App Performance

iPhone and iPod touch Development
Fall 2009 — Lecture 27

Announcements

- Each of the 3 project deliverables were posted to the course blog last week (more details on this in a moment...)
- Added a post earlier today which details easy ways of acquiring screenshots off the simulator or a device
- Determination of final presentation & demo ordering before we get started with tonight's material

Final Project Deliverables

- Monday, December 14th
  - Project source code must be submitted by 11:59 pm
- Tuesday, December 15th
  - Special office hours (5:30 – 6:45 pm) if you wish to test your project on the demo hardware
  - Mock App Store page must be submitted by 11:59 pm
- Wednesday, December 16th
  - Presentation slides must be submitted by 11:59 pm
- Thursday, December 17th
  - Presentations and demos 6:00 – 8:00 pm

Questions?

Today's Topics

- Performance
  - Device vs. Simulator
  - Shark
  - Instruments
- Detecting Memory Leaks
  - Instruments
- Instruments Odd & Ends
- Zombies

Notes

- I'm showing the relevant portions of the view controller interfaces and implementations in these notes
- Remember to release relevant memory in the -dealloc methods — they are not shown here
- You will also need to wire up outlets and actions in IB
- Where delegates are used, they too require wiring in IB

Performance

“premature optimization is the root of all evil”

— Donald Knuth

Dan's Interpretation of what this Means

- Design (while avoiding higher-order approaches)
  - We already know that designing something that's $O(n^3)$ is probably not going to perform well
- Code based on the design
- Then, profile & benchmark to find bottlenecks
- Interpret results, re-design, re-code, re-profile, repeat

Dan's Interpretation of what this Means

- Usually no sense in spending hours getting that last little bit of performance out of something if it's only responsible for 1% of the overall cost
  - Get the most bang for your buck
- There's often a trade between readability vs. optimization
  - If you waste time "optimizing" something that's of little return, then may have done more harm than good if the code is not easily understandable and maintainable

iPhone Profiling Tools

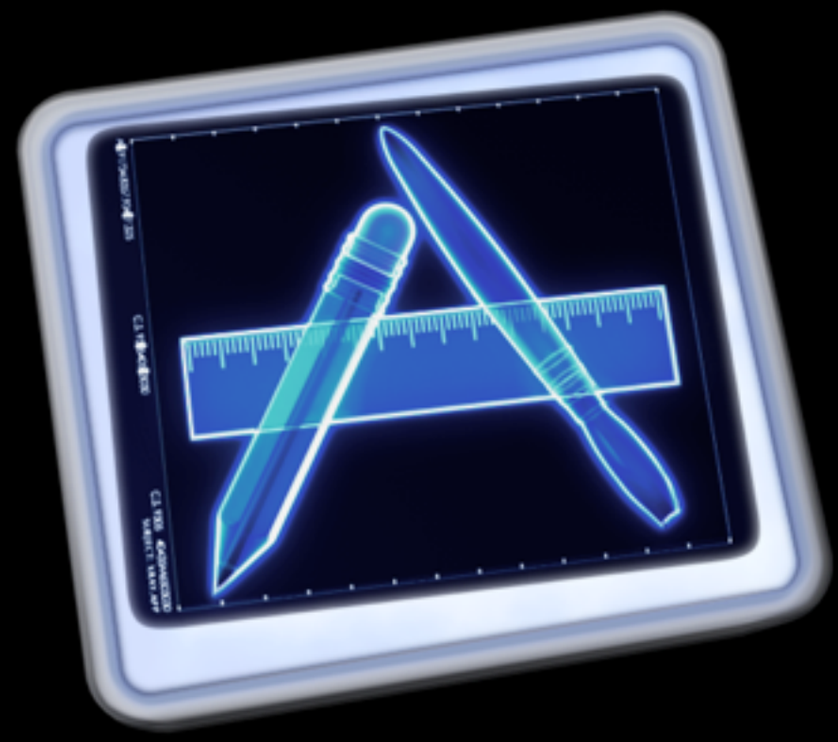
- Apple has provided several profiling tools as part of the iPhone SDK...
  - Shark
  - Instruments

Shark

- At the simplest level, Shark profiles the system while your code is running to see where time is being spent
- It can also produce profiles of hardware and software performance events such as cache misses, virtual memory activity, memory allocations, function calls, or instruction dependency stalls
- This information is an invaluable first step in your performance tuning effort so you can see which parts of your code or the system are the bottlenecks

Instruments

- Instruments provides a data gathering interface that lets you know how your app uses resources, such as...
  - CPU
  - Memory
  - File system
  - Network utilization
  - etc...



An Instrument

- Instruments uses software-based data gathering tools, each known as an instrument, to collect performance data
- Each instrument collects a specific type of data, such as network activity or memory usage
- You can drag and drop an instrument from a library of tools, much like interface builder

DTrace

- Instruments is backed by the DTrace utility was created by Sun Microsystems originally for Solaris
  - Licensed under Sun's open source CDDL and ported to other UNIXes including Mac OS X
- DTrace scripts (which look like a cross between C and awk) are used to define which probes to act on
- DTrace was designed to...
  - Minimize probe effect when tracing is active
  - Have no performance impact for a disabled probe
- Instruments allows you to create custom DTrace intruments

Performance Example

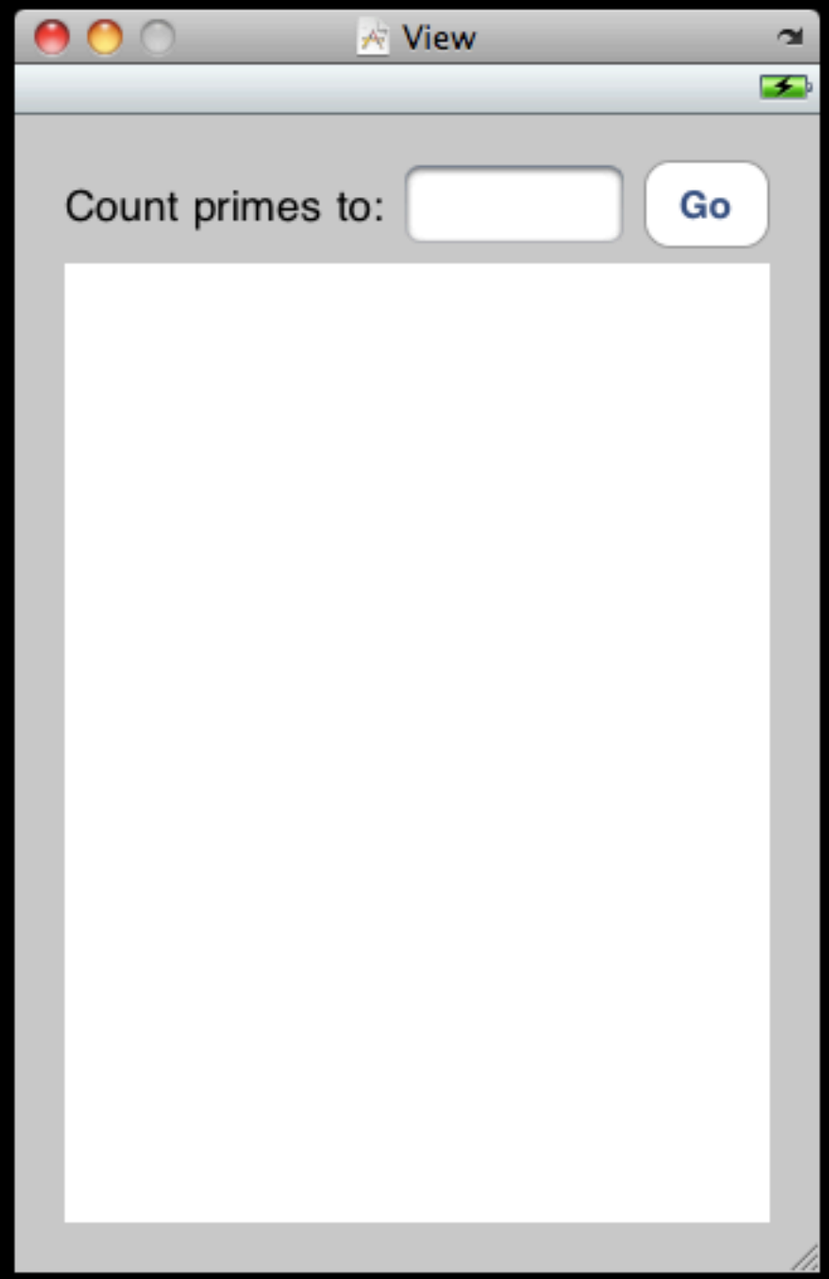
Adapted from Apache Licensed PathologicalPrimeCounter code from iPhone SDK Development

Prime Number Finder Example

- For this example, we'll use a small app that allows the user to enter a number, and the app will find all prime numbers up (including) the user specified number

PrimesViewController.xib

- Our interface for this example is pretty simple...
  - A text field for the user to enter the number they want to go up to
  - A "Go" button which will call an action to find all primes in range
  - A multi-line text field where our app will dump the primes



PrimesViewController.h

```
#import <UIKit/UIKit.h>

@interface PrimesViewController : UIViewController <UITextFieldDelegate> {

  UITextField *primesUpTo;
  UITextView *primesView;

}
@property(nonatomic, retain) IBOutlet UITextField *primesUpTo;
@property(nonatomic, retain) IBOutlet UITextView *primesView;

- (IBAction)findPrimeNumbers;

@end
```

PrimesViewController.m

```objc
#import "PrimesViewController.h"

@implementation PrimesViewController

@synthesize primesUpTo;
@synthesize primesView;

// naive prime number check
- (BOOL)isPrime:(int)n {
  if (n < 2) {
    return NO;
  }
  for (int i = 2; i < n; i++) {
    if (n % i == 0) {
      return NO;
    }
  }
  return YES;
}

// ...
```

PrimesViewController.m

```
// ...

- (IBAction)findPrimeNumbers {
  [self.primesUpTo resignFirstResponder];
  self.primesView.text = @"";
  int max = [self.primesUpTo.text intValue];
  for (int i = 0; i <= max; i++) {
  if ([self isPrime: i]) {
    self.primesView.text = [self.primesView.text stringByAppendingFormat:@"%d ", i];
    }
  }
}

- (BOOL)textFieldShouldReturn:(UITextField *)textField {
  [self findPrimeNumbers];
  return YES;
}

- (void)dealloc {
  self.primesUpTo = nil;
  self.primesView = nil;
  [super dealloc];
}

@end
```

Guesses on Performance

- Any guesses on how this is going to do?

Performance in iPhone Simulator

- When running on my unibody MacBook Pro it took...
- About 2.5 seconds to get through 10,000 numbers
- About 50 seconds to get through 50,000 numbers

Performance on Device

- Whereas, running on my iPhone 3G, it took about 80 seconds to get through 10,000 numbers
- That's nearly 32 times slower!
- The moral of this story...
- Test on an actual device to accurately gauge performance!

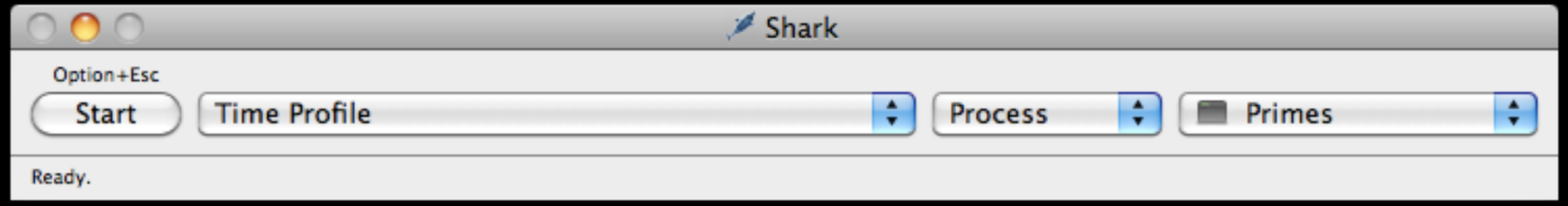
≠

Sampling with Shark

- The Shark profiling tool can be found under the iPhone SDK install at...
  - /Developer/Applications/Performance Tools/Shark.app
- Or, you can launch via Spotlight

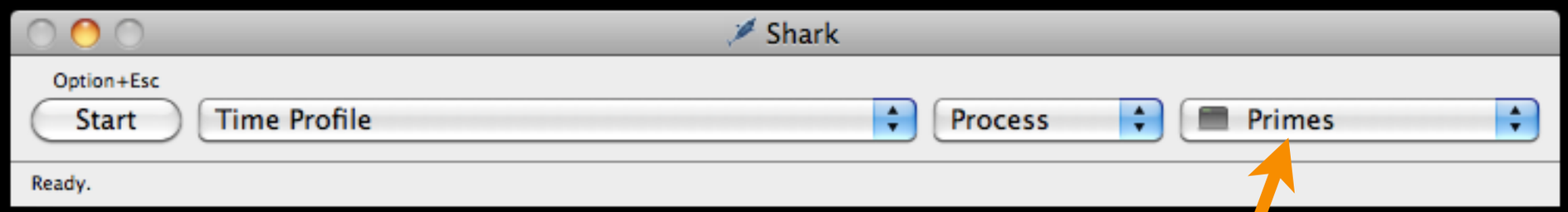
Shark UI

- When you launch Shark, you're presented with the following, rather minimalistic UI...



Starting Shark

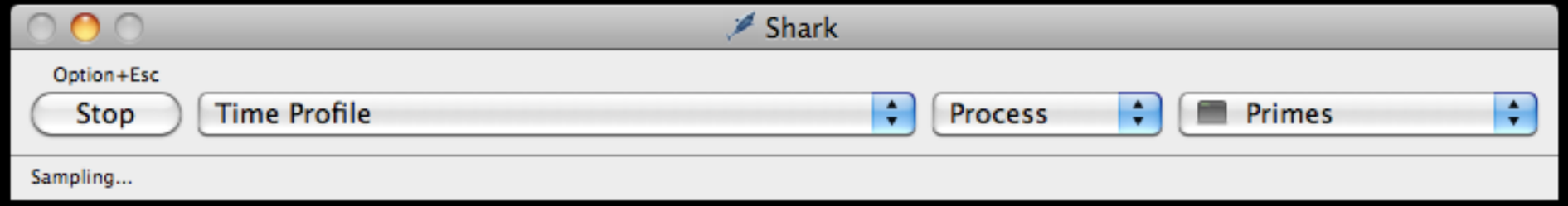
- In order to profile your app, it must first be running, so go ahead and start up your app in the iPhone Simulator
- Then, select the process from the right-most drop down and when you're ready to start sampling press Start



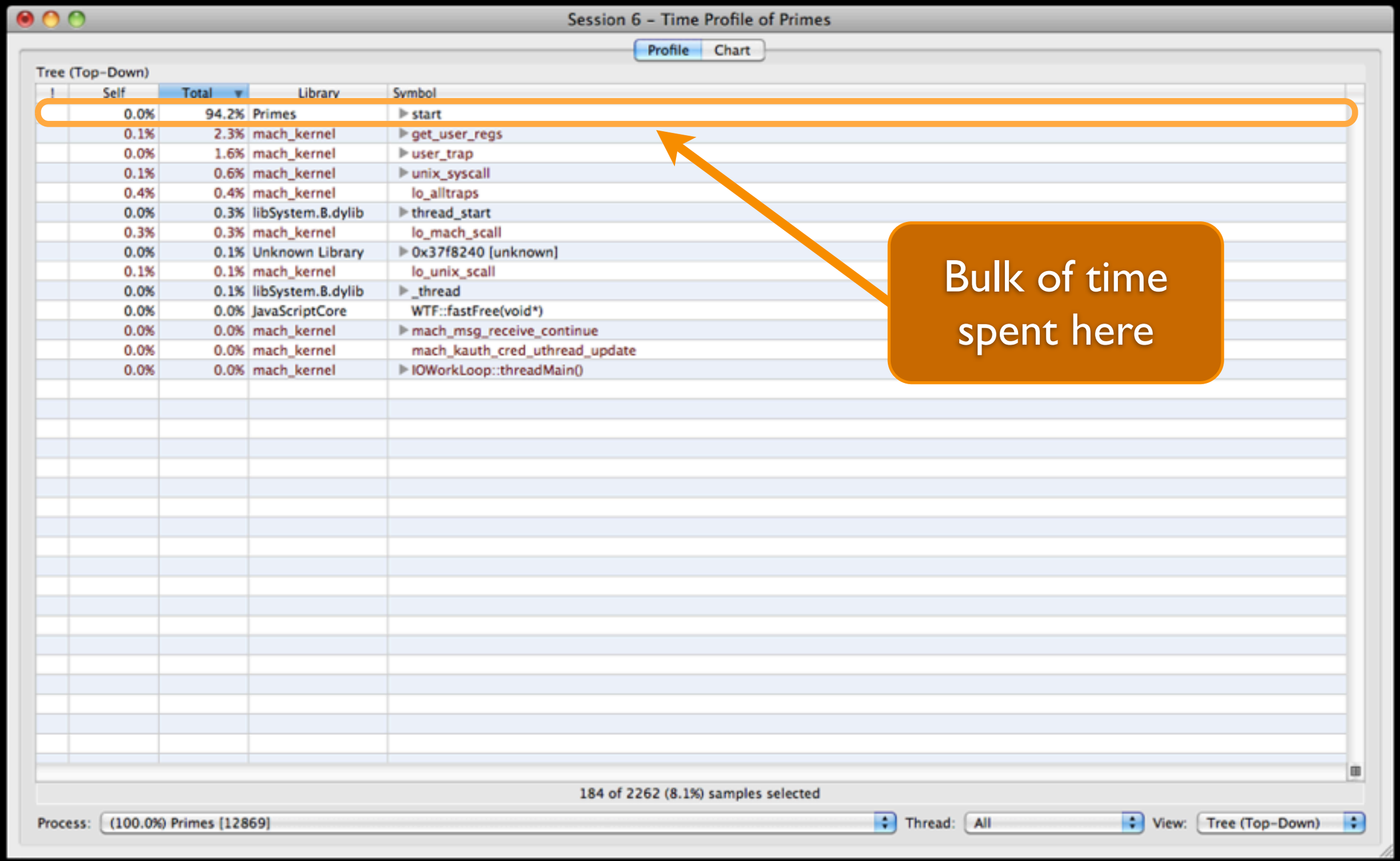
Select your app from this list

Stopping Shark

- Once you're done exercising the section of code you're interested in profiling press the stop button to have Shark aggregate and display the results...



