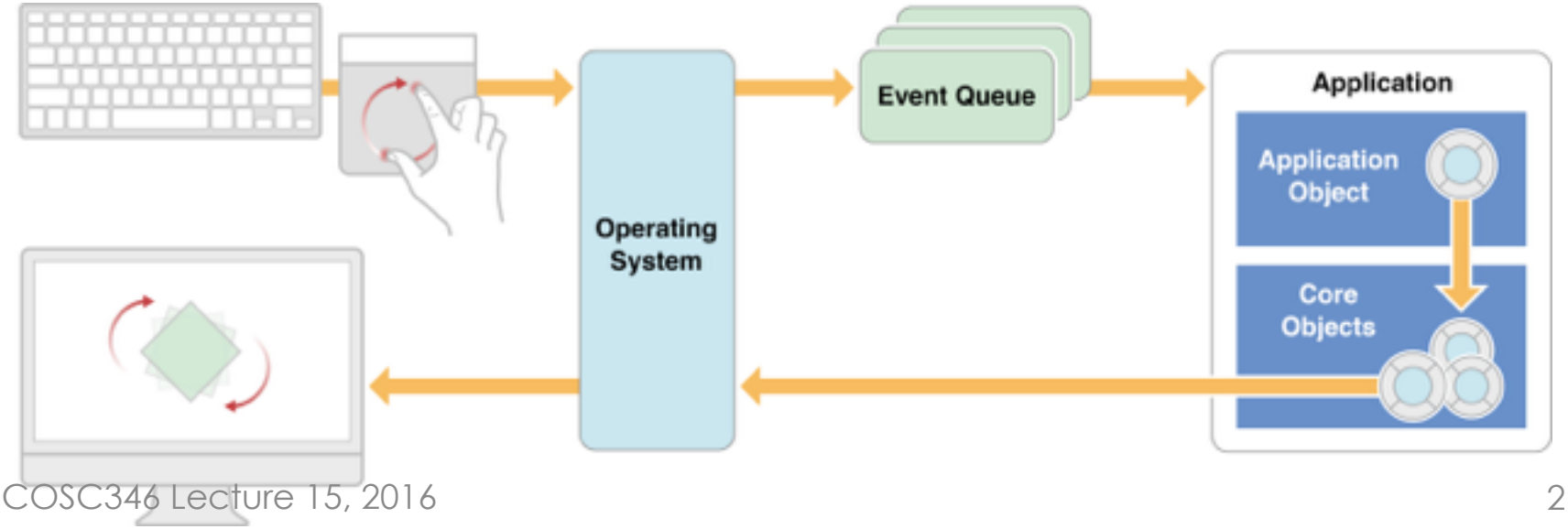
Application Programming on the Mac

COSC346

Mac OS X Application

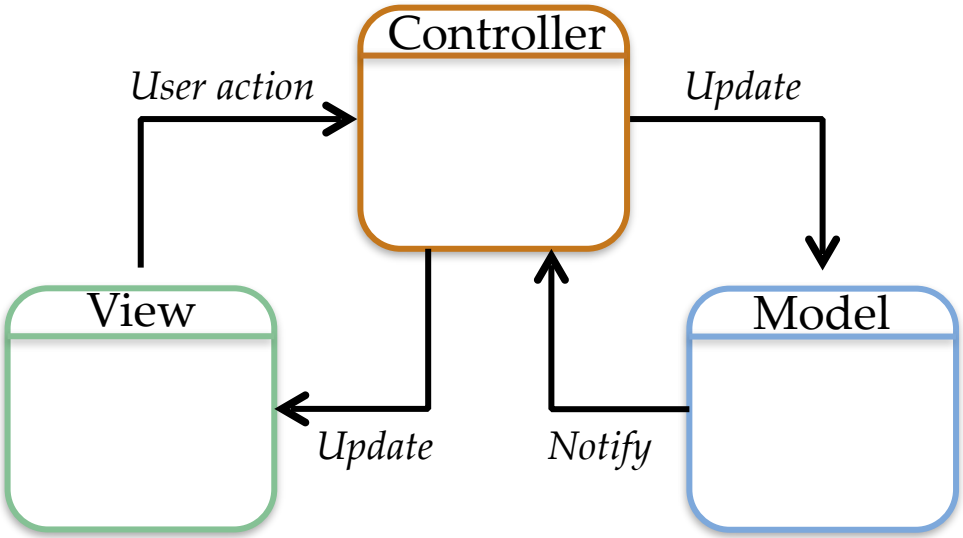
- An application is a complex system: made of many subcomponents
  - Graphical interface
  - Event handling
  - Multi-threading
  - Data processing
  - Storage



Cocoa Environment

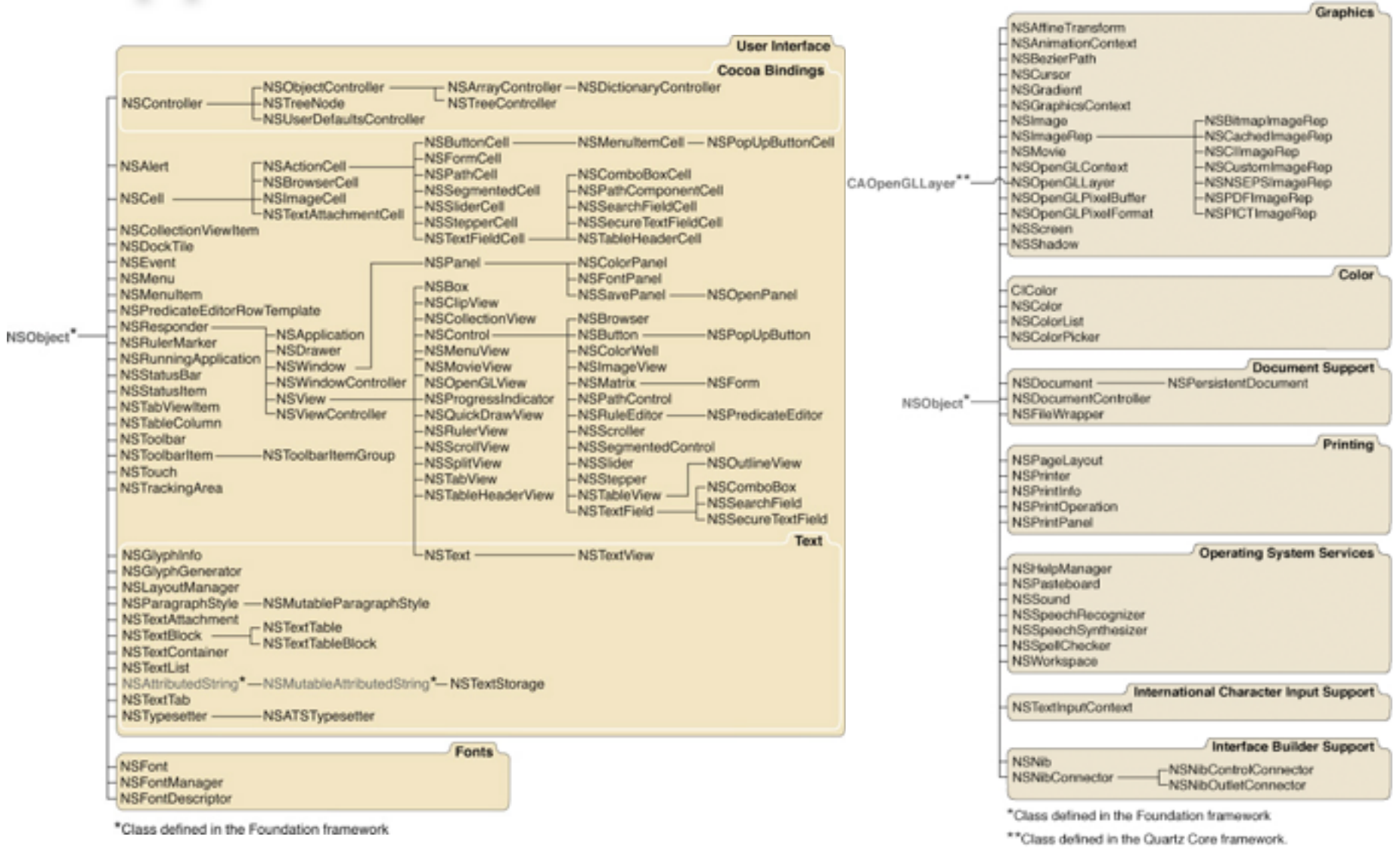
- Cocoa is a collection of frameworks & libraries. Key parts:
- **AppKit**
  - Provides a set of elements for GUI: windows, views, buttons,…
  - Provides controllers that glue model & views together
  - Abstracts away most of the logic "under-the-hood"—such as the mouse and keyboard event handling, etc.
- **Core Data**
  - Abstracts away data storage
  - Options for XML, binary files, or SQLite database for storage
- **Foundation Framework**
  - Library for custom logic binding all the other elements together

