iOS Performance and Power Optimization with Instruments

Session 312

Tim Lee
iOS Performance

Introduction

- Performance and power are important!
- Key aspect of App Store reviews
- You have the tools and skills to improve performance
- Today: cover common cases and strategies

What You'll Learn

- How to measure performance and power scenarios
- How to improve key scenarios
  - Speedy interaction
  - Slim memory footprint
  - Use network and battery efficiently

Measuring Performance

Measuring Performance

Strategy

- Don’t guess
- Take measurements
- It has to feel right

Measuring Performance

Focus on scenarios

- Measure a single interaction
- Change code
- Measure again

Measuring Performance

Strategy

- Test with realistic data sets
- Test on the slowest device you plan to support

Measuring Performance

Tools

- Instruments
- Logging
  - `NSLog(@”That took %g seconds\n”, timeWeWaited);`
  - Log to file
  - Control with `#define`, environment variables, or user preferences
- Simulator (for memory only)
- Side-by-Side

Improving Performance

Speed and responsiveness

Speed and Responsiveness

Importance

- Slow performance will cause OS to abort your app
    - Maintain system responsiveness
    - Beware the system “watchdog”
- Launch and resume are particularly important
- Smooth scrolling

Watchdog Checks	Maximum Time
Launch	20 sec
Resume	10 sec
Suspend	10 sec
Quit	6 sec
Complete operation	10 min

Optimizing for Speed

Strategies

- Do less work
- Do work later
- Do work faster
  - Focus on slowest paths
  - Use placeholders until the work is done
  - Do slow work in the background

Improving Performance

Slim memory footprint

Slim Memory

Jetsam and memory warnings

- “Jetsam”
  - Watches memory pressure
  - Instant lightweight termination of applications
- Larger suspended apps are first
- Memory warnings are your chance to save yourself

Slim Memory

Areas to focus on

- Spikes
- Leaks
- Abandoned memory

Slim Memory

Spikes

- Definition: individual brief allocations all present simultaneously
- Processing large quantities of data
    - Break into independent batches
- Autoreleased objects
    - Reduce object lifetimes

Slim Memory

Autorelease

- “Used to avoid worrying about retain/release”

Slim Memory

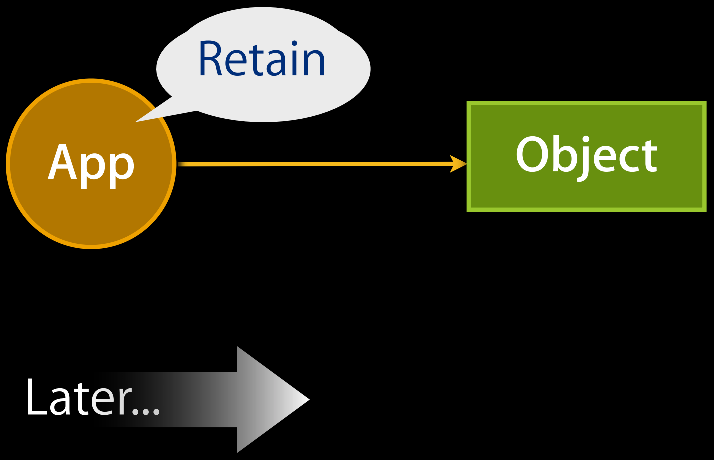
Autorelease

- ~~“Used to avoid worrying about retain/release”~~
- Way for frameworks to manage object ownership

Slim Memory

Retain/Release

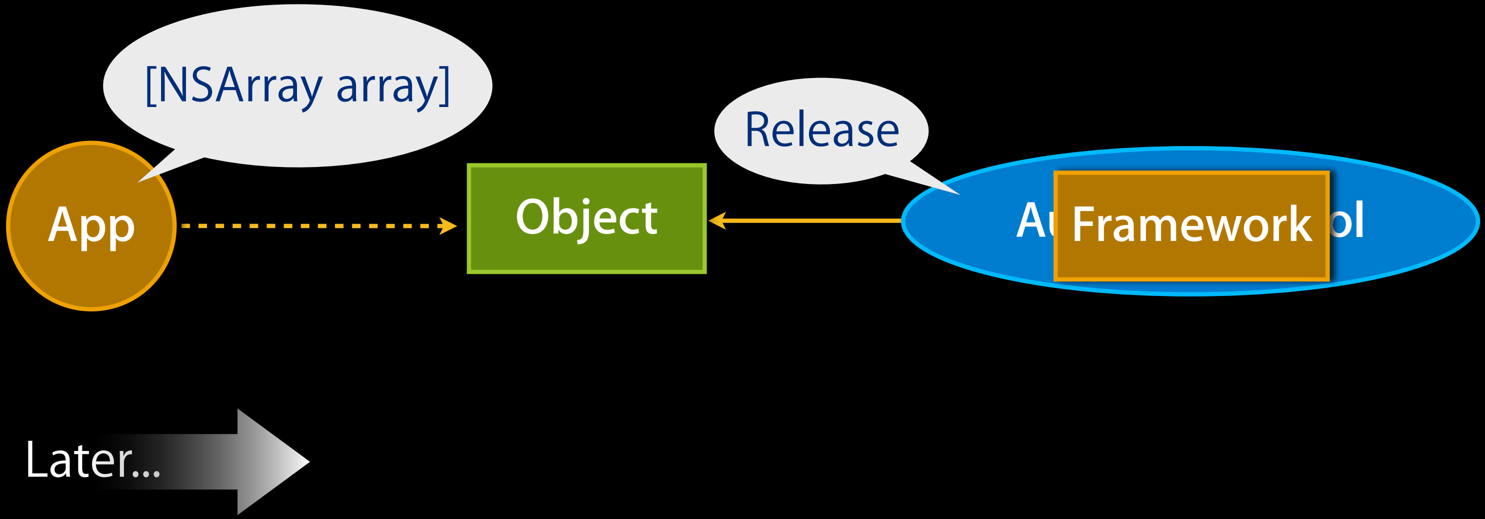
- App asks for an object (alloc/init)
- App is responsible for releasing it