Top Down View



Top Down View

Session 6 – Time Profile of Primes

Profile | Chart

Tree (Top-Down)

!	Self	Total	Library	Symbol
	0.0%	94.2%	Primes	start
	0.0%	94.2%	Primes	main
	0.0%	94.2%	UIKit	UIApplicationMain
	0.0%	94.2%	GraphicsServices	GSEventRun
	0.0%	94.2%	GraphicsServices	GSEventRunModal
	0.0%	94.2%	CoreFoundation	CFRunLoopRunInMode
	0.0%	94.2%	CoreFoundation	CFRunLoopRunSpecific
	0.0%	92.7%	GraphicsServices	PurpleEventCallback
	0.0%	92.6%	UIKit	_UIApplicationHandleEvent
	0.0%	91.9%	UIKit	-[UIApplication sendEvent:]
	0.0%	91.2%	UIKit	-[UIWindow _sendTouchesForEvent:]
	0.0%	91.2%	UIKit	-[UIControl touchesEnded:withEvent:]
	0.0%	91.2%	UIKit	-[UIControl(Internal) _sendActionsForEvents:withEvent:]
	0.0%	91.2%	UIKit	-[UIControl sendAction:to:forEvent:]
	0.0%	91.2%	UIKit	-[UIApplication sendAction:to:from:forEvent:]
	0.0%	91.2%	Primes	-[PrimesViewController findPrimeNumbers]
	0.0%	0.5%	UIKit	-[UIWindow sendEvent:]
	0.0%	0.2%	UIKit	-[UIApplication handleEvent:withNewEvent:]
	0.0%	0.7%	Foundation	NSPopAutoreleasePool
	0.0%	0.0%	UIKit	-[UIView(Geometry) hitTest:withEvent:]
	0.0%	0.0%	libobjc.A.dylib	objc_msgSend
	0.0%	0.8%	CoreFoundation	__CFRunLoopDoObservers
	0.0%	0.3%	Foundation	__NSFireTimer
	0.0%	0.1%	CoreFoundation	__CFRunLoopModeFindSourceForMachPort
	0.0%	0.1%	libSystem.B.dylib	mach_msg
	0.0%	0.0%	libSystem.B.dylib	OSAtomicCompareAndSwapIntBarrier
	0.0%	0.0%	CoreFoundation	CFRunLoopTimerInvalidate
	0.0%	0.0%	QuartzCore	CA::timer_callback(__CFRunLoopTimer*, void*)
	0.0%	0.0%	CoreFoundation	_CFRelease
	0.0%	0.0%	libSystem.B.dylib	__spin_lock
	0.1%	2.3%	mach_kernel	get_user_regs
	0.0%	1.6%	mach_kernel	user_trap
	0.1%	0.6%	mach_kernel	unix_syscall

Bulk of time spent in findPrimeNumbers

No samples selected, 2262 of 2262 (100.0%) samples displayed

Process: (100.0%) Primes [12869] Thread: All View: Tree (Top-Down)

Top Down View

Session 6 – Time Profile of Primes

Profile | Chart

Tree (Top-Down)

!	Self	Total	Library	Symbol
	0.0%	94.2%	Primes	start
	0.0%	94.2%	Primes	main
	0.0%	94.2%	UIKit	UIApplicationMain
	0.0%	94.2%	GraphicsServices	GSEventRun
	0.0%	94.2%	GraphicsServices	GSEventRunModal
	0.0%	94.2%	CoreFoundation	CFRunLoopRunInMode
	0.0%	94.2%	CoreFoundation	CFRunLoopRunSpecific
	0.0%	92.7%	GraphicsServices	PurpleEventCallback
	0.0%	92.6%	UIKit	_UIApplicationHandleEvent
	0.0%	91.9%	UIKit	-[UIApplication sendEvent:]
	0.0%	91.2%	UIKit	-[UIWindow _sendTouchesForEvent:]
	0.0%	91.2%	UIKit	-[UIControl touchesEnded:withEvent:]
	0.0%	91.2%	UIKit	-[UIControl(Internal) _sendActionsForEvents:withEvent:]
	0.0%	91.2%	UIKit	-[UIControl sendAction:to:forEvent:]
	0.0%	91.2%	UIKit	-[UIApplication sendAction:to:from:forEvent:]
	0.0%	91.2%	Primes	-[PrimesViewController findPrimeNumbers]
	0.0%	72.0%	UIKit	-[UITextView setText:]
	0.0%	18.1%	UIKit	-[UITextView text]
	0.8%	0.8%	Primes	-[PrimesViewController isPrime:]
	0.0%	0.2%	Foundation	-[NSStri[illegible]stringByAppendingFormat:]
	0.0%	0.0%	libobjc.A.dylib	objc_msg[illegible]d
	0.0%	0.0%	UIKit	-[UITextField r[illegible]ignFirstResponder]
	0.0%	0.0%	Foundation	-[NSCFString lengt[illegible]
	0.0%	0.5%	UIKit	-[UIWindow sendEvent:]
	0.0%	0.2%	UIKit	-[UIApplication handleEvent:withNewE[illegible]nt:]
	0.0%	0.7%	Foundation	NSPopAutoreleasePool
	0.0%	0.0%	UIKit	-[UIView(Geometry) hitTest:withEvent:]
	0.0%	0.0%	libobjc.A.dylib	objc_msgSend
	0.0%	0.8%	CoreFoundation	__CFRunLoopDoObservers
	0.0%	0.3%	Foundation	__NSFireTimer
	0.0%	0.1%	CoreFoundation	__CFRunLoopModeFindSourceForMachPort
	0.0%	0.1%	libSystem.B.dylib	mach_msg
	0.0%	0.0%	libSystem.B.dylib	OSAtomicCompareAndSwapIntBarrier

What? Only .8% of time spent in isPrime:!

18 of 2262 (0.8%) samples selected

Process: (100.0%) Primes [12869] Thread: All View: Tree (Top-Down)

Top Down View

Session 6 – Time Profile of Primes

Profile | Chart

Tree (Top-Down)

!	Self	Total	Library	Symbol
	0.0%	94.2%	Primes	start
	0.0%	94.2%	Primes	main
	0.0%	94.2%	UIKit	UIApplicationMain
	0.0%	94.2%	GraphicsServices	GSEventRun
	0.0%	94.2%	GraphicsServices	GSEventRunModal
	0.0%	94.2%	CoreFoundation	CFRunLoopRunInMode
	0.0%	94.2%	CoreFoundation	CFRunLoopRunSpecific
	0.0%	92.7%	GraphicsServices	PurpleEventCallback
	0.0%	92.6%	UIKit	_UIApplicationHandleEvent
	0.0%	91.9%	UIKit	-[UIApplication sendEvent:]
	0.0%	91.2%	UIKit	-[UIWindow _sendTouchesForEvent:]
	0.0%	91.2%	UIKit	-[UIControl touchesEnded:withEvent:]
	0.0%	91.2%	UIKit	-[UIControl(Internal) _sendActionsForEvents:withEvent:]
	0.0%	91.2%	UIKit	-[UIControl sendAction:to:forEvent:]
	0.0%	91.2%	UIKit	-[UIApplication sendAction:to:from:forEvent:]
	0.0%	91.2%	Primes	-[PrimesViewController findPrimeNumbers]
	0.0%	72.0%	UIKit	-[UITextView setText:]
	0.0%	18.1%	UIKit	-[UITextView text]
	0.8%	0.8%	Primes	-[PrimesViewC…eller isPrime:]
	0.0%	0.2%	Foundation	-[NSString string…ppendingFormat:]
	0.0%	0.0%	libobjc.A.dylib	objc_msgSend
	0.0%	0.0%	UIKit	-[UITextField resignFirstR…sponder]
	0.0%	0.0%	Foundation	-[NSCFString length]
	0.0%	0.5%	UIKit	-[UIWindow sendEven…
	0.0%	0.2%	UIKit	-[UIApplication handl…
	0.0%	0.7%	Foundation	NSPopAutoreleasePool
	0.0%	0.0%	UIKit	-[UIView(Geometry) hitT…
	0.0%	0.0%	libobjc.A.dylib	objc_msgSend
	0.0%	0.8%	CoreFoundation	__CFRunLoopDoObservers
	0.0%	0.3%	Foundation	__NSFireTimer
	0.0%	0.1%	CoreFoundation	__CFRunLoopModeFindSour…
	0.0%	0.1%	libSystem.B.dylib	mach_msg
	0.0%	0.0%	libSystem.B.dylib	OSAtomicCompareAndSwap…

Bulk of time in UITextView's setText: method, good chunk also in UITextView's test method

18 of 2262 (0.8%) samples selected

Process: (100.0%) Primes [12869] Thread: All View: Tree (Top-Down)

Bottom-Up View

Session 6 – Time Profile of Primes

Profile | Chart

Heavy (Bottom-Up)

!	Self ▼	Total	Library	Symbol
	11.6%	11.6%	WebCore	▶ WebCore::WidthIterator::advance(int, WebCore::GlyphBuffer*)
	9.1%	9.1%	WebCore	▶ WebCore::StringImpl::find(unsigned short, int)
	6.9%	6.9%	CoreFoundation	▶ CFStringFindWithOptionsAndLocale
	6.1%	6.1%	WebCore	▶ WebCore::BidiResolver<WebCore::InlineIterator, WebCore::BidiRun>::createBidiRunsForLine(WebCore::InlineIterator const&, bool, bool)
	5.0%	5.0%	WebCore	▶ WebCore::RenderBlock::findNextLineBreak(WebCore::BidiResolver<WebCore::InlineIterator, WebCore::BidiRun>&, WebCore::EClear*)
	3.7%	3.7%	WebCore	▶ WebCore::Font::glyphDataForCharacter(int, bool, bool) const
	3.4%	3.4%	WebCore	▶ WebCore::SimpleFontData::widthForGlyph(unsigned short) const
	3.4%	3.4%	WebCore	▶ WebCore::GlyphWidthMap::widthForGlyph(unsigned short)
	2.7%	2.7%	WebCore	▶ WebCore::Font::canUseGlyphCache(WebCore::TextRun const&) const
	1.7%	1.7%	libobjc.A.dylib	▶ objc_msgSend
	1.6%	1.6%	WebCore	▶ WebCore::nextBreakablePosition(unsigned short const*, int, int, bool)
	1.2%	1.2%	WebCore	▶ WebCore::RenderText::width(unsigned int, unsigned int, WebCore::Font const&, int) const
	1.2%	1.2%	WebCore	▶ WebCore::Font::floatWidth(WebCore::TextRun const&) const
	1.2%	1.2%	CoreFoundation	▶ CFStringFindAndReplace
	1.1%	1.1%	CoreFoundation	▶ CFStringGetCharactersPtr
	1.1%	1.1%	libicucore.A.dylib	▶ ubidi_getClass
	0.9%	0.9%	WebCore	▶ WebCore::WidthIterator::WidthIterator(WebCore::Font const*, WebCore::TextRun const&)
	0.9%	0.9%	CoreFoundation	▶ CFStringGetCStringPtr
	0.9%	0.9%	mach_kernel	▶ ml_set_interrupts_enabled
	0.8%	0.8%	libicucore.A.dylib	▶ u_charDirection
	0.8%	0.8%	CoreFoundation	▶ __CFStringChangeSizeMultiple
	0.8%	0.8%	Primes	▶ -[PrimesViewController isPrime:]
	0.8%	0.8%	libSystem.B.dylib	▶ __memcpy
	0.7%	0.7%	libSystem.B.dylib	▶ ceilf
	0.7%	0.7%	libSystem.B.dylib	▶ malloc_zone_free
	0.7%	0.7%	CoreFoundation	▶ CFStringGetLength
	0.6%	0.6%	WebCore	▶ WebCore::Font::isFixedPitch() const
	0.6%	0.6%	WebCore	▶ 0x2c24b70 [800.4KB]
	0.5%	0.5%	CoreFoundation	▶ CFStringGetCharacters
	0.4%	0.4%	WebCore	▶ WebCore::Font::floatWidthForSimpleText(WebCore::TextRun const
	0.4%	0.4%	mach_kernel	▶ lck_mtx_lock
	0.4%	0.4%	libSystem.B.dylib	▶ __spin_lock
	0.4%	0.4%	WebCore	▶ WebCore::RenderObject::baselinePosition(bool, bool) const

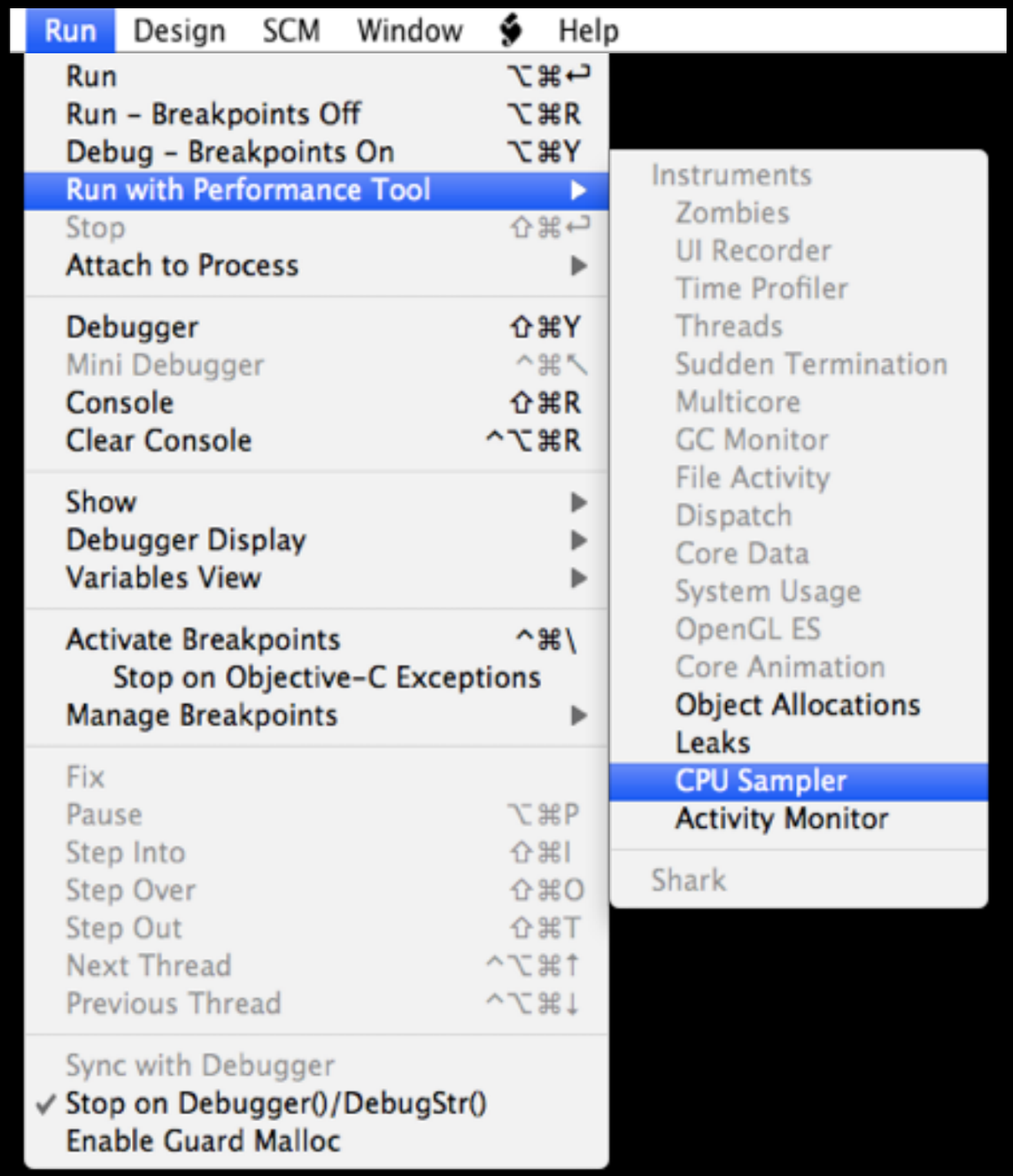
Confirms the top-down view's information (.8%)

