#WWDC18

Image and Graphics Best Practices

Session 219

Kyle Sluder, iOS System Experience

UIImage and UIImageView

Custom drawing with UIKit

Advanced CPU and GPU techniques

Memory

CPU

Memory

CPU

Battery Life

Responsiveness

Memory

CPU

Battery Life

Responsiveness

Memory

CPU

Battery Life

Responsiveness


Cancel
Done

UIImage

Content


UIImage
Content
UIImage
Iconography
Cancel
Done


UIImage
Iconography
UIImage
Content
UIImageView


Model
UIImage
View
UIImageView


Model
Load
View
Render

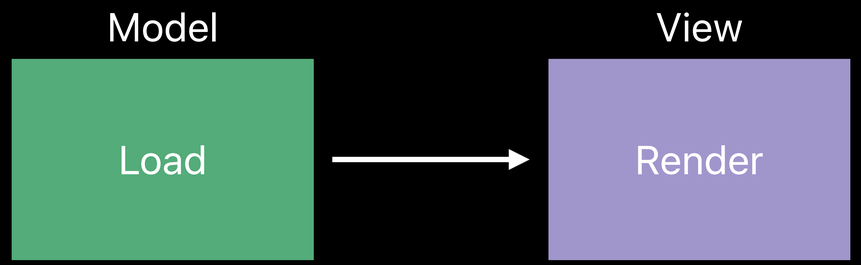
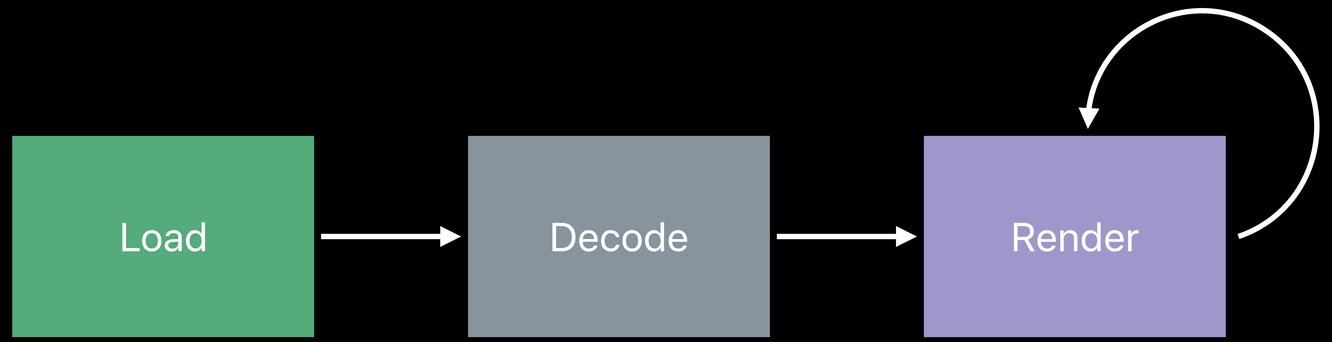
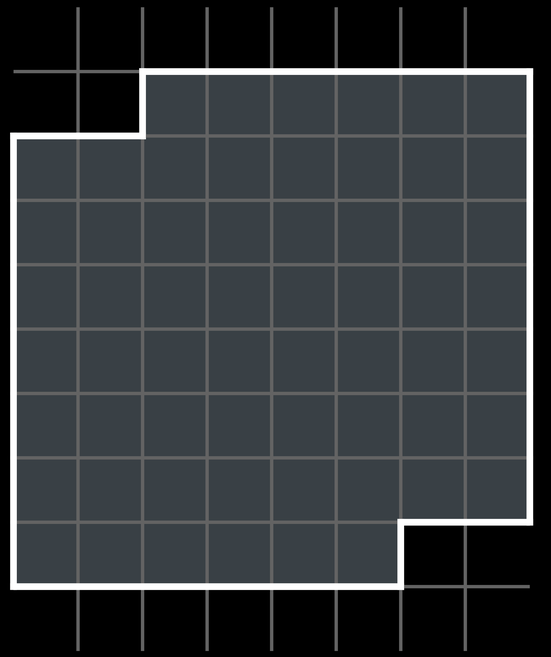

Model
View
Load
Render

Image Rendering Pipeline



Buffers

Contiguous region of memory



Buffers

Contiguous region of memory

Often viewed as sequence of elements

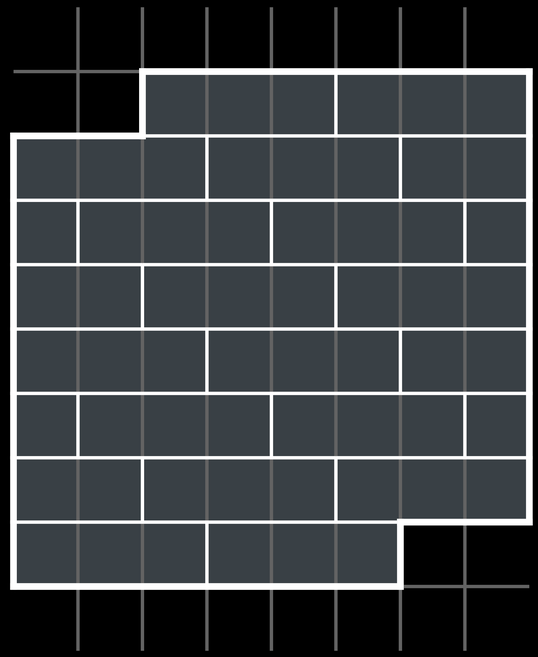


Image Buffers

In-memory representation of an image

Each element describes color of a single pixel

Buffer size is proportional to image size

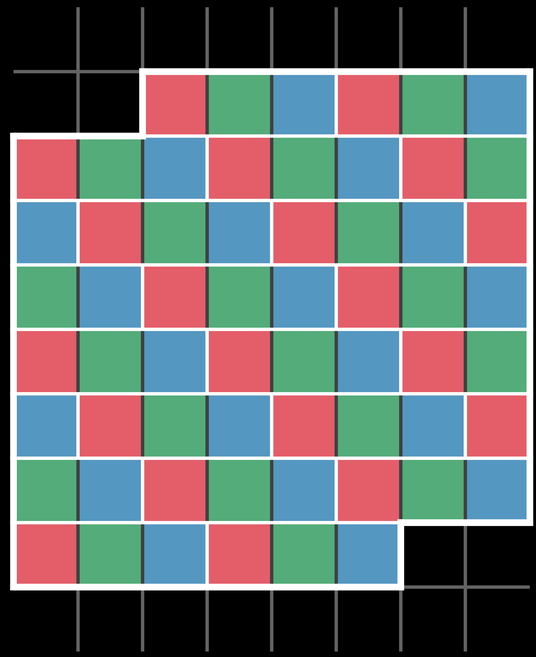
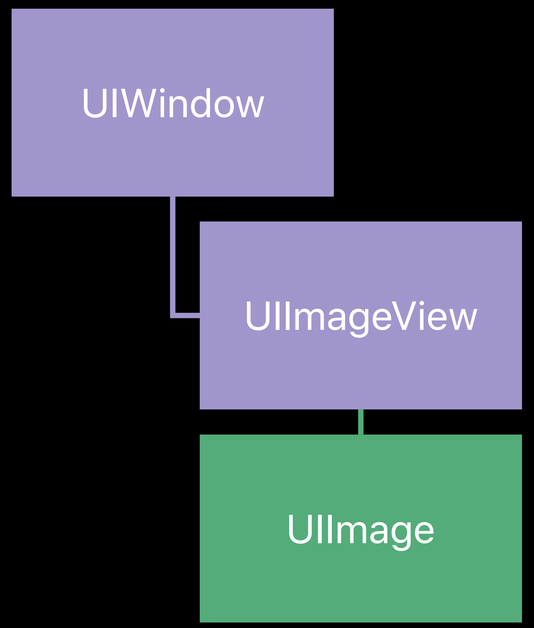
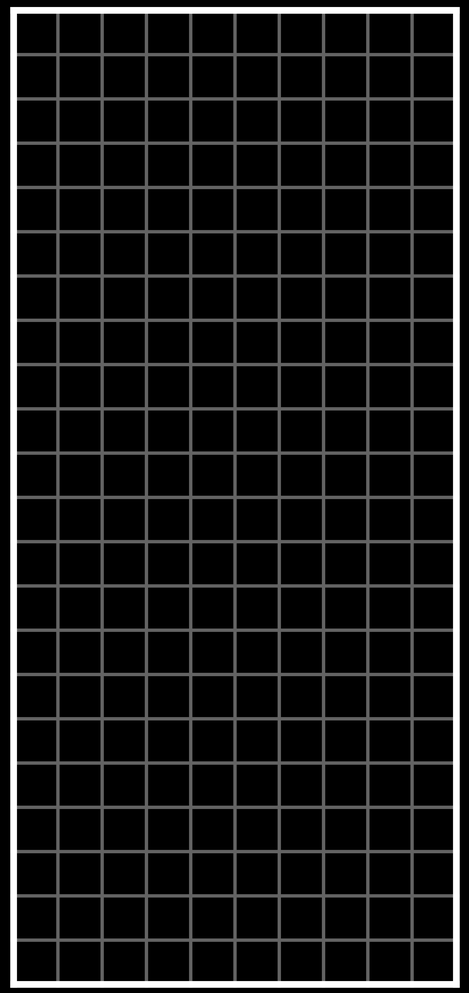


