iOS App Performance

Memory

Session 242

Morgan Grainger
Software Engineer

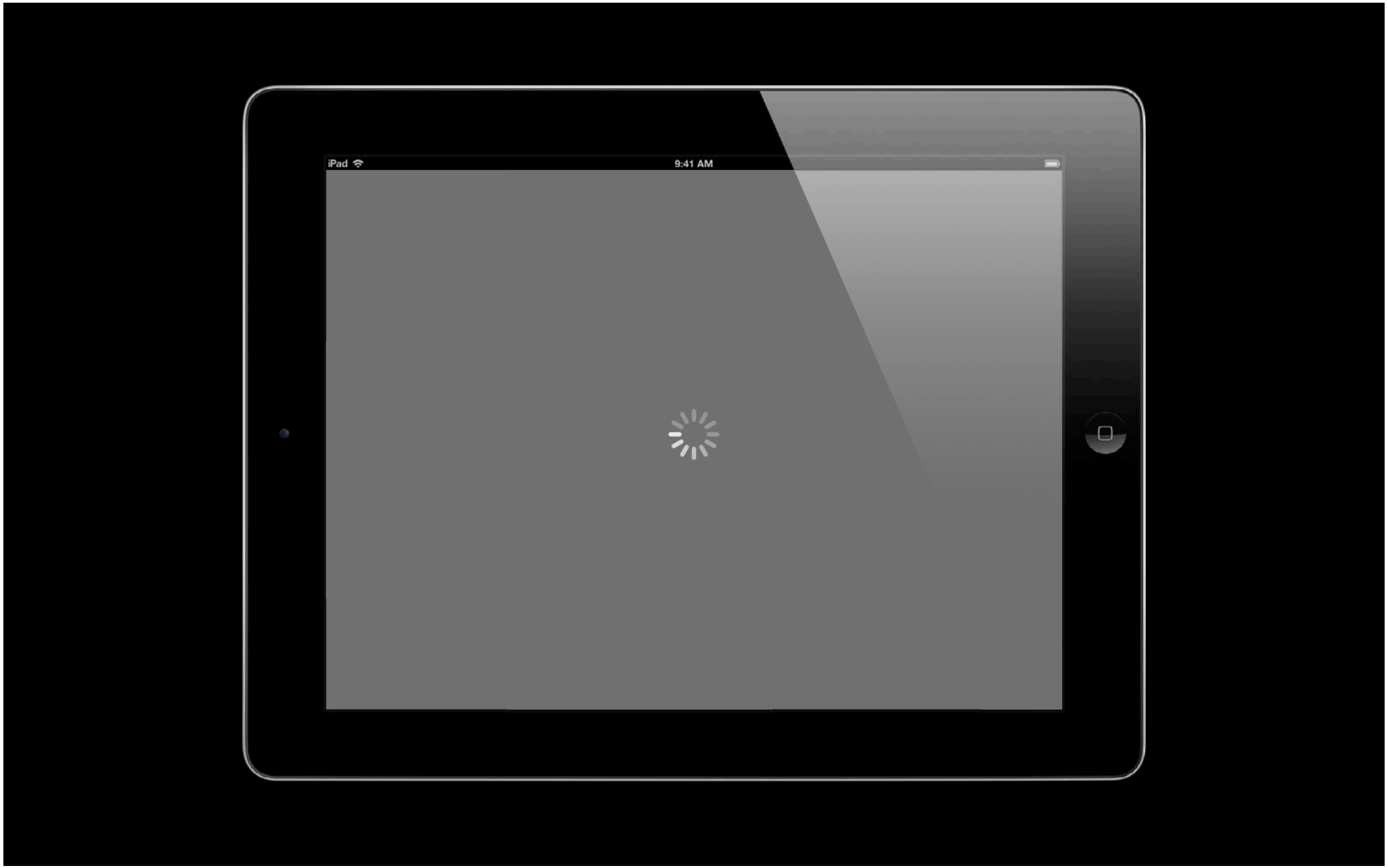
Agenda

- Memory Landscape
- iOS Memory Fundamentals
- Memory Pressure
- Finding Memory Issues
- Tools Tips and Tricks

iOS Memory Landscape

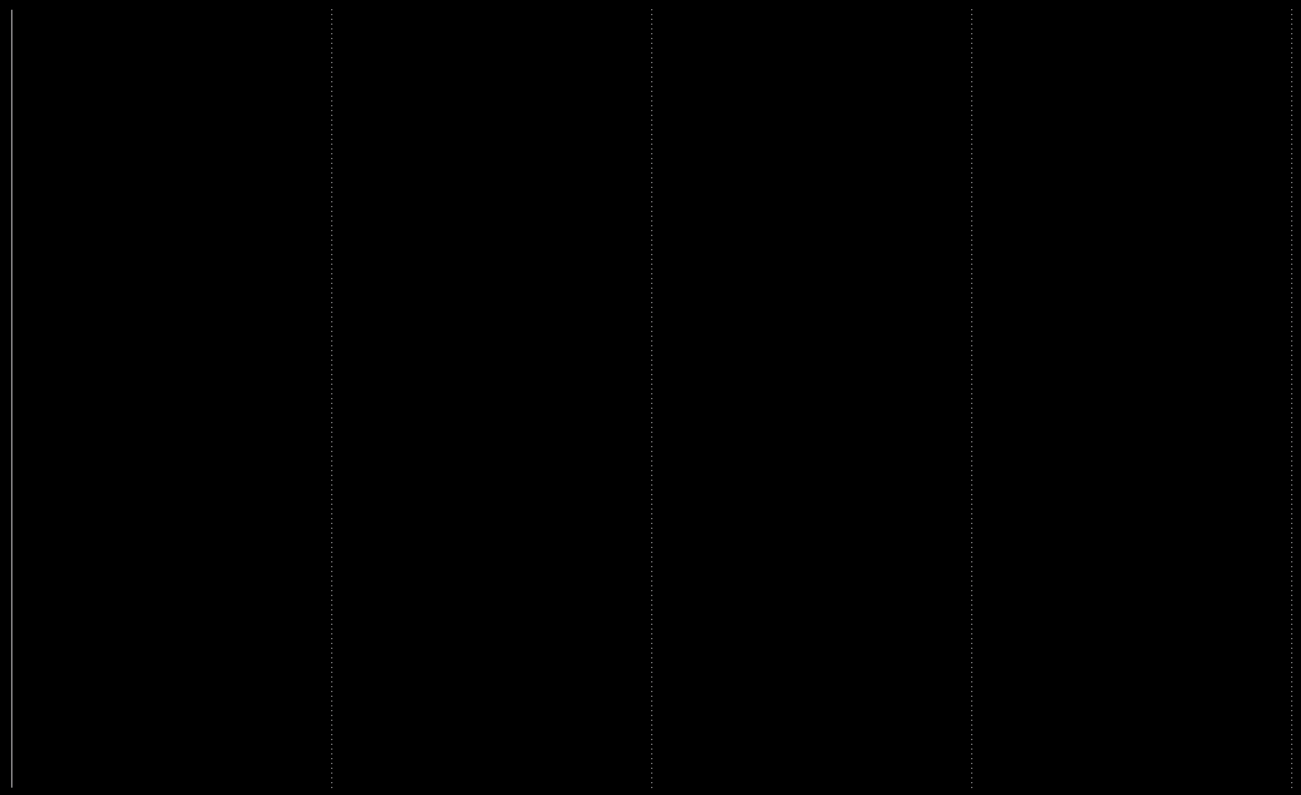

iPad
9:41 AM


iPad
9:41 AM
Messages
FaceTime
Photos
Camera
Maps
YouTube
Clock
Photo Booth
Monday
11
Calendars
Contacts
Notes
Reminders
Newsstand
iTunes
App Store
Game Center
Settings
Safari
Mail
Videos
Music


iPad
9:41 AM

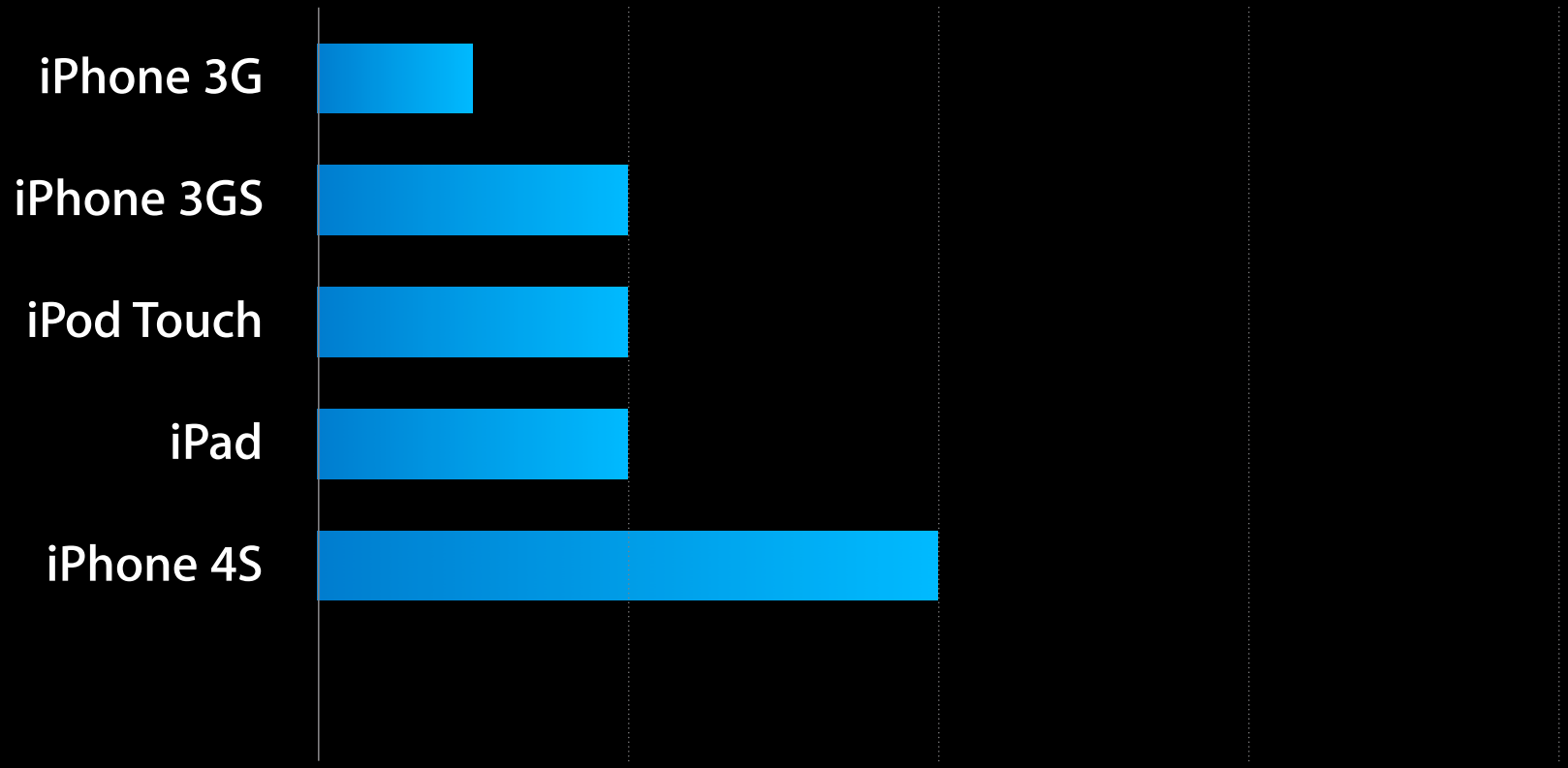
“I love this app, but it always crashes after a few minutes.”

The March of Progress

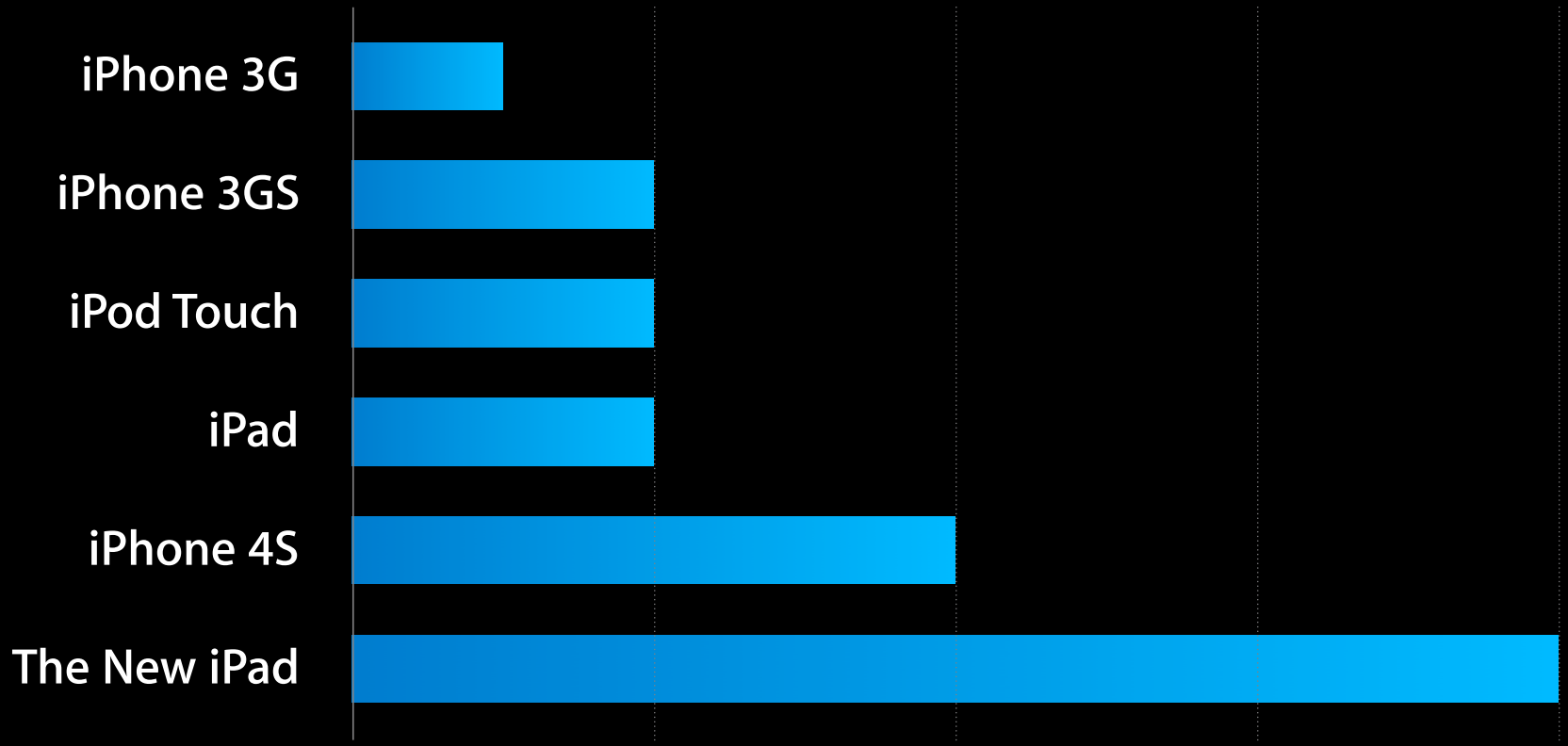


The March of Progress

The March of Progress



The March of Progress




iPad
9:41 AM
Friday
7
Messages
Calendar
Notes
Reminders
Maps
YouTube
Videos
Contacts
Game Center
iTunes
App Store
Newsstand
FaceTime
Camera
Photo Booth
Settings
Safari
Mail
Photos
Music


iPad
9:41 AM
Messages
FaceTime
Photos
Camera
Maps
YouTube
Clock
Photo Booth
Monday
11
Calendars
Contacts
Notes
Reminders
Newsstand
iTunes
App Store
Game Center
Settings
Safari
Mail
Videos
Music

iOS Memory Fundamentals

Understanding Low Memory

Understanding Low Memory

```
Incident Identifier: C6CCECE6-E2B8-4426-B4D1-BE56599AE468
CrashReporter Key:   4b2eb6dfff066742d067aa5d505790a9338464cb
Hardware Model:      iPhone2,1
OS Version:          iPhone OS 6.0
UDID:                e1cbbbfbd5af450138d6c7a144900e62d171d48f
Date:                2012-06-11 09:41:00 -0700

Free pages:        507
Active pages:      1062
Inactive pages:    646
Throttled pages:   47919
Purgeable pages:   0
Wired pages:       14566
Largest process:   YourApp

Processes
Name                        <UUID>                            rpages          [reason]       (state)

      YourApp    <c6bf10738ad63bac97674e0b50d3ba55>          23197            [vm]       (frontmost) (continuous)
         afcd    <cb4f085516ba37c89a093107b0633c31>            114            [vm]       (daemon) (idle)
  MobilePhone    <baac0168db6f30b4917122274bd18450>            944            [vm]       (resume) (continuous)
         tccd    <10049f303aea3df3a4a8f26a9716fb2f>            172            [vm]       (daemon)
          kbd    <ff9db8dd78203f279706712c9d98bb6c>            375            [vm]       (daemon)
         ptpd    <efffd169c8d33a79a4ba5d30ea2962b6>           1682                       (daemon)
     powerlog    <d7555671415f3fd3800623765604b587>            524                       (daemon)
      syslogd    <7477c8ba4f0e356bb3ea615fc9976685>            147                       (daemon)
        wifid    <c84d495dee503822a1ac7d11701c526b>            497                       (daemon)
   aosnotifyd    <ead3ebb514a3339b80ea4ed739ce53c5>            401                       (daemon)
          atc    <8b3cbb041b453e3fa0521c11f52b731c>            778                       (daemon)
iaptransportd    <04a9afc8f43035b58faa81e1b3831bb9>            187                       (daemon)
    locationd    <04d2854a2b6b3a34ab5d0e0330b72e4e>           1067                       (daemon)
  SpringBoard    <ad72e5c25ca43ebbaedf427ee56d153f>           4003
```

