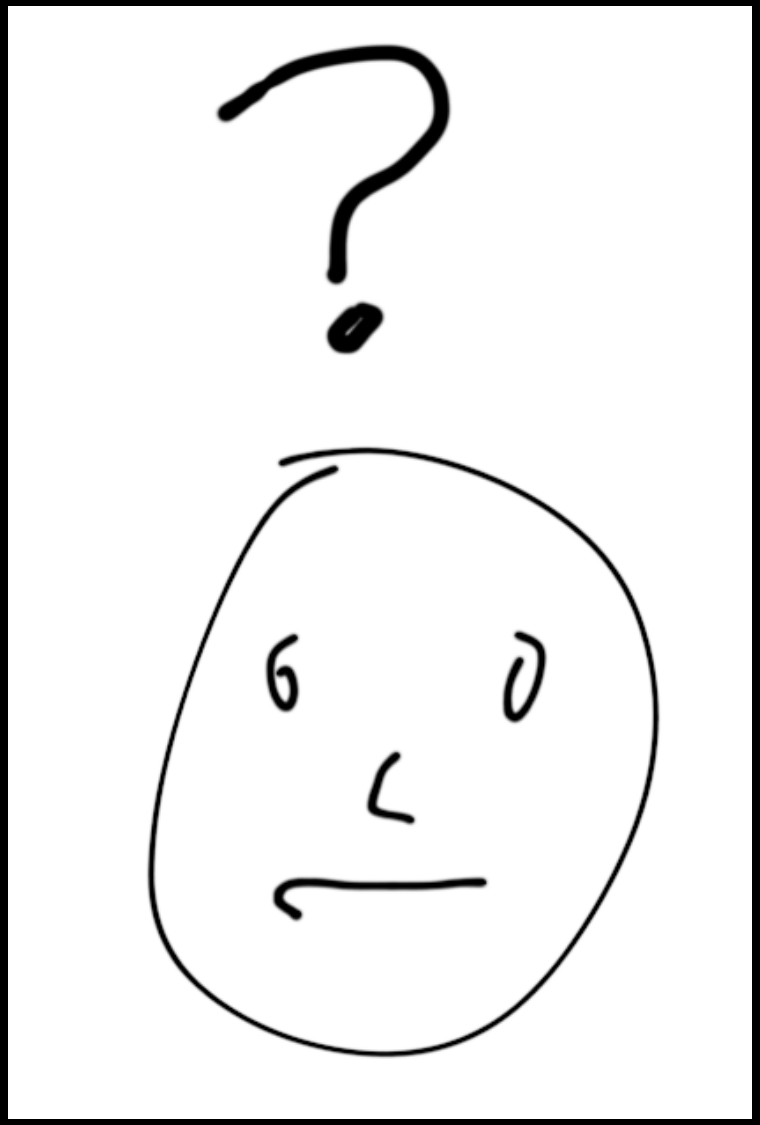
Automatic Reference Counting

Mike Ash

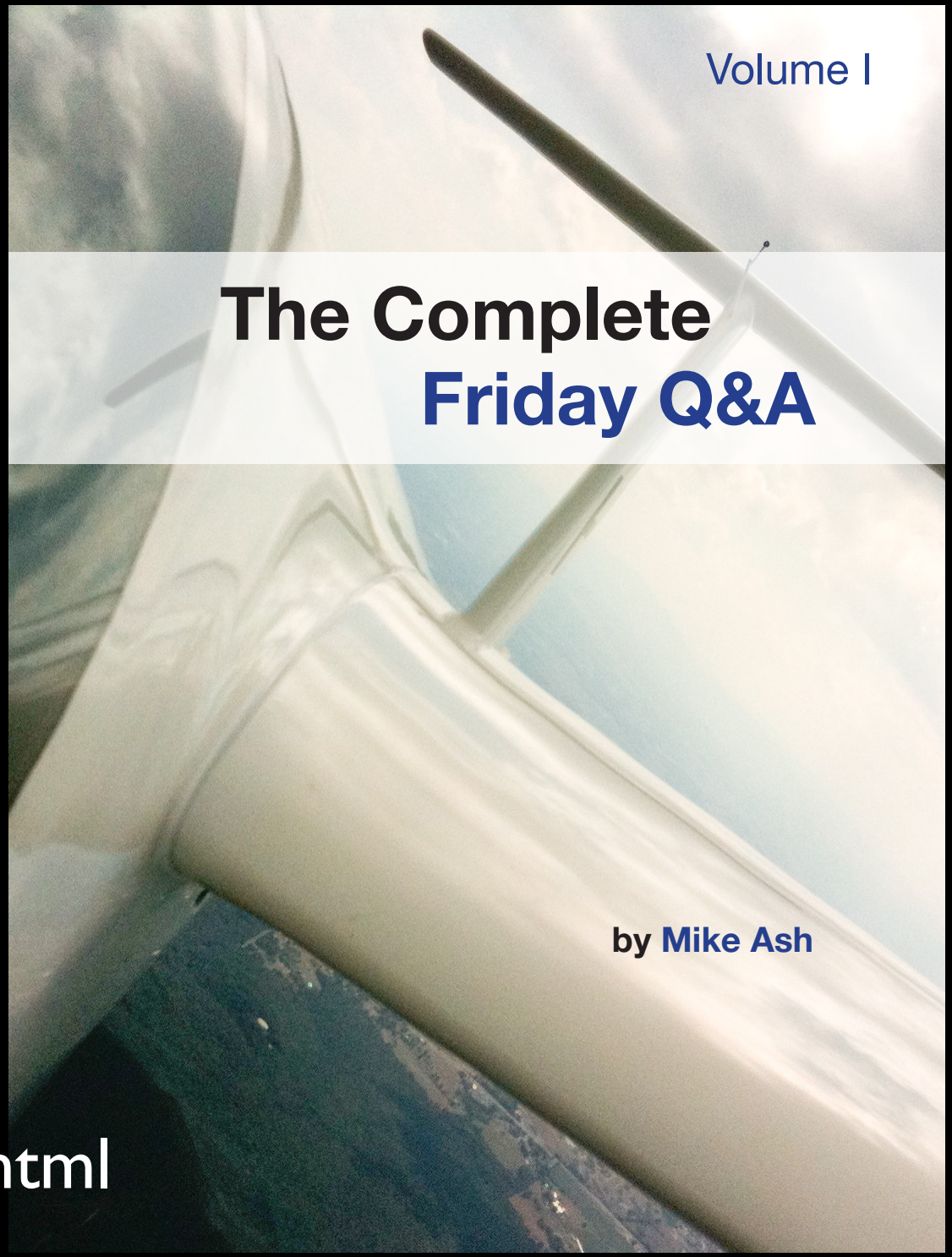
CAUTION: use at your own risk. No warranty expressed or implied. Do not take internally. If ingested, consult a mortician. This is not legal advice; if you require legal advice, consult an attorney. Not valid unless signed. Void where prohibited. Consult your physician if memory leaks persist for more than four hours. Save the whales. Feed the hungry. Free the mallocs.

