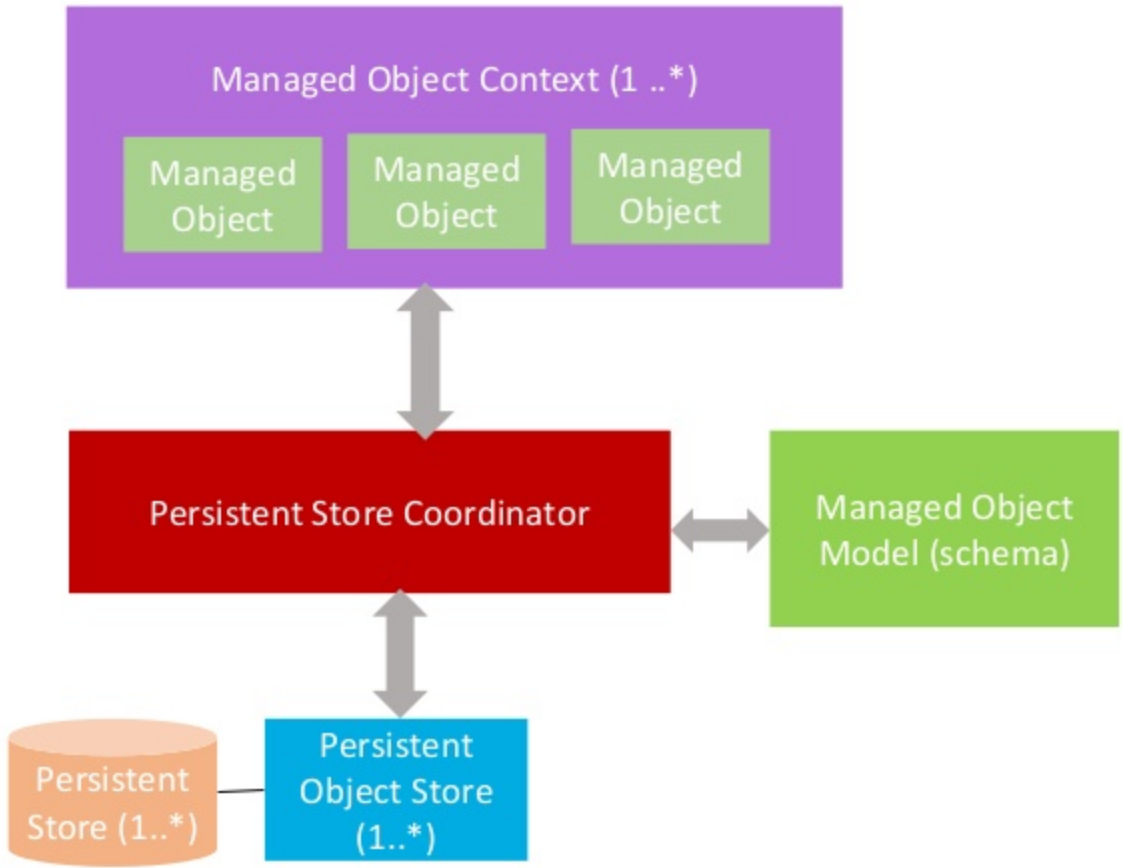
Core Data Migrations and …can we do better?

