What's New in Core Data

Session 242

Melissa Turner Software Engineer
Scott Perry Software Engineer

Roadmap

Query Generations

Concurrency

Core Data Stack Configuration

New API

Swift

Xcode Integration

Query Generations

Some problems are challenging

Faults in Core Data

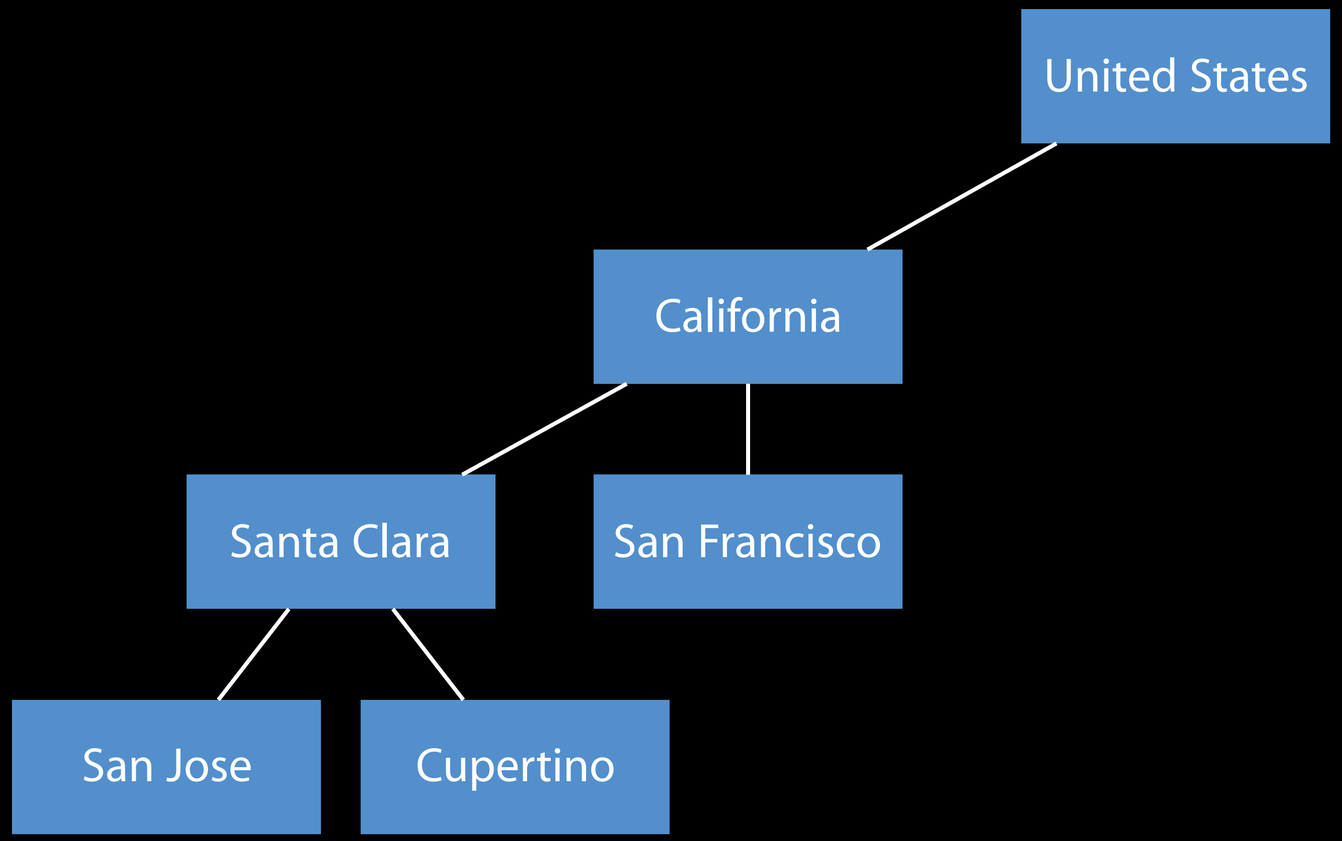
Faults are tricksy things

Managed objects

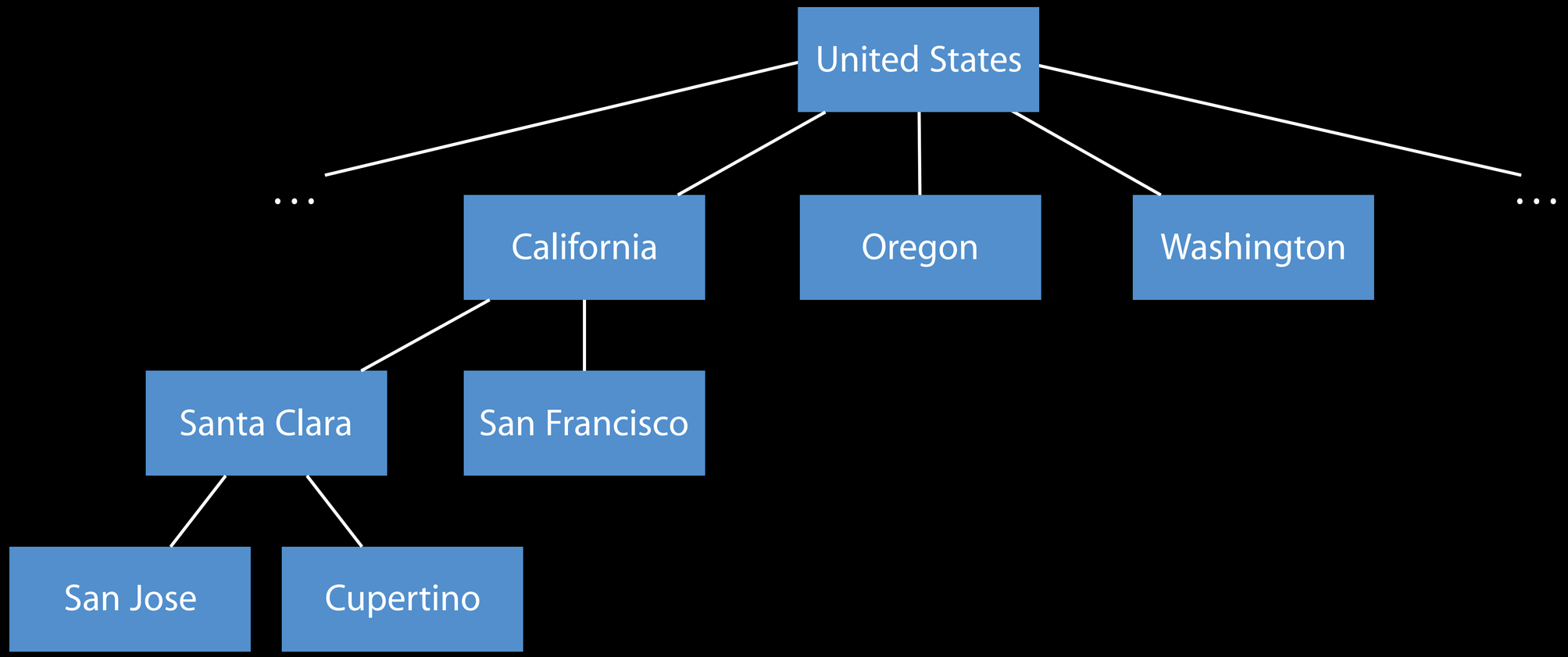
Relationships

Batch fetching result array

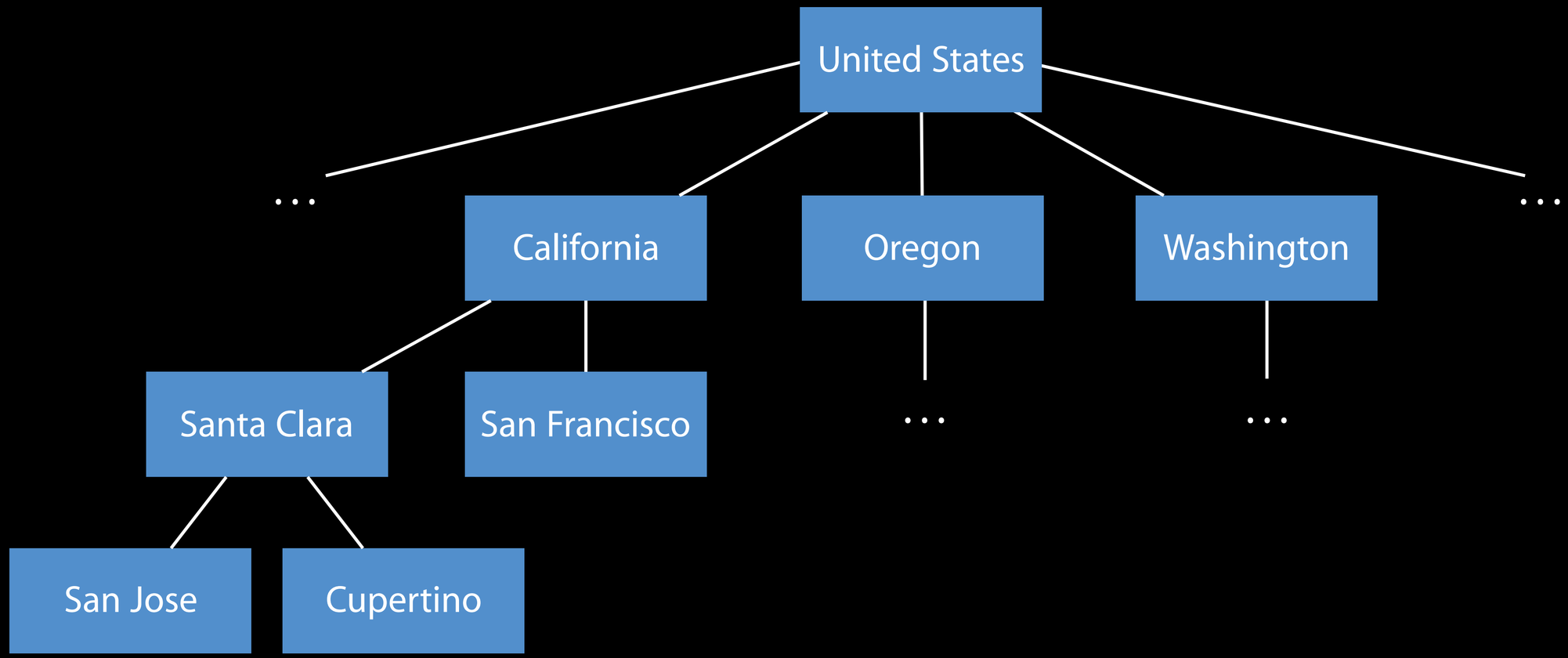
Faults in Pictures



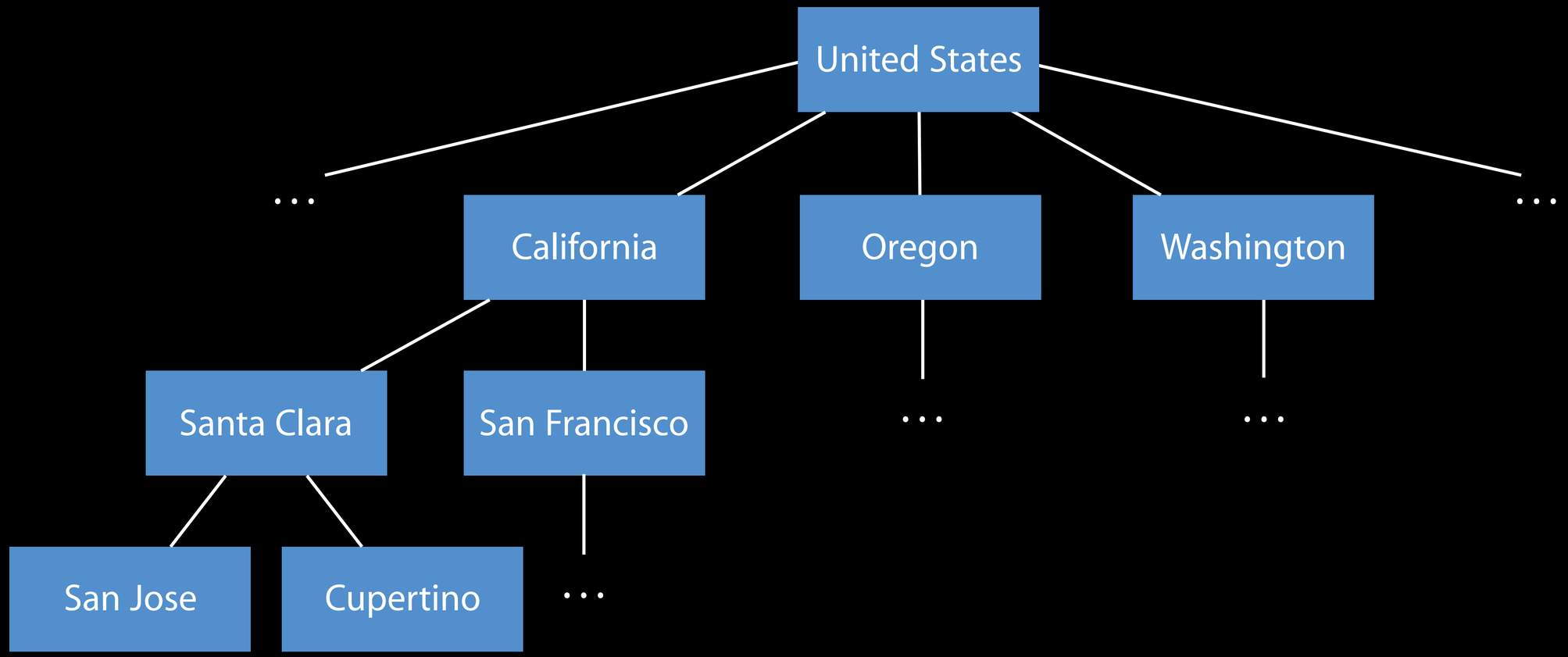
Faults in Pictures



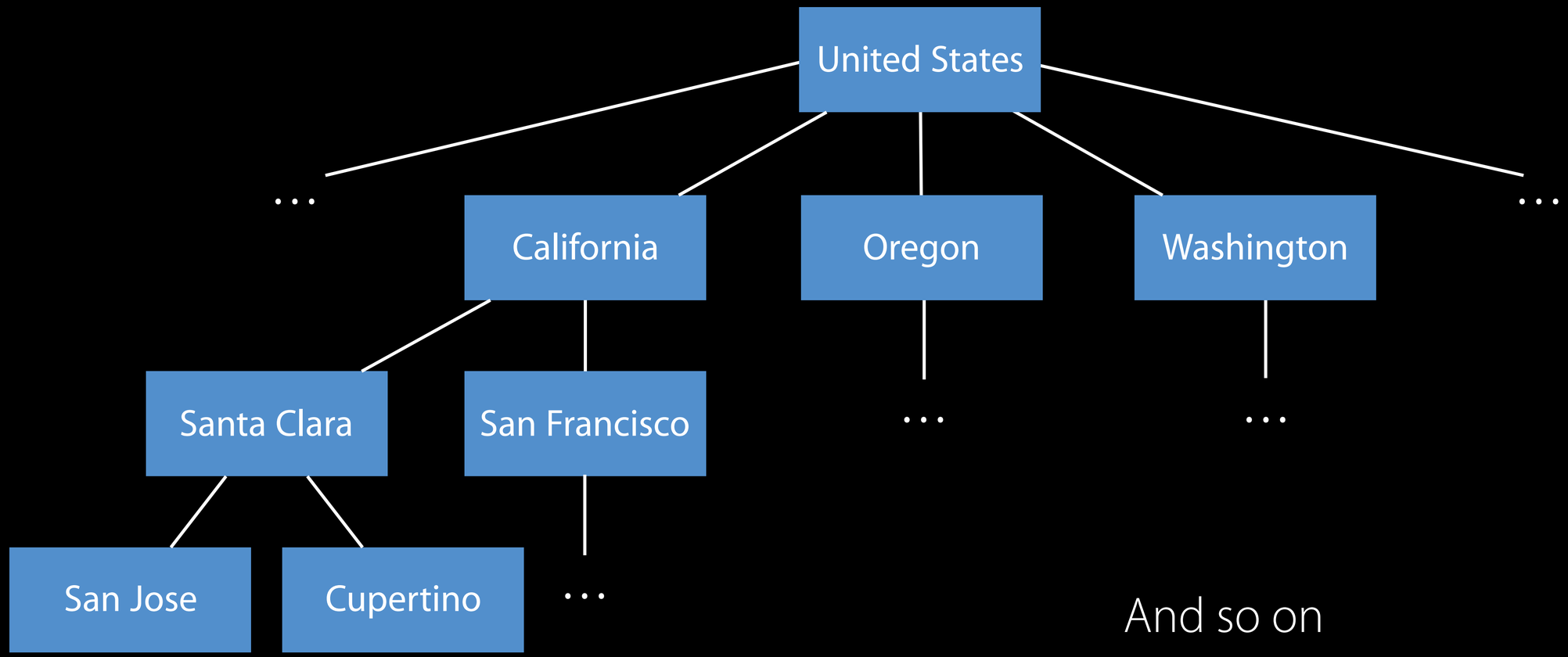
Faults in Pictures



Faults in Pictures

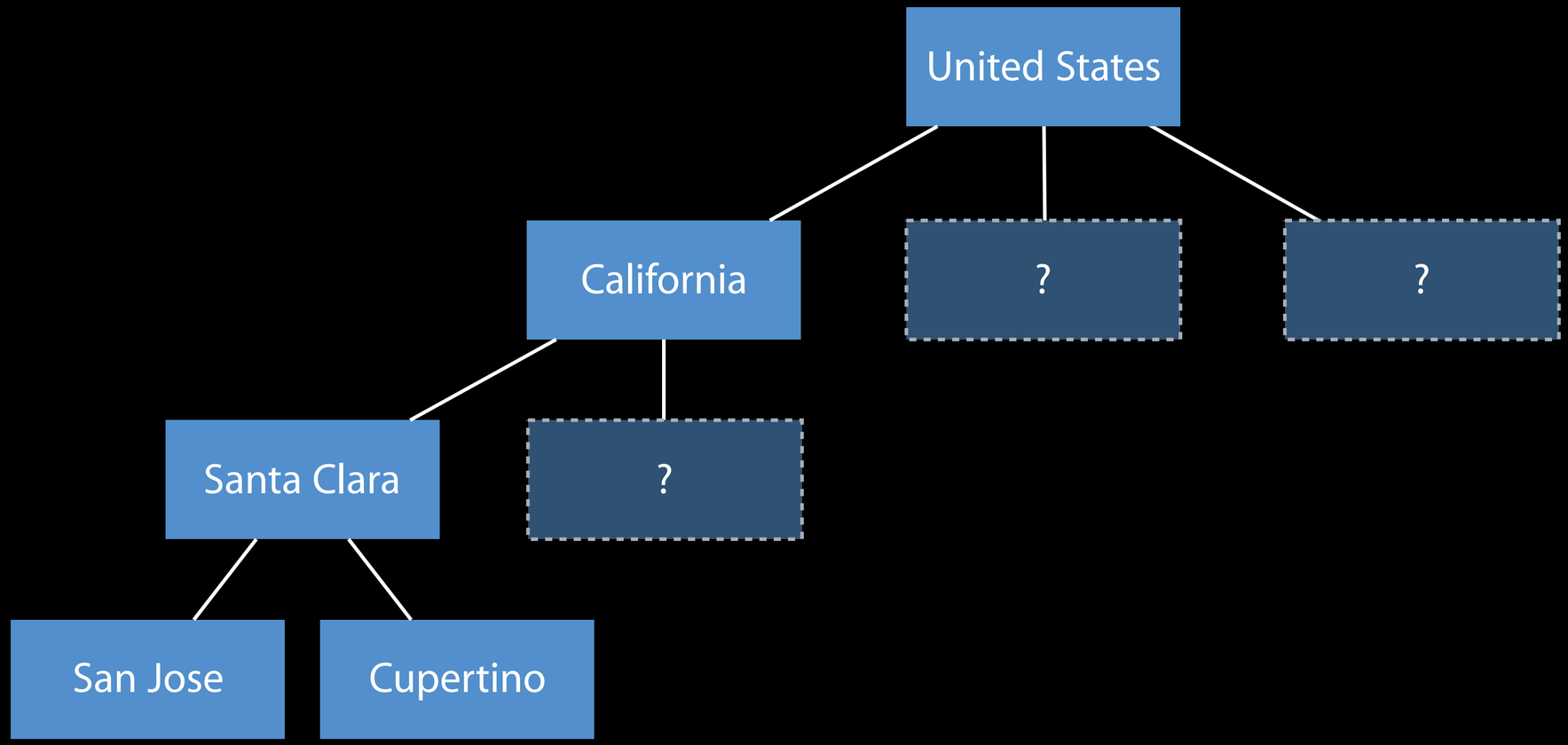


Faults in Pictures

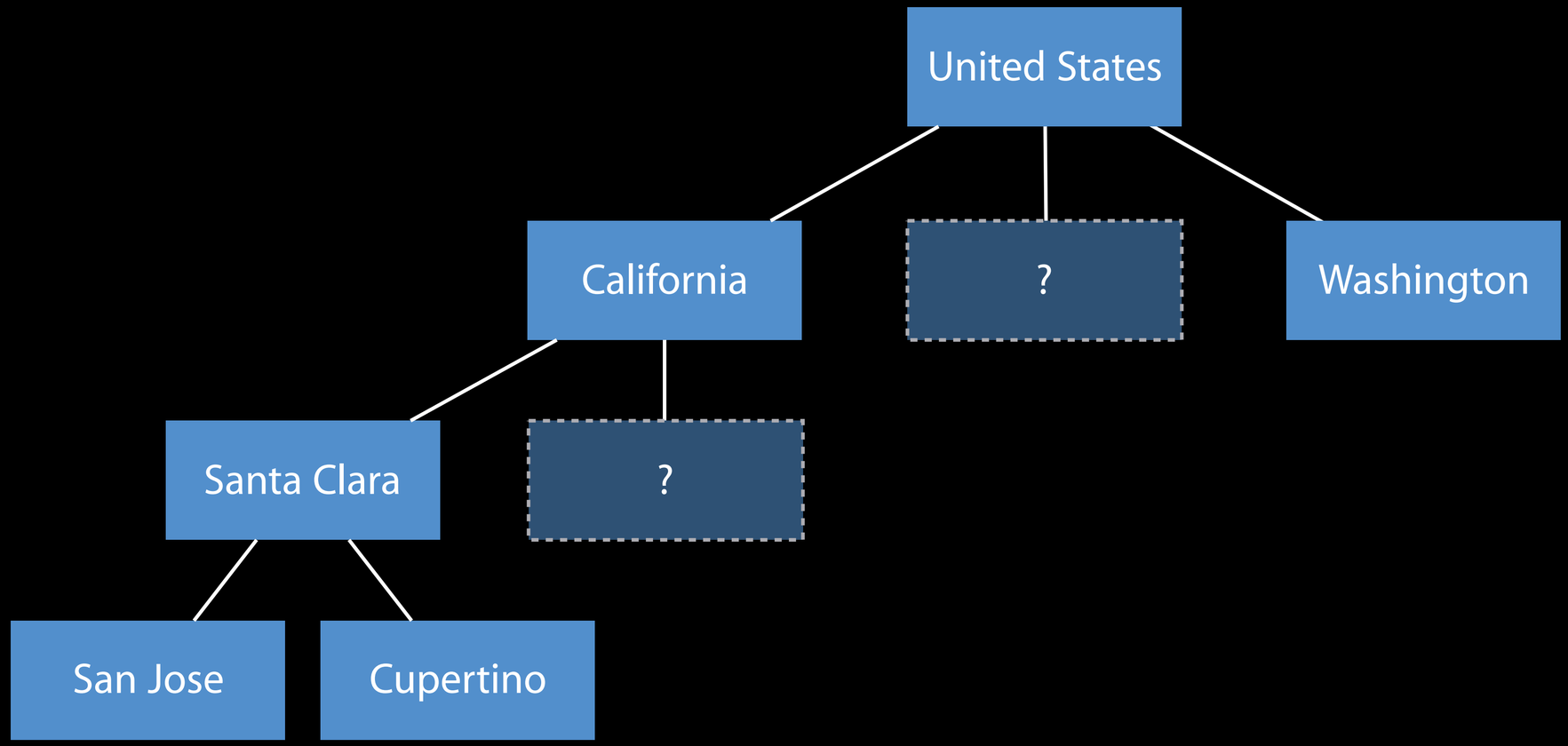


And so on

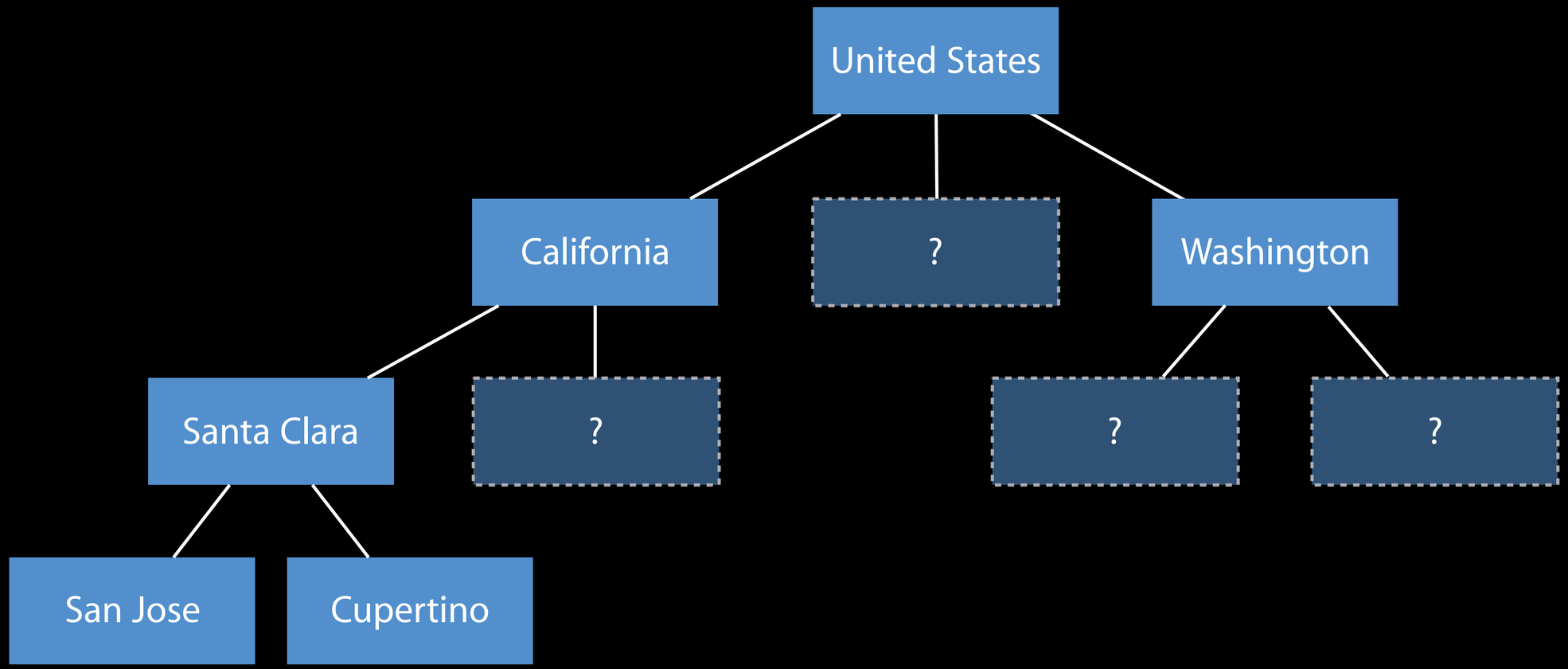
Faults in Pictures



Faults in Pictures



Faults in Pictures



Faults

A rose by any other name

Futures

Promises

Lazy loading

Why Faults?

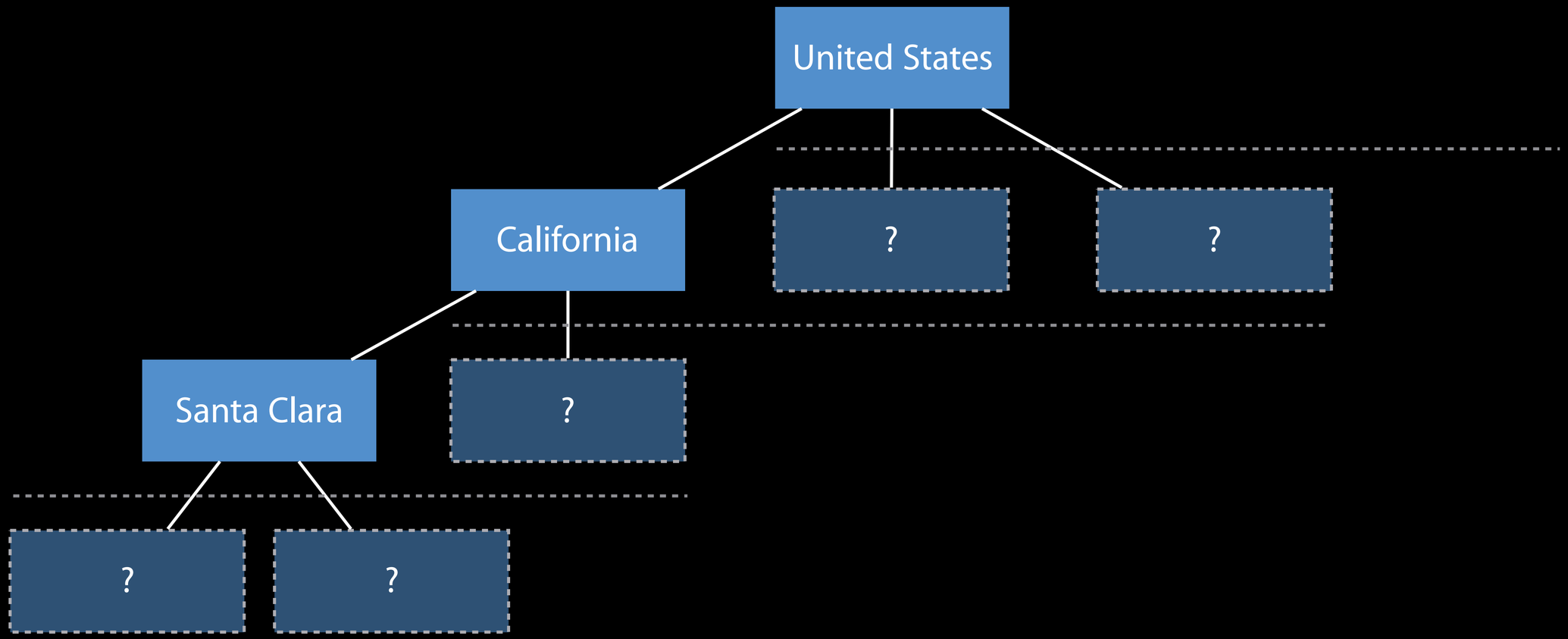
Zero work is optimally efficient in some cases

Avoid I/O

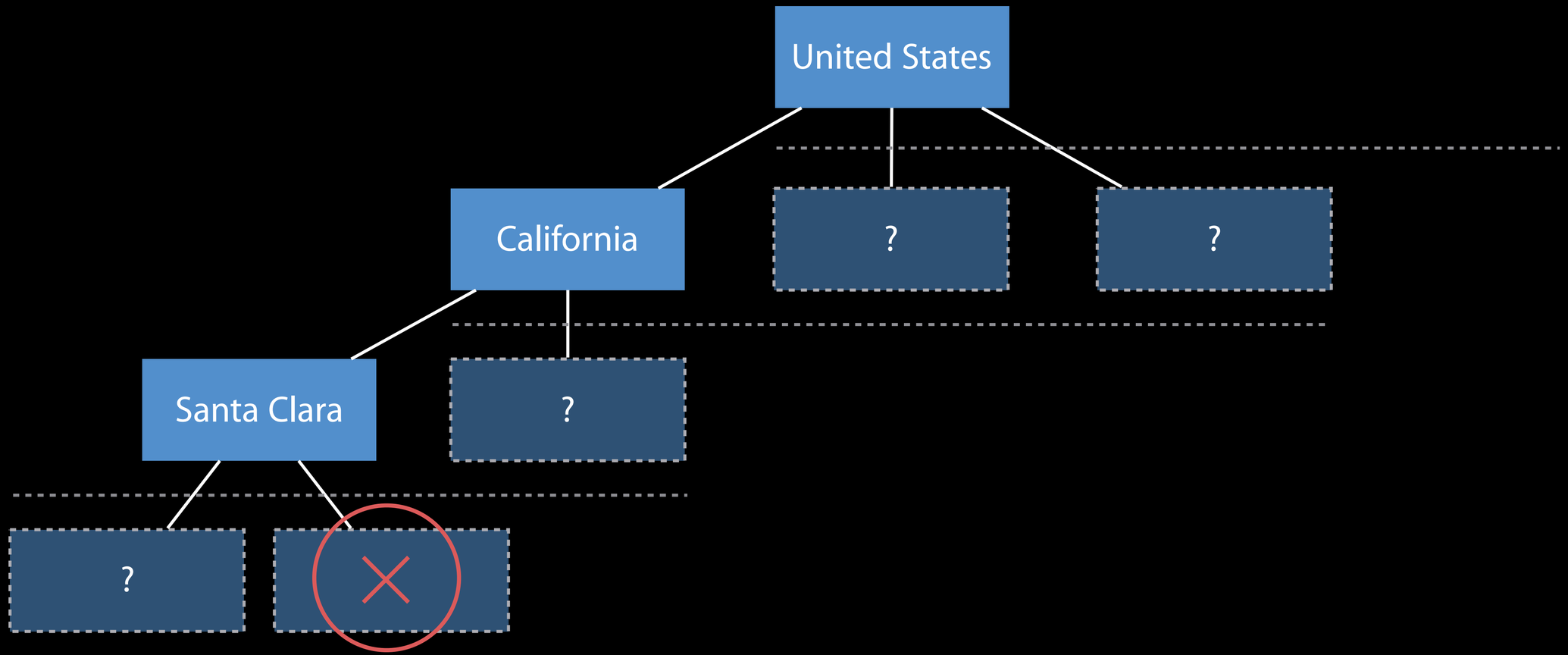
Avoid CPU usage

Avoid memory consumption

Faults in Pictures



Faults in Pictures



Why Faults?

Zero work is optimally efficient in some cases

Avoid I/O

Avoid CPU usage

Avoid memory consumption

Why Faults?

Zero work is optimally efficient in some cases

Avoid I/O

Avoid CPU usage

Avoid memory consumption

What happens when row data is deleted?

Why Faults?

Zero work is optimally efficient in some cases

