Blockhead

Blocks in Objective-C

Outline

- The Idea of Blocks
- Basic Block Syntax
- Simple Examples
- Block Theory *
- More Involved Examples
- Building Blocks

Blocks: The Idea

Blocks

- Blocks of code that can be passed around
- Carry with them their own copies of local data
- Follow standard C scoping rules
- Immutable
- Thread-safe

Blocks … in C*

Snow Leopard iOS 4

Closures

1960 1970 1980 1990 2000 2010

C Multicore

* and Objective-C / C++

A block, per se

$$c(t) = c_0 (1+r)^t$$

```
float bacteriaStart = 100, growthRate = 0.005;
NSDate *startDate = [NSDate date];

float t = [[NSDate date] timeIntervalSinceDate:startDate];
float newCount = bacteriaStart * powf( 1+growthRate, t );
NSLog( @"Now there are %f bacteria!", newCount );
```

```
Now there are 100.000015 bacteria!
```

A block, per se

```
float bacteriaStart = 100, growthRate = 0.005;
NSDate *startDate = [NSDate date];

{
  float t = [[NSDate date] timeIntervalSinceDate:startDate];
  float newCount = bacteriaStart * powf(1+growthRate,t);
  NSLog( @”Now there are %f bacteria!”, newCount );
}
```

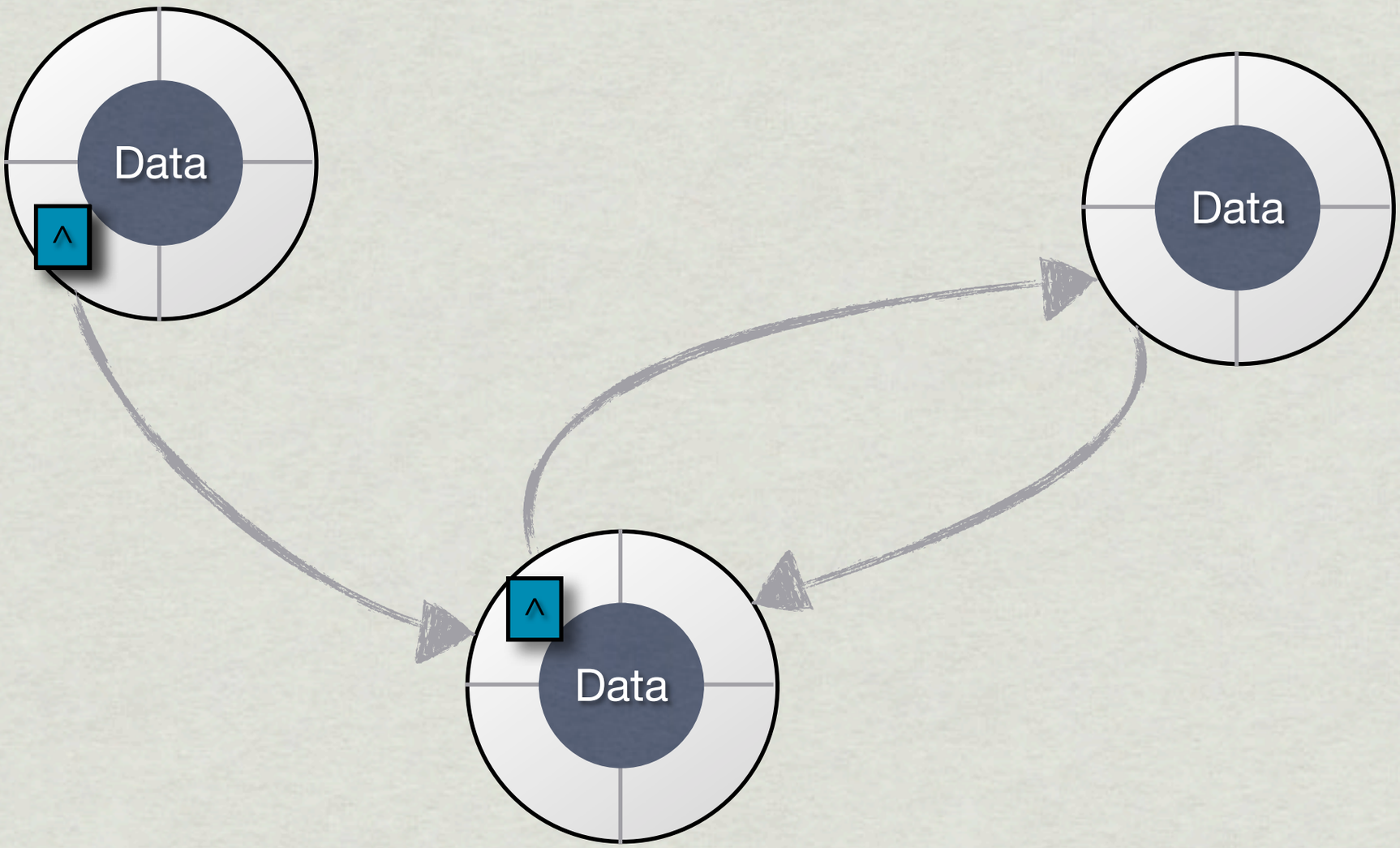
```
{
  float t = [[NSDate date] timeIntervalSinceDate:startDate];
  float newCount = bacteriaStart * powf(1+growthRate,t);
  NSLog( @”Now there are %f bacteria!”, newCount );
}
```

A block, per se

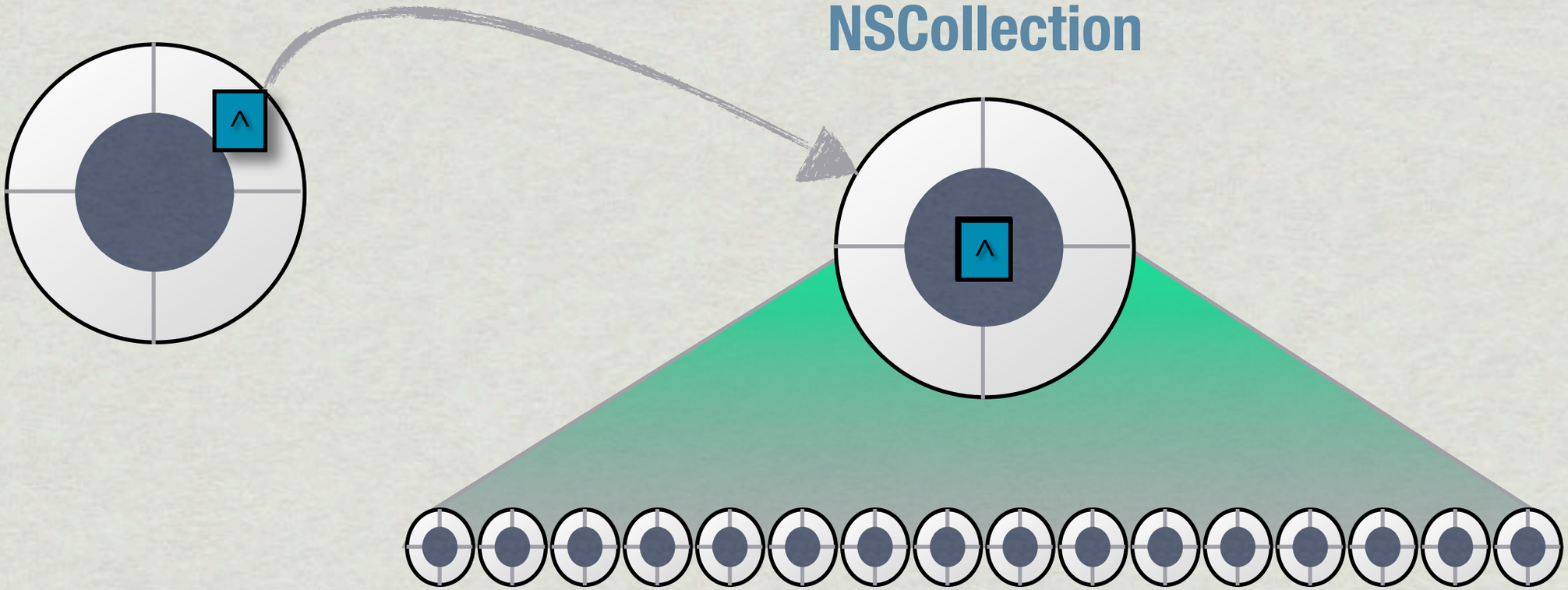
```
float bacteriaStart = 100, growthRate = 0.005;
NSDate *startDate = [NSDate date];

bacteriaCheck = ^{
  float t = [[NSDate date] timeIntervalSinceDate:startDate];
  float newCount = bacteriaStart * powf(1+growthRate,t);
  NSLog( @"Now there are %f bacteria!", newCount );
};

bacteriaStart = 500;
```

```
bacteriaCheck();
```

