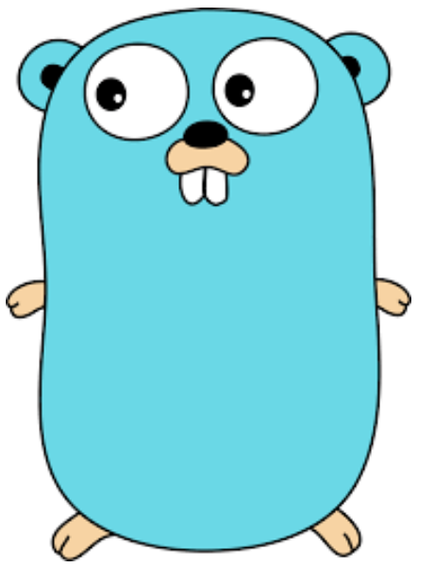
Swift vs Go

CS263, Runtime Systems

Eirik Sandberg



Swift

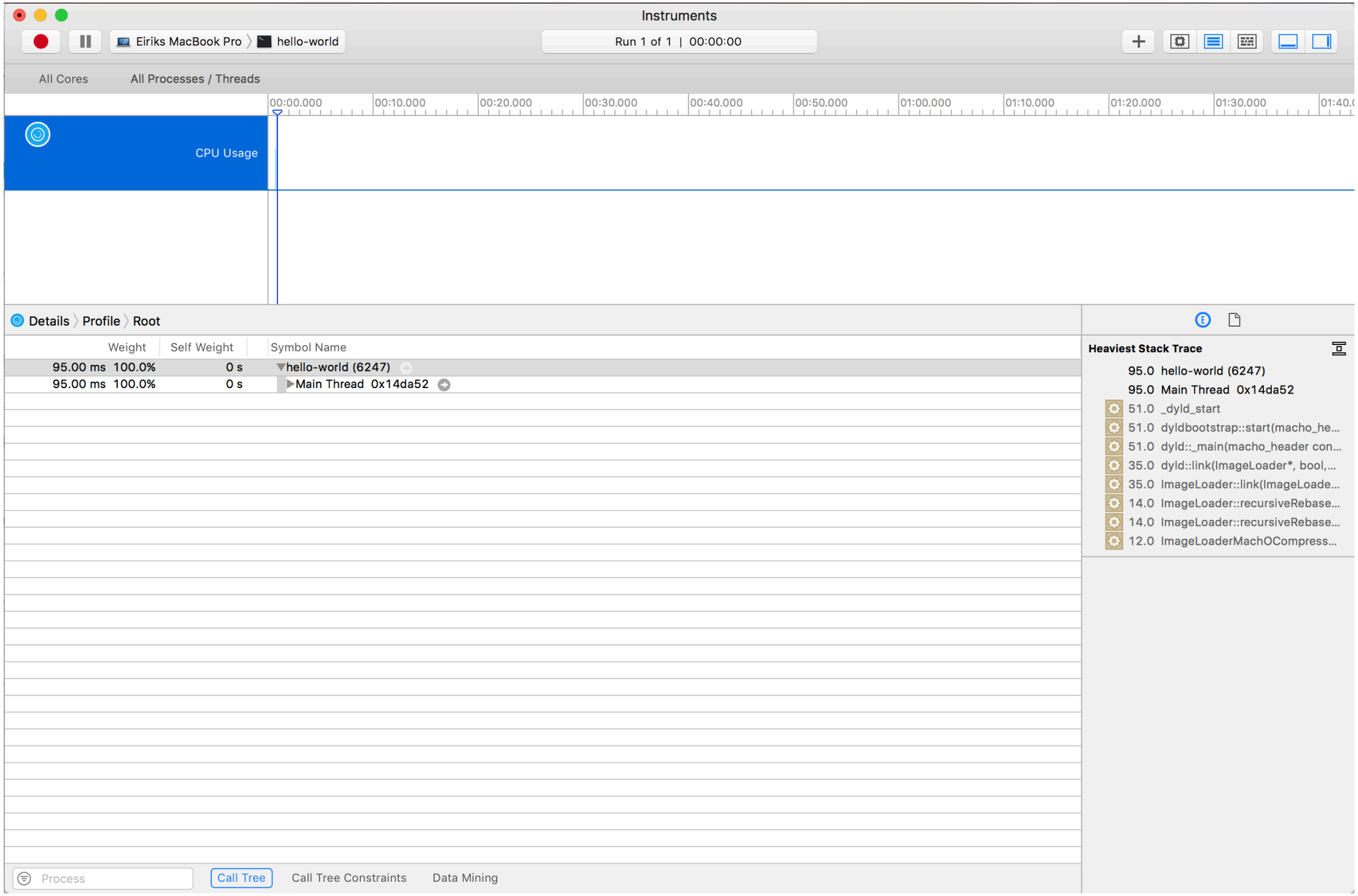
- Developed by Apple
  - iOS and Mac
- Swift itself is statically typed
- Objective-C runtime
  - Dynamically typed
- Garbage Collection
  - ARC

