iOS App Performance

Responsiveness

Session 235

Ben Nham
iOS Performance

Tim Lee
iOS Performance

Introduction

- Responsiveness: How quickly app reacts to user actions
- Performance: Getting an app's work done efficiently

What You'll Learn

- Measuring performance
- Fast app launch
- Performance strategies
- Speedy event handling

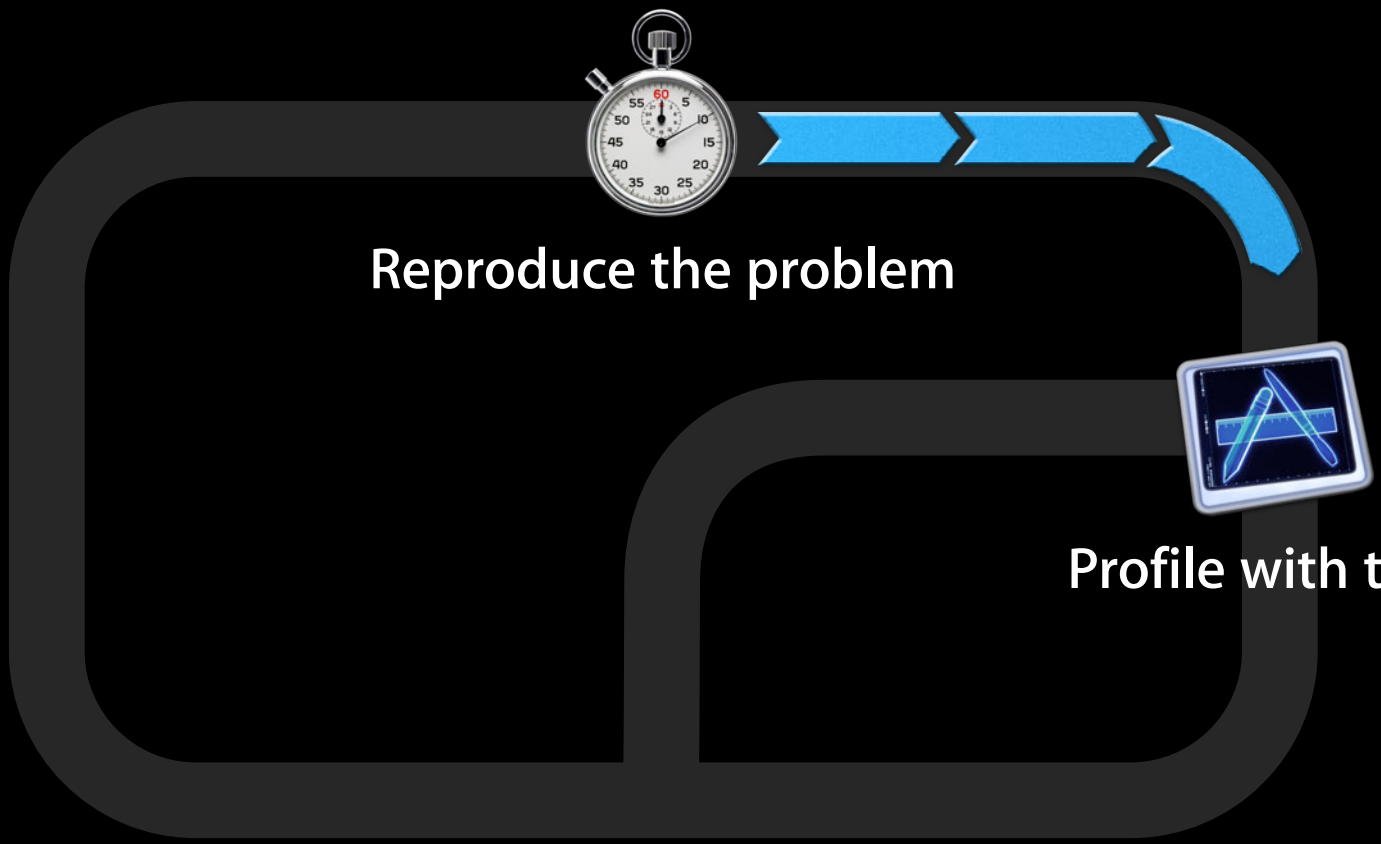
Performance Workflow

Performance Workflow

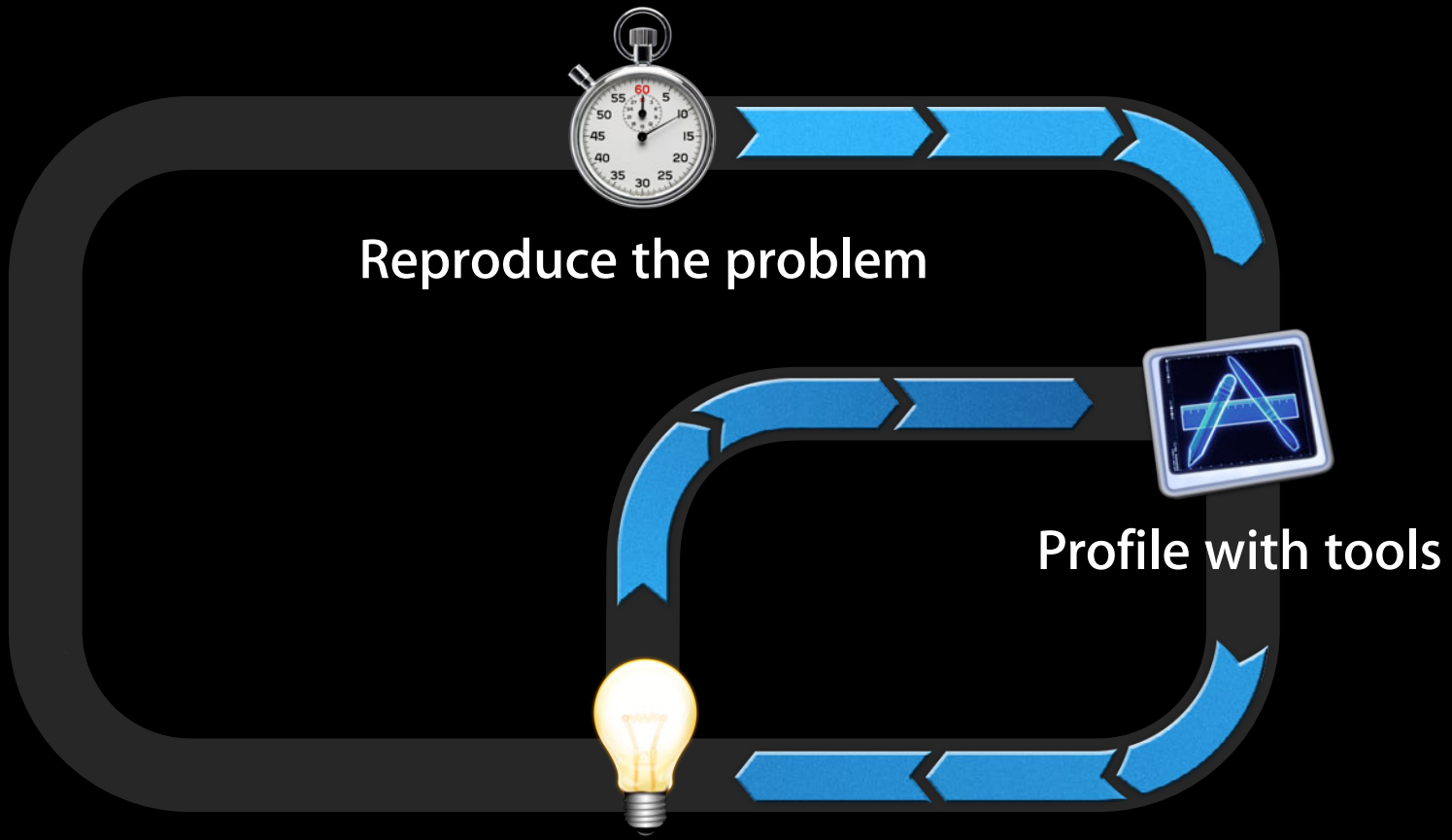
Performance Workflow

Reproduce the problem

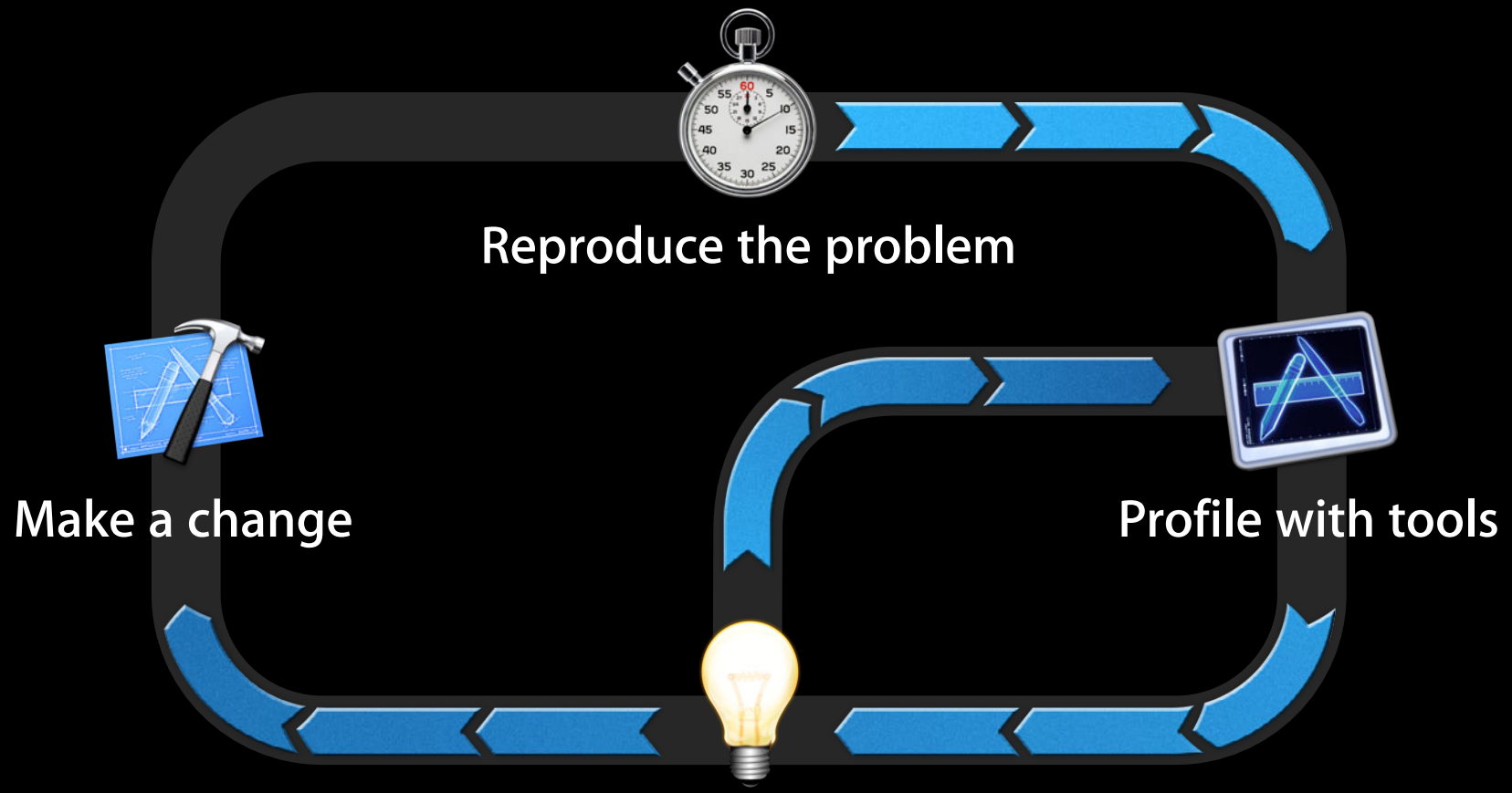
Performance Workflow



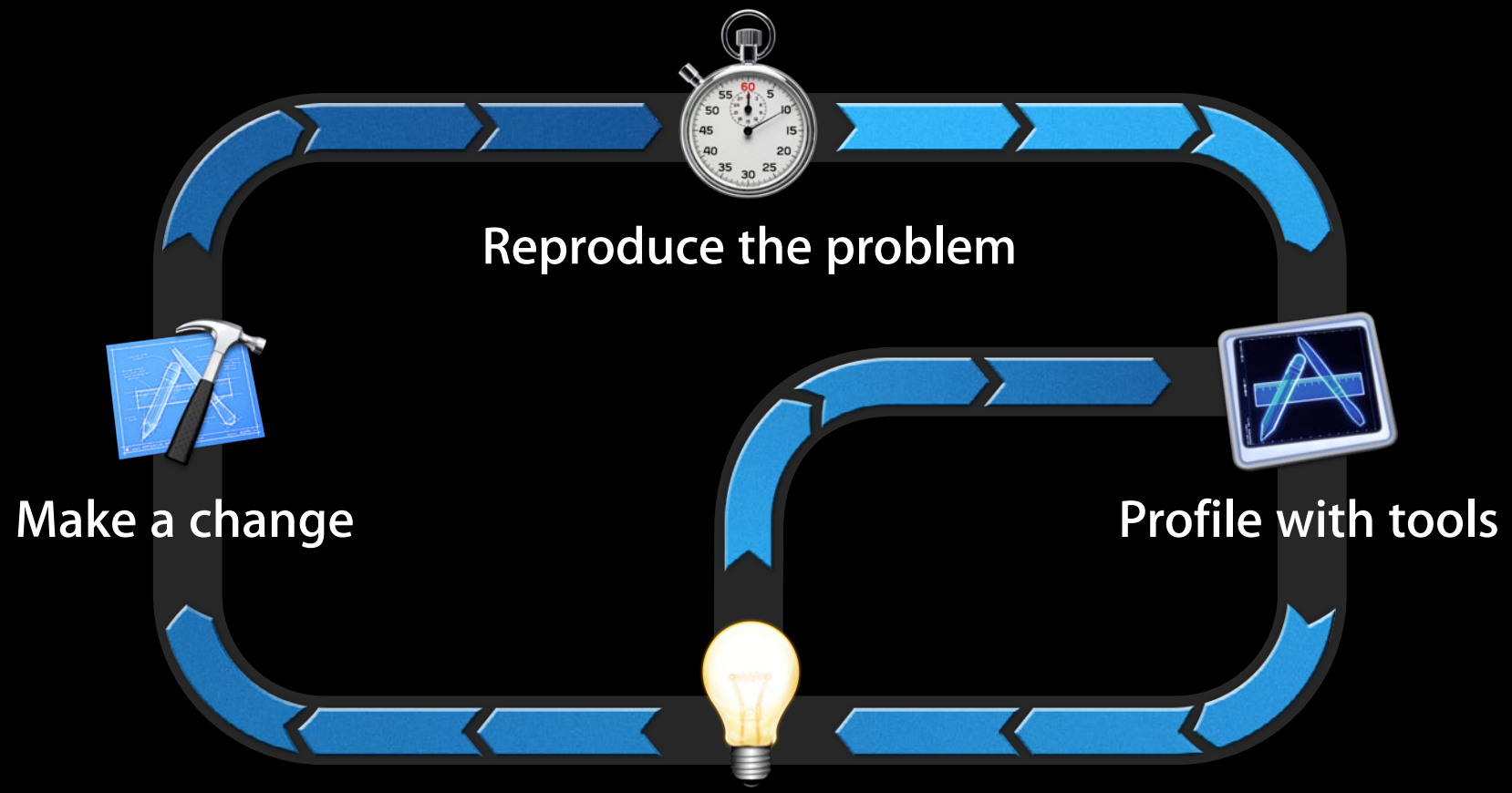
Performance Workflow



Performance Workflow



Performance Workflow



App Launch

App Launch

- Launch time is the first measure of responsiveness
- Apps are launched concurrently with zoom animation
  - 400 ms on iPhone
  - 500 ms on iPad
- Strive for “instant” app launch

Beware the Watchdog

- System watchdog terminates app if it launches slowly
- Xcode disables watchdog while debugging
- Users give up before timeout

Scenario	Watchdog Timeout
Launch	20 seconds
Resume	10 seconds
Suspend	10 seconds
Quit	6 seconds
Background task	10 minutes

Measuring Launch Time

Choose an endpoint

- Watchdog cares about end of first CATransaction
  - First layout and draw
  - Currently in `-[UIApplication _reportAppLaunchFinished]`
- Users may care about another metric
  - Camera app should measure time to enabling shutter

Measuring Launch Time

Logging time to first frame

- Get start time in main()

```
int main(int argc, char **argv) {
    StartTime = CFAbsoluteTimeGetCurrent();
```

- Stop timer after launch run loop

```
- (void)applicationDidFinishLaunching:(UIApplication *)app {
    dispatch_async(dispatch_get_main_queue(), ^{
      NSLog(@"Launched in %f sec", CFAbsoluteTimeGetCurrent() - StartTime);
    });
```

Measuring Launch Time

Using Time Profiler to measure time to first frame

- Switch to CPU strategy view
- Search for `-[UIApplication _reportAppLaunchFinished]`
- Find last sample containing `_reportAppLaunchFinished`

Demo

Measuring App Launch in Time Profiler

Phases of App Launch

- Linking and loading
- UIKit initialization
- Application callbacks
- First Core Animation transaction

Phases of App Launch

- Linking and loading
- UIKit initialization
- Application callbacks
- First Core Animation transaction

