Layer-Backed Views

AppKit + Core Animation

Session 217

Corbin Dunn
AppKit Software Engineer

Before We Begin

Expectations

- Experience with traditional AppKit or UIKit drawing
- No experience with Core Animation is required

Effectively Using Layer-Backed NSViews

- Layer-backed views use Core Animation
- Layer-backing gives smoother, faster animations
  - Animations use less CPU as work is done in the GPU
  - Allows the use of the NSView hierarchy, events, and responder chain
- Historically layer-backed views use traditional AppKit drawing
  - AppKit has had the ability to use layers for many releases

Demo

Animation performance

Effectively Using Layer-Backed NSViews

Agenda

- Drawing
- Animating
  - Contents Updating
  - Synchronized Subview Animations
- Best Practices
  - Text and Font Smoothing
  - Focus Rings
- More Details and Tips

Drawing

Drawing

- Traditional NSView drawing model
- Using Core Animation layers and how they work
- Differences in `-setNeedsDisplay:` for redrawing

Traditional NSView Drawing Model

Custom NSView

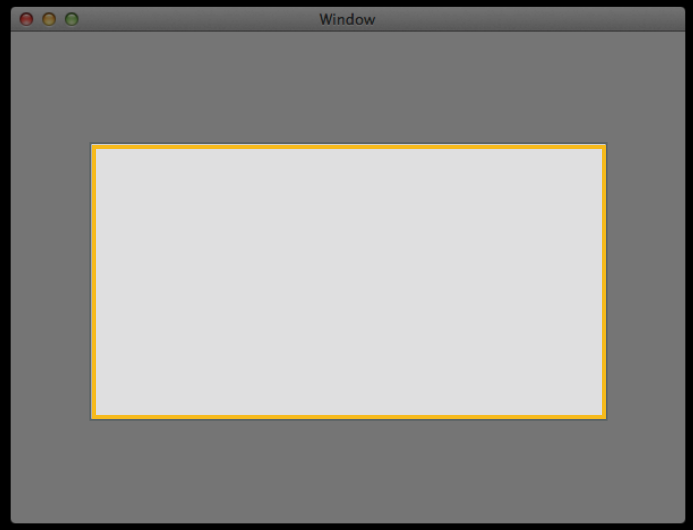
Traditional NSView Drawing Model

Custom drawRect: implementation


```
- (void)drawRect:(NSRect)dirtyRect {
    // Fill the contents
    [[NSColor lightGrayColor] set];
    NSRectFill(self.bounds);
    // Draw the border
    [[NSColor grayColor] set];
    NSFrameRect(self.bounds);
    // Draw the title
    [@"Tandem Unicycle" drawInRect:titleRect withAttributes:...];
    // Draw the image
    [image drawInRect:imageRect fromRect:.. operation:.. fraction:..];
}
```

Traditional NSView Drawing Model

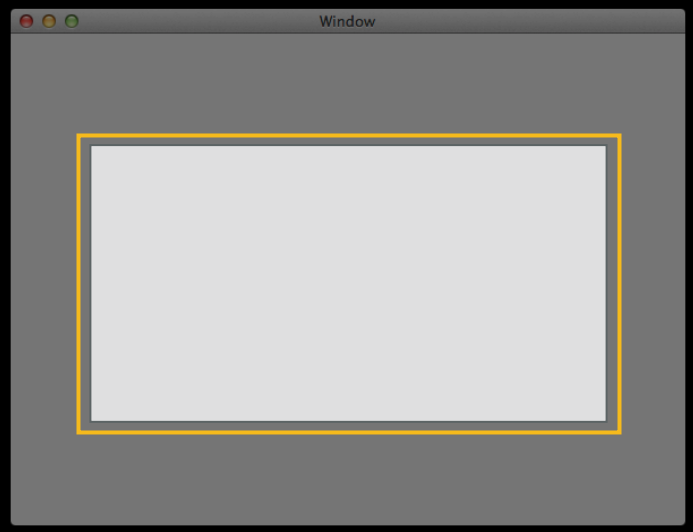
Custom drawRect: implementation




```
- (void)drawRect:(NSRect)dirtyRect {
    // Fill the contents
    [[NSColor lightGrayColor] set];
    NSRectFill(self.bounds);
    // Draw the border
    [[NSColor grayColor] set];
    NSFrameRect(self.bounds);
    // Draw the title
    [@"Tandem Unicycle" drawInRect:titleRect withAttributes:...];
    // Draw the image
    [image drawInRect:imageRect fromRect:.. operation:.. fraction:..];
}
```

Traditional NSView Drawing Model

Custom drawRect: implementation




```
- (void)drawRect:(NSRect)dirtyRect {
    // Fill the contents
    [[NSColor lightGrayColor] set];
    NSRectFill(self.bounds);
    // Draw the border
    [[NSColor grayColor] set];
    NSFrameRect(self.bounds);
    // Draw the title
    [@"Tandem Unicycle" drawInRect:titleRect withAttributes:...];
    // Draw the image
    [image drawInRect:imageRect fromRect:.. operation:.. fraction:..];
}
```

Traditional NSView Drawing Model

Custom drawRect: implementation


```
- (void)drawRect:(NSRect)dirtyRect {
    // Fill the contents
    [[NSColor lightGrayColor] set];
    NSRectFill(self.bounds);
    // Draw the border
    [[NSColor grayColor] set];
    NSFrameRect(self.bounds);
    // Draw the title
    [@"Tandem Unicycle" drawInRect:titleRect withAttributes:...];
    // Draw the image
    [image drawInRect:imageRect fromRect:.. operation:.. fraction:..];
}
```

Traditional NSView Drawing Model

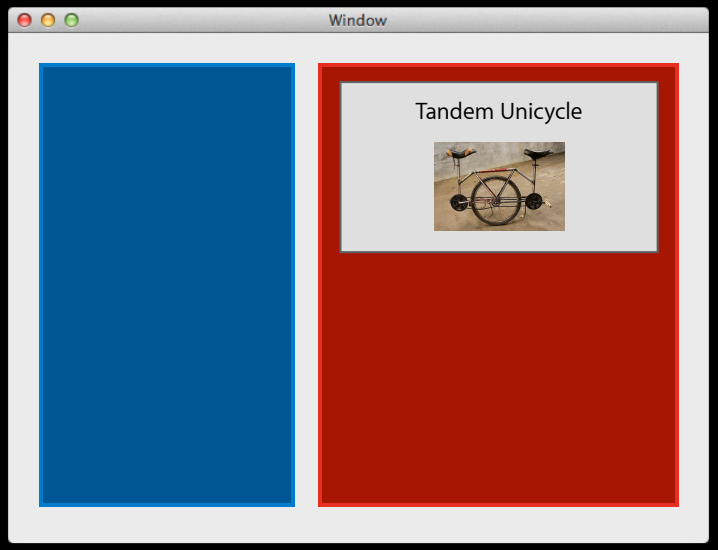
Custom drawRect: implementation


```
- (void)drawRect:(NSRect)dirtyRect {
    // Fill the contents
    [[NSColor lightGrayColor] set];
    NSRectFill(self.bounds);
    // Draw the border
    [[NSColor grayColor] set];
    NSFrameRect(self.bounds);
    // Draw the title
    [@"Tandem Unicycle" drawInRect:titleRect withAttributes:...];
    // Draw the image
    [image drawInRect:imageRect fromRect:.. operation:.. fraction:..];
}
```

Traditional NSView Drawing Model

What happens in NSWindow draw time

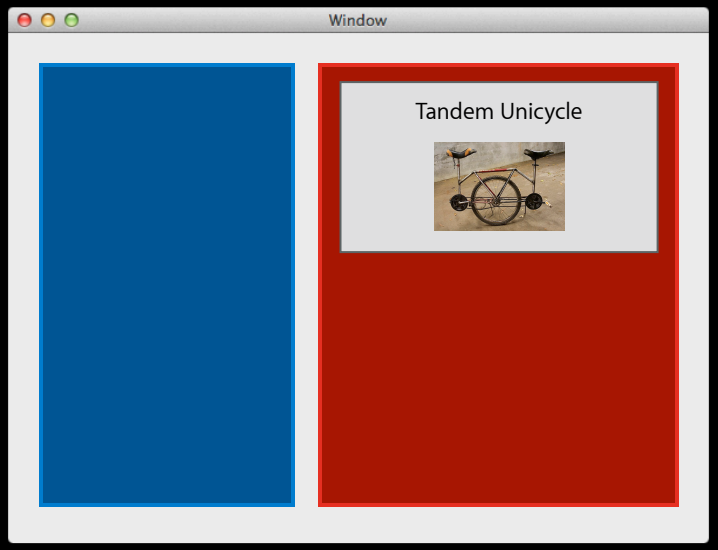
- NSWindow recursively draws all views in a dirty region
- Children draw on top of the parent



Traditional NSView Drawing Model

What happens in NSWindow draw time

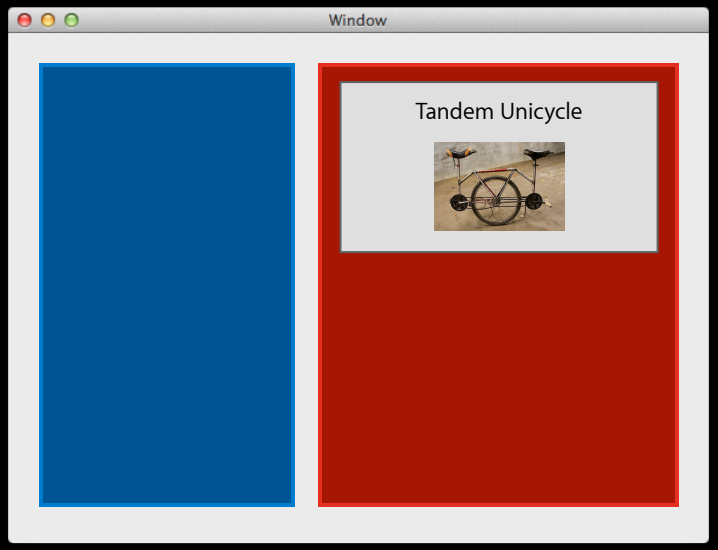
- NSWindow recursively draws all views in a dirty region
- Children draw on top of the parent



Traditional NSView Drawing Model

What happens in NSWindow draw time

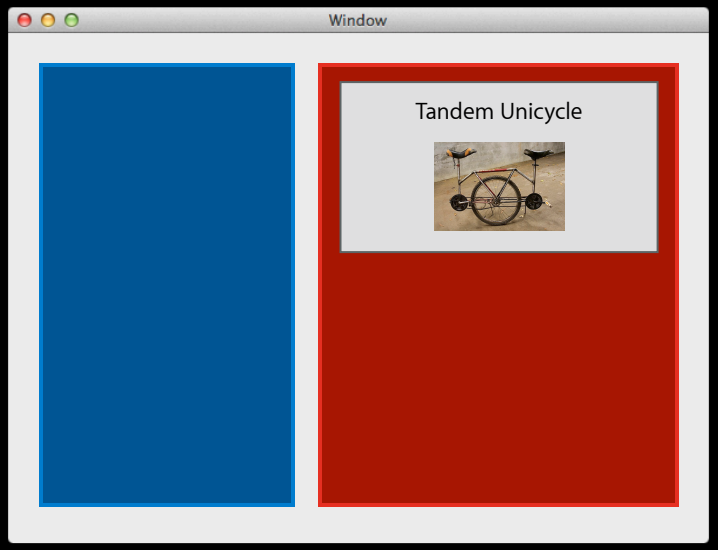
- NSWindow recursively draws all views in a dirty region
- Children draw on top of the parent



Traditional NSView Drawing Model

What happens in NSWindow draw time

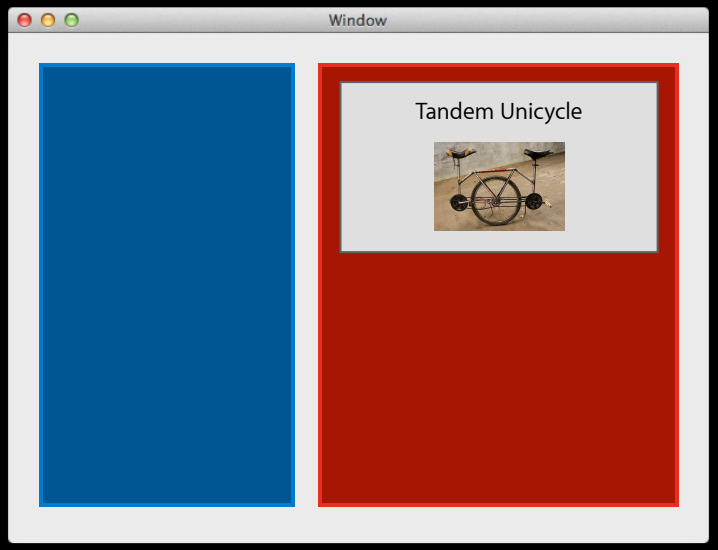
- NSWindow recursively draws all views in a dirty region
- Children draw on top of the parent



Traditional NSView Drawing Model

What happens in NSWindow draw time

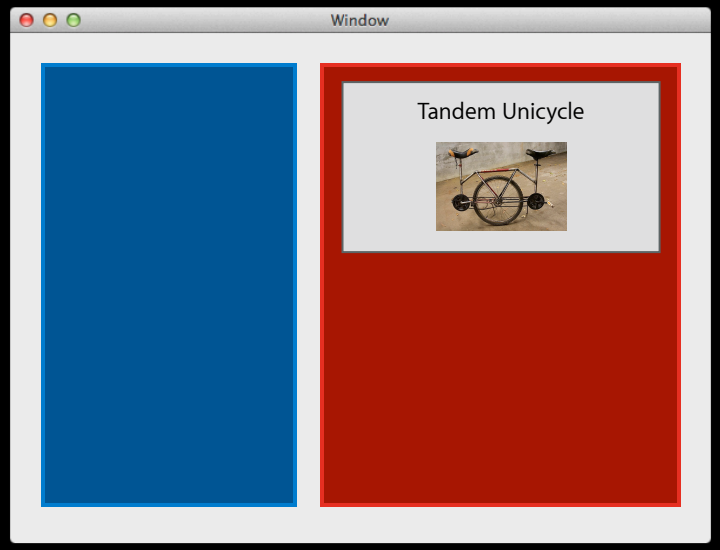
- NSWindow recursively draws all views in a dirty region
- Children draw on top of the parent



Traditional NSView Drawing Model

What happens in NSWindow draw time

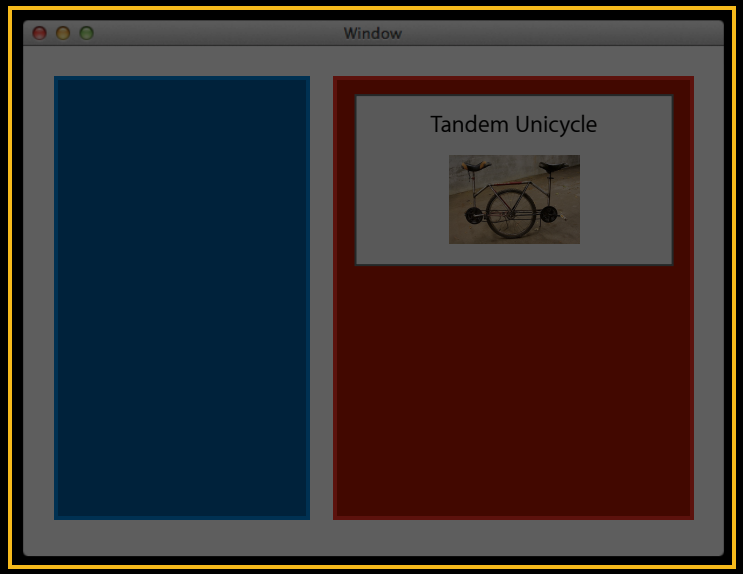
- Drawing into the window's backing store
- Similar to drawing into a large image



