Fault Injection

Lecture 12: CPEN 400P

Karthik Pattabiraman, UBC

(Some Slides based on Wes Weimer's course at U. Mich. and the Netflix Technical Blog - https://netflixtechblog.com/)

Outline

Resilience Engineering

Fault Injection

Software Fault Injection

LLFI: LLVM-based Fault Injector

Fault Injection in Cloud Applications

Resilience: Definition

Property of a system to “bounce back” after a failure or disruption

- Treats failures as common occurrence rather than exceptions
- Need to have (sufficient) redundancy to recover from failure
- Closely related to the notion of availability
- Availability - Fraction of time the system is up for service

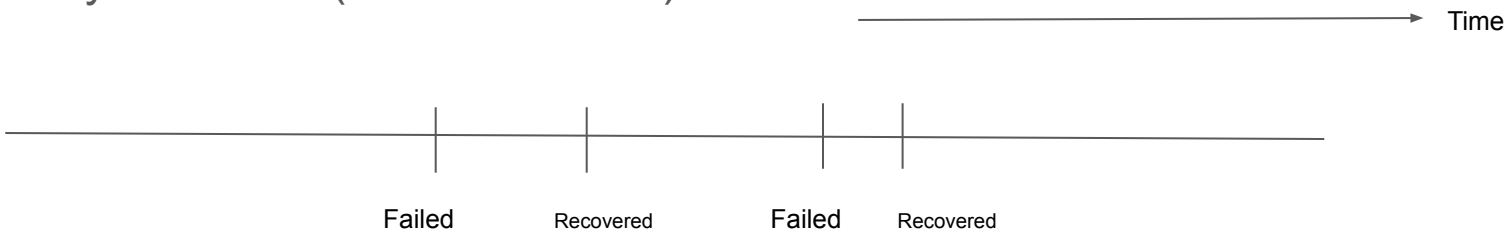
Availability

Readiness of system for service - probability that the system is up for service

MTTF: Mean Time to Failure

MTTR: Mean Time to Recovery

Availability = MTTF / (MTTF + MTTR)



Resilience

Resilience engineering attempts to maximize availability by **decreasing** the MTTR

Bug-finding approaches via static and dynamic analysis attempt to reduce MTTF

Need a systematic way to evaluate the resilience of the system

- Many bugs are often found in error-detection and recovery code [IBM study]
- Hidden dependencies in the system may prevent (Fast) recovery
- Eliminate need for manual problem identification and fixing

Outline

Resilience Engineering

Fault Injection

Software Fault Injection

LLFI: LLVM-based Fault Injector

Fault Injection in Cloud Applications

Fault Injection

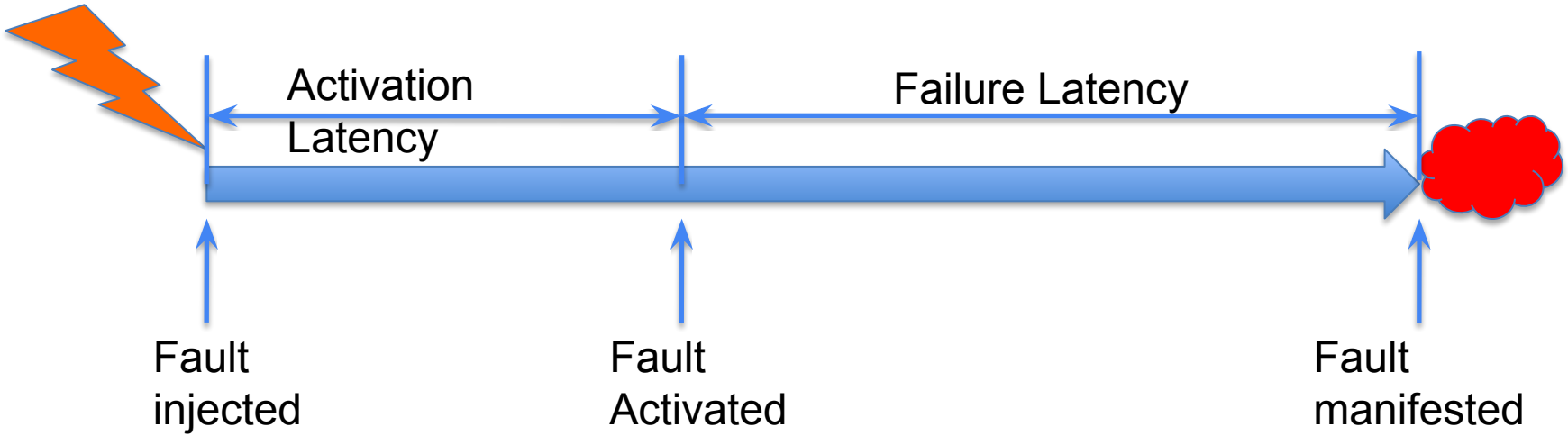
- Fault-injection (or fault-insertion) is the act of deliberately introducing faults into the system in a controlled and scientific manner, in order to study the system's response to the fault
  - Can be used to estimate resilience (e.g., detection, recovery)
  - Also used to understand inherent fault tolerance of the system
  - To obtain reliability estimates of the system prior to deployment (requires statistical projection)

