Library-Level Fault Injection

Sijing You

Motivation

- Heavy reliance on shared libraries in many commonly used applications
- Mostly closed source
- Little/outdated documentation

What could possibly go wrong?

```
17 int i= foo(n)
18 float f= log(i)
```

Fault Model

- Some return values rarely happen, but when they do, the system should not crash
- Some return values may not be documented

➔ Fault == unexpected return value

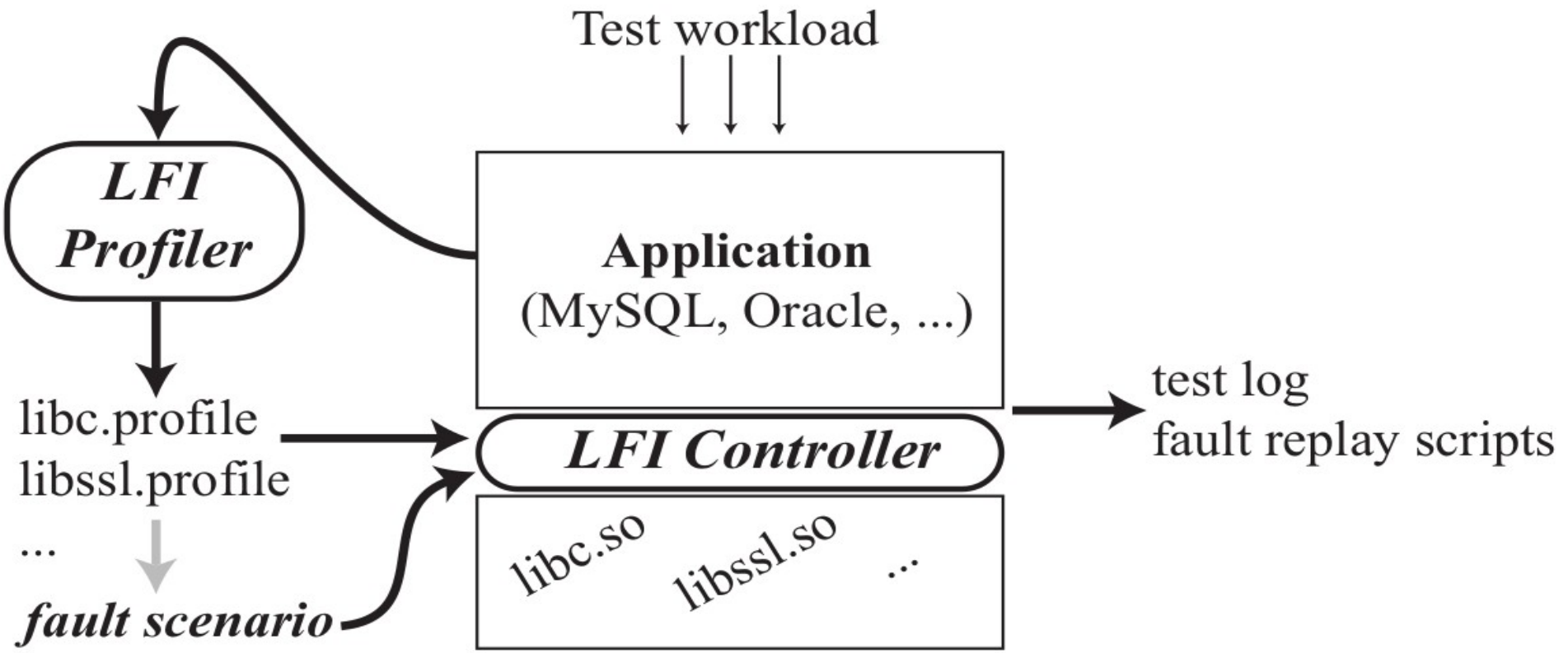
The Manual Approach

```
 1 #include <stdio.h>
 2 #include "mylib.h"
 3
 4 int main (void){
 5         int i;
 6         for (i=0;i<5;i++){
 7                 int x = /*sign(i)*/ mockup(i);
 8                 printf("Test: %d\n",x);
 9         }
10         return 0;
11 }
12
13 int mockup(int i){
14   return -1;
15 }
```

LFI – Library-Level Fault Injector

- Developed by Paul D. Marinescu and George Candea
- Aims to automate the checking process

