FIDL

A Fault Injection Description Language for Compiler-based Tools

Maryam Raiyat Aliabadi
Karthik Pattabiraman
University of British Columbia (UBC) Canada

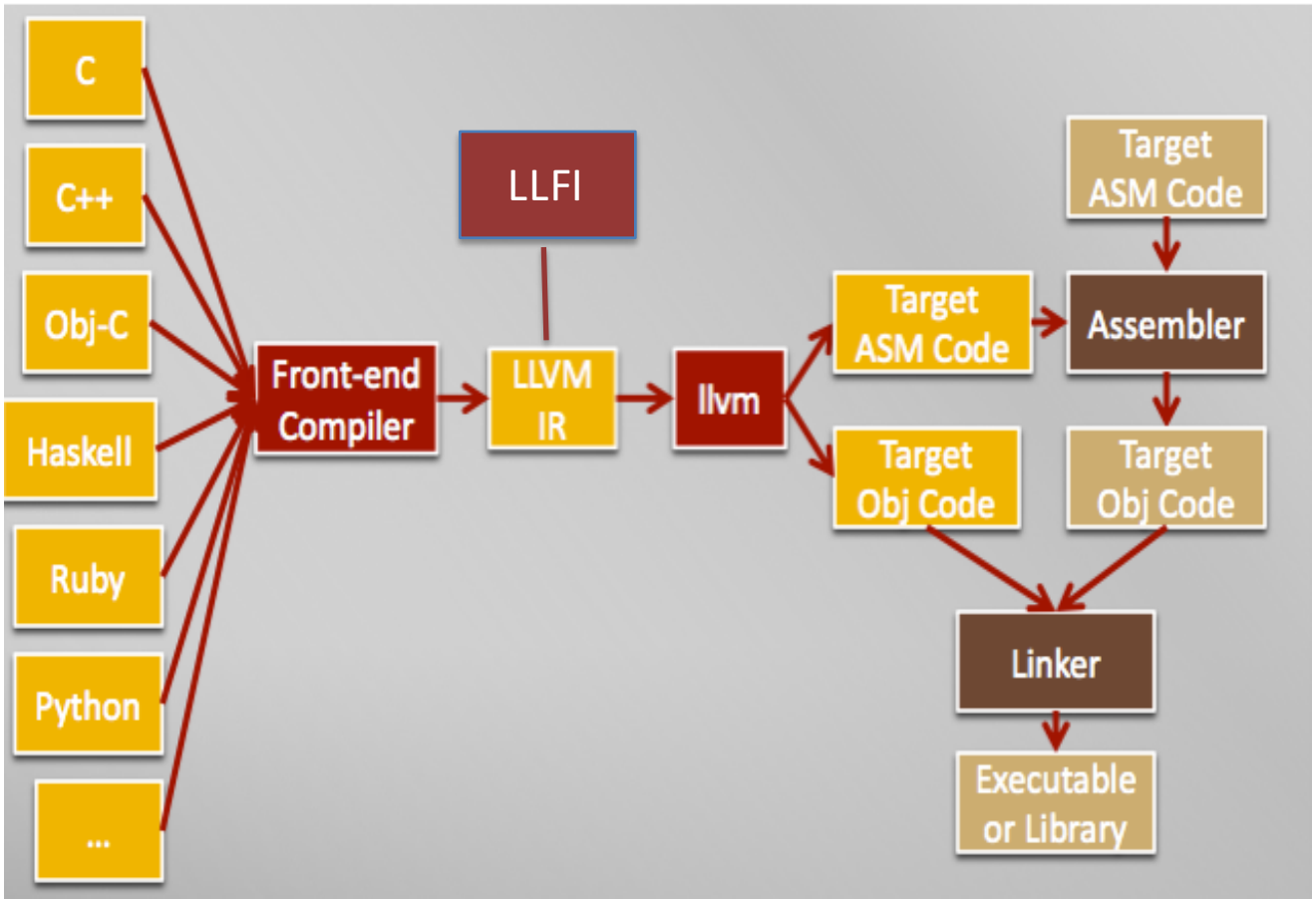
Introduction

Testing the code against faults is a ‘must’