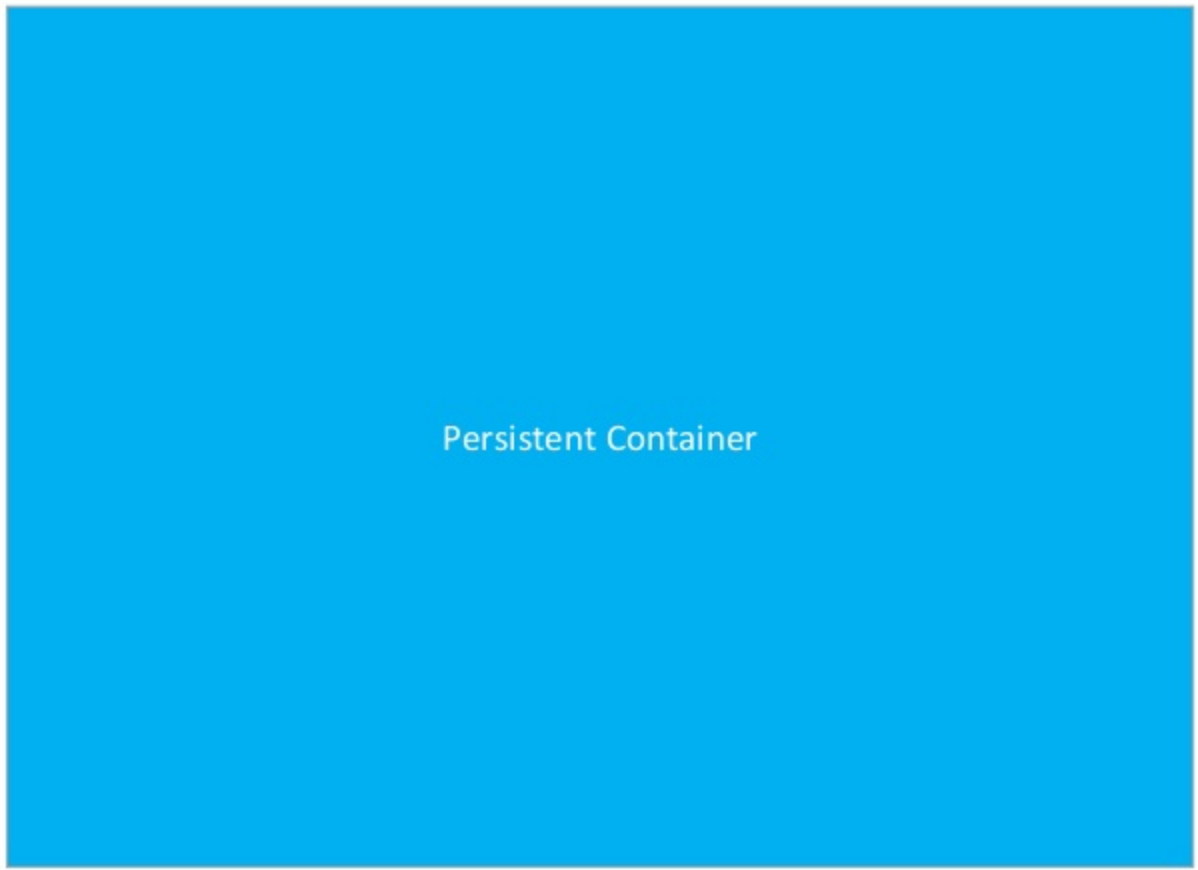
Priya Rajagopal
@rajagp

Core Data

Core Data Stack



Core Data Stack : iOS10



Managed Object Model

- Schema Definition
- Entities , Attributes and Relationships
- Versioning

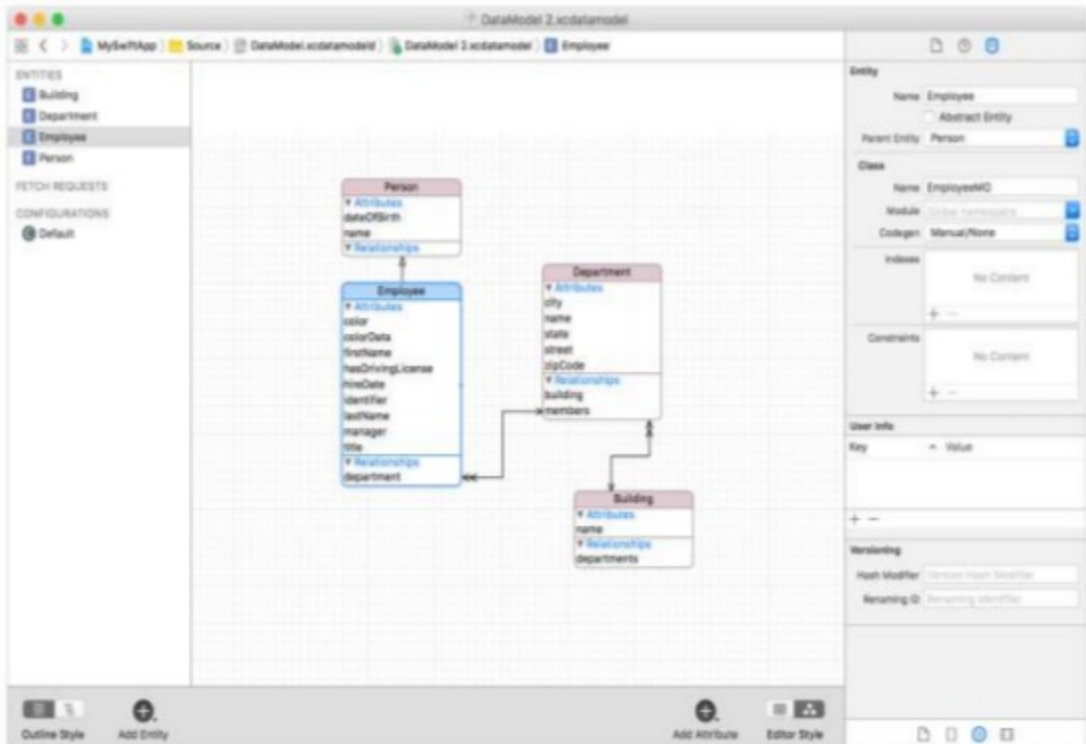


Image source: https://developer.apple.com/

Schema Changes

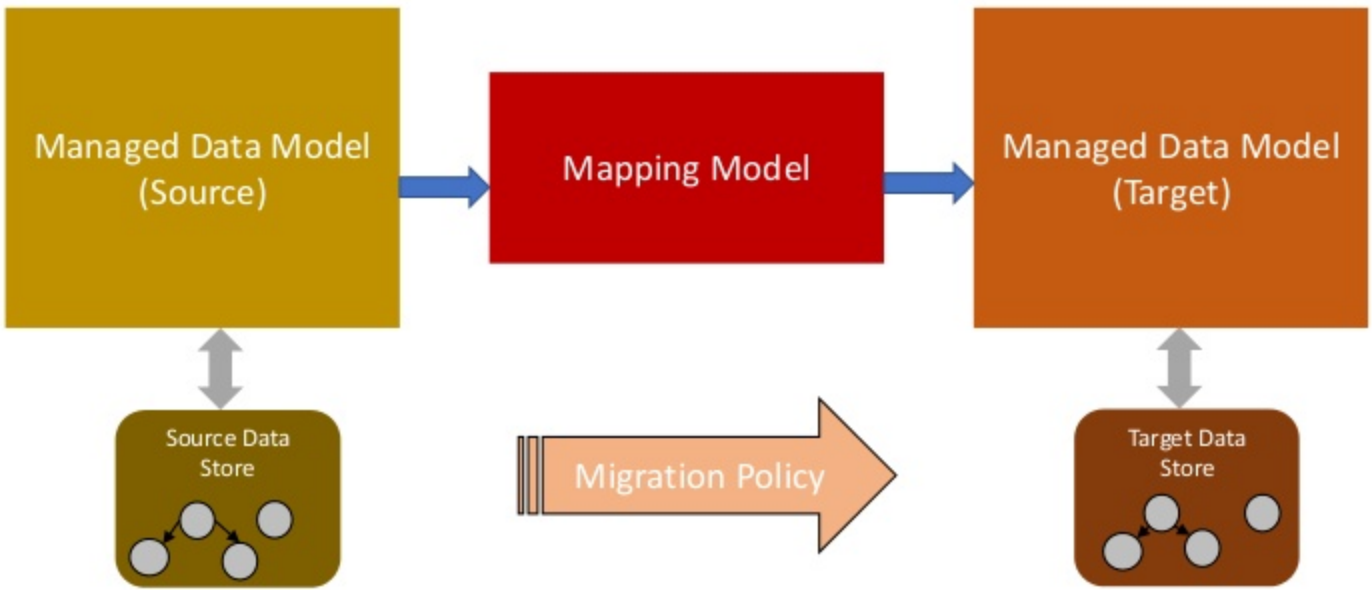
The Dreaded Persistent Store / Data Model Mismatch Exception!

```
CoreData: error: -addPersistentStoreWithType:SQLite configuration:(null) URL:file:///Users/priya.rajagopal/Library/
Developer/CoreSimulator/Devices/13C8EDF8-4ABE-4CC4-8A3A-C709EF4C9EFA/data/Containers/Data/Application/
70B8CB7A-22DD-4D91-A24E-B5C3455BEEB0/DocumentsUniversityDataModel.sqlite options:{
    NSInferMappingModelAutomaticallyOption = 0;
    NSMigratePersistentStoresAutomaticallyOption = 0;
} ... returned error Error Domain=NSCocoaErrorDomain Code=134100 "(null)" UserInfo={metadata={
    NSPersistenceFrameworkVersion = 754;
    NSStoreModelVersionHashes =     {
        University = <7be9d69e acfa46dc e50a44e8 1e1b9a38 cd5a408a d76cddc9 8d653e86 66f1b7cd>;
    };
    NSStoreModelVersionHashesVersion = 3;
    NSStoreModelVersionIdentifiers =     (
        ""
    );
    NSStoreType = SQLite;
    NSStoreUUID = "F2D1918A-5AFE-4511-BD29-F0F9F6BDF610";
    "_NSAutoVacuumLevel" = 2;
}, reason=The model used to open the store is incompatible with the one used to create the store} with userInfo
dictionary {
    metadata =     {
        NSPersistenceFrameworkVersion = 754;
        NSStoreModelVersionHashes =         {
            University = <7be9d69e acfa46dc e50a44e8 1e1b9a38 cd5a408a d76cddc9 8d653e86 66f1b7cd>;
        };
        NSStoreModelVersionHashesVersion = 3;
        NSStoreModelVersionIdentifiers =         (
            ""
        );
        NSStoreType = SQLite;
        NSStoreUUID = "F2D1918A-5AFE-4511-BD29-F0F9F6BDF610";
        "_NSAutoVacuumLevel" = 2;
    };
    reason = "The model used to open the store is incompatible with the one used to create the store";
}
##Thread is <NSThread: 0x61000006eb00>{number = 3, name = (null)}, main is <NSThread: 0x60000006a080>{number = 1, name
```

