Technology

#STUFFMEETING

Concurrency on Darwin

Advanced usage of NSOperations

Yannick Heinrich iOS Developer

Summary

Concurrency vs Parallelism

Threads

Run Loops

GCD

NSOperation

Advanced NSOperations

Frameworks

Sample Code

Concurrency vs Parallelism

Concurrency vs Parallelism

Concurrency

Programming as the composition of independently executing processes.

Source: Rob Pike - Concurrency is Not Parallelism

Concurrency vs Parallelism

Parallelism

Programming as the simultaneous execution of (possibly related) computations.

Source: Rob Pike - Concurrency is Not Parallelism

Concurrency vs Parallelism

Conclusion

Concurrency is about dealing with lots of things at once.

Parallelism is about doing lots of things at once.

Not the same, but related.

Concurrency is about structure, parallelism is about execution.

Concurrency provides a way to structure a solution to solve a problem that may (but not necessarily) be parallelizable.

Source: Rob Pike - Concurrency is Not Parallelism

Concurrency vs Parallelism

Darwin tools

`NSThread`, `NSLock`, `OSAtomic`, `@synchronize`, `pthread`, ...

NSRunLoop

GCD

`NSOperation`

Source: Rob Pike - Concurrency is Not Parallelism

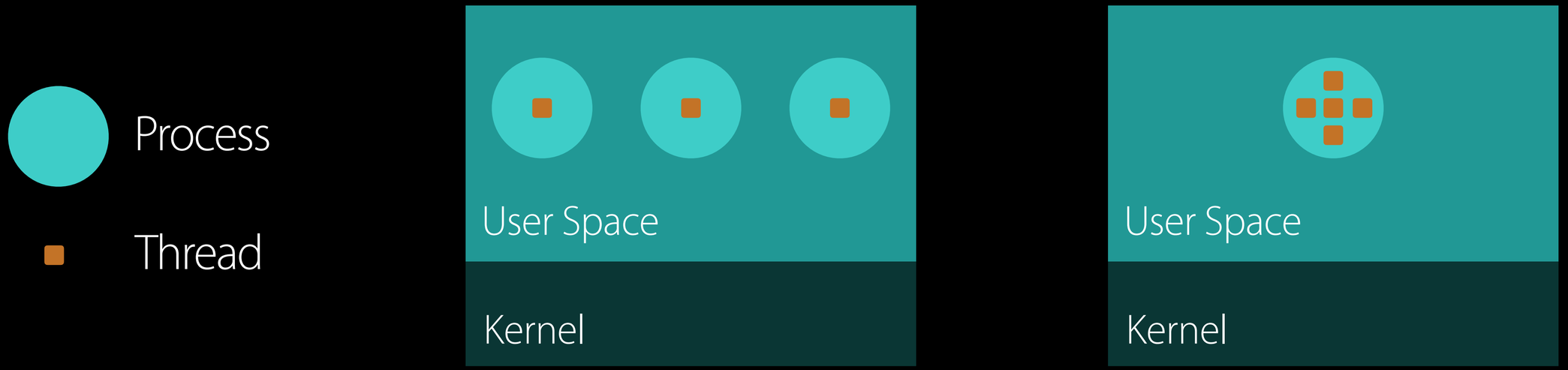
Threads

Threads

Processes and Threads

Process: unit having its own virtual memory

Thread: sub-unit of process that shared virtual memory



Threads

Scheduler

The OS allocates a specific amount of time to each unit

Each unit can be stopped, woken up, created and destroyed

Time allocation strategy is generally Round-Robin

Each unit consumes memory in Kernel space

Context switching and thread creation cost times.

Item	Approximate Cost
Kernel data structures	Approximately 1 KB
Stack space	512 KB (secondary threads) 8 MB (OS X main thread 1 MB (iOS main thread)
Creation time	Approximately 90 microseconds

Source:Thread Programming Guide

Threads

Producer/Consumer

```
/* Producer */
id condLock = [[NSConditionLock alloc] initWithCondition:NO_DATA];

while(true)
{
    [condLock lock];
    /* Add data to the queue. */
    [condLock unlockWithCondition:HAS_DATA];
}

/* Consumer */
while (true)
{
    [condLock lockWhenCondition:HAS_DATA];
    /* Remove data from the queue. */
    [condLock unlockWithCondition:(isEmpty ? NO_DATA : HAS_DATA)];

    // Process the data locally.
}
```

Threads

Download asynchronous URL in background, get notified in main thread

```objc
@implementation AFURLConnectionOperation
/* Call in NSThread initWithTarget:selector:object */

+ (void)networkRequestThreadEntryPoint:(id)__unused object {
    @autoreleasepool {
        [[NSThread currentThread] setName:@"AFNetworking"];

        NSRunLoop *runLoop = [NSRunLoop currentRunLoop];
        [runLoop addPort:[NSMachPort port] forMode:NSDefaultRunLoopMode];
        [runLoop run];
    }
}

- (void)start
{
    /* ... */
    [self performSelector:@selector(operationDidStart)
                 onThread:[[self class] networkRequestThread] withObject:nil];
    /* ... */
}

- (void)operationDidStart {
    self.connection = [[NSURLConnection alloc] initWithRequest:self.request
                                                      delegate:self startImmediately:NO];

    NSRunLoop *runLoop = [NSRunLoop currentRunLoop];
    for (NSString *runLoopMode in self.runLoopModes) {
        [self.connection scheduleInRunLoop:runLoop forMode:runLoopMode];
    }

    [self.connection start];
}
```

Threads

Drawbacks

You may feel reinventing the wheel

Boilerplate code

Synchronisation with the main thread ?

