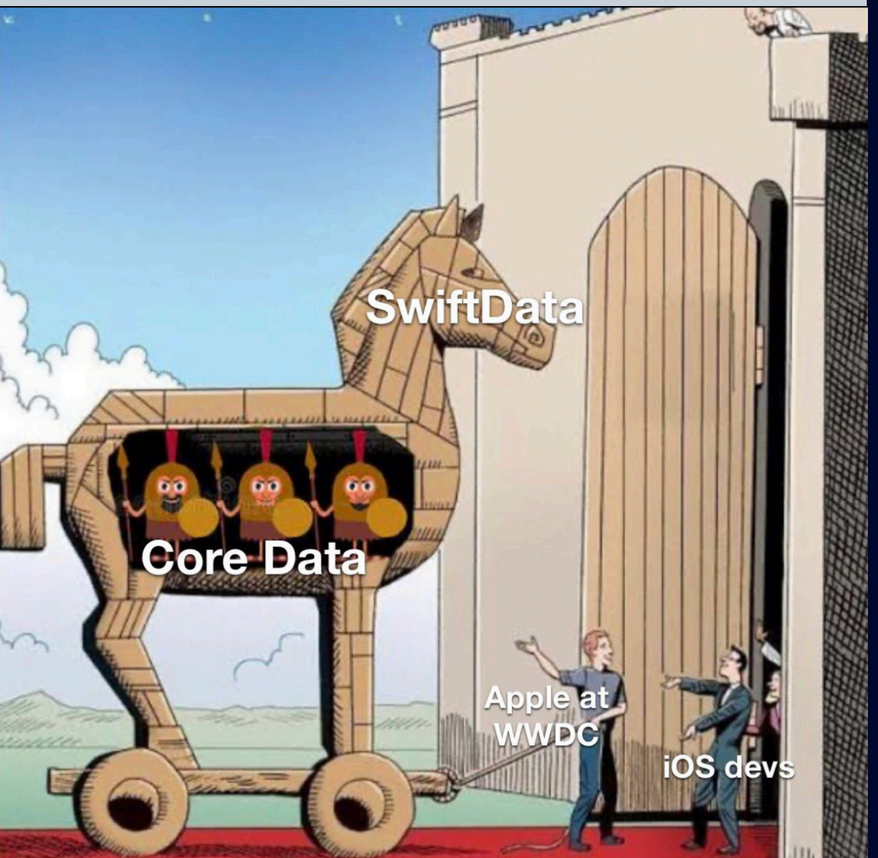
ТИНЬКОФФ

Кто ты, ~~воин~~ SwiftData

Сегодня



- Основные концепции SwiftData
- Отличие от CoreData и миграция
- Детали реализации SwiftData
- Итоги

SwiftData - @Model

- Определяем схему Entity прямо в коде
- При помощи макроса добавляем SwiftData функциональность
- Связи устанавливаются из кода с атрибутами

```
@Model
final class Post {

    @Attribute(.unique)
    var id: String
    var message: String
    var timestamp: Date

    @Relationship(deleteRule: .cascade, inverse: \Snippet.folder)
    var attachments: [Attachment]
    var author: User
    @Transient
    var isDraft: Bool = false
}
```

SwiftData - ModelContainer

Контейнер настраивает хранилище

```
let container = try ModelContainer(
    for: Post.self,
    migrationPlan: MobiusMigrationPlan.self,
    configurations: .init(isStoredInMemoryOnly: true)
)
```

