

Using Concurrency To Improve The Responsiveness of iPhone Applications

Danton Chin

danton@iphonedevjournal.com

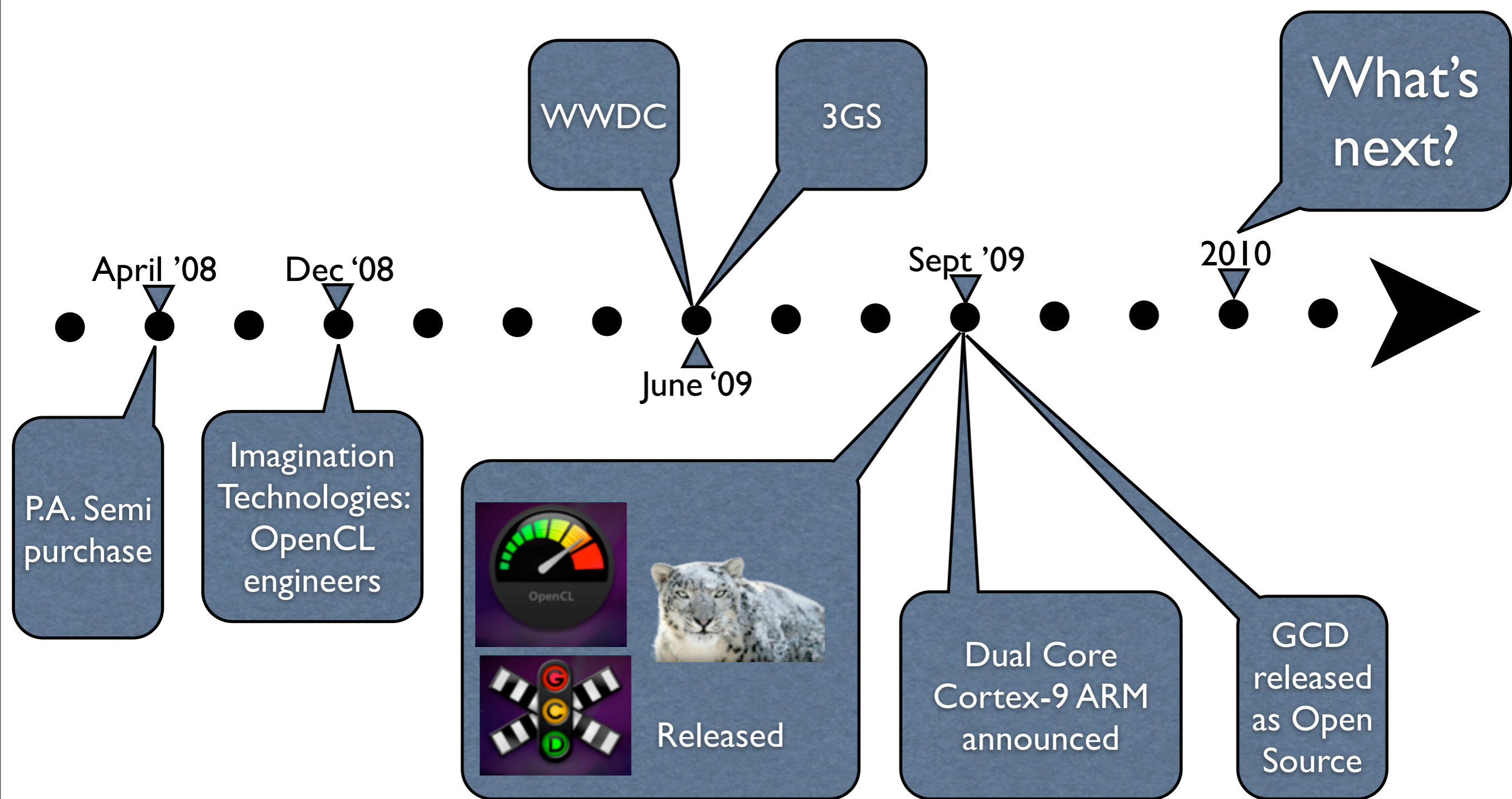
<http://iphonedevjournal.com/>

Agenda

- Motivation and Predictions
- Surveying the Concurrency Landscape
- Operations and Queues
- NSInvocationOperation
- NSOperationQueue
- NSOperation
- Resources

