Up next:
Performance Tuning

Mark Dalrymple
CocoaConf, March 2013

@borkware

http://borkware.com/cocoaconf

What is Performance Tuning?

- Making slow software fast
  - For your definition of “slow” and “fast”
- Knowing it’s slow means you’ve identified the problem and can measure it
  - “Fast” gives you criteria for when to stop
- Actually a huge topic
  - Fundamentally a subset of "Debugging"

When to Optimize

- "We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil"

  — D. Knuth

- “The First Rule of Program Optimization: Don't do it. The Second Rule of Program Optimization (for experts only!): Don't do it yet.”

  — M. Jackson

How to Optimize?

- Discover what's slow
- Figure out why it's slow
- Fix it

We Got Tools

measure, measure, measure!

- Measure first. Establish a baseline
- Fix
- Measure again so you don't regress

measure first before changing stuff!

Things to Keep in Mind

- Forget any assumptions about where problems are
- Be consistent with your test data
- The Simulator has vastly different performance characteristics than the Device
- Throw large data sets at your program often

Orders of Magnitude

Kind of Product	**# of records**	**Data Structure**
Short-term Weight Tracker™	1 - 10	ivars / C array
Personal WeighMonster™	10 - 1,000	NSArray
Pittsburgh Fitness Weigh-Yinz™	1,000 - 1,000,000	Core Data / sqllite db
LA Fatness Übertrack™	1,000,000 - 100,000,000	Database Server(s) / Amazon services
Google WeighIn™ (Beta)	100,000,000 - 100,000,000,000	Distributed db cluster / BigTable / Data Centers

Numbers of Interest: Jeff Dean

L1 cache reference	0.5 ns	
Branch Mispredict	5 ns	
L2 cache reference	7 ns	
mutex lock/unlock	100 ns	
Main memory reference	100 ns	
Compress 1K bytes with Zippy	10,000 ns	
Send 2k bytes over 1 Gbps network	20,000 ns	
Read 1 MB sequentially from memory	250,000 ns	0.25 ms
Round trip in datacenter	500,000 ns	0.5 ms
Disk seek	10,000,000 ns	10 ms
Read 1 MB sequentially from network	10,000,000 ns	10 ms
Read 1 MB sequentially from disk	30,000,000 ns	30 ms
Send Packet CA->Netherlands->CA	150,000,000 ns	150 ms

http://bit.ly/jeff-dean-numbers

Numbers of Interest: Mike Ash

	Mac Pro 10.5	iPhone 4
IMP-cached message send	0.7 ns	18 ns
C++ virtual function call	1.1 ns	17 ns
Integer Division	2.4 ns	71 ns
Objective-C message send	4.9 ns	54 ns
Floating-point division	9.2 ns	101 ns
16 byte memcpy	2.9 ns	34 ns
16 byte malloc/free	56 ns	559 ns
NSInvocation message send	77 ns	619 ns
NSObject alloc/init/release	290 ns	4,825 ns
NSAutoreleasePool alloc/init/release	357 ns	1,315 ns
16MB malloc/free	4,485 ns	12,736 ns
Read 16 byte file	21,219 ns	187,450 ns
zero-second delayed perform	42,211 ns	231,307 ns
pthread create/join	56,633 ns	160,274 ns
Write 16 byte file	492,040 ns	1,053,244 ns
NSTask process spawn	6,096,478 ns	N/A
Read 16MB file	28,619,582 ns	188,647 ns
Write 16MB file	361,767,087 ns	667,922 ns

Timing in Code

```
kern_return_t mach_timebase_info (mach_timebase_info_t info);

uint64_t mach_absolute_time (void);

struct mach_timebase_info {
    uint32_t    numer;
    uint32_t    denom;
};
```

Timing in Code

```
CGFloat BNRTimeBlock (void (^block)(void)) {
    mach_timebase_info_data_t info;
    if (mach_timebase_info(&info) != KERN_SUCCESS) return -1.0;

    uint64_t start = mach_absolute_time ();

    block ();

    uint64_t end = mach_absolute_time ();
    uint64_t elapsed = end - start;

    uint64_t nanos = elapsed * info.numer / info.denom;
    return (CGFloat)nanos / NSEC_PER_SEC;

} // BNRTimeBlock
```

Big Nerd Ranch Weblog: A Timing Utility

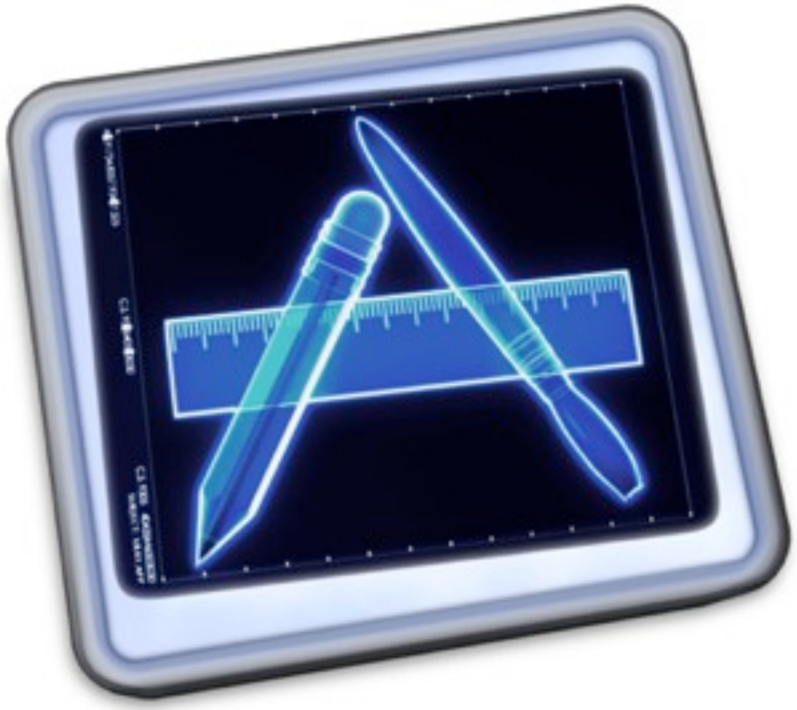
Timing in Code

```
CGFloat time = BNRTimeBlock (^{
        for (NSString *line in split) {
            if ([line hasPrefix: @"#"]) continue;

            BWThingie *thingie = [BWThingie thingieWithString: line];

            if (thingie == nil) continue;

            [_thingies addObject: thingie];
        }
    });

NSLog (@"time it took: %f", time);
```

Timing in The Shell

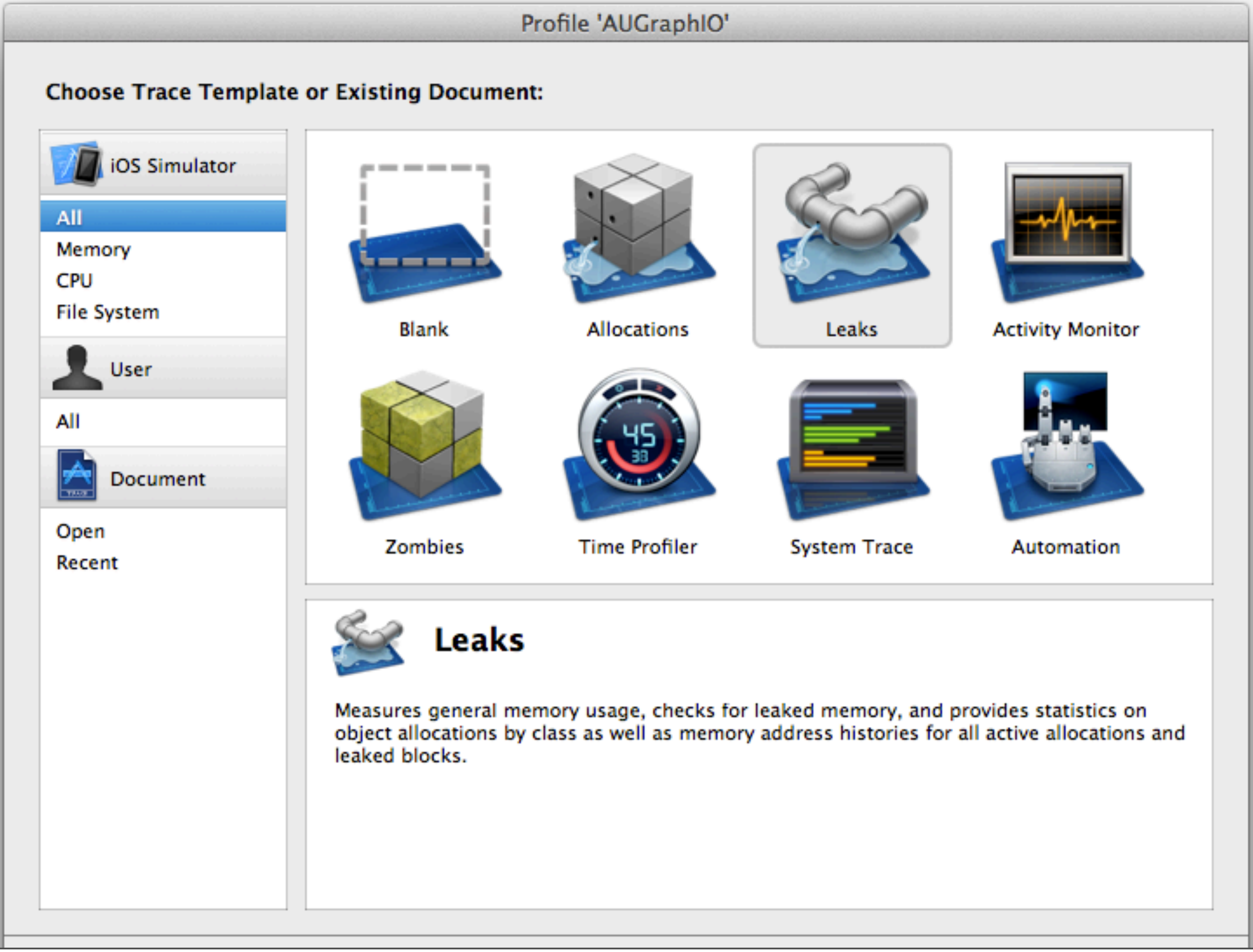
```
% time build/BigShow.app/Contents/MacOS/BigShow
3.300u 0.800s 1:44.78 3.9%      0+0k 0+22io 0pf+0w
```

Instruments



- Apple’s Toy Chest
  - er, Profiling Suite

Instrument Templates

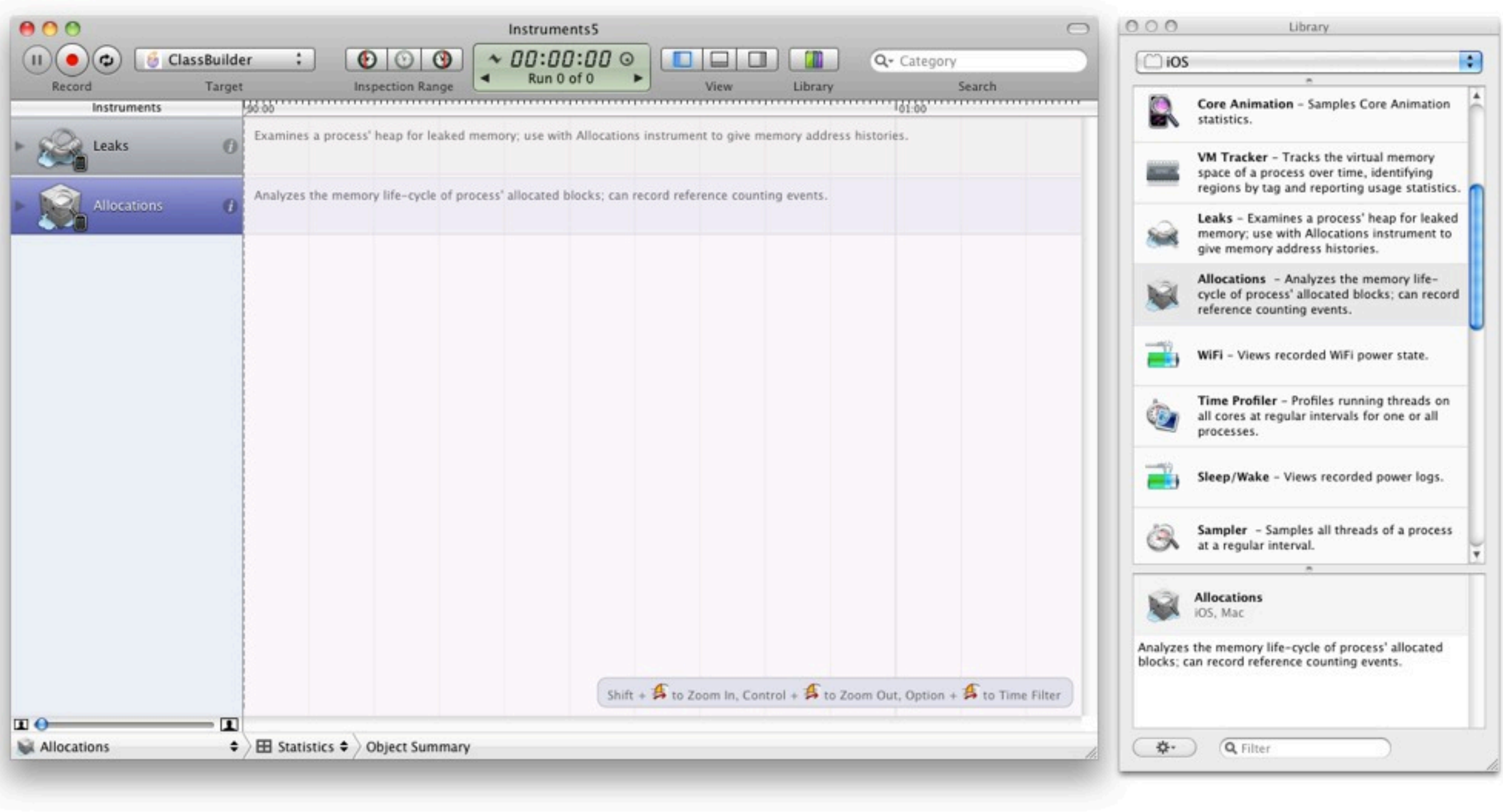


Choose Trace Template or Existing Document:

Leaks

Measures general memory usage, checks for leaked memory, and provides statistics on object allocations by class as well as memory address histories for all active allocations and leaked blocks.

Instruments and Library



Examines a process' heap for leaked memory; use with Allocations instrument to give memory address histories.

Analyzes the memory life-cycle of process' allocated blocks; can record reference counting events.

Core Animation – Samples Core Animation statistics.

VM Tracker – Tracks the virtual memory space of a process over time, identifying regions by tag and reporting usage statistics.

Leaks – Examines a process' heap for leaked memory; use with Allocations instrument to give memory address histories.

Allocations – Analyzes the memory life-cycle of process' allocated blocks; can record reference counting events.

WiFi – Views recorded WiFi power state.

Time Profiler – Profiles running threads on all cores at regular intervals for one or all processes.

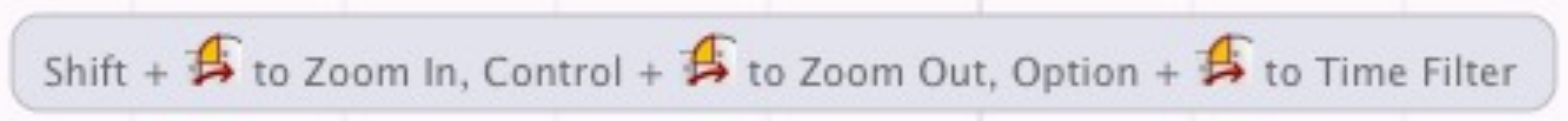
Sleep/Wake – Views recorded power logs.

Sampler – Samples all threads of a process at a regular interval.

Allocations
iOS, Mac

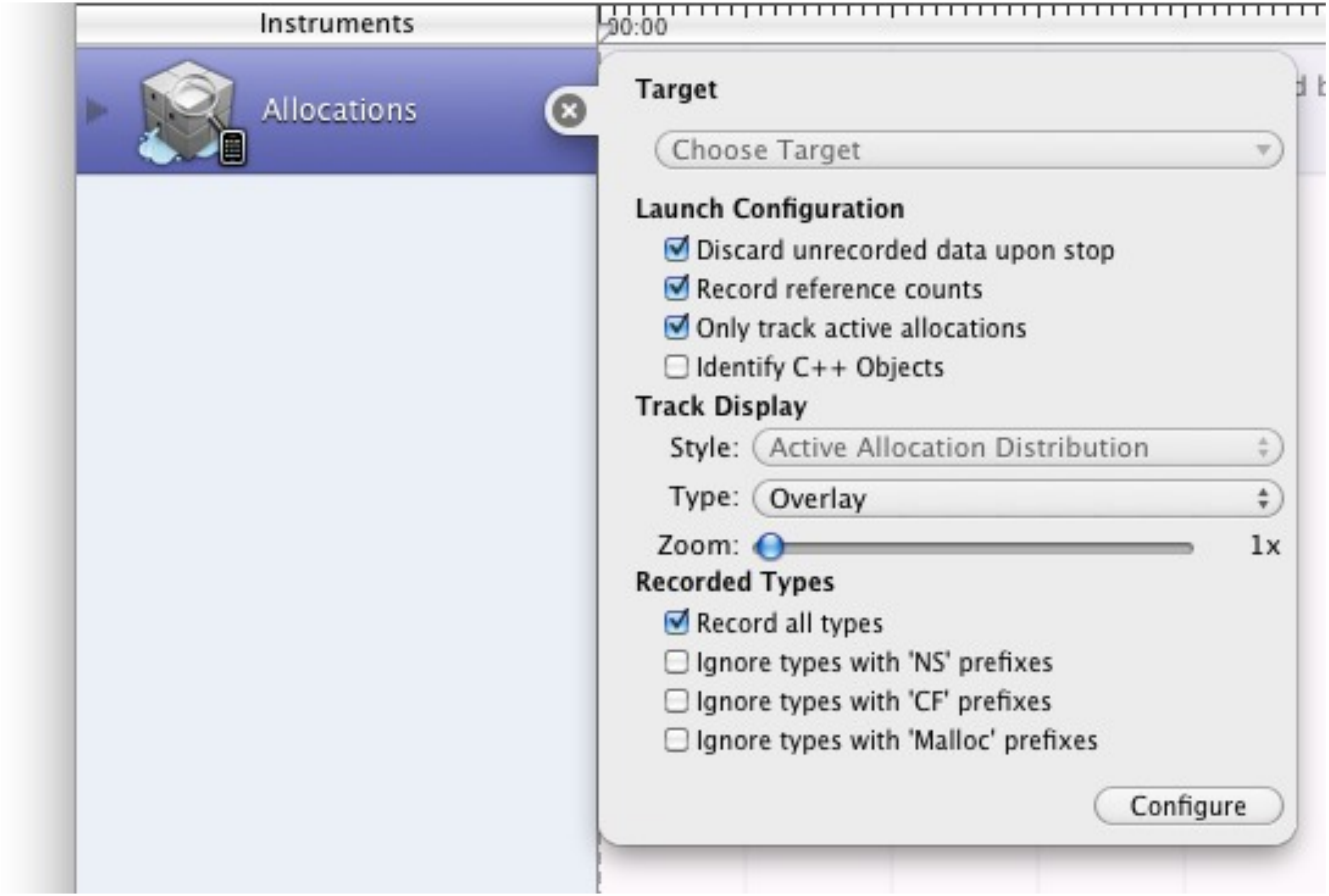
Analyzes the memory life-cycle of process' allocated blocks; can record reference counting events.

Penguin Profile



Shift + to Zoom In, Control + to Zoom Out, Option + to Time Filter

Look Under the Rocks



Target

Choose Target

Launch Configuration

☑ Discard unrecorded data upon stop
☑ Record reference counts
☑ Only track active allocations
□ Identify C++ Objects

Track Display

Style: Active Allocation Distribution
Type: Overlay
Zoom: 1x

Recorded Types

☑ Record all types
□ Ignore types with 'NS' prefixes
□ Ignore types with 'CF' prefixes
□ Ignore types with 'Malloc' prefixes

Aside: Tracking Down Problems

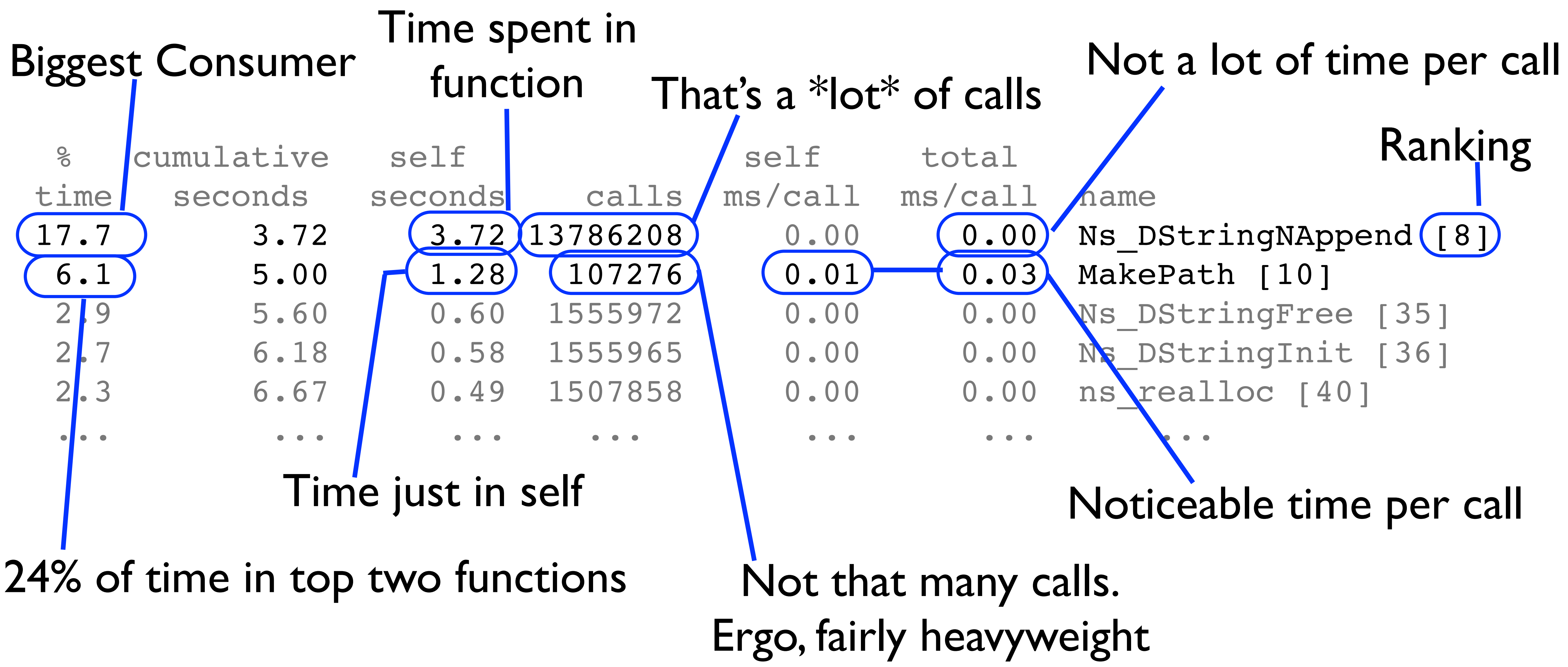
gprof

Kicking it Old School!

- Not a GNU program
- compile with -pg, run program
- gprof gmon.out > profile.txt

http://bit.ly/cocoaconf-gprof
"how to read gprof output"

The Flat Profile



Biggest Consumer

Time spent in function

That's a *lot* of calls

Not a lot of time per call

Ranking

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 17.7      3.72     3.72  13786208     0.00     0.00  Ns_DStringNAppend [8]
  6.1      5.00     1.28    107276     0.01     0.03  MakePath [10]
  2.9      5.60     0.60   1555972     0.00     0.00  Ns_DStringFree [35]
  2.7      6.18     0.58   1555965     0.00     0.00  Ns_DStringInit [36]
  2.3      6.67     0.49   1507858     0.00     0.00  ns_realloc [40]
  ...       ...      ...       ...        ...      ...      ...
```

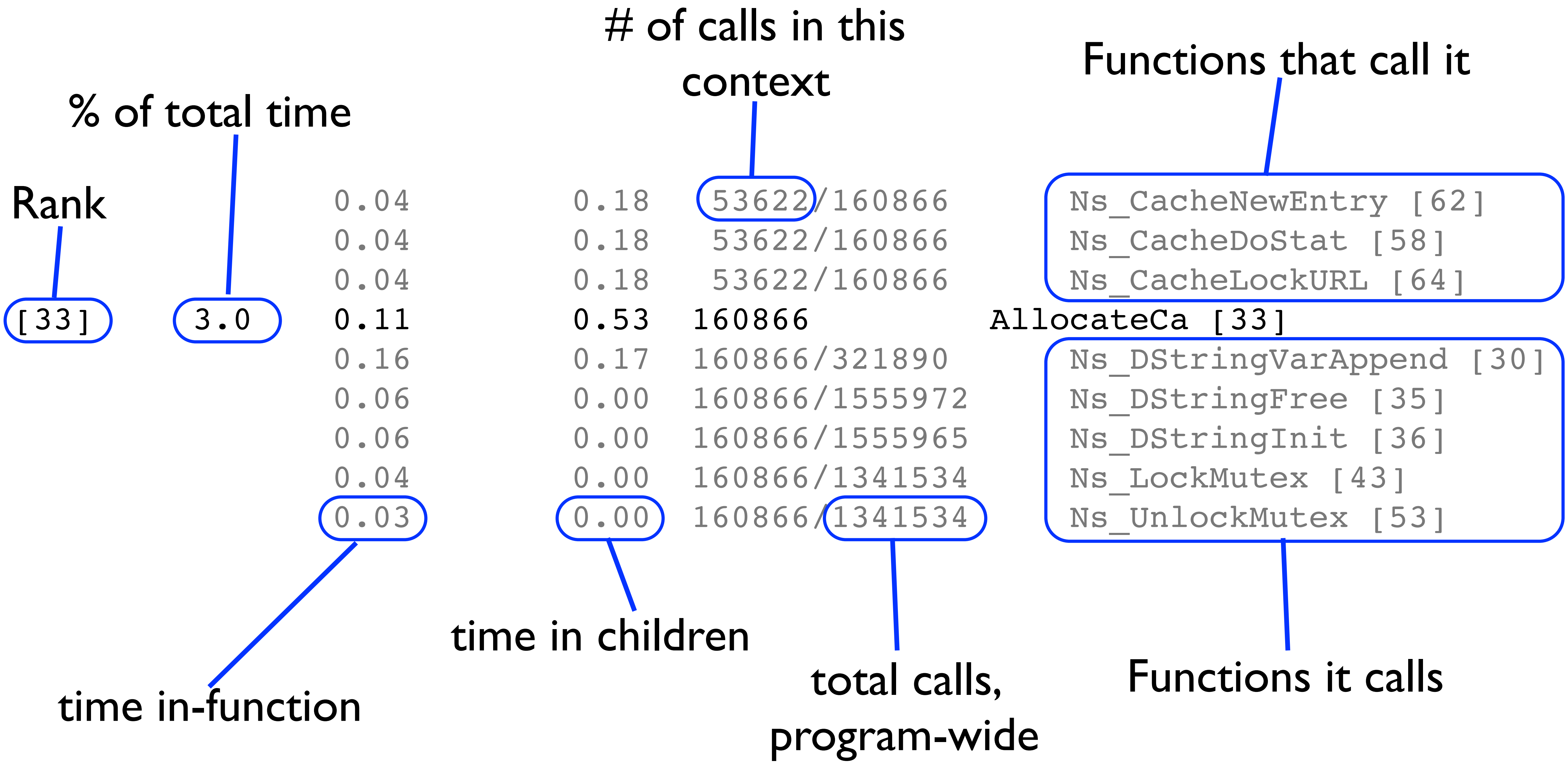
Time just in self

Noticeable time per call

24% of time in top two functions

Not that many calls.
Ergo, fairly heavyweight

The Call Graph



# of calls in this context

Functions that call it

% of total time

Rank