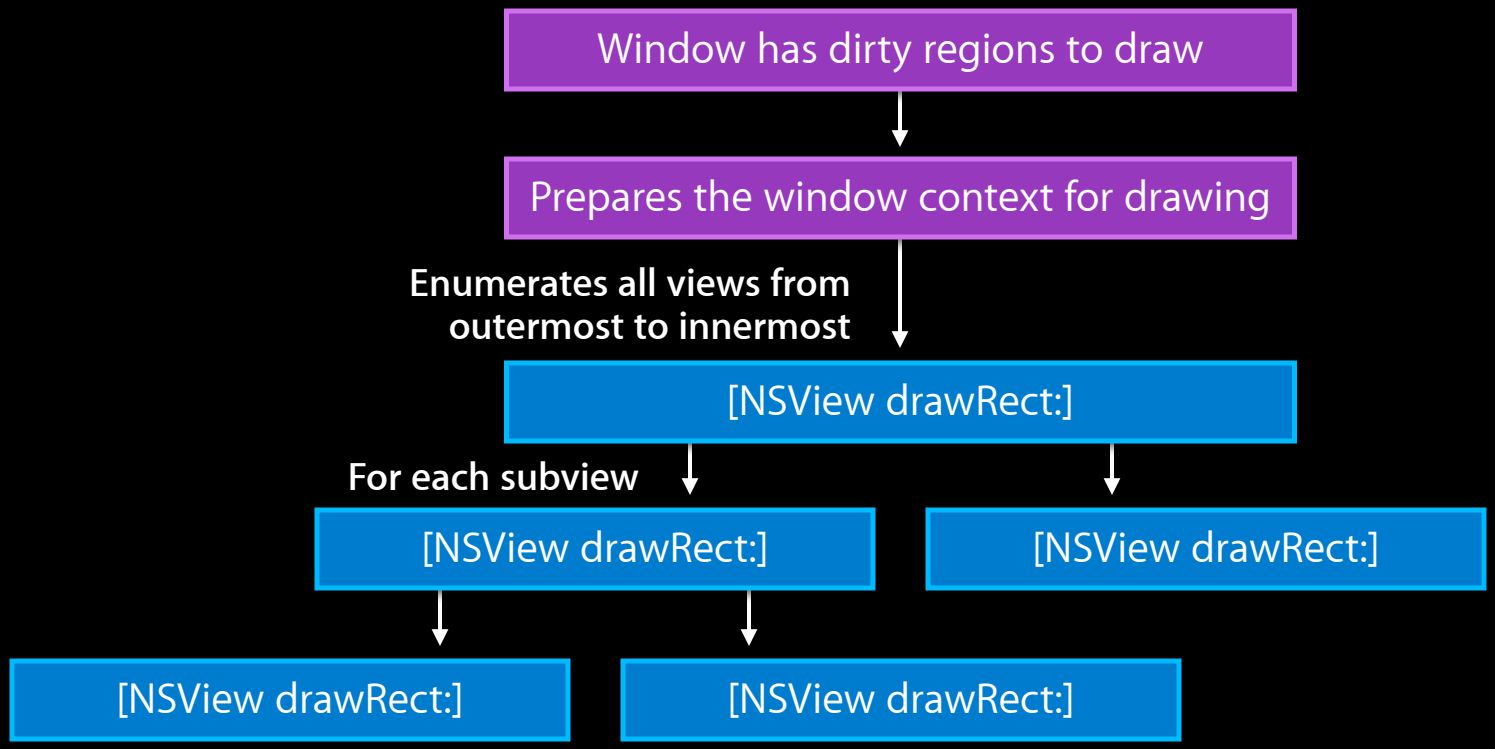
Traditional NSView Drawing Model

What happens in NSWindow draw time

- Drawing into the window's backing store
- Similar to drawing into a large image