iOS Memory Fundamentals

What you'll learn

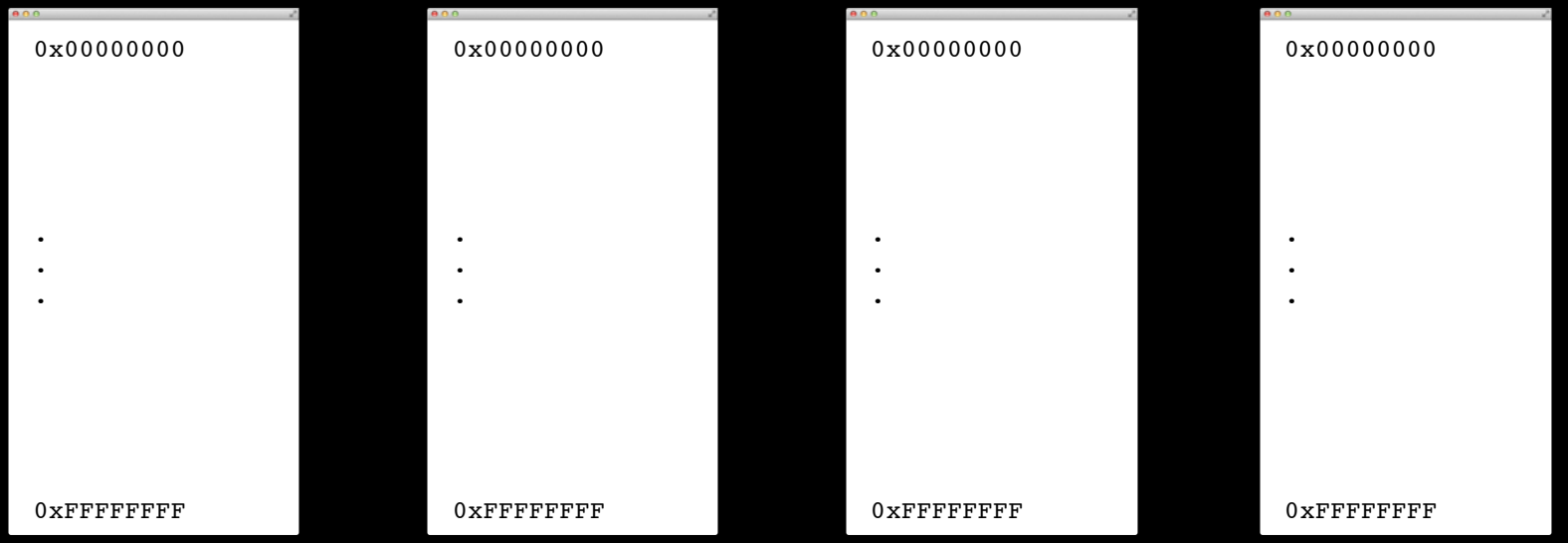
- How is memory allocated and managed on iOS?
- What types of memory use matter?
  - Clean and dirty
- What happens when iOS runs low on memory?

iOS Memory Fundamentals

What you'll learn

- How is memory allocated and managed on iOS?
- What types of memory use matter?
  - Clean and dirty
- What happens when iOS runs low on memory?

Address Space Fundamentals



Safari Mail Calendar Contacts

Do the Math

Do the Math

$$2^{32} = 4\ GB$$

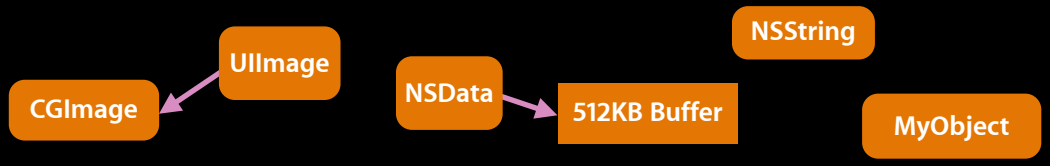
Pointer Range

Virtual Memory

- Physical memory divided into 4 KB pages
- Not all pages in memory at once

Virtual Memory

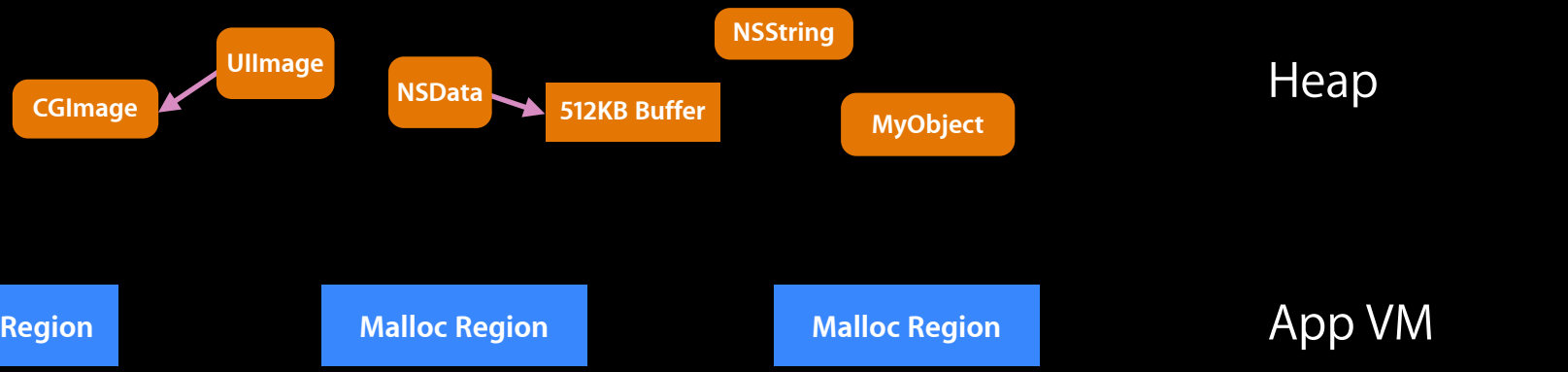
Virtual Memory



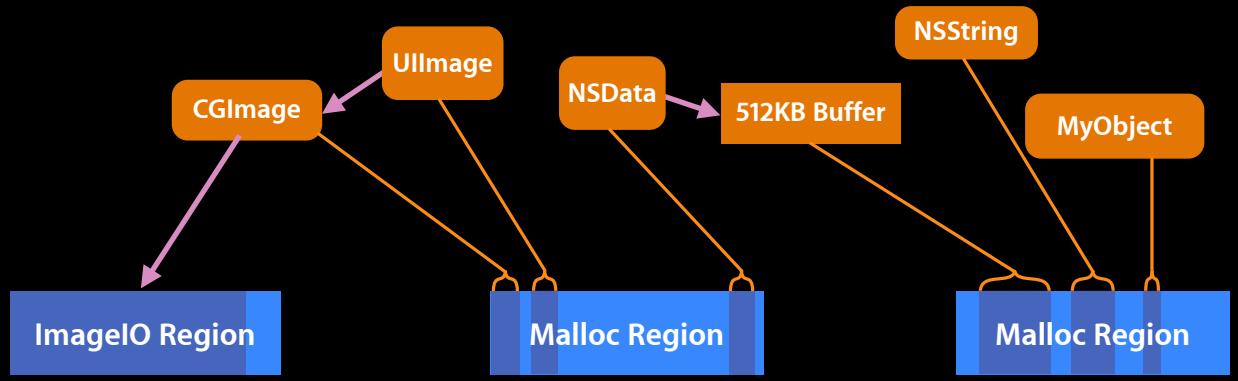
Heap

Physical Mem

Virtual Memory



Virtual Memory

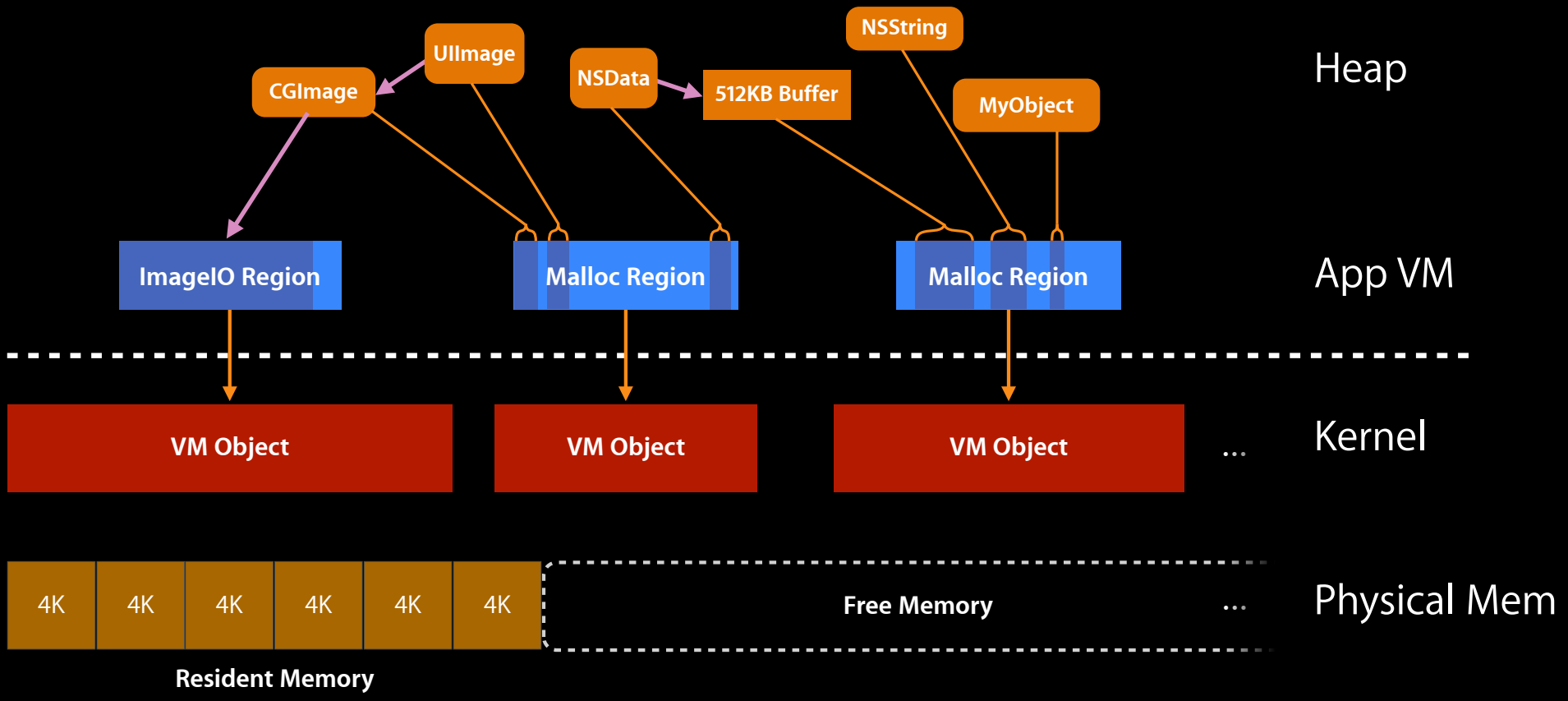


Heap

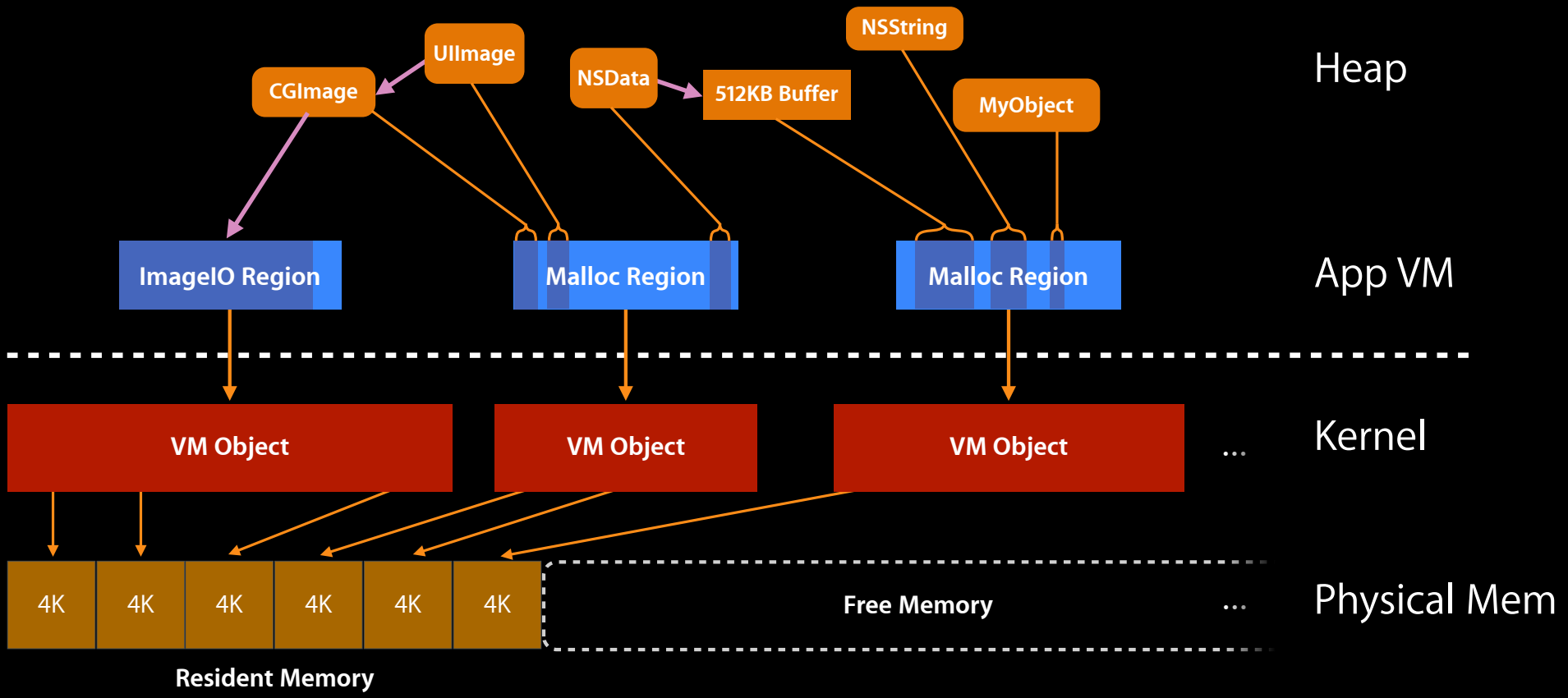
App VM

Physical Mem

Virtual Memory

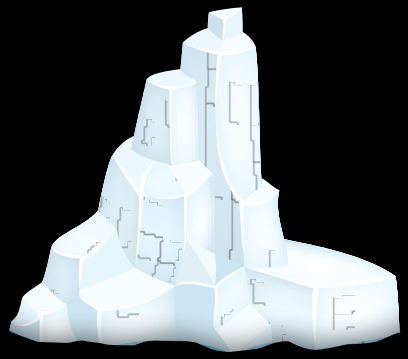


Virtual Memory



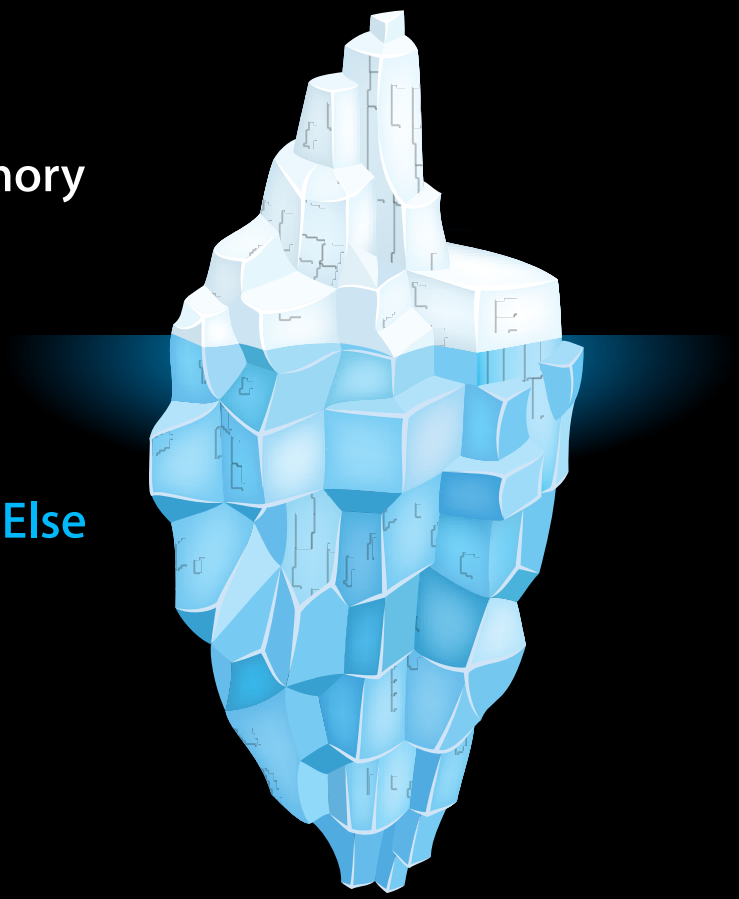
Memory Footprint

Heap Memory



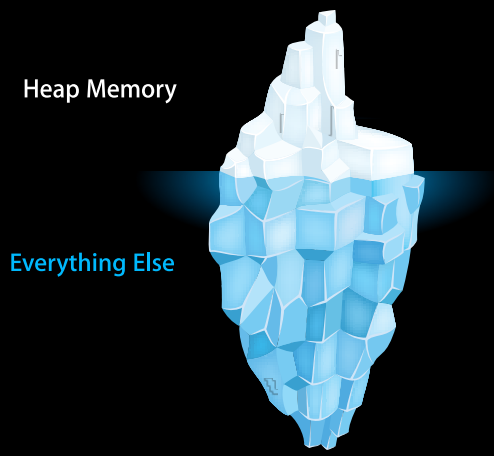
Memory Footprint

Heap Memory



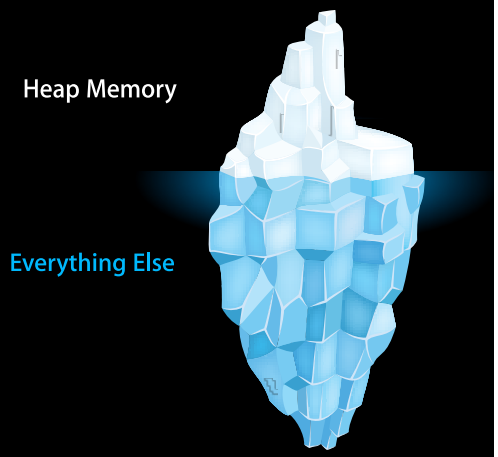
Everything Else

Memory Footprint



More Than Just Objects

- Heap memory
  - +[NSObject alloc]/malloc
  - Objects/buffers allocated by frameworks
- Other memory
  - Code and globals (__TEXT, __DATA)
  - Thread stacks
  - Image data
  - CALayer backing stores
  - Database caches
- Additional memory outside of your application!



