Core Data

IZA, Martin Hrubý, FIT VUT, 2019

Úvod

- Přehled možností ukládat data v aplikaci.
  - Prop. soubory, UserDefaults, dokumenty, objektové kontejn.
  - Kdy se hodí dokumenty, kdy se hodí DB.
- **Objektový kontejner** — správa persistentních objektů.
  - Třída/entita.
  - Reference mezi objekty.
  - Technická implementace — CD (Sqlite3, XML, …), CloudKit.

Co je CoreData (CD)

- **Objektový kontejner**.
- Rozhraní k uložišti persistentních objektů:
  - Vkládání a rušení objektů.
  - Správa vazeb mezi objekty.
  - Dotazy (fetch).
- CD je z Foundation, tj. přenositelnost kódu na macOS (až na FRC).

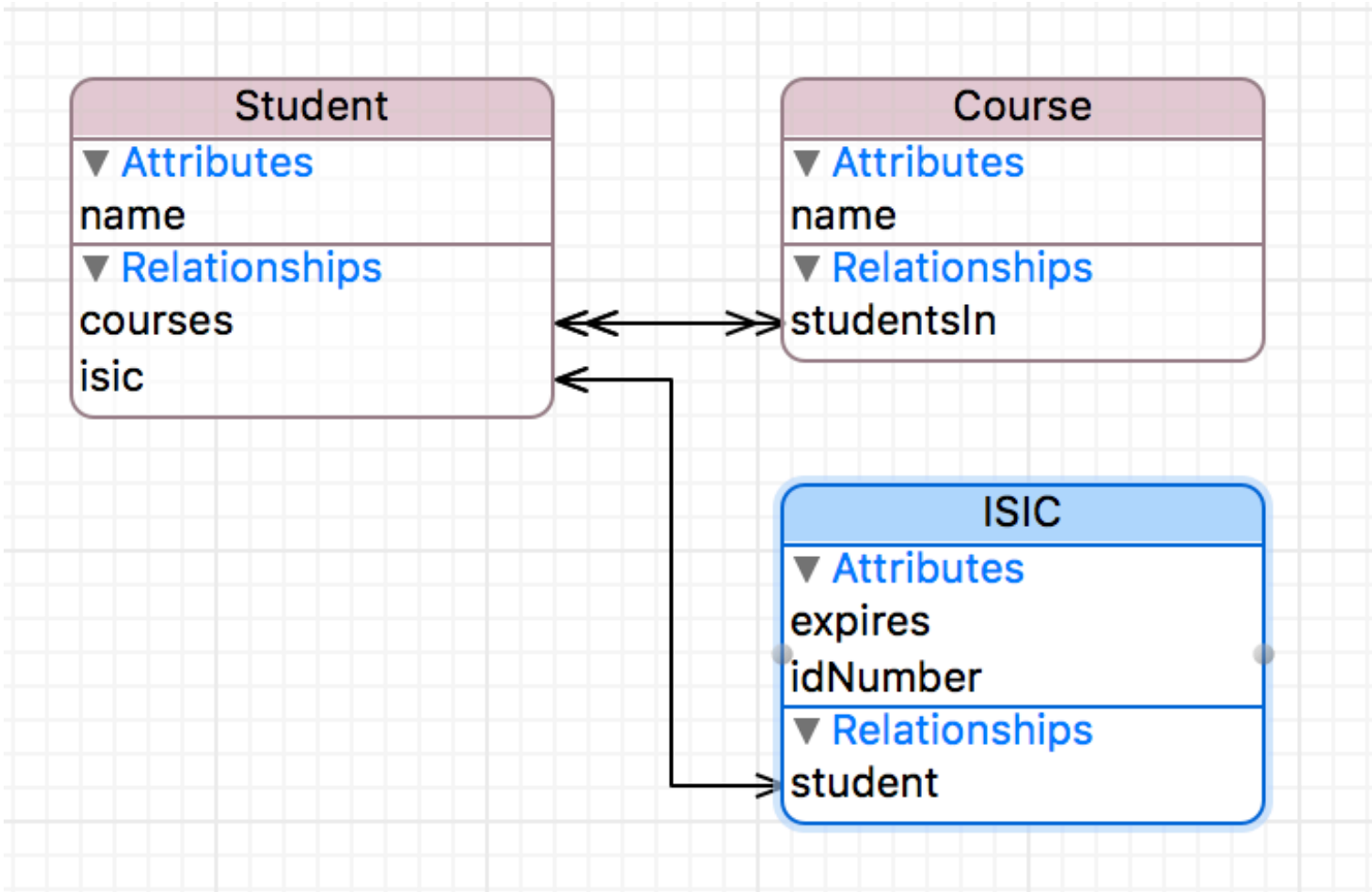
CoreData versus Rel DB

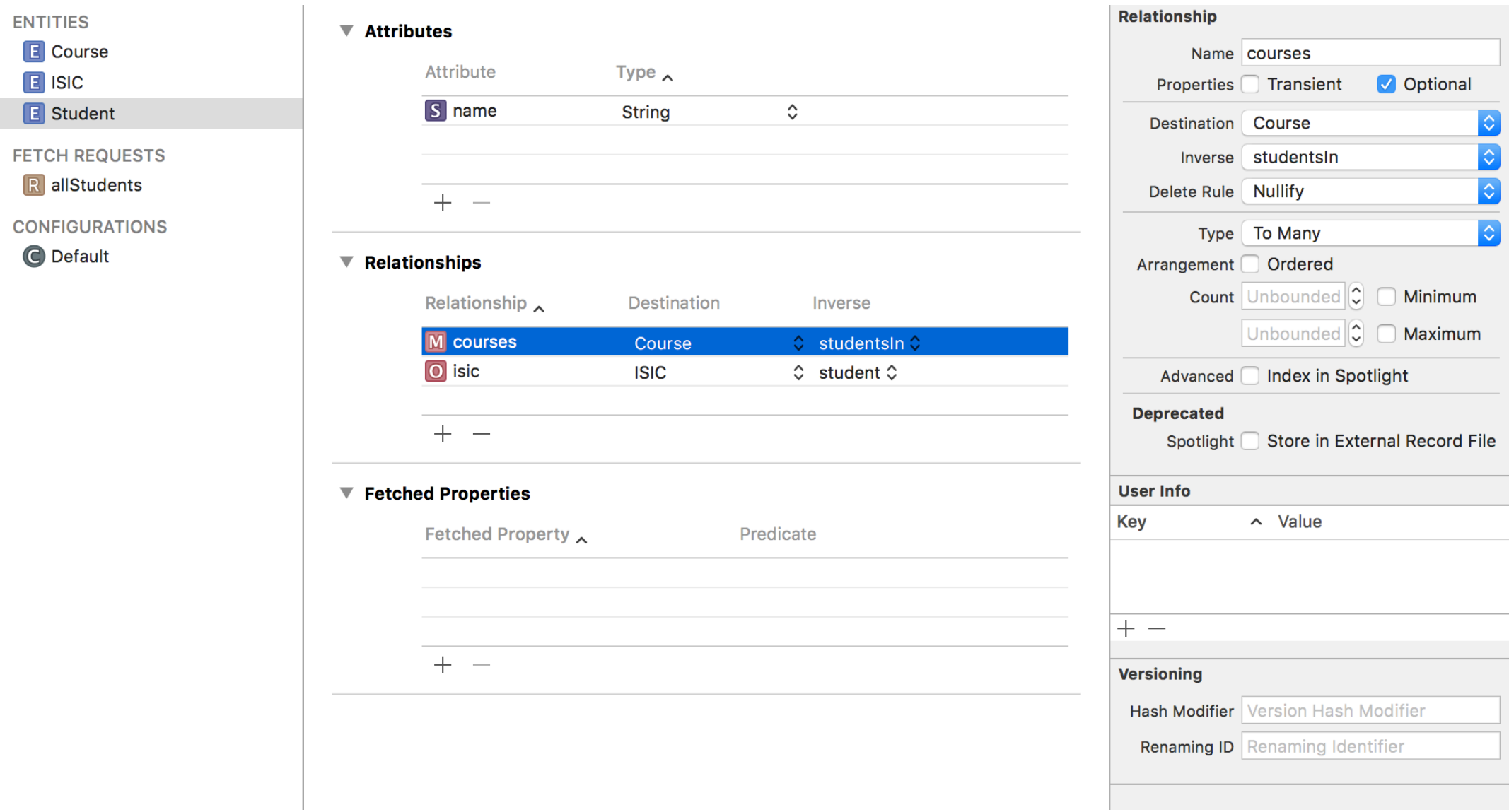
- CD nesplňuje typický relační datový model:
  - Projekce — nelze provést SELECT nad vybranými sloupci.
  - Join — nelze sloučit dvě relace.
- CD je pouze sada persistentních polí typu [ManagedObject].
  - Selekce, uspořádání, počet prvků, vkládání, rušení.

Atributy Entity

- Uložené atributy — název, datový typ.
- Relationships — název, typ (1:1, 1:N), cíl, *inverzní vztah*.
- Fetched Properties — ????.
