Mastering Core Data

Miguel Sanchez and Adam Swift
Core Data Engineering

Introduction

- Core Data helps applications on all our platforms manage their data

- This session will help you become more proficient with Core Data

What You'll Learn

- Modeling tips and tricks
- Managed Object Lifecycle
- Multithreading
- Fetching
- Data migration

Modeling Tips and Tricks

Managed Object Model

Your key contract with us

- Let us help you
- Model building blocks
  - Entities
  - Attributes
  - Relationships
- Design model around your access patterns

Going Beyond NSManagedObject Instances

Yes, you can subclass

- Move away from KVC idioms and use true accessors

```
[myObj setName:@”Miguel”];
```

instead of

```
[myObj setValue:@”Miguel” forKey:@”name”];
```

- Improve code readability
- Faster execution

Accessor in Subclasses

We do most of the work for you

- Use @property to declare

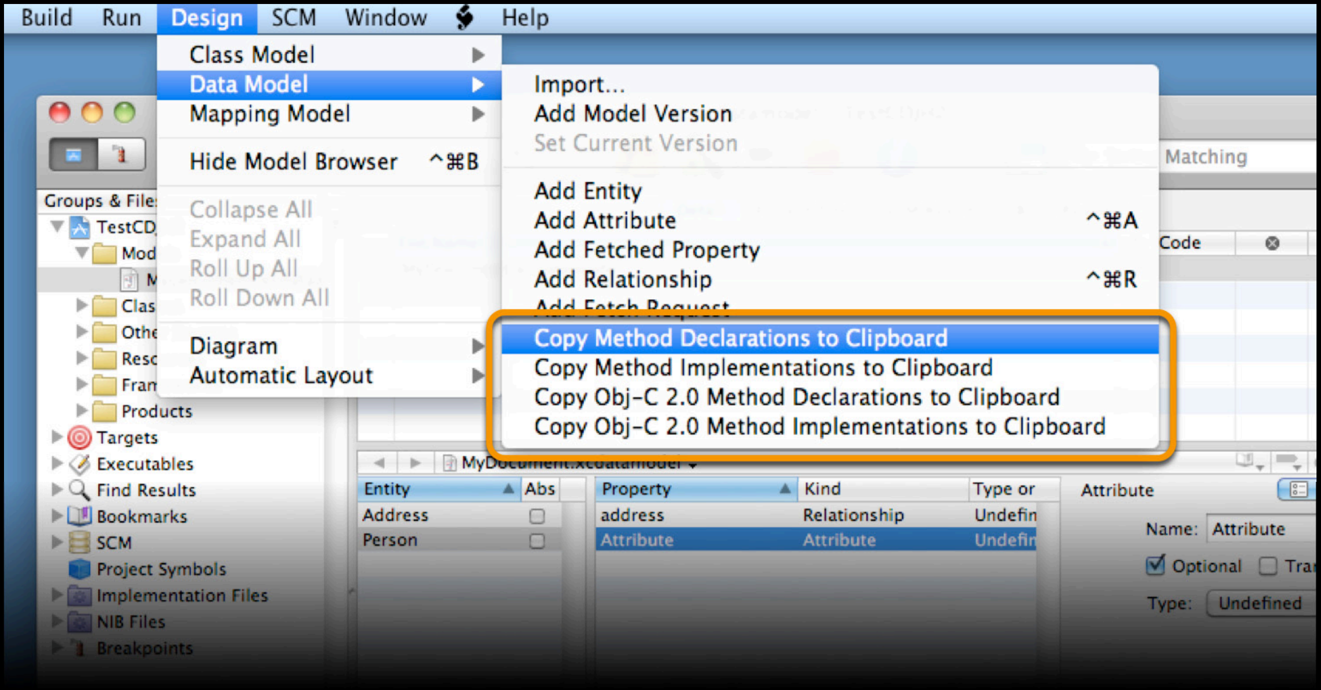
```
@property(nonatomic, retain) NSString* firstName;
@property(nonatomic, retain) Employee* manager;
@property(nonatomic, retain) NSSet* directReports;
```

- Use @dynamic and we'll dynamically resolve accessors for you

NSManagedObject Subclasses

Property code generation

- You can generate code for individual properties



NSManagedObject Subclasses

Tips and tricks

- Avoid method names from NSManagedObject and NSObject
  - description
  - deleted
- This includes all KVC resolutions
  - deleted
  - isDeleted
  - getDeleted
  - setDeleted:

Transient Properties

Modeled, but not persisted

- Require accessor to compute
- Gain benefits of a modeled property
  - Change tracking
- Flexibility from store schema
  - Adding transients doesn’t require migration

Transient Attribute Example

Basic computation and caching

- fullName = firstName + lastName

```
- (NSString *)fullName {
    [self willAccessValueForKey:@”fullName”];
    NSString *fullName = [self primitiveFullName];
    [self didAccessValueForKey:@”fullName”];

    if (fullName == nil) {
        fullName = [NSString stringWithFormat:@”%@ %@”,
                                self.firstName, self.lastName];
        [self setPrimitiveFullName:fullName];
    }

    return fullName;
}
```