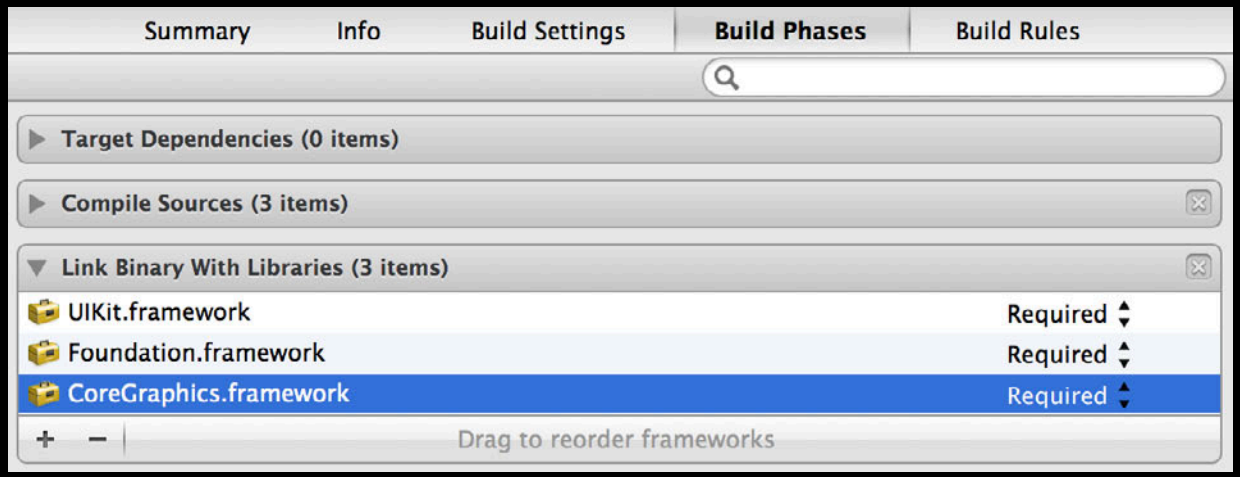
Linking and Loading

- Shows up in dyld in Time Profiler
- Libraries are mapped into address space
- Bindings are fixed up
- Static initializers are run

Linking and Loading

Minimize linked frameworks

- Each Objective-C framework adds small time and memory cost
- Avoid linking unnecessary frameworks



Linking and Loading

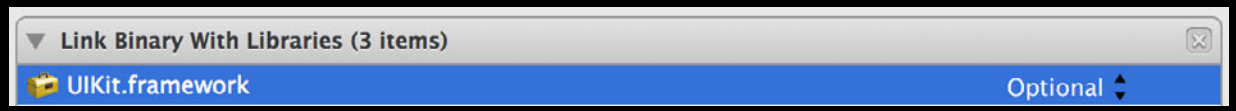
Optional frameworks

- Optional frameworks may cause linker to do extra work
- Do not mark necessary frameworks as optional

Linking and Loading

Optional frameworks

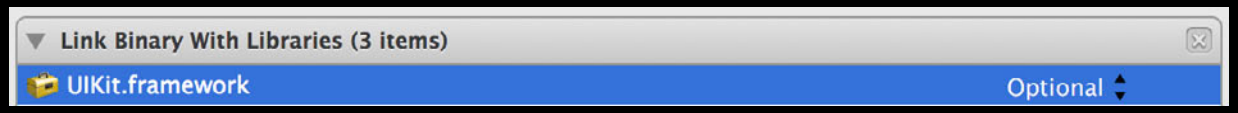
- Optional frameworks may cause linker to do extra work
- Do not mark necessary frameworks as optional



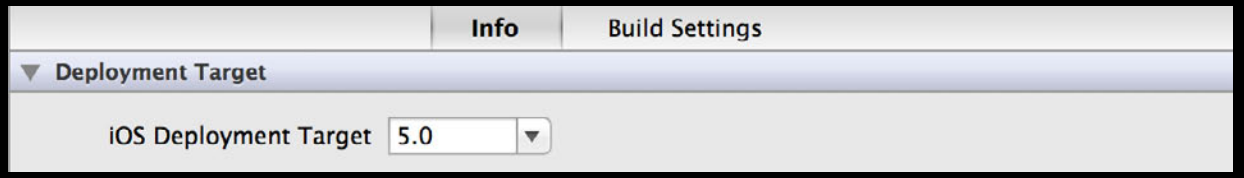
Linking and Loading

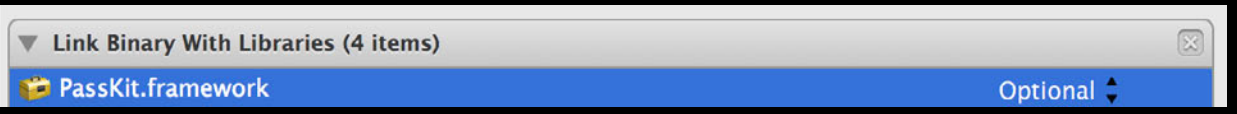
Optional frameworks

- Optional frameworks may cause linker to do extra work
- Do not mark necessary frameworks as optional



- Use optional for frameworks released after deployment target





Linking and Loading

Avoid static initializers

- Avoid creating global C++ objects

```
static std::map<int, int> GlobalMap = {{1, 2}, {3, 4}, {5, 6}};
```

- Avoid code that runs at load time

```
+ (void)load {}
__attribute__((constructor)) void DoSomeInitializationWork {}
```

- Causes extra code to always run before main
- Explicitly initialize at runtime instead
  - The `+initialize` method is okay: Runs on first use

Phases of App Launch

- Linking and loading
- UIKit initialization
- Application callbacks
- First Core Animation transaction

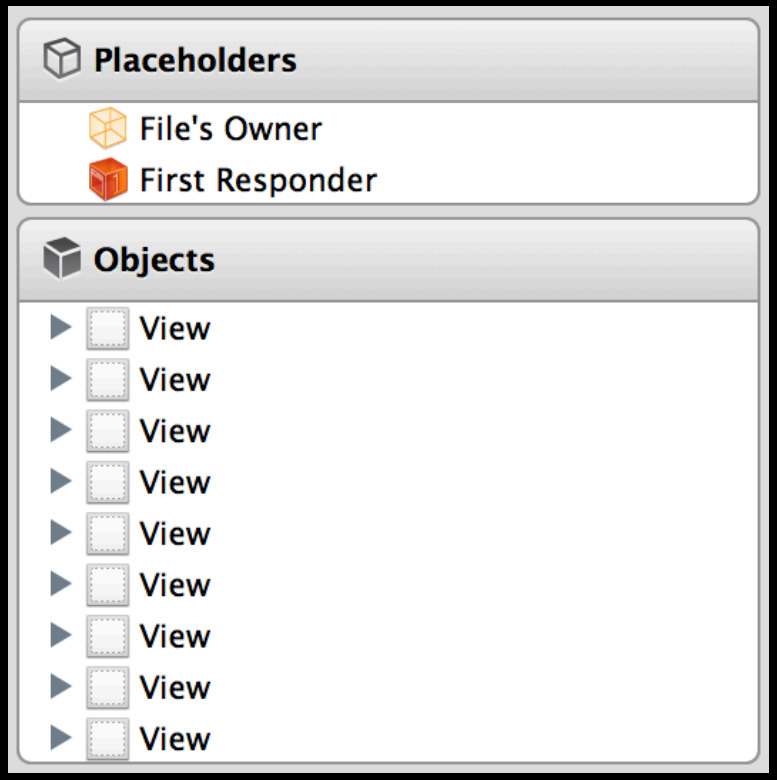
UIKit Initialization

- Fonts, status bar, user defaults, main nib initialized
- Shows up in:

```
UIApplicationInitialize
UIApplicationInstantiateSingleton
-[UIApplication _createStatusBarWithRequestedStyle: ...]
-[UIApplication _loadMainNibFileNamed:bundle:]
```

UIKit Initialization

Minimize size of main nib



UIKit Initialization

Do not store too much data in preferences

- Preferences are stored as property list files
- Property lists are deserialized all at once

```
NSUserDefaults *ud = [NSUserDefaults standardUserDefaults];
NSData *largeImage = UIImagePNGRepresentation(image);
[ud setObject:largeImage forKey:@"favoriteImage”];
```

Phases of App Launch

- Linking and loading
- UIKit initialization
- Application callbacks
- First Core Animation transaction

Application Callbacks

- UIKit calls into your code to finish launching
  - Calls `application:willFinishLaunchingWithOptions:`
  - Restores application state
  - Calls `application:didFinishLaunchingWithOptions:`
- Your app is now in control

Phases of App Launch

- Linking and loading
- UIKit initialization
- Application callbacks
- First Core Animation transaction

First Core Animation Transaction

- Shows up as time in `CA::Transaction::commit`
  - Usually happens automatically at end of run loop
  - Also happens in `-[UIApplication _reportAppLaunchFinished]` after launch
- Important phases of commit
  - Preparation: Decompressing images
  - Layout: Sizes all layers (`-layoutSubviews`)
  - Drawing: `-drawRect:`