Slim Memory

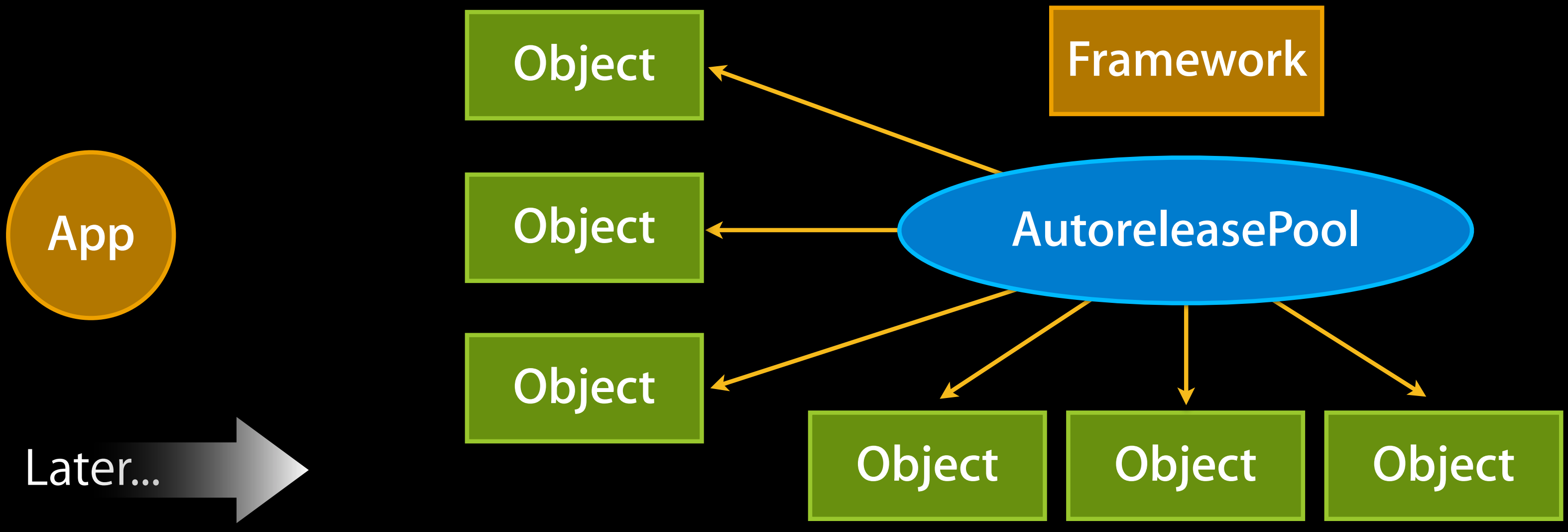
Autorelease

- App asks a framework for an object
- Framework doesn't know when app is done with the object
- Leaves it to AutoreleasePool to release



Slim Memory
Autorelease

- Can lead to memory spikes
- ARC alleviates this problem
- Use nested autorelease pools to fix



Slim Memory

Leaks

- Definition: allocated memory that is inaccessible
- Leaks instrument
  - Examines the heap for leaked memory
  - Identifies moment of allocation
    - Problem is usually lack of release, but this provides context
- Common mistakes
  - Unbalanced retain/release
  - Forget to release property’s original value
- ARC largely removes this problem (if project uses it)

Slim Memory

Abandoned memory

- Definition: left over; accessible, but will never be used again
- Allocations Instrument offers “heapshot”
- Two snapshots in time
- Look at (unexpected) differences
- ARC doesn’t help here