Motivation and Predictions

Connect The Dots



images from <http://www.apple.com/>

Writing for Multiple Platforms



2G, 3G, 3GS,
iPod Touch

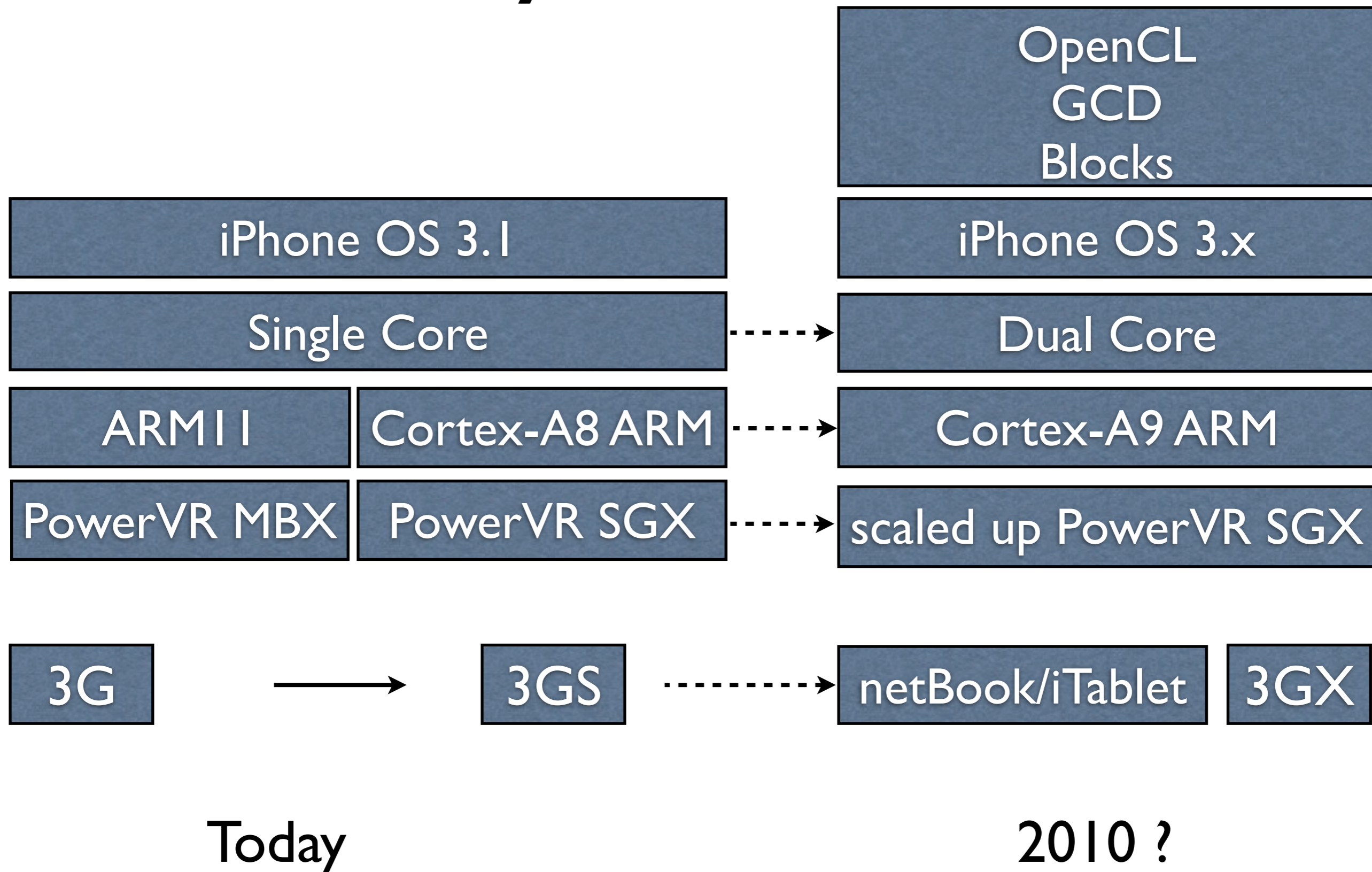
3GX ?

netBook/iTablet?

Today

2010

Today and Tomorrow?