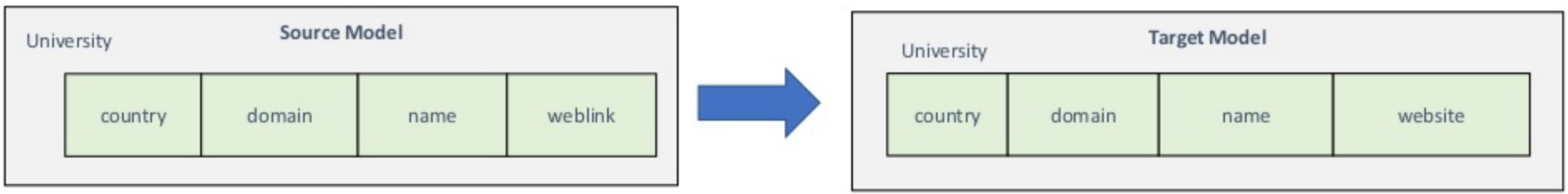
Option 1 : Delete the app and reinstall

Migrations

Option 2: Core Data Migrations



Case 1



Lightweight Migrations

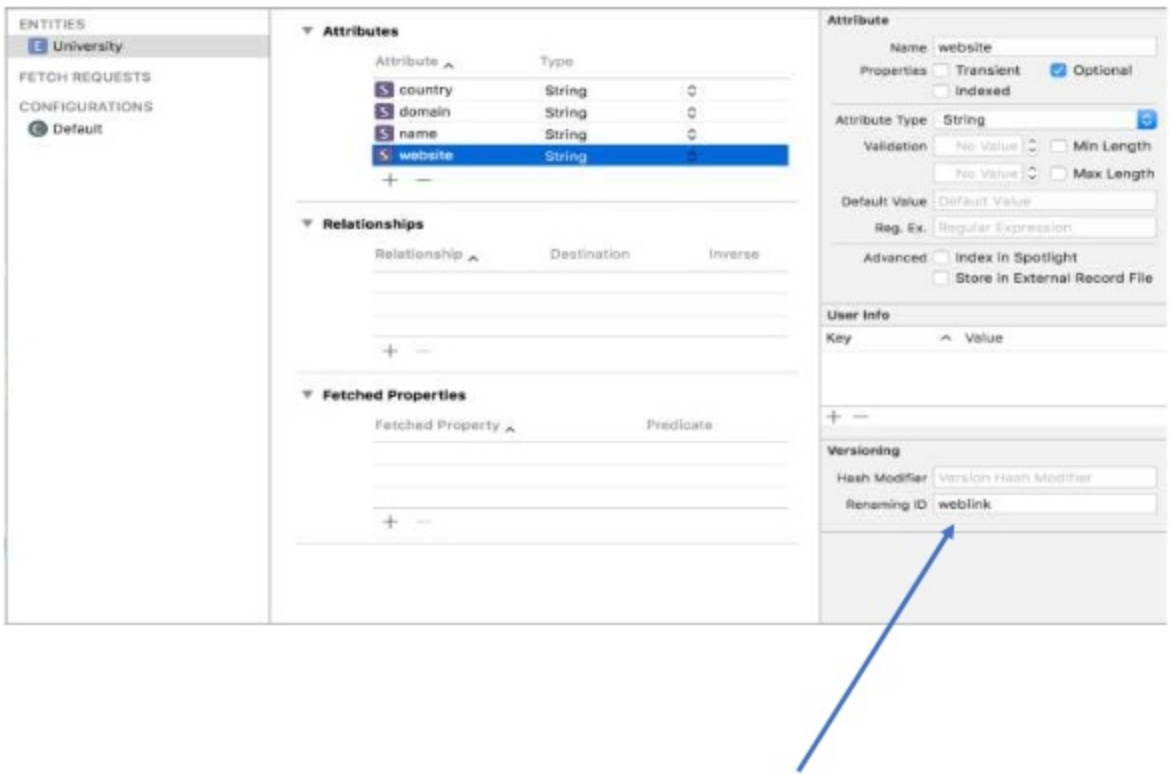
- Automatic Migrations
- Core Data infers mapping model from source to target model
- Fast
- Handles following Changes
  - Simple addition /removal of a new attribute
  - A non-optional attribute becoming optional
  - An optional attribute becoming non-optional, *and defining a default value*
  - Renaming an entity or property
  - Simple Relationship changes
  - Simple entity hierarchy changes
- Complete list

https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreDataVersioning/Articles/vmLightweightMigration.html#//apple_ref/doc/uid/TP40004399-CH4-SW2