"One Pattern to Rule them All"



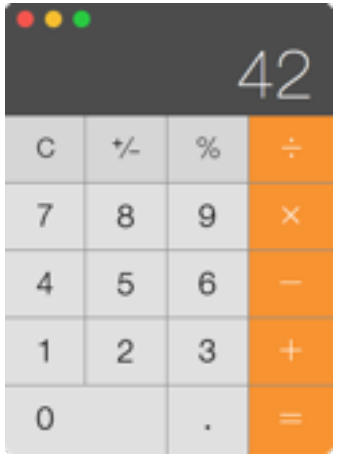
- Model-View-Controller (MVC)
  - **Model**—information storage
  - **View**—interface that allows user to interact with information
  - **Controller**—coordinates interaction between view & model
    Sole purpose: decouple view & model as much as possible
- Cocoa Framework heavily utilises the MVC pattern

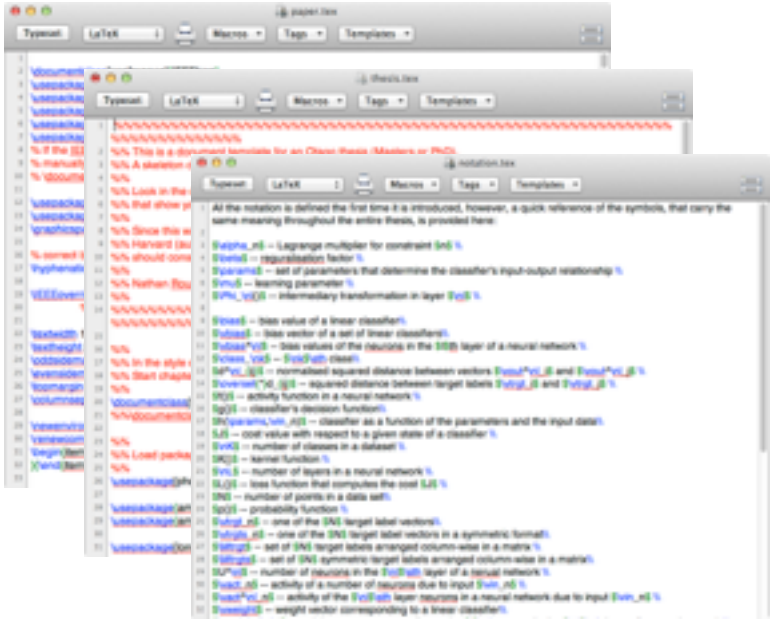
AppKit


User Interface
Cocoa Bindings
NSController
NSObjectController
NSTreeNode
NSUserDefaultsController
NSArrayController
NSTreeController
NSDictionaryController
NSAlert
NSCell
NSActionCell
NSBrowserCell
NSImageCell
NSTextAttachmentCell
NSButtonCell
NSFormCell
NSPathCell
NSSegmentedCell
NSSliderCell
NSStepperCell
NSTextFieldCell
NSMenuItemCell
NSComboBoxCell
NSPathComponentCell
NSSearchFieldCell
NSSecureTextFieldCell
NSTableHeaderCell
NSPopUpButtonCell
NSCollectionViewItem
NSDockTile
NSEvent
NSMenu
NSMenuItem
NSPredicateEditorRowTemplate
NSObject*
NSResponder
NSRulerMarker
NSRunningApplication
NSStatusBar
NSStatusItem
NSTabViewItem
NSTableColumn
NSToolbar
NSToolbarItem
NSTouch
NSTrackingArea
NSApplication
NSDrawer
NSWindow
NSWindowController
NSView
NSViewController
NSToolbarItemGroup
NSPanel
NSColorPanel
NSFontPanel
NSSavePanel
NSOpenPanel
NSBox
NSClipView
NSCollectionView
NSControl
NSMenuView
NSMovieView
NSOpenGLView
NSProgressIndicator
NSQuickDrawView
NSRulerView
NSScrollView
NSSplitView
NSTabView
NSTableHeaderView
NSBrowser
NSButton
NSColorWell
NSImageView
NSMatrix
NSPathControl
NSRuleEditor
NSScroller
NSSegmentedControl
NSSlider
NSStepper
NSTableView
NSTextField
NSPopUpButton
NSForm
NSPredicateEditor
NSOutlineView
NSComboBox
NSSearchField
NSSecureTextField
Text
NSGlyphInfo
NSGlyphGenerator
NSLayoutManager
NSParagraphStyle
NSTextAttachment
NSTextBlock
NSTextContainer
NSTextList
NSAttributedString*
NSTextTab
NSTypesetter
NSMutableParagraphStyle
NSTextTable
NSTextTableBlock
NSMutableAttributedString*
NSTextStorage
NSText
NSTextView
NSATSTypesetter
Fonts
NSFont
NSFontManager
NSFontDescriptor
*Class defined in the Foundation framework
Graphics
CAOpenGLLayer**
NSAffineTransform
NSAnimationContext
NSBezierPath
NSCursor
NSGradient
NSGraphicsContext
NSImage
NSImageRep
NSMovie
NSOpenGLContext
NSOpenGLLayer
NSOpenGLPixelBuffer
NSOpenGLPixelFormat
NSScreen
NSShadow
NSBitmapImageRep
NSCachedImageRep
NSCIImageRep
NSCustomImageRep
NSNSEPSImageRep
NSPDFImageRep
NSPICTImageRep
Color
CIColor
NSColor
NSColorList
NSColorPicker
Document Support
NSObject*
NSDocument
NSDocumentController
NSFileWrapper
NSPersistentDocument
Printing
NSPageLayout
NSPrinter
NSPrintInfo
NSPrintOperation
NSPrintPanel
Operating System Services
NSHelpManager
NSPasteboard
NSSound
NSSpeechRecognizer
NSSpeechSynthesizer
NSSpellChecker
NSWorkspace
International Character Input Support
NSTextInputContext
Interface Builder Support
NSNib
NSNibConnector
NSNibControlConnector
NSNibOutletConnector
*Class defined in the Foundation framework.
**Class defined in the Quartz Core framework.

