Praktikum Entwicklung von Mediensystemen mit iOS

SS 2011

Michael Rohs
michael.rohs@ifi.lmu.de
MHCI Lab, LMU München

Milestones

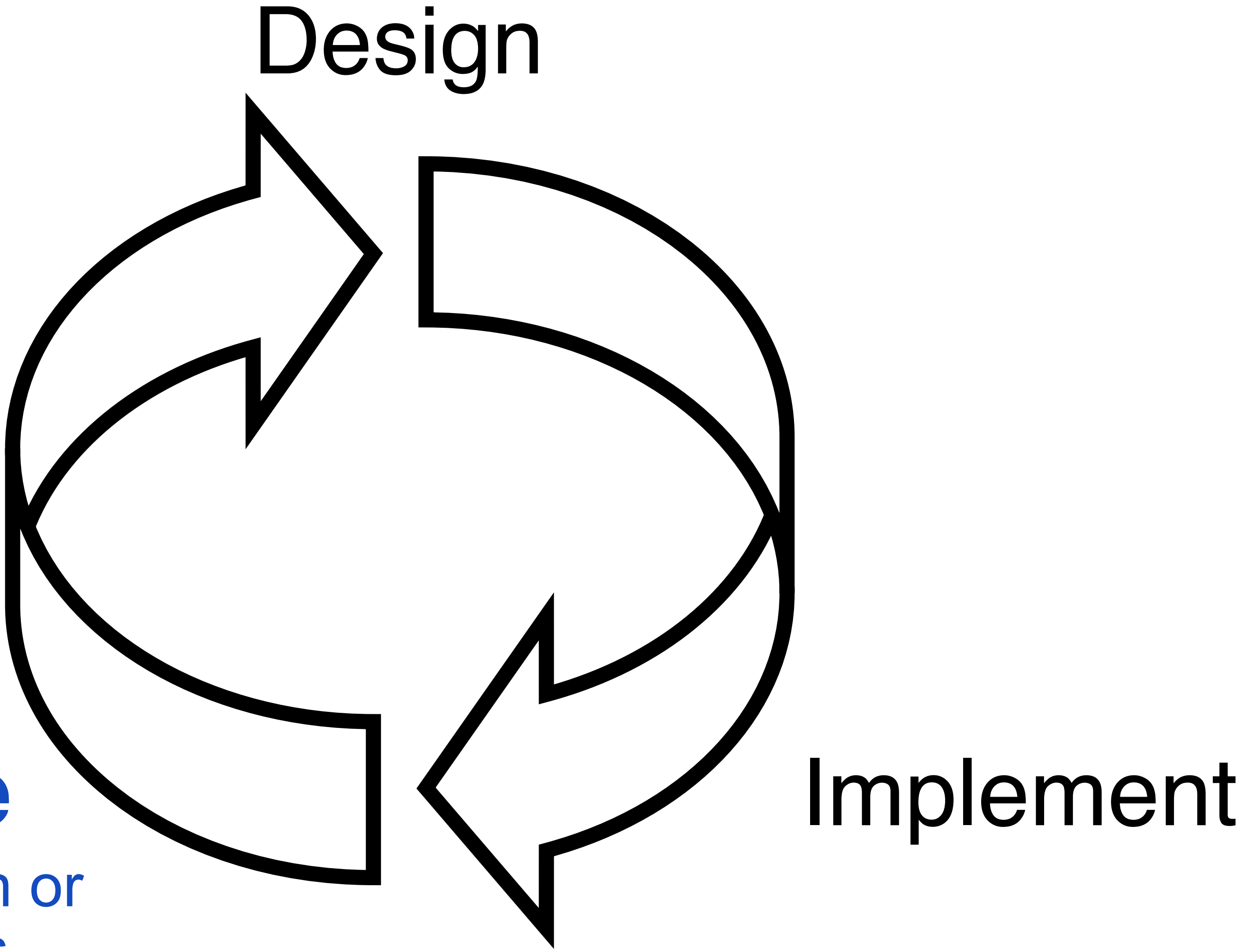
- 26.5.
  - Project definition, brainstorming, main functions, persona
- 9.6. (week 1)
  - Identify user needs (interview or observation)
  - Storyboarding, low fidelity paper prototyping
- 16.6. (weeks 2,3)
  - Test paper prototype with users
  - Start of software prototype development
- 30.6. (week 4)
  - Heuristic evaluation of software prototype
- 7.7. (weeks 5,6)
  - Think-aloud user study on software prototype
- 21.7. (week 7)
  - Completion of software prototype, preparation of presentation
- 28.7.
  - Presentation of project results