```
                0.04        0.18       53622/160866      Ns_CacheNewEntry [62]
                0.04        0.18       53622/160866      Ns_CacheDoStat [58]
                0.04        0.18       53622/160866      Ns_CacheLockURL [64]
[33]     3.0    0.11        0.53  160866               AllocateCa [33]
                0.16        0.17  160866/321890          Ns_DStringVarAppend [30]
                0.06        0.00  160866/1555972         Ns_DStringFree [35]
                0.06        0.00  160866/1555965         Ns_DStringInit [36]
                0.04        0.00  160866/1341534         Ns_LockMutex [43]
                0.03        0.00  160866/1341534         Ns_UnlockMutex [53]
```

time in children

total calls, program-wide

Functions it calls

time in-function

Stochastic Profiling

- Using the Debugger for profiling

Beware Convenience

```
TString *timestamp =
    month + “/” + day + “/” + year + “ “
    + hours + “:” + minutes + “:”
    + seconds;
```

DTrace

```
syscall::read:entry
{
    self->ts = timestamp;
}
```

```
syscall::read:return
/self->ts/
{
    delta = timestamp - self->ts;
    @averagetime[execname] = avg(delta);
    @callcount[execname] = count();
    @mintime[execname] = min(delta);
    @maxtime[execname] = max(delta);
    self->ts = 0;
}
```

```
call count
      mDNSResponder 2
         emacs-i386 3
           Terminal 4
                mds 6
          fseventsd 9
       VoodooPad Pro 270
             Safari 622
```

```
average time
           Terminal 7941
         emacs-i386 9985
      mDNSResponder 17781
             Safari 24666
      VoodooPad Pro 55339
          fseventsd 527863979
                mds 551164939
```

The Point

- How much time was spent in X
  - Who called it, how often, how much time
- Whom does X call
  - How often, how much time

For Example...

- I spent 10 seconds runtime, 10% of my app in drawing a ride profile.
- I drew it 300 times in the space of four minutes from the ride screen.
  (That's a reasonable number given the app)
- Oh look, it called UIImage initWithFoobage 300 times. For the same image. I can cache that"

For Example...

- I spent 50 seconds runtime in Core Graphics out of 3 minutes of application run time.
- I called a bunch of functions, all of which bottlenecked down to ConvertCYMKToRGB, spending most of the time in that utility function.
- I can pre-convert those images at build time to avoid this work at run time"

So, what can be slow?

Basically, everything

- CPU
- Memory
- Disk / File System
- Network
- Power
- Graphics

So, what can be slow?

Basically, everything

- CPU *
- Memory *
- Disk / File System
- Network
- Power
- Graphics

CPU

CPU

- Processors are pegged and fans are revving
- Usually means you’re doing too much work
    - Bad algorithm
    - Wrong Data Structure
    - Over-eager processing
- Spread the work over more cores

The Free Lunch Is Over

- It's been over for awhile
- Moore's Law Continues
- Concurrency is Now!
- Optimization and Performance Tuning is important again

Algorithms and Data Structures Are Important

- Be aware of the computational complexity of the tools you use.
  - Some are documented
  - Some you can infer
  - Some you can determine experimentally

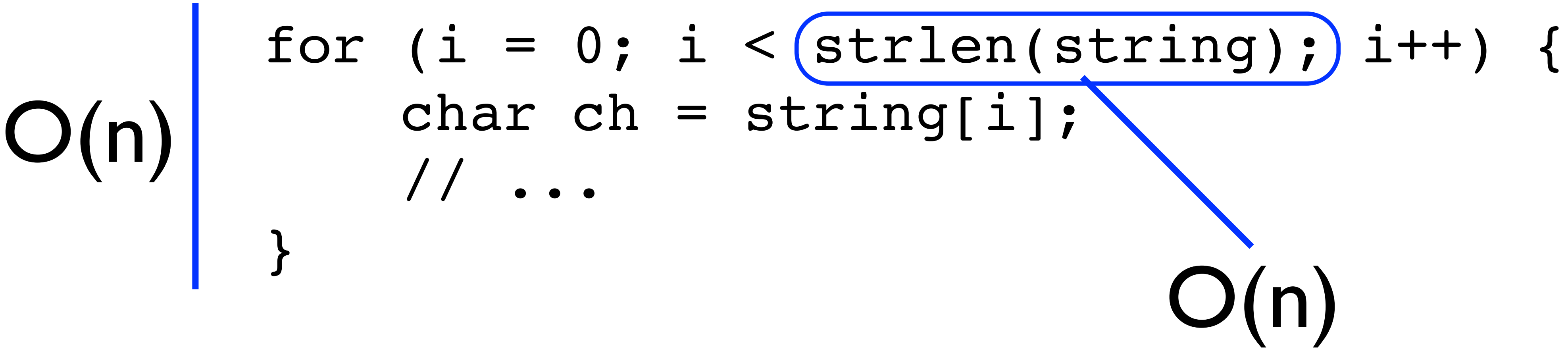
Orders of Complexity

			Processing 1000 items
O(1)	Constant	Indexing a C array Hash table lookup	1
O(log n)	Logarithmic	Binary search Search in balanced tree	10
O(n)	Linear	Search in linked list Inserting into C array	1,000
O(n log n)	n log n	Most sorts	10,000
$O(n^2)$	Quadratic	Bubble sort	1,000,000
$O(c^n)$	Exponential	Recursive Fibonacci	$1.07x10^{301}$
O(n!)	Factorial	Brute-Force Traveling Salesman	$4.02x10^{2567}$

What’s wrong with this?

```
for (i = 0; i < strlen(string); i++) {
    char ch = string[i];
    // ...
}
```

What’s wrong with this?



O(n)

```
for (i = 0; i < strlen(string); i++) {
    char ch = string[i];
    // ...
}
```

O(n)

O(n) done O(n) times == O(n^2)

Check the Headers

CFArray.h

"The access time for a value in the array is guaranteed to be at worst O(lg n) for any implementation, current and future, but will often be O(1) (constant time).

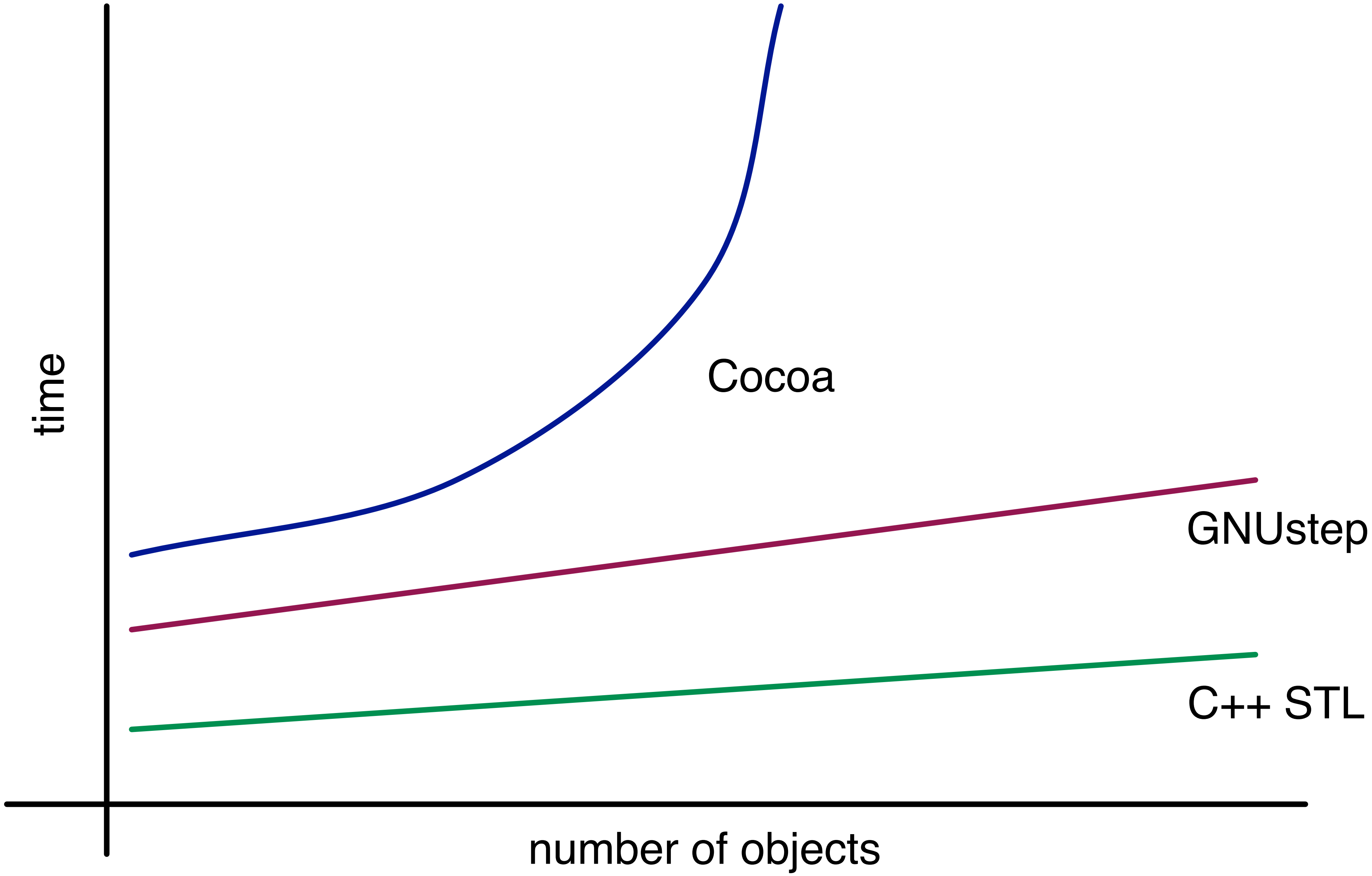
Linear search operations similarly have a worst case complexity of O(n log n), though typically the bounds will be tighter, and so on"

Collection Meltdown

- “MarkD, why does Cocoa Suck So Much?”
- “I’m putting 4 million objects into an NSArray, and it never finishes processing”

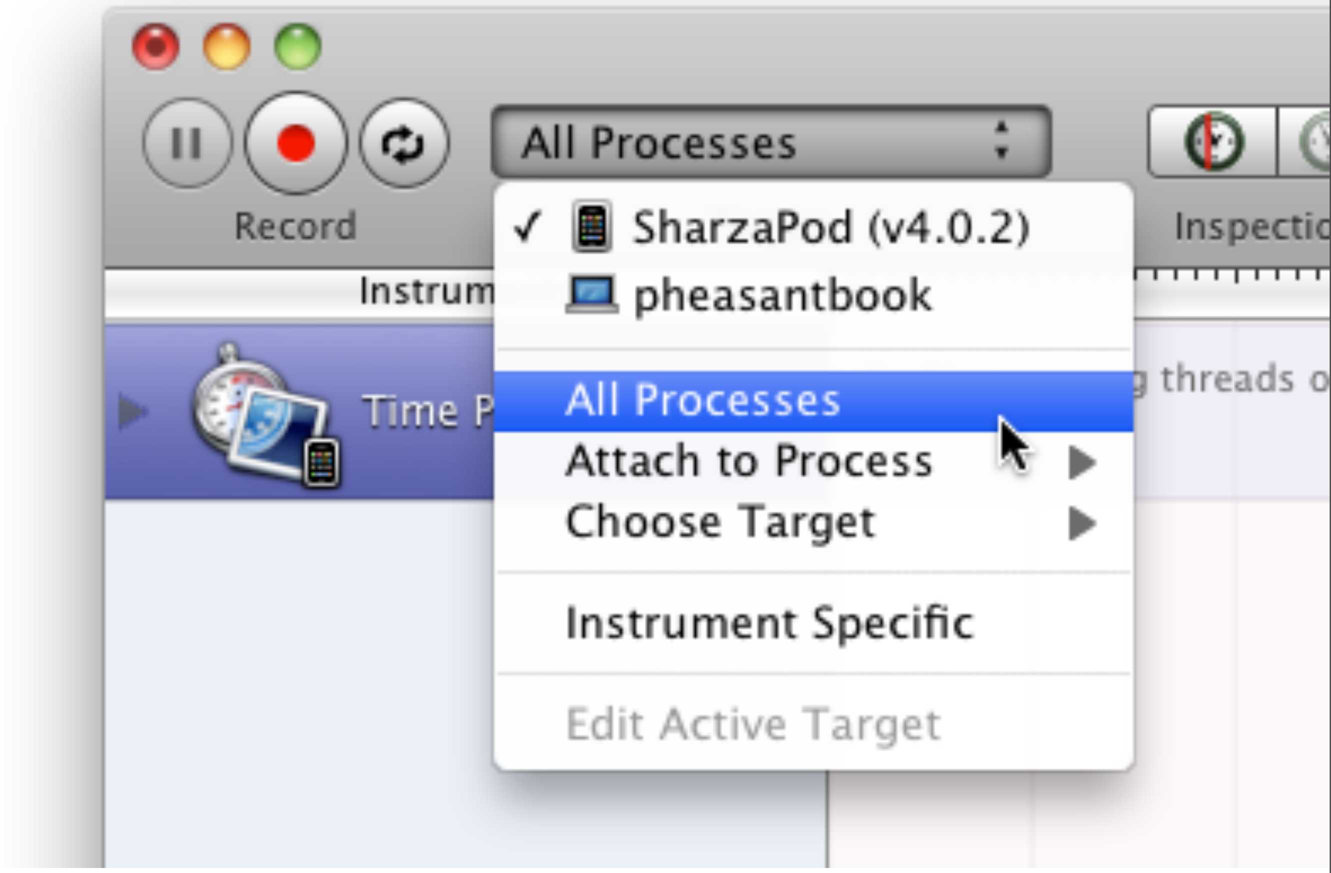
Run Time

4 meeeeeelion objects



Time Profiler Instrument

- gprof on steroids
- Can target one process
- Or the system as a whole
  - even the phone



Demo

- See where stuff is chewing a lot of CPU time

(Business Monitor)

Memory

Memory

- Memory is the New I/O

The G5 can do 16 to 50 vector adds in the time it takes to load a cache line (a sequence of bytes) from memory

Vector code that converts unsigned char data to float and then applies a 9th order polynomial to it is still marginally faster than hand-tuned scalar code that does a lookup into a 256 entry lookup table containing floats.

Locality Of Reference

- How close memory operations are to each other
- Sequential operations are most efficient
- Cache Lines
- Hard to control without work

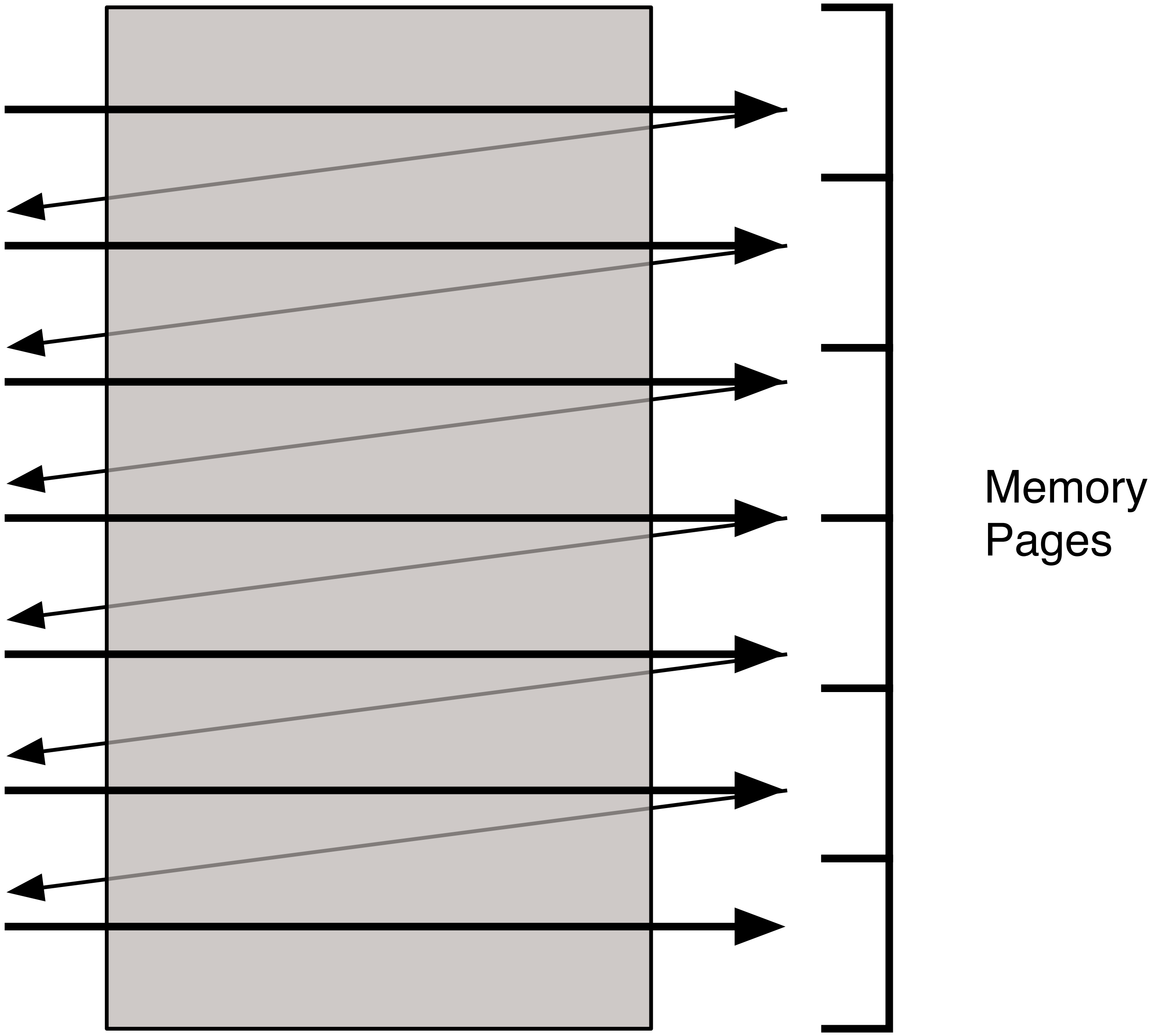
Locality Of Reference

```
#define ARRAYSIZE (10000)
int a[ARRAYSIZE][ARRAYSIZE]; // make a huge array

    for (i = 0; i < ARRAYSIZE; i++){
        for(j = 0; j < ARRAYSIZE; j++){
            a[i][j] = 1;
        }
    }

    for (i = 0; i < ARRAYSIZE; i++){
        for(j = 0; j < ARRAYSIZE; j++){
            a[j][i] = 1;
        }
    }
```

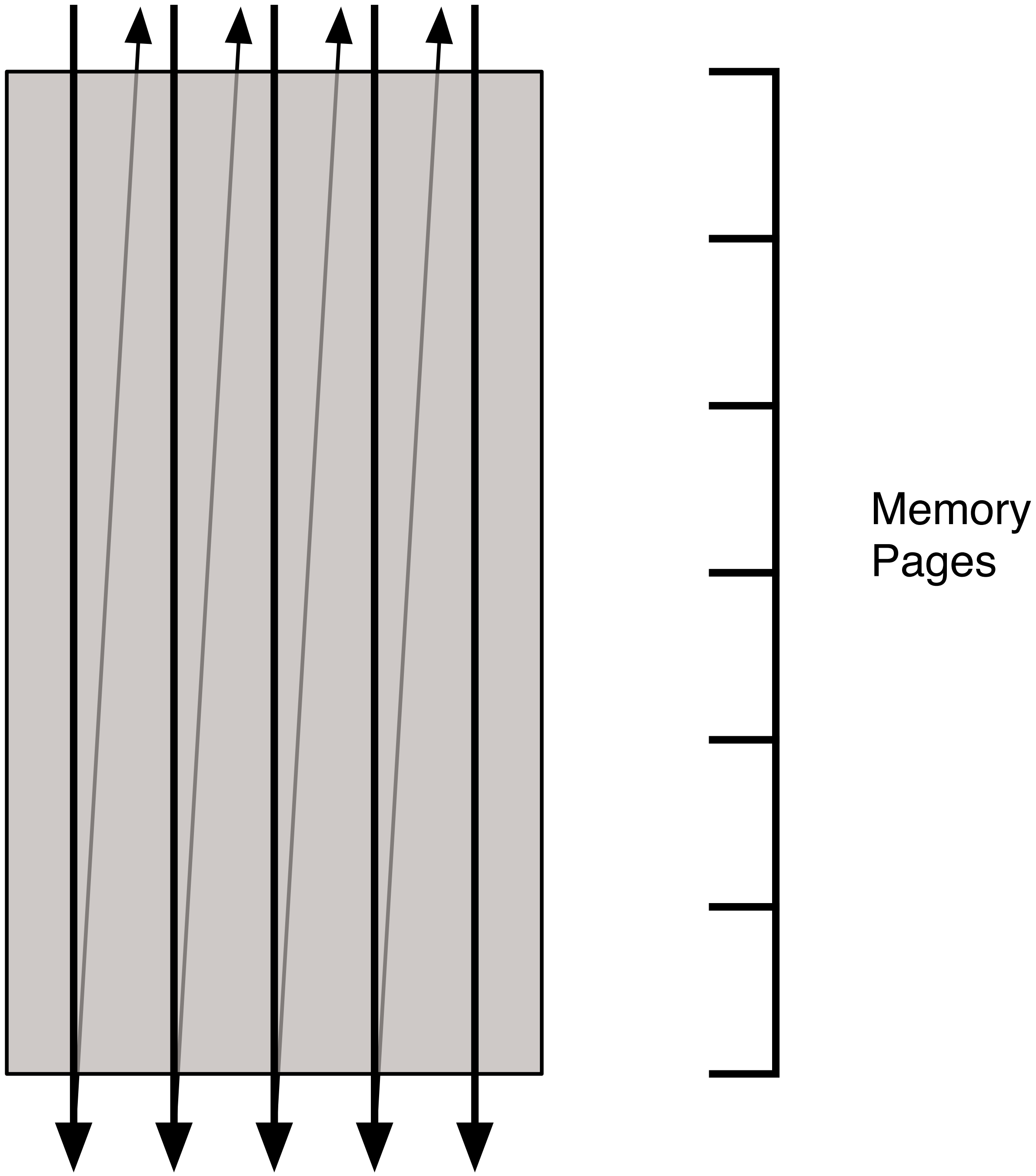
```
% ./locality
100000000 i,j operations in 21 seconds.
100000000 j,i operations in 106 seconds.
```

Holy quintuplets, Batman!

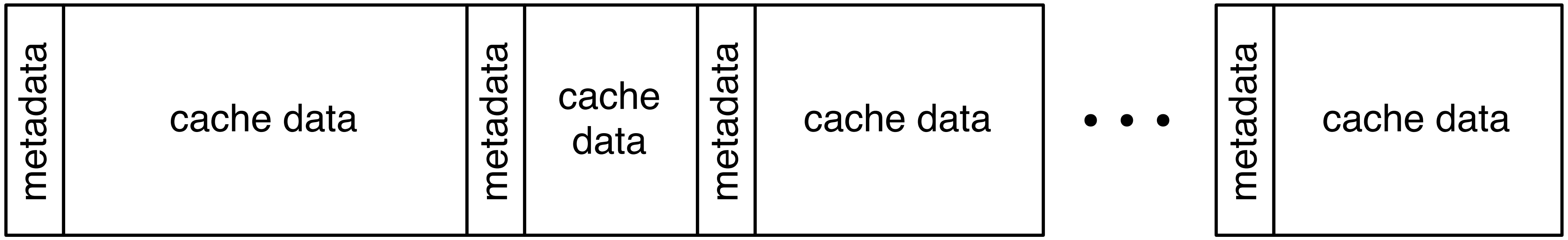
Good Access Pattern



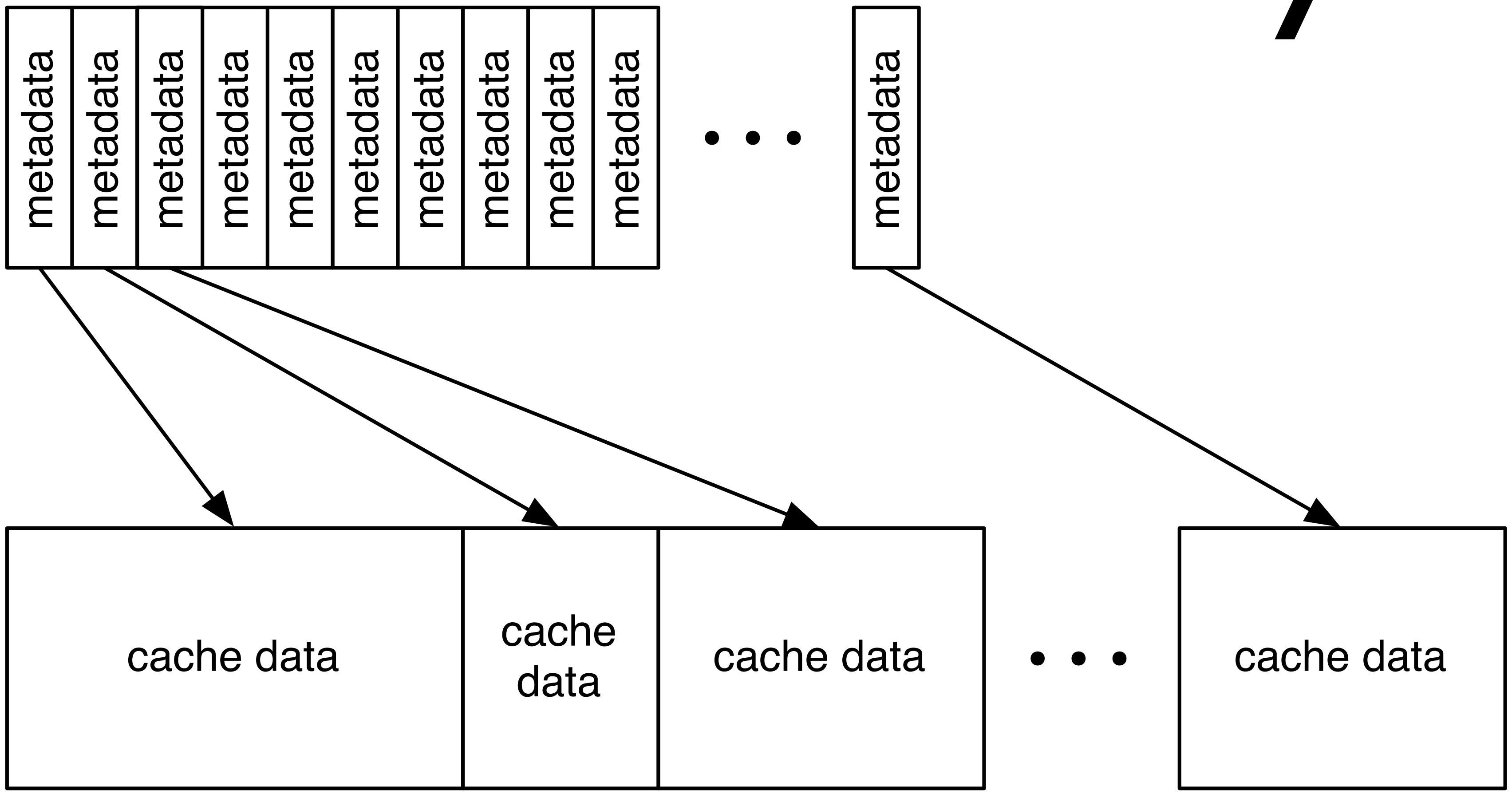
Bad Access Pattern



Bad Locality



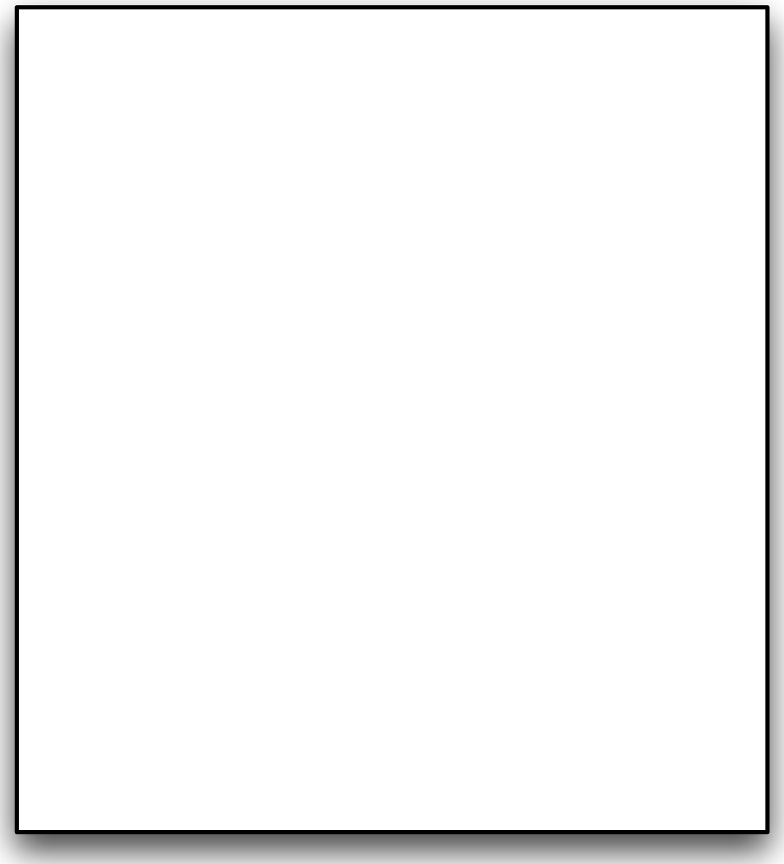
Good Locality



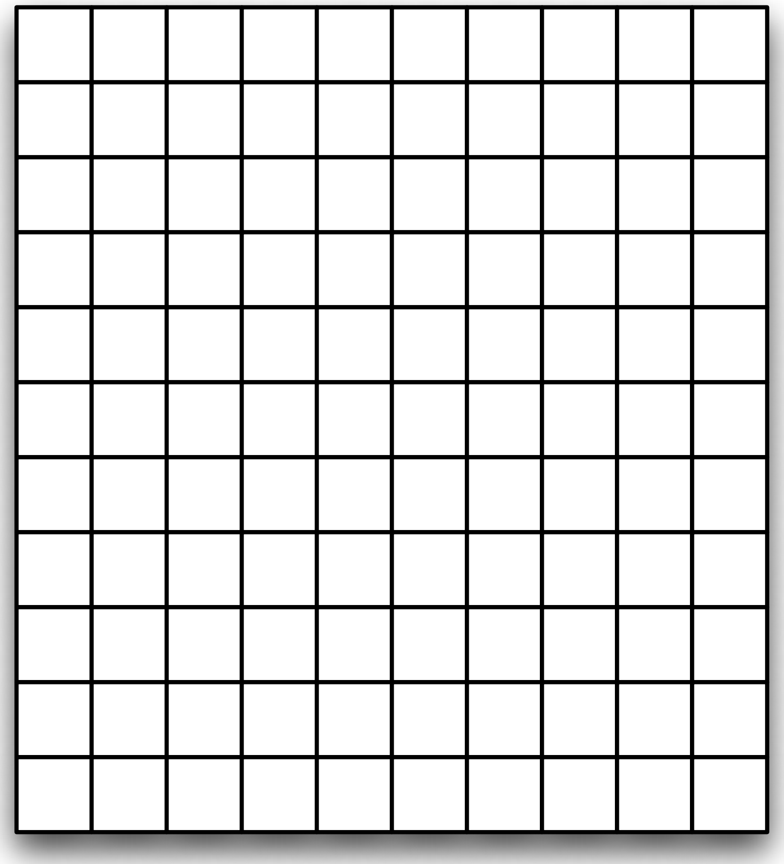
Dynamic Memory is expensive

- malloc
- +alloc
- operator new

Block suballocation



base = malloc(100 * sizeof(node))



subdivided into nodes

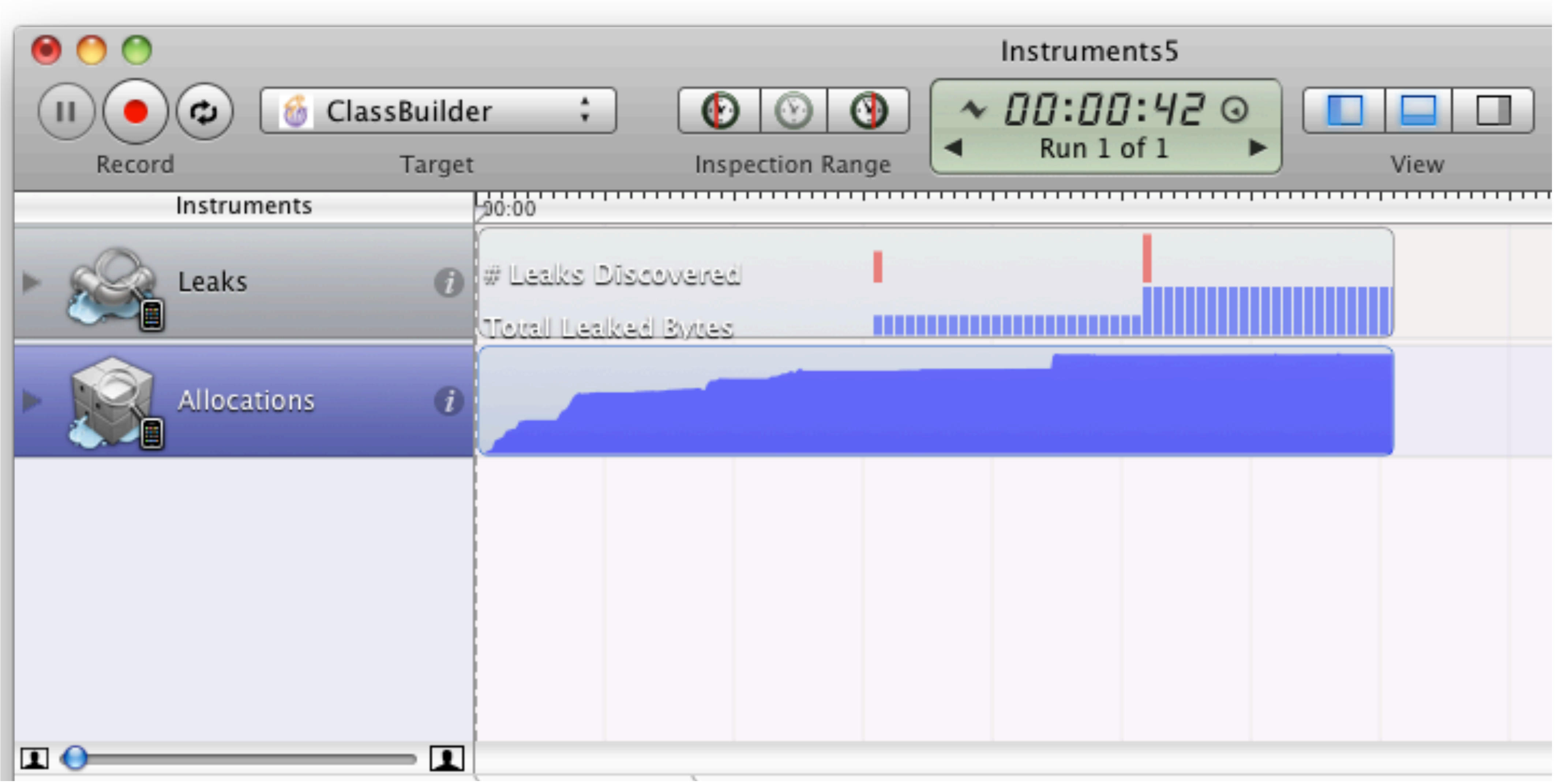
address of node x at

