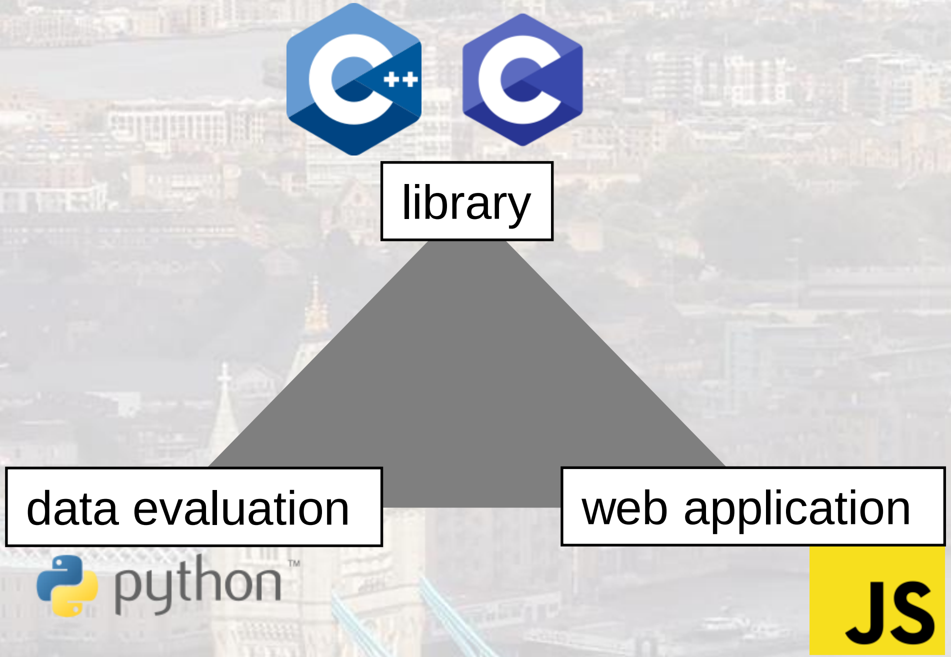
CC-BY: Ray in Manila

Engineering

→ combining best-suitable materials!

Software engineering?
→ cross-language interoperability!



library

data evaluation

web application

Extending Sulong for cross-language interoperability between C++/Swift and Java, JavaScript or Python

Christoph Pichler, Johannes Kepler University Linz (Austria)

Supervised by **Roland Schatz**

Extending Sulong for cross-language interoperability between C++/Swift and Java, JavaScript or Python

Christoph Pichler, Johannes Kepler University Linz (Austria)

