Building Responsive and Efficient Apps with GCD

Session 718

Anthony J. Chivetta  Darwin Runtime Engineer
Daniel A. Steffen  Darwin Runtime Engineer
Quality of Service Introduction

GCD Design Patterns with QoS

Threads, Queues, and Run Loops

GCD and Crash Reports
Handling Events
Handling Events

Main Thread
Handling Events

Main Thread
- NSRunLoop
- UIKit
Handling Events

Main Thread

- NSRunLoop
- UIKit

Wait for Events
Handling Events

Main Thread

- NSRunLoop
- UIKit
- Wait for Events

Delegate Method Callout

UIApp Delegate

MyAwesomeApp
Handling Events

Main Thread
- NSRunLoop
- UIKit

Wait for Events
- Delegate Method Callout

UIApp Delegate
- Read from Database

MyAwesomeApp
Handling Events

Delegation:
- MyAwesomeApp

Main Thread:
- NSRunLoop
- UIKit

Delegate Method Callout:
- Update UI

UIKit Delegate:
- Read from Database

Flow:
- Wait for Events
- Delegate Method Callout
- Update UI
Handling Events

Main Thread

- NSRunLoop
- UIKit

Delegate Method Callout

UIApp Delegate

Wait for Events

Update UI

Read from Database

UIKit

MyAwesomeApp
Handling Events

Main Thread

- NSRunLoop
- UIKit

Wait for Events

Delegate Method Callout

UIApp Delegate

Update UI

Read from Database

MyAwesomeApp
Handling Events Asynchronously

Main Thread
- NSRunLoop
- UIKit
- Wait for Events
  - Delegate Method Callout
  - UIApp Delegate
- Wait for Events
Handling Events Asynchronously

Main Thread

- NSRunLoop
- UIKit

Wait for Events

Delegate Method Callout

UIKit Delegate

GCD Queue

Wait for Events
Handling Events Asynchronously

- **NSRunLoop**
- **UIKit**
  - **Wait for Events**
  - **Delegate Method Callout**

- **UIApp Delegate**
  - **dispatch_async()**

- **GCD Queue**
  - **^{…}**

- MyAwesomeApp
Handling Events Asynchronously

dispatch_async()

Update UI

UIKit

Wait for Events

NSRunLoop

UIKit

Wait for Events

Delegate Method Callout

UIApplication Delegate

dispatch_async()

Update UI

Wait for Events

GCD Queue

dispatch_async()
Handling Events Asynchronously

```swift
dispatch_async()
```

UIKit

Wait for Events

NSRunLoop

GCD Queue

dispatch_async()

Delegate Method Callout

UITApp Delegate

^{} ...

Wait for Events

Update UI

^{} ...

^{} ...

Wait for Events
Competing Threads

MyAwesomeApp
Quality of Service Classes

UI: User Interactive
IN: User Initiated
UT: Utility
BG: Background
Quality of Service Classes

Complex resource controls

CPU scheduling priority
I/O priority
Timer coalescing
CPU throughput vs. efficiency
More…
Quality of Service Classes

Complex resource controls

- CPU scheduling priority
- I/O priority
- Timer coalescing
- CPU throughput vs. efficiency
- More…