```
base + (x * sizeof(node))
```

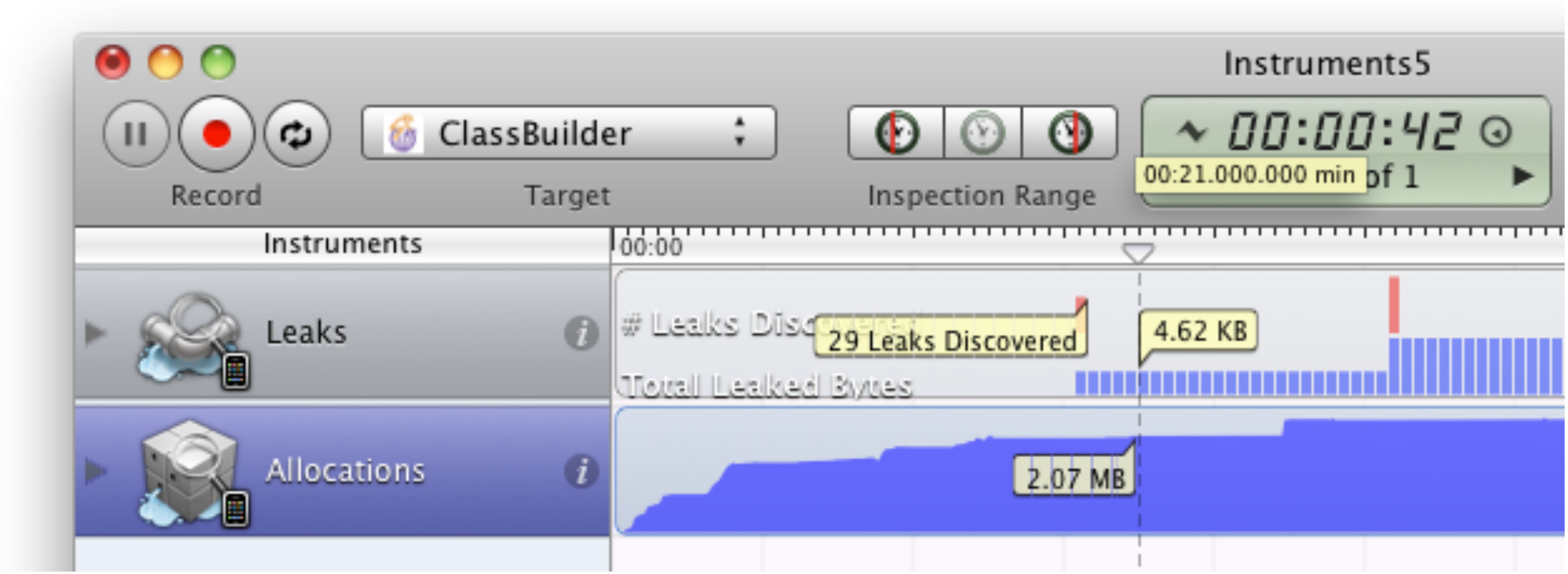
Oh, and fix your leaks

- Leaked / Abandoned memory not usually an immediate performance issue
- Can bite you if memory builds up

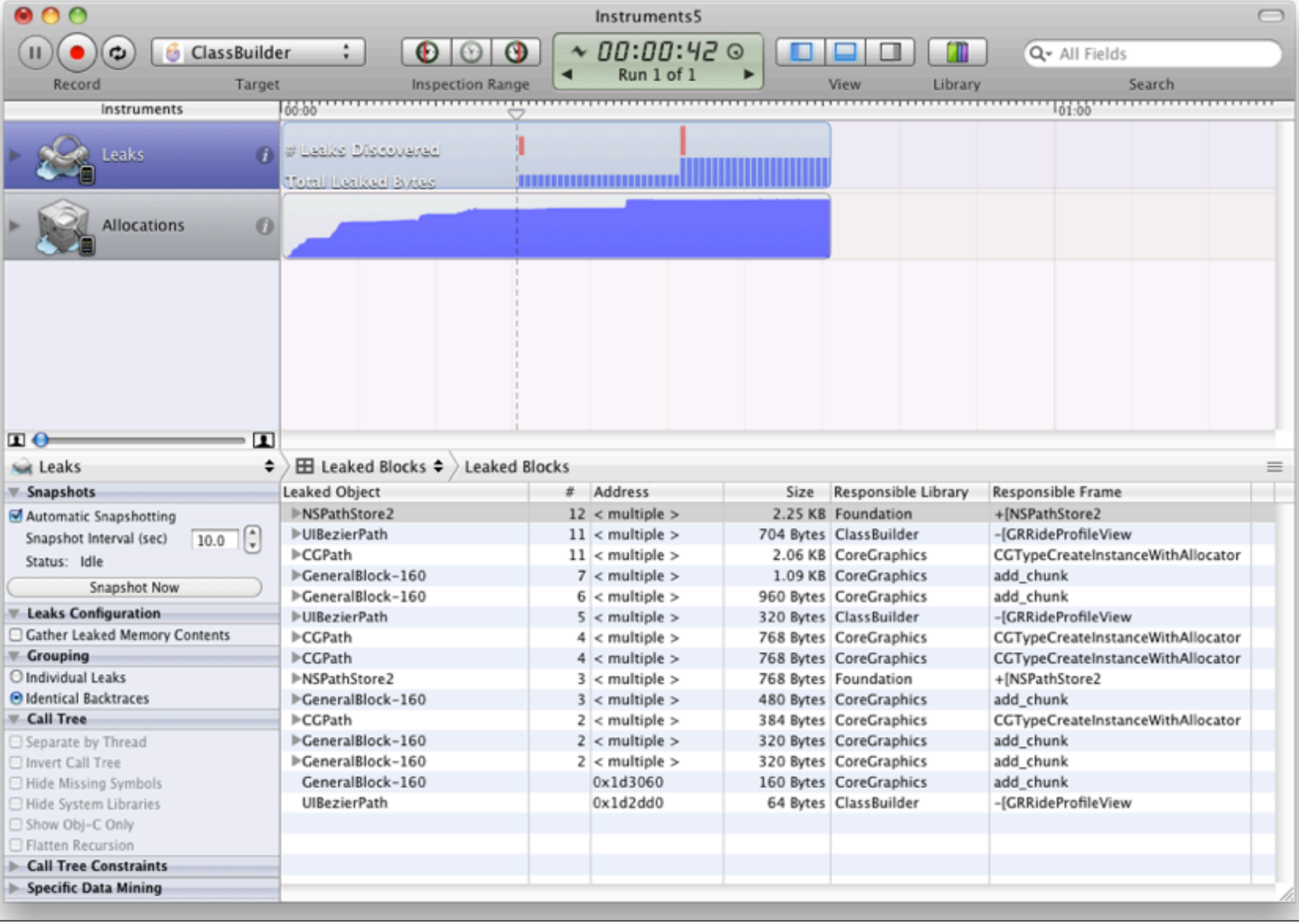
Memory Instruments



Playhead Messages



Look at the Leaked Blocks



Snapshots

☑ Automatic Snapshotting

Snapshot Interval (sec) 10.0

Status: Idle

Snapshot Now

Leaks Configuration

□ Gather Leaked Memory Contents

Grouping

○ Individual Leaks

● Identical Backtraces

Call Tree

□ Separate by Thread

□ Invert Call Tree

□ Hide Missing Symbols

□ Hide System Libraries

□ Show Obj-C Only

□ Flatten Recursion

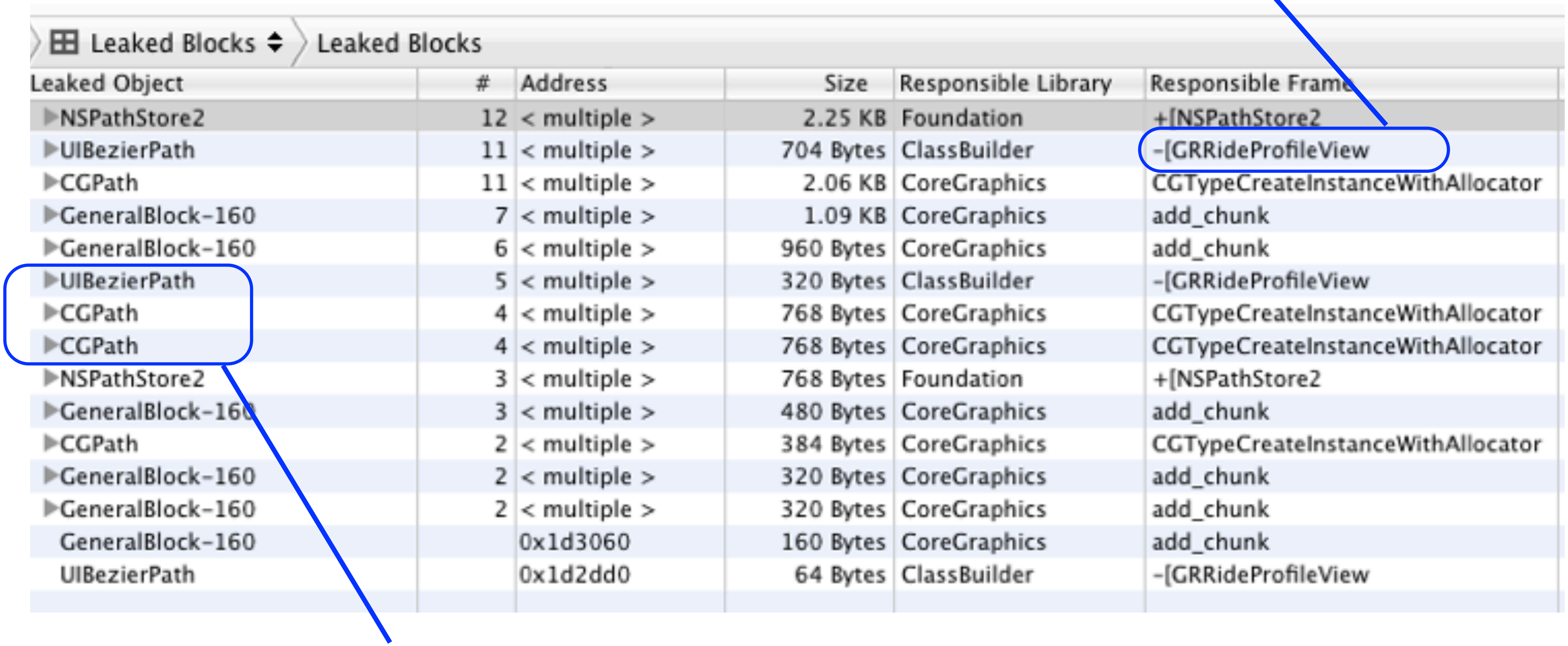
Call Tree Constraints

Specific Data Mining

Leaked Object	#	Address	Size	Responsible Library	Responsible Frame
▶NSPathStore2	12	< multiple >	2.25 KB	Foundation	+[NSPathStore2
▶UIBezierPath	11	< multiple >	704 Bytes	ClassBuilder	-[GRRideProfileView
▶CGPath	11	< multiple >	2.06 KB	CoreGraphics	CGTypeCreateInstanceWithAllocator
▶GeneralBlock-160	7	< multiple >	1.09 KB	CoreGraphics	add_chunk
▶GeneralBlock-160	6	< multiple >	960 Bytes	CoreGraphics	add_chunk
▶UIBezierPath	5	< multiple >	320 Bytes	ClassBuilder	-[GRRideProfileView
▶CGPath	4	< multiple >	768 Bytes	CoreGraphics	CGTypeCreateInstanceWithAllocator
▶CGPath	4	< multiple >	768 Bytes	CoreGraphics	CGTypeCreateInstanceWithAllocator
▶NSPathStore2	3	< multiple >	768 Bytes	Foundation	+[NSPathStore2
▶GeneralBlock-160	3	< multiple >	480 Bytes	CoreGraphics	add_chunk
▶CGPath	2	< multiple >	384 Bytes	CoreGraphics	CGTypeCreateInstanceWithAllocator
▶GeneralBlock-160	2	< multiple >	320 Bytes	CoreGraphics	add_chunk
▶GeneralBlock-160	2	< multiple >	320 Bytes	CoreGraphics	add_chunk
GeneralBlock-160		0x1d3060	160 Bytes	CoreGraphics	add_chunk
UIBezierPath		0x1d2dd0	64 Bytes	ClassBuilder	-[GRRideProfileView

Look at the Leaked Blocks

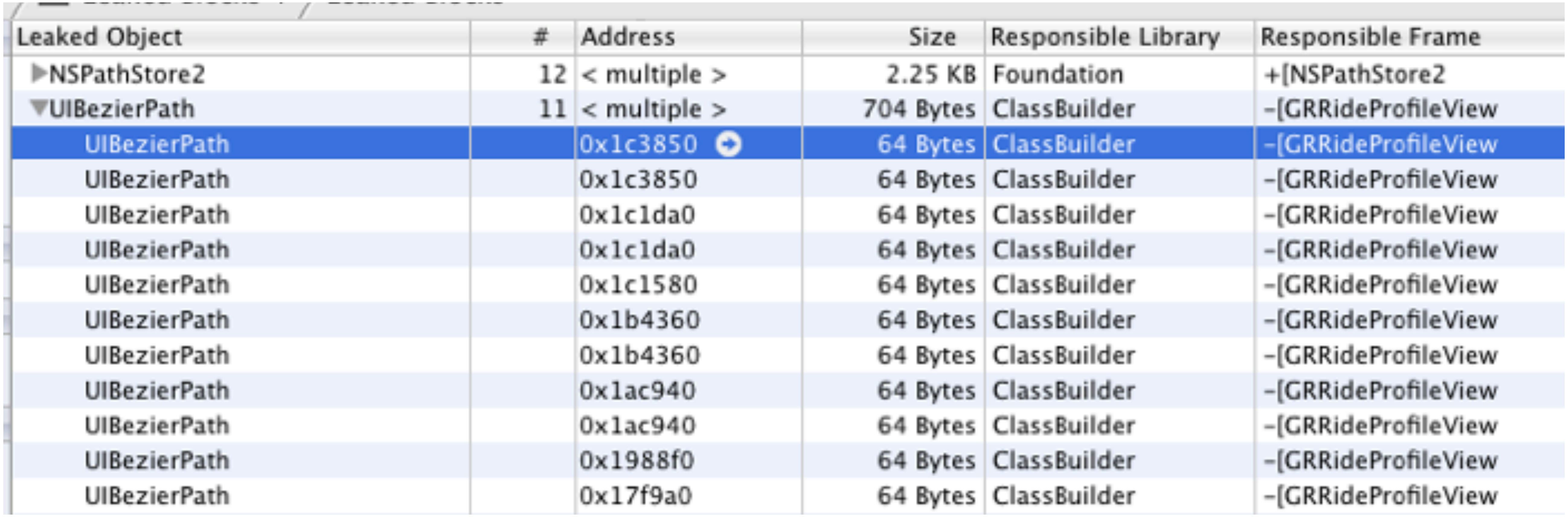
Probably the culprit



Leaked Object	#	Address	Size	Responsible Library	Responsible Frame
▶NSPathStore2	12	< multiple >	2.25 KB	Foundation	+[NSPathStore2
▶UIBezierPath	11	< multiple >	704 Bytes	ClassBuilder	-[GRRideProfileView
▶CGPath	11	< multiple >	2.06 KB	CoreGraphics	CGTypeCreateInstanceWithAllocator
▶GeneralBlock-160	7	< multiple >	1.09 KB	CoreGraphics	add_chunk
▶GeneralBlock-160	6	< multiple >	960 Bytes	CoreGraphics	add_chunk
▶UIBezierPath	5	< multiple >	320 Bytes	ClassBuilder	-[GRRideProfileView
▶CGPath	4	< multiple >	768 Bytes	CoreGraphics	CGTypeCreateInstanceWithAllocator
▶CGPath	4	< multiple >	768 Bytes	CoreGraphics	CGTypeCreateInstanceWithAllocator
▶NSPathStore2	3	< multiple >	768 Bytes	Foundation	+[NSPathStore2
▶GeneralBlock-160	3	< multiple >	480 Bytes	CoreGraphics	add_chunk
▶CGPath	2	< multiple >	384 Bytes	CoreGraphics	CGTypeCreateInstanceWithAllocator
▶GeneralBlock-160	2	< multiple >	320 Bytes	CoreGraphics	add_chunk
▶GeneralBlock-160	2	< multiple >	320 Bytes	CoreGraphics	add_chunk
GeneralBlock-160		0x1d3060	160 Bytes	CoreGraphics	add_chunk
UIBezierPath		0x1d2dd0	64 Bytes	ClassBuilder	-[GRRideProfileView

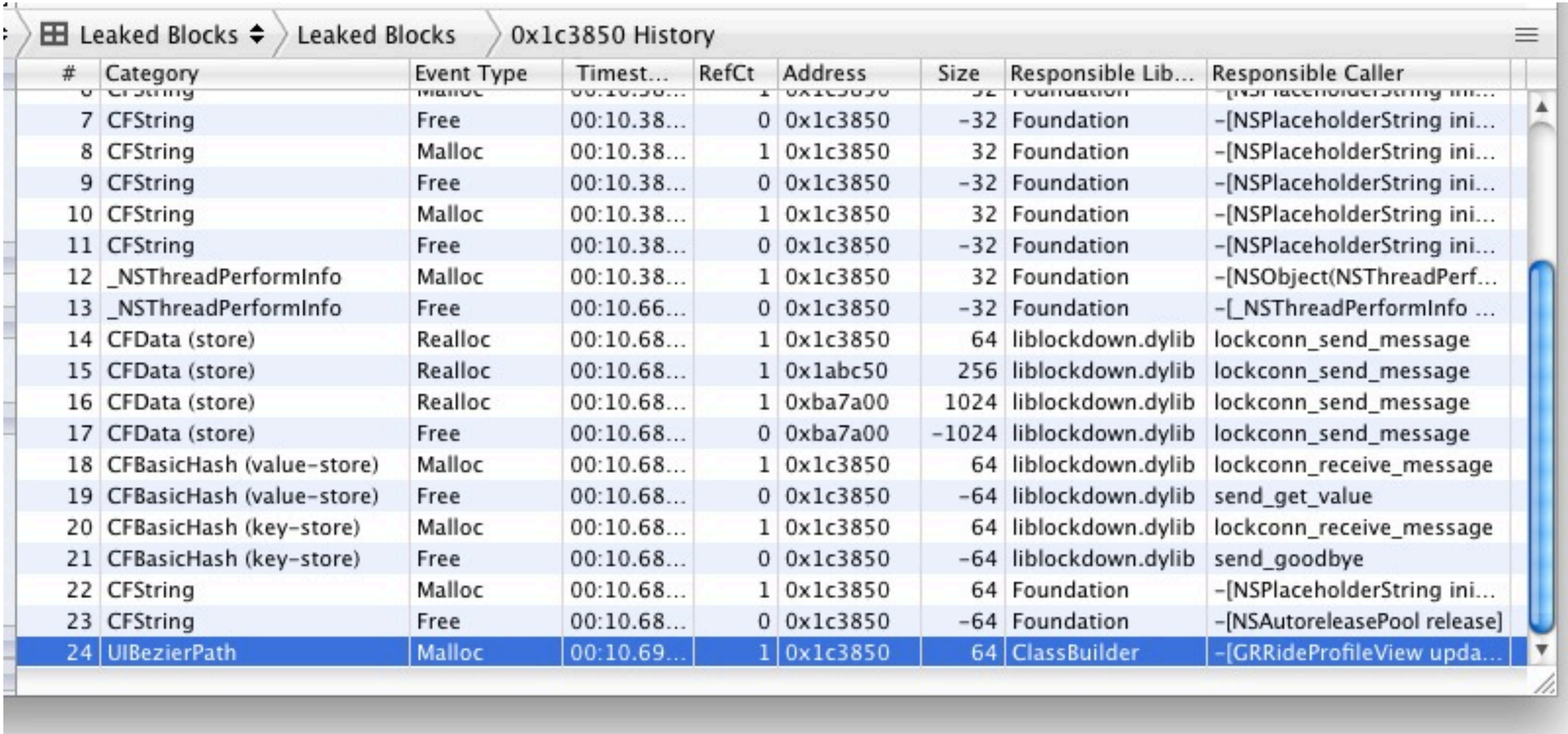
Seems to be a pattern

List o’ blocks



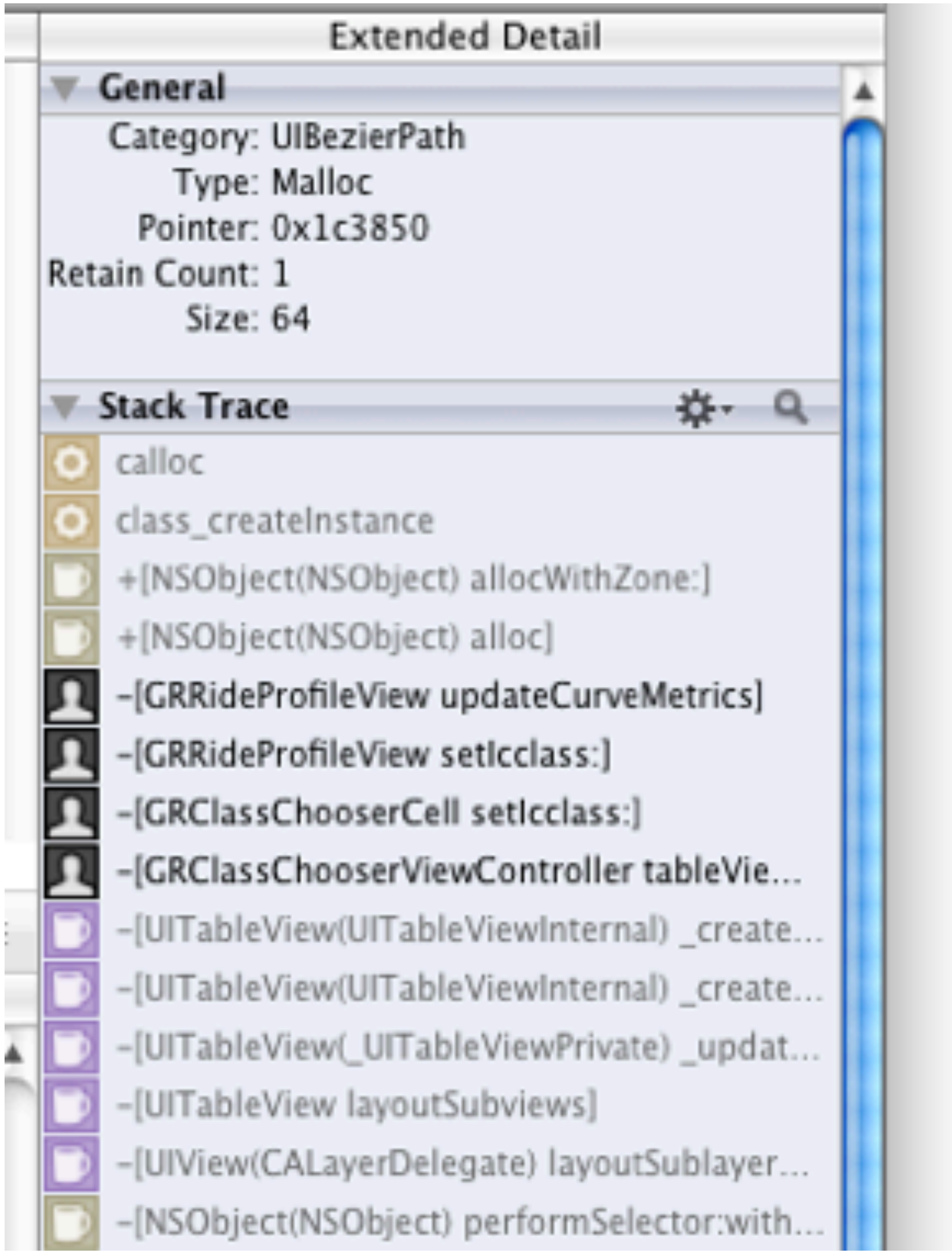
Leaked Object	#	Address	Size	Responsible Library	Responsible Frame
▶NSPathStore2	12	< multiple >	2.25 KB	Foundation	+[NSPathStore2
▼UIBezierPath	11	< multiple >	704 Bytes	ClassBuilder	-[GRRideProfileView
UIBezierPath		0x1c3850 ➔	64 Bytes	ClassBuilder	-[GRRideProfileView
UIBezierPath		0x1c3850	64 Bytes	ClassBuilder	-[GRRideProfileView
UIBezierPath		0x1c1da0	64 Bytes	ClassBuilder	-[GRRideProfileView
UIBezierPath		0x1c1da0	64 Bytes	ClassBuilder	-[GRRideProfileView
UIBezierPath		0x1c1580	64 Bytes	ClassBuilder	-[GRRideProfileView
UIBezierPath		0x1b4360	64 Bytes	ClassBuilder	-[GRRideProfileView
UIBezierPath		0x1b4360	64 Bytes	ClassBuilder	-[GRRideProfileView
UIBezierPath		0x1ac940	64 Bytes	ClassBuilder	-[GRRideProfileView
UIBezierPath		0x1ac940	64 Bytes	ClassBuilder	-[GRRideProfileView
UIBezierPath		0x1988f0	64 Bytes	ClassBuilder	-[GRRideProfileView
UIBezierPath		0x17f9a0	64 Bytes	ClassBuilder	-[GRRideProfileView

Allocation History



#	Category	Event Type	Timest...	RefCt	Address	Size	Responsible Lib...	Responsible Caller
6	CFString	Malloc	00:10.38...	1	0x1c3850	32	Foundation	-[NSPlaceholderString ini...
7	CFString	Free	00:10.38...	0	0x1c3850	-32	Foundation	-[NSPlaceholderString ini...
8	CFString	Malloc	00:10.38...	1	0x1c3850	32	Foundation	-[NSPlaceholderString ini...
9	CFString	Free	00:10.38...	0	0x1c3850	-32	Foundation	-[NSPlaceholderString ini...
10	CFString	Malloc	00:10.38...	1	0x1c3850	32	Foundation	-[NSPlaceholderString ini...
11	CFString	Free	00:10.38...	0	0x1c3850	-32	Foundation	-[NSPlaceholderString ini...
12	_NSThreadPerformInfo	Malloc	00:10.38...	1	0x1c3850	32	Foundation	-[NSObject(NSThreadPerf...
13	_NSThreadPerformInfo	Free	00:10.66...	0	0x1c3850	-32	Foundation	-[_NSThreadPerformInfo ...
14	CFData (store)	Realloc	00:10.68...	1	0x1c3850	64	liblockdown.dylib	lockconn_send_message
15	CFData (store)	Realloc	00:10.68...	1	0x1abc50	256	liblockdown.dylib	lockconn_send_message
16	CFData (store)	Realloc	00:10.68...	1	0xba7a00	1024	liblockdown.dylib	lockconn_send_message
17	CFData (store)	Free	00:10.68...	0	0xba7a00	-1024	liblockdown.dylib	lockconn_send_message
18	CFBasicHash (value-store)	Malloc	00:10.68...	1	0x1c3850	64	liblockdown.dylib	lockconn_receive_message
19	CFBasicHash (value-store)	Free	00:10.68...	0	0x1c3850	-64	liblockdown.dylib	send_get_value
20	CFBasicHash (key-store)	Malloc	00:10.68...	1	0x1c3850	64	liblockdown.dylib	lockconn_receive_message
21	CFBasicHash (key-store)	Free	00:10.68...	0	0x1c3850	-64	liblockdown.dylib	send_goodbye
22	CFString	Malloc	00:10.68...	1	0x1c3850	64	Foundation	-[NSPlaceholderString ini...
23	CFString	Free	00:10.68...	0	0x1c3850	-64	Foundation	-[NSAutoreleasePool release]
24	UIBezierPath	Malloc	00:10.69...	1	0x1c3850	64	ClassBuilder	-[GRRideProfileView upda...

Stack Trace



Extended Detail

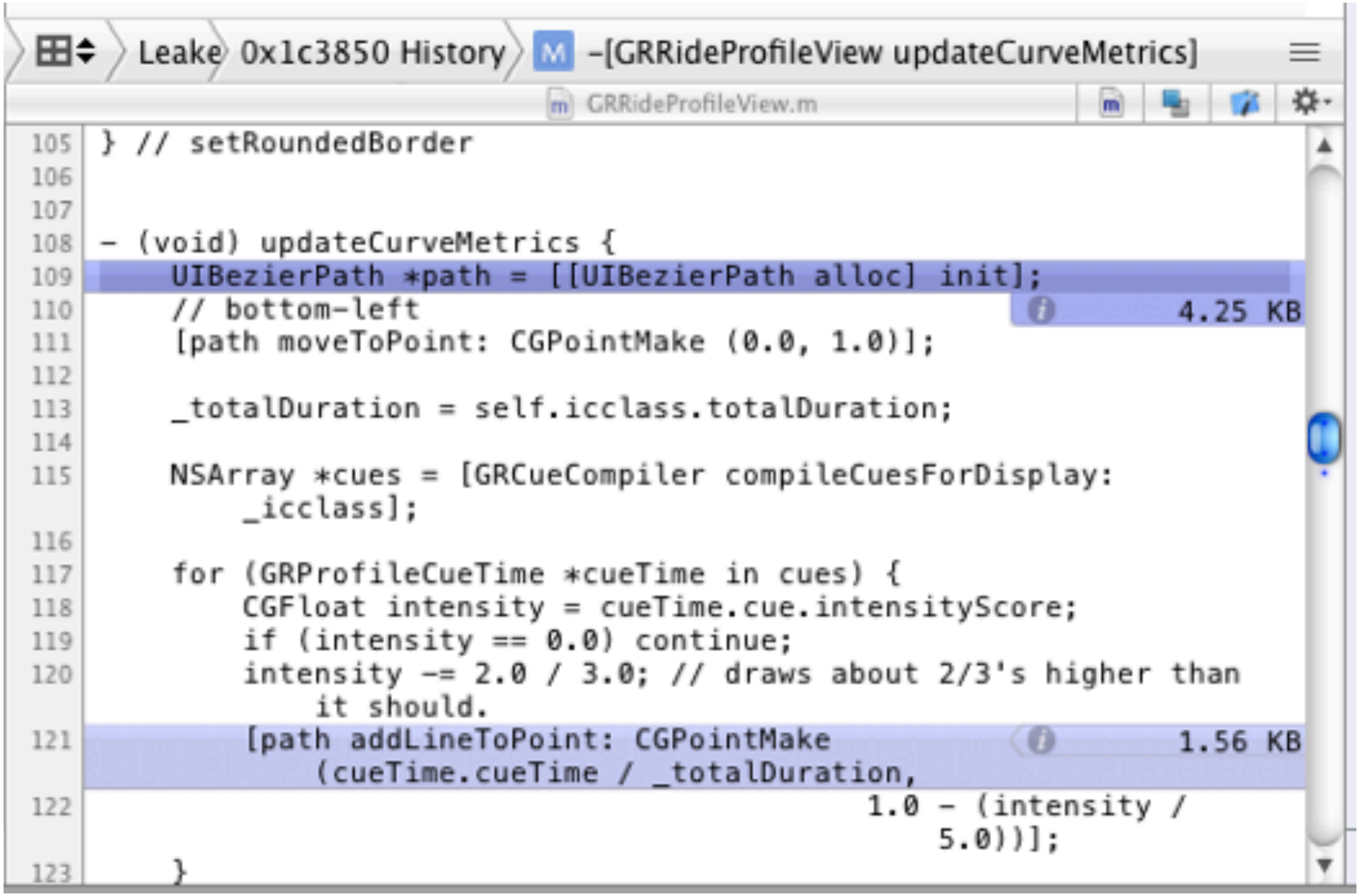
General

Category: UIBezierPath
Type: Malloc
Pointer: 0x1c3850
Retain Count: 1
Size: 64

Stack Trace