18 of 2262 (0.8%) samples selected

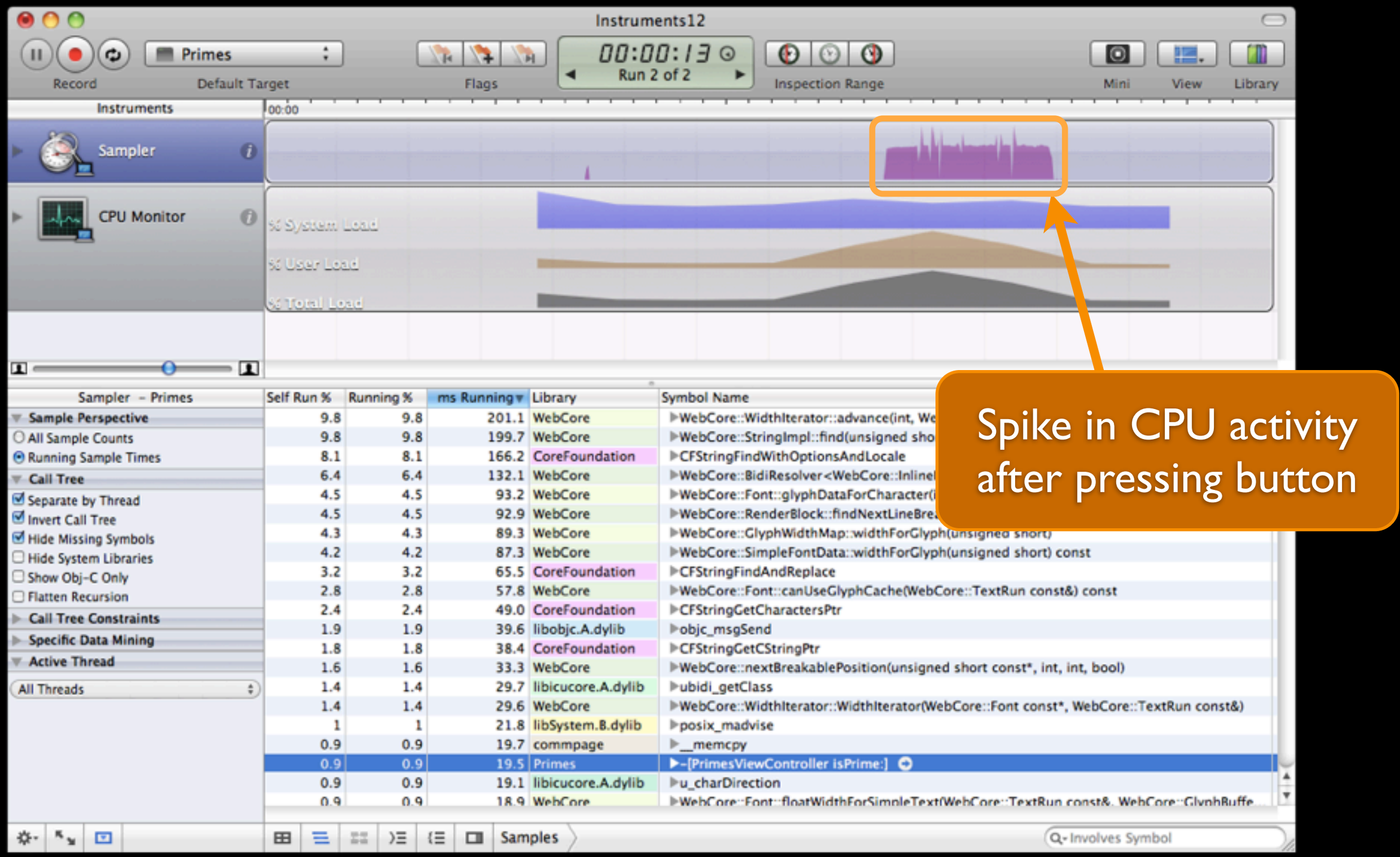
Process: (100.0%) Primes [12869] Thread: All View: Heavy (Bottom-Up)

Sampling with Instruments

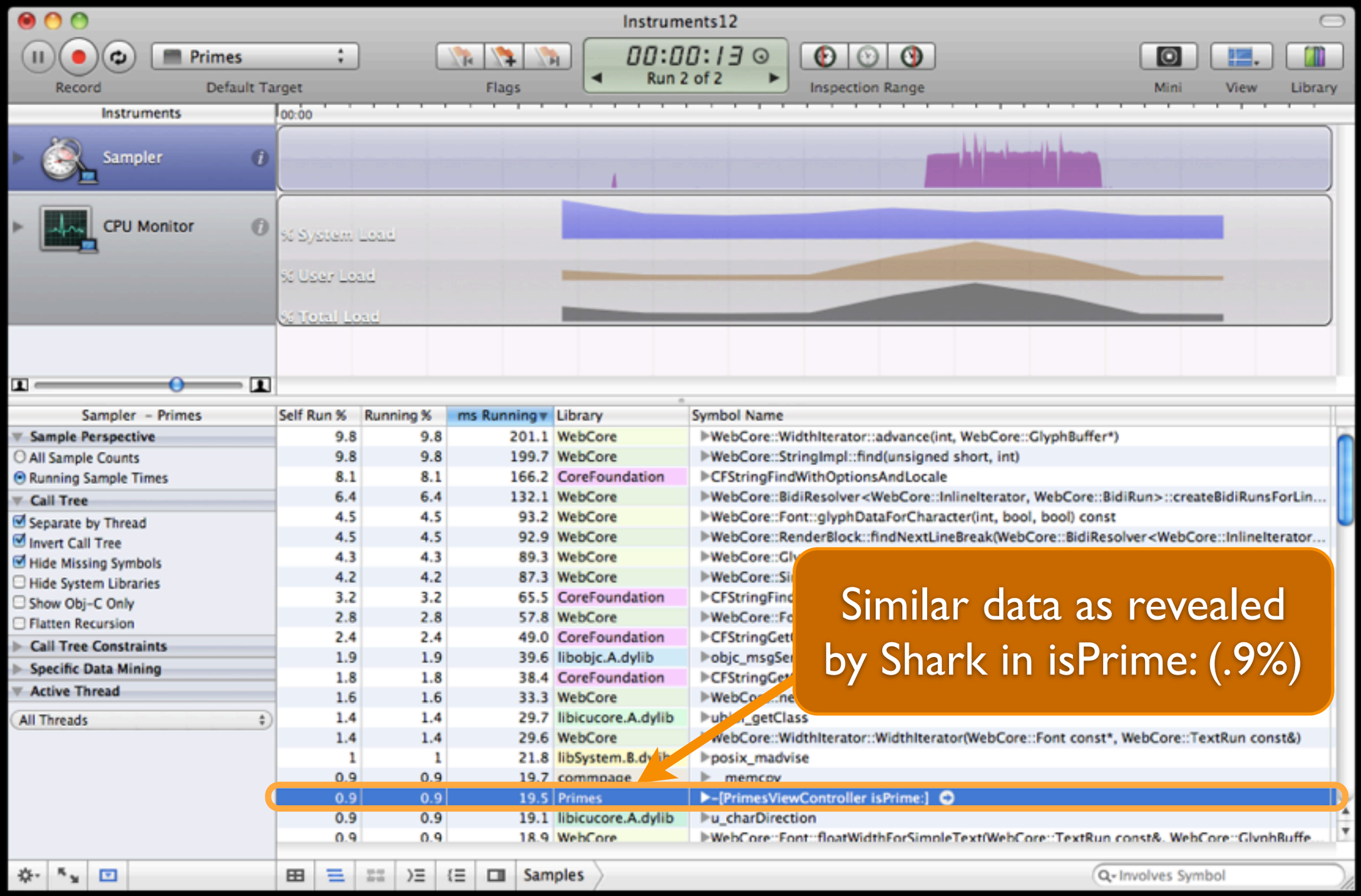
- Instruments is a bit more integrated with Xcode than Shark is
- Simply select CPU Sampler under the Run with Performance Tool item from the Run menu
- Unlike Shark, Instruments will automatically (re)start the application



Instruments



Instruments

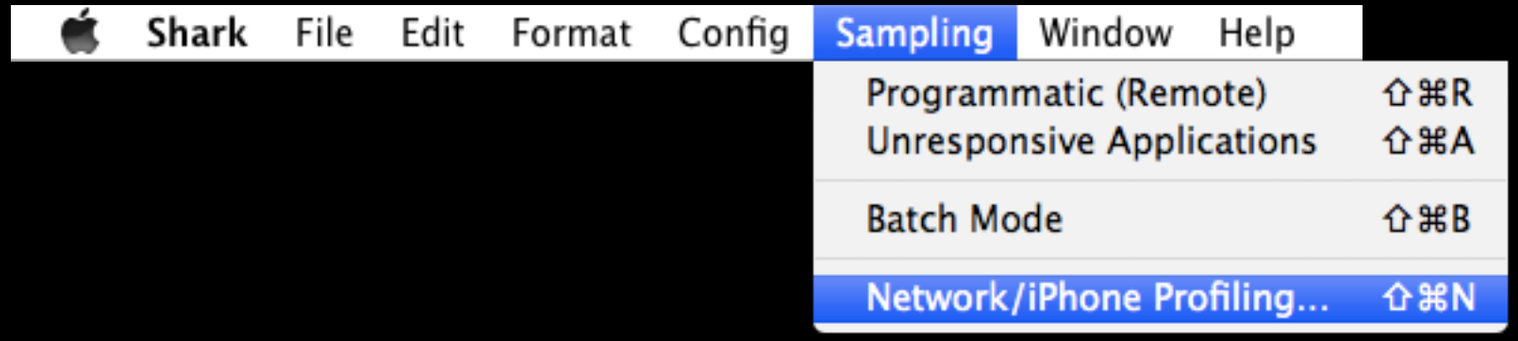


On Device Performance

- But wait! Didn't way say that on-device performance should be used for gather performance data as well
  - Yes!
  - Both Shark and Instruments support capturing their data off of an actual iPhone OS device in addition to the simulator

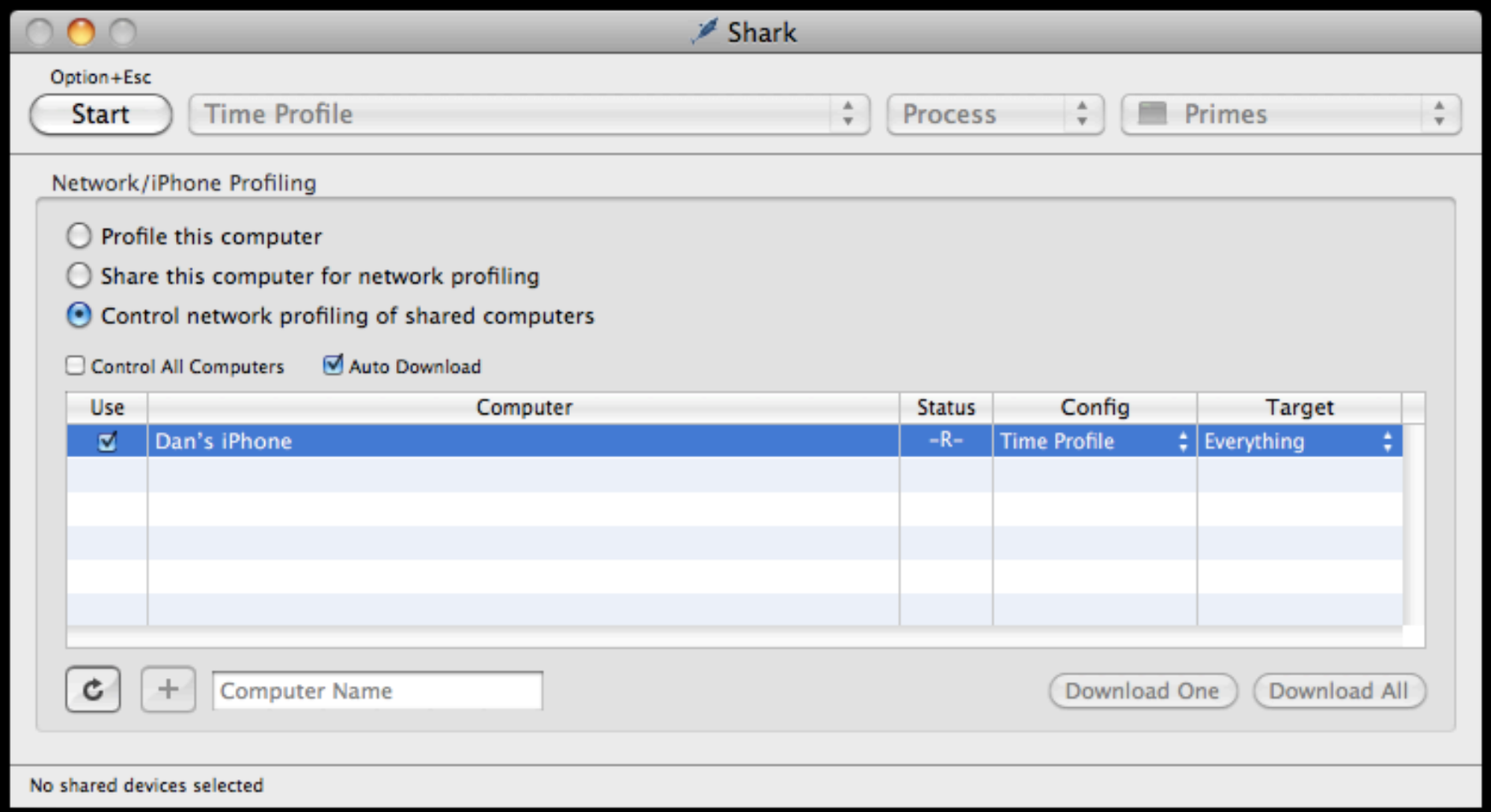
Shark On-Device Profiling

- To profile an app on-device, simply select the Network/ iPhone Profiling option from Shark's Sampling menu item...

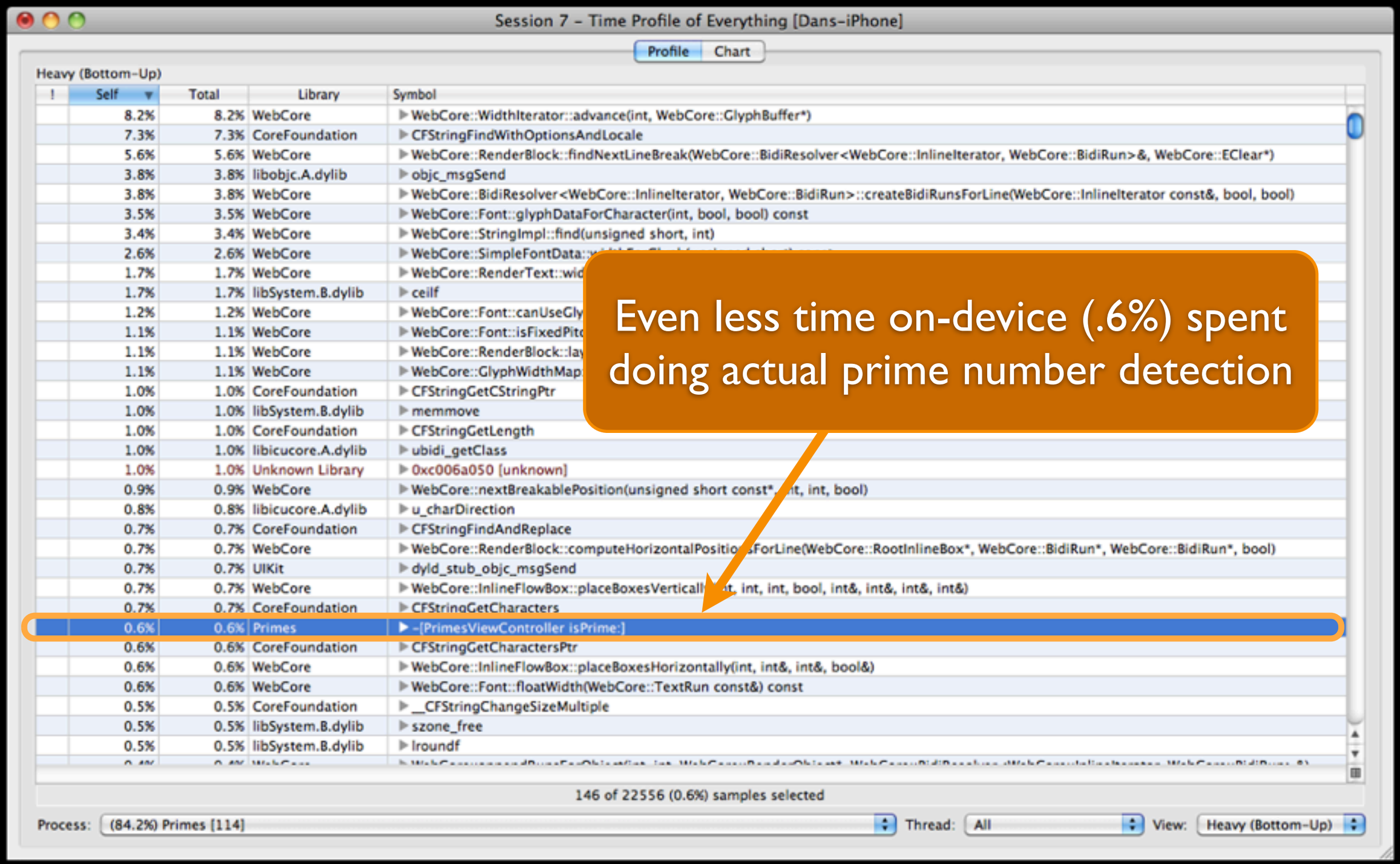


Shark On-Device Profiling

- Then select the device you wish to sample, and press Start...



