What's New in Core Data

The new stuff you need to know

Session 225
Melissa Turner
Sr. Software Engineer

© 2014 Apple Inc. All rights reserved. Redistribution or public display not permitted without written permission from Apple.

Roadmap

Batch updates

Asynchronous fetching

Incremental stores

Concurrency changes

iCloud update

Swift

Batch Updates

Batch Updates

What, why

Make changes directly in the database

- Attributes

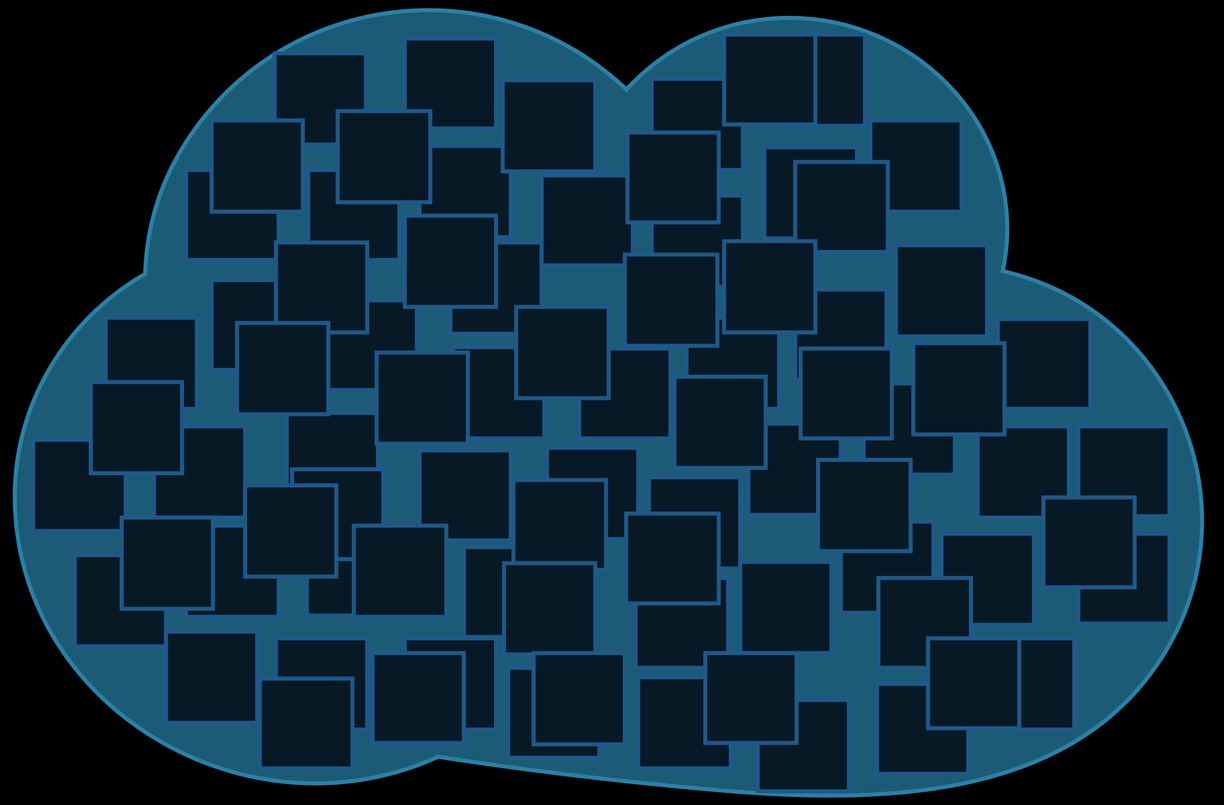
Performance optimization

The Old Way

Version one

The Old Way

Version one



The Old Way

Version one

The Problem

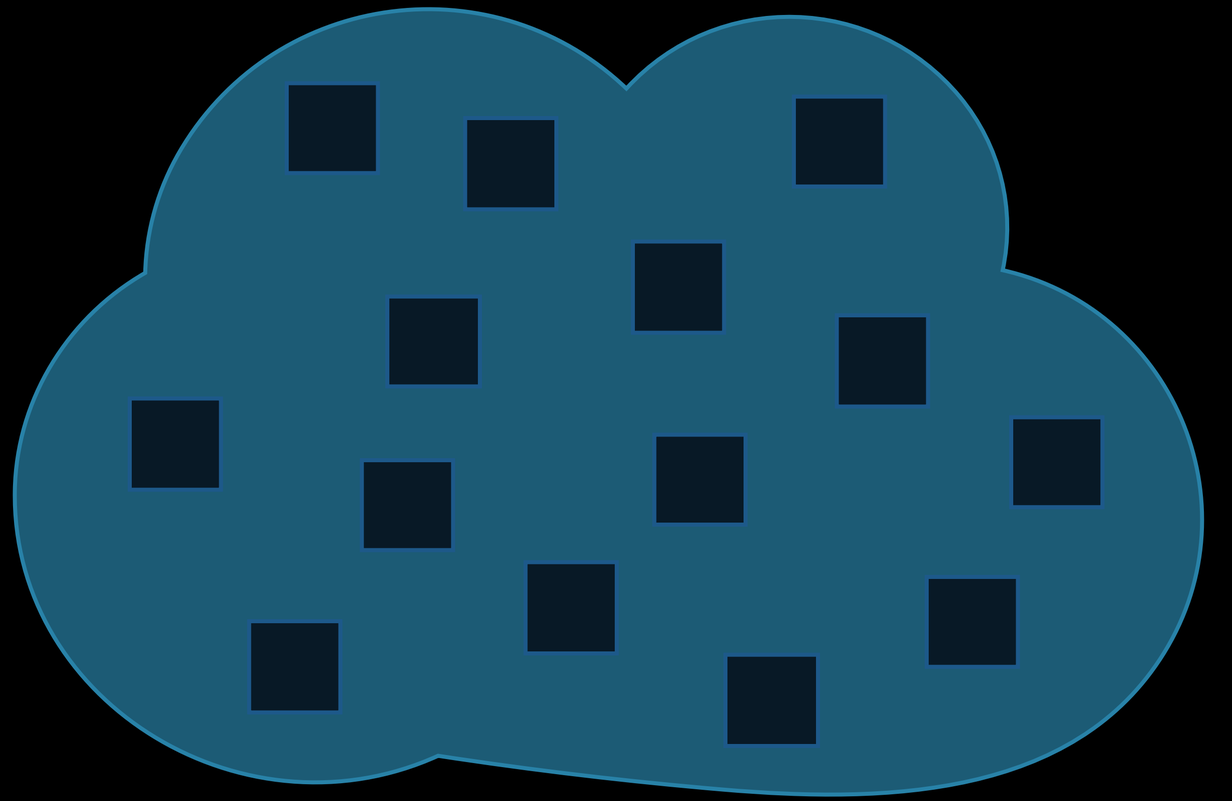
Version one

The Old Way

Version two

The Old Way

Version two



The Old Way

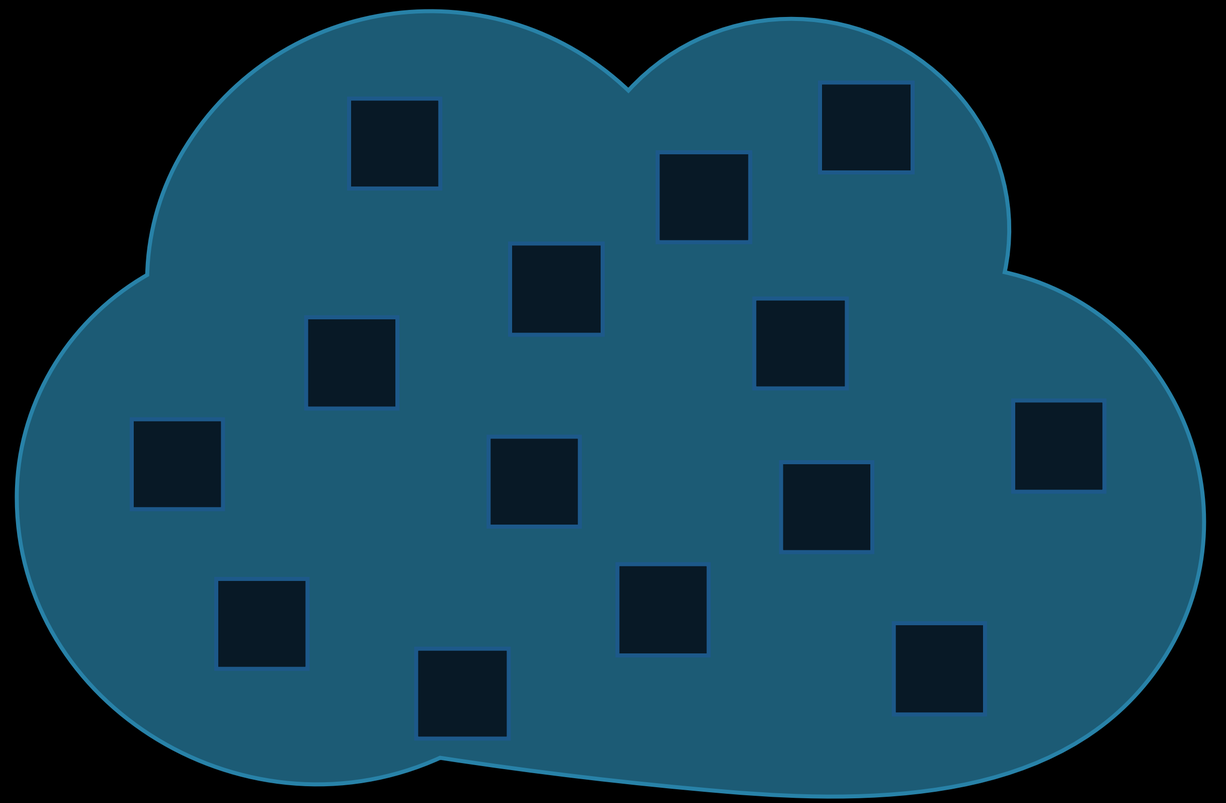
Version two

The Old Way

Version two

The Old Way

Version two



The Old Way

Version two

The Problem

Version two

The Problem

Version two

Batch Updates

Where?

-[NSManagedObjectContext executeRequest: error:]

- Takes NSPersistentStoreRequest
- Returns NSPersistentStoreResult

NSBatchUpdateRequest

NSBatchUpdateResult

NSBatchUpdateRequest

How?

Entity

Store(s)

Predicate

Properties to update

- Property as key
- NSExpression describing the desired update as value

NSBatchUpdateResult

What?

Success/failure

Count of rows changed

Object IDs of rows changed

Batch Updates

Updating your database in one easy invocation

Changes are not reflected in the context

Validation rules are not run

Does update optimistic locking version in database

- Create merge conflicts on yourself

Demo

Batch updates—Faster by design

Batch Updates

Updating your database in one easy invocation

Changes are not reflected in the context

Validation rules are not run

Does update optimistic locking version in database

- Create merge conflicts on yourself

Batch Updates

Updating your database in one easy invocation

Changes are not reflected in the context

Validation rules are not run

Does update optimistic locking version in database

- Create merge conflicts on yourself

Asynchronous Fetching

Asychronous Fetching

What, why, where, how?