iOS Memory Fundamentals

What you'll learn

- How is memory allocated and managed on iOS?
- What types of memory use matter?
  - Clean and dirty
- What happens when iOS runs low on memory?

Not Enough to Go Around

- Memory on iOS is limited

Not Enough to Go Around

- Memory on iOS is limited
- How can the system reclaim memory?

Not Enough to Go Around

- Memory on iOS is limited
- How can the system reclaim memory?
  - Persist it (page it out)

Not Enough to Go Around

- Memory on iOS is limited
- How can the system reclaim memory?

  Persist it (page it out) 

Not Enough to Go Around

- Memory on iOS is limited
- How can the system reclaim memory?
  - Persist it (page it out) 
  - Evict it without storing

Not Enough to Go Around

- Memory on iOS is limited
- How can the system reclaim memory?
  - Persist it (page it out) 
  - Evict it without storing 

Evicting Memory

Evicting Memory

- Destructive if memory cannot be retrieved or recreated
  - Only recourse is to terminate the owning process

Evicting Memory

- Destructive if memory cannot be retrieved or recreated
  - Only recourse is to terminate the owning process
- *Clean memory:* memory for which a copy exists on disk
  - Code, frameworks, memory-mapped files

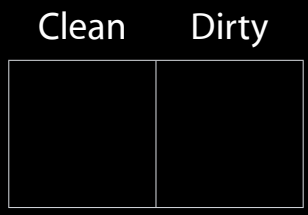
Evicting Memory

- Destructive if memory cannot be retrieved or recreated
  - Only recourse is to terminate the owning process
- *Clean memory:* memory for which a copy exists on disk
  - Code, frameworks, memory-mapped files
- *Dirty memory:* everything else
  - Heap allocations, decompressed images, database caches

Clean or Dirty?

The game show

Clean or Dirty?




```
- (void)displayWelcomeMessage {

    NSString *welcomeMessage = [NSString stringWithUTF8String:
                                    “Welcome to WWDC!”];

    self.alertView.title = welcomeMessage;
    [self.alertView show];
}
```

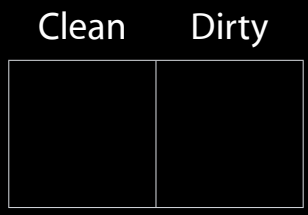
Clean or Dirty?


```
- (void)displayWelcomeMessage {

    NSString *welcomeMessage = [NSString stringWithUTF8String:
                                    “Welcome to WWDC!”];

    self.alertView.title = welcomeMessage;
    [self.alertView show];
}
```

Clean or Dirty?

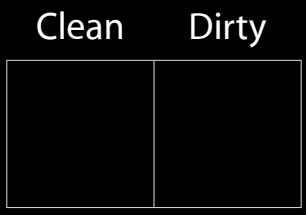



```
- (void)displayWelcomeMessage {
    NSString *welcomeMessage = @"Welcome to WWDC!";
    self.alertView.title = welcomeMessage;
    [self.alertView show];
}
```

Clean or Dirty?


```
- (void)displayWelcomeMessage {
    NSString *welcomeMessage = @"Welcome to WWDC!";
    self.alertView.title = welcomeMessage;
    [self.alertView show];
}
```

Clean or Dirty?

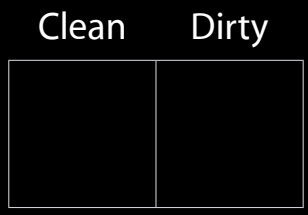



```
- (void)allocateSomeMemory {
    void *buf = malloc(10 * 1024 * 1024);
    …
}
```

Clean or Dirty?


```
- (void)allocateSomeMemory {
    void *buf = malloc(10 * 1024 * 1024);
    …
}
```

Clean or Dirty?

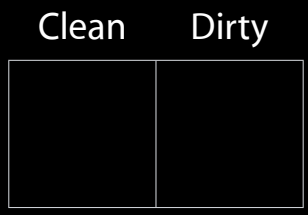



```
- (void)allocateSomeMemory {
    void *buf = malloc(10 * 1024 * 1024);
    for (unsigned int i = 0; i < sizeof(buf), i++) {
        buf[i] = (char)random();
    }
    …
}
```

Clean or Dirty?

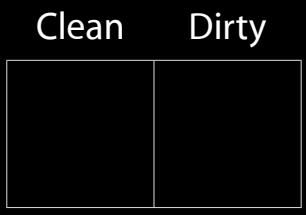

```
- (void)allocateSomeMemory {
    void *buf = malloc(10 * 1024 * 1024);
    for (unsigned int i = 0; i < sizeof(buf), i++) {
        buf[i] = (char)random();
    }
    …
}
```

Clean or Dirty?

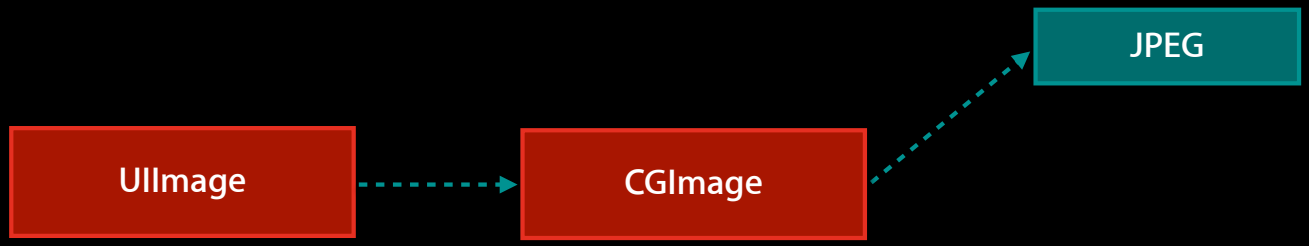



```
UIImage *wwdcLogo = [UIImage imageNamed:@”WWDC12Logo”];
```

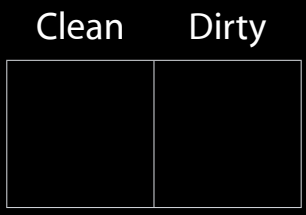
Clean or Dirty?



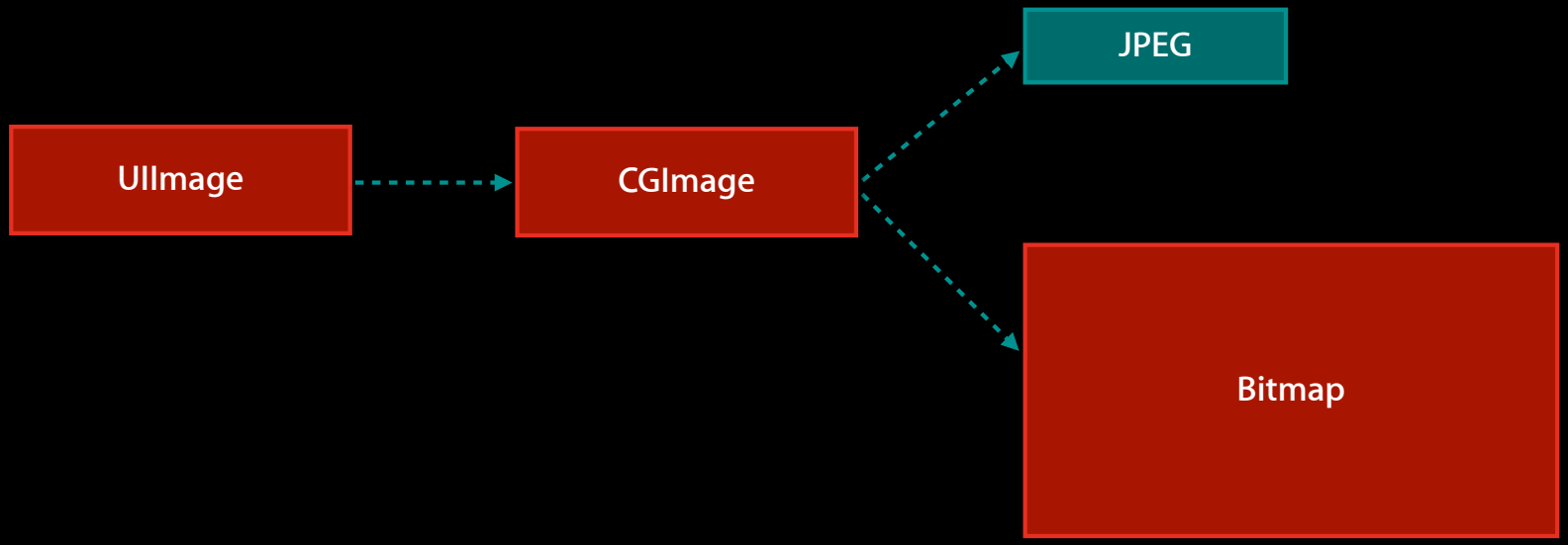

```
UIImage *wwdcLogo = [UIImage imageNamed:@"WWDC12Logo"];
```



Clean or Dirty?



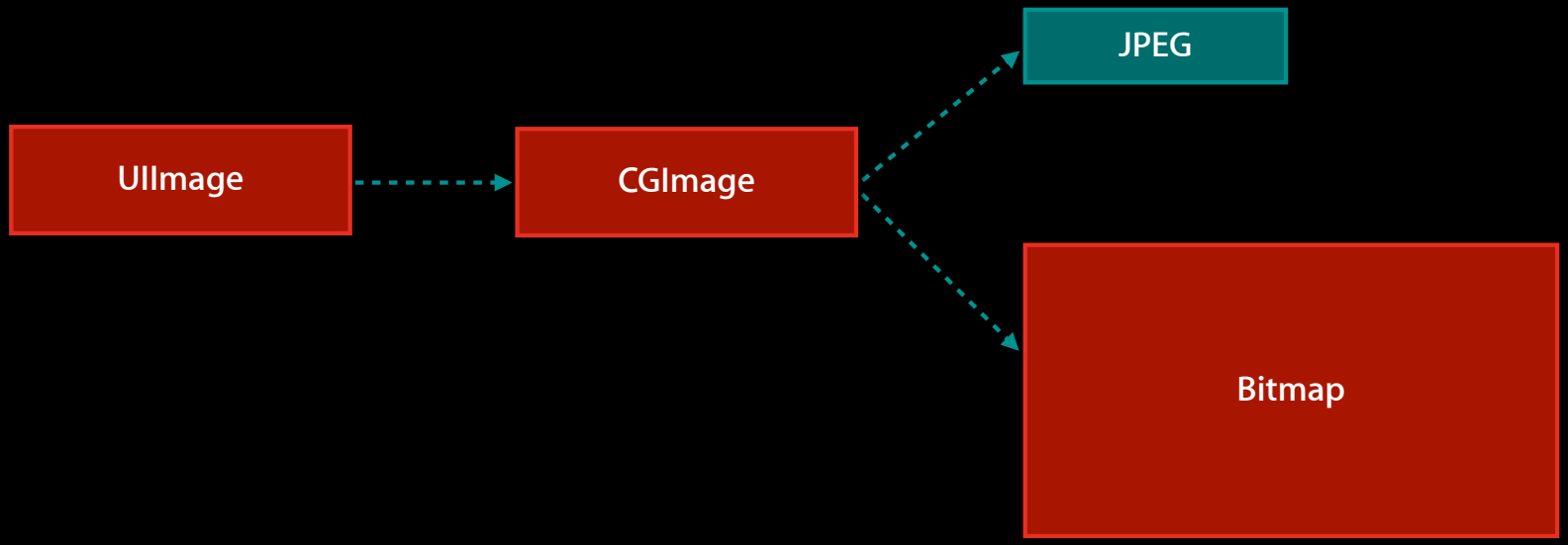

```
UIImage *wwdcLogo = [UIImage imageNamed:@"WWDC12Logo"];
UIImageView *view = [[UIImageView alloc] initWithImage:wwdcLogo];
[contentView addSubview:view];
…
```



Clean or Dirty?

Clean	Dirty
	✓

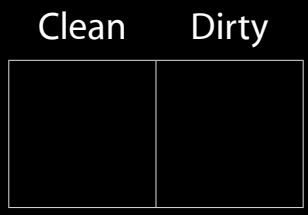
```
UIImage *wwdcLogo = [UIImage imageNamed:@"WWDC12Logo"];
UIImageView *view = [[UIImageView alloc] initWithImage:wwdcLogo];
[contentView addSubview:view];
…
```



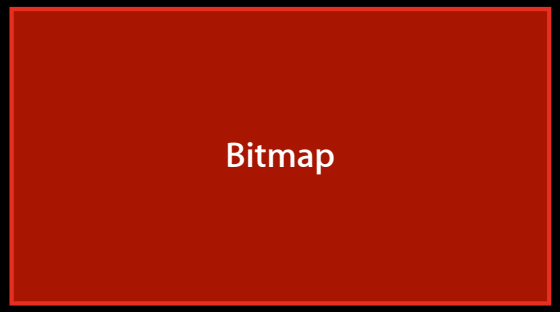
Clean or Dirty?


```
UIGraphicsBeginImageContext();
```

Clean or Dirty?




```
UIGraphicsBeginImageContext();
[[myview layer] renderInContext:UIGraphicsGetCurrentContext()];
```



Clean or Dirty?


```
UIGraphicsBeginImageContext();
[[myview layer] renderInContext:UIGraphicsGetCurrentContext()];
UIImage *snapshot = UIGraphicsGetImageFromCurrentImageContext();
```

Clean or Dirty?

Clean	Dirty
	✓

```
UIGraphicsBeginImageContext();
[[myview layer] renderInContext:UIGraphicsGetCurrentContext()];
UIImage *snapshot = UIGraphicsGetImageFromCurrentImageContext();
```

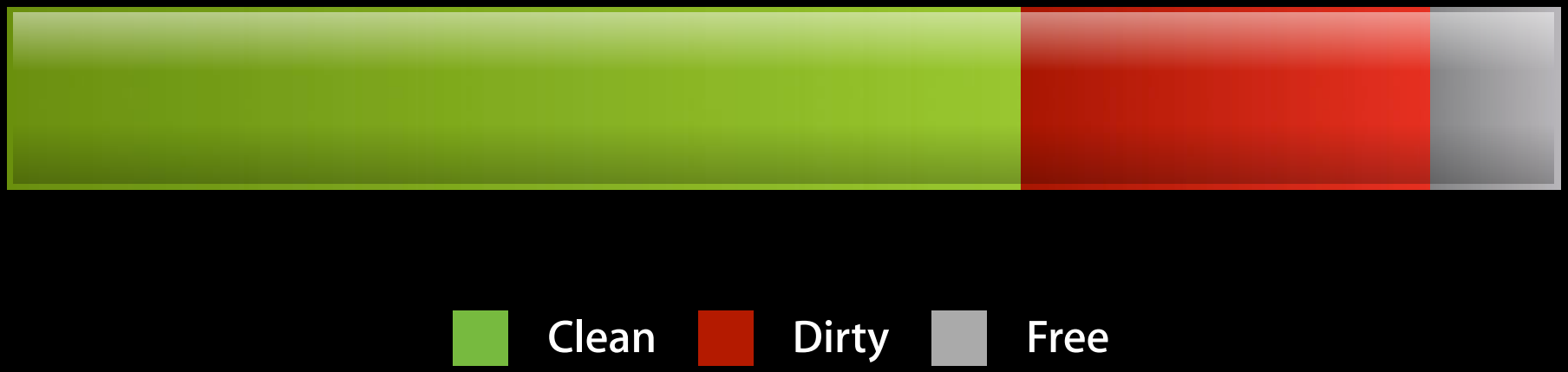
Most App Allocations are Dirty

iOS Memory Fundamentals

What you'll learn

- How is memory allocated and managed on iOS?
- What types of memory use matter?
  - Clean and dirty
- What happens when iOS runs low on memory?

