CS193E
Lecture 20

Cocoa Animation

Announcements

- Final Projects
    - Due Wed
- Project Demos
    - Thursday, March 20, 2008
    - 3:30 - 6:30 PM
    - Skilling 193

Development Resources

- Xcode Documentation
- Developer Connection http://developer.apple.com
  - Free Online Membership
  - Student Membership Available
- WWDC
  - Typically a student scholarship program

Architecture

- Core Animation engine
- Layer based

Ease of use

Power and Simplicity

NSViews

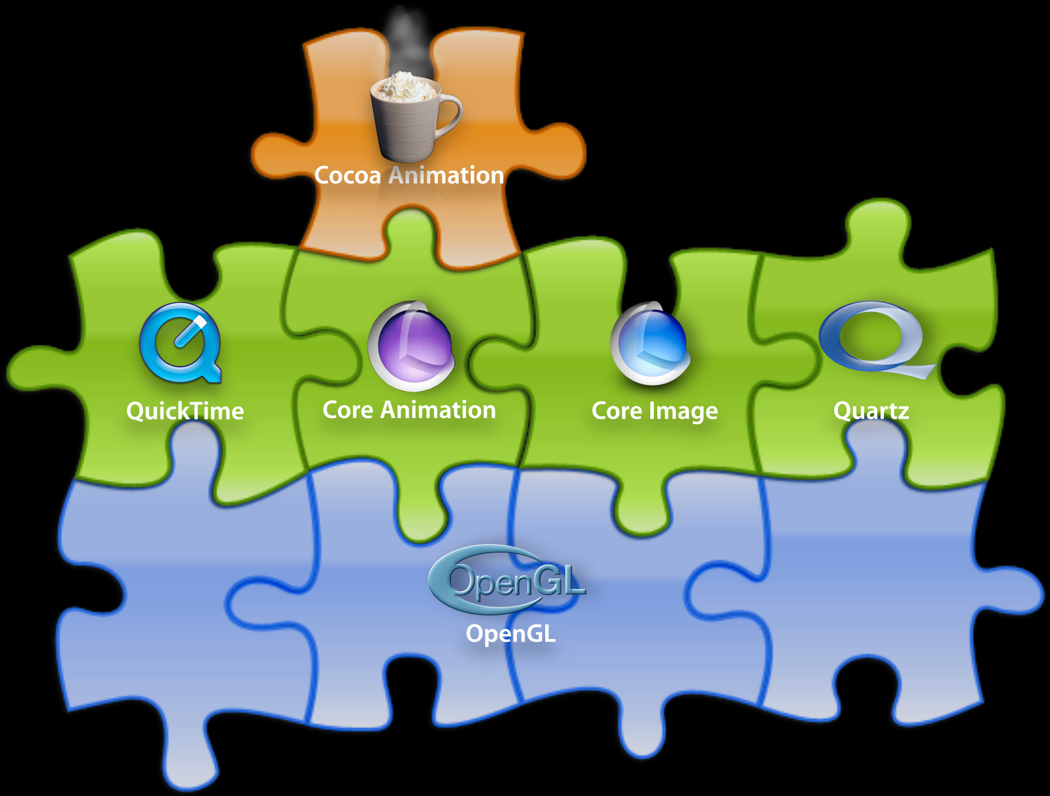
Graphics unification

Layers

Performance

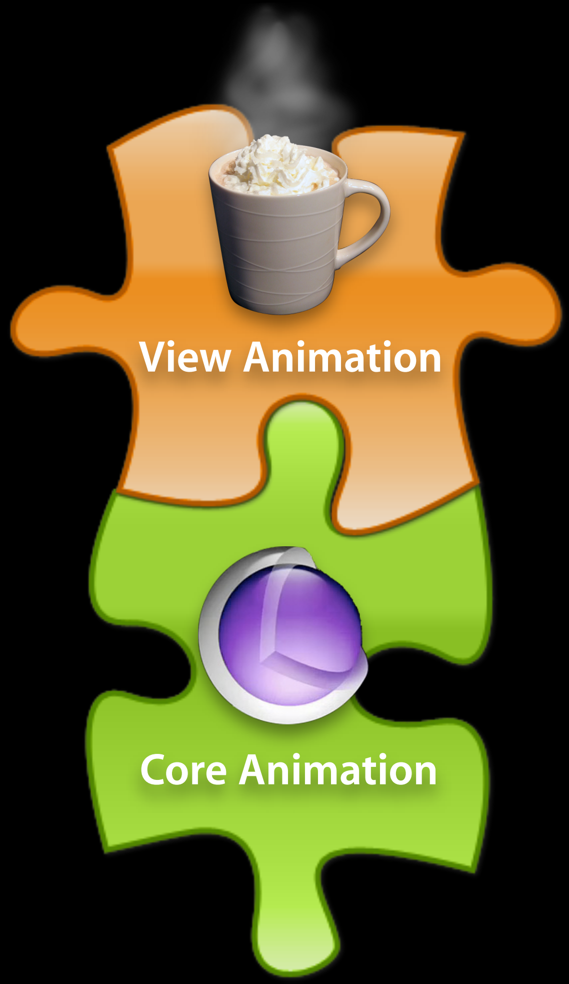
Hardware acceleration

Multi threaded



Core Animation

- Which Layer should I use?
- Start at the NSView level
        - Easy to implement
        - Provides built-in behaviors
        - Future-proofs your UI
    - Call underlying layer based effects when you need to
    - Replace a placeholder view with your own Layer tree



