Advanced Performance Optimization on iPhone OS

Part 1: Animations, responsiveness, and battery life

David Chan
iOS Performance

Peter Handel
iOS Power

“The iPad is a far slower machine than a modern MacBook in terms of raw hardware performance, but it feels faster in many ways, because you never have to wait for it.”

John Gruber, Daring Fireball

Introduction

- Great performance is all about creating an outstanding experience
- This session is for our most advanced developers
  - Part 2 covers memory, data, and I/O
- Solving your application's performance challenges
  - Learn about the system
  - Think creatively
  - Measure progress

What You'll Learn About

- Animation and scrolling
- Responsiveness
- Power and battery life

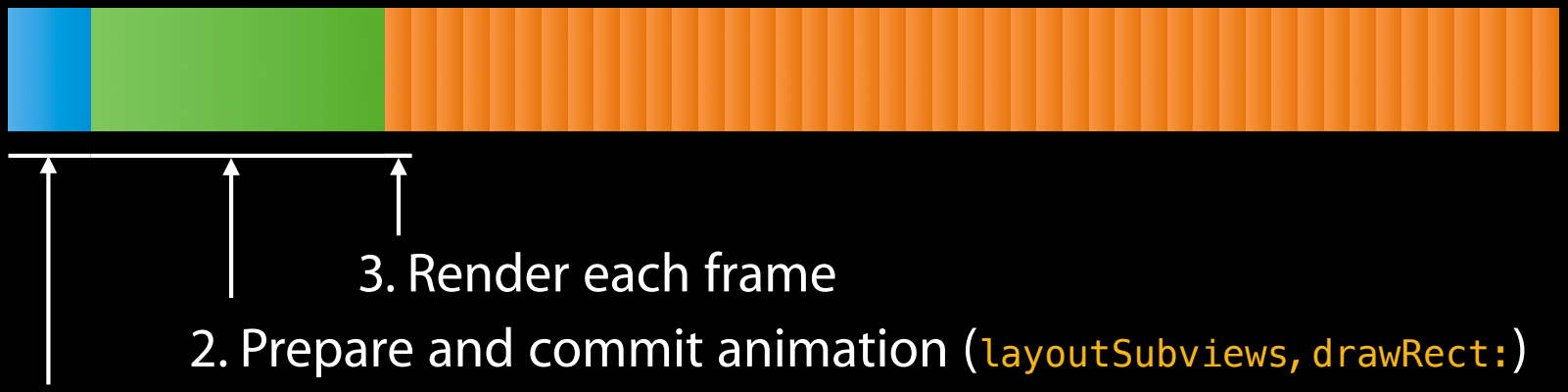
Animation and Scrolling

Animation and Scrolling

- Behind the scenes
- Responsive animations
- Smooth animations
- Smooth scrolling
- Device considerations

Behind the Scenes

Stages of an animation



3. Render each frame

2. Prepare and commit animation (`layoutSubviews`, `drawRect:`)

1. Create animation and update view hierarchy

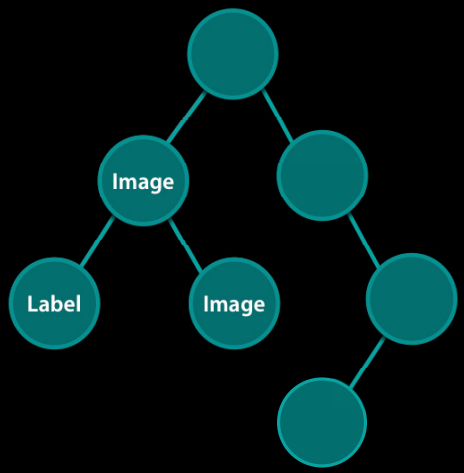
Creating an Animation

```
view = [[InsideView alloc] initWithFrame:frame];
view.transform = CGAffineTransformMakeScale(
 1 / width, 1 / height);
[UIView beginAnimations:nil context:nil];
[UIView setAnimationDuration:0.5];
[self addSubview:view];
view.transform = CGAffineTransformIdentity;
[UIView commitAnimations];
```

