Optimizing Core Data Performance on iPhone OS

Melissa Turner

Roadmap

- Performance
  - Design
  - Analysis
- Managing
  - Space
  - Time

$$G\mu\nu = 8\pi T\mu\nu$$

Know Your Environment

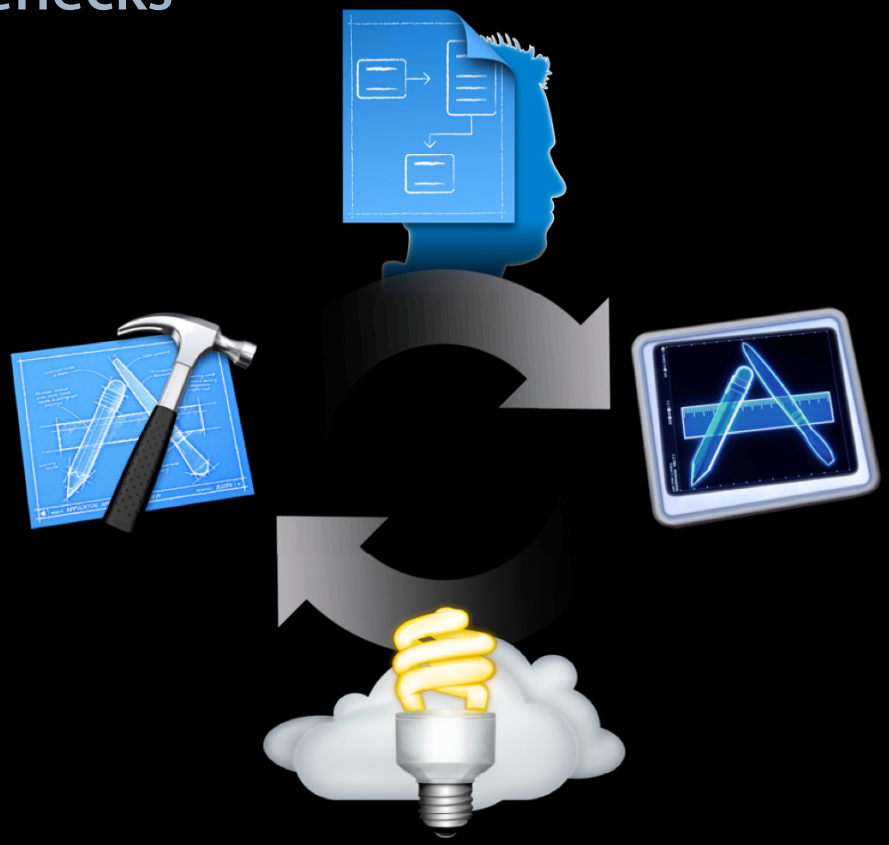
Different devices have different constraints

- Available memory
  - Memory bus speed
- Processor speed
- I/O speed
- Responsiveness requirements

Know Your Application

Expected vs. Unexpected Bottlenecks

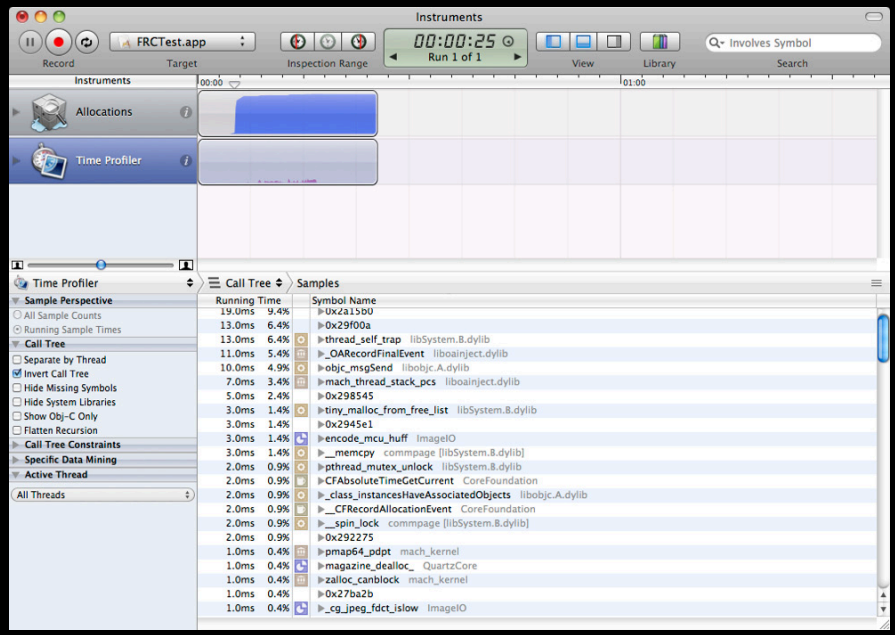
- Have a mental model
- Gather data
- Resolve disparities
- Repeat



Know Your Problem

Don’t optimize the 1% problem first

- Measure, measure, measure
  - Instruments for Simulator
  - Instruments on iPhone
  - CoreData logging



Instruments for Simulator

Examining your application's innards

- Not an emulator
- Memory use
- File system activity
- Core Data static probes

Instruments on iPhone

Examining your application's innards

- CPU use
- Graphics
- System activity
- Supplement with Core Data logging
  - -com.apple.CoreData.SQLDebug 1

Demo

Finding the bottleneck

In the Beginning

Importing data

Getting Started: Importing Data

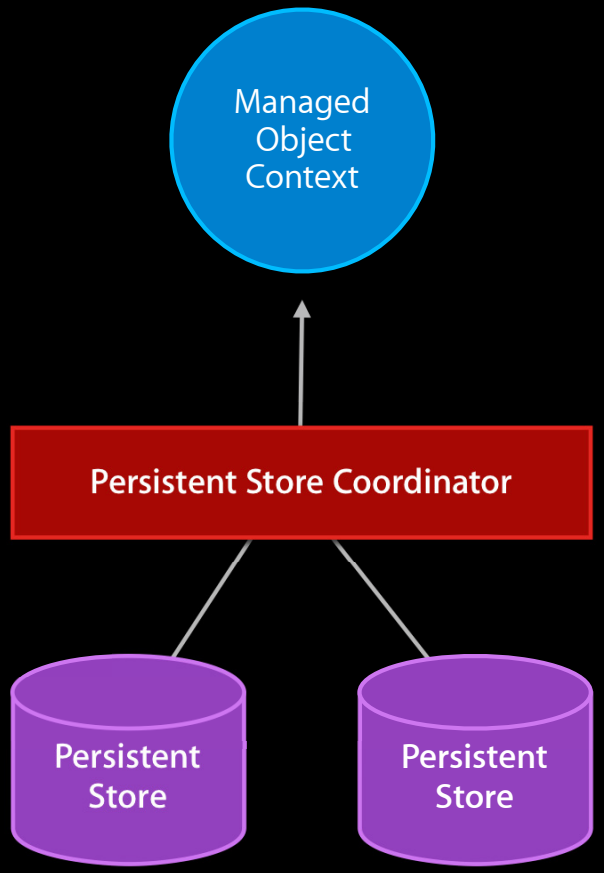
When and why?