Surveying The Concurrency Landscape

Improving Performance Using Concurrency

- Multiple threads in a single-core system can be used to improve ***perceived*** performance while a multi-core system can produce real performance improvements
- Since the iPhone at this point only has one core threads are used to improve perceived performance
- Use concurrency and asynchronous functions to avoid blocking the main thread

Concurrency is Not Free

- Concurrent programming can be difficult and error prone
- Potential problems with race conditions, deadlocks, synchronization, and so forth
- Creation costs, memory costs

Grand Central Dispatch



- Introduced in Snow Leopard
- New approach to taking advantage of multiple cores
- Shifts responsibility of managing threads and the execution of jobs from the application developer to the OS using units of work (blocks) and queues
- Introduced with a language extension and new APIs

Blocks or Closures



- Extension to Objective-C (C, C++ too) to make it easy to define self-contained units of work or tasks
- Are Objective-C objects but look like function pointers

x = ^{ NSLog(@"An NSLog statement in a block"); }

Although the iPhone does not have an official implementation from Apple -- yet -- there is Plausible Blocks an implementation from Landon Fuller at <http://code.google.com/p/plblocks/>



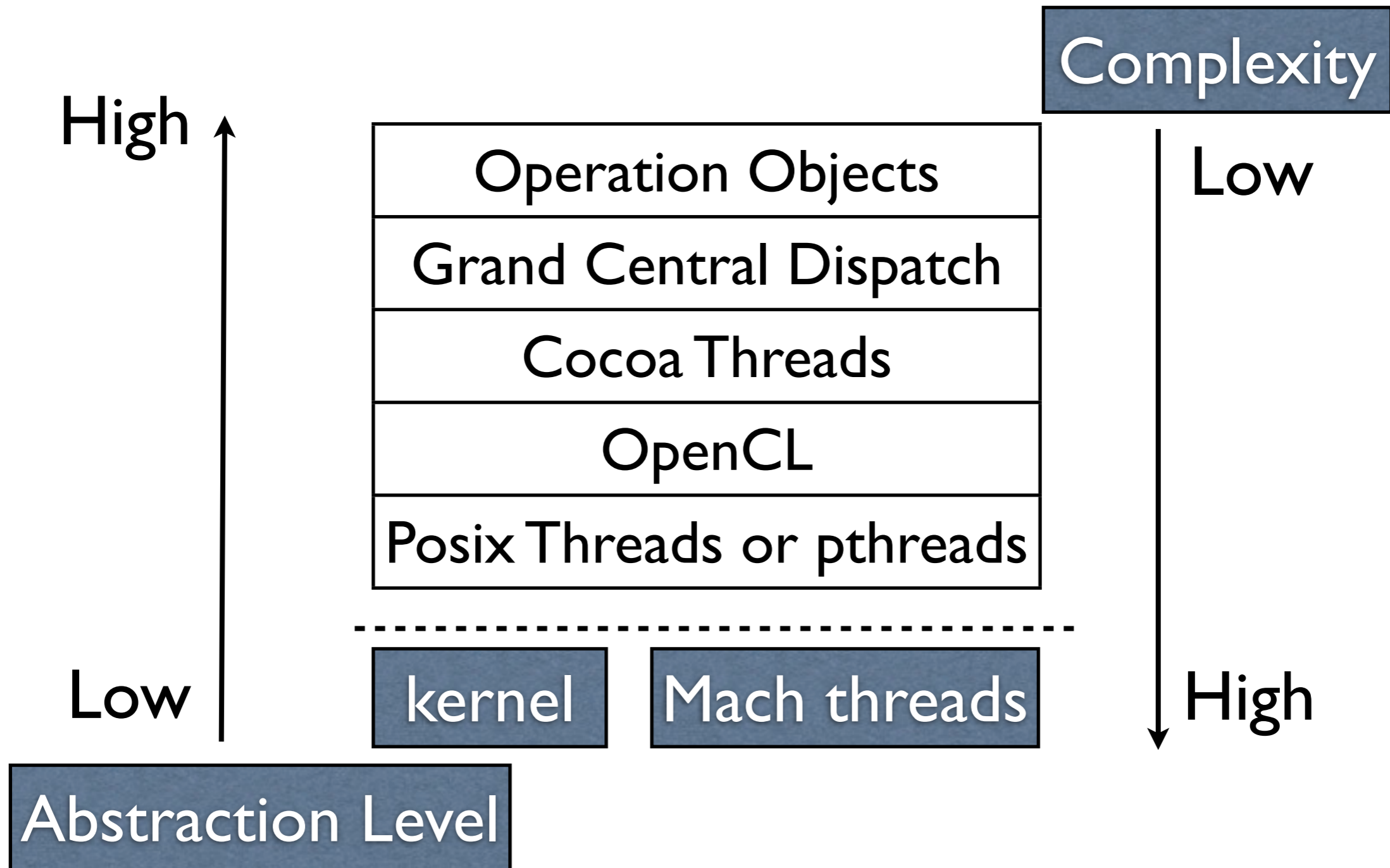
OpenCL

- OpenCL or Open Computing Language
- Low-level C-API for general purpose parallel computations using all the computational resources available -- CPU, GPU, etc.

Concurrency Choices

Technology	Description	iPhone OS	Mac OS X
Operation Objects	NSOperation objects are wrappers for tasks NSOperationQueue objects manages the execution of tasks	Yes	10.5+
Grand Central Dispatch	Framework to transfer the responsibility of managing threads and their execution to the OS. Uses blocks and queues	No	10.6
Cocoa Threads	NSThread NSObject	Yes	Yes
OpenCL	C-level API to use all the computational resources available	No	10.6
Posix Threads or pthreads	C-based interface for creating threads	Yes	Yes

Two Metrics



Alternatives To Concurrency

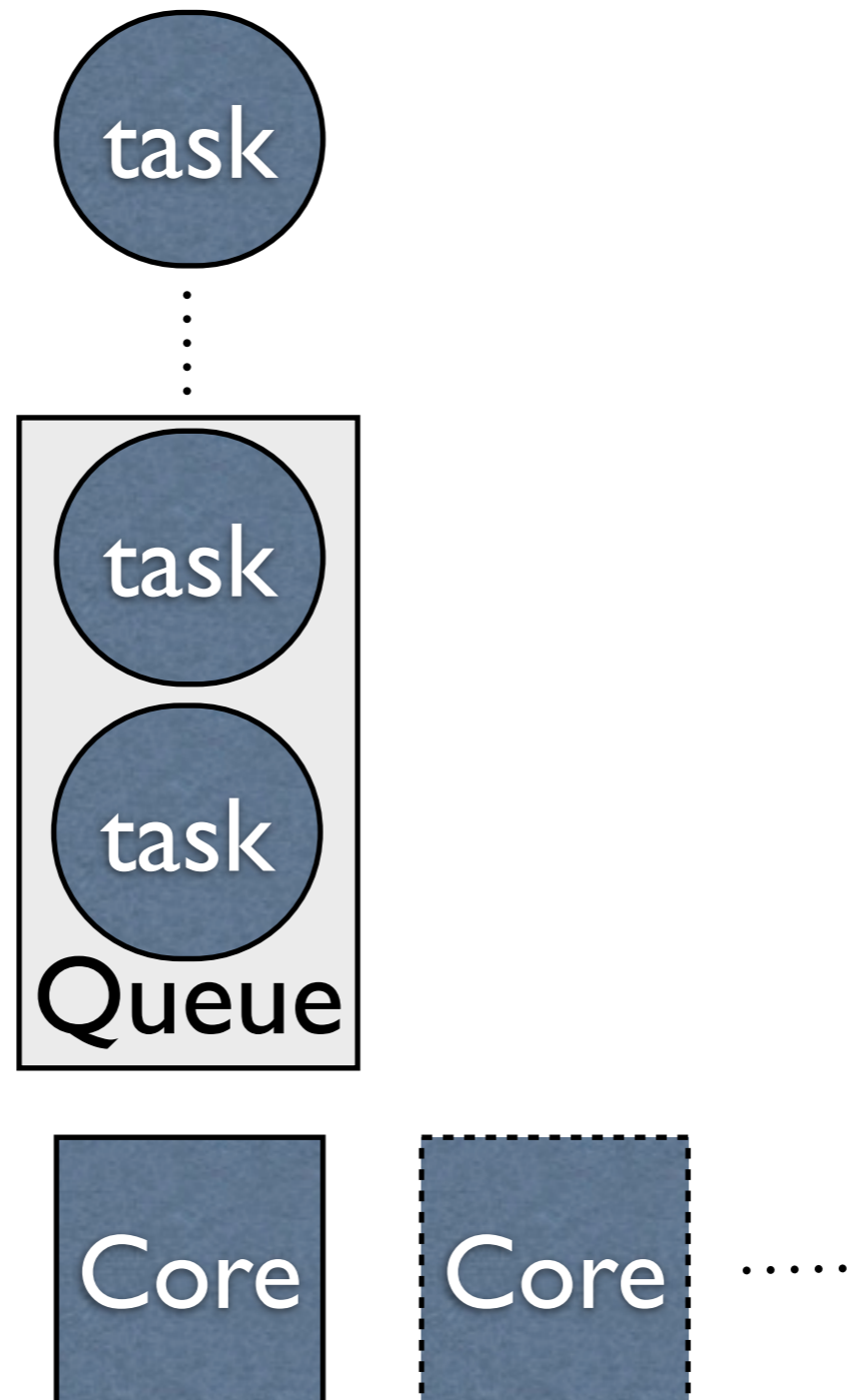
Technology	Description	iPhone OS	Mac OS X
Asynchronous methods	Some APIs have both synchronous and asynchronous versions of the same method	Yes	Yes
Idle-time notifications	Use NSNotificationQueue with a posting style of NSPostWhenIdle to receive notifications when the run loop becomes idle	Yes	Yes
Timers	Use Timers to perform periodic, discrete tasks	Yes	Yes