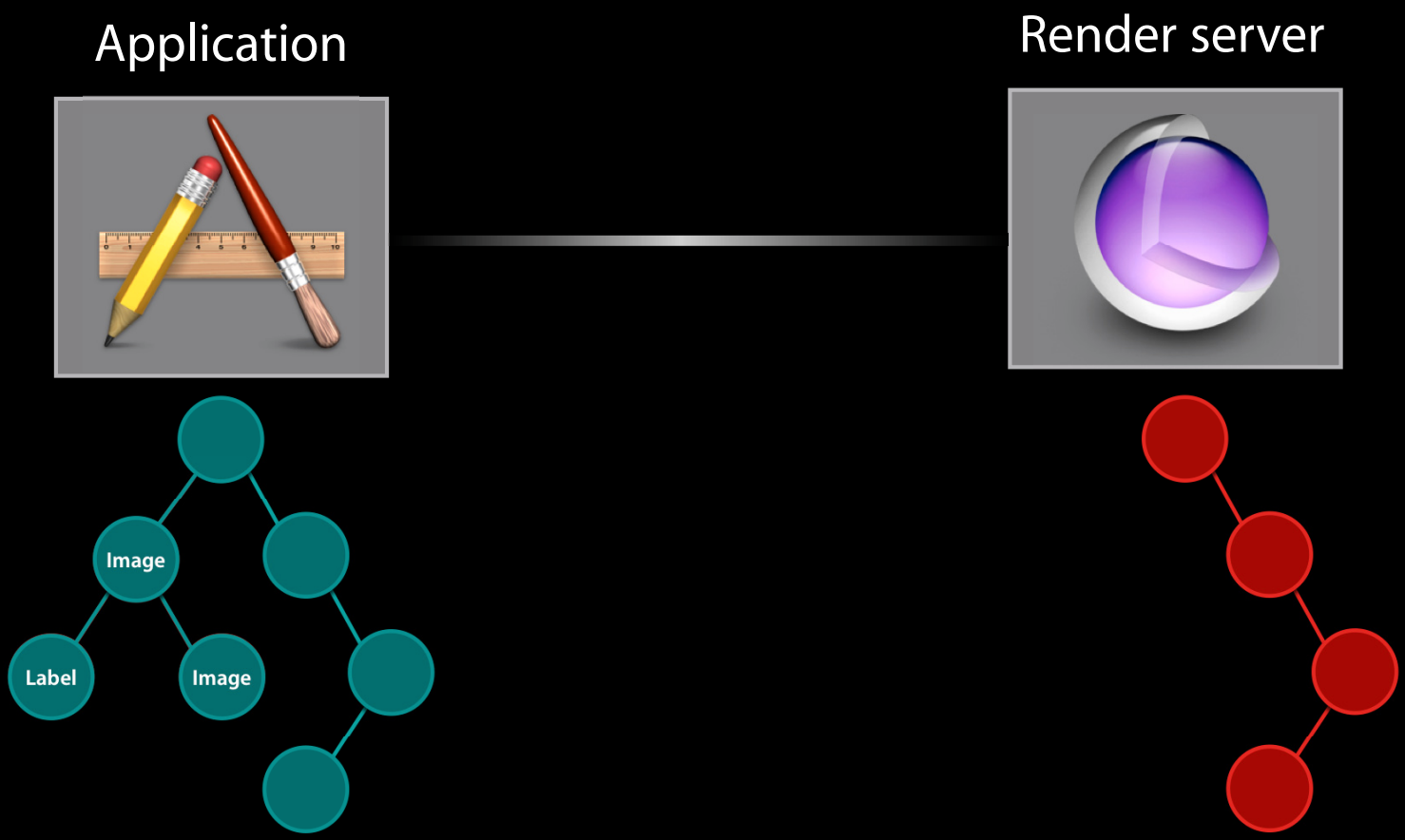
Preparing the Animation

1. Create new views or change properties in an animation block

2. The animation is prepared for commit by calling `layoutSubviews` and `drawRect` on each new view

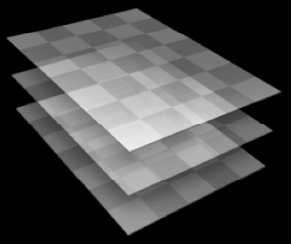


Committing the Animation



Rendering the Animation

1. Updates render tree for current time according to animations



2. Calculates screen regions to be updated

4. Presents rendered update to display

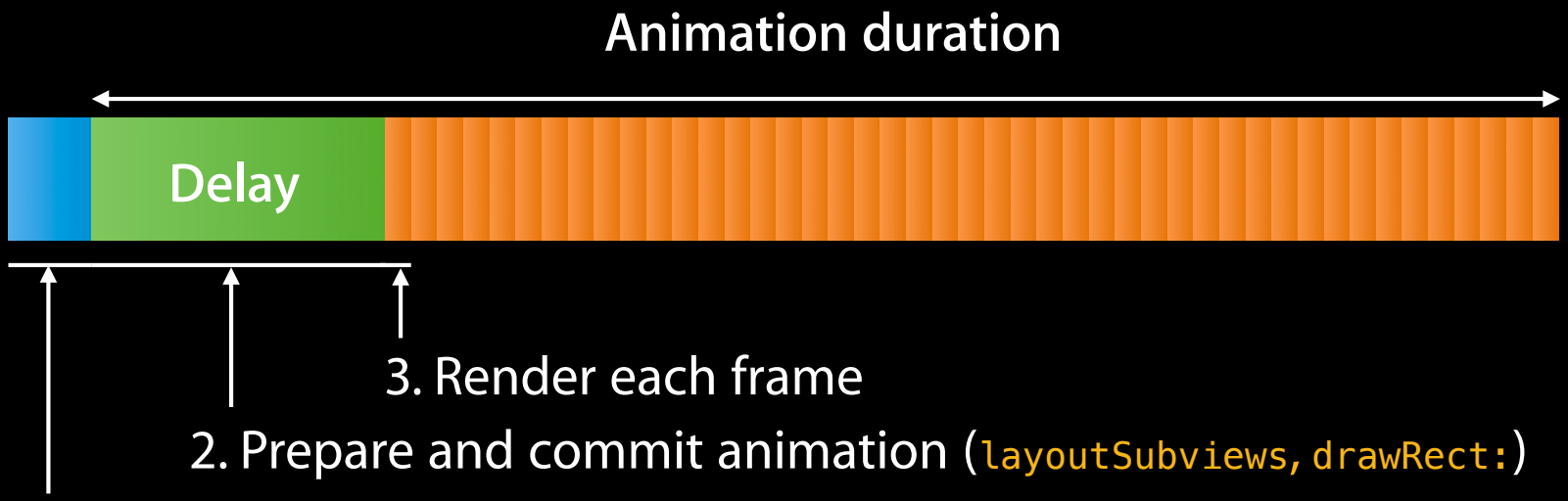
3. Generates commands to render updated region

Animations and Scrolling

- Behind the scenes
- Responsive animations
- Smooth animations
- Smooth scrolling
- Device considerations

Responsive Animations

Finding the delay



Responsive Animations

Draw less while preparing

- Only invalidate views that need to be updated
  - Only call `setNeedsDisplay` on visible views
  - Only implement `drawRect:` when absolutely needed
- Invalidate smaller regions of large views
  - Implement a smart `drawRect:` and use `setNeedsDisplayInRect:` instead
  - Decompose views into static and dynamic parts

Responsive Animations

Dealing with images

- Only use sizes and formats appropriate for the device
  - Decompress and rescale big images sparingly
  - iPhone-optimized PNGs, JPEGs, and TIFFs
- Avoid copying of custom CGImages by using UIGraphics functions
  - Detect using "Color Copied Images" debug option

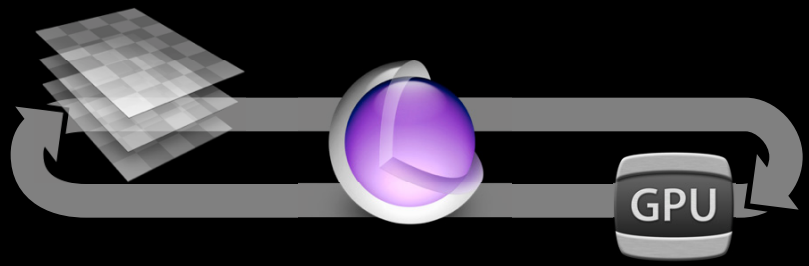
Smooth Animations

- Rendering each frame
- Reduce view blending
- Reduce offscreen rendering
- Dynamic flattening

Smooth Animations

Rendering each frame

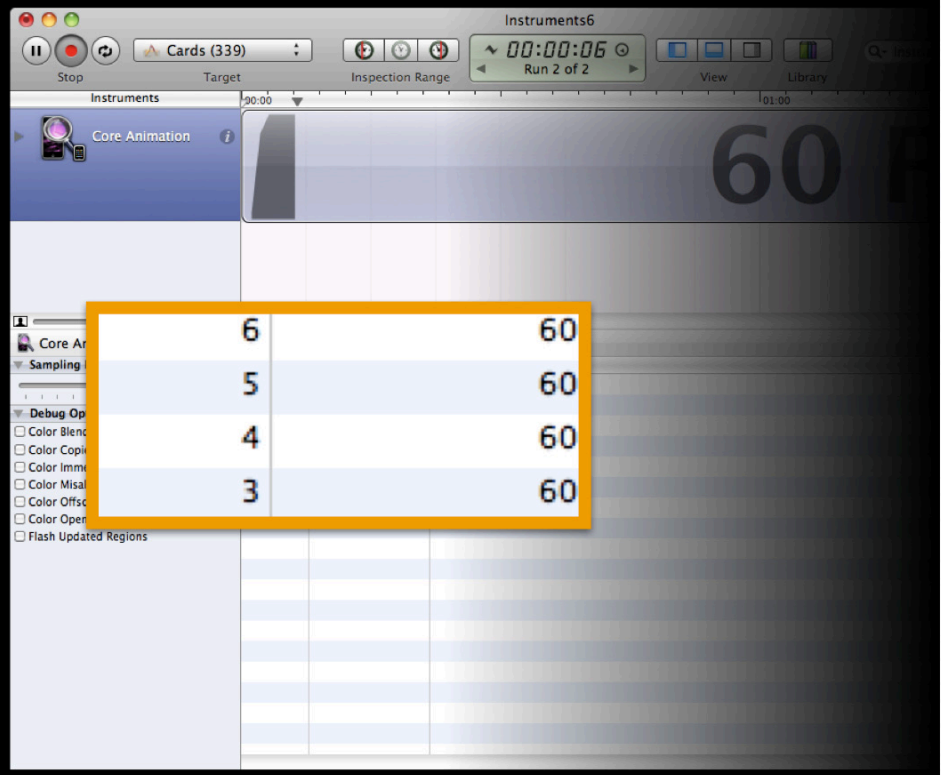
- Server tries to render each frame of your animation 60 times per second
- Fewer pixels to render means smoother animations
  - Fewer input pixels
  - Fewer output pixels
  - Fewer rendering passes



Smooth Animations

Measuring improvements

- Core Animation Instrument
- Always measure baseline and changes
- Reported fps is a count, not a rate
  - e.g., 18 frames/300 ms = 60 fps
- Lengthen animation over a few seconds for a better measurement



Smooth Animations

What's being rendered?

