iOS App Performance

Graphics and Animations

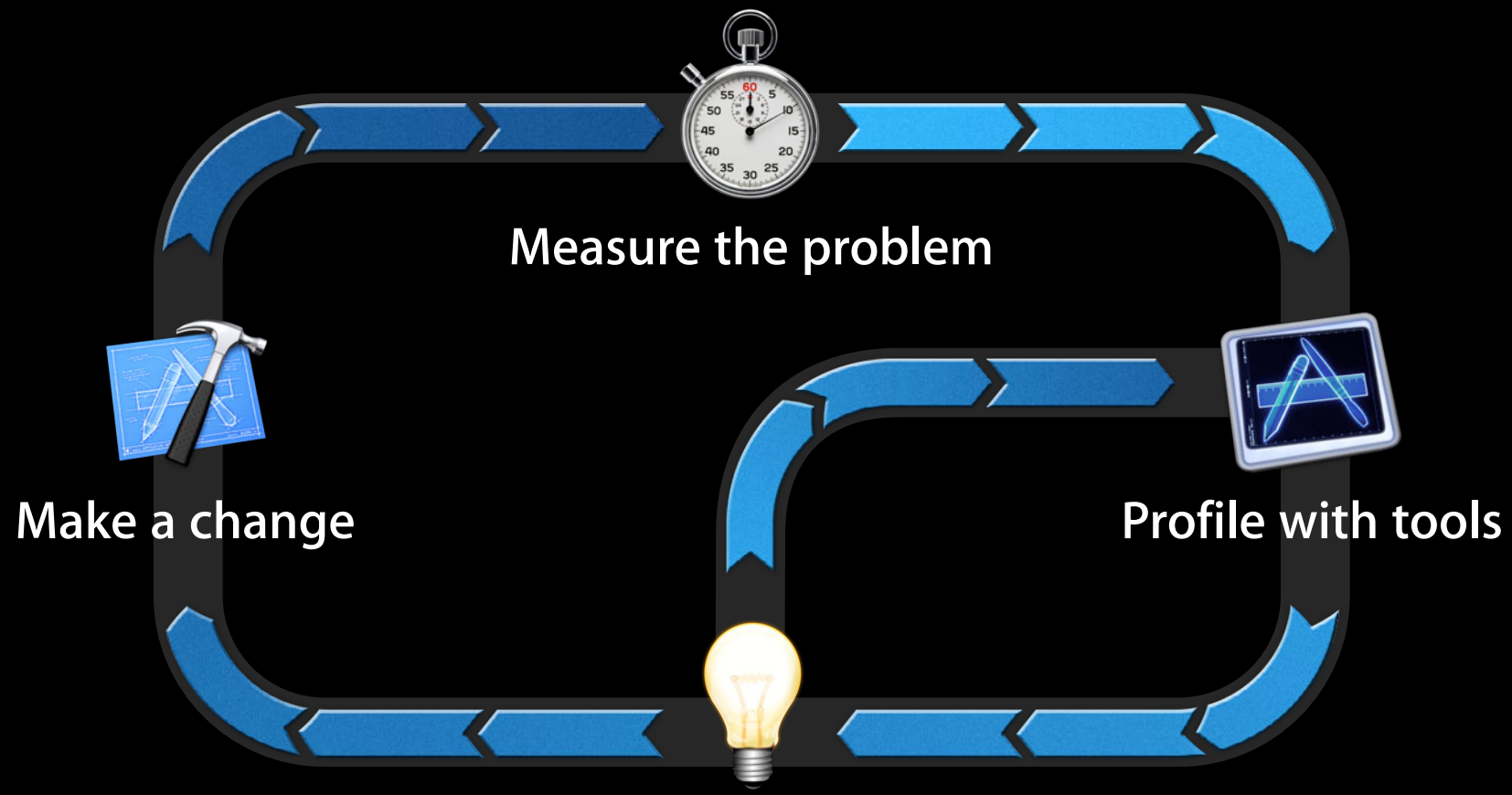
Session 238

Dan Crosby
iOS Performance Team

Introduction

- Introduction to Animations
- Responsive Animations
- Smooth Animations
- Scrolling

Performance Bug Workflow

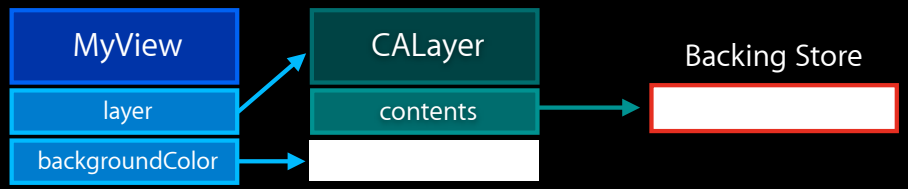


Graphics and Animations

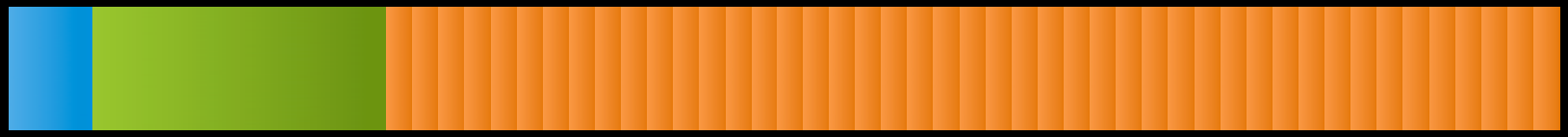
- Introduction to Animations
- Responsive Animations
- Smooth Animations
- Scrolling

Views, Layers, and Animations

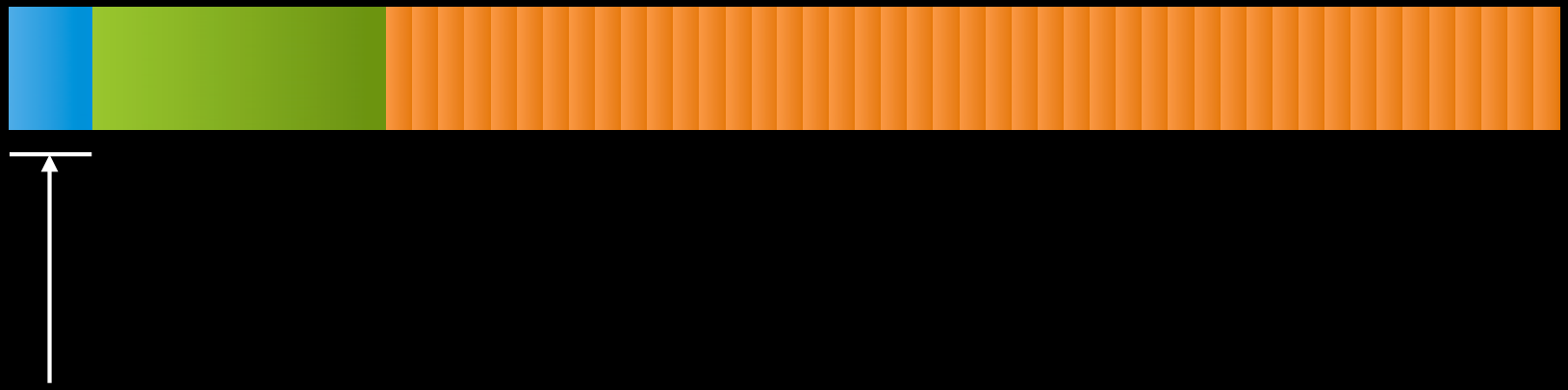
- Every UIView is backed by a CALayer
- View layout is really layer layout
- `-drawRect:` draws into a CALayer backing store
- Layer properties and animations handled by render server
- Changes happen in `CA::Transaction::commit()`



