CoreData sucks!

So we did something about it.

Javan Wood

(Jay•van)

@javn.wd

iOS Engineer

REDEYE

In this talk…

- Shipping code
- Building a bespoke persistence layer
- Engineering for developer happiness (among other things)

Context

- Working on RedEyeWFM
  - Enterprise SaaS
  - Live customers!

Context

→ Super stable

→ Moving to offline-first

→ Continuous releases

Context

→ Super stable

→ Moving to offline-first

→ Continuous releases

= (Solid persistence layer)

Solid persistence layer

- Versioned
- Easy to work with
- Support relationships with not-yet fetched objects

Support relationships with not-yet fetched objects

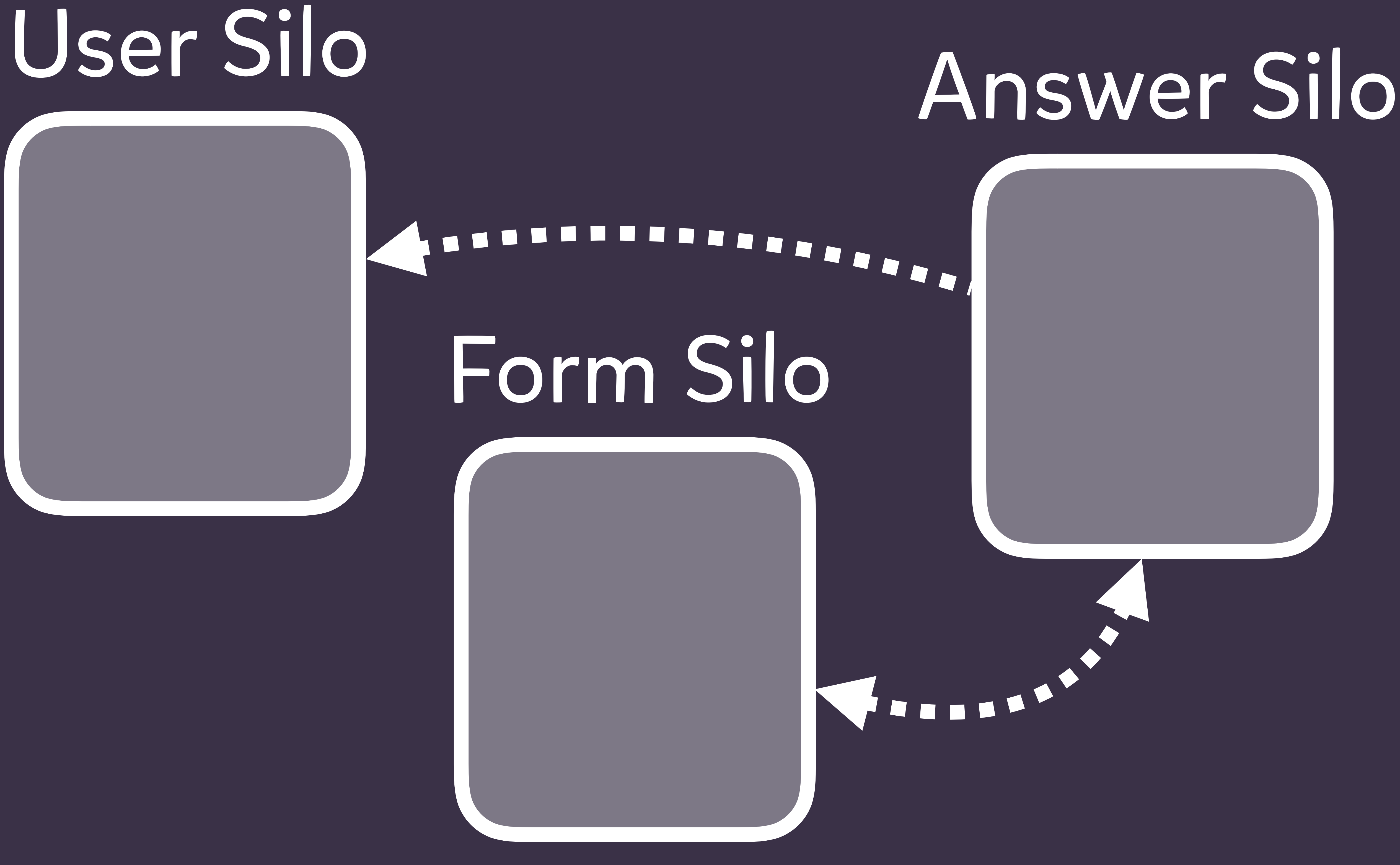
Support relationships with not-yet fetched objects

Wat?!

Not Your Average Data-Store™

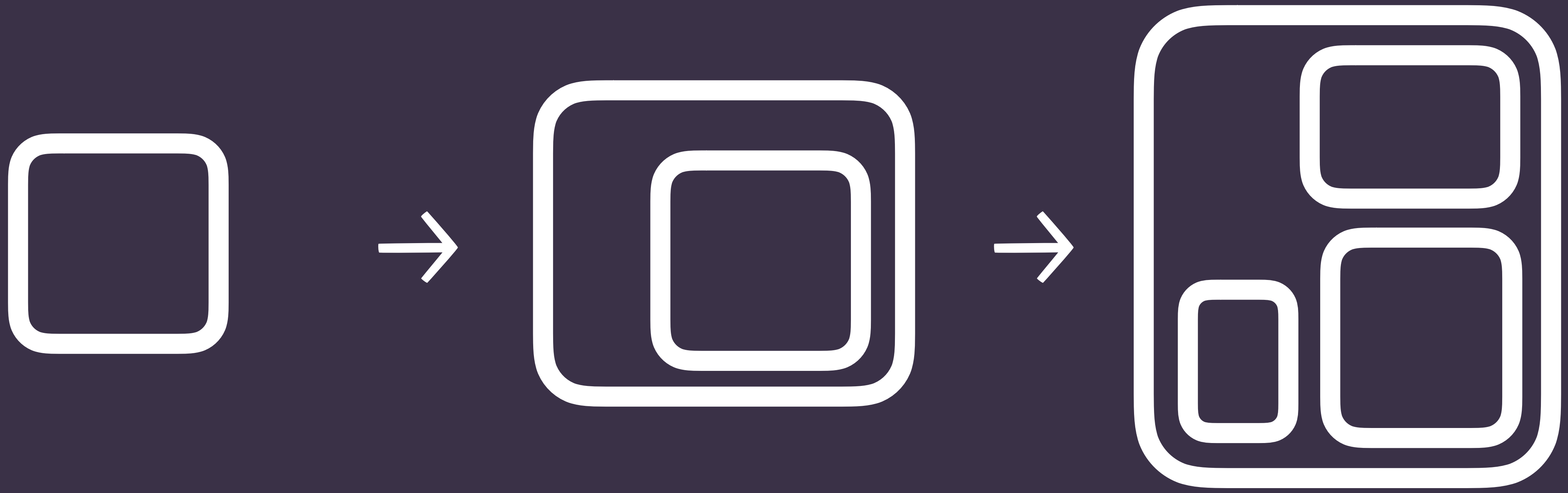
```
protocol Identifiable {
    var id: UniqueId { get set }
}

enum RelatedItem<T: Identifiable> {
    case unresolved(id: UniqueId)
    indirect case resolved(T)
}
```