Code not always easy to read/understand

Easy to enter in race conditions

Efficiency vs speed in coding ?

Run Loop

Run Loop

Basics

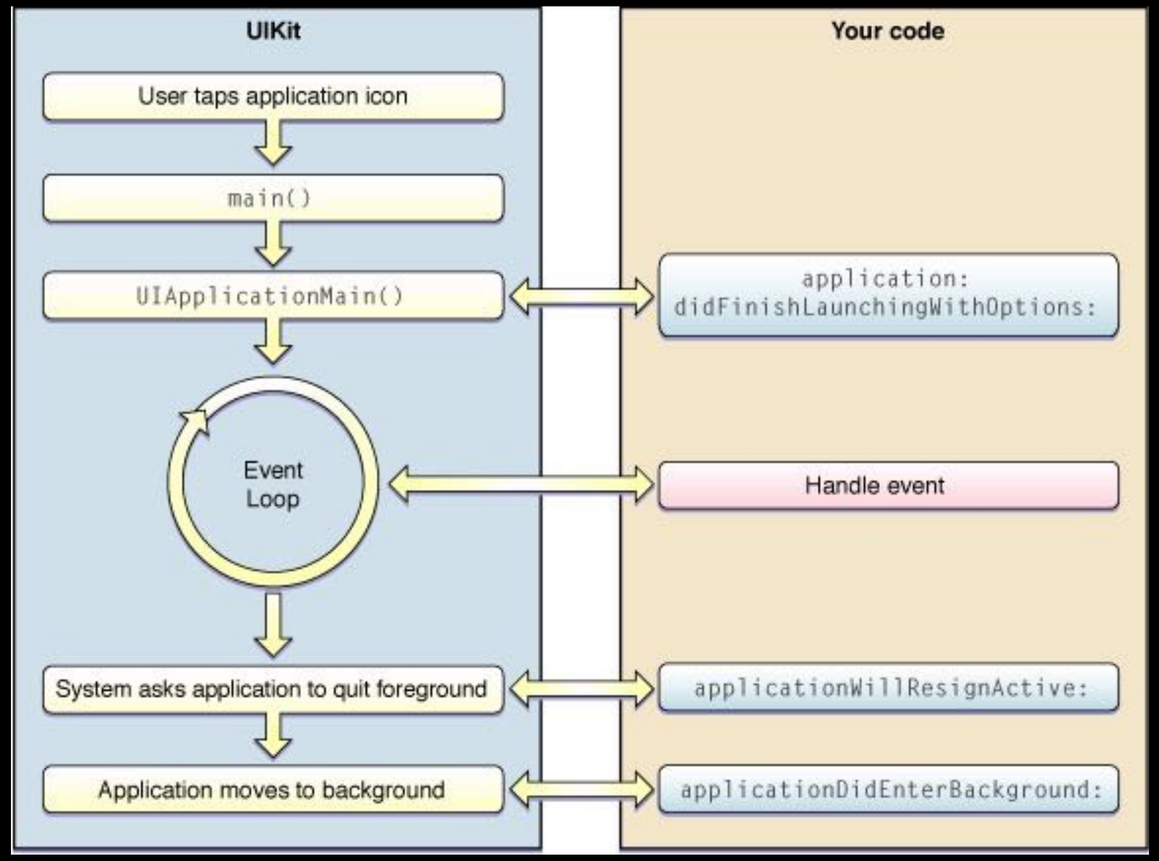
A run loop is a single threaded loop that dispatch events synchronously to theirs handlers. This thread is generally called the main thread

On BSD/Darwin, it is based on **`kqueue`**, on others systems, it relies on **`select`**, **`poll`** or **`eppoll`** system calls

```
# -> No busy wait due to system calls or kqueue

while there are still events to process:


    e = get the next event

    if there is a callback/handler associated with e:

        call the callback/handler
```