FI: A *systematic way to model faults*

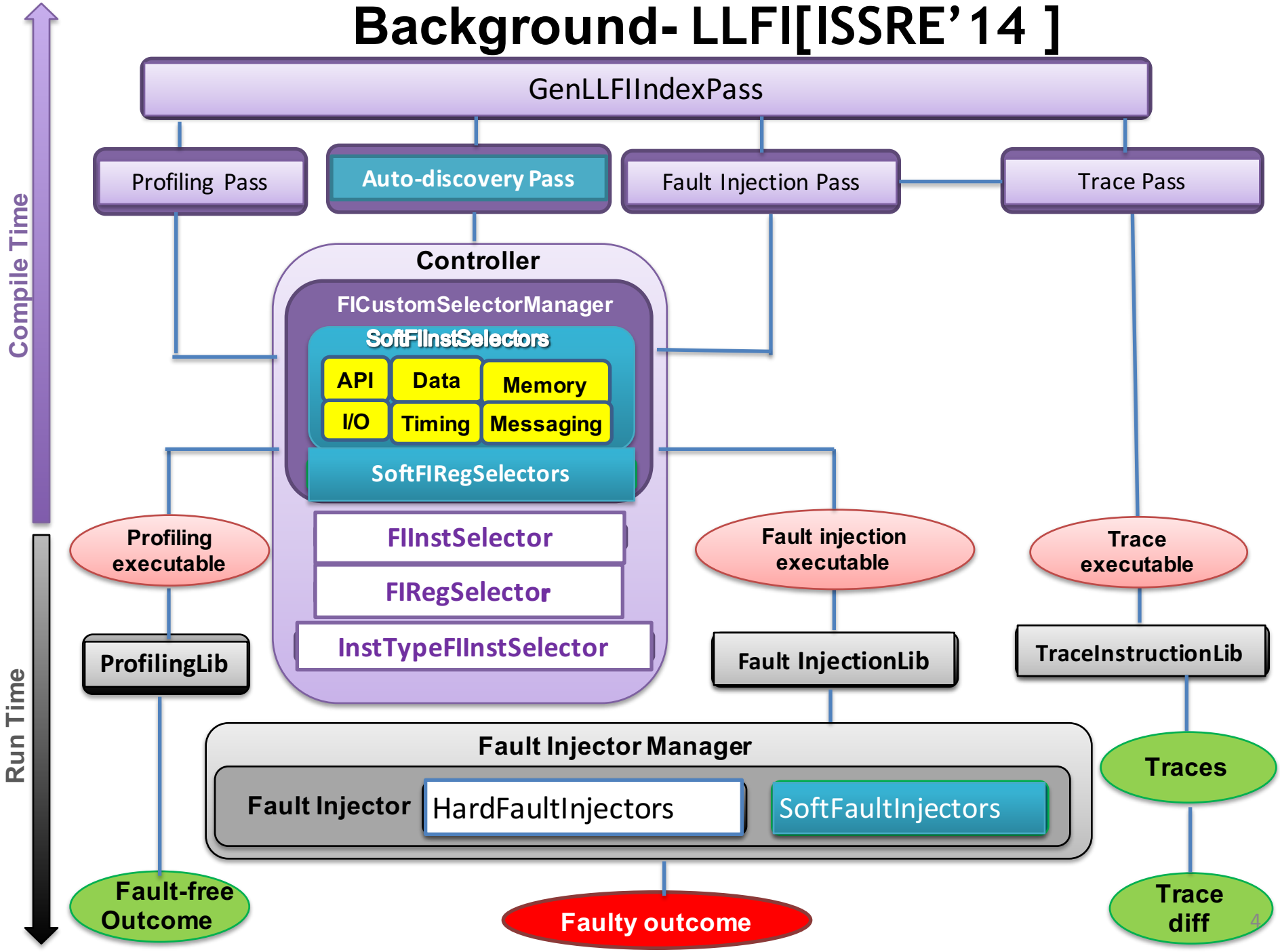
Background-LLVM

❖ **Widely used in industry [Lattner'05]**



Wei J, Thomas A, Li G, Pattabiraman K. Quantifying the accuracy of high-level fault injection techniques for hardware faults. In DSN'14.