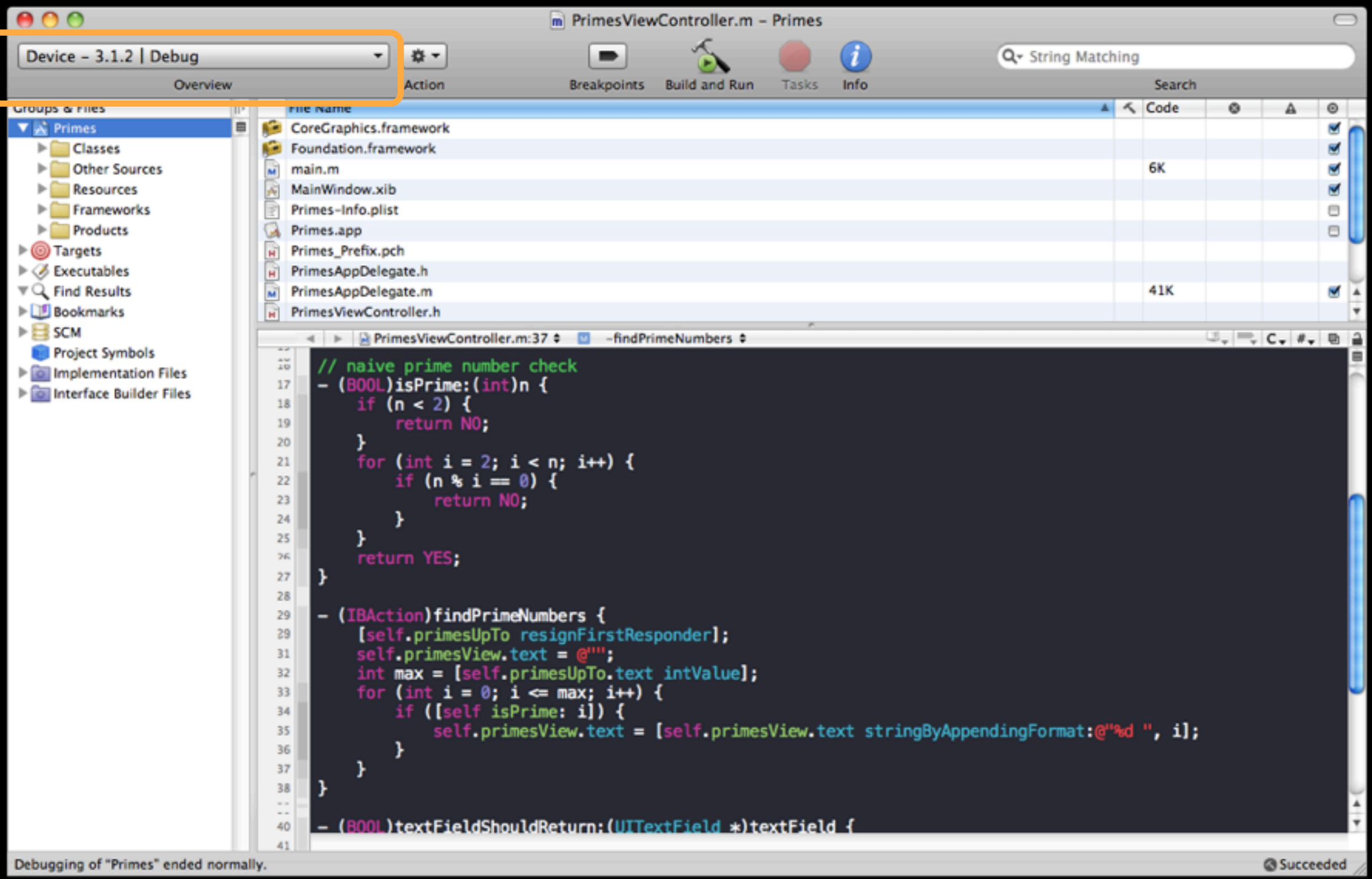
Shark On-Device Profiling Results



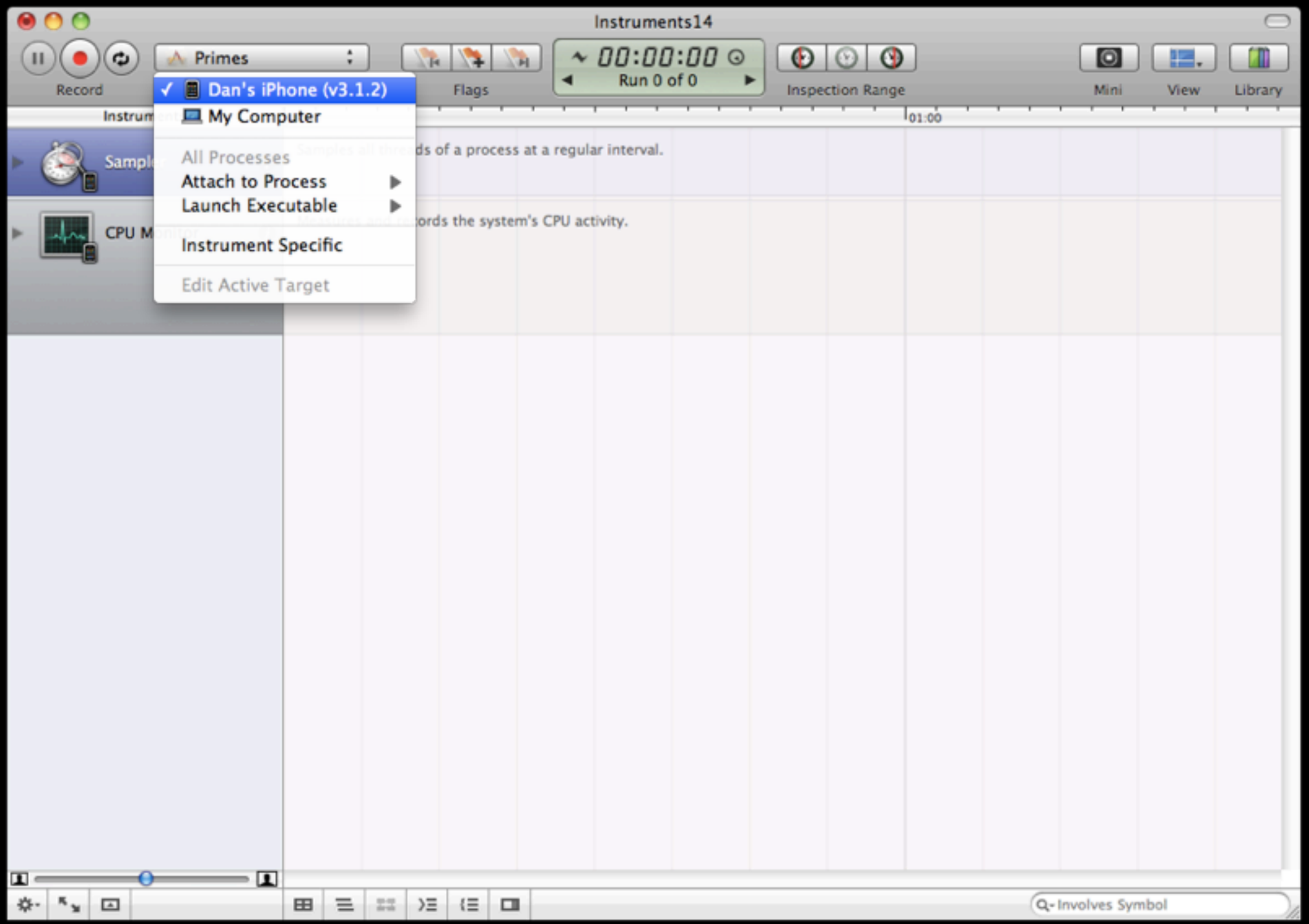
Instruments On-Device Profiling

- The easiest way to do on-device sampling from Instruments is to select the Device setting under Overview, and select the Leaks option from the Run with Performance Tool option
- From Instruments...
  - Be sure that the Device (as opposed to the simulator) is selected
  - Set the launch executable to the app you wish to debug
- Once you do this once, you should be able to simply use the pull downs from within Xcode

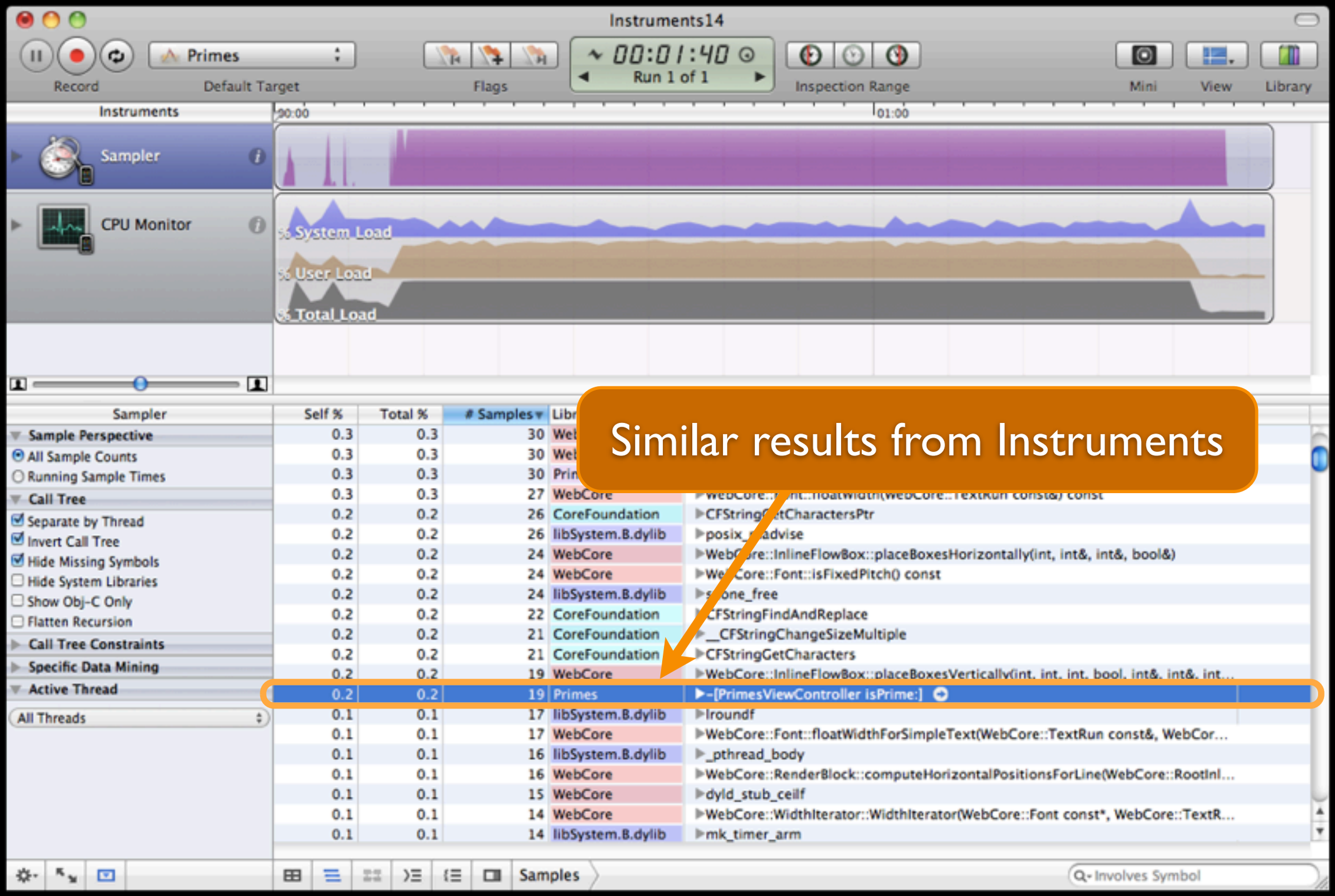
Selecting On-Device Development in Xcode