Go

- Developed by Google
- Statically typed language
  - Static method dispatch
  - Dynamic dispatch achieved using interfaces
- AOT compilation
- Garbage collection
  - Mark and sweep

Instruments

- Profiling tool from Apple
  - CPU, Memory allocation, Activity monitor, Time profile etc.
  - Battery usage, Wi-Fi access, Animation etc…
- Works with xCode projects and binary
- Some instruments won’t work with other languages
- Do not require extra code


Instruments
Eiriks MacBook Pro
hello-world
Run 1 of 1 | 00:00:00
All Cores
All Processes / Threads
00:00.000
00:10.000
00:20.000
00:30.000
00:40.000
00:50.000
01:00.000
01:10.000
01:20.000
01:30.000
CPU Usage
Details
Profile
Root
Weight
Self Weight
Symbol Name
95.00 ms 100.0%
0 s
hello-world (6247)
95.00 ms 100.0%
0 s
Main Thread 0x14da52
Heaviest Stack Trace
95.0 hello-world (6247)
95.0 Main Thread 0x14da52
51.0 _dyld_start
51.0 dyldbootstrap::start(macho_he...
51.0 dyld::_main(macho_header con...
35.0 dyld::link(ImageLoader*, bool,...
35.0 ImageLoader::link(ImageLoade...
14.0 ImageLoader::recursiveRebase...
14.0 ImageLoader::recursiveRebase...
12.0 ImageLoaderMachOCompress...
Process
Call Tree
Call Tree Constraints
Data Mining

pprof

- Is a package that is included in Go
  - CPU-profiling, memory-profiling, block profile, tracing etc
- Requires user to add code to the program
  - Get stats from web by running a web-server
  - Write benchmarks and create flags
  - External library

pprof implementation

- Profile Library written by Dave Cheney
  - Go contributor
- Generates .pprof file which can be inspected

```
Welcome to pprof!  For help, type 'help'.
(pprof)
(pprof) top5
Total: 1652 samples
     197  11.9%  11.9%      382  23.1% scanblock
     189  11.4%  23.4%     1549  93.8% main.FindLoops
     130   7.9%  31.2%      152   9.2% sweepspan
     104   6.3%  37.5%      896  54.2% runtime.mallocgc
      98   5.9%  43.5%      100   6.1% flushptrbuf
(pprof)
```

Without library:

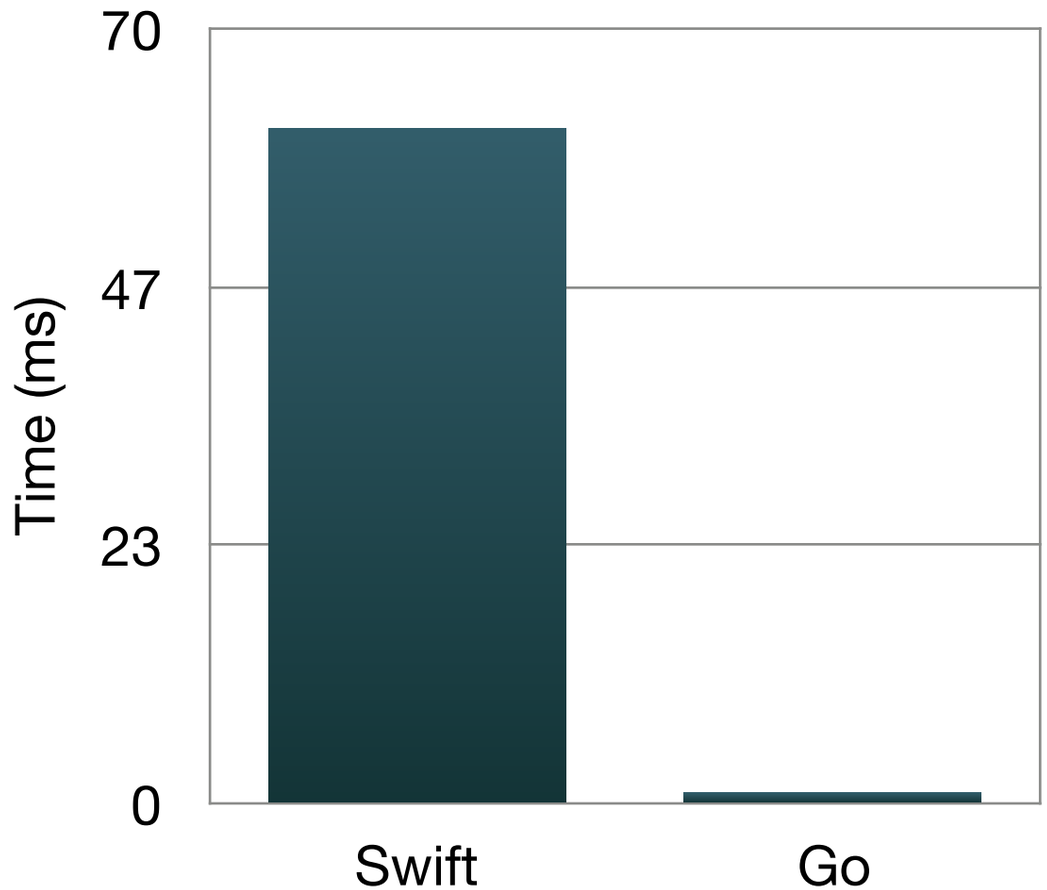
```
1  var cpuprofile = flag.String("cpuprofile", "", "write cpu profile to file")
2
3  func main() {
4      flag.Parse()
5      if *cpuprofile != "" {
6          f, err := os.Create(*cpuprofile)
7          if err != nil {
8              log.Fatal(err)
9          }
10         pprof.StartCPUProfile(f)
11         defer pprof.StopCPUProfile()
12     }
13     ...
14
```

With library:

```
1  import "github.com/pkg/profile"
2
3  func main(){
4    p := profile.Start(profile.MemProfile, profile.ProfilePath("."), profile.NoShutdownHook)
5
6    ...
7    p.Stop()
8  }
```

