

#WWDC19

Making Apps with Core Data

Scott Perry, Core Data Engineer

Getting started

The needs of the Controller

Scaling your app

Testing

Getting started

The needs of the Controller

Scaling your app

Testing



9:41



FaceTime



Calendar



Photos



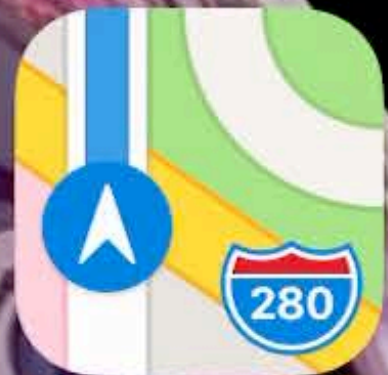
Camera



Mail



Clock



Maps



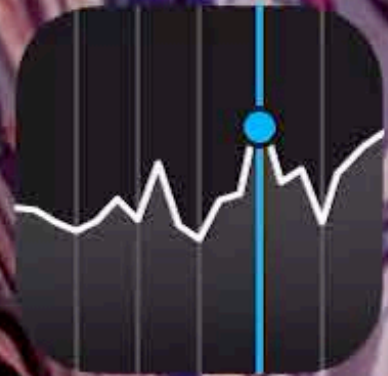
Weather



Reminders



Notes



Stocks



News



Books



App Store



Podcasts



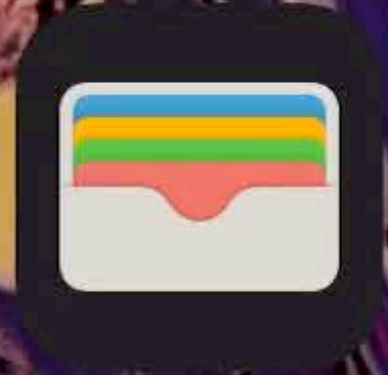
TV



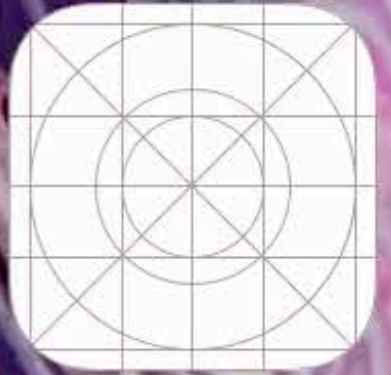
Health



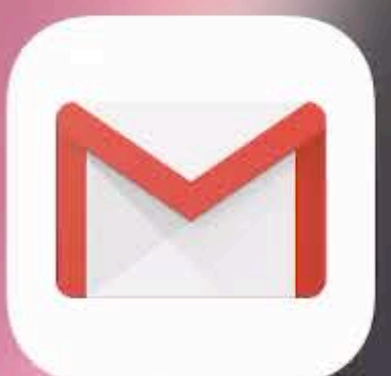
Home



Wallet



CoreDataCloudKit





9:41



FaceTime



Calendar



Photos



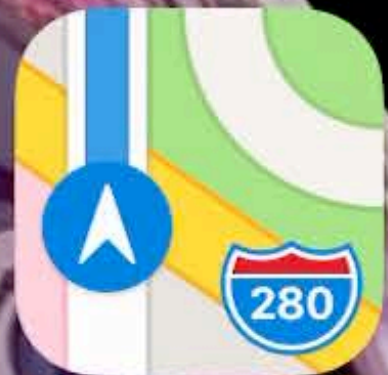
Camera



Mail



Clock



Maps



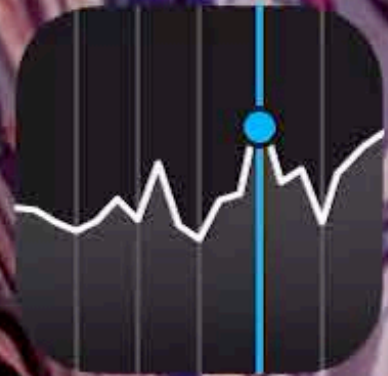
Weather



Reminders



Notes



Stocks



News



Books



App Store



Podcasts



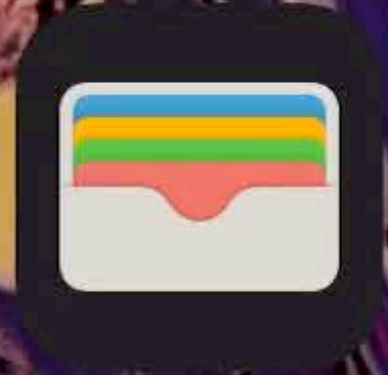
TV



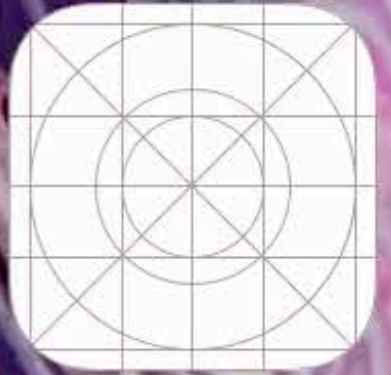
Health



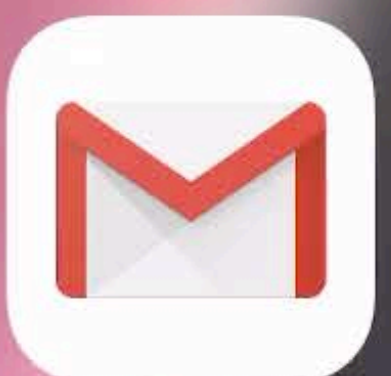
Home



Wallet



CoreDataCloudKit



9:41



Edit

Posts



Visit the labs!



demo

Welcome to the session!



demo



9:41



Edit

Posts



Visit the labs!



demo

Welcome to the session!



demo



9:41



Edit

Posts



Untitled 9:41:00 AM



Visit the labs!



demo

Welcome to the session!



demo



9:41



Edit

Posts



Untitled 9:41:00 AM



Visit the labs!

demo



Welcome to the session!

demo



9:41



[← Posts](#)

Detail

[Edit](#)

TITLE

Untitled 9:41:00 AM

CONTENT

TAGS



ATTACHMENTS



9:41



[← Posts](#)

Detail

[Edit](#)

TITLE

Untitled 9:41:00 AM

CONTENT

TAGS



ATTACHMENTS



9:41



[← Posts](#)

Detail

[Edit](#)

TITLE

Hi WWDC!

CONTENT

TAGS

[demo](#)



ATTACHMENTS



9:41



[← Posts](#)

Detail

[Edit](#)

TITLE

Hi WWDC!

CONTENT

TAGS

[demo](#)



ATTACHMENTS



9:41



Edit

Posts



Hi WWDC!



demo

Visit the labs!



demo

Welcome to the session!



demo



9:41



Edit

Posts



Hi WWDC!



demo

Visit the labs!



demo

Welcome to the session!



demo



9:41 ↗



Edit

Tag



demo (3)

Dismiss

9:41 ↗



Edit

Tag



demo (3)

Dismiss

9:41



Edit

Posts



Hi WWDC!



demo

Visit the labs!



demo

Welcome to the session!



demo



9:41



Edit

Posts



Hi WWDC!



demo

Visit the labs!



demo

Welcome to the session!



demo



9:41



Edit

Posts



Hi WWDC!

demo



Untitled 9:41:00 AM - 0

cats demo dogs



Untitled 9:41:00 AM - 1

dogs demo



Untitled 9:41:00 AM - 10

demo dogs cats



Untitled 9:41:00 AM - 100

demo cats



Untitled 9:41:00 AM - 101

cats demo dogs



Untitled 9:41:00 AM - 102

cats dogs



Untitled 9:41:00 AM - 103

demo cats



Untitled 9:41:00 AM - 104

dogs demo cats



Untitled 9:41:00 AM - 105



9:41



Edit

Posts



Hi WWDC!