Selecting Device



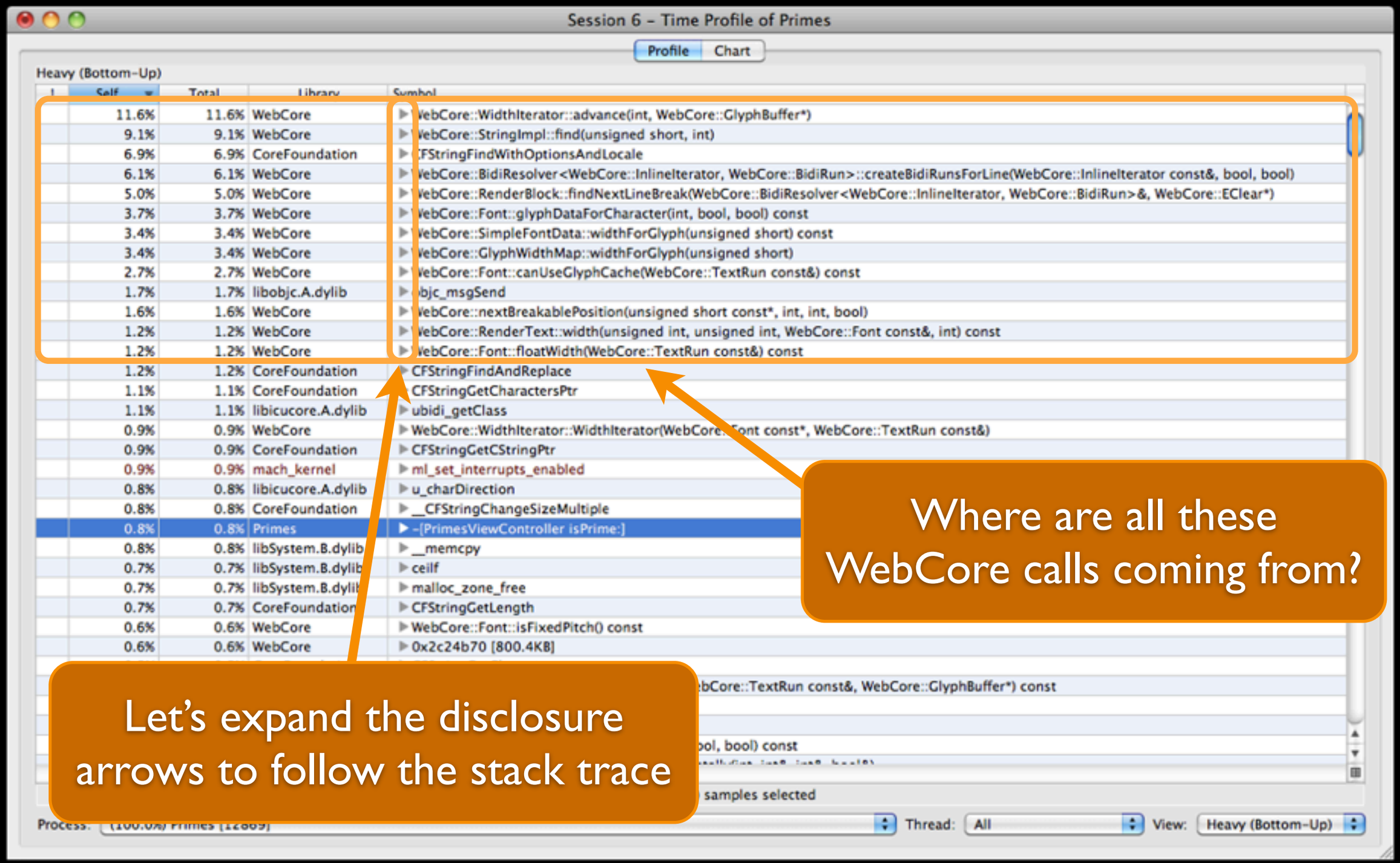
Instruments On-Device Profiling Results



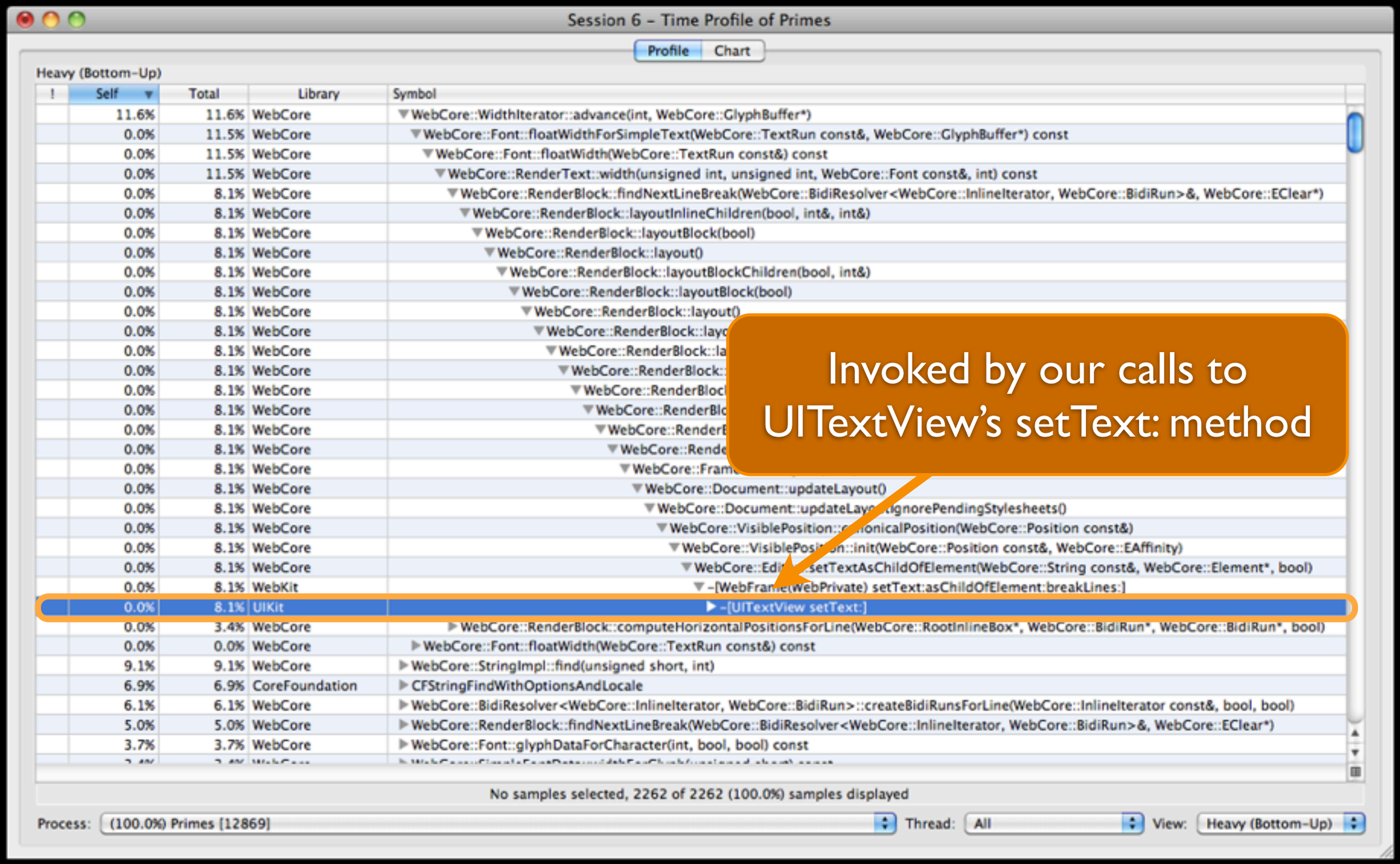
Observations

- So, we see that less than 1% of time is spent in our naive isPrime: implementation
  - Good thing we didn't spend time prematurely optimizing!
- Most of the top 10 CPU hogs are from the WebCore library
  - These top 10 account for about a significant chunk of time
  - Gains in this area will have substantial returns
- What's calling these WebCore methods?
  - Let's look at the stack traces...

Tracing the Source



Tracing the Source



Re-Design & Re-Code

- Seems that UITextView's setText: method is causing all of these WebCore methods to get called
  - This is chewing up a lot of CPU cycles
  - Based on method names, this is presumably layout related
- Can we do anything to address this?

Re-Design & Re-Code

- Yes!
  - We can store the string in a buffer, then when we have all the results, we can do a single setText:

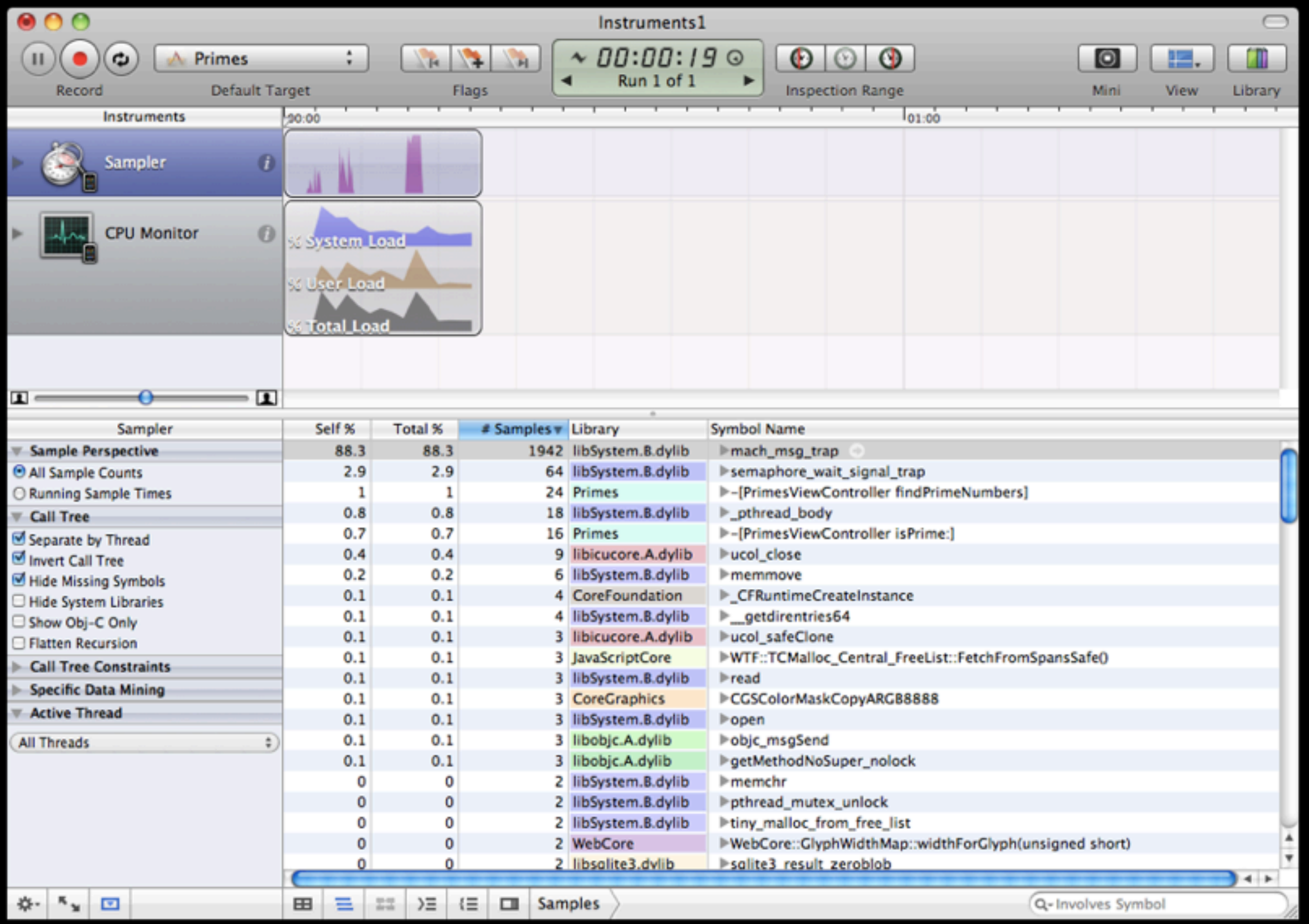
New & Improved PrimesViewController.m

```
#import "PrimesViewController.h"

@implementation PrimesViewController

// ... everything else the same ...

- (IBAction)findPrimeNumbers {
  NSMutableString *buffer = [NSMutableString string];
  [self.primesUpTo resignFirstResponder];
  self.primesView.text = @"";
  int max = [self.primesUpTo.text intValue];
  for (int i = 0; i <= max; i++) {
    if ([self isPrime: i]) {
      [buffer appendFormat:@"%d ", i];
    }
  }
  self.primesView.text = buffer;
}

// ... everything else the same ...

@end
```

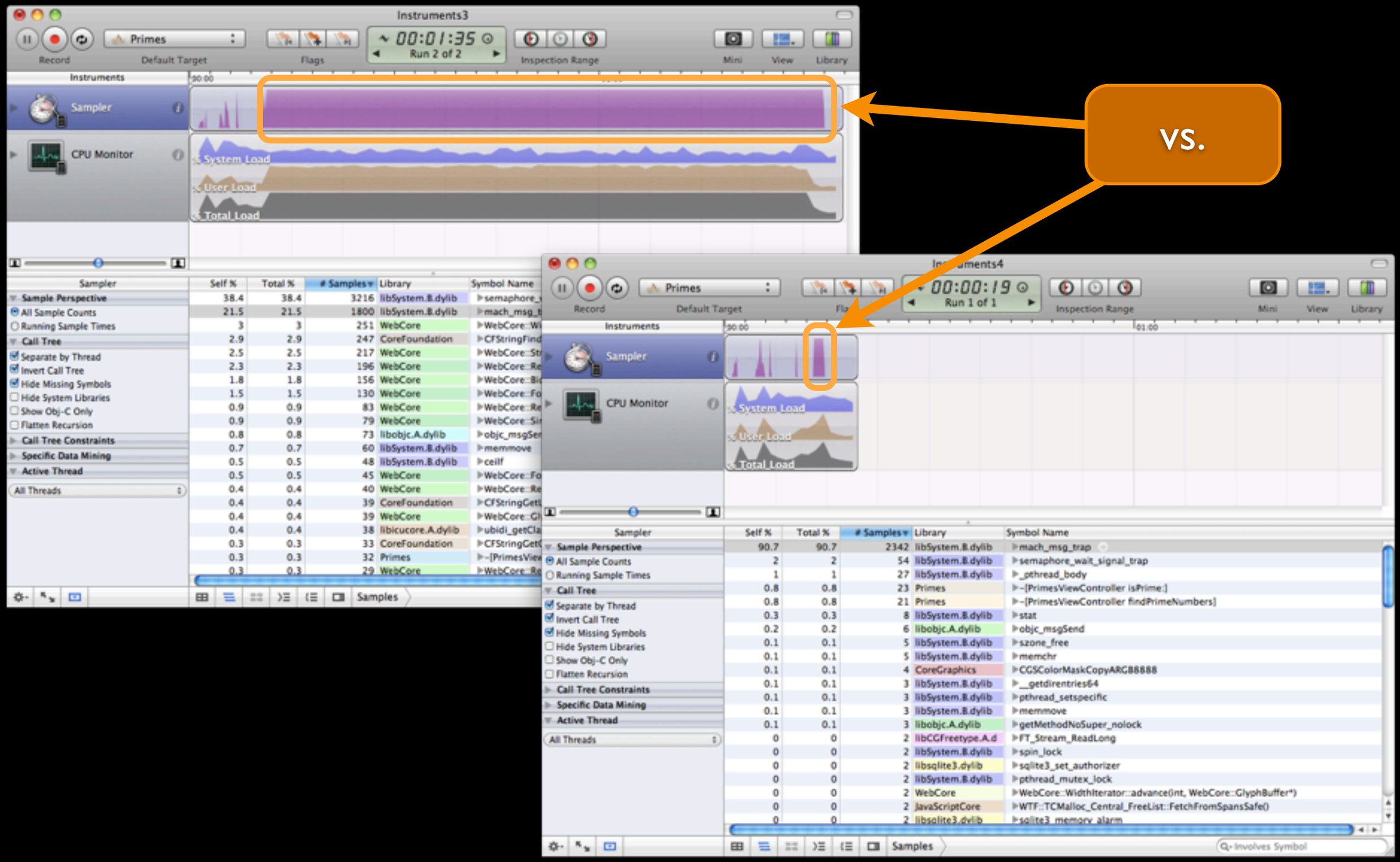
New & Improved Performance in iPhone Simulator

- When running on my unibody MacBook Pro it took...
- Near instantaneous to get through 10,000 numbers
- Less than a second to get through 50,000 numbers

New and Improved Version Running on Device



Before & After



New & Improved Performance in iPhone Simulator

- Running the optimized version on my iPhone 3G, we see substantial gains
- For example, when computing primes up to (including) 10,000...
  - Before: ~80 seconds
  - After: ~2 seconds

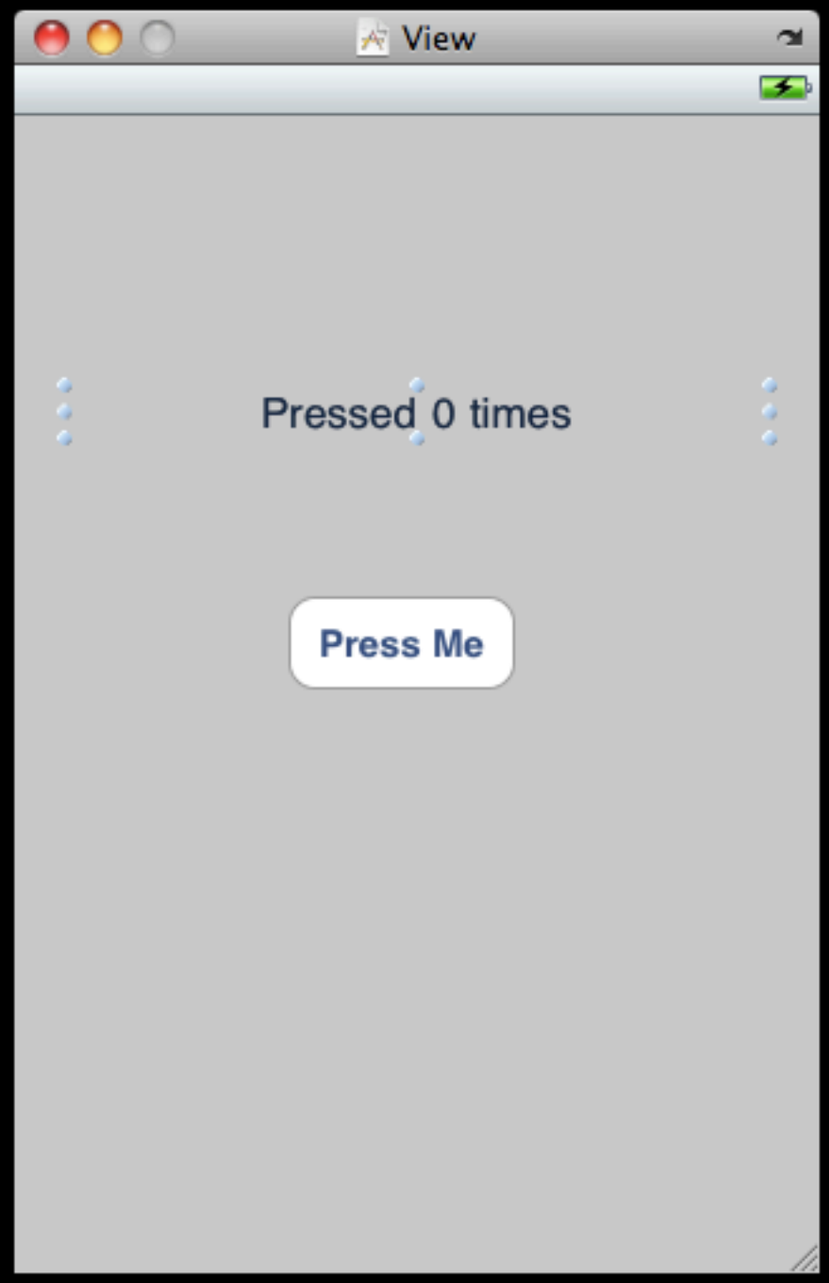
Detecting Memory Leaks

Detecting Memory Leaks

- There's a particular configuration of instruments called "Leaks" is a particular set of instruments that allow you to detect memory leaks in the Instruments tool

LeakViewController.xib

- To demonstrate Leak's capabilities, we'll create a simple UI that leaks some memory each time a button's pressed
- The button simply calls an action which allocates a new string to update the label with (but doesn't release it)