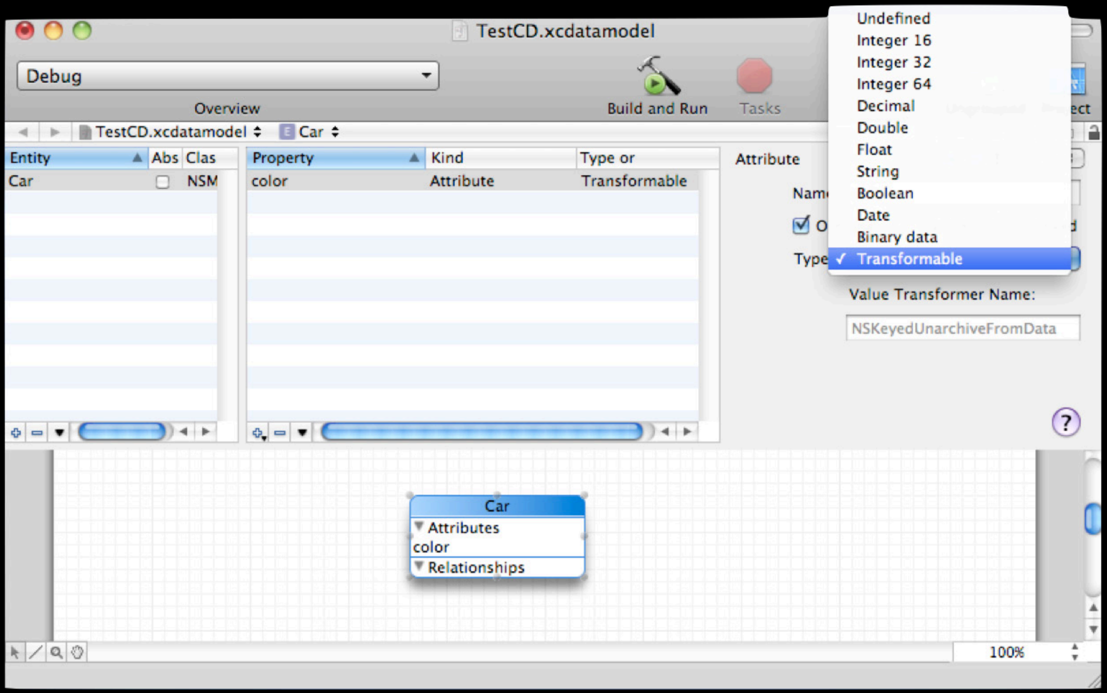
Transient Attribute Example #2

Reference to external resources

```
- (MyDocument *)documentObject {
 // NOT FULL METHOD

  MyDocument *result;
  ...
  NSString *documentPath = self.persistedDocumentPath;

  result = [MyDocument documentFromPath: documentPath];
  ...

  return result;
}
```

Transformable Attributes

Storing custom types



Entity	Abs	Clas
Car	☐	NSM

Property	Kind	Type or
color	Attribute	Transformable

Attribute

Type: Transformable

Value Transformer Name:

NSKeyedUnarchiveFromData

Using Transformable Attributes

Your types stored as NSData instances

- Default transformer is NSKeyedUnarchiveFromData
- Declare property to eliminate compiler warnings
  - .h
    - `@property (nonatomic, retain) NSColor *color;`
  - .m
    - `@dynamic color;`

Subclassing NSValueTransformer

Adapting to your needs

- Encrypt property data
- Improve performance vs. keyed archiving
- Don’t forget to account for endianness

Subclassing NSValueTransformer

Direction of transformation

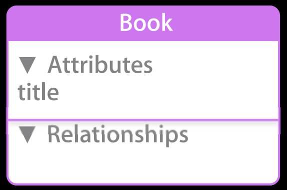
PhotoLocations example
Apple Developer website

```
- (id)transformedValue:(id)value {

   // Your Type --> NSData

    // return NSData instance
}

- (id)reverseTransformedValue:(id)value {

   // NSData --> Your Type

    // return Your Type instance
}
```

Adapting to Access Patterns

Example: Searching on book title

- Book entity
- Match on title



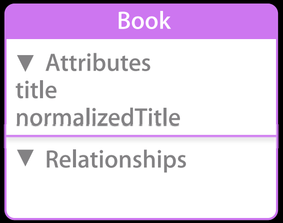
```
title contains[dc] $searchValue
```

Adapting to Access Patterns

Precomputing normalized title

DerivedProperty example
Apple Developer website

- Remove diacritical marks
- Change predicate to type-along prefix matching

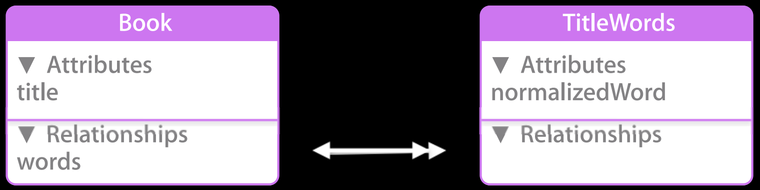


```
normalizedTitle >= $prefix and normalizedTitle < $nextPrefix

normalizedTitle >= ‘star’ and normalizedTitle < ‘stas’
```