demo



Untitled 9:41:00 AM - 0

cats demo dogs



Untitled 9:41:00 AM - 1

dogs demo



Untitled 9:41:00 AM - 10

demo dogs cats



Untitled 9:41:00 AM - 100

demo cats



Untitled 9:41:00 AM - 101

cats demo dogs



Untitled 9:41:00 AM - 102

cats dogs



Untitled 9:41:00 AM - 103

demo cats



Untitled 9:41:00 AM - 104

dogs demo cats



Untitled 9:41:00 AM - 105

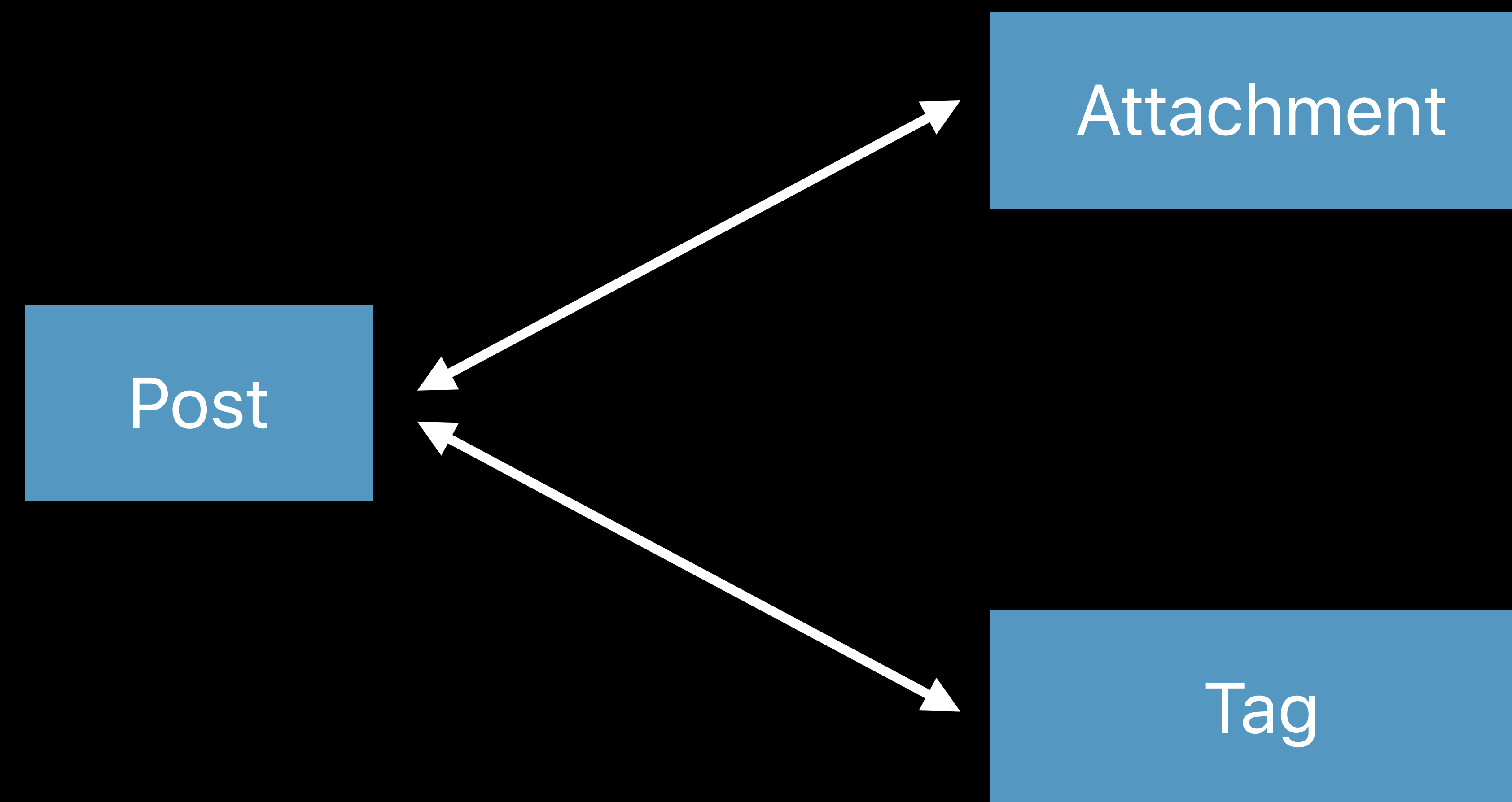


Modeling Data

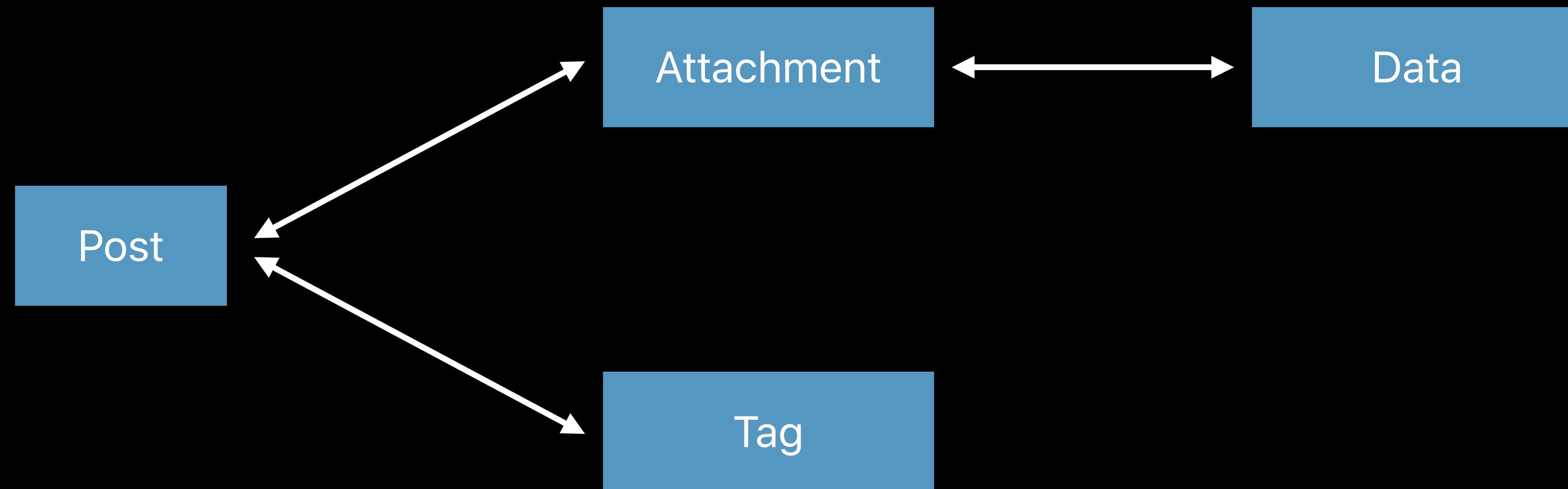
Modeling Data

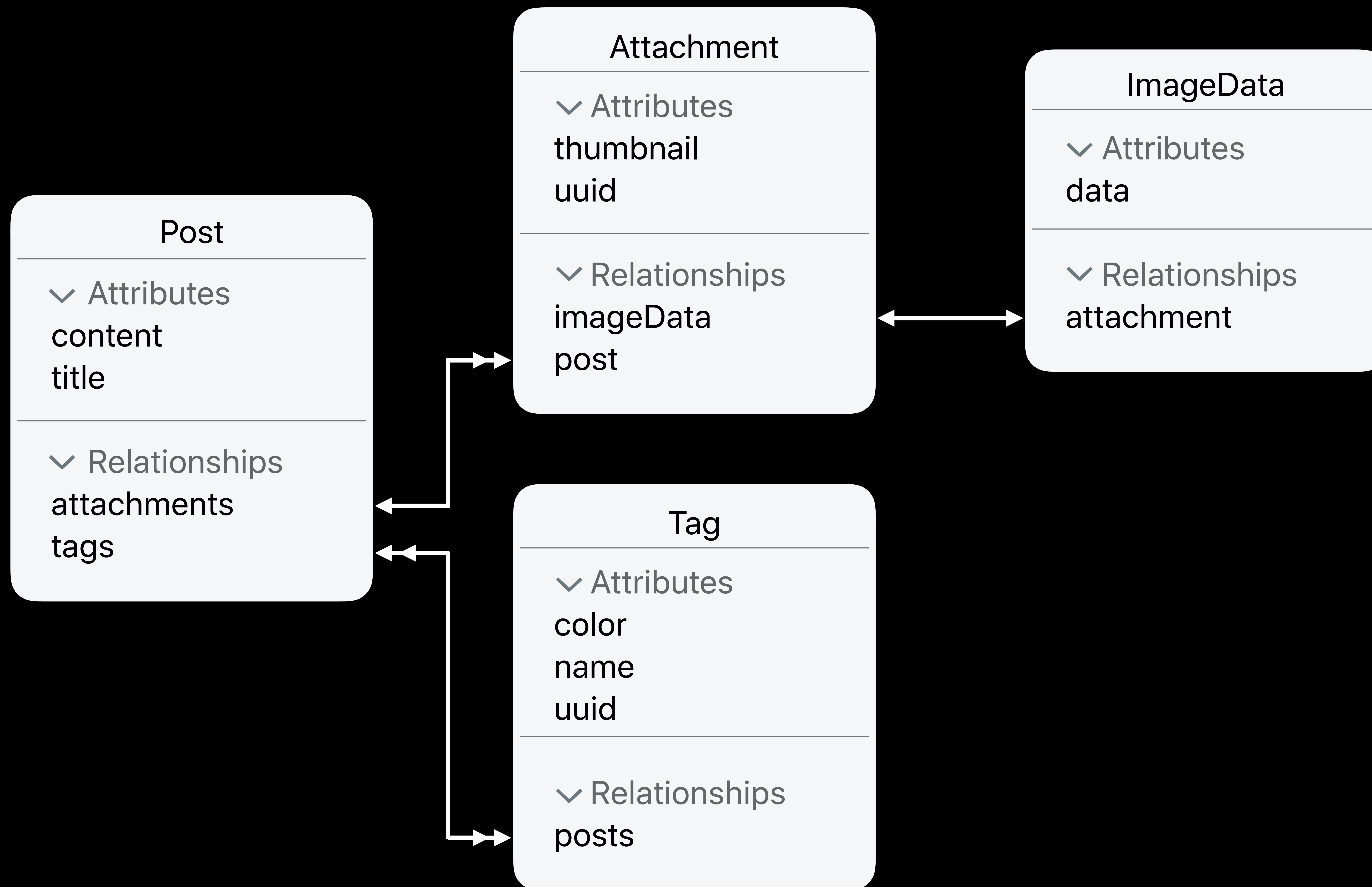
Post

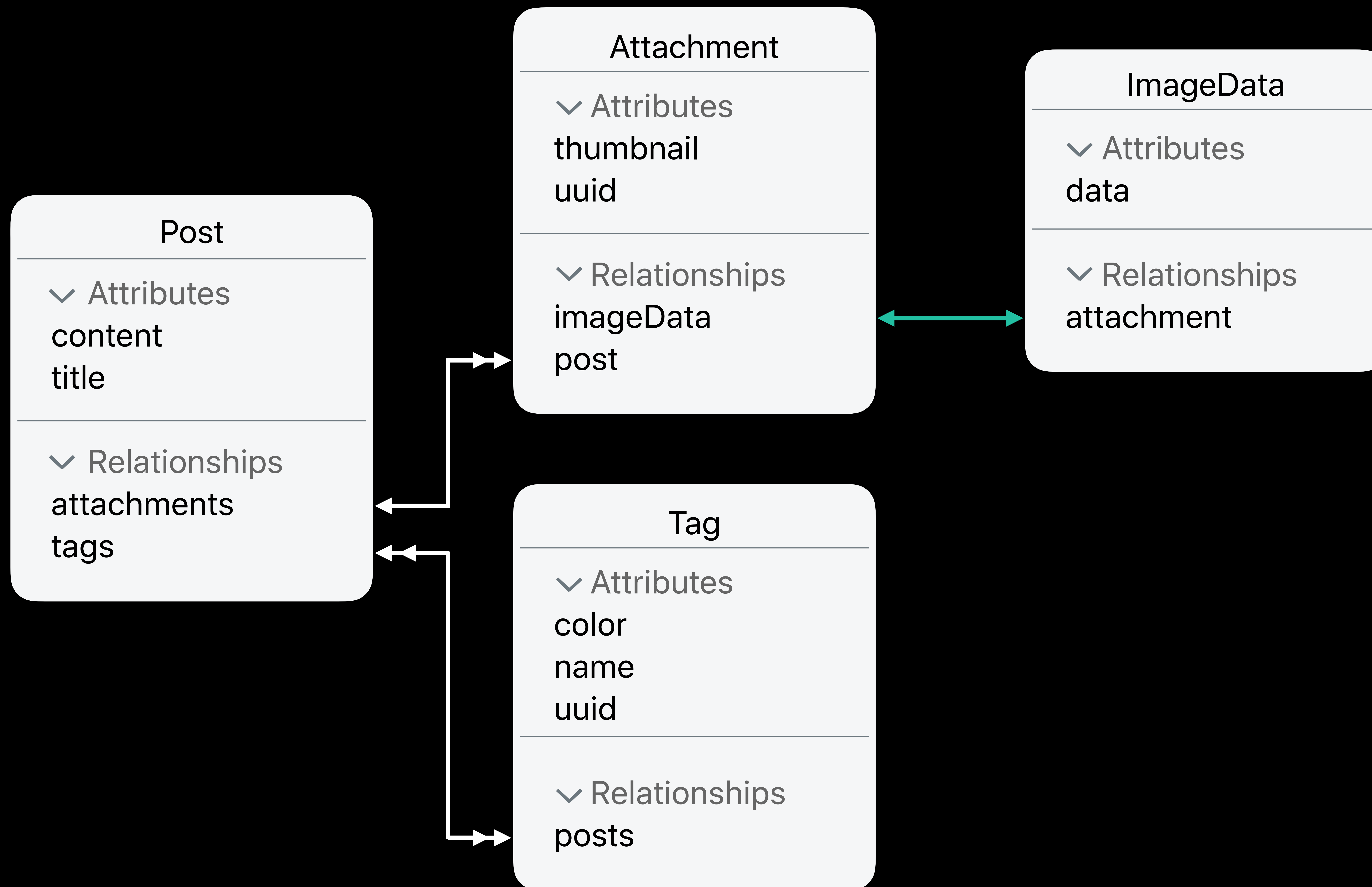
Modeling Data

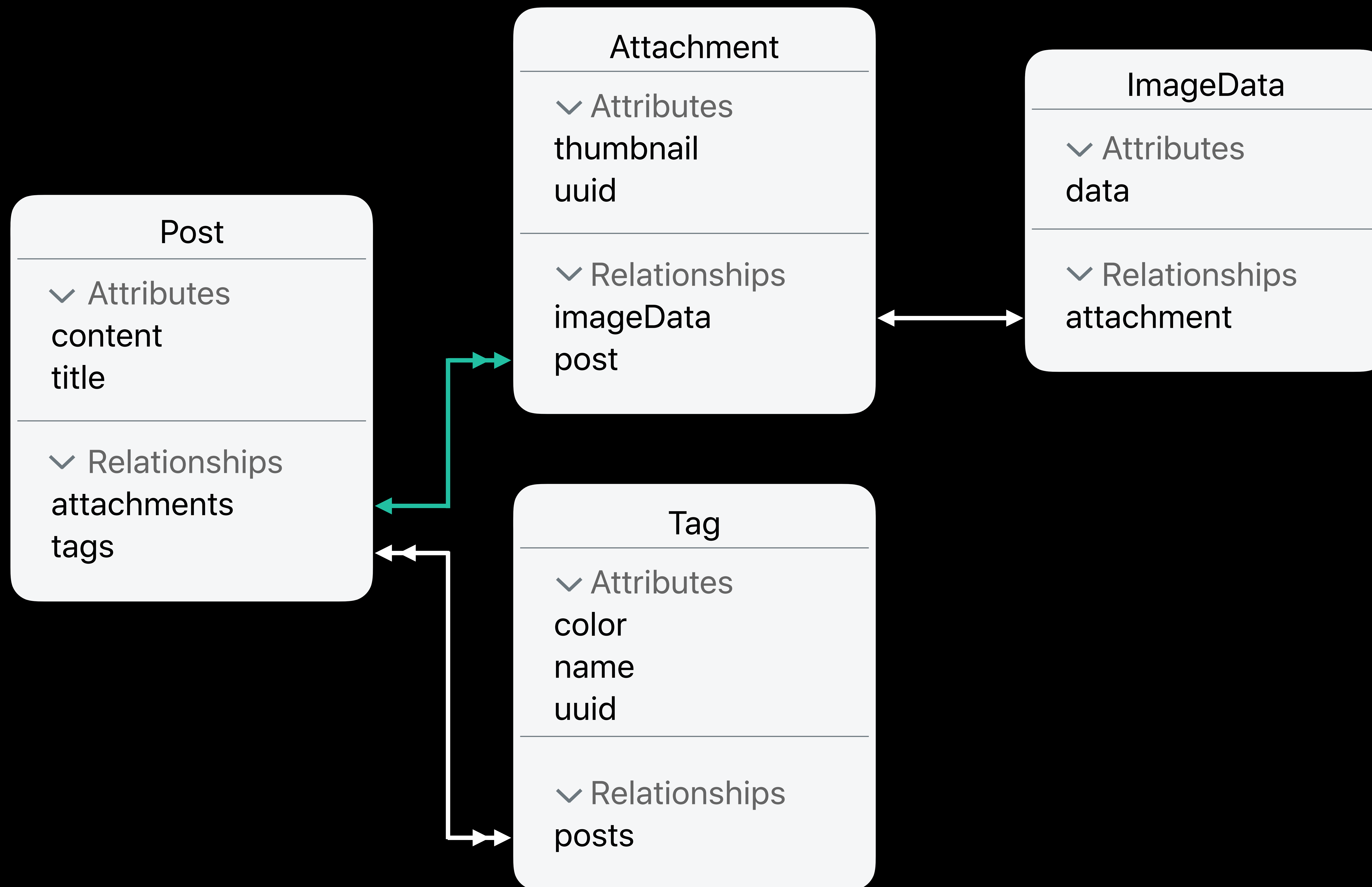


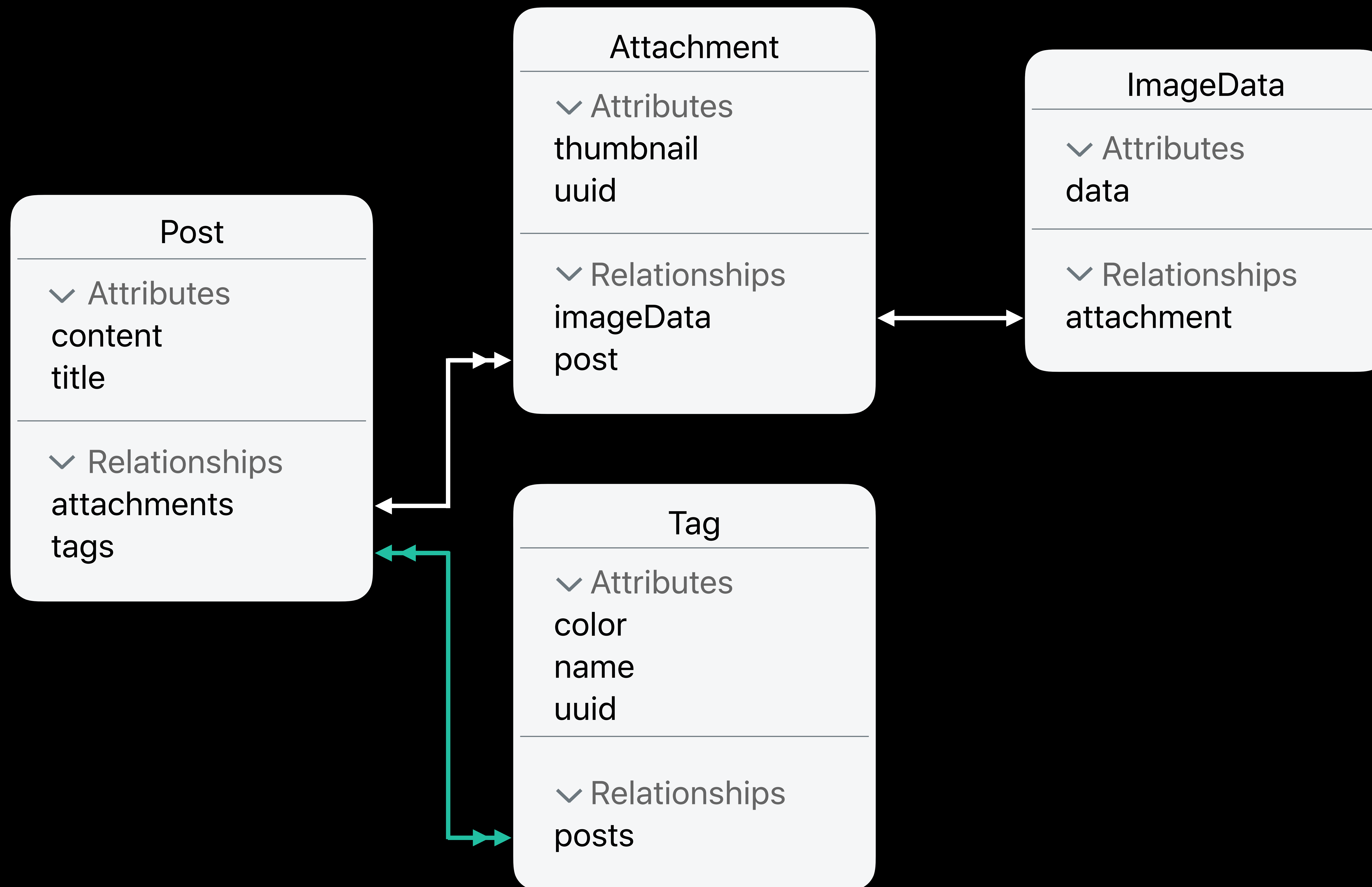
