Optimizing 2D Graphics and Animation Performance

Session 506

Tim Oriol
Mike Funk

These are confidential sessions—please refrain from streaming, blogging, or taking pictures

Agenda

Overview of topics for this session

- Supporting Retina Display
- Optimizing 2D graphics (Quartz 2D + Core Animation)
- Identify and fix common Retina Display pitfalls
- Using CGDisplayStream to get real-time display updates

Prerequisites

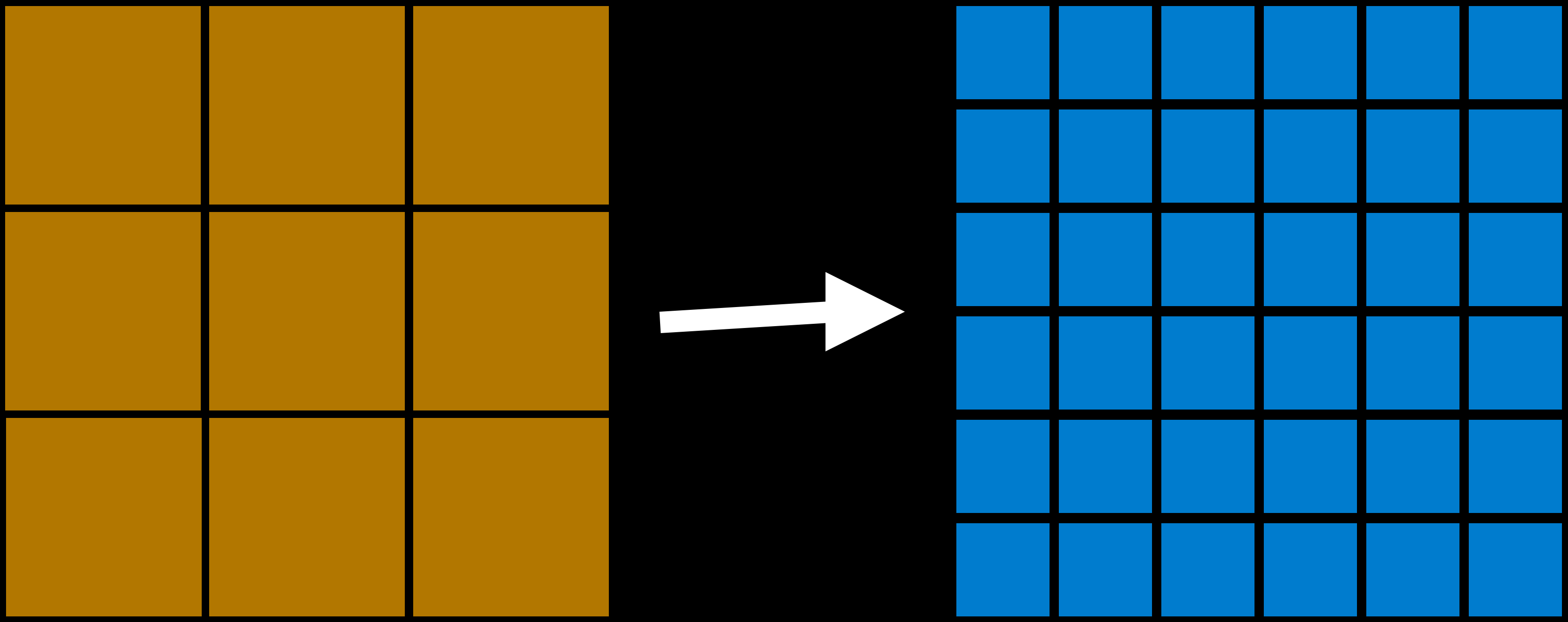
What you should know

- Core Animation framework
- Quartz 2D drawing techniques
- Basic knowledge of UIView and NSView

What Changes with Retina Displays?

Retina Displays

Today's Retina Displays have 4x the pixels of previous displays



Points Versus Pixels

What’s the point

- Points have nothing to do with typographer’s “points”
- Points are logical coordinates
- Pixels are actual device display coordinates
- One point is not always equal to one pixel
- The “scale factor” is the number of pixels per point
- Use points with Quartz 2D, UIKit, AppKit, and Core Animation

Retina Displays

Set up your scale factor

- Set the contentsScale property of layers that you would like to provide high-resolution content
- Text, shapes, Quartz 2D drawing, and any layers that you have provided high-resolution images as content
- UIKit/AppKit will set the appropriate contentsScale for layers they create

```
layer.contentsScale = [UIScreen mainScreen].scale;
```

Retina Displays

Set up your scale factor

- The CGContext provided to you via CALayer’s drawInContext will be set up correctly according to its contentsScale property
- Any CGContextBitmap you create yourself should be set up with pixel dimensions and scale your drawing appropriately
- On iOS, use this method to draw to a bitmap context:

```
void UIGraphicsBeginImageContextWithOptions(
   CGSize size,    //size in Points
   BOOL opaque,    //opaque drawing is much faster
   CGFloat scale   //the scale factor
);
```

Retina Displays

What do you need to do?

- Quartz 2D and CALayer based drawing is scaled using a scale factor
- This includes lines, text, shadows, and paths
- Make sure to set the scale factor for any contexts you create yourself that should provide high-resolution content
- Higher resolution images should to be provided (use “@2x” suffix)

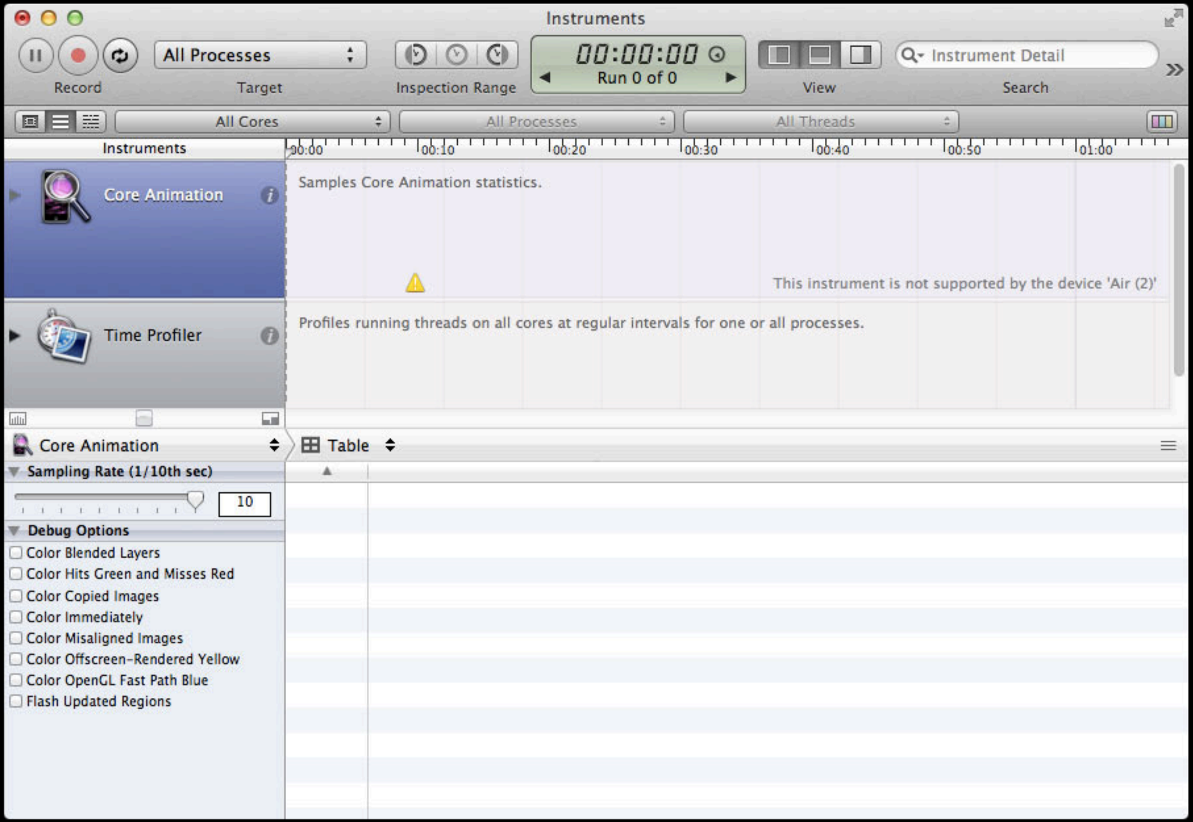
Retina Displays

Optimize

- Having 4x the pixels magnifies any drawing performance issues
- You simply can’t afford not to optimize your drawing code anymore

Core Animation in Instruments

Performance Tools



Samples Core Animation statistics.

This instrument is not supported by the device 'Air (2)'

Profiles running threads on all cores at regular intervals for one or all processes.

Core Animation

Sampling Rate (1/10th sec)

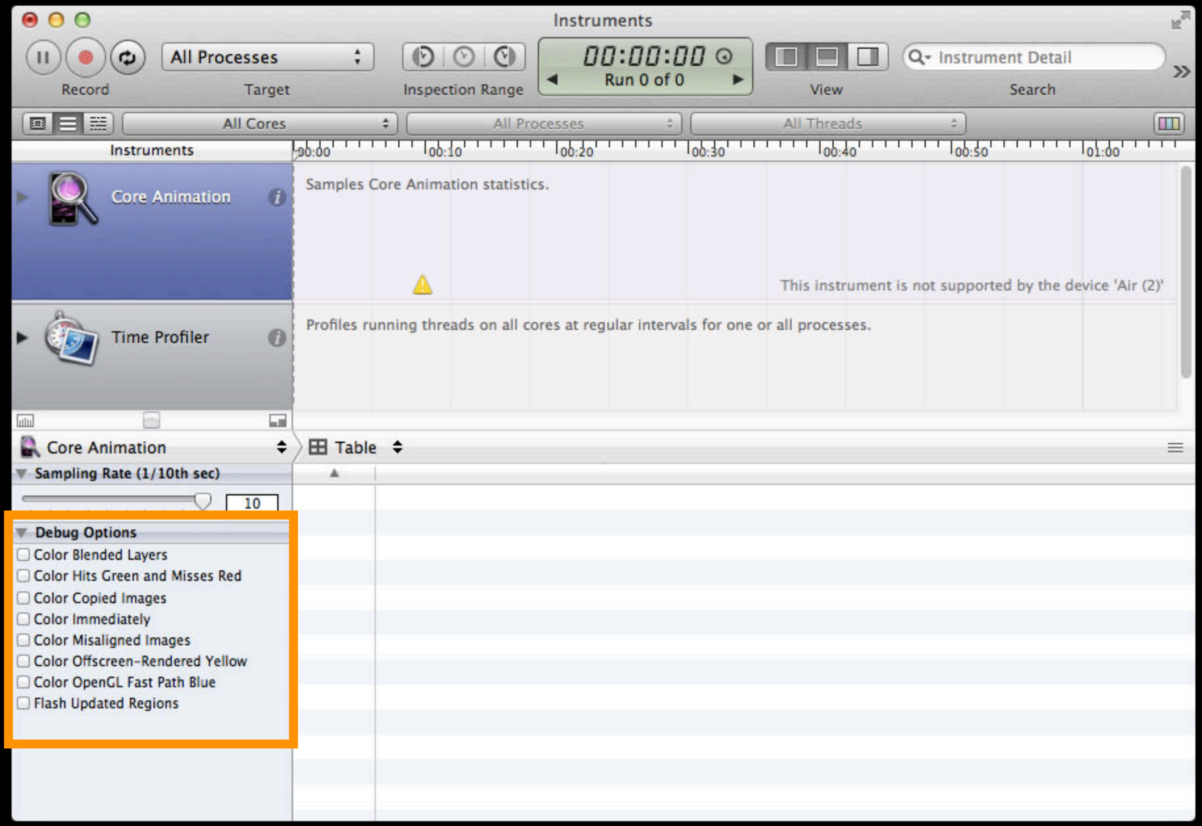
10

Debug Options

- □ Color Blended Layers
- □ Color Hits Green and Misses Red
- □ Color Copied Images
- □ Color Immediately
- □ Color Misaligned Images
- □ Color Offscreen-Rendered Yellow
- □ Color OpenGL Fast Path Blue
- □ Flash Updated Regions

Core Animation in Instruments

Performance Tools



Samples Core Animation statistics.

This instrument is not supported by the device 'Air (2)'

Profiles running threads on all cores at regular intervals for one or all processes.

