CoreData

What is CoreData?

- Handles persistance for complex object graphs.
- Backed by ObjC runtime.
- De facto ORM for iOS and macOS

Contrary to popular belief, Core Data is not an Object-Relational Mapper, but rather an object graph and persistence framework, capable of [...]. Using Core Data as an ORM necessarily limits the capabilities of Core Data and muddies its conceptual purity.

NSHipster.com

Backend

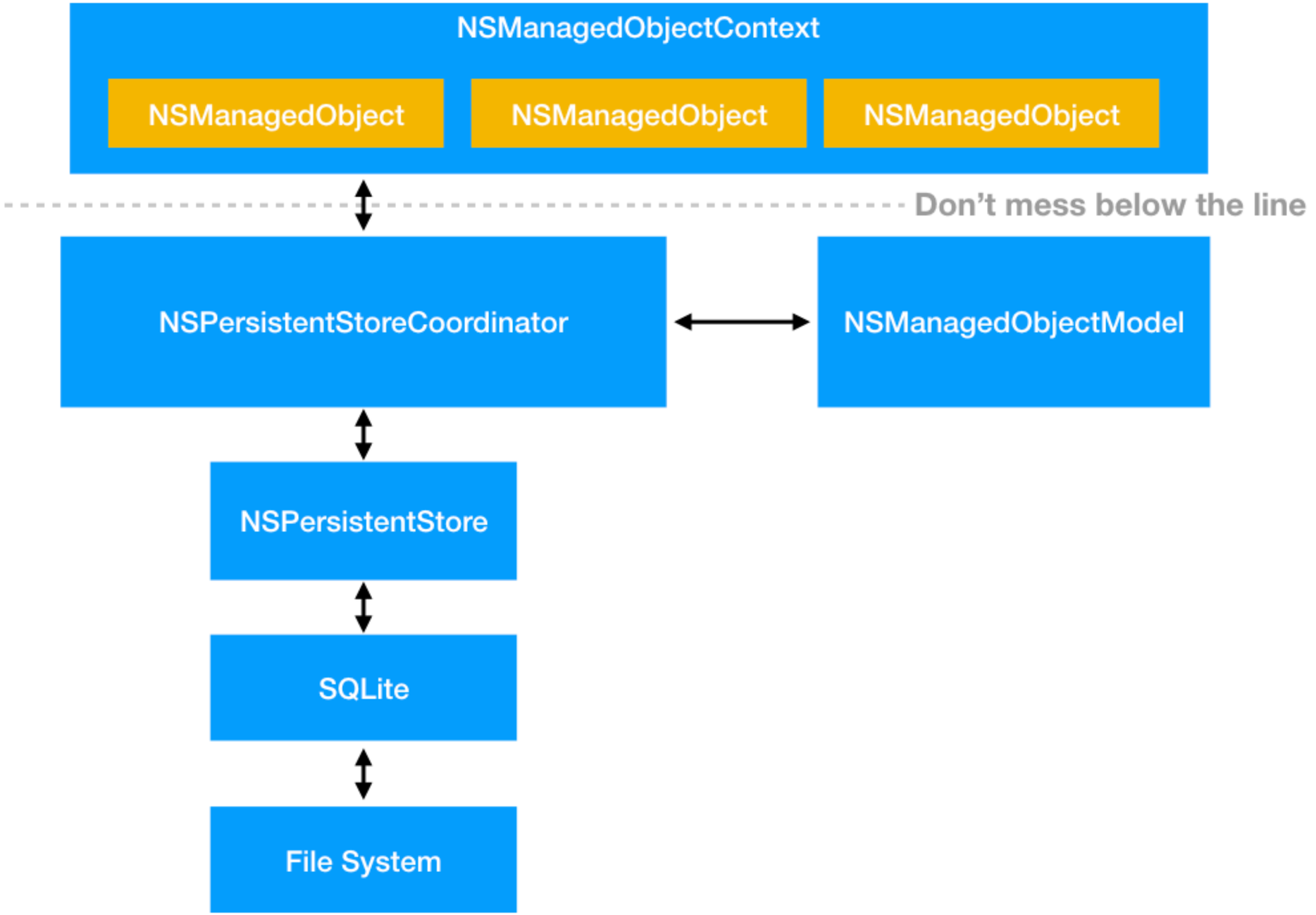
- NSBinaryStoreType
- NSInMemoryStoreType
- NSSQLiteStoreType

Features

- Object uniquing
- Lazy load or *faulting*
- Migrations from database schema
  - Mostly automatic
- `NSPredicate`
- Change tracking via KVO
- Undo/Redo