Introduction

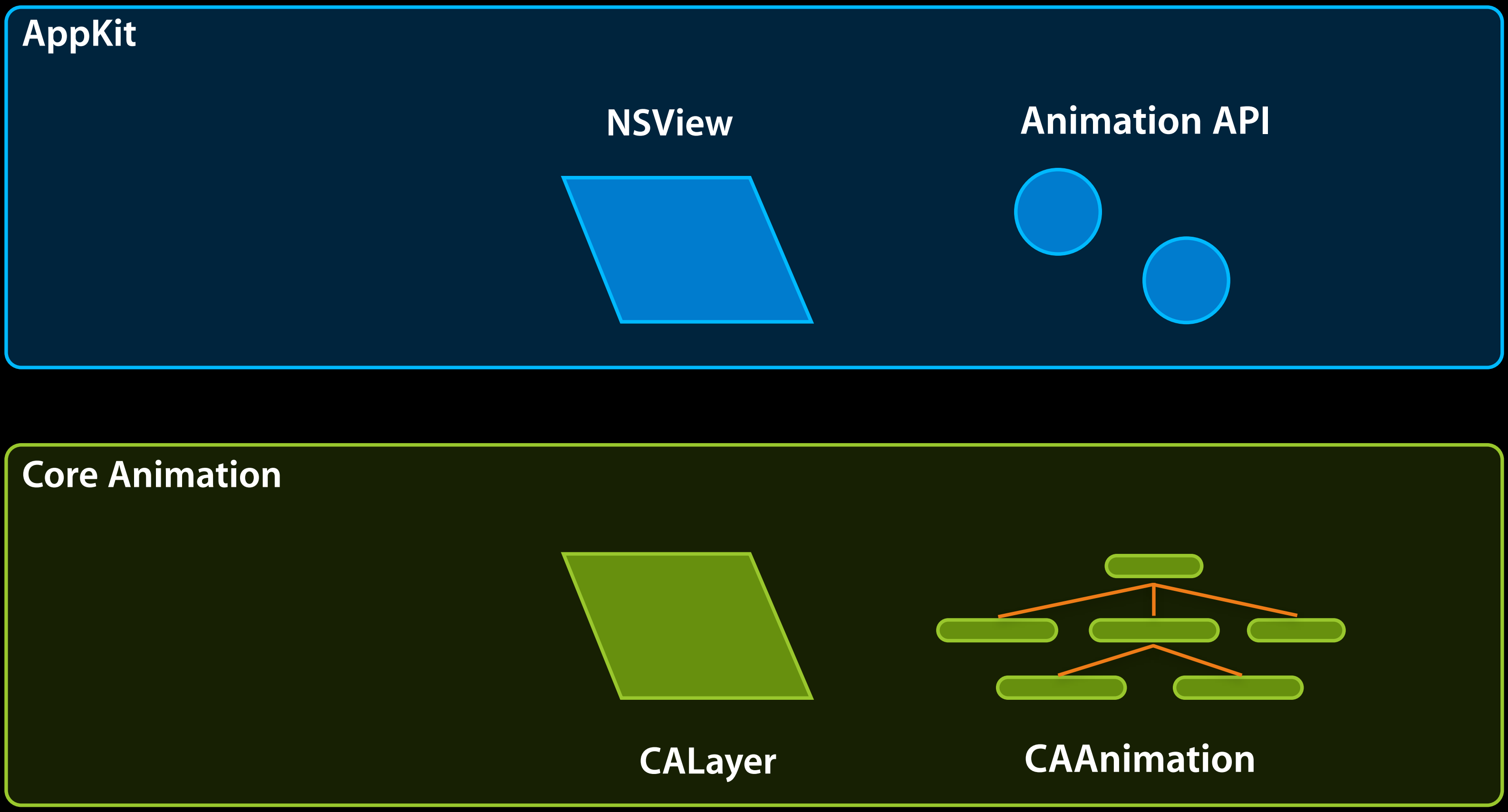
- User interfaces are becoming more fluid, cinematic
- NSWindow and NSView are the core classes used to build Cocoa user interfaces
- AppKit has added simple, flexible API for animation
- Core Animation provides a powerful foundation for compositing and animation
- AppKit harnesses Core Animation to extend its functionality to Views