- First time setup
- Application updates
- Standard workflow

Handling Large Imports

Avoid the import entirely

- Multiple stores
  - One store for seed data
  - One store for user data
- Best when data is disjoint

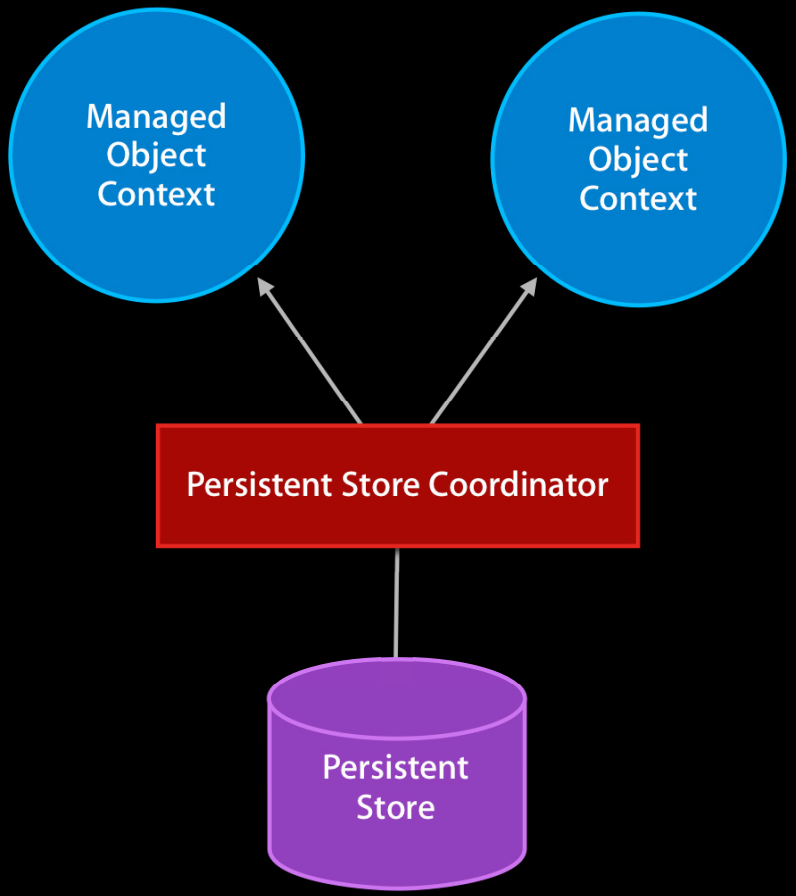


Handling Large Imports

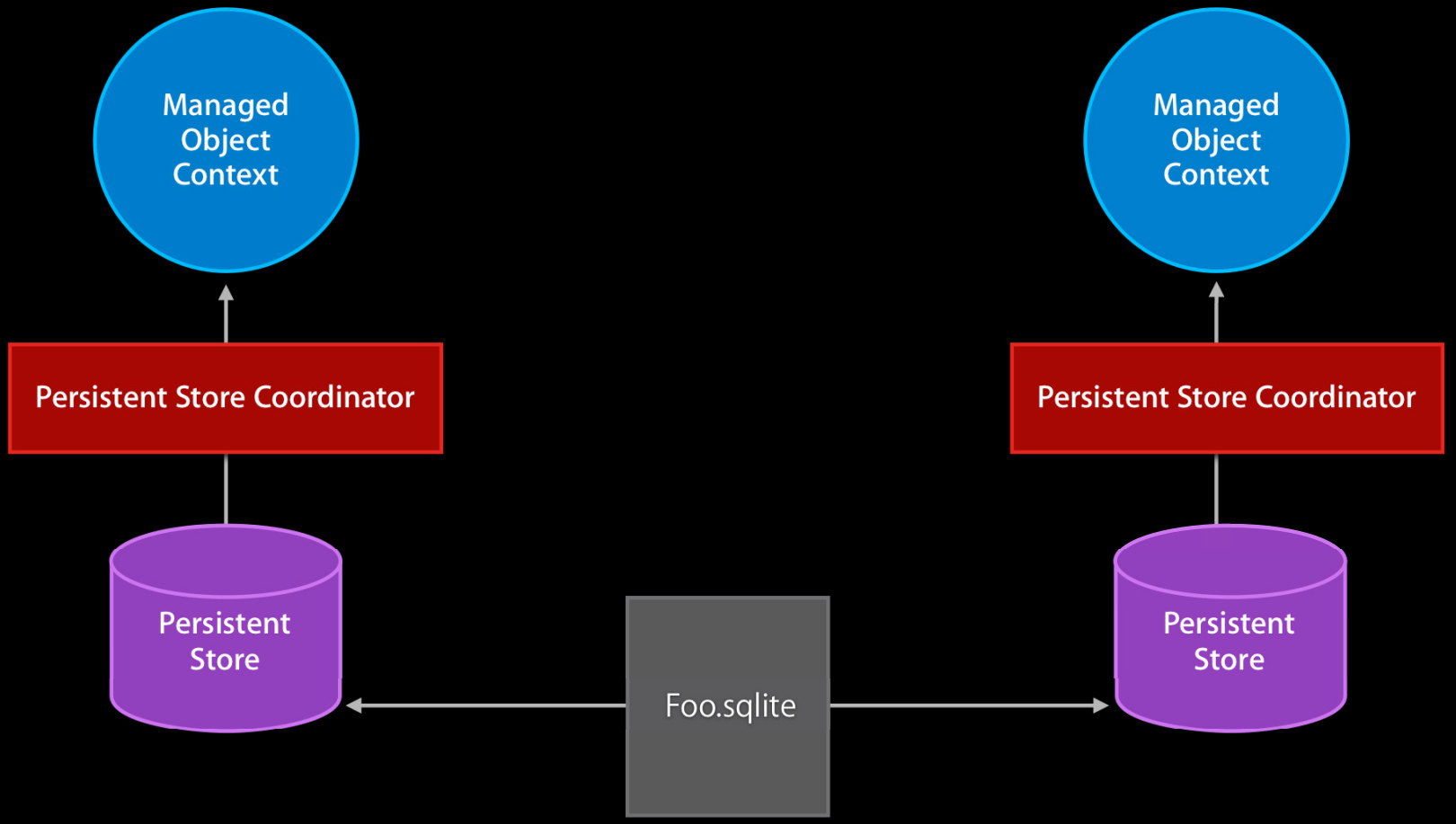
Strategies

- Use a separate context
- Do batch saves
- Use nested autorelease pools
- Avoid unnecessary fetching
  - Prefetch
  - Cache unique objects

