Simulator-based diff-time performance (regression) testing

Ivan Postolski, Victor Braberman, Diego Garbervetsky, Sebastián Uchitel

LaFHIS, ICC, CONICET, Argentina

ICSE 2019

The problem: Detect Performance Regressions

- Focus on regressions produced by ***changes*** in the program that result in ***expensive calls*** executed ***more times*** than expected

The problem: Detect Performance Regressions

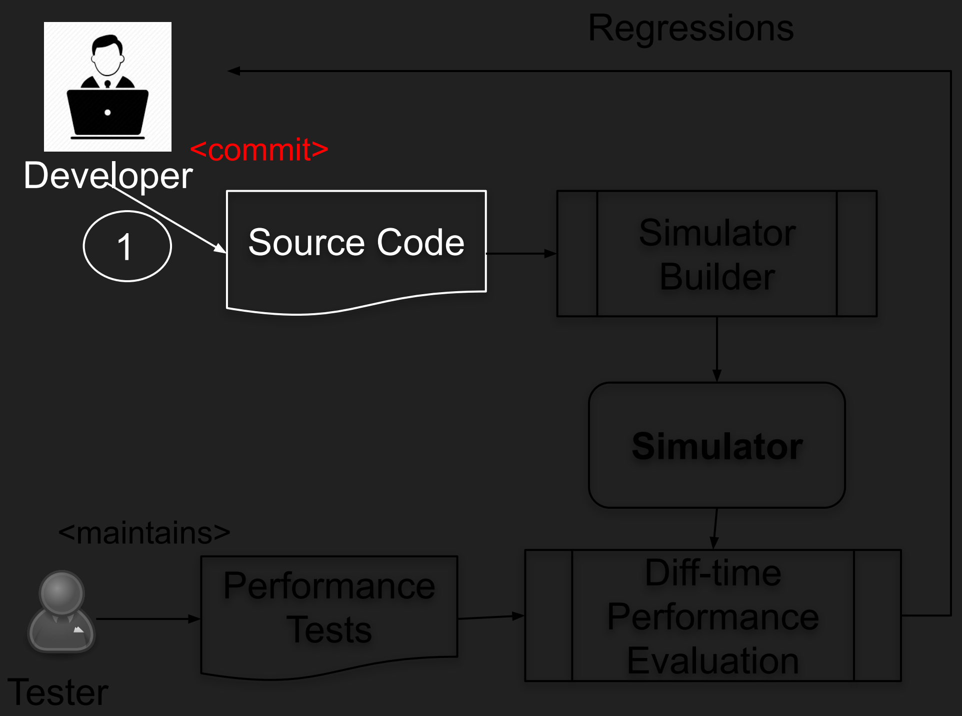
- Focus on regressions produced by ***changes*** in the program that result in ***expensive calls*** executed ***more times*** than expected

- We aim to flag this regressions in ***diff-time*** (i.e. right after a commit is introduced)