Image Buffers

The frame buffer





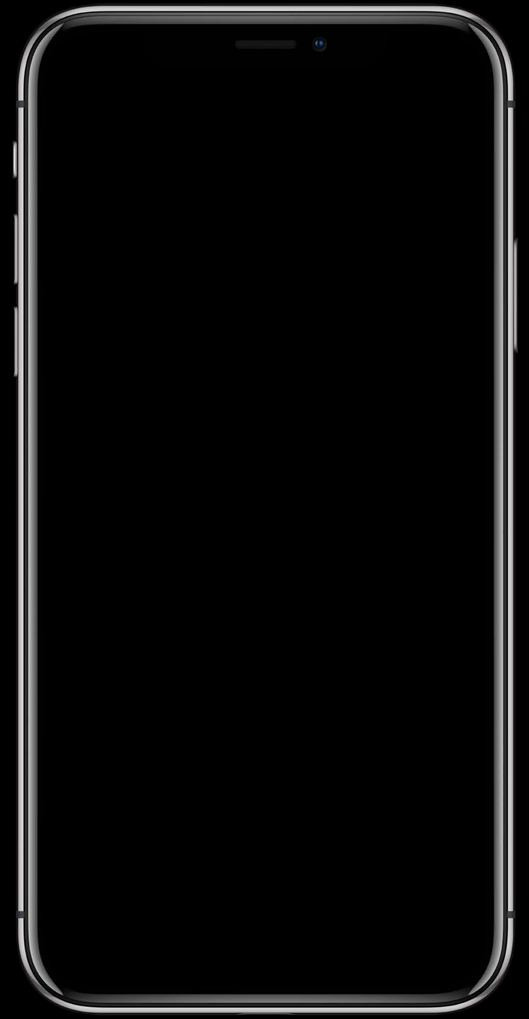


Image Buffers

The frame buffer

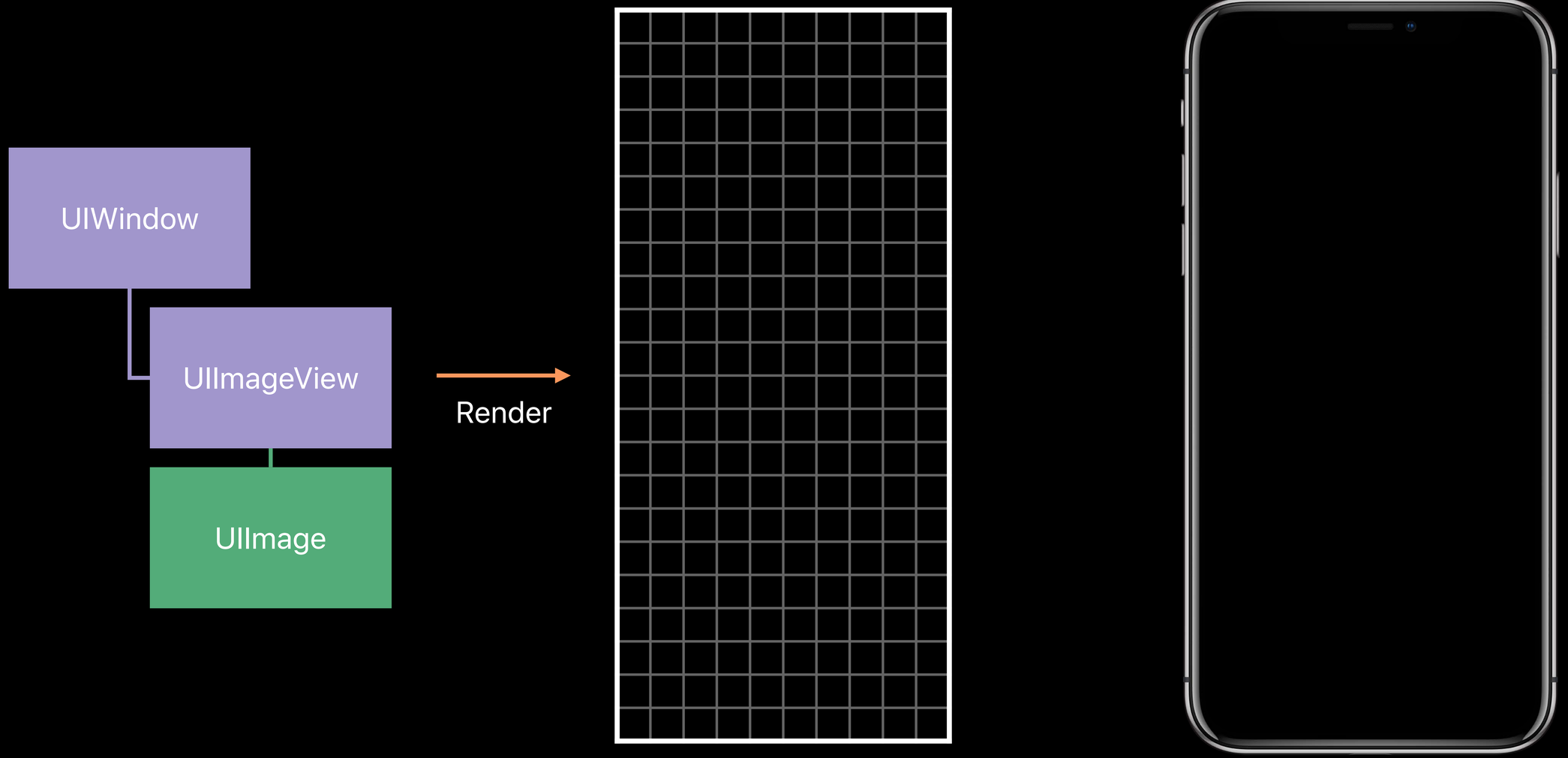


Image Buffers

The frame buffer

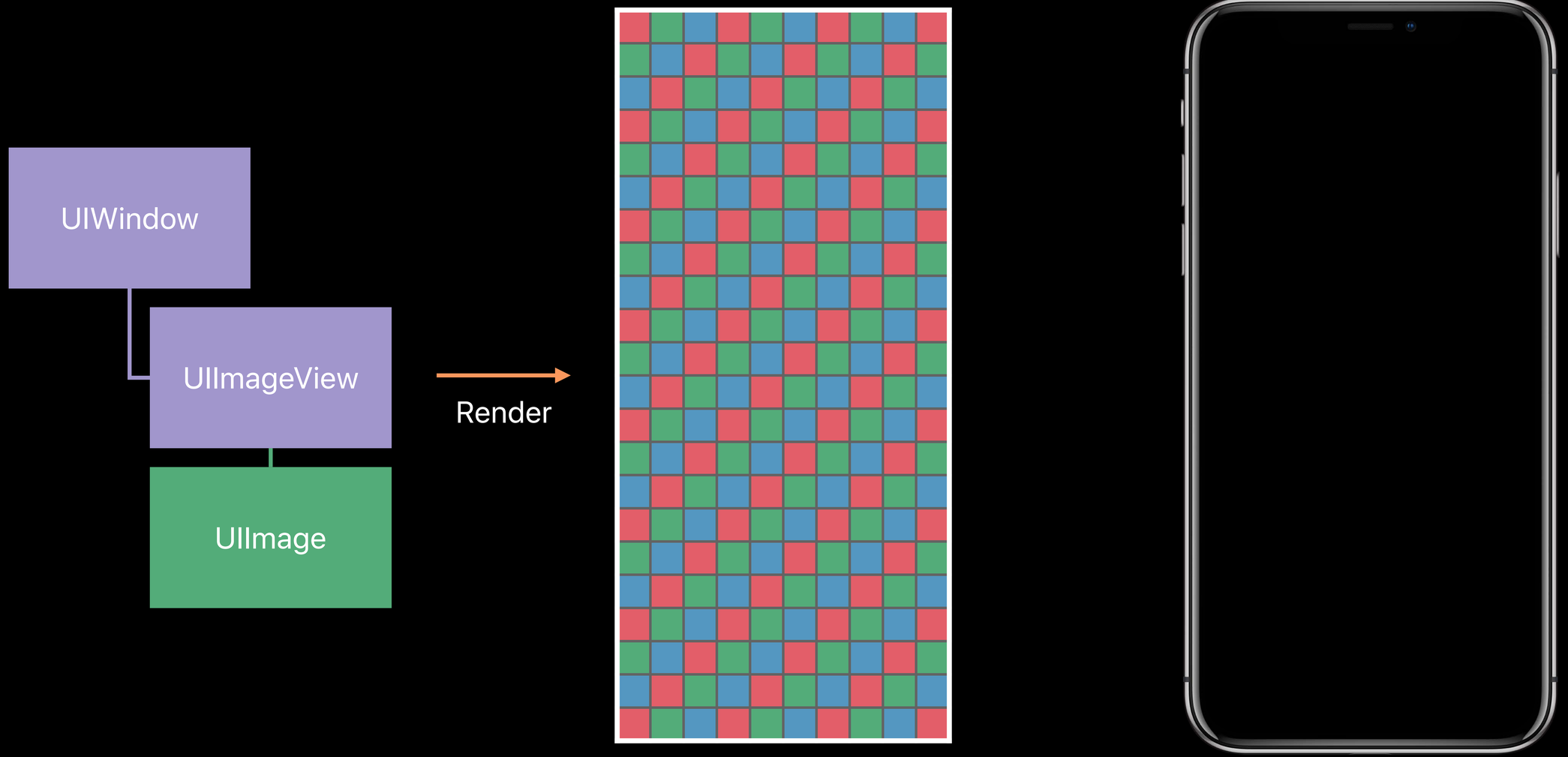


Image Buffers

The frame buffer

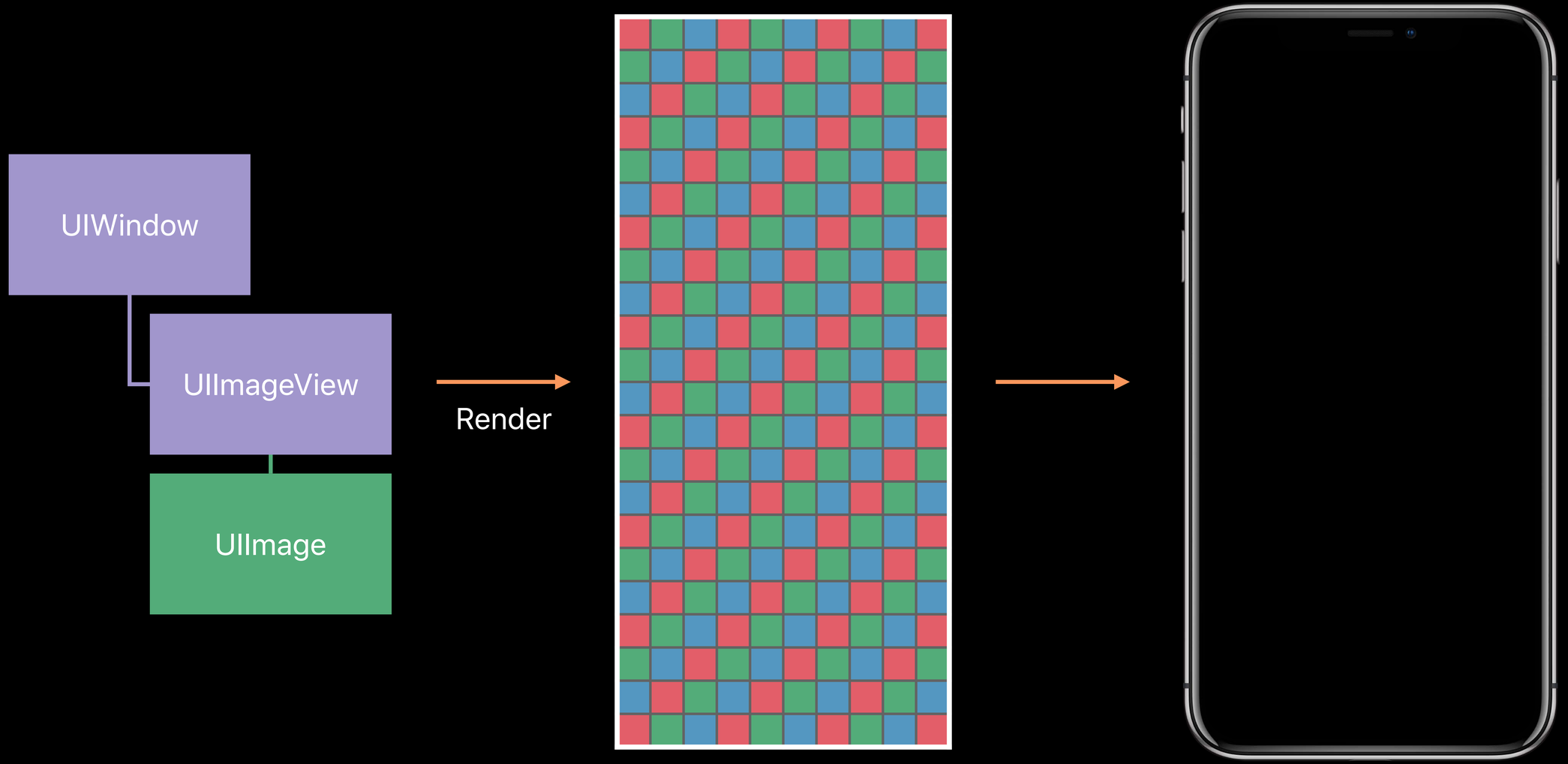


Image Buffers

The frame buffer

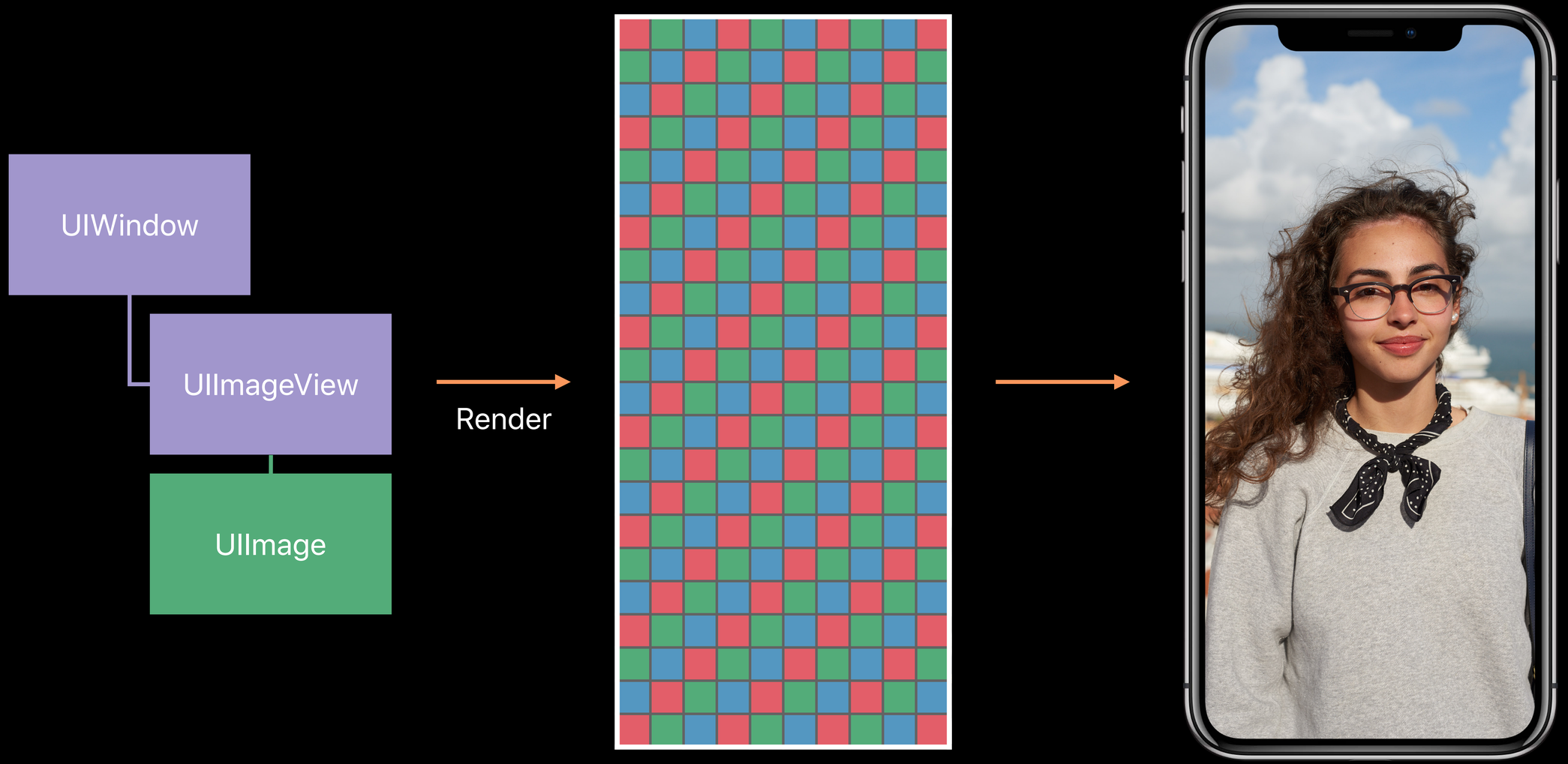


Image Buffers

