Performance on iOS and watchOS
Strategies and tools
Session 230

Ben Englert iOS Performance
Introduction

Why should I think about performance?
Introduction

Why should I think about performance?
How should I think about performance?
Introduction

Why should I think about performance?
How should I think about performance?
Specific strategies
Introduction

Why should I think about performance?
How should I think about performance?
Specific strategies
New platform: watchOS 2
Performance Is a Feature
Performance Is a Feature
Performance Is a Feature

Responsiveness delights and engages users
Performance Is a Feature

Responsiveness delights and engages users

Be a good neighbor, especially in Multitasking on iPad
Performance Is a Feature

Responsiveness delights and engages users
Be a good neighbor, especially in Multitasking on iPad
Efficient apps extend battery life
Performance Is a Feature

Responsiveness delights and engages users
Be a good neighbor, especially in Multitasking on iPad
Efficient apps extend battery life
Supports the whole range of iOS 9 hardware
Thinking About Performance
Thinking About Performance

Choosing technologies
Thinking About Performance

Choosing technologies
Taking measurements
Thinking About Performance

Choosing technologies
Taking measurements
Setting goals
Thinking About Performance

Choosing technologies
Taking measurements
Setting goals
Performance workflow
Use the Right Tool for the Job

Proactively architect your app for great performance
Use the Right Tool for the Job
Proactively architect your app for great performance

Know the technologies
Use the Right Tool for the Job
Proactively architect your app for great performance

Know the technologies
Pick the best ones for your app
Use the Right Tool for the Job
Proactively architect your app for great performance

Know the technologies
Pick the best ones for your app
Apple technologies are optimized
(we use them)
Use the Right Tool for the Job
Proactively architect your app for great performance

Know the technologies
Pick the best ones for your app
Apple technologies are optimized (we use them)
Benefit from software updates
Measuring Performance
Measuring Performance

Animations
Measuring Performance

Animations

• Instruments: Core Animation
Measuring Performance

Animations

• Instruments: Core Animation

Responsiveness
Measuring Performance

Animations
• Instruments: Core Animation

Responsiveness
• Code instrumentation
Measuring Performance

Animations
• Instruments: Core Animation

Responsiveness
• Code instrumentation
• Instruments: System Trace
Measuring Performance

Animations
- Instruments: Core Animation

Responsiveness
- Code instrumentation
- Instruments: System Trace

Memory
Measuring Performance

Animations
• Instruments: Core Animation

Responsiveness
• Code instrumentation
• Instruments: System Trace

Memory
• Xcode debugger
Measuring Performance

Animations
- Instruments: Core Animation

Responsiveness
- Code instrumentation
- Instruments: System Trace

Memory
- Xcode debugger
- Instruments: Allocations
Measuring Performance

Animations
• Instruments: Core Animation

Responsiveness
• Code instrumentation
• Instruments: System Trace

Memory
• Xcode debugger
• Instruments: Allocations
• Instruments: Leaks
@IBAction func showImageTapped(sender: UIButton) {
    let myData = NSData(contentsOfFile: self.path)!
    let myImage = self.watermarkedImageFromData(myData)
    self.imageView.image = myImage
}
@IBAction func showImageTapped(sender: UIButton) {
    let startTime = CFAbsoluteTimeGetCurrent()
    let myData = NSData(contentsOfFile: self.path)!
    let myImage = self.watermarkedImageFromData(myData)
    self.imageView.image = myImage
    let endTime = CFAbsoluteTimeGetCurrent()
}
@IBAction func showImageTapped(sender: UIButton) {
    let startTime = CFAbsoluteTimeGetCurrent()
    let myData = NSData(contentsOfFile: self.path)!
    let myImage = self.watermarkedImageFromData(myData)
    self.imageView.image = myImage
    let endTime = CFAbsoluteTimeGetCurrent()
    let totalTime = (endTime - startTime) * 1000
    print("showImageTappedTimed took \(totalTime) milliseconds")
}
Don’t ship your instrumentation

```swift
@IBAction func showImageTapped(sender: UIButton) {
    #if MEASURE_PERFORMANCE
    let startTime = CFAbsoluteTimeGetCurrent()
    #endif

    let myData = NSData(contentsOfFile: self.path)!
    let myImage = self.watermarkedImageFromData(myData)
    self.imageView.image = myImage

    #if MEASURE_PERFORMANCE
    let endTime = CFAbsoluteTimeGetCurrent()
    let totalTime = (endTime - startTime) * 1000
    print("showImageTappedTimed took \(totalTime) milliseconds")
    #endif
}
```
Code Instrumentation