Adapting to Access Patterns

Prefix matching on any word in title

- Put title words in a relationship
- Search on TitleWords and traverse relationship back to Book



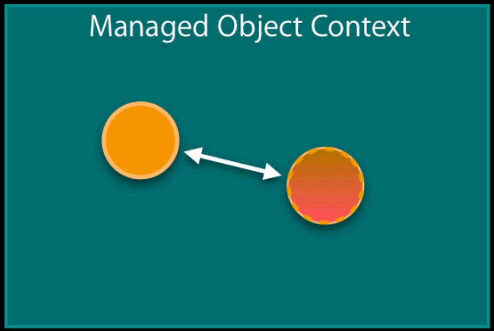
```
normalizedWord >= $prefix and normalizedWord < $nextPrefix
```

Managed Object Life Cycle

The Life of Managed Objects

From birth to death, and in between

- Creation
  - Insertion and fetching
- Active use
  - Updating, saving, undoing
- Cleanup
  - Deletion, reverting to faults



Hooking into Managed Object Life Events

Method overrides vs. listening to notifications

- Overriding methods in NSManagedObject
    - Per instance actions
- Processing Managed Object Context notifications
    - Graph change actions
- Reacting to errors from Managed Object Context actions

Awake Methods

Good place for initialization logic

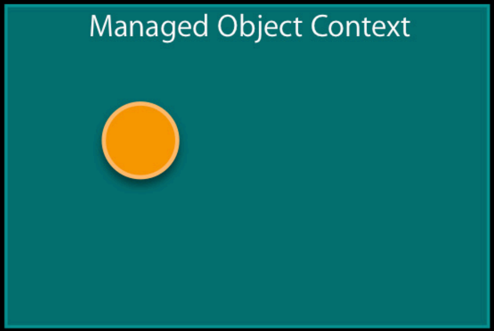
```
-(void)awakeFromInsert

-(void)awakeFromFetch

-(void)awakeFromSnapshotEvents:(NSSnapshotEventType)flag
```

Awaking from Insertion

- (void)awakeFromInsert

- More complicated initialization
- Set baseline values
- Avoid setting up relationships here

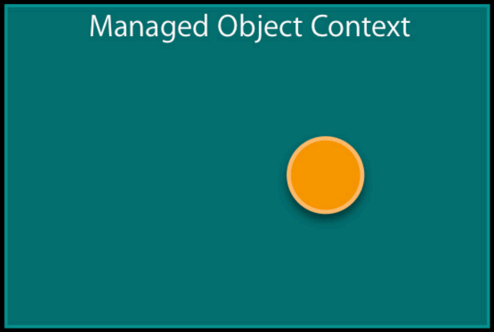
```
- (void)awakeFromInsert {
    [super awakeFromInsert];
    self.employeeID = // next employee id;
}
```



Awaking from Fetching

- (void)awakeFromFetch

- Compute derived state
  - Transient properties
  - Unmodeled state
- Avoid setting up relationships here



```
- (void)awakeFromFetch{

    [super awakeFromFetch];

    self.fullName = [NSString stringWithFormat:@”%@ %@”,
                                self.firstName, self.lastName];

}
```

Awaking from Undo and Refresh

- (void)awakeFromSnapshotEvents:(NSSnapshotEventType)flag

- Event can be
  - Undo from insert, delete update
  - Refresh
- Clear out cached values

```
- (void)awakeFromSnapshotEvents:(NSSnapshotEventType)flag {
    [super awakeFromSnapshotEvents:flag];
    self.fullName = nil;
}
```

Processing Changed Objects

Notification from the Managed Object Context

- NSManagedObjectContextObjectsDidChangeNotification
  - Inserts
  - Updates
  - Deletes
- Communicates what will happen on next save

Processing Changed Objects

When is notification sent?