Demo

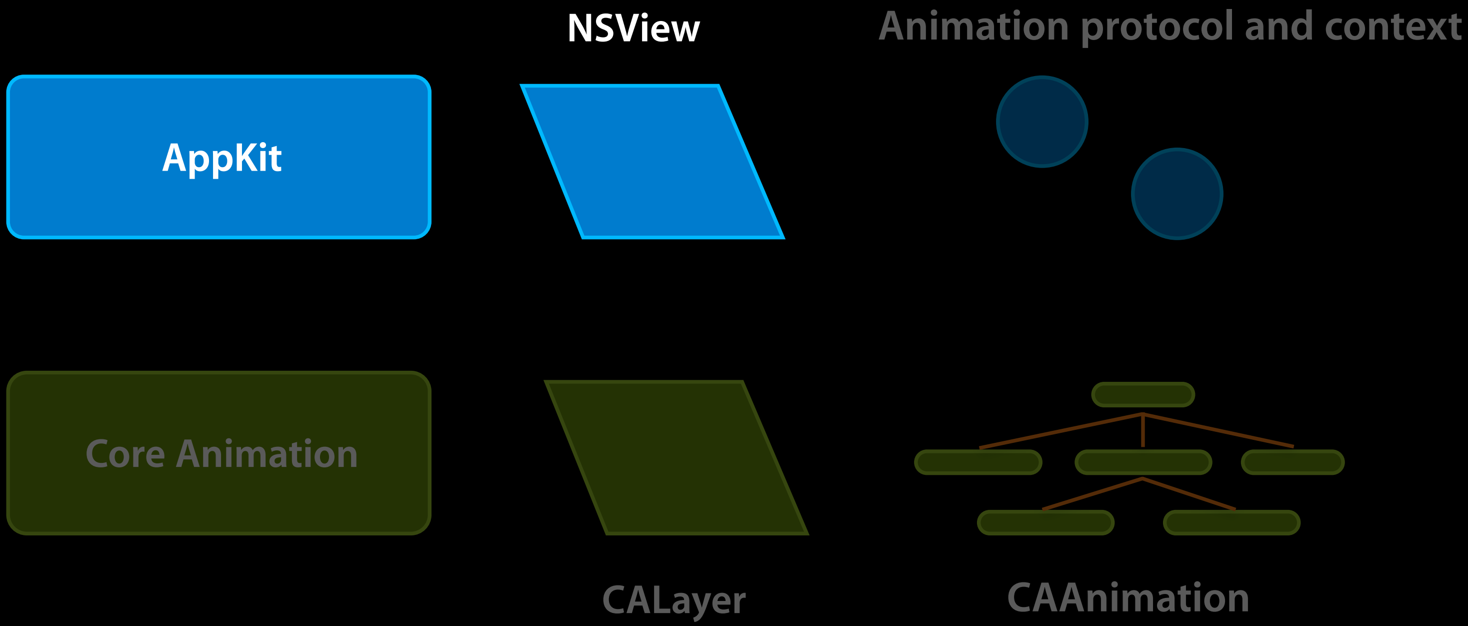
Cocoa Shuffle

James Dempsey
Application Frameworks Engineer

Technology Framework

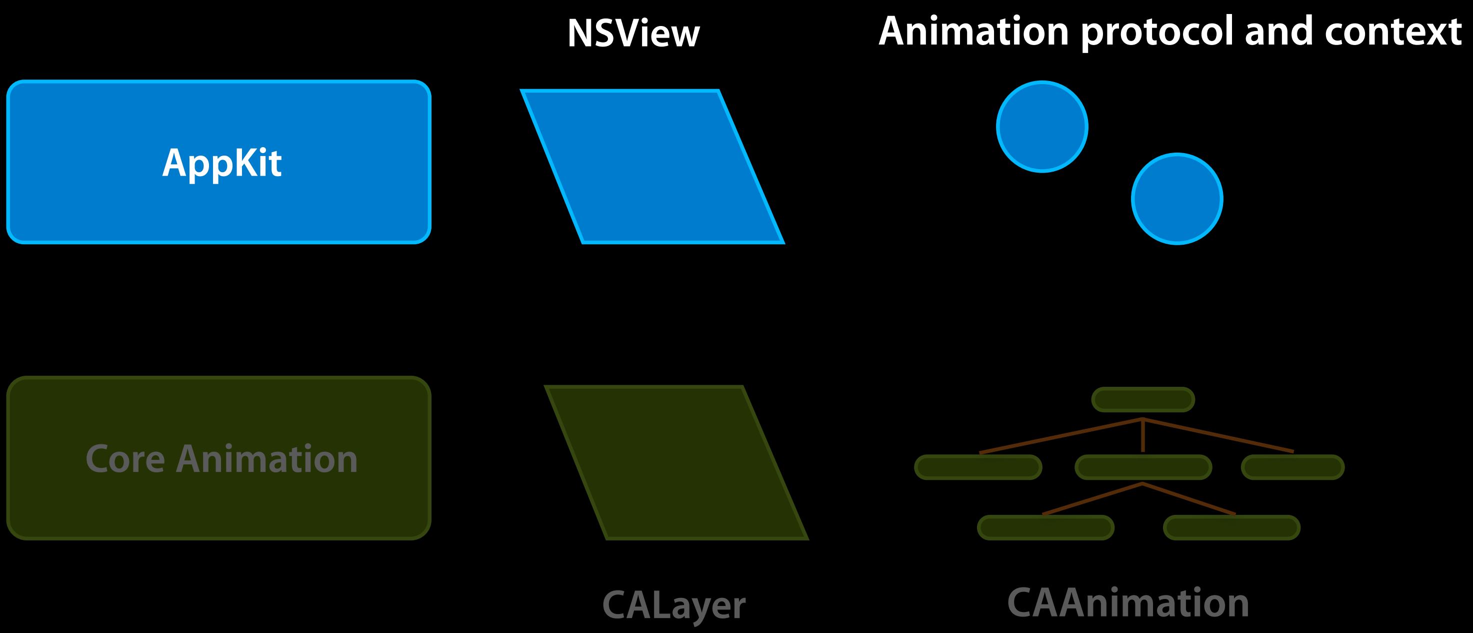


Technology Framework



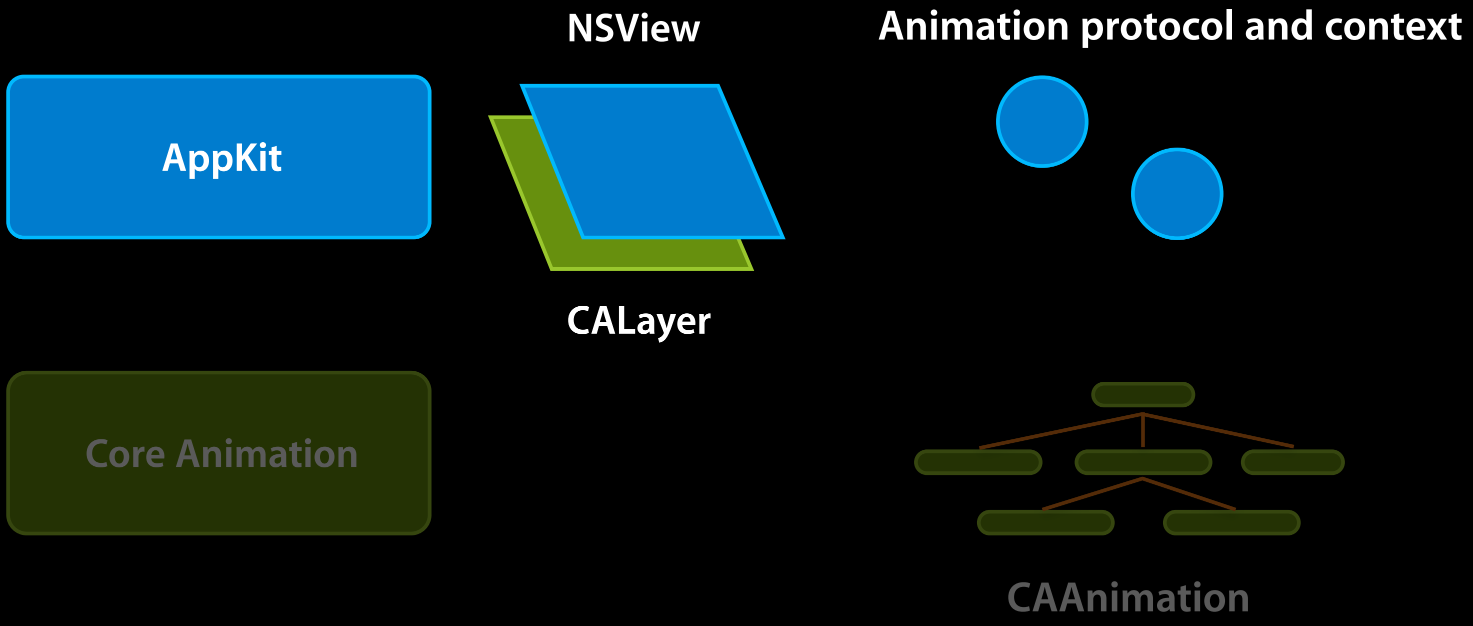
Technology Framework

Scenario 1: Basic, common animations



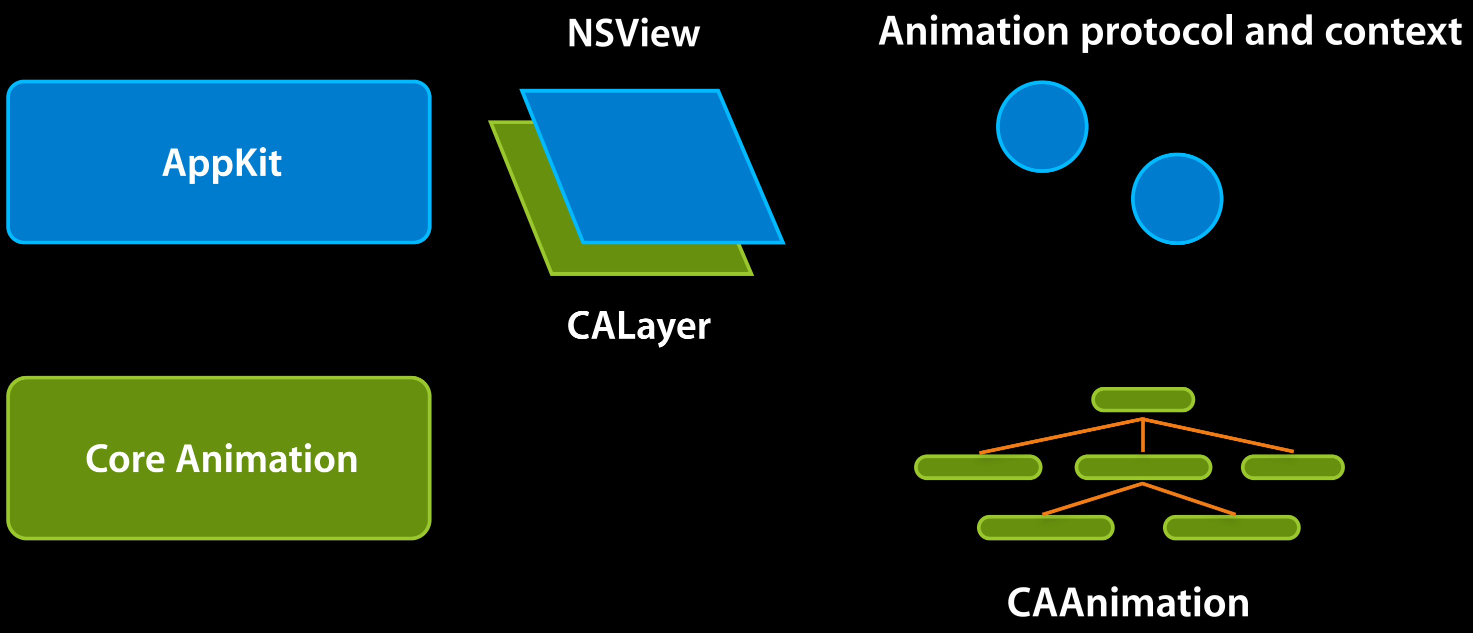
Technology Framework

Scenario 2: Layer-backed Views—cached content, new visual effects, same animation goodness