- Abstraktní entita — nemá uloženou podobu (tabulku), bude sloužit jako ”parent”.
- Parent entity (dědičnost entit).

XCode DB Model Editor




ENTITIES
Course
ISIC
Student
FETCH REQUESTS
allStudents
CONFIGURATIONS
Default
Attributes
Attribute
Type
name
String
Relationships
Relationship
Destination
Inverse
courses
Course
studentsIn
isic
ISIC
student
Fetched Properties
Fetched Property
Predicate
Relationship
Name
courses
Properties
Transient
Optional
Destination
Course
Inverse
studentsIn
Delete Rule
Nullify
Type
To Many
Arrangement
Ordered
Count
Unbounded
Minimum
Unbounded
Maximum
Advanced
Index in Spotlight
Deprecated
Spotlight
Store in External Record File
User Info
Key
Value
Versioning
Hash Modifier
Version Hash Modifier
Renaming ID
Renaming Identifier

Objektová DB

- Význam referencí mezi objekty:
  - Nejsou režijní atributy relačního modelu jako klíče pro spojování záznamů (objektově orientované DB — GemStone).
  - 1:1 — Student ”má” ISIC,
  - 1:N — Student ”má zapsáno” [Course].
  - 1:N — Course ”má” [Student]. V důsledků M:N.
- Ukládání referencí je ve správě/režii CD.
  - Obecně transformace DB-Modelu na DB-schema.

