

iAsync

Functional Programming in Objective-C

Alexander Dodatko
2013





Over 900,000 apps

(as of June 2013)







```
NSData* jsonData = nil;  
jsonData = [ NSData dataWithContentsOfURL: RESOURCE ];
```


That's not
good





Asynchronous Programming

@protocol NSDataDelegate
<NSURLConnectionDelegate>

@optional

- (void)connection: didReceiveResponse:
- (void)connection: didReceiveData:
- (void)connectionDidFinishLoading:
- (void)connection: didFailWithError:

@end

Too Much Code

Objective-C Blocks





```
NS_AVAILABLE(10_7, 5_0);
```

```
+ (void)sendAsynchronousRequest:  
                                queue:  
                                completionHandler:
```



```
void (^HANDLER_BLOCK)(  
    NSURLResponse* response,  
    NSData* data,  
    NSError* connectionError)
```




Progress



Cancel



Dependencies



When using methods which return blocks they can be very convenient. However, when you have to string a few of them together it gets messy really quickly

```
[remoteAPIWithURL:url1 success:^(int status){
    [remoteAPIWithURL:url2 success:^(int status){
        [remoteAPIWithURL:url3 success:^(int status){
            [remoteAPIWithURL:url2 success:^(int status){
                //success!!!
            };
        };
    };
};
};
```



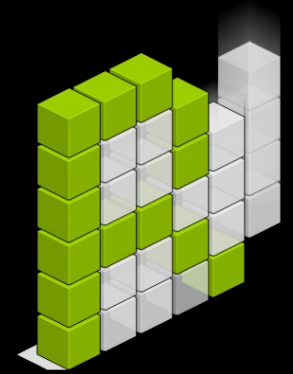
It was not uncommon when using AFNetworking in Gowalla to have calls chained together in success blocks. My advice would be to factor the network requests and serializations as best you can into class methods in your model. Then, for requests that need to make sub-requests, you can call those methods in the success block.

Callback Hell



```
- (void)suspendURLIfDownloading:(NSURL *)url automaticallyResumeLater:(BOOL)autoResume
{
    [_session getTasksWithCompletionHandler:^(NSArray *dataTasks, NSArray *uploadTasks, NSArray *downloadTasks) {
        [downloadTasks enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
            NSURLSessionDownloadTask *task = (NSURLSessionDownloadTask *)obj;
            if ([task.originalRequest.URL.absoluteString isEqualToString:url.absoluteString]) {
                *stop = YES;
                [task cancelByProducingResumeData:^(NSData *resumeData) {
                    [_queue addOperationWithBlock:^(
                        [_db executeUpdate:
                            @"UPDATE FCDownloadQueue SET state = ?, resumeData = ? WHERE url = ?",
                            @( autoResume ? FCDownloadStatePausedAutoResume : FCDownloadStatePausedDoNotAutoResume ),
                            resumeData ? resumeData : [NSNull null],
                            url.absoluteString
                        ];
                    [self.delegate downloadQueue:self didPauseDownloadingURL:url userInfo:[self userInfoForURL:url]];
                    [self ensureEnoughDownloadsAreRunning];
                }];
            }
        }];
    }];
}
```

[illegible]



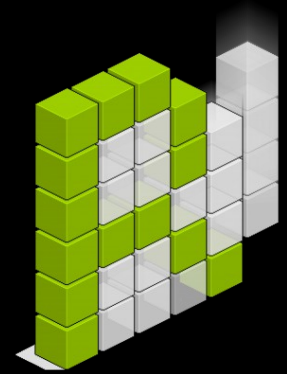
Working with independent data sets in parallel and then combining them into a final result is non-trivial in Cocoa, and often involves a lot of synchronization

```
__block NSArray *databaseObjects, fileContents;
NSOperationQueue *backgroundQueue = [[NSOperationQueue alloc] init];
NSBlockOperation *databaseOperation = [NSBlockOperation blockOperationWithBlock:^(
    databaseObjects = [databaseClient fetchObjectsMatchingPredicate:predicate];
)];
```