Start up time

- Swift: 61 ms
- Go 1 ms



Go

Details 〉 Profile 〉 Root

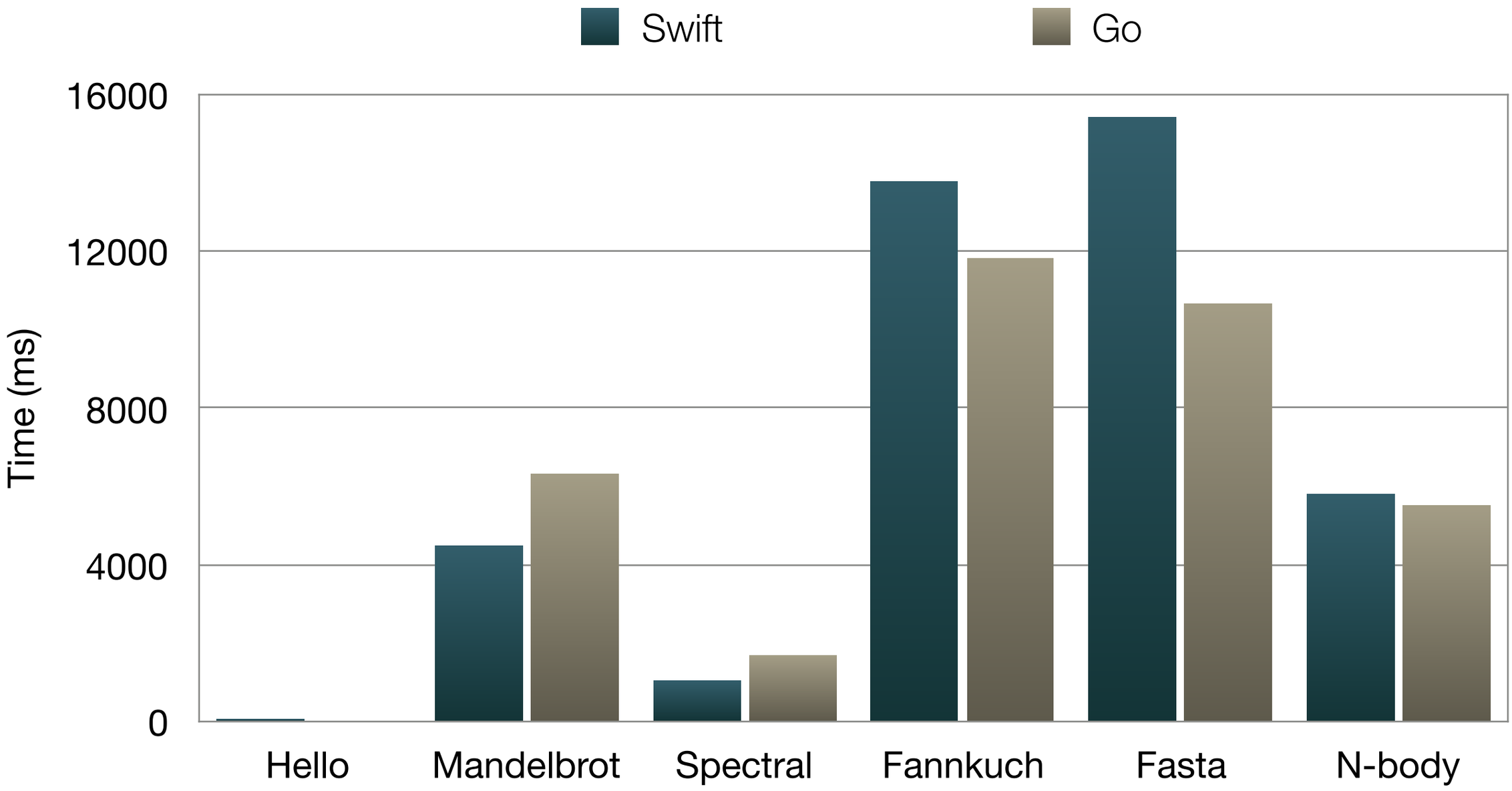
Weight		Self Weight		Symbol Name
1.00 ms	100.0%	0 s		▼hello-world (2648)
1.00 ms	100.0%	0 s		▼<Unnamed Thread> 0x6a8d1
1.00 ms	100.0%	0 s		▼runtime.schedinit hello-world
1.00 ms	100.0%	0 s		▼runtime.mallocinit hello-world
1.00 ms	100.0%	1.00 ms		runtime.mmap hello-world

