Tools

#WWDC14

Improving Your App with Instruments

Session 418

Daniel Delwood
Software Radiologist

Agenda

What's new in Instruments

Memory management

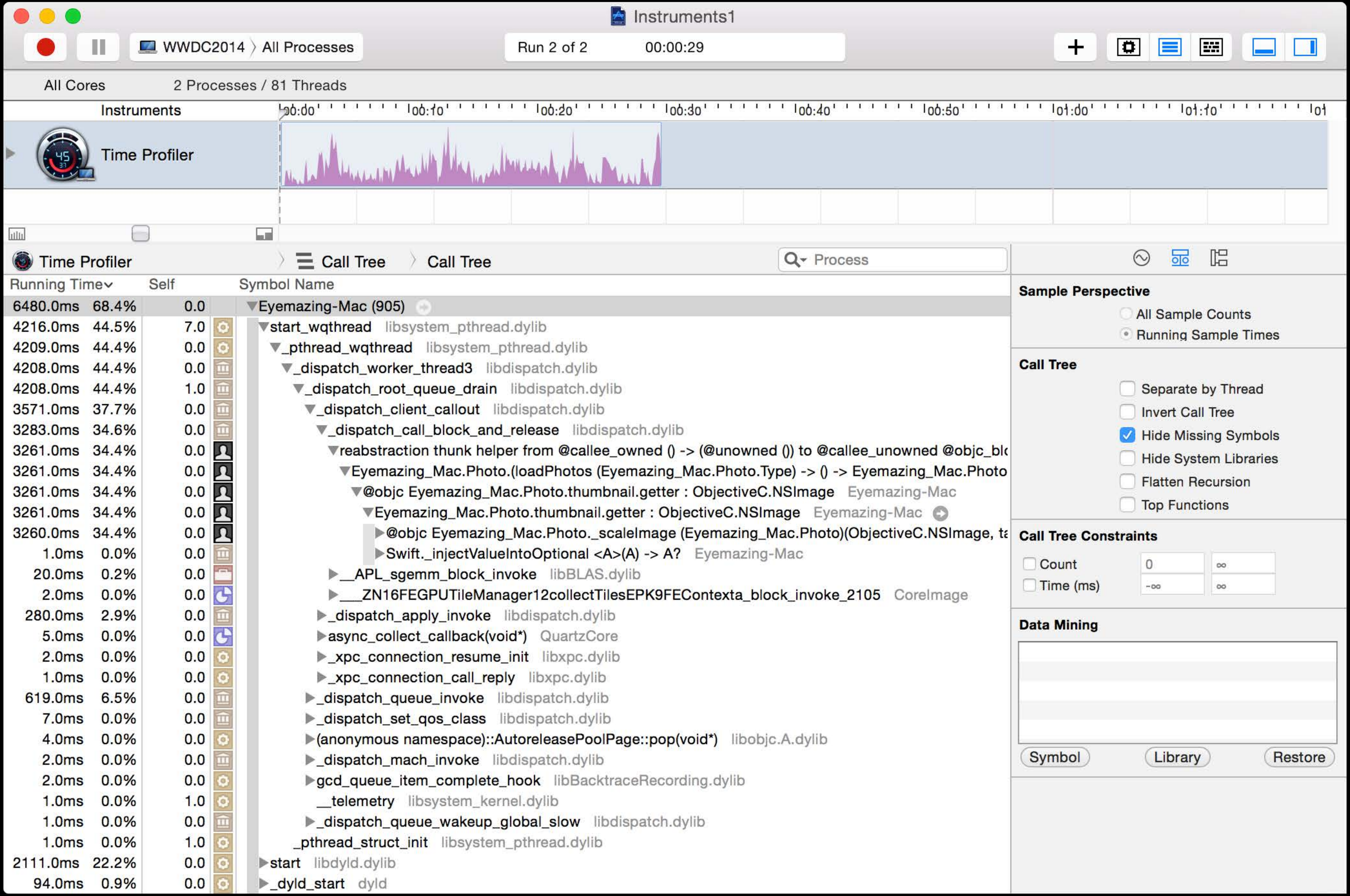
Time profiling

Performance counters

DTrace improvements

What's in Instruments

What's NEW in Instruments



```
Running Time     Self   Symbol Name
6480.0ms 68.4%   0.0    ▼Eyemazing-Mac (905)
4216.0ms 44.5%   7.0     ▼start_wqthread  libsystem_pthread.dylib
4209.0ms 44.4%   0.0      ▼_pthread_wqthread  libsystem_pthread.dylib
4208.0ms 44.4%   0.0       ▼_dispatch_worker_thread3  libdispatch.dylib
4208.0ms 44.4%   1.0        ▼_dispatch_root_queue_drain  libdispatch.dylib
3571.0ms 37.7%   0.0         ▼_dispatch_client_callout  libdispatch.dylib
3283.0ms 34.6%   0.0          ▼_dispatch_call_block_and_release  libdispatch.dylib
3261.0ms 34.4%   0.0           ▼reabstraction thunk helper from @callee_owned () -> (@unowned ()) to @callee_unowned @objc_bl
3261.0ms 34.4%   0.0            ▼Eyemazing_Mac.Photo.(loadPhotos (Eyemazing_Mac.Photo.Type) -> () -> Eyemazing_Mac.Photo
3261.0ms 34.4%   0.0             ▼@objc Eyemazing_Mac.Photo.thumbnail.getter : ObjectiveC.NSImage  Eyemazing-Mac
3261.0ms 34.4%   0.0              ▼Eyemazing_Mac.Photo.thumbnail.getter : ObjectiveC.NSImage  Eyemazing-Mac
3260.0ms 34.4%   0.0               ▶@objc Eyemazing_Mac.Photo._scaleImage (Eyemazing_Mac.Photo)(ObjectiveC.NSImage, ta
   1.0ms  0.0%   0.0               ▶Swift._injectValueIntoOptional <A>(A) -> A?  Eyemazing-Mac
  20.0ms  0.2%   0.0           ▶__APL_sgemm_block_invoke  libBLAS.dylib
   2.0ms  0.0%   0.0           ▶___ZN16FEGPUTileManager12collectTilesEPK9FEContexta_block_invoke_2105  CoreImage
 280.0ms  2.9%   0.0          ▶_dispatch_apply_invoke  libdispatch.dylib
   5.0ms  0.0%   0.0          ▶async_collect_callback(void*)  QuartzCore
   2.0ms  0.0%   0.0          ▶_xpc_connection_resume_init  libxpc.dylib
   1.0ms  0.0%   0.0          ▶_xpc_connection_call_reply  libxpc.dylib
 619.0ms  6.5%   0.0         ▶_dispatch_queue_invoke  libdispatch.dylib
   7.0ms  0.0%   0.0         ▶_dispatch_set_qos_class  libdispatch.dylib
   4.0ms  0.0%   0.0         ▶(anonymous namespace)::AutoreleasePoolPage::pop(void*)  libobjc.A.dylib
   2.0ms  0.0%   0.0         ▶_dispatch_mach_invoke  libdispatch.dylib
   2.0ms  0.0%   0.0         ▶gcd_queue_item_complete_hook  libBacktraceRecording.dylib
   1.0ms  0.0%   1.0          __telemetry  libsystem_kernel.dylib
   1.0ms  0.0%   0.0         ▶_dispatch_queue_wakeup_global_slow  libdispatch.dylib
   1.0ms  0.0%   1.0        _pthread_struct_init  libsystem_pthread.dylib
2111.0ms 22.2%   0.0     ▶start  libdyld.dylib
  94.0ms  0.9%   0.0     ▶_dyld_start  dyld
```

Sample Perspective

- ○ All Sample Counts
- ● Running Sample Times

Call Tree

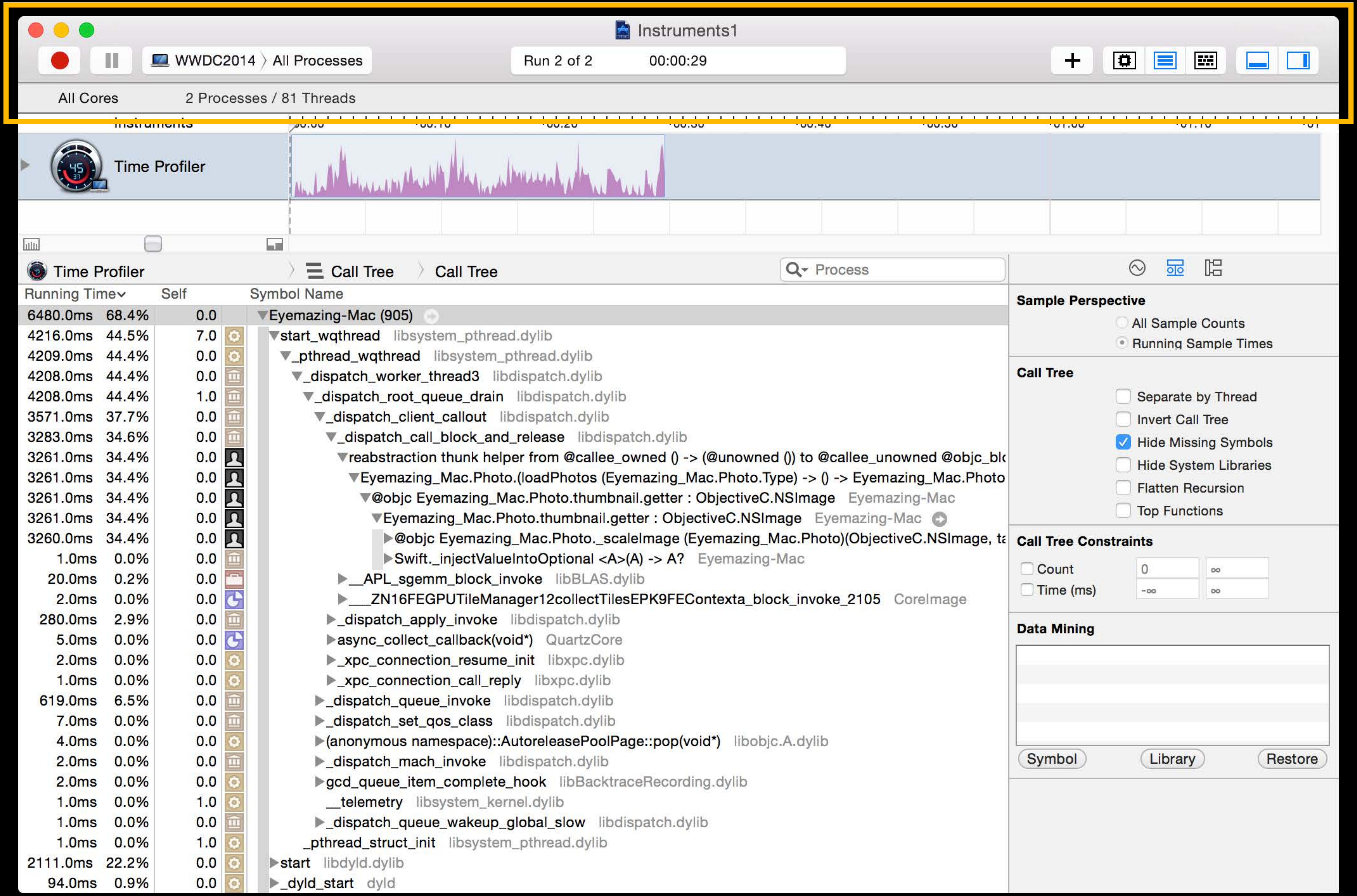
- ☐ Separate by Thread
- ☐ Invert Call Tree
- ☑ Hide Missing Symbols
- ☐ Hide System Libraries
- ☐ Flatten Recursion
- ☐ Top Functions

Call Tree Constraints

☐ Count	0	∞
☐ Time (ms)	-∞	∞

Data Mining

Symbol Library Restore



```
Running Time   Self  Symbol Name
6480.0ms 68.4%  0.0  ▼Eyemazing-Mac (905)
4216.0ms 44.5%  7.0   ▼start_wqthread  libsystem_pthread.dylib
4209.0ms 44.4%  0.0    ▼_pthread_wqthread  libsystem_pthread.dylib
4208.0ms 44.4%  0.0     ▼_dispatch_worker_thread3  libdispatch.dylib
4208.0ms 44.4%  1.0      ▼_dispatch_root_queue_drain  libdispatch.dylib
3571.0ms 37.7%  0.0       ▼_dispatch_client_callout  libdispatch.dylib
3283.0ms 34.6%  0.0        ▼_dispatch_call_block_and_release  libdispatch.dylib
3261.0ms 34.4%  0.0         ▼reabstraction thunk helper from @callee_owned () -> (@unowned ()) to @callee_unowned @objc_bl
3261.0ms 34.4%  0.0          ▼Eyemazing_Mac.Photo.(loadPhotos (Eyemazing_Mac.Photo.Type) -> () -> Eyemazing_Mac.Photo
3261.0ms 34.4%  0.0           ▼@objc Eyemazing_Mac.Photo.thumbnail.getter : ObjectiveC.NSImage  Eyemazing-Mac
3261.0ms 34.4%  0.0            ▼Eyemazing_Mac.Photo.thumbnail.getter : ObjectiveC.NSImage  Eyemazing-Mac
3260.0ms 34.4%  0.0             ▶@objc Eyemazing_Mac.Photo._scaleImage (Eyemazing_Mac.Photo)(ObjectiveC.NSImage, t
   1.0ms  0.0%  0.0             ▶Swift._injectValueIntoOptional <A>(A) -> A?  Eyemazing-Mac
  20.0ms  0.2%  0.0         ▶__APL_sgemm_block_invoke  libBLAS.dylib
   2.0ms  0.0%  0.0         ▶___ZN16FEGPUTileManager12collectTilesEPK9FEContexta_block_invoke_2105  CoreImage
 280.0ms  2.9%  0.0        ▶_dispatch_apply_invoke  libdispatch.dylib
   5.0ms  0.0%  0.0        ▶async_collect_callback(void*)  QuartzCore
   2.0ms  0.0%  0.0        ▶_xpc_connection_resume_init  libxpc.dylib
   1.0ms  0.0%  0.0        ▶_xpc_connection_call_reply  libxpc.dylib
 619.0ms  6.5%  0.0       ▶_dispatch_queue_invoke  libdispatch.dylib
   7.0ms  0.0%  0.0       ▶_dispatch_set_qos_class  libdispatch.dylib
   4.0ms  0.0%  0.0       ▶(anonymous namespace)::AutoreleasePoolPage::pop(void*)  libobjc.A.dylib
   2.0ms  0.0%  0.0       ▶_dispatch_mach_invoke  libdispatch.dylib
   2.0ms  0.0%  0.0       ▶gcd_queue_item_complete_hook  libBacktraceRecording.dylib
   1.0ms  0.0%  1.0        __telemetry  libsystem_kernel.dylib
   1.0ms  0.0%  0.0       ▶_dispatch_queue_wakeup_global_slow  libdispatch.dylib
   1.0ms  0.0%  1.0     _pthread_struct_init  libsystem_pthread.dylib
2111.0ms 22.2%  0.0   ▶start  libdyld.dylib
  94.0ms  0.9%  0.0   ▶_dyld_start  dyld
```