- MOC processPendingChanges
- MOC save
- End of event loop
- Before fetching

Context Notifications on Save

Gives you all objects involved

- NSManagedObjectContextWillSaveNotification
  - Setting timestamps
- NSManagedObjectContextDidSaveNotification
  - Telling others of the save

Communicating Changes to Other Contexts

Merging changes from a save

- Inserts, updates, and deletes are applied to destination MOC

```
- (void)mergeChangesFromContextDidSaveNotification:(NSNotification *)ncn
```

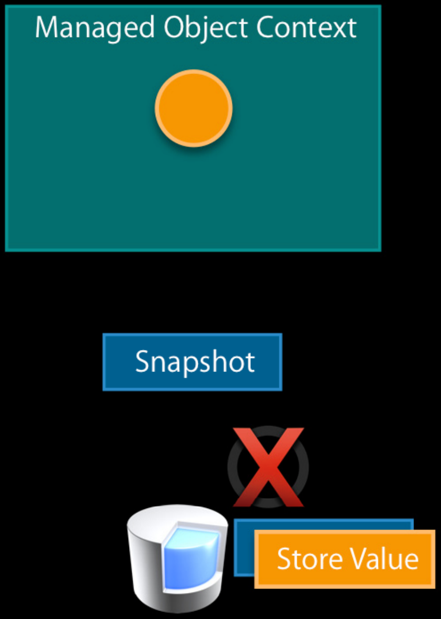
Saving the Managed Object Context

- (BOOL)save:(NSError **)error

- Check for validation errors
  - See CoreDataErrors.h for codes
  - NSDetailedErrorsKey in userInfo chains multiple errors
- Optimistic locking failures

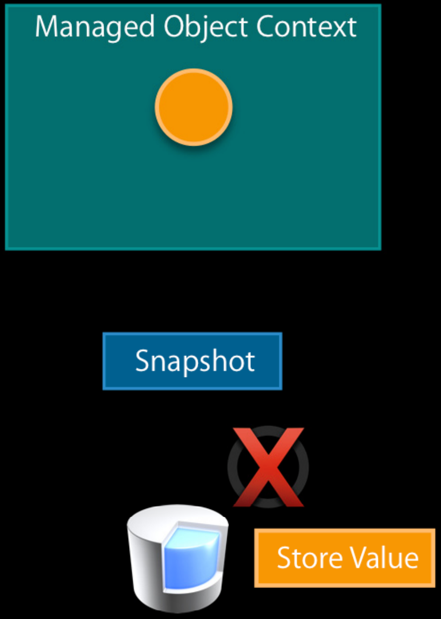
Optimistic Locking

Multi-writer conflict detection



Optimistic Locking

Set merge policy on managed object context



```
NSErrorMergePolicy

NSMergeByPropertyStoreTrumpMergePolicy

NSMergeByPropertyObjectTrumpMergePolicy

NSOverwriteMergePolicy

NSRollbackMergePolicy
```

Cleaning Up

Deletion

- Remember that deletion doesn’t actually happen until MOC saves
- Can’t access relationships in deleted objects in MOC save notification
- Hook to cache external resources to delete

```
– (void)prepareForDeletion
```

Cleaning Up

Becoming a fault again

- Don't override dealloc
  - `- (void)willTurnIntoFault`

- Clearing out custom caches
- Clear out KVO dependencies

Turning Objects Back to Faults

Trimming the Object Graph

- Turning single objects back to faults
  - `(void)refreshObject:(NSManagedObject *) mergeChanges:(BOOL)flag`
    - Do not call dirty objects with mergeChanges:NO

- Resetting all objects in a MOC
  - `(void)reset`

Multithreading

Considering Multithreaded Core Data

Asynchronous execution

- Improve UI responsiveness
- Background fetching
- Improved batched saving

Re-Considering Multithreaded Core Data

Potential issues

- Thread switching isn’t free
- Resource contention
- Increased complexity

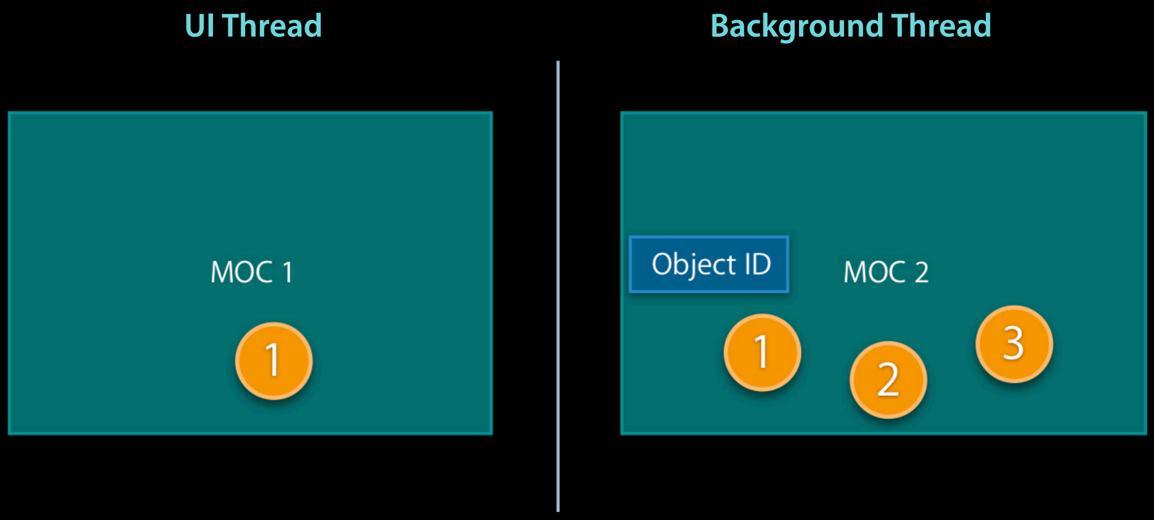
Thread Confinement

Core Data's Golden Rule

- Each "thread" gets own Managed Object Context
  - GCD: Each concurrent block gets own MOC
- Don't pass managed objects between threads
  - Pass objectIDs to get local copies