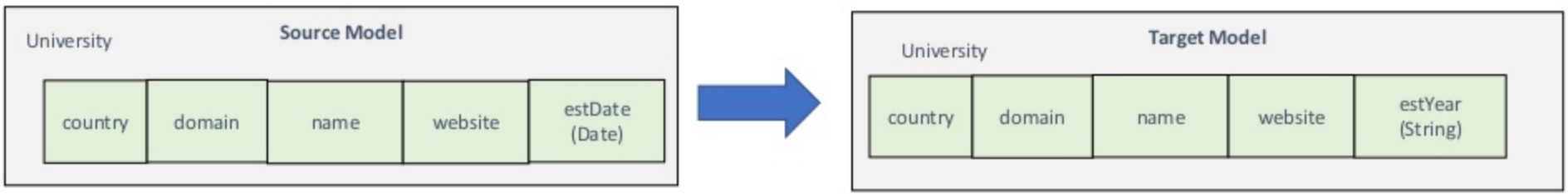
Configure Core Data Stack

```
let persistentStoreDesc = NSPersistentStoreDescription(url: storeUrl)

persistentStoreDesc.shouldMigrateStoreAutomatically = true

persistentStoreDesc.shouldInferMappingModelAutomatically = true

self.persistentContainer?.persistentStoreDescriptions = [persistentStoreDesc]
```

Some help from editor



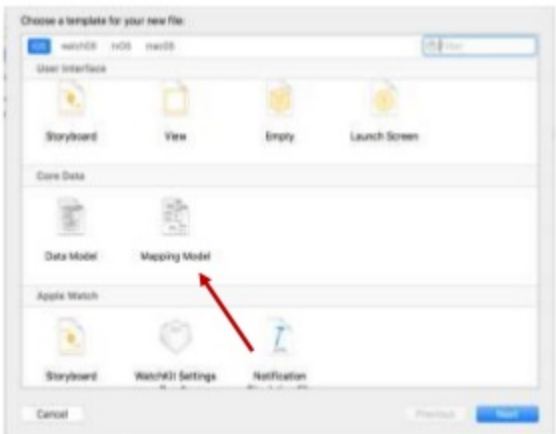
Case 2

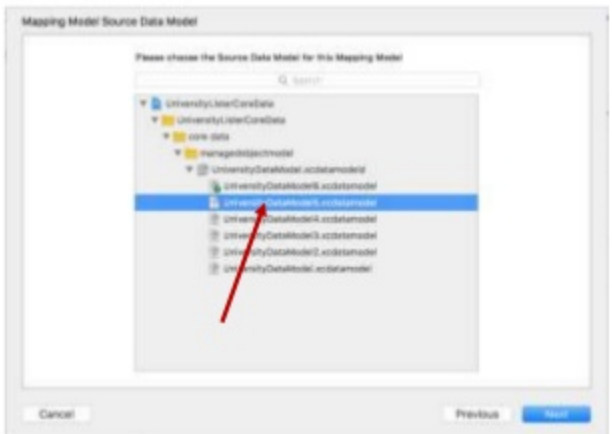


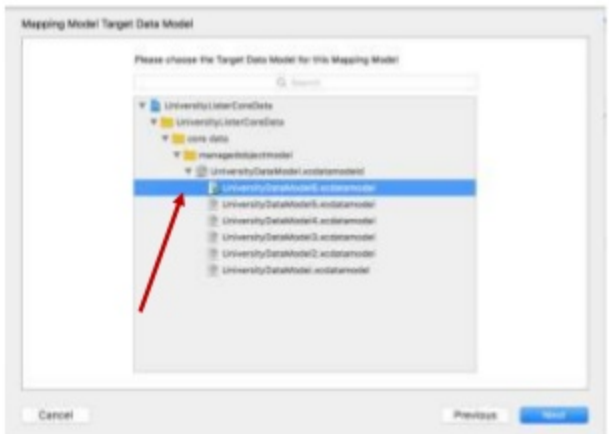
Custom Migration Policy

```
// Transform estDate of type Date to estYear of type String

class UniversityDataMigrationPolicy: NSEntityMigrationPolicy {


    func transformEstDateToEstYear(_ estDate:NSDate)->String {

        let calendar = NSCalendar.current
        let components =  calendar.dateComponents(Set<Calendar.Component>([.year]), from: estDate as
Date)

        return "\(components.year!)"

    }
}
```