Supervised by **Roland Schatz**

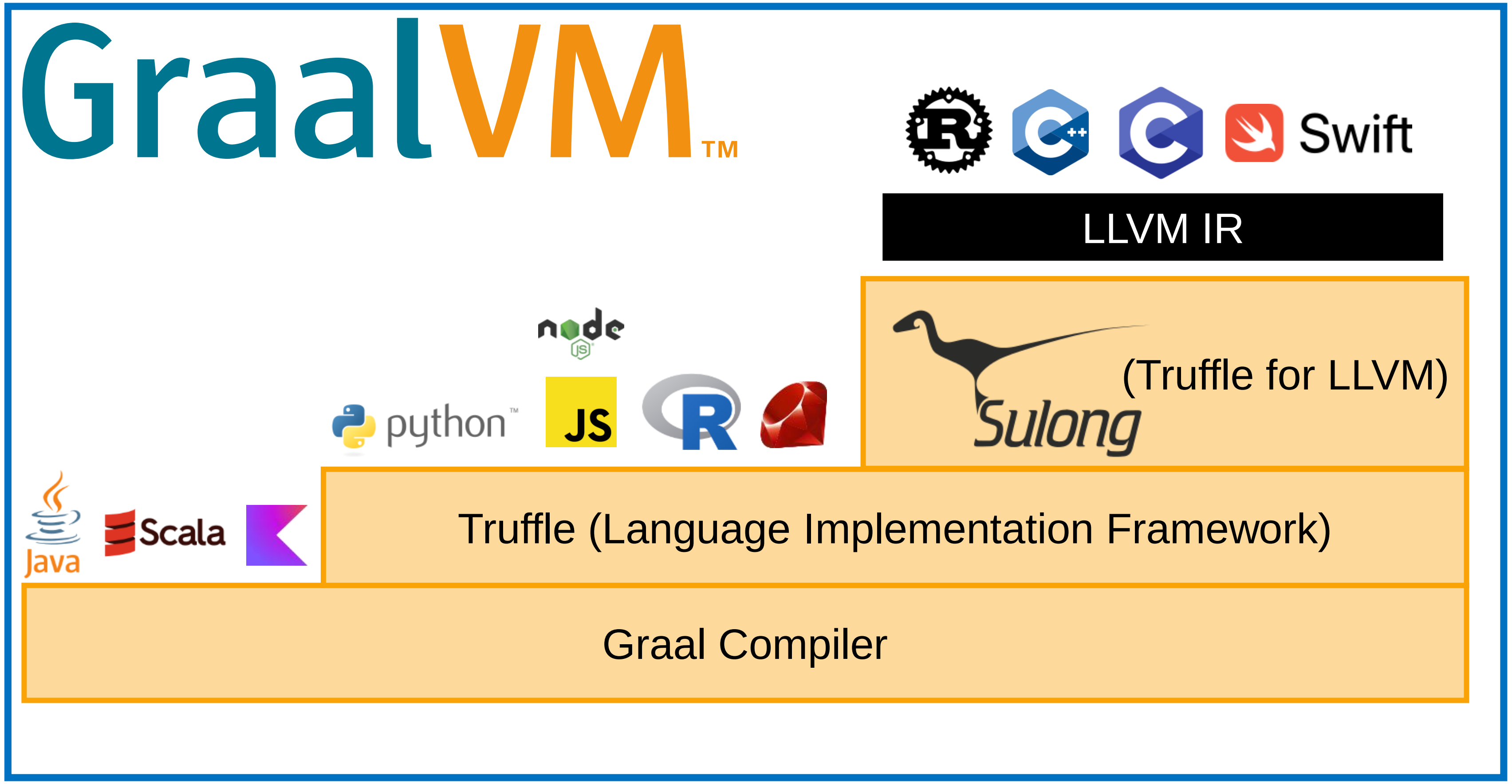
Extending Sulong for object-oriented cross-language interoperability

Christoph Pichler, Johannes Kepler University Linz (Austria)