Measuring responsiveness
Code Instrumentation

Measuring responsiveness

Taps and button presses
Code Instrumentation

Measuring responsiveness

Taps and button presses

- IBAction
Code Instrumentation

Measuring responsiveness

Taps and button presses

- IBAction
- touchesEnded
Code Instrumentation

Measuring responsiveness

Taps and button presses
- IBAction
- touchesEnded
- UIGestureRecognizer target
Code Instrumentation

Measuring responsiveness

Taps and button presses
- IBAction
- touchesEnded
- UIGestureRecognizer target

Tabs and modal views
Code Instrumentation
Measuring responsiveness

Taps and button presses
- IBAction
- touchesEnded
- UIGestureRecognizer target

Tabs and modal views
- viewWillAppear and viewDidAppear
Setting Performance Goals
Setting Performance Goals

60fps scrolling and animations
Setting Performance Goals

60fps scrolling and animations
Setting Performance Goals
Setting Performance Goals

Respond to user actions in 100ms
Setting Performance Goals

Respond to user actions in 100ms

...on older devices!
Setting Performance Goals

Respond to user actions in 100ms

...on older devices!
Performance Workflow
Performance Workflow

Don’t guess
Performance Workflow

Don’t guess

Avoid premature optimization
Performance Workflow

Don’t guess

Avoid premature optimization

Make one change at a time
Performance Workflow

Don’t guess
Avoid premature optimization
Make one change at a time
Just like ordinary debugging
Performance Workflow

1. Change code
2. Measure
3. Reproduce
4. Profile

Flow: Change code → Measure → Reproduce → Profile → Change code
Performance Workflow

1. Change code
2. Measure
3. Reproduce
4. Profile

Flow:
- Change code to Reproduce
- Reproduce to Measure
- Measure to Profile
- Profile to Reproduce
- Reproduce to Change code
Profiling vs. Measuring

Profiling: Understanding overall app activity
- Xcode debugger
- Instruments: Time Profiler

Measuring: Instrumenting a specific action
- CFAbsoluteTimeGetCurrent
- Instruments: System Trace
Responsiveness
Reacting to user input
Main Thread Consumes User Input

- Touches and scrolling
- Orientation
- Multitasking resizes
Main Thread Consumes User Input

Touches and scrolling
Orientation
Multitasking resizes
A responsive main thread makes your app feel great
Main Thread Consumes User Input

Touches and scrolling
Orientation
Multitasking resizes

A responsive main thread makes your app feel great
Busy main thread makes your app appear frozen
Avoid Using the Main Thread for...

CPU-intensive work
Tasks that depend on external resources
Avoid Using the Main Thread for...

CPU-intensive work
Tasks that depend on external resources
What’s a Blocking Call?

Any code path that ends up making a syscall
What’s a Blocking Call?

Any code path that ends up making a syscall
Accessing resources not currently in memory
What’s a Blocking Call?

Any code path that ends up making a syscall
Accessing resources not currently in memory
• Disk I/O
What’s a Blocking Call?

Any code path that ends up making a syscall
Accessing resources not currently in memory
  • Disk I/O
  • Network access
What’s a Blocking Call?

Any code path that ends up making a syscall

Accessing resources not currently in memory

• Disk I/O

• Network access

Waiting for work to complete on another thread
What’s a Blocking Call?