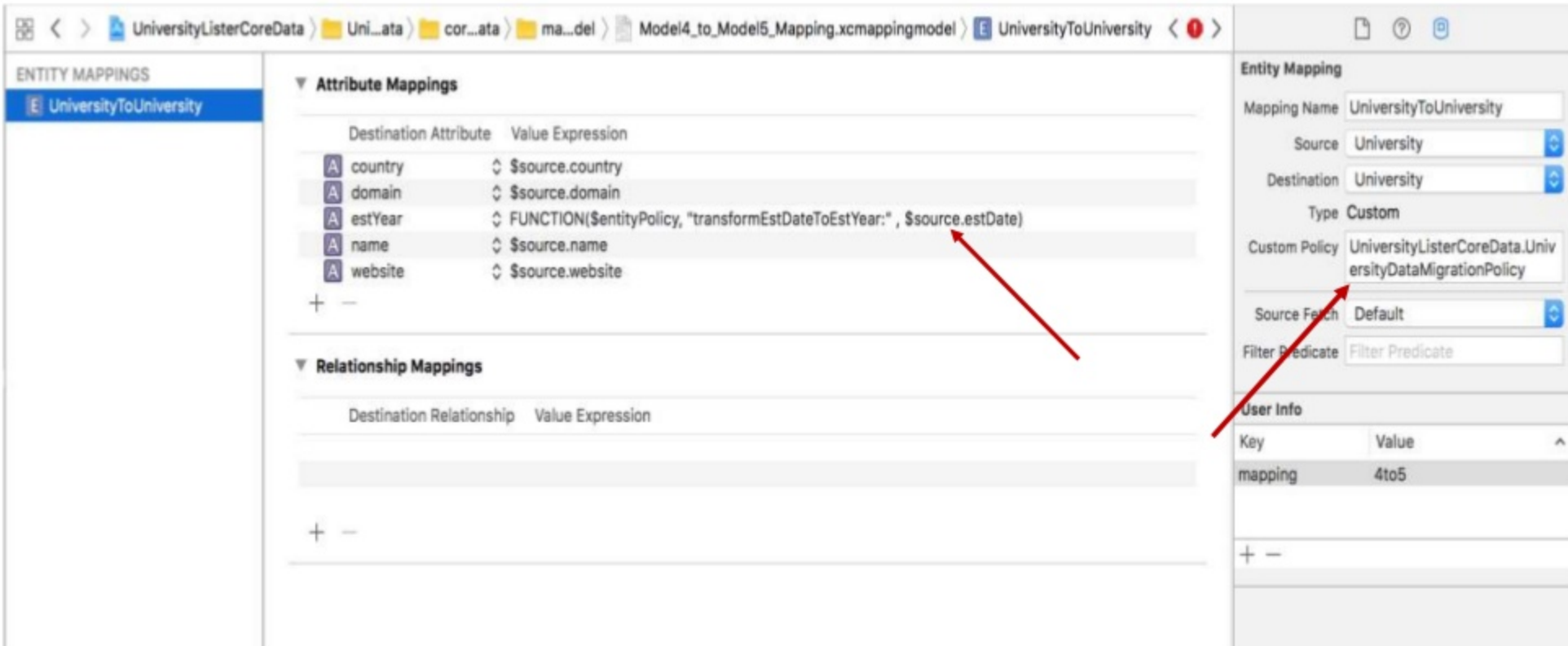
Define Custom Mapping



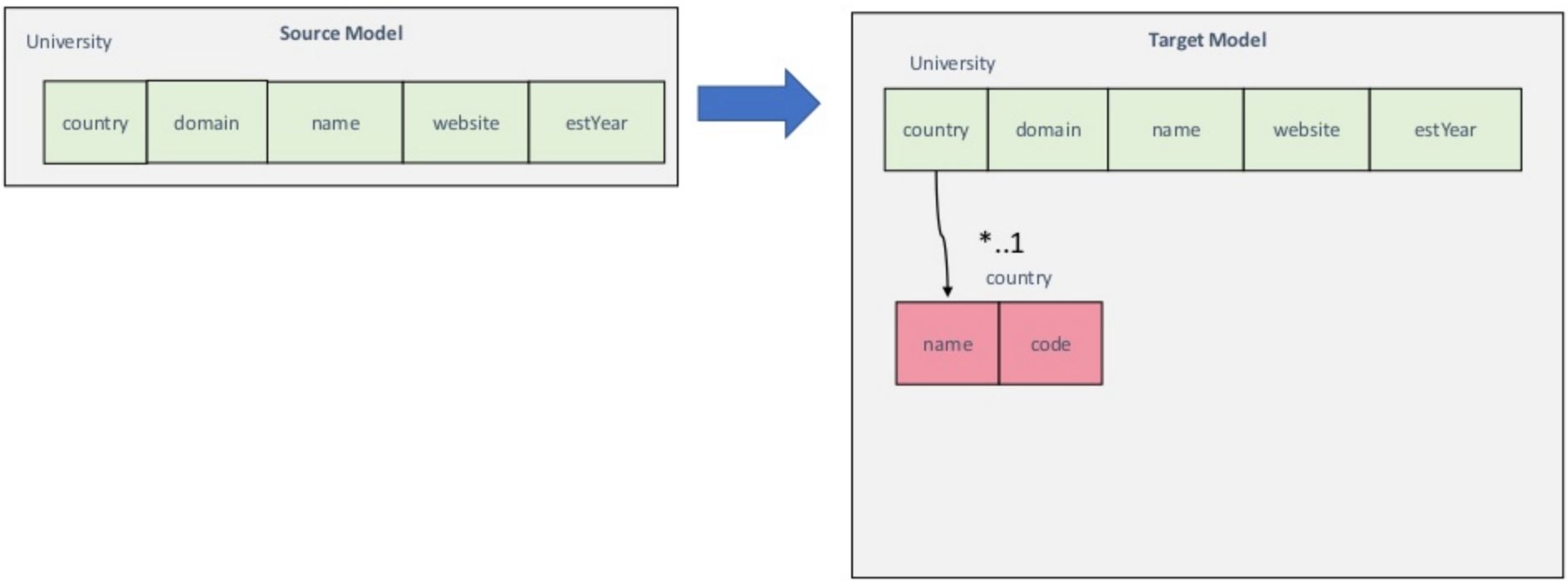




Custom Mapping



Case 3


University
Source Model
country
domain
name
website
estYear
Target Model
University
country
domain
name
website
estYear
*..1
country
name
code

Custom Migration Policy (1/3)

```
class UniversityMigrationPolicy: NSEntityMigrationPolicy {

    override func createDestinationInstances(forSource sInstance: NSManagedObject, in mapping:
NSEntityMapping, manager: NSMigrationManager) throws {

        // 1. Create destination Instance corresponding to the source instance
        let dInstance = NSEntityDescription.insertNewObject(forEntityName: mapping.destinationEntityName!,
into: manager.destinationContext)

        let sKeys = sInstance.entity.attributesByName.keys
        let sValues = sInstance.dictionaryWithValues(forKeys: Array(sKeys))
        let dKeys = dInstance.entity.attributesByName.keys
        for dKey in dKeys {

          if let matchingValueInSrc = sValues[dKey] {
             dInstance.setValue(matchingValueInSrc, forKey:  dKey)
          }

       }
```