Managing System-Wide Memory



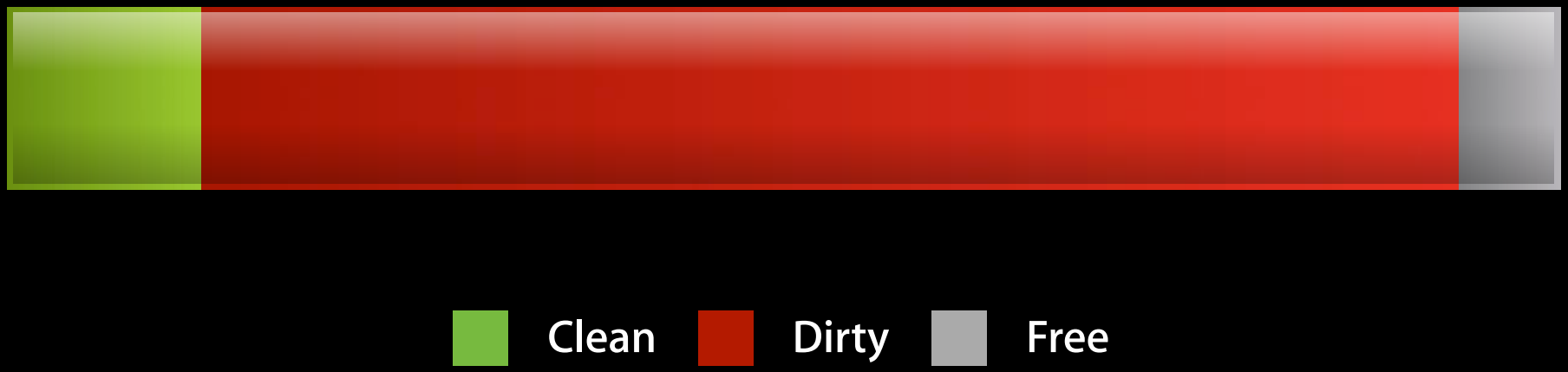
Managing System-Wide Memory

Launch Your App

Clean Dirty Free

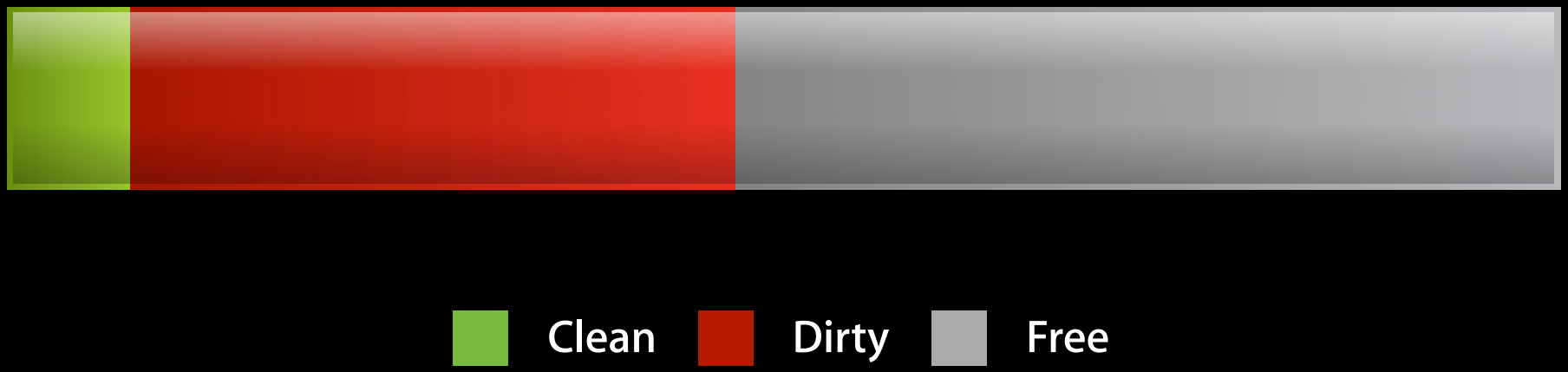
Managing System-Wide Memory

Memory Pressure



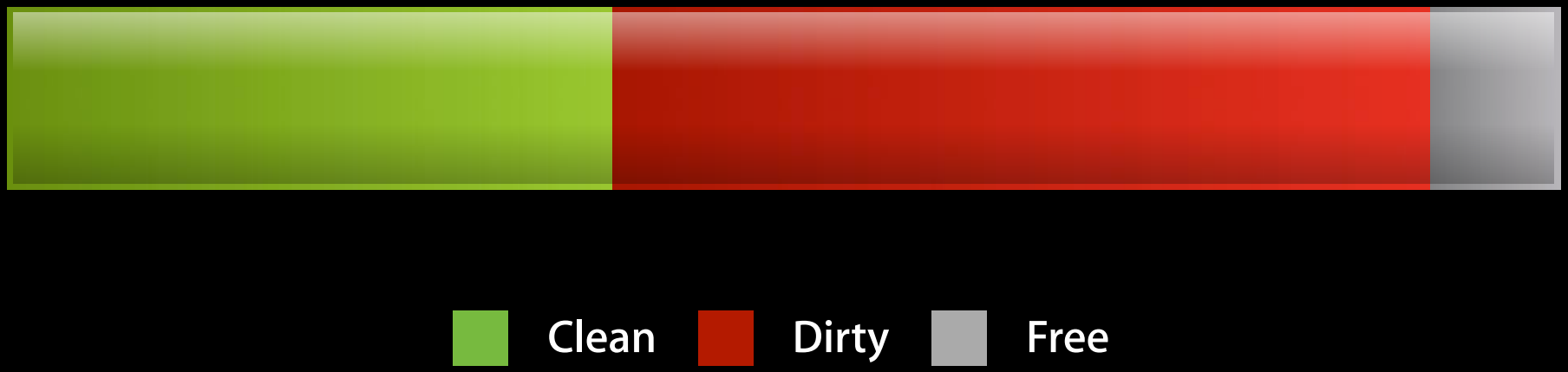
Managing System-Wide Memory

Terminate Background Apps



Managing System-Wide Memory

Run Some Other Apps



Memory Warnings

A challenge

- Fact of life on memory-constrained devices

Memory Warnings

A challenge

- Fact of life on memory-constrained devices
- Last chance to preserve user experience

Memory Warnings

A challenge

- Fact of life on memory-constrained devices
- Last chance to preserve user experience
- Ensure that your application can respond

Memory Warnings

A challenge

- Fact of life on memory-constrained devices
- Last chance to preserve user experience
- Ensure that your application can respond
  - Notifications arrive on main thread

Memory Warnings

A challenge

- Fact of life on memory-constrained devices
- Last chance to preserve user experience
- Ensure that your application can respond
  - Notifications arrive on main thread
  - Avoid large, rapid allocations

Memory Warnings

A challenge

- Fact of life on memory-constrained devices
- Last chance to preserve user experience
- Ensure that your application can respond
  - Notifications arrive on main thread
  - Avoid large, rapid allocations
- Stay safe in the background

Memory Warnings

A challenge

- Fact of life on memory-constrained devices
- Last chance to preserve user experience
- Ensure that your application can respond
  - Notifications arrive on main thread
  - Avoid large, rapid allocations
- Stay safe in the background
  - `-[id <UIApplicationDelegate> -applicationDidEnterBackground:]`

Memory Warnings

An opportunity

Memory Warnings

An opportunity

- Free as much as possible
  - But don’t sacrifice user experience

Memory Warnings

An opportunity

- Free as much as possible
  - But don't sacrifice user experience
- Many ways to respond
  - `UIApplicationDidReceiveMemoryWarningNotification`
  - `-[id <UIApplicationDelegate> -applicationDidReceiveMemoryWarning:]`
  - `-[UIViewController didReceiveMemoryWarning]`

Memory Warnings

An opportunity

- Free as much as possible
  - But don't sacrifice user experience
- Many ways to respond
  - `UIApplicationDidReceiveMemoryWarningNotification`
  - `-[id <UIApplicationDelegate> -applicationDidReceiveMemoryWarning:]`
  - `-[UIViewController didReceiveMemoryWarning]`
- No longer necessary or called
  - ~~`-[UIViewController viewDidUnload]`~~

Memory Limits on Devices

Memory Limits on Devices

- How much can you use?

Memory Limits on Devices

- How much can you use?
- Test on each device

Memory Limits on Devices

- How much can you use?
- Test on each device
- Limit of 650 MB on the new iPad
  - Provides certainty

Memory Limits on Devices

- How much can you use?
- Test on each device
- Limit of 650 MB on the new iPad
  - Provides certainty
- Use less if possible

Demo

VM Tracker

Pay Attention to Dirty Memory!

Avoid Usage Spikes

Avoid Usage Spikes

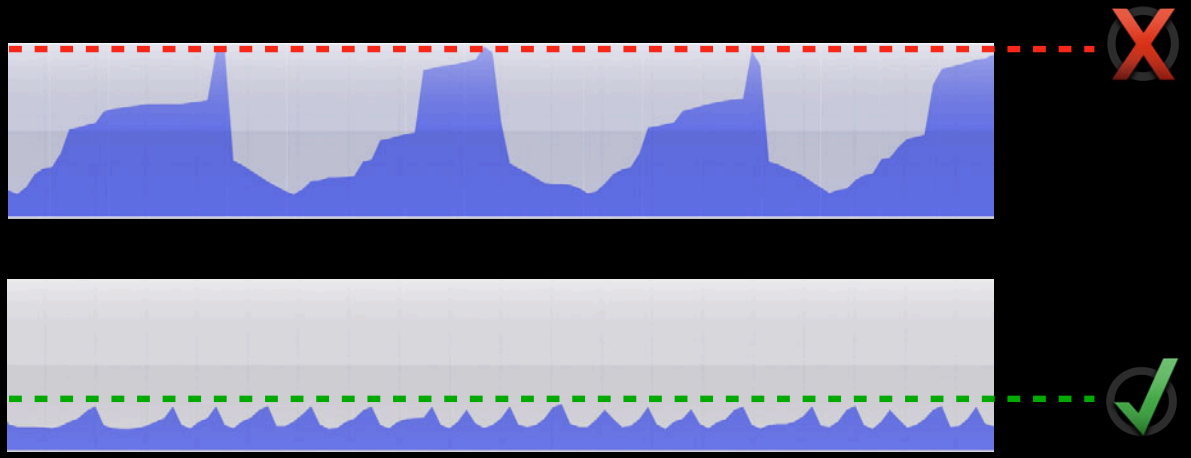
- Memory high-water mark matters

Avoid Usage Spikes

- Memory high-water mark matters
  - Use Allocations and VM Tracker graphs to identify spikes

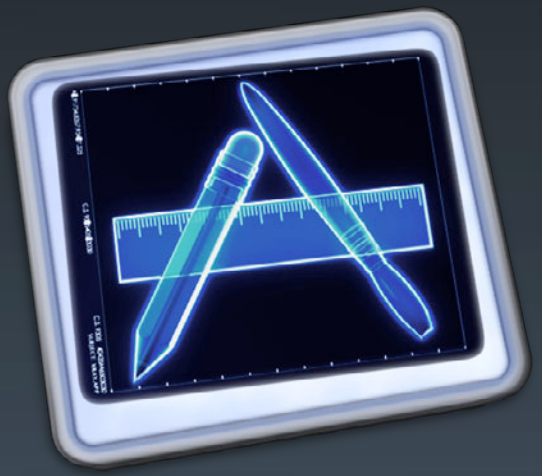
Avoid Usage Spikes

- Memory high-water mark matters
  - Use Allocations and VM Tracker graphs to identify spikes

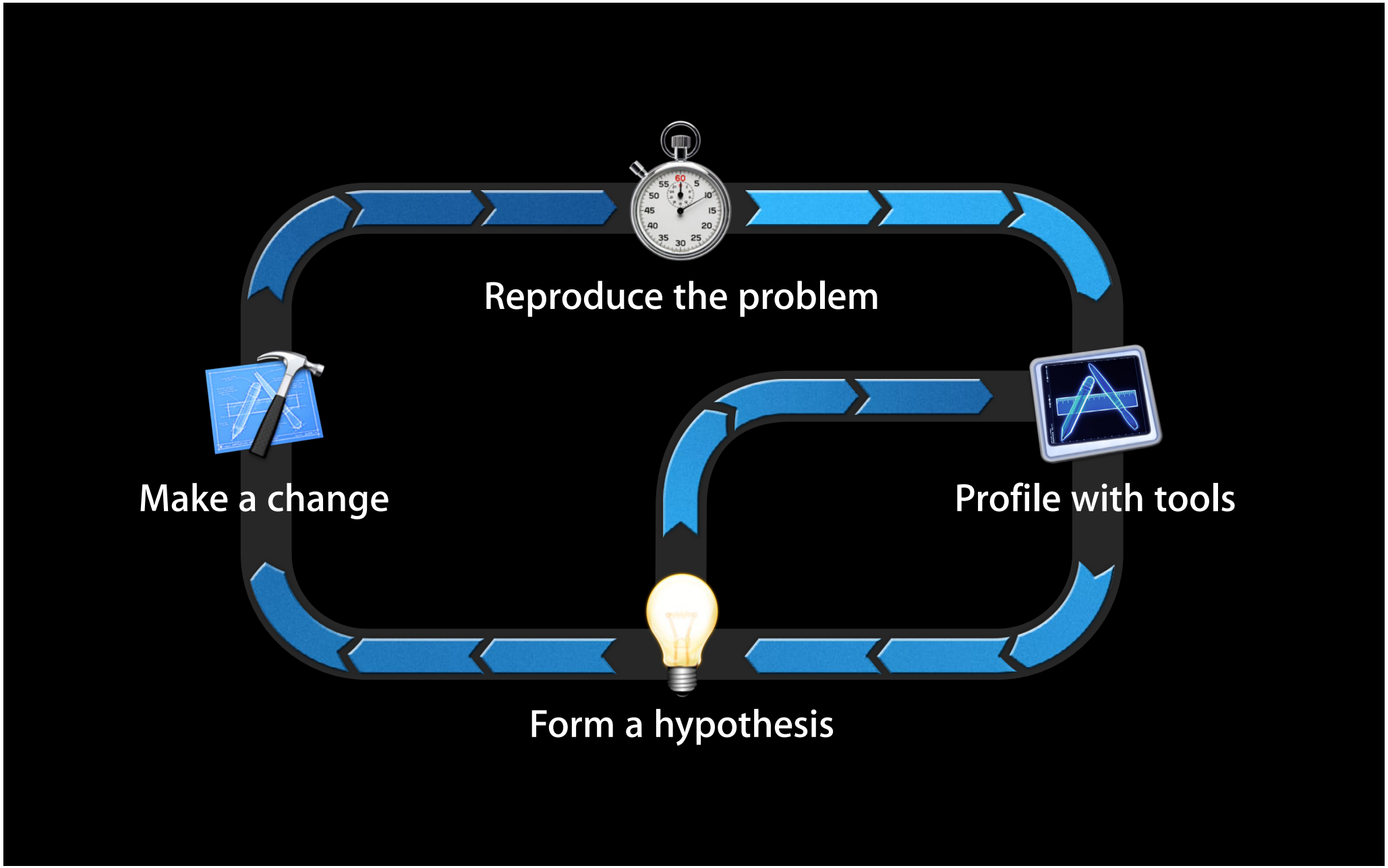


  - @autoreleasepool can help in Objective-C code

Finding Memory Issues

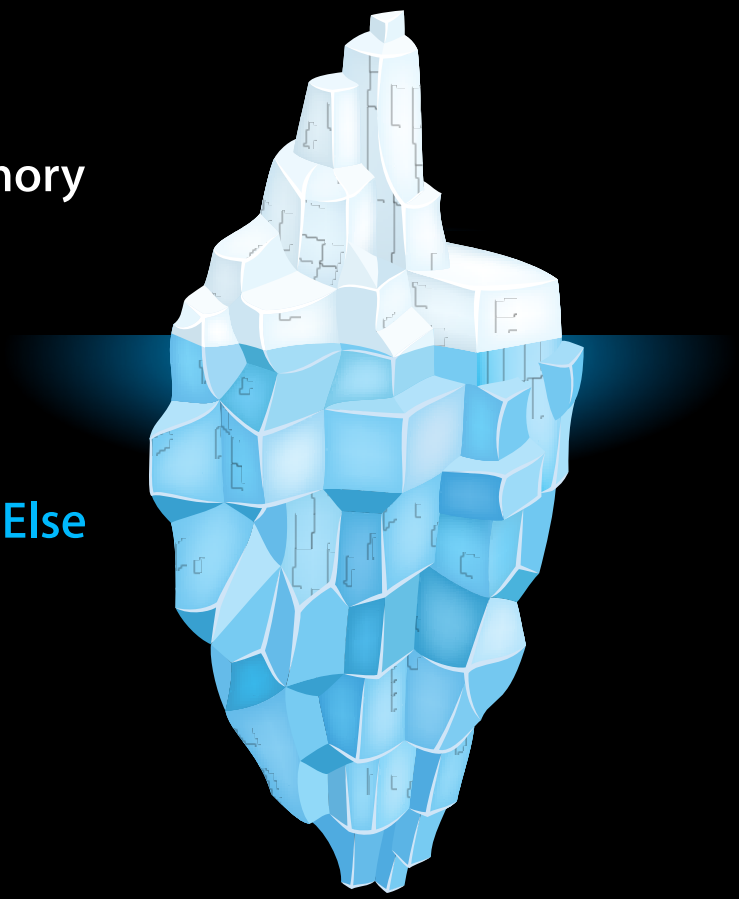


Daniel Delwood
Software Engineer


Reproduce the problem
Profile with tools
Form a hypothesis
Make a change

Memory Footprint

Heap Memory



Everything Else

Most Dirty Memory is
Related to the Heap

Reducing Memory Usage

What you can do

Reducing Memory Usage

What you can do

- Understand your view hierarchy
  - The more pixels you draw…

Reducing Memory Usage

What you can do

- Understand your view hierarchy
  - The more pixels you draw…
- Avoid recurring heap growth
  - Doesn't matter if the objects are small

