Introducing Automatic Reference Counting

Session 323

Chris Lattner
ARChitect
Low Level Tools & Objective C

Automatic Reference Counting

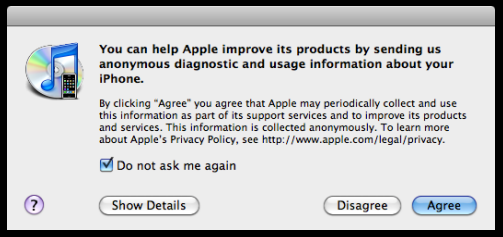
Automatic Memory Management for Objective-C

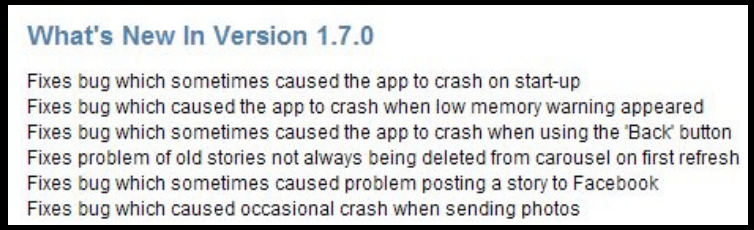
- Simplified retain/release programming:
  - Easier to learn
  - More productive
  - Easier to maintain
  - Safer and more stable

You should move your Objective-C code to ARC

Apps Crash

"I am getting bad reviews and reports from users (from what I can gather less than 1% of users) that my app is crashing on startup...





About a day after releasing the game, I got a mail saying basically "well, you got my money, but I can't play your app because it crashes."...

We pulled <XXX> from the App Store due to crashes

We had issues already since the very first launch of the iPhone app. Unfortunately these issues were due to bad memory management decisions right from the start, something that we could not easily fix without completely reengineering the app.

#1 Cause of App Rejection

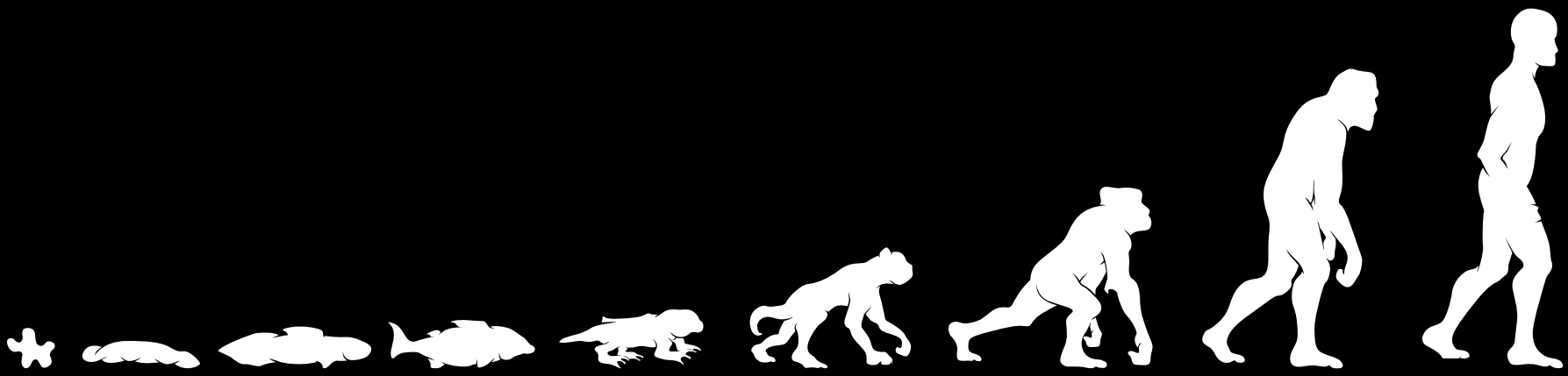
Memory Management is Harder Than It Looks…

- Instruments
  - Allocations, Leaks, Zombies
- Xcode Static Analyzer
- Heap
- ObjectAlloc
- vmmap
- MallocScribble
- debugger watchpoints
- …and lots more

Evolution of Objective-C

Simpler and safer through automation

- Object Oriented C
- Retain and Release
- Properties
- Blocks
- ARC



Programming with Retain/Release

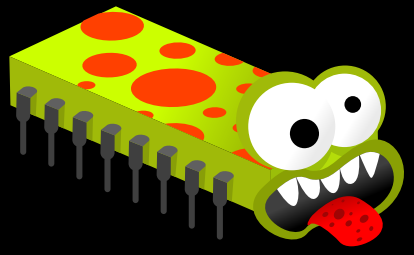
Swimming in details

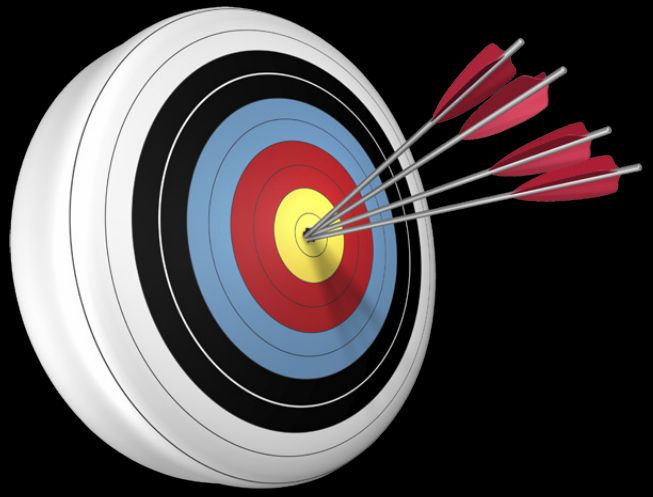
- Cocoa naming conventions
- Autorelease pools
- Block_copy
- Autoreleasing or not?
  - [NSString stringWith…
  - [[NSString alloc] initWith…]

Cognitive Overload!

