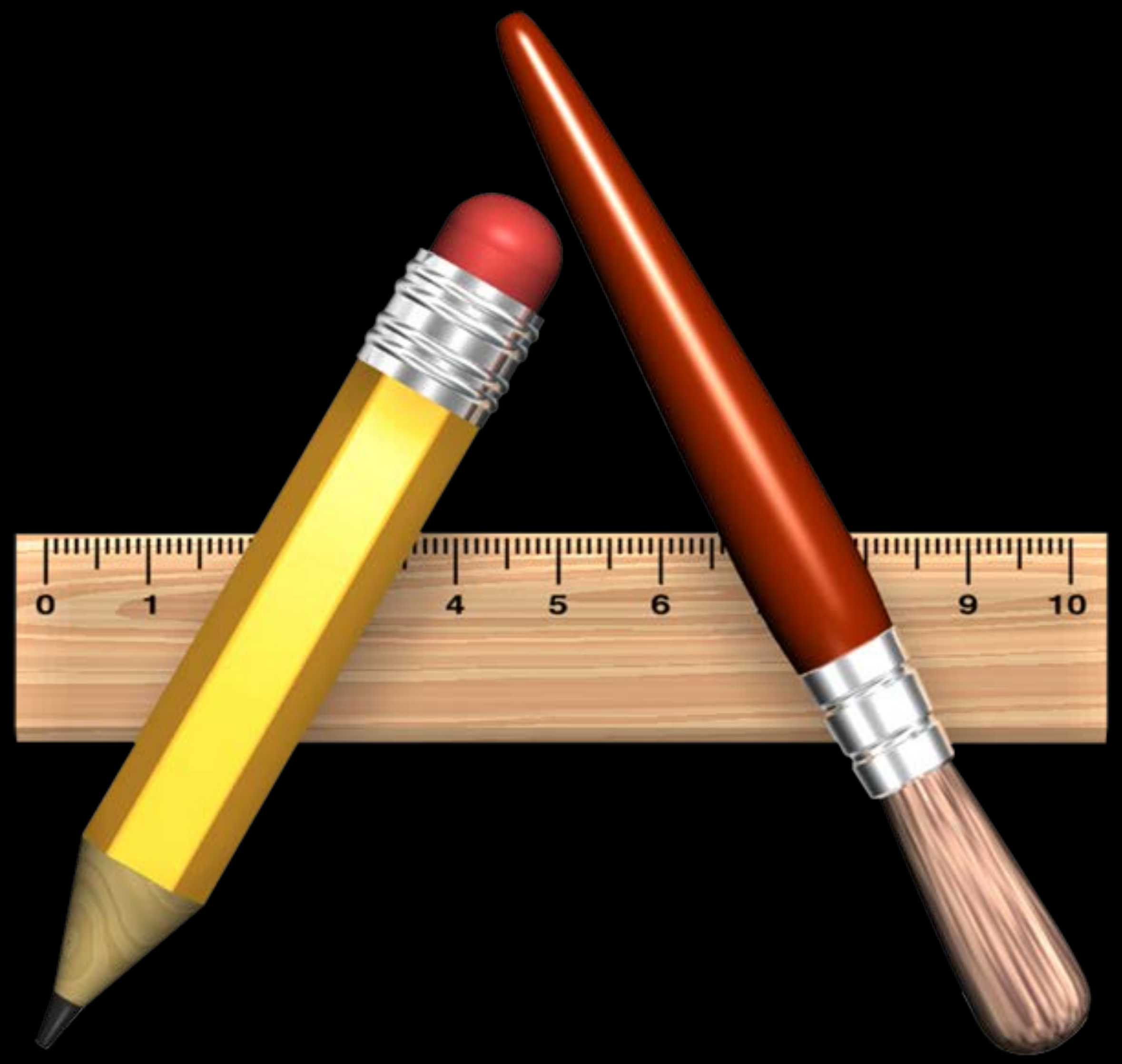
Fixing Memory Issues

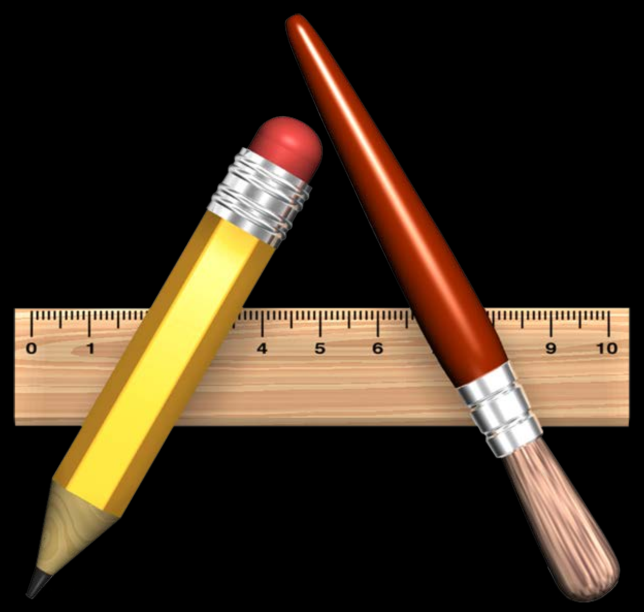
iOS and OS X techniques

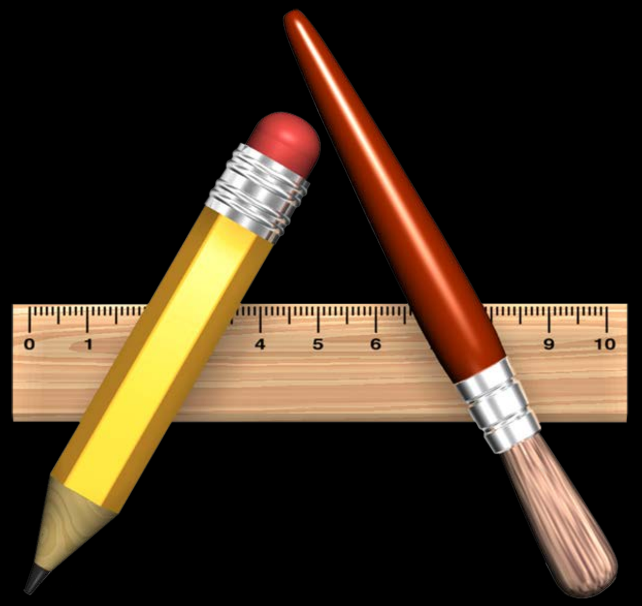
Session 410

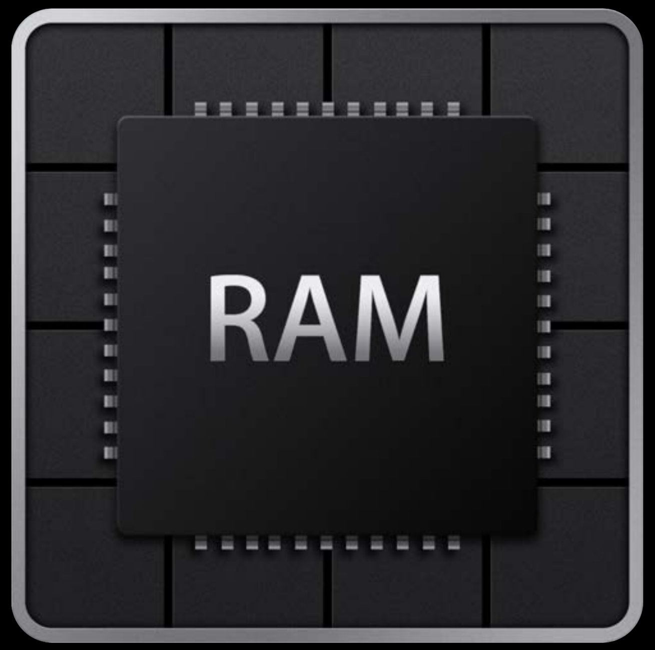
Kate Stone
Software Behavioralist

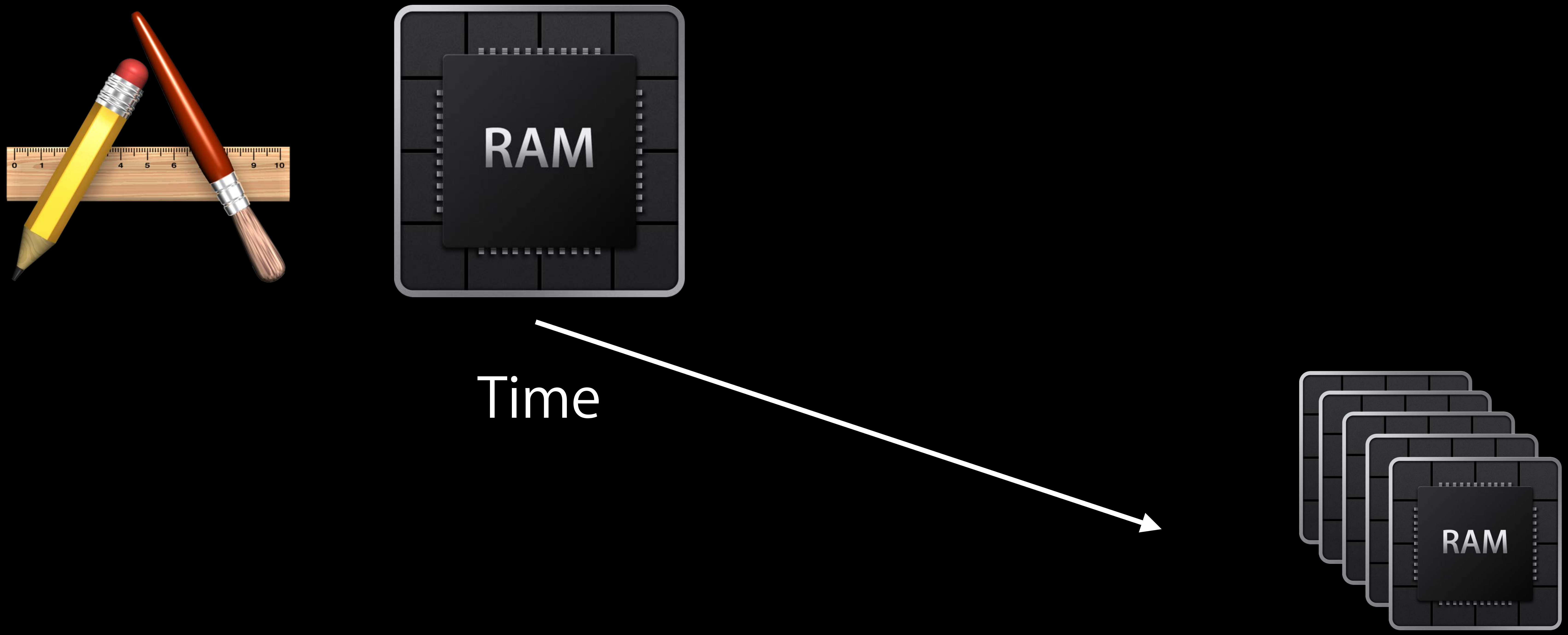
Daniel Delwood
Software Radiologist


0
1
4
5
6
9
10


0 1 4 5 6 9 10

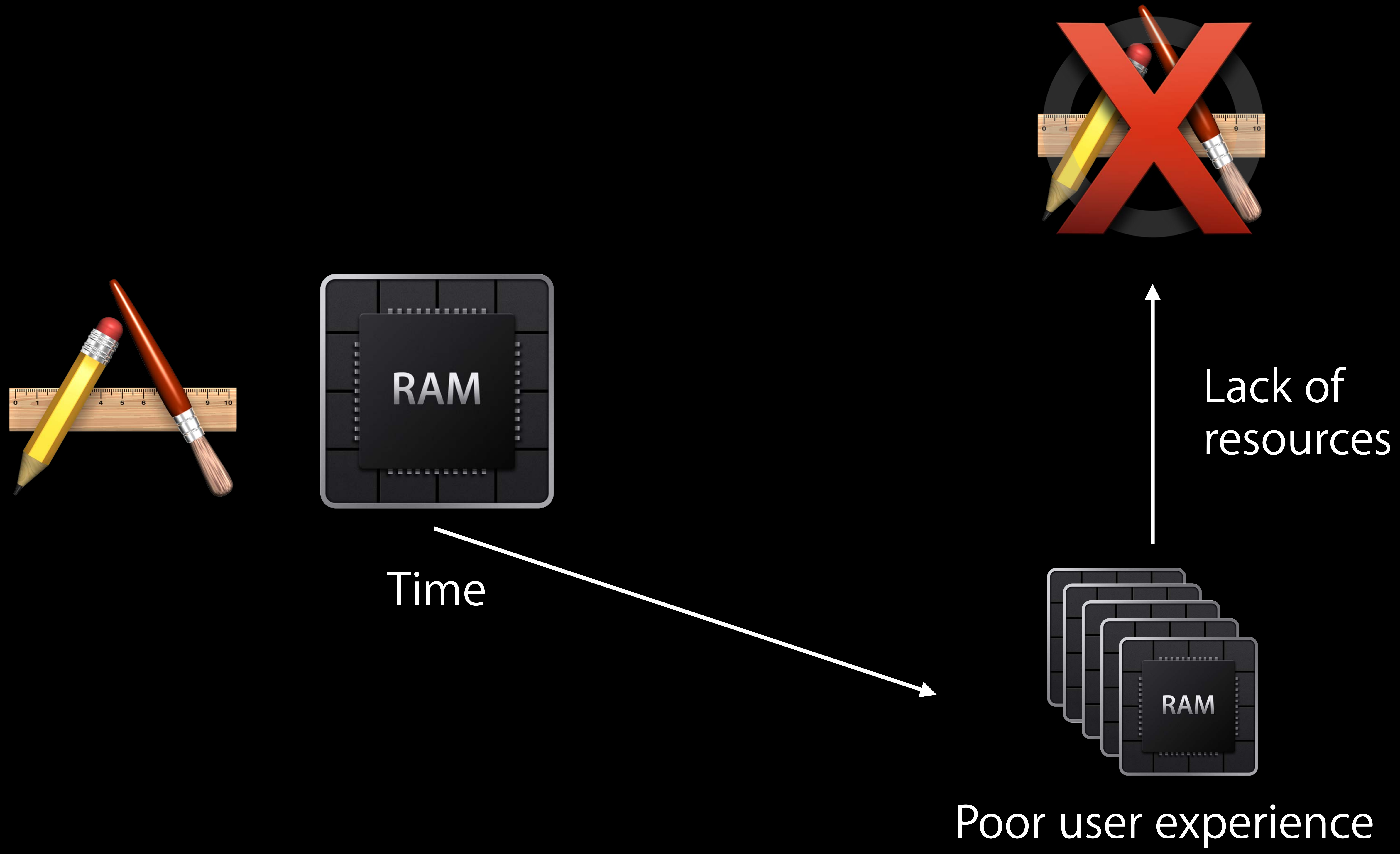



RAM



Time

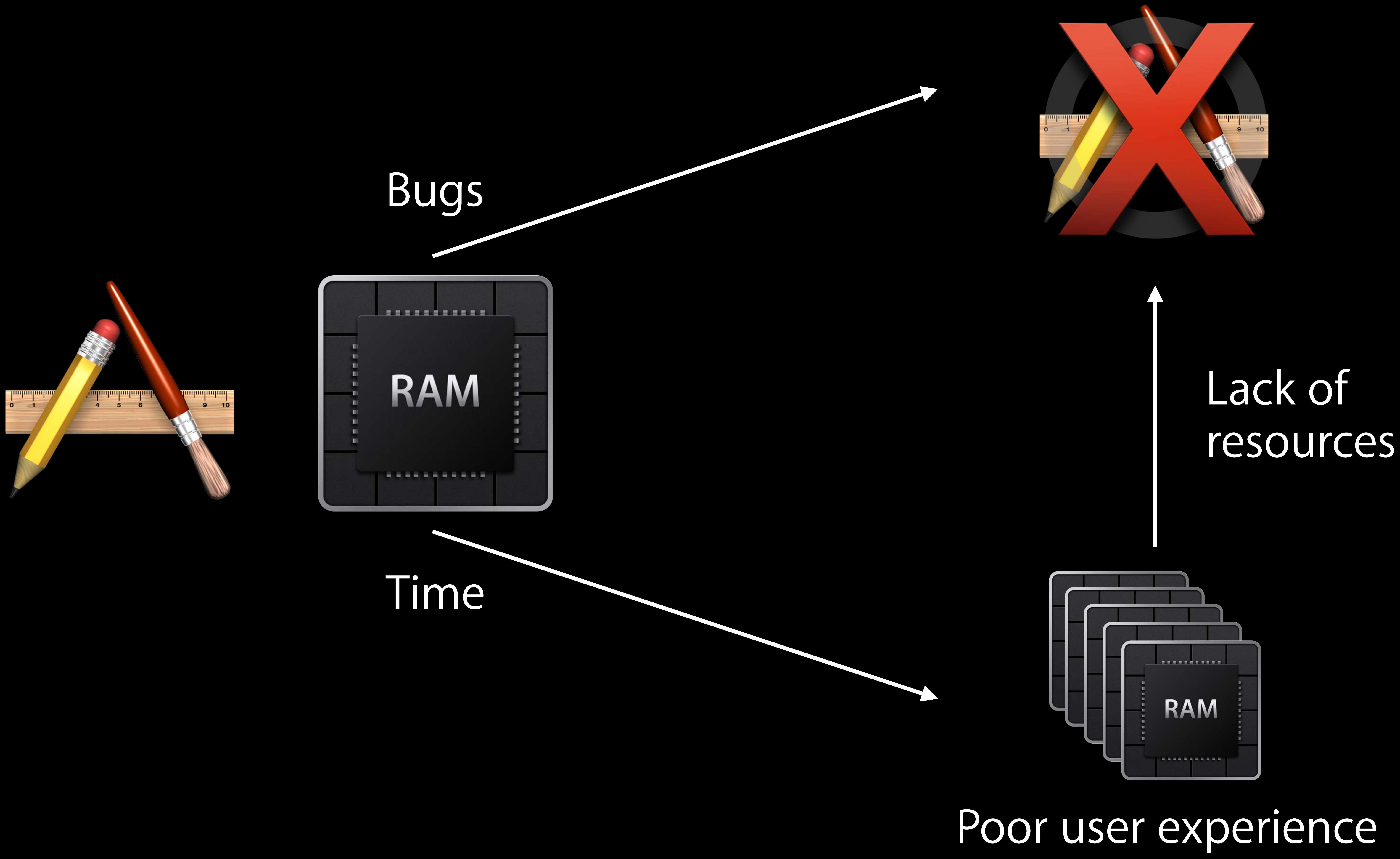
Poor user experience



Time

Lack of resources

Poor user experience



Bugs

Time

Lack of resources

Poor user experience

Agenda

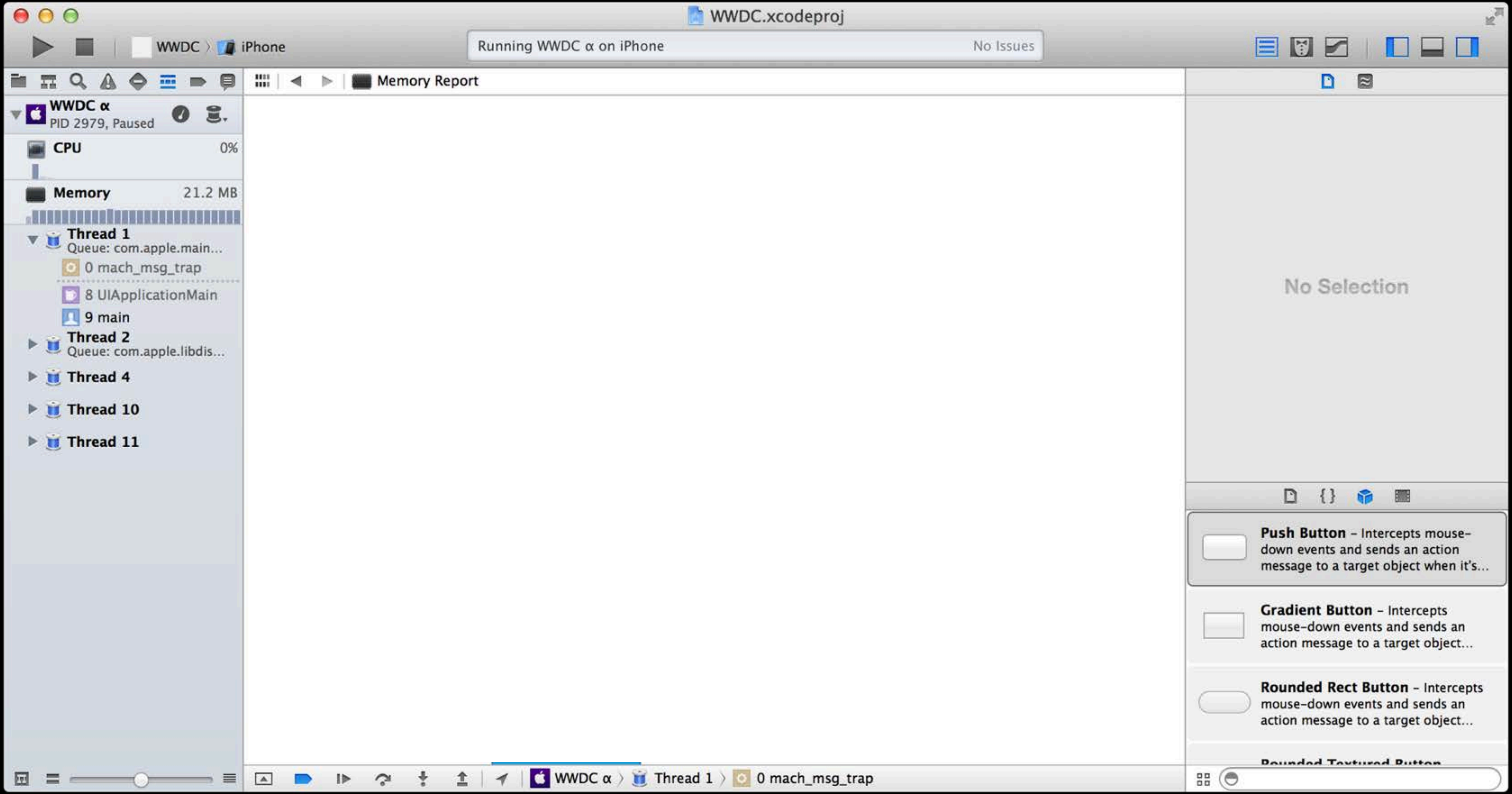
- Overview of app memory
- Heap memory issues
- Objective-C, retain/release
- Being a good citizen

Overview of App Memory

Measurement and fundamentals

Xcode Gauges

Memory at a glance



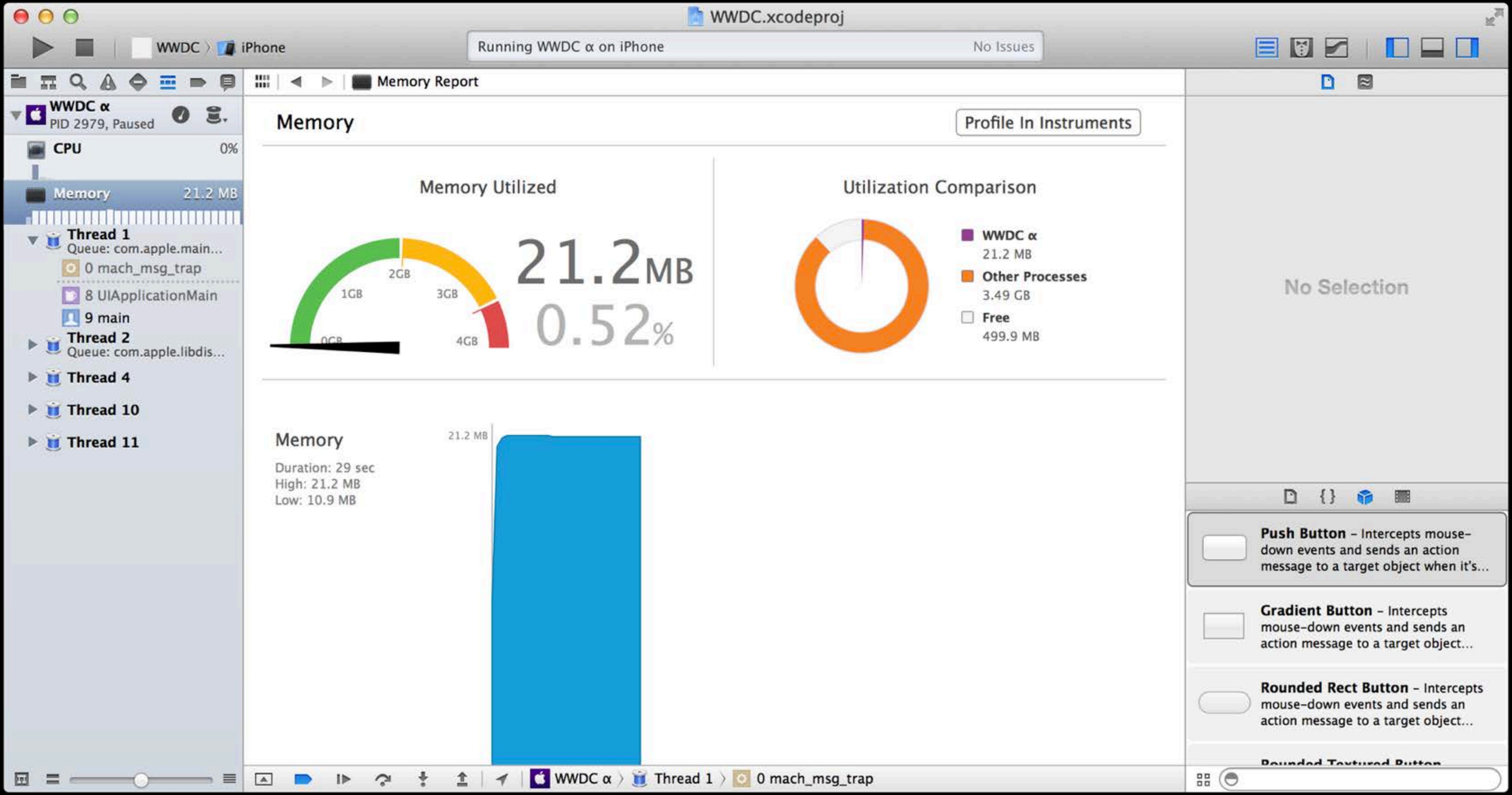
Push Button – Intercepts mouse-down events and sends an action message to a target object when it's...

Gradient Button – Intercepts mouse-down events and sends an action message to a target object...

Rounded Rect Button – Intercepts mouse-down events and sends an action message to a target object...

Xcode Gauges

Memory at a glance



Memory

Duration: 29 sec
High: 21.2 MB
Low: 10.9 MB

Push Button – Intercepts mouse-down events and sends an action message to a target object when it's...

