CoreData Multithreading

First things first

Arguments Passed On Launch

- -com.apple.CoreData.SQLDebug [1,2,3]
- -com.apple.CoreData.SyntaxColoredLogging 1
- -com.apple.CoreData.SQLiteIntegrityCheck 1
- -com.apple.CoreData.ThreadingDebug [1,2,3]
- -com.apple.CoreData.SQLiteDebugSynchronous [0,1,2]

Let's dive...

NSManagedObjectContext

You should not initialize a context on one thread then pass it to a different thread.

— Apple Documentation

NSManagedObject

— Is associated with one context

— Accessing is not thread safe

Getting an object in different thread

— Passing object is forbidden

— Only *safe* way using NSManagedObjectID

```
NSManagedObject *myObject = [mainContext insert...];
NSManagedObjectID *objectID = [myObject objectID];
...
dispatch_async(dispatch_get_global_queue(...), ^{
    NSManagedObjectContext *context = ... // get context
    NSManagedObjectID *objectIDInOtherThread = [context objectWithID:objectID];
}
```

One step backwards...

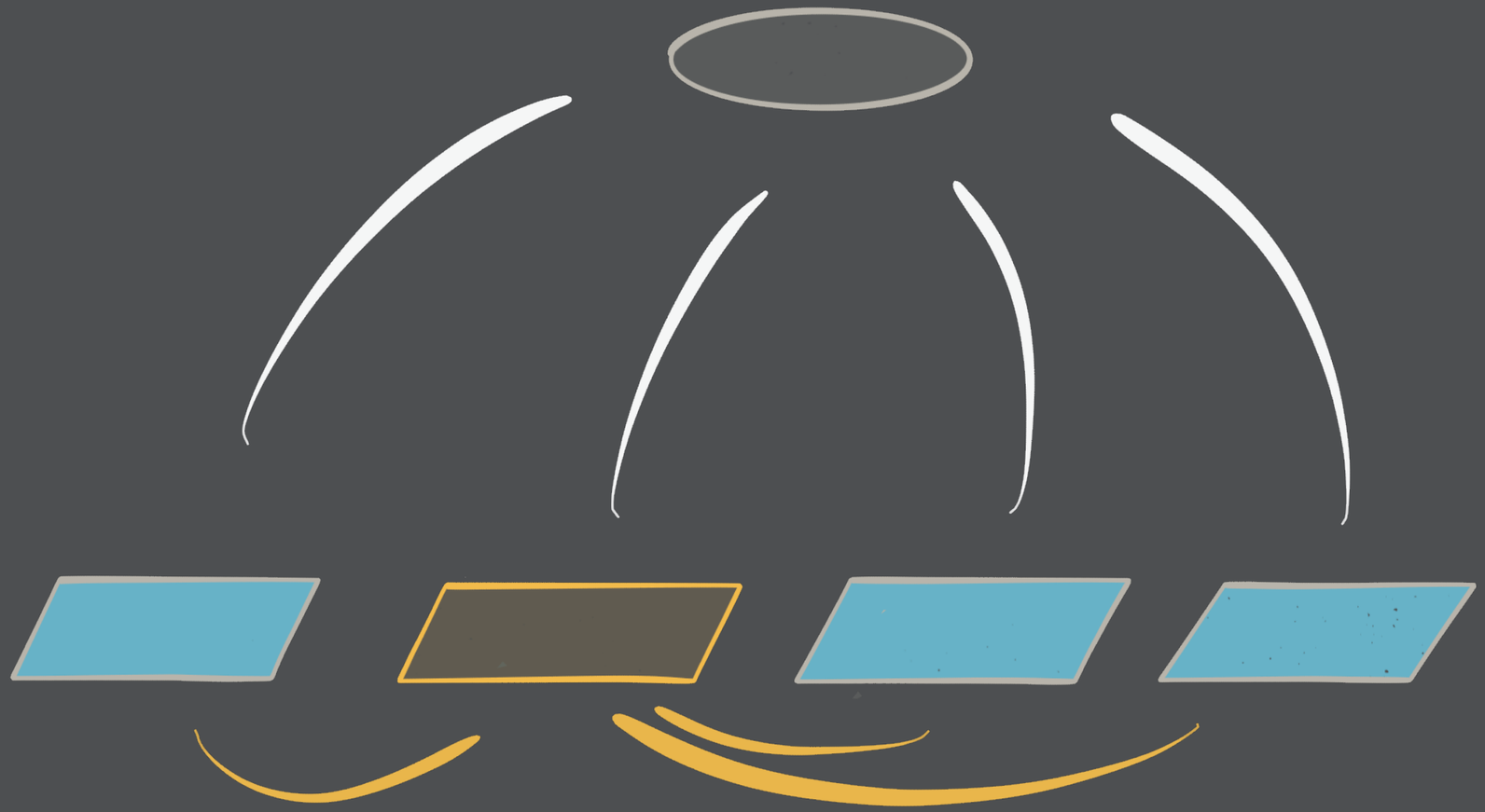
Concurrency models

- Confinement contexts model (old)
- Parent-child contexts model (current)