Demo

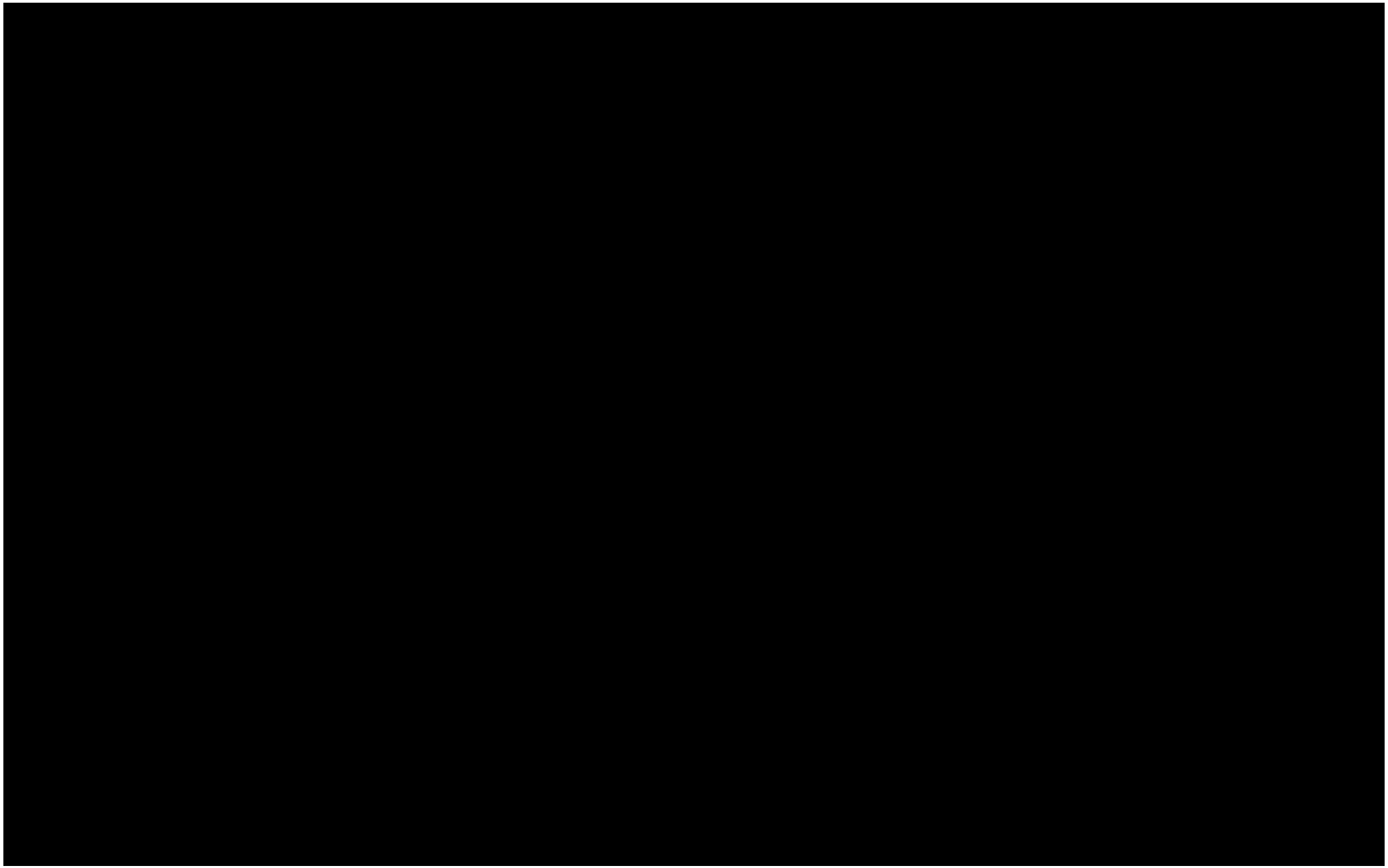
Phases of App Launch in WWDC App

App Launch

Conclusion

- Launch is the first user interaction—it should be responsive
- Measure launch time
- Profile with Time Profiler
  - Understand phases of app launch
- Observe best practices

Performance Strategies



Profile Your App

Don’t guess!

Performance Strategies

- Don’t do it
- Don’t do it again
- Do it faster
- Do it beforehand
- Do it afterwards
- Do it at scale

Performance Strategies

- Don’t do it
- Don’t do it again
- Do it faster
- Do it beforehand
- Do it afterwards
- Do it at scale

Avoid Unnecessary Work

- Profiling often reveals useless work
- Examples
  - Unnecessary shadows and masks
  - Multiple queries for the same data
  - Hundreds of milliseconds in logging at launch time

Performance Strategies

- Don’t do it
- Don’t do it again
- Do it faster
- Do it beforehand
- Do it afterwards
- Do it at scale

Reuse Instead of Recreating

- Certain classes are expensive to initialize
  - Table view cells
  - Date/number formatters
  - Regular expressions
  - SQLite statements
- Reuse the expensive-to-create object instead of recreating it

Reuse Instead of Recreating

Date formatters

```
- (UITableViewCell *)tableView:(UITableView *)view
         cellForRowAtIndexPath:(NSIndexPath *path)
{
    // dequeue or create cell...
    NSDateFormatter *formatter = [NSDateFormatter new];
    [formatter setDateFormat:@"MMMM"];
    cell.textLabel.text = [formatter stringFromDate:date];
    [formatter release];
```

February

- For commonly used date formats:
  - Cache one formatter per date format
  - Invalidate cache on `NSLocaleDidChangeNotification`
- Setting format is as expensive as recreating

Reuse Instead of Recreating

Calendars

- Calling `NSLog` makes a new calendar for each line logged
  - Avoid calling `NSLog` excessively
- Calling `+[NSCalendar currentCalendar]` returns a new instance for each call
  - Save the instance if using repeatedly

```
for (Event *event in events) {
  NSCalendar *calendar = [NSCalendar currentCalendar];
  NSDateComponents *components =
    [calendar components:NSYearCalendarUnit fromDate:date];
  [sections addEvent:event forYear:[components year]];
}
```

Reuse Instead of Recreating

SQLite statements

- Each SQLite statement is a compiled program
  - Calling `sqlite3_prepare` compiles SQL query into bytecode
- Use bind parameters and reuse prepared statements

```
NSString *format = @"SELECT * FROM Tracks WHERE id=%d";
NSString *query = [NSString stringWithFormat:format, rowid];
sqlite3_prepare_v2(db, [query UTF8String], -1, &stmt, NULL);
// use stmt
```

```
const char *query = "SELECT * FROM TRACKS WHERE id=?";
sqlite3_prepare_v2(db, query, -1, &stmt, NULL);
sqlite3_bind_int(stmt, 1);
// use stmt
```

Performance Strategies

- Don’t do it
- Don’t do it again
- Do it faster
- Do it beforehand
- Do it afterwards
- Do it at scale

Work Efficiently

- Choose the right data structure and algorithms
  - Refer to Collections Programming Topics
- Choose a faster algorithm

Work Efficiently

Data formats

- Property lists are for smaller pieces of data
  - Must deserialize entire plist to access a single object in it
  - Use binary format for plists
- Certain APIs are implemented with plists underneath
  - Preferences
  - Serialization via `NSCoding`
- Use Core Data or SQLite for storing lots of data
  - Allows for incremental loading

Work Efficiently

Optimize database queries

- Find slow queries with `sqlite3_trace` and `sqlite3_profile`

```
static void profile(void *context, const char *sql, sqlite3_uint64 ns) {
    syslog(LOG_WARNING, "Query: %s\n", sql);
    syslog(LOG_WARNING, "Execution Time: %llu ms\n”, ns / 1000000);
}

sqlite3_profile(conn, &profile, NULL);
```

- Understand problematic queries with EXPLAIN QUERY PLAN

```
sqlite3> EXPLAIN QUERY PLAN
    ...> SELECT * FROM Track WHERE AlbumID=2 ORDER BY AlbumOrder;

TABLE Track WITH INDEX TrackAlbumIDOrderIndex ORDER BY
```

Performance Strategies

- Don’t do it
- Don’t do it again
- Do it faster
- Do it beforehand
- Do it afterwards
- Do it at scale

Precompute Results

- Results of expensive calculations can be precomputed
- Example: Recurring events
  - Recurring events can take a long time to expand
  - Meeting on first Monday, Wednesday, and Friday of every month except February
  - Solution
    - Pre-expand recurrences into occurrences
    - Store occurrences in database

Precompute Results

Beware of memory growth

- Precomputing and caching certain objects has large memory impact
- Caching images is especially problematic
  - Backing bitmap persists in memory for lifetime of object

```
static UIImage *ScreenSizedImage = nil;

if (!ScreenSizedImage) {
  ScreenSizedImage = [UIImage imageNamed:@"wallpaper.png"];
}
```

Performance Strategies

- Don’t do it
- Don’t do it again
- Do it faster
- Do it beforehand
- Do it afterwards
- Do it at scale

Asynchronous Loading

- Showing data synchronously is a better user experience
- If not possible, use GCD or other APIs to postpone work

