Introducing the Swift Bridge

For Frida and r2frida

Abdelrahman Eid (@hot3eed) at NowSecure

Roadmap

- Words of Wisdom
- Background
- Metadata
- Calling Convention
- Instrumentation
- The Future

“Unfortunately, the most common metaphor for software development is building construction. [...] Rather than construction, software is more like gardening--it is more organic than concrete.[...] You constantly monitor the health of the garden, and make adjustments (to the soil, the plants, the layout) as needed.”

The Pragmatic Programmer

Background

Dynamic vs. Static Dispatch

Static: the compiler selects the (polymorphic) implementation of a method during compile time, e.g. Swift

Dynamic: the compiler defers the selection of a method implementation to a runtime system, e.g. Objective-C

```
ldr x0, [self]
ldr x1, [someSelectorWithArg:]
ldr x2, [first_arg]
blr objc_msgSend
```

```
ldr x20, [self]
ldr x0, [first_arg]
blr some_method
```

Background
Objective-C Runtime

```
objc_getClassList()
objc_lookUpClass()
objc_copyProtocolList()
class_addMethod()
method_getNumberOfArguments()
method_setImplementation()
ivar_getName()
ivar_getOffset()
objc_retain()
…
```

The Swift runtime is nowhere near dynamic… or is it?

Background

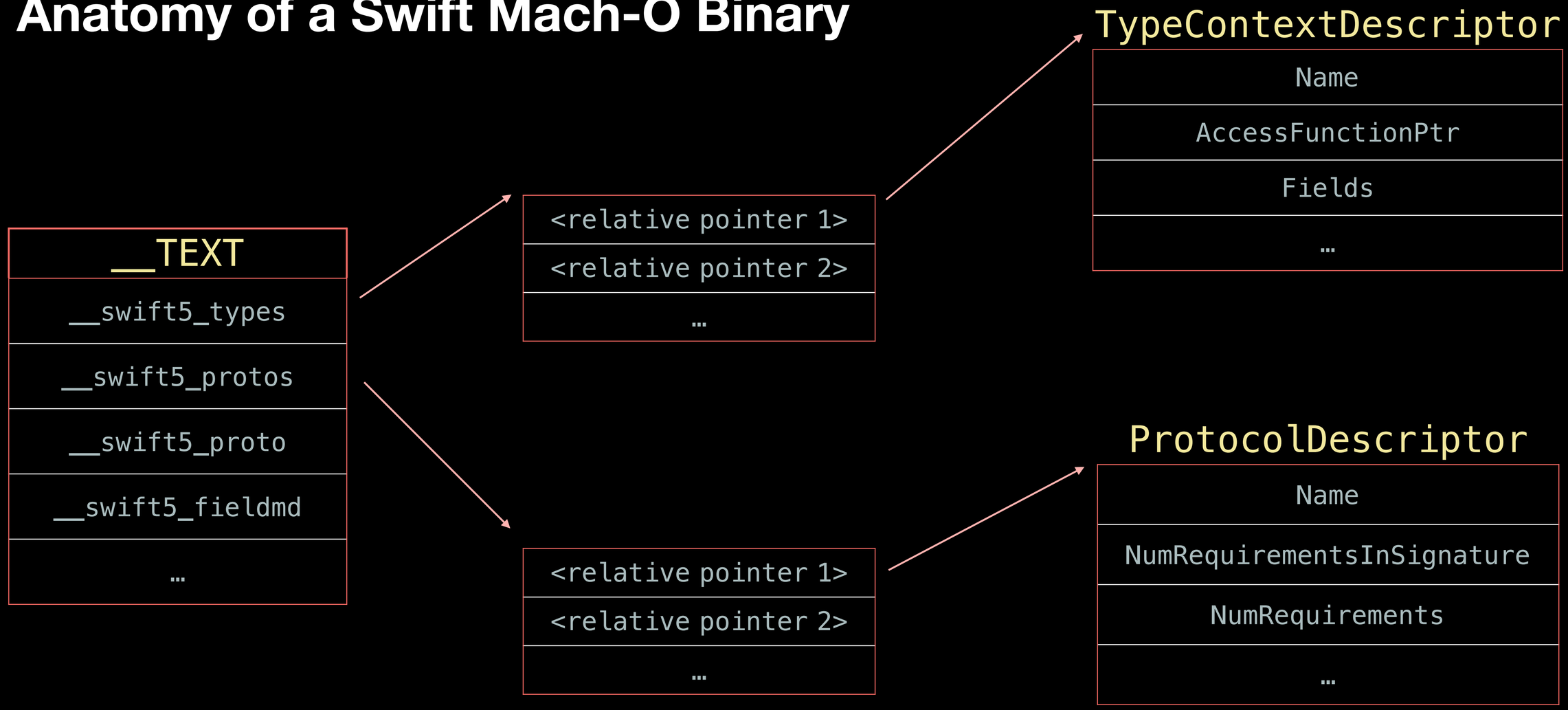
Swift Runtime

```
swift_dynamicCastUnknownClass()
swift_getDynamicType()
swift_getObjectType()
swift_conformsToProtocol()
swift_getEnumTagSinglePayloadGeneric()
swift_getEnumCaseMultiPayload()
swift_tryRetain()
swift_copyPOD()
swift_once()
...
```