Way to execute a fetch without blocking

Cancellable

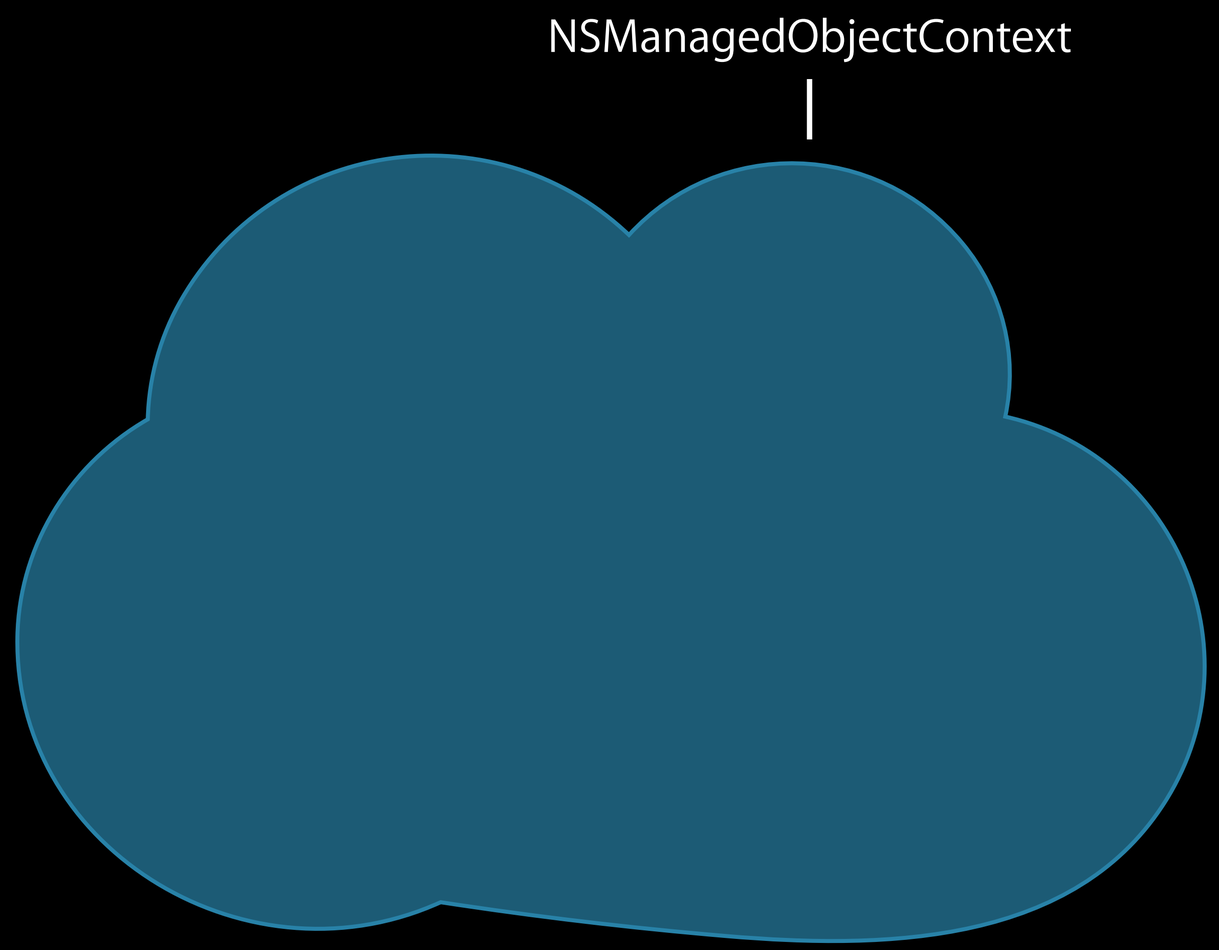
Progress reporting

Synchronous Fetching

You know this

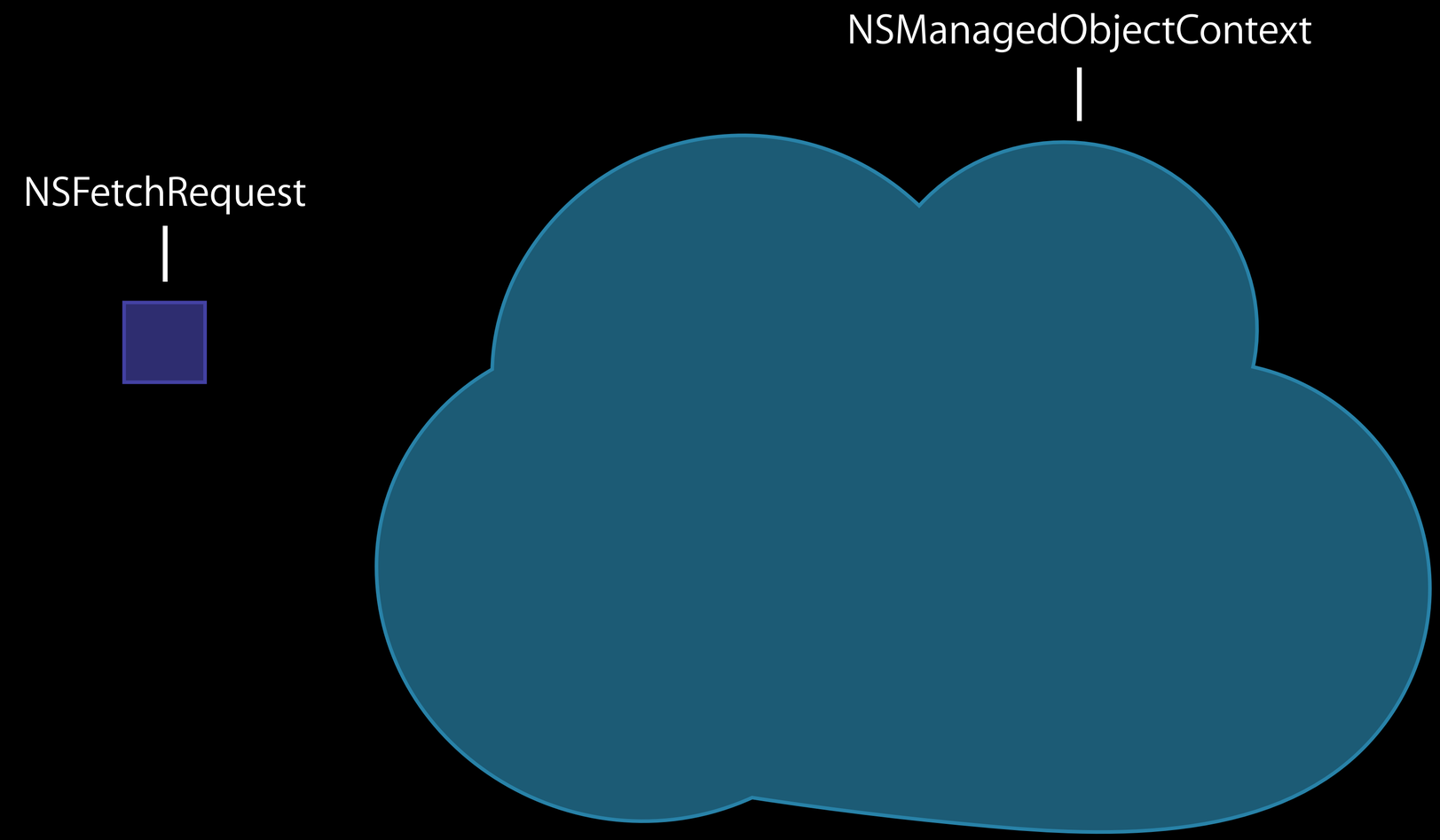
Synchronous Fetching

You know this



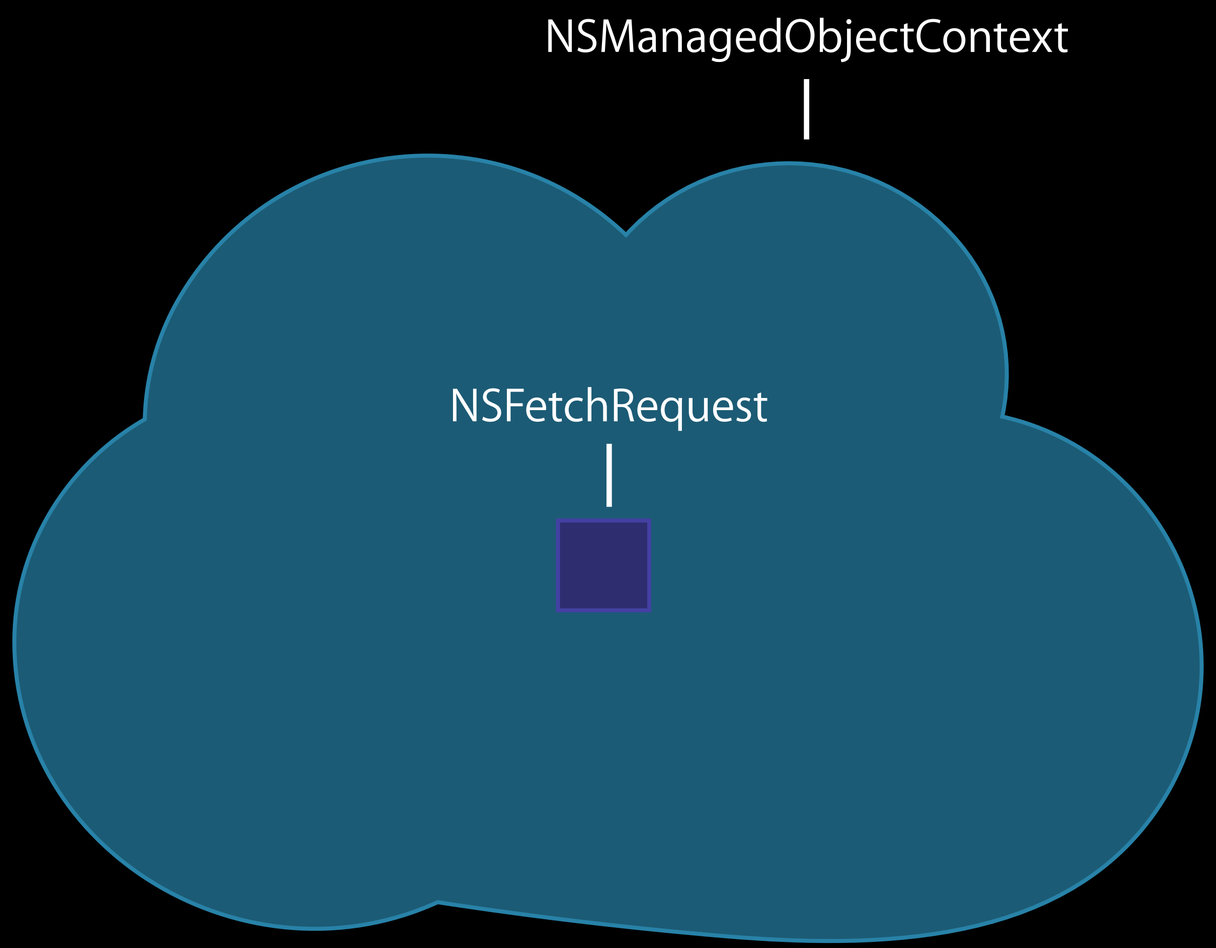
Synchronous Fetching

You know this



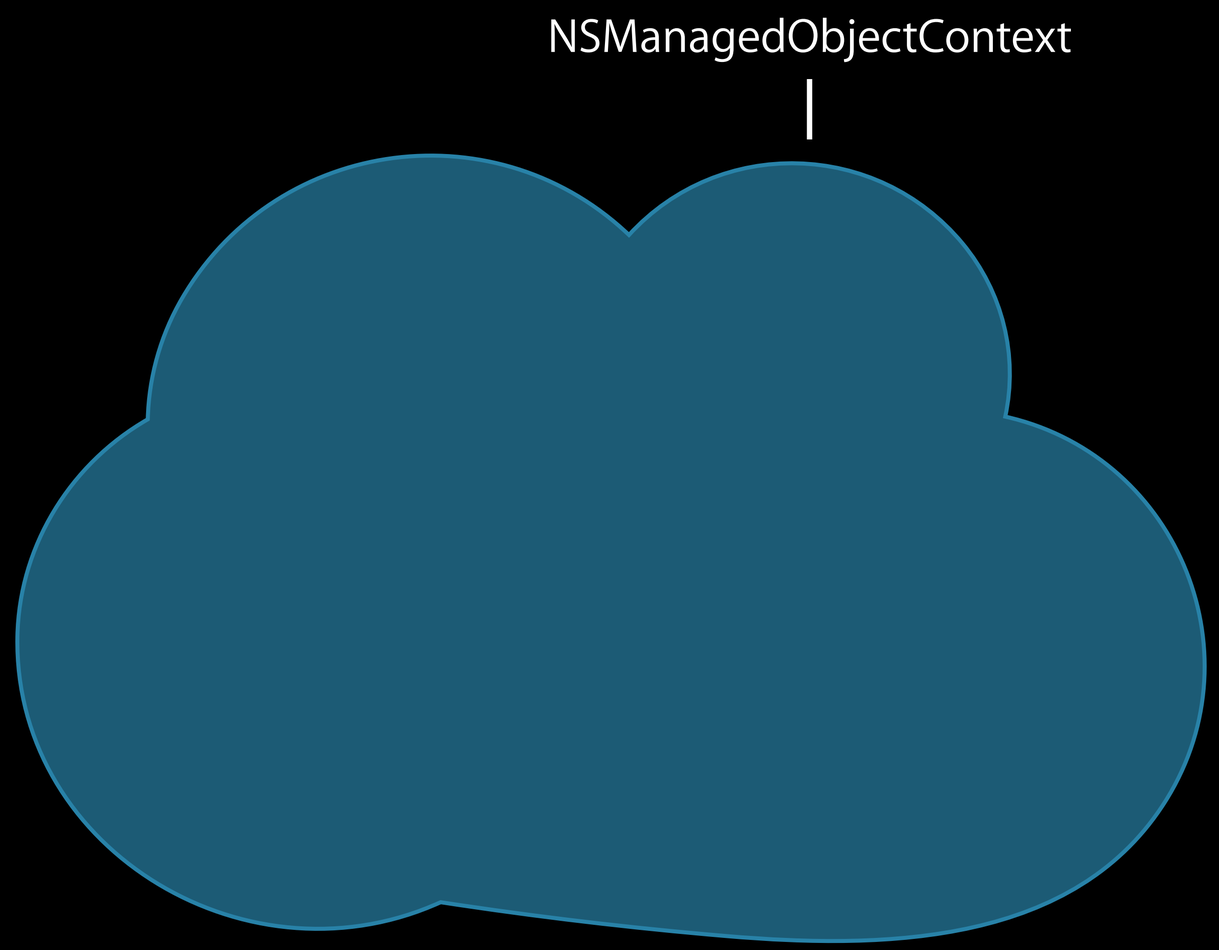
Synchronous Fetching

You know this



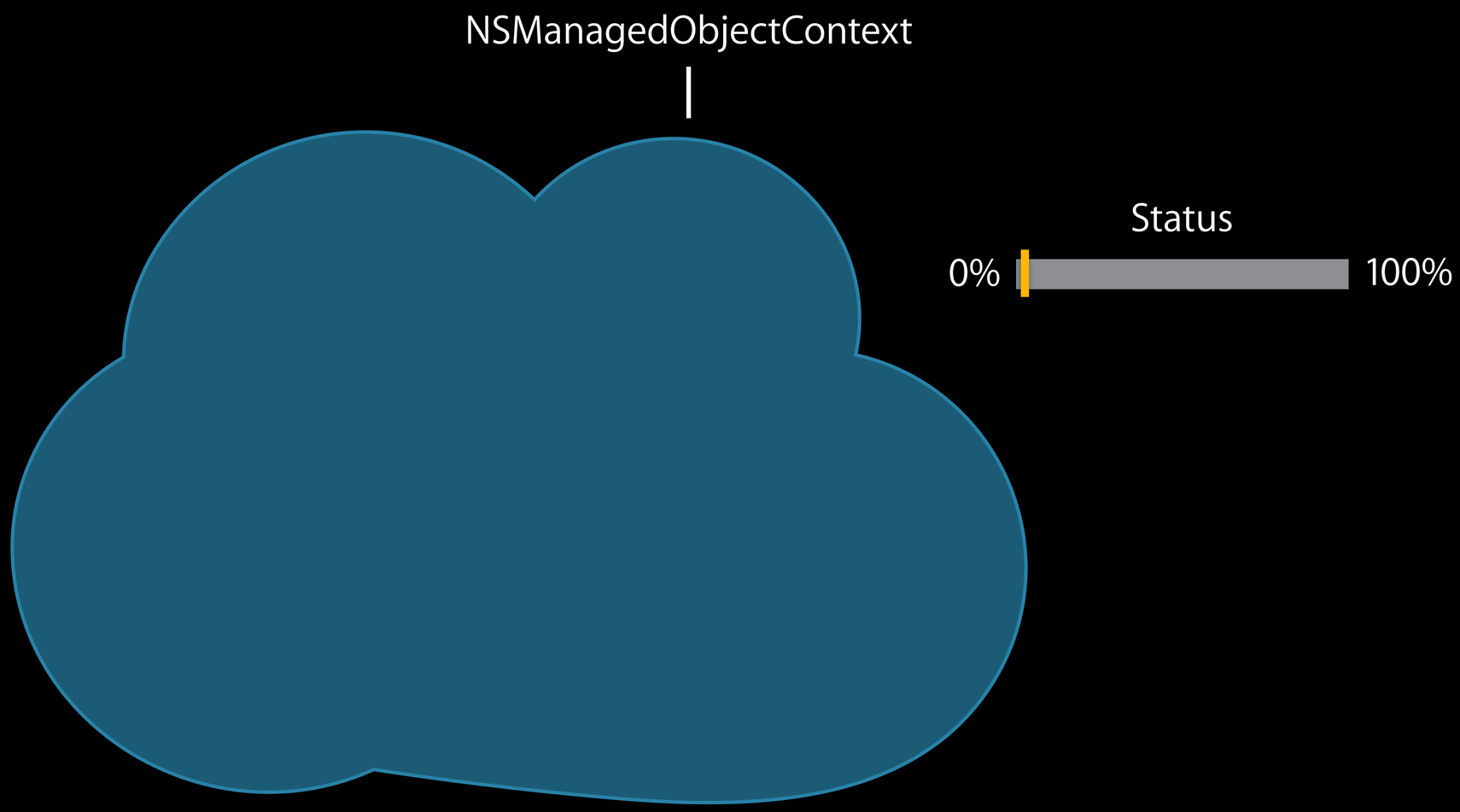
Synchronous Fetching

You know this



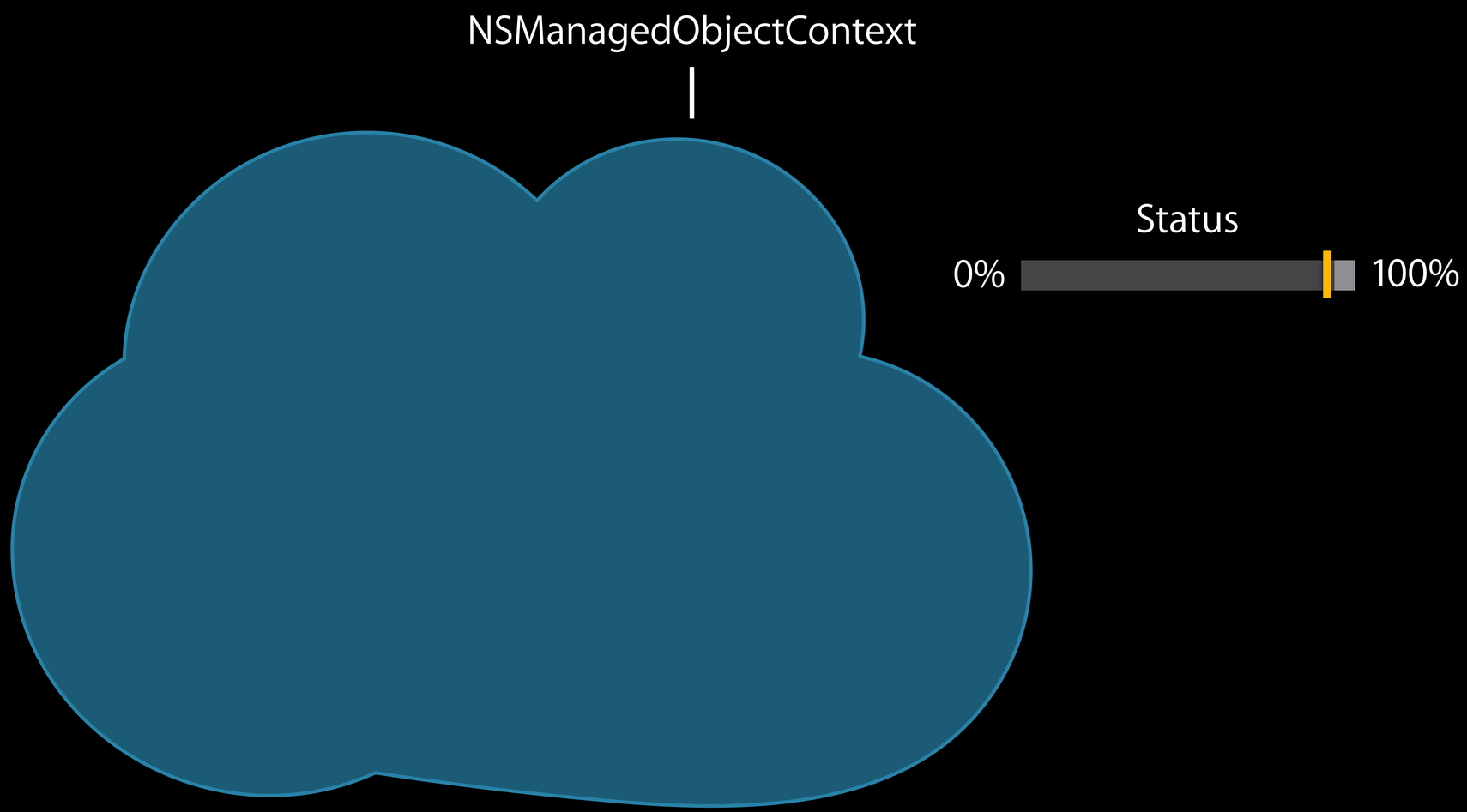