Operations and Queues

Task

- Tasks are independent work units
 - with no dependencies on the state or result of other tasks

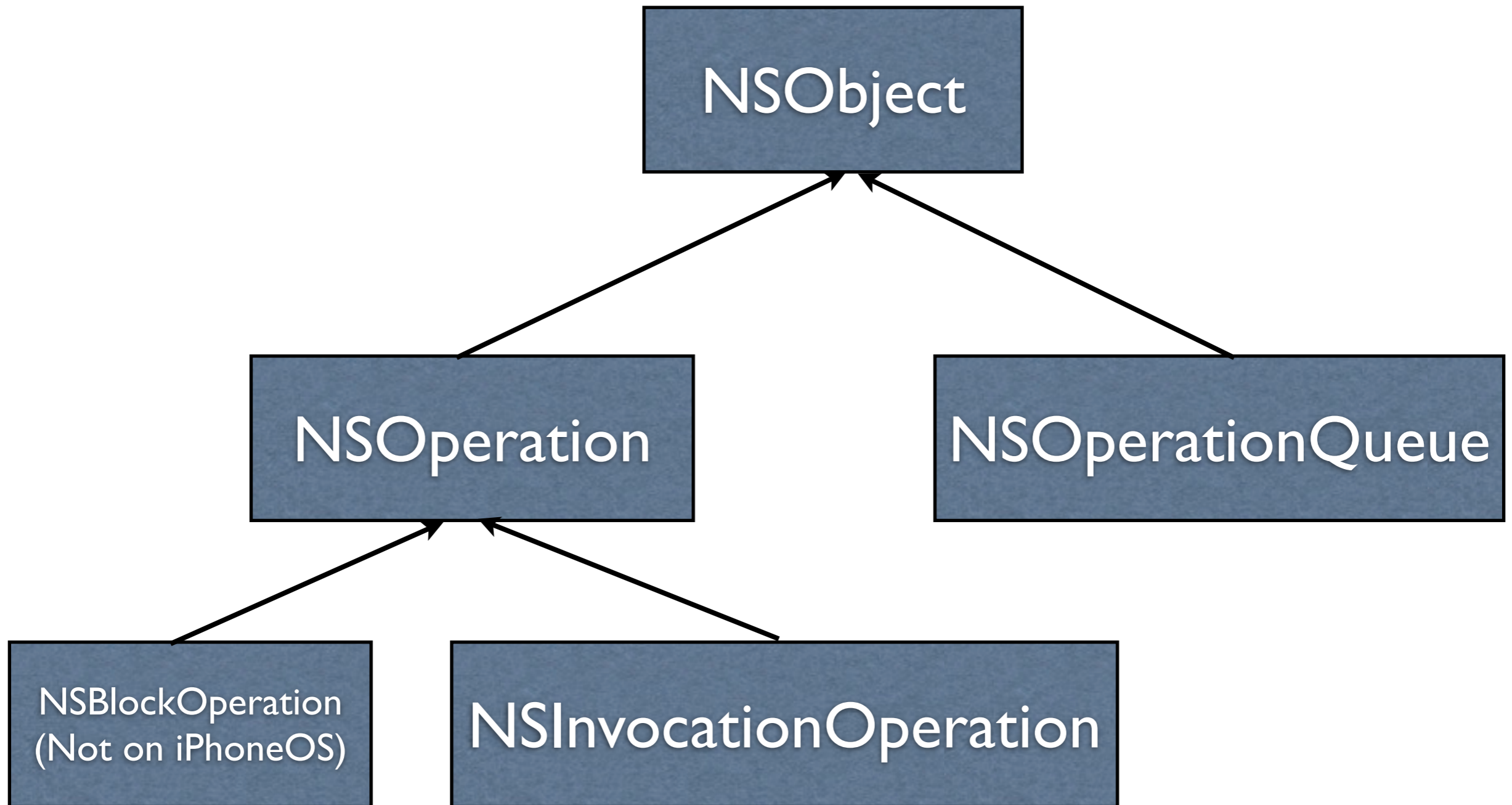
Operations and Queues



Asynchronous Approach

- Look at the work performed by your application
- factor out the work that can be done concurrently vs work that must be done serially due to shared data structures

Class Hierarchy



Operation Objects

Class	Description	iPhone OS	Mac OS X
NSOperation	Abstract class used to create/define custom operation objects	Yes	Yes
NSInvocationOperation	Lightweight operation - uses an object and a selector from your app to create an operation	Yes	Yes
NSBlockOperation	Use this class to execute one or more blocks concurrently	No	Yes
NSOperationQueue	Tasks are added to NSOperationQueue and the queue manages the execution of the operation objects	Yes	Yes

NSInvocationOperation

NSInvocationOperation

- Concrete subclass of NSOperation
- Easiest way to define operation objects to be added to an operation queue for execution
 - provided the application has a method that encapsulates the task
- creates an non-concurrent operation

Advantages of Using NSInvocationOperation

- Quick and easy
 - allows you to just use your classes which have self-contained tasks in an operation queue
- Can be dynamic
- Eliminates creating custom operation objects

Creating an NSInvocationOperation

```
-(id)initWithTarget:(id)target selector:(SEL)sel object:(id)arg
```

target - object that defines the selector

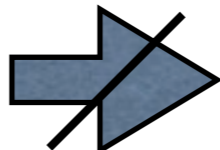
sel - selector to be invoked; selector may take 0-1 parameters whose type must be id. The method may return void, a scalar value, or an object that can be returned as an id