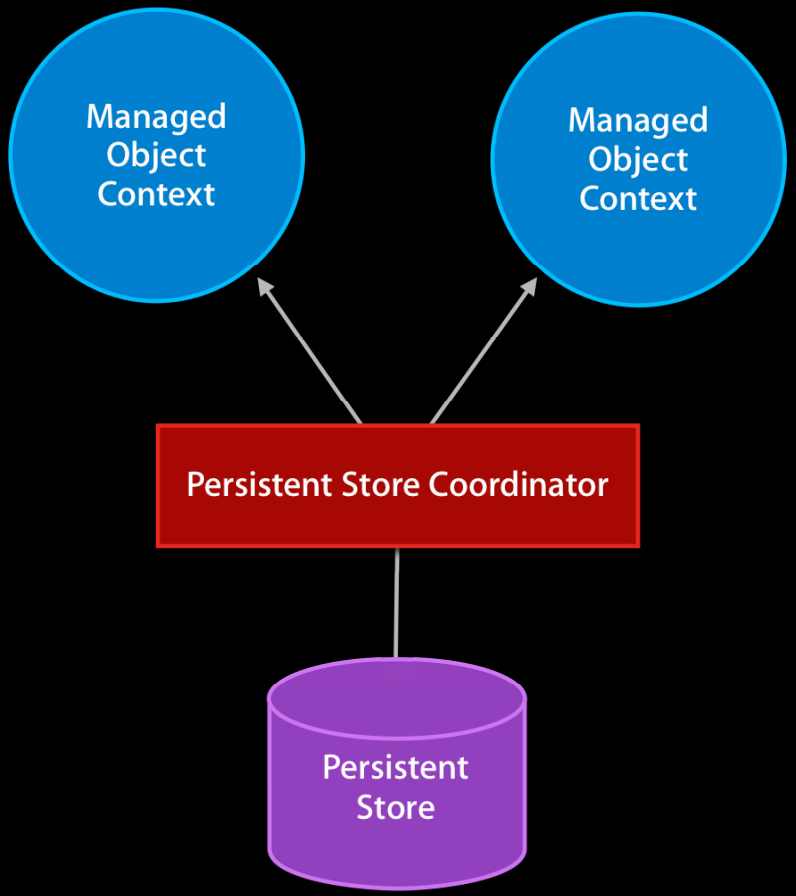
Peer Contexts



Parallel Stacks



Peer Contexts



Importing Data

```
- (void)createNewObjectsForContentsOfArray:(NSArray*)dictArray {
    NSManagedObjectContext *importContext = [self
        createManagedObjectContextForCoordinator: [self persistentStoreCoordinator]];
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    ...
    for (NSDictionary *dict in dictArray) {
      [self createAndPopulateObjectFromDictionary: dict inContext: importContext];
      ...
      if (timeToSave) {
        [context save: &error];
        [context reset];
        [pool drain];
        pool = [[NSAutoreleasePool alloc] init];
        }
    }
    [pool drain];
}
```

Importing Data

```
- (void)createNewObjectsForContentsOfArray:(NSArray*)dictArray {
    NSManagedObjectContext *importContext = [self
        createManagedObjectContextForCoordinator: [self persistentStoreCoordinator]];
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    ...
    for (NSDictionary *dict in dictArray) {
      [self createAndPopulateObjectFromDictionary: dict inContext: importContext];
      ...
      if (timeToSave) {
        [context save: &error];
        [context reset];
        [pool drain];
        pool = [[NSAutoreleasePool alloc] init];
        }
    }
    [pool drain];
}
```

Importing Data

```
- (void)createNewObjectsForContentsOfArray:(NSArray*)dictArray {
    NSManagedObjectContext *importContext = [self
        createManagedObjectContextForCoordinator: [self persistentStoreCoordinator]];
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    ...
    for (NSDictionary *dict in dictArray) {
      [self createAndPopulateObjectFromDictionary: dict inContext: importContext];
      ...
      if (timeToSave) {
        [context save: &error];
        [context reset];
        [pool drain];
        pool = [[NSAutoreleasePool alloc] init];
        }
    }
    [pool drain];
}
```

Importing Data

```
- (void)createNewObjectsForContentsOfArray:(NSArray*)dictArray {
    NSManagedObjectContext *importContext = [self
        createManagedObjectContextForCoordinator: [self persistentStoreCoordinator]];
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    ...
    for (NSDictionary *dict in dictArray) {
      [self createAndPopulateObjectFromDictionary: dict inContext: importContext];
      ...
      if (timeToSave) {
        [context save: &error];
        [context reset];
        [pool drain];
        pool = [[NSAutoreleasePool alloc] init];
        }
    }
    [pool drain];
}
```

Handling Large Imports

Strategies

- Use a separate context
- Do batch saves
- Use nested autorelease pools
- Avoid unnecessary fetching
  - Prefetch
  - Cache unique objects

Handling Large Imports

Strategies

- Use a separate context
- Do batch saves
- Use nested autorelease pools
- Avoid unnecessary fetching
  - Prefetch
  - Cache unique objects
- Use a background thread

End of the Line

Deleting data

End of the Line: Deleting Data

When and why?

- User initiated
- Application update
- Aging out

Handling Large Deletes

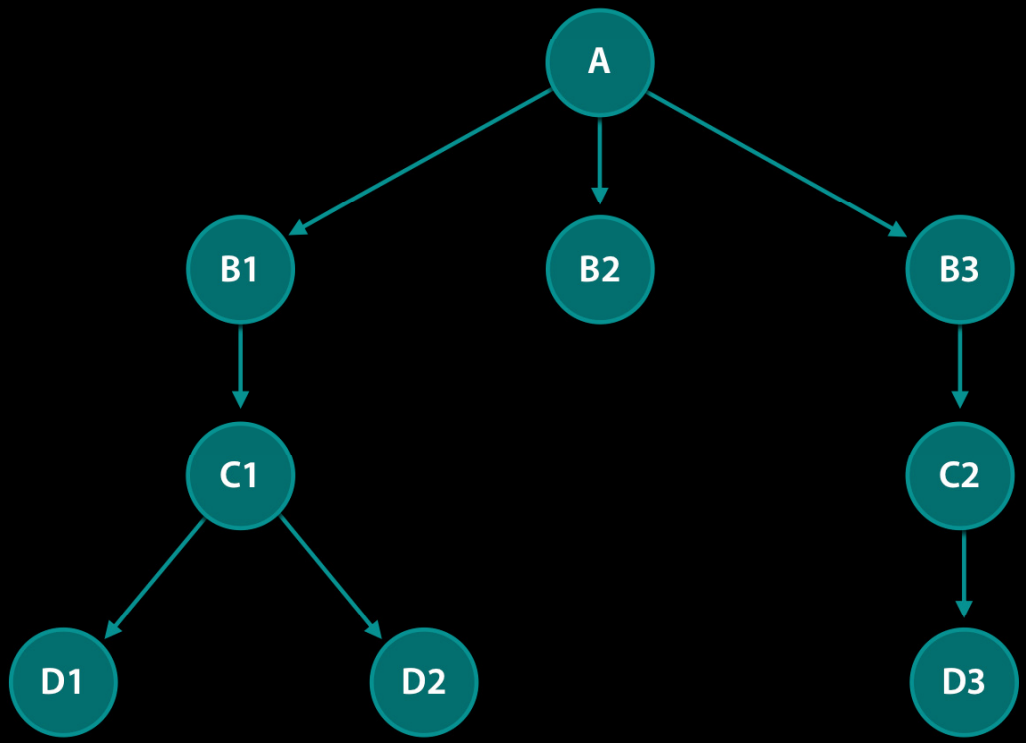
Improving delete performance

- Remove store
- Single store strategies
  - Batch deletes
  - Autorelease pool
  - Prefetch cascaded objects