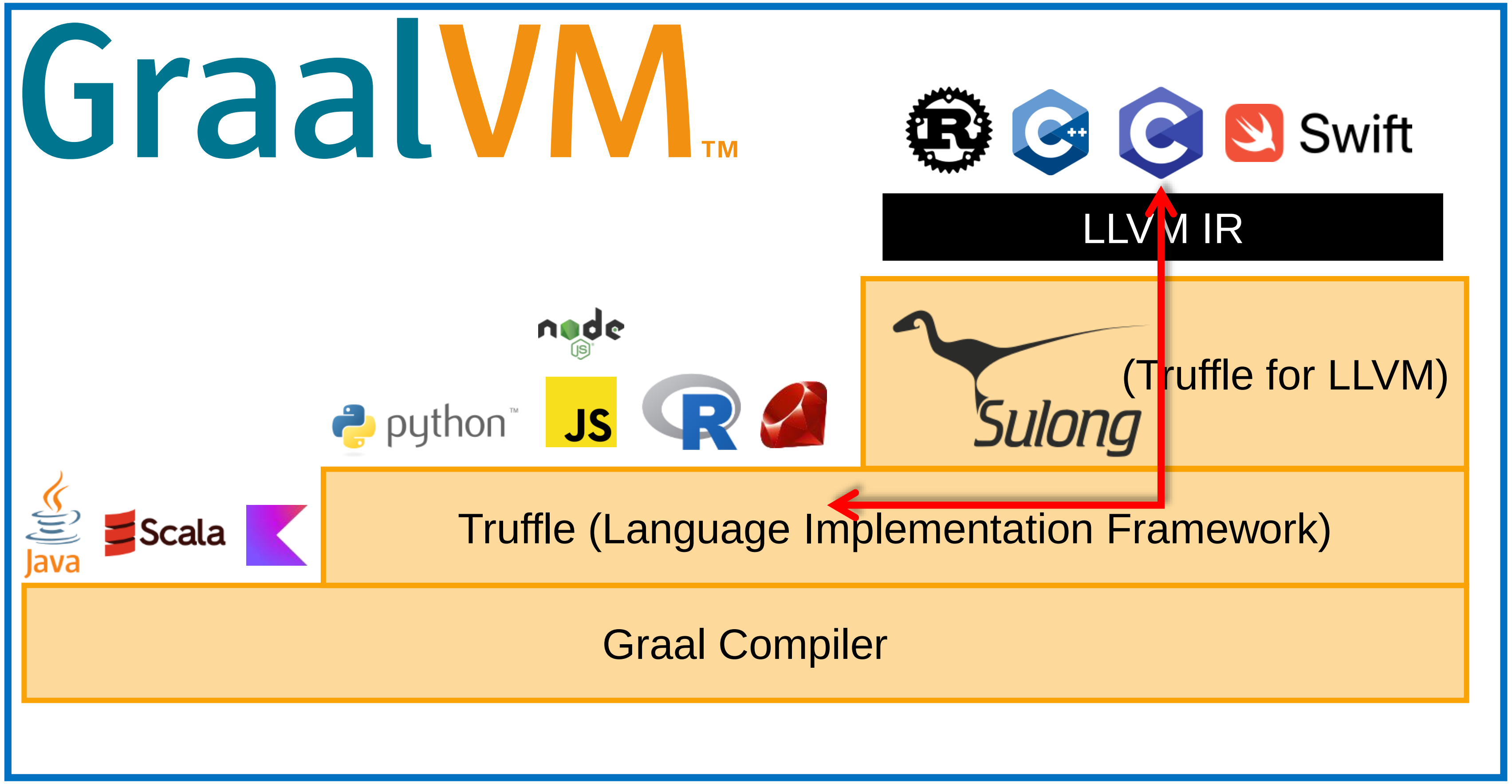
Supervised by **Roland Schatz**

"lli of GraalVM"