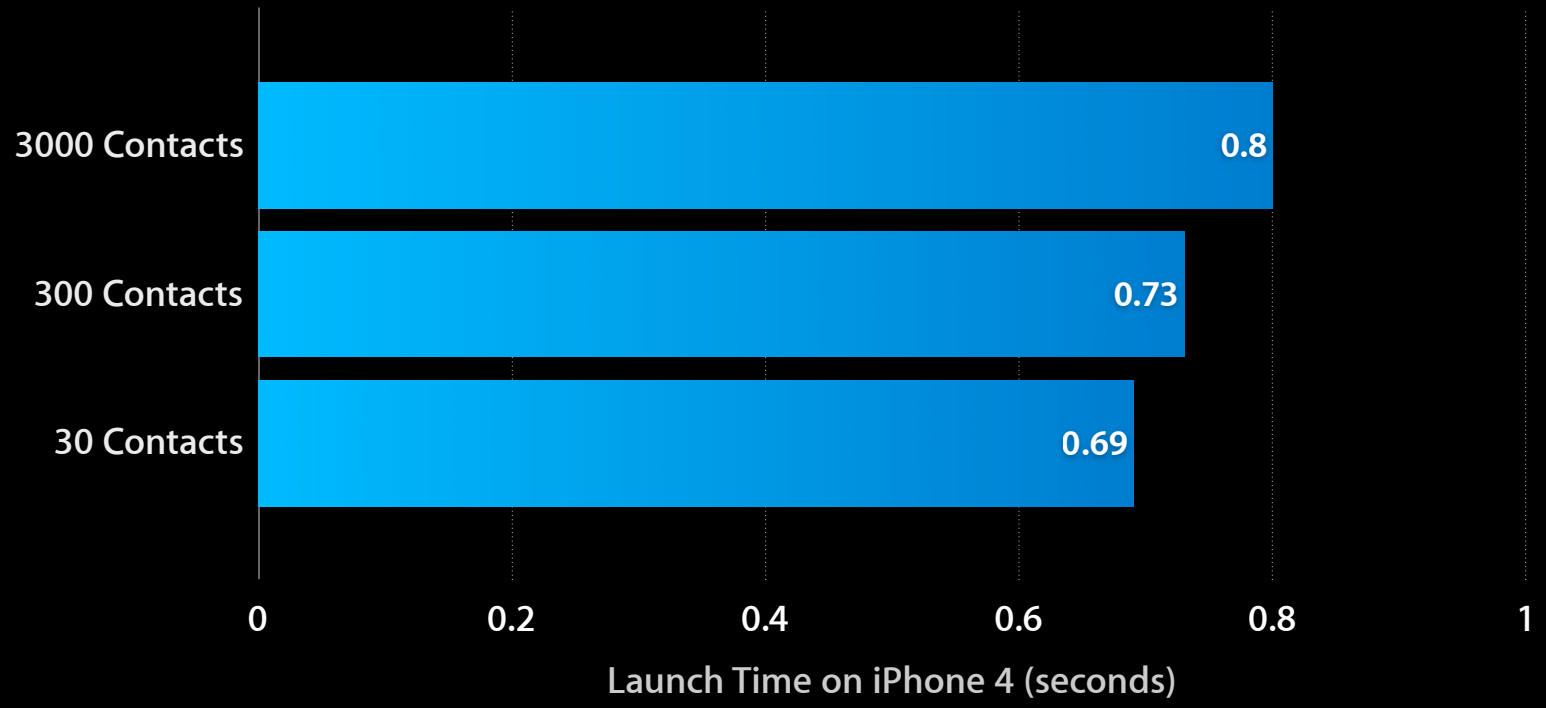
- Example: Calendar
  - Launches to a responsive interface with no events
  - Events are loaded asynchronously

Performance Strategies

- Don’t do it
- Don’t do it again
- Do it faster
- Do it beforehand
- Do it afterwards
- Do it at scale

Scale to Large Data Sets

Contacts Launch Time



Scale to Large Data Sets

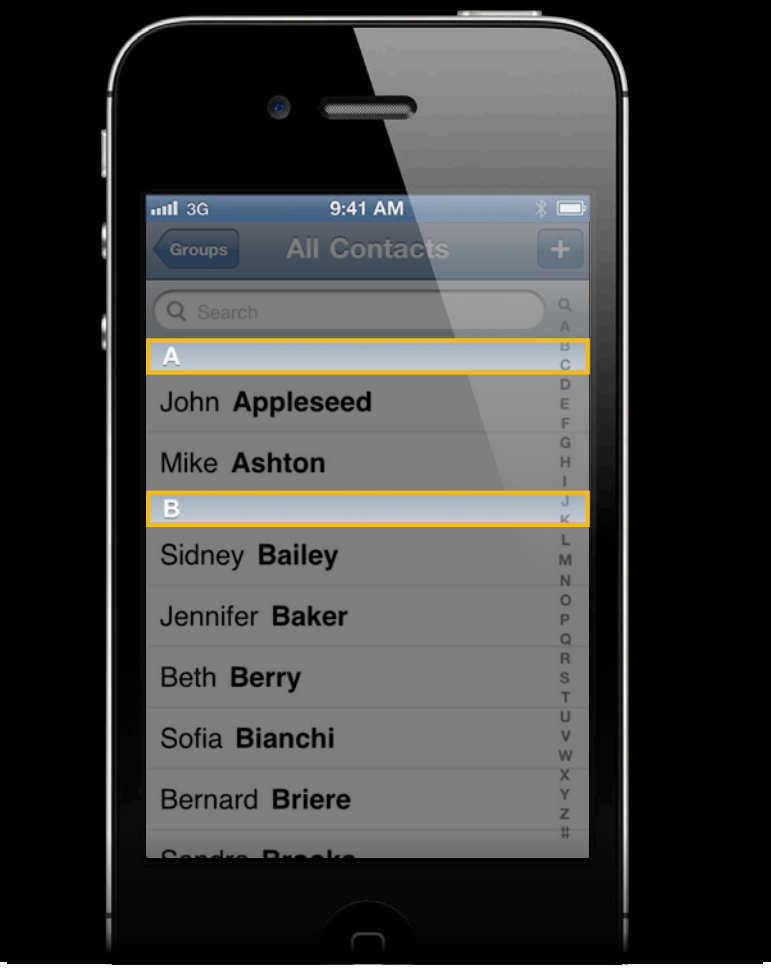
Make critical methods fast

Scale to Large Data Sets

Make critical methods fast

- Loading sections

```
-numberOfSectionsInTableView:
-tableView:titleForHeaderInSection:
-tableView:numberOfRowsInSection:
```



Scale to Large Data Sets

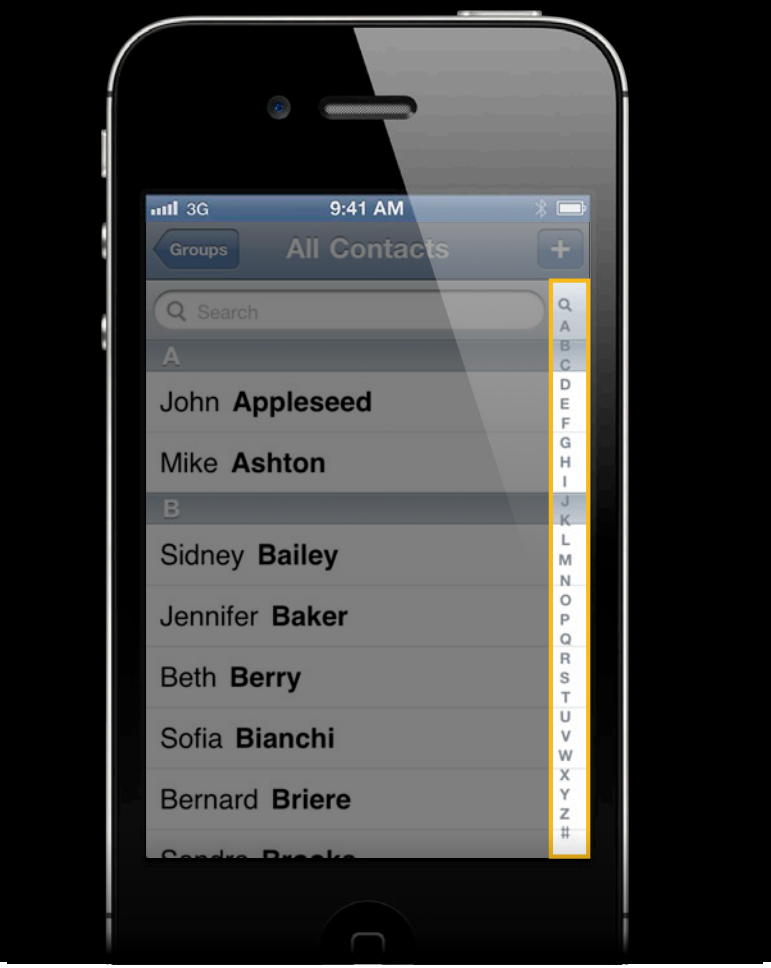
Make critical methods fast

- Loading sections

```
-numberOfSectionsInTableView:
-tableView:titleForHeaderInSection:
-tableView:numberOfRowsInSection:
```