Core Animation

Sampling Rate (1/10th sec)

10

Debug Options

- □ Color Blended Layers
- □ Color Hits Green and Misses Red
- □ Color Copied Images
- □ Color Immediately
- □ Color Misaligned Images
- □ Color Offscreen-Rendered Yellow
- □ Color OpenGL Fast Path Blue
- □ Flash Updated Regions

Demo

Finger painting app for iPad and Instruments

Useful Tools for Performance Optimization

See what's happening

- Instruments, particularly the Core Animation tool
- Quartz Debug (only on the Mac)
    - How to get Quartz Debug
        - Xcode->OpenDeveloperTool->MoreDeveloperTools…
        - Download and install the "Graphics Tools for Xcode" package

Quartz 2D Drawing Optimization

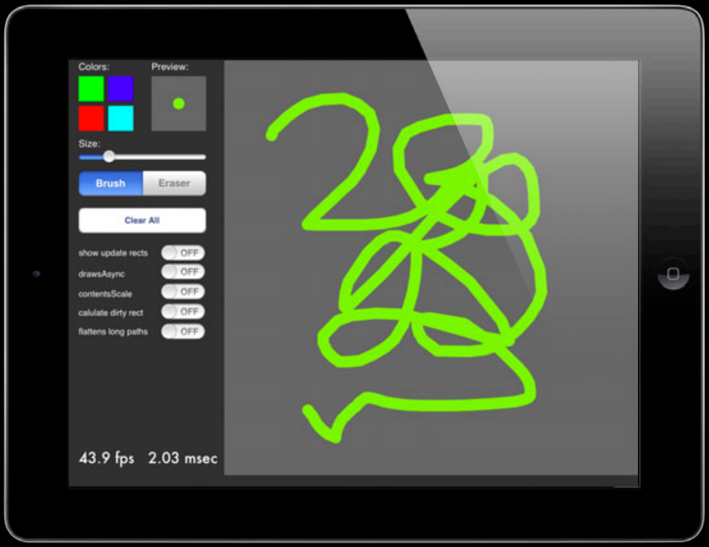
General Graphics Optimization

The Golden Rule

- Never draw more than you actually need to

Quartz 2D

Redraw only what has changed



show update rects OFF

drawsAsync OFF

contentsScale OFF

calulate dirty rect OFF

flattens long paths OFF

43.9 fps 2.03 msec

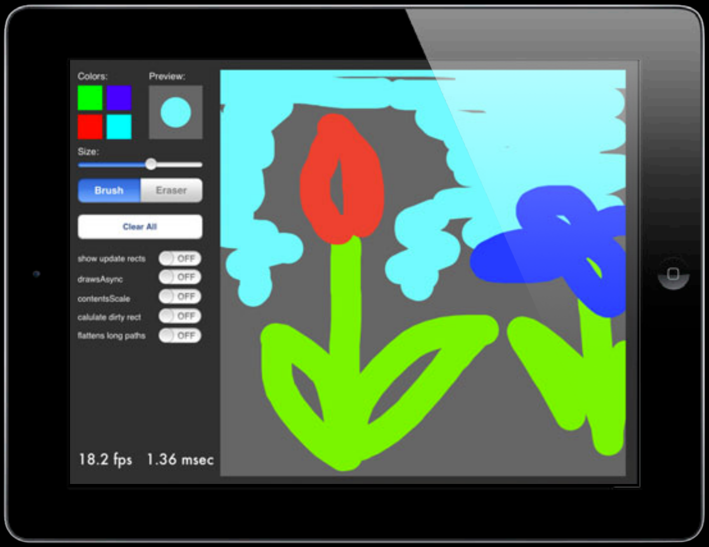
Quartz 2D

Redraw only what has changed

- Call setNeedsDisplayInRect: with the area you know as changed
- This will set up the clipRect for your drawRect: code
- You don’t need to change your drawing code
- Quartz 2D will automatically cull any drawing outside of the clipRect

Quartz 2D

Set up once and reuse



show update rects OFF
drawsAsync OFF
contentsScale OFF
calulate dirty rect OFF
flattens long paths OFF

18.2 fps 1.36 msec

Quartz 2D

Create state outside of drawRect:

- Don’t set up the same CGColors, CGPaths, clipShapes every draw call
- Make them once on initialization and reuse when drawing
- Even nonstatic items can benefit

Quartz 2D

Use offscreen buffers to flatten content

- Drawing complex CGPaths can be slow
- When appending to a large CGPath, don't redraw the entire path
- Flatten existing drawing to a bitmap
- Only draw the new elements

Quartz 2D

Use offscreen buffers to flatten content

- Drawing complex CGPaths can be slow
- When appending to a large CGPath, don't redraw the entire path
- Flatten existing drawing to a bitmap
- Only draw the new elements

Demo

Finger painting app for iPad with optimizations

Core Animation Optimization

Place Static Content into a Separate View

- Items that you expect to change rarely or not at all
- Core Animation maintains a bitmap cache and composites in hardware

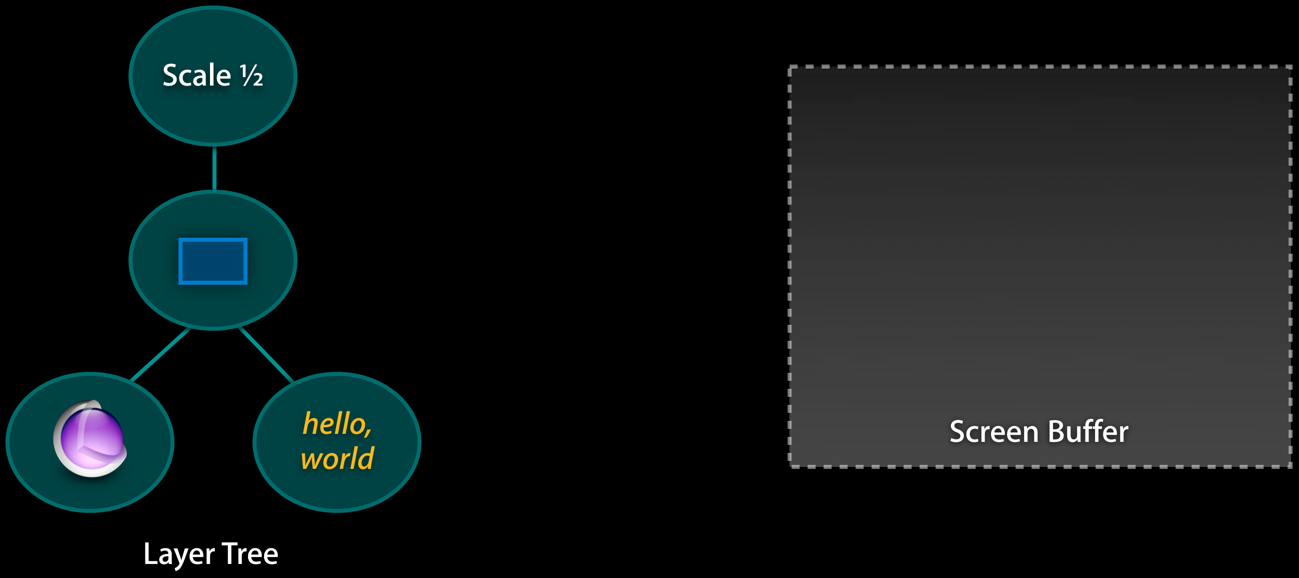
CALayer.shouldRasterize

Layer subtree bitmap caching

- This can also be done on a per-layer basis
- Setting the shouldRasterize property on the base CALayer containing the static content subtree
- Rasterizing locks the layer image to a particular size
- Always set the rasterizationScale whenever you use shouldRasterize

```
layer.rasterizationScale = layer.contentsScale;
```