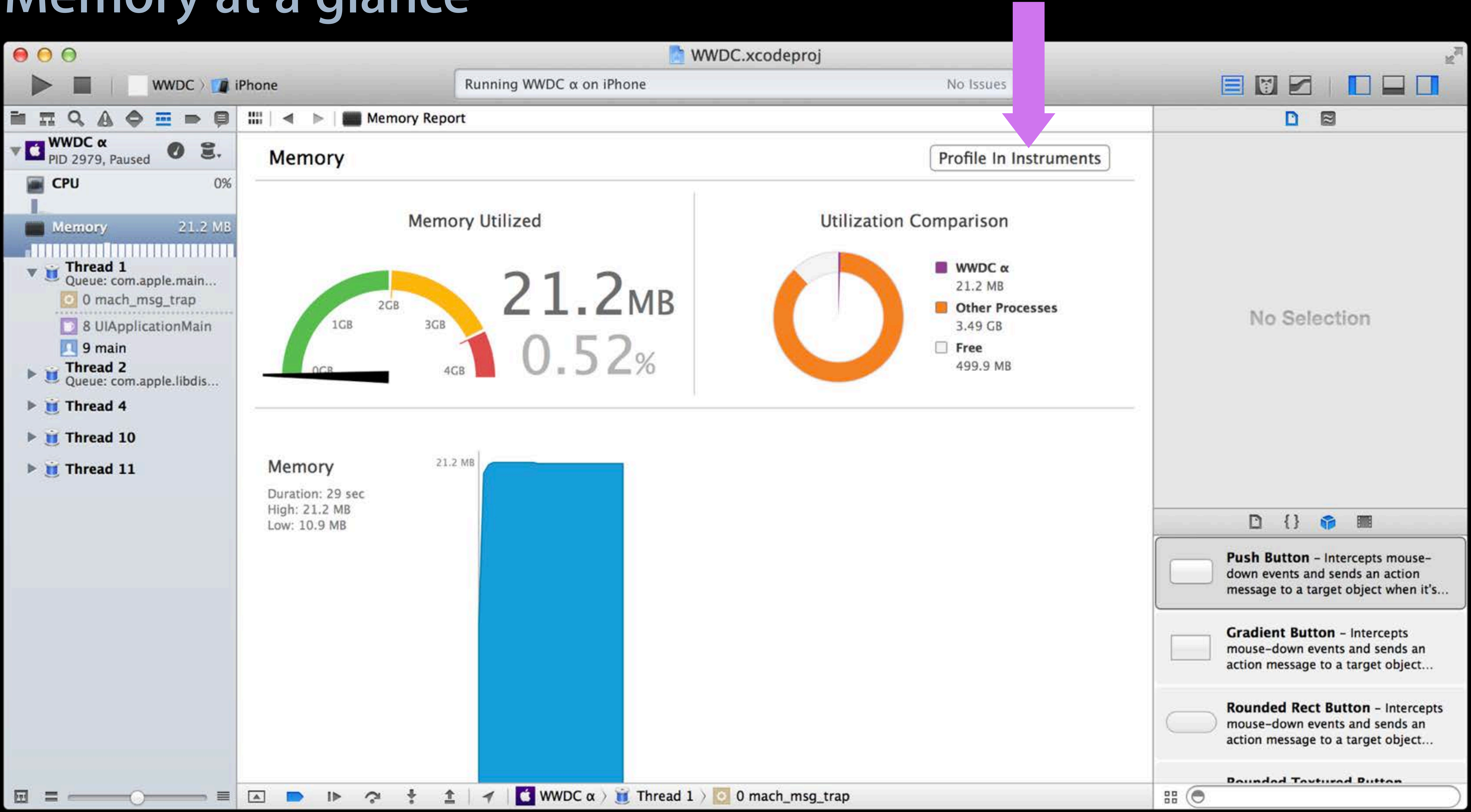
Gradient Button – Intercepts mouse-down events and sends an action message to a target object...

Rounded Rect Button – Intercepts mouse-down events and sends an action message to a target object...

Rounded Textured Button

Xcode Gauges

Memory at a glance



Memory: the Big Picture

- Instruments focused on allocation heap

Heap
Your Process

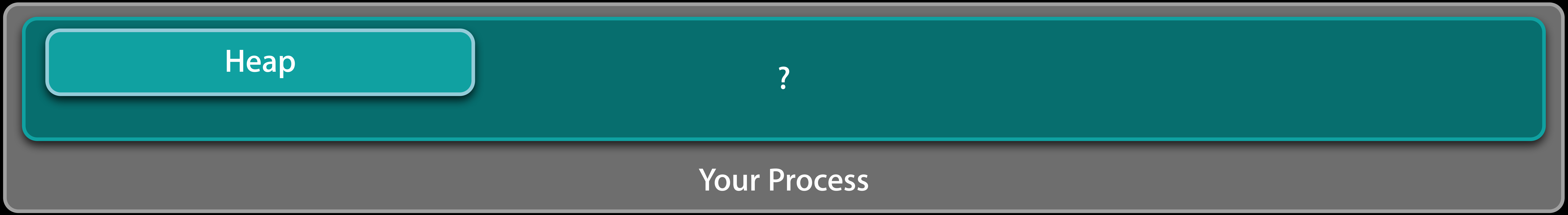
Memory: the Big Picture

- Instruments focused on allocation heap
- Processes contain more than just heap memory
  - Application code
  - Images and other media

Heap
Your Process

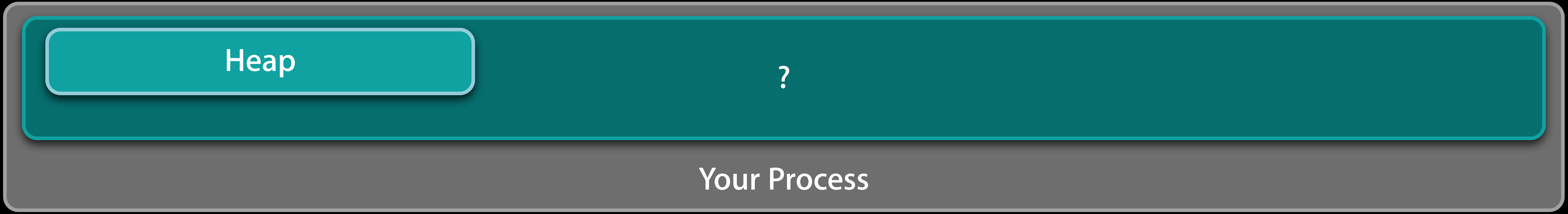
Memory: the Big Picture

- Instruments focused on allocation heap
- Processes contain more than just heap memory
  - Application code
  - Images and other media



Memory: the Big Picture

- Instruments focused on allocation heap
- Processes contain more than just heap memory
  - Application code
  - Images and other media
- Measurements depend on what you are measuring and how



Demo

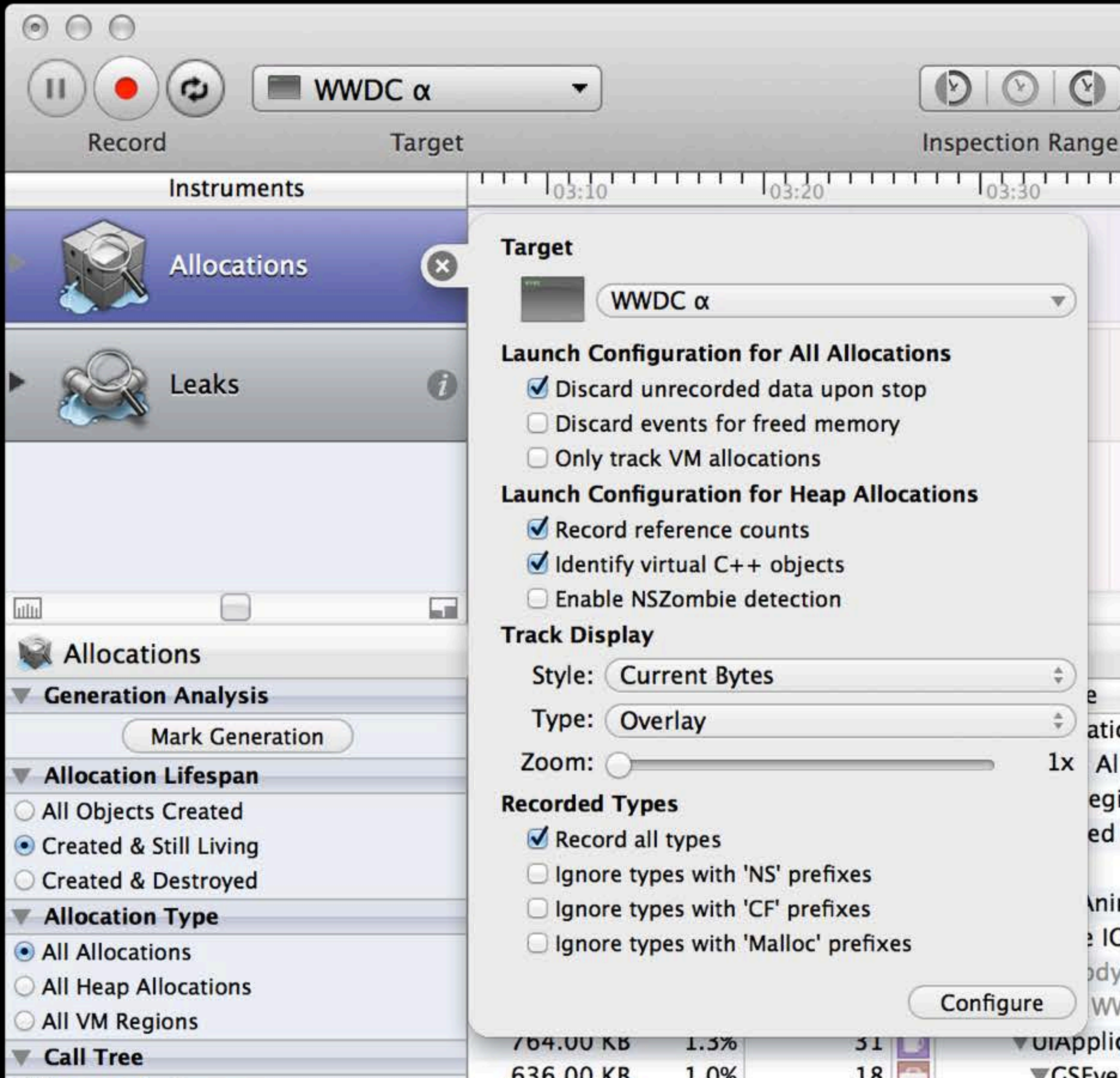
Xcode to instruments — memory tools

Allocations and Virtual Memory

- Familiar instrument… with a twist
  - Backtraces for VM region activity
  - Call trees for all allocations
  - Efficient alternative: VM *only*
- Exposes previously hidden details
  - Who mapped a file?
  - What non-heap memory contributes to my footprint?
- Page-level statistics
  - Snapshot using VM Tracker instrument

Allocations and Virtual Memory

- Familiar instrument… with a twist
  - Backtraces for VM region activity
  - Call trees for all allocations
  - Efficient alternative: VM *only*
- Exposes previously hidden details
  - Who mapped a file?
  - What non-heap memory contributes to my footprint?
- Page-level statistics
  - Snapshot using VM Tracker instrument



Allocations

Generation Analysis

Mark Generation

Allocation Lifespan

○ All Objects Created
● Created & Still Living
○ Created & Destroyed

Allocation Type

● All Allocations
○ All Heap Allocations
○ All VM Regions

Call Tree

Target

WWDC α

Launch Configuration for All Allocations

☑ Discard unrecorded data upon stop
☐ Discard events for freed memory
☐ Only track VM allocations

Launch Configuration for Heap Allocations

☑ Record reference counts
☑ Identify virtual C++ objects
☐ Enable NSZombie detection

Track Display

Style: Current Bytes
Type: Overlay
Zoom: 1x

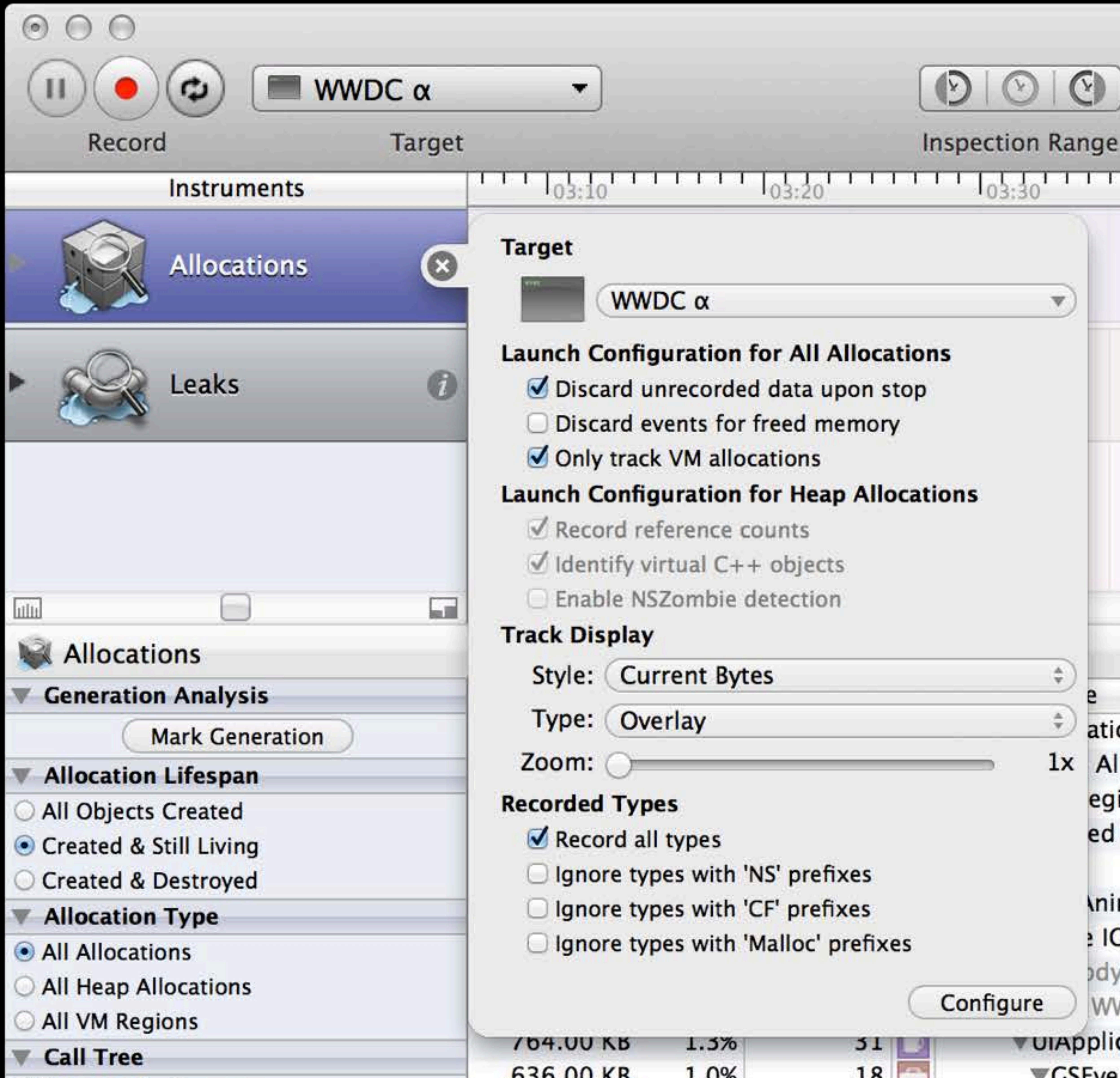
Recorded Types

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

Configure

Allocations and Virtual Memory

- Familiar instrument… with a twist
  - Backtraces for VM region activity
  - Call trees for all allocations
  - Efficient alternative: VM *only*
- Exposes previously hidden details
  - Who mapped a file?
  - What non-heap memory contributes to my footprint?
- Page-level statistics
  - Snapshot using VM Tracker instrument



Target: WWDC α

Launch Configuration for All Allocations

☑ Discard unrecorded data upon stop
☐ Discard events for freed memory
☑ Only track VM allocations

Launch Configuration for Heap Allocations

☑ Record reference counts
☑ Identify virtual C++ objects
☐ Enable NSZombie detection

Track Display

Style: Current Bytes
Type: Overlay
Zoom: 1x

Recorded Types

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

Generation Analysis

Allocation Lifespan

○ All Objects Created
● Created & Still Living
○ Created & Destroyed

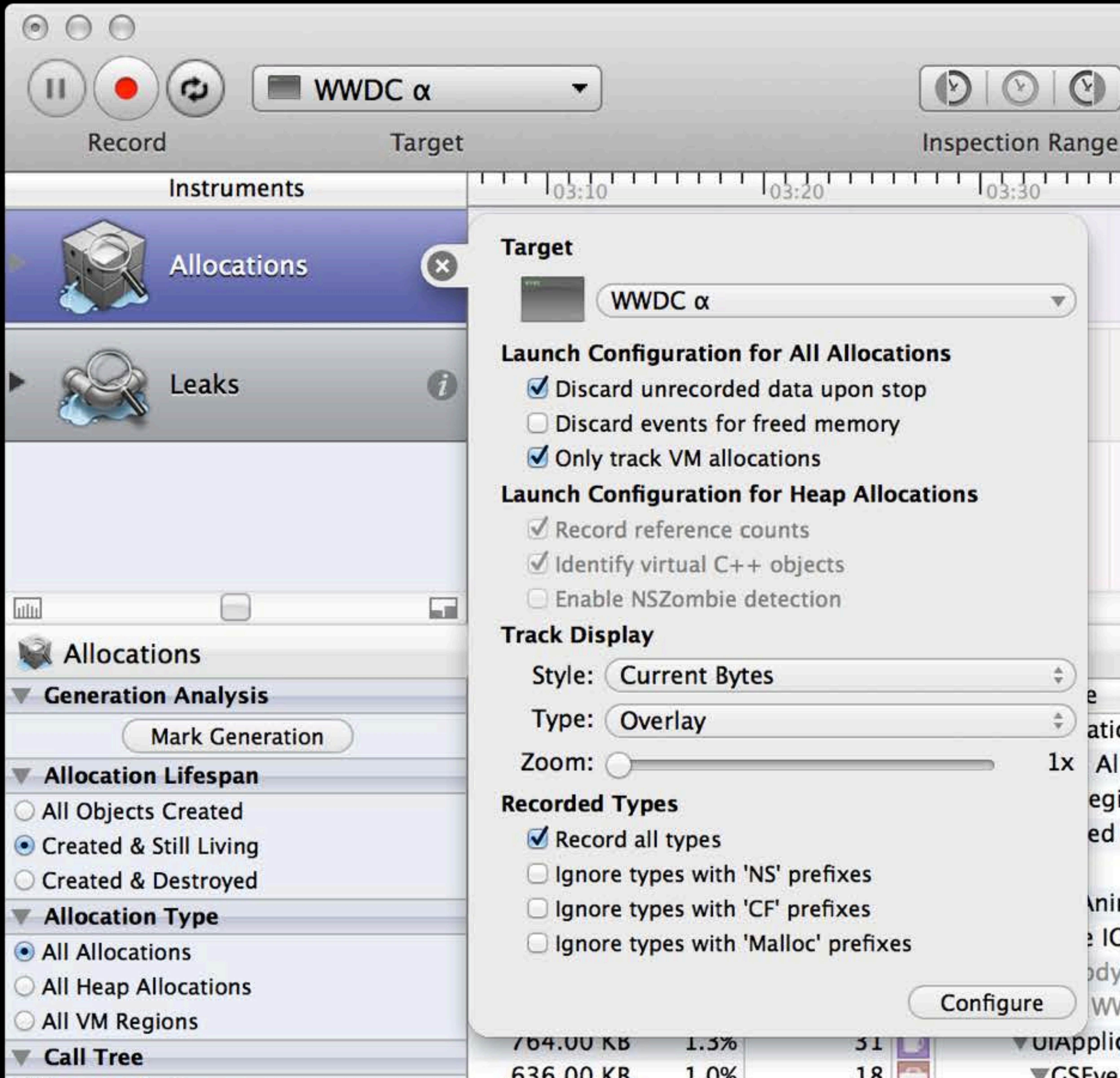
Allocation Type

● All Allocations
○ All Heap Allocations
○ All VM Regions

Call Tree

Allocations and Virtual Memory

- Familiar instrument… with a twist
  - Backtraces for VM region activity
  - Call trees for all allocations
  - Efficient alternative: VM *only*
- Exposes previously hidden details
  - Who mapped a file?
  - What non-heap memory contributes to my footprint?



Allocations

Generation Analysis

Mark Generation

Allocation Lifespan

○ All Objects Created
● Created & Still Living
○ Created & Destroyed

Allocation Type

● All Allocations
○ All Heap Allocations
○ All VM Regions

Call Tree

Target

WWDC α

Launch Configuration for All Allocations

☑ Discard unrecorded data upon stop
□ Discard events for freed memory
☑ Only track VM allocations

Launch Configuration for Heap Allocations

☑ Record reference counts
☑ Identify virtual C++ objects
□ Enable NSZombie detection

Track Display

Style: Current Bytes
Type: Overlay
Zoom: 1x

Recorded Types

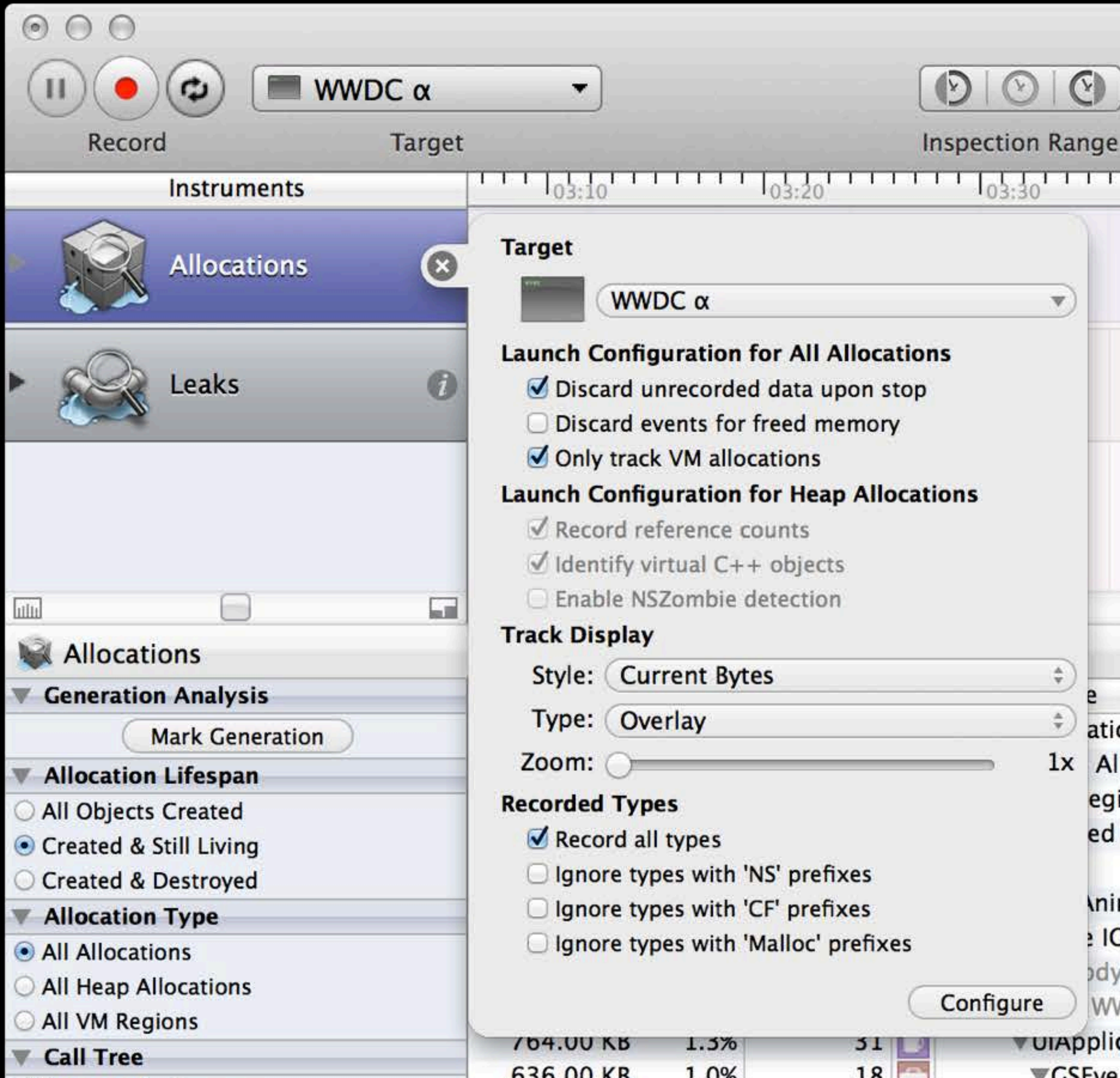
☑ Record all types
□ Ignore types with 'NS' prefixes
□ Ignore types with 'CF' prefixes
□ Ignore types with 'Malloc' prefixes

Configure

```
764.00 KB   1.3%   31
636.00 KB   1.0%   18
```

Allocations and Virtual Memory

- Familiar instrument… with a twist
  - Backtraces for VM region activity
  - Call trees for all allocations
  - Efficient alternative: VM *only*
- Exposes previously hidden details
  - Who mapped a file?
  - What non-heap memory contributes to my footprint?
- Page-level statistics
  - Snapshot using VM Tracker instrument



Allocations

Generation Analysis

Allocation Lifespan
○ All Objects Created
● Created & Still Living
○ Created & Destroyed

Allocation Type
● All Allocations
○ All Heap Allocations
○ All VM Regions

Call Tree

Target: WWDC α

Launch Configuration for All Allocations
☑ Discard unrecorded data upon stop
☐ Discard events for freed memory
☑ Only track VM allocations

Launch Configuration for Heap Allocations
☑ Record reference counts
☑ Identify virtual C++ objects
☐ Enable NSZombie detection

Track Display
Style: Current Bytes
Type: Overlay
Zoom: 1x

Recorded Types
☑ Record all types
☐ Ignore types with 'NS' prefixes
☐ Ignore types with 'CF' prefixes
☐ Ignore types with 'Malloc' prefixes

764.00 KB 1.3% 31
636.00 KB 1.0% 18

Virtual Memory

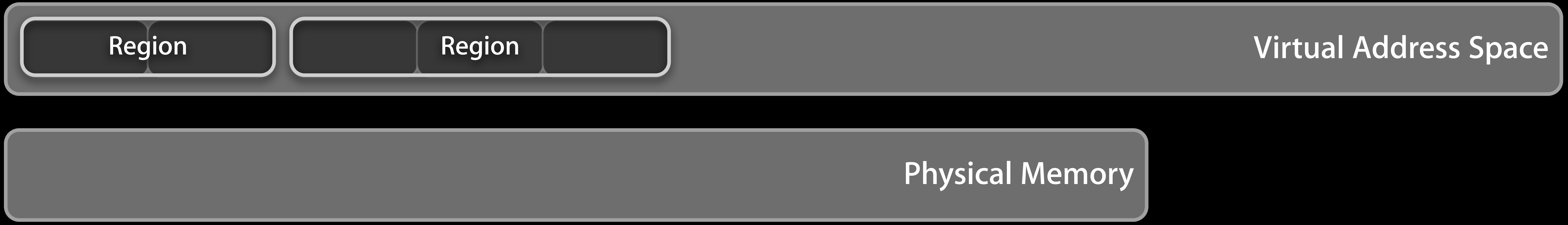
Key concepts

- Virtual vs. Resident
- Clean vs. Dirty
- Private vs. Shared

Virtual Memory

Key concepts

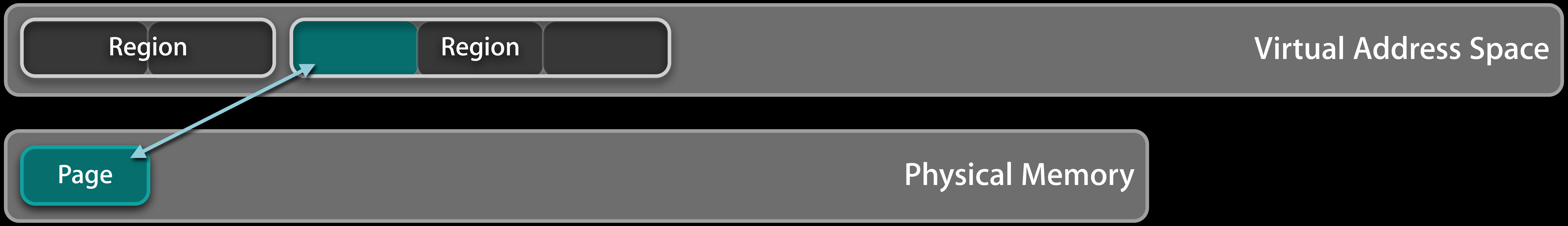
- Virtual vs. Resident
  - Virtual memory reserved as regions
    - 4KB page aligned
  - Pages mapped to physical memory on first read/write
    - Zero-filled or read from storage
    - Once mapped, virtual memory is also resident
  - Physical memory typically more constrained