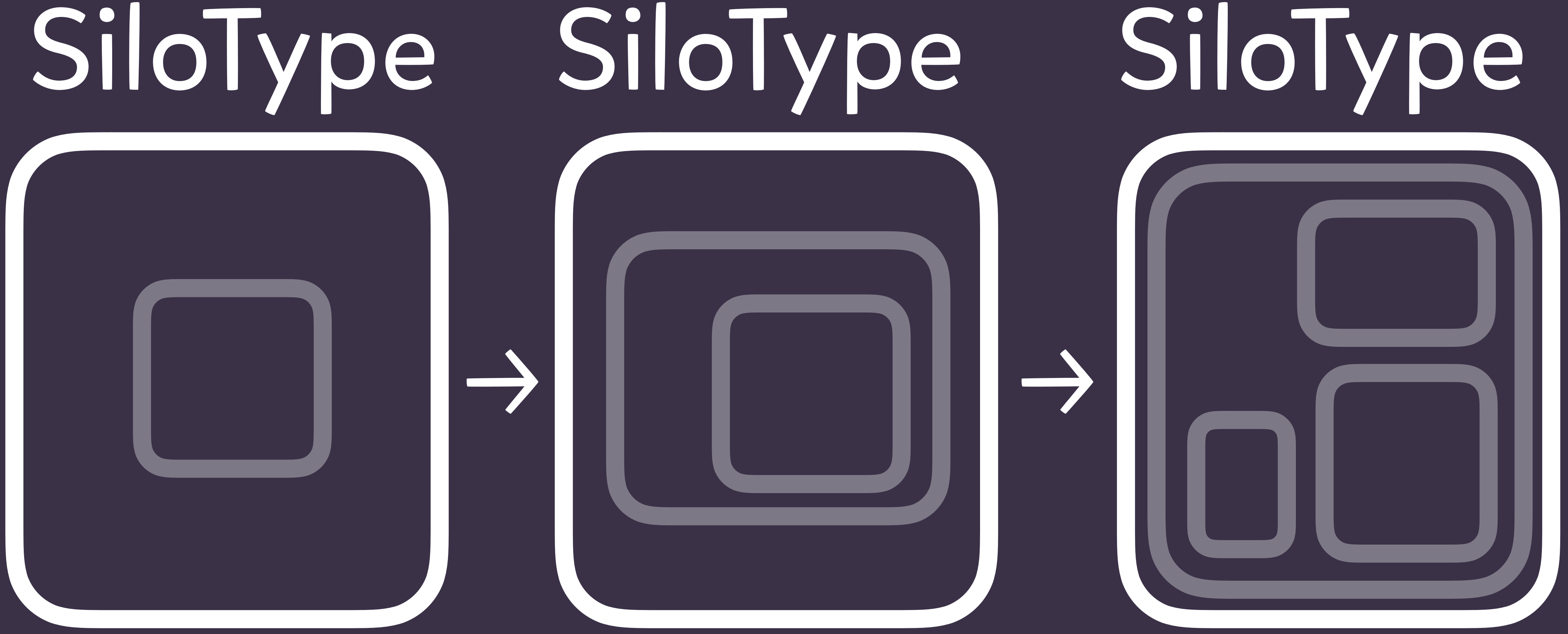

User Silo
Answer Silo
Form Silo

But we have to ship…

Solidifying incrementally



Solidifying incrementally



SiloTypes swappable!

Persistence incrementally

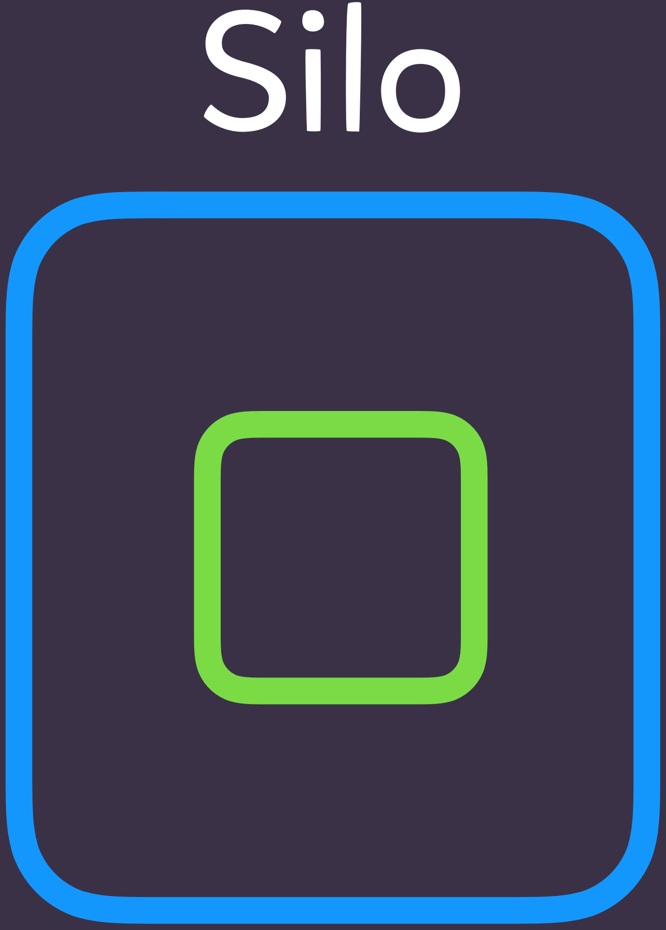
1. No silos
2. In-memory only silos
3. Disk-backed silos

Silo

```
protocol ValueStoreType {
    associatedtype ValueType
    ...
    // save / load etc.
}
```

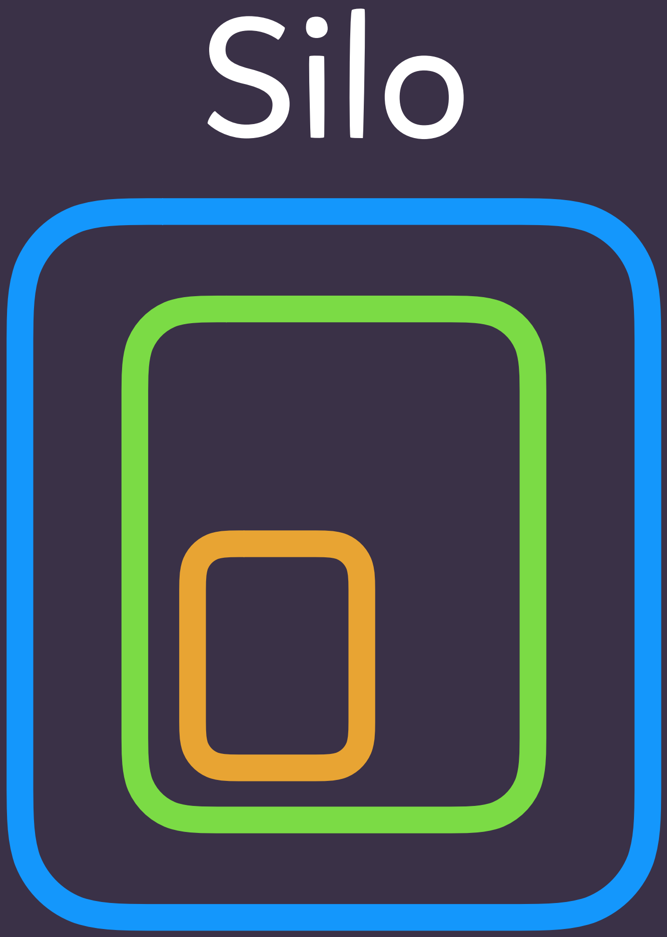
In-memory only

```
class InMemoryValueStore
    <ValueType>:
    ValueStoreType { ... }
```

