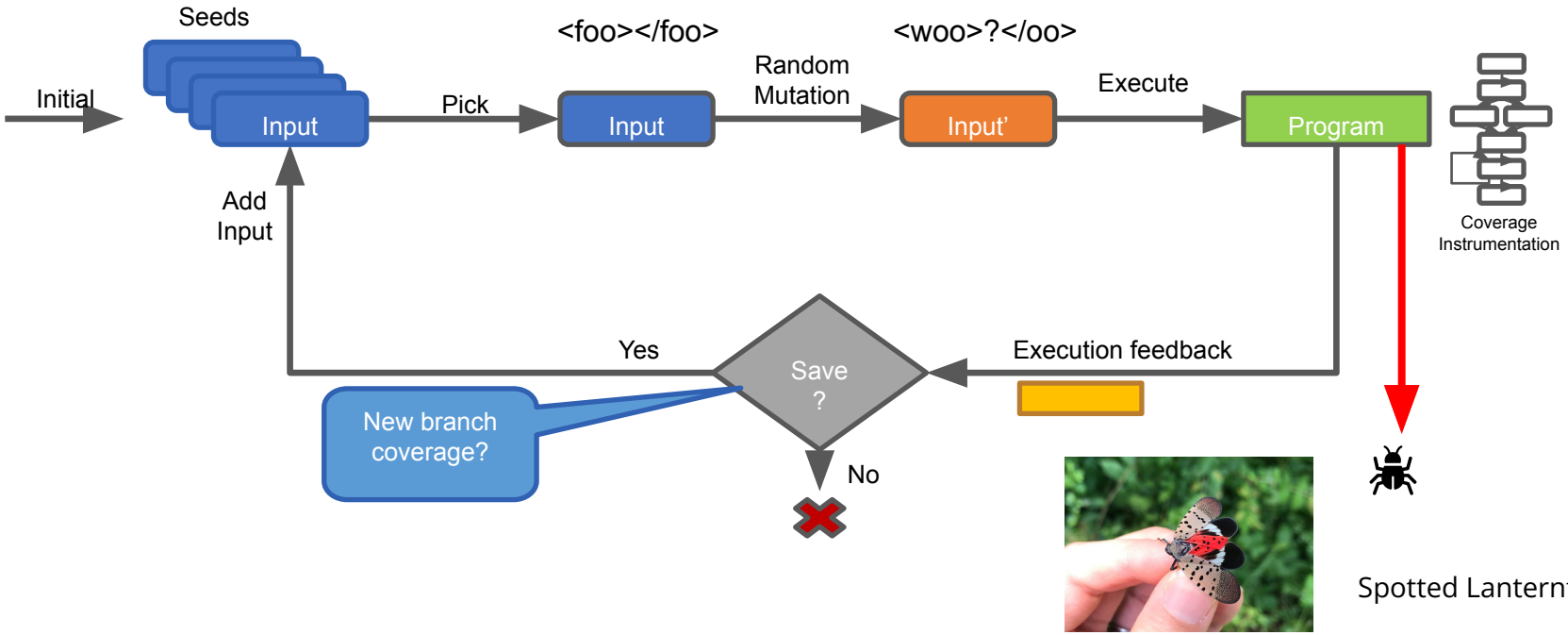
Impact: security, reliability, performance, correctness

Mutation-Based Fuzzing (e.g. Radamsa)



Hercules Beetle

Coverage-Guided Fuzzing (e.g. AFL)

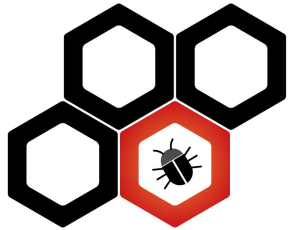


Mutation Heuristics

- Binary input
  - Bit flips, byte flips
  - Change random bytes
  - Insert random byte chunks
  - Delete random byte chunks
  - Set randomly chosen byte chunks to *interesting* values e.g. INT_MAX, INT_MIN, 0, 1, -1, …
- Text input
  - Insert random symbols relevant to format (e.g. “<“ and “>” for xml)
  - Insert keywords from a dictionary (e.g. “<project>” for Maven POM.xml)
- GUI input
  - Change targets of clicks
  - Change type of clicks
  - Select different buttons
  - Change text to be entered in forms
  - … Much harder to design