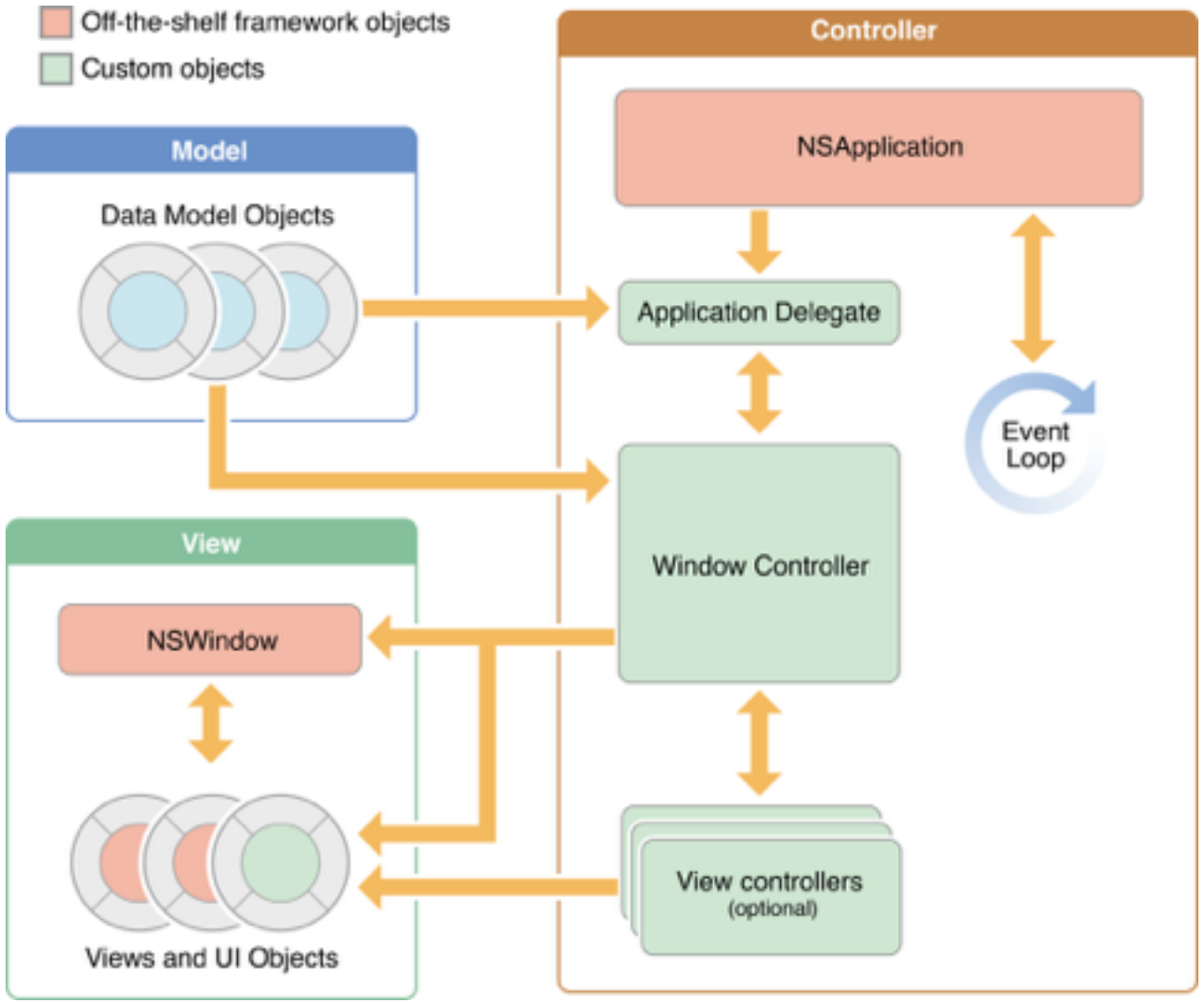
Application types

- Single-window utility app
- Single-window library-style app
- Multi-window document-based app





Single-window app



NSApplication class

- Every Cocoa application runs exactly one instance of an `NSApplication` object
  - Manages the lifecycle of an application
  - `NSApplication` is a singleton
  - Instantiated and run from the main function of your program executable
  - `NSApp` is a global reference for the `NSApplication` object instance
- Handles the loading of GUI at the start and keeps track of windows
  - For instance, which window has the focus, in terms of user input
- Runs the main event loop
  - Collects and dispatches application events, such as user input
  - Handles redrawing
- Your program becomes a delegate of the `NSApplication` object instance, called after the application is loaded

NSApplication class

- To get a reference to the running application

```
let application = NSApplication.sharedApplication()
```

or

```
let app = NSApp as! NSApplication
```

- Has methods for:
  - Terminating the application
  - Maximising/minimising/hiding windows
  - Updating windows
  - Managing menus