Bitmap Caching



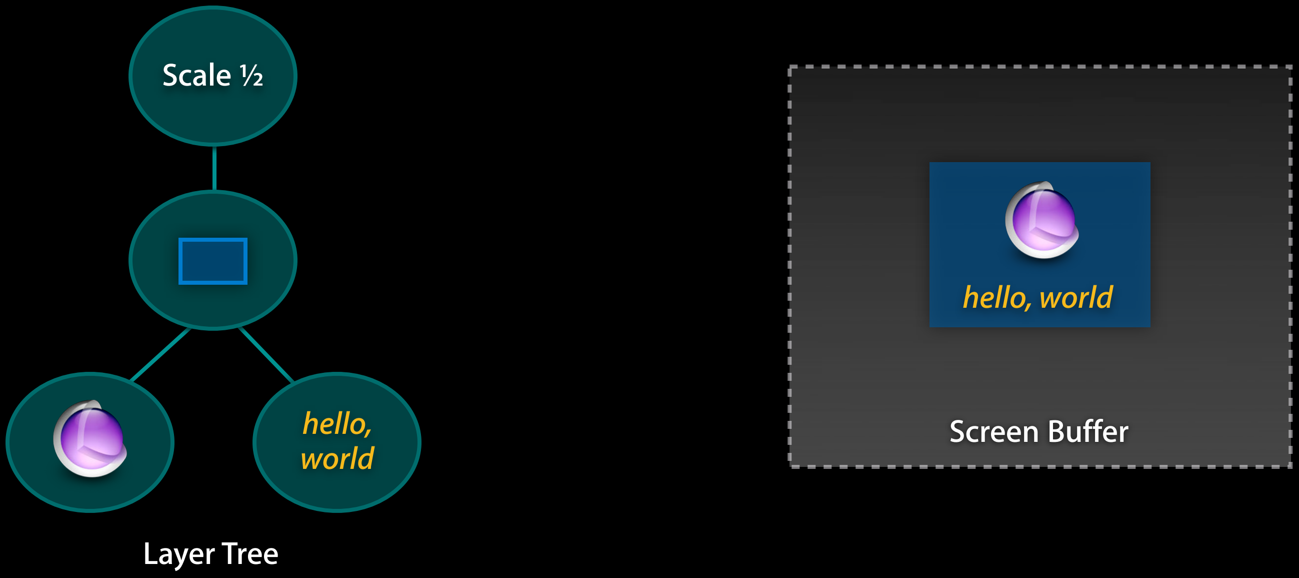
Scale ½

hello, world

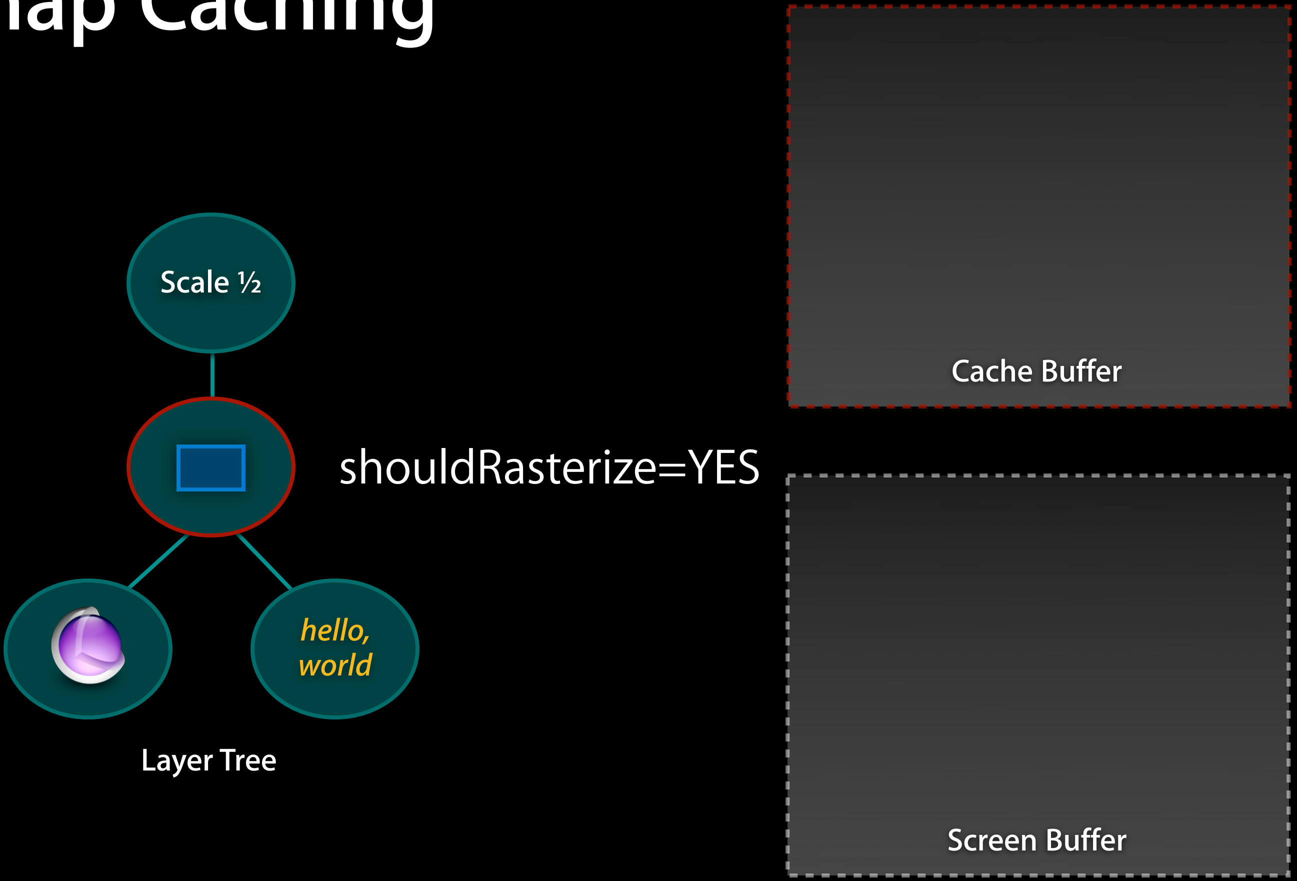
Layer Tree

Screen Buffer

Bitmap Caching



Bitmap Caching



Scale ½

shouldRasterize=YES

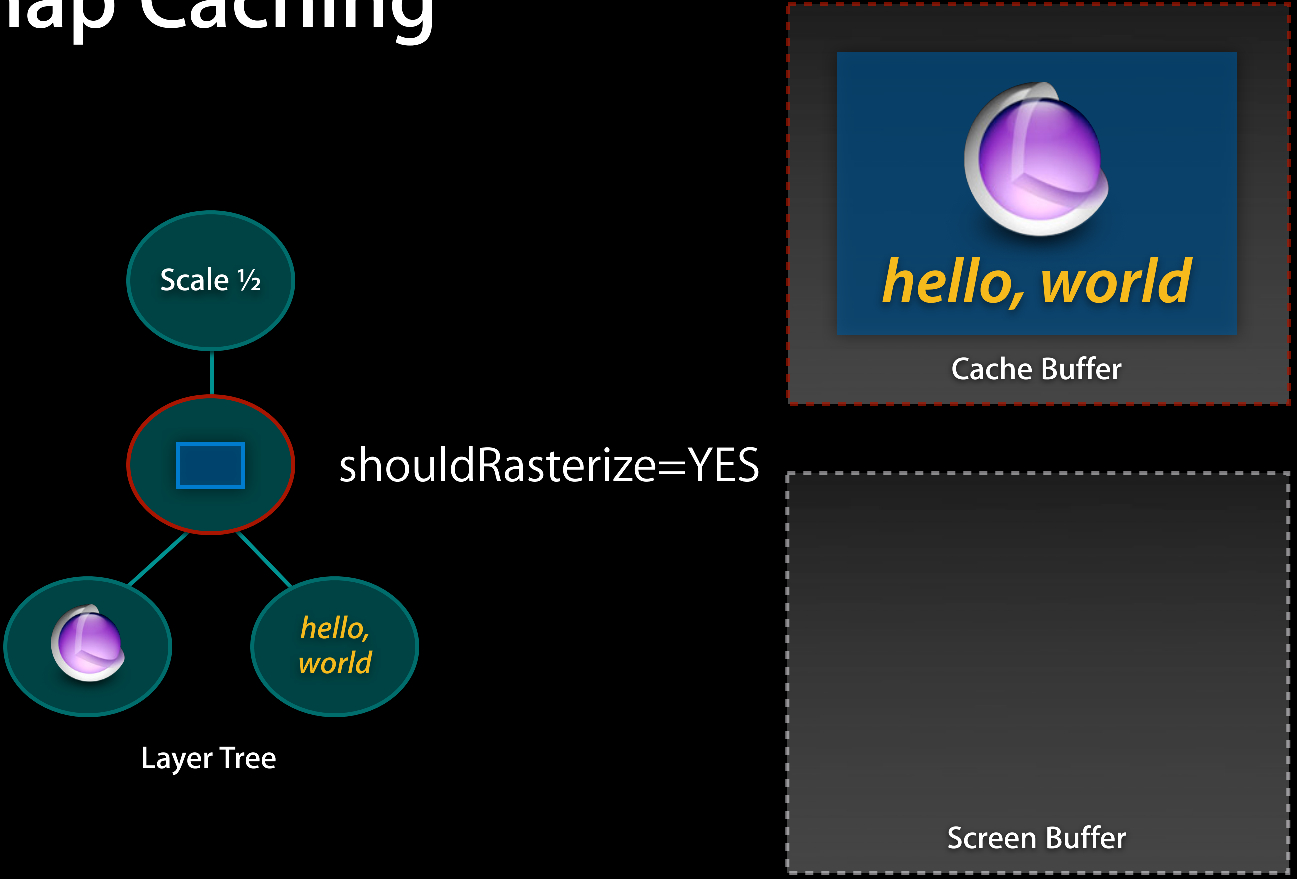
hello, world

Layer Tree

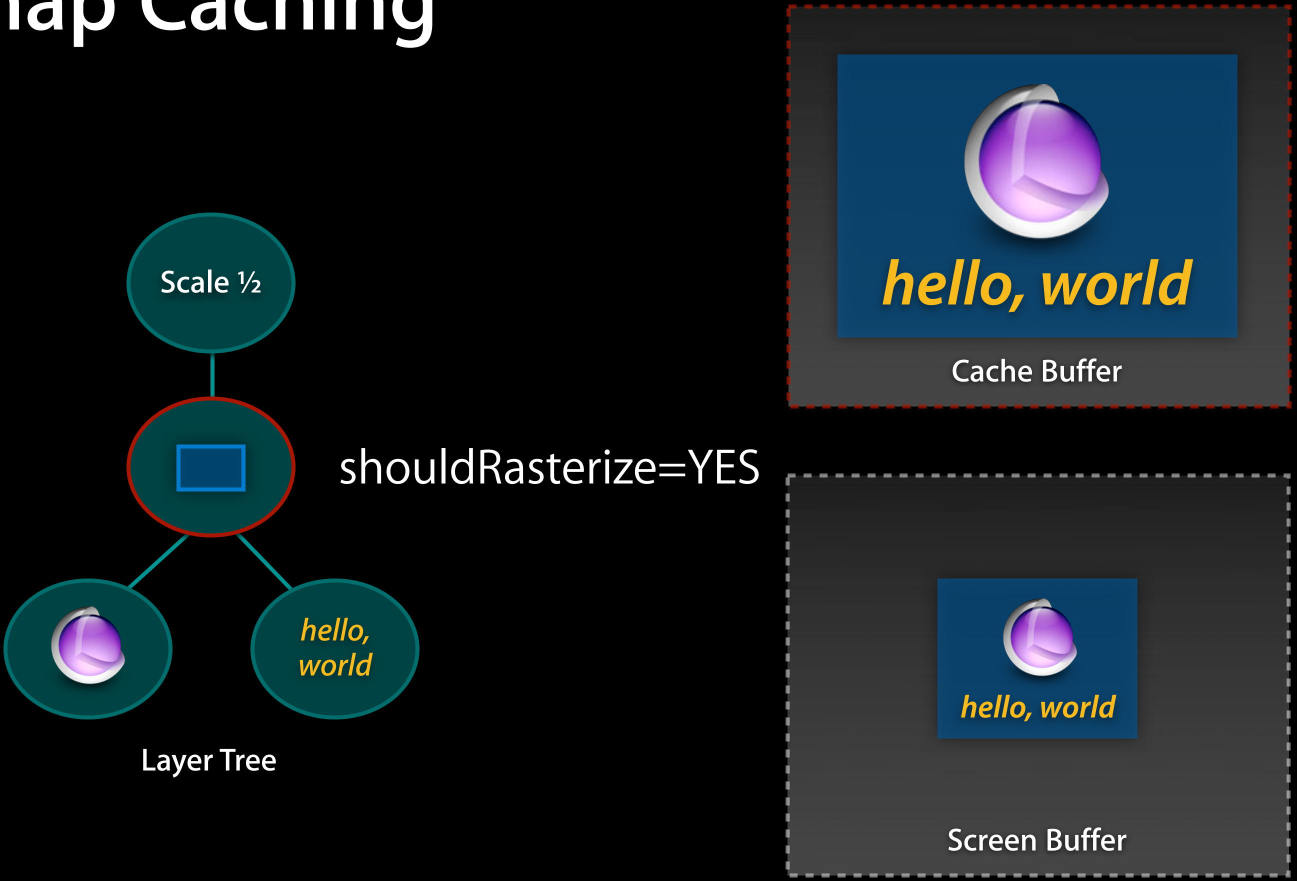
Cache Buffer

Screen Buffer

Bitmap Caching

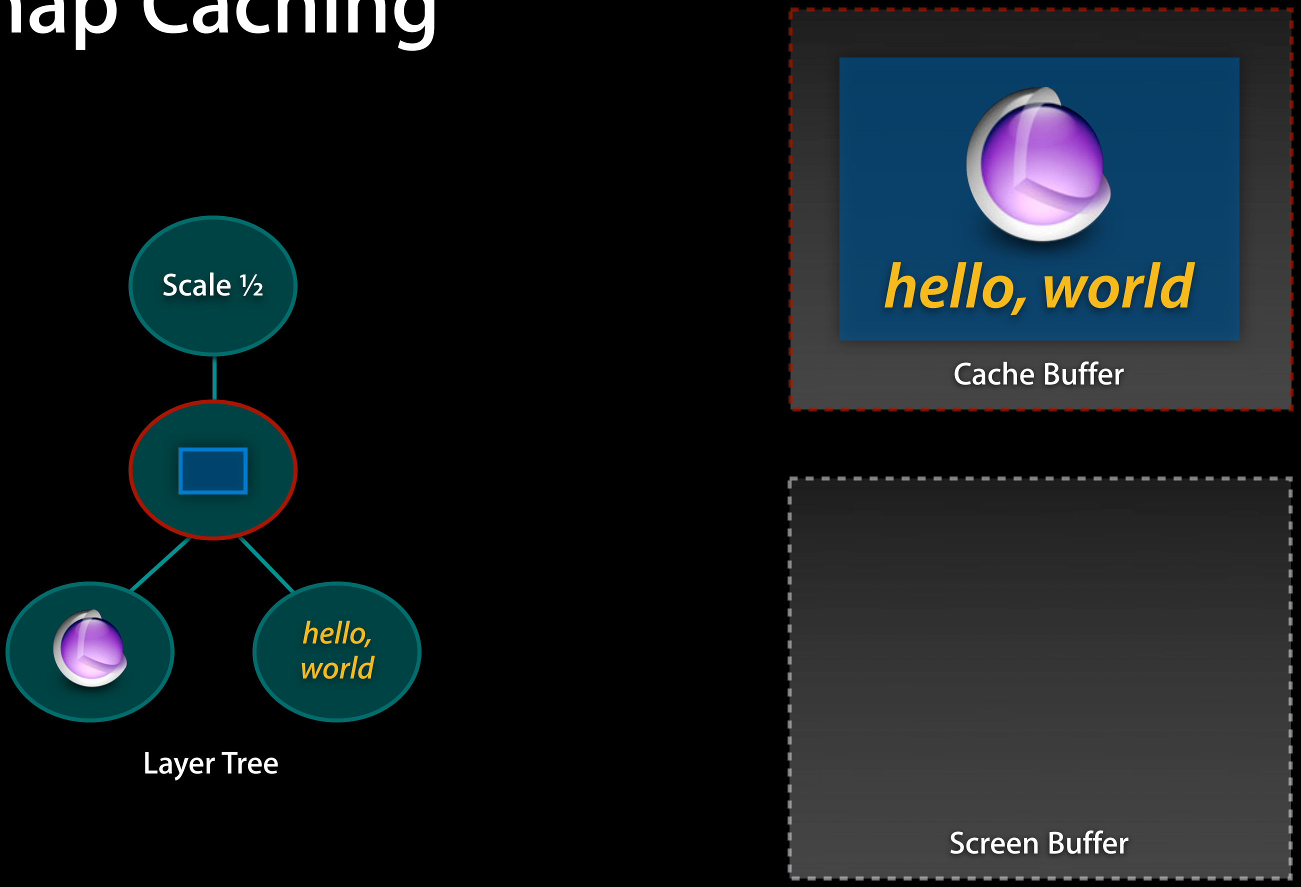


Bitmap Caching



shouldRasterize=YES

Bitmap Caching



Scale ½

hello, world

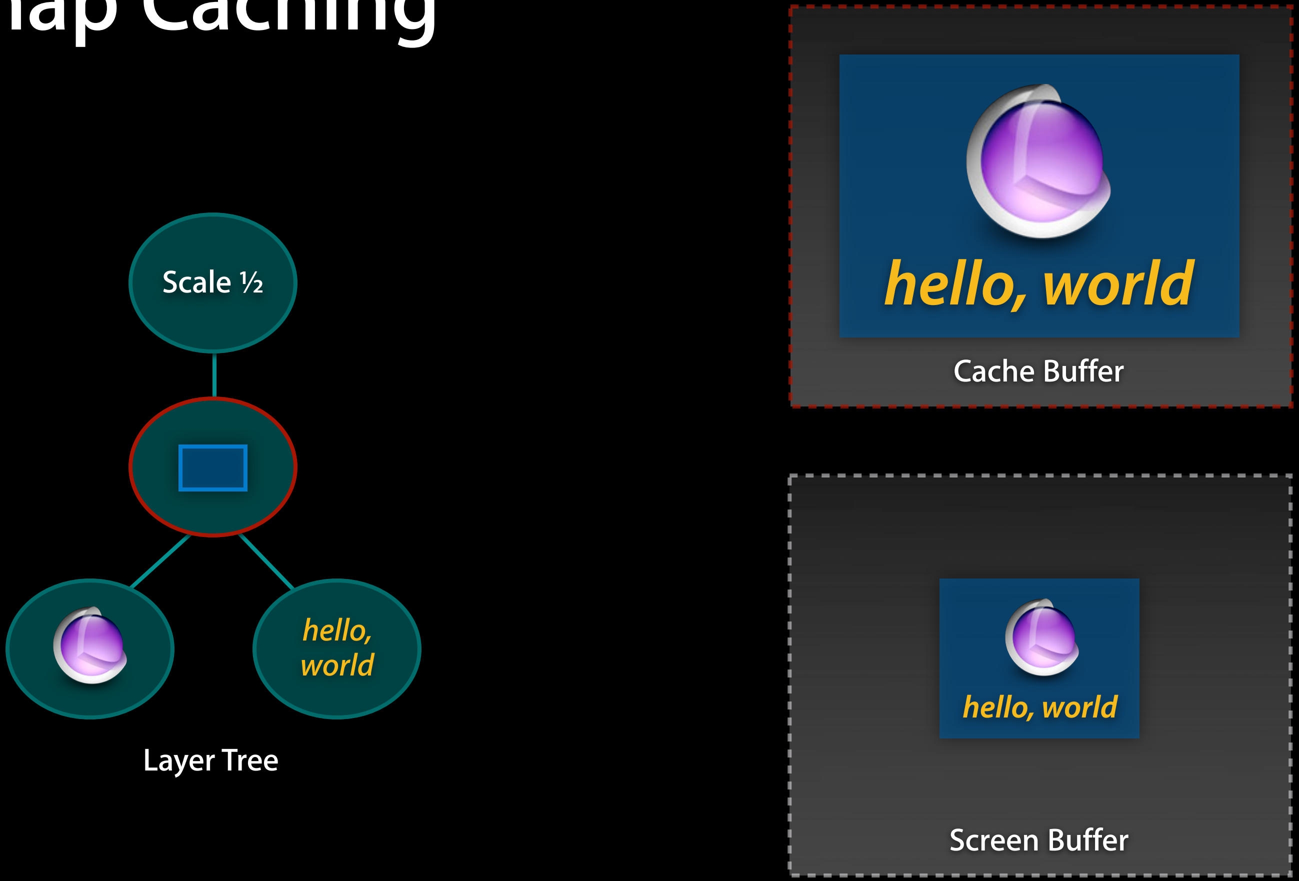
Layer Tree

hello, world

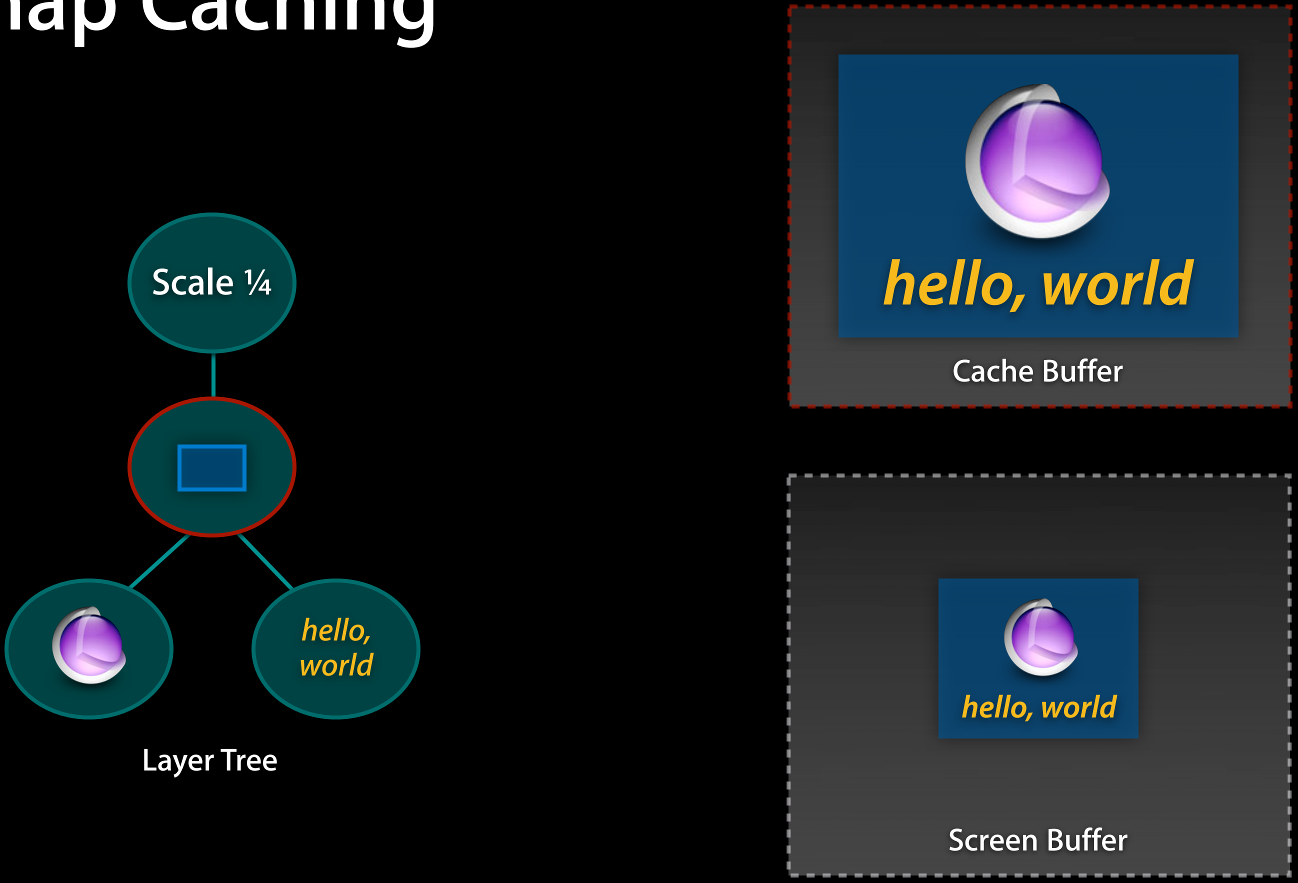
Cache Buffer

Screen Buffer

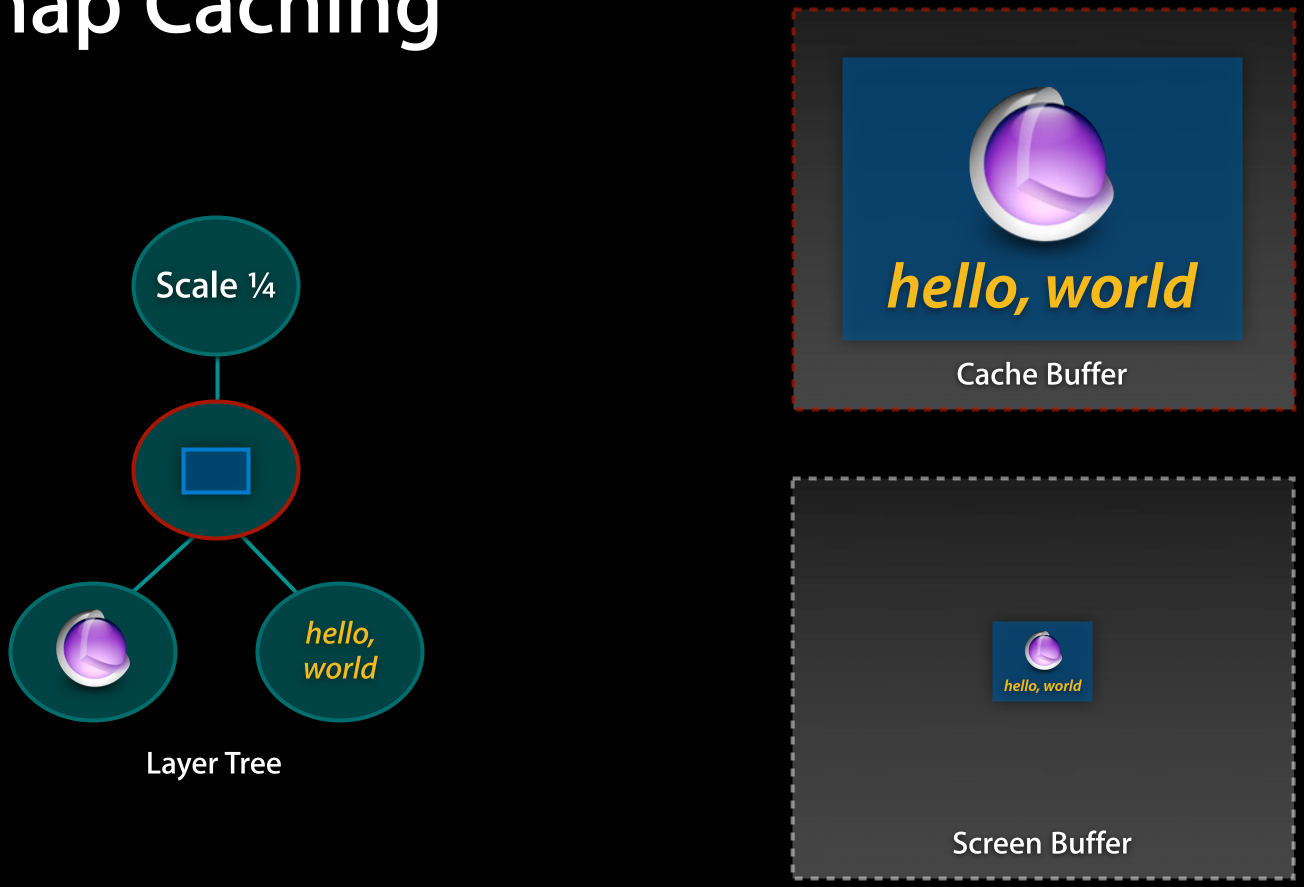
Bitmap Caching



Bitmap Caching



Bitmap Caching



CALayer.shouldRasterize

Layer subtree bitmap caching

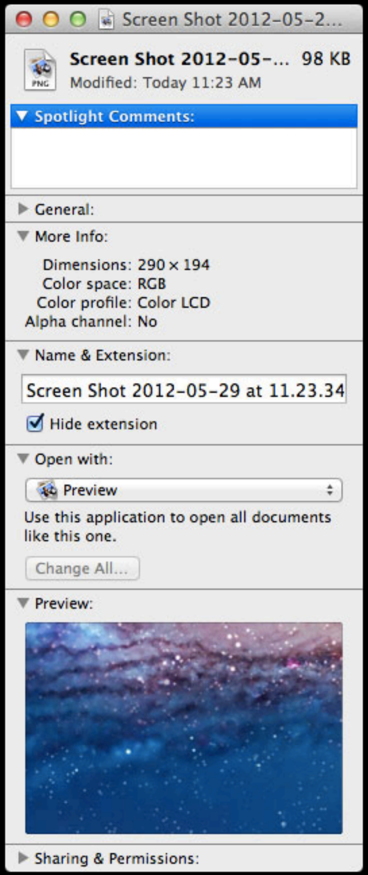
- Rasterization occurs before the mask is applied
- Caching and not reusing is more expensive than not caching at all
- This is a time vs. memory trade-off

Core Animation

Alpha blending

- Alpha blending is much slower than drawing opaque content