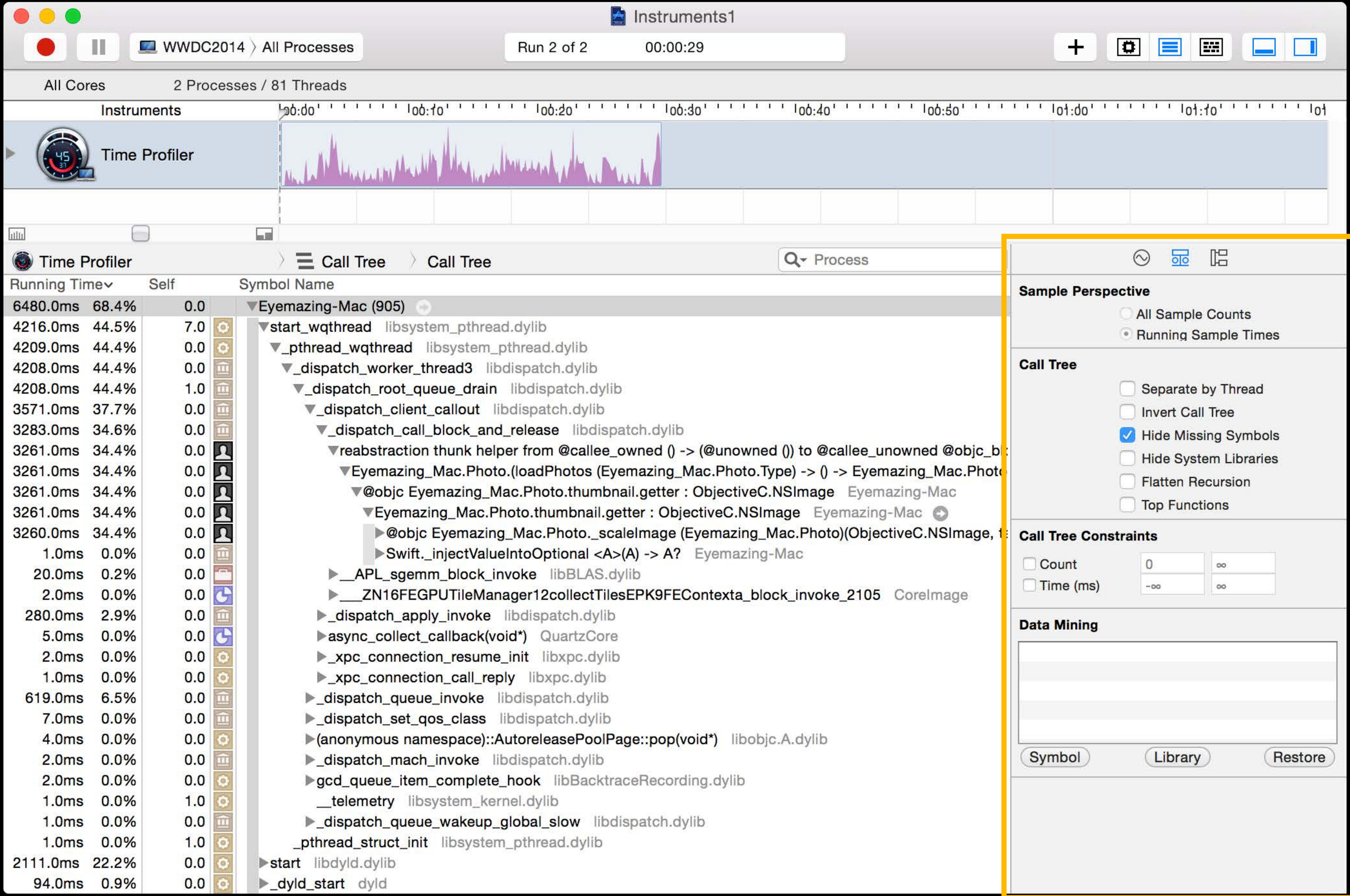
Sample Perspective
- All Sample Counts
- [✓ Running Sample Times]

Call Tree
- ☐ Separate by Thread
- ☐ Invert Call Tree
- ☑ Hide Missing Symbols
- ☐ Hide System Libraries
- ☐ Flatten Recursion
- ☐ Top Functions

Call Tree Constraints

☐ Count	0	∞
☐ Time (ms)	-∞	∞

Data Mining



Instruments1

WWDC2014 › All Processes — Run 2 of 2 — 00:00:29

All Cores — 2 Processes / 81 Threads

Instruments: Time Profiler

Time Profiler › Call Tree › Call Tree

```
Running Time      Self   Symbol Name
6480.0ms  68.4%   0.0    ▼Eyemazing-Mac (905)
4216.0ms  44.5%   7.0     ▼start_wqthread  libsystem_pthread.dylib
4209.0ms  44.4%   0.0      ▼_pthread_wqthread  libsystem_pthread.dylib
4208.0ms  44.4%   0.0       ▼_dispatch_worker_thread3  libdispatch.dylib
4208.0ms  44.4%   1.0        ▼_dispatch_root_queue_drain  libdispatch.dylib
3571.0ms  37.7%   0.0         ▼_dispatch_client_callout  libdispatch.dylib
3283.0ms  34.6%   0.0          ▼_dispatch_call_block_and_release  libdispatch.dylib
3261.0ms  34.4%   0.0           ▼reabstraction thunk helper from @callee_owned () -> (@unowned ()) to @callee_unowned @objc_bl
3261.0ms  34.4%   0.0            ▼Eyemazing_Mac.Photo.(loadPhotos (Eyemazing_Mac.Photo.Type) -> () -> Eyemazing_Mac.Photo
3261.0ms  34.4%   0.0             ▼@objc Eyemazing_Mac.Photo.thumbnail.getter : ObjectiveC.NSImage  Eyemazing-Mac
3261.0ms  34.4%   0.0              ▼Eyemazing_Mac.Photo.thumbnail.getter : ObjectiveC.NSImage  Eyemazing-Mac
3260.0ms  34.4%   0.0               ▶@objc Eyemazing_Mac.Photo._scaleImage (Eyemazing_Mac.Photo)(ObjectiveC.NSImage, t
   1.0ms   0.0%   0.0               ▶Swift._injectValueIntoOptional <A>(A) -> A?  Eyemazing-Mac
  20.0ms   0.2%   0.0           ▶__APL_sgemm_block_invoke  libBLAS.dylib
   2.0ms   0.0%   0.0           ▶___ZN16FEGPUTileManager12collectTilesEPK9FEContexta_block_invoke_2105  CoreImage
 280.0ms   2.9%   0.0          ▶_dispatch_apply_invoke  libdispatch.dylib
   5.0ms   0.0%   0.0          ▶async_collect_callback(void*)  QuartzCore
   2.0ms   0.0%   0.0          ▶_xpc_connection_resume_init  libxpc.dylib
   1.0ms   0.0%   0.0          ▶_xpc_connection_call_reply  libxpc.dylib
 619.0ms   6.5%   0.0         ▶_dispatch_queue_invoke  libdispatch.dylib
   7.0ms   0.0%   0.0         ▶_dispatch_set_qos_class  libdispatch.dylib
   4.0ms   0.0%   0.0         ▶(anonymous namespace)::AutoreleasePoolPage::pop(void*)  libobjc.A.dylib
   2.0ms   0.0%   0.0         ▶_dispatch_mach_invoke  libdispatch.dylib
   2.0ms   0.0%   0.0         ▶gcd_queue_item_complete_hook  libBacktraceRecording.dylib
   1.0ms   0.0%   1.0          __telemetry  libsystem_kernel.dylib
   1.0ms   0.0%   0.0         ▶_dispatch_queue_wakeup_global_slow  libdispatch.dylib
   1.0ms   0.0%   1.0       _pthread_struct_init  libsystem_pthread.dylib
2111.0ms  22.2%   0.0     ▶start  libdyld.dylib
  94.0ms   0.9%   0.0     ▶_dyld_start  dyld
```

Sample Perspective

- All Sample Counts
- [✓ Running Sample Times]

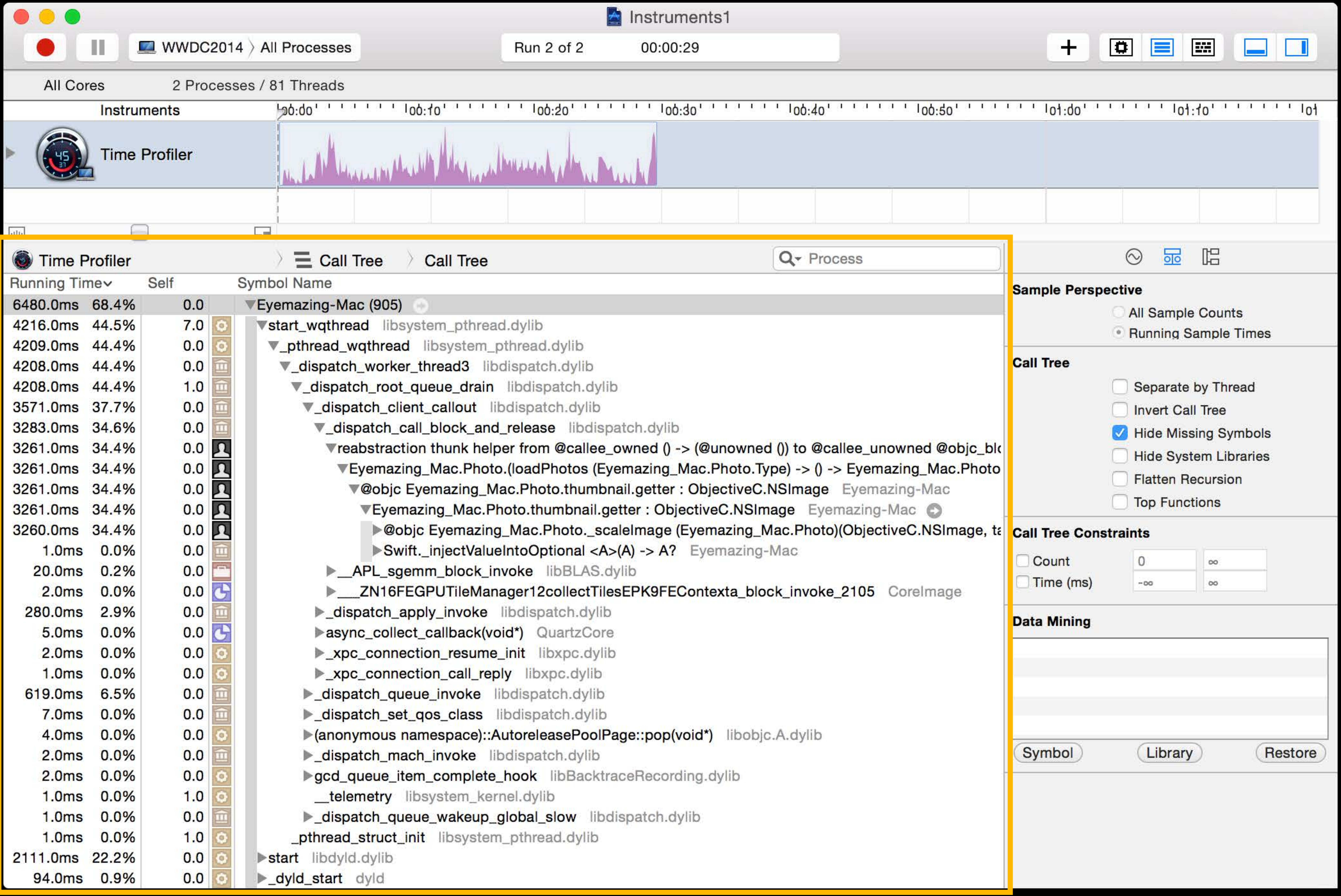
Call Tree

- ☐ Separate by Thread
- ☐ Invert Call Tree
- ☑ Hide Missing Symbols
- ☐ Hide System Libraries
- ☐ Flatten Recursion
- ☐ Top Functions

Call Tree Constraints

☐ Count	0	∞
☐ Time (ms)	-∞	∞

Data Mining



Running Time		Self	Symbol Name
6480.0ms	68.4%	0.0	▼Eyemazing-Mac (905)
4216.0ms	44.5%	7.0	▼start_wqthread libsystem_pthread.dylib
4209.0ms	44.4%	0.0	▼_pthread_wqthread libsystem_pthread.dylib
4208.0ms	44.4%	0.0	▼_dispatch_worker_thread3 libdispatch.dylib
4208.0ms	44.4%	1.0	▼_dispatch_root_queue_drain libdispatch.dylib
3571.0ms	37.7%	0.0	▼_dispatch_client_callout libdispatch.dylib
3283.0ms	34.6%	0.0	▼_dispatch_call_block_and_release libdispatch.dylib
3261.0ms	34.4%	0.0	▼reabstraction thunk helper from @callee_owned () -> (@unowned ()) to @callee_unowned @objc_bl
3261.0ms	34.4%	0.0	▼Eyemazing_Mac.Photo.(loadPhotos (Eyemazing_Mac.Photo.Type) -> () -> Eyemazing_Mac.Photo
3261.0ms	34.4%	0.0	▼@objc Eyemazing_Mac.Photo.thumbnail.getter : ObjectiveC.NSImage Eyemazing-Mac
3261.0ms	34.4%	0.0	▼Eyemazing_Mac.Photo.thumbnail.getter : ObjectiveC.NSImage Eyemazing-Mac
3260.0ms	34.4%	0.0	▶@objc Eyemazing_Mac.Photo._scaleImage (Eyemazing_Mac.Photo)(ObjectiveC.NSImage, t
1.0ms	0.0%	0.0	▶Swift._injectValueIntoOptional <A>(A) -> A? Eyemazing-Mac
20.0ms	0.2%	0.0	▶__APL_sgemm_block_invoke libBLAS.dylib
2.0ms	0.0%	0.0	▶___ZN16FEGPUTileManager12collectTilesEPK9FEContexta_block_invoke_2105 CoreImage
280.0ms	2.9%	0.0	▶_dispatch_apply_invoke libdispatch.dylib
5.0ms	0.0%	0.0	▶async_collect_callback(void*) QuartzCore
2.0ms	0.0%	0.0	▶_xpc_connection_resume_init libxpc.dylib
1.0ms	0.0%	0.0	▶_xpc_connection_call_reply libxpc.dylib
619.0ms	6.5%	0.0	▶_dispatch_queue_invoke libdispatch.dylib
7.0ms	0.0%	0.0	▶_dispatch_set_qos_class libdispatch.dylib
4.0ms	0.0%	0.0	▶(anonymous namespace)::AutoreleasePoolPage::pop(void*) libobjc.A.dylib
2.0ms	0.0%	0.0	▶_dispatch_mach_invoke libdispatch.dylib
2.0ms	0.0%	0.0	▶gcd_queue_item_complete_hook libBacktraceRecording.dylib
1.0ms	0.0%	1.0	__telemetry libsystem_kernel.dylib
1.0ms	0.0%	0.0	▶_dispatch_queue_wakeup_global_slow libdispatch.dylib
1.0ms	0.0%	1.0	_pthread_struct_init libsystem_pthread.dylib
2111.0ms	22.2%	0.0	▶start libdyld.dylib
94.0ms	0.9%	0.0	▶_dyld_start dyld

Sample Perspective
- All Sample Counts
- [✓ Running Sample Times]

Call Tree
- ☐ Separate by Thread
- ☐ Invert Call Tree
- ☑ Hide Missing Symbols
- ☐ Hide System Libraries
- ☐ Flatten Recursion
- ☐ Top Functions

Call Tree Constraints

☐ Count	0	∞
☐ Time (ms)	-∞	∞

Data Mining

Memory Management

Heap Memory

Everything Else

Objective-C's Ownership Model

Retain/Release

Reference counting ownership model based on retain, release

When the count drops to zero, object is freed

Retain/release/autorelease rules established and easy to learn

- *Advanced Memory Management Programming Guide*

Deterministic, simple, and fast

Objective-C's Ownership Model

Managed Retain/Release

Reference counting ownership model based on retain, release

When the count drops to zero, object is freed

Retain/release/autorelease rules established and easy to learn

- *Advanced Memory Management Programming Guide*

Deterministic, simple, and fast

Automated Reference Counting (ARC)

Objective-C's Ownership Model

Managed Retain/Release

Reference counting ownership model based on retain, release

When the count drops to zero, object is freed

Retain/release/autorelease rules established and easy to learn

- *Advanced Memory Management Programming Guide*

Deterministic, simple, and fast

Automated Reference Counting (ARC)

- Still have to manage autorelease pools

```
@autoreleasepool { /* code */ }
```

Swift's Ownership Model

Managed Retain/Release

Reference counting ownership model based on retain, release

When the count drops to zero, object is freed

Deterministic, simple, and fast

Automated Reference Counting (ARC)

Swift's Ownership Model

Managed Retain/Release

Reference counting ownership model based on retain, release

When the count drops to zero, object is freed

Deterministic, simple, and fast

Automated Reference Counting (ARC)

- Working with Objective-C? Still have to manage autorelease pools

```
autoreleasepool { /* code */ }
```

Allocations

What does it report?