- Flash Updated Regions
- Parts of your application will flash yellow when the renderer is invoked to update that region
- Simplify structure of view hierarchy
- Remove unnecessary or invisible views

Smooth Animations

- Rendering each frame
- Reduce view blending
- Reduce offscreen rendering
- Dynamic flattening

Smooth Animations

Reduce view blending

- Color Blended Layers
- Opaque regions shaded green
- Blended regions shaded red
  - Deeper blending darkens red

Smooth Animations

Reduce view blending

- Graphics system can perform certain number of pixel operations per frame to maintain smooth frame rate
  - Blending requires more operations per on-screen pixel
- Graphics system supports efficient hidden surface removal
  - Only avoids views that are completely occluded by opaque views

Smooth Animations

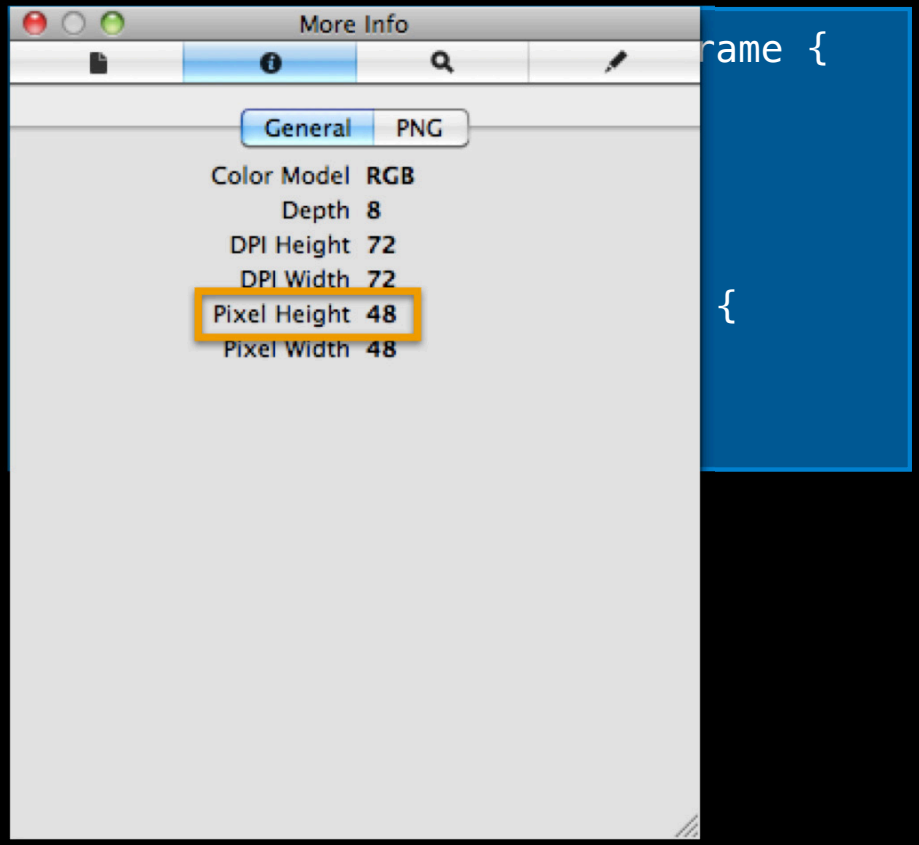
Reduce view blending

Approximate pixel operations per frame at 60 fps

Smooth Animations

Reduce view blending

- Contents determine blending
- Keep views opaque
- Use image assets without alpha
- Create opaque CGImages using UIGraphics



Smooth Animations

- Rendering each frame
- Reduce view blending
- Reduce offscreen rendering
- Dynamic flattening

Smooth Animations

Reduce offscreen rendering

- Color Offscreen-Rendered Yellow
- Regions shaded yellow when compositor used a temporary offscreen region to render the final result
- Switching between main and offscreen contexts stalls pipeline
- Necessary to achieve some effects
- Avoiding requires creative solutions

Smooth Animations

Reduce offscreen rendering

- Example: Fade opacity of image with a background color
- To composite correctly, the image must be composited over the color offscreen and then blended

```
UIImageView *view = [[UIImageView alloc]
 initWithImage:image];
view.backgroundColor = [UIColor brownColor];
[UIView beginAnimations:nil context:nil];
view.alpha = 0;
[UIView commitAnimations];
```

Smooth Animations

Reduce offscreen rendering

- Workaround: Composite background color and image together in `drawRect:`

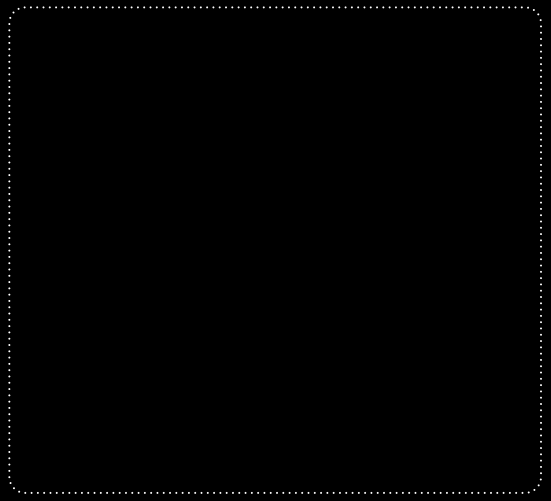
```
- (void)drawRect:(CGRect)rect {
    [[UIColor brownColor] setFill];
    UIRectFill(rect);
    [image drawInRect:rect];
}
```

Smooth Animations

Reduce offscreen rendering

- Workaround: If fading over a static background, try fading in background over view instead

```
UIView *view = [[UIView alloc]
 initWithFrame:self.bounds];
view.backgroundColor = [UIColor blackColor];
view.alpha = 0;

[UIView beginAnimations:nil context:nil];
[UIView setAnimationDuration:0.2];
view.alpha = 1;
[UIView commitAnimations];
```



Smooth Animations

Reduce offscreen rendering

- Example: Animating view with rounded corner mask
- Subviews must be composited together before any complex masking

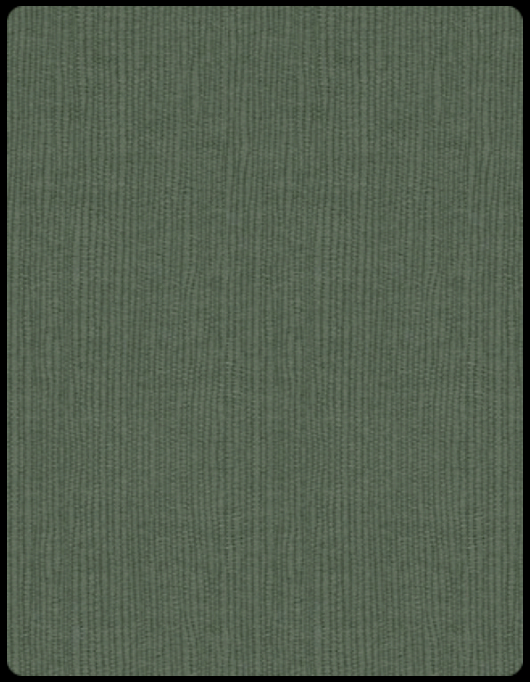
```
view.layer.cornerRadius = 10.0;
view.layer.masksToBounds = YES;
```

Smooth Animations

Reduce offscreen rendering

- Workaround: Mask background in `drawRect:`

```
- (void)drawRect:(CGRect)rect {
    [[UIBezierPath bezierPathWithRoundedRect:rect
 cornerRadius:10.0] addClip];
    [image drawInRect:rect];
}
```

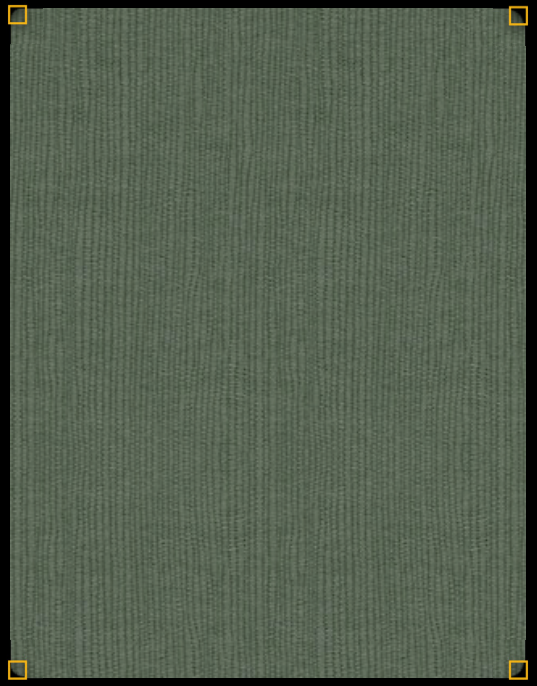


Smooth Animations

Reduce offscreen rendering

- Workaround: Decompose rounded corners into separate views

```
- (void)drawRect:(CGRect)rect {
  // …
  CGContextBeginPath(c);
  CGContextAddArc(c, r, r, r, M_PI, 3*M_PI_2, 0);
  CGContextAddLineToPoint(c, 0, 0);
  CGContextClosePath(c);
  CGContextClip(c);
  [[UIColor blackColor] setFill];
  UIRectFill(rect);
}
```