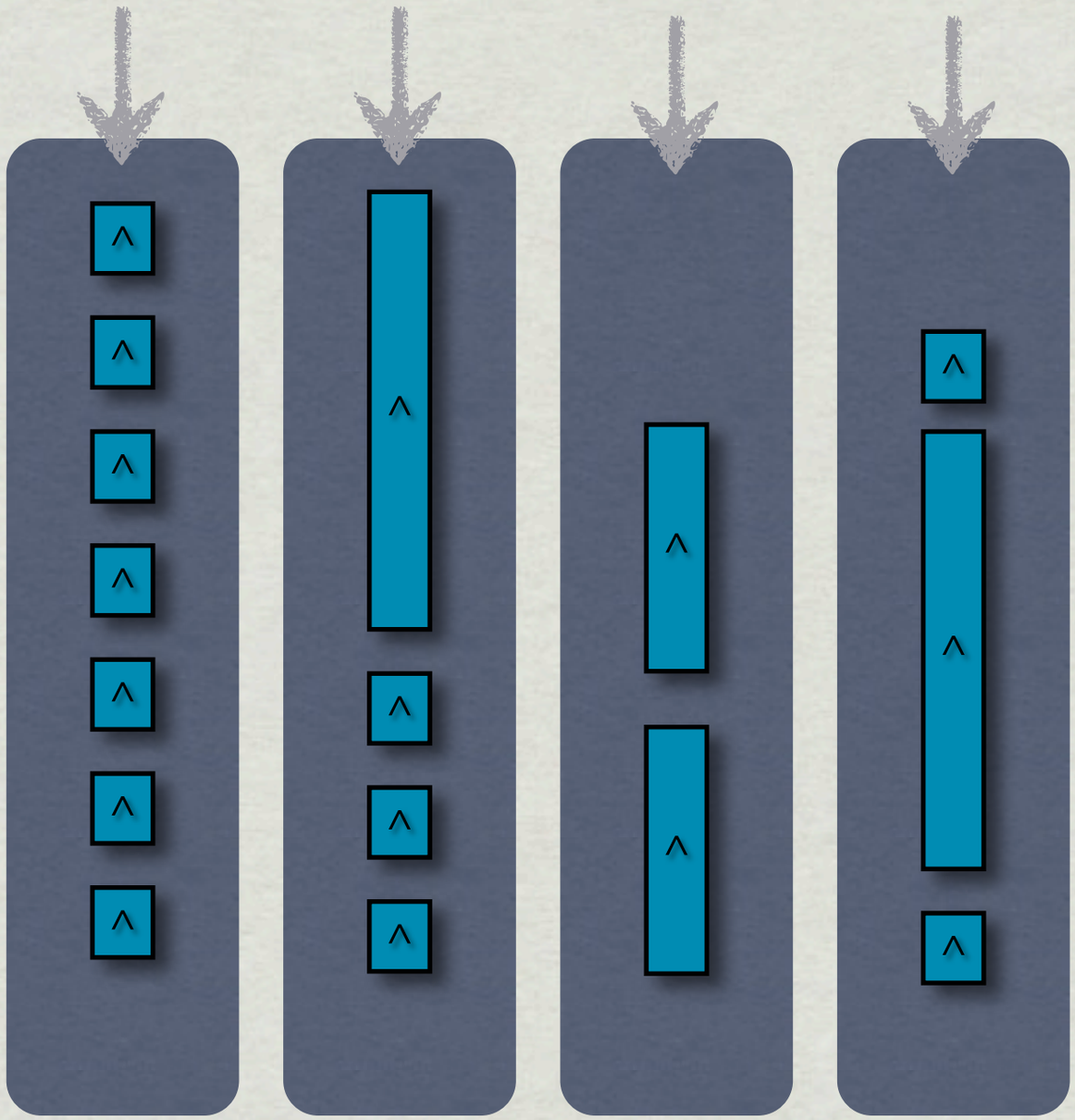
Blocks & Objects



Blocks & Objects



Blocks & Concurrency

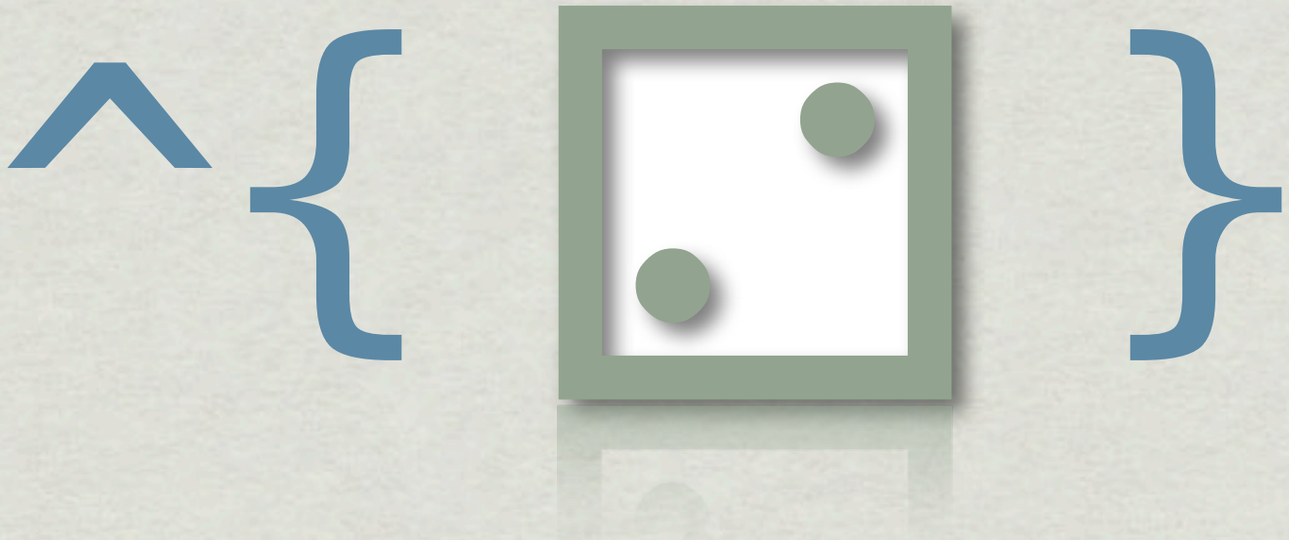


GCD / NSOperation Queues

New Design Patterns

- Passing units of work around
- Enumerating over collections
- Easy(er) concurrency / Entry point to GCD
- Replacement for the callback pattern
- Nestable, scoped functions
- much more besides

Block Syntax



Block syntax

```
myBlock = ^{ ... };

myBlock();

^{ ... }();

[someObject someMethod:^{ ... }];
```

Block literal syntax

```
^void (void) { printf( “hi\n” ); }

    ^(void) { printf( “hi\n” ); }

^void       { printf( “hi\n” ); }

           ^{ printf( “hi\n” ); }
```