Deleting Data

```
- (void)deleteObjects:(NSArray*) objectsToBeDeleted {
    for (NSManagedObject *obj in objectsToBeDeleted) {
      [[self managedObjectContext] deleteObject:anObject];
    }
    NSError *error = nil;
    if (![[self managedObjectContext] save: &error]) {
      // Handle error
    }
}
```

Deletes May Cascade



Deleting Data

```
- (void)deleteOb      :(NSArra      bjectsToBeDeleted {
    for (NSManaged      t *ob      objectsToBeDeleted) {
      [[self manage      t      xt] deleteObject:anObject];
    }
    NSError *error = n
    if (![[self manac      Context] save: &error]) {
      // Handle err
    }
}
```

Deleting Data

```
- (void)deleteObjects:(NSArray*)objectsToBeDeleted {
    NSFetchRequest *fetch = [[[NSFetchRequest alloc] init] autorelease];
    [fetch setEntity: [[model entitiesByName] valueForKey:@"Destination"]];
    [fetch setPredicate: [NSPredicate predicateWithFormat: @"source IN %@",
        objectsToBeDeleted]];
    [fetch setRelationshipKeyPathsForPrefetching:[NSArray arrayWithObjects:
        @"aRelationship", @"anotherRelationship", nil]];
    NSError *error = nil;
    NSArray *results = [context executeFetchRequest: request error:&error];
    if (nil == results) {
      // Handle error
    }
    for (NSManagedObject *obj in objectsToBeDeleted) {
      [[self managedObjectContext] deleteObject:anObject];
    }
    if (![context save: &error]) {
      // Handle error
    }
  }
```

Deleting Data

```
- (void)deleteObjects:(NSArray*)objectsToBeDeleted {
    NSFetchRequest *fetch = [[[NSFetchRequest alloc] init] autorelease];
    [fetch setEntity: [[model entitiesByName] valueForKey:@"Destination"]];
    [fetch setPredicate: [NSPredicate predicateWithFormat: @"source IN %@",
        objectsToBeDeleted]];
    [fetch setRelationshipKeyPathsForPrefetching:[NSArray arrayWithObjects:
        @"aRelationship", @"anotherRelationship", nil]];
    NSError *error = nil;
    NSArray *results = [context executeFetchRequest: request error:&error];
    if (nil == results) {
      // Handle error
    }
    for (NSManagedObject *obj in objectsToBeDeleted) {
      [[self managedObjectContext] deleteObject:anObject];
    }
    if (![context save: &error]) {
      // Handle error
    }
  }
```

Deleting Data

```
- (void)deleteObjects:(NSArray*)objectsToBeDeleted {
    NSFetchRequest *fetch = [[[NSFetchRequest alloc] init] autorelease];
    [fetch setEntity: [[model entitiesByName] valueForKey:@"Destination"]];
    [fetch setPredicate: [NSPredicate predicateWithFormat: @"source IN %@",
        objectsToBeDeleted]];
    [fetch setRelationshipKeyPathsForPrefetching:[NSArray arrayWithObjects:
        @"aRelationship", @"anotherRelationship", nil]];
    NSError *error = nil;
    NSArray *results = [context executeFetchRequest: request error:&error];
    if (nil == results) {
      // Handle error
    }
    for (NSManagedObject *obj in objectsToBeDeleted) {
      [[self managedObjectContext] deleteObject:anObject];
    }
    if (![context save: &error]) {
      // Handle error
    }
  }
```

Deleting Data

```
- (void)deleteObjects:(NSArray*)objectsToBeDeleted {
    NSFetchRequest *fetch = [[[NSFetchRequest alloc] init] autorelease];
    [fetch setEntity: [[model entitiesByName] valueForKey:@"Destination"]];
    [fetch setPredicate: [NSPredicate predicateWithFormat: @"source IN %@",
        objectsToBeDeleted]];
    [fetch setRelationshipKeyPathsForPrefetching:[NSArray arrayWithObjects:
        @"aRelationship", @"anotherRelationship", nil]];
    NSError *error = nil;
    NSArray *results = [context executeFetchRequest: request error:&error];
    if (nil == results) {
      // Handle error
    }
    for (NSManagedObject *obj in objectsToBeDeleted) {
      [[self managedObjectContext] deleteObject:anObject];
    }
    if (![context save: &error]) {
      // Handle error
    }
  }
```

Handling Large Deletes

Improving delete performance

- Remove store
- Single store strategies
  - Batch deletes
  - Autorelease pool
  - Prefetch cascaded objects

Handling Large Deletes

Improving delete performance

- Remove store
- Single store strategies
  - Batch deletes
  - Autorelease pool
  - Prefetch cascaded objects
  - Background thread

Optimal Concurrency

Getting it right

Concurrency

Why and when to use concurrency

- Avoid being terminated
- Large operations
- Network operations

Queues

How: Making it easy

- libdispatch
  - Default serial
- NSOperationQueue
  - Default concurrent
  - setMaxConcurrentOperationCount:1