Modeling Data











The Core Data Stack

The Core Data Stack

Model

`NSManagedObjectModel`

The Core Data Stack

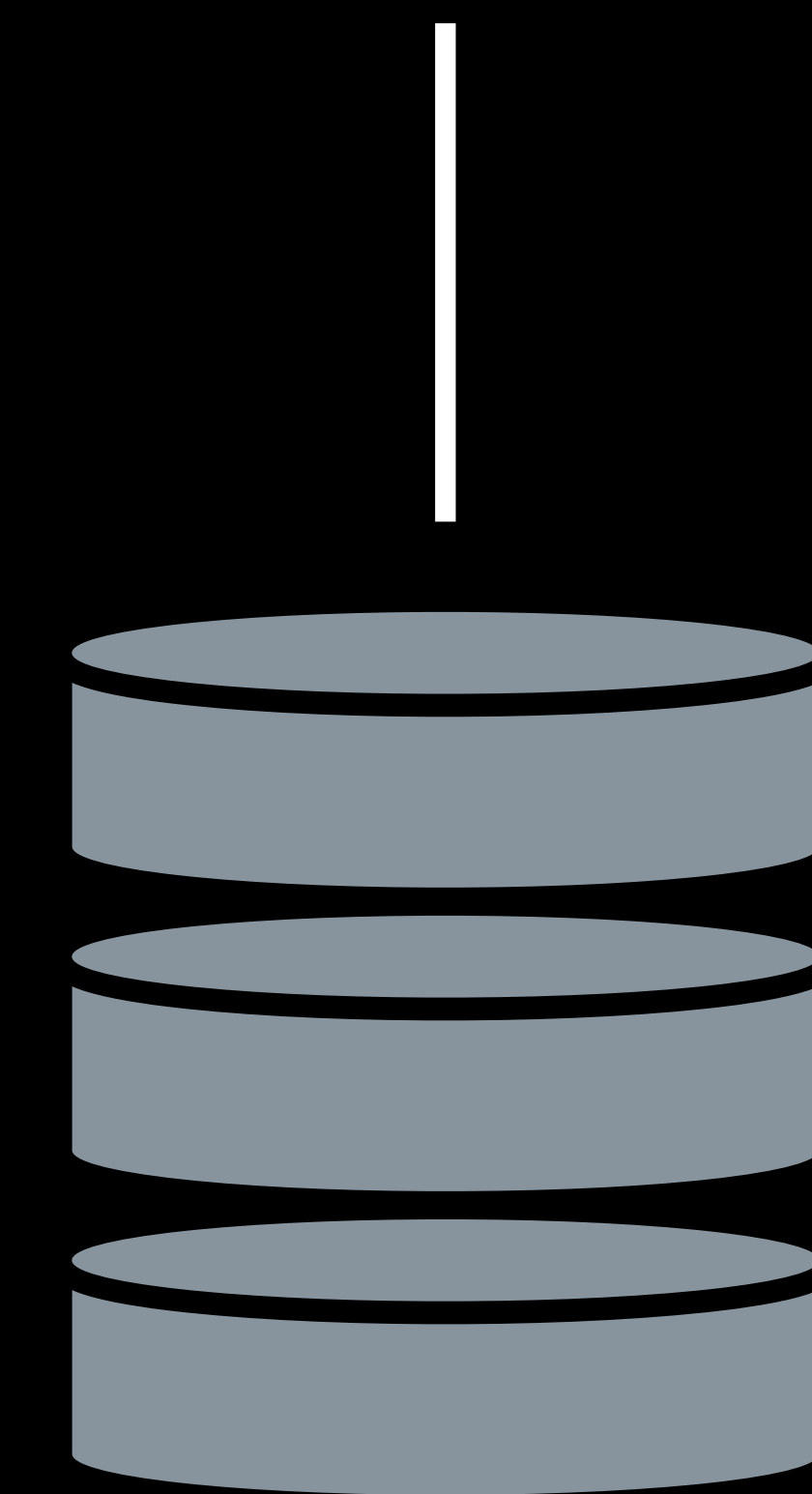
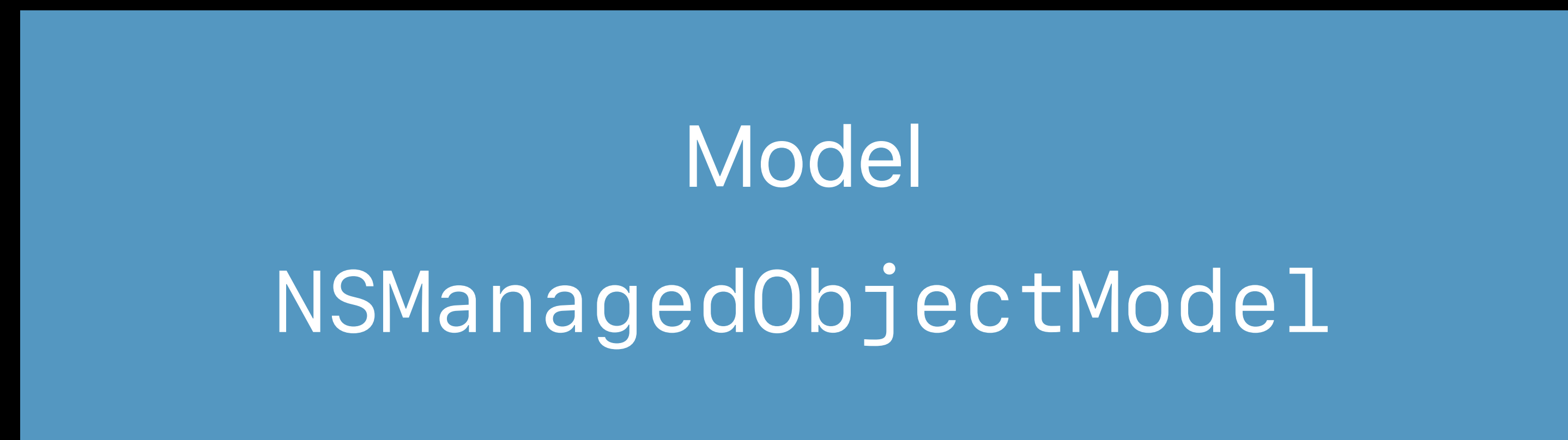
Model

`NSManagedObjectModel`

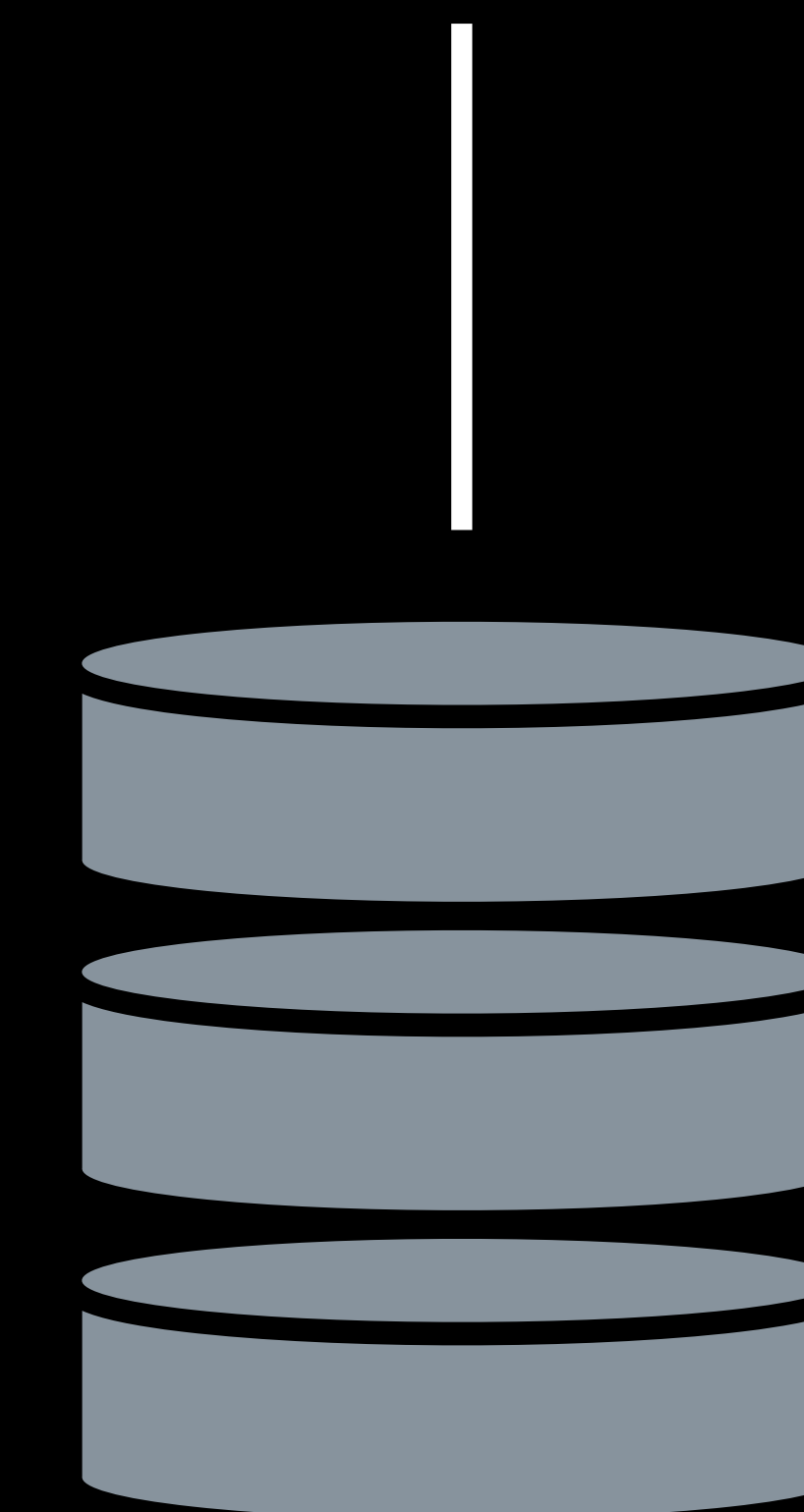
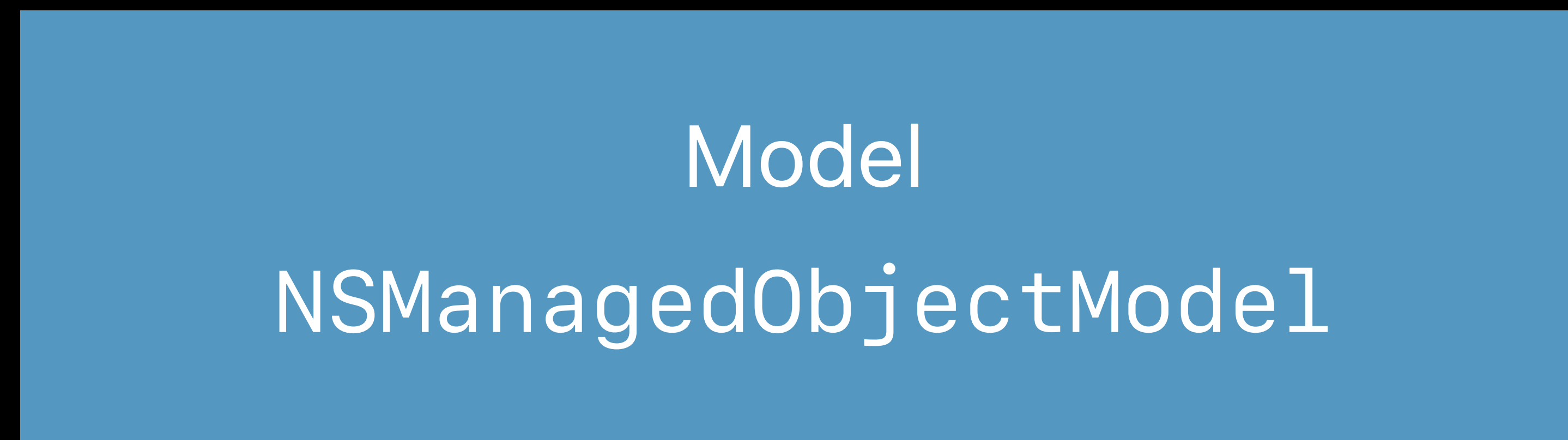
Store coordinator