Window Drawing Flow Chart

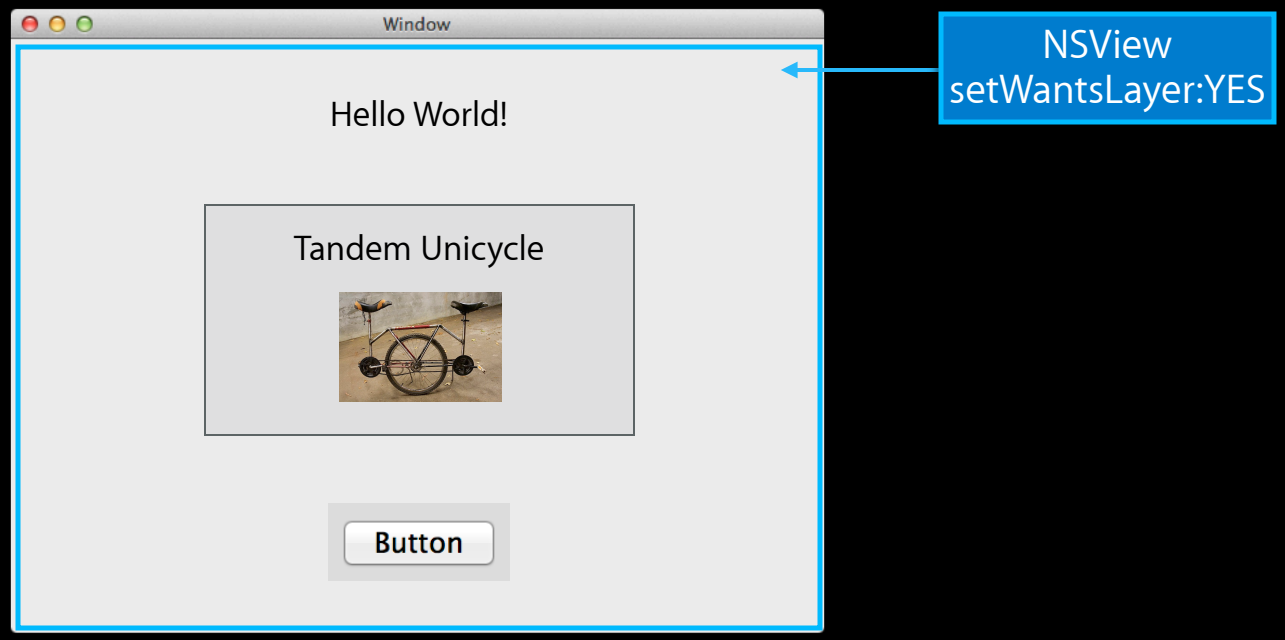


Core Animation and AppKit

How to use Core Animation layers

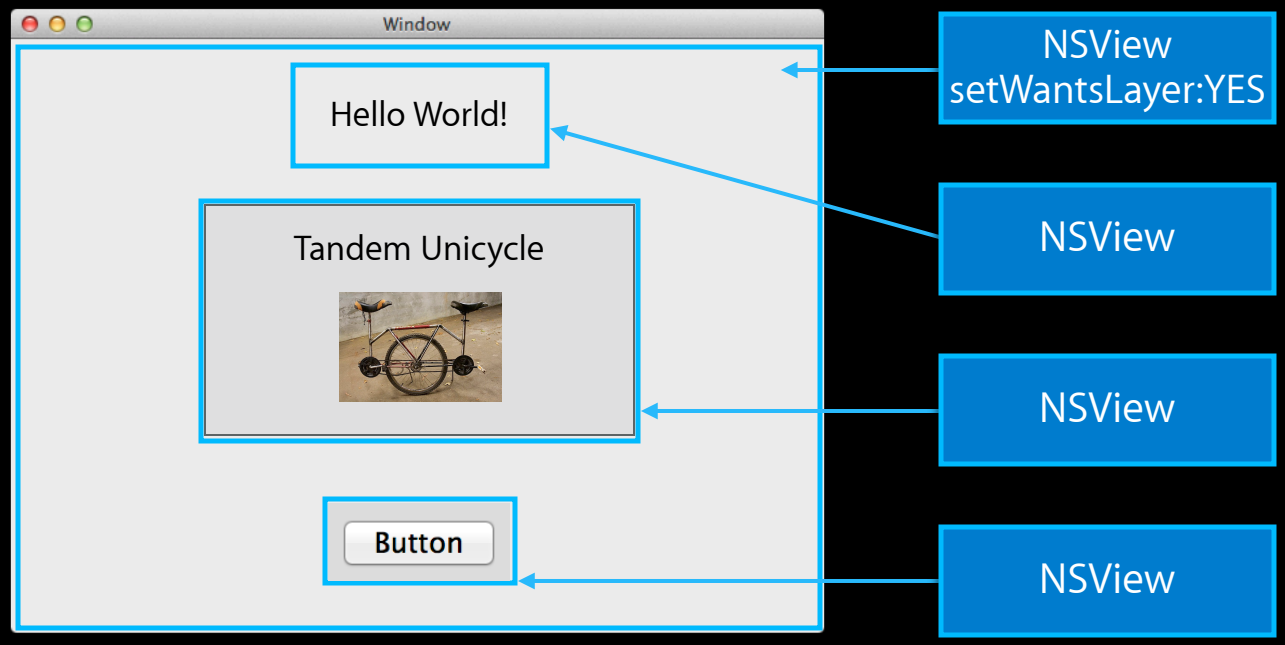
Core Animation and AppKit

Call setWantsLayer:YES on parent NSView



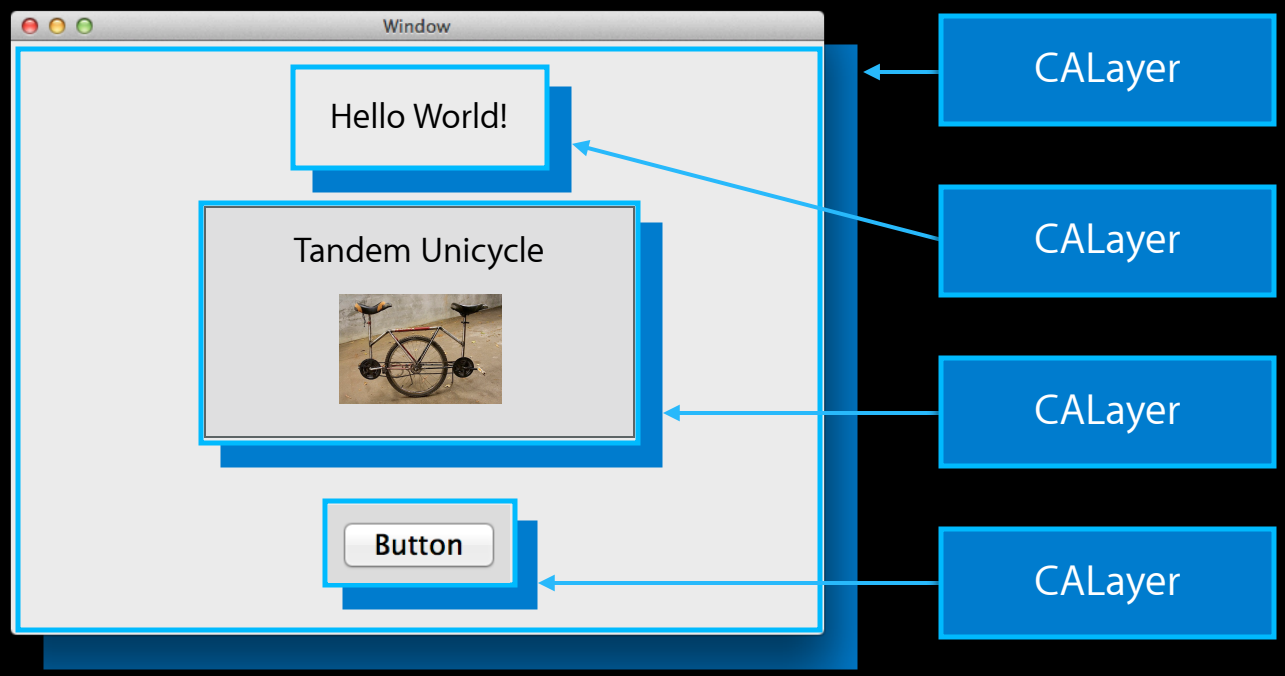
Core Animation and AppKit

All children views become layer-backed



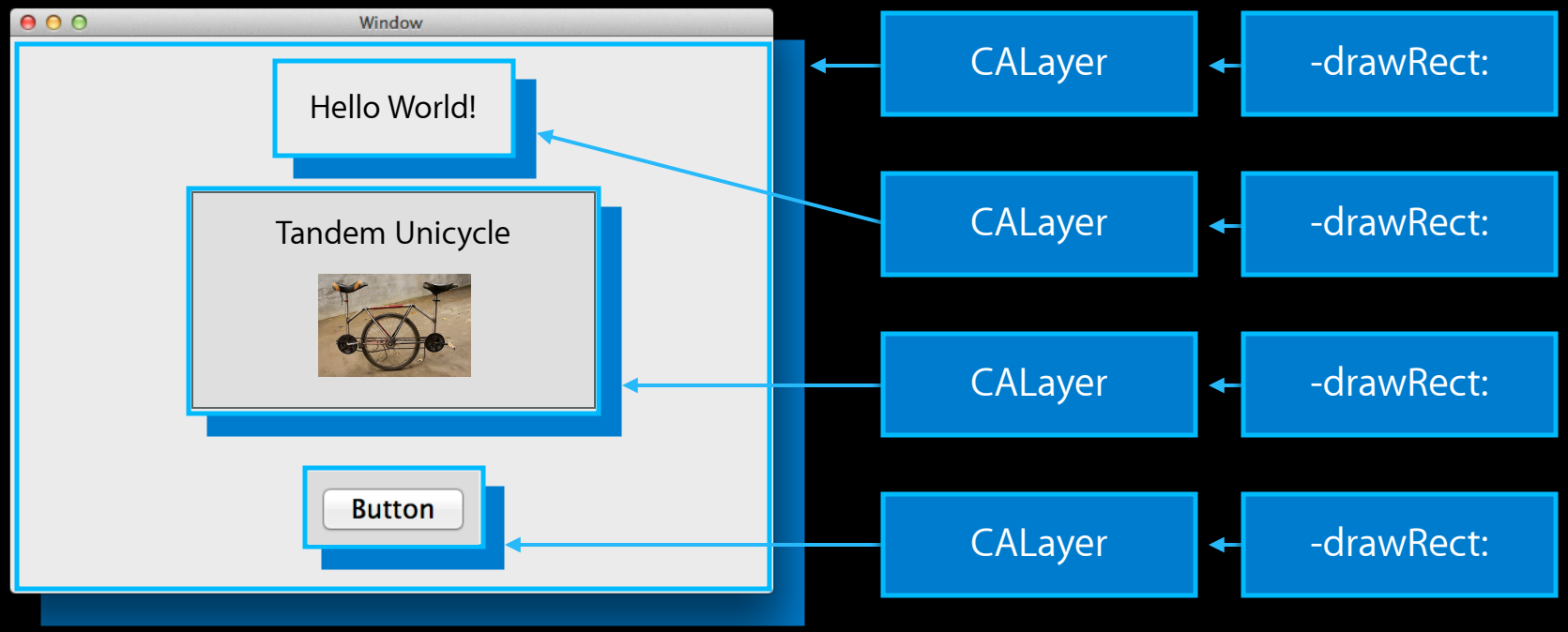
Core Animation and AppKit

All children views become layer-backed

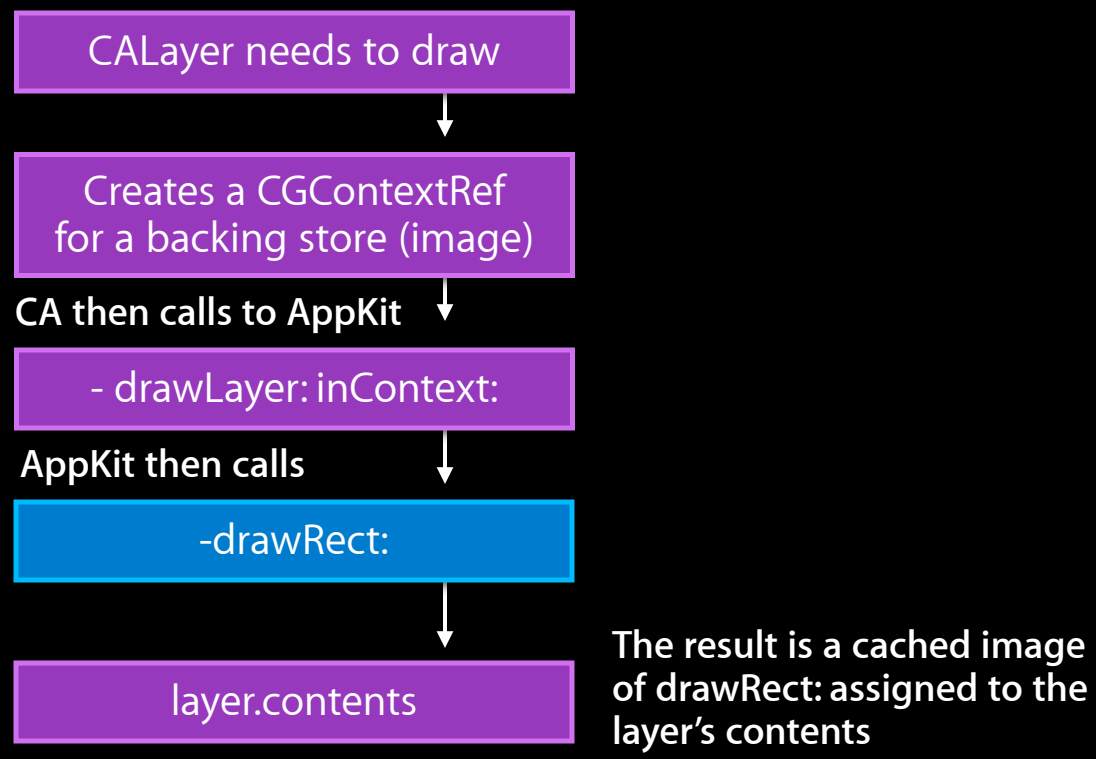


Core Animation and AppKit

Each filled in via drawRect:

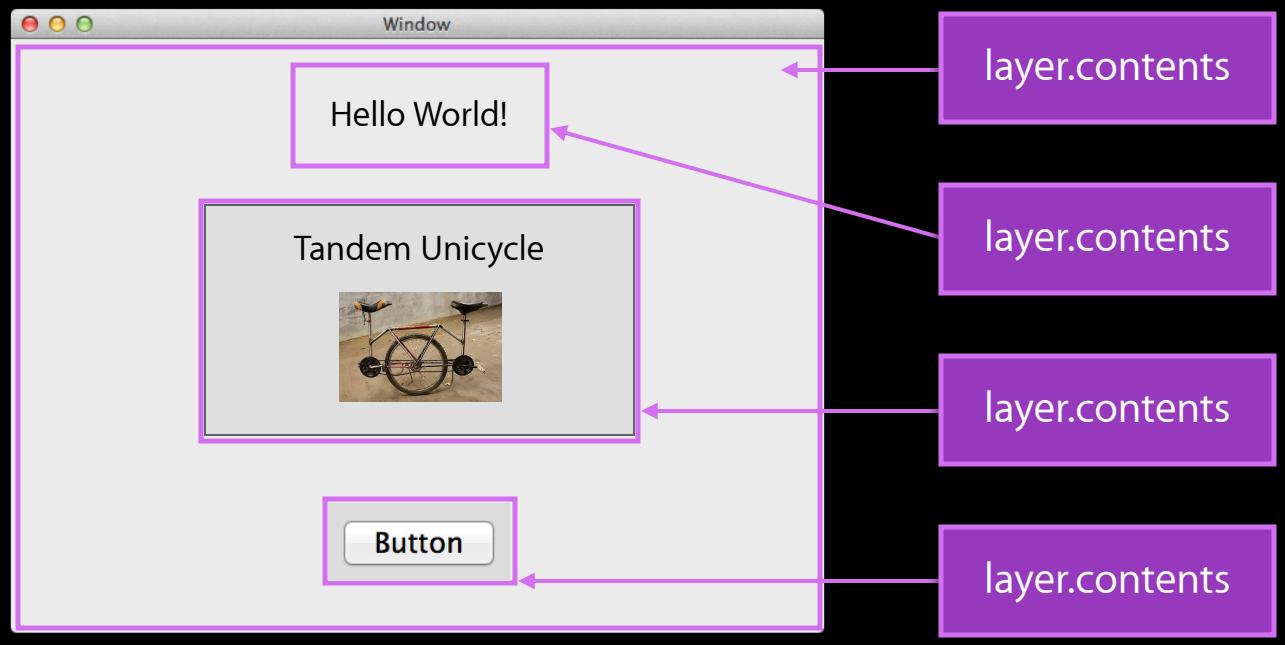


Layer Drawing Flow Chart



Layer Drawing Flow Chart

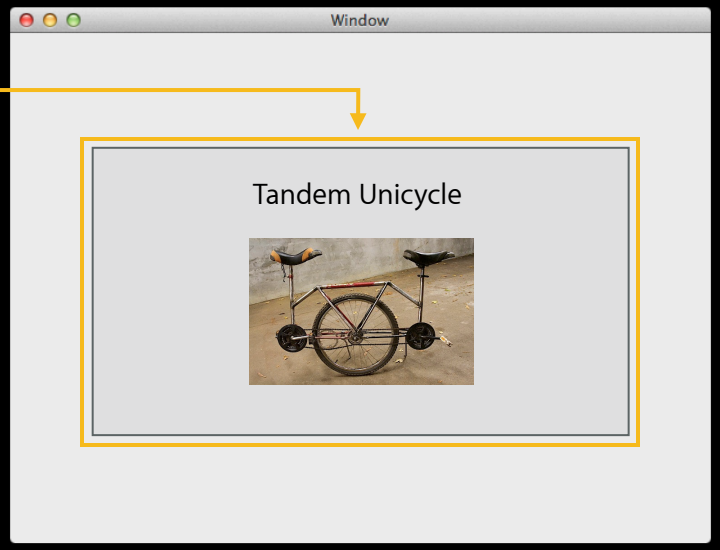
Layer contents are quickly composited to screen



Traditional NSView Drawing Model

Marking a view dirty

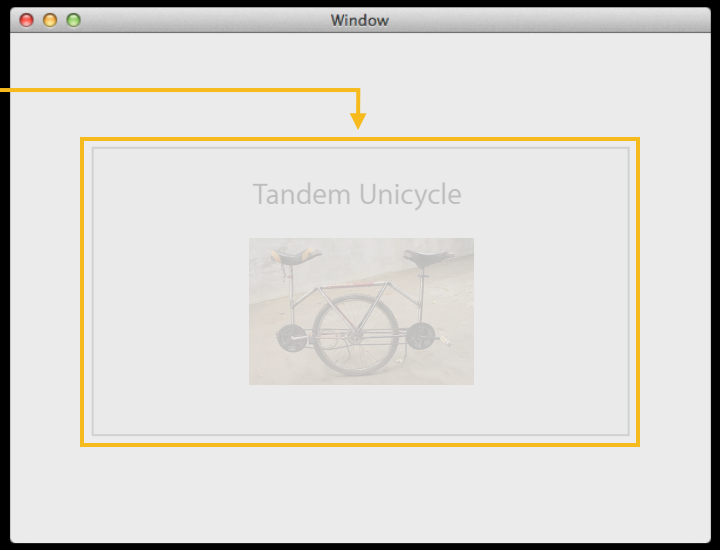
- Custom NSView marked as dirty via `setNeedsDisplay:YES`



Traditional NSView Drawing Model

Marking a view dirty

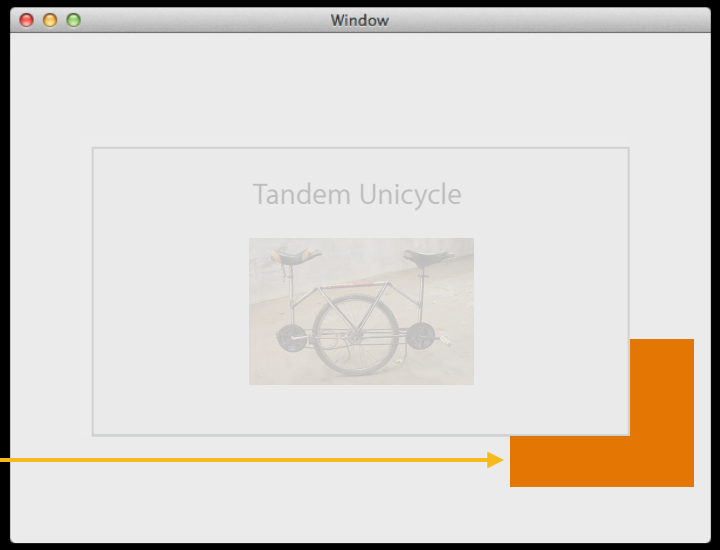
- Custom NSView marked as dirty via `setNeedsDisplay:YES`
- NSWindow keeps track of the dirty region



Traditional NSView Drawing Model

Marking a view dirty

- Custom NSView marked as dirty via `setNeedsDisplay:YES`
- NSWindow keeps track of the dirty region
- All NSViews in that region will redisplay
  - Including one like this under the other view



Traditional NSView Drawing Model

Marking a view dirty

- Subsequently, if this view is marked as `setNeedsDisplay:YES`

Traditional NSView Drawing Model

Marking a view dirty

- Subsequently, if this view is marked as `setNeedsDisplay:YES`
- This custom view will also be redrawn

Layers and setNeedsDisplay:

Each layer tracks its own dirty rect

- If this view has `setNeedsDisplay:YES` called on it
  - Only that view is marked as dirty
  - The window is not marked as dirty
  - This custom view will not get redrawn

Layers and setNeedsDisplay:

Each layer tracks its own dirty rect

- If this view has `setNeedsDisplay:YES` called on it
  - Only that view is marked as dirty
  - The window is not marked as dirty
  - This custom view will not get redrawn

Drawing

What we covered

- Traditional NSView drawing model
- Using Core Animation layers and how they work
- Differences in `-setNeedsDisplay:` for redrawing

Animating

Animating in AppKit

The animator proxy

```
@protocol NSAnimatablePropertyContainer

- (id)animator;
...
@end
```

- Implemented on NSView and NSWindow
- An opaque proxy object that can be treated just like original object
- Initiates implied animations on property changes
- The animator proxy starts and possibly drives the animation

Traditional Animating in AppKit

Use the animator proxy (layer-backed or not layer-backed)

- Use the animator proxy object to perform animations

```
NSRect frame = [view frame]; // size: 100, 100
frame.size = NSMakeSize(300, 300);
[[view animator] setFrame:frame];
// At this point the view.frame is still 100, 100
```

The proxy calls `setFrame:` on each step of the animation

- `drawRect:` is then invoked on each step of the animation

Traditional Animating in AppKit

Animations use the animator proxy

size = (100, 100) → size = (175, 175) → size = (250, 250) → size = (300, 300)

drawRect: → drawRect: → drawRect: → drawRect:

Tandem Unicycle

Tandem Unicycle

Tandem Unicycle

Tandem Unicycle

Done by AppKit on the main thread (non-optimal)

Core Animation + Frame Animations

CALayers do not need to have an animator proxy

```
CGRect frame = [layer frame]; // size: 100, 100
frame.size = CGSizeMake(300, 300);
layer.frame = frame; // Implicitly animates
// At this point the layer.frame is 300, 300!
```




Done by CA in a background thread

Core Animation + Frame Animations

CALayers do not need to have an animator proxy

- Layers simply stretch their image contents
- How they stretch is based on various CALayer properties

Done by CA in a background thread

Animating in AppKit with Core Animation

- All changes in Core Animation animate without the need for an animator proxy

```
layer.frame = newLayerFrame; // This will animate
```

- Layer-backed views in AppKit disable layer animations unless you are using the proxy object

```
[[view animator] setFrame:frame];
```

Turns on CALayer animations

Allows the layer to animate the frame

Redrawing Layer-Backed Views

Lion introduced – `[NSView layerContentsRedrawPolicy]`

- This property tells when AppKit should mark the layer as needing display
  - NSViewLayerContentsRedrawDuringViewResize
  - NSViewLayerContentsRedrawOnSetNeedsDisplay
  - NSViewLayerContentsRedrawBeforeViewResize
  - NSViewLayerContentsRedrawNever
- These options only apply to layer-backed views

Redrawing Layer-Backed Views

Lion introduced – `[NSView layerContentsRedrawPolicy]`

- This property tells when AppKit should mark the layer as needing display
  - NSViewLayerContentsRedrawDuringViewResize
  - NSViewLayerContentsRedrawOnSetNeedsDisplay
  - NSViewLayerContentsRedrawBeforeViewResize
  - NSViewLayerContentsRedrawNever
- These options only apply to layer-backed views

Redrawing Layer-Backed Views

`NSViewLayerContentsRedrawDuringViewResize`

- This is the default value for NSView!
- setNeedsDisplay is called whenever the frame changes

It is the most compatible with traditional `drawRect:` animations

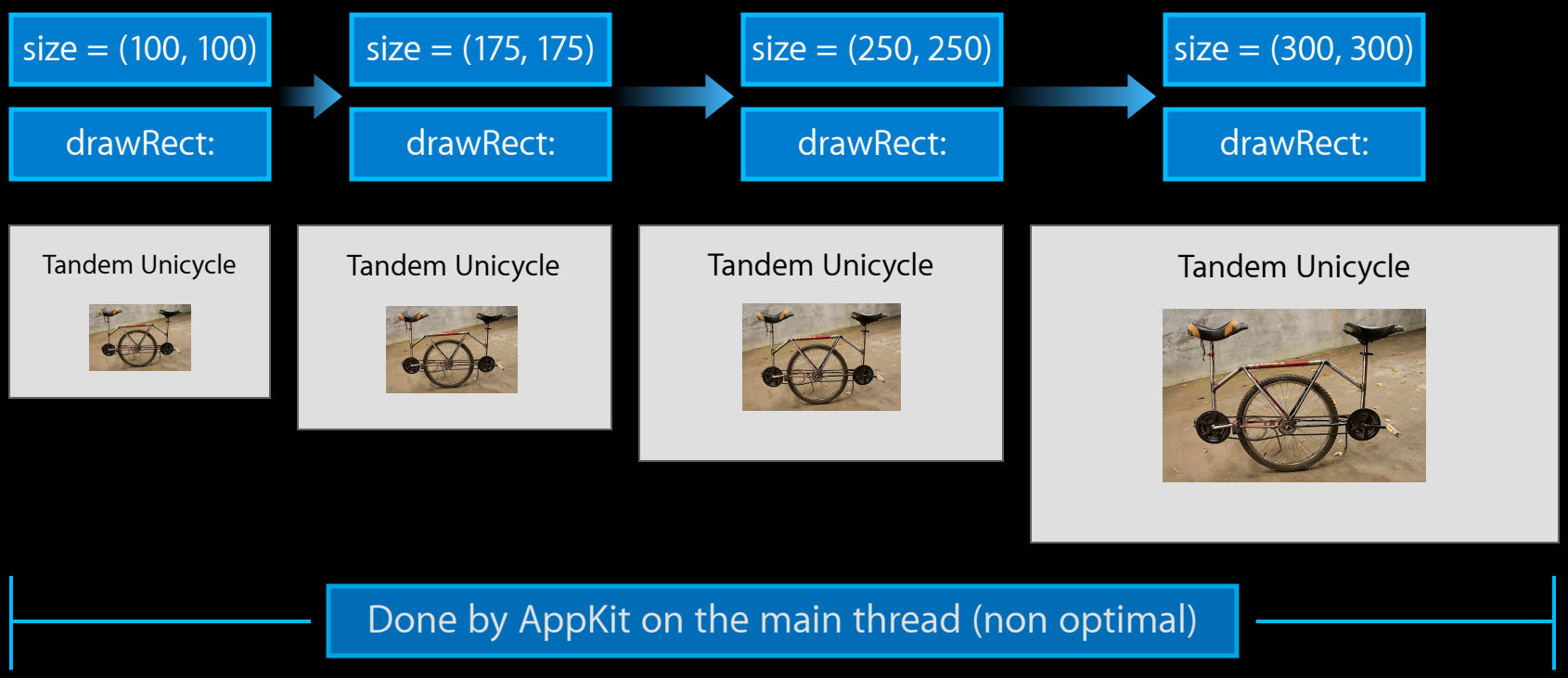
```
NSRect frame = [view frame]; // size: 100, 100
frame.size = NSMakeSize(300, 300);
[[view animator] setFrame:frame];
// At this point the view.frame is still 100, 100
```

The proxy calls `setFrame:` on each step of the animation

`drawRect:` is then called on each step of the animation

Layer-Backed View

Traditional animations (even when layer-backed)



Done by AppKit on the main thread (non optimal)

Redrawing Layer-Backed Views

Lion introduced – `[NSView layerContentsRedrawPolicy]`

- This property tells when AppKit should mark the layer as needing display
  - NSViewLayerContentsRedrawDuringViewResize
  - NSViewLayerContentsRedrawOnSetNeedsDisplay
  - NSViewLayerContentsRedrawBeforeViewResize
  - NSViewLayerContentsRedrawNever

Redrawing Layer-Backed Views

`NSViewLayerContentsRedrawOnSetNeedsDisplay`

- Doing: `[view setNeedsDisplay:YES]`
  - Means "invalidate the layer and lazily redraw"
- AppKit does not call `setNeedsDisplay:` when the frame changes!
- NOT the default value
  - Therefore, you MUST set it!