Tasks

- Present milestone results at meetings
- Meet with your group regularly
- 9.6.
  - Present project idea, present persona, narrow down functionality
- 16.6.
  - Present interview results, storyboard, first paper prototype
- 30.6.
  - Present paper prototype test results (and plan for revision)
- 7.7.
  - Present results of heuristic evaluation (and plan for revision)
- 21.7. ⟵ today
  - Present results of think-aloud user study (and plan for revision)
- 28.7.
  - Present complete project

EVALUATION

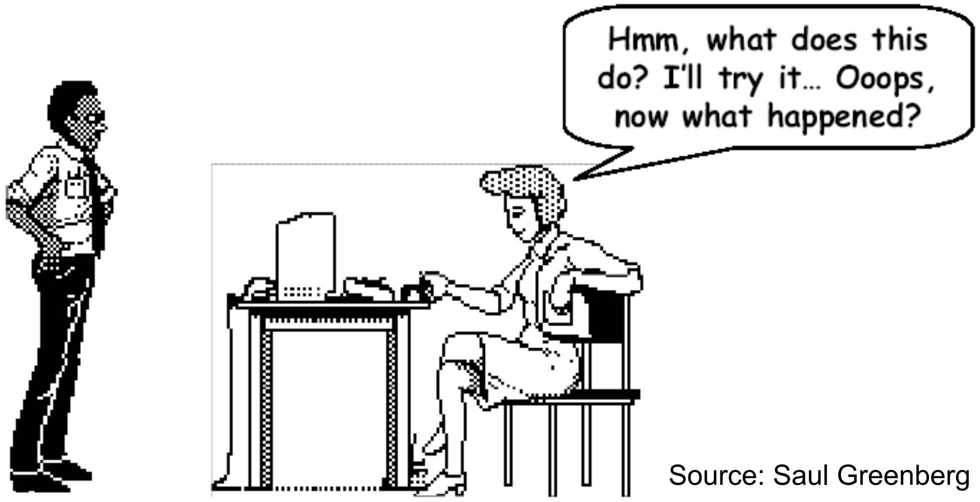
DIA Cycle: When to evaluate?



Analyze

Evaluate with or without users

Think Aloud


Hmm, what does this do? I'll try it... Ooops, now what happened?

Source: Saul Greenberg

- As Silent Observation, but user is asked to say aloud
  - What he thinks is happening (state)
  - What he is trying to achieve (goals)
  - Why he is doing something specific (actions)
- Most common method in industry

+ Good to get some insight into user's thinking, but:
  - Talking is hard while focusing on a task
  - Feels weird for most users to talk aloud
  - Conscious talking can change behavior

MEMORY MANAGEMENT & INSTRUMENTS

Reference Counting

- Object reference life cycle:

```
myobject = [[MyClass alloc] init];          // reference count = 1 after alloc
[myobject retain];       // increment reference count (retainCount == 2)
[myobject release];      // decrement reference count (retainCount == 1)
[myobject release];      // decrement reference count (retainCount == 0)
// at this point myobject is no longer valid, memory has been reclaimed
[myobject someMethod]; // error: this will crash!
```

- Can inspect current reference count:

```
NSLog(@"retainCount = %d", [textField retainCount]);
```

- Can autorelease (system releases at some point in future)

```
[myobject autorelease];
```

  Used when returning objects from methods.

