App Performance

iOS App Development
Fall 2010 — Lecture 27

Announcements

- Posted some notes on using Subversion (SVN) with Xcode last week
- Each of the 3 project deliverables were posted to the course blog this past weekend (more details on this in a moment...)
- Determination of final presentation & demo ordering before we get started with tonight's material

Final Project Deliverables

- Monday, December 13th
  - Project source code must be submitted by 11:59 pm
- Tuesday, December 14th
  - Special office hours (5:30 – 6:30 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 15th
  - Presentation slides must be submitted by 11:59 pm
- Thursday, December 16th
  - 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 or data sources 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!) is probably not going to end 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

iOS Profiling Tools

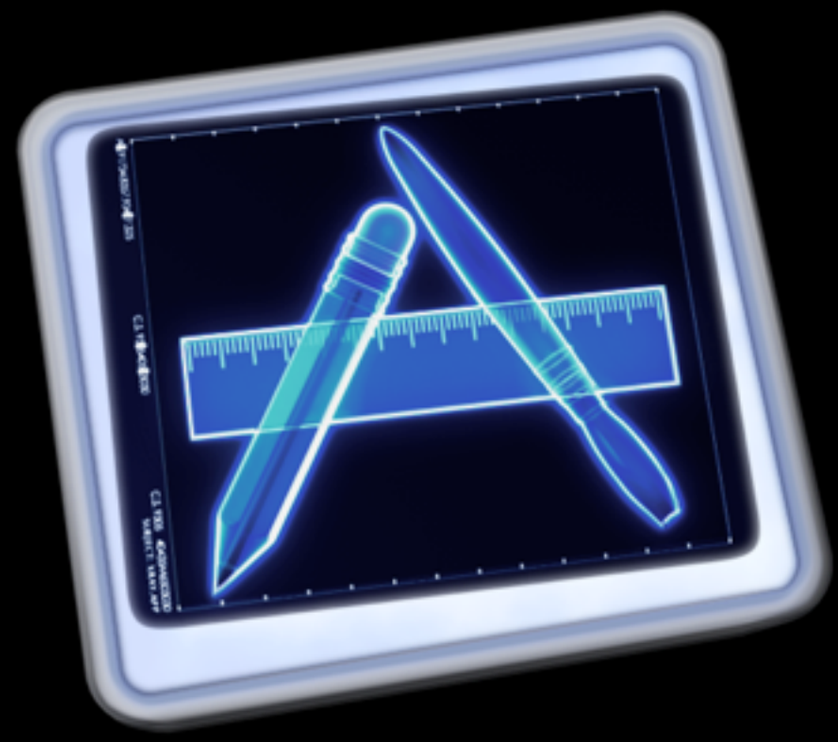
- Apple has provided several profiling tools as part of the iOS SDK...
  - Shark
  - Instruments

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 instruments

Performance Example

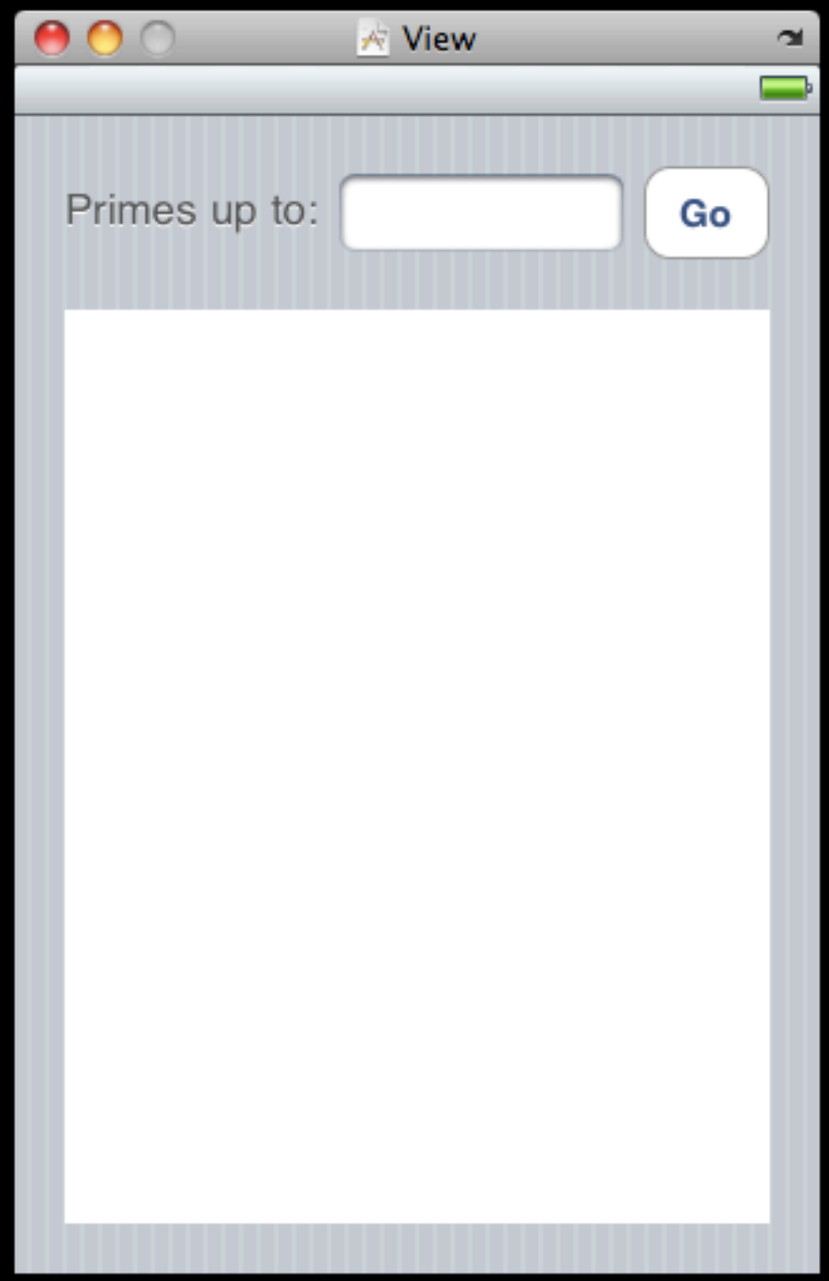
Adapted from Apache Licensed PathologicalPrimeCounter code from iPhone SDK Development