Avoid I/O

Avoid CPU usage

Avoid memory consumption

What happens when row data is deleted?

- (Oops, did I need that after all?)

Handling Deletions

Current state of affairs

`managedObjectContext.shouldDeleteInaccessibleFaults`

Prefetch everything

- Extra I/O, CPU, memory

Write lots of code

Stepping Back

User viewing UI doesn't care

- Stability over immediacy
- Deferred/batched UI updates

Stepping Back

User viewing UI doesn't care

- Stability over immediacy
- Deferred/batched UI updates

User saving data doesn't care

- This is why we have merge policies

What if … ?

What if . . . ?

Provide UI with stable data?

What if . . . ?

Provide UI with stable data?

Handle changes deterministically?

What if … ?

```
CoreData could not fulfill a fault for <x-coredata://
  855894C5-1826-4344-9B1C-95190FF77733/Employee/p1>
```

AKA: The problem

Query Generations

We can be tricksier

All reads from a context see a single generation of data

Advance at predictable times

Never see “could not fulfill a fault”

Query Generations

We can be tricksier

All reads from a context see a single generation of data

Advance at predictable times

Never see “could not fulfill a fault”

Efficiently

For Visual Learners

Your App

Database

id : 1

name : ‘fred’

For Visual Learners

Your App

Database

x : 1

y : ‘a’

For Visual Learners

Your App

Gen 1

x : 1

y : ‘a’

For Visual Learners

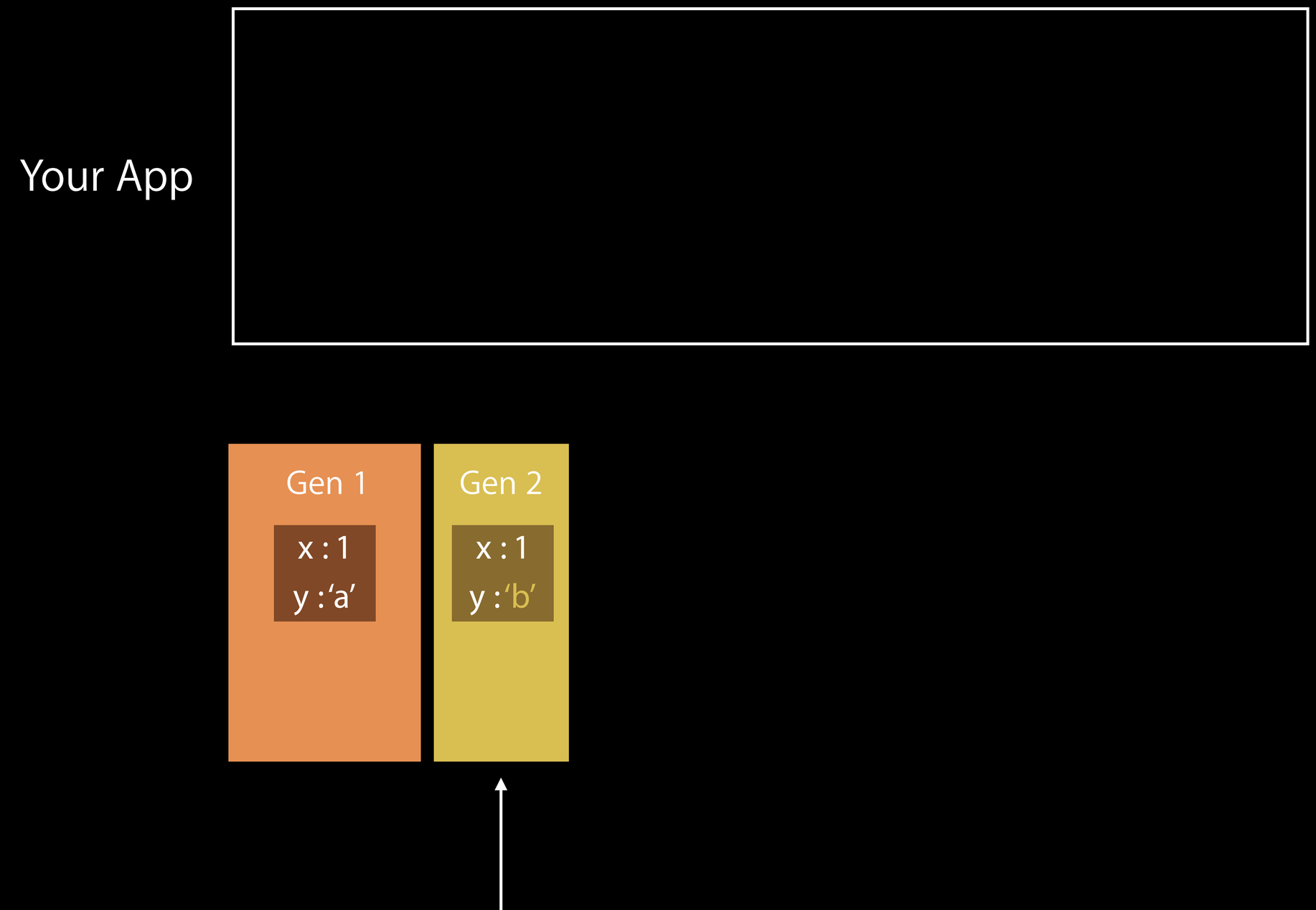
Your App

Gen 1

x : 1

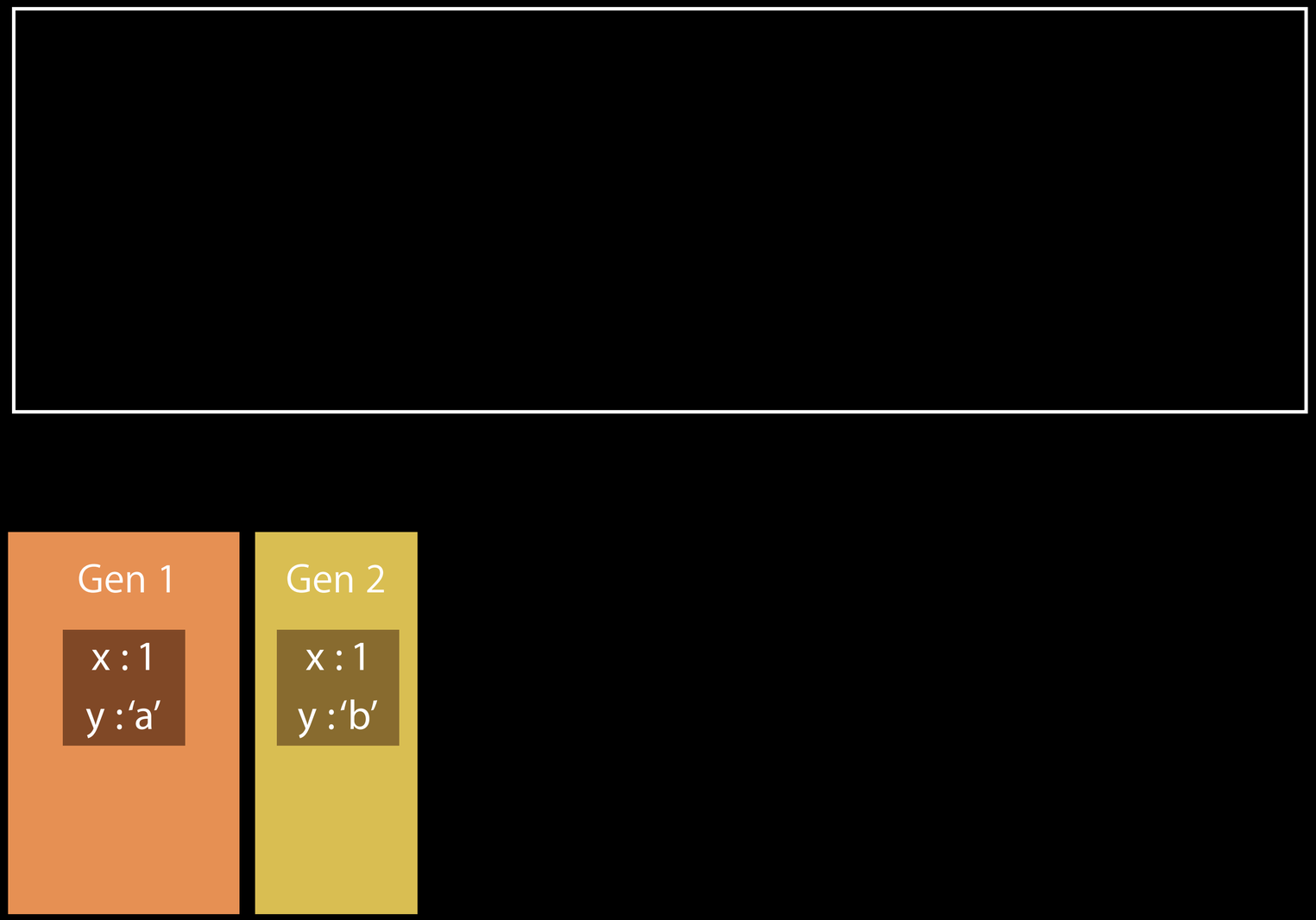
y : ‘a’