Background- LLFI[ISSRE'14]



Primary Motivation

- How to make LLFI programmable?
  - Complex coding VS. Simple scripts

```
#include "llvm/IR/Instructions.h"
#include "llvm/IR/Constants.h"
#include "llvm/IR/IntrinsicInst.h"
#include <fstream>
using namespace std;
namespace llfi {
   class FuncArgRegSelector: public
SoftwareFIRegSelector {
   public:
      FuncArgRegSelector(int target_arg) :
pos_argument(target_arg), specified_arg(true) {};
      FuncArgRegSelector():pos_argument(0),
specified_arg(false) {};
   private:
      int pos_argument;
      bool specified_arg;
      virtual bool isRegofInstFITarget(Value *reg,
Instruction *inst);
      virtual bool isRegofInstFITarget(Value* reg,
Instruction* inst, int pos);};
   class FuncDestRegSelector: public
SoftwareFIRegSelector {
   private:
      virtual bool isRegofInstFITarget(Value *reg,
Instruction *inst);
    bool
FuncArgRegSelector::isRegofInstFITarget(Value *reg,
Instruction *inst){
       if(isa<CallInst>(inst) == false){
          return false }else{
          CallInst* CI = dyn_cast<CallInst>(inst);
          if(this->specified_arg == true){
```

```
1  Failure_Class: Class1
2  Failure_Mode: FMode1
3
4  New_Failure_Mode:
5      Trigger:
6          call: [fopen, open]
7      Target: dst
8      Action: Corrupt
```

Challenges - Why a new language?

combinatorial explosion

distinct roles

Extensibility

Solution: FIDL

- **FIDL: Fault Injection Description Language**
  - ***Efficiently*** makes SFI tool programmable
  - ***Simplified*** and ***accelerated*** fault model design
  - ***Dynamically*** extends SFI tool

FIDL - Test management

- A systematic way to fault scenario design
  - ***Flexibly manages combinatorial explosion***
- Combines heuristics with fault model
  - Security analysis

FIDL - Extensibility

- **FIDL:** an Aspect-Oriented Programming (AOP) Language

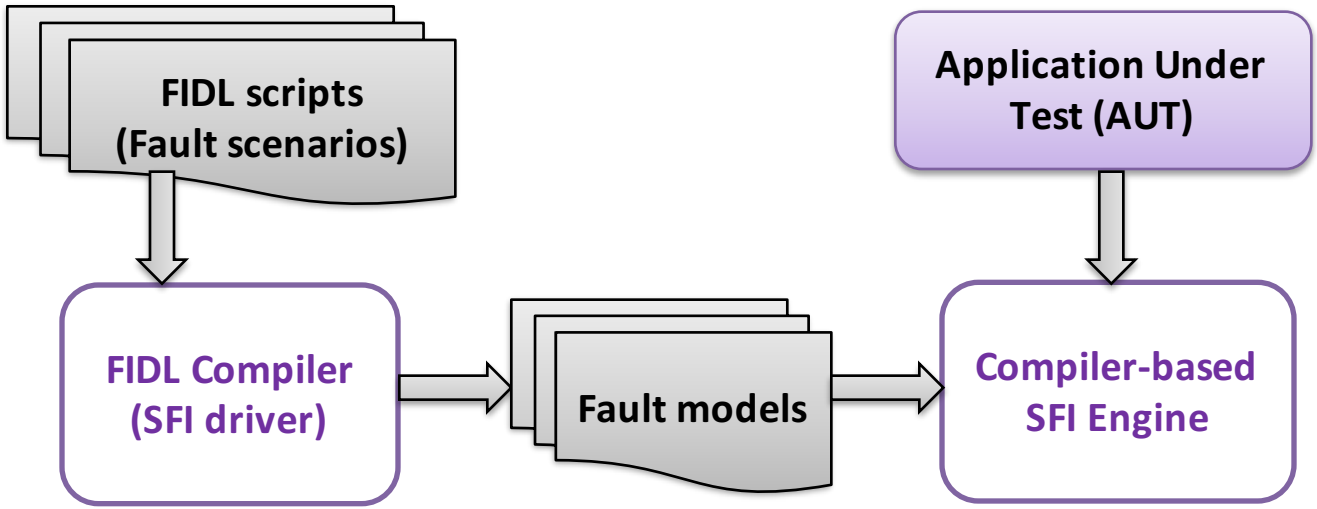
  - Aspects in FIDL scripts
  - Weaves aspects into the source code of SFI tool
  - ***Automatically extends SFI tool***