Avoiding Memory Growth

Top three to look for

Avoiding Memory Growth

Top three to look for

- Leaked memory
  - Inaccessible—no more pointers to it
  - Can't ever be used again

Avoiding Memory Growth

Top three to look for

- Leaked memory
  - Inaccessible—no more pointers to it
  - Can't ever be used again
- Abandoned memory
  - Still referenced, but wasted
  - Won't ever be used again

Avoiding Memory Growth

Top three to look for

- Leaked memory
  - Inaccessible—no more pointers to it
  - Can't ever be used again
- Abandoned memory
  - Still referenced, but wasted
  - Won't ever be used again
- Cached memory
  - Referenced and waiting
  - May never be used again

Memory Growth

How to detect it

- Memory shouldn't grow without bound when repeating an operation

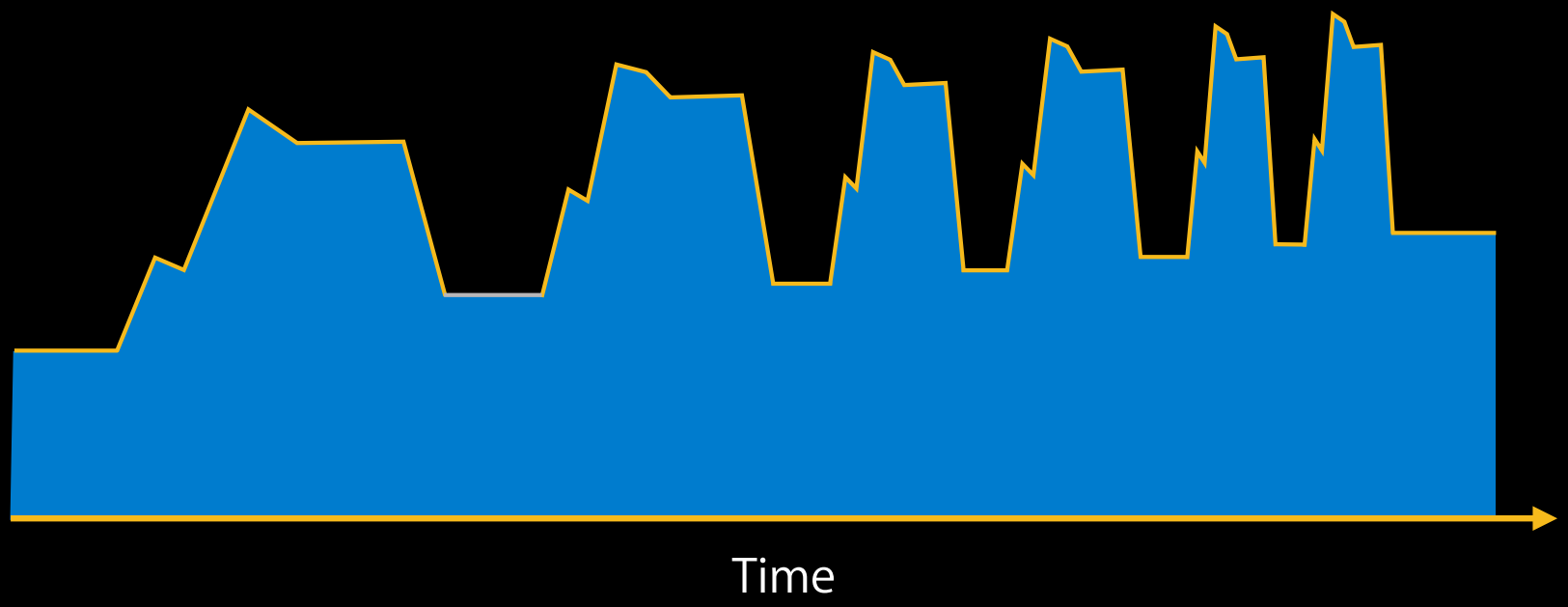
Memory Growth

How to detect it

- Memory shouldn't grow without bound when repeating an operation
  - For example
    - Pushing and popping a view controller
    - Scrolling in a table view
    - Performing a database search