arg - parameter object to be passed to the selector; specify nil if the selector does not take an argument



returns: initialized NSInvocationOperation or nil if target does not implement sel

NSOperationQueue

NSOperationQueue

- Operations are added to an operation queue to be managed by the queue for execution
- Queues work with the system to determine the number of concurrent operations
 - $f(\text{number of cores, system load})$
 - more queues  more operations

NSOperationQueue KVO properties

property	returns
operations	an array containing the operations currently in the queue
operationCount	number of operations in queue 
maxConcurrentOperationCount	the max number of operations the receiver can execute as an NSInteger
suspended	whether queue is suspended
name	queue name 

Creating an NSOperationQueue

Create just like any other object:

```
NSOperationQueue *myQueue = [ [NSOperationQueue alloc] init];
```

Adding Operations To A Queue

Use `NSOperationQueue` method:

```
-(void)addOperation:(NSOperation *)operation
```

```
[myQueue addOperation:myOp];
```

- An operation can only be in one queue at a time
- There is no *removeOperation:* method

Note. Docs state that you can add an array of operations but that has not been implemented in iPhone OS 3.0 only on Mac OS 10.6

Canceling Operations

Can cancel all operations in a queue

```
-(void)cancelAllOperations
```

a cancel message is sent to all operations in the queue

To cancel a single operation call its ***cancel*** method

Cancellation and KVO

A “canceled” operation is considered “finished” and dependent operations still receive KVO notification that the dependency has been fulfilled

Because of this it is more common to cancel all operations rather than a single operation