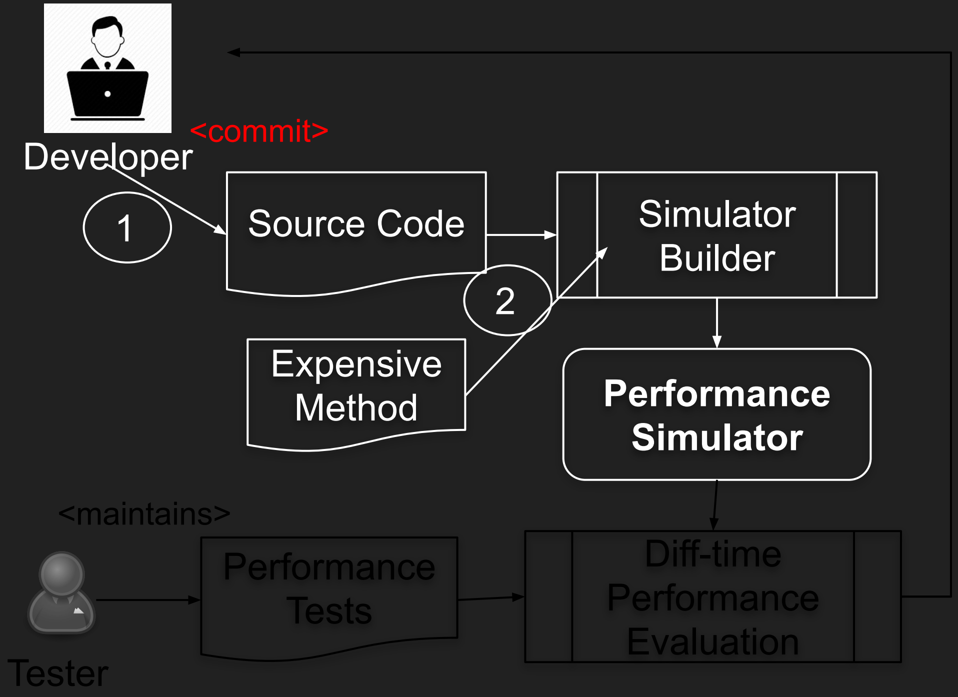
The problem: What we assume

- The expensive calls are **known**, selected by a third party
- The PUA **code** is available
- There are **functional** and **performance tests** in place

How do we plan to solve it?



How do we plan to solve it?

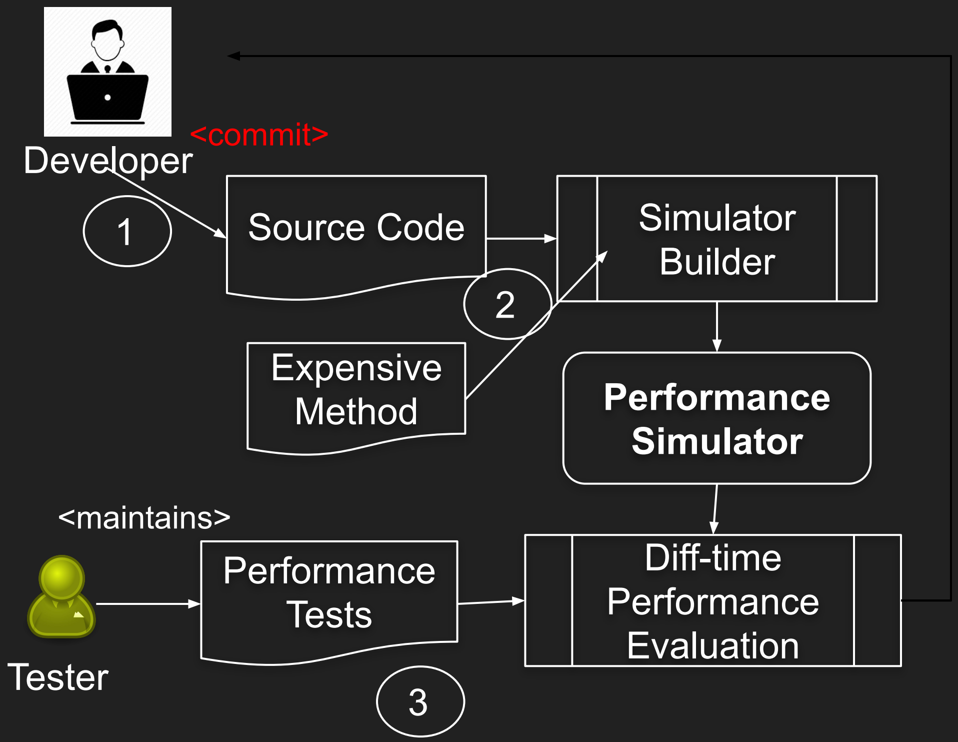


Performance simulator def:

A **new program,** that given an **input** (*i*) and an expensive method (*m*) returns the **exact number** of times that the original program **would have executed *m*** given *i*.

Run **faster** than the original program

How do we plan to solve it?