“synchronous” is a synonym for blocking

`NSURLConnection.sendSynchronousRequest(...`
What’s a Blocking Call?

“synchronous” is a synonym for blocking

NSURLConnection.sendSynchronousRequest(...
Strategies for Avoiding Blocking Calls

In many cases, there is an existing asynchronous API you can switch to

```swift
NSURLConnection.sendSynchronousRequest(...)
```
Strategies for Avoiding Blocking Calls

In many cases, there is an existing asynchronous API you can switch to

NSURLConnection.sendAsynchronousRequest(...
Strategies for Avoiding Blocking Calls

In many cases, there is an existing asynchronous API you can switch to
Some restructuring required

NSURLConnection.sendAsynchronousRequest(...
Strategies for Avoiding Blocking Calls

In other cases, there isn’t an async equivalent
Strategies for Avoiding Blocking Calls

In other cases, there isn’t an async equivalent

Use Grand Central Dispatch (GCD)
Strategies for Avoiding Blocking Calls

In other cases, there isn’t an async equivalent

Use Grand Central Dispatch (GCD)

GCD manages a global thread pool
Strategies for Avoiding Blocking Calls

In other cases, there isn’t an async equivalent

Use Grand Central Dispatch (GCD)

GCD manages a global thread pool

Express tasks as closures (a.k.a. blocks)
Strategies for Avoiding Blocking Calls

In other cases, there isn’t an async equivalent
Use Grand Central Dispatch (GCD)
GCD manages a global thread pool
Express tasks as closures (a.k.a. blocks)
Closures run on an arbitrary thread
Strategies for Avoiding Blocking Calls

In other cases, there isn’t an async equivalent

Use Grand Central Dispatch (GCD)

GCD manages a global thread pool

Express tasks as closures (a.k.a. blocks)

Closures run on an arbitrary thread

Ensure operations performed are thread-safe!
Thread Safety

Some objects are restricted to the main thread
Thread Safety

Some objects are restricted to the main thread

Some objects, once created, can be used from any thread
Thread Safety

Some objects are restricted to the main thread
Some objects, once created, can be used from any thread
  • Protection is not built-in
Thread Safety

Some objects are restricted to the main thread
Some objects, once created, can be used from any thread
• Protection is not built-in
• Implement protection using serial GCD queues
Thread Safety

Some objects are restricted to the main thread

Some objects, once created, can be used from any thread

• Protection is not built-in
• Implement protection using serial GCD queues

Read the headers
@IBAction func showImageTapped(sender: UIButton) {
    let myData = NSData(contentsOfFile: self.path)!
    let myImage = self.watermarkedImageFromData(myData)
    self.imageView.image = myImage
}
@IBAction func showImageTapped(sender: UIButton) {
    let myData = NSData(contentsOfFile: self.path)!
    let myImage = self.watermarkedImageFromData(myData)
    self.imageView.image = myImage
}
@IBAction func showImageTapped(sender: UIButton) {
    let myData = NSData(contentsOfFile: self.path)!
    let myImage = self.watermarkedImageFromData(myData)
    self.imageView.image = myImage
}
Strategies for Avoiding Blocking Calls

@IBAction func showImageTapped(sender: UIButton) {
    let myData = NSData(contentsOfFile: self.path)!
    let myImage = self.watermarkedImageFromData(myData)
    self.imageView.image = myImage

Grand Central Dispatch

Current implementation

Button press

- Load file data
- Decode and filter image
- Update image view

Main thread
Grand Central Dispatch

Current implementation

Button press

- Load file data
- Decode and filter image
- Update image view

Scroll or rotation

Main thread
Grand Central Dispatch
Current implementation

Button press
- Load file data
- Decode and filter image
- Update image view
- Handle input
Scroll or rotation
Main thread
Grand Central Dispatch
User input delayed
dispatch_async(dispatch_get_global_queue(QOS_CLASS_USER_INITIATED, 0)) {
    let myData = NSData(contentsOfFile: self.path)!
    let myImage = self.watermarkedImageFromData(myData)
    self.imageView.image = myImage
}
dispatch_async(dispatch_get_global_queue(QOS_CLASS_USER_INITIATED, 0)) {
    let myData = NSData(contentsOfFile: self.path)!
    let myImage = self.watermarkedImageFromData(myData)
    self.imageView.image = myImage
}
Strategies for Avoiding Blocking Calls