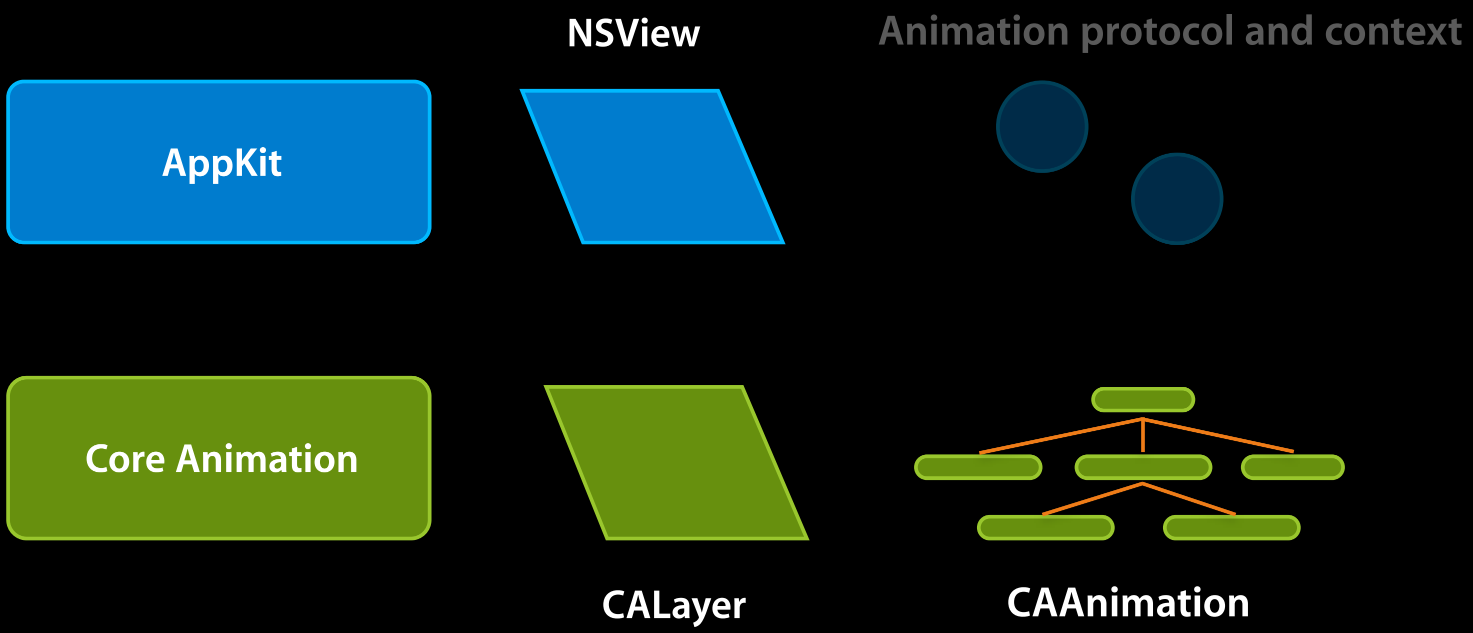
Technology Framework

Scenario 3: Layer-backed Views and Custom Animation



Technology Framework

Scenario 4: CALayer hierarchy hosted in a single NSView



Building Blocks

- Fundamentals
  - The animator and animation context
  - Layer-backed views
- Custom Animations
  - Defining animations
  - Setting animations for properties
  - Creating your own animatable properties

Building Blocks

- Fundamentals
  - The animator and animation context
  - Layer-backed views
- Custom Animations
  - Defining animations
  - Setting animations for properties
  - Creating your own animatable properties

The Fundamentals

Basic Animation

- Animation in a nutshell
- Objects have properties
- Animation is simply varying a property over time
- In Cocoa, methods already set new property values

```
[view setFrame:rect];
```

- Need a mechanism for setting a new property value and triggering an animation to the new value

Introducing the Animator

- A proxy object for initiating animations
  - Views and windows have animators
- Use value-set messages to initiate animations

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

- Use the proxy like the object you got it from
  - Send it messages, including value-set messages
  - Pass it to code that expects an object of the original type (e.g. an NSView)

Default Animations

- Get a lot done with half a line of code
- Default animations provided for all animatable properties
- Default duration for animations is 0.25 seconds
- All animations in a single event loop triggered simultaneously
    - No additional code to synchronize animations

NSAnimationContext

- New in Leopard
  - Each thread has a stack of these (like NSGraphicsContext)
  - Groups multiple animations to occur at once
  - Holds the default duration for animator-initiated animations
  - Typical usage:

```
[NSAnimationContext beginGrouping];
[[NSAnimationContext currentContext] setDuration:0.5];

/* Talk to some animator proxies; start some animations. */
 [[imageView animator] setFrameOrigin: newImageLocation];
 [[albumView animator] setFrameOrigin: newAlbumLocation];

[NSAnimationContext endGrouping];
```

Animator and Animation Context

- General additions to Cocoa for animating
- Can animate basic visual properties regardless of how content is rendered
- NSView
    - frame, frameOrigin, frameSize, frameRotation
    - bounds, boundsOrigin, boundsSize
- NSWindow
    - alphaValue
    - frame

Core Animation Layers

- At the core of Core Animation
- Analogous to Views
- Per-Layer Content Buffering
- CoreImage Filters and Transitions
- Shadows, Masking
- Combine media types
    - Quartz
    - OpenGL
    - QuickTime
    - Quartz Composer