Virtual Memory

Key concepts

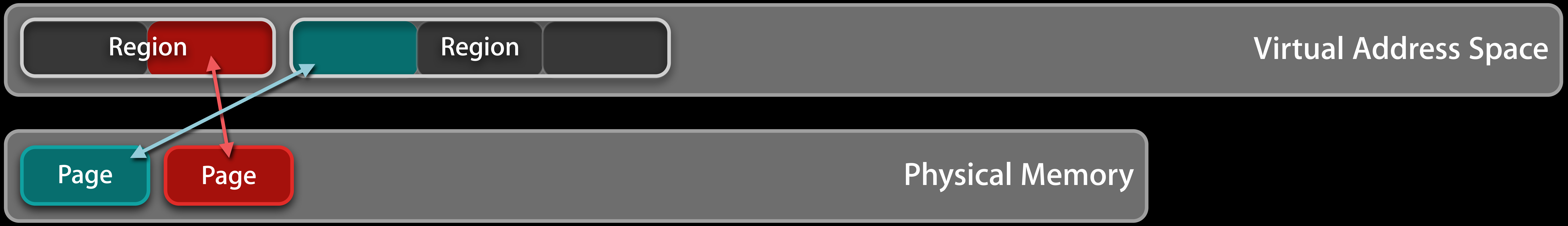
- Virtual vs. Resident
  - Virtual memory reserved as regions
    - 4KB page aligned
  - Pages mapped to physical memory on first read/write
    - Zero-filled or read from storage
    - Once mapped, virtual memory is also resident
  - Physical memory typically more constrained



Virtual Memory

Key concepts

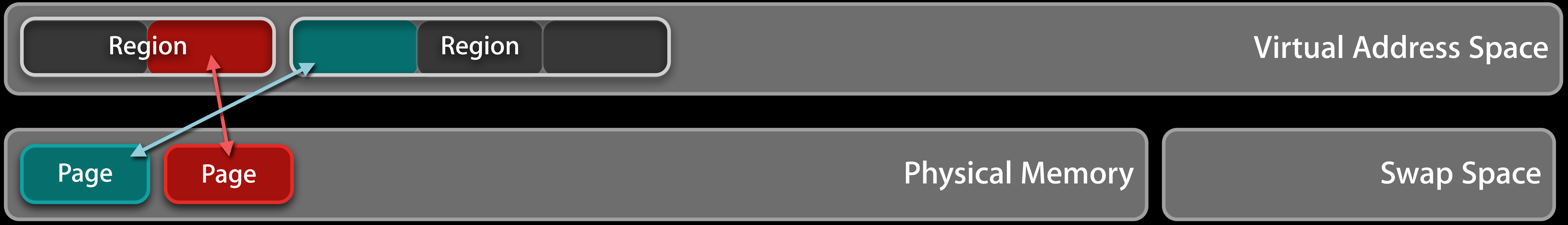
- Virtual vs. Resident
  - Virtual memory reserved as regions
    - 4KB page aligned
  - Pages mapped to physical memory on first read/write
    - Zero-filled or read from storage
    - Once mapped, virtual memory is also resident
  - Physical memory typically more constrained



Virtual Memory

Key concepts

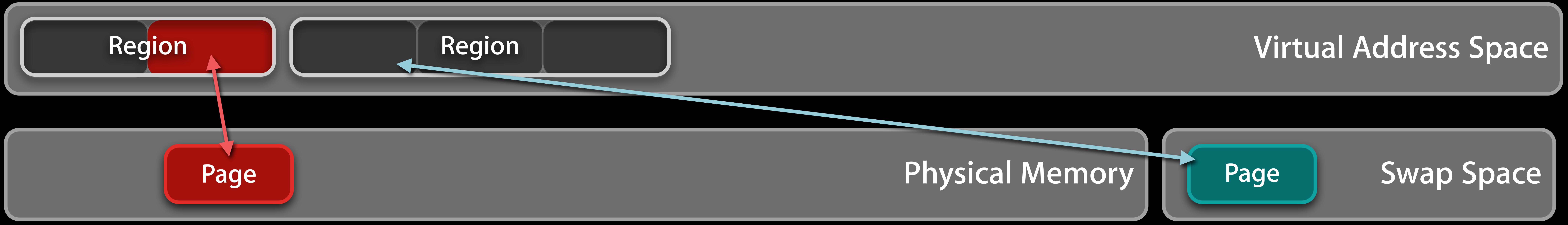
- Virtual vs. Resident
- Clean vs. Dirty
  - Clean pages can be discarded and recreated
    - Memory mapped files, executable __TEXT segments, purgeable memory
  - Changing a page marks it as dirty
    - Malloc heap, global variables, stacks, etc.
    - Can be swapped to compressed form or storage on OS X



Virtual Memory

Key concepts

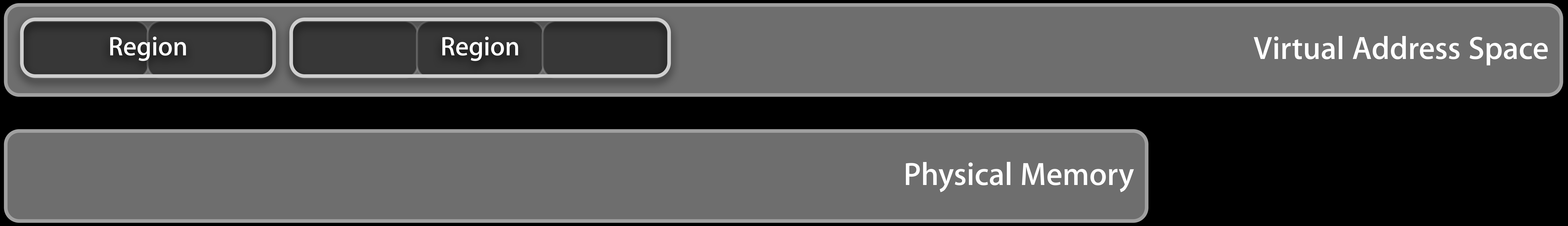
- Virtual vs. Resident
- Clean vs. Dirty
  - Clean pages can be discarded and recreated
    - Memory mapped files, executable __TEXT segments, purgeable memory
  - Changing a page marks it as dirty
    - Malloc heap, global variables, stacks, etc.
    - Can be swapped to compressed form or storage on OS X



Virtual Memory

Key concepts

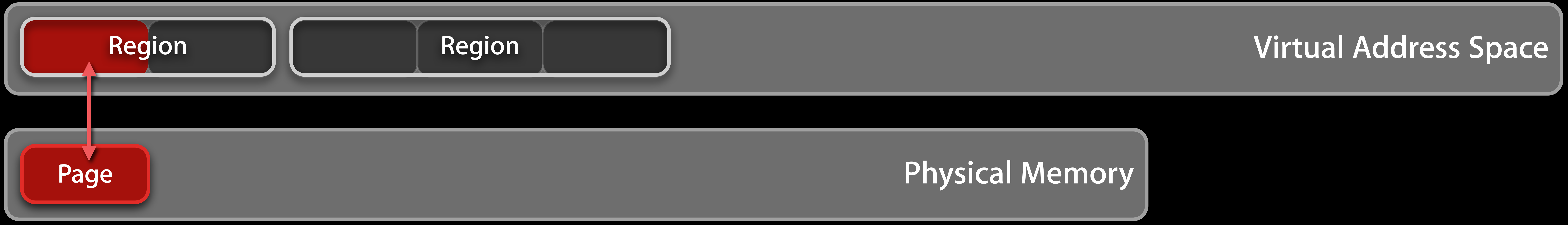
- Virtual vs. Resident
- Clean vs. Dirty
- Private vs. Shared
  - Virtual memory can be named to enable sharing
  - Mapped files are implicitly shareable



Virtual Memory

Key concepts

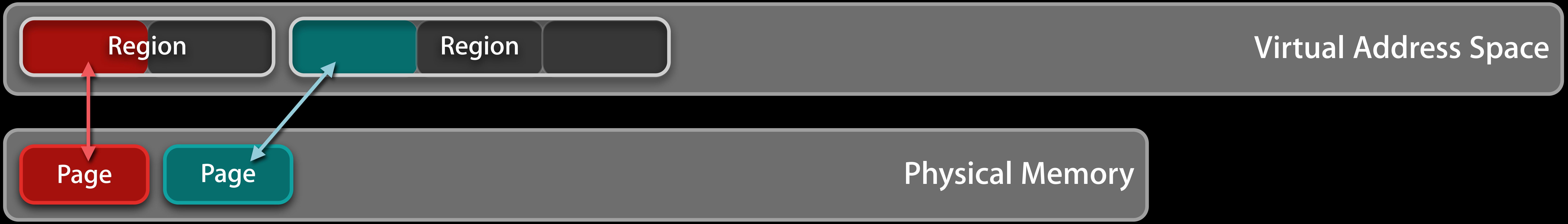
- Virtual vs. Resident
- Clean vs. Dirty
- Private vs. Shared
  - Virtual memory can be named to enable sharing
  - Mapped files are implicitly shareable



Virtual Memory

Key concepts

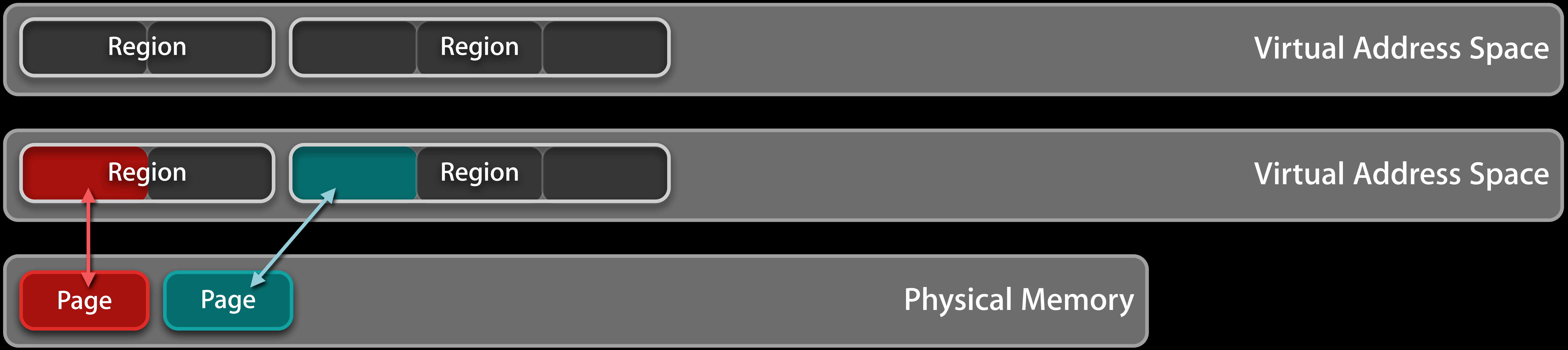
- Virtual vs. Resident
- Clean vs. Dirty
- Private vs. Shared
  - Virtual memory can be named to enable sharing
  - Mapped files are implicitly shareable



Virtual Memory

Key concepts

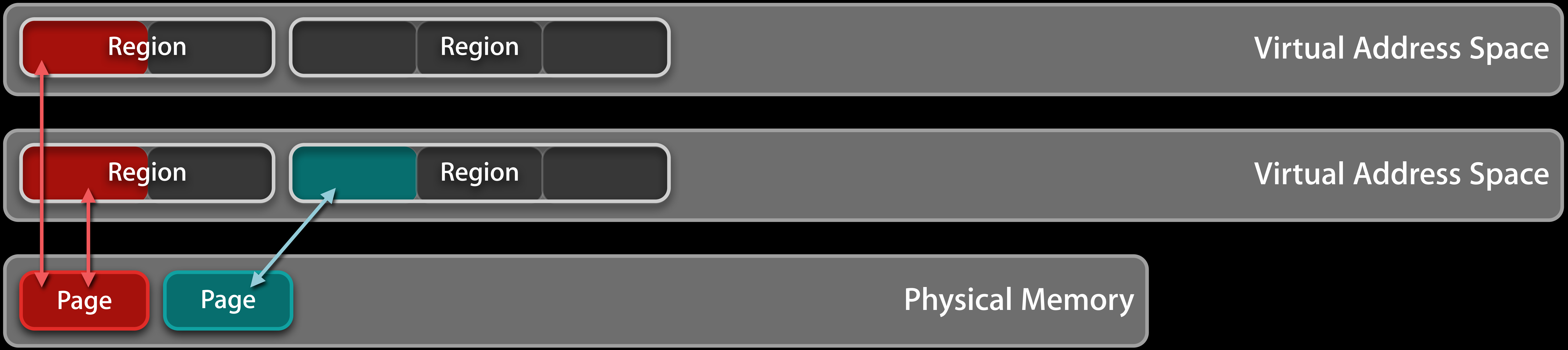
- Virtual vs. Resident
- Clean vs. Dirty
- Private vs. Shared
  - Virtual memory can be named to enable sharing
  - Mapped files are implicitly shareable



Virtual Memory

Key concepts

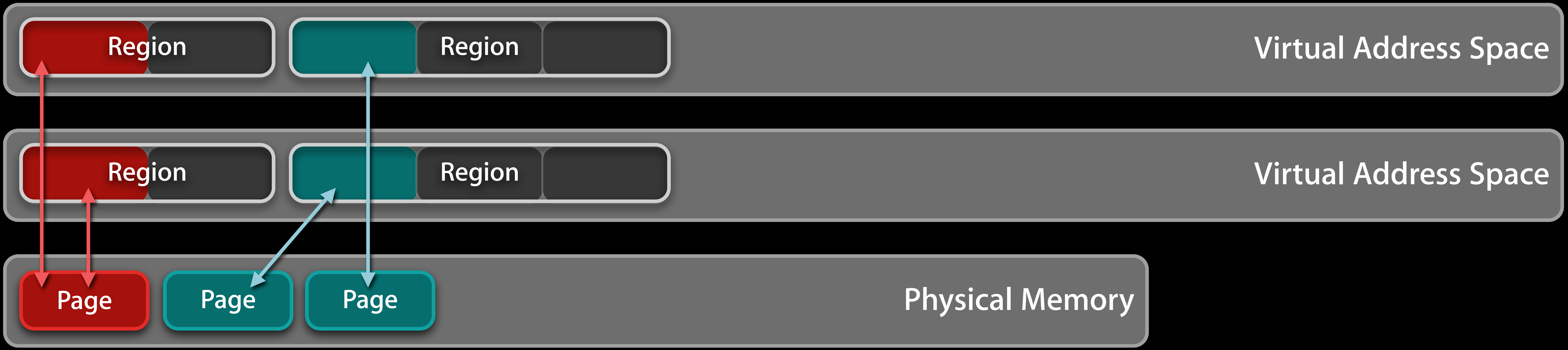
- Virtual vs. Resident
- Clean vs. Dirty
- Private vs. Shared
  - Virtual memory can be named to enable sharing
  - Mapped files are implicitly shareable



Virtual Memory

Key concepts

- Virtual vs. Resident
- Clean vs. Dirty
- Private vs. Shared
  - Virtual memory can be named to enable sharing
  - Mapped files are implicitly shareable



Heap Memory Issues

Tools and tactics

What is the Heap?

Storage for malloc() calls

- Dynamic allocations using malloc or variants
  - **malloc** in C
  - **[NSObject alloc]** in Objective-C
  - **new** operators in C++
- Allocated directly or indirectly through framework API
- Backed by VM: MALLOC regions

Investigating the Heap

Expensive Types

- VM is about bytes, heap is about counts

Investigating the Heap

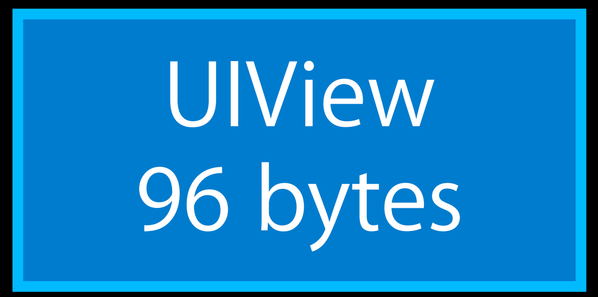
Expensive Types

- VM is about bytes, heap is about counts
- Small object can have a large graph

Investigating the Heap

Expensive Types

- VM is about bytes, heap is about counts
- Small object can have a large graph

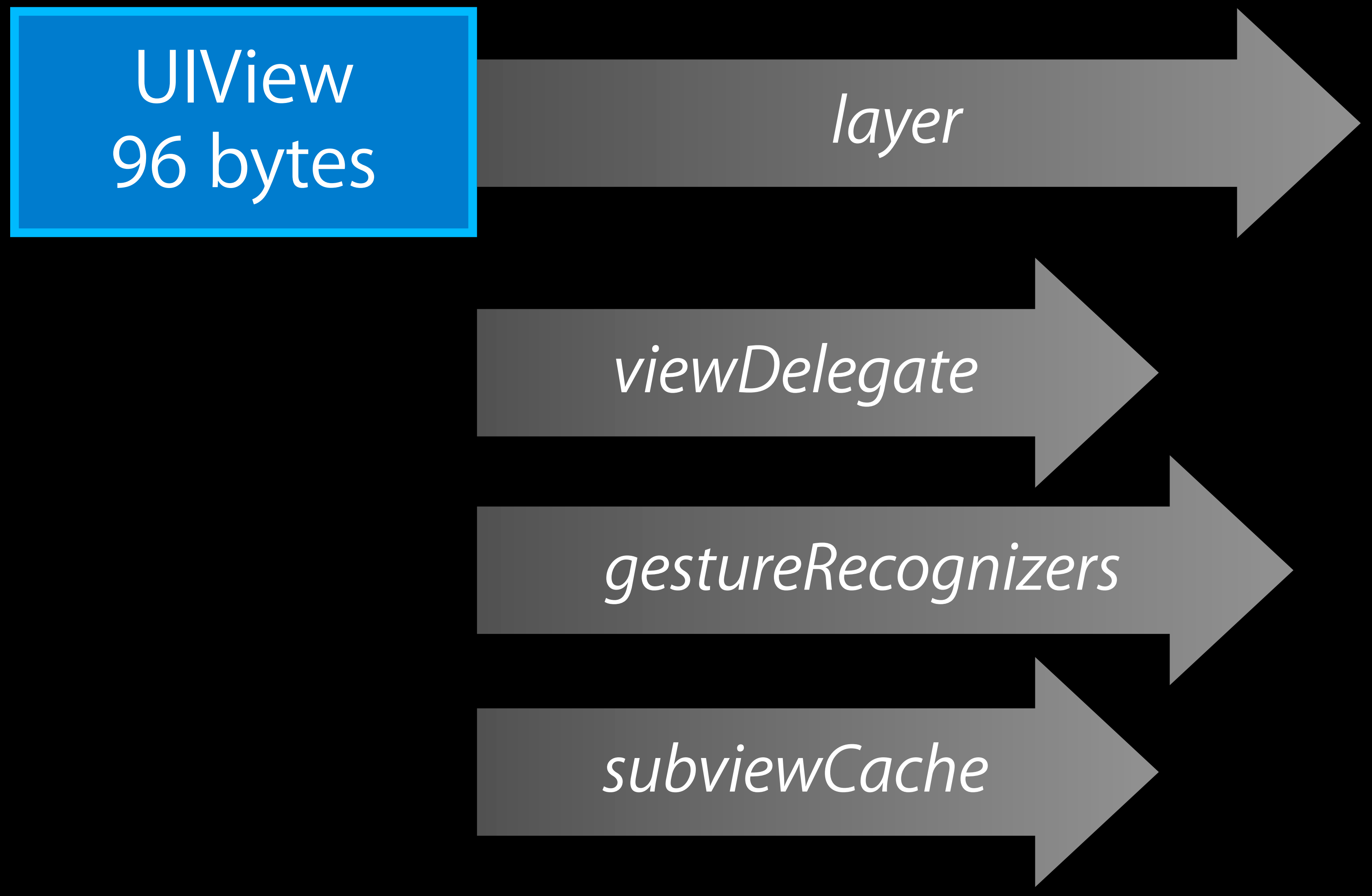


UIView
96 bytes

Investigating the Heap

Expensive Types

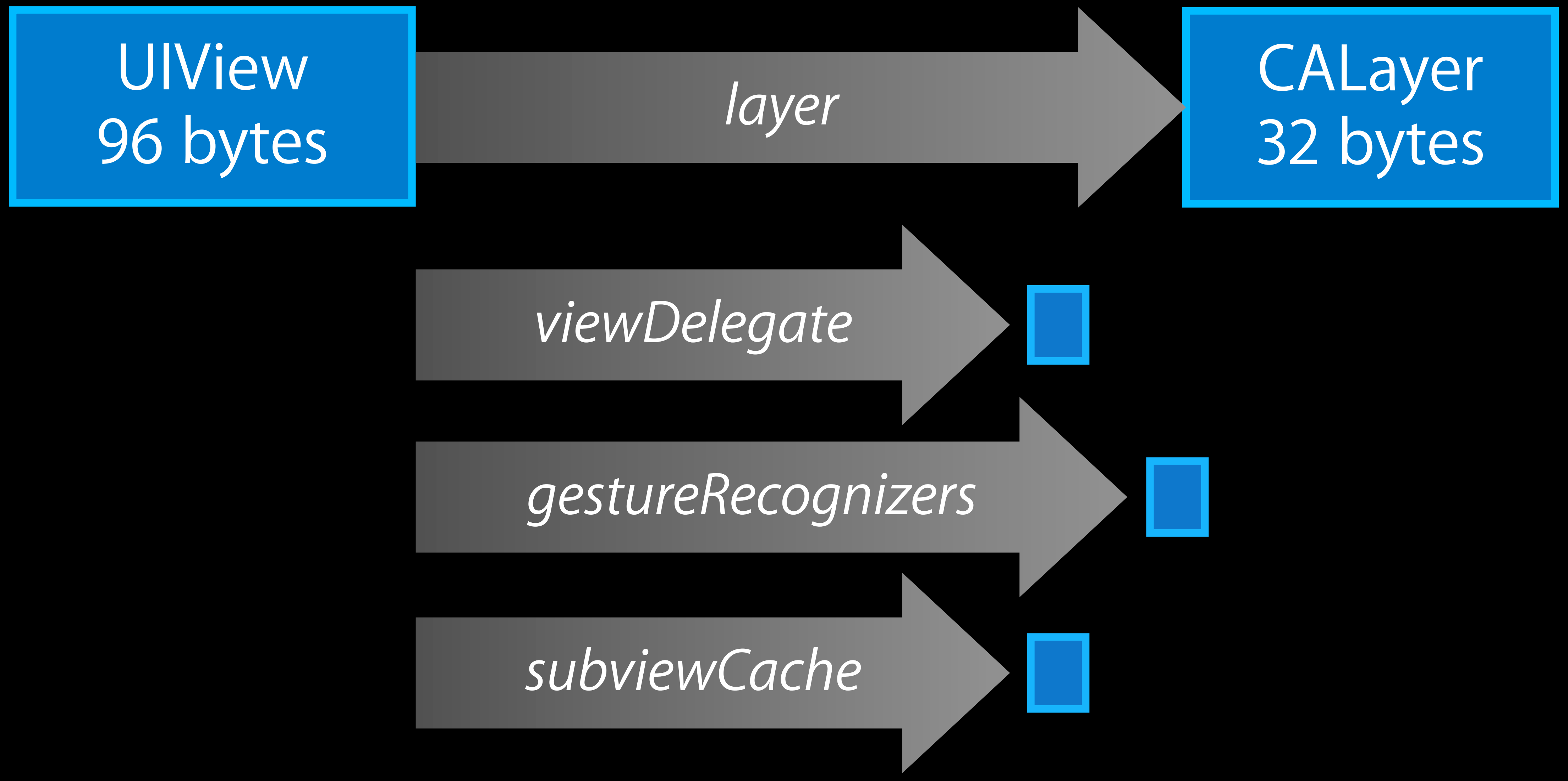
- VM is about bytes, heap is about counts
- Small object can have a large graph



Investigating the Heap

Expensive Types

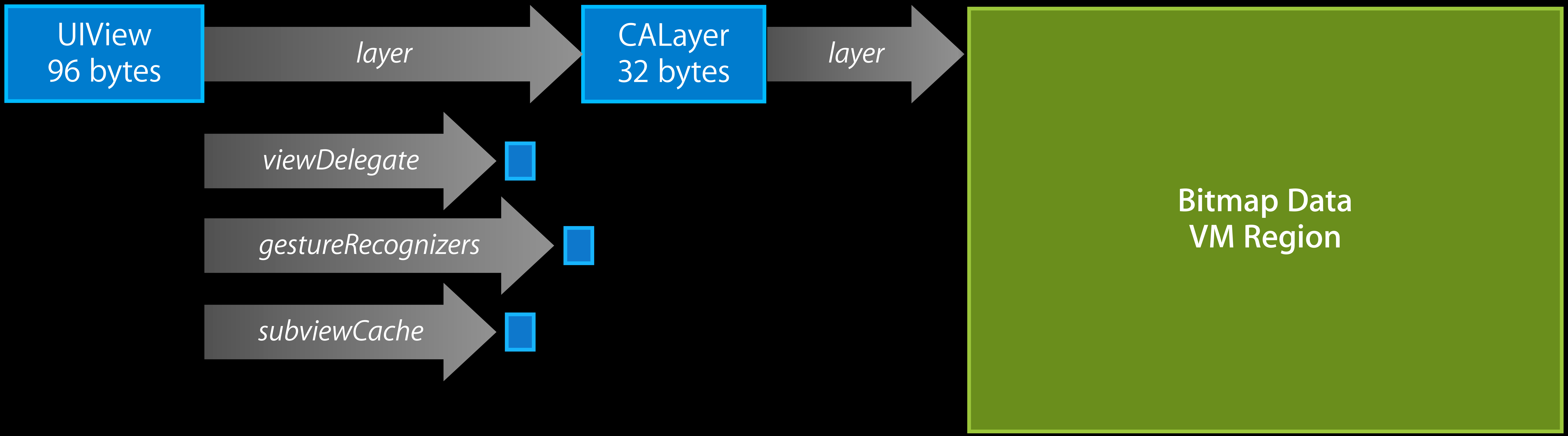
- VM is about bytes, heap is about counts
- Small object can have a large graph