It is not.

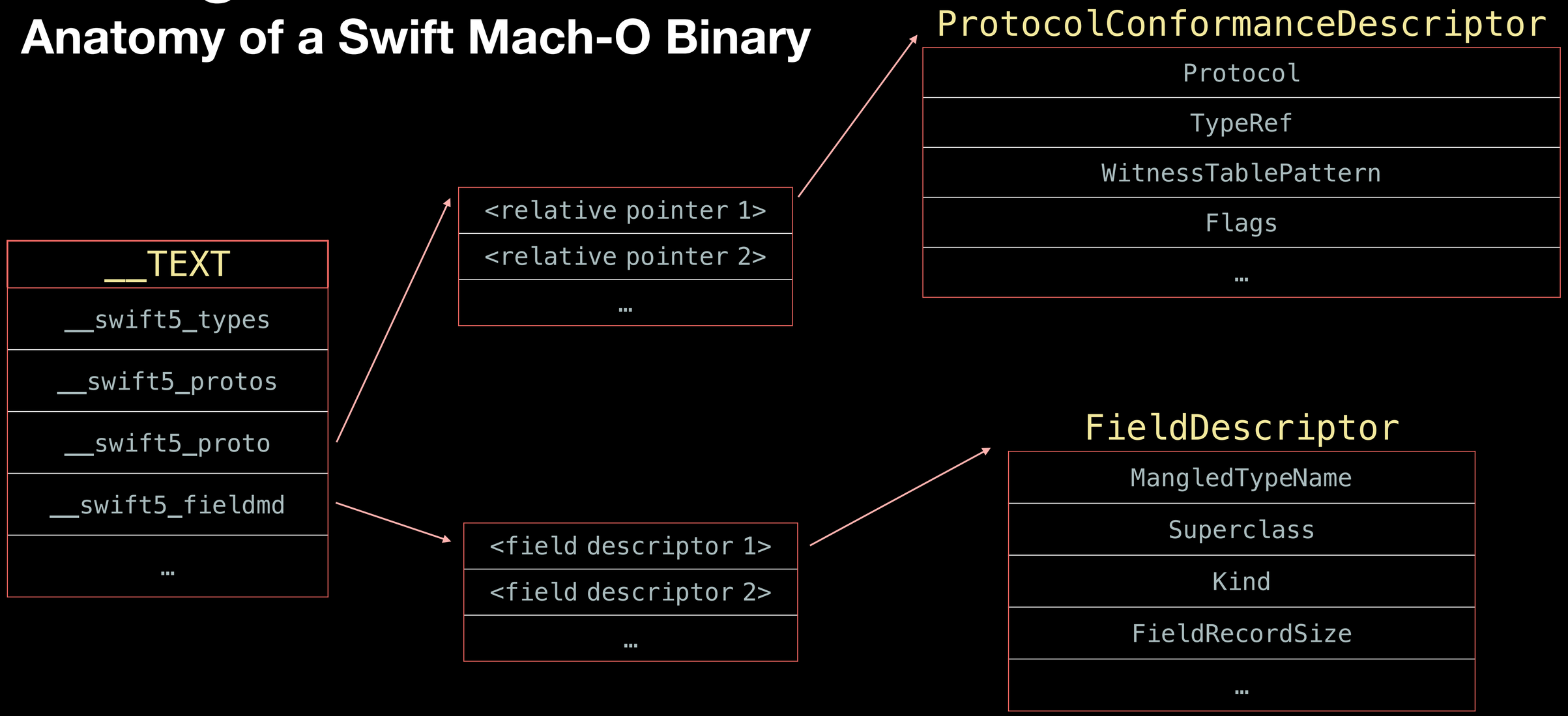
Background

Anatomy of a Swift Mach-O Binary



Background

Anatomy of a Swift Mach-O Binary



Metadata

Nominal Descriptors

Data structures that contain a good amount of metadata about a type.