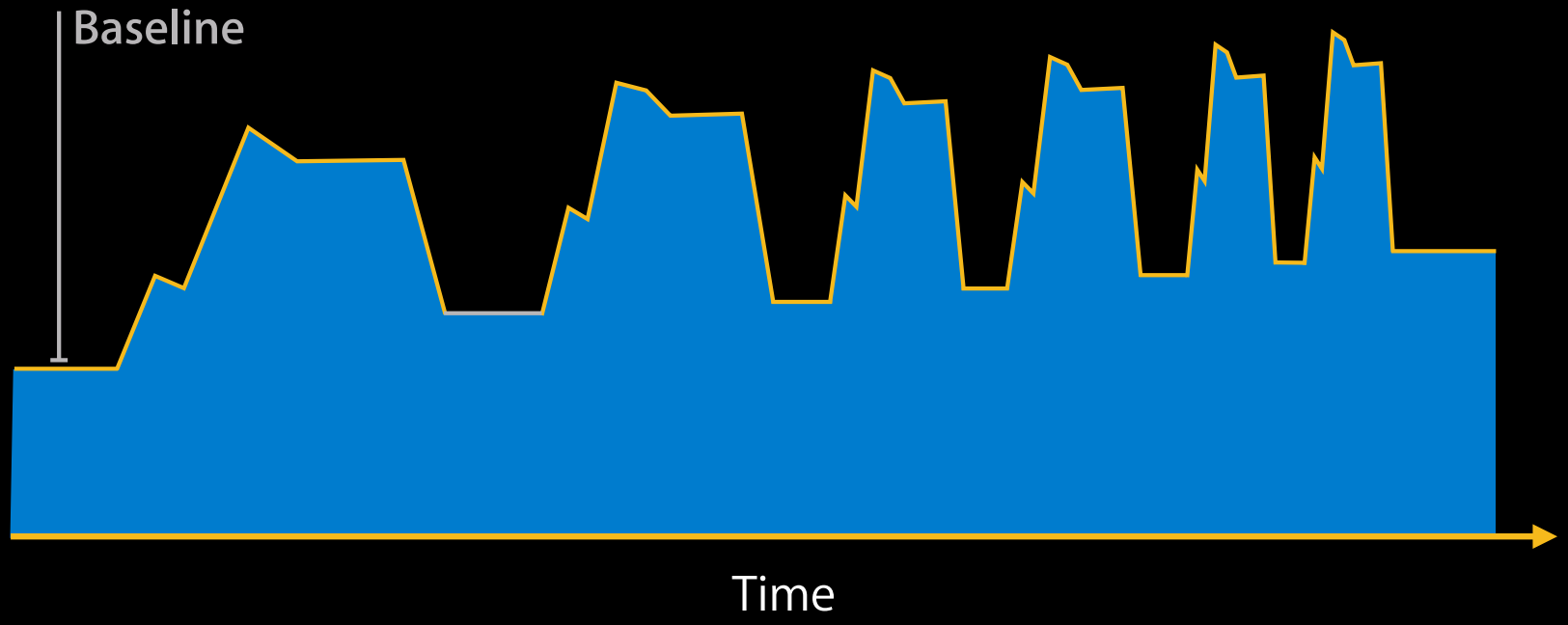
Avoiding Memory Growth

Repetition reveals waste



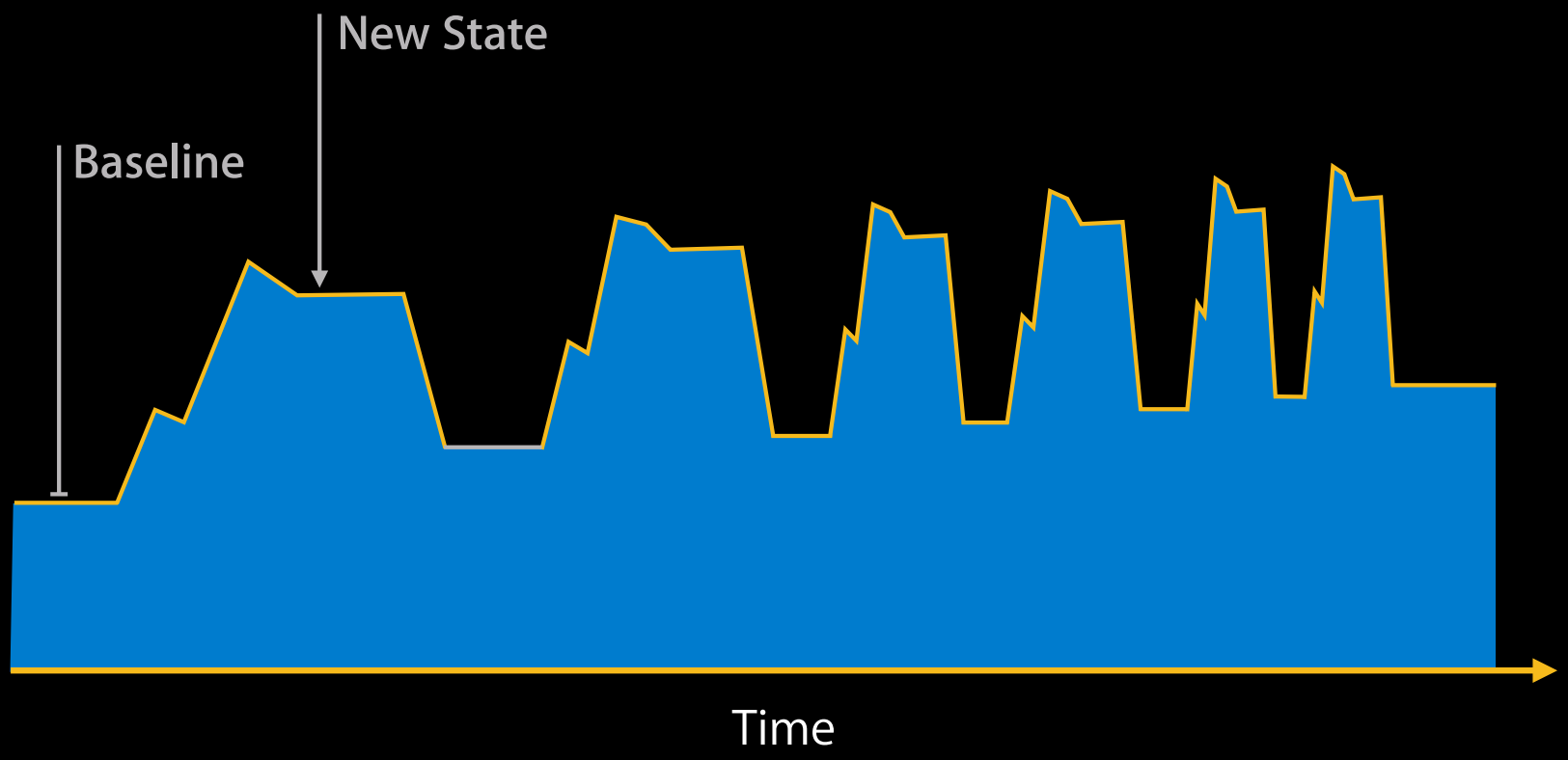
Avoiding Memory Growth

Repetition reveals waste



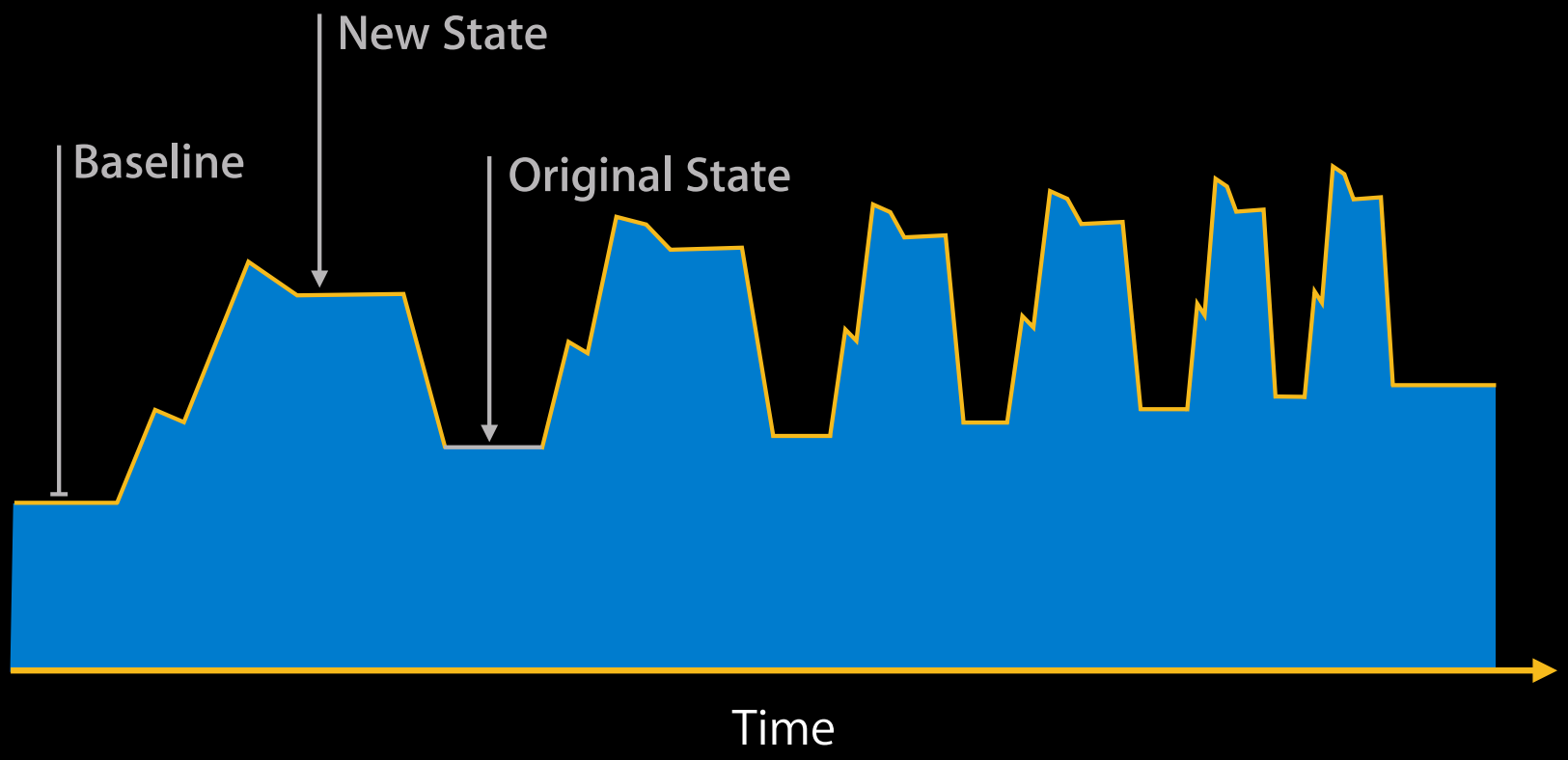
Avoiding Memory Growth

Repetition reveals waste



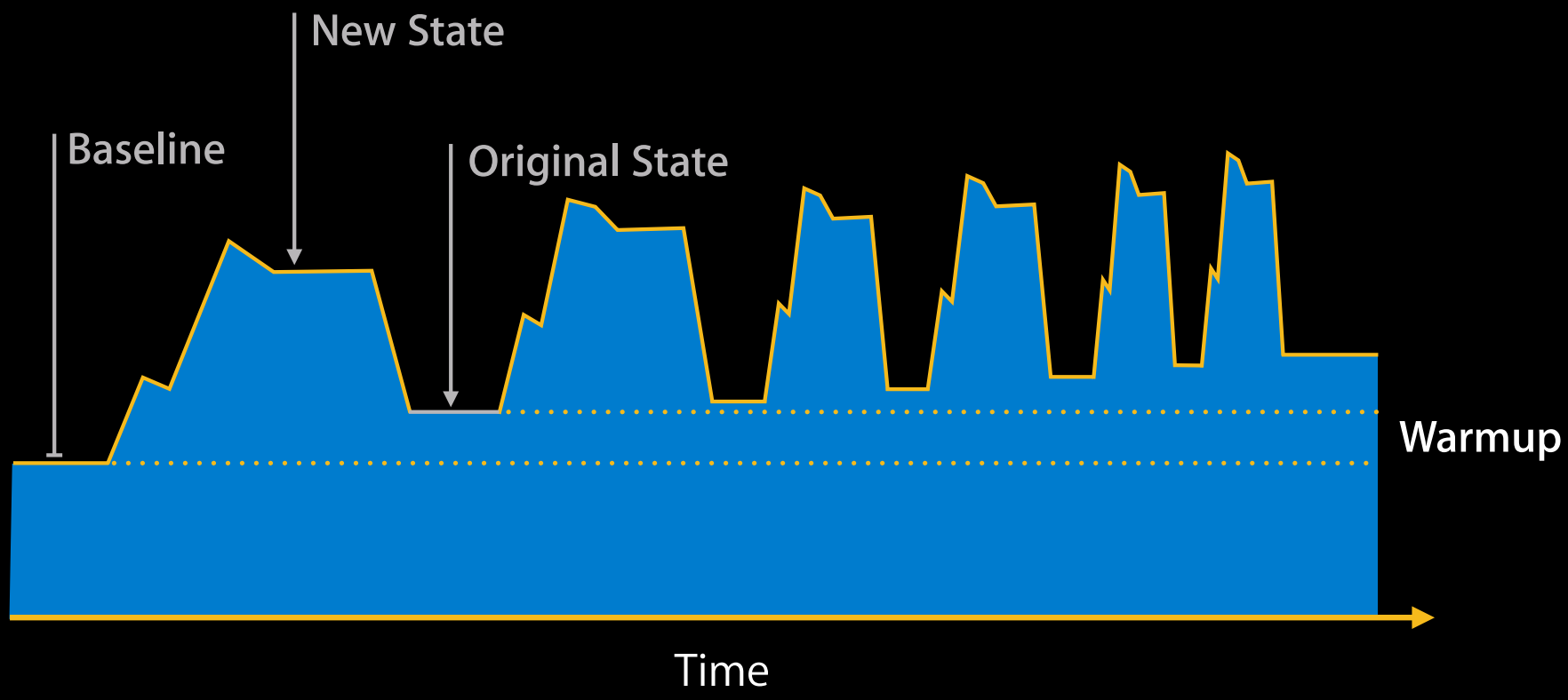
Avoiding Memory Growth

Repetition reveals waste



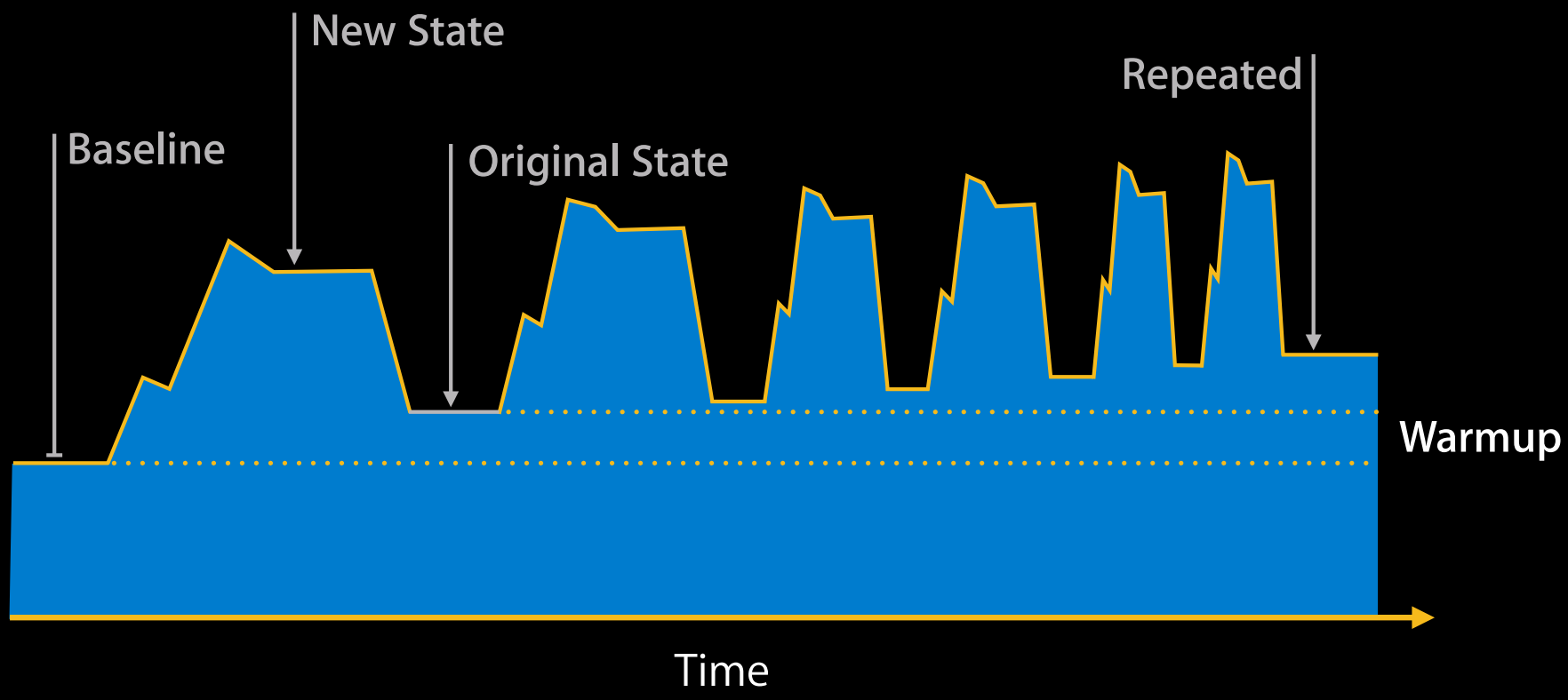
Avoiding Memory Growth

Repetition reveals waste



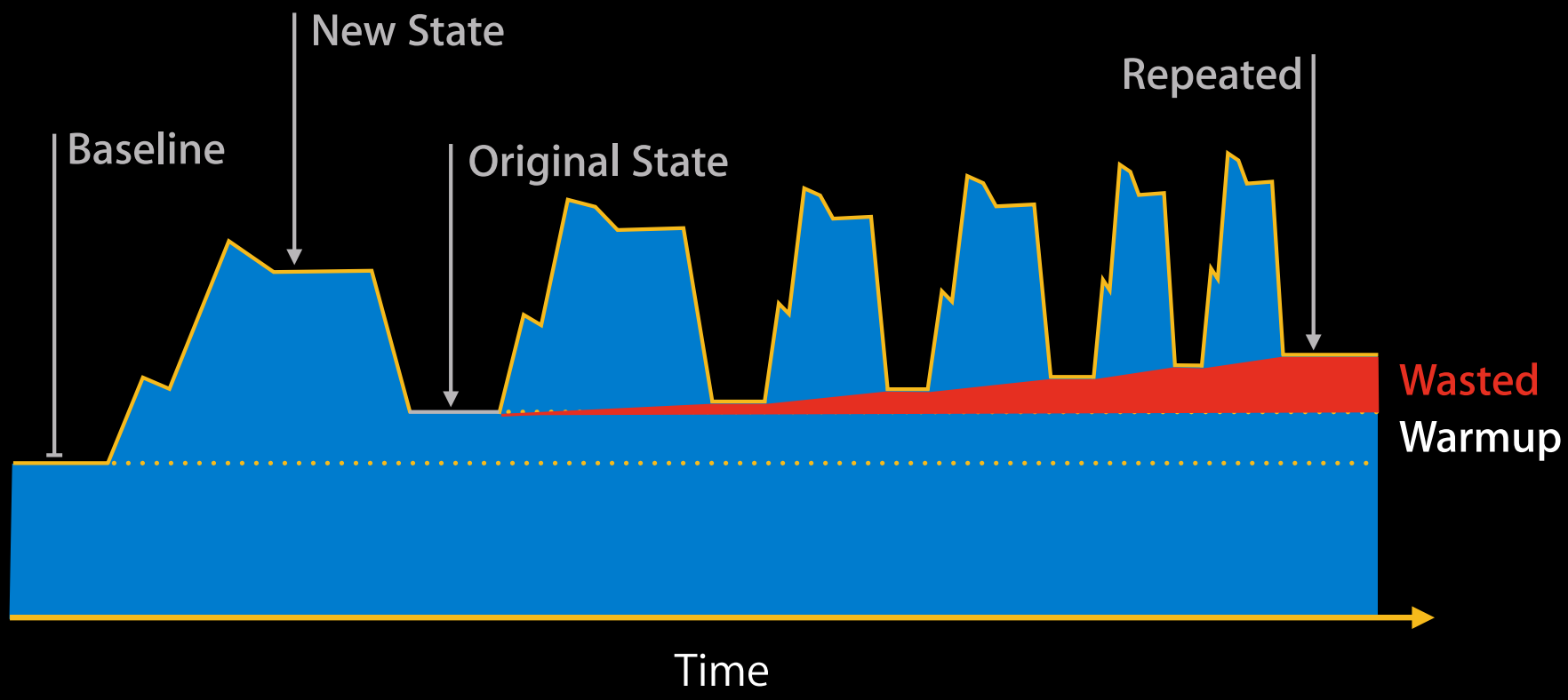
Avoiding Memory Growth

Repetition reveals waste



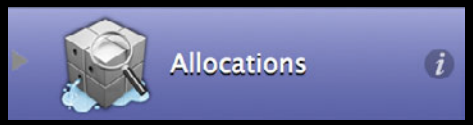
Avoiding Memory Growth

Repetition reveals waste



Allocations Instrument

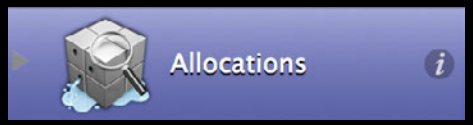
A targeted tool



Allocations Instrument

A targeted tool

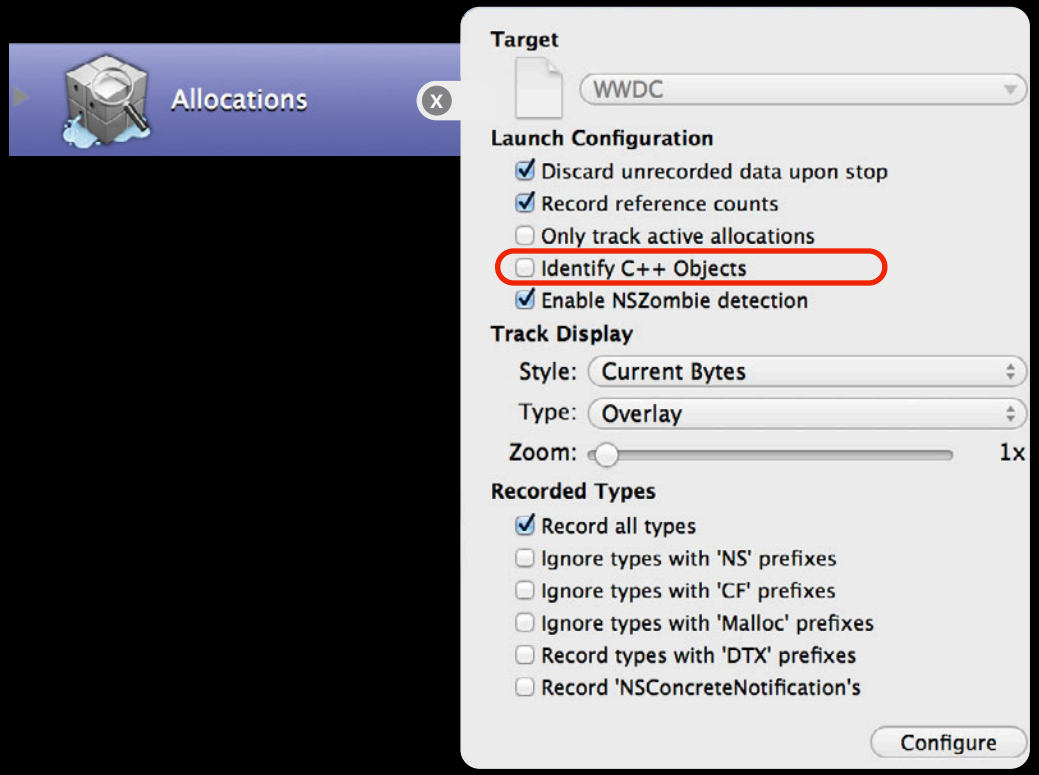
- Tracks all heap allocations



Allocations Instrument

A targeted tool

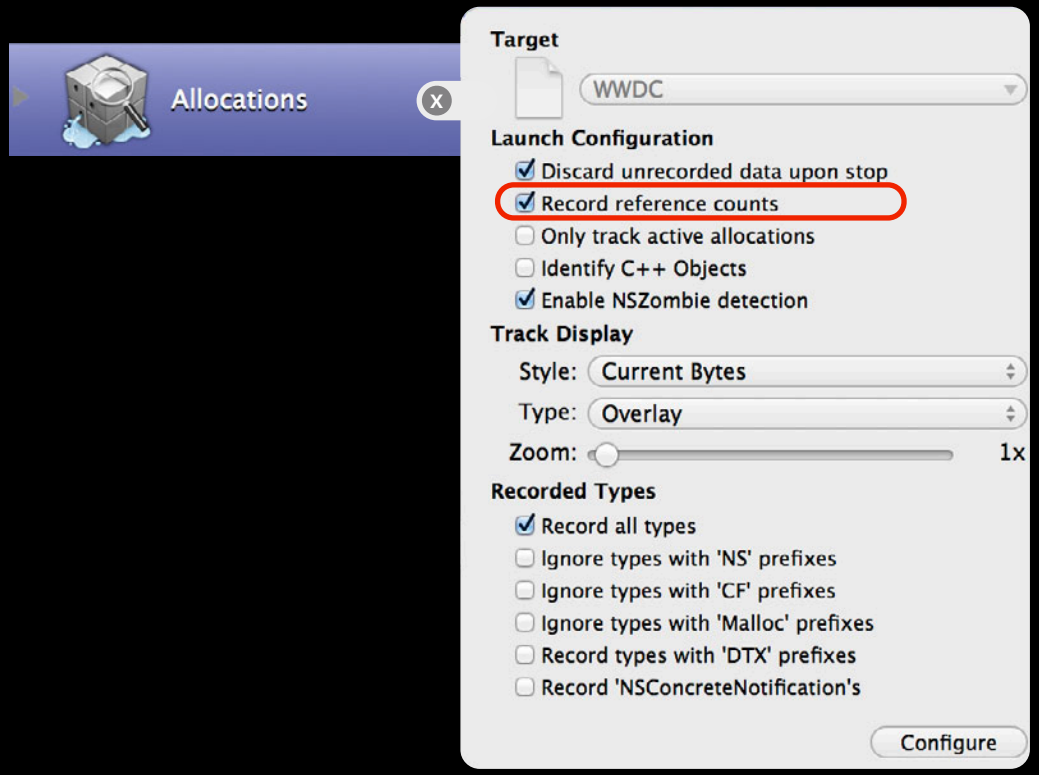
- Tracks all heap allocations
- Objective-C, C++ objects



Allocations Instrument

A targeted tool

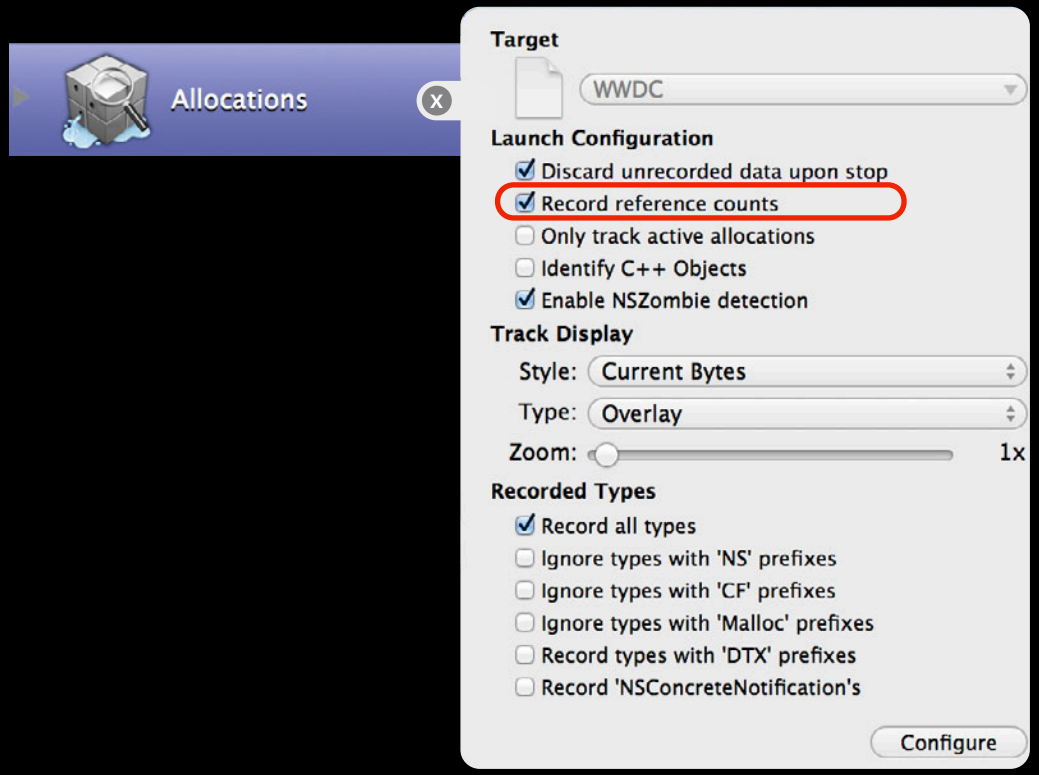
- Tracks all heap allocations
- Objective-C, C++ objects
- Malloc, Free, Retain, Release, Autorelease