Investigating the Heap

Expensive Types

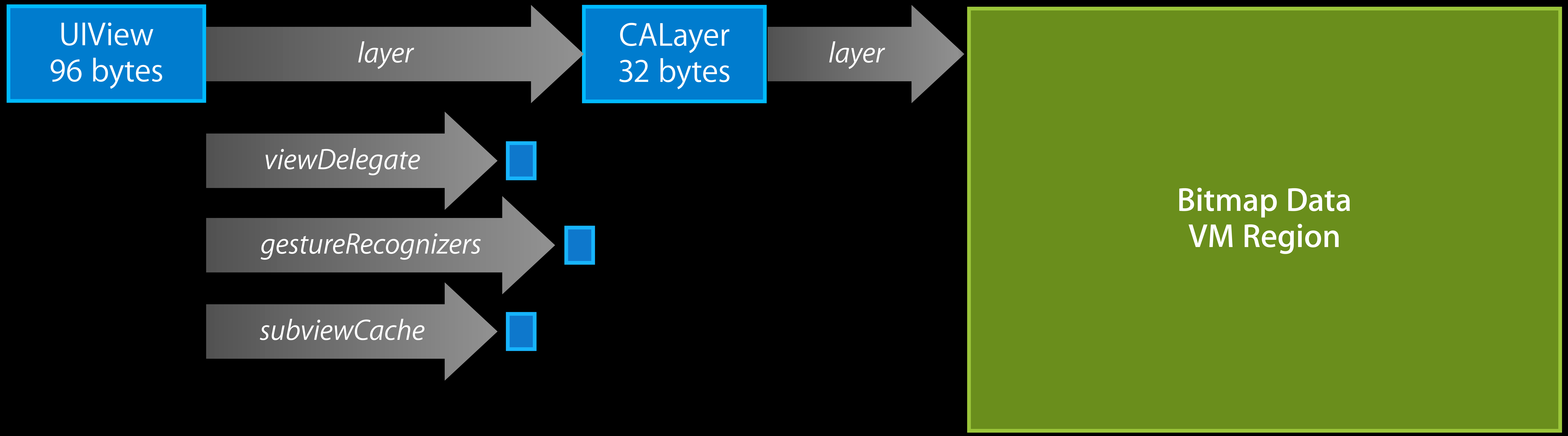
- VM is about bytes, heap is about counts
- Small object can have a large graph



Investigating the Heap

Expensive Types

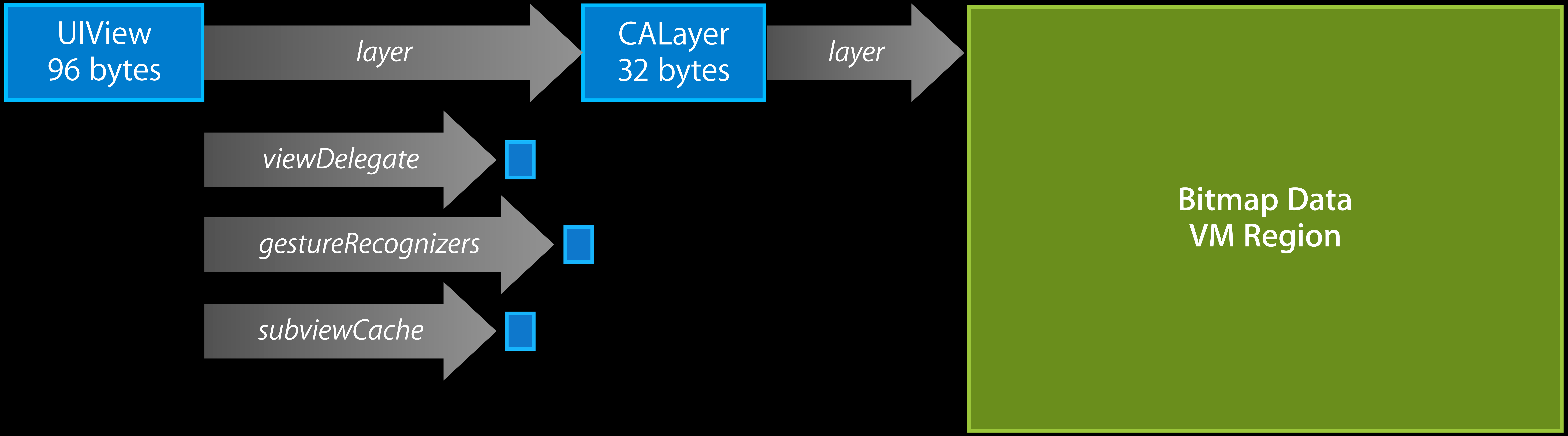
- VM is about bytes, heap is about counts
- Small object can have a large graph
- Obvious containers: NSSet, NSDictionary, NSArray, …



Investigating the Heap

Expensive Types

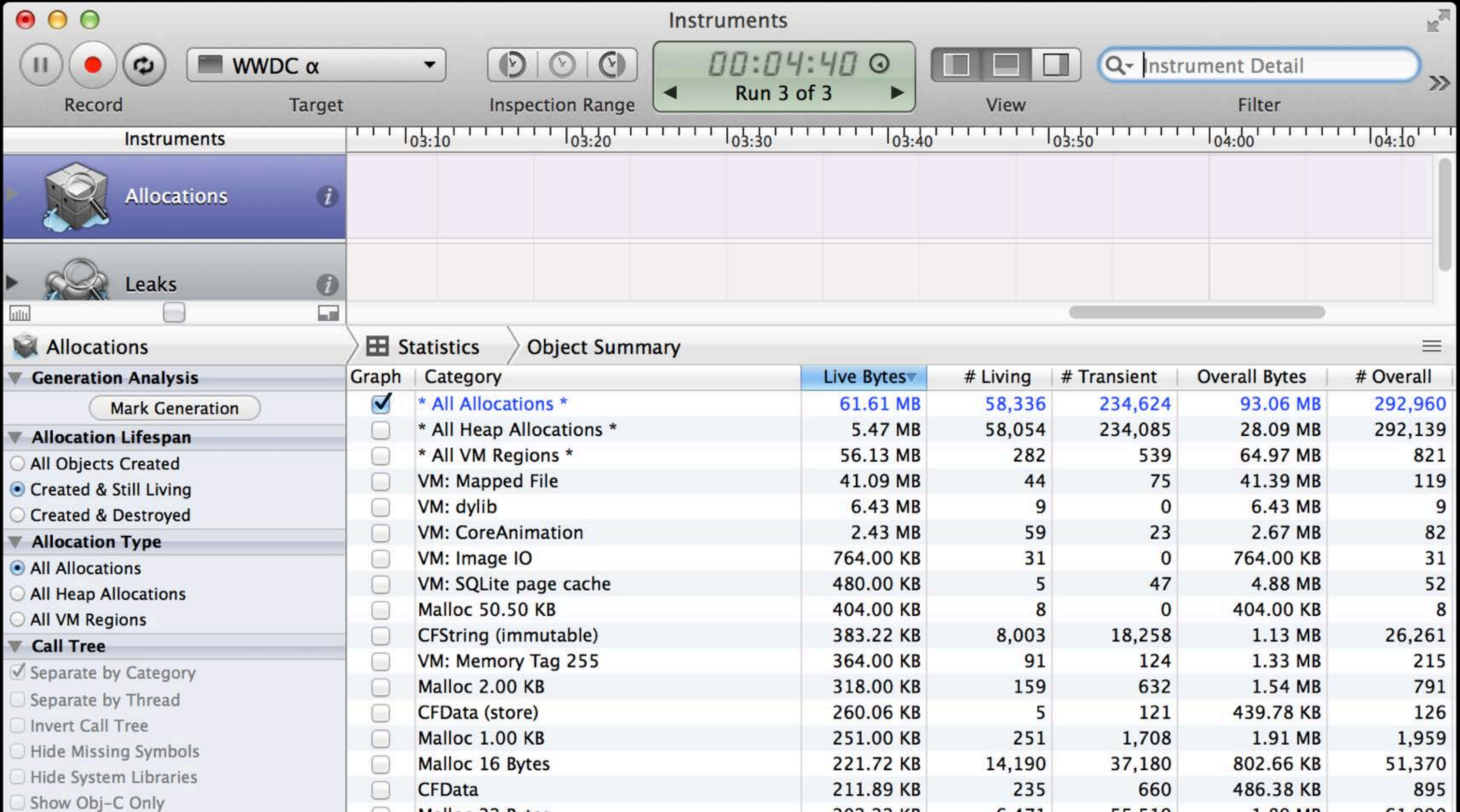
- VM is about bytes, heap is about counts
- Small object can have a large graph
- Obvious containers: NSSet, NSDictionary, NSArray, …
- Less obvious: UIView, UIViewController, NSImage, …



Investigating the Heap

- Your classes!
  - Prefixes are helpful (e.g. ABCViewController)
- New type identifications
  - Better at C++ classes
  - dispatch and xpc types
  - Heap-copied ^blocks (__NSMallocBlock__)

Investigating the Heap



Generation Analysis

Mark Generation

Allocation Lifespan

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

Allocation Type

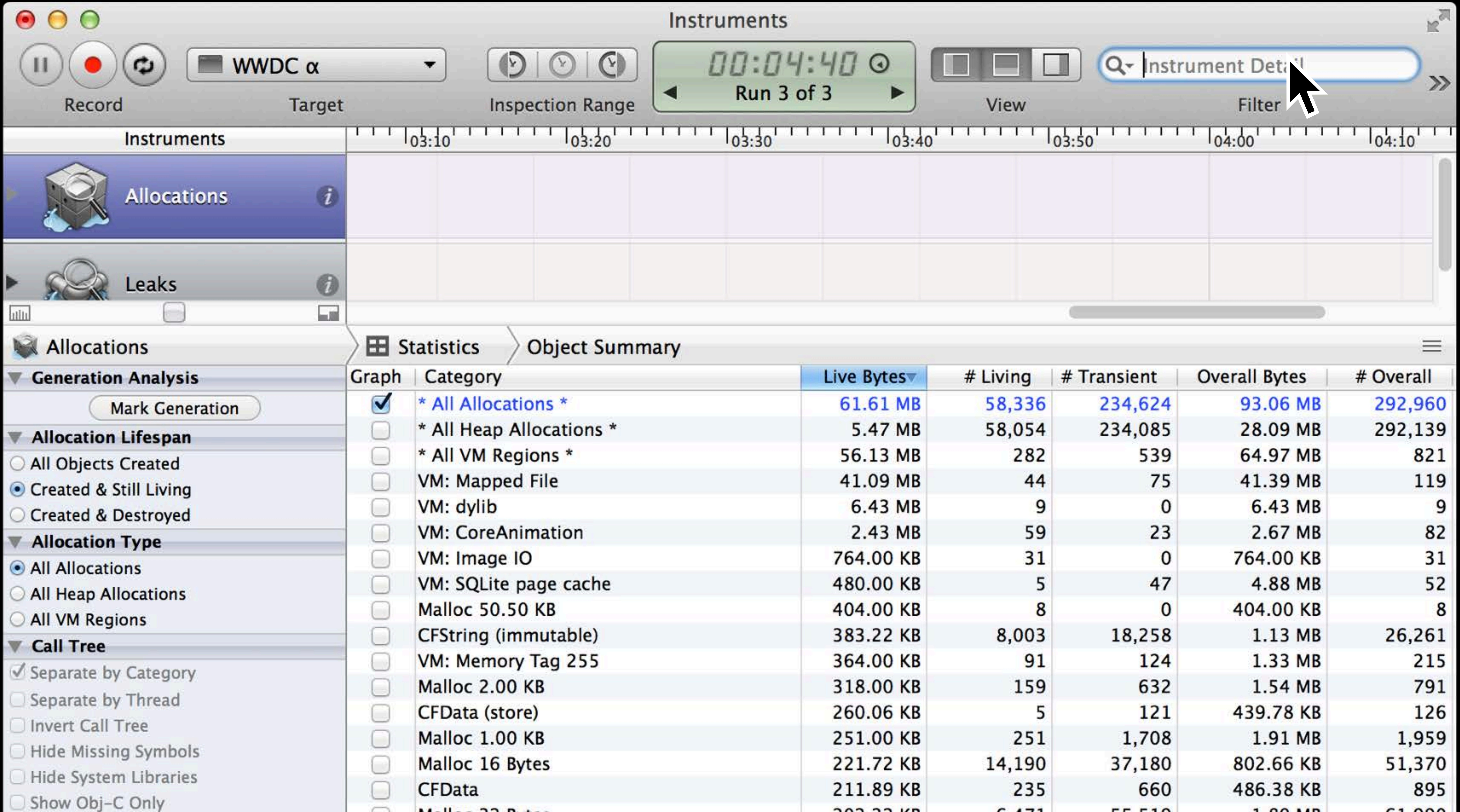
- ● All Allocations
- ○ All Heap Allocations
- ○ All VM Regions

Call Tree

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

Graph	Category	Live Bytes	# Living	# Transient	Overall Bytes	# Overall
☑	* All Allocations *	61.61 MB	58,336	234,624	93.06 MB	292,960
□	* All Heap Allocations *	5.47 MB	58,054	234,085	28.09 MB	292,139
□	* All VM Regions *	56.13 MB	282	539	64.97 MB	821
□	VM: Mapped File	41.09 MB	44	75	41.39 MB	119
□	VM: dylib	6.43 MB	9	0	6.43 MB	9
□	VM: CoreAnimation	2.43 MB	59	23	2.67 MB	82
□	VM: Image IO	764.00 KB	31	0	764.00 KB	31
□	VM: SQLite page cache	480.00 KB	5	47	4.88 MB	52
□	Malloc 50.50 KB	404.00 KB	8	0	404.00 KB	8
□	CFString (immutable)	383.22 KB	8,003	18,258	1.13 MB	26,261
□	VM: Memory Tag 255	364.00 KB	91	124	1.33 MB	215
□	Malloc 2.00 KB	318.00 KB	159	632	1.54 MB	791
□	CFData (store)	260.06 KB	5	121	439.78 KB	126
□	Malloc 1.00 KB	251.00 KB	251	1,708	1.91 MB	1,959
□	Malloc 16 Bytes	221.72 KB	14,190	37,180	802.66 KB	51,370
□	CFData	211.89 KB	235	660	486.38 KB	895
□	Malloc 32 Bytes	202.22 KB	6,471	55,510	1.80 MB	61,981

Investigating the Heap



Generation Analysis

Mark Generation

Allocation Lifespan

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

Allocation Type

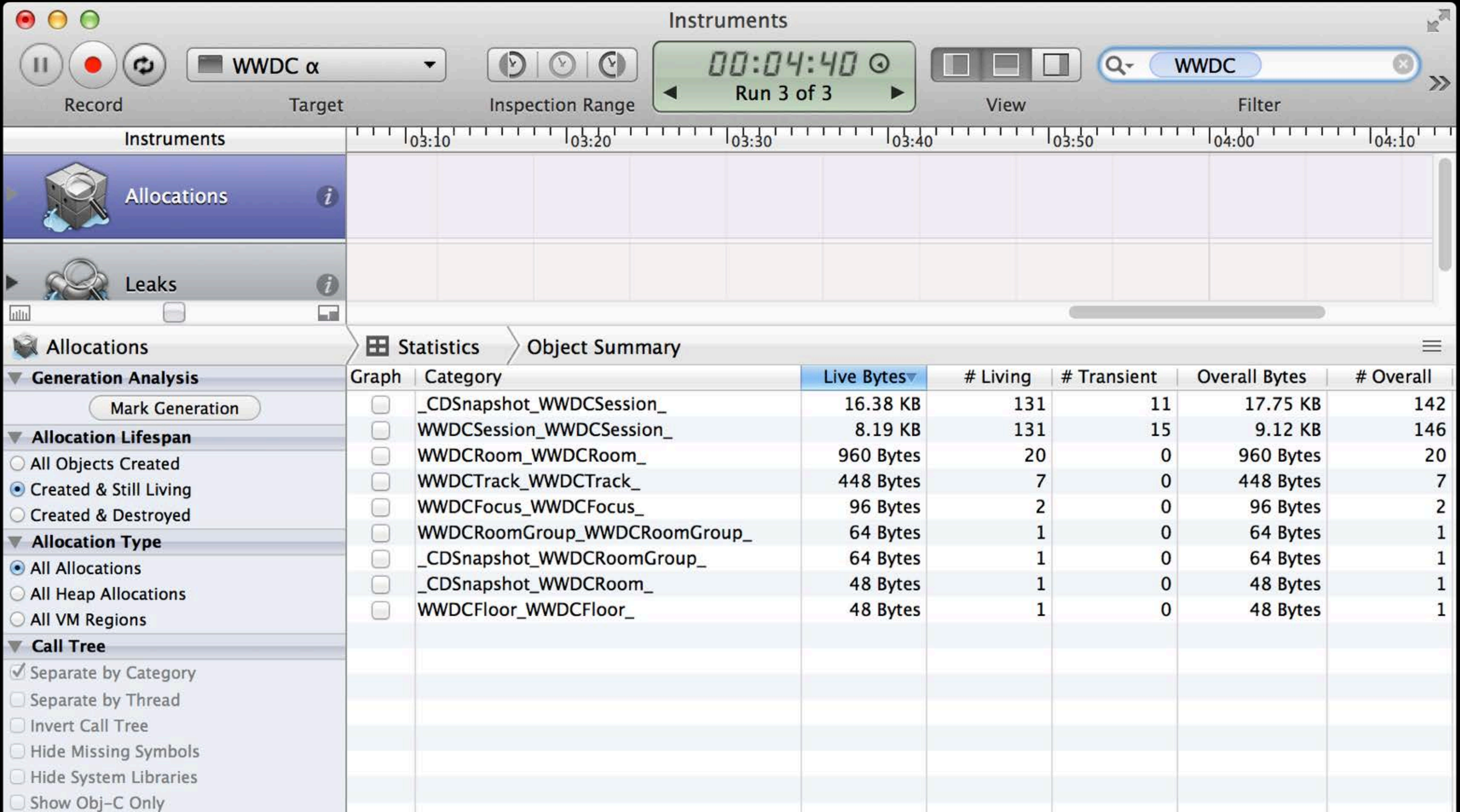
- ● All Allocations
- ○ All Heap Allocations
- ○ All VM Regions

Call Tree

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

Graph	Category	Live Bytes	# Living	# Transient	Overall Bytes	# Overall
☑	* All Allocations *	61.61 MB	58,336	234,624	93.06 MB	292,960
□	* All Heap Allocations *	5.47 MB	58,054	234,085	28.09 MB	292,139
□	* All VM Regions *	56.13 MB	282	539	64.97 MB	821
□	VM: Mapped File	41.09 MB	44	75	41.39 MB	119
□	VM: dylib	6.43 MB	9	0	6.43 MB	9
□	VM: CoreAnimation	2.43 MB	59	23	2.67 MB	82
□	VM: Image IO	764.00 KB	31	0	764.00 KB	31
□	VM: SQLite page cache	480.00 KB	5	47	4.88 MB	52
□	Malloc 50.50 KB	404.00 KB	8	0	404.00 KB	8
□	CFString (immutable)	383.22 KB	8,003	18,258	1.13 MB	26,261
□	VM: Memory Tag 255	364.00 KB	91	124	1.33 MB	215
□	Malloc 2.00 KB	318.00 KB	159	632	1.54 MB	791
□	CFData (store)	260.06 KB	5	121	439.78 KB	126
□	Malloc 1.00 KB	251.00 KB	251	1,708	1.91 MB	1,959
□	Malloc 16 Bytes	221.72 KB	14,190	37,180	802.66 KB	51,370
□	CFData	211.89 KB	235	660	486.38 KB	895

Investigating the Heap



Generation Analysis

Mark Generation

Allocation Lifespan

○ All Objects Created
● Created & Still Living
○ Created & Destroyed

Allocation Type

● All Allocations
○ All Heap Allocations
○ All VM Regions

Call Tree

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

Graph	Category	Live Bytes	# Living	# Transient	Overall Bytes	# Overall
□	_CDSnapshot_WWDCSession_	16.38 KB	131	11	17.75 KB	142
□	WWDCSession_WWDCSession_	8.19 KB	131	15	9.12 KB	146
□	WWDCRoom_WWDCRoom_	960 Bytes	20	0	960 Bytes	20
□	WWDCTrack_WWDCTrack_	448 Bytes	7	0	448 Bytes	7
□	WWDCFocus_WWDCFocus_	96 Bytes	2	0	96 Bytes	2
□	WWDCRoomGroup_WWDCRoomGroup_	64 Bytes	1	0	64 Bytes	1
□	_CDSnapshot_WWDCRoomGroup_	64 Bytes	1	0	64 Bytes	1
□	_CDSnapshot_WWDCRoom_	48 Bytes	1	0	48 Bytes	1
□	WWDCFloor_WWDCFloor_	48 Bytes	1	0	48 Bytes	1

Investigating the Heap

- Your classes!
  - Prefixes are helpful (e.g. ABCViewController)
- New type identifications
  - Better at C++ classes
  - dispatch and xpc types
  - Heap-copied ^blocks (__NSMallocBlock__)

Investigating the Heap

- Your classes!
  - Prefixes are helpful (e.g. ABCViewController)
- New type identifications
  - Better at C++ classes
  - dispatch and xpc types
  - Heap-copied ^blocks (__NSMallocBlock__)

Types of Heap Memory Growth

More memory over time

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

Types of Heap Memory Growth

More memory over time

- 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

Types of Heap Memory Growth

More memory over time

- 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
  - Might never be used again

Generational Analysis

Detecting abandoned memory and excessive caching

Generational Analysis

Detecting abandoned memory and excessive caching

- Technique for measuring memory growth
    1. Reach a steady state
    2. Record first “generation” of active allocations
    3. Perform a series of operations, returning to steady state
    4. Record a new “generation” of incremental allocations
    5. Repeat steps 3 and 4

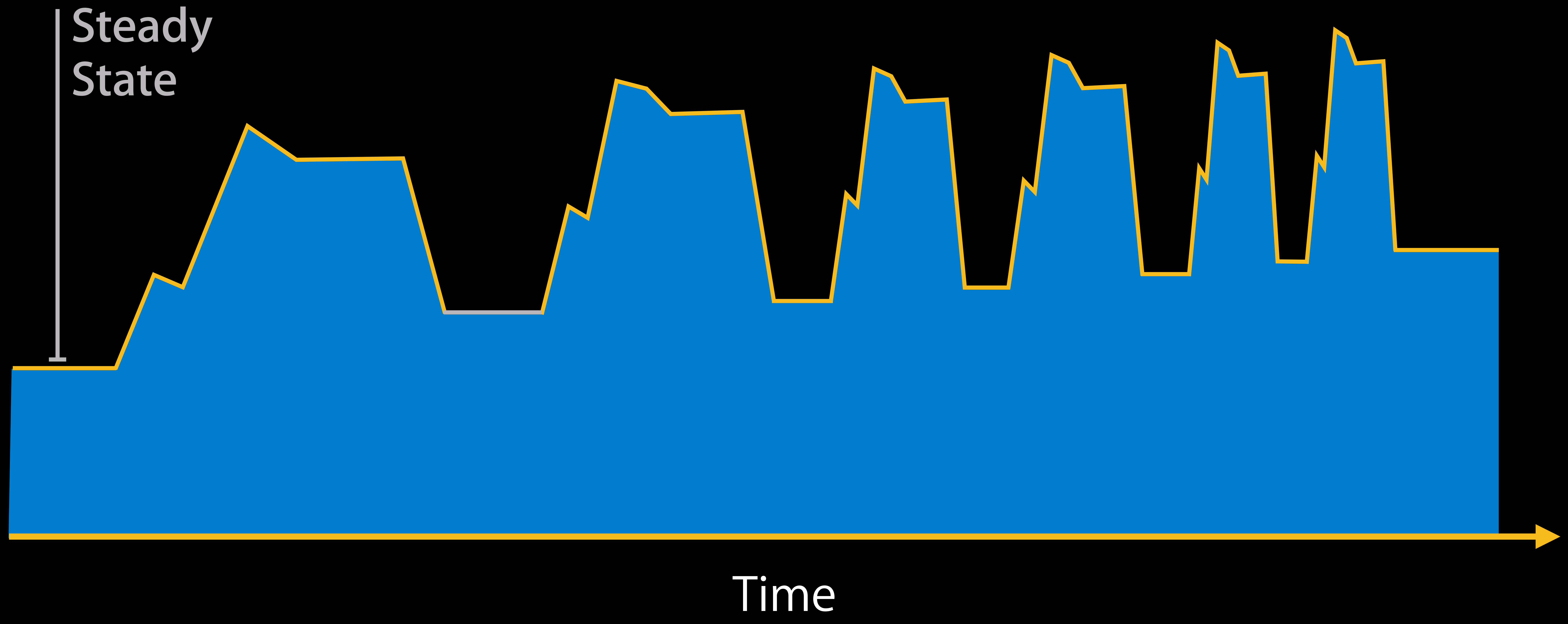
Generational Analysis

Detecting abandoned memory and excessive caching

- Technique for measuring memory growth
  1. Reach a steady state
  2. Record first "generation" of active allocations
  3. Perform a series of operations, returning to steady state
  4. Record a new "generation" of incremental allocations
  5. Repeat steps 3 and 4
- Incremental allocations represent potential problems
  - One-time growth, typical
  - Repeatable memory growth, a real problem