Redrawing Layer-Backed Views

NSViewLayerContentsRedrawOnSetNeedsDisplay

```
NSRect frame = [view frame]; // size: 100, 100
frame.size = NSMakeSize(300, 300);
[[view animator] setFrame:frame]; // Animates via Core Animation
// At this point the view frame is 300, 300!
```




Done by CA in a background thread

Problems with This Approach

`NSViewLayerContentsRedrawOnSetNeedsDisplay`

- Animation driven by Core Animation\
- `drawRect:` is NOT called on each step of the animation
- The contents are stretched

Problems with This Approach

`NSViewLayerContentsRedrawOnSetNeedsDisplay`

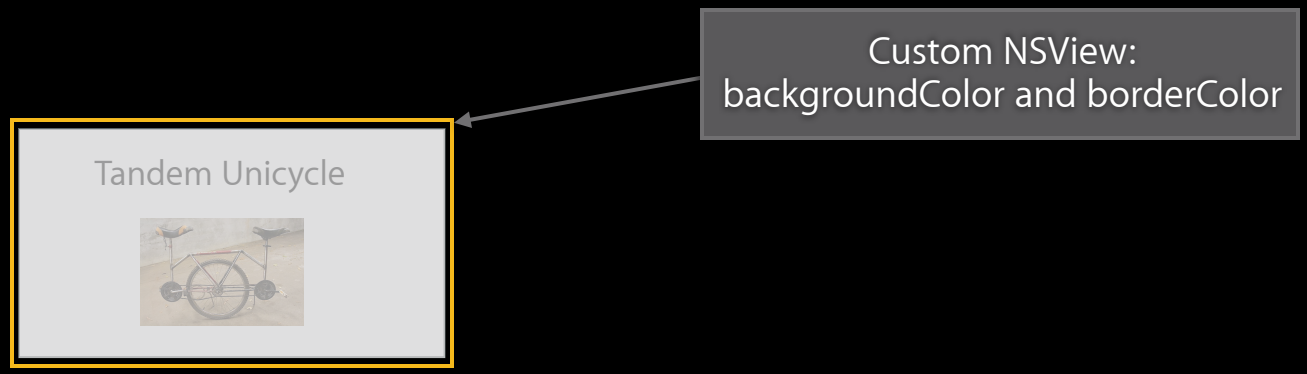
- Animation driven by Core Animation\
- `drawRect:` is NOT called on each step of the animation
- The contents are stretched

Text is stretched/blurry
and the image is stretched

Solution to This Approach

`NSViewLayerContentsRedrawOnSetNeedsDisplay`

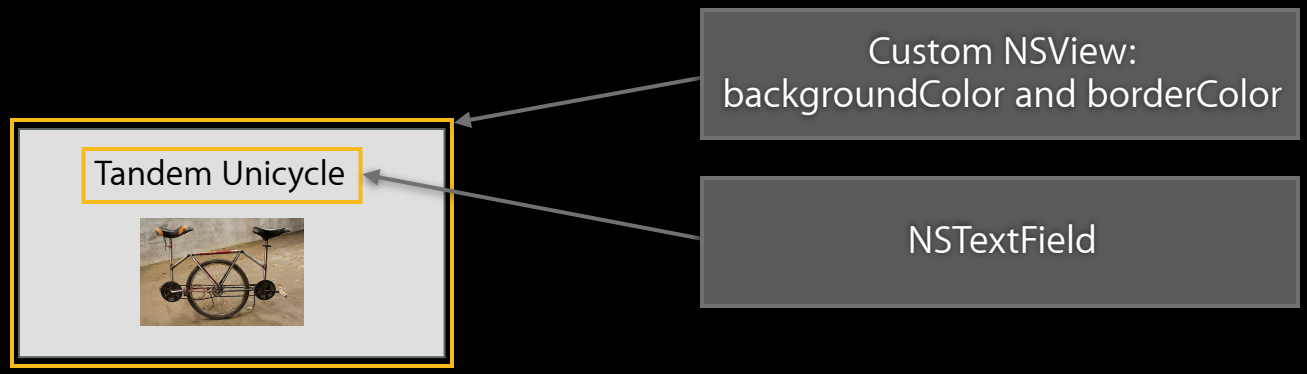
- Be sure to use the proper autoresizingMask or auto layout
- Layout views in Interface Builder or runtime



Solution to This Approach

`NSViewLayerContentsRedrawOnSetNeedsDisplay`

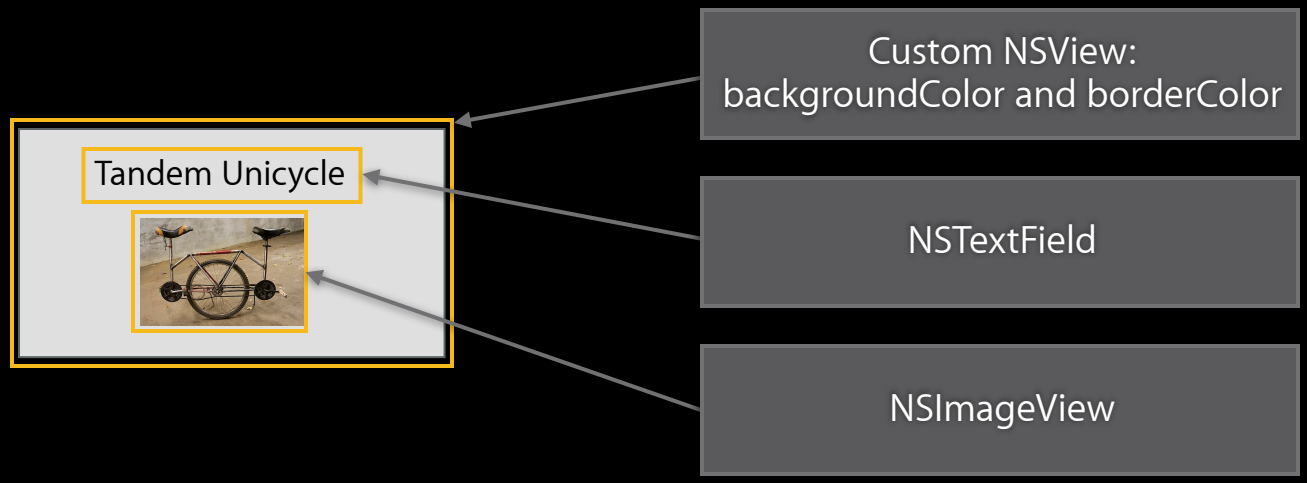
- Be sure to use the proper autoresizingMask or auto layout
- Layout views in Interface Builder or runtime



Solution to This Approach

`NSViewLayerContentsRedrawOnSetNeedsDisplay`

- Be sure to use the proper autoresizingMask or auto layout
- Layout views in Interface Builder or runtime



Compositing with Subviews

`NSViewLayerContentsRedrawOnSetNeedsDisplay`

- Animation is smooth and not redrawing (just moving layers)
- Text does not change size
  - Just moves the position
- The individual image view stretches its contents properly

Tandem Unicycle

Tandem Unicycle

Tandem Unicycle

Redrawing Layer-Backed Views

Lion introduced – `[NSView layerContentsRedrawPolicy]`

- This property tells when AppKit should mark the layer as needing display
  - NSViewLayerContentsRedrawDuringViewResize
  - NSViewLayerContentsRedrawOnSetNeedsDisplay
  - NSViewLayerContentsRedrawBeforeViewResize
  - NSViewLayerContentsRedrawNever

Redrawing Layer-Backed Views

NSViewLayerContentsRedrawBeforeViewResize

- Redraws the layer once using the "final size" right before the frame animation starts
- Thus uses that "final size" image for the contents while animating
- Contents look crisp at the end of the animation
  - However, they might look shrunken at the start of the animation

Layer-Backed View

NSViewLayerContentsRedrawBeforeViewResize

```
[[view animator] setFrame:(0,0,300,300)];
```

size = (100, 100)

drawRect:

size = (300, 300)

drawRect:

Tandem Unicycle

Layer-Backed View

NSViewLayerContentsRedrawBeforeViewResize

```
[[view animator] setFrame:(0,0,300,300)];
```

- At the animation start, the final size is redrawn for the layer contents

size = (100, 100)

drawRect:

size = (300, 300)

drawRect:

Tandem Unicycle

Layer-Backed View

NSViewLayerContentsRedrawBeforeViewResize

```
[[view animator] setFrame:(0,0,300,300)];
```

- At the animation start, the final size is redrawn for the layer contents

size = (100, 100)

drawRect:

size = (300, 300)

drawRect:

The layer contents will be shrunken and down sampled. This might look bad!

Layer-Backed View

NSViewLayerContentsRedrawBeforeViewResize

```
[[view animator] setFrame:(0,0,300,300)];
```

- At the animation start, the final size is redrawn for the layer contents

size = (100, 100)

drawRect:

size = (300, 300)

drawRect:

Layer-Backed View

NSViewLayerContentsRedrawBeforeViewResize

```
[[view animator] setFrame:(0,0,300,300)];
```

- At the animation start, the final size is redrawn for the layer contents

size = (100, 100)

drawRect:

size = (300, 300)

drawRect:

Tandem Unicycle

Redrawing Layer-Backed Views

Lion introduced – `[NSView layerContentsRedrawPolicy]`

- This property tells when AppKit should mark the layer as needing display
  - NSViewLayerContentsRedrawDuringViewResize
  - NSViewLayerContentsRedrawOnSetNeedsDisplay
  - NSViewLayerContentsRedrawBeforeViewResize
  - NSViewLayerContentsRedrawNever

Redrawing Layer-Backed Views

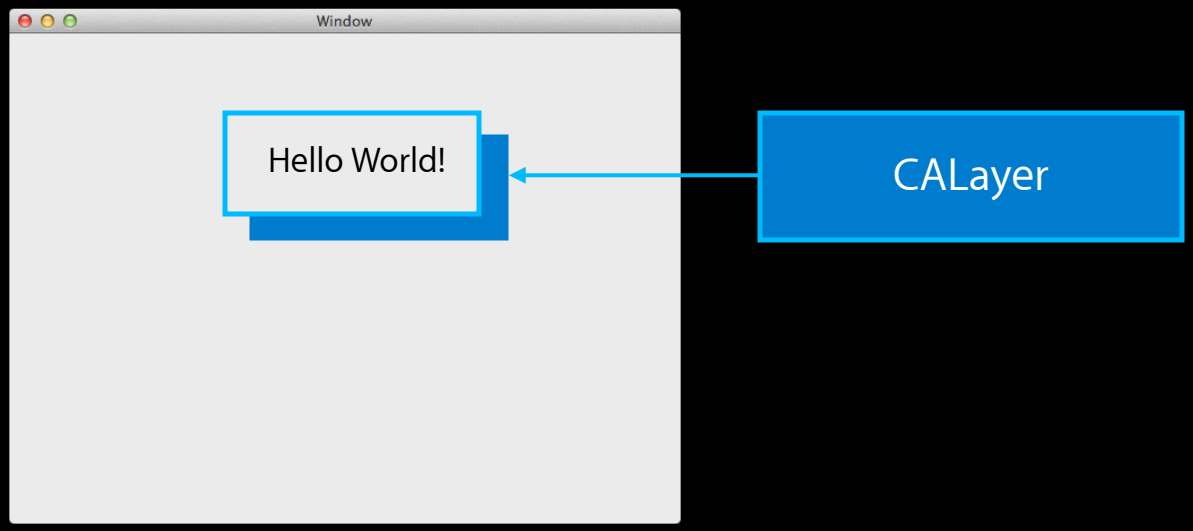
`NSViewLayerContentsRedrawNever`

- Tells the view to never redraw the layer
- Calling `[view setNeedsDisplay:YES]` does nothing!
- Generally, this is only useful in limited cases
- Good for layer-hosted views

Layer-Backed vs. Layer-Hosted

Layer-backed

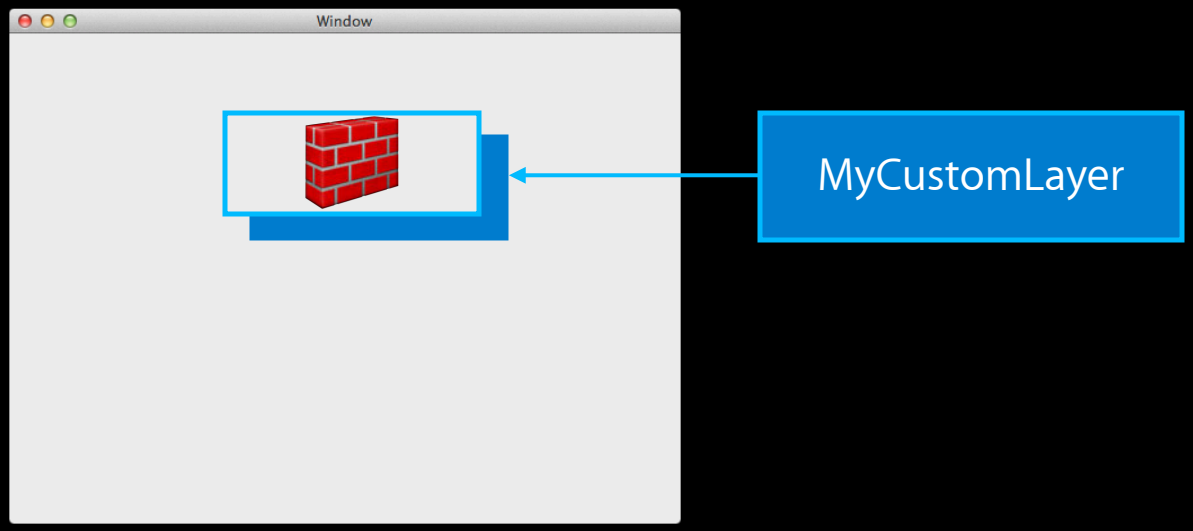
- The layer behind the view is created and managed by AppKit
- AppKit creates and "owns" the layer and will control most properties