Run Loop

Main Thread



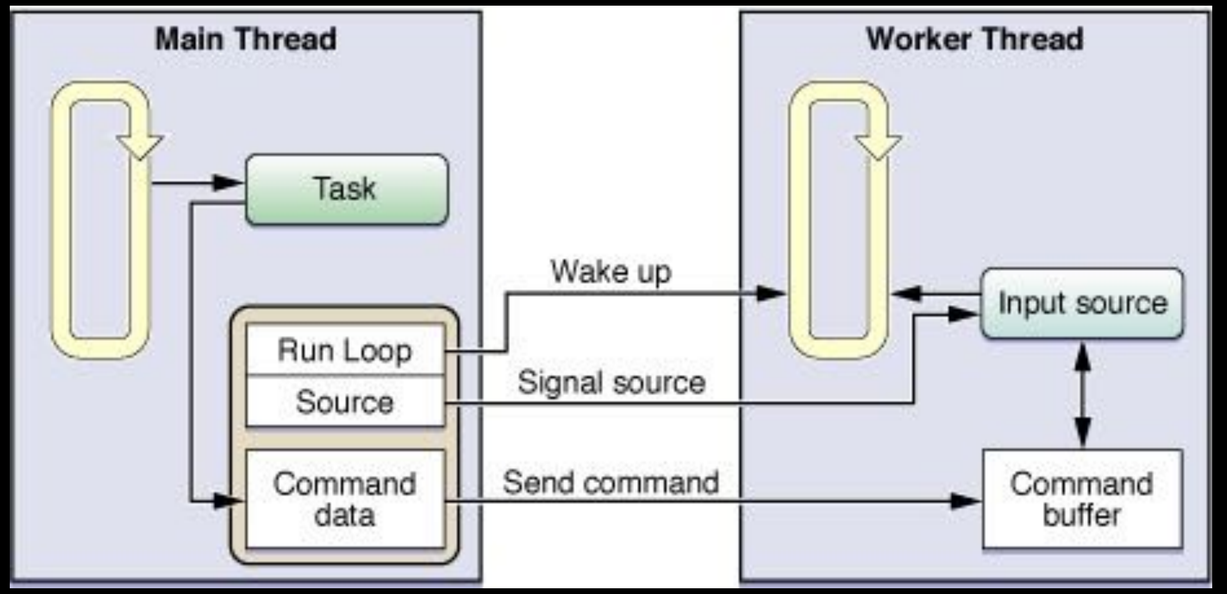
Source: Thread Programming Guide

Run Loop

Worker Thread

Handlers for events are created using the **`CFRunLoopSourceRef`** object.

For input events as mouse or keayboard events, **`NSPort`**, **`NSConnection`** and **`NSTimer`**, the corresponding **`CFRunLoopSourceRef`** are managed automatically.



Source: Thread Programming Guide

Run Loop

More Information

Reactor Pattern
en.wikipedia.org/wiki/Reactor_pattern

An introduction to **`libuv`**
http://nikhilm.github.io/uvbook/introduction.html

GCD

GCD

C libdispatch library

`libdispatch` is open source

Block oriented (Objective-C2.0)

Premption capability with priorities

Optimized synchronisation tools (I/O, Buffer, General)

Queues represented by a unique structure : `dispatch_queue_t`

Fully integrated with the debugger

GCD

Basics

Queues are concurrent or serial/FIFO

On iOS, there are 5 global queues:

- ➡ 1 serial queue for the main thread
- ➡ 4 concurrent queues with descending priorities

`QOS_CLASS_USER_INTERACTIVE`

`QOS_CLASS_USER_INITIATED`

`QOS_CLASS_UTILITY`

`QOS_CLASS_BACKGROUND`

GCD

Sample

```
/* Get the main thread queue */
dispatch_queue_t * mainQueue = dispatch_get_main_queue();

/* Get globals concurrent queue */
dispatch_queue_t * concurrent = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_HIGH,0);
dispatch_queue_t * concurrent2 = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT,0);
dispatch_queue_t * concurrent3 = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_LOW,0);
dispatch_queue_t * concurrent4 = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_BACKGROUND,0
    );

/* Create new Serial */
dispatch_queue_t * serial = dispatch_queue_create("com.company.myapp", DISPATCH_QUEUE_SERIAL );
dispatch_queue_t * serial2 = dispatch_queue_create("com.company.myapp", NULL );

/* Create new concurrent queue (Since iOS 5) */
dispatch_queue_t * concurrent5 = dispatch_queue_create("com.company.myapp",
    DISPATCH_QUEUE_CONCURRENT );
```

GCD

Groups

```
/// Main group usage

dispatch_group_t group = dispatch_group_create();

dispatch_block_t task1 = ^{ printf("Task 1\n");},
task2 = ^{ printf("Task 2\n");},
task3 = ^{
    NSInteger i;
    for(i = 0; i < INT16_MAX; ++i);
};

dispatch_queue_t queue;
queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT,0);

dispatch_group_async(group,queue,task1);
dispatch_group_async(group,queue,task2);
dispatch_group_async(group,queue,task3);

/* Wait for the oustanding tasks counter to be null*/
dispatch_group_wait(group, DISPATCH_TIME_FOREVER);

/// Manual enter/leave group
dispatch_group_enter(group);

dispatch_group_async(group,queue,^{
    [self awesome_function_with_callback:^{

        /* ... */
        dispatch_group_leave(group);
    }];

});

dispatch_group_wait(group, DISPATCH_TIME_FOREVER);

/// Dispatch end task

dispatch_group_notify(group, queue, ^{
    printf("I am called at the end !\n");
});
```

GCD

Barriers

```
/// Barrier usage

dispatch_block_t readTask1 = ^{ printf("Reading ...\n");},
readTask2 = ^{ printf("Reading ...\n");},
readTask3 = ^{ printf("Reading ...\n");},
writeTask4 = ^{printf("Writing \n");};

dispatch_queue_t queue;
queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT,0);

dispatch_async(queue,readTask1);
dispatch_async(queue,readTask2);

dispatch_barrier_async(queue,writeTask4);

dispatch_async(queue,readTask3);
```

GCD

Further

Lot of other features:

Target Queues, Semaphores, Dispatch sources, Queue Data, Dispatch I/O

Documentations:

- Concurrency Programming Guide
- NSBlog - Topics on GCD
- CocoaSamurai - Guide to blocks and grand central dispatch
- WWDC 2011 - Mastering Grand Central Dispatch
- WWDC 2010 - Introducing Blocks and Grand Central Dispatch on iPhone
- WWDC 2010 - Simplifying iPhone App Development with Grand Central Dispatch

GCD

Drawbacks

C API

To many options for simple problems

No cancellations

NSOperation

NSOperation

Higher-level of abstraction

Objective-C API build over GCD

Ready to use with Apple Framework : `UIKit`, `AVFoundation`, `MapKit`, `OpenGL`, . . .