`NSPersistentStoreCoordinator`

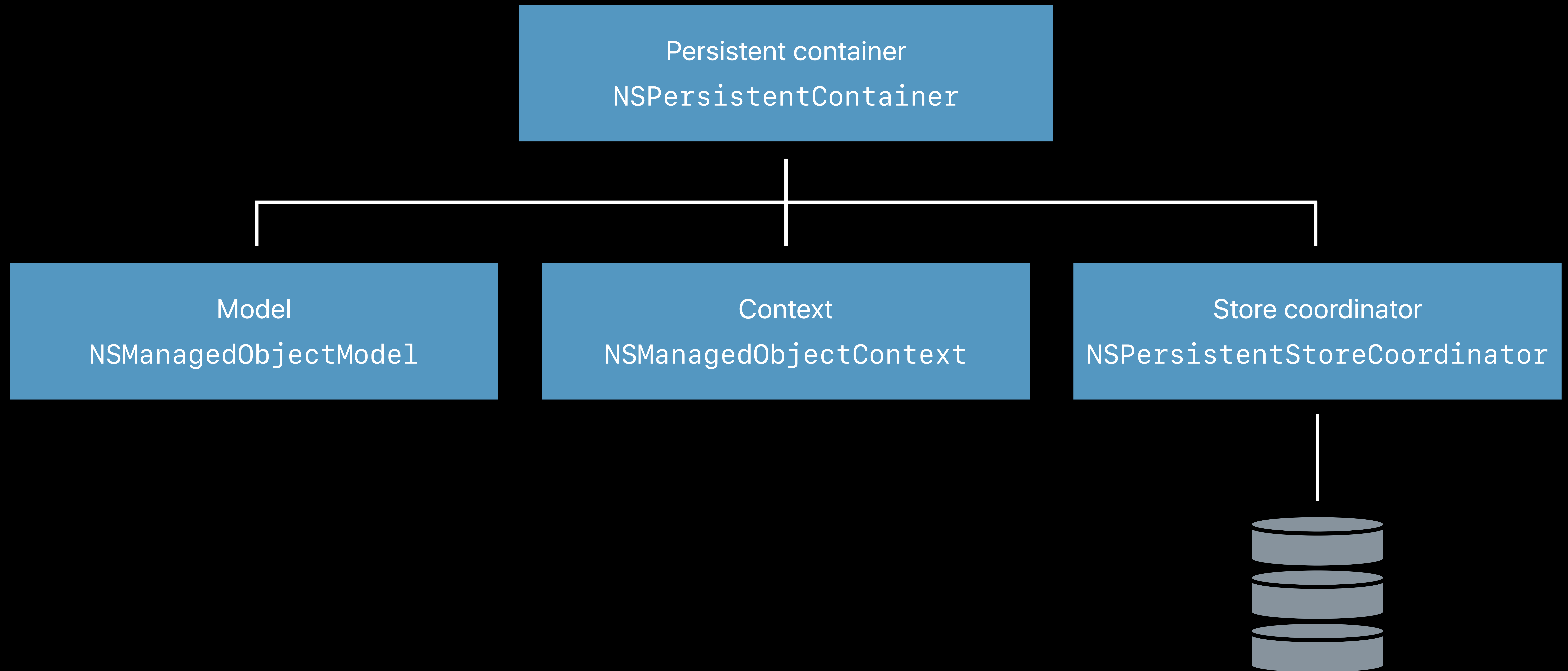
The Core Data Stack



The Core Data Stack



The Core Data Stack



Setting Up the Stack

By name

```
let container = NSPersistentCloudKitContainer(name: "CoreDataCloudKitDemo")
```

Setting Up the Stack

By name

```
let container = NSPersistentCloudKitContainer(name: "CoreDataCloudKitDemo")
```

With a model

```
let container = NSPersistentCloudKitContainer(  
    name: "CoreDataCloudKitDemo",  
    managedObjectModel:model)
```


Setting Up the Stack

By name

```
let container = NSPersistentCloudKitContainer(name: "CoreDataCloudKitDemo")
container.loadPersistentStores { _, error in /* ... */ }
```

With a model

```
let container = NSPersistentCloudKitContainer(
    name: "CoreDataCloudKitDemo",
    managedObjectModel:model)
container.loadPersistentStores { _, error in /* ... */ }
```

Configuring Managed Object Contexts

Configuring Managed Object Contexts

Query generations provide stability

```
try container.viewContext.setQueryGenerationFrom(.current)
```


Configuring Managed Object Contexts

Query generations provide stability

```
try container.viewContext.setQueryGenerationFrom(.current)
```

Automatic merging provides freshness

```
context.automaticallyMergesChangesFromParent = true
```

Using Managed Object Contexts

Using Managed Object Contexts

```
context.performAndWait {  
    /* ... */  
}
```

```
context.perform {  
    /* ... */  
}
```

```
container.performBackgroundTask { context in  
    /* ... */  
}
```


Apps Need Data!

Use `init(context:)` to create individual managed objects

Apps Need Data!

Use `init(context:)` to create individual managed objects

```
context.perform {  
    let post = Post(context: context)  
    post.title = "Hello, world!"  
    try? context.save()  
}
```

Apps need *more* data!

Batch Insertions

Insert many managed objects with a fraction of the overhead

```
let rawPostsData: Data = // Server response ...
if let postDicts = try? JSONSerialization.jsonObject(with:rawPostsData) as? [[String : Any]] {
    context.perform {
        let insertRequest = NSBatchInsertRequest(entity: Post.entity(), objects: postDicts)
        let insertResult = try? context.execute(insertRequest) as! NSBatchInsertRequest
        let success = insertResult.result as! Bool
    }
}
```

Batch Insertions

NEW

Insert many managed objects with a fraction of the overhead

```
let rawPostsData: Data = // Server response ...
if let postDicts = try? JSONSerialization.jsonObject(with:rawPostsData) as? [[String : Any]] {
    context.perform {
        let insertRequest = NSBatchInsertRequest(entity: Post.entity(), objects: postDicts)
        let insertResult = try? context.execute(insertRequest) as! NSBatchInsertRequest
        let success = insertResult.result as! Bool
    }
}
```

Batch Insertions



NEW

Insert many managed objects with a fraction of the overhead

```
let rawPostsData: Data = // Server response ...
if let postDicts = try? JSONSerialization.jsonObject(with:rawPostsData) as? [[String : Any]] {
    context.perform {
        let insertRequest = NSBatchInsertRequest(entity: Post.entity(), objects: postDicts)
        let insertResult = try? context.execute(insertRequest) as! NSBatchInsertRequest
        let success = insertResult.result as! Bool
    }
}
```

Batch Insertions

```
[  
  [  
    "content": "Lorem ipsum dolor sit amet..."  
    "title"   : "Hello, world!",  
  ],  
  [  
    "content": "This post has no title!"  
  ],  
  [  
    "title": "Content coming soon!"  
  ]  
]
```

Post
∨ Attributes
content
title
∨ Relationships
attachment
tags

Batch Insertions



NEW

Insert many managed objects with a fraction of the overhead

```
let rawPostsData = // ...
if let postDicts = try? JSONSerialization.jsonObject(with:rawPostsData) as? [String : Any] {
    moc.perform {
        let insertRequest = NSBatchInsertRequest(entity: Post.entity(), objects: postDicts)
        let insertResult = try? moc.execute(insertRequest) as! NSBatchInsertRequest
        let success = insertResult.result as! Bool
    }
}
```

Batch Insertions

NEW

Insert many managed objects with a fraction of the overhead

```
let rawPostsData = // ...
if let postDicts = try? JSONSerialization.jsonObject(with:rawPostsData) as? [String : Any] {
    moc.perform {
        let insertRequest = NSBatchInsertRequest(entity: Post.entity(), objects: postDicts)
        let insertResult = try? moc.execute(insertRequest) as! NSBatchInsertRequest
        let success = insertResult.result as! Bool
    }
}
```

But What About...

But What About...

Unique constraints

But What About...

Unique constraints

Omitted keys

But What About...

Unique constraints

Omitted keys

Relationships

But What About...

