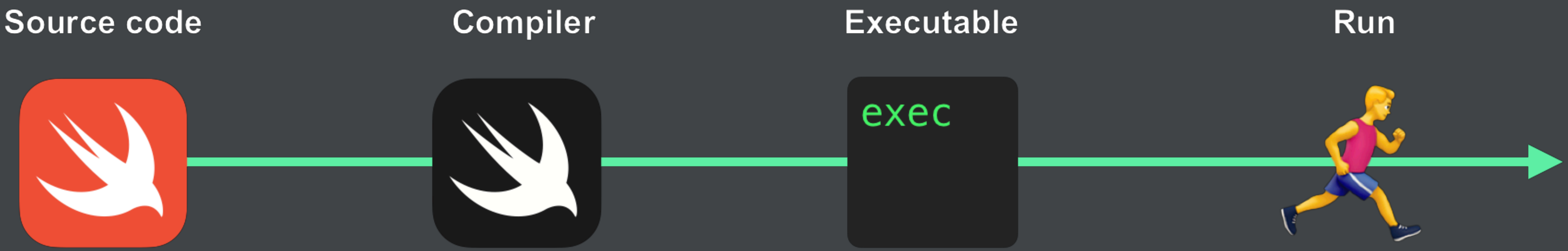
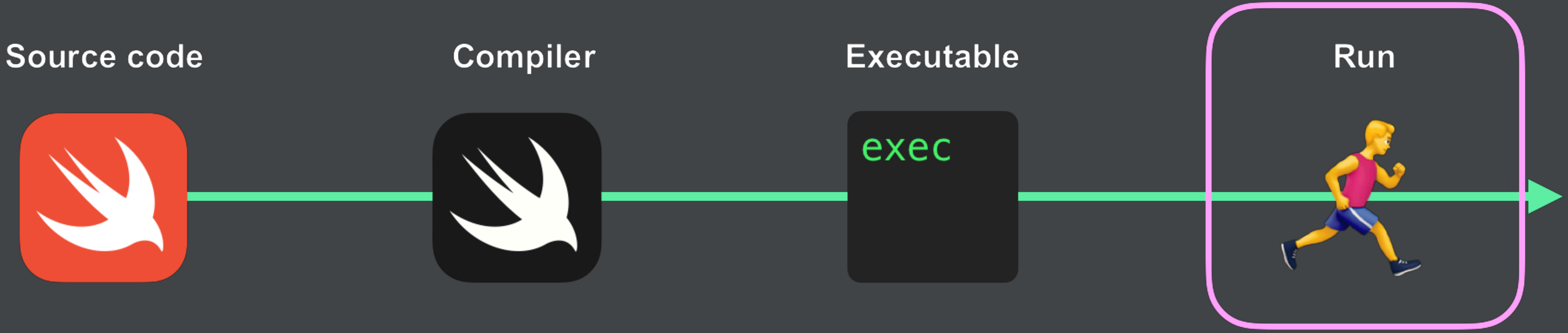
Swift Type Metadata

@kateinoigakukun
try! Swift 2019


Source code
Compiler
Executable
exec
Run


Source code
Compiler
Executable
exec
Run

```swift
let typeName = String(describing: Int.self)
```

```swift
extension UITableView {
    func register<Cell>(nibWithCellClass: Cell.Type) where Cell: UITableViewCell {
        let typeName = String(describing: Cell.self)
        let nib = UINib(nibName: typeName, bundle: Bundle.main)
        register(nib, forCellReuseIdentifier: typeName)
    }
}

tableView.register(nibWithCellClass: TweetCell.self)
```

Agenda

1. What is type metadata?
2. Explore `String(describing: Int.self)`
3. How to use metadata in Swift
4. Use cases in OSS

What is type metadata?

- Type information in Swift runtime
- Used in Swift internal dynamic behavior
- Metatype is pointer to metadata

```
let metatype: Int.Type = Int.self
```

```swift
extension String {
  public init<Subject: CustomStringConvertible>(describing instance: Subject) { ... }
  public init<Subject>(describing instance: Subject) { ... }
}
let typeName = String(describing: Int.self) // "Int"
```

```swift
extension Int.Type: CustomStringConvertible { // 🚫 Cannot extend a metatype 'Int.Type'
    var description: String {
        return "Int"
    }
}
```