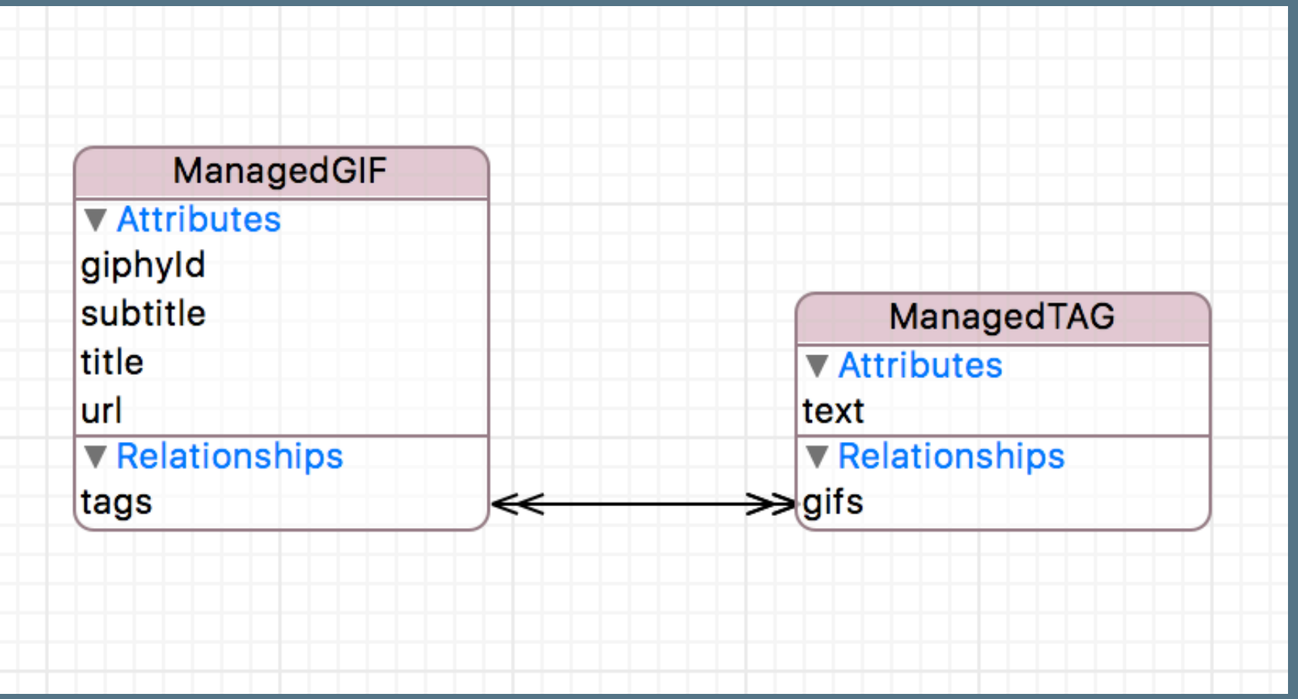
Stack

- This is what you need to create and hook to use the database.
- It's fairly complex
  - Because of it's flexibility.
  - Optimal stack setup depends on the use case.


NSManagedObjectContext
NSManagedObject
NSManagedObject
NSManagedObject
Don’t mess below the line
NSPersistentStoreCoordinator
NSManagedObjectModel
NSPersistentStore
SQLite
File System

NSManagedObjectModel

- Describes the relationships between your data model objects.
- Entitites are represented by `NSManagedObjects`
- Relationship between entities:
  - 1:1
  - 1:N
  - N:N



Model.xcdatamodeld

🛠

Model.momd

NSManagedObject

- Represents the basic data entities.
  - `GIF`
- Is not used directly, but rather subclassed.
  - `ManagedGIF`
- Backed by ObjC runtime
  - `@NSManaged` attributes
- Can be *faulted* if neccesary

NSManagedObject

- Contains properties, relationships and helper methods to modify these.
  - Only modify relationships via the *accessors* ⚠️
- Relationships are unordered by default.

```swift
@objc(ManagedGIF)
public class ManagedGIF: NSManagedObject {}

extension ManagedGIF {

    @NSManaged public var url: String?
    @NSManaged public var giphyId: String?
    @NSManaged public var title: String?
    @NSManaged public var subtitle: String?
    @NSManaged public var tags: NSSet?

}
```