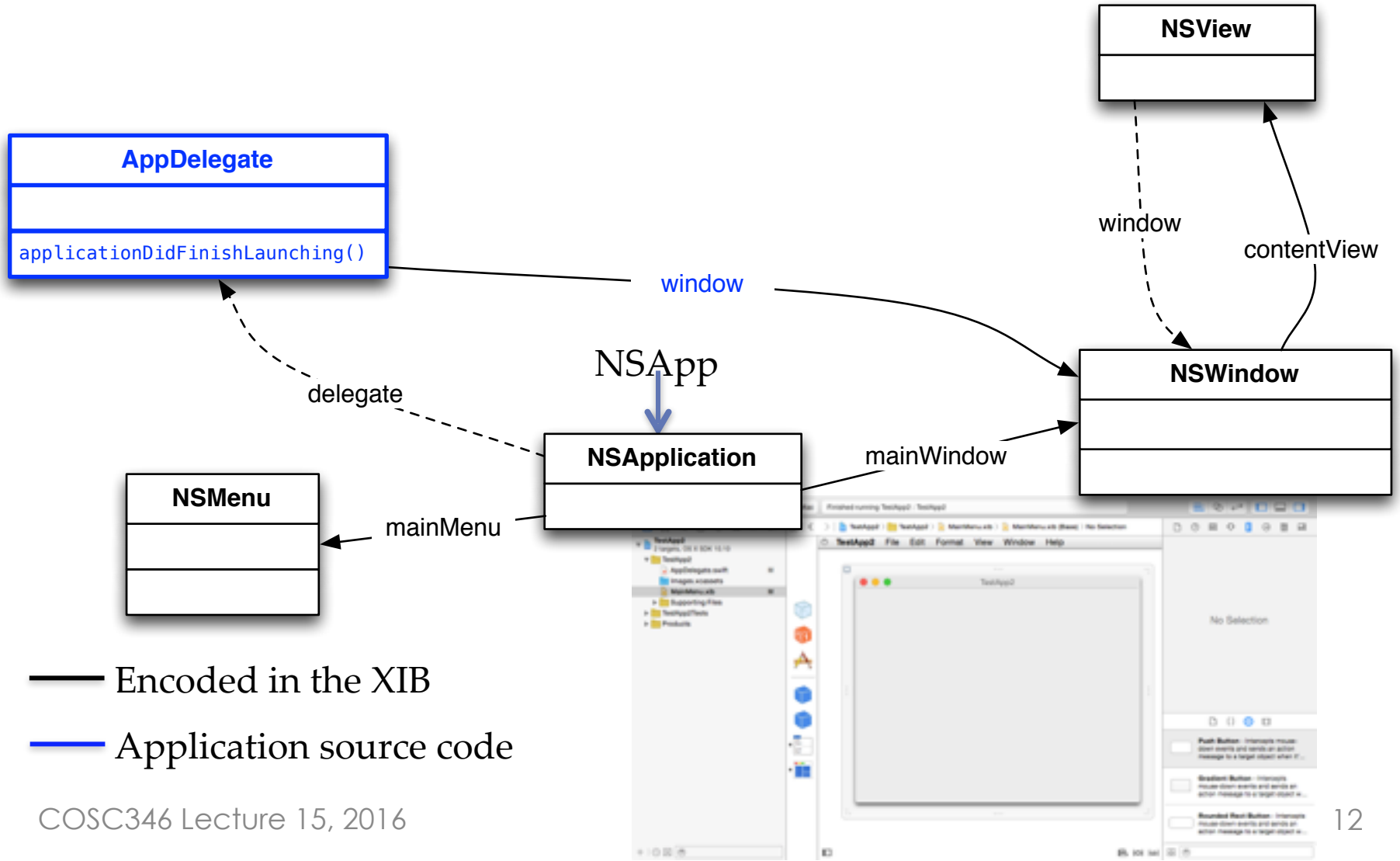
XIB and NIB

- XIB—"Xcode/XML Interface Builder"
  - The XML file in your Xcode project that contains description of all the visual components added in the Interface Builder
  - The interface shown in the Interface Builder is a rendering corresponding to the contents of this file
  - You do not edit this file directly—when you modify your app's interface in the Interface Builder, the contents of XIB will change
- NIB
  - The compiled code corresponding to the XIB file
  - This is a binary file that saves all the objects corresponding to the AppKit's classes specified in the XIB file
  - This file becomes part of your application bundle, but you never access it directly—the application will load it automatically when it starts

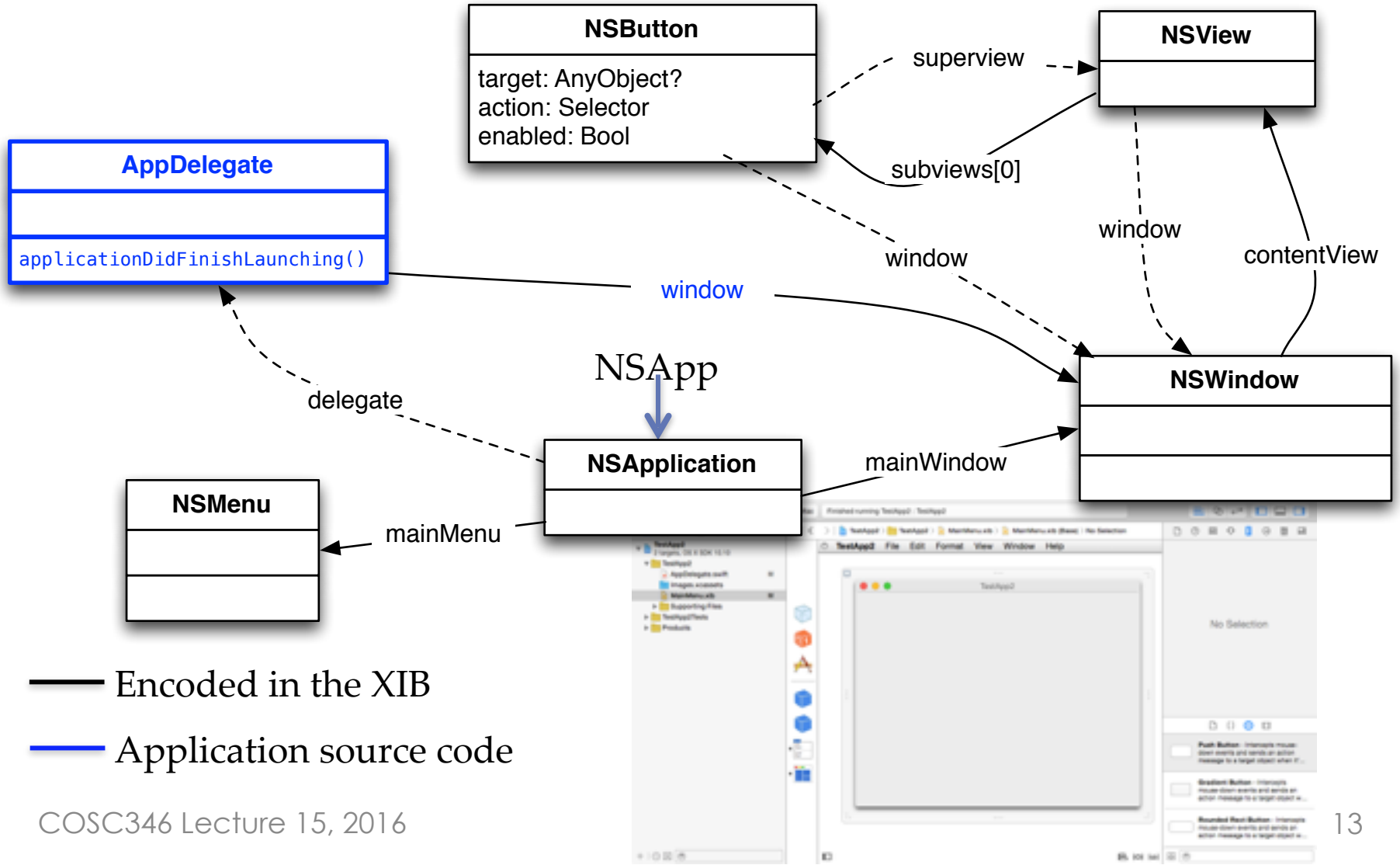
Application bundle contents

- Executable file that starts the application
  - Mac OS X hides the fact that the bundle is a directory—it looks just like a single file
  - Clicking on .app bundle runs the executable code in the bundle
- NIB file
  - This file stores all the graphical elements from Xcode's Interface Builder that are part of your application
  - When your application starts, the NIB file is one of the first things to get loaded
- Other files that you included in your Xcode project
  - Images, media, etc.
  - You can access these resources by loading them from the main bundle—the path of the bundle can be derived from the `NSBundle` object corresponding to your application bundle