LeakViewController.h

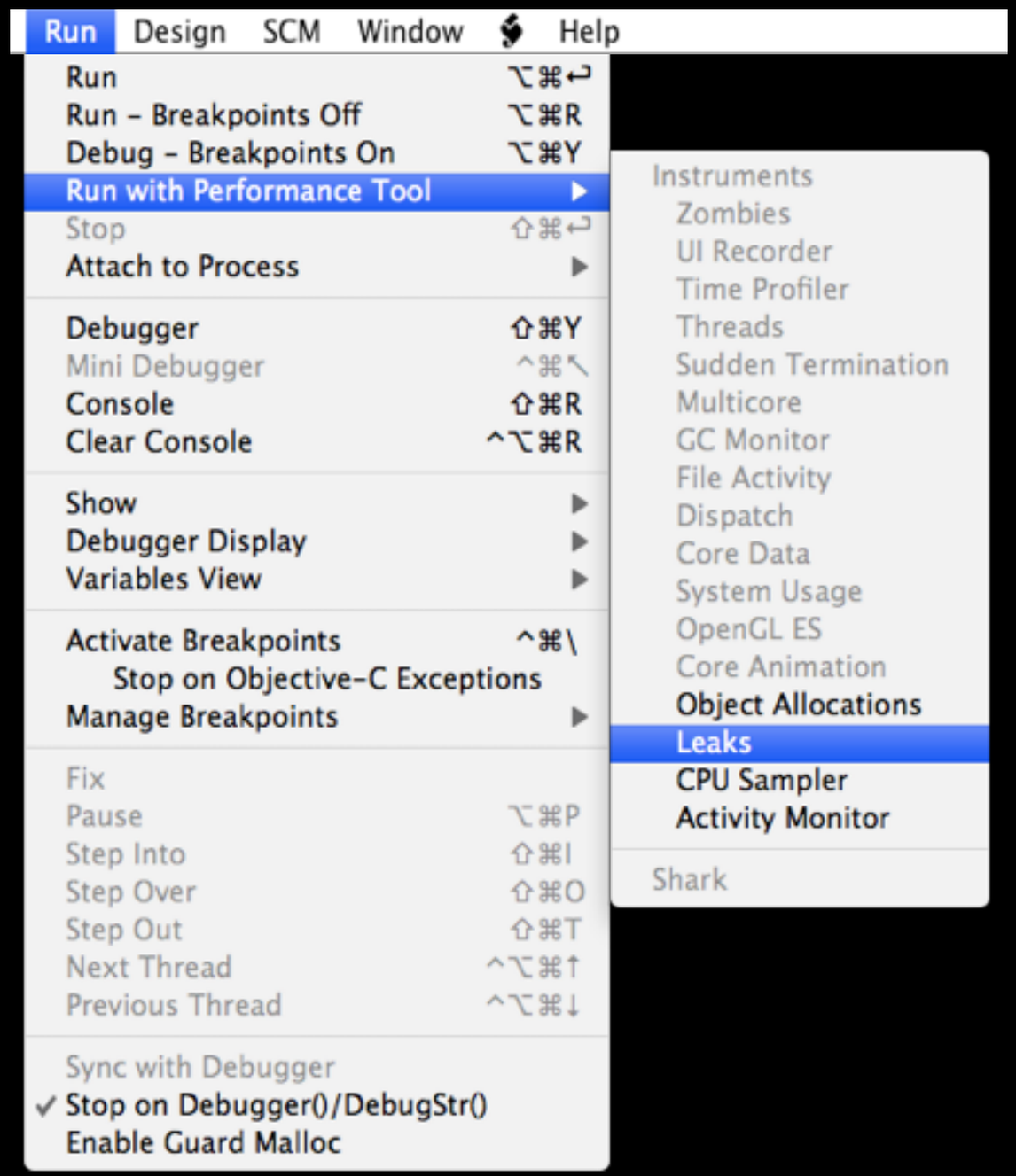
```
#import <UIKit/UIKit.h>

@interface LeakViewController : UIViewController {

  UILabel *label;
  int count;

}
@property(nonatomic, retain) IBOutlet UILabel *label;

- (IBAction)buttonPress;

@end
```

LeakViewController.m

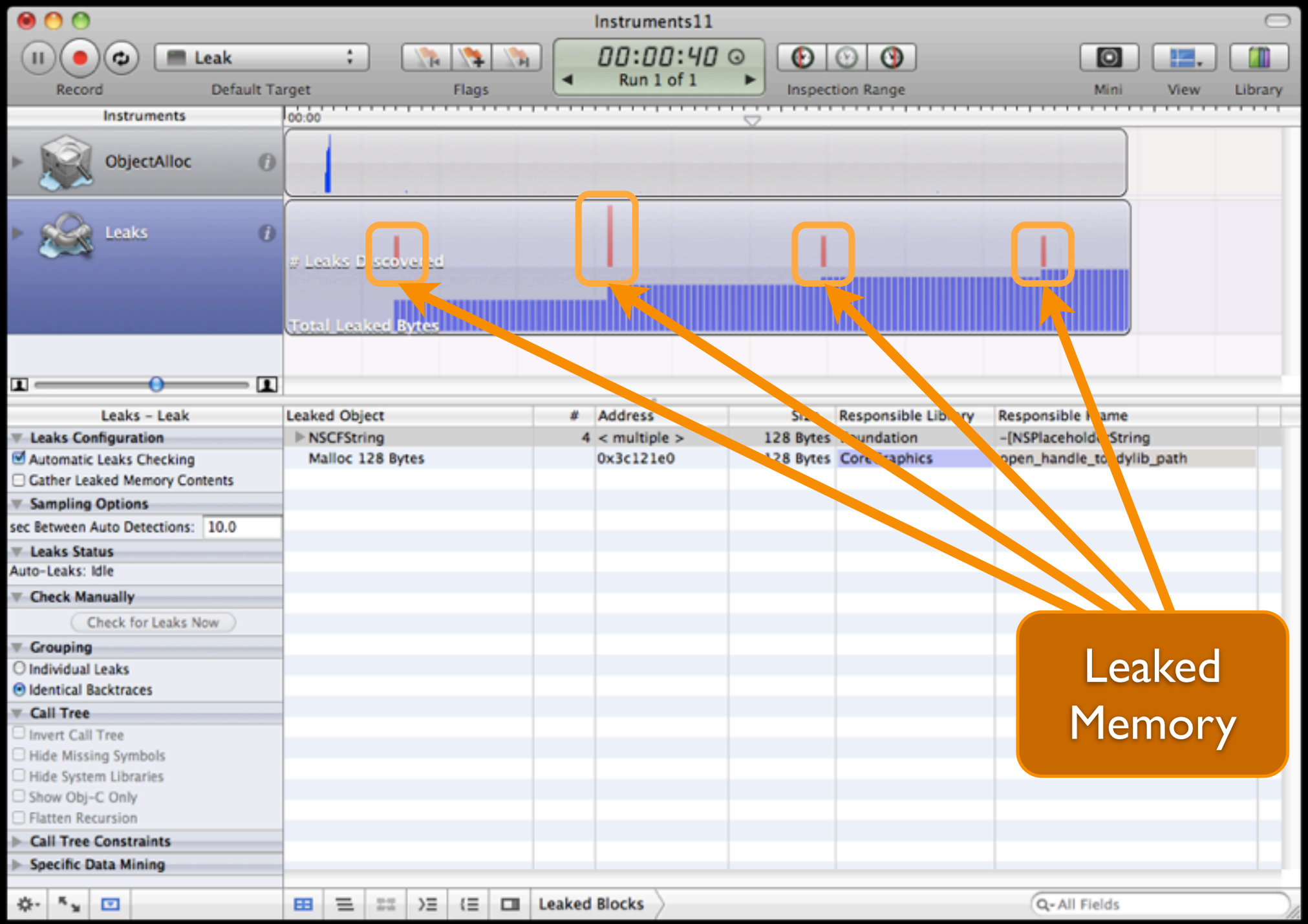
```
#import "LeakViewController.h"

@implementation LeakViewController

@synthesize label;

- (IBAction)buttonPress {
  self.label.text = [[NSString alloc] initWithFormat:@"Pressed %d times", ++count];
}

- (void)dealloc {
  self.label = nil;
  [super dealloc];
}

@end
```

Run with Performance Tools

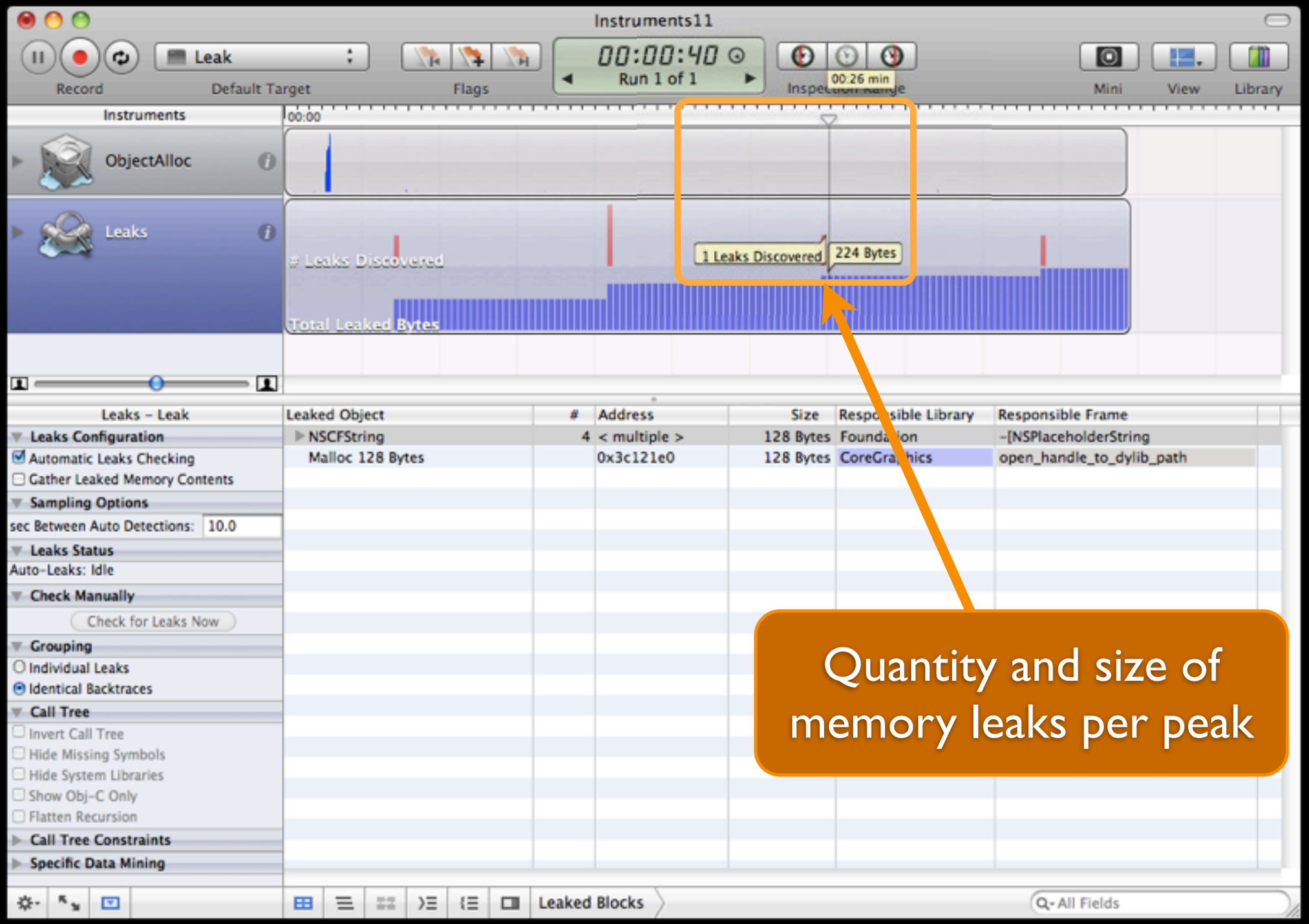
- The Leaks configuration can be opened from the Run menu under Xcode