- Loading the index bar

```
-tableView:sectionIndexTitlesForTableView
```



Scale to Large Data Sets

Make critical methods fast

- Loading sections

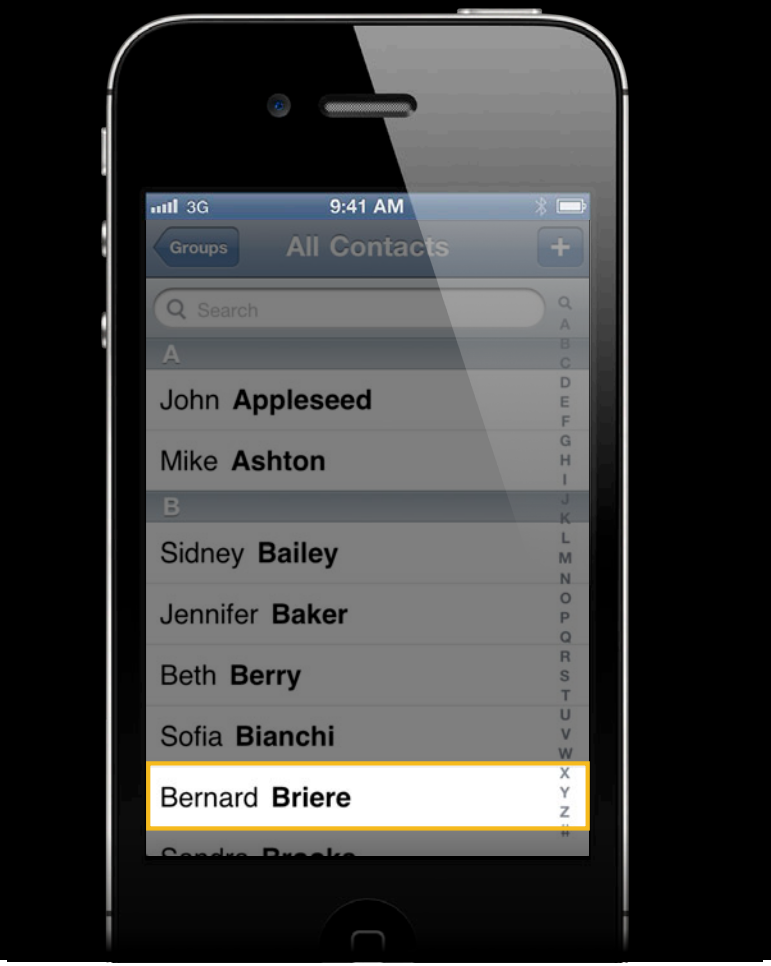
```
-numberOfSectionsInTableView:
-tableView:titleForHeaderInSection:
-tableView:numberOfRowsInSection:
```

- Loading the index bar

```
-tableView:sectionIndexTitlesForTableView
```

- Loading visible cells

```
-tableView:cellForRowAtIndexPath:
```



Scale to Large Data Sets

Loading section information

- UITableView requires section counts and titles up front
  - Slow: load entire data set and group into sections
  - Faster: store section counts separately
- CoreData users get this for free

```
-[NSFetchedResultsController initWithFetchRequest:(NSFetchRequest *)fetchRequest
                             managedObjectContext:(NSManagedObjectContext *)context
                               sectionNameKeyPath:(NSString *)sectionNameKeyPath
                                        cacheName:(NSString *)name]
```

Performance Strategies

Conclusion

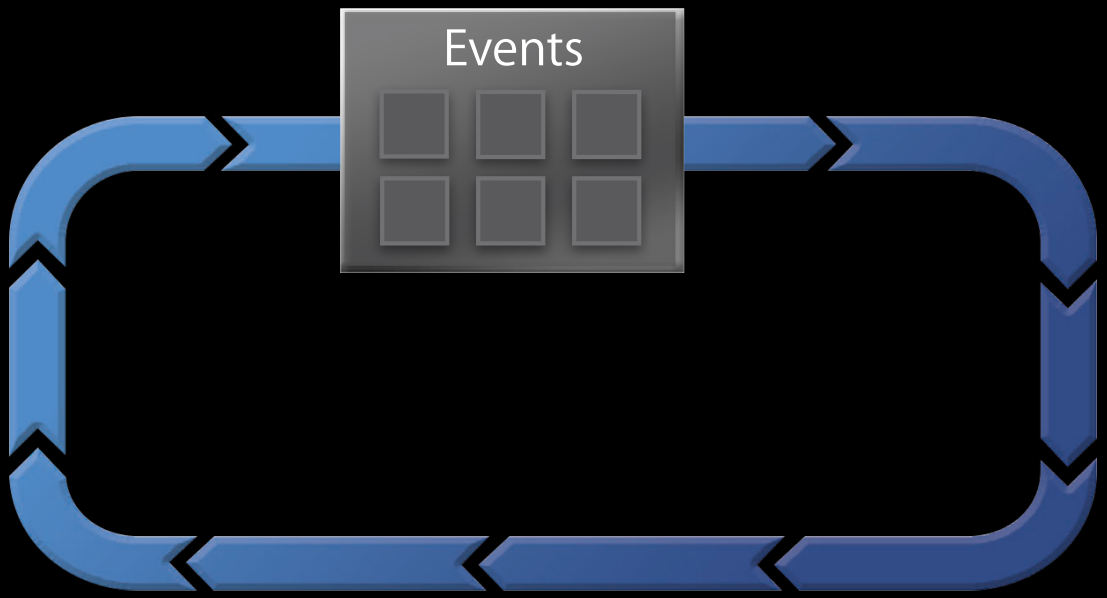
- Profile your app
- Avoid unnecessary work
- Test with large data sets

Event Handling

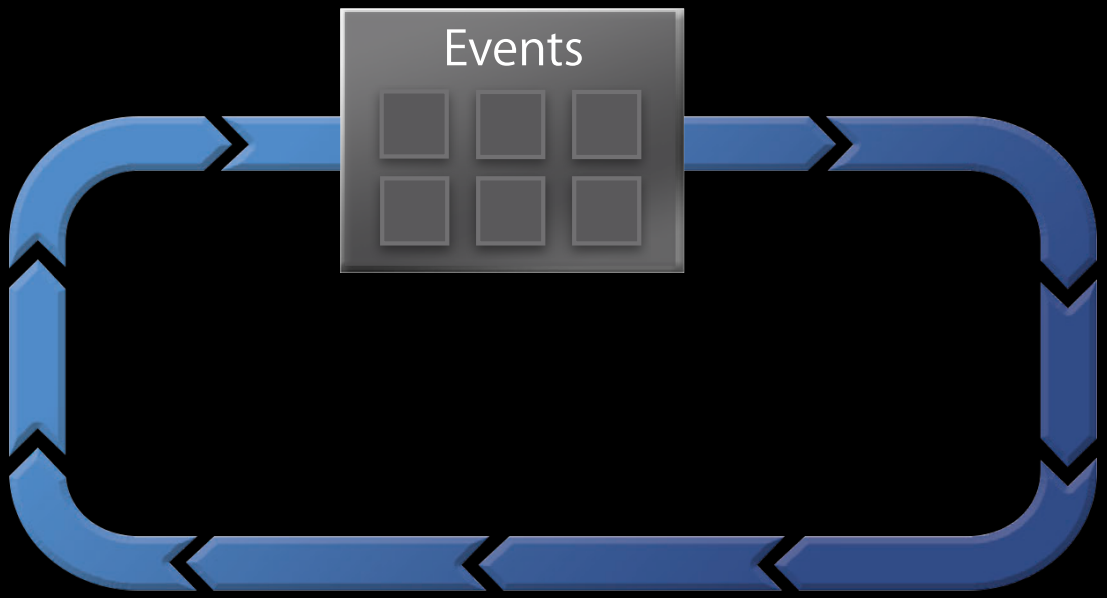
Processing User Events

- User events are processed on main thread's run loop
  - Touch
  - Scrolling
  - Accelerometer
  - Proximity sensor
- Keep main run thread free to process events