- Always use opaque images if possible

Strip Alpha Channels from Opaque Images



Screen Shot 2012-05-2...

Screen Shot 2012-05-... 98 KB
Modified: Today 11:23 AM

Spotlight Comments:

General:

More Info:

Dimensions: 290 × 194
Color space: RGB
Color profile: Color LCD
Alpha channel: No

Name & Extension:

Screen Shot 2012-05-29 at 11.23.34

☑ Hide extension

Open with:

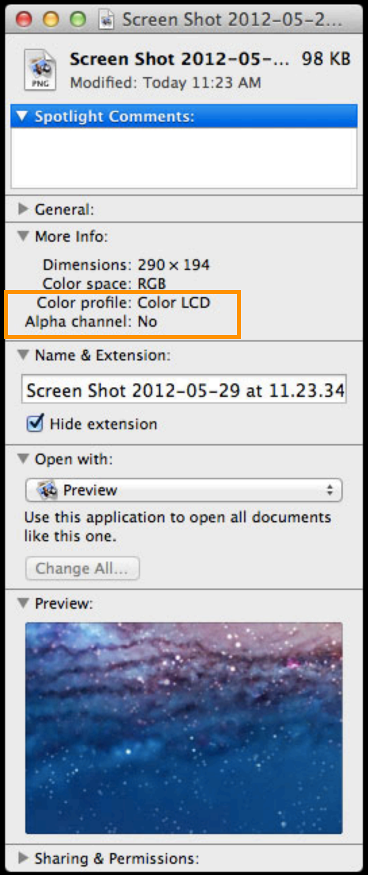
Preview

Use this application to open all documents like this one.

Preview:

Sharing & Permissions:

Strip Alpha Channels from Opaque Images



Screen Shot 2012-05-2...

Screen Shot 2012-05-... 98 KB
Modified: Today 11:23 AM

▼ Spotlight Comments:

▶ General:

▼ More Info:

Dimensions: 290 × 194
Color space: RGB
Color profile: Color LCD
Alpha channel: No

▼ Name & Extension:

Screen Shot 2012-05-29 at 11.23.34

☑ Hide extension

▼ Open with:

Preview

Use this application to open all documents like this one.

Change All...

▼ Preview:

▶ Sharing & Permissions:

Core Animation

Drop shadows

- Shadows are expensive to generate