Quality of Service (QoS)

dispatch_async(dispatch_get_global_queue(QOS_CLASS_USER_INITIATED, 0)) {
    let myData = NSData(contentsOfFile: self.path)!
    let myImage = self.watermarkedImageFromData(myData)
    self.imageView.image = myImage
}
Strategies for Avoiding Blocking Calls

dispatch_async(dispatch_get_global_queue(QOS_CLASS_USER_INITIATED, 0)) {
    let myData = NSData(contentsOfFile: self.path)!
    let myImage = self.watermarkedImageFromData(myData)
    self.imageView.image = myImage
}
Strategies for Avoiding Blocking Calls

dispatch_async(dispatch_get_global_queue(QOS_CLASS_USER_INITIATED, 0)) {
    let myData = NSData(contentsOfFile: self.path)!
    let myImage = self.watermarkedImageFromData(myData)
    self.imageView.image = myImage
}
Strategies for Avoiding Blocking Calls

```swift
var myImage: UIImage? = nil
dispatch_async(dispatch_get_global_queue(QOS_CLASS_USER_INITIATED, 0)) {
    let myData = NSData(contentsOfFile: self.path)!
    myImage = self.watermarkedImageFromData(myData)
}

self.imageView.image = myImage
```
var myImage: UIImage? = nil

dispatch_async(dispatch_get_global_queue(QOS_CLASS_USER_INITIATED, 0)) {  
    let myData = NSData(contentsOfFile: self.path)!
    myImage = self.watermarkedImageFromData(myData)
}

self.imageView.image = myImage
Grand Central Dispatch

Button press

- GCD
- Update image view

Load file data

Decode and filter image

Main thread

Dispatch queue
Grand Central Dispatch

Button press

myImage == nil

GCD
Update image view

Main thread

Load file data
Decode and filter image

Dispatch queue
Strategies for Avoiding Blocking Calls

dispatch_async(dispatch_get_global_queue(QOS_CLASS_USER_INITIATED, 0)) {
    let myData = NSData(contentsOfFile: self.path)!
    let myImage = self.watermarkedImageFromData(myData)

    dispatch_async(dispatch_get_main_queue()) {
        self.imageView.image = myImage
    }
}
Strategies for Avoiding Blocking Calls

dispatch_async(dispatch_get_global_queue(QOS_CLASS_USER_INITIATED, 0)) {
  let myData = NSData(contentsOfFile: self.path)!
  let myImage = self.watermarkedImageFromData(myData)

  dispatch_async(dispatch_get_main_queue()) {
    self.imageView.image = myImage
  }
}
dispatch_async(dispatch_get_global_queue(QOS_CLASS_USER_INITIATED, 0)) {
    let myData =NSData(contentsOfFile: self.path)!
    let myImage = self.watermarkedImageFromData(myData)

    dispatch_async(dispatch_get_main_queue()) {
        self.imageView.image = myImage
    }
}

Strategies for Avoiding Blocking Calls
Grand Central Dispatch
Timely and thread-safe object access

Button press

GCD
Load file data
Decode and filter image
GCD

Main thread

Update image view

Dispatch queue
Grand Central Dispatch
Timely handling of user input

Button press → GCD

Scroll or rotation

Handle input

Main thread

Load file data → Decode and filter image → GCD

Update image view

Dispatch queue
Common Blocking Calls
Common Blocking Calls

Networking: NSURLConnection and friends

- Use asynchronous API
Common Blocking Calls

Networking: NSURLConnection and friends

- Use asynchronous API
- NSURLSession background session
Common Blocking Calls

Networking: NSURLConnection and friends
- Use asynchronous API
- NSURLSession background session

