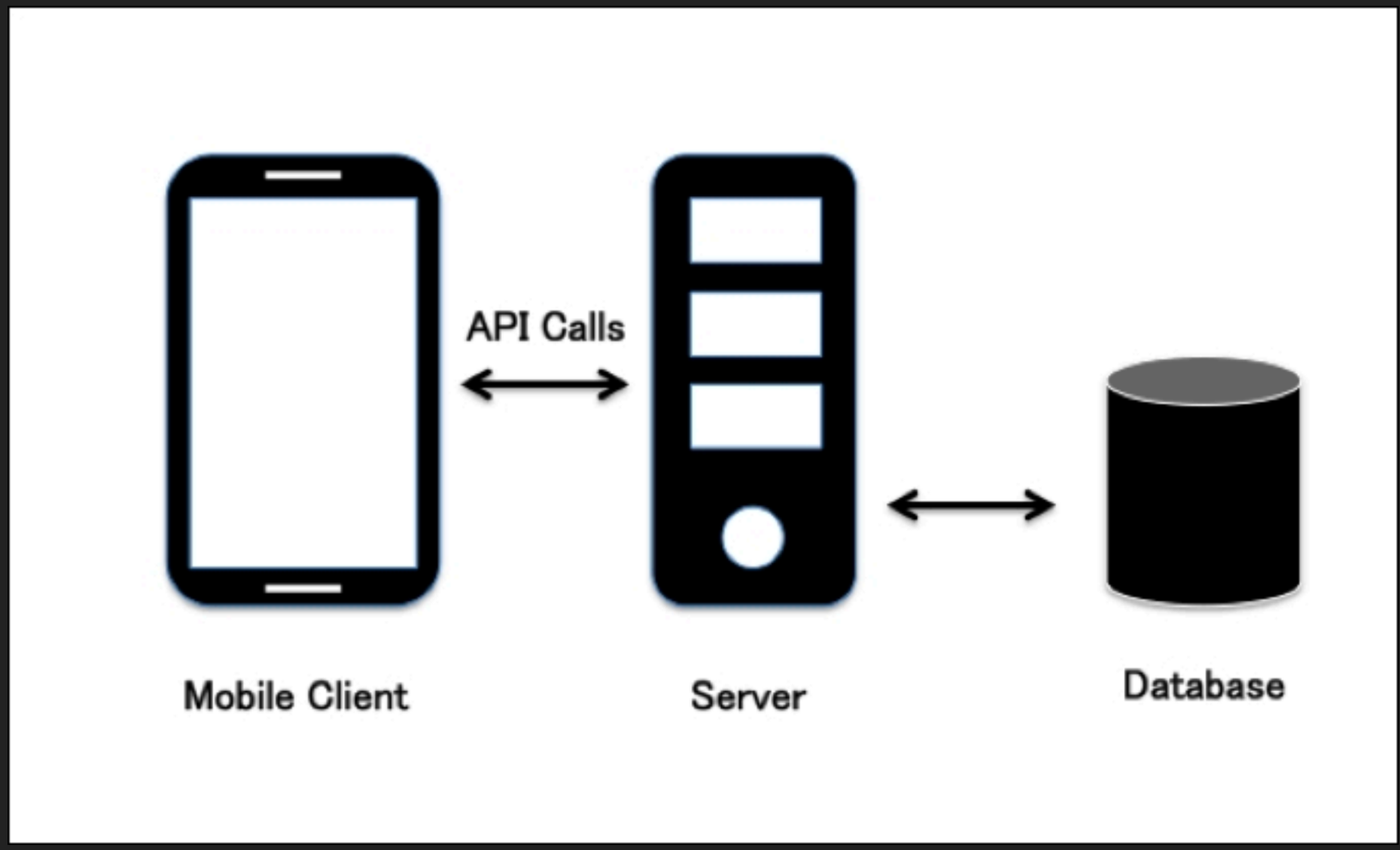
A.STEFANCHUK IT STEP

DATA PERSISTENCE