Block literal syntax

```
^int (int x) { return x*3; }

    ^(int x) { return x*3; }

          ^{ return 3; }
```

Block literal syntax

```
^                   { doStuff();                }

^float              { return aFloat;            }

^         (int x) { doStuff( x );               }

^float (int x) { return aFloat( x );  }

       ^(int x) { return aFloat( x );  }
```

Block literal syntax

- execute now:

```
^{ doStuff(); }();

y = ^{ return aFloat; }();

^(int x) { doStuff(x); }( 5 );

y = ^(int x) { return aFloat(x); }( 5 );
```

Block literal syntax

- use in method now:

```
[object method:^{ doStuff(); }];
```

- usually you also need to include arguments:

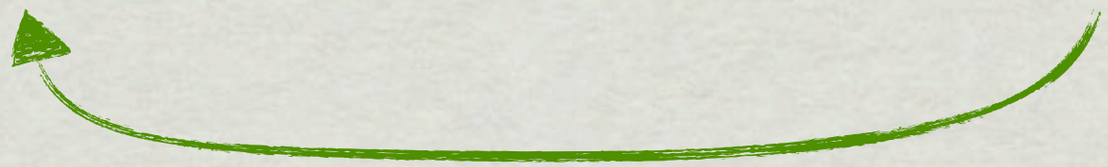
```
[object method:^(id obj) { ... }];
```

Defining blocks

- No frills:

```
void (^frank)(void);
float (^legs)(int, NSString *, BOOL *);
```

```
float (^chuck)(int) = ^float (int x) {…};
```



- Simplest block definition:

```
void (^chuck)() = ^{ printf( “hi” ); };
```

Typedef cleanup

- Not too bad if just one function
- But is error prone and not visually clear

```
typedef float (^intFloatFn)(int x);
intFloatFn cecil = ^float (int x) {…};
intFloatFn bob   = ^float (int x) {…};
```

- Many find useful typedefs like:

```
typedef void (^BasicBlock)(void);
```

Code Examples!

Blocks in Cocoa

Core Animations

```
[UIView beginAnimations:nil context:NULL];
[UIView setAnimationDuration:3.0];

box.alpha = 0.5;
box.origin.x = 200.0;

[UIView setAnimationDidStopSelector:@selector(...)];

[UIView commitAnimations];
```

Core Animations

```
[UIView animateWithDuration:3.0
                 animations:^{

    box.alpha = 0.5;
    box.origin.x = 200.0;
}];
```

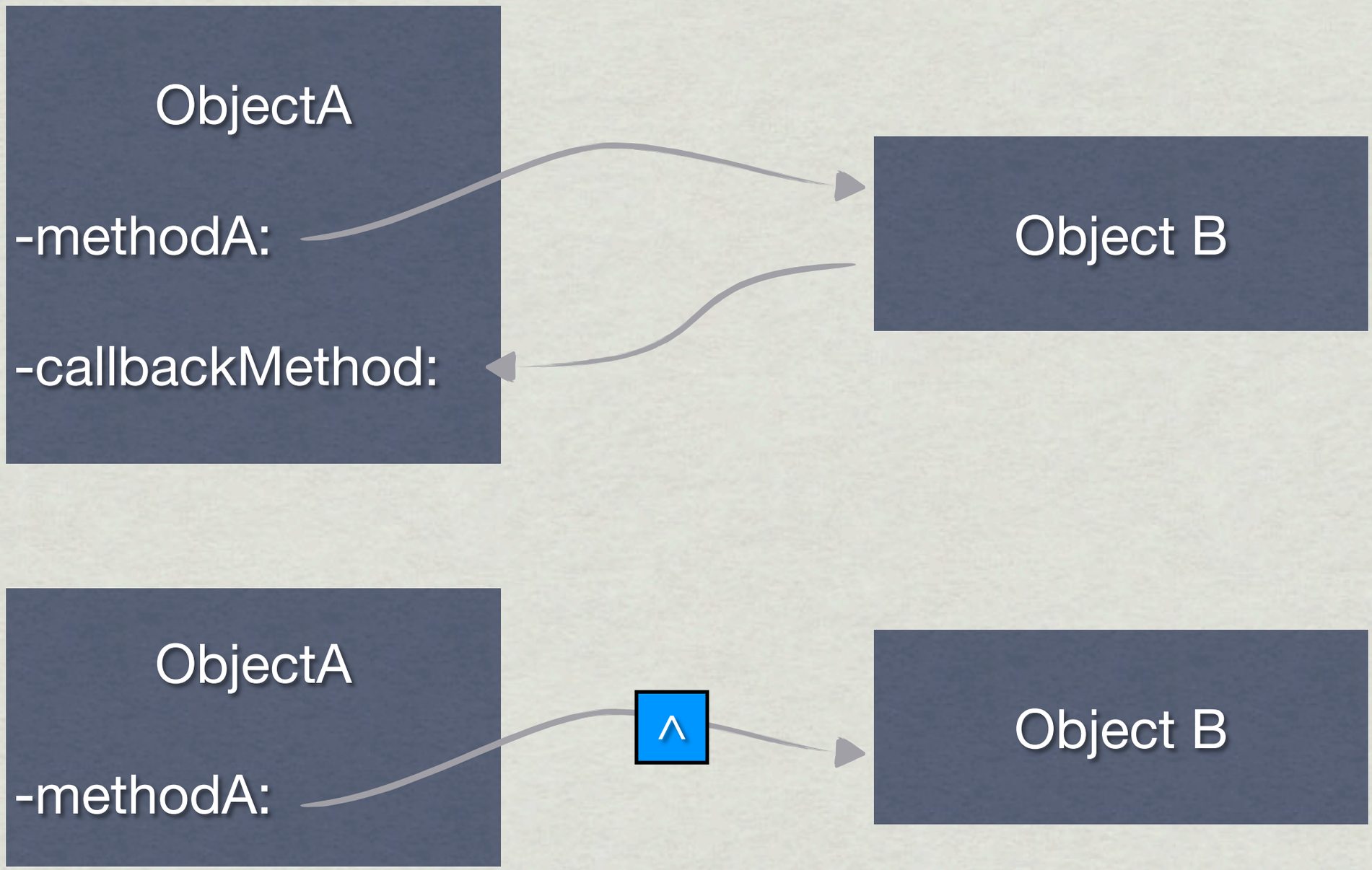
Core Animations

```
[UIView animateWithDuration:3.0
                 animations:^{

    box.alpha = 0.5;
    box.origin.x = 200.0;
}
                 completion:^{

    [box setBackgroundColor:[UIColor blueColor]];
}];
```

Very rare to have 2-block methods

(Discouraged)

Callbacks replaced



Callbacks replaced

```
UIPageViewController *pageVC;

[pageVC setViewControllers:...
                 direction:...
                  animated:...
                completion:^(BOOL finished){

    // stuff to do when page has been turned
}];
```

Collections

- NS classes

  `NSArray`, **`NSDictionary`**, **`NSRegularExpression`**, **`NSSet`**, **`NSIndexSet`**, **`NSOrderedSet`**, **`NSFileManager`**

- Block Iterating vs NSEnumerator / fast enumeration

  both object *and* index — as fast if not faster
  both key and value — concurrency/reversing

Collections

- Fast enumeration is a great feature in ObjC 2.0

```
for (id obj in collection) {...}
```

- Iterating over NSSet / NSArray

```
-enumerateObjectsUsingBlock:
-enumerateObjectsWithOptions: usingBlock:
```

- Iterating over NSDictionaries:

```
-enumerateKeysAndObjectsUsingBlock:
-enumerateKeysAndObjectsWithOptions: usingBlock:
```

Collections

```
NSSet *people;

[people enumerateObjectsUsingBlock:^(id person,
                                     BOOL *stop){

    [person doSomethingFancy];
}];
```