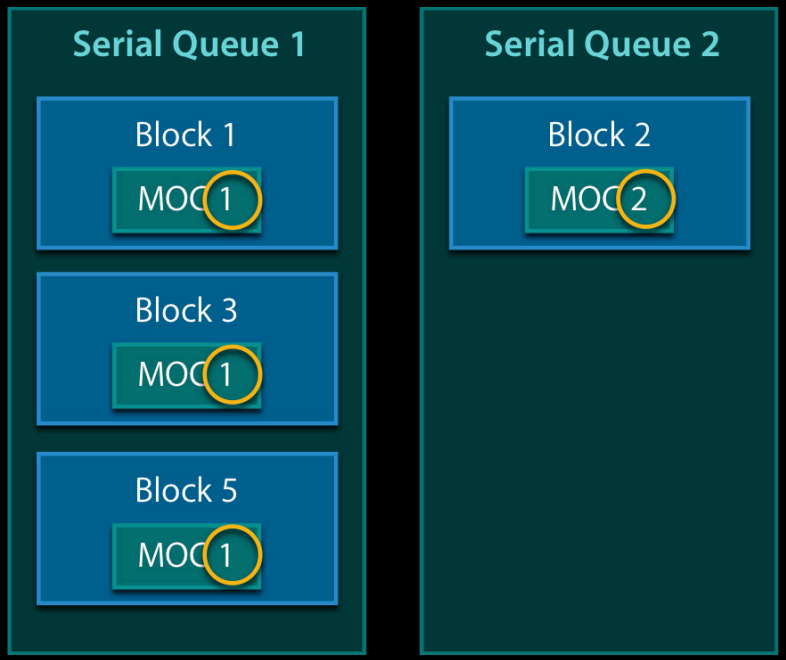
Concurrency and Core Data

Still running with scissors

- Context confined to thread
- Managed objects accessed from context's thread

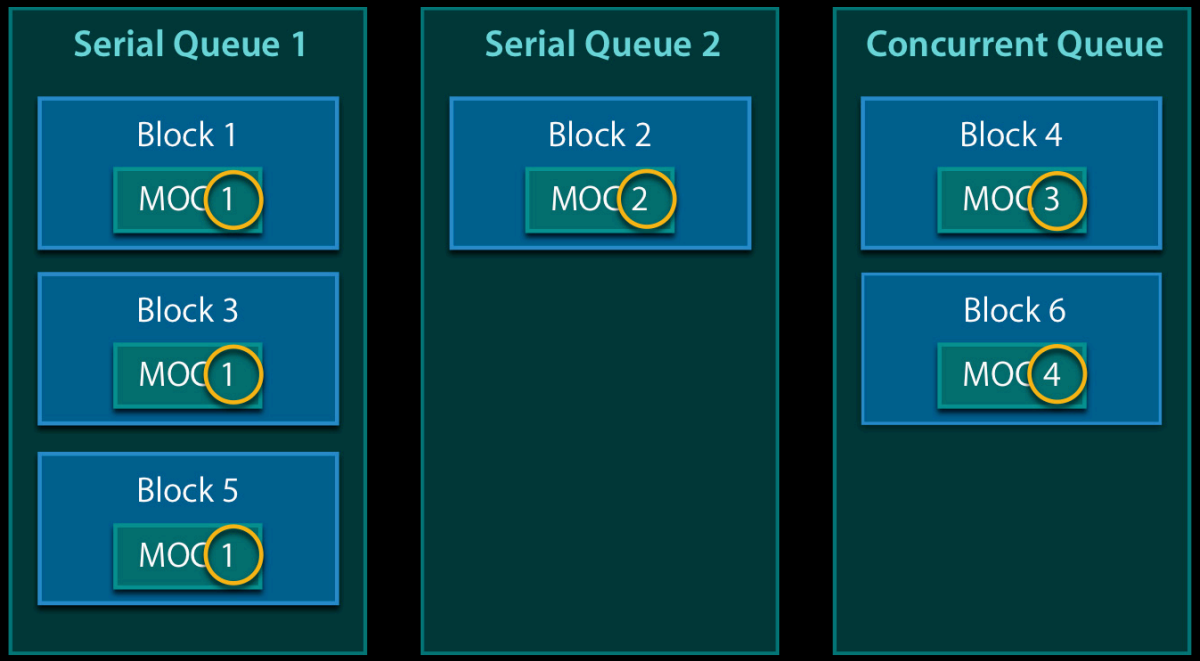
Managed Object Context Confinement

Threads or queues



Managed Object Context Confinement

Threads or queues



Concurrency and Core Data

Still running with scissors

- Background threads may terminate with work uncompleted
  - Have main thread postpone exit
  - Track completion and rerun if necessary

Importing Data

```
- (id)init {
  if (nil != (self = [super init])) {
    ...
    importQueue = [[NSOperationQueue alloc] init];
    [importQueue setMaxConcurrentOperationCount:1];
    [self startImport];
  }
  return self;
}

- (void)startImport {
  [importQueue addOperationWithBlock: ^{
      NSArray *newElements = [self loadDataForImport];
      if (nil != newElements) {
        [self createNewObjectsForContentsOfArray:newElements];
        [[NSOperationQueue mainQueue] addOperationWithBlock: ^{
            [self completeImport];}];
      }
    }];
}
```

Importing Data

```
- (id)init {
  if (nil != (self = [super init])) {
    ...
    importQueue = [[NSOperationQueue alloc] init];
    [importQueue setMaxConcurrentOperationCount:1];
    [self startImport];
  }
  return self;
}

- (void)startImport {
  [importQueue addOperationWithBlock: ^{
      NSArray *newElements = [self loadDataForImport];
      if (nil != newElements) {
        [self createNewObjectsForContentsOfArray:newElements];
        [[NSOperationQueue mainQueue] addOperationWithBlock: ^{
            [self completeImport];}];
      }
    }];
}
```

Importing Data

```
- (id)init {
  if (nil != (self = [super init])) {
    ...
    importQueue = [[NSOperationQueue alloc] init];
    [importQueue setMaxConcurrentOperationCount:1];
    [self startImport];
  }
  return self;
}

- (void)startImport {
  [importQueue addOperationWithBlock: ^{
      NSArray *newElements = [self loadDataForImport];
      if (nil != newElements) {
        [self createNewObjectsForContentsOfArray:newElements];
        [[NSOperationQueue mainQueue] addOperationWithBlock: ^{
            [self completeImport];}];
      }
    }];
}
```

Importing Data

```
- (id)init {
  if (nil != (self = [super init])) {
    ...
    importQueue = [[NSOperationQueue alloc] init];
    [importQueue setMaxConcurrentOperationCount:1];
    [self startImport];
  }
  return self;
}

- (void)startImport {
  [importQueue addOperationWithBlock: ^{
      NSArray *newElements = [self loadDataForImport];
      if (nil != newElements) {
        [self createNewObjectsForContentsOfArray:newElements];
        [[NSOperationQueue mainQueue] addOperationWithBlock: ^{
            [self completeImport];}];
      }
    }];
}
```

Importing Data

```
- (id)init {
  if (nil != (self = [super init])) {
    ...
    importQueue = [[NSOperationQueue alloc] init];
    [importQueue setMaxConcurrentOperationCount:1];
    [self startImport];
  }
  return self;
}

- (void)startImport {
  [importQueue addOperationWithBlock: ^{
      NSArray *newElements = [self loadDataForImport];
      if (nil != newElements) {
        [self createNewObjectsForContentsOfArray:newElements];
        [[NSOperationQueue mainQueue] addOperationWithBlock: ^{
            [self completeImport];}];
      }
    }];
}
```

Communicating Changes Between Threads

- Pass object IDs, not managed objects
- NSManagedObjectContextDidSave notification