Configuration values tuned for each platform/device
Single abstract parameter
Communicate developer intent
Explicit classification of work
• Move away from dictating specific configuration values
# Quality of Service Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Interactive</td>
<td>Main thread, animations</td>
</tr>
<tr>
<td>User Initiated</td>
<td>Immediate results</td>
</tr>
<tr>
<td>Utility</td>
<td>Long-running tasks</td>
</tr>
<tr>
<td>Background</td>
<td>Not user visible</td>
</tr>
<tr>
<td>Quality of Service Classes</td>
<td></td>
</tr>
<tr>
<td>----------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>User Interactive</strong></td>
<td>Is this work actively involved in updating the UI?</td>
</tr>
<tr>
<td><strong>User Initiated</strong></td>
<td>Immediate results</td>
</tr>
<tr>
<td><strong>Utility</strong></td>
<td>Long-running tasks</td>
</tr>
<tr>
<td><strong>Background</strong></td>
<td>Not user visible</td>
</tr>
</tbody>
</table>
# Quality of Service Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Interactive</td>
<td>Is this work actively involved in updating the UI?</td>
</tr>
<tr>
<td>User Initiated</td>
<td>Is this work required to continue user interaction?</td>
</tr>
<tr>
<td>Utility</td>
<td>Long-running tasks</td>
</tr>
<tr>
<td>Background</td>
<td>Not user visible</td>
</tr>
</tbody>
</table>
# Quality of Service Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Interactive</td>
<td>Is this work actively involved in updating the UI?</td>
</tr>
<tr>
<td>User Initiated</td>
<td>Is this work required to continue user interaction?</td>
</tr>
<tr>
<td>Utility</td>
<td>Is the user aware of the progress of this work?</td>
</tr>
<tr>
<td>Background</td>
<td>Not user visible</td>
</tr>
</tbody>
</table>
## Quality of Service Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Interactive</td>
<td>Is this work actively involved in updating the UI?</td>
</tr>
<tr>
<td>User Initiated</td>
<td>Is this work required to continue user interaction?</td>
</tr>
<tr>
<td>Utility</td>
<td>Is the user aware of the progress of this work?</td>
</tr>
<tr>
<td>Background</td>
<td>Is the user unaware of this work?</td>
</tr>
</tbody>
</table>
GCD Design Patterns with QoS

Daniel A. Steffen  Darwin Runtime Engineer
GCD and QoS

Fundamentals

QoS can be specified on Blocks and on queues

dispatch_async() automatically propagates QoS

Some priority inversions are resolved automatically
Asynchronous Work

Main Thread

- NSRunLoop
- UIKit

Wait for Events

Delegate Method Callout

GCD Queue

dispatch_async()

^{}

UIApp Delegate

^{}

UIKit

Wait for Events

Wait for Events

Update UI

dispatch_async()
Asynchronous Work

Main Thread
- NSRunLoop
- UIKit

Wait for Events
- Delegate Method Callout
- Update UI

UIKit

GCD Queue
- dispatch_async()

MyAwesomeApp

^{…}
Asynchronous Work

Main Thread
- NSRunLoop
- UIKit

Wait for Events

Delegate Method Callout

UIKit

Wait for Events

UIKit Delegate

GCD Queue

dispatch_async()

^{...}

dispatch_async()

^ {...}

dispatch_async()
Asynchronous Work

Main Thread
- NSRunLoop
- UIKit
  - Wait for Events
  - Delegate Method Callout
  - UIKit
    - Wait for Events
    - Update UI
    - ^{…}
    - UIKit
      - Wait for Events
      - Delegate Method Callout
      - Update UI
      - ^{…}

GCD Queue
- MyAwesomeApp
  - dispatch_async()
  - ^{…}
  - dispatch_async()
  - ^{…}
  - dispatch_async()
  - ^{…}
  - dispatch_async()
Asynchronous Work

Main Thread
- NSRunLoop
- UIKit
  - Wait for Events
  - Delegate Method Callout
  - Update UI

UIT App Delegate
- Dispatch async()
- Update UI

GCD Queue
- Dispatch async()

MyAwesomeApp
Asynchronous Work

Main Thread
- NSRunLoop
- UIKit
  - Wait for Events
  - Delegate Method Callout
  - Wait for Events
  - UIKit
    - Wait for Events

GCD Queue
- Delegate Method Callout
- dispatch_async()
- UIApp Delegate
- ^{...}
- dispatch_async()
- ^{...}
- dispatch_async()

MyAwesomeApp
Asynchronous Work

Main Thread
- NSRunLoop
- UIKit

Wait for Events

Delegate Method Callout

UIKit

GCD Queue

^{}

dispatch_async()