Rules

- Memory rule: You are responsible for objects you allocate or copy (i.e. “allocate” or “copy” is some part of the name)!

- Not responsible:

```
NSData *data = [NSData dataWithContentsOfFile:@"file.dat"];
```

- Responsible:

```
NSData *data = [[NSData alloc] initWithContentsOfFile:@"file.dat"];
```

- Responsible:

```
NSData *data2 = [data copy];
```

- Never release objects you are not responsible for!

Objective C - Class

In .h file:

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

@interface Employee : NSObject
{ //Instance vars here
    NSString *name;
    int salary;
    int bonus;
}
// methods outside curly brackets
-   (void)setSalary:(int)cash withBonus:(int)extra
@end
```

Objective C Properties

- .h file:

```
@interface MyDetailViewController : UIViewController {
    NSString *labelText;
}
@property (nonatomic, retain) NSString *labelText;
@end
```

- .m file:

```
@synthesize labelText;
    -(void)someMethod {
    self.labelText = @”hello”;
}
```

creates accessor methods: setLabelText (retains/releases) and getLabelText.

dot-syntax means: use property’s setLabelText accessor method, will retain the object

equivalent to
[self setLabelText:@”hello”];

Implicit Setter/Getter Accessor Methods

- .h file: @property (nonatomic, retain) NSString *labelText;
- .m file: @synthesize labelText;
- Automatic creation of accessor methods:

```
- (void) setLabelText:(NSString *)newLabelText {
    [labelText release];
    labelText = newLabelText;
    [labelText retain];
}
- (NSString*) getLabelText {
    return labelText;
}
```

decrement reference counter on old object (if any)

increment reference counter on new object (if any)

- Properties are accessible from other classes, data members only if declared @public

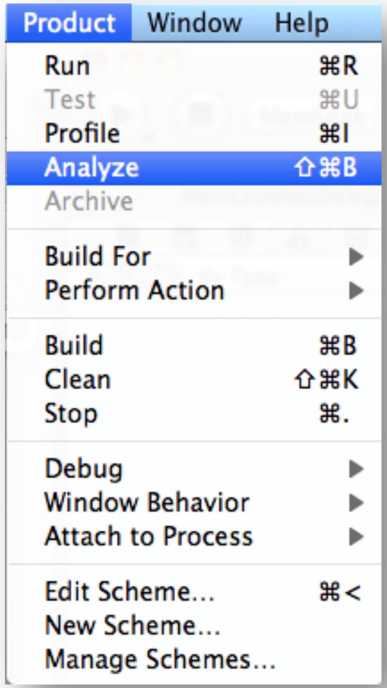
Property Attributes

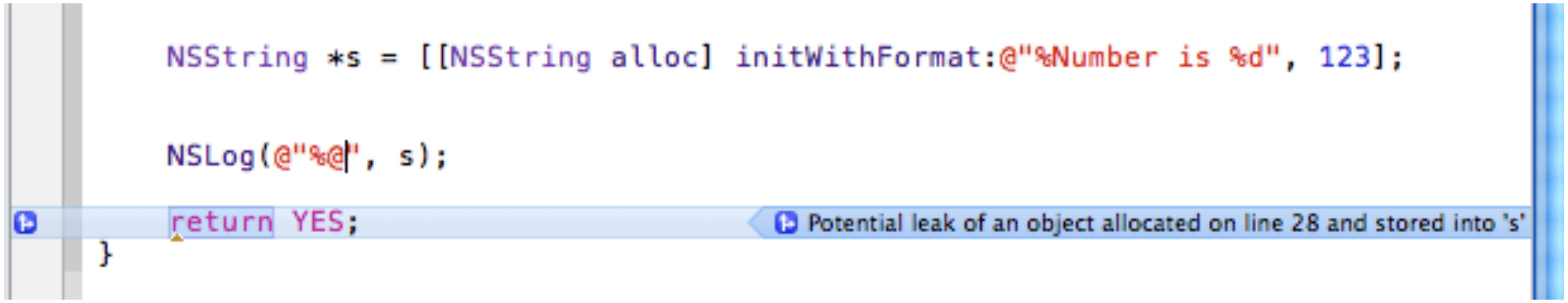
- Writability: readwrite (default), readonly
- Setter semantics: assign, retain, copy
- Atomicity: atomic (default), nonatomic

- “readonly” means only a getter, but no setter accessor method is generated by @synthesize

Analyzing Code

- Xcode static analysis for simple problems





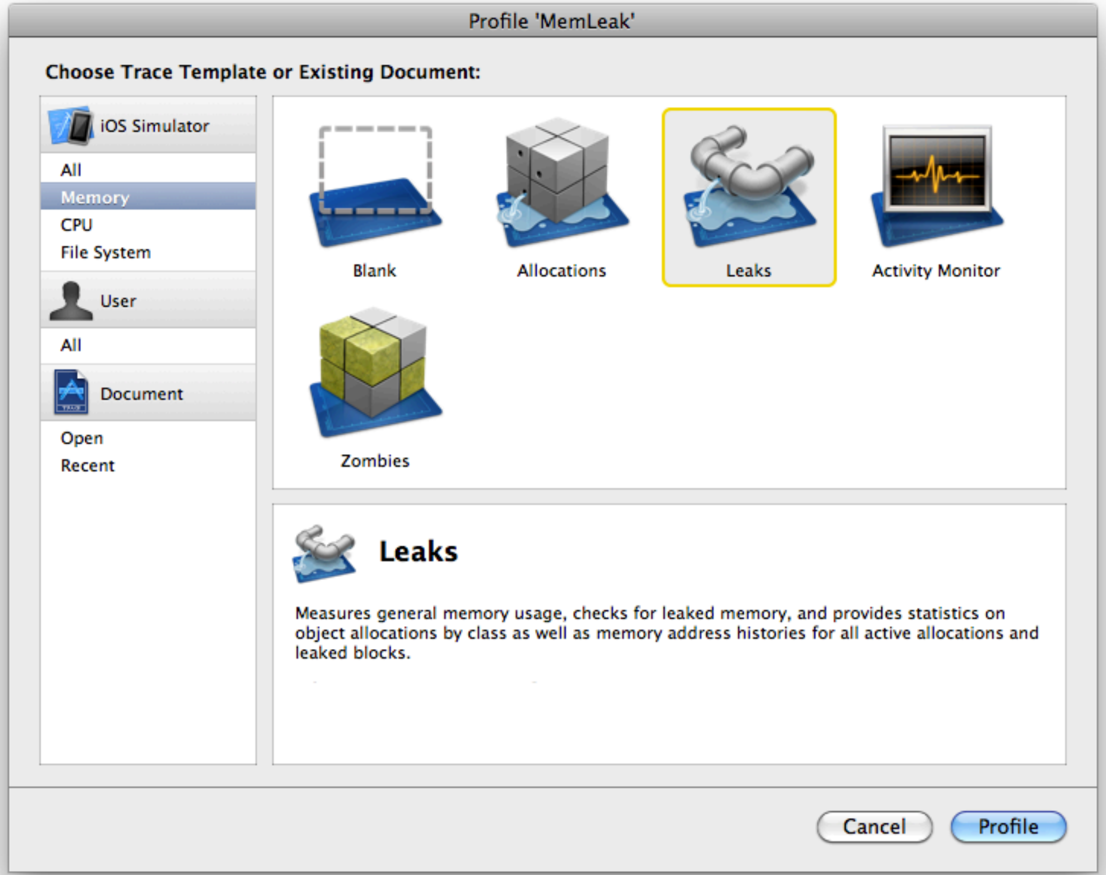
```
    NSString *s = [[NSString alloc] initWithFormat:@"%Number is %d", 123];