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> {

}

@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;
}

/* ... */

@end
```

Guesses on Performance

- Any guesses on how this is going to do?

Performance in iPhone Simulator

- When running on my (late 2008) MacBook Pro it took...
- About 2.5 seconds to get through 10,000 numbers
- About 70 seconds to get through 50,000 numbers

Performance on Device

- To get through 10,000 numbers...
  - iPhone 4 (iOS 4.2.1) — 20 seconds
  - iPhone 3G (iOS 3.1.2) — 65 seconds
- That's nearly 8–26 times slower depending upon iOS version and hardware revision!
- The moral of this story...
  - Test on an actual device to accurately gauge performance!

≠

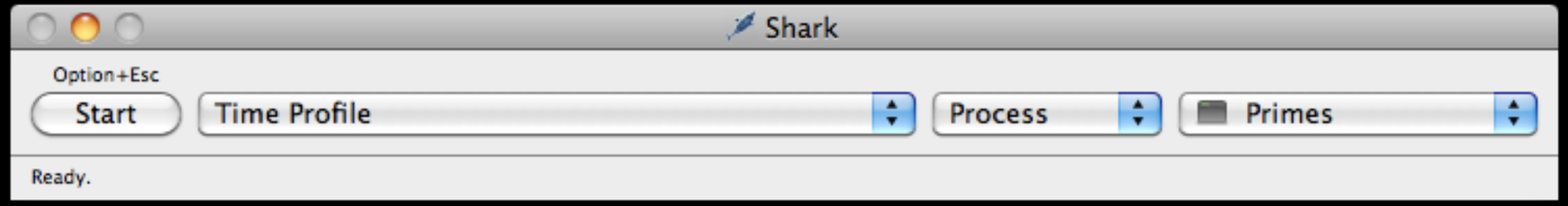
≠

Sampling with Shark

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

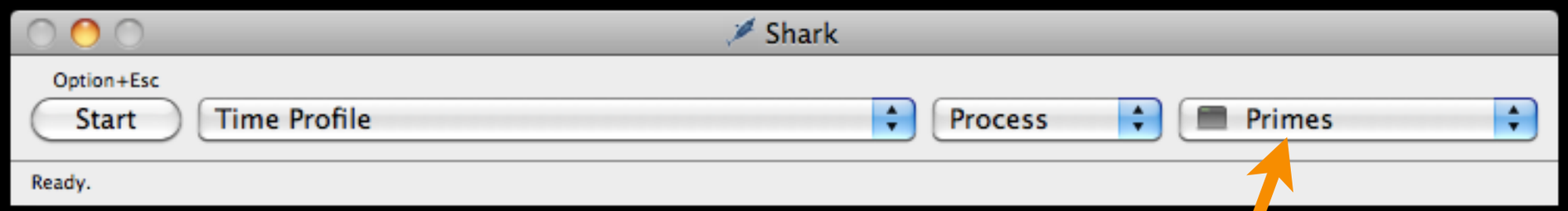
Shark UI

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



Starting Shark

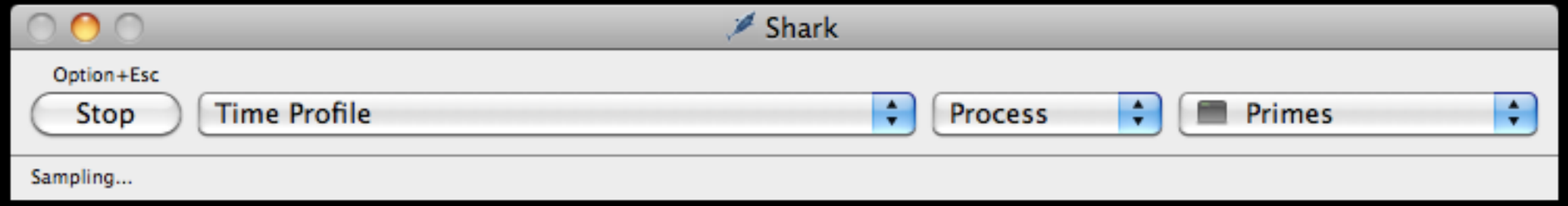
- In order to profile your app, it must first be running, so go ahead and start up your app in the iPhone Simulator
- Then, select the process from the right-most drop down and when you're ready to start sampling press Start



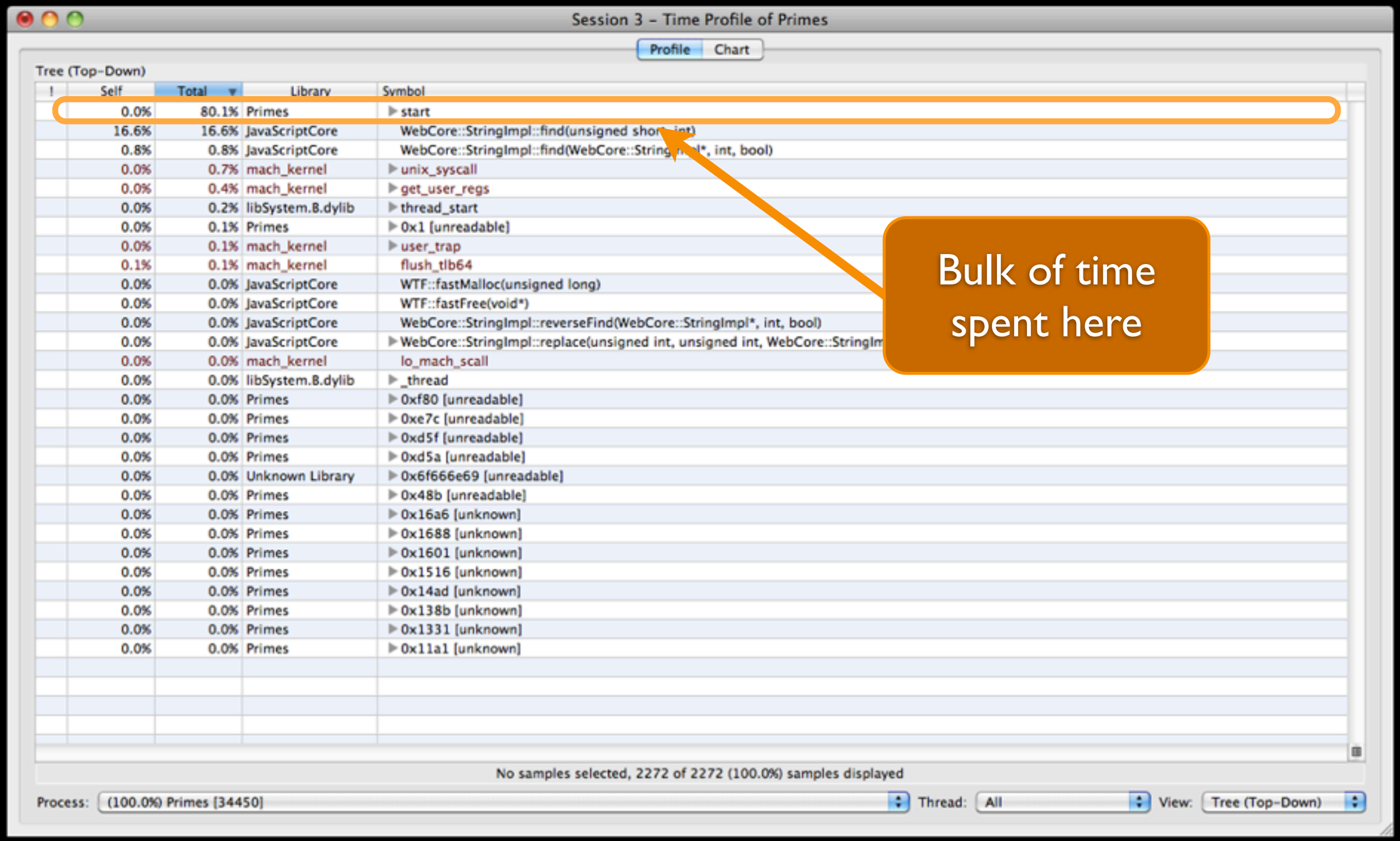
Select your app from this list

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 3 – Time Profile of Primes

Profile | Chart

Tree (Top-Down)

!	Self	Total	Library	Symbol
	0.0%	80.1%	Primes	start
	0.0%	80.1%	Primes	main
	0.0%	80.1%	UIKit	UIApplicationMain
	0.0%	80.1%	GraphicsServices	GSEventRun
	0.0%	80.1%	GraphicsServices	GSEventRunModal
	0.0%	80.1%	CoreFoundation	CFRunLoopRunInMode
	0.0%	80.1%	CoreFoundation	CFRunLoopRunSpecific
	0.0%	80.1%	CoreFoundation	__CFRunLoopRun
	0.0%	79.7%	CoreFoundation	__CFRunLoopDoSource1
	0.0%	79.7%	CoreFoundation	__CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE1_PERFORM_FUNCTION__
	0.0%	79.7%	GraphicsServices	PurpleEventCallback
	0.0%	79.7%	UIKit	_UIApplicationHandleEvent
	0.0%	79.0%	UIKit	-[UIApplication sendEvent:]
	0.0%	78.7%	UIKit	-[UIWindow _sendTouchesForEvent:]
	0.0%	78.7%	UIKit	-[UIControl touchesEnded:withEvent:]
	0.0%	78.7%	UIKit	-[UIControl(Internal) _sendActionsForEvents:withEvent:]
	0.0%	78.7%	UIKit	-[UIControl sendAction:to:forEvent:]
	0.0%	78.7%	UIKit	-[UIApplication sendAction:to:from:forEvent:]
	0.0%	78.7%	Primes	-[PrimesViewController findPrimeNumbers]
	0.0%	0.2%	UIKit	-[UIApplication handleEvent:withNewEvent:]
	0.0%	0.1%	UIKit	-[UIWindow sendEvent:]