Some Perfection Required

- Need to write perfect code:
  - Never forget a release on an error case
  - Never dereference a dangling pointer
- Must be great at debugging subtle crashes

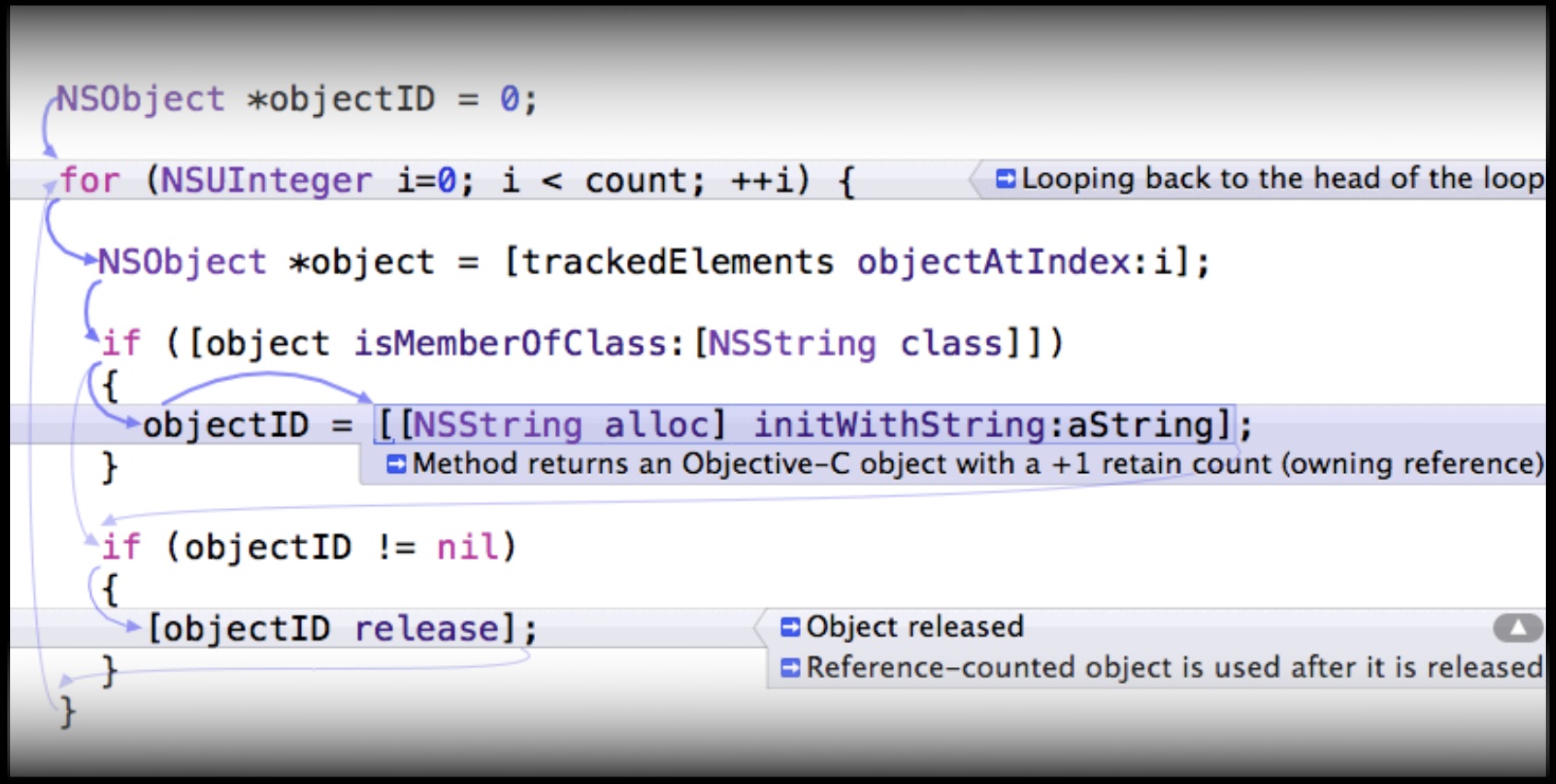




Can’t we make this better?

Xcode Static Analyzer

Compiler analysis to find retain/release bugs



Lessons Learned from the Analyzer

- Dozens or hundreds of leaks are common
  - Developers think the analyzer is buggy!
- Cocoa code has strong patterns
  - Naming conventions
  - "-init" patterns
- Common sources of false positives:
  - Core Foundation APIs
  - Unions, structs, etc

Can We Automate This?

- Huge opportunity to simplify Objective-C
  - Aim to subtract complexity
- Challenges ahead
  - Compiler must be perfect!
  - Analyzer is not fast

A hard problem, but worth solving!

Talk Roadmap

- What is ARC?
- How does it work?
- How do you switch?

Welcome to ARC!