Small API - Enough for most basic cases

Supports cancellation

All magic hidden behind `NSOperation` and `NSOperationQueue`

Fully integrated with the debugger

NSOperation

Basics

```
/// Creations

/* Get Main Queue */
NSOperationQueue * mainQueue = [NSOperationQueue mainQueue];

/* Create Serial Queue */
NSOperationQueue * serialQueue = [[NSOperationQueue alloc] init];
serialQueue.name = @"com.company.app.serial";
serialQueue.maxConcurrentOperationCount = 1;

/* Create Concurrent Queue */
NSOperationQueue * concurrentQueue = [[NSOperationQueue alloc] init];

/// Synchronisation

NSOperation* op = [[NSOperation alloc] init];

[concurrentQueue addOperationWithBlock:^{ printf("Hello \n");}];
[concurrentQueue addOperationWithBlock:^{ printf("World \n");}];

[concurrentQueue addOperation:op];

[concurrentQueue waitUntilAllOperationsAreFinished];

// Dependencies

NSBlockOperation * a = [NSBlockOperation blockOperationWithBlock:^{ printf("A\n");}];
NSBlockOperation * b = [NSBlockOperation blockOperationWithBlock:^{ printf("B\n");}];
NSBlockOperation * c = [NSBlockOperation blockOperationWithBlock:^{ printf("C\n");}];

[a addDependency:b];
[b addDependency:c];

c.completionBlock = ^{ printf("End of C\n");};

NSOperationQueue * queue = [[NSOperationQueue alloc] init];

[queue addOperation:a];
[queue addOperation:b];
[queue addOperation:c];
```

NSOperation

Standart integration

1. Implement `start`, `main`, `asynchronous`, `isFinished` and `isExecuting`
2. Test for cancel events in your `main` and `start` methods when necessary
3. Add operation to a queue

Warning: Depending on OSX and iOS version, `asynchronous` and `isConcurrent` behavior may changes

NSOperation

Example

```
2016-06-16 18:23:13.042 OpTest[2033:1113783] Value 0 count:0:
2016-06-16 18:23:13.042 OpTest[2033:1113783] Value 0 count:1:
2016-06-16 18:23:13.042 OpTest[2033:1113783] Value 0 count:2:
2016-06-16 18:23:13.043 OpTest[2033:1113783] Value 0 count:3:
2016-06-16 18:23:13.043 OpTest[2033:1113783] Value 0 count:4:
2016-06-16 18:23:13.043 OpTest[2033:1113783] Value 0 count:5:
2016-06-16 18:23:13.043 OpTest[2033:1113783] Value 0 count:6:
2016-06-16 18:23:13.043 OpTest[2033:1113783] Value 0 count:7:
2016-06-16 18:23:13.043 OpTest[2033:1113783] Value 0 count:8:
2016-06-16 18:23:13.043 OpTest[2033:1113783] Value 0 count:9:
2016-06-16 18:23:13.044 OpTest[2033:1113775] Value 1 count:0:
2016-06-16 18:23:13.044 OpTest[2033:1113775] Value 1 count:1:
2016-06-16 18:23:13.044 OpTest[2033:1113775] Value 1 count:2:
2016-06-16 18:23:13.044 OpTest[2033:1113775] Value 1 count:3:
2016-06-16 18:23:13.044 OpTest[2033:1113775] Value 1 count:4:
2016-06-16 18:23:13.045 OpTest[2033:1113775] Value 1 count:5:
2016-06-16 18:23:13.045 OpTest[2033:1113775] Value 1 count:6:
2016-06-16 18:23:13.045 OpTest[2033:1113775] Value 1 count:7:
2016-06-16 18:23:13.071 OpTest[2033:1113775] Value 1 count:8:
2016-06-16 18:23:13.071 OpTest[2033:1113775] Value 1 count:9:
2016-06-16 18:23:13.071 OpTest[2033:1113807] Value 2 count:0:
2016-06-16 18:23:13.071 OpTest[2033:1113807] Value 2 count:1:
2016-06-16 18:23:13.071 OpTest[2033:1113807] Value 2 count:2:
2016-06-16 18:23:13.072 OpTest[2033:1113807] Value 2 count:3:
2016-06-16 18:23:13.072 OpTest[2033:1113807] Value 2 count:4:
2016-06-16 18:23:13.072 OpTest[2033:1113807] Value 2 count:5:
2016-06-16 18:23:13.072 OpTest[2033:1113807] Value 2 count:6:
2016-06-16 18:23:13.072 OpTest[2033:1113807] Value 2 count:7:
2016-06-16 18:23:13.072 OpTest[2033:1113807] Value 2 count:8:
2016-06-16 18:23:13.072 OpTest[2033:1113807] Value 2 count:9:
```

```objc
self.queue = [[NSOperationQueue alloc] init];
self.queue.maxConcurrentOperationCount = 1;

[self.queue addOperation:[[CustomOperation alloc] init]];
[self.queue addOperation:[[CustomOperation alloc] init]];
[self.queue addOperation:[[CustomOperation alloc] init]];
```

```objc
static int Counter = 0;

@interface CustomOperation()

@property(nonatomic, assign) int value;
@property(nonatomic, readwrite, getter=isExecuting) BOOL executing;
@property(nonatomic, readwrite, getter=isFinished) BOOL finished;
@end

@implementation CustomOperation
@synthesize finished, executing;

- (instancetype) init {

    if(self = [super init]) {
        self.name = @"Custom Operation";
        self.value = Counter;
        Counter++;
    };
    return self;
}

- (void) main {

    self.executing = YES;
    self.finished = NO;

    for(NSUInteger i = 0; i < 10; i++) {
        NSLog(@"Value %i count:%lu:", self.value, (unsigned long)i);
    }

    self.executing = NO;
    self.finished = YES;

}
@end
```