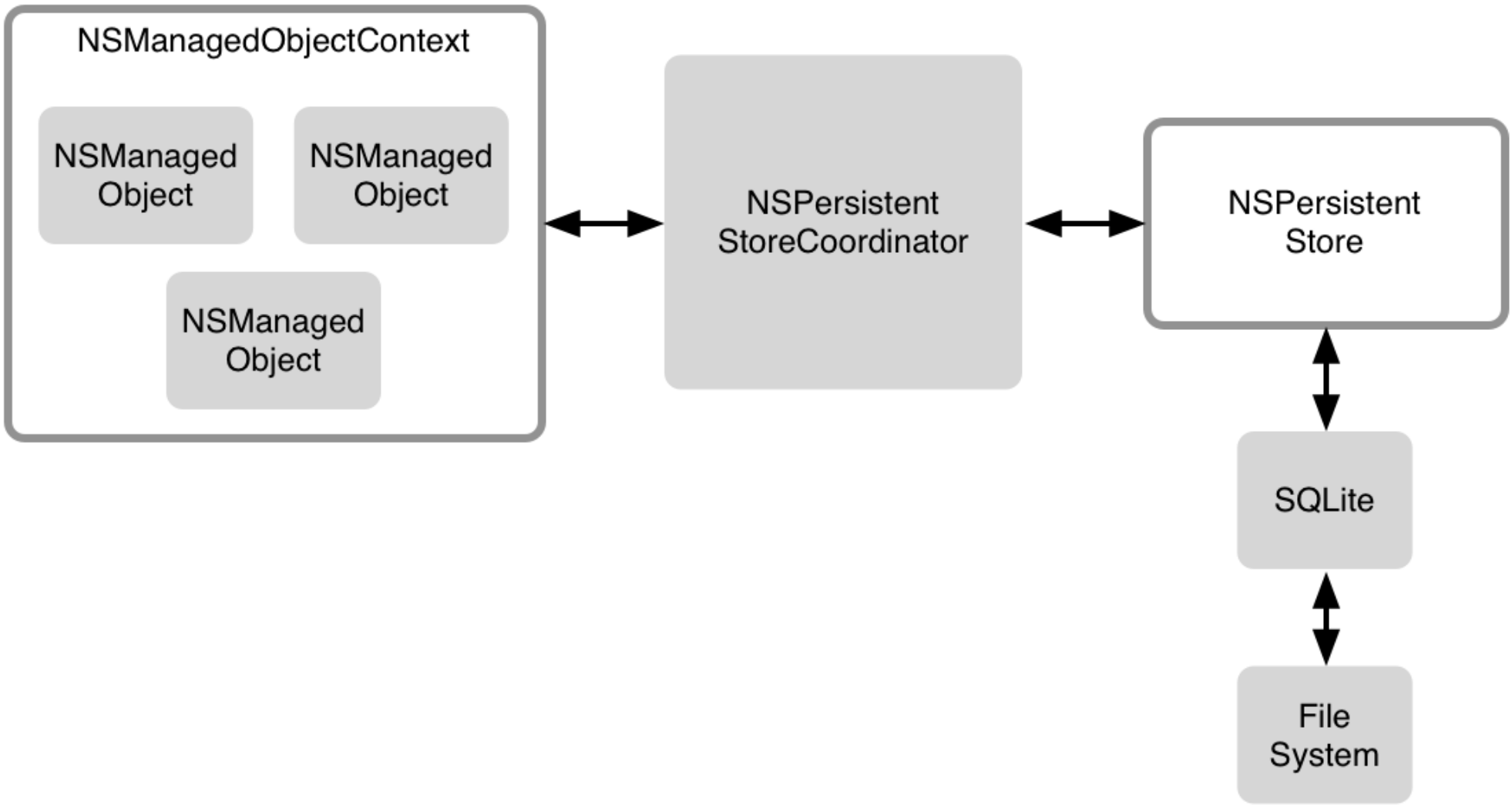
Co nás bude zajímat

- Architektura CoreData knihovny.
- Formulace datového modelu aplikace.
- Způsob integrace CD do aplikace.
- Způsob fungování CD.
- Synchronizovaná CD (Cloud) — ve spojení s CloudKit.
  - Návaznost na: UIDocument (kódování dat), CloudKit.

Architektura CD

- CD staví na principu *Key-Value Coding*, tj. pracujeme s *NSObject* (Foundation).
- *NSManagedObject* — persistentní objekt.
- *NSManagedObjectModel* — editor v XCode.
- *NSManagedObjectContext* — OO paměť.
- *NSPersistentStoreCoordinator* — řadič na fyzická uložiště.
- *NSPersistentContainer* — novinka od iOS 10.

Typická infrastruktura CD



NSManagedObjectModel

- *NSEntityDescription, NSPropertyDescription*.
- Inicializuje se z URL v App-Bundle (spec formát).
- Verzování:
  - Automatické — ”Add model version…” (XCode)
  - Manuální — stará/nová DB, transformace objektů.

NSPersistentStoreCoordinator

- Konstrukce: objekt s DB Modelem. Drží model.
- Umožňuje transformace DB (verzování).
  - V podstatě řízená kopie dat z jedné DB/Modelu do druhé DB/Modelu.
- Vykonává (serializuje) DB operace.
- NSPersistentStore — ovladač nad konkrétním uložištěm (SQLite3, XML, další).
  - Vlastní uživatelské ovladače na další typy uložišť.