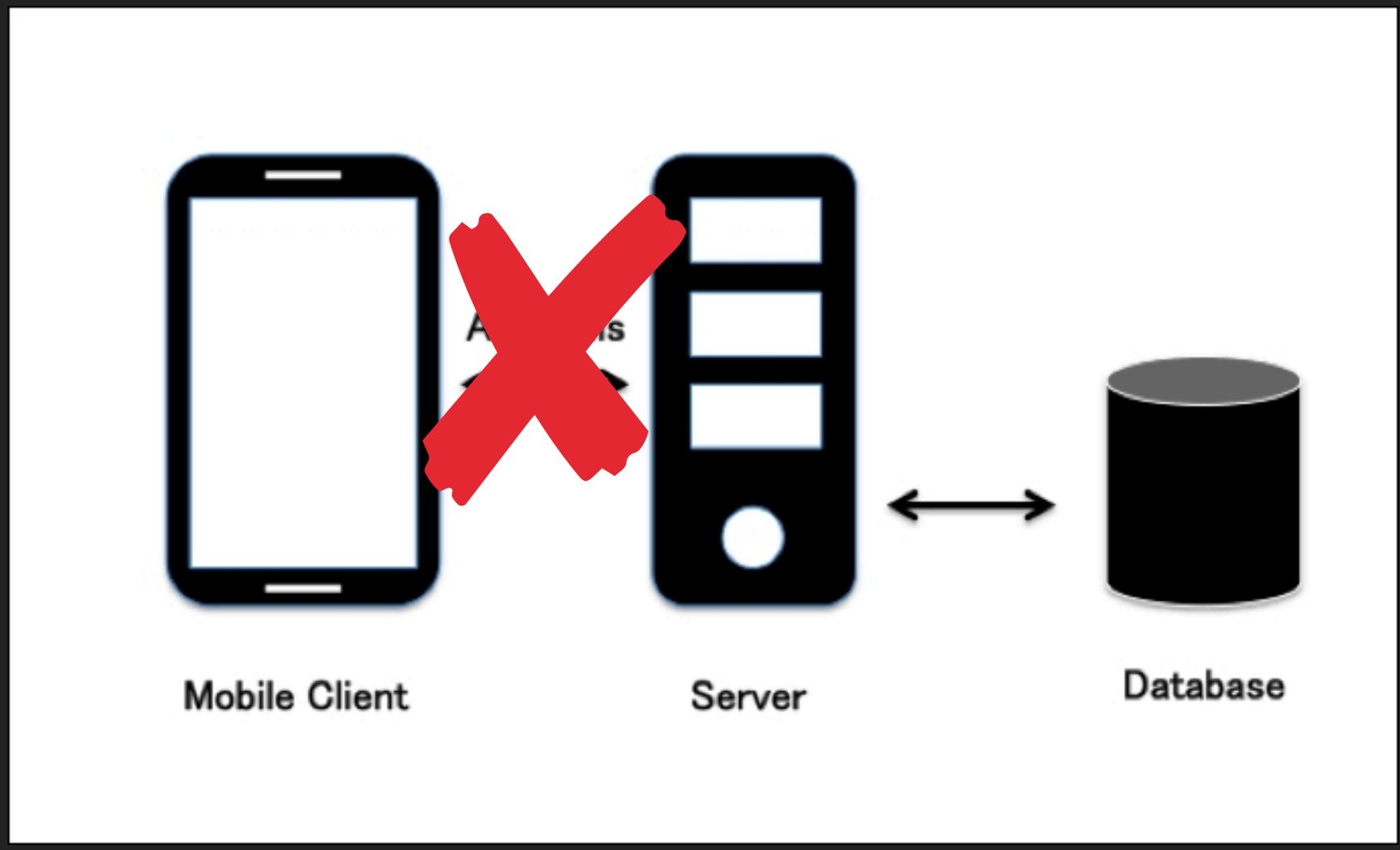
HOW TO SAVE DATA IN YOUR APP

HAVING INTERNET IS GREAT!