Stages of an Animation

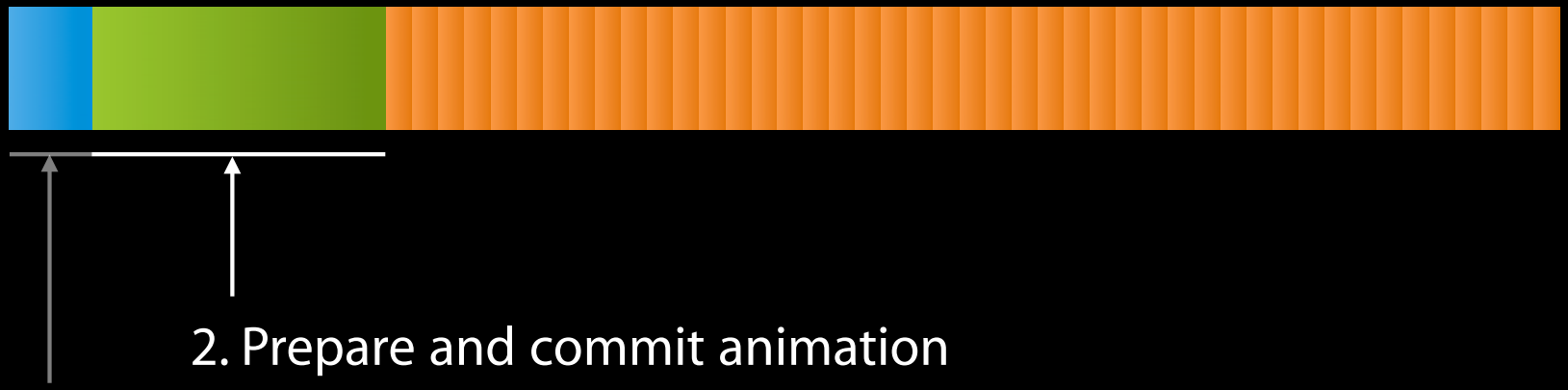


Stages of an Animation

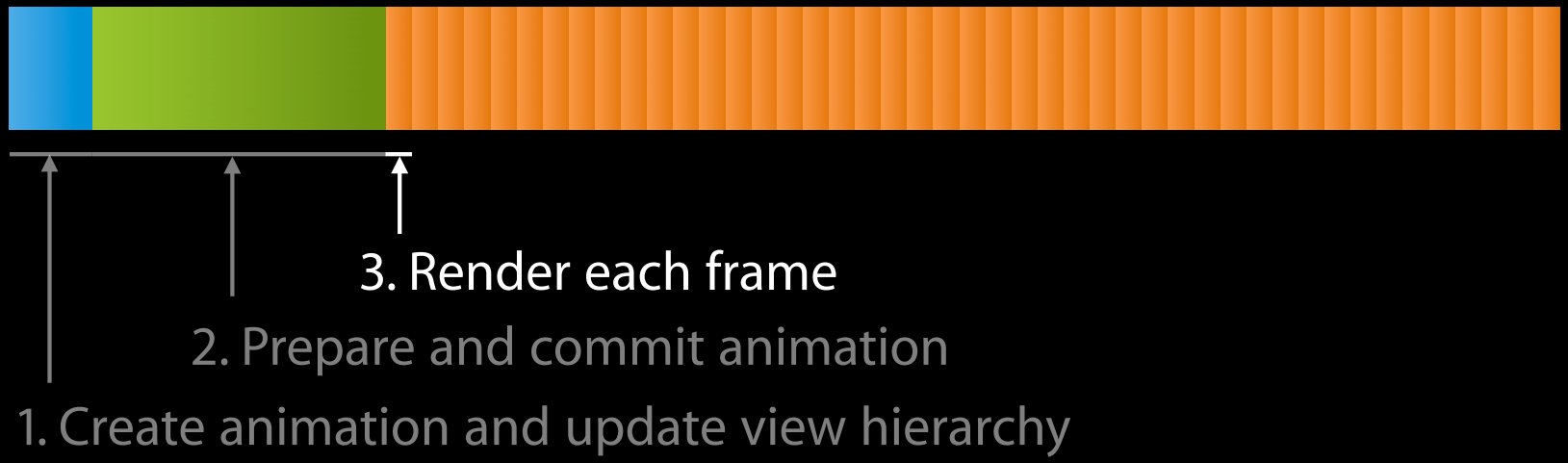


1. Create animation and update view hierarchy

Stages of an Animation



Stages of an Animation



Creating an Animation

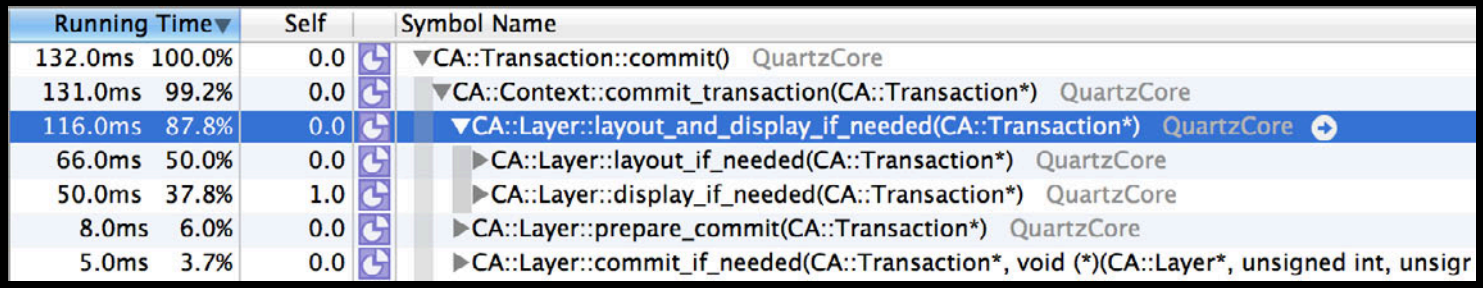
```
view = [[InsideView alloc] initWithFrame:frame];
view.transform = CGAffineTransformMakeScale(0.01, 0.01);
[UIView animateWithDuration:0.5 animations:^{
  [self addSubview:view];
  view.transform = CGAffineTransformIdentity;
}];
```

Creating an Animation

```
view = [[InsideView alloc] initWithFrame:frame];
view.transform = CGAffineTransformMakeScale(0.01, 0.01);
[UIView animateWithDuration:0.5 animations:^{
  [self addSubview:view];
  view.transform = CGAffineTransformIdentity;
}];
```

Preparing the Animation

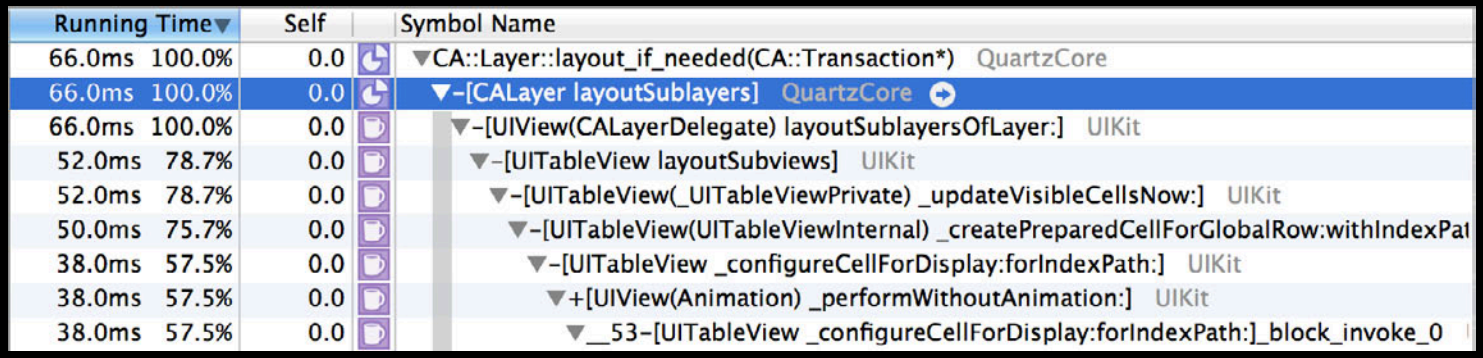
- Layout sets up the views
- Display draws the views
- Prepare does CoreAnimation work
- Commit packages up layers and sends them to render server



Running Time		Self	Symbol Name
132.0ms	100.0%	0.0	▼CA::Transaction::commit() QuartzCore
131.0ms	99.2%	0.0	▼CA::Context::commit_transaction(CA::Transaction*) QuartzCore
116.0ms	87.8%	0.0	▼CA::Layer::layout_and_display_if_needed(CA::Transaction*) QuartzCore
66.0ms	50.0%	0.0	▶CA::Layer::layout_if_needed(CA::Transaction*) QuartzCore
50.0ms	37.8%	1.0	▶CA::Layer::display_if_needed(CA::Transaction*) QuartzCore
8.0ms	6.0%	0.0	▶CA::Layer::prepare_commit(CA::Transaction*) QuartzCore
5.0ms	3.7%	0.0	▶CA::Layer::commit_if_needed(CA::Transaction*, void (*)(CA::Layer*, unsigned int, unsigr

Layout

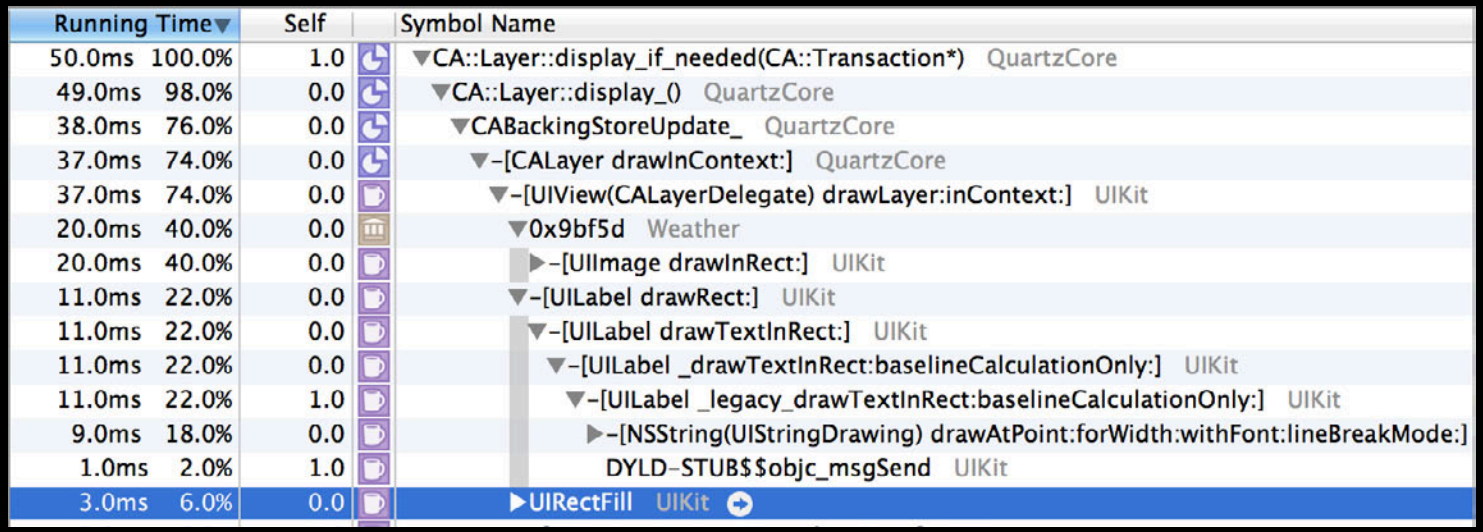
- Often has expensive view creation and layer graph management
- May need to do expensive data lookup
- May block on I/O or work done in another thread or process
- CPU (and sometimes I/O) bound



Running Time		Self	Symbol Name
66.0ms	100.0%	0.0	▼CA::Layer::layout_if_needed(CA::Transaction*) QuartzCore
66.0ms	100.0%	0.0	▼-[CALayer layoutSublayers] QuartzCore
66.0ms	100.0%	0.0	▼-[UIView(CALayerDelegate) layoutSublayersOfLayer:] UIKit
52.0ms	78.7%	0.0	▼-[UITableView layoutSubviews] UIKit
52.0ms	78.7%	0.0	▼-[UITableView(_UITableViewPrivate) _updateVisibleCellsNow:] UIKit
50.0ms	75.7%	0.0	▼-[UITableView(UITableViewInternal) _createPreparedCellForGlobalRow:withIndexPat
38.0ms	57.5%	0.0	▼-[UITableView _configureCellForDisplay:forIndexPath:] UIKit
38.0ms	57.5%	0.0	▼+[UIView(Animation) _performWithoutAnimation:] UIKit
38.0ms	57.5%	0.0	▼__53-[UITableView _configureCellForDisplay:forIndexPath:]_block_invoke_0

Display

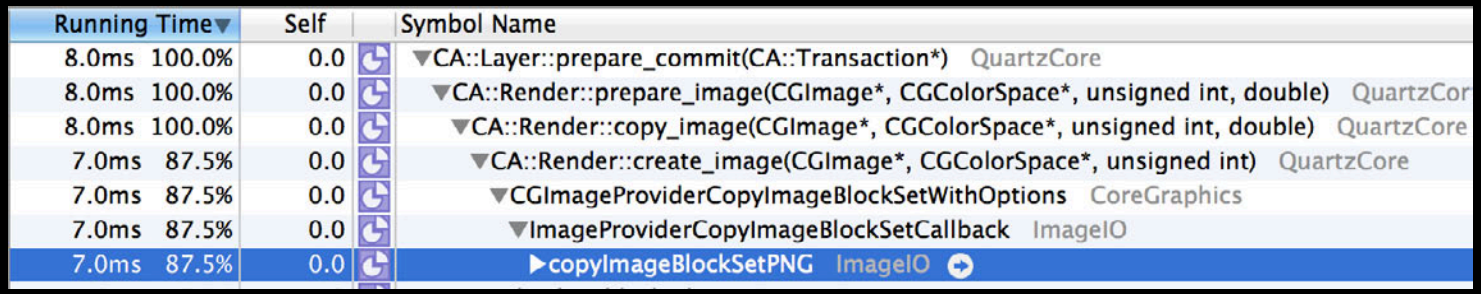
- `-drawRect:` for any class where you've overridden it
- String drawing or other expensive drawing
- Usually CPU bound



