Alex Hoppen

CocoaHeads Aachen
July '20

Swift Memory Performance — For Beginners, Advanced, Experts And Fanatics

For Beginners

Stack

- Cheap allocation
  - Decrement and increment stack pointer
- Lifetime: Function call

Heap

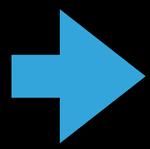
- Expensive allocation
  - Lookup of free memory in advanced data structure
- Lifetime: Dynamic (until freed)
  - Memory leaks possible

Demo

Heap vs. Stack Allocation in C

Stack

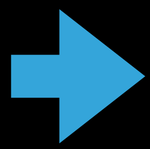
- Cheap allocation
  - Decrement and increment stack pointer
- Lifetime: Function call



`struct`

Heap

- Expensive allocation
  - Lookup of free memory in advanced data structure
- Lifetime: Dynamic (until freed)
  - Memory leaks possible



`class`

Reference Counting

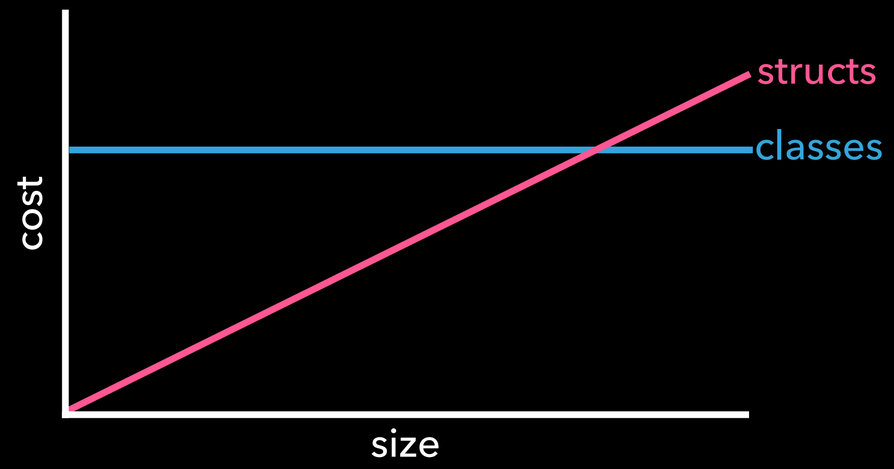
- Lifetime on Heap is managed through reference counting
- See slides 40 – 49 of “Understanding Swift Performance” from WWDC16
  - https://developer.apple.com/wwdc16/416

Retain/Release

- Costly because they need to be thread safe
- Costly = ~7ns = ~20 processor cycles
- Object needs to be locked before they can be retained or released

Passing Structs Around

- All members need to be copied
  - All referenced result in a retain call



Demo

Passing Swift Classes and Structs Around

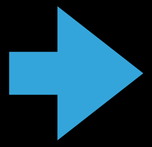
Value-Semantics vs Reference-Semantics

- Value-semantics
  - Better safety guarantee since the state can't change „under our feet"
- Reference-semantics
  - Allow shared state
  - Necessary if shared state is required 😉

For Advanced

VALUE/REFERENCE-TYPE ≠ VALUE/REFERENCE-SEMANTICS

	Value-semantics	Reference-semantics
Value-type		
Reference-type		

 Copy-On-Write (COW) Types

Copy-On-Write Type

- Value type that contains a reference type storage
- Custom setters for each member variable
  - Copy storage if it is not uniquely referenced
  - uniquely referenced = has a retain count of 1
- Useful for structs that
  - Contain a lot of data
  - Contain a lot of reference types

Demo

Copy-On-Write-Type

Enums

- Normal enums
  - Store values inline
  - → Value types
- Indirect enums
  - Allow circular reference
  - → Reference types with value semantics

Demo

Passing Direct and Indirect Enums Around

For Experts

Protocols

- Size of implementing object unknown
- Type of implementing object (value vs. reference) unknown
- Existential container performs abstraction
- There is an abstraction cost
- See slides 142 – 154 of “Understanding Swift Performance” from WWDC16
  - https://developer.apple.com/wwdc16/416

Demo

Protocols

Use Enums Instead of Protocols

- If set of protocol-implementing types is known, pass enum around instead of protocol
- Avoids existential abstraction layer