Confinement queue contexts model (old)

- One context per thread
- NSConfinementConcurrencyType
- save: on worker context sends notification
- mergeChangesFromContextDidSaveNotification: on main context to merge changes from worker context

Confinement



Confinement Background thread

```
NSManagedObjectContext *backgroundContext = [[NSManagedObjectContext alloc]
                        initWithConcurrencyType:NSConfinementConcurrencyType];
backgroundContext.persistentStoreCoordinator = ...; // PSC passed around
... // actual work
[backgroundContext save:NULL];
```

Confinement Merging main

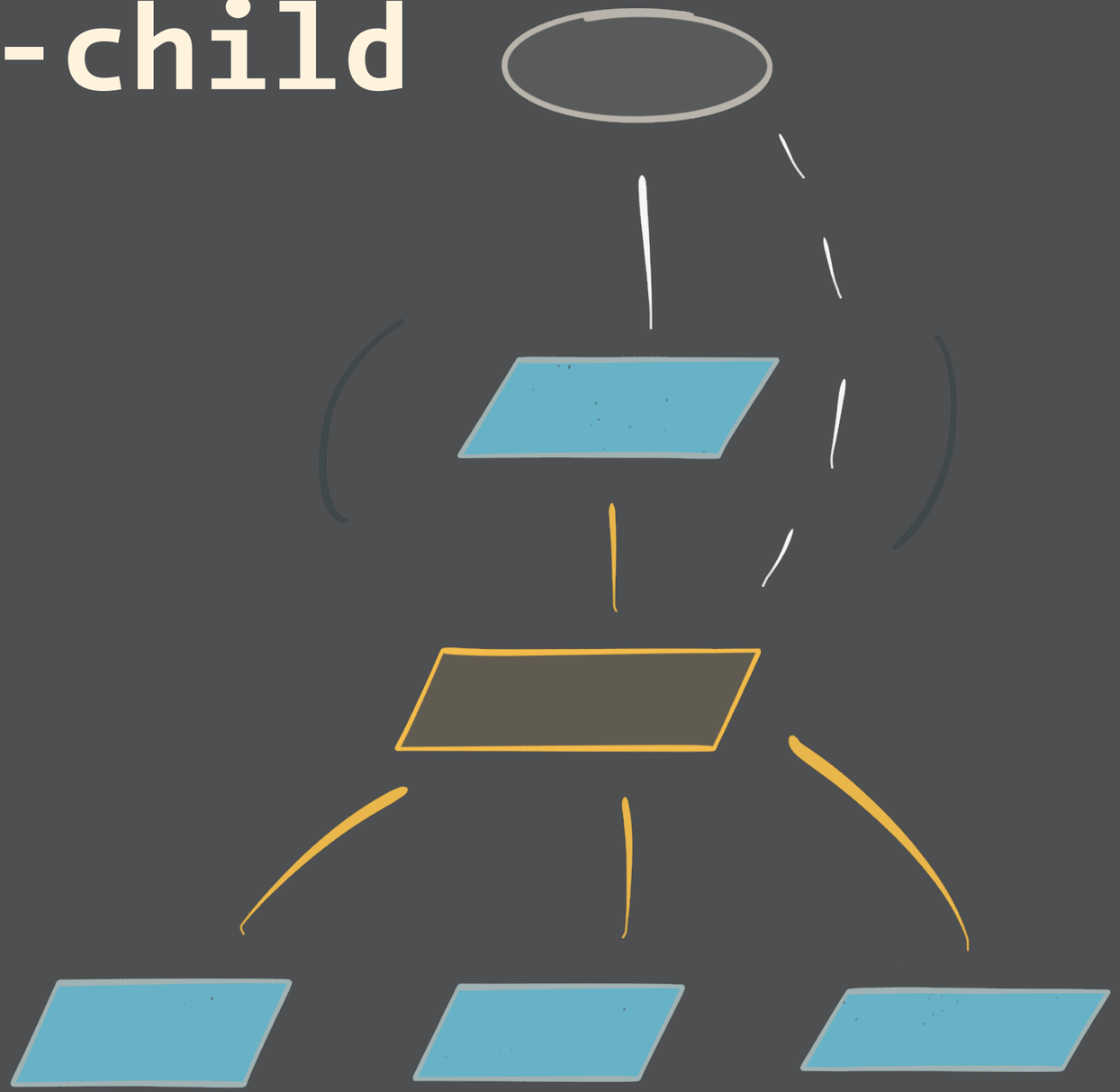
```
// Somewhere in initialization
[[NSNotificationCenter defaultCenter] addObserver:self
    selector:@selector(contextHasChanged:) name:NSManagedObjectContextDidSaveNotification object:nil];

...
- (void)contextHasChanged:(NSNotification*)notification
{
  if ([notification object] == [self mainObjectContext]) return;
  if (![NSThread isMainThread]) {
    [self performSelectorOnMainThread:@selector(contextHasChanged:) withObject:notification waitUntilDone:YES];
    return;
  }
  [[self mainObjectContext] mergeChangesFromContextDidSaveNotification:notification];
}
```