Synchronous Fetching

You know this



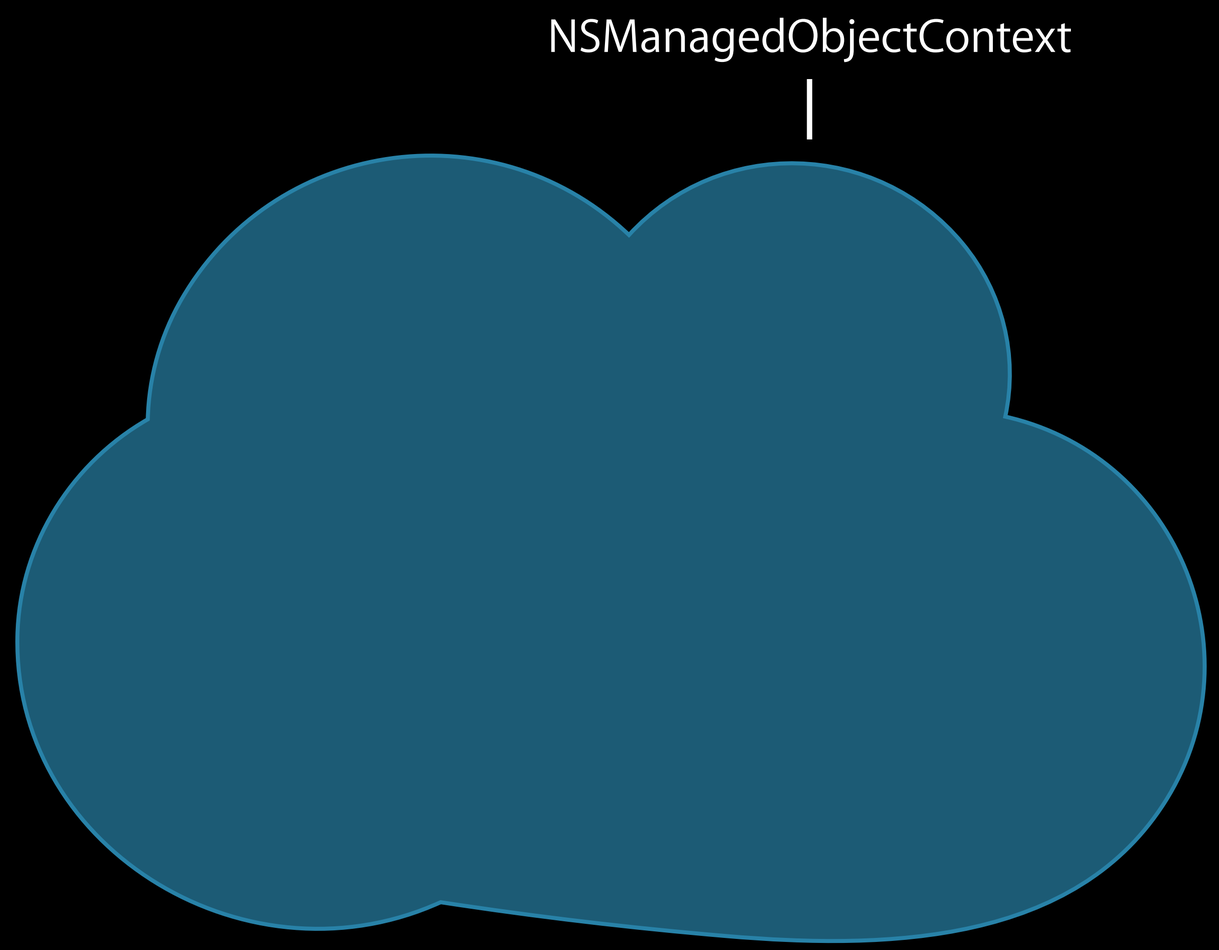
Synchronous Fetching

You know this



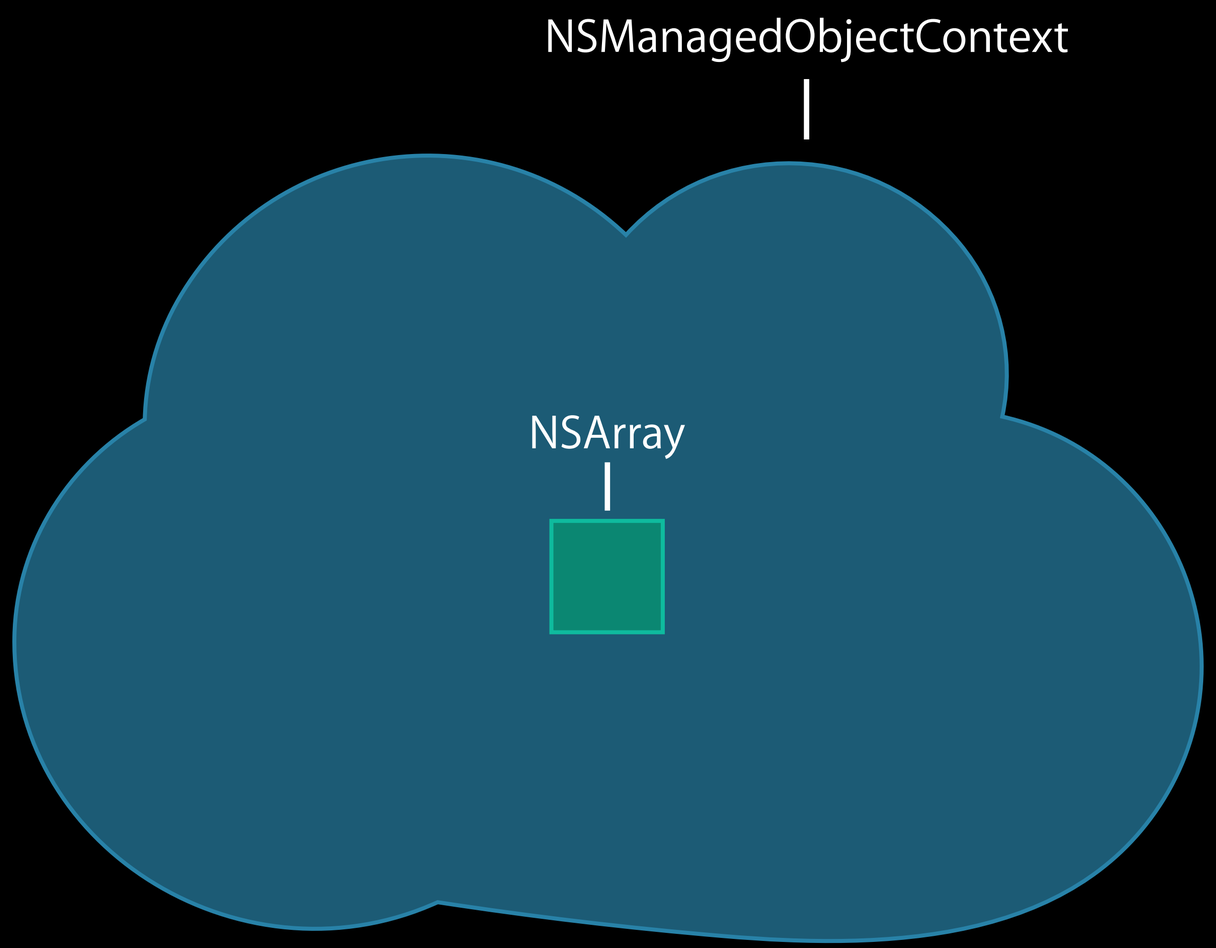
Synchronous Fetching

You know this



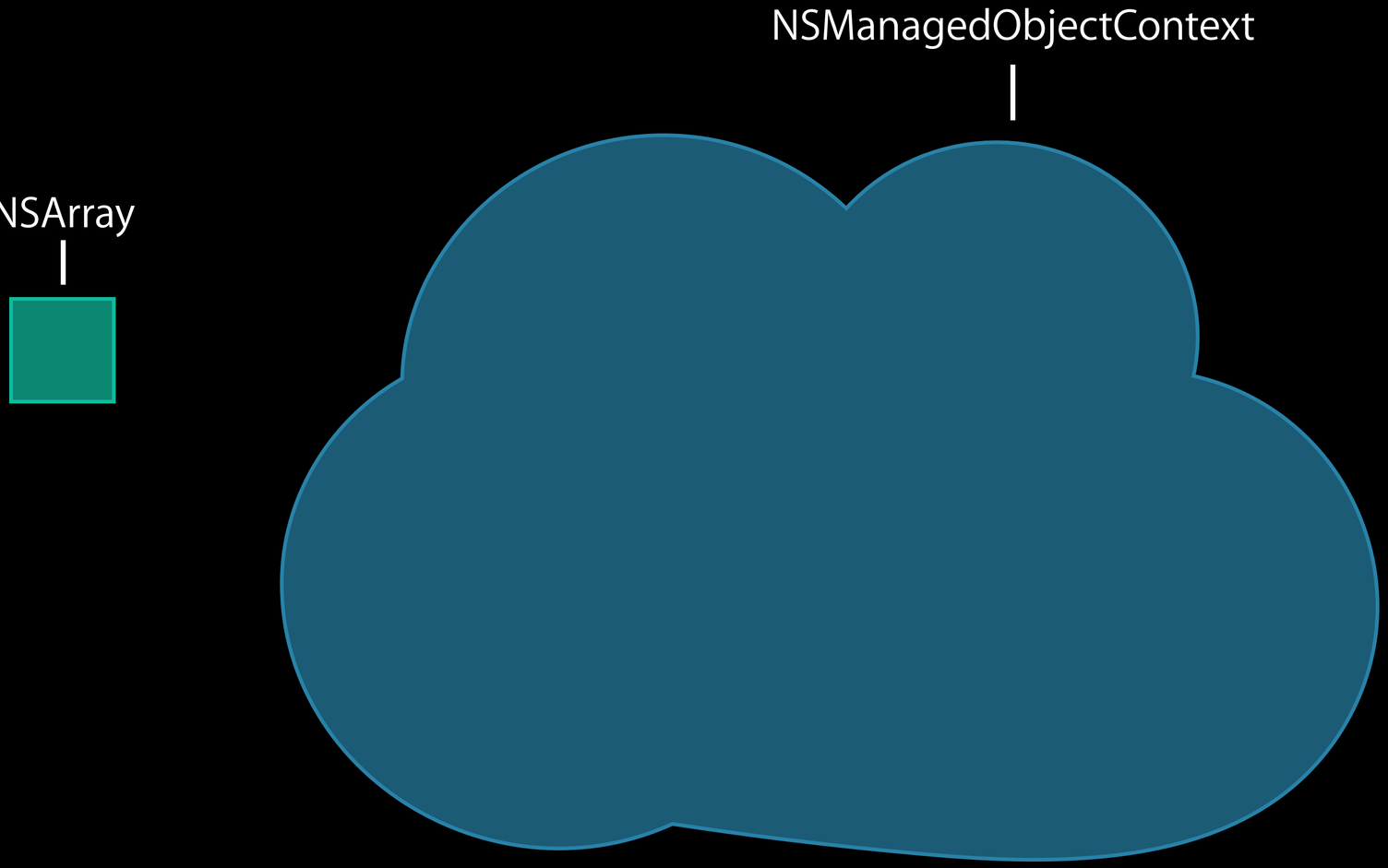
Synchronous Fetching

You know this



Synchronous Fetching

You know this



Asychronous Fetching

Returns a future

- NSAsynchronousFetchResult

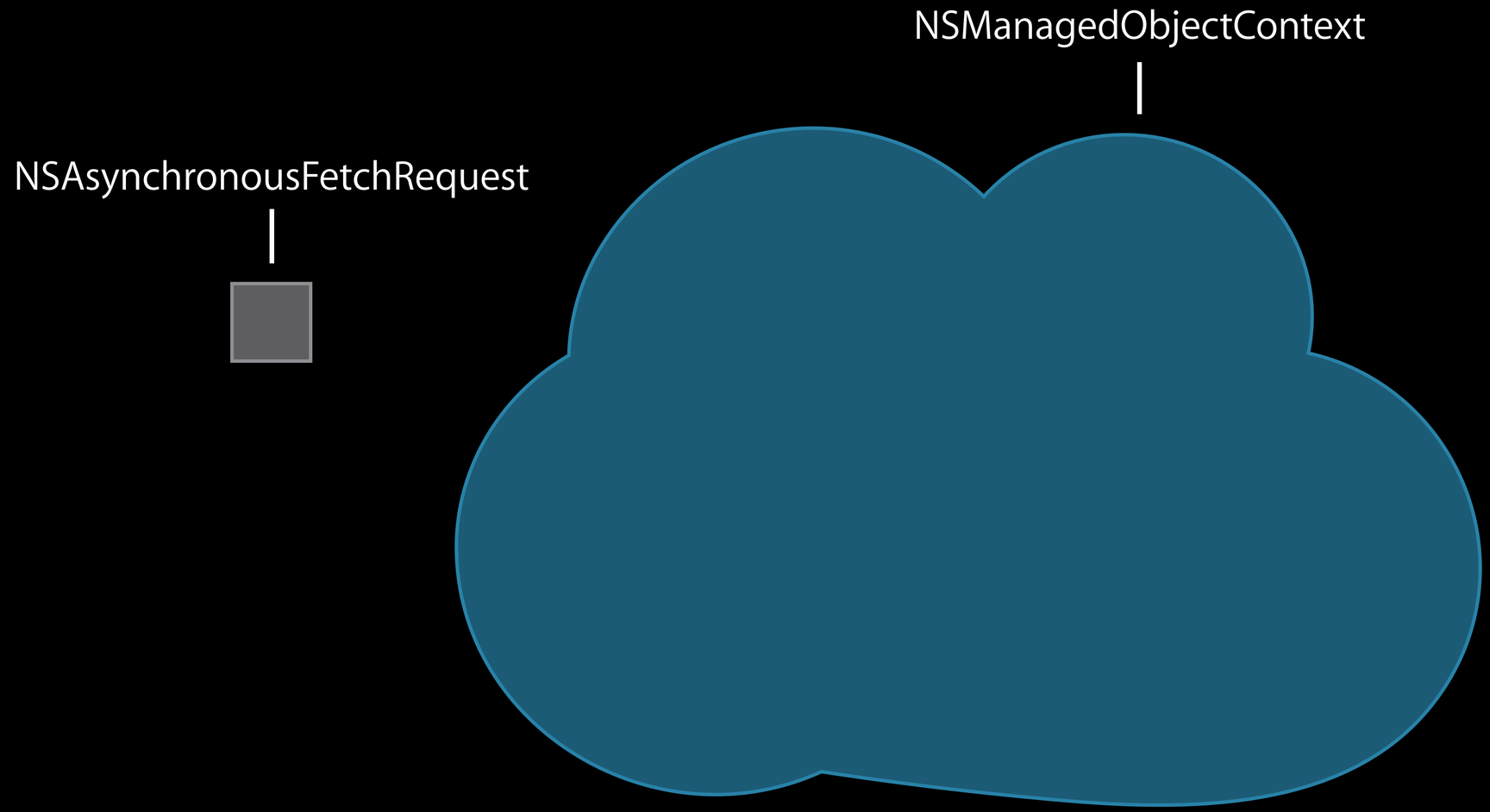
Callback block executed when fetch is complete