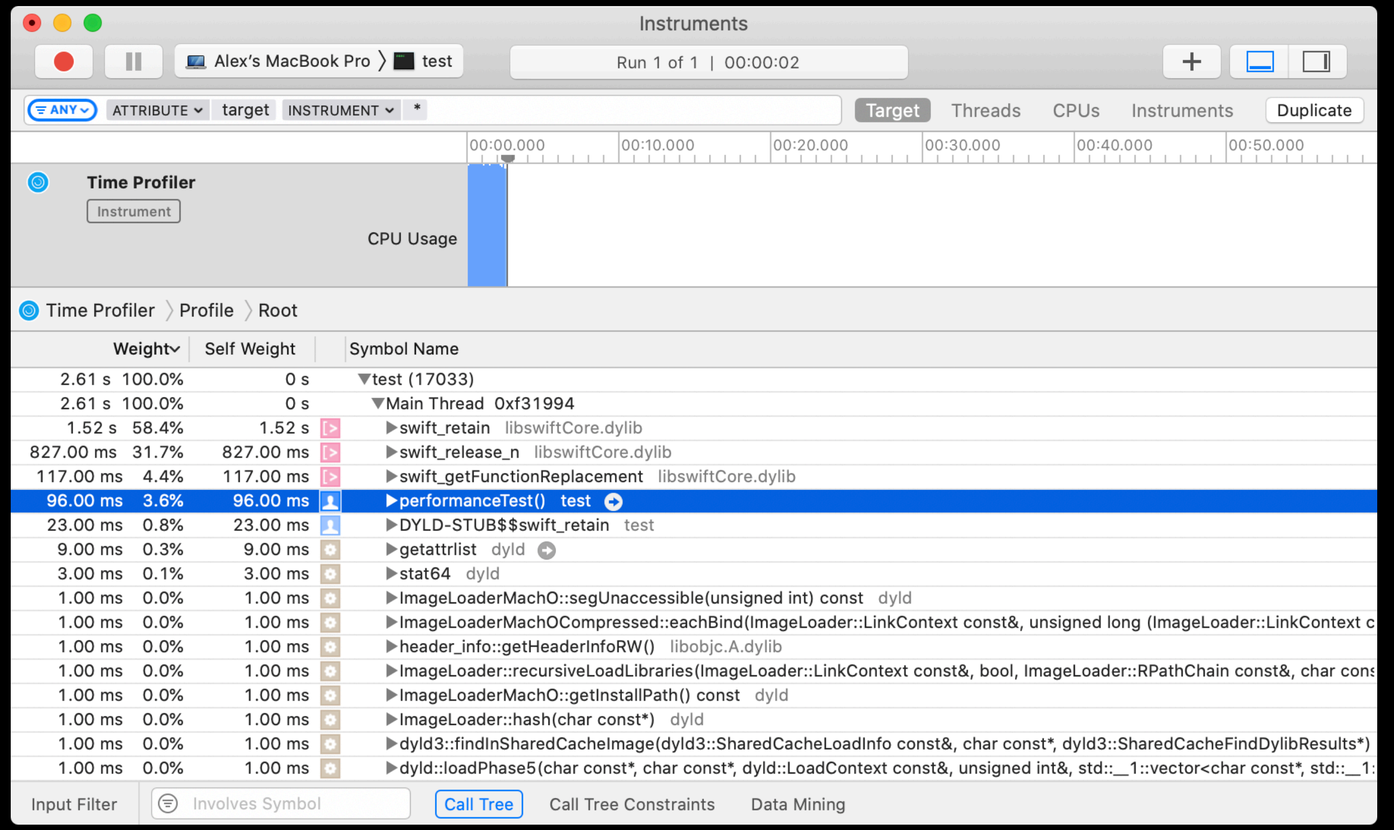
```
protocol Shape {}


struct Line: Shape {}
class Polygon: Shape {}


extension Shape { … }
```

```
enum Shape {
  case line(Line)
  case polygon(Polygon)
}


struct Shape: ShapeMixin {}
class Polygon: ShapeMixin {}


protocol ShapeMixin {}
extension ShapeMixin { … }
```

For Performance-Nerds

Measure Performance Impact using Instruments

- Run performance critical code in Instruments
- Invert Call tree to find expensive functions
- You have a retain/release bottleneck if it starts showing up right at the top



Logging Retain and Release Calls

- Set symbolic breakpoint on `swift_retain, swift_retain_n` and/or `swift_release, swift_release_n`
- `po $arg1` prints name of class being retained
- `p $arg1` prints memory address of retained object
- For _n functions `p $arg2` prints number by which retain count should be increased/decreased, for other functions it's 1

Logging Retain and Release Calls

- If object of type `MyClass` at `4302715776` is retained, then print object using `expr -l Swift -- unsafeBitCast(4302715776, to: MyClass.self)`
  - If class can’t be found, step out into Swift context, afterwards class is known

Demo

Measure Performance Impact and Log Retain/Release Calls

Output SIL Level

- SIL (Swift Intermediate Language) contains much more low-level code
  - Including retain and release calls
- Compile with the `-emit-sil` instead of `-c` to output SIL instead of object code
  - Retrieve Swift compiler call from Build Log
  - Might need to remove options like `-incremental` that are incompatible with `-emit-sil`

Debugging SIL

- Add `-g -Xfrontend -gsil` to “Other Swift Flags”
- Debugger will consider SIL as source language instead of Swift
- Stepping is based on SIL
- Set symbolic breakpoint using mangled name of interesting function
- To unmangle names use `xcrun swift-demangle <demangled-name>`

Demo

Output and Debug SIL

Swift Calling Conventions

- @guaranteed (aka +0): Caller guarantees that argument is alive for entire function call
  - Callee needs to retain object if it is being stored
- @owned (aka +1): Caller passes the argument with a reference count of +1
  - Callee needs to release object if it doesn’t store it
- @inout: Caller passes an argument, caller may modify argument
  - Callee needs to retain object if it is being stored

Demo

Use inout For Argument Passing