Layer-Backed Views

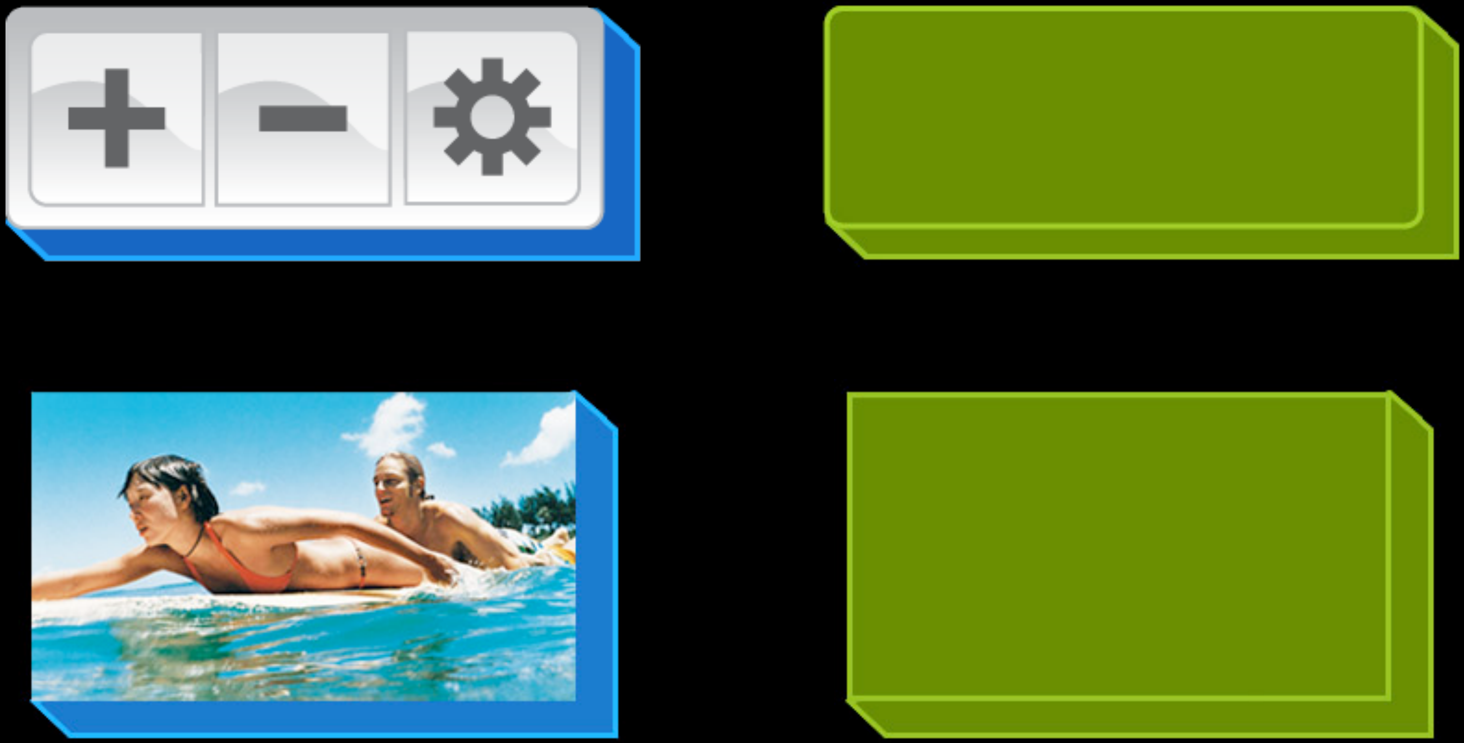
- Using a Core Animation layer to draw a view
- Per-Layer Content Buffering
- Asynchronous Animation
- Transition Animations
- CoreImage Effects, Shadows, Masking
- Combine OpenGL content

Animation API

Scenario 1: Basic, Common Animations

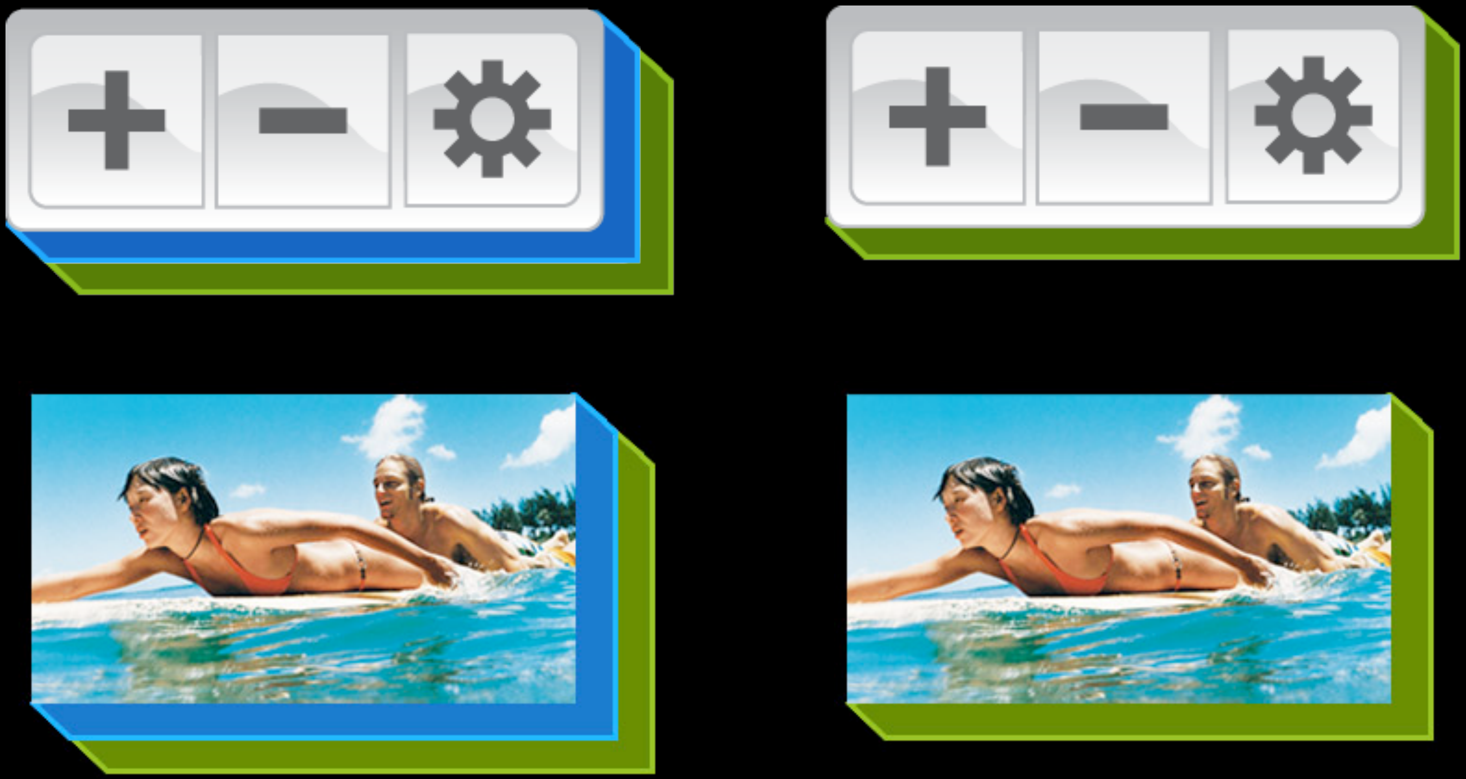
Layer-Backed Views

Scenario 2: cached content, new visual effects



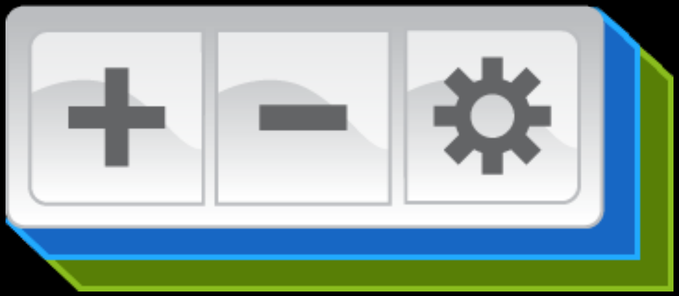
Layer-Backed Views

Scenario 2: cached content, new visual effects



Layer-Backed Views

Scenario 2: cached content, new visual effects



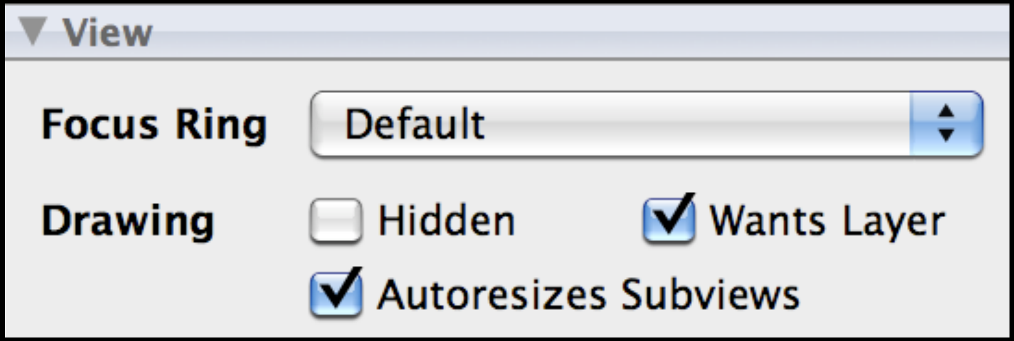
Core Image filters

New visual properties

Etc.