Running Time		Self	Symbol Name
50.0ms	100.0%	1.0	▼CA::Layer::display_if_needed(CA::Transaction*) QuartzCore
49.0ms	98.0%	0.0	▼CA::Layer::display_() QuartzCore
38.0ms	76.0%	0.0	▼CABackingStoreUpdate_ QuartzCore
37.0ms	74.0%	0.0	▼-[CALayer drawInContext:] QuartzCore
37.0ms	74.0%	0.0	▼-[UIView(CALayerDelegate) drawLayer:inContext:] UIKit
20.0ms	40.0%	0.0	▼0x9bf5d Weather
20.0ms	40.0%	0.0	▶-[UIImage drawInRect:] UIKit
11.0ms	22.0%	0.0	▼-[UILabel drawRect:] UIKit
11.0ms	22.0%	0.0	▼-[UILabel drawTextInRect:] UIKit
11.0ms	22.0%	0.0	▼-[UILabel _drawTextInRect:baselineCalculationOnly:] UIKit
11.0ms	22.0%	1.0	▼-[UILabel _legacy_drawTextInRect:baselineCalculationOnly:] UIKit
9.0ms	18.0%	0.0	▶-[NSString(UIStringDrawing) drawAtPoint:forWidth:withFont:lineBreakMode:]
1.0ms	2.0%	1.0	DYLD-STUB$$objc_msgSend UIKit
3.0ms	6.0%	0.0	▶UIRectFill UIKit

Prepare

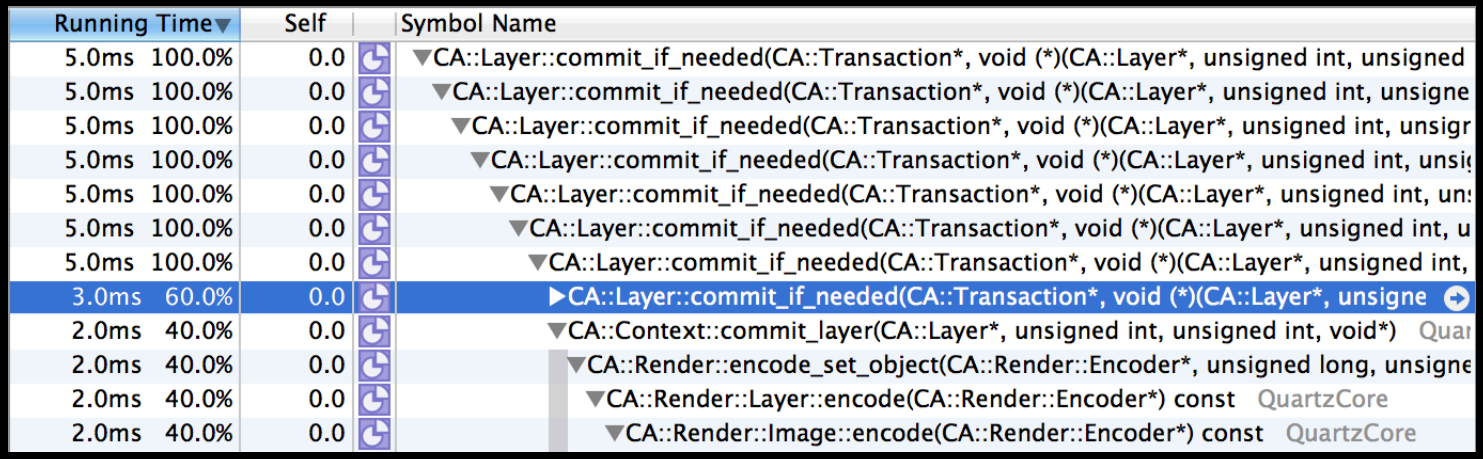
- CA does non-`drawRect:` work like image decoding here
- Watch out for work dispatched to another thread



Running Time ▼		Self		Symbol Name
8.0ms	100.0%	0.0		▼CA::Layer::prepare_commit(CA::Transaction*) QuartzCore
8.0ms	100.0%	0.0		▼CA::Render::prepare_image(CGImage*, CGColorSpace*, unsigned int, double) QuartzCor
8.0ms	100.0%	0.0		▼CA::Render::copy_image(CGImage*, CGColorSpace*, unsigned int, double) QuartzCore
7.0ms	87.5%	0.0		▼CA::Render::create_image(CGImage*, CGColorSpace*, unsigned int) QuartzCore
7.0ms	87.5%	0.0		▼CGImageProviderCopyImageBlockSetWithOptions CoreGraphics
7.0ms	87.5%	0.0		▼ImageProviderCopyImageBlockSetCallback ImageIO
7.0ms	87.5%	0.0		▶copyImageBlockSetPNG ImageIO

Commit

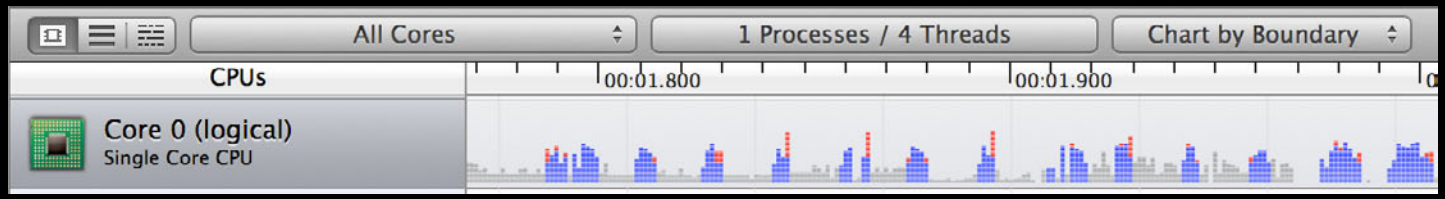
- Layers are packaged up and sent to render server over IPC
- May be expensive if layer tree is especially complex



Running Time ▼		Self		Symbol Name
5.0ms	100.0%	0.0		▼CA::Layer::commit_if_needed(CA::Transaction*, void (*)(CA::Layer*, unsigned int, unsigned
5.0ms	100.0%	0.0		▼CA::Layer::commit_if_needed(CA::Transaction*, void (*)(CA::Layer*, unsigned int, unsigne
5.0ms	100.0%	0.0		▼CA::Layer::commit_if_needed(CA::Transaction*, void (*)(CA::Layer*, unsigned int, unsigr
5.0ms	100.0%	0.0		▼CA::Layer::commit_if_needed(CA::Transaction*, void (*)(CA::Layer*, unsigned int, unsi(
5.0ms	100.0%	0.0		▼CA::Layer::commit_if_needed(CA::Transaction*, void (*)(CA::Layer*, unsigned int, uns
5.0ms	100.0%	0.0		▼CA::Layer::commit_if_needed(CA::Transaction*, void (*)(CA::Layer*, unsigned int, u
5.0ms	100.0%	0.0		▼CA::Layer::commit_if_needed(CA::Transaction*, void (*)(CA::Layer*, unsigned int,
3.0ms	60.0%	0.0		▶CA::Layer::commit_if_needed(CA::Transaction*, void (*)(CA::Layer*, unsigne
2.0ms	40.0%	0.0		▼CA::Context::commit_layer(CA::Layer*, unsigned int, unsigned int, void*) Quar
2.0ms	40.0%	0.0		▼CA::Render::encode_set_object(CA::Render::Encoder*, unsigned long, unsigne
2.0ms	40.0%	0.0		▼CA::Render::Layer::encode(CA::Render::Encoder*) const QuartzCore
2.0ms	40.0%	0.0		▼CA::Render::Image::encode(CA::Render::Encoder*) const QuartzCore

Rendering the Animation

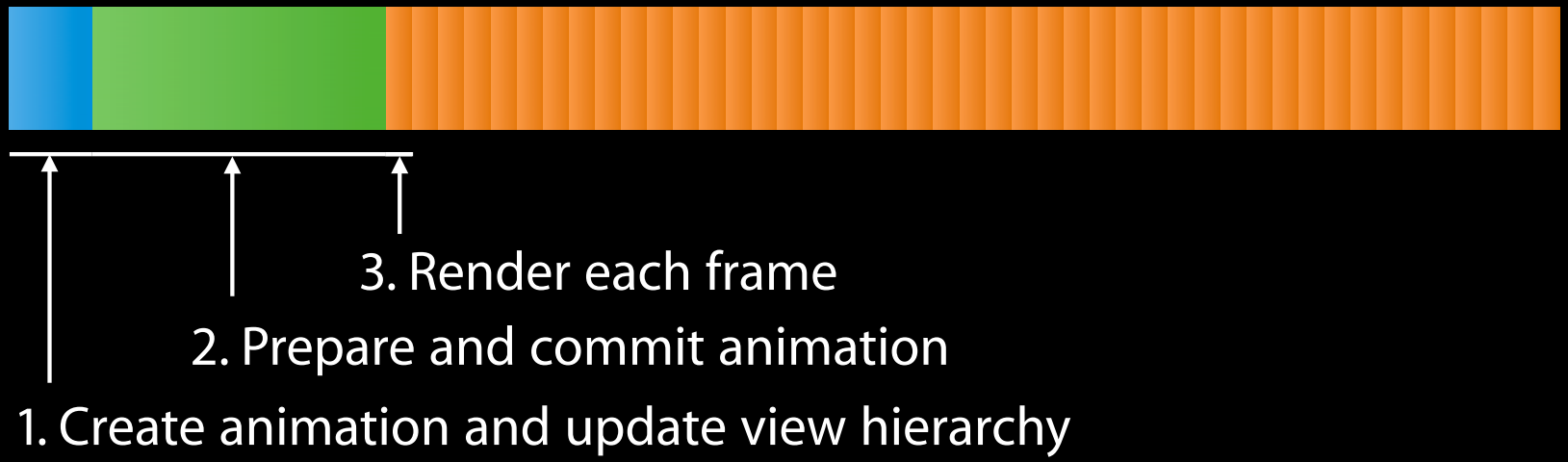
- Usually GPU bound
- Render server CPU work contends with app work