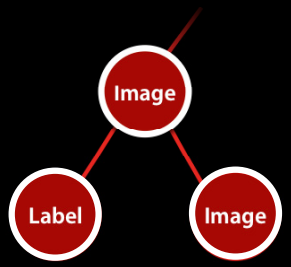
Smooth Animations

- Rendering each frame
- Reduce view blending
- Reduce offscreen rendering
- Dynamic flattening

Smooth Animations

Dynamic flattening

- Animating changes to a complex view hierarchy can be choppy
- Renders hierarchy on every frame
- Animations smoother with a flattened hierarchy…
- Now you can flatten without changing the view hierarchy using `shouldRasterize`



Smooth Animations

Dynamic flattening

- CALayer property `shouldRasterize`
- Turn on before animation
- Turn off after animation

```
view.transform = CGAffineTransformMakeScale(…);
view.layer.shouldRasterize = YES;
[self addSubview:view];
[UIView animateWithDuration:0.3
  animations:^{ view.transform = CGAffineTransformIdentity; }
  completion:^(BOOL finished) { view.layer.shouldRasterize = NO; }
];
```

Smooth Animations

Dynamic flattening

- Hint compositor to render view hierarchy offscreen and cache
- Offscreen rendering for good
- Can hurt more than help!
- Limited cache size
- Cache thrown away if anything in hierarchy changes

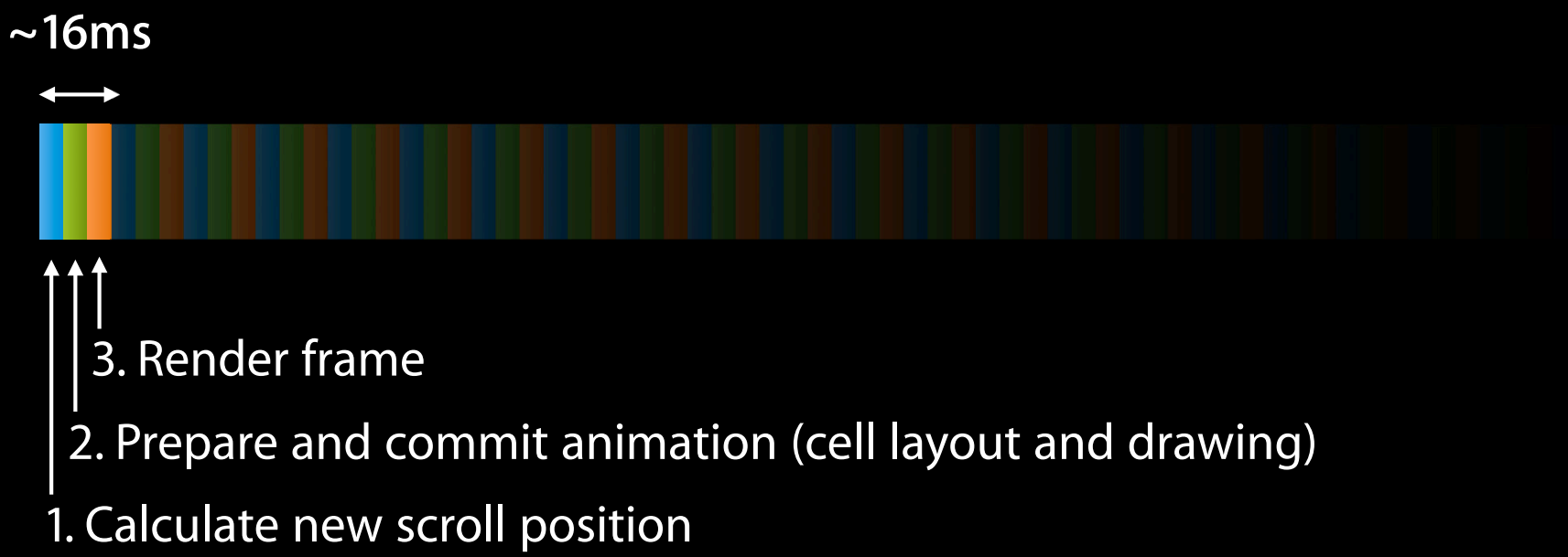
Smooth Animations

- Rendering each frame
- Reduce view blending
- Reduce offscreen rendering
- Dynamic flattening

Smooth Scrolling

- Each frame of scrolling is a little animation
  - Calculate new scroll position
  - Prepare and commit animation
  - Compositor renders new frame
- Animation advice applies
  - Prepare cells quickly
  - Render quickly

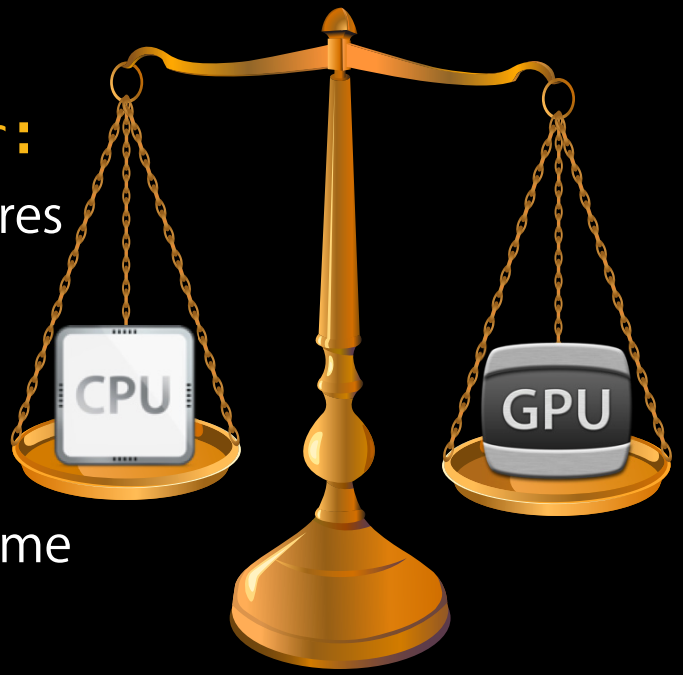
Smooth Scrolling



Smooth Scrolling

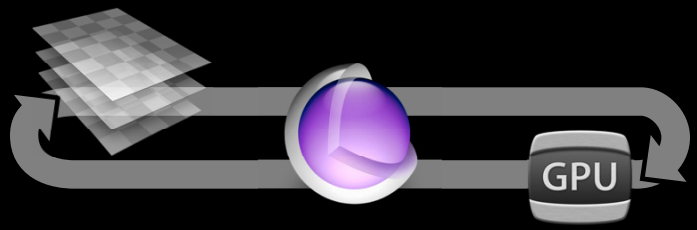
Prepare cells quickly

- Always reuse table cells
  - `dequeueReusableCellWithIdentifier:`
  - Save time creating objects and backing stores
  - Use unique identifiers for similar cells
  - Save time laying out views
- Flatten view hierarchy…to a point
  - Balance cell drawing time with rendering time
  - Consider flattening rasterized elements (text, paths, etc.), but let the renderer composite images
  - Measure and experiment



Smooth Scrolling

Render quickly



- Fewer pixels to render means smoother scrolling too
- Recall lessons from smooth animations
  - Simplify structure of view hierarchy
  - Remove unnecessary or invisible views
  - Reduce view blending
  - Reduce offscreen rendering
  - Dynamic flattening

Device Considerations

iPhone 3G
iPod touch (2008)

iPhone 3GS
iPod touch (2009)

iPad

iPhone 4

Animations and Scrolling

- Behind the scenes
- Responsive animations
- Smooth animations
- Smooth scrolling
- Device considerations