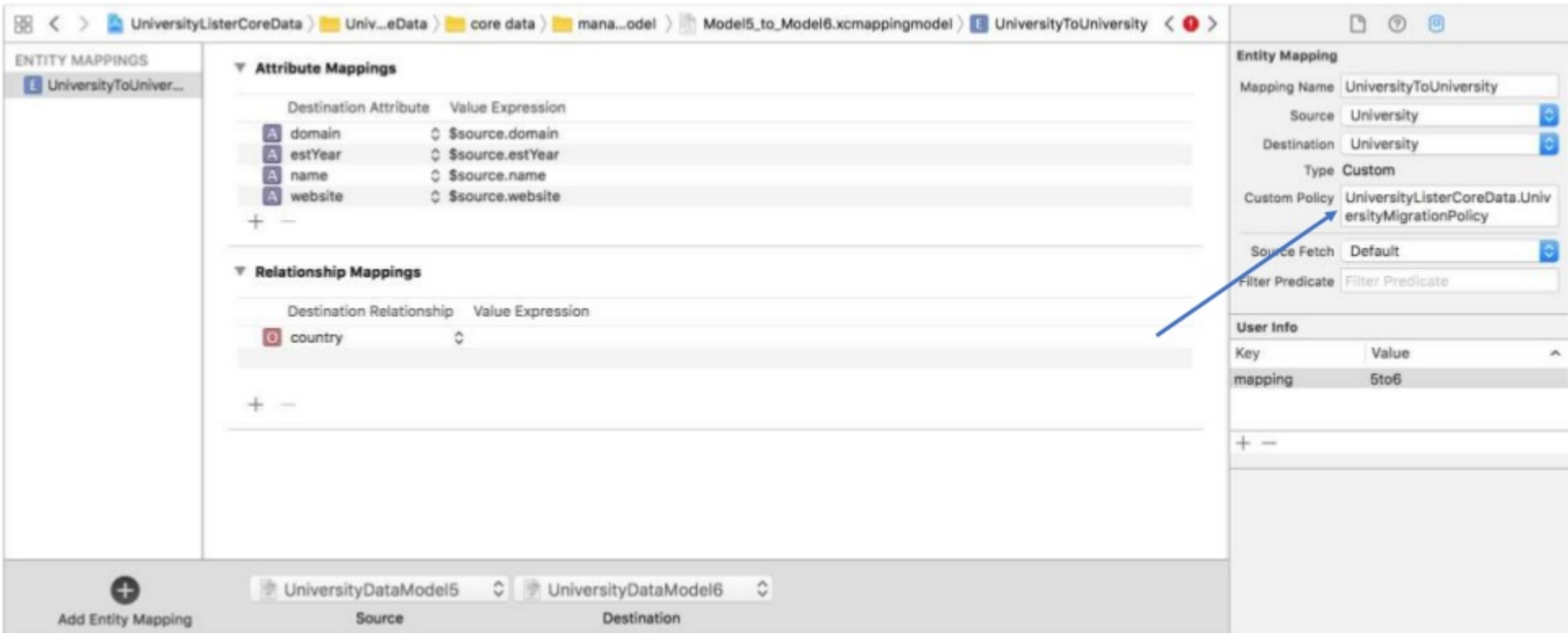
Custom Migration Policy (2/3)

```
let universityName = sValues["name"]
if let countryName = sInstance.value(forKey: "country") as? String                {

                    // 2 . We are tracking the Country managed objects in Migration Manager's userInfo
                    //      This is to ensure we create a single instance of Country for every countryName

    if let userInfo = manager.userInfo , let country = userInfo[countryName] as? NSManagedObject{
       // link university entity to country
       dInstance.setValue(country, forKey: "country")

    }
    else {
```

Custom Migration Policy (3/3)

```
        // 3 . Create Country object
                    let country = NSEntityDescription.insertNewObject(forEntityName: "Country", into: manager.destinationContext)
        // Set country name
        country.setValue(countryName, forKey:  "name")

        // 4. Link University object with the corresponding Country object
        dInstance.setValue(country, forKey: "country")

        // 5. Track the newly created country object in Migration Manager's userInfo
        if let _ = manager.userInfo {
            manager.userInfo?[countryName] = country
        }
        else {
            manager.userInfo = [:]

            manager.userInfo?[countryName] = country
        }

    }
            // 4. Associate the source with destination instance

    manager.associate(sourceInstance: sInstance, withDestinationInstance: dInstance, for: mapping)

}
```

Custom Mapping



Manual

```
let persistentStoreDesc = NSPersistentStoreDescription(url: storeUrl)
persistentStoreDesc.shouldMigrateStoreAutomatically = false
persistentStoreDesc.shouldInferMappingModelAutomatically = false
self.persistentContainer?.persistentStoreDescriptions = [persistentStoreDesc]
```

- Create Custom Migration Policies
- Create Mapping Models

1. Test if Migration is Needed

```
func isMigrationNeeded()->Bool {
        ......
    let srcMetadata = try NSPersistentStoreCoordinator.metadataForPersistentStore(ofType: NSSQLiteStoreType, at: sourceUrl,
options: nil)

    let momPathValue = Bundle.main.path(forResource: "UniversityDataModel", ofType:"momd")

    guard let destModel = NSManagedObjectModel(contentsOf: URL(fileURLWithPath: momPathValue)) else { return false }

    return destModel.isConfiguration(withName: nil, compatibleWithStoreMetadata: srcMetadata) == false ? true: false
......

    }
```

2. Migrate Stores Progressively

```
func progressivelyMigrate() throws {

    // Get the model version that is compatible with the current data store
    let idx = try indexOfCompatibleMom(at: sourceUrl, moms: self.models)
    let remaining = self.models.suffix(from: (idx + 1))
    guard remaining.count > 0 else {
        // The stored DB is compatible with the latest model
        return // migration not necessary
    }

     _ = try remaining.reduce(self.models[idx]) { smom, dmom in
         do {
                try self.migrate(srcStoreUrl: sourceUrl, srcModel: smom, dstModel: dmom)
            }
             catch {
                        // handle error
            }
            return dmom
    }

}
```

3. Identify Model compatible with Store

```
func indexOfCompatibleMom(at storeURL: URL, moms: [NSManagedObjectModel]) throws -> Int {

    let meta = try NSPersistentStoreCoordinator.metadataForPersistentStore(ofType: NSSQLiteStoreType, at: storeURL)
    guard let idx = moms.index(where: { $0.isConfiguration(withName: nil, compatibleWithStoreMetadata: meta) }) else {
        throw MigrationError.IncompatibleModels
    }
    return idx

}
```