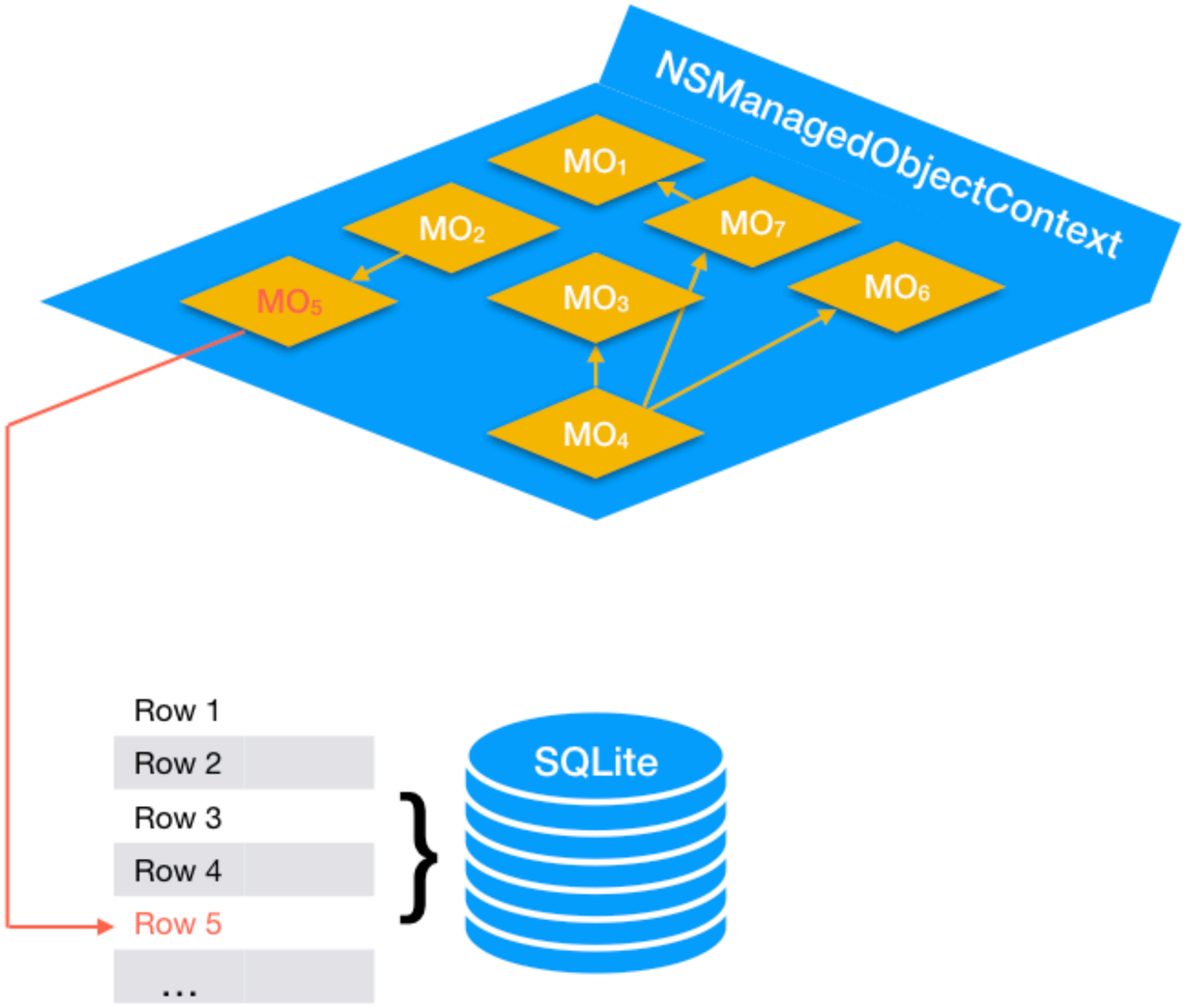
```swift
// MARK: Generated accessors for tags
extension ManagedGIF {

    @objc(addTagsObject:)
    @NSManaged public func addToTags(_ value: ManagedTAG)

    @objc(removeTagsObject:)
    @NSManaged public func removeFromTags(_ value: ManagedTAG)

    @objc(addTags:)
    @NSManaged public func addToTags(_ values: NSSet)

    @objc(removeTags:)
    @NSManaged public func removeFromTags(_ values: NSSet)

}
```

NSManagedObjectContext

- Stores a object graph from the store
- Only contains *some* objects from the store, either:
  - Fully fetched.
  - Faulted.
- Not thread-safe.
  - One context per-thread

NSManagedObjectContext

- Using it you'll do operations on objects:
  - Creation
  - Erase
  - Saving
  - Search
  - Undo/Redo


NSManagedObjectContext
MO1
MO2
MO7
MO5
MO3
MO6
MO4
Row 1
Row 2
Row 3
Row 4
Row 5
...
SQLite