- Automatic Object memory management
  - Compiler synthesizes retain/release calls
  - Compiler obeys and enforces library conventions
  - Full interoperability with retain/release code
- New runtime features:
  - Zeroing weak pointers
  - Advanced performance optimizations
  - Compatibility with Snow Leopard and iOS4

What ARC Is Not…

- No new runtime memory model
- No automation for malloc/free, CF, etc.
- No garbage collector
  - No heap scans
  - No whole app pauses
  - No non-deterministic releases

How Does It Work?

- Compiler inserts retain/release/autorelease for you

```
- (NSString*) fullName {
  return [[NSString alloc] initWithFormat: @"%@ %@",
                    self.firstName, self.lastName];
}

- (NSString*) fullName {
  return [NSString stringWithFormat: @"%@ %@",
                self.firstName, self.lastName];
}
```

How Do I Think About It?

- Pointers keep objects alive
  - Objects die when no more references remain
- Pointers imply "ownership"
  - Think about your object graph!
- Stop thinking about retain/release/autorelease calls

What Does This Do to Your Code?

Mostly, removes a ton of code

```
- (void)dealloc {
    [identifier release];
    [title release];
    [HTML release];
    [category release];
    [super dealloc];
}
```

What Does This Do to Your Code?

Additions makes it easier to read and understand

```
@interface MyClass : NSObject {
  NSArray *A;  // unretained!
...
```

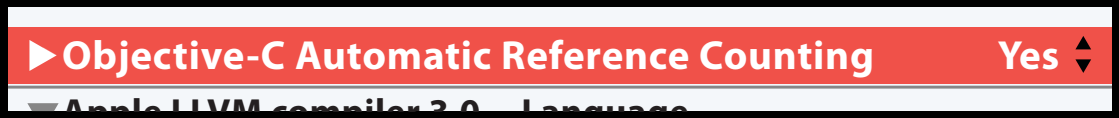
Manual Reference Counting

```
@interface MyClass : NSObject {
  __weak NSArray *A;
...
```

Automatic Reference Counting

How Is It Compatible?

- New (opt-in) compiler flag: `-fobjc-arc`
  - No behavior change for existing code



- New projects default to ARC
  - Existing code must migrate to ARC (or stay Manual)

How Do We Make ARC Reliable?

- Problem:
  - Cannot rely on heuristics like the static analyzer
- Solution:
  - Formalize and automate best practice

Rule #1/4: No Access to Memory Methods

- Memory mangement is part of the language
  - Cannot call retain/release/autorelease…
  - Cannot implement these methods
- Solution
  - The compiler takes care of it
  - NSObject performance optimizations
  - Better patterns for singletons

```
while ([x retainCount] != 0)
  [x release];
```

Broken code, Anti-pattern

Rule #2/4: No Object Pointers in C Structs

- Compiler must know when references come and go
  - Pointers must be zero initialized
  - Release when reference goes away

- Solution: Just use objects
  - Better tools support
  - Best practice for Objective-C

```
struct Pair {
  NSString *Name;  // retained!
  int Value;
};
```

```
Pair *P = malloc(...);
...
...
free(P);  // Must drop references!
```

Rule #3/4: No Casual Casting id ↔ void*

- Compiler must know whether void* is retained
- New CF conversion APIs
- Three keywords to disambiguate casts

```
CFStringRef W = (__bridge          CFStringRef)A;
NSString *X   = (__bridge          NSString *)B;
CFStringRef Y = (__bridge_retain   CFStringRef)C;
NSString *Z   = (__bridge_transfer NSString *)D;
```

Objective-C Advancements In-Depth | Mission Friday 11:30AM

Rule #4/4: No NSAutoreleasePool

- Compiler must reason about autoreleased pointers
- NSAutoreleasePool not a real object—cannot be retained

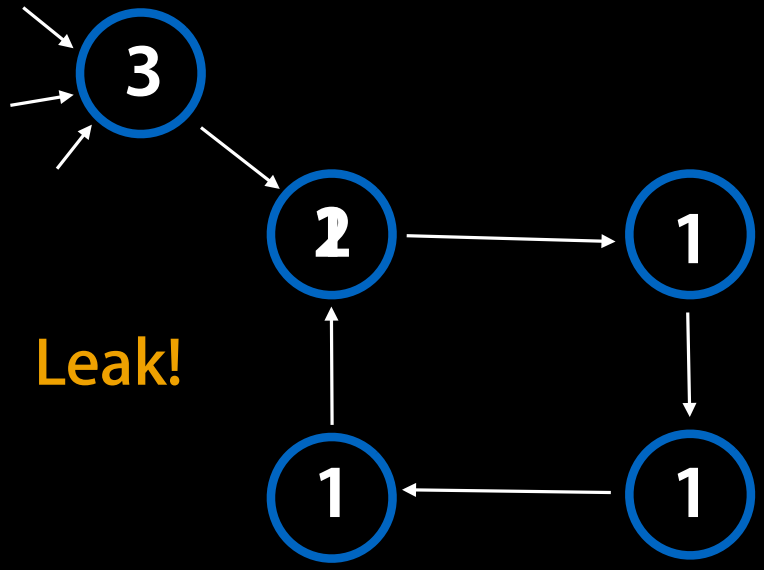
```
int main(int argc, char *argv[]) {
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}
```

```
int main(int argc, char *argv[]) {
  @autoreleasepool {
    return UIApplicationMain(argc, argv, nil, nil);
  }
}
```

Works in all modes!

What About Object Graph Cycles?