LFI Architecture



[1] Marinescu&Candea, DSN 2009
LFI: A Practical and General Library-Level Fault Injector

Fault Scenario

- Describes what faults to inject

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <plan>
3          <trigger id="d1" class="DelayTrigger">
4            <args>
5                  <Delay>5</Delay>
6            </args>
7          </trigger>
8          <trigger id="rnd1" class="RandomTrigger">
9            <args>
10                 <percent>50</percent>
11           </args>
12         </trigger>
13   <!-- inject call -->
14         <function name="sign" retval="-1" >
15                 <!-- <triggerx ref="d1" /> -->
16                 <triggerx ref="rnd1" />
17         </function>
18 </plan>
```

Triggers

- Specify conditions, when to inject
- LFI comes with some predefined triggers
- Possible to write own triggers

Trigger Name	Description
SingleTrigger	Fires only on the first call
RandomTrigger	Fires with a configurable probability
CallCountTrigger	Fires after n calls to the specified function
TimerTrigger	Fires (on function call) after a certain time has passed
...	

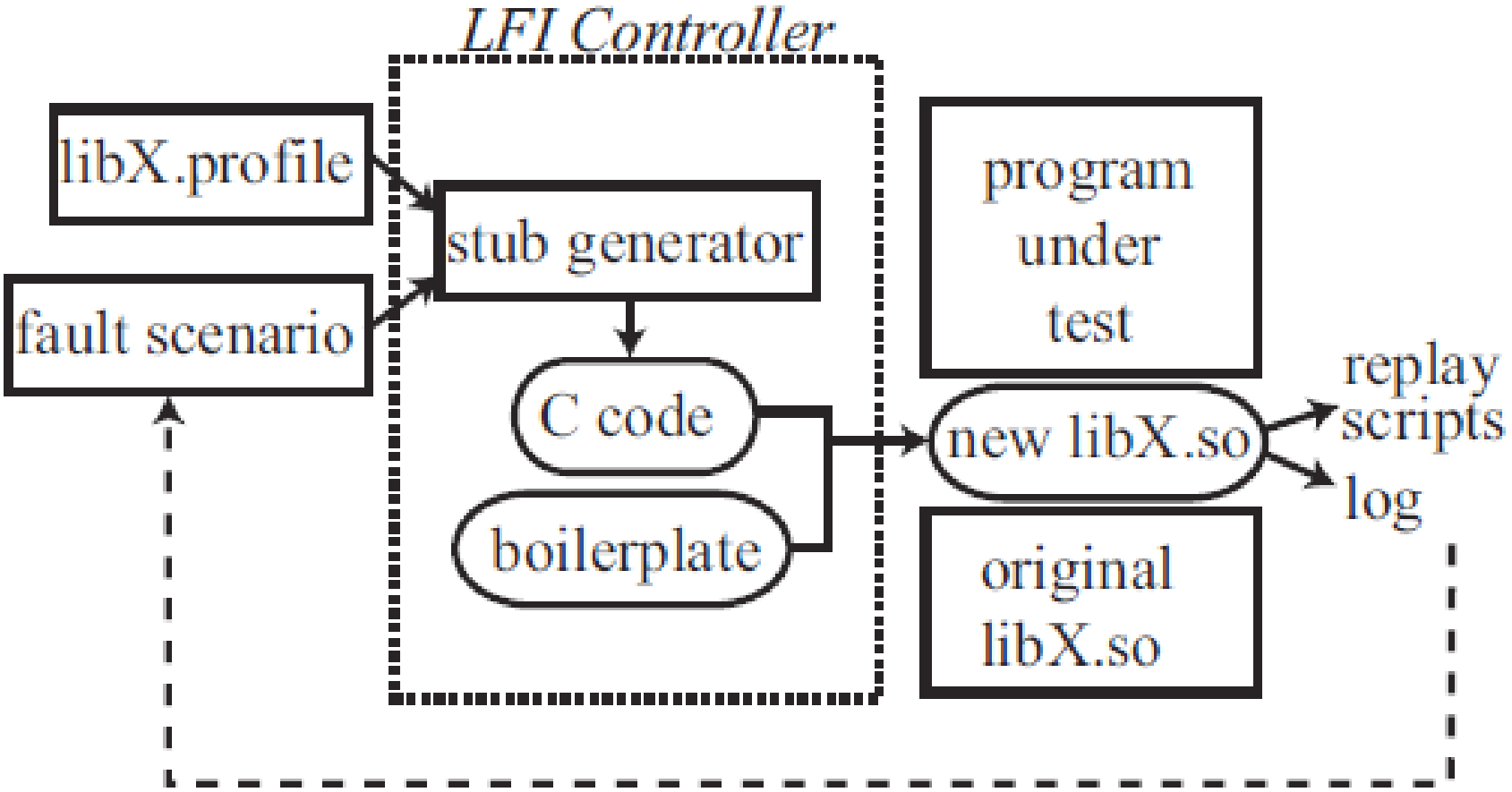
Triggers