Wait for Events

Update UI

UIApp Delegate

^{}

dispatch_async()

dispatch_async()
Asynchronous Work

Main Thread
- NSRunLoop
- UIKit
  - Wait for Events
  - Delegate Method Callout
  - Update UI
  - Wait for Events

GCD Queue
- dispatch_async()

UIKit
- dispatch_async()

MyAwesomeApp

^{…}

^{…}
Asynchronous Work

Main Thread
- NSRunLoop
- UIKit

Wait for Events

Delegate Method Callout

UIApp Delegate

GCD Queue
- dispatch_async()

Update UI

UIKit

Wait for Events

dispatch_async()
Asynchronous Work

Main Thread
- NSRunLoop
- UIKit

Wait for Events

Delegate Method Callout

UIKit Delegate

GCD Queue

dispatch_async()

Update UI

dispatch_async()
Asynchronous Work

Main Thread
- NSRunLoop
- UIKit
  - Wait for Events
  - Delegate Method Callout
  - Update UI

UI
- UIApp Delegate
  - dispatch_async()

GCD Queue
- dispatch_async()

MyAwesomeApp
- IN

…”
QoS Propagation
Inferred QoS
QoS Propagation

Inferred QoS

QoS captured at the time of Block submission

- User Interactive translated to User Initiated
QoS Propagation

Inferred QoS

QoS captured at the time of Block submission

- User Interactive translated to User Initiated

Used if destination does not have QoS specified

- Does not lower QoS
Long-Running Job

Start Calculation

dispatch_async(q, ^{...})

Main Thread

NSRunLoop

UIKit

UIApplication

Delegates

GCD Queue

^{...}
Long-Running Job

MyAwesomeApp

dispatch_async(dispatch_get_main_queue(), ^{
// Update Progress UI
})

GCD Queue

dispatch_async(dispatch_get_main_queue(), ^{
// Update Progress UI
})

Start Calculation

dispatch_async(q, ^{...
})

Progress Updates
Long-Running Job

MyAwesomeApp

Main Thread
- NSRunLoop
- UIKit
- UIApp Delegates

Start Calculation

Progress Updates

GCD Queue

^{...}
Long-Running Job

Start Calculation

Progress Updates

MyAwesomeApp

NSRunLoop

UIKit

UIApp Delegates

GCD Queue

^{...}
Long-Running Job

Start Calculation

Progress Updates

NSRunLoop
UIKit
UIApp Delegates

GCD Queue

MyAwesomeApp
Long-Running Job

Start Calculation

Progress Updates

Main Thread
- NSRunLoop
- UIKit
- UIApp
- Delegates

GCD Queue

MyAwesomeApp

UT
Long-Running Job

Start Calculation

UI

UT

GCD Queue

Main Thread
- NSRunLoop
- UIKit
- UIApp Delegates

MyAwesomeApp

^{...}
Long-Running Job

Start Calculation

\[
b = \text{dispatch\_block\_create\_with\_qos\_class}(\text{0, QOS\_CLASS\_UTILITY, 0, }^\{\ldots\})
\]

\[
\text{dispatch\_async(q, b)}
\]
Long-Running Job

Start Calculation

\[
b = \text{dispatch\_block\_create\_with\_qos\_class}(0, \text{QOS\_CLASS\_UTILITY}, 0, ^{\ldots})
\]

\[
dispatch\_async(q, b)
\]
Block QoS

Block created with explicit QoS attribute

- When work of another class is generated
Block QoS

Block created with explicit QoS attribute
- When work of another class is generated

Captured at Block object creation
- DISPATCH_BLOCK_ASSIGN_CURRENT
- Store a callback Block for later submission
Maintenance Task

Main Thread

- NSRunLoop
- UIKit
- Wait for Events

Delegate Method Callout

- UIApp Delegate

Wait for Events
Maintenance Task

Main Thread

NSRunLoop
UIKit

Wait for Events
Delegate Method Callout

UIApp Delegate

Wait for Events

MyAwesomeApp
Maintenance Task

Main Thread

UI

UIKit

NSRunLoop

Wait for Events

Delegate Method Callout

UIApp Delegate

dispatch_async()

