Misc. ARC

NSInvocation

- "Freeze-Dried" message expression
- Fill in target, selector, arguments
- Execute at a later time by:

```
[myInvocation invoke];
```

- As many times as you want.

```
LineGraphic *graphic = ...

NSInvocation *drawInvocation =
    [NSInvocation  invocationWithMethodSignature:
          [graphic methodSignatureForSelector:
                @selector(drawWithColor:width:)]];

[drawInvocation setTarget: graphic];
[drawInvocation setSelector:
      @selector(drawWithColor:width:)];
```

```
NSColor *color = [NSColor redColor];
float linewidth = 2.0;

[drawInvocation setArgument: &color atIndex: 2];
[drawInvocation setArgument: &linewidth atIndex: 3];

...

[drawInvocation invoke];
```

- NSInvocations do not automatically retain their target or arguments.
- If you want them retained, you must:

```
[myInvocation retainArguments];
```

- Even with ARC !
- Curiously it adds a retain to the arguments every time you execute `invoke`. It does all the releases at once when the invocation object goes away

A trap under manual reference counting:

```
int(^getDoublerBlock())(int)
{
    int(^doublerBlock)(int) =
            ^(int n){ return 2*n; };

  return doublerBlock;
}
```

Blocks start out on the stack. Under manual reference counting you must copy them (which moves them to the heap) before passing them out of their original scope. Like any other `copy` you must balance it with a `release`.

```
int(^getDoublerBlock())(int)
{
    int(^doublerBlock)(int) =
            ^(int n){ return 2*n; };
  return [[doublerBlock copy] autorelease];
}

int(^doubler )(int) = getDoublerBlock();
int sevenDoubled = doubler(7);
```

Under ARC you don’t have to worry about it. ARC will copy the block to the heap for you.

You don’t have to think about it except...

Another trap under manual reference counting

```
-(NSMutableArray*) getArrayWithBlock
{
  NSMutableArray *array = [NSMutableArray array];

  void (^blk)() = ^(){ doSomething; };

  [array addObject: blk];

  return array;    // WRONG !
}
```

The `retain` from adding the block to the array does nothing, since the (uncopied) block is still stack-based.

- Without ARC you have to copy the block yourself, and put the copy in the array.
- Under ARC you *still* have to copy the block yourself, and put the copy in the array.
- Any block that goes down the stack into something that will retain it must be copied, even with ARC.
- Apple considers this a bug in Clang. May be fixed in Xcode some day.

CF objects returned from methods

```
UIColor *aUIColor = ...
NSMutableArray *colorArray = [NSMutableArray array];
id aCGColor = (id)[aUIColor CGColor];
[array addObject: aCGColor];
```

This is OK. ARC understands OC method naming conventions. CGColor doesn't begin with "alloc", "new", "copy" or "mutableCopy" so it doesn't transfer ownership.

But...

```
CGColorRef color =
    [UIColor colorWithRed: 0.2
                    green: 0.3
                     blue: 0.4
                    alpha: 1.0].CGColor;
[[self.view layer] setBackgroundColor: color];
```

Under ARC this crashes on a device ! The interior pointer problem strikes again ?

Interior Pointer Problem

- An automatic system manages an object's lifetime.
- The object manages the lifetime of some resource that the automatic system *doesn't* manage.
- You get a pointer to the interior resource.
- There are no more references to the object, so the automatic system gets rid of it. The object, going away, releases the interior resource.
- You are stuck with a dead pointer.

GC interior pointer problem

```
NSMutableData *mutableData =  ...
char *mutableBytes = [mutableData mutableBytes];
NSUInteger length = [mutableData length];
while ( length )
{
    // Do something with mutableBytes.
    length--;
 }
...
```

Fix #1

```
CGColorRef color =
    CGColorRetain( [UIColor colorWithRed: 0.2
                                   green: 0.3
                                    blue: 0.4
                                   alpha: 1.0].CGColor);
[[self.view layer] setBackgroundColor: color];
CGColorRelease( color );
```

Fix #2

```
UIColor *uiColor =[UIColor colorWithRed: 0.2
                            green: 0.3
                             blue: 0.4
                            alpha: 1.0];

CGColorRef color = uiColor.CGColor;
[[self.view layer] setBackgroundColor: color];
[uiColor self];
```

The real fix?
(Quote from Clang ARC spec)

"An Objective-C method returning a non-retainable pointer may be annotated with the `objc_returns_inner_pointer` attribute to indicate that it returns a handle to the internal data of an object, and that this reference will be invalidated if the object is destroyed. When such a message is sent to an object, the object's lifetime will be extended until at least the earliest of:

- the last use of the returned pointer, or any pointer derived from it, in the calling function or
- the autorelease pool is restored to a previous state."