SwiftCore

- Swift standard library
- Fundamental types and interfaces

SwiftRuntime

- Swift runtime library
- Dynamic behavior

stdlib/public/core/Mirror.swift

```swift
struct String {
  public init<Subject>(describing instance: Subject) {
    _print_unlocked(instance, &self)
  }
}
```

stdlib/public/core/Misc.swift

```
public func _typeName(_ type: Any.Type, qualified: Bool = true) -> String {
  let (stringPtr, count) = _getTypeName(type, qualified: qualified)
  return String._fromUTF8Repairing(
    UnsafeBufferPointer(start: stringPtr, count: count)).0
}

@_silgen_name("swift_getTypeName")
public func _getTypeName(_ type: Any.Type, qualified: Bool) -> (UnsafePointer<UInt8>, Int)
```

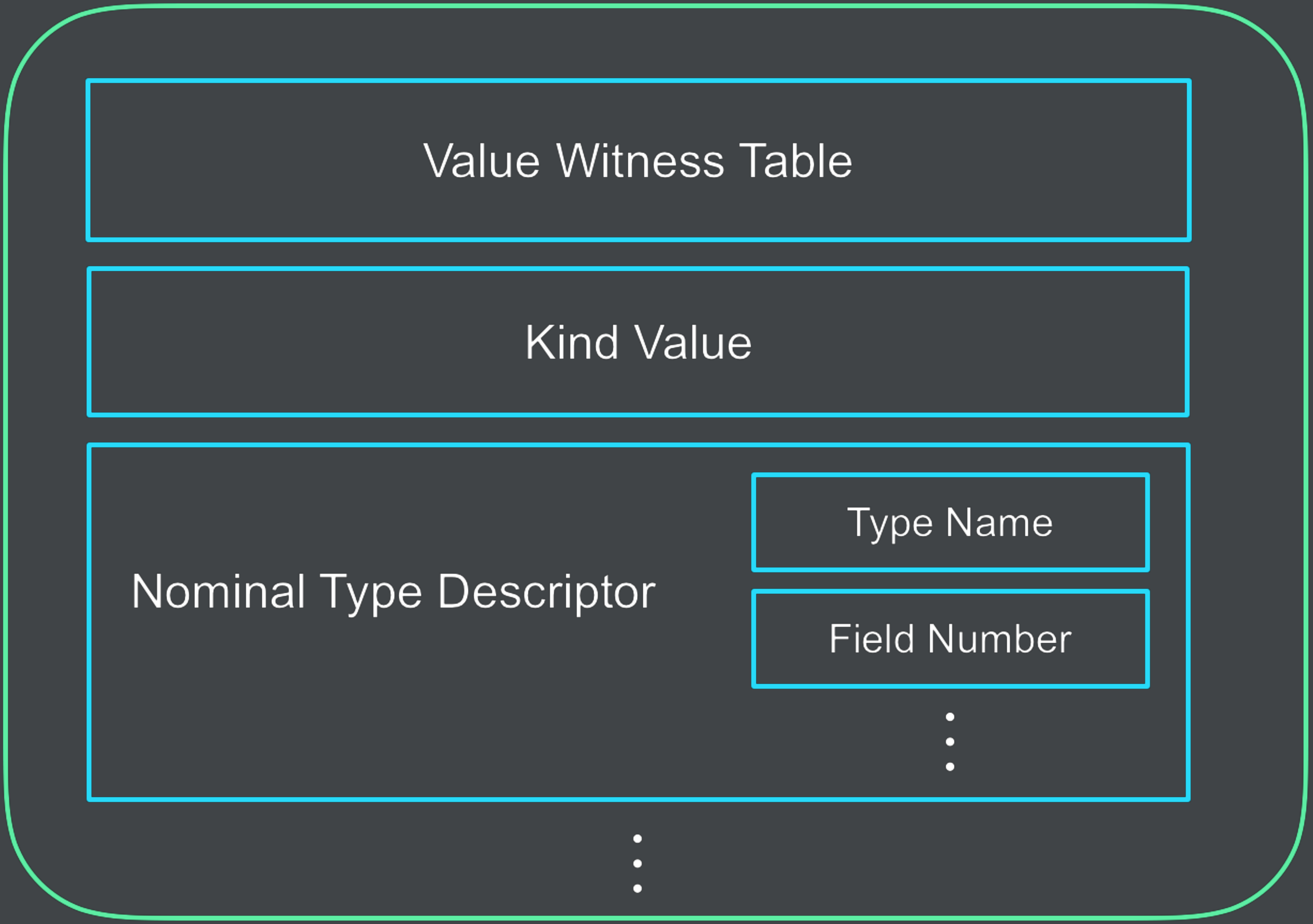
stdlib/public/core/Misc.swift

```
public func _typeName(_ type: Any.Type, qualified: Bool = true) -> String {
  let (stringPtr, count) = _getTypeName(type, qualified: qualified)
  return String._fromUTF8Repairing(
    UnsafeBufferPointer(start: stringPtr, count: count)).0
}

@_silgen_name("swift_getTypeName")
public func _getTypeName(_ type: Any.Type, qualified: Bool) -> (UnsafePointer<UInt8>, Int)
```