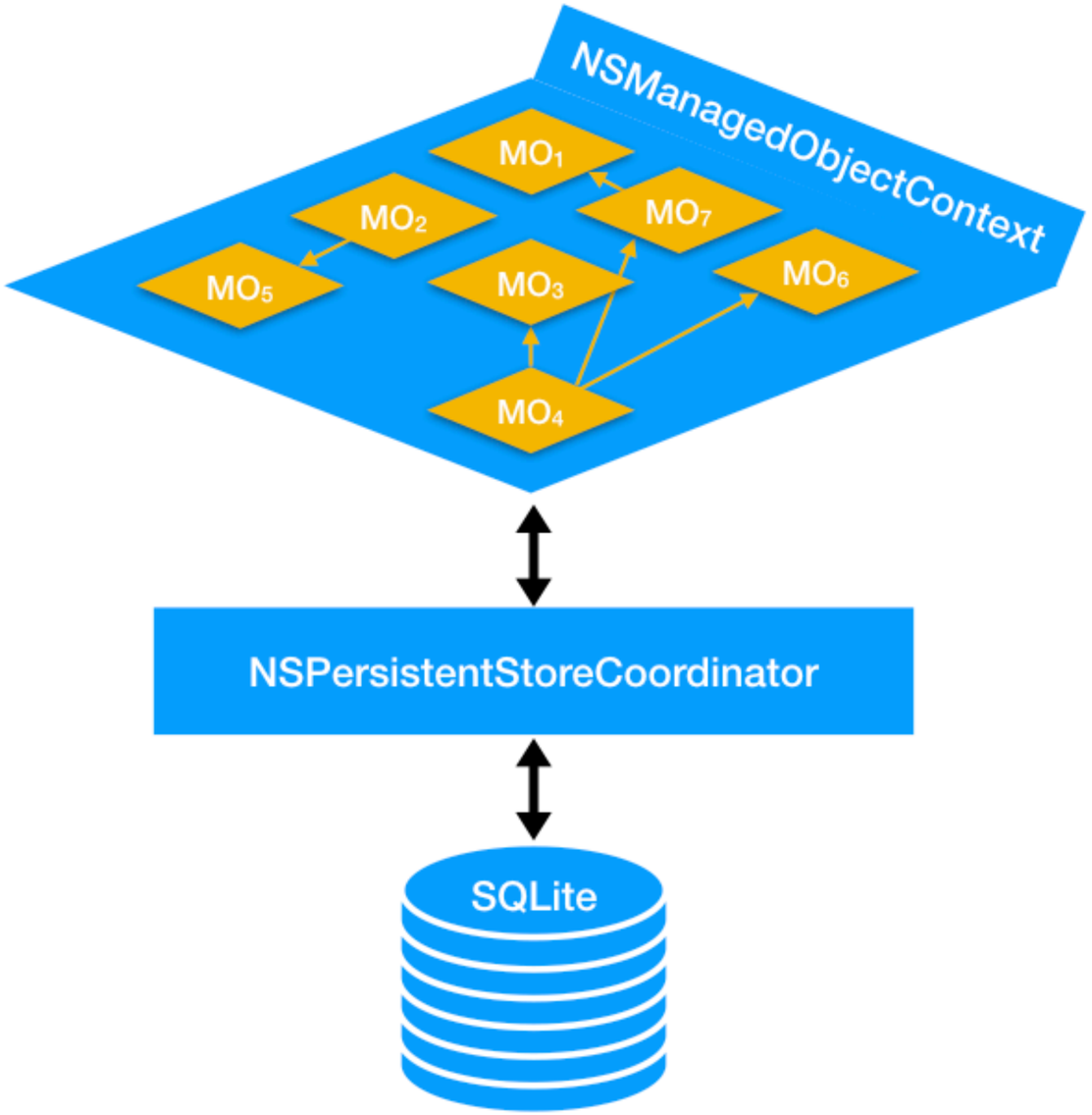
Stack kinds:

- Unique context
- Nested contexts
- Sibling contexts
- Hybrid contexts

Unique contexts

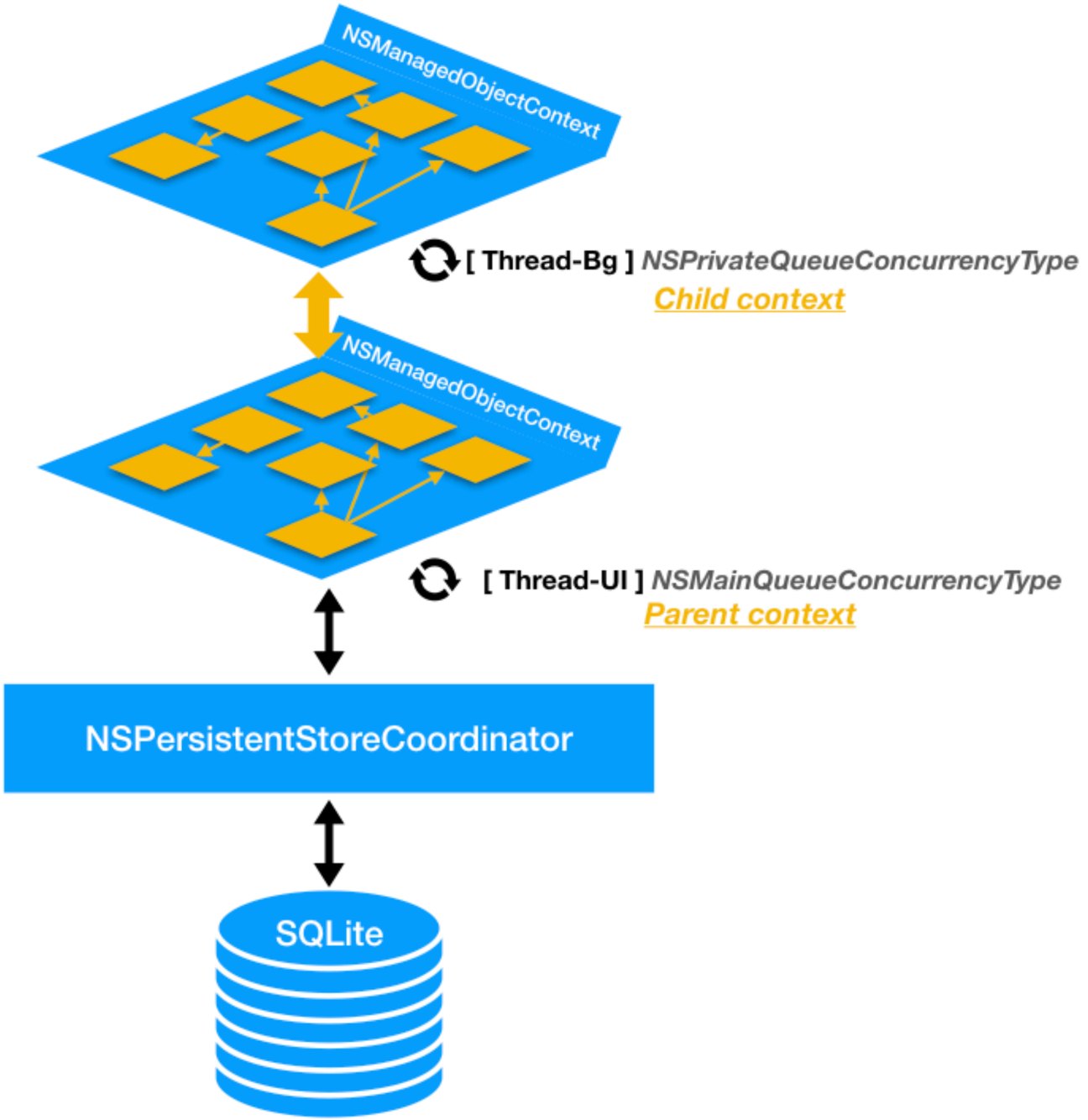
- Simple but low performance
  - One context to read write
  - It'll block the main thread