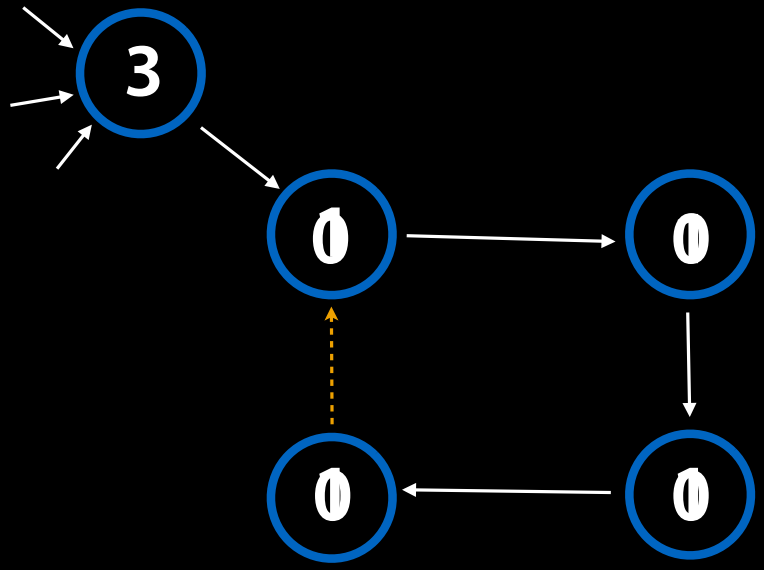
- Just like in MRC, cycles cause leaks in ARC
- Standard tricks still work:
  - Explicitly nil instance variables
  - Use potentially dangling pointers: "assign properties"



Zeroing Weak References

- Safe, non-retaining reference
  - Drops to nil automatically
- Now supported in ARC:

```
@property (weak) NSView *V;
id __weak P;
```



What About Blocks in ARC?

They just work!

```
dispatch_block_t getfoo(int i) {
  return [[^{
    print(i);
  } copy] autorelease];
}
```

Manual Reference Counting

```
dispatch_block_t getfoo(int i) {
  return ^{
    print(i);
  };
}
```

Automatic Reference Counting

Blocks and Grand Central Dispatch in Practice	Pacific Heights Wednesday 10:15AM
Objective-C Advancements In-Depth	Mission Friday 11:30AM

Does ARC Cost Me Performance?

- No GC overhead:
  - No delayed deallocations
  - No app pauses, no non-determinism
- Objective-C Performance Improvements
  - 2.5x faster NSObject retain/release
  - 6x faster @autoreleasepool
  - 33% faster objc_msgSend
- "Reclaim" from autorelease pools
  - Lower memory pressure
  - 20x faster retain/autorelease returns

This Is ARC!

- Compiler synthesized retain/release
- Zeroing weak pointers
- New language rules
- A simpler place to be

How ARC Works

John McCall
ARChmage
Compiler Team

Memory Management Is Hard

- Lots of rules and conventions
- High hurdles for new developers
- Constant attention for existing developers
- Requires perfection

Let's Write a Stack

```
@implementation Stack { NSMutableArray *_array; }
- (id) init {
  if (self = [super init])
    _array = [NSMutableArray array];
  return self;
}
- (void) push: (id) x {
  [_array addObject: x];
}
- (id) pop {
  id x = [_array lastObject];
  [_array removeLastObject];
  return x;
}
@end
```

Let's Write a Stack

```
@implementation Stack { NSMutableArray *_array; }
- (id) init {
  if (self = [super init])
    _array = [[NSMutableArray array] retain];
  return self;
}
- (void) push: (id) x {
  [_array addObject: x];
}
- (id) pop {
  id x = [_array lastObject];
  [_array removeLastObject];
  return x;
}
@end
```

This array is autoreleased!
System will deallocate it
out from under us

Let's Write a Stack

```
@implementation Stack { NSMutableArray *_array; }
- (id) init {
  if (self = [super init])
    _array = [[NSMutableArray array] retain];
  return self;
}
- (void) push: (id) x {
  [_array addObject: x];
}
- (id) pop {
  id x = [_array lastObject];
  [_array removeLastObject];
  return x;
}
- (void) dealloc { [_array release]; [super dealloc]; }
@end
```

Now it is leaked!
Need a dealloc method

Let's Write a Stack

```
@implementation Stack { NSMutableArray *_array; }
- (id) init {
  if (self = [super init])
    _array = [[NSMutableArray array] retain];
  return self;
}
- (void) push: (id) x {
  [_array addObject: x];
}
- (id) pop {
  id x = [[_array lastObject] retain];
  [_array removeLastObject];
  return x;
}
- (void) dealloc { [_array release]; [super dealloc]; }
@end
```

This might invalidate x

Works only if there are other references to it

Let's Write a Stack

```
@implementation Stack { NSMutableArray *_array; }
- (id) init {
  if (self = [super init])
    _array = [[NSMutableArray array] retain];
  return self;
}
- (void) push: (id) x {
  [_array addObject: x];
}
- (id) pop {
  id x = [[_array lastObject] retain];
  [_array removeLastObject];
  return [x autorelease];
}
- (void) dealloc { [_array release]; [super dealloc]; }
@end
```

Now we're returning a retained pointer
Violates convention; probably a leak

Let's Write a Stack

```
@implementation Stack { NSMutableArray *_array; }
- (id) init {
  if (self = [super init])
    _array = [[NSMutableArray array] retain];
  return self;
}
- (void) push: (id) x {
  [_array addObject: x];
}
- (id) pop {
  id x = [[_array lastObject] retain];
  [_array removeLastObject];
  return [x autorelease];
}
- (void) dealloc { [_array release]; [super dealloc]; }
@end
```