NSOperation

Example

```
self.queue = [[NSOperationQueue alloc] init];

dispatch_after(dispatch_time(DISPATCH_TIME_NOW, \
                             (int64_t)(5.0 * NSEC_PER_SEC)), dispatch_get_main_queue(), ^{
    [self.queue cancelAllOperations];
});

[self.queue addOperation:[[WaitCancelOperation alloc] init]];
[self.queue addOperation:[[WaitCancelOperation alloc] init]];
[self.queue addOperation:[[WaitCancelOperation alloc] init]];
```

```
2016-06-16 18:37:53.613 OpTest[2319:1199510] Value 2 count:0:
2016-06-16 18:37:53.613 OpTest[2319:1199497] Value 0 count:0:
2016-06-16 18:37:53.613 OpTest[2319:1199506] Value 1 count:0:
2016-06-16 18:37:55.686 OpTest[2319:1199497] Value 0 count:1:
2016-06-16 18:37:55.686 OpTest[2319:1199510] Value 2 count:1:
2016-06-16 18:37:55.686 OpTest[2319:1199506] Value 1 count:1:
2016-06-16 18:37:57.735 OpTest[2319:1199510] Value 2 count:2:
2016-06-16 18:37:57.735 OpTest[2319:1199506] Value 1 count:2:
2016-06-16 18:37:57.735 OpTest[2319:1199497] Value 0 count:2:
2016-06-16 18:37:59.808 OpTest[2319:1199510] Canceling operation: 2
2016-06-16 18:37:59.808 OpTest[2319:1199506] Canceling operation: 1
2016-06-16 18:37:59.808 OpTest[2319:1199497] Canceling operation: 0
```

```
static int Counter = 0;
@interface WaitCancelOperation()

@property(nonatomic, assign) int value;
@property(nonatomic, readwrite, getter=isExecuting) BOOL executing;
@property(nonatomic, readwrite, getter=isFinished) BOOL finished;
@end

@implementation WaitCancelOperation
@synthesize finished, executing;

- (instancetype) init {

    if(self = [super init]) {
        self.name = @"Wait Operation";
        self.value = Counter;
        Counter++;
    };
    return self;
}

- (void) main {

    if(self.cancelled) {
        [self finish];
    }

    self.executing = YES;
    self.finished = NO;

    for(NSUInteger i = 0; i < 10; i++) {

        if(self.cancelled) {
            [self finish];
            return;
        }

        NSLog(@"Value %i count:%lu:", self.value, (unsigned long)i);
        [NSThread sleepForTimeInterval:2.0];
    }
}

- (void) finish {
    NSLog(@"Canceling operation: %i", self.value);
    self.executing = NO;
    self.finished = YES;
}

@end
```

NSOperation

Asynchronous == NO

```
@interface AsynchronousOperation()

@property(nonatomic, readwrite, getter=isExecuting) BOOL executing;
@property(nonatomic, readwrite, getter=isFinished) BOOL finished;
@end

@implementation AsynchronousOperation
@synthesize finished, executing;


- (BOOL) isAsynchronous {
    return YES;
}

- (void) main {
    self.executing = YES;
    self.finished = NO;

    NSLog(@"Start downloading....");
    NSURL * url = [NSURL URLWithString:@"https://www.google.ch"];
    [[[NSURLSession sharedSession] dataTaskWithURL:url completionHandler:^(NSData * _Nullable data, NSURLResponse * _Nullable response
        , NSError * _Nullable error) {
        NSLog(@"Response: %@", [[NSString alloc] initWithData:data encoding:NSASCIIStringEncoding]);

        self.executing = NO;
        self.finished = YES;

    }] resume];

}
@end
```