    NSLog(@"%@", s);

    return YES;              Potential leak of an object allocated on line 28 and stored into 's'
}
```

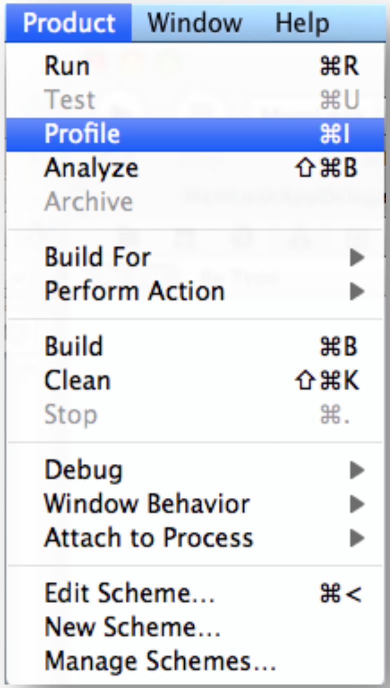
Profiling Code

- Analyzing runtime behavior

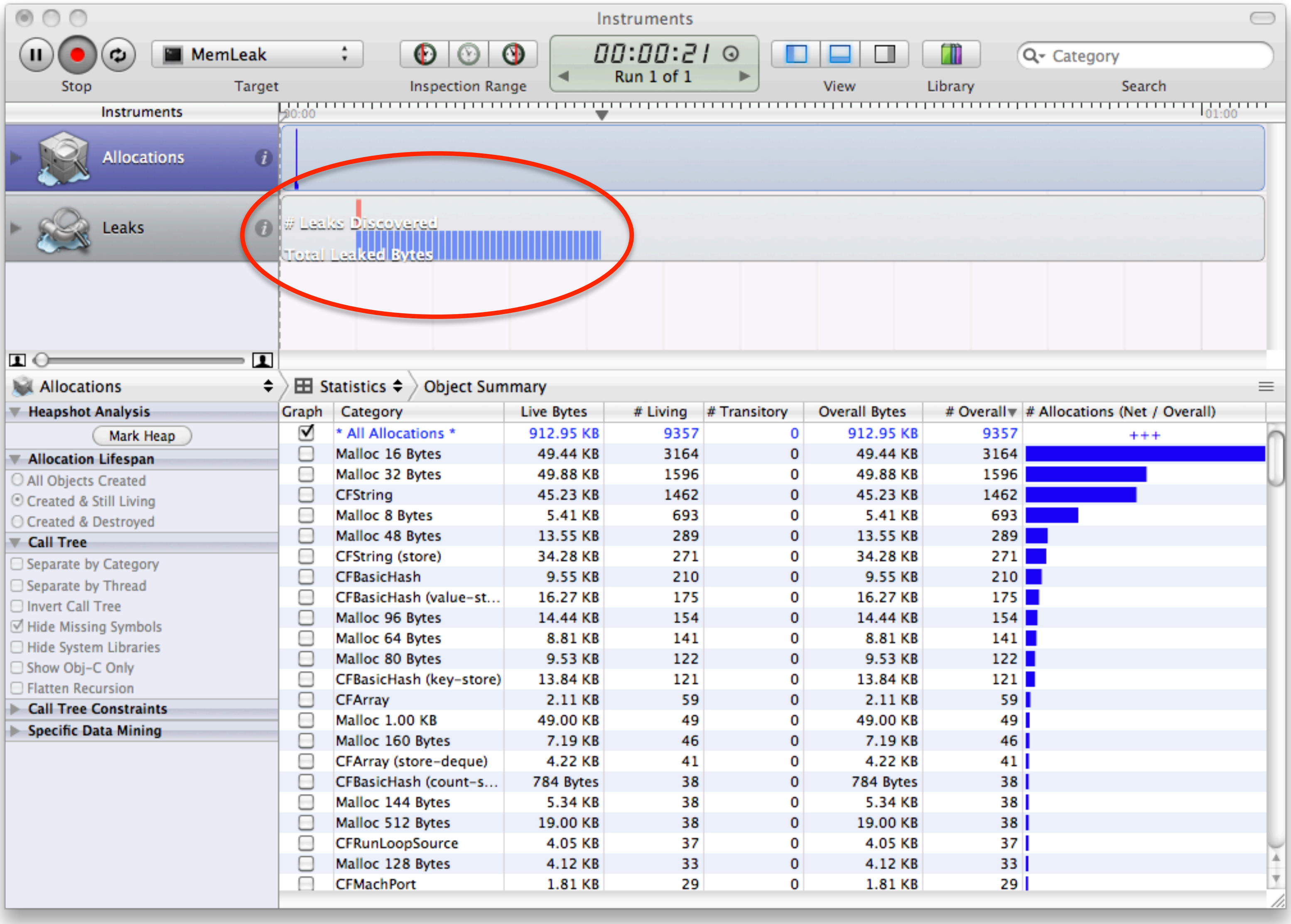


Leaks

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.



Profiling Code



Allocations | Statistics | Object Summary

Heapshot Analysis
Mark Heap

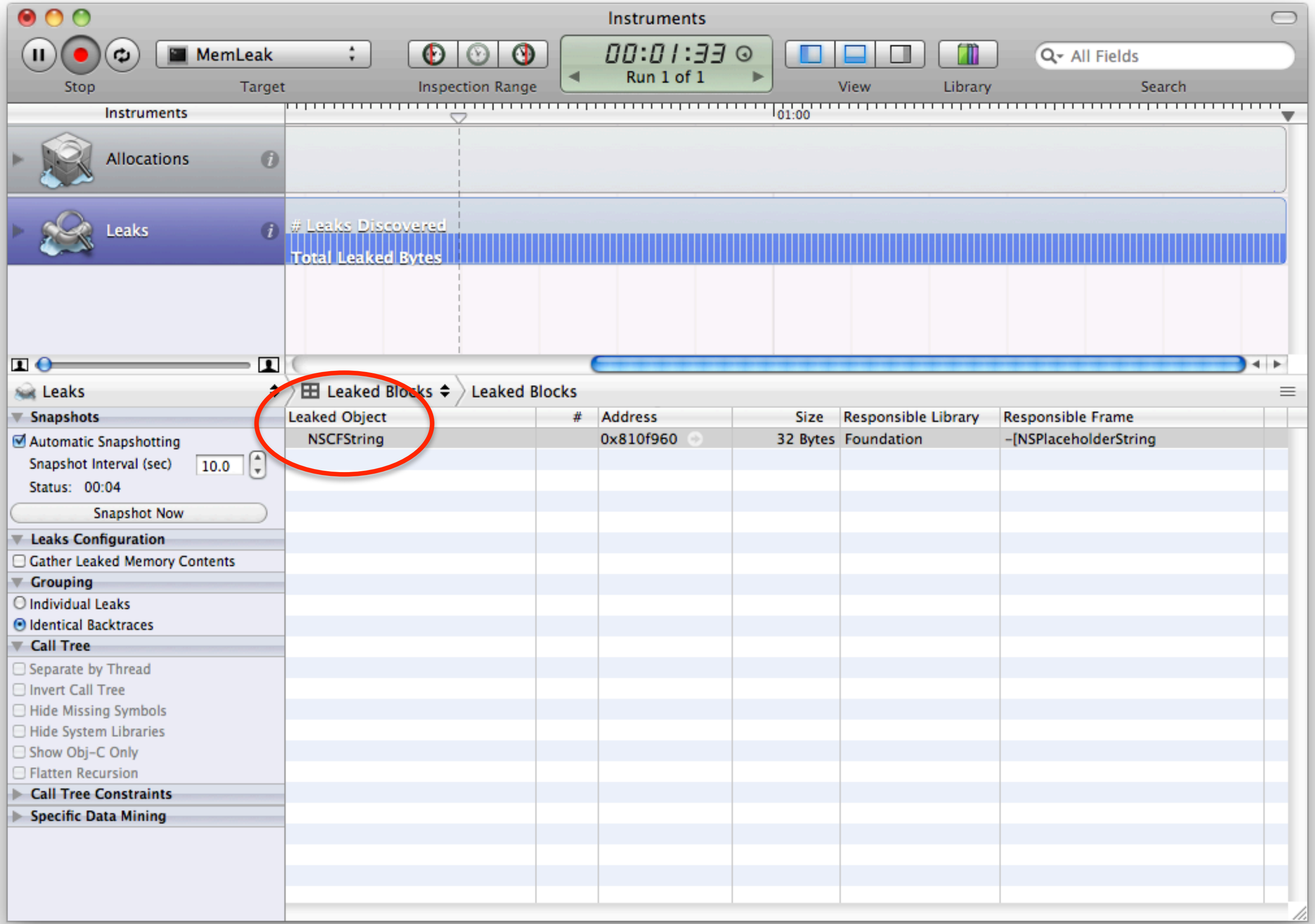
Allocation Lifespan
- All Objects Created
- [✓ Created & Still Living]
- Created & Destroyed

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

Call Tree Constraints
Specific Data Mining

Graph	Category	Live Bytes	# Living	# Transitory	Overall Bytes	# Overall	# Allocations (Net / Overall)
☑	* All Allocations *	912.95 KB	9357	0	912.95 KB	9357	+++
☐	Malloc 16 Bytes	49.44 KB	3164	0	49.44 KB	3164	