Fuzzing in practice

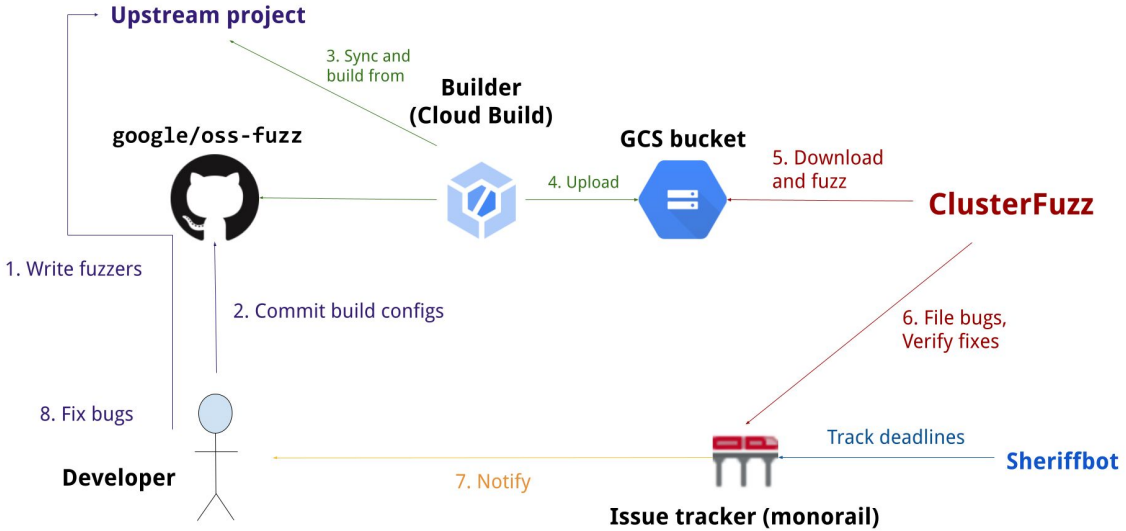
- Google uses ClusterFuzz to fuzz all Google products
- Supports multiple fuzzing strategies
- *“As of February 2023, ClusterFuzz has found ~27,000 bugs in Google (e.g. Chrome).”*



Fuzzing in practice

- After the OpenSSL Heartbleed vulnerability discovered in 2016, Google launched OSS-Fuzz
- Free service for open source projects
  - *"The project must have a significant user base and/or be critical to the global IT infrastructure."*
- OSS-Fuzz privately alerts developers to the bugs detected.
- Supports CI (e.g., triggered from GitHub actions)

OSS-Fuzz: Free Fuzzing for Open Source Software



"As of August 2023, OSS-Fuzz has helped identify and fix over 10,000 vulnerabilities and 36,000 bugs across 1,000 projects."

Some projects include: nodejs, django, mysql-server, redis-py, apache-httpd, openvpn, openssl

Activity:

Pick one scenario based on where you are seating

- E-Commerce Web Application (front rows)
- Automotive Software for Self-Driving Cars (middle rows)
- Mobile Gaming Application (back rows)

Discuss in groups of 2-3 the applicability of fuzz testing in your scenario, considering:

- One type of input to fuzz.
- One potential vulnerability or bug fuzz testing might uncover.
- One specific challenge in implementing fuzz testing for the scenario.

Bonus: How fuzz testing could be integrated into the development cycle for that particular application?

Performance Testing and Debugging

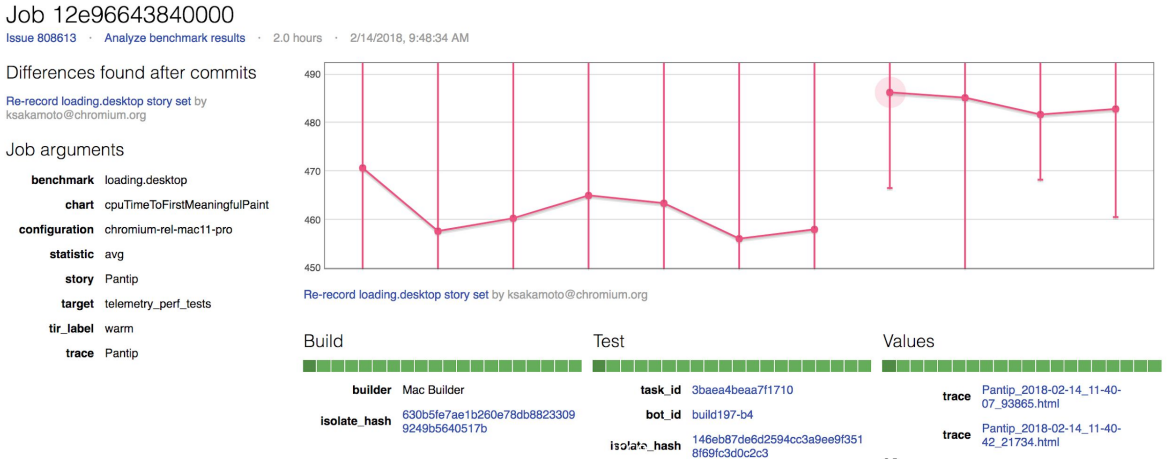
Performance Testing

- Goal: Identify *performance bugs*. What are these?
  - Unexpected bad performance on some subset of inputs
  - Performance degradation over time
  - Difference in performance across versions or platforms

- Not as easy as functional testing. What's the baseline?
  - Fast = good, slow = bad // but what's the threshold?
  - How to get reliable measurements?
  - How to debug where the issue lies?

