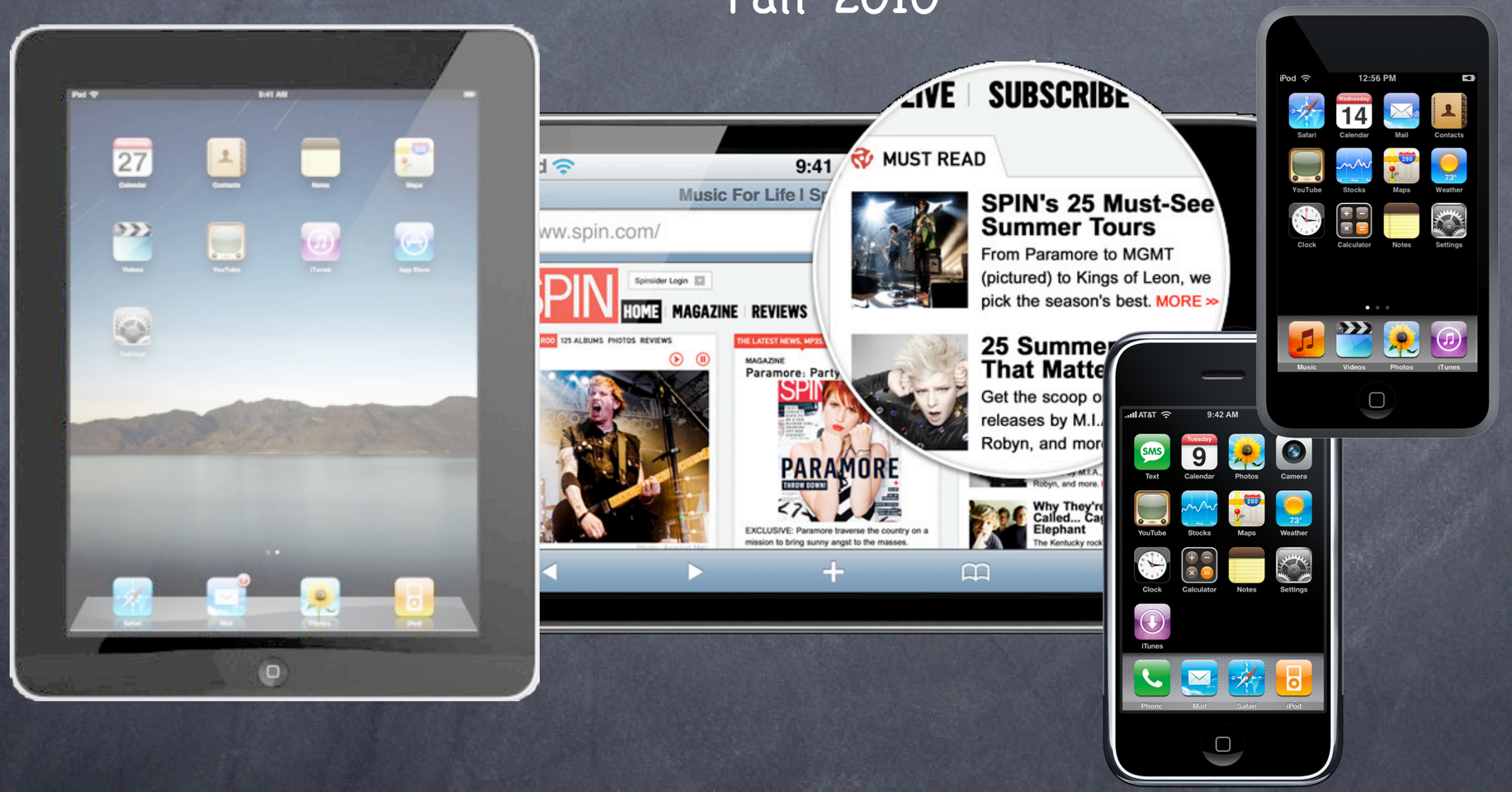
Stanford CS193p

Developing Applications for iPhone 4, iPod Touch, & iPad

Fall 2010



Today

- Blocks
  Language syntax for declaring a function "on the fly."
- Grand Central Dispatch
  C API for leveraging blocks to make writing multithreaded code much easier.

Blocks

- What is a block?

  A block of code (i.e. a sequence of statements inside {}).
  Usually included "in-line" with the calling of method that is going to use the block of code.
  Very smart about local variables, referenced objects, etc.

- What does it look like?