Responsiveness

Responsiveness

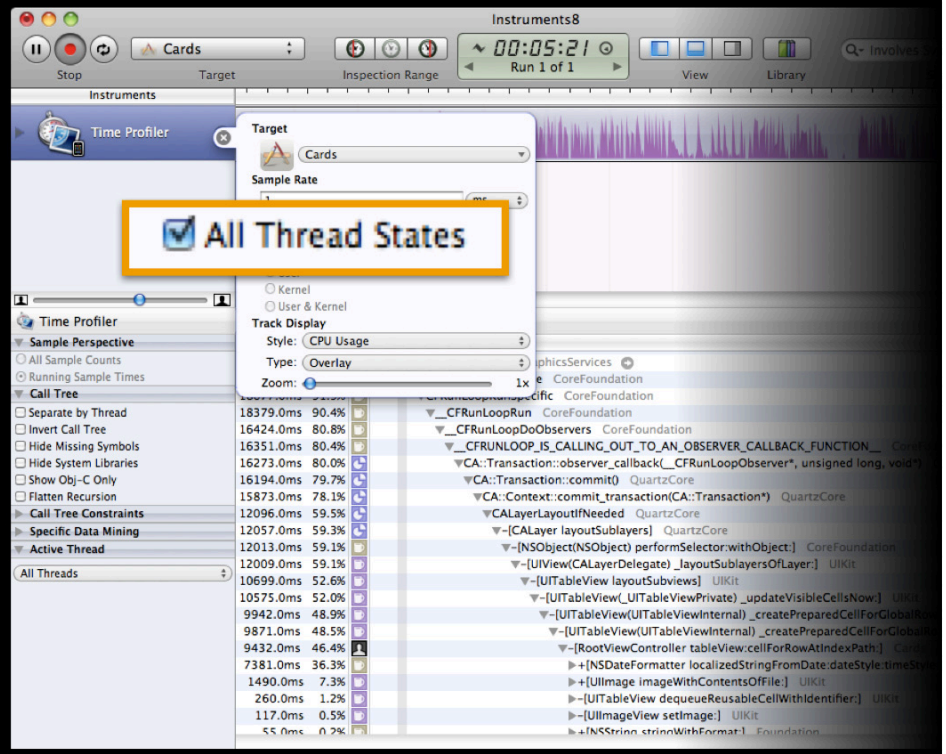
Don’t make your users wait

- Measuring
- Launch delays
- Interaction delays
- CPU optimization

Responsiveness

Measuring

- Time Profiler Instrument
  - New in iPhone SDK 4
- Great overview during scenario
  - Measure first
  - Find the problem
- Shows time spent on CPU
- "All Thread States" shows time spent blocking



Responsiveness

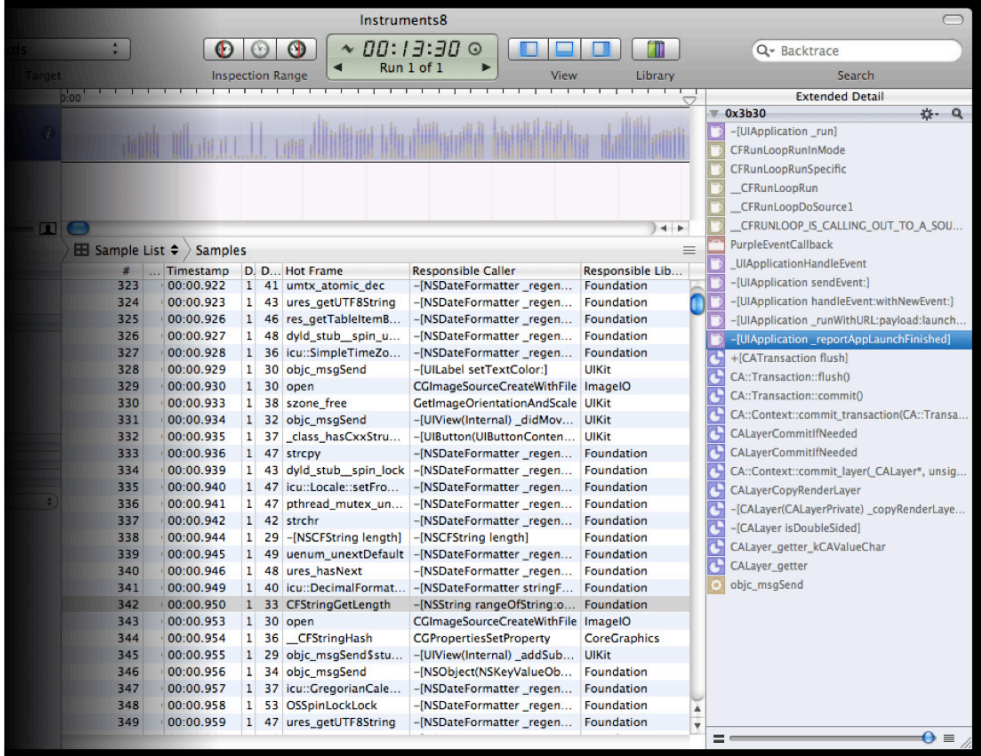
Measuring

- Measure problem scenarios
  - Baseline and improvements
- Simply time start and end using `CFAbsoluteTimeGetCurrent`
  - Wall clock time

```
NSTimeInterval start = CFAbsoluteTimeGetCurrent();
// ...
NSLog(@"It took %f seconds.", CFAbsoluteTimeGetCurrent() -
start);
```

Responsiveness

Launch delays

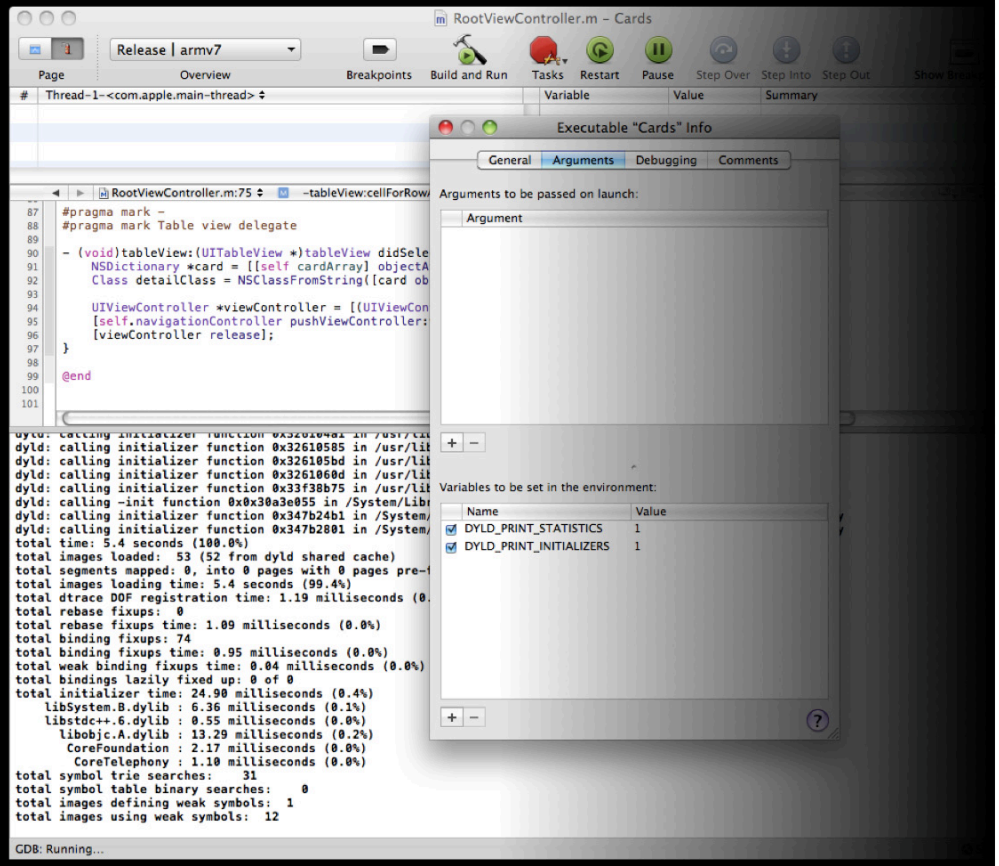


- Tricky to measure total launch
  - Time between start of `main` and `applicationDidFinishLaunching:`
- Launch timing using Time Profiler can be useful relative measurement
- Figure out what your application is doing on launch