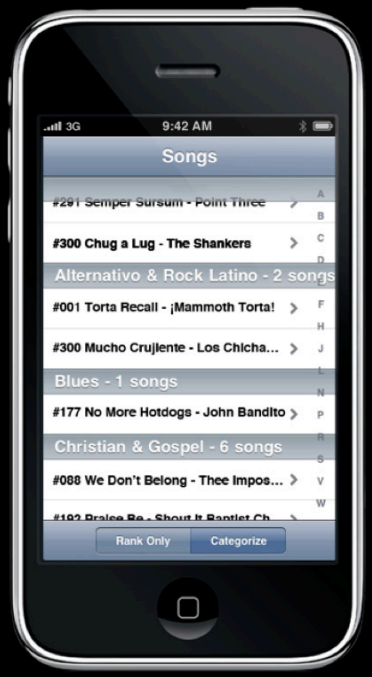
In the Middle: Memory

Managing space

Fetch What You Need

When you will need an object eventually

- Batching
- Partial faults



Fetch What You Need

When you will need an object eventually

- Batching
- Partial faults

Fetch What You Need

When you will need an object eventually

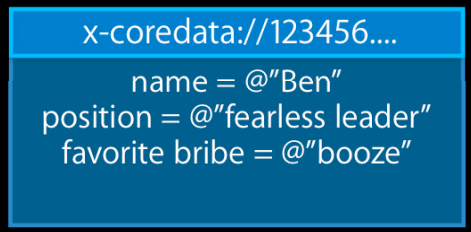
- Batching
- Partial faults

name = @"Ben"
position = @"fearless leader"
favorite bribe = @"booze"

Fetch What You Need

When you will need an object eventually

- Batching
- Partial faults
- Object IDs



Not What You Don't

When you need meta-information

- countForFetchRequest: with limit
- existingObjectWithID:error:
- Dictionary results
  - Unique values
  - Aggregates

`NSFetchRequest.h`

Let It Go When You're Done

- Use autorelease pools
- Prune the object graph
  - [context reset]
  - [context refreshObject: object mergeChanges: NO]

In the Middle: CPU

Managing time

About That Database

SQL: Structured Query Language

- Let the database do the filtering
- Let the database do the sorting
- Let the database do the calculations

Predicates and Strings

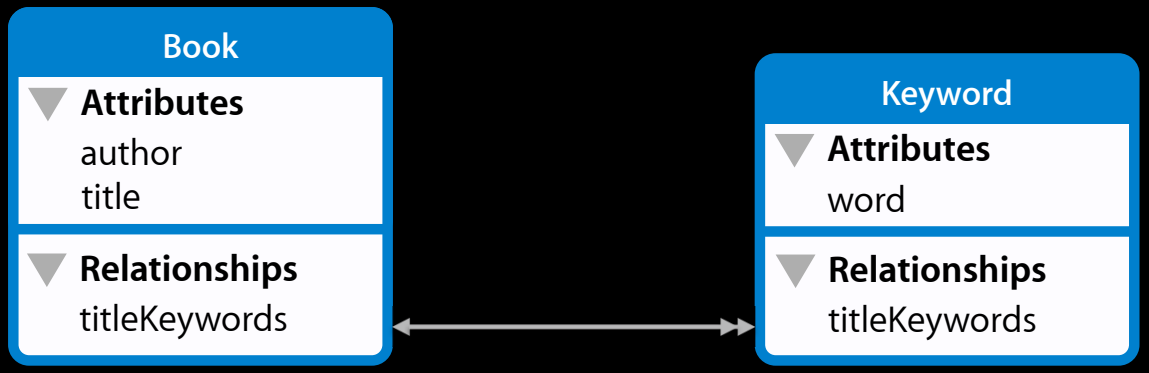
Techniques for minimizing computation overhead

- Unicode aware string comparisons are expensive
  - Use normalized derived attribute
- Use <= and < instead of BEGINSWITH
- Prefer prefix searching
  - Avoid ENDSWITH, CONTAINS, LIKE, MATCHES if possible

```
name BEGINSWITH[cd] @"franç"
@"franc" <= searchName AND searchName < @"frand"
```

Keyword Searching

- Normalize keywords into separate entity
  - Search on new entity
  - Follow relationship back




```
wordtitle@"CONTAINS[cd]@"tree"
```

Effective Predicates

Speeding up your queries

- Use IN instead of OR
- Order complex predicates last
  - String queries
  - Relationship traversal

```
[NSPredicate predicateWithFormat:
@"author = %@ OR author = %@",
author1, author2]

[NSPredicate predicateWithFormat:
@"author IN %@", authorArray]

@"date < 1989/06/10 AND name
MATCHES '.*[A-Z]+.*'"
```

Subqueries

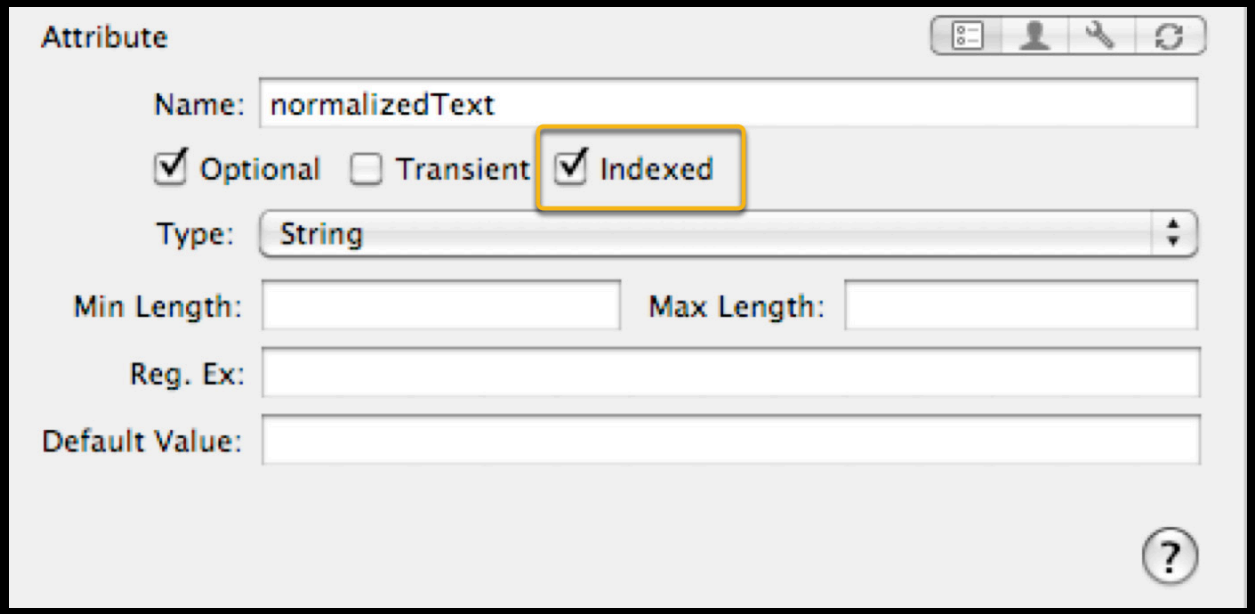
The exception to “don’t cross relationships”