Layer-Backed vs. Layer-Hosted

Layer-hosted

- The layer behind the view is assigned by the developer
- Use -[NSView setLayer:]
- AppKit keeps a “hands-off” approach to the layer

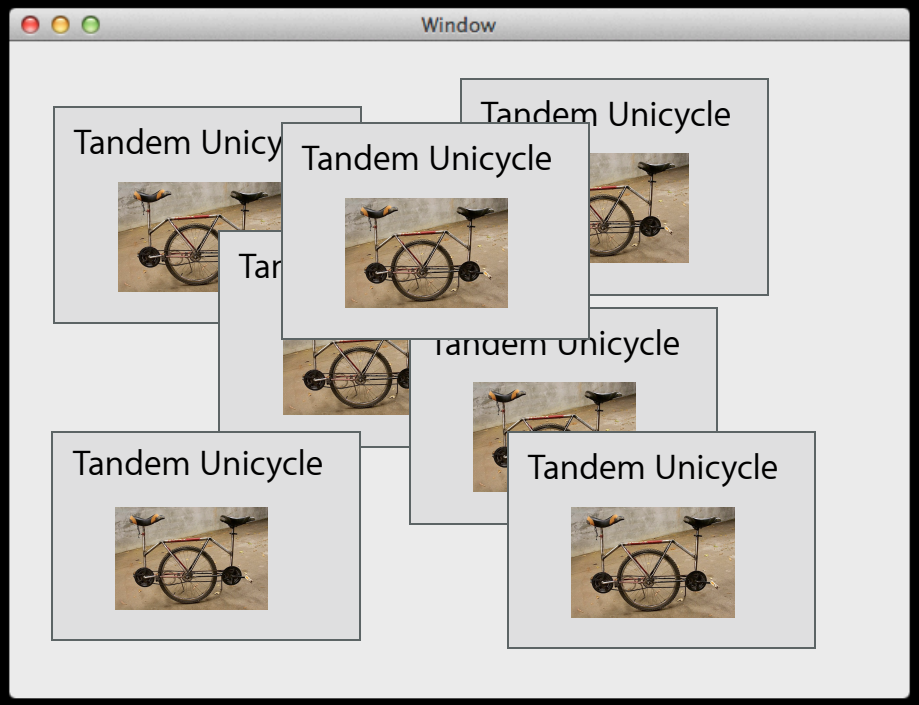


Animating

Contents updating

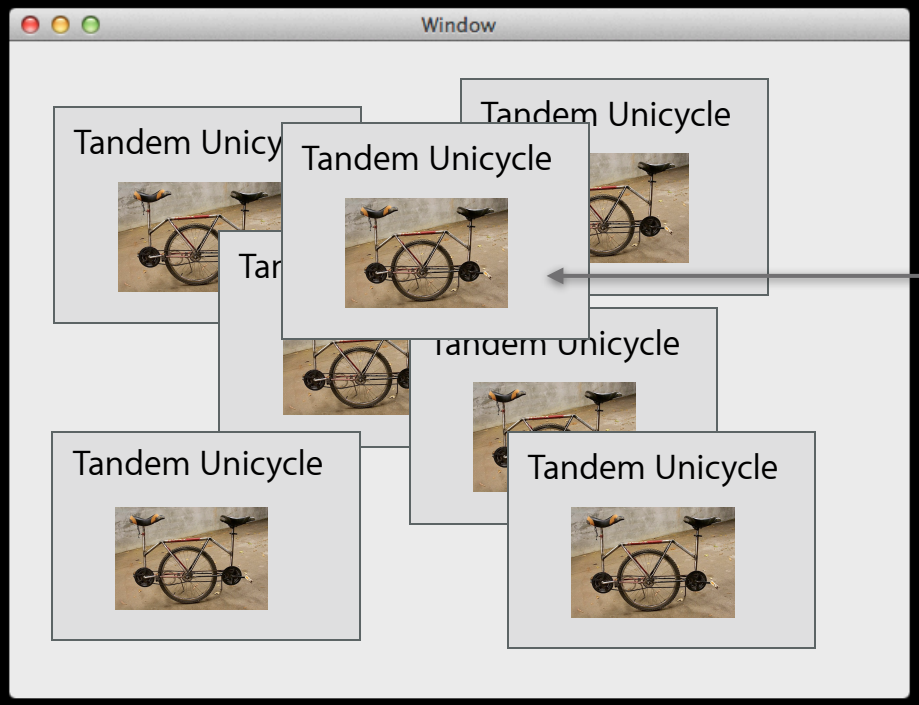
Great! Animations Are Smooth

…Now memory usage is high!



Great! Animations Are Smooth

…Now memory usage is high!



Each CALayer has its own backing image filled in by -drawRect:

Improving Layer-Backed Memory Use

Do not use -drawRect:!

```
- (void)d      t:(NSRect)d      ct {
    // Fil      contents
    [[NSColo      tGray      set];
    NSRectFil      b      ;
    // Draw the
    [[NSColor gra      set];
    NSFrameRect(s      nds);
    // Draw the
    [@"Tandem      cle      vInRect:titleRect withAttributes:...];
    // Draw      age
    [image      nRect:ima      fromRect:.. operation:.. fraction:..];
}
```

Improving Layer-Backed Memory Use

Techniques to save memory

- Use CALayer properties

  `backgroundColor`

  `borderColor`

- Directly set the `layer.contents` (to an image)
  - Share the same contents in multiple views
  - Stretch small images larger

Improving Layer-Backed Memory Use

Understanding CALayer Properties

```
CALayer *layer = [CALayer layer];
layer.backgroundColor = NSColor.whiteColor.CGColor;
layer.borderColor = NSColor.redColor.CGColor;
layer.borderWidth = 2.0;
```

Improving Layer-Backed Memory Use

Understanding CALayer Properties


```
CALayer *layer = [CALayer layer];
layer.backgroundColor = NSColor.whiteColor.CGColor;
layer.borderColor = NSColor.redColor.CGColor;
layer.borderWidth = 2.0;
```

Improving Layer-Backed Memory Use

Understanding CALayer Properties


```
CALayer *layer = [CALayer layer];
layer.backgroundColor = NSColor.whiteColor.CGColor;
layer.borderColor = NSColor.redColor.CGColor;
layer.borderWidth = 2.0;
```

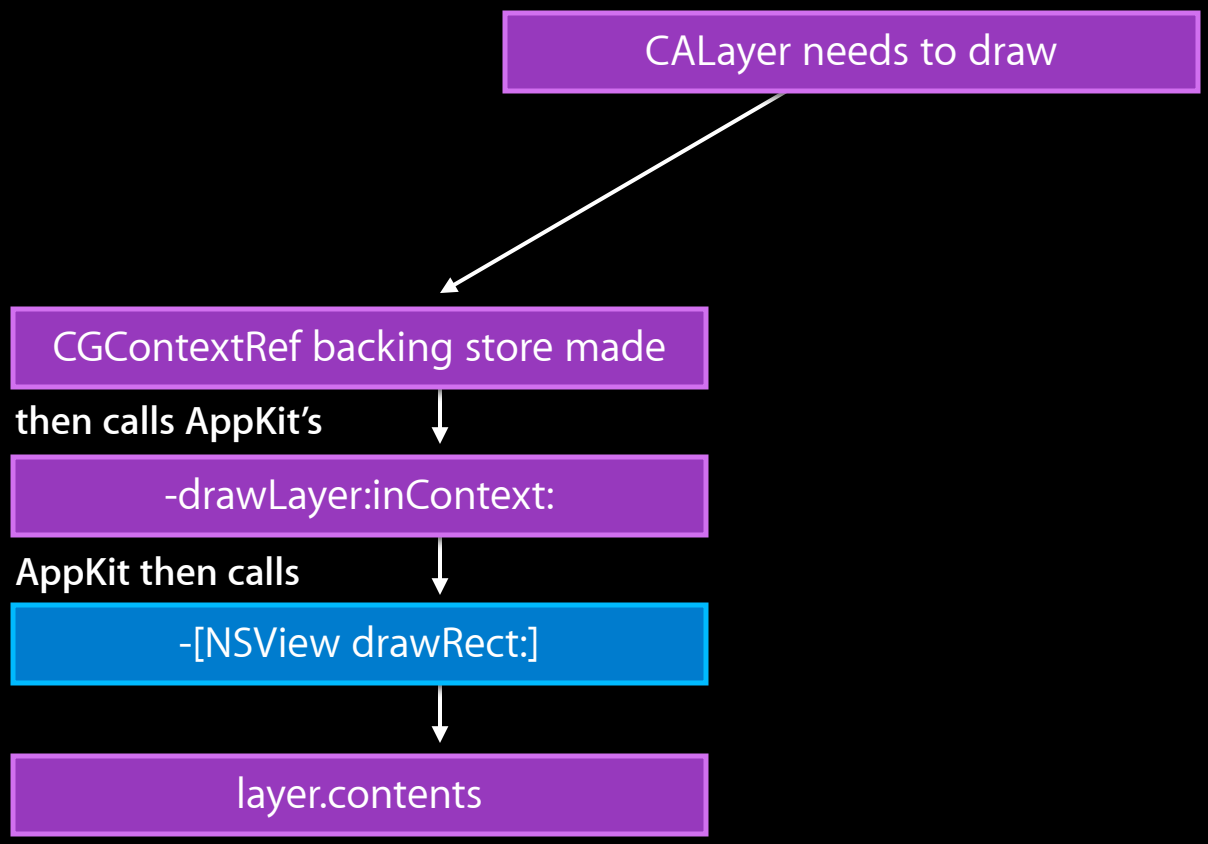
CGColorRef

Improving Layer-Backed Memory Use

Understanding CALayer Properties

```
CALayer *layer = [CALayer layer];
layer.backgroundColor = NSColor.whiteColor.CGColor;
layer.borderColor = NSColor.redColor.CGColor;
layer.borderWidth = 2.0;
layer.contents = [NSImage imageNamed:@"Unicycle"];
```

Traditional Layer Updating Flow Chart



Mountain Lion Layer Updating Flow Chart

Improving Layer-Backed Memory Use

Use -wantsUpdateLayer and -updateLayer


```
- (BOOL)wantsUpdateLayer {
   return YES;
}
- (void)updateLayer {
    self.layer.backgroundColor = NSColor.whiteColor.CGColor;
    self.layer.borderColor = NSColor.redColor.CGColor;
}
```

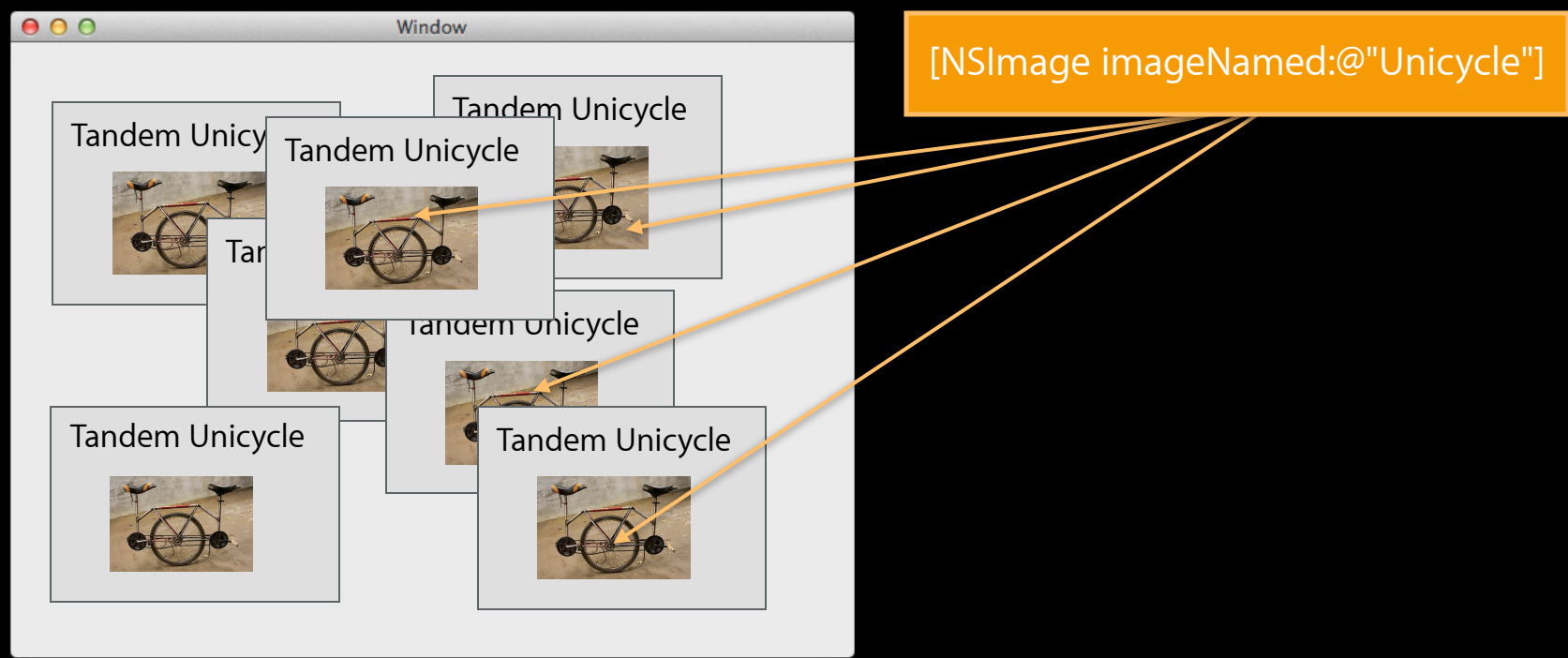
Improving Layer-Backed Memory Use

Alternative to -drawRect:

```
- (void)updateLayer {
    self.layer.backgroundColor = NSColor.whiteColor.CGColor;
    self.layer.borderColor = NSColor.redColor.CGColor;
    self.layer.contents = [NSImage imageNamed:@"Unicycle"];
}
```