Layer-Backed View Rendering

```
[view setWantsLayer:YES];
```



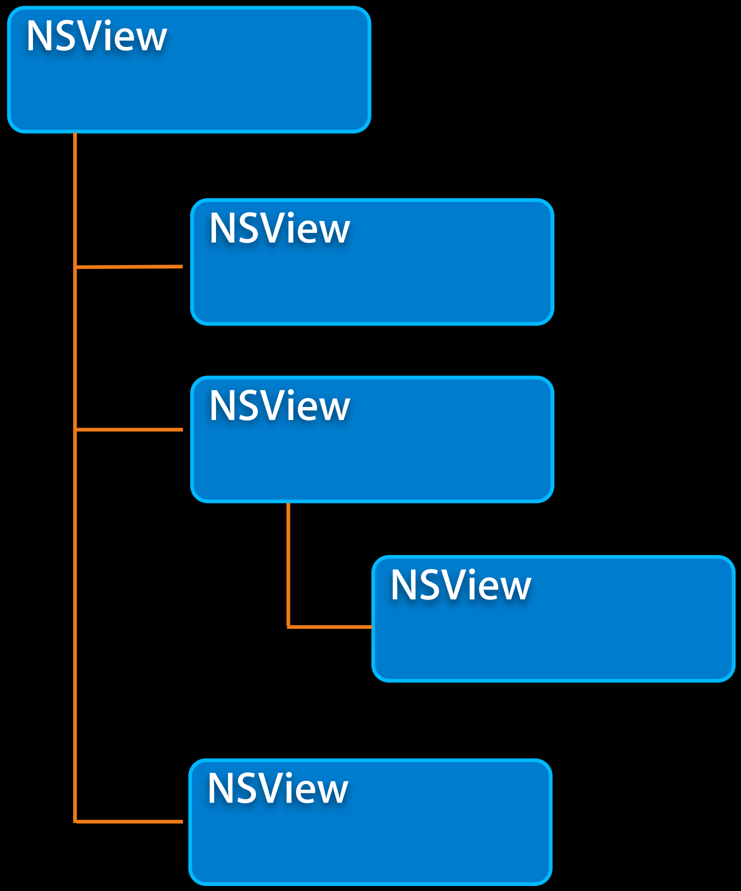
Focus Ring: Default

Drawing: [ ] Hidden [✓] Wants Layer [✓] Autoresizes Subviews

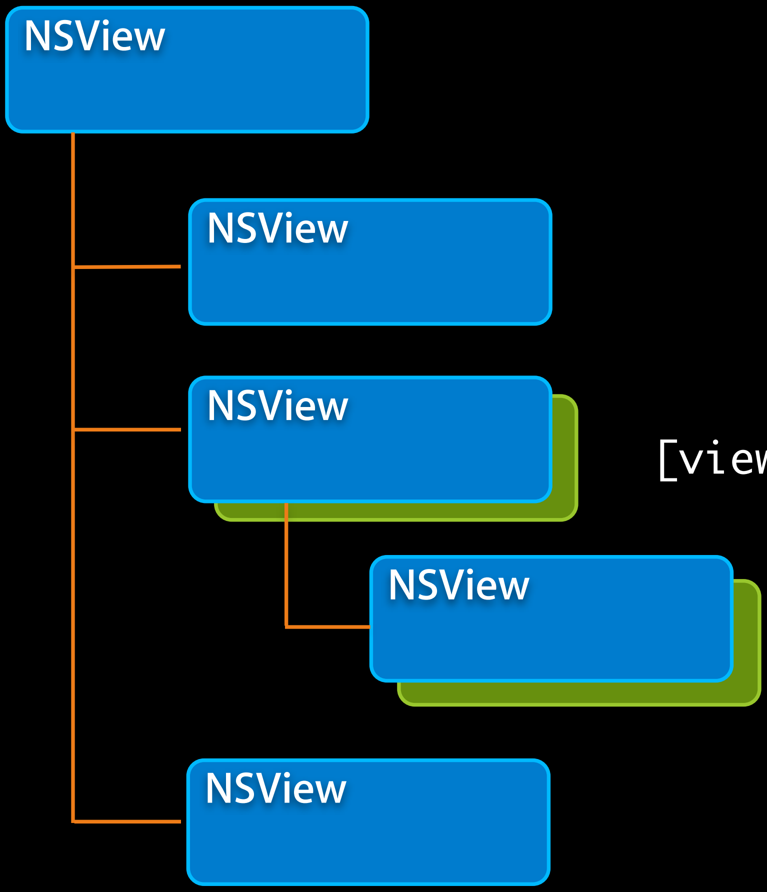
Layer-Backed View Rendering

- Little flag, big effects
  - AppKit mirrors the view subtree into a layer tree
  - Views draw into their layers, via -drawRect:
  - “setNeedsDisplay” for a view carries over to its layer
  - View property changes map to layer properties
  - AppKit implements animation of non-layer properties
  - Any “wantsLayer” setting further down in the subtree is ignored