Responsiveness

Launch delays

- Do only what's necessary on launch
  - Can you defer?
  - Could you do it on demand?
- Reduce number of linked frameworks
- When using libraries, look out for:
  - Static initializers
    - `DYLD_PRINT_STATISTICS=1`
    - `DYLD_PRINT_INITIALIZERS=1`
  - Weak exports (`WEAK_DEFINES`)
    - `otool -hv` (your binary)



Responsiveness

Interaction delays

- Do not block the main thread
- Long-running tasks should be spun off into background
- Factor into executable units of work so you can show progress
- Remember to make UI updates on the main thread
- Now even easier with `NSOperationQueue` and blocks…

Responsiveness

Background tasks


```
NSOperationQueue *q = [[NSOperationQueue alloc] init];
[q addOperationWithBlock:^{
  UIGraphicsBeginImageContextWithOptions(rect.size, YES, 0.0);
  …
  UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
  UIGraphicsEndImageContext();
  [[NSOperationQueue mainQueue] addOperationWithBlock:^{
    UIImageView *imageView = [[UIImageView alloc]
    initWithImage:image];
    [window addSubview:imageView];
    [imageView release];
  }];
}];
[q release];
```

Responsiveness

Make URL requests asynchronously

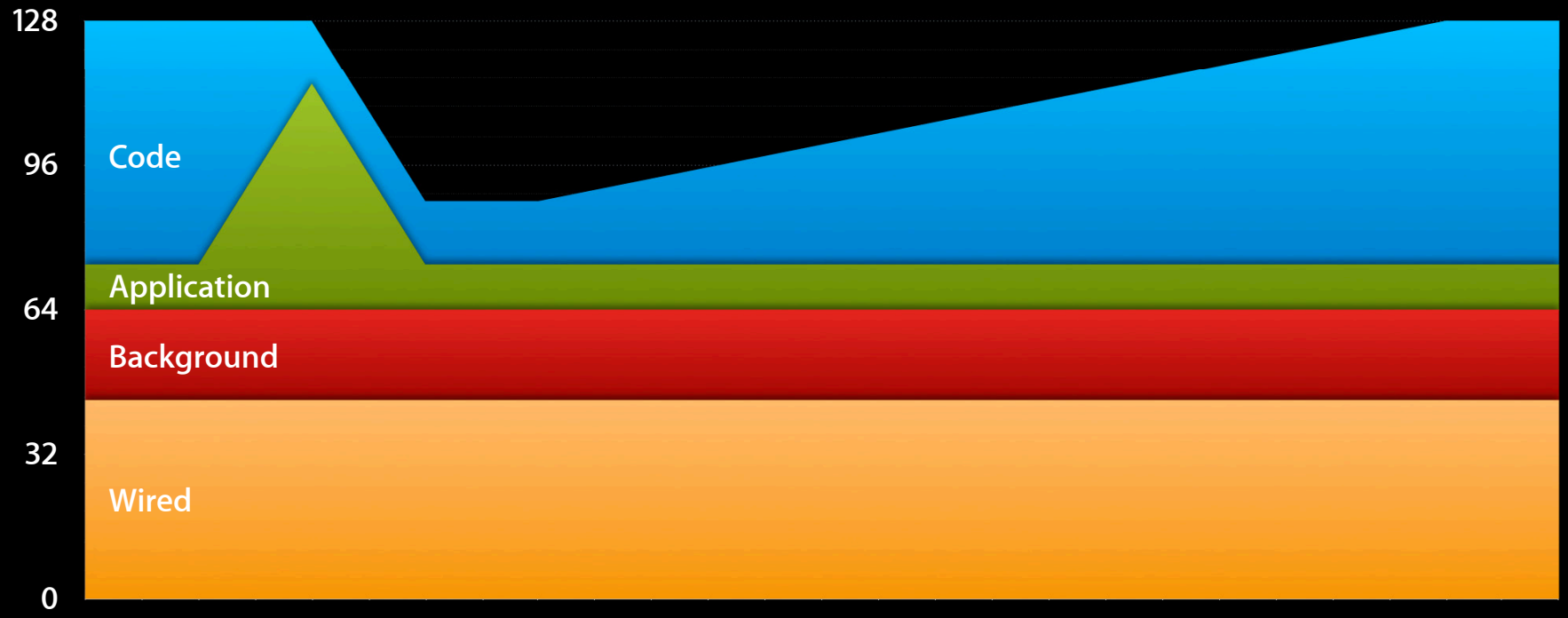
```
d = [NSURLConnection sendSynchronousRequest:[NSURLRequest
 requestWithURL:url] returningResponse:&response error:&error];
```

```
d = [NSMutableData data];
c = [NSURLConnection connectionWithRequest:[NSURLRequest
 requestWithURL:url] delegate:self]];

- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)
 data{ [d appendData:data]; }

- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError
 *)error { /* Handle error */ }

- (void)connectionDidFinishLoading:(NSURLConnection *)connection {
 /* Use downloaded data in d... */ }
```

Responsiveness

Spikes in memory usage may cause delays

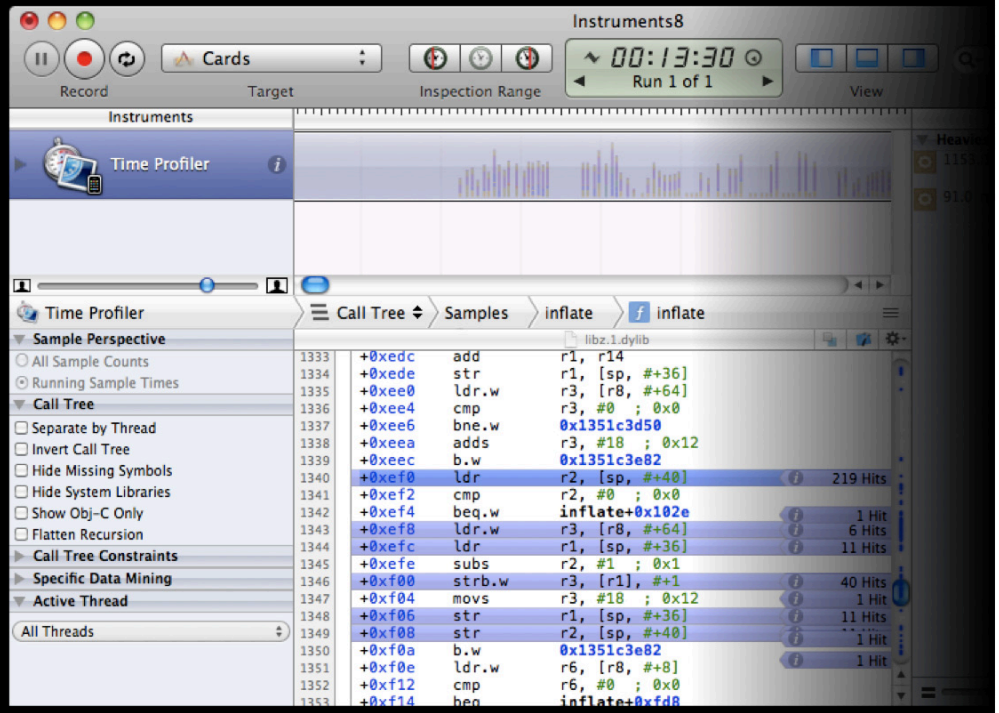
- To accommodate high memory usage, code is evicted
- Code must be read back in from storage to proceed



Responsiveness

CPU optimization

- Use Time Profiler to find hot spots
- Vector processing can speed up CPU-bound tasks
  - Process several elements at once
- Easier using Accelerate framework