Swift

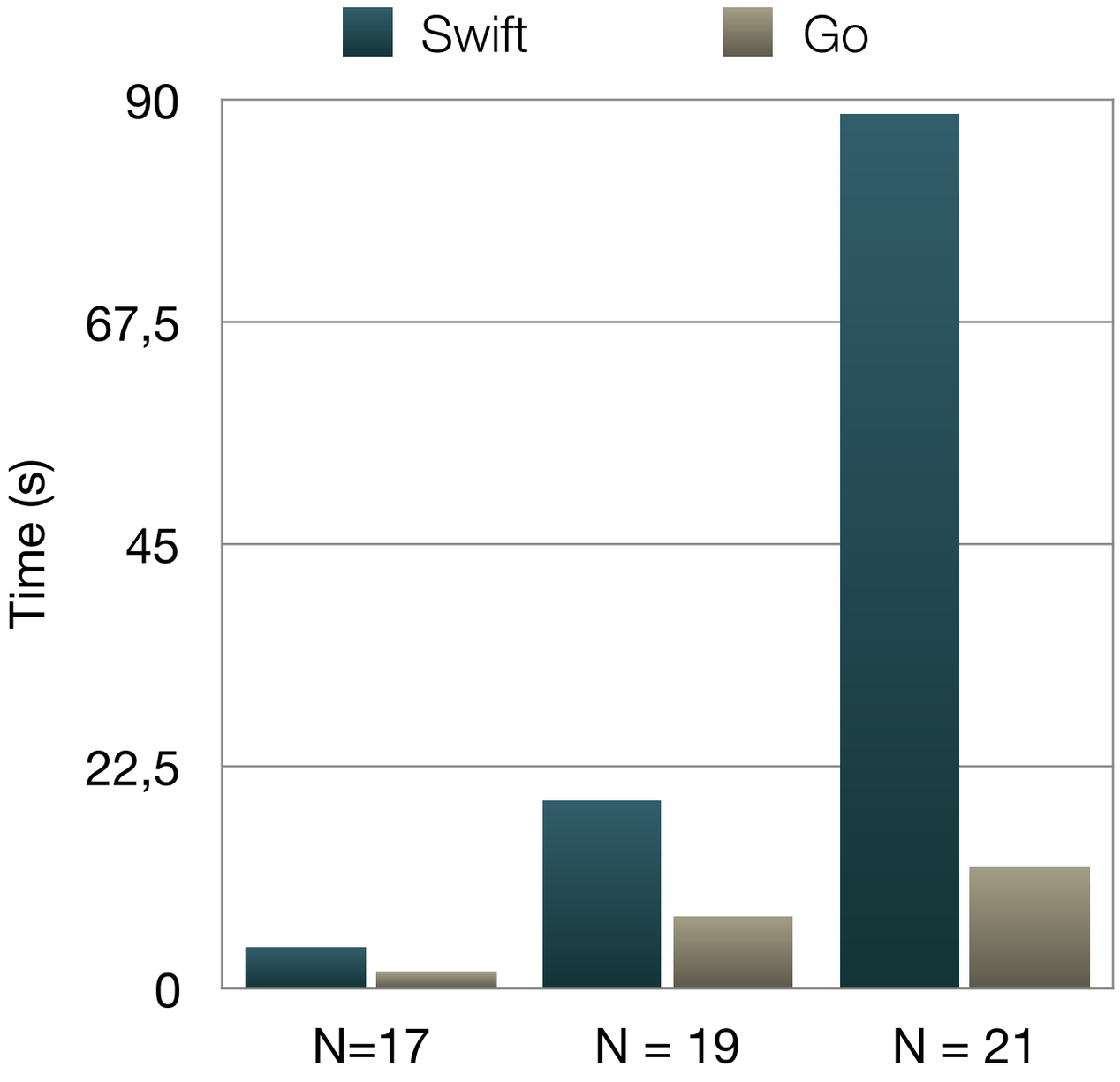
Details 〉 Profile 〉 Root

Weight		Self Weight		Symbol Name
61.00 ms	100.0%	0 s		▼hello-world (2473)
61.00 ms	100.0%	0 s		▼Main Thread 0x6140b
32.00 ms	52.4%	0 s		▼_dyld_start dyld
32.00 ms	52.4%	0 s		▼dyldbootstrap::start(macho_header const*, int, char const**, long, macho_header const*, unsigned long*) dyld
32.00 ms	52.4%	0 s		▼dyld::_main(macho_header const*, unsigned long, int, char const**, char const**, char const**, unsigned long*) dyld
19.00 ms	31.1%	0 s		▼dyld::link(ImageLoader*, bool, bool, ImageLoader::RPathChain const&, unsigned int) dyld
19.00 ms	31.1%	0 s		▼ImageLoader::link(ImageLoader::LinkContext const&, bool, bool, bool, ImageLoader::RPathChain const&, char const*) dyld
8.00 ms	13.1%	0 s		▼ImageLoader::recursiveLoadLibraries(ImageLoader::LinkContext const&, bool, ImageLoader::RPathChain const&, char const*) dylc
6.00 ms	9.8%	0 s		▼dyld::libraryLocator(char const*, bool, char const*, ImageLoader::RPathChain const*, unsigned int&) dyld
6.00 ms	9.8%	0 s		▶dyld::load(char const*, dyld::LoadContext const&, unsigned int&) dyld
2.00 ms	3.2%	0 s		▶ImageLoader::recursiveLoadLibraries(ImageLoader::LinkContext const&, bool, ImageLoader::RPathChain const&, char const*) dy
7.00 ms	11.4%	0 s		▶ImageLoader::recursiveBind(ImageLoader::LinkContext const&, bool, bool) dyld
4.00 ms	6.5%	0 s		▶ImageLoader::recursiveRebase(ImageLoader::LinkContext const&) dyld
13.00 ms	21.3%	0 s		▶dyld::initializeMainExecutable() dyld