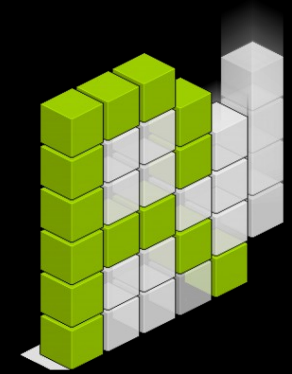
```
NSBlockOperation *filesOperation = [NSBlockOperation blockOperationWithBlock:^(
    NSMutableArray *filesInProgress = [NSMutableArray array];
    for (NSString *path in files) {
        [filesInProgress addObject:[NSData dataWithContentsOfFile:path]];
    }

    fileContents = [filesInProgress copy];
)];
```

```
NSBlockOperation *finishOperation = [NSBlockOperation blockOperationWithBlock:^(
    [self finishProcessingDatabaseObjects:databaseObjects fileContents:fileContents];
    NSLog(@"Done processing");
)];
[finishOperation addDependency:databaseOperation];
[finishOperation addDependency:filesOperation];
[backgroundQueue addOperation:databaseOperation];
[backgroundQueue addOperation:filesOperation];
[backgroundQueue addOperation:finishOperation];
```



Rather than using mutable variables which are replaced and modified in-place, RAC provides signals (represented by `RACSignal`) that capture present and future values.



Self-loading properties

When Design Patterns are NOT enough



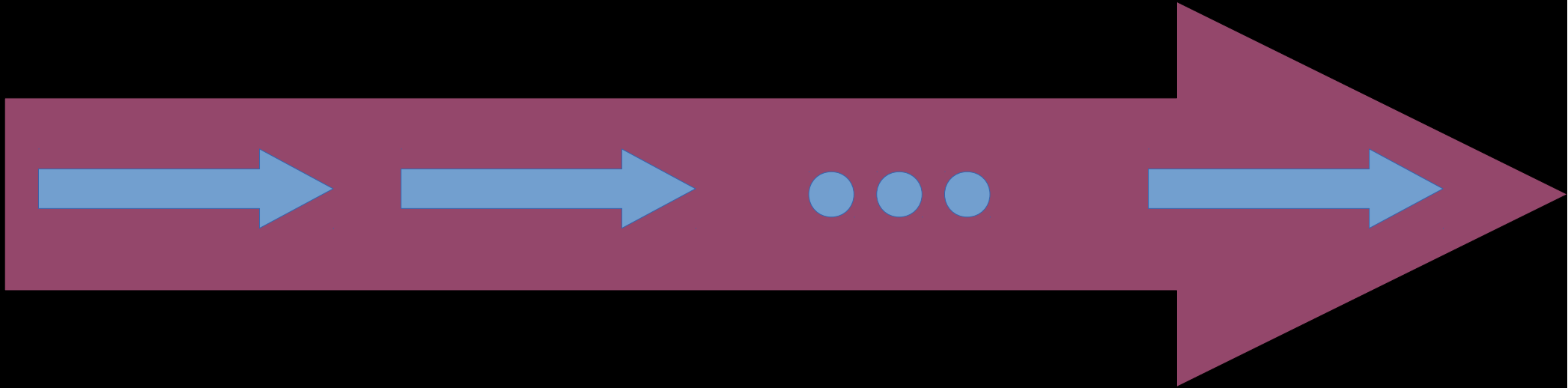
Meet the Async Operation

```
CancelBlock (^AsyncOperation)  
( ProgressCallback,  
  CancelCallback,  
  CompletionCallback ) {...};
```


Completion Callback

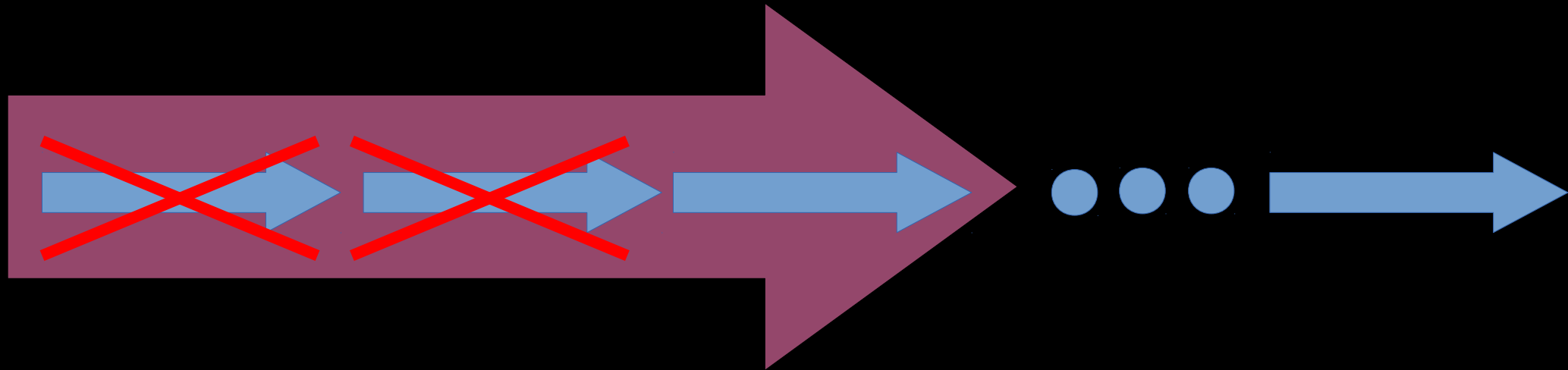
```
void (^CompletionCallback)(  
    id operationResult,  
    NSError* operationError)
```