Responsiveness

Don’t make your users wait

- Measuring
- Launch delays
- Interaction delays
- CPU optimization

Power and Battery Life

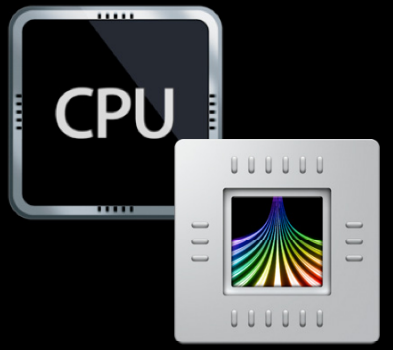
Peter Handel
iOS Power

Power Consumption

What we'll cover

Radio

Core Location

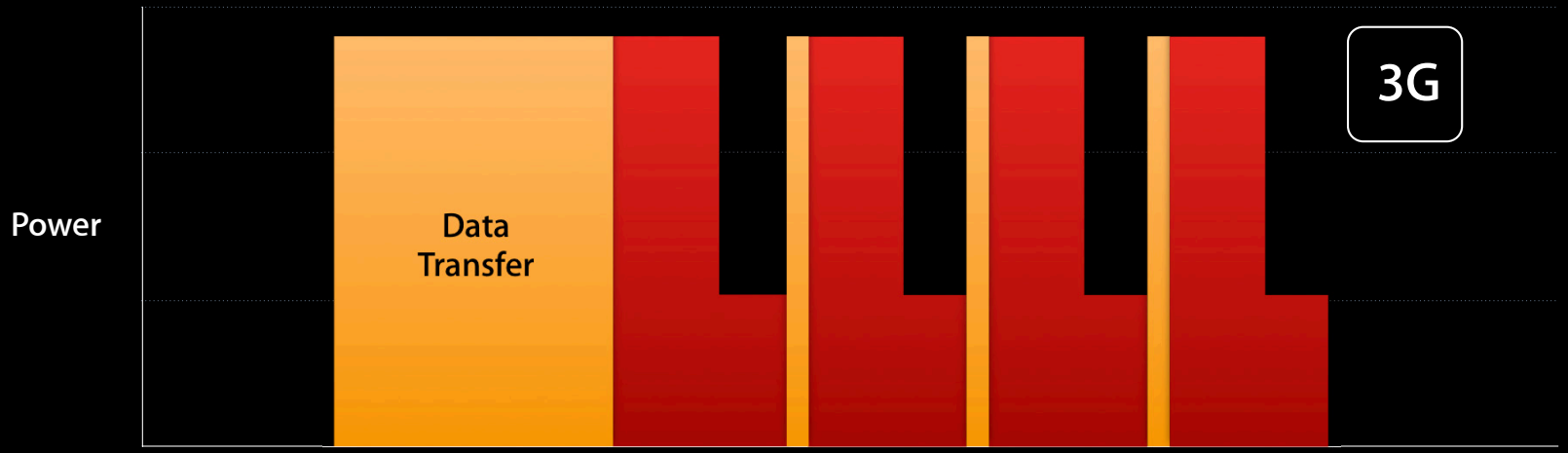


CPU/GPU

Power Consumption

Radios: 3G

- Very expensive to send data
- 3G networks require phones stay in high-power state for a few seconds after last packet is sent or received

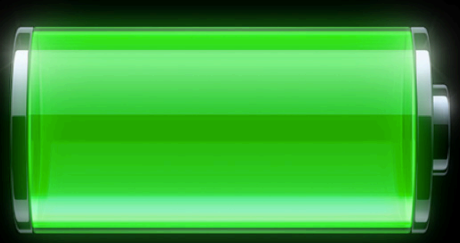


Power Consumption

Radios: 3G

- Optimizing 3G transmissions
  - Use Instruments—Activity Monitor
  - Coalesce data into large chunks, rather than thin stream
    - Do not poll: Use Apple Push Notification service
  - Minimize amount of data transmitted—Use compact data formats!
  - Be careful reusing legacy or third party code!
    - They often assume ethernet

- Poor networking frequently causes drain!
- 3G radio chip: Let that chip idle!



Power Consumption

Radios: Wi-Fi

- Wi-Fi uses less power than 3G
  - …but it still uses a fair bit!
- Wi-Fi radios idle immediately after transmission

- Detect when you're on Wi-Fi versus cell
  - Example: more extreme data coalescence over cell

From SystemConfiguration:

```
if (flags & kSCNetworkReachabilityFlagsIsWWAN) {
```

Power Consumption

Radios: Wi-Fi vs. 3G vs. 2G

- 2G
  - Power consumption is between 3G and Wi-Fi
  - 2G network allows radios to idle immediately after data transfer
  - …much slower!

Power Consumption

What we'll cover

- Radio
- Core Location
- CPU/GPU

Power Consumption

Core Location

- Lots of apps use Core Location
- Lets you know where device is to varying degrees of accuracy


```
CLLocationManager *locationManager = [[CLLocationManager alloc] init];
locationManager.desiredAccuracy = kCLLocationAccuracyHundredMeters;
locationManager.distanceFilter = 100;
[locationManager startUpdatingLocation];
[locationManager stopUpdatingLocation];
```

Power Consumption

Core Location

- 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.distanceFilter = 100;
[locationManager startUpdatingLocation];
[locationManager stopUpdatingLocation];
```

Power Consumption

Core Location

- `distanceFilter`—dictates how often you receive location changed notifications
  - Set it appropriately
  - The default (`kCLDistanceFilterNone`) receives all movement updates
    - Can result in unnecessary events = higher CPU usage

```
CLLocationManager *locationManager = [[CLLocationManager alloc] init];
locationManager.desiredAccuracy = kCLLocationAccuracyHundredMeters;
locationManager.distanceFilter = 100;
[locationManager startUpdatingLocation];
[locationManager stopUpdatingLocation];
```

Power Consumption

Core Location

- Call `stopUpdatingLocation` after reaching desired accuracy
- CoreLocation manages GPS power for you
  - …so call `stopUpdatingLocation` as soon as you're finished
- GPS chip: Let that chip idle!

```
CLLocationManager *locationManager = [[CLLocationManager alloc] init];
locationManager.desiredAccuracy = kCLLocationAccuracyHundredMeters;
locationManager.distanceFilter = 100;
[locationManager startUpdatingLocation];
[locationManager stopUpdatingLocation];
```

Power Consumption

Core Motion

- Same is true for Core Motion
  - After `start{Accelerometer,DeviceMotion,Gyro}Updates`, be sure to call `stop{Accelerometer,DeviceMotion,Gyro}Updates`
  - If your app is backgrounded, turn off the sensors