Will update context if specified

NSPrivateQueueConcurrencyType and NSMainQueueConcurrencyType only

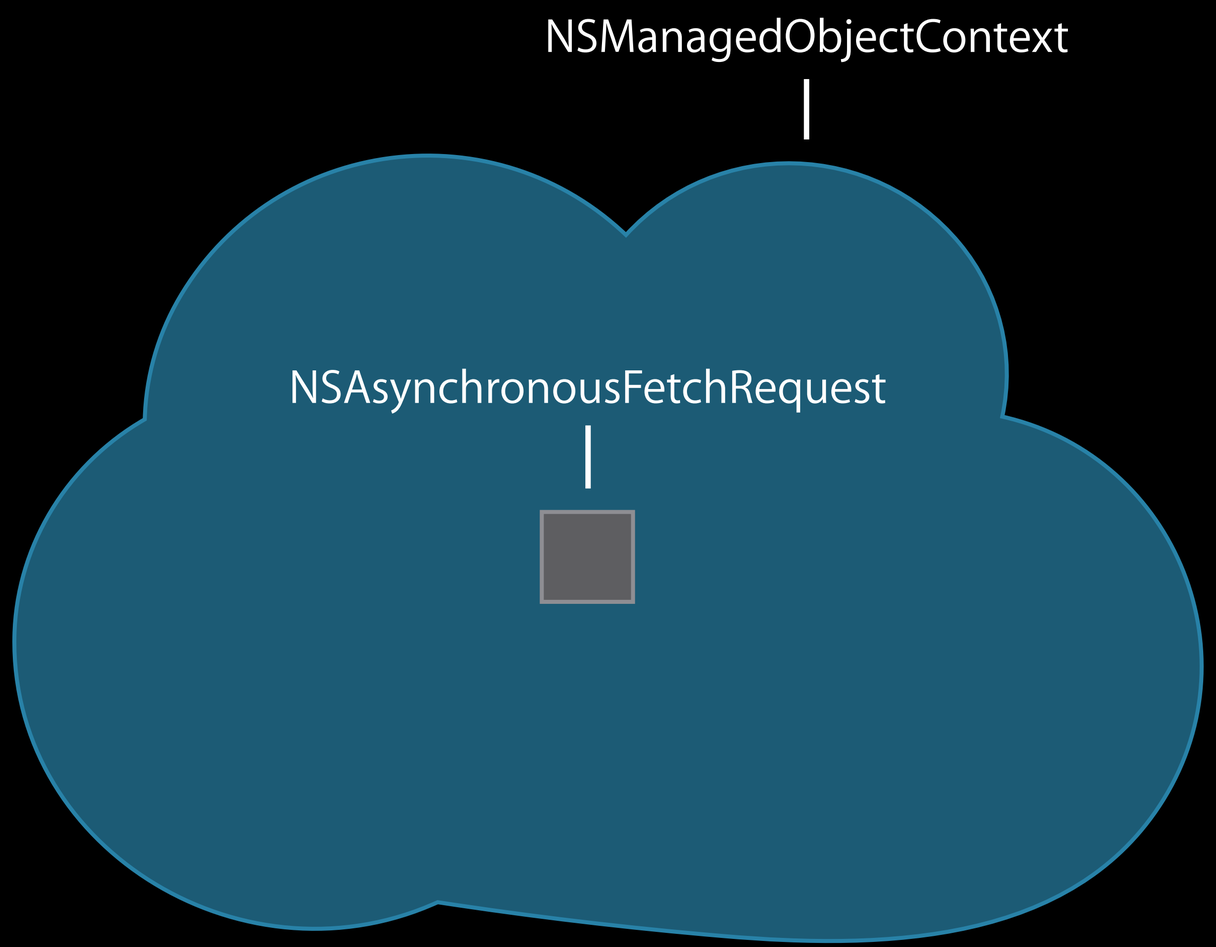
Asynchronous Fetching

Fire and forget



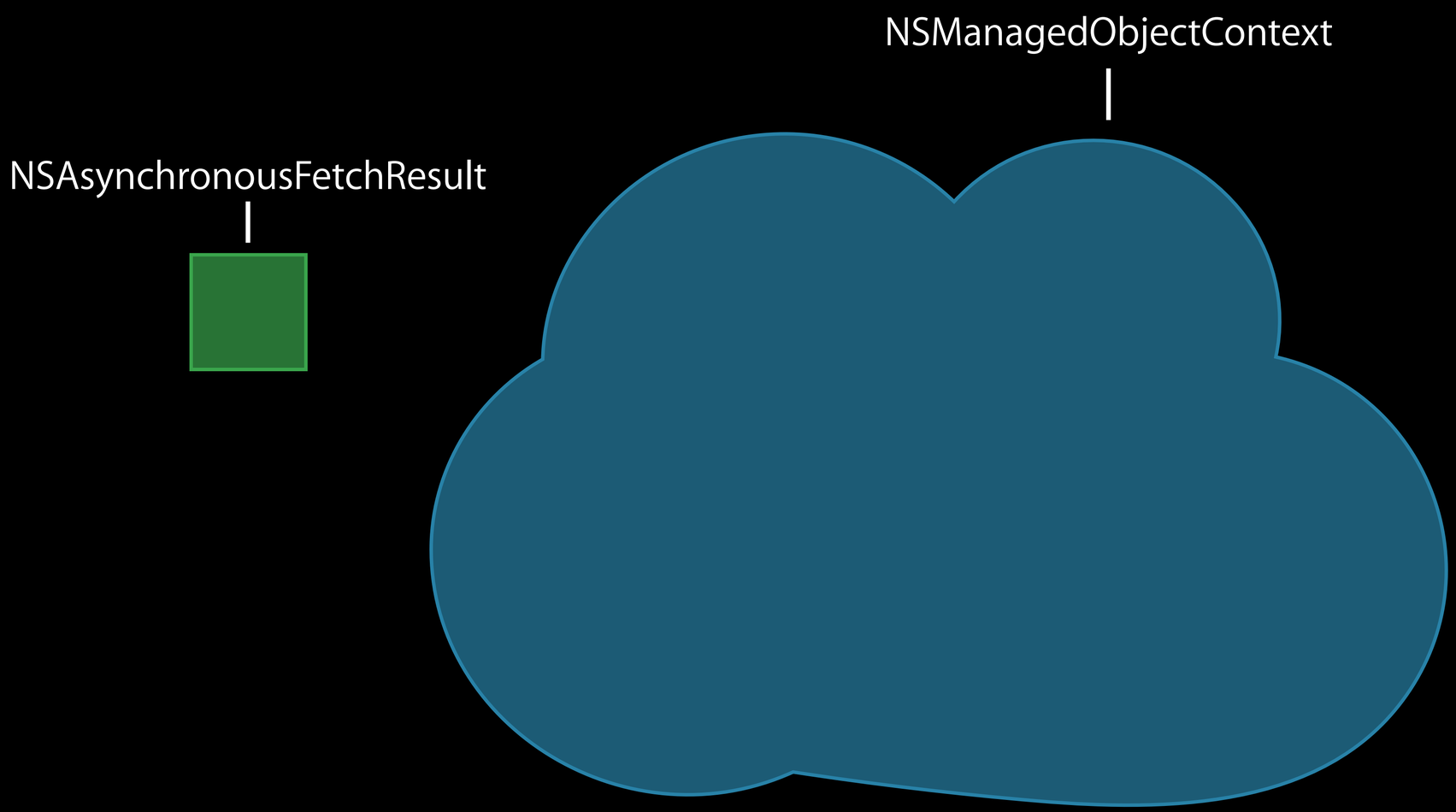
Asynchronous Fetching

Fire and forget



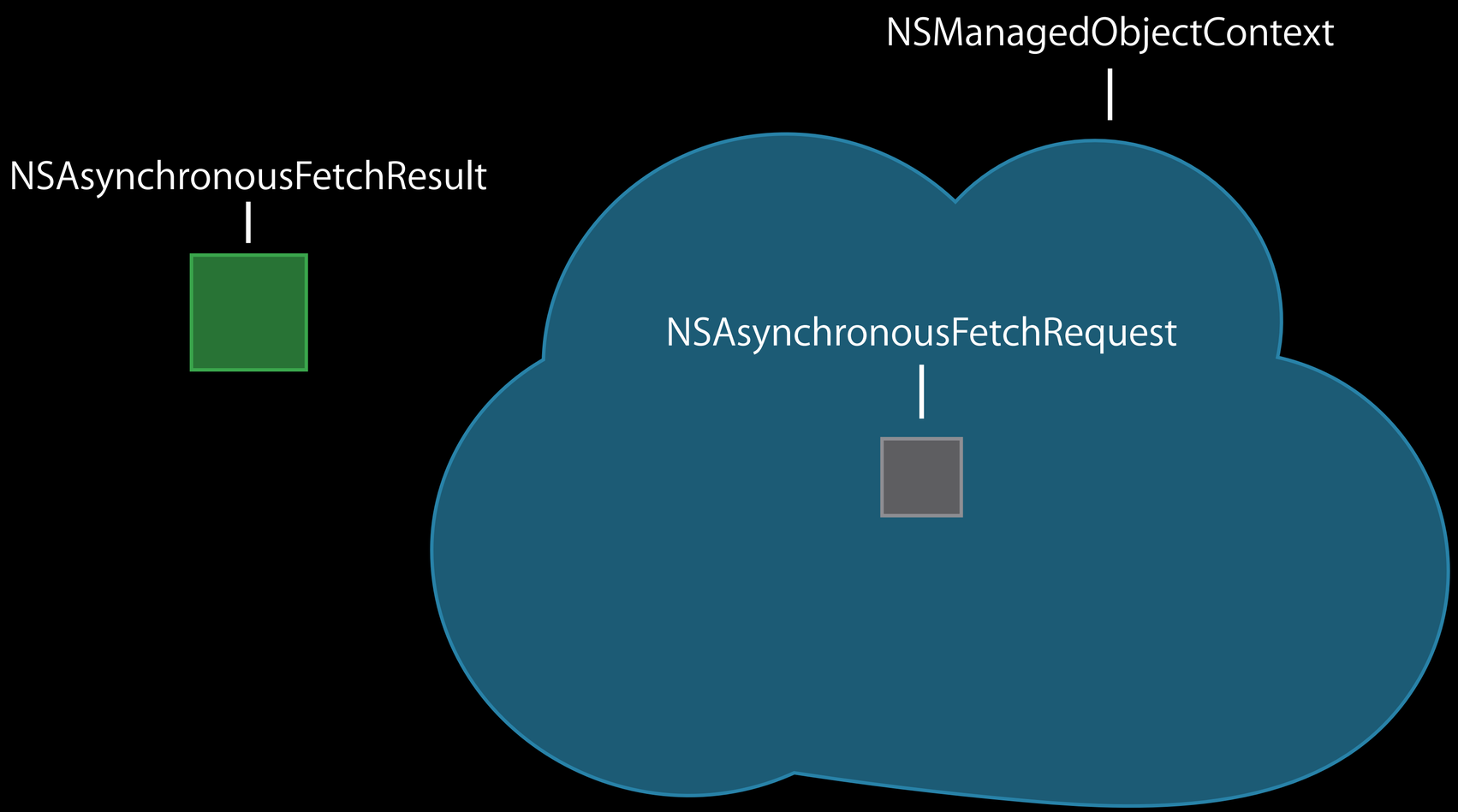
Asynchronous Fetching

Fire and forget



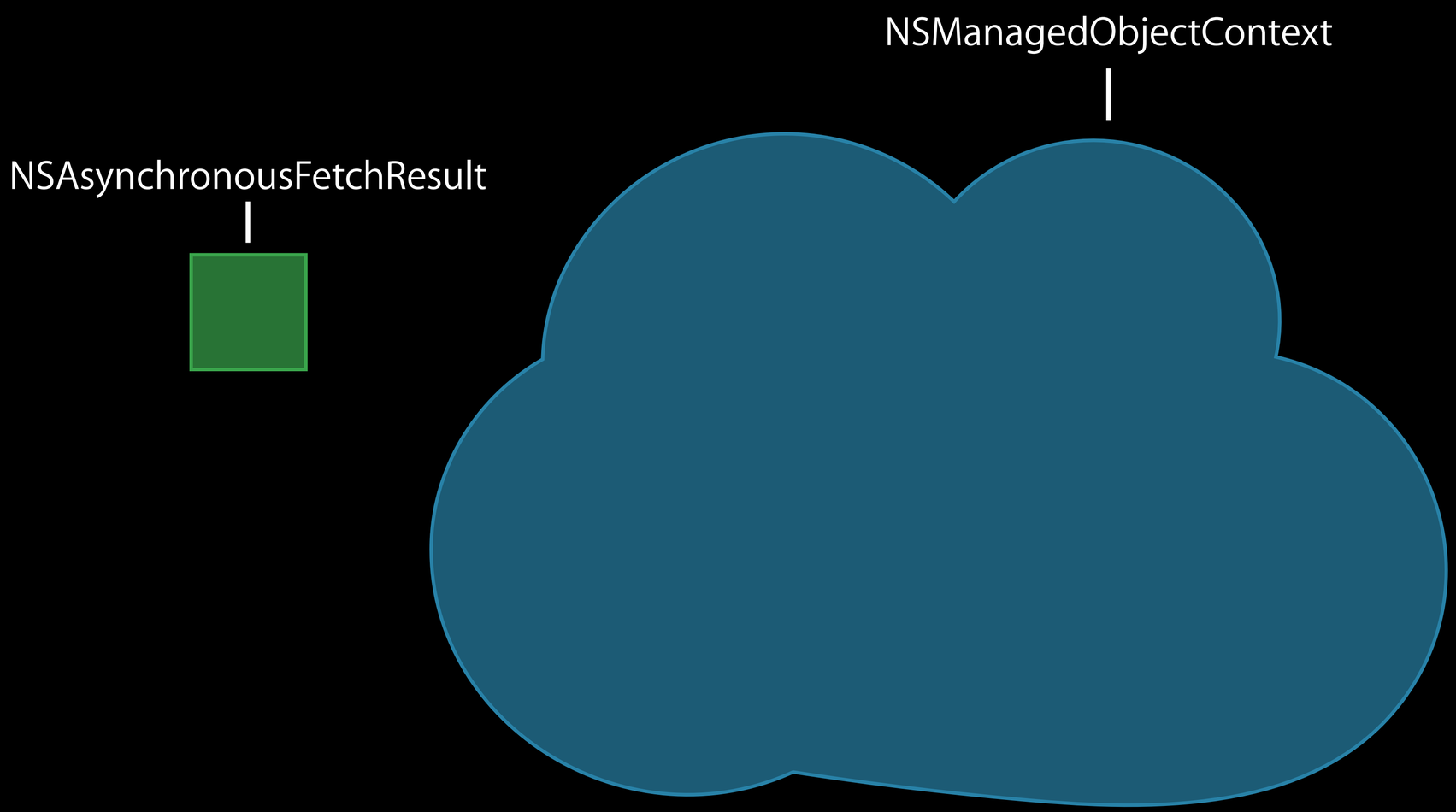
Asynchronous Fetching

Fire and forget