```
calloc
class_createInstance
+[NSObject(NSObject) allocWithZone:]
+[NSObject(NSObject) alloc]
-[GRRideProfileView updateCurveMetrics]
-[GRRideProfileView setIcclass:]
-[GRClassChooserCell setIcclass:]
-[GRClassChooserViewController tableVie...
-[UITableView(UITableViewInternal) _create...
-[UITableView(UITableViewInternal) _create...
-[UITableView(_UITableViewPrivate) _updat...
-[UITableView layoutSubviews]
-[UIView(CALayerDelegate) layoutSublayer...
-[NSObject(NSObject) performSelector:with...
```

The Culprit

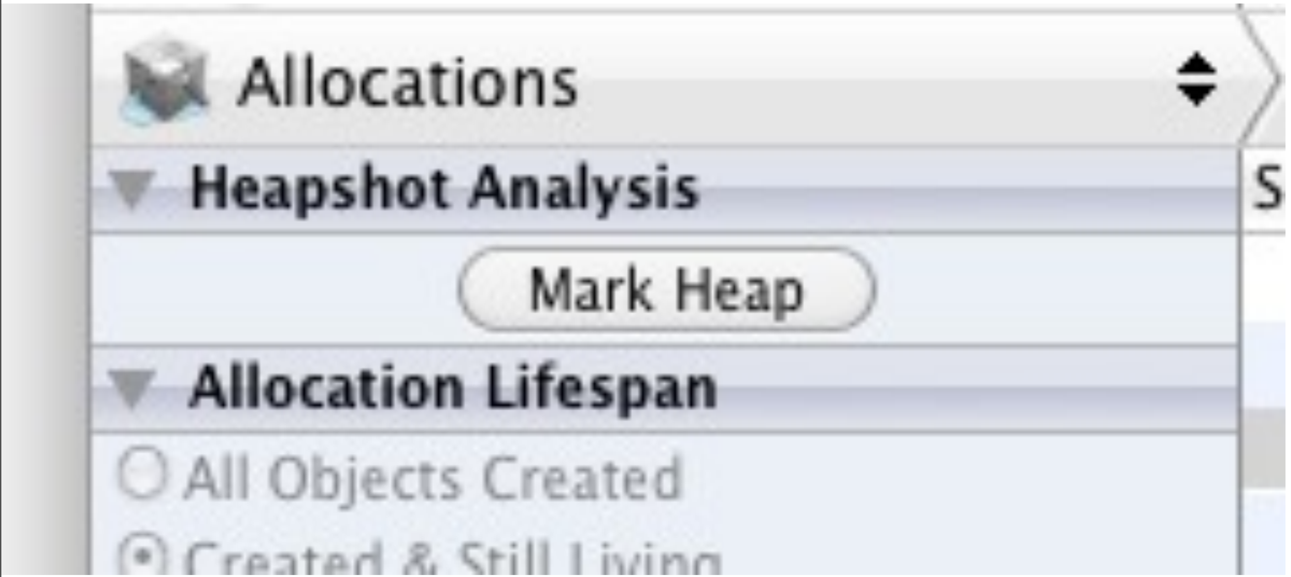


```
} // setRoundedBorder


- (void) updateCurveMetrics {
    UIBezierPath *path = [[UIBezierPath alloc] init];
    // bottom-left
    [path moveToPoint: CGPointMake (0.0, 1.0)];

    _totalDuration = self.icclass.totalDuration;

    NSArray *cues = [GRCueCompiler compileCuesForDisplay:
        _icclass];

    for (GRProfileCueTime *cueTime in cues) {
        CGFloat intensity = cueTime.cue.intensityScore;
        if (intensity == 0.0) continue;
        intensity -= 2.0 / 3.0; // draws about 2/3's higher than
            it should.
        [path addLineToPoint: CGPointMake
            (cueTime.cueTime / _totalDuration,
                                              1.0 - (intensity /
                                                  5.0))];
    }
```

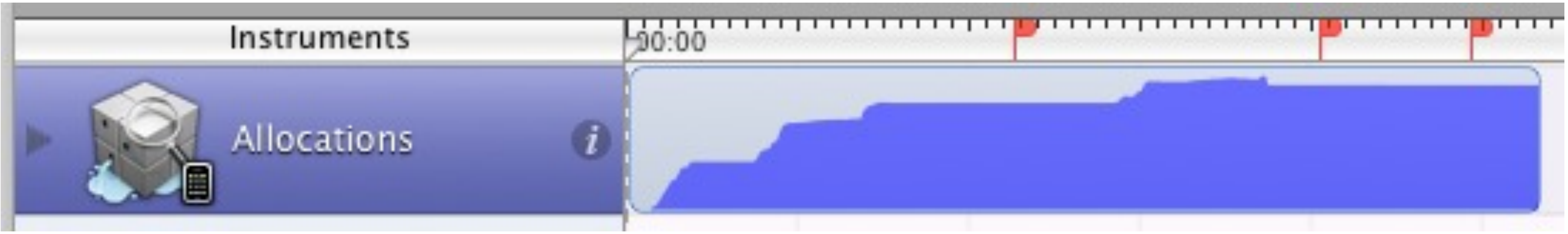
Abandoned Memory

- Leaked : allocated memory with no reference
- Abandoned : referenced, but not used
  - Left over caches
  - View added to superview and never removed
  - Harder to detect, more false positives

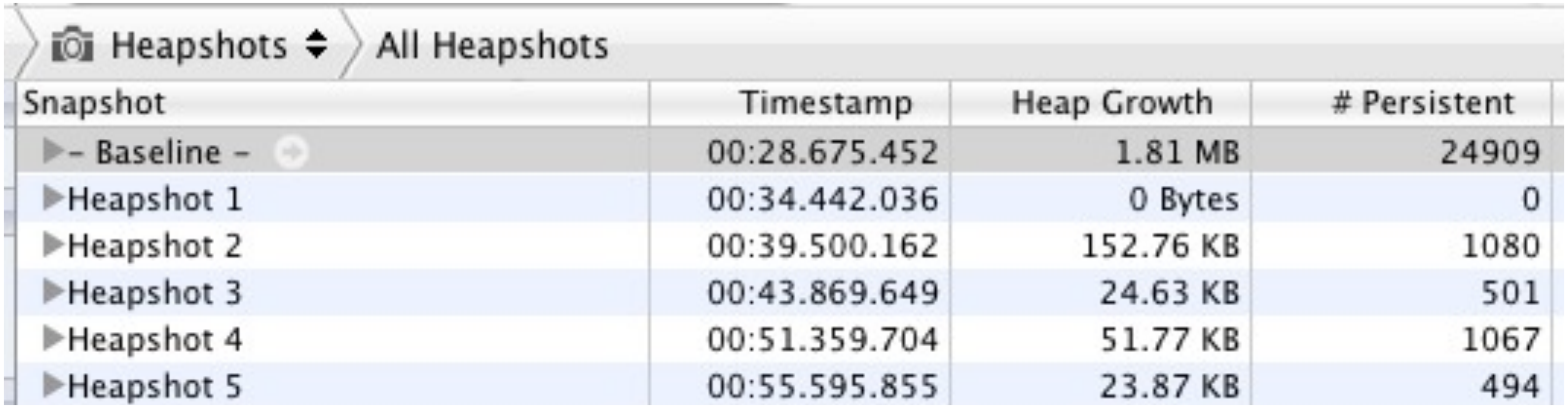
HeapShots



- Mark a heap to create a baseline
- Mark it again (and again, and again)
- Diff two heapshots to see what's new



Tableview Crash

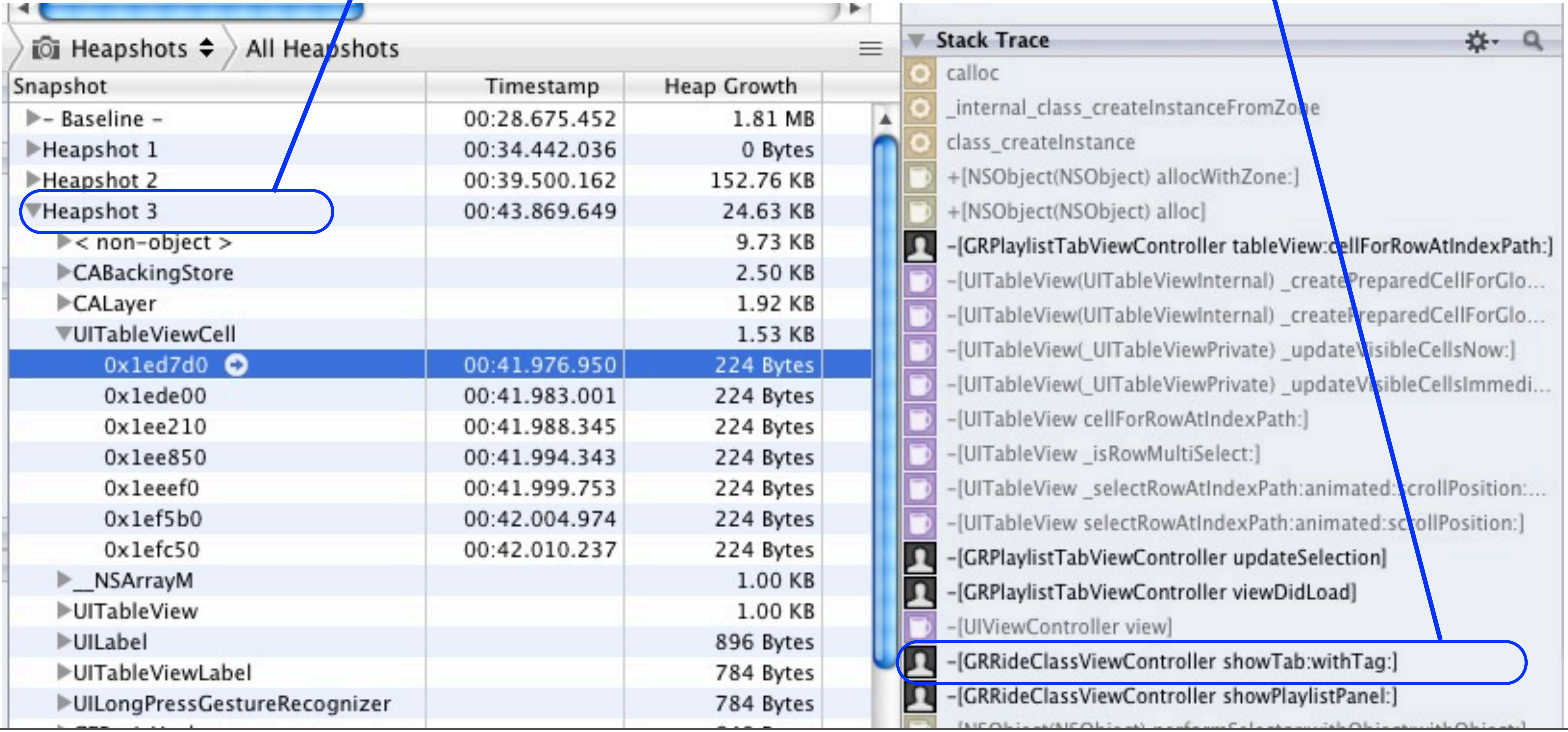


Snapshot	Timestamp	Heap Growth	# Persistent
- Baseline -	00:28.675.452	1.81 MB	24909
Heapshot 1	00:34.442.036	0 Bytes	0
Heapshot 2	00:39.500.162	152.76 KB	1080
Heapshot 3	00:43.869.649	24.63 KB	501
Heapshot 4	00:51.359.704	51.77 KB	1067
Heapshot 5	00:55.595.855	23.87 KB	494

Looking at a Heapshot

Heapshot of interest

Initiated the load



Snapshot	Timestamp	Heap Growth
▶ - Baseline -	00:28.675.452	1.81 MB
▶ Heapshot 1	00:34.442.036	0 Bytes
▶ Heapshot 2	00:39.500.162	152.76 KB
▼ Heapshot 3	00:43.869.649	24.63 KB
▶ < non-object >		9.73 KB
▶ CABackingStore		2.50 KB
▶ CALayer		1.92 KB
▼ UITableViewCell		1.53 KB
0x1ed7d0	00:41.976.950	224 Bytes
0x1ede00	00:41.983.001	224 Bytes
0x1ee210	00:41.988.345	224 Bytes
0x1ee850	00:41.994.343	224 Bytes
0x1eeef0	00:41.999.753	224 Bytes
0x1ef5b0	00:42.004.974	224 Bytes
0x1efc50	00:42.010.237	224 Bytes
▶ __NSArrayM		1.00 KB
▶ UITableView		1.00 KB
▶ UILabel		896 Bytes
▶ UITableViewLabel		784 Bytes
▶ UILongPressGestureRecognizer		784 Bytes

Stack Trace

calloc

_internal_class_createInstanceFromZone

class_createInstance

+[NSObject(NSObject) allocWithZone:]

+[NSObject(NSObject) alloc]

-[GRPlaylistTabViewController tableView:cellForRowAtIndexPath:]

-[UITableView(UITableViewInternal) _createPreparedCellForGlo...

-[UITableView(UITableViewInternal) _createPreparedCellForGlo...

-[UITableView(_UITableViewPrivate) _updateVisibleCellsNow:]

-[UITableView(_UITableViewPrivate) _updateVisibleCellsImmedi...

-[UITableView cellForRowAtIndexPath:]

-[UITableView _isRowMultiSelect:]

-[UITableView _selectRowAtIndexPath:animated:scrollPosition:...

-[UITableView selectRowAtIndexPath:animated:scrollPosition:]

-[GRPlaylistTabViewController updateSelection]

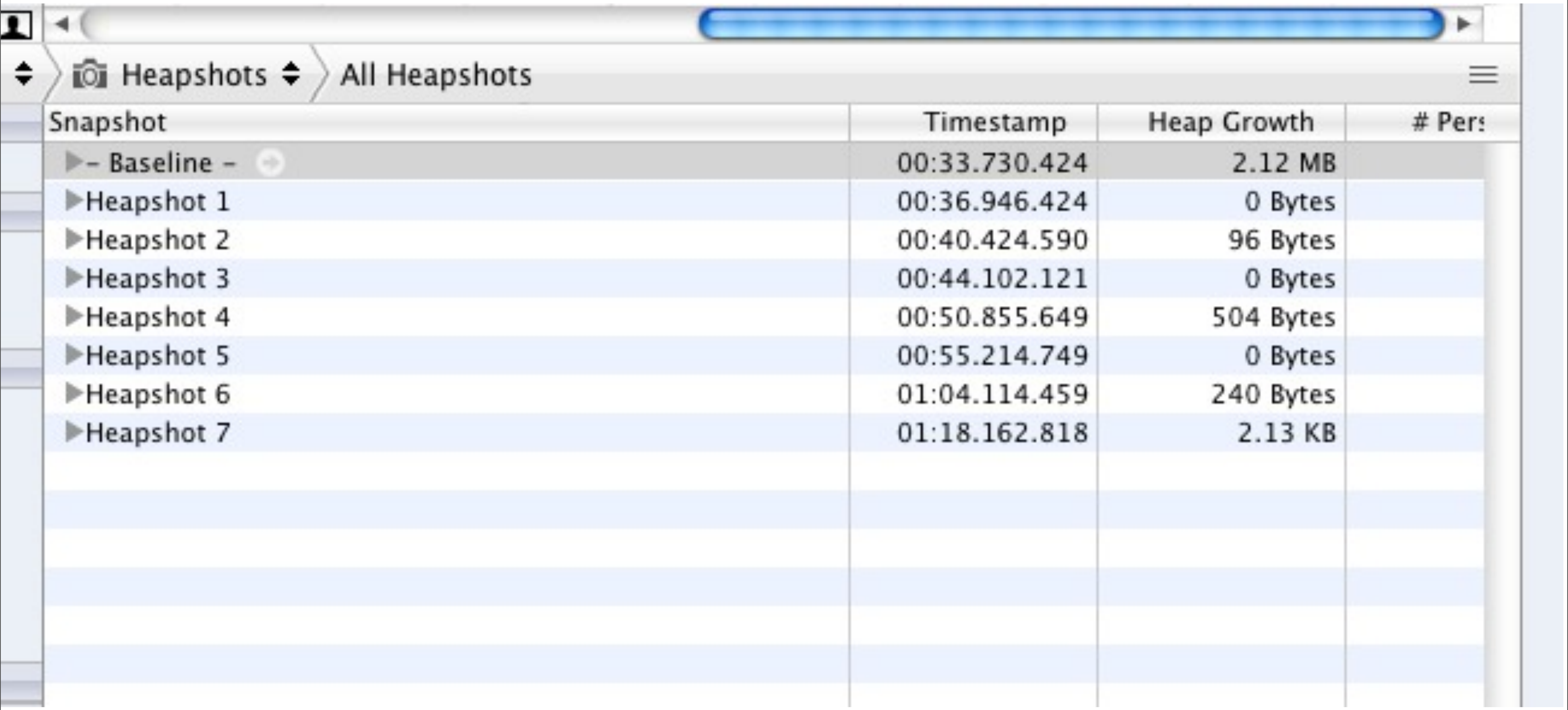
-[GRPlaylistTabViewController viewDidLoad]

-[UIViewController view]

-[GRRideClassViewController showTab:withTag:]

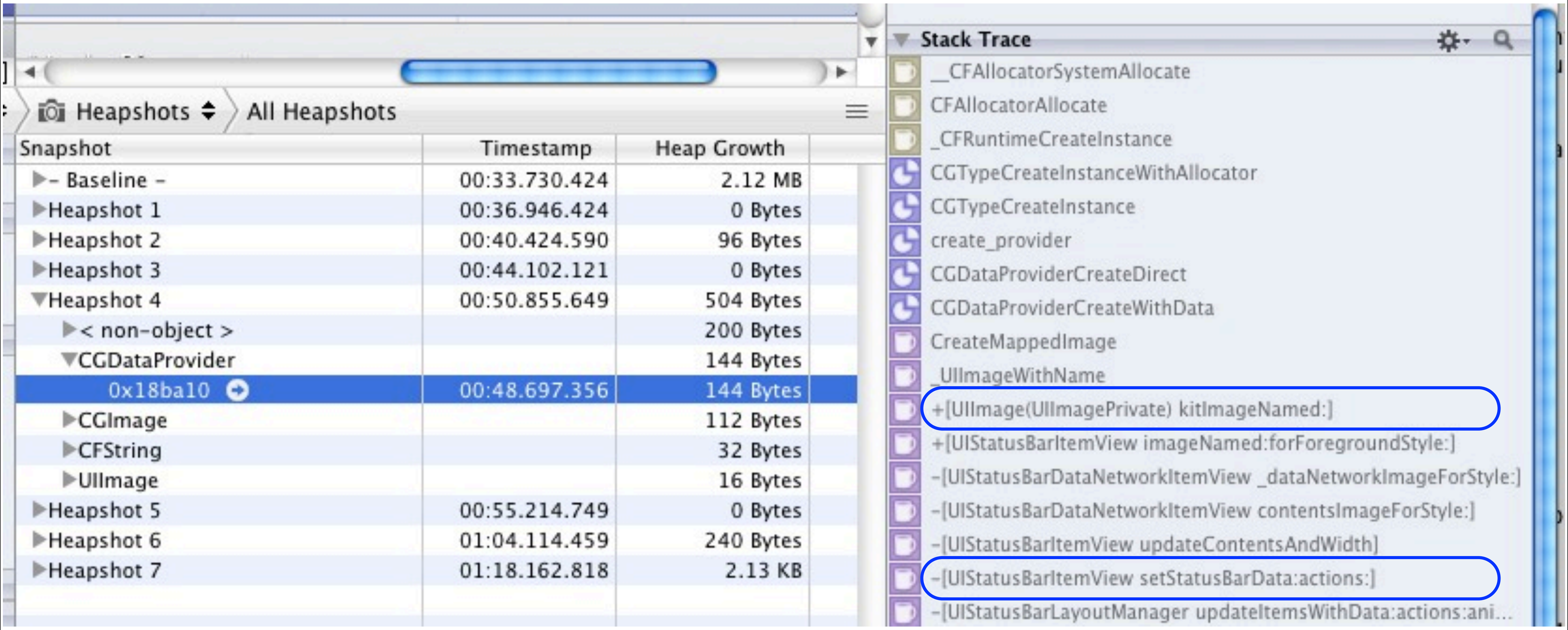
-[GRRideClassViewController showPlaylistPanel:]

After the Fix



Snapshot	Timestamp	Heap Growth	# Per:
– Baseline –	00:33.730.424	2.12 MB	
Heapshot 1	00:36.946.424	0 Bytes	
Heapshot 2	00:40.424.590	96 Bytes	
Heapshot 3	00:44.102.121	0 Bytes	
Heapshot 4	00:50.855.649	504 Bytes	
Heapshot 5	00:55.214.749	0 Bytes	
Heapshot 6	01:04.114.459	240 Bytes	
Heapshot 7	01:18.162.818	2.13 KB	

From the System

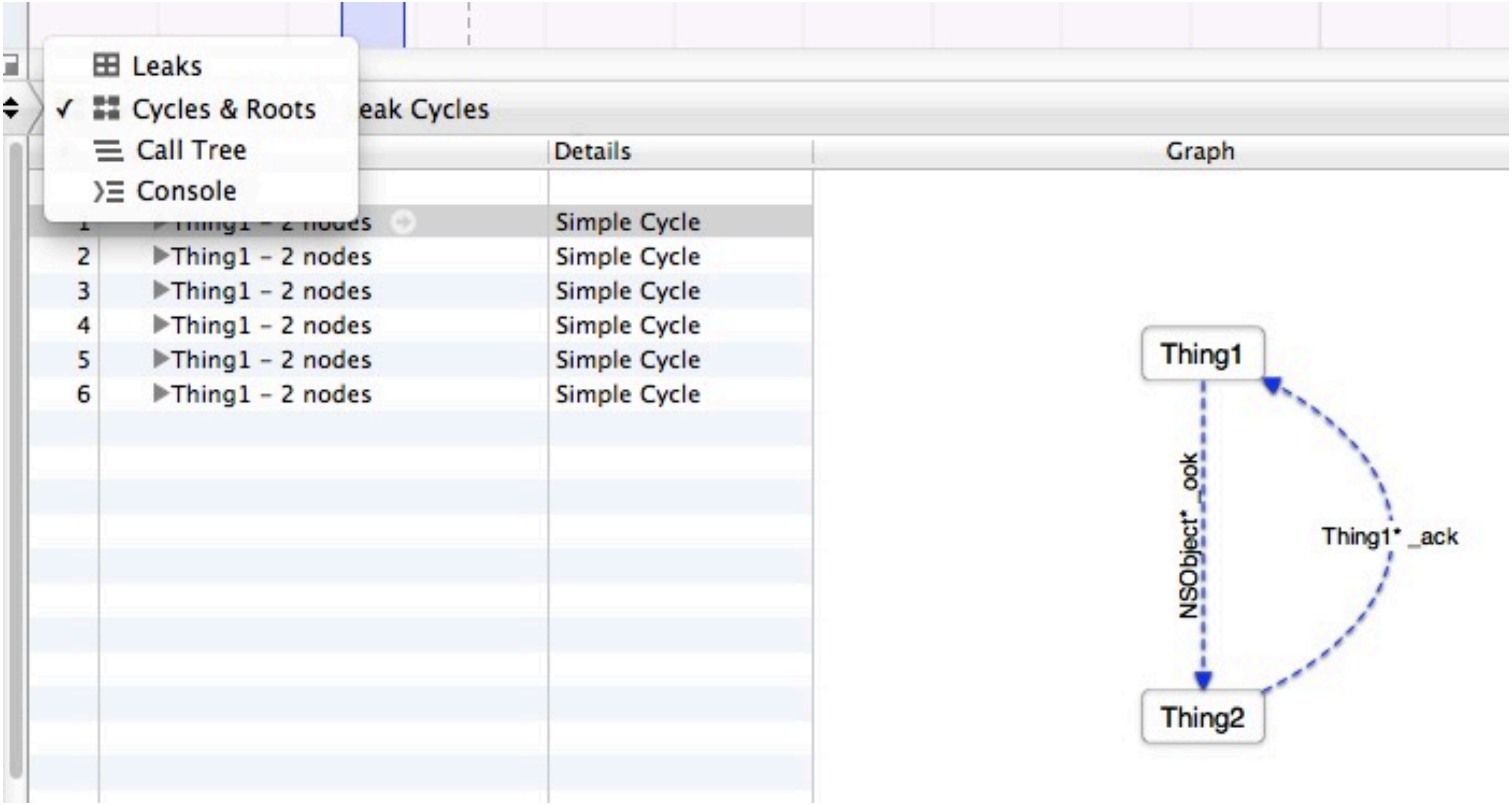


Snapshot	Timestamp	Heap Growth
▶ - Baseline -	00:33.730.424	2.12 MB
▶ Heapshot 1	00:36.946.424	0 Bytes
▶ Heapshot 2	00:40.424.590	96 Bytes
▶ Heapshot 3	00:44.102.121	0 Bytes
▼ Heapshot 4	00:50.855.649	504 Bytes
▶ < non-object >		200 Bytes
▼ CGDataProvider		144 Bytes
0x18ba10	00:48.697.356	144 Bytes
▶ CGImage		112 Bytes
▶ CFString		32 Bytes
▶ UIImage		16 Bytes
▶ Heapshot 5	00:55.214.749	0 Bytes
▶ Heapshot 6	01:04.114.459	240 Bytes
▶ Heapshot 7	01:18.162.818	2.13 KB

Stack Trace