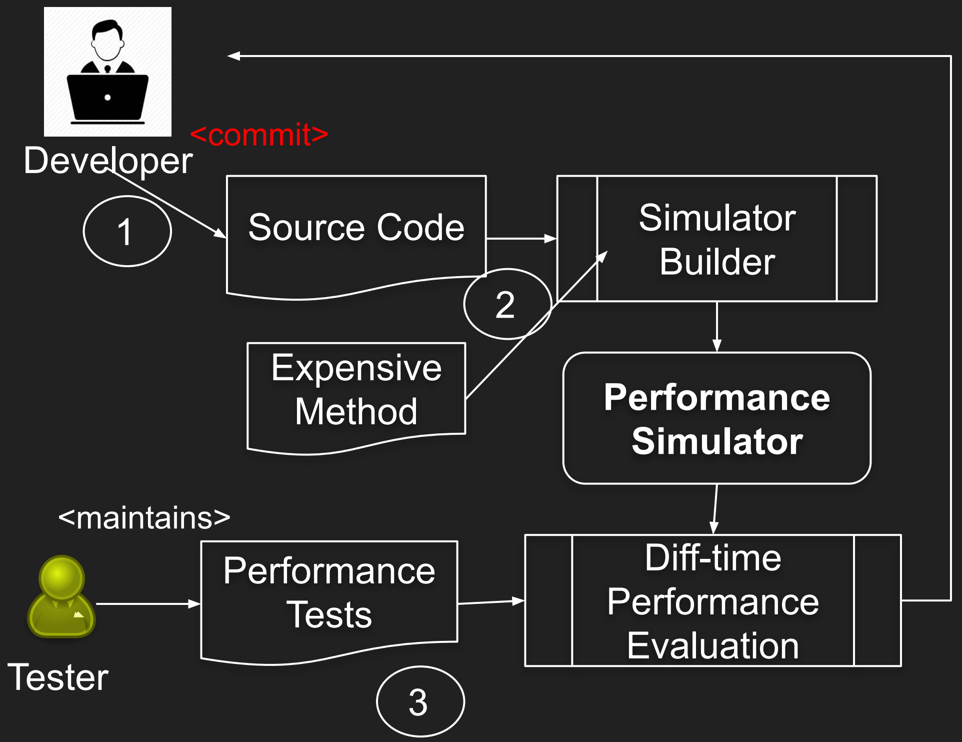


Performance simulator def:

A **new program,** that given an **input** (*i*) and an expensive method (*m*) returns the **exact number** of times that the original program **would have executed *m*** given *i*.

Run faster than the original program

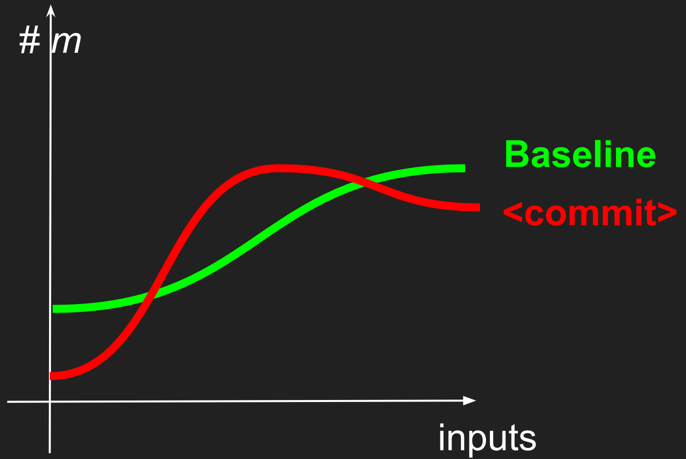
How do we plan to solve it?



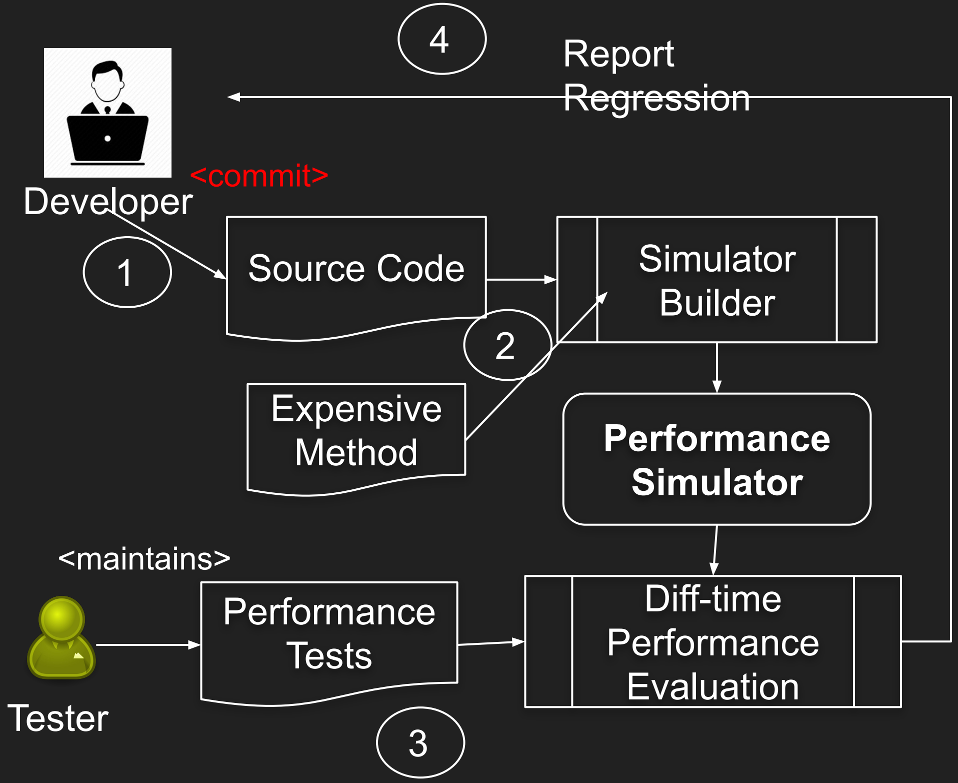
Performance simulator def:

A **new program,** that given an **input** (*i*) and an expensive method (*m*) returns the **exact number** of times that the original program **would have executed *m*** given *i*.