ARC



The Man, The Legend, The Landmark

- @mikeash
- mikeash.com
- Friday Q&A
- Written lots and lots of bugs



mikeash.com/book.html

Credentials

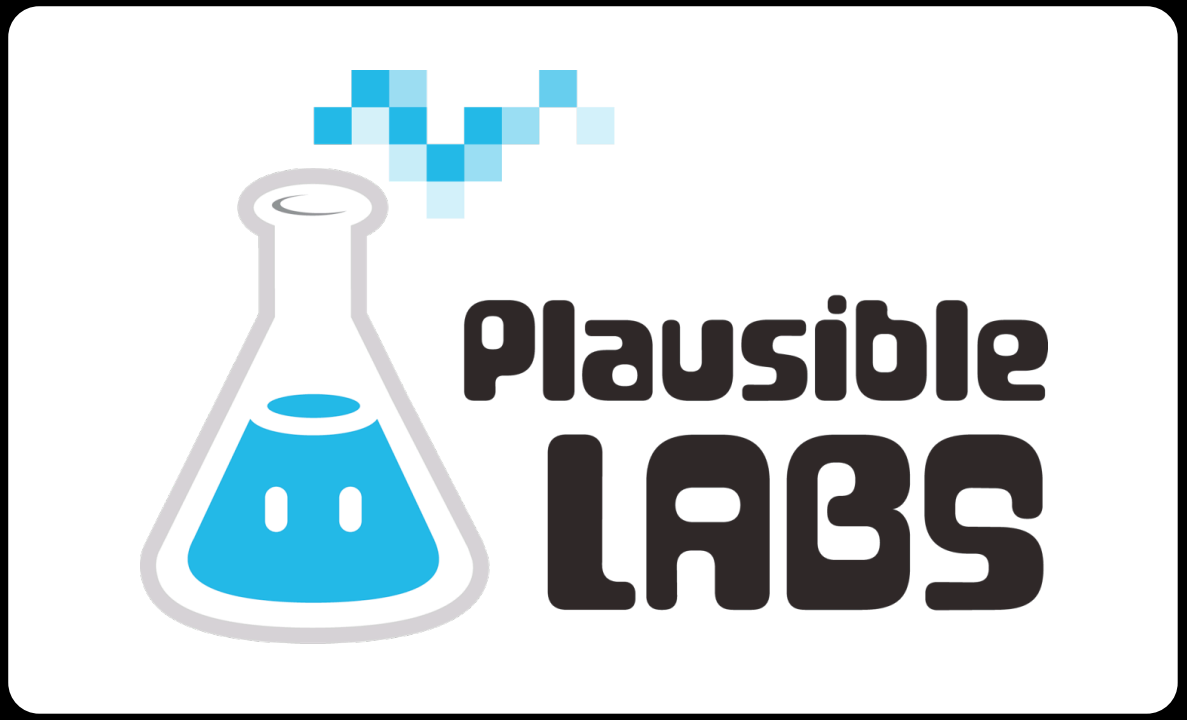
Qualifications

http://github.com/mikeash/

MAZeroingWeakRef

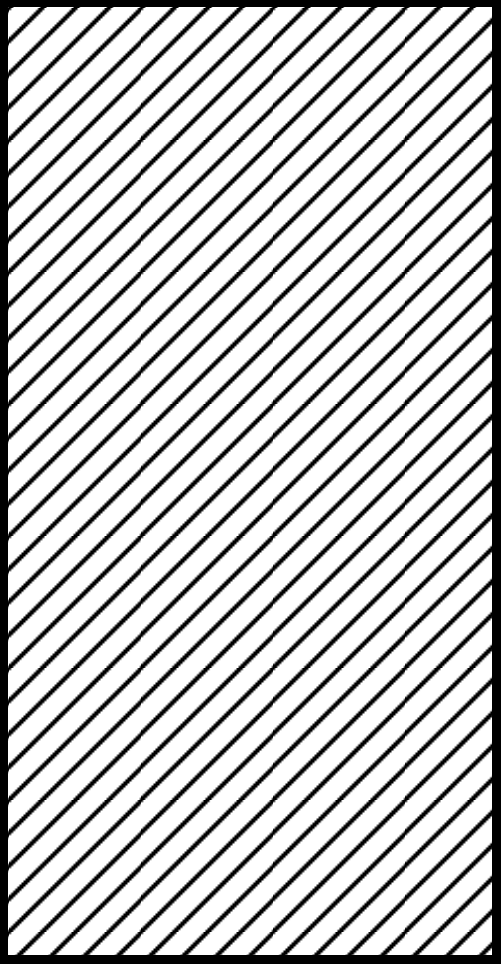
MAGenerator

MABlockClosure

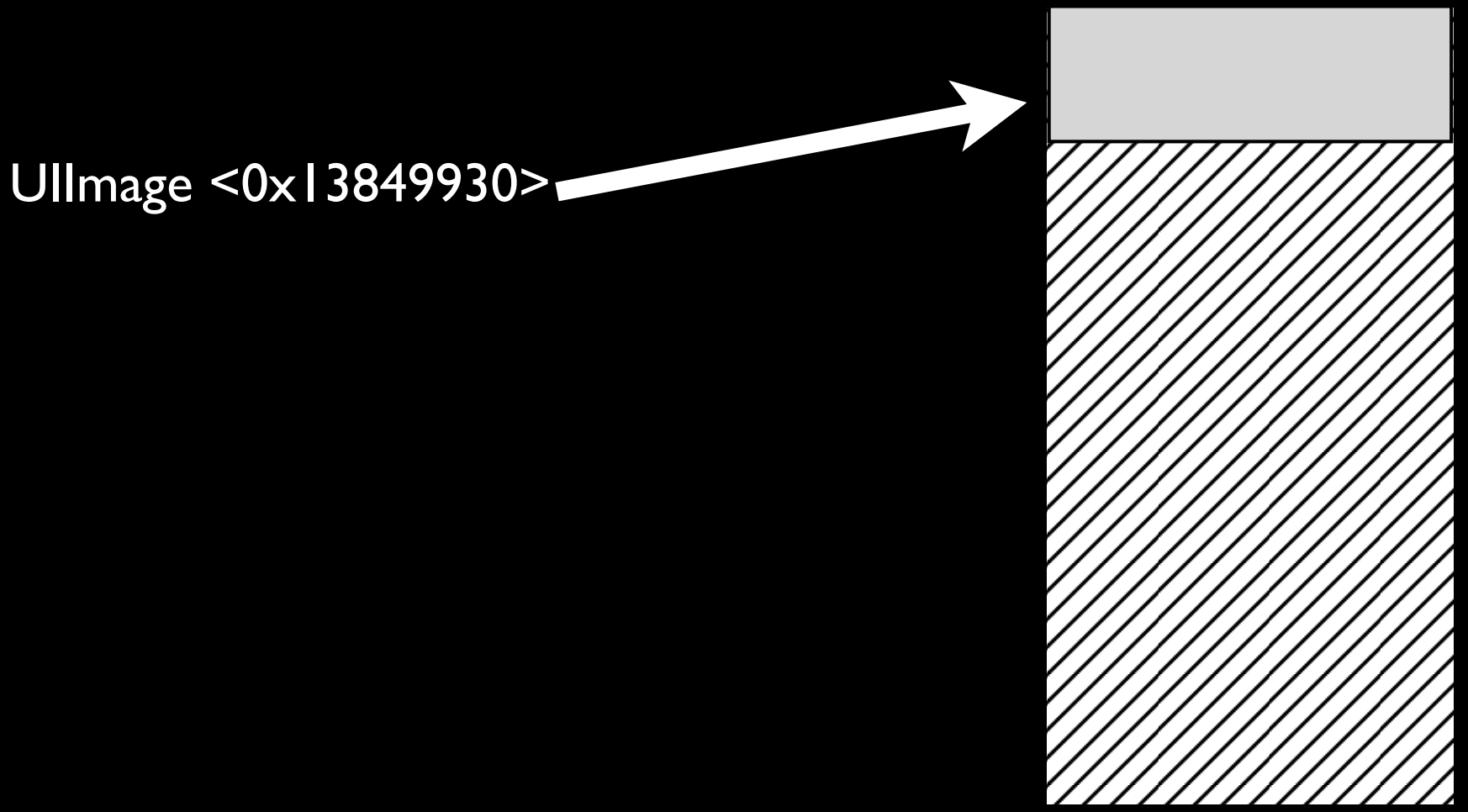

Plausible
LABS

Mac and iOS

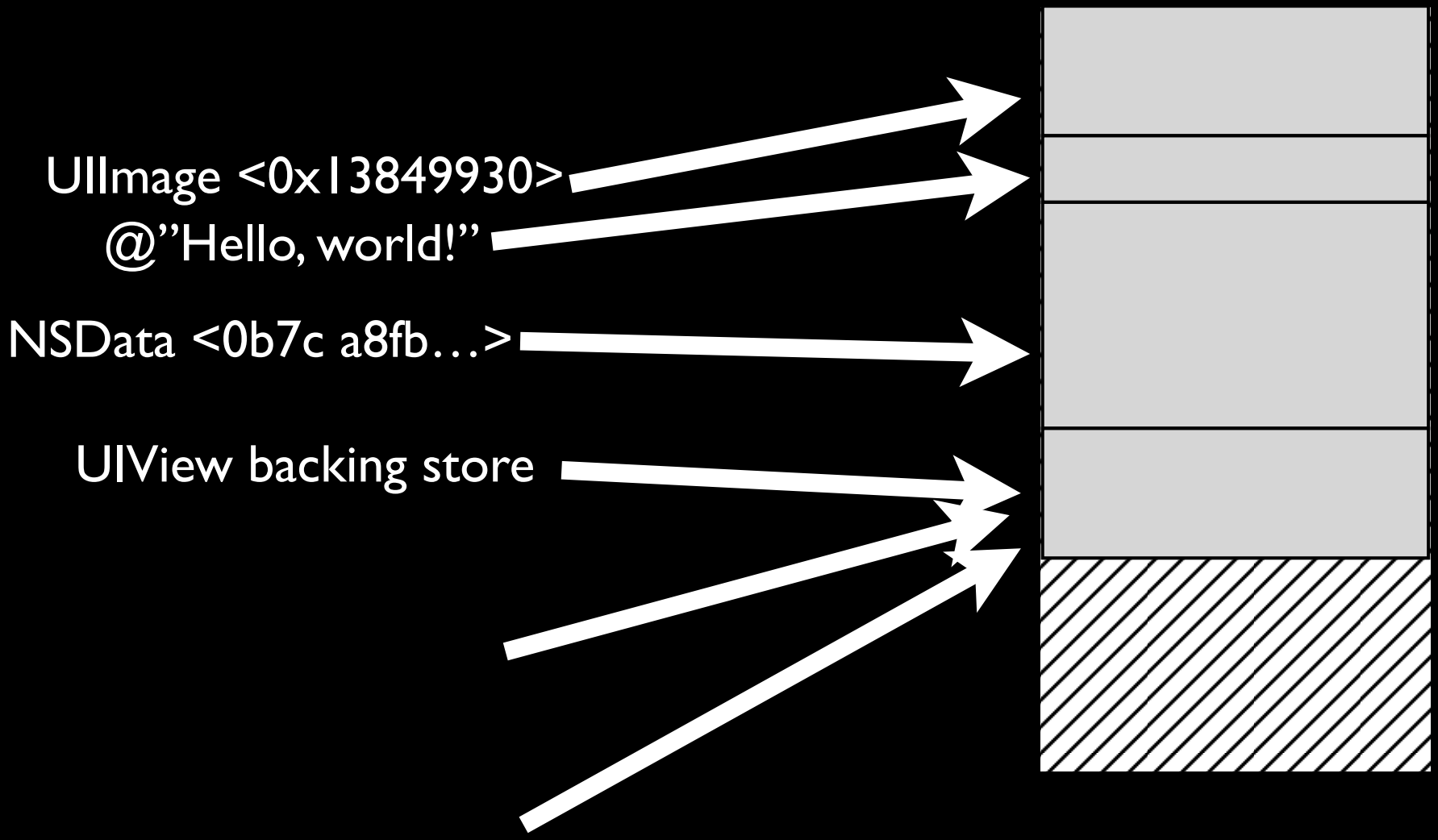
Memory Management 101



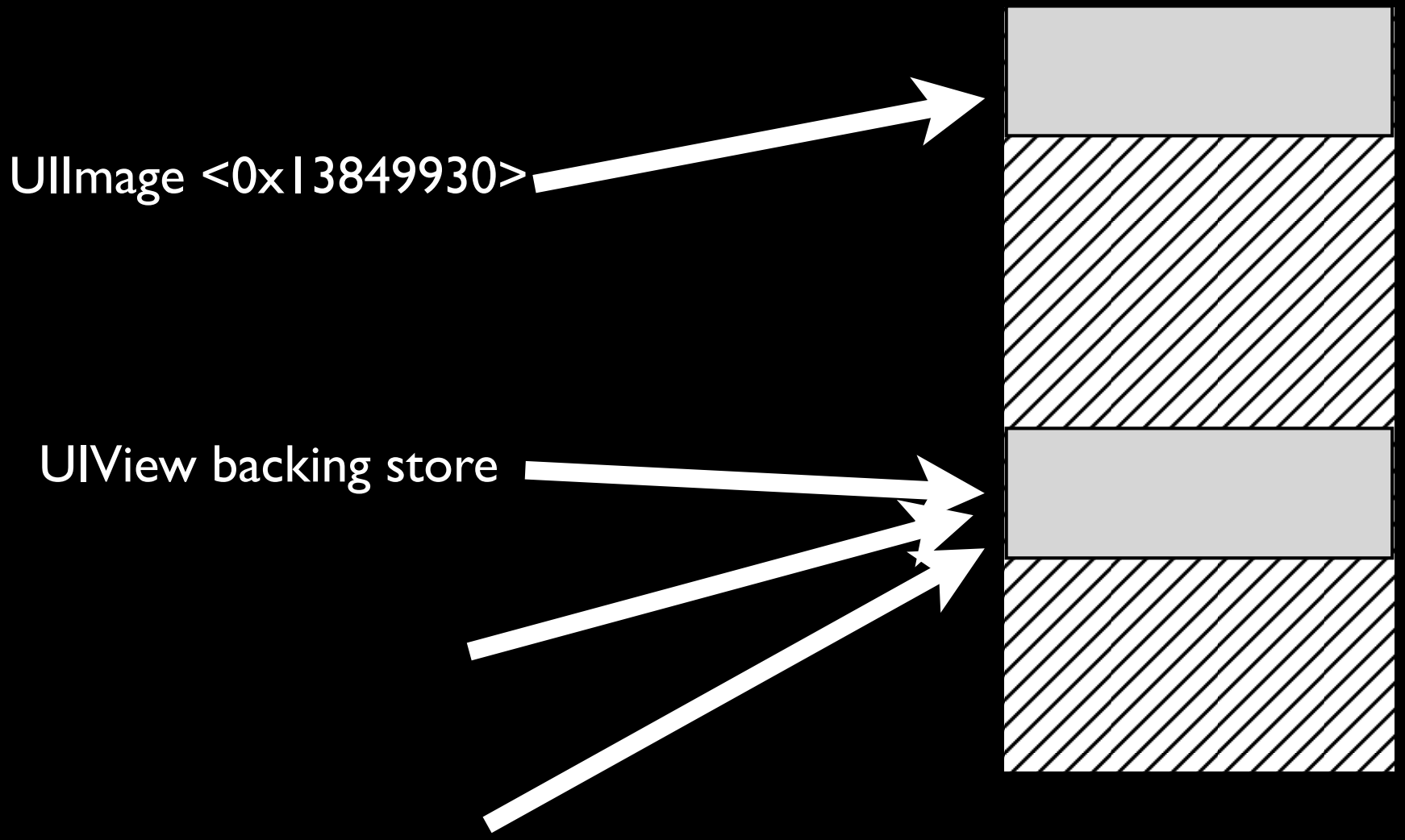
Memory Management 101



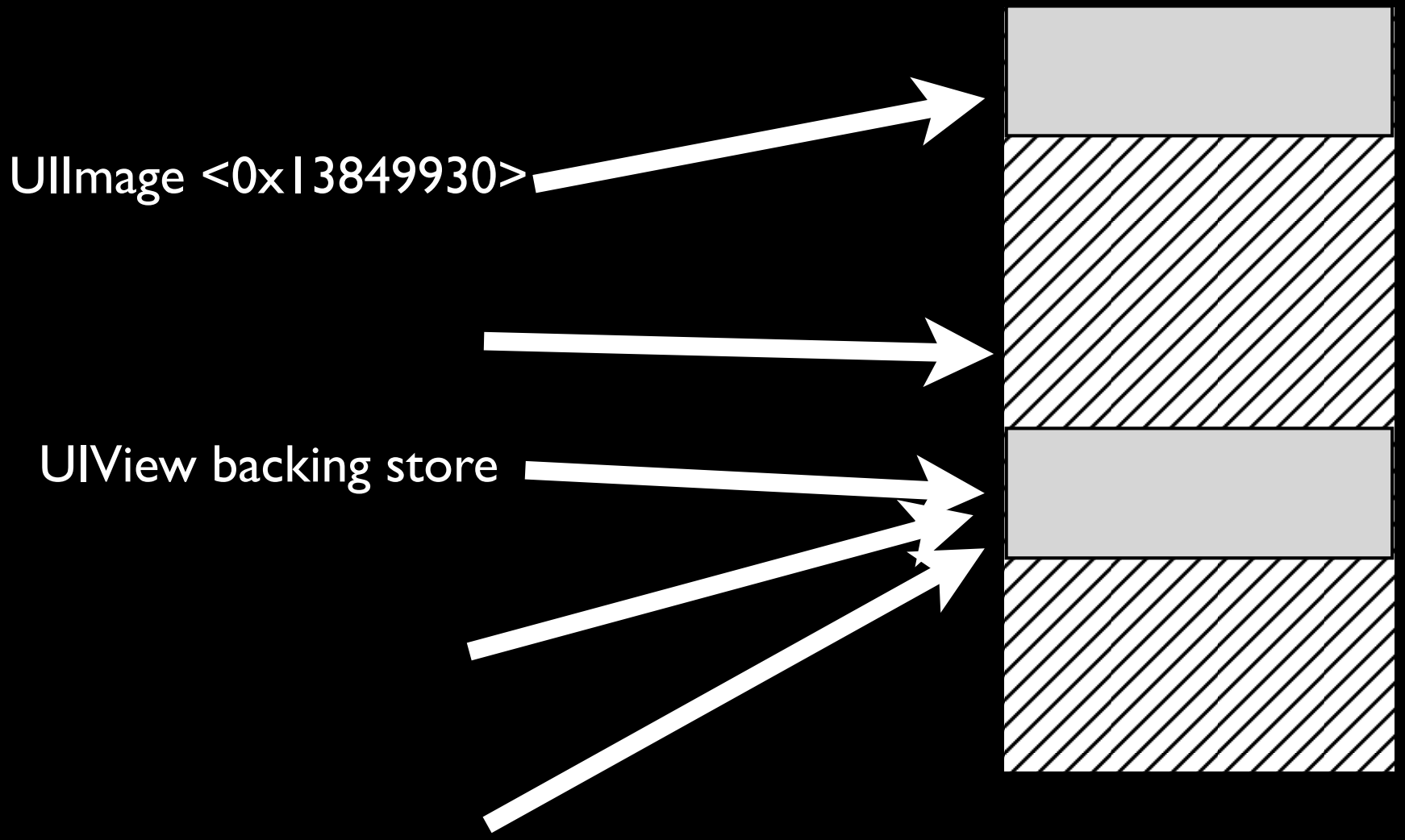
Memory Management 101



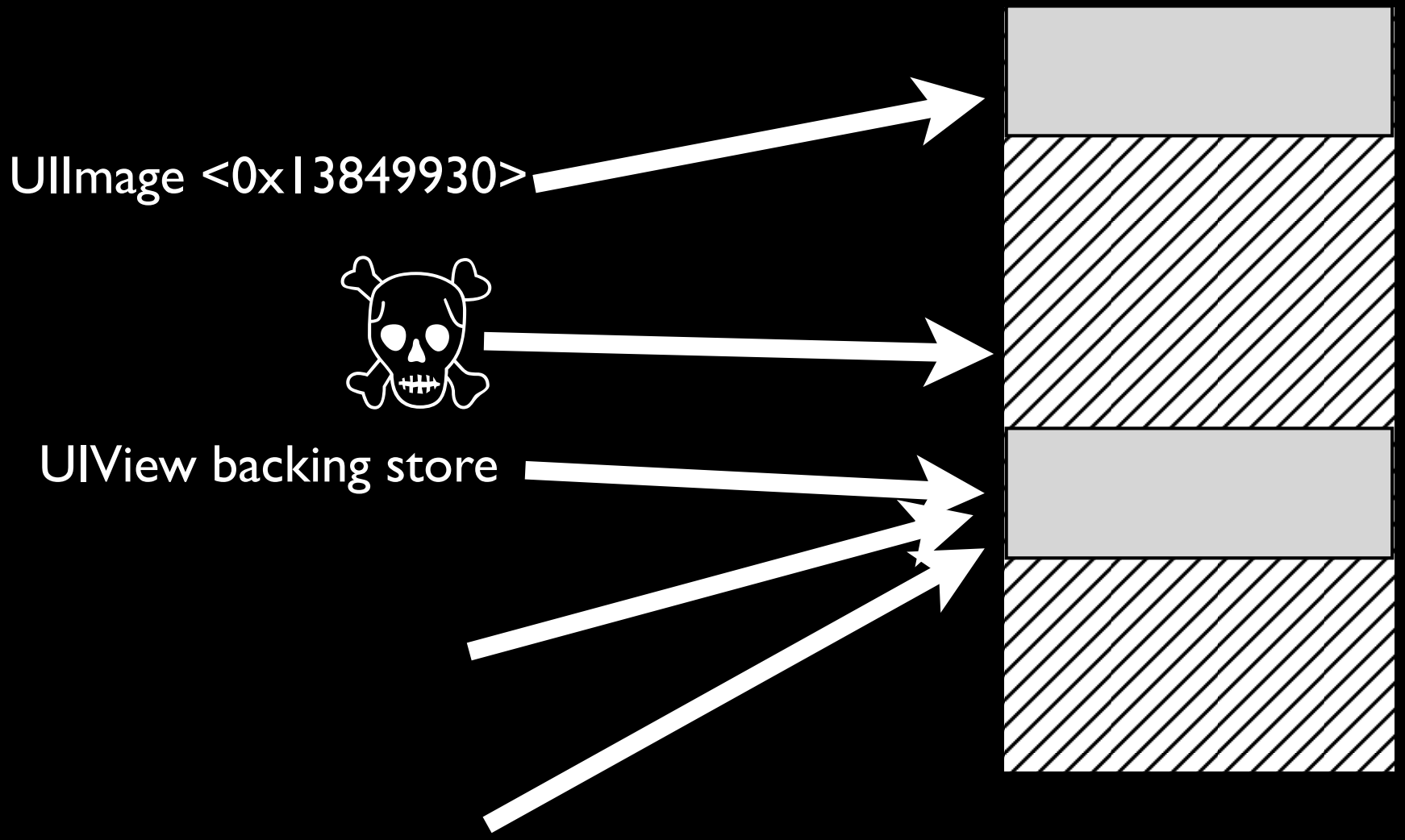
Memory Management 101