- Use shadowPath to define the opaque regions
- Generate once and use shouldRasterize

Core Animation

Drop shadows

- Shadows are expensive to generate
- Use shadowPath to define the opaque regions
- Generate once and use shouldRasterize

```
layer.shadowPath = myOutlinePath;
```

Core Animation

Use shadowPath to specify opaque areas

Core Animation

Use shadowPath to specify opaque areas

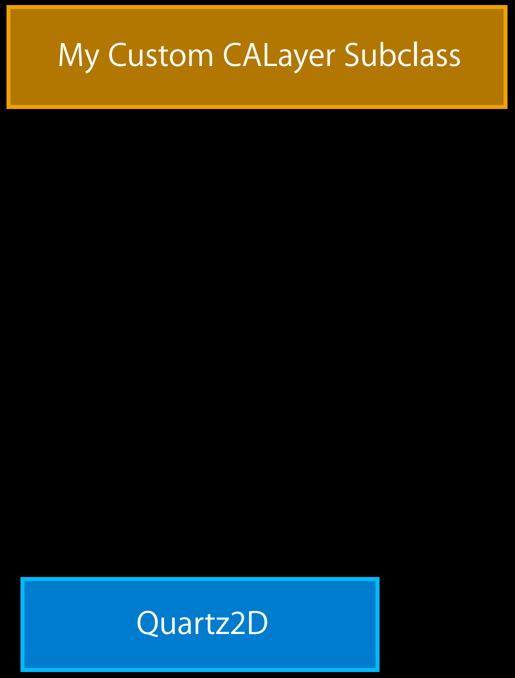
CALayer.drawsAsynchronously

When should this be used

- When supplying content to a CALayer via `-drawInContext:` method there are two ways Core Animation can render
  - Normal drawing will block the calling thread until complete
  - Asynchronous drawing will render in the background freeing up the caller to perform other tasks

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

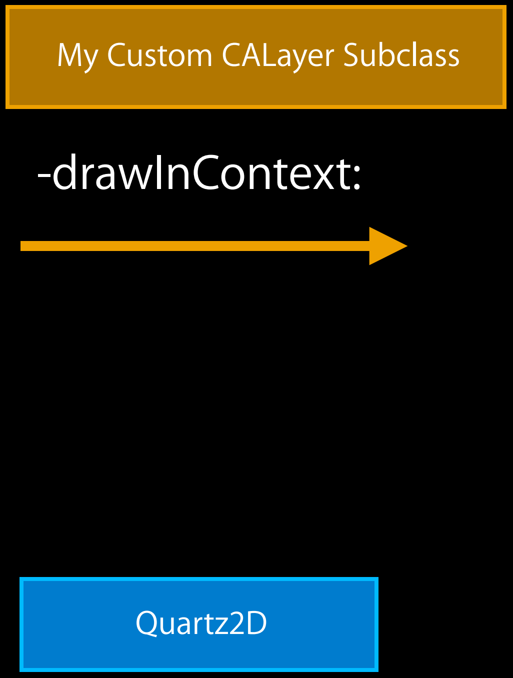
CALayer Normal Drawing Mode



My Custom CALayer Subclass

Quartz2D

CALayer Normal Drawing Mode

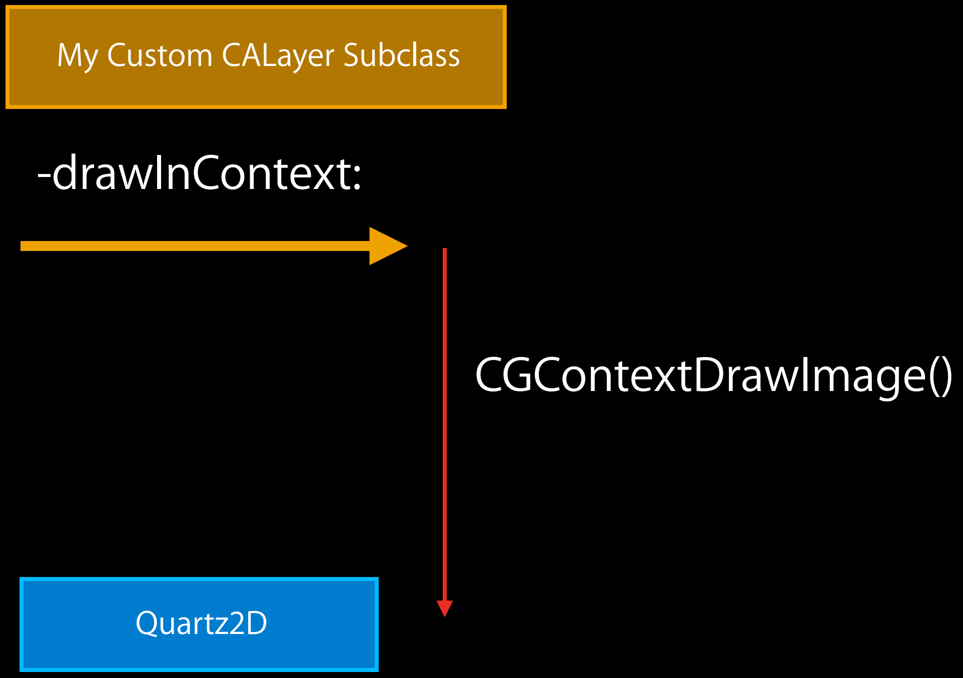


My Custom CALayer Subclass

-drawInContext:

Quartz2D

CALayer Normal Drawing Mode



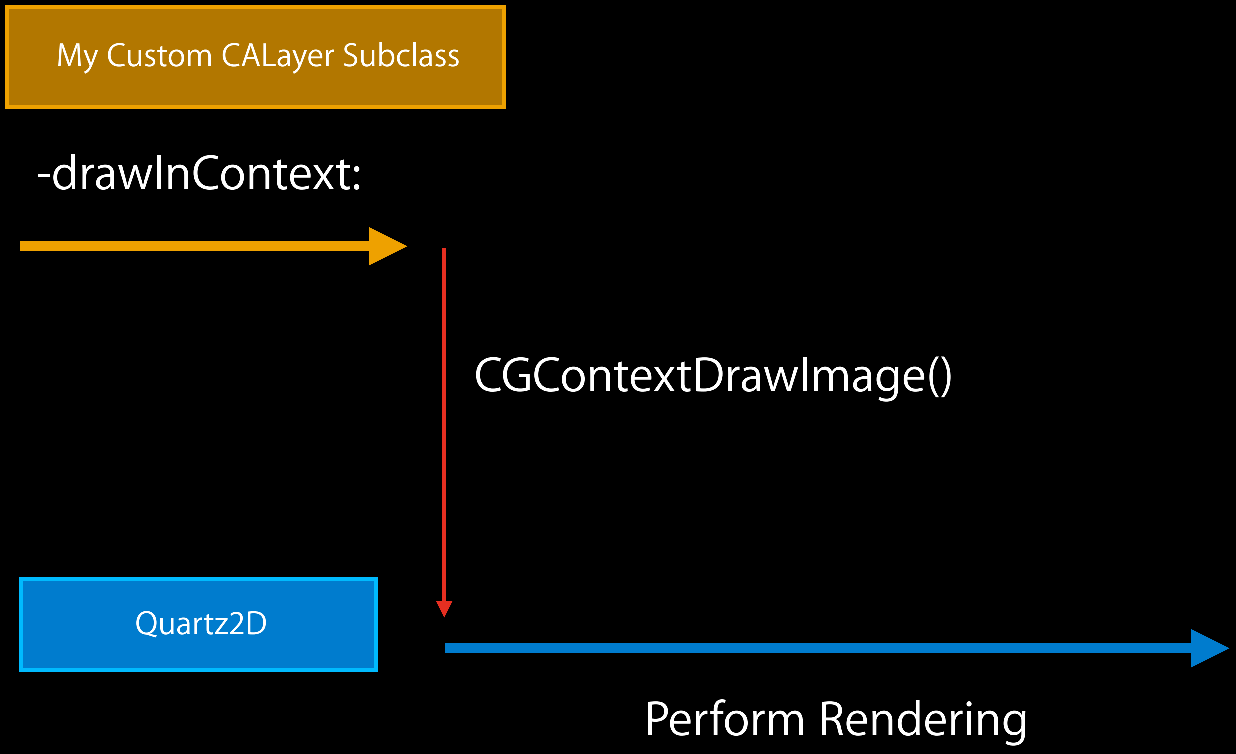
My Custom CALayer Subclass

-drawInContext:

CGContextDrawImage()

Quartz2D

CALayer Normal Drawing Mode



My Custom CALayer Subclass

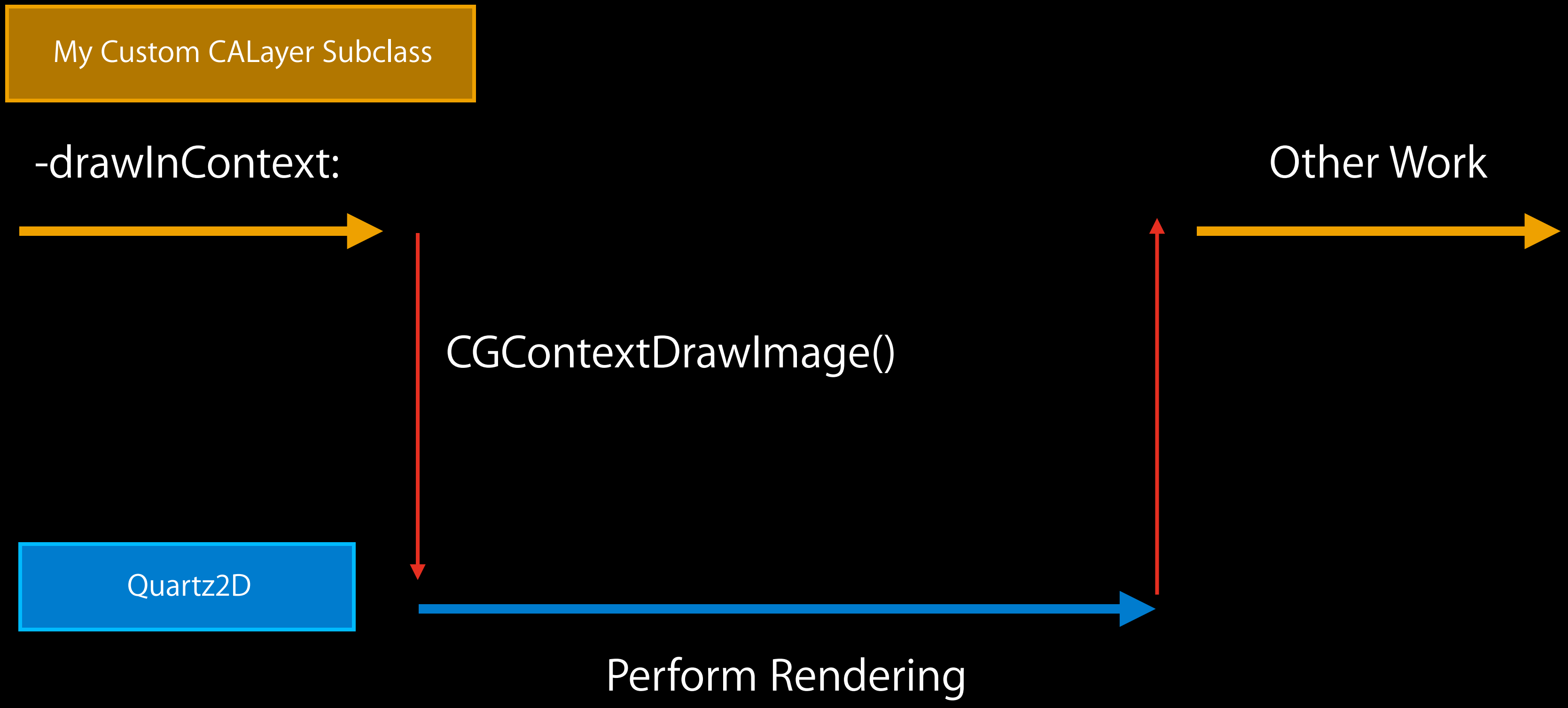
-drawInContext:

CGContextDrawImage()

Quartz2D

Perform Rendering

CALayer Normal Drawing Mode



My Custom CALayer Subclass

-drawInContext:

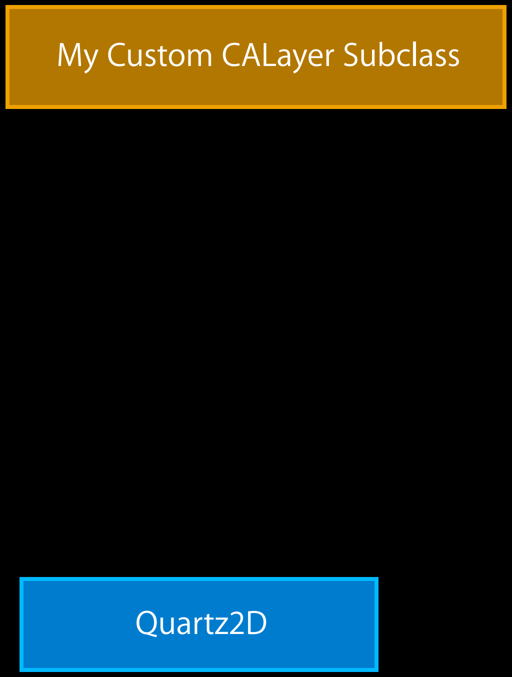
Other Work

CGContextDrawImage()

Quartz2D

Perform Rendering

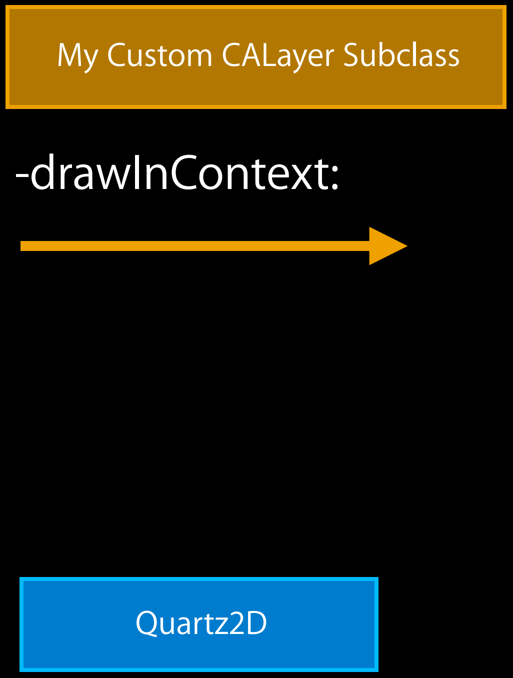
CALayer.drawsAsynchronously



My Custom CALayer Subclass

Quartz2D

CALayer.drawsAsynchronously

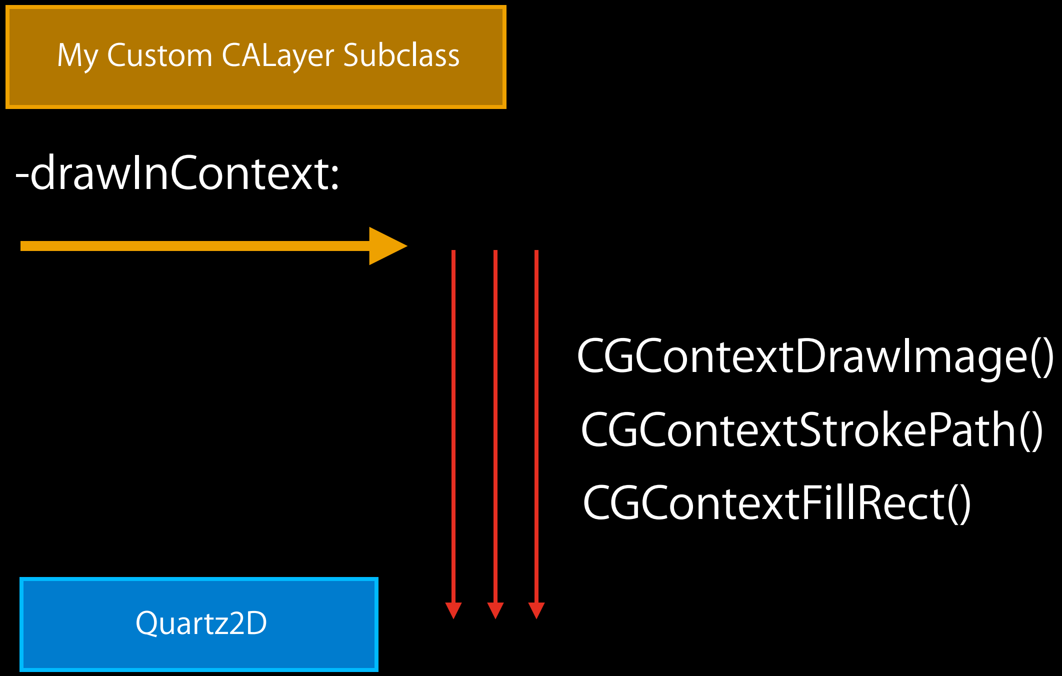


My Custom CALayer Subclass

-drawInContext:

Quartz2D

CALayer.drawsAsynchronously



My Custom CALayer Subclass

-drawInContext:

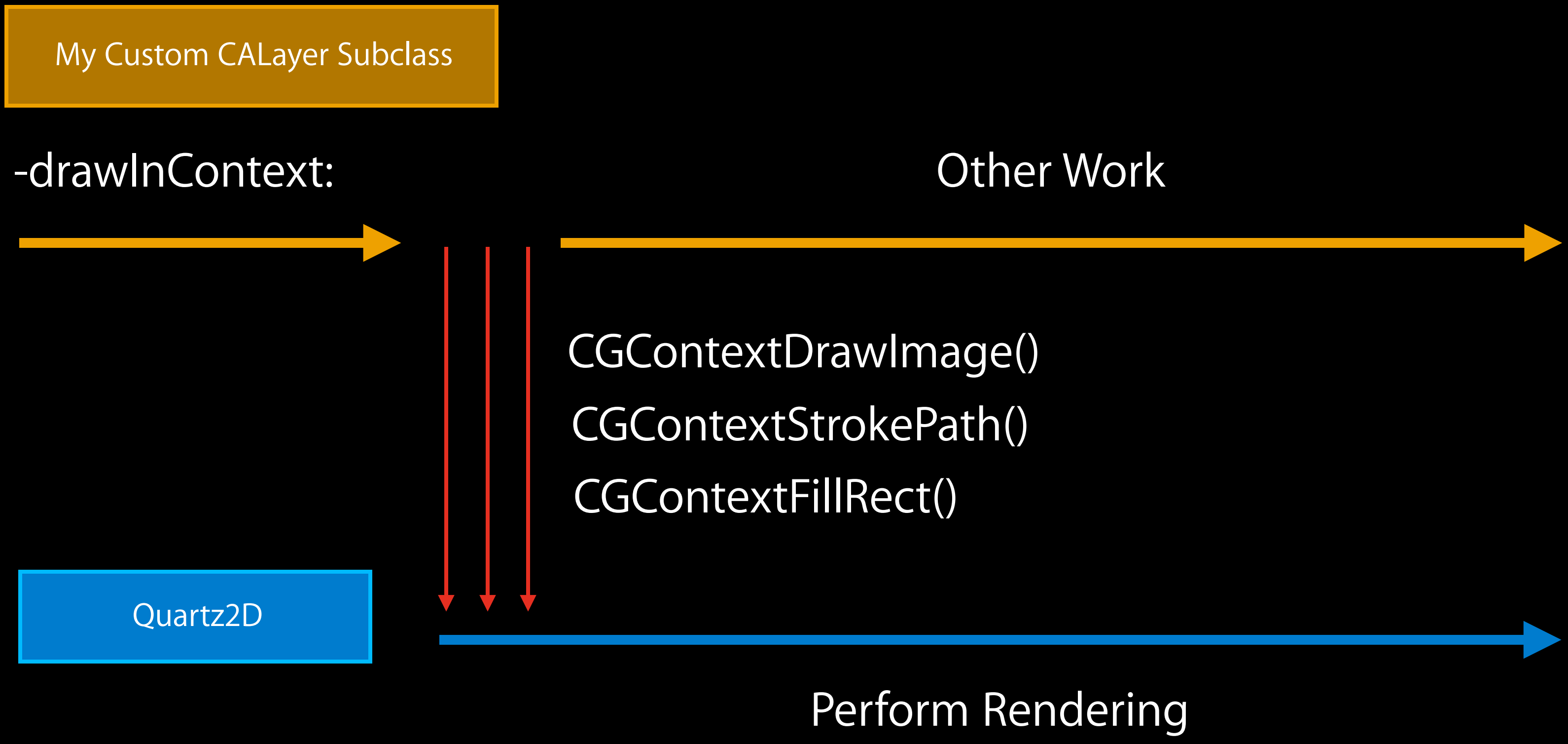
CGContextDrawImage()

CGContextStrokePath()

CGContextFillRect()

Quartz2D

CALayer.drawsAsynchronously



My Custom CALayer Subclass

-drawInContext:

Other Work

CGContextDrawImage()

CGContextStrokePath()

CGContextFillRect()

Quartz2D

Perform Rendering

CALayer.drawsAsynchronously

When should this be used

- Not always a win, disabled by default
- Usually helpful with large regions of the context being drawn with images, rectangles, shadings, etc.
- Really a case-by-case basis
- Measure, measure, measure

Demo

Final version of Finger Painting app for iPad

CGDisplayStream

CGDisplayStream

Display capture performance issues

- Round-trip copies from VRAM to RAM to VRAM kill performance
- 4x pixels greatly exacerbates this problem
- Ideally, captures should stay in VRAM for GPU-based processing: YUV conversion, scaling, etc.