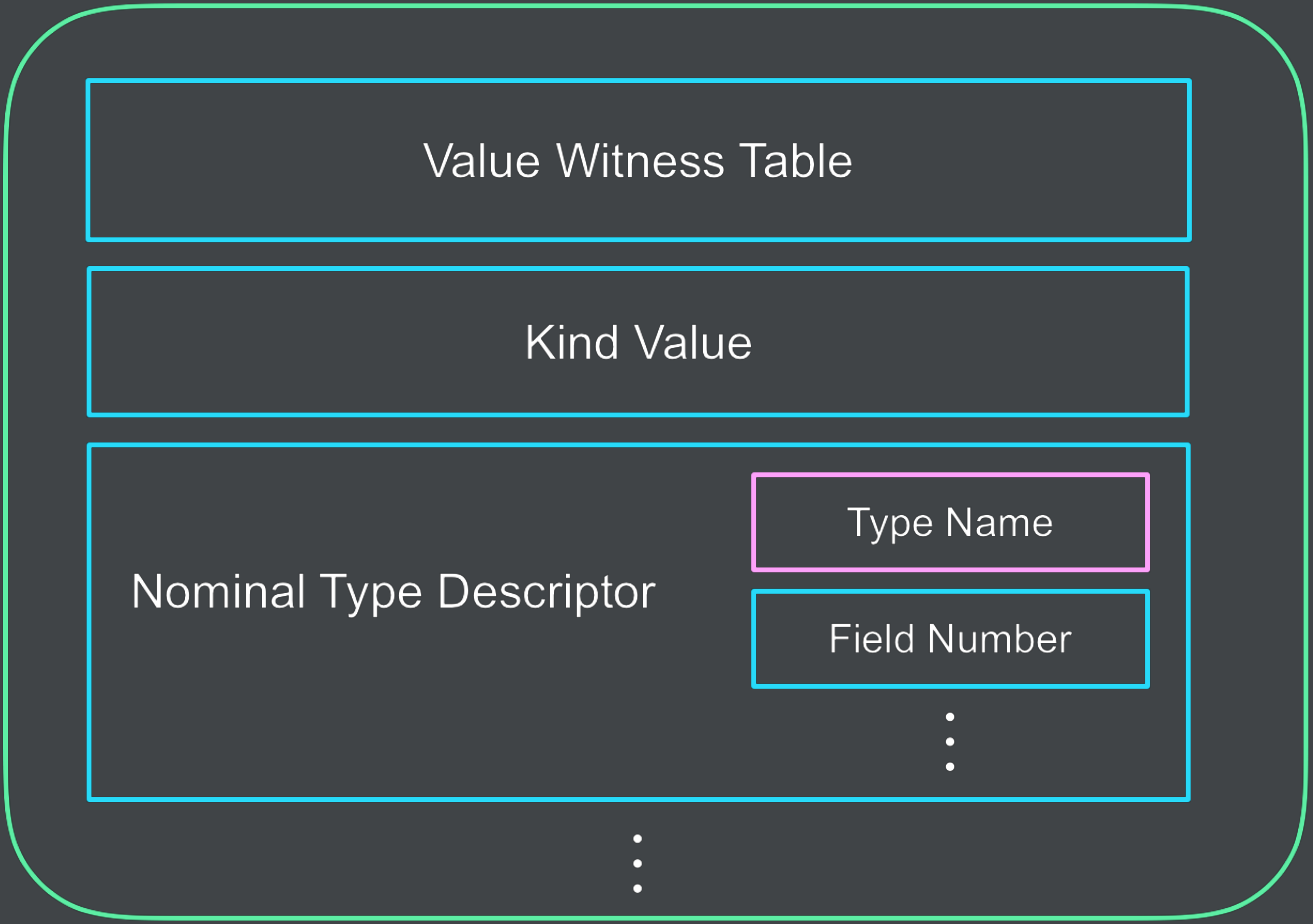
stdlib/public/core/Misc.swift

```swift
public func _typeName(_ type: Any.Type, qualified: Bool = true) -> String {
  let (stringPtr, count) = _getTypeName(type, qualified: qualified)
  return String._fromUTF8Repairing(
    UnsafeBufferPointer(start: stringPtr, count: count)).0
}

@_silgen_name("swift_getTypeName")
public func _getTypeName(_ type: Any.Type, qualified: Bool) -> (UnsafePointer<UInt8>, Int)
```