Run faster than the original program



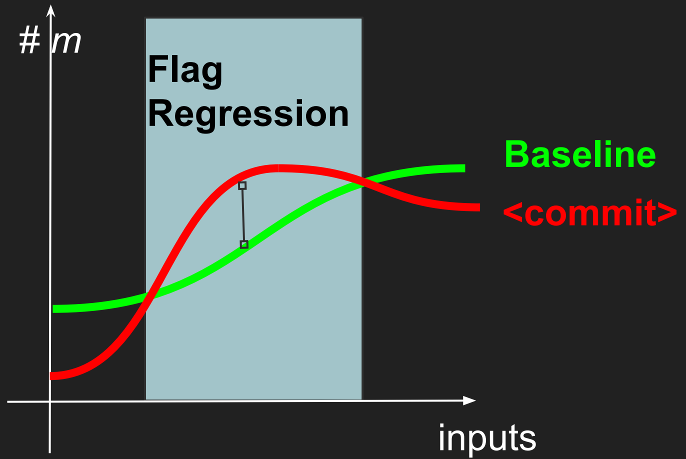
How do we plan to solve it?



Performance simulator def:

A **new program,** that given an **input** (*i*) and an expensive method (*m*) returns the **exact number** of times that the original program **would have executed *m*** given *i*.

Run faster than the original program



How do we build a performance simulator?

- A trivial simulator:
  - Adding a counter (*cm*) inside *m* body
  - Returning *cm* value at program exit

How do we build a performance simulator?

- ~~A trivial simulator:~~
  - ~~Add a counter (cm) inside *m* body~~
  - ~~Return the counter at program exit~~

- Bad simulator: cannot be faster than the original program. (not suitable for diff-time)

Using program **slicing**

- If we instrument a counter (*cm*) inside *m*, and we slice the original program, this yields a simulator

Using program **slicing.** A decision

- If we instrument a counter (*cm*) inside *m*, and we slice the original program, this yields a simulator

Using program **slicing.** A decision

- If we instrument a counter (*cm*) inside *m*, and we slice the original program, this yields a simulator

Using program **slicing.** A decision

- If we instrument a counter (*cm*) inside *m*, and we slice the original program, this yields a simulator

- Precision vs. Scale
- Adds more statements than necessary

Use **executable dynamic** slicing as simulators

- Fewer statements than static, likely to be **faster** than the original program ✅

Use **executable dynamic** slicing as simulators

- Fewer statements than static, likely to be **faster** than the original program ✅
- Problem**: May not be correct** for other inputs than the inputs used to build it

Use **executable dynamic** slicing as simulators

- Fewer statements than static, likely to be **faster** than the original program ✅

- Problem**: May not be correct** for other inputs than the inputs used to build it

- Propose a novel solution: Given an input, we try to proof that the dynamic slice result will be correct

A Dynamic Slice **Correctness Certificate**

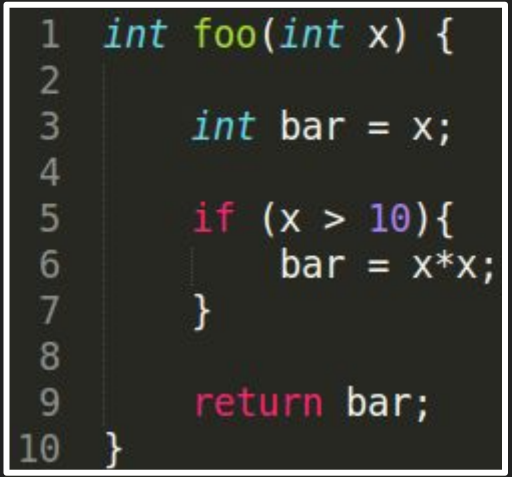
- Given an input *i,* and a dynamic slice *ds,* a **certificate is a program** that returns true if *ds* is correct for *i*

- Then given a performance test input *i,* if the certificate returns true, we will execute the input under the simulator