Allocations Instrument

A targeted tool

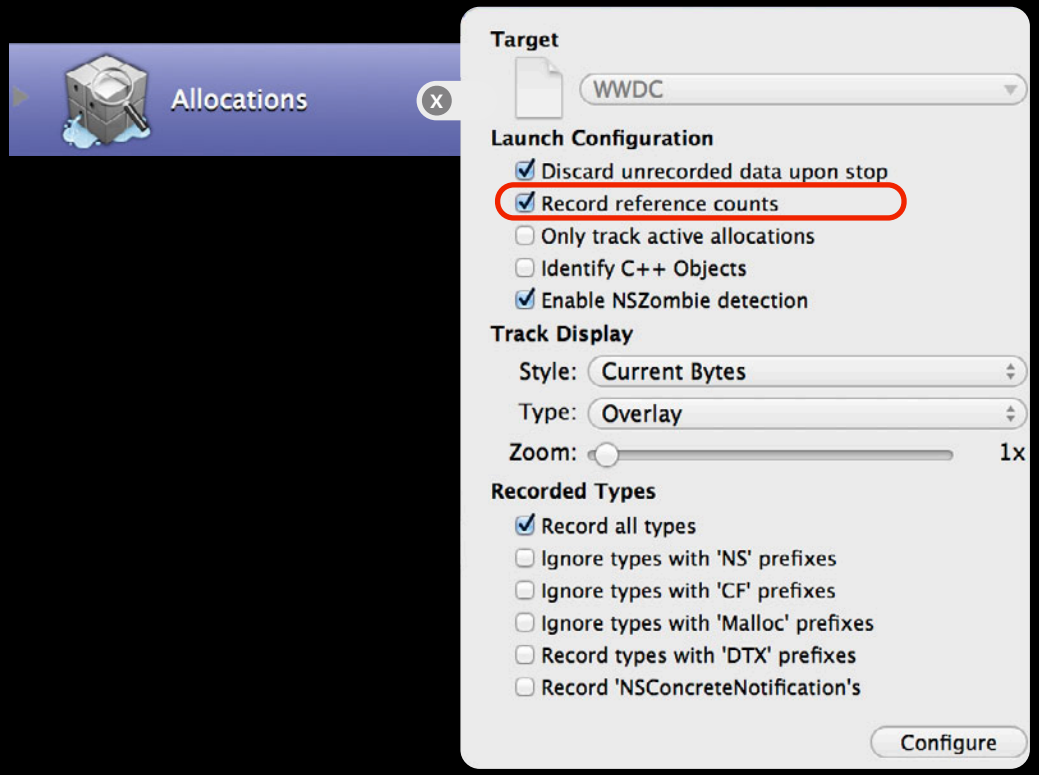
- Tracks all heap allocations
- Objective-C, C++ objects
- Malloc, Free, Retain, Release, Autorelease
- Statistics by allocation type



Allocations Instrument

A targeted tool

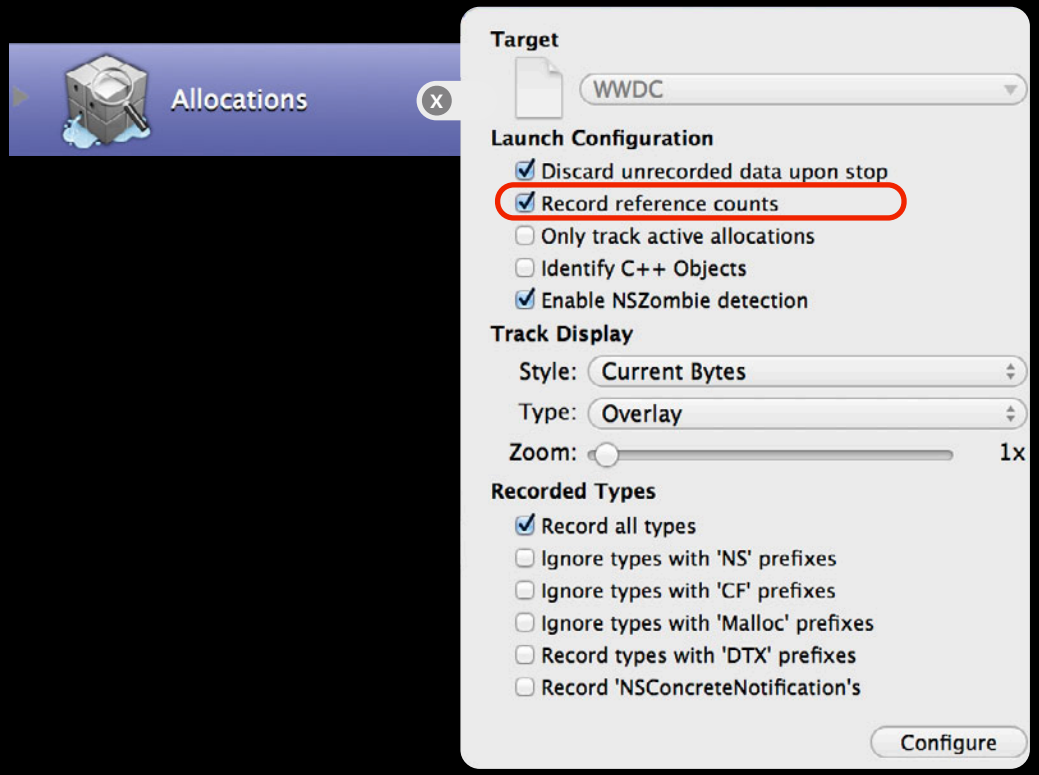
- Tracks all heap allocations
- Objective-C, C++ objects
- Malloc, Free, Retain, Release, Autorelease
- Statistics by allocation type
- Call Trees



Allocations Instrument

A targeted tool

- Tracks all heap allocations
- Objective-C, C++ objects
- Malloc, Free, Retain, Release, Autorelease
- Statistics by allocation type
- Call Trees
- Heap snapshots



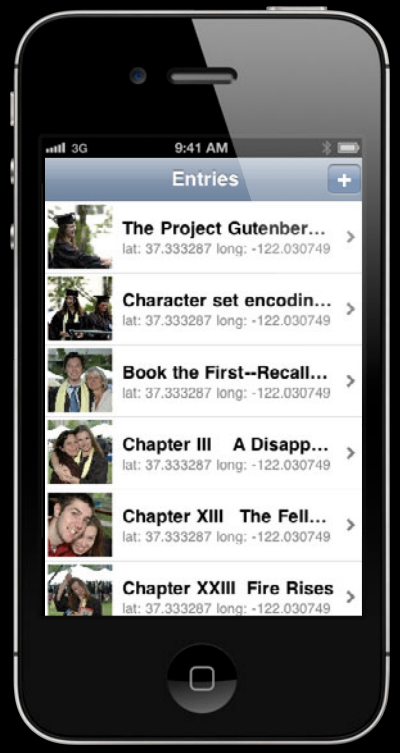
Heap Snapshots

A practical example

Heap Snapshots

A practical example

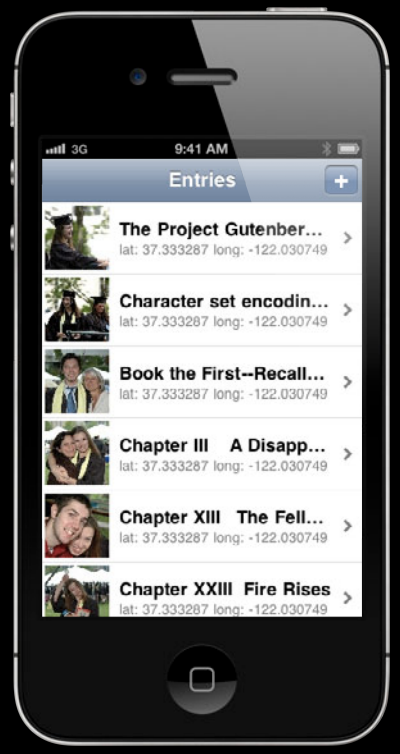
1. Launch the app



Heap Snapshots

A practical example

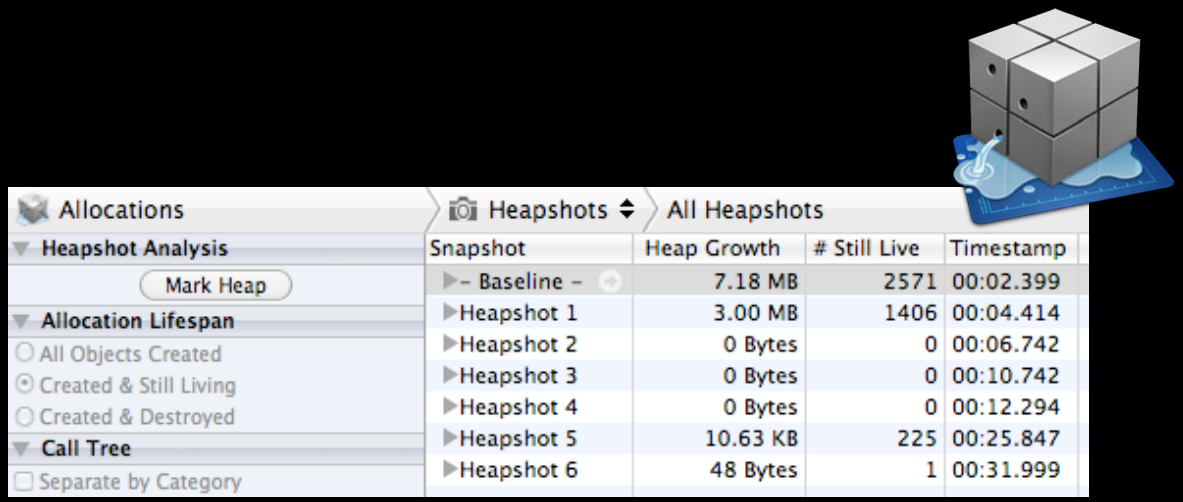
1. Launch the app
2. Push and pop a view controller

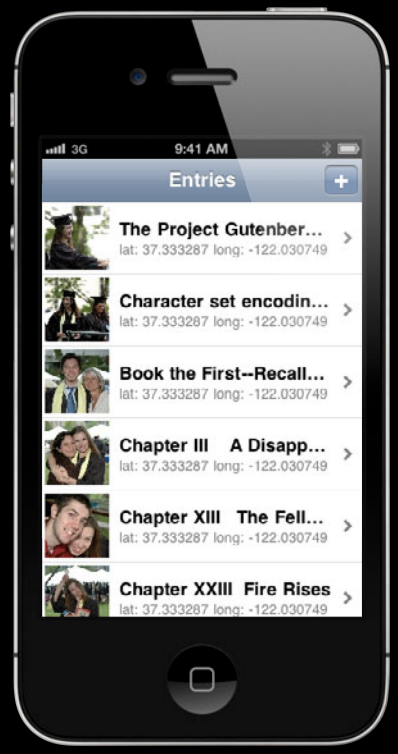


Heap Snapshots

A practical example

1. Launch the app
2. Push and pop a view controller
3. Take a snapshot of the heap





Heap Snapshots

A practical example

1. Launch the app
2. Push and pop a view controller
3. Take a snapshot of the heap

Repeat!

Allocations | Heapshots | All Heapshots

Heapshot Analysis

Mark Heap

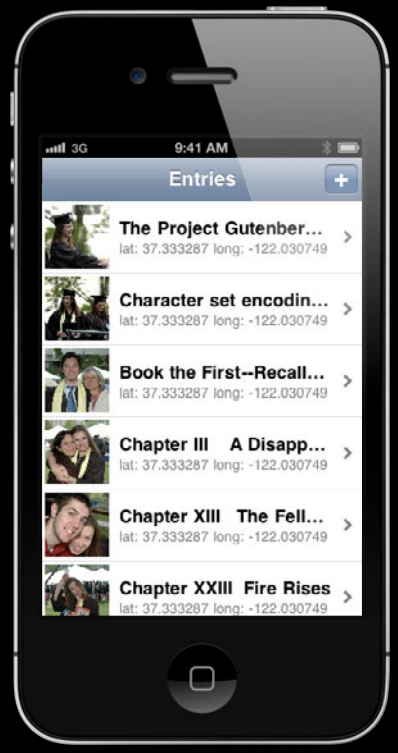
Allocation Lifespan

- All Objects Created
- Created & Still Living
- Created & Destroyed

Call Tree

- Separate by Category

Snapshot	Heap Growth	# Still Live	Timestamp
– Baseline –	7.18 MB	2571	00:02.399
Heapshot 1	3.00 MB	1406	00:04.414
Heapshot 2	0 Bytes	0	00:06.742
Heapshot 3	0 Bytes	0	00:10.742
Heapshot 4	0 Bytes	0	00:12.294
Heapshot 5	10.63 KB	225	00:25.847
Heapshot 6	48 Bytes	1	00:31.999



Heap Snapshots

A practical example

1. Launch the app
2. Push and pop a view controller
3. Take a snapshot of the heap

Repeat!

Allocations | Heapshots | All Heapshots

Heapshot Analysis

Mark Heap

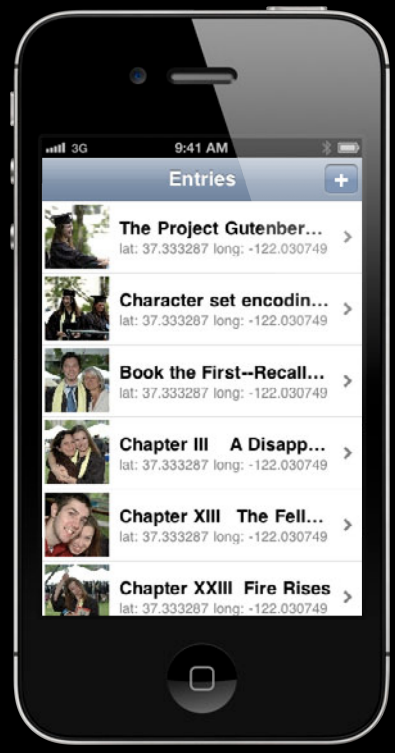
Allocation Lifespan

- All Objects Created
- Created & Still Living
- Created & Destroyed

Call Tree

- Separate by Category

Snapshot	Heap Growth	# Still Live	Timestamp
– Baseline –	7.18 MB	2571	00:02.399
Heapshot 1	3.00 MB	1406	00:04.414
Heapshot 2	0 Bytes	0	00:06.742
Heapshot 3	0 Bytes	0	00:10.742
Heapshot 4	0 Bytes	0	00:12.294
Heapshot 5	10.63 KB	225	00:25.847
Heapshot 6	48 Bytes	1	00:31.999



Demo

Detecting and fixing memory growth

Tools Tips and Tricks

Memory Tips

Getting to know your app

Memory Tips

Getting to know your app

- Pay attention to objects holding resources
  - UIImage, UIViewController, NSOperation, etc.
  - Anything that wraps large data or many objects

Memory Tips

Getting to know your app

- Pay attention to objects holding resources
  - UIImage, UIViewController, NSOperation, etc.
  - Anything that wraps large data or many objects
- Track your objects
  - Filter for class name prefix
  - Validate expected patterns

Memory Tips

Getting to know your app

- Pay attention to objects holding resources
  - UIImage, UIViewController, NSOperation, etc.
  - Anything that wraps large data or many objects
- Track your objects
  - Filter for class name prefix
  - Validate expected patterns
- Simulate memory warnings

Saving Time

Memory bugs are expensive

- Switch to ARC!
  - Allows you to think about object relationships

Saving Time

Memory bugs are expensive

- Switch to ARC!
  - Allows you to think about object relationships
- Run the static analyzer

Saving Time

Memory bugs are expensive

- Switch to ARC!
  - Allows you to think about object relationships
- Run the static analyzer
- Use the Leaks instrument
  - Make one fix at a time, re-run

Fixing Leaks

Plugging the holes

- Retain cycles
  - Use the “Cycles & Roots” view
  - Be aware of ^block captures