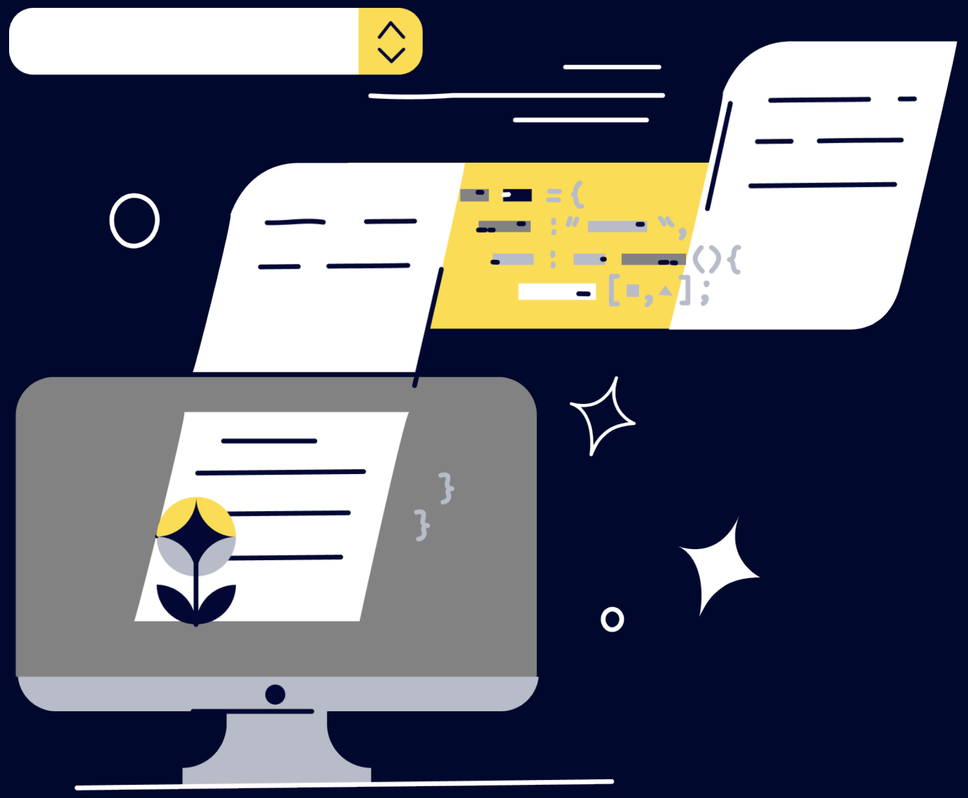
SwiftData - ModelContainer & ModelContext

- Контейнер настраивает хранилище
- Контекст привязан к @MainActor
- Через контекст добавляются и удаляются объекты

```
let container = try ModelContainer(for: Post.self)
let context = container.mainContext
let newContext = ModelContext(container)
newContext.autosaveEnabled = false

let post = Post(message: "Привет!")
newContext.insert(post)
try newContext.save()
```

ModelContext



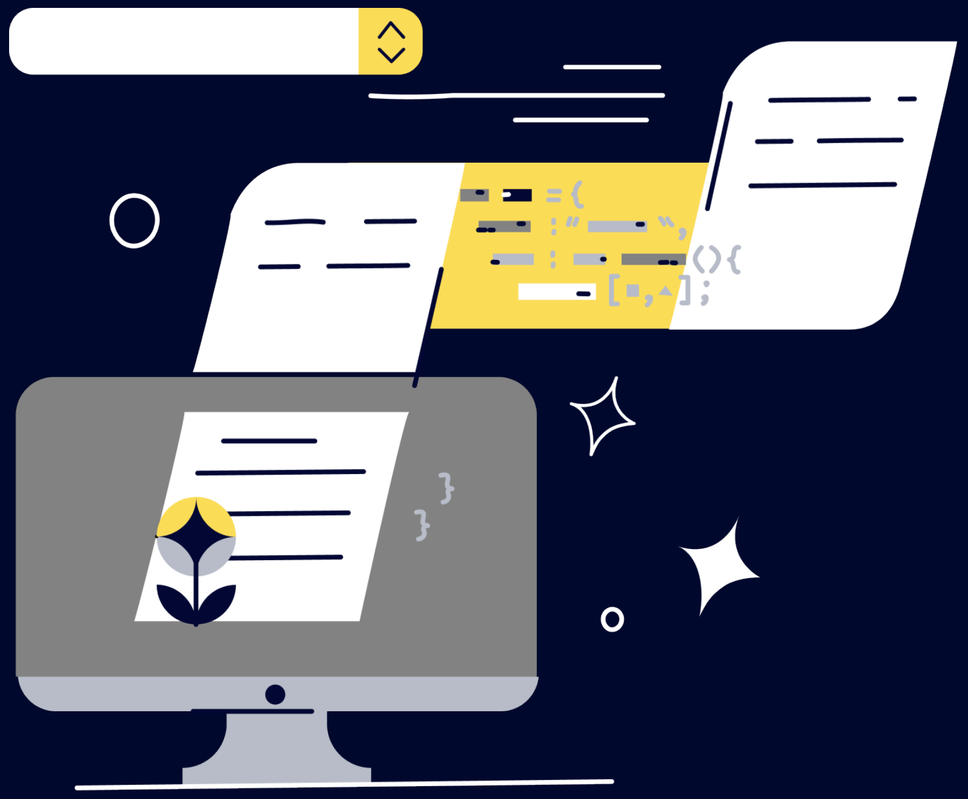
- ModelContext - Хранит слепок данных запрошенных из хранилища
- Следит за изменениями в хранилище по тем объектами которые используются
- Пробрасывает изменения в ModelContainer при изменении
- Позволяет откатить последние изменения. undo/redo и autosave