29.00 ms	47.5%	0 s		▼start libdyld.dylib
27.00 ms	44.2%	0 s		▼main hello-world
19.00 ms	31.1%	0 s		▼print([Any], separator : String, terminator : String) -> () libswiftCore.dylib
18.00 ms	29.5%	1.00 ms		▼specialized print([Any], separator : String, terminator : String) -> () libswiftCore.dylib
17.00 ms	27.8%	0 s		▼specialized print([Any], separator : String, terminator : String) -> () libswiftCore.dylib
16.00 ms	26.2%	0 s		▼specialized specialized _print<A where ...> ([Any], separator : String, terminator : String, to : inout A) -> () libswiftCore.dylib
16.00 ms	26.2%	0 s		▼specialized specialized _print<A where ...> ([Any], separator : String, terminator : String, to : inout A) -> () libswiftCore.dylib
12.00 ms	19.6%	1.00 ms		▼specialized _print_unlocked<A, B where ...> (A, inout B) -> () libswiftCore.dylib
8.00 ms	13.1%	2.00 ms		▼swift_dynamicCast libswiftCore.dylib
6.00 ms	9.8%	0 s		▶_dynamicCastToExistential(swift::OpaqueValue*, swift::OpaqueValue*, swift::TargetMetadata<swift::InProcess> const*, swift:
3.00 ms	4.9%	0 s		▼swift_getGenericMetadata libswiftCore.dylib
2.00 ms	3.2%	2.00 ms		0x10e34dfc0 libswiftCore.dylib
1.00 ms	1.6%	1.00 ms		0x10e356890 libswiftCore.dylib
2.00 ms	3.2%	0 s		▶specialized _StringCore.append(_StringCore) -> () libswiftCore.dylib
2.00 ms	3.2%	2.00 ms		initializeBufferWithCopyOfBuffer for String libswiftCore.dylib
1.00 ms	1.6%	1.00 ms		_HeapBufferStorage.__deallocating_deinit libswiftCore.dylib