For Visual Learners



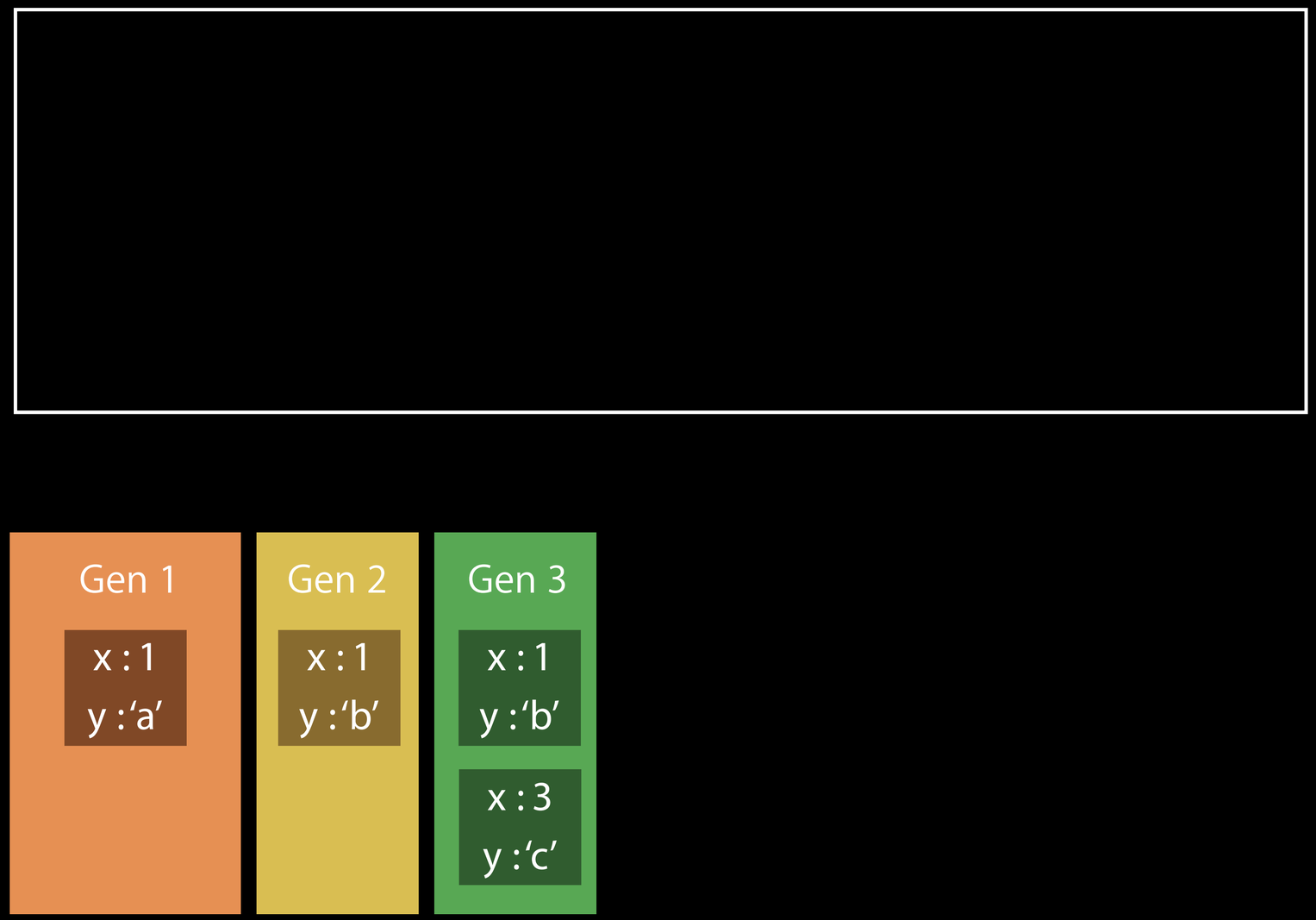
For Visual Learners

Your App

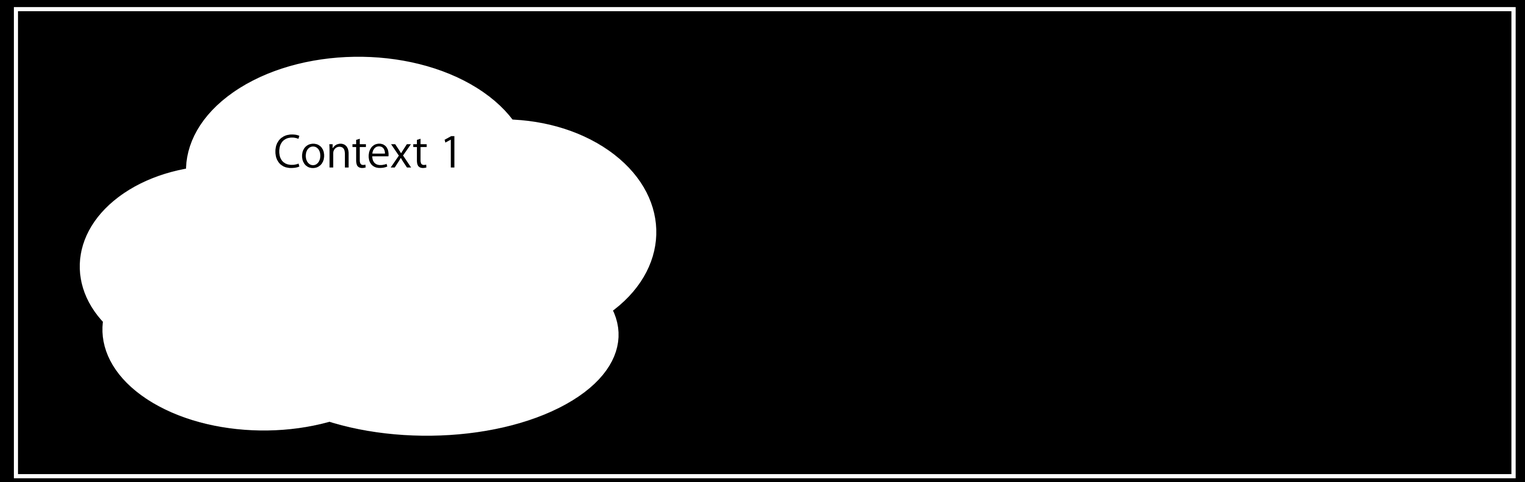


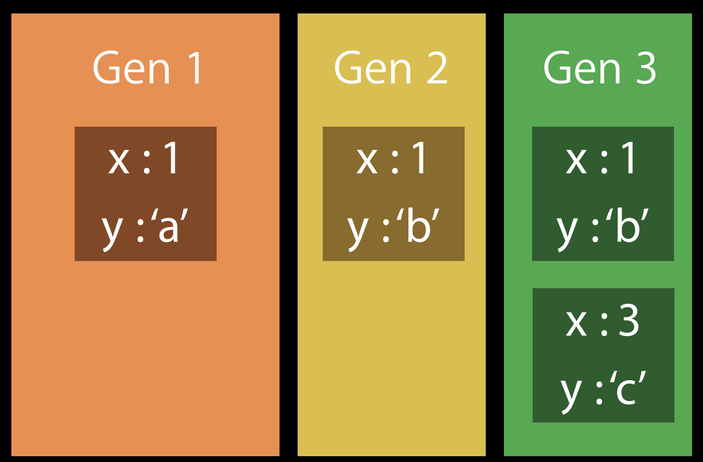
For Visual Learners

Your App

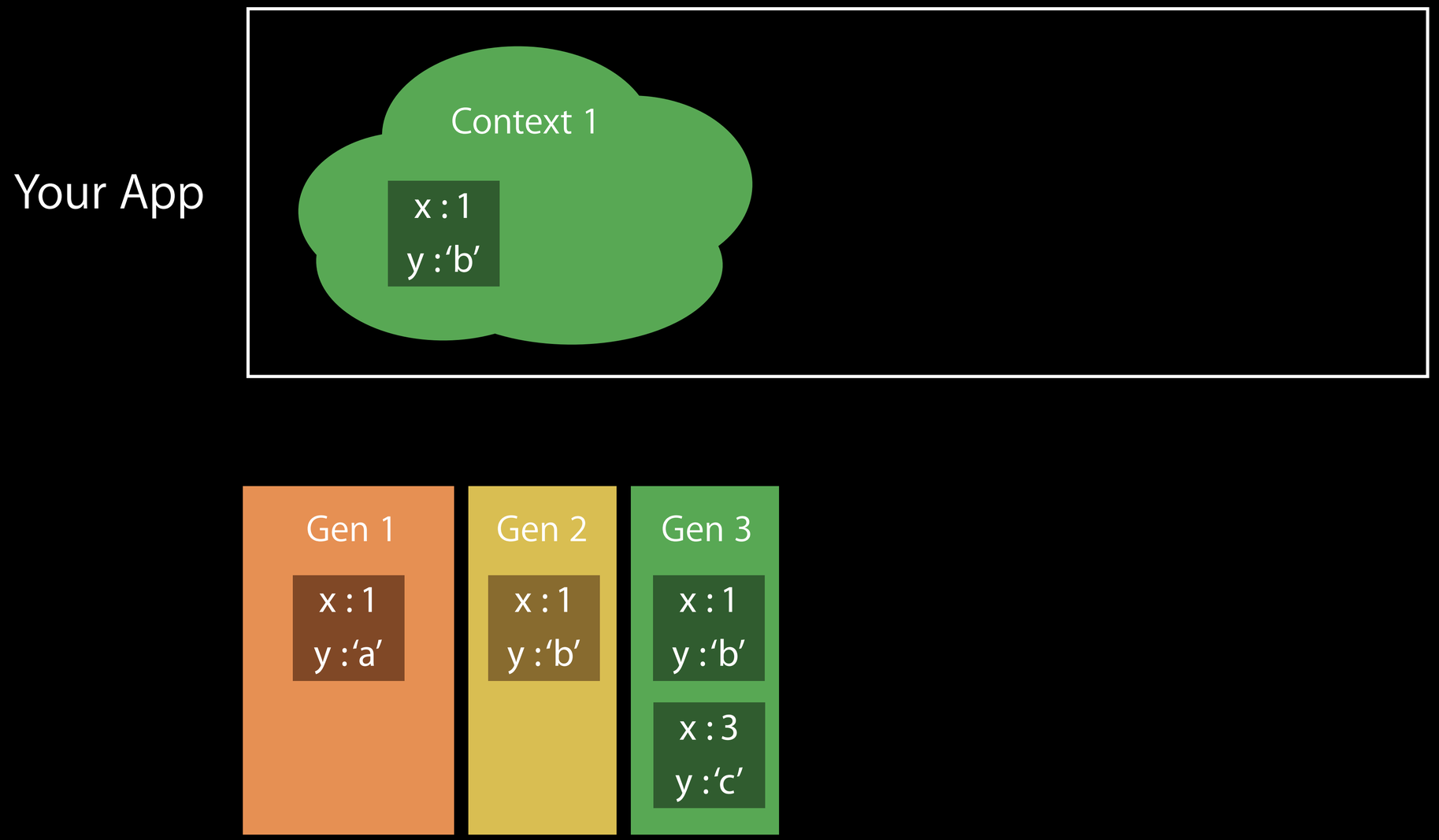


For Visual Learners

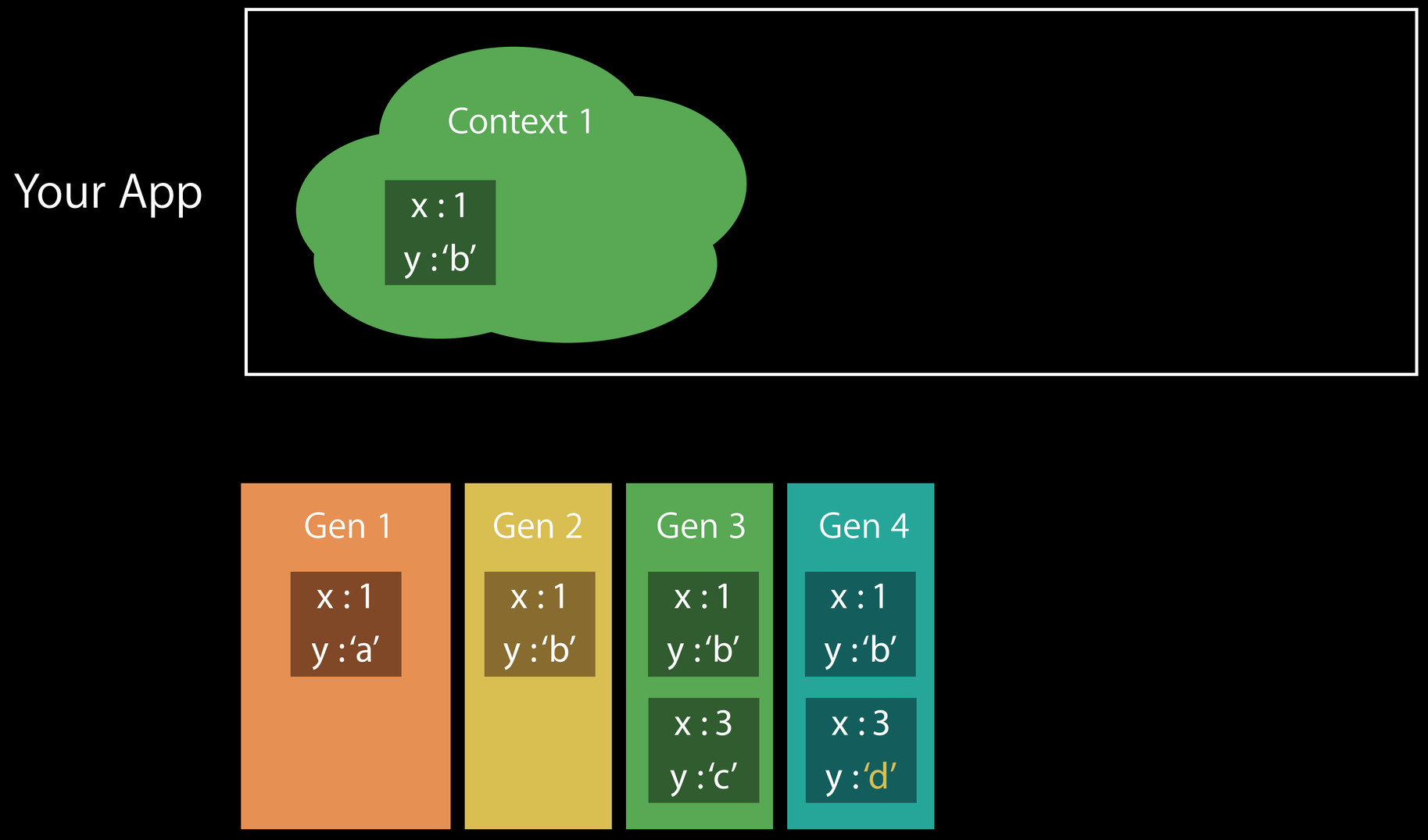




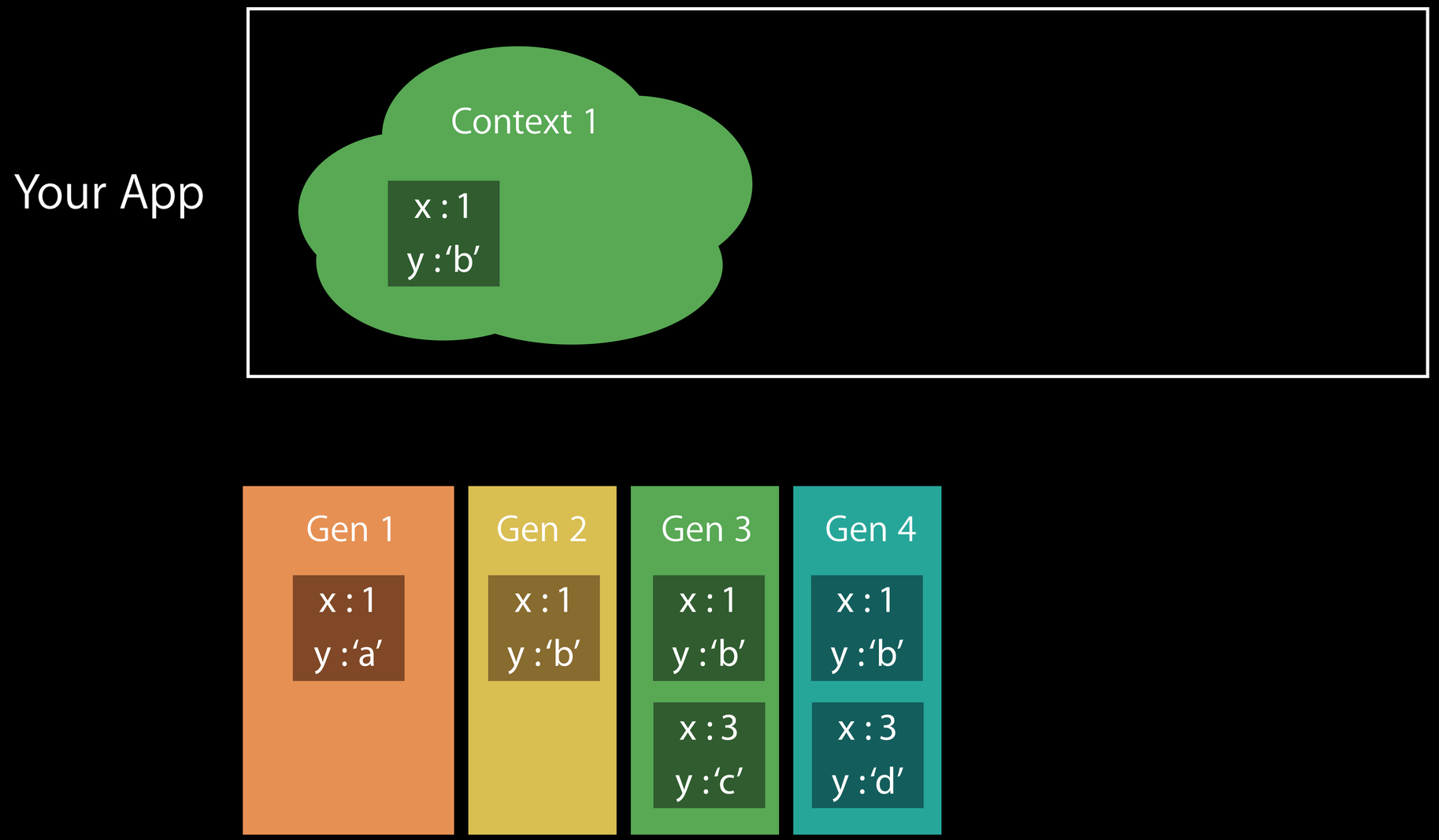
For Visual Learners



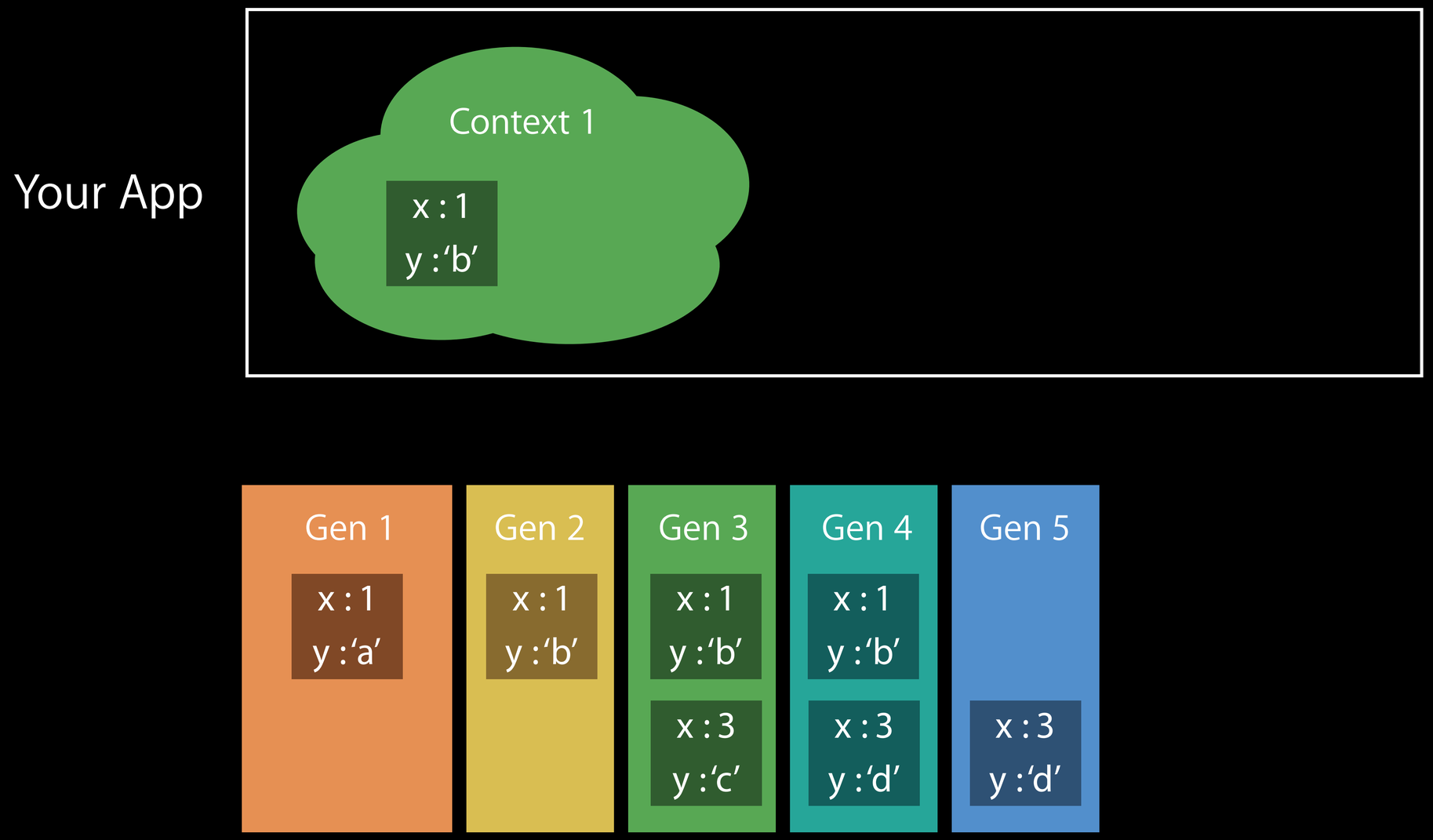
For Visual Learners



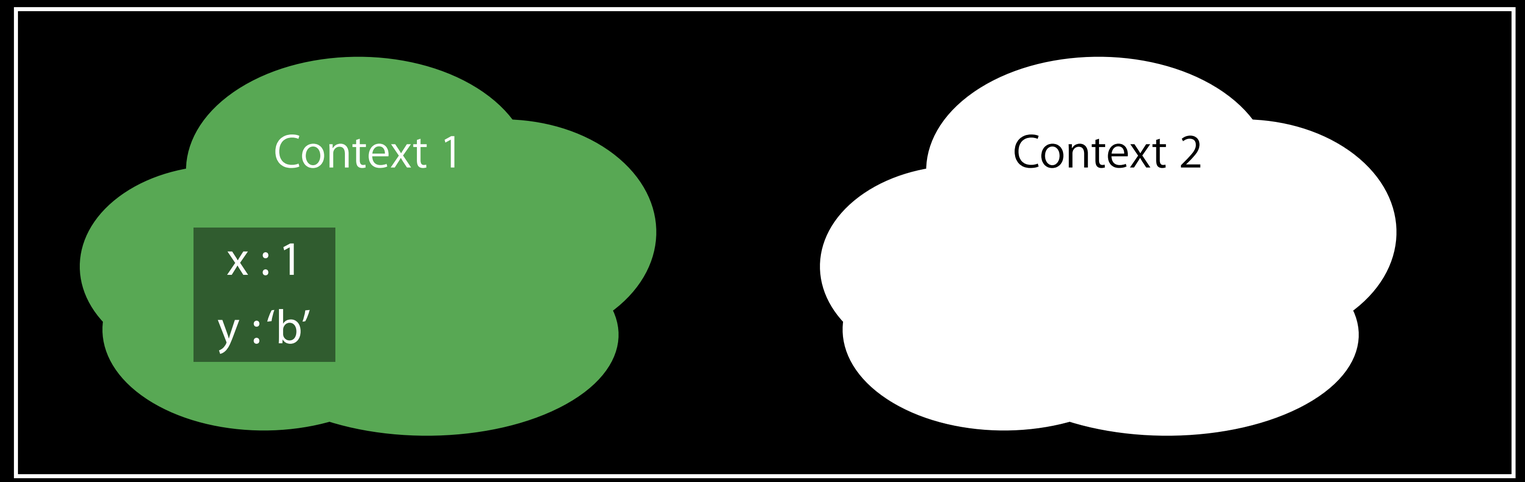
For Visual Learners

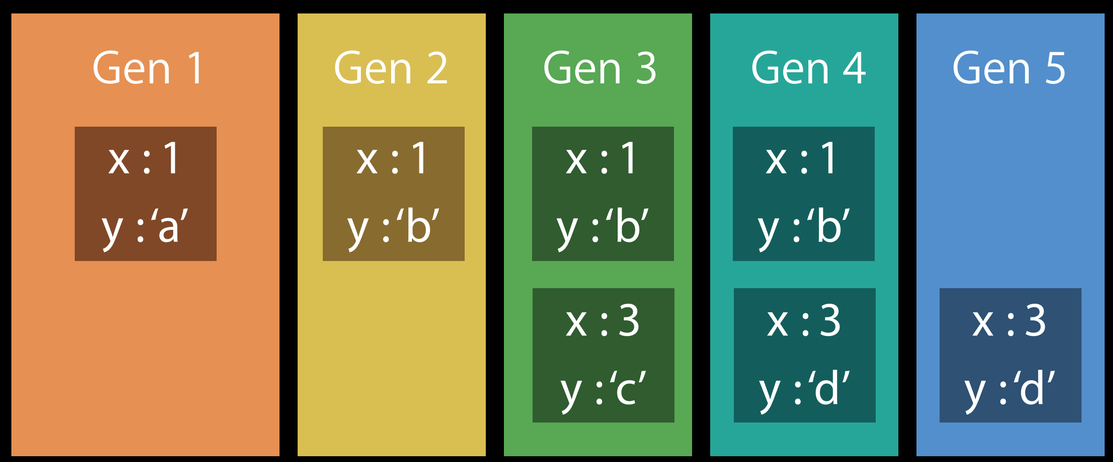


For Visual Learners

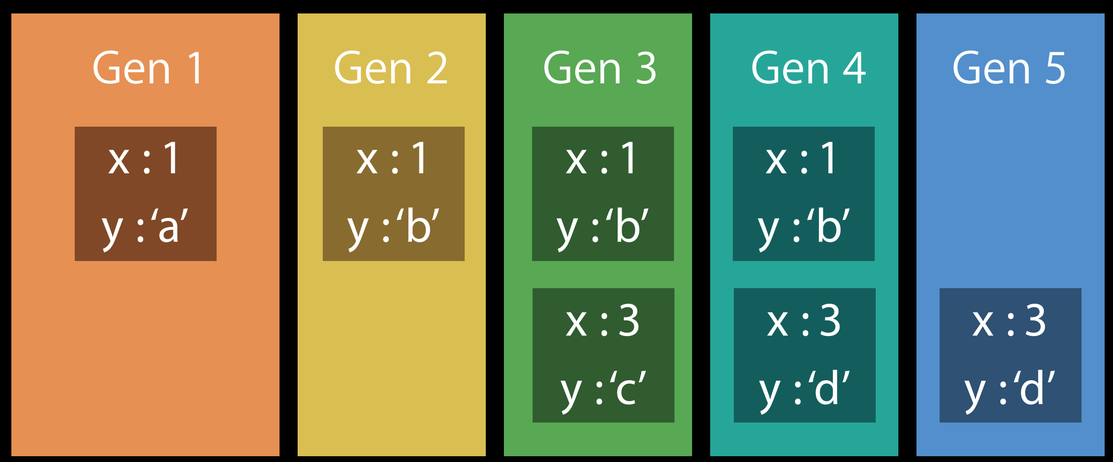


For Visual Learners

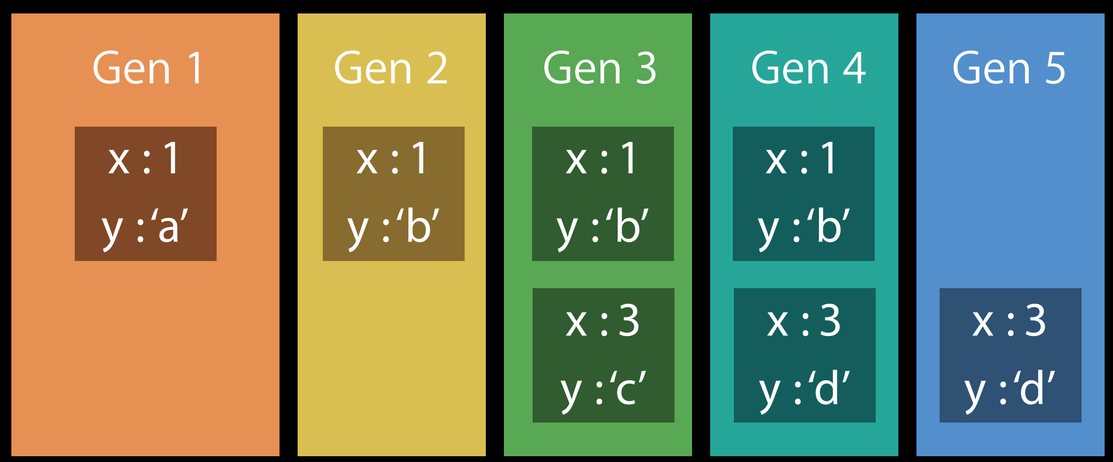




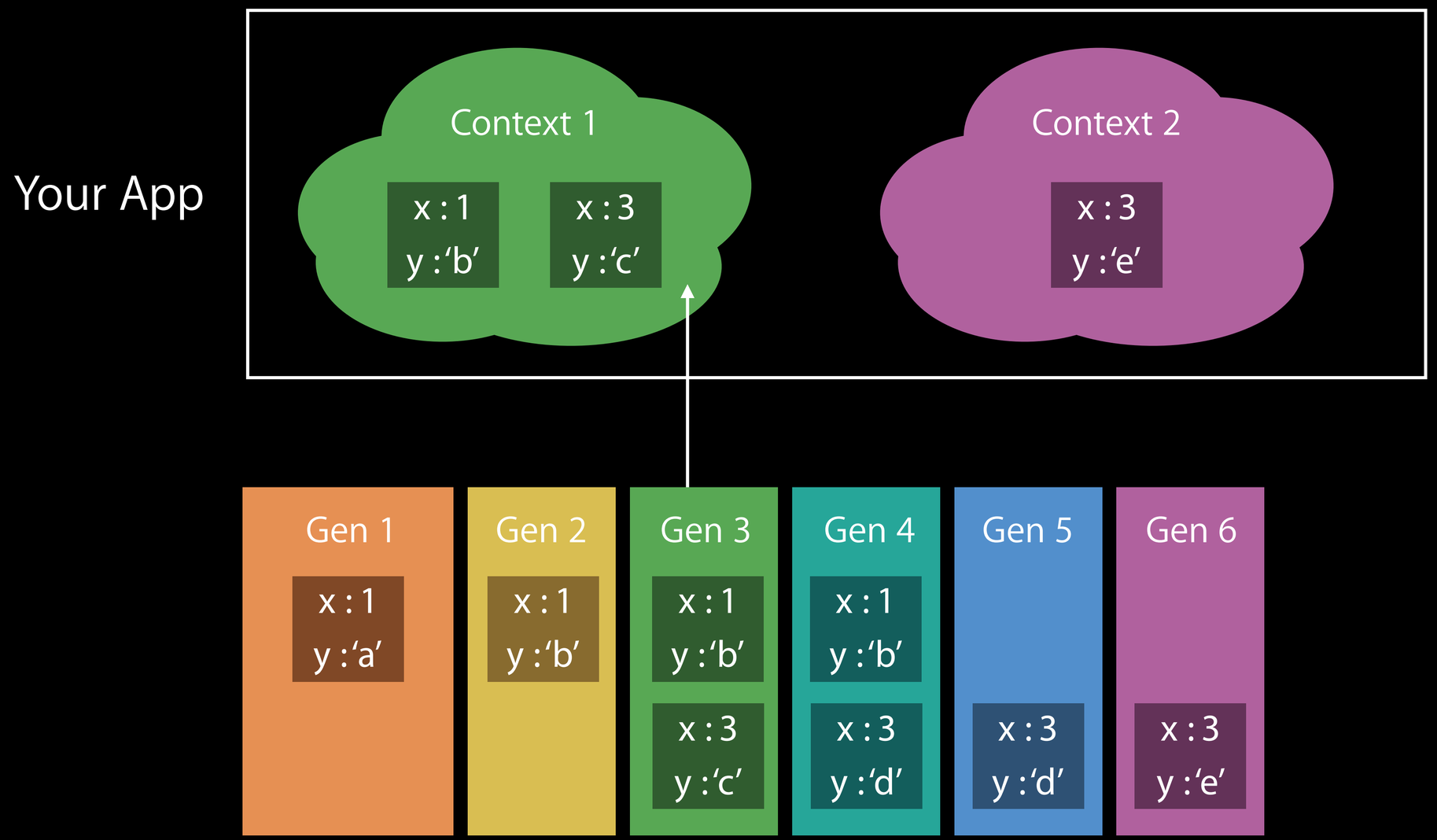
For Visual Learners



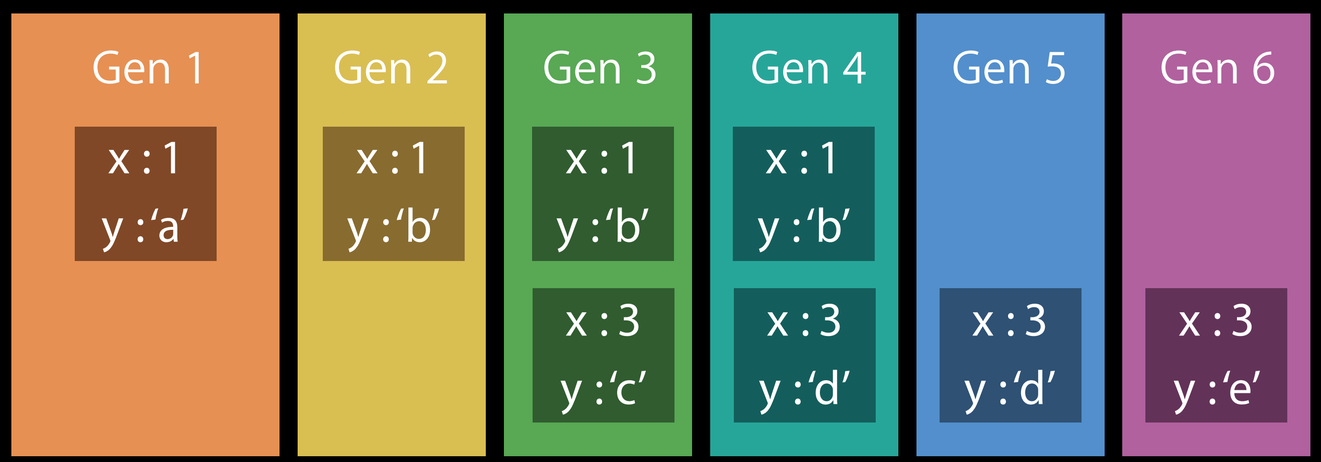
For Visual Learners



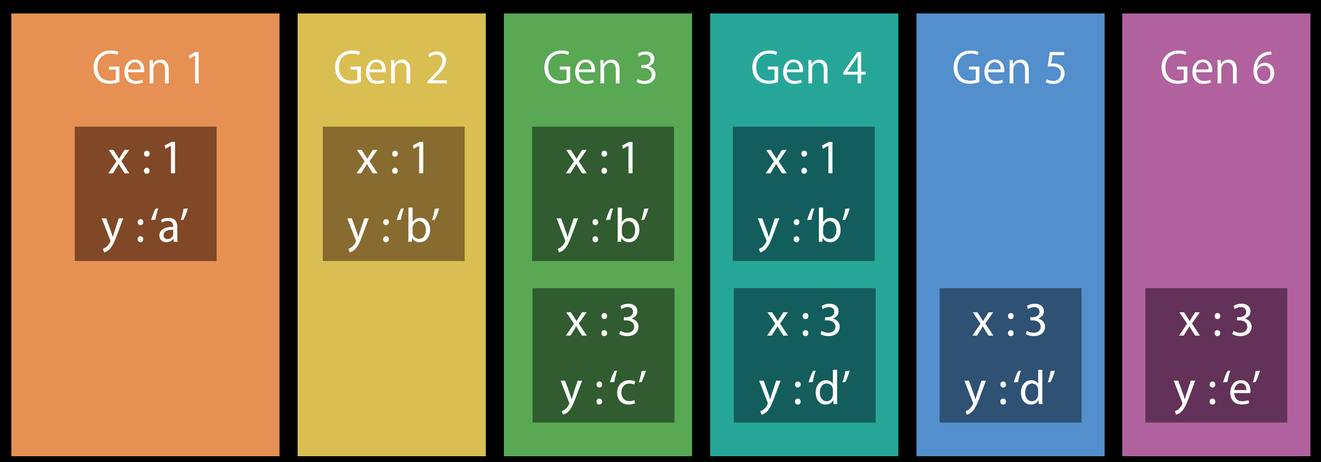
For Visual Learners



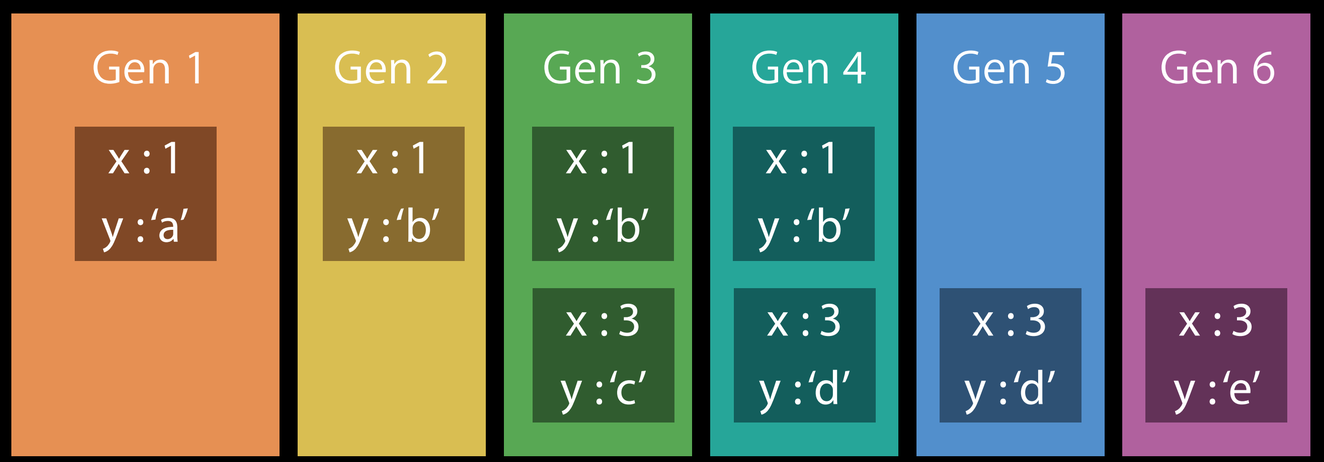
For Visual Learners



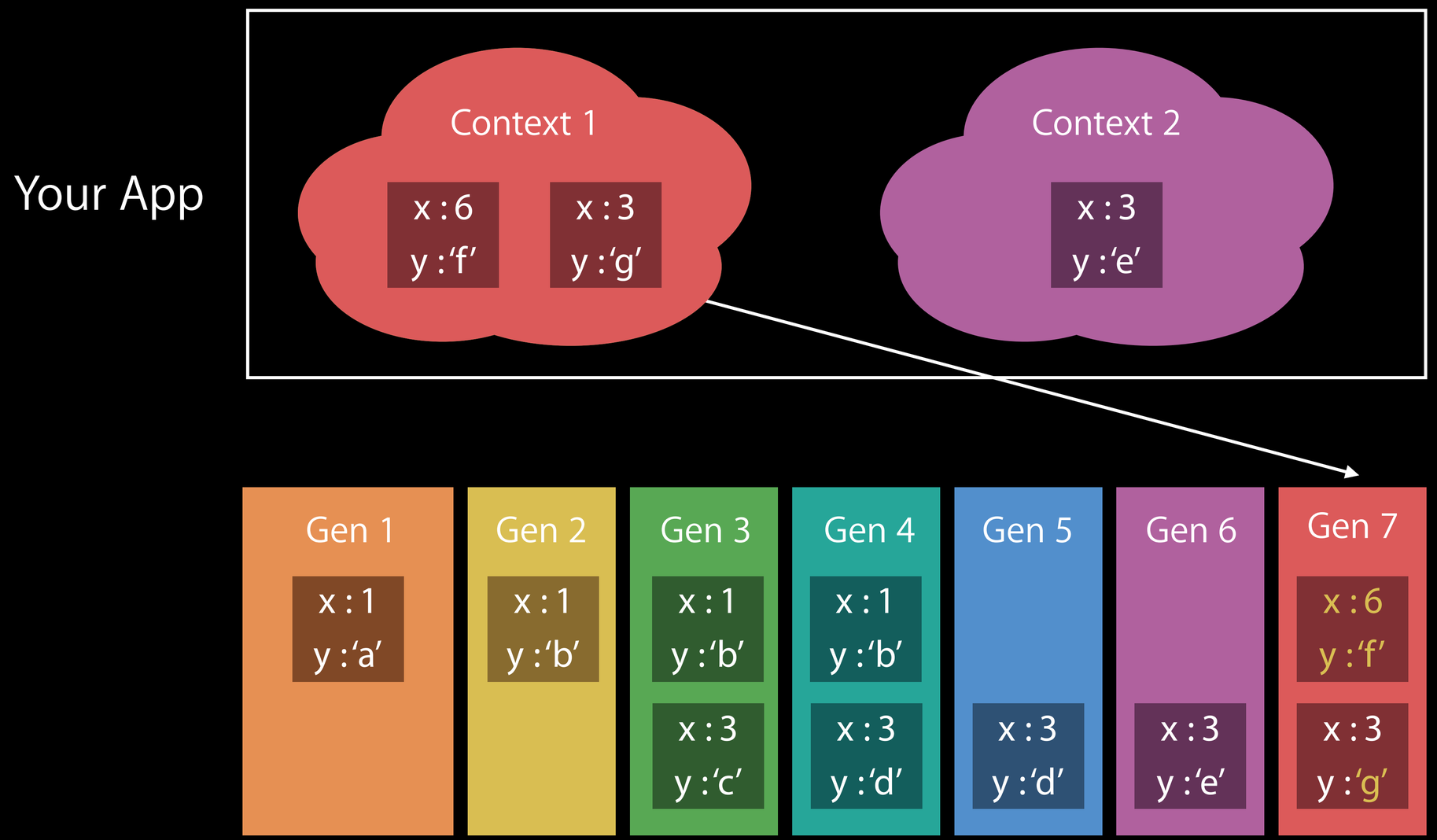
For Visual Learners



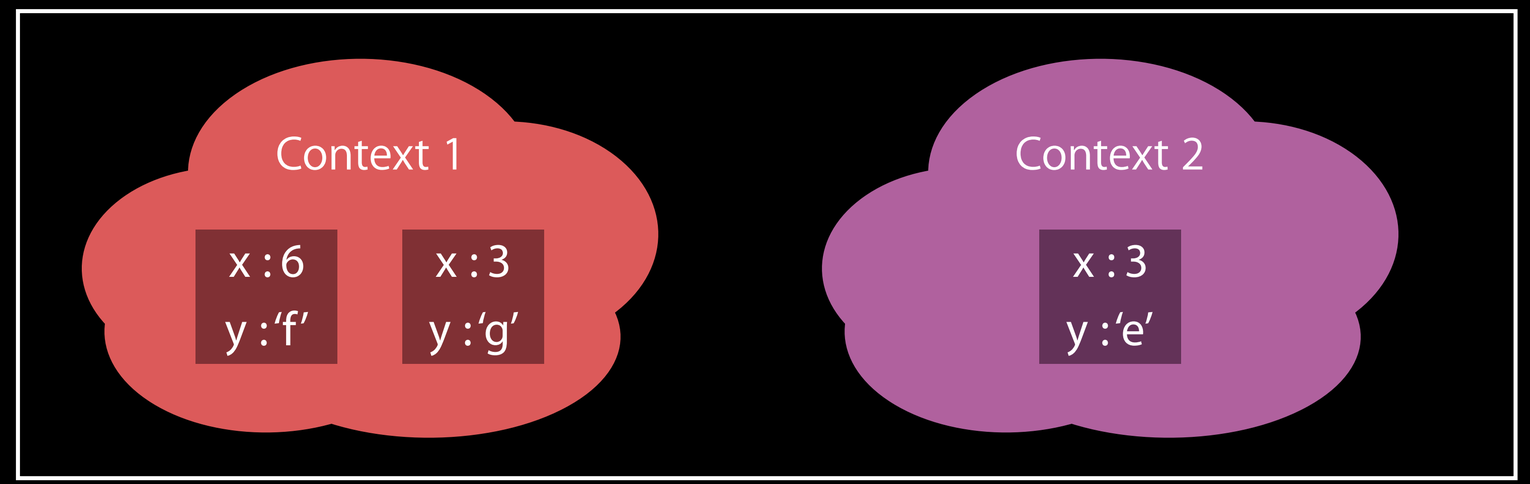
For Visual Learners

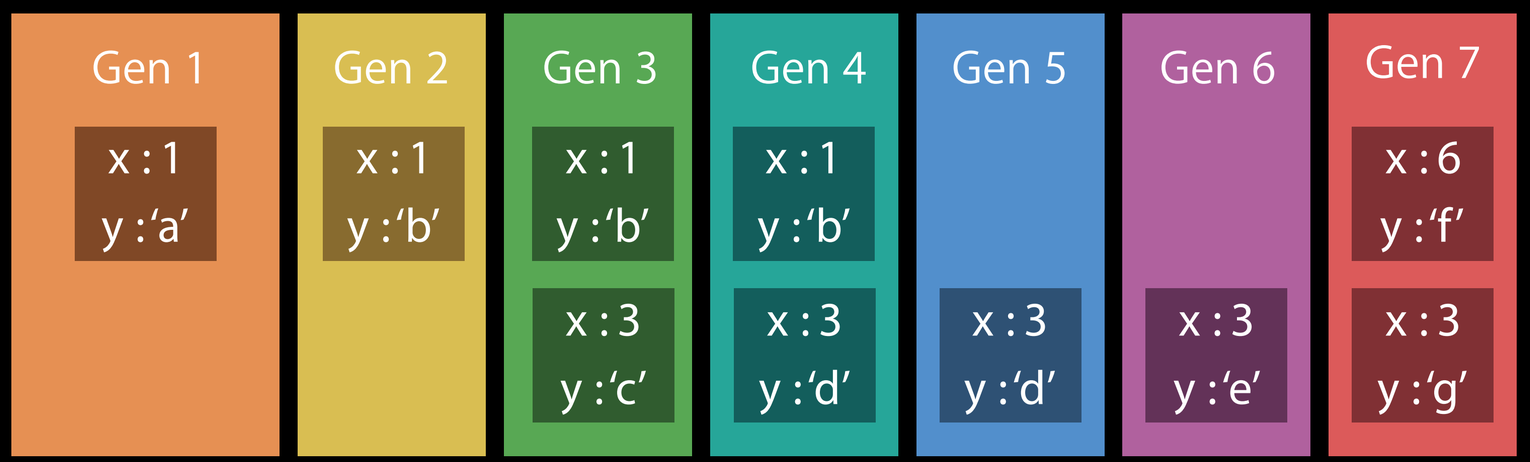


For Visual Learners

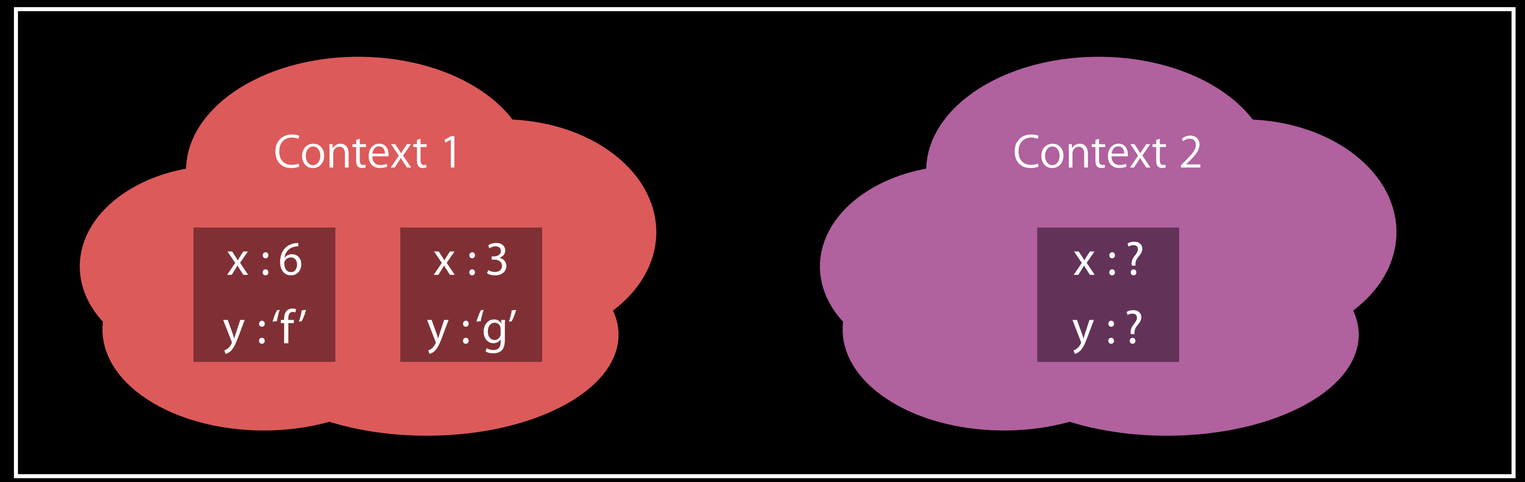


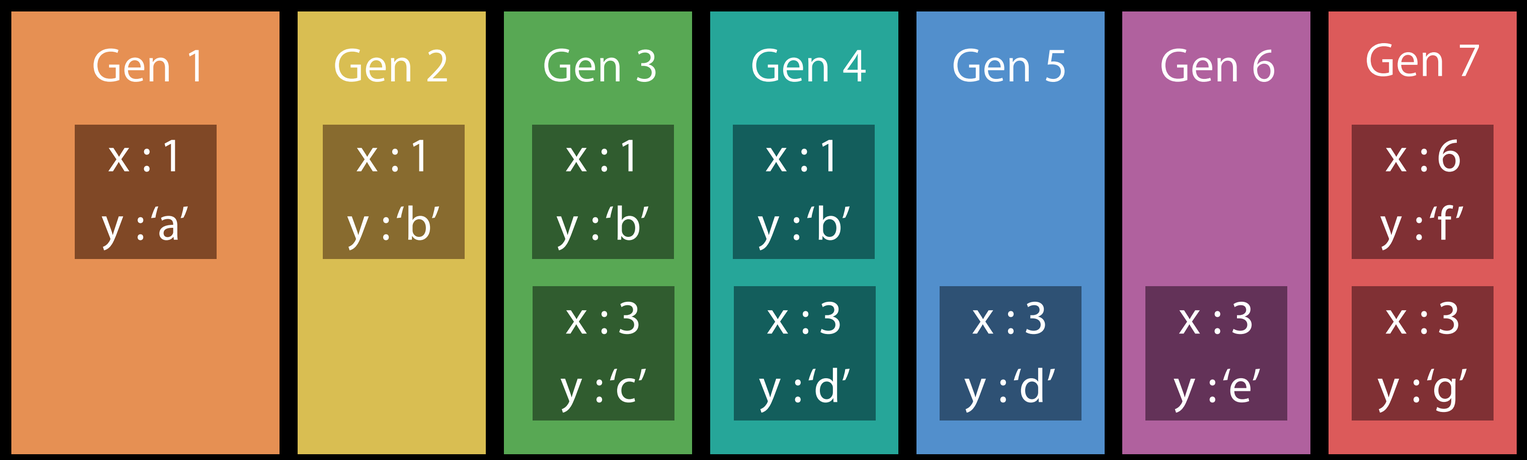
For Visual Learners



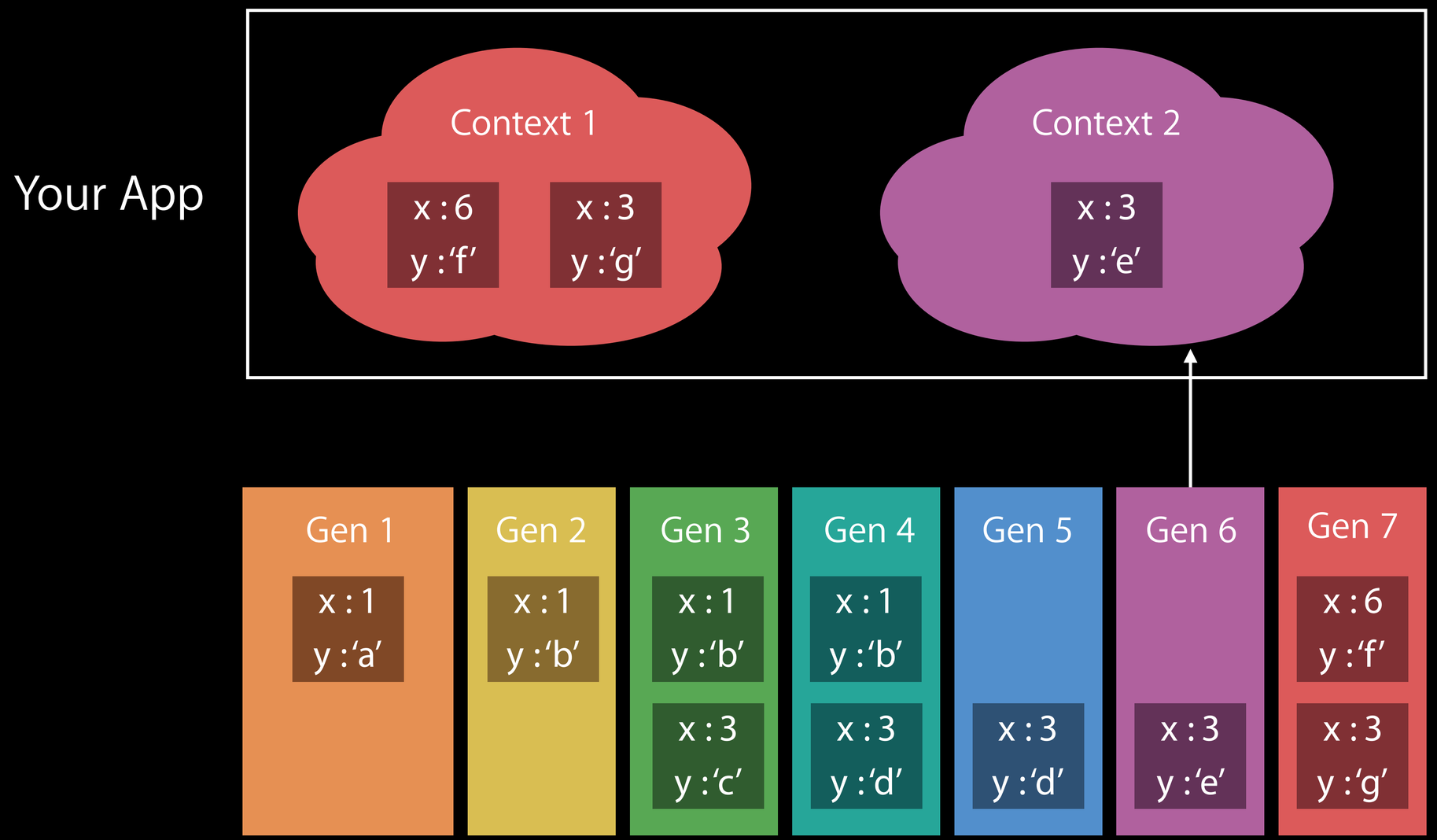


For Visual Learners





For Visual Learners



Benefits

Transactionality at context level

Isolation of work in peer contexts

Minimize speculative/preventative prefetching

Benefits

Transactionality at context level

Isolation of work in peer contexts

Minimize speculative/preventative prefetching

- Everyone wins!

Query Generations

The basics

Context chooses behavior

- Unpinned (current, default)
- Pin when data first loaded
- Pin to specific generation

Query Generations

The basics