SwiftData - Predicate & FetchDescriptor

- Лимиты в выборке
- Оффсеет

```
let today = Date()
let kittyFeed = #Predicate<Post> {
    $0.timestamp < today &&
    $0.message.contains("котики")
}
var descriptor = FetchDescriptor<Post>(
    predicate: kittyFeed,
    sortBy: [SortDescriptor(\.timestamp)]
)
descriptor.fetchLimit = 40
descriptor.fetchOffset = 20
let posts = try newContext.fetch(descriptor)
```

Predicate & FetchDescriptor



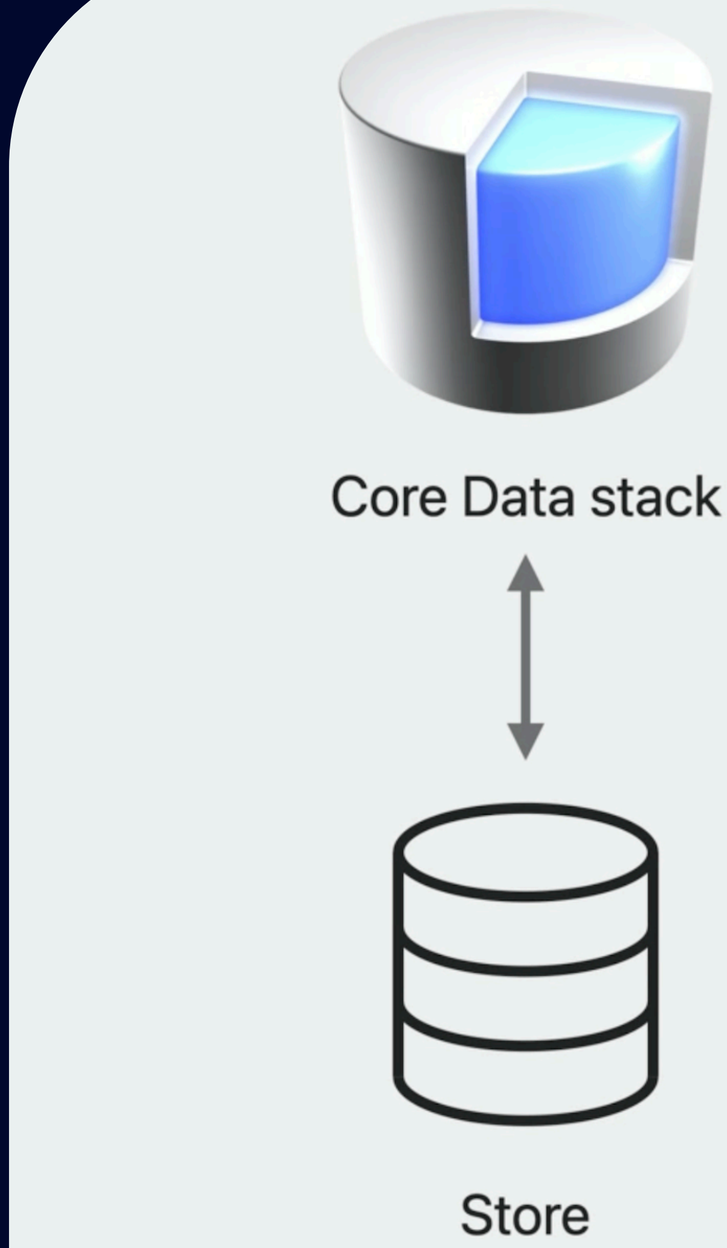
- Предикаты поддерживают подзапросы с Join-ми
- Предикаты транслируются в SQL запросы к базе
- Compile-time валидация запроса
- Доступны параметры offset/limit, faulting/prefetching