The frame buffer

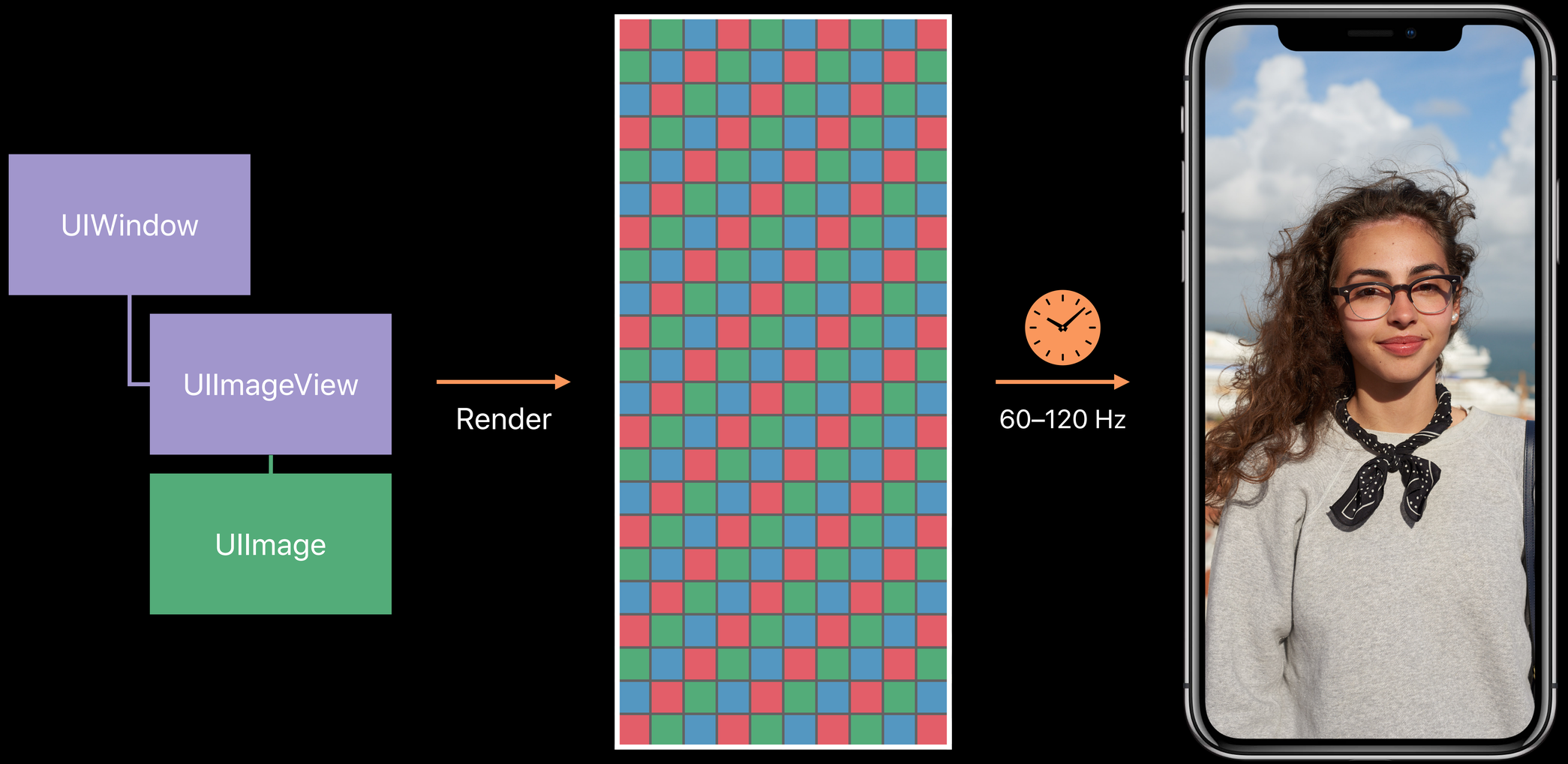


Image Buffers

The frame buffer

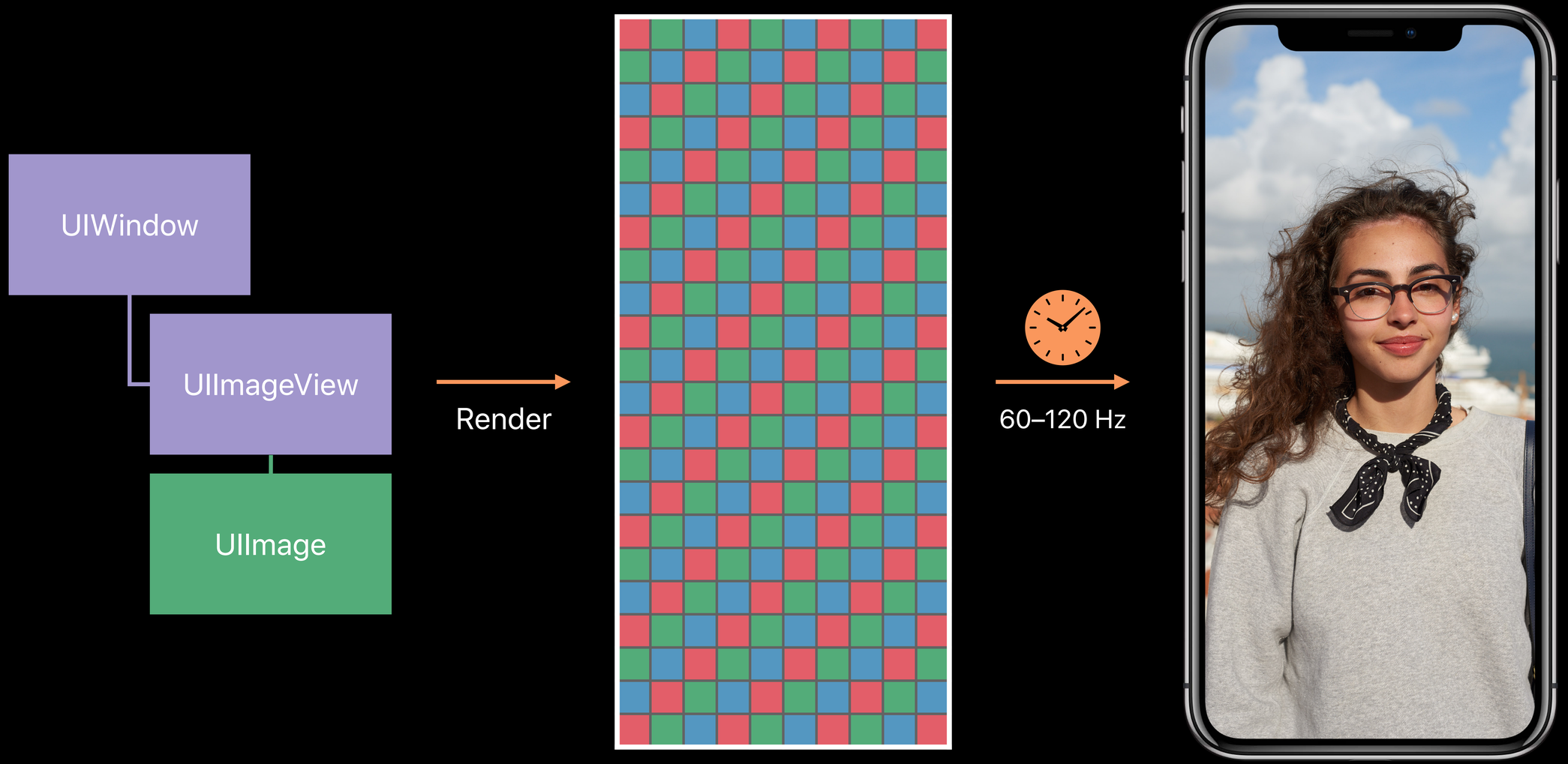


Image Buffers

The frame buffer

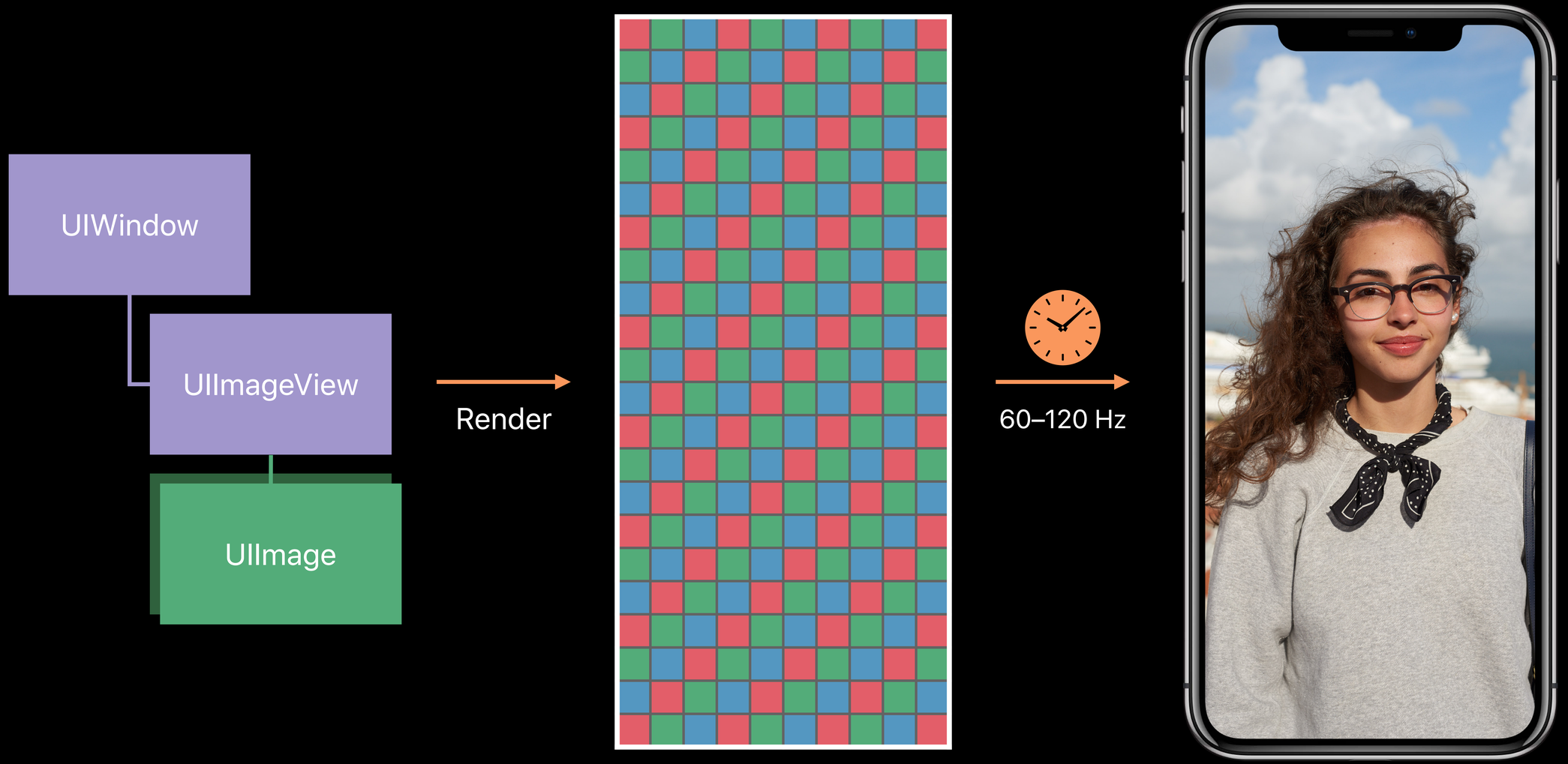


Image Buffers

The frame buffer

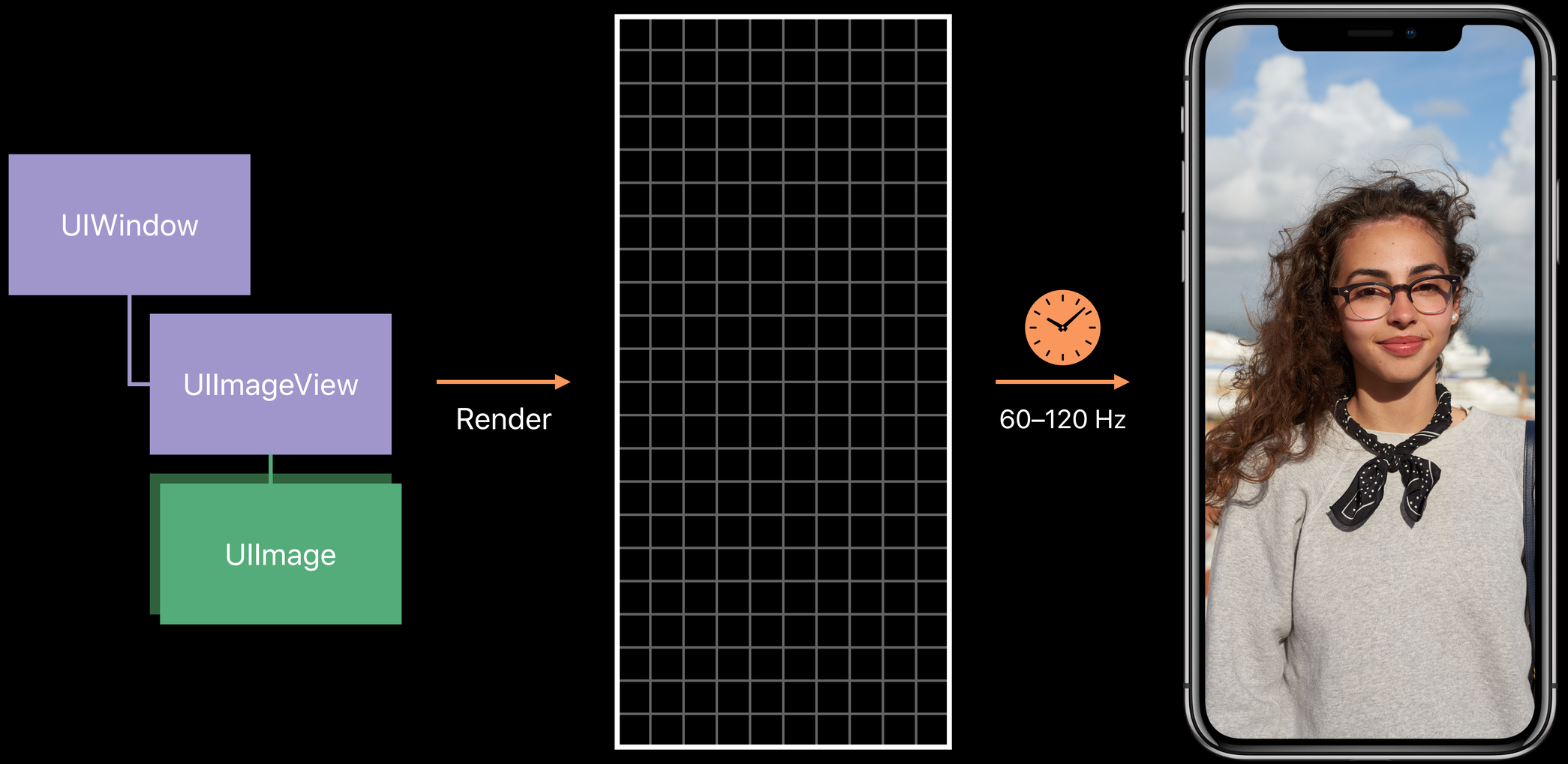


Image Buffers

The frame buffer

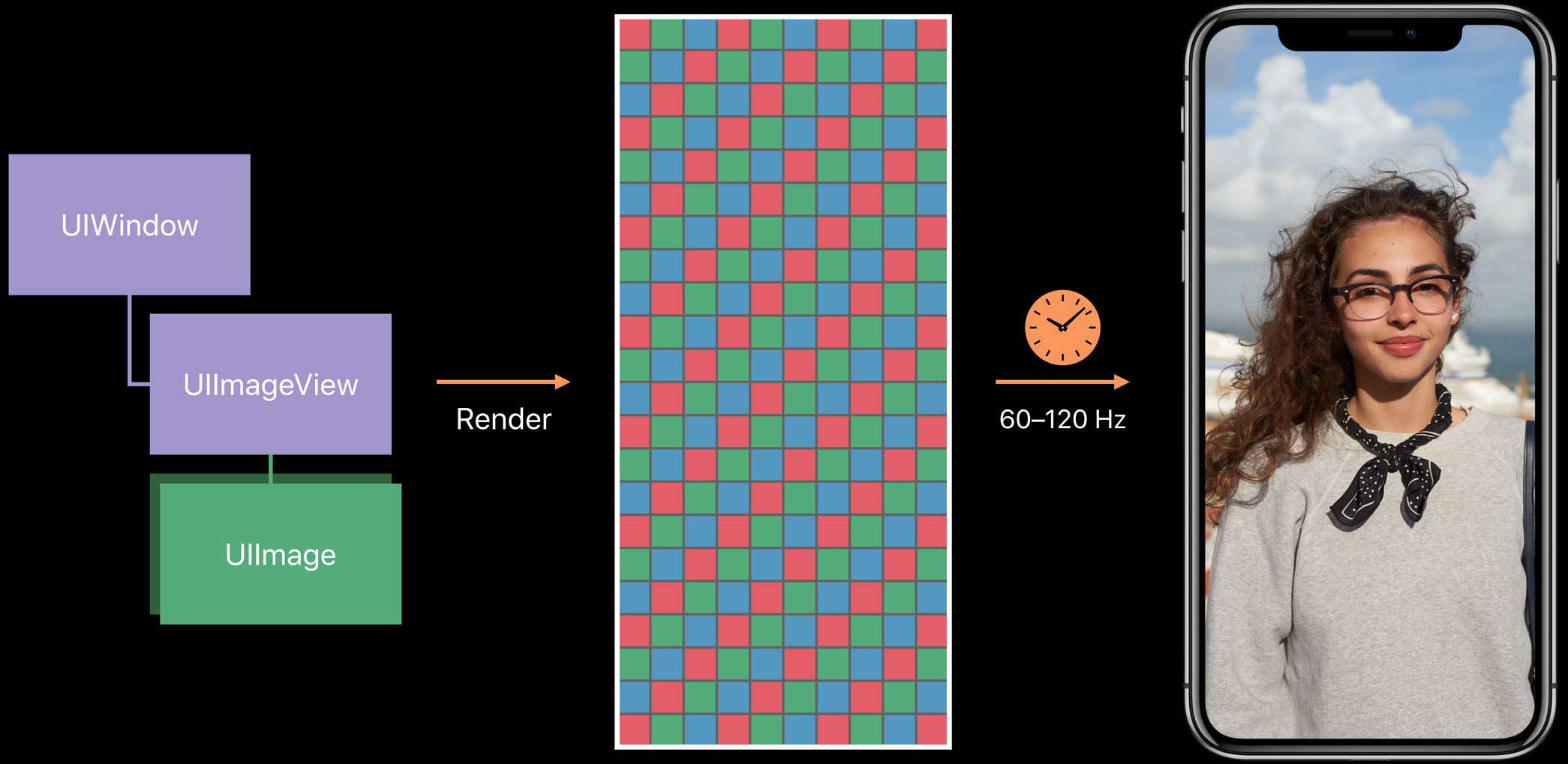
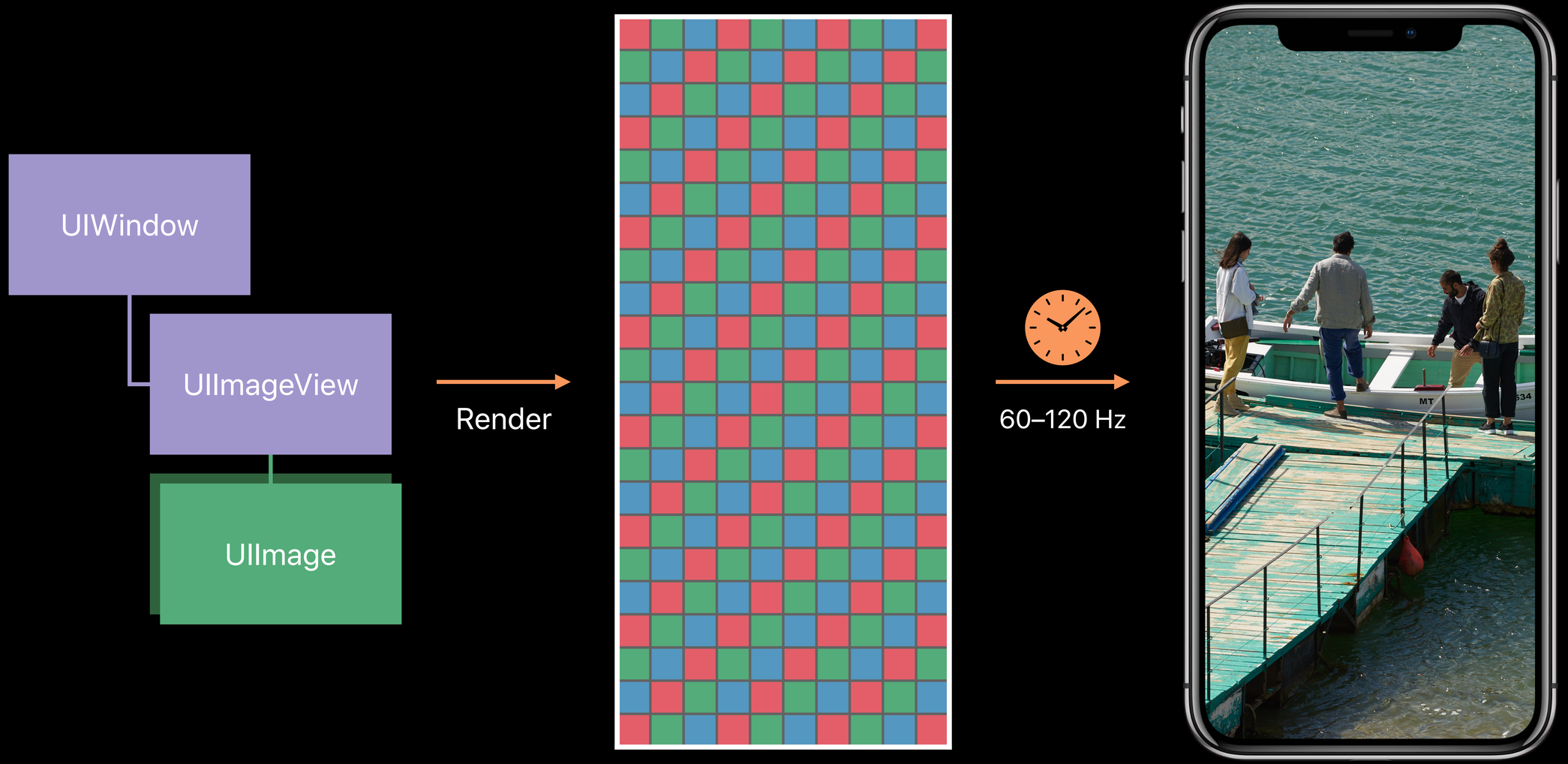