	0.0%	0.7%	Foundation	-[NSAutoreleasePool release]
	0.0%	0.3%	CoreFoundation	__CFRunLoopDoObservers
	0.0%	0.1%	CoreFoundation	__CFRunLoopDoTimer
	16.6%	16.6%	JavaScriptCore	WebCore::StringImpl::find(unsigned short, int)
	0.8%	0.8%	JavaScriptCore	WebCore::StringImpl::find(WebCore::StringImpl*, int, bool)
	0.0%	0.7%	mach_kernel	unix_syscall
	0.0%	0.4%	mach_kernel	get_user_regs
	0.0%	0.2%	libSystem.B.dylib	thread_start
	0.0%	0.1%	Primes	0x1 [unreadable]
	0.0%	0.1%	mach_kernel	user_trap
	0.1%	0.1%	mach_kernel	flush_tlb64
	0.0%	0.0%	JavaScriptCore	WTF::fastMalloc(unsigned long)

Bulk of time spent in findPrimeNumbers

No samples selected, 2272 of 2272 (100.0%) samples displayed

Process: (100.0%) Primes [34450] Thread: All View: Tree (Top-Down)

Top Down View

Session 3 – Time Profile of Primes

Profile | Chart

Tree (Top-Down)

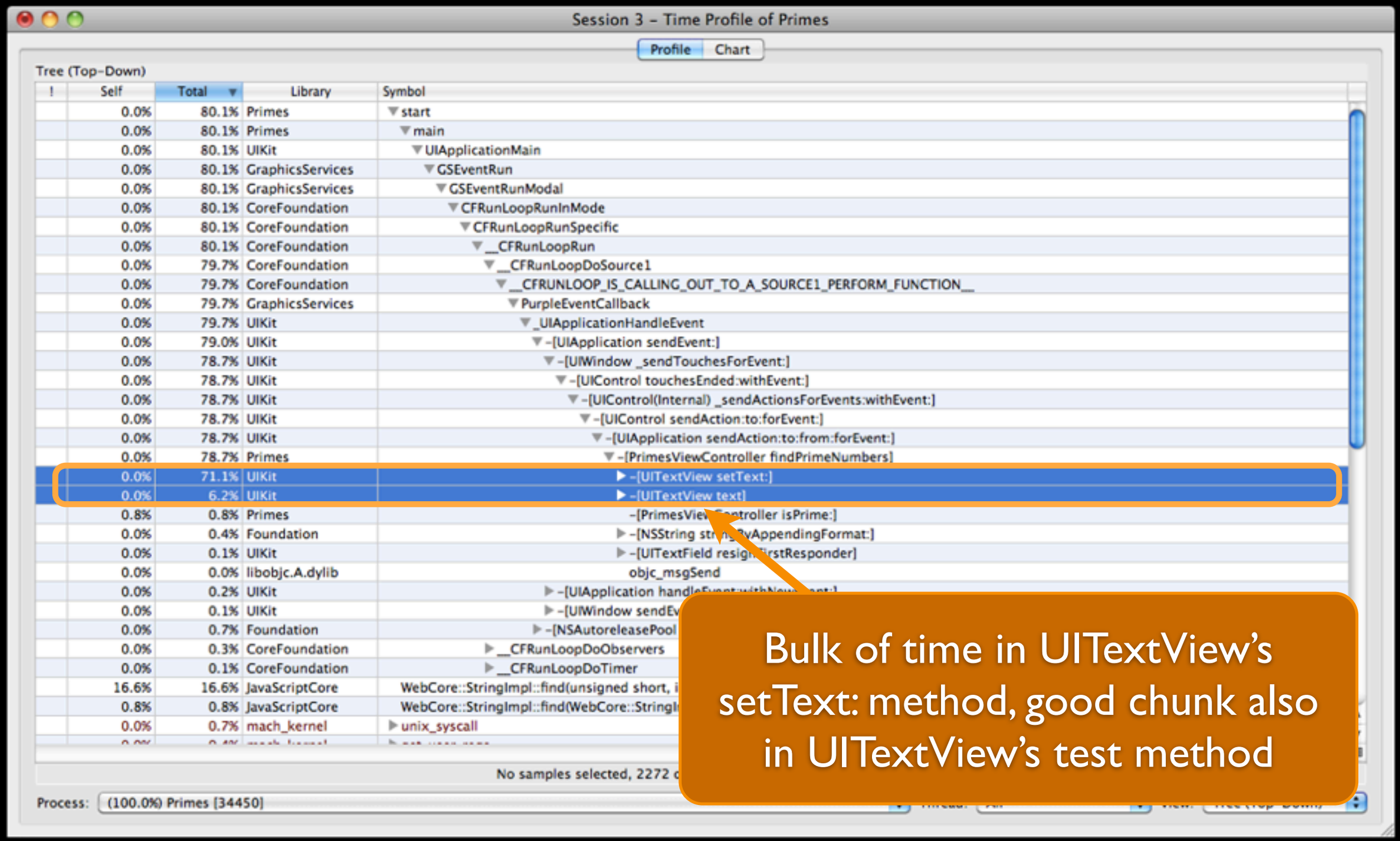
!	Self	Total	Library	Symbol
	0.0%	80.1%	Primes	start
	0.0%	80.1%	Primes	main
	0.0%	80.1%	UIKit	UIApplicationMain
	0.0%	80.1%	GraphicsServices	GSEventRun
	0.0%	80.1%	GraphicsServices	GSEventRunModal
	0.0%	80.1%	CoreFoundation	CFRunLoopRunInMode
	0.0%	80.1%	CoreFoundation	CFRunLoopRunSpecific
	0.0%	80.1%	CoreFoundation	__CFRunLoopRun
	0.0%	79.7%	CoreFoundation	__CFRunLoopDoSource1
	0.0%	79.7%	CoreFoundation	__CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE1_PERFORM_FUNCTION__
	0.0%	79.7%	GraphicsServices	PurpleEventCallback
	0.0%	79.7%	UIKit	_UIApplicationHandleEvent
	0.0%	79.0%	UIKit	-[UIApplication sendEvent:]
	0.0%	78.7%	UIKit	-[UIWindow _sendTouchesForEvent:]
	0.0%	78.7%	UIKit	-[UIControl touchesEnded:withEvent:]
	0.0%	78.7%	UIKit	-[UIControl(Internal) _sendActionsForEvents:withEvent:]
	0.0%	78.7%	UIKit	-[UIControl sendAction:to:forEvent:]
	0.0%	78.7%	UIKit	-[UIApplication sendAction:to:from:forEvent:]
	0.0%	78.7%	Primes	-[PrimesViewController findPrimeNumbers]
	0.0%	71.1%	UIKit	-[UITextView setText:]
	0.0%	6.2%	UIKit	-[UITextView text]
	0.8%	0.8%	Primes	-[PrimesViewController isPrime:]
	0.0%	0.4%	Foundation	-[NSString stringByAppendingFormat:]
	0.0%	0.1%	UIKit	-[UITextField resignFirstResponder]
	0.0%	0.0%	libobjc.A.dylib	objc_msgSend
	0.0%	0.2%	UIKit	-[UIApplication handleEventWithNewEvent:]
	0.0%	0.1%	UIKit	-[UIWindow sendEvent:]
	0.0%	0.7%	Foundation	-[NSAutoreleasePool release]
	0.0%	0.3%	CoreFoundation	__CFRunLoopDoObservers
	0.0%	0.1%	CoreFoundation	__CFRunLoopDoTimer
	16.6%	16.6%	JavaScriptCore	WebCore::StringImpl::find(unsigned short, int)