Applied to View Hierarchy

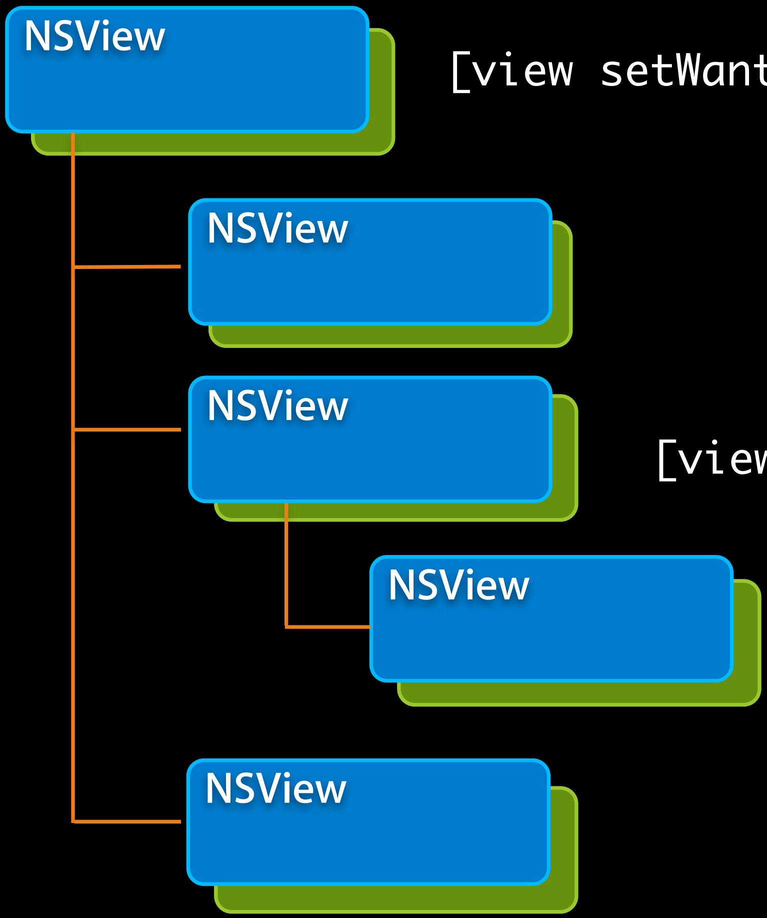


Applied to View Hierarchy



```
[view setWantsLayer:YES];
```

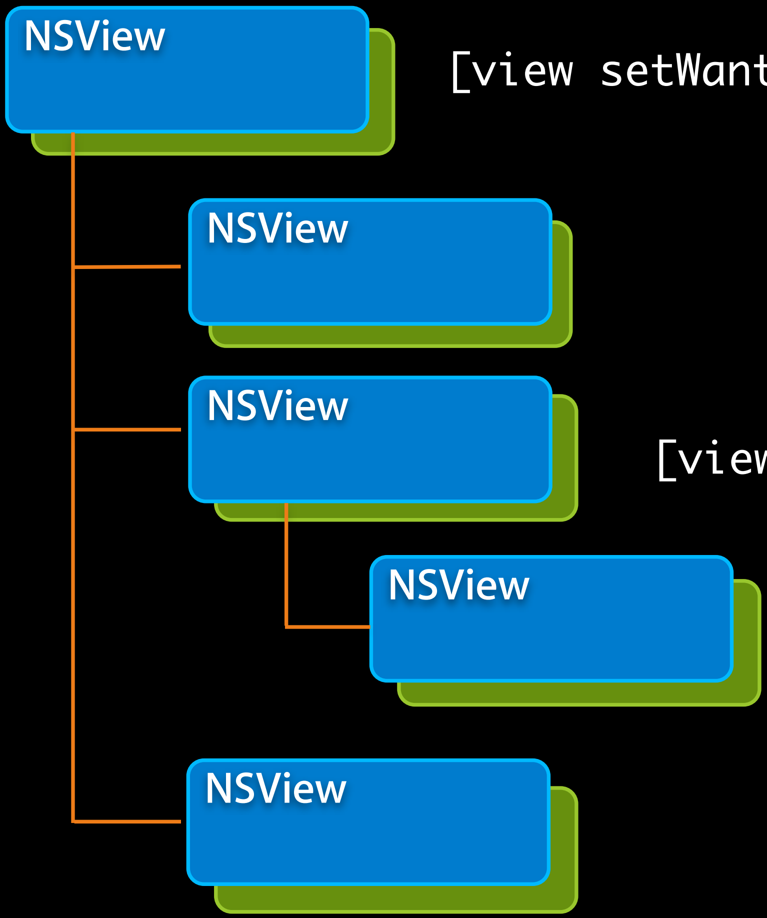
Applied to View Hierarchy



```
[view setWantsLayer:YES];
```

```
[view setWantsLayer:YES];
```

Applied to View Hierarchy

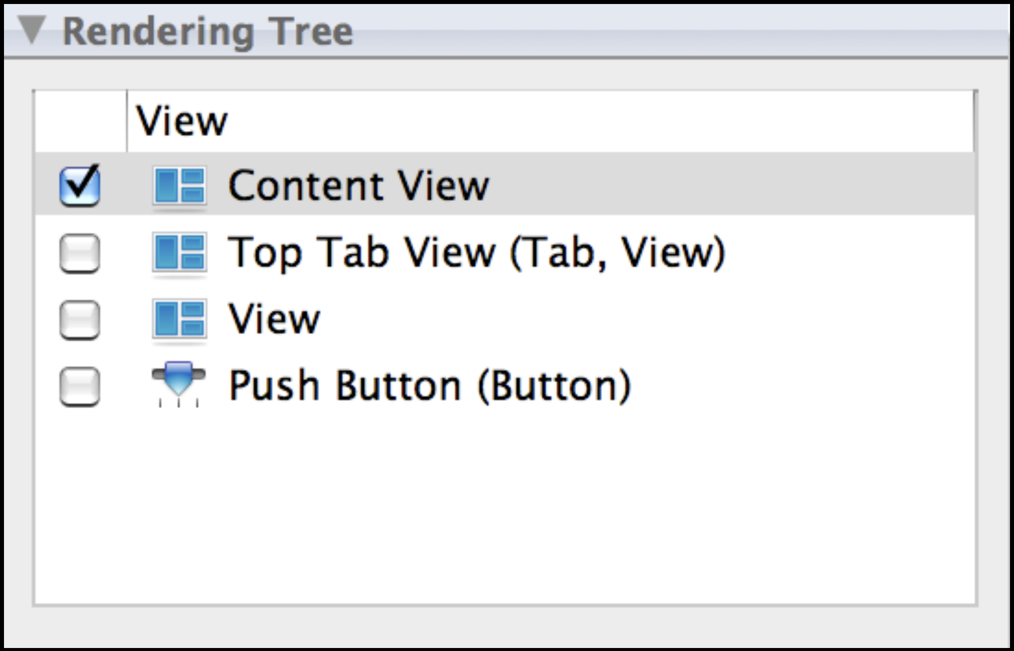


```
[view setWantsLayer:YES];
```

```
[view setWantsLayer:NO];
```