Why fault-injection ?

- **Versus Model-based**
  - More realistic, as it evaluates actual system
  - No need to worry about mathematical feasibility
  - No need to supply input parameters

- **Versus operational measurements**
  - Failures take a ***long*** time to occur and when they do, are often not reproducible or analyzable
  - Failures provide limited insight into what ***can*** go wrong
  - Need to wait until the system is deployed - often too late

Measures to Compute



- What fraction of injected faults are activated ?
- What fraction of activated faults manifest as failures ?
- What are the average activation and failure latencies ?

Assumptions/Requirements

- A representative set of faults must be injected
  - Need to include enough faults to give confidence in the measures being studies
- Only one or controlled no. of faults injected
  - Ability to map the outcome to a set of faults
- Need to have a specification of correct behavior to distinguish incorrect outcomes
  - May need to determine golden run ahead of time

Outline

Resilience Engineering

Fault Injection

Software Fault Injection

LLFI: LLVM-based Fault Injector

Fault Injection in Cloud Applications

Software-based Fault-Injection (SWiFI)

Pros

- Do not require expensive hardware modifications
- Can target applications and OS errors
- Many hardware faults do not require probes, e.g, register data corruption

Cons

- Restricted to inject only faults that S/W can see
- May perturb the workload that is running on the system, resulting in missing many heisenbugs
- Coarser-grained time resolution than h/w

SWiFI: Types

Compile-time

- Modify source code or machine code of the program prior to execution
- Can be used to model software defects
- Requires going through the compile-run cycle each time

Runtime

- Modify the program or its data during runtime
- Can be done through the debugger, kernel or with support from compiler
- No need to go through compile-run cycle each time

Compile-time Injection

- Modify program's code prior to execution
  - Model hardware transient faults in machine code
  - Model software errors that are deterministic (Bohrbugs) on specific code paths
  - Typically only inject into the first dynamic instance of an instruction
- Main advantage: Take advantage of static analysis of the code to customize injection

Runtime Injection

- Advantages
  - Can inject faults without recompiling - speed
  - Faults can occur deeper in the execution. e.g., one-millionth iteration of a loop
    - Some of the errors can be non-deterministic (e.g., Mandelbugs)
  - Fault can depend on runtime conditions. e.g., if memory usage exceeds a threshold, inject fault (includes aging-related bugs)

Outline

Resilience Engineering

Fault Injection

Software Fault Injection

LLFI: LLVM-based Fault Injector

Fault Injection in Cloud Applications

LLFI

- **A fault injector based on LLVM (http://llvm.org)**
  - Intermediate representation (IR) level injection
  - Hybrid compile-time and runtime injection

- **Features**
  - Easy to customize the fault injection
  - Easy to analyze fault propagation
  - Accurate compared to assembly level injection