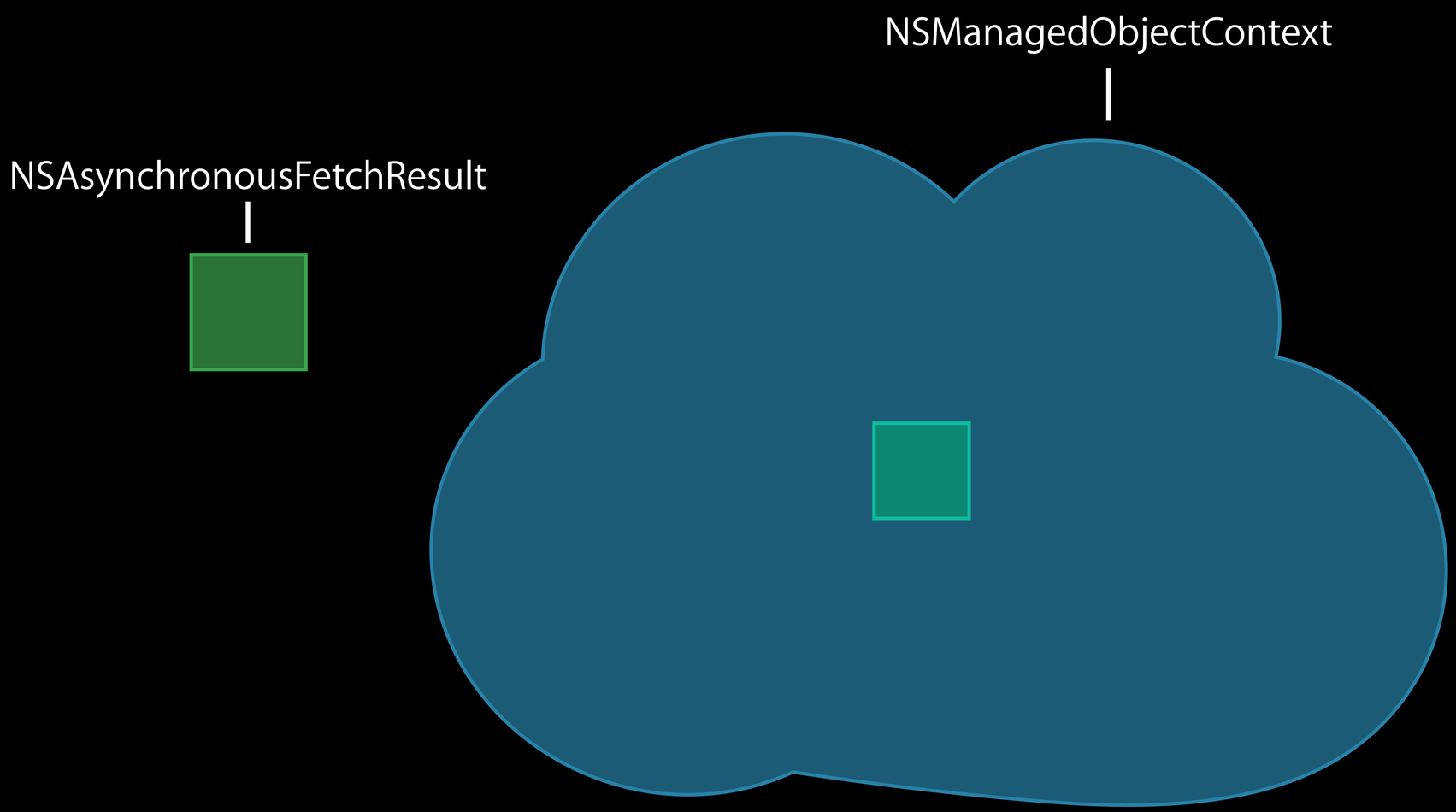
Asynchronous Fetching

Fire and forget



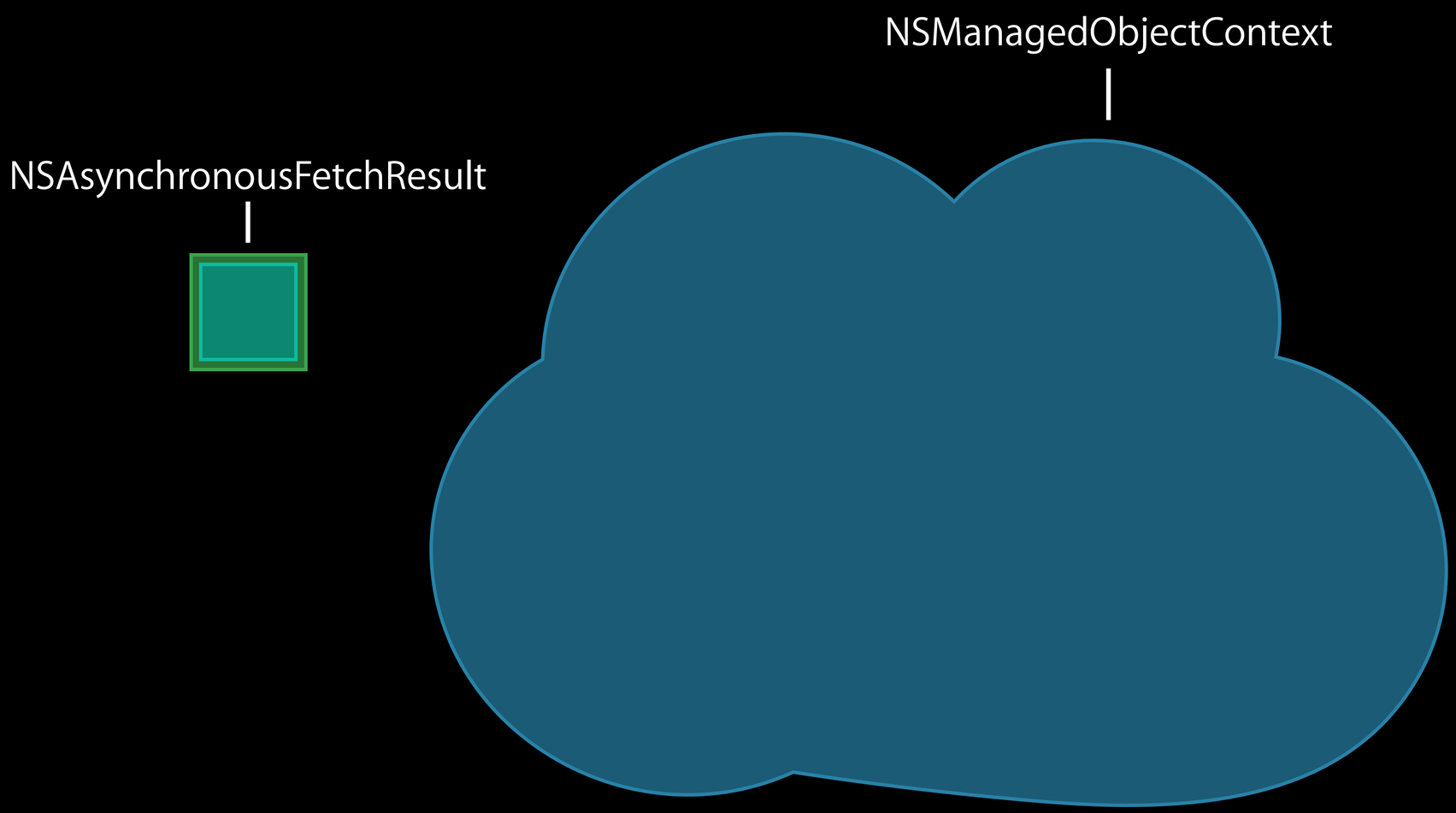
Asynchronous Fetching

Fire and forget



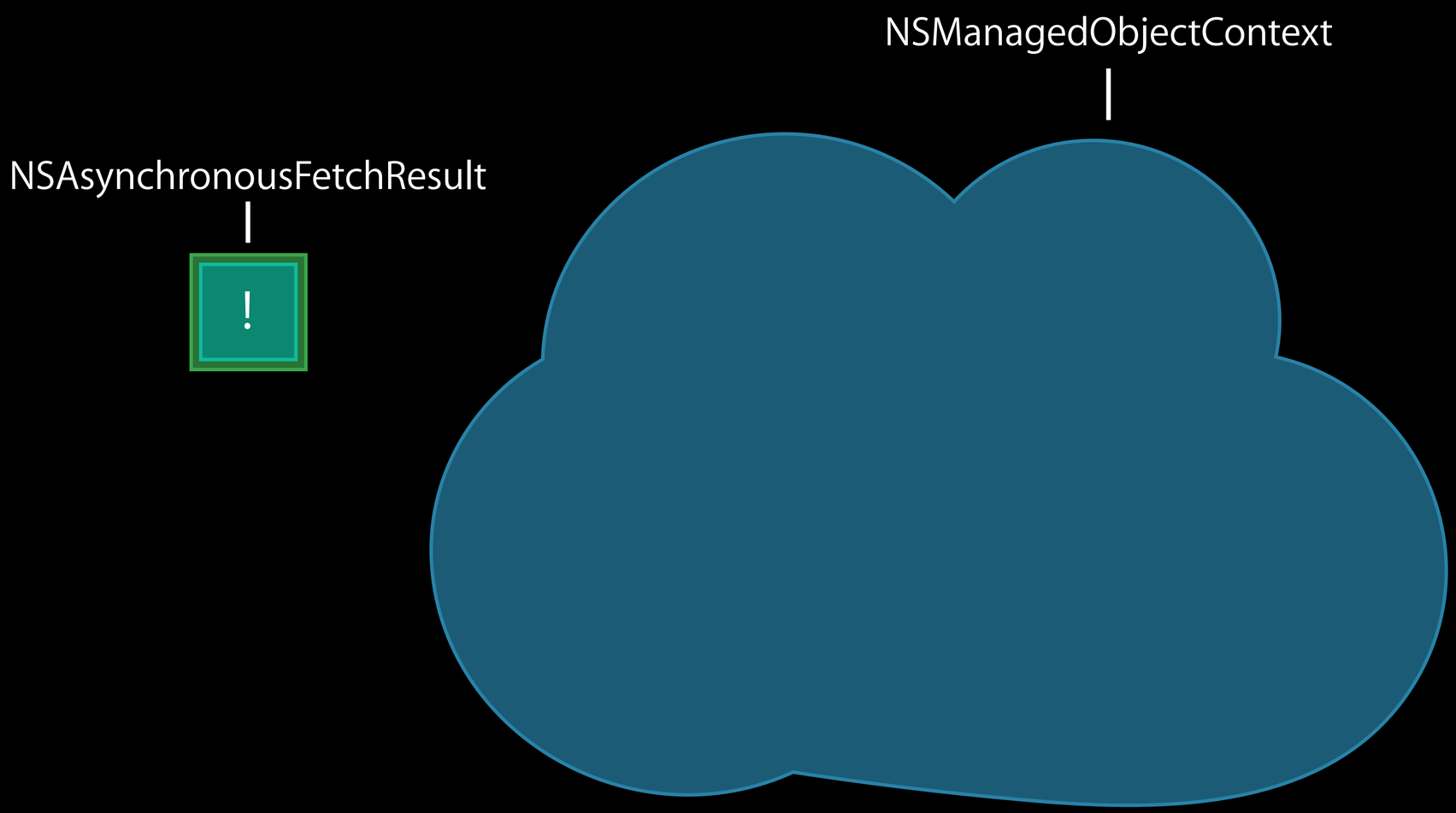
Asynchronous Fetching

Fire and forget



Asynchronous Fetching

Fire and forget



NSAsynchronousFetchRequest

New NSPersistentStoreRequest subclass

Initialized with fetch request and completion block

Passed to -executeRequest:error:

NSAsynchronousFetchResult

New NSPersistentStoreResult subclass

Provides results or error after completion

Returned immediately from -executeRequest:error:

Asynchronous Fetching Progress

```
NSFetchRequest *request = [NSFetchRequest
fetchRequestWithEntityName:”MyEntity”];
NSAsynchronousFetchRequest *async = [[NSAsynchronousFetchRequest alloc]
    initWithFetchRequest: request completionBlock^(id result) {
        if (result.finalResult) {
        … }
}];
[context performBlock: ^() {
    NSError *error = nil;
    asyncResult = [moc executeRequest: asyncRequest error:&blockError];
}];
```

Asynchronous Fetching Progress

Uses NSProgress

Create your own NSProgress before -executeRequest:error

NSManagedObjectContext will create nested NSProgress

Allows cancellation from NSProgress

Asynchronous Fetching Progress

```
NSProgress *progress = [NSProgress progressWithTotalUnitCount: 1];
[progress becomeCurrentWithPendingUnitCount: 1];
[context performBlock: ^() {
    [context executeRequest: asyncRequest error:&error]
}];
[progress resignCurrent];
```

Demo

Asynchronous fetching—Faster by design

Ben Trumbull
Core Data Engineer Manager

Incremental Stores

Incremental Implications

Adjusting your store's expectations

-executeRequest:withContext:

- Core Data request/response types
- Can add your own

Please fail gracefully

Incremental Implications

New request types

Use – `[NSManagedObjectContext executeRequest:error:]`

- Does serialization for you

Create a request/response pair

Context will return aggregated result

Default store types will not recognize custom request types

Incremental Implications

Why implement your own?

Minimizing trips to the store

- Fetching disjoint entities

Object refresh

Status checks

Incremental Implications

Asynchronicity

Return future immediately

Message future when request completes

Use performBlock: to update context

Concurrency

Sub Roadmap

Retrospective

- NSLocking
- Thread confinement
- Concurrency types

New advice

Retrospective

Making sense of Stack Overflow

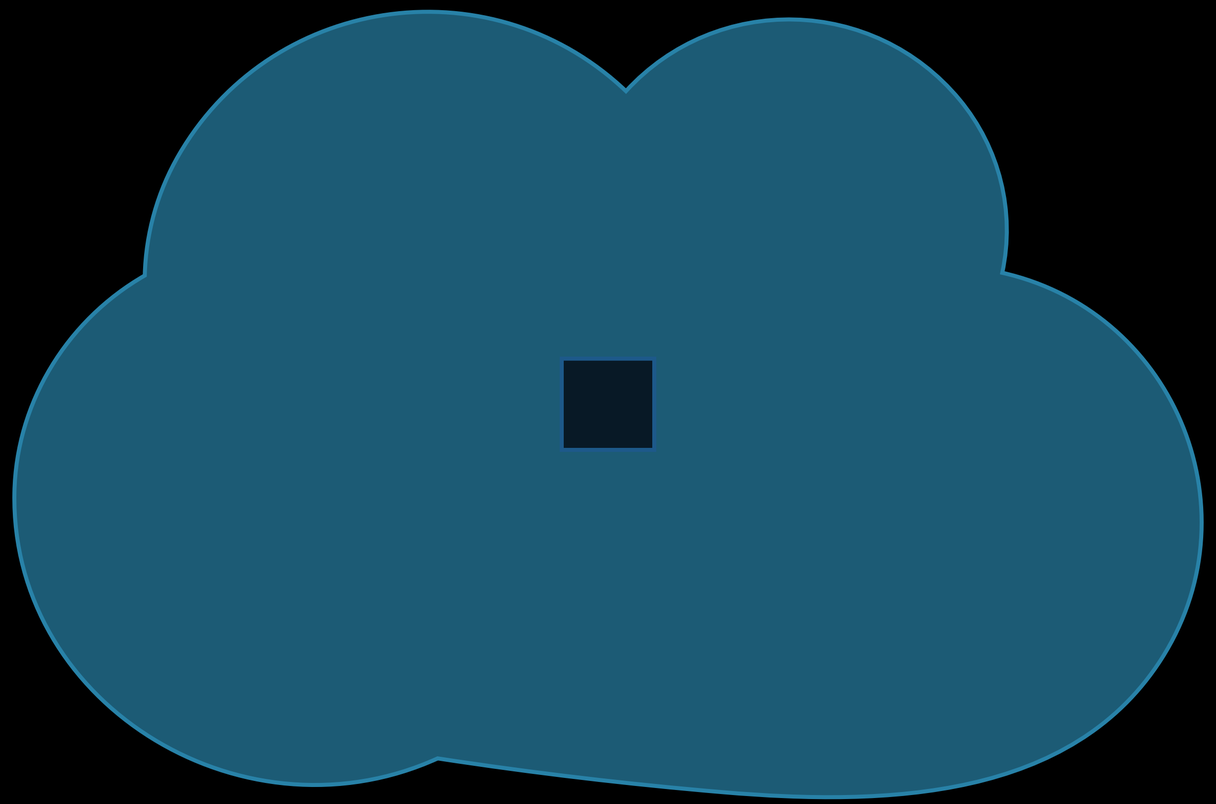
In the Beginning

NSLocking protocol

Developer had to lock the context before use

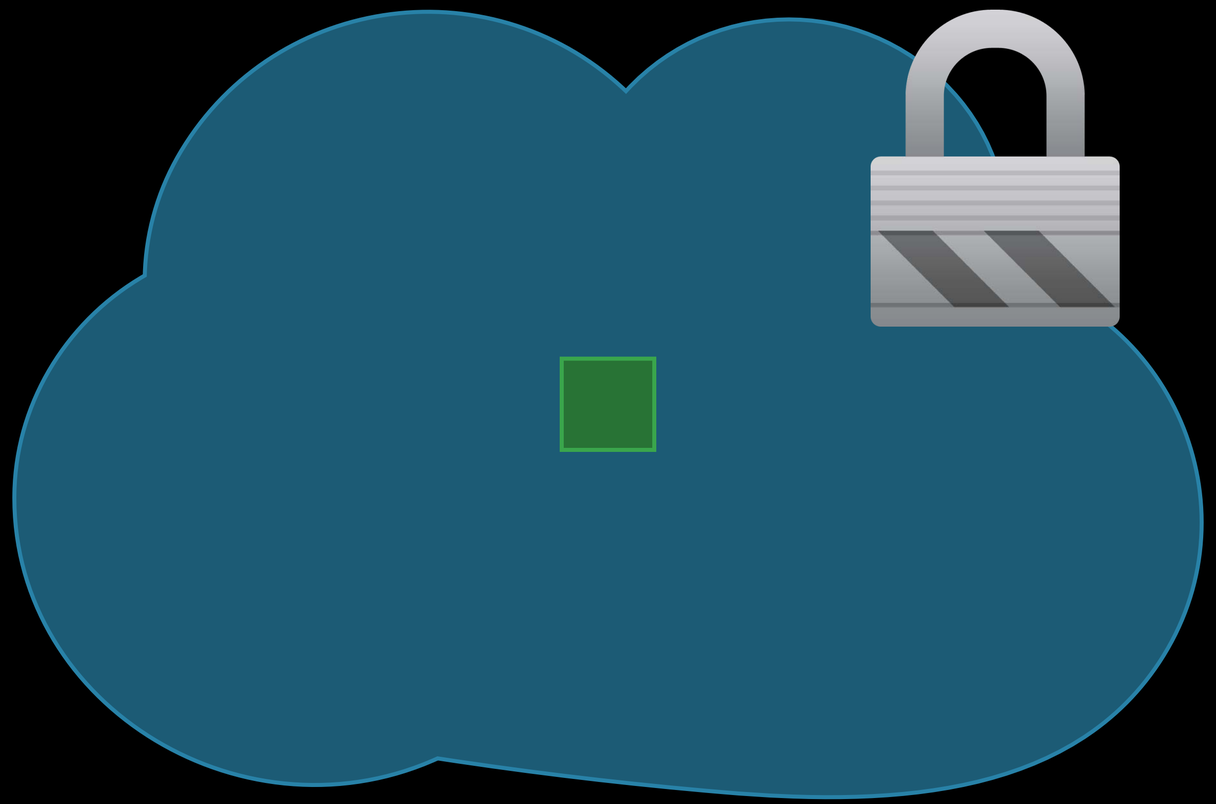
Developer had to lock the coordinator before use