Demo

Time Profiler, Allocations, Leaks Instruments

Speed and Responsiveness

Review

- System watchdog will terminate slow apps
- Do less, do later, do faster
- Do slow operations in the background
- Optimize time-consuming activities
- Only load what you need at launch

Slim Memory

Review

- Spikes, Leaks, Abandonments
- Jetsam will terminate your app
    - Memory warnings are your last chance
- Instruments: Leaks, Allocations, VMTracker
- Add extra autorelease pools to avoid spikes
- Use ARC

iOS Networking and Power Optimizations

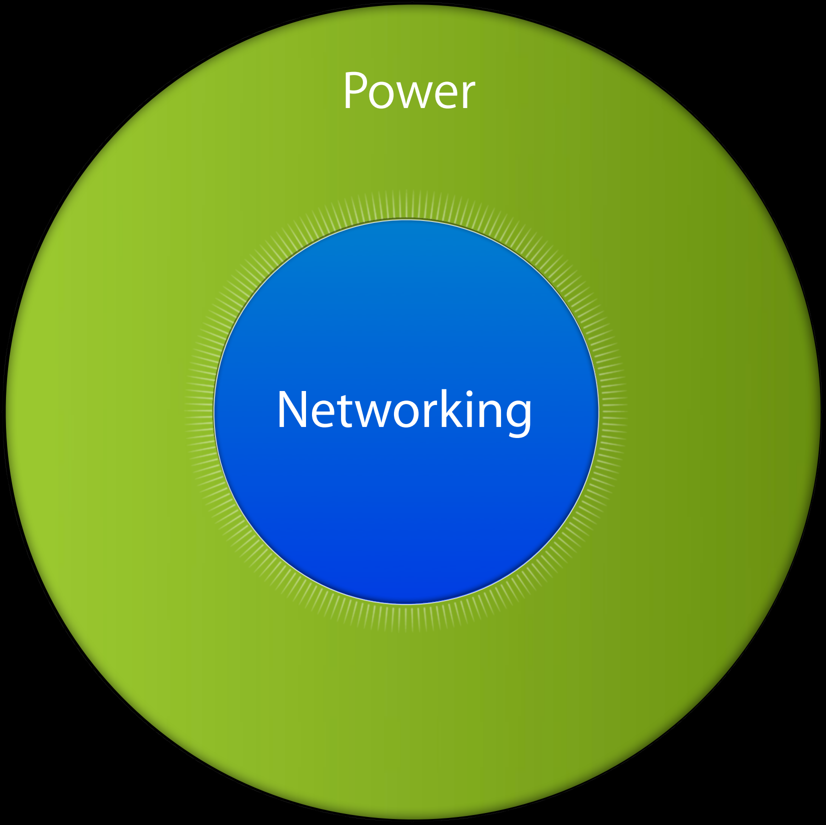
Chad Woolf
Performance Tools Engineer

Performance Optimization

- Not just about speed
- Efficiency
- Faster code is a benefit

Networking and Power

Optimizations



Optimizations

Networking and power

1. Reducing Network Traffic
2. Bursting
3. CoreLocation Accuracy
4. Sleep/Wake
5. Dynamic Frame Rates

1

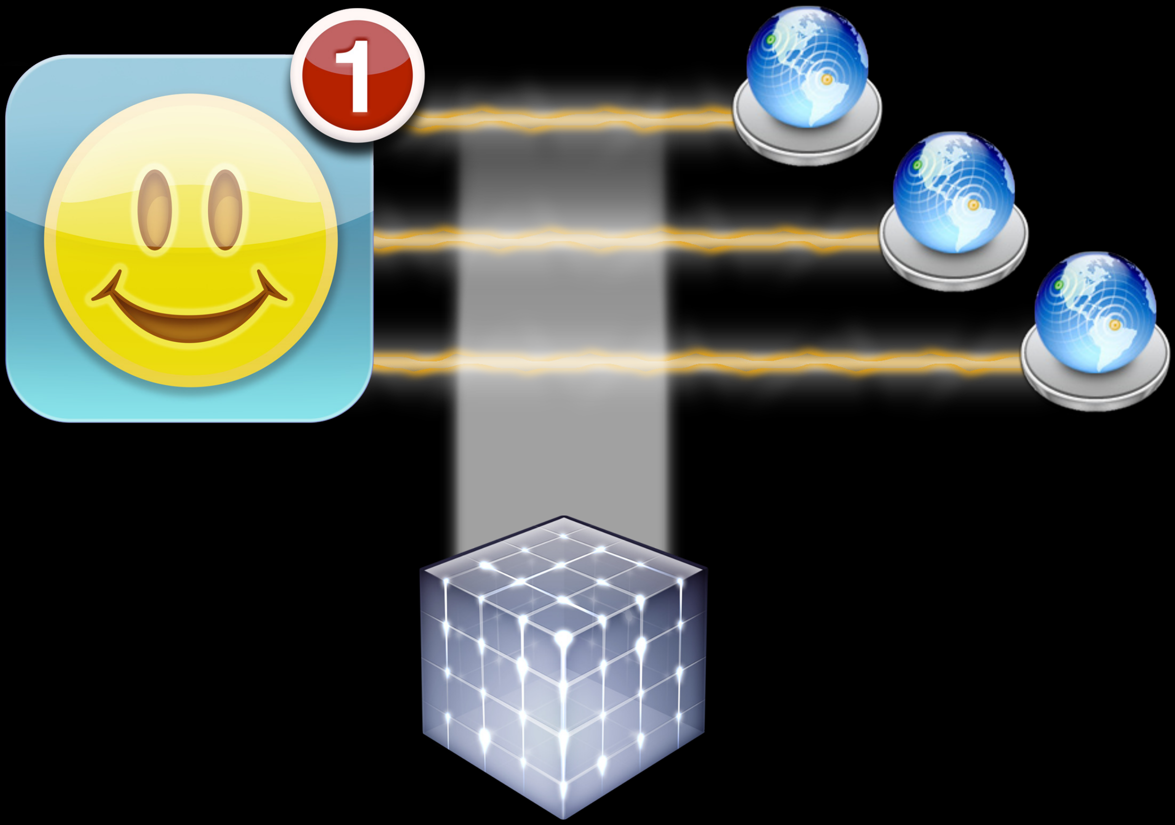
Reducing Network Traffic

Reducing Networking Traffic

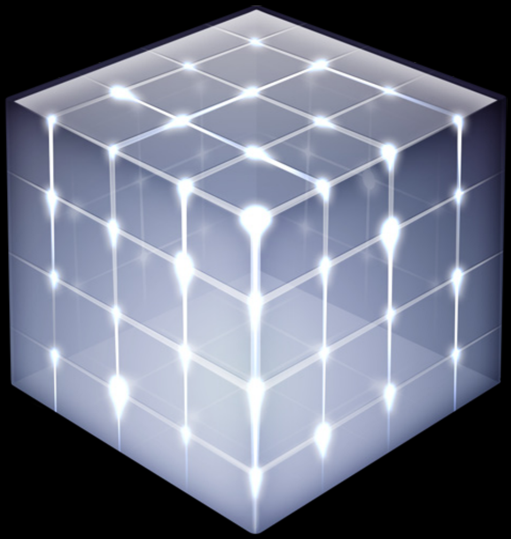
Opportunity

- Reduces network congestion
- Saves customers money
- Saves energy (battery life)

Measuring Traffic



Network Connections Instrument



- Measure data volume
- TCP/IP and UDP/IP
- Performance metrics

Demo

Network connections

Caching Content

Optimization technique

- Redundant downloads are bad
- Use URL Loading System in Foundation
  - HTTP aware
- `NSURLCache`
  - Memory
  - Persistence in iOS 5

Compression

Optimization technique

- Start with compact formats
- Compress when possible
- Reduce large images

Resumable Transfers

Optimization technique

- Connections break often
- Restarting redundant
- Resuming is better
  - HTTP `Range: 100000-`

Download Profiling

Optimization technique

- Watch how your customers use your app
- Don’t download more than they’re likely to consume
- Add logging and send statistics

Reduce Traffic

Summary

- Measure first
- Cache content
- Compress content
- Use resumable transfers
- Download only what's likely to be used