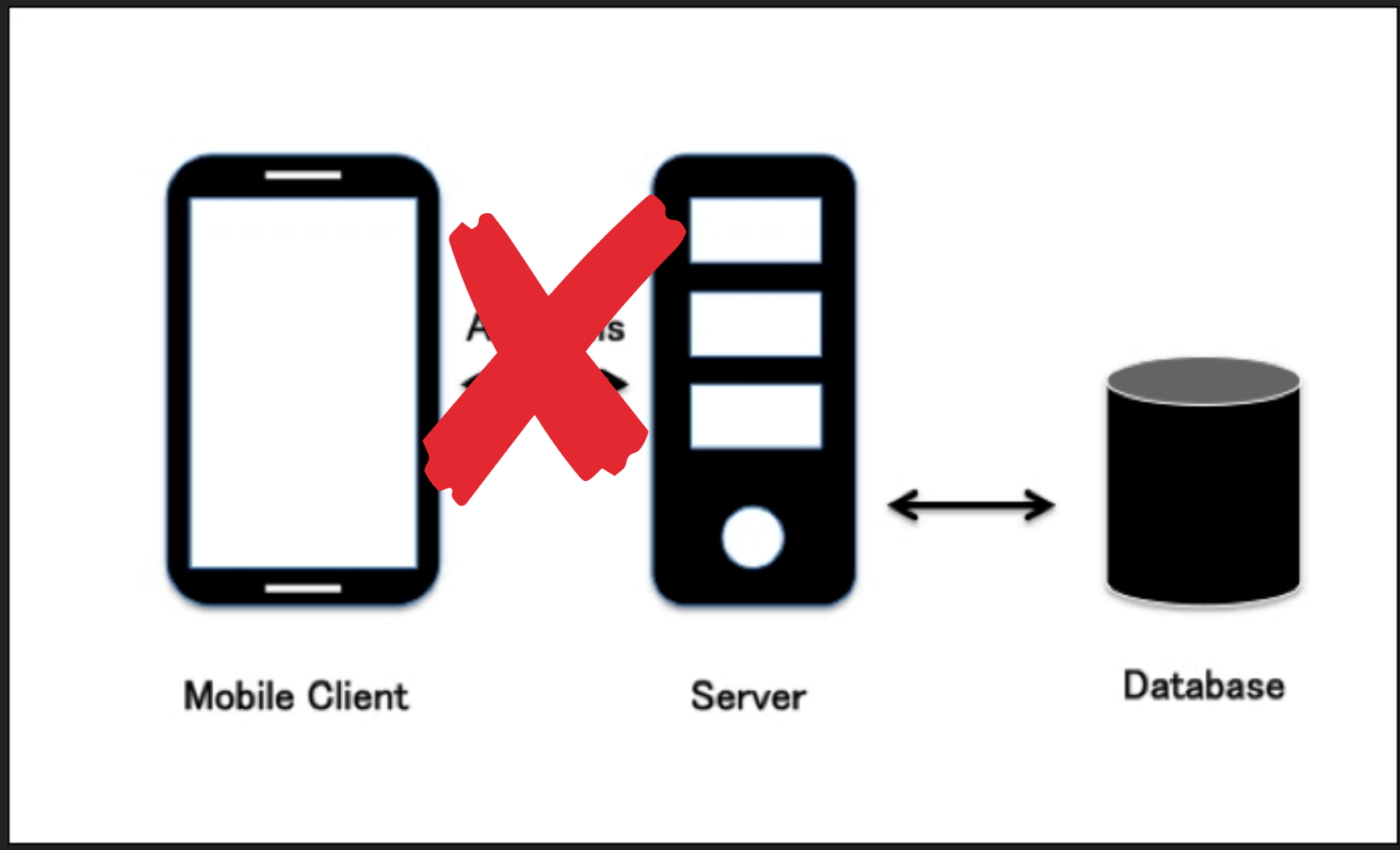
YOU CAN ACCESS EVERY PIECE OF INFO YOU NEED



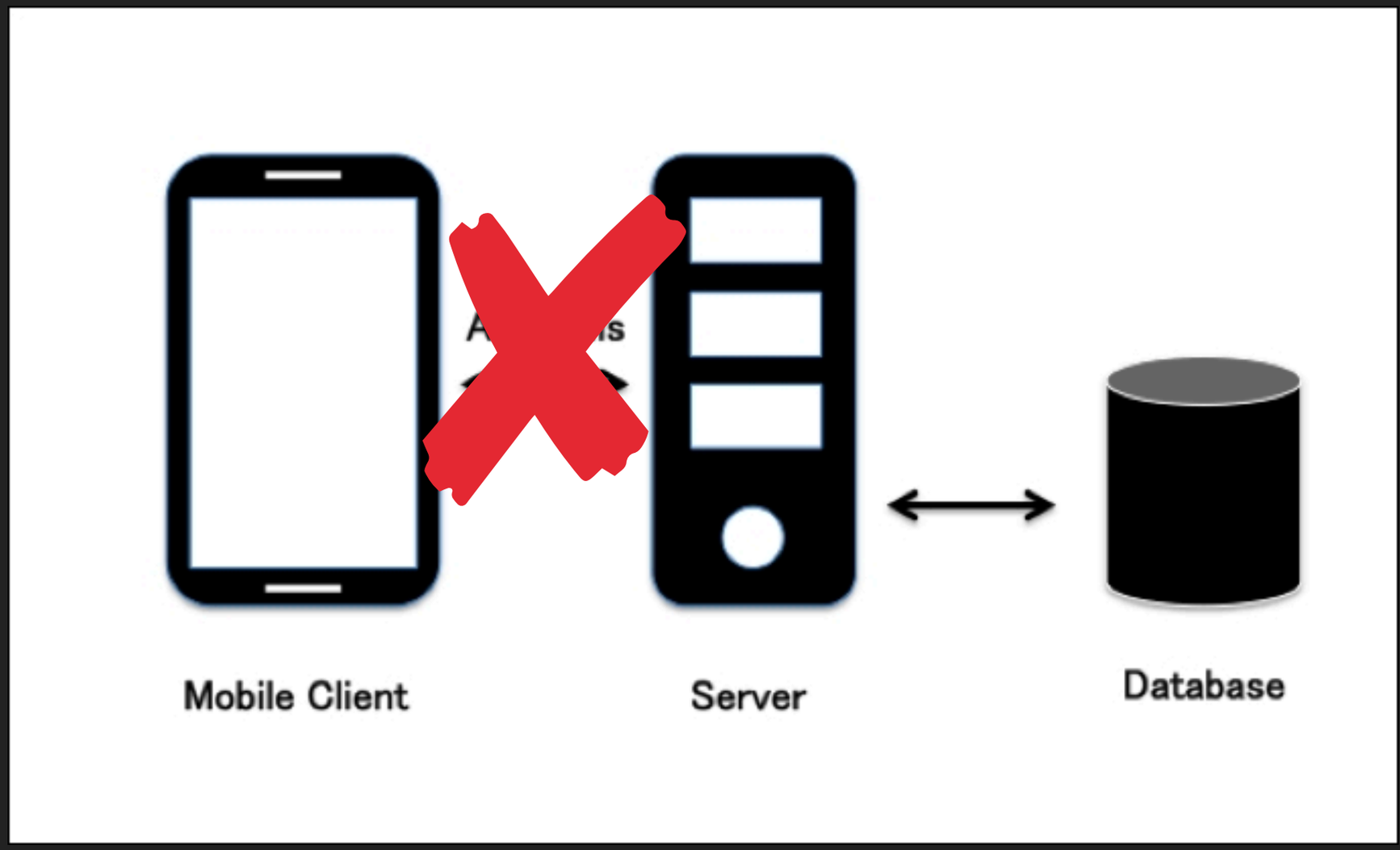
Typical scenario for the app



But what if



But what if there is no internet



Or we just want to have our data locally

SO WE NEED SOME WAY TO SAVE IT ON THE PHONE!

USE CASES

- You save all data locally, without backend (fitness tracker, finance tracker, todo app)