	0.8%	0.8%	JavaScriptCore	WebCore::StringImpl::find(WebCore::StringImpl*, int, bool)
	0.0%	0.7%	mach_kernel	unix_syscall

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

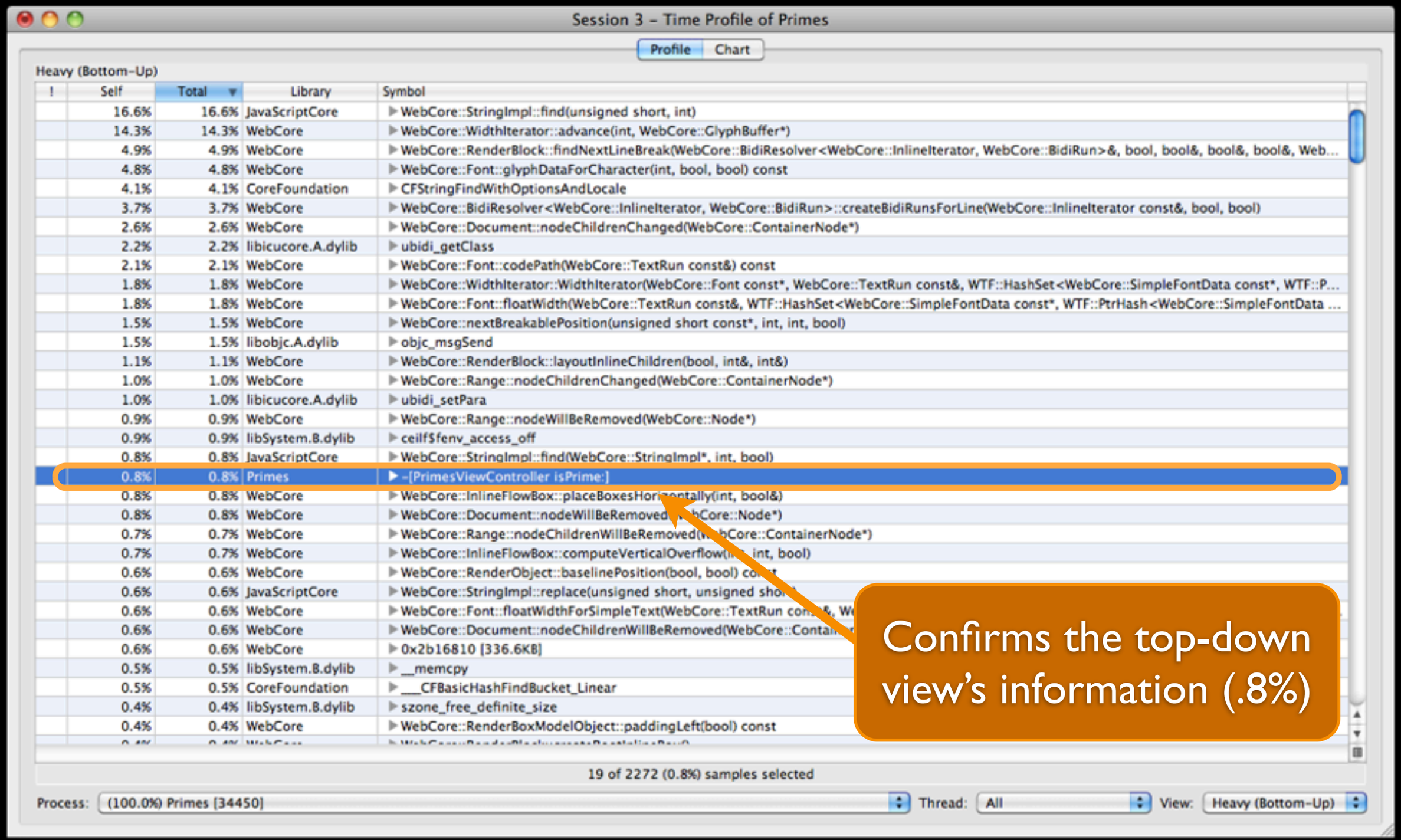
19 of 2272 (0.8%) samples selected

Process: (100.0%) Primes [34450] Thread: All View: Tree (Top-Down)

Top Down View

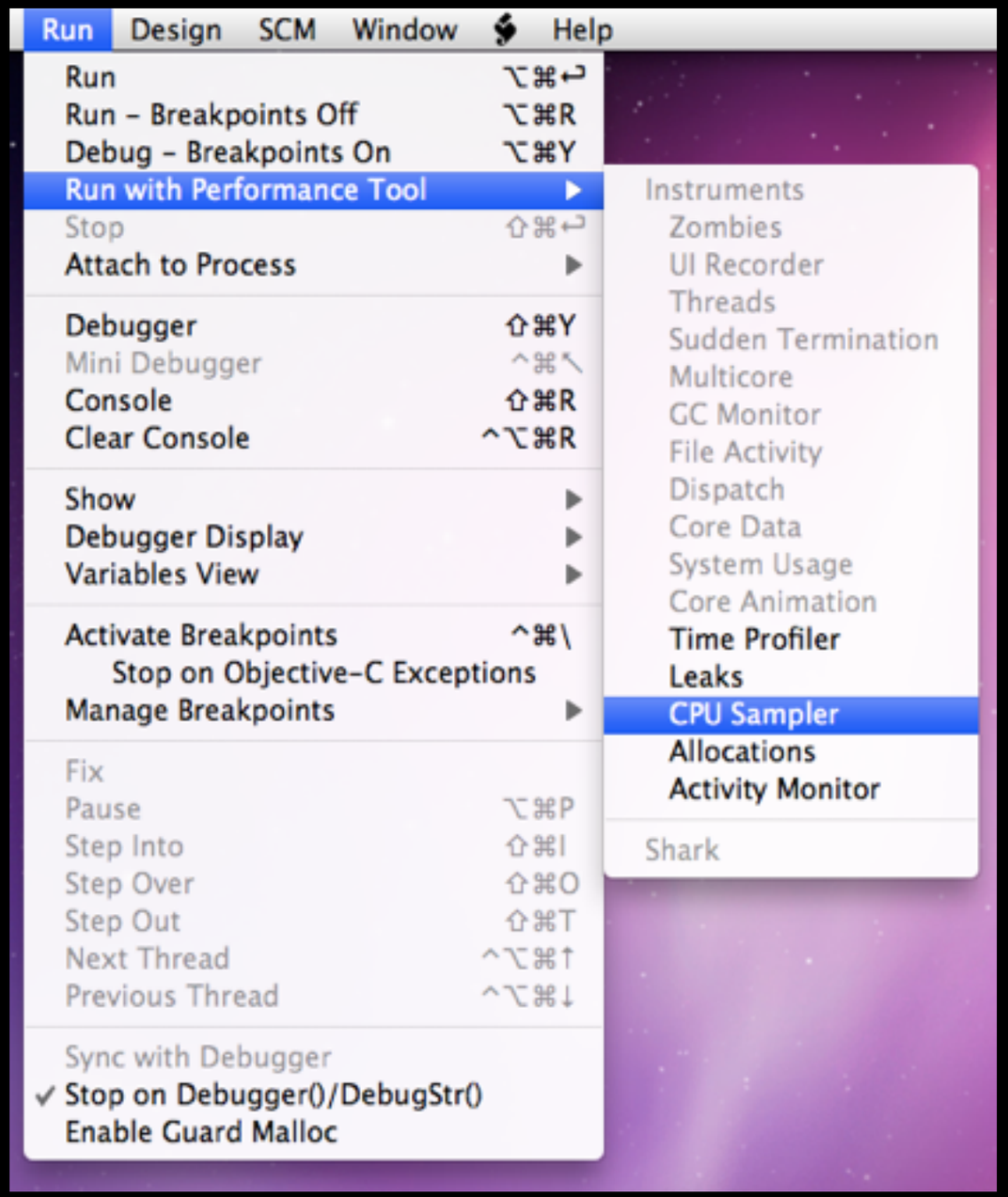


Bottom-Up View

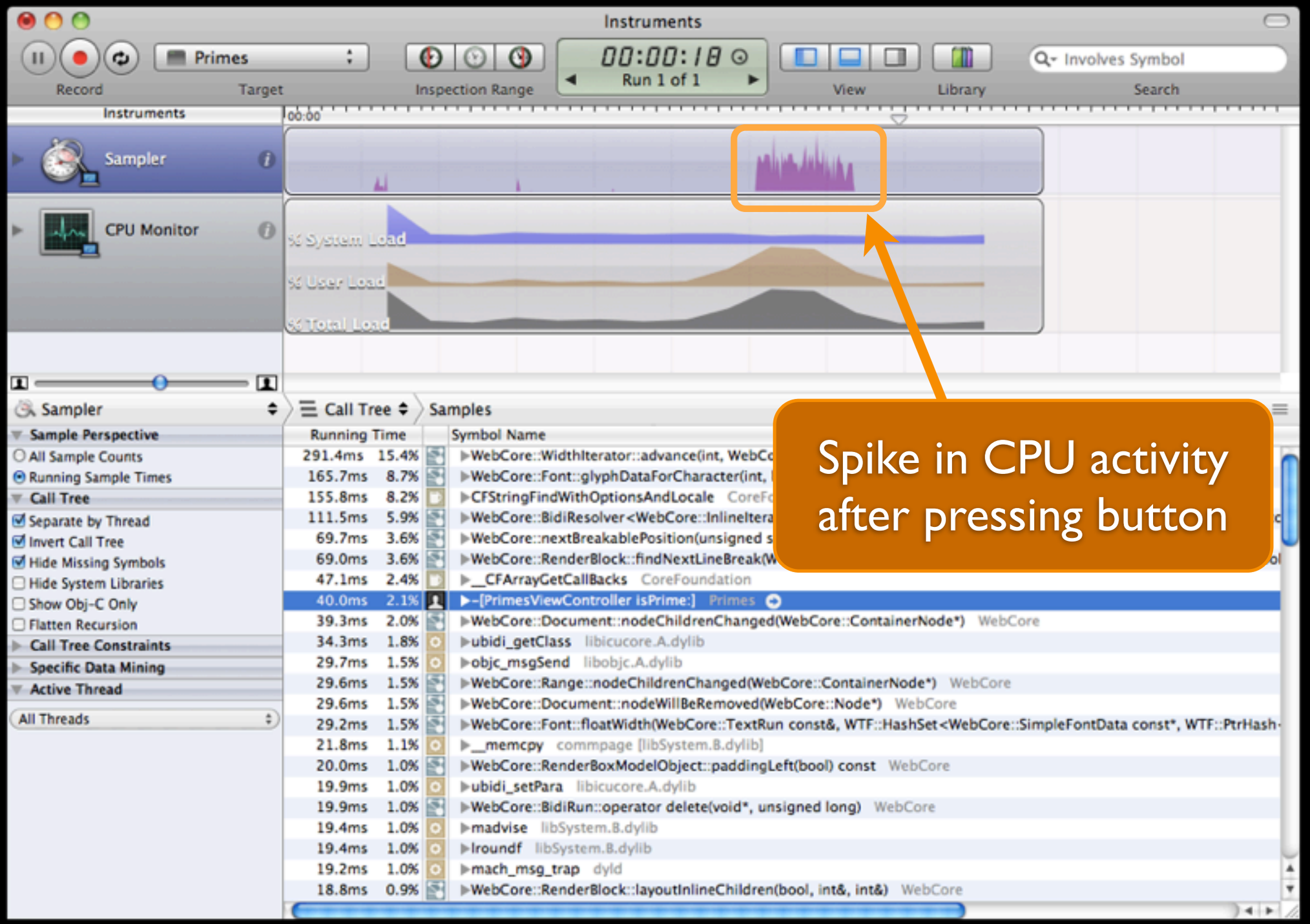


Sampling with Instruments

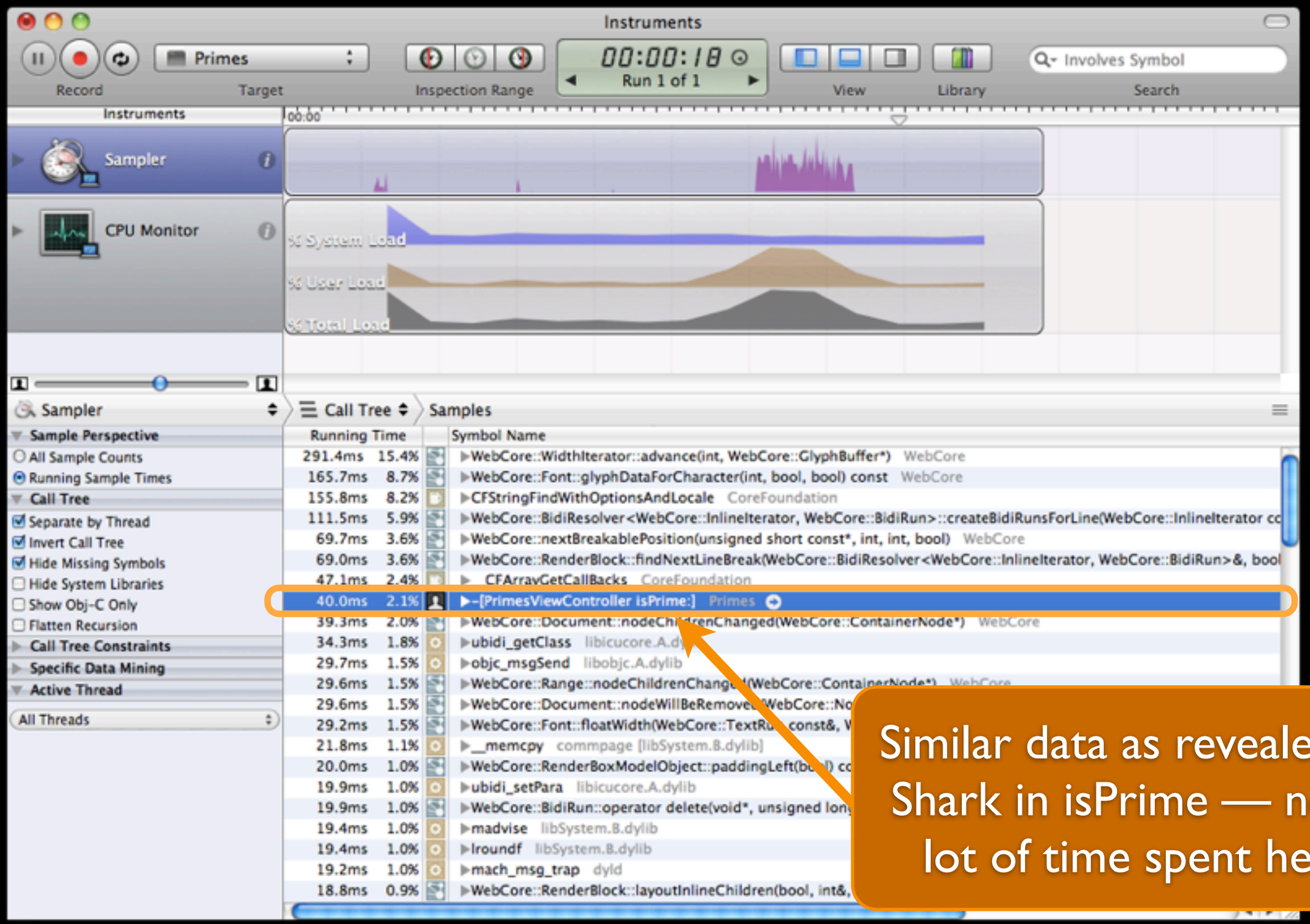
- Instruments is a bit more integrated with Xcode than Shark is
- Simply select CPU Sampler under the Run with Performance Tool item from the Run menu
- Unlike Shark, Instruments will automatically (re)start the application



Instruments



Instruments

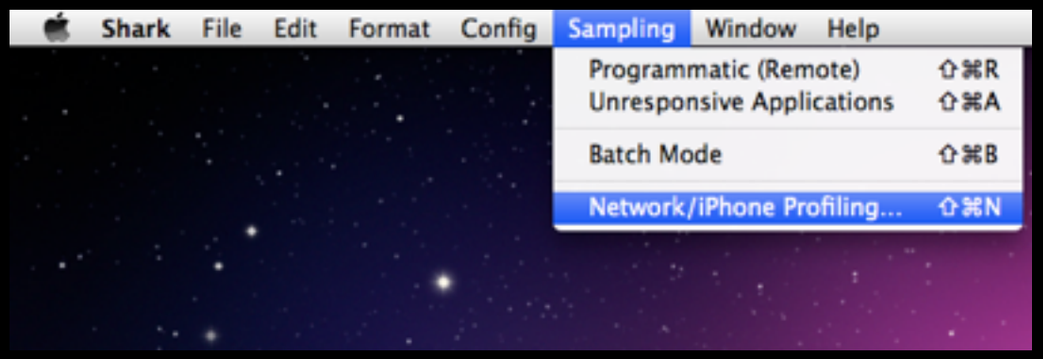


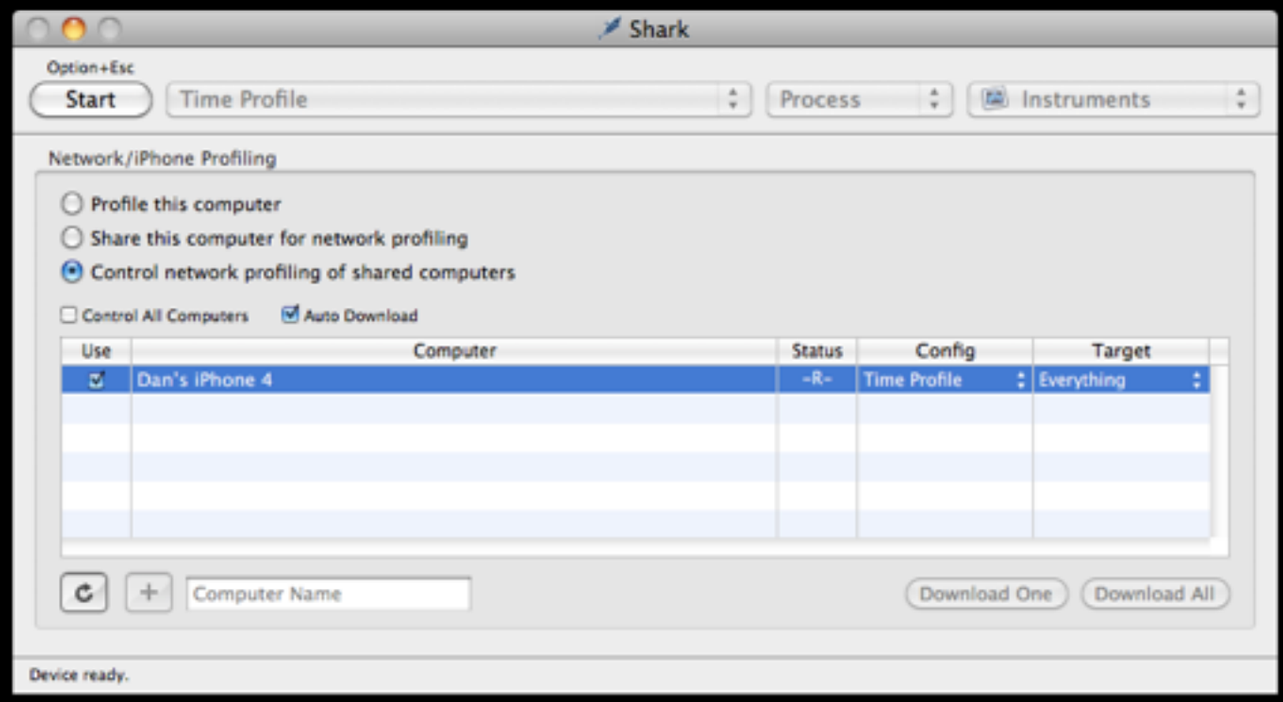
On Device Performance

- But wait!
- Didn't we say that on-device performance should be used for gather performance data as well?
  - Yes!

Shark On-Device Profiling

- You used to be able to perform on-device profiling using the Shark tool, but it looks like this is non-functional (at least with my iOS 4.2.1 install and my iPhone 4)
- Seems in-line expectations that Apple is phasing out Shark in favor of Instruments



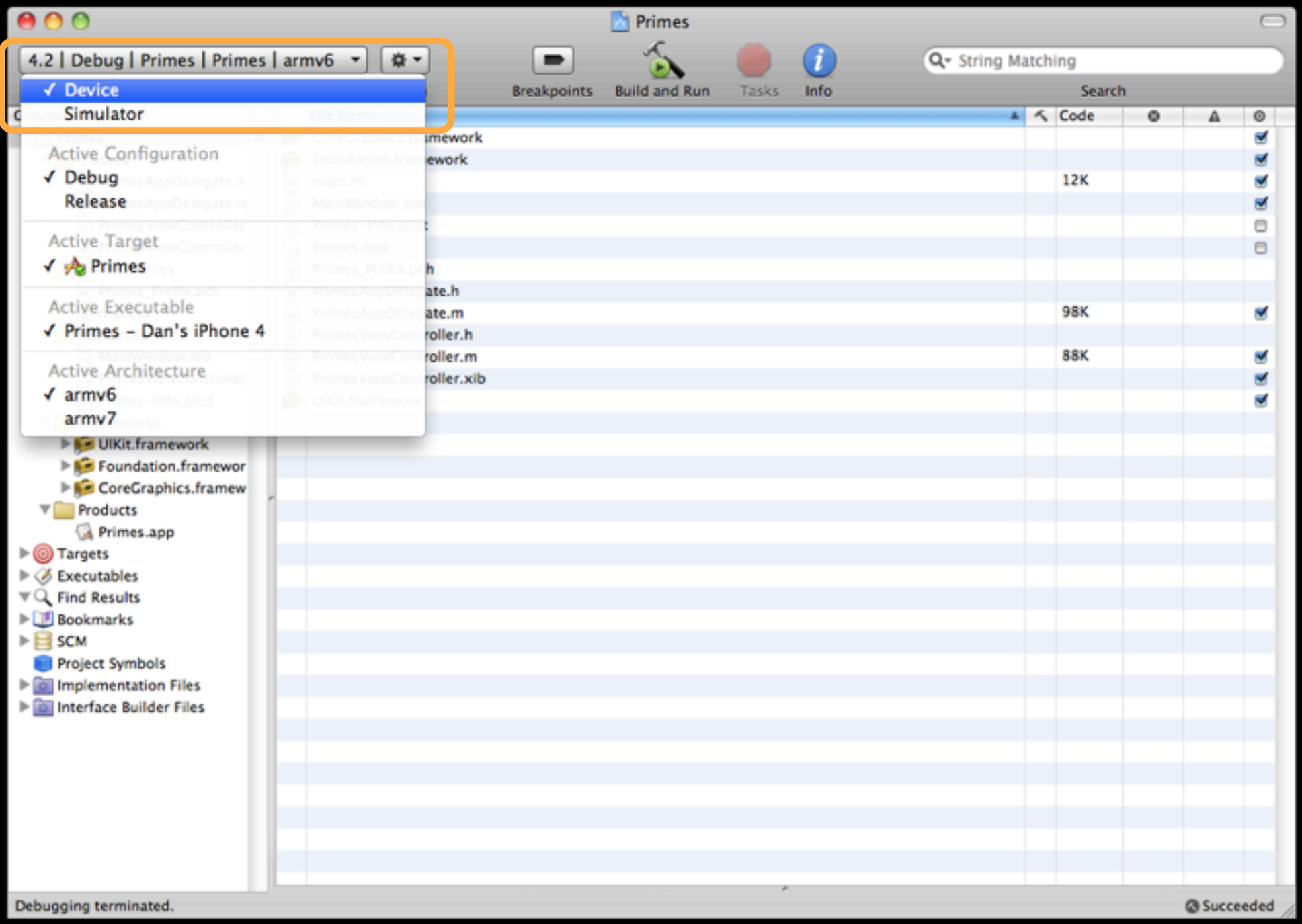


Instruments On-Device Profiling

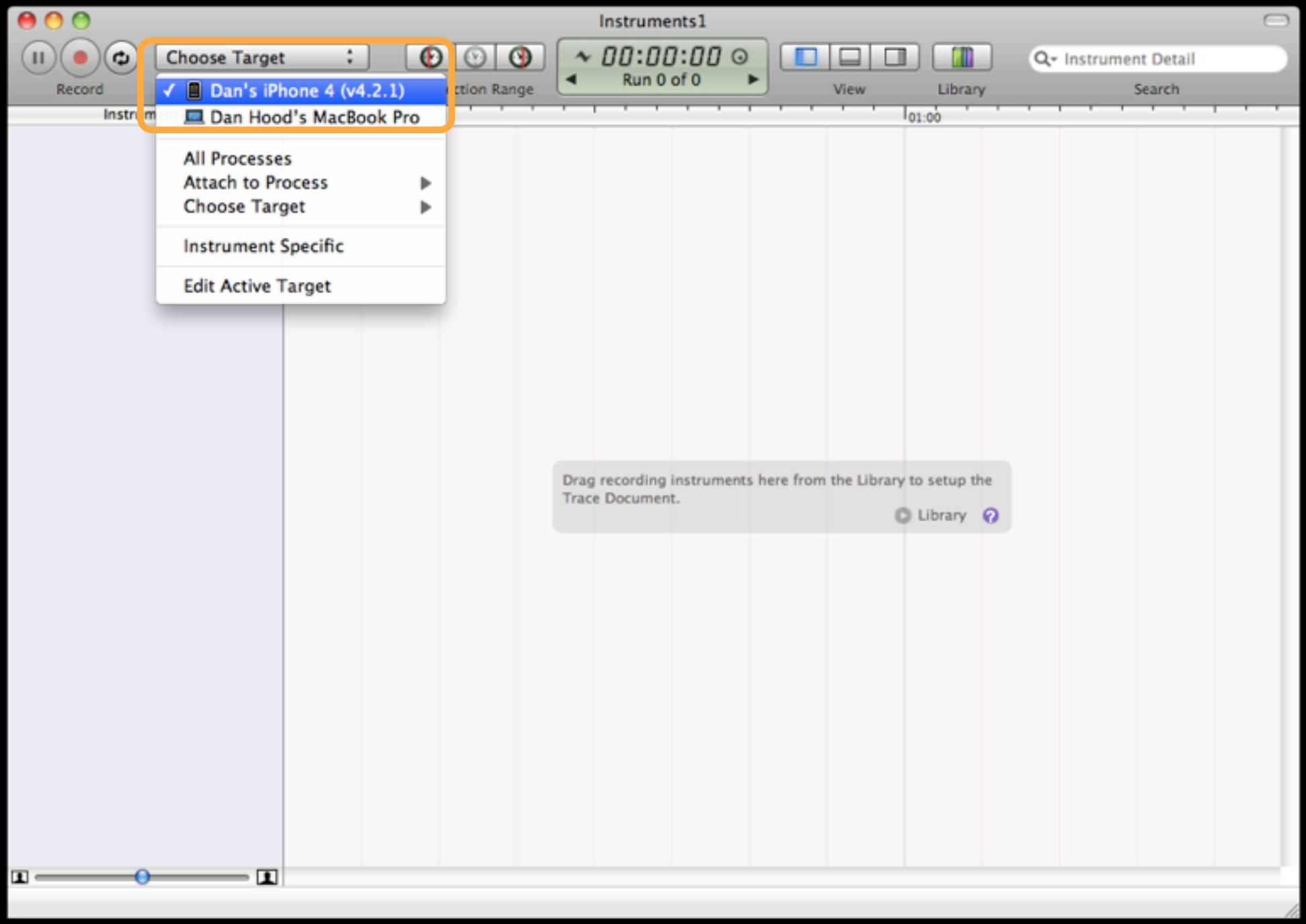
- The easiest way to do on-device sampling from Instruments is to select the Device setting under Overview, and select the Leaks option from the Run with Performance Tool option
- From Instruments...
  - Be sure that the Device (as opposed to the simulator) is selected
  - Set the launch executable to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

These latter two seem to be done automatically with newer SDK releases

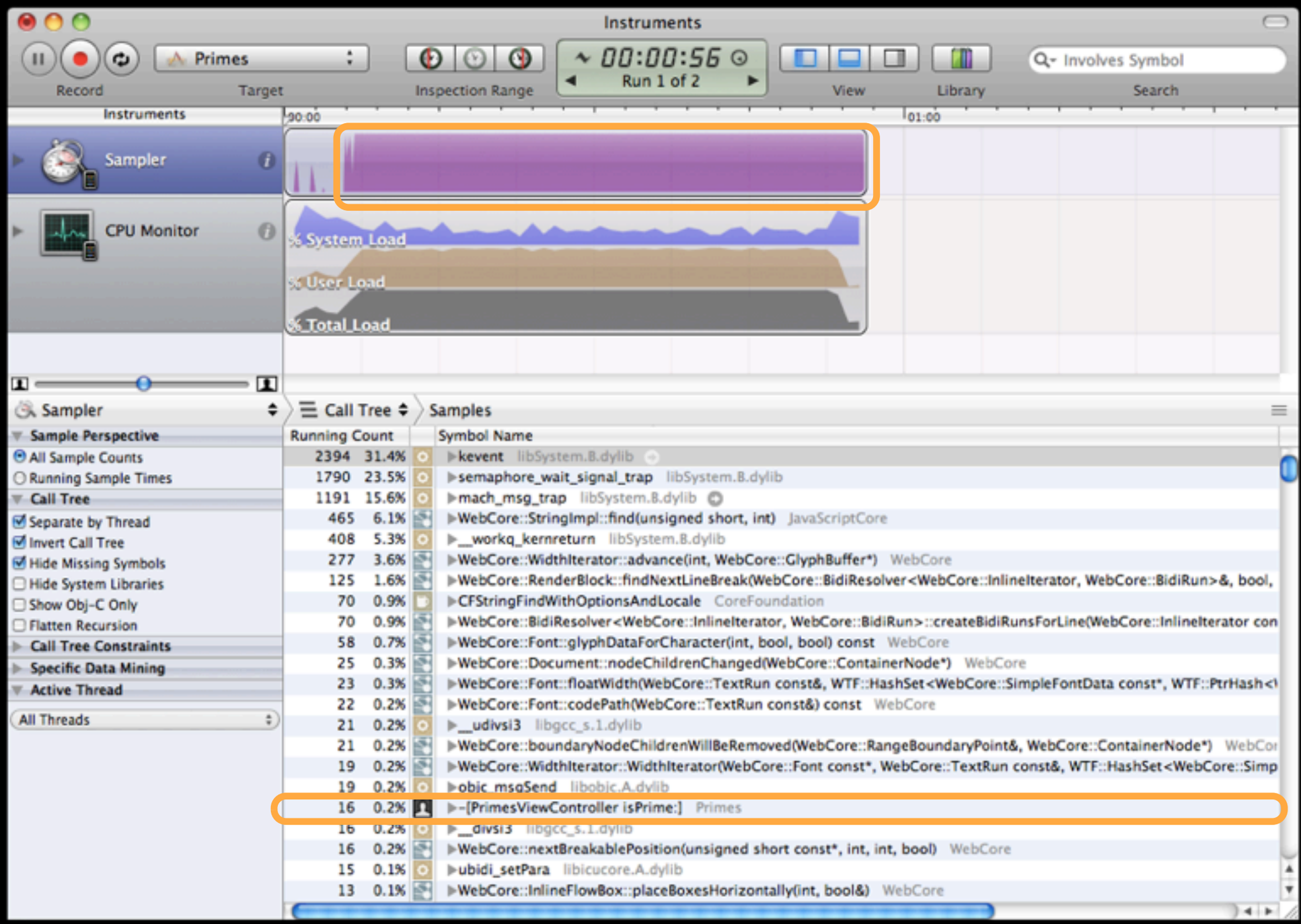
Selecting On-Device Development in Xcode



Running on my iPhone



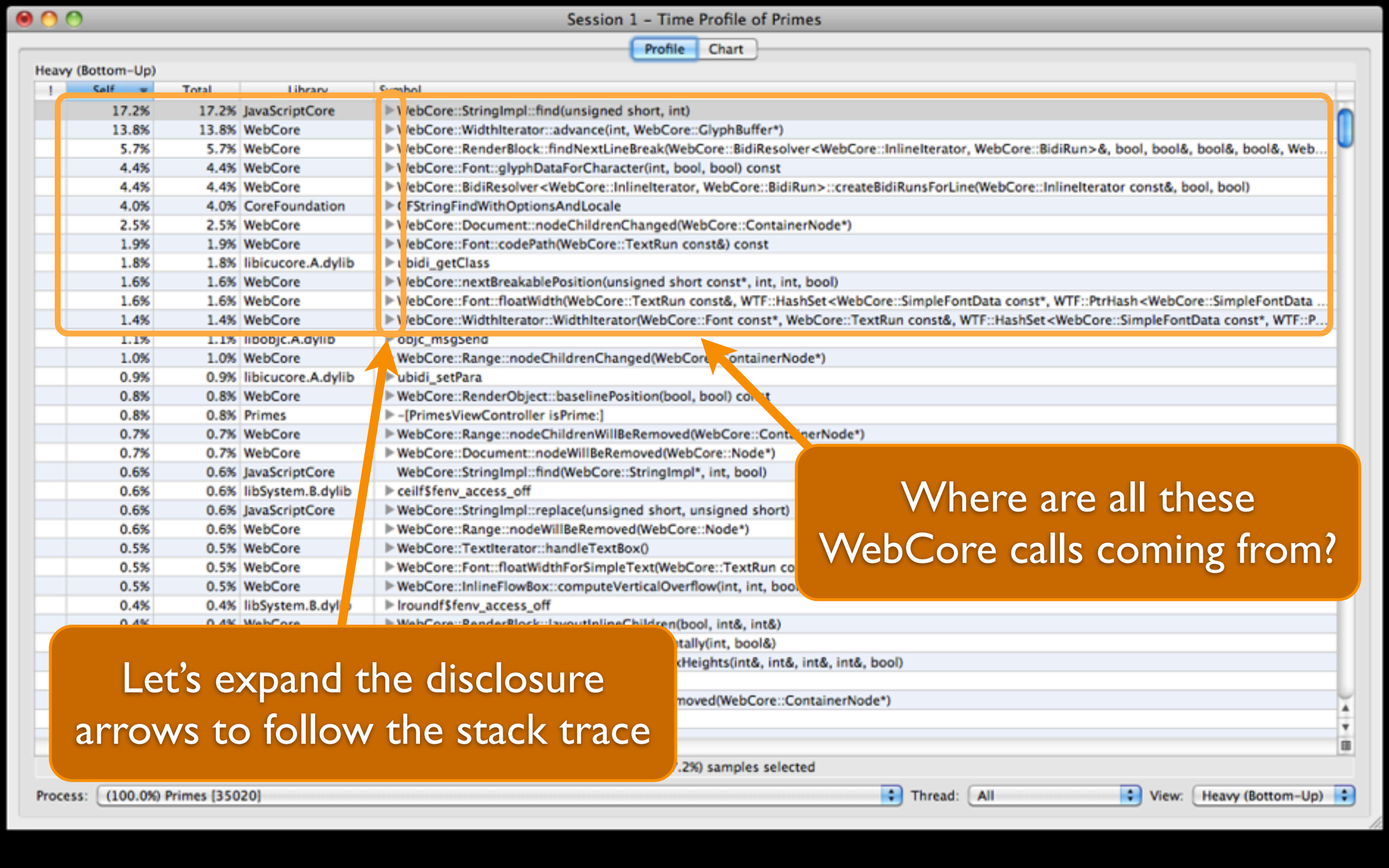
Instruments On-Device Profiling Results



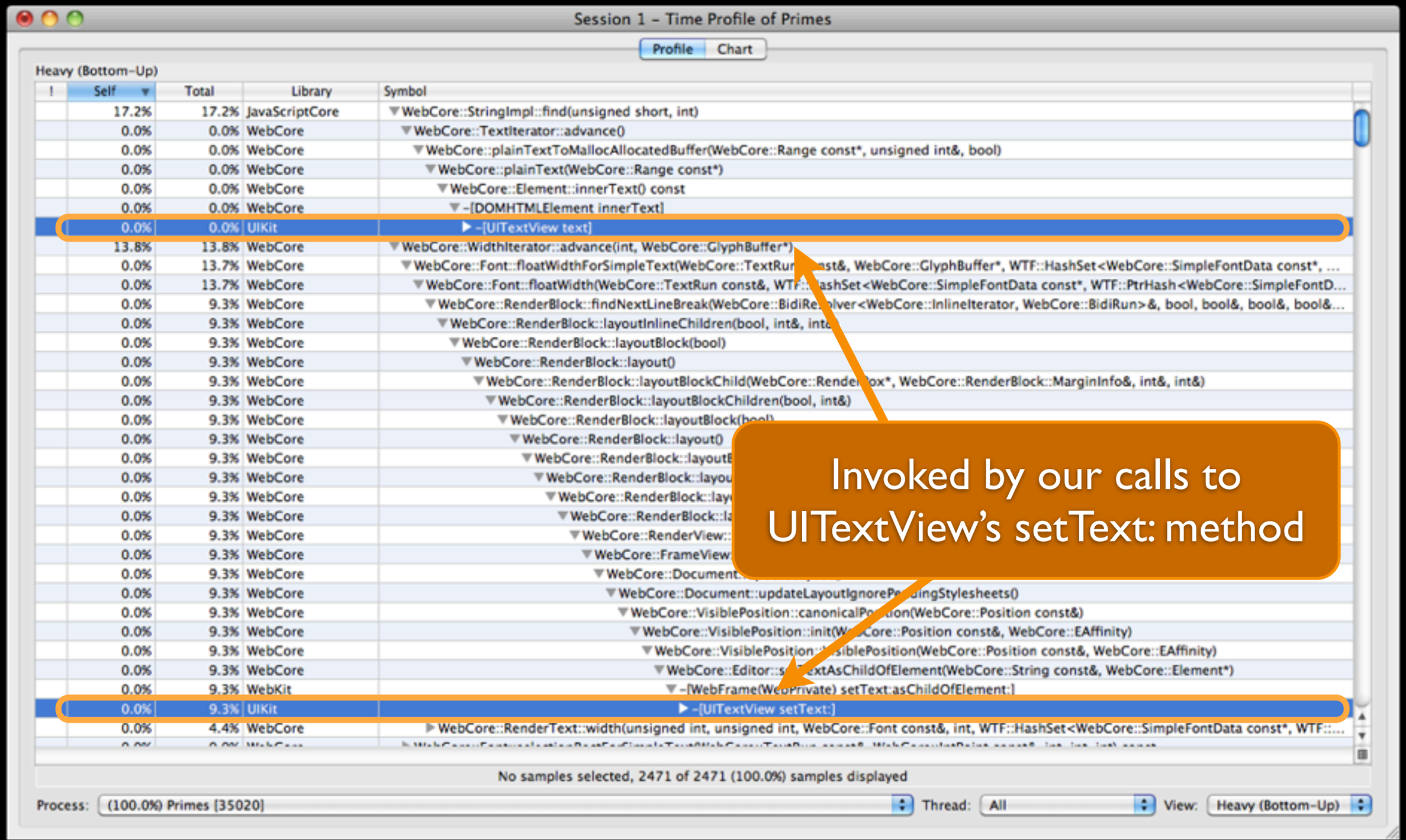
Observations

- 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 15 CPU hogs are from the WebCore library
  - These top 15 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...

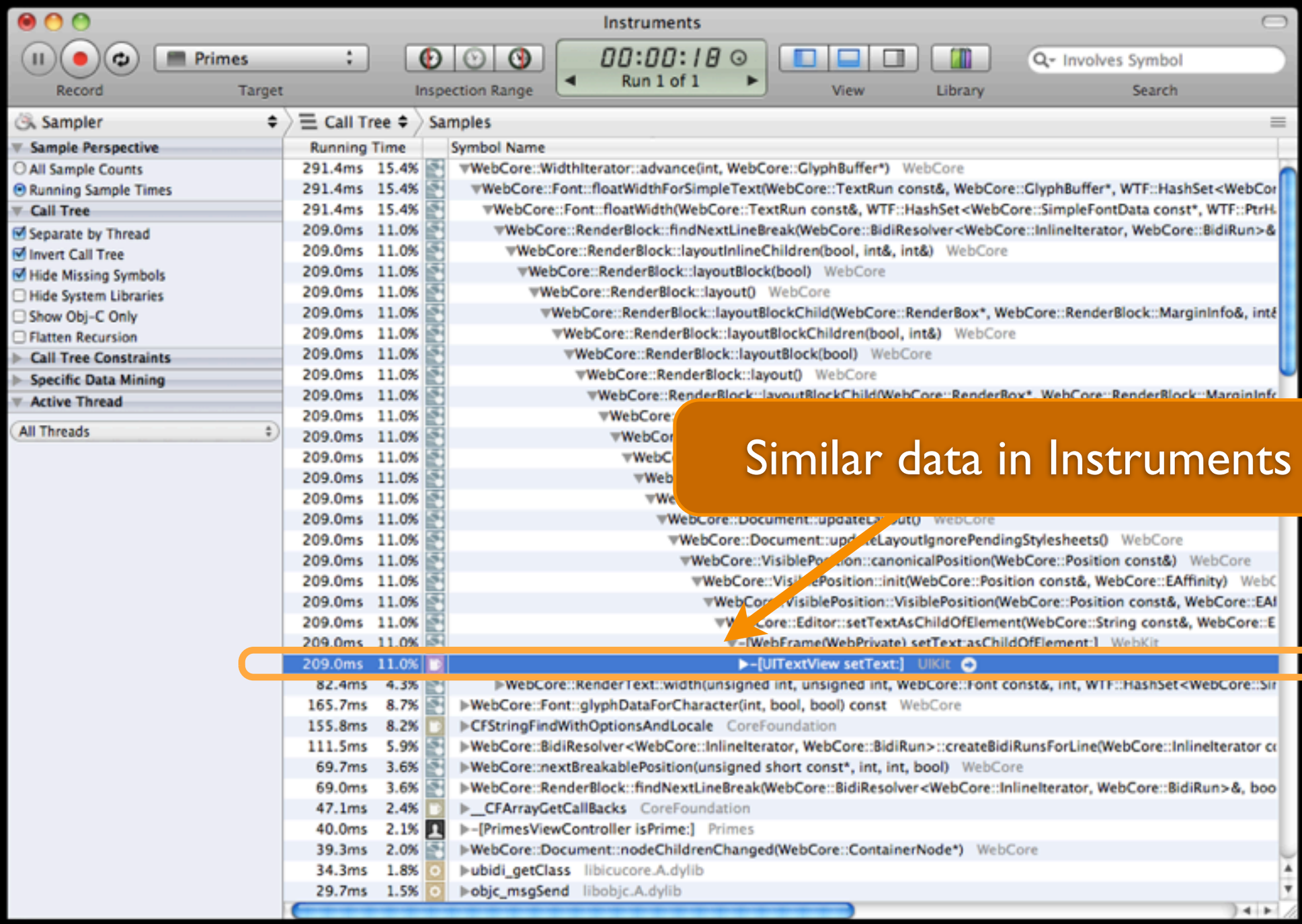
Tracing the Source (Shark)



Tracing the Source (Shark)



Tracing the Source (Instruments)



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:

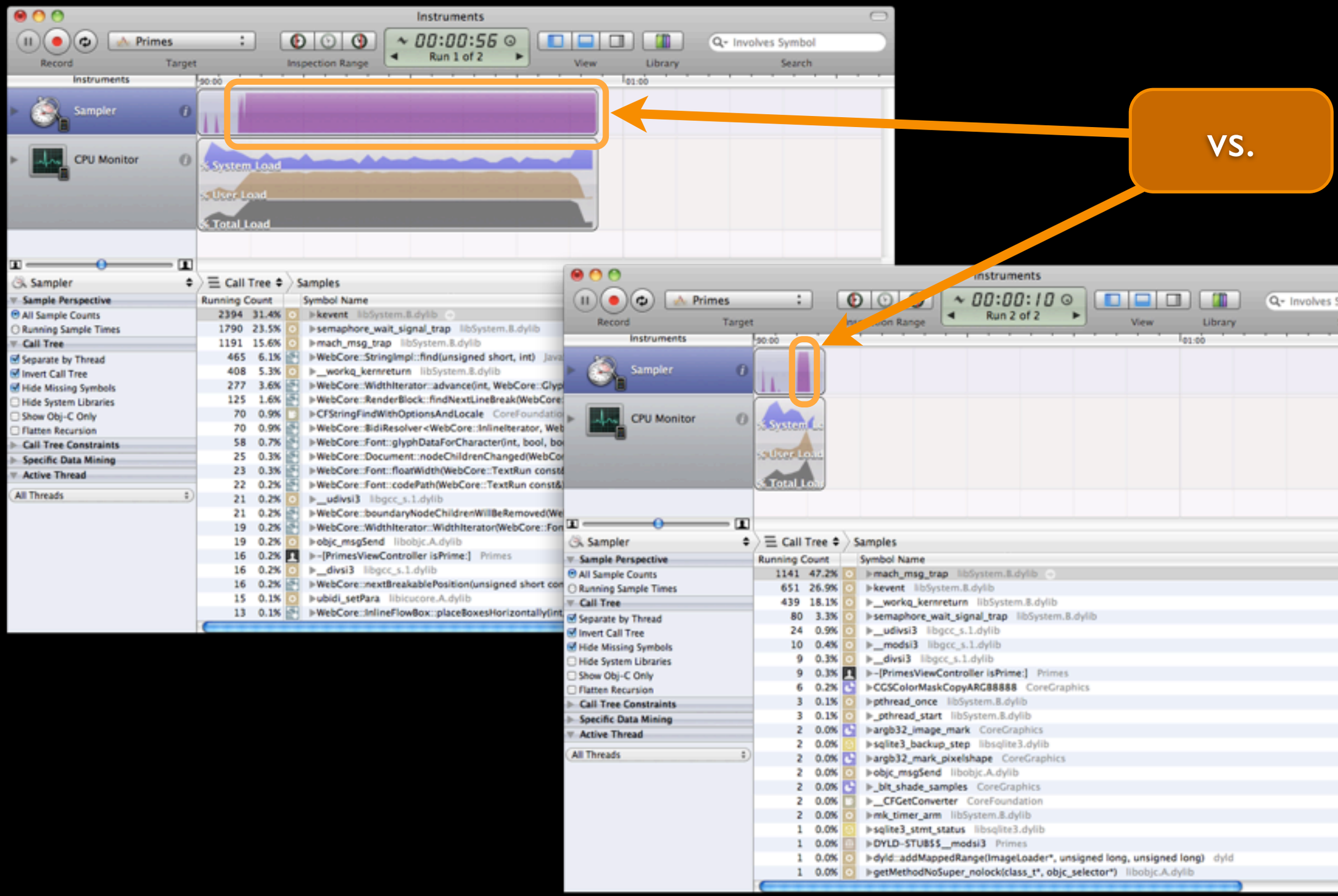
New & Improved PrimesViewController.m