```
__CFAllocatorSystemAllocate
CFAllocatorAllocate
_CFRuntimeCreateInstance
CGTypeCreateInstanceWithAllocator
CGTypeCreateInstance
create_provider
CGDataProviderCreateDirect
CGDataProviderCreateWithData
CreateMappedImage
_UIImageWithName
+[UIImage(UIImagePrivate) kitImageNamed:]
+[UIStatusBarItemView imageNamed:forForegroundStyle:]
-[UIStatusBarDataNetworkItemView _dataNetworkImageForStyle:]
-[UIStatusBarDataNetworkItemView contentsImageForStyle:]
-[UIStatusBarItemView updateContentsAndWidth]
-[UIStatusBarItemView setStatusBarData:actions:]
-[UIStatusBarLayoutManager updateItemsWithData:actions:ani...
```

Retain Cycles



		Details
1	▶Thing1 – 2 nodes	Simple Cycle
2	▶Thing1 – 2 nodes	Simple Cycle
3	▶Thing1 – 2 nodes	Simple Cycle
4	▶Thing1 – 2 nodes	Simple Cycle
5	▶Thing1 – 2 nodes	Simple Cycle
6	▶Thing1 – 2 nodes	Simple Cycle

Disk / File System

Disk / File System

- **Extremely** slow
  - SSDs helping, but still slow
- Large files have locality of reference
- Avoid when you can

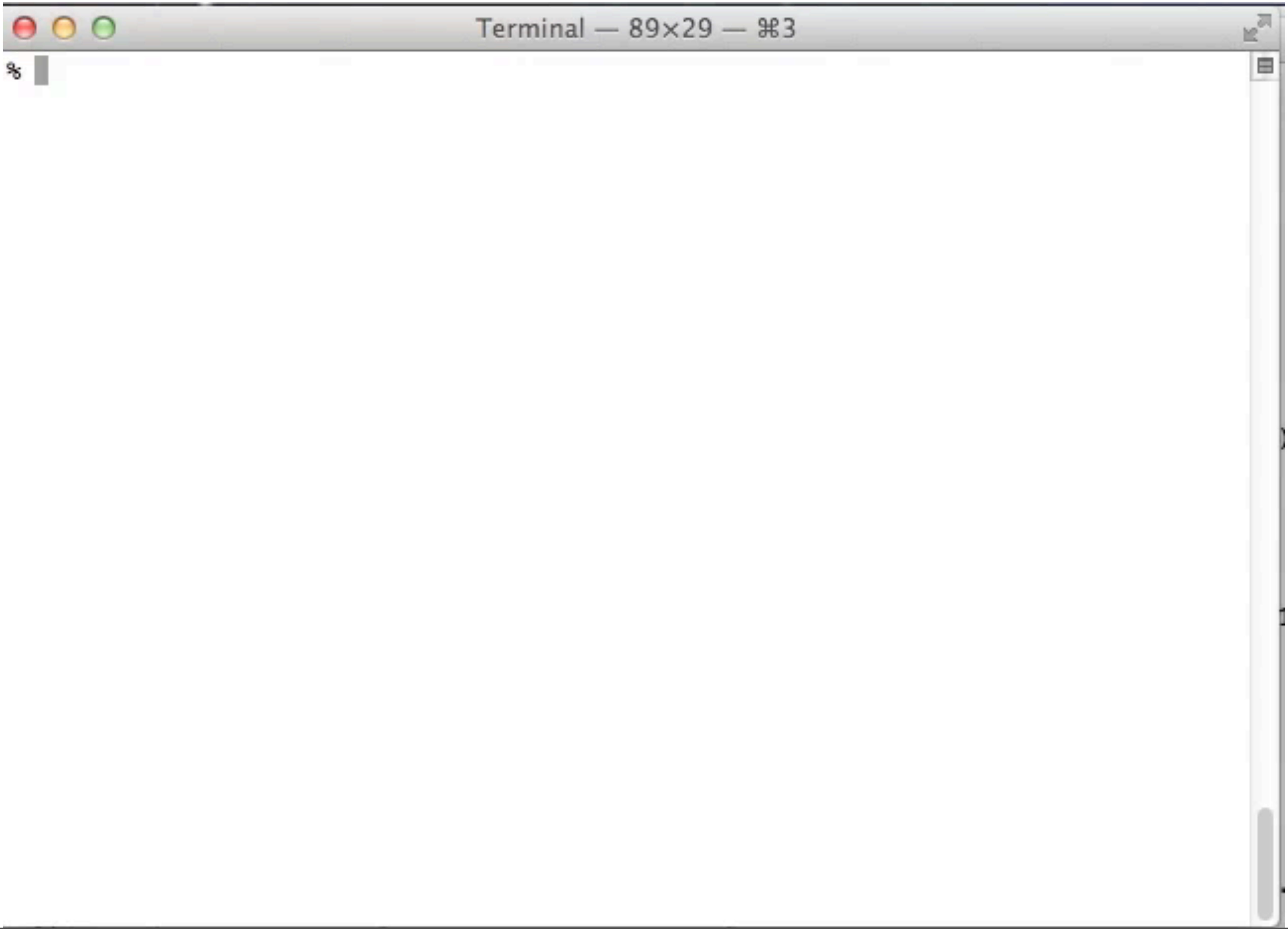
fs_usage

```
18:24:32  close                                   0.000053   Safari
18:24:32  select                                  0.100163 W Safari
18:24:32  mkdir   /Users/markd/Library/Cookies    0.000027   Safari
18:24:32  open    ibrary/Cookies/Cookies.plist    0.000672   Safari
18:24:32  read                                    0.569160 W fseventsd
18:24:32  lstat   /Users/markd/Library/Cookies    0.000038   fseventsd
18:24:32  read                                    0.569514 W mds
18:24:32  select                                  0.100035 W Safari
18:24:32  select                                  0.100033 W Safari
18:24:32  write                                   0.004993 W Safari
18:24:32  close                                   0.000544   Safari
18:24:32  read                                    0.238500 W fseventsd
18:24:32  read                                    0.238142 W mds
18:24:32  fcntl                                   0.000018   mds
18:24:32  read                                    0.000021   Safari
18:24:32  sendto                                  0.000018   Safari
18:24:32  select                                  0.067144 W Safari
18:24:32  recvfro                                 0.000007   Safari
```

sc_usage

```
BigShow           0 preemptions    0 context switches    2 threads    18:27:18
                  0 faults         0 system calls                      0:00:43

TYPE                            NUMBER        CPU_TIME   WAIT_TIME
-------------------------------------------------------------------------------
System         Idle                                      0:39.162( 0:00.903)
System         Busy                                      0:02.757( 0:00.100)
BigShow        Usermode                       0:00.184

mach_msg_trap                   323           0:00.003   0:41.342( 0:01.002) W
semwait_signal                    2           0:00.000   0:40.654( 0:01.002) W
mach_port_insert_member           9           0:00.000   0:00.001
io_connect_method                47           0:00.000   0:00.000
io_service_get_matching           1           0:00.001   0:00.000
vm_deallocate                     5           0:00.000   0:00.000
munmap                           48           0:00.000   0:00.000
getuid                            1           0:00.000
geteuid                           3           0:00.000

CURRENT_TYPE              LAST_PATHNAME_WAITED_FOR     CUR_WAIT_TIME THRD# PRI
-------------------------------------------------------------------------------
mach_msg_trap                                                 0:38.820   0  46
semwait_signal                                                0:40.653   1  47
```

sc_usage