```
CMMotionManager *motionManager = [[CMMotionManager alloc] init];
[motionManager startDeviceMotionUpdates];
...
[motionManager stopDeviceMotionUpdates];
```

Power Consumption

Core Location

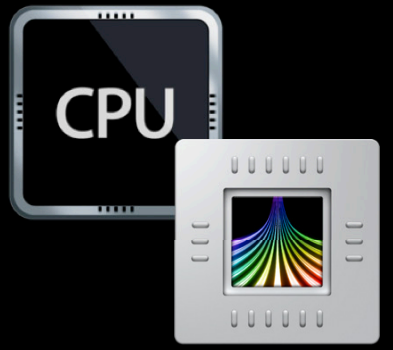
- Use new iOS 4 API
  - Significant location changed (`startMonitoringSignificantLocationChanges`)
  - Region monitoring (`startMonitoringForRegion`)

Power Consumption

What we'll cover

Radio

Core Location

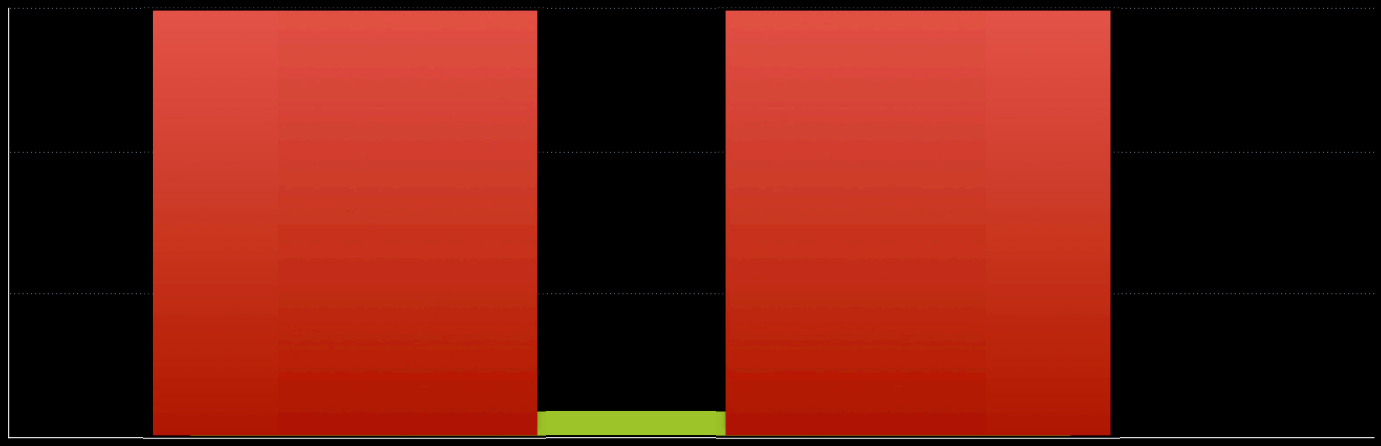


CPU/GPU

Power Consumption

CPU: Performance

- Improving performance results in better battery life
  - Fast code = less CPU time = less power
  - CPU: Let that chip idle!



Power Consumption

CPU: Polling vs. events

- iOS 4 is event-based
  - You might want to poll a condition—don't!
- Subscribe to events whenever possible
- If you must poll, use a timer with a low frequency

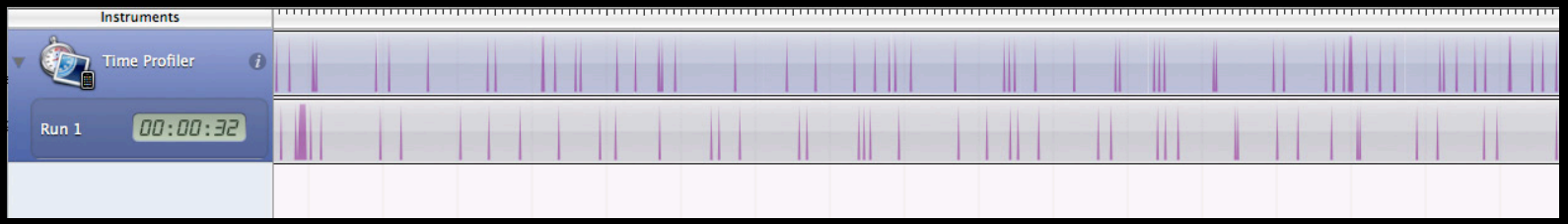
- Accelerometer: Use Shake API (UIResponder) rather than UIAccelerometer

```
- (void)motionEnded:(UIEventSubtype)motion withEvent:(UIEvent *)
event {
      if (motion == UIEventSubtypeMotionShake) {
```

Power Consumption

CPU: Be bursty

- Be bursty! Consolidate CPU usage into short bursts
  - Allows CPU to enter idle state
  - May require code restructuring or different algorithm
  - Use Instruments: Time Profiler to check CPU activity level
  - Audio playback schedules its work in bursts
    - Allows CPU to idle for long periods between work



Power Consumption

CPU: Procrastinate

- Delay work, possibly forever?
  - Example: When should a game write its state?

Power Consumption

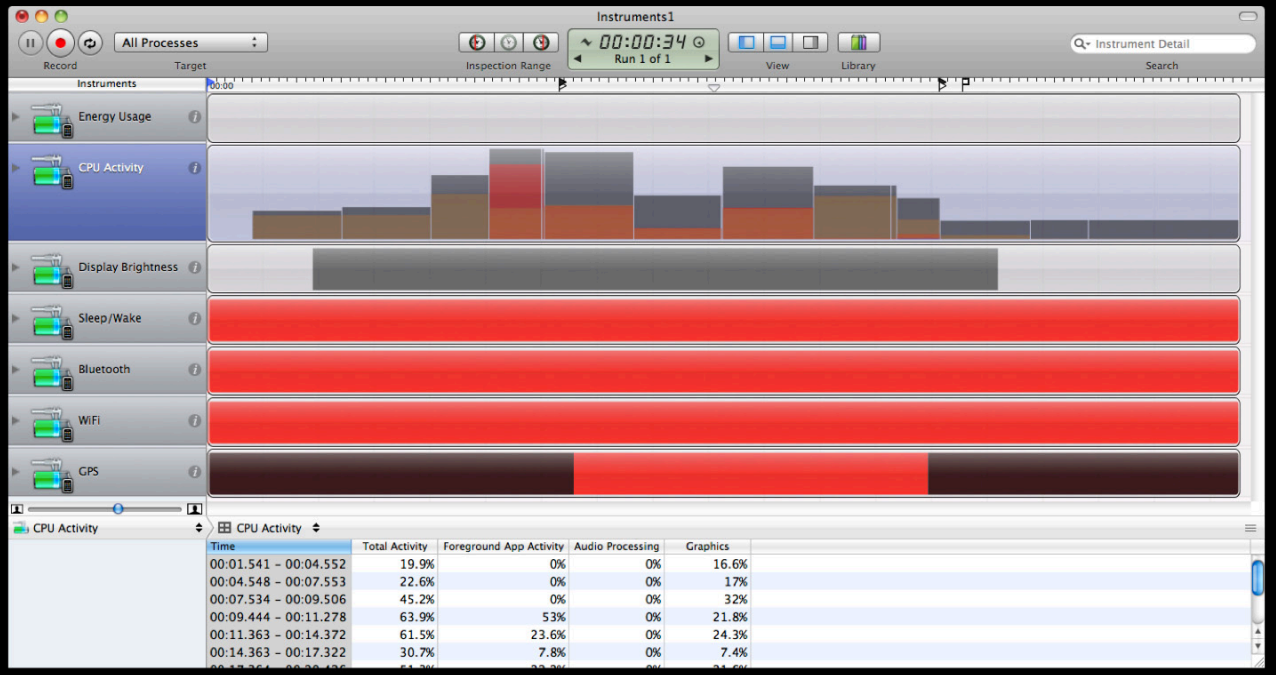
GPU

- When using OpenGL ES/GPU
  - Pick a fixed frame rate—30 fps—using `CADisplayLink` rather than `NSTimer`
    - Minimizes appearance of dropped frames—frame limiting!
  - If frame hasn't changed, don't redraw
    - Example: chess game

Power Consumption

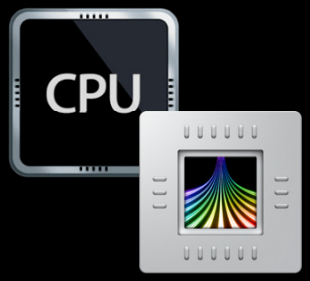
Tools

- Use Instruments—Energy diagnostics tool (see Session 309)



Power Consumption

Summary



- Radios
  - Data transmission is expensive
  - Coalesce/compress data
- Core Location
  - Use least amount of accuracy you need
  - Unsubscribe from notifications when finished
- CPU/GPU
  - Optimizing for performance = optimizing for power
  - Be bursty, procrastinate
  - GPU: Use fixed frame rate (30fps), don't unnecessarily redraw frames
- Let those chips idle!

Summary

- Use knowledge about system to come up with creative solutions
- Always measure baseline and changes
- Fewer pixels to render means smoother animations
- Prepare and render quickly for smoother scrolling
- Don’t block the main thread
- Let those chips idle

Related Sessions

Advanced Performance Optimization on iPhone OS, Part 2	Mission Friday 11:30AM
Optimizing Core Data Performance on iPhone OS	Presidio Thursday 4:30PM

The Accelerate Framework for iPhone OS	Nob Hill Tuesday 11:30AM
Advanced Performance Analysis with Instruments	Mission Thursday 9:00AM
Performance Optimization on iPhone OS	Presidio Thursday 2:00PM
Core Animation in Practice, Part 2	Nob Hill Thursday 2:00PM

Labs

Core Animation Lab	Graphics and Media Lab D Thursday 3:15PM
Animation Lab	Application Frameworks Lab C Thursday 4:30PM
iPhone OS Performance Lab	Developer Tools Lab A Thursday 4:30PM
iPhone OS Performance Lab	Developer Tools Lab A Friday 9:00AM

More Information

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

Bill Dudney
Application Frameworks Evangelist
dudney@apple.com

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


WWDC10