Heap allocations

- Class names — e.g. NSMutableArray, MyApp.MainViewController
- Reference types only (`class`, not `struct`)
- Retain/Release histories

Virtual Memory (VM) allocations

- Paths for mapped files

Stack traces for all

Demo

Allocations + App Extension

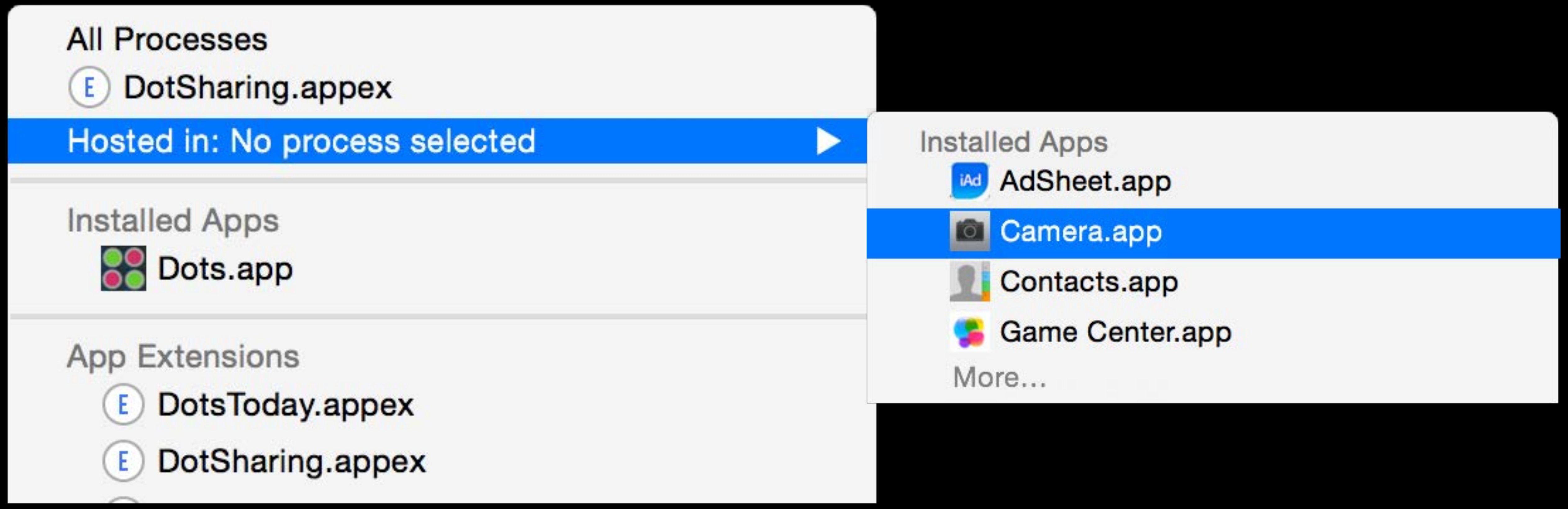
App Extensions

Profiling with Instruments

Specify host App

- When profiling Xcode scheme
- In Instruments

Transient, but memory matters



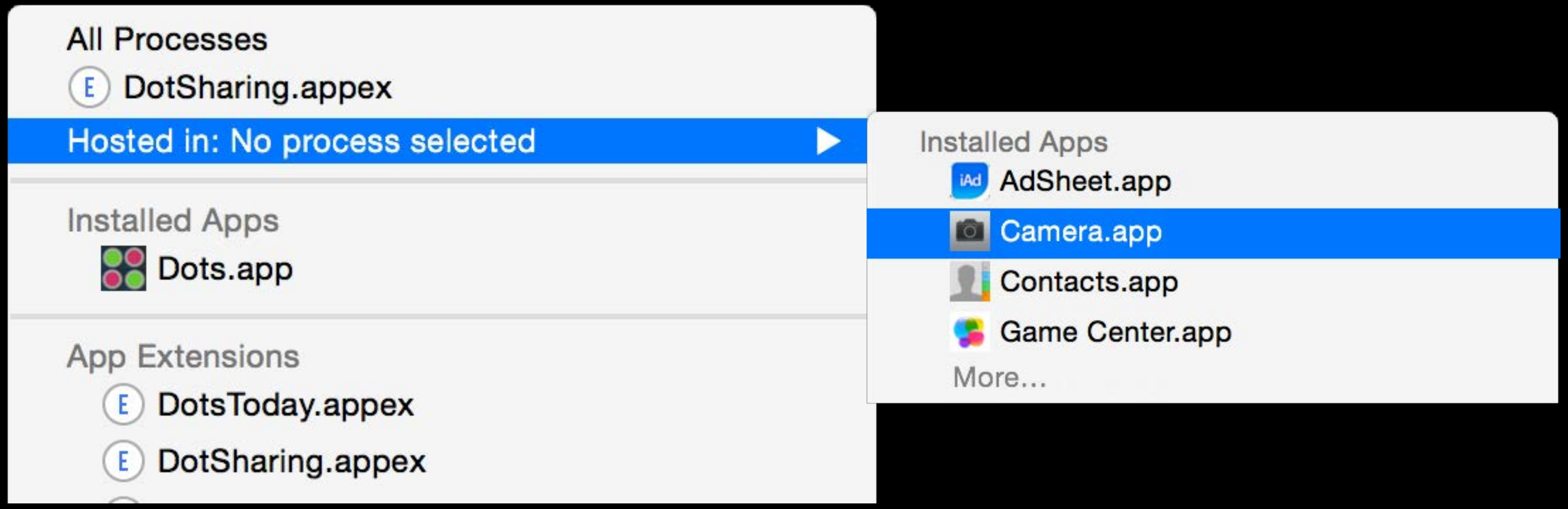
App Extensions

Profiling with Instruments

Specify host App

- When profiling Xcode scheme
- In Instruments

Transient, but memory matters



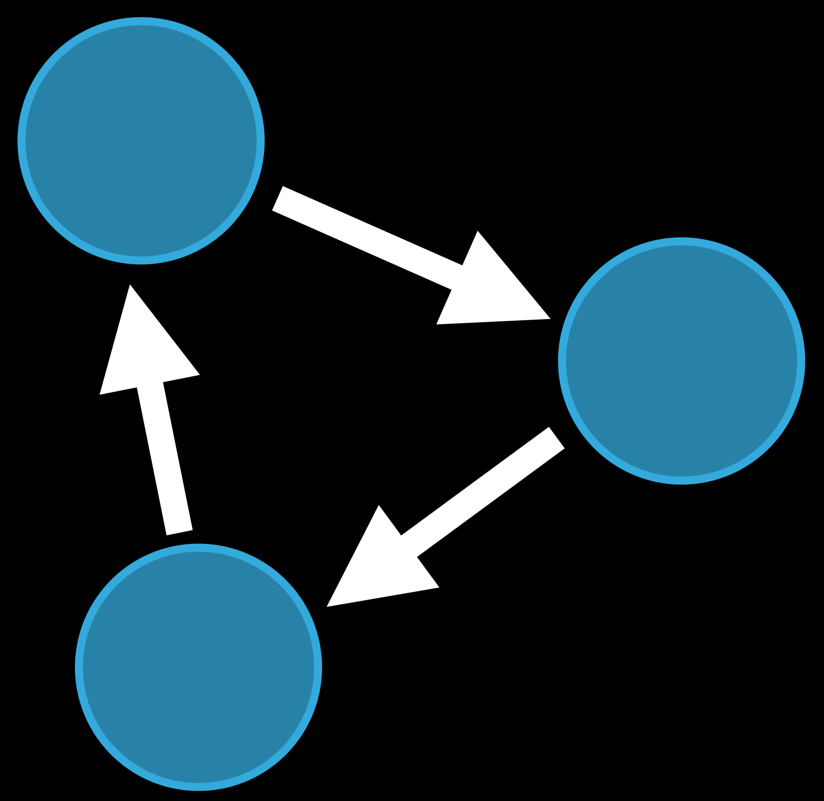
● Creating Extensions for iOS and OS X, Part 1	Mission	Tuesday 2:00PM
● Creating Extensions for iOS and OS X, Part 2	Mission	Wednesday 11:30AM

Memory Management with Swift

Language tools

Obj-C code can still mismatch Retain/Release

Can still form cycles in Swift



Memory Management with Swift

Language tools

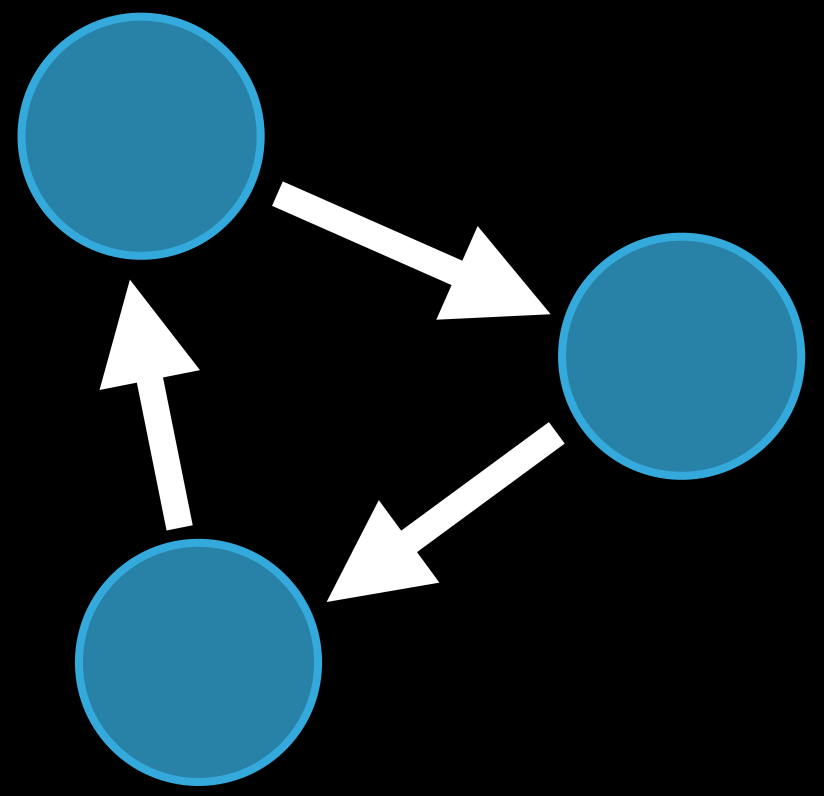
Obj-C code can still mismatch Retain/Release

Can still form cycles in Swift

Manage graph, not retain/release

`weak`

`unowned`



Memory Management with Swift

Language tools

Obj-C code can still mismatch Retain/Release

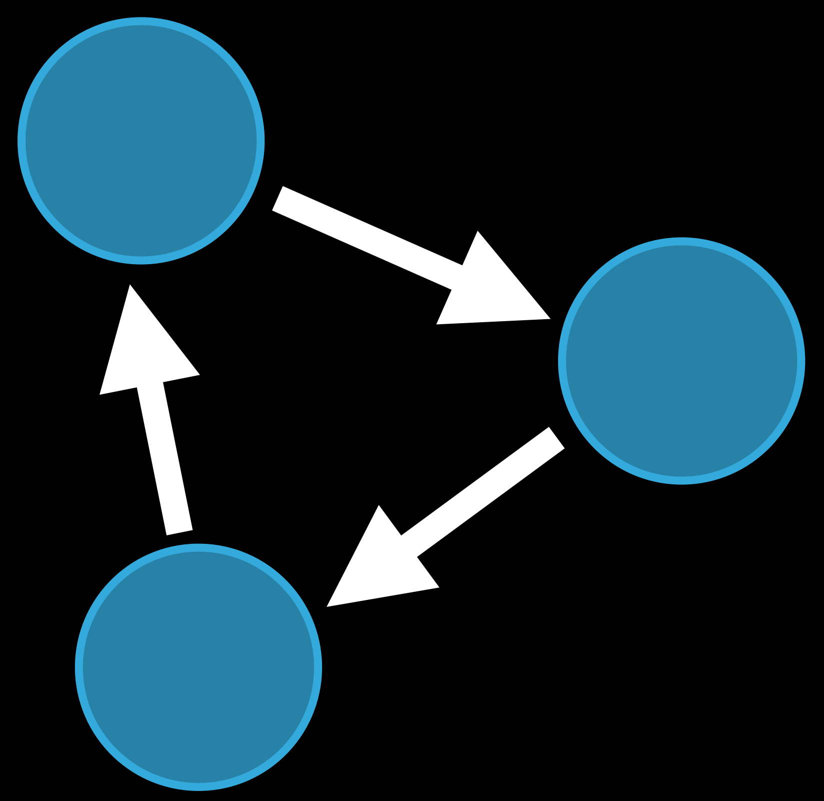
Can still form cycles in Swift

Manage graph, not retain/release

`weak` var x : Optional<T> / T? = object

Returns `T` or `nil` when accessed, based on existence of object

`unowned`

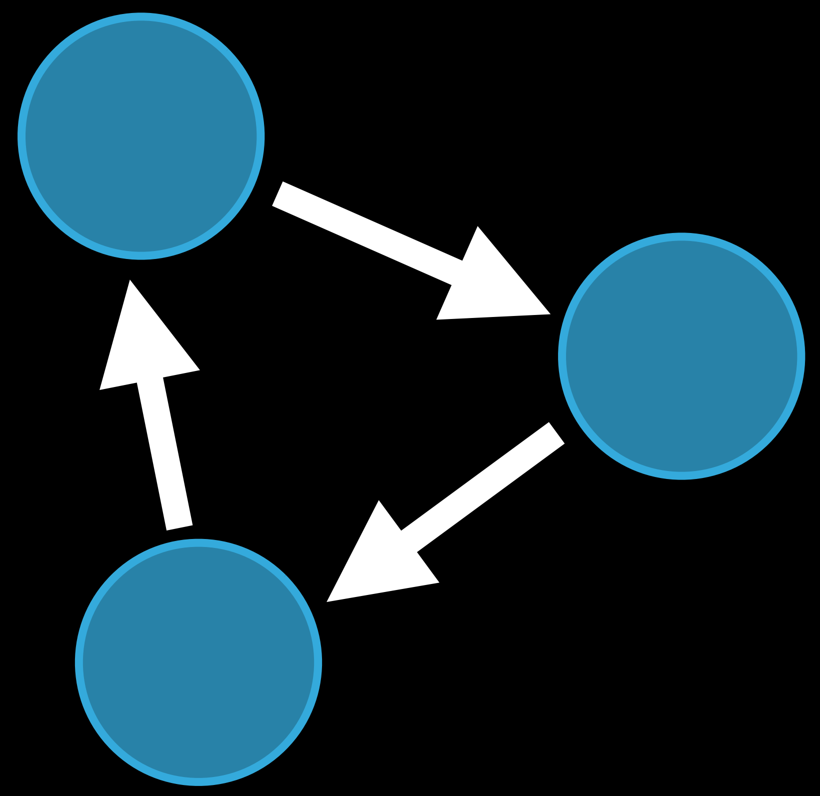


Memory Management with Swift

Language tools

Obj-C code can still mismatch Retain/Release

Can still form cycles in Swift



Manage graph, not retain/release

`weak` var x : Optional<T> / T? = object

Returns `T` or `nil` when accessed, based on existence of object

`unowned` let / var x : T = object

Returns `T` always, but if object doesn't exist… deterministic 💣

^block Captures

Here be dragons

```
[self.currentGame registerForStateChanges:^{
    if (self.currentGame == newGame) {
        [self.tableView reloadData];
    }
}];
```

'self' and 'newGame' captured strongly

^block Captures

Here be dragons

```
__weak typeof(newGame) weakGame = newGame;
__weak typeof(self) weakSelf = self;
[self.currentGame registerForStateChanges:^{
    if (self.currentGame == newGame) {
        [self.tableView reloadData];
    }
}];
```

'self' and 'newGame' captured strongly

^block Captures

Here be dragons

```
__weak typeof(newGame) weakGame = newGame;
__weak typeof(self) weakSelf = self;
[self.currentGame registerForStateChanges:^{
    if (weakSelf.currentGame == weakGame) {
        [weakSelf.tableView reloadData];
    }
}];
```

Swift Closures

Behold, the power of capture lists

```
currentGame.registerForStateChanges() {
    if self.currentGame == newGame {
        self.tableView!.reloadData()
    }
}
```

Swift Closures

Behold, the power of capture lists

```
currentGame.registerForStateChanges() {[weak self, newGame] in
    if self.currentGame == newGame {
        self.tableView!.reloadData()
    }
}
```

Swift Closures

Behold, the power of capture lists

```
currentGame.registerForStateChanges() {[weak self, newGame] in
    if self?.currentGame == newGame {
        self?.tableView!.reloadData()
    }
}
```

Swift Closures

Behold, the power of capture lists

```
currentGame.registerForStateChanges() {[weak self, newGame] in
    if self?.currentGame == newGame {
        self?.tableView!.reloadData()
    }
}
```

● Swift Interoperability In-Depth	Presidio	Wednesday 3:15PM
● Advanced Swift	Presidio	Thursday 11:30AM
● Fixing Memory Issues	Session 410	WWDC13 Videos

Time Profiling

Kris Markel
Performance Tools Engineer

Why?