Collections

user dictionary:

email -> info

```
NSDictionary  *people  = ...;
EmailerDaemon *emailer = ...;
NSString *mailerString = ...;

[people enumerateKeysAndObjectsWithOptions:
                                    NSEnumerationConcurrent

        usingBlock:^(id key, id value, BOOL *stop) {

    [emailer sendMessage:mailerString
             withSubject:[...:@"Hi %@!", value.name]
               toAddress:key];
}];
```

Collections

```
NSSet *servers;

[servers enumerateObjectsWithOptions:
                                    NSEnumerationConcurrent
                                usingBlock:^(id server,
                                             BOOL *stop){

    if ([server hasLowLoad]) {
        [server doSomethingComplicated];
        *stop = YES;
    }
}];
```

BOOL *stop Pattern

NSArray

NSAttributedString

NSDictionary

NSIndexSet

NSLinguisticTagger

NSOrderedSet

NSRegularExpression

NSSet

NSString

NSDragging

NSDraggingSession

NSEvent

ALAssetsGroup

ALAssetsLibrary

EKEventStore

MPMediaEntity

Collections

```
NSIndexSet *lifeSet = [planets

    indexesOfObjectsPassingTest:^(id obj,
                                  NSUInteger index,
                                        BOOL *stop) {

    return [obj mightContainLife];
}];

[planets enumerateObjectsAtIndexs:lifeSet
                          options:nil
                       usingBlock:^(id planet,
                  NSUInteger idx, BOOL *stop) {

    [planet hail];
    [planet welcomeNewAlienOverlords];
}];
```

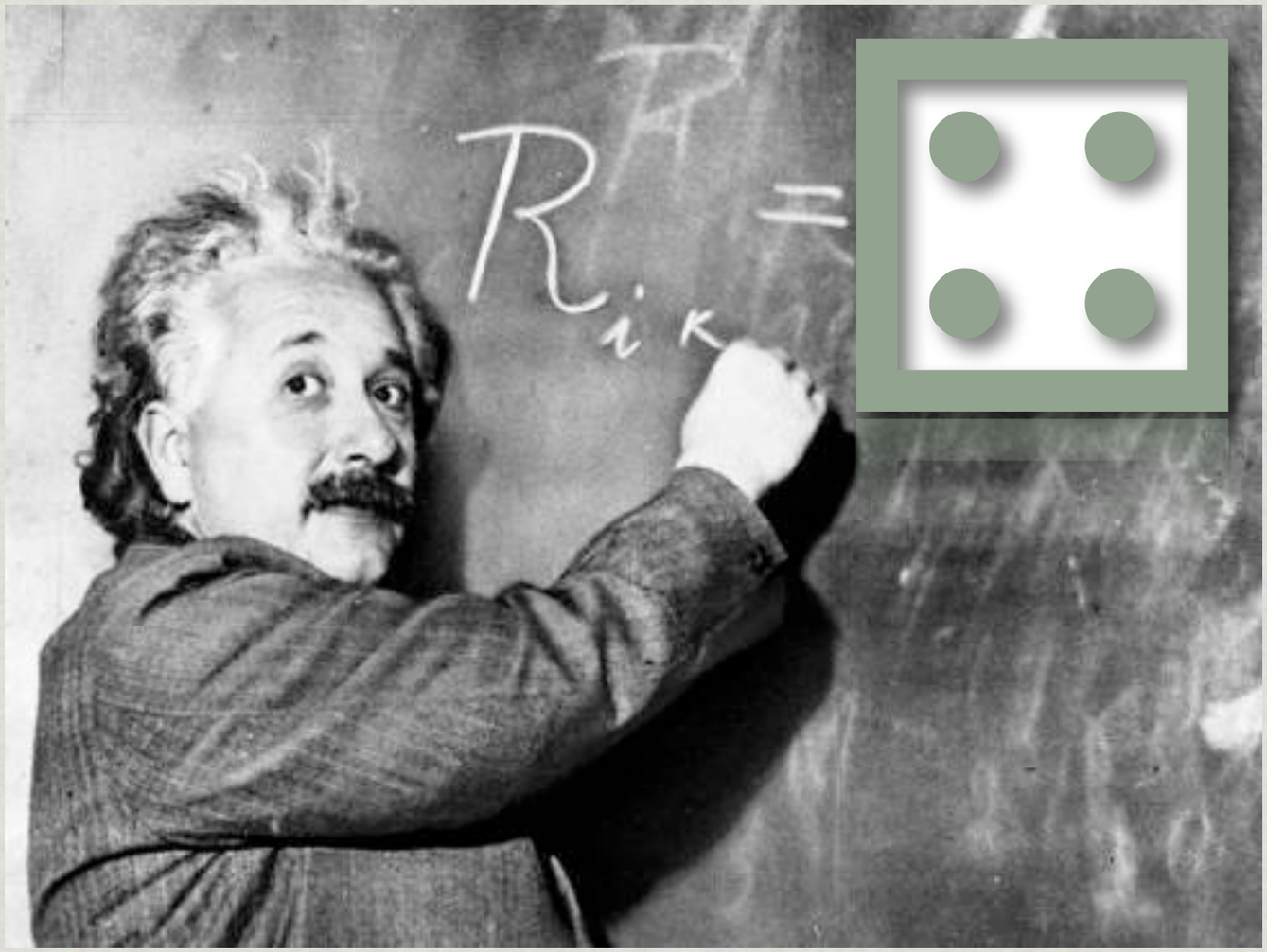
Collections

```
NSIndexSet *lifeSet = [planets

    indexesOfObjectsPassingTest:^(id obj,
                                  NSUInteger index,
                                  BOOL *stop) {

    return [obj mightContainLife];
}];

[lifeSet enumerateIndexesUsingBlock:
                      ^(NSUInteger idx, BOOL *stop) {

    [[explorers objectAtIndex:idx]
        sendOnTripTo:[planets objectAtIndex:idx]];
}];
```

More C-level routines

```
CFRunLoopPerformBlock
heapsort_b
mergesort_b
psort_b
qsort_b
atexit_b
bsearch_b
err_set_exit_b
glob_b
scandir_b
```

Block Theory



Block as Closures

- Blocks “capture” / freeze the state of local vars
- Can also address external mutables
- Can outlive the scope they were defined in

Blocks and Memory

Fast →

Stack

Dynamic memory →

`(m|c)alloc` (inc. NSObjects)