- The certificate also needs to be fast

How to build a **Correctness Certificate**

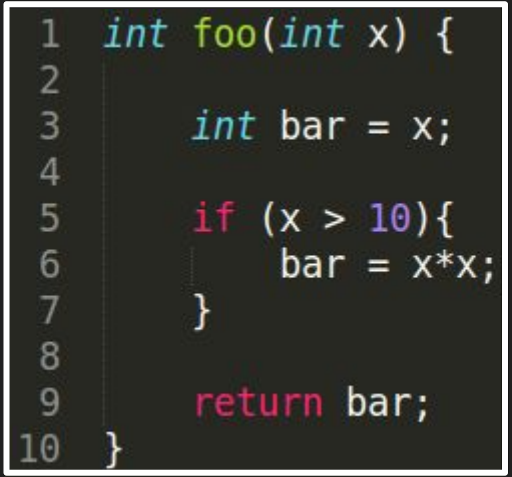
Original Program



```
int foo(int x) {

    int bar = x;

    if (x > 10){
        bar = x*x;
    }

    return bar;
}
```

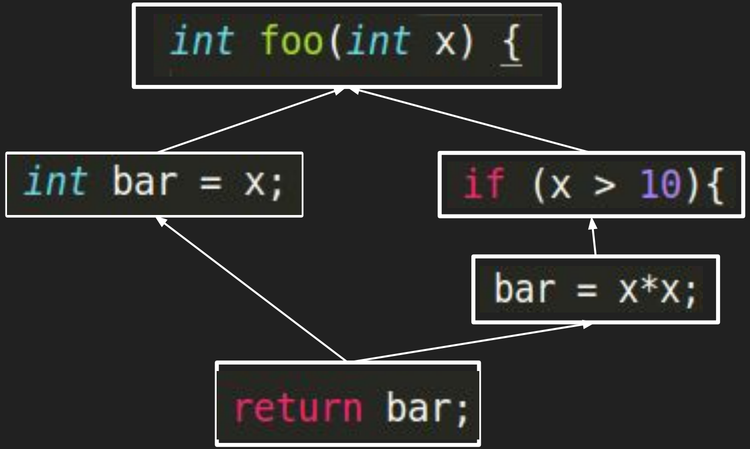
How to build a **Correctness Certificate**

Original Program



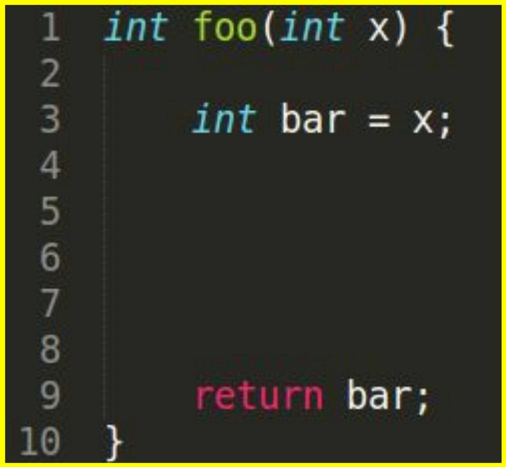
```
int foo(int x) {

    int bar = x;

    if (x > 10){
        bar = x*x;
    }

    return bar;
}
```

Dependency Graph



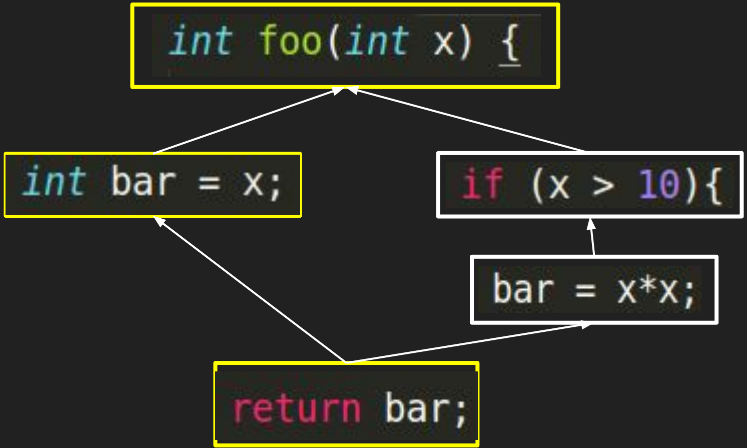
How to build a **Correctness Certificate**

Dynamic Slice (x := 1)



```
int foo(int x) {

    int bar = x;






    return bar;
}
```