Performance regression testing helps identify trends

- Measure execution time of critical components
- Log execution times and compare over time



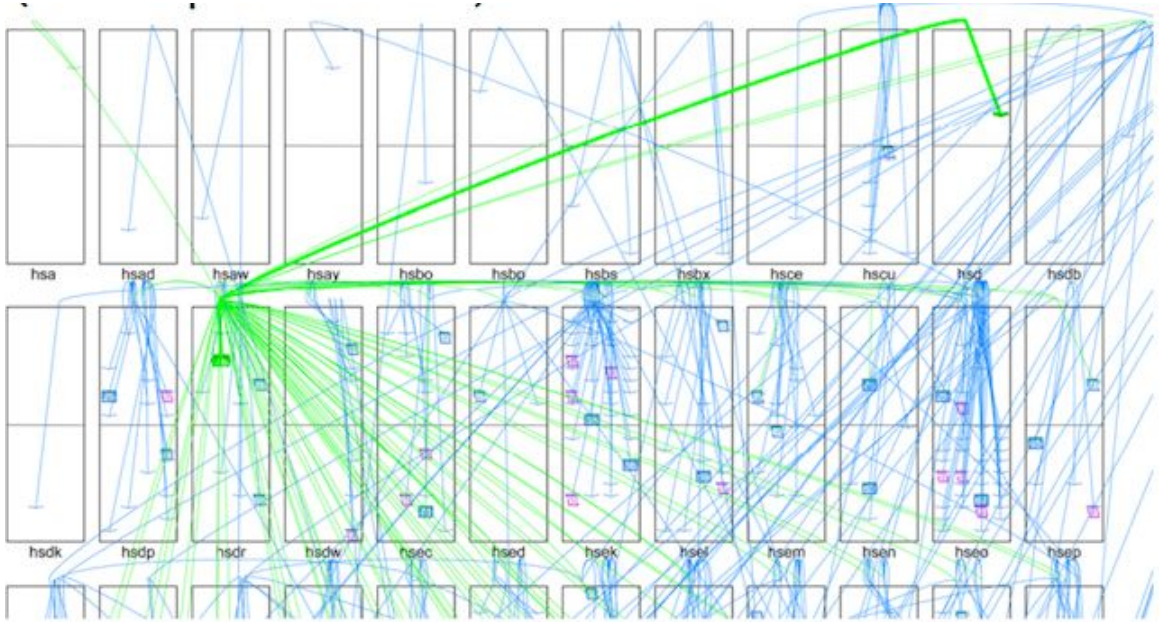
Performance bugs are “bad” bugs

Discovering, Reporting, and Fixing Performance Bugs

Adrian Nistor[1], Tian Jiang[2], and Lin Tan[2]
[1]University of Illinois at Urbana-Champaign, [2]University of Waterloo
nistor1@illinois.edu, {t2jiang, lintan}@uwaterloo.ca

- Fixing performance bugs is usually more difficult than fixing non-performance bugs
- Performance bugs usually don’t generate incorrect results or crashes
- Difficult to diagnose:
  - system load, hardware configuration, network conditions, user-specific workflows, interactions with other systems
- Big impact on user experience

A search query in Google Data Centers



Dick Sites - "Data Center Computers: Modern Challenges in CPU Design"

Catching performance bugs

- Observation in natural environment
- Real-time data collection
- Understanding ecosystem impact
- Non-invasive techniques
- Behavioral pattern analysis

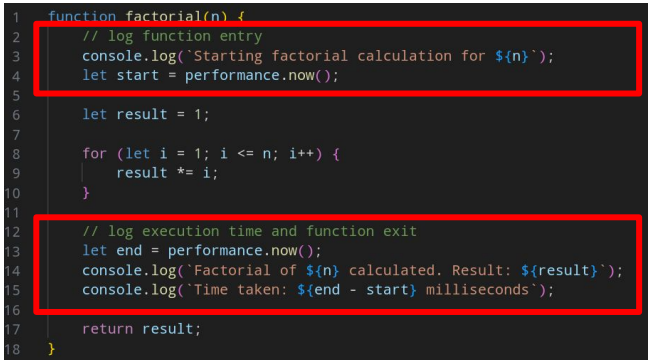
- Observation in real-time operation
- Monitoring system performance & resources
- Analyzing software's interaction with its surroundings
- Minimal impact on running application
- Detecting anomalies and performance issues

Profiling and tracing

- Profiling is a process to analyze and measure the performance of a program or specific parts of its code (e.g., functions).
- Tracing is about understanding the flow of execution and the behavior of a program.
  - Record sequential events (function calls) that occur during the execution of a program
- Both can be used to identify bottlenecks in execution time and memory

Performance analysis via instrumentation

- Embedding additional code to monitor the program's behavior
- Usage:
  - Source Code (**Static**): Additional instructions for data collection.
  - Binary Files (**Dynamic**): Inserting monitoring code at runtime without altering the source.
- Applications:
  - **Profiling**: Execution time, function call frequency, and resource usage.
  - **Tracing**: Record detailed execution flow, tracking function entries/exits and event sequences.