Context chooses behavior

- Unpinned (current, default)
- Pin when data first loaded
- Pin to specific generation

Nested contexts always unpinned

- Inherit data from parent

Changing Generations

The basics

When explicitly updated

On save

- Linear stream of changes

During `managedObjectContext.mergeChanges()`

As a result of `managedObjectContext.reset()`

Answers

To questions you haven't thought to ask yet

Registered objects not refreshed on generation update

`managedObjectContext.fetch()`

`managedObjectContext.refreshAllObjects()`

You have control

Requirements

SQL store

WAL mode

Fails gracefully

- Non-SQL stores have a single “TOT” generation

Query Generations

Nuts and bolts: API

Opaque token to track generation

- New class `NSQueryGenerationToken`
- `NSQueryGenerationToken.current()`

Query Generations

Nuts and bolts: API

Opaque token to track generation

- New class `NSQueryGenerationToken`
- `NSQueryGenerationToken.current()`

Methods on NSManagedObjectContext that use it

- `managedObjectContext.queryGenerationToken`
- `managedObjectContext.setQueryGenerationFrom()`

Query Generations

Nuts and bolts: Corners

Generation will not include stores added after token creation

- No results from stores not in generation

Does not prevent you from removing stores from coordinator

- Error if no stores from generation remain

Concurrency

Something is always new

Concurrency in Core Data

Current stuff

Actor model for NSManagedObjectContext

- Use `perform()` , `performAndWait()`
- `confinementConcurrencyType` is deprecated

Actor model for NSPersistentStoreCoordinator

- Use `perform()` , `performAndWait()`

Coordinator serializes contexts' requests

And Now an Important Message

Why did stuff start breaking after I recompiled?