The building block for type metadata in the compiler is `TargetTypeContextDescriptor`, defined in `include/swift/ABI/Metadata.h`

```
template <typename Runtime>
class TargetTypeContextDescriptor
    : public TargetContextDescriptor<Runtime> {
public:
  TargetRelativeDirectPointer<Runtime, const char, /*nullable*/ false> Name;
  TargetRelativeDirectPointer<Runtime, MetadataResponse(...),
                              /*Nullable*/ true> AccessFunctionPtr;

  TargetRelativeDirectPointer<Runtime, const reflection::FieldDescriptor,
                              /*nullable*/ true> Fields;

  bool isReflectable() const { return (bool)Fields; }
```

Metadata

Nominal Descriptors

```
class TargetEnumDescriptor final
    : public TargetValueTypeDescriptor<Runtime>,
  uint32_t getNumCases() const {
    return getNumPayloadCases() + NumEmptyCases;
  }
...


struct TypeLayout {
  ValueWitnessTypes::size size;
  ValueWitnessTypes::stride stride;
  ValueWitnessTypes::flags flags;
  ValueWitnessTypes::extraInhabitantCount extraInhabitantCount;
...
```

Metadata

Value Witness Table

A list of gadget methods used by the compiler to perform operations like “copy” and “assign”, *without having to know anything about its memory layout or ownership semantics.*
Comes to play with protocols

`include/Swift/ABI/ValueWitness.def:`

```
FUNCTION_VALUE_WITNESS(initializeBufferWithCopyOfBuffer,
                       InitializeBufferWithCopyOfBuffer,
                       MUTABLE_VALUE_TYPE,
                       (MUTABLE_BUFFER_TYPE, MUTABLE_BUFFER_TYPE, TYPE_TYPE))

FUNCTION_VALUE_WITNESS(getEnumTagSinglePayload,
                       GetEnumTagSinglePayload,
                       UINT_TYPE,
                       (IMMUTABLE_VALUE_TYPE, UINT_TYPE, TYPE_TYPE))
…

bool isValueInline() const {
  return flags.isInlineStorage();
}
```

Metadata

Bridge APIs

- Swift.modules
- Swift.classes
- Swift.structs
- Swift.enums
- Swift.protocols
- Swift.modules.<module name>.classes
  - etc.

Demo (Metadata + APIs + r2frida)

Calling Convention
Physical Lowering

- Named `swiftcall`
- Implemented by the back-end, LLVM
- Usable via the attributes in clang:
  - `__attribute__(swiftcall)`
  - `__attribute__(swift_indirect_result)`
  - `__attribute(swift_error_result)`
- Passes and return structs of size < 4 registers directly in registers
- On AARCH64, `x20` is used for the self pointer and `x21` is used for the error return pointer

Calling Convention

Physical Lowering: Our Implementation

Just-in-time compile adaptors to handle the translation from `SystemV` to `swiftcall`

```
const buffer = Memoy.alloc(adaptorSize);
Memory.patchCode(buffer, adaptorSize, (code) => {
  if (context !== undefined) {
    writePutLdrRegAddress("x20", context);
  }

  ...

  writer.putLdrRegAddress("x15", this.#returnBuffer);

  let i = 0, offset = 0;

  for (; offset < this.#returnBufferSize; i++, offset += 8) {
      const reg = `x${i}` as Arm64Register;
      writer.putStrRegRegOffset(reg, "x15", offset);
  }

  ...
});
```

Calling Convention

Semantic Lowering

Done by the compiler's front-end

Means figuring out whether a value is passed indirectly and its ownership conventions (among other things)

Some types are address-only because their actual type is not known at compile time, e.g. generic and protocol parameters

Some types need to be registered with the runtime at all times, i.e. in memory, such as those with a weak reference

The Value Witness Table includes info about indirectness of a type.

Calling Convention

Bridge APIs

- `Swift.NativeFunction(address, retType, argTypes[, context, error])`
- `new Swift.ProtocolComposition(…types[]);`

Demo (Calling Convention + APIs)

Instrumentation

Our Implementation

Parse function arguments and return values to retrieve their metadata by name

Leverage the fact that we have type metadata and are able to call Swift functions to create JS wrappers for Swift objects and values

Bind the Swift object’s getters, setters, and methods to the JS wrapper to be able to instrument it at runtime

Instrumentation

Bridge APIs

- `new Swift.Object(handle);`
- `new Swift.Struct(type, options);`
- `Swift.enums.<enum name>.<case name>;`
- `Swift.Interceptor.attach(target, callbacks);`

Demo (Instrumentation APIs)

The Future

v3.0.0

https://github.com/frida/frida-swift-bridge/projects/1