  Here's an example of calling a method that takes a block as an argument.

```
[aDictionary enumerateKeysAndObjectsUsingBlock:^(id key, id value, BOOL *stop) {
    NSLog(@"value for key %@ is %@", key, value);
    if ([@"ENOUGH" isEqualToString:key]) {
        *stop = YES;
    }
}];
```

  This NSLog()s every key and value in aDictionary (but stops if the key is ENOUGH).

- Blocks start with the magical character caret ^

  Then it has (optional) arguments in parentheses, then {, then code, then }.

Blocks

- Can use local variables declared before the block inside the block

```
double stopValue = 53.5;
[aDictionary enumerateKeysAndObjectsUsingBlock:^(id key, id value, BOOL *stop) {
    NSLog(@"value for key %@ is %@", key, value);
    if ([@"ENOUGH" isEqualToString:key] || ([value doubleValue] == stopValue)) {
        *stop = YES;
    }
}];
```

- But they are read only!

```
BOOL stoppedEarly = NO;
double stopValue = 53.5;
[aDictionary enumerateKeysAndObjectsUsingBlock:^(id key, id value, BOOL *stop) {
    NSLog(@"value for key %@ is %@", key, value);
    if ([@"ENOUGH" isEqualToString:key] || ([value doubleValue] == stopValue)) {
        *stop = YES;
        stoppedEarly = YES;  // ILLEGAL
    }
}];
```

Blocks

- Unless you mark the local variable as __block

```
__block BOOL stoppedEarly = NO;
double stopValue = 53.5;
[aDictionary enumerateKeysAndObjectsUsingBlock:^(id key, id value, BOOL *stop) {
    NSLog(@"value for key %@ is %@", key, value);
    if ([@"ENOUGH" isEqualToString:key] || ([value doubleValue] == stopValue)) {
        *stop = YES;
        stoppedEarly = YES;  // this is legal now
    }
}];
if (stoppedEarly) NSLog(@"I stopped logging dictionary values early!");
```

- Or if the variable is an instance variable

  Because instance variables are really just a special case of an object being accessed in the block.

  Let's talk some more about that …

Blocks

- So what about objects accessed inside the block?

```
NSString *stopKey = [@“Enough” uppercaseString];
__block BOOL stoppedEarly = NO;
double stopValue = 53.5;
[aDictionary enumerateKeysAndObjectsUsingBlock:^(id key, id value, BOOL *stop) {
    NSLog(@“value for key %@ is %@”, key, value);
    if ([stopKey isEqualToString:key] || ([value doubleValue] == stopValue)) {
        *stop = YES;
        stoppedEarly = YES;  // this is legal now
    }
}];
if (stoppedEarly) NSLog(@“I stopped logging dictionary values early!”);
```

stopKey is automatically retained until the block goes out of scope or the block itself is released.

Why does that matter?

And what does it mean for “the block itself to be released?”

Blocks

- Imagine we added the following method to `CalculatorBrain`

  `- (void)addUnaryOperation:(NSString *)operation whichExecutesBlock:...;`

  This method adds another operation to the brain like `sqrt` which you get to specify the code for.
  For now, we'll not worry about the syntax for passing the block.
  (but the mechanism for that is the same as for defining `enumerateKeysAndObjectsUsingBlock:`).

- That block we pass in will not be executed until much later

  i.e. it will be executed when that "operation" is pressed in some UI somewhere.

- Example call of this …

```
NSNumber *secret = [NSNumber numberWithDouble:42.0];
[brain addUnaryOperation:@"MoLtUaE" whichExecutesBlock:^(double operand) {
    return operand * [secret doubleValue];
}];
```

  Imagine if `secret` was not automatically `retain`ed here.
  What would happen later when this block executed (when `MoLtUaE` operation was pressed)?
  Bad things. Luckily, `secret` is automatically `retain`ed.

Blocks

- How would we define that method?

  Blocks are kind of like "objects" with an unusual syntax for declaring variables that hold them.

  Usually if we are going to store a block in a variable, we typedef a type for that variable, e.g.,

```
typedef double (^unary_operation_t)(double op);
```

  This declares a type called "unary_operation_t" for variables which can store a block.
  (specifically, a block which takes a double as its only argument and returns a double)

  Then we could declare a variable, square, of this type and give it a value …

```
unary_operation_t square;
square = ^(double operand) {
    return operand * operand;
}
```

  And then use the variable square like this …

```
double squareOfFive = square(5.0);  // squareOfFive would have the value 25.0 after this
```

  (You don't have to typedef, for example, the following is also a legal way to create square …)

```
double (^square)(double op) = ^(double op) { return op * op; };
```