CGDisplayStream

Traditional display capture scenario

CGDisplayStream

Traditional display capture scenario

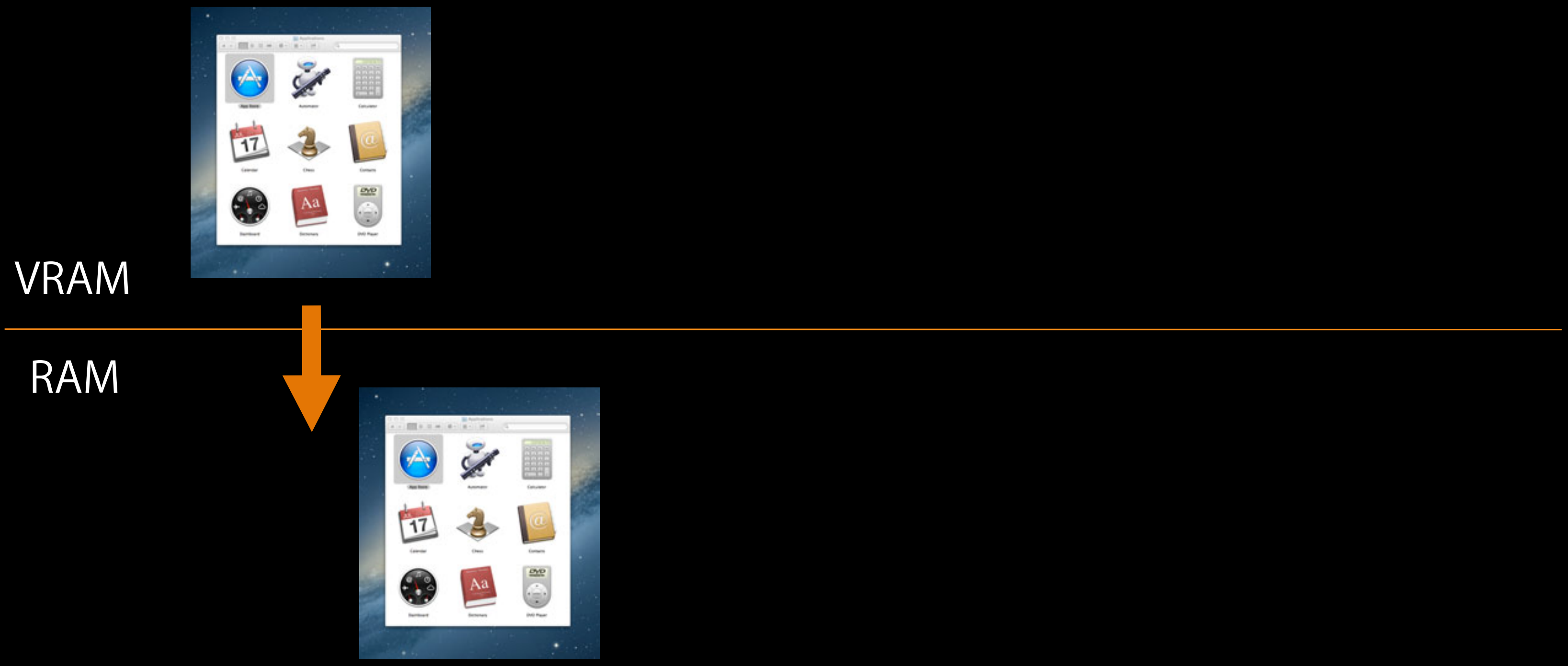
VRAM

RAM

Step 1: Framebuffer content starts in VRAM

CGDisplayStream

Traditional display capture scenario



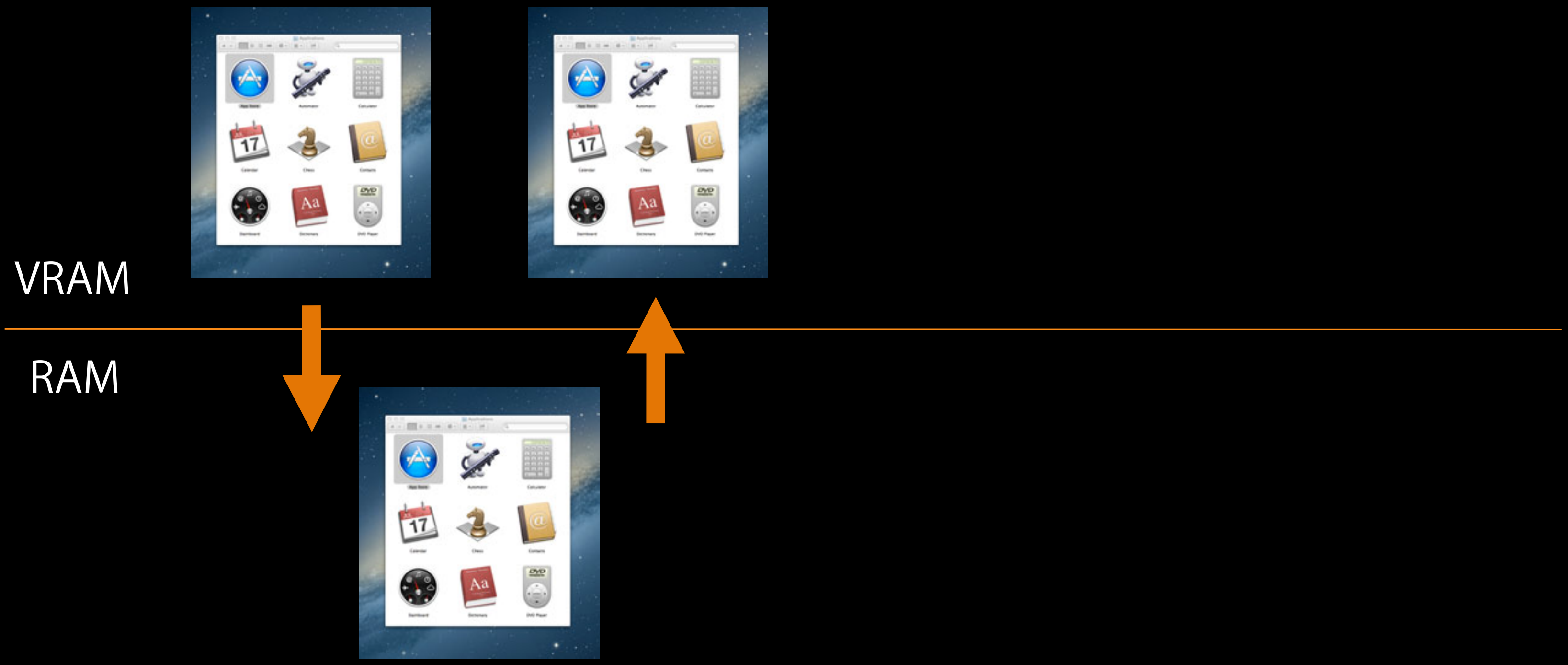
VRAM

RAM

Step 2: Display capture copies framebuffer data into RAM

CGDisplayStream

Traditional display capture scenario



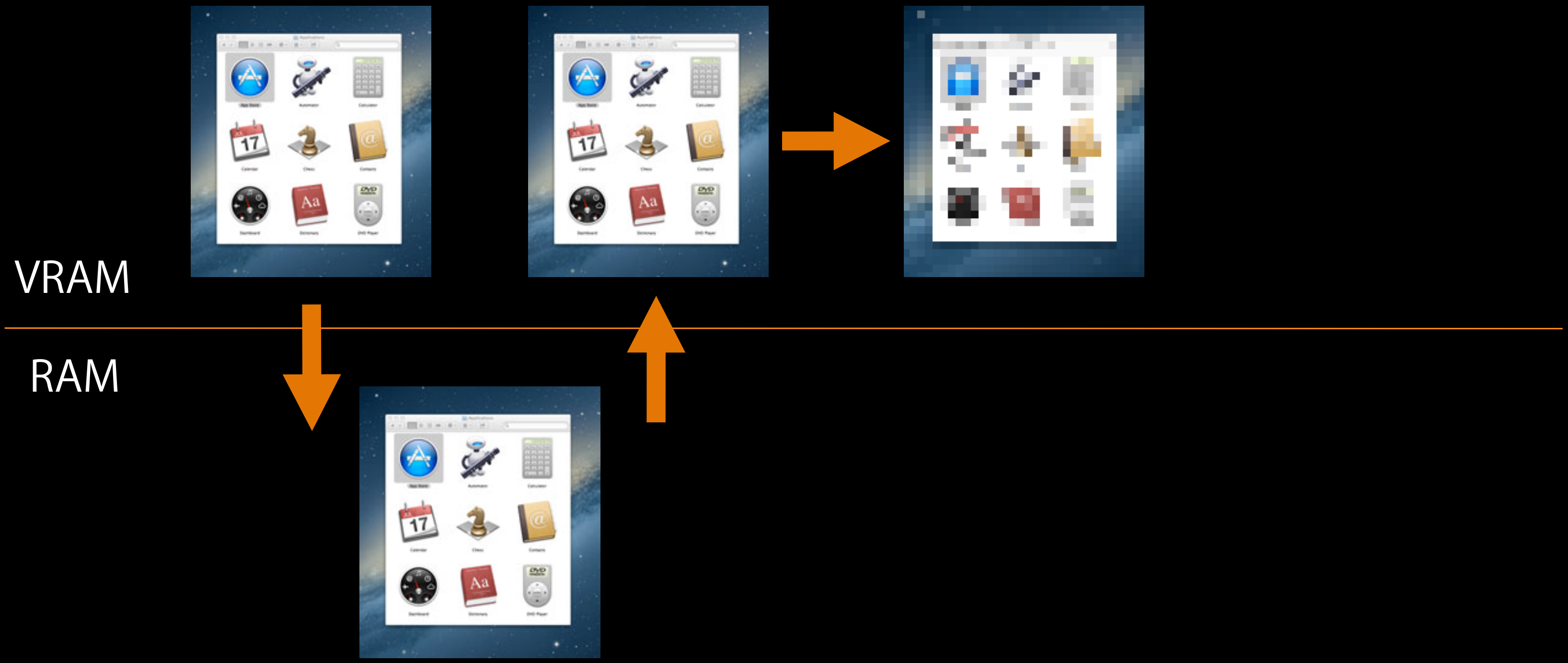
VRAM

RAM

Step 3: Capture data sent back to VRAM for processing

CGDisplayStream

Traditional display capture scenario



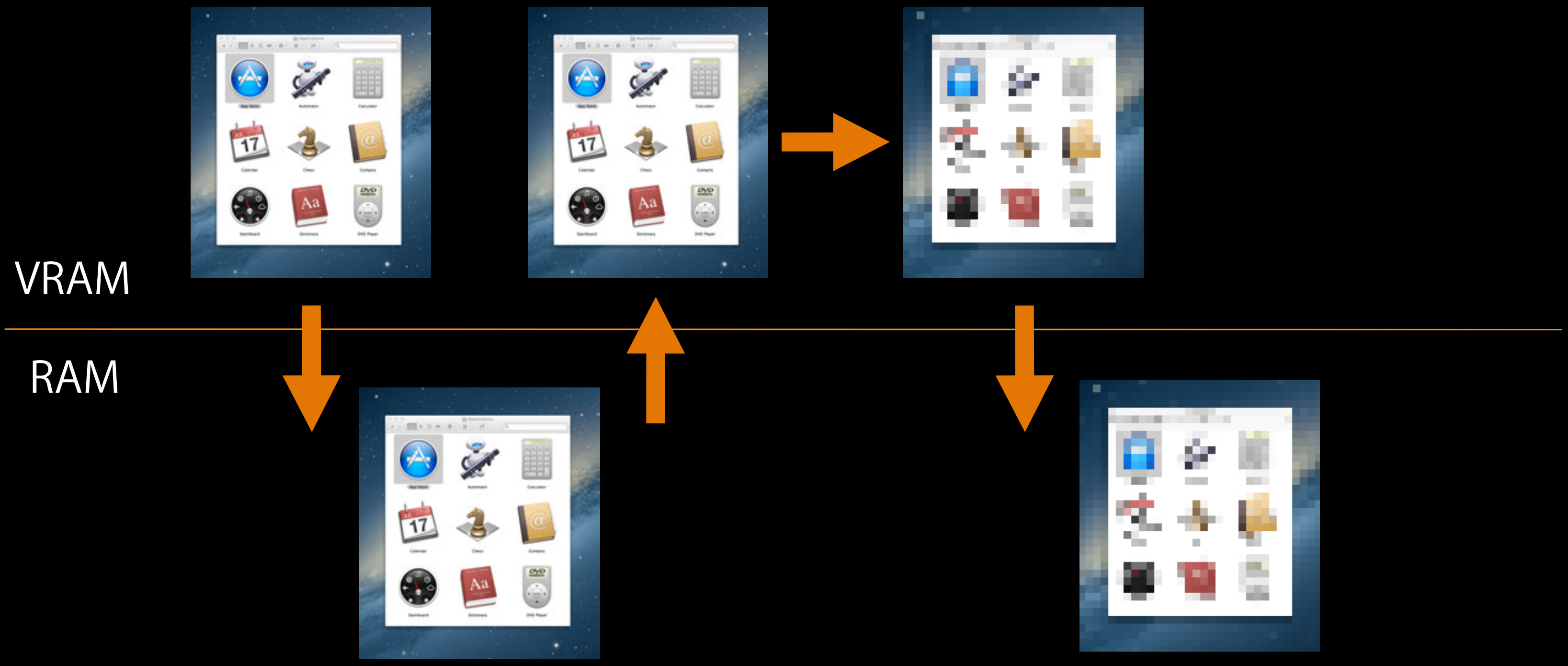
VRAM

RAM

Step 4: Process the capture data in the GPU

CGDisplayStream

Traditional display capture scenario



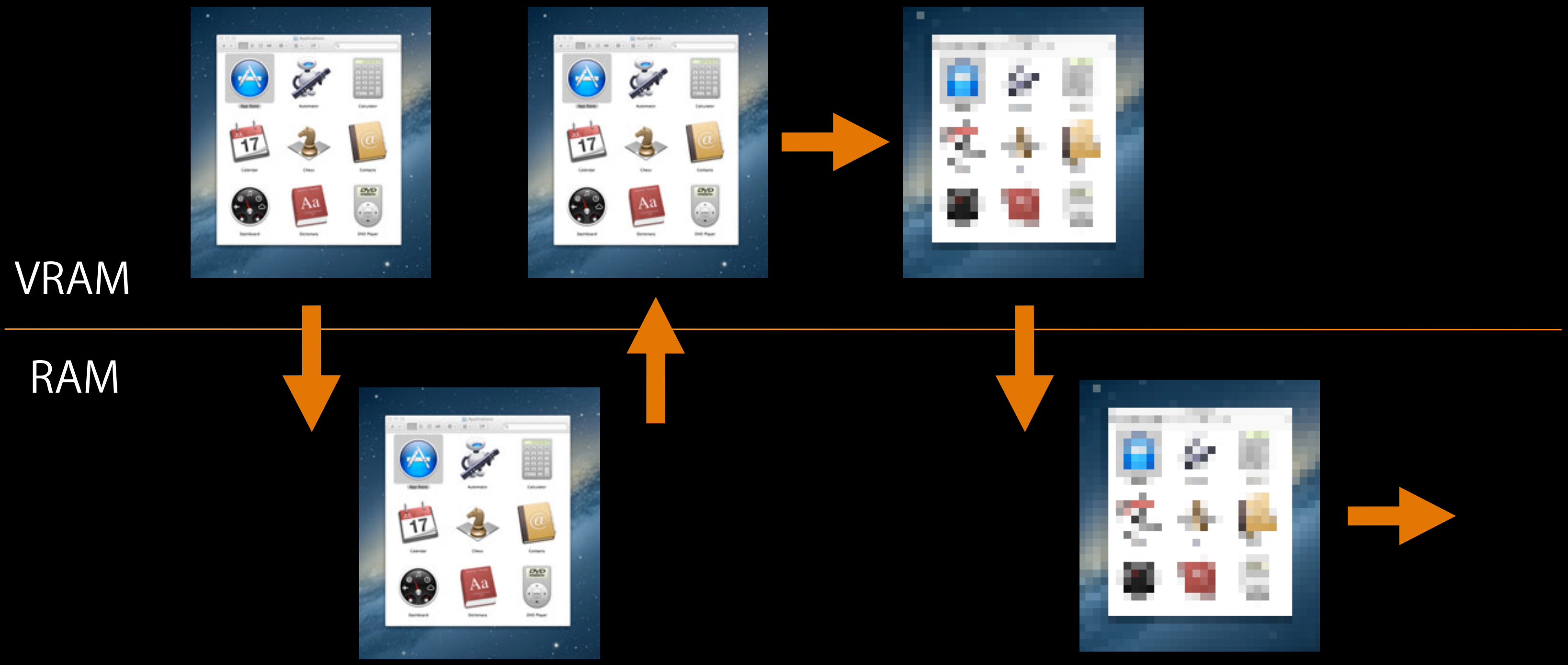
VRAM

RAM

Step 5: Pull processed data back out of VRAM

CGDisplayStream

Traditional display capture scenario



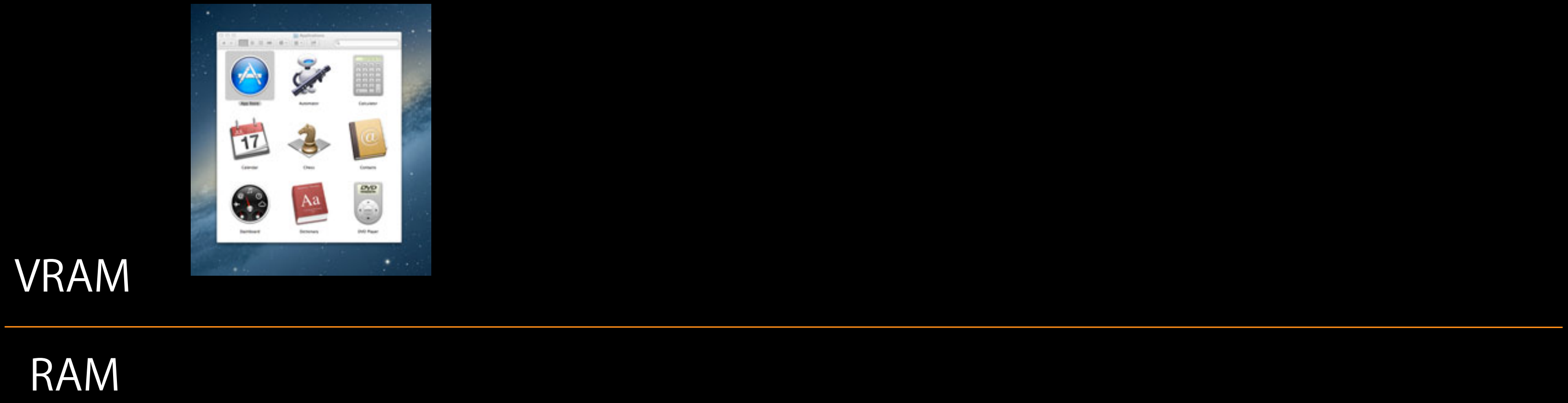
Step 6: Capture data is ready for use by application

CGDisplayStream

High-performance display capture scenario

CGDisplayStream

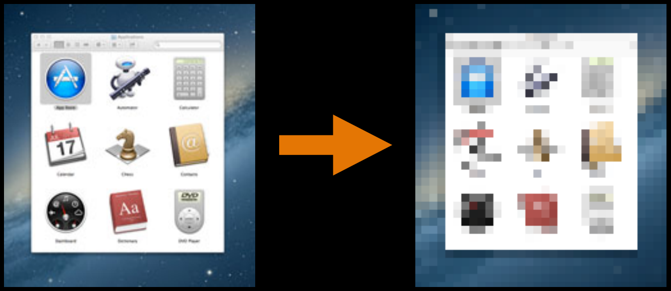
High-performance display capture scenario