[Thread-UI] *NSMainQueueConcurrencyType*



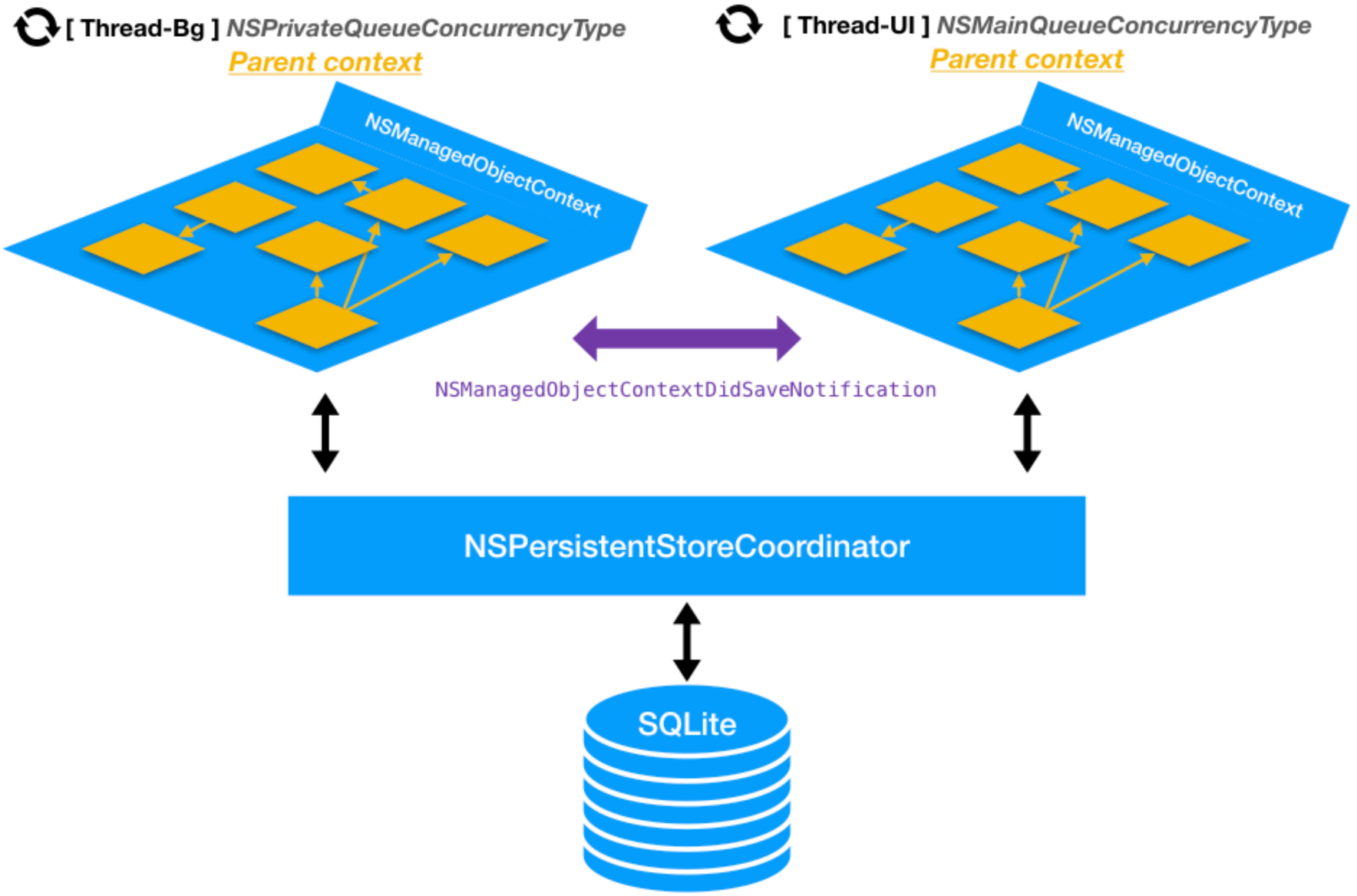
Nested contexts

- Compromise between simplicity and performance.
- Imports are performed in a background thread
- Reads are perfomed in main thread
  - Automatic sync


NSManagedObjectContext
[Thread-Bg] NSPrivateQueueConcurrencyType
Child context
NSManagedObjectContext
[Thread-UI] NSMainQueueConcurrencyType
Parent context
NSPersistentStoreCoordinator
SQLite