GraalVM and Sulong



Existing interoperability

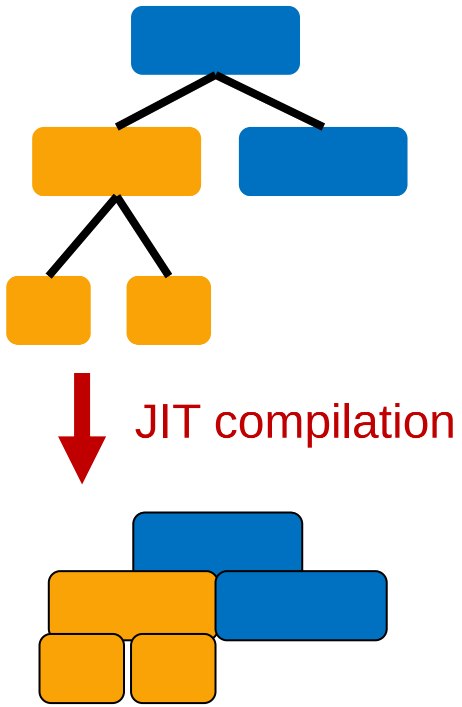


LLVM IR

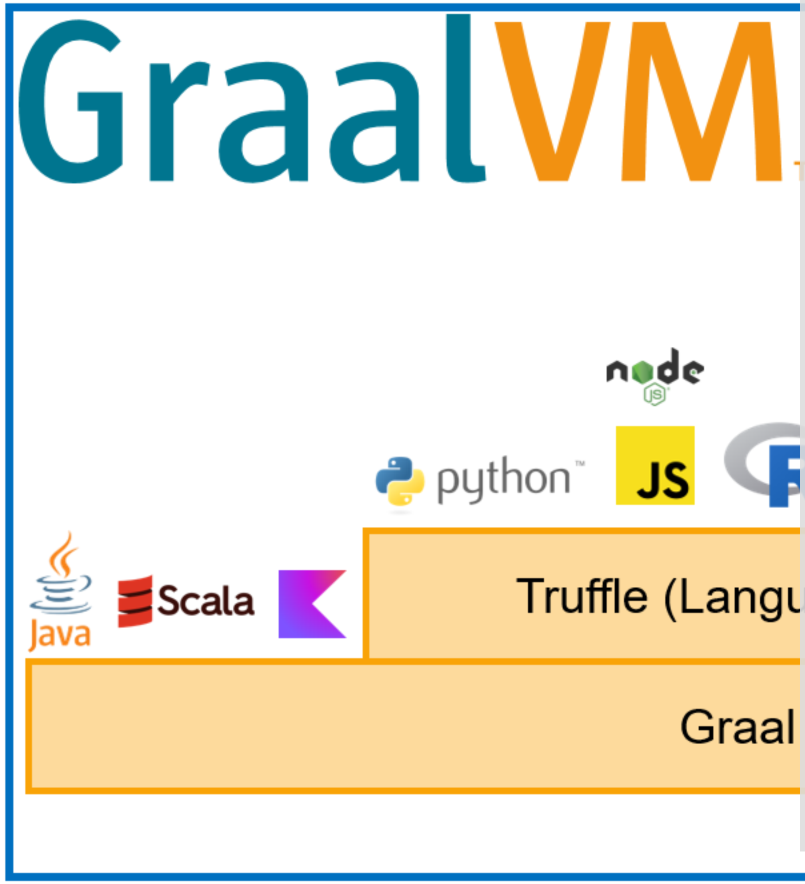
(Truffle for LLVM)

Truffle (Language Implementation Framework)

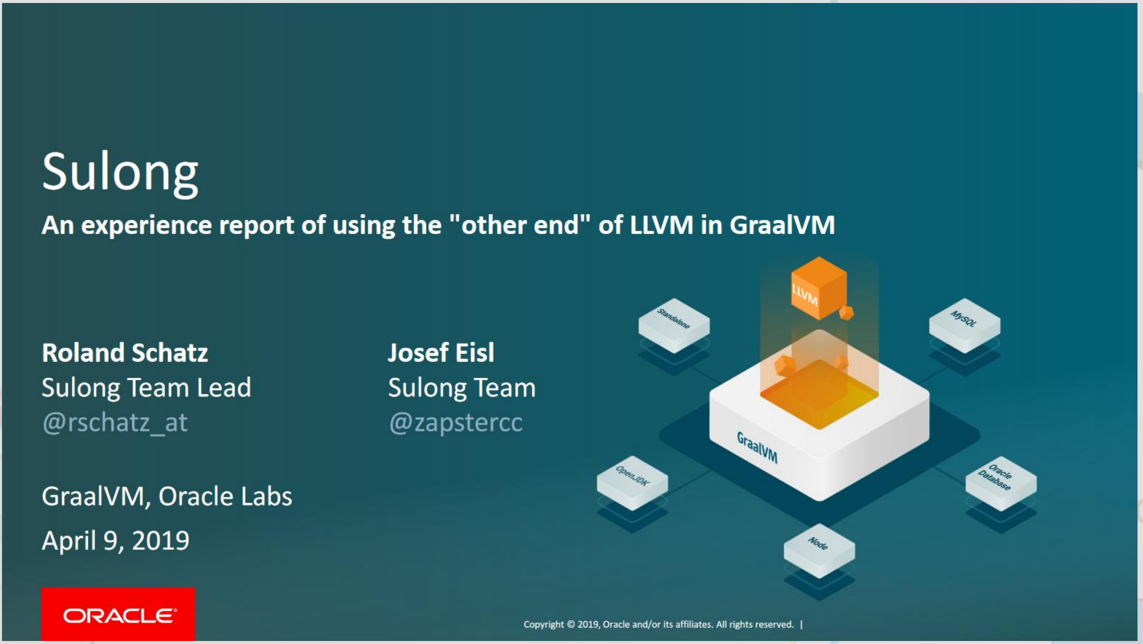
Graal Compiler


JIT compilation

Existing interoperability



→ LLVM Developers‘ meeting 2019!