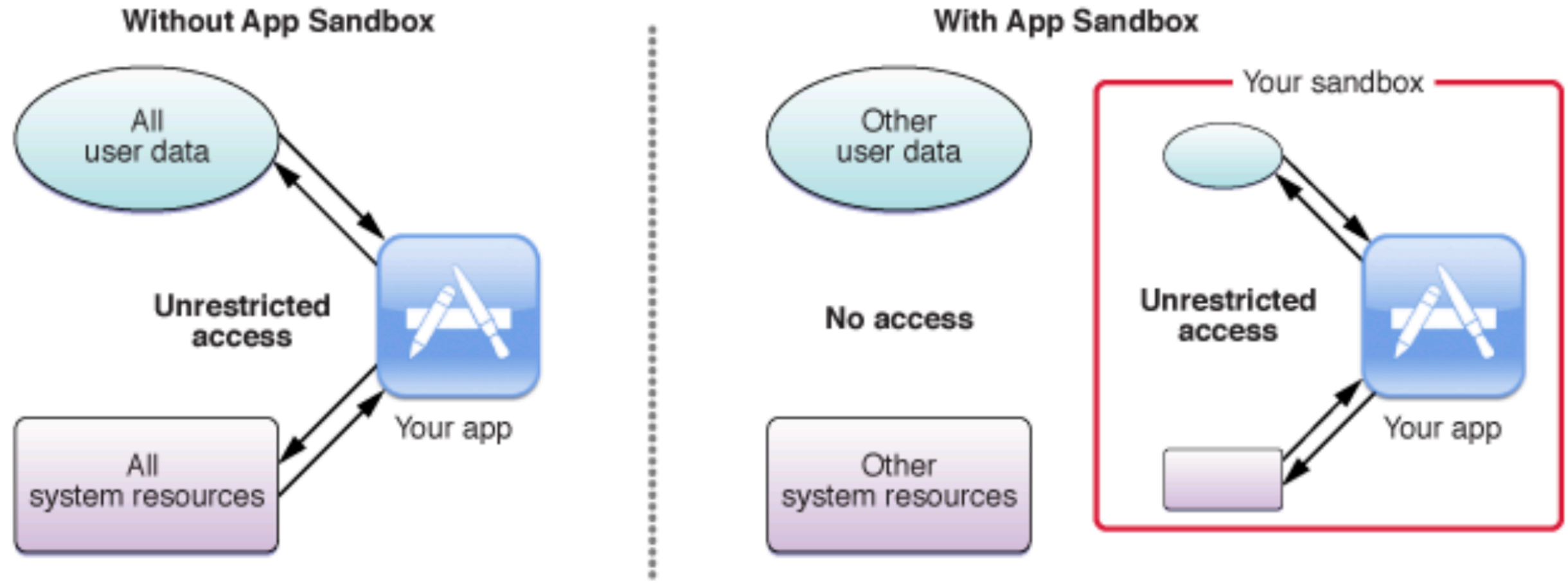
USE CASES

- You save all data locally, without backend (fitness tracker, finance tracker, todo app)
- Offline mode + data caching

USE CASES

- You save all data locally, without backend (fitness tracker, finance tracker, todo app)
- Offline mode + data caching
- Web sockets data updating