4. Migrate Store from src to dst model

```
func migrate(srcStoreUrl:URL, srcModel:NSManagedObjectModel,dstModel:NSManagedObjectModel)throws {

  // Prepare temp directory for destination
  let dir = URL(fileURLWithPath: NSTemporaryDirectory()).appendingPathComponent(UUID().uuidString)
  try FileManager.default.createDirectory(at: dir, withIntermediateDirectories: true, attributes: nil)
  defer {
        _ = try? FileManager.default.removeItem(at: dir)
   }

   // Perform migration
   let mapping = try findMapping(from: srcModel, to: dstModel)
   let destURL = dir.appendingPathComponent(srcStoreUrl.lastPathComponent)
   let manager = NSMigrationManager(sourceModel: srcModel, destinationModel: dstModel)
   try autoreleasepool {
      try manager.migrateStore(
         from: srcStoreUrl,
              sourceType: NSSQLiteStoreType,
              options: nil,
              with: mapping,
              toDestinationURL: destURL,
              destinationType: NSSQLiteStoreType,
              destinationOptions: nil
          )
   }
```

5. Find Mapping Model

```
fileprivate func findMapping(from smom: NSManagedObjectModel, to dmom: NSManagedObjectModel) throws ->
NSMappingModel {

    if let mapping = NSMappingModel(from: Bundle.allBundles, forSourceModel: smom, destinationModel: dmom) {
        return mapping // found custom mapping
    }
    // Return inferred mapping model if a custom model not defined.
    return try NSMappingModel.inferredMappingModel(forSourceModel: smom, destinationModel: dmom)
}
```


CAN WE DO BETTER?
imgflip.com

NoSQL Option for Persisting Data

- Non Relational
- Unstructured or Semi Structured Data
- Scalable – both up and down
- SQL-type Queries

Key-Value
- Couchbase
- Riak
- BerkeleyDB
- Redis
- …

Document
- Couchbase
- MongoDB
- DynamoDB
- DocumentDB

Graph
- OrientDB
- Neo4J
- DEX
- GraphBase

Wide Column
- Hbase
- Cassandra
- Hypertable

Data Modeling

Data Concern	Core Data	JSON Document Model (Couchbase)
Rich Structure	▪ Managed Object Model - entities	▪ Documents = containers of related KV pairs
Relationships	▪ Represented	▪ Represented (embed/reference)
Structure Evolution	▪ Migrations	✓ Flexible ✓ Dynamic change
Value Evolution	▪ Data can be updated	▪ Data can be updated
Query Interface	▪ NSFetchRequest ▪ NSFetchedResultsControllerDelegate	▪ Query / LiveQuery

JSON to Native Mapping

```
{
  "type": "UniversityClub",
  "name":"Michigan Movie Club",
  "ID":"A123456",
  "activeStudents":["S100"],
  "alumini":["A900"]
}
```

```
struct UniversityClub :Codable {
    let type:Type = Type.UniversityClub
    let ID:String
    var name:String
    var students:[String]
    var alumini:[String]
}
```

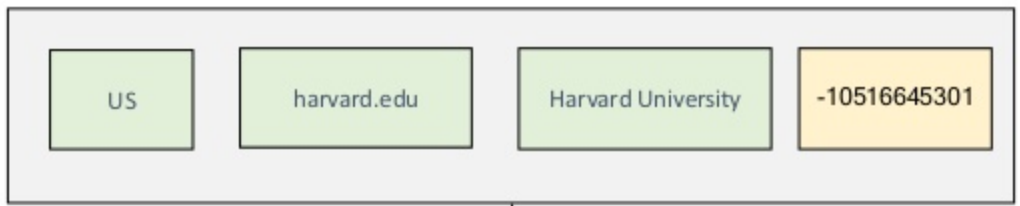
```
{
  "type": "Student",
  "name":"Jane Doe",
  "ID":"S100",
  "skills":["writing","public speaking"]
}
```

```
struct Student:Codable {
    let type:Type = Type.Student
    let ID:String
    var name:String
    var skills:[String]
}
```

```
{
  "type": "Alum",
  "name":"John Appleseed",
  "ID":"A900",
  "experience":[
    {
      "company":"Dreamworks",
      "role":"Designer"
    }
  ]
}
```

```
struct Alum:Codable {
    let type:Type = Type.Alum
    let ID:String
    var name:String
    var experience:[Experience]

    struct Experience:Codable {
        let company:String
        let role:String
    }
}
```

NoSQL: Example 1

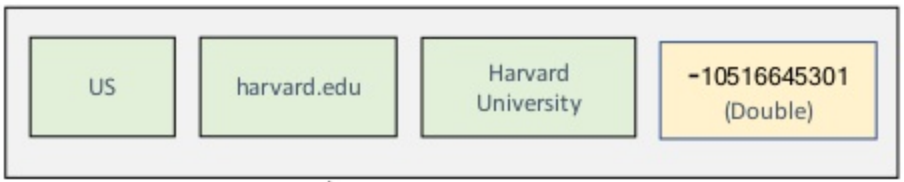


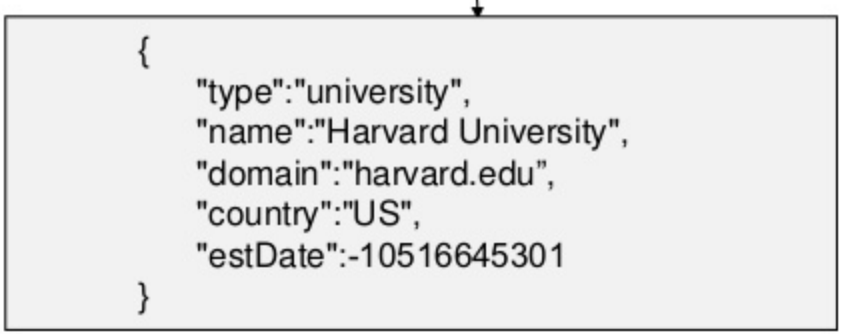

```
{
    "type":"university",
     "name":"University of Michigan",
     "domain":"umich.edu",
     "country":"US"
}
```