2

Bursting

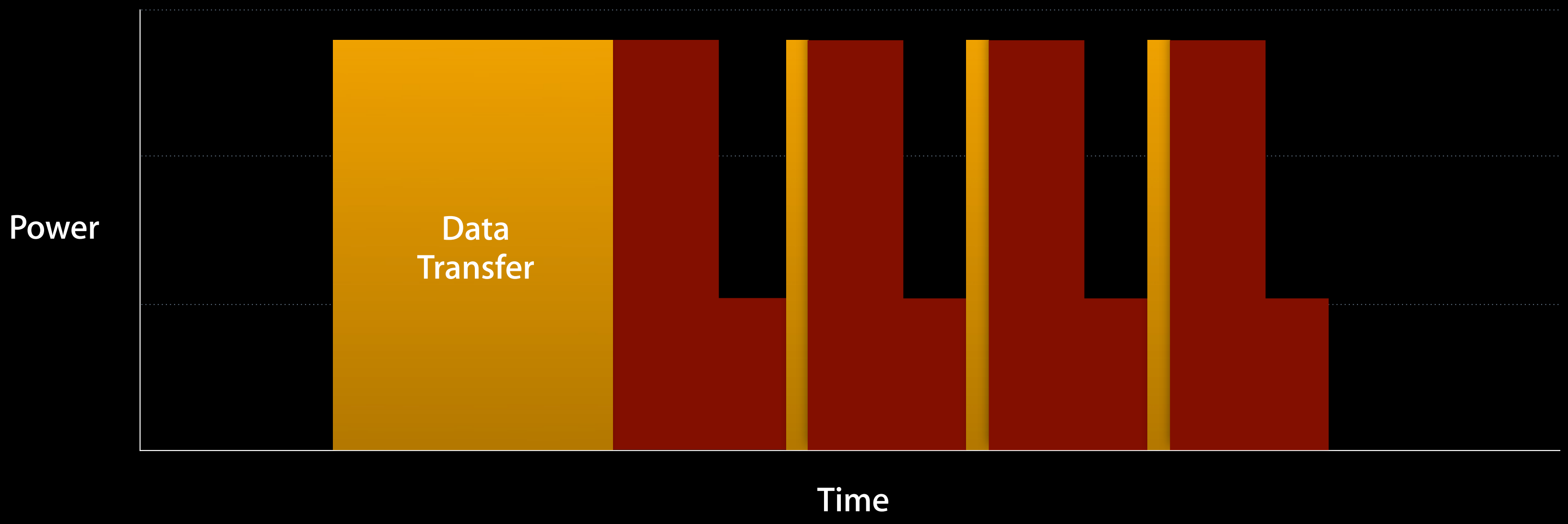
Bursting

Opportunity

- Transmit/receive all at once
- Don’t use the network in between
- Saves on energy consumption

Bursting
Opportunity

- Sending and receiving consumes significant energy
- Radio power stays high for up to 10 seconds
- The next packet resets the timer



Bursting

Measuring

- Energy Diagnostics template in Instruments
  - Energy
  - CPU
  - Power states
- Network Activity instrument
  - New in iOS 5
- Energy usage sampled more frequently

Demo

Energy Diagnostics and Bursting

Bursting

Optimization techniques

- Accumulate outgoing data
- Delay transmission
- Exceptions
    - Real-time streaming
    - Real-time multiplayer gaming

❸

CoreLocation Accuracy

CoreLocation

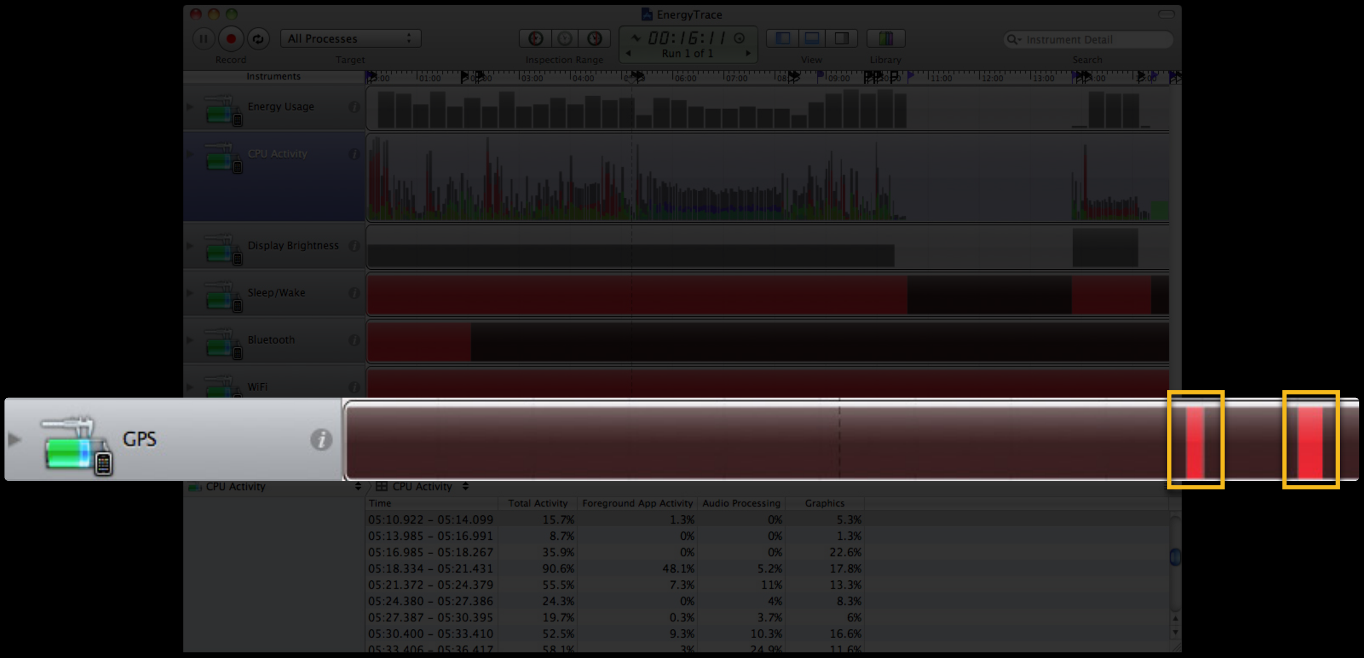
Opportunity

- Several levels of accuracy
- Higher accuracy requires more energy
- Choose the most suitable accuracy