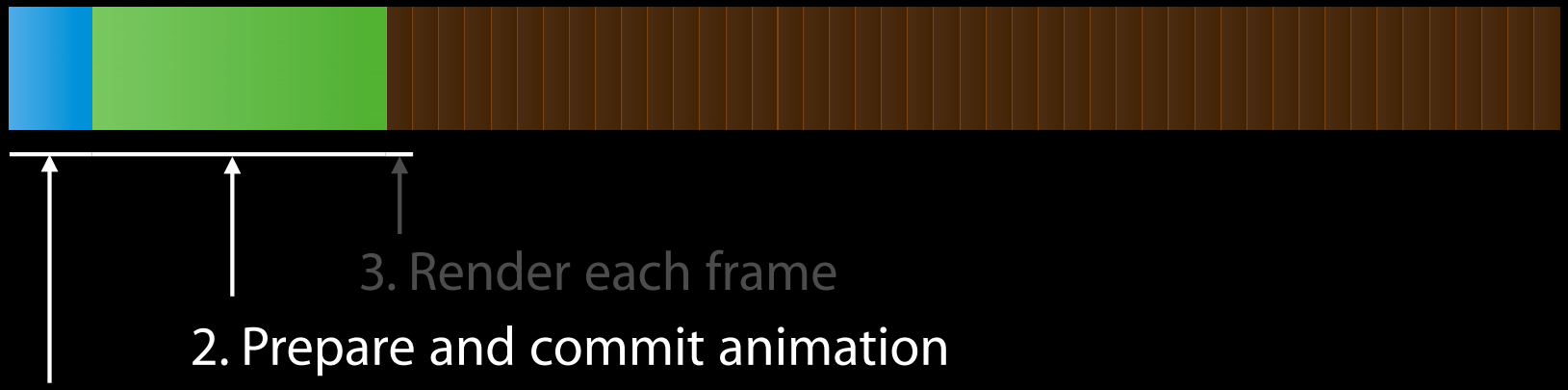
Graphics and Animations

- Introduction to Animations
- Responsive Animations
- Smooth Animations
- Scrolling

Responsiveness



Responsiveness



What Delays an Animation

Layout

- Slow layout can be caused by
  - Complex hierarchy
  - Lazy construction of views
  - Database or flash storage access to populate views

What Delays an Animation

Drawing

- Slow drawing can be
  - `-drawRect:`
  - String drawing
  - Image decoding

Improving Responsiveness

- Do less setup
- Reduce drawing
- Be smart with images
- `-drawsAsynchronously`
- Speculative preparation

Improving Responsiveness

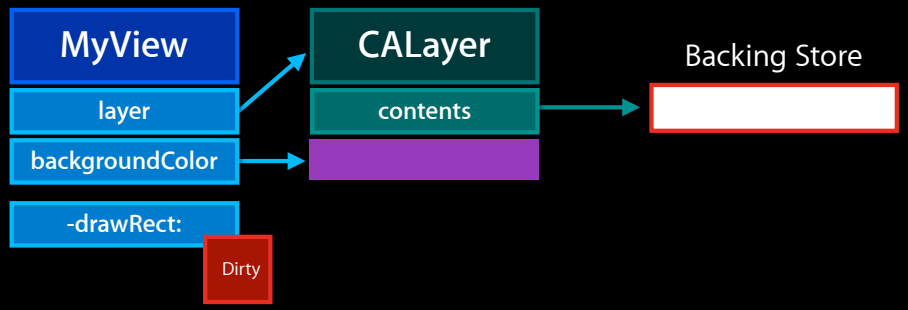
Do less setup

- Try to avoid CPU-heavy or blocking operations during layout
- Use in-memory caches
- Ensure database has appropriate indices for perf-critical lookups
- Always reuse cells and views whenever possible

Improving Responsiveness

Reduce drawing

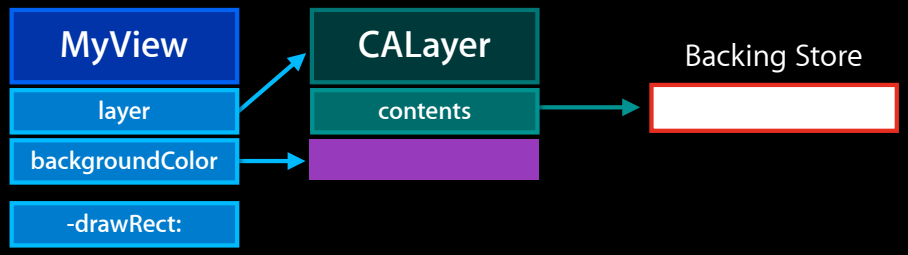
- Only call `-setNeedsDisplay` when needed
- Avoid overriding `-drawRect:`
- Implement smart `-drawRect:` and use `-setNeedsDisplayInRect:`
- When possible, use CALayer properties instead



Improving Responsiveness

Reduce drawing

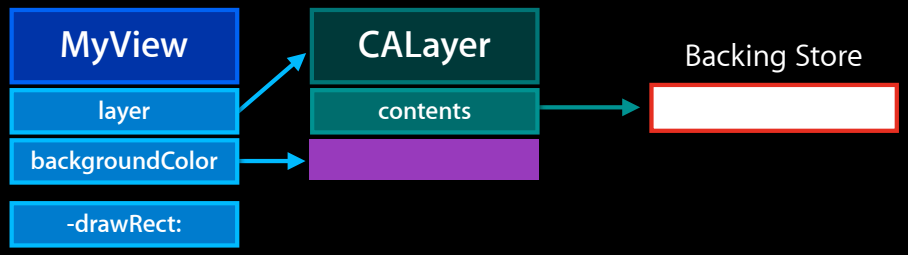
- Only call `-setNeedsDisplay` when needed
- Avoid overriding `-drawRect:`
- Implement smart `-drawRect:` and use `-setNeedsDisplayInRect:`
- When possible, use CALayer properties instead



Improving Responsiveness

Reduce drawing

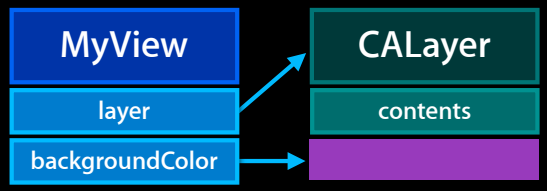
- Only call `-setNeedsDisplay` when needed
- Avoid overriding `-drawRect:`



Improving Responsiveness

Reduce drawing

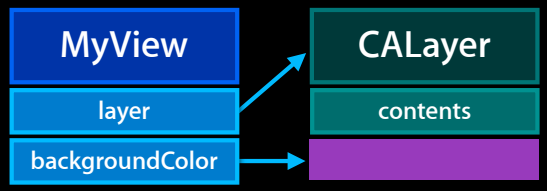
- Only call `-setNeedsDisplay` when needed
- Avoid overriding `-drawRect:`



Improving Responsiveness

Reduce drawing

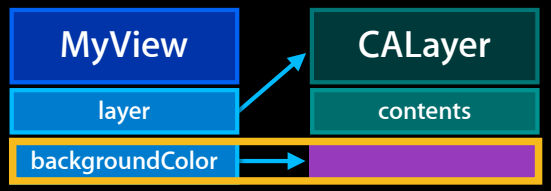
- Only call `-setNeedsDisplay` when needed
- Avoid overriding `-drawRect:`
- Implement smart `-drawRect:` and use `-setNeedsDisplayInRect:`



Improving Responsiveness

Reduce drawing

- Only call `-setNeedsDisplay` when needed
- Avoid overriding `-drawRect:`
- Implement smart `-drawRect:` and use `-setNeedsDisplayInRect:`
- When possible, use CALayer properties instead



Improving Responsiveness

Reduce drawing—Using CALayer properties

```
-(void)drawRect:(CGRect)rect {
    [[UIColor redColor] setFill];
    UIRectFill([self bounds]);
}
```

```
[myView setBackgroundColor:[UIColor redColor]];
```

Improving Responsiveness

Reduce drawing—Using CALayer properties

```
-(void)drawRect:(CGRect)rect {
    [[UIColor redColor] setFill];
    UIRectFill([self bounds]);
}
```

```
[myView setBackgroundColor:[UIColor redColor]];
```

Improving Responsiveness

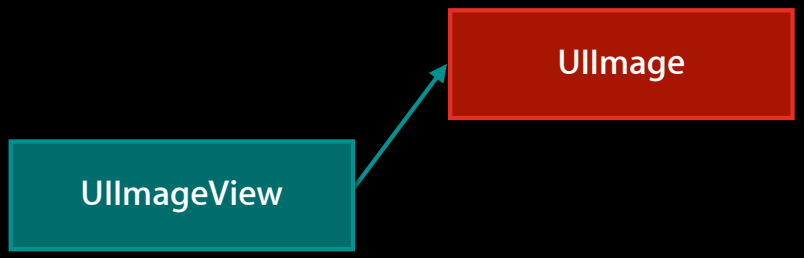
Reduce drawing—Using CALayer properties

```
-(void)drawRect:(CGRect)rect {
    [[UIColor redColor] setFill];
    UIRectFill([self bounds]);
}
```

```
[myView setBackgroundColor:[UIColor redColor]];
```

Improving Responsiveness

Images and Layers



Improving Responsiveness

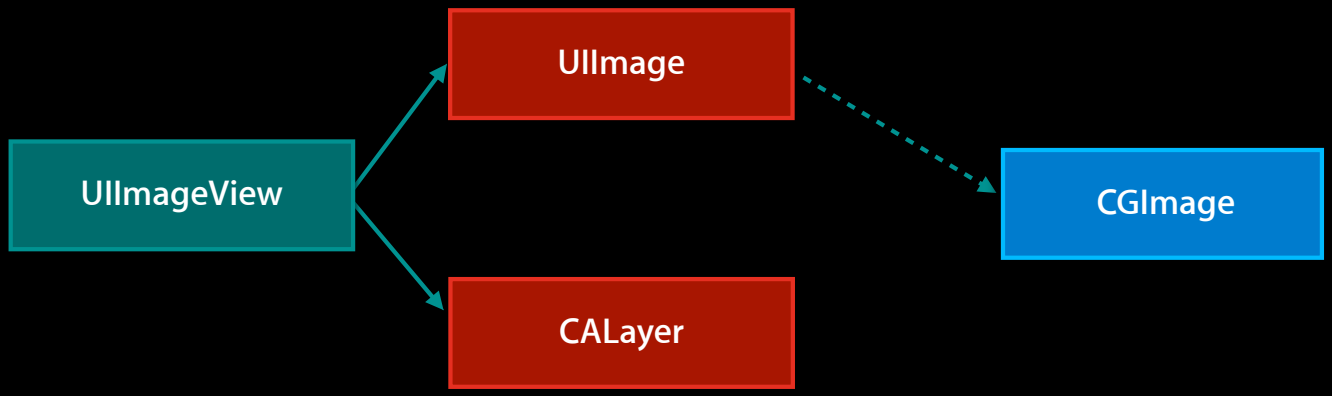
Images and Layers

- UIImage is a lightweight wrapper around CGImage

Improving Responsiveness

Images and Layers

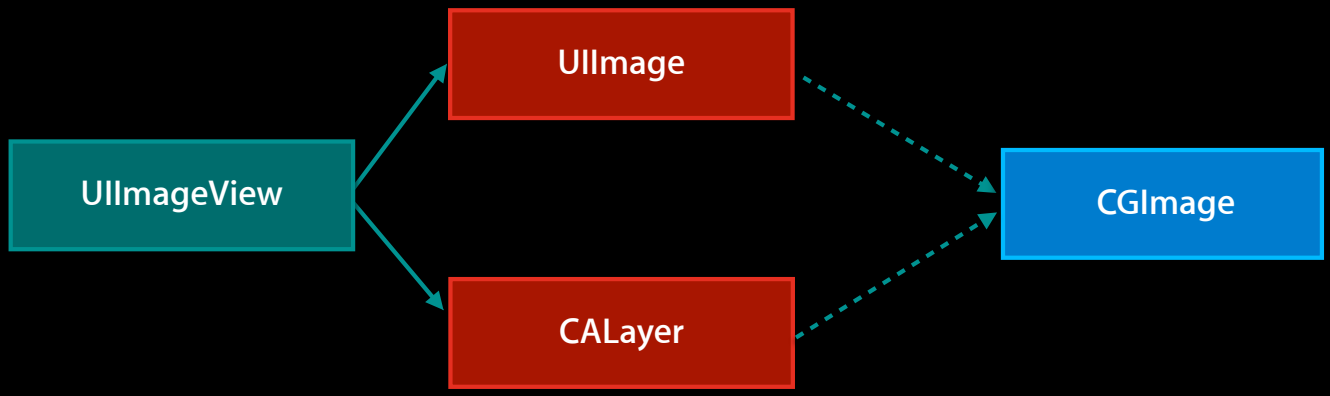
- UIImage is a lightweight wrapper around CGImage