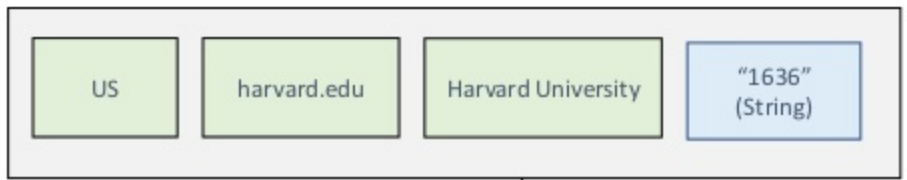


```
{
    "type":"university",
     "name":"Harvard University",
    "domain":"harvard.edu",
    "country":"US",
     "estDate":-10516645301
}
```

NoSQL : Example 2








```
let estDate = university.double(forKey: "estDate")

if estDate != 0.0 {
        // Handle estDate
 }else if let estYear = university.string(forKey: "estYear") {
        // Handle estYear
}
```

```
{
    "type":"university",
    "name":"Harvard University",
    "domain":"harvard.edu",
    "country":"US",
    "estYear": "1636"
}
```

A NoSQL Option for iOS : A Sneak Peek

Couchbase Mobile Platform: Full Stack Database Platform

Couchbase Lite

Sync Gateway

Couchbase Server

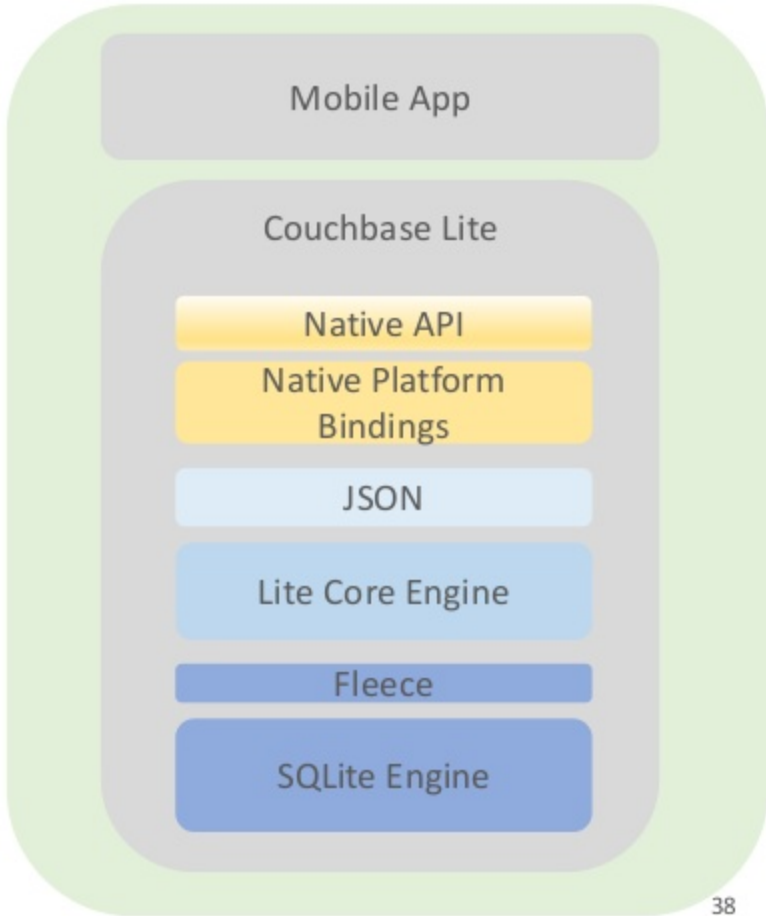
NoSQL Document Style

EMBEDDED DATABASE | SYNCHRONIZATION | DATABASE SERVER

SECURITY

Couchbase Lite

- Embedded NoSQL Database
- JSON Document Store
- Open Source
- iOS, Android, .NET…
- Encryption Support
- Powerful Query Interface, FTS
- Sync (full Couchbase stack)
- Conflict Resolver



Resources

- **Couchbase Developer Portal**
  - https://developer.couchbase.com
- **Source Code**
  - **https://github.com/couchbaselabs**
  - https://github.com/couchbase/
- **“Office Hours”**

Integrating into your iOS Project

developer.couchbase.com

Carthage

1. Install Carthage
2. Add `github "couchbase/couchbase-lite-ios" "2.0DB012"` to your **Cartfile**.
3. Run `carthage update --platform ios`.
4. Drag **CouchbaseLiteSwift.framework** from **Carthage/Build/** to the Xcode navigator.
5. Click on Project > General > Embedded Binary and add **CouchbaseLiteSwift.framework** to this section.

CocoaPods

1. Install Cocoapods
2. In your `Podfile`, add the following.

```
target '<your target name>' do
  use_frameworks!
  pod 'CouchbaseLiteSwift', :git => 'https://github.com/couchbase/couchbase-lite-ios.git',
:tag => '2.0DB012', :submodules => true
end
```

3. Install the pods and open the .xcworkspace file generated by Cocoapods.

```
pod install
```

Creating / Opening a Database

```
// Set Database configuration
var config = DatabaseConfiguration()
config.directory = userFolderPath
config.encryptionKey( /*encryption key object*/ )
config. conflictResolver(/*Set custom resolver */)

// Create / Open a database with specified name and configuration
_db = try Database(name: kDBName, config: options)
```

Inserting Document

```
// Create a new document or update existing document
// If Id is not provided, system generates Id
let document = Document.init(id, dictionary: props)
try database.save(document)
```

Supported Data Types :

- *Date*
- *Number*
- *Null*
- *String*
- *Array*
- *Blob*
- *Dictionary*

Typed Accessors

Live Queries (w/ Full Text Search)

```
let query = Query.select(SelectResult.all())
        .from(DataSource.database(database))
        .where((Expression.property("skills").in(["music","writing"]))
           .and(Expression.property("type").match("student")))
        .orderBy(Ordering.property("name").ascending()). toLive();

 // Register for live query changes
_liveQueryListener = _bookingQuery?.addChangeListener({ [weak self](change) in
            for (_, row) in (change.rows?.enumerated())! {
           // handle row.document
           default:
 }})

query.run();
```

Thank you

@rajagp

Couchbase