NSLocking



NSLocking

NSLocking



NSLocking

NSLocking

NSLocking

NSLocking

In the Beginning

NSLocking protocol

Developer had to lock the context before use

Developer had to lock the coordinator before use

Easy to forget a lock or unlock

Thread Confinement

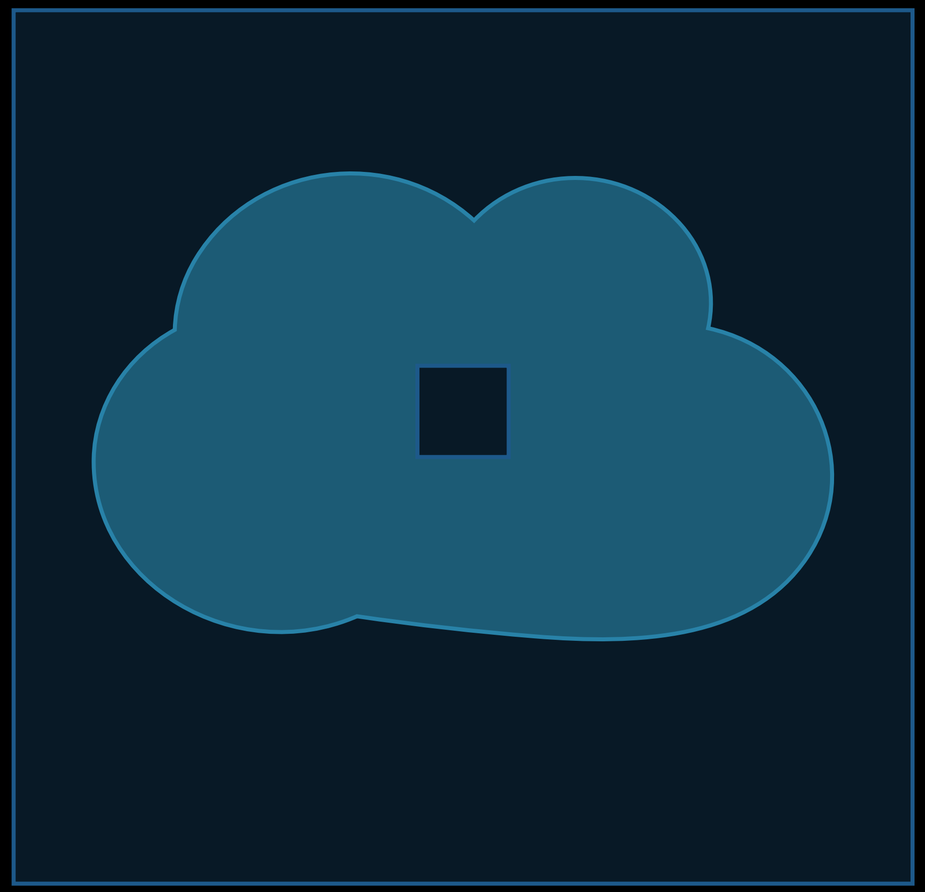
The first step

Developer responsible for making sure only one thread is using a context

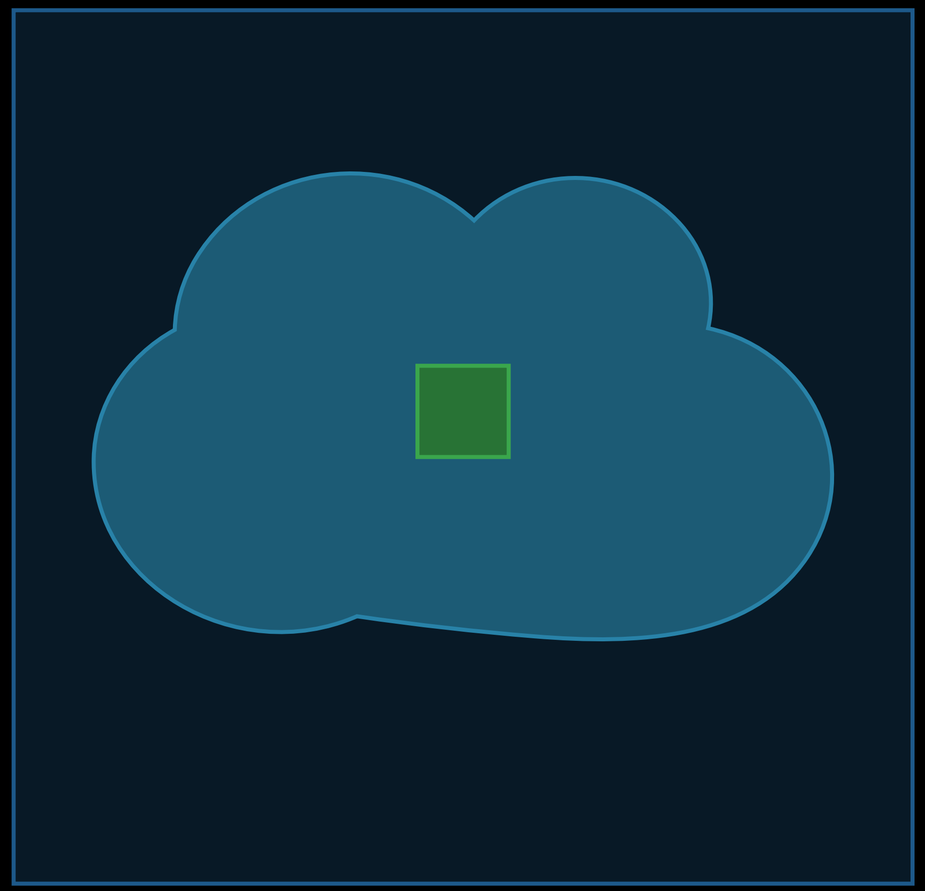
- Thread local variables

Developer had to lock coordinator before use

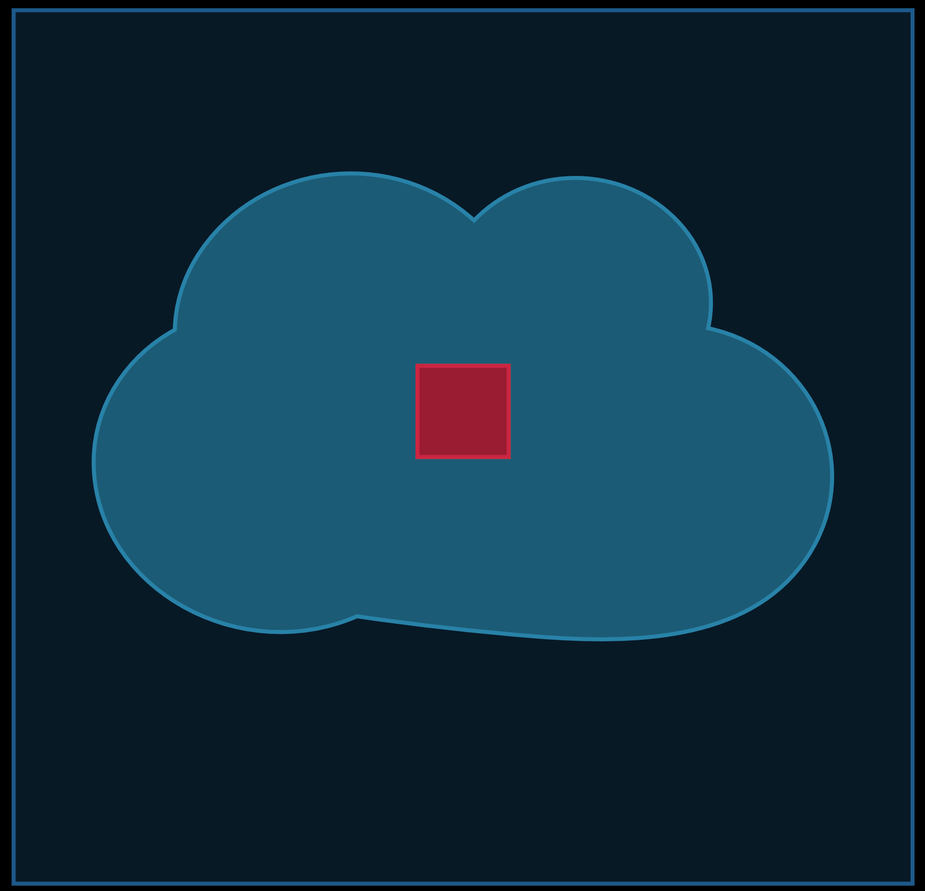
Thread Confinement



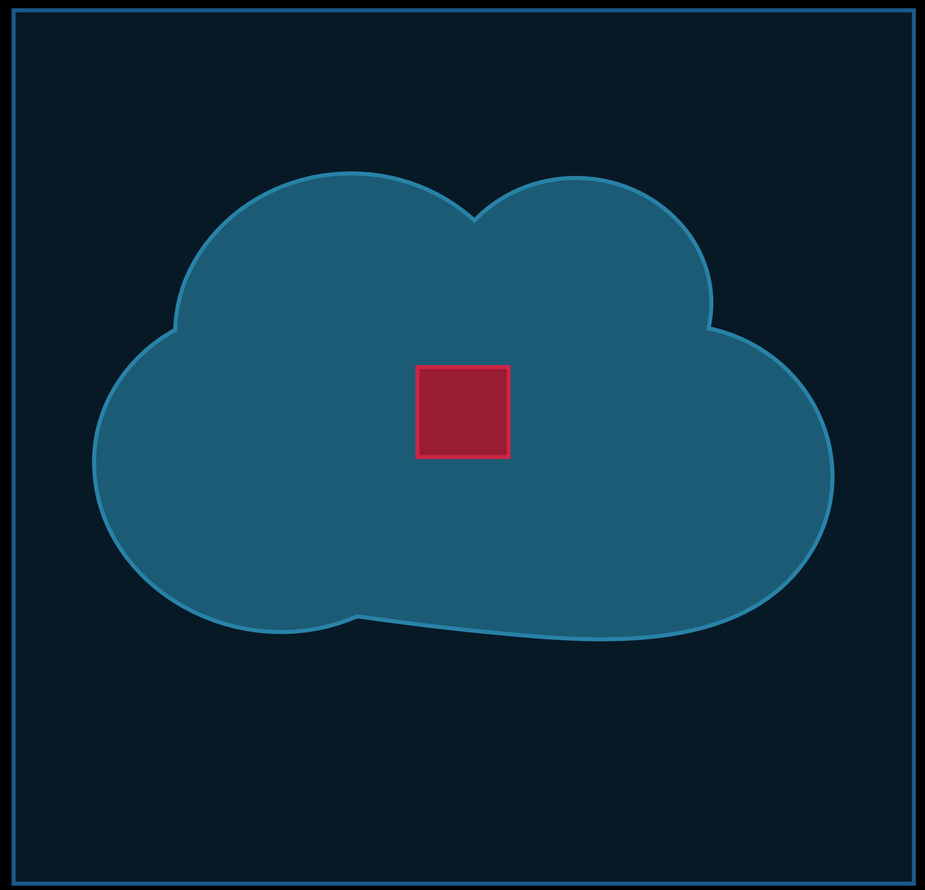
Thread Confinement



Thread Confinement



Thread Confinement



Thread Confinement

The first step

Developer responsible for making sure only one thread is using a context

- Thread local variables

Developer had to lock coordinator before use

Still difficult to get right

Concurrency Types

Making working with the context easier

Context encapsulates threading model

Concurrency types on context

- Private, main thread, confinement
- performBlock:/performBlockAndWait:

Developer no longer had to lock coordinator before use

Concurrency Types

Making working with the context easier

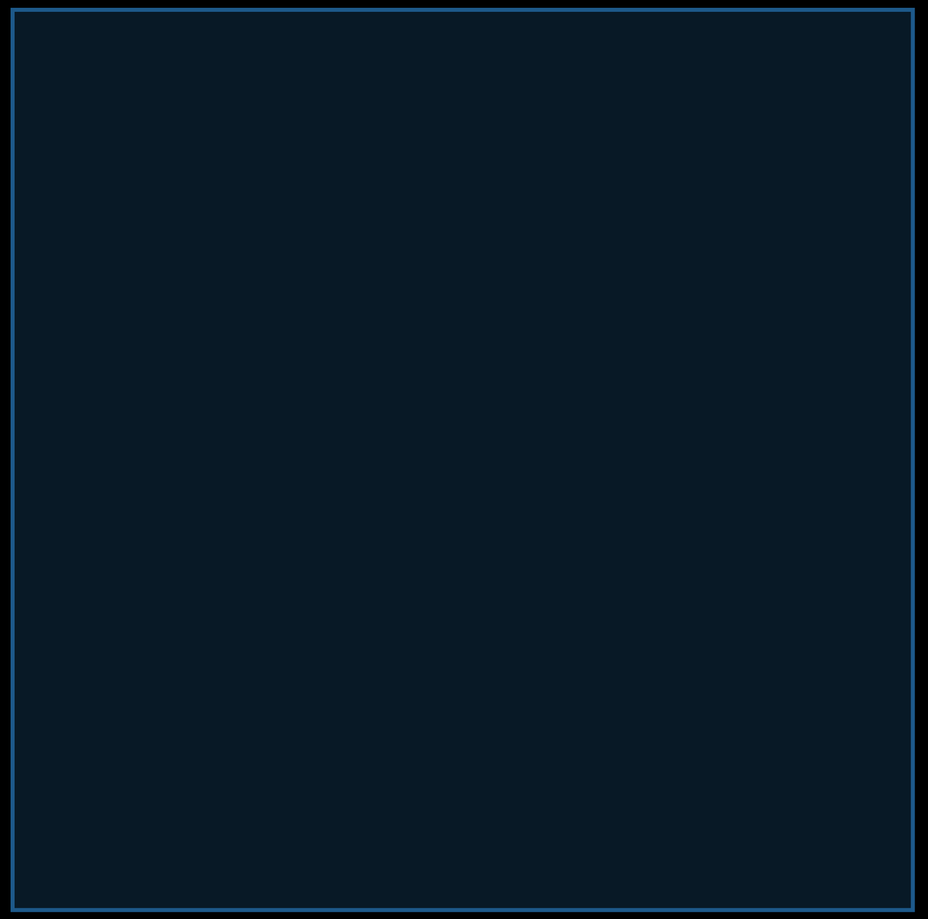
NSMainQueueConcurrencyType

NSPrivateQueueConcurrencyType

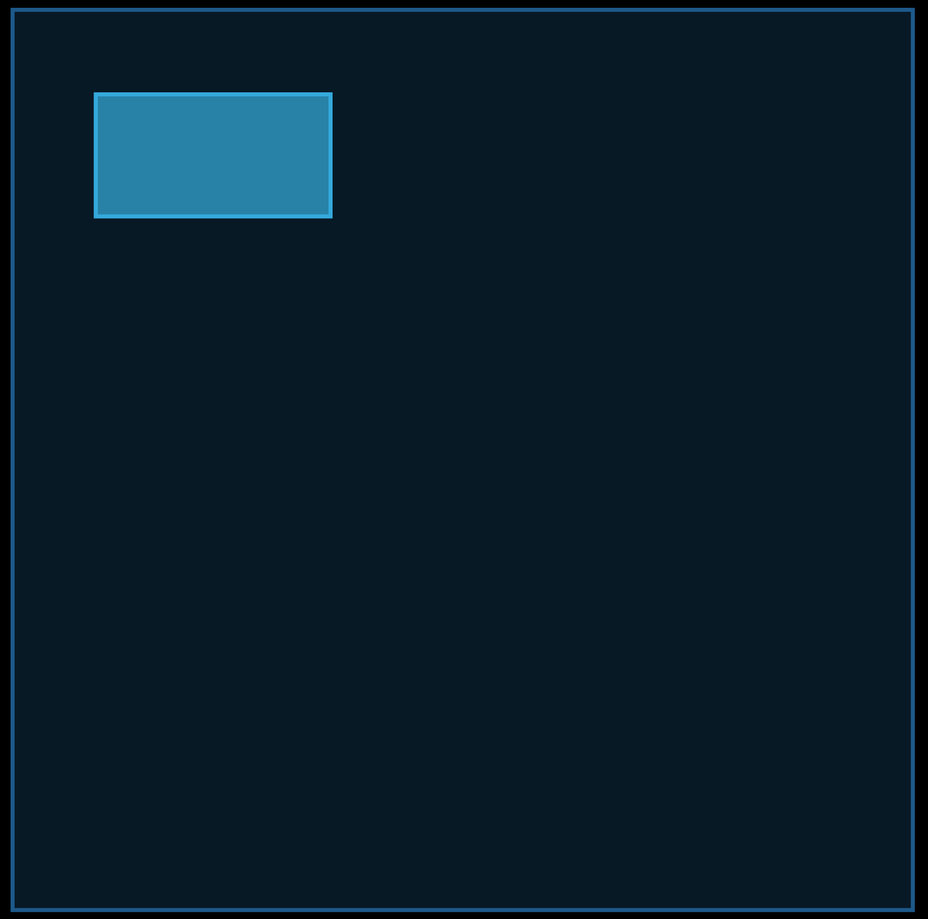
NSConfinementConcurrencyType

Inverting the Previous Order

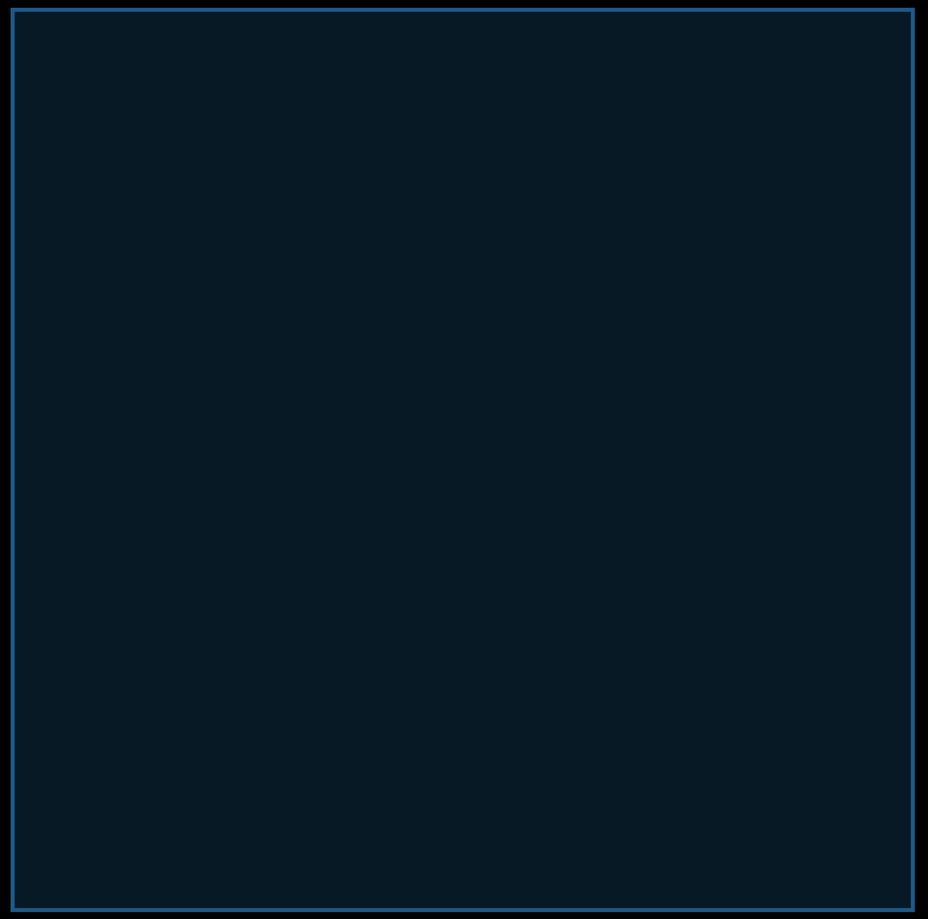
Inverting the Previous Order



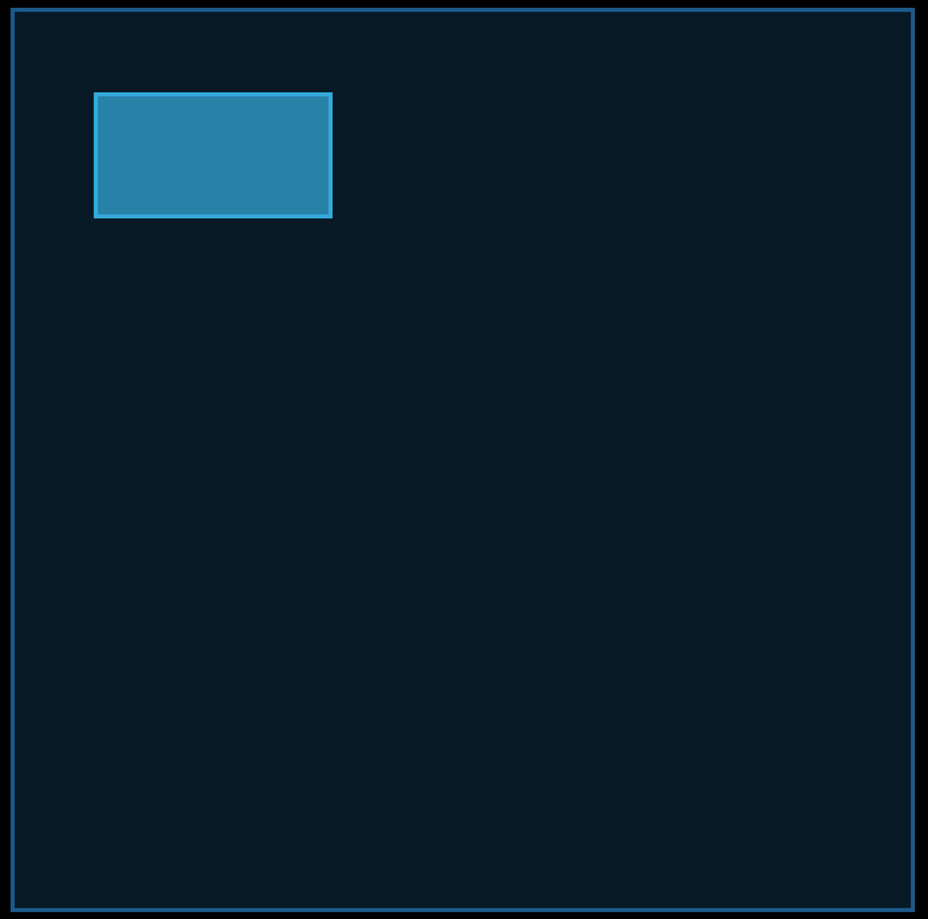
Inverting the Previous Order



Inverting the Previous Order



Inverting the Previous Order



Debugging

Had a mechanism for debugging concurrency

- com.apple.CoreData.ConcurrencyDebug 1

Required downloading a _debug version of the framework

Not available on iOS

Today

Preparing developers for tomorrow

OS X 10.10 and iOS 8.0

Same actors

Context encapsulates threading model

Concurrency types on context

- Private, main thread, confinement
- performBlock:/performBlockAndWait:

OS X 10.10 and iOS 8.0

More actors

NSPersistentStoreCoordinator gets performBlock:/performBlockAndWait:

- Existing methods wrap these

Always uses a private queue

OS X 10.10 and iOS 8.0

Debugging made easier

Debugging default now always available

- com.apple.CoreData.ConcurrencyDebug 1

Works on iOS

Looking Ahead

Predictions, not promises

Thread confinement obsolete

- That includes NSConfinementConcurrencyType

Bonus API

Because debugging is hard enough already

Name property on NSManagedObjectContext

- Only applies to NSPrivateQueueConcurrencyType

Name property on NSPersistentStoreCoordinator

Visible in Xcode/LLDB while debugging

iCloud

Core Data and iCloud

Transitioning to new infrastructure

Reliability improvements

Performance enhancements

Transparent to developers

iCloud Key Value Store

Great for application preferences

Asynchronously kept up to date

Data limit constraints

iCloud Documents

For document-centric apps

Simple API

Full offline cache on OS X

Unstructured data

Tied to the file system

iCloud Core Data

Keep private structured data in sync

Replicated between devices

Single user data

CloudKit

NEW

Client server model

No local storage

Predicate-based queries

Application-centric data

CloudKit

Public data

Structured and bulk data

Large data set

Use iCloud accounts

Client-directed data transfer

Swift

You have questions—We have answers

Core Data MUST[1] Work

[1] See RFC 2119

Swift

Yes

Full access to Core Data from Swift

Subclass NSManagedObject in Swift

Mix and match

Swift

Things to remember—Subclassing

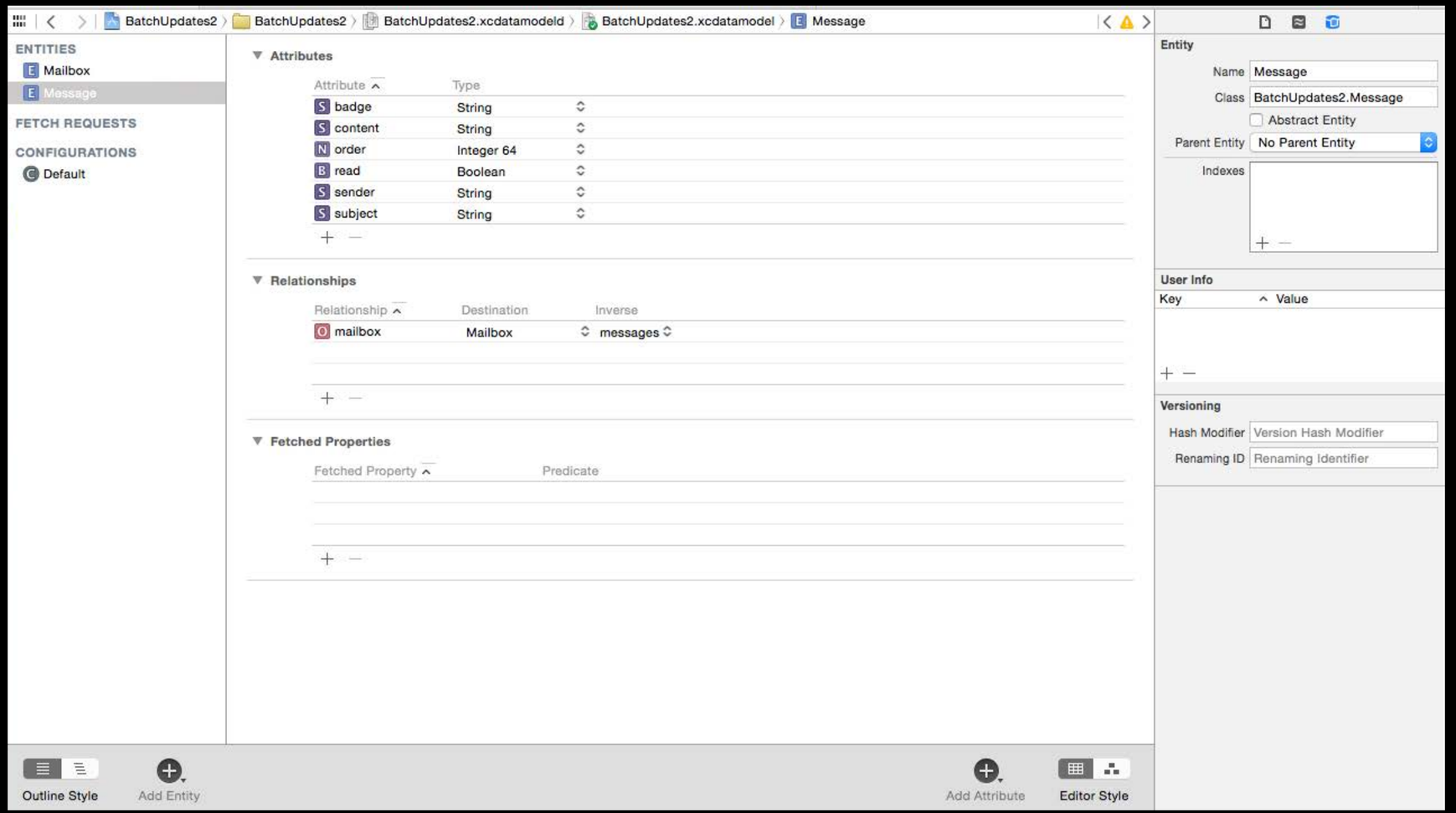
Swift subclasses work mostly like Objective C

Use @NSManaged

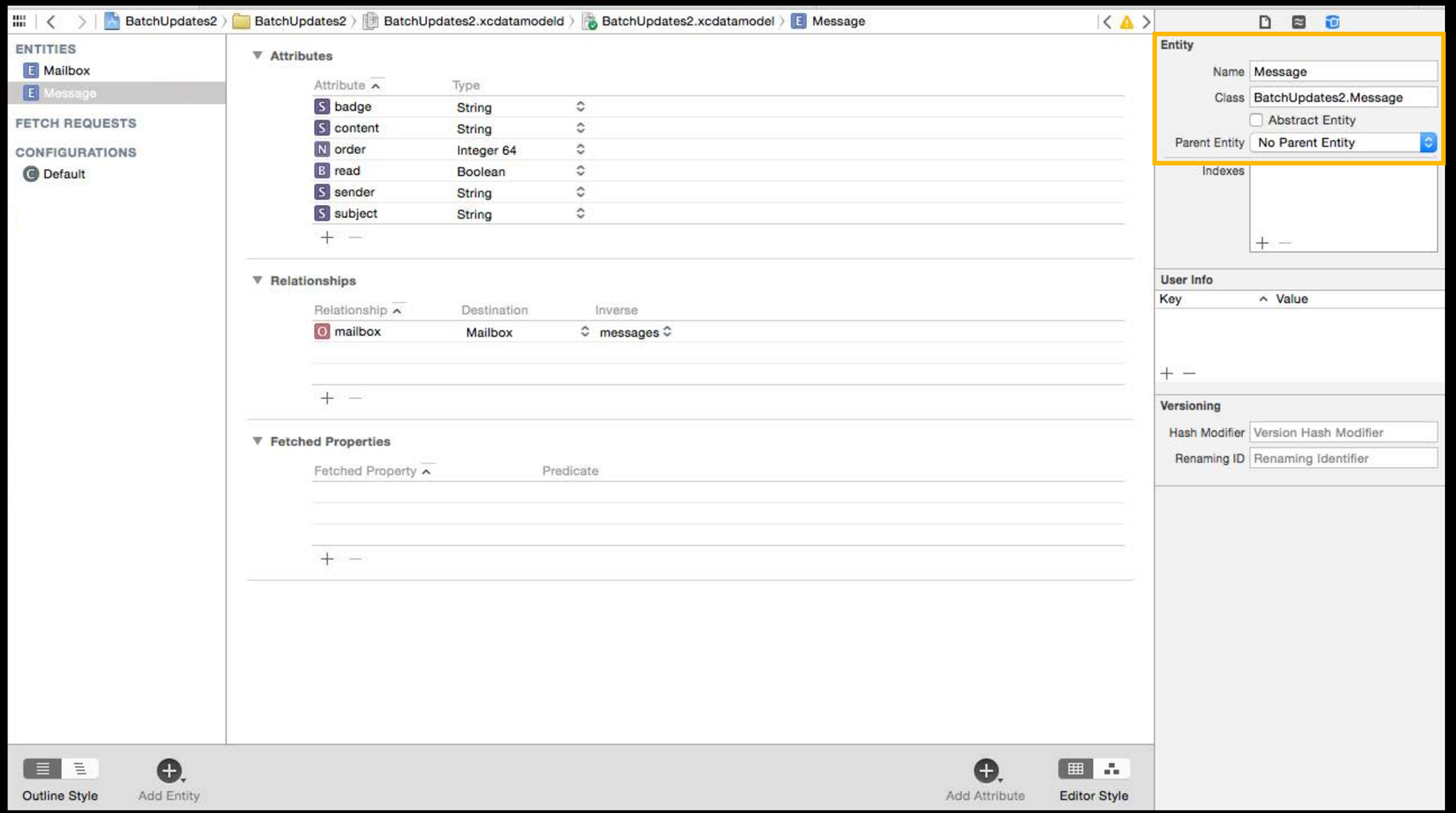
- Core Data specific

Module name in data model

Class Names in the Model



Class Names in the Model



Comparison

ObjC managed-object subclass

```
#import <CoreData/CoreData.h>

@interface Mailbox : NSManagedObject

@property (nonatomic) NSString *name;
@property (nonatomic) NSSet *messages;

@end
```

Comparison

ObjC managed-object subclass

```
#import “Mailbox.h”

@implementation Mailbox

@dynamic name;
@dynamic messages;

@end
```

Comparison

Swift object subclass

```
import CoreData

class Mailbox : NSManagedObject {

    @NSManaged var name : NSString
    @NSManaged var messages : NSSet

}
```

Swift

Things to remember—Types

CoreData doesn’t use type constraints

If you do

- Custom subclasses are best
- NSManagedObject otherwise

Swift

Things to remember—Types

CoreData doesn’t use type constraints

If you are

- Custom subclasses are best
- NSManagedObject otherwise

```
var myArray = MyManagedSubclass[]
func myFunction<T : MyManagedSubclass>(first: T) {}
```

Demo

Batch updates redux

Roadmap

Batch updates

Asynchronous fetching

Incremental stores

Concurrency changes

iCloud update

Swift

http://bugreport.apple.com

We can’t fix what we don’t know about

Bug reports

- Steps to reproduce
- Sample App bonus

Feature requests

Enhancement requests

Performance issues

Documentation requests

More Information

David DeLong
Technology Evangelist
delong@apple.com

Cocoa Feedback
cocoa-feedback@apple.com

Core Data Documentation
Programming Guide, Example, Tutorials
http://developer.apple.com

Apple Developer Forums
http://devforums.apple.com

Related Sessions

• Introducing CloudKit	Mission	Tuesday 3:15 PM

Labs

• Core Data	Services Lab B	Wednesday 9:00AM
• Core Data	Services Lab B	Thursday 10:15AM
• Core Data	Frameworks Lab	Friday 9:00AM

WWDC14