Sampling stack traces

```
def f1(): return f2()

def f2(): return f3()

def f3():
    stack_trace = traceback.extract_stack()
    printStack(stack_trace)
    return f4()

def f4(): return f5()

def f5(): return f6()

def f6(): return f7()

def f7(): return f8()

def f8(): return f9()

def f9(): return f10()

def f10():
    return

f1()
```

```
  File "test_python_traceback.py", line 34, in <module>
    f1()
  File "test_python_traceback.py", line 10, in f1
    def f1(): return f2()
  File "test_python_traceback.py", line 12, in f2
    def f2(): return f3()
  File "test_python_traceback.py", line 15, in f3
    stack_trace = traceback.extract_stack()
```

Flame Graphs

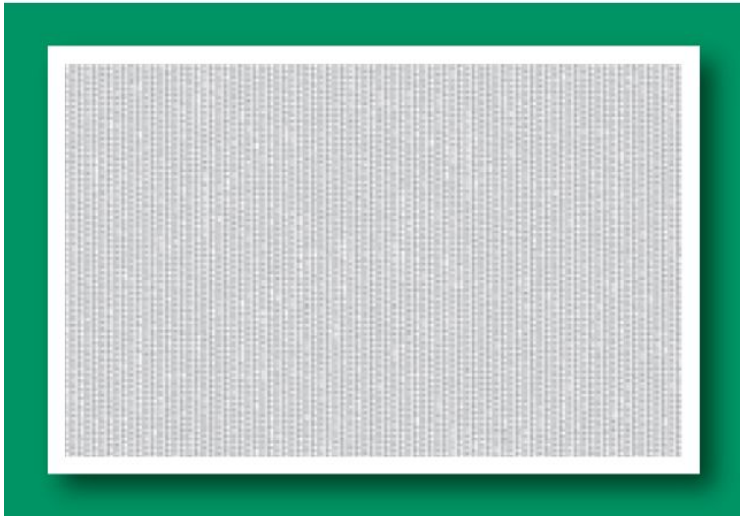
The Flame Graph

THIS VISUALIZATION OF SOFTWARE EXECUTION IS A NEW NECESSITY FOR PERFORMANCE

BRENDAN GREGG, NETFLIX

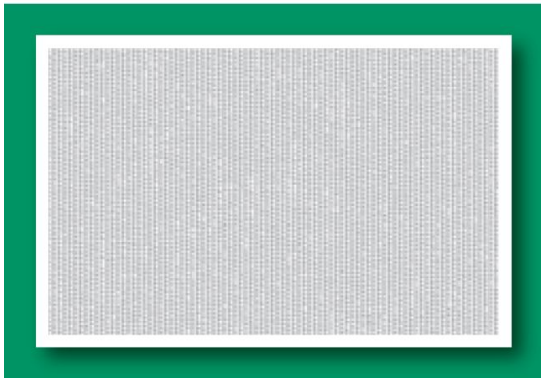
An everyday problem in our industry is understanding how software is consuming resources, particularly CPUs. What exactly is consuming how much, and how did this change since the last software version?