Why?

To provide a great user experience

Why?

To provide a great user experience

- Faster app launch times

Why?

To provide a great user experience

- Faster app launch times
- Keep the frame rate at 60fps

Why?

To provide a great user experience

- Faster app launch times
- Keep the frame rate at 60fps
- Buttery-smooth scrolling

Why?

To provide a great user experience

- Faster app launch times
- Keep the frame rate at 60fps
- Buttery-smooth scrolling
- Responsive UI

What?

An instrument that samples stack trace information at prescribed intervals

Provides an idea of how much time is spent in each method

When?

When?

Investigate specific problems

When?

Investigate specific problems

- If you see stuttering or frame rate slowdowns

When?

Investigate specific problems

- If you see stuttering or frame rate slowdowns
- Some part of your app is taking too long

When?

Investigate specific problems

- If you see stuttering or frame rate slowdowns
- Some part of your app is taking too long

Identify and fix hotspots before they become problems

When?

Investigate specific problems

- If you see stuttering or frame rate slowdowns
- Some part of your app is taking too long

Identify and fix hotspots before they become problems

- Keep an eye on the CPU gauge in Xcode

Demo

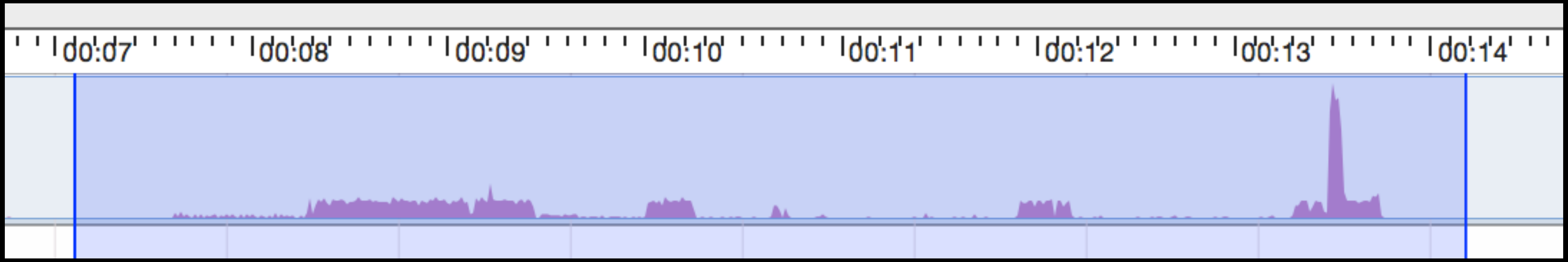
Time Profiler in action

Review

Track view

Identify and zoom into problem areas

- Drag to apply a time range filter
- Shift+drag to zoom in
- Control+drag to zoom out



Review

New Inspector panes

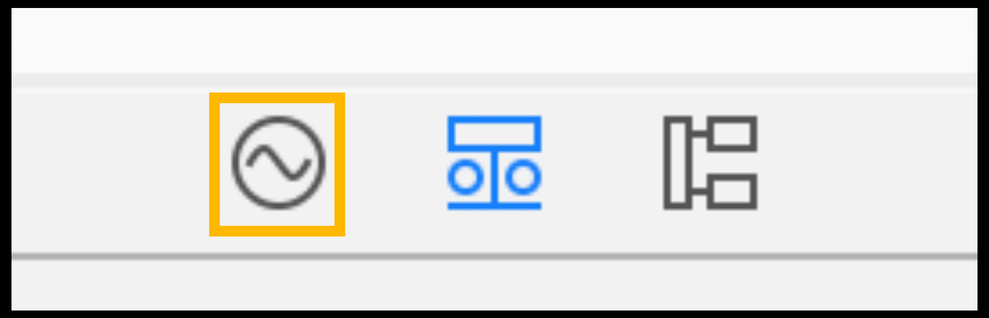
Use keyboard shortcuts to
quickly move between panes

Review

New Inspector panes

Use keyboard shortcuts to quickly move between panes

- ⌘1—Record settings

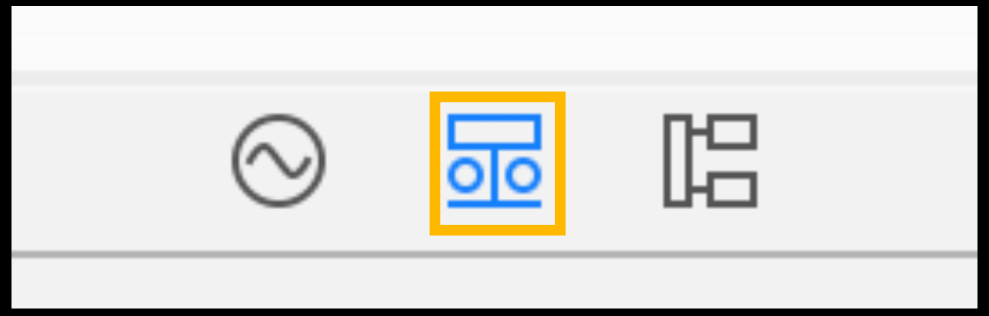


Review

New Inspector panes

Use keyboard shortcuts to quickly move between panes

- ⌘1—Record settings
- ⌘2—Display settings

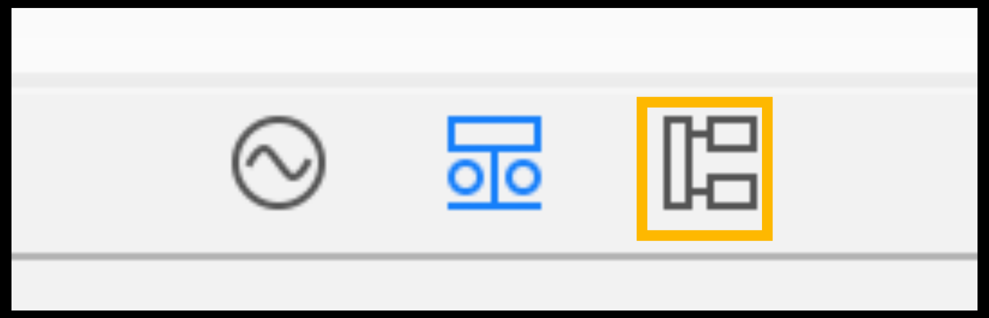


Review

New Inspector panes

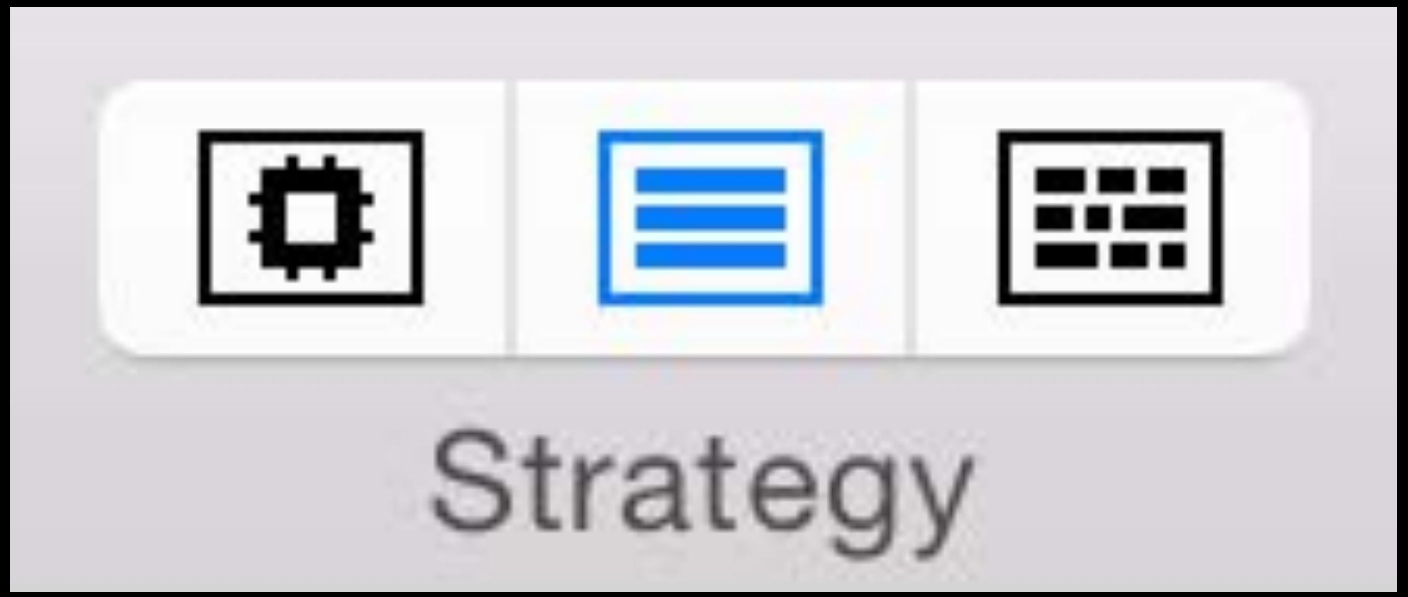
Use keyboard shortcuts to quickly move between panes