Blocks

- We could then use the unary_operation_t to define our method

```
typedef double (^unary_operation_t)(double op);
- (void)addUnaryOperation:(NSString *)op whichExecutesBlock:(unary_operation_t)opBlock {
    [operationDictionary setObject:opBlock forKey:op];
}
```

Notice that we can treat the block somewhat like an object (adding it to a dictionary, in fact).
The only "messages" we might send to a block, though, are copy, retain, release or autorelease.
Unfortunately, blocks are allocated initially on the stack (they're not really "objects" in that way).
To get a heap-allocated block, we'd send [opBlock copy] as our argument to setObject:forKey:.
We'd also want to autorelease that copy (since it gets retained by the dictionary).

Later in our CalculatorBrain we could use an operation added with the method above like this ...

```
- (double)performOperation:(NSString *)operation
{
    unary_operation_t unaryOp = [operationDictionary objectForKey:operation];
    if (unaryOp) {
        self.operand = unaryOp(self.operand);
    }
    . . .
}
```

Blocks

- Back to our calling of this method

```
NSNumber *secret = [NSNumber numberWithDouble:42.0];
[brain addUnaryOperation:@“MoLtUaE” whichExecutesBlock:^(double operand) {
    return operand * [secret doubleValue];
}];
```

We said earlier that the object `secret` will be `retain`ed until the `block` is `release`d.
So when will this `block` be `release`d?
The `block` will be `release`d if and when `CalculatorBrain` removes it from its `operationDictionary`.
Or when the `CalculatorBrain` is `release`d (it will `release` `operationDictionary` in its `dealloc`).

As you might expect, if you access an instance variable in your `block`, `self` will be `retain`ed.

Blocks

- Back to blocks as method arguments

  When a block is an argument to a method and is used immediately, often there is no typedef.

  Here is the declaration of the dictionary enumerating method we showed earlier …

  ```
  - (void)enumerateKeysAndObjectsUsingBlock:(void (^)(id key, id obj, BOOL *stop))block;
  ```

  Notice, no typedef for this block.

  The syntax is exactly the same as the typedef except that the name of the typedef is not there.

  For reference, here's what a typedef for this argument would look like this …

  ```
  typedef void (^enumeratingBlock)(id key, id obj, BOOL *stop);
  ```

  (i.e. the underlined part is not used in the method argument)

Blocks

- Some shorthand allowed when defining a block

  ("Defining" means you are writing the code between the {}.)

  You do not have to declare the return type if it can be inferred from your code in the block.
  If there are no arguments to the block, you do not need to have any parentheses.
  Recall this code (no return type, see?):

```
NSNumber *secret = [NSNumber numberWithDouble:42.0];
[brain addUnaryOperation:@"MoLtUaE" whichExecutesBlock:^(double operand) {
    return operand * [secret doubleValue];
}];
```

- Another example …

```
[UIView animateWithDuration:5.0 animations:^{
    view.opacity = 0.5;
}];
```

  No arguments, so ^{ } is all that is needed.

Blocks

- When do we use blocks in iOS?

  Enumeration
  View Animations (more on that later in the course)
  Sorting (sort this thing using a block as the comparison method)
  Notification (when something happens, execute this block)
  Error handlers (if an error happens while doing this, execute this block)
  Completion handlers (when you are done doing this, execute this block)

- And a super-important use: Multithreading

  With Grand Central Dispatch API

Grand Central Dispatch

- GCD is a C API
- The basic idea is that you have queues of operations
  The operations are specified using blocks.
  Most queues run their operations serially (a true "queue").
  We're only going to talk about serial queues today.
- The system runs operations from queues in separate threads
  Though there is no guarantee about how/when this will happen.
  All you know is that your queue's operations will get run (in order) at some point.
  The good thing is that if your operation blocks, only that queue will block.
  Other queues will continue to run.
- So how can we use this to our advantage?
  Get blocking activity (e.g. network) out of our user-interface (main) thread.
  Do time-consuming activity concurrently in another thread.

Grand Central Dispatch

- Important functions in this C API

  Creating and releasing queues

```
dispatch_queue_t dispatch_queue_create(const char *label, NULL);
void dispatch_release(dispatch_queue_t);
```

  Putting blocks in the queue

```
typedef void (^dispatch_block_t)(void);
void dispatch_async(dispatch_queue_t queue, dispatch_block_t block);
```

  Getting the current or main queue

```
dispatch_queue_t dispatch_get_current_queue();
dispatch_queue_t dispatch_get_main_queue();
```

Grand Central Dispatch

- What does it look like to call these?