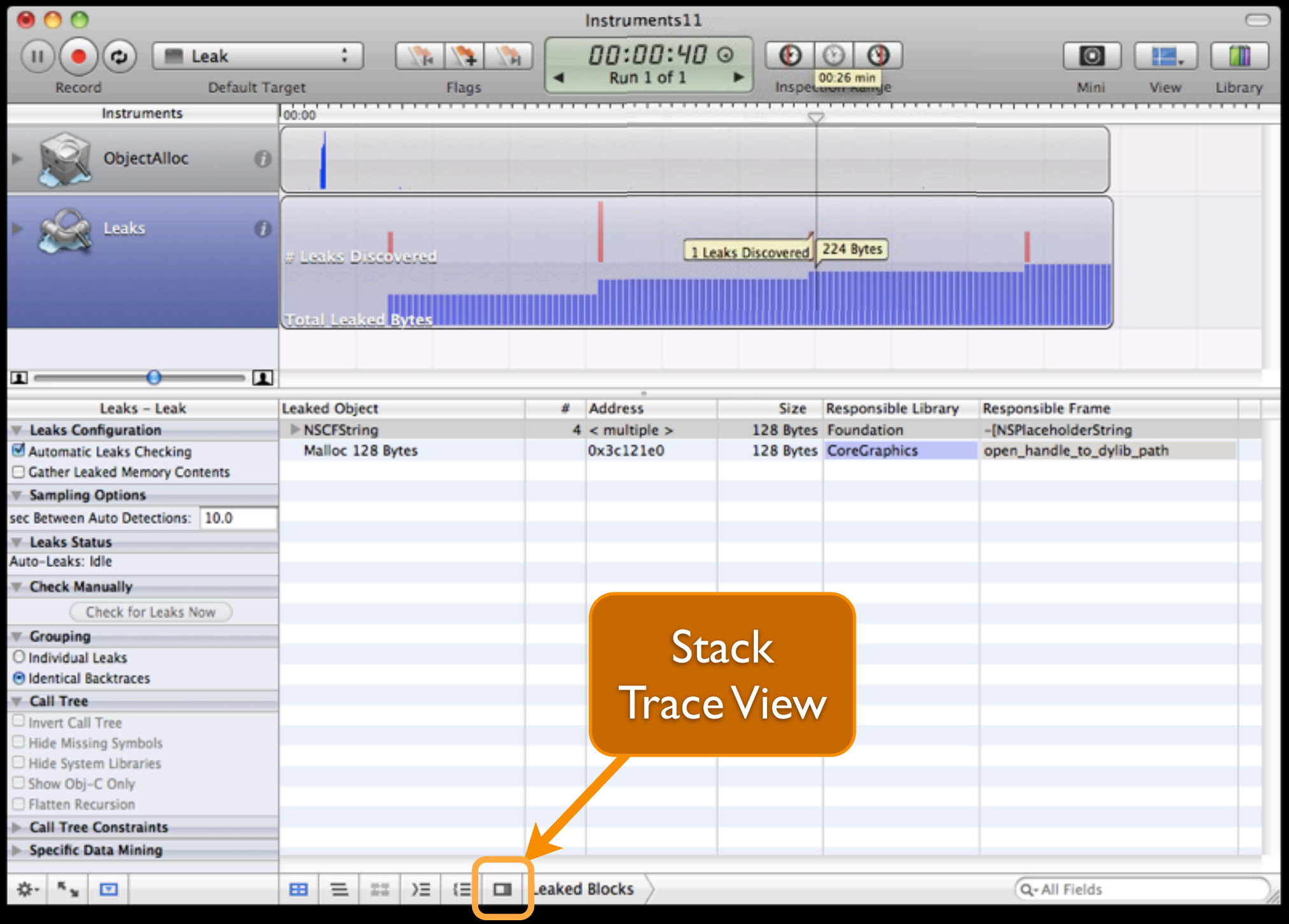
Instruments Output



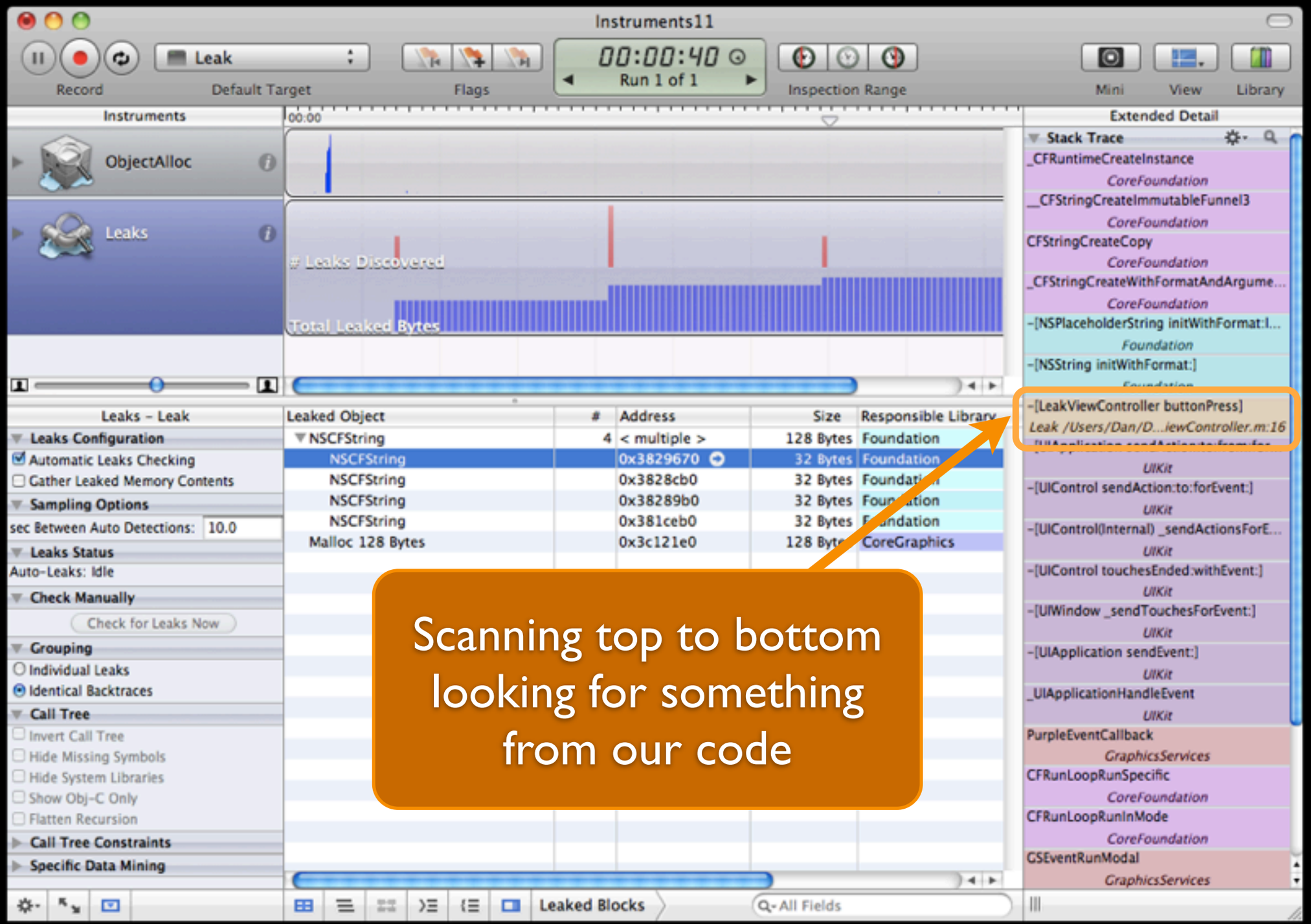
Dragging the Slider



Enabling Stack Traces

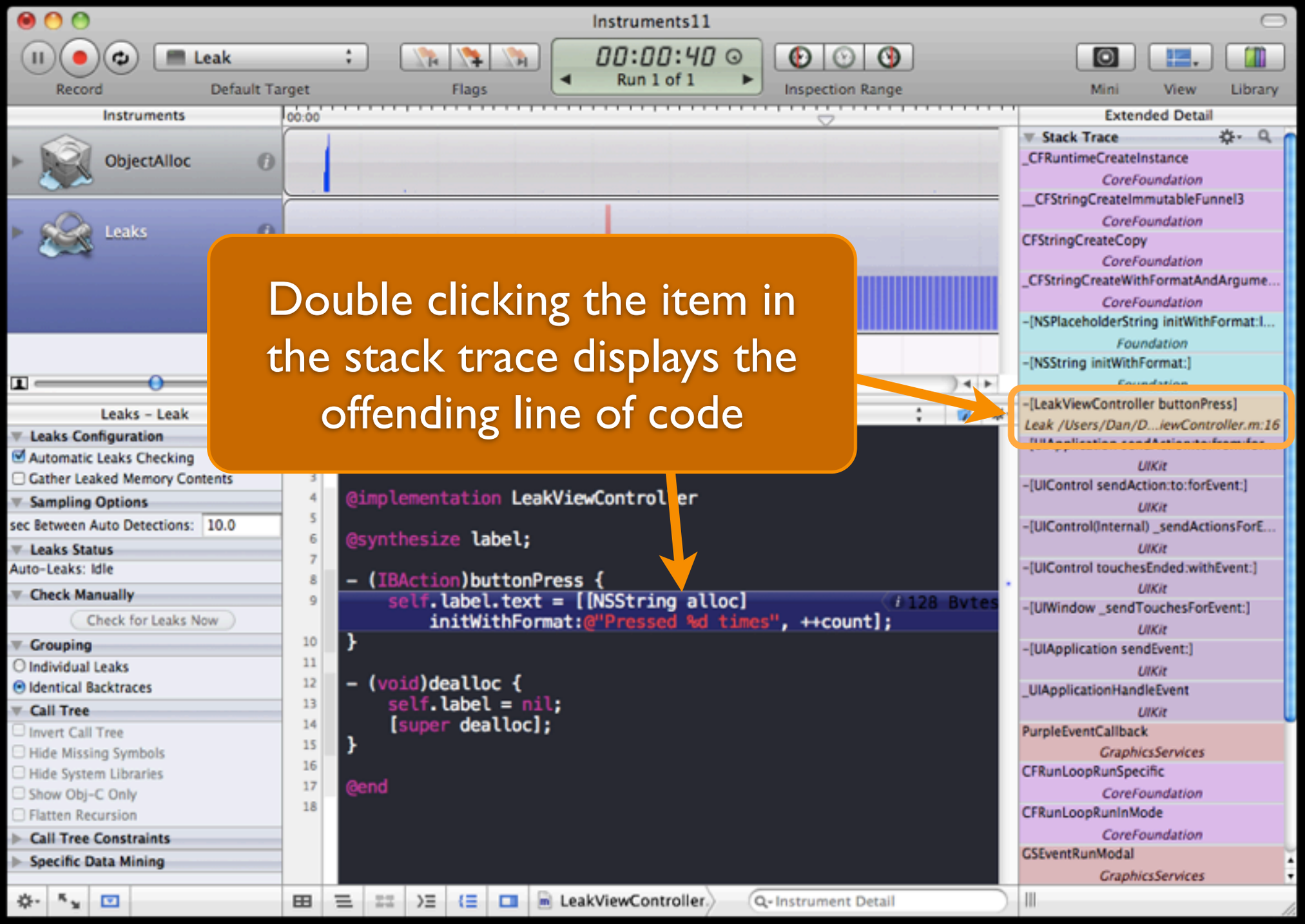


Locating App Code



Scanning top to bottom looking for something from our code

Tracing Back to Code

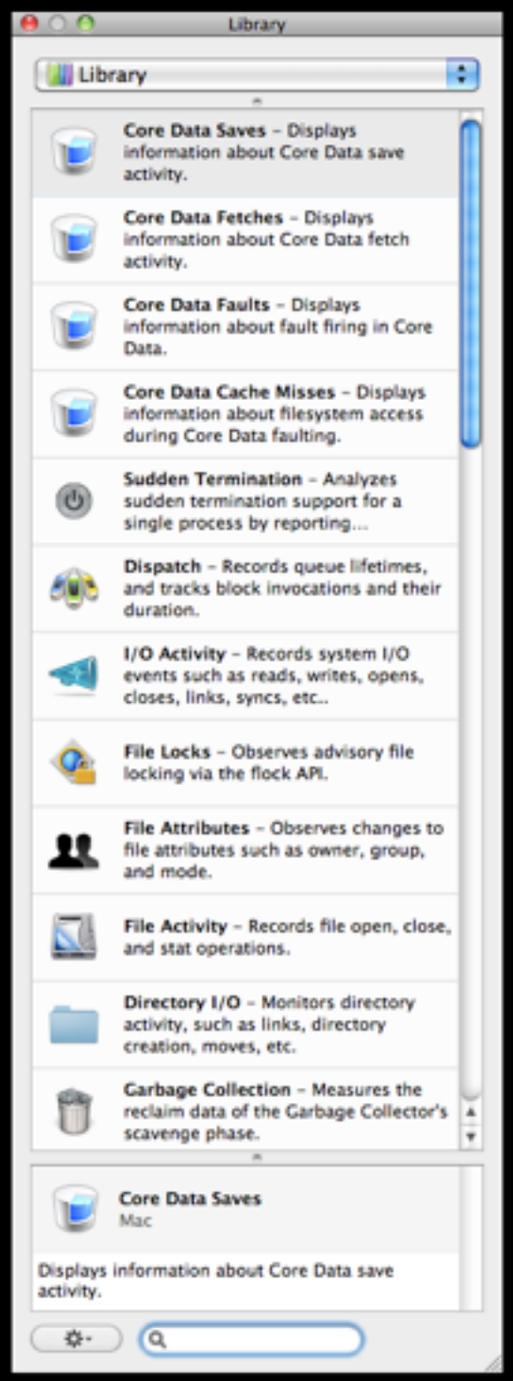


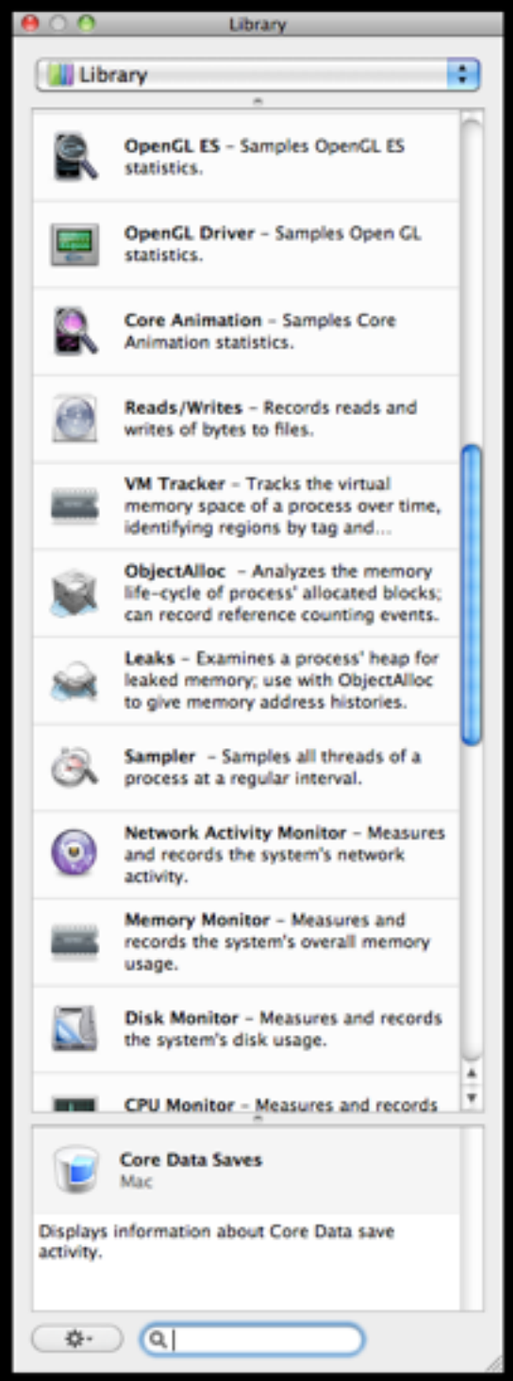
Instruments Odd & Ends

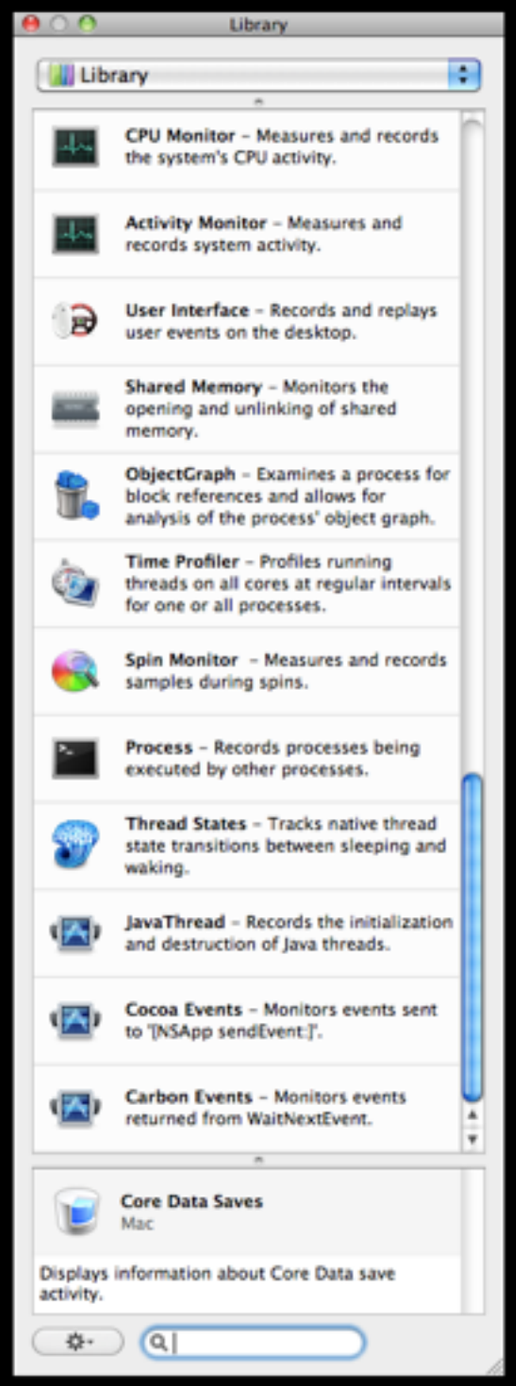
Instruments Odd & Ends

- I've barely scratched the surface on what Instruments can do
- There are many other instruments in the tool that provide insight to all sorts of things
- I encourage you to take a look at the Instruments User Guide to get a feel for some of the other instruments

Tons of Instruments in the Library







Zombies

Zombies

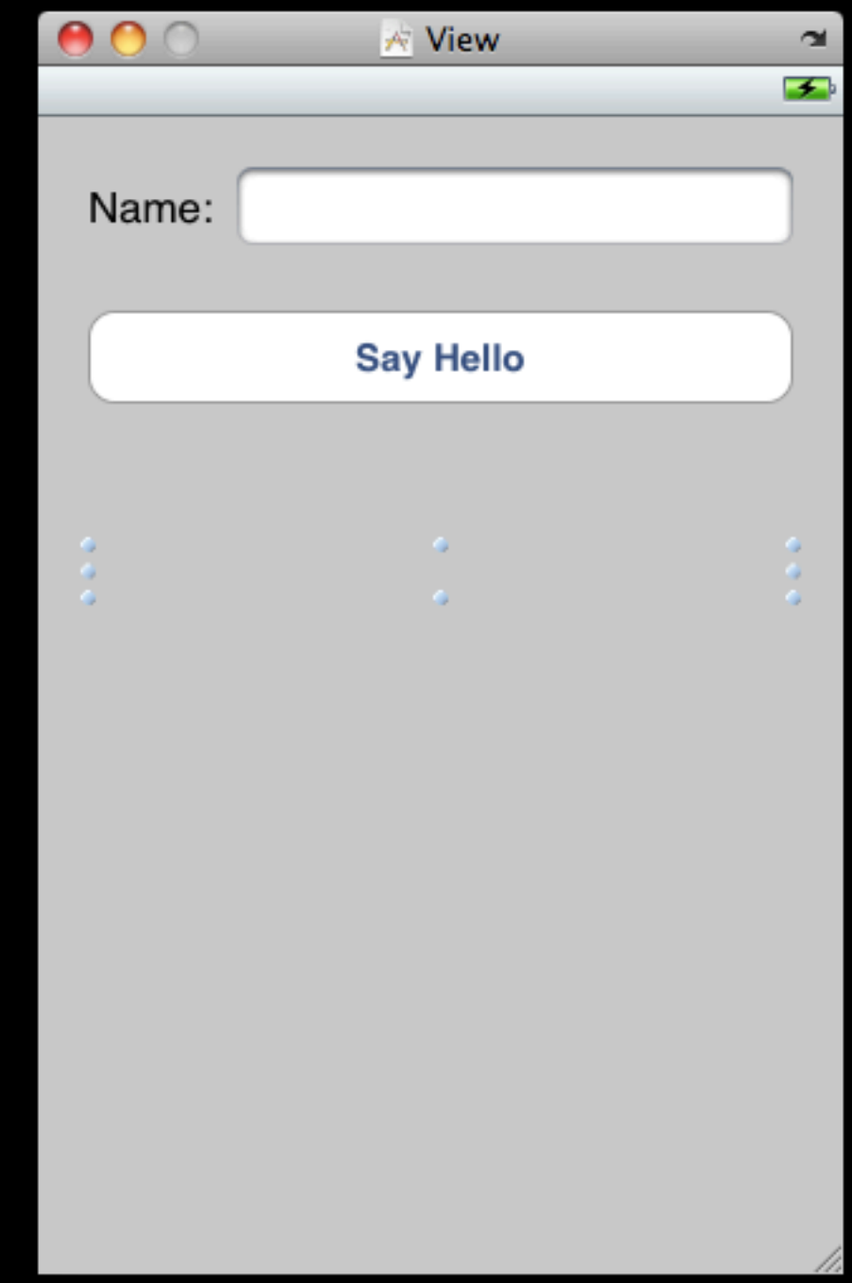
- When zombies are enabled instead of a deallocated object actually being freed, it's class is changed to _NSZombie
- Useful for determining causes of over-releases and more generally messages sent to deallocated objects
- Zombies can be enabled via with NSZombieEnabled environment variable

A Simple Example

- Let's model a common beginner's mistake in this example
- A common mistake when starting out with ObjC memory management might be to release and autoreleased object

Zombie.xib

- Our app will consist of the following...
  - A text field for the user to enter their name
  - A “Say Hello” button that will read the value of the text field
  - A label that will be used to print “Hello X” where X is the value in the text field