Improving Responsiveness

Images and Layers

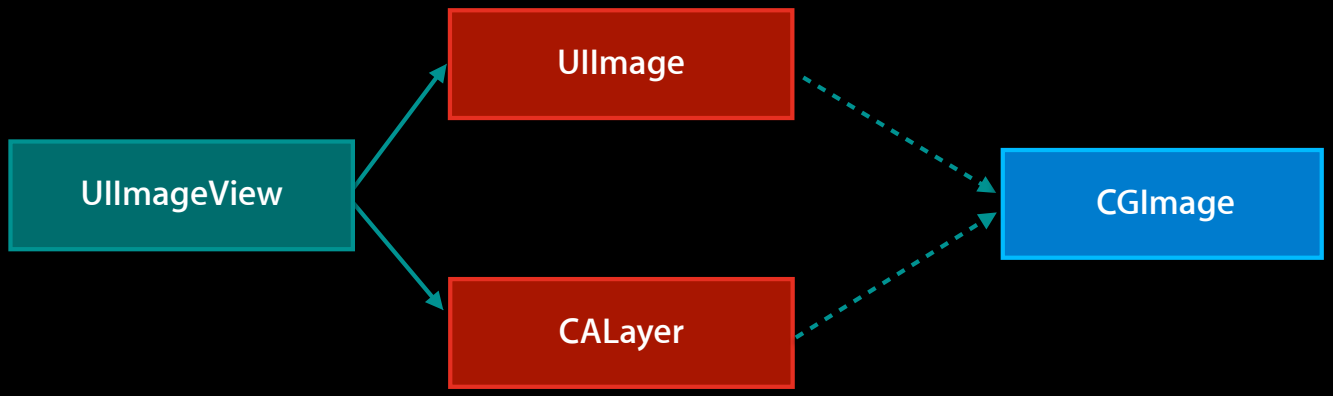
- UIImage is a lightweight wrapper around CGImage
- CALayer also has CGImage as contents



Improving Responsiveness

Images and Layers

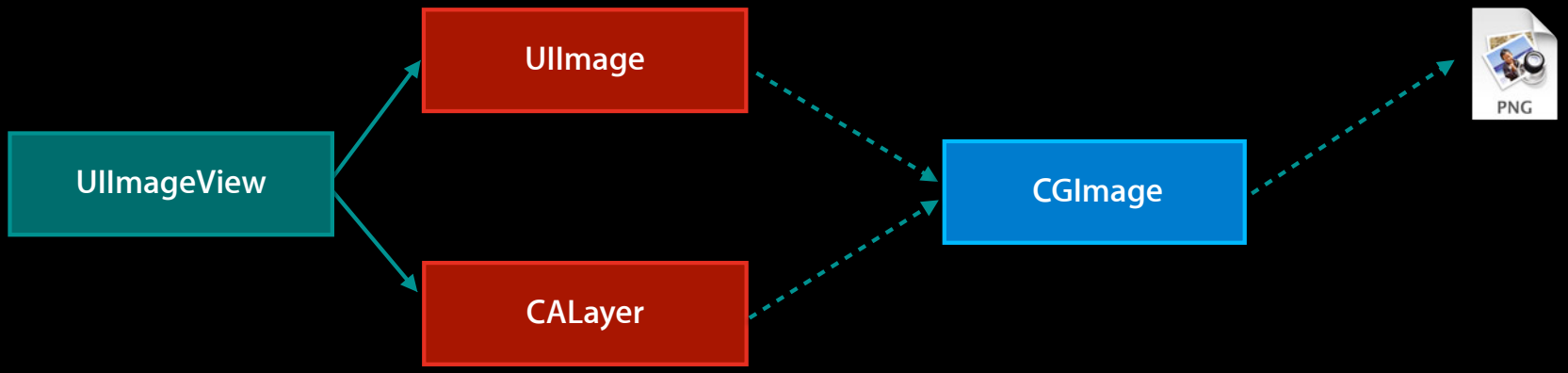
- UIImage is a lightweight wrapper around CGImage
- CALayer also has CGImage as contents
- CGImage backed by file or data, eventually by bitmap



Improving Responsiveness

Images and Layers

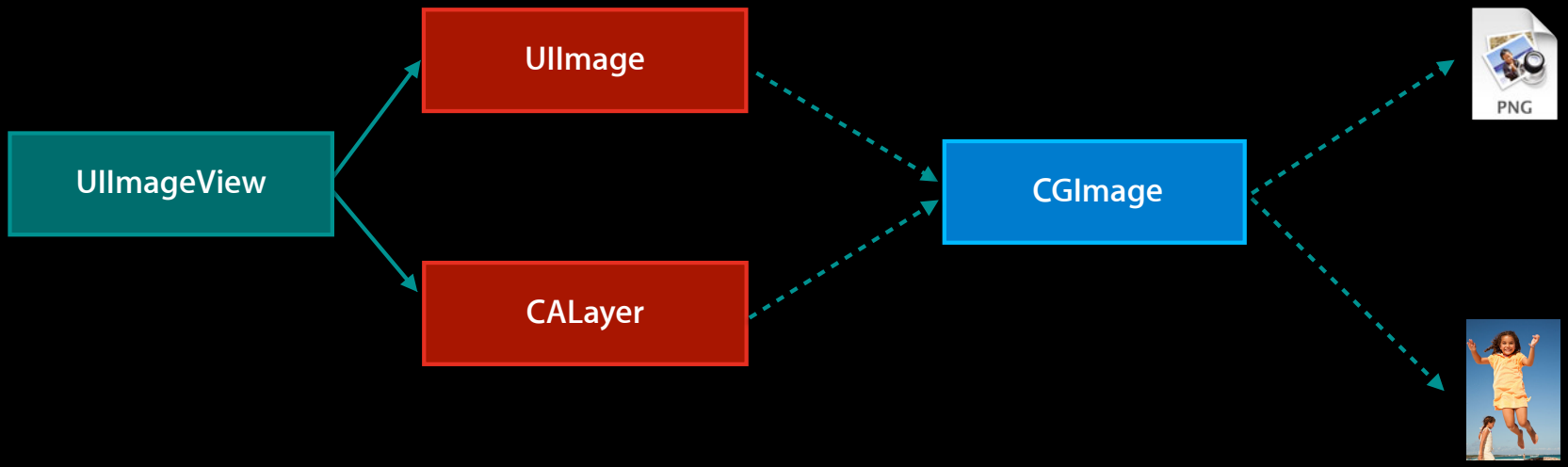
- UIImage is a lightweight wrapper around CGImage
- CALayer also has CGImage as contents
- CGImage backed by file or data, eventually by bitmap



Improving Responsiveness

Images and Layers

- UIImage is a lightweight wrapper around CGImage
- CALayer also has CGImage as contents
- CGImage backed by file or data, eventually by bitmap



Improving Responsiveness

UIImageView vs -[UIImage draw...]

- Use UIImageView instead of drawing image directly (usually)
  - CA can get the bitmap from the CGImage directly
  - Allow blending to happen on GPU
  - Built-in bitmap caching

Improving Responsiveness

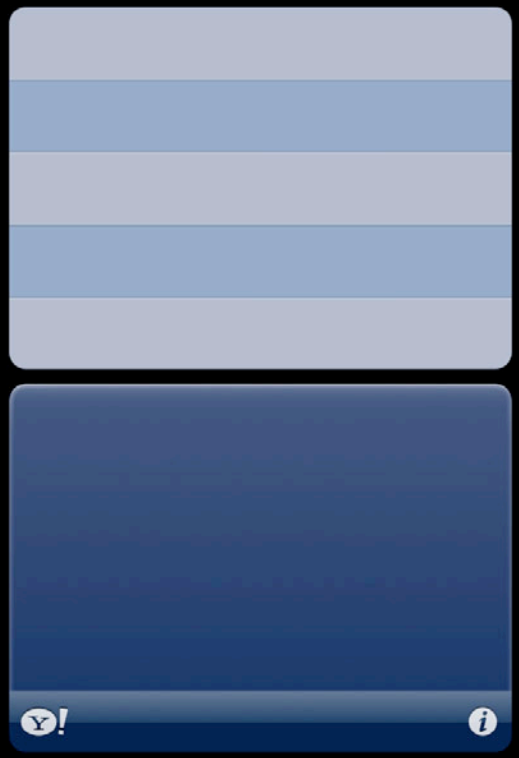
Images—General tips

- Size images appropriately for the view
  - Decoded image uses 4 bytes per pixel
  - Keep thumbnails as separate images
- Use images without alpha if you can
- Use the appropriate image format for the type of image

Improving Responsiveness

Image formats—PNG

- Xcode-optimized PNG should be your default format for assets
- Great for artwork with lots of solid color, gradients, or repeated patterns
- Lossless compression
- Very noisy images and photos have poor compression and are slow
- Rule of thumb—If PNG has good compression, use it



Improving Responsiveness

PNG optimizations

- Xcode does some important PNG optimizations, possibly including
  - Premultiply alpha, and byte-swap
  - Turn off some PNG compression modes
  - Allow concurrent decoding of a single image
- Optimizations are primarily for performance
- Xcode won't do lossy or not-known-safe optimizations
- Other image optimizers may help, but do Xcode optimization too