Sulong

An experience report of using the "other end" of LLVM in GraalVM

Roland Schatz
Sulong Team Lead
@rschatz_at

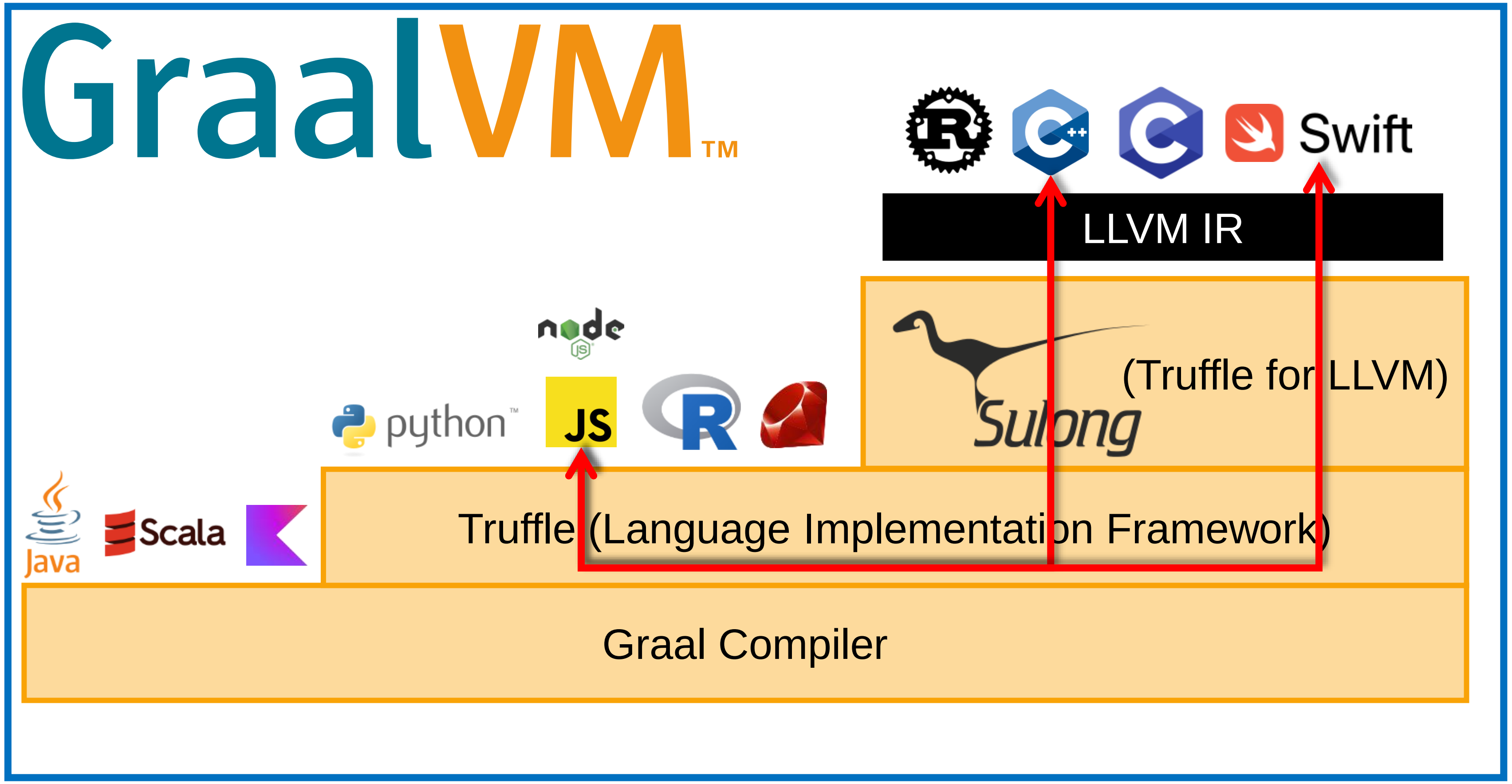
Josef Eisl
Sulong Team
@zapstercc

GraalVM, Oracle Labs
April 9, 2019

Copyright © 2019, Oracle and/or its affiliates. All rights reserved. |

https://llvm.org/devmtg/2019-04/talks.html#Talk_13

Extending Sulong‘s interoperability for object-oriented languages



The following examples take place along the red axes!

Sample extension: Dynamic binding [1]

Example.js

```
llvmFile = eval("Example.swift")
p = llvmFile.createParent()

p.square()
```