```
#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
```

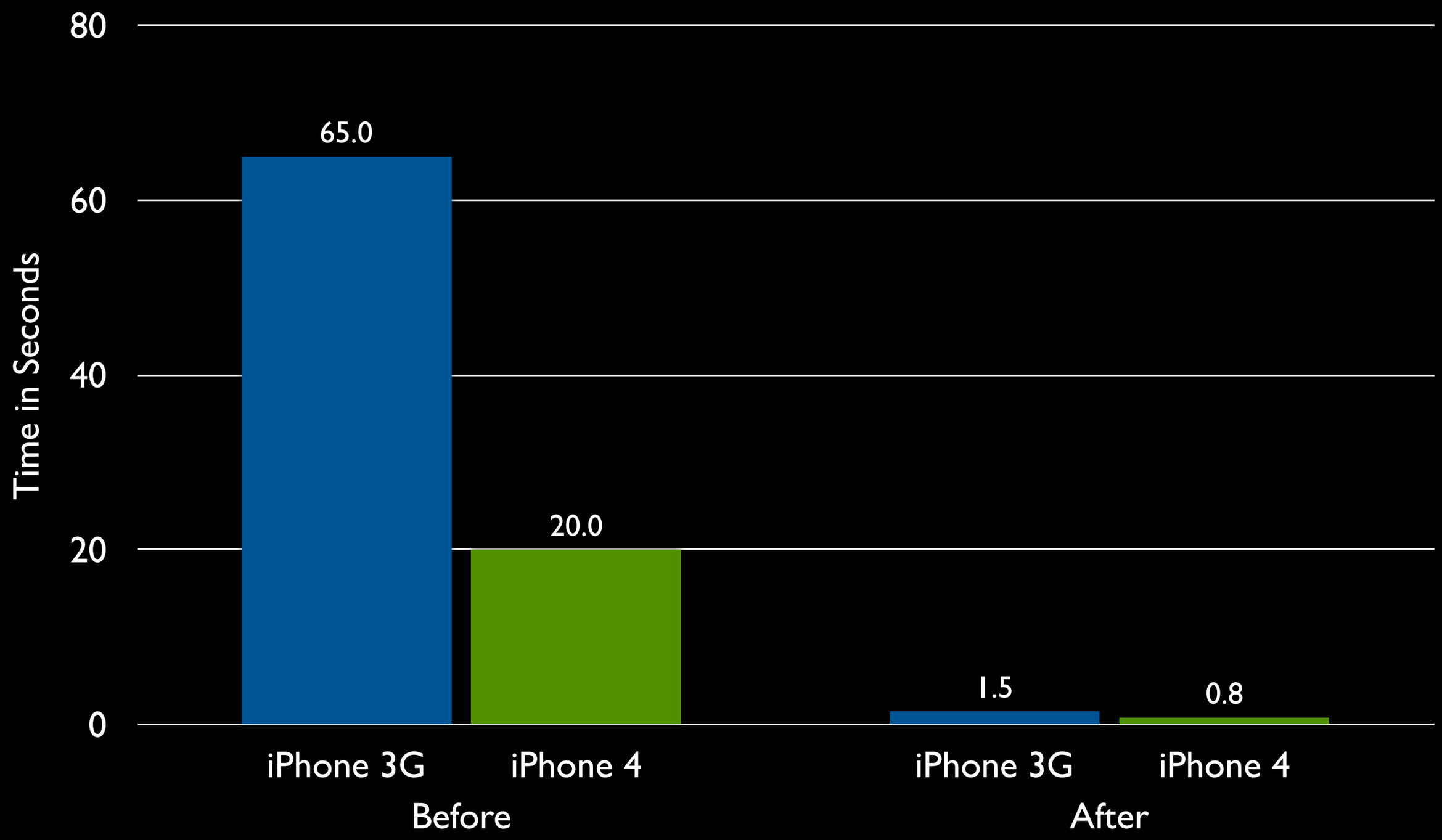
New & Improved Version in iPhone Simulator

- When running on my unibody MacBook Pro it took...
- Near instantaneous to get through 10,000 numbers (was ~2.5 seconds)
- Less than a second to get through 50,000 numbers (was ~70 seconds)

Before & After (On-Device)



New & Improved On-Device Performance (Primes up to10,000)



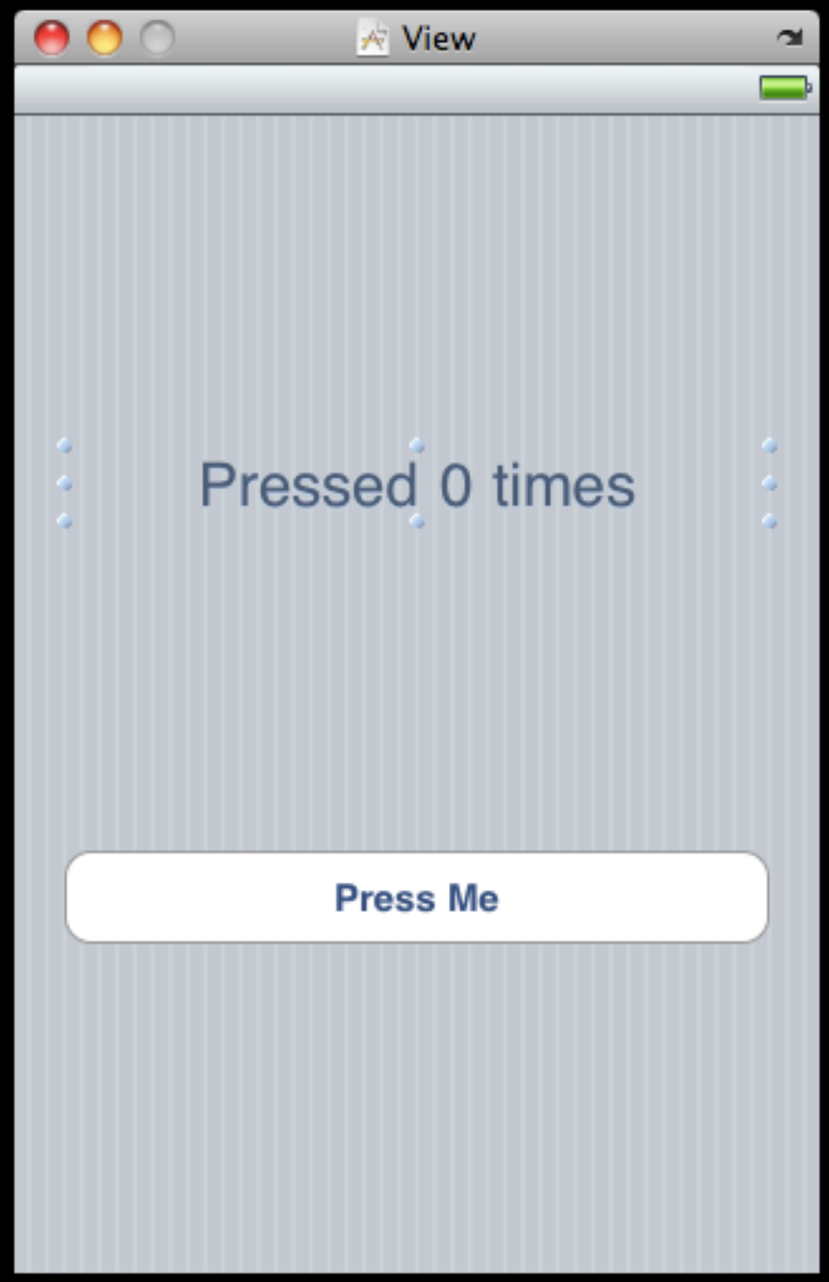
Detecting Memory Leaks

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

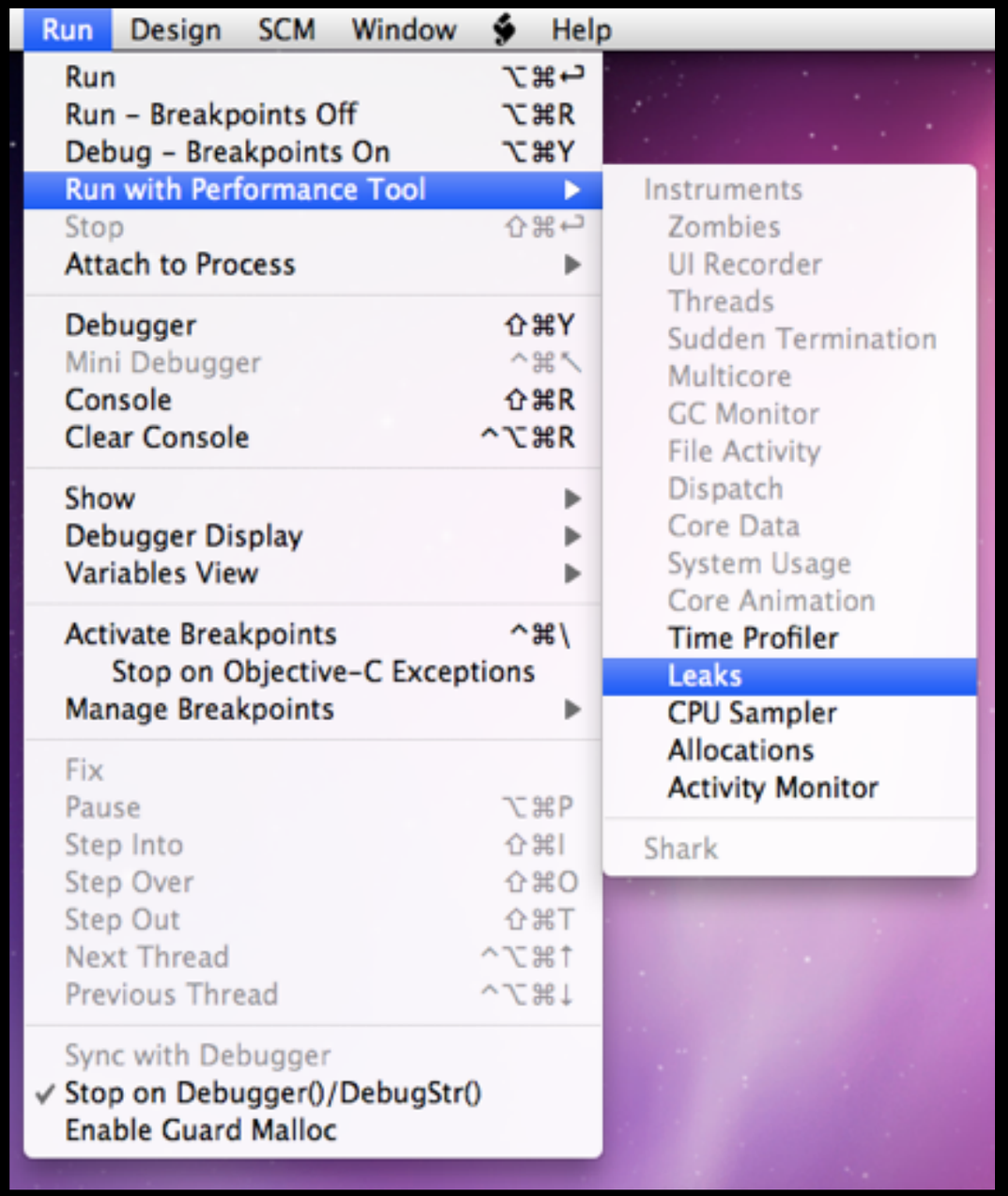
```
#import <UIKit/UIKit.h>