ARC Is Easy

- Write code naturally
- Break cycles when necessary
- Stop worrying about retains
- Write great apps

How ARC Works

- Automates what programmers do anyway
- Applies local rules
- Guarantees local correctness
- Optimizes away redundancies

Rules for Variables

- Objective-C, block pointers
- Locals, globals, parameters, instance variables, …
- Four different kinds of ownership

Strong References

```
NSString *name;
```

```
__strong NSString *name;
```

- Default for all variables
- Like a retain property

Creating a Variable

```
NSString *name;
```

```
NSString *name = nil;
```

- Initialized to nil
- Safer for ARC-generated code
- Safer for your code

Destroying a Variable

```
if (i < 10) {
  NSString *name = ...;
}
```

```
if (i < 10) {
  NSString *name = ...;
  [name release];
}
```

- Current value is implicitly released
- All variables, even ivars

Reading and Writing

```
name = newName;
```

```
[newName retain];
NSString *oldName = name;
name = newName;
[oldName release];
```

- Nothing special for reads
- For writes:
  - Retain the new value
  - Release the old value

Autoreleasing References

```
- (void) runWithError:
    (NSError **) err {
  if (!valid_)
    *err = ...;
}
```

```
- (void) runWithError:
    (__autoreleasing NSError **) err {
  if (!valid_)
    *err = [[... retain] autorelease];
}
```

- Describes out-parameters
- Only on the stack
- Not for general use

Unsafe References

```
__unsafe_unretained NSString *unsafeName = name;
```

- Like a traditional variable, or an assign property
- Not initialized
- No extra logic
- No restrictions

Weak References

```
__weak NSString *weakName = name;
char c = [weakName
           characterAtIndex: 0];
```

```
__weak NSString *weakName = nil;
objc_storeWeak(&weakName, name);
char c = [objc_readWeak(&weakName)
            characterAtIndex: 0];
objc_storeWeak(&weakName, nil);
```

- Runtime manages reads and writes
- Becomes `nil` as soon as object starts deallocation

Rules for Return Values

- Does this transfer ownership?
- Return values can be “returned retained”
- Similar concepts for other transfers into or out of ARC
- Decided by method family

Method Families

- Naming convention
- First “word” in first part of selector
- `alloc`, `copy`, `init`, `mutableCopy`, `new` transfer ownership
- Everything else does not

Normal Returns

```
- (NSString*) serial {
  return _serial;
}
```

```
- (NSString*) serial {
  NSString *returnValue
    = [_serial retain];
  return [returnValue autorelease];
}
```

- No transfer
- Retain immediately
- Autorelease after leaving all scopes

Retained Returns

```
- (NSString*) newSerial {
  return _serial;
}
```

```
- (NSString*) newSerial {
  NSString *returnValue
    = [_serial retain];
  return returnValue;
}
```

- Passes back ownership
- Like a normal return without the autorelease

Accepting a Retained Return

```
- (void) logSerial {
  NSLog(“%@\n”, [self newSerial]);
  ...
}
```

```
- (void) logSerial {
  NSString *returnValue
    = [self newSerial];
  NSLog(“%@\n”, returnValue);
  [returnValue release];
  ...
}
```

- Takes ownership
- Return value released at end of statement

Back to the Stack

```
- (id) init {
  if (self = [super init])
    _array = [NSMutableArray array];
  return self;
}
```

```
- (id) init {
  if (self = [super init]) {
    NSMutableArray *oldArray = _array;
    _array = [[NSMutableArray array]
                retain];
    [oldArray release];
  }
  return self;
}
- (void) dealloc {
  [_array release];
  [super dealloc];
}
```

Back to the Stack

```
- (id) pop {
  id x = [_array lastObject];
  [_array removeLastObject];
  return x;
}
```

```
- (id) pop {
  id x = [[_array lastObject] retain];
  [_array removeLastObject];
  return [x autorelease];
}
```

- Local rules make this work
- Unnecessary retain/release are optimized away

Natural Code Just Works

```
@implementation Stack { NSMutableArray *_array; }
- (id) init {
  if (self = [super init])
    _array = [NSMutableArray array];
  return self;
}
- (void) push: (id) x {
  [_array addObject: x];
}
- (id) pop {
  id x = [_array lastObject];
  [_array removeLastObject];
  return x;
}
@end
```

Putting It Together

- ARC follows the conventions for you
- Stop worrying about retains
- Focus on making great apps

Migrating to ARC

Patrick Beard
ARC Shark
Objective-C Runtime Team

Migrating to ARC

Overview

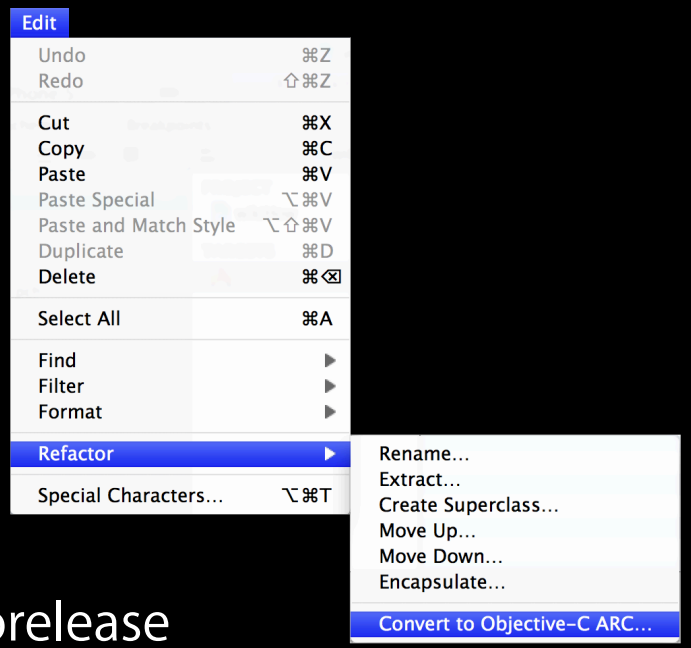
- Migration Steps
- Common Issues
- Deployment Options
- Demo