Unique constraints

Omitted keys

Relationships

Notifications

Getting started

The needs of the Controller

Scaling your app

Testing

Fetching an Object

```
let fetchRequest : NSFetchRequest<Tag> = Tag.fetchRequest()

fetchRequest.predicate = NSPredicate(format: "name = %@", tagName)

if let tag = try? fetchRequest.execute().first {

    tagLabel.text = tag.name
    tagLabel.textColor = tag.color as? UIColor

}
```

Fetching an Object

```
let fetchRequest : NSFetchRequest<Tag> = Tag.fetchRequest()

fetchRequest.predicate = NSPredicate(format: "name = %@", tagName)

if let tag = try? fetchRequest.execute().first {

    tagLabel.text = tag.name
    tagLabel.textColor = tag.color as? UIColor

}
```

Fetching an Object

```
let fetchRequest : NSFetchRequest<Tag> = Tag.fetchRequest()

fetchRequest.predicate = NSPredicate(format: "name = %@", tagName)

if let tag = try? fetchRequest.execute().first {

    tagLabel.text = tag.name
    tagLabel.textColor = tag.color as? UIColor

}
```

Fetching an Object

```
let fetchRequest : NSFetchRequest<Tag> = Tag.fetchRequest()
```

```
fetchRequest.predicate = NSPredicate(format: "name = %@", tagName)
```

```
if let tag = try? fetchRequest.execute().first {
```

```
    tagLabel.text = tag.name
```

```
    tagLabel.textColor = tag.color as? UIColor
```

```
}
```


Fetching an Object

```
let fetchRequest : NSFetchRequest<Tag> = Tag.fetchRequest()
```

```
fetchRequest.predicate = NSPredicate(format: "name = %@", tagName)
```

```
if let tag = try? fetchRequest.execute().first {
```

```
    tagLabel.text = tag.name
```

```
    tagLabel.textColor = tag.color as? UIColor
```

```
}
```

Fetching an Object

```
let fetchRequest : NSFetchRequest<Tag> = Tag.fetchRequest()

fetchRequest.predicate = NSPredicate(format: "name = %@", tagName)

if let tag = try? fetchRequest.execute().first {

    tagLabel.text = tag.name
    tagLabel.textColor = tag.color as? UIColor

}
```

Fetching an Object

```
let fetchRequest : NSFetchRequest<Tag> = Tag.fetchRequest()

fetchRequest.predicate = NSPredicate(format: "name = %@", tagName)

if let tag = try? fetchRequest.execute().first {

    tagLabel.text = tag.name
    tagLabel.textColor = tag.color as? UIColor

}
```

Fetching an Object

```
let fetchRequest : NSFetchRequest<Tag> = Tag.fetchRequest()

fetchRequest.predicate = NSPredicate(format: "name = %@", tagName)

if let tag = try? fetchRequest.execute().first {

    tagLabel.text = tag.name
    tagLabel.textColor = tag.color as? UIColor

}
```


Wiring Views

```
let fetchRequest : NSFetchRequest<Tag> = Tag.fetchRequest()

fetchRequest.predicate = NSPredicate(format: "name = %@", tagName)

if let tag = try? fetchRequest.execute().first {

    nameSubscription = tag.publisher(for: \.name)
        .assign(to: \.text, on: tagLabel)

    colorSubscription = tag.publisher(for: \.color)
        .map({ $0 as? UIColor })
        .assign(to: \.textColor, on: tagLabel)

}
```

Wiring Views

```
let fetchRequest : NSFetchRequest<Tag> = Tag.fetchRequest()

fetchRequest.predicate = NSPredicate(format: "name = %@", tagName)

if let tag = try? fetchRequest.execute().first {

    nameSubscription = tag.publisher(for: \.name)
        .assign(to: \.text, on: tagLabel)

    colorSubscription = tag.publisher(for: \.color)
        .map({ $0 as? UIColor })
        .assign(to: \.textColor, on: tagLabel)

}
```

Wiring Detail Views

```
if let tag = tag {  
  
    nameSubscription = tag.publisher(for: \.name)  
        .assign(to: \.text, on: tagLabel)  
  
    colorSubscription = tag.publisher(for: \.color)  
        .map({ $0 as? UIColor })  
        .assign(to: \.textColor, on: tagLabel)  
  
}
```

Fetching Many Objects

Sort results

```
fetchRequest.sortDescriptors = [NSSortDescriptor(key: "name", ascending: true)]
```


Fetching Many Objects

Sort results

```
fetchRequest.sortDescriptors = [NSSortDescriptor(key: "name", ascending: true)]
```

Batched fetching

```
fetchRequest.fetchBatchSize = 50
```

Live Queries

Live Queries

```
let fetchRequest: NSFetchedRequest<Post> = Post.fetchRequest()  
  
fetchRequest.sortDescriptors = [NSSortDescriptor(key: "title", ascending: true)]  
fetchRequest.fetchBatchSize = 50
```

Live Queries

```
let fetchRequest: NSFetchRequest<Post> = Post.fetchRequest()

fetchRequest.sortDescriptors = [NSSortDescriptor(key: "title", ascending: true)]
fetchRequest.fetchBatchSize = 50

let controller = NSFetchedResultsController(fetchRequest: fetchRequest,
                                           managedObjectContext: moc,
                                           sectionNameKeyPath: nil, cacheName: nil)

controller.delegate = self

try! controller.performFetch()
```


Fetches Results Controller Delegate Methods

```
controllerWillChangeContent(:)
```

```
controller(:didChange:atSectionIndex:for:)
```

```
controller(:didChange:at:for:newIndexPath:)
```

```
controllerDidChangeContent(:)
```

Fetches Results Controller Delegate Methods

```
controllerWillChangeContent(:)
```

```
controller(:didChange:atSectionIndex:for:)
```

```
controller(:didChange:at:for:newIndexPath:)
```

```
controllerDidChangeContent(:)
```

Fetches Results Controller Delegate Methods

```
controllerWillChangeContent(:)
```

```
controller(:didChange:atSectionIndex:for:)
```

```
controller(:didChange:at:for:newIndexPath:)
```

```
controllerDidChangeContent(:)
```

Fetches Results Controller Delegate Methods

```
controllerWillChangeContent(:)
```

```
controller(:didChange:atSectionIndex:for:)
```

```
controller(:didChange:at:for:newIndexPath:)
```

```
controllerDidChangeContent(:)
```


Fetches Results Controller Delegate Methods

```
controllerWillChangeContent(:)
```

```
controller(:didChange:atSectionIndex:for:)
```

```
controller(:didChange:at:for:newIndexPath:)
```

```
controllerDidChangeContent(:)
```

Fetches Results Controller Delegate Methods

```
controllerWillChangeContent(:)
```

```
controller(:didChange:atSectionIndex:for:)
```

```
controller(:didChange:at:for:newIndexPath:)
```

```
controllerDidChangeContent(:)
```

Fetches Results Controller Delegate Methods

```
controllerWillChangeContent(:)
```

```
controller(:didChange:atSectionIndex:for:)
```

```
controller(:didChange:at:for:newIndexPath:)
```

```
controllerDidChangeContent(:)
```

Fetches Results Controller Delegate Methods

```
controllerWillChangeContent(:)
```

```
controller(:didChange:atSectionIndex:for:)
```

```
controller(:didChange:at:for:newIndexPath:)
```

```
controllerDidChangeContent(:)
```

Fetches Results Controller Delegate Methods

```
controllerWillChangeContent(:)
```

```
controller(:didChange:atSectionIndex:for:)
```

```
controller(:didChange:at:for:newIndexPath:)
```

```
controllerDidChangeContent(:)
```


Fetches Results Controller Delegate Methods

`controllerWillChangeContent(:)`



`beginUpdates()`

`controller(:didChange:atSectionIndex:for:)`



`insertSections(:with:)`
`deleteSections(:with:)`

`controller(:didChange:at:for:newIndexPath:)`



`insertRows(at:with:)`
`deleteRows(at:with:)`

`controllerDidChangeContent(:)`



`endUpdates()`

Fetches Results Controller Delegate Methods

`controllerWillChangeContent(:)`

`beginUpdates()`

`controller(:didChange:atSectionIndex:for:)`

`insertSections(:with:)`

`deleteSections(:with:)`

`controller(:didChange:at:for:newIndexPath:)`

`insertRows(at:with:)`

`deleteRows(at:with:)`

`controllerDidChangeContent(:)`

`endUpdates()`

Displaying Fetched Results Using Snapshots

New delegate method vends instances of `NSDiffableDataSourceSnapshot`

Snapshots encode the section and row state of a collection view

Displaying Fetched Results Using Snapshots



NEW

New delegate method vends instances of `NSDiffableDataSourceSnapshot`

Snapshots encode the section and row state of a collection view

Displaying Fetched Results Using Snapshots

NEW

New delegate method vends instances of `NSDiffableDataSourceSnapshot`

Snapshots encode the section and row state of a collection view

```
func controller(
    _ controller: NSFetchedResultsController<NSFetchRequestResult>,
    didChangeContentWith snapshot: NSDiffableDataSourceSnapshotReference<NSManagedObjectID,
    NSString>
) {
    collectionViewDataSource.applySnapshot(snapshot as! NSDiffableDataSourceSnapshot)
}
```


Displaying Fetched Results Using Differences

New delegate method vends instances of `CollectionDifference`

Only supported when `sectionNameKeyPath` is `nil`

Great for driving individual sections of a complex view!

Displaying Fetched Results Using Differences



NEW

New delegate method vends instances of `CollectionDifference`

Only supported when `sectionNameKeyPath` is nil

Great for driving individual sections of a complex view!

Displaying Fetched Results Using Differences

NEW

New delegate method vends instances of `CollectionDifference`

Only supported when `sectionNameKeyPath` is `nil`

Great for driving individual sections of a complex view!