Metadata


Value Witness Table
Kind Value
Nominal Type Descriptor
Type Name
Field Number
⋮
⋮

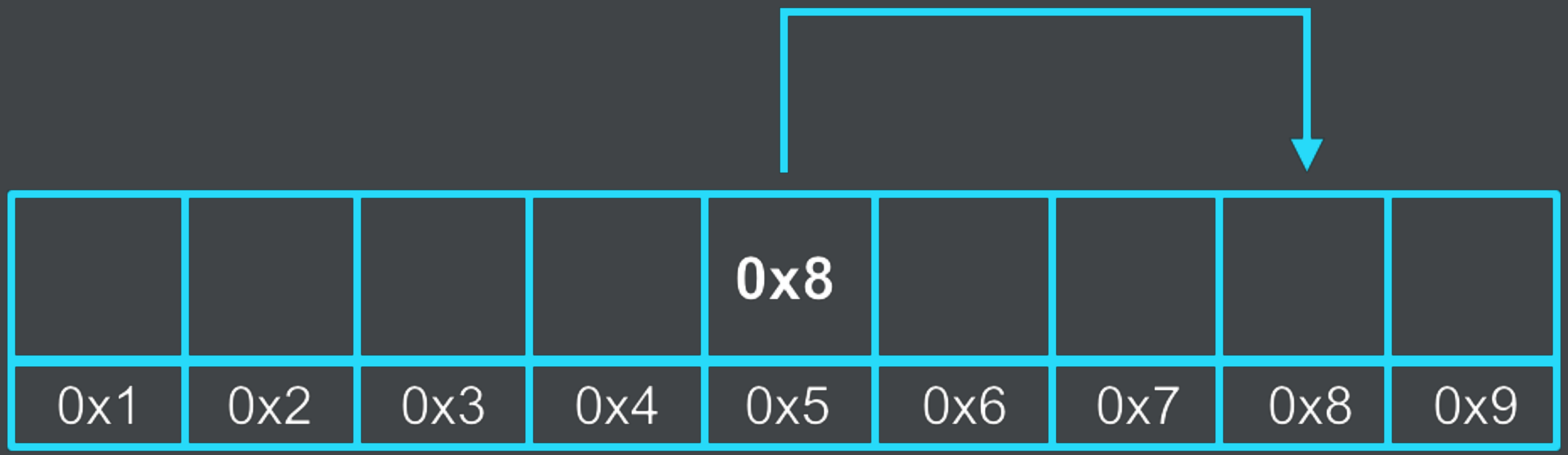
Metadata


Value Witness Table
Kind Value
Nominal Type Descriptor
Type Name
Field Number
⋮
⋮

```swift
struct StructMetadata {
    let kind: Int
    let typeDescriptor: UnsafePointer<StructTypeDescriptor>
}

struct StructTypeDescriptor {
    let flags: Int32
    let parent: Int32
    let name: RelativePointer<CChar>
}
```