Sequence



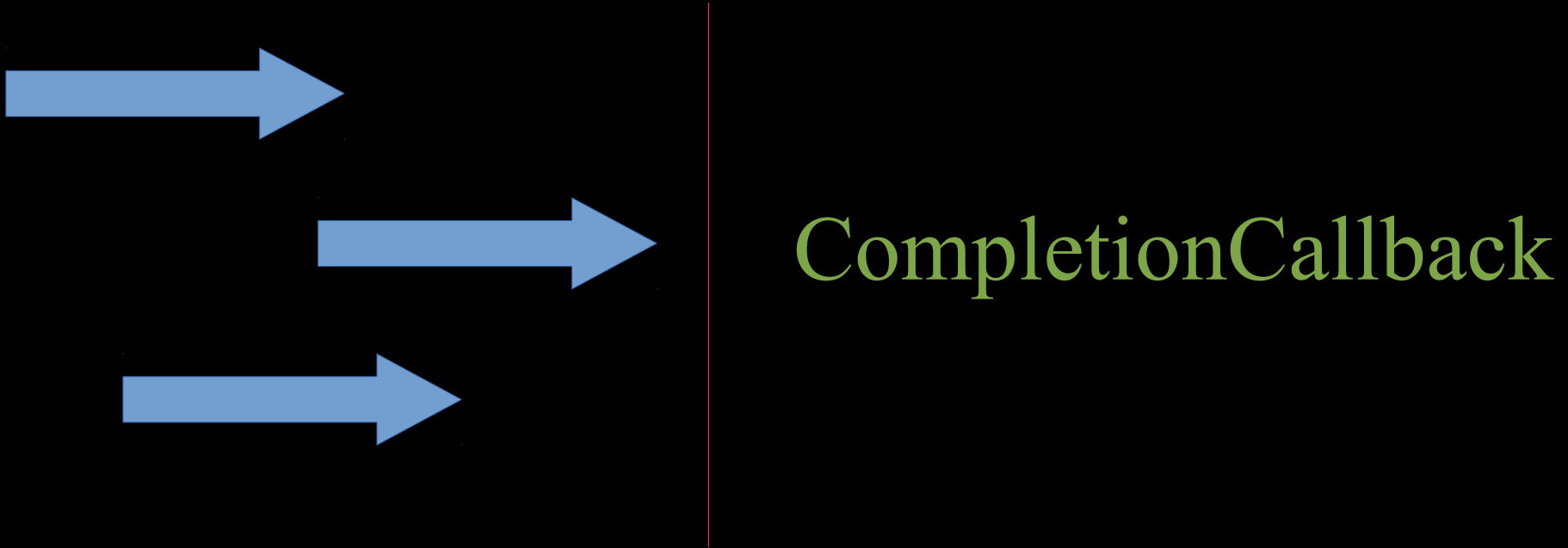
```
JFFAsyncOperation sequence =  
  sequenceOfAsyncOperations(  
    @ [ op1, op2, opN ] );
```