Improving Responsiveness

Image Formats—JPEG

- Great compression, small files
- Quality is good for photos or noisy artwork
- Relatively fast to decode
- Sometimes noticeable artifacts
- Can’t have alpha

Improving Responsiveness

Image formats—Other

- Generally, don't use anything else
- PNG and JPEG have been optimized for iOS and should get even better
- Wins in decode time from other formats usually outweighed by I/O

Improving Responsiveness

Image formats—Other

- Generally, don't use anything else
- PNG and JPEG have been optimized for iOS and should get even better
- Wins in decode time from other formats usually outweighed by I/O

Improving Responsiveness

Images—Caching

- When drawing into a bitmap context
  - `+[UIImage imageNamed:]` caches in purgeable memory and in image table
  - `+[UIImage imageWithContentsOfFile:]` does not
- All CGImages cache when set as contents of a layer
- `kCGImageSourceShouldCache` if you create CGImages directly
- Generally, don't cache images yourself

Improving Responsiveness

Efficiently using images

```
- (void)drawRect:(CGRect)rect
{
    [self.image drawInRect:[self bounds]
blendMode:kCGBlendModeNormal alpha:1.0];
}
```

Improving Responsiveness

Efficiently using images

```
- (void)drawRect:(CGRect)rect
{
    [self.image drawInRect:[self bounds]
blendMode:kCGBlendModeNormal alpha:1.0];
}
```

```
myView.layer.contents = (id)[self.image
CGImage];
```

Improving Responsiveness

Efficiently using images

```
- (void)drawRect:(CGRect)rect
{
    [self.image drawInRect:[self bounds]
blendMode:kCGBlendModeNormal alpha:1.0];
}
```

```
myView.layer.contents = (id)[self.image
CGImage];
```

Improving Responsiveness

-drawsAsynchronously

- CG will queue up draw commands to have the GPU fill backing store
- High setup cost
- High fixed memory hit
- Good for lots of drawing into a single view
- Always test performance before enabling

```
myView.layer.drawsAsynchronously = YES;
```

Improving Responsiveness

Speculative work

- Look up data for upcoming rows, and stuff into cache
- Do image decoding or drawing in a background thread
- Some work will be wasted, and caching means a memory hit
- Not simple to do this safely and performantly

Demo

Expensive drawing in a painting app

Graphics and Animations

- Introduction to Animations
- Responsive Animations
- Smooth Animations
- Scrolling

Smooth Animation Requirements

Smooth Animation Requirements

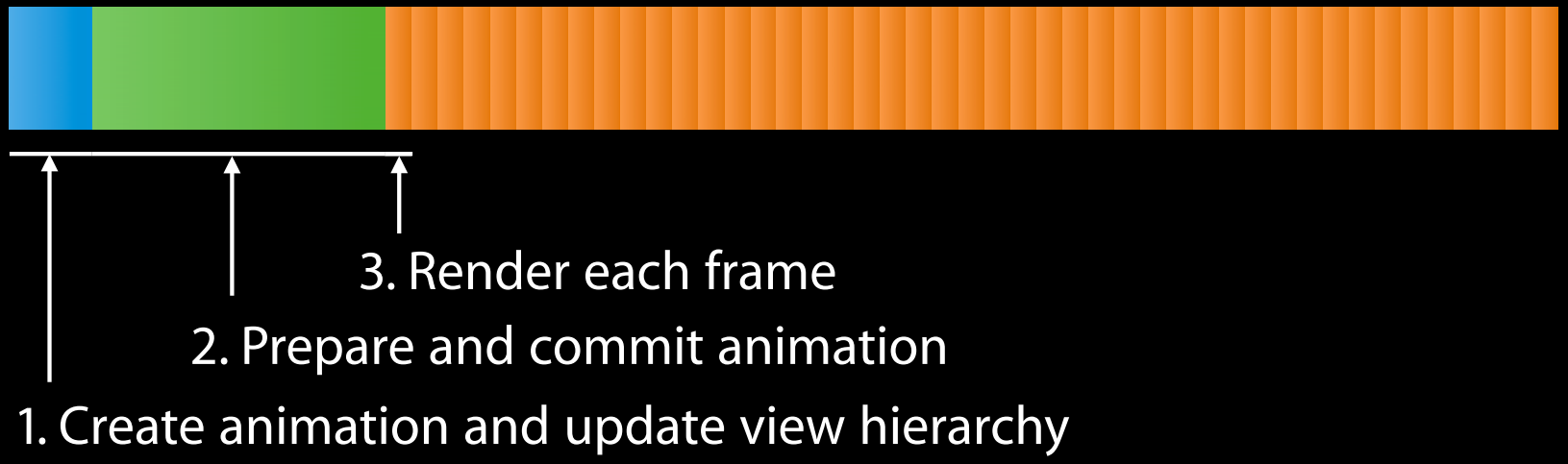
60 fps

Smooth Animation Requirements

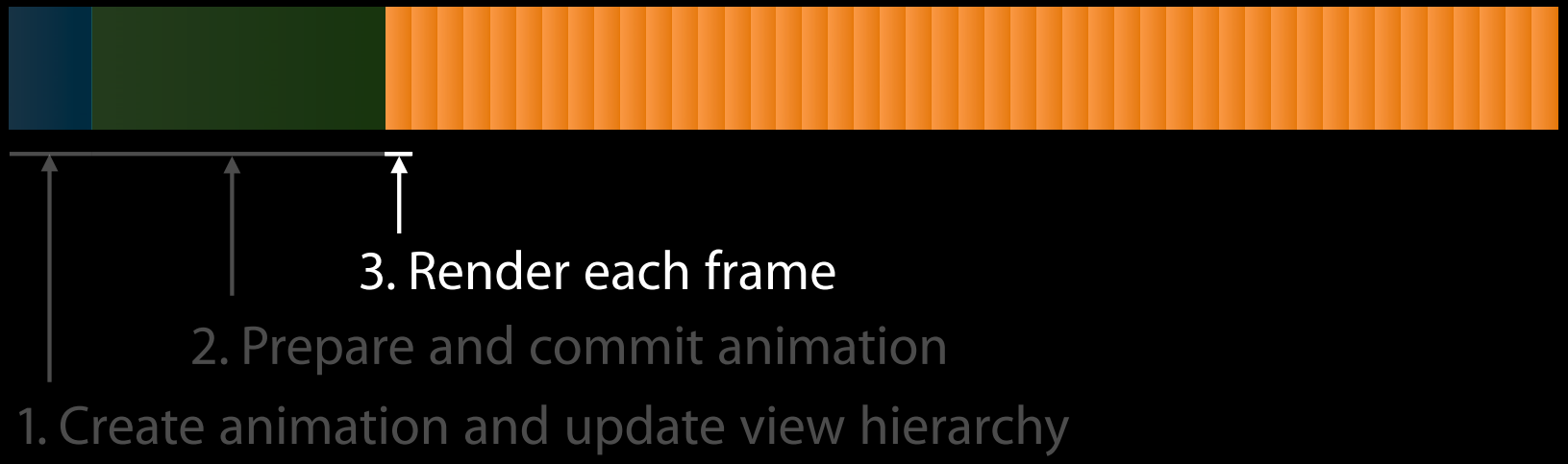
60 fps

16 ms/frame

Smoothness

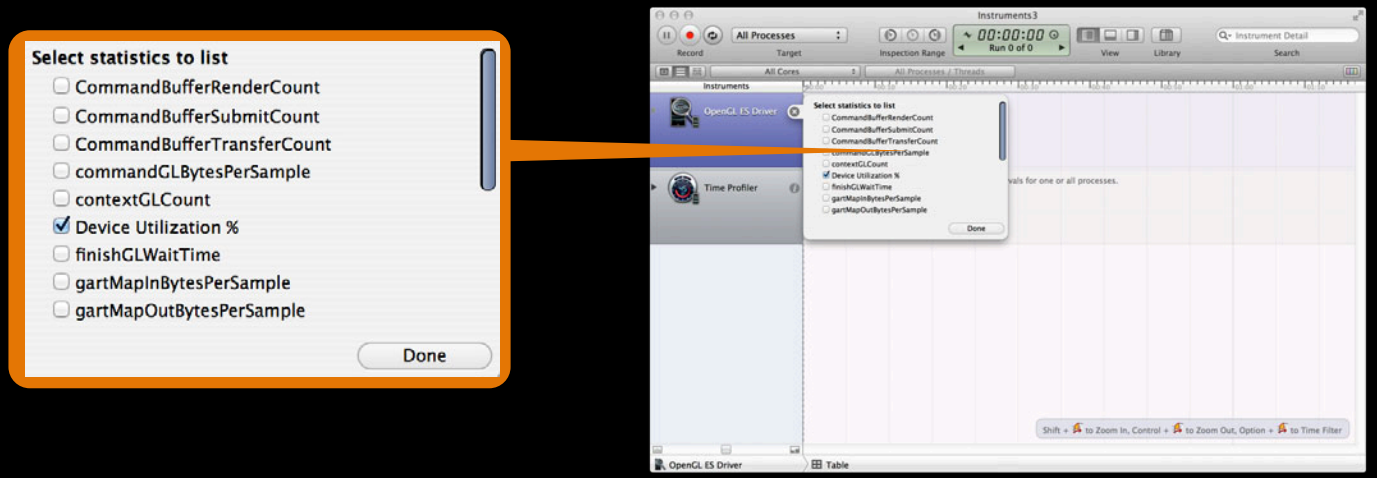

3. Render each frame
2. Prepare and commit animation
1. Create animation and update view hierarchy

Smoothness


3. Render each frame
2. Prepare and commit animation
1. Create animation and update view hierarchy

CPU bound vs. GPU bound

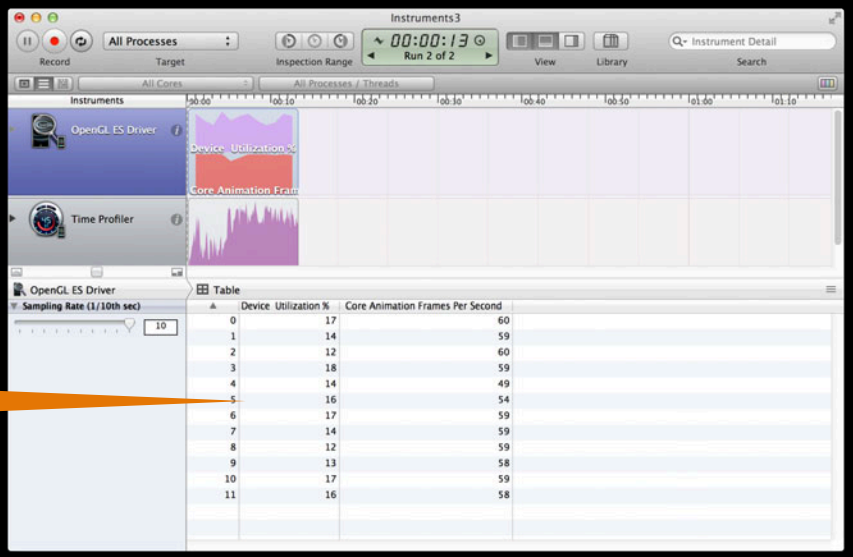
- CG drawing and ImageIO work is CPU bound
- Render server does work on CPU per layer, per frame
- Rendering itself is GPU bound
- Check device utilization in OpenGL ES instrument