Architektura celkově

- Instancuje se Model.
- Nad Modelem se instancuje StoreCoordinator.
  - Přidají se PersistentStores.
- Nad StoreCoordinatorem se konstruuje ManagedObjectContext (MOC).
- MOC je zapouzdřen v PersistentContainer.

Podporované typy ulož. atr.

- Primitivní:
  - String.
  - Int, Bool, Double — na rozdíl od Obj-C typy přímo (bylo NSNumber).
  - Date — NSDate.
  - BinaryData — NSData (BLOB) — kódovaná příloha (UIDocument, obrázek, soubor).

NSManagedObject

- V XCode Model editoru sestavíme Entities.
  - Codegen: Class definition (extension), Manual/None.
- Menu Editor-”Create Managed Subclass” generujeme třídy pro entity:
  - soubor Entity+CoreDataClass.swift
  - soubor Entity+CoreDataProperties.swift
- Třída pro Entitu je @objc, odvozená z NSManagedObject.

NSManagedObject properties

- Uložené atributy Entity se generují do kódu třídy jako *@NSManaged var Nazev: Typ?*
- *Typ?* — objekt se konstruuje bez hodnot (odložené načtení z DB, prázdná hodnota, …).
- *@NSManaged* — v Objective-C je jako *@dynamic* property (překladač negeneruje její getter / setter) — KVC, dynamicky.

Dynamika NSManagedObject

- Objekt se instancuje (záležitost jazyka Swift/Objective-C).
- Stane se ”managed” — je pod správou MOC:
  - iniciováno fetchem — naplnění daty je odloženo.
  - iniciováno aplikací (vytvoření nového) — nemá záznam v DB.
- Spojení datových atributů MO s DB podstatou.
  - KVC, dynamika.

Pozadí managed property

- Čtení object ”.” property:
  - V Objective-C vždy primitivní atribut ”_property” a volání getteru
  - -(Typ) property { return _property; }
- Property je @dynamic (@NSManaged)
  - Volání getteru je nahrazeno:
  - *valueForKey(key: ”property”)* — Objective-C.
  - *value(forKey: ”property”)* — Swift.

value(forKey: ”property”)

- MO zná svůj datový model (soupis properties).
- Pokud je ”property” uložený atribut CD, pak:
  - pokud je objekt načtený, vrátí primitivní hodnotu,
  - pokud objekt není načtený (faulting), pak aktivuje načtení ze svého záznamu a vrátí pak primitivní hodnotu.
  - tj. -> MOC -> PersistentCoordinator -> PersistenStore.
- Podobně setValueForKey.
- Režie při přístupu na atributy. Vlákno.

Celkově MO

- Stále je to třída/objekt — další prvky OOP (metody, atributy apod). Uložené atributy.
  - instanciace: NSEntityDescription, MOC.
- Settery/Gettery uložených atributů jsou ”managed”.
- Stav MO. Pozor na referencování.
  - objectID.
  - isInserted, isUpdated, isDeleted, …

ManagedObjectContext

- Správa MO. MO má vazby na MOC.
  - **MO je stále objektem v kontextu jazyka**…datový delete ho odstraňuje z MOC/CD, ale může být referencován jinde.
  - … tj přemýšlet nad vkládáním MO do kolekcí vzhledem k operaci delete.
- Rozhraní pro operace vkládání, rušení, fetch.
- Všechno je odložené! Načítání, inserty, update.
  - saveContext() — vykoná insert/update nad objekty.

CD z pohledu aplikace

- Globální aplikační objekt (AppDelegate) vlastní reference na CD objekty (MOC, Model, PersistentStoreCoordinator) — ”trojice”.
- Od iOS10 zavedeno zapouzdření ”trojice” do NSPersistentContainer.
  - lazy var property, persisentContainer: NSPersistentContainer.
  - UIApplication.shared.delegate as! AppDelegate

Vytvoření nového objektu

- NSEntityDescription — Metadata o zadané entitě. Datový model se bere z MOC.
- Nový MO se stává součástí kontextu, ovšem zatím pouze v paměti MOC. Datový insert je odložen.

```
// application delegate
let _appd = UIApplication.shared.delegate as! AppDelegate
// managed object context
let _vc = _appd.persistentContainer.viewContext
// novy MO
let _newStudent = NSEntityDescription.insertNewObject(forEntityName:
"Student", into: _vc) as! Student
// generovana procedura pro vyprazdneni cache MOC
_appd.saveContext()
```

Generované třídy pro MOs

```
@objc(ISIC)
public class ISIC: NSManagedObject {
    @NSManaged public var expires: NSDate?
    @NSManaged public var idNumber: String?
    @NSManaged public var student: Student?
}
```

```
@objc(Student)
public class Student: NSManagedObject {
    @NSManaged public var name: String?
    @NSManaged public var courses: NSSet?
    @NSManaged public var isic: ISIC?
    //
    @objc(addCoursesObject:)
    @NSManaged public func addToCourses(_ value: Course)

    @objc(removeCoursesObject:)
    @NSManaged public func removeFromCourses(_ value: Course)
}
```

Příklad konstrukce MO

```
@objc(Student)
public class Student: NSManagedObject {
    //
    class func create(inMOC: NSManagedObjectContext,
                      name: String) -> Student
    {
        // Pozn.: dnes zkratka: Student(context: inMOC)
        let _mo = NSEntityDescription.insertNewObject(forEntityName:
"Student",
                                                      into: inMOC) as!
Student

        //
        _mo.name = name

        //
        return _mo
    }
```

Extension třídy MO

- Každé entitě DB-Schematu přísluší třída ve Swiftu!
  - Lze ji rozšířit ”extension Student”,
  - Může mít uložené (nejsou-managed) atributy,
  - Může mít instanční a třídní metody.