ZombieViewController.h

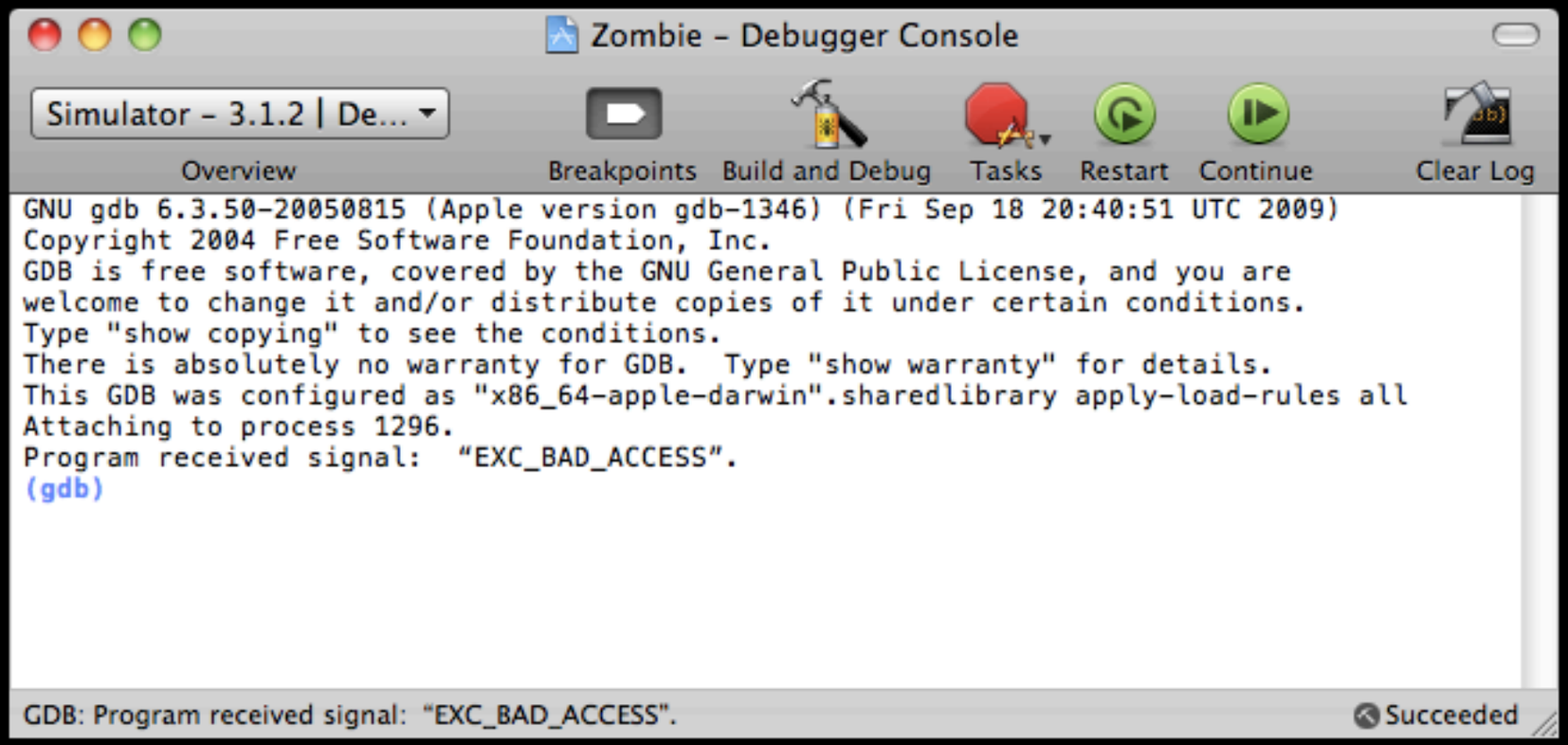
```
#import <UIKit/UIKit.h>

@interface ZombieViewController : UIViewController {

    UITextField *nameField;
    UILabel *greetingLabel;
}

@property(nonatomic, retain) IBOutlet UITextField *nameField;
@property(nonatomic, retain) IBOutlet UILabel *greetingLabel;

- (IBAction)sayHello;

@end
```

ZombieViewController.m

```
#import "ZombieViewController.h"

@implementation ZombieViewController

@synthesize nameField;
@synthesize greetingLabel;

- (IBAction)sayHello {
  NSString *msg = [NSString stringWithFormat:@"Hello %@", self.nameField.text];
  self.greetingLabel.text = msg;

  // beginner mistake - over-release
  [msg release];
}

- (void)dealloc {
  self.nameField = nil;
  self.greetingLabel = nil;
  [super dealloc];
}

@end
```

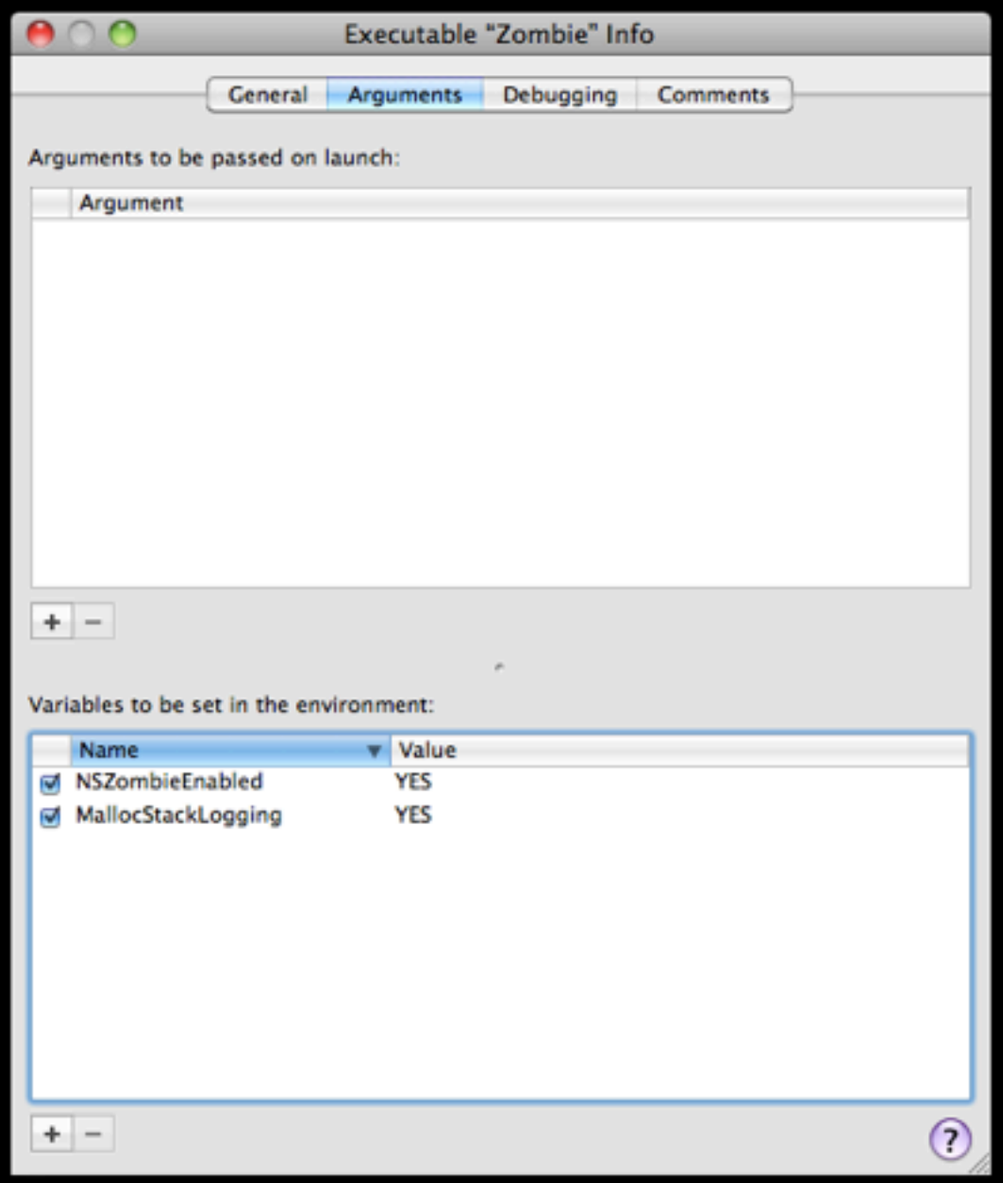
Running the Buggy App

- If we were to run the app normally our app would crash and return to the home screen without any message
- If we ran in debug mode, we'd see the following message in the console...



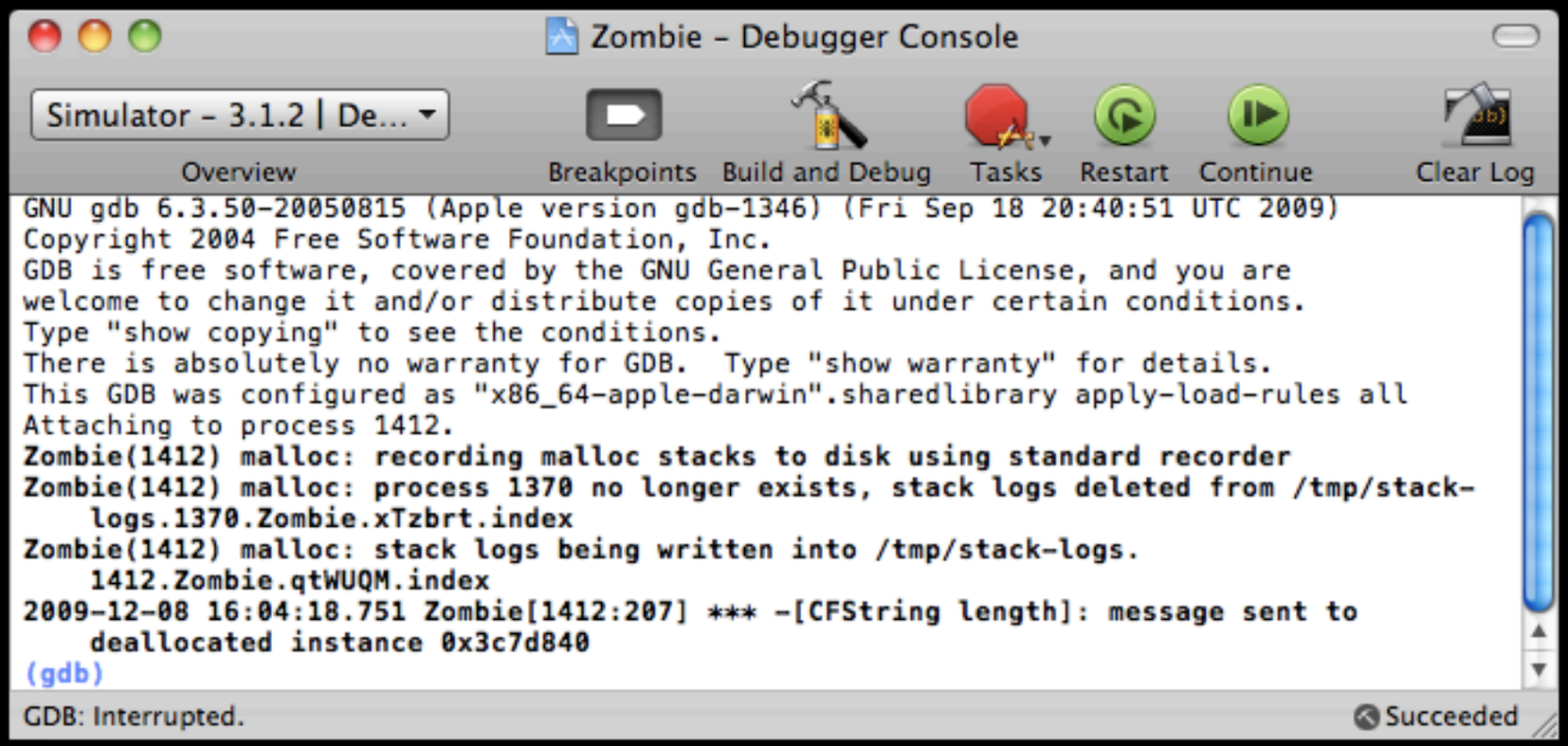
Enabling Zombies

- Expand out your app's executable and bring up the Info window by pressing ⌘I
- Add NSZombieEnabled & MallocStackLogging to the "Variables to be set in the environment" section, both with a value of YES



Zombie Messages

- Now if we run the app, we get the following in the console...



Tracking the Source

- We can further glean insight into the source of the problem using by typing the following command into the console...
  - shell malloc_history <process id> <address>

Zombie – Debugger Console

Simulator – 3.1.2 | D... | Overview | Breakpoints | Build and Debug | Tasks | Restart | Continue | Clear Log

```
2009-12-08 16:04:18.751 Zombie[1412:207] *** -[CFString length]: message sent to
    deallocated instance 0x3c7d840
(gdb) shell malloc_history 1412 0x3c7d840

ALLOC 0x3c7d820-0x3c7d99f [size=384]: thread_a050f500 |start | main | UIApplicationMain
    | GSEventRun | GSEventRunModal | CFRunLoopRunInMode | CFRunLoopRunSpecific |
    PurpleEventCallback | _UIApplicationHandleEvent | -[UIApplication sendEvent:] | -
    [UIApplication handleEvent:withNewEvent:] | -[UIKeyboardLayoutStar
    handleHardwareKeyDownFromSimulator:] | -[UIKeyboardLayoutStar deleteAction] | -
    [UIKeyboardImpl handleDelete] | -[UIKeyboardImpl deleteFromInput] | -[UIKeyboardImpl
    generateCandidates:] | -[UIKeyboardInputManagerZephyr autocorrection] |
    KBInputManager::autocorrection() | KBInputManager::lookup() |
    KBInputManager::fill_result_or_capitalized_word_map(KB::Vector<KB::Word>&,
    KB::Vector<KB::Word> const&, KB::String const&, KB::String const&, KB::String
    const&) | KB::Vector<KB::Word>::push_back(KB::Word const&) |
    KB::Vector<KB::Word>::ensure_capacity(unsigned int) | malloc | malloc_zone_malloc
----
FREE  0x3c7d820-0x3c7d99f [size=384]: thread_a050f500 |start | main | UIApplicationMain
    | GSEventRun | GSEventRunModal | CFRunLoopRunInMode | CFRunLoopRunSpecific |
    PurpleEventCallback | _UIApplicationHandleEvent | -[UIApplication sendEvent:] | -
    [UIApplication handleEvent:withNewEvent:] | -[UIKeyboardLayoutStar
    handleHardwareKeyDownFromSimulator:] | -[UIKeyboardLayoutStar deleteAction] | -
    [UIKeyboardImpl handleDelete] | -[UIKeyboardImpl deleteFromInput] | -[UIKeyboardImpl
    generateCandidates:] | -[UIKeyboardInputManagerZephyr autocorrection] |
    KBInputManager::autocorrection() | KB::Vector<KB::Word>::~Vector() | free

ALLOC 0x3c7d840-0x3c7d947 [size=264]: thread_a050f500 |start | main | UIApplicationMain
    | GSEventRun | GSEventRunModal | CFRunLoopRunInMode | CFRunLoopRunSpecific |
    PurpleEventCallback | _UIApplicationHandleEvent | -[UIApplication sendEvent:] | -
    [UIApplication handleEvent:withNewEvent:] | -[UIKeyboardLayoutStar
    handleHardwareKeyDownFromSimulator:] | -[UIKeyboardLayoutStar deleteAction] | -
    [UIKeyboardImpl handleDelete] | -[UIKeyboardImpl deleteFromInput] | -[UIKeyboardImpl
    updateCandidateDisplay] | -[UIKeyboardImpl removeAutocorrectPrompt] | objc_msgSend |
    _class_lookup                                    hod | _cache_fill | _cache_malloc |
    _calloc_inte
----
FREE  0x3c7d840-                                 0f500 |start | main | UIApplicationMain
    | GSEventRun                                 nInMode | CFRunLoopRunSpecific |
    __CFRunLoopD                                 server_callback(__CFRunLoopObserver*,
    unsigned lon                                 mit() | CA::Context::commit_transaction
    (CA::Transac                                 -[CALayer layoutSublayers] |
    objc_msgSend                                 ne | lookUpMethod | _cache_fill |
    _cache_collect                               ree

ALLOC 0x3c7d840-0x3c7d85f [size=32]: thread_a050f500 |start | main | UIApplicationMain |
    GSEventRun | GSEventRunModal | CFRunLoopRunInMode | CFRunLoopRunSpecific |
    PurpleEventCallback | _UIApplicationHandleEvent | -[UIApplication sendEvent:] | -
    [UIWindow _sendTouchesForEvent:] | -[UIControl touchesEnded:withEvent:] | -
    [UIControl(Internal) _sendActionForEvents:withEvent:] | -[UIControl
    sendAction:to:forEvent:] | -[UIApplication sendAction:to:from:forEvent:] | -
    [ZombieViewController sayHello] | +[NSString stringWithFormat:] | -
    _CFStringCreateWithFormatAndArgumentsAux | CFStringCreateCopy |
    __CFStringCreateImmutableFunnel3 | _CFRuntimeCreateInstance | malloc_zone_malloc
(gdb)
```

Where it was created

GDB: Interrupted. Succeeded

Additional Resources

- Shark User Guide
  - http://developer.apple.com/iphone/library/documentation/DeveloperTools/Conceptual/SharkUserGuide/
- Instruments User Guide
  - http://developer.apple.com/iphone/library/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/