Suspending/Resuming a Queue

-(void)setSuspended:(BOOL)suspend

YES, queue stops scheduling operations for execution

NO, queue starts scheduling operations again

Suspending a queue does not suspend a currently executing operation

Max number of Concurrent Operations

Use

```
-(void)setMaxConcurrentOperationCount:(NSInteger)count
```

order of execution still depends on the readiness of each operation and its assigned priority

Demo

HelloWorldSimpleInvocation

NSOperation

NSOperation

- abstract class
- subclass to encapsulate the data and methods belonging to a task

KVO properties

can use these KVO properties to control your app

isCancelled	whether operation has been cancelled
isConcurrent	whether the operation runs asynchronously
isExecuting	whether the operation is currently executing
isFinished	whether the operation is done executing
isReady	whether the operation can be performed
dependencies	read-only
queuePriority	read-write

Operation Dependencies

- an operation object can have dependencies
- use dependencies to serialize execution
- operation object will not run until all its dependent operation objects have “finished executing”
 - remember that a canceled operation is considered to have “finished executing”
- configure an operations dependencies before adding it to a queue

Adding and Removing Dependencies

Use:

```
-(void)addDependency:(NSOperation *)operation
```

```
-(void)removeDependency:(NSOperation *)operation
```

Operation Priorities

To set the relative priority of an operation when the operation is added to an operation queue use:

```
-(void)setQueuePriority:(NSOperationQueuePriority)priority
```

NSOperationQueuePriorityVeryLow	-8
NSOperationQueuePriorityLow	-4
NSOperationQueuePriorityNormal	0
NSOperationQueuePriorityHigh	4
NSOperationQueuePriorityVeryHigh	8

Execution Order

- Operation priorities are applied to all operations that are **ready** to execute

not ready
Op A
high priority

ready
Op B
low priority

Op B will be executed before Op A

Two Ways To Use a Subclass of NSOperation

- have an operation queue execute the operation after it is added to the queue
- manually execute the operation by calling the operation object's **start** method

Non-Concurrent Operations

- NSOperation objects are non-concurrent by default, i.e.,

isConcurrent = NO

- Let the operation queue manage concurrency for the operation
- In a non-concurrent operation the operation's task is performed synchronously, i.e., the **operation object** does not create a new thread to perform the task

Non-Concurrent Operations

- When a non-concurrent operation is added to an operation queue the queue creates a thread on which to run the operation which allows the operation queue to move on to the next operation
- Result:
 - asynchronous execution of your operation

Concurrent Operations

isConcurrent = YES

- the operation object will manage how it wants to handle concurrency, either by
 - starting a thread or
 - calling an asynchronous function

In iPhone OS operation queues launch concurrent operations from the current thread



In Snow Leopard operation queues use GCD and both concurrent and non-concurrent operations are launched from a separate thread.

Creating Custom NSOperation Objects

NSOperation Queue handles concurrency	non-concurrent custom object	if operation is always added to an operation queue then operation is executed asynchronously
You handle concurrency	concurrent custom object	if operation needs to be run asynchronously without adding it to an operation queue

Creating a Non-Concurrent Subclass

- Need to implement:
 - init to handle any initialization for your task
 - main to execute your task
 - check for cancellation
 - create an autorelease pool to manage objects that are autoreleased

Responding To Cancellations

- Responding to cancellations is voluntary so operation objects must frequently check for cancellation, clean up memory, and exit
- Use ***isCancelled*** method to check
 - before doing any work
 - at least once before each iteration and more frequently for long running iterations
 - anywhere where it would be easy to abort

Demo

HelloWorldNonConcurrentOp
SimpleInvocationInCocoa

Creating a Concurrent Subclass

Methods to override:

start	<ul style="list-style-type: none">• set up the thread or execution environment in which the task will run• must not call super
main	implement task
isExecuting isFinished	<ul style="list-style-type: none">• maintain state information for task• generate KVO notifications and be thread safe
isConcurrent	<ul style="list-style-type: none">• return YES

Demo

Resources

- Apple Documentation
 - Concurrency Programming Guide
 - NSInvocationOperation Class Reference
 - NSOperation Class Reference
 - NSOperationQueue Class Reference

Thank You!

Have Fun With Operations and Queues!

Danton Chin

danton@iphonedevjournal.com