FIDL - High level abstraction

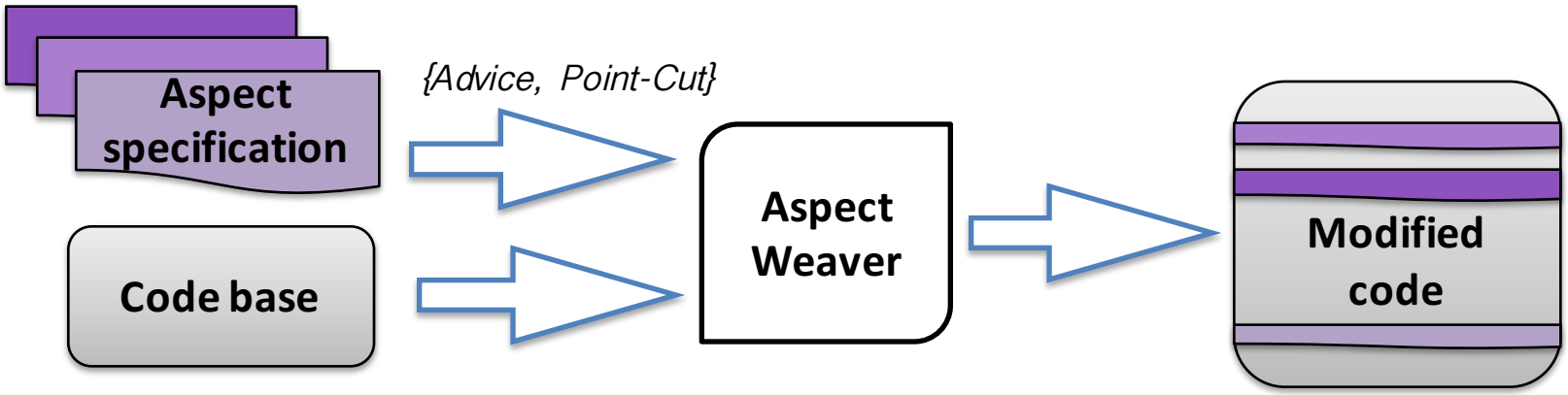
- Encapsulation
  - Hides compiler details
  - Simplifies fault model design process
  - ***Separates roles of tester and SFI tool developer***
- Acceleration
  - Plug & play design