GraalVM JavaScript

invoke
Example.swift, p, "square"

Example.swift

```
class Parent {
  func
  square(n: Int) -> Int {…}
}
```

```
class Child: Parent {
  override func
  square(n: Int) -> Int {…}
}
```

LLVM IR

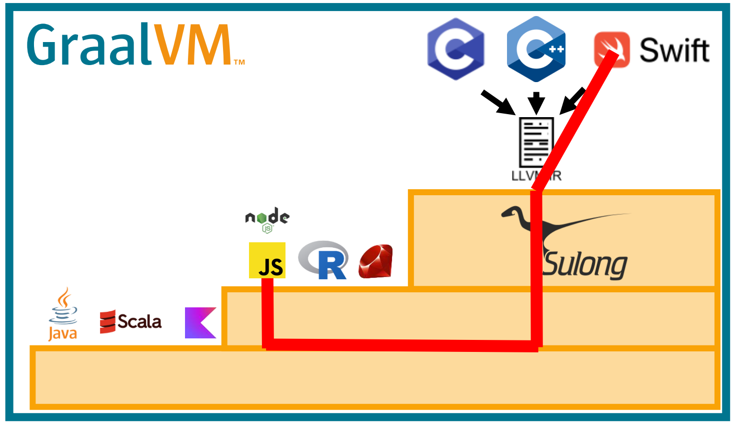
Example.so

```
%func $s6ParentC6squareSiyF {…}
%func $s5ChildC6squareSiyF {…}
```

*invoke Parent::square()**?***
*invoke Child:: square()**?***

invoke
p, "square"

Truffle Framework



Sample extension: Dynamic binding [2]

LLVM IR

Example.so

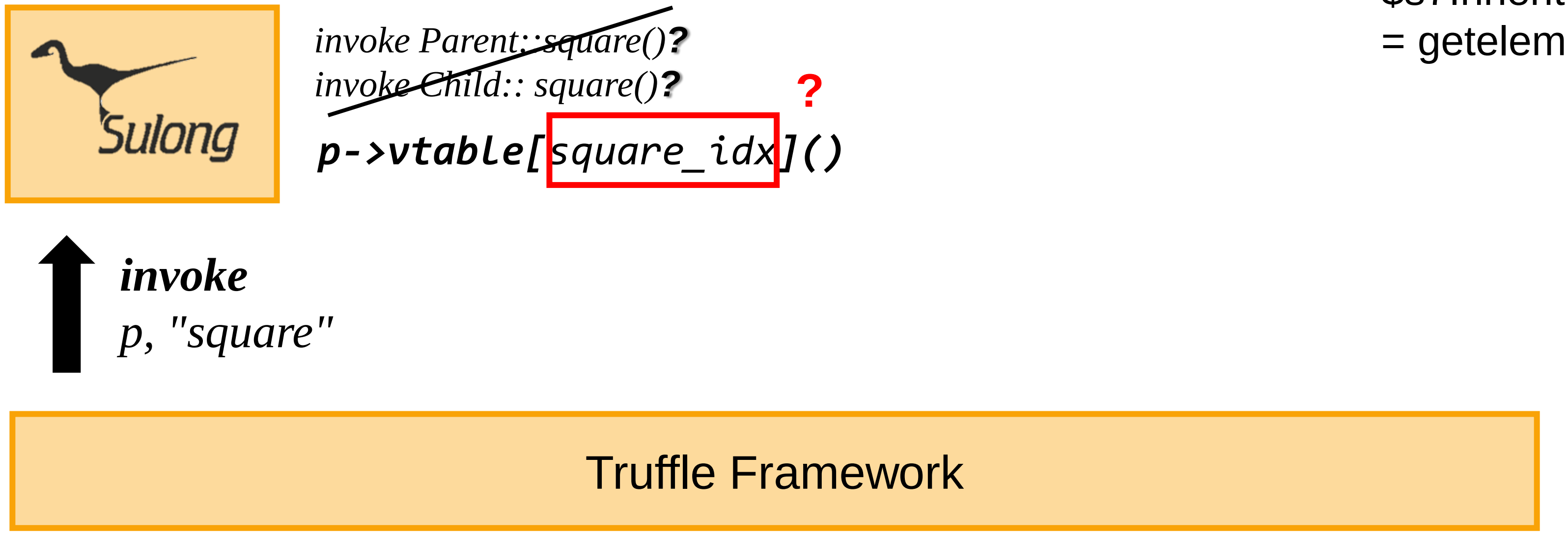
```
%func $s6ParentC6squareSiyF {…}
%func $s5ChildC6squareSiyF {…}
```