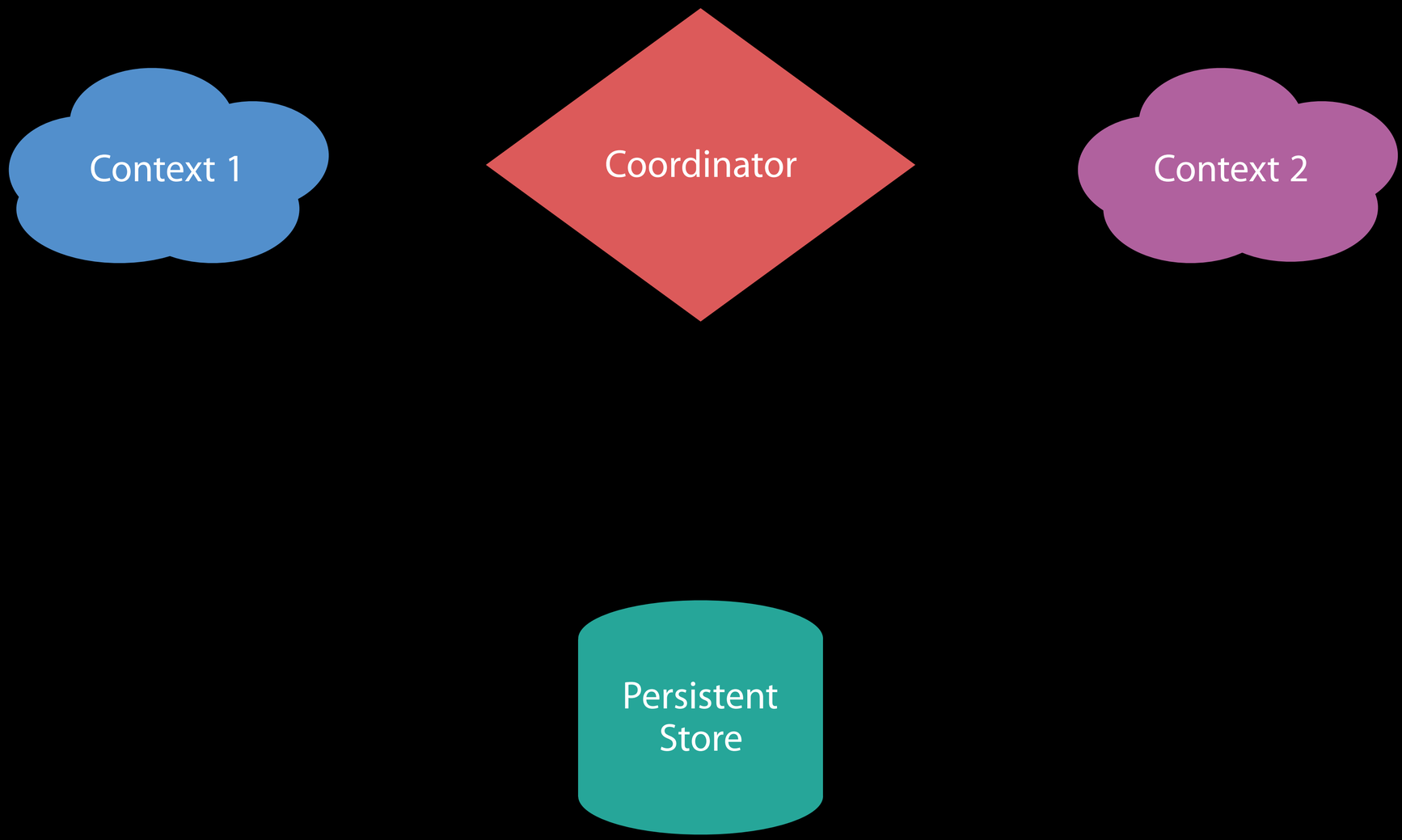
`-performBlockAndWait:` now has an autorelease pool