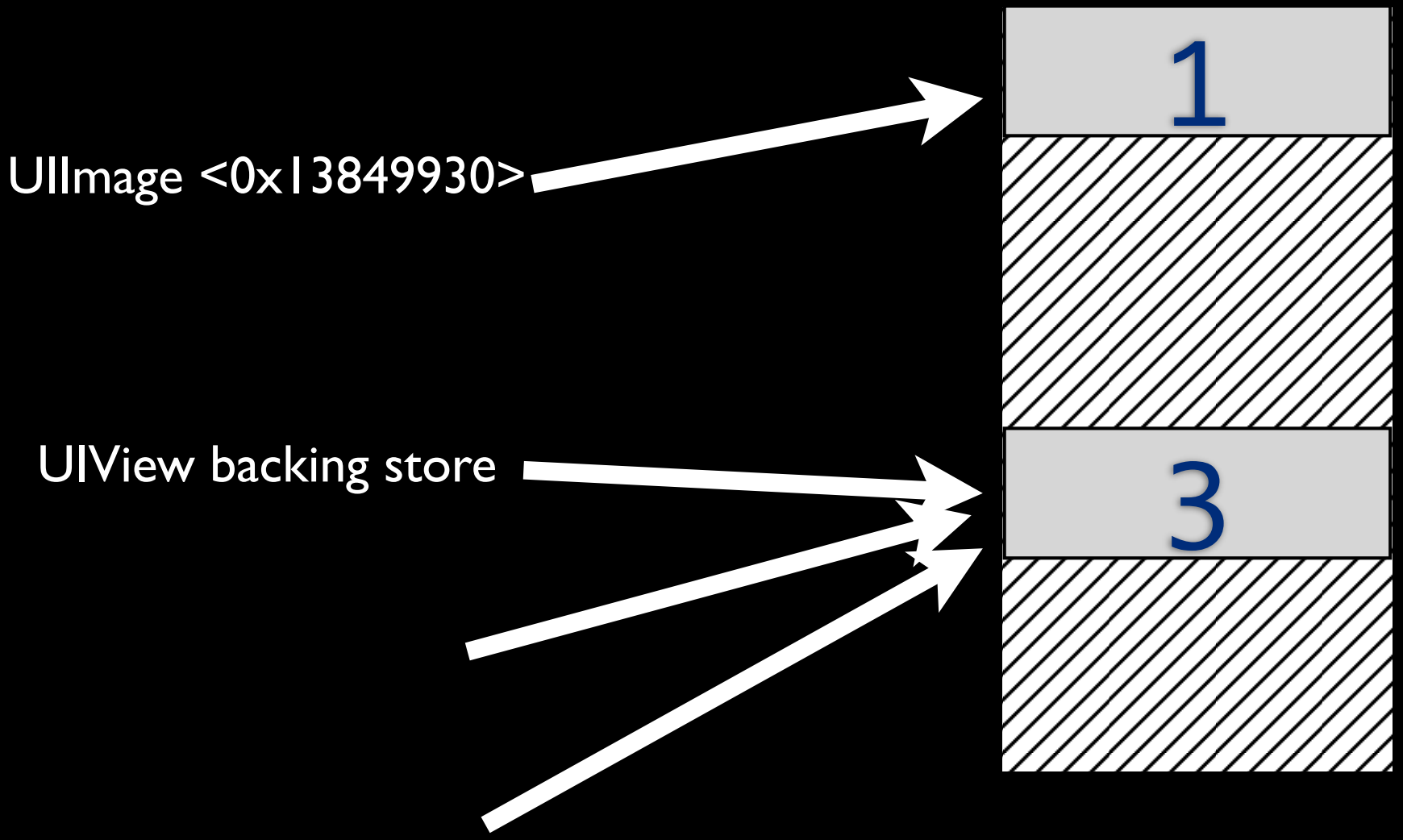
malloc/free



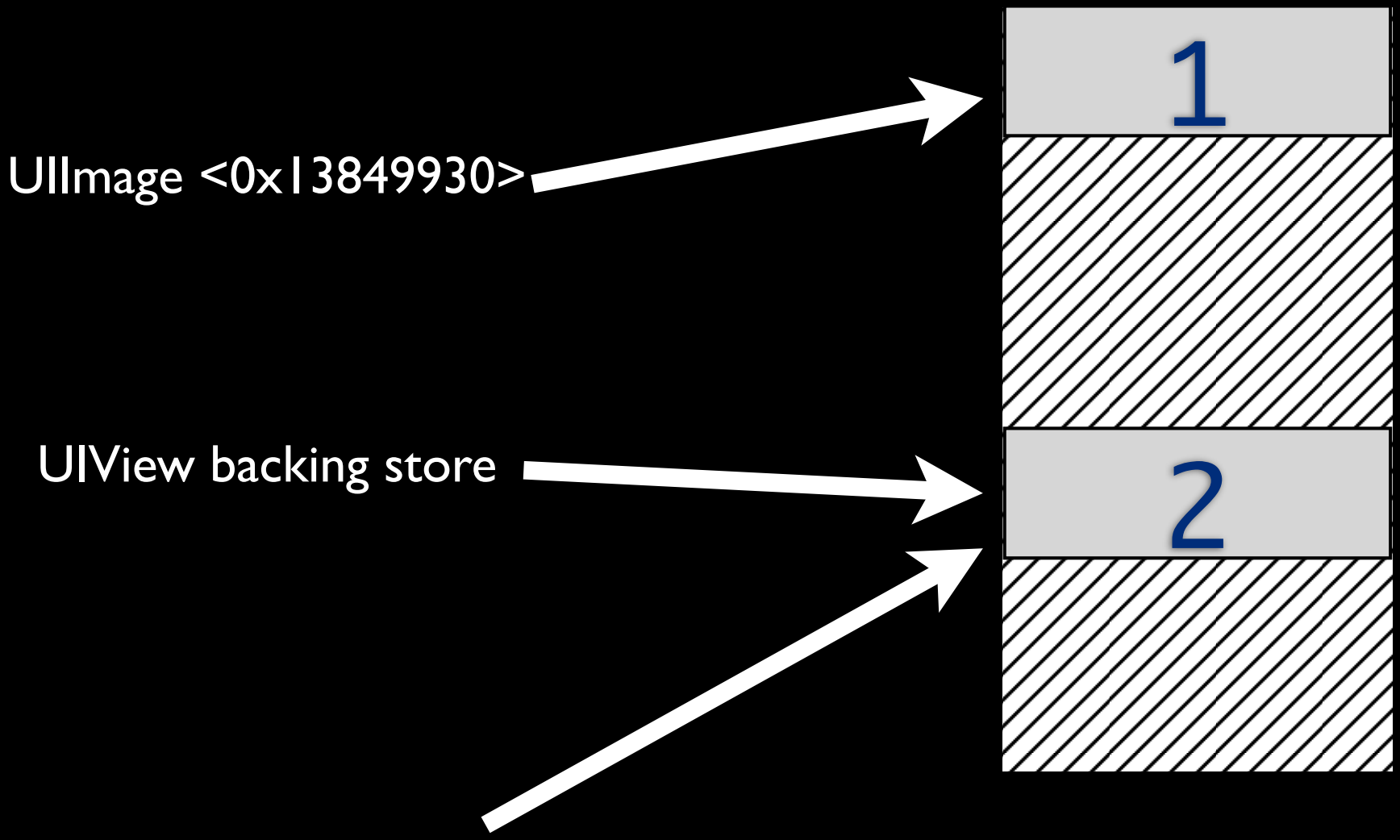
malloc/free



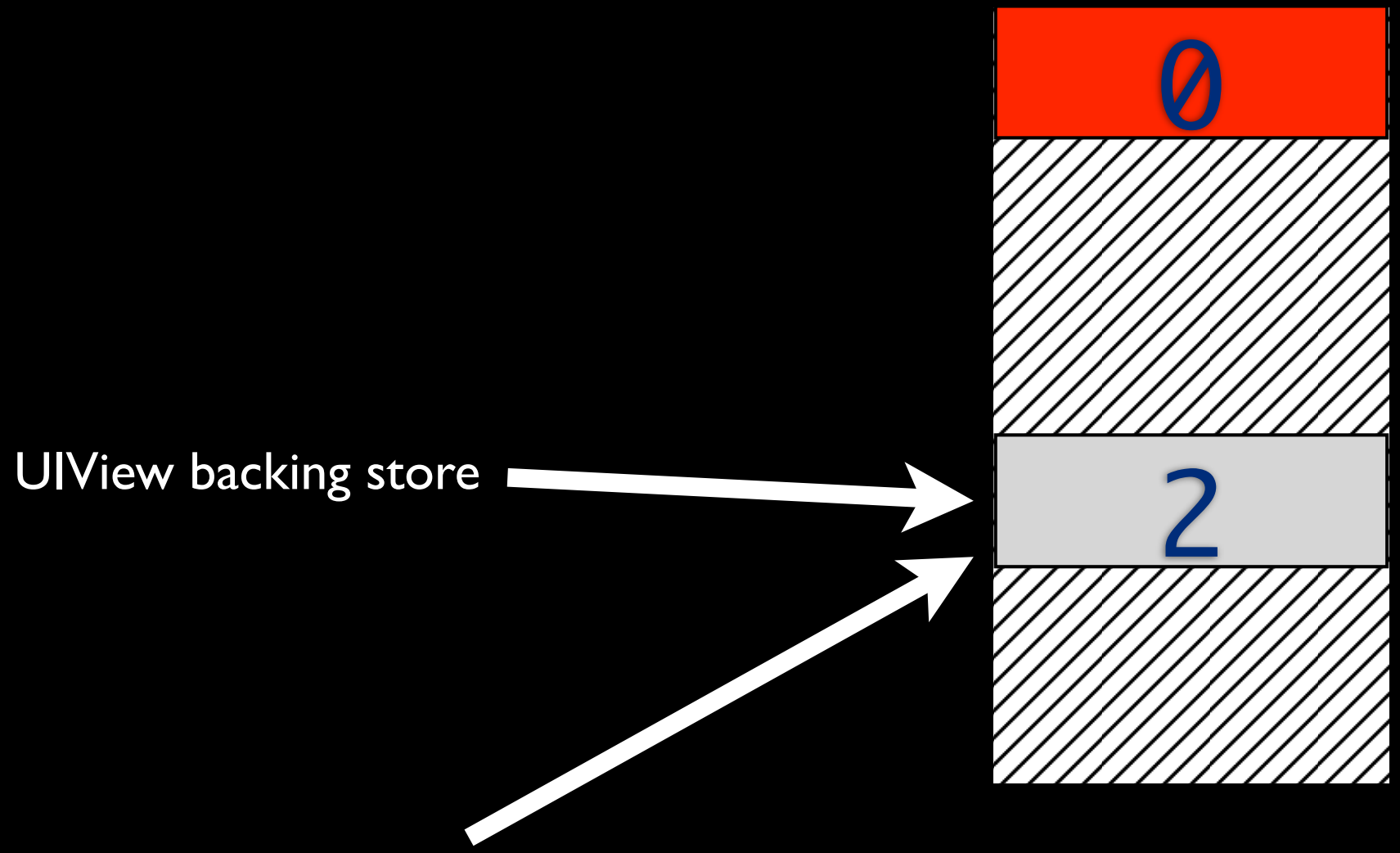
Manual Reference Counting



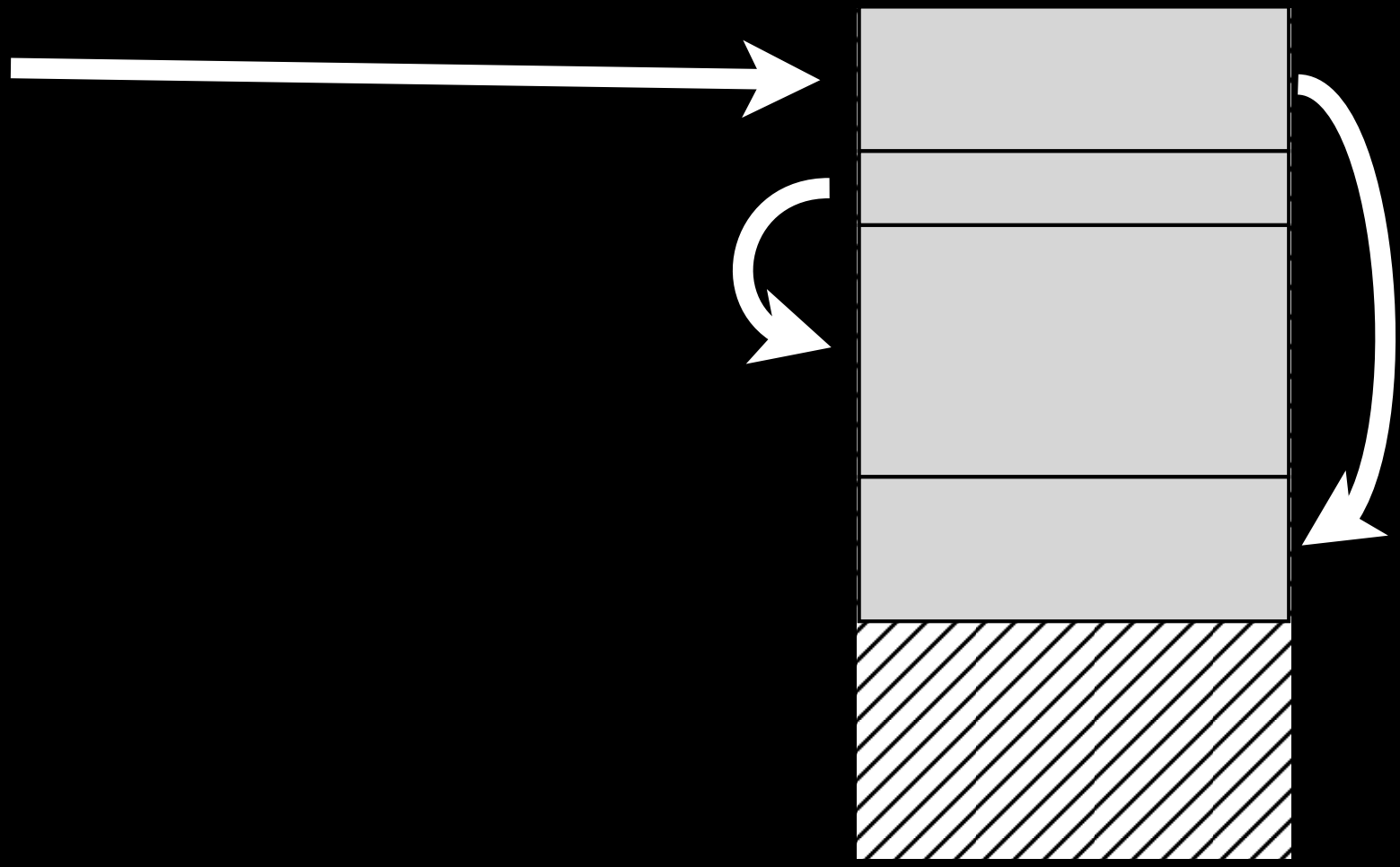
Manual Reference Counting



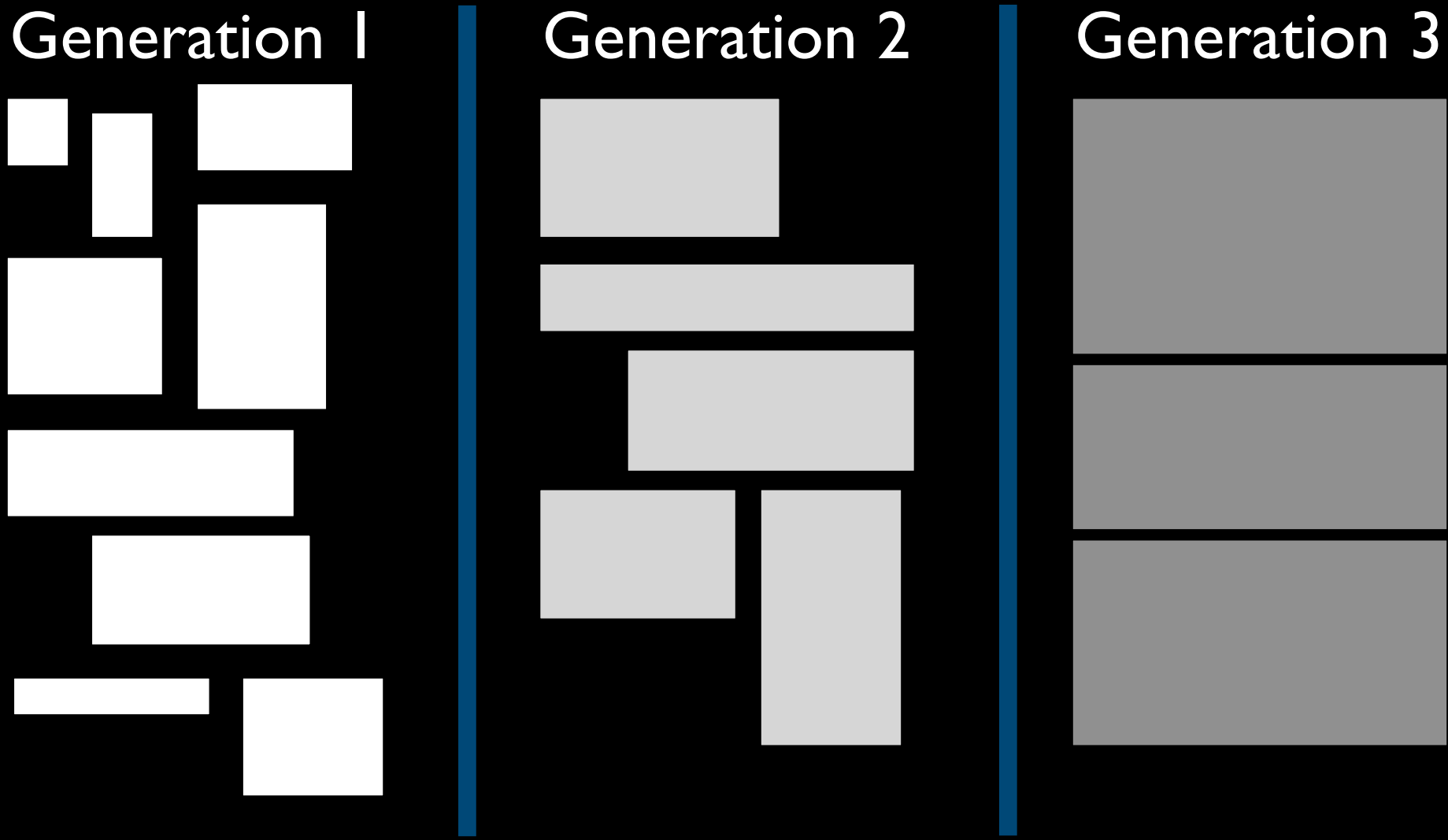
Manual Reference Counting



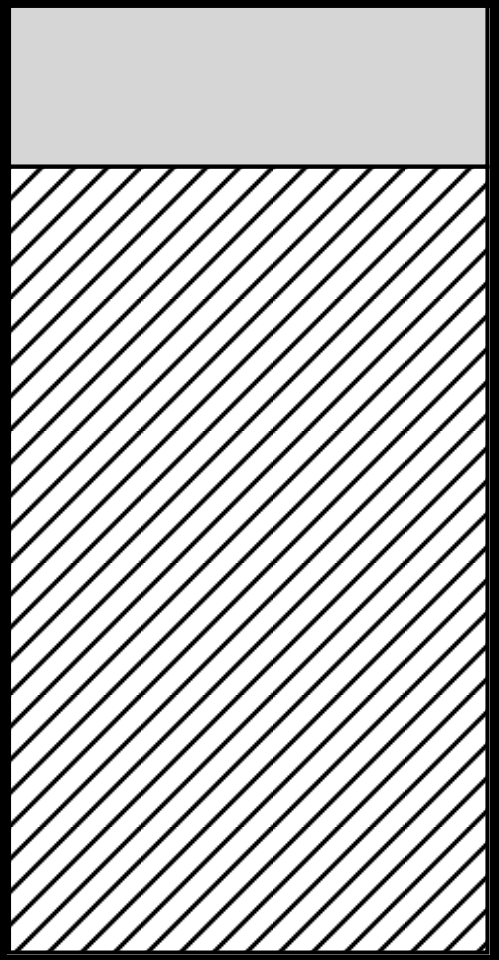
Garbage Collection 101



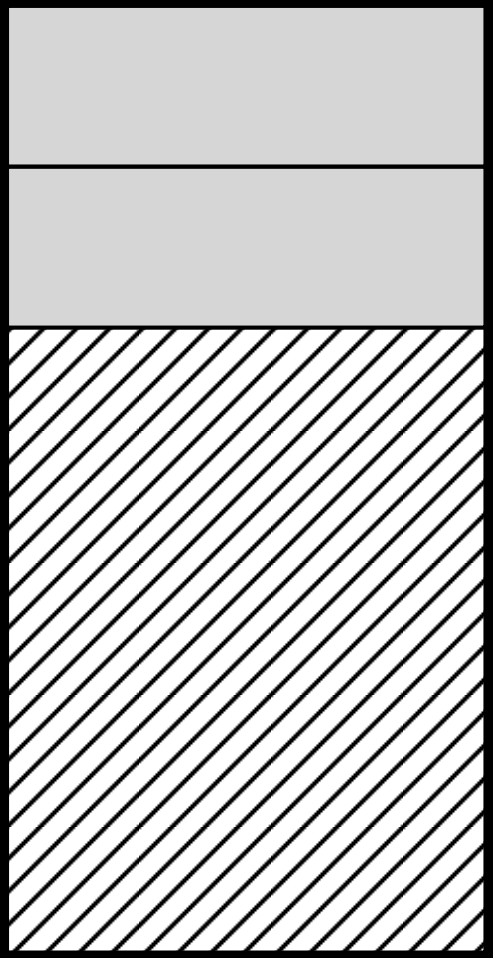
Garbage Collection 312



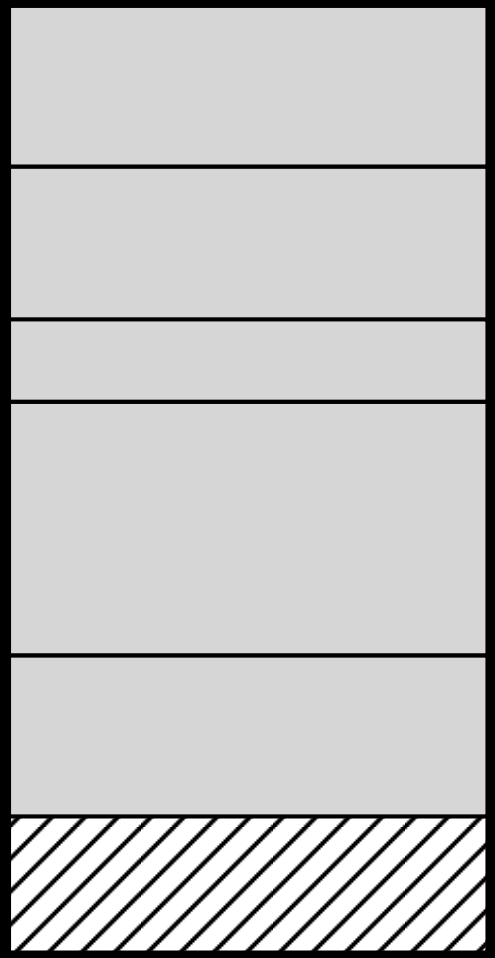
Garbage Collection 441



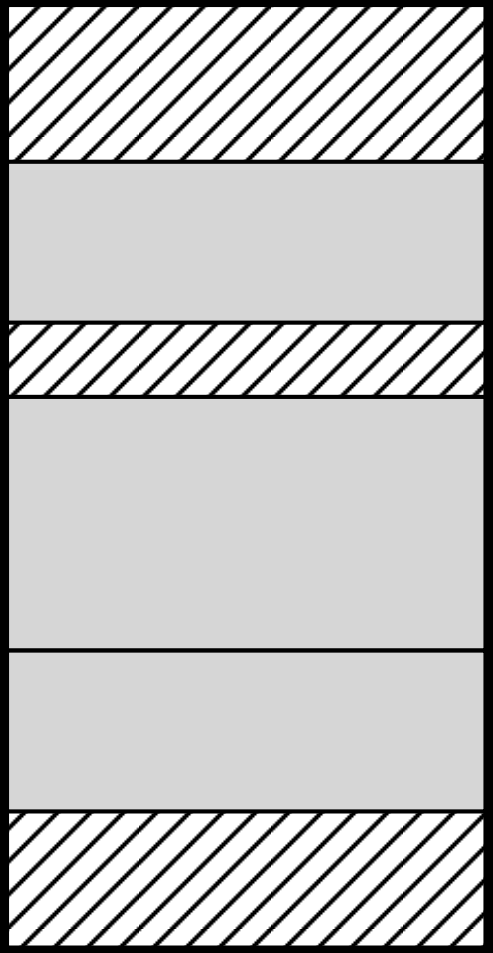
Garbage Collection 441



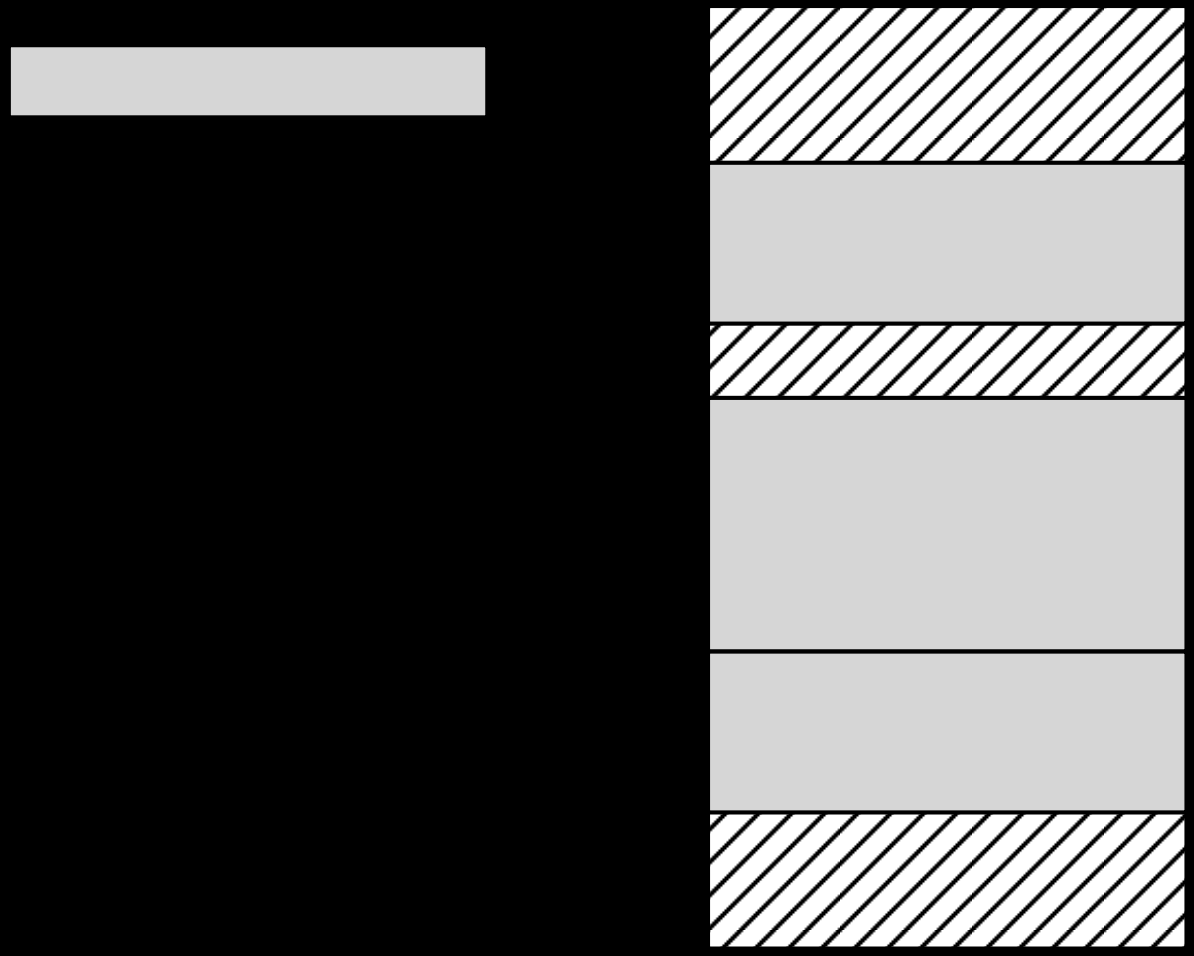
Garbage Collection 441



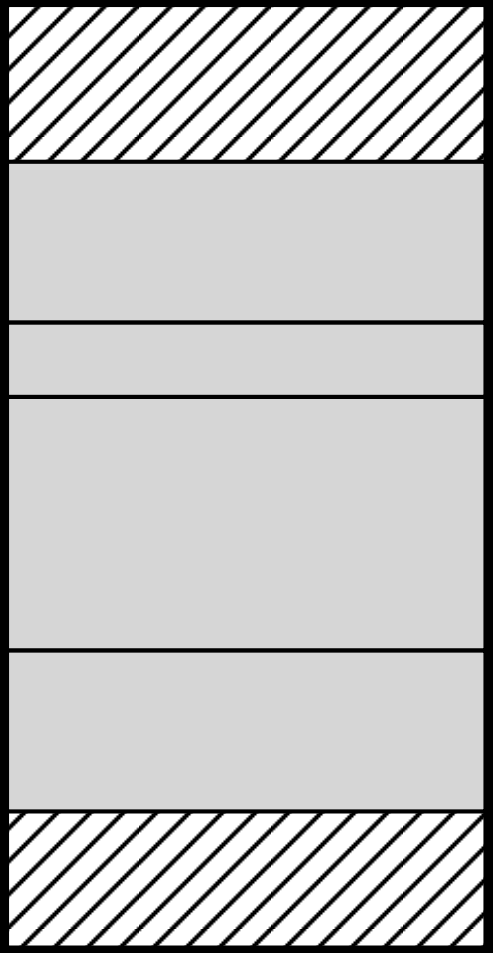