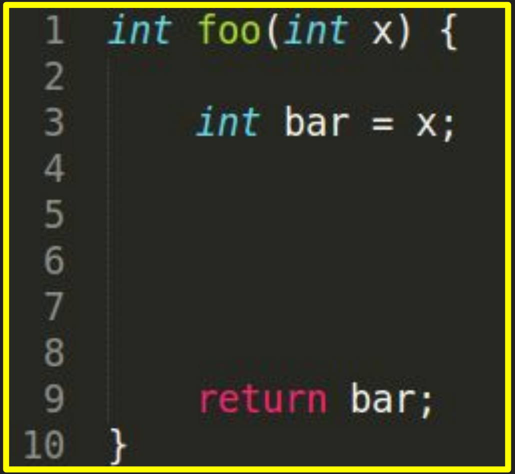
Dependency Graph



```
int foo(int x) {
int bar = x;
if (x > 10){
bar = x*x;
return bar;
```

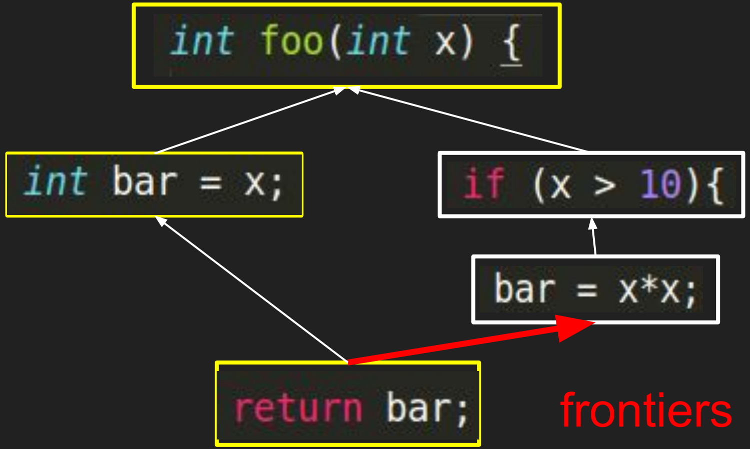
How to build a **Correctness Certificate**

Dynamic Slice (x := 1)



```
int foo(int x) {

    int bar = x;






    return bar;
}
```

Dependency Graph



```
int foo(int x) {
```

```
int bar = x;
```

```
if (x > 10){
```

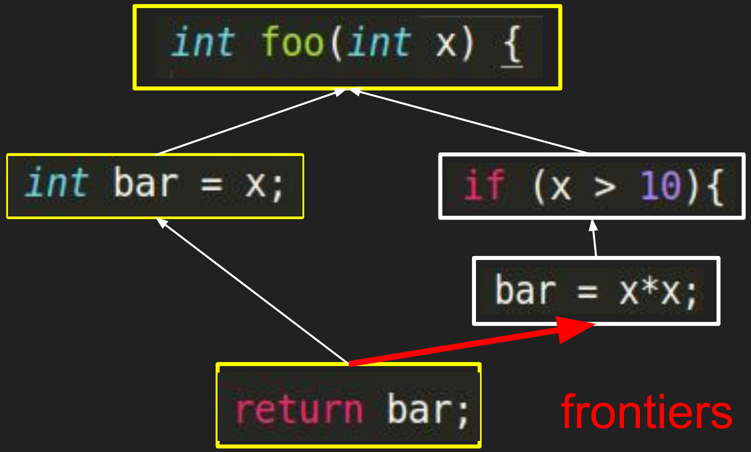
```
bar = x*x;
```

```
return bar;
```

frontiers

How to build a **Correctness Certificate**

Dependency Graph



```
int foo(int x) {
```

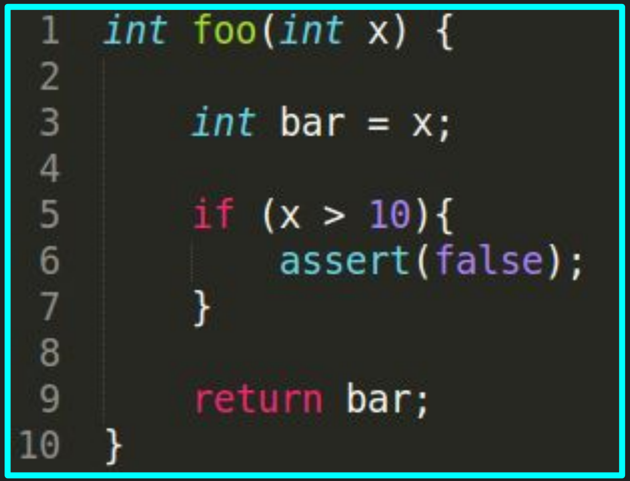
```
int bar = x;
```

```
if (x > 10){
```

```
bar = x*x;
```

```
return bar;
```