GCD Queue

^{...}

Wait for Events

MyAwesomeApp
Maintenance Task

Main Thread
- NSRunLoop
- UIKit
- Wait for Events
- Delegate Method Callout
- UI Delegate

GCD Queue
- dispatch_async()
- ^{...}

MyAwesomeApp
- BG
Main Thread

```python
qos_attr =
    dispatch_queue_attr_make_with_qos_class(
        attr, QOS_CLASS_BACKGROUND, 0)
q = dispatch_queue_create("cleanup", qos_attr)
^{
```

Wait for Events

Delegate Method Callout

dispatch_async()
Maintenance Task

Main Thread

- NSRunLoop
- UIKit

UI

Delegate Method Callout

GCD Queue

dispatch_async()

UIApp Delegate

Wait for Events

Wait for Events

MyAwesomeApp

BG

^ {...}
Maintenance Task

Main Thread
- NSRunLoop
- UIKit
  - Wait for Events
  - Delegate Method Callout

UI
- UI Delegate
  - dispatch_async()

GCD Queue
- Q

BG

MyAwesomeApp
- ^{...}
Maintenance Task

Main Thread
- NSRunLoop
- UIKit

UI

Delegate Method Callout

UIDelegate

dispatch_async()

GCD Queue

BG

Wait for Events

MyAwesomeApp
**Maintenance Task**

**Main Thread**
- NSRunLoop
- UIKit

Wait for Events

Delegate Method Callout

```
dispatch_block_create(
    DISPATCH_BLOCK_DETACHED, ^{…})
```

dispatch_async()

**GCD Queue**

Wait for Events

**MyAwesomeApp**

^…^
Maintenance Task

Main Thread
- NSRunLoop
- UIKit
- Wait for Events

UI

GCD Queue

MyAwesomeApp

BG^{...}
Maintenance Task

Main Thread
- NSRunLoop
- UIKit
- Wait for Events

Logout

dispatch_async()
Maintenance Task

Main Thread
- NSRunLoop
- UIKit
- Wait for Events

Logout

dispatch_async()

GCD Queue

MyAwesomeApp

BG

Logout

^{...}
Maintenance Task

Main Thread
- NSRunLoop
- UIKit
- Wait for Events

UI

GCD Queue

dispatch_block_create(
    DISPATCH_BLOCK_ENFORCE_QOS_CLASS, ^{...})

Logout

dispatch_async()
Maintenance Task

MyAwesomeApp

Main Thread

NSRunLoop

UIKit

Wait for Events

Logout

dispatch_async()
Asynchronous Priority Inversion

High QoS Block submitted to serial queue

- Queue already contains Blocks with lower QoS
Asynchronous Priority Inversion

High QoS Block submitted to serial queue
• Queue already contains Blocks with lower QoS
QoS is raised until high QoS Block is reached
• Invisible to Blocks themselves
Queue QoS

Recap

Queues that are single purpose

• Detached Blocks may also be appropriate
Queue QoS

Recap

Queues that are single purpose
- Detached Blocks may also be appropriate

Ignore QoS in asynchronous Blocks
- Exceptional cases can enforce Block QoS
Queues as Locks

Main Thread
- NSRunLoop
- UIKit
- UIApp
- Delegates

Locked Access to Data Structure

Data Structure
Queues as Locks

Main Thread
- NSRunLoop
- UIKit
- UIApp
- Delegates

Locked Access to Data Structure

Data Structure

```
dispatch_queue_create("com.example.data", DISPATCH_QUEUE_SERIAL)
```
Queues as Locks