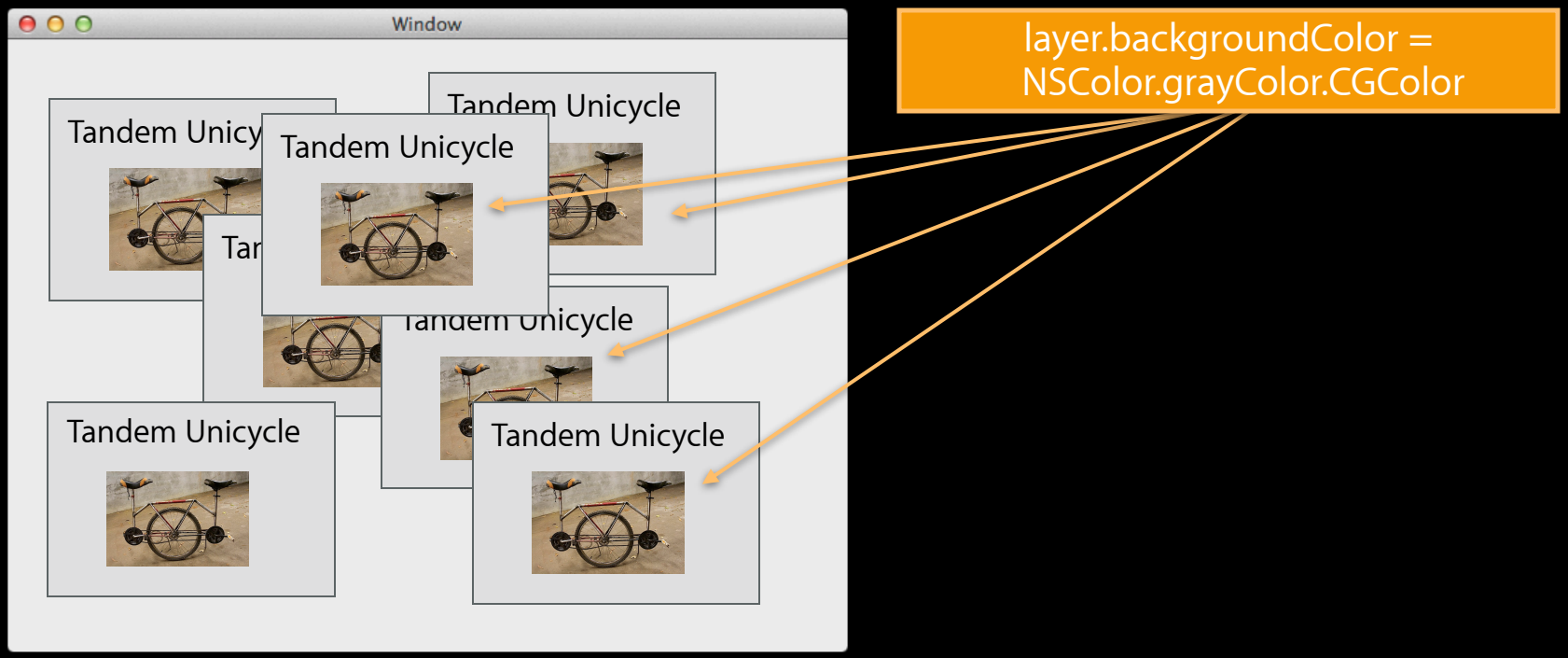
Improving Layer-Backed Memory Use

Share the same layer.contents in multiple views



Improving Layer-Backed Memory Use

Share the same layer.contents in multiple views



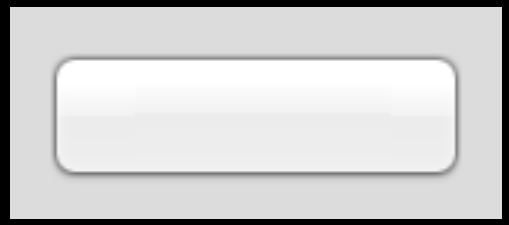
Contents Updating in -updateLayer

Nine part background image such as a button

- Goal is a button that can properly stretch when resized

Contents Updating in -updateLayer

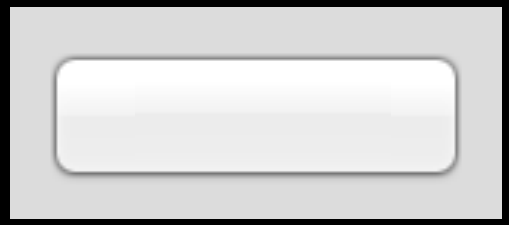
Start with a background view



- Background stretchable image 
- Image is to be shared directly as the contents among all buttons

Contents Updating in -updateLayer

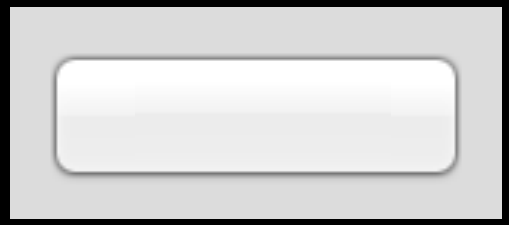
Start with a background view



```
- (void)updateLayer {
    self.layer.contents = [NSImage imageNamed: ]; // Pseudo-code
    self.layer.contentsCenter = CGRectMake(0.5, 0.5, 1e-5, 1e-5);
}
```

Contents Updating in -updateLayer

Start with a background view

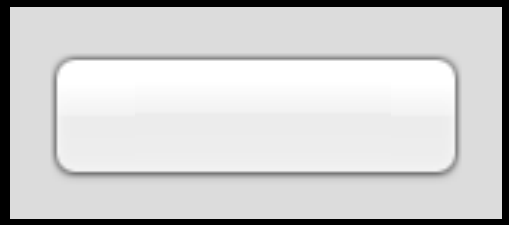


Note that this image is not the size of the view's bounds!

```
- (void)updateLayer {
    self.layer.contents = [NSImage imageNamed: ]; // Pseudo-code
    self.layer.contentsCenter = CGRectMake(0.5, 0.5, 1e-5, 1e-5);
}
```

Contents Updating in -updateLayer

Start with a background view



Note that this image is not the size of the view's bounds!

```
- (void)updateLayer {
    self.layer.contents = [NSImage imageNamed: ]; // Pseudo-code
    self.layer.contentsCenter = CGRectMake(0.5, 0.5, 1e-5, 1e-5);
}
```

This stretches the middle pixel.
See CALayer.h for more information.

Contents Updating in -updateLayer

Start with a background view


```
- (void)layout {
    if (_textField == nil) {
        _textField = [[NSTextField alloc] initWithFrame:frame];
        _textField.title = @"Button";
    } else {
       _textField.frame = // Update the location
    }
    [super layout];
}
```

Contents Updating in -updateLayer

Start with a background view

Works when using auto layout and/or using layer backing


```
- (void)layout {
    if (_textField == nil) {
        _textField = [[NSTextField alloc] initWithFrame:frame];
        _textField.title = @"Button";
    } else {
       _textField.frame = // Update the location
    }
    [super layout];
}
```

Contents Updating in -updateLayer

Dealing with multiple states (a “pressed” state)

```
- (void)updateLayer {
    if (self.pressed) {
        self.layer.contents = [NSImage imageNamed: ];
    } else {
        self.layer.contents = [NSImage imageNamed: ];
    }
    self.layer.contentsCenter = CGRectMake(0.5, 0.5, 1e-5, 1e-5);
}
```

Contents Updating in -updateLayer

Dealing with multiple states (a “pressed” state)

```
- (void)updateLayer {
    if (self.pressed) {
         self.layer.contents = [NSImage imageNamed: ];
    } else {
         self.layer.contents = [NSImage imageNamed: ];
    }
    self.layer.contentsCenter = CGRectMake(0.5, 0.5, 1e-5, 1e-5);
}

- (void)mouseDown:(NSEvent *)event {
     self.pressed = YES;
     [self setNeedsDisplay:YES];
}
```

Contents Updating in -updateLayer

Update the title correctly


```
- (void)setTitle:(NSString *)title {
    _textField.title = title;
    [self setNeedsLayout:YES];
}
```

Contents Updating in -updateLayer

Update the title correctly


```
- (void)setTitle:(NSString *)title {
    _textField.title = title;
    [self setNeedsLayout:YES];
}
```

The title is in a separate view (NSTextField) that redraws itself

Contents Updating in -updateLayer

Update the title correctly


```
- (void)setTitle:(NSString *)title {
    _textField.title = title;
    [self setNeedsLayout:YES];
}
```

The title is in a separate view (NSTextField) that redraws itself

Do NOT call setNeedsDisplay:—the background does not need to redraw!

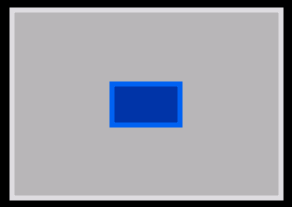
However, the textField location needs updating and re-layout

Animating

Synchronized subview animations

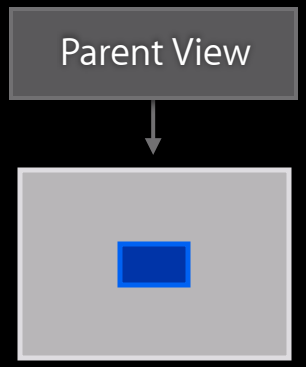
Synchronized Subview Animations

Consider this…



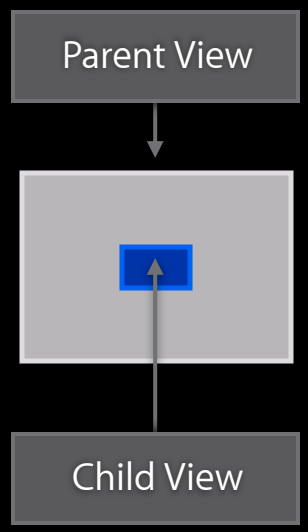
Synchronized Subview Animations

Consider this…



Synchronized Subview Animations

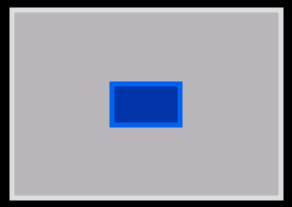
Consider this…



Synchronized Subview Animations

Consider this…

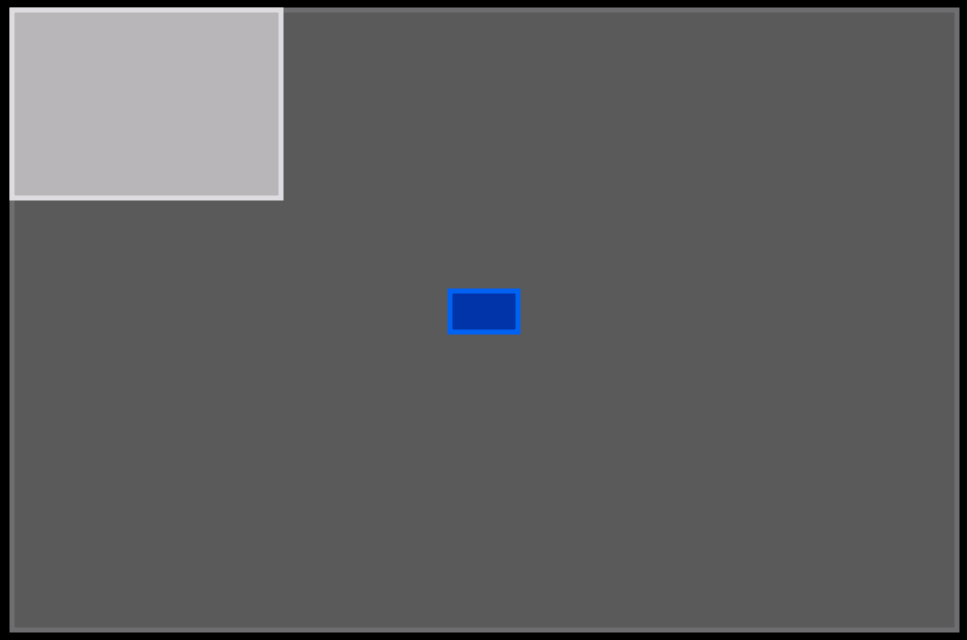
```
[[parentView animator] setFrame:NSMakeRect(0, 0, 500, 500)];
```



Synchronized Subview Animations

Consider this…

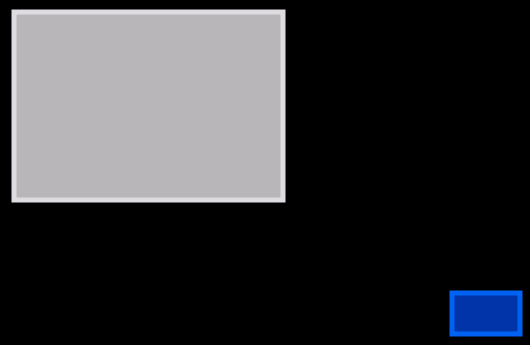
- The parent view sees itself at its final size
- The child view is moved to the final layout position



Synchronized Subview Animations

Consider this…

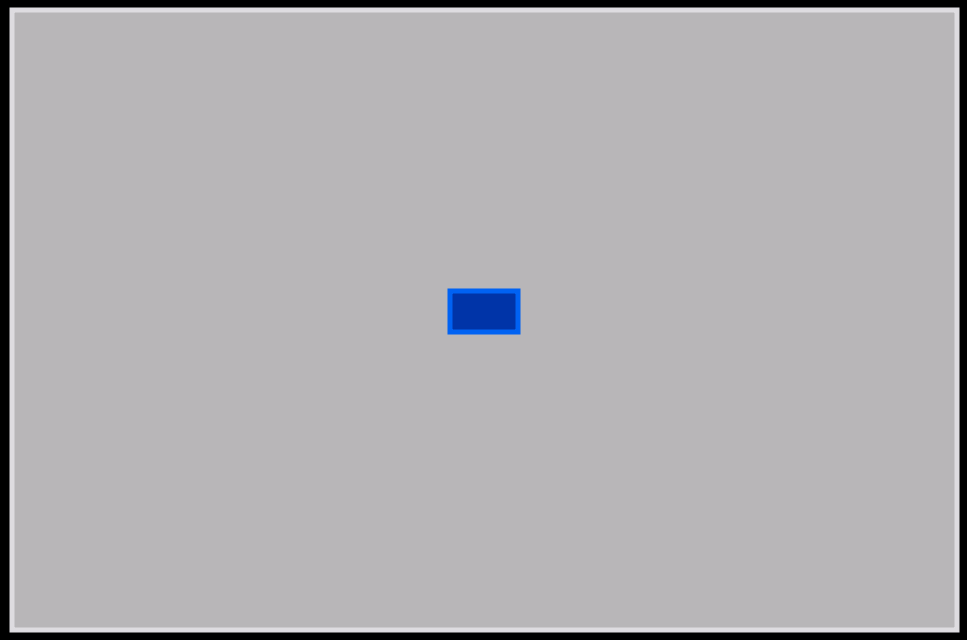
- The parent view sees itself at its final size
- The child view is moved to the final layout position



Synchronized Subview Animations

Consider this…

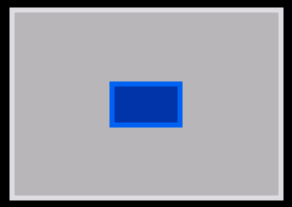
- The parent view sees itself at its final size
- The child view is moved to the final layout position



Synchronized Subview Animations

What we really want is this

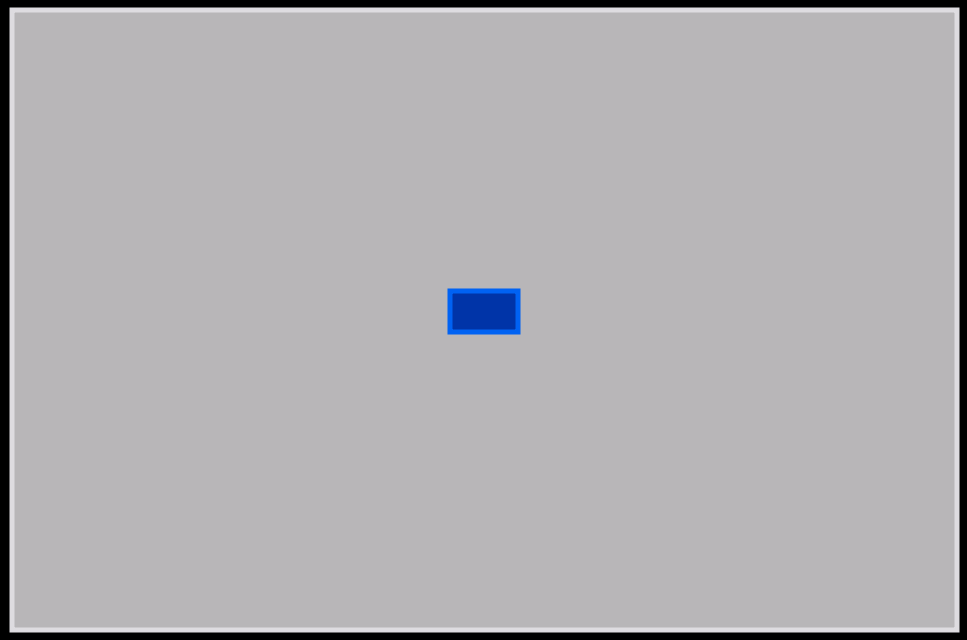
```
[[parentView animator] setFrame:NSMakeRect(0, 0, 500, 500)];
```