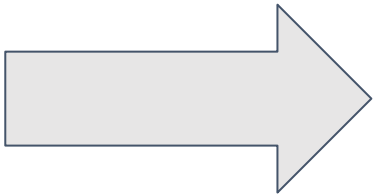
FIGURE 3: **FULL MYSQL DTRACE PROFILE OUTPUT**

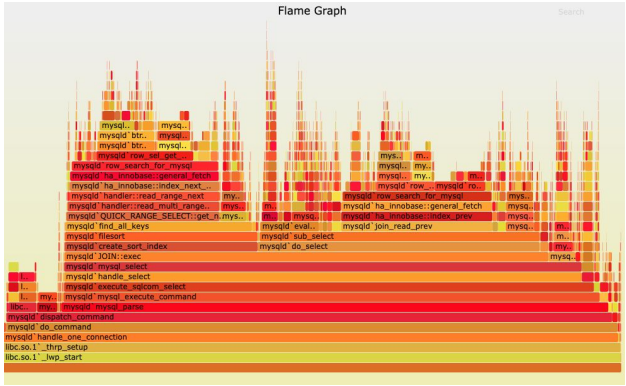


Flame Graphs

FIGURE 3: **FULL MYSQL DTRACE PROFILE OUTPUT**

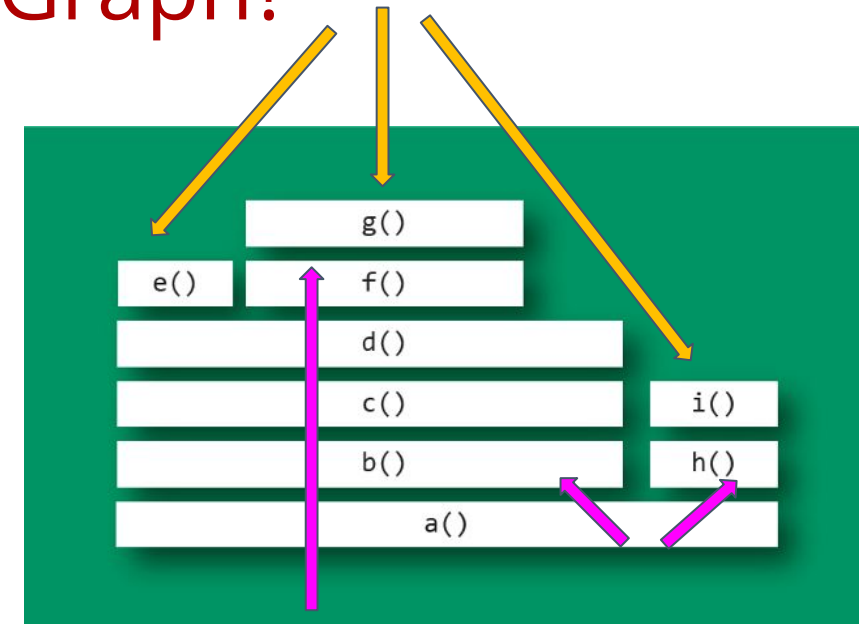






How to read a Flame Graph?

- Top edges of the flame graph show the functions that were running on when the stack trace was collected
- Top down shows ancestry
- Box width proportional to presence in stack traces



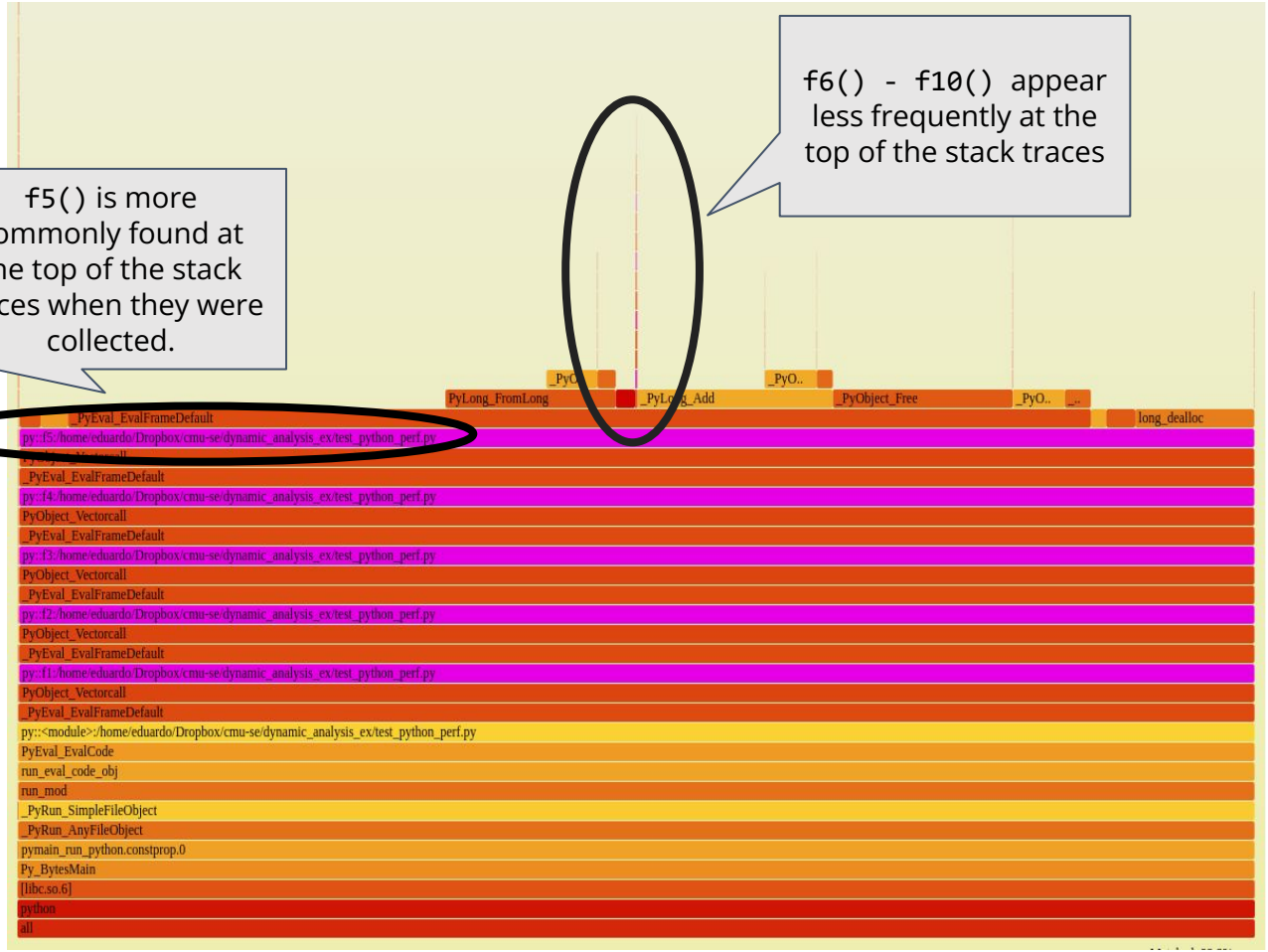
Q. What does the flame graph for this code look like?

```
def f1(): return f2()

def f2(): return f3()

def f3(): return f4()

def f4(): return f5()

def f5():
    uselessSum=0
    for i in range(10000):
        uselessSum += 1
    return f6() + uselessSum

def f6(): return f7()

def f7(): return f8()

def f8(): return f9()

def f9(): return f10()

def f10():
    return 42

while True:
    f1()
```