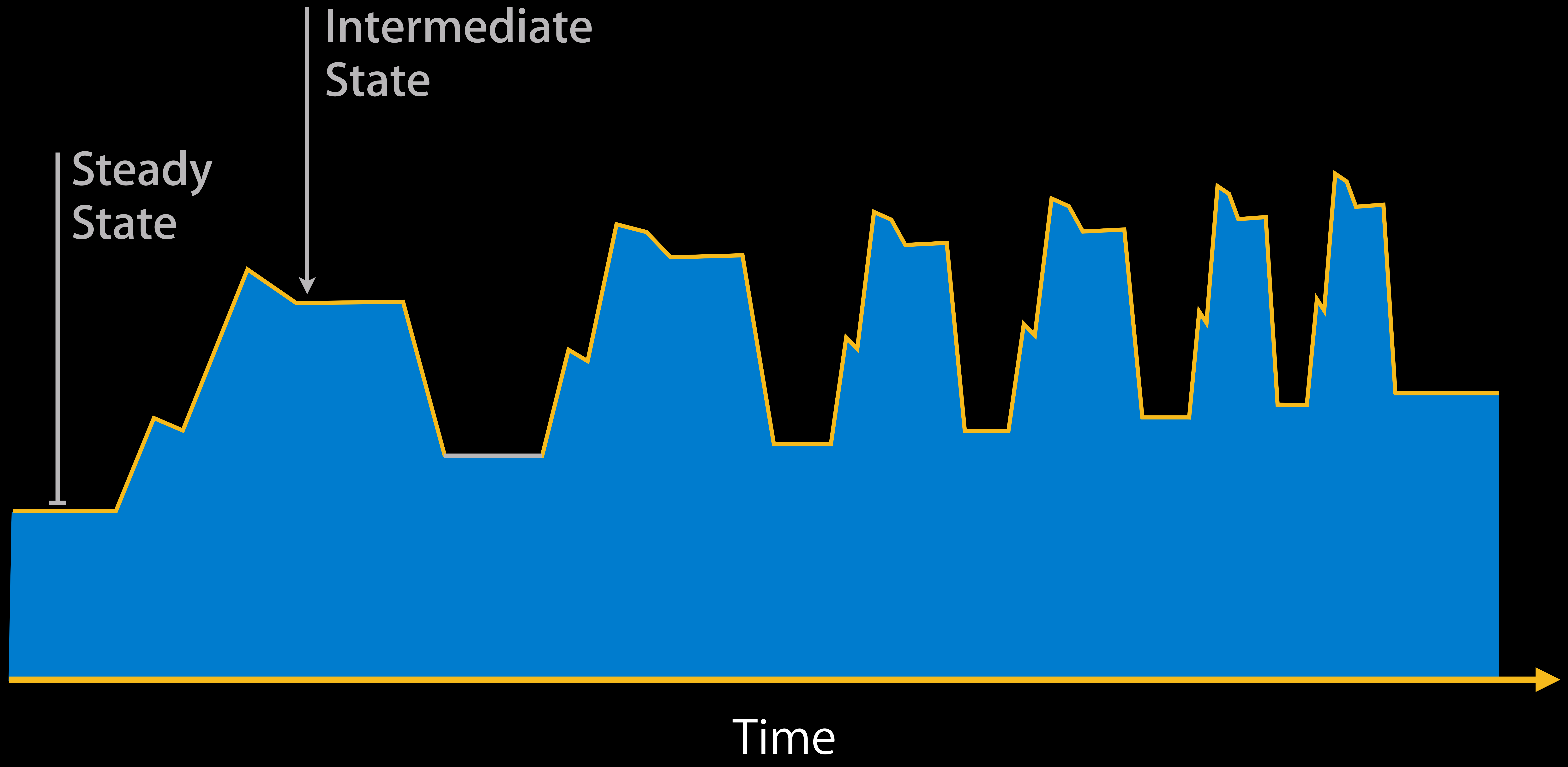
Avoiding Memory Growth

Repetition reveals waste



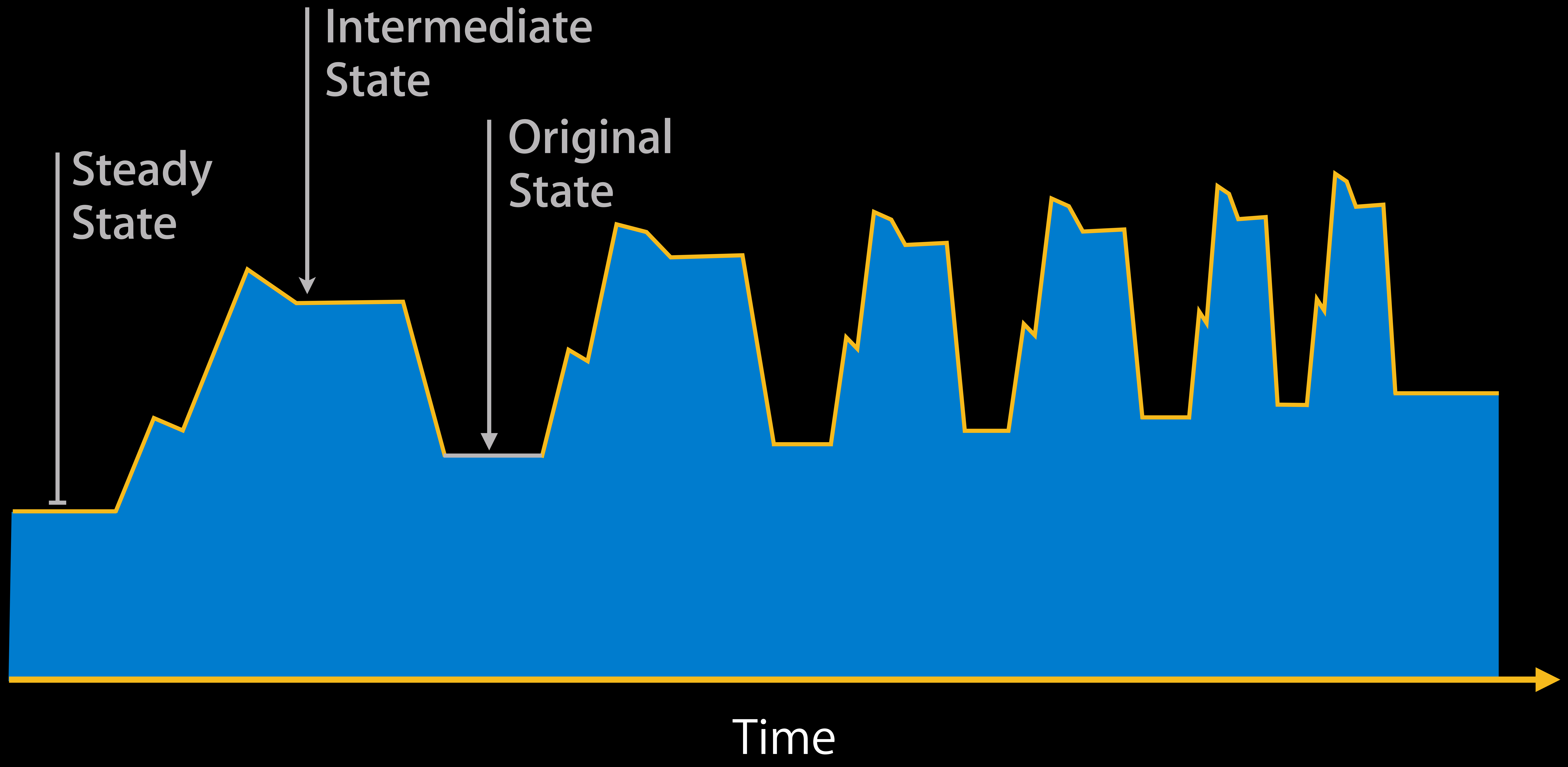
Avoiding Memory Growth

Repetition reveals waste



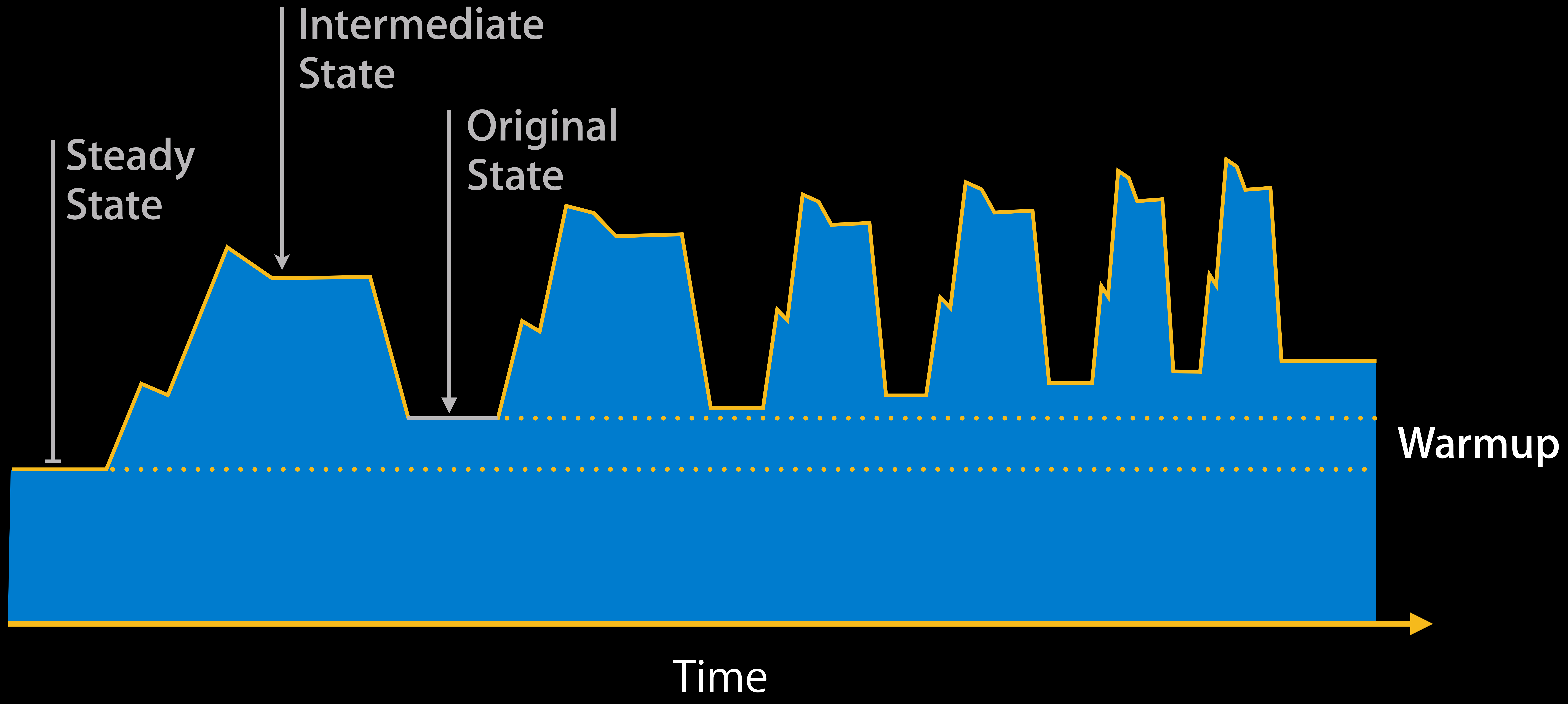
Avoiding Memory Growth

Repetition reveals waste



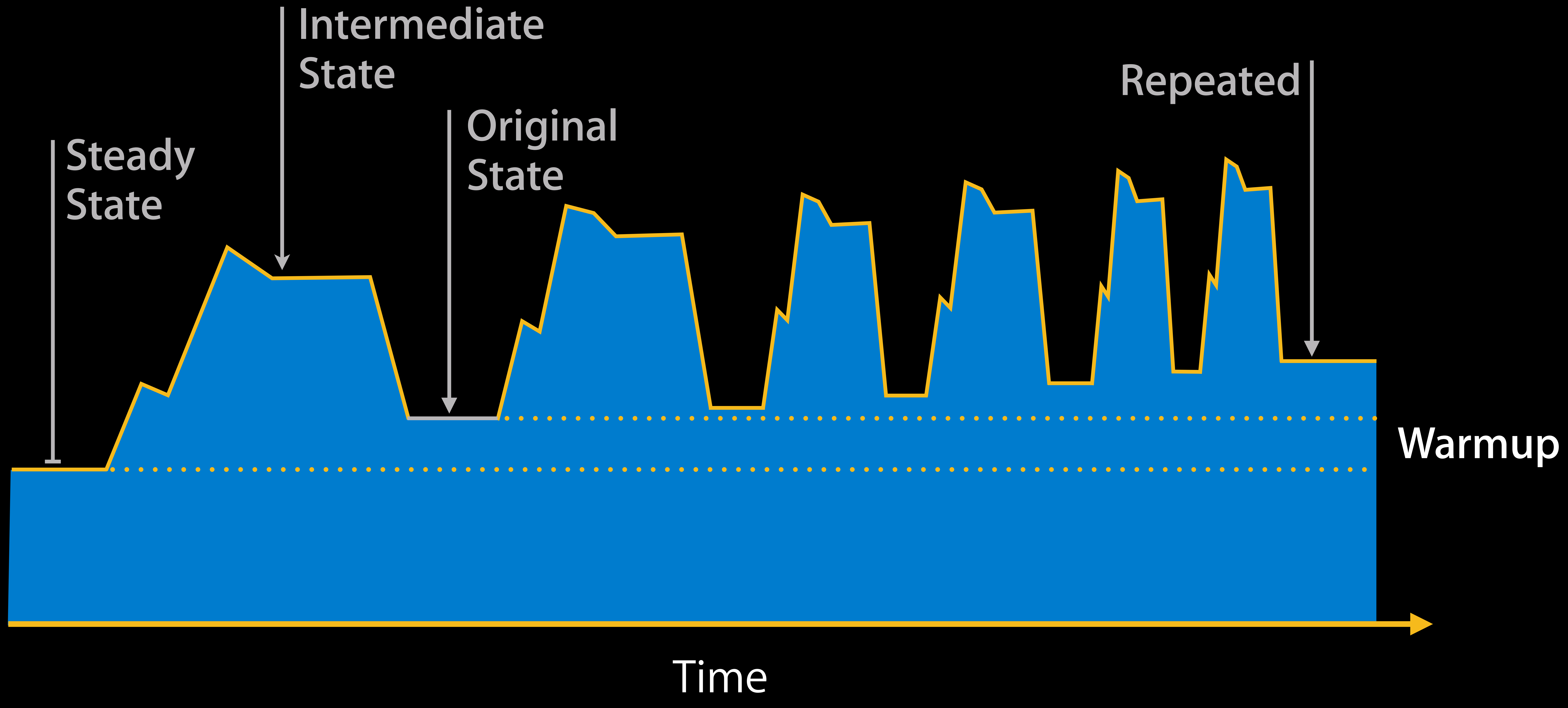
Avoiding Memory Growth

Repetition reveals waste



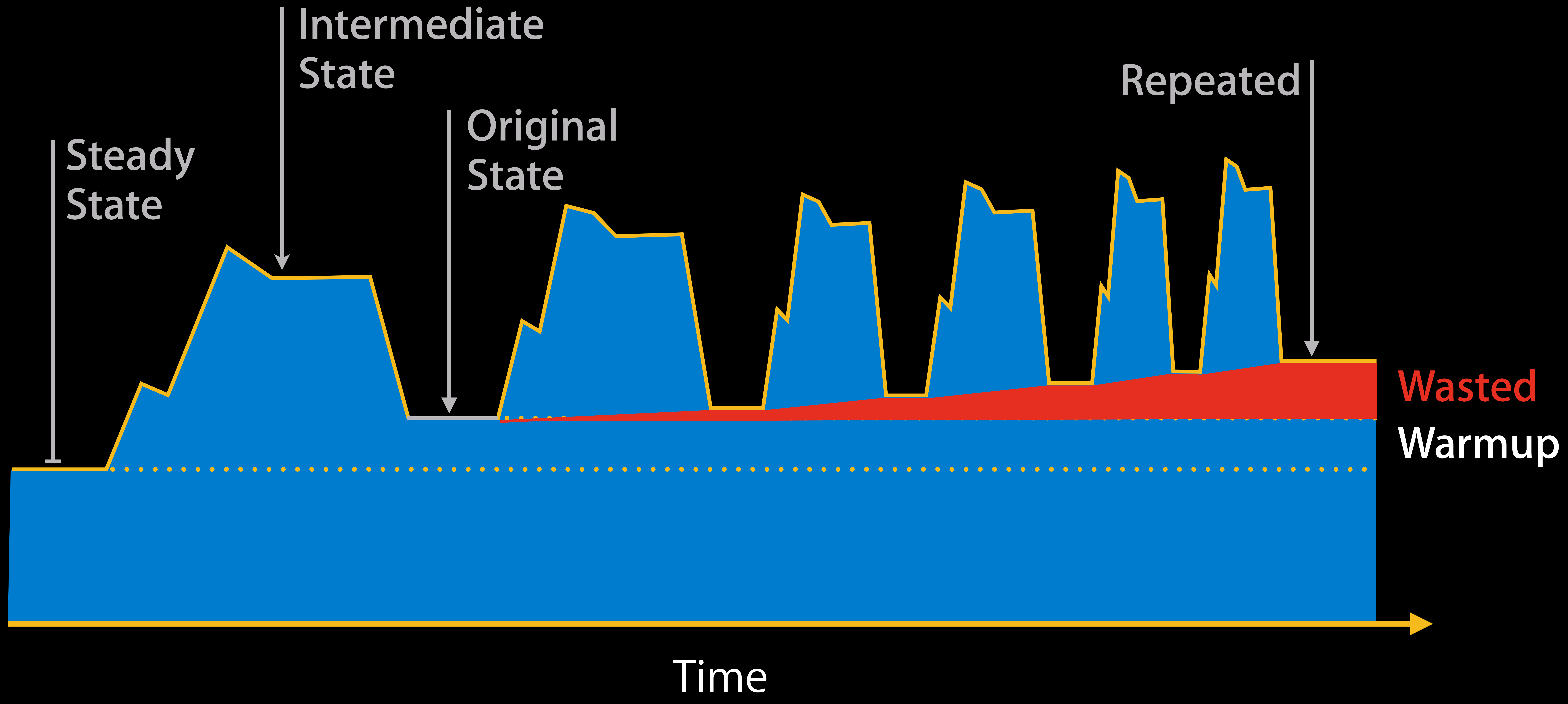
Avoiding Memory Growth

Repetition reveals waste



Avoiding Memory Growth

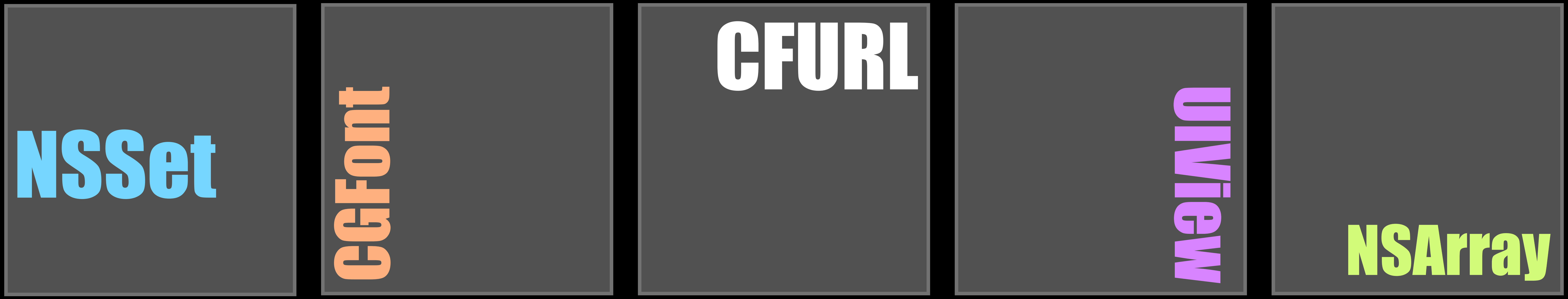
Repetition reveals waste



When Free Memory Isn’t

Heap fragmentation

- Fragmentation is poor utilization of malloc VM regions
- Effectively wasted space
- Impossible for system to reclaim



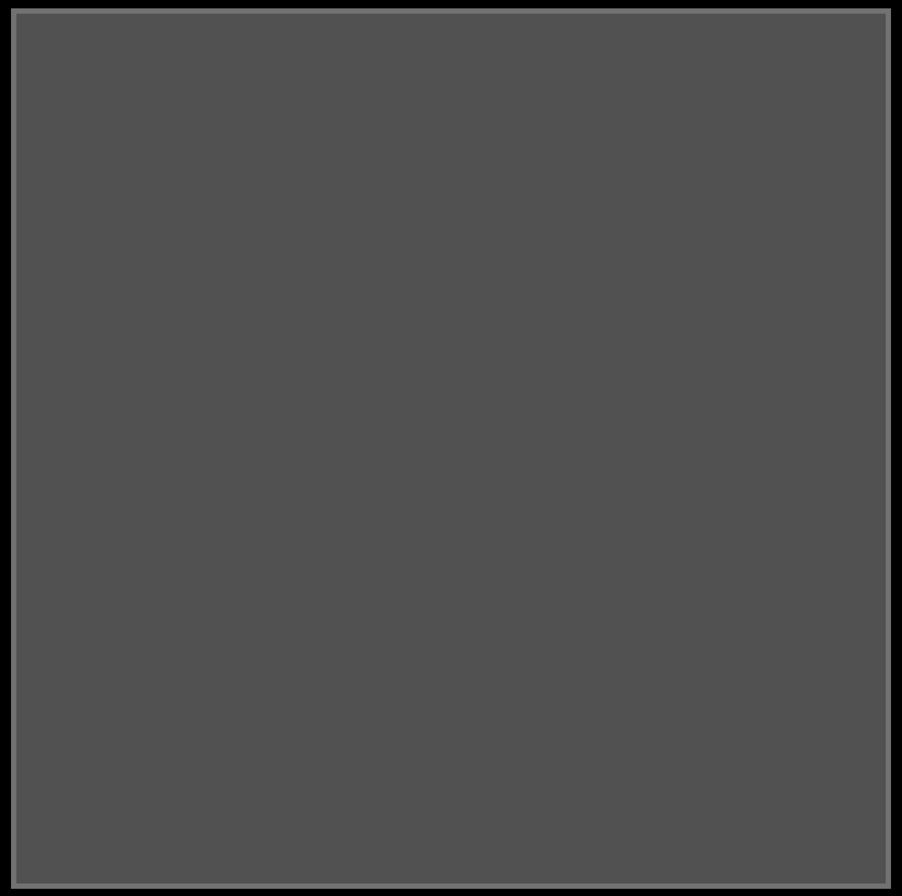
Heap Fragmentation

How it happens

Heap Fragmentation

How it happens

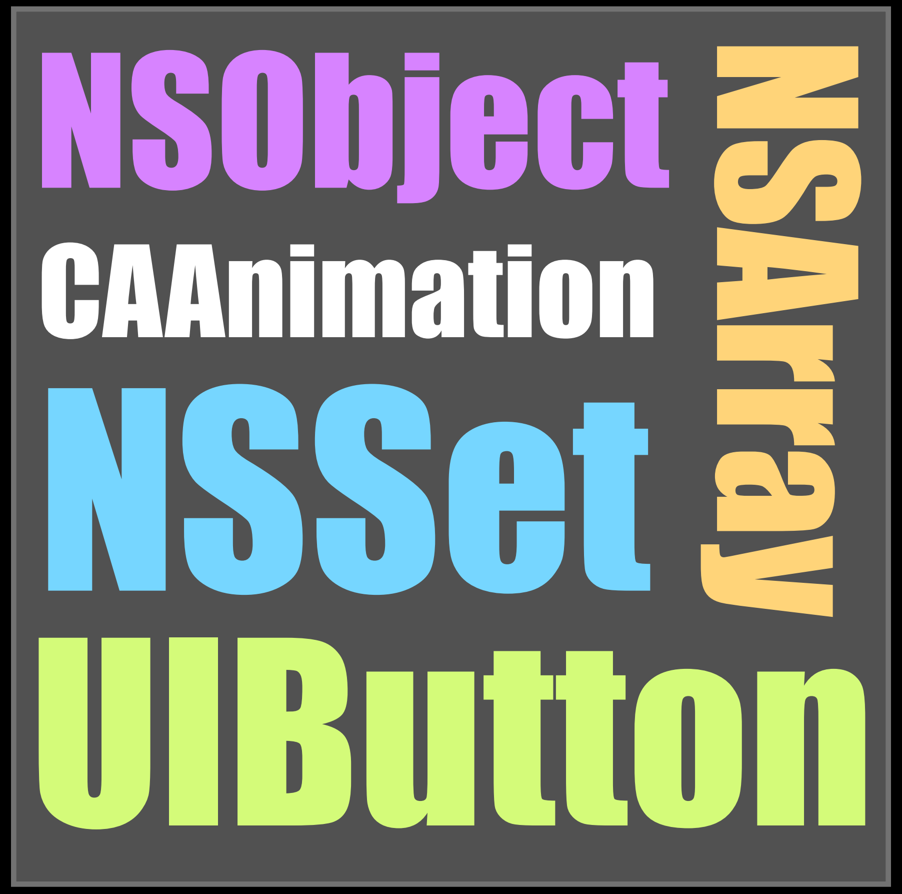
1. New malloc VM region is needed



Heap Fragmentation

How it happens

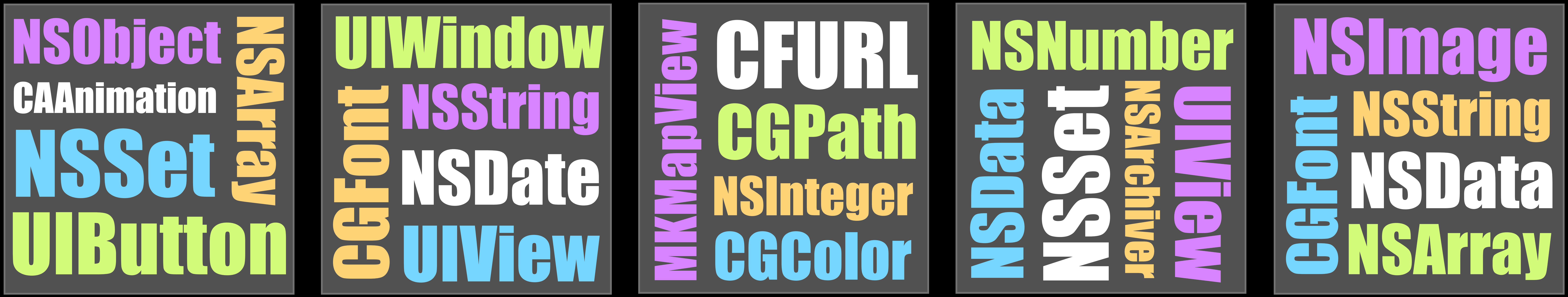
1. New malloc VM region is needed
2. Region is filled until it can’t fit more blocks



Heap Fragmentation

How it happens

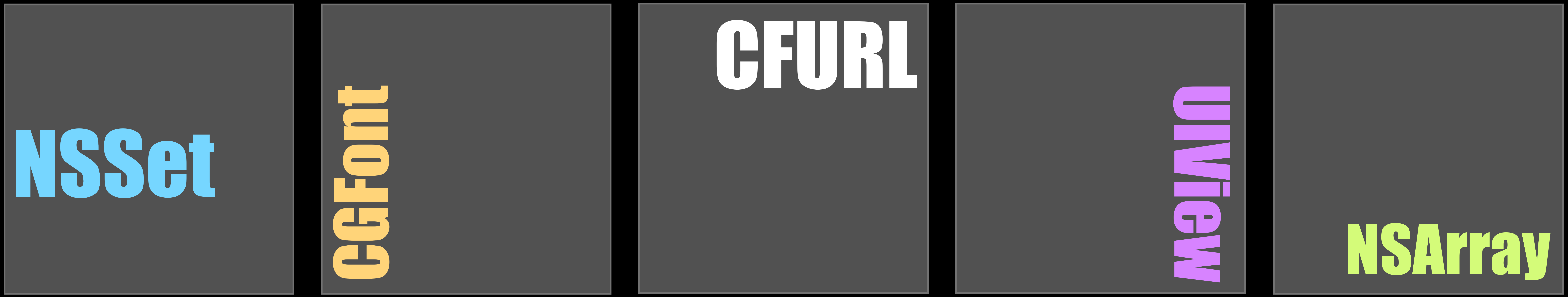
1. New malloc VM region is needed
2. Region is filled until it can't fit more blocks
3. Repeat steps 1 and 2 several times



Heap Fragmentation

How it happens

1. New malloc VM region is needed
2. Region is filled until it can't fit more blocks
3. Repeat steps 1 and 2 several times
4. Most blocks are then freed, but not all



Heap Fragmentation

Avoidance is the best policy

Heap Fragmentation

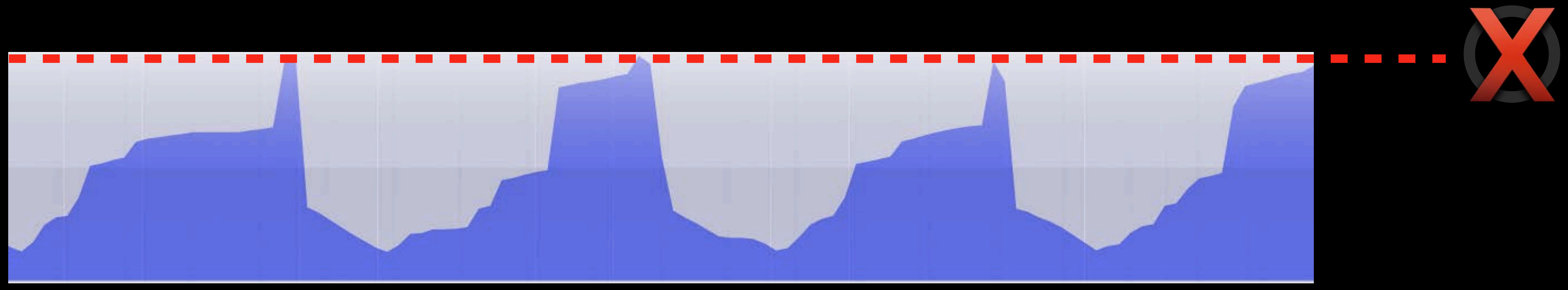
Avoidance is the best policy

- Use Allocations instrument to identify
  - Clear indicator is “All Heap Allocations” graph