NSOperation

Asynchronous == NO

```
self.queue = [[NSOperationQueue alloc] init];
[self.queue addOperation:[[AsynchronousOperation alloc] init]];
```

```
2016-06-16 18:51:53.204 OpTest[2610:1279152] Start downloading....
2016-06-16 18:51:55.753 OpTest[2610:1279142] Response:
sig=0_T_59Uz9QDs1k1Leu9_OeWqiXuao%3D&amp;hl=it&amp;source=homepage" data-
ved="0ahUKEwiCuM_xgK3NAhWH2CwKHQyyBWQQ2ZgBCAc">Italiano</a>  <a href="https://www.google.ch/
setprefs?sig=0_T_59Uz9QDs1k1Leu9_OeWqiXuao%3D&amp;hl=rm&amp;source=homepage" data-
ved="0ahUKEwiCuM_xgK3NAhWH2CwKHQyyBWQQ2ZgBCAg">Rumantsch</a> </div></div></div><span
id="footer"><div style=« font-size:……
```

NSOperation

Preemption

As in GCD, you can juggle with priorities:

➡ `NSOperationQueue` & `NSOperation` has 5 levels for resource access (`NSQualityOfService`)

➡NSOperation has 5 level for queuing/dequeuing (`NSOperationQueuePriority`)

For more informations on those topics:

Energy Efficiency Guide for iOS Apps

NSOperation

Drawbacks

Lot of boilerplate code

Many properties to look at

Asynchronous vs Synchronous ?