Propojování vazeb

- Apple velmi silně doporučuje definovat veškeré *relationships jako obousměrné* (každá má inverzní variantu).
- CoreData spravuje obě vazby, tj. stačí nastavit (nulovat) jednu.
  - 1:1 — jednoduché.
  - 1:N — nastavit ze strany 1:
  - M:N — generované procedury.

Relationships

Relationship	Destination	Inverse
M studentsIn	Student	courses

Relationship	Destination	Inverse
M courses	Course	studentsIn
O isic	ISIC	student

Relationship	Destination	Inverse
O student	Student	isic

Propojování vazeb

```
let _courseIZA = Course(context: _vc)

// nastaveni vazby 1:1
_newISIC.student = _newStudent
// ekvivalent je (neni treba provadet)
// (CoreData nastavuje inverzi automaticky)
_newStudent.isic = _newISIC
// 1:N, M:N
_newStudent.addToCourses(_courseIZA)
// ekvivalent
// opet, CD provede automaticky
// z predchoziho prikazu
_courseIZA.addToStudentsIn(_newStudent)
```

Rušení vazeb

- Nastavit na nil (setter v MO opět provede i inverzní variantu vazby).
  - Pozor na MO, které pak zůstanou nereferencované.
- Generované procedury — add/remove.
- Správa vazeb v okamžiku mazání (delete) MO.
  - Je údajem o relationship (editor modelu).
  - Delete rule — No Action, Nullify, Cascade, Deny.

Mazání objektu, Delete rule

- A -> B. Chceme smazat objekt A, který se váže na B prostřednictvím relation R (delete rule).
  - No Action — referencovaný objekt B nedostává žádnou zprávu. Smysl???
  - Nullify (impl.) — provede se nulování inverzní vazby a B zůstává.
  - Cascade — maže navázaný objekt B. Dále kaskádově.
  - Deny — odmítne akci, tj. A lze smazat pouze pokud nemá vazbu na B. Vazba R zamítne smazání A.

Fetch request

- DB operace typu SELECT … FROM … WHERE
  - Pozor: načítá se objekt z kontejneru. Není zde projekce.
- NSFetchRequest<CLASS>(entityName: String)
- Volitelné parametry:
  - predicate — NSPredicate
  - sortDescriptors — [NSSortDescriptor]
  - fetchLimit — Int.
- try MOC.fetch(request) ….as! [CLASS]

FetchRequest, resultType

- *NSFetchRequestResultType*
  - managedObjectResultType — vrátí MO objekty.
  - managedObjectIDResultType — vrátí ID objektů.
  - dictionaryResultType — vrátí Dictionary klíč-hodnota (???). Lze specifikovat soupis atributů.
  - countResultType — vrátí počet záznamů, které by vyšly na dotaz.
- Pozn.!!! MOC.fetch(…) je paměťová operace, tj. kombinuje MO v paměti a disk.

Agregace apod

- Lineární fetch. Zbytek nutno vypočítat v programu.
- Vycházíme z představy OO modelu, kde s MO jde společně i množina referencí, tudíž konvenční JOIN zde neexistuje.

NSPredicate

- NSPredicate(format: String, …)
  - %@ — symbol pro vložení hodnoty objektu z argumentů.
  - %K — key-path.
  - ==, =, <, >, AND, OR, …
  - BEGINSWITH — format: ”%@ BEGINSWITH ‘Hell’”, ”Hello”
  - CONTAINS[cd] — (c)ase insensitivity, (d)iacritic insensitivity.
  - LIKE — *, ?

NSPredicate, keypaths

- Uložené atributy.
- Relationships (keyPaths):
  - ”isic.idNumber == ‘123’”
  - ”studentsIn.@count > 0” - @count
- Pozn.: compound predicate
- Pozn.: transformace predikátu na SQL select where.

NSSortDescriptor

- Jméno klíče, ascending: Bool.
- fetch.sortDescriptors = [Sd1, sd2, …]

CoreData jako dataSource

- TableViewController, fetch, pole výsledků.
- Dynamika:
  - zpráva o případné změně obsahu nějakého Fetch, tj. …
  - … do výběru objektů (select) by se dostal nový další,
  - … objekt z výběru změnil obsah.
  - NSFetchedResultsController.