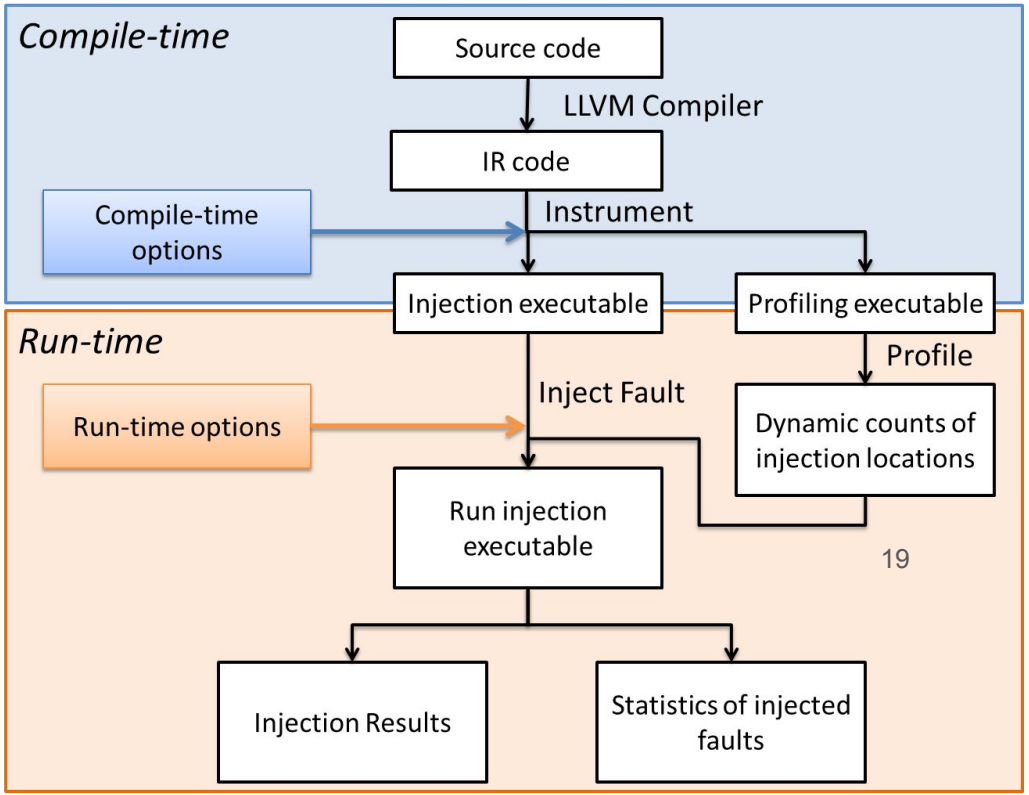
LLFI: Hybrid Compile/Runtime Injection

- **Source-level instrumentation of programs**
  - Integrated with a compiler framework, LLVM
  - Precise targeting of selected code constructs