Image Buffers

The frame buffer



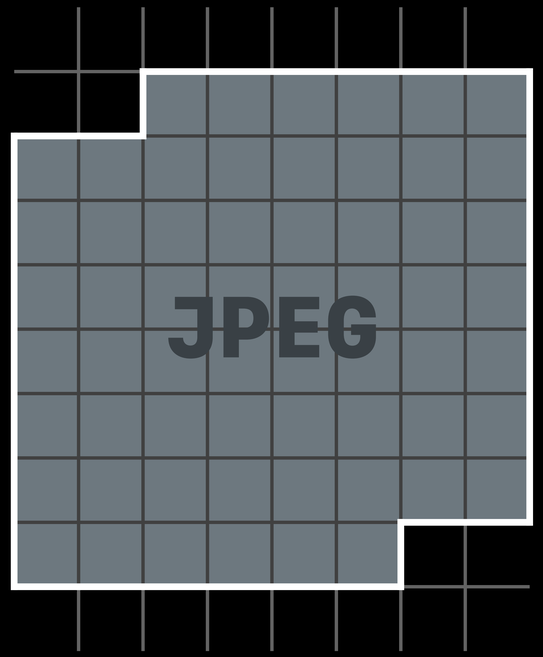
Data Buffers

Store contents of an image file in memory

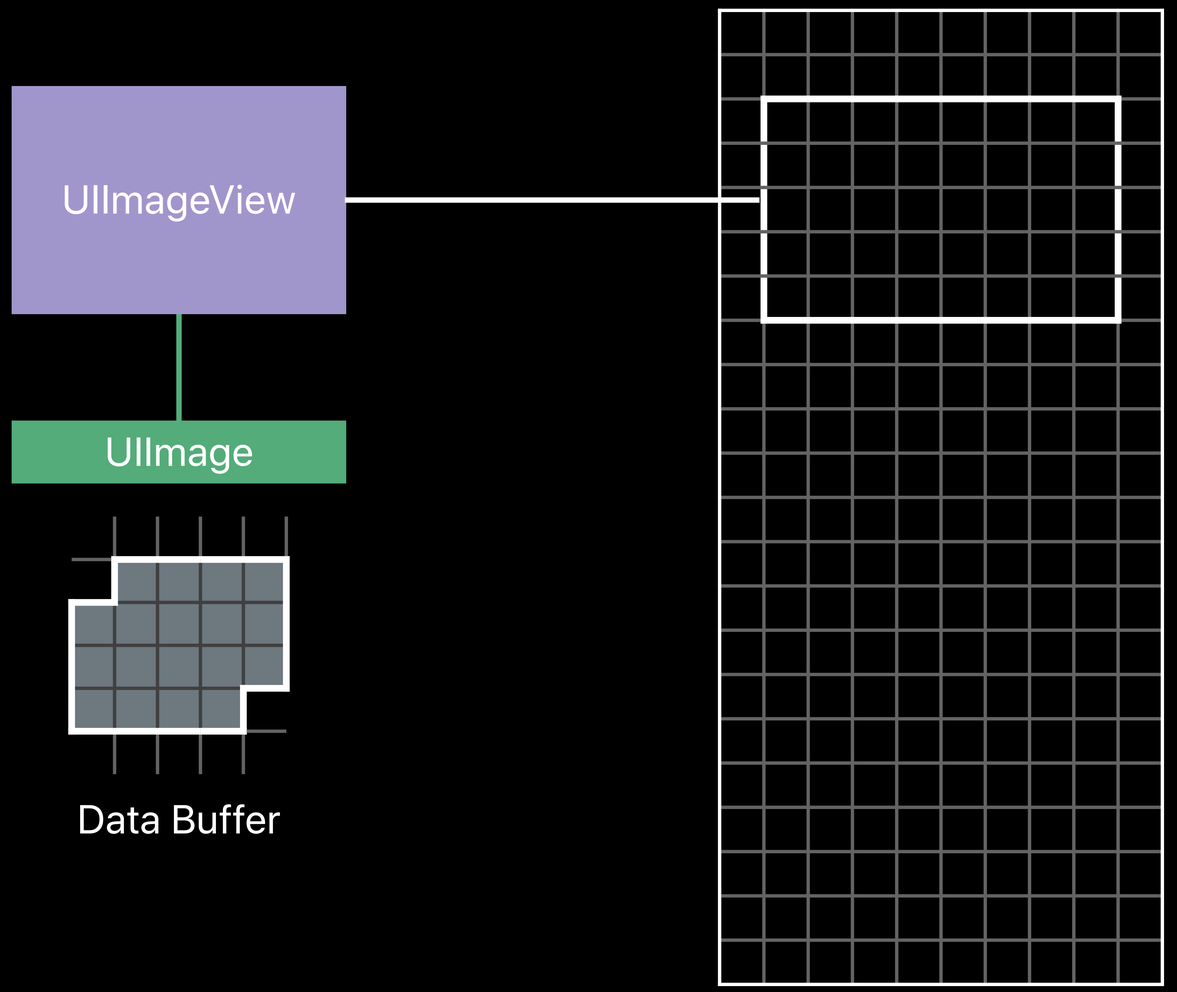
Metadata describing dimensions of image

Image itself encoded as JPEG, PNG, or other (usually compressed) form

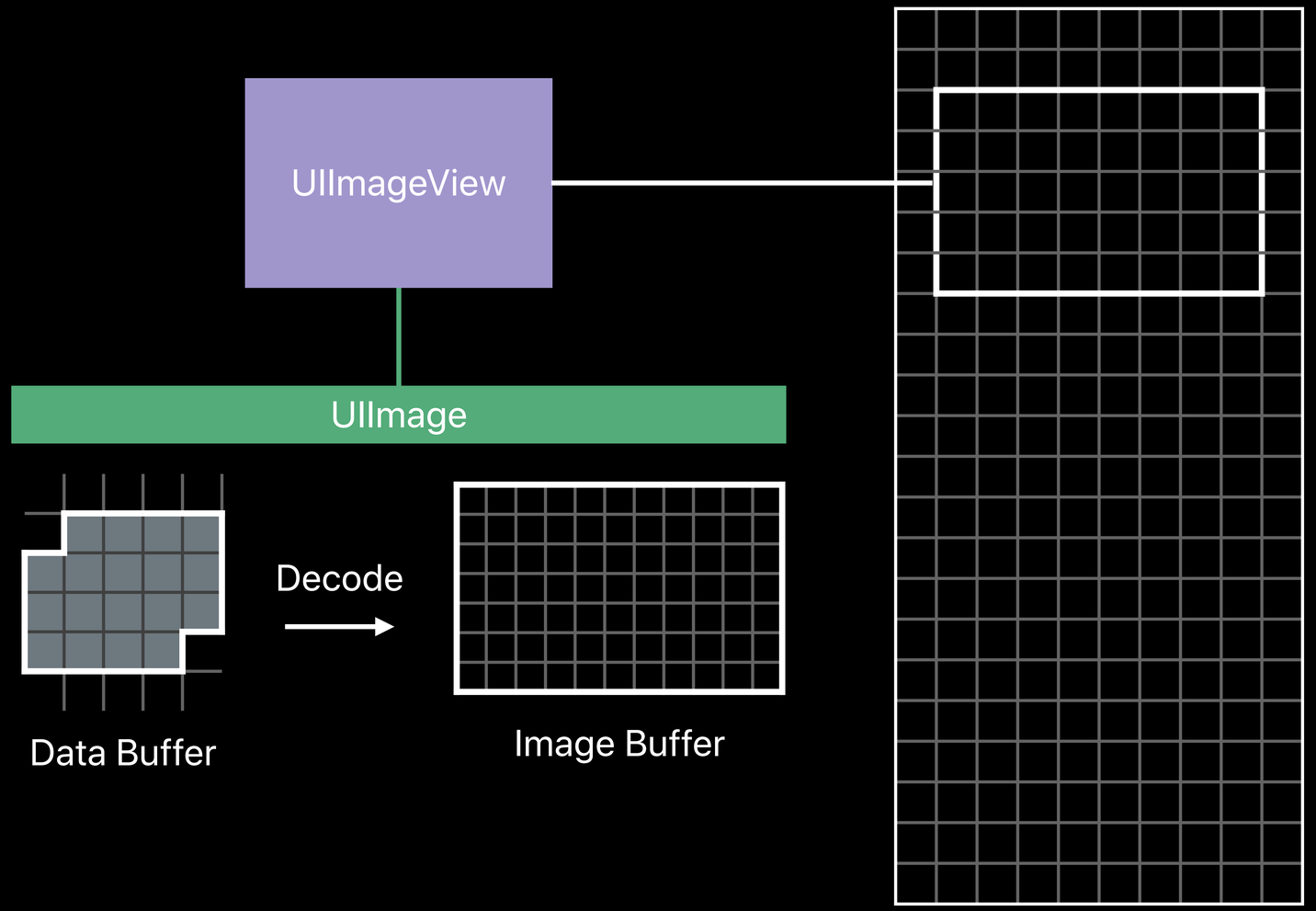
Bytes do not directly describe pixels



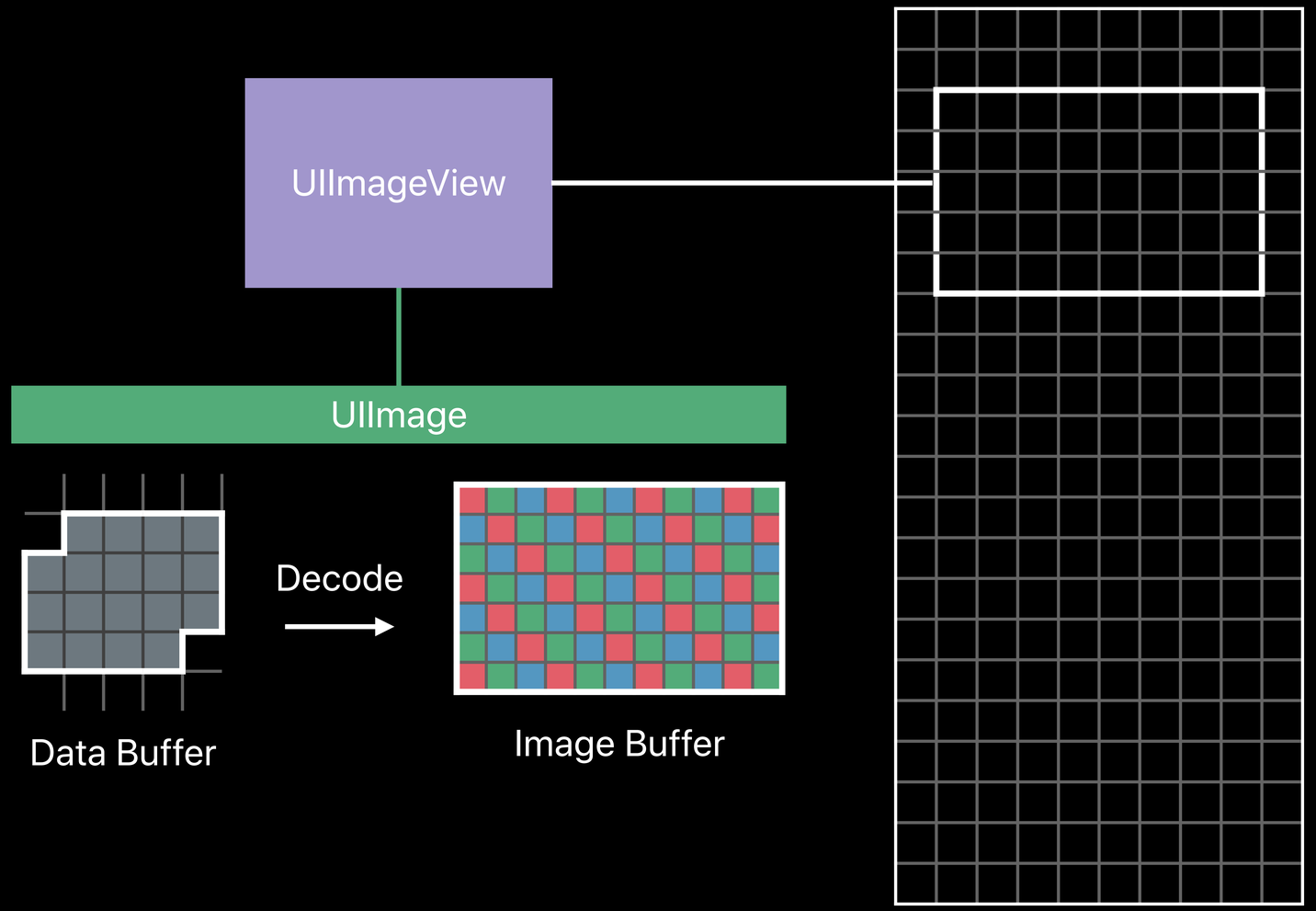
Pipeline in Action



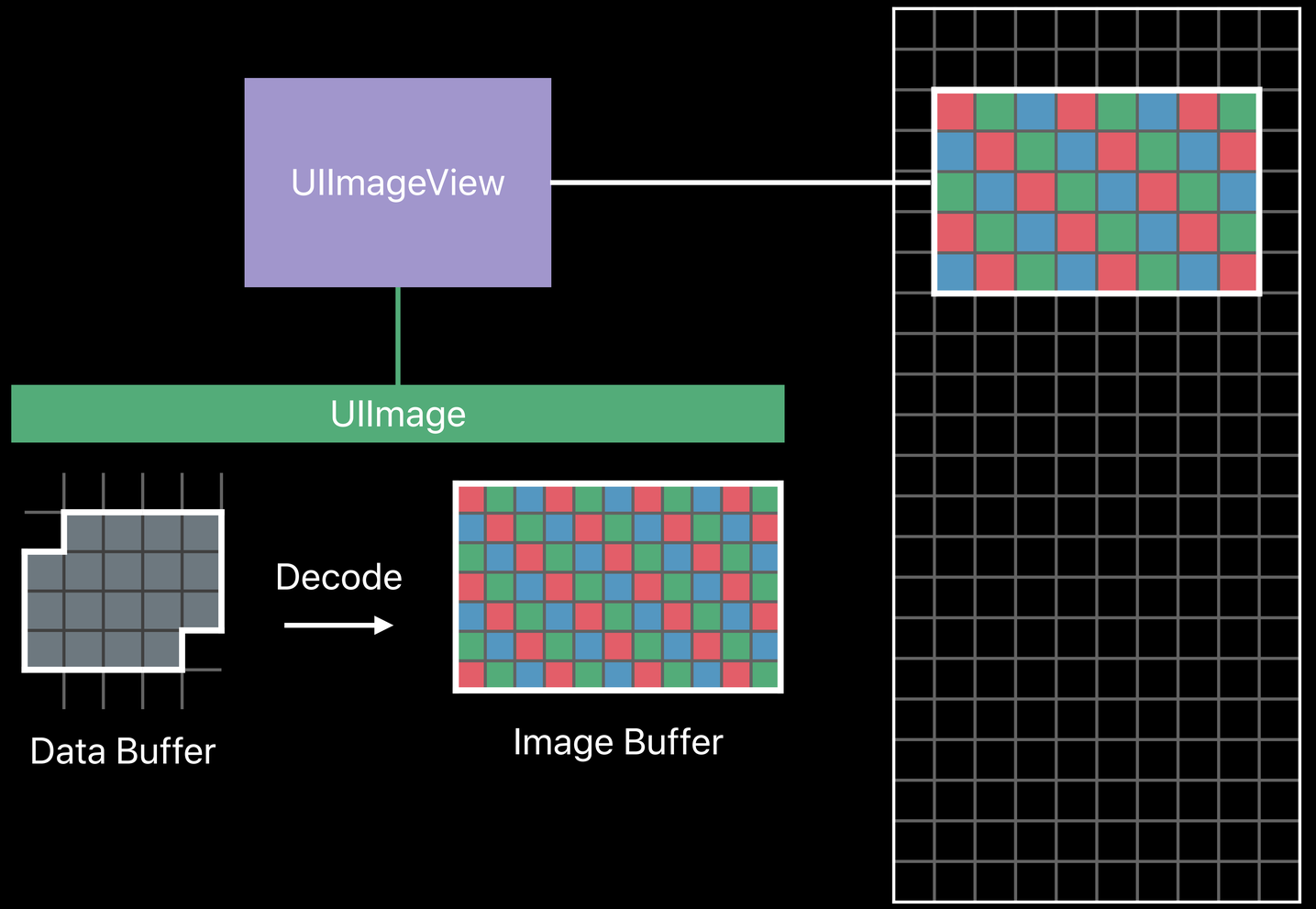
Pipeline in Action



Pipeline in Action



Pipeline in Action



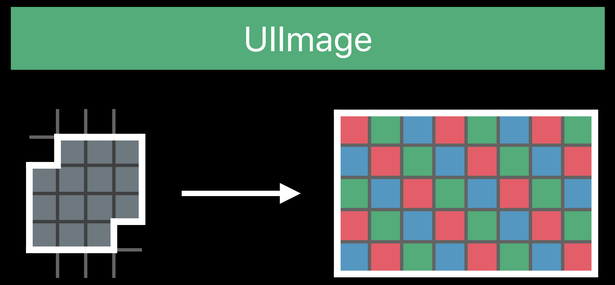
Decoding Concerns

CPU-intensive process

Retained for repeat rendering

Persistent large memory allocation

Proportional to original image size, not view size



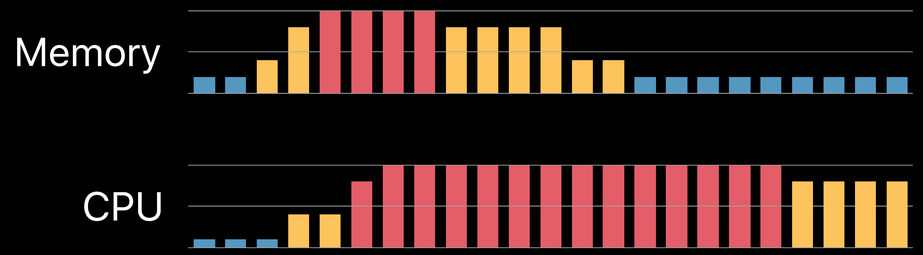
Consequences of Excessive Memory Usage

Increased fragmentation

Poor locality of reference

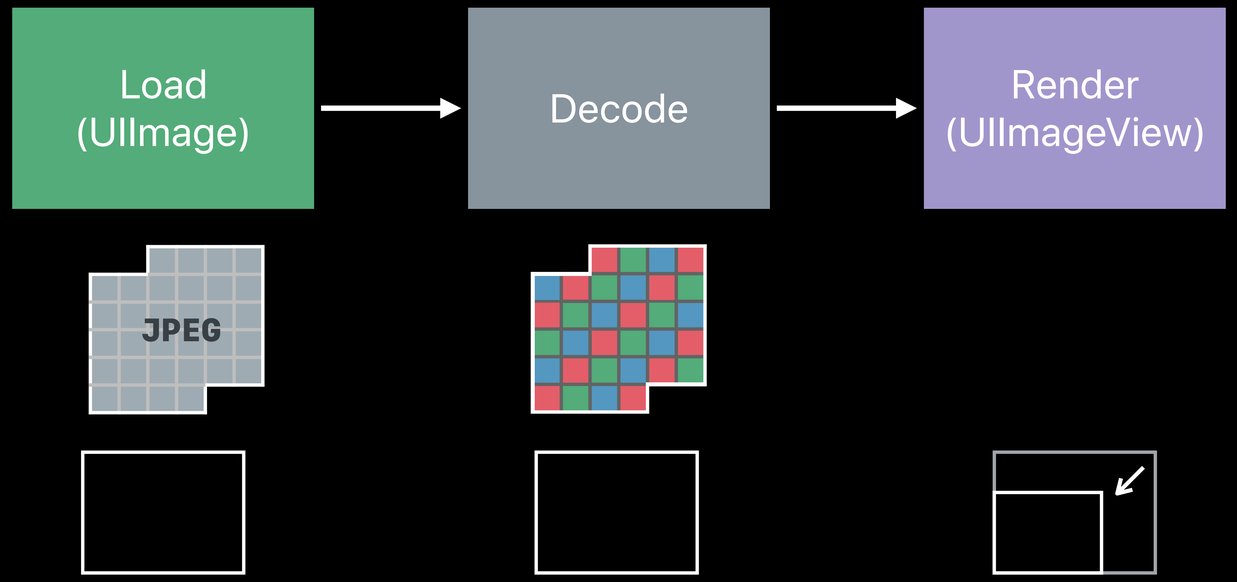
System starts compressing memory

Process termination



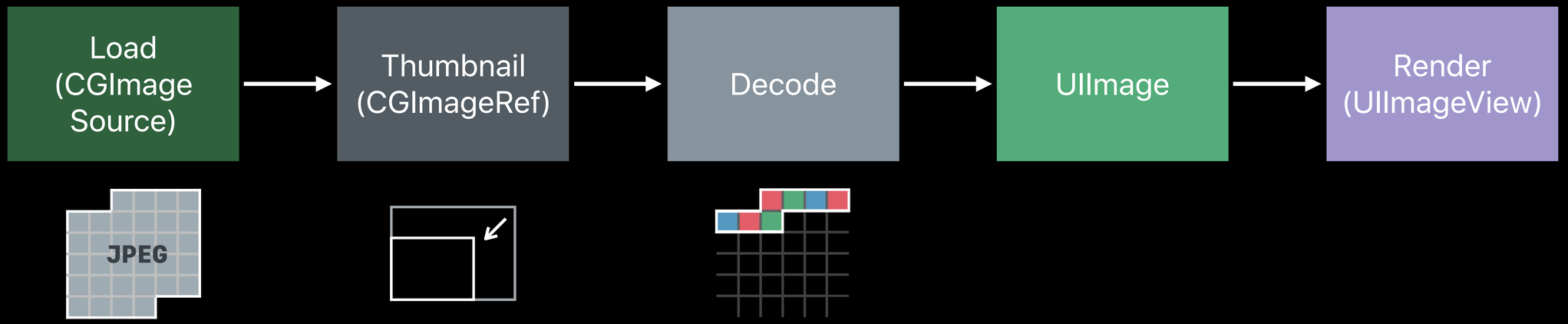
Proactively Saving Memory

Downsampling



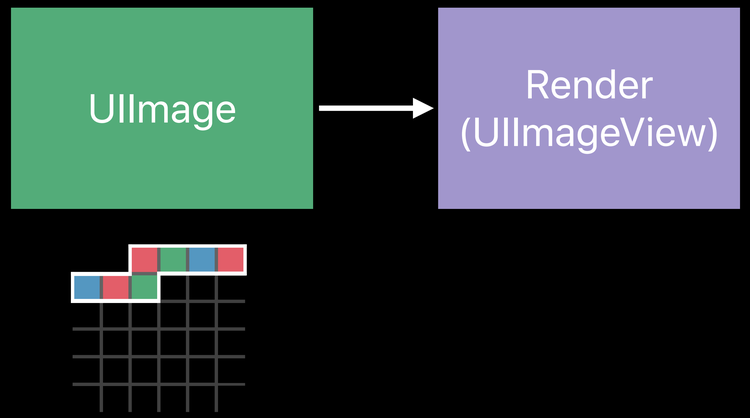
Proactively Saving Memory

Downsampling



Proactively Saving Memory

Downsampling



```swift
// Downsampling large images for display at smaller size

func downsample(imageAt imageURL: URL, to pointSize: CGSize, scale: CGFloat) -> UIImage {
    let imageSourceOptions = [kCGImageSourceShouldCache: false] as CFDictionary
    let imageSource = CGImageSourceCreateWithURL(imageURL as CFURL, imageSourceOptions)!

    let maxDimensionInPixels = max(pointSize.width, pointSize.height) * scale
    let downsampleOptions =
        [kCGImageSourceCreateThumbnailFromImageAlways: true,
         kCGImageSourceShouldCacheImmediately: true,
         kCGImageSourceCreateThumbnailWithTransform: true,
         kCGImageSourceThumbnailMaxPixelSize: maxDimensionInPixels] as CFDictionary

    let downsampledImage =
        CGImageSourceCreateThumbnailAtIndex(imageSource, 0, downsampleOptions)!
    return UIImage(cgImage: downsampledImage)
}
```

```swift
// Downsampling large images for display at smaller size

func downsample(imageAt imageURL: URL, to pointSize: CGSize, scale: CGFloat) -> UIImage {
    let imageSourceOptions = [kCGImageSourceShouldCache: false] as CFDictionary
    let imageSource = CGImageSourceCreateWithURL(imageURL as CFURL, imageSourceOptions)!

    let maxDimensionInPixels = max(pointSize.width, pointSize.height) * scale
    let downsampleOptions =
        [kCGImageSourceCreateThumbnailFromImageAlways: true,
         kCGImageSourceShouldCacheImmediately: true,
         kCGImageSourceCreateThumbnailWithTransform: true,
         kCGImageSourceThumbnailMaxPixelSize: maxDimensionInPixels] as CFDictionary

    let downsampledImage =
        CGImageSourceCreateThumbnailAtIndex(imageSource, 0, downsampleOptions)!
    return UIImage(cgImage: downsampledImage)
}
```

```swift
// Downsampling large images for display at smaller size

func downsample(imageAt imageURL: URL, to pointSize: CGSize, scale: CGFloat) -> UIImage {
    let imageSourceOptions = [kCGImageSourceShouldCache: false] as CFDictionary
    let imageSource = CGImageSourceCreateWithURL(imageURL as CFURL, imageSourceOptions)!

    let maxDimensionInPixels = max(pointSize.width, pointSize.height) * scale
    let downsampleOptions =
        [kCGImageSourceCreateThumbnailFromImageAlways: true,
         kCGImageSourceShouldCacheImmediately: true,
         kCGImageSourceCreateThumbnailWithTransform: true,
         kCGImageSourceThumbnailMaxPixelSize: maxDimensionInPixels] as CFDictionary

    let downsampledImage =
        CGImageSourceCreateThumbnailAtIndex(imageSource, 0, downsampleOptions)!
    return UIImage(cgImage: downsampledImage)
}
```

```swift
// Downsampling large images for display at smaller size

func downsample(imageAt imageURL: URL, to pointSize: CGSize, scale: CGFloat) -> UIImage {
    let imageSourceOptions = [kCGImageSourceShouldCache: false] as CFDictionary
    let imageSource = CGImageSourceCreateWithURL(imageURL as CFURL, imageSourceOptions)!

    let maxDimensionInPixels = max(pointSize.width, pointSize.height) * scale
    let downsampleOptions =
        [kCGImageSourceCreateThumbnailFromImageAlways: true,
         kCGImageSourceShouldCacheImmediately: true,
         kCGImageSourceCreateThumbnailWithTransform: true,
         kCGImageSourceThumbnailMaxPixelSize: maxDimensionInPixels] as CFDictionary

    let downsampledImage =
        CGImageSourceCreateThumbnailAtIndex(imageSource, 0, downsampleOptions)!
    return UIImage(cgImage: downsampledImage)
}
```

```swift
// Downsampling large images for display at smaller size

func downsample(imageAt imageURL: URL, to pointSize: CGSize, scale: CGFloat) -> UIImage {
    let imageSourceOptions = [kCGImageSourceShouldCache: false] as CFDictionary
    let imageSource = CGImageSourceCreateWithURL(imageURL as CFURL, imageSourceOptions)!

    let maxDimensionInPixels = max(pointSize.width, pointSize.height) * scale
    let downsampleOptions =
        [kCGImageSourceCreateThumbnailFromImageAlways: true,
         kCGImageSourceShouldCacheImmediately: true,
         kCGImageSourceCreateThumbnailWithTransform: true,
         kCGImageSourceThumbnailMaxPixelSize: maxDimensionInPixels] as CFDictionary

    let downsampledImage =
        CGImageSourceCreateThumbnailAtIndex(imageSource, 0, downsampleOptions)!
    return UIImage(cgImage: downsampledImage)
}
```

```swift
// Downsampling large images for display at smaller size

func downsample(imageAt imageURL: URL, to pointSize: CGSize, scale: CGFloat) -> UIImage {
    let imageSourceOptions = [kCGImageSourceShouldCache: false] as CFDictionary
    let imageSource = CGImageSourceCreateWithURL(imageURL as CFURL, imageSourceOptions)!

    let maxDimensionInPixels = max(pointSize.width, pointSize.height) * scale
    let downsampleOptions =
        [kCGImageSourceCreateThumbnailFromImageAlways: true,
         kCGImageSourceShouldCacheImmediately: true,
         kCGImageSourceCreateThumbnailWithTransform: true,
         kCGImageSourceThumbnailMaxPixelSize: maxDimensionInPixels] as CFDictionary

    let downsampledImage =
        CGImageSourceCreateThumbnailAtIndex(imageSource, 0, downsampleOptions)!
    return UIImage(cgImage: downsampledImage)
}
```