CoreLocation

Measuring

- Use the Energy Diagnostics template
- GPS “on” means more energy



Time	Total Activity	Foreground App Activity	Audio Processing	Graphics
05:10.922 – 05:14.099	15.7%	1.3%	0%	5.3%
05:13.985 – 05:16.991	8.7%	0%	0%	1.3%
05:16.985 – 05:18.267	35.9%	0%	0%	22.6%
05:18.334 – 05:21.431	90.6%	48.1%	5.2%	17.8%
05:21.372 – 05:24.379	55.5%	7.3%	11%	13.3%
05:24.380 – 05:27.386	24.3%	0%	4%	8.3%
05:27.387 – 05:30.395	19.7%	0.3%	3.7%	6%
05:30.400 – 05:33.410	52.5%	9.3%	10.3%	16.6%
05:33.406 – 05:36.417	58.1%	3%	24.9%	11.6%

CoreLocation

Optimization technique

- Use least amount of accuracy—default is `kCLLocationAccuracyBest`

GPS:	`kCLLocationAccuracyBest, BestForNavigation`
GPS:	`kCLLocationAccuracyNearestTenMeters`
Wi-Fi:	`kCLLocationAccuracyHundredMeters`
Cell/Wi-Fi:	`kCLLocationAccuracyKilometer, ThreeKilometers`

```
CLLocationManager *locationManager = [[CLLocationManager alloc] init];
locationManager.desiredAccuracy = kCLLocationAccuracyHundredMeters;
[locationManager startUpdatingLocation];
[locationManager stopUpdatingLocation];
```

4

Sleep/Wake

Zzz...

Sleep/Wake

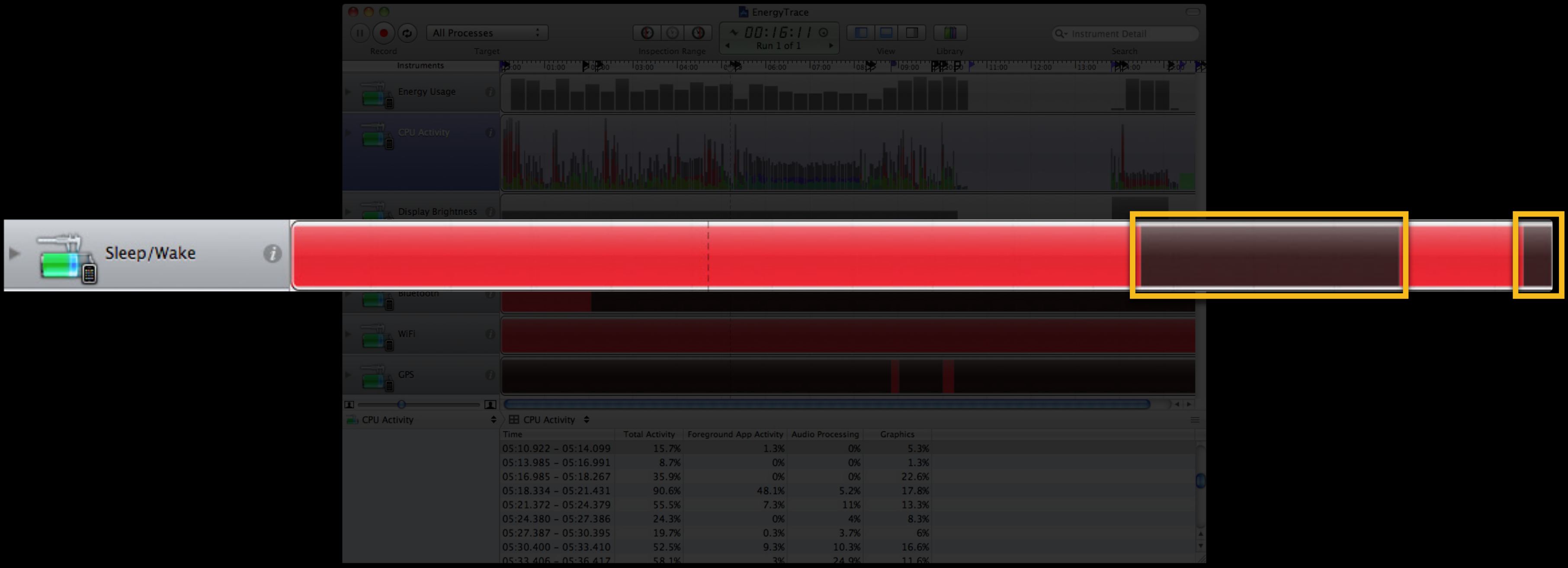
Opportunity

- Battery life depends on sleep
- Don’t keep the device awake
- Don’t wake the device unnecessarily