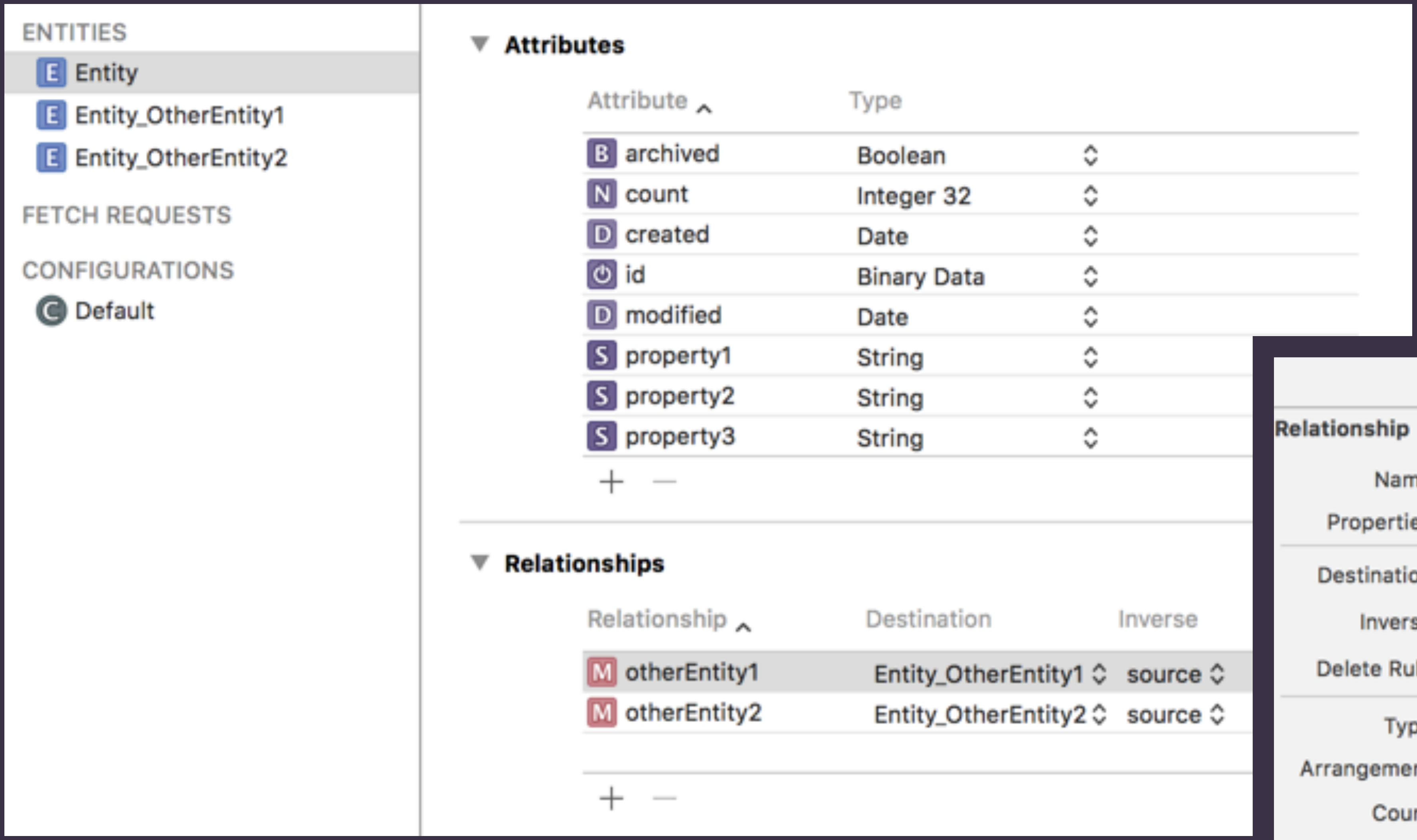


Persisted

```
class ManagedObjectValueStore
    <ValueType,
     ManagedObjectType>:
    ValueStoreType { ... }
```



What this looked like



ENTITIES
- Entity
- Entity_OtherEntity1
- Entity_OtherEntity2

FETCH REQUESTS

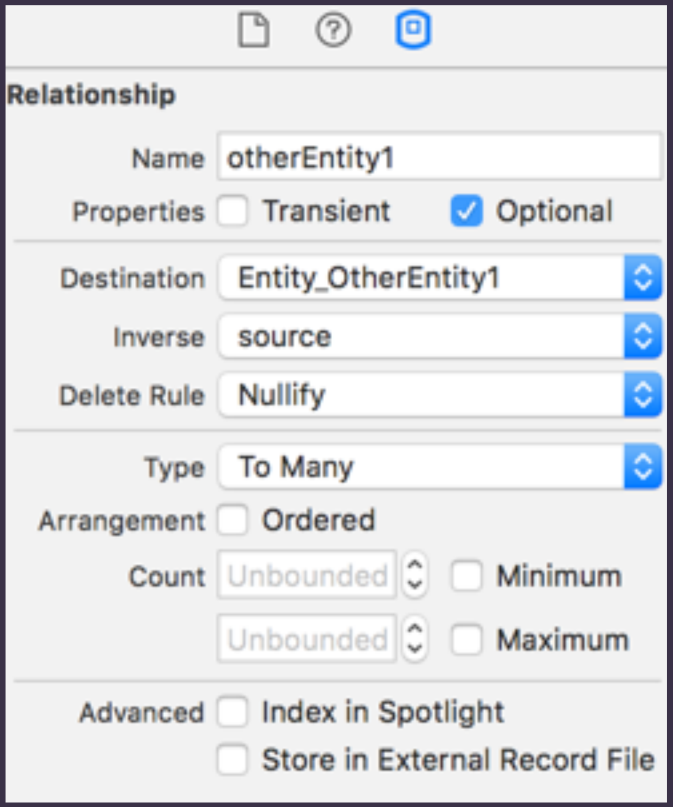
CONFIGURATIONS
- Default

Attributes

Attribute	Type
archived	Boolean
count	Integer 32
created	Date
id	Binary Data
modified	Date
property1	String
property2	String
property3	String

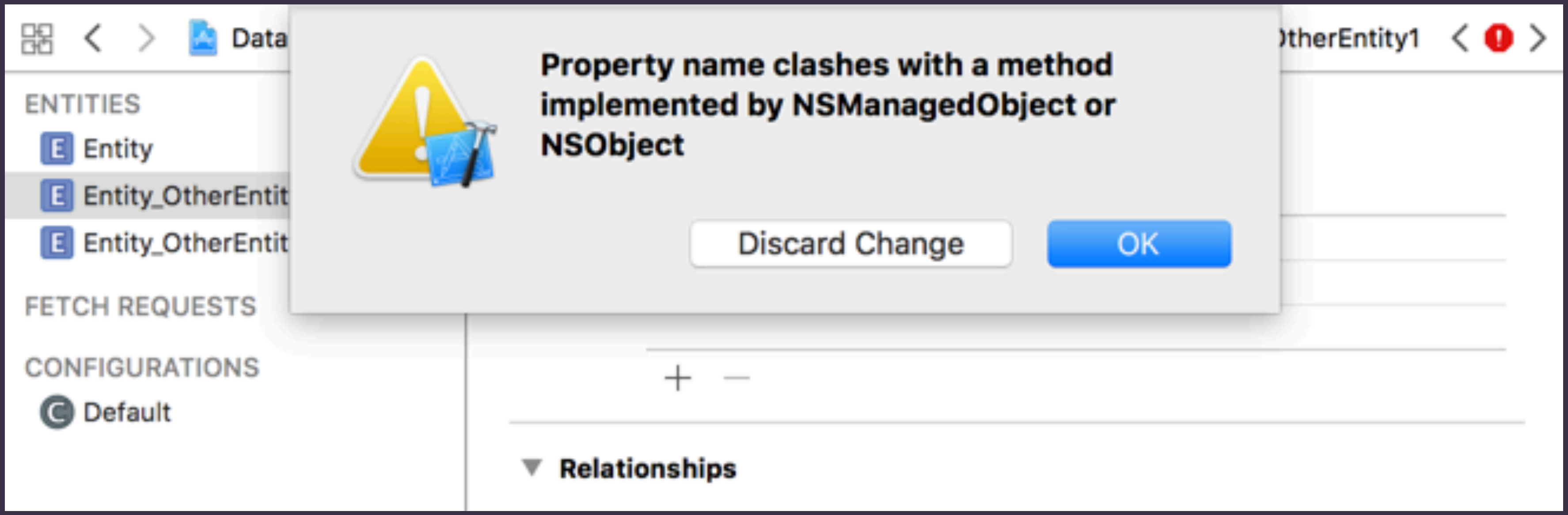
Relationships

Relationship	Destination	Inverse
otherEntity1	Entity_OtherEntity1	source
otherEntity2	Entity_OtherEntity2	source



Relationship

Name	otherEntity1
Properties	Transient [✓ Optional]
Destination	Entity_OtherEntity1
Inverse	source
Delete Rule	Nullify
Type	To Many
Arrangement	Ordered
Count	Unbounded Minimum
	Unbounded Maximum
Advanced	Index in Spotlight
	Store in External Record File


Property name clashes with a method implemented by NSManagedObject or NSObject

x Limited to Obj-C representations

x Limited control over naming

x Model is mangled with persistence implementation

Limitations of CoreData

- Tied us too deeply to a persistence implementation
- Incredibly easy to misuse, hard to use well
- Not even ACID compliant