SwiftData Основные концепции

- Code-first
- Макросы
- CRUD
- Swift Concurrency

Core Data stack

- Один поток-один контекст
- In-memory/Sqlite/XML
- Objective-C



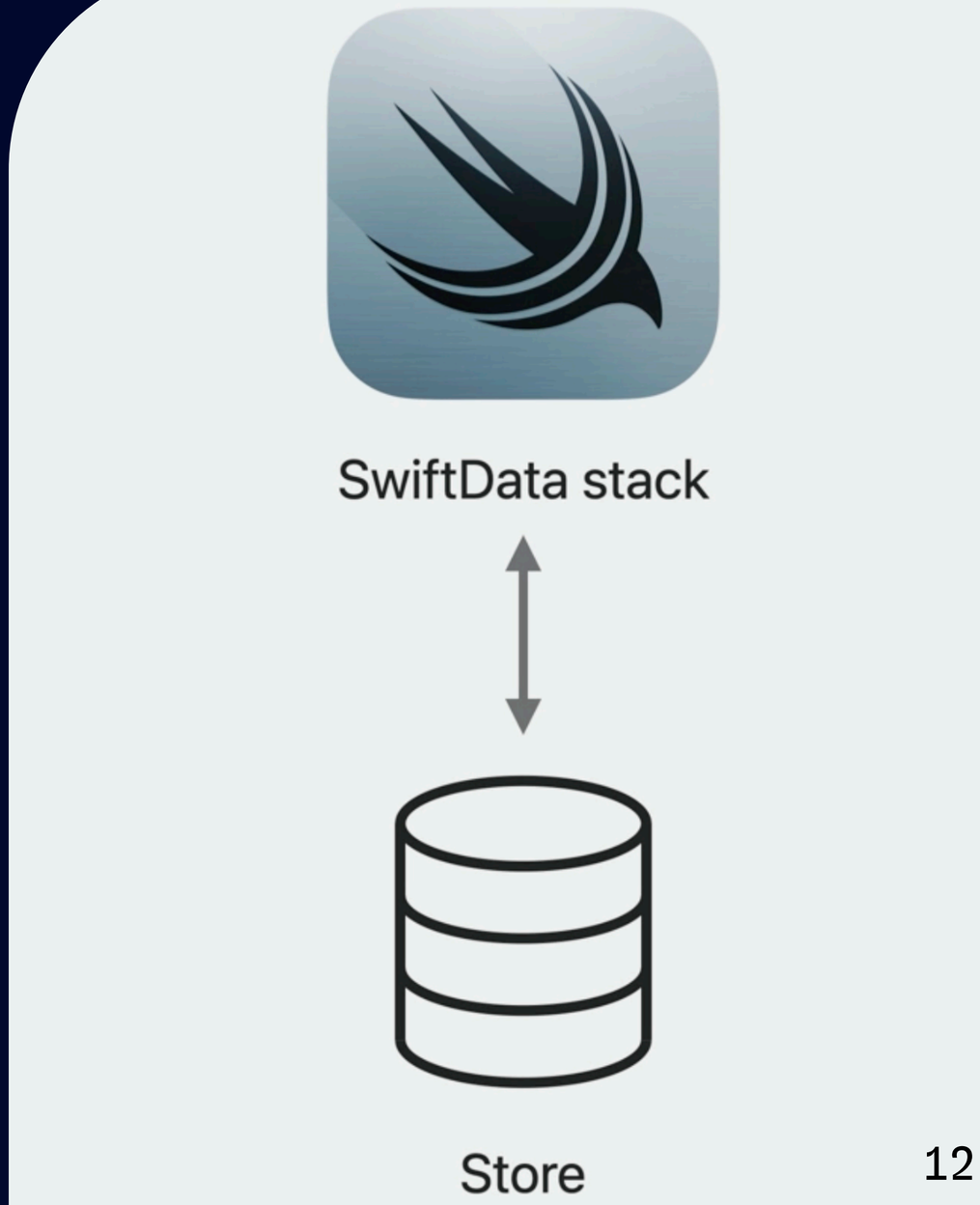
Многопоточная CoreData

- Один поток один контекст
- Передача объектов между потоками через objectId

```
func update(post: CoreData.Post, message: String) async throws {
    guard let context = post.managedObjectContext else {
        throw CoreData.CoreError.noContext
    }
    try await context.perform {
        post.message = message
        try context.save()
    }
}
```

SwiftData stack

- mainContext = @MainActor
- In-memory/Sqlite
- Swift



Многопоточная SwiftData

- Контекст привязан к очереди
- Несколько контекстов на одной очереди

```
Task.detached {
    let privateContext = ModelContext(container)
}
```