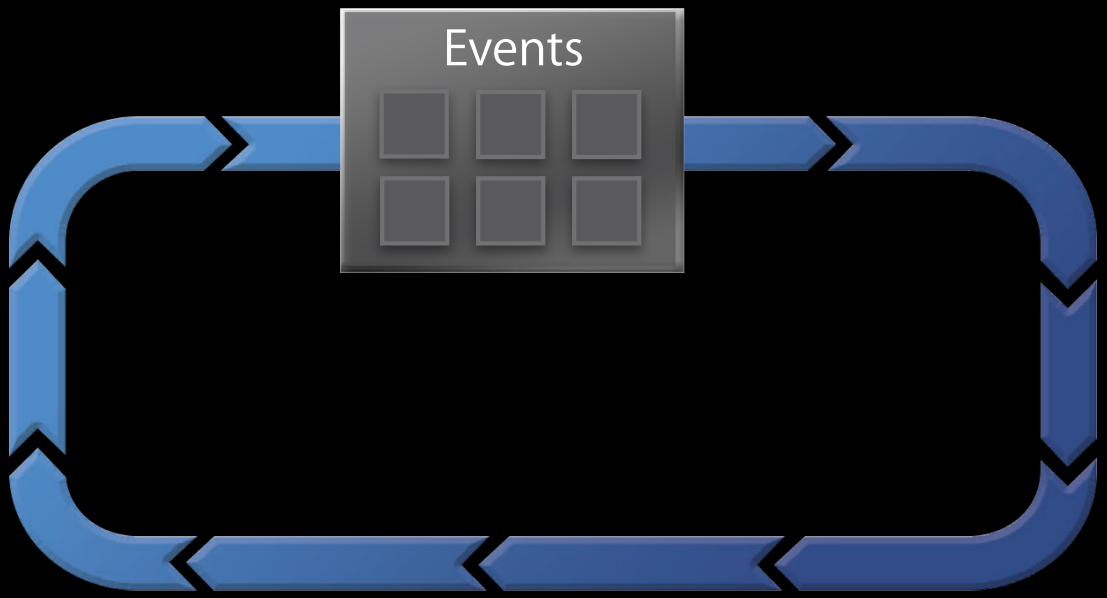
Main Run Loop



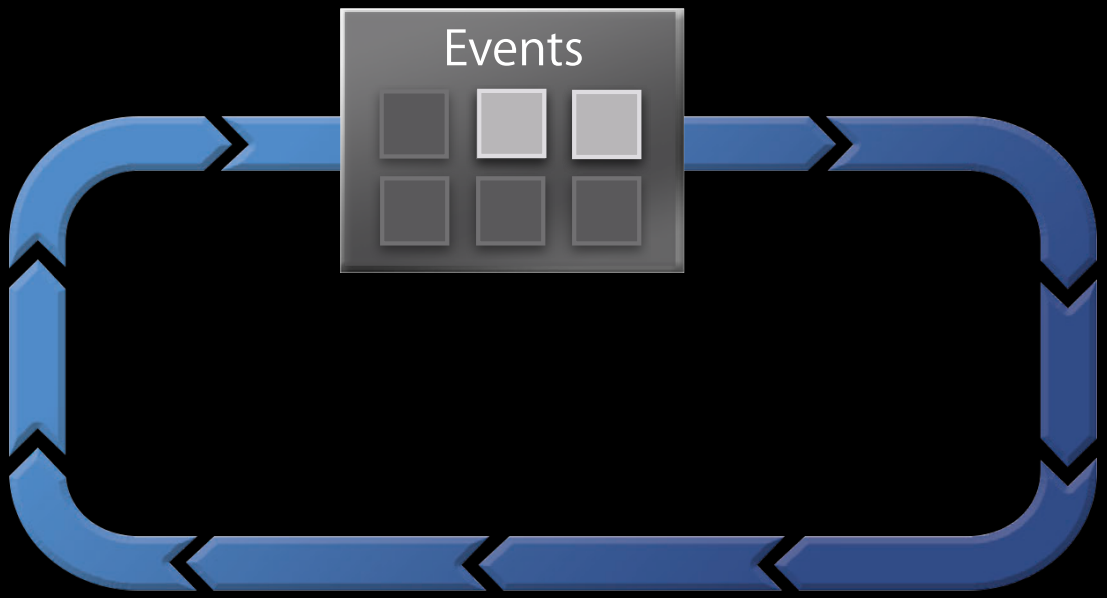
Main Run Loop



Main Run Loop



Main Run Loop



Optimizing Event Handling

- Minimize CPU time in main thread
- Move work off the main thread
- Don’t block the main thread

Optimizing Event Handling

- Minimize CPU time in main thread
- Move work off the main thread
- Don’t block the main thread

Minimizing CPU Work

- Performance strategies also apply to event handling
- Use Time Profiler to measure hotspots

Demo

Switching tabs in the WWDC App

Optimizing Event Handling

- Minimize CPU time in main thread
- Move work off the main thread
- Don’t block the main thread

Moving Work Off the Main Thread

Two categories

- Implicit concurrency
- Explicit concurrency

Moving Work Off the Main Thread

Implicit concurrency

- View and layer animations
- Layer compositing
- PNG decoding
- Important: Scrolling is not an animation!

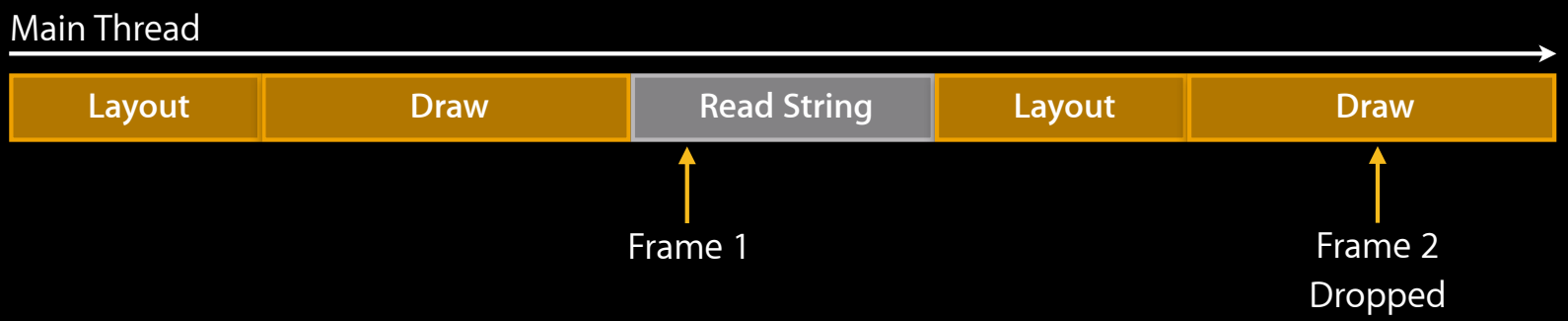
Moving Work Off the Main Thread

Explicit concurrency

- Grand Central Dispatch
- NSOperationQueue
- NSThread

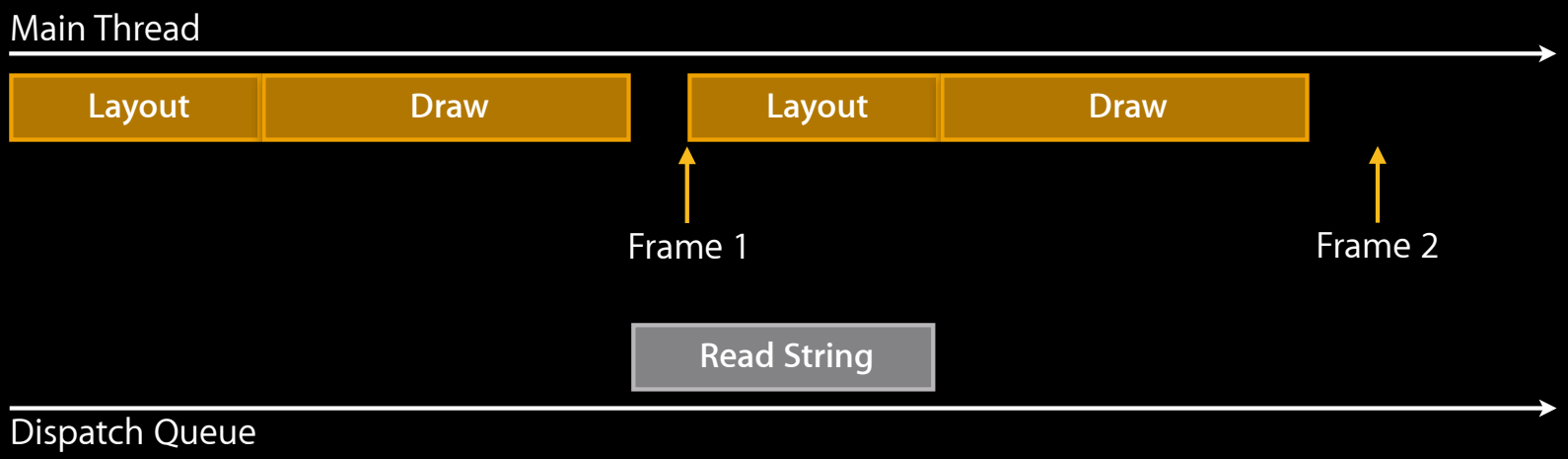
Grand Central Dispatch

Reading a file off the main thread



Grand Central Dispatch

Reading a file off the main thread



Grand Central Dispatch

Reading a file off the main thread

```
    NSError *err = nil;
    NSStringEncoding encoding;
    NSString *myText = [NSString stringWithContentsOfFile:myFile
usedEncoding:&encoding error:&err];
    if (err == nil) {
         [myTextField setText:myText];
    }
```