Fetch, demo

```
class ViewController: UITableViewController {
    // pole objektu pro tabulku
    var content: [Student] = []
    //
    override func viewDidLoad() {
        //
        super.viewDidLoad()
        // pristup na MOC referencovany z AppDelegate
        let moc = (UIApplication.shared.delegate as!
AppDelegate).persistentContainer.viewContext
        // konstrukce dotazu -> template, vysledek [Student]
        let fetchr = NSFetchRequest<Student>(entityName: "Student")
        // provedeni dotazu
        if let _results = try? moc.fetch(fetchr) {
            //
            content = _results; tableView.reloadData()
        }
    }
}
```

PC, MOC

```
//
func PC() -> NSPersistentContainer {
    //
    return (UIApplication.shared.delegate as! AppDelegate).persistentContainer
}

//
func MOC() -> NSManagedObjectContext {
    //
    return PC().viewContext
}
```

```
class ViewController: UITableViewController {
    // pole objektu pro tabulku
    var content: [Student] = []
    //
    override func viewDidLoad() {
        //
        super.viewDidLoad()
        // fetch se provede az po vykresleni VC
        DispatchQueue.main.async {
            // pristup na MOC referencovany z AppDelegate
            let moc = (UIApplication.shared.delegate as!
AppDelegate).persistentContainer.viewContext
            // konstrukce dotazu -> template, vysledek [Student]
            let fetchr = NSFetchRequest<Student>(entityName: "Student")
            // provedeni dotazu
            if let _results = try? moc.fetch(fetchr) {
                //
                self.content = _results; self.tableView.reloadData()
            }
        }
    }
}
```

Vláknová vsuvka do CD…

- Globální aplikační MOC je v režimu hlavního vlákna, tj.
  - všechny druhy přístupů na MOC/MO v MT,
  - insert/delete,
  - getter/setter na MO.
- Budeme totiž chtít přejít s CD do globálních vláken.
  - Budeme chtít zapnout/vypnout napojení UI.

NSFetchedResultsController

- Controller nad FetchResults. Observer MOC.
  - Dostává zprávy o změnách na Entitě (insert/update/delete).
  - Analyzuje změny a formuluje elementární změny ve svém datasource.
- Atributy FRC:
  - fetchRequest — sortDescriptor povinný.
  - řazené pole objektů MO. Přistupujeme *frc.object(at: IndexPath)*
  - delegate.

FRC, sestavení

- FetchRequest, sortDescriptors.
- Perform Fetch:
  - provede první fetch a uspořádá si objekty do interního pole,
  - další změny nad objekty analyzuje a předává svému delegate.
- FRC je pouze pro iOS.
  - NSArrayController v Cocoa.
  - v Cocoa (macOS) mírně odlišný koncept tabulek (sloupce).

```
//
    var FRC: NSFetchedResultsController<Student>!

    //
    func initFRCStudents() {
        // FR
        let _fetchStudents = NSFetchRequest<Student>(entityName:
"Student")
        let _sortByNameUP = NSSortDescriptor(key: "name", ascending:
true)

        // povinna polozka FRC, jinak lehne
        _fetchStudents.sortDescriptors = [ _sortByNameUP ]

        // bez pouziti sections
        FRC = NSFetchedResultsController(fetchRequest: _fetchStudents,
                                         managedObjectContext: MOC,
                                         sectionNameKeyPath: nil,
                                         cacheName: nil)

        // NSFetchedResultsControllerDelegate
        FRC.delegate = self

        // Prvotni nacteni obsahu, inicializace
        try! FRC.performFetch()
    }
```

NSFetchedResultsControllerDelegate

- FRC je schopno synchronizovat načtené pořadí objektů s případným znovu-fetchem.
- controllerDidChangeContent — obecná zpráva, pak zřejmě tableView.reloadData()

```
func controllerDidChangeContent(_ controller:
NSFetchedResultsController<Student>)
    {
        //
        self.tableView.reloadData()
    }
```

FRC Delegate, editace tabulky

- willChangeContent,
- didChangeObject,
- controllerDidChangeContent

```
func controllerWillChangeContent(_ controller:
NSFetchedResultsController<NSFetchRequestResult>)
    {
        tableView.beginUpdates()
    }

    //
    func controllerDidChangeContent(_ controller:
NSFetchedResultsController<NSFetchRequestResult>)
    {
        tableView.endUpdates()
    }
```

...

```
//
    func controller(_ controller:
NSFetchedResultsController<NSFetchRequestResult>,
                    didChange anObject: Any,
                    at indexPath: IndexPath?,
                    for type: NSFetchedResultsChangeType,
                    newIndexPath: IndexPath?)
    {
        //
        switch type {
        case .insert:
            //
            tableView.insertRows(at: [newIndexPath!], with: .fade)
        default:
            //
            ()
        }
    }
```

FRC jako obecný dataSource

- Konstrukce FRC a obsluha delegátského protokolu — stále stejný kód.
  - Nutno: NSFetchRequest<NSFetchRequestResult>, pak přetypovávat objekty pole.
- Liší se pouze procedura pro konfiguraci buňky.
- Univerzální dataSource.

FRC, UITableViewDataSource

```
    func numberOfSections(in tableView: UITableView) -> Int {
        //
        return FRC.sections!.count
    }
    func tableView(_ tableView: UITableView,
                   numberOfRowsInSection section: Int) -> Int
    {
        //
        return FRC.sections![section].numberOfObjects
    }
    func tableView(_ tableView: UITableView,
                   cellForRowAt indexPath: IndexPath) -> UITableViewCell
    {
        //
        let _cell = tableView.dequeueReusableCell(withIdentifier:
"cell",
                                                  for: indexPath)
        //
        delegate(FRC.object(at: indexPath), _cell)
        //
        return _cell
    }
```

FRC, princip

- MOC generuje NSNotification na události (update property, insert, delete).
- FRC je observer (notifCenter).
- Filtruje události přes svoji FetchRequest, případně přeposílá.
  - Pokud plánujeme rozsáhlejší aktualizaci obsahu MOC, pak raději v lokálním/dočasném MOC, pak uložit změny.
  - … jinak to bude hýbat UI řízenou FRC.