Can outlive local scope

Has to be managed

Heap

Data: Uninitialized

Data: Initialized

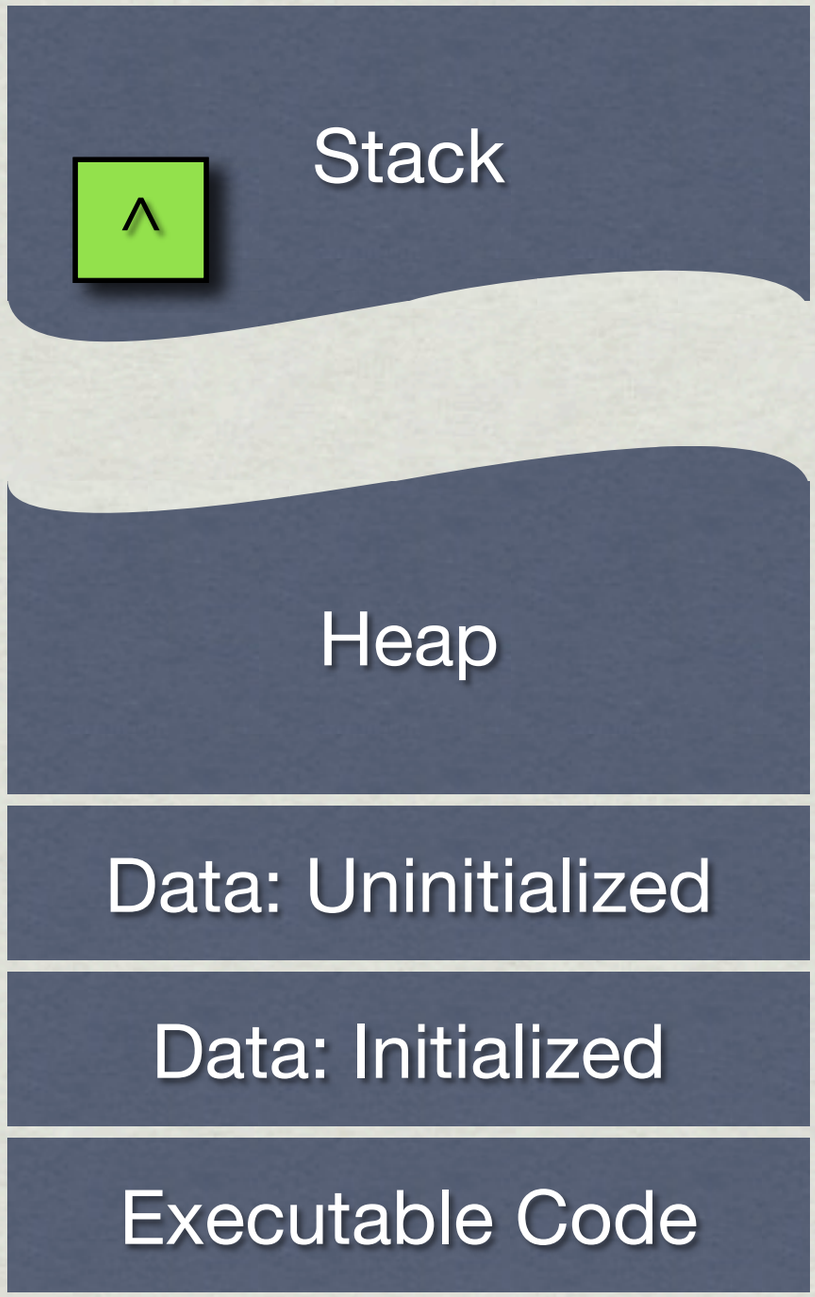
Executable Code

Blocks and Memory

Blocks created here

Copied to heap
when you need them
to outlive their scope

^ Can then be sent out
into the world



Blocks and Memory

Blocks created here → Stack

Blocks also here → Heap

Possssibly here too → Data: Uninitialized

Actual functions → Data: Initialized

Executable Code

Blocks as NSObjects

- Minimal NSObjects
  - -copy, -retain, -release
  - standard stuff (-class, -description…)
  - -invoke

Non-Documented Method Warning
Use -invoke for debugging only

Blocks as NSObjects

- A block's C struct contains, among other things: isa

```
(gdb) print *myBlock
$5 = {
  __isa = 0xa02a3f20,
  __flags = 50331649,
  __reserved = 0,
  __FuncPtr = 0x17422,
  __descriptor = 0x34e80,
  y = 50,
  x = 0xe1110a0
}
```

Blocks as NSObjects

- **`-copy -retain -release -autorelease`**
- **`Block_copy() Block_release()`**
- *Pre-ARC:*
  - Pushed to the heap with **`-copy`** (*not* **`-retain`**)
  - Released with **`-release`**

Retain Rules for Blocks

- locals ➛ const
- objects ➛ [obj retain]
- ivars ➛ [self retain]

```
int y = 50;


a = ^{
    NSString *n = [bob name];
    NSString *m = myname;
    x += y;
};
```

__block

- A new C storage class!
  - What's a storage class?
  - Famous storage classes: **`extern`**, **`static`**
  - Less famous: **`auto`**, **`register`**
  - And now: **`__block`**

storage class: auto


```
{
  auto int x;
  // some
  // super
  // awesome
  // code
}
```

Implied for all variables not otherwise marked

storage class: extern

RUN — time — **QUIT**

```
extern int x;
{
  // code
}

{
  // code
}

{
  // code
}
```

Implied for function prototypes

storage class: static

RUN time **QUIT**

must be constant expression, done once at compile-time

```
{
  static int x;
  // some
  // super
  // awesome
  // code
}
```

storage class: __block

RUN — time — QUIT

```
{
  __block int x;
  blockType a = ^{
    // super
    // awesome
    // code
  };
  b = [a copy];
  [c use:b];
}
```

```
b();
    b();
        b();
            b();
```

__block vars

- Mutable and Outlive local scope
- Multiple block instances share same __block var
- No arrays! Pointers to arrays & structs are OK.
  - BUT: lifetime of pointers must match block's!
- *Non-ARC*: Objects not sent -retain messages
- *Non-ARC*: Vars/objects manually mem-managed

__block uses

- Global-ish / singleton-y variable among blocks
- Update block behavior from outside
- To do fancy things among multiple threads
  - *BUT* setting __block vars is *not* thread safe

Example

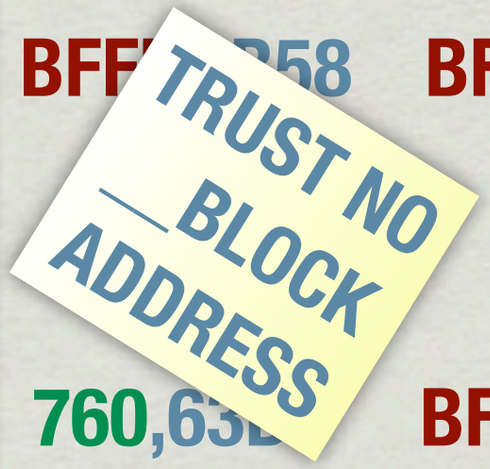
```
NSString *x = [NSString stringWithContentsOfFile:...];

NSMutableArray *colors = [NSMutableArray array];
__block int colorCounter = 0;

[x enumerateLinesUsingBlock:^(NSString *line, BOOL *stop) {

  if ([line rangeOfString:@"color"].location != NSNotFound) {
    [colors addObject:line];
    ++colorCounter;
  }

}];
```

```
(gdb) print...
```

```
__block int x;
int y = 50;


a = ^{ x += y; };


b = [a copy];


a(); // or b();
```

```
{
    x += y;
}
```