  Example … let's make our Flickr fetch of an image in PhotoViewController work properly.

```
- (void)viewWillAppear:(BOOL)animated
{
    NSData *imageData = [FlickrFetcher imageDataForPhotoWithURLString:photo.URL];
    UIImage *image = [UIImage imageWithData:imageData];
    self.imageView.image = image;
    self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
    self.scrollView.contentSize = image.size;
}
```

Grand Central Dispatch

- What does it look like to call these?

  Example … let's make our Flickr fetch of an image in PhotoViewController work properly.

```
- (void)viewWillAppear:(BOOL)animated
{
    dispatch_queue_t downloadQueue = dispatch_queue_create("Flickr downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [FlickrFetcher imageDataForPhotoWithURLString:photo.URL];
        UIImage *image = [UIImage imageWithData:imageData];
        self.imageView.image = image;
        self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
        self.scrollView.contentSize = image.size;
    });
}
```

Problem! UIKit calls can only happen in the main thread!

Grand Central Dispatch

- What does it look like to call these?

  Example … let's make our Flickr fetch of an image in PhotoViewController work properly.

```
- (void)viewWillAppear:(BOOL)animated
{
    dispatch_queue_t downloadQueue = dispatch_queue_create("Flickr downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [FlickrFetcher imageDataForPhotoWithURLString:photo.URL];
        dispatch_async(dispatch_get_main_queue(), ^{
            UIImage *image = [UIImage imageWithData:imageData];
            self.imageView.image = image;
            self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
            self.scrollView.contentSize = image.size;
        });
    });
}
```

Grand Central Dispatch

- What does it look like to call these?

Example … let's make our Flickr fetch of an image in PhotoViewController work properly.

```
- (void)viewWillAppear:(BOOL)animated
{
    dispatch_queue_t downloadQueue = dispatch_queue_create("Flickr downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [FlickrFetcher imageDataForPhotoWithURLString:photo.URL];
        dispatch_async(dispatch_get_main_queue(), ^{
            UIImage *image = [UIImage imageWithData:imageData];
            self.imageView.image = image;
            self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
            self.scrollView.contentSize = image.size;
        });
    });
}
```

Problem! `NSManagedObjectContext` is not thread safe,
so we can't call `photo.URL` in `downloadQueue`'s thread!

Grand Central Dispatch

- What does it look like to call these?

  Example … let's make our Flickr fetch of an image in PhotoViewController work properly.

```
- (void)viewWillAppear:(BOOL)animated
{
    NSString *url = photo.URL;
    dispatch_queue_t downloadQueue = dispatch_queue_create("Flickr downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [FlickrFetcher imageDataForPhotoWithURLString:url];
        dispatch_async(dispatch_get_main_queue(), ^{
            UIImage *image = [UIImage imageWithData:imageData];
            self.imageView.image = image;
            self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
            self.scrollView.contentSize = image.size;
        });
    });
}
```

Grand Central Dispatch

- What does it look like to call these?

Example … let's make our Flickr fetch of an image in PhotoViewController work properly.

```
- (void)viewWillAppear:(BOOL)animated
{
    NSString *url = photo.URL;
    dispatch_queue_t downloadQueue = dispatch_queue_create("Flickr downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [FlickrFetcher imageDataForPhotoWithURLString:url];
        dispatch_async(dispatch_get_main_queue(), ^{
            UIImage *image = [UIImage imageWithData:imageData];
            self.imageView.image = image;
            self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
            self.scrollView.contentSize = image.size;
        });
    });
}
```

Problem! This leaks. We need to release the `downloadQueue`.

Grand Central Dispatch

- What does it look like to call these?

  Example … let's make our Flickr fetch of an image in PhotoViewController work properly.

```
- (void)viewWillAppear:(BOOL)animated
{
    NSString *url = photo.URL;
    dispatch_queue_t downloadQueue = dispatch_queue_create("Flickr downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [FlickrFetcher imageDataForPhotoWithURLString:url];
        dispatch_async(dispatch_get_main_queue(), ^{
            UIImage *image = [UIImage imageWithData:imageData];
            self.imageView.image = image;
            self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
            self.scrollView.contentSize = image.size;
        });
    });
    dispatch_release(downloadQueue);  // won't actually go away until queue is empty
}
```

Coming Up

- Demo
  Add a PhotoViewController to Shutterbug
  Stop it from blocking the main thread
- Homework
  Current homework still due on Wednesday
  Next homework might be assigned next Tuesday, due the following Monday
- Next Lecture
  CoreLocation and MapKit