Garbage Collection 441



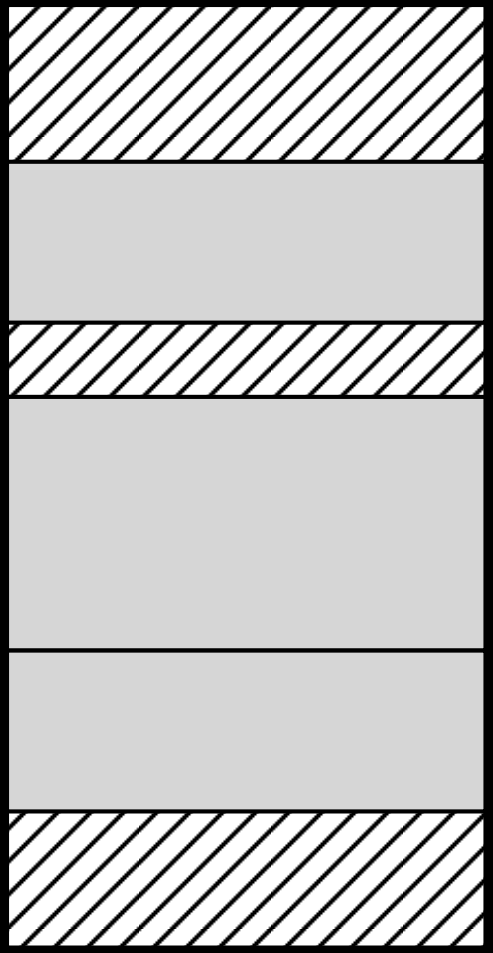
Garbage Collection 441



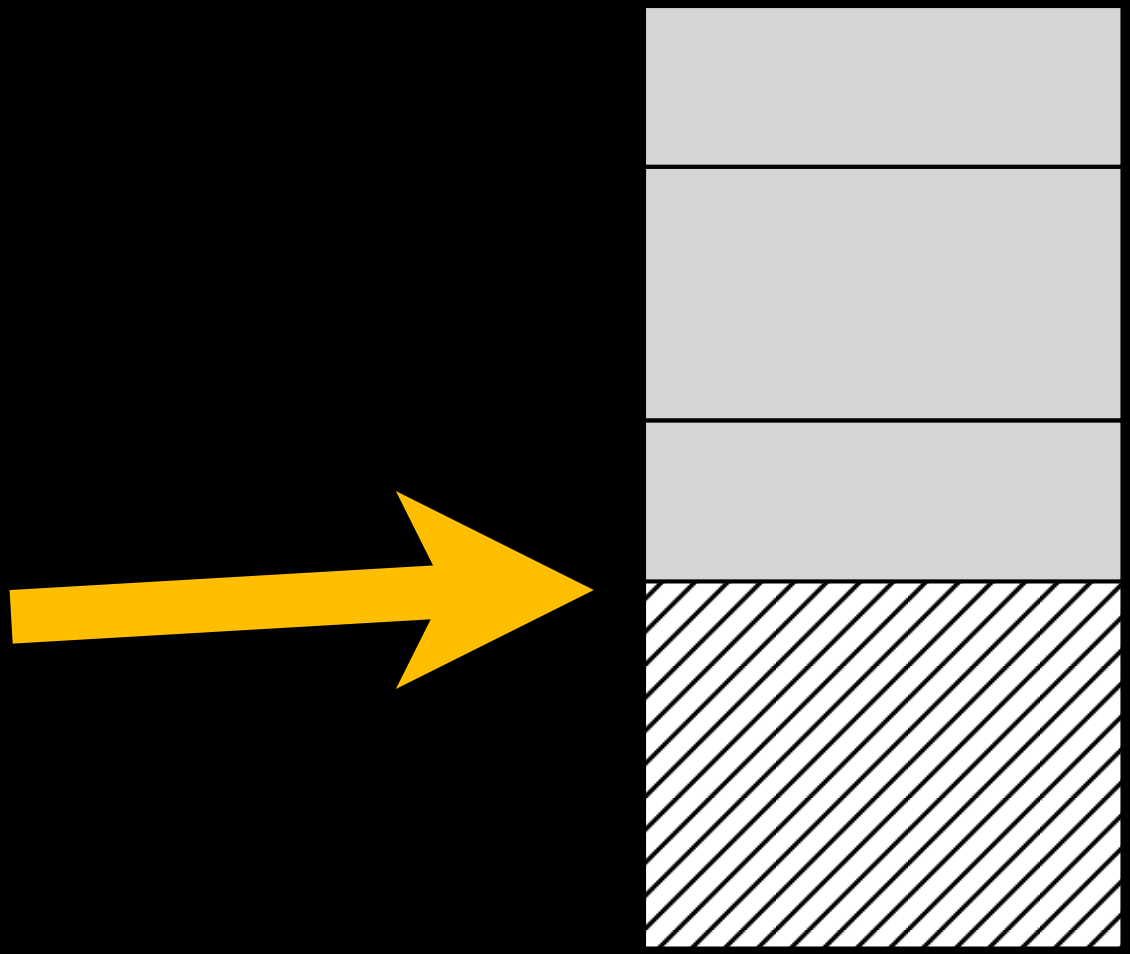
Garbage Collection 441



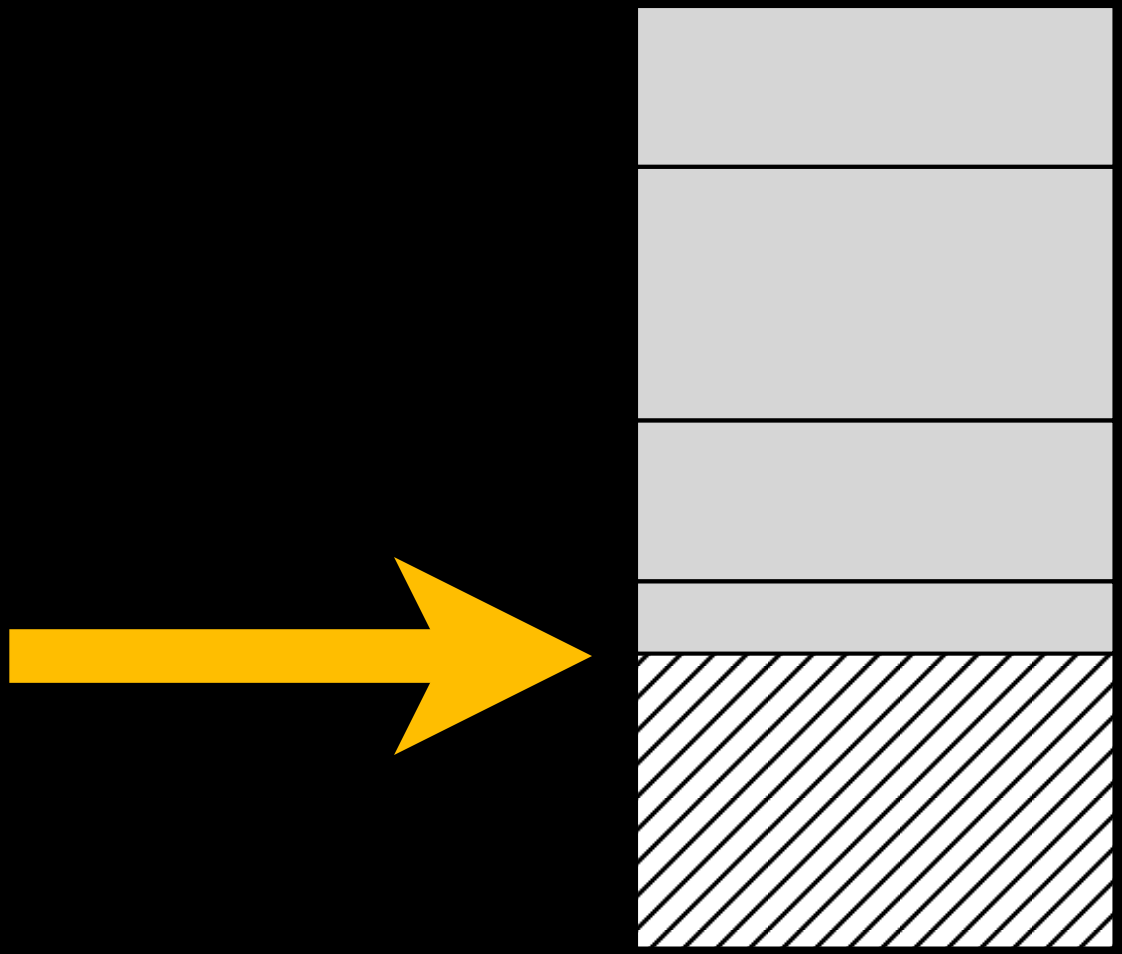
Garbage Collection 441



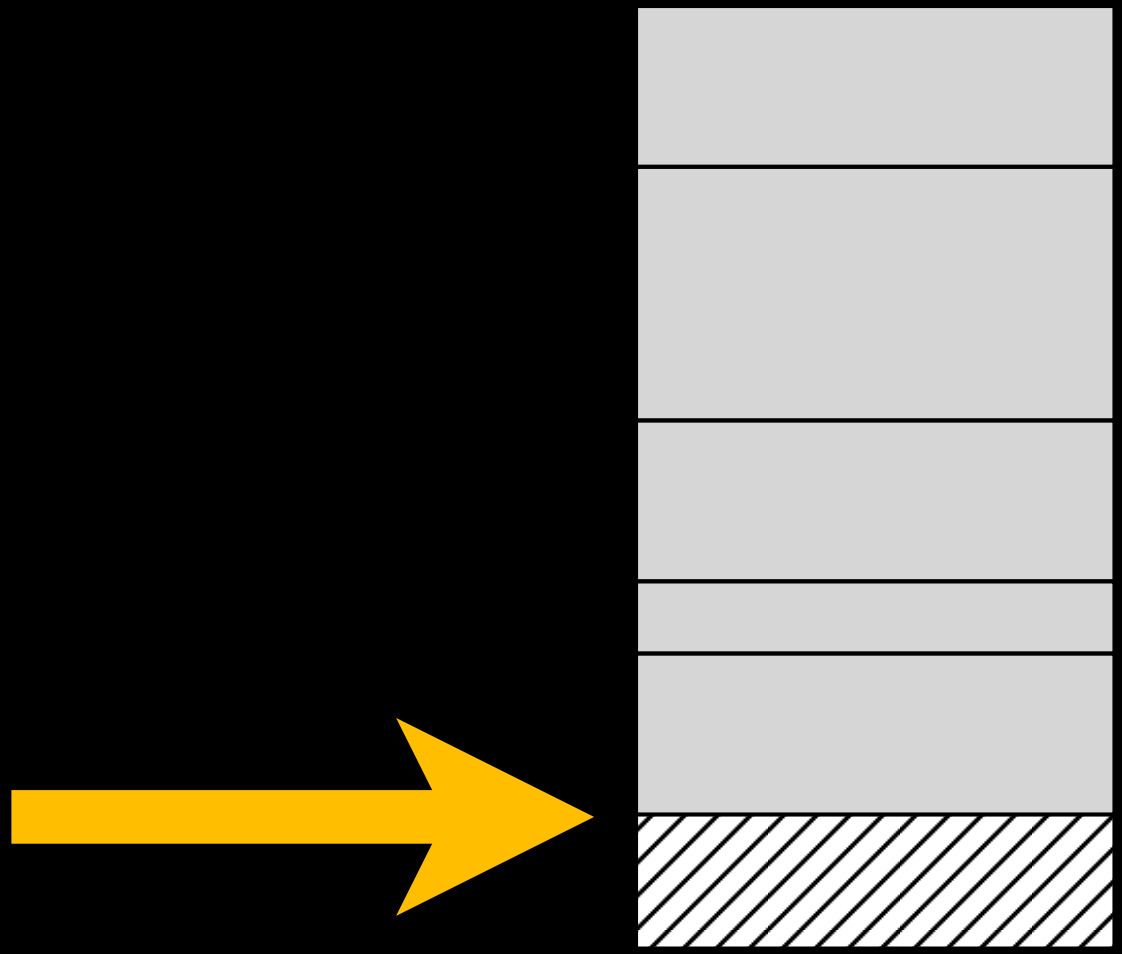
Garbage Collection 441



Garbage Collection 441



Garbage Collection 441

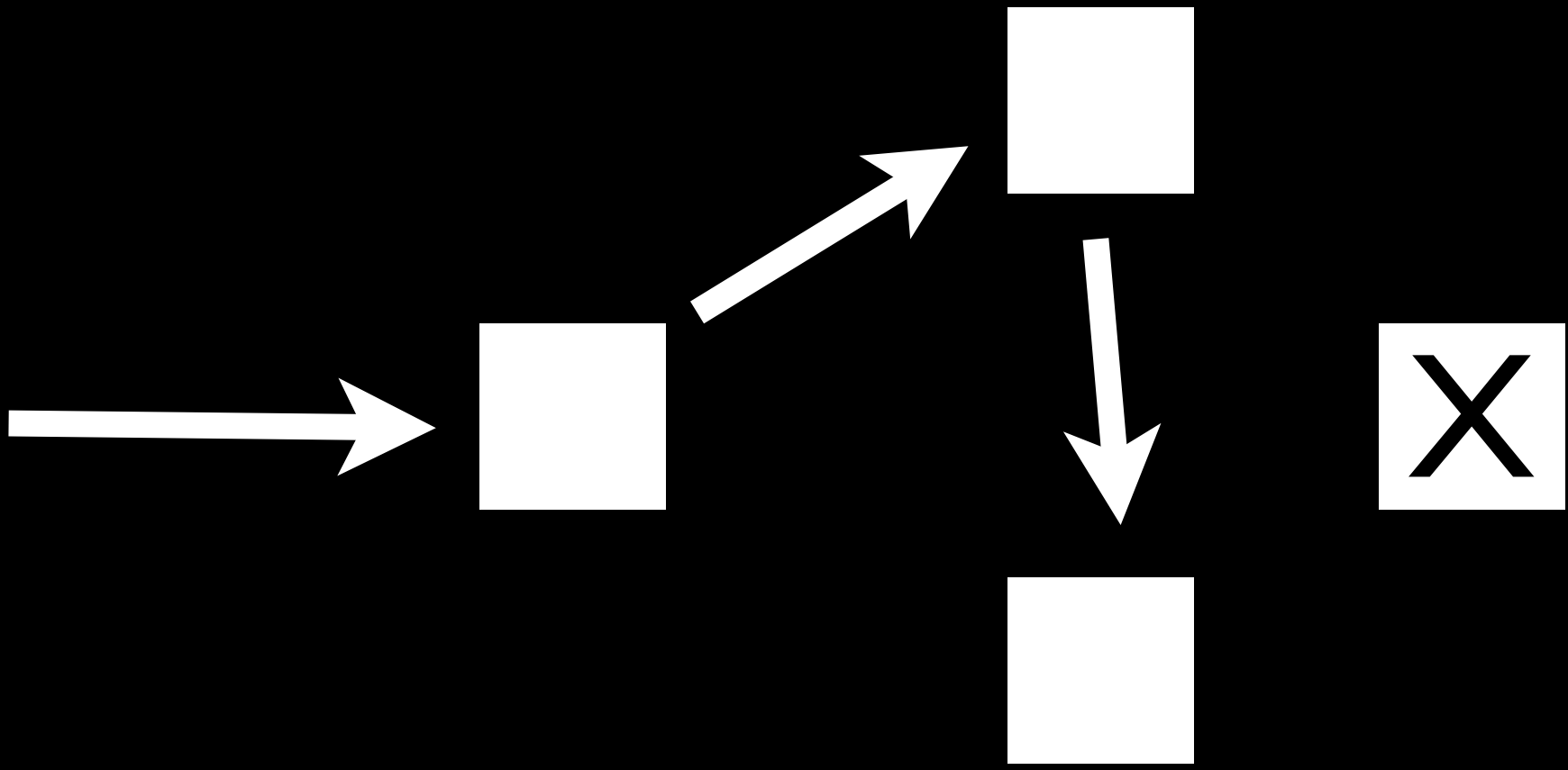


Garbage Collection 792

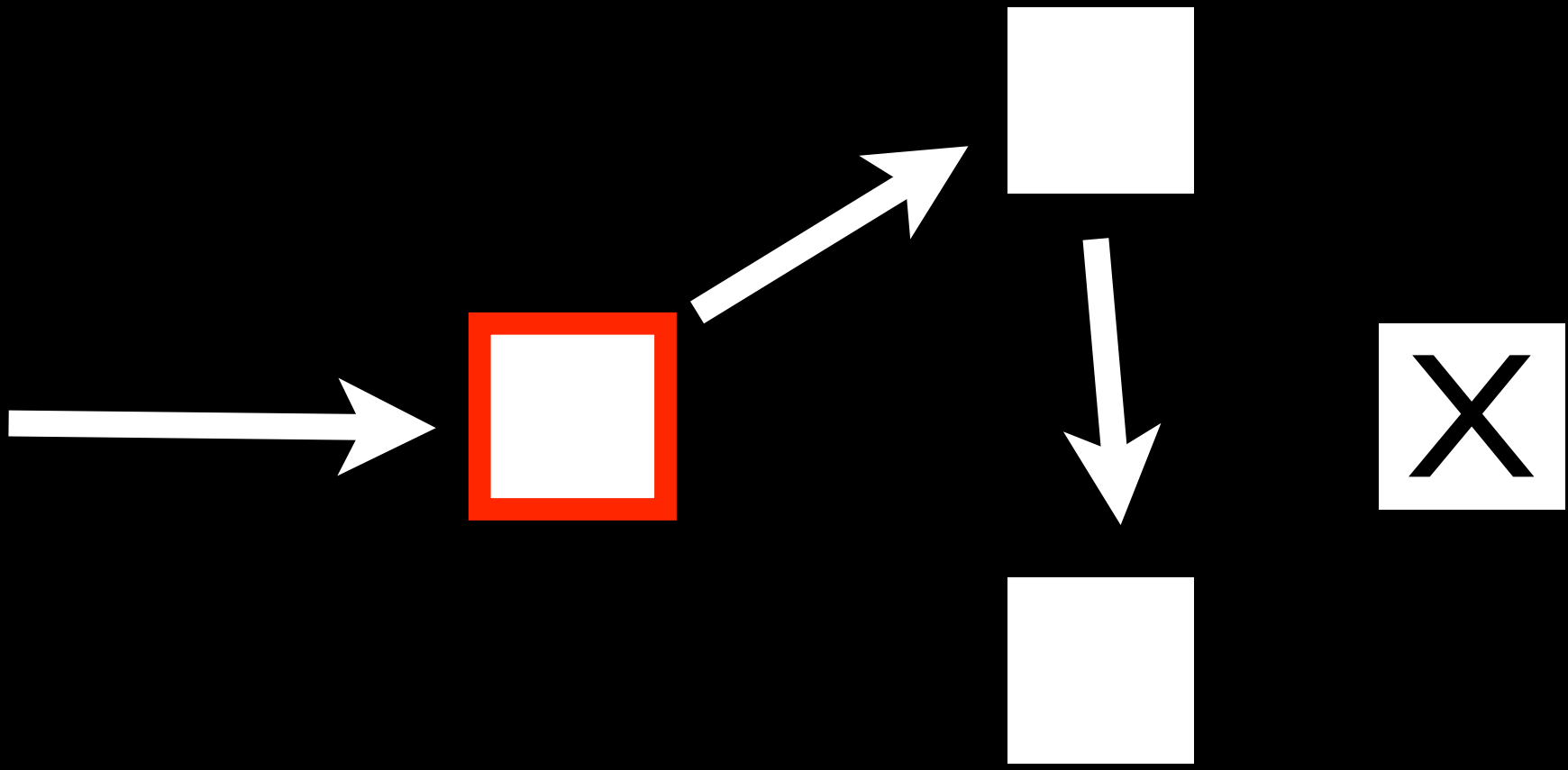
®

High performance
concurrent
Java garbage collector

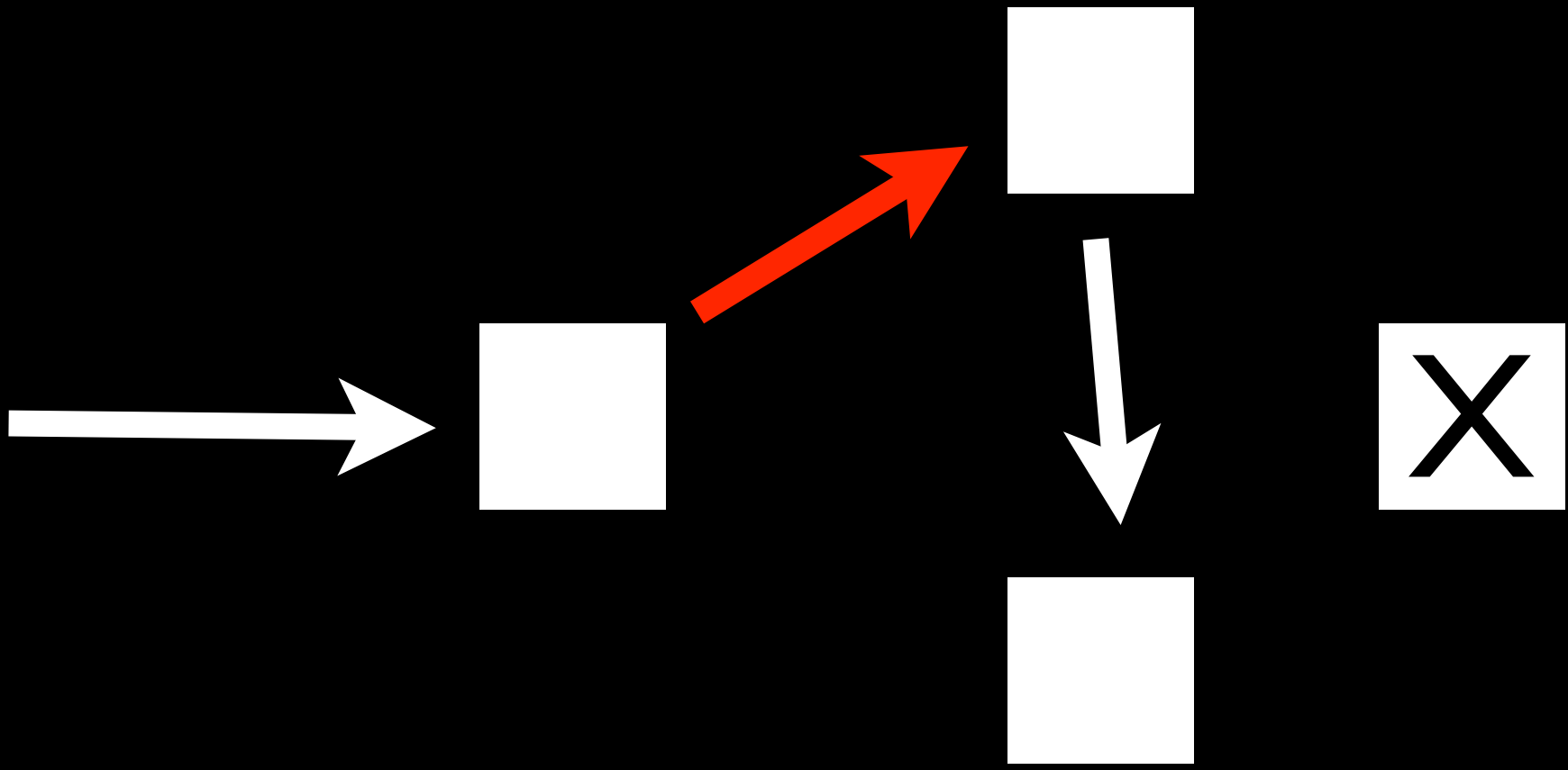
Garbage Collection 792



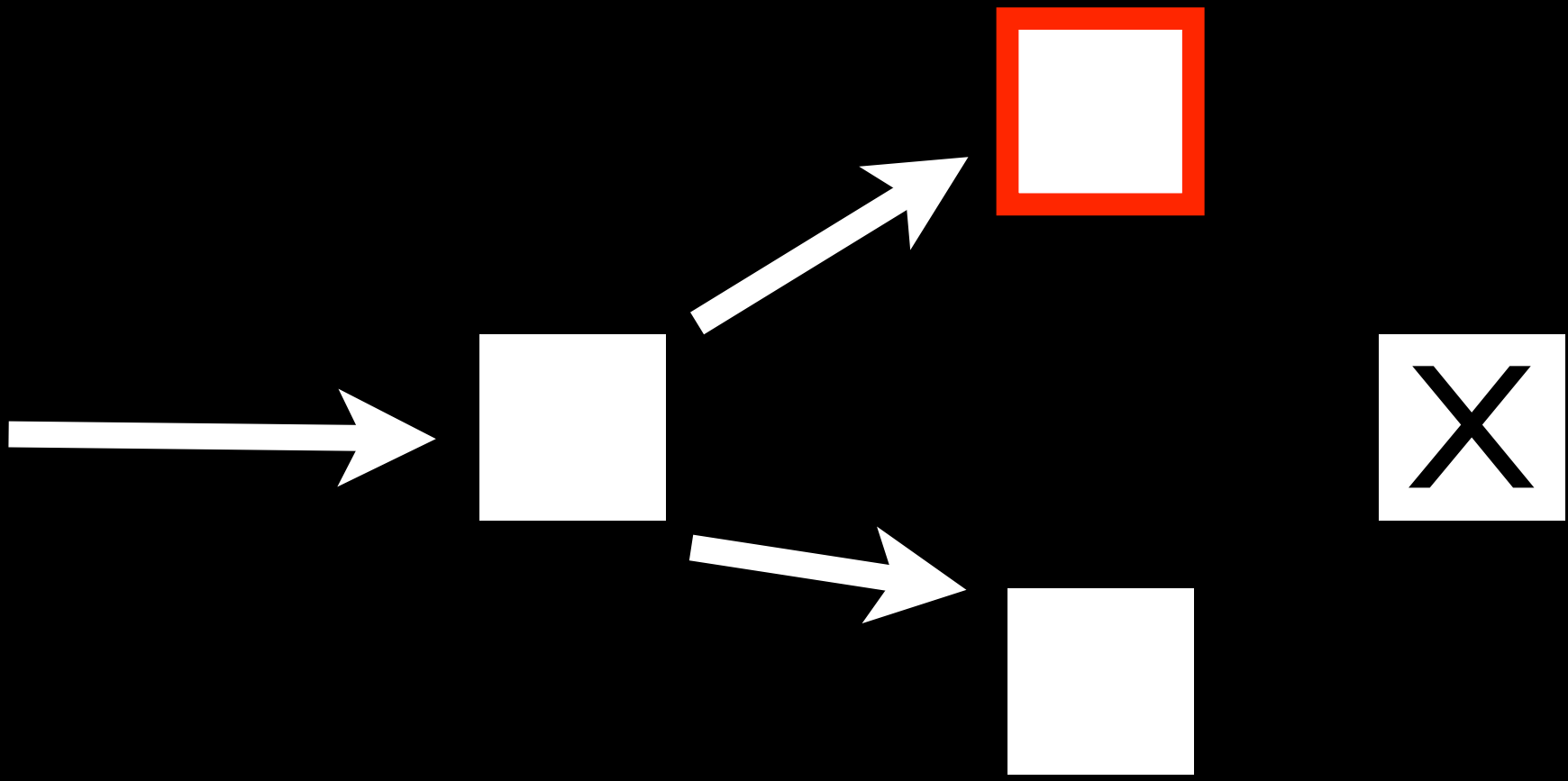
Garbage Collection 792



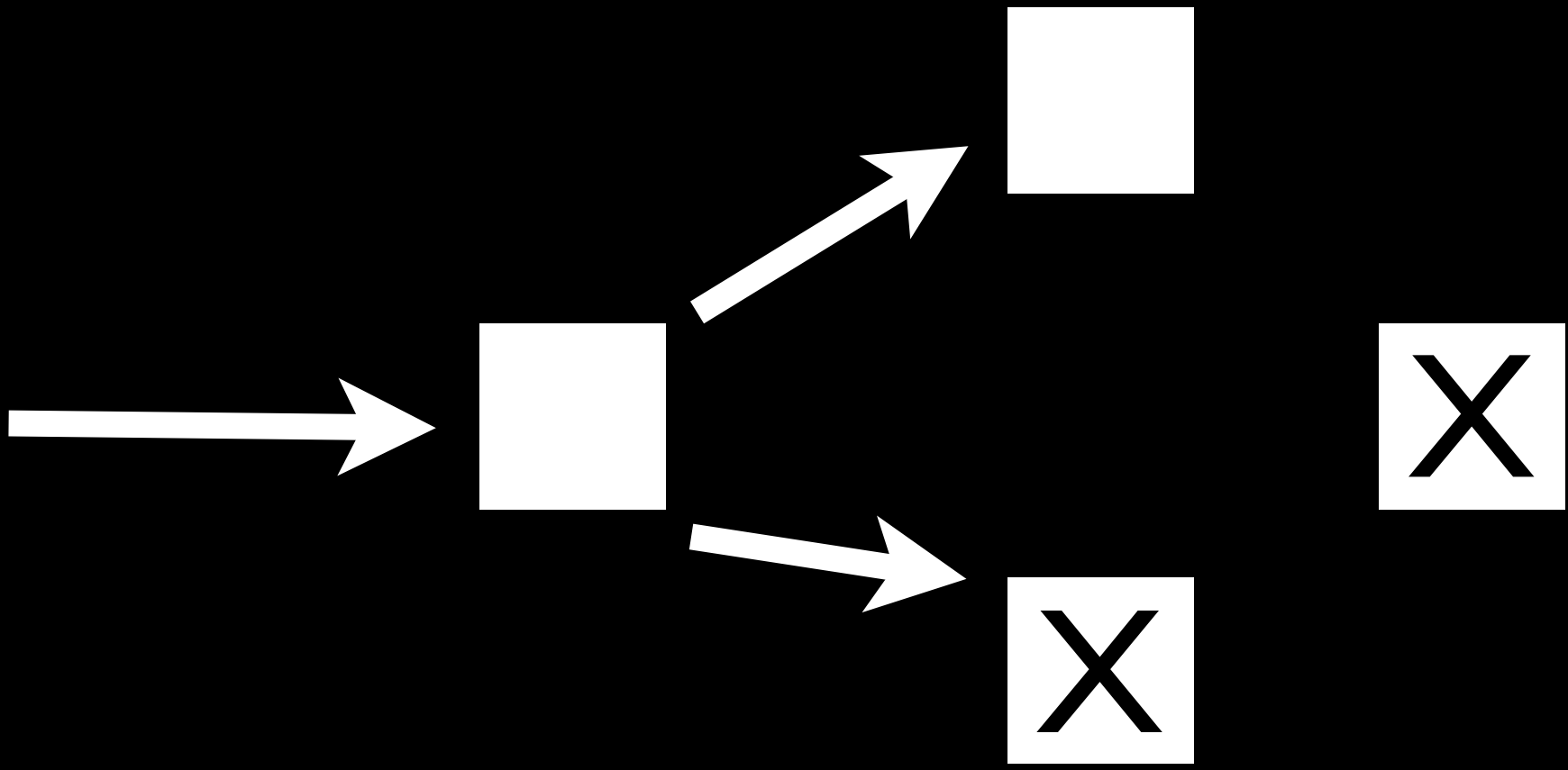