Ordered Collection Diffing (SE-0240)

Introducing Combine and Advances in Foundation

WWDC 2019

```
func controller(
    _ controller: NSFetchedResultsController<NSFetchRequestResult>,
    didChangeContentWith diff: CollectionDifference<NSManagedObjectID>
) {
    collectionView.performBatchUpdates({
        for change in diff {
            switch change {
            case .insert(offset: let newRow, element: _, associatedWith: let assoc):
                if let oldRow = assoc {
                    collectionView.moveItem(
                        at: IndexPath(row: oldRow, section: frcSection),
                        to: IndexPath(row: newRow, section: frcSection))
                } else {
                    collectionView.insertItems(
                        at: [IndexPath(row: newRow, section: frcSection)])
                }
            case .remove(offset: let oldRow, element: _, associatedWith: let assoc):
                if assoc == nil {
                    collectionView.deleteItems(
                        at: [IndexPath(row: oldRow, section: frcSection)])
                }
            }
        }
    }, completion: nil)
}
```



```
func controller(
    _ controller: NSFetchedResultsController<NSFetchRequestResult>,
    didChangeContentWith diff: CollectionDifference<NSManagedObjectID>
) {
    collectionView.performBatchUpdates({
        for change in diff {
            switch change {
            case .insert(offset: let newRow, element: _, associatedWith: let assoc):
                if let oldRow = assoc {
                    collectionView.moveItem(
                        at: IndexPath(row: oldRow, section: frcSection),
                        to: IndexPath(row: newRow, section: frcSection))
                } else {
                    collectionView.insertItems(
                        at: [IndexPath(row: newRow, section: frcSection)])
                }
            case .remove(offset: let oldRow, element: _, associatedWith: let assoc):
                if assoc == nil {
                    collectionView.deleteItems(
                        at: [IndexPath(row: oldRow, section: frcSection)])
                }
            }
        }
    }, completion: nil)
}
```



```
func controller(
    _ controller: NSFetchedResultsController<NSFetchRequestResult>,
    didChangeContentWith diff: CollectionDifference<NSManagedObjectID>
) {
    collectionView.performBatchUpdates({
        for change in diff {
            switch change {
                case .insert(offset: let newRow, element: _, associatedWith: let assoc):
                    if let oldRow = assoc {
                        collectionView.moveItem(
                            at: IndexPath(row: oldRow, section: frcSection),
                            to: IndexPath(row: newRow, section: frcSection))
                    } else {
                        collectionView.insertItems(
                            at: [IndexPath(row: newRow, section: frcSection)])
                    }
                case .remove(offset: let oldRow, element: _, associatedWith: let assoc):
                    if assoc == nil {
                        collectionView.deleteItems(
                            at: [IndexPath(row: oldRow, section: frcSection)])
                    }
            }
        }
    }, completion: nil)
}
```

```
func controller(
    _ controller: NSFetchedResultsController<NSFetchRequestResult>,
    didChangeContentWith diff: CollectionDifference<NSManagedObjectID>
) {
    collectionView.performBatchUpdates({
        for change in diff {
            switch change {
            case .insert(offset: let newRow, element: _, associatedWith: let assoc):
                if let oldRow = assoc {
                    collectionView.moveItem(
                        at: IndexPath(row: oldRow, section: frcSection),
                        to: IndexPath(row: newRow, section: frcSection))
                } else {
                    collectionView.insertItems(
                        at: [IndexPath(row: newRow, section: frcSection)])
                }
            case .remove(offset: let oldRow, element: _, associatedWith: let assoc):
                if assoc == nil {
                    collectionView.deleteItems(
                        at: [IndexPath(row: oldRow, section: frcSection)])
                }
            }
        }
    }, completion: nil)
}
```

```
func controller(
    _ controller: NSFetchedResultsController<NSFetchRequestResult>,
    didChangeContentWith diff: CollectionDifference<NSManagedObjectID>
) {
    collectionView.performBatchUpdates({
        for change in diff {
            switch change {
            case .insert(offset: let newRow, element: _, associatedWith: let assoc):
                if let oldRow = assoc {
                    collectionView.moveItem(
                        at: IndexPath(row: oldRow, section: frcSection),
                        to: IndexPath(row: newRow, section: frcSection))
                } else {
                    collectionView.insertItems(
                        at: [IndexPath(row: newRow, section: frcSection)])
                }
            case .remove(offset: let oldRow, element: _, associatedWith: let assoc):
                if assoc == nil {
                    collectionView.deleteItems(
                        at: [IndexPath(row: oldRow, section: frcSection)])
                }
            }
        }
    }, completion: nil)
}
```



```
func controller(
    _ controller: NSFetchedResultsController<NSFetchRequestResult>,
    didChangeContentWith diff: CollectionDifference<NSManagedObjectID>
) {
    collectionView.performBatchUpdates({
        for change in diff {
            switch change {
            case .insert(offset: let newRow, element: _, associatedWith: let assoc):
                if let oldRow = assoc {
                    collectionView.moveItem(
                        at: IndexPath(row: oldRow, section: frcSection),
                        to: IndexPath(row: newRow, section: frcSection))
                } else {
                    collectionView.insertItems(
                        at: [IndexPath(row: newRow, section: frcSection)])
                }
            case .remove(offset: let oldRow, element: _, associatedWith: let assoc):
                if assoc == nil {
                    collectionView.deleteItems(
                        at: [IndexPath(row: oldRow, section: frcSection)])
                }
            }
        }
    }, completion: nil)
}
```

```
func controller(
    _ controller: NSFetchedResultsController<NSFetchRequestResult>,
    didChangeContentWith diff: CollectionDifference<NSManagedObjectID>
) {
    collectionView.performBatchUpdates({
        for change in diff {
            switch change {
            case .insert(offset: let newRow, element: _, associatedWith: let assoc):
                if let oldRow = assoc {
                    collectionView.moveItem(
                        at: IndexPath(row: oldRow, section: frcSection),
                        to: IndexPath(row: newRow, section: frcSection))
                } else {
                    collectionView.insertItems(
                        at: [IndexPath(row: newRow, section: frcSection)])
                }
            case .remove(offset: let oldRow, element: _, associatedWith: let assoc):
                if assoc == nil {
                    collectionView.deleteItems(
                        at: [IndexPath(row: oldRow, section: frcSection)])
                }
            }
        }
    }, completion: nil)
}
```



```
func controller(
    _ controller: NSFetchedResultsController<NSFetchRequestResult>,
    didChangeContentWith diff: CollectionDifference<NSManagedObjectID>
) {
    collectionView.performBatchUpdates({
        for change in diff {
            switch change {
            case .insert(offset: let newRow, element: _, associatedWith: let assoc):
                if let oldRow = assoc {
                    collectionView.moveItem(
                        at: IndexPath(row: oldRow, section: frcSection),
                        to: IndexPath(row: newRow, section: frcSection))
                } else {
                    collectionView.insertItems(
                        at: [IndexPath(row: newRow, section: frcSection)])
                }
            case .remove(offset: let oldRow, element: _, associatedWith: let assoc):
                if assoc == nil {
                    collectionView.deleteItems(
                        at: [IndexPath(row: oldRow, section: frcSection)])
                }
            }
        }
    }, completion: nil)
}
```

```
func controller(
    _ controller: NSFetchedResultsController<NSFetchRequestResult>,
    didChangeContentWith diff: CollectionDifference<NSManagedObjectID>
) {
    collectionView.performBatchUpdates({
        for change in diff {
            switch change {
            case .insert(offset: let newRow, element: _, associatedWith: let assoc):
                if let oldRow = assoc {
                    collectionView.moveItem(
                        at: IndexPath(row: oldRow, section: frcSection),
                        to: IndexPath(row: newRow, section: frcSection))
                } else {
                    collectionView.insertItems(
                        at: [IndexPath(row: newRow, section: frcSection)])
                }
            case .remove(offset: let oldRow, element: _, associatedWith: let assoc):
                if assoc == nil {
                    collectionView.deleteItems(
                        at: [IndexPath(row: oldRow, section: frcSection)])
                }
            }
        }
    }, completion: nil)
}
```

```
func controller(
    _ controller: NSFetchedResultsController<NSFetchRequestResult>,
    didChangeContentWith diff: CollectionDifference<NSManagedObjectID>
) {
    collectionView.performBatchUpdates({
        for change in diff {
            switch change {
            case .insert(offset: let newRow, element: _, associatedWith: let assoc):
                if let oldRow = assoc {
                    collectionView.moveItem(
                        at: IndexPath(row: oldRow, section: frcSection),
                        to: IndexPath(row: newRow, section: frcSection))
                } else {
                    collectionView.insertItems(
                        at: [IndexPath(row: newRow, section: frcSection)])
                }
            case .remove(offset: let oldRow, element: _, associatedWith: let assoc):
                if assoc == nil {
                    collectionView.deleteItems(
                        at: [IndexPath(row: oldRow, section: frcSection)])
                }
            }
        }
    }, completion: nil)
}
```


Displaying Fetched Results

Drive declarative list views from `controllerDidChangeContent(:)`

Denormalization

Improving Performance Using Denormalization

Adding redundant data to make access faster and more convenient

Redundant data requires more work

Tag

∨ Attributes

color

name

uuid

∨ Relationships

posts

Tag

∨ Attributes

color

name

postCount

uuid

∨ Relationships

posts

Derived Attributes

CoreData-managed metadata



Derived Attributes

CoreData-managed metadata



NEW

Derived Attributes

CoreData-managed metadata



Multiple supported functions

Derived Attributes

CoreData-managed metadata



NEW

Multiple supported functions

Defined in managed object model

Derived Attributes

CoreData-managed metadata



NEW

Multiple supported functions

Defined in managed object model

Available on all properties of the entity

Demo

Supported Derivations

Supported Derivations

Data duplication

Supported Derivations

Data duplication

Data transformation

Supported Derivations

Data duplication

Data transformation

To-many aggregate functions

Supported Derivations

Data duplication

Data transformation

To-many aggregate functions

Zero-parameter functions

Getting started

The needs of the Controller

Scaling your app

Testing

Fetching Persistent History

Fetching Persistent History

NEW

Fetching Persistent History



NEW

Process history generated by a specific author

Fetching Persistent History



NEW

Process history generated by a specific author

Fetch history affecting a specific type

Fetching Persistent History



NEW

Process history generated by a specific author

Fetch history affecting a specific type

Changes that happened between a two tokens

NSPersistentHistoryTransaction and NSPersistentHistoryChange



NEW

```
class func entityDescription(
    withContext context: NSManagedObjectContext
) -> NSEntityDescription?

class var entityDescription: NSEntityDescription? { get }

class var fetchRequest: NSFetchedRequest? { get }
```

NSPersistentHistoryTransaction and NSPersistentHistoryChange



NEW

```
class func entityDescription(
    withContext context: NSManagedObjectContext
) -> NSEntityDescription?

class var entityDescription: NSEntityDescription? { get }

class var fetchRequest: NSFetchedRequest? { get }
```