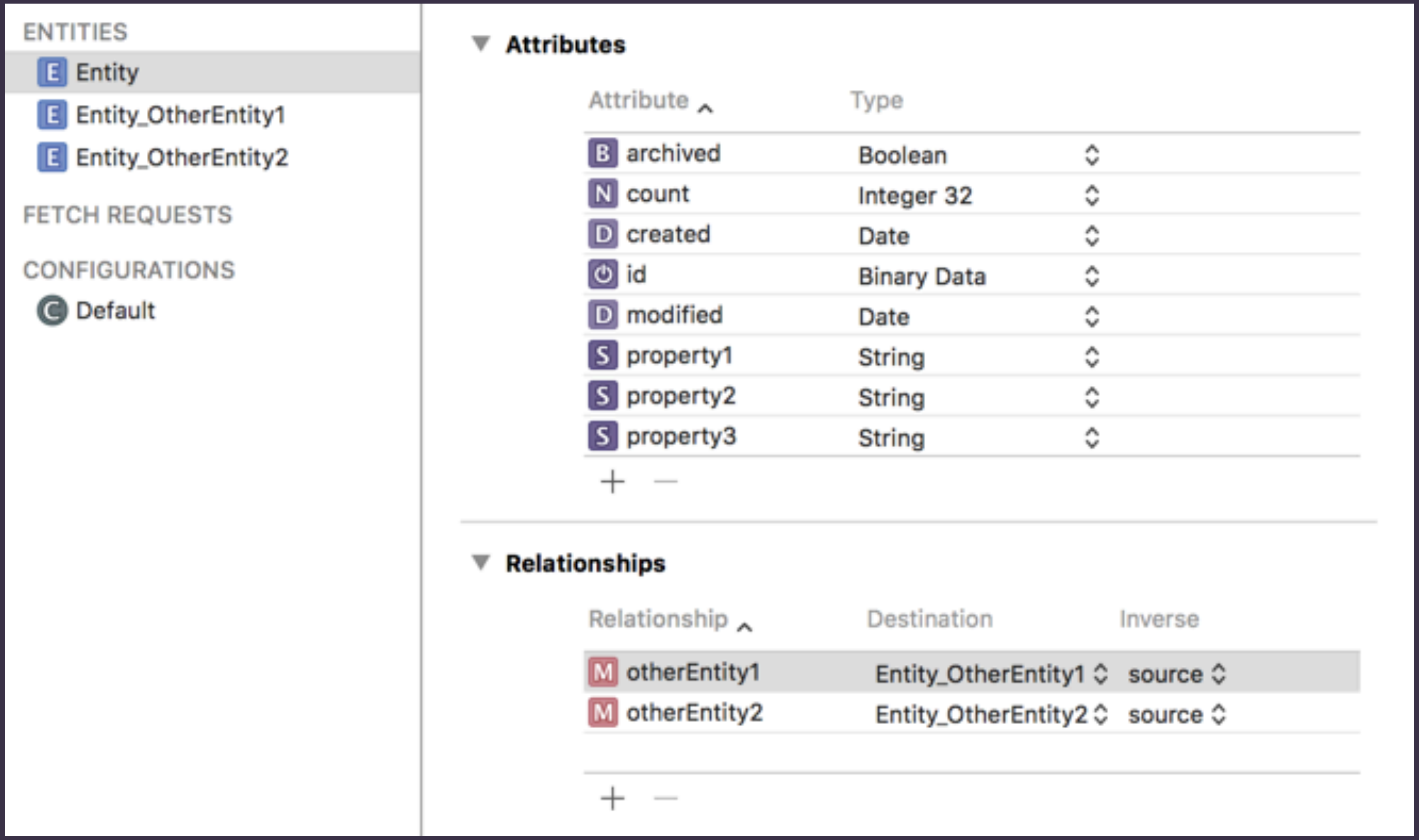
https://fatalerror.fm/episodes/2017/4/24/27-core-data

19 Entities in WFM

3 references to other silos per entity

→ 76 CoreData Entities

19x



ENTITIES

- E Entity
- E Entity_OtherEntity1
- E Entity_OtherEntity2

FETCH REQUESTS

CONFIGURATIONS

- C Default

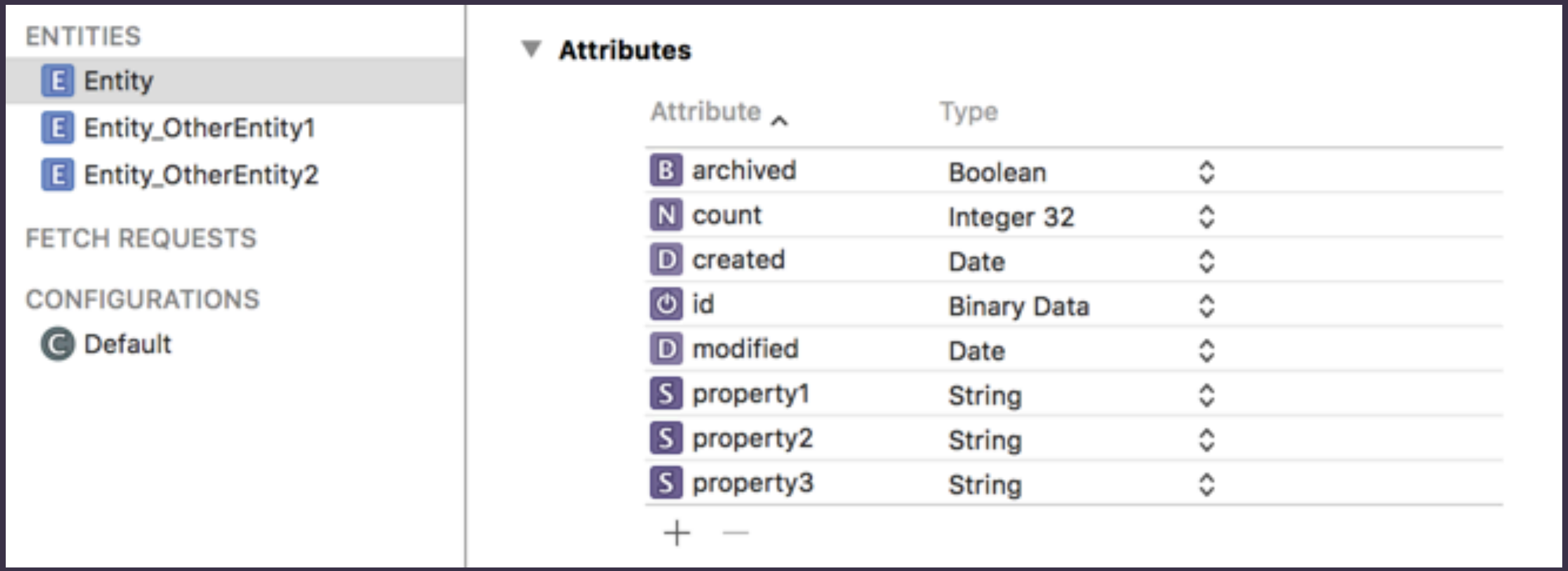
Attributes

Attribute	Type
B archived	Boolean
N count	Integer 32
D created	Date
id	Binary Data
D modified	Date
S property1	String
S property2	String
S property3	String

Relationships

Relationship	Destination	Inverse
M otherEntity1	Entity_OtherEntity1	source
M otherEntity2	Entity_OtherEntity2	source

x Can't define objects procedurally


ENTITIES
- Entity
- Entity_OtherEntity1
- Entity_OtherEntity2

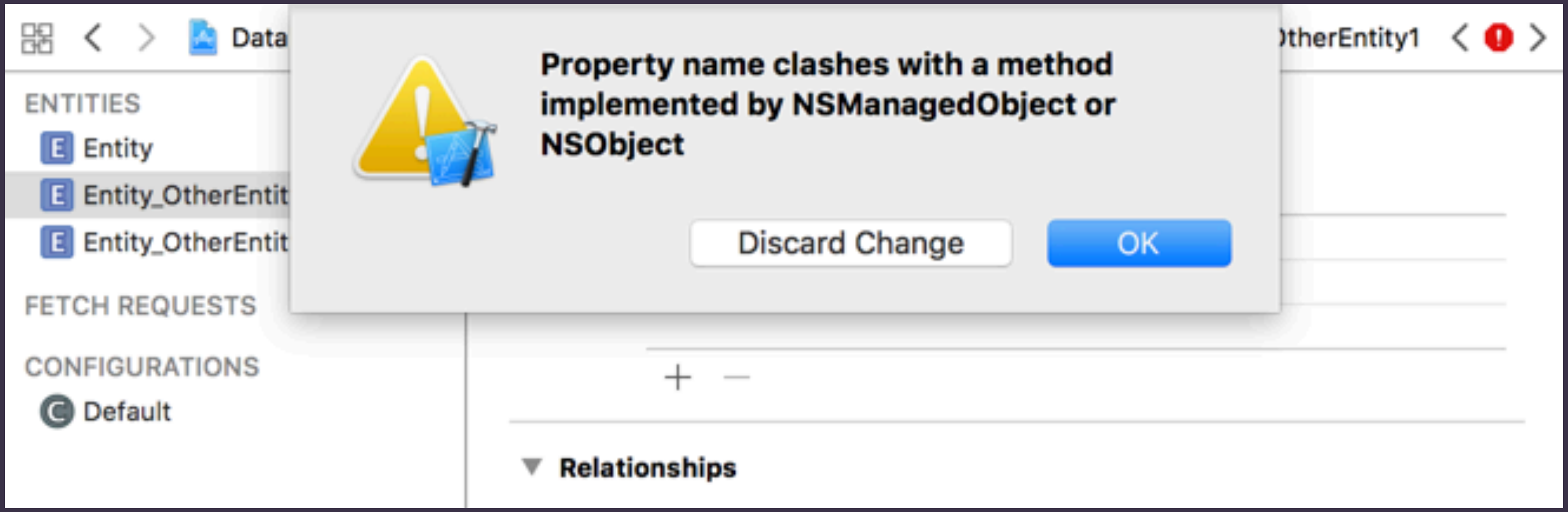
FETCH REQUESTS

CONFIGURATIONS
- Default

Attributes

Attribute	Type
archived	Boolean
count	Integer 32
created	Date
id	Binary Data
modified	Date
property1	String
property2	String
property3	String