- Watch your NSError **

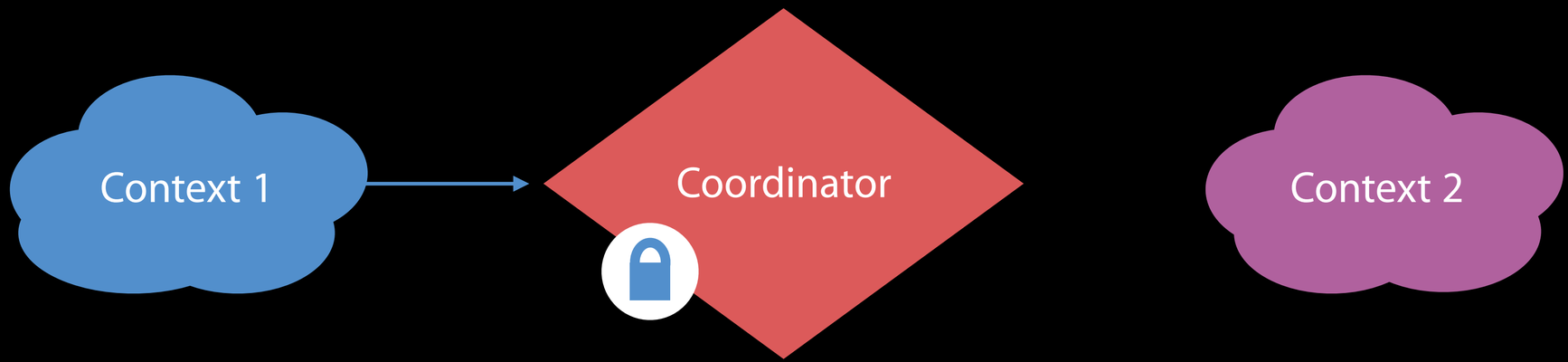
Affects MRR

Link time check

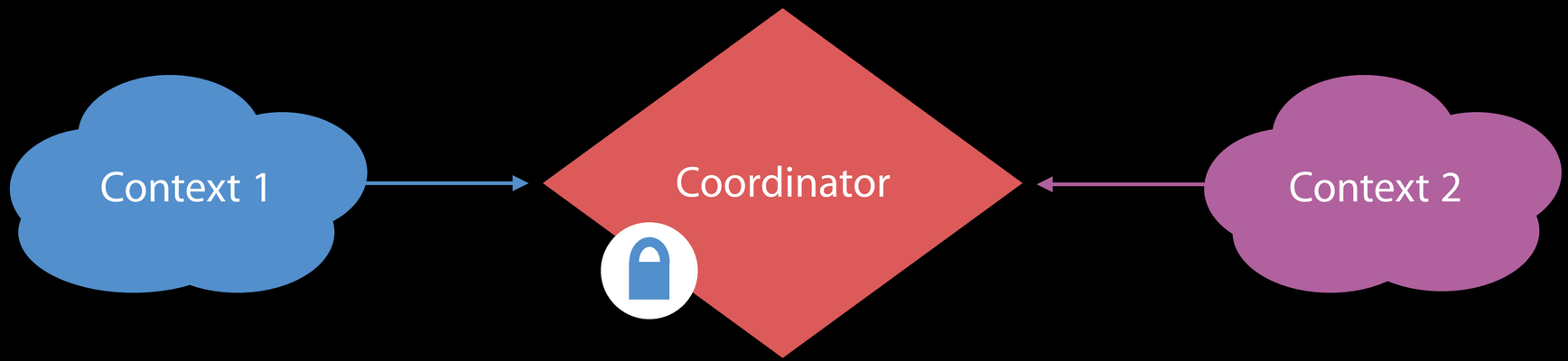
Current State of Affairs



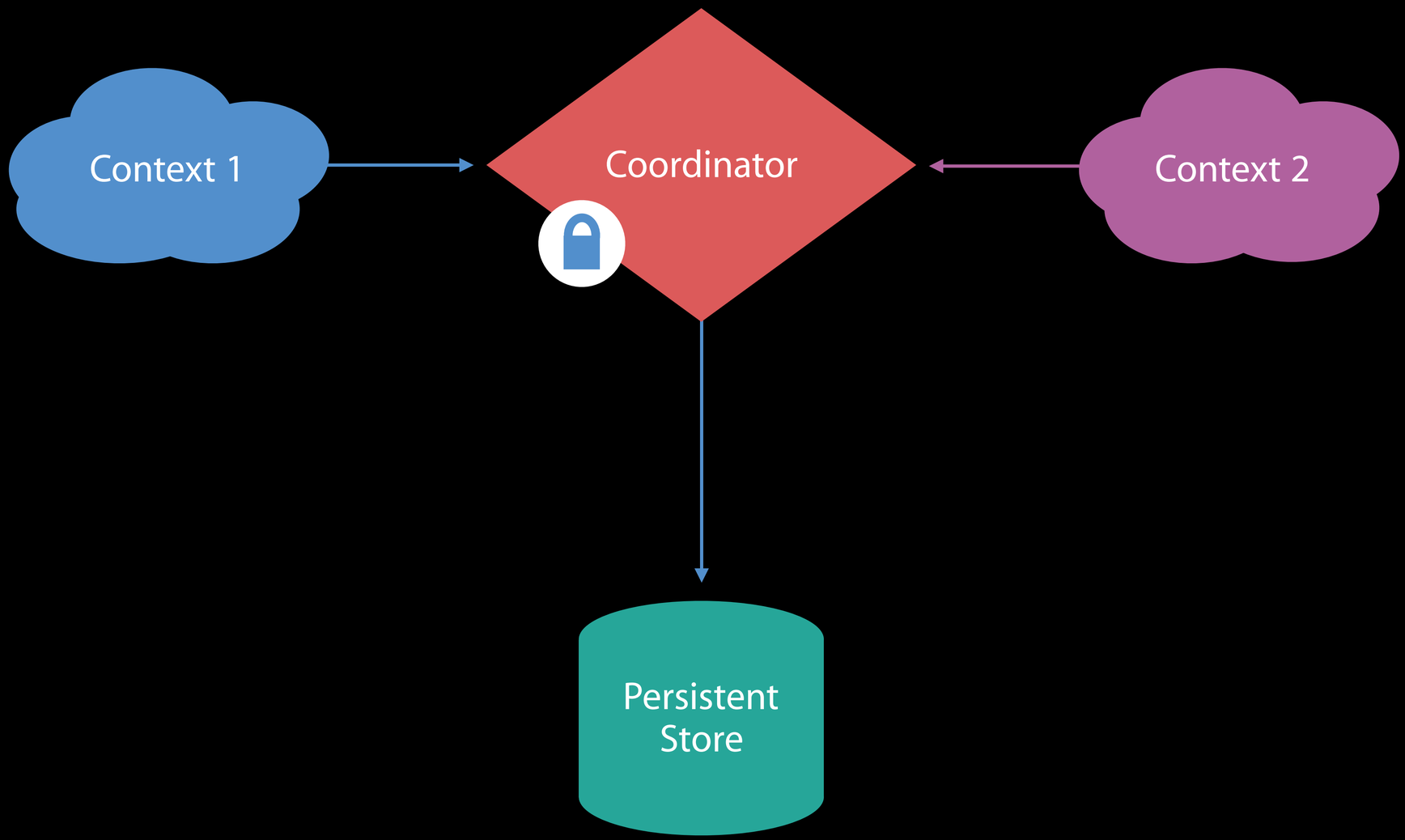
Current State of Affairs



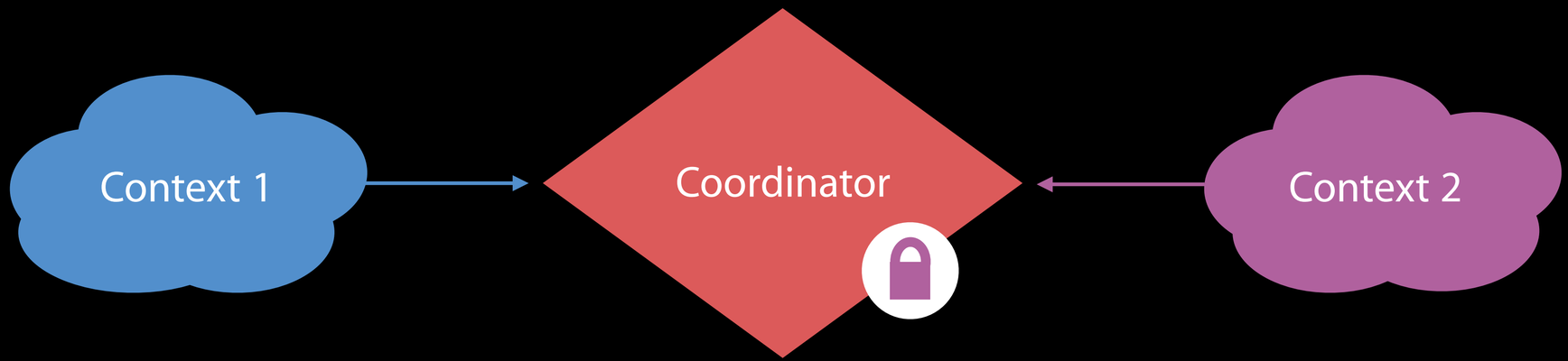
Current State of Affairs



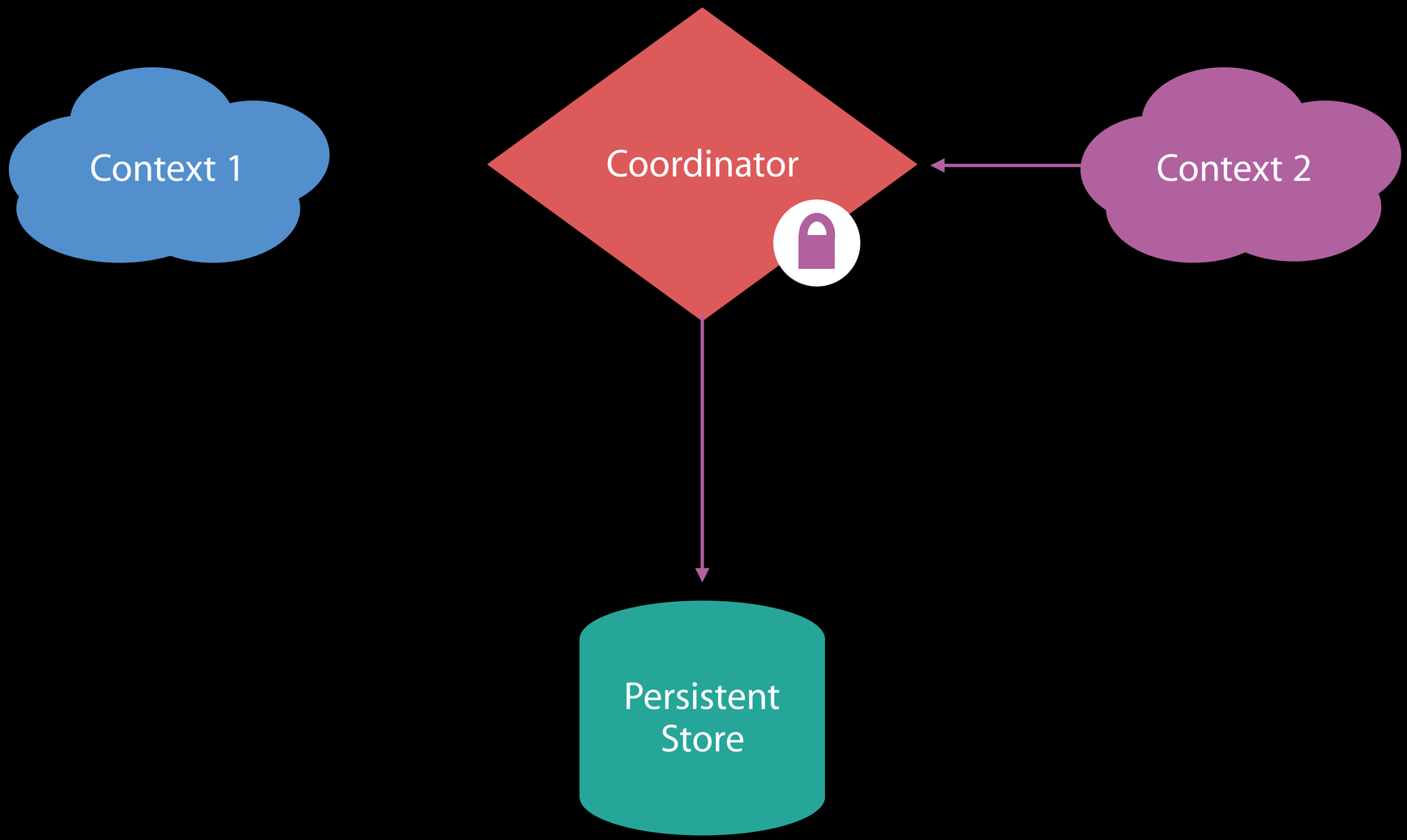
Current State of Affairs



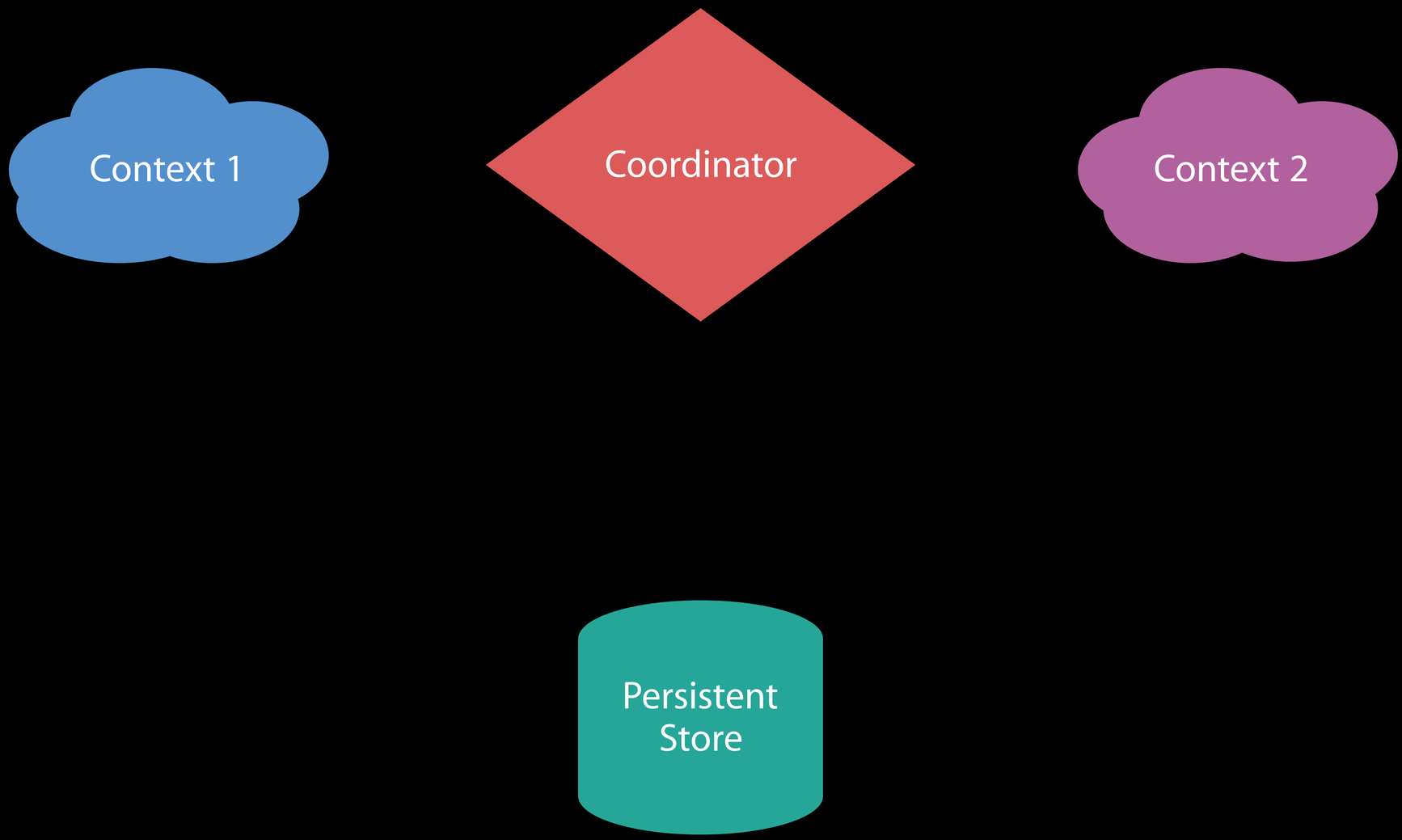
Current State of Affairs



Current State of Affairs



Current State of Affairs



New Stuff

I thought concurrency was hard, then I refactored

SQL store can now handle multiple concurrent requests

- Multiple readers
- Single writer

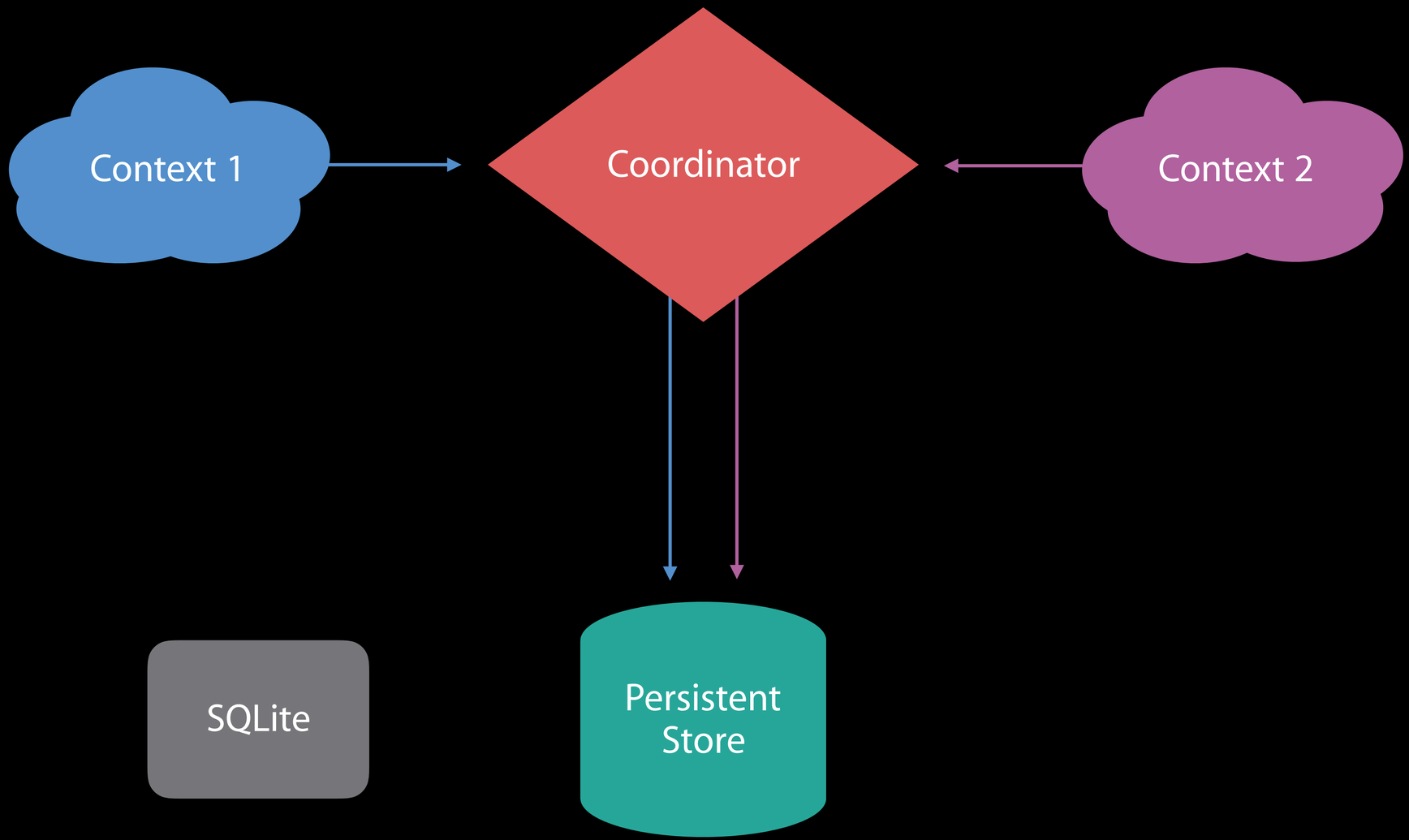
Connection pool

Moving the Lock Down

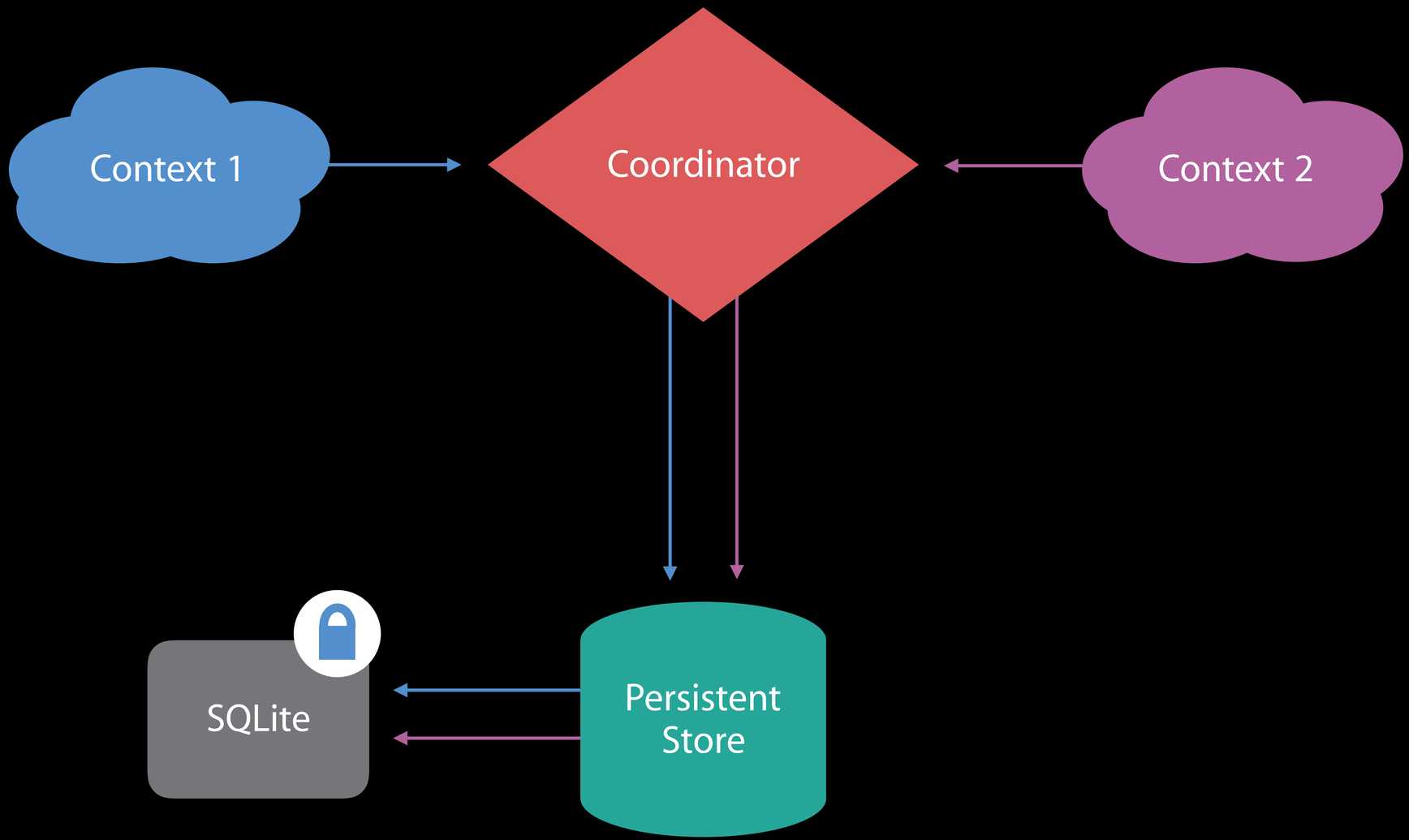
Moving the Lock Down

Moving the Lock Down

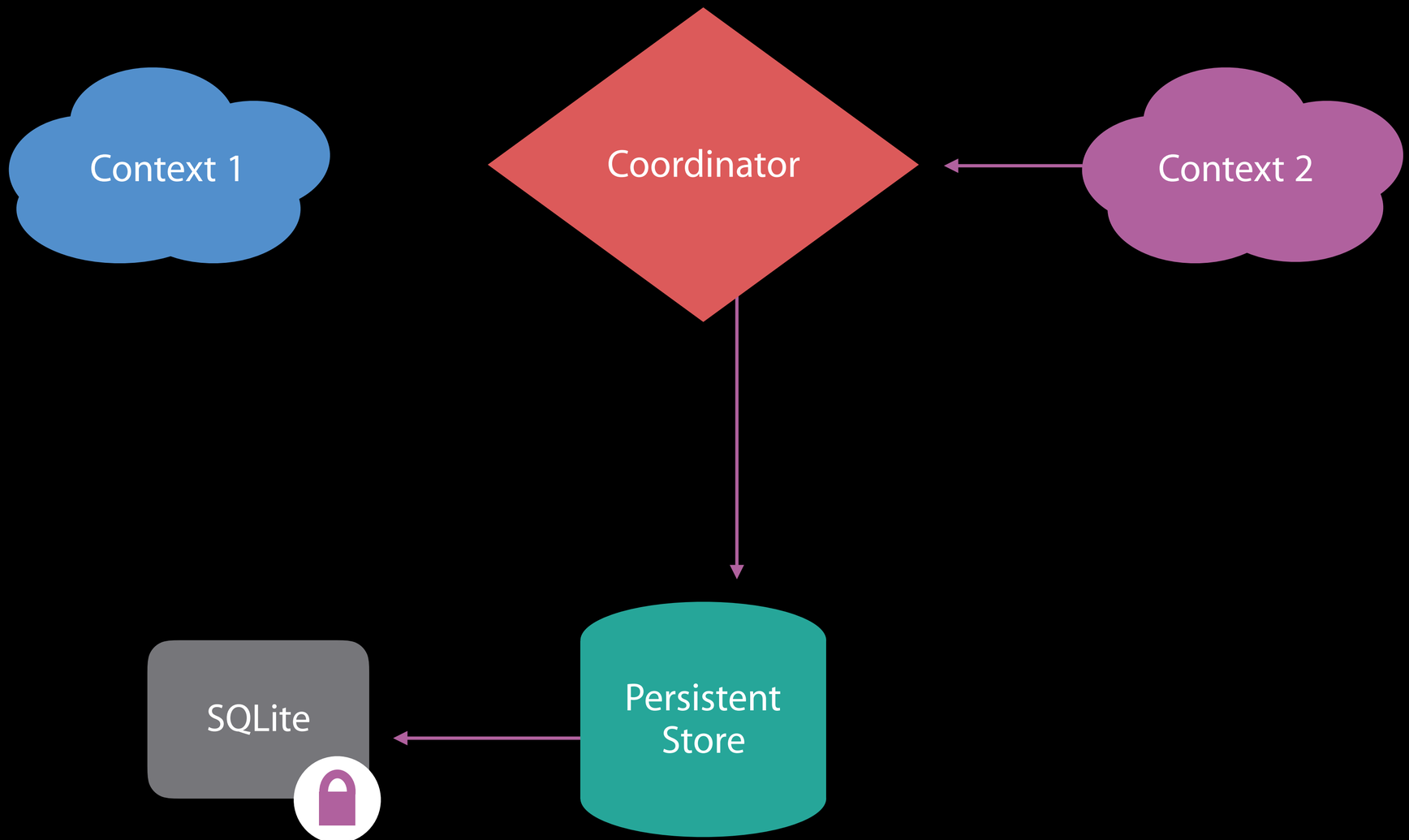
Moving the Lock Down



Moving the Lock Down



Moving the Lock Down



Moving the Lock Down

What does this mean?

More responsive UI

- Fault/fetch while background work occurs

Simpler application architecture

- Less need for multiple stacks
- Fewer complicated handoffs
- Bonus: shared row cache means lower memory footprint

Connection Pool

Nuts and bolts: API

On by default

- SQL stores only
- All coordinated stores must be SQL stores

New store option: `NSPersistentStoreConnectionPoolMaxSizeKey`

- Specify max pool size
- Set to 1 if you want serialized request handling

We reserve the right to adjust pool size

Considerations

How does it affect your code?

Should* be transparent

Better performance overall under contention

Code simplification

* Timing changed. If you depended on it, you may notice.

Next Up, Scott

Setting Up Core Data

Scott Perry Chordate

```
// Add a Persistent Store

persistentStoreCoordinator.addPersistentStore(ofType: type
                                  configurationName: configuration,
                                                 at: url
                                            options: options
```

type

configuration

url

options

NSPersistentStoreDescription

NEW

NSPersistentStoreDescription

NEW

Easy customization of common options

- Read-only
- Timeout
- Automatic lightweight migration
- Automatic mapping model inference

NSPersistentStoreDescription

NEW

Easy customization of common options

- Read-only
- Timeout
- Automatic lightweight migration
- Automatic mapping model inference
- Add store asynchronously

```
// Adding a Store to a Coordinator
```

```
persistentStoreCoordinator.addPersistentStore(with: storeDescription,
                                              completionHandler: { (storeDescription, error) in
    if let error = error {
        // Handle the error appropriately.
    }
})
```

```
// Adding a Store to a Coordinator, Asynchronously


storeDescription.shouldAddStoreAsynchronously = true

persistentStoreCoordinator.addPersistentStore(with: storeDescription,
                                              completionHandler: { (storeDescription, error) in
    if let error = error {
        // Handle the error appropriately.
    } else {
        // Handle success appropriately.
    }
})
```

Representing a Core Data Stack

NSManagedObjectModel

NSPersistentStoreCoordinator

NSManagedObjectContext

Representing a Core Data Stack

Representing a Core Data Stack

NEW

NSPersistentContainer

NSManagedObjectModel

NSPersistentStoreCoordinator

NSManagedObjectContext

Representing a Core Data Stack

NEW

Encapsulates model configuration

Name

Store descriptions

Loads stores

```swift
    lazy var applicationDocumentsDirectory: NSURL = {
        let urls = NSFileManager.defaultManager().urlsForDirectory(.documentDirectory,
                                                           inDomains: .userDomainMask)
        return urls[urls.count-1]
    }()

    lazy var managedObjectModel: NSManagedObjectModel = {
        let modelURL = NSBundle.main().urlForResource("AppName", withExtension: "momd")!
        return NSManagedObjectModel(contentsOf: modelURL)!
    }()

    lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator = {
        let coordinator = NSPersistentStoreCoordinator(managedObjectModel: self.managedObjectModel)
        let url = self.applicationDocumentsDirectory.appendingPathComponent("AppName.sqlite")
       do {
            try coordinator.addPersistentStore(ofType: NSSQLiteStoreType, configurationName: nil,
                                               at: url, options: nil)
        } catch {
            // Replace this with code to handle the error appropriately.
           fatalError("Unresolved error \(error), \(error.userInfo)")
        }
        return coordinator
    }()

    lazy var managedObjectContext: NSManagedObjectContext = {
        let coordinator = self.persistentStoreCoordinator
        var managedObjectContext = NSManagedObjectContext(concurrencyType: .mainQueueConcurrencyType)
        managedObjectContext.persistentStoreCoordinator = coordinator
        return managedObjectContext
    }()
```

```
    lazy var persistentContainer: NSPersistentContainer = {
        let container = NSPersistentContainer(name: "AppName")
        container.loadPersistentStores(completionHandler: { (storeDescription, error) in
            if let error = error {
                // Replace this implementation with code to handle the error appropriately.
                fatalError("Unresolved error \(error), \(error.userInfo)")
            }
        })
        return container
    }()
```

Where Did It All Go?

Default values and behavior

Where Did It All Go?

Default values and behavior

Model looked up by container name

Where Did It All Go?

Default values and behavior

Model looked up by container name

Store filename determined by container name

Where Did It All Go?

Default values and behavior

Model looked up by container name

Store filename determined by container name

Store directory provided by the container class

NSPersistentContainer

Support for common workflows

NSPersistentContainer

Support for common workflows

Main queue context property

NSPersistentContainer

Support for common workflows

Main queue context property

Private queue context factory method

NSPersistentContainer

Support for common workflows

Main queue context property

Private queue context factory method