Locked Access to Data Structure

Main Thread
- NSRunLoop
- UIKit
- UIAppDelegate
  - Delegates

Data Structure

dispatch_sync(data.q, ^{...})
Queues as Locks

Locked Access to Data Structure

dispatch_sync(data.q, ^{...})
Queues as Locks

MyAwesomeApp

Main Thread
- NSRunLoop
- UIKit
- UIApp Delegates

Data Structure
- Q
- ^{}

Locked Access to Data Structure
dispatch_sync(data.q, ^{})

dispatch_sync()
Synchronous Priority Inversion

High QoS thread waiting on lower QoS work
QoS of waited on work is raised for

- `dispatch_sync()` and `dispatch_block_wait()` of Blocks on serial queues
- `pthread_mutex_lock()`
Queues, Threads, and Run Loops

Anthony J. Chivetta Darwin Runtime Engineer
Run Loop Versus Queue

dispatch_async(q, ^{
    [self performSelector:@selector(thing) withObject:nil afterDelay:1];
});
run Loop Versus Queue

dispatch_async(q, ^{
    [self performSelector:@selector(thing) withObject:nil afterDelay:1];
});
Run Loop Versus Queue

dispatch_async(q, ^{
    [self performSelector:@selector(thing) withObject:nil afterDelay:1];
});
Run Loop Versus Queue

dispatch_async(q, ^{
    [self performSelector:@selector(thing) withObject:nil afterDelay:1];
});
Run Loop Versus Queue

dispatch_async(q, ^{
    [self performSelector:@selector(thing) withObject:nil afterDelay:1];
});

Main Thread

NSRunLoop

dispatch_async()
Run Loop Versus Queue

Properties

Run Loop
- Bound to a thread
- Gets delegate method callbacks
- Autorelease pool pops after each iteration
- Can be used reentrantly

Serial Queue
- Uses ephemeral threads
- Block callbacks
- Autorelease pool pops when thread idle
- Will deadlock if used reentrantly

The Main Thread’s Run Loop is also exposed as the Main Queue
RunLoop Versus Queue

Timer APIs

RunLoop

-[NSObject performSelector:withObject:afterDelay:]
+[NSTimer scheduledTimerWithTimeInterval:]

Queue

dispatch_after()
dispatch_source_set_timer()
Thread Creation and Pooling

GCD Thread Pool
Thread Creation and Pooling

GCD Thread Pool
Thread Creation and Pooling

GCD Thread Pool
Thread Creation and Pooling

dispatch_async() → GCD Thread Pool → dispatch_async() → dispatch_async()
Waiting

A thread waits (blocks) when it needs to wait for a resource such as I/O or locks.
When a thread waits, GCD may spin up a new thread to ensure one thread per core.
Thread Creation and Waiting
Thread Creation and Waiting

\[ \wedge \{\ldots\} \quad \wedge \{\ldots\} \quad \wedge \{\ldots\} \quad \wedge \{\ldots\} \]
Thread Creation and Waiting
Thread Explosion
Thread Explosion
Thread Explosion Causing Deadlock

Main Thread

- NSRunLoop
- UIKit

Concurrent Queue

Serial Queue

GCD Thread Pool
Thread Explosion Causing Deadlock

for (int i = 0; i < 999; i++) dispatch_async(q, ^{...})

dispatch_sync(dispatch_get_main_queue(), ^{...})

Main Thread
- NSRunLoop
- UIKit

Concurrency Queues
- Concurrent Queue
  - LIMIT HIT
- Serial Queue

GCD Thread Pool
Thread Explosion Causing Deadlock

for (int i = 0; i < 999; i++) dispatch_async(q, ^{});

dispatch_sync(dispatch_get_main_queue(), ^{});

dispatch_async(q, ^{});

Main Thread

NSRunLoop

UIKit

Concurrent Queue

LIMIT HIT

Serial Queue

async Block

GCD Thread Pool
Thread Explosion Causing Deadlock

Main Thread

NSRunLoop
UIKit

for (int i = 0; i < 999; i++) dispatch_async(q, ^{...})

dispatch_sync(dispatch_get_main_queue(), ^{...})

dispatch_async(q, ^{...})
dispatch_sync(q, ^{...})
Thread Explosion Causing Deadlock

Main Thread

NSRunLoop
UIKit

GCD Thread Pool

Concurrent Queue
LIMIT HIT

Serial Queue
sync Block
No Threads!