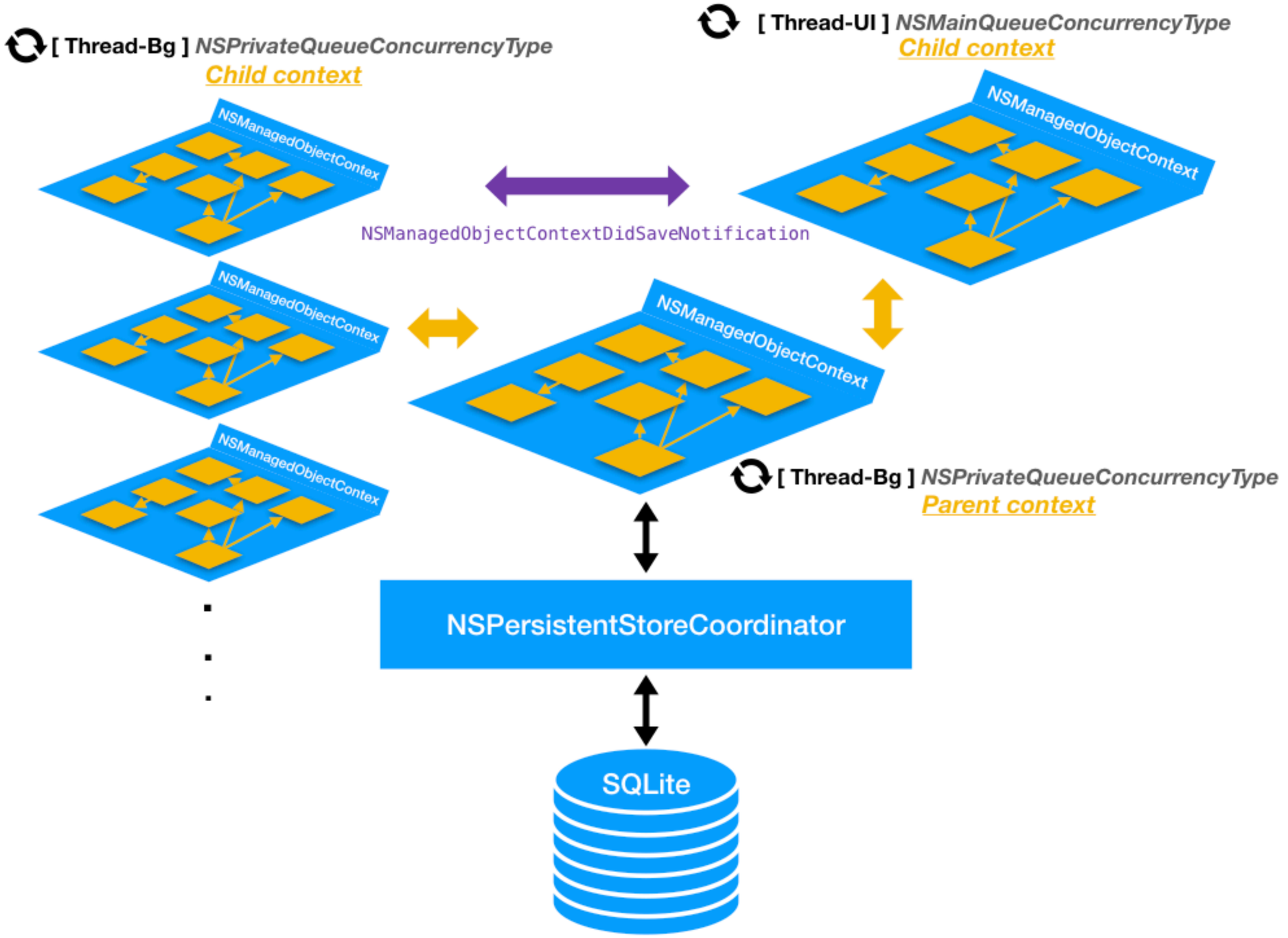
Sibling contexts

- Improved performance.
- Contexts are indipendent from each other.
  - Thread-wise they are also indipendent.
- Requires manual synchronization via the `NSManagedObjectContextDidSaveNotification`.
  - This will block the main context.


[Thread-Bg] NSPrivateQueueConcurrencyType
Parent context
NSManagedObjectContext
[Thread-UI] NSMainQueueConcurrencyType
Parent context
NSManagedObjectContext
NSManagedObjectContextDidSaveNotification
NSPersistentStoreCoordinator
SQLite

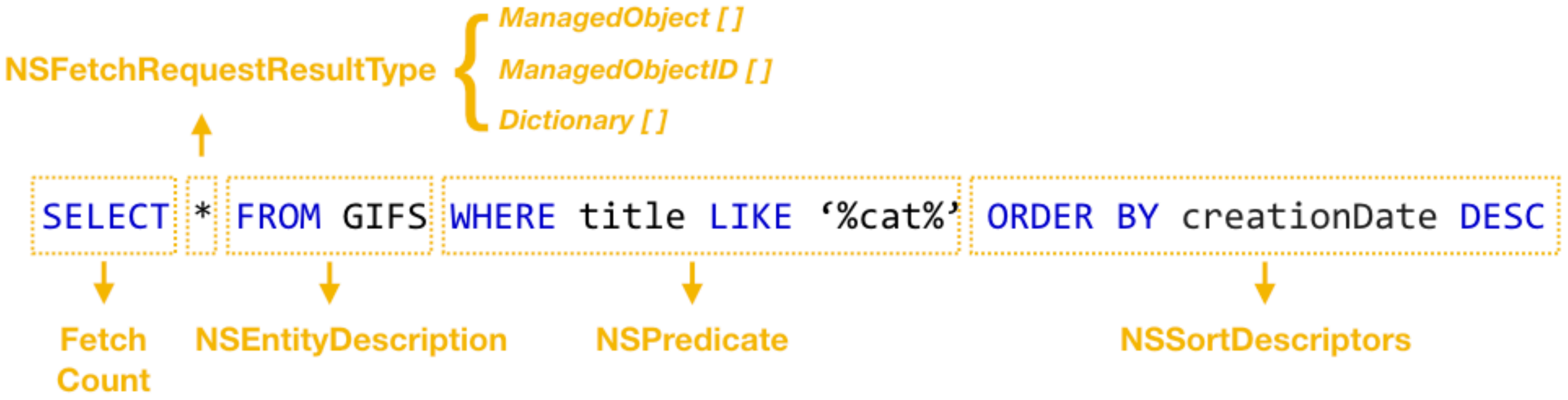
Hybrid Stack

- Adds a parent context for saving in a background thread.
- Then, a child context will be bound to the main thread.
- Saving is performed using transient contexts, also child of the main one.
- ⚠️ It's the most complex one, since race conditions may occur.
  - GCD queues to guard against these are recommended


[Thread-Bg] NSPrivateQueueConcurrencyType
Child context
[Thread-UI] NSMainQueueConcurrencyType
Child context
NSManagedObjectContext
NSManagedObjectContext
NSManagedObjectContextDidSaveNotification
NSManagedObjectContext
NSManagedObjectContext
[Thread-Bg] NSPrivateQueueConcurrencyType
Parent context
NSPersistentStoreCoordinator
SQLite