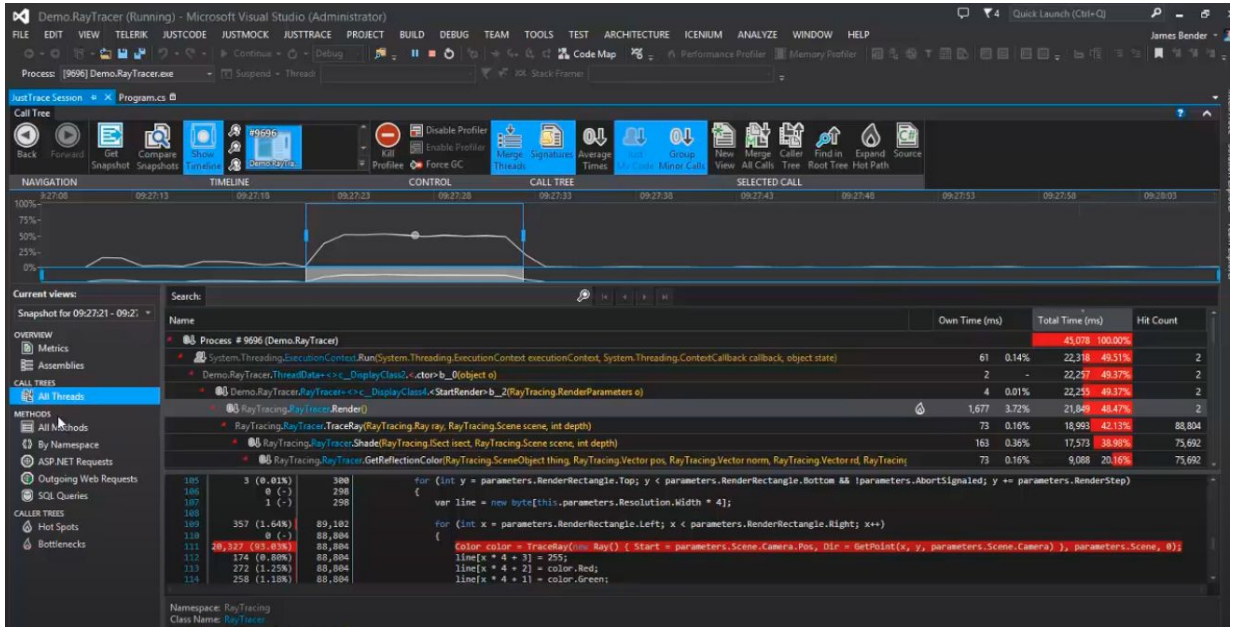
```python
def f1(): return f2()

def f2(): return f3()

def f3(): return f4()

def f4(): return f5()

def f5():
    uselessSum=0
    for i in range(10000):
        uselessSum += 1
    return f6() + uselessSum

def f6(): return f7()

def f7(): return f8()

def f8(): return f9()

def f9(): return f10()

def f10():
    return 42

while True:
    f1()
```

`f5()` is more commonly found at the top of the stack traces when they were collected.

`f6()` - `f10()` appear less frequently at the top of the stack traces



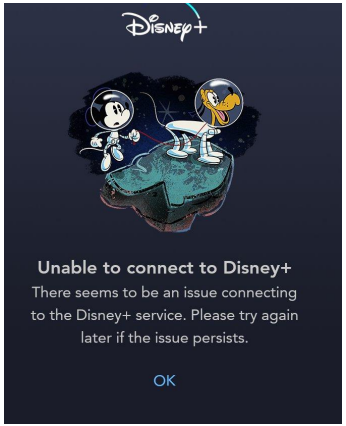
Profilers often included in IDEs



Stress testing

- Scalability/Robustness testing technique: test beyond the limits of normal operation.
- Can apply at any level of system granularity.
- Key idea: throw large amounts of input / requests and see how the program behaves
- Often a way to test the error-handling capabilities of the application

Real Issues: Disney+ Launch



- Lots of issues reported on launch day.
- Disney had planned for a spike in traffic.
  - Tested massive concurrent video streaming capability.
- BUT: the stress was in paths other than streaming
  - User account creation
  - Logins and auth
  - Browsing old titles

Soak testing

- A system may behave exactly as expected under artificially limited execution conditions, but fail in production after extended use.
  - E.g., Memory leaks may take longer to lead to failure
- **Soak testing** a system involves applying a significant load over a significant period of time and observing system resilience.
- Time-consuming to run but useful to apply at big release milestones or when making infrastructure changes.