APP SANDBOX


Without App Sandbox
All user data
Unrestricted access
Your app
All system resources
With App Sandbox
Other user data
No access
Other system resources
Your sandbox
Unrestricted access
Your app

USER DEFAULTS

- It would be the first thing mentioned on Stackoverflow

USER DEFAULTS

- It would be the first thing mentioned on Stackoverflow
- It’s good to save some user settings values like theme chosen or if tutorial was completed

USER DEFAULTS

- It would be the first thing mentioned on Stackoverflow
- It’s good to save some user settings values like theme chosen or if tutorial was completed
- It’s bad idea to store user password here, because everybody can get this file and read it

USER DEFAULTS

- It would be the first thing mentioned on Stackoverflow
- It’s good to save some user settings values like theme chosen or if tutorial was completed
- It’s bad idea to store user password here, because everybody can get this file and read it
- So, it’s easy to use but has limited usage and you cannot really store your objects here

USER DEFAULTS

```
struct UserDefaultsKeys {
    static let theme = "AppTheme"
    static let isTutorialComplete = "IsTutorialComplete"
}
```

```
func testUserDefaults() {
    let defaults = UserDefaults.standard
    defaults.set("Dark", forKey: UserDefaultsKeys.theme)
    defaults.set(false, forKey: UserDefaultsKeys.isTutorialComplete)

    if let theme = defaults.string(forKey: UserDefaultsKeys.theme) {
        print("Current theme is \(theme)")
    }

    print("User completed tutorial -> \(defaults.bool(forKey: UserDefaultsKeys.isTutorialComplete))")
}
```

KEYCHAIN

- It’s like user defaults, but encrypted

KEYCHAIN

- It’s like user defaults, but encrypted
- It is safe to save some credentials here (passwords and others)

KEYCHAIN

- It’s like user defaults, but encrypted
- It is safe to save some credentials here (passwords and others)
- It can be shared across same Apple ID devices