Step 1: Framebuffer content starts in VRAM

CGDisplayStream

High-performance display capture scenario



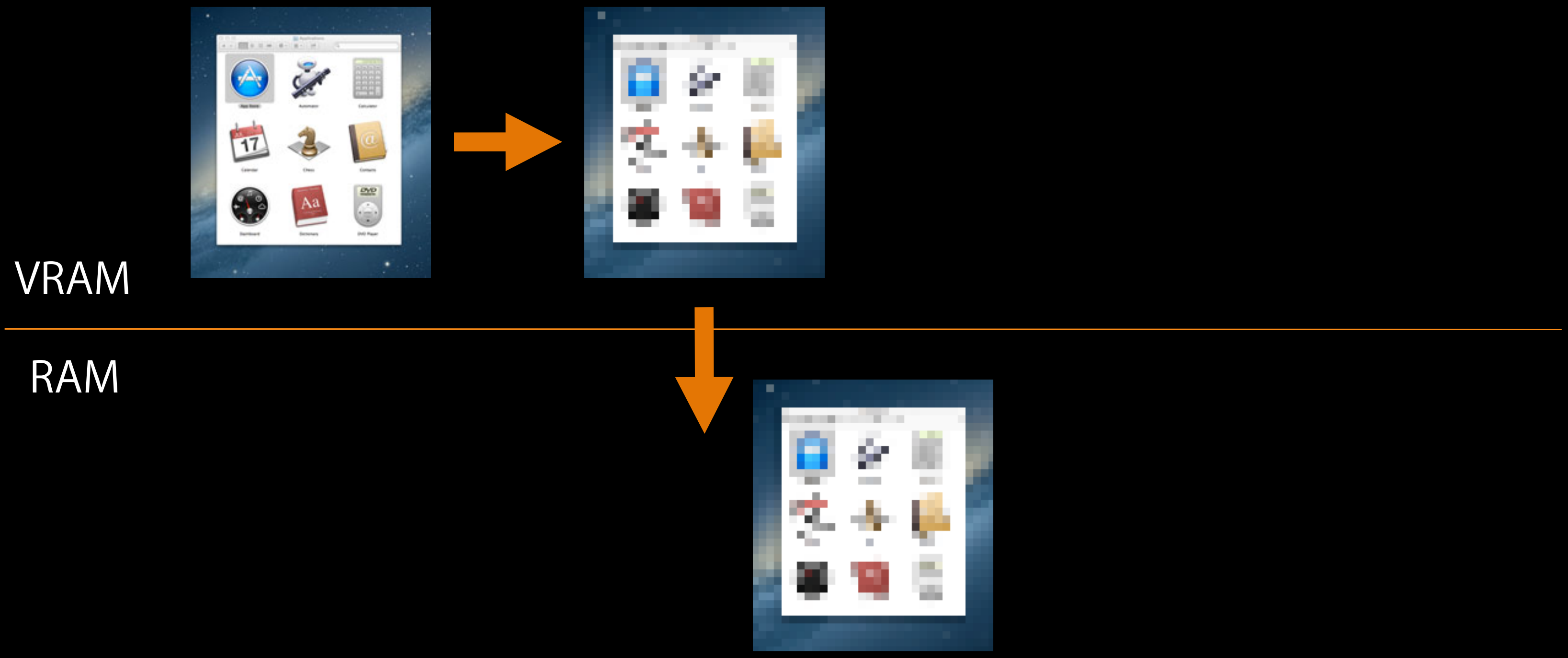
VRAM

RAM

Step 2: Data is captured and processed without leaving VRAM

CGDisplayStream

High-performance display capture scenario



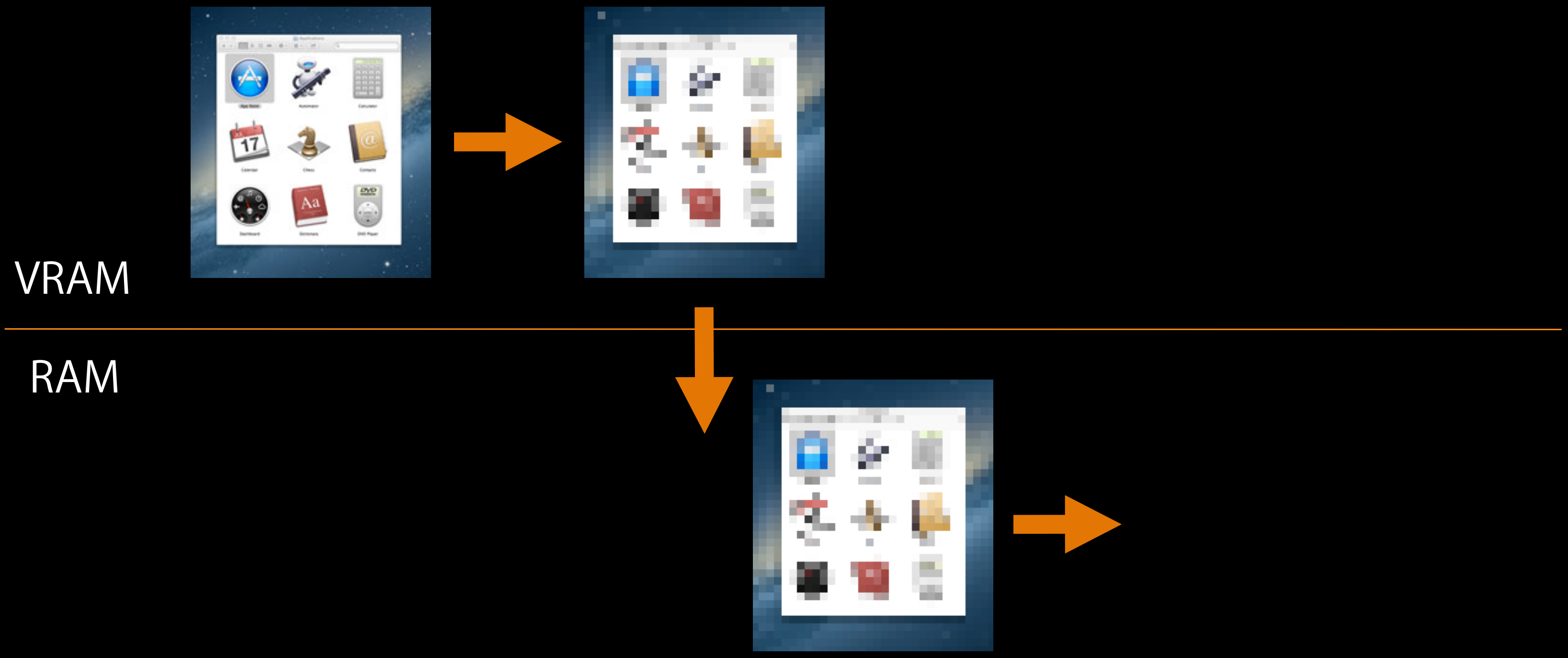
VRAM

RAM

Step 3: Pull processed data out of VRAM

CGDisplayStream

High-performance display capture scenario



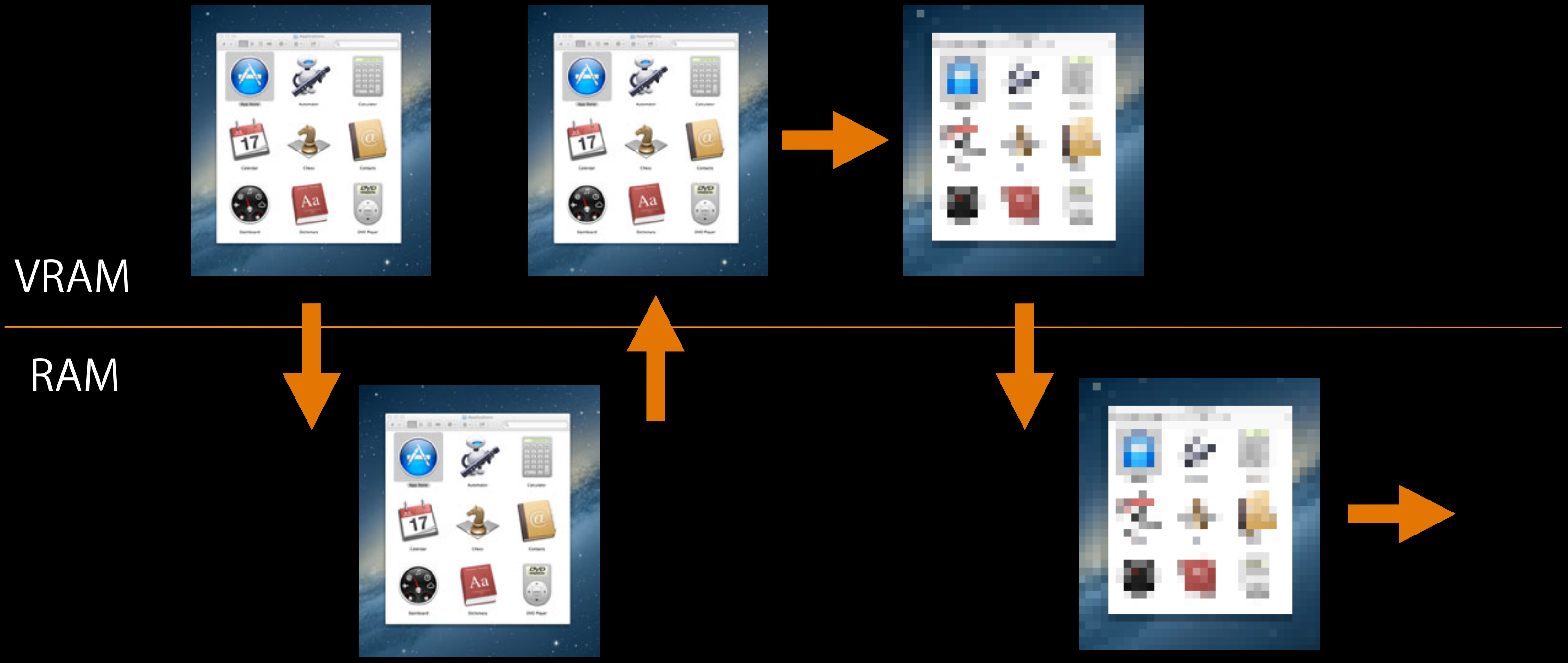
VRAM

RAM

Step 4: Capture data is ready for use by application

CGDisplayStream

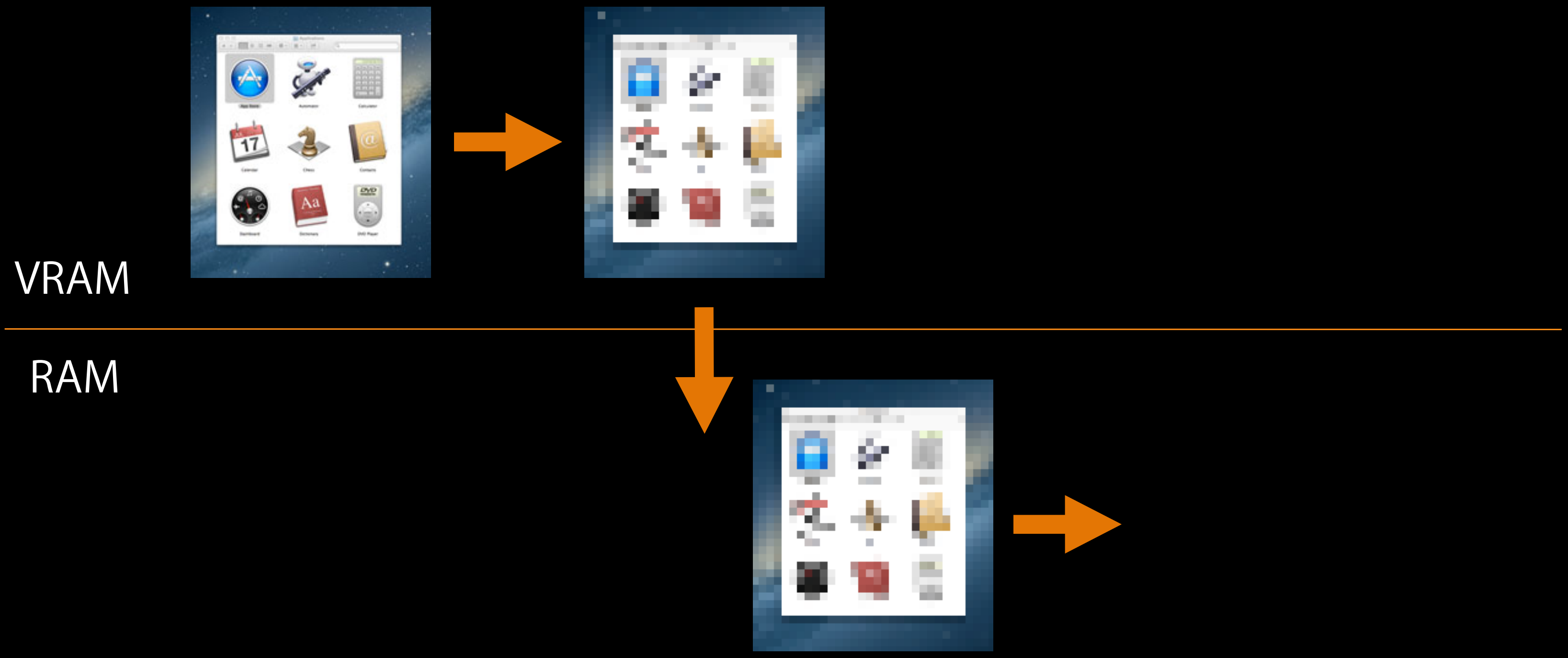
Traditional display capture scenario



Step 6: Capture data is ready for use by application

CGDisplayStream

High-performance display capture scenario



Step 4: Capture data is ready for use by application

CGDisplayStream

Existing display capture techniques

- CGDisplayCreateImage for capturing single frames

CGDisplayStream

Existing display capture techniques

- CGDisplayCreateImage for capturing single frames
- AV Foundation for recording to a QuickTime file

CGDisplayStream

Existing display capture techniques

- CGDisplayCreateImage for capturing single frames
- AV Foundation for recording to a QuickTime file
- Raw framebuffer access: Highly deprecated, highly unreliable

CGDisplayStream

Existing display capture techniques

- CGDisplayCreateImage for capturing single frames
- AV Foundation for recording to a QuickTime file
- Raw framebuffer access: Highly deprecated, highly unreliable

CGDisplayStream

Existing display capture techniques

- CGDisplayCreateImage for capturing single frames
- AV Foundation for recording to a QuickTime file
- ~~Raw framebuffer access: Highly deprecated, highly unreliable~~

CGDisplayStream

Introducing CGDisplayStream

- New real-time display capture API
- OS X Mountain Lion only
- Can be used for non-interactive applications:
  One-shot screen captures, screen recording
- Can be used for interactive, real-time applications:
  Remote display, USB projectors

CGDisplayStream

When to use CGDisplayStream

- Real-time processing of screen updates
- Integrated with CFRunLoop and dispatch queues
- GPU-based image scaling and colorspace conversion
- Provides update rects for each captured frame

CGDisplayStream

Creating the DisplayStream

```
CGDisplayStreamRef CGDisplayStreamCreate(CGDirectDisplayID display,
                                          size_t outputWidth,
                                          size_t outputHeight,
                                          int32_t pixelFormat,
                                          CFDictionaryRef properties,
                        CGDisplayStreamFrameAvailableHandler handler)
```

CGDisplayStream

CGDisplayStream properties

- kCGDisplayStreamQueueDepth—defaults to 3, should be no more than 8
- kCGDisplayStreamSourceRect
- kCGDisplayStreamPreserveAspectRatio
- kCGDisplayStreamColorSpace

CGDisplayStream

Managing the DisplayStream

```
CFRunLoopSourceRef
CGDisplayStreamGetRunLoopSource(CGDisplayStreamRef displayStream)


CGError
CGDisplayStreamStart(CGDisplayStreamRef displayStream)


CGError
CGDisplayStreamStop(CGDisplayStreamRef displayStream)
```

CGDisplayStream

Processing the DisplayStream

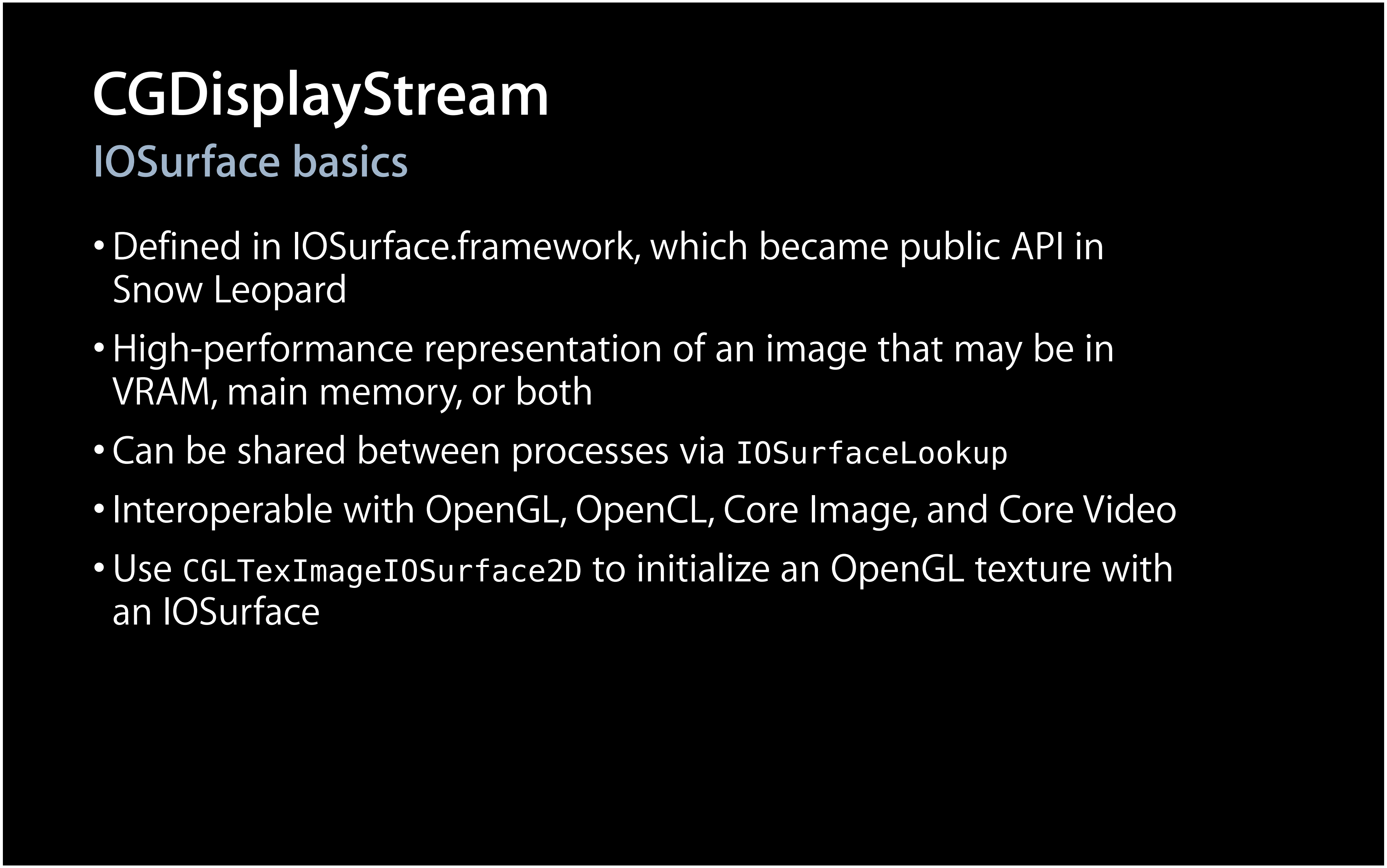
```
void
^CGDisplayStreamFrameAvailableHandler(CGDisplayStreamFrameStatus status,
                                       uint64_t displayTime,
                                       IOSurfaceRef frameSurface,
                                       CGDisplayStreamUpdateRef updateRef);
```

CGDisplayStream

Examining the DisplayStream

```
const CGRect *
CGDisplayStreamUpdateGetRects(CGDisplayStreamUpdateRef updateRef,
                              CGDisplayStreamUpdateRectType rectType,
                              size_t *rectCount)
```

```
CGDisplayStreamUpdateRef
CGDisplayStreamUpdateCreateMergedUpdate(CGDisplayStreamUpdateRef firstUpdate,
                                        CGDisplayStreamUpdateRef secondUpdate)
```



CGDisplayStream

IOSurface basics

- Defined in IOSurface.framework, which became public API in Snow Leopard
- High-performance representation of an image that may be in VRAM, main memory, or both
- Can be shared between processes via `IOSurfaceLookup`
- Interoperable with OpenGL, OpenCL, Core Image, and Core Video
- Use `CGLTexImageIOSurface2D` to initialize an OpenGL texture with an IOSurface

Demo

CGDisplayStream in practice

More Information

Allan Schaffer

Graphics and Game Technologies Evangelist
aschaffer@apple.com

Mailing List

quartz-dev@lists.apple.com

Documentation

https://developer.apple.com/technologies/mac/graphics-and-animation.html

High-Resolution Guidelines for OS X

http://developer.apple.com/library/mac/#documentation/GraphicsAnimation/Conceptual/HighResolutionOSX

Apple Developer Forums

http://devforums.apple.com

Related Sessions

Introduction to High Resolution on OS X	Presidio Wednesday 9:00AM
Layer-Backed Views: AppKit + Core Animation	Nob Hill Wednesday 10:15AM
Delivering Web Content on High Resolution Displays	Nob Hill Wednesday 11:30AM

Labs

High Resolution on OS X Lab	Essentials Lab B Wednesday 11:30AM
Quartz 2D Lab	Graphics, Media & Games Lab B Wednesday 9:00AM
Quartz 2D Lab	Graphics, Media & Games Lab C Thursday 9:00AM
Core Animation Lab	Graphics, Media & Games Lab A Wednesday 9:00AM
Core Animation Lab	Graphics, Media & Games Lab C Thursday 11:30AM

WWDC2012