Default Cocoa application

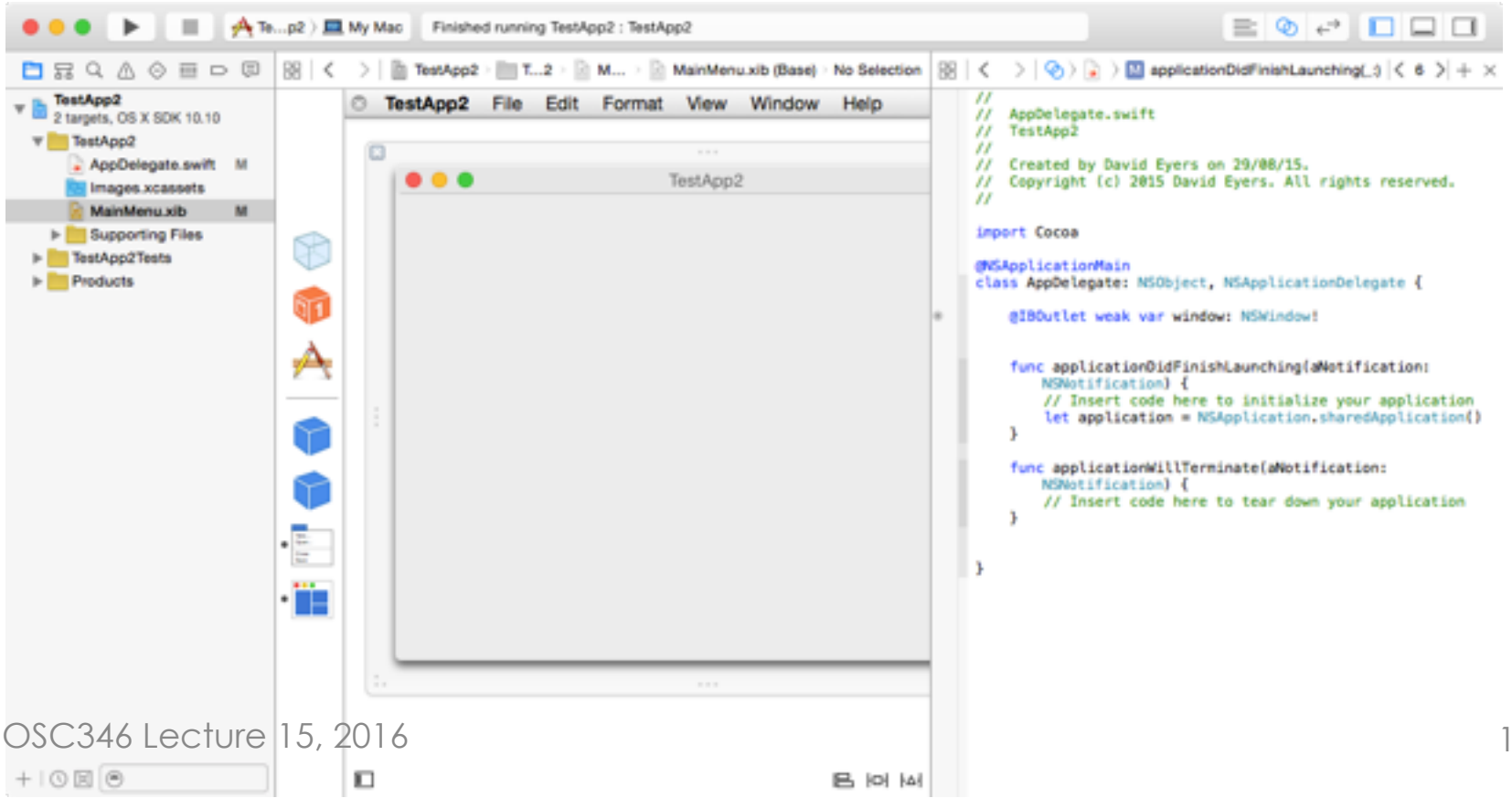


Custom Cocoa application



Application Delegate

- Code to control your application goes in the "AppDelegate.swift" file

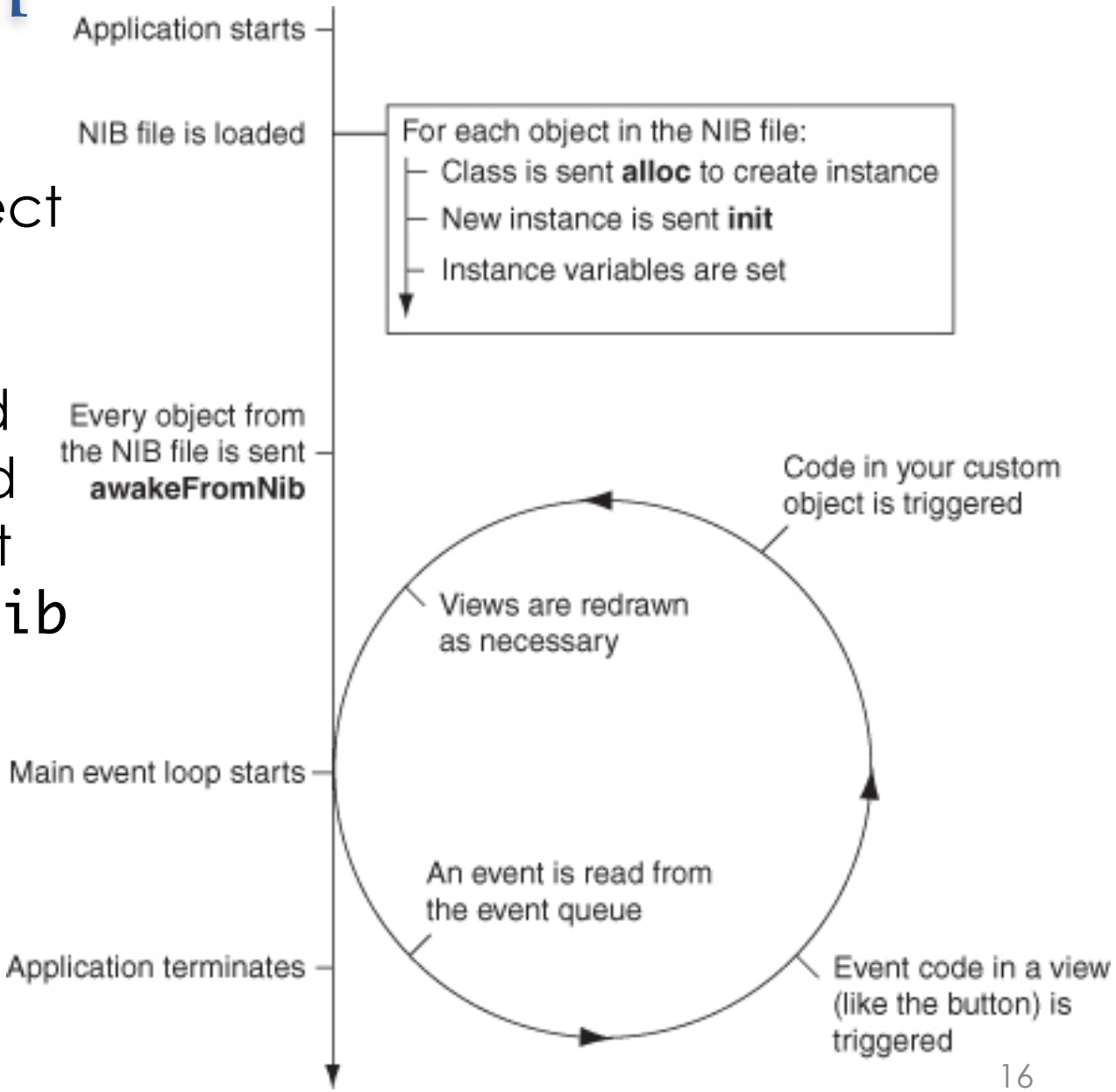


Application Delegate

- Application delegate serves the `NSApplication` object
- A.D. subscribes to the `NSApplicationDelegate` protocol, which contains optional methods for
  - Launching, terminating, managing the active status, and hiding your application
  - Managing windows and dock actions associated with your application.
  - Opening and printing files.
- Your code should be placed in the `applicationDidFinishLaunching`: method, which gets invoked by `NSApplication` after it's done loading the GUI from NIB