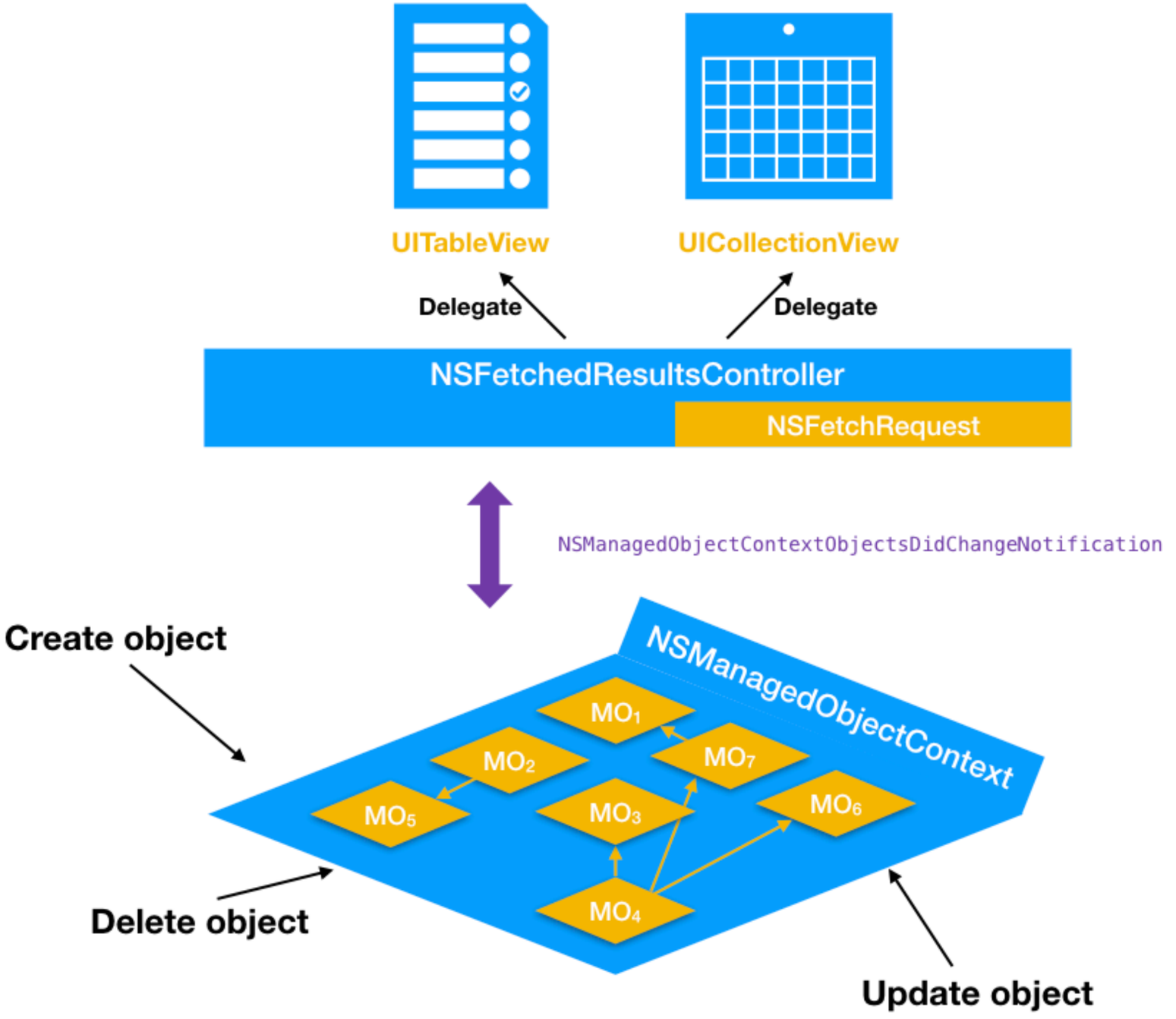
NSFetchRequest

- The equivalent of an SQL statement.
- Searches, creates, filters and limits the objects within a CoreData context.
- High computational cost, may be async.


NSFetchRequestResultType
ManagedObject [ ]
ManagedObjectID [ ]
Dictionary [ ]
SELECT * FROM GIFS WHERE title LIKE '%cat%' ORDER BY creationDate DESC
Fetch Count
NSEntityDescription
NSPredicate
NSSortDescriptors

NSFetchedResultsController

- Observes the changes within an `NSFetchRequest` in a *reactive* way.
- Conceived for integrating with a `UITableView`/`UICollectionView` through a delegate
- Listens for changes in objects within a context with `NSManagedObjectContextObjectsDidChangeNotification`.


UITableView
UICollectionView
Delegate
Delegate
NSFetchedResultsController
NSFetchRequest
NSManagedObjectContextObjectsDidChangeNotification
Create object
NSManagedObjectContext
MO1
MO2
MO7
MO5
MO3
MO6
MO4
Delete object
Update object