Sleep/Wake

Measuring

- Use the Energy Diagnostics template
- Watching for periodic wakes



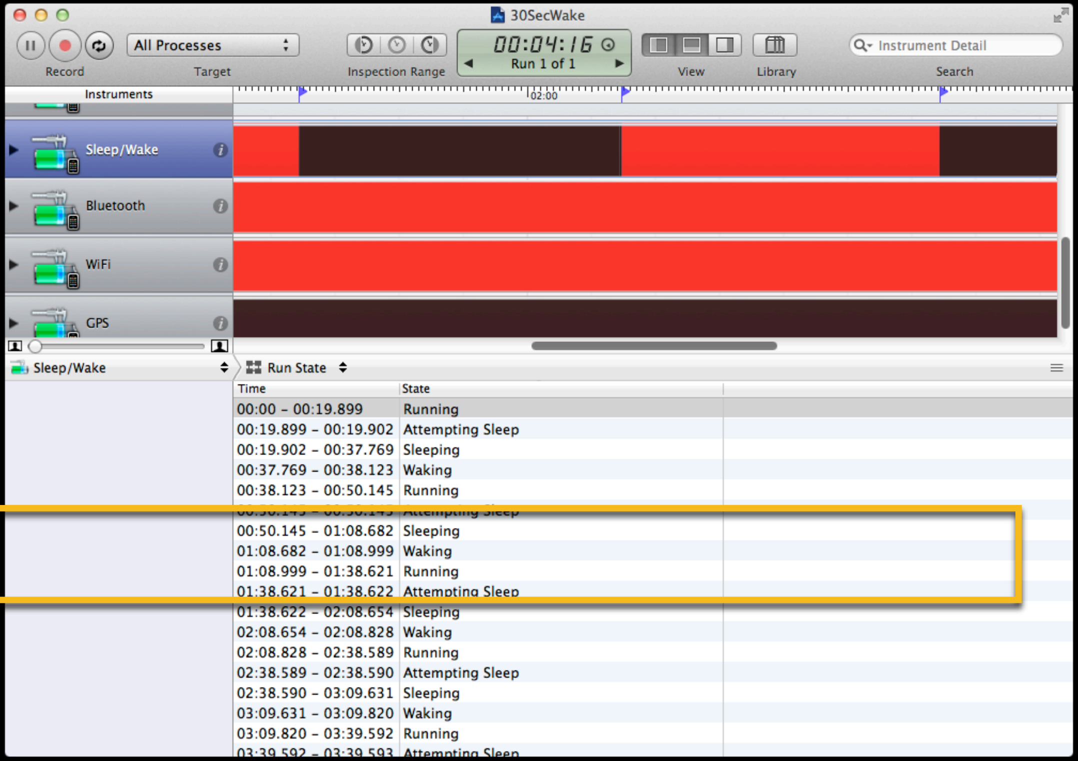
CPU Activity

Time	Total Activity	Foreground App Activity	Audio Processing	Graphics
05:10.922 – 05:14.099	15.7%	1.3%	0%	5.3%
05:13.985 – 05:16.991	8.7%	0%	0%	1.3%
05:16.985 – 05:18.267	35.9%	0%	0%	22.6%
05:18.334 – 05:21.431	90.6%	48.1%	5.2%	17.8%
05:21.372 – 05:24.379	55.5%	7.3%	11%	13.3%
05:24.380 – 05:27.386	24.3%	0%	4%	8.3%
05:27.387 – 05:30.395	19.7%	0.3%	3.7%	6%
05:30.400 – 05:33.410	52.5%	9.3%	10.3%	16.6%
05:33.406 – 05:36.417	58.1%	3%	24.9%	11.6%

Sleep/Wake

Case study

- Normal
  - ~300 hours standby
- Woken every 30s
  - ~30 hours standby



Time	State
00:00 – 00:19.899	Running
00:19.899 – 00:19.902	Attempting Sleep
00:19.902 – 00:37.769	Sleeping
00:37.769 – 00:38.123	Waking
00:38.123 – 00:50.145	Running
00:50.145 – 00:50.145	Attempting Sleep
00:50.145 – 01:08.682	Sleeping
01:08.682 – 01:08.999	Waking
01:08.999 – 01:38.621	Running
01:38.621 – 01:38.622	Attempting Sleep
01:38.622 – 02:08.654	Sleeping
02:08.654 – 02:08.828	Waking
02:08.828 – 02:38.589	Running
02:38.589 – 02:38.590	Attempting Sleep
02:38.590 – 03:09.631	Sleeping
03:09.631 – 03:09.820	Waking
03:09.820 – 03:39.592	Running
03:39.592 – 03:39.593	Attempting Sleep