&x	&y	a	b
BFFF,CB58	BFFF,CB6C		
BFFF,CB58	BFFF,CB6C	BFFF,CB2C	
760,63B0	BFFF,CB6C	BFFF,CB2C	760,1B40
760,63B0	BFFF,CAD8		

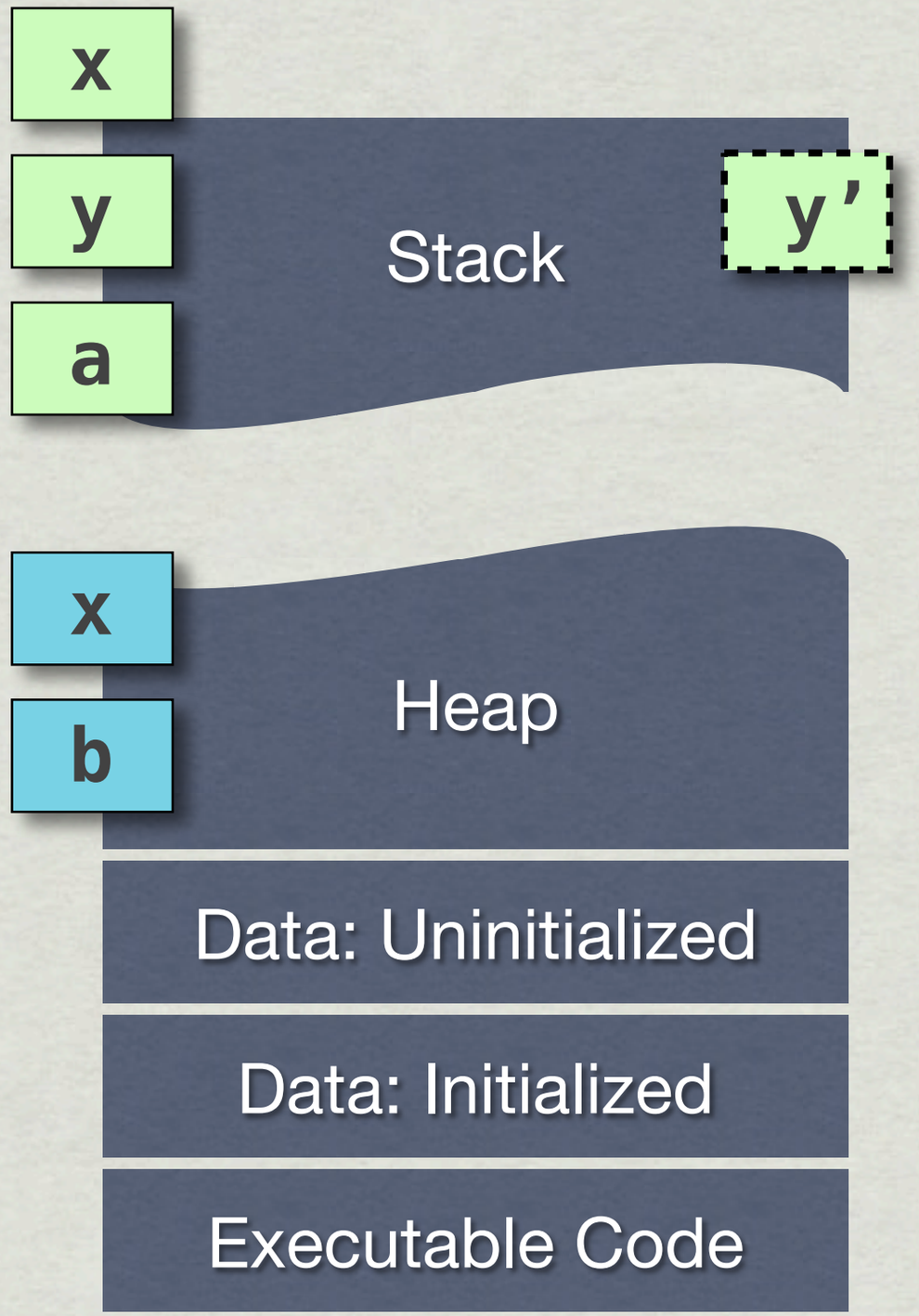


```
a = ^{ x += y; };

b = [a copy];

c = ^{ return 3; };
```

```
(gdb) po [a class]
__NSStackBlock__

(gdb) po [b class]
__NSMallocBlock__

(gdb) po [c class]
__NSGlobalBlock__
```

```
__block int x;
int y = 50;


a = ^{ x += y; };


b = [a copy];


a(); // or b();
```

```
{
    x += y;
}
```



```
__block int x;
int y = 50;


a = ^{ x += y; };


b = [a copy];


a(); // or b();
```

```
(gdb) print *b
$5 = {
  __isa = 0xa02a3f20,
  __flags = 50331649,
  __reserved = 0,
  __FuncPtr = 0x17422,
  __descriptor = 0x34e80,
  y = 50,
  x = 0xe1110a0
}
```

note: *not* **&x**

a

```
{ ...
  x = 0xbfffcb44
}
```

b

```
{ ...
  x = 0xc429bb0
}
```

```
{ ...
  __forwarding = 0xc429bb0
}
```

```
{ ...
  __forwarding = 0xc429bb0
}
```

“x” (as these blocks know it)

Retain Rules for Blocks

- locals ➛ const
- __block ➛ heap
- objects ➛ [obj retain]
- ivars ➛ [self retain]
- cycling gotchas

```
int y = 50;
__block int x;


a = [^{

    NSString *n = [bob name];

    NSString *m = myname;

    x += y;

} copy];
```

Retain Cycling

Non-ARC:

```
huntBlock = Block_copy(^{
   if([self isHungry])
      [self hunt];
});
```

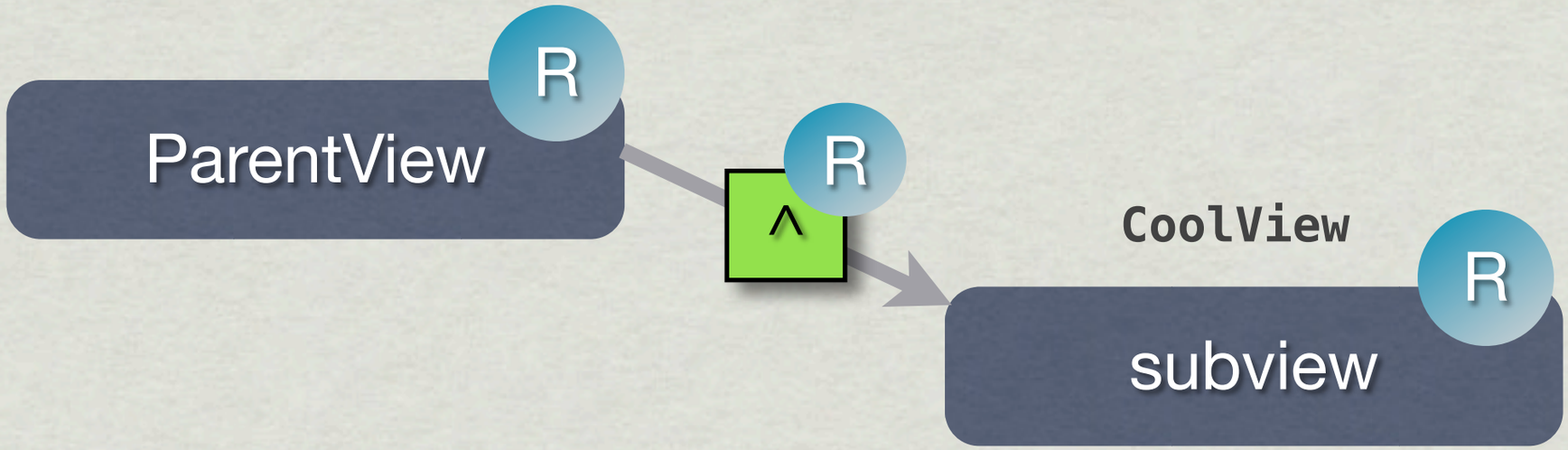
could possibly
create a retain cycle

Retain Cycling

Non-ARC:

```
__block id me = self;

huntBlock = Block_copy(^{
   if([me isHungry])
      [me hunt];
});
```

Retain Cycle Example




```
@property (copy) drawBlock;
```

```
[PV addSubview:[CoolView viewWithDrawMethod:^{
    [parentColor set];
    //...
}]
```

ivars and your bad self

- Using self or ivar retains self, possible retain cycle!
  - iVars: `id ivarSub = ivar;`
  - Self:
    - Non-ARC: `__block id mySelf = self;`
    - ARC (old): `__block...` later: `mySelf = nil;`
    - ARC (new): `__weak id mySelf = self;`

ivars and your bad self

ARC & iOS4 / Mac OS X Tiger:

```
__block Person *me = self;

someBlock = ^() {
    [me doSomething];
    me = nil;
};
```