- Correlated subqueries
  - Filtering by relationship content
- Relationship aggregates

```
SUBQUERY(roommates, $roommate,
  $roommate.YES.firstName <= @"jane" AND
  $roommate.lastName == @"doe").@count != 0
```

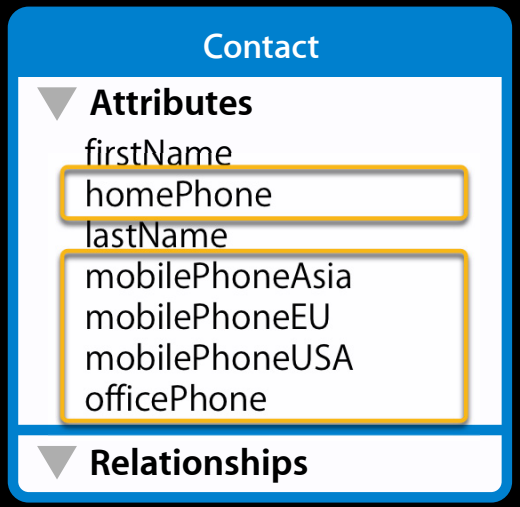
Make Model Reflect Workflow

Use indices when it makes sense



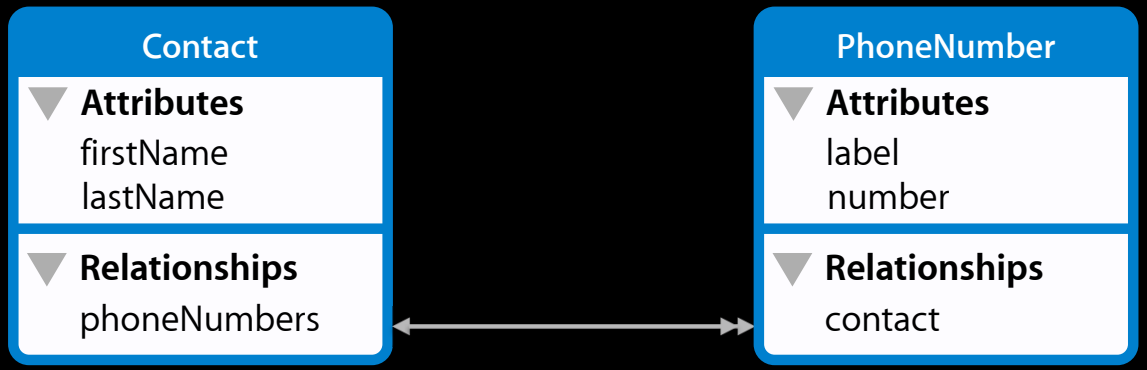
Make Model Reflect Workflow

Normalization: Separate unlike things



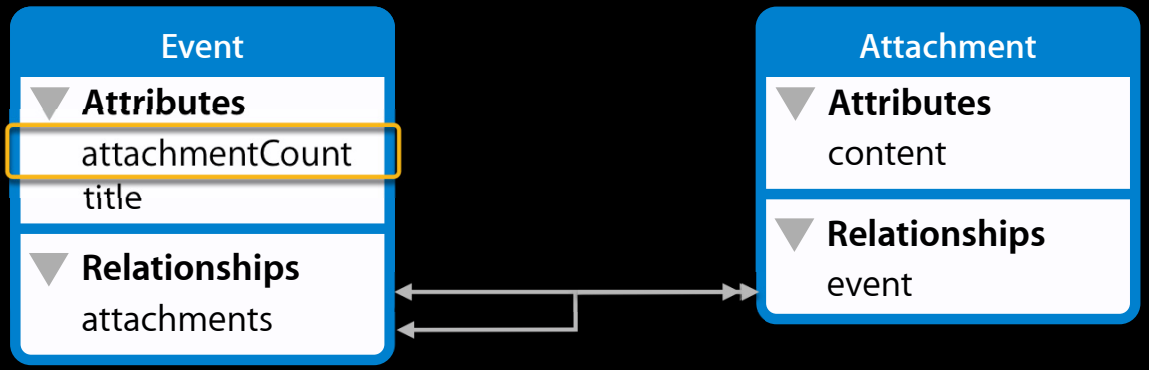
Make Model Reflect Workflow

Normalization: Separate unlike things



Make Model Reflect Workflow

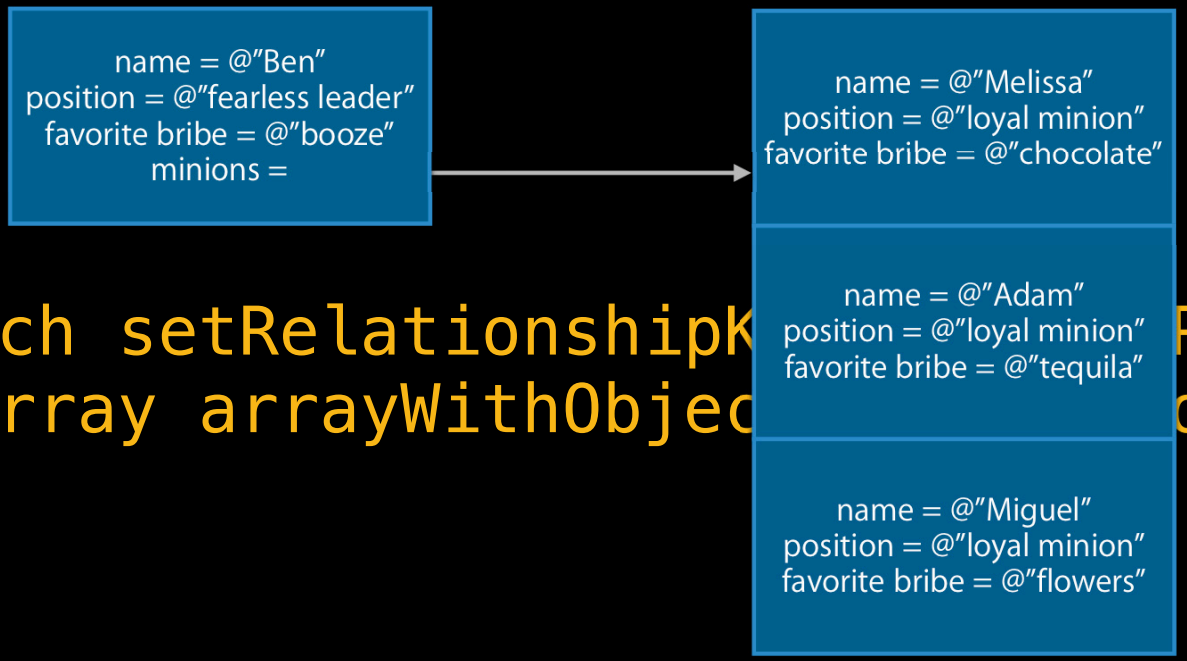
Denormalization: Avoid unnecessary joins