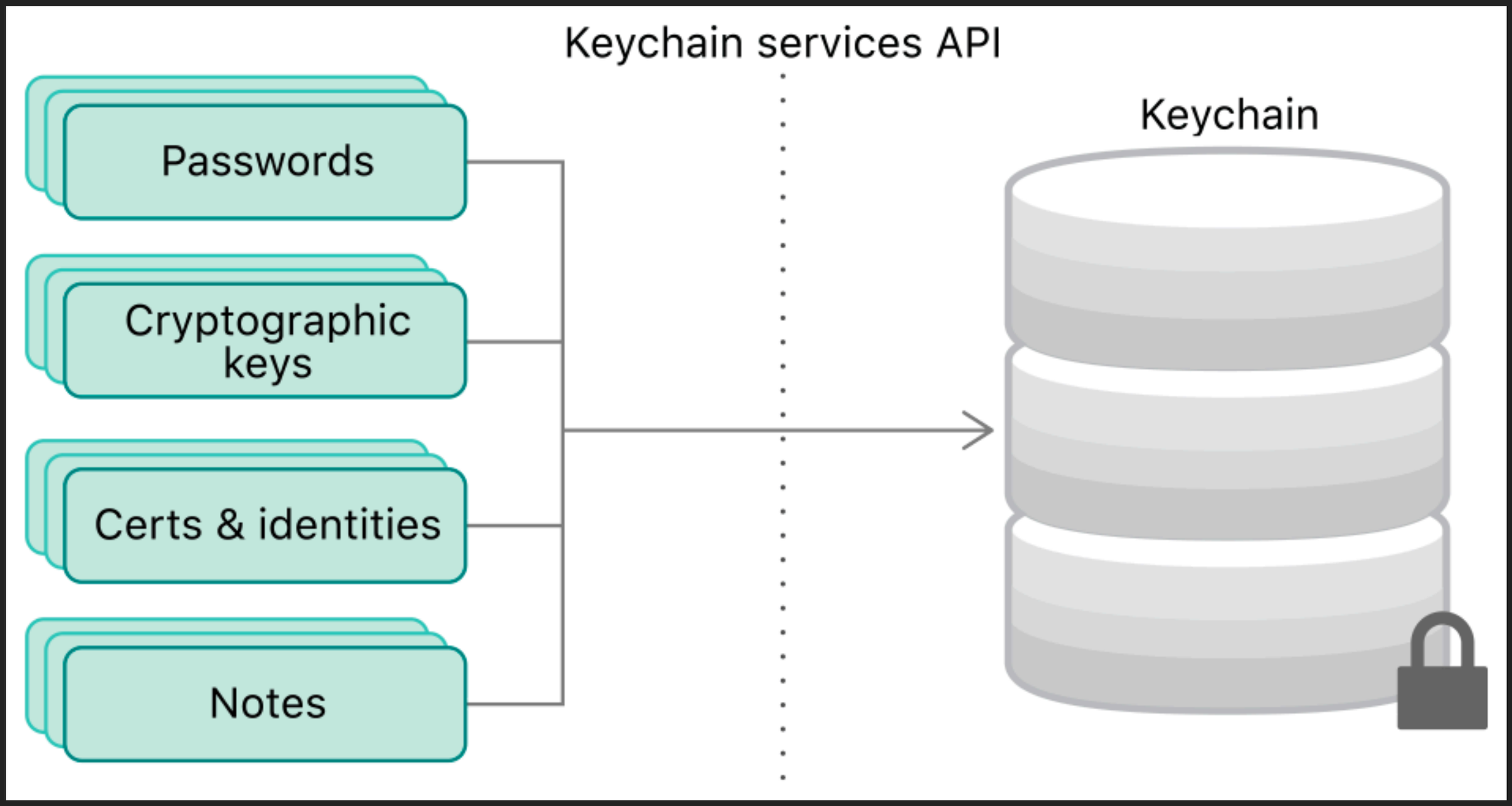
KEYCHAIN

- It’s like user defaults, but encrypted
- It is safe to save some credentials here (passwords and others)
- It can be shared across same Apple ID devices
- It is hard to use it with native code, so you should use some 3rd party library to manage it

KEYCHAIN

- It’s like user defaults, but encrypted
- It is safe to save some credentials here (passwords and others)
- It can be shared across same Apple ID devices
- It is hard to use it with native code, so you should use some 3rd party library to manage it
- Still cannot really save your data objects

KEYCHAIN


Keychain services API
Passwords
Cryptographic keys
Certs & identities
Notes
Keychain

WRITING TO FILE

- You can write every structure you want to file (like JSON, Protobuf, etc.)

WRITING TO FILE

- You can write every structure you want to file (like JSON, Protobuf, etc.)
- You can have multiple files, directories

WRITING TO FILE

- You can write every structure you want to file (like JSON, Protobuf, etc.)
- You can have multiple files, directories
- But:
- Every time you want to modify some data you need to read it from file and write it back

WRITING TO FILE

- You can write every structure you want to file (like JSON, Protobuf, etc.)
- You can have multiple files, directories
- But:
- Every time you want to modify some data you need to read it from file and write it back
- With every saved object file becomes larger

WRITING TO FILE

- You can write every structure you want to file (like JSON, Protobuf, etc.)
- You can have multiple files, directories
- But:
- Every time you want to modify some data you need to read it from file and write it back
- With every saved object file becomes larger
- You can manage multiple files, but there is a lot of headache

PROPERTY LIST FILES

- Also file, but with key value semantic
- Good for storing something really simple, but not more

PROPERTY LIST FILES

- Also file, but with key value semantic
- Good for storing something really simple, but not more

SQLITE

- That a real database with all the queries and stuff, but it not necessarily complicated and there are better options for this task

SO HERE IS OUR FIGHT!

SO HERE IS OUR FIGHT!

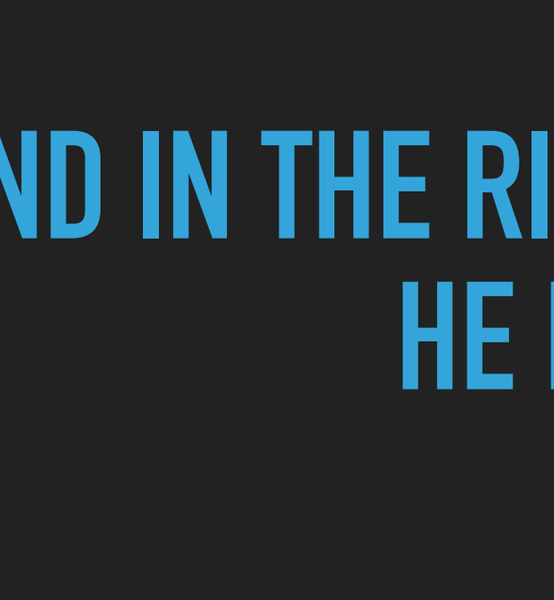
IN THE LEFT CORNER WE HAVE CORE DATA – IOS NATIVE CHAMPION

IN THE LEFT CORNER WE HAVE CORE DATA – IOS NATIVE CHAMPION

AND IN THE RIGHT CORNER WE HAVE REALM – HE LIKE FLASH, SO MEET

VS

AND IN THE RIGHT CORNER WE HAVE REALM – HE LIKE FLASH, SO MEET



VS

REALM VS CORE DATA

REALM

CORE DATA

REALM VS CORE DATA

REALM

Easy to learn and setup

CORE DATA

Take more time to get into

REALM VS CORE DATA

REALM

Easy to learn and setup

Faster read and write

CORE DATA

Take more time to get into

Bit slower operations

REALM VS CORE DATA

REALM	CORE DATA
Easy to learn and setup	Take more time to get into
Faster read and write	Bit slower operations
3rd party	Native