Without downsampling

31.5MiB

Persistent memory use

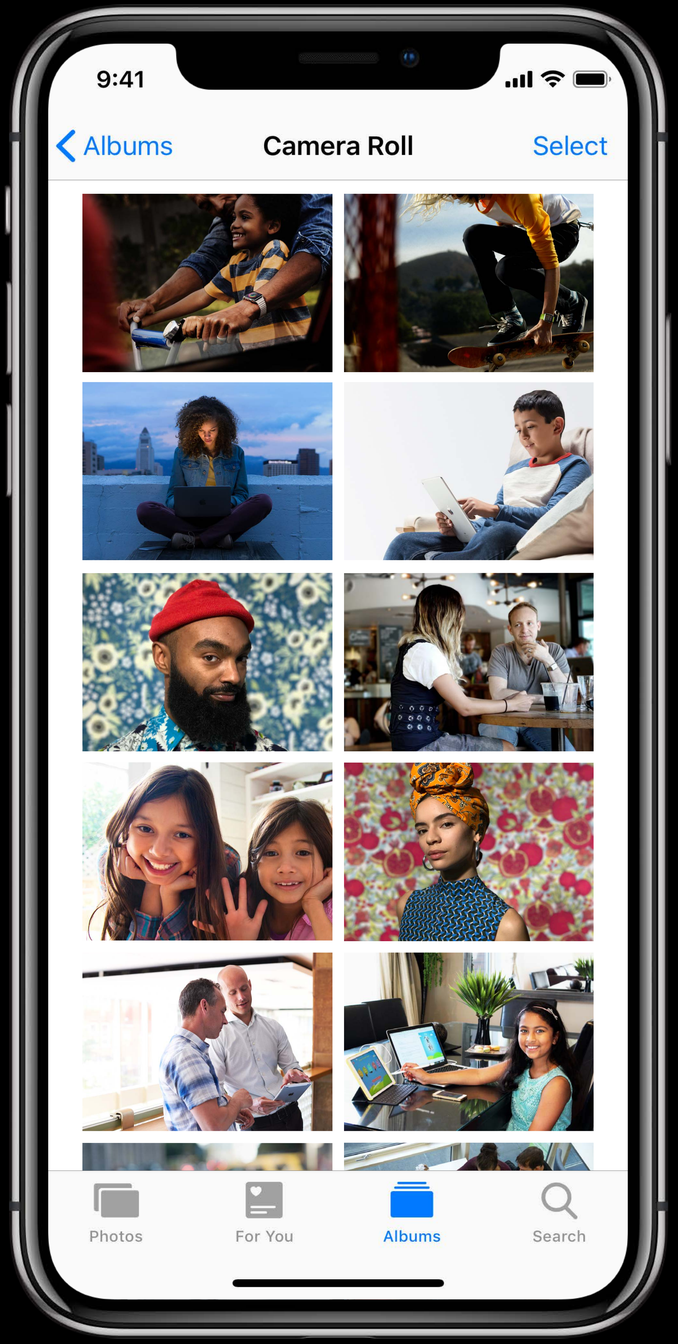
Without downsampling

31.5MiB

Persistent memory use

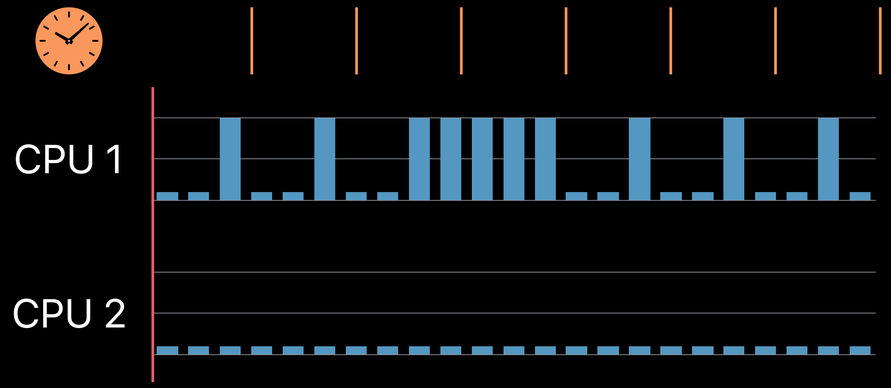
With downsampling

18.4MiB

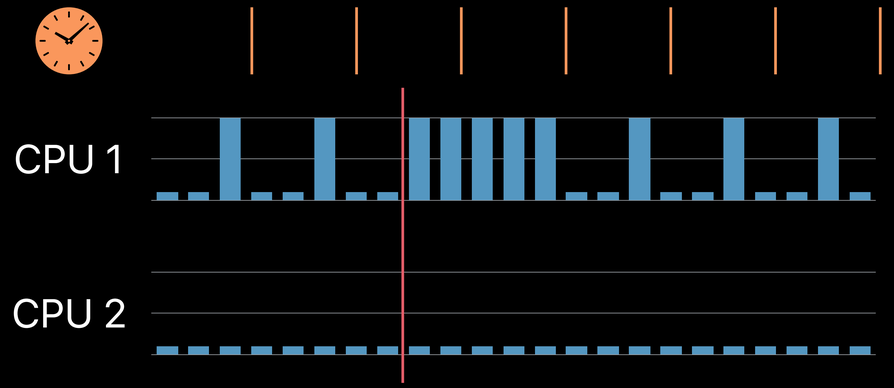

9:41
Albums
Camera Roll
Select
Photos
For You
Albums
Search

```
// Downsampling large images for display at smaller size

func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath)
  -> UICollectionViewCell {
    let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "Cell",
      for: indexPath) as! MyCollectionViewCell
    cell.layoutIfNeeded() // Ensure imageView is its final size.

    let imageViewSize = cell.imageView.bounds.size
    let scale = collectionView.traitCollection.displayScale
    cell.imageView.image = downsample(imageAt: imageURLs[indexPath.item],
      to: imageViewSize, scale: scale)

    return cell
}
```