Grand Central Dispatch

Reading a file off the main thread

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0),
^{
    NSError *err = nil;
    NSStringEncoding encoding;
    NSString *myText = [NSString stringWithContentsOfFile:myFile
 usedEncoding:&encoding error:&err];
    if (err == nil) {
       dispatch_async(dispatch_get_main_queue(), ^{
           [myTextField setText:myText];
       });
    }
});
```

GCD Gotchas

Too many threads

- It's possible for GCD to make too many threads for you
- Having too many threads adds overhead
- There's also a hard limit

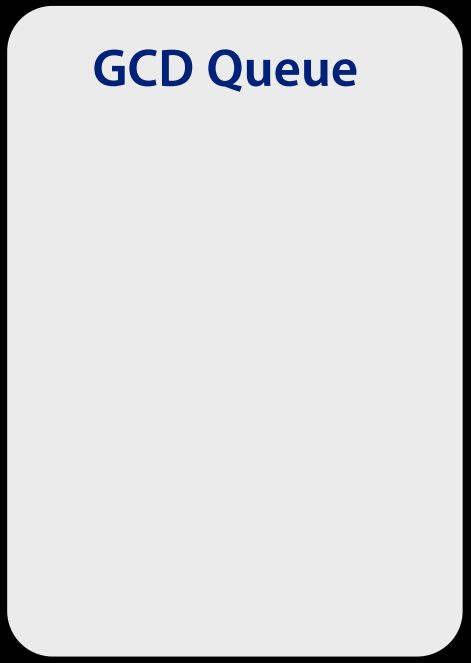
GCD Gotchas

Too many threads

GCD Gotchas

Too many threads

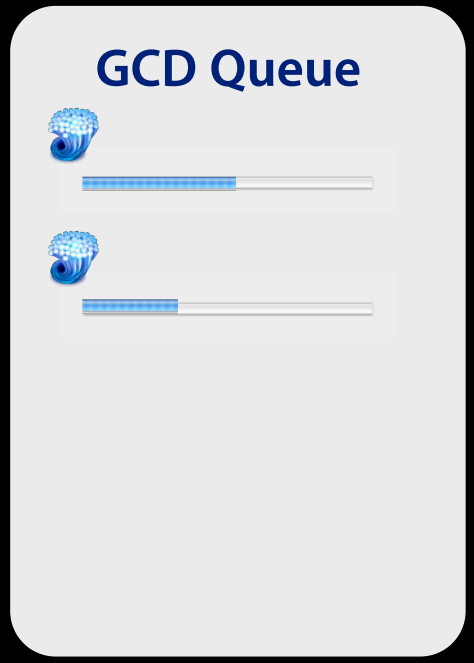
- Concurrent queue—ok



GCD Gotchas

Too many threads

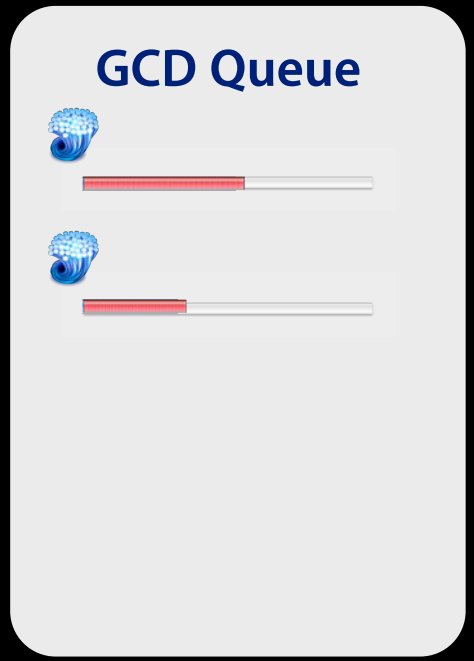
- Concurrent queue—ok
- Add some blocks—ok



GCD Gotchas

Too many threads

- Concurrent queue—ok
- Add some blocks—ok
- The blocks make long blocking calls—bad!



GCD Gotchas

Too many threads

- Concurrent queue—ok
- Add some blocks—ok
- The blocks make long blocking calls—bad!

GCD Gotchas

Too many threads

- Concurrent queue—ok
- Add some blocks—ok
- The blocks make long blocking calls—bad!

```
dispatch_queue_t queue =
  dispatch_get_global_queue(0, 0);

for (NSURLRequest *req in requests) {
  dispatch_async(queue, ^{
    NSData *data = sendSyncURLReq(req);
    processData(data);
  });
}
```

GCD Gotchas

Too many threads

- Solutions
  - Serial queue
  - Dispatch sources
  - NSOperationQueue with limit
  - NSURLConnection async methods

GCD Gotchas

Thread safety

GCD Gotchas

Thread safety

- Main thread only—UIKit
  - Exceptions: UIGraphics, UIBezierPath, UIImage

GCD Gotchas

Thread safety

- Main thread only—UIKit
  - Exceptions: UIGraphics, UIBezierPath, UIImage
- Any thread (with synchronization)—Most of CG, CA, Foundation
  - Can't access from two threads simultaneously

GCD Gotchas

Thread safety

- Main thread only—UIKit
  - Exceptions: UIGraphics, UIBezierPath, UIImage
- Any thread (with synchronization)—Most of CG, CA, Foundation
  - Can't access from two threads simultaneously
- Thread-safe—Objective-C introspection
  - Coarse locks in thread-safe frameworks can lead to contention
  - Use System Trace to detect contention

GCD Gotchas

Background queues

- iOS 4.3 added `DISPATCH_QUEUE_PRIORITY_BACKGROUND`
- Background is extremely low priority
  - I/O is throttled
  - May not run for seconds
    - What happens if bg queue holds lock that main thread needs?
- Only use for truly background operations
  - Consider using `DISPATCH_QUEUE_PRIORITY_LOW` instead

Optimizing Event Handling

- Minimize CPU time in main thread
- Move work off the main thread
- Don’t block the main thread

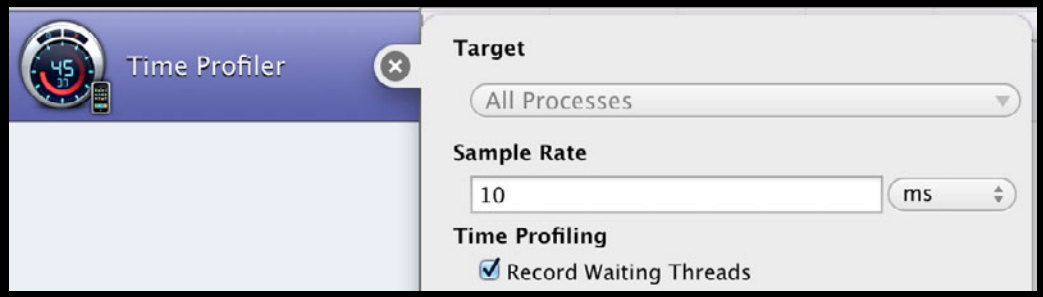
Don’t Block the Main Thread

- Main thread may be unresponsive even if it uses little CPU
- Main thread may block for:
  - Disk
  - Network
  - Locks or `dispatch_sync`
  - Sending messages to other processes or threads
- How do you detect these issues?
  - Regular Time Profile only detects CPU usage issues

Don’t Block the Main Thread

Profiling with Time Profiler

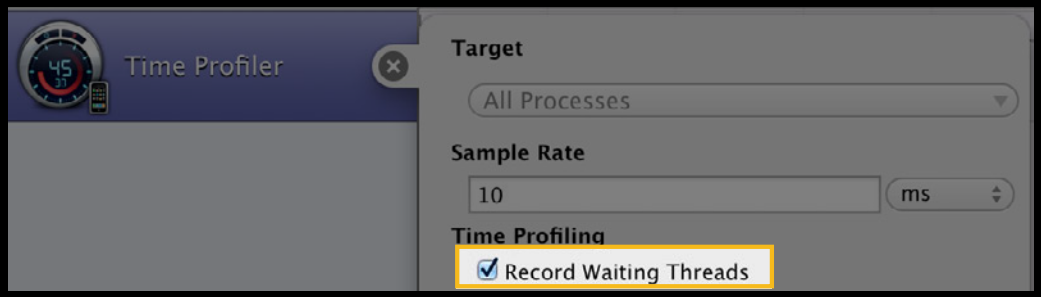
- Good: Use regular Time Profile
  - Switch to CPU strategy view
  - Highlight main thread
- Better: Use “Record Waiting Threads” in Time Profile



Don’t Block the Main Thread

Profiling with Time Profiler

- Good: Use regular Time Profile
  - Switch to CPU strategy view
  - Highlight main thread
- Better: Use “Record Waiting Threads” in Time Profile



Don’t Block the Main Thread

Profiling with System Trace

- Most blocking events are associated with a system call
- Common blocking syscalls
  - Reading/writing a file: `read/write`
  - Sending/receiving network data: `send/recv`
  - Acquiring lock: `psynch_mutex_wait`
  - IPC: `mach_msg`
- System Trace records all system calls
  - Also time spent waiting on each system call

Demo

Finding blocking calls with System Trace

Summary

- Profile your application
- Understand app launch
- Don’t block the main thread

More Information

Michael Jurewitz
Developer Tools and Performance Evangelist
jurewitz@apple.com

Documentation
iOS App Programming Guide
http://developer.apple.com/library/ios/#DOCUMENTATION/iPhone/Conceptual/iPhoneOSProgrammingGuide/Introduction/Introduction.html

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

Related Sessions

iOS App Performance: Graphics and Animations	Presidio Thursday 3:15PM
iOS App Performance: Memory	Presidio Thursday 4:30PM
Learning Instruments	Presidio Wednesday 4:30PM
Core Data Best Practices	Mission Wednesday 9:00AM
Building Concurrent User Interfaces on iOS	Pacific Heights Wednesday 9:00AM

Labs

OS X Performance Lab	Developer Tools Lab A Friday 9:00AM
Xcode Lab	Developer Tools Lab B Friday 9:00AM


WWDC2012