Method for enqueuing background work

```
// Doing Background Work With a Container
```

```
let context = NSManagedObjectContext(concurrencyType: .privateQueueConcurrencyType)
context.persistentStoreCoordinator = persistentStoreCoordinator
context.perform({
    // …
})
```

```
// Doing Background Work With a Container
```

```
container.performBackgroundTask({ (context) in
    // …
})
```

NSManagedObjectContext

NEW

Speaking of common workflows…

NSManagedObjectContext

Speaking of common workflows…

NEW

New property `automaticallyMergesChangesFromParent`

NSManagedObjectContext

Speaking of common workflows…

NEW

New property `automaticallyMergesChangesFromParent`

Works with generation tokens

Generics

Generics

Better living through types

Generics

Better living through types

New protocol `NSFetchRequestResult`

Generics

Better living through types

New protocol `NSFetchRequestResult`

- `NSManagedObject` (and subclasses)

Generics

Better living through types

New protocol `NSFetchRequestResult`

- `NSManagedObject` (and subclasses)
- `NSManagedObjectID`

Generics

Better living through types

New protocol `NSFetchRequestResult`

- `NSManagedObject` (and subclasses)
- `NSManagedObjectID`
- `NSDictionary`

Generics

Better living through types

New protocol `NSFetchRequestResult`

- `NSManagedObject` (and subclasses)
- `NSManagedObjectID`
- `NSDictionary`
- `NSNumber`

Generics

Better living through types

New protocol `NSFetchRequestResult`

- `NSManagedObject` (and subclasses)
- `NSManagedObjectID`
- `NSDictionary`
- `NSNumber`

`NSFetchRequest<ResultType>`

Generics

Better living through types

New protocol `NSFetchRequestResult`

- `NSManagedObject` (and subclasses)
- `NSManagedObjectID`
- `NSDictionary`
- `NSNumber`

`NSFetchRequest<ResultType>`

`NSManagedObjectContext.fetch(NSFetchRequest<ResultType>) throws -> [ResultType]`

Generics

Better living through types

New protocol `NSFetchRequestResult`

- `NSManagedObject` (and subclasses)
- `NSManagedObjectID`
- `NSDictionary`
- `NSNumber`

`NSFetchRequest<ResultType>`

`NSManagedObjectContext.fetch(NSFetchRequest<ResultType>) throws -> [ResultType]`

`NSFetchedResultsController<ResultType>`

NSFetchedResultsController

❤️ UICollectionViewDataSourcePrefetching

NSFetchedResultsController

❤️ UICollectionViewDataSourcePrefetching

```
func collectionView(_ collectionView: UICollectionView,
                    prefetchItemsAt indexPaths: [IndexPath]) {
    let fetchRequest: NSFetchRequest<Event> = Event.fetchRequest()
    fetchRequest.returnObjectsAsFaults = false
    let objects = indexPaths.map({ (index) -> Event in
        self.fetchedResultsController.object(at: index)
    })
    fetchRequest.predicate = Predicate(format: "SELF IN %@", objects as CVarArg)
    let asyncFetchRequest = NSAsynchronousFetchRequest(fetchRequest: fetchRequest,
                                                       completionBlock: nil)
    do {
        try fetchedResultsController.managedObjectContext.execute(asyncFetchRequest)
    } catch {}
}
```

NSFetchedResultsController

❤️ UICollectionViewDataSourcePrefetching

```
func collectionView(_ collectionView: UICollectionView,
                    prefetchItemsAt indexPaths: [IndexPath]) {
    let fetchRequest: NSFetchRequest<Event> = Event.fetchRequest()
    fetchRequest.returnObjectsAsFaults = false
    let objects = indexPaths.map({ (index) -> Event in
        self.fetchedResultsController.object(at: index)
    })
    fetchRequest.predicate = Predicate(format: "SELF IN %@", objects as CVarArg)
    let asyncFetchRequest = NSAsynchronousFetchRequest(fetchRequest: fetchRequest,
                                                       completionBlock: nil)
    do {
        try fetchedResultsController.managedObjectContext.execute(asyncFetchRequest)
    } catch {}
}
```

What's New in UICollectionView | Presidio | Thursday 9:00AM

Now Available

NSFetchedResultsController on macOS

Common Operations

Type casts and strings

Common Operations

Type casts and strings

```
// Get an entity description

NSEntityDescription.entity(forEntityName: "Entity", in: managedObjectContext)
```

Common Operations

Type casts and strings

```
// Get an entity description

NSEntityDescription.entity(forEntityName: "Entity", in: managedObjectContext)
```

```
// Create a fetch request

NSFetchRequest(entityName: "Entity") as! NSFetchRequest<Subclass>
```

Common Operations

Type casts and strings

```
// Get an entity description

NSEntityDescription.entity(forEntityName: "Entity", in: managedObjectContext)


// Create a fetch request

NSFetchRequest(entityName: "Entity") as! NSFetchRequest<Subclass>


// Create a managed object

NSEntityDescription.insertNewObjectForEntityForName("Entity", inManagedObjectContext:
managedObjectContext) as! Subclass
```

Common Operations

Subclasses gain superpowers

NEW

```
// Get an entity description

NSEntityDescription.entity(forEntityName: "Entity", in: managedObjectContext)


// Create a fetch request

NSFetchRequest(entityName: "Entity") as! NSFetchRequest<Subclass>


// Create a managed object

NSEntityDescription.insertNewObjectForEntityForName("Entity", inManagedObjectContext:
managedObjectContext) as! Subclass
```

Common Operations

Subclasses gain superpowers

NEW

```
// Get an entity description

NSEntityDescription.entity(forEntityName: "Entity", in: managedObjectContext)

// Create a fetch request

NSFetchRequest(entityName: "Entity") as! NSFetchRequest<Subclass>

// Create a managed object

NSEntityDescription.insertNewObjectForEntityForName("Entity", inManagedObjectContext:
managedObjectContext) as! Subclass
```

Common Operations

Subclasses gain superpowers

NEW

```
// Get an entity description
        .entity()

// Create a fetch request
NSFetchRequest(entityName: "Entity") as! NSFetchRequest<Subclass>

// Create a managed object
NSEntityDescription.insertNewObjectForEntityForName("Entity", inManagedObjectContext:
managedObjectContext) as! Subclass
```

Common Operations

Subclasses gain superpowers

NEW

```
// Get an entity description

Subclass.entity()


// Create a fetch request

NSFetchRequest(entityName: "Entity") as! NSFetchRequest<Subclass>


// Create a managed object

NSEntityDescription.insertNewObjectForEntityForName("Entity", inManagedObjectContext:
managedObjectContext) as! Subclass
```