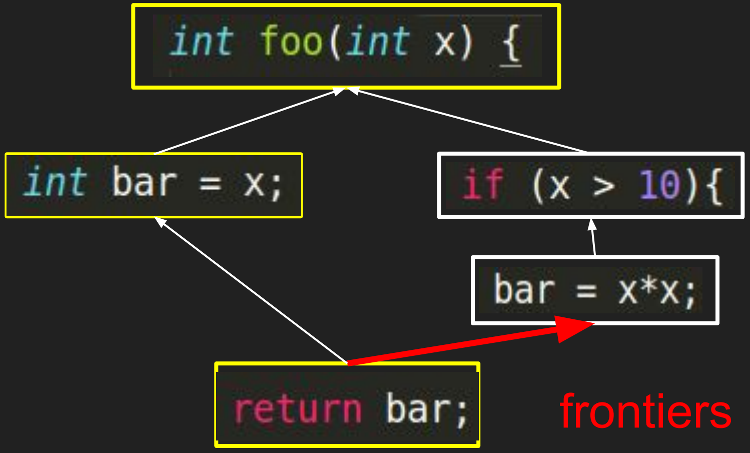
Correctness Certificate



```
int foo(int x) {

    int bar = x;

    if (x > 10){
        assert(false);
    }

    return bar;
}
```

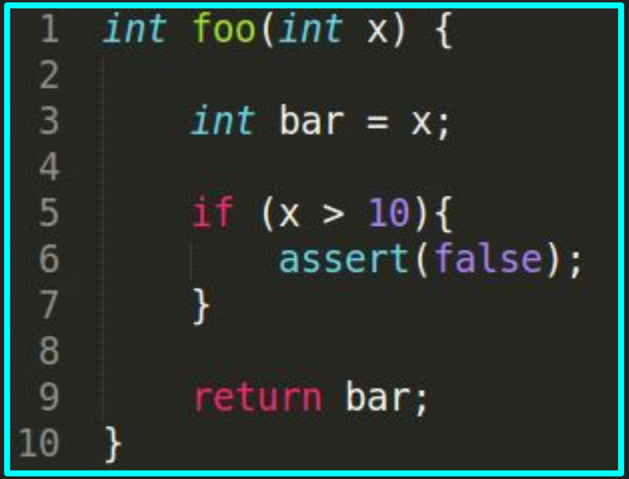
How to build a **Correctness Certificate**

Dependency Graph



frontiers

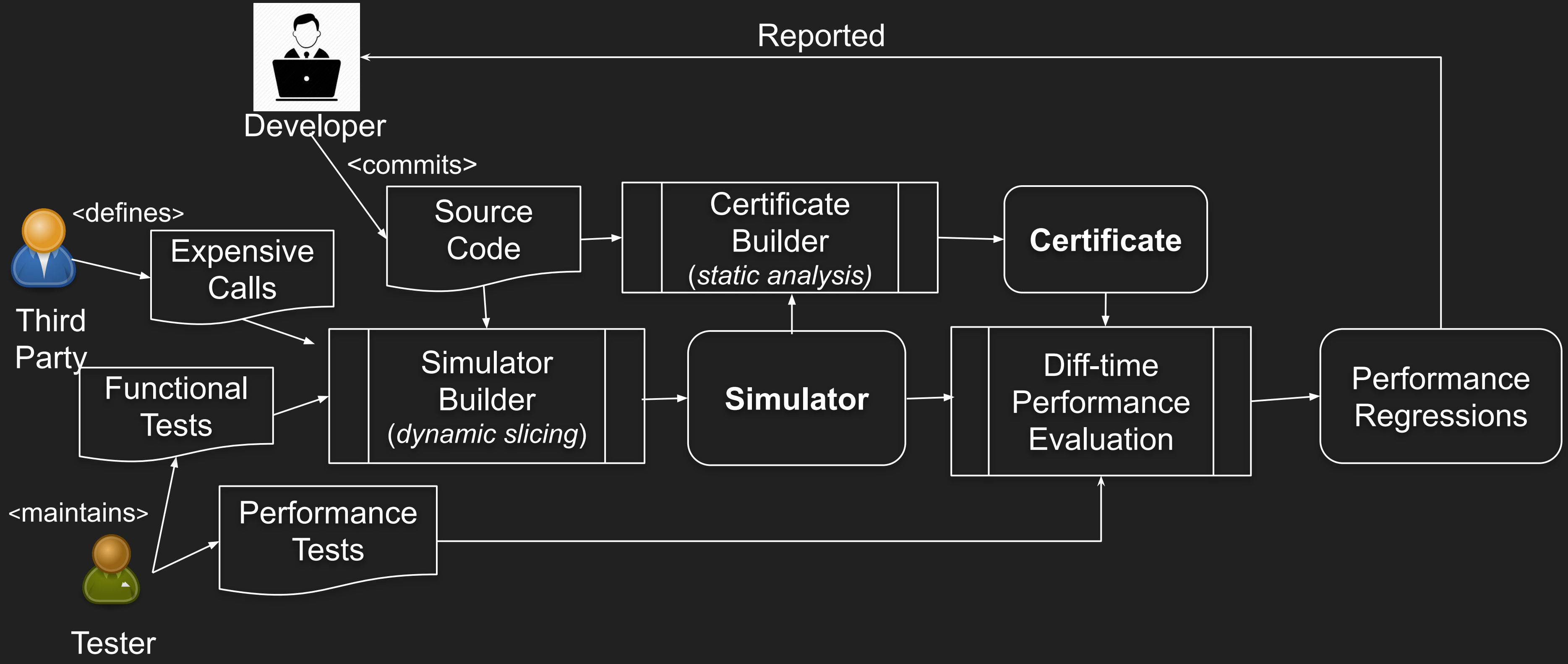
Correctness Certificate



```
int foo(int x) {

    int bar = x;

    if (x > 10){
        assert(false);
    }

    return bar;
}
```