☐	Malloc 32 Bytes	49.88 KB	1596	0	49.88 KB	1596	
☐	CFString	45.23 KB	1462	0	45.23 KB	1462	
☐	Malloc 8 Bytes	5.41 KB	693	0	5.41 KB	693	
☐	Malloc 48 Bytes	13.55 KB	289	0	13.55 KB	289	
☐	CFString (store)	34.28 KB	271	0	34.28 KB	271	
☐	CFBasicHash	9.55 KB	210	0	9.55 KB	210	
☐	CFBasicHash (value-st...	16.27 KB	175	0	16.27 KB	175	
☐	Malloc 96 Bytes	14.44 KB	154	0	14.44 KB	154	
☐	Malloc 64 Bytes	8.81 KB	141	0	8.81 KB	141	
☐	Malloc 80 Bytes	9.53 KB	122	0	9.53 KB	122	
☐	CFBasicHash (key-store)	13.84 KB	121	0	13.84 KB	121	
☐	CFArray	2.11 KB	59	0	2.11 KB	59	
☐	Malloc 1.00 KB	49.00 KB	49	0	49.00 KB	49	
☐	Malloc 160 Bytes	7.19 KB	46	0	7.19 KB	46	
☐	CFArray (store-deque)	4.22 KB	41	0	4.22 KB	41	
☐	CFBasicHash (count-s...	784 Bytes	38	0	784 Bytes	38	
☐	Malloc 144 Bytes	5.34 KB	38	0	5.34 KB	38	
☐	Malloc 512 Bytes	19.00 KB	38	0	19.00 KB	38	
☐	CFRunLoopSource	4.05 KB	37	0	4.05 KB	37	
☐	Malloc 128 Bytes	4.12 KB	33	0	4.12 KB	33	
☐	CFMachPort	1.81 KB	29	0	1.81 KB	29	

Profiling Code



Snapshots
- [✓] Automatic Snapshotting
- Snapshot Interval (sec) 10.0
- Status: 00:04
- 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
NSCFString		0x810f960	32 Bytes	Foundation	-[NSPlaceholderString

Best Method Avoiding Memory Leaks

- Program carefully, think hard
- Follow the memory management rules

- Ugly truth:
  Some leaks are in the frameworks as well!

Presentation Structure

- Target audience of presentation: investors
  - Imagine getting funding for a startup company
- Suggested presentation outline (7 minutes per group)
  - Group and product introduction (1 min)
  - Target user group (1 min)
  - Important features (1 min)
  - Role-play of usage scenario,
    presentation of interaction techniques (3 min)
  - Design process, design principles, challenges (1 min)