NSPersistentHistoryTransaction and NSPersistentHistoryChange



NEW

```
class func entityDescription(
    withContext context: NSManagedObjectContext
) -> NSEntityDescription?

class var entityDescription: NSEntityDescription? { get }

class var fetchRequest: NSFetchedRequest? { get }
```


NSPersistentHistoryTransaction and NSPersistentHistoryChange



NEW

```
class func entityDescription(
    withContext context: NSManagedObjectContext
) -> NSEntityDescription?

class var entityDescription: NSEntityDescription? { get }

class var fetchRequest: NSFetchedRequest? { get }
```

Fetching Persistent History

NEW

Fetching Persistent History



NEW

```
open class NSPersistentHistoryChangeRequest : NSPersistentStoreRequest {  
  
    open class func fetchHistory(  
        withFetch fetchRequest: NSFetchedRequest<NSFetchRequestResult>  
    ) -> Self  
  
    open var fetchRequest: NSFetchedRequest<NSFetchRequestResult>?  
  
}
```

Fetching Persistent History

NEW

```
open class NSPersistentHistoryChangeRequest : NSPersistentStoreRequest {  
  
    open class func fetchHistory(  
        withFetch fetchRequest: NSFetchedRequest<NSFetchRequestResult>  
    ) -> Self  
  
    open var fetchRequest: NSFetchedRequest<NSFetchRequestResult>?  
  
}
```

Fetching Persistent History

NEW

```
open class NSPersistentHistoryChangeRequest : NSPersistentStoreRequest {  
  
    open class func fetchHistory(  
        withFetch fetchRequest: NSFetchedRequest<NSFetchRequestResult>  
    ) -> Self  
  
    open var fetchRequest: NSFetchedRequest<NSFetchRequestResult>?  
  
}
```


Discovering New History

Discovering New History

Poll the store?

Dispatch source?

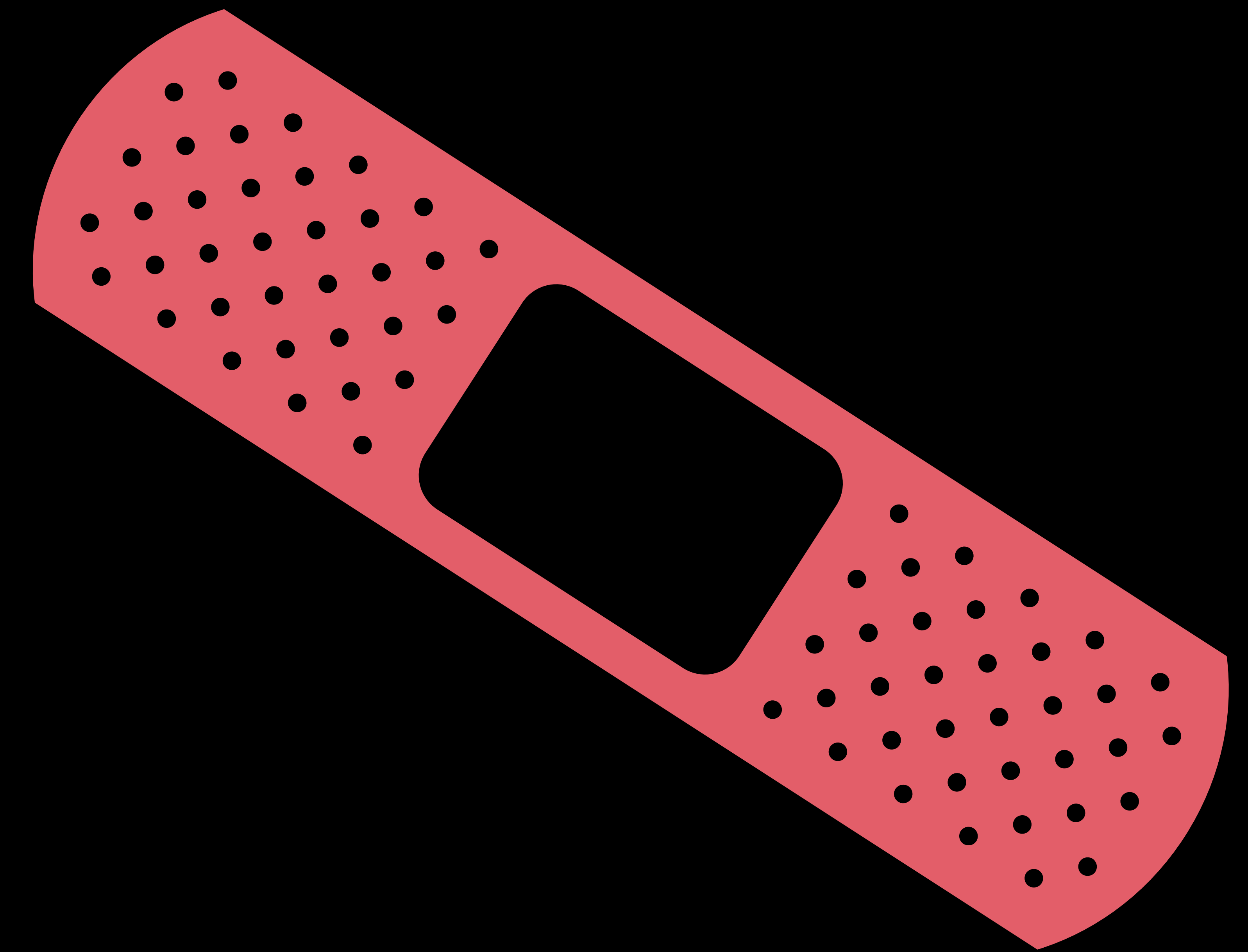
FS Events?

Discovering New History

Poll the store?

Dispatch source?

FS Events?



Remote Change Notifications

Cross-coordinator save notifications

Asynchronous

Remote Change Notifications



NEW

Cross-coordinator save notifications

Asynchronous

Remote Change Notifications

NEW

Cross-coordinator *change* notifications

Remote Change Notifications

NEW

Remote Change Notifications



NEW

```
let description: NSPersistentStoreDescription = /* ... */

description.setOption(
    true as NSNumber,
    forKey: NSPersistentStoreRemoteChangeNotificationPostOptionKey)

description.setOption(true as NSNumber, forKey: NSPersistentHistoryTrackingKey)
```

Remote Change Notifications

NEW

Remote Change Notifications

NEW

```
func storeRemoteChange(_ notification: Notification) {  
  
    precondition(notification.name == NSNotification.Name.NSPersistentStoreRemoteChange)  
  
    let storeURL = notification.userInfo?[NSPersistentStoreURLKey]!  
    let token = notification.userInfo?[NSPersistentHistoryTokenKey]!  
  
    print("Store at \(storeURL) was changed in transaction \(token).")  
  
}
```


Persistent History Tokens

Persistent History Tokens

NEW

```
extension NSPersistentStoreCoordinator {  
  
    func currentPersistentHistoryToken(  
        from stores: [NSPersistentStore]?  
    ) -> NSPersistentHistoryToken?  
  
}
```

Demo

Getting started

The needs of the controller

Scaling your app

Testing

Testing Against Core Data

Test against actual performance goals

Testing Against Core Data

Test against actual performance goals

Run integration tests in multiple configs

 **Build**
1 target

 **Run**
Debug

 **Test**
Debug

 **Profile**
Release

 **Analyze**
Debug

 **Archive**
Release

Info

Arguments

Options

Diagnostics

▼ **Arguments Passed On Launch**

-com.apple.CoreData.ConcurrencyDebug 1

+ -


▼ **Environment Variables**

Name	Value
------	-------

No Environment Variables

+ -

Expand Variables Based On






 CoreDataCloudKitDemo

Duplicate Scheme

Manage Schemes...

Shared

Close

- ▶  **Build**
1 target
- ▶ ▶ **Run**
Debug
- ▶  **Test**
Debug
- ▶  **Profile**
Release
- ▶  **Analyze**
Debug
- ▶  **Archive**
Release

Info Arguments Options Diagnostics

▼ Arguments Passed On Launch

-com.apple.CoreData.ConcurrencyDebug 1


+ -

▼ Environment Variables

Name	Value
------	-------

No Environment Variables

+ -

Expand Variables Based On  CoreDataCloudKitDemo

Duplicate Scheme

Manage Schemes...

Shared

Close

Testing Against Core Data

Test against actual performance goals

Run integration tests in multiple configs

Testing Against Core Data

Test against actual performance goals

Run integration tests in multiple configs

Use in-memory stores where appropriate

Named In-Memory Stores

```
let container = NSPersistentCloudKitContainer(name: "CoreDataCloudKitDemo")

let description = container.persistentStoreDescriptions.first!

description.url = URL(fileURLWithPath: "/dev/null")

container.loadPersistentStores(completionHandler: { (_, error) in
    guard let error = error as NSError? else { return }
    fatalError("###\(#function): Failed to load persistent stores:\(error)")
})
```

Named In-Memory Stores

```
let container = NSPersistentCloudKitContainer(name: "CoreDataCloudKitDemo")

let description = container.persistentStoreDescriptions.first!

description.url = URL(fileURLWithPath: "/dev/null")

container.loadPersistentStores(completionHandler: { (_, error) in
    guard let error = error as NSError? else { return }
    fatalError("###\(#function): Failed to load persistent stores:\(error)")
})
```

Named In-Memory Stores

```
let container = NSPersistentCloudKitContainer(name: "CoreDataCloudKitDemo")

let description = container.persistentStoreDescriptions.first!

description.url = URL(fileURLWithPath: "/dev/null").appendingPathComponent(str)

container.loadPersistentStores(completionHandler: { (_, error) in
    guard let error = error as NSError? else { return }
    fatalError("###\(#function): Failed to load persistent stores:\(error)")
})
```

Sanitizers

Sanitizers

Address Sanitizer

Sanitizers

Address Sanitizer

Thread Sanitizer

Sanitizers

Address Sanitizer

Thread Sanitizer

Undefined Behavior Sanitizer

Getting started

The needs of the controller

Scaling your app

Testing

Feedback Assistant

More Information

developer.apple.com/wwdc19/230

 WWDC19