Common Operations

Subclasses gain superpowers

NEW

```
// Get an entity description

Subclass.entity()


// Create a fetch request

NSFetchRequest(entityName: "Entity") as! NSFetchRequest<Subclass>


// Create a managed object

NSEntityDescription.insertNewObjectForEntityForName("Entity", inManagedObjectContext:
managedObjectContext) as! Subclass
```

Common Operations

Subclasses gain superpowers

NEW

```
// Get an entity description

Subclass.entity()


// Create a fetch request

Subclass  etchRequest()


// Create a managed object

NSEntityDescription.insertNewObjectForEntityForName("Entity", inManagedObjectContext:
managedObjectContext) as! Subclass
```

Common Operations

Subclasses gain superpowers

NEW

```
// Get an entity description

Subclass.entity()


// Create a fetch request

Subclass.fetchRequest()


// Create a managed object

NSEntityDescription.insertNewObjectForEntityForName("Entity", inManagedObjectContext:
managedObjectContext) as! Subclass
```

Common Operations

Subclasses gain superpowers

NEW

```
// Get an entity description

Subclass.entity()


// Create a fetch request

Subclass.fetchRequest()


// Create a managed object

NSEntityDescription.insertNewObjectForEntityForName("Entity", inManagedObjectContext:
managedObjectContext) as! Subclass
```

Common Operations

Subclasses gain superpowers

NEW

```
// Get an entity description

Subclass.entity()


// Create a fetch request

Subclass.fetchRequest()


// Create a managed object

Subclass( ontext: managedObjectContext)
```

Common Operations

Subclasses gain superpowers

NEW

```
// Get an entity description

Subclass.entity()


// Create a fetch request

Subclass.fetchRequest()


// Create a managed object

Subclass(context: managedObjectContext)
```

Common Operations

Subclasses gain superpowers

NEW

```
// Get an entity description

Subclass.entity()


// Create a fetch request

Subclass.fetchRequest()


// Create a managed object

Subclass(context: managedObjectContext)
```

Common Operations

Subclasses gain superpowers

NEW

```
// Get an entity description

Subclass.entity()


// Create a fetch request

Subclass.fetchRequest()


// Create a managed object

Subclass(context: managedObjectContext)


// Execute a fetch request

try context.fetch(fetchRequest)
```

Common Operations

Subclasses gain superpowers

NEW

```
// Get an entity description

Subclass.entity()


// Create a fetch request

Subclass.fetchRequest()


// Create a managed object

Subclass(context: managedObjectContext)


// Execute a fetch request

try context.fetch(fetchRequest)
```

Common Operations

Subclasses gain superpowers

NEW

```
// Get an entity description

Subclass.entity()


// Create a fetch request

Subclass.fetchRequest()


// Create a managed object

Subclass(context: managedObjectContext)


// Execute a fetch request

try fetchRequest.       ()
```

Common Operations

Subclasses gain superpowers

NEW

```
// Get an entity description

Subclass.entity()


// Create a fetch request

Subclass.fetchRequest()


// Create a managed object

Subclass(context: managedObjectContext)


// Execute a fetch request

try fetchRequest.execute()
```

Automatic Subclass Generation

Always up to date

Automatic Subclass Generation

Always up to date

Configured per entity

Automatic Subclass Generation

Always up to date

Configured per entity

DerivedData

Automatic Subclass Generation

Always up to date

Configured per entity

DerivedData

Automatically regenerated

Automatic Subclass Generation

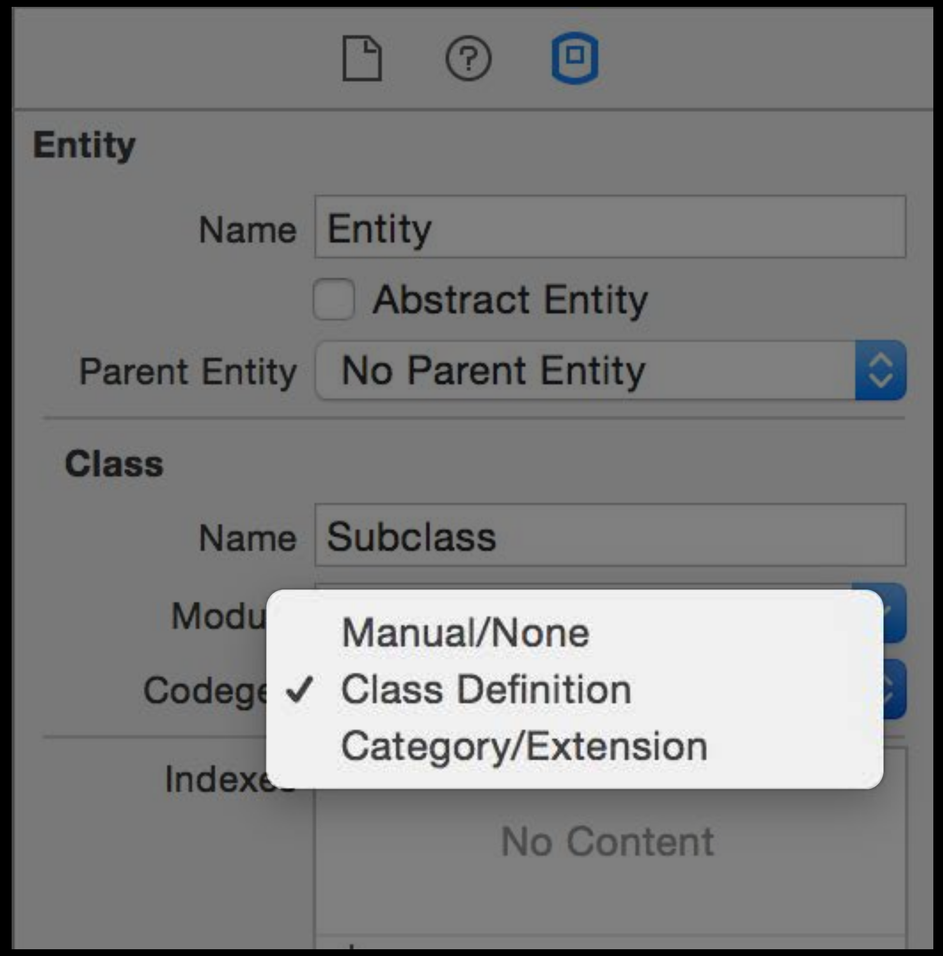
Always up to date

Configured per entity

DerivedData

Automatically regenerated

Class or extension

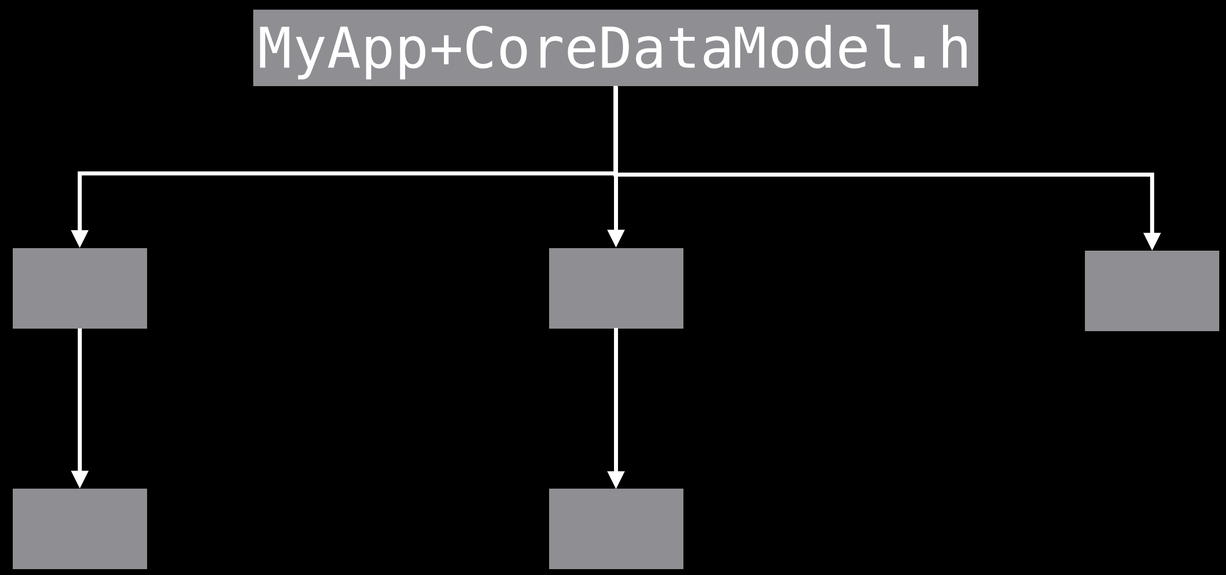


Automatic Subclass Generation

Always up to date

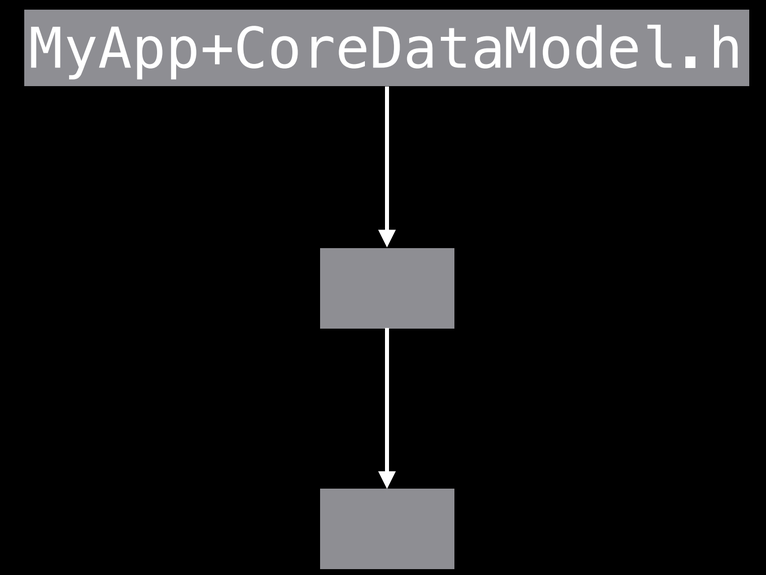
Automatic Subclass Generation

Always up to date



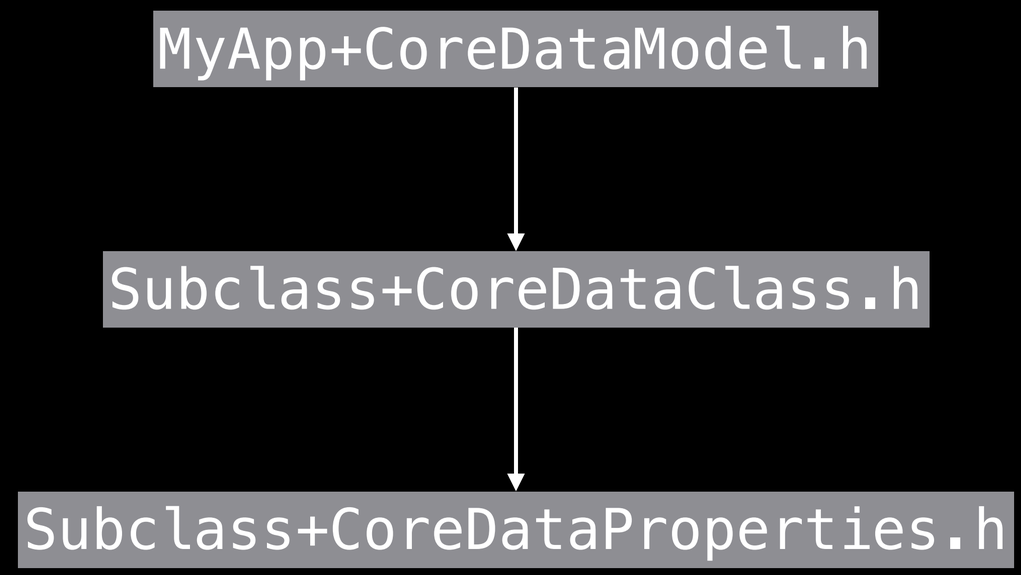
Automatic Subclass Generation

Always up to date

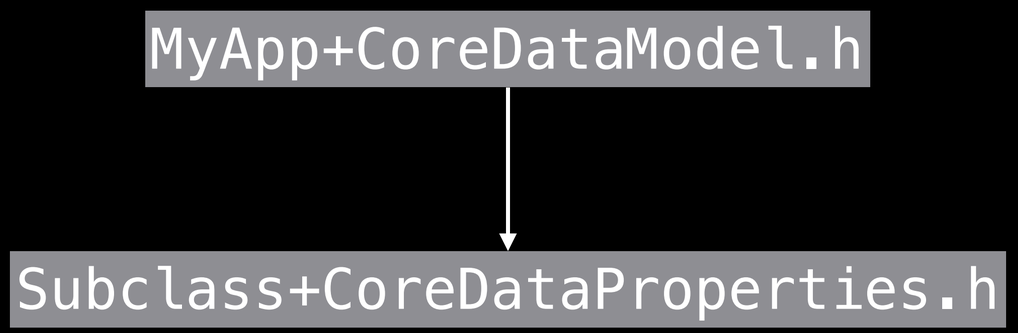


Automatic Subclass Generation

Class generation

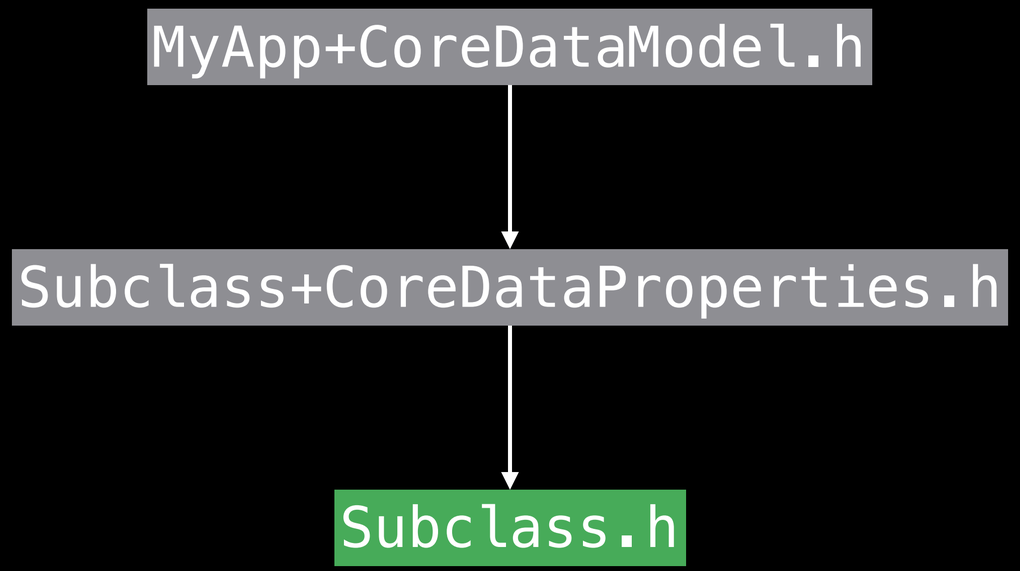


Automatic Subclass Generation



Automatic Subclass Generation

Category/Extension generation



Demo

Walking tour

What's New in SQLite

Multithreading Assertions

NEW

Multithreading Assertions

NEW

Database connections are not thread-safe

Multithreading Assertions

NEW

Database connections are not thread-safe

Multithreading bugs manifest as crashes deep inside SQLite

Multithreading Assertions

NEW

Database connections are not thread-safe

Multithreading bugs manifest as crashes deep inside SQLite

```
SQLITE_ENABLE_THREAD_ASSERTIONS=1
```

Built-in Logging

Built-in Logging

SQLite allows user-defined logging functions via `SQLITE_CONFIG_LOG`

Built-in Logging

SQLite allows user-defined logging functions via `SQLITE_CONFIG_LOG`

`sqlite3_config` must be called before library initialization

Built-in Logging

NEW

SQLite allows user-defined logging functions via `SQLITE_CONFIG_LOG`

`sqlite3_config` must be called before library initialization

```
SQLITE_ENABLE_LOGGING=1
```

File Operations and SQLite

Or: How to corrupt your database

File Operations and SQLite

Or: How to corrupt your database

Databases are represented by multiple files

File Operations and SQLite

Or: How to corrupt your database

Databases are represented by multiple files

File operations cannot be atomic across multiple files

File Operations and SQLite

Or: How to corrupt your database

Databases are represented by multiple files

File operations cannot be atomic across multiple files

All file operations are unsafe

What's New in SQLite

The problem with file operations

`mydata.db`

`mydata.db-journal`

What's New in SQLite

The problem with file operations

Corrupt Data

What's New in SQLite

The problem with file operations

`mydata.db`

`mydata.db-wal`

`mydata.db-shm`

What's New in SQLite

The problem with file operations

`sqlite3`

`mydata.db`

`mydata.db-wal`

`mydata.db-shm`

What's New in SQLite

The problem with file operations

`sqlite3`

`atadym.db`

`mydata.db-wal`

`mydata.db-shm`

What's New in SQLite

The problem with file operations

`sqlite3`

`mydata.db-wal`

`mydata.db-shm`

`sqlite3`

`atadym.db`

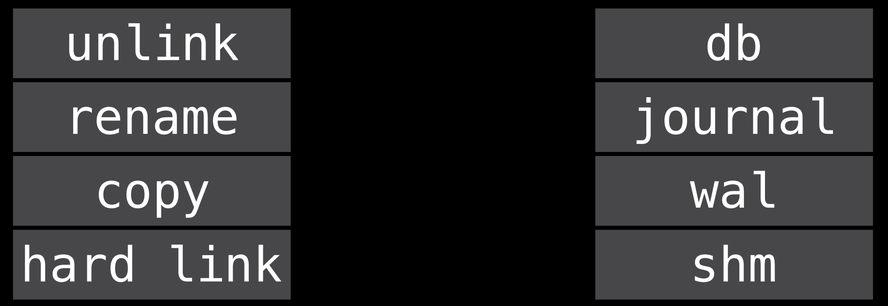
`atadym.db-wal`

`atadym.db-shm`

Corrupt Data

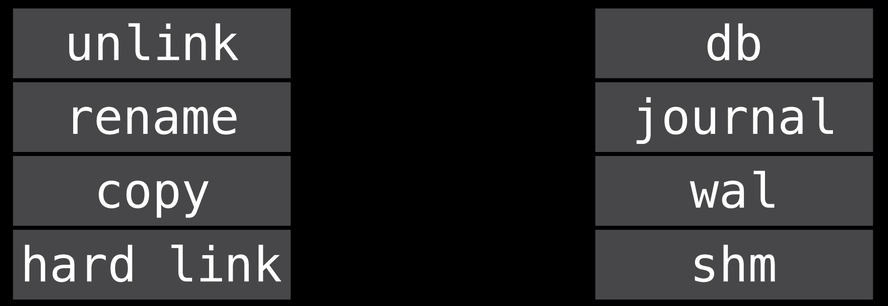
What's New in SQLite

The problem with file operations



What's New in SQLite

The problem with file operations



Corrupt Data

What's New in SQLite

Database file tracking

NEW

What's New in SQLite

Database file tracking

NEW

SQLite now uses dispatch sources to track illegal file operations

What's New in SQLite

Database file tracking

NEW

SQLite now uses dispatch sources to track illegal file operations

API calls after illegal operations will return `SQLITE_IOERR_VNODE`

What's New in SQLite

Database file tracking

NEW

SQLite now uses dispatch sources to track illegal file operations

API calls after illegal operations will return `SQLITE_IOERR_VNODE`

```
SQLITE_ENABLE_FILE_ASSERTIONS=1
```

What's New in SQLite

Database file tracking

NEW

SQLite now uses dispatch sources to track illegal file operations

API calls after illegal operations will return `SQLITE_IOERR_VNODE`

```
SQLITE_ENABLE_FILE_ASSERTIONS=1
```

---

https://www.sqlite.org/howtocorrupt.html

What Is Safe?

How to avoid database corruption

What Is Safe?

How to avoid database corruption

Clear database ownership

What Is Safe?

How to avoid database corruption

Clear database ownership

Guarantee exclusive file access

What Is Safe?

How to avoid database corruption

Clear database ownership

Guarantee exclusive file access

Use Core Data!

What Is Safe?

How to avoid database corruption

Clear database ownership

Guarantee exclusive file access

Use Core Data!

- `replacePersistentStore()`

What Is Safe?

How to avoid database corruption

Clear database ownership

Guarantee exclusive file access

Use Core Data!

- `replacePersistentStore()`
- `destroyPersistentStore()`

Summary

Query generations

Concurrency

Core Data stack configuration

New API

Swift

Xcode integration

SQLite

More Information

https://developer.apple.com/wwdc16/242

Related Sessions

What's New in Swift	Presidio	Tuesday 9:00AM
What's New in Cocoa	Nob Hill	Tuesday 11:00AM

Labs

Core Data Lab	Frameworks Lab D	Friday 11:00AM

WWDC16