Garbage Collection 792



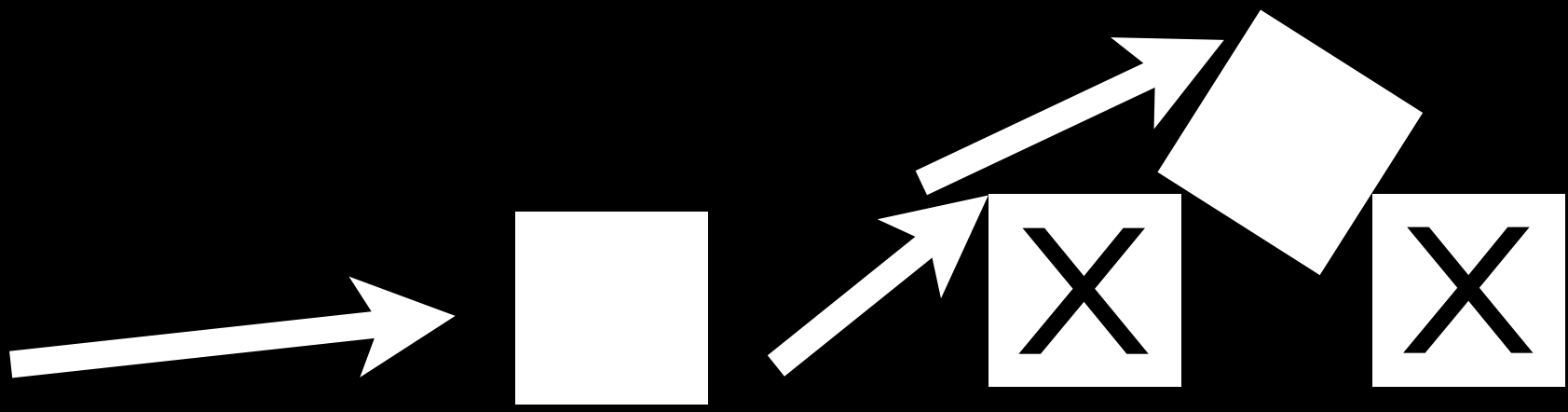
Garbage Collection 792



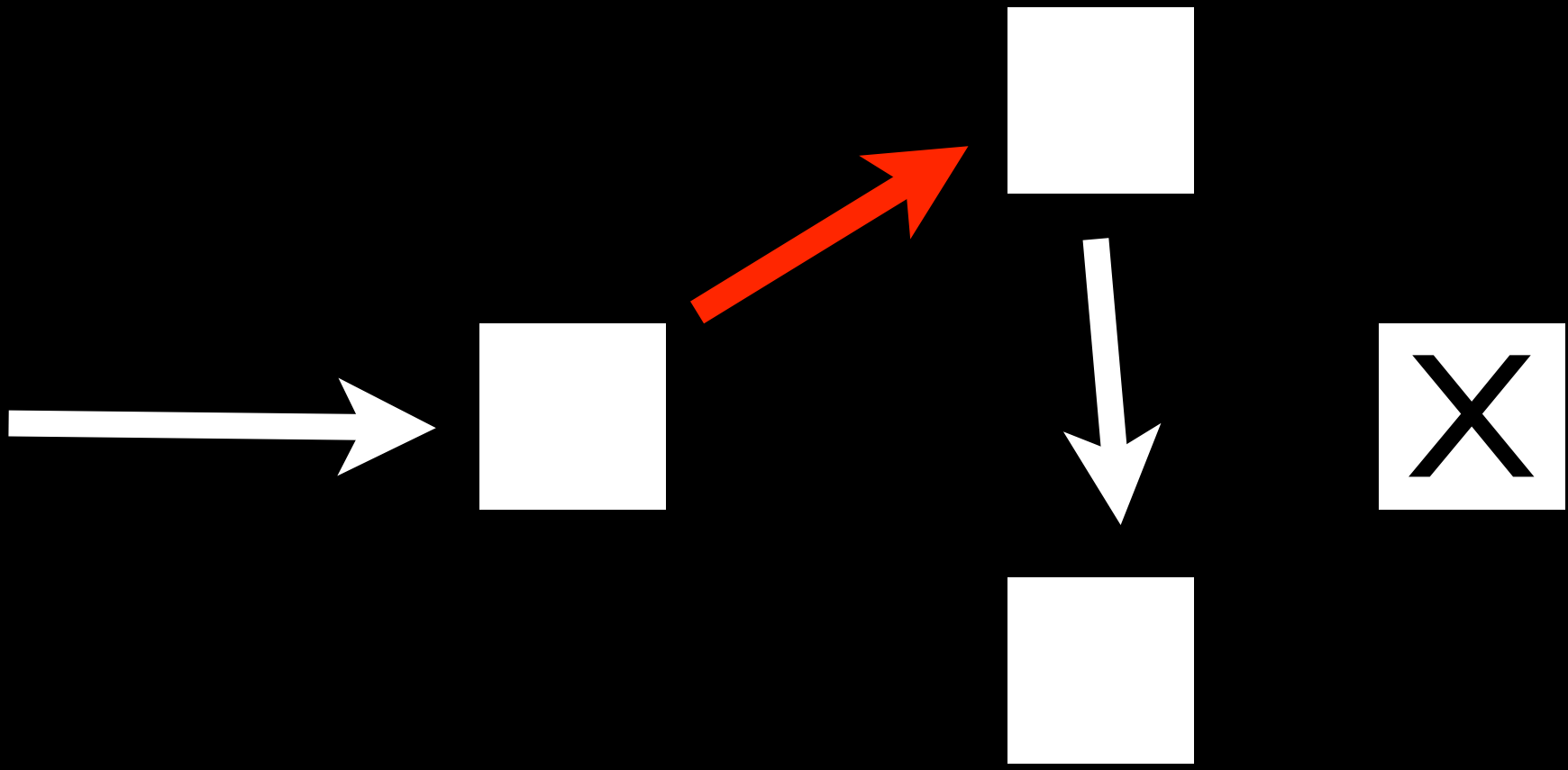
Garbage Collection 792



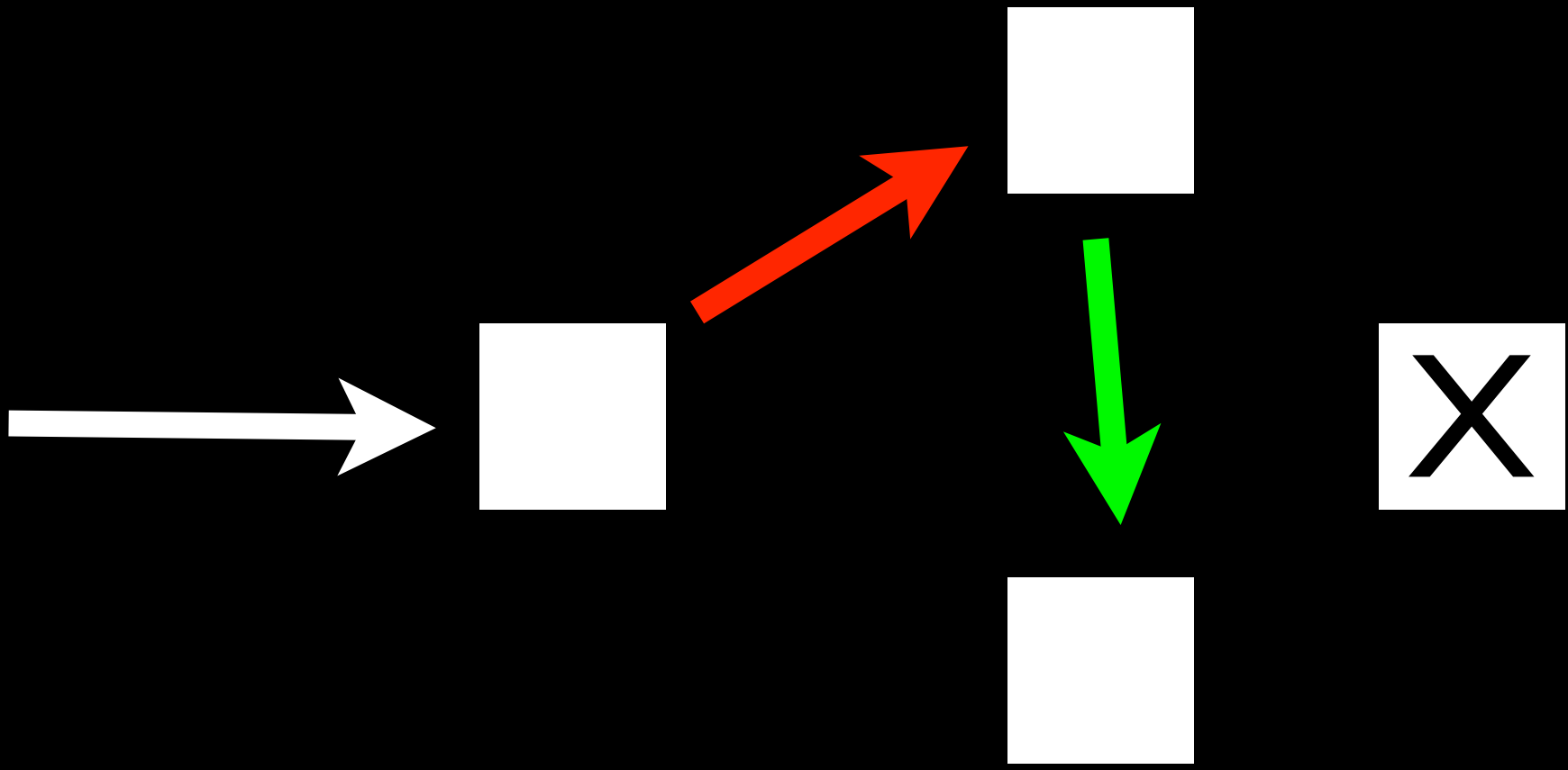
Garbage Collection 792



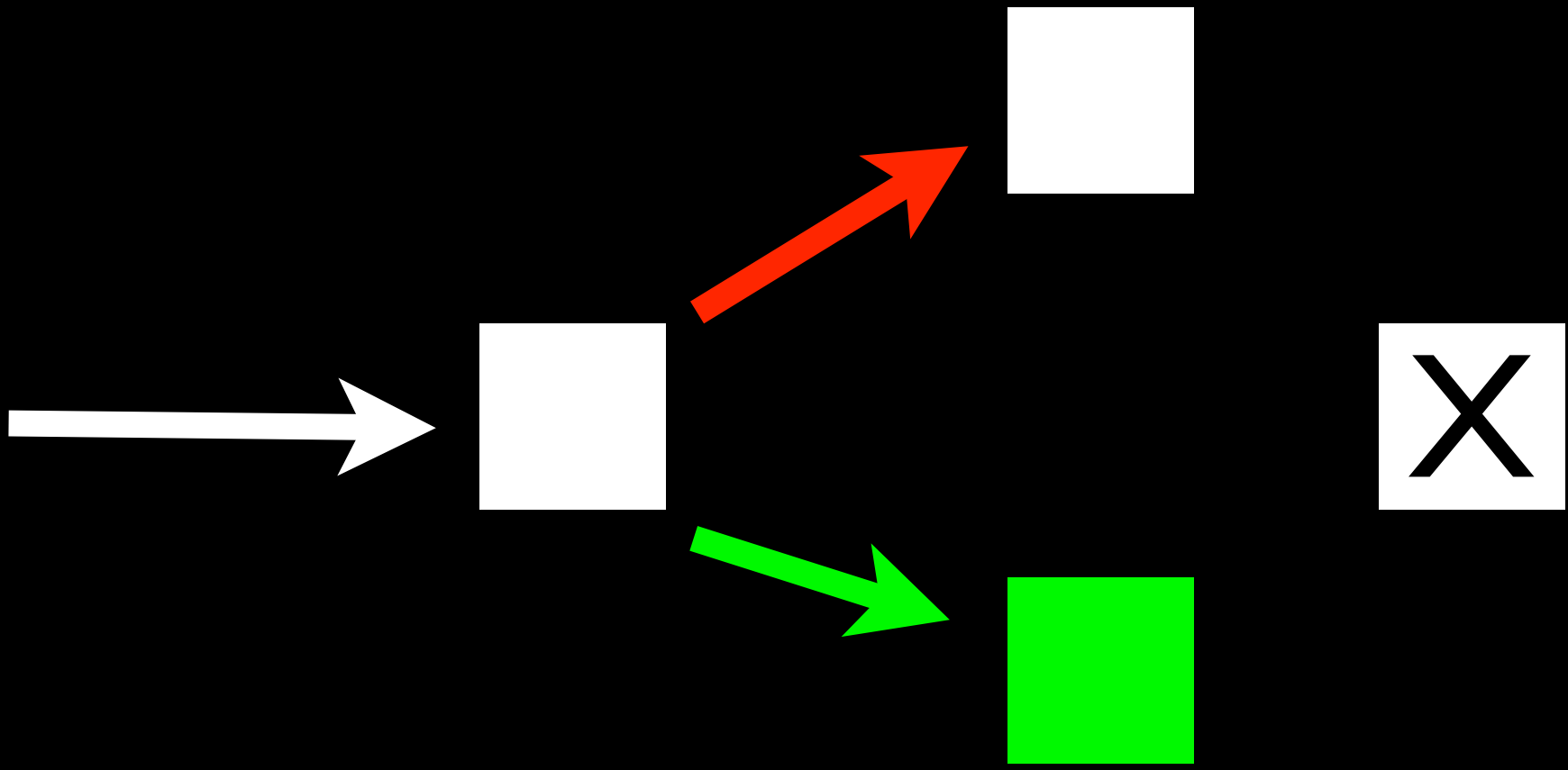
Garbage Collection 792



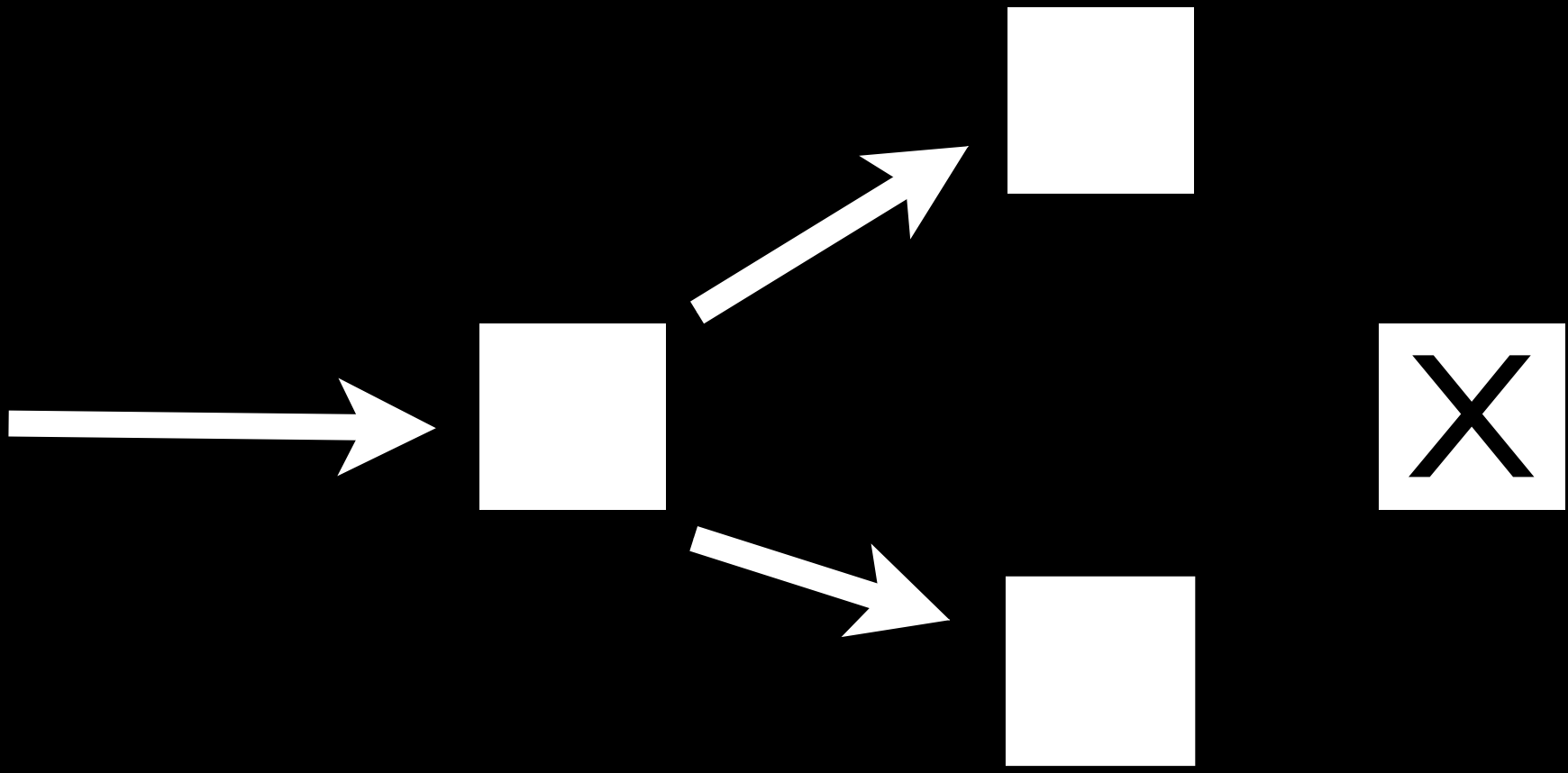
Garbage Collection 792



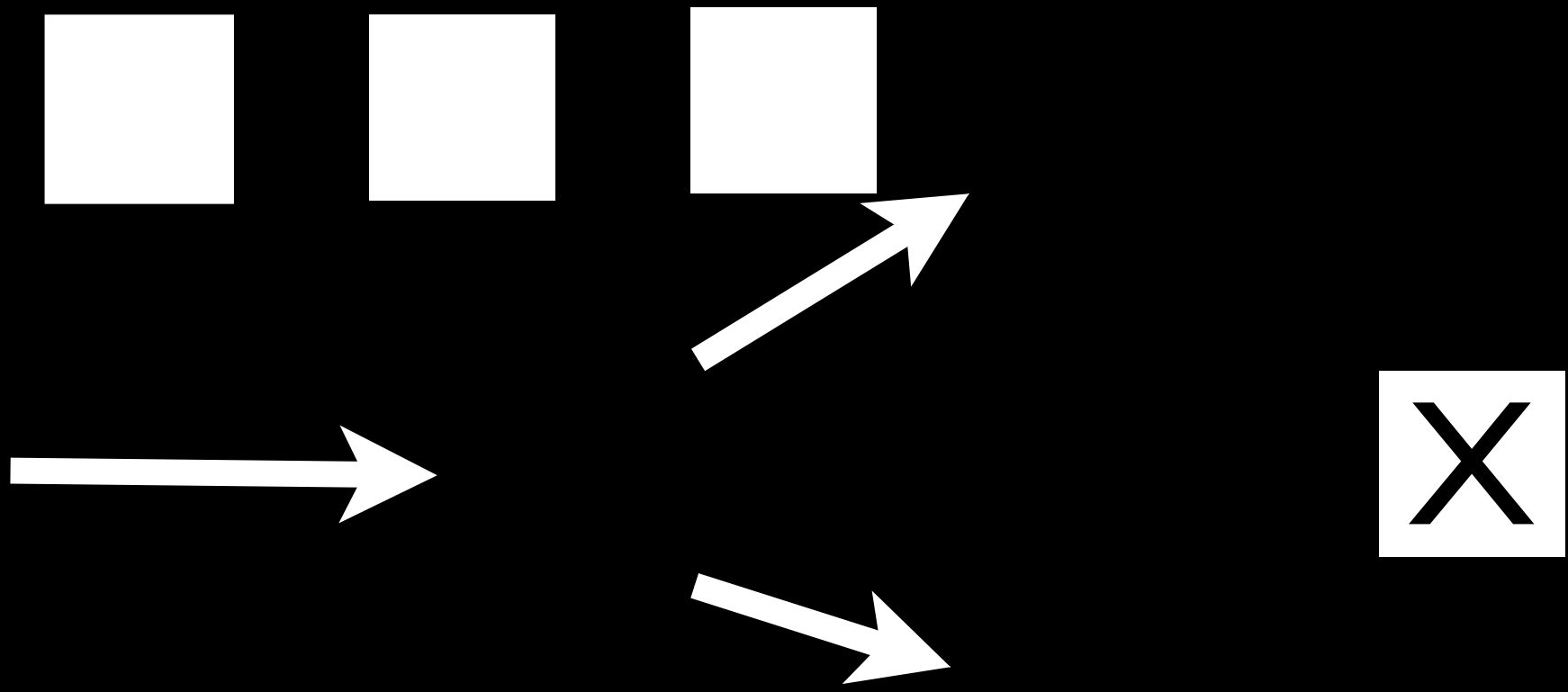
Garbage Collection 792



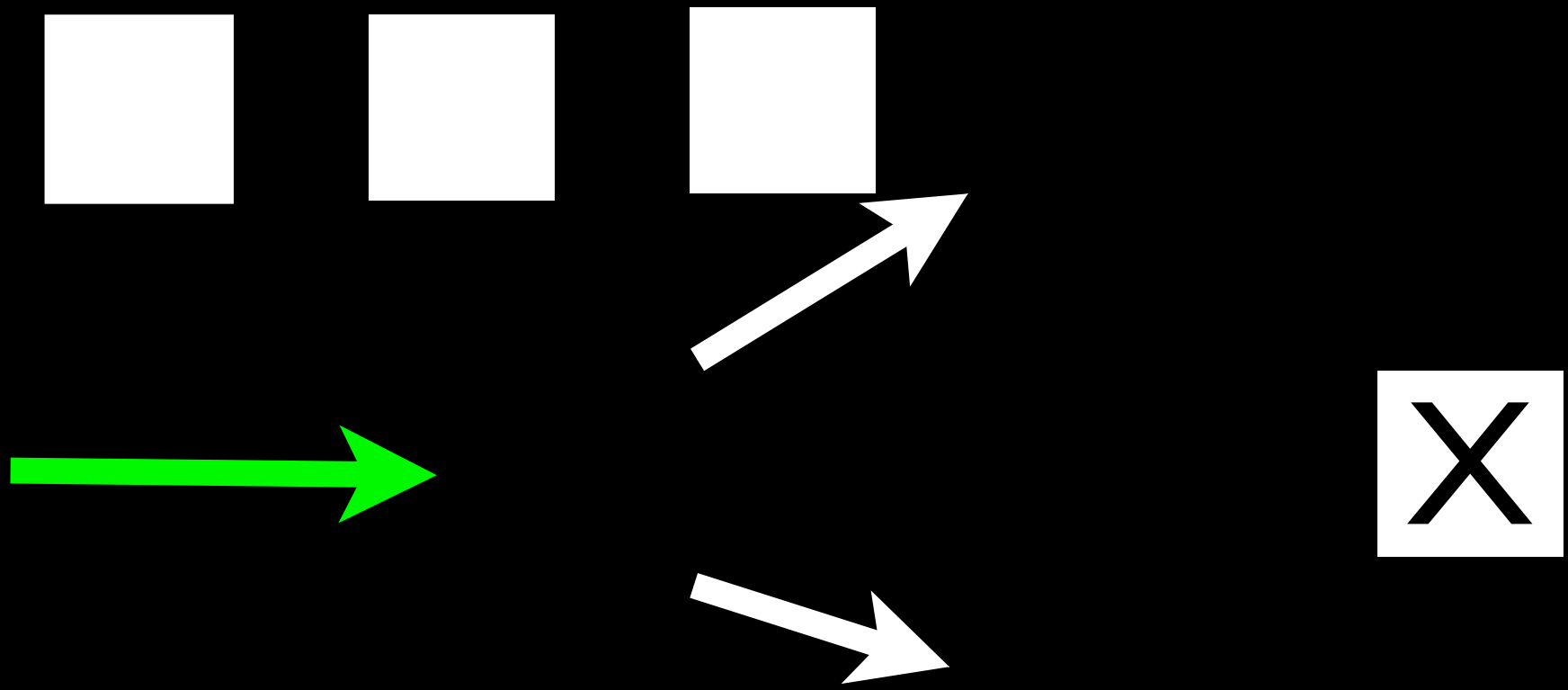
Garbage Collection 792



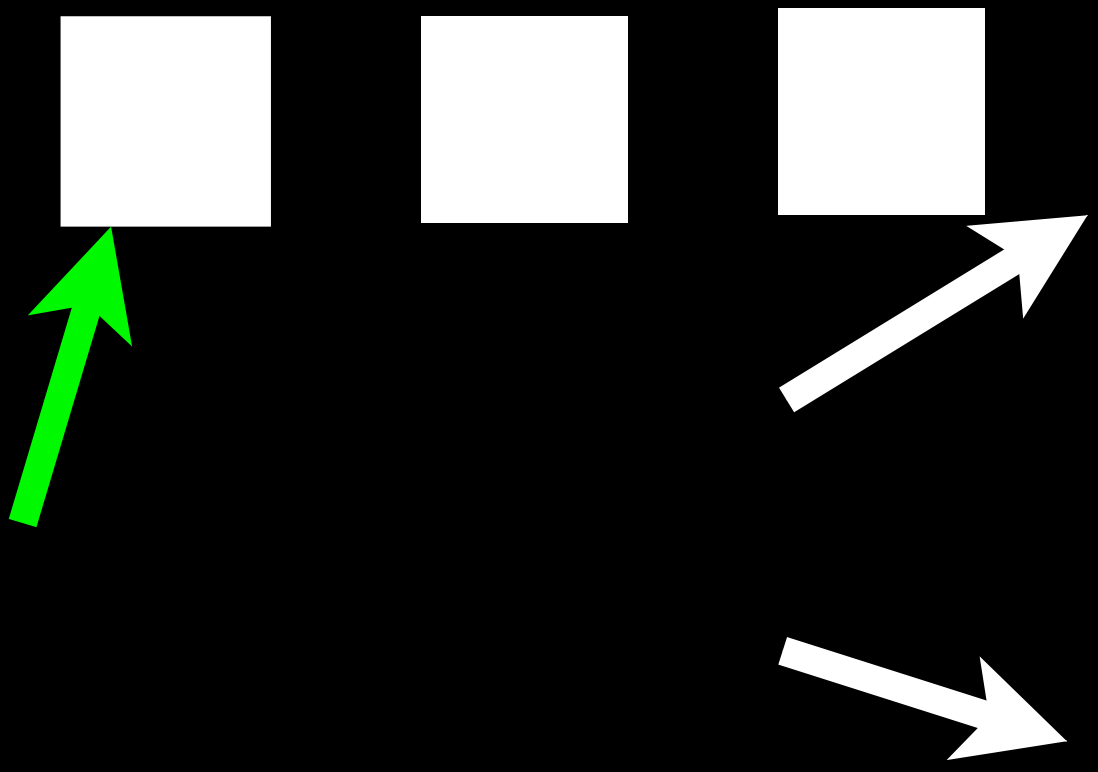
Garbage Collection 792



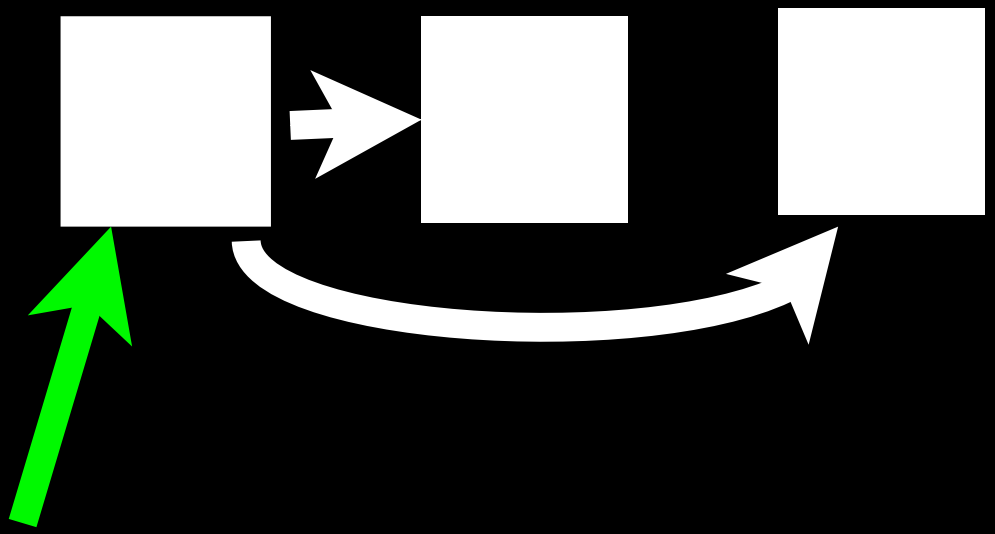
Garbage Collection 792



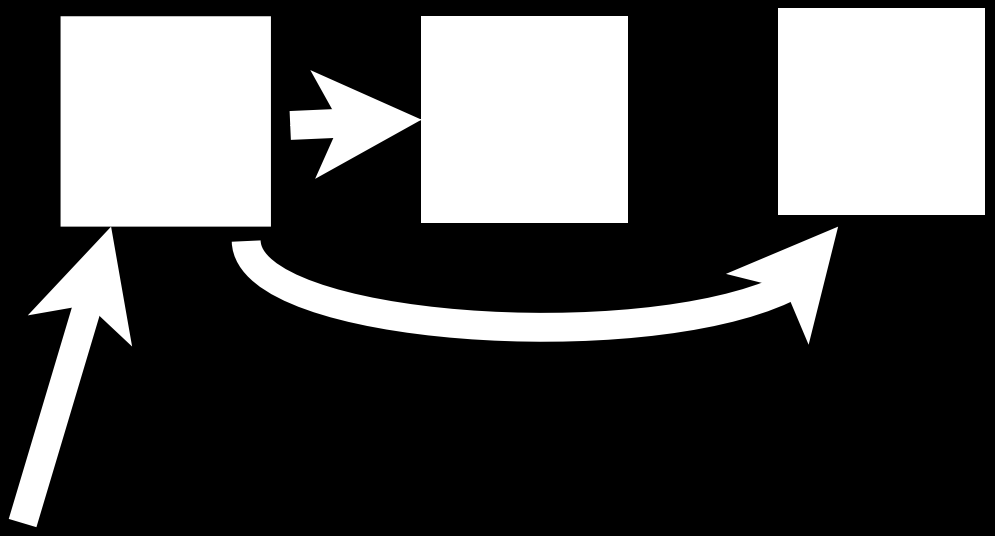