CPU bound vs. GPU bound

- 100% device utilization is a good indicator of GPU-bound animation

	Device Utilization %	Core Animation Frames Per Second
0	17	60
1	14	59
2	12	60
3	18	59
4	14	49
5	16	54



Core Animation Instrument

Color Blended Layers

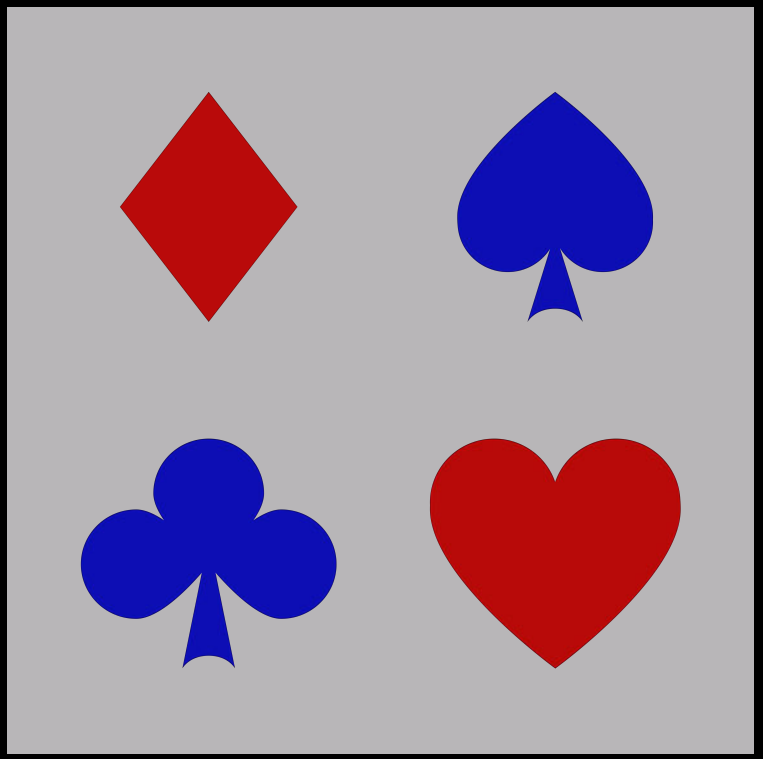
- Green means opaque, red blended
- Deeper red means more blending
- Try to figure out why there is so much, or flatten view hierarchy
- Starting with iPhone 3GS, GPU can handle overdraw of ~2.5 at 60fps

Flattening

- Flattening view hierarchy can help if you're GPU bound
- Not a magic solution!
  - More CPU up-front in client for less render server work
  - Hurts responsiveness
- Some CPU-bound scenarios can also benefit
- Measure, test, and iterate

Flattening

- Flattening view hierarchy can help if you're GPU bound
- Not a magic solution!
  - More CPU up-front in client for less render server work
  - Hurts responsiveness
- Some CPU-bound scenarios can also benefit
- Measure, test, and iterate



Flattening

Strategies for flattening

- Draw into a single view
- Use `-setShouldRasterize:` but with caveats
  - Limited cache space, depends on device
  - Cache invalidated when contents change
  - Verify with "Color cache hits/misses" in CoreAnimation instrument
  - Sometimes makes not-GPU-bound scenarios worse

Core Animation Instrument

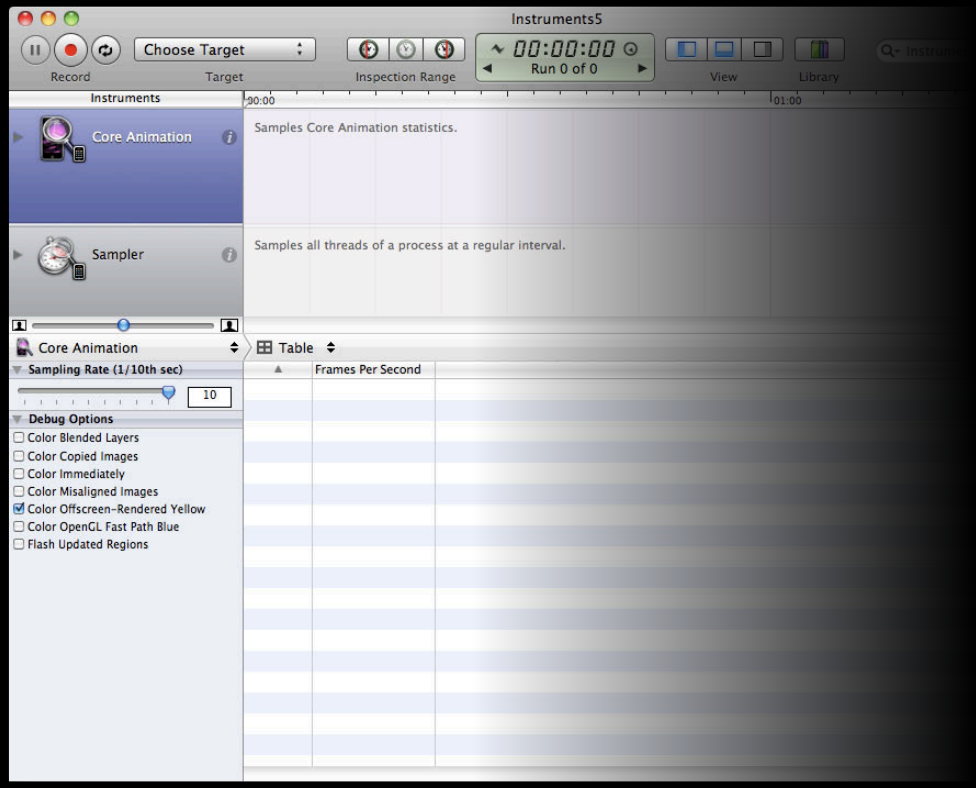
Offscreen Rendering

- Yellow means drawn into a separate offscreen context
- Most often needed for masking, but might be avoidable
- Also happens at least once with `-setShouldRasterize`

Core Animation Instrument

Offscreen Rendering

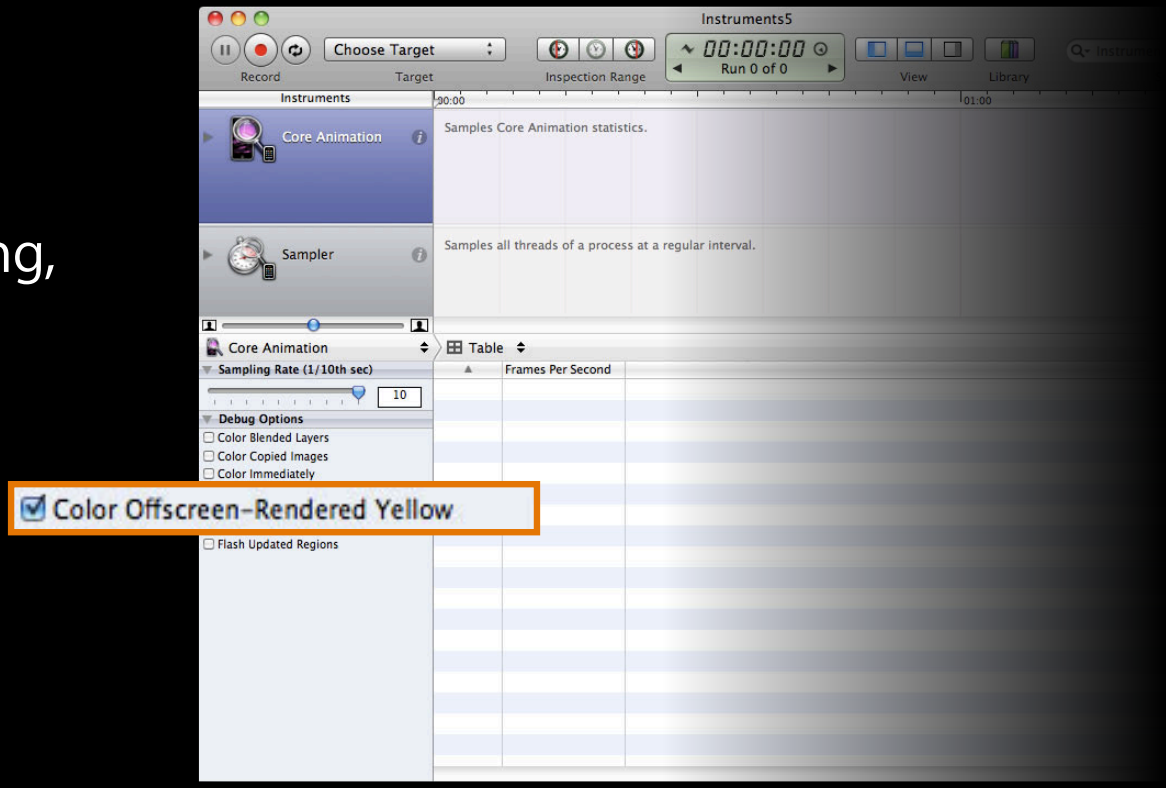
- Yellow means drawn into a separate offscreen context
- Most often needed for masking, but might be avoidable
- Also happens at least once with `-setShouldRasterize`



Core Animation Instrument

Offscreen Rendering

- Yellow means drawn into a separate offscreen context
- Most often needed for masking, but might be avoidable
- Also happens at least once with `-setShouldRasterize`



Core Animation Instrument

Offscreen Rendering

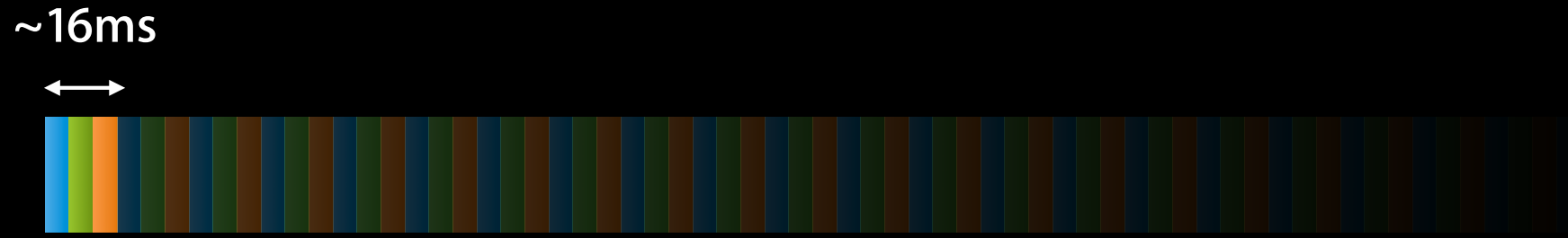
- Yellow means drawn into a separate offscreen context
- Most often needed for masking, but might be avoidable
- Also happens at least once with `-setShouldRasterize`