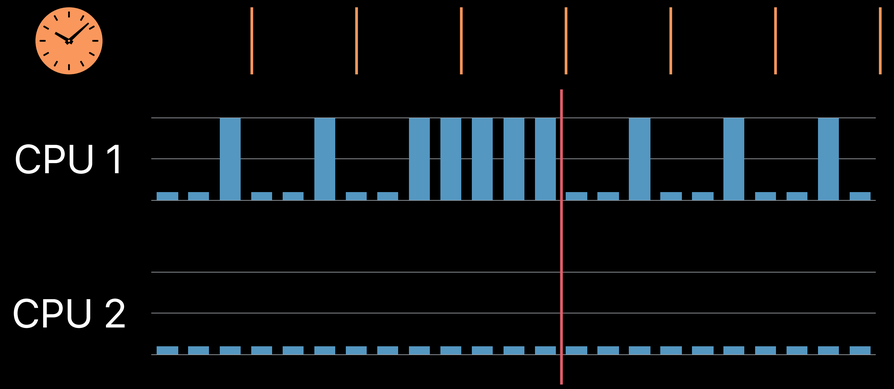
Decoding in Scrollable Views



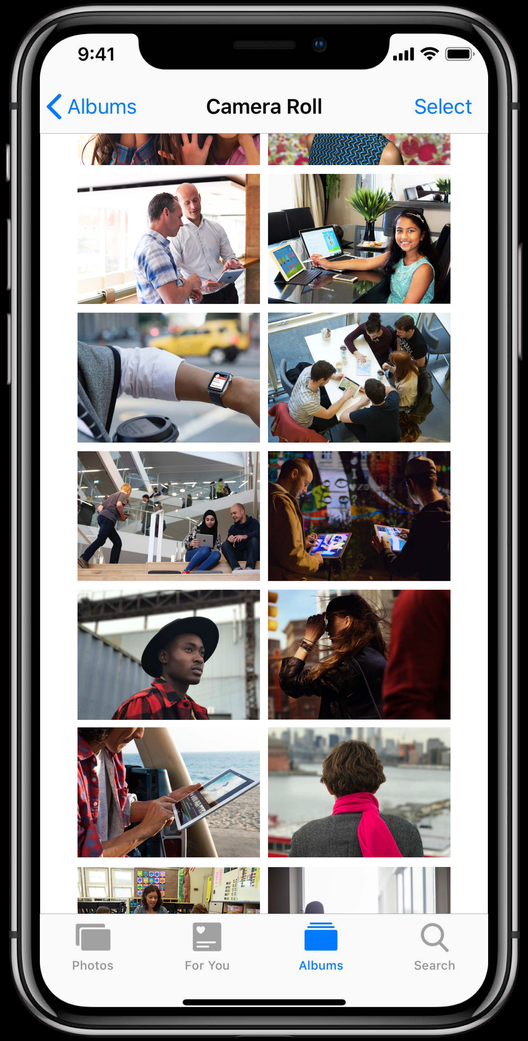
Decoding in Scrollable Views

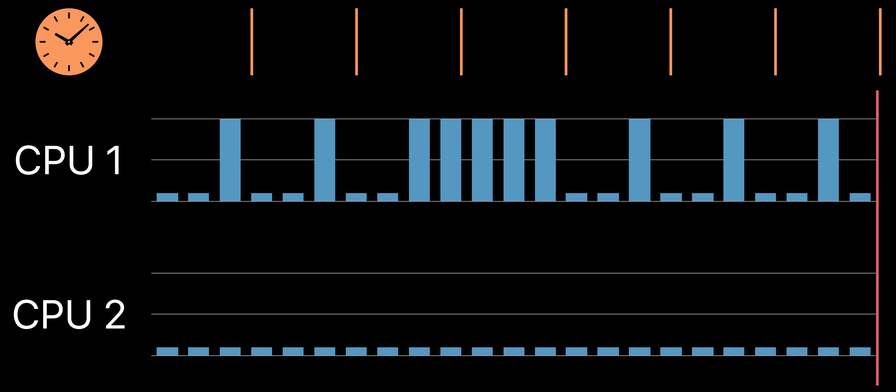


Decoding in Scrollable Views

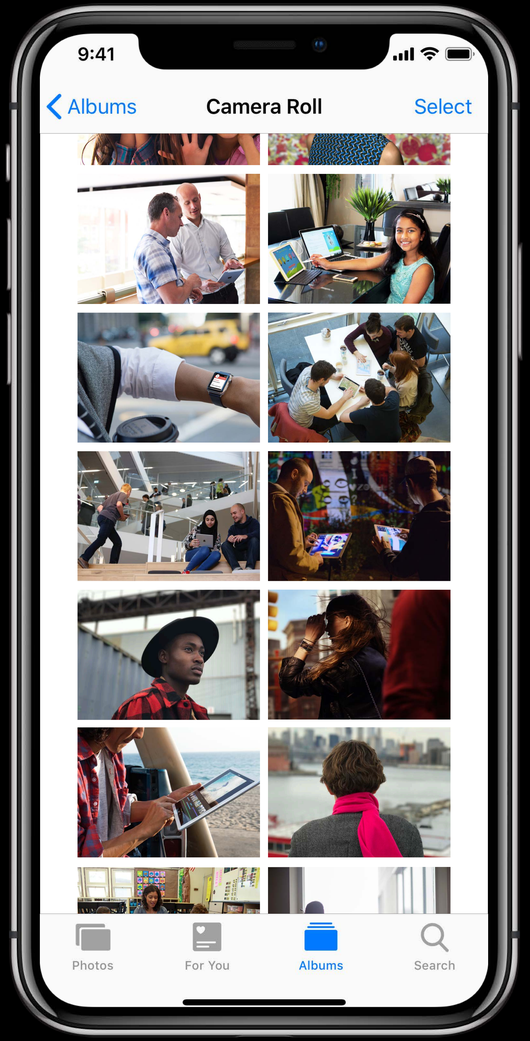


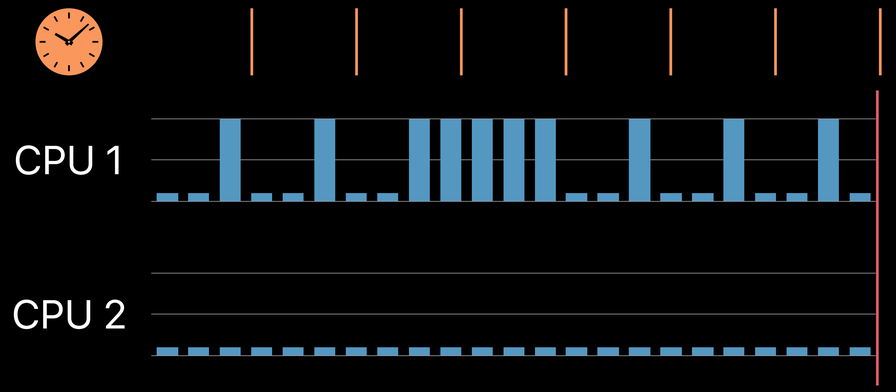
Decoding in Scrollable Views



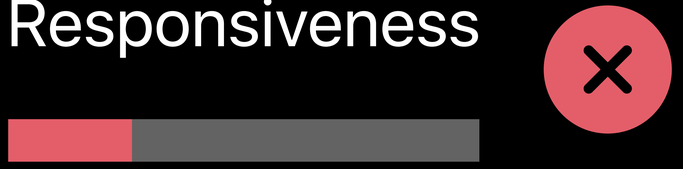


Decoding in Scrollable Views





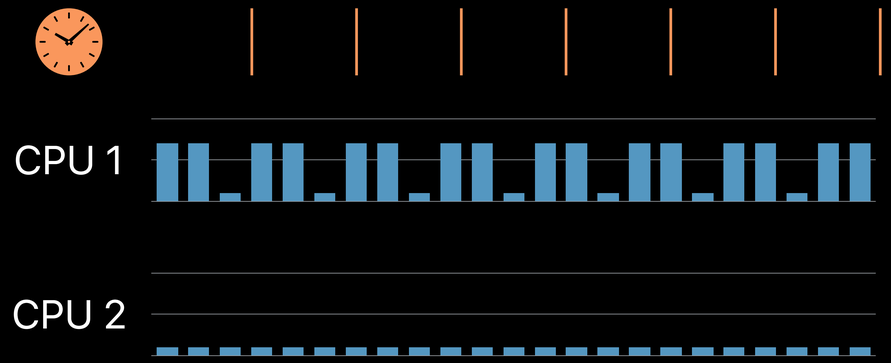
Responsiveness



Battery Life

Decoding in Scrollable Views

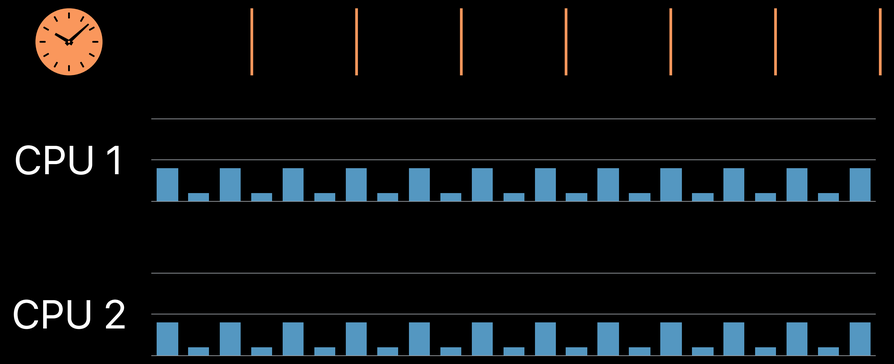
Prefetching



Decoding in Scrollable Views

Prefetching

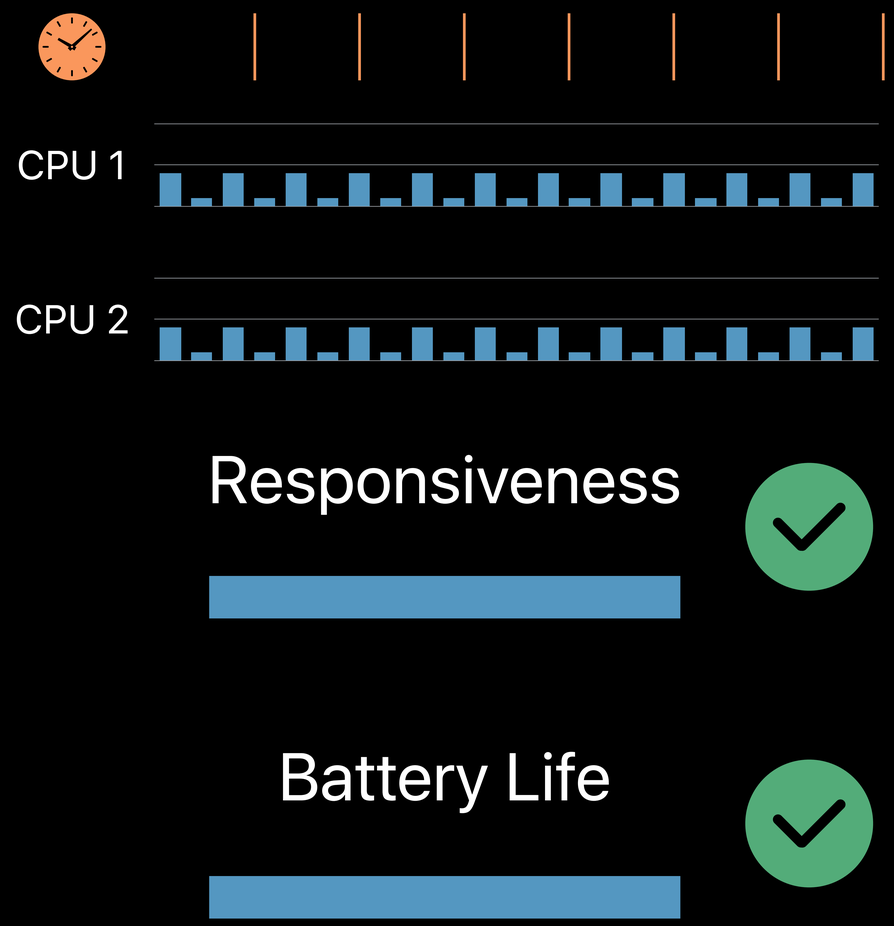
Background decoding/downsampling



Decoding in Scrollable Views

Prefetching

Background decoding/downsampling



```swift
// Asynchronously downsampling on a global queue

func collectionView(_ collectionView: UICollectionView,
                    prefetchItemsAt indexPaths: [IndexPath]) {
    // Asynchronously decode and downsample every image we are about to show
    for indexPath in indexPaths {
        DispatchQueue.global(qos: .userInitiated).async {
            let downsampledImage = downsample(images[indexPath.row])
            DispatchQueue.main.async { self.update(at: indexPath, with: downsampledImage) }
        }
    }
}
```

```
// Asynchronously downsampling on a global queue

func collectionView(_ collectionView: UICollectionView,
                    prefetchItemsAt indexPaths: [IndexPath]) {
    // Asynchronously decode and downsample every image we are about to show
    for indexPath in indexPaths {
        DispatchQueue.global(qos: .userInitiated).async {
            let downsampledImage = downsample(images[indexPath.row])
            DispatchQueue.main.async { self.update(at: indexPath, with: downsampledImage) }
        }
    }
}
```

```swift
// Asynchronously downsampling on a global queue

func collectionView(_ collectionView: UICollectionView,
                    prefetchItemsAt indexPaths: [IndexPath]) {
    // Asynchronously decode and downsample every image we are about to show
    for indexPath in indexPaths {
        DispatchQueue.global(qos: .userInitiated).async {
            let downsampledImage = downsample(images[indexPath.row])
            DispatchQueue.main.async { self.update(at: indexPath, with: downsampledImage) }
        }
    }
}
```

```
// Asynchronously downsampling on a global queue

func collectionView(_ collectionView: UICollectionView,
                    prefetchItemsAt indexPaths: [IndexPath]) {
    // Asynchronously decode and downsample every image we are about to show
    for indexPath in indexPaths {
        DispatchQueue.global(qos: .userInitiated).async {
            let downsampledImage = downsample(images[indexPath.row])
            DispatchQueue.main.async { self.update(at: indexPath, with: downsampledImage) }
        }
    }
}
```

Thread Explosion

More images to decode than available CPUs

GCD continues creating threads as new work is enqueued

Each thread gets less time to actually decode images

```
// Avoiding thread explosion when doing asynchronous work

let serialQueue = DispatchQueue(label: "Decode queue")
func collectionView(_ collectionView: UICollectionView,
                    prefetchItemsAt indexPaths: [IndexPath]) {
    // Asynchronously decode and downsample every image we are about to show
    for indexPath in indexPaths {
        serialQueue.async {
            let downsampledImage = downsample(images[indexPath.row])
            DispatchQueue.main.async { self.update(at: indexPath, with: downsampledImage) }
        }
    }
}
```

```
// Avoiding thread explosion when doing asynchronous work

let serialQueue = DispatchQueue(label: "Decode queue")
func collectionView(_ collectionView: UICollectionView,
                    prefetchItemsAt indexPaths: [IndexPath]) {
    // Asynchronously decode and downsample every image we are about to show
    for indexPath in indexPaths {
        serialQueue.async {
            let downsampledImage = downsample(images[indexPath.row])
            DispatchQueue.main.async { self.update(at: indexPath, with: downsampledImage) }
        }
    }
}
```

```
// Avoiding thread explosion when doing asynchronous work

let serialQueue = DispatchQueue(label: "Decode queue")
func collectionView(_ collectionView: UICollectionView,
                    prefetchItemsAt indexPaths: [IndexPath]) {
    // Asynchronously decode and downsample every image we are about to show
    for indexPath in indexPaths {
        serialQueue.async {
            let downsampledImage = downsample(images[indexPath.row])
            DispatchQueue.main.async { self.update(at: indexPath, with: downsampledImage) }
        }
    }
}
```

Image Sources

Image assets in asset catalog

Files in application/framework bundle

Files in Documents and Caches directories

Data downloaded from network

Image Sources

Image assets in asset catalog

Files in application/framework bundle

Files in Documents and Caches directories

Data downloaded from network

Prefer Image Assets

For artwork bundled with your app

Optimized name- and trait-based lookup

Smarter buffer caching

Per-device thinning

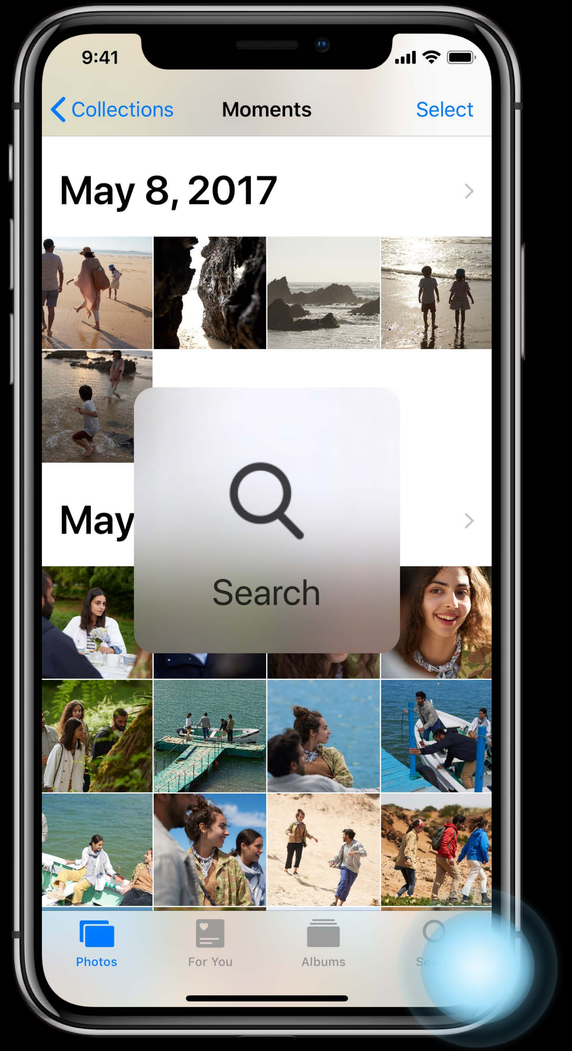
Vector artwork

Vector Artwork

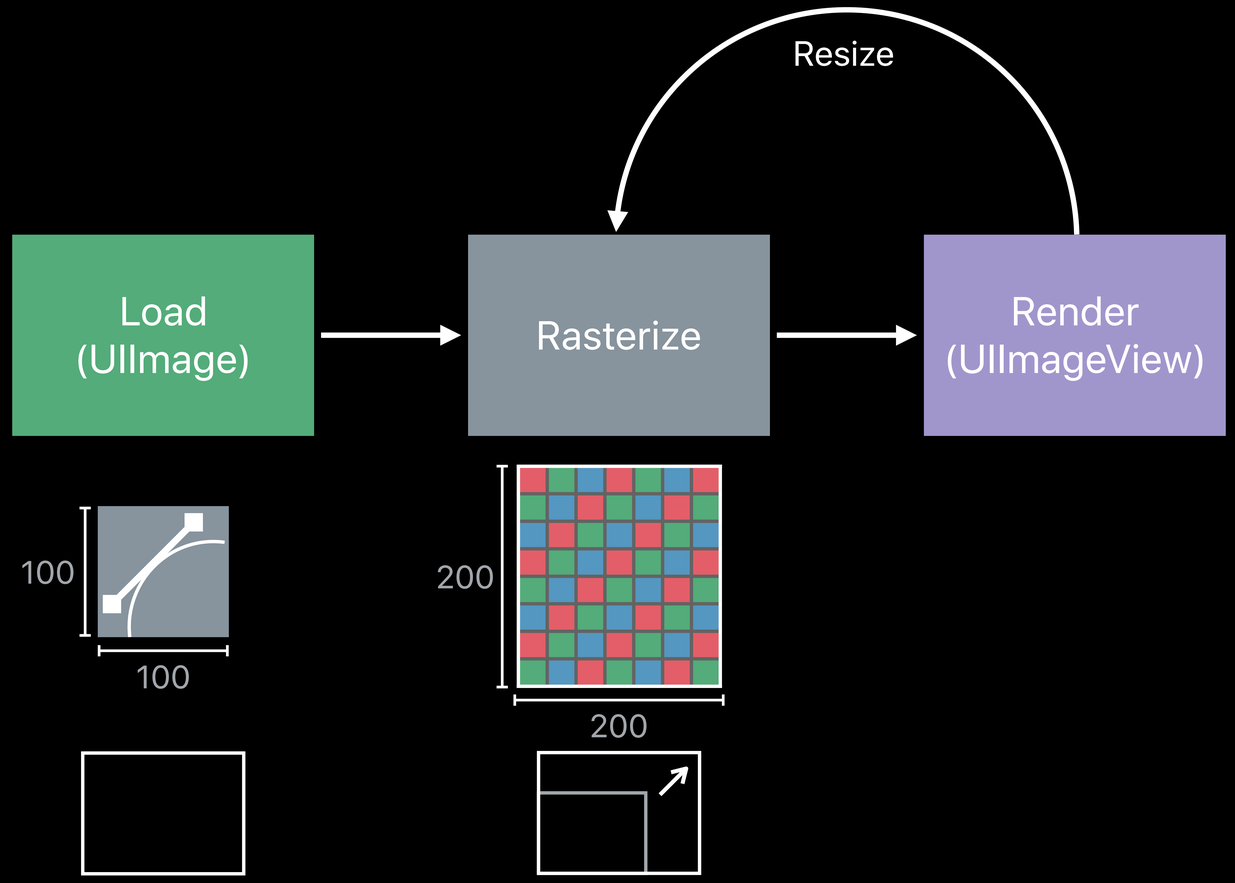
Since iOS 11, image assets support
“Preserve Vector Data”

Avoids blurriness and aliasing when drawn
larger or smaller than natural size

Preserves legibility of icons in accessibility HUD



Vector Artwork Pipeline

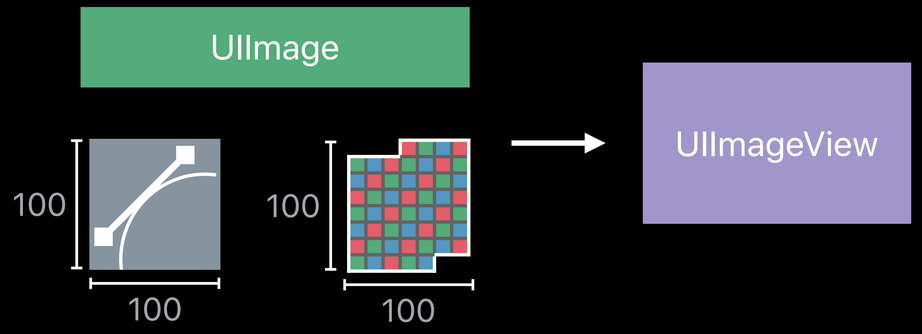


Vector Artwork Optimizations

Xcode rasterizes artwork for relevant scale factors while compiling

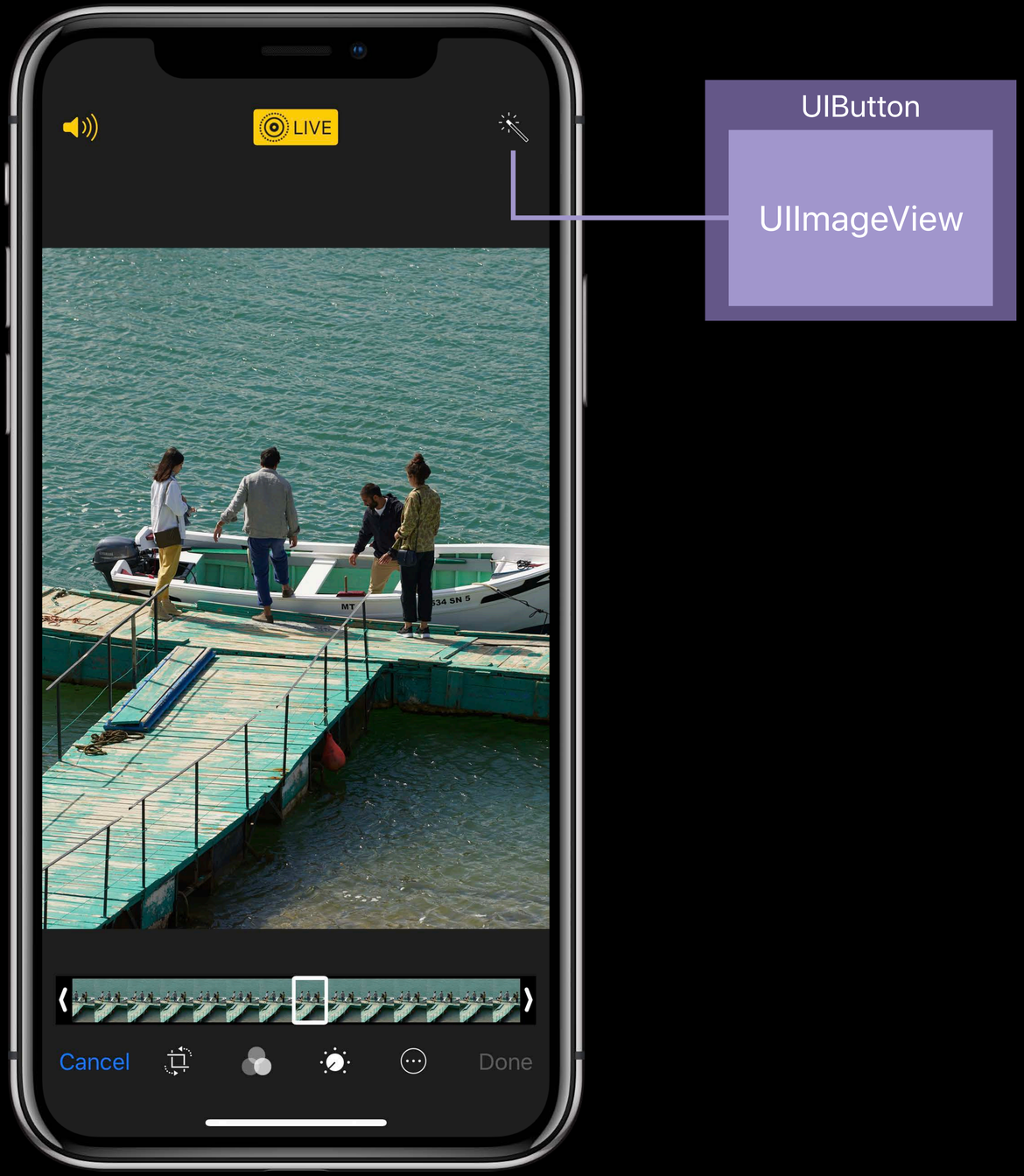
Prerasterized artwork used when image is drawn at natural size

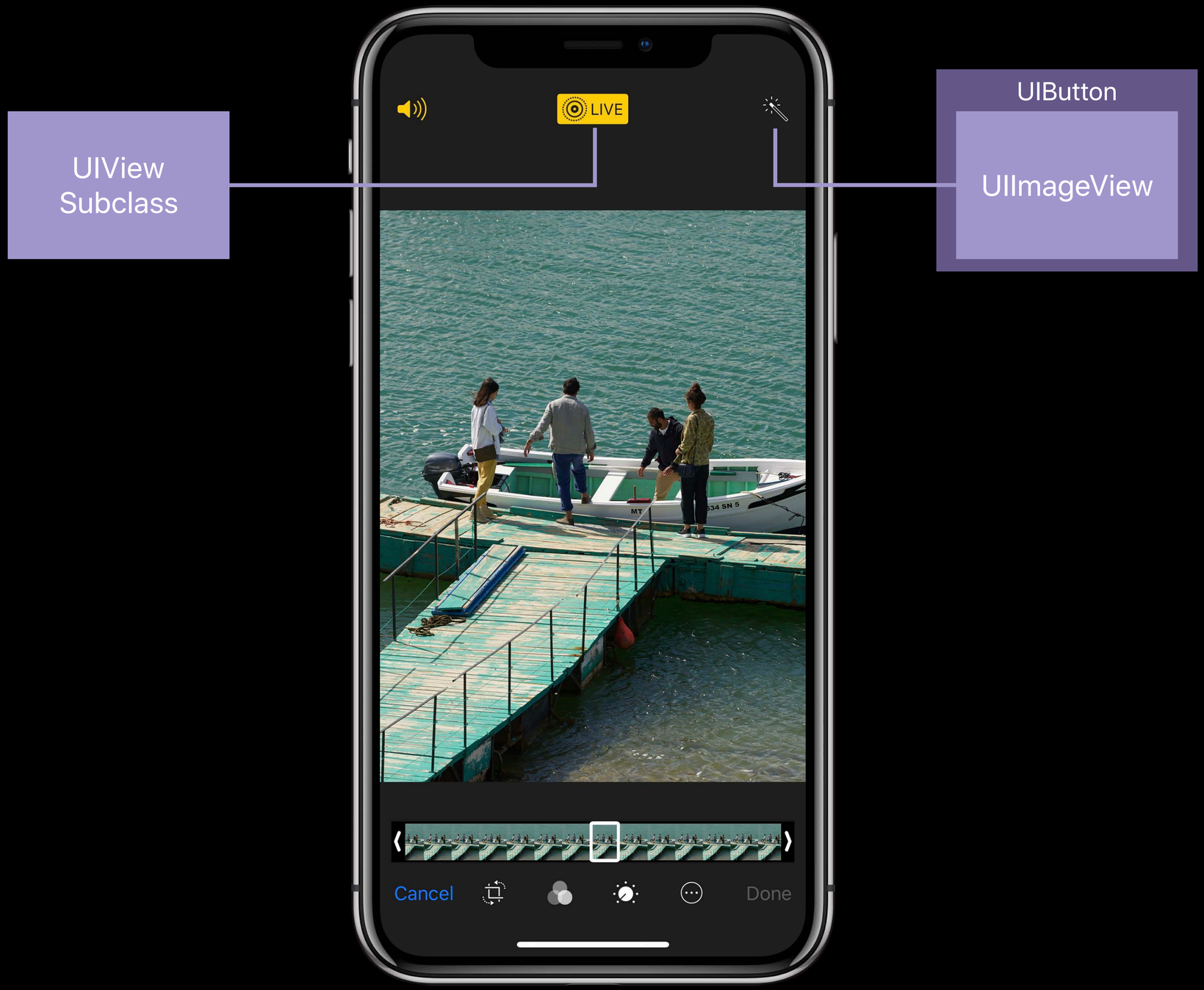
If artwork has fixed sizes, use multiple image assets instead of relying on vector rasterization