Parent-child model (current)

- `MainQueueConcurrencyType` and `PrivateQueueConcurrencyType`
- Many worker contexts - one per thread
- Every worker context is *child* of main
- save: on *child* context merges changes to the parent context

Parent-child



Sample stack setup

#1

1. Private Queue Context - writes to disk
2. Main Queue Context - context that lives on main thread, used for all user interactions
3. Zero or many Private Queue child contexts with Main as parent, created on demand

#1 Creating contexts

```
NSURL *modelURL = [[NSBundle mainBundle] URLForResource:@"ModelName" withExtension:@"momd"];
NSManagedObjectModel *mom = [[NSManagedObjectModel alloc] initWithContentsOfURL:modelURL];

NSPersistentStoreCoordinator *coordinator = [[NSPersistentStoreCoordinator alloc] initWithManagedObjectModel:mom];

self.privateContext = [[NSManagedObjectContext alloc] initWithConcurrencyType:NSPrivateQueueConcurrencyType];
self.privateContext.persistentStoreCoordinator = coordinator;

self.mainContext = [[NSManagedObjectContext alloc] initWithConcurrencyType:NSMainQueueConcurrencyType];
self.mainContext.parentContext = self.privateContext;
```

#1 Creating store

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_BACKGROUND, 0), ^{
        NSPersistentStoreCoordinator *psc = self.privateContext.persistentStoreCoordinator;

        // setup options

        NSURL *documentsURL = [self documentsURLFromFileManager];
        NSURL *storeURL = [documentsURL URLByAppendingPathComponent:@"StoreName.sqlite"];

        NSError *error = nil;
        if(![psc addPersistentStoreWithType:NSSQLiteStoreType configuration:nil URL:storeURL options:options error:&error]) {
            // handle error
            return;
        }

        // update UI
        if (!self.initCallback) return;
        dispatch_sync(dispatch_get_main_queue(), self.initCallback);
});
```

#1 Save

```
- (void)save {
    if (![self.privateContext hasChanges] && ![self.mainContext hasChanges]) return;

    [self.mainContext performBlockAndWait:^{
        NSError *error = nil;
        if (![self.mainContext save:&error]) {
            // handle error
        }

        [self.privateContext performBlock:^{
            NSError *privateError = nil;
            if (![self.privateContext save:&privateError]) {
                // handle error
            }
        }];
    }];
}
```

#2 (alternative)

1. Private Queue Context - writes to disk
2. Main Queue Context - context that lives on main thread, used for all user interactions
3. One background Private Queue Context for all background operations

#2 Creating contexts

```
self.privateContext = [[NSManagedObjectContext alloc] initWithConcurrencyType:NSPrivateQueueConcurrencyType];
self.privateContext.persistentStoreCoordinator = coordinator;

// main is public
self.mainContext = [[NSManagedObjectContext alloc] initWithConcurrencyType:NSMainQueueConcurrencyType];
self.mainContext.parentContext = self.privateContext;

// background is public
self.backgroundContext = [[NSManagedObjectContext alloc] initWithConcurrencyType:NSPrivateQueueConcurrencyType];
self.backgroundContext.parentContext = self.mainContext;
```

#2 Save

— Manual

— NSManagedObjectContext subclass

```
- (BOOL)save:(NSError **)error {
    if !([super save:error]) {
        // handle error
        return NO;
    }

    if (self.parentContext) {
        [self.parentContext performBlock:^{
            [self.parentContext save:error];
        }];
    }
    return YES;
}
```

Worker context: one vs. many

Accessing objects

Block API

- `performBlock:`
- `performBlockAndWait:^{ /* actual work */ }`

Working with objects

```
Employee *employee = ...; // Get object on main thread
...
NSManagedObjectID *employeeID = [employee objectID]; // Get in the proper context!
...
[privateQueueContext performBlock:^{
    Employee *safeEmployee = [privateQueueContext objectWithID:employeeID];
    ...
    // background work
    ...
    [privateQueueContext save:NULL];
}];
```

Hands on!

Assignment

- Check out branch: multithreading-assignment-1
- Go to Model Contoller and find parseResponseData:completion: method
- This method is now called in background (see updateDataWithCompletion:)
- Implement correct concurrent version of parsing code

Assignment Solution

Branch: multithreading-assignment-1-solution

Thanks!