DEBUG INFORMATION

!DISubprogram "square":
  …,
  virtualIndex: 0, …)

GLOBALS

$s7Inherit6ParentC6square1nS2i_tF
= getelementptr (baseObj, [0, 13])



~~invoke Parent::square()?~~

~~invoke Child:: square()?~~

?

p->vtable[square_idx]()

invoke

p, "square"

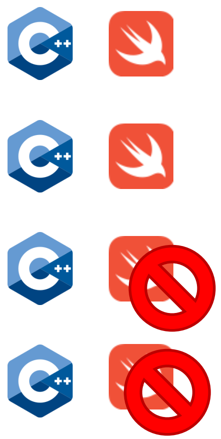
Truffle Framework

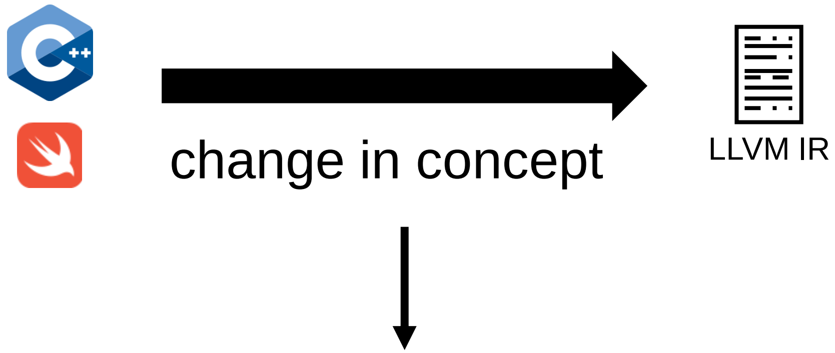
Live Demo

Files and step-by-step instructions can be found at
https://github.com/pichristoph/eurollvm2022

Higher-level semantics across languages!

- Name mangling
- **Dynamic binding**
- Field access
- Exception flow





Modifications necessary!

Open Source - ~~Do~~ TRY It Yourself!

GraalVM™

- graalvm.org
- github.com/oracle/graal

Concerning this talk:

- Slides, live demo files, step-by-step instructions, …: https://github.com/pichristoph/eurollvm2022

Related Talks at EuroLLVM 2019:

- Sulong and its interoperability for C-compiled LLVM (Roland Schatz, Josef Eisl): Sulong: An experience report of using the "other end" of LLVM in GraalVM.
- Sulong's debugging tools (Jacob Kreindl): LLVM IR in GraalVM: Multi-Level, Polyglot Debugging with Sulong

CC-BY: Ray in Manila