Garbage Collection 792



Garbage Collection 792



Garbage Collection 792

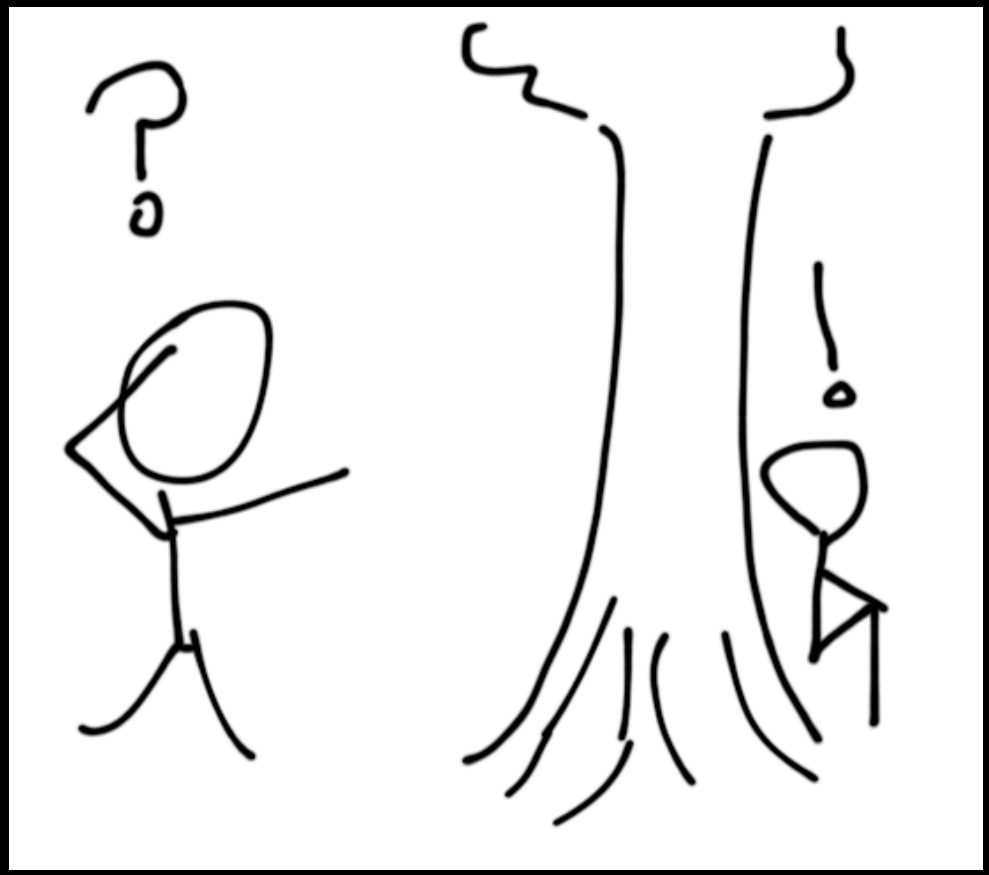


C vs GC

C vs GC

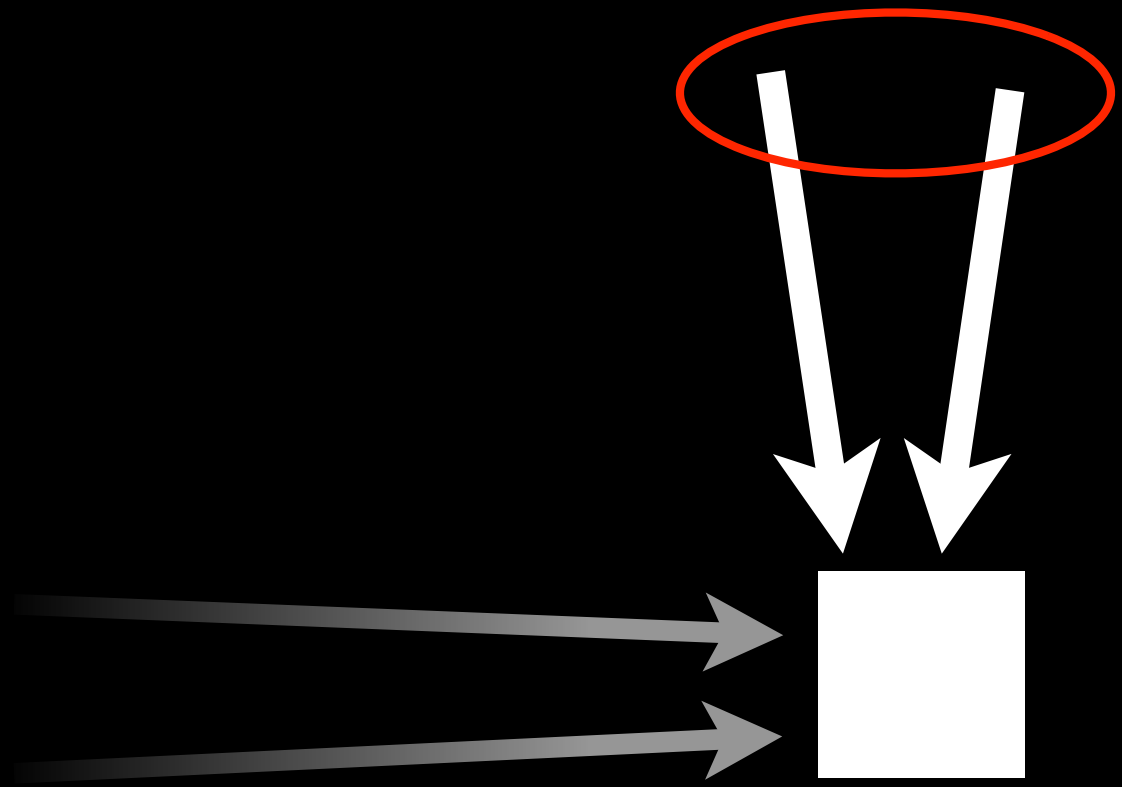
FIGHT!

Pointer Hide and Seek

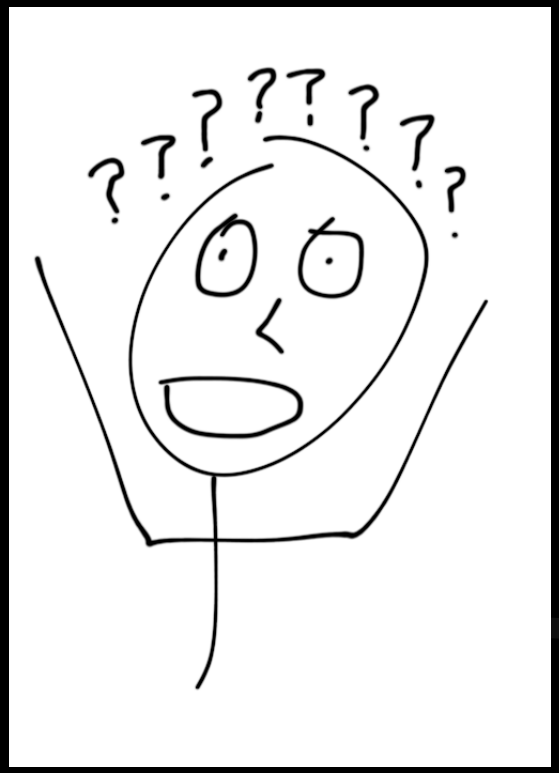


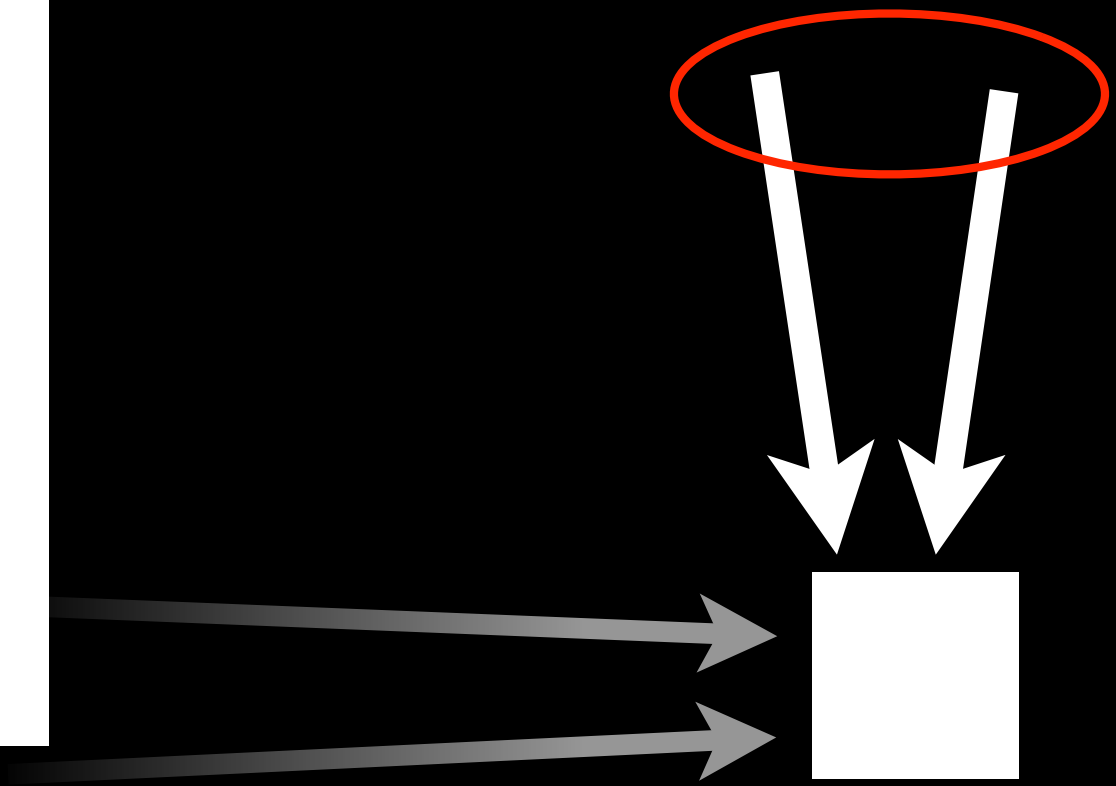
Apple did it anyway!

ObjC GC Limitations



ObjC GC Limitations



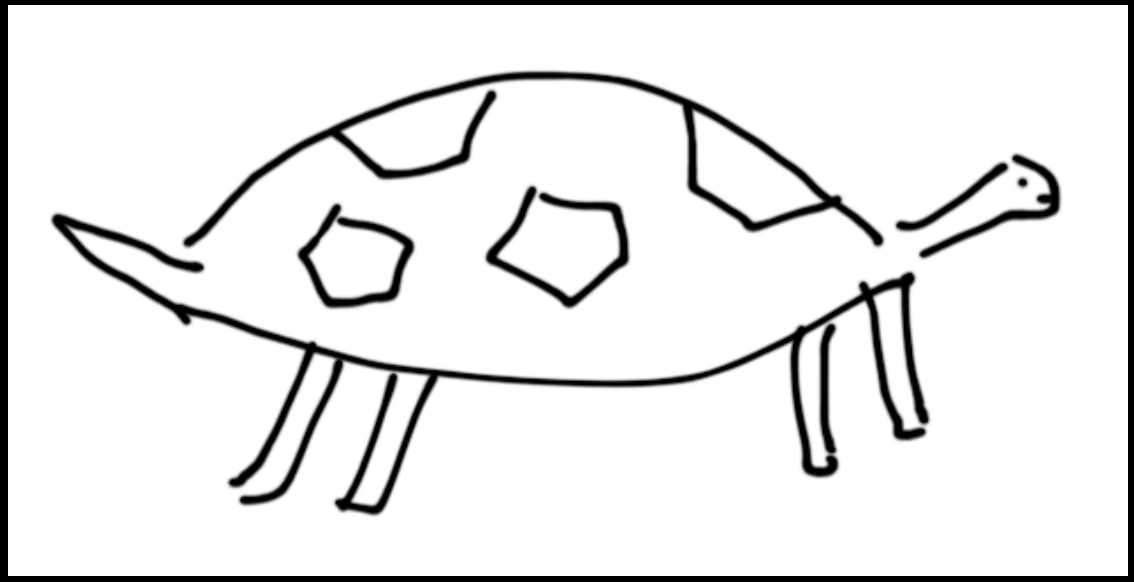


ObjC GC Limitations

```
NSData *data = [obj someData];
const char *bytes = [data bytes];

NSUInteger len = [data length];
for(NSUInteger i = 0; i < len; i++)
    printf(“%x”, bytes[i]);
```

ObjC GC Limitations

ObjC GC Limitations



vs

GC

FIGHT!

vs

FIGHT!