x Old versions live in…


Property name clashes with a method implemented by NSManagedObject or NSObject

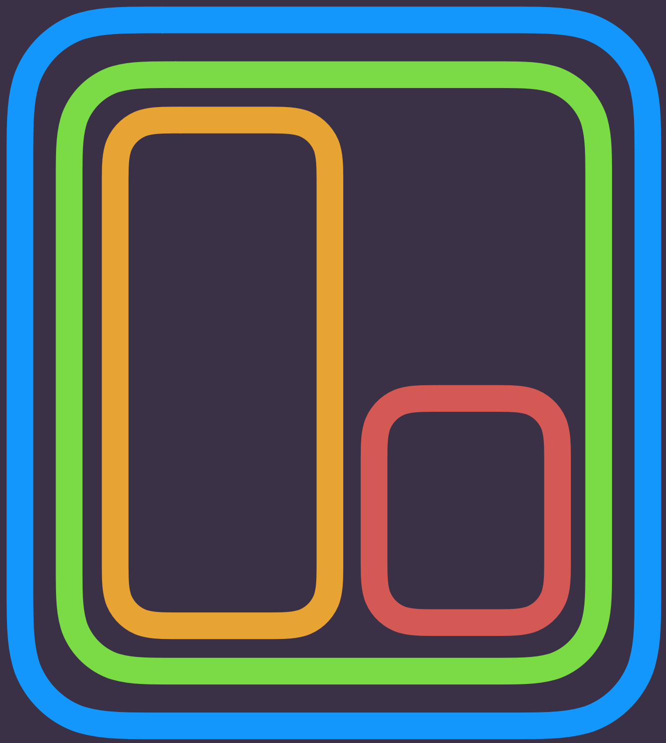
CoreData Evaluation

- Versioned NOT TESTABLE
- Easy to work with NO
- Support relationships with not-yet fetched objects YES

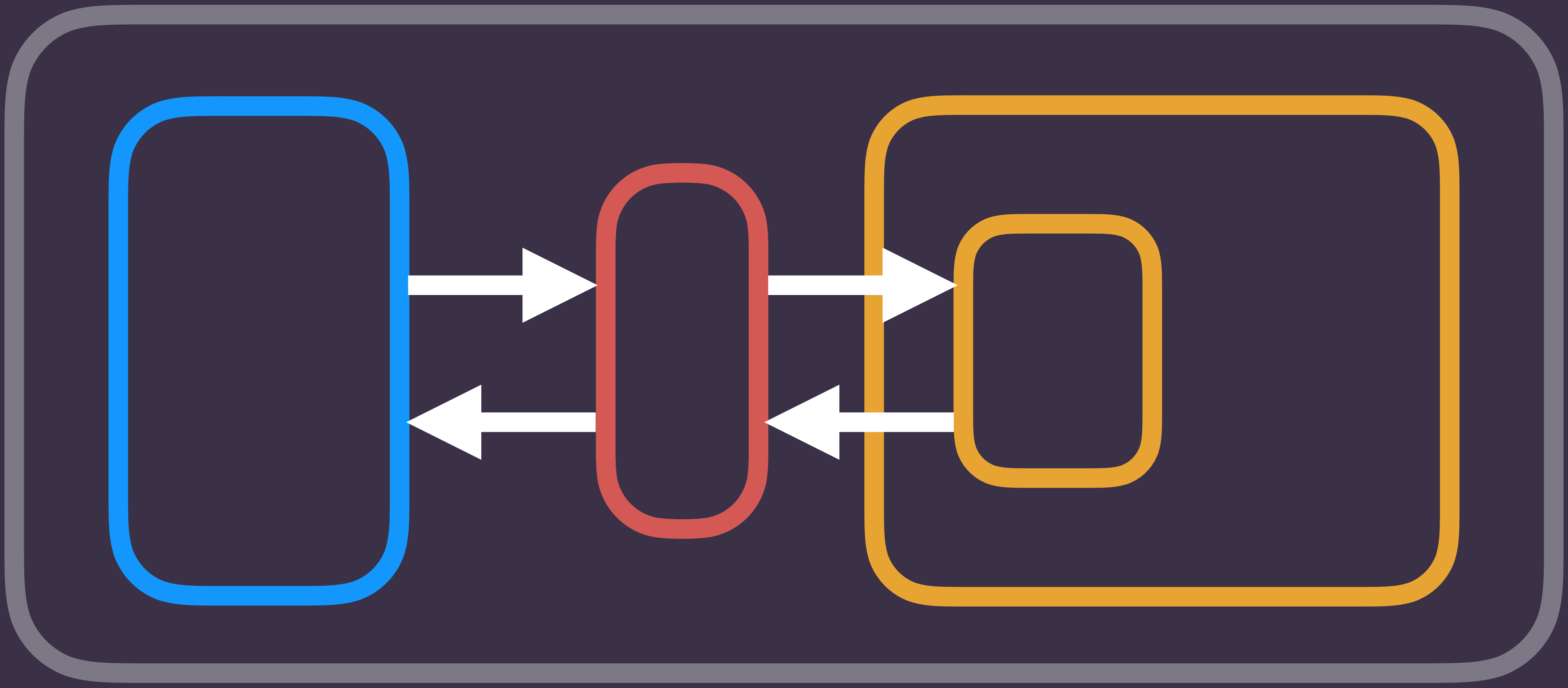
Our No-CoreData Stack™

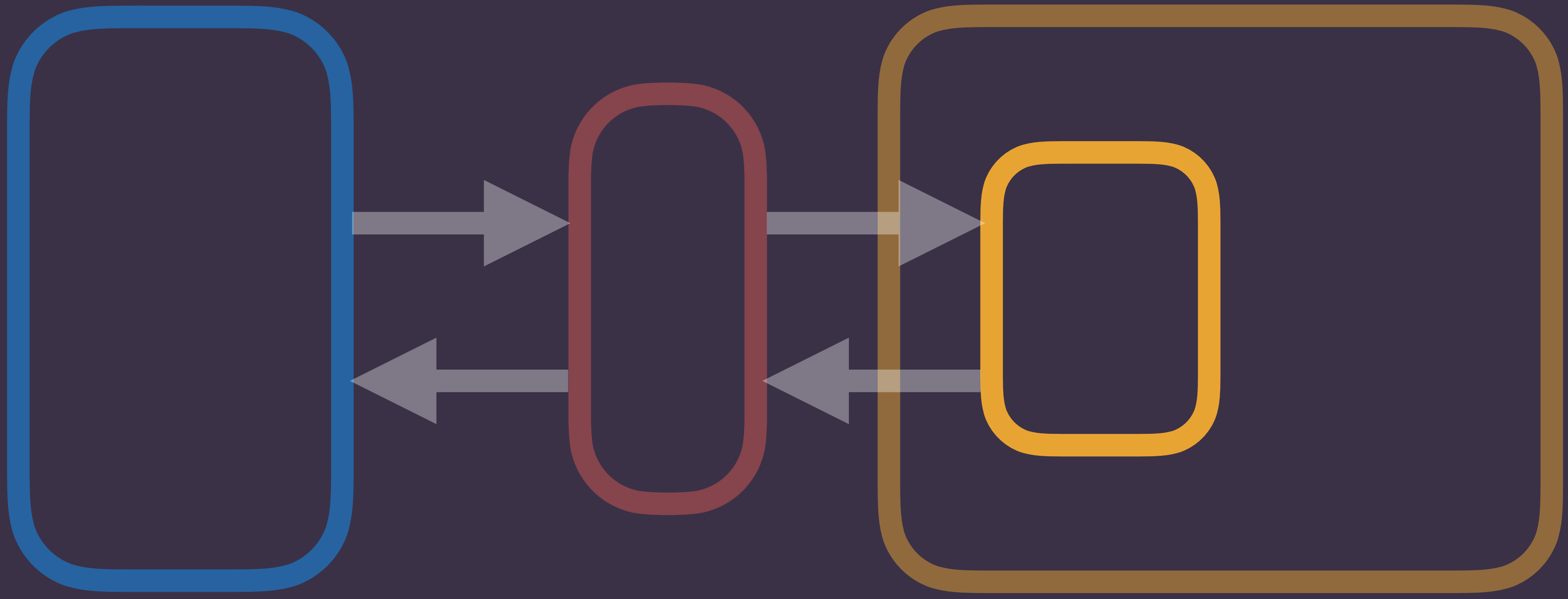
```
class SQLiteValueStore
    <ValueType,
     MapperType,
     EntityVersion>:
    ValueStoreType { ... }
```