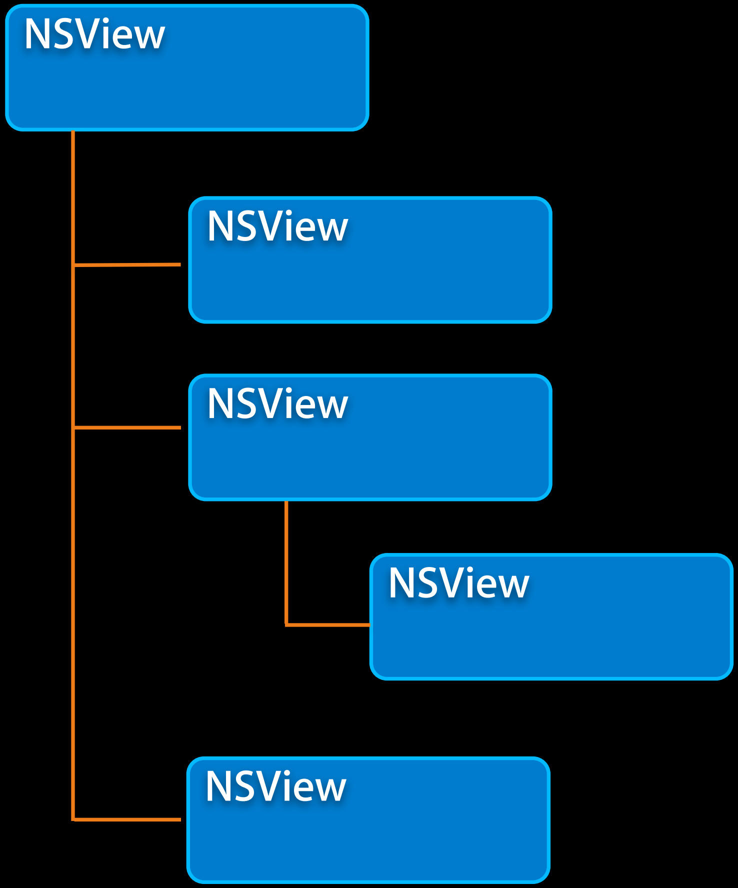
Interface Builder Inspector

Shows who wants layer in view hierarchy

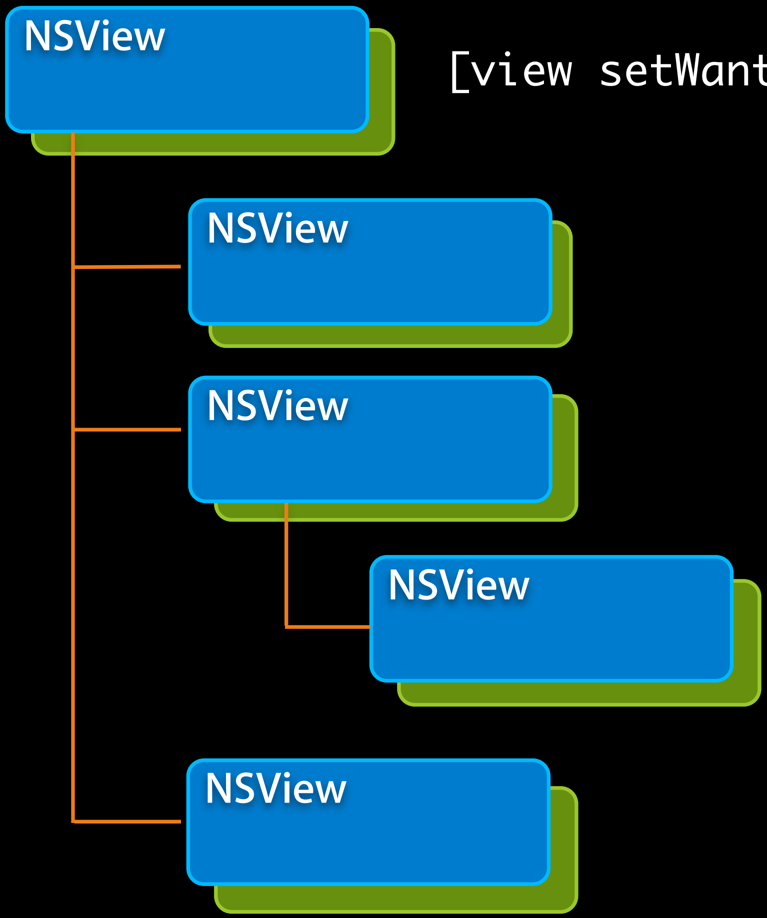


	View
✓	Content View
☐	Top Tab View (Tab, View)
☐	View
☐	Push Button (Button)

Toggle Layer-Backed Mode as Needed

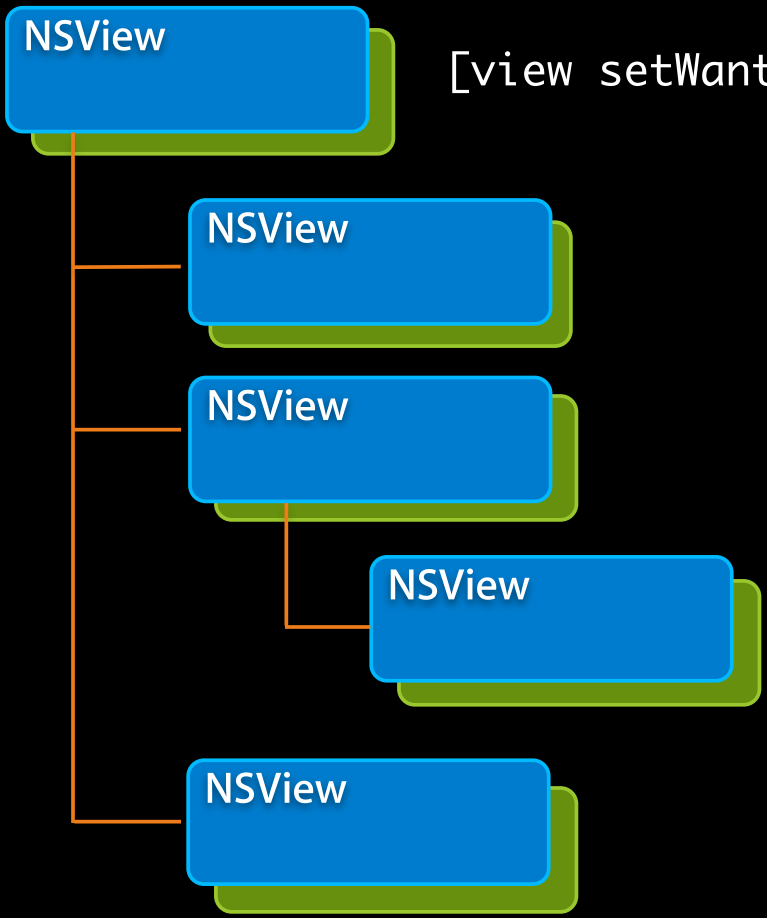


Toggle Layer-Backed Mode as Needed



```
[view setWantsLayer:YES];
```

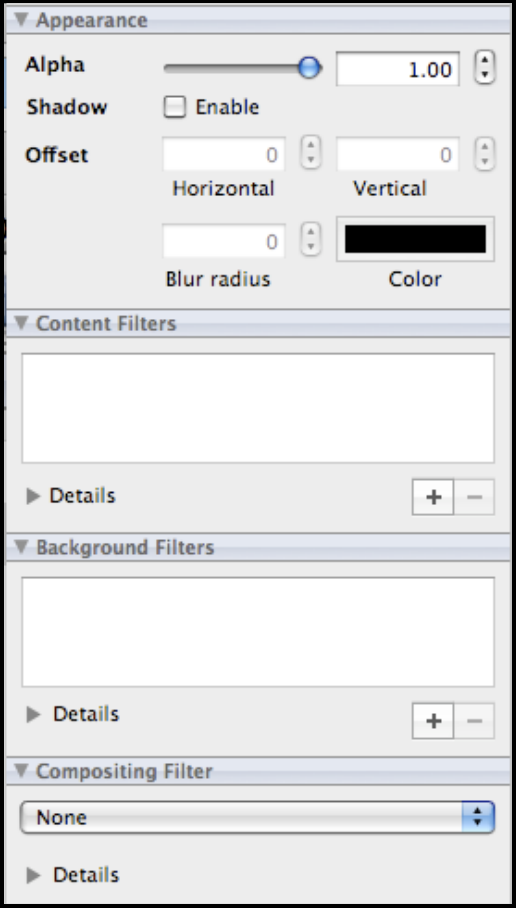
Toggle Layer-Backed Mode as Needed



```
[view setWantsLayer:NO];
```

New NSView Properties

- Properties used only by layer-backed views
- Visual Properties
    - alphaValue
    - shadow
    - contentFilters
    - backgroundFilters
    - compositingFilter



Demo

Using Default Animations

James Dempsey
Application Frameworks Engineer

Fundamentals Summary

- To trigger a default animation in a view or window, use the animator

  [[window animator] setFrame: newFrameRect];

- Use NSAnimationContext to
  - Group multiple changes together
  - Change the duration of an animation
- Make any view hierarchy layer-backed
  [view setWantsLayer: YES];

Questions?