Advanced NSOperations

Advanced NSOperations

Why

WWDC2015 - Advanced NSOperations

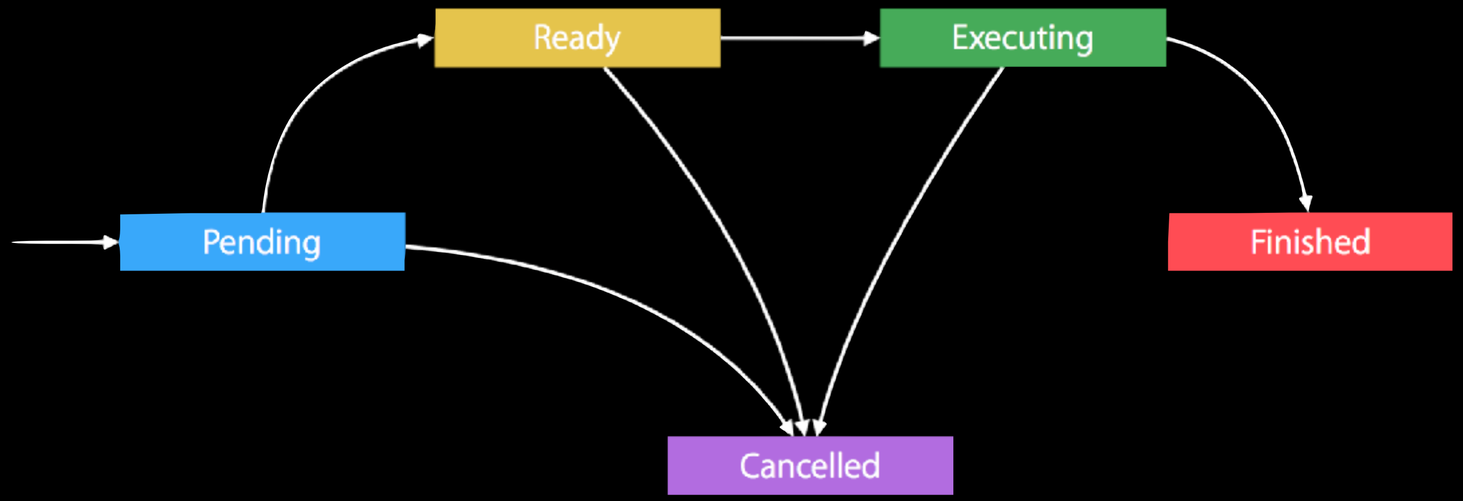
Avoid boilerplate code

Kind of pattern that could helps a lot

Improve reusability, testability and decoupling

Advanced NSOperations

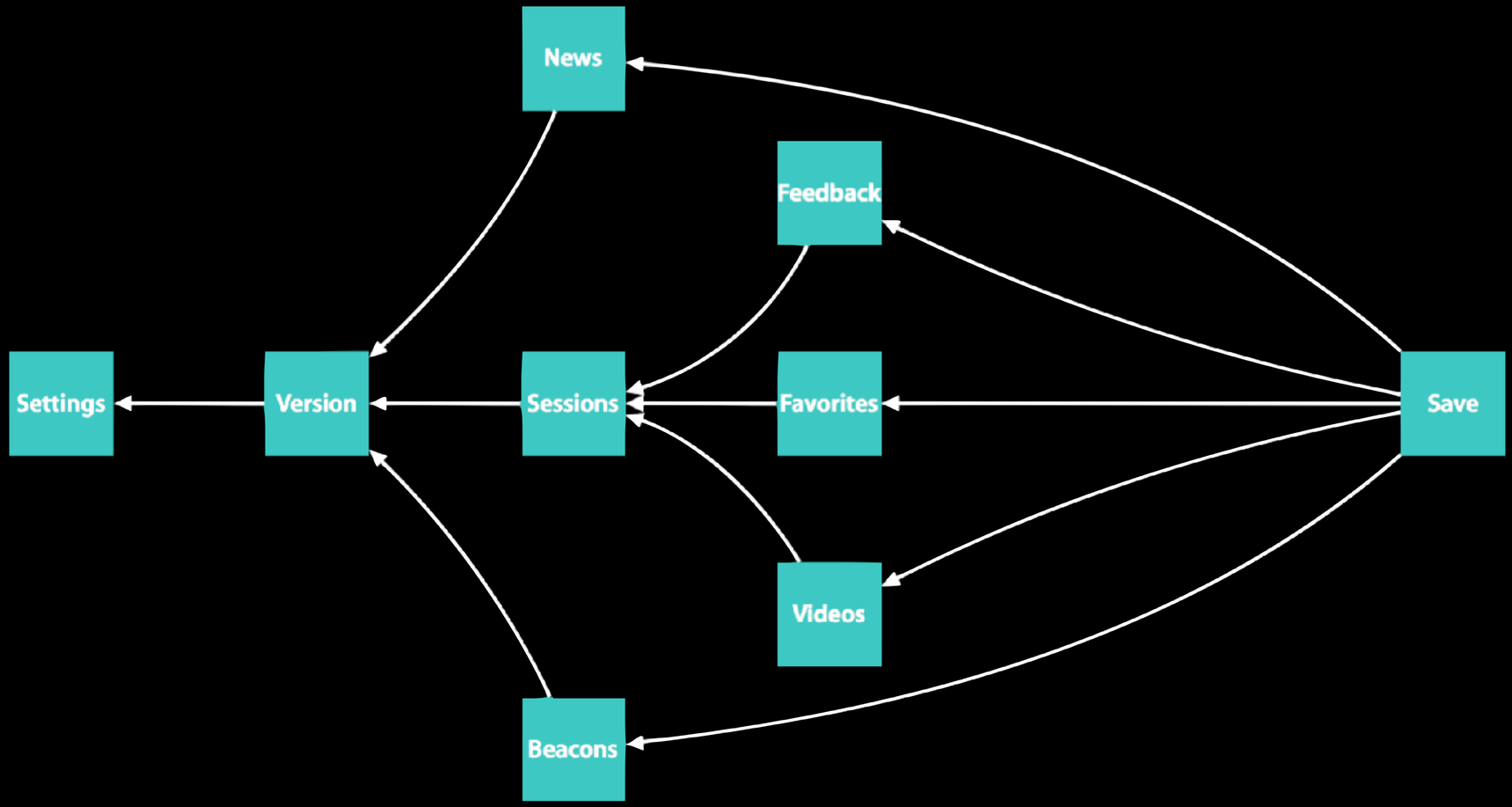
One lifecycle to run them all



Source: Dave DeLong WWDC 2015 - Advanced NSOperations

Advanced NSOperations

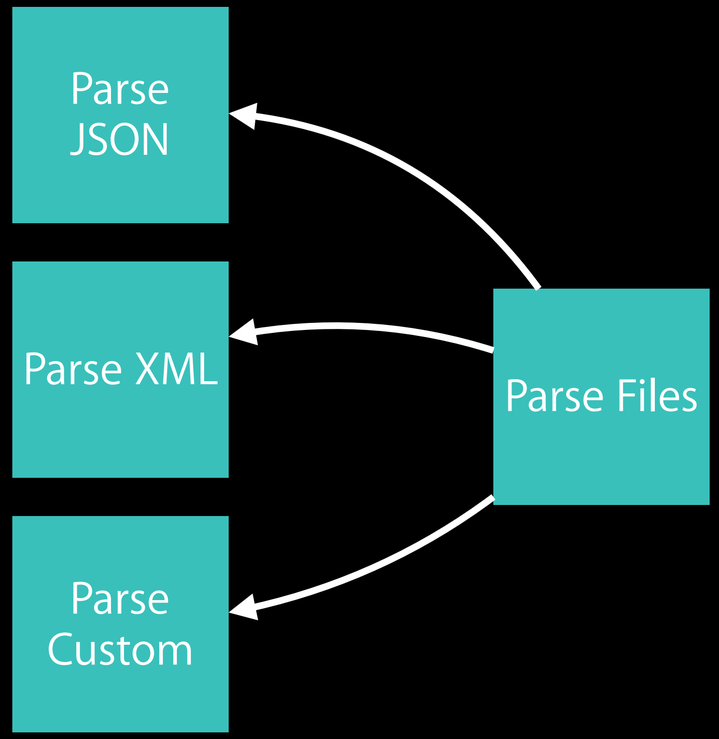
Easily create dependencies - WWDC App



Source: Dave DeLong WWDC 2015 - Advanced NSOperations

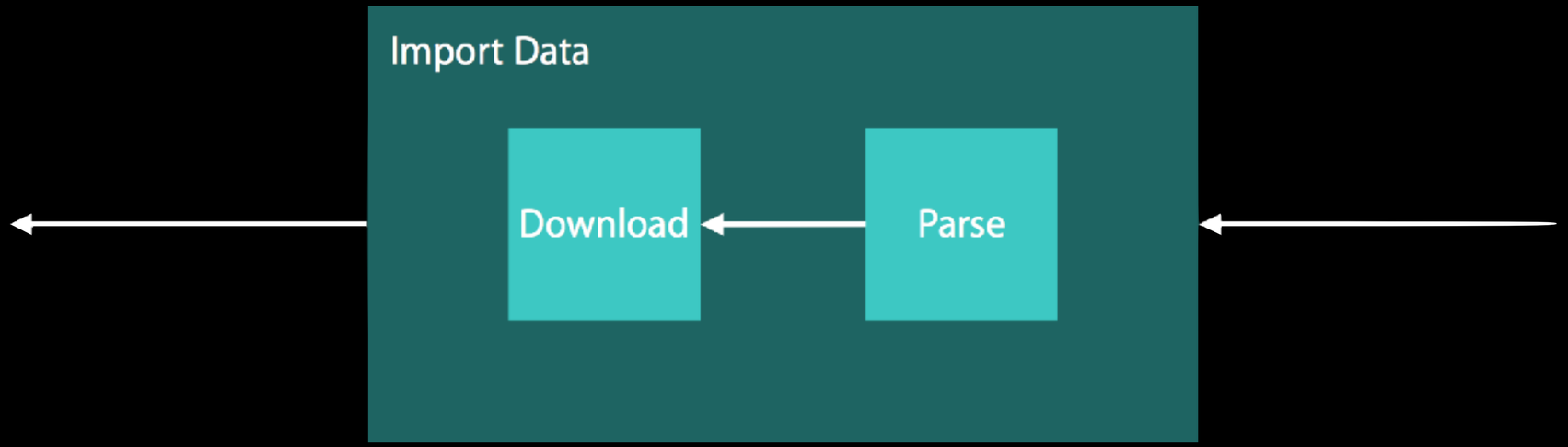
Advanced NSOperations

Generate Dependencies



Advanced NSOperations

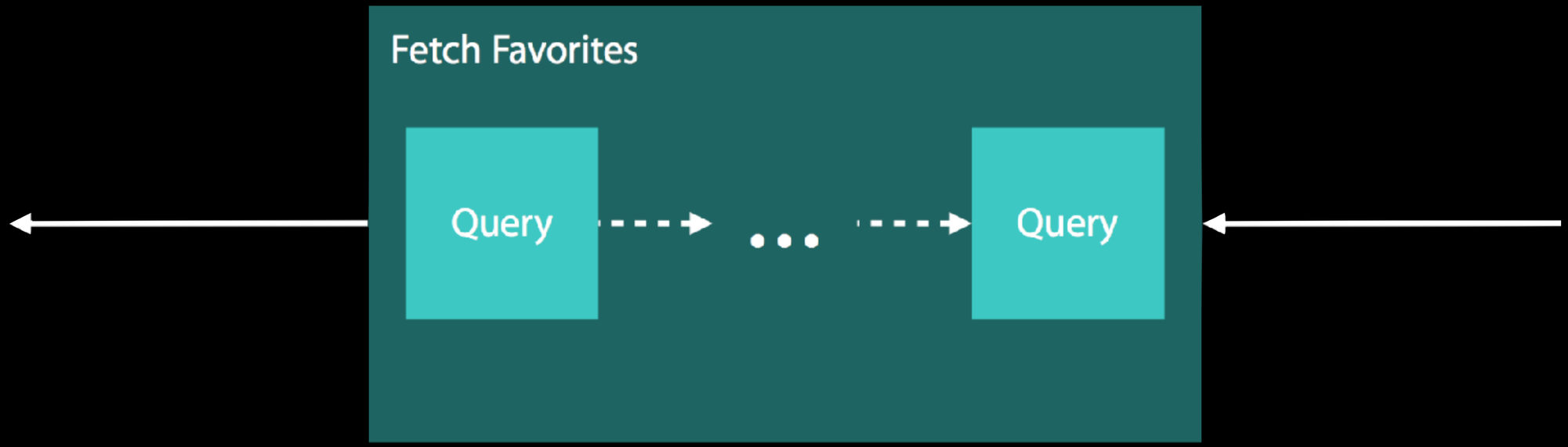
Compose Operations



Source: Dave DeLong WWDC 2015 - Advanced NSOperations

Advanced NSOperations

Compose Operations



Source: Dave DeLong WWDC 2015 - Advanced NSOperations

Advanced NSOperations

Tools

Condition

Observers

MutualExclusivity

Simplify error management

Advanced NSOperations

Advantages

Logical pieces of process are isolated

Improve reusability

Improve readability

Improve testability

Improve modularity

Statically checked

Full integration in Apple environment

Frameworks

Frameworks

Advanced NSOperations Sample Code (needs to be updated for Swift 2.2)

Operations - https://github.com/danthorpe/Operations - Swift

PSOperations - https://github.com/pluralsight/PSOperations - Swift

AdvancedNSOperations - https://github.com/purrrminator/AdvancedNSOperations - Objc

Sample Code

Sample Code

Create one Operation

```
class SimpleOperation: Operation {

    override func execute() {

    }

    override func cancel() {

    }
}
```

Sample Code

Tips for Group Operation

Passing results through chained operations or GroupOperation :

- ➡ Use implicit elements: HDD File on disk (See Earthquakes `GetEarthquakesOperation`)
- ➡ Use singleton
- ➡ Create context object (See MachineKit Bluetooth Operations)

More Information

Advanced NSOperations
WWDC2015
developer.apple.com/wwdc

Sample Framework
danthorpe
github.com/danthorpe/operations