Time profiling

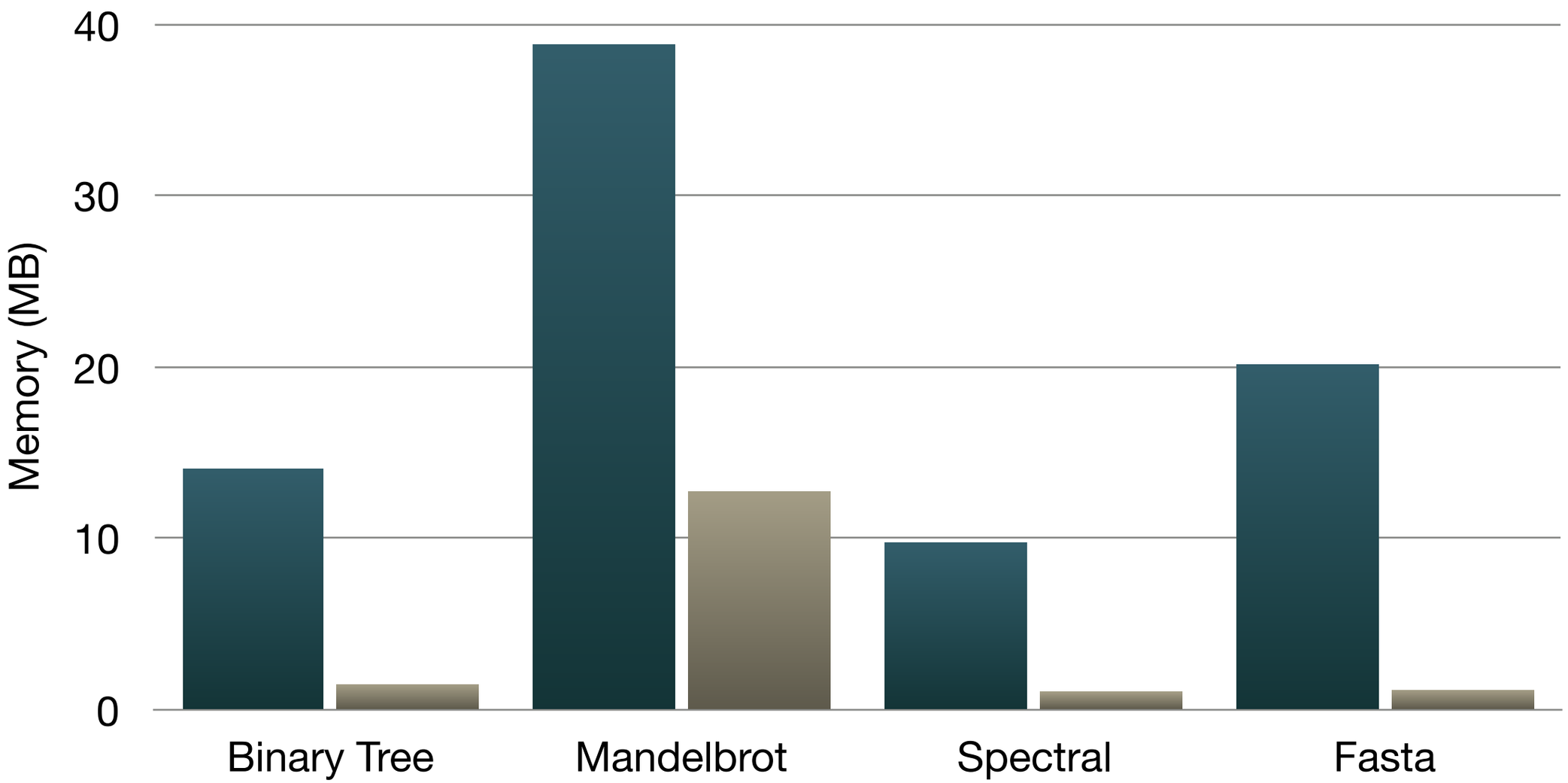


Binary tree



N	Swift	Go	% Faster
15	0,852	0,318	**2,670**
17	4,170	1,723	**2,420**
19	18,927	7,389	**2,562**
21	88,534	12,177	**7,271**

Memory usage



Memory usage

Program	# Persistent	# Transient
Hello World	96	105
Binary Trees	196 782	6 248 062
Mandelbrot	87	281
Spectral norm	84	342
Fannkuch-redux	69	689
Fasta	432	8134
N-body	184	199

Conclusion

- Go is more
  - Lightweight
  - Efficient
- Swift uses
  - Less memory
  - Performs well with few memory allocations
- ARC seriously impacts runtime
  - But keeps memory in use low