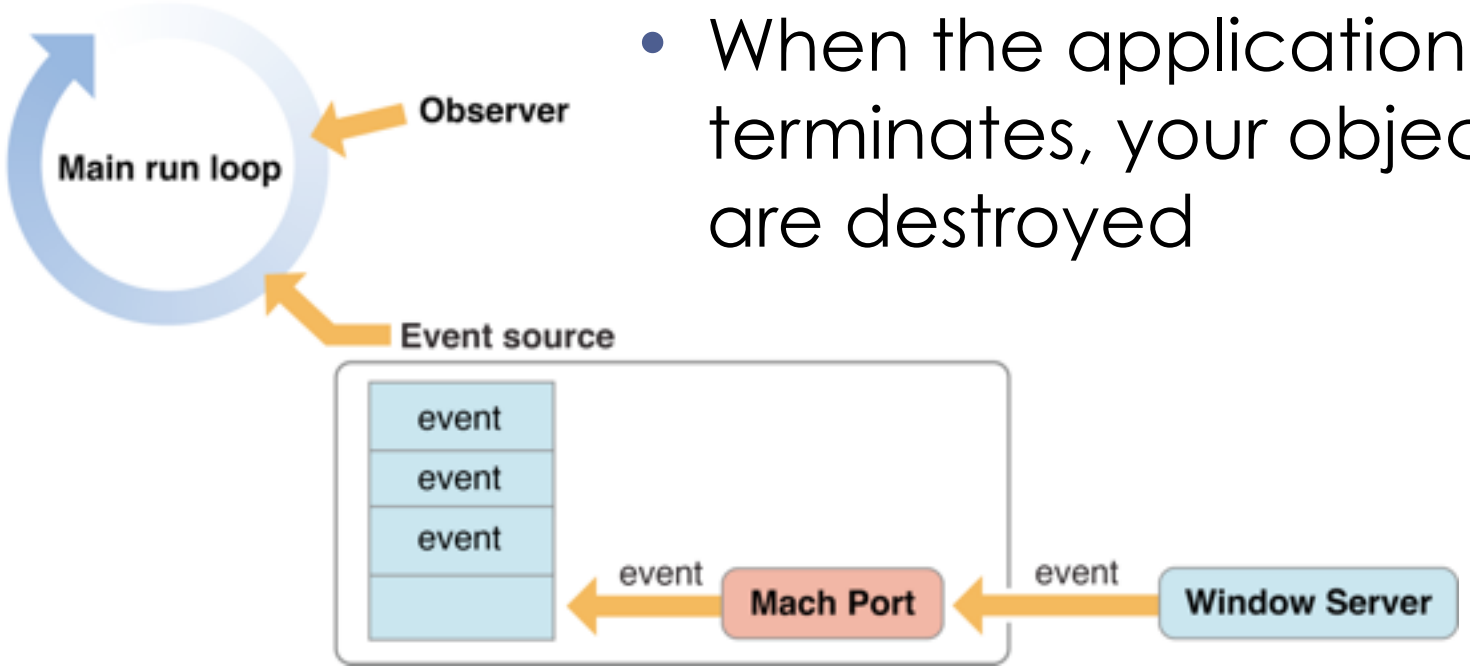
Application execution

- After each object is unarchived from the NIB file and connected (via actions and outlets), it is sent an `awakeFromNib` message.



Main event loop

- Waits for events from the OS and dispatches them to appropriate handlers
- The autorelease pool is drained after each pass through the event loop
- When the application terminates, your objects are destroyed



NSRunLoop class

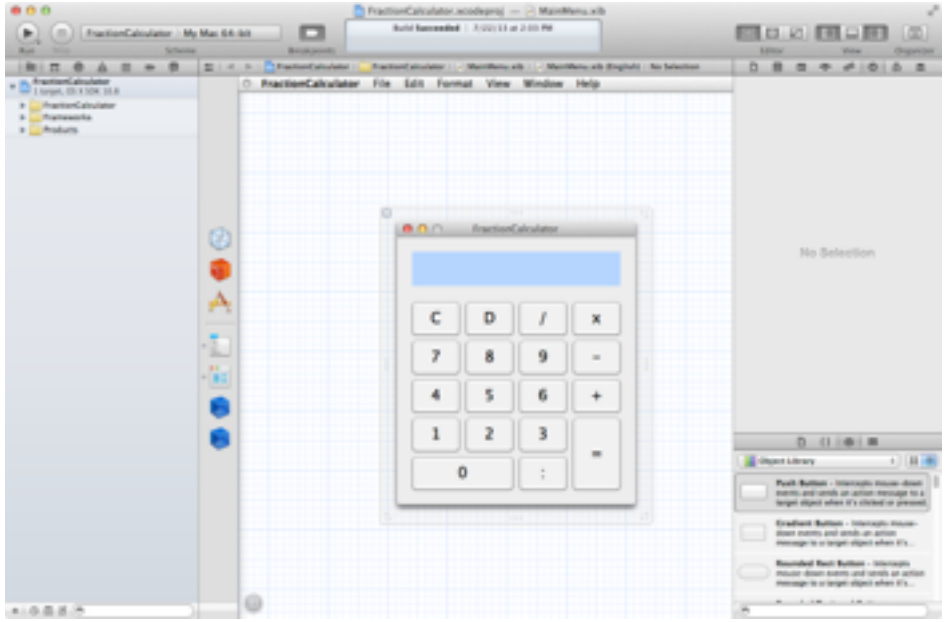
- You can get a reference to the main loop running in the **`NSApplication`** instance as follows:

```
let loop = NSRunLoop.currentRunLoop()
```

- Reference to the current loop is useful for adding timers and communication ports