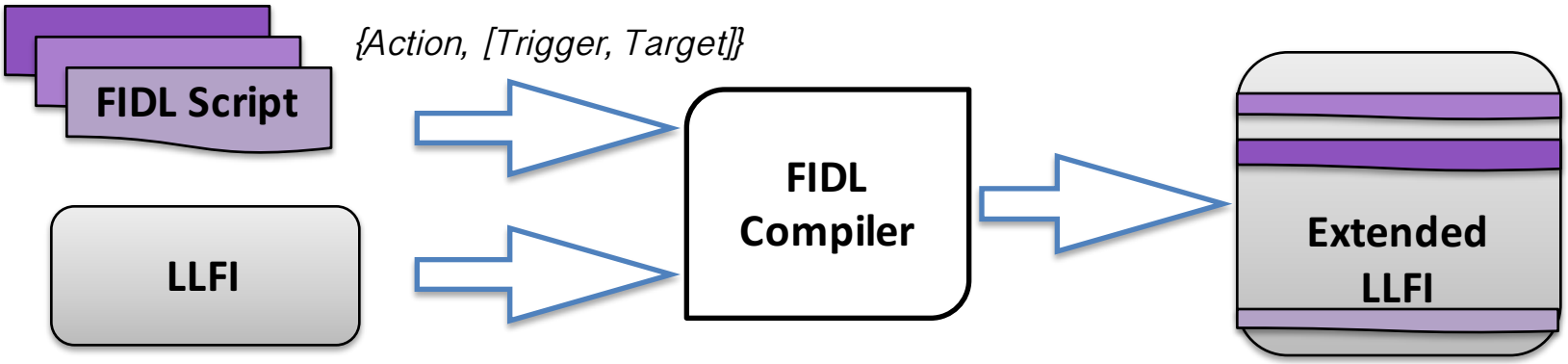
FIDLFI: A programmable framework

- FIDLFI = FIDL + Compiler-based SFI tool



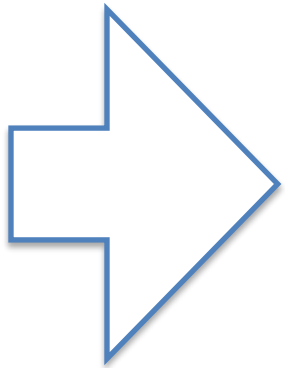
FIDLFI workflow



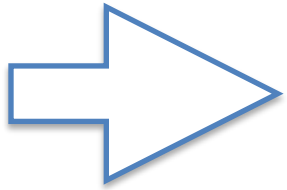


FIDL Script Specification

```
Failure_Class:
Failure_Mode:
New_Failure_Mode:
{
    Trigger:  <IR instructions>

    Trigger*: <Refined IR instructions>

    Target:    <IR registers>



    Action:
     <Corrupt/SetValue/Freeze/Delay/Perturb>

}
```



Fault trigger module



Fault injector module

FIDL script Example- Heart bleed

```
1 Failure_Class: Memory
2 Failure_Mode: Heartbleed
3 New_Failure_Mode:
4    Trigger:  call: [memcpy]

6    Target: memcpy :: src[2]
7    Action: Perturb :: CustomInjector
8    Custom_Injector:
9            *Target = *Target + 10000;
```

Application injected by Heart bleed

```
11:   const char src[50] = "http://www.tutorialspoint.com";
12:      char dest[50];
13:      printf("Before memcpy dest = %s\n", dest);
14:        memcpy(dest, src, strlen(src)+1);
15:    printf("After memcpy dest = %s\n", dest);
......
25:   strcpy(temp, message2);
26:    printf("\n\nOriginal message: %s", temp);
27:    memcpy(temp + 4, temp + 16, 10);
28:    printf("\nAfter memcpy() without overlap:\t%s", temp);
29:    strcpy(temp, message2);
30:    memcpy(temp + 6, temp + 4, 10);
.....
43:  char myname[] = "Pierre de Fermat";
/* using memcpy to copy string: */
44: memcpy ( person.name, myname, strlen(myname)+1 );
......
456: void *(memcpy)(void * restrict s1, const void * restrict s2, size_t n)
457: {
458:    char *dst = s1;
459:    const char *src = s2;
460:    while (n-- != 0)
461:        *dst++ = *src++;
```

FIDL script Example- Heart bleed

```
1 Failure_Class: Memory
2 Failure_Mode: Heartbleed
3 New_Failure_Mode:
4     Trigger:  call: [memcpy]
5     Trigger*: [14,44]
6     Target: memcpy :: src[2]
7     Action: Perturb :: CustomInjector
8     Custom_Injector:
9             *Target = *Target + 10000;
```

Experimental Evaluation

- **Experimental setup**
  - Five sample fault models
  - Four standard benchmarks, and the Null-httpd web server
  - 2000-run SFI campaign for every fault model

- **Evaluation metrics**
  - Time overhead
  - Implementation overhead
  - Complexity