Foundation initializers
Common Blocking Calls

Networking: NSURLConnection and friends
- Use asynchronous API
- NSURLSession background session

Foundation initializers
- contentsOfFile:
Common Blocking Calls

Networking: NSURLConnection and friends
- Use asynchronous API
- NSURLSession background session

Foundation initializers
- contentsOfFile:
- contentsOfURL:
Common Blocking Calls

Networking: NSURLConnection and friends
- Use asynchronous API
- NSURLSession background session

Foundation initializers
- contentsOfFile:
- contentsOfURL:

Core Data
Common Blocking Calls

Networking: NSURLConnection and friends
• Use asynchronous API
• NSURLSession background session

Foundation initializers
• contentsOfFile:
• contentsOfURL:

Core Data
• Move some Core Data work to different concurrency modes
Common Blocking Calls

Networking: NSURLConnection and friends
• Use asynchronous API
• NSURLConnection background session
Foundation initializers
• contentsOfFile:
• contentsOfURL:
Core Data
• Move some Core Data work to different concurrency modes

What’s New in Core Data

Mission
Thursday 2:30PM
Strategies for Avoiding Blocking Calls

Switch to asynchronous API
Strategies for Avoiding Blocking Calls

Switch to asynchronous API
Use GCD
Strategies for Avoiding Blocking Calls

Switch to asynchronous API

Use GCD
Memory
Memory

Multitasking requires memory tuning
Memory

Multitasking requires memory tuning

watchOS considerations
Memory

Multitasking requires memory tuning
watchOS considerations
Older hardware
Memory

Multitasking requires memory tuning
watchOS considerations
Older hardware
Extensions
iOS Memory System

Never enough to go around
iOS Memory System

Never enough to go around
Suspended apps are not persisted
iOS Memory System

Never enough to go around
Suspended apps are not persisted
They are evicted without storing
iOS Memory System

Never enough to go around
Suspended apps are not persisted
They are evicted without storing
Memory

Memory is time

Reclaiming memory takes time
Memory

Memory is time

Reclaiming memory takes time

Sudden high-memory demand impacts responsiveness
Memory

Memory is time

Reclaiming memory takes time
Sudden high-memory demand impacts responsiveness
Preserves state in the background
Rationalize Your App’s Memory Footprint
Rationalize Your App’s Memory Footprint

Resources
Rationalize Your App’s Memory Footprint

Resources

• Strings
Rationalize Your App’s Memory Footprint

Resources
- Strings
- Images
Rationalize Your App’s Memory Footprint

Resources

• Strings
• Images
• Core Data managed objects
Rationalize Your App’s Memory Footprint

Resources
- Strings
- Images
- Core Data managed objects

Create a mental model of accessed resources
Rationalize Your App’s Memory Footprint

Resources
- Strings
- Images
- Core Data managed objects

Create a mental model of accessed resources
Check your work using Xcode debugger
Rationalize Your App’s Memory Footprint

Resources

• Strings
• Images
• Core Data managed objects

Create a mental model of accessed resources

Check your work using Xcode debugger

Instruments: Allocations and Leaks
Rationalize Your App’s Memory Footprint

Resources
• Strings
• Images
• Core Data managed objects

Create a mental model of accessed resources
Check your work using Xcode debugger
Instruments: Allocations and Leaks
/*
Copyright (C) 2014 Apple Inc. All Rights Reserved.
See LICENSE.txt for this sample's licensing information.

Abstract:
    The application's delegate.
*/

#import "AAPLAppDelegate.h"

@implementation AAPLAppDelegate

@end
Copyright (C) 2014 Apple Inc. All Rights Reserved.
See LICENSE.txt for this sample's licensing information

Abstract:

The application's delegate.