MOC, undo management

- MOC sbírá události o změnách ve svých MO.
  - insert, delete, update (setter z MO).
- processPendingChanges()
  - automaticky invokováno z RunLoop. **Nevíme kdy…**
  - MOC rozešle (sync) událost o provedení změny.
  - FRC reaguje: začátek změn, popis změn, konec změn.

Aktualizace MO na pozadí

- mainMOC.save()
  - mainMOC.automaticallyMergesChangesFromParent = true
- mainMOC -> newMOC
  - fetch, insert, delete, aktualizace MO — bez vazby na UI.
  - globální vlákno.
  - newMOC.save() -> promítne změny do mainMOC
- mainMOC.save()

Aktualizace MO na pozadí

```
func onBackground() {
        //
        let _appd = UIApplication.shared.delegate as! AppDelegate
        let _vc = _appd.persistentContainer
        // detsky MOC k hlavnimu MOC, pojede v separatni fronte
        let _nb = _vc.newBackgroundContext()
        // save() provede merge do MOC
        MOC.automaticallyMergesChangesFromParent = true
        // do background vlakna/fronty
        _nb.perform {
            //
            let _fr = NSFetchRequest<Course>(entityName: "Course")
            //
            if let _res = try? _nb.fetch(_fr) {
                //
                for _c in _res {
                    //
                    _c.name = "Zapsano-" + _c.name!
                }
            }
            // proved zmeny do hlavniho MOCu
            try! _nb.save()
        }
    }
```

Tabulky nad objekty, kaskáda

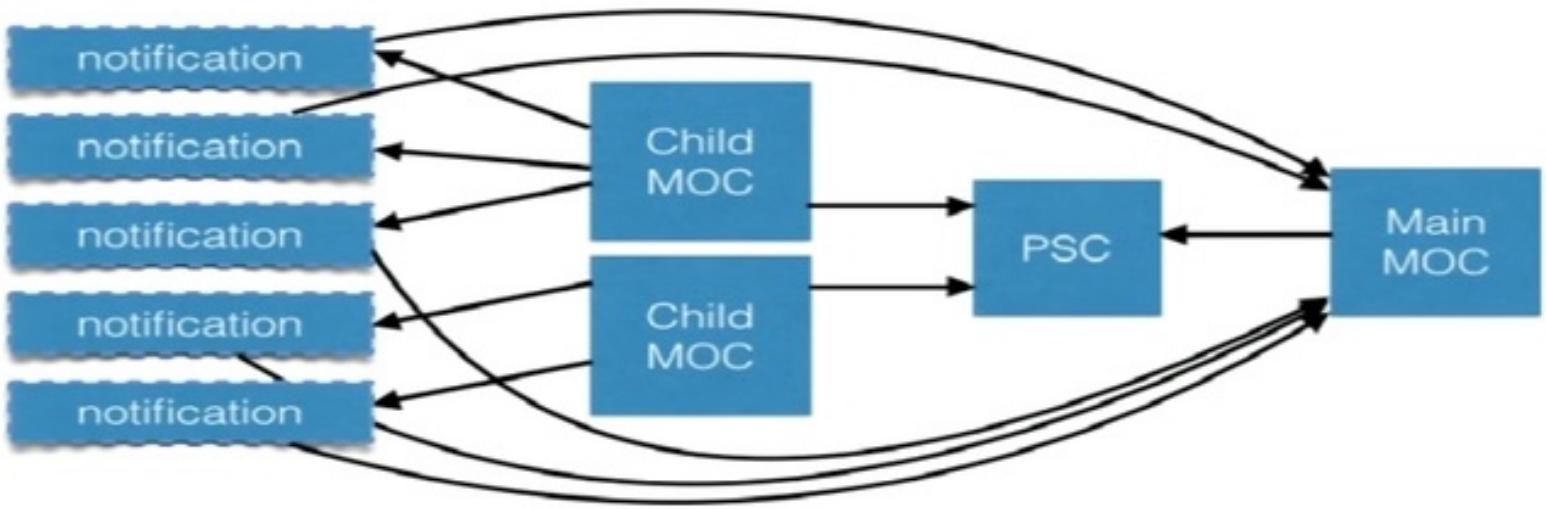
- DataSource — fetch, fetchResultsController, primitivní pole.
- Situace:
  - Table-VC-1 na FRC[Course]. Na selekci TVCell další VC s detailem na course.studentsIn — Set<Student>
  - Table-VC-2: buď nad polem [Student] nebo nad FRC s patřičným NSPredicate.
  - _fetchStudents.predicate = NSPredicate(format: "courses CONTAINS %@", selectedCourse)