Passing Objects Between Threads

Object IDs are thread safe



```
– (NSManagedObject *)objectWithID:(NSManagedObjectID *)moid
```

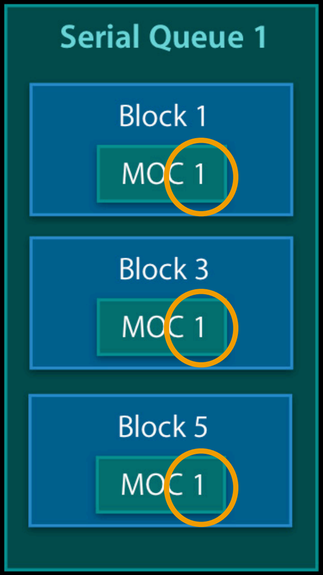
Communicating Unsaved Objects

Must save first

- Unsaved objects have temporary IDs
- Saving makes ID permanent
- Only pass saved object IDs to other contexts

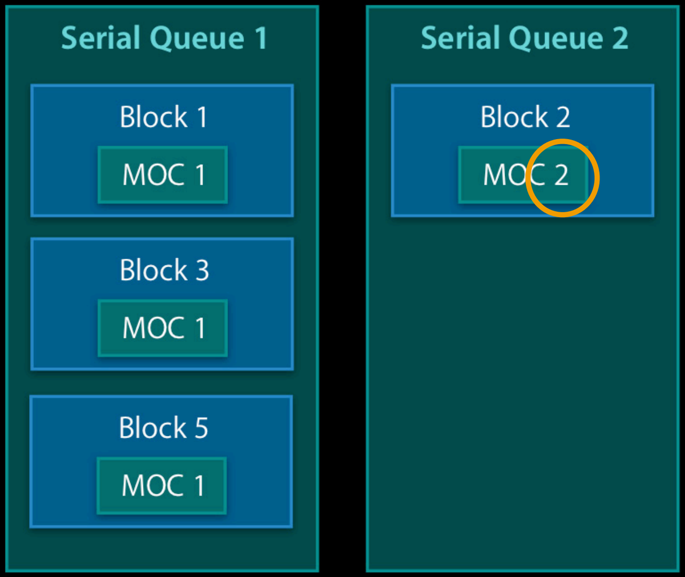
Queue Setup with GCD

Each potentially concurrent unit gets own MOC



Queue Setup with GCD

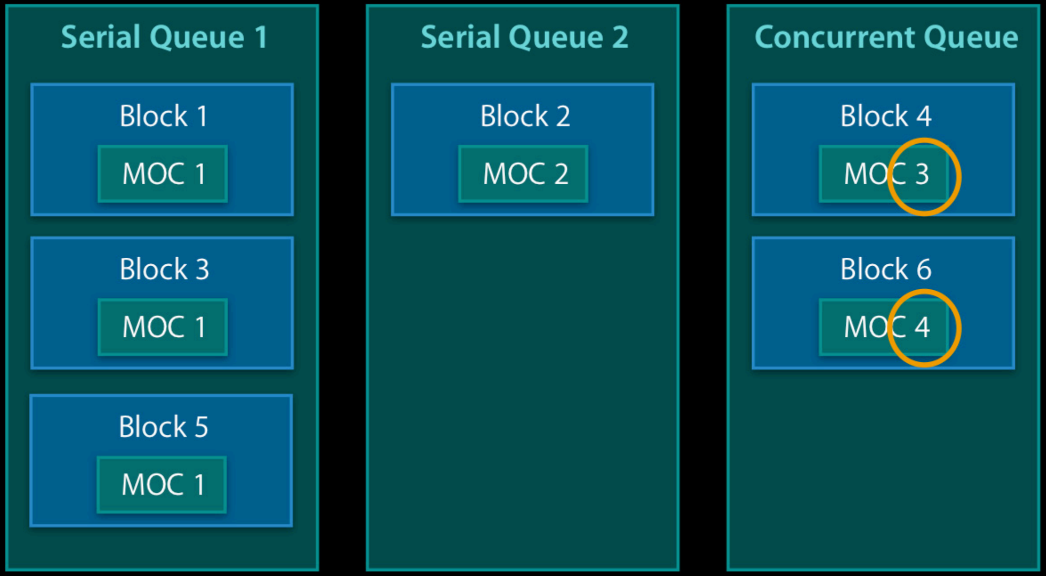
Each potentially concurrent unit gets own MOC