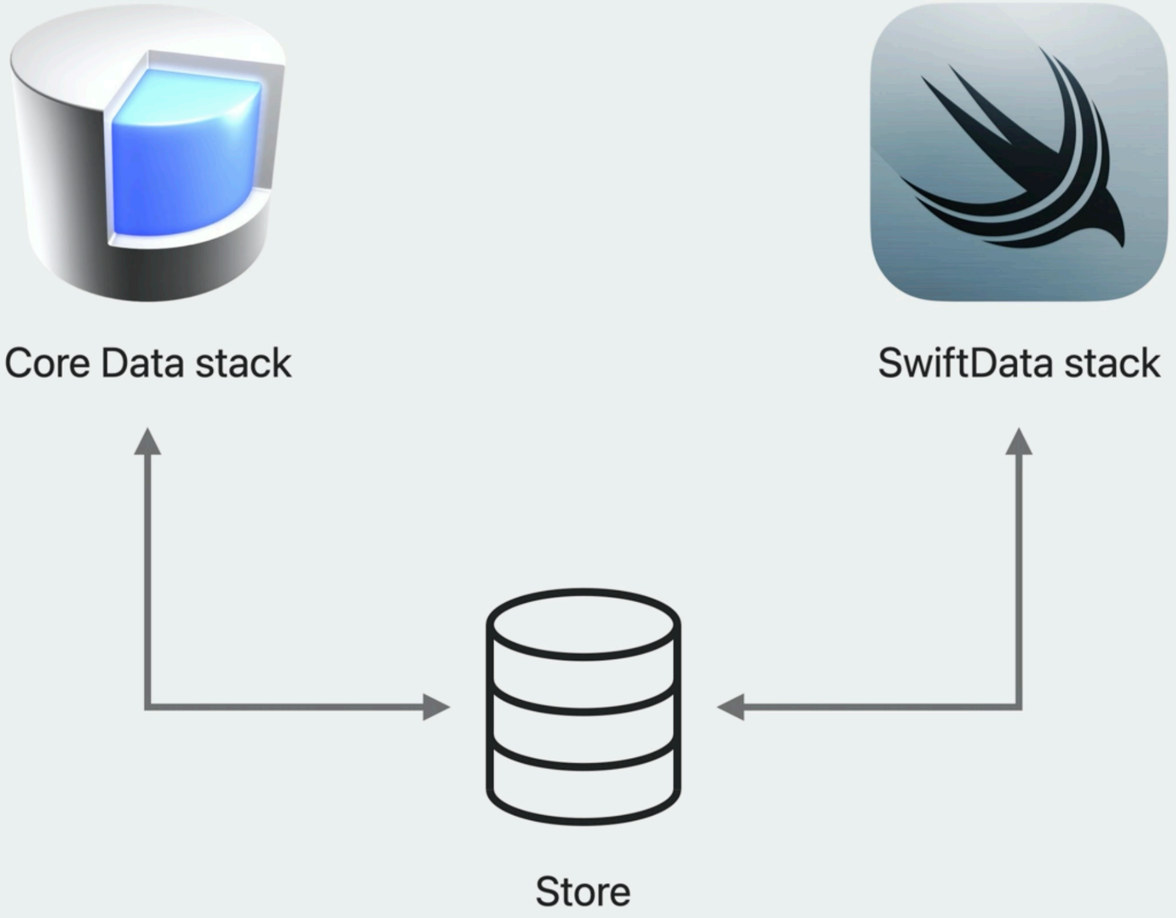
Многопоточная SwiftData

- Акторы
- Макрос

```
@ModelActor
actor MobiusModelActor {
    let modelExecutor: any ModelExecutor
    let modelContainer: ModelContainer
    init(modelContainer: ModelContainer) {
        let modelContext = ModelContext(modelContainer)
        modelExecutor = DefaultSerialModelExecutor(modelContext: modelContext)
        self.modelContainer = modelContainer
    }
}
```

Core Data + SwiftData

- Единый стор
- Герерация SwiftData по xcdatamodeld
- Разные имена классов
- iOS 17+

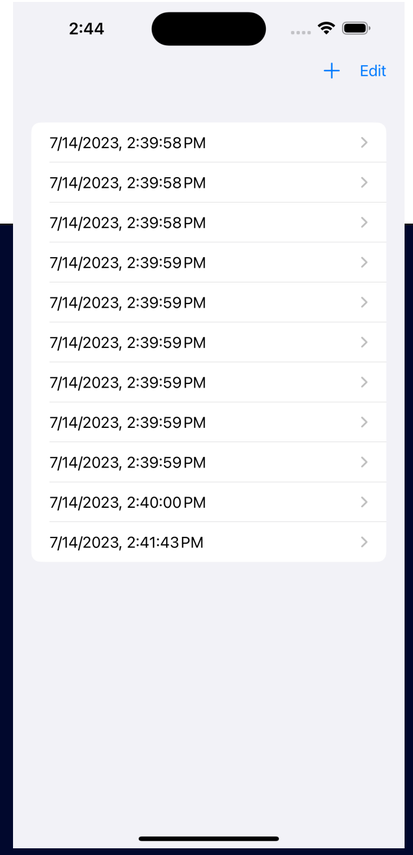


CoreData vs SwiftData

CoreData	SwiftData
NSManagedModelObject	@Model
NSPersistentContainer	ModelContainer
NSManagedModelContext	ModelContext
NSPredicate	#Predicate
NSFetchRequest	FetchDescriptor
NSSortDescriptor	
NSFetchedResultsController	@Query