Experimental Results: Complexity

- 10X complexity reduction

Fault Model	FIDL Script (LOC)	FFM (LOC)	OFM (LOC)
Buffer Overflow	**9**	96	**68**
Memory leak	**11**	71	**68**
Data Corruption	**8**	64	**61**
Wrong API	**11**	111	**109**
G-Heartbleed	**10**	112	**81**

OFM (Original Fault Model) : primarily developed in LLFI in C++ language
FFM (FIDL-generated Fault Model) : translated from FIDL script to C++ code

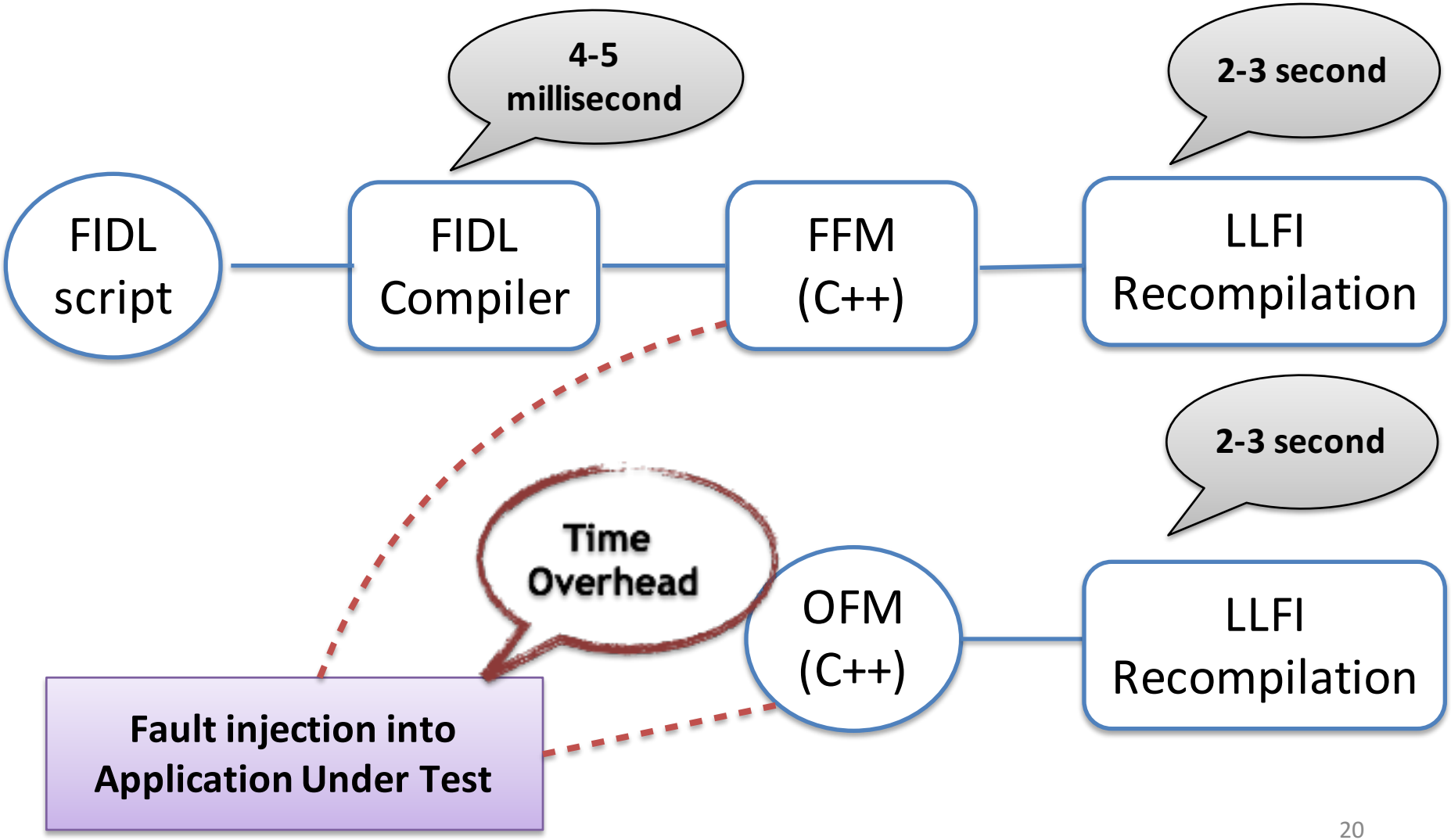
Experimental Results: Space overhead

- 4-18 % space overhead

Fault Model	FIDL Script (LOC)	FFM (LOC)	OFM (LOC)
Buffer Overflow	9	**96**	**68**
Memory leak	11	**71**	**68**
Data Corruption	8	**64**	**61**
Wrong API	11	**111**	**109**
G-Heartbleed	10	**112**	**81**

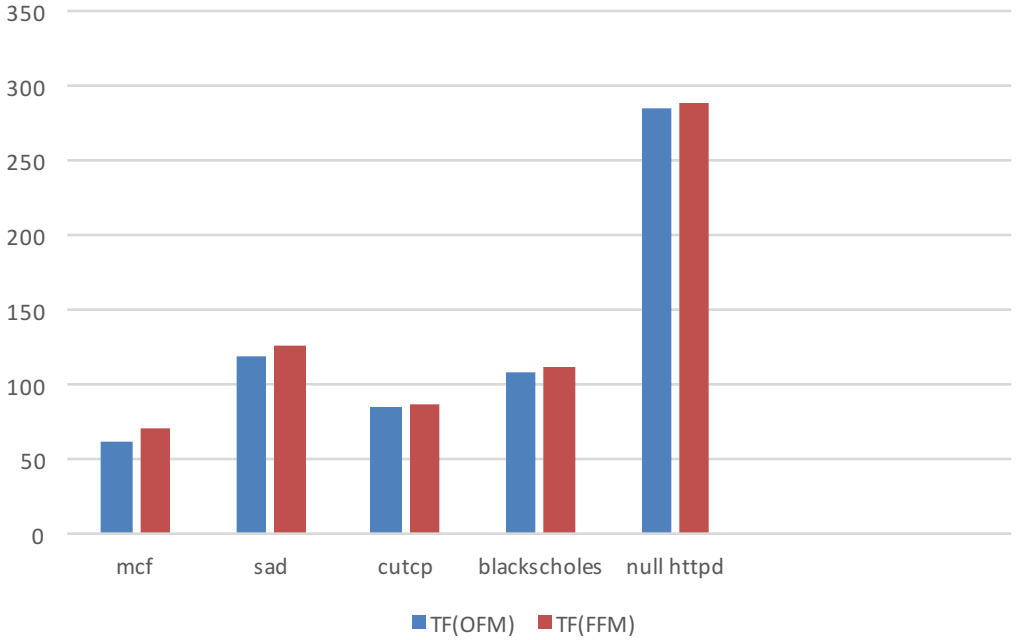
OFM (Original Fault Model) : primarily developed in LLFI in C++ language
FFM (FIDL-generated Fault Model) : translated from FIDL script to C++ code

Overall Performance



Experimental results: Time overhead

- Maximum time overhead : 6.7 %
- Average time overhead : 3.9%



TF(OFM) : Average Time overhead for an original fault model

TF(FFM): Average Time overhead for a FIDL-generated fault model

Figure1. Comparing Time overhead (%) of memory leak's fault model across benchmarks.

Previous work Vs FIDLFI

FI Tool	Programmability	High level Abstraction	Extensibility
FIG	Domain Specific Language (DSL)	Yes	No
LFI	XML-based	No	No
FAIL*	DSL (FAIL)	Yes	No
PREFAIL	Policy-based	Yes	No
EDFI	Command Line Arguments	No	No
FIDLFI	**FIDL**	**Yes**	**Yes**

Summary

- **FIDL: Fault Injection Description Language**
  - Drives compiler-based SFI tools
  - Develops fault models in a Plug & play fashion
  - Reduces the complexity of fault models by 10 times with negligible time overhead.
- **Integrated into LLFI Framework (BSD license)**

https://github.com/DependableSystemsLab/LLFI

raiyat@ece.ubc.ca