Blocks in serial queues can execute concurrently with blocks in other queues

Queue Setup with GCD

Each potentially concurrent unit gets own MOC



Multi-Party Edits and Deletes

Carefully define the workflow of your app

- Refresh single object

```
– (void)refreshObject:(NSManagedObject *)mo mergeChanges:(BOOL)flag
```

- Merge saved changes

```
– (void)mergeChangesFromContextDidSaveNotification:(NSNotification *)nfc
```

Fetching

Adam Swift
Core Data Engineering

Focus on Performance

User experience

- Be responsive
- Scale gracefully

Fetching Strategies for Performance

Memory and I/O

- Only fetch what you need
- UI defines your “working set”
- Amortize database I/O

3G
9:42 AM
All Contacts
Search
A
Katie Abeles
Aaron Ackermann
Kevin Angel
John Appleseed
B
Jessica Barbieri
Dave Becker
Favorites
Recents
Contacts
Keypad
Voicemail

Fetching Is I/O

Balance size vs. frequency

- Fetch objects and data in batches
- Avoid hangs and thrashing

Leverage Database Power

Keep the working set small

- Let the database do the heavy lifting
- Use predicate to filter results
- Sort descriptors to order them

Using Predicates

Working in the database

- Faster than fetching
- Use less memory

Predicates Can Replace Faulting

Evaluated using SQL

- Use @count to avoid fetching a to-many relationship
- Example
    - Music playlists without any songs

```
songs.@count == 0
```

Predicates and To-Many Relationships

Use a SUBQUERY to access related data

- Test attributes related through a to-many
- Example
  - Artists with songs longer than 10 minutes

```
SUBQUERY(songs, $s, $s.length > 600).@count > 0
```

Unique Attribute Values

Read-only results

- Fetch ONLY the distinct values
- Returns read-only dictionaries
- Example
    - All unique album names

```
[request setReturnsDistinctResults: YES];
[request setResultType: NSDictionaryResultType];
[request setPropertiesToFetch: [NSArray arrayWithObject: @”name”]];
[request setEntity: albumEntity];
```

Fetch Calculated, Aggregate Data

Evaluated in the database

- Fetch calculated results as dictionaries
- Example
  - totalTime = sum length of all songs

```
ex = [NSExpression expressionForFunction: @"sum:" arguments: [NSArray
  arrayWithObject: [NSExpression expressionForKeypath: @"length"]]];
sumED = [[NSExpressionDescription alloc] init];
[sumED setExpression: ex];
[sumED setExpressionResultType: NSDoubleAttributeType];
[sumED setName: @"totalTime"];
[request setEntity: songEntity];
[request setPropertiesToFetch: [NSArray arrayWithObject: sumED]];
[request setResultType: NSDictionaryResultType];
```

How Many?

Just fetch the count

- Use `countForFetchRequest:`
- Bonus points
  - Sort and use a fetch limit to fetch first few

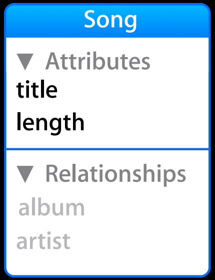
```
[request setSortDescriptors: playlistOrderKeys];
[request setFetchLimit: 3];
```

Playlist	# of Songs	Titles
Driving	911	Song A1, Song A2, Song A3, …
Exercise	5,000	Song B1, Song B2, Song B3, …
Chill Out	1,056	Song C1, Song C2, Song C3, …

Fetching Managed Objects

Use the attribute data now?

- 'Faulted' managed object
- All attributes
- Relationships are faults

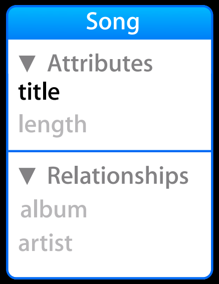


```
[request setReturnsObjectsAsFaults: NO];
```

Fetching Faults

A managed object placeholder

- Attributes fetched on demand
- Partial faults can prefetch subset of attributes

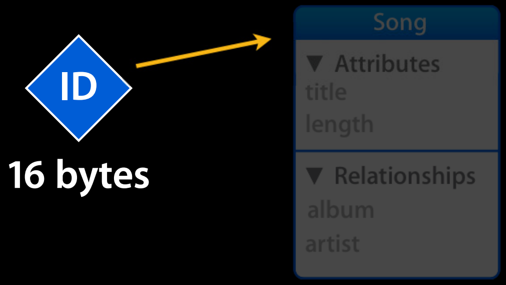


```
[request setPropertiesToFetch:[NSArray arrayWithObject: @”title”]];
```

Managed Object ID

Like a URL to Managed Objects

- Small and threadsafe
- Perfect for predicates

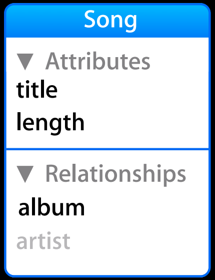


```
[request setResultType: NSManagedObjectIDResultType];
[request setIncludesPropertyValues: NO];
```

Relationship Faulting

Need the related data now

- Master table shows related data
- Prefetch to avoid faulting individually
- Example
  - List playlist songs and album name



```
[request setRelationshipKeypathsForPrefetching:
  [NSArray arrayWithObject: @”album”]];
```

Batching I/O

What if you can't control access?