Погружаемся в детали SwiftData

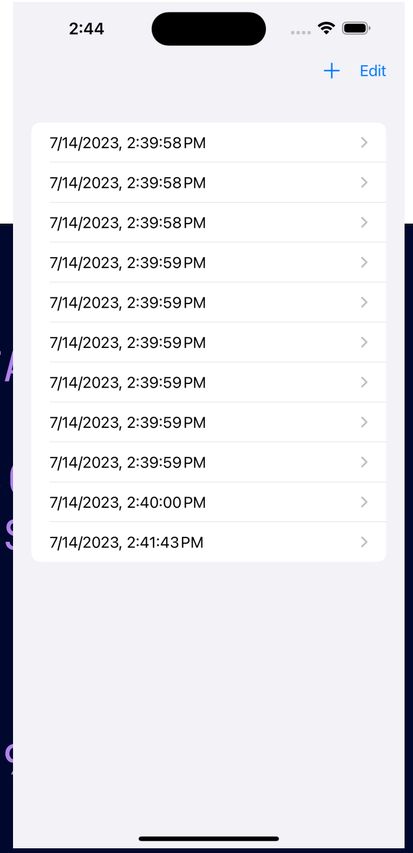
-com.apple.**CoreData**.SQLDebug 1




```
@Model
final class MobiusEntity1 {
    var timestamp: Date
    var oneToOne: MobiusEntity2
    var oneToMany: [MobiusEntity3]
    init(timestamp: Date) {
        self.timestamp = timestamp
        self.oneToOne = MobiusEntity2(timestamp: timestamp)
        self.oneToMany = [MobiusEntity3(timestamp: timestamp)]
    }
}
```

