## Announcements

- **Lab 5 is due tonight at midnight**
- **We will start the project update presentations on Wednesday**
  - Present in the same order as the project proposal



## Today's Topics

- **Memory Usage**
  - Blocks
  - System warnings
- **Concurrency**
  - Threads
  - Operations and queues
- **Audio**

## Memory on the iPhone

- **Starting points for performance**
  - Load lazily
  - Don't leak
  - Watch your autorelease footprint
  - Reuse memory
- **System memory warnings are a last resort**
  - Respond to warnings or be terminated



## Loading Lazily

- **Pervasive in Cocoa frameworks**
- **Do only as much work as is required**
  - Application launch time!
- **Think about where your code really belongs**
- **Use multiple NIBs for your user interface**
  - If you are not using Storyboard

## Loading a Resource Too Early

- What if it's not needed until much later? Or not at all?

```
- (id)init
{
    self = [super init];
    if (self) {
    // Too early...
    myImage = [self readSomeHugeImageFromDisk];
    }
return self;
}
```



## Loading a Resource Lazily

- Wait until someone actually requests it, then create it

```
- (UIImage *) myImage {
   if (myImage == nil) {
     myImage = [self readSomeHugeImageFromDisk];
   }
}
```

- Ends up benefiting both memory and launch time
- Not always the right move, consider your specific situation
- Notice that the above implementation is not thread-safe!

## Autorelease and You

- **Autorelease simplifies your code**
  - Worry less about the scope and lifetime of objects
- **When an autorelease pool pops, it calls release on each object**
- **An autorelease pool is created automatically for each iteration of your application's run loop**



## So What's the Catch?

- **What if many objects are autoreleased before the pool pops?**
- **Consider the maximum memory footprint of your application**

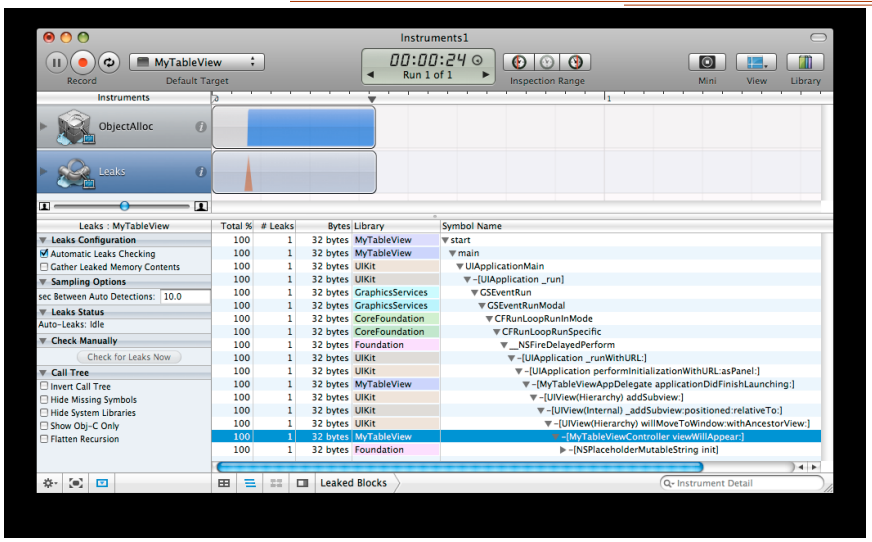
## A Crowded Pool...




## Reducing Your High-Water Mark

- **When many objects will be autoreleased, create and release your own pool**
  - Usually not necessary, don't do this without thinking!
  - Tools can help identify cases where it's needed
  - Loops are the classic case

# Memory Allocation and Leaks






# Demo

## Autorelease in a Loop

- Remember that many methods return autoreleased objects

```
for (int i = 0; i < someLargeNumber; i++) {
    NSString *string = ...;
    string = [string lowercaseString];
    string = [string stringByAppendingString:...];
    NSLog(@"%@", string);
}
```



## (Old Way) Creating an Autorelease Pool

- One option is to create and release for each iteration

```
for (int i = 0; i < someLargeNumber; i++) {
   NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

   NSString *string = ...;
   string = [string lowercaseString];
   string = [string stringByAppendingString:...];
   NSLog(@"%@", string);

  [pool release];
}
```

## (New Way) Creating an Autorelease Pool

- One option is to create and release for each iteration

```
for (int i = 0; i < someLargeNumber; i++) {
  @autoreleasepool {

    NSString *string = ...;
    string = [string lowercaseString];
    string = [string stringByAppendingString:...];
    NSLog(@"%@", string);

  }
}
```



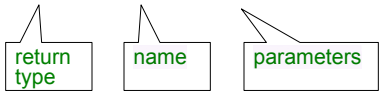
## Blocks

- A variable type that stores executable code
- Lets you create "blocks" of code to pass around like an object
- Example

```
^{
   NSDate *date = [NSDate date];
   NSLog(@"The date and time is %@",date);
 };
```

# Assigning Blocks to Variables

```
void  (^now)  (void)
```




```
void (^now) (void) = ^{
   NSDate *date = [NSDate date];
   NSLog(@"The date and time is %@",date);
 };

now();
```