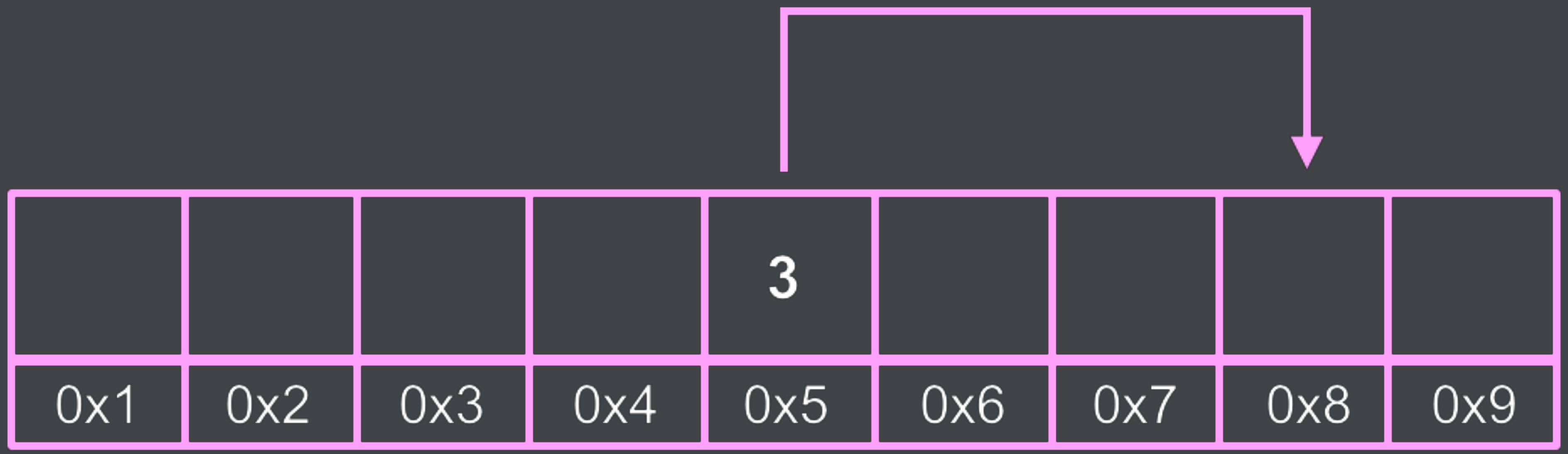
```swift
struct StructMetadata {
    let kind: Int
    let typeDescriptor: UnsafePointer<StructTypeDescriptor>
}

struct StructTypeDescriptor {
    let flags: Int32
    let parent: Int32
    let name: RelativePointer<CChar>
}
```

Absolute Pointer



Relative Pointer

0x5 + 3 = 0x8



```swift
func getTypeName<Subject>(of type: Subject.Type) -> String {
    let metadataPointer = unsafeBitCast(
        type, to: UnsafePointer<StructTypeMetadata>.self
    )
    let namePointer: UnsafePointer<CChar> = metadataPointer.pointee
                                   .typeDescriptor.pointee
                                   .name.advancedPointer()
    return String(cString: namePointer)
}
```

```swift
func getTypeName<Subject>(of type: Subject.Type) -> String {
    let metadataPointer = unsafeBitCast(
        type, to: UnsafePointer<StructTypeMetadata>.self
    )
    let namePointer: UnsafePointer<CChar> = metadataPointer.pointee
                        .typeDescriptor.pointee
                        .name.advancedPointer()
    return String(cString: namePointer)
}
```

```swift
func getTypeName<Subject>(of type: Subject.Type) -> String {
    let metadataPointer = unsafeBitCast(
        type, to: UnsafePointer<StructTypeMetadata>.self
    )
    let namePointer: UnsafePointer<CChar> = metadataPointer.pointee
                          .typeDescriptor.pointee
                          .name.advancedPointer()
    return String(cString: namePointer)
}
```

```swift
func getTypeName<Subject>(of type: Subject.Type) -> String {
    let metadataPointer = unsafeBitCast(
        type, to: UnsafePointer<StructTypeMetadata>.self
    )
    let namePointer: UnsafePointer<CChar> = metadataPointer.pointee
                         .typeDescriptor.pointee
                         .name.advancedPointer()
    return String(cString: namePointer)
}
```

```swift
func getTypeName<Subject>(of type: Subject.Type) -> String {
    let metadataPointer = unsafeBitCast(
        type, to: UnsafePointer<StructTypeMetadata>.self
    )
    let namePointer: UnsafePointer<CChar> = metadataPointer.pointee
                                .typeDescriptor.pointee
                                .name.advancedPointer()
    return String(cString: namePointer)
}
```

```swift
let typeName = getTypeName(of: Int.self) // "Int"
```

Use cases inside of Swift

- Allocate instance
    - Value Witness Table
- Dynamic method dispatch
    - VTable
- Reflection
    - Mirror API

😈

Method swizzling

Method swizzling

```
class Animal {
    func bar() { print("bar") }
    func foo() { print("foo") }
}

struct ClassMetadata {
    ...
    // VTable
    var barRef: FunctionRef
    var fooRef: FunctionRef
}
```