Heap Fragmentation

Avoidance is the best policy

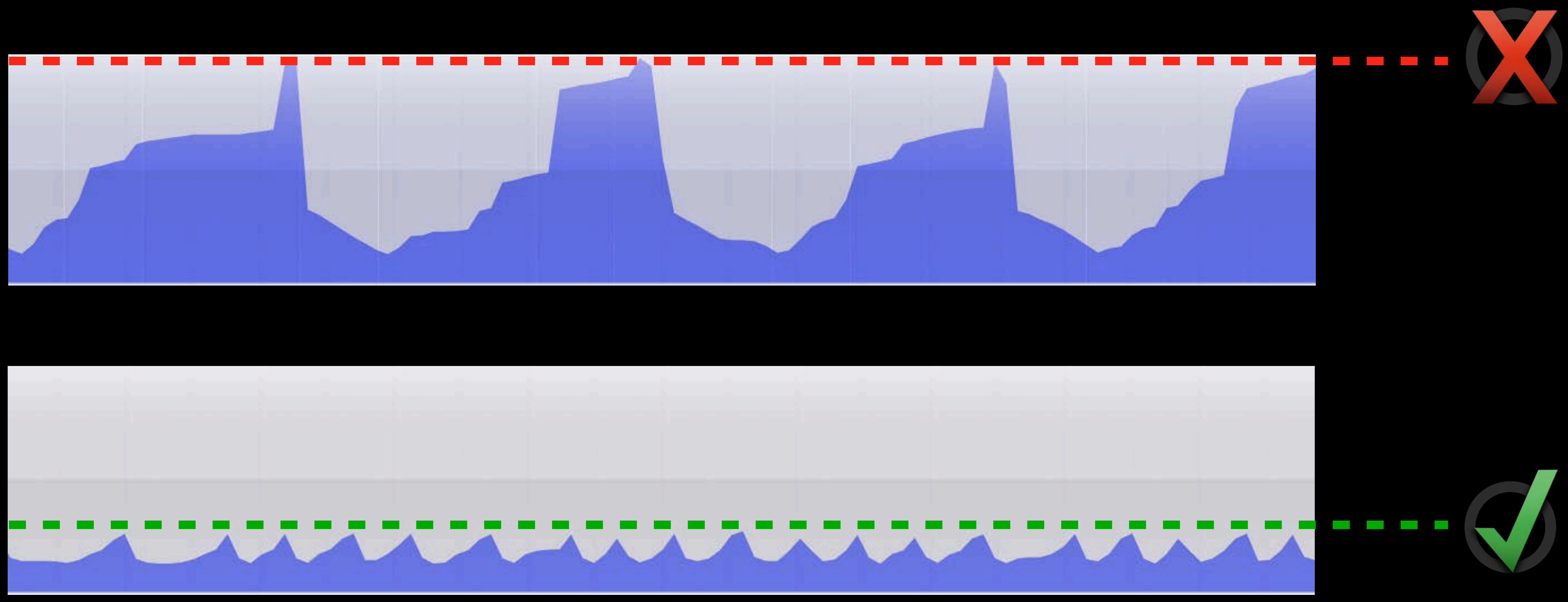
- Use Allocations instrument to identify
  - Clear indicator is “All Heap Allocations” graph



Heap Fragmentation

Avoidance is the best policy

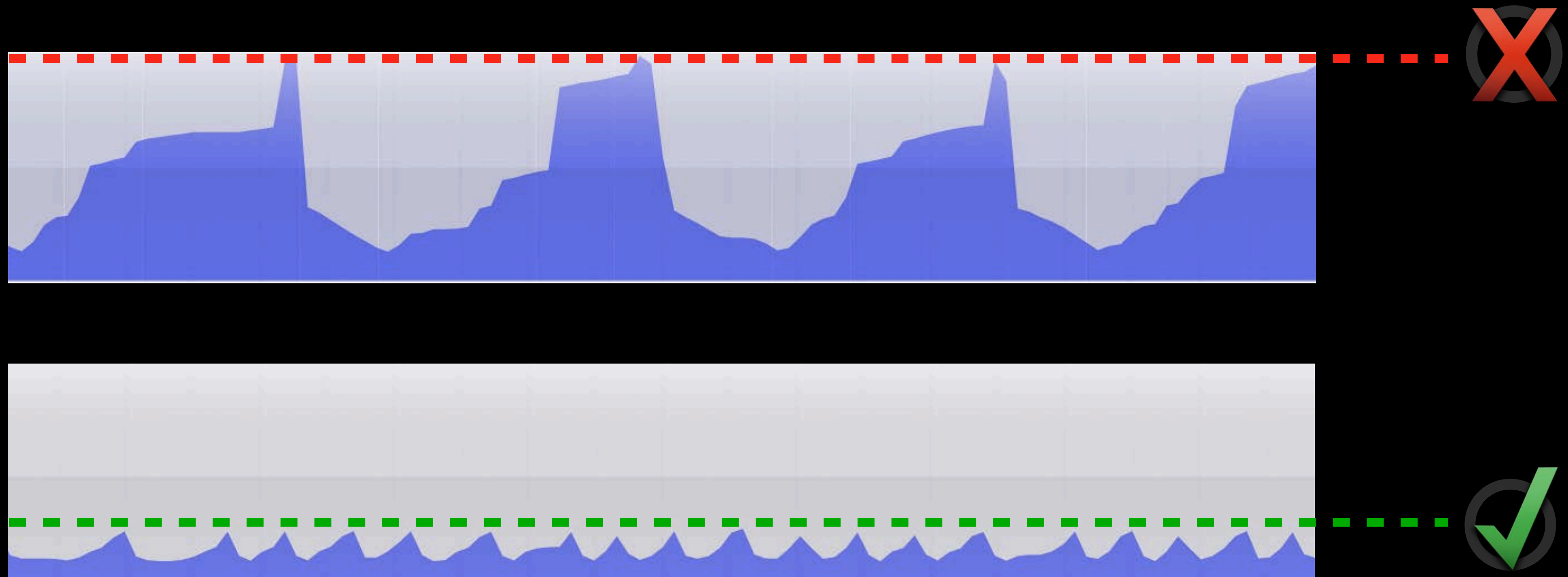
- Use Allocations instrument to identify
    - Clear indicator is “All Heap Allocations” graph



Heap Fragmentation

Avoidance is the best policy

- Use Allocations instrument to identify
  - Clear indicator is “All Heap Allocations” graph



- Objective-C @autoreleasepool can help

Objective-C, Retain/Release

Common problems and patterns

Daniel Delwood
Software Radiologist

Objective-C’s Ownership Model

Retain/Release

- Reference counting ownership model based on **-retain**, **-release**
  - When the count drops to zero, object is freed
  - **-autorelease** just a delayed release
  - Retain/release/autorelease rules established and easy to learn

Objective-C's Ownership Model

Retain/Release

- Reference counting ownership model based on **-retain**, **-release**
    - When the count drops to zero, object is freed
    - **-autorelease** just a delayed release
    - Retain/release/autorelease rules established and easy to learn
        - *Advanced Memory Management Programming Guide*

Objective-C’s Ownership Model

Retain/Release

- Reference counting ownership model based on **-retain**, **-release**
  - When the count drops to zero, object is freed
  - **-autorelease** just a delayed release
  - Retain/release/autorelease rules established and easy to learn
    - *Advanced Memory Management Programming Guide*
- Deterministic, simple, and fast

Objective-C’s Ownership Model

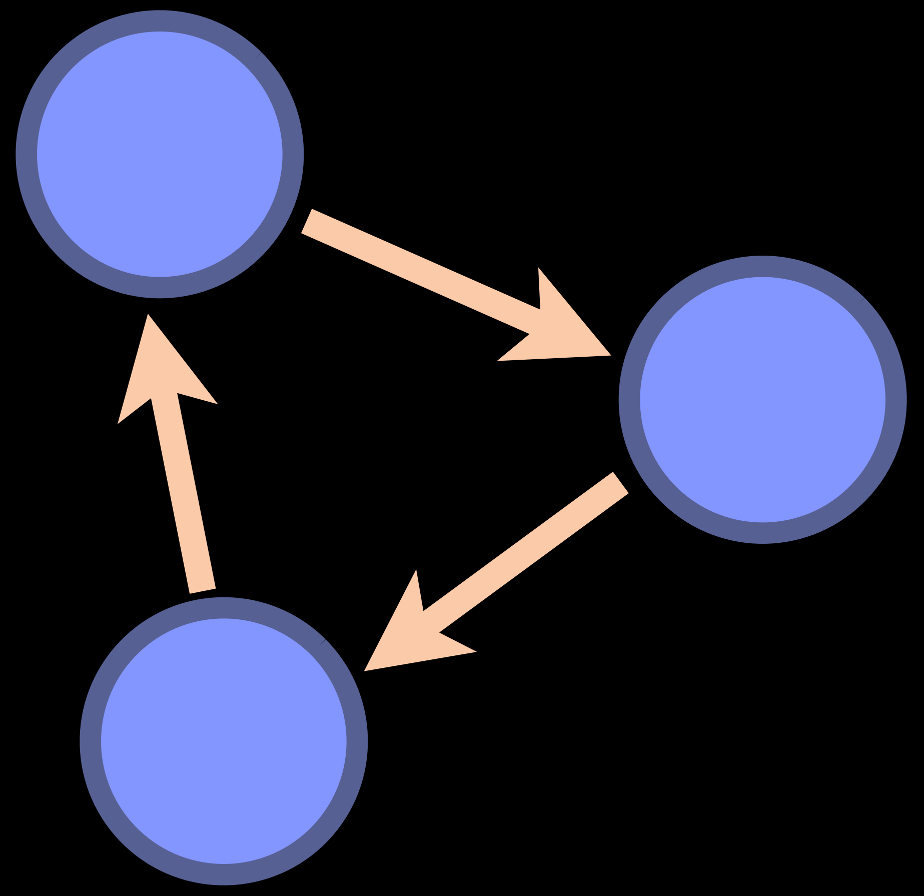
Understanding ARC

- Manual -retain/-release is tedious, error-prone
    - Too many retains → leaks
    - Too many releases → crashes

Objective-C’s Ownership Model

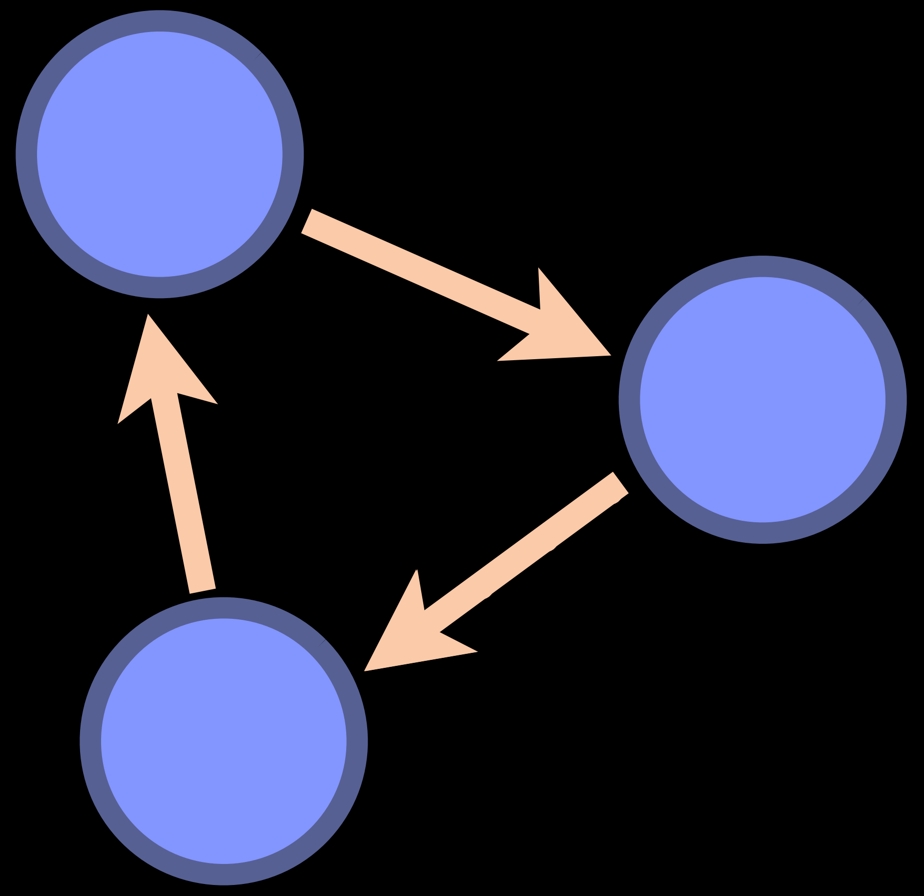
Understanding ARC

- Manual -retain/-release is tedious, error-prone
    - Too many retains → leaks
    - Too many releases → crashes
    - Retain cycles → more leaks



Objective-C's Ownership Model

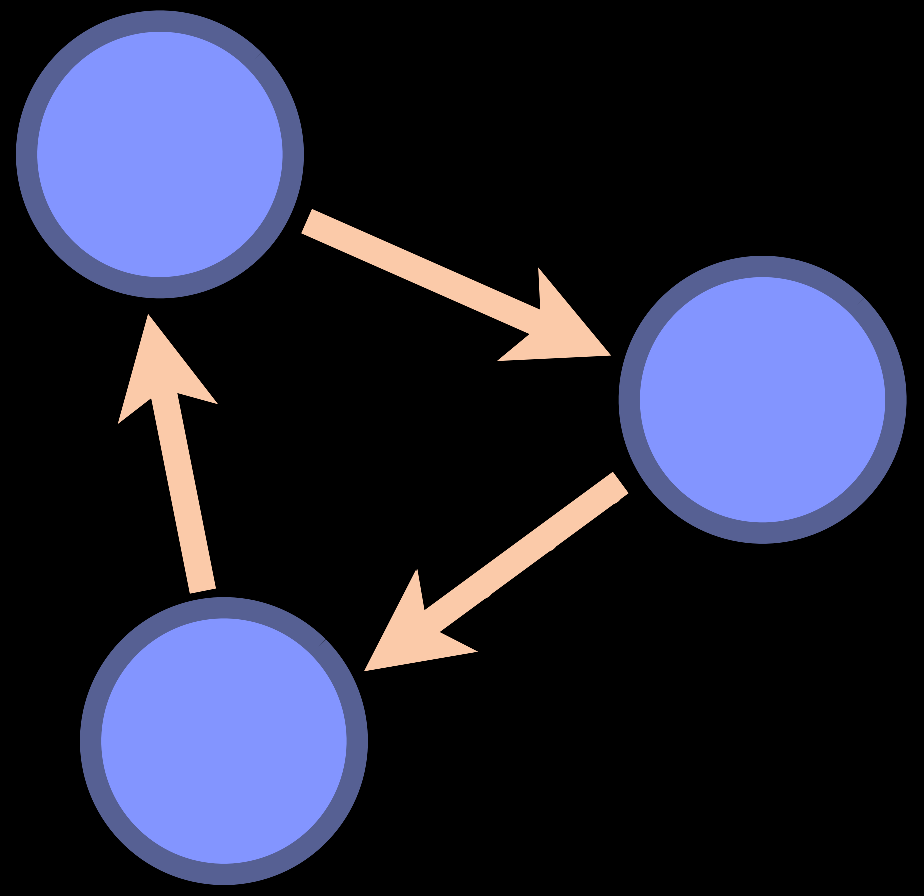
Understanding ARC



- Manual -retain/-release is tedious, error-prone
  - Too many retains → leaks
  - Too many releases → crashes
  - Retain cycles → more leaks
- Automatic Reference Counting (ARC)
  - Compiler-assisted -retain/-release convention enforcement

Objective-C's Ownership Model

Understanding ARC

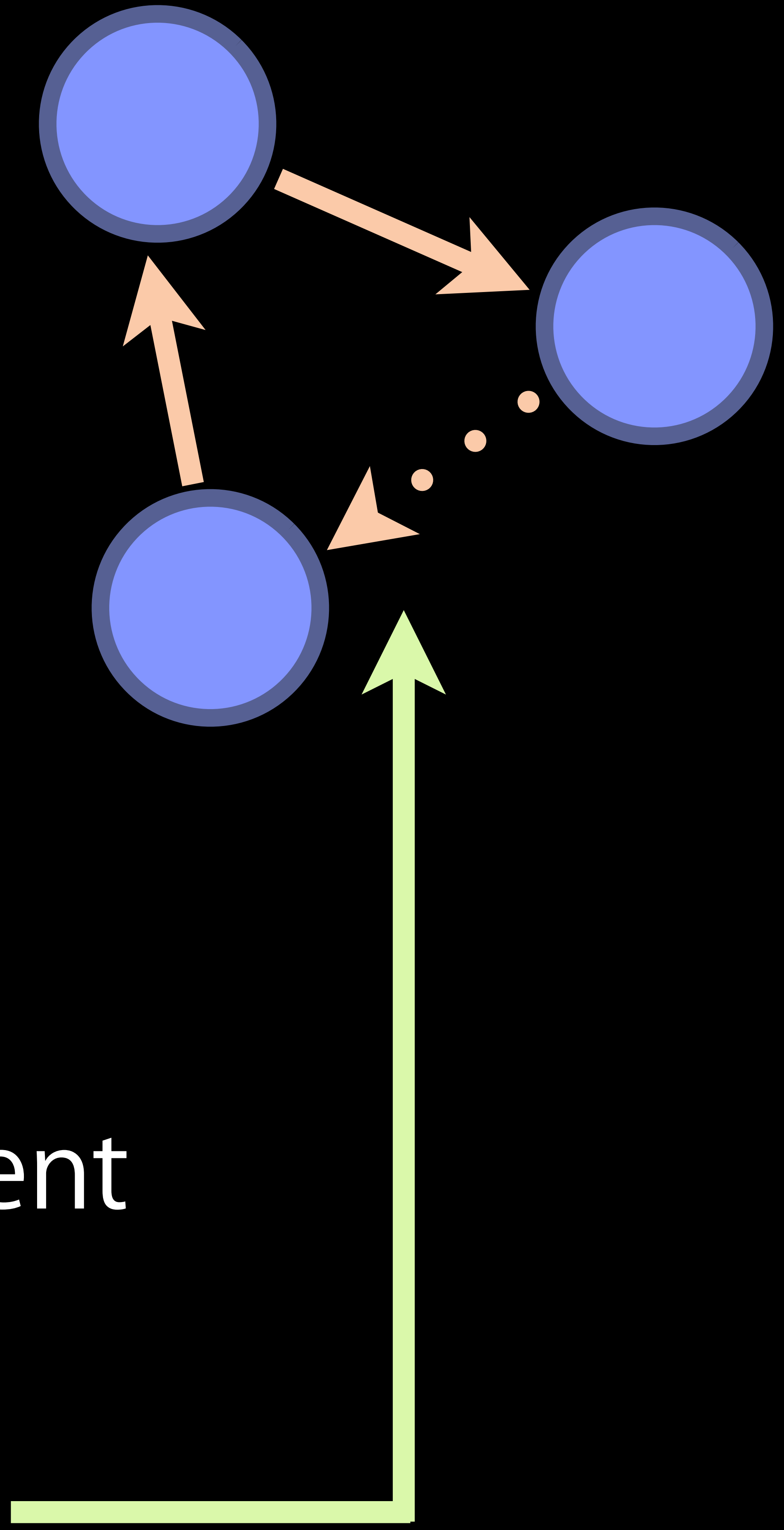


- Manual -retain/-release is tedious, error-prone
  - Too many retains → leaks
  - Too many releases → crashes
  - Retain cycles → more leaks
- Automatic Reference Counting (ARC)
  - Compiler-assisted -retain/-release convention enforcement
  - Doesn't solve retain cycles on its own

Objective-C's Ownership Model

Understanding ARC

- Manual -retain/-release is tedious, error-prone
  - Too many retains → leaks
  - Too many releases → crashes
  - Retain cycles → more leaks
- Automatic Reference Counting (ARC)
  - Compiler-assisted -retain/-release convention enforcement
  - Doesn't solve retain cycles on its own
  - Provides additional tools like zeroing-weak references



Objective-C's Ownership Model

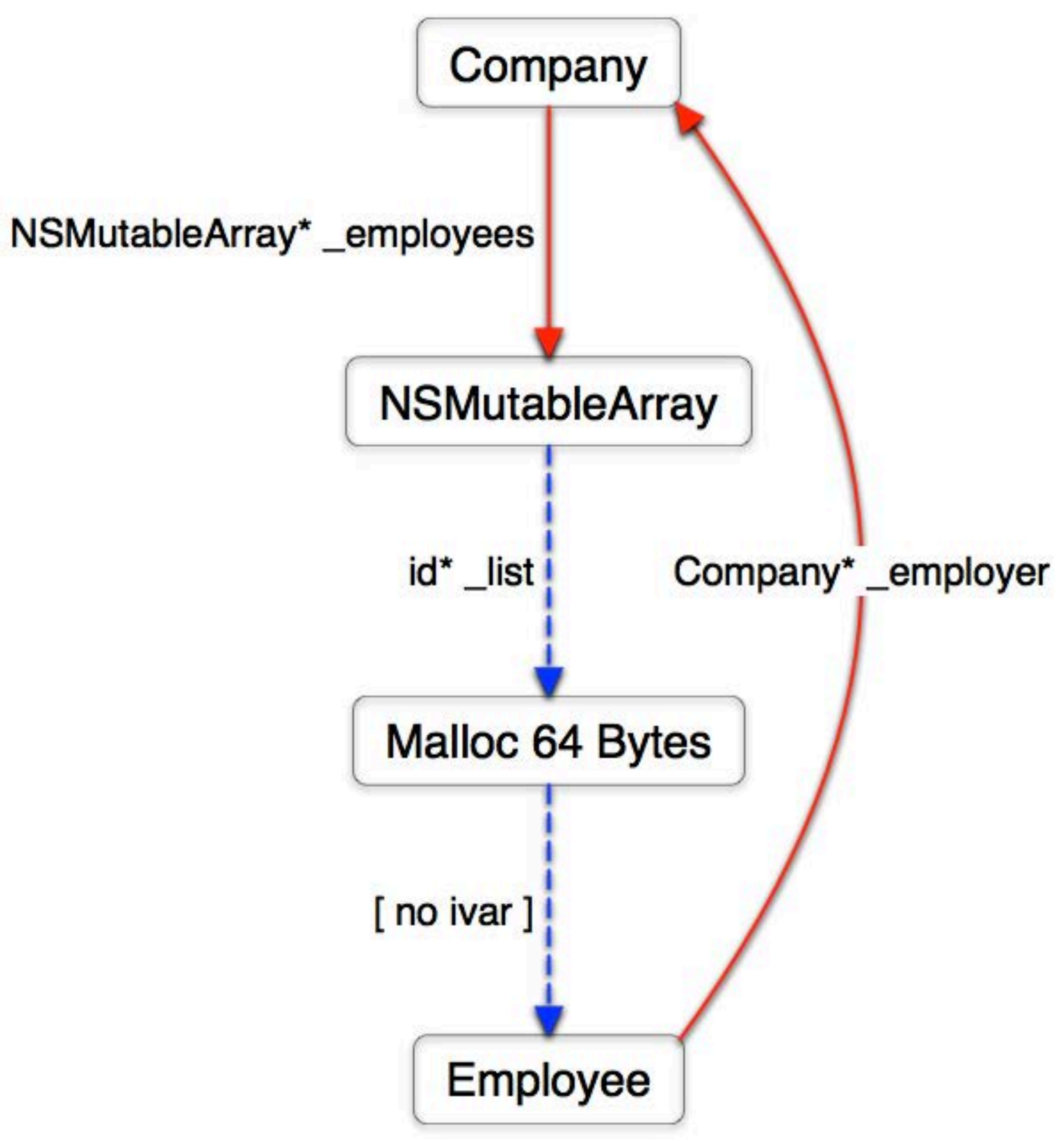
Common problems under ARC

- Memory growth
  - Unreferenced retain cycles → Leaks template
  - Abandoned objects → Generational analysis
- Messaging deallocated objects
  - Undefined/non-deterministic behavior
  - Best case: reproducible crashes — usually in:
    - objc_* (e.g. objc_msgSend, objc_storeStrong)
    - -[NSObject doesNotRespondToSelector:]
  - Worst case: works 99% of the time

Objective-C’s Ownership Model

Common problems under ARC

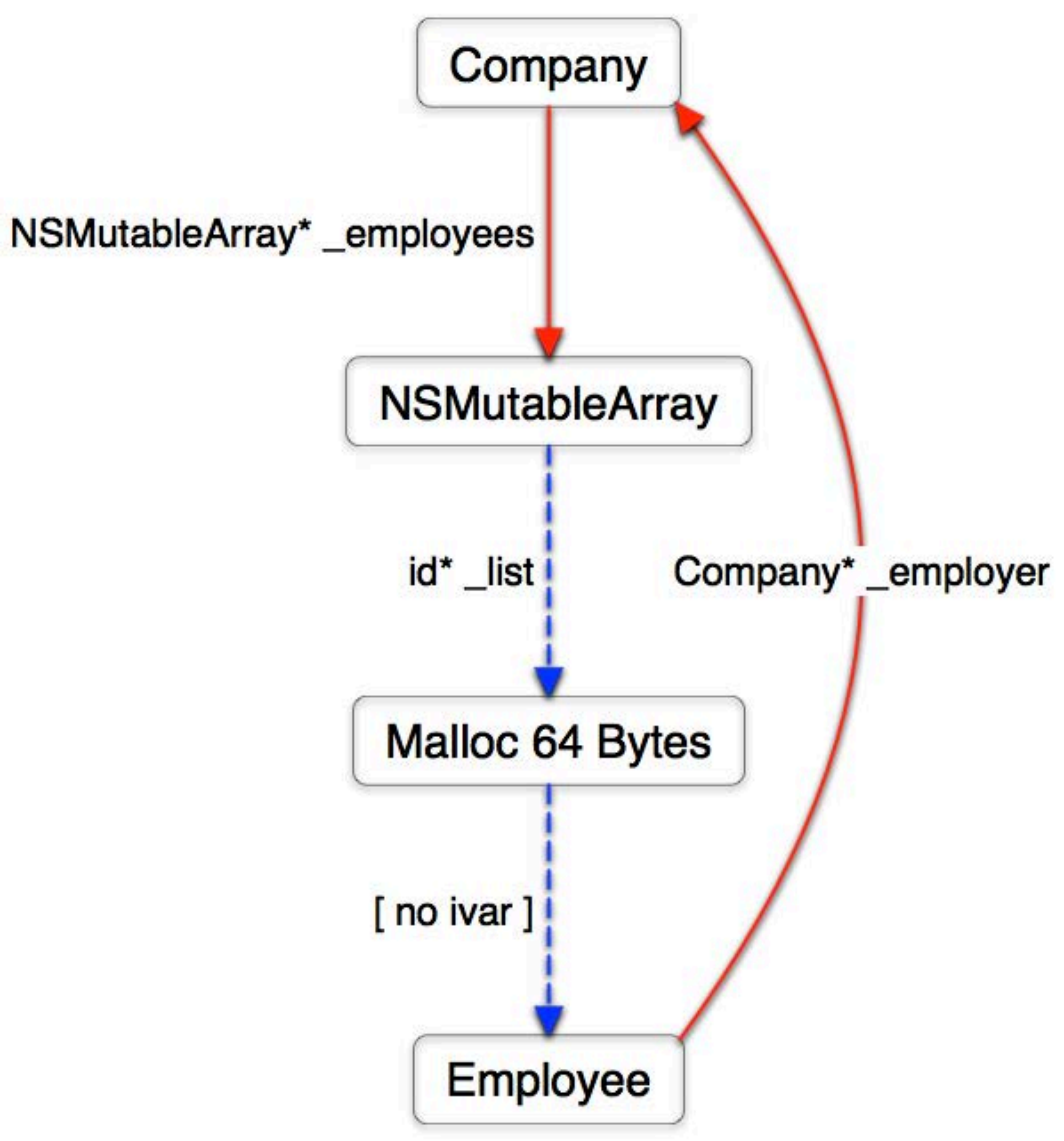
- Memory growth
  - Unreferenced retain cycles → Leaks template
  - Abandoned objects → Generational analysis
- Messaging deallocated objects
  - Undefined/non-deterministic behavior
  - Best case: reproducible crashes — usually in:
    - objc_* (e.g. objc_msgSend, objc_storeStrong)
    - -[NSObject doesNotRespondToSelector:]
  - Worst case: works 99% of the time