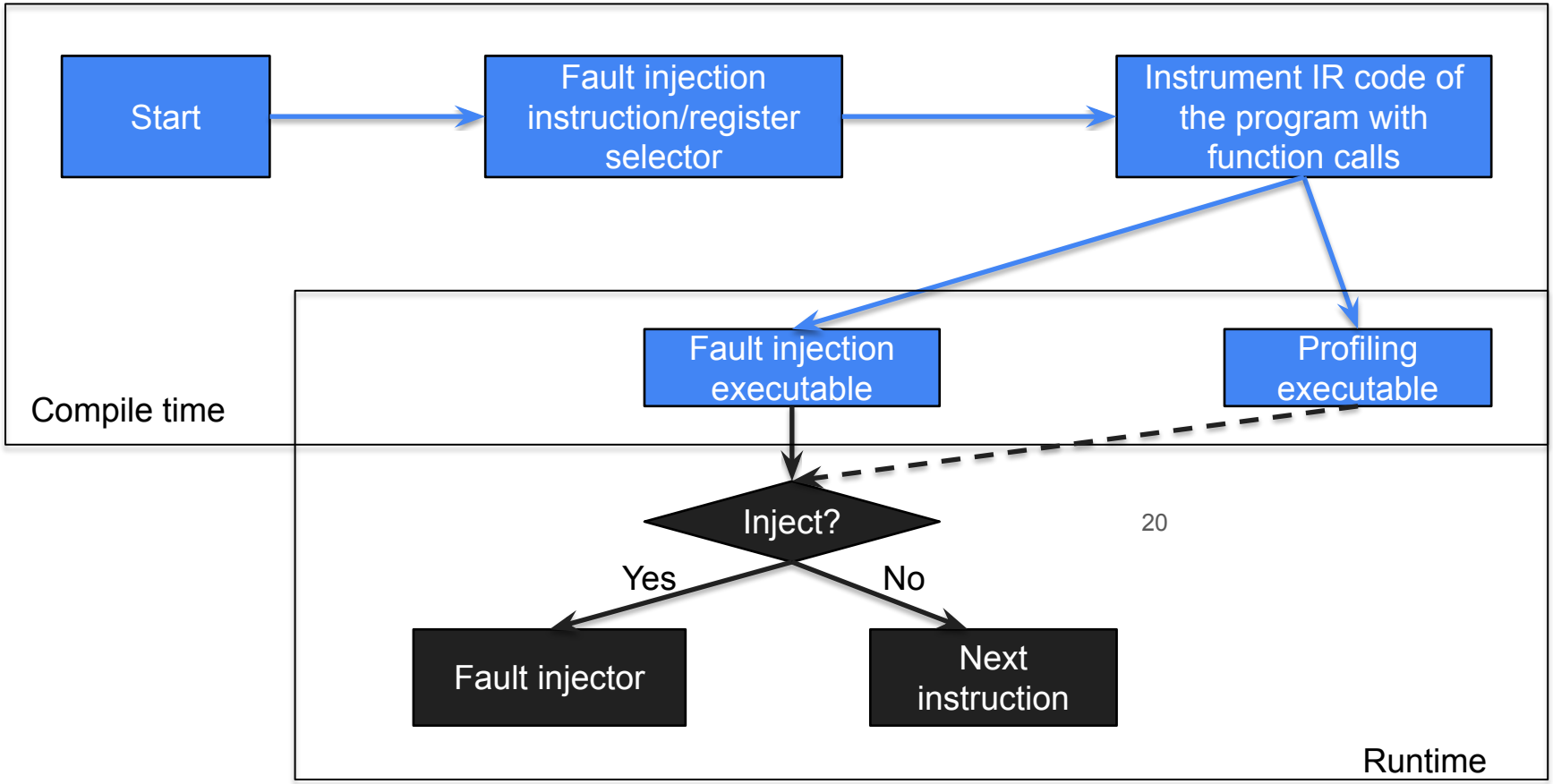
- **Fault-injection is done at program runtime**
  - Avoid going through the compile-cycle every time
  - Ability to use run-time information to inject faults

- **Tracing of faults after injecting them**
  - Map the fault back to the source code and data