This will *cause* a retain cycle in ARC — so…

ivars and your bad self

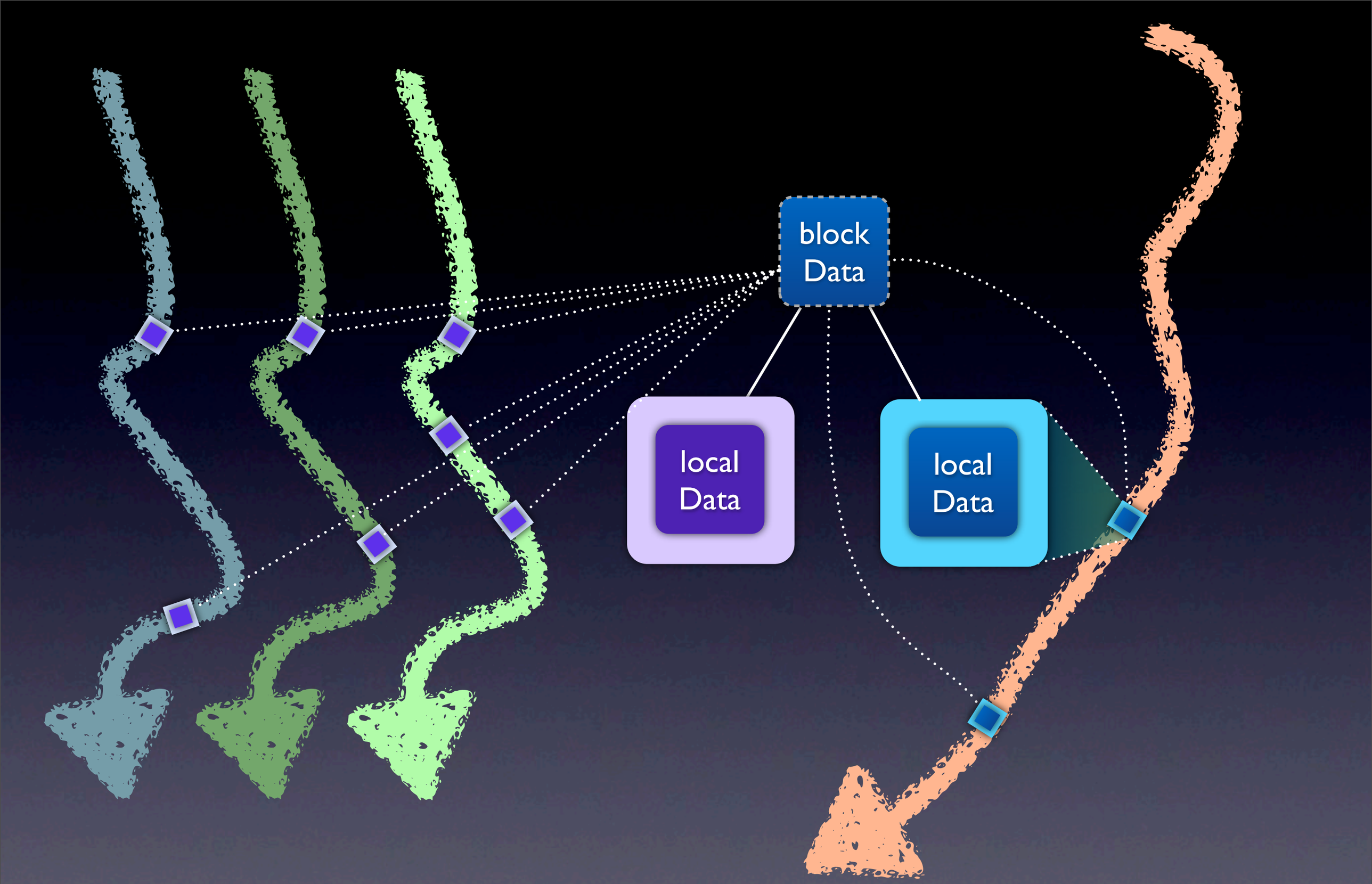
ARC, iOS5 / Leopard on up:

```
__weak Person *me = self;

someBlock = ^() {

    [me giveBigRaise];

};
```

ivars and your bad self

ARC, iOS5 / Leopard on up,
non-trivial blocks:

```
__weak Person *me = self;

someBlock = ^() {

    Person *strongMe = me;

    if (strongMe) { ... }

    else { ... }

};
```


block
Data
local
Data
local
Data

Pre-ARC Gotchas

```
void (^a[3])(void);

for( i=0; i<3; ++i ) {

     a[i] = ^{ ... };

}
```

```
void (^a)(void);

if( condition ) {

     a = ^{ ... };

}
```

- *Both* examples don't work
- In each case the scope of the block literal is inside

Pre-ARC Gotchas

```
void (^a[3])(void);

for( i=0; i<3; ++i ) {

     a[i] = Block_copy( ^{ ... } );

}
```

```
void (^a)(void);

if( condition ) {

     a = Block_copy( ^{ ... } );

}
```

Blocks in NS Collections

- If you have a bunch of blocks you want in an array, which have not already been put on the heap:

```
[NSArray arrayWithObjects:
    [[blockA copy] autorelease],
    [[blockB copy] autorelease], nil];
```

- Or inline it:

```
[NSArray arrayWithObjects:
    [[^{...} copy] autorelease],
    [[^{...} copy] autorelease], nil];
```

Blocks in NS Collections

- ARC version:

```
[NSArray arrayWithObjects:
    [blockA copy],
    [blockB copy], nil];
```

- ARC version:

```
[NSArray arrayWithObjects:
    [^{...} copy],
    [^{...} copy], nil];
```

Blocks in Objects

- return syntax: **`- (float (^)(int))method...`**
- When return blocks from methods:

```
return [[blocky copy] autorelease];
return [[^{ ... } copy] autorelease];
```

- ARC:

```
return blocky;
return ^{ ... };
```

- Blocks can be **`@properties`**
- Scoped / nested functions *safely*

Factorial Example

```
int (^factorial)(int) = ^(int n) {

    if (n==1) return 1;
    else return n * factorial( n-1 );
};
```

```
__block int (^factorial)(int) = ^(int n) {

    if (n==1) return 1;
    else return n * factorial( n-1 );
};
```