Minimize I/O Overhead

Make fewer trips to the database

- Relationship prefetching

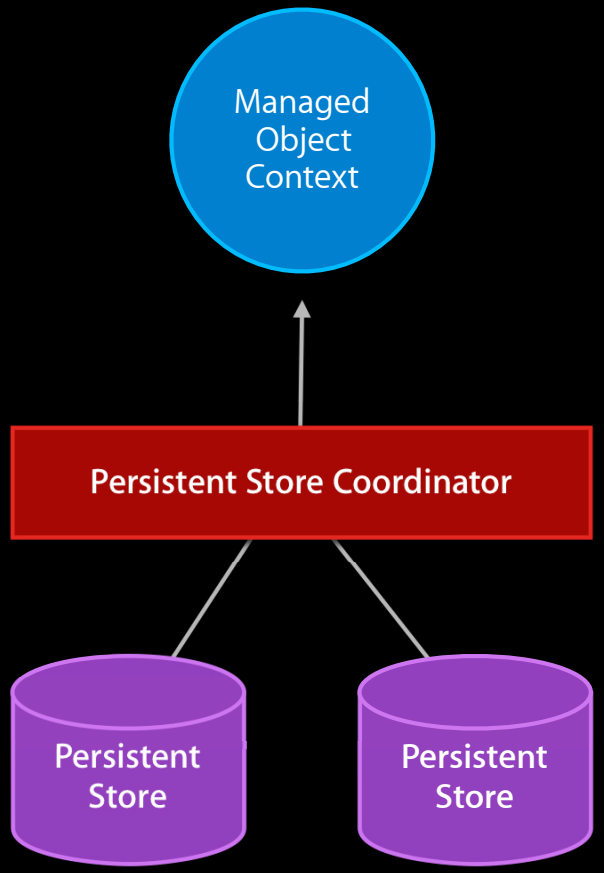



```
[fetch setRelationshipK                Prefetching:
[NSArray arrayWithObjec                ons", nil]];
```

Minimize I/O Overhead

Make fewer trips to the database

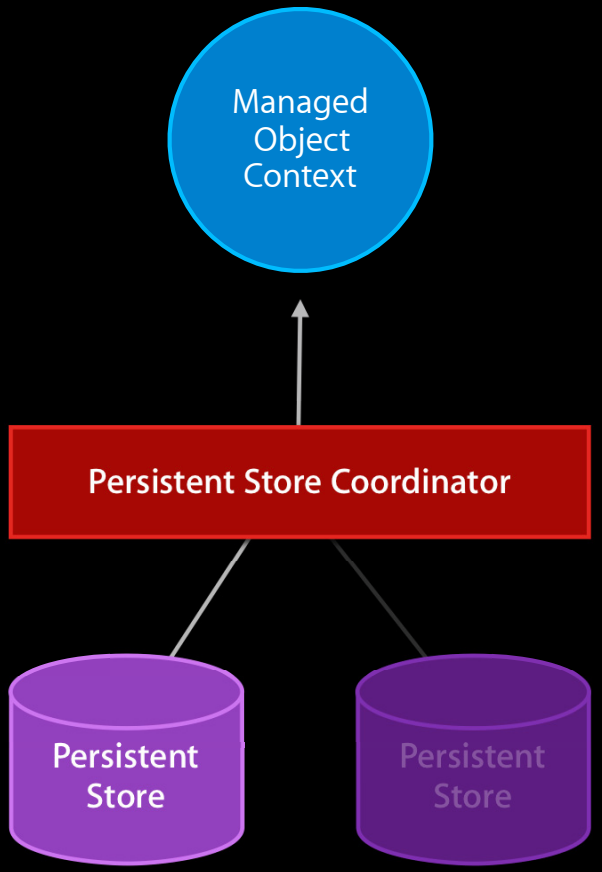
- Relationship prefetching
- setAffectedStores:



Minimize I/O Overhead

Make fewer trips to the database

- Relationship prefetching
- setAffectedStores:

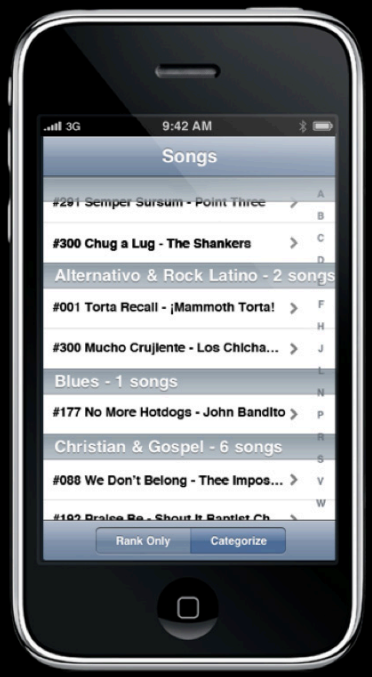


In the Middle: Fetched Results Controller

Managing time

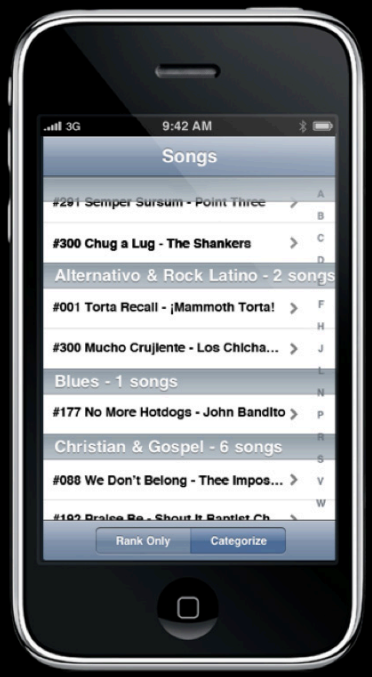
In the Middle: Fetched Results Controller

- Manages your data for you
  - Integrated with UITableView
  - Manages sections
  - Does change tracking



In the Middle: Fetched Results Controller

- Easy to configure
  - NSFetchRequest
  - Section name keypath (optional)
  - Delegate (optional)
  - Cache name (optional)



NSFetchedResultsController

Change tracking

- Content automatically updated
- Enabled by setting delegate
  - In 3.2 and later delegate must respond to at least one of the delegate change tracking methods
- Not supported for NSDictionaryResultsType

See Core Data Navigation based application template

Fetch Result Controller

Caching

- Persistent caching of fetch results
  - Speed application launch
- Enabled by setting cache name
  - Different cache name for different queries
- Fetch request MUST not be changed
  - Includes the predicate

http://bugreport.apple.com

- We don't know unless you tell us
- Bugs fixed faster with
  - Steps to reproduce
  - Sample project
- Also use for
  - Feature requests
  - Enhancement requests
  - Performance issues
  - Documentation requests

Summary

- Performance Analysis
- Managing Space
- Managing Time
- $\mathbf{G}\mu\nu = 8\pi\mathbf{T}\mu\nu$

Puppy. Also Kitten.

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


WWDC10