REALM VS CORE DATA

REALM	CORE DATA
Easy to learn and setup	Take more time to get into
Faster read and write	Bit slower operations
3rd party	Native
Cloud sync	

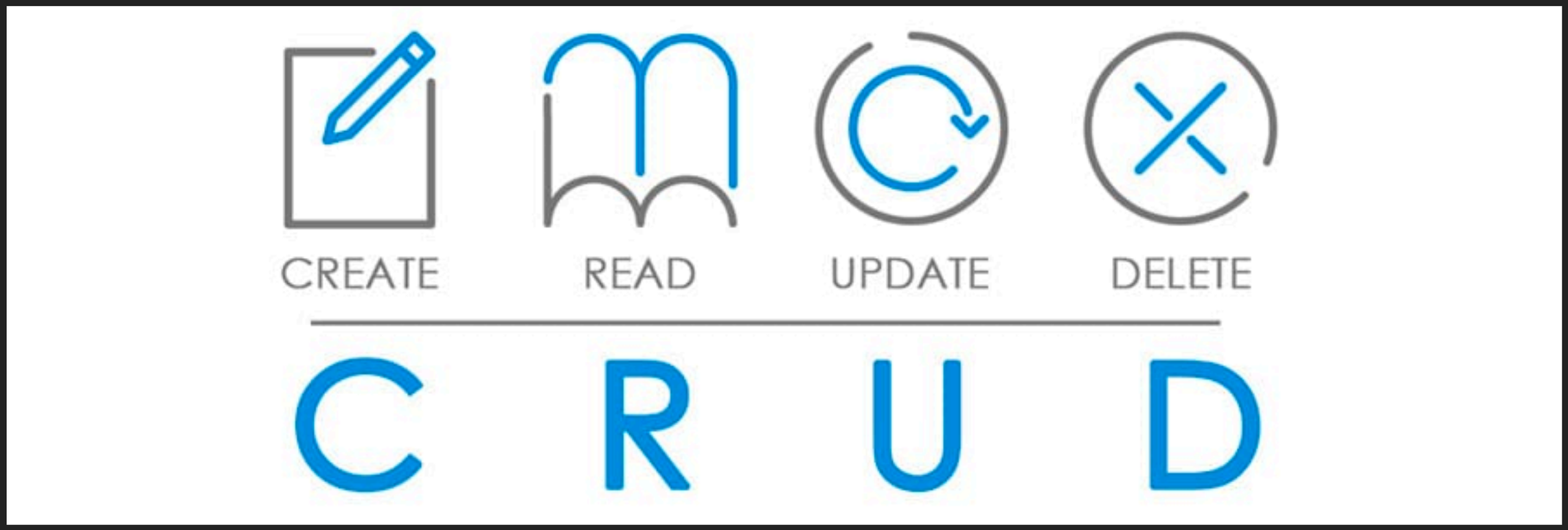
REALM VS CORE DATA

REALM	CORE DATA
Easy to learn and setup	Take more time to get into
Faster read and write	Bit slower operations
3rd party	Native
Cloud sync	
Simple to create schema and relations	Bit harder to create schema