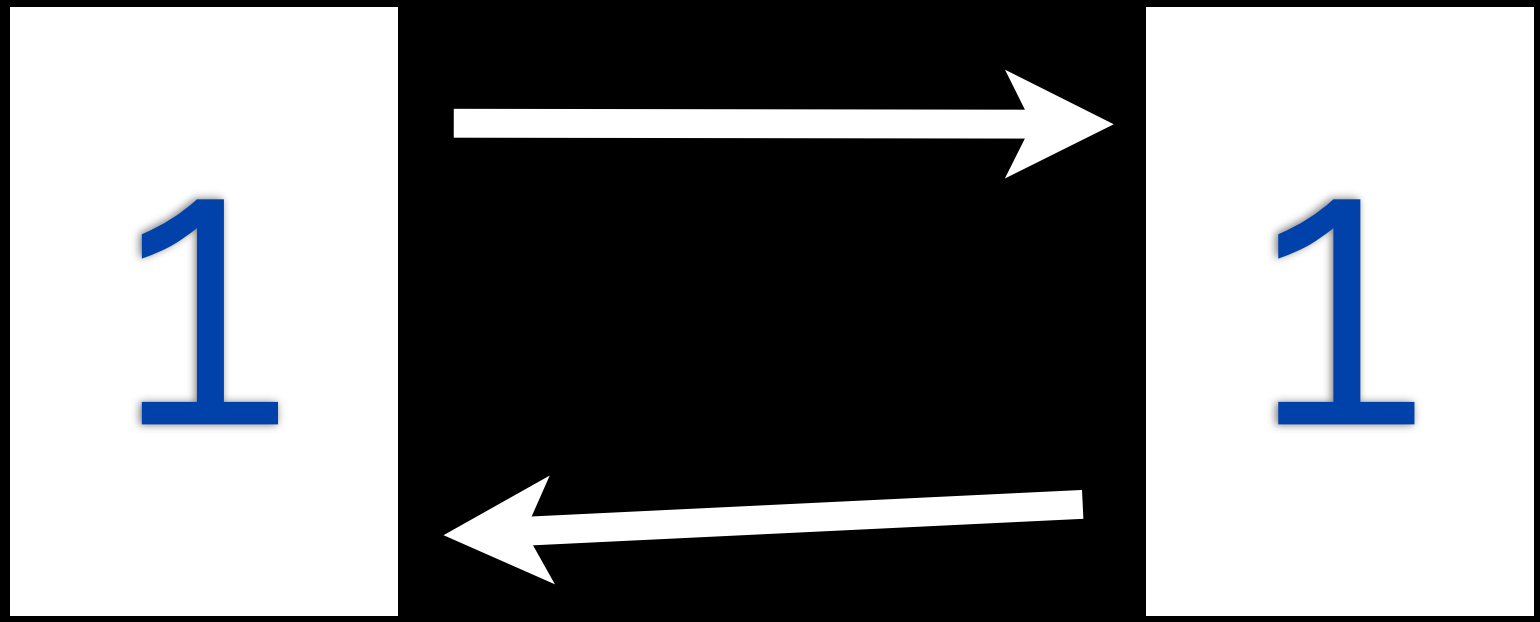
C

Automatic Reference Counting

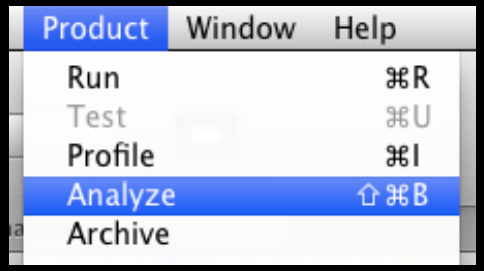
One day a student came to Moon and said: "I understand how to make a better garbage collector. We must keep a reference count of the pointers to each cons."

Moon patiently told the student the following story: "One day a student came to Moon and said: 'I understand how to make a better garbage collector...

Automatic Reference Counting



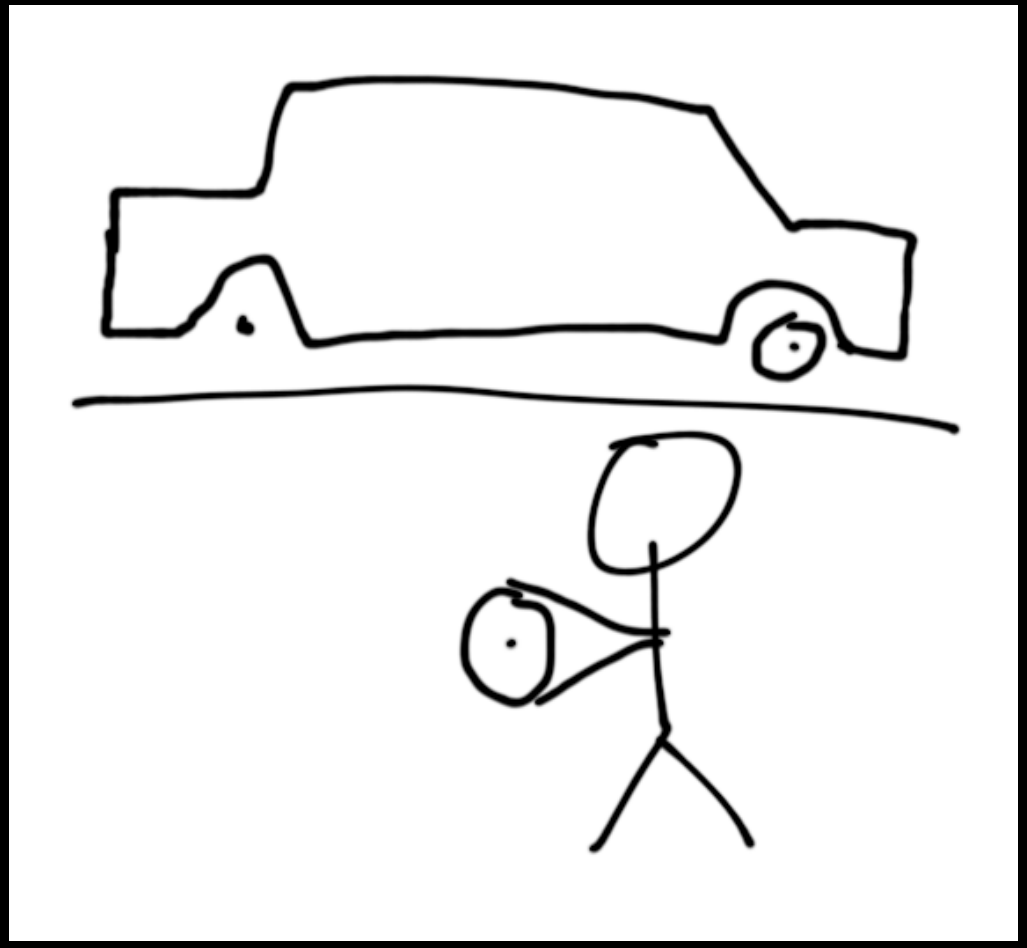
Static Analysis




```
- (NSString *)someString
{
    return [[NSString alloc] initWithFormat: @"%p", self];
}
```

1. Method returns an Objective-C object with a +1 retain count

Analysis + Auto-Fix + Goodies



System Requirements

- iOS5 and Mac OS X 10.7
- iOS4 and 10.6, partial (10.6 only with 64-bit)

Converting to ARC

```
- (NSArray *)array
{
    NSArray *strings;

    NSMutableArray *array = [[NSMutableArray alloc] init];
    for(NSString *string in strings)
    {
        MyClass *obj = [[MyClass alloc] initWithString: string];
        [array addObject: obj];
        [obj release];          'release' is unavailable: not available in automatic reference counting mode
    }
    return [array autorelease]; 'autorelease' is unavailable: not available in automatic reference counting mode
}
```

Converting to ARC

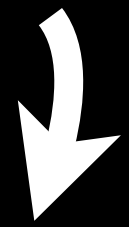
```
- (NSArray *)array
{
    NSArray *strings;

    NSMutableArray *array = [[NSMutableArray alloc] init];
    for(NSString *string in strings)
    {
        MyClass *obj = [[MyClass alloc] initWithString: string];
        [array addObject: obj];
    }
    return array;
}
```

Converting to ARC

```
NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
// code here
[pool release];
```

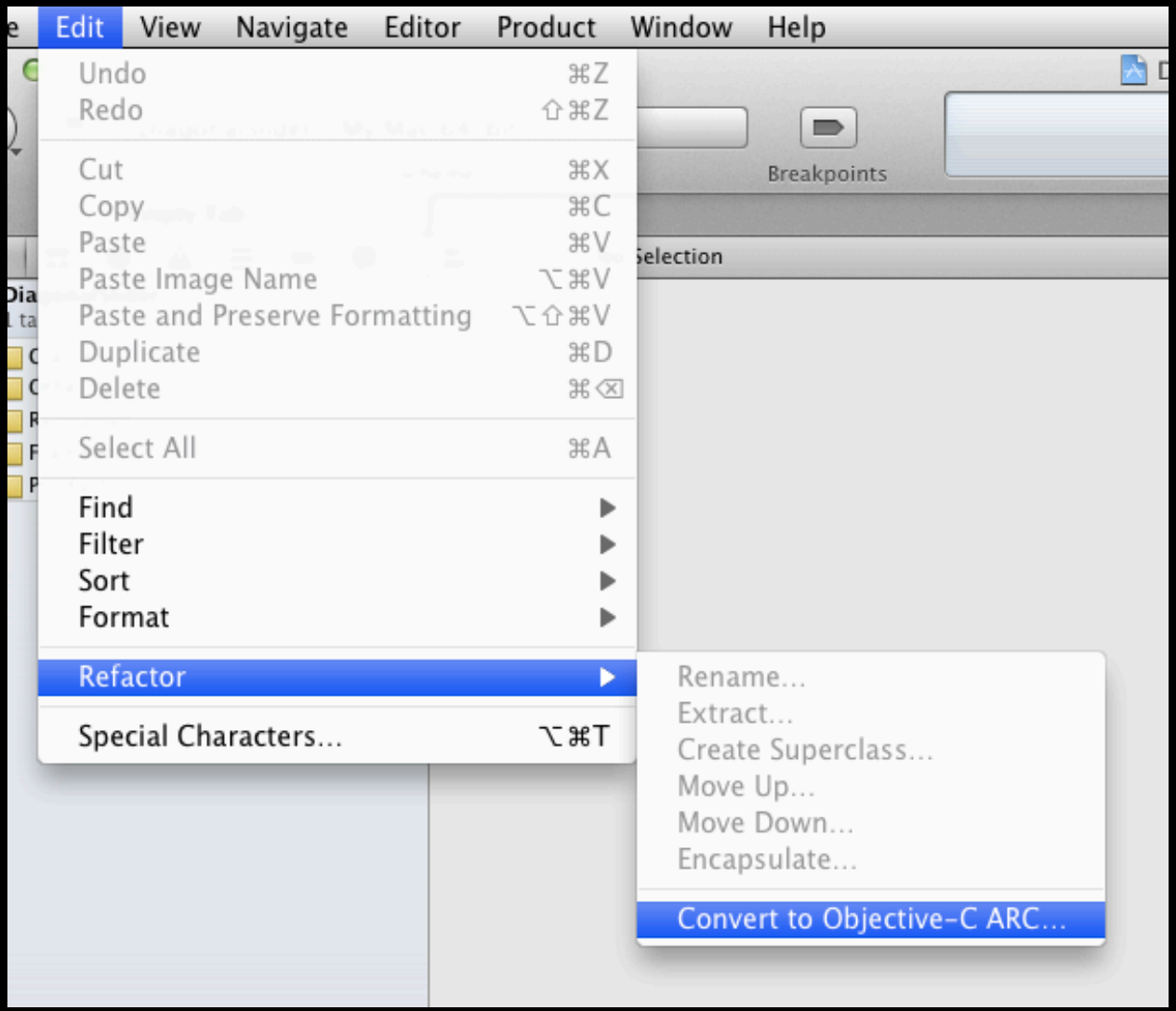
```
@autoreleasepool {
    // code here
}
```

```
NSString *s = [(id)CFStringCreateWhatever() autorelease];
```

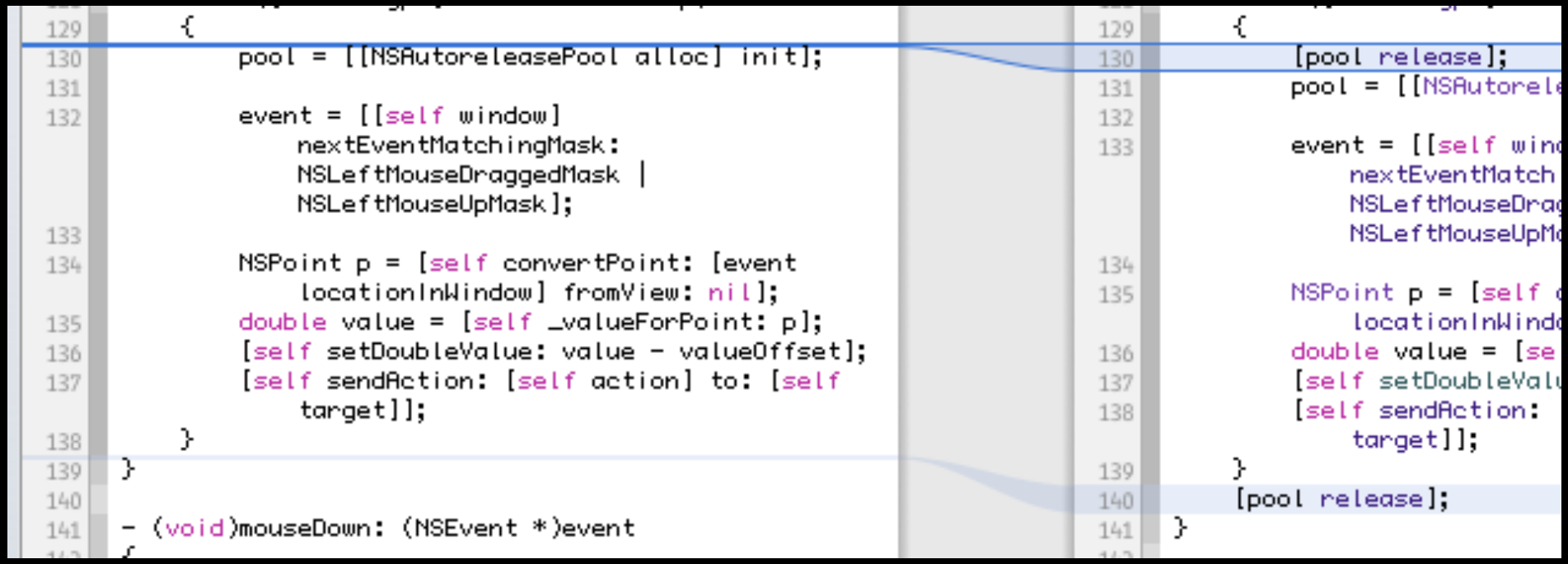


```
NSString *s = CFBridgingRelease(CFStringCreateWhatever());
```

Automatic Conversion



Automatic Conversion



```
129     {
130         pool = [[NSAutoreleasePool alloc] init];
131
132         event = [[self window]
                nextEventMatchingMask:
                NSLeftMouseDraggedMask |
                NSLeftMouseUpMask];
133
134         NSPoint p = [self convertPoint: [event
                locationInWindow] fromView: nil];
135         double value = [self _valueForPoint: p];
136         [self setDoubleValue: value - valueOffset];
137         [self sendAction: [self action] to: [self
                target]];
138     }
139 }
140
141 - (void)mouseDown: (NSEvent *)event
```

```
129     {
130         [pool release];
131         pool = [[NSAutorele
132
133         event = [[self win
                nextEventMatch
                NSLeftMouseDrag
                NSLeftMouseUpM
134
135         NSPoint p = [self
                locationInWindo
136         double value = [se
137         [self setDoubleValu
138         [self sendAction:
                target]];
139     }
140     [pool release];
141 }
```

Not Perfect!

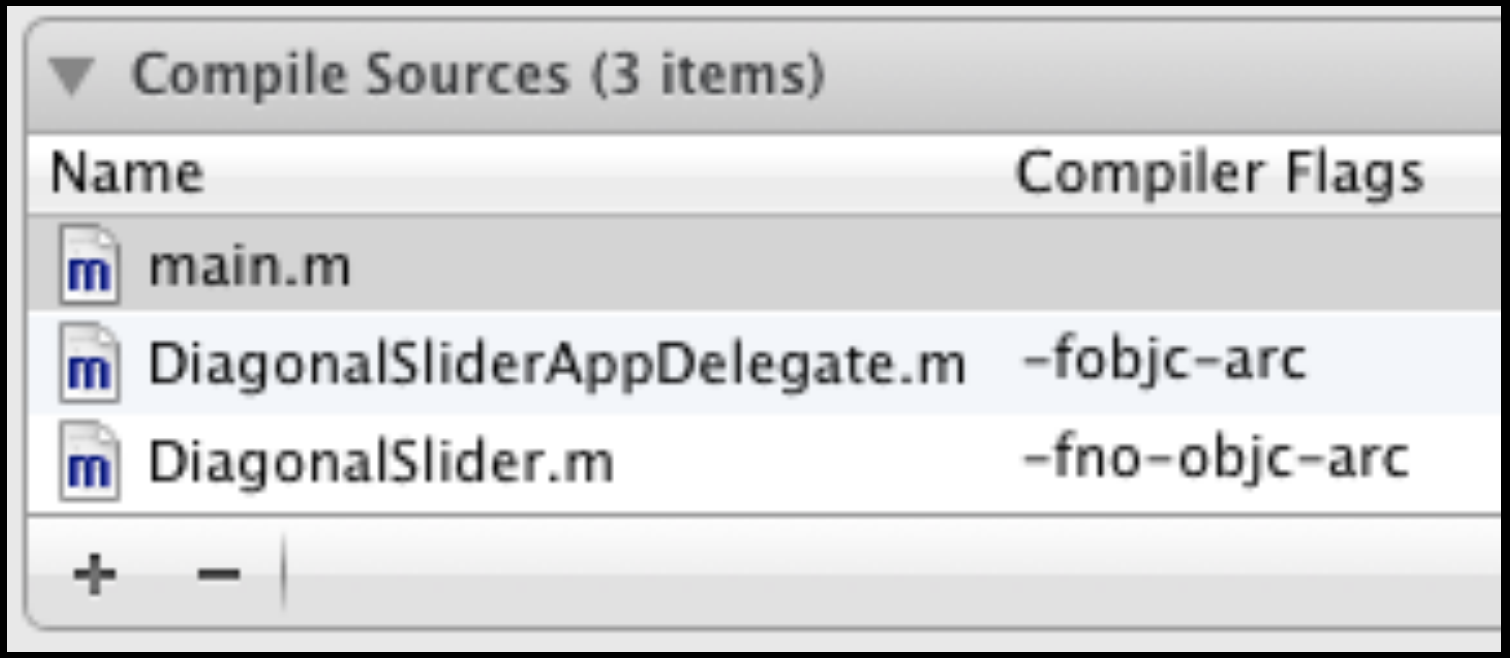
Weird autorelease pools

Toll-free bridging

Structs

Stupid Pointer Tricks

Partial Migration



How ARC Works

1. If you alloc, copy, retain, new:
   YOU MUST release or autorelease

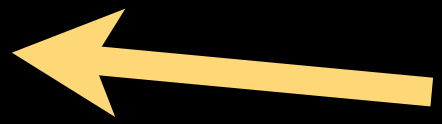
2. If you store in an ivar:
   YOU MUST alloc, copy, retain, or new

3. If you return a value:
   USE autorelease to pass to the caller

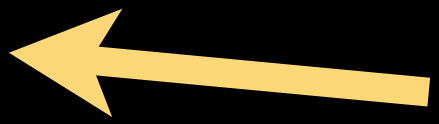
4. If you have ivars
   YOU MUST implement dealloc to release them

```
Foo *obj = [[Foo alloc] init];
[foo bar];
```

```
Foo *obj = [[Foo alloc] init];
[foo bar];
[foo release];
```



```
Foo *obj = [[Foo alloc] init];
[foo bar];
objc_release(foo);
```



retain retain release
release
retain
retain release
retain
release
release retain retain
release release
retain retain

```
Foo *foo = [self foo];
[foo bar];
[self setFoo: newFoo];
[foo baz];
```

CRASH

```
Foo *foo = objc_retain([self foo]);
[foo bar];
[self setFoo: newFoo];
[foo baz];
objc_release(foo);
```

```
- (Foo *)foo
{
    return _foo;
}
```

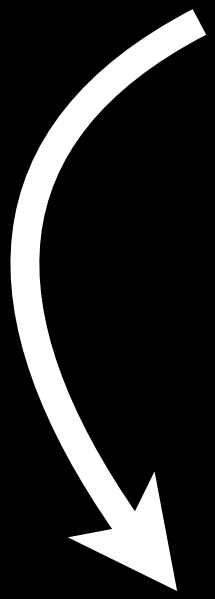
```
- (Foo *)foo
{
    return [[_foo retain] autorelease];
}
```

```
- (Foo *)foo
{
    return objc_retainAutoreleaseReturnValue(_foo);
}
```

```
retain
autorelease
retain
release
```

```
retain
autorelease
retain
release
```

objc_retainAutoreleaseReturnValue



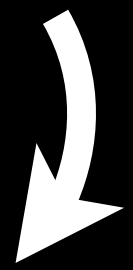
objc_retainAutoreleasedReturnValue

retain

release

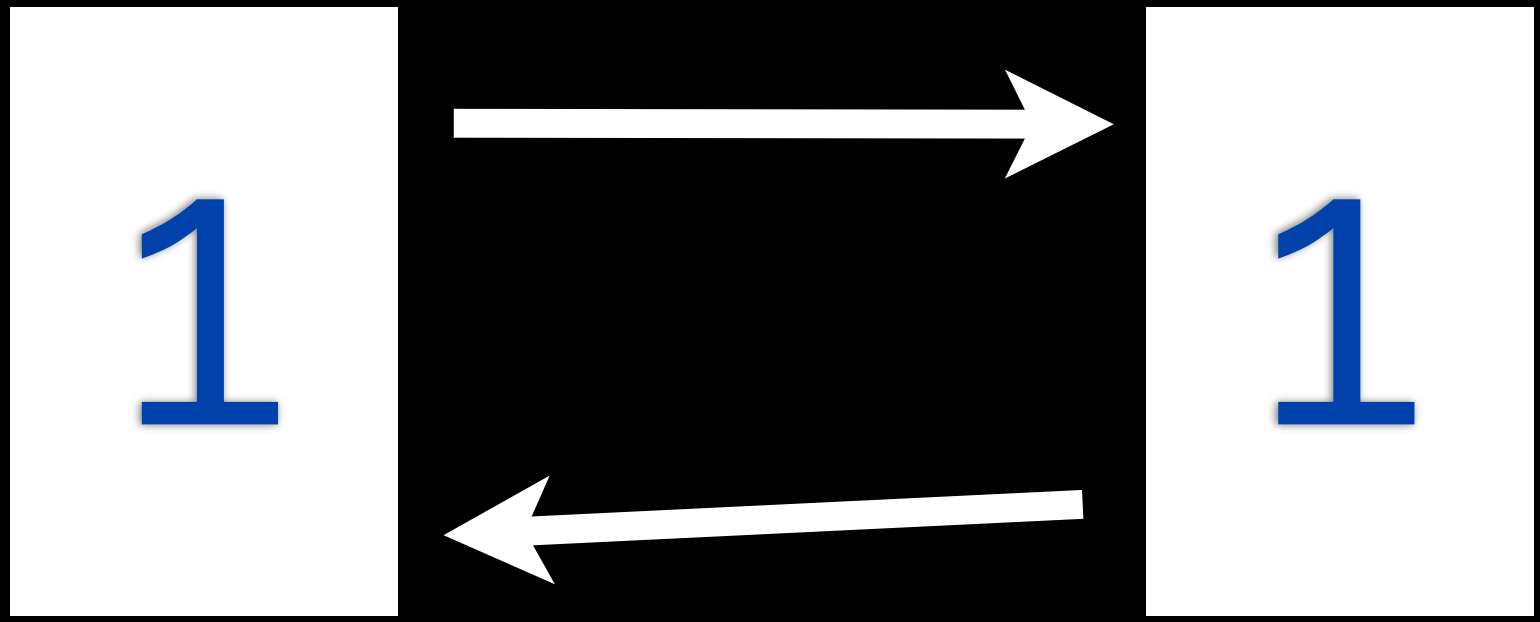
~~dealloc~~

```
- (void)dealloc
{
    [ivar1 release];
    [ivar2 release];
    free(buffer);

    [super dealloc];
}
```