Activity:

Pick one scenario based on where you are seating

- E-Commerce Web Application (**front rows**)
- Automotive Software for Self-Driving Cars (**middle rows**)
- Mobile Gaming Application (**back rows**)

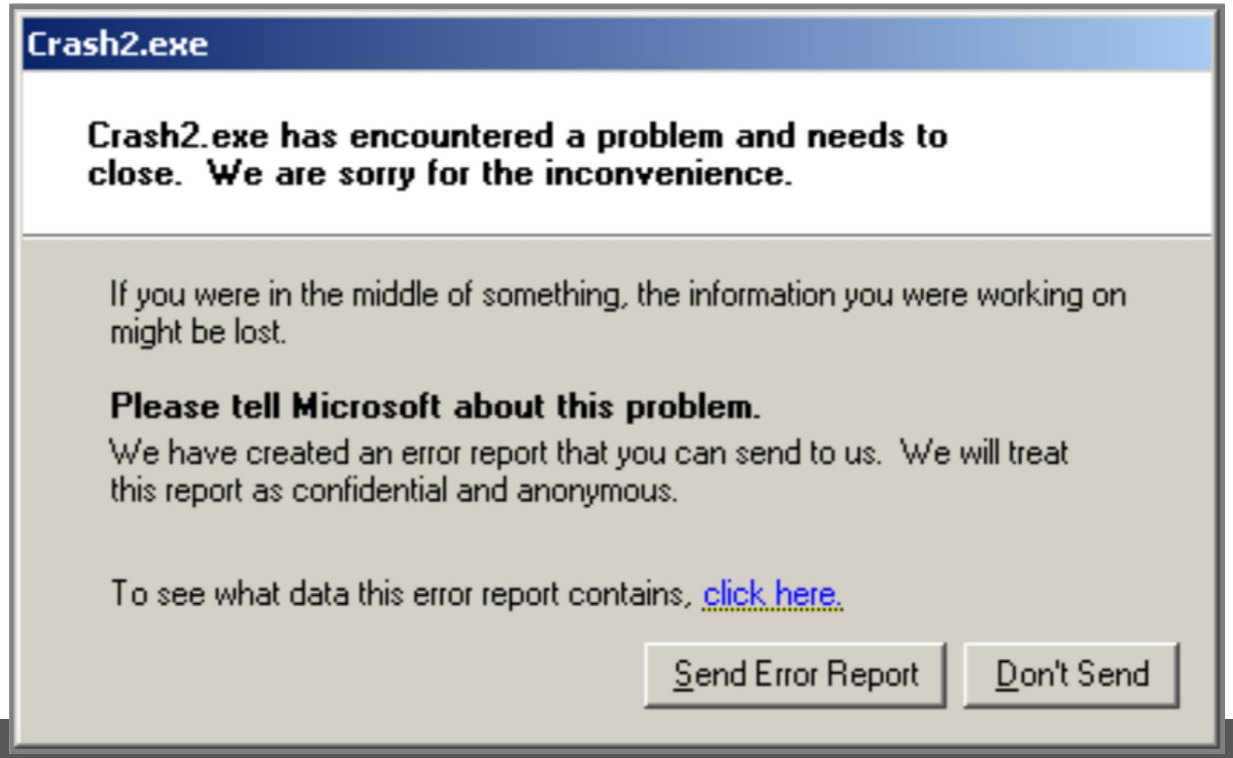
Discuss in groups of 2-3:

- Enumerate specific performance challenges in the your scenario.
- Pick one dynamic analysis technique to address some of these challenges.

Testing in Production

Beta testing

Telemetry


Crash2.exe
Crash2.exe has encountered a problem and needs to close. We are sorry for the inconvenience.
If you were in the middle of something, the information you were working on might be lost.
Please tell Microsoft about this problem.
We have created an error report that you can send to us. We will treat this report as confidential and anonymous.
To see what data this error report contains, click here.
Send Error Report
Don't Send

Reliability testing

- What happens when some components of a large complex system fail? Can the system recover and keep working?
- How can you test the reliability of something as complex as Netflix or Google maps or Instagram?
- One idea: simulate a large-scale deployment and induce random failures in various components
- Another idea… Test in Production with **Chaos Engineering**

What is chaos engineering?

"Chaos Engineering is the discipline of experimenting on a system in order to build confidence in the system's capability to withstand turbulent conditions in production.“

principlesofchaos.org

Chaos Engineering: Testing in Production

- Purposefully take down components in a **live deployment**.
- Observe system response. Do failovers work correctly?
- Tests the failure-handling and fallback capabilities of large systems.
- Useful in preparing for natural disasters or cyberattacks.

Example: Google

Terminate network in Sao Paulo for testing:

- Hidden dependency takes down links in Mexico which would have remained undiscovered without testing

Turn off data center to find that machines won't come back:

- Ran out of DHCP leases (for IP address allocation) when a large number of machines come back online unexpectedly.

Why would you break things on purpose?

Firefighters train in realistic, high-stress simulations. Sometimes so real they aren't even told it's a drill.