Погружаемся в детали SwiftData

-com.apple.**CoreData**.SQLDebug 1



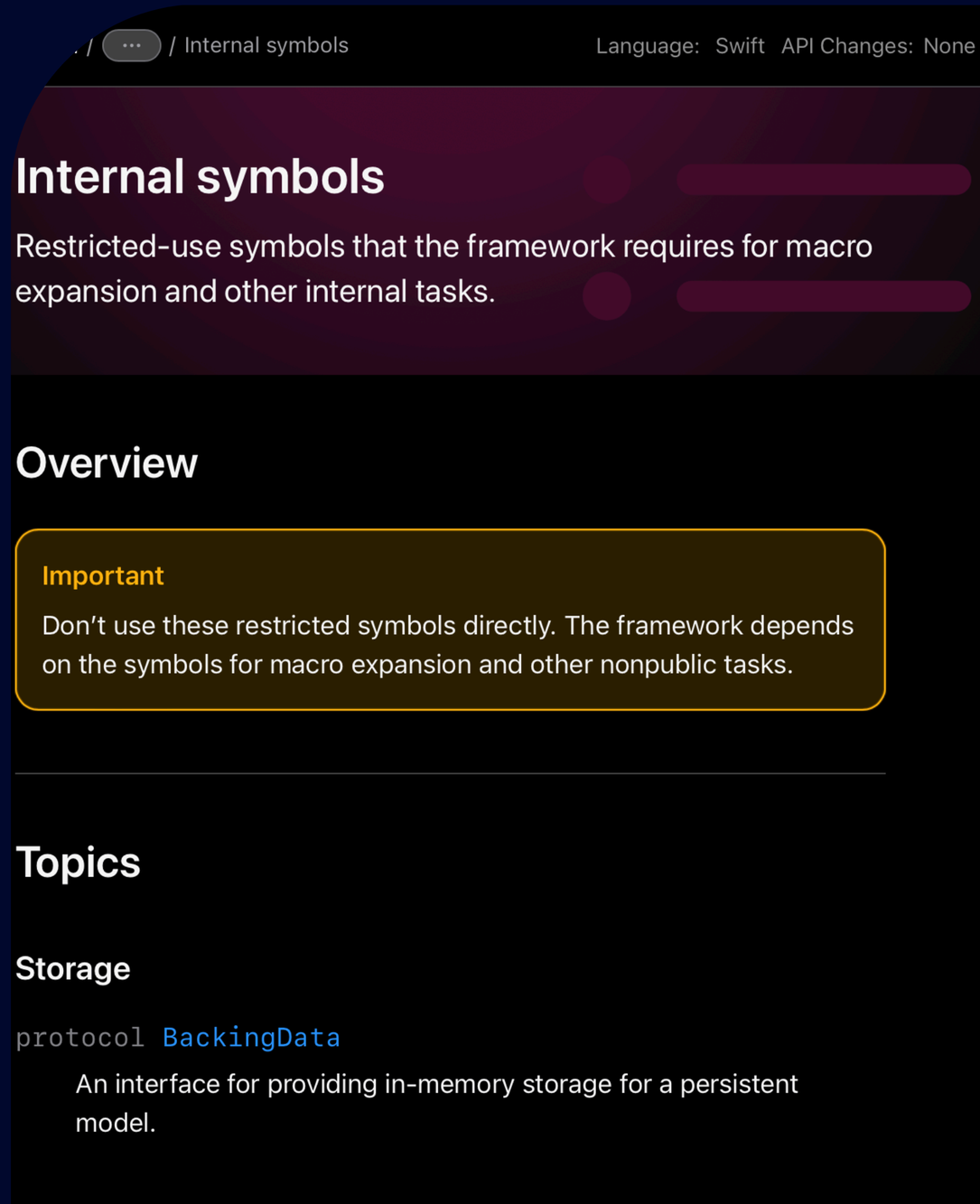

```
CoreData: sql: BEGIN EXCLUSIVE
CoreData: sql: INSERT INTO ZMOBIUSENTITY1(Z_PK, Z_ENT, Z_OPT, ZONETOONE, ZTIMESTA
VALUES(?, ?, ?, ?, ?)
CoreData: sql: INSERT INTO ZMOBIUSENTITY2(Z_PK, Z_ENT, Z_OPT, ZTIMESTAMP) VALUES(
CoreData: sql: INSERT INTO ZMOBIUSENTITY3(Z_PK, Z_ENT, Z_OPT, Z1ONETOMANY, ZTIMES                    ?,
?, ?, ?)
CoreData: sql: COMMIT
CoreData: debug: Remote Change Notification - Posting for
com.apple.coredata.NSPersistentStoreRemoteChangeNotification.remotenotification.9
C9A8-4268-9627-2A347F0A2C5B
CoreData: sql: SELECT 0, t0.Z_PK, t0.Z_OPT, t0.ZTIMESTAMP, t0.ZONETOONE FROM ZMOBIUSENTITY1 t0
LIMIT 4294967295
CoreData: annotation: sql connection fetch time: 0.0000s
CoreData: annotation: total fetch execution time: 0.0001s for 11 rows.
```

SQLite

- Написана на C
- Покрыта тестами
- Public domain

Как они это сделали?

- BackingData
- _DefaultBackingData
- Макросы



Погружаемся в детали SwiftData

```
public extension BackingData {
    var managedObject: NSManagedObject? {
        guard let object = getMirrorChildValue(
            of: self,
            childName: "_managedObject") as? NSManagedObject else {
            return nil
        }
        return object
    }
}
```

Погружаемся в детали SwiftData

```
let post = Post(entity: Post.entity(), insertInto: nil)
```

```
let post = Post(message: "Mobius 2023 Autumn")
container.mainContext.insert(post)
if let managedObject = managedObject as? (any BackingData<Post>) {
    Post(backingData: managedObject)
}
```

Что мы поняли?

- CoreData это текущая реализация BackingData
- Макросы позволяют менять реализацию
- Ключевое отличие - работа с контекстом
- Мы все еще не можем использовать SwiftData до iOS 17

Используем SwiftData до iOS 17!

- Code-first
- Макросы
- CRUD
- Swift Concurrency

CoreStore - CoreStoreObject

Определяем схему Entity
прямо в коде

```
final class Post: CoreStoreObject {
   @Field.Stored(“latitude")
   var id: String
   @Field.Stored(“latitude")
   var message: String

   @Field.Relationship(“attachments")
   var attachments: [Attachment]
   @Field.Relationship("author")
   var author: User
}
```

CoreStore - DataStack

Контейнер настраивает хранилище

```
let dataStack = DataStack(
    CoreStoreSchema(
        entities: [
            Entity<Post>("Post"),
            Entity<Attachment>("Attachment"),
            Entity<User>("User")
        ]
    )
)
```