for (int i = 0; i < 999; i++) dispatch_async(q, ^{...})

dispatch_sync(dispatch_get_main_queue(), ^{...})

dispatch_async(q, ^{...})
dispatch_sync(q, ^{...})
dispatch_sync(q, ^{...})
Avoiding Thread Explosion

Always good advice: use asynchronous APIs, especially for I/O

Use serial queues

Use NSOperationQueues with concurrency limits

```
NSOperationQueue.maxConcurrentOperationCount
```

Don’t generate unlimited work…
Avoiding Thread Explosion

Mixing sync and async

// fast, just a lock
dispatch_sync(q, ^{...});

// fast, just an atomic enqueue
dispatch_async(q, ^{...});

// slow, has to wait far a thread to complete above Block
dispatch_sync(q, ^{...});

Be super careful about mixing these from the main thread!
Avoiding Thread Explosion

dispatch_apply

// DANGEROUS – may cause thread explosion and deadlocks
for (int i = 0; i < 999; i++){
    dispatch_async(q, ^{...});
}
dispatch_barrier_sync(q, ^{ });

// GOOD – GCD will manage parallelism
dispatch_apply(999, q, ^(size_t i){...});
Avoiding Thread Explosion

`dispatch_semaphore`

```c
#define CONCURRENT_TASKS 4
sema = dispatch_semaphore_create(CONCURRENT_TASKS);
for (int i = 0; i < 999; i++){
    dispatch_async(q, ^{
        // do work
        dispatch_semaphore_signal(sema);
    });
    dispatch_semaphore_wait(sema, DISPATCH_TIME_FOREVER);
}
```
GCD and Crash Reports
Thread 1:: Dispatch queue: com.apple.libdispatch-manager

0  libsystem_kernel.dylib  0x00007fff8967e08a  kevent_qos + 10
1  libdispatch.dylib      0x00007fff8be05811  _dispatch_mgr_invoke + 251
2  libdispatch.dylib      0x00007fff8be05465  _dispatch_mgr_thread + 52
Reading the Tea Leaves

Idle GCD thread

Thread 6:
0  libsystem_kernel.dylib       0x00007fff8967d772 __workq_kernreturn + 10
1  libsystem_pthread.dylib     0x00007fff8fd317d9 _pthread_wqthread + 1283
2  libsystem_pthread.dylib     0x00007fff8fd2ed95 start_wqthread + 13
Thread 3 Crashed:: Dispatch queue: <queue name>

<table>
<thead>
<tr>
<th></th>
<th>Library</th>
<th>Address</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>libdispatch.dylib</td>
<td>0x07fff8fcfd323</td>
<td>_dispatch_call_block_and_release</td>
</tr>
<tr>
<td>8</td>
<td>libdispatch.dylib</td>
<td>0x07fff8fcf8c13</td>
<td>_dispatch_client_callout + 8</td>
</tr>
<tr>
<td>9</td>
<td>libdispatch.dylib</td>
<td>0x07fff8fcfc365</td>
<td>_dispatch_queue_drain + 1100</td>
</tr>
<tr>
<td>10</td>
<td>libdispatch.dylib</td>
<td>0x07fff8fcfdecc</td>
<td>_dispatch_queue_invoke + 202</td>
</tr>
<tr>
<td>11</td>
<td>libdispatch.dylib</td>
<td>0x07fff8fcfb6b7</td>
<td>_dispatch_root_queue_drain + 463</td>
</tr>
<tr>
<td>12</td>
<td>libdispatch.dylib</td>
<td>0x07fff8fd09fe4</td>
<td>_dispatch_worker_thread3 + 91</td>
</tr>
<tr>
<td>13</td>
<td>libsystem_pthread.dylib</td>
<td>0x07fff93c17637</td>
<td>_pthread_wqthread + 729</td>
</tr>
<tr>
<td>14</td>
<td>libsystem_pthread.dylib</td>
<td>0x07fff93c1540d</td>
<td>start_wqthread + 13</td>
</tr>
</tbody>
</table>
Thread 0 Crashed: Dispatch queue: com.apple.main-thread