```
% 
```

Network

Network

- Networks can be slow, especially WAN
- Beware latency
- Don't block the main thread
  - Like with DNS lookups
  - Prefer CFHost to gethostbyname2
- Double-buffer if you can

WWDC 2012 Session 706 :
Networking Best Practices

Tools You Can Use

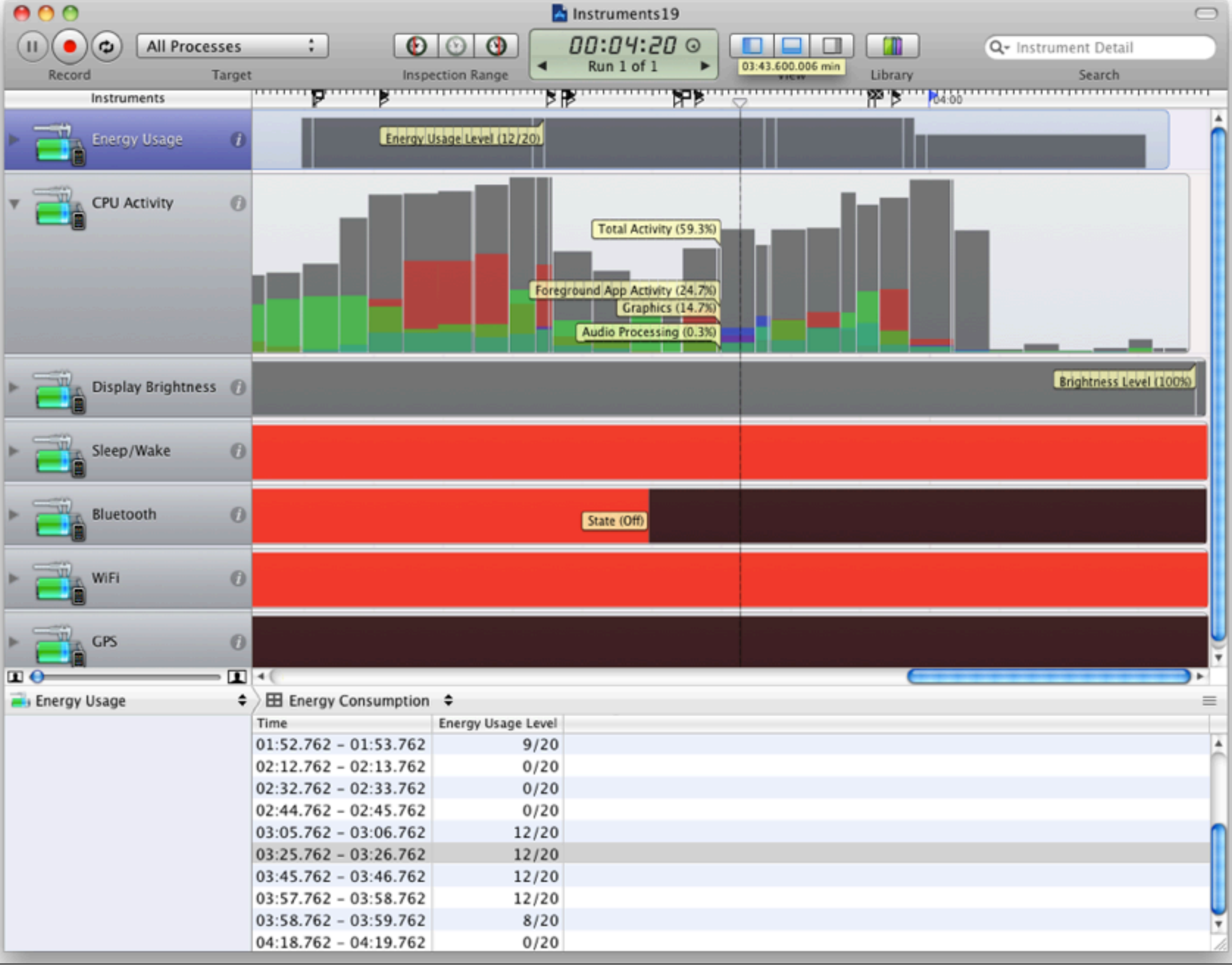
- Instruments
  - Network Activity Monitor
- Network Line Conditioner
- Charles web debugging proxy
- WireShark, etc

Power

Power

- Very important for mobile users - phones and laptops
- They're not happy if their phone shuts down in the middle of the day
- Usually fixed by fixing other problems, especially CPU and Graphics
- Try to be bursty in your use, letting the chip move to a low-power state when waiting for the user

So Much Power *It’s Just Ridiculous*



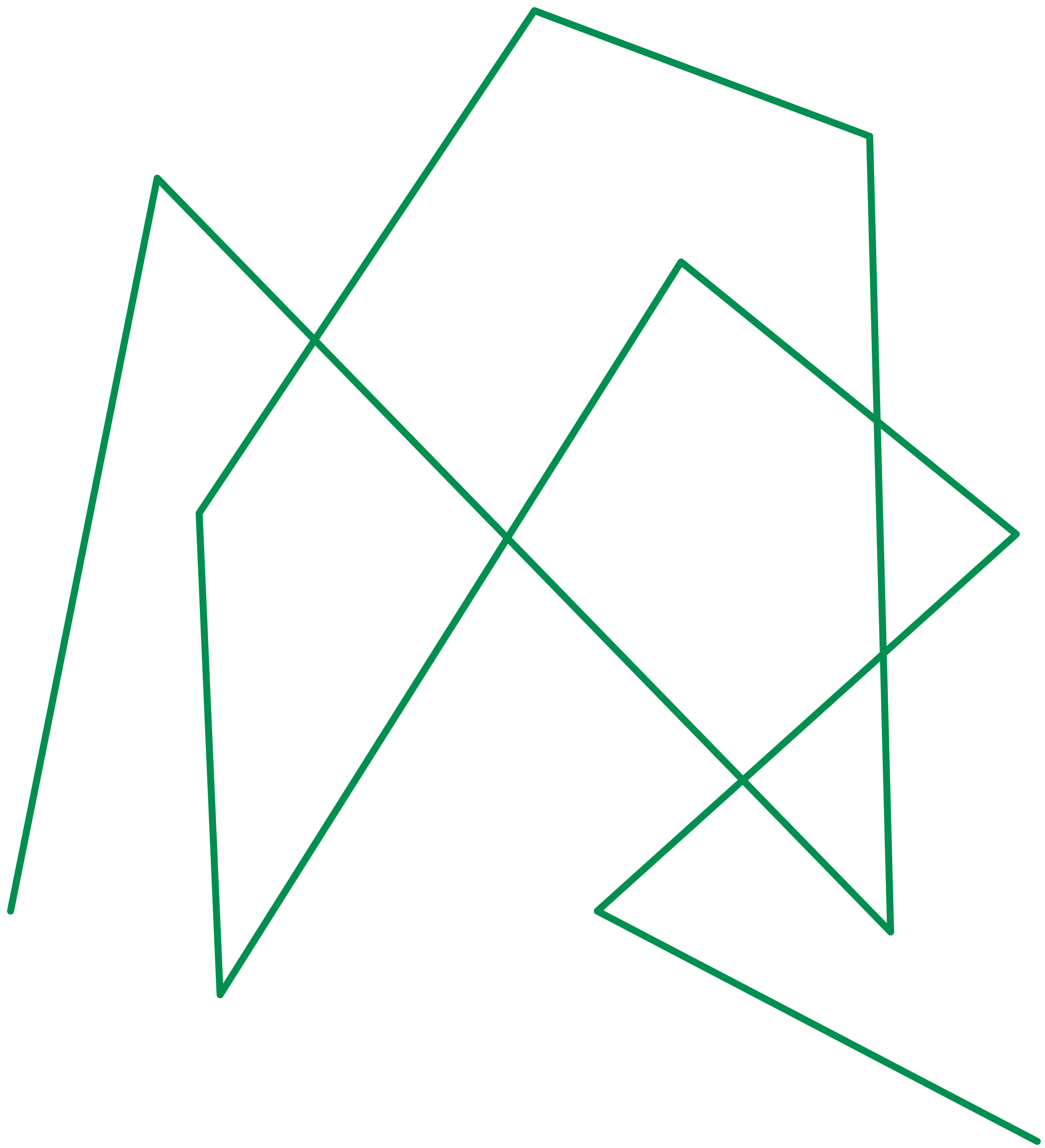
Time	Energy Usage Level
01:52.762 - 01:53.762	9/20
02:12.762 - 02:13.762	0/20
02:32.762 - 02:33.762	0/20
02:44.762 - 02:45.762	0/20
03:05.762 - 03:06.762	12/20
03:25.762 - 03:26.762	12/20
03:45.762 - 03:46.762	12/20
03:57.762 - 03:58.762	12/20
03:58.762 - 03:59.762	8/20
04:18.762 - 04:19.762	0/20

Graphics

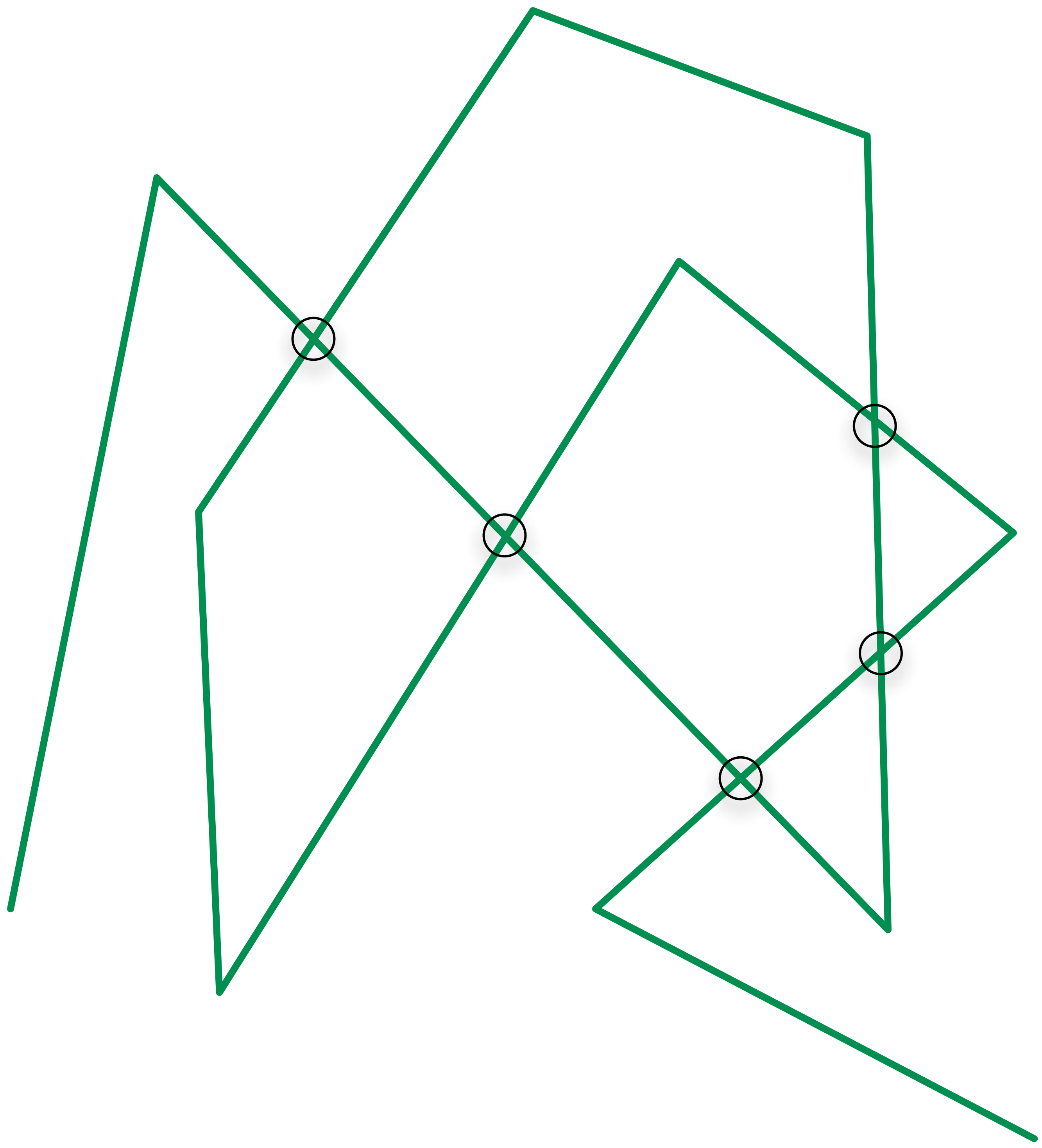
Graphics

- Loading / initializing images is expensive
- Transparency is expensive
- Blending images is expensive
- Resizing images is expensive
- Quartz line crossings are expensive

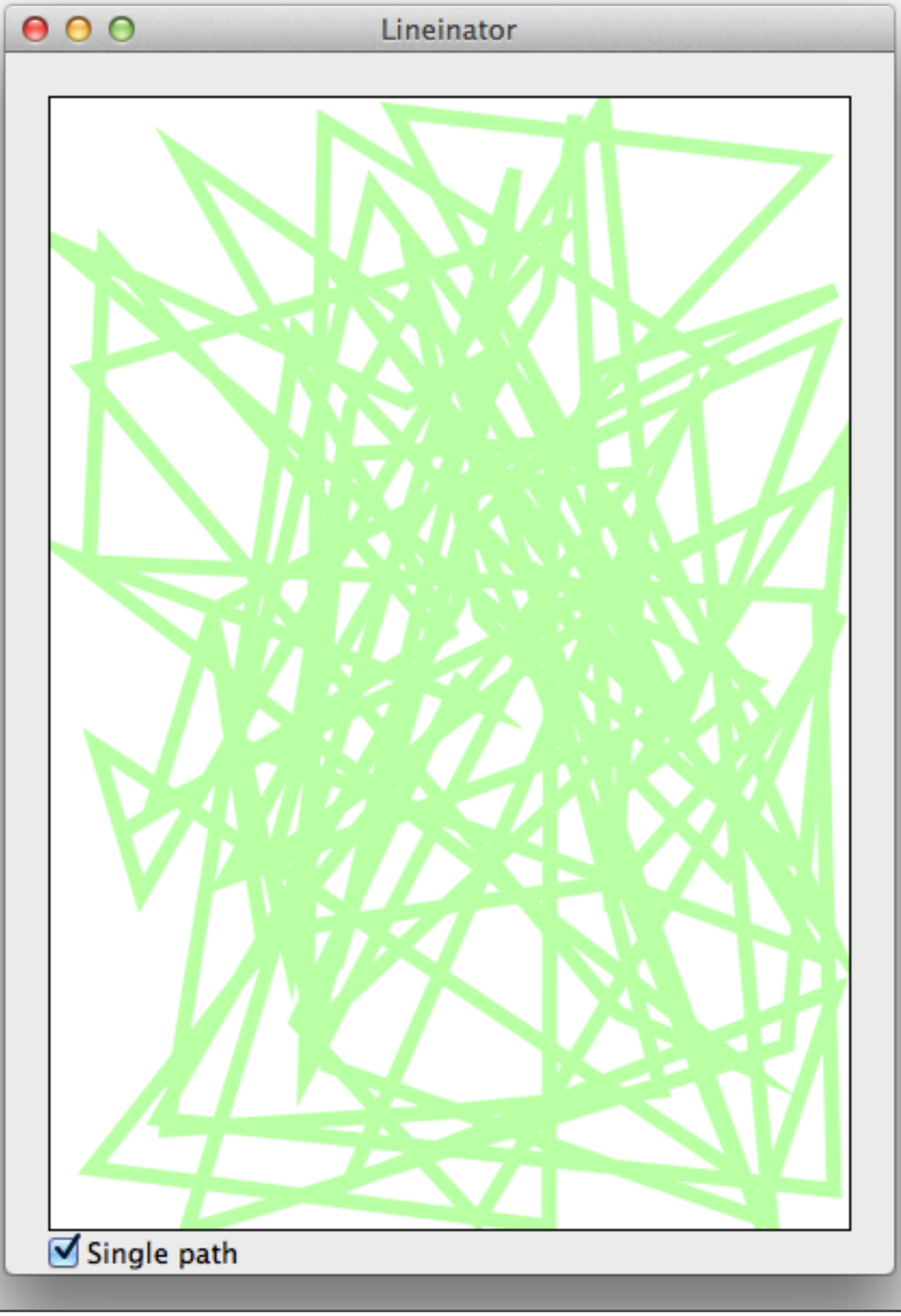
Line Crossings?



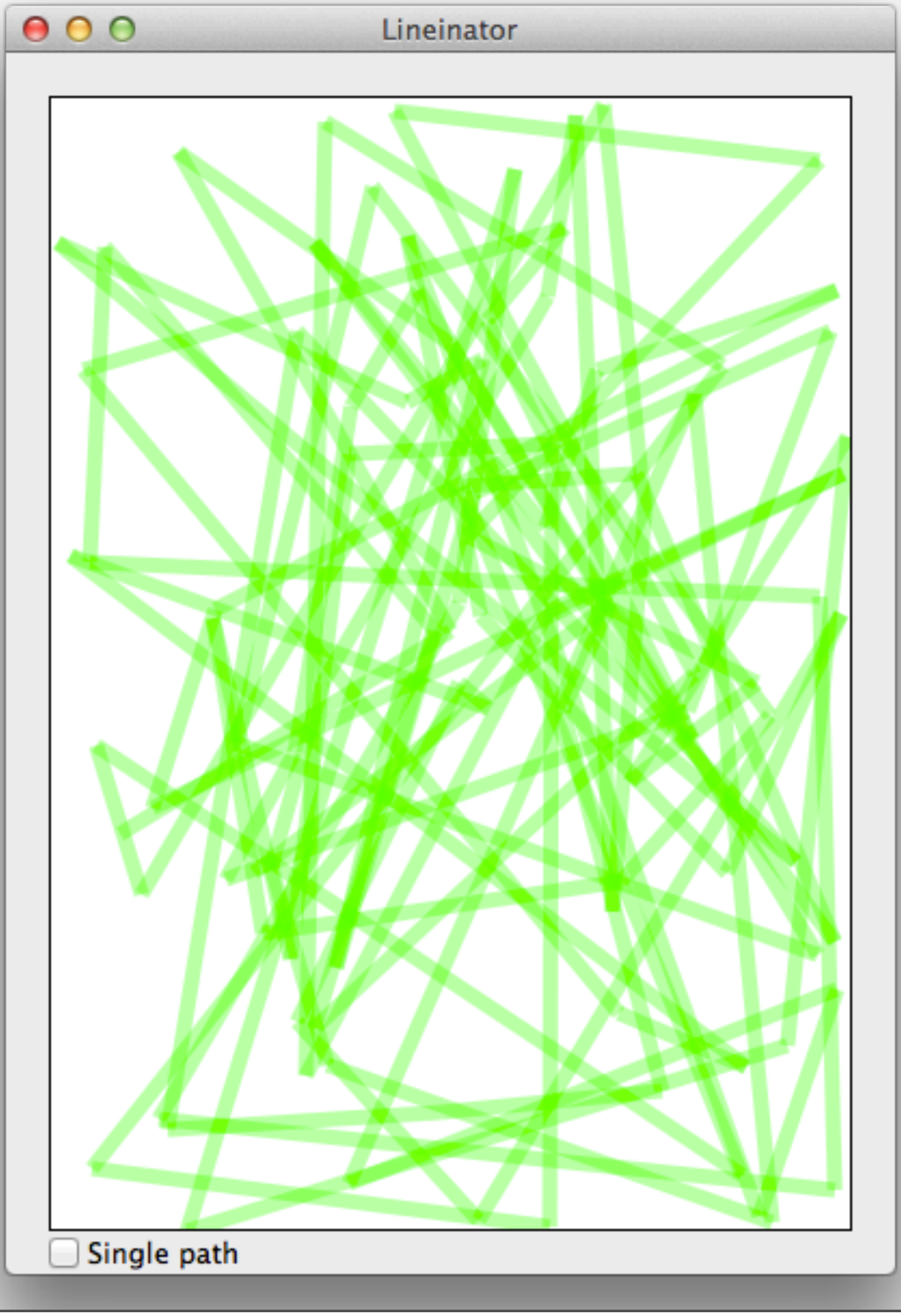
Line Crossings?



Line Crossings!



☑ Single path

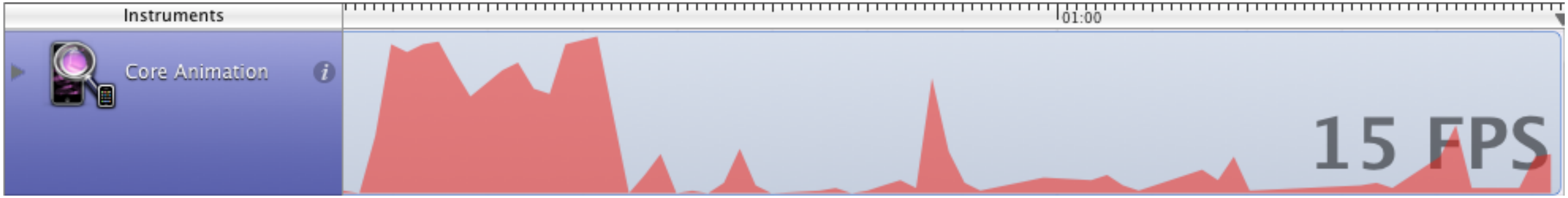


□ Single path

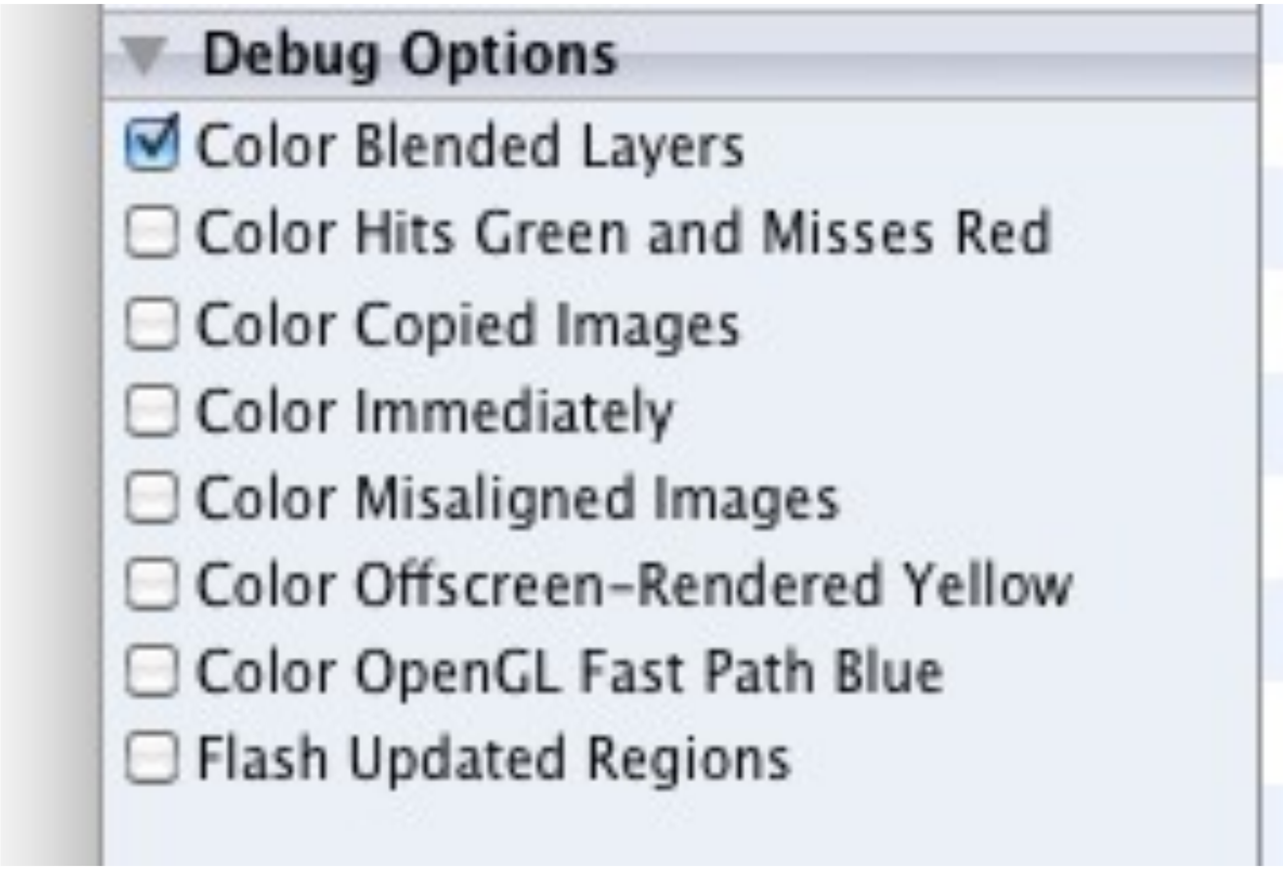
Lines in Motion

(Lineinator)

Core Animation



Core Animation

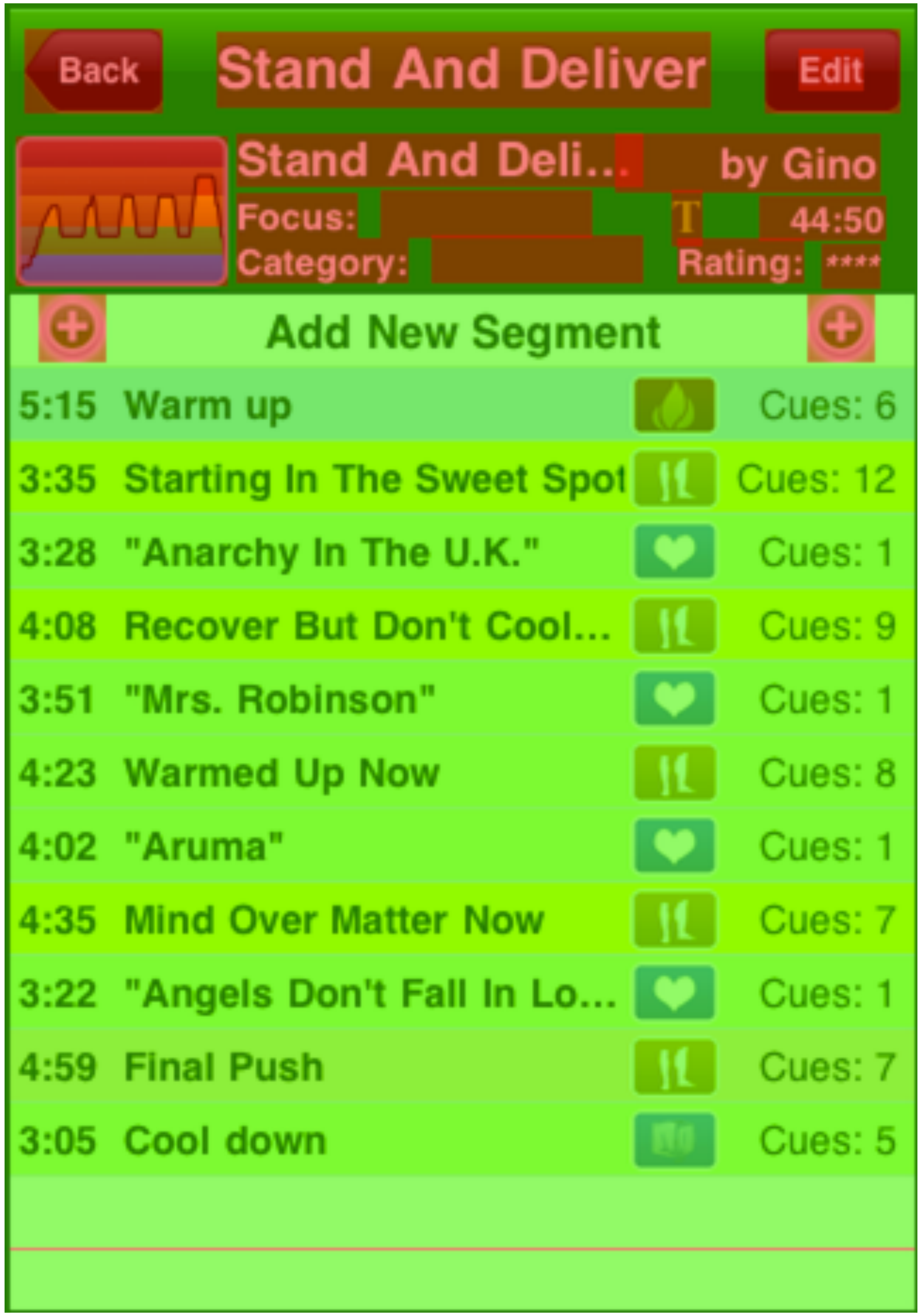


Debug Options

- ☑ Color Blended Layers
- □ Color Hits Green and Misses Red
- □ Color Copied Images
- □ Color Immediately
- □ Color Misaligned Images
- □ Color Offscreen-Rendered Yellow
- □ Color OpenGL Fast Path Blue
- □ Flash Updated Regions

static

scrolling



Back Stand And Deliver Edit

Stand And Deli... by Gino

Focus: T 44:50

Category: Rating: ****

Add New Segment

5:15 Warm up Cues: 6

3:35 Starting In The Sweet Spot Cues: 12

3:28 "Anarchy In The U.K." Cues: 1

4:08 Recover But Don't Cool... Cues: 9

3:51 "Mrs. Robinson" Cues: 1

4:23 Warmed Up Now Cues: 8

4:02 "Aruma" Cues: 1

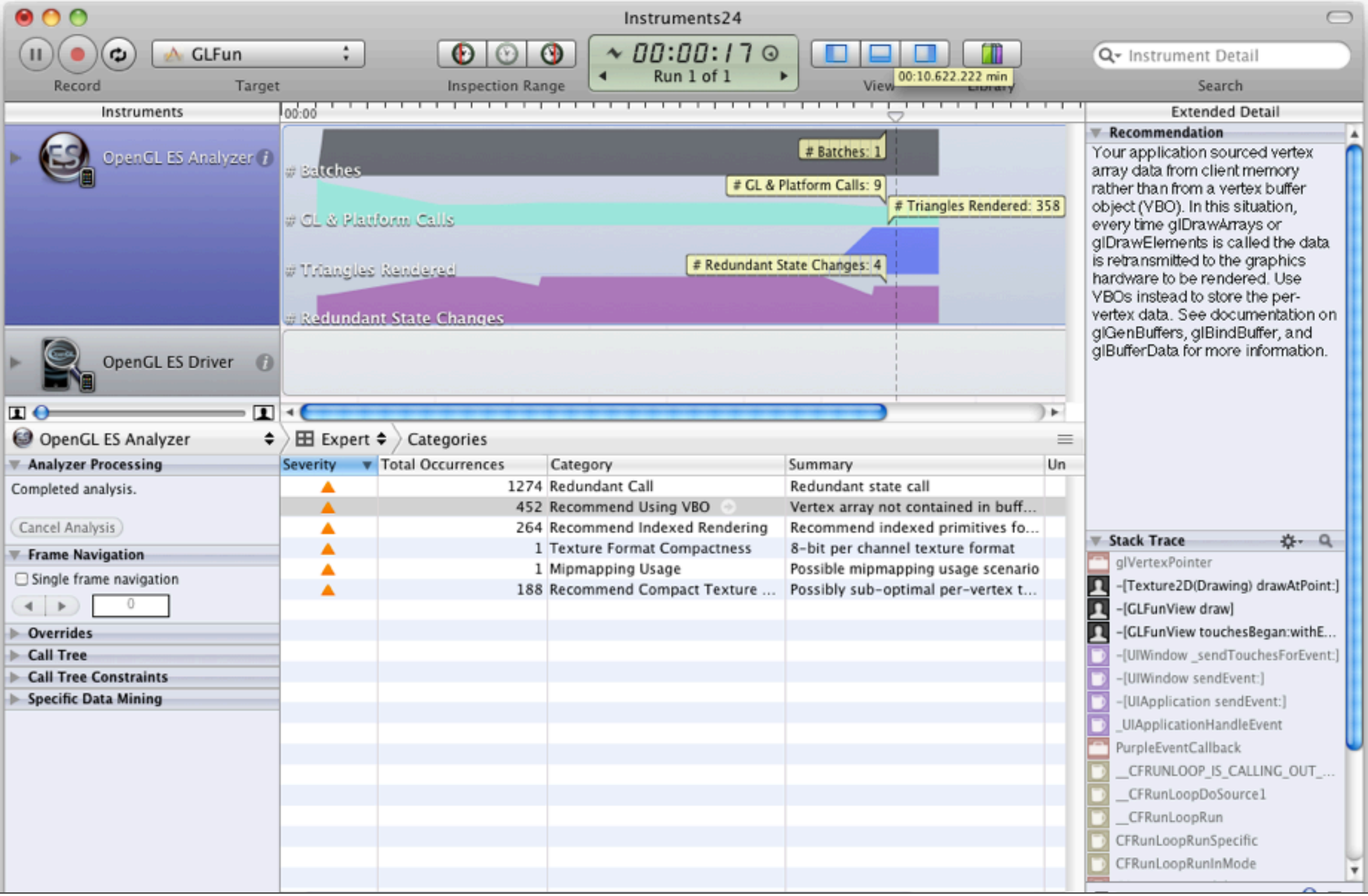
4:35 Mind Over Matter Now Cues: 7

3:22 "Angels Don't Fall In Lo... Cues: 1

4:59 Final Push Cues: 7

3:05 Cool down Cues: 5

OpenGL Analyzer



Completed analysis.

☐ Single frame navigation

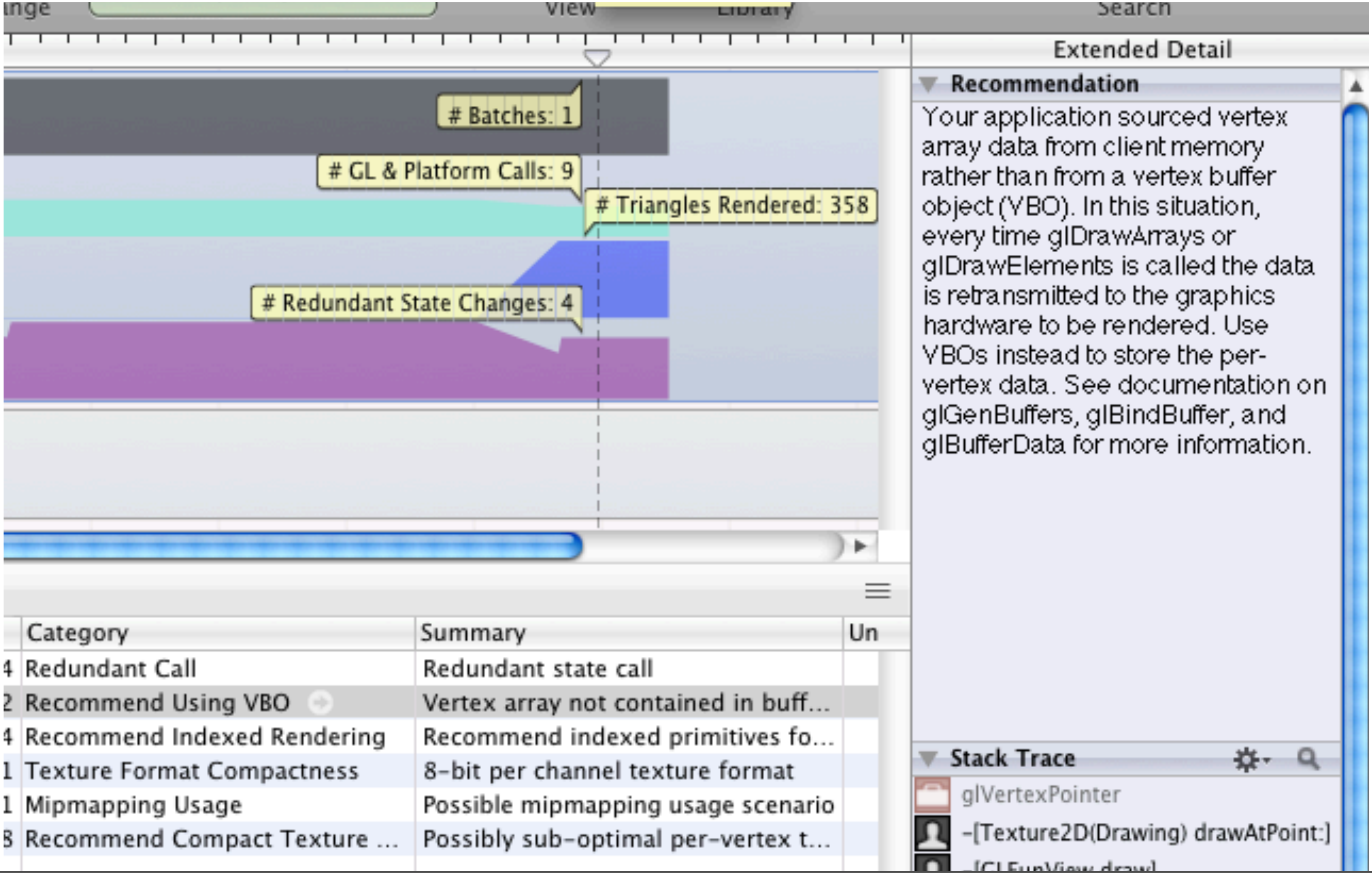
0

Severity	Total Occurrences	Category	Summary	Un
▲	1274	Redundant Call	Redundant state call	
▲	452	Recommend Using VBO	Vertex array not contained in buff...	
▲	264	Recommend Indexed Rendering	Recommend indexed primitives fo...	
▲	1	Texture Format Compactness	8-bit per channel texture format	
▲	1	Mipmapping Usage	Possible mipmapping usage scenario	
▲	188	Recommend Compact Texture ...	Possibly sub-optimal per-vertex t...	

Your application sourced vertex array data from client memory rather than from a vertex buffer object (VBO). In this situation, every time glDrawArrays or glDrawElements is called the data is retransmitted to the graphics hardware to be rendered. Use VBOs instead to store the per-vertex data. See documentation on glGenBuffers, glBindBuffer, and glBufferData for more information.

- glVertexPointer
- -[Texture2D(Drawing) drawAtPoint:]
- -[GLFunView draw]
- -[GLFunView touchesBegan:withE...
- -[UIWindow _sendTouchesForEvent:]
- -[UIWindow sendEvent:]
- -[UIApplication sendEvent:]
- _UIApplicationHandleEvent
- PurpleEventCallback
- __CFRUNLOOP_IS_CALLING_OUT_...
- __CFRunLoopDoSource1
- __CFRunLoopRun
- CFRunLoopRunSpecific
- CFRunLoopRunInMode

OpenGL Analyzer



Recommendation

Your application sourced vertex array data from client memory rather than from a vertex buffer object (VBO). In this situation, every time glDrawArrays or glDrawElements is called the data is retransmitted to the graphics hardware to be rendered. Use VBOs instead to store the per-vertex data. See documentation on glGenBuffers, glBindBuffer, and glBufferData for more information.

	Category	Summary	Un
4	Redundant Call	Redundant state call	
2	Recommend Using VBO	Vertex array not contained in buff...	
4	Recommend Indexed Rendering	Recommend indexed primitives fo...	
1	Texture Format Compactness	8-bit per channel texture format	
1	Mipmapping Usage	Possible mipmapping usage scenario	
8	Recommend Compact Texture ...	Possibly sub-optimal per-vertex t...	

Stack Trace

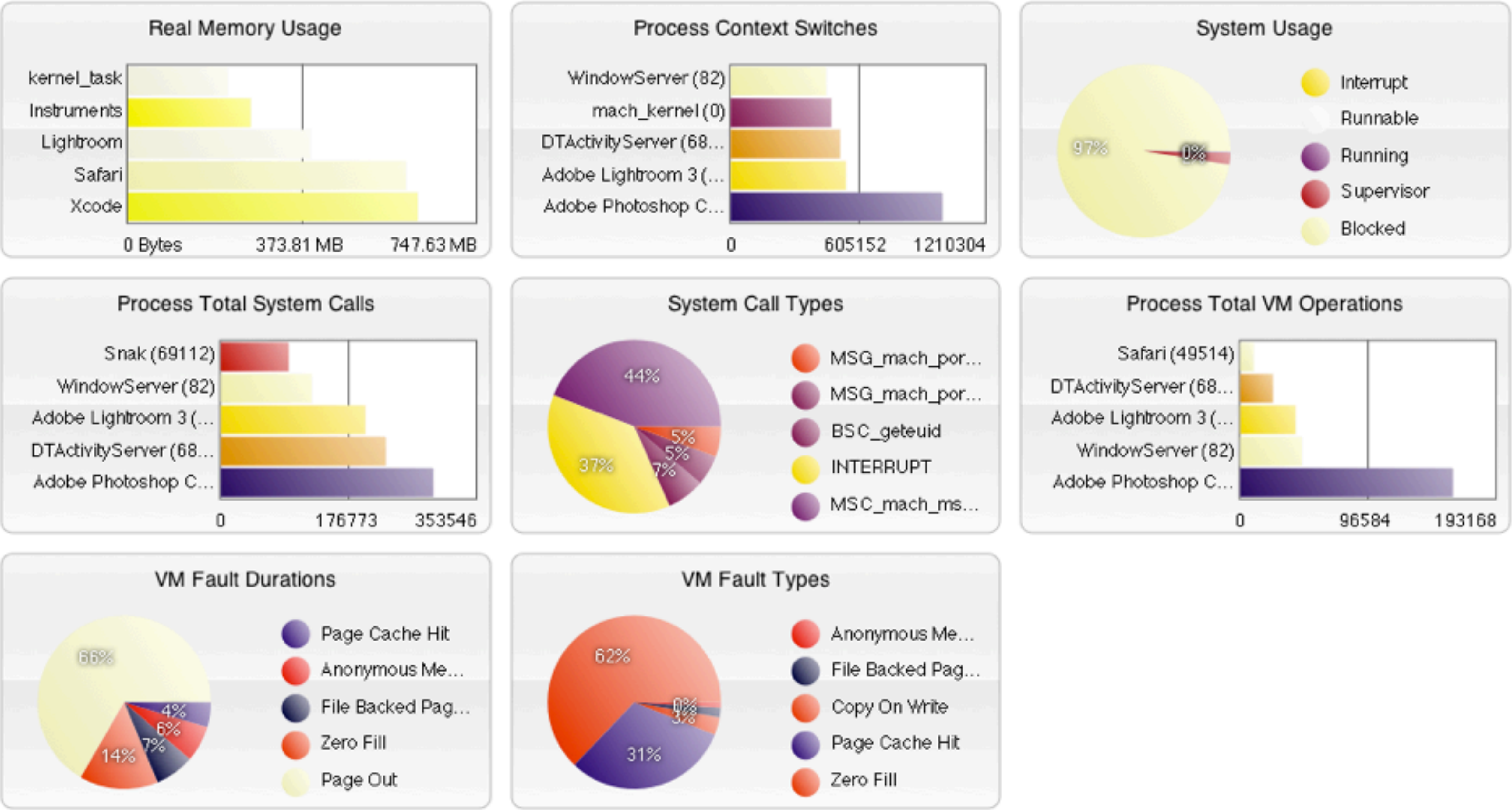
glVertexPointer
-[Texture2D(Drawing) drawAtPoint:]
-[GLFunView draw]

Other Instruments

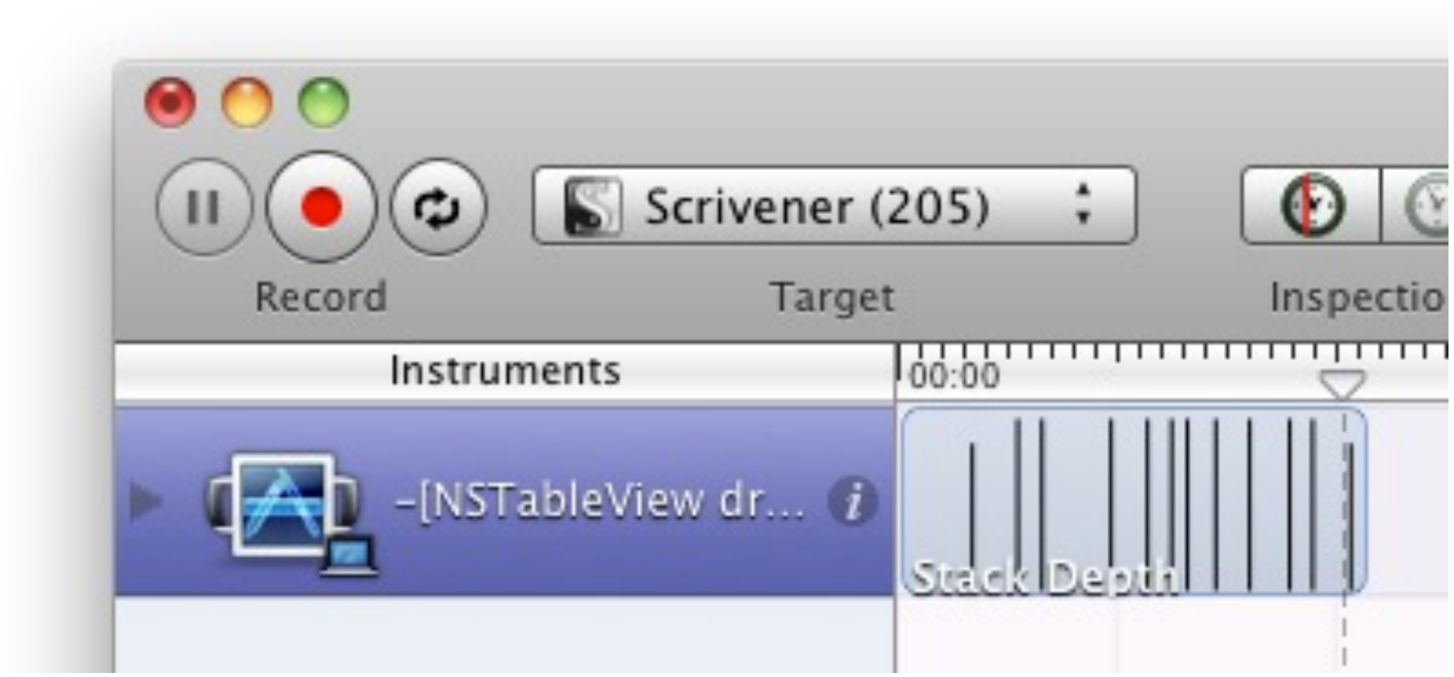
Other Instruments

- Dispatch   (queue lifetimes, invocations)
- ~~Garbage Collection   (scavenging)~~
- Activity Monitor   (lots of metrics)
- Core Data   (fetches / faults / cache misses)

Low-Level tracking

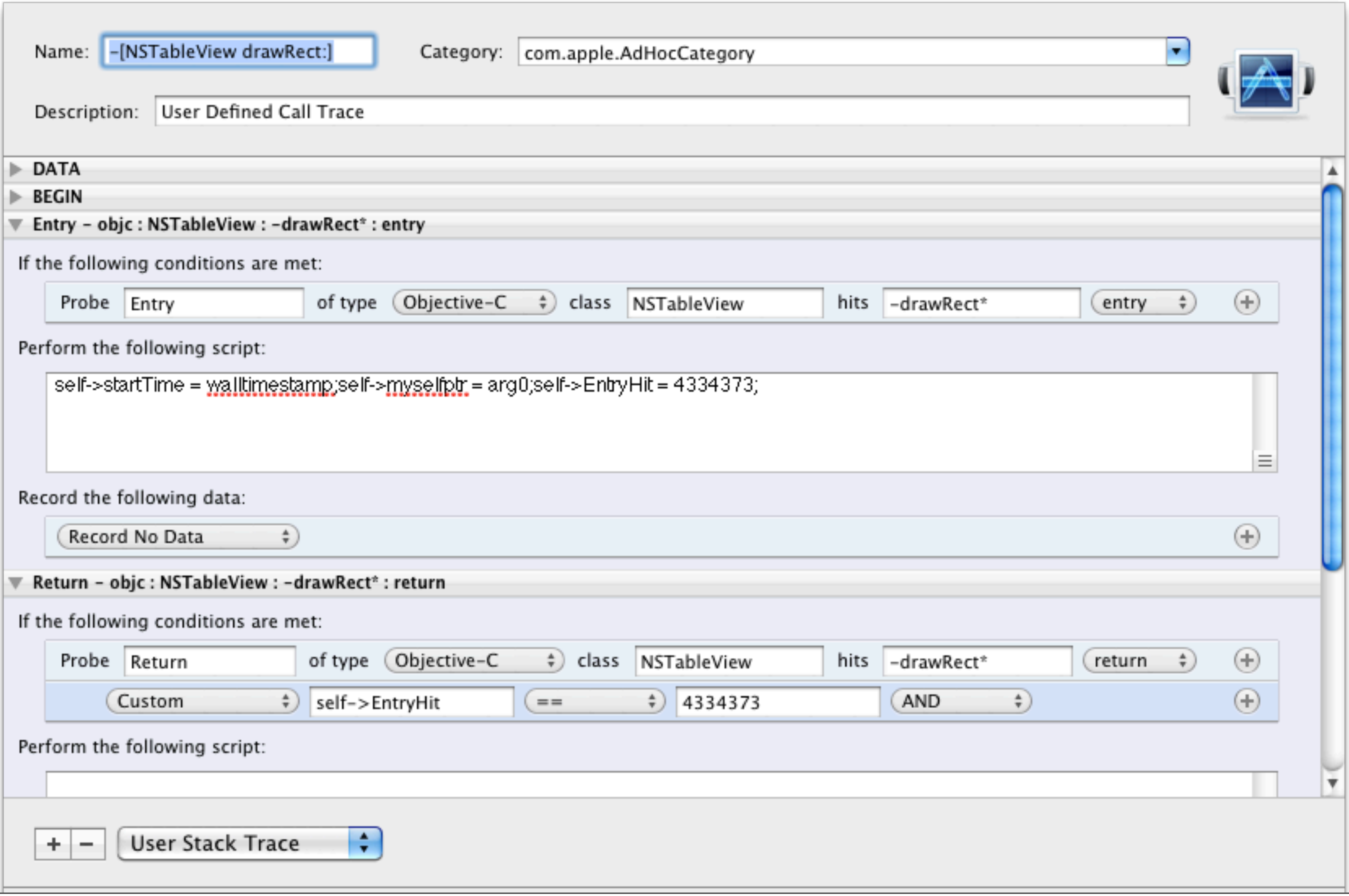


Symbol Trace



-NSTableView drawRect:

DTrace



Name: -[NSTableView drawRect:]

Category: com.apple.AdHocCategory

Description: User Defined Call Trace

Entry - objc : NSTableView : -drawRect* : entry

If the following conditions are met:

Probe Entry of type Objective-C class NSTableView hits -drawRect* entry

Perform the following script:

```
self->startTime = walltimestamp;self->myselfptr = arg0;self->EntryHit = 4334373;
```

Record the following data:

Record No Data

Return - objc : NSTableView : -drawRect* : return

If the following conditions are met:

Probe Return of type Objective-C class NSTableView hits -drawRect* return

Custom self->EntryHit == 4334373 AND

Perform the following script:

Thursday, March 21, 13

Wrap-Up

So What do you do?

- Reduce memory usage
- Change algorithms
  - Reducing a constant can help
- Not doing work
- Take advantage of your hardware
- Code tweaks

Each Situation is Different

- Cache values so you don’t have to recalculate them
  - Recalculate easy to figure out values so you don’t have to store them
- Pre-load stuff from disk
  - Lazy-load stuff from disk
- More small packets for lower latency
  - Fewer big packets for throughput

That’s All Folks

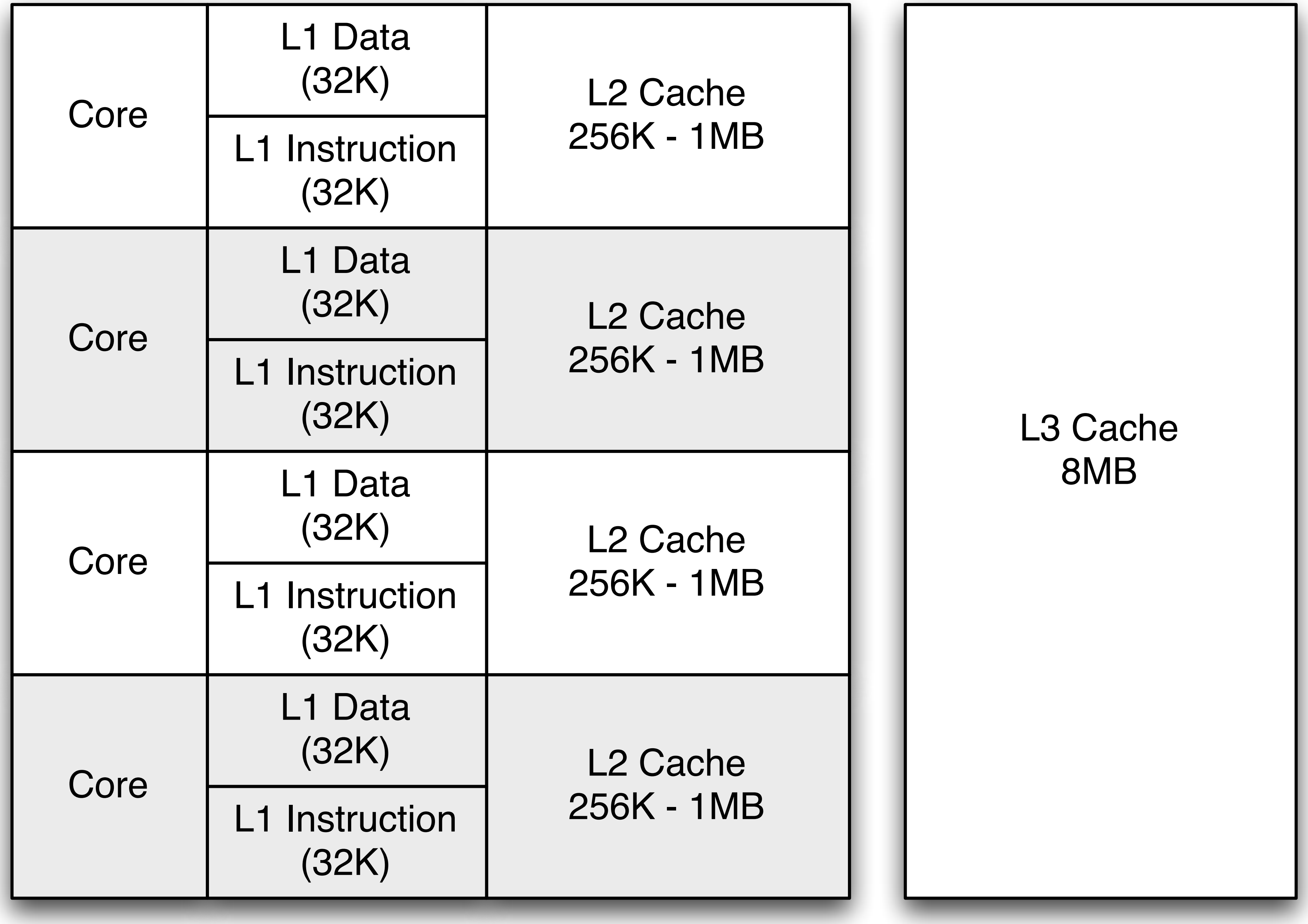
- Discover what’s slow
- Figure out why it’s slow
- Fix it

@borkware

http://borkware.com/cocoaconf

Holding Pen

Cache And Carry



Core

L1 Data (32K)

L1 Instruction (32K)

L2 Cache 256K - 1MB

Core

L1 Data (32K)

L1 Instruction (32K)

L2 Cache 256K - 1MB

Core

L1 Data (32K)

L1 Instruction (32K)

L2 Cache 256K - 1MB

Core

L1 Data (32K)

L1 Instruction (32K)

L2 Cache 256K - 1MB

L3 Cache 8MB

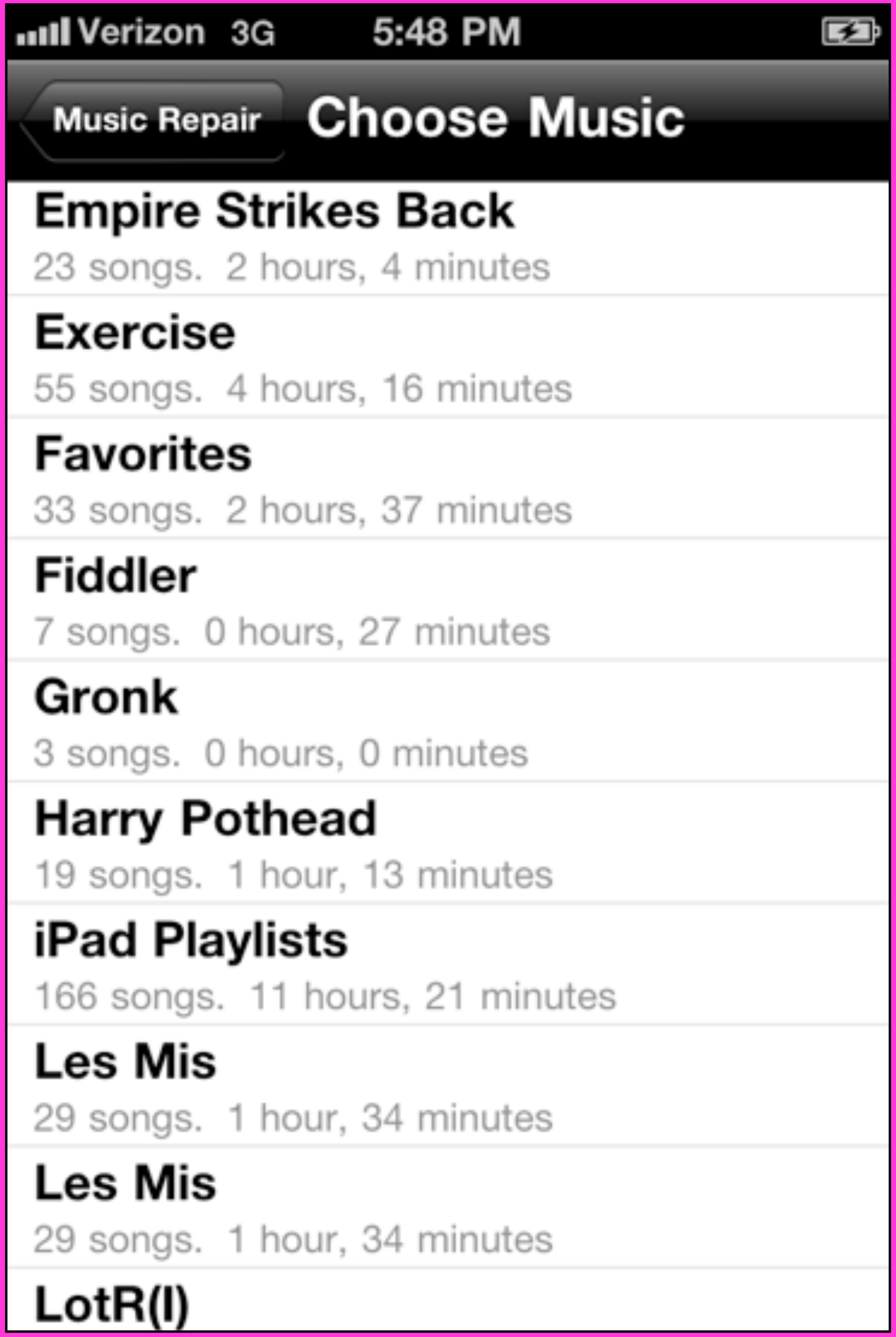
L1 cache reference: 0.5 ns

L2 cache reference: 7 ns

Main memory reference: 100 ns

The Tale of Woe

- I needed to choose Playlists
- Need to know playlist duration
- You don't get that info directly from MPMediaFooby



Empire Strikes Back
23 songs. 2 hours, 4 minutes

Exercise
55 songs. 4 hours, 16 minutes

Favorites
33 songs. 2 hours, 37 minutes

Fiddler
7 songs. 0 hours, 27 minutes

Gronk
3 songs. 0 hours, 0 minutes

Harry Pothead
19 songs. 1 hour, 13 minutes

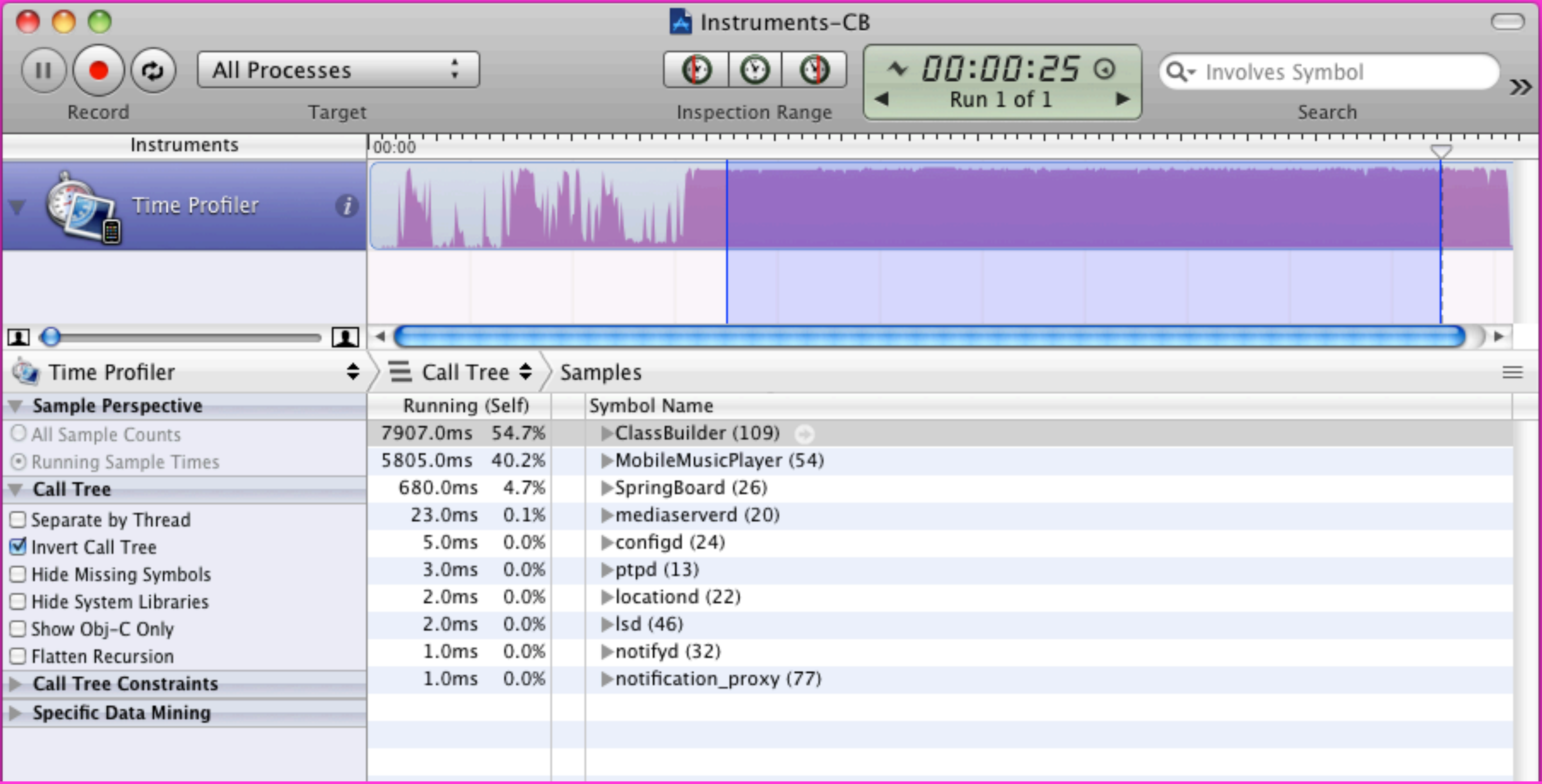
iPad Playlists
166 songs. 11 hours, 21 minutes

Les Mis
29 songs. 1 hour, 34 minutes

Les Mis
29 songs. 1 hour, 34 minutes

LotR(I)

Time Profiler



Time Profiler

▼ Sample Perspective

○ All Sample Counts

● Running Sample Times

▼ Call Tree

□ Separate by Thread

☑ Invert Call Tree

□ Hide Missing Symbols

□ Hide System Libraries

□ Show Obj-C Only

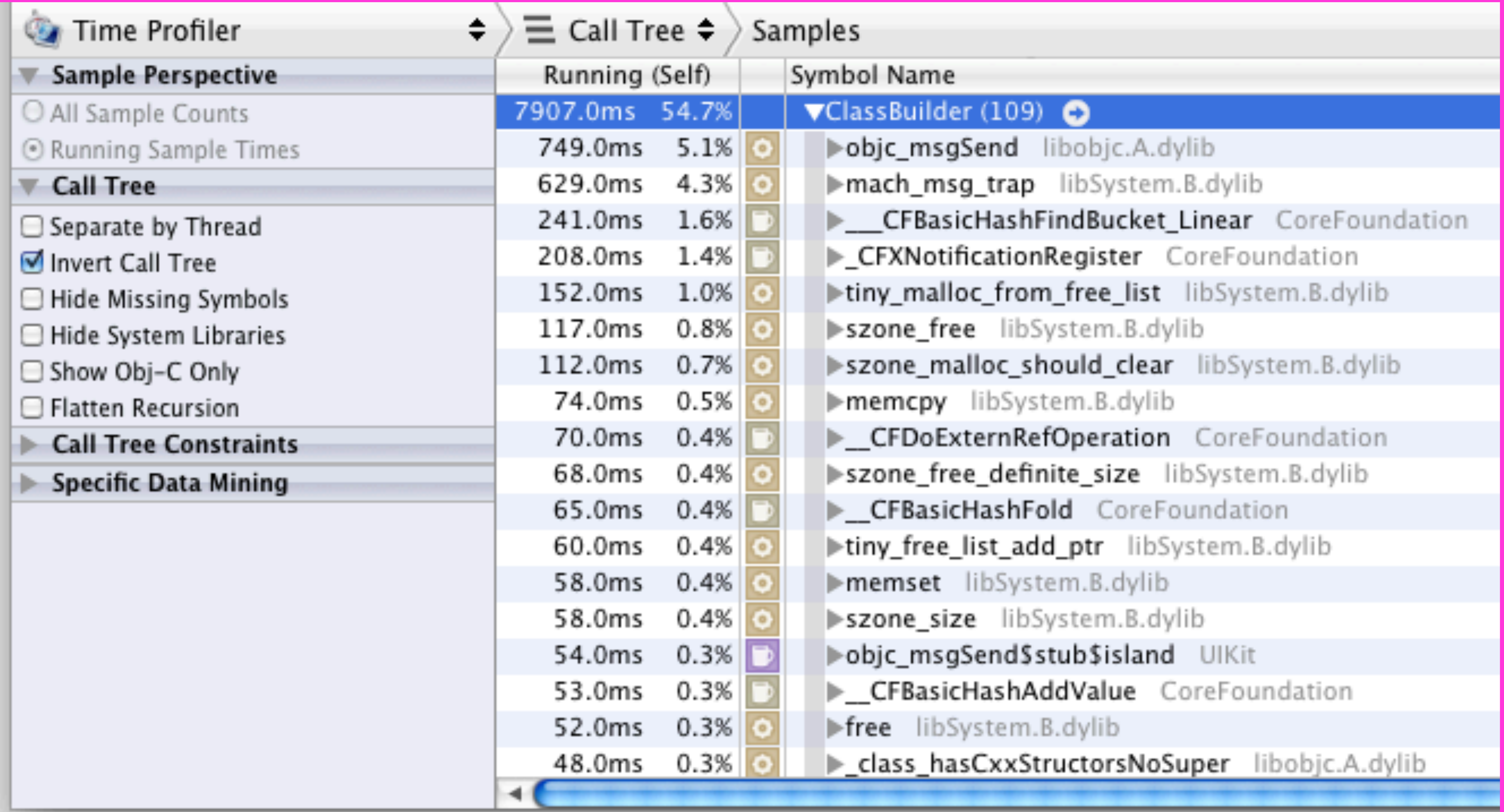
□ Flatten Recursion

▶ Call Tree Constraints

▶ Specific Data Mining

Running (Self)		Symbol Name
7907.0ms	54.7%	▶ClassBuilder (109)
5805.0ms	40.2%	▶MobileMusicPlayer (54)
680.0ms	4.7%	▶SpringBoard (26)
23.0ms	0.1%	▶mediaserverd (20)
5.0ms	0.0%	▶configd (24)
3.0ms	0.0%	▶ptpd (13)
2.0ms	0.0%	▶locationd (22)
2.0ms	0.0%	▶lsd (46)
1.0ms	0.0%	▶notifyd (32)
1.0ms	0.0%	▶notification_proxy (77)

Call Tree



Time Profiler ⇕ 〉 Call Tree ⇕ 〉 Samples

Sample Perspective

○ All Sample Counts

● Running Sample Times

Call Tree

☐ Separate by Thread

☑ Invert Call Tree

☐ Hide Missing Symbols

☐ Hide System Libraries

☐ Show Obj-C Only

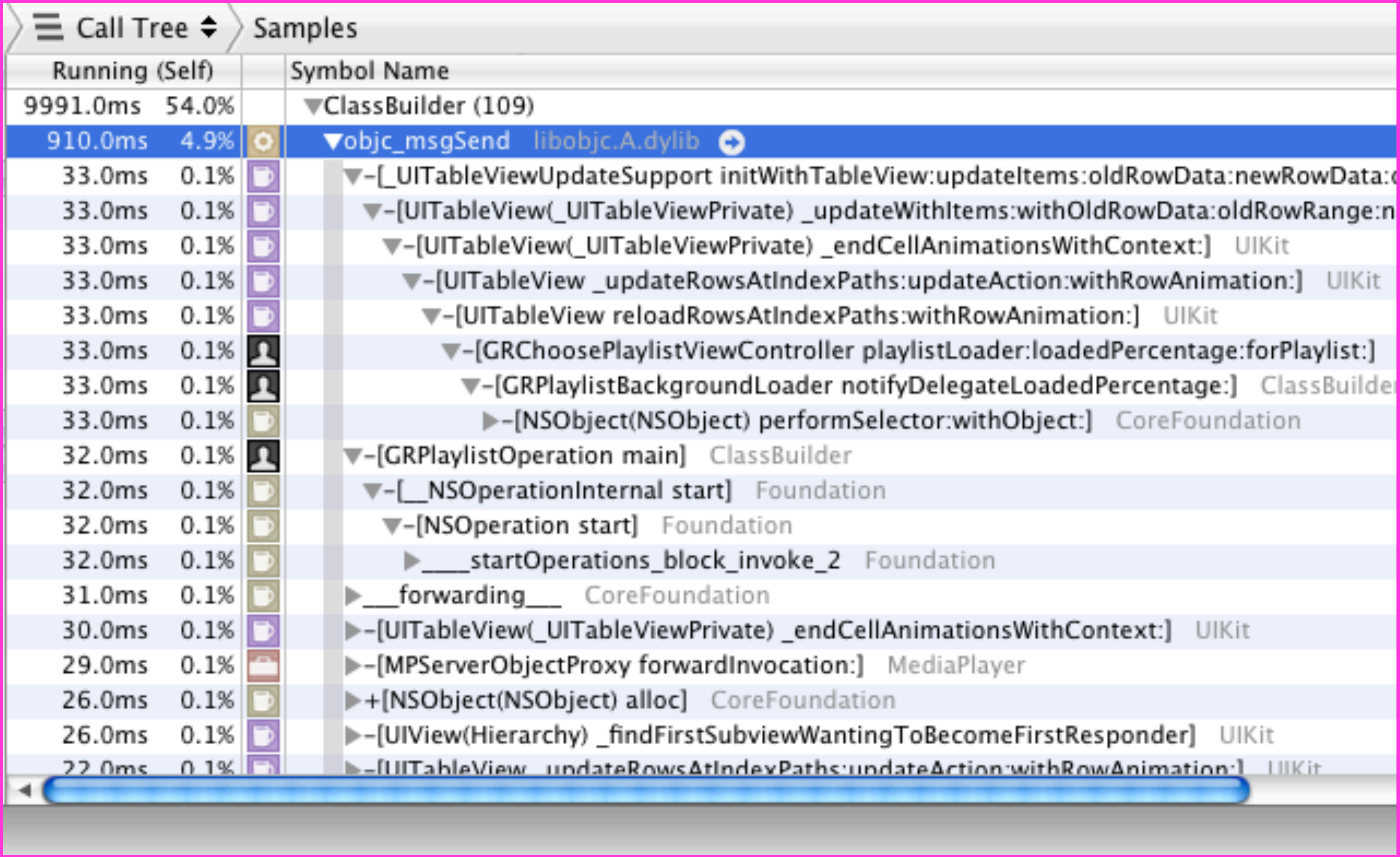
☐ Flatten Recursion

Call Tree Constraints

Specific Data Mining

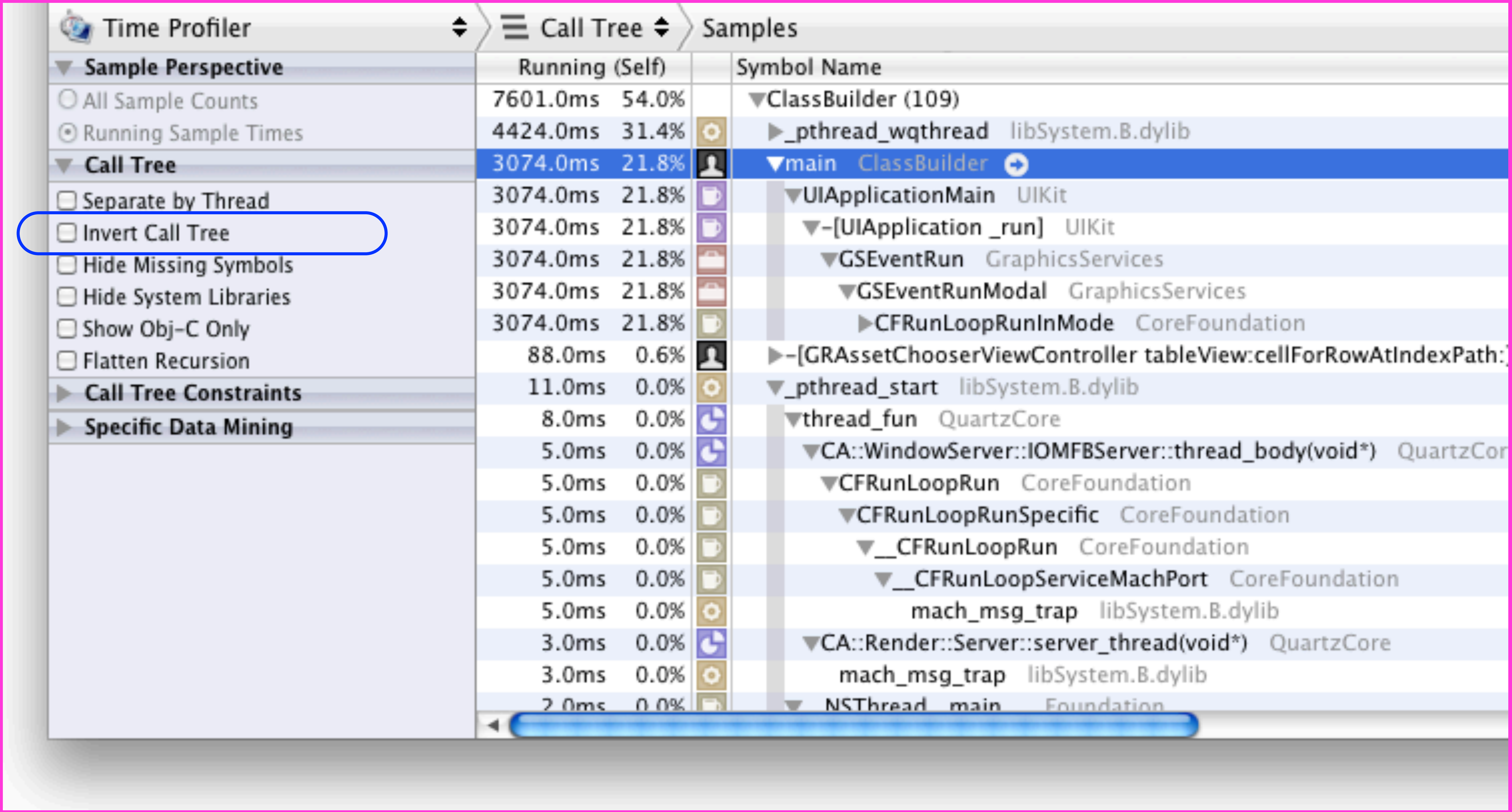
Running (Self)		Symbol Name	
7907.0ms	54.7%	▼ClassBuilder (109)	
749.0ms	5.1%	▶objc_msgSend	libobjc.A.dylib
629.0ms	4.3%	▶mach_msg_trap	libSystem.B.dylib
241.0ms	1.6%	▶___CFBasicHashFindBucket_Linear	CoreFoundation
208.0ms	1.4%	▶_CFXNotificationRegister	CoreFoundation
152.0ms	1.0%	▶tiny_malloc_from_free_list	libSystem.B.dylib
117.0ms	0.8%	▶szone_free	libSystem.B.dylib
112.0ms	0.7%	▶szone_malloc_should_clear	libSystem.B.dylib
74.0ms	0.5%	▶memcpy	libSystem.B.dylib
70.0ms	0.4%	▶__CFDoExternRefOperation	CoreFoundation
68.0ms	0.4%	▶szone_free_definite_size	libSystem.B.dylib
65.0ms	0.4%	▶__CFBasicHashFold	CoreFoundation
60.0ms	0.4%	▶tiny_free_list_add_ptr	libSystem.B.dylib
58.0ms	0.4%	▶memset	libSystem.B.dylib
58.0ms	0.4%	▶szone_size	libSystem.B.dylib
54.0ms	0.3%	▶objc_msgSend$stub$island	UIKit
53.0ms	0.3%	▶__CFBasicHashAddValue	CoreFoundation
52.0ms	0.3%	▶free	libSystem.B.dylib
48.0ms	0.3%	▶_class_hasCxxStructorsNoSuper	libobjc.A.dylib

Call Tree