- Some API wants full array
- Set the batch size
- Array subclass automatically batches

```
[request setFetchBatchSize: 100];
```

Performance Analysis

Focus your efforts

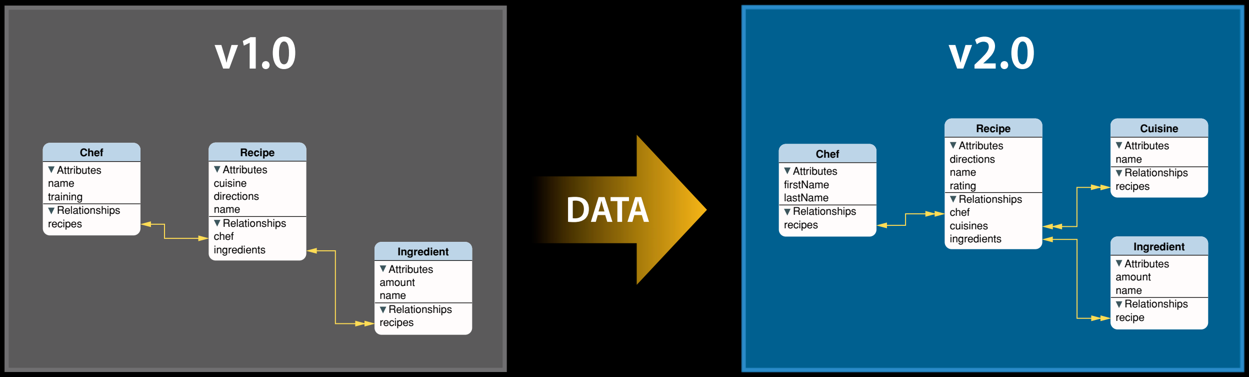
- Use Instruments
- Track faulting and fetching hot spots

- Take a look
  - NSFetchRequest.h
  - NSExpression.h
  - Predicate Programming Guide

Migration

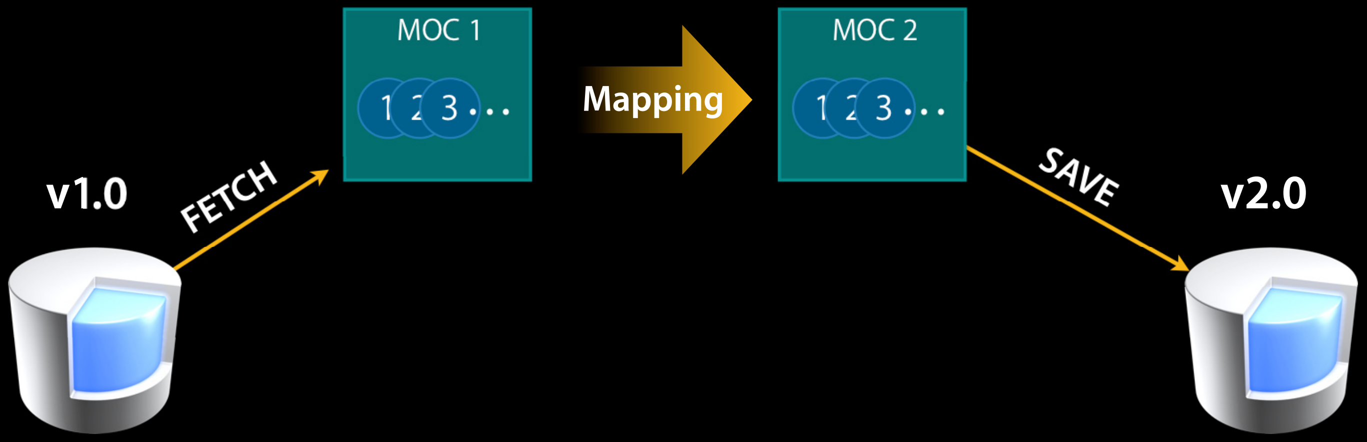
Why Is Migration Needed?