- ⌘1—Record settings
- ⌘2—Display settings
- ⌘3—Extended detail



Review

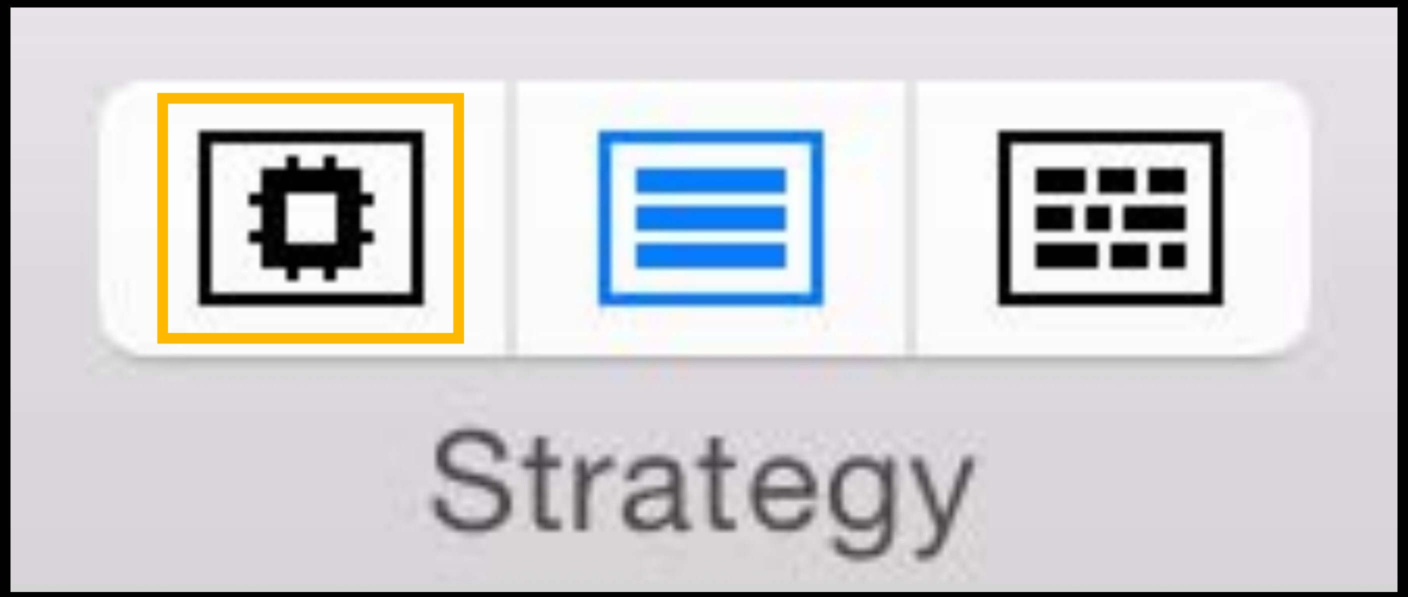
Strategy views



Review

Strategy views

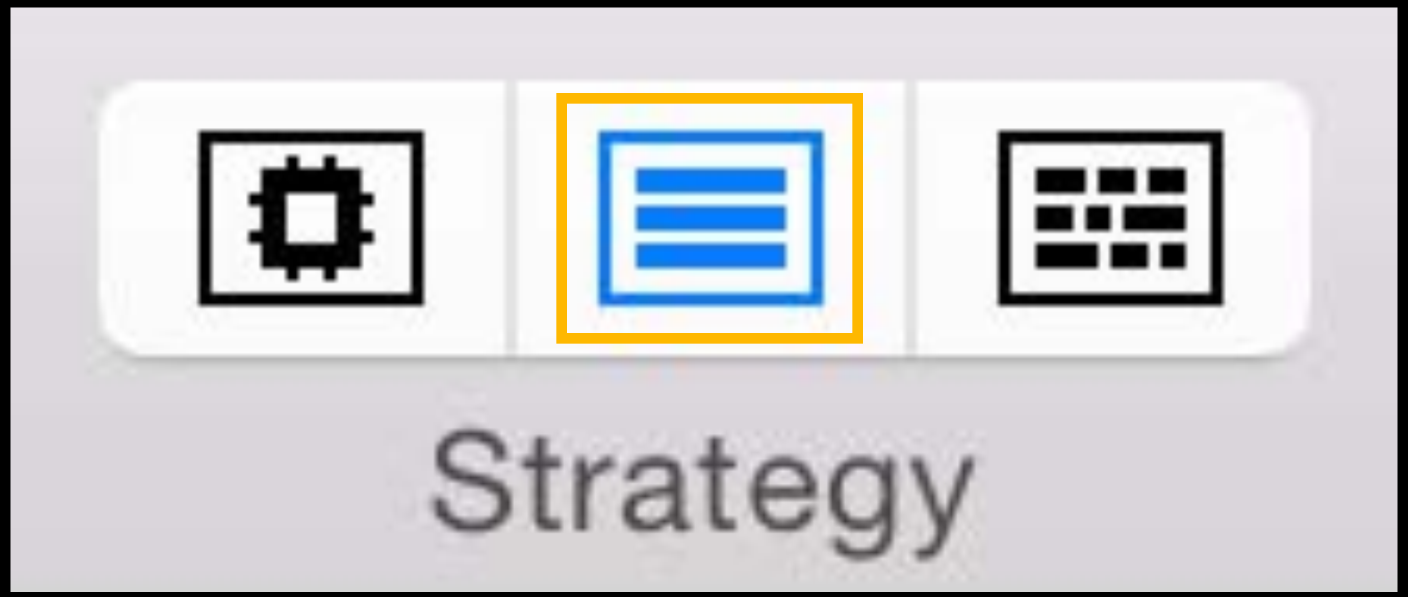
- Cores strategy



Review

Strategy views

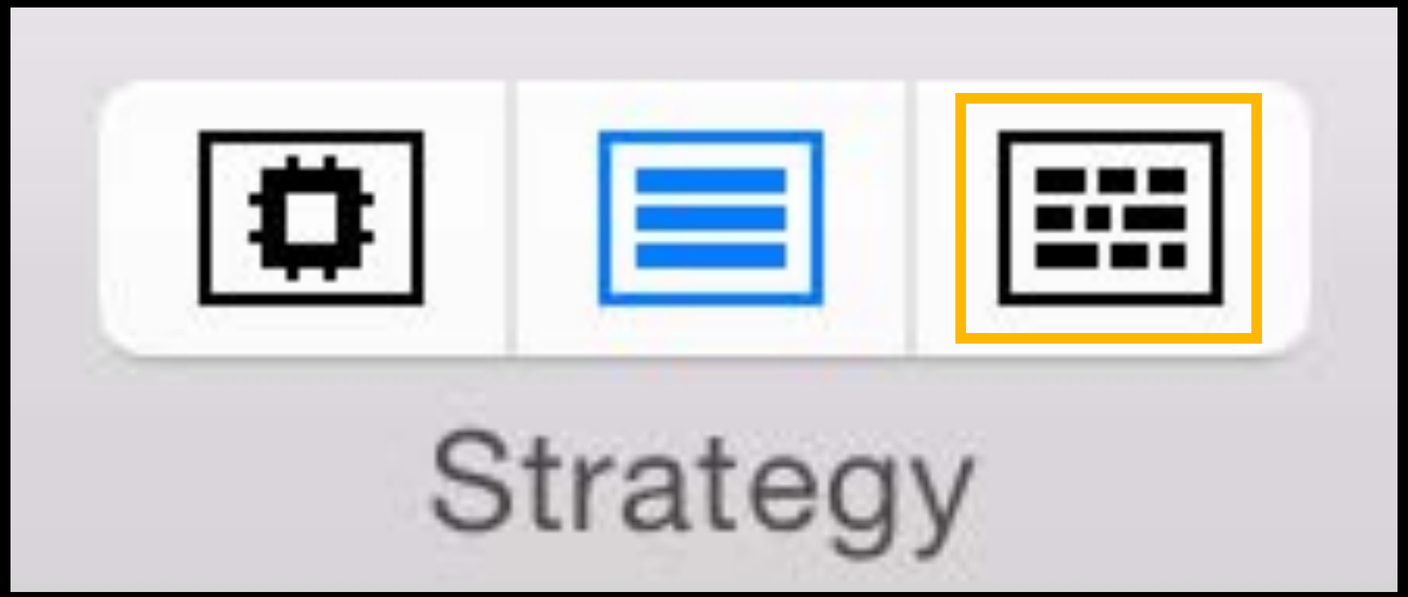
- Cores strategy
- Instruments strategy



Review

Strategy views

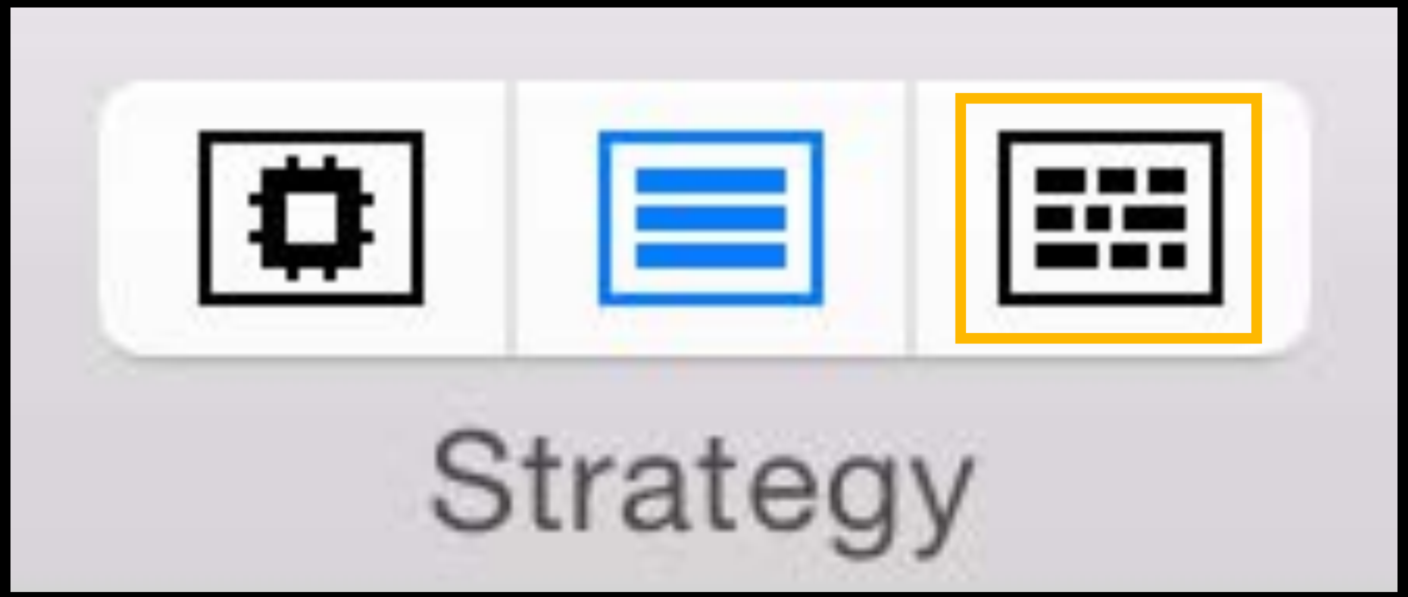
- Cores strategy
- Instruments strategy
- Threads strategy

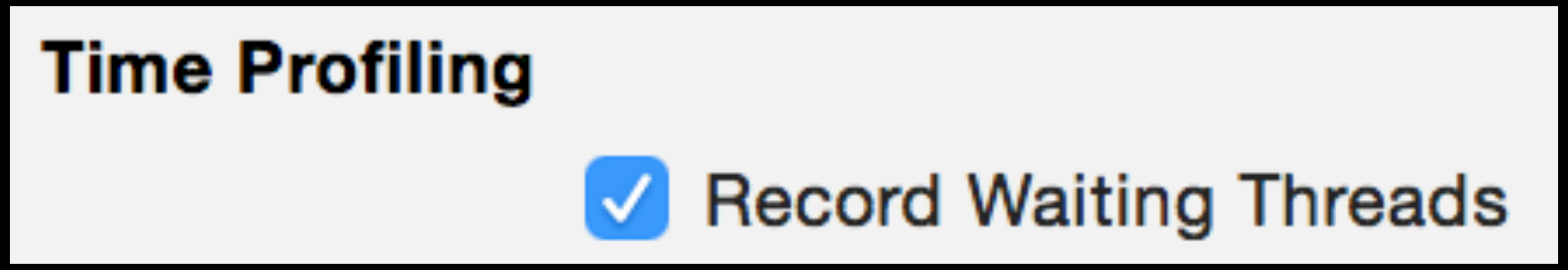


Review

Strategy views

- Cores strategy
- Instruments strategy
- Threads strategy
  - Enable Record Waiting Threads to expose blocked threads

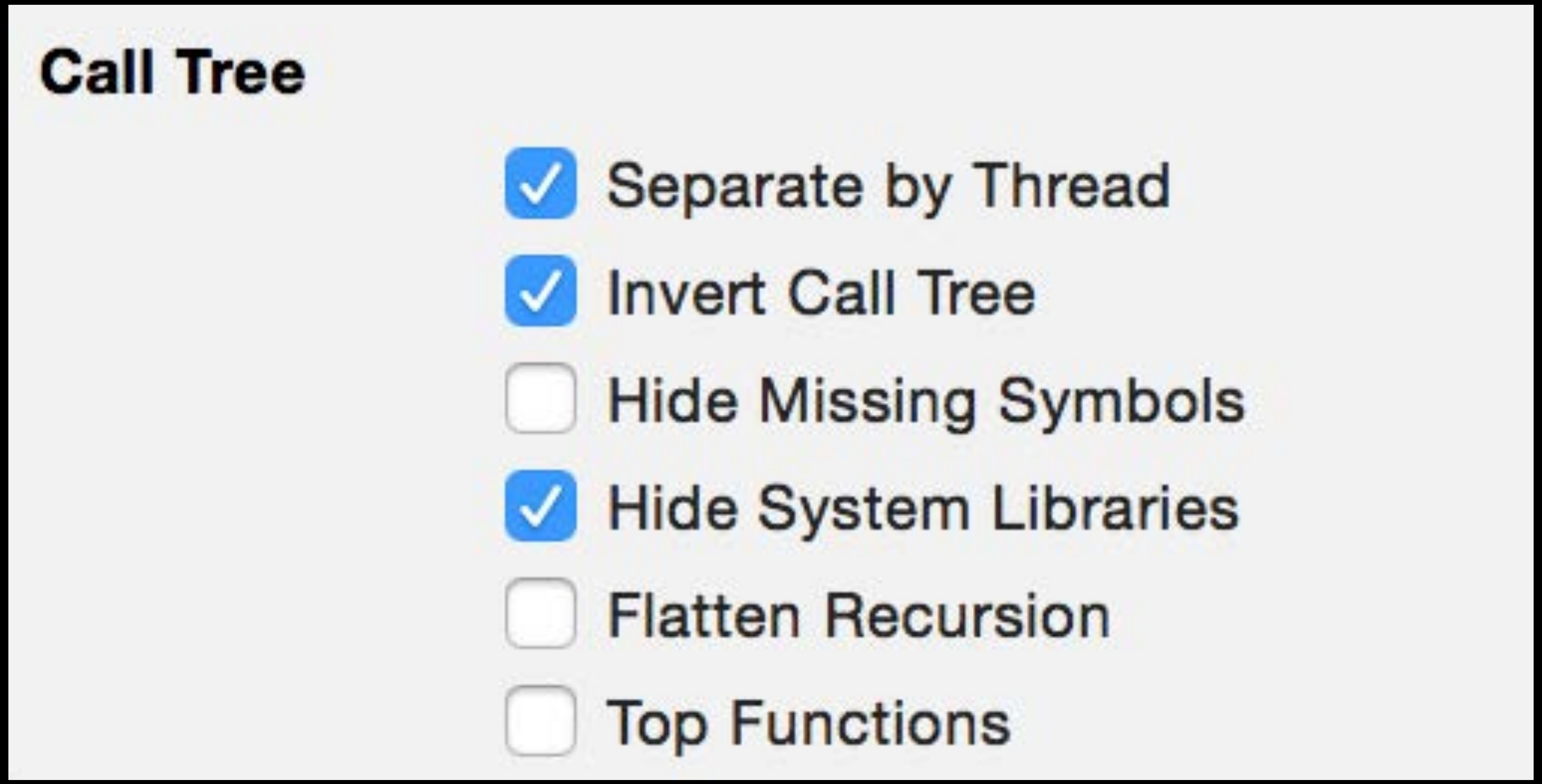



Time Profiling

✓ Record Waiting Threads

Review

Call Tree settings



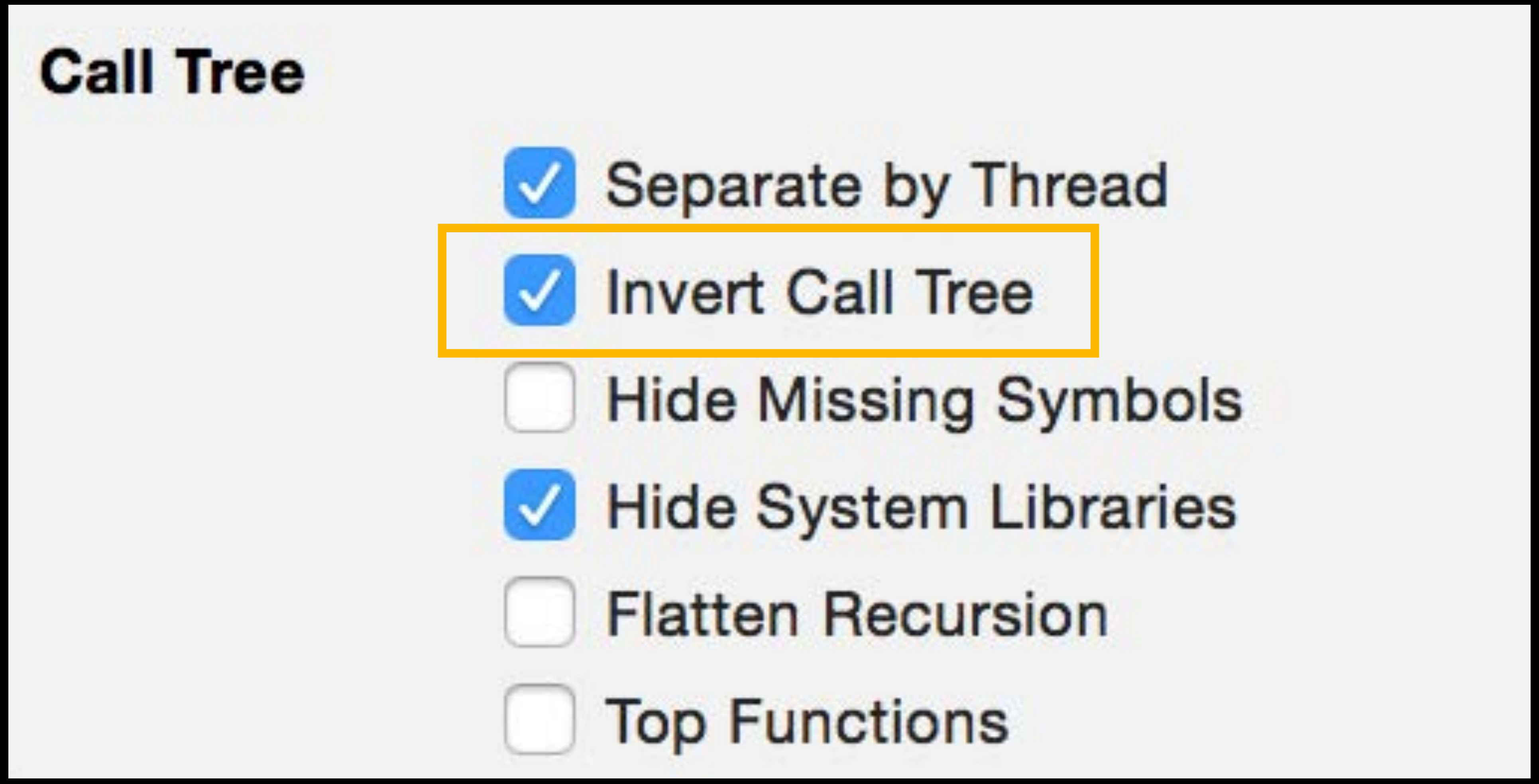
Call Tree

- ✓ Separate by Thread
- ✓ Invert Call Tree
- ☐ Hide Missing Symbols
- ✓ Hide System Libraries
- ☐ Flatten Recursion
- ☐ Top Functions