# Blocks

- Blocks are closures
  - They close around variables in scope when the block is declared

```
NSDate *date = [NSDate date];

void (^now)(void) =  ^{
     NSLog(@"The date and time is %@",date);
 };

now();

sleep(5);

date = [NSDate date];

now();
```

# Demo



# Object Creation Overhead

- Most of the time, creating and deallocating objects is not a significant hit to application performance
- In a tight loop, though, it can become a problem...

```
for (int i = 0; i < someLargeNumber; i++) {
    MyObject *object = [[MyObject alloc] initWithValue:...];

    [object doSomething];
    [object release];
}
```

## Reusing Objects

- Update existing objects rather than creating new ones
- Combine intuition and evidence to decide if it's necessary

```
MyObject *myObject = [[MyObject alloc] init];

for (int i = 0; i < someLargeNumber; i++) {
    myObject.value = ...;
    [myObject doSomething];
}

[myObject release];
```

- Remember -[UITableView dequeueReusableCellWithIdentifier]



## Memory Warnings

- Coexist with system applications
- Memory warnings issued when memory runs out
- Respond to memory warnings or face dire consequences!



# Responding to Memory Warnings

- Every view controller gets –didReceiveMemoryWarning
  - By default, releases the view if it's not visible
  - Release other expensive resources in your subclass

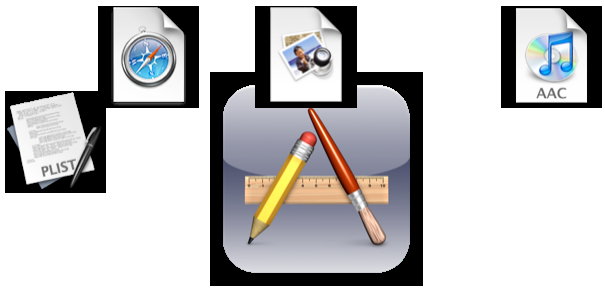
```
- (void)didReceiveMemoryWarning
{
    // Always call super
    [super didReceiveMemoryWarning];

    // Release expensive resources
    [expensiveResource release];
    expensiveResource = nil;
}
```



# What Other Resources Do I Release?

- Images
- Sounds
- Cached data



## Use SQLite for Large Data Sets

- Many data formats keep everything in memory
- SQLite can work with your data in chunks



# Concurrency

## Why Concurrency?

- With a single thread, long-running operations may interfere with user interaction
- Multiple threads allow you to load resources or perform computations without locking up your entire application



## Threads on the iPhone

- Based on the POSIX threading API
  - /usr/include/pthread.h
- Higher-level wrappers in the Foundation framework

## NSThread Basics

- **Run loop automatically instantiated for each thread**
  - Each NSThread needs to create its own autorelease pool
  - Convenience methods for messaging between threads



## (Old way) Typical NSThread Use Case

```
- (void)someAction:(id)sender
{
    // Fire up a new thread
    [NSThread detachNewThreadSelector:@selector(doWork:)
          withTarget:self object:someData];
}

- (void)doWork:(id)someData
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    [someData doLotsOfWork];

    // Message back to the main thread
    [self performSelectorOnMainThread:@selector(allDone:)
    withObject:[someData result] waitUntilDone:NO];

    [pool release];
}
```

## (New way) Typical NSThread Use Case

```
- (void)someAction:(id)sender
{
    // Fire up a new thread
    [NSThread detachNewThreadSelector:@selector(doWork:)
          withTarget:self object:someData];
}

- (void)doWork:(id)someData
{
  @autoreleasepool {
    [someData doLotsOfWork];

    // Message back to the main thread
    [self performSelectorOnMainThread:@selector(allDone:)
    withObject:[someData result] waitUntilDone:NO];

  }
}
```



## UIKit and Threads

- **Unless otherwise noted, UIKit classes are not threadsafe**
  - What does it mean to be thread safe?
  - Objects must be created and messaged from the main thread

# Demo: Threads and Xcode



# Locks

- Protect critical sections of code, mediate access to shared data
- NSLock and subclasses

```
-(void)init{
  myLock = [[NSLock alloc] init];
}

-(void)someMethod
{
    [myLock lock];
    // We only want one thread executing this code at once
    [myLock unlock]
}
```

## Conditions

- NSCondition is useful for producer/consumer model

```
// On the producer thread
- (void)produceData
{
  [condition lock];

  // Produce new data
  newDataExists = YES;

  [condition signal];
  [condition unlock];
}
```

```
// On the consumer thread
- (void)consumeData
{
  [condition lock];
  while(!newDataExists) {
    [condition wait];
  }

  // Consume the new data
  newDataExists = NO;

  [condition unlock];
}
```

- Wait is equivalent to: unlock, sleep until signalled, lock



## The Danger of Locks

- Very difficult to get locking right!
- All it takes is one client poorly behaved client
  - Accessing shared data outside of a lock
  - Deadlocks
  - Priority inversion