Custom Drawing with UIKit


LIVE
Cancel
Done

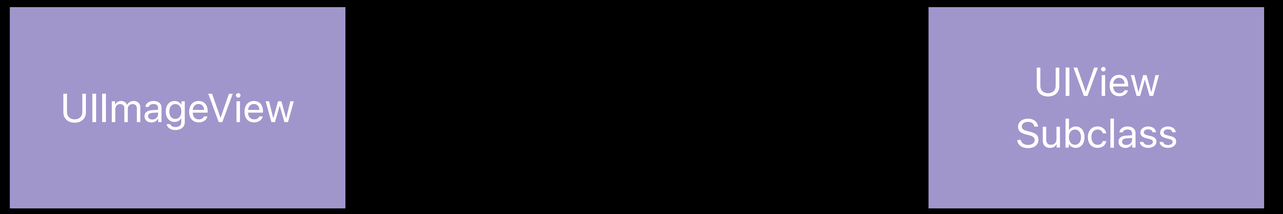

LIVE
Cancel
Done
UIButton
UIImageView


UIView
Subclass
LIVE
UIButton
UIImageView
Cancel
Done

```swift
// Drawing a custom view by overriding UIView.draw(_:)
override func draw(_ rect: CGRect) {
    // Draw rounded rectangle background.
    let roundRectPath = UIBezierPath(roundedRect: self.bounds, cornerRadius: 4.0)
    UIColor.yellow.set()
    roundRectPath.fill()

    // Draw Live Photo icon.
    let image = UIImage(named: "LivePhotosIcon")
    image.draw(at: CGPoint(x: 2.0, y: 2.0))

    // Draw label.
    let text: NSAttributedString(string: "LIVE", attributes: ...)
    text.draw(at: CGPoint(x: 20.0, y: 2.0))
}
```

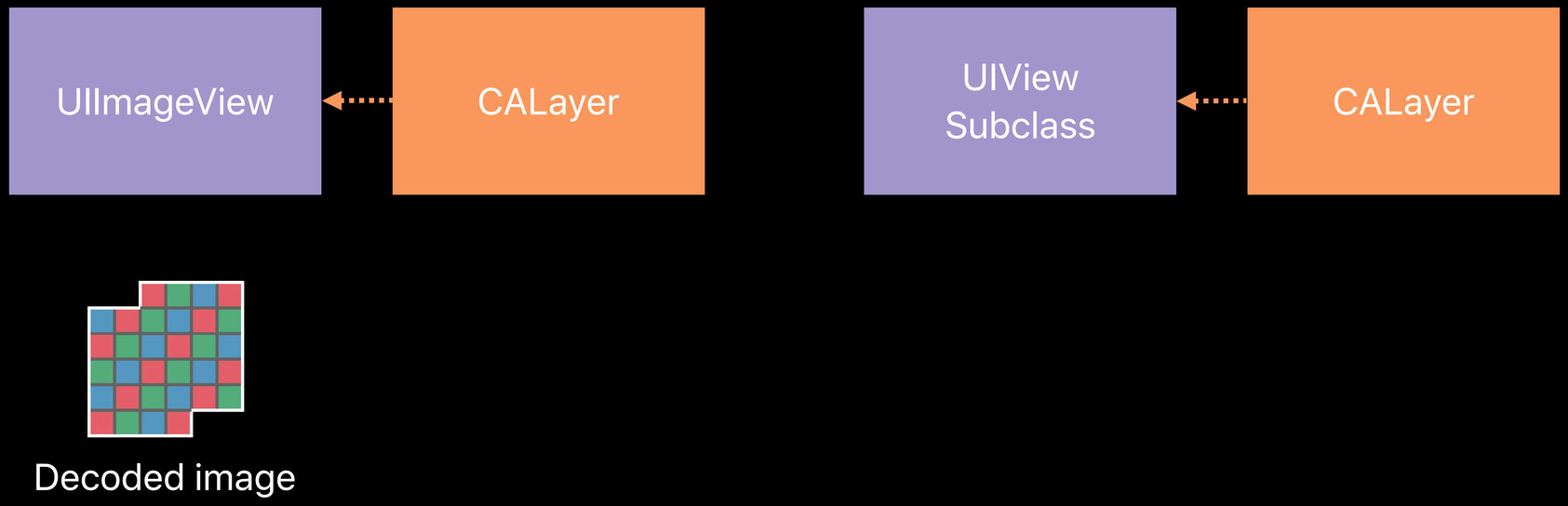
```
// Drawing a custom view by overriding UIView.draw(_:)
override func draw(_ rect: CGRect) {
    // Draw rounded rectangle background.
    let roundRectPath = UIBezierPath(roundedRect: self.bounds, cornerRadius: 4.0)
    UIColor.yellow.set()
    roundRectPath.fill()

    // Draw Live Photo icon.
    let image = UIImage(named: "LivePhotosIcon")
    image.draw(at: CGPoint(x: 2.0, y: 2.0))

    // Draw label.
    let text: NSAttributedString(string: "LIVE", attributes: ...)
    text.draw(at: CGPoint(x: 20.0, y: 2.0))
}
```