Review

Call Tree settings

- Expensive calls are frequently near the end of the call stack



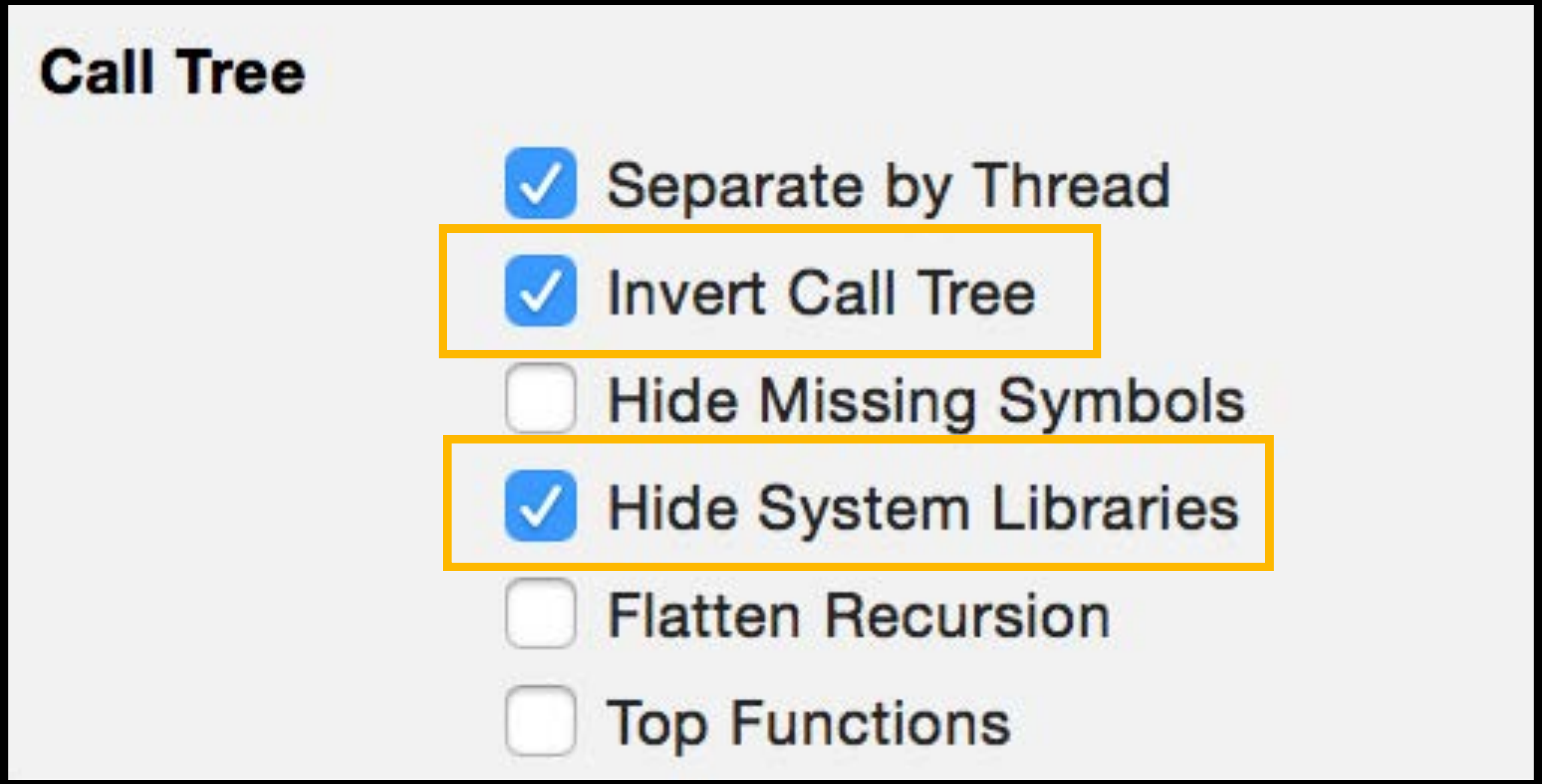
Call Tree

[✓] Separate by Thread
[✓] Invert Call Tree
[ ] Hide Missing Symbols
[✓] Hide System Libraries
[ ] Flatten Recursion
[ ] Top Functions

Review

Call Tree settings

- Expensive calls are frequently near the end of the call stack
- Focus on your own code



Call Tree

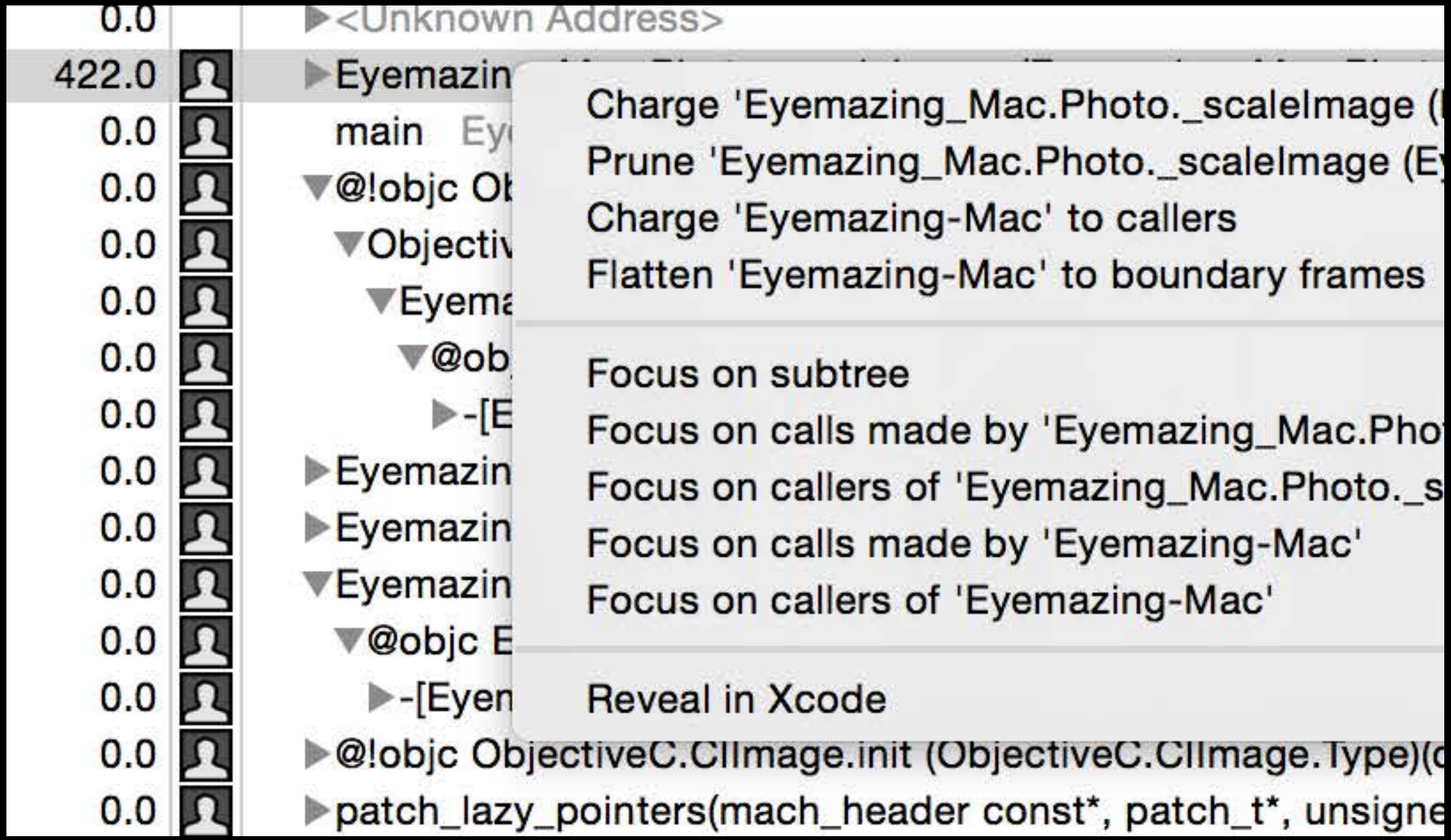
- [✓] Separate by Thread
- [✓] Invert Call Tree
- [ ] Hide Missing Symbols
- [✓] Hide System Libraries
- [ ] Flatten Recursion
- [ ] Top Functions

Tips

Focus and Prune

Ignore unwanted data

- Charge moves the associated cost
- Prune removes the associated cost
- Focus is “prune everything but”

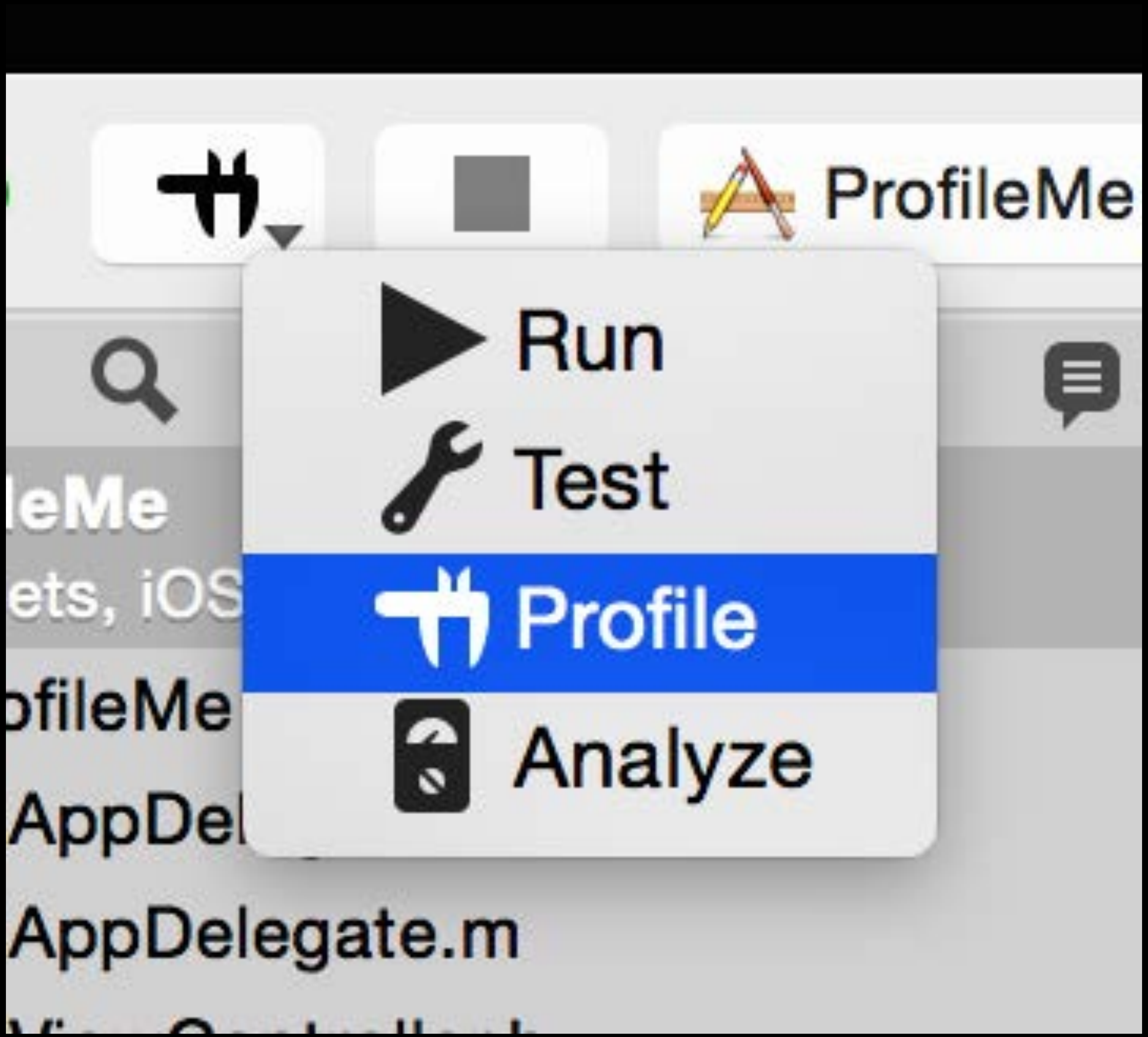


```
  0.0    ▶<Unknown Address>
422.0    ▶Eyemazin
  0.0     main  Ey
  0.0    ▼@!objc Ob
  0.0     ▼Objectiv
  0.0      ▼Eyema
  0.0       ▼@ob
  0.0        ▶-[E
  0.0    ▶Eyemazin
  0.0    ▶Eyemazin
  0.0    ▼Eyemazin
  0.0     ▼@objc E
  0.0      ▶-[Eyen
  0.0    ▶@!objc ObjectiveC.CIImage.init (ObjectiveC.CIImage.Type)(c
  0.0    ▶patch_lazy_pointers(mach_header const*, patch_t*, unsigne
```

Charge 'Eyemazing_Mac.Photo._scaleImage (
Prune 'Eyemazing_Mac.Photo._scaleImage (E
Charge 'Eyemazing-Mac' to callers
Flatten 'Eyemazing-Mac' to boundary frames
Focus on subtree
Focus on calls made by 'Eyemazing_Mac.Pho
Focus on callers of 'Eyemazing_Mac.Photo._s
Focus on calls made by 'Eyemazing-Mac'
Focus on callers of 'Eyemazing-Mac'
Reveal in Xcode