Migration Steps

- Compile your code with the LLVM compiler 3.0
- Use "Convert to Objective-C ARC" command in Xcode
- Fix issues until everything compiles
- Migration tool then modifies your code and project

"Convert to Objective-C ARC"

- Supported in Xcode 4.2



- Makes your source ARC compatible
  - Removes all calls to retain, release, autorelease
  - Replaces NSAutoreleasePool with @autoreleasepool
  - @property(assign) becomes @property(weak) for object pointers

Two Phases of ARC Migration

- Analysis
  - Migration problems presented as compile errors
  - Fix errors, run analysis again
- Conversion
  - After successful analysis, changes automatically applied
  - Turns on -fobjc-arc

Migrating to ARC

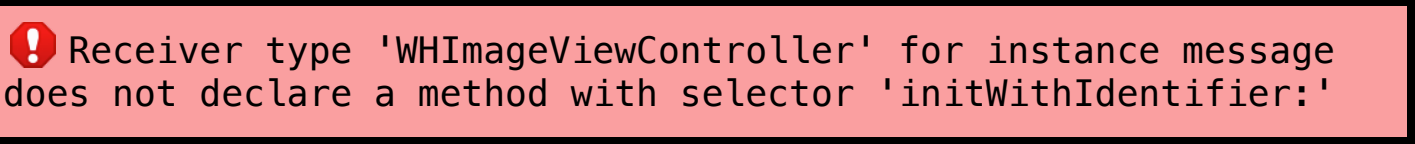
Missing method declarations

```
@interface WHImageViewController : UIViewController
@property (nonatomic, readonly) NSString *identifier;
@end

- (void)pushViewControllerWithIdentifier:(NSString *)identifier
                                animated:(BOOL)animated {
  UIViewController *viewController;
  viewController = [[WHImageViewController alloc]
                    initWithIdentifier:identifier];
  [self pushViewController:viewController animated:animated];
  [viewController release];
}
```

Migrating to ARC

Missing method declarations

```
@interface WHImageViewController : UIViewController
@property (nonatomic, readonly) NSString *identifier;
@end

- (void)pushViewControllerWithIdentifier:(NSString *)identifier
                                animated:(BOOL)animated {

  UIViewController *viewController;
  viewController = [[WHImageViewController alloc]
                    initWithIdentifier:identifier];
```




```
  [self pushViewController:viewController animated:animated];
  [viewController release];
}
```

Migrating to ARC

Unstructured Autorelease Pools

```
- (void)encodeFile {
  /* setup */
  BOOL done = NO, shouldDrain = NO;
  NSAutoreleasePool *loopPool = [NSAutoreleasePool new];
  while (!done) {
    /* part A. */
    if (shouldDrain) {
      [loopPool drain];
      loopPool = [NAutoreleasePool new];
    }
    /* part B. */
  }
  [loopPool drain];
}
```

Migrating to ARC

Unstructured Autorelease Pools

```
- (void)encodeFile {
  /* setup */
  BOOL done = NO, shouldDrain = NO;
  NSAutoreleasePool *loopPool = [NSAutoreleasePool new];

                  'NSAutoreleasePool' is unavailable: not
                  available in automatic reference counting mode

  while (!done) {
    /* part A. */
    if (shouldDrain) {
      [loopPool drain];
      loopPool = [NAutoreleasePool new];
    }
    /* part B. */
  }
  [loopPool drain];
}
```

Migrating to ARC

Block-structured @autoreleasepool

```
- (void)encodeFile {
  /* setup */
  BOOL done = NO;
  while (!done) {
    @autoreleasepool {
      /* part A. */
      /* part B. */
    }
  }
}
```

- Empty @autoreleasepool on Lion is 6x faster than SnowLeopard

Migrating to ARC

Declarations after case labels

```
- (void)decide {
  switch (currentState) {
  case INITIAL_STATE:;
    NSDate *date = [NSDate date];
    NSLog(@"started at %@", date);
    break;
  case MIDDLE_STATE:
  /* … */
  case FINAL_STATE:
    /* date is in-scope here */
    break;
  }
}
```

Migrating to ARC

Declarations after case labels

```
- (void)decide {
  switch (currentState) {
  case INITIAL_STATE:;
    NSDate *date = [NSDate date];
    NSLog(@"started at %@", date);
    break;
  case MIDDLE_STATE:
                    Switch case is in protected scope
  /* … */
  case FINAL_STATE:
    /* date is in-scope here */
    break;
  }
}
```

Migrating to ARC

Declarations after case labels require curly braces

```
- (void)decide {
  switch (currentState) {
  case INITIAL_STATE:
  {
    NSDate *date = [NSDate date];
    NSLog(@"started at %@", date);
    break;
  }
  /* … */
  case FINAL_STATE:
    /* date is in-scope here */
    break;
  }
}
```