Failures in Microservice Architectures

Network may be partitioned

Server instance may be down

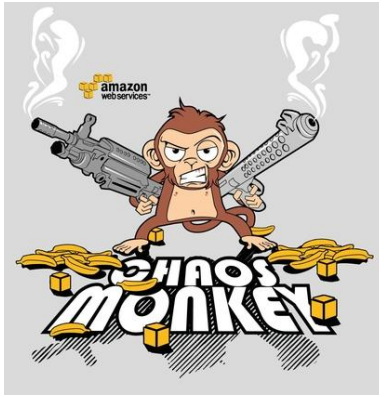
Communication between services may be delayed

Server could be overloaded and responses delayed

Server could run out of memory or CPU

Example: Netflix

Significant deployment on AWS cloud. Hundreds of updates to microservices and infrastructure through the day.



Chaos Monkey randomly takes down AWS instances or network connections or randomly changes config files.

How to tell ”are we still good?”
Key metric: Stream Starts per Second (SPS)
Measures *availability*

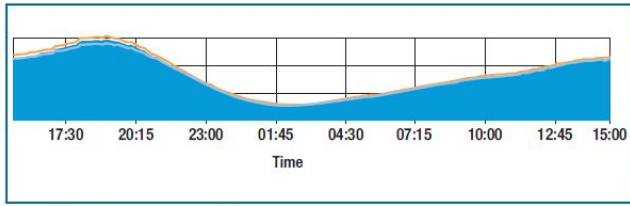


FIGURE 2. A graph of SPS ([stream] starts per second) over a 24-hour period. This metric varies slowly and predictably throughout a day. The orange line shows the trend for the prior week. The *y*-axis isn't labeled because the data is proprietary.

Testing GUIs and Usability

Automating GUI/Web Testing

- This is hard
- Capture and Replay Strategy
  - mouse actions
  - system events
- Test Scripts: (click on button labeled "Start" expect value X in field Y)
- Lots of tools and frameworks
  - e.g. Selenium for browsers
- Can avoid load on GUI testing by separating model from GUI
- Beyond functional correctness?

Usability: A/B testing

- Controlled randomized experiment with two variants, A and B, which are the control and treatment.
- One group of users given A (current system); another random group presented with B; outcomes compared.
- Often used in web or GUI-based applications, especially to test advertising or GUI element placement or design decisions.

Example

- A company sends an advertising email to its customer database, varying the photograph used in the ad…

Example: group A (99% of users)

Act now!

Sale ends soon!

Example: group B (1%)

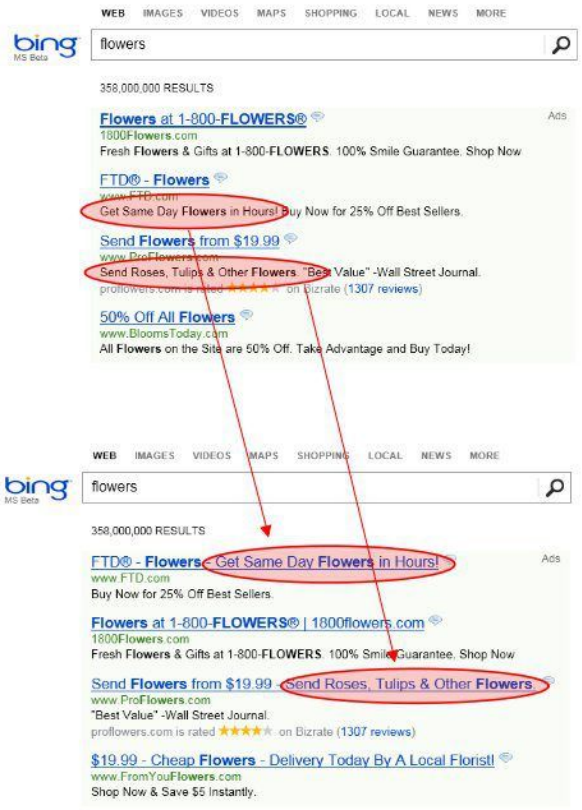
My name is "Jack"

Act now!

Sale ends soon!

Bing Experiment

- Experiment: Ad Display at Bing
- Suggestion prioritized low
- Not implemented for 6 months
- Ran A/B test in production
- Within 2h revenue-too-high alarm triggered suggesting serious bug (e.g., double billing)
- Revenue increase by 12% - $100M annually in US
- Did not hurt user-experience metrics



Kohavi, Ron, Diane Tang, and Ya Xu. "Trustworthy Online Controlled Experiments: A Practical Guide to A/B Testing." 2020.

A/B Testing

- Requires monitoring tools and telemetry
- Requires good metrics and statistical tools to identify significant differences.
  - E.g. clicks, purchases, video plays
- Must control for confounding factors
- Automation:
  - Stop experiments when confident in results
  - Stop experiments resulting in bad outcomes (crashes, very low sales)
  - Automated reporting, dashboards

Learning Goals

- Describe random test-input generation strategies such as fuzz testing
- Identify and discuss the key challenges associated with performance testing in software development.
- Understand the ideas behind chaos engineering and how it is used to test resiliency of cloud-based applications
- Describe A/B testing for usability
- Recommend appropriate dynamic analysis techniques for specific software quality issues.