*/

#import "AAPLAppDelegate.h"

@implementation AAPLAppDelegate

@end
<table>
<thead>
<tr>
<th>Resource</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>0%</td>
</tr>
<tr>
<td>Memory</td>
<td>9.7 MB</td>
</tr>
<tr>
<td>Energy Impact</td>
<td></td>
</tr>
<tr>
<td>Disk</td>
<td>Zero KB/s</td>
</tr>
<tr>
<td>Network</td>
<td>Zero KB/s</td>
</tr>
<tr>
<td>FPS</td>
<td></td>
</tr>
</tbody>
</table>
SamplePhotosApp PID 1496

CPU: 0%
Memory: 9.3 MB
Energy Impact
Disk: Zero KB/s
Network: Zero KB/s
FPS
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>0%</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>9.3 MB</td>
</tr>
<tr>
<td><strong>Energy Impact</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Disk</strong></td>
<td>Zero KB/s</td>
</tr>
<tr>
<td><strong>Network</strong></td>
<td>Zero KB/s</td>
</tr>
<tr>
<td><strong>FPS</strong></td>
<td></td>
</tr>
</tbody>
</table>
Application Lifecycle

Use NSCache
Application Lifecycle

Use NSCache

Listen for notifications
Application Lifecycle

Use NSCache

Listen for notifications

- UIApplicationDidEnterBackgroundNotification
Application Lifecycle

Use NSCache

Listen for notifications

• UIApplicationDidEnterBackgroundNotification
• UIApplicationDidReceiveMemoryWarningNotification
Application Lifecycle