Objective-C's Ownership Model

Common problems under ARC

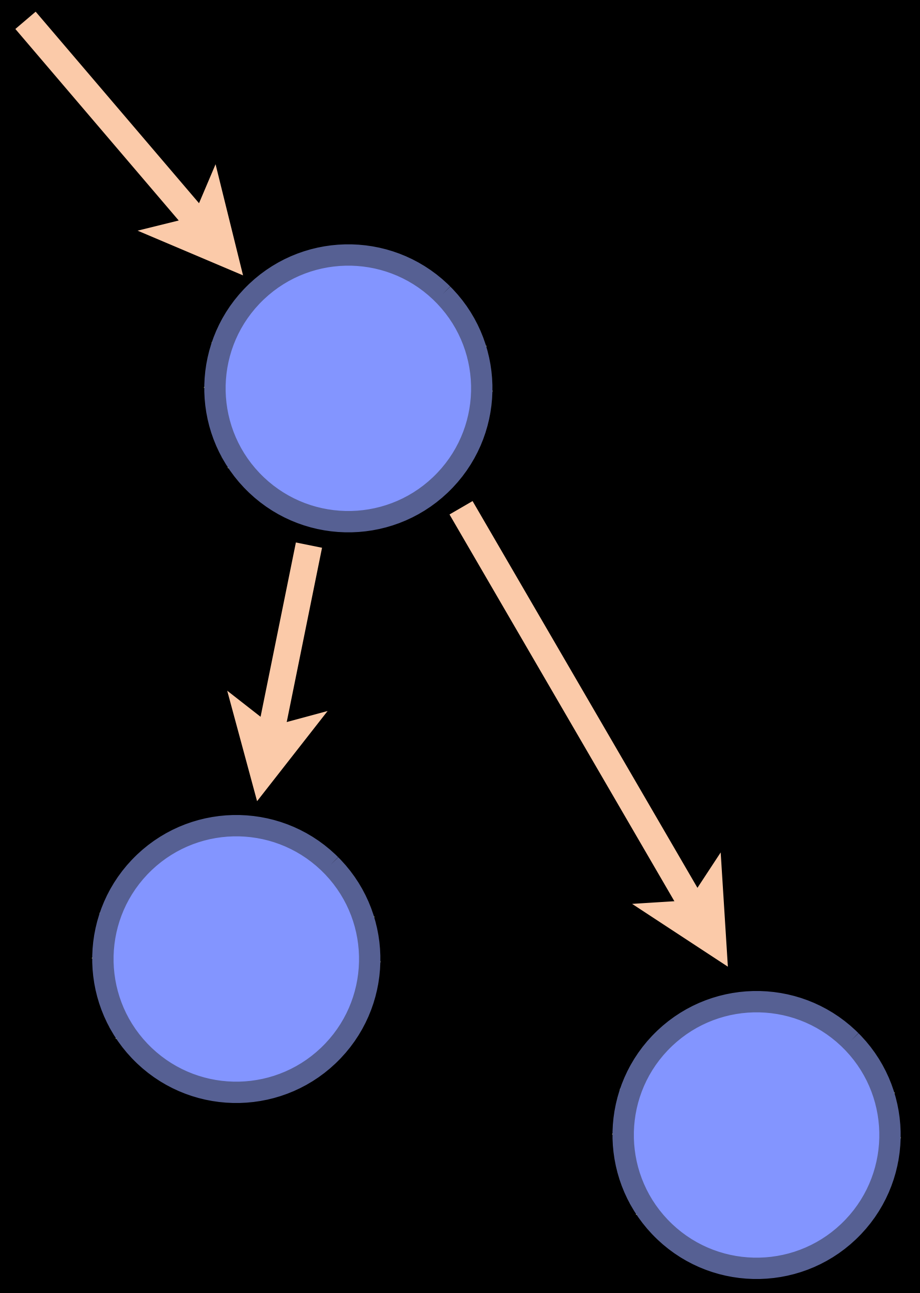
- Memory growth
  - Unreferenced retain cycles → Leaks template
  - Abandoned objects → Generational analysis
- Messaging deallocated objects
  - Undefined/non-deterministic behavior
  - Best case: reproducible crashes — usually in:
    - objc_* (e.g. objc_msgSend, objc_storeStrong)
    - -[NSObject doesNotRespondToSelector:]
  - Worst case: works 99% of the time



Messaging Deallocated Objects

Seeking predictability and fixes

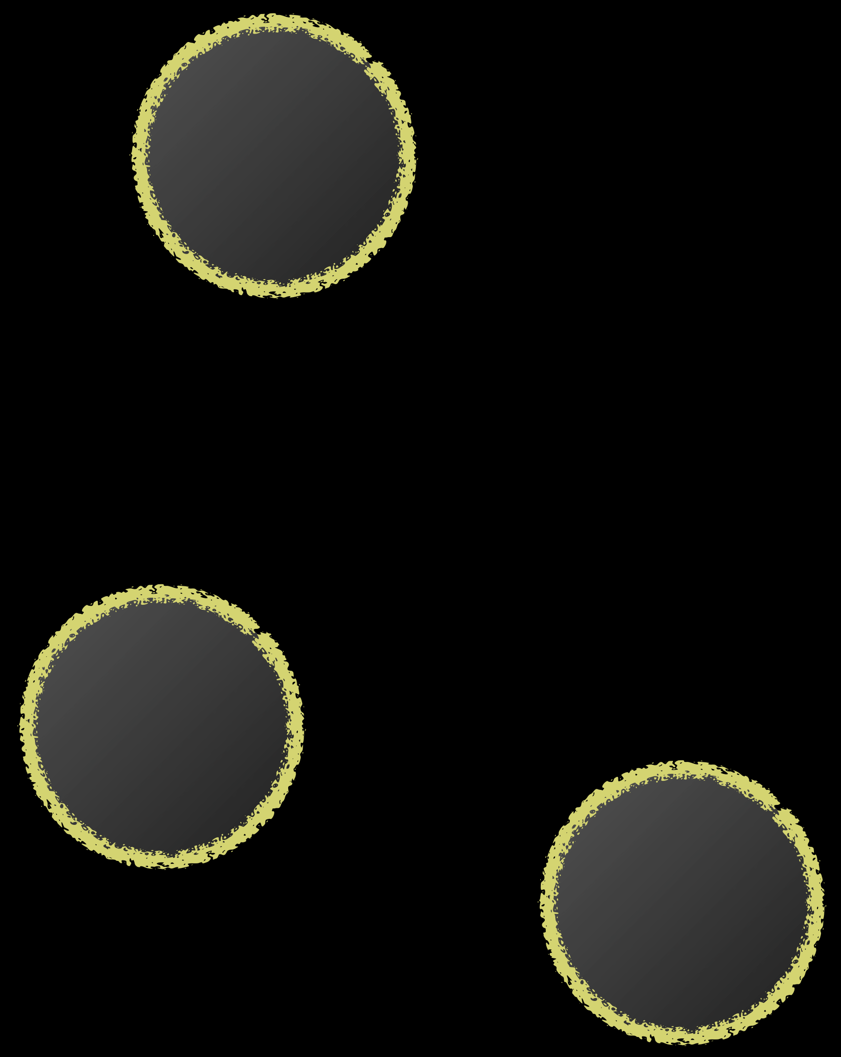
- Zombies template
  - Debugging environment: NSZombieEnabled=1



Messaging Deallocated Objects

Seeking predictability and fixes

- Zombies template
  - Debugging environment: NSZombieEnabled=1
  - Instead of being deallocated, objects changed into “Zombies”



Messaging Deallocated Objects

Seeking predictability and fixes

- Zombies template
  - Debugging environment: NSZombieEnabled=1
  - Instead of being deallocated, objects changed into “Zombies”
  - Zombies objects always crash when messaged

-[NSObject description] →

Messaging Deallocated Objects

Seeking predictability and fixes

- Zombies template
    - Debugging environment: NSZombieEnabled=1
    - Instead of being deallocated, objects changed into “Zombies”
    - Zombies objects always crash when messaged
- Implications
    - More deterministic — memory isn’t unchanged or reused
    - Every zombie object leaks
        - Don’t use Zombies and Leaks together

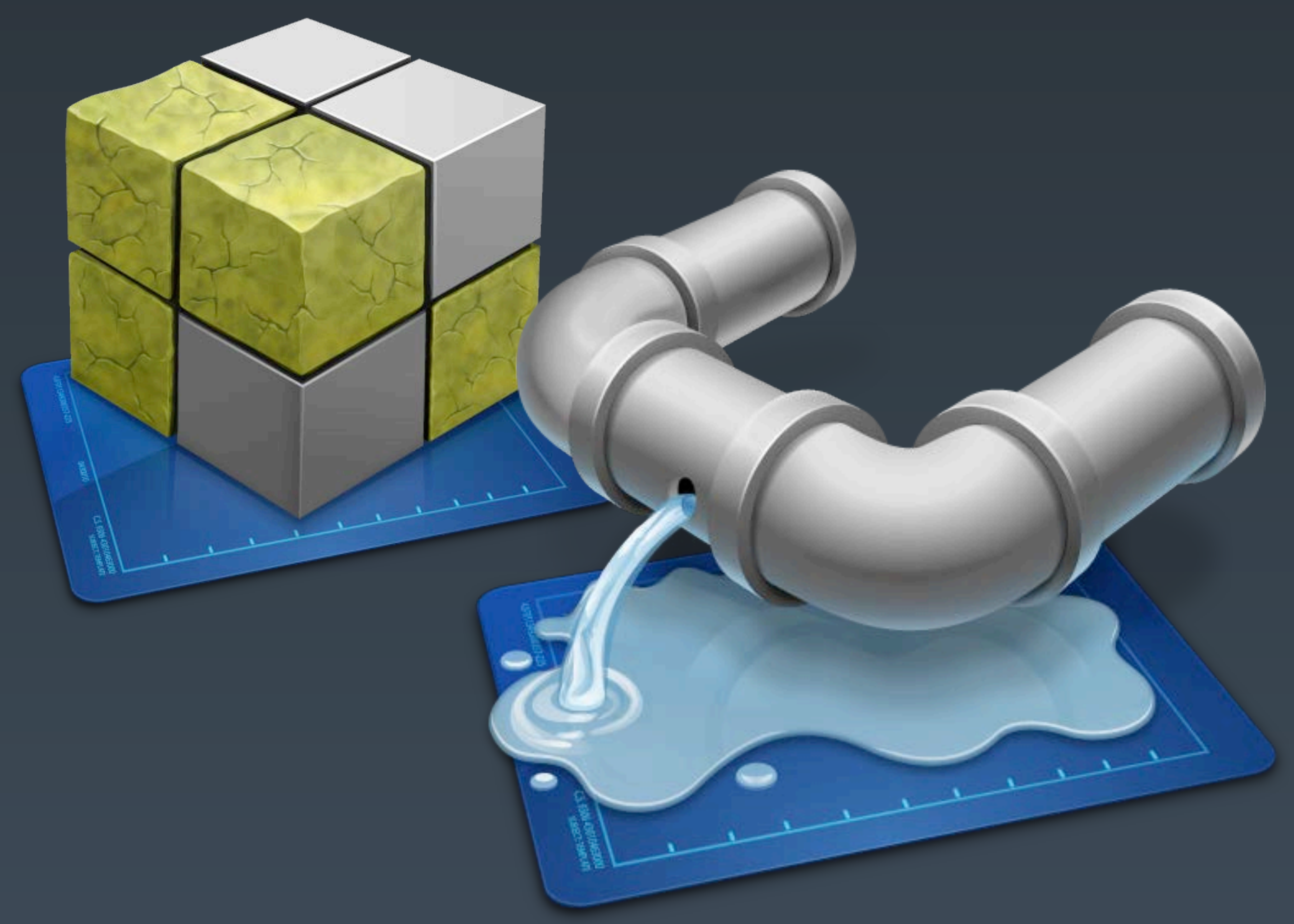
Messaging Deallocated Objects

Seeking predictability and fixes

- Zombies template
  - Debugging environment: NSZombieEnabled=1
  - Instead of being deallocated, objects changed into “Zombies”
  - Zombies objects always crash when messaged
- Implications
  - More deterministic — memory isn’t unchanged or reused
  - Every zombie object leaks
    - Don’t use Zombies and Leaks together
- Now available for iOS 7 devices

Demo

Retain/Release, leaks and crashes



Applying It to Your App

Steps to fix leaks/crashes

- Switch to ARC
- Run the Static Analyzer

Applying It to Your App

Steps to fix leaks/crashes

- Switch to ARC
- Run the Static Analyzer
- Zombies template is a great first resort for crashes

Applying It to Your App

Steps to fix leaks/crashes

- Switch to ARC
- Run the Static Analyzer
- Zombies template is a great first resort for crashes
- Backtrace for +alloc does not tell the whole story

Applying It to Your App

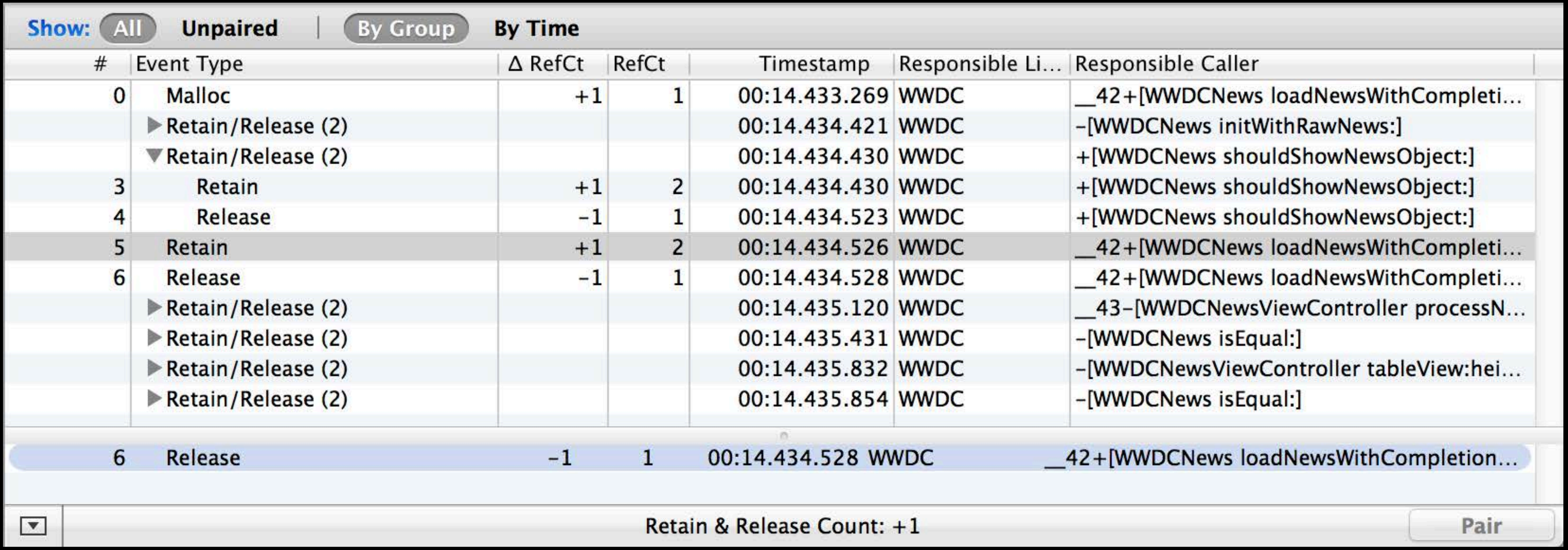
Steps to fix leaks/crashes

- Switch to ARC
- Run the Static Analyzer
- Zombies template is a great first resort for crashes
- Backtrace for +alloc does not tell the whole story
- Save time by pairing Retain/Releases

Retain/Release Pairing

Needle in a smaller haystack

- Manual pairing assistant
- Heuristic-based automatic pairing (better in ARC and -o0)



#	Event Type	Δ RefCt	RefCt	Timestamp	Responsible Li...	Responsible Caller
0	Malloc	+1	1	00:14.433.269	WWDC	__42+[WWDCNews loadNewsWithCompleti...
	▶ Retain/Release (2)			00:14.434.421	WWDC	-[WWDCNews initWithRawNews:]
	▼ Retain/Release (2)			00:14.434.430	WWDC	+[WWDCNews shouldShowNewsObject:]
3	Retain	+1	2	00:14.434.430	WWDC	+[WWDCNews shouldShowNewsObject:]
4	Release	-1	1	00:14.434.523	WWDC	+[WWDCNews shouldShowNewsObject:]
5	Retain	+1	2	00:14.434.526	WWDC	__42+[WWDCNews loadNewsWithCompleti...
6	Release	-1	1	00:14.434.528	WWDC	__42+[WWDCNews loadNewsWithCompleti...
	▶ Retain/Release (2)			00:14.435.120	WWDC	__43-[WWDCNewsViewController processN...
	▶ Retain/Release (2)			00:14.435.431	WWDC	-[WWDCNews isEqual:]
	▶ Retain/Release (2)			00:14.435.832	WWDC	-[WWDCNewsViewController tableView:hei...
	▶ Retain/Release (2)			00:14.435.854	WWDC	-[WWDCNews isEqual:]

6	Release	-1	1	00:14.434.528	WWDC	__42+[WWDCNews loadNewsWithCompletion...

Retain & Release Count: +1

Retain/Release Pairing

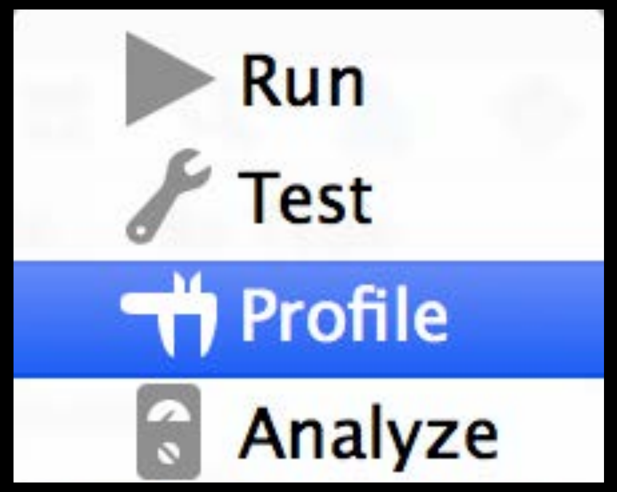
Profiling “Debug” configuration

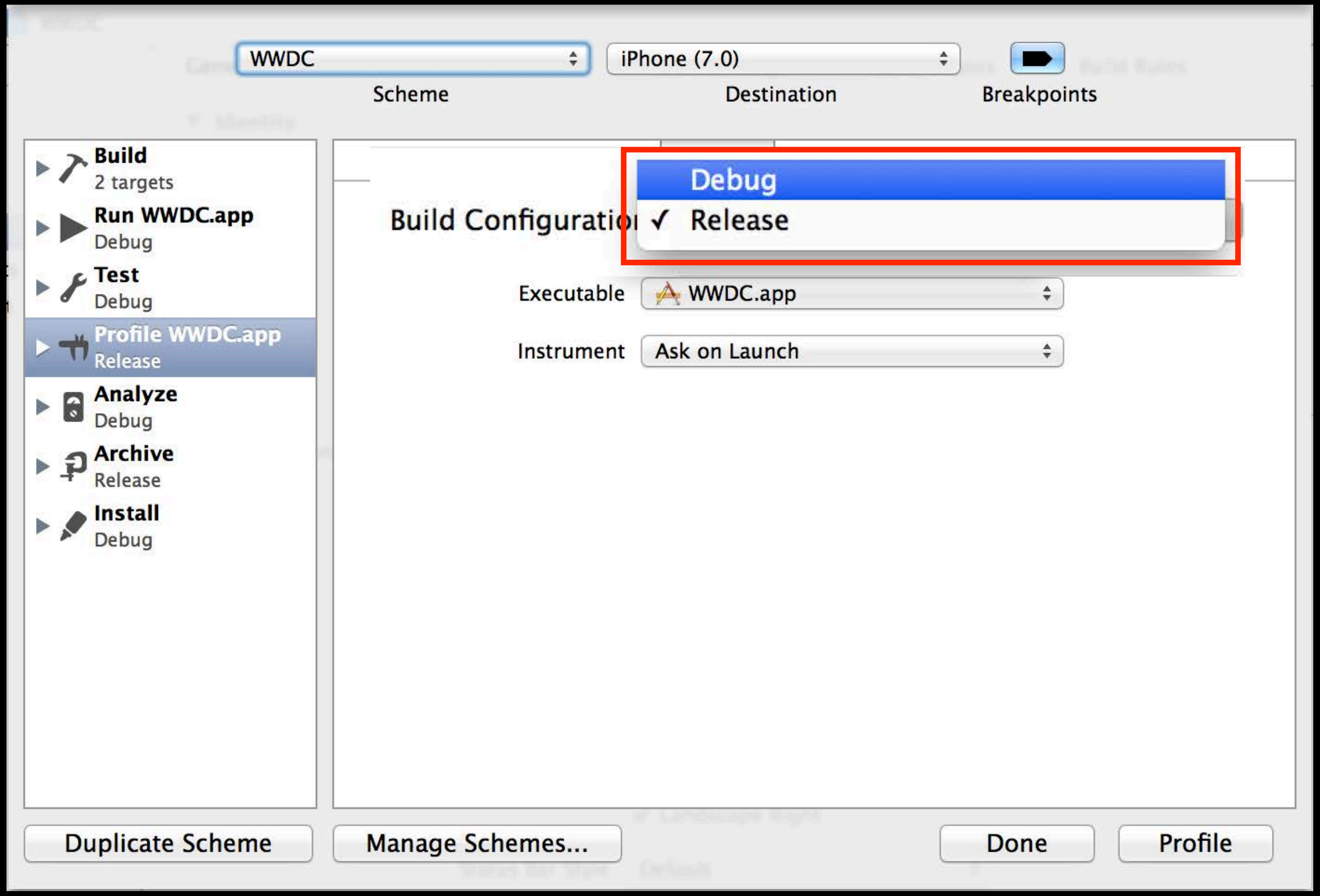
- “Profile” action defaults to Release configuration
    - Release great for Time Profiling
    - Debug useful for memory tools

Retain/Release Pairing

Profiling "Debug" configuration

- "Profile" action defaults to Release configuration
  - Release great for Time Profiling
  - Debug useful for memory tools





Build Configuration: Debug / ✓ Release

Executable: WWDC.app

Instrument: Ask on Launch

Common Objective-C Issues

With great keywords comes great responsibility

- ^block captures and retain cycles
- __weak variables
- __unsafe_unretained
- @autoreleasepool and __autoreleasing
- Working with C-APIs and __bridge casts

Common Objective-C Issues

^block captures and retain cycles

- ^blocks capture referenced objects strongly by default
- Instance variables implicitly reference ‘self’

Common Objective-C Issues

^block captures and retain cycles

- ^blocks capture referenced objects strongly by default
- Instance variables implicitly reference ‘self’

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

Common Objective-C Issues

^block captures and retain cycles

- ^blocks capture referenced objects strongly by default
- Instance variables implicitly reference 'self'

'_obsToken' retained by 'self'

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

Common Objective-C Issues

^block captures and retain cycles

- ^blocks capture referenced objects strongly by default
- Instance variables implicitly reference 'self'

'_obsToken' retained by 'self'

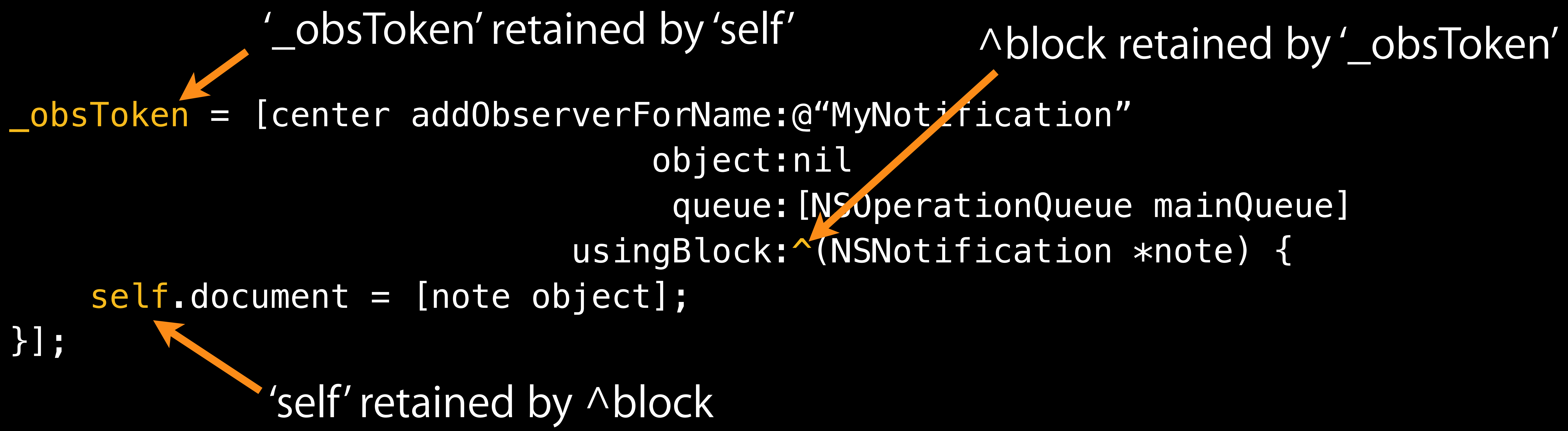
^block retained by '_obsToken'

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

Common Objective-C Issues

^block captures and retain cycles

- ^blocks capture referenced objects strongly by default
- Instance variables implicitly reference ‘self’



‘_obsToken’ retained by ‘self’

^block retained by ‘_obsToken’

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

‘self’ retained by ^block

Common Objective-C Issues

^block captures and retain cycles

- ^blocks capture referenced objects strongly by default
- Instance variables implicitly reference ‘self’

‘_obsToken’ retained by ‘self’

^block retained by ‘_obsToken’

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

‘self’ retained by ^block

✗ Retain Cycle

Common Objective-C Issues

^block captures and retain cycles

- ^blocks capture referenced objects strongly by default
- Instance variables implicitly reference ‘self’
- Use __weak keyword to break cycles
    - When non-ARC, use __block to indicate “don’t retain”

Common Objective-C Issues

^block captures and retain cycles

- ^blocks capture referenced objects strongly by default
- Instance variables implicitly reference ‘self’
- Use __weak keyword to break cycles
  - When non-ARC, use __block to indicate “don’t retain”

✓

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

Common Objective-C Issues

^block captures and retain cycles

- ^blocks capture referenced objects strongly by default
- Instance variables implicitly reference 'self'
- Use __weak keyword to break cycles
  - When non-ARC, use __block to indicate "don't retain"