- Data model describes structure
- Changing model requires new structure
- Adapt old data to new structure



Custom Mapping
Hand-tuned, flexible logic

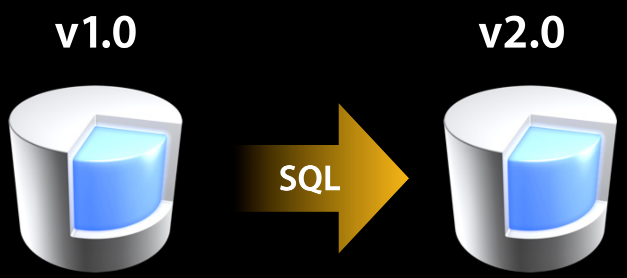
- Total control over changes
- Migrate objects in-memory



Lightweight Migration

Changes automatically inferred

- Automatic—for basic changes
- Migrate in-place via SQL



Inferable Changes

Supported by lightweight migration

- Add, remove, rename
- Attributes—numerical type conversion
- Relationships—promote to-one to to-many
- Entities—change entity inheritance

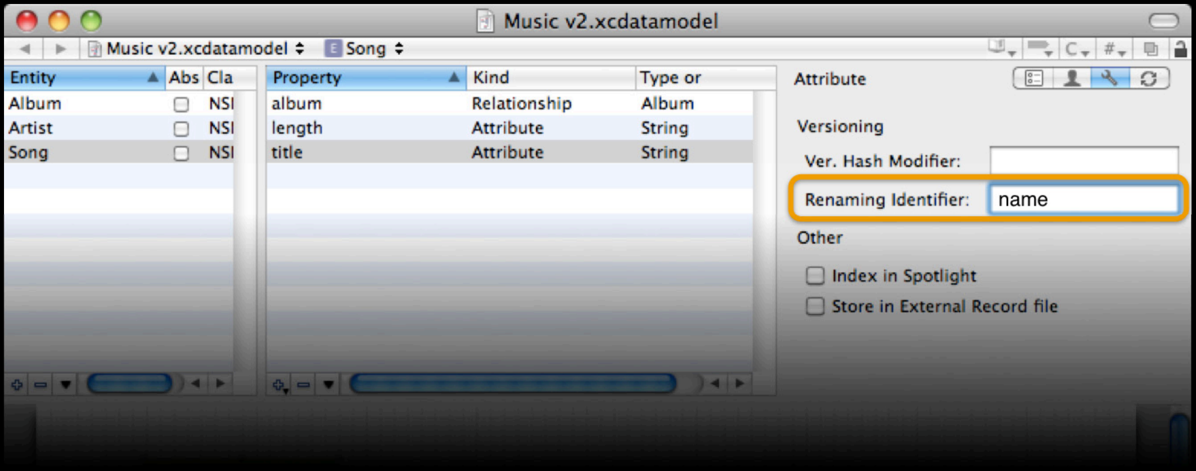
What You Have to Do

- You must keep the old model
  - Need to read old data
  - Xcode: Design › Data Model › Add model version
- Set options at store open
  - Migrate persistent stores automatically = YES
  - Infer mapping model automatically = YES

```
Cocoa error 134130: reason = “Can't find model for source store”
```

Renaming
How it works

- Set Renaming Identifier
- Example
  - Change song “name” to “title”



Entity	Abs	Cla
Album	☐	NSI
Artist	☐	NSI
Song	☐	NSI

Property	Kind	Type or
album	Relationship	Album
length	Attribute	String
title	Attribute	String

Attribute

Versioning

Ver. Hash Modifier:

Renaming Identifier: name

Other

☐ Index in Spotlight

☐ Store in External Record file

Tips

Core Data Model Versioning and Data Migration Programming Guide

- Transient to persistent == add new
- New attributes must be optional or have default value
- New relationships must be optional

Migration Post-Processing

Add custom, flexible logic

- Open store (with migration options)
- Check metadata for custom key, e.g. "DonePostProcessing"
- Do post-processing…
  - Populate derived attributes
  - Insert or delete objects
  - Set store metadata ("DonePostProcessing" = YES)
- Save changes and metadata

```
- (void)loadStoreWithMigration:(NSURL *)url {
    ...
    store = [psc addPersistentStoreWithType: NSSQLiteStoreType
        configuration: nil URL: url options: opts error: &err];

    m = [store metadata];
    key = @"DonePostProcessing";
    if (m && ([[m objectForKey: key] integerValue] < 2) ) {

        [self createNormalizedTitlesForBooksInContext: context];

        m2 = [[m mutableCopy] autorelease];
        [m2 setObject: [NSNumber numberWithInteger: 2]
            forKey: key];

        [store setMetadata: m2];

        ok = [context save:&err];
    }
}
```

Summary

- Core Data offers many paths to maturing your application
- Focus on a good initial model of your data
- Adapt to your evolving access patterns with incremental changes

http://bugreport.apple.com

More Information

Michael Jurewitz
Developer Tools Evangelist
jurewitz@apple.com

Core Data Documentation
Programming Guides, Examples, and Tutorials
http://developer.apple.com

Apple Developer Forums
http://devforums.apple.com

Related Sessions

Optimizing Core Data Performance on iPhone OS	Presidio Thursday 4:30PM

Labs

Core Data Lab	Application Frameworks Lab A Tuesday 4:30PM
Core Data Lab	Application Frameworks Lab B Wednesday 4:30PM
Core Data Lab	Application Frameworks Lab A Thursday 9:00AM

WWDC10

[redacted]