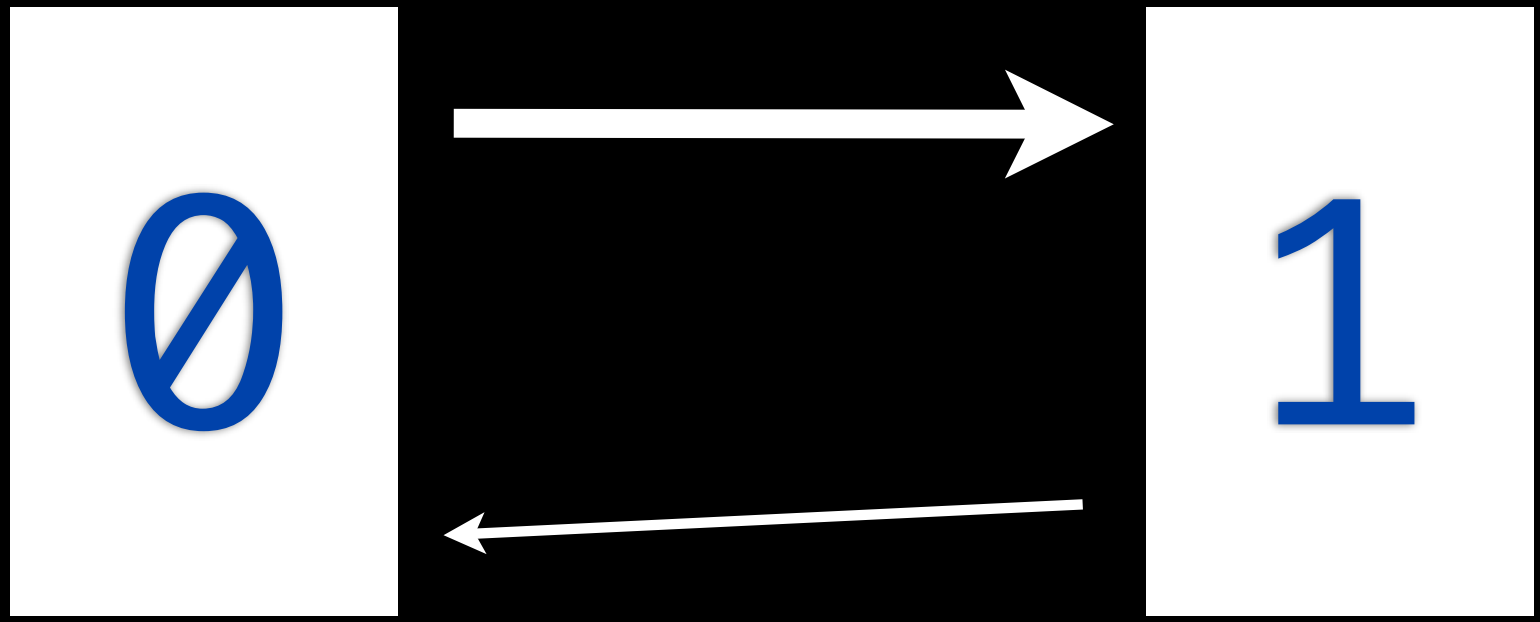


```
- (void)dealloc
{
    free(buffer);
}
```

Cycles

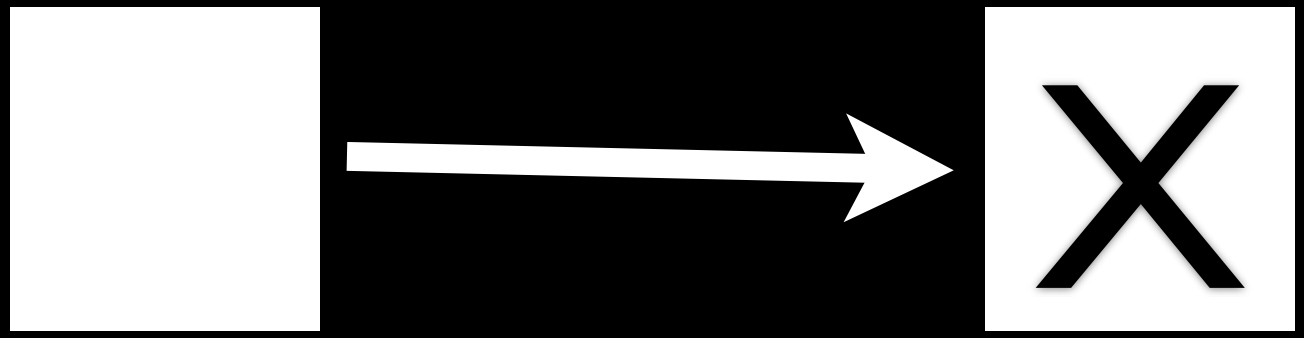


Cycles

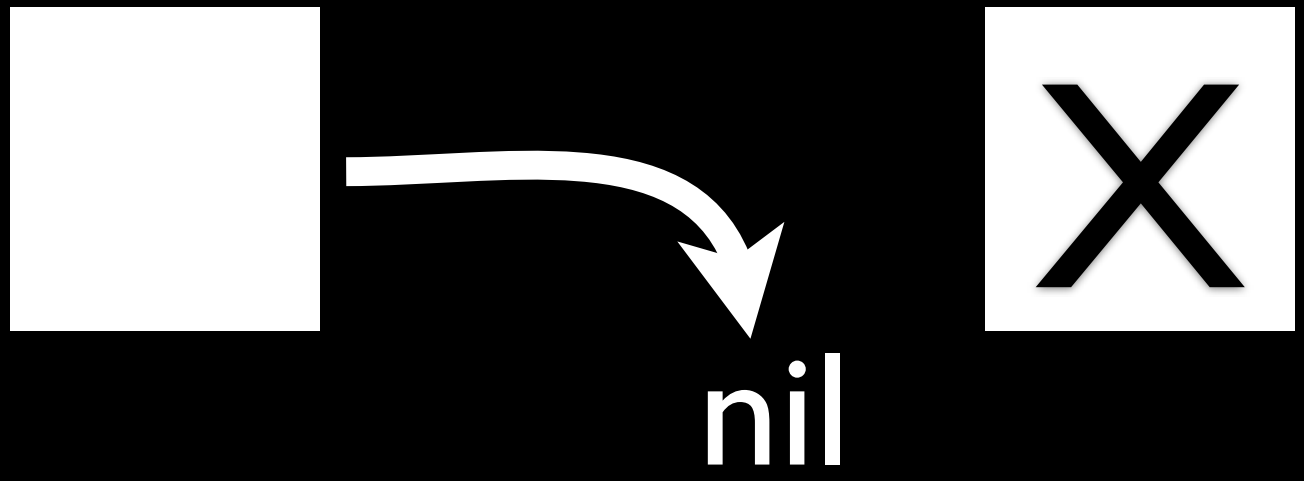


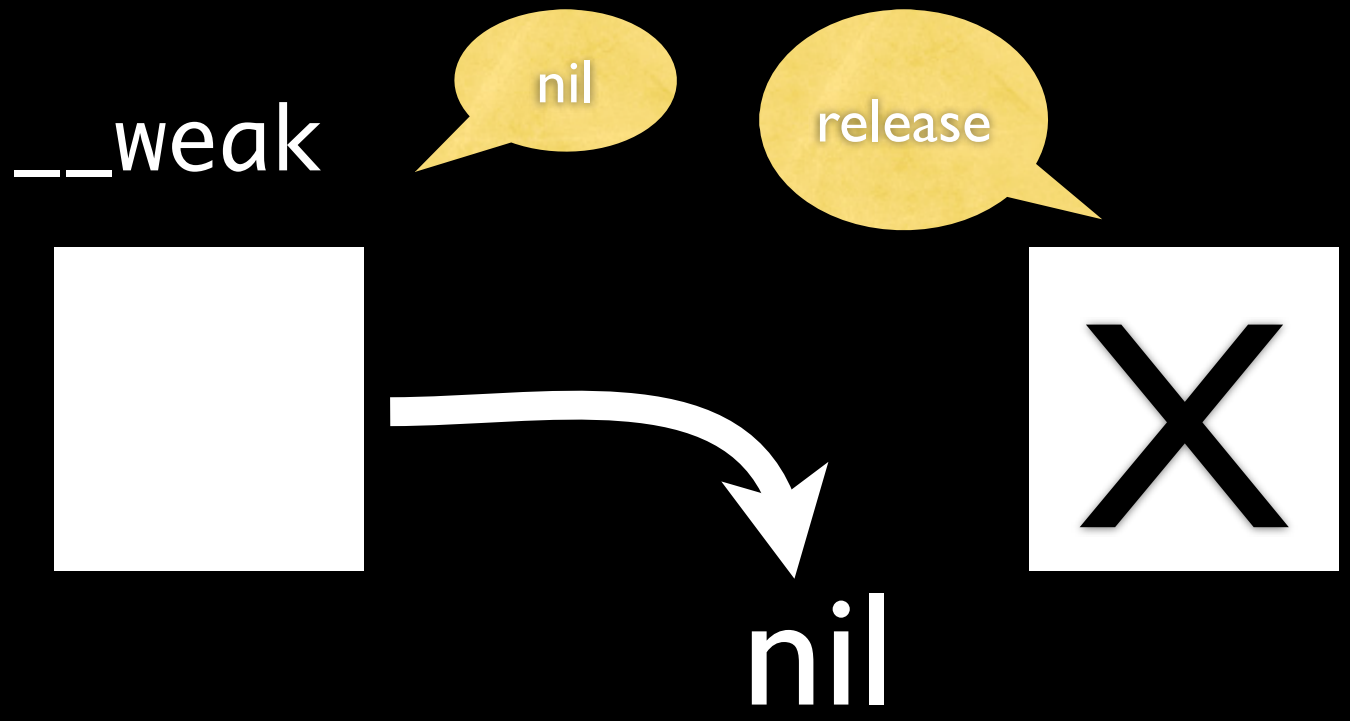
```
__weak id weakPointer;
```

Unsafe weak reference



__weak




__weak
nil
release
nil

Good news, everybody!

- Mostly a Mac thing

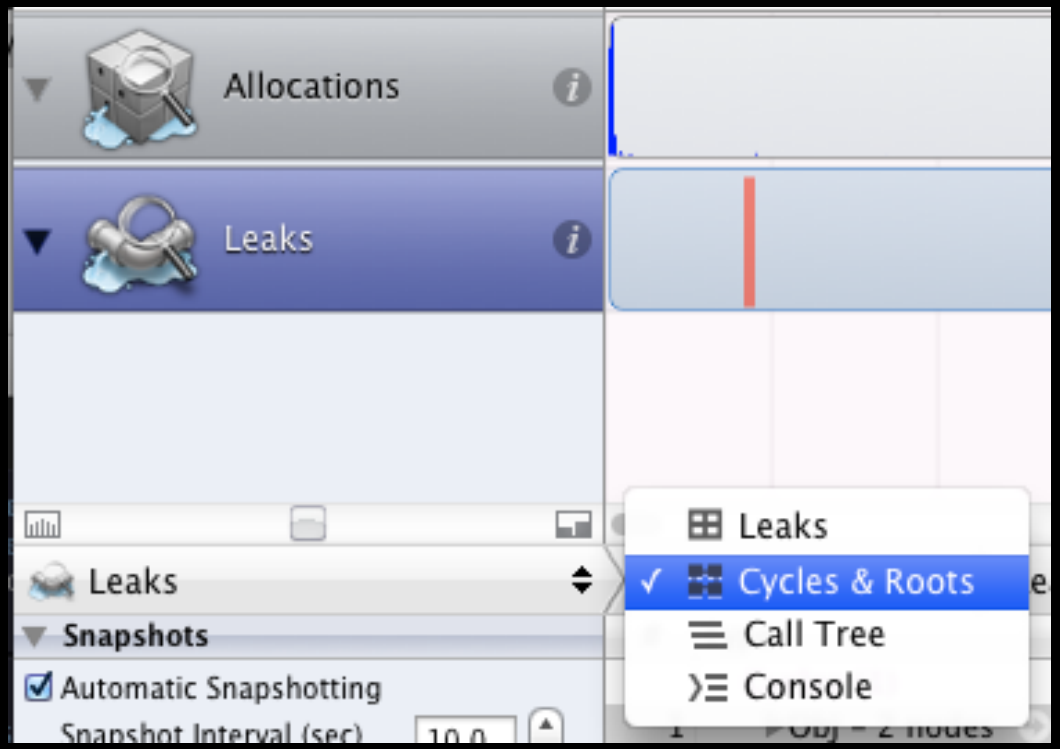
```
objc[52138]: cannot form weak reference to
instance (0x104d18260) of class NSWindow
Illegal instruction: 4
```

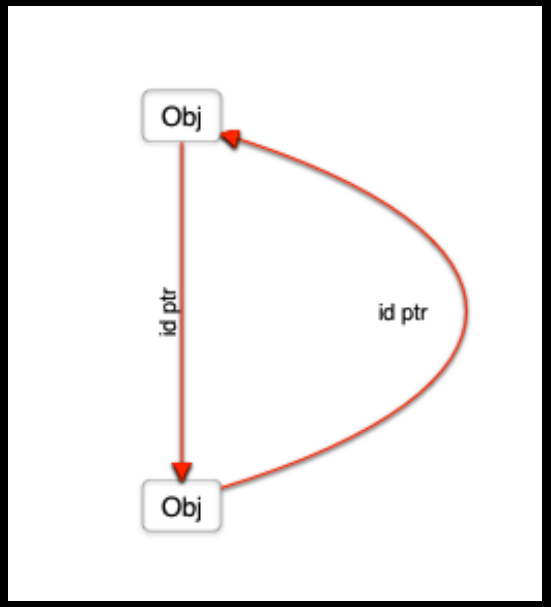
PLWeakCompatibility

- __weak for 10.6 and iOS 4

http://github.com/plausiblelabs/PLWeakCompatibility

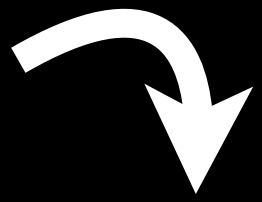
Instruments





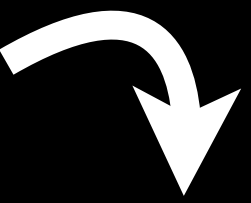
@property

@property (retain)



@property (strong)

@property (assign)



~~@property (unsafe_unretained)~~

@property (weak)

Blocks

```
void (^block)(void) = ^{
    [self doSomethingImportant];
};

ivar = [block copy];
```

Blocks

```
void (^block)(void) = ^{
    [self doSomethingImportant];
};

ivar = block;
```

Blocks

```
return ^{
    [self doSomethingGreat];
};
```

But!

```
[array addObject: ^{ ... }];

- (id)method
{
    return ^{ ... };
}
```

Blocks

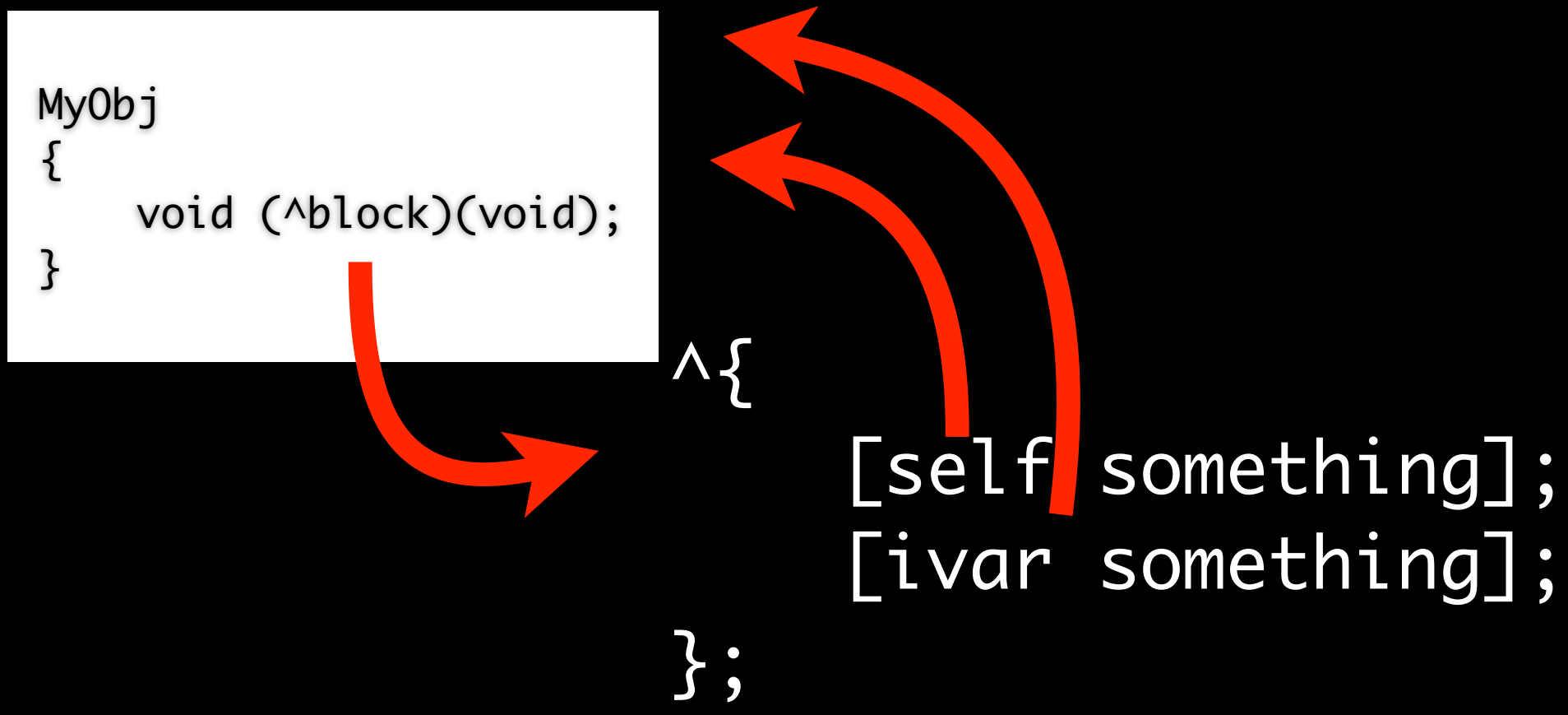
```
[[[[[[[[[[block copy] copy]
    copy] copy] copy] copy]
    copy] copy] copy] copy]
```

Blocks

Beware `__block`!

Use `__weak`

Blocks and Cycles



Blocks and Cycles

```
__weak MyClass *weakSelf = self;
^{
    [weakSelf something];
};
```

Blocks and Cycles

```
__weak MyClass *weakSelf = self;
^{
    if(weakSelf)
        [weakSelf->ivar something];
};
```

Blocks and Cycles

```
__weak MyClass *weakSelf = self;
^{
    if(weakSelf)
        [weakSelf->ivar something];
};
```

Blocks and Cycles

```
__weak MyClass *weakSelf = self;
^{
    MyClass *localSelf = weakSelf;
    if(localSelf)
        [localSelf->ivar something];
};
```

Toll-Free Bridging

```
NSString *string = (id)CFDictionaryGetValue(cfDict, key);
```

Toll-Free Bridging

~~NSString *string = (id)CFDictionaryGetValue(cfDict, key);~~

```
NSString *string = (__bridge id)CFDictionaryGetValue(cfDict, key);
```

Toll-Free Bridging

```
CFStringRef cfstr = CFPreferencesCopyAppValue(...);
NSString *str = (__bridge id)cfstr;
CFRelease(cfstr);
```

Toll-Free Bridging

```
CFStringRef cfstr = CFPreferencesCopyAppValue(...);
NSString *str = CFBridgingRelease(cfstr);
```

Toll-Free Bridging

```
NSString *str = [dict objectForKey: key];
CFStringRef cfstr = CFBridgingRetain(str);
CFDoSomething(cfstr);
CFRelease(cfstr);
```

Toll-Free Bridging

```
NSString *str = [dict objectForKey: key];
CFStringRef cfstr = (__bridge CFStringRef)str;
CFDoSomething(cfstr);
```

Context Pointers

```
void *contextPtr = CFBridgingRetain(obj);
...
id obj = CFBridgingRelease(contextPtr);
```

Bridging

`__bridge` (cast)

direct transfer, no ownership

`CFBridgingRetain()`

transfer ownership from ARC into CF

`CFBridgingRelease()`

transfer ownership out of CF and into ARC

Structs

Run away!

Conclusion

Garbage collection is cool!

Objective-C is cool!

Objective-C plus garbage collection is **not cool!**

ARC is not garbage collection

But... close enough?

Zeroing weak references are cool!

`__weak` to break cycles

`__unsafe_unretained` when you can't use `__weak`

PLWeakCompatibility for `__weak` on older OSes

Leaks Instruments finds cycles!

ARC only does Objective-C

`__bridge, CFBridgingRetain, CFBridgingRelease`

- ARC is good
- Turn it on
- Convert your code
- Saves you time
- Saves you money
- Saves the world