CoreStore - Fetching

- Привычный синтаксис
- Макросы

```
try! dataStack.fetchAll(
    From<Post>()
        .where(\.$isFavourite == true && \.$publishDate > .now )
        .orderBy(.ascending(\.$publishDate))
)
```

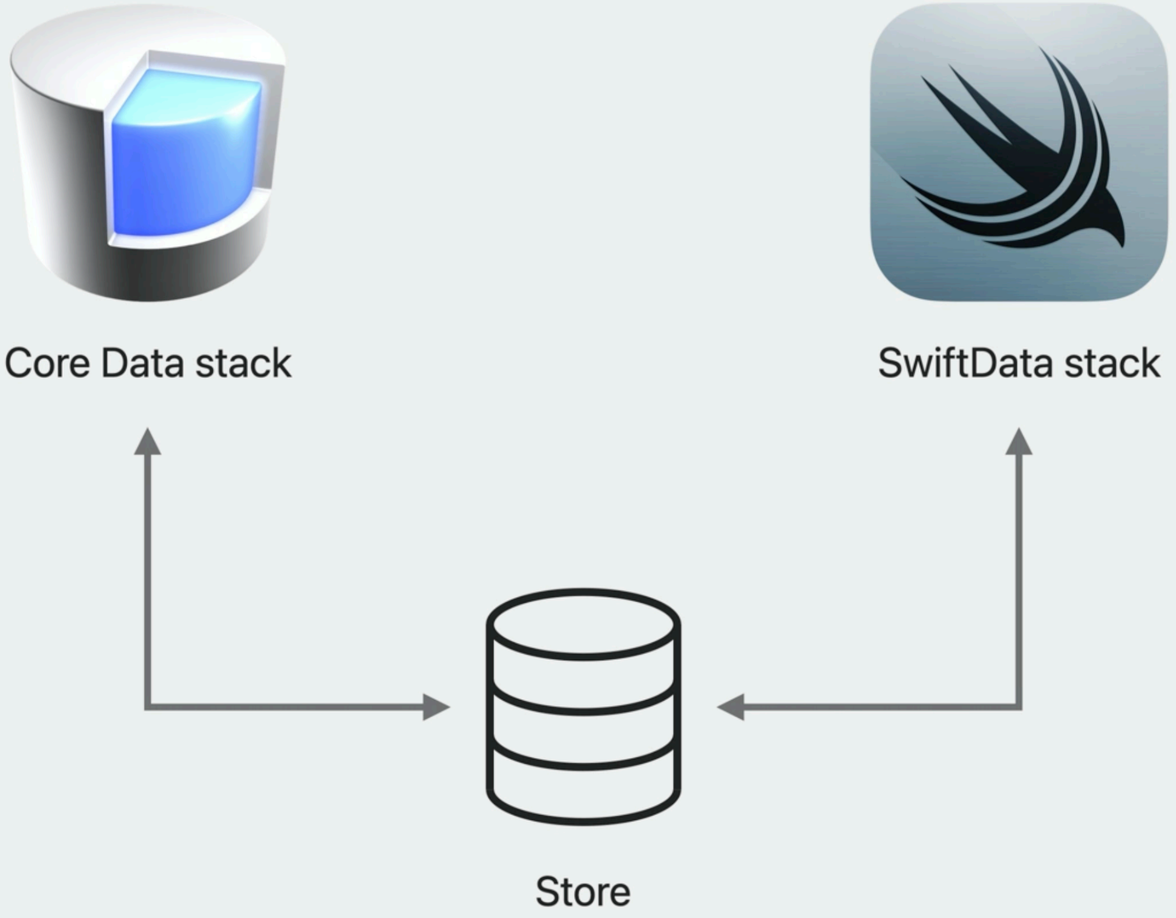
Многопоточная CoreData-Swift 5.9

- Акторы
- Макросы

```
@ModelActor
actor CoreDataPerformer {}
extension CoreDataPerformer {
    func update(post: Post, message: String) throws {
        post.message = message
        try modelContext.save()
    }
}
```

Готово!

- Code-first
- Макросы
- Swift Concurrency



Итоги

Обратная совместимость

Можно использовать один стор и реализовывать новые фичи со SwiftData

Автосохранение

Ускоряет разработку и проектирование

Удобные миграции

SchemeMigration в коде, работает автоматически с большинством кейсов

Swift Concurrency + Макросы

Избавляемся от когнитивной нагрузки и спагетти-кода

ТИНЬКОФФ

@IOS_BROADCAST

Андрей Зонов | Ведущий разработчик

linkedin.com/in/avzonov/

t.me/avzonov

ТИНЬКОФФ