## Threading Pitfalls

- Subtle, nondeterministic bugs may be introduced
- Code may become more difficult to maintain
- In the worst case, more threads can mean slower code



## Alternatives to Threading

- Asynchronous (nonblocking) functions
  - Specify target/action or delegate for callback
  - NSURLConnection has synchronous and asynchronous variants
- Timers
  - One-shot or recurring
  - Specify a callback method
  - Managed by the run loop
- Higher level constructs like operations

## NSOperation

- Abstract superclass
- Manages thread creation and lifecycle
- Encapsulate a unit of work in an object
- Specify priorities and dependencies



## NSOperationQueue

- Operations are typically scheduled by adding to a queue
- Choose a maximum number of concurrent operations
- Queue run operations based on priority and dependencies

## More on Concurrent Programming

- Grand Central Dispatch (GCD)
- "Threading Programming Guide"
- https://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/Multithreading



## Blocks with Grand Central Dispatch (GCD)

- Blocks are closures
  - They close around variables in scope when the block is declared

```
NSDate *date = [NSDate date];

void (^now)(void) =  ^{
    sleep(5);
    NSDate *date = [NSDate date];
    NSLog(@"The date and time is %@",date);
  };

//now();
dispatch_async(dispatch_get_main_queue(),now);

NSLog (@"The original date is %@",date);
```

# Audio



## Uses for Audio

- **Sound effects**
  - button clicks
  - alert sounds
  - short sounds accompanying user actions
- **Arbitrary length sounds (music, podcasts, spoken content)**
- **Streamed content from web services**
- **Recording audio**

## How to do it?

- **Could be complex:**
  - Potentially multiple simultaneous sources
  - Numerous possible outputs
  - Dynamic events, often out of user's control
  - Different priorities for seemingly similar actions
- **The OS manages the sound system**
  - You can ask for behavior, but the OS has control



## CoreAudio

- **High level, easy to use**
  - System Sound API - short sounds
  - AVAudioPlayer class - ObjC, simple API
- **Lower level, takes more effort but much more control**
  - Audio Toolbox - recording and playback, streaming, full control
  - Audio Units - processing audio
  - OpenAL - 3D positional sound
- **Which one you use depends on what you're trying to do**
  - Many of you are fine with System Sounds and AVAudioPlayer

## Playing Short Sounds

- "short" means less than 5 seconds
- Very simple API, but has restrictions
  - No looping
  - No volume control
  - Immediate playback
  - Limited set of formats
  - Linear PCM or IMA4
  - .caf, .aif or .wav file



## Playing Short Sounds

- Two step process
  - Register the sound, get a "sound ID" in return
  - Play the sound
  - Optionally can get callback when sound finishes playing

```
NSURL *fileURL = ... // url to a file
SystemSoundID myID;

// First register the sound
AudioServicesCreateSystemSoundID ((CFURLRef)fileURL, &myID);

// Then you can play the sound
AudioServicesPlaySystemSound (myID);
```

## Playing Short Sounds

- **Clean up**
  - Dispose of sound ID when you're done
  - Or if you get a memory warning

```
SystemSoundID myID;
// dispose of the previously registered sound
AudioServicesDisposeSystemSoundID (myID);
```



## Converting Sounds

- **Command line utility to convert sounds**
  **/usr/bin/afconvert**
- **Supports wide variety of input and output formats**
- **See man page for details**
- **Easily convert sounds to System Sounds formats**

```
/usr/bin/afconvert -f aiff -d BEI16 input.mp3 output.aif
```

# AVAudioPlayer

- Play longer sounds (> 5 seconds)
- Locally stored files or in-memory (no network streaming)
- Can loop, seek, play, pause
- Provides metering
- Play multiple sounds simultaneously
- Cocoa-style API
  - Initialize with file URL or data
  - Allows for delegate
- Supports many more formats
  - Everything the AudioFile API supports



# AVAudioPlayer

- Create from file URL or data
- Simple methods for starting/stopping

```
AVAudioPlayer *player;
NSString *path = [[NSBundle mainBundle] pathForResource...];
NSURL *url = [NSURL fileURLWithPath:path];
player = [[AVAudioPlayer alloc] initWithContentsOfURL:url];

if (!player.playing) {
    [player play];
} else {
    [player pause];
}
```

## AVAudioPlayerDelegate

- Told when playback finishes
- Informed of audio decode errors
- Given hooks for handling interruptions
  - Incoming phone calls



## OpenAL

- High level, cross-platform API for 3D audio mixing
  - Great for games
  - Mimics OpenGL conventions
- Models audio in 3D space
  - Buffers: Container for Audio
  - Sources: 3D point emitting Audio
  - Listener: Position where Sources are heard
- More Information: http://www.openal.org/

## Playing Video

- **Uses for Video:**
  - Provide cut-scene animation in a game
  - Stream content from web sites
  - Play local movies
- **Play videos from application bundle or remote URL**
  - Always full screen
  - Configurable scaling modes
  - Optional controls
- **Supports:**
  - .mov, .mp4, .m4v, .3gp



## Audio Demo