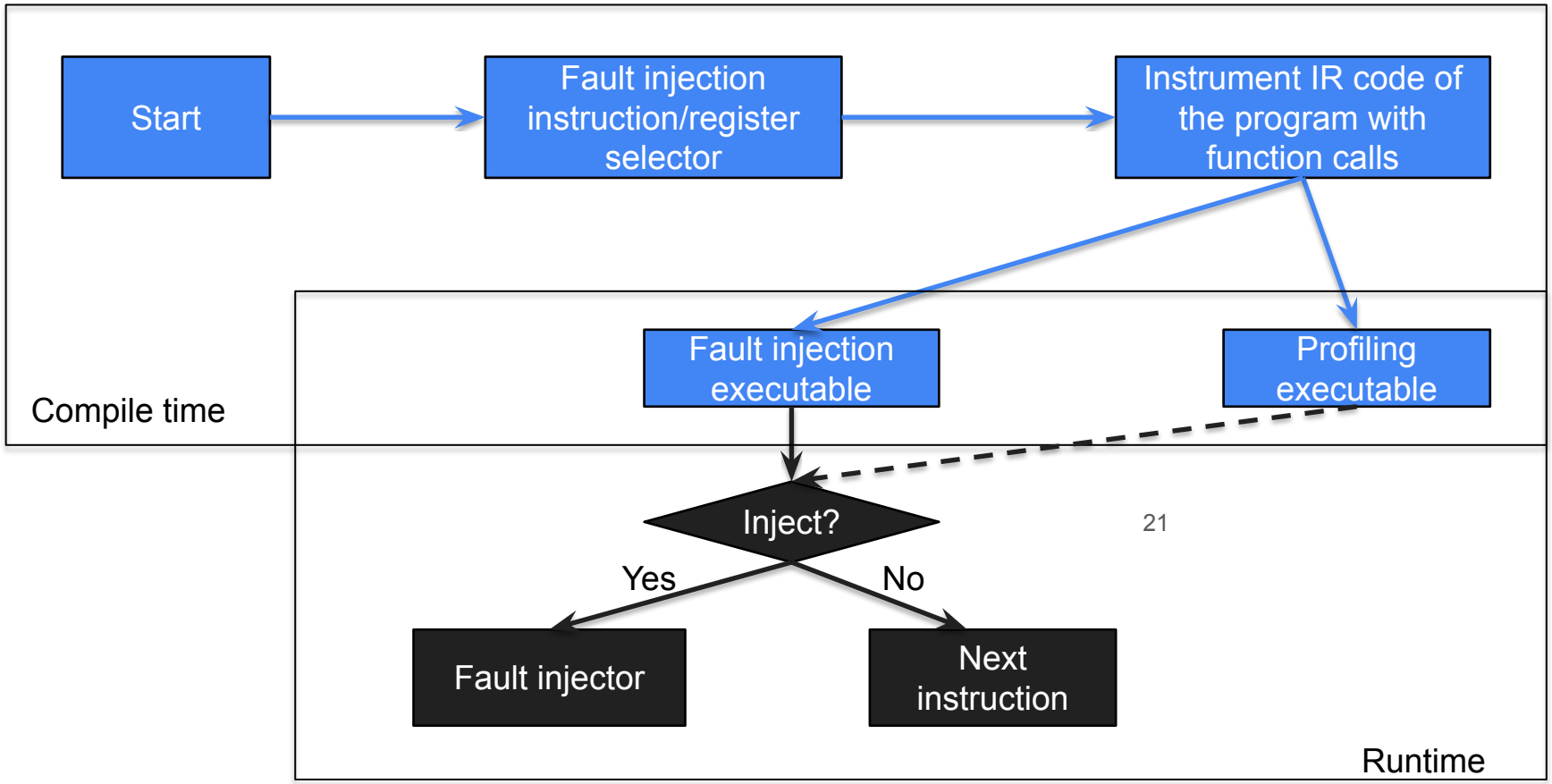
LLFI: Workflow



How does LLFI work?



How does LLFI work?



LLFI: Injection – Example

```
char* buf = “Hello World”; char* p;
void foo(int size) {
        p = (char*)malloc(size);
        memcpy(p, buf, size);
}
void goo() {
     free(p);
}
```