Blocks in the gdb

- invoke-block

```
(gdb) invoke-block myBlock arg1 arg2
```

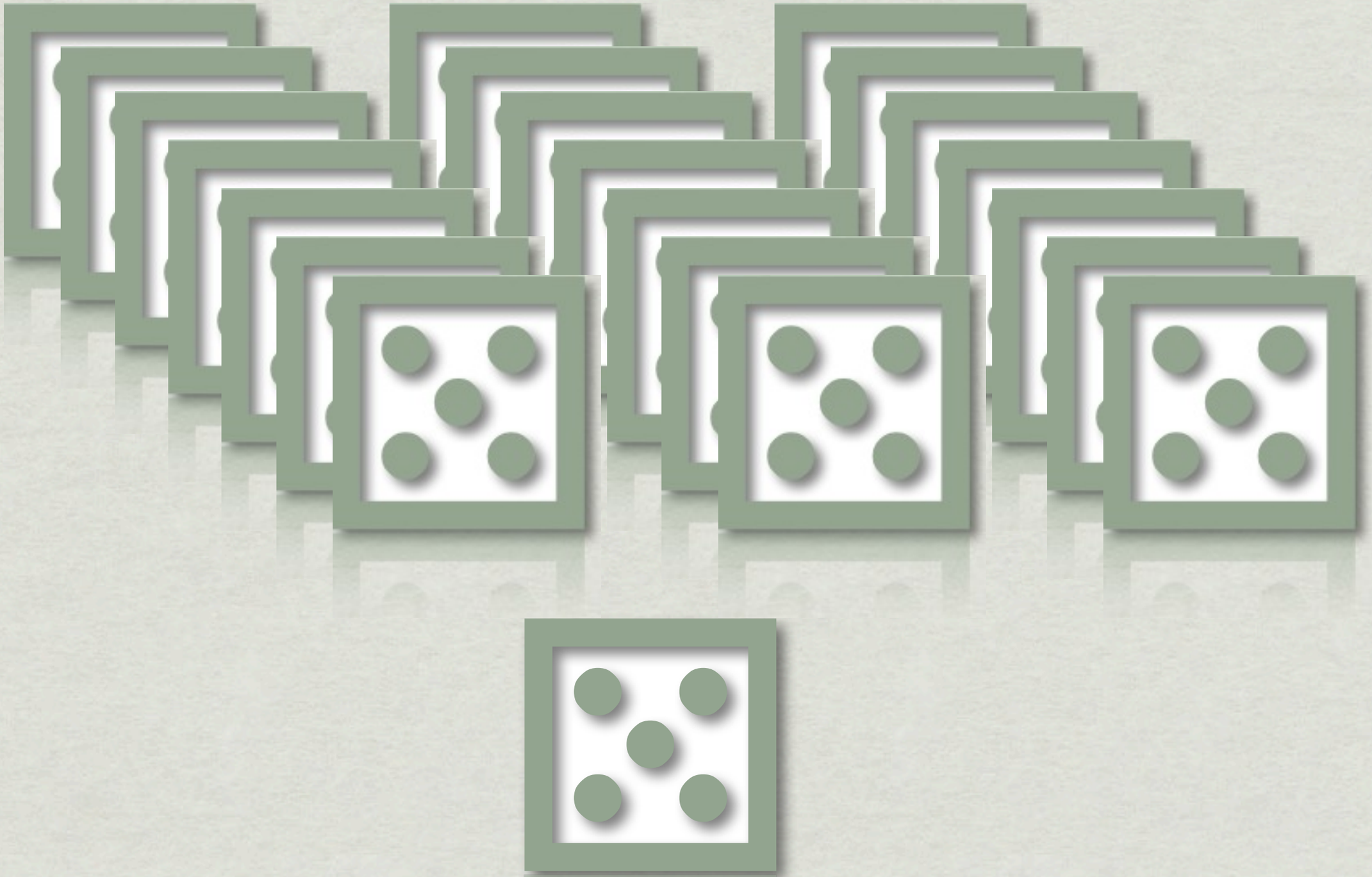
- invoke: only for blocks that take no arguments

```
(gdb) print (void) [myBlock invoke]
```

```
(gdb) po [myBlock invoke]
```

Non-Documented Method Warning
Use -invoke for debugging only

Blocks & Concurrency



Blocks and Threading

- NSOperationQueue
- Grand Central Dispatch

Grand Central Dispatch

- C-level queuing system
- Useful even on single core systems - very efficient
- GCD works seemlessly with blocks
- Low overhead
- Queue management instead of thread management

dispatch_ing Blocks

```
dispatch_queue_t someQueue =
    dispatch_queue_create( “com.a.purpose”, NULL );

dispatch_async( someQueue, ^{ greatStuff(); } );

dispatch_release( someQueue);
```

- Takes care of locking issues!

```
dispatch_async( dispatch_get_main_queue(), ^{

    [someView someDisplayStuff];
} );
```

NSBlockOperation

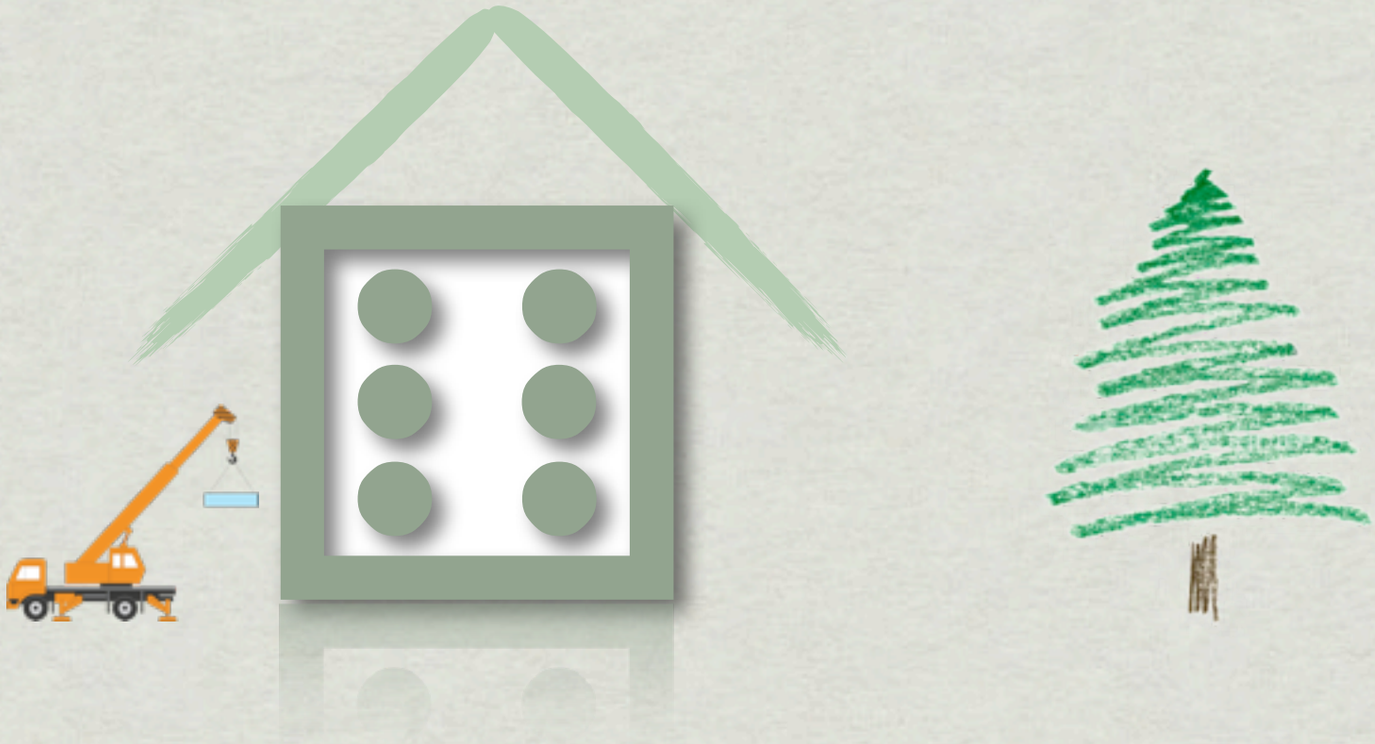
- Group of n concurrent blocks
- Can use **KVO** notifications
- Can **cancel** NSOperations mid-flight
- Make use of graph-based **dependencies**
- NSOperations can be **archived**
- If already using NSOperationQueues, you don't need to also start using dispatch_queues

NSBlockOperation

```
NSOperationQueue *opQueue = [NSOperationQueue new];
NSBlockOperation *blockOp;

blockOp = [NSBlockOperation blockOperationWithBlock:^{
    [someView someDisplayStuff];
}];

[opQueue addOperation:blockOp];
```

```
NSOperationQueue *opQueue = [NSOperationQueue new];

[opQueue addOperationWithBlock:^{
    [someView someDisplayStuff];
}];
```

Building Blocks



Blocks in your own code

- Think about ways in which blocks can improve the overall design of your apps
  - Check out Cocoa API patterns
    - Ex: block `@property` pattern
  - Remember this is C / C++ / Objective-C stuff
  - Low overhead concurrency
  - Borrow patterns from other languages

Methods taking blocks

- Convention: one block only, last argument

  Keeps multipart method names together, even when used with an inline block

```
[someArray   indexOfObject:
             inSortedRange:
                   options:
          usingComparator:^(...) {

    // code code code

}];
```

Methods taking blocks

- Multiple blocks can be fine in some circumstances, especially if inlining unlikely

```
[dataSet compareFunction:fn1 withFunction:fn2];
```

- Of course, in your own code, explore stylings

Case in Point

- Anecdote: Recent iPhone app
  - 16 Blocks in Animations
  - 2 Blocks in non-animation Cocoa methods
  - 2 Custom Blocks, primarily as scoped fns
  - 1 Pure C Block (qsortb)

The end. Thanks!