Method swizzling

```
class Animal {
    func bar() { print("bar") }
    func foo() { print("foo") }
}

struct ClassMetadata {
    ...
    // VTable
    var barRef: FunctionRef
    var fooRef: FunctionRef
}
```

```swift
let metadata = unsafeBitCast(
    Animal.self, to: UnsafeMutablePointer<ClassMetadata>.self
)

let bar = withUnsafeMutablePointer(to: &metadata.pointee.barRef) { $0 }
let foo = withUnsafeMutablePointer(to: &metadata.pointee.fooRef) { $0 }

bar.pointee = foo.pointee

let animal = Animal()
animal.bar() // foo
```

```swift
let metadata = unsafeBitCast(
    Animal.self, to: UnsafeMutablePointer<ClassMetadata>.self
)

let bar = withUnsafeMutablePointer(to: &metadata.pointee.barRef) { $0 }
let foo = withUnsafeMutablePointer(to: &metadata.pointee.fooRef) { $0 }

bar.pointee = foo.pointee

let animal = Animal()
animal.bar() // foo
```

```swift
let metadata = unsafeBitCast(
    Animal.self, to: UnsafeMutablePointer<ClassMetadata>.self
)

let bar = withUnsafeMutablePointer(to: &metadata.pointee.barRef) { $0 }
let foo = withUnsafeMutablePointer(to: &metadata.pointee.fooRef) { $0 }

bar.pointee = foo.pointee

let animal = Animal()
animal.bar() // foo
```

```
let metadata = unsafeBitCast(
    Animal.self, to: UnsafeMutablePointer<ClassMetadata>.self
)

let bar = withUnsafeMutablePointer(to: &metadata.pointee.barRef) { $0 }
let foo = withUnsafeMutablePointer(to: &metadata.pointee.fooRef) { $0 }

bar.pointee = foo.pointee

let animal = Animal()
animal.bar() // foo
```

```
let metadata = unsafeBitCast(
    Animal.self, to: UnsafeMutablePointer<ClassMetadata>.self
)

let bar = withUnsafeMutablePointer(to: &metadata.pointee.barRef) { $0 }
let foo = withUnsafeMutablePointer(to: &metadata.pointee.fooRef) { $0 }

bar.pointee = foo.pointee

let animal = Animal()
animal.bar() // foo
```

Use cases

- Zewo/Reflection
- wickwirew/Runtime
- alibaba/HandyJSON
- kateinoigakukun/StubKit

alibaba/HandyJSON

```
struct Item: HandyJSON {
    var name: String = ""
    var price: Double?
    var description: String?
}

if let item = Item.deserialize(from: jsonString) {
    // ...
}
```