Sleep/Wake

Optimization techniques

- Use push notifications carefully
- Don’t wake if the user isn’t responding
- Let the device sleep as long as possible

5

Dynamic Frame Rates

Dynamic Frame Rates

Opportunity

- Smoothest animations are 60 fps
- Some scenes require less
- Reduce when quality isn't impacted

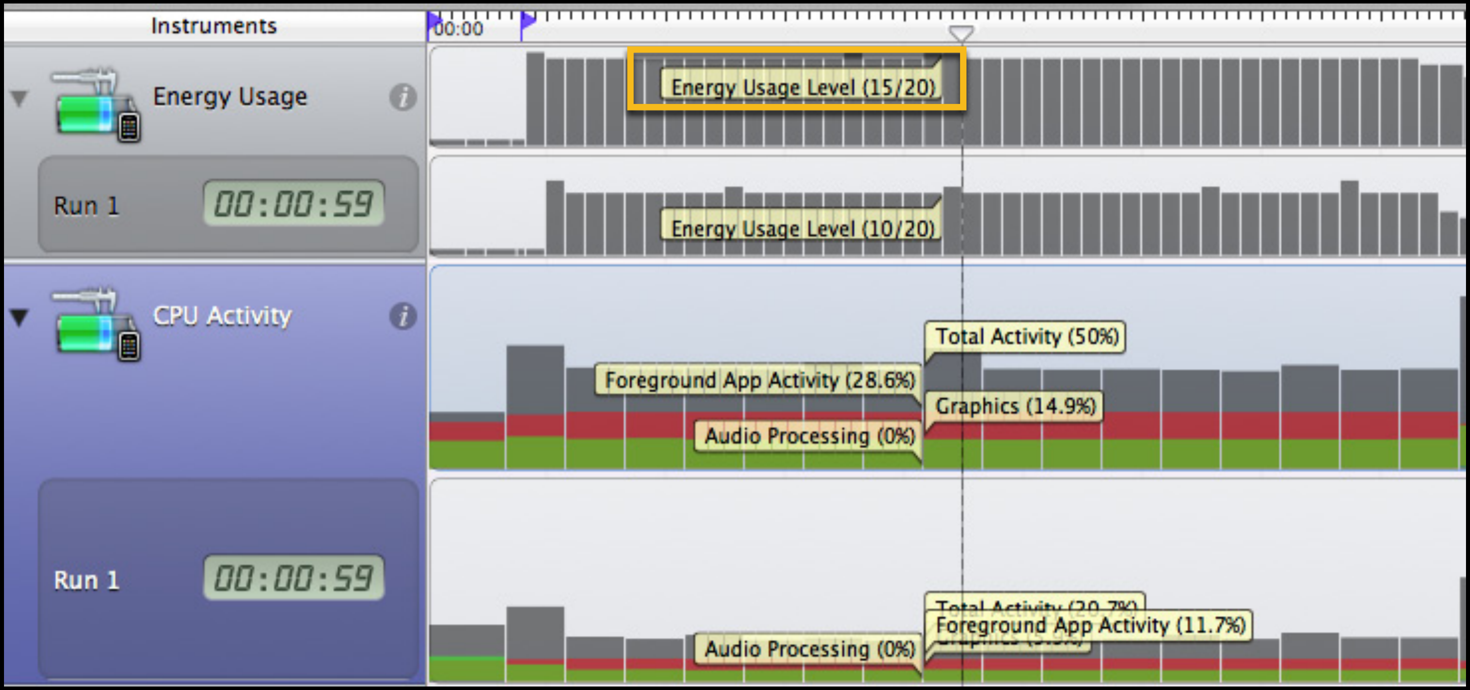
Dynamic Frame Rates

Measuring

- Use Energy Diagnostics template
- Use Core Animation template
- Look for high Foreground App and Graphics activity
- Watch the Energy Usage instrument

Dynamic Frame Rates

Case study



Dynamic Frame Rates

Optimization technique

- Draw only what’s new
- Experiment
- Reduce your CPU and GPU activity

Wrapping Up

Review

- Measure
  - Instruments
  - iTunes Connect reports
- Goal: Lean Apps
- Performance is about efficiency

More Information

Michael Jurewitz
Developer Tools Evangelist
jurewitz@apple.com

Documentation
Instruments User Guide
http://developer.apple.com

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

Related Sessions

Introducing Automatic Reference Counting	Presidio Tuesday 4:30PM
What’s New in Instruments	Marina Wednesday 2:00PM
iOS Performance In-Depth	Presidio Thursday 4:30PM

Related Labs

iOS App Performance Lab	Developer Tools Lab A Thursday 9:00AM

WWDC2011