```
36 void DelayTrigger::Init(xmlNodePtr initData)
37 {
38   xmlNodePtr nodeElement, textElement;
39
40   nodeElement = initData->children;
41   while (nodeElement)
42   {
43     if (XML_ELEMENT_NODE == nodeElement->type &&
44       !xmlStrcmp(nodeElement->name, (const xmlChar*)"Delay"))
45     {
46       textElement = nodeElement->children;
47       if (XML_TEXT_NODE == textElement->type)
48         Delay=atoi((char*)textElement->content);
49     }
50     nodeElement = nodeElement->next;
51   }
52 }
```

Triggers

```
53 bool DelayTrigger::Eval(const string*, ...)
54 {
55   sleep(Delay);
56   return false;
57 }
```

LFI Controller

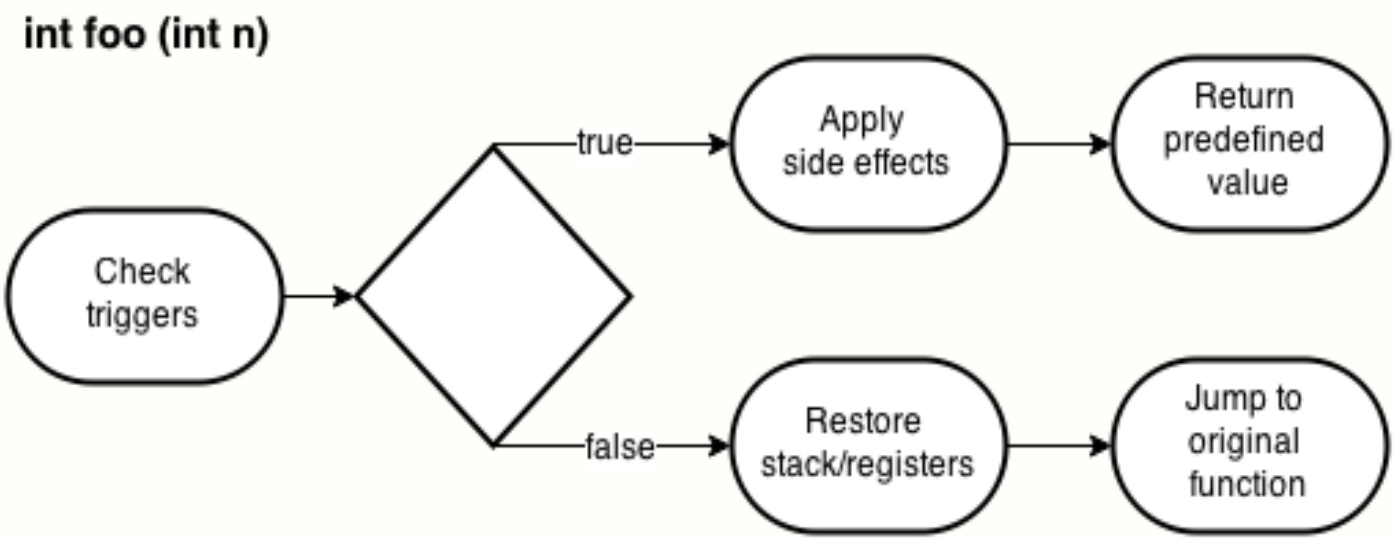


[1] Marinescu&Candea, DSN 2009
LFI: A Practical and General Library-Level Fault Injector

LFI Controller

- Compile custom library
- Run program with LD_PRELOAD giving priority to the newly generated library

LFI Controller



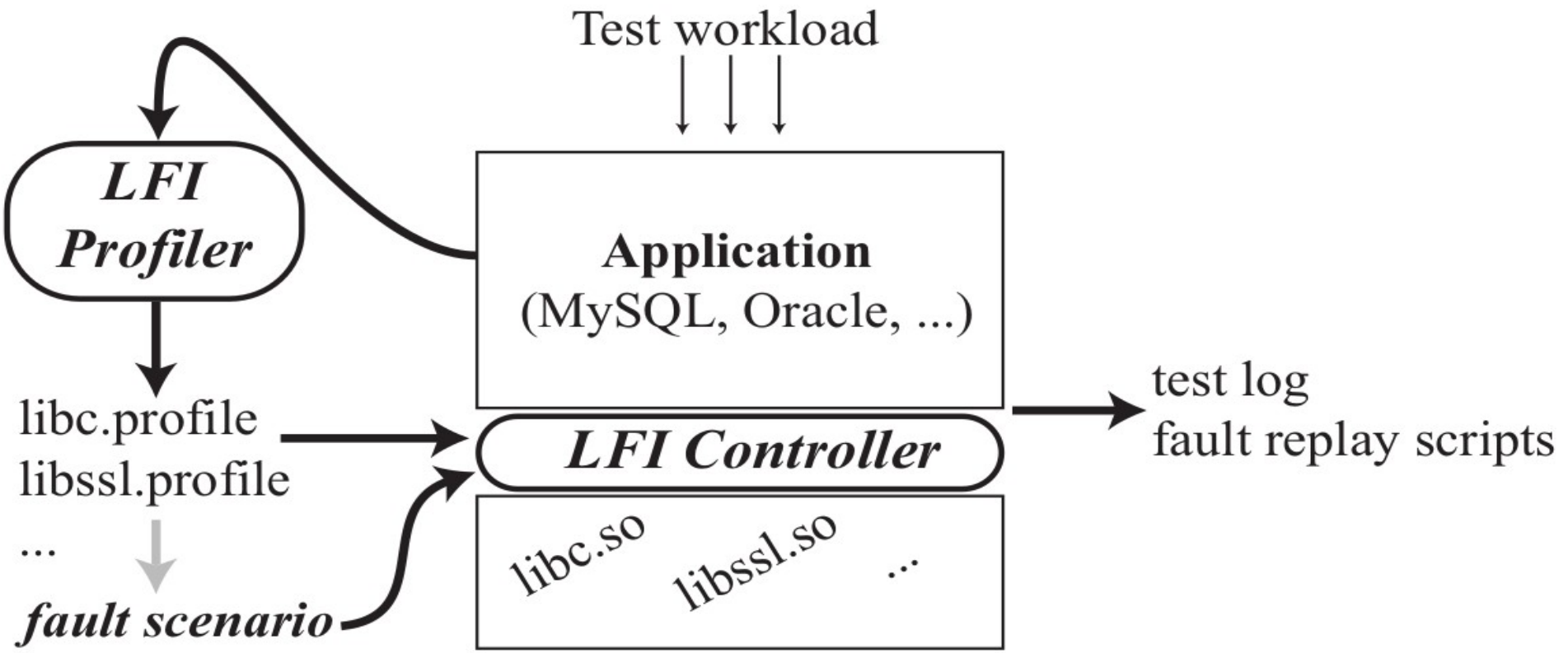
Demo

- <show Profiler>
- <show cs-analyzer>
- <show lfi injection>

Evaluation

- Good for testing edge cases
- Doesn't require changes in source code
- As provided, still too much manual work required

LFI Architecture



[1] Marinescu&Candea, DSN 2009
LFI: A Practical and General Library-Level Fault Injector

Summary

- **Fault Model**
  - unexpected return values
- **Purpose**
  - Test fault tolerance of the system
- **Trigger mechanism**
  - Time-based, location-based, chance-based
- **Injection time**
  - At library load time
- **Injection level**
  - Intermediate code representation (between program and library)

References

- [1] Paul D. Marinescu, George Candea
  - LFI: A Practical and General Library-Level Fault Injector (DSN 2009)
- Paul D. Marinescu, George Candea
  - Efficient Testing of Recovery Code Using Fault Injection (ACM 2011)
- LFI Git repository
  - https://github.com/dslab-epfl/lfi