Synchronized Subview Animations

What we really want is this

```
[[parentView animator] setFrame:NSMakeRect(0, 0, 500, 500)];
```



Synchronized Subview Animations

- The NSAnimationContext allows grouping of animations
- Allows one to control animation properties for that group

```
@interface NSAnimationContext : NSObject { }

+ (void)beginGrouping;
+ (void)endGrouping;

+ (NSAnimationContext *)currentContext;

@property NSTimeInterval duration;
@property(retain) CAMediaTimingFunction *timingFunction;

@end
```

Synchronized Subview Animations

New to Mountain Lion allowsImplicitAnimations

- Controls implicit layer animations

```
@interface NSAnimationContext : NSObject { }
...

@property BOOL allowsImplicitAnimation NS_AVAILABLE_MAC(10_8);
...
@end
```

Synchronized Subview Animations

New to Mountain Lion allowsImplicitAnimations

- Defaults to NO
- When `allowsImplicitAnimation == YES`
  - All view/layer animatable properties will animate
  - There is no need to use the `-animator` proxy
  - Allows nesting of animations and side effect animations
    - Such as layout that is done in `setFrame:`
  - Similar to `[UIView setAnimationsEnabled:YES]`
- Only applies to layer-backed (or layer-hosted) views

Synchronized Subview Animations

allowsImplicitAnimations is automatically set

```
[[parentView animator] setFrame:NSMakeRect(0, 0, 500, 500)];
```

Synchronized Subview Animations

allowsImplicitAnimations is automatically set

```
[[parentView animator] setFrame:NSMakeRect(0, 0, 500, 500)];
```

```
context.allowsImplicitAnimation = YES;
view.frame = 500, 500;
context.allowsImplicitAnimation = NO;
```

Synchronized Subview Animations

allowsImplicitAnimations is automatically set

```
[[parentView animator] setFrame:NSMakeRect(0, 0, 500, 500)];
```

Synchronized Subview Animations

allowsImplicitAnimations is automatically set

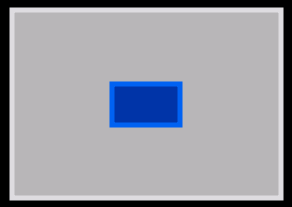
```
[[parentView animator] setFrame:NSMakeRect(0, 0, 500, 500)];
```

```
context.allowsImplicitAnimation = YES;
        view.frame = 500, 500;
        // implicitly -layout is called
        subview.frame = // centered frame
context.allowsImplicitAnimation = NO;
```

Synchronized Subview Animations

Keeping frame animations in sync

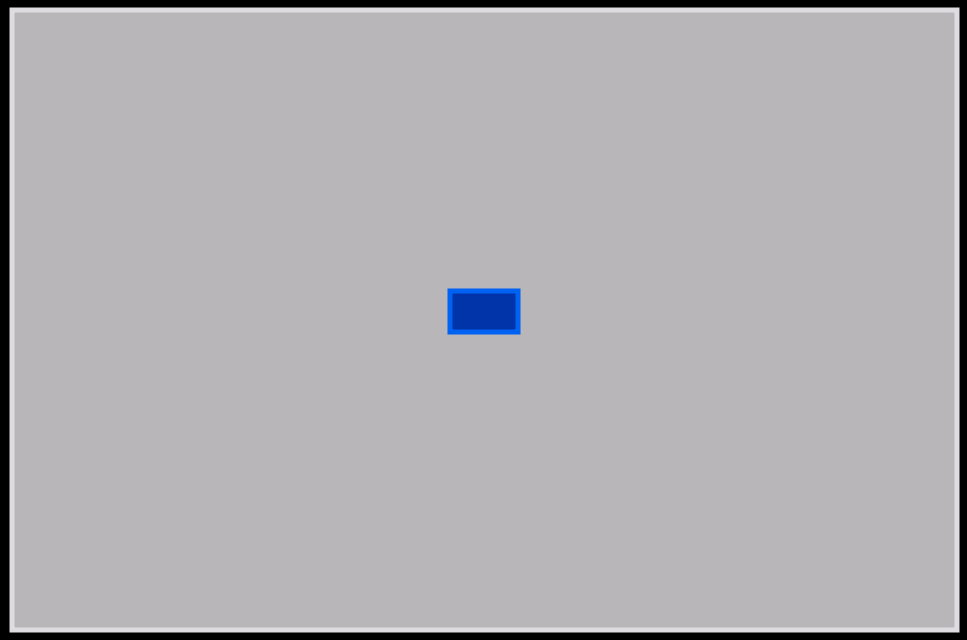
```
[[parentView animator] setFrame:NSMakeRect(0, 0, 500, 500)];
```



Synchronized Subview Animations

Keeping frame animations in sync

```
[[parentView animator] setFrame:NSMakeRect(0, 0, 500, 500)];
```



Best Practices

Text and font smoothing

LCD Font Smoothing

Also known as subpixel anti-aliasing

LCD Font Smoothing

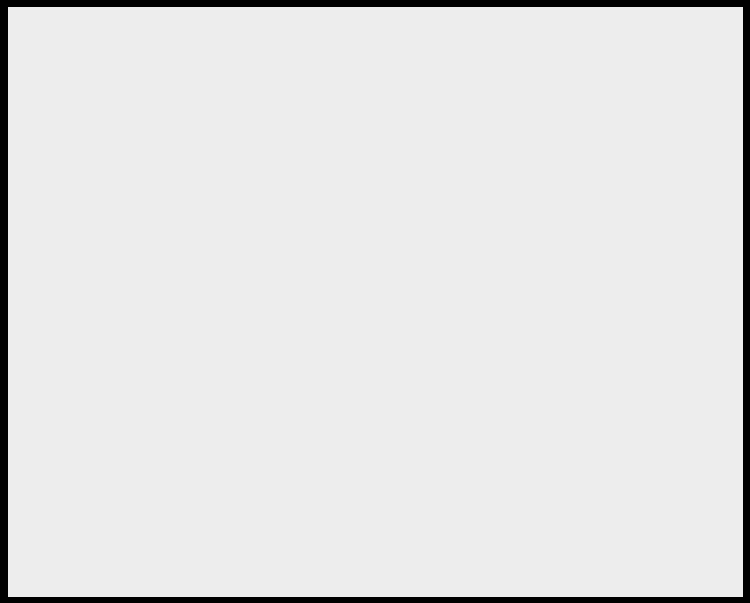
Drawing text in a layer

- LCD font smoothing requires an opaque area to composite with
- Normal AppKit drawing composites into one big image

LCD Font Smoothing

Drawing text in a layer

- LCD font smoothing requires an opaque area to composite with
- Normal AppKit drawing composites into one big image



LCD Font Smoothing

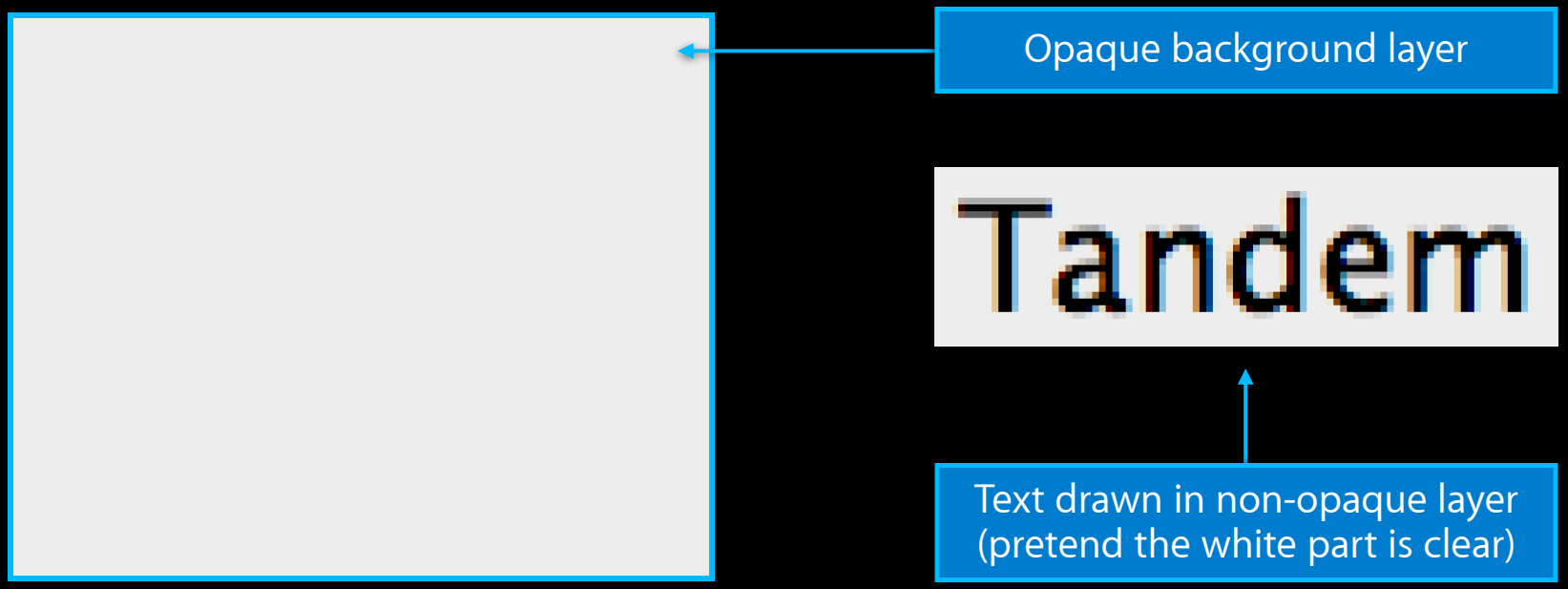
Drawing text in a layer

- LCD font smoothing requires an opaque area to composite with
- Normal AppKit drawing composites into one big image

LCD Font Smoothing

Drawing text in a layer

- Layers draw into their own mini-images which are then composited together



LCD Font Smoothing

Text drawn in non-opaque layer looks bad

- Text appears bolder than it should
  - (Opaque white background shown for clarity)

LCD Font Smoothing

Font smoothing

- Text looks better with font smoothing turned off
  - (Opaque white background shown for clarity)

LCD Font Smoothing

Font smoothing may be turned off by AppKit

- On when the view says YES to isOpaque

[NSView -isOpaque] = NO

[NSView -isOpaque] = YES

LCD Font Smoothing

Font smoothing may be turned off by AppKit

- Manually turn it on when you draw text in a layer in a non-opaque view

[NSView -isOpaque] = NO

LCD Font Smoothing

Font smoothing may be turned off by AppKit

- Manually turn it on when you draw text in a layer in a non-opaque view

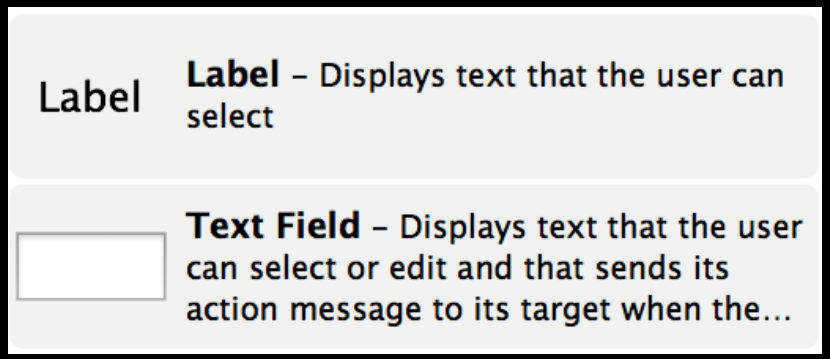
[NSView -isOpaque] = NO

```
CGContextRef ctx =
NSGraphicsContext.currentContext.graphicsPort;
CGContextSetShouldSmoothFonts(ctx, true);
[@"Tandem" drawInRect: ... ];
```

Ideally Use NSTextField!

NSTextField implements LCD font smoothing

- Works even when in a non-opaque layer
- Use NSTextField instead of manually drawing text
- Caveat: Requires at least one opaque ancestor layer



Best Practices

Focus rings

How You Typically Draw Focus Rings

Usually done in -drawRect:

```
- (void)drawRect:(NSRect)dirtyRect {
    // Normal drawing code here ...

    // Draw the focus ring
    if (self.window.firstResponder == self) {
        NSSetFocusRingStyle(NSFocusRingOnly);
        NSRectFill(self.bounds);
    }
}
```

- Note the focus ring draws slightly outside the view's bounds!

Focus Rings and Layers

Focus rings look wrong!

- The focus ring is captured as part of the layer's contents
- The layer clips to its bounds

Focus Rings on Lion

For regular views or layer-backed views

- Use the Lion API

```
- (NSRect)focusRingMaskBounds;
- (void)drawFocusRingMask;
- (void)noteFocusRingMaskChanged;
```

Focus Rings

focusRingMaskBounds

- This is for the enclosing shape of the focus ring
- Only called if your view has focus (is the firstResponder)

```
- (NSRect)focusRingMaskBounds {
    return self.bounds;
}
```

- Return an empty rect to not have a focus ring

Focus Rings

drawFocusRingMask

- Draw your content; the focus ring will automatically appear around it

```
- (NSRect)drawFocusRingMask {
    NSBezierPath *strangeShape = ...;
    [strangeShape fill]; // Focus ring appears around this shape
}
```

Focus Rings

noteFocusRingMaskChanged

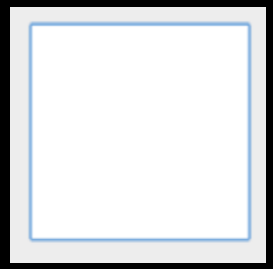
- Tell AppKit when your focus ring has changed
- To invalidate the focus ring shape, call

```
[self noteFocusRingMaskChanged]
```

Focus Rings

How the API works when layer-backed

- The focus ring is drawn into a separate layer



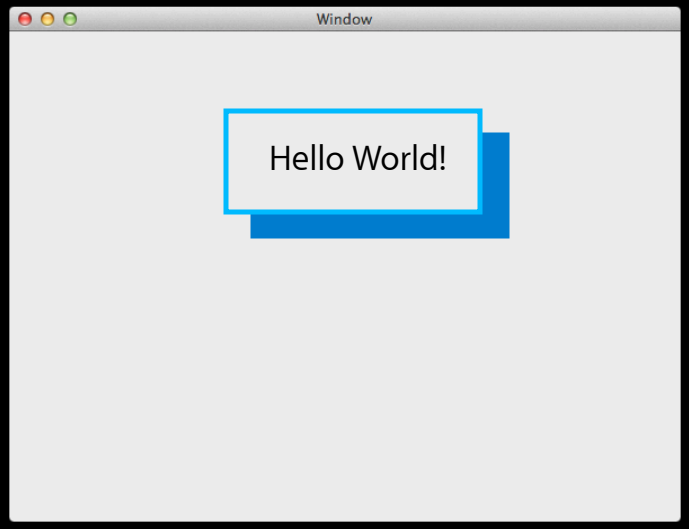
- AppKit adds the layer above the focused view