Two More Guidelines

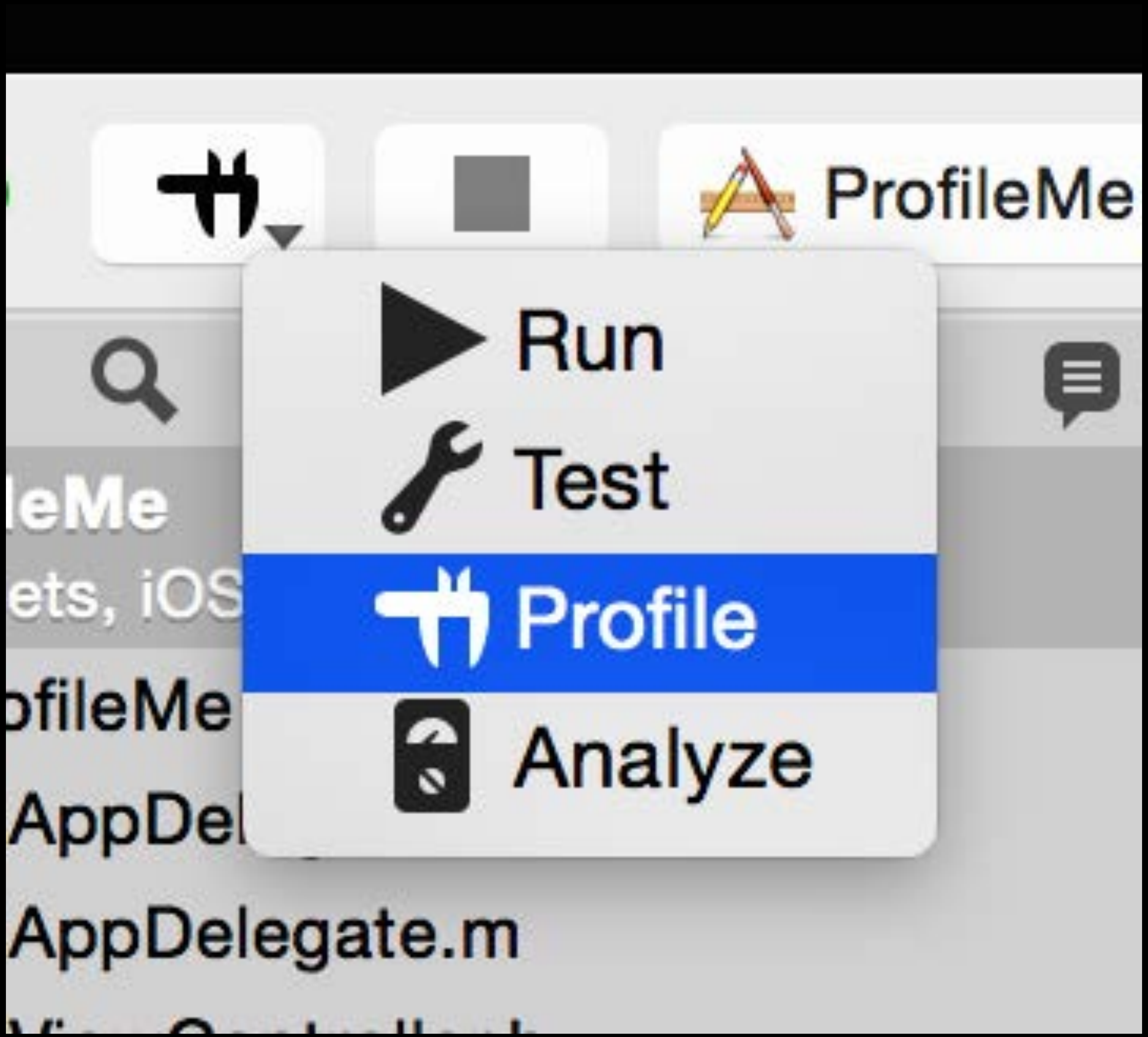
When using Time Profiler



Two More Guidelines

When using Time Profiler

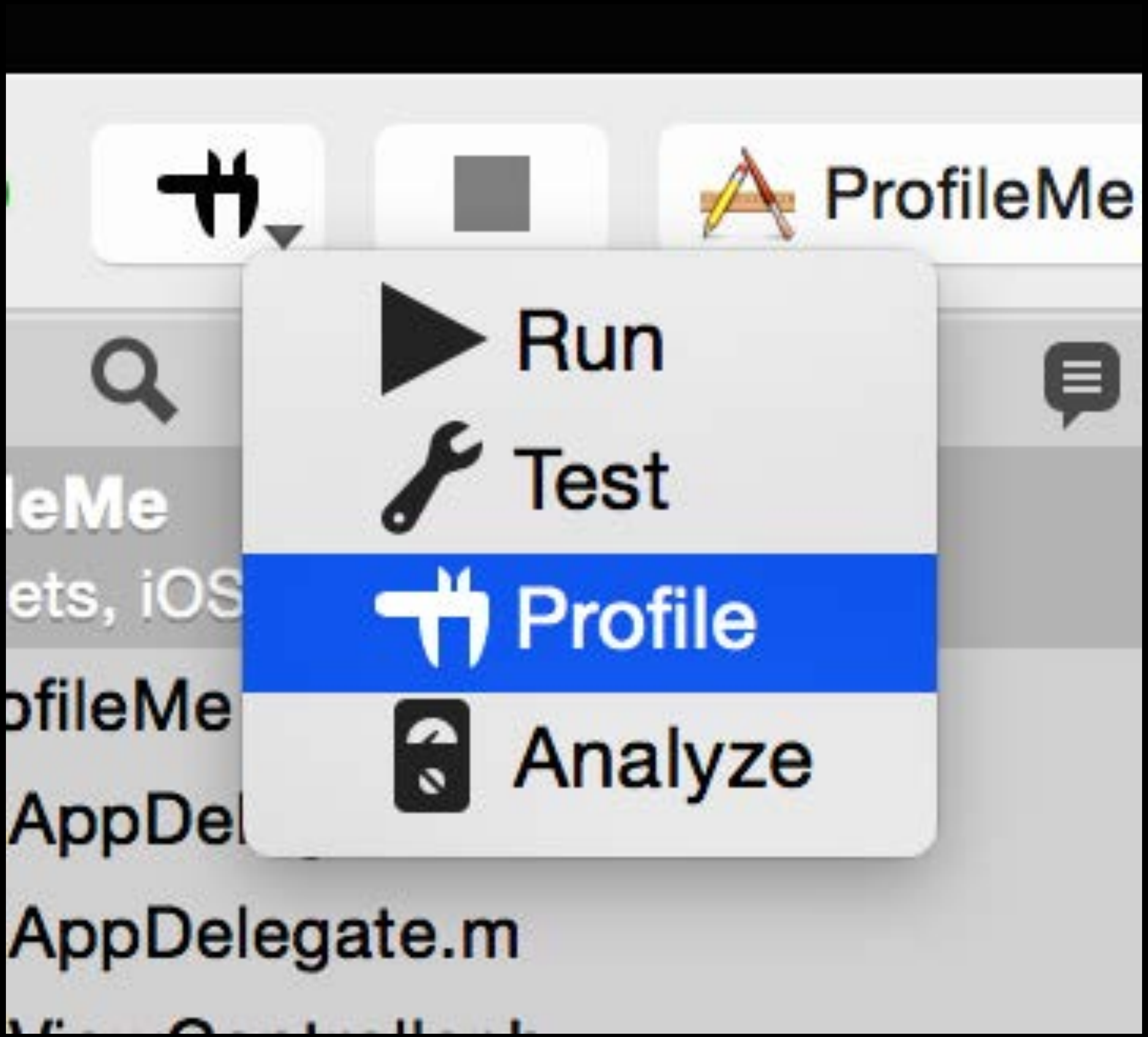
- Profile Release builds



Two More Guidelines

When using Time Profiler

- Profile Release builds
- For iOS, profile on the device



Performance Counters

Joe Grzywacz
Performance Tools Engineer

What Are Counters?

Each processor core contains a small number of 64-bit hardware registers

What Are Counters?

Each processor core contains a small number of 64-bit hardware registers

- Typically only four to eight per core

What Are Counters?

Each processor core contains a small number of 64-bit hardware registers

- Typically only four to eight per core
- Separate from the integer and floating point registers

What Are Counters?

Each processor core contains a small number of 64-bit hardware registers

- Typically only four to eight per core
- Separate from the integer and floating point registers

Each register can be configured to either:

What Are Counters?

Each processor core contains a small number of 64-bit hardware registers

- Typically only four to eight per core
- Separate from the integer and floating point registers

Each register can be configured to either:

- Count one of a small number of events

What Are Counters?

Each processor core contains a small number of 64-bit hardware registers

- Typically only four to eight per core
- Separate from the integer and floating point registers

Each register can be configured to either:

- Count one of a small number of events
    - Instructions executed, L2 Cache Misses, Branches Taken, …

What Are Counters?

Each processor core contains a small number of 64-bit hardware registers

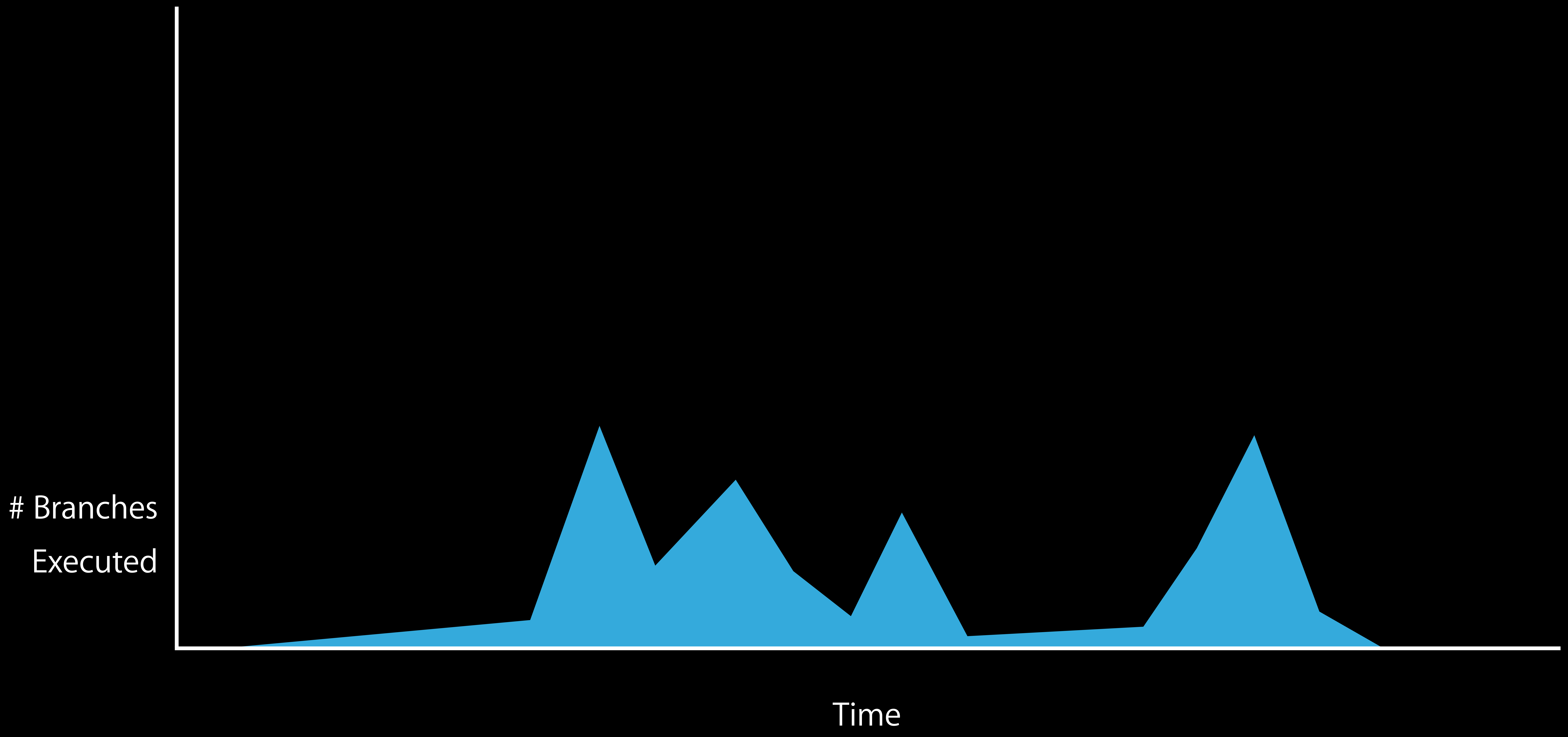
- Typically only four to eight per core
- Separate from the integer and floating point registers

Each register can be configured to either:

- Count one of a small number of events
    - Instructions executed, L2 Cache Misses, Branches Taken, …
- Take a callstack every time a predetermined number of events occurs

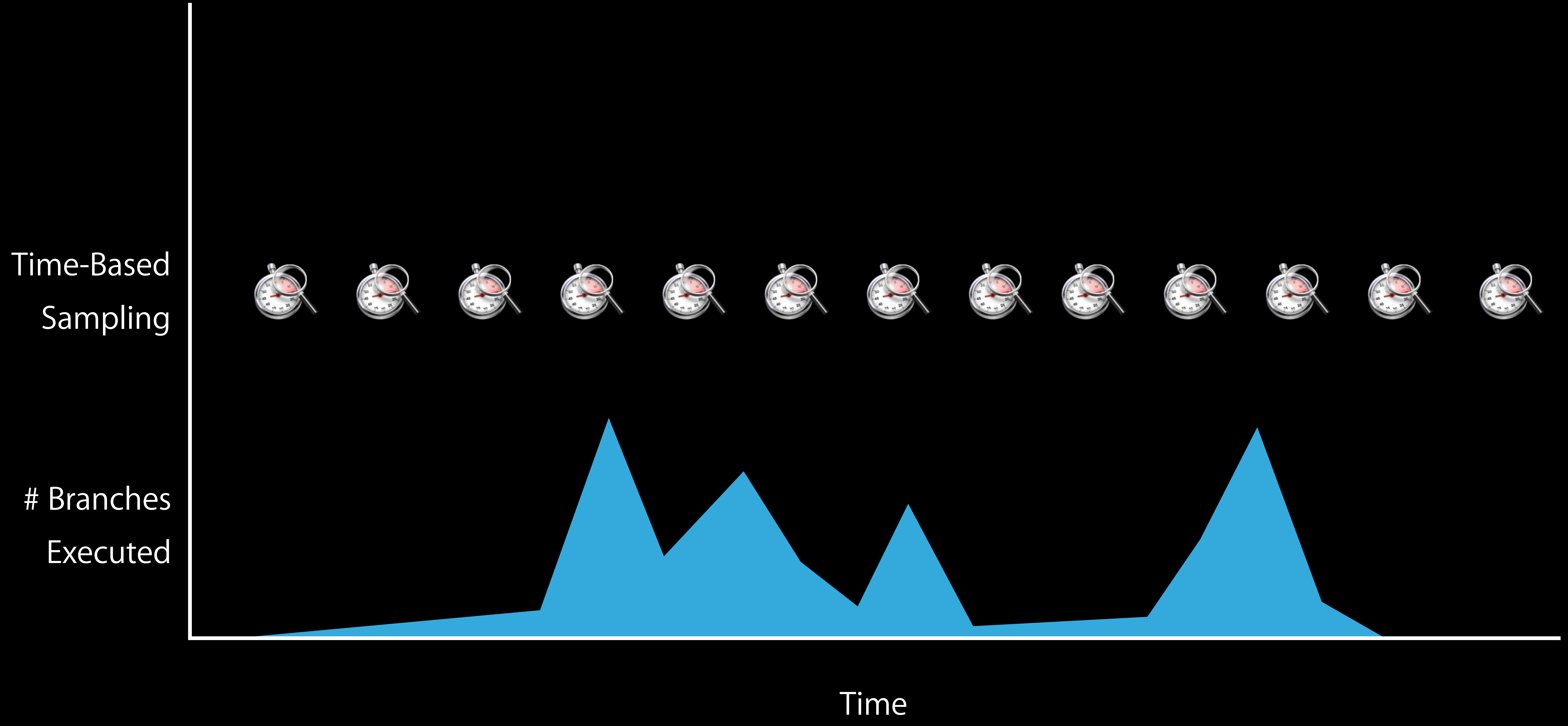
Performance Monitoring Interrupts

(PMIs)



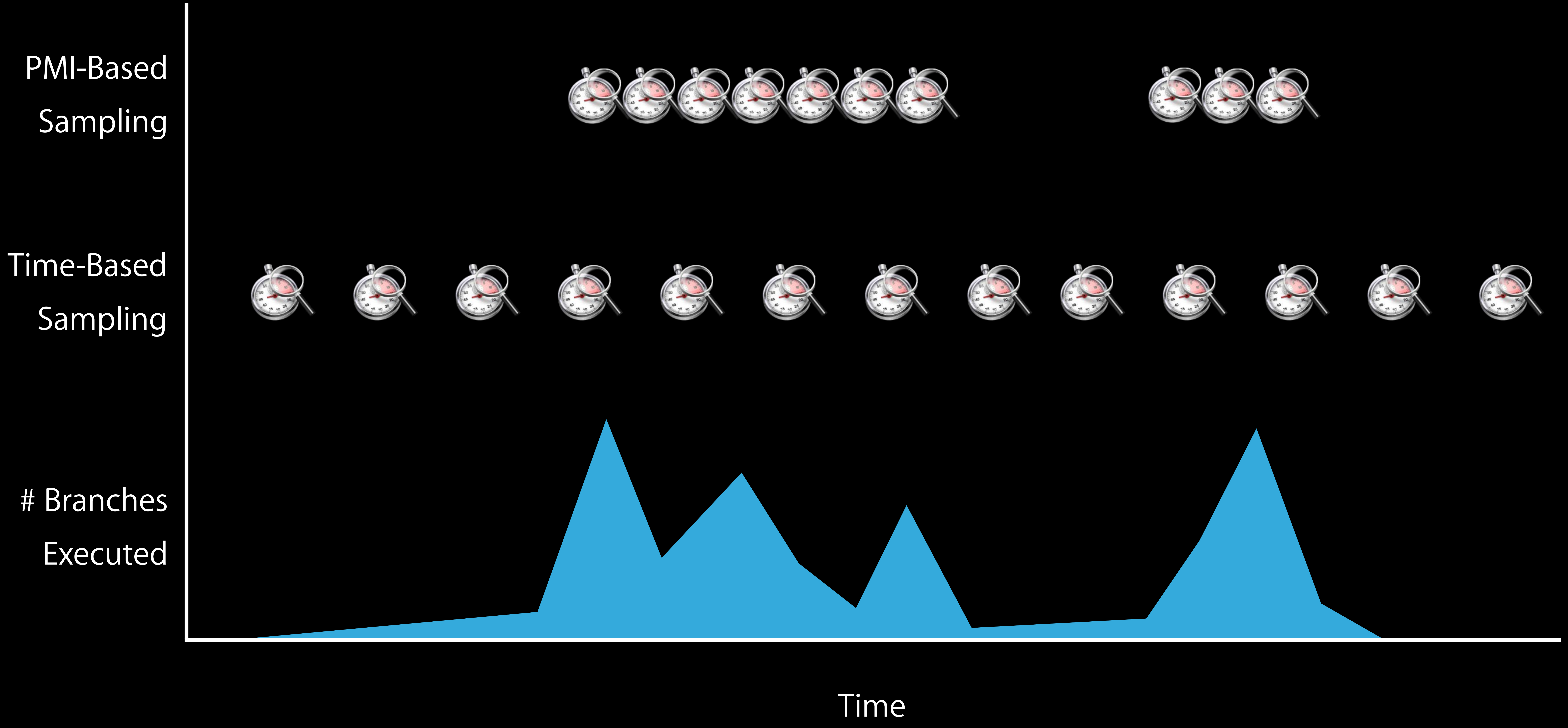
Performance Monitoring Interrupts

(PMIs)



Performance Monitoring Interrupts

(PMIs)



Performance Counters

How are they useful?