Migrating to ARC

Singleton pattern

```
/* Shared Network Activity Indicator */
@interface ActivityIndicator : NSObject {
  int count;
}
+ (ActivityIndicator *)sharedIndicator;
- (void)show;
- (void)hide;
@end
```

Singleton Pattern

Overriding retain/release/autorelease

```
@implementation ActivityIndicator

- (id)retain { return self; }
- (oneway void)release {}
- (id)autorelease { return self; }
- (NSUInteger) retainCount { return NSUIntegerMax; }

+ (ActivityIndicator *)sharedIndicator {...}

@end
```

Singleton Pattern

Overriding retain/release/autorelease

```
@implementation ActivityIndicator

- (id)retain { return self; }
- (oneway void)release {}
- (id)autorelease { return self; }
- (NSUInteger) retainCount { return NSUIntegerMax; }

+ (ActivityIndicator *)sharedIndicator {...}

@end
```

ARC forbids implementation of ‘retain’

ARC forbids implementation of ‘release’

ARC forbids implementation of ‘autorelease’

ARC forbids implementation of ‘retainCount’

- ARC forbids these overrides

Singleton Pattern

Overriding +allocWithZone:

```
@implementation ActivityIndicator

+ (id)allocWithZone:(NSZone *)zone {
  static id sharedInstance;
  if (sharedInstance == nil)
    sharedInstance = [super allocWithZone:NULL];
  return sharedInstance;
}

+ (ActivityIndicator *)sharedIndicator {...}

@end
```

- init method will have to guard against multiple calls on the shared instance

Singleton Pattern

Use simple shared instance method

```
@implementation ActivityIndicator

+ (ActivityIndicator *)sharedIndicator {
  static ActivityIndicator *sharedIndicator;
  if (sharedIndicator == nil) sharedIndicator = [ActivityIndicator new];
  return sharedIndicator;
}

- (void)show {...}

- (void)hide {...}

@end
```

Singleton Pattern

Use dispatch_once for thread-safety

```
@implementation ActivityIndicator

+ (ActivityIndicator *)sharedIndicator {
  static ActivityIndicator *sharedIndicator;
  static dispatch_once_t done;
  dispatch_once(&done, ^{ sharedIndicator = [ActivityIndicator new]; });
  return sharedIndicator;
}

- (void)show {...}

- (void)hide {...}

@end
```

Singleton Pattern

Classes ARE singletons

```
@implementation ActivityIndicator

static NSInteger count;

+ (void)show {
  if (count++ == 0)
  [UIApplication sharedApplication].networkActivityIndicatorVisible = YES;
}

+ (void)hide {
  if (count && --count == 0)
  [UIApplication sharedApplication].networkActivityIndicatorVisible = NO;
}

@end
```

Migrating to ARC

Delegate pattern

```
@protocol PageViewDelegate;

@interface PageView : NSView {
  id <PageViewDelegate> delegate;
  /* ... */
}

@property(assign) id <PageViewDelegate> delegate;

@end


@implementation PageView

@synthesize delegate;

/* ... */

@end
```

Delegate Pattern

Assign migrates to weak

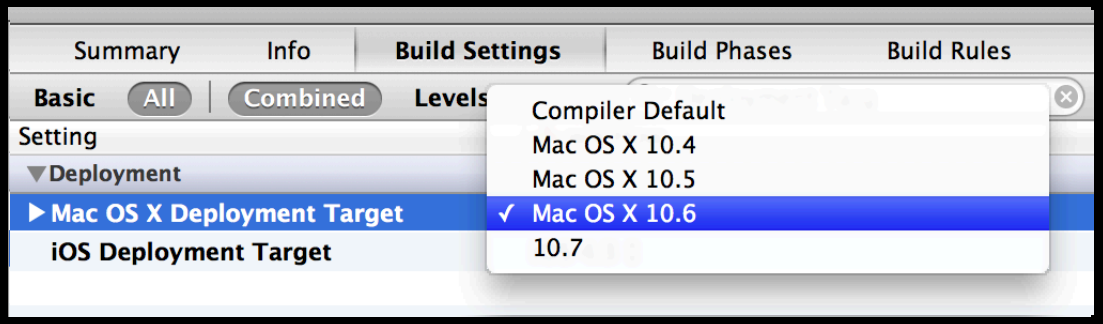
```
@protocol PageViewDelegate;
@interface PageView : NSView {
  __weak id <PageViewDelegate> delegate;
  /* ... */
}
@property(weak) id <PageViewDelegate> delegate;
@end

@implementation PageView
@synthesize delegate;
/* ... */
@end
```

ARC Deployment

ARC Deployment

Compatibility Library

- Lets ARC code run on older versions of Mac OS X and iOS
- Automatically linked into your application if needed
  - Selected by deployment target



Compatibility Library

Weak references

- No weak references on Mac OS 10.6 / iOS 4
- Migration tool will use __unsafe_unretained for assign properties

```
@interface PageView : NSView {
  __unsafe_unretained id <PageViewDelegate> delegate;
  /* ... */
}
@property(unsafe_unretained) id <PageViewDelegate> delegate;
@end
```