```
Running (Self)          Symbol Name
9991.0ms  54.0%         ▼ClassBuilder (109)
 910.0ms   4.9%          ▼objc_msgSend  libobjc.A.dylib ➜
  33.0ms   0.1%            ▼-[_UITableViewUpdateSupport initWithTableView:updateItems:oldRowData:newRowData:
  33.0ms   0.1%              ▼-[UITableView(_UITableViewPrivate) _updateWithItems:withOldRowData:oldRowRange:n
  33.0ms   0.1%                ▼-[UITableView(_UITableViewPrivate) _endCellAnimationsWithContext:]  UIKit
  33.0ms   0.1%                  ▼-[UITableView _updateRowsAtIndexPaths:updateAction:withRowAnimation:]  UIKit
  33.0ms   0.1%                    ▼-[UITableView reloadRowsAtIndexPaths:withRowAnimation:]  UIKit
  33.0ms   0.1%  👤                  ▼-[GRChoosePlaylistViewController playlistLoader:loadedPercentage:forPlaylist:]
  33.0ms   0.1%  👤                    ▼-[GRPlaylistBackgroundLoader notifyDelegateLoadedPercentage:]  ClassBuilde
  33.0ms   0.1%                          ▶-[NSObject(NSObject) performSelector:withObject:]  CoreFoundation
  32.0ms   0.1%  👤        ▼-[GRPlaylistOperation main]  ClassBuilder
  32.0ms   0.1%              ▼-[__NSOperationInternal start]  Foundation
  32.0ms   0.1%                ▼-[NSOperation start]  Foundation
  32.0ms   0.1%                  ▶____startOperations_block_invoke_2  Foundation
  31.0ms   0.1%            ▶___forwarding___  CoreFoundation
  30.0ms   0.1%            ▶-[UITableView(_UITableViewPrivate) _endCellAnimationsWithContext:]  UIKit
  29.0ms   0.1%            ▶-[MPServerObjectProxy forwardInvocation:]  MediaPlayer
  26.0ms   0.1%            ▶+[NSObject(NSObject) alloc]  CoreFoundation
  26.0ms   0.1%            ▶-[UIView(Hierarchy) _findFirstSubviewWantingToBecomeFirstResponder]  UIKit
  22.0ms   0.1%            ▶-[UITableView _updateRowsAtIndexPaths:updateAction:withRowAnimation:]  UIKit
```

Uninverted Tree



▼ Sample Perspective

○ All Sample Counts

● Running Sample Times

▼ Call Tree

□ Separate by Thread

□ Invert Call Tree

□ Hide Missing Symbols

□ Hide System Libraries

□ Show Obj-C Only

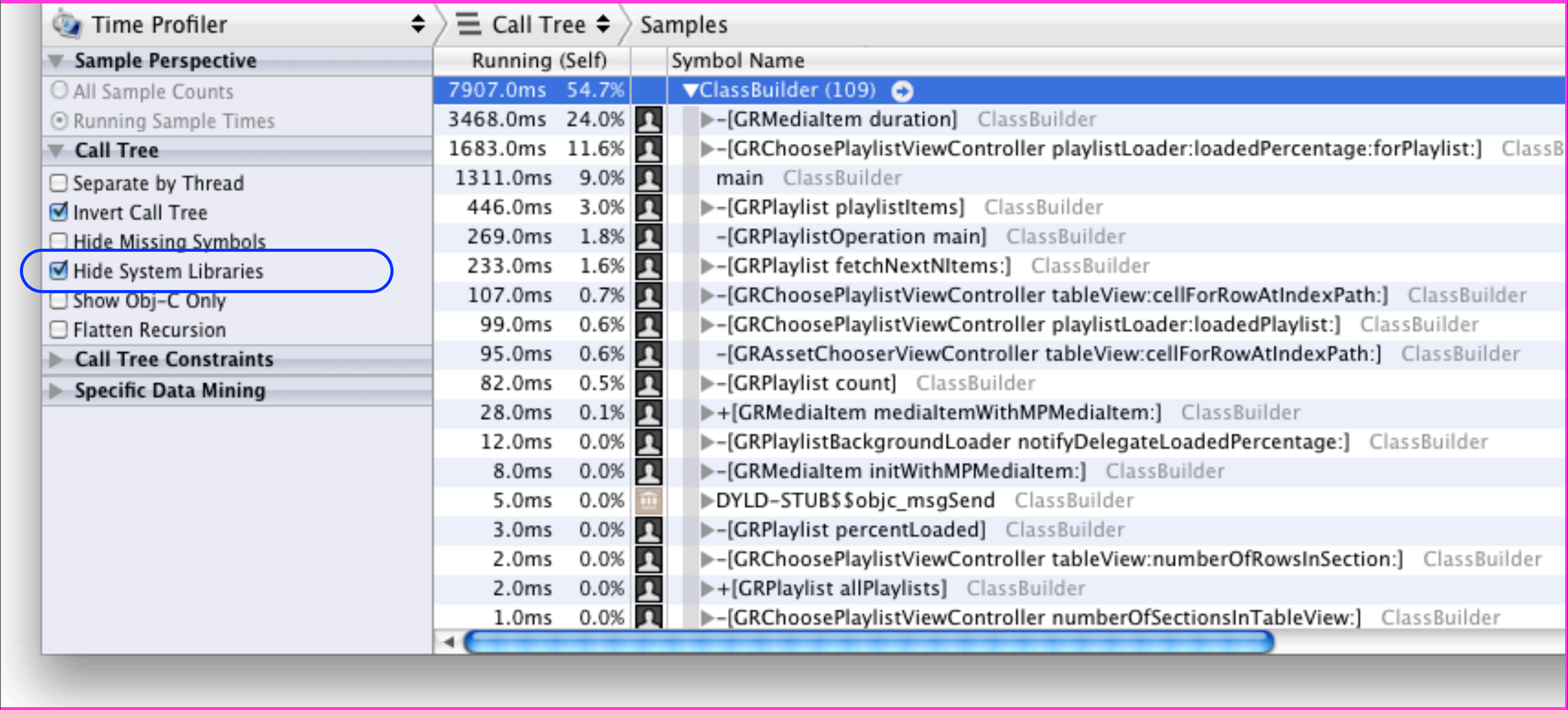
□ Flatten Recursion

▶ Call Tree Constraints

▶ Specific Data Mining

```
Running (Self)     Symbol Name
7601.0ms  54.0%    ▼ClassBuilder (109)
4424.0ms  31.4%      ▶_pthread_wqthread  libSystem.B.dylib
3074.0ms  21.8%      ▼main  ClassBuilder
3074.0ms  21.8%        ▼UIApplicationMain  UIKit
3074.0ms  21.8%          ▼-[UIApplication _run]  UIKit
3074.0ms  21.8%            ▼GSEventRun  GraphicsServices
3074.0ms  21.8%              ▼GSEventRunModal  GraphicsServices
3074.0ms  21.8%                ▶CFRunLoopRunInMode  CoreFoundation
  88.0ms   0.6%      ▶-[GRAssetChooserViewController tableView:cellForRowAtIndexPath:
  11.0ms   0.0%      ▼_pthread_start  libSystem.B.dylib
   8.0ms   0.0%        ▼thread_fun  QuartzCore
   5.0ms   0.0%          ▼CA::WindowServer::IOMFBServer::thread_body(void*)  QuartzCor
   5.0ms   0.0%            ▼CFRunLoopRun  CoreFoundation
   5.0ms   0.0%              ▼CFRunLoopRunSpecific  CoreFoundation
   5.0ms   0.0%                ▼__CFRunLoopRun  CoreFoundation
   5.0ms   0.0%                  ▼__CFRunLoopServiceMachPort  CoreFoundation
   5.0ms   0.0%                      mach_msg_trap  libSystem.B.dylib
   3.0ms   0.0%          ▼CA::Render::Server::server_thread(void*)  QuartzCore
   3.0ms   0.0%              mach_msg_trap  libSystem.B.dylib
   2.0ms   0.0%        ▼ NSThread__main  Foundation
```

Data Mining : Hiding Syslibs



Sample Perspective

○ All Sample Counts
● Running Sample Times

Call Tree

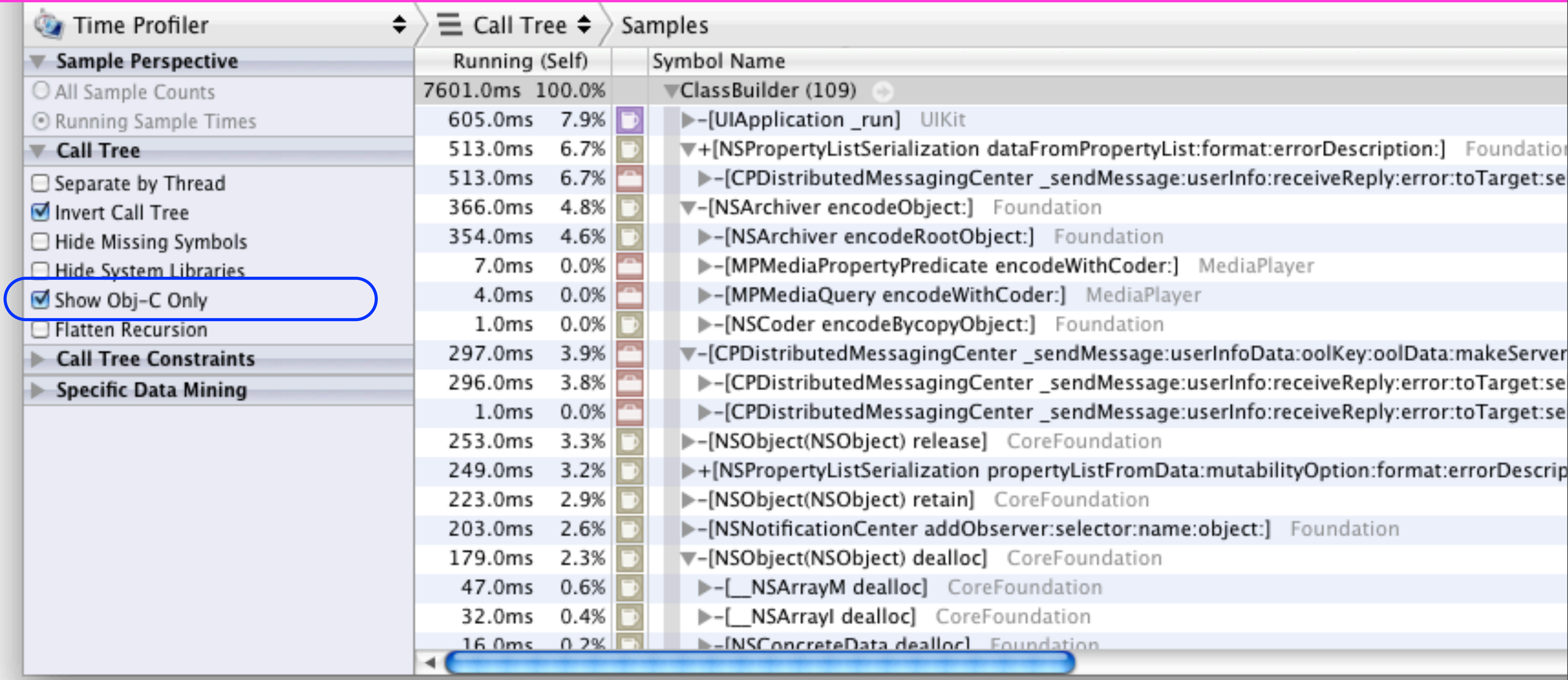
□ Separate by Thread
☑ Invert Call Tree
□ Hide Missing Symbols
☑ Hide System Libraries
□ Show Obj-C Only
□ Flatten Recursion

Call Tree Constraints

Specific Data Mining

Running (Self)			Symbol Name
7907.0ms	54.7%		▼ClassBuilder (109)
3468.0ms	24.0%	👤	▶-[GRMediaItem duration] ClassBuilder
1683.0ms	11.6%	👤	▶-[GRChoosePlaylistViewController playlistLoader:loadedPercentage:forPlaylist:] ClassB
1311.0ms	9.0%	👤	main ClassBuilder
446.0ms	3.0%	👤	▶-[GRPlaylist playlistItems] ClassBuilder
269.0ms	1.8%	👤	-[GRPlaylistOperation main] ClassBuilder
233.0ms	1.6%	👤	▶-[GRPlaylist fetchNextNItems:] ClassBuilder
107.0ms	0.7%	👤	▶-[GRChoosePlaylistViewController tableView:cellForRowAtIndexPath:] ClassBuilder
99.0ms	0.6%	👤	▶-[GRChoosePlaylistViewController playlistLoader:loadedPlaylist:] ClassBuilder
95.0ms	0.6%	👤	-[GRAssetChooserViewController tableView:cellForRowAtIndexPath:] ClassBuilder
82.0ms	0.5%	👤	▶-[GRPlaylist count] ClassBuilder
28.0ms	0.1%	👤	▶+[GRMediaItem mediaItemWithMPMediaItem:] ClassBuilder
12.0ms	0.0%	👤	▶-[GRPlaylistBackgroundLoader notifyDelegateLoadedPercentage:] ClassBuilder
8.0ms	0.0%	👤	▶-[GRMediaItem initWithMPMediaItem:] ClassBuilder
5.0ms	0.0%	🏛	▶DYLD-STUB$$objc_msgSend ClassBuilder
3.0ms	0.0%	👤	▶-[GRPlaylist percentLoaded] ClassBuilder
2.0ms	0.0%	👤	▶-[GRChoosePlaylistViewController tableView:numberOfRowsInSection:] ClassBuilder
2.0ms	0.0%	👤	▶+[GRPlaylist allPlaylists] ClassBuilder
1.0ms	0.0%	👤	▶-[GRChoosePlaylistViewController numberOfSectionsInTableView:] ClassBuilder

Data Mining : Objective-C



Sample Perspective

○ All Sample Counts
● Running Sample Times

Call Tree

□ Separate by Thread
☑ Invert Call Tree
□ Hide Missing Symbols
□ Hide System Libraries
☑ Show Obj-C Only
□ Flatten Recursion

Call Tree Constraints

Specific Data Mining

Running (Self)		Symbol Name
7601.0ms	100.0%	▼ClassBuilder (109)
605.0ms	7.9%	▶-[UIApplication _run] UIKit
513.0ms	6.7%	▼+[NSPropertyListSerialization dataFromPropertyList:format:errorDescription:] Foundatio
513.0ms	6.7%	▶-[CPDistributedMessagingCenter _sendMessage:userInfo:receiveReply:error:toTarget:se
366.0ms	4.8%	▼-[NSArchiver encodeObject:] Foundation
354.0ms	4.6%	▶-[NSArchiver encodeRootObject:] Foundation
7.0ms	0.0%	▶-[MPMediaPropertyPredicate encodeWithCoder:] MediaPlayer
4.0ms	0.0%	▶-[MPMediaQuery encodeWithCoder:] MediaPlayer
1.0ms	0.0%	▶-[NSCoder encodeBycopyObject:] Foundation
297.0ms	3.9%	▼-[CPDistributedMessagingCenter _sendMessage:userInfoData:oolKey:oolData:makeServer
296.0ms	3.8%	▶-[CPDistributedMessagingCenter _sendMessage:userInfo:receiveReply:error:toTarget:se
1.0ms	0.0%	▶-[CPDistributedMessagingCenter _sendMessage:userInfo:receiveReply:error:toTarget:se
253.0ms	3.3%	▶-[NSObject(NSObject) release] CoreFoundation
249.0ms	3.2%	▶+[NSPropertyListSerialization propertyListFromData:mutabilityOption:format:errorDescrip
223.0ms	2.9%	▶-[NSObject(NSObject) retain] CoreFoundation
203.0ms	2.6%	▶-[NSNotificationCenter addObserver:selector:name:object:] Foundation
179.0ms	2.3%	▼-[NSObject(NSObject) dealloc] CoreFoundation
47.0ms	0.6%	▶-[__NSArrayM dealloc] CoreFoundation
32.0ms	0.4%	▶-[__NSArrayI dealloc] CoreFoundation
16.0ms	0.2%	▶-[NSConcreteData dealloc] Foundation