Performance Counters

How are they useful?

Provide a deeper understanding of your app's performance beyond just time

Performance Counters

How are they useful?

Provide a deeper understanding of your app's performance beyond just time

- How well CPU resources are being used
  - Caches, execution units, TLBs, …

Performance Counters
How are they useful?

Provide a deeper understanding of your app's performance beyond just time

- How well CPU resources are being used
  - Caches, execution units, TLBs, …
- Runtime process traits
  - Branch frequency, instruction mix, …

What's New with Counters

Formulas support

$$\text{IPC} = \frac{\text{Instructions}}{\text{Cycles}}$$

$$\textbf{Branch Mispredict Rate} = \frac{\text{BranchesMispredicted}}{\text{BranchesExecuted}}$$

$$\textbf{L1 Cache Miss \%} = 100 \times \frac{(\text{L1CacheLoadMisses} + \text{L1CacheStoreMisses})}{(\text{L1CacheLoads} + \text{L1CacheStores})}$$

What’s New with Counters

iOS 8 support

- 64-bit ARM devices only

What’s New with Counters

iOS 8 support

- 64-bit ARM devices only

Event Profiler instrument is deprecated

- Same PMI functionality is available within the Counters instrument

Demo

iOS Performance Counters

Counters Summary

Counters Summary

Collects data in a similar manner to Time Profiler

Counters Summary

Collects data in a similar manner to Time Profiler

- This is a statistical representation of your application

Counters Summary

Collects data in a similar manner to Time Profiler

- This is a statistical representation of your application

Counters supports Performance Monitoring Interrupts (PMI)

Counters Summary

Collects data in a similar manner to Time Profiler

- This is a statistical representation of your application

Counters supports Performance Monitoring Interrupts (PMI)

- Allows sampling based on the number of events

Counters Summary

Collects data in a similar manner to Time Profiler

- This is a statistical representation of your application

Counters supports Performance Monitoring Interrupts (PMI)

- Allows sampling based on the number of events
- Note that PMI instruction locations can be imprecise

Counters Summary

Collects data in a similar manner to Time Profiler

- This is a statistical representation of your application

Counters supports Performance Monitoring Interrupts (PMI)

- Allows sampling based on the number of events
- Note that PMI instruction locations can be imprecise

Formulas allow you to combine raw event counts in custom ways

Counters Summary

Collects data in a similar manner to Time Profiler

- This is a statistical representation of your application

Counters supports Performance Monitoring Interrupts (PMI)

- Allows sampling based on the number of events
- Note that PMI instruction locations can be imprecise

Formulas allow you to combine raw event counts in custom ways

- Be sure to save your common formulas in a template

What’s New with DTrace

What's New with DTrace

Dynamic tracemem

Dynamic tracemem, provides a way to trace dynamically sized arrays

- tracemem(address, nbytes_max, nbytes)
  - nbytes_max: maximum size of the array, must be known at compile time
  - nbytes: the actual size of the array you want to copy
  - Example:

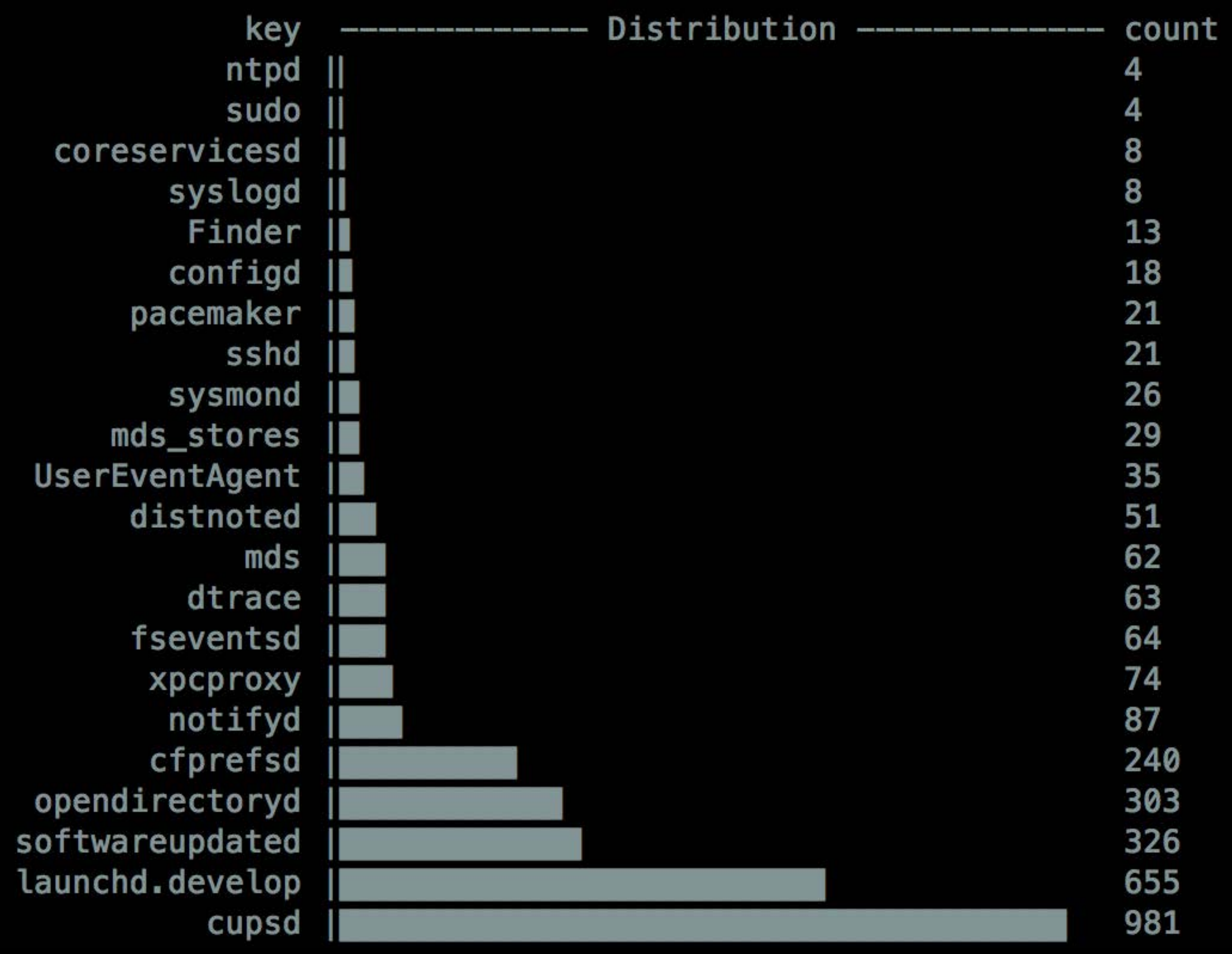
```
void CGContextFillRects(CGContextRef c, const CGRect rects[], size_t count);

pid$pid_MyAppName::CGContextFillRects:entry
{
 this->array = copyin(arg1, sizeof(struct CGRect) * arg2);
 tracemem(this->array, 512, sizeof(struct CGRect) * arg2);
}
```

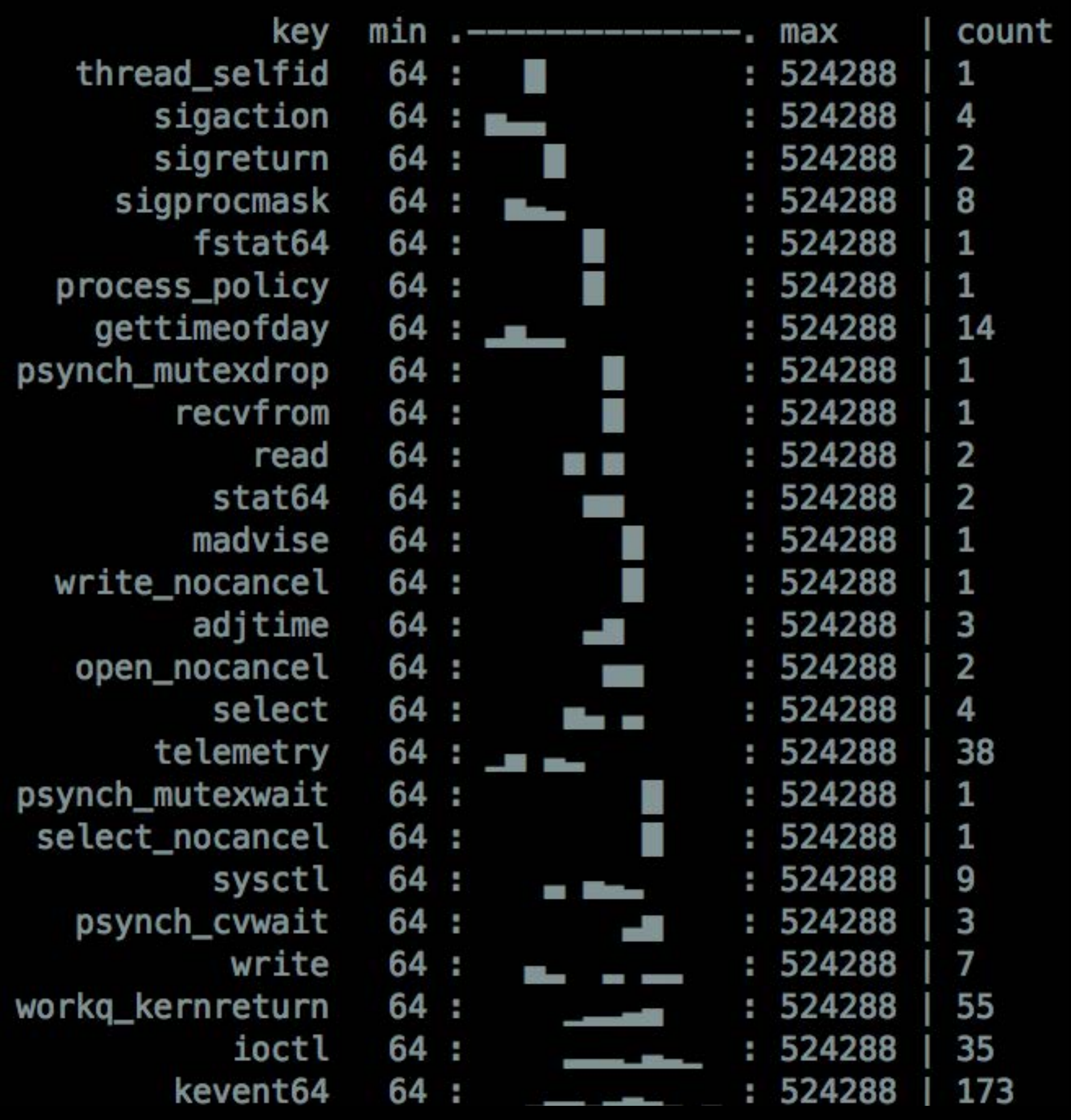
Improved Histograms

Histogram improvements: `agghist`, `aggzoom`, `aggpack`

http://dtrace.org/blogs/bmc/2013/11/10/agghist-aggzoom-and-aggpack/



```
            key  ------------- Distribution -------------  count
           ntpd |                                          4
           sudo |                                          4
  coreservicesd |                                          8
        syslogd |                                          8
         Finder |                                          13
        configd |                                          18
      pacemaker |                                          21
           sshd |                                          21
        sysmond |                                          26
     mds_stores |                                          29
 UserEventAgent |                                          35
      distnoted |                                          51
            mds |                                          62
         dtrace |                                          63
      fseventsd |                                          64
       xpcproxy |                                          74
        notifyd |                                          87
       cfprefsd |                                          240
opendirectoryd  |                                          303
softwareupdated |                                          326
launchd.develop |                                          655
          cupsd |                                          981
```



```
             key min .---------------. max    | count
   thread_selfid  64 :               : 524288 | 1
       sigaction  64 :               : 524288 | 4
       sigreturn  64 :               : 524288 | 2
     sigprocmask  64 :               : 524288 | 8
         fstat64  64 :               : 524288 | 1
  process_policy  64 :               : 524288 | 1
    gettimeofday  64 :               : 524288 | 14
psynch_mutexdrop  64 :               : 524288 | 1
        recvfrom  64 :               : 524288 | 1
            read  64 :               : 524288 | 2
          stat64  64 :               : 524288 | 2
         madvise  64 :               : 524288 | 1
   write_nocancel 64 :               : 524288 | 1
         adjtime  64 :               : 524288 | 3
    open_nocancel 64 :               : 524288 | 2
          select  64 :               : 524288 | 4
       telemetry  64 :               : 524288 | 38
psynch_mutexwait  64 :               : 524288 | 1
  select_nocancel 64 :               : 524288 | 1
          sysctl  64 :               : 524288 | 9
    psynch_cvwait 64 :               : 524288 | 3
           write  64 :               : 524288 | 7
  workq_kernreturn 64 :              : 524288 | 55
           ioctl  64 :               : 524288 | 35
        kevent64  64 :               : 524288 | 173
```

Other New Features

Wait for process to start with –W

```
dtrace –Z –W MyAppName ‘pid$target::*CALayer*:entry’
```

Other New Features

Wait for process to start with –W

```
dtrace –Z –W MyAppName ‘pid$target::*CALayer*:entry’
```

Tunable internal DTrace variables

```
# List the tunable variables
sysctl kern.dtrace
```

Other New Features

Wait for process to start with –W

```
dtrace –Z –W MyAppName ‘pid$target::*CALayer*:entry’
```

Tunable internal DTrace variables

```
# List the tunable variables
sysctl kern.dtrace
```

Updated documentation

```
man dtrace
```

Summary

Profile Swift and Objective-C alike

Be proactive

Don't assume—profile, change, and iterate

More Information

Dave DeLong
Developer Tools Evangelist
delong@apple.com

Instruments Documentation
Instruments User Guide
Instruments User Reference
http://developer.apple.com

Apple Developer Forums
http://devforums.apple.com

Related Sessions

● Creating Extensions for iOS and OS X, Part 1	Mission	Tuesday 2:00PM
● Integrating Swift with Objective-C	Presidio	Wednesday 9:00AM
● Creating Extensions for iOS and OS X, Part 2	Mission	Wednesday 11:30AM
● Swift Interoperability In-Depth	Presidio	Wednesday 3:15PM
● Advanced Swift Debugging in LLDB	Mission	Friday 9:00AM

Labs

●	Swift Lab	Tools Lab A	All Week
●	Instruments Lab	Tools Lab B	Wednesday 9:00AM
●	Power and Performance Lab	Core OS Lab B	Wednesday 2:00PM
●	Instruments Lab	Tools Lab B	Thursday 9:00AM
●	Power and Performance Lab	Core OS Lab A	Thursday 3:15PM

WWDC14