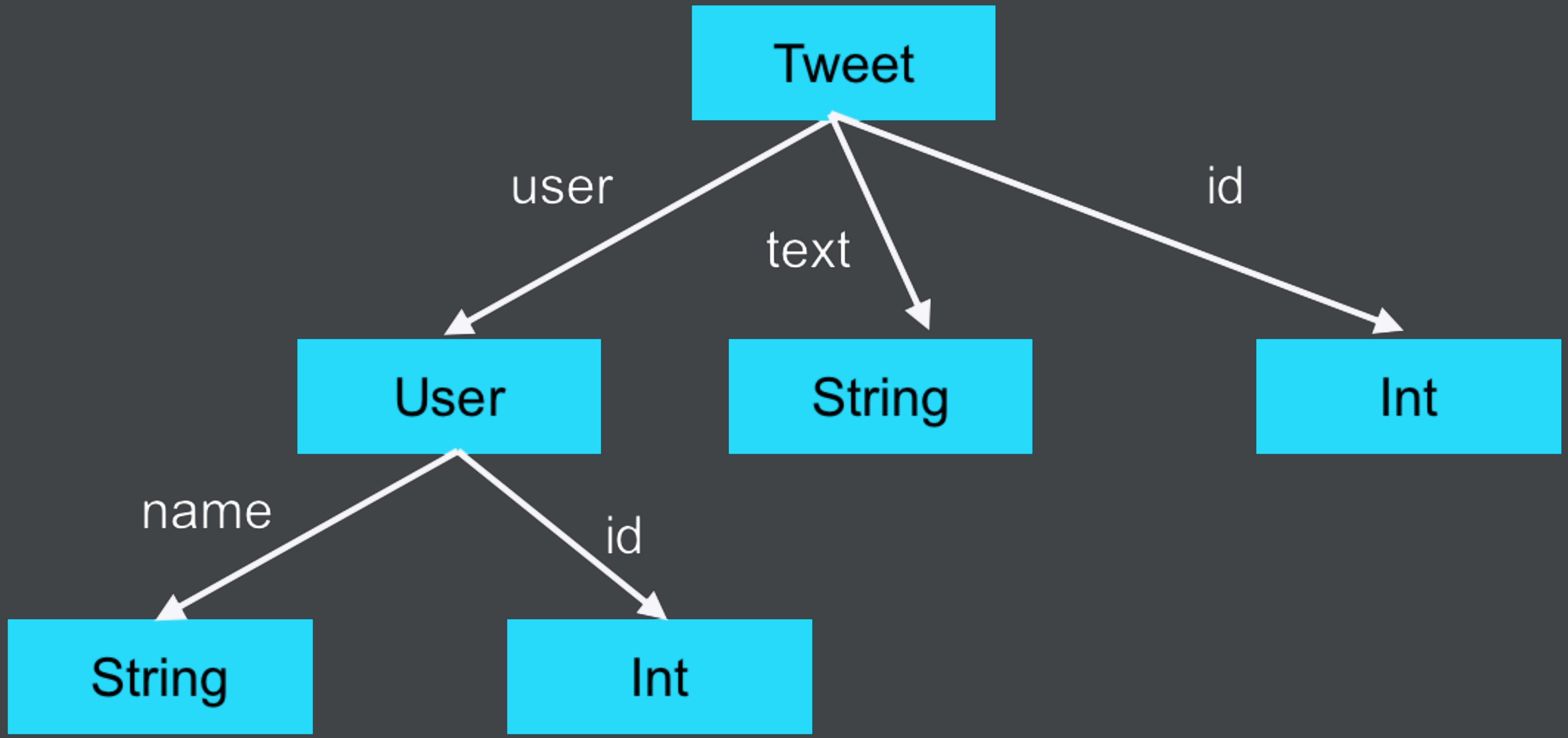
Use cases

- Zewo/Reflection
- wickwirew/Runtime
- alibaba/HandyJSON
- kateinoigakukun/StubKit

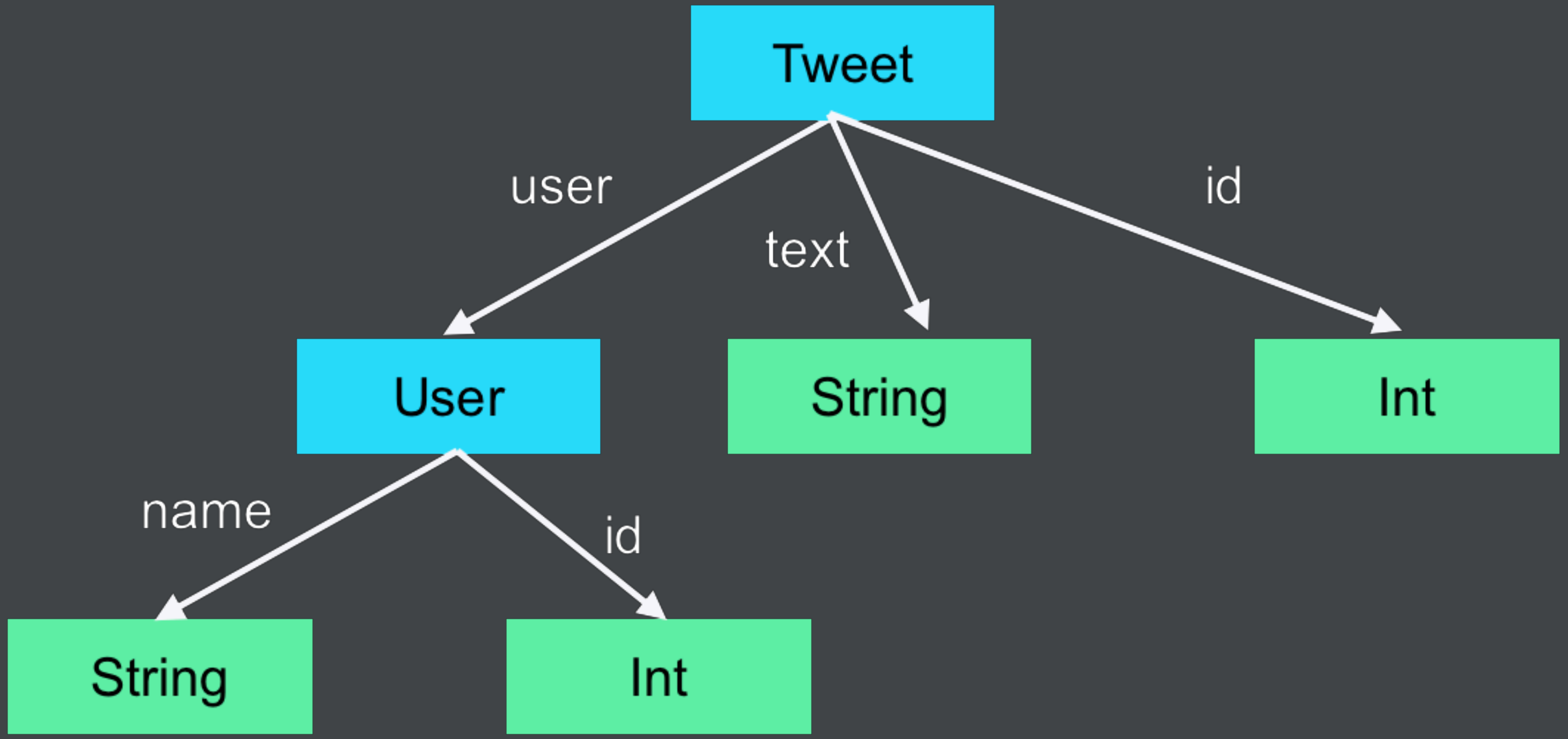
kateinoigakukun/StubKit

```swift
import StubKit

struct User: Codable {
    let name: String
    let age: UInt
}

let user = try Stub.make(User.self)
// User(name: "This is stub string", age: 12345)
```