Silo



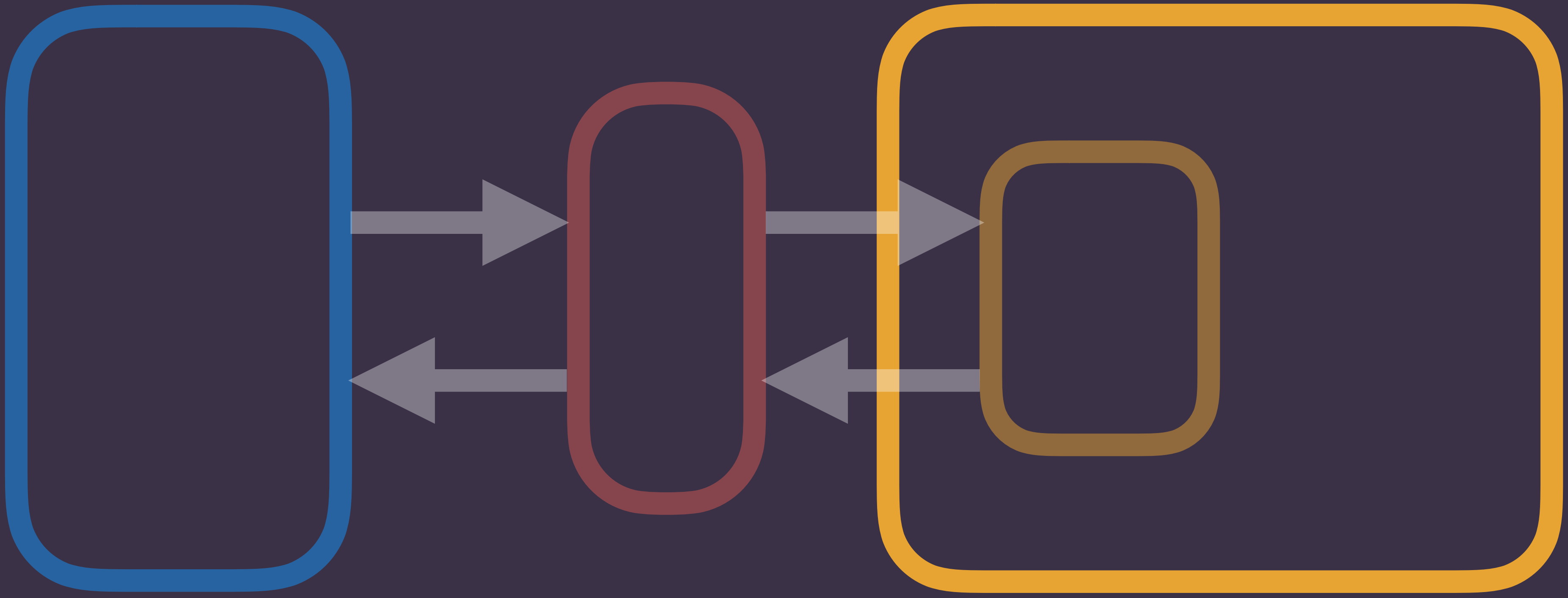
```
class SQLiteValueStore
    <ValueType, MapperType, EntityVersion>:
    ValueStoreType { ... }
```





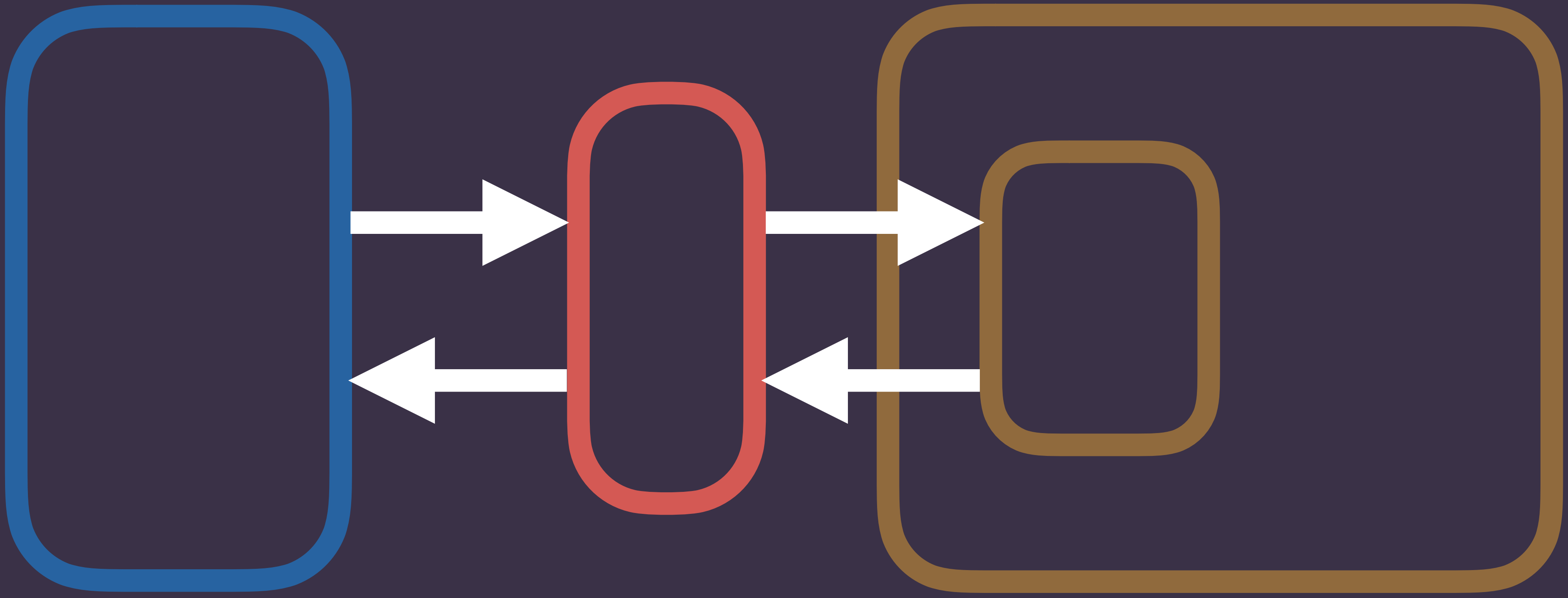
Canonical Representation:

Exact representation of Value to be persisted



EntityVersion:

Stores the canonical representation

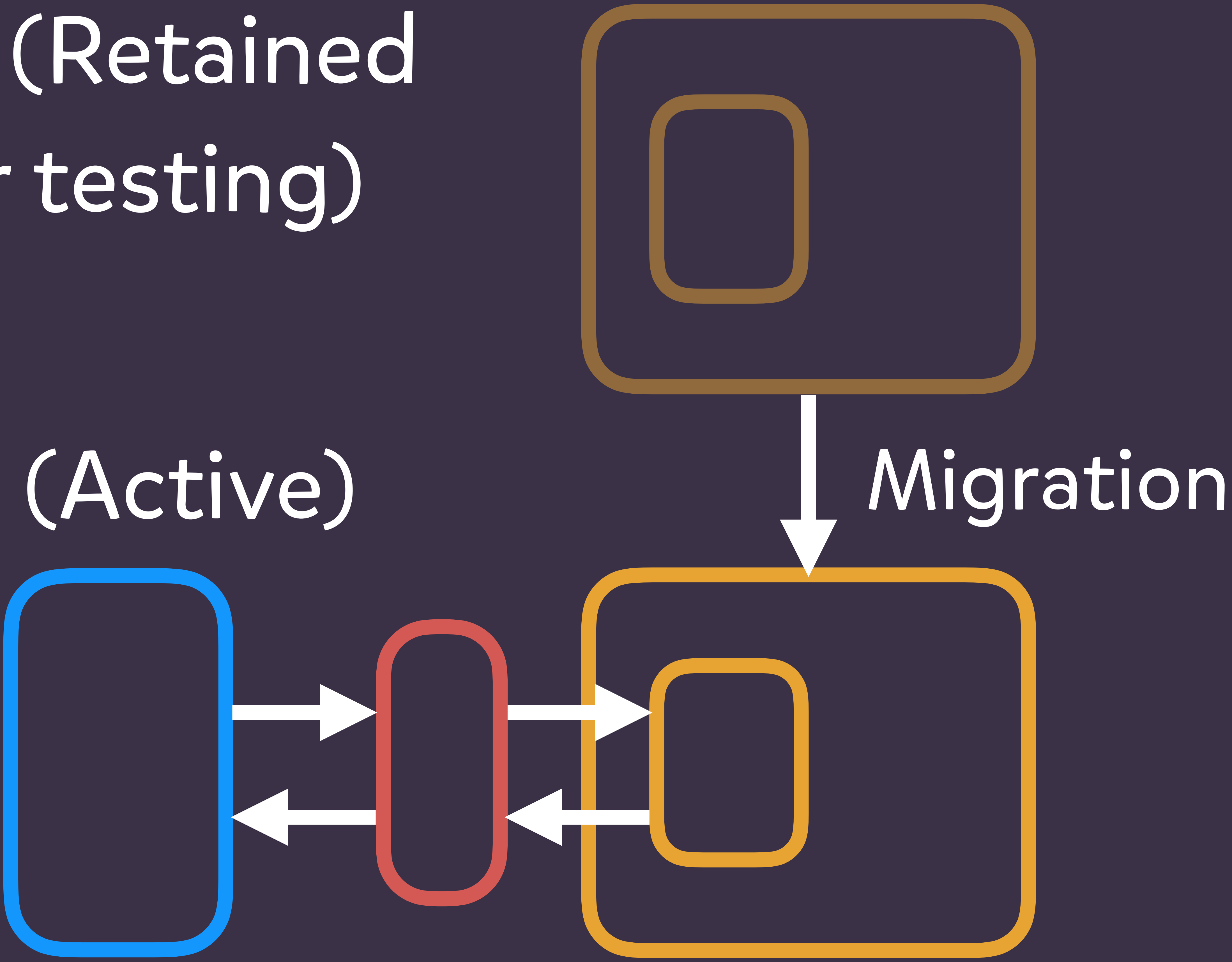


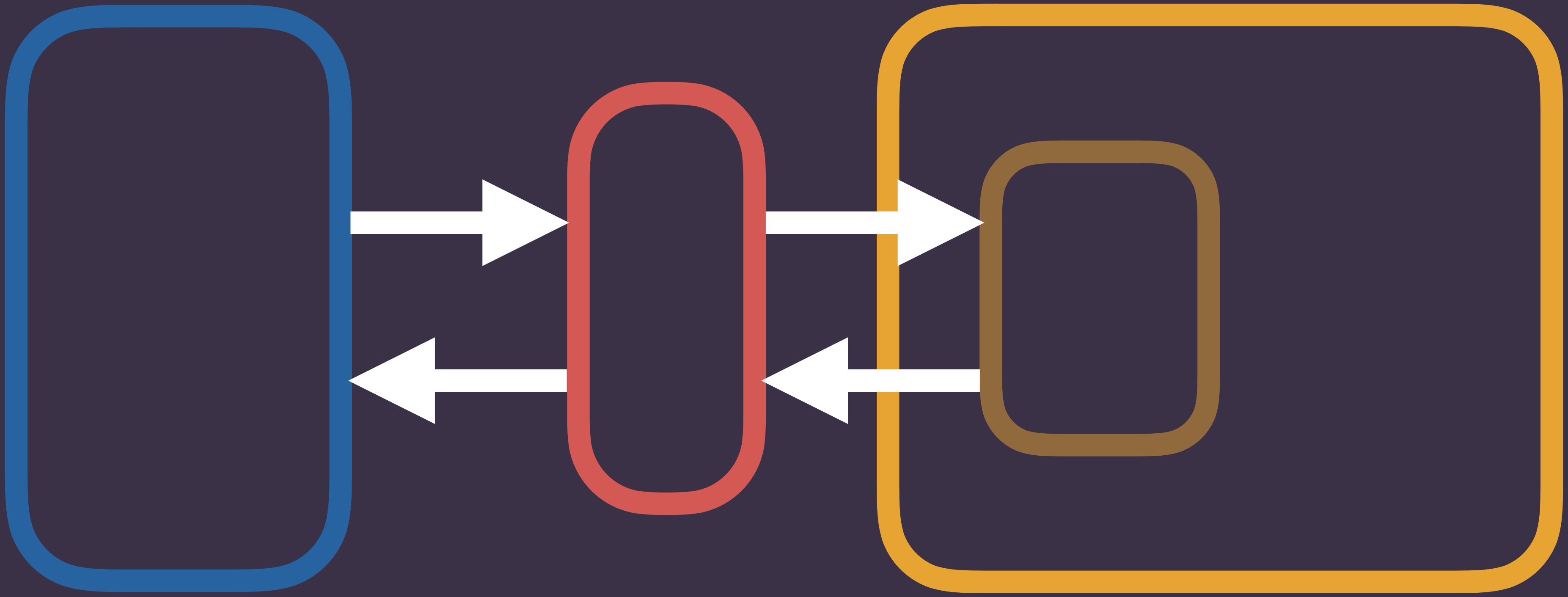
Mapper:

Maps the value type to the canonical representation

V1 (Retained for testing)

V2 (Active)





Mapper and EntityVersion

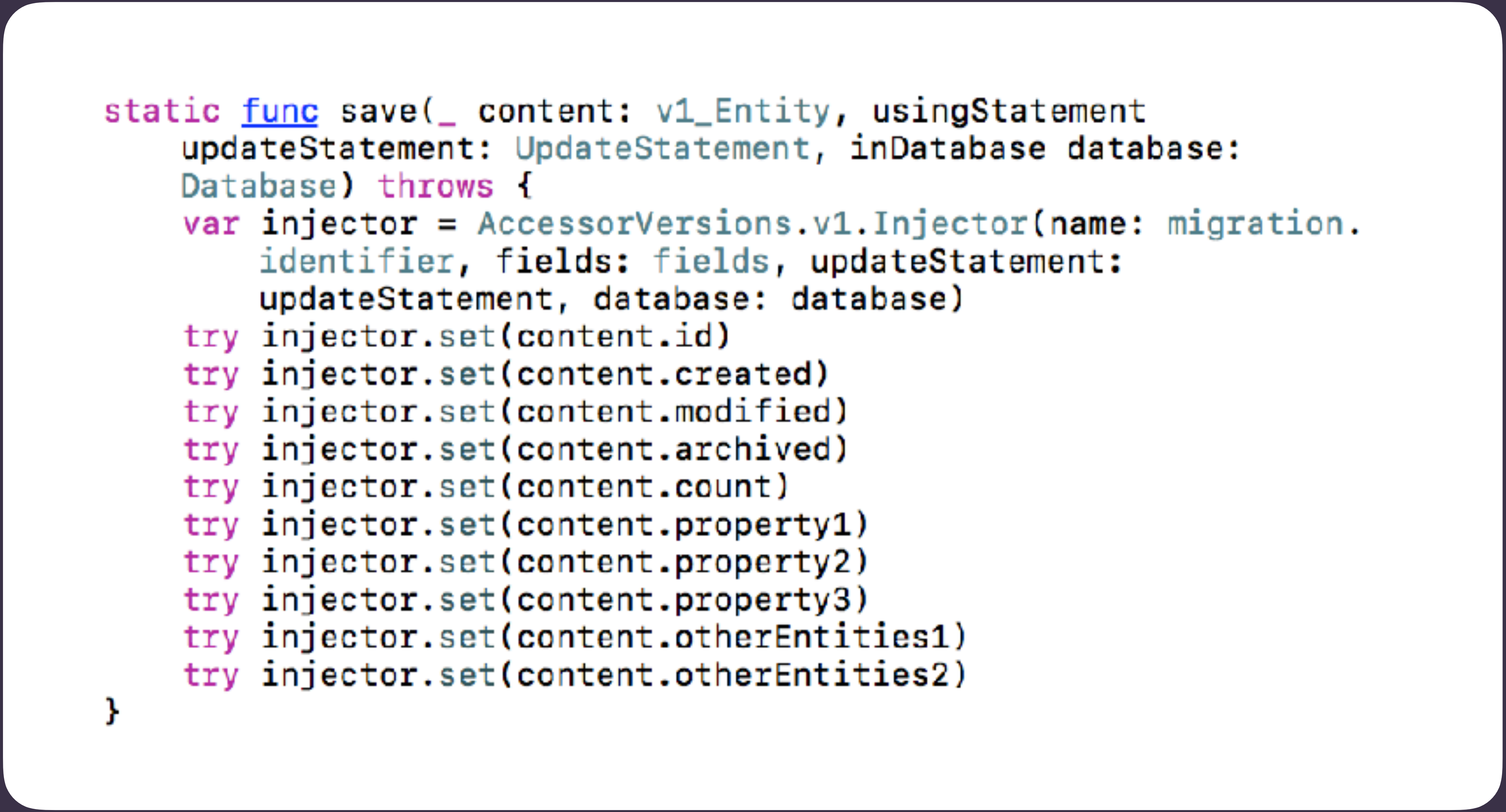
kinda suck to write

No-CoreData Evaluation

- Versioned YES
- Easy to work with STILL WRITING LOTS OF BOILERPLATE
- Support relationships with not-yet fetched objects YES

If writing boilerplate, at least make it easy boilerplate

→ Move complexity to smaller section



```
static func save(_ content: v1_Entity, usingStatement
    updateStatement: UpdateStatement, inDatabase database:
    Database) throws {
    var injector = AccessorVersions.v1.Injector(name: migration.
        identifier, fields: fields, updateStatement:
        updateStatement, database: database)
    try injector.set(content.id)
    try injector.set(content.created)
    try injector.set(content.modified)
    try injector.set(content.archived)
    try injector.set(content.count)
    try injector.set(content.property1)
    try injector.set(content.property2)
    try injector.set(content.property3)
    try injector.set(content.otherEntities1)
    try injector.set(content.otherEntities2)
}
```