Accelerated by “Failure-directed program trimming,” K. Ferles, V. Wustholz, M. Christakis, and I. Dillig, in FSE’17.

Technique Overview with Correctness Certificate



Emerging Results

Subjects	Simulator and Certificate	Results
A real world perf. regression (MySQL #46011) 2.2 Mloc		
An artificial perf. regression (Olden BH) 2 Kloc		

Emerging Results

Subjects	Simulator and Certificate	Results
A real world perf. regression (MySQL #46011) 2.2 Mloc	Observationally sliced a few lines, no frontiers were found statically.	
An artificial perf. regression (Olden BH) 2 Kloc	Traditional dynamic slice deleted 90% lines, non trivial certificate program.	

Emerging Results

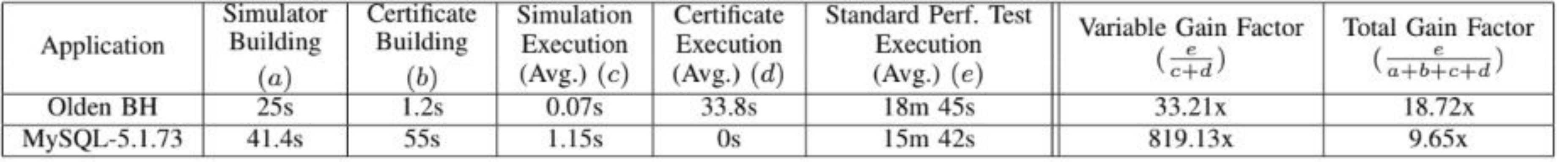
Subjects	Simulator and Certificate	Results
A real world perf. regression (MySQL #46011) 2.2 Mloc	Observationally sliced a few lines, no frontiers were found statically.	Regression detected with a total gain of 9.65x. (15m ⇝ 1.5m)
An artificial perf. regression (Olden BH) 2 Kloc	Traditional dynamic slice deleted 90% lines, non trivial certificate program.	Regression detected with a total gain of 18.72x (19m ⇝ 1m)

Backlog

- Study technique precision and recall
- Explore techniques to suggest expensive calls
- Cope with program non-determinism
- Study probabilistic approaches
- Fully automate an instance of the technique

Questions

Appendix. Emerging Results (Full table)



Application	Simulator Building (a)	Certificate Building (b)	Simulation Execution (Avg.) (c)	Certificate Execution (Avg.) (d)	Standard Perf. Test Execution (Avg.) (e)	Variable Gain Factor $(\frac{e}{c+d})$	Total Gain Factor $(\frac{e}{a+b+c+d})$
Olden BH	25s	1.2s	0.07s	33.8s	18m 45s	33.21x	18.72x
MySQL-5.1.73	41.4s	55s	1.15s	0s	15m 42s	819.13x	9.65x