Swift Bindings for LLVM

Egor Zhdan

LLVM Developer Meeting | Apple, Inc. | Nov 2022

Swift/C++ Interoperability

Swift is a memory-safe language with functional and declarative features

Swift/C++ interoperability is an ongoing effort to make Swift and C++ interoperate in an ergonomic and performant way

Vision: Idiomatic Swift Interface for LLVM

Vision: Idiomatic Swift Interface for LLVM

Make LLVM APIs convenient to use from Swift

Vision: Idiomatic Swift Interface for LLVM

Make LLVM APIs convenient to use from Swift

- Manipulate LLVM containers through Swift collection methods

Vision: Idiomatic Swift Interface for LLVM

Make LLVM APIs convenient to use from Swift

- Manipulate LLVM containers through Swift collection methods
- Leverage Swift’s automatic reference counting, DSL features

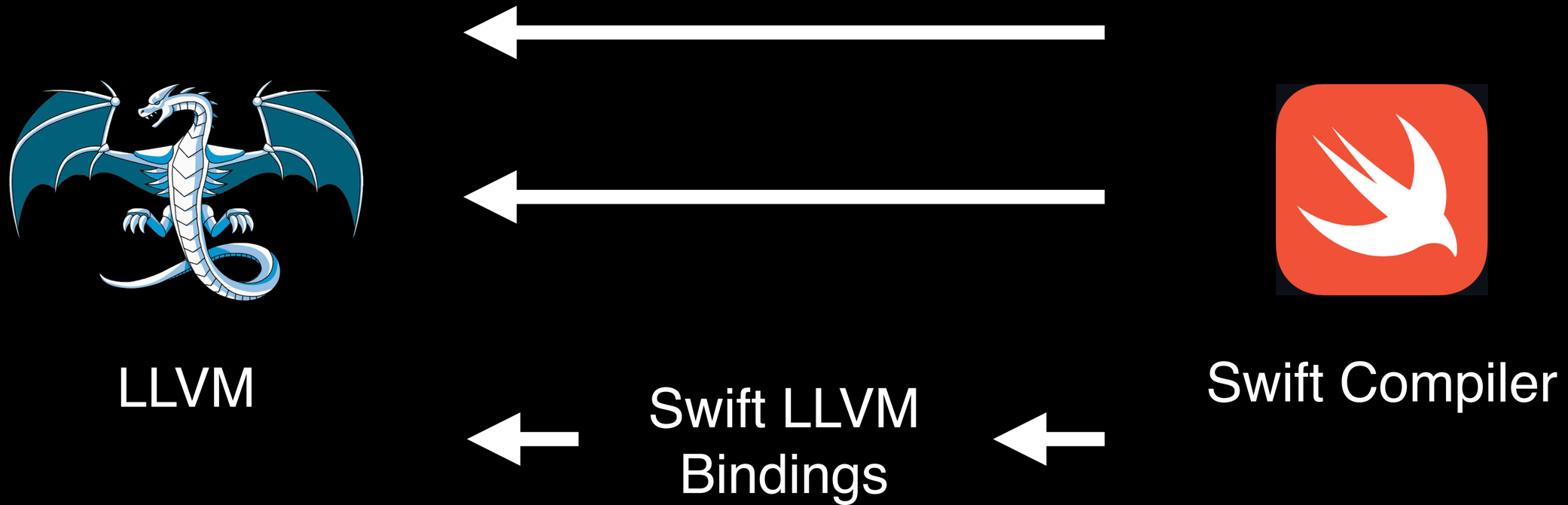
Vision: Idiomatic Swift Interface for LLVM

Make LLVM APIs convenient to use from Swift

- Manipulate LLVM containers through Swift collection methods
- Leverage Swift's automatic reference counting, DSL features

Opportunity to rethink LLVM APIs for a functional, declarative paradigm

Using Bindings in the Swift Compiler



Writing Collection Transformations in Swift

```
// Mapping arguments to their parameter indices
func argumentsToParamIndices(call: llvm.CallBase) {
    Dictionary(
        grouping: call.args().enumerated(),
        by: { $0.1 }
    ).mapValues { $0.0 }
}

// Input:
call @compute(%x, %y, %x)

// Output:
[%x : [0, 2], %y : [1]]
```

Writing Collection Transformations in Swift

```
// Mapping arguments to their parameter indices
func argumentsToParamIndices(call: llvm.CallBase) {
    Dictionary(
        grouping: call.args().enumerated(),
        by: { $0.1 }
    ).mapValues { $0.0 }
}

// Input:
call @compute(%x, %y, %x)

// Output:
[%x : [0, 2], %y : [1]]
```

Writing Collection Transformations in Swift

```
// Mapping arguments to their parameter indices
func argumentsToParamIndices(call: llvm.CallBase) {
    Dictionary(
        grouping: call.args().enumerated(),
        by: { $0.1 }
    ).mapValues { $0.0 }
}

// Input:
call @compute(%x, %y, %x)

// Output:
[%x : [0, 2], %y : [1]]
```

Writing Collection Transformations in Swift

```
// Mapping arguments to their parameter indices
func argumentsToParamIndices(call: llvm.CallBase) {
    Dictionary(
        grouping: call.args().enumerated(),
        by: { $0.1 }
    ).mapValues { $0.0 }
}

// Input:
call @compute(%x, %y, %x)

// Output:
[%x : [0, 2], %y : [1]]
```

Writing Collection Transformations in Swift

```
// Mapping arguments to their parameter indices
func argumentsToParamIndices(call: llvm.CallBase) {
    Dictionary(
        grouping: call.args().enumerated(),
        by: { $0.1 }
    ).mapValues { $0.0 }
}

// Input:
call @compute(%x, %y, %x)

// Output:
[%x : [0, 2], %y : [1]]
```

Bindings Implementation

Conform LLVM collections to Swift Collection protocols

```
// C++
typedef llvm.SmallVector<int> SmallVectorOfInt;


// Swift
extension SmallVectorOfInt : RandomAccessCollection {
}
```

map, filter, reduce, etc. are available for C++ types imported into Swift

```
let args = call.args()
let argNames = args.map { $0.getName() }
```

Bindings Implementation

Conform LLVM collections to Swift Collection protocols

```
// C++
typedef llvm.SmallVector<int> SmallVectorOfInt;


// Swift
extension SmallVectorOfInt : RandomAccessCollection {
}
```

Automatic

map, filter, reduce, etc. are available for C++ types imported into Swift

```
let args = call.args()
let argNames = args.map { $0.getName() }
```

Join Us in Swift/C++ Interoperability and LLVM Bindings Efforts!

Swift/C++ interoperability is an open-source effort with large community participation.

swift.org/cxx-interop-workgroup

Swift LLVM Bindings is an open-source project.

github.com/apple/swift-llvm-bindings

We would be thrilled to talk to you if you would like to learn more!

Zoe Carver

Alex Lorenz

Egor Zhdan