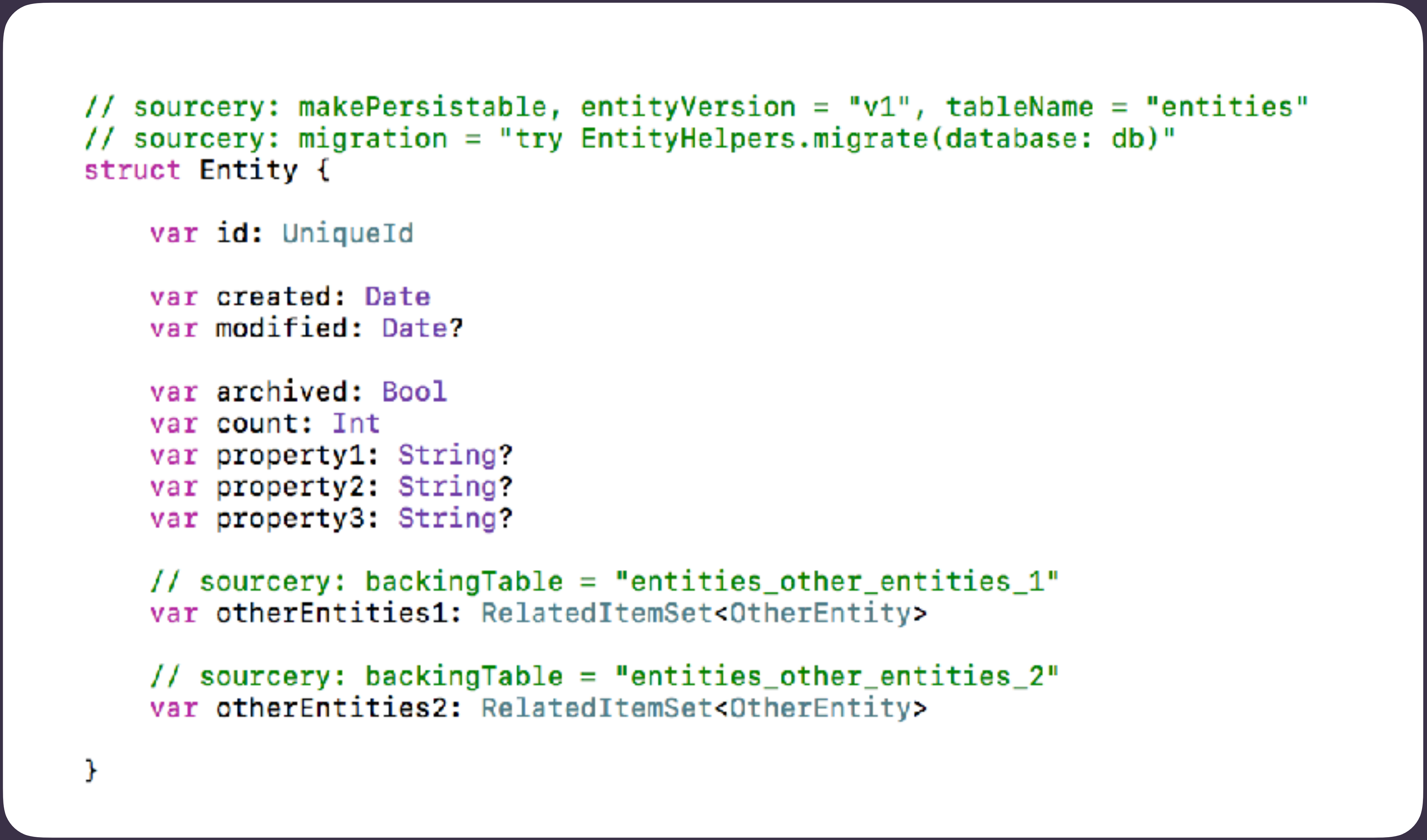
^ Dumb boilerplate

Introducing Sourcery

- Scans Swift code (using SourceKit)
- Uses templating to generate boilerplate

https://github.com/krzysztofzablocki/Sourcery

Persistable Model Entity



```swift
// sourcery: makePersistable, entityVersion = "v1", tableName = "entities"
// sourcery: migration = "try EntityHelpers.migrate(database: db)"
struct Entity {

    var id: UniqueId

    var created: Date
    var modified: Date?

    var archived: Bool
    var count: Int
    var property1: String?
    var property2: String?
    var property3: String?

    // sourcery: backingTable = "entities_other_entities_1"
    var otherEntities1: RelatedItemSet<OtherEntity>

    // sourcery: backingTable = "entities_other_entities_2"
    var otherEntities2: RelatedItemSet<OtherEntity>

}
```

Migration

```
enum EntityHelpers {

    static func migrate(database db: Database) throws {
        try db.execute("CREATE TABLE entities (id BLOB PRIMARY KEY,
            created INTEGER NOT NULL, modified INTEGER, archived INTEGER
            NOT NULL, count INTEGER NOT NULL, property1 TEXT, property2
            TEXT, property3 TEXT)")
        try db.execute(AccessorVersions.v1.
            unorderedRelatedItemTableDefinitionSQL(forTable:
            "entities_other_entities_1", referencing: "entities"))
        try db.execute(AccessorVersions.v1.
            unorderedRelatedItemTableDefinitionSQL(forTable:
            "entities_other_entities_2", referencing: "entities"))
    }

}
```

SQLite+SourceGen

- Versioned YES
- Easy to work with YES
- Support relationships with not-yet fetched objects YES

Surely you’ve sacrificed performance…

100 Objects

CoreData Backed:

Read 2ms / Write 15ms

SQLite Backed:

Read 22ms / Write 5ms

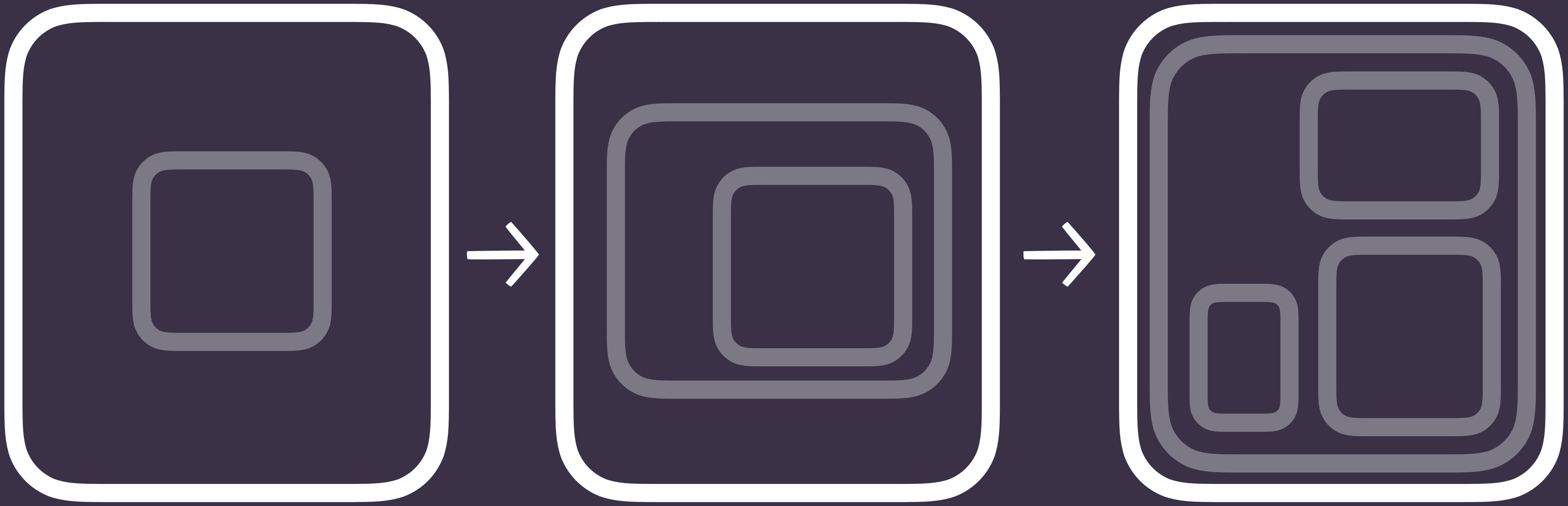
1000 Objects

CoreData Backed:

Read 113ms / Write 10,277ms

SQLite Backed:

Read 23ms / Write 6ms



What’s shipped?

- Upload Queue using SQLite implementation (woo! ACID)
- Caching some entities InMemory

What’s next?

- Moving to caching all in-memory in preparation for true offline
- Working on Caching and deletion (including ‘managed’ objects)

We talked about…

- Shipping code
- Building a bespoke persistence layer
- Engineering for developer happiness (among other things)

We talked about…

- Shipping code
- Building a bespoke persistence layer
- Engineering for developer happiness (among other things)

Ask me anything

@javn.wd

REDEYE