LLFI: Injection – Buffer Overflow

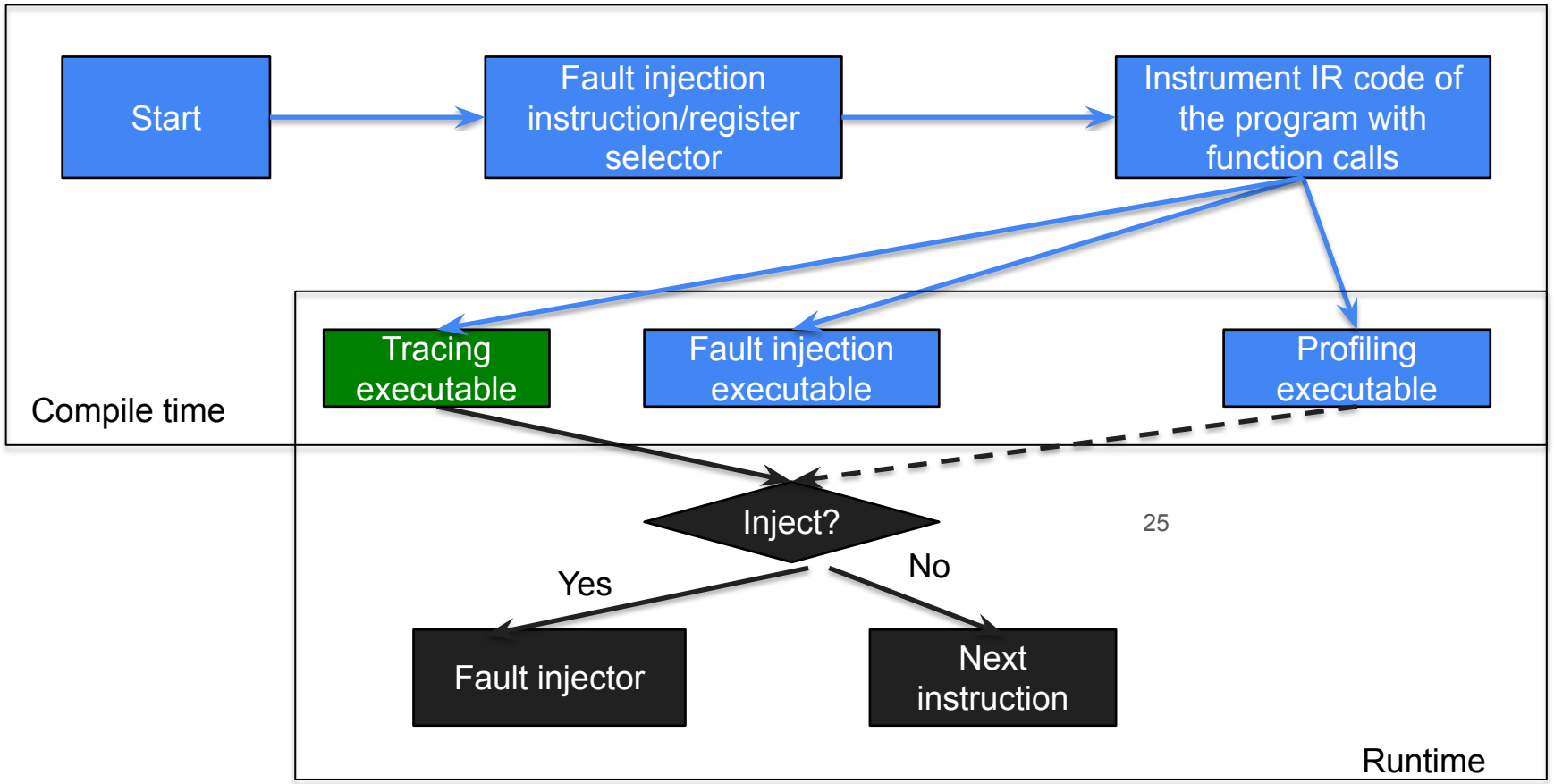
```
char* buf = “Hello World”; char* p; int count = 0;
void foo(int size) {
        p = (char*)malloc( perturbData(size, count++));
        memcpy(p, buf, size);
}
void goo() {
     free(p);
}
```

LLFI: More complex cases

- **Can inject into data of specific types/structures**
  - Example: Code that manipulates linked list nodes
  - Example: Arguments of certain function calls

- **Can inject faults at specific execution points in the program – based on the program's state**
  - Example: 100th iteration of a loop
  - Example: Call to a function when arg=some_value
  - Example: when the size of the heap > 100 KB

How does LLFI work?