kateinoigakukun/StubKit



kateinoigakukun/StubKit



kateinoigakukun/StubKit

```swift
func leafStub<T>(of type: T.Type) -> T {
    guard let stubbable = type as? Stubbable else { return nil }
    return type.stub
}

extension Int: Stubbable {
    var stub: Int { return 12345 }
}

extension enum: Stubbable { // 🚫 Can't extend
    var stub: Self {
        return enumStub()
    }
}
```

kateinoigakukun/StubKit

```swift
func enumStub<T>(of type: T.Type) -> T? {
    if isEnum(type: type) {
        let rawValue = 0
        let rawPointer = withUnsafePointer(to: rawValue) { UnsafeRawPointer($0) }
        return rawPointer.assumingMemoryBound(to: T.self).pointee
    }
    return nil
}

func isEnum<T>(type: T.Type) -> Bool {
    let metadata = unsafeBitCast(type, to: UnsafePointer<EnumMetadata>.self).pointee
    return metadata.kind == 1 // kind value of enum is 1
}
```

kateinoigakukun/StubKit

```swift
func enumStub<T>(of type: T.Type) -> T? {
    if isEnum(type: type) {
        let rawValue = 0
        let rawPointer = withUnsafePointer(to: rawValue) { UnsafeRawPointer($0) }
        return rawPointer.assumingMemoryBound(to: T.self).pointee
    }
    return nil
}

func isEnum<T>(type: T.Type) -> Bool {
    let metadata = unsafeBitCast(type, to: UnsafePointer<EnumMetadata>.self).pointee
    return metadata.kind == 1 // kind value of enum is 1
}
```

kateinoigakukun/StubKit

```swift
func enumStub<T>(of type: T.Type) -> T? {
    if isEnum(type: type) {
        let rawValue = 0
        let rawPointer = withUnsafePointer(to: rawValue) { UnsafeRawPointer($0) }
        return rawPointer.assumingMemoryBound(to: T.self).pointee
    }
    return nil
}

func isEnum<T>(type: T.Type) -> Bool {
    let metadata = unsafeBitCast(type, to: UnsafePointer<EnumMetadata>.self).pointee
    return metadata.kind == 1 // kind value of enum is 1
}
```

kateinoigakukun/StubKit

```swift
func enumStub<T>(of type: T.Type) -> T? {
    if isEnum(type: type) {
        let rawValue = 0
        let rawPointer = withUnsafePointer(to: rawValue) { UnsafeRawPointer($0) }
        return rawPointer.assumingMemoryBound(to: T.self).pointee
    }
    return nil
}

func isEnum<T>(type: T.Type) -> Bool {
    let metadata = unsafeBitCast(type, to: UnsafePointer<EnumMetadata>.self).pointee
    return metadata.kind == 1 // kind value of enum is 1
}
```

Caution

- ABI stability
- Responsibility

Summary

- Swift uses metadata for dynamic behavior
- We can use metadata in Swift
- Let's write meta programming libraries!