Sequence of Attempts



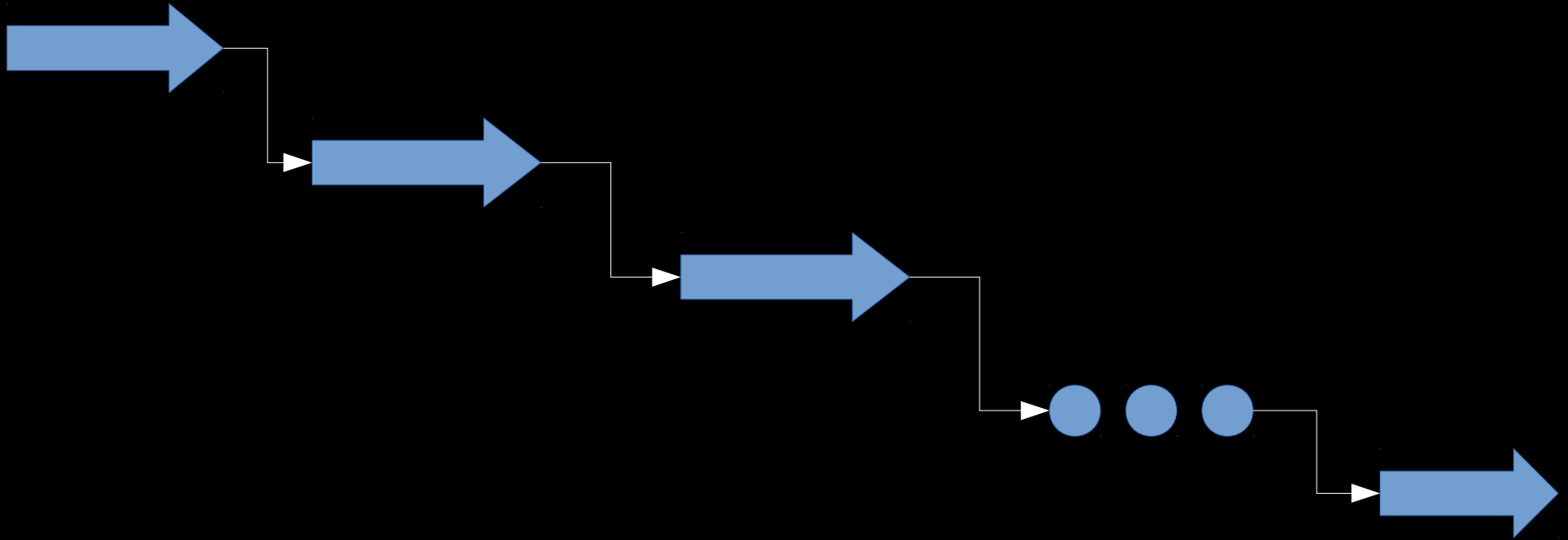
```
JFFAsyncOperation sequence =  
    trySequenceOfAsyncOperations(  
        @[ op1, op2, opN ] );
```

Parallel execution

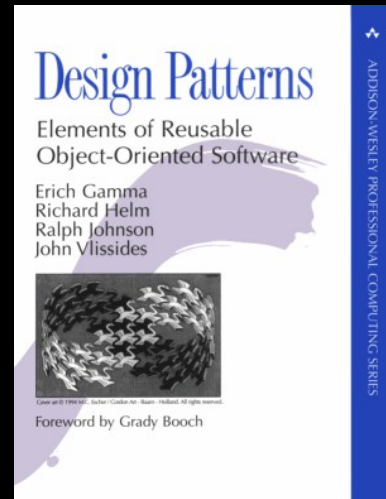


```
JFFAsyncOperation sequence =  
  groupOfAsyncOperations(  
    @[ op1, op2, opN ] );
```

Waterfall



```
JFFAsyncOperation sequence =  
  bindSequenceOfAsyncOperations(  
    op1, @[ b1, b2, bN ] );
```



Function = Command + Composite

Any Complexity Tasks

```
loadFromNet = sequenceOfAsyncOperations(  
@[ login, loadFromNet ] );
```

```
loadRawData = trySequenceOfAsyncOperations(  
@[ loadFromFSCache, loadFromNet ] );
```

```
JFFAsyncOperation getData =  
bindSequenceOfAsyncOperations(  
loadRawData, @[ parseBinder, filterBinder ] );
```


Demo

<http://bit.ly/18HkcMa>

iAsync

bit.ly/1bLdN59

Does anyone use it?

Wishdates



Sitecore



<http://bit.ly/1aEqAku>

Pitfalls

Large codebase

No PodSpec

No Documentation

No Community

No Versioning policy



DEVELOPERS
DEVELOPERS
DEVELOPERS
DEVELOPERS

iAsync Author

Vladimir Gorbenko



Gorbenko.Vova@gmail.com



Vova.Gorbenko.mac



github.com/volodg



Oleksandr Dodatko



dodikk88.tutor@gmail.com



alexander.dodatko.work



[@dodikk88](https://twitter.com/dodikk88)



github.com/dodikk