Interface Builder

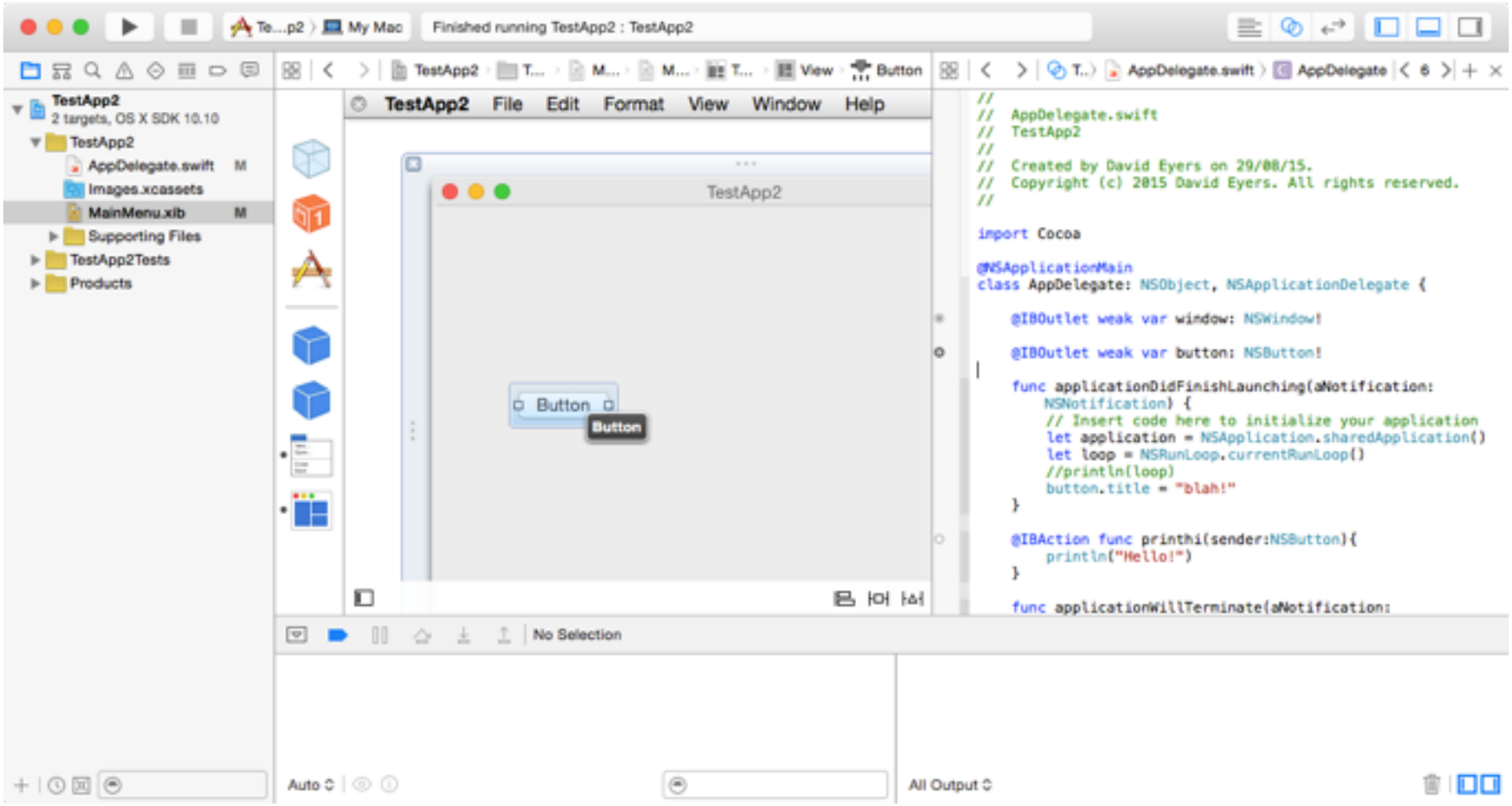
- Xcode's GUI for creating Cocoa Applications
- You can design the look of your application by dropping various visual elements in the application window
- You can connect graphically various visual elements to your application
  - **Target** and **Action**—connect controls to code that is invoked when user interacts with the control
  - **Outlets**—references in your code to various visual elements, so that they can be controlled programmatically



Outlets

- How do you reference in your code the objects corresponding to the UI elements created in the Interface Builder?
- In Cocoa, these references are called **outlets**
  - When you create a single-window project, the application delegate gets automatically set up with an outlet called `window`, which is a reference to the main window object of your application
- In the interface definition for the class, which is going to contain a reference to a given UI object, define a `weak var` preceded by the `@IBOutlet` annotation
  - The `@IBOutlet` annotation does not change anything in terms of the program, except for being a special marker for the Interface Builder for keeping track of outlets
- In the Interface Builder you can control-click a UI element and connect it to an outlet

Outlets


TestApp2
2 targets, OS X SDK 10.10
TestApp2
AppDelegate.swift
Images.xcassets
MainMenu.xib
Supporting Files
TestApp2Tests
Products
TestApp2 File Edit Format View Window Help
TestApp2
Button
Button
No Selection
Auto
All Output
//
// AppDelegate.swift
// TestApp2
//
// Created by David Eyers on 29/08/15.
// Copyright (c) 2015 David Eyers. All rights reserved.
//
import Cocoa
@NSApplicationMain
class AppDelegate: NSObject, NSApplicationDelegate {
@IBOutlet weak var window: NSWindow!
@IBOutlet weak var button: NSButton!
func applicationDidFinishLaunching(aNotification:
NSNotification) {
// Insert code here to initialize your application
let application = NSApplication.sharedApplication()
let loop = NSRunLoop.currentRunLoop()
//println(loop)
button.title = "blah!"
}
@IBAction func printhi(sender:NSButton){
println("Hello!")
}
func applicationWillTerminate(aNotification:

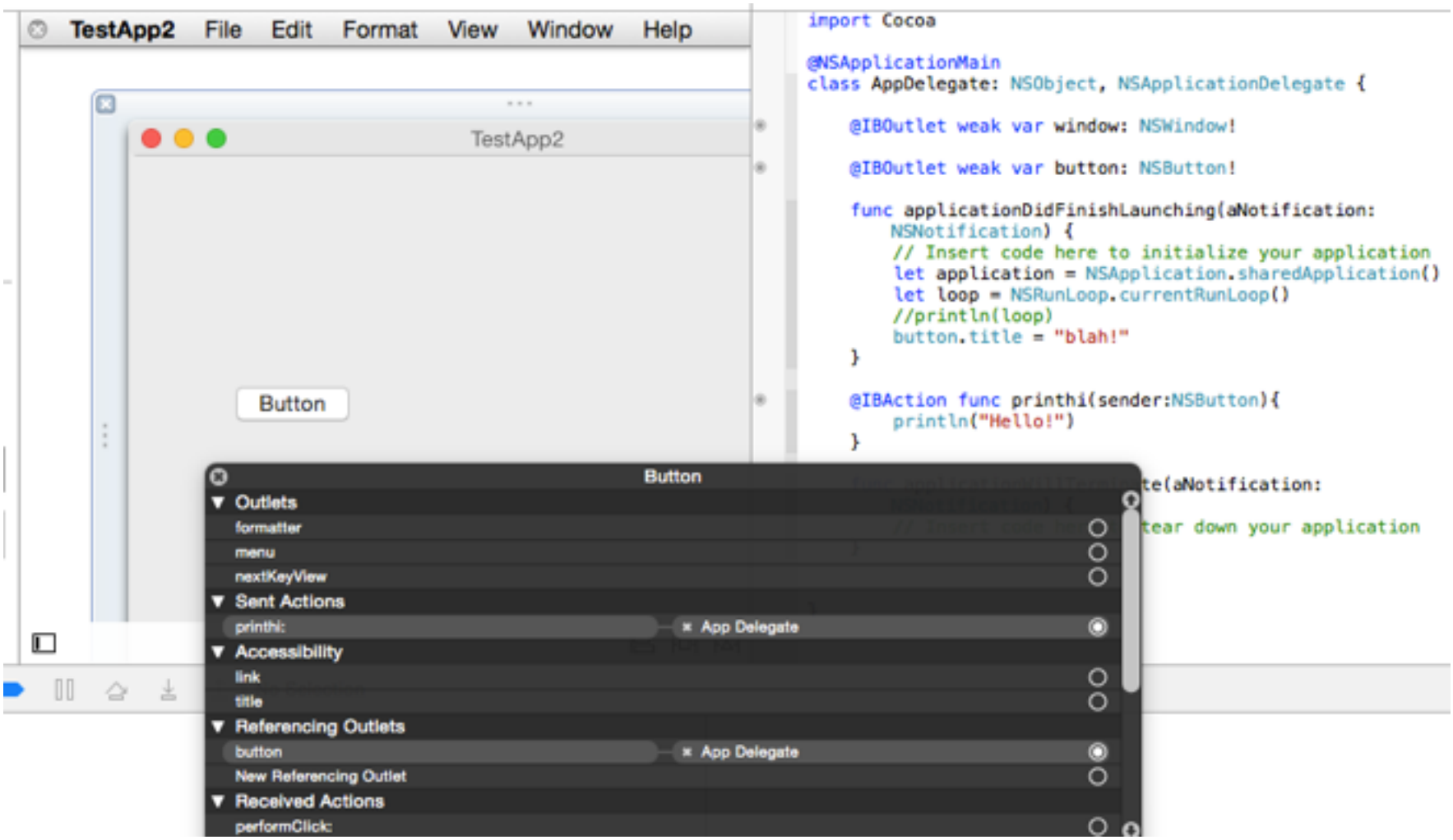
Targets and Actions

- What do you want to happen when a user clicks on a button, or a slider, or a checkbox, or other UI control element?
- Create an **action**—it is a method that implements the logic in response to a user interacting with a control element
  - An action method is any method that returns nothing and accepts one parameter (identifying the sender)
- The object implementing the **action** for some control element is referred to as the **target**

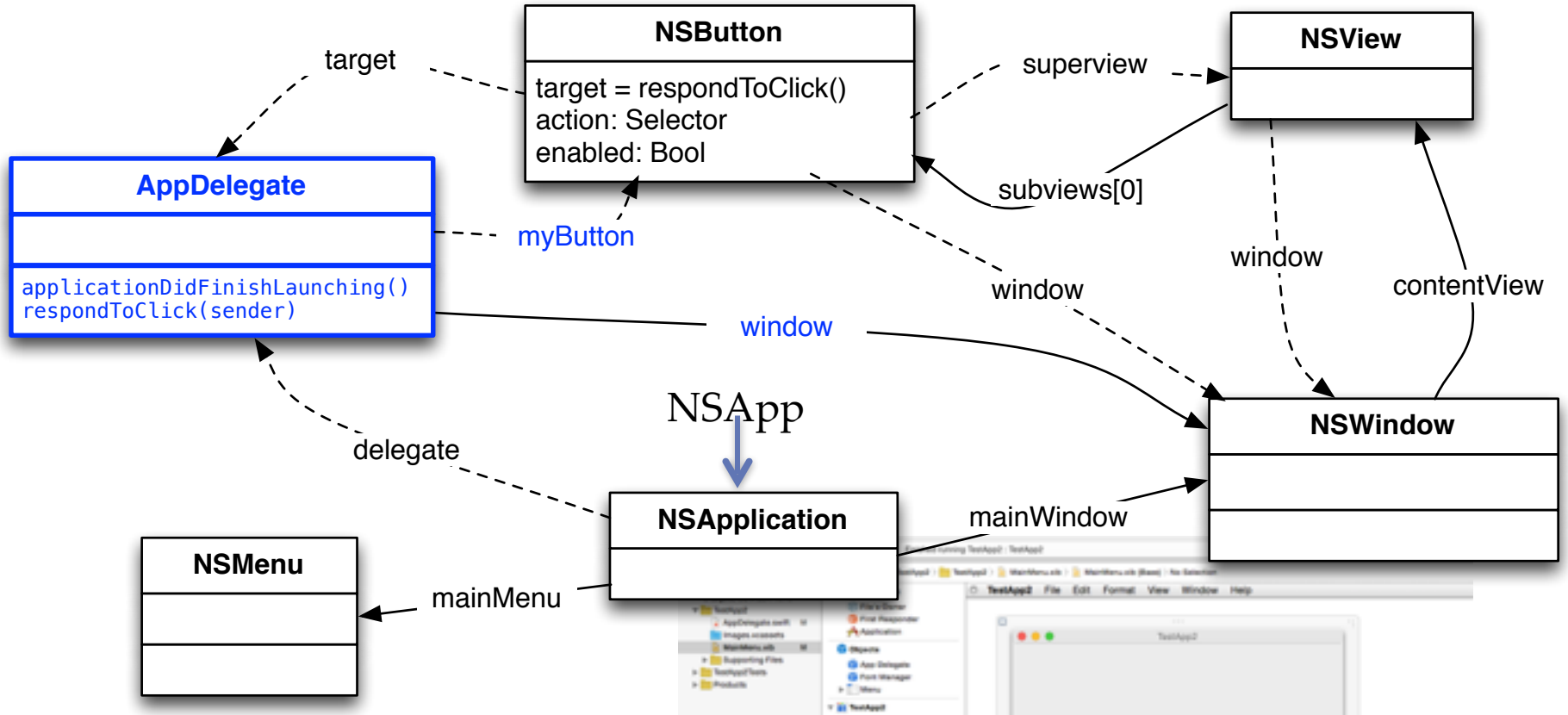
Targets and Actions

- Connection of UI elements to corresponding targets and actions can be done graphically in the Interface Builder
- In the interface for the target class specify an action method using the `@IBAction` annotation
  - The `@IBAction` annotation is used by the Interface Builder to indicate an action method
- In the Interface Builder control-click a UI element and connect it to a specific action

Targets and Actions



Custom Cocoa application



Summary

In this lecture we examined the anatomy of the default Cocoa application. We have also covered the outlet, target and action mechanism which connects each GUI object to the code that dictates their behaviour.

- `NSApplication`—singleton class that runs your application
- `NSApplicationDelegate`—protocol for application delegate with methods for handling various application events
- XIB/NIB—file storing the visual elements
- `NSLoop`—event loop class, useful for running timers
- `@IBOutlet`—annotation that lets Interface Builder know that following pointer is a reference to a GUI element
- `@IBAction`—annotation that lets Interface Builder know that following definition is a an action method to be invoked when a user interacts with a GUI control element