Fixing Leaks

Plugging the holes

- Retain cycles
  - Use the “Cycles & Roots” view
  - Be aware of ^block captures

```
_observer = [center addObserverForName:@“MyNotification”
                object:nil
                 queue:[NSOperationQueue mainQueue]
            usingBlock:^(NSNotification *note) {

    self.document = [note object]
}];
```

Fixing Leaks

Plugging the holes

- Retain cycles
  - Use the “Cycles & Roots” view
  - Be aware of ^block captures

```
_observer = [center addObserverForName:@“MyNotification”
                                object:nil
                                 queue:[NSOperationQueue mainQueue]
                            usingBlock:^(NSNotification *note) {
    self.document = [note object]
}];
```

Fixing Leaks

Plugging the holes

- Retain cycles
  - Use the “Cycles & Roots” view
  - Be aware of ^block captures

```
__strong id capturedSelf = self;
_observer = [center addObserverForName:@“MyNotification”
                                object:nil
                                 queue:[NSOperationQueue mainQueue]
                            usingBlock:^(NSNotification *note) {
    capturedSelf.document = [note object]
}];
```

Fixing Leaks

Plugging the holes

- Retain cycles
  - Use the "Cycles & Roots" view
  - Be aware of ^block captures

```
_observer = [center addObserverForName:@"MyNotification"
                object:nil
                 queue:[NSOperationQueue mainQueue]
            usingBlock:^(NSNotification *note) {

    _document = [note object]
}];
```

Fixing Leaks

Plugging the holes

- Retain cycles
  - Use the “Cycles & Roots” view
  - Be aware of ^block captures

```
_observer = [center addObserverForName:@“MyNotification”
                                object:nil
                                 queue:[NSOperationQueue mainQueue]
                            usingBlock:^(NSNotification *note) {
    _document = [note object]
}];
```

Fixing Leaks

Plugging the holes

- Retain cycles
  - Use the "Cycles & Roots" view
  - Be aware of ^block captures

```
__strong id capturedSelf = self;
_observer = [center addObserverForName:@"MyNotification"
                                object:nil
                                 queue:[NSOperationQueue mainQueue]
                            usingBlock:^(NSNotification *note) {
    capturedSelf->_document = [note object]
}];
```

Fixing Leaks

Plugging the holes

- Retain cycles
  - Use the “Cycles & Roots” view
  - Be aware of ^block captures

```
_observer = [center addObserverForName:@“MyNotification”
                                object:nil
                                 queue:[NSOperationQueue mainQueue]
                            usingBlock:^(NSNotification *note) {
    self.document = [note object]
}];
```

Fixing Leaks

Plugging the holes

- Retain cycles
  - Use the "Cycles & Roots" view
  - Be aware of ^block captures

```
__weak id weakNotifiedSelf = self;
_observer = [center addObserverForName:@"MyNotification"
                                object:nil
                                 queue:[NSOperationQueue mainQueue]
                            usingBlock:^(NSNotification *note) {
    self.document = [note object]
}];
```

Fixing Leaks

Plugging the holes

- Retain cycles
  - Use the “Cycles & Roots” view
  - Be aware of ^block captures

```
__weak id weakNotifiedSelf = self;
_observer = [center addObserverForName:@“MyNotification”
                                object:nil
                                 queue:[NSOperationQueue mainQueue]
                            usingBlock:^(NSNotification *note) {
    weakNotifiedSelf.document = [note object]
}];
```

Fixing Leaks

Plugging the holes

- Incorrect retain/release
  - Possible with incorrect ARC bridging
  - Focus on the reference counting history

__NSMallocBlock__	1	0xb0a3550
NSConcreteMutableData	1	0x9b9acd0
__NSMallocBlock__	1	0xb0a34d0

NSConcreteMutableData	Malloc	1	00:01.893.340
NSConcreteMutableData	Retain	2	00:01.893.342
NSConcreteMutableData	Retain	3	00:01.893.344
NSConcreteMutableData	Release	2	00:01.893.344
NSConcreteMutableData	Release	1	00:01.993.307

Memory-related Crashes

In case of emergency

Memory-related Crashes

In case of emergency

```
Exception Type:  EXC_BAD_ACCESS (SIGBUS)
Exception Codes: KERN_PROTECTION_FAILURE at 0x00000010
Crashed Thread:  0

Thread 0 Crashed:
0  libobjc.dylib  0x0000286c objc_msgSend + 16
1  Foundation     0x0001219c -[NSString stringByAppendingFormat:] + 84
2  Reader         0x000031d4 -[RootViewController tableView:cellForRowAtIndexPath:] + 32
3  UIKit          0x0007e18c -[UITableView _createPreparedCellForGlobalRow:withIndexPath:] + 492
4  UIKit          0x0007ded8 -[UITableView _createPreparedCellForGlobalRow:] + 28
5  UIKit          0x000530e2 -[UITableView(_UITableViewPrivate) _updateVisibleCellsNow:] + 930
6  UIKit          0x000514da -[UITableView layoutSubviews] + 134
7  UIKit          0x0000f874 -[UIView(CALayerDelegate) _layoutSublayersOfLayer:] + 20
8  CoreFoundation 0x000277f8 -[NSObject(NSObject) performSelector:withObject:] + 16
```

Memory-related Crashes

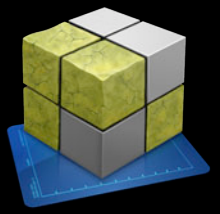
In case of emergency

```
Exception Type:  EXC_BAD_ACCESS (SIGBUS)
Exception Codes: KERN_PROTECTION_FAILURE at 0x00000010
Crashed Thread:  0

Thread 0 Crashed:
0 libobjc.dylib 0x0000286c objc_msgSend + 16
1  Foundation     0x0001219c -[NSString stringByAppendingFormat:] + 84
2  Reader         0x000031d4 -[RootViewController tableView:cellForRowAtIndexPath:] + 32
3  UIKit          0x0007e18c -[UITableView _createPreparedCellForGlobalRow:withIndexPath:] + 492
4  UIKit          0x0007ded8 -[UITableView _createPreparedCellForGlobalRow:] + 28
5  UIKit          0x000530e2 -[UITableView(_UITableViewPrivate) _updateVisibleCellsNow:] + 930
6  UIKit          0x000514da -[UITableView layoutSubviews] + 134
7  UIKit          0x0000f874 -[UIView(CALayerDelegate) _layoutSublayersOfLayer:] + 20
8  CoreFoundation 0x000277f8 -[NSObject(NSObject) performSelector:withObject:] + 16
```

Memory Crashers

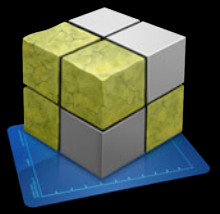
In case of emergency



- Zombies template in iOS Simulator

Memory Crashers

In case of emergency



- Zombies template in iOS Simulator
- Messages to deallocated objects

Memory Crashers

In case of emergency

- Zombies template in iOS Simulator
- Messages to deallocated objects
  - NSNotificationCenter and Key-Value Observing

Memory Crashers

In case of emergency

- Zombies template in iOS Simulator
- Messages to deallocated objects
  - NSNotificationCenter and Key-Value Observing
  - Incorrect bridging

Memory Crashers

In case of emergency

- Zombies template in iOS Simulator
- Messages to deallocated objects
  - NSNotificationCenter and Key-Value Observing
  - Incorrect bridging
  - __unsafe_unretained references

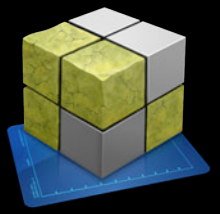
Memory Crashers

In case of emergency

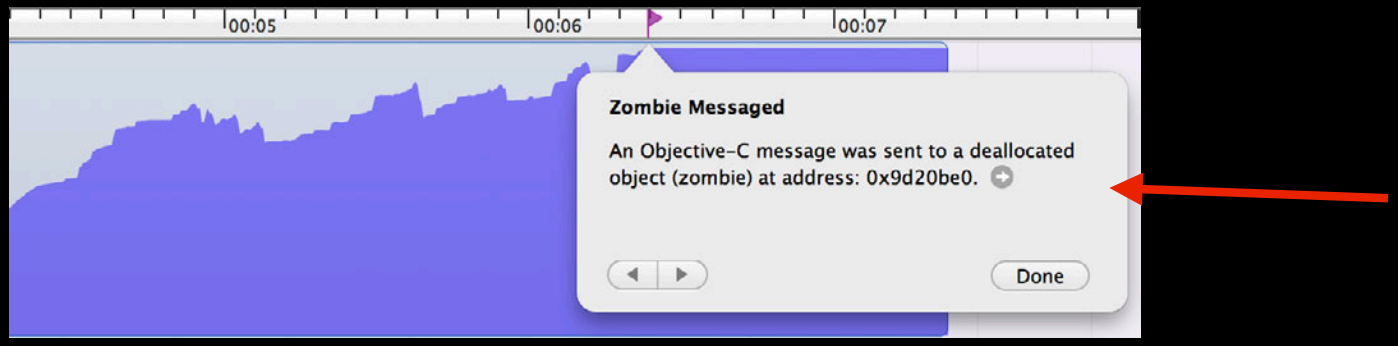
- Zombies template in iOS Simulator
- Messages to deallocated objects
  - NSNotificationCenter and Key-Value Observing
  - Incorrect bridging
  - __unsafe_unretained references
  - __autoreleasing NSError* and @autoreleasepool

Memory Crashers

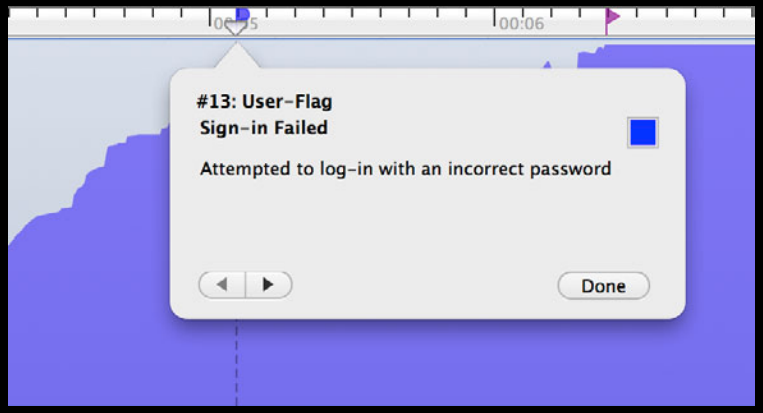
In case of emergency



- Zombies template in iOS Simulator
- Messages to deallocated objects
  - NSNotificationCenter and Key-Value Observing
  - Incorrect bridging
  - __unsafe_unretained references
  - __autoreleasing NSError* and @autoreleasepool

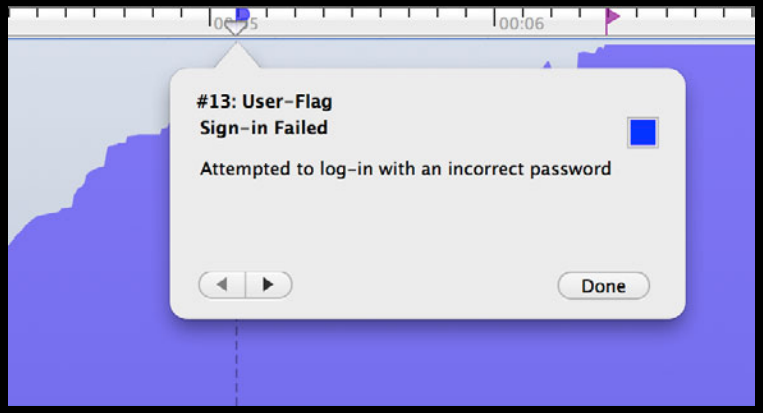


Tracing Tips



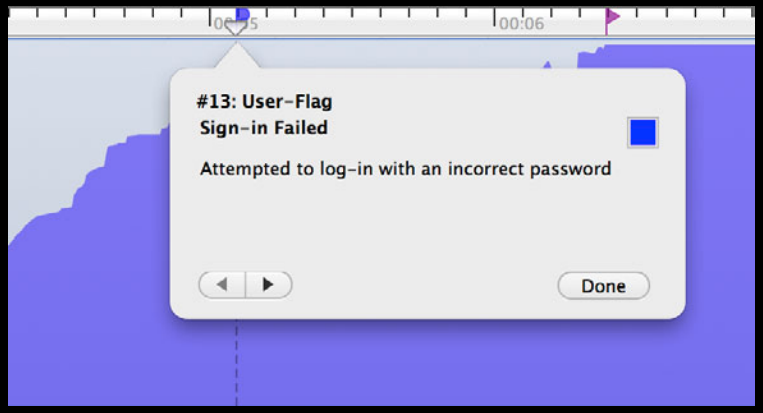
Tracing Tips

- Make notes with each trace



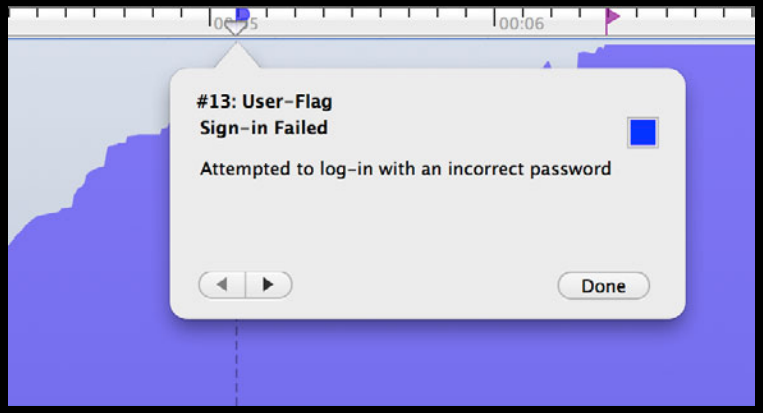
Tracing Tips

- Make notes with each trace
  - Flags comments are invaluable later



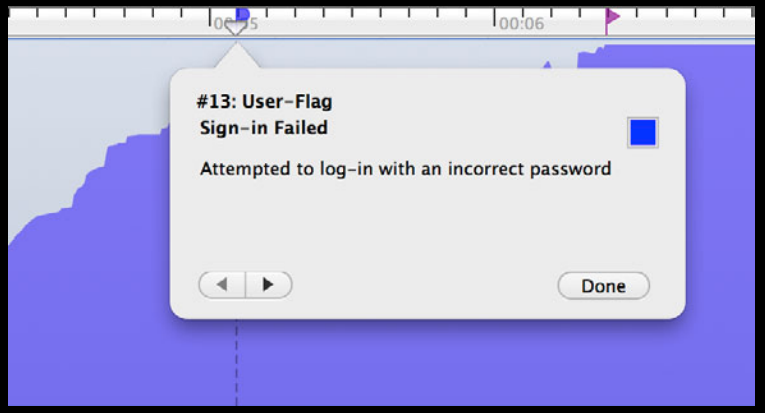
Tracing Tips

- Make notes with each trace
  - Flags comments are invaluable later
- Filter to specific time intervals



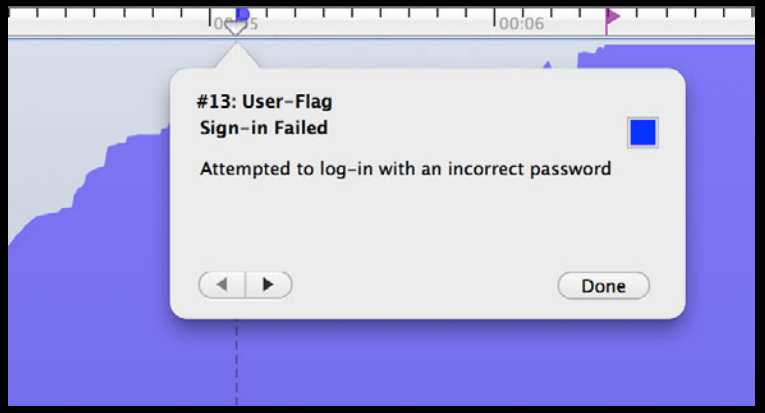
Tracing Tips

- Make notes with each trace
  - Flags comments are invaluable later
- Filter to specific time intervals
- Be conscious of snapshot intervals



Tracing Tips

- Make notes with each trace
  - Flags comments are invaluable later
- Filter to specific time intervals
- Be conscious of snapshot intervals
  - Leaks and VM Tracker will cause app pauses



Summary

Summary

- Great apps are efficient

Summary

- Great apps are efficient
- Use as little as possible

Summary

- **Great apps are efficient**
- Use as little as possible
  - …but consider the user experience implications

Summary

- **Great apps are efficient**
- Use as little as possible
  - …but consider the user experience implications
- Measure, change, and iterate

More Information

Michael Jurewitz
Developer Tools & Performance Evangelist
jury@apple.com

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

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

Related Sessions

Learning Instruments	Presidio Wednesday 4:30PM
iOS App Performance: Responsiveness	Presidio Thursday 11:30AM
iOS App Performance: Graphics and Animations	Presidio Thursday 3:15PM
Polishing Your Interface Rotations	Mission Thursday 4:30PM
Adopting Automatic Reference Counting	Nob HIll Friday 11:30AM
Asynchronous Design Patterns with Blocks, GCD, and XPC	Pacific Heights Friday 9:00AM

Labs

OS X Performance Lab	Developer Tools Lab A Friday 9:00AM


WWDC2012