Ukládání managed kontextu

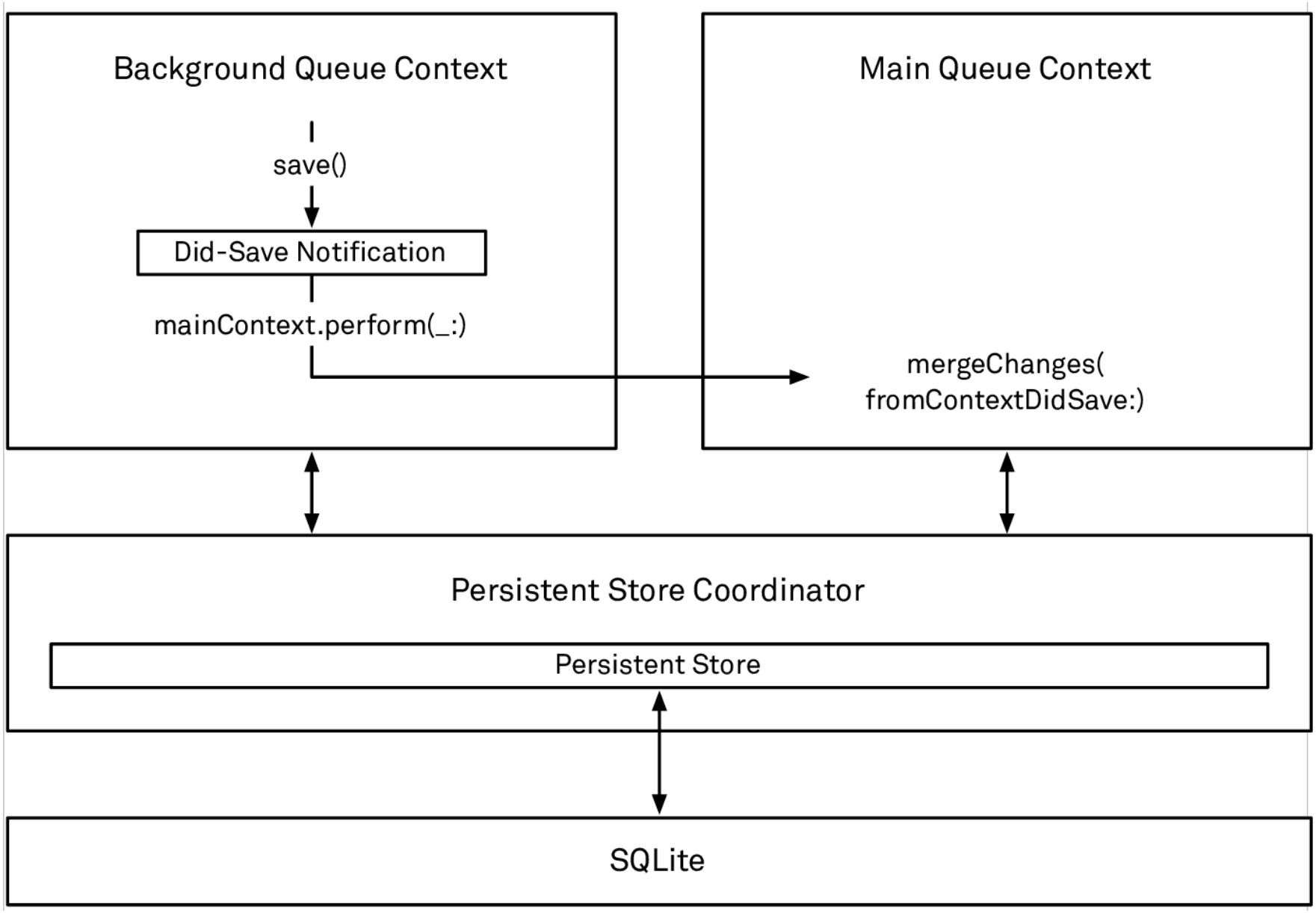
- *Veškeré operace nad objekty MO jsou s odloženým zápisem!*
- Kdy provádět MOC.save():
  - podle důležitosti při každé změně MO,
  - applicationDidEnterBackground,
  - applicationWillTerminate,
  - … asi i před spuštěním child-MOC vlákna.

Threading, CD

- Operace MOC by měly být vykonávány přes nějakou sériovou frontu: MainQueue nebo privátní fronta MOC.
- Více-vláknová App — klony MOC. Klony MO.



Slučování MOC



```
let jsonArray = … //JSON data to be imported into Core Data
let moc = … //Our primary context on the main queue

let privateMOC =
NSManagedObjectContext(concurrencyType: .PrivateQueueConcurrencyType)
privateMOC.parentContext = moc

privateMOC.performBlock {
    for jsonObject in jsonArray {
        let mo = … //Managed object that matches the incoming JSON structure
        //update MO with data from the dictionary
    }
    do {
        try privateMOC.save()
        moc.performBlockAndWait {
            do {
                try moc.save()
            } catch {
                fatalError("Failure to save context: \(error)")
            }
        }
    } catch {
        fatalError("Failure to save context: \(error)")
    }
}
```

Aktualizace MO na pozadí

```
let jsonArray = …
let container = self.persistentContainer

container.performBackgroundTask() { (context) in
    for jsonObject in jsonArray {
        let mo = EmployeeMO(context: context)
        mo.populateFromJSON(jsonObject)
    }
    do {
        try context.save()
    } catch {
        fatalError("Failure to save context: \(error)")
    }
}
```

Příště…

- Realm DB — Filip Klembara.
- UIDocuments, kódování dat.
- CloudKit.
- Synchronizace dat mezi databázemi na různých zařízeních uživatele v rámci aplikace.