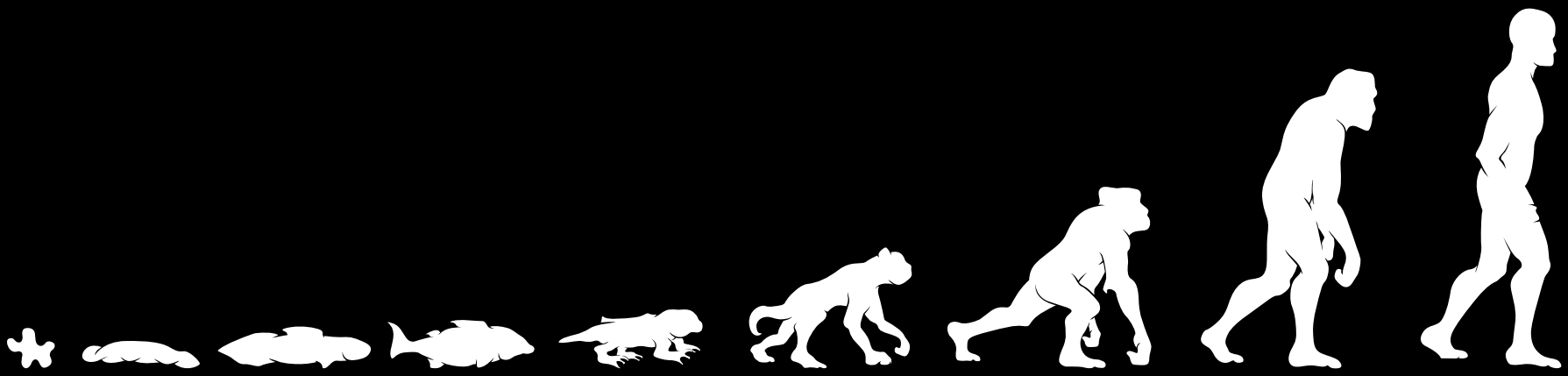
Demo

ARC Migration with Xcode 4.2

Evolution of Objective-C

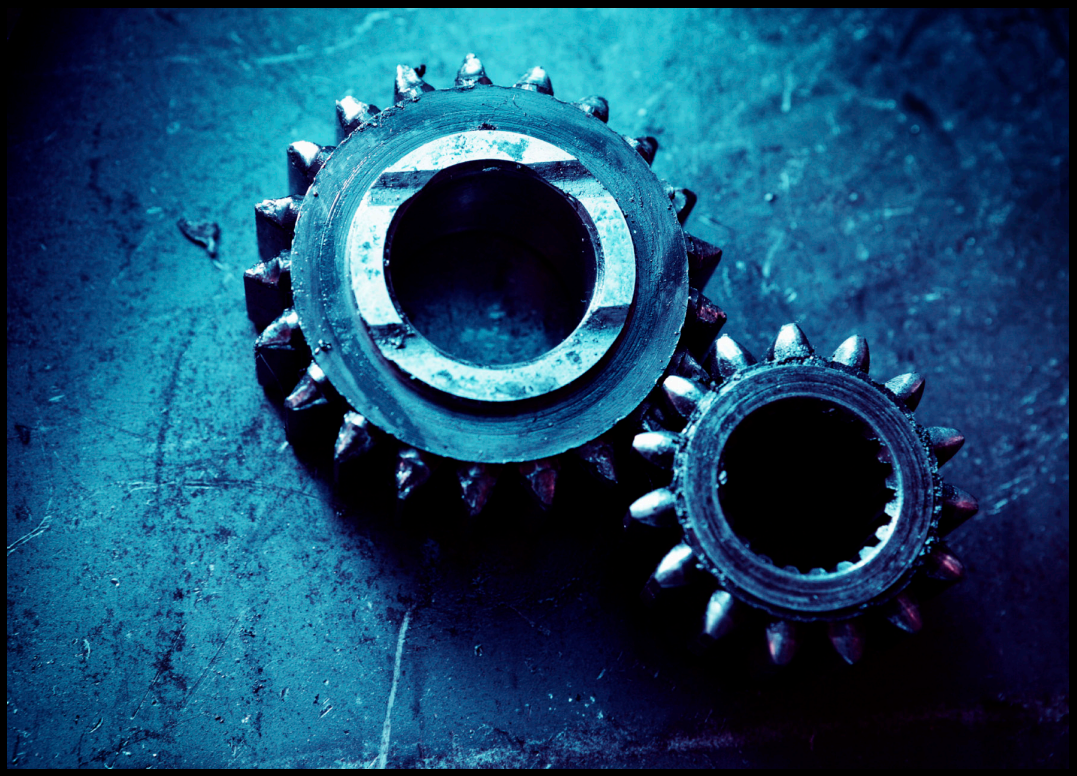
Simpler and safer through automation

- Object Oriented C
- Retain and Release
- Properties
- Blocks
- ARC



For More Details

- Non-ARC improvements
- Blocks improvements
- Performance details
- Runtime tricks
- Q & A



Objective-C Advancements – In Depth	Mission Friday 11:30AM

More Information

Michael Jurewitz
Developer Tools Evangelist
jurewitz@apple.com

“Programming with ARC” Release Notes

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

Related Sessions

Moving to Apple LLVM compiler	Nob Hill Wednesday 10:15AM
Objective-C Advancements In-Depth	Mission Friday 11:30AM

Labs

Objective–C and Automatic Reference Counting Lab	Developer Tools Lab A Wednesday 9:00AM
Objective–C and Automatic Reference Counting Lab	Developer Tools Lab B Thursday 2:00PM


WWDC2011