More Details and Tips

Layer-Backed and Layer-Hosted

CALayer properties that AppKit always manages

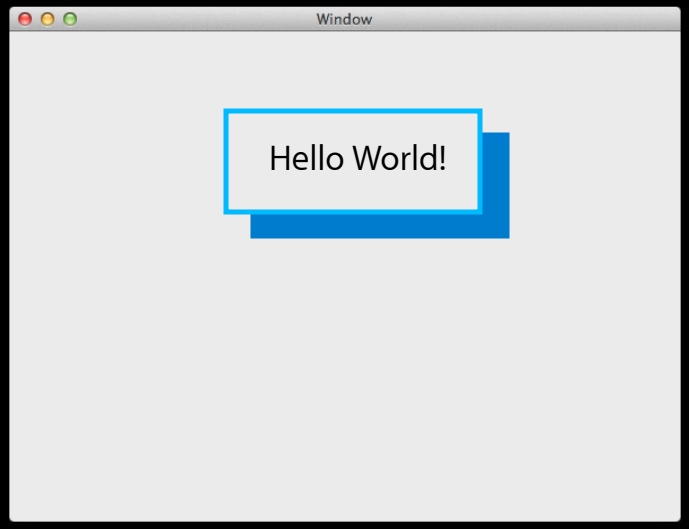
```
geometryFlipped, bounds, frame (implied), position, anchorPoint, transform,
shadow[Color, Offset, Opacity, Radius], hidden, filters, and
compositingFilter
```



Layer-Backed and Layer-Hosted

CALayer properties that AppKit always manages

`geometryFlipped`, `bounds`, `frame` (implied), `position`, `anchorPoint`, `transform`, `shadow[Color, Offset, Opacity, Radius]`, `hidden`, `filters`, and `compositingFilter`



Layers and Coordinates

Prior to Mountain Lion

- Layer coordinates where not equal to view coordinates
- AppKit would vary the anchorPoint
  - `isFlipped = YES`, then the `anchorPoint` is (0, 1)
  - `isFlipped = NO`, then the `anchorPoint` is (0, 0)
- You previously used `–convertPointToLayer:` and `–convertPointFromLayer:`
  - `–convertPointToLayer:` flips the y location based on `–isFlipped`
- Changing the `geometryFlipped` property was still not recommended

Layers and Coordinates

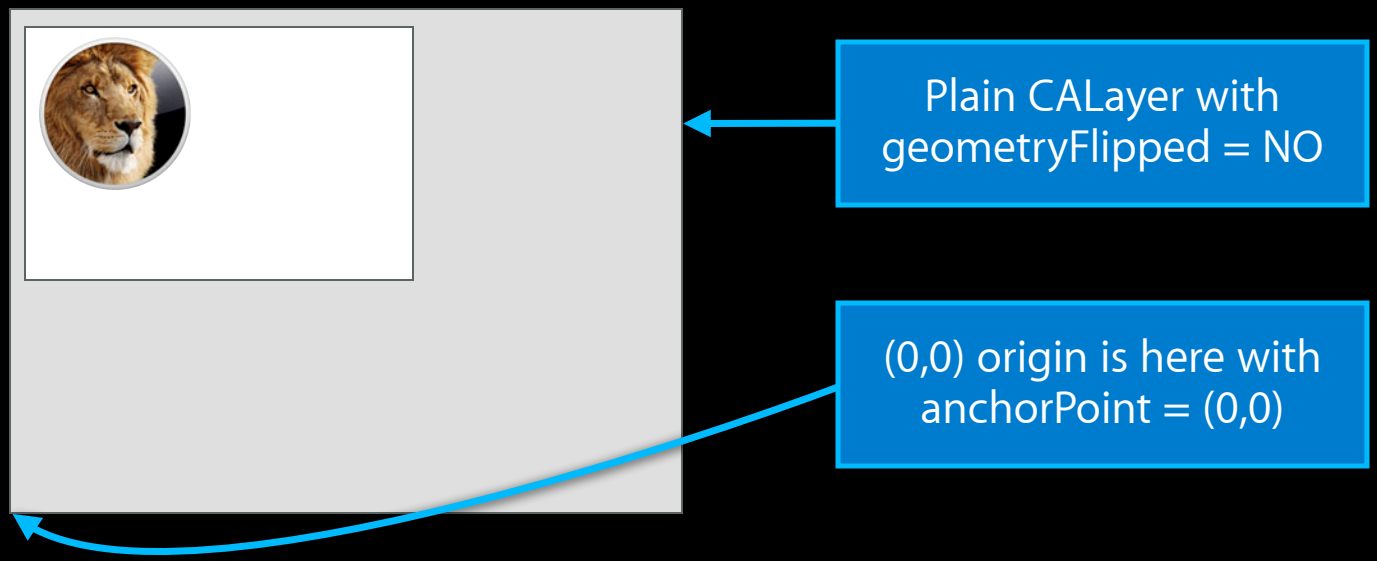
On Mountain Lion and above

- Layer coordinates now equal view coordinates!
- AppKit does not change the `anchorPoint`
  - The `anchorPoint` is always (0, 0)
- AppKit does this by managing `geometryFlipped`
- You do not have to do anything for this!

Layers and Coordinates

Understanding CALayer's geometryFlipped

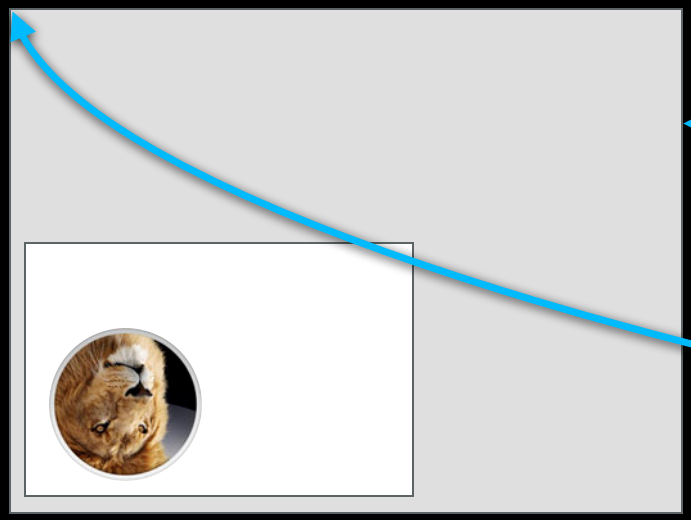
`geometryFlipped` affects the parent layer and all child sublayers



Layers and Coordinates

Understanding CALayer's geometryFlipped

`geometryFlipped` affects the parent layer and all child sublayers



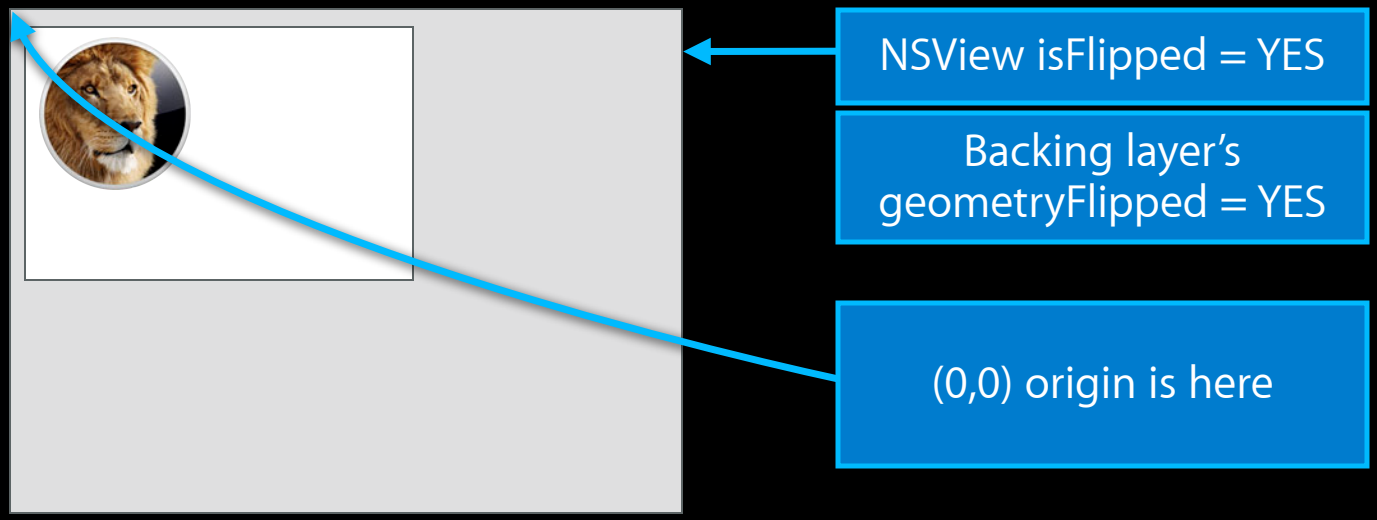
Same CALayer with geometryFlipped = YES

(0,0) origin is here with anchorPoint = (0,0)

Layers and Coordinates

On Mountain Lion

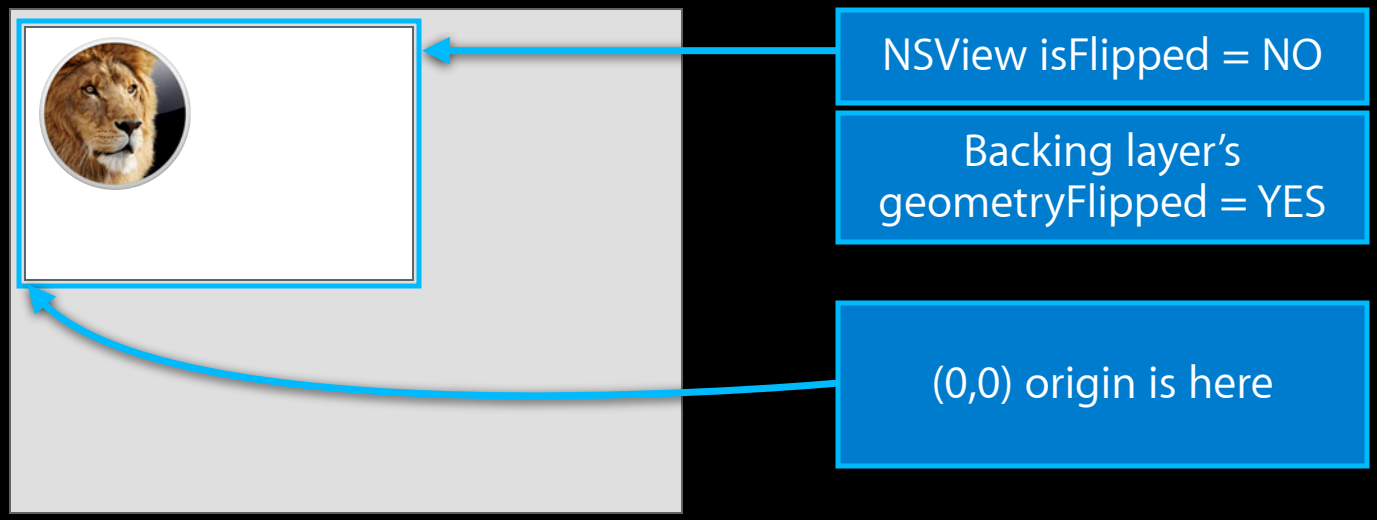
- Layer coordinates equal view coordinates
- This is accomplished by AppKit managing `-[CALayer geometryFlipped]`



Layers and Coordinates

On Mountain Lion

- Layer coordinates equal view coordinates
- This is accomplished by AppKit managing `-[CALayer geometryFlipped]`



Supporting High Resolution

Use a layer-backed NSView and not a direct CALayer

- Avoid using CALayers directly (layer-hosting)
  - No mouse or tracking support
  - No events or responder chain support
  - No automatic High Resolution support
- If you must use a direct CALayer, use the delegate method

```
- (BOOL)layer:(CALayer *)layer
    shouldInheritContentsScale:(CGFloat)newScale
                    fromWindow:(NSWindow *)window
```

You can still use -drawRect:

(Instead of -updateLayer)

- Sometimes it isn't possible to refactor your views to use subviews
- Animating views works fine if the frame size never changes
  - Such as custom buttons or small custom views
- Implementing `drawRect:` and `updateLayer` allows you to support 10.7 and 10.8 when layer-backed

Printing and -drawRect:

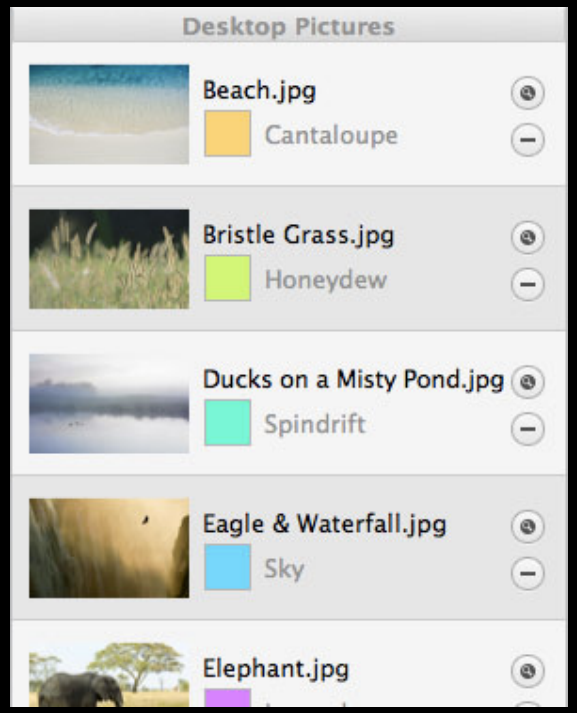
-drawRect: is still used for printing

- Most controls and user interface views don't need to print
- Standard AppKit controls still support `–drawRect:` and will always print correctly
  - NSTextView
  - NSImageView
  - …and others

Legacy Cell Based Controls

Layer-backing does not use layers for each cell

- Cell based NSTableView
  - Cells can not be layer-backed
- Use a View-Based TableView
  - See last year's talk
  - Each view can be layer-backed



Legacy Cell Based Controls

Layer-backing does not use layers for each cell

- NSMatrix
  - Use regular views instead
  - Radio button groups interact together
  - Consider using NSCollectionView
- NSForm
  - Use NSTextFields

Subclassing Standard AppKit Controls

- Most AppKit controls implement `–wantsUpdateLayer` and return `YES`
  - They add subviews and update layer contents with `–updateLayer:`
- However, `–wantsUpdateLayer` returns NO if someone subclasses and overrides any AppKit drawing methods
  - NSView's `–drawRect:`, NSCell's `drawWithFrame:inView:`, `drawInteriorWithFrameInView:`, `drawTitleWithFrame:inView:`, etc.

Subclassing Standard AppKit Controls

- To extend an existing control
  - Implement `–wantsUpdateLayer` and return YES
  - Add extra subviews where necessary in `–layout`
  - Use the existing position methods to customize where AppKit added subviews are located
    - On NSCell: `titleRectForBounds:`, `imageRectForBounds:`, etc.

Conclusion

Take Away Slide

Important stuff

- Use `layerContentsRedrawPolicy = NSViewLayerContentsRedrawOnSetNeedsDisplay`
- Use subviews whenever possible
- When possible use `–wantsUpdateLayer` and `–updateLayer`
  - Directly set the `layer.contents` and `layer.contentsCenter`
- Use NSTextField for adding text to get proper font smoothing
- Use the `–animator` proxy to start animations
- Use the Lion focus ring drawing API

More Information

Jake Behrens
UI Frameworks Evangelist
behrens@apple.com

Documentation
Core Animation Programming Guide
http://developer.apple.com/

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

Related Sessions

Advanced Tips and Tricks for High Resolution on OS X	Mission Friday 10:15AM

Labs

Cocoa & Layer-Backed Views on OS X Lab	Essentials Lab B Wednesday 2:00PM
High Resolution on OS X Lab	Essentials Lab B Wednesday 11:30AM
OS X Gestures and Cocoa Lab	Essentials Lab B Thursday 2:00PM
Cocoa and XPC Lab	Essentials Lab A Friday 10:15AM

WWDC2012

The last 3 slides after the logo are intentionally left blank for all presentations.

The last 3 slides after the logo are intentionally left blank for all presentations.

The last 3 slides after the logo are intentionally left blank for all presentations.