✓

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

Common Objective-C Issues

^block captures and retain cycles

- ^blocks capture referenced objects strongly by default
- Instance variables implicitly reference ‘self’
- Use __weak keyword to break cycles
  - When non-ARC, use __block to indicate “don’t retain”
- Look out for persisting relationships
  - Registrations (e.g. NSNotifications, error callbacks)
  - Recurrences (e.g. timers, handlers)
  - One-time executions (dispatch_async, dispatch_after) are fine

__weak variables

Weak on demand

__weak variables

Weak on demand

- Every use of __weak validates reference
    - nil is always a possible result

__weak variables

Weak on demand

- Every use of __weak validates reference
  - nil is always a possible result
- Avoid consecutive uses

__weak variables

Weak on demand

- Every use of __weak validates reference
  - nil is always a possible result
- Avoid consecutive uses
- Never do –> dereference

__weak variables

Weak on demand

- Every use of __weak validates reference
  - nil is always a possible result
- Avoid consecutive uses
- Never do –> dereference

```
if (weakObject) {
    [weakObject->delegate customerNameChanged:name]
}
```

✗

__weak variables

Weak on demand

- Every use of __weak validates reference
  - nil is always a possible result
- Avoid consecutive uses
- Never do –> dereference

```
if (weakObject) {
    [weakObject->delegate customerNameChanged:name]
}
```

✗

__weak variables

Weak on demand

- Every use of __weak validates reference
    - nil is always a possible result
- Avoid consecutive uses
- Never do –> dereference

```
if (weakObject) {
    [weakObject->delegate customerNameChanged:name]
}
```

✗

```
[weakObject.delegate customerNameChanged:name]
                                      or
id strongObject = weakObject;
if (strongObject) {
    [strongObject->delegate customerNameChanged:name]
}
```

✓

__weak variables

Weak on demand

- Every use of __weak validates reference
  - nil is always a possible result
- Avoid consecutive uses
- Never do –> dereference

```
if (weakObject) {
    [weakObject->delegate customerNameChanged:name]
}
```

✗

```
[weakObject.delegate customerNameChanged:name]
```

or

```
id strongObject = weakObject;
if (strongObject) {
    [strongObject->delegate customerNameChanged:name]
}
```

✓

__weak variables

Weak on demand

- Every use of __weak validates reference
  - nil is always a possible result
- Avoid consecutive uses
- Never do –> dereference

```
if (weakObject) {
    [weakObject->delegate customerNameChanged:name]
}
```

✗

```
[weakObject.delegate customerNameChanged:name]
```

or

```
id strongObject = weakObject;
if (strongObject) {
    [strongObject->delegate customerNameChanged:name]
}
```

✓

__weak variables

Weak on demand

- Every use of __weak validates reference
  - nil is always a possible result
- Avoid consecutive uses
- Never do –> dereference

```
if (weakObject) {
    [weakObject->delegate customerNameChanged:name]
}
```

✗

```
[weakObject.delegate customerNameChanged:name]
```

or

```
id strongObject = weakObject;
if (strongObject) {
    [strongObject->delegate customerNameChanged:name]
}
```

✓

__weak variables

Weak on demand

- Every use of __weak validates reference
  - nil is always a possible result
- Avoid consecutive uses
- Never do –> dereference
- Do not over-use __weak
  - __weak variables are not free

__unsafe_unretained

Risk vs. Reward

__unsafe_unretained

Risk vs. Reward

- No ARC-managed -retain/-release

__unsafe_unretained

Risk vs. Reward

- No ARC-managed -retain/-release
- @property (assign) id => __unsafe_unretained id

__unsafe_unretained

Risk vs. Reward

- No ARC-managed -retain/-release
- @property (assign) id => __unsafe_unretained id
- Easily can lead to crashes — dangling references

__unsafe_unretained

Risk vs. Reward

- No ARC-managed -retain/-release
- @property (assign) id => __unsafe_unretained id
- Easily can lead to crashes — dangling references
- Unretained framework references

__unsafe_unretained

Risk vs. Reward

- No ARC-managed -retain/-release
- @property (assign) id => __unsafe_unretained id
- Easily can lead to crashes — dangling references
- Unretained framework references
- Last resort keyword

__autoreleasing

ARC and out-parameters

- Object sent -retain/-autorelease upon assignment

__autoreleasing

ARC and out-parameters

- Object sent -retain/-autorelease upon assignment
- Out-parameters are __autoreleasing by default (e.g. NSError **)

__autoreleasing

ARC and out-parameters

- Object sent -retain/-autorelease upon assignment
- Out-parameters are __autoreleasing by default (e.g. NSError **)
- If @autoreleasepool wraps assignment, crashes happen

__autoreleasing

ARC and out-parameters

- Object sent -retain/-autorelease upon assignment
- Out-parameters are __autoreleasing by default (e.g. NSError **)
- If @autoreleasepool wraps assignment, crashes happen

```
- (BOOL)startWithConfigurationURL:(NSURL*)url error:(NSError**)outError {
    @autoreleasepool {
        // < get response from url >
        NSDictionary *parsed = [NSJSONSerialization JSONObjectWithData:response
                                                               options:0 error:outError];
        if (parsed) {
            // < use dictionary >
            return YES;
        } else {
            return NO;
        }
    }
}
```

__autoreleasing

ARC and out-parameters

- Object sent -retain/-autorelease upon assignment
- Out-parameters are __autoreleasing by default (e.g. NSError **)
- If @autoreleasepool wraps assignment, crashes happen

```
- (BOOL)startWithConfigurationURL:(NSURL*)url error:(NSError**)outError {
    @autoreleasepool {
        // < get response from url >
        NSDictionary *parsed = [NSJSONSerialization JSONObjectWithData:response
                                                               options:0 error:outError];

        if (parsed) {
            // < use dictionary >
            return YES;
        } else {
            return NO;
        }
    }
}
```

Assignment to __autoreleasing ‘outError’

__autoreleasing

ARC and out-parameters

- Object sent -retain/-autorelease upon assignment
- Out-parameters are __autoreleasing by default (e.g. NSError **)
- If @autoreleasepool wraps assignment, crashes happen

```
- (BOOL)startWithConfigurationURL:(NSURL*)url error:(NSError**)outError {
    @autoreleasepool {
        // < get response from url >
        NSDictionary *parsed = [NSJSONSerialization JSONObjectWithData:response
                                                               options:0 error:outError];
        if (parsed) {
            // < use dictionary >
            return YES;
        } else {
            return NO;
        }
    }
}
```

Assignment to __autoreleasing ‘outError’

Leaving @autoreleasepool {} scope triggers “delayed” -release

__autoreleasing

ARC and out-parameters

- Object sent -retain/-autorelease upon assignment
- Out-parameters are __autoreleasing by default (e.g. NSError **)
- If @autoreleasepool wraps assignment, crashes happen

```
- (BOOL)startWithConfigurationURL:(NSURL*)url error:(NSError**)outError {
    @autoreleasepool {
        // < get response from url >
        NSDictionary *parsed = [NSJSONSerialization JSONObjectWithData:response
                                                     options:0 error:outError];
        if (parsed) {
            // < use dictionary >
            return YES;
        } else {
            return NO;
        }
    }
}
```

Assignment to __autoreleasing ‘outError’

✗ Returns deallocated NSError object to caller

Leaving @autoreleasepool {} scope triggers “delayed” -release

__autoreleasing

ARC and out-parameters

- Object sent -retain/-autorelease upon assignment
- Out-parameters are __autoreleasing by default (e.g. NSError **)
- If @autoreleasepool wraps assignment, crashes happen

```
- (BOOL)startWithConfigurationURL:(NSURL*)url error:(NSError**)outError {
    NSError *localError = nil;
    BOOL wasSuccessful = YES;
    @autoreleasepool {
        // < get response from url >
        NSDictionary *parsed = [NSJSONSerialization JSONObjectWithData:response
                                                               options:0 error:&localError];
        if (parsed) {
            // < use dictionary >
        } else {
            wasSuccessful = NO;
        }
    }
    if (!wasSuccessful && outError) *outError = localError;
    return wasSuccessful;
}
```

__autoreleasing

ARC and out-parameters

- Object sent -retain/-autorelease upon assignment
- Out-parameters are __autoreleasing by default (e.g. NSError **)
- If @autoreleasepool wraps assignment, crashes happen

```
- (BOOL)startWithConfigurationURL:(NSURL*)url error:(NSError**)outError {
    NSError *localError = nil;
    BOOL wasSuccessful = YES;
    @autoreleasepool {
        // < get response from url >
        NSDictionary *parsed = [NSJSONSerialization JSONObjectWithData:response
                                                               options:0 error:&localError];
        if (parsed) {
            // < use dictionary >
        } else {
            wasSuccessful = NO;
        }
    }
    if (!wasSuccessful && outError) *outError = localError;
    return wasSuccessful;
}
```

__autoreleasing

ARC and out-parameters

- Object sent -retain/-autorelease upon assignment
- Out-parameters are __autoreleasing by default (e.g. NSError **)
- If @autoreleasepool wraps assignment, crashes happen

```
- (BOOL)startWithConfigurationURL:(NSURL*)url error:(NSError**)outError {
    NSError *localError = nil;
    BOOL wasSuccessful = YES;
    @autoreleasepool {
        // < get response from url >
        NSDictionary *parsed = [NSJSONSerialization JSONObjectWithData:response
                                                                 options:0 error:&localError];
        if (parsed) {
            // < use dictionary >
        } else {
            wasSuccessful = NO;
        }
    }
    if (!wasSuccessful && outError) *outError = localError;
    return wasSuccessful;
}
```

__autoreleasing

ARC and out-parameters

- Object sent -retain/-autorelease upon assignment
- Out-parameters are __autoreleasing by default (e.g. NSError **)
- If @autoreleasepool wraps assignment, crashes happen

```
- (BOOL)startWithConfigurationURL:(NSURL*)url error:(NSError**)outError {
    NSError *localError = nil;
    BOOL wasSuccessful = YES;
    @autoreleasepool {
        // < get response from url >
        NSDictionary *parsed = [NSJSONSerialization JSONObjectWithData:response
                                                               options:0 error:&localError];
        if (parsed) {
            // < use dictionary >
        } else {
            wasSuccessful = NO;
        }
    }
    if (!wasSuccessful && outError) *outError = localError;
    return wasSuccessful;
}
```

__autoreleasing

ARC and out-parameters

- Object sent -retain/-autorelease upon assignment
- Out-parameters are __autoreleasing by default (e.g. NSError **)
- If @autoreleasepool wraps assignment, crashes happen

```
- (BOOL)startWithConfigurationURL:(NSURL*)url error:(NSError**)outError {
    NSError *localError = nil;
    BOOL wasSuccessful = YES;
    @autoreleasepool {
        // < get response from url >
        NSDictionary *parsed = [NSJSONSerialization JSONObjectWithData:response
                                                       options:0 error:&localError];
        if (parsed) {
            // < use dictionary >
        } else {
            wasSuccessful = NO
        }
    }
    if (!wasSuccessful && outError) *outError = localError;
    return wasSuccessful;
}
```

✓ __autoreleasing assignment outside @autoreleasepool {}

__bridge casts

Working with C-based APIs

- ARC manages at Objective-C level

__bridge casts

Working with C-based APIs

- ARC manages at Objective-C level
- C-based APIs: CoreFoundation, CoreGraphics, void* context, …

__bridge casts

Working with C-based APIs

- ARC manages at Objective-C level
- C-based APIs: CoreFoundation, CoreGraphics, void* context, …
- Three conversion primitives:
  - **__bridge T** : just type casting
  - **__bridge_transfer T** / CFBridgingRelease() : also issues a -release
  - **__bridge_retained T** / CFBridgingRetain() : also issues a -retain

__bridge casts

Working with C-based APIs

- ARC manages at Objective-C level
- C-based APIs: CoreFoundation, CoreGraphics, void* context, …
- Three conversion primitives:
  - **__bridge T** : just type casting
  - **__bridge_transfer T** / CFBridgingRelease() : also issues a -release
  - **__bridge_retained T** / CFBridgingRetain() : also issues a -retain
- Incorrect bridging leads to leaks/crashes

__bridge casts

Using them correctly

1. CF +1 → ARC-managed ‘id’ : __bridge_transfer T
2. CF +0 → ARC-managed ‘id’ : __bridge T
3. ARC-managed ‘id’ → CF +0 : __bridge T
4. ARC-managed ‘id’ → CF +1: __bridge_retained T

__bridge casts

Using them correctly

1. CF +1 → ARC-managed 'id' : __bridge_transfer T
2. CF +0 → ARC-managed 'id' : __bridge T
3. ARC-managed 'id' → CF +0 : __bridge T
   - Effectively creates an __unsafe_unretained CF reference!
4. ARC-managed 'id' → CF +1: __bridge_retained T

__bridge casts

Using them correctly

1. CF +1 → ARC-managed ‘id’ : __bridge_transfer T
2. CF +0 → ARC-managed ‘id’ : __bridge T
3. ARC-managed ‘id’ → CF +0 : __bridge T
    - Effectively creates an __unsafe_unretained CF reference!

```
CFStringRef stringRef = NULL;
...
NSString *logInfo = [[NSString alloc] initWithFormat:...];
stringRef = (__bridge CFStringRef)logInfo;
...
CFURLRef url = CFURLCreateWithString(NULL, stringRef, baseURL);
```

4. ARC-managed ‘id’ → CF +1: __bridge_retained T

__bridge casts

Using them correctly

1. CF +1 → ARC-managed 'id' : __bridge_transfer T
2. CF +0 → ARC-managed 'id' : __bridge T
3. ARC-managed 'id' → CF +0 : __bridge T
   - Effectively creates an __unsafe_unretained CF reference!

```
CFStringRef stringRef = NULL;
...
NSString *logInfo = [[NSString alloc] initWithFormat:...];
stringRef = (__bridge CFStringRef)logInfo;
...
CFURLRef url = CFURLCreateWithString(NULL, stringRef, baseURL);
```

4. ARC-managed 'id' → CF +1: __bridge_retained T

__bridge casts

Using them correctly

1. CF +1 → ARC-managed ‘id’ : __bridge_transfer T
2. CF +0 → ARC-managed ‘id’ : __bridge T
3. ARC-managed ‘id’ → CF +0 : __bridge T
   - Effectively creates an __unsafe_unretained CF reference!

```
CFStringRef stringRef = NULL;
...
NSString *logInfo = [[NSString alloc] initWithFormat:...];
stringRef = (__bridge CFStringRef)logInfo;
...
CFURLRef url = CFURLCreateWithString(NULL, stringRef, baseURL);
```

‘logInfo’ leaves scope, released

4. ARC-managed ‘id’ → CF +1: __bridge_retained T

__bridge casts

Using them correctly

1. CF +1 → ARC-managed ‘id’ : __bridge_transfer T
2. CF +0 → ARC-managed ‘id’ : __bridge T
3. ARC-managed ‘id’ → CF +0 : __bridge T
   - Effectively creates an __unsafe_unretained CF reference!

```
CFStringRef stringRef = NULL;
...
NSString *logInfo = [[NSString alloc] initWithFormat:...];
stringRef = (__bridge CFStringRef)logInfo;
...
CFURLRef url = CFURLCreateWithString(NULL, stringRef, baseURL);
```

‘logInfo’ leaves scope, released

‘stringRef’ not valid

4. ARC-managed ‘id’ → CF +1: __bridge_retained T

__bridge casts

Using them correctly

1. CF +1 → ARC-managed ‘id’ : __bridge_transfer T
2. CF +0 → ARC-managed ‘id’ : __bridge T
3. ARC-managed ‘id’ → CF +0 : __bridge T
   - Effectively creates an __unsafe_unretained CF reference!
4. ARC-managed ‘id’ → CF +1: __bridge_retained T

```
CFStringRef stringRef = NULL;
...
NSString *logInfo = [[NSString alloc] initWithFormat:...];
stringRef = (__bridge_retained CFStringRef)logInfo;
...
CFURLRef url = CFURLCreateWithString(NULL, stringRef, baseURL);
CFRelease(stringRef)
```

Being a Good Citizen

Testing and tips

Memory Testing

Real-world user scenarios

Memory Testing

Real-world user scenarios

- Test on constrained devices

Memory Testing

Real-world user scenarios

- Test on constrained devices
- First install/first launch

Memory Testing

Real-world user scenarios

- Test on constrained devices
- First install/first launch
- Large dataset

Memory Testing

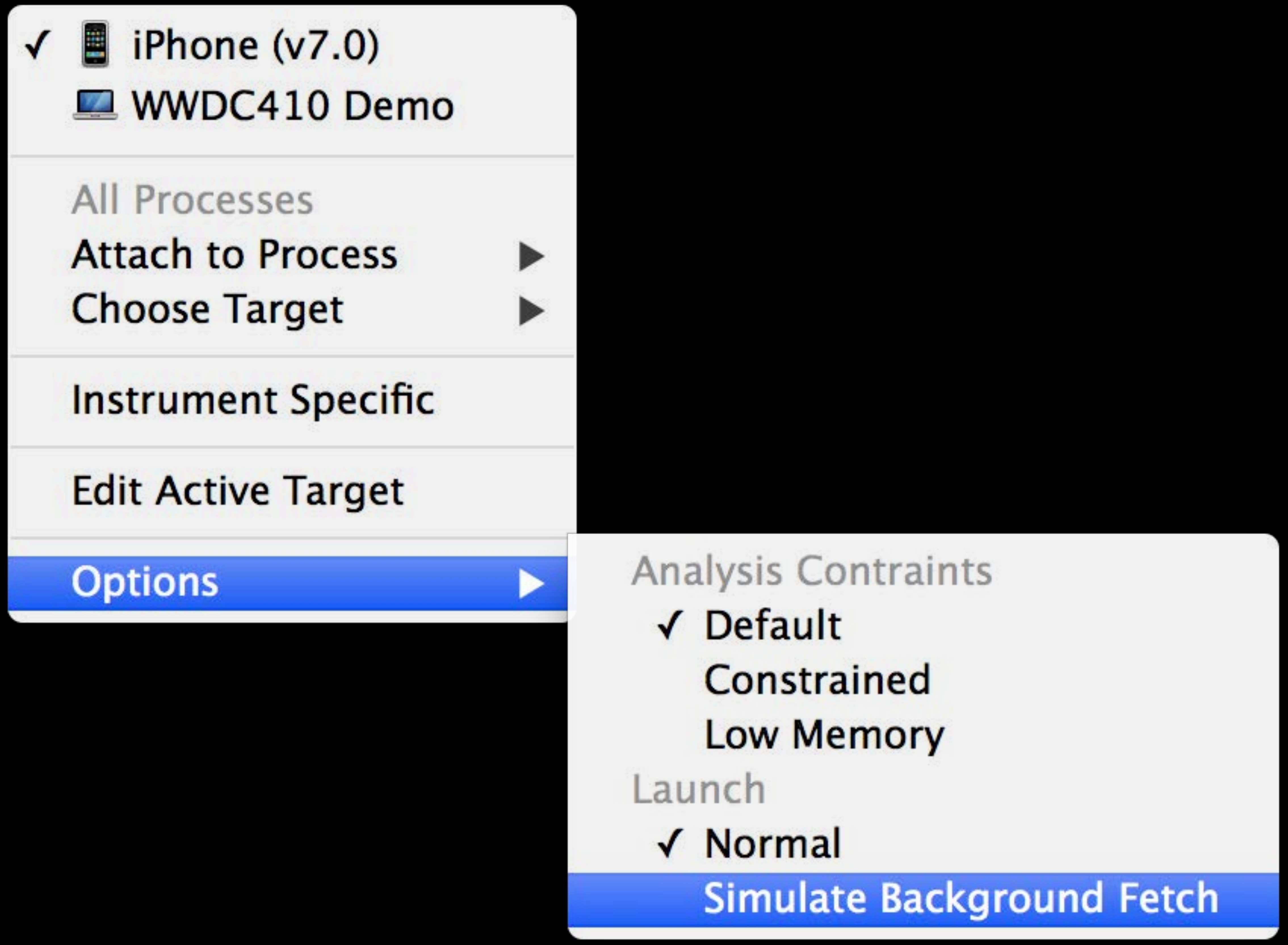
Real-world user scenarios

- Test on constrained devices
- First install/first launch
- Large dataset
- Background launch

Memory Testing

Real-world user scenarios

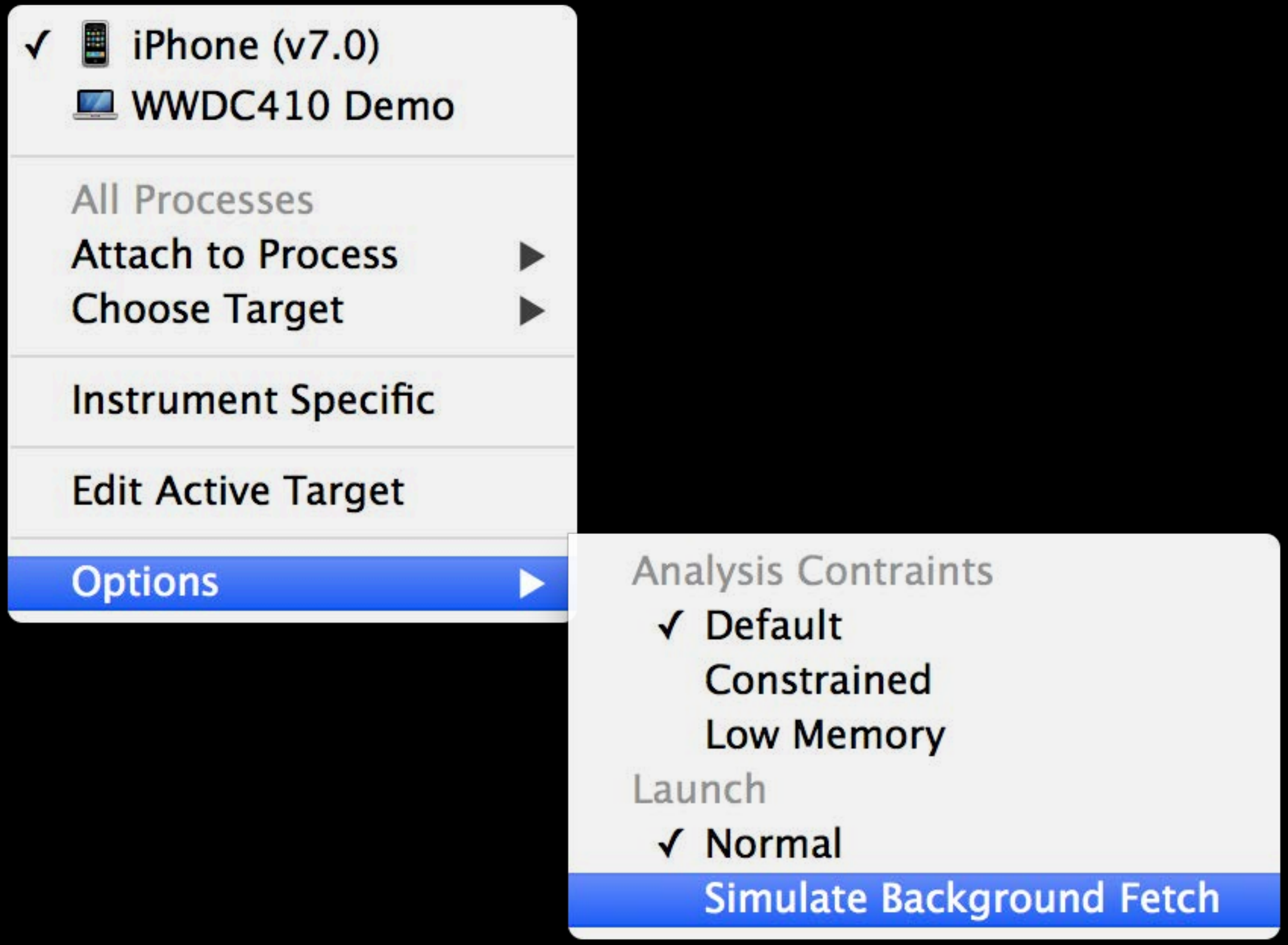
- Test on constrained devices
- First install/first launch
- Large dataset
- Background launch



Memory Testing

Real-world user scenarios

- Test on constrained devices
- First install/first launch
- Large dataset
- Background launch
  - Especially for leaked/abandoned memory



System Memory Pressure

Where there are not enough free pages

- Pages must be evicted
- Clean and purgeable pages can be thrown away
- Dirty pages

System Memory Pressure

Where there are not enough free pages

- Pages must be evicted
- Clean and purgeable pages can be thrown away
- Dirty pages
  - On OSX, compressed or saved to disk (expensive)

Building Efficient OS X Apps

Nob Hill
Tuesday 4:30PM

System Memory Pressure

Where there are not enough free pages

- Pages must be evicted
- Clean and purgeable pages can be thrown away
- Dirty pages
  - On OSX, compressed or saved to disk (expensive)

Building Efficient OS X Apps — Nob Hill, Tuesday 4:30PM

  - On iOS, memory warnings issued and processes terminated

iOS: Memory Warnings

… and avoiding termination

iOS: Memory Warnings

… and avoiding termination

- Do not be the biggest
  - Dirty memory is what counts — VM Tracker

iOS: Memory Warnings

… and avoiding termination

- Do not be the biggest
    - Dirty memory is what counts — VM Tracker
- Make sure your application gets a chance to respond
    - Avoid large, rapid allocations
    - Notifications arrive on main thread

iOS: Memory Warnings

… and avoiding termination

- Do not be the biggest
  - Dirty memory is what counts — VM Tracker
- Make sure your application gets a chance to respond
  - Avoid large, rapid allocations
  - Notifications arrive on main thread

```
UIApplicationDidReceiveMemoryWarningNotification
–[id <UIApplicationDelegate> -applicationDidReceiveMemoryWarning:]
–[UIViewController didReceiveMemoryWarning]
```

iOS: Memory Warnings

… and avoiding termination

- Do not be the biggest
  - Dirty memory is what counts — VM Tracker
- Make sure your application gets a chance to respond
  - Avoid large, rapid allocations
  - Notifications arrive on main thread

```
UIApplicationDidReceiveMemoryWarningNotification
–[id <UIApplicationDelegate> –applicationDidReceiveMemoryWarning:]
–[UIViewController didReceiveMemoryWarning]
```

- Consider freeing up memory before entering background

iOS: Memory Warnings

… and avoiding termination

- Do not be the biggest
  - Dirty memory is what counts — VM Tracker
- Make sure your application gets a chance to respond
  - Avoid large, rapid allocations
  - Notifications arrive on main thread

```
UIApplicationDidReceiveMemoryWarningNotification
–[id <UIApplicationDelegate> –applicationDidReceiveMemoryWarning:]
–[UIViewController didReceiveMemoryWarning]
```

- Consider freeing up memory before entering background

```
–[id <UIApplicationDelegate> –applicationDidEnterBackground:]
```

Summary

- Be proactive: monitor, test, investigate
- Avoid memory spikes
- Don’t allow unbounded heap/VM growth
- Use language tools effectively: __weak, @autoreleasepool, etc.

More Information

Dave DeLong
Developer Tools Evangelist
delong@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

Advances in Objective-C	Mission Tuesday 4:30PM	🟧
Building Efficient OS X Apps	Nob Hill Tuesday 4:30PM	🟩
Core Data Performance Optimization and Debugging	Nob Hill Wednesday 2:00PM	🟦
Energy Best Practices	Marina Thursday 10:15AM	🟩
Designing Code for Performance	Nob Hill Friday 9:00AM	🟦

Labs

Objective-C and LLVM Lab	Tools Lab C Thursday 2:00PM
Instruments and Performance Lab	Tools Lab B Thursday 3:15PM
LLDB and Instruments Lab	Tools Lab C Friday 10:15AM

WWDC2013