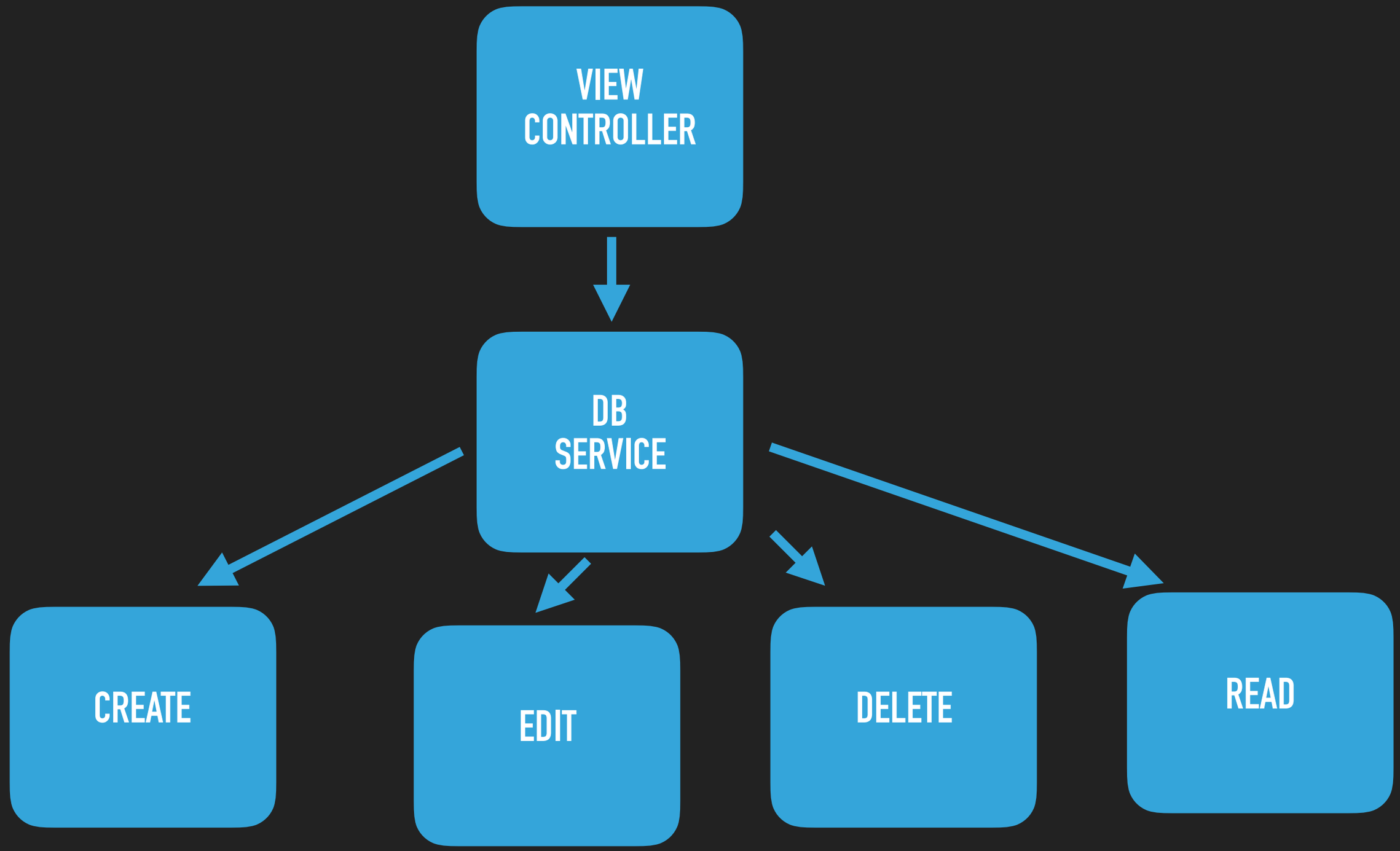
SO OUR WINNER IS

SO OUR WINNER IS

FIRST THING FIRST



WOULDN'T IT BE COOL IF WE CAN CHANGE DB WITH ONE LINE OF CODE?



MODEL CREATION

```
@objcMembers
class Restaurant: Object {
    dynamic var name: String = ""
    dynamic var rating: Int = 0
}
```

Dynamic and @objc is needed
to sync properties with DB

MODEL SAVING

```swift
// Create it as any other Swift object
let mcDonalds = Restaurant()
mcDonalds.name = "McDonald's"
mcDonalds.rating = 5

// That's object to manage Realm database (it's actually singletone)
let realm = try! Realm()

// This saves object in DB and from now on tracks it
try? realm.write {
    realm.add(mcDonalds)
}
```

Every write, edit, delete should
be inside write block

MODEL UPDATE

```
try? realm.write {
    mcDonalds.rating = 4
}
```

If object is managed by Realm every change of it should look like this, otherwise it crashes

MODEL DELETE

```
try? realm.write {
    realm.delete(mcDonalds)
}

print(mcDonalds.name)
```

This will crash application as object is invalidated

MODEL RETRIEVING

```swift
// Fetches all restaurants
let allRestaurants = realm.objects(Restaurant.self)

// Fetches restaurants with rating 4 and higher
let topRestaurants = realm.objects(Restaurant.self)
    .filter("rating >= 4")

// Sort restaurants by price category (more expensive first)
let sortedRestaurants = realm.objects(Restaurant.self)
    .sorted(by: { $0.priceCategory > $1.priceCategory })
```

Such operations are very fast

SUPPORTED DATA TYPES

- Bool
- Int
- Double, Float
- String
- Date
- Data

OPTIONALS

- Data, Date and String can be just optional (String?)
- Other types should be wrapped inside RealmOptional

PRIMARY KEY

```
@objcMembers
class Restaurant: Object {
    dynamic var id: Int = 0
    dynamic var name: String = ""
    dynamic var rating: Int = 0

    override class func primaryKey() -> String? {
        return "id"
    }
}
```

REFERENCES

- To give reference to some object create optional variable for that (one-to-one or many-to-one)

```
@objcMembers
class Restaurant: Object {
    dynamic var name: String = ""
    dynamic var rating: Int = 0
    dynamic var owner: Person?
}

@objcMembers
class Person: Object {
    dynamic var name: String = ""
}
```

LISTS

- You can hold list of objects

```
let employees = List<Person>()
```

INVERSE LISTS

- You can get inverse references for objects

```
@objcMembers
class Person: Object {
    dynamic var name: String = ""

    let restaurants = LinkingObjects(fromType: Restaurant.self, property: "owner")
}
```

Demo