0  libsyste UITableViewCell 0x00007fff906614de mach_msg_trap + 10
1  libsecko.dylib 0x00007fff9066064f mach_msg + 55
2  com.apple.CoreFoundation 0x00007fff9a8c1eb4 __CFRunLoopServiceMachPort
3  com.apple.CoreFoundation 0x00007fff9a8c137b __CFRunLoopRun + 1371
4  com.apple.CoreFoundation 0x00007fff9a8c0bd8 CFRRunLoopRunSpecific + 296
...
10  com.apple.AppKit 0x00007fff8e823c03 -[NSApplication run] + 594
11  com.apple.AppKit 0x00007fff8e7a0354 NSApplicationMain + 1832
12  com.example 0x0000000001000013b4 start + 52
Thread 0 **Crashed**: Dispatch queue: com.apple.main-thread

```
<my code>
```

```
12  com.apple.Foundation      0x00007fff931157e8 __NSBLOCKOPERATION_IS_CALLING_OUT_TO_A_BLOCK__ + 7
13  com.apple.Foundation      0x00007fff931155b5 -[NSBlockOperation main] + 9
14  com.apple.Foundation      0x00007fff93114a6c -[__NSOperationInternal _start:] + 653
15  com.apple.Foundation      0x00007fff93114543 __NSOQSchedule_f + 184
16  libdispatch.dylib         0x00007fff935d6c13 _dispatch_client_callout + 8
17  libdispatch.dylib         0x00007fff935e2cbf _dispatch_main_queue_callback_4CF + 861
18  com.apple.CoreFoundation  0x00007fff8d9223f9 __CFRunLoop_IS_SERVICING_THE_MAIN_DISPATCH_QUEUE__
19  com.apple.CoreFoundation  0x00007fff8d8dd68f __CFRunLoopRun + 2159
20  com.apple.CoreFoundation  0x00007fff8d8dcbd8 CFRunLoopRunSpecific + 296
...
26  com.apple.AppKit          0x00007fff999a1bd3 -[NSApplication run] + 594
27  com.apple.AppKit          0x00007fff9991e324 NSApplicationMain + 1832
28  libdyld.dylib             0x00007fff9480f5c9 start + 1
```
An efficient and responsive application must be able to adopt to diverse environments
An efficient and responsive application must be able to adopt to diverse environments.

QoS classes enable the system to manage resources appropriately.
An efficient and responsive application must be able to adopt to diverse environments. QoS classes enable the system to manage resources appropriately. Integrate QoS into your application and existing use of GCD.
Summary

An efficient and responsive application must be able to adopt to diverse environments.

QoS classes enable the system to manage resources appropriately.

Integrate QoS into your application and existing use of GCD.

Consider your app’s use of GCD and avoid thread explosion.
More Information

Documentation
Concurrency Programming Guide

Energy Efficiency Guide for Mac Apps

Energy Efficiency Guide for iOS Apps
More Information

Technical Support
Apple Developer Forums
http://developer.apple.com/forums

Developer Technical Support
http://developer.apple.com/support/technical

General Inquiries
Paul Danbold, Core OS Evangelist
danbold@apple.com
<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Achieving All-Day Battery Life</td>
<td>Nob Hill</td>
<td>Wednesday 9:00AM</td>
</tr>
<tr>
<td>Optimizing Your App for Multitasking on iPad in iOS 9</td>
<td>Presidio</td>
<td>Wednesday 3:30PM</td>
</tr>
<tr>
<td>Advanced NSOperations</td>
<td>Presidio</td>
<td>Friday 9:00AM</td>
</tr>
<tr>
<td>Performance on iOS and watchOS</td>
<td>Presidio</td>
<td>Friday 11:00AM</td>
</tr>
<tr>
<td>Related Labs</td>
<td>Frameworks Lab C</td>
<td>Friday 12:00PM</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Power and Performance Lab</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>