Custom Drawing Versus UIImageView

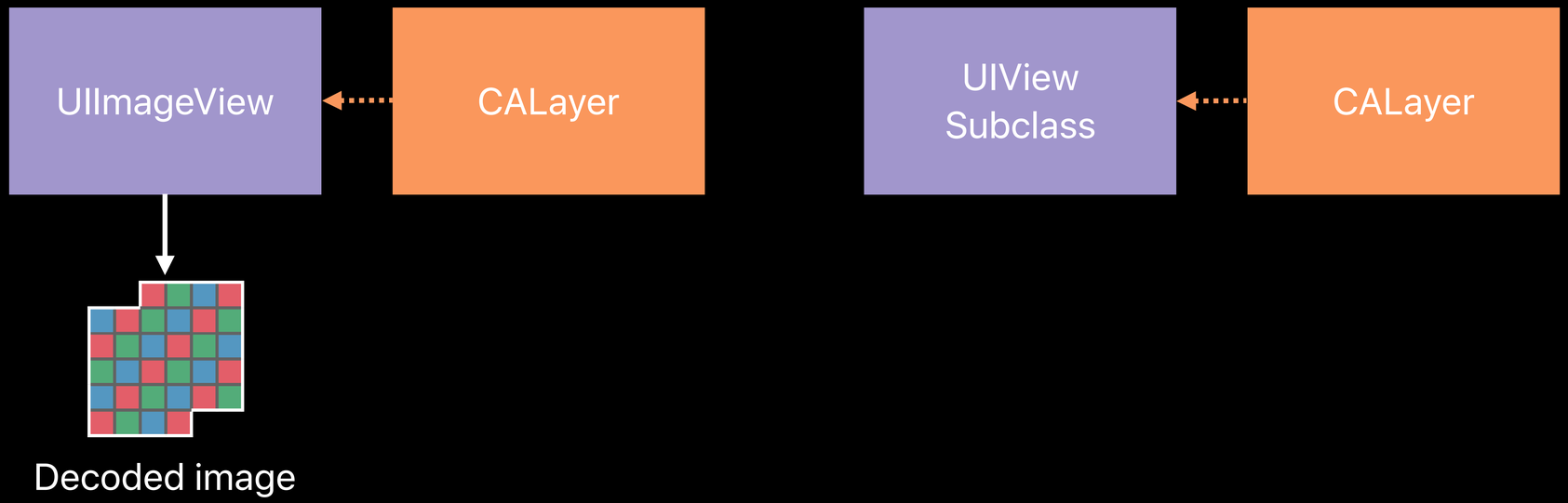


Custom Drawing Versus UIImageView

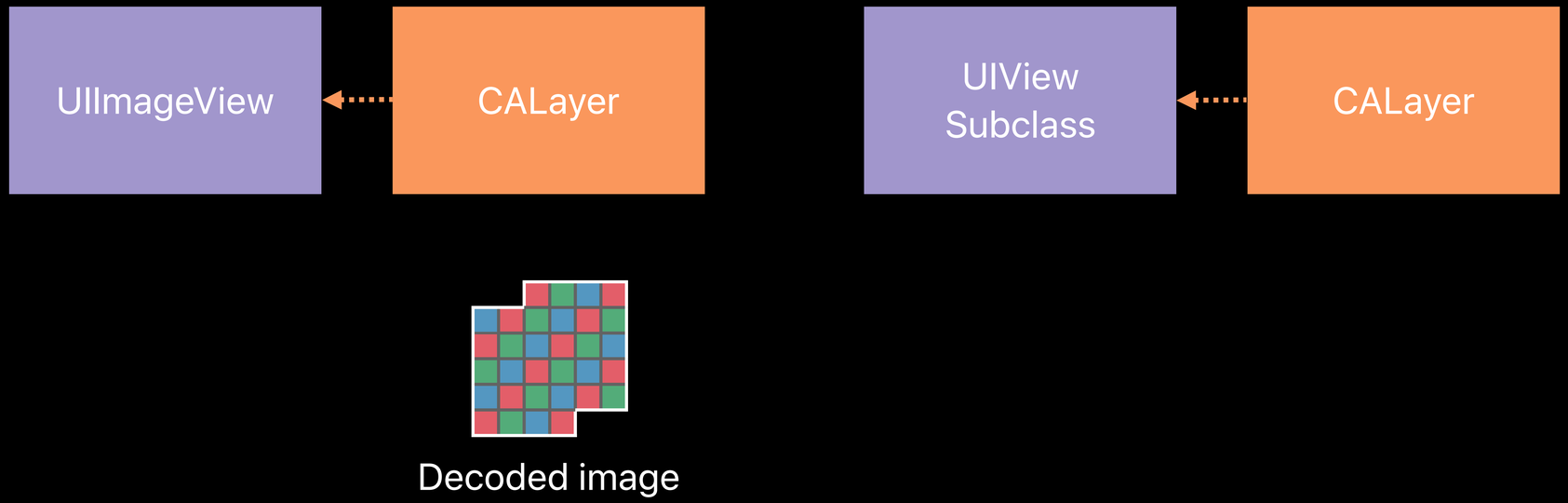
Custom Drawing Versus UIImageView



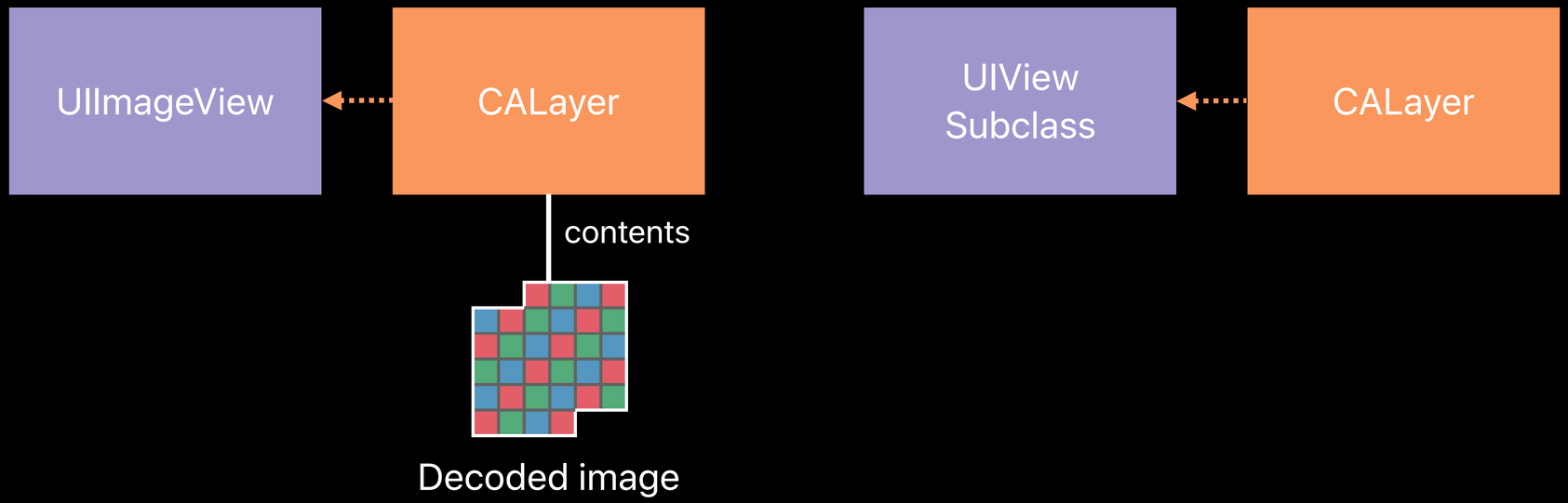
Custom Drawing Versus UIImageView



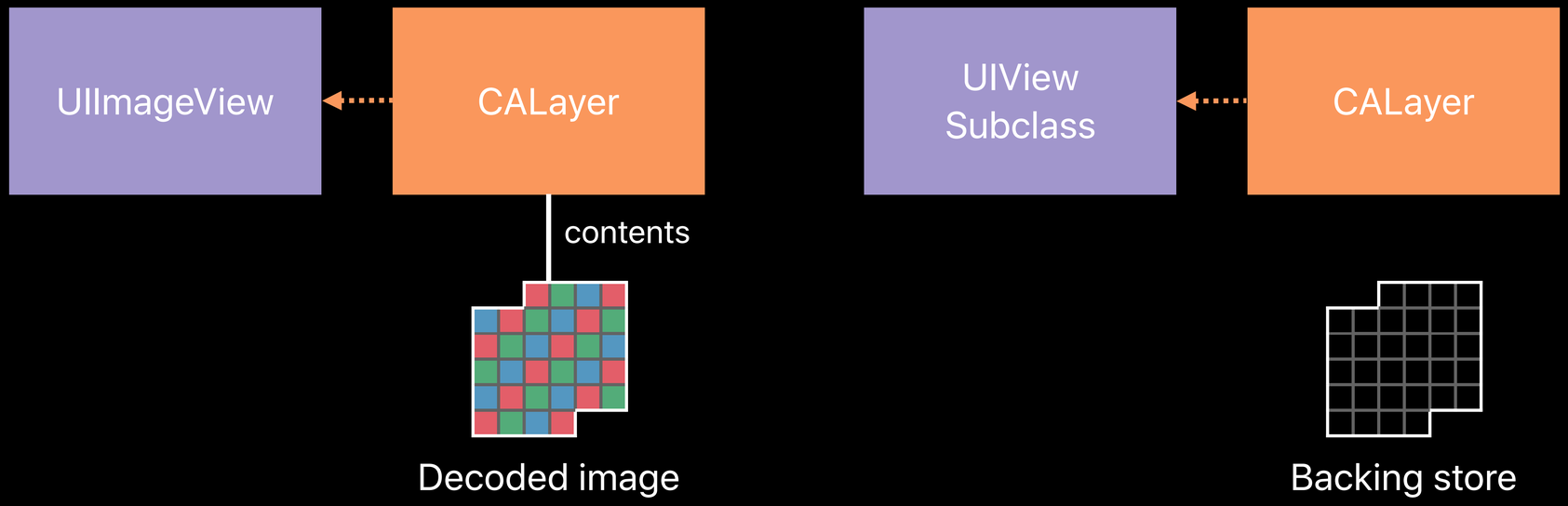
Custom Drawing Versus UIImageView



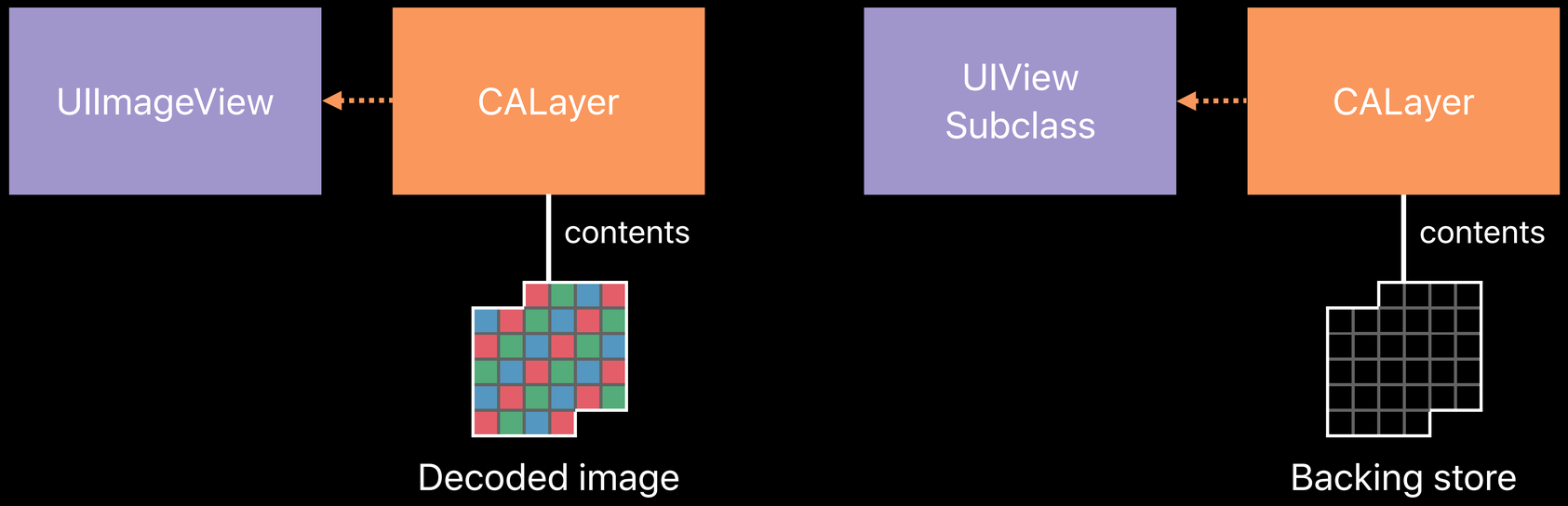
Custom Drawing Versus UIImageView



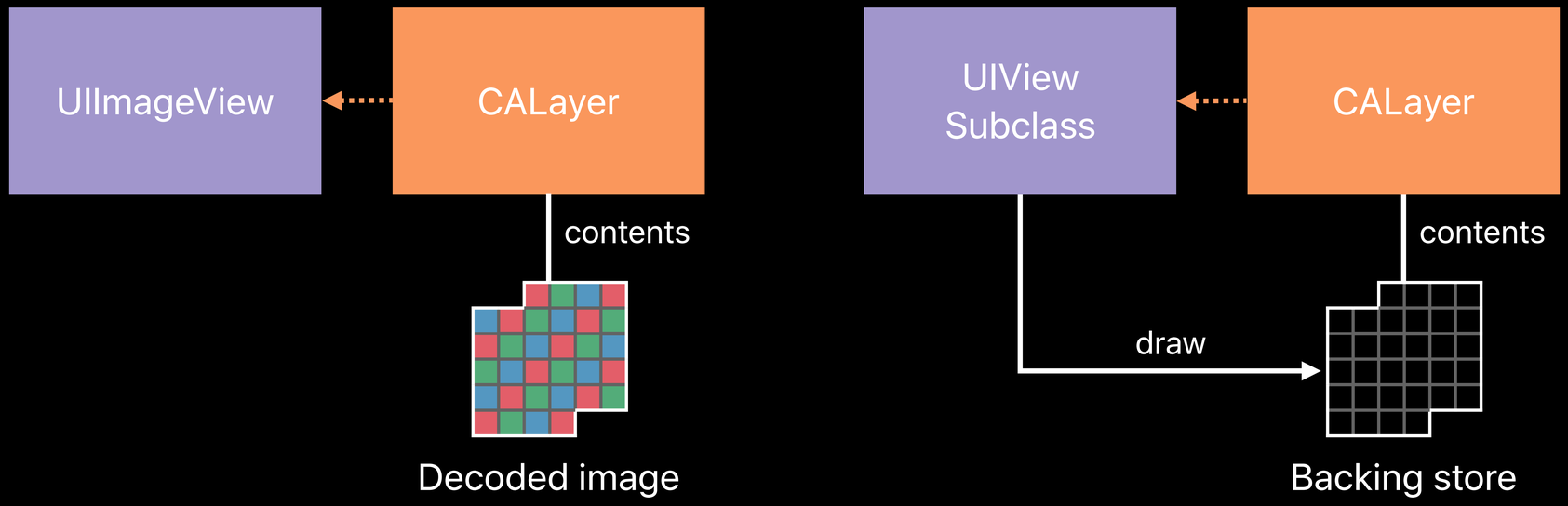
Custom Drawing Versus UIImageView



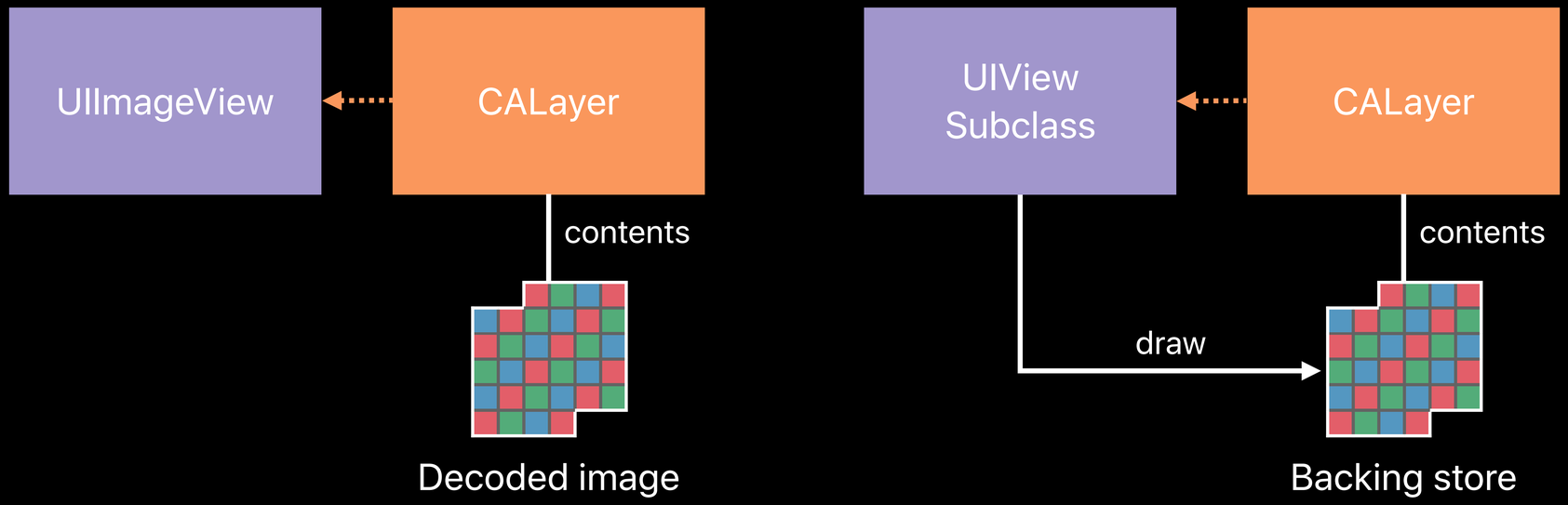
Custom Drawing Versus UIImageView



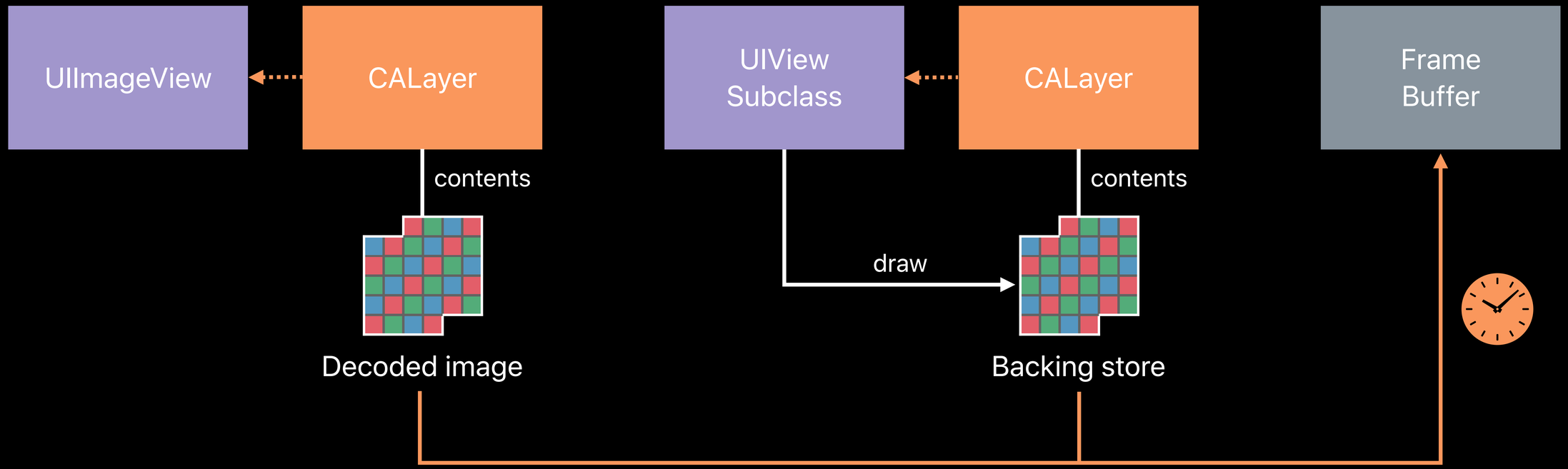
Custom Drawing Versus UIImageView



Custom Drawing Versus UIImageView



Custom Drawing Versus UIImageView



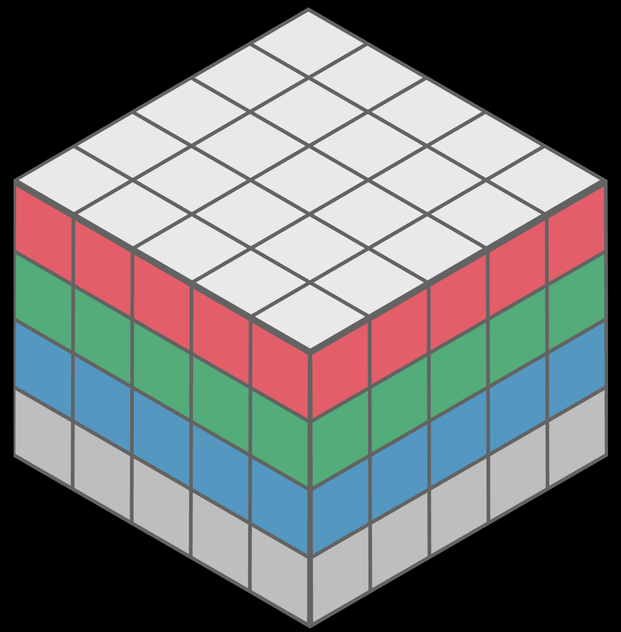
Backing Store Memory Costs

Proportional to pixel size of view

NEW Element size grows to accommodate color range used by drawing

Setting `CALayer.contentsFormat` opts out

Update `layerWillDraw(_:)` implementations



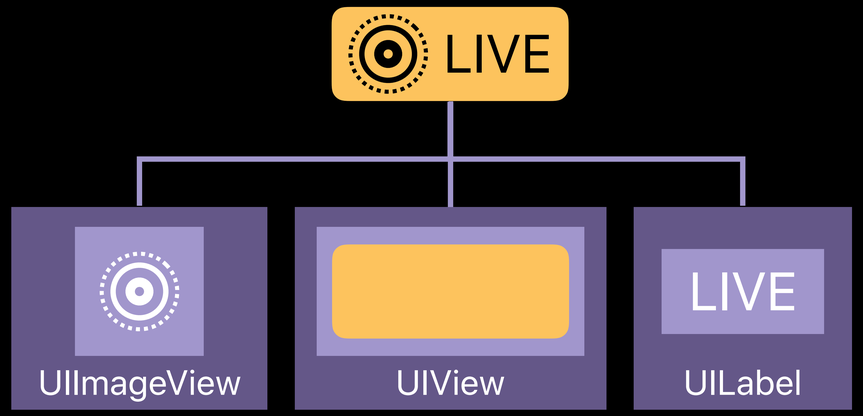
Reducing Backing Store Use

Refactor larger views into subview hierarchies

Reduce or eliminate overrides of `draw(_:)`

Eliminate duplicate copies of image data

Use optimized view properties and subclasses



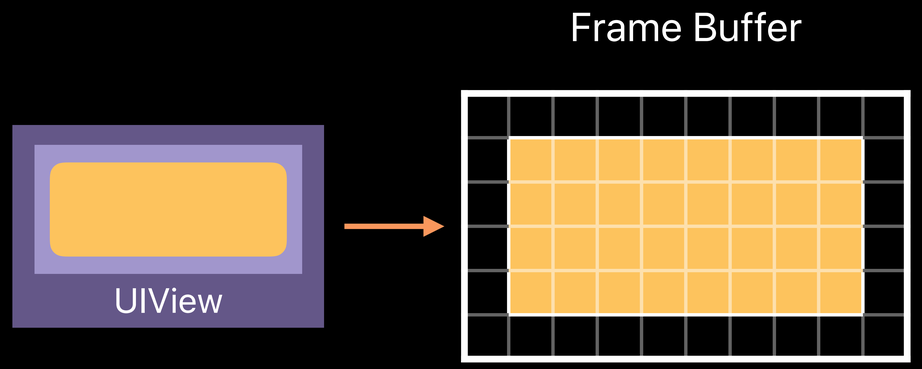
Reducing Backing Store Use

Alternatives to custom drawing

Overriding `draw(_:)` opts into backing store

`UIView.backgroundColor` can render directly to frame buffer without a backing store

- ...except for pattern colors
- Use UIImageView with tiling image instead



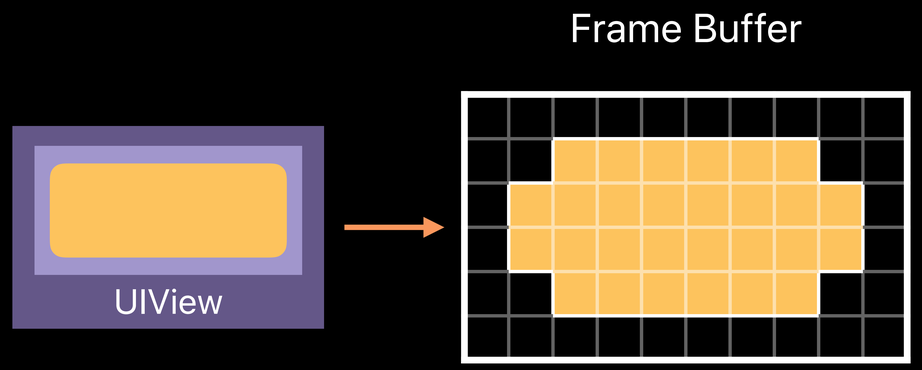
Reducing Backing Store Use

Masking versus corner radius

`UIView.maskView` and `CALayer.maskLayer` render view hierarchy into temporary image buffer

`CALayer.cornerRadius` does not require any image buffer

Consider UIImageView with resizable image instead of masking for transparent backgrounds



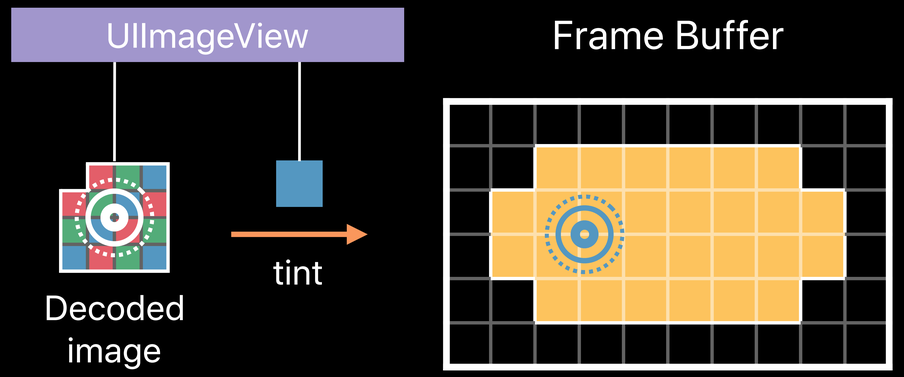
Reducing Backing Store Use

Eliminating duplicate image data

UIImageView can colorize monochrome images while rendering directly into frame buffer

`UIImage.withRenderingMode(_:)` or Rendering Mode popup in asset inspector

Set `tintColor` of image view to any solid color



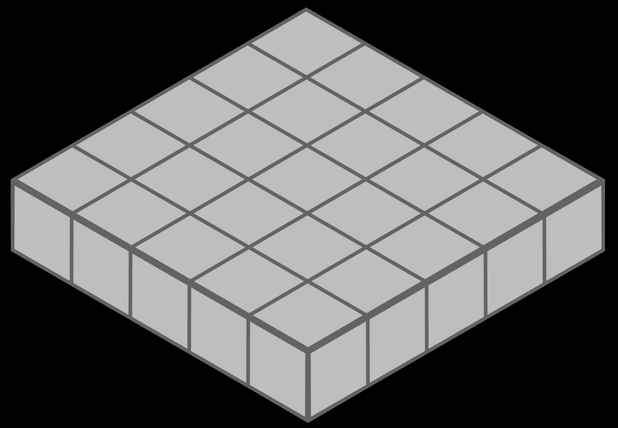
Reducing Backing Store Use

UILabel optimizations for rendering text

UILabel is optimized for monochrome strings

Uses 75% smaller backing store when possible

Automatically upgrades to larger backing store for multicolor strings, emoji



Drawing Off-Screen

Use UIGraphicsImageRenderer to create and draw to an image buffer

Supports Wide Color, unlike `UIGraphicsBeginImageContext()`

Combine with UIImageView for efficient offscreen rendering

Drawing Off-Screen

Optimizing image buffers

NEW

Similar automatic Wide Color support as backing stores

Prior to iOS 12 and tvOS 12, defaults to Wide Color support based on hardware

Set `prefersExtendedRange` on UIGraphicsImageRendererFormat for direct control

`UIImage.imageRendererFormat` may offer an intermediate representation

Advanced CPU and GPU Techniques

Advanced Image Effects

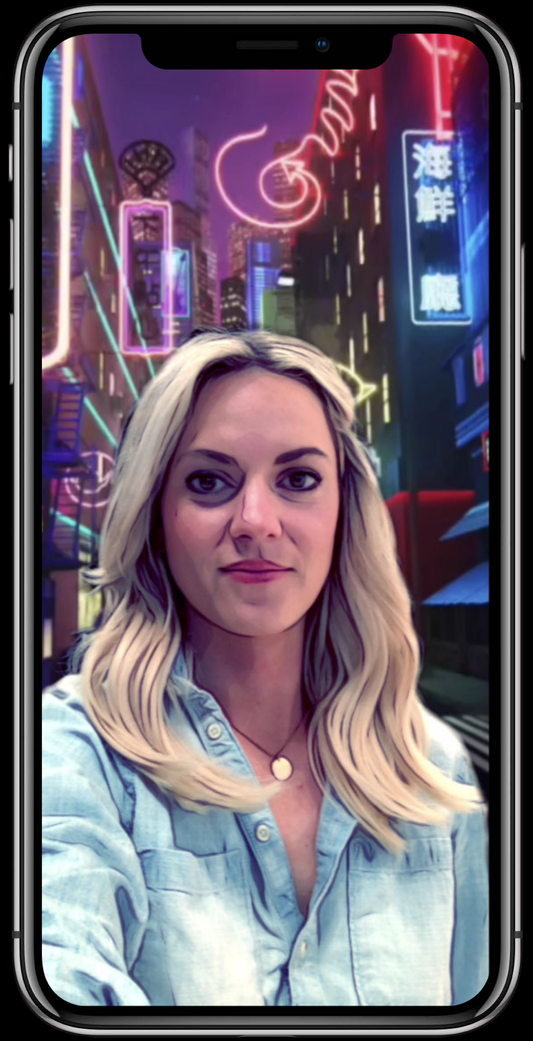
Core Image

Consider Core Image for realtime effects

Executes on GPU, freeing up CPU

UIImageView renders CIImages efficiently

```
UIImage.init(ciImage:)
```



Advanced Image Processing

Interfacing with other frameworks

Use CVPixelBuffer to move data to frameworks like Metal, Vision, and Accelerate

Use the best initializer—don’t unwind work that’s already been done

Guard against moving work between GPU and CPU

Ensure buffers are correct format for Accelerate

Using Accelerate and simd Executive Ballroom Tuesday 10:00AM

Summary

Implement prefetching to prepare asynchronously

Reduce backing store usage by using UIImageView and UILabel

Don’t accidentally disable new optimizations for custom drawing

Prefer image assets for bundled artwork

Avoid over-reliance on Preserve Vector Data

More Information

https://developer.apple.com/wwdc18/219

Practical Approaches to Great App Performance	Hall 1	Wednesday 5:00PM
UIKit and Layout	Technology Lab 2	Thursday 11:00AM
UIKit and Collection View Lab	Technology Lab 11	Friday 9:00AM


WWDC18