@interface LeakViewController : UIViewController {

  int count;

}

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

- (IBAction)buttonPress;

@end
```

LeakViewController.m

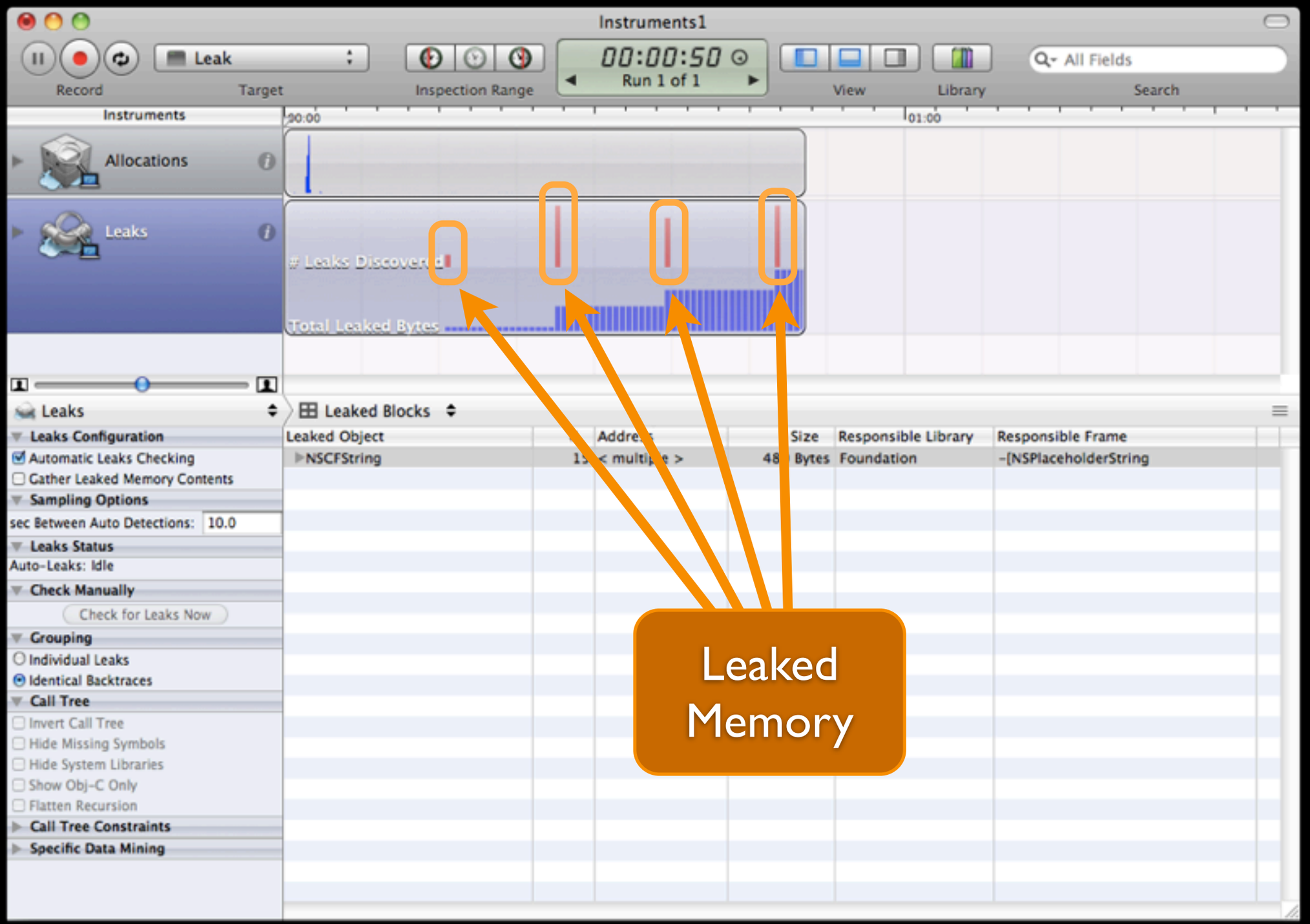
```
#import "LeakViewController.h"

@implementation LeakViewController

@synthesize label;