Responding to changes

```swift
init() {
    NSNotificationCenter.defaultCenter().
        addObserverForName(UIApplicationDidReceiveMemoryWarningNotification,
                           object: self,
                           queue: NSOperationQueue.mainQueue())
        {[unowned self] (NSNotification notification) -> Void in
            self.purgeCaches() // custom cache purging behavior
        }
}

deinit {
    NSNotificationCenter.defaultCenter().removeObserver(self)
}
Application Lifecycle

Responding to changes

init() {
    NotificationCenter.defaultCenter()
        .addObserverForName(UIApplicationDidReceiveMemoryWarningNotification,
                           object: self,
                           queue: NSOperationQueue.mainQueue())
    {
        [unowned self] (NSNotification notification) -> Void in
            self.purgeCaches() // custom cache purging behavior
    }
}

deinit {
    NotificationCenter.defaultCenter().removeObserver(self)
}
Application Lifecycle

Responding to changes

init() {
    NotificationCenter.defaultCenter().
    addObserverForName(UIApplicationDidReceiveMemoryWarningNotification,
        object: self,
        queue: NSOperationQueue.mainQueue())
    {
        [unowned self] (NSNotification notification) -> Void in
            self.purgeCaches() // custom cache purging behavior
    }
}

deinit {
    NotificationCenter.defaultCenter().removeObserver(self)
}
Memory Strategies
Covered in detail
Memory Strategies
Covered in detail

Optimizing Your App for Multitasking on iPad in iOS 9
Presidio
Wednesday 3:30PM
Memory Strategies
Covered in detail

Resource types and access patterns
Memory Strategies
Covered in detail

Resource types and access patterns
Responding to system memory state while running
New Platform
Native code on watchOS
New Platform
Native code on watchOS

Designing for Apple Watch
Presidio
Wednesday 4:30PM
New Platform
Native code on watchOS

Reuse what makes sense
New Platform
Native code on watchOS

Reuse what makes sense
- Your existing code
New Platform
Native code on watchOS

Reuse what makes sense
• Your existing code
• Familiar APIs and frameworks
New Platform
Native code on watchOS

Reuse what makes sense
• Your existing code
• Familiar APIs and frameworks

Implement new mechanisms

Designing for Apple Watch
Presidio
Wednesday 4:30PM
watchOS
Quick and simple

Short, simple interactions
watchOS
Quick and simple

Short, simple interactions
Recent and relevant data in Apps, Notifications, Glances
watchOS
Quick and simple

Short, simple interactions
Recent and relevant data in Apps, Notifications, Glances
Launch time is critical
watchOS Performance Strategies

Minimize network traffic and processing

Implementing new server logic
watchOS Performance Strategies
Minimize network traffic and processing

Implementing new server logic
• Send appropriately sized and formatted responses
watchOS Performance Strategies

Minimize network traffic and processing

Implementing new server logic

- Send appropriately sized and formatted responses
- Remove unused keys from JSON or XML blobs
watchOS Performance Strategies

Minimize network traffic and processing

Implementing new server logic

- Send appropriately sized and formatted responses
- Remove unused keys from JSON or XML blobs
- Send appropriately sized images
watchOS Performance Strategies
Minimize network traffic and processing

Implementing new server logic

• Send appropriately sized and formatted responses
• Remove unused keys from JSON or XML blobs
• Send appropriately sized images
• Send an appropriate number of records (one screen)
watchOS Performance Strategies
Show fresh, relevant information

Keep app context updated
watchOS Performance Strategies

Show fresh, relevant information

Keep app context updated

• Bidirectional shared state
watchOS Performance Strategies

Show fresh, relevant information

Keep app context updated

• Bidirectional shared state

• `WCSession.defaultSession().updateApplicationContext(...)`
watchOS Performance Strategies
Show fresh, relevant information

Keep app context updated
• Bidirectional shared state
• `WCSession.defaultSession().updateApplicationContext(…)`
• Benefit from Background App Refresh
watchOS Performance Strategies

Minimize network traffic and processing

Relying on existing server logic
watchOS Performance Strategies

Minimize network traffic and processing

Relying on existing server logic

• Implement a lightweight service on iPhone
watchOS Performance Strategies
Minimize network traffic and processing

Relying on existing server logic

• Implement a lightweight service on iPhone
• `WCSession.defaultSession().sendMessage(...)`
watchOS Performance Strategies
Minimize network traffic and processing

Relying on existing server logic

• Implement a lightweight service on iPhone

• `WCSession.defaultSession().sendMessage(...)`

• Parse and pare down server responses on iPhone
watchOS Performance Strategies
Minimize network traffic and processing

Relying on existing server logic
• Implement a lightweight service on iPhone
• `WCSession.defaultSession().sendMessage(...)`
• Parse and pare down server responses on iPhone
• Reply over WCSession with minimal working set
Summary

Performance is a feature
Efficient apps feel great, build trust, and save power
Learn about Apple technologies and choose the best ones for your app
Keep your main thread ready for user input
Understand when and why your app uses memory
On watchOS, fetch and process a minimal set of information
More Information

Documentation
Performance Overview
Instruments User Guide
Concurrency Programming Guide
Threading Programming Guide

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

General Inquiries
Curt Rothert, App Frameworks Evangelist
rothert@apple.com
## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimizing Your App for Multitasking on iPad in iOS 9</td>
<td>Presidio</td>
<td>Wednesday 3:30PM</td>
</tr>
<tr>
<td>Designing for Apple Watch</td>
<td>Presidio</td>
<td>Wednesday 4:30PM</td>
</tr>
<tr>
<td>What's New in Core Data</td>
<td>Mission</td>
<td>Thursday 2:30PM</td>
</tr>
<tr>
<td>Profiling in Depth</td>
<td>Mission</td>
<td>Thursday 3:30PM</td>
</tr>
<tr>
<td>Building Responsive and Efficient Apps with GCD</td>
<td>Nob Hill</td>
<td>Friday 10:00AM</td>
</tr>
<tr>
<td>iOS App Performance: Memory</td>
<td></td>
<td>WWDC12</td>
</tr>
<tr>
<td>Advanced Graphics and Animations for iOS Apps</td>
<td></td>
<td>WWDC14</td>
</tr>
<tr>
<td>Improving Your App with Instruments</td>
<td></td>
<td>WWDC14</td>
</tr>
<tr>
<td>Related Labs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Power and Performance Lab</strong></td>
<td><strong>Frameworks Lab C</strong></td>
<td><strong>Friday 12:00PM</strong></td>
</tr>
</tbody>
</table>