LLFI: Tracing Example

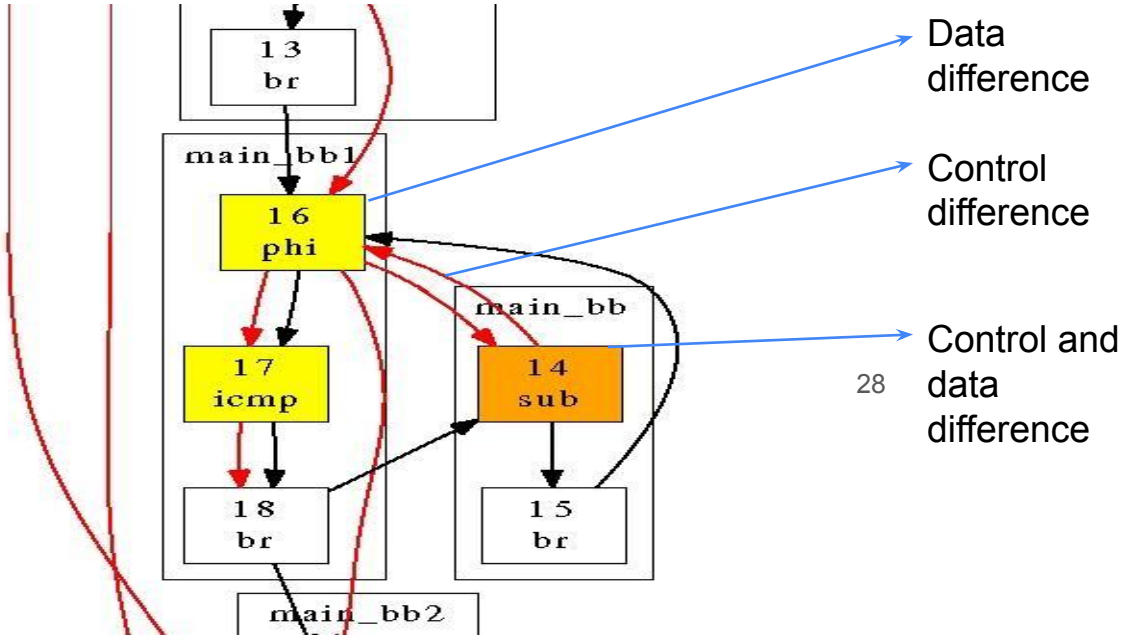
```
char* buf = “Hello World”; char* p; int count = 0;
void foo(int size) {
        p = (char*)malloc( perturbData(size, count++));
        memcpy(p, buf, size);
}
void goo() {
     free(p);
}
```

LLFI: Tracing Example

```
char* buf = “Hello World”; char* p; int count = 0;
void foo(int size) {
        p = (char*)malloc( perturbData(size, count++));
        memcpy( trace(p), buf, trace(size) );
}
void goo() {
    free( trace(p) );
}
```

LLFI: Tracing

- Graphical output of trace differences as dot file



Outline

Resilience Engineering

Fault Injection

Software Fault Injection

LLFI: LLVM-based Fault Injector

Fault Injection in Cloud Applications

Cloud Applications

Depend on many different components and services - any of these could fail

Distributed across many nodes, and even across data-centers

Failures are the norm, not the exception

Need to introduce failures systematically, in a controlled manner, **in production**

- Get engineers to think about failures from the get go during development
- Practice failure drills and ensure that playbook for failure handling is solid

ChaosMonkey: Philosophy

Learn about the system via injecting actual failures and watching what happens

Prove or disprove hypothesis about failures by observing what happens

"*We have created Chaos Monkey, a program that randomly chooses a server and disables it during its usual hours of activity. Some will find that crazy, but we could not depend on the random occurrence of an event to test our behavior in the face of the very consequences of this event. Knowing that this would happen frequently has created a strong alignment among engineers to build redundancy and process automation to survive such incidents, without impacting the millions of Netflix users. Chaos Monkey is one of our most effective tools to improve the quality of our services.*"

- Greg Orzell, Netflix Chaos Monkey Upgraded

Challenge - 1

Limit the “blast radius” of the failure, while breaking things in realistic ways

One solution: Treat fault injection as a service and isolate it to a specific set of servers or accounts

- Netflix implemented this via Failure Injection Testing (FIT)
- User writes the Failure Injection service
- Zuul executes the FIT service at the appropriate points and ensures that only the expected accounts/devices are in fact impacted

Challenge - 2

Characterize the normal behavior of the system via metrics

Compare with behavior after failure is injected to find anomalies

Challenge: How to find normal behavior, especially if it’s time varying ?

SPS Metric variation over time at Netflix (Source: https://www.oreilly.com/content/chaos-engineering/

Results of Chaos Engineering

Significant benefits in real systems (see articles on Piazza)

Used by all the big cloud companies to test their infrastructure (Gameday)

AWS introduced AWS Fault Injection Service in 2021

Curated list of resources for fault injection:

https://github.com/dastergon/awesome-chaos-engineering

Outline

Resilience Engineering

Fault Injection

Software Fault Injection

LLFI: LLVM-based Fault Injector

Fault Injection in Cloud Applications