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

/* ... */

@end
```

Run with Performance Tools

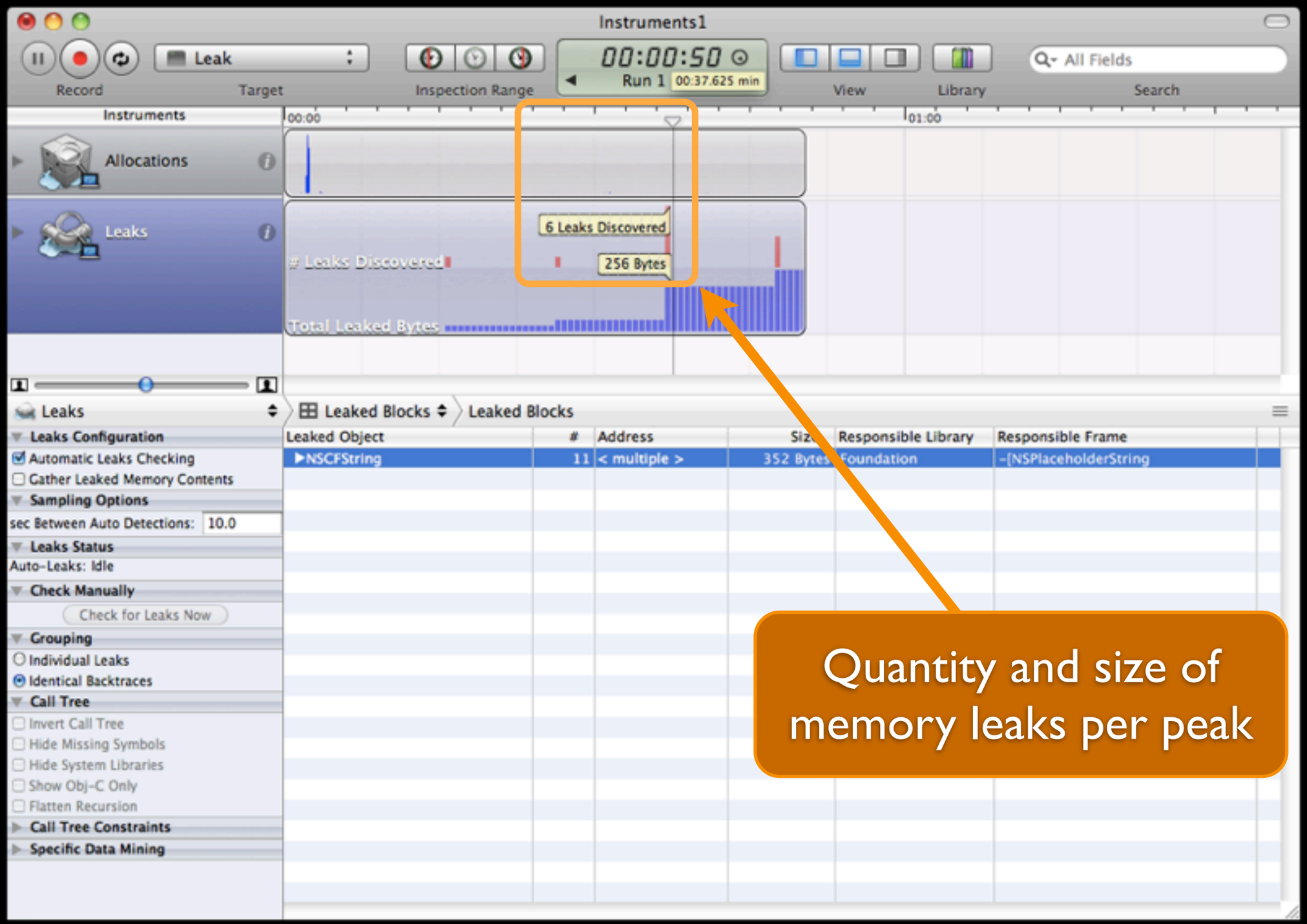
- The Leaks configuration can be opened from the Run menu under Xcode



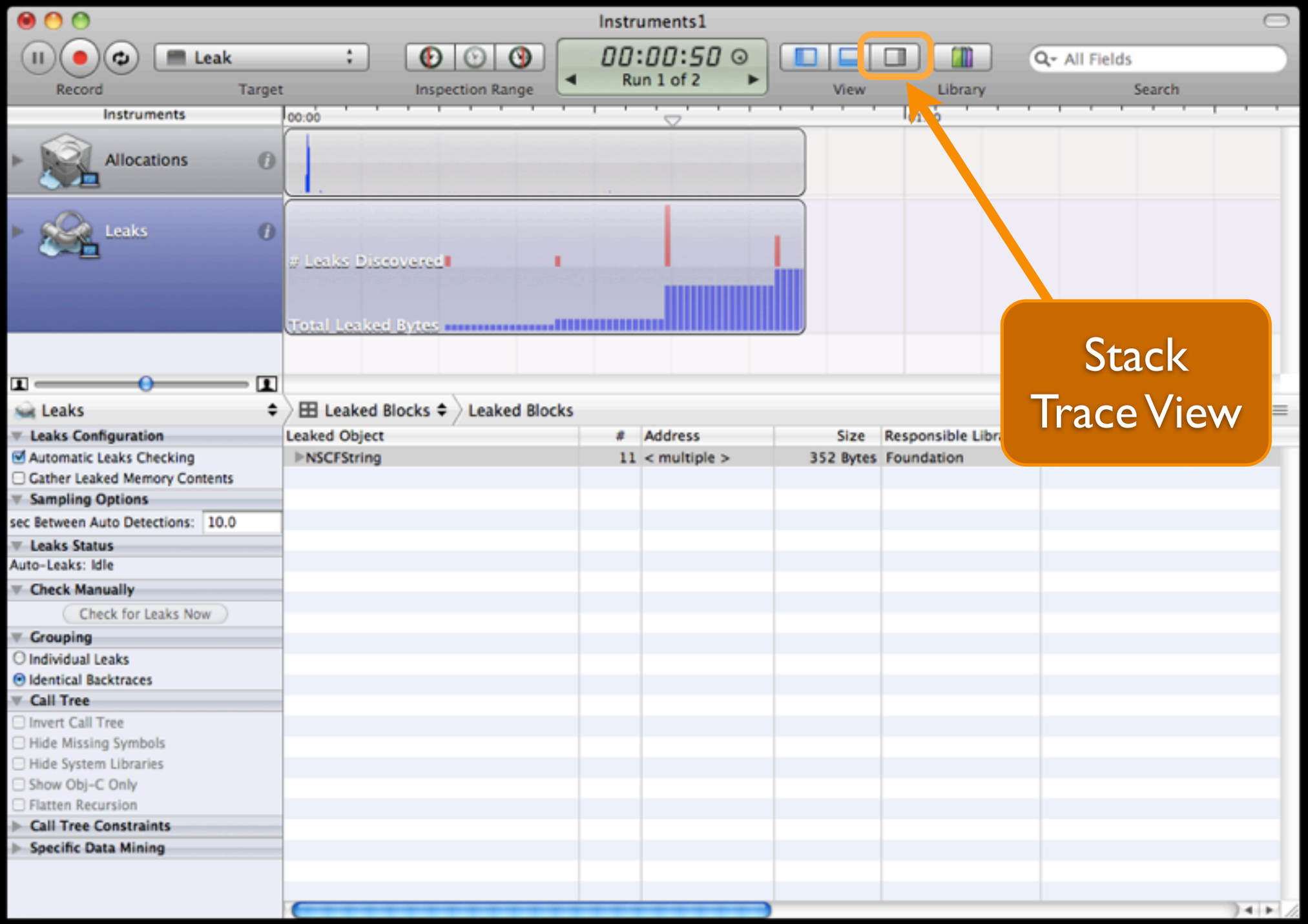
Instruments Output



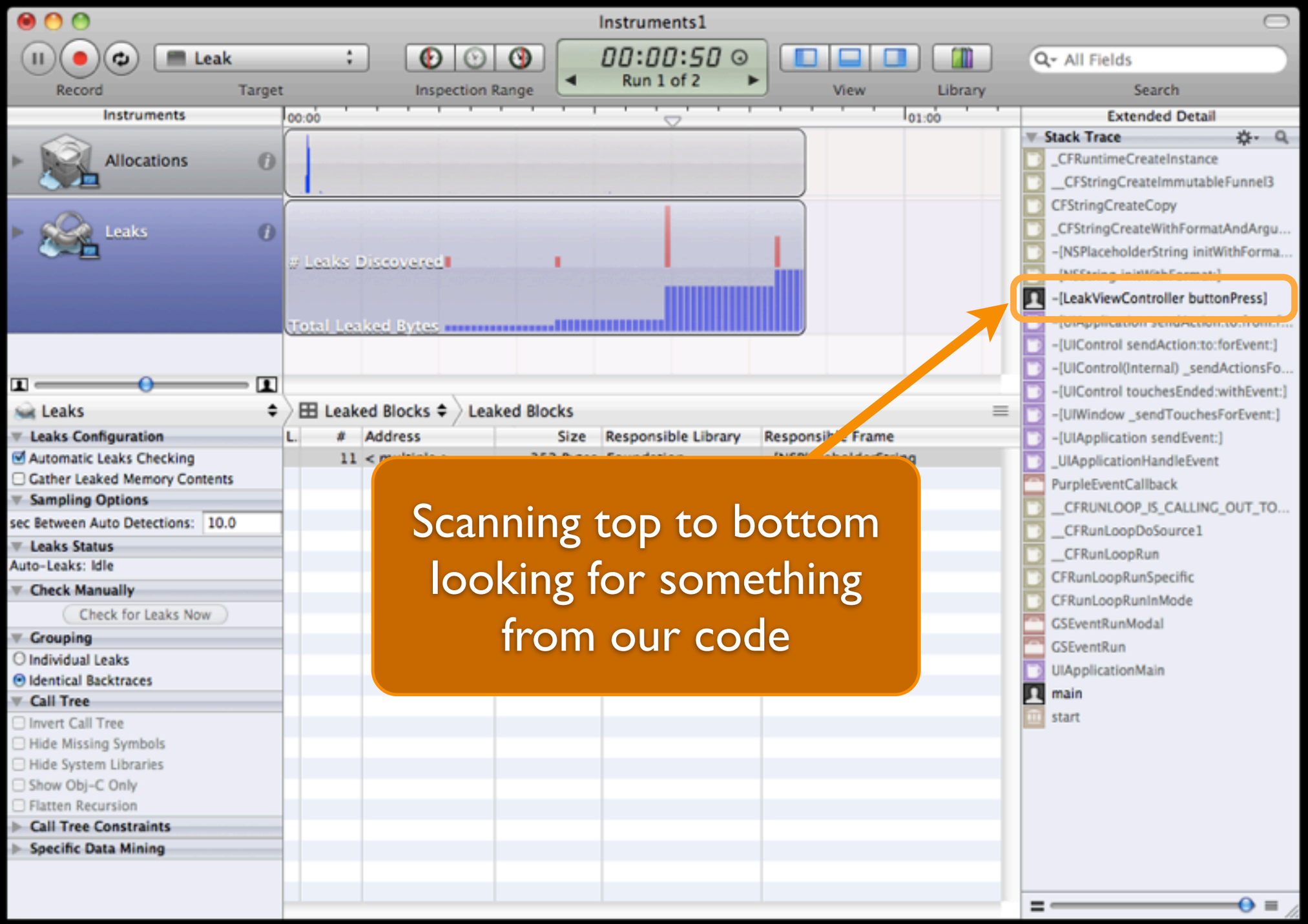
Dragging the Slider



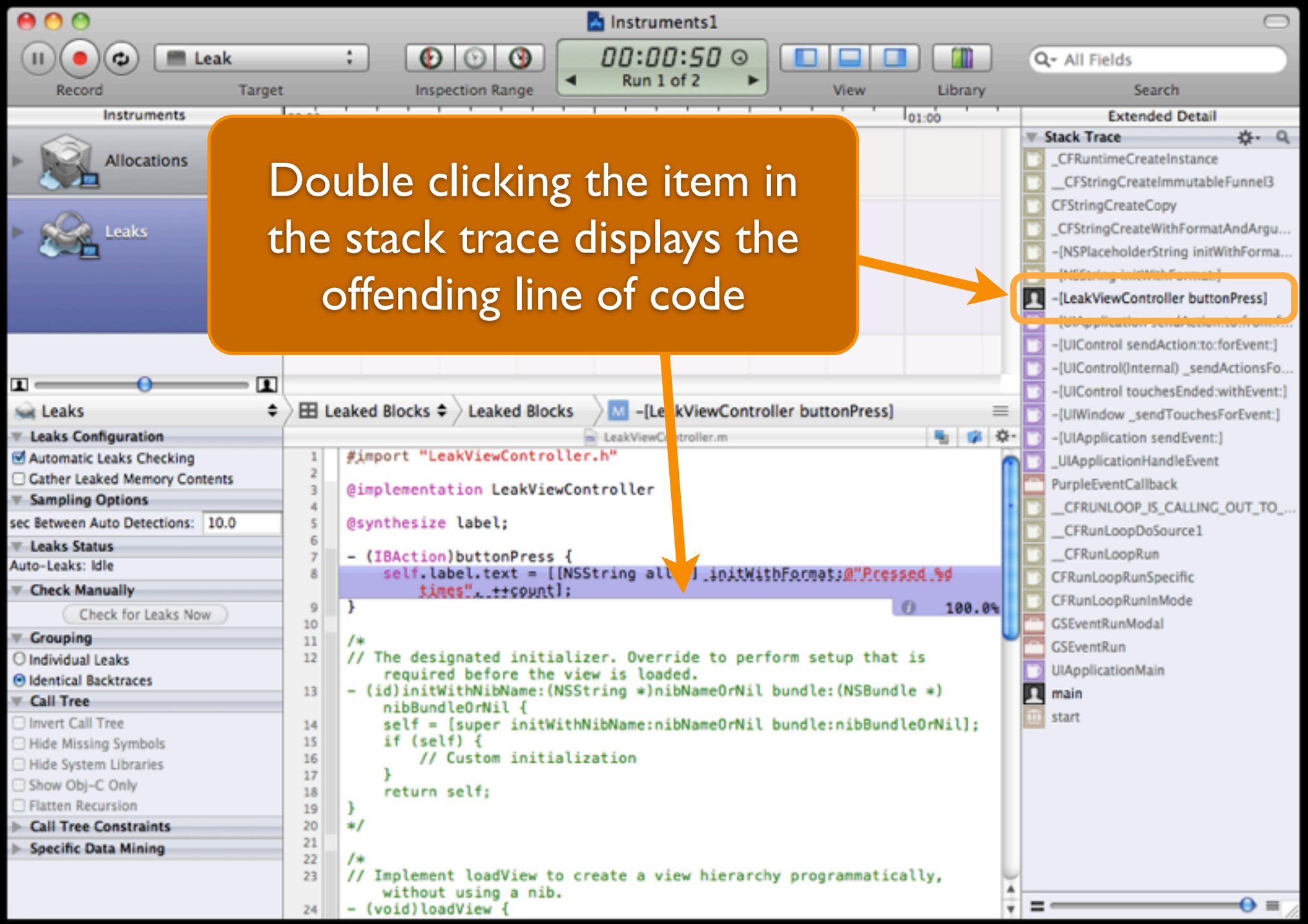
Enabling Stack Traces



Locating App Code



Tracing Back to Code

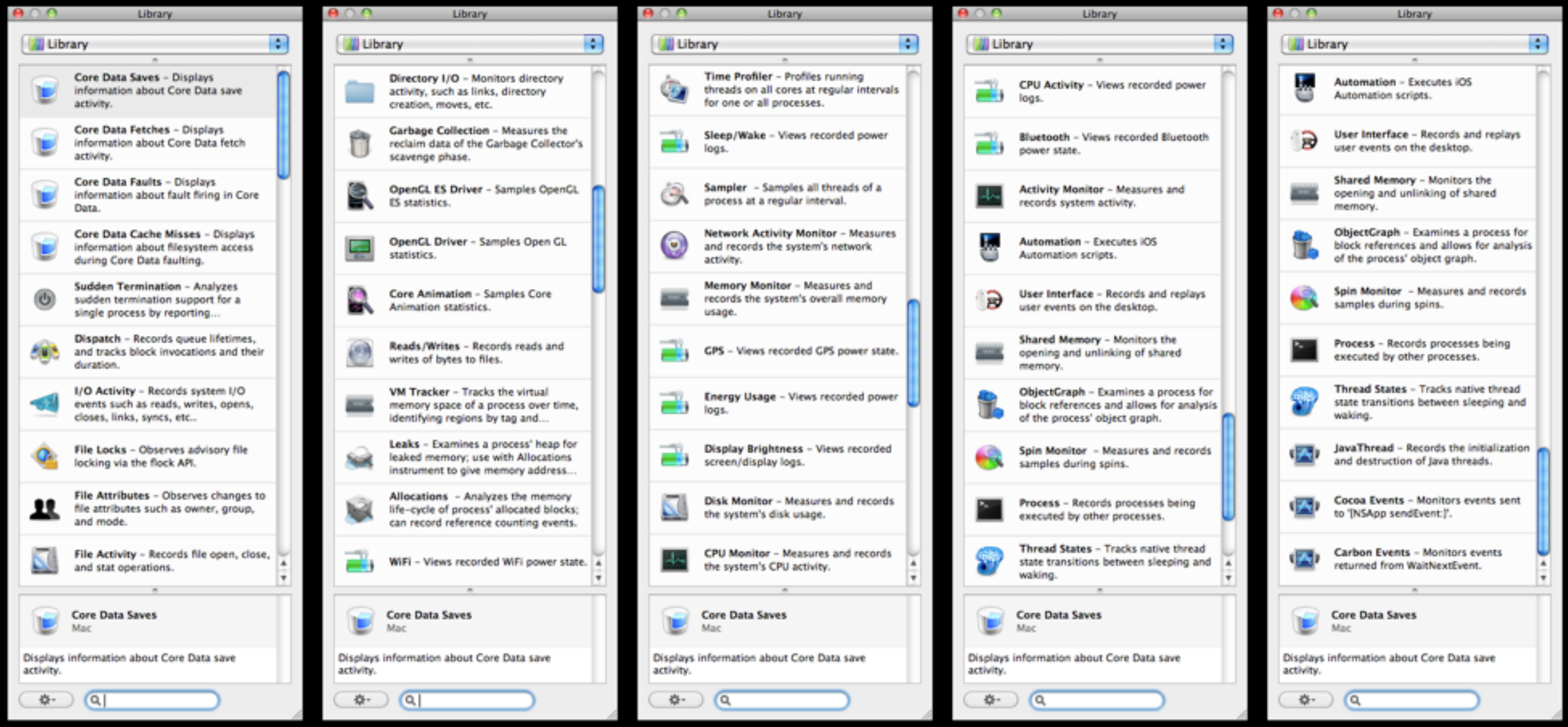


Instruments Odd & Ends

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

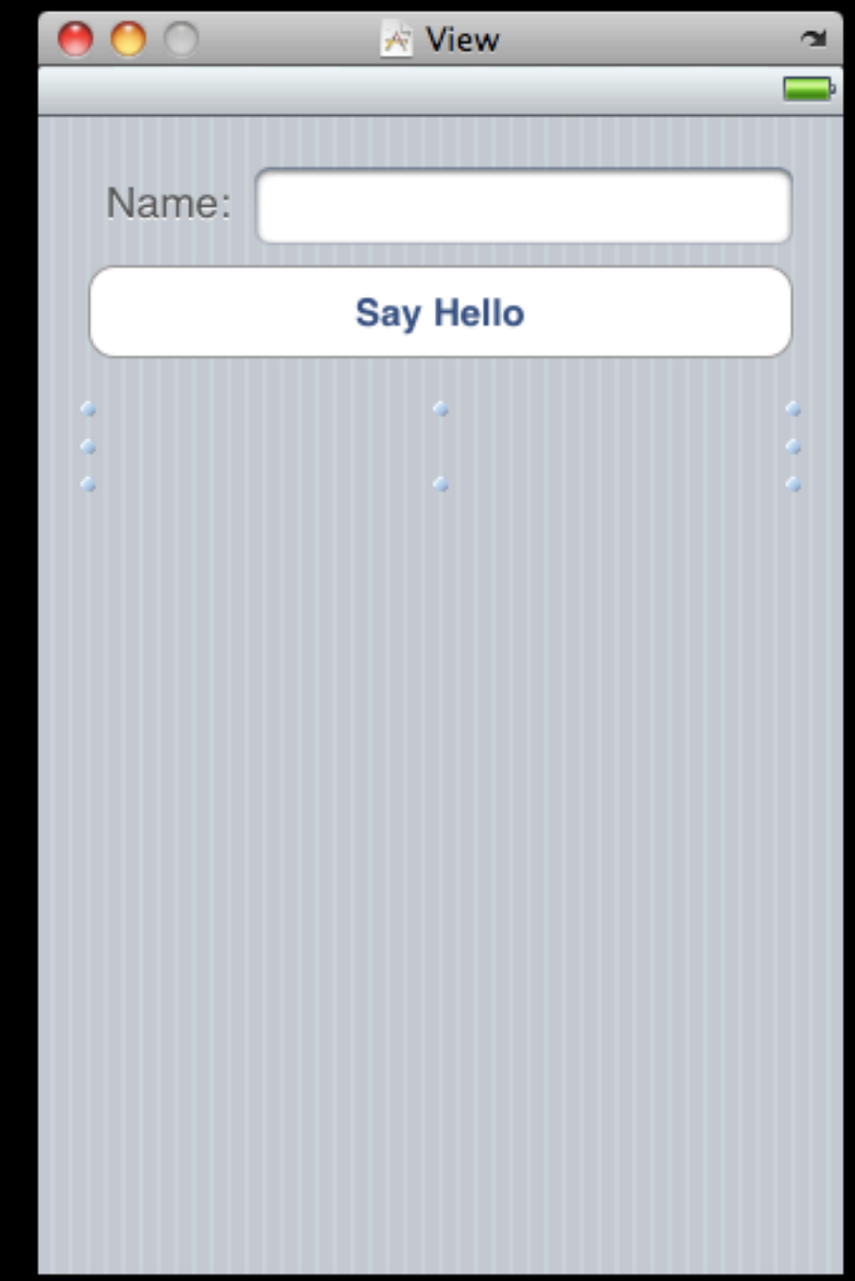
- When zombies are enabled instead of a deallocated object actually being freed, it's class is changed to _NSZombie
- Useful for determining causes of over-releases and more generally messages sent to deallocated objects
- Zombies can be enabled via with NSZombieEnabled environment variable

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

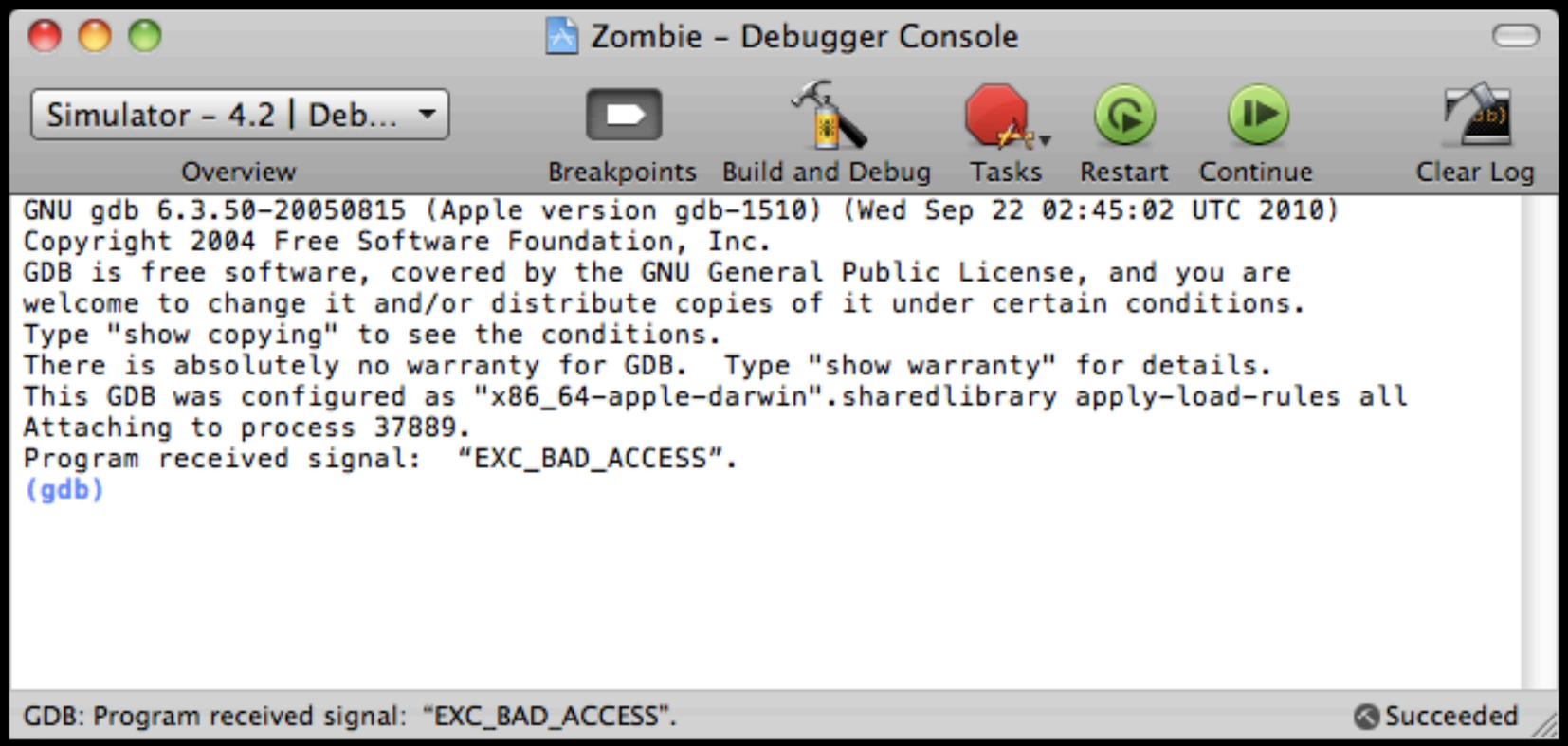
```
#import <UIKit/UIKit.h>

@interface ZombieViewController : UIViewController <UITextFieldDelegate> {

}

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

- (IBAction)sayHello;

@end
```

ZombieViewController.m

```
#import "ZombieViewController.h"

@implementation ZombieViewController

@synthesize nameField, greetingLabel;

- (BOOL)textFieldShouldReturn:(UITextField *)textField {
  [self.nameField resignFirstResponder];
  [self sayHello];
  return YES;
}

- (IBAction)sayHello {

  NSString *msg = [NSString stringWithFormat:@"Hello %@", self.nameField.text];
  self.greetingLabel.text = msg;

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

/* ... */

@end
```

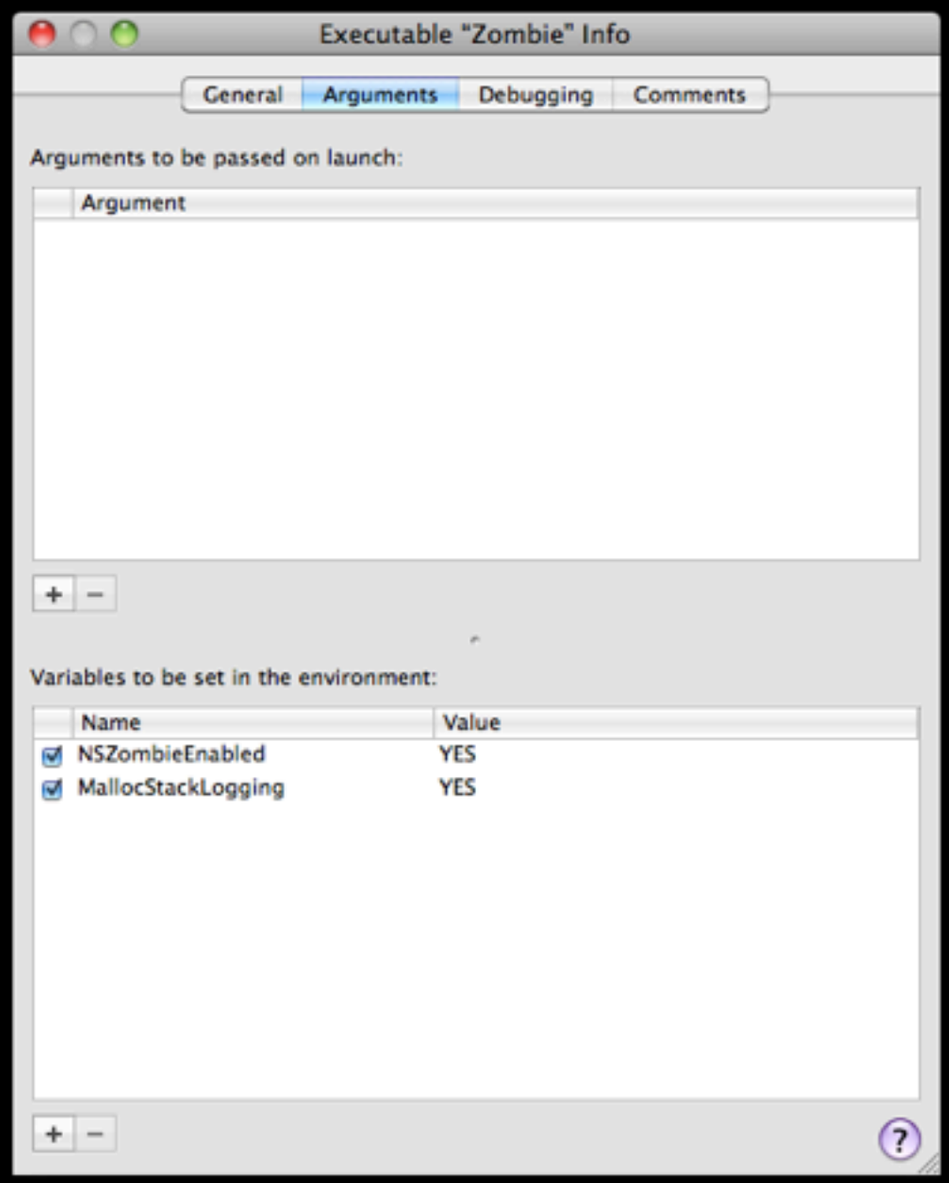
Running the Buggy App

- 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...

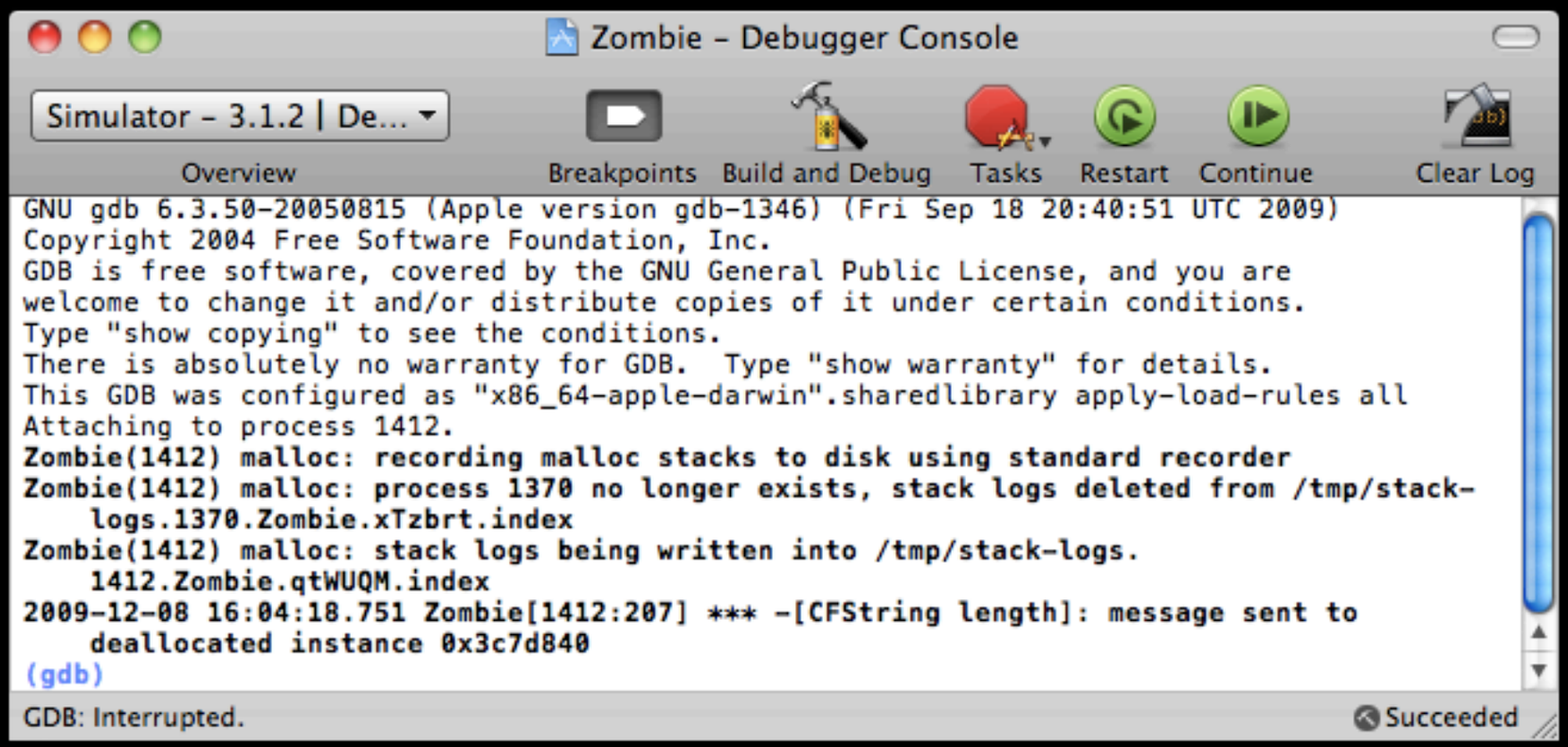


Enabling Zombies

- 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



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 – 4.2 | Debu... | Overview | Breakpoints | Build and Debug | Tasks | Restart | Continue | Clear Log

```
GNU gdb 6.3.50-20050815 (Apple version gdb-1510) (Wed Sep 22 02:45:02 UTC 2010)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "x86_64-apple-darwin".sharedlibrary apply-load-rules all
Attaching to process 37972.
Zombie(37972) malloc: recording malloc stacks to disk using standard recorder
Zombie(37972) malloc: process 37961 no longer exists, stack logs deleted from /tmp/stack-
    logs.37961.Zombie.2Dz8zf.index
Zombie(37972) malloc: stack logs being written into /tmp/stack-logs.
    37972.Zombie.Daou9Z.index
2010-12-07 00:10:43.629 Zombie[37972:207] *** -[CFString length]: message sent to
    deallocated instance 0x4ba6840
(gdb) shell malloc_history 37972 0x4ba6840
malloc_history Report Version:  2.0
Process:         Zombie [37972]
Path:            /Users/Dan/Library/Application Support/iPhone Simulator/4.2/Applications/
    5ACF2A0A-B1B2-442B-B9EF-26D6DE1E352D/Zombie.app/Zombie
Load Address:    0x1000
Identifier:      Zombie
Version:         ??? (???)
Code Type:       X86 (Native)
Parent Process:  gdb-i386-apple-darwin [37973]

Date/Time:       2010-12-07 00:12:03.202 -0500
OS Version:      Mac OS X 10.6.5 (10H574)
Report Version:  6

ALLOC 0x4ba6840-0x4ba685f [size=32]: thread_a05b8540 |start | main | UIApplicationMain |
    GSEventRun | GSEventRunModal | CFRunLoopRunInMode | CFRunLoopRunSpecific |
    __CFRunLoopRun | __CFRunLoopDoSource1 |
    __CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE1_PERFORM_FUNCTION__ | PurpleEventCallback |
    _UIApplicationHandleEvent | -[UIApplication sendEvent:] | -[UIApplication
    handleEvent:withNewEvent:] | -[UIKeyboardImpl handleKeyEvent:] | -[UIKeyboardImp
    addInputString:fromVariantKey:] | -[UIKeyboardImpl callShouldInsertText:] | -
    [ZombieViewController textFieldShouldReturn:] | -[ZombieViewController sayHello] | +
    [NSString stringWithFormat:] | __NSPlaceholderString initWithFormat:locale:arguments:]
    | _CFStringCreateWithFormatAndArgumentsAux | CFStringCreateCopy |
    __CFStringCreateImmutableFunnel3 | _CFRuntimeCreateInstance | malloc_zone_malloc

Binary Images:
    0x1000 -     0x2ffb +Zombie ??? (???) <7D1E1664-5A7E-5E92-222B-5F3DA0665A81> /Users/
        Dan/Library/Application Support/iPhone Simulator/4.2/Applications/5ACF2A0A-
        B1B2-4428-B9EF-26D6DE1E352D/Zombie.app/Zombie
    0x6000 -   0x1c7fe7 +Foundation 751.49.0 (compatibility 300.0.0) <DB9A4461-C768-9B7B-
        E463-4568E3FAA179> /Developer-4.2.1/Platforms/iPhoneSimulator.platform/Developer/
        SDKs/iPhoneSimulator4.2.sdk/System/Library/Frameworks/Foundation.framework/
        Foundation
  0x2a4000 -   0x7b0ff3 +UIKit 1400.0.0 (compatibility 1.0.0) <EE783C53-A647-D7F8-62CF-
      FB3F7DD16C54> /Developer-4.2.1/Platforms/iPhoneSimulator.platform/Developer/SDKs/
      iPhoneSimulator4.2.sdk/System/Library/Frameworks/UIKit.framework/UIKit
  0x9f5000 -   0xc28ff7  com.apple.CoreGraphics 1.600.0 (???) <78926D2F-9A6C-8B48-
      BD99-72B3373872BD> /Developer-4.2.1/Platforms/iPhoneSimulator.platform/Developer/
      SDKs/iPhoneSimulator4.2.sdk/System/Library/Frameworks/CoreGraphics.framework/
      CoreGraphics
  0xc91000 -   0xca2ff7 +libSystem.dylib 125.0.0 (compatibility 1.0.0) <76CF85EC-AAC8-
```

GDB: Interrupted. Succeeded

Where it was created

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/

For Next Class

- Safari Web Content Guide
  - http://developer.apple.com/library/safari/#documentation/AppleApplications/Reference/SafariWebContent/
- Preparing Your Web Content for iPad
  - http://developer.apple.com/library/ios/#technotes/tn2010/tn2262/
- User Experience Coding How-To's for Safari on iPhone
  - http://developer.apple.com/library/safari/#codinghowtos/Mobile/UserExperience/