User Interfaces

Lecture 19

Cocoa: Mouse and Keyboard Events

Hamza Bennani
hamza@hamzabennani.com

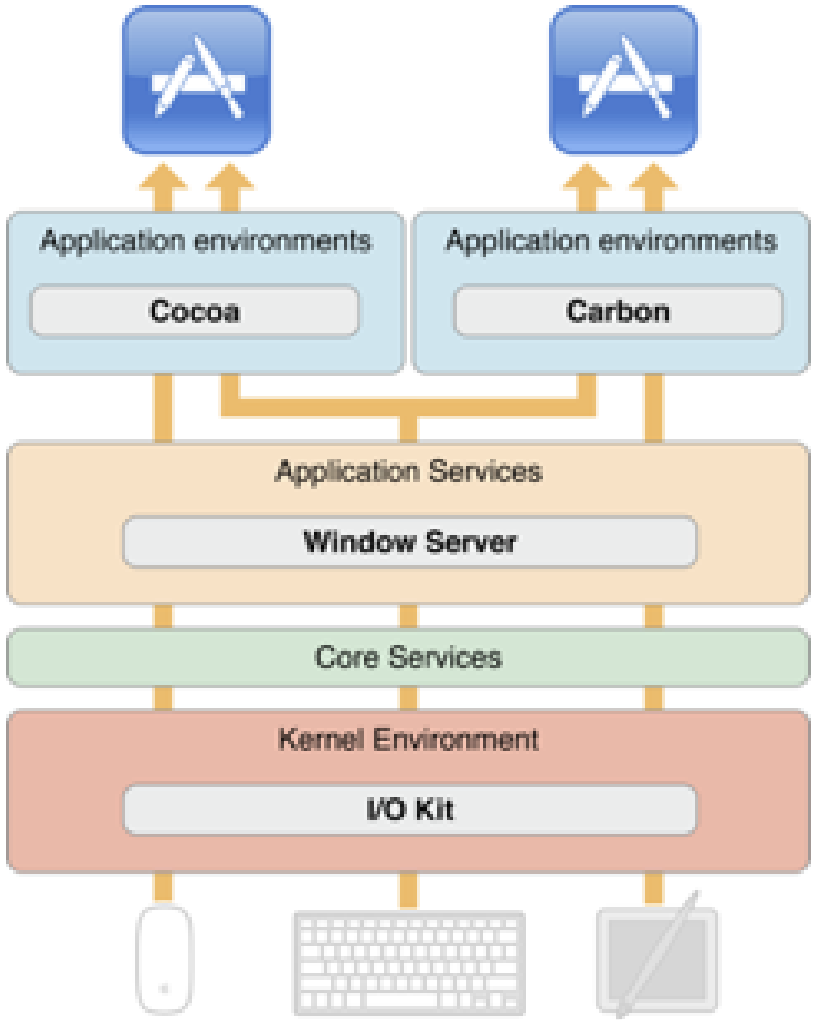
September 18, 2018

Last Lecture

Where did we stop?