Graphics and Animations

- Introduction to Animations
- Responsive Animations
- Smooth Animations
- Scrolling

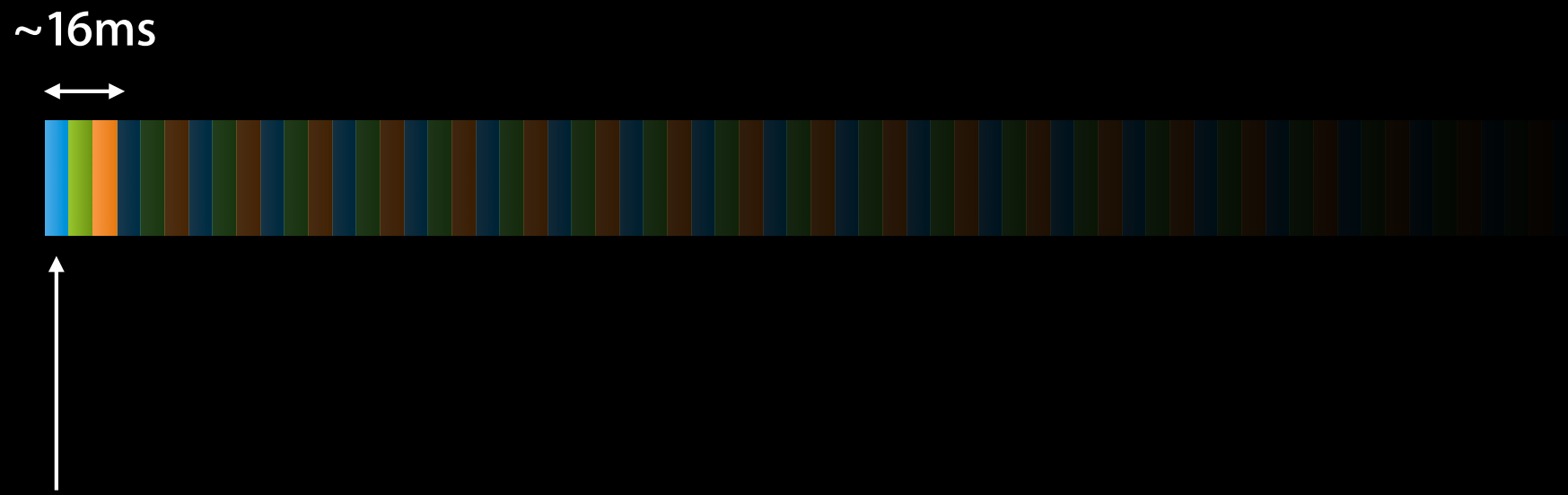
How Scrolling Works

- Each scroll update is its own animation



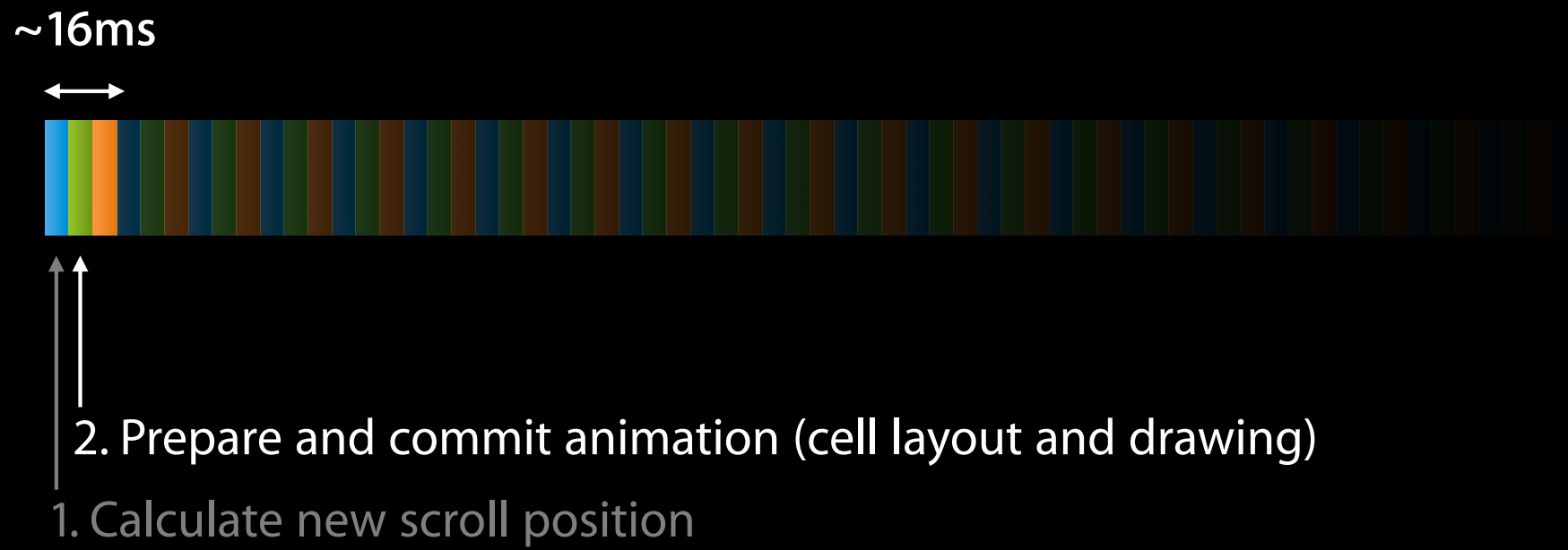
How Scrolling Works

- Each scroll update is its own animation



How Scrolling Works

- Each scroll update is its own animation



How Scrolling Works

- Each scroll update is its own animation

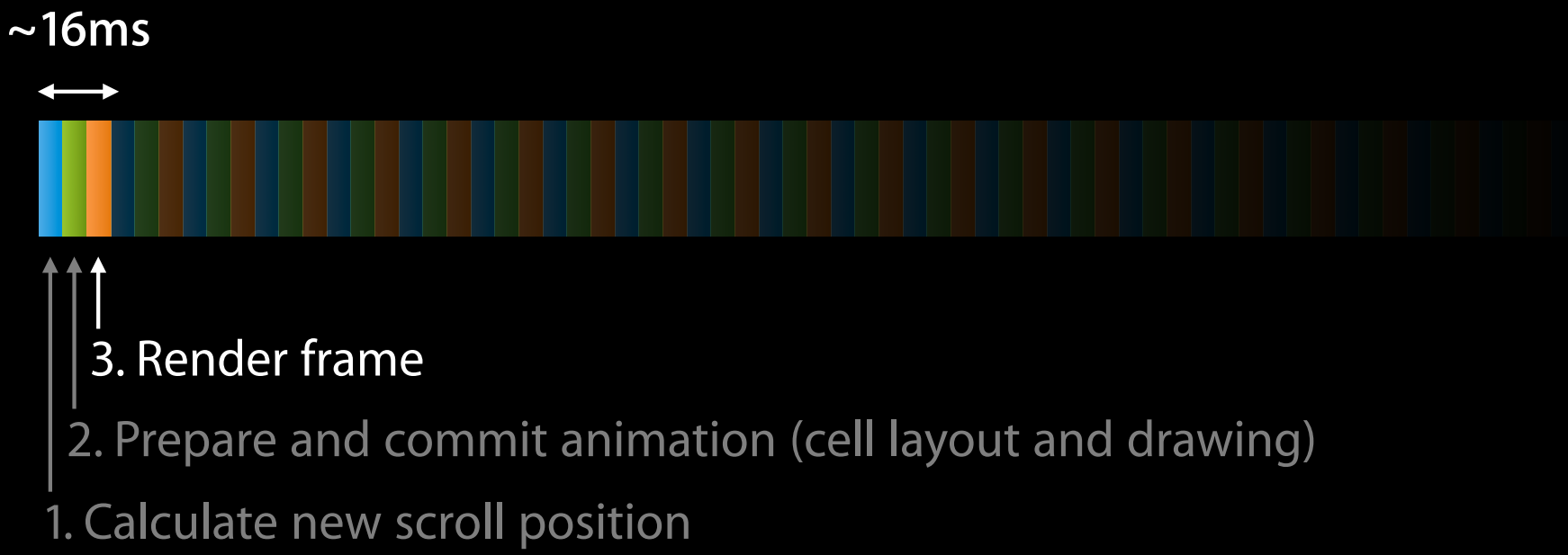
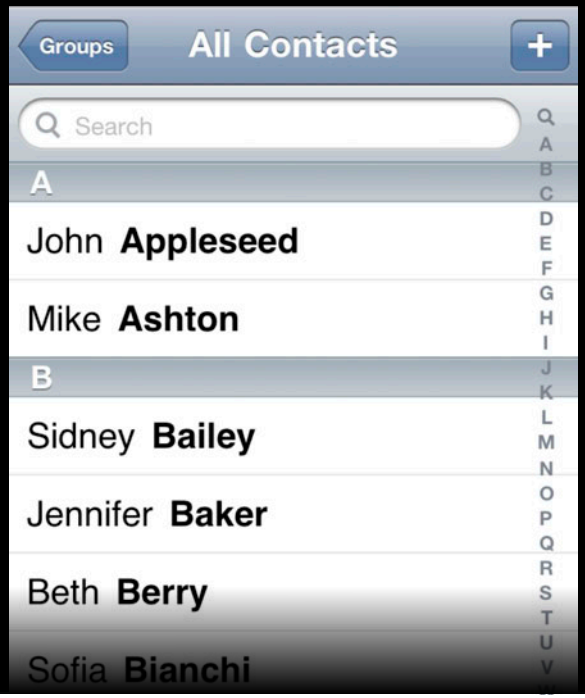


Table View Scrolling

- Must complete one new row *minimum* in 16ms
- Fast scrolling means possibly a full screen update in a single frame!



Preparing Cells Quickly

- Reuse cells and views
- Minimize layout and drawing time
- Speculative work
- Flatten view hierarchy, but test and iterate

Demo

Finding a scrolling bottleneck

Final Thoughts

- Test animations on a range of devices
  - Not just a matter of raw performance—limitations are different
- Different scenarios call for different solutions
- Measure, test, and iterate

More Information

Michael Jurewitz
Performance Evangelist Extraordinaire
jury@apple.com

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

Related Sessions

Learning Instruments	Presidio Wednesday 4:30PM
iOS App Performance: Responsiveness	Presidio Thursday 11:30AM
iOS App Performance: Memory	Presidio Thursday 4:30PM

Labs

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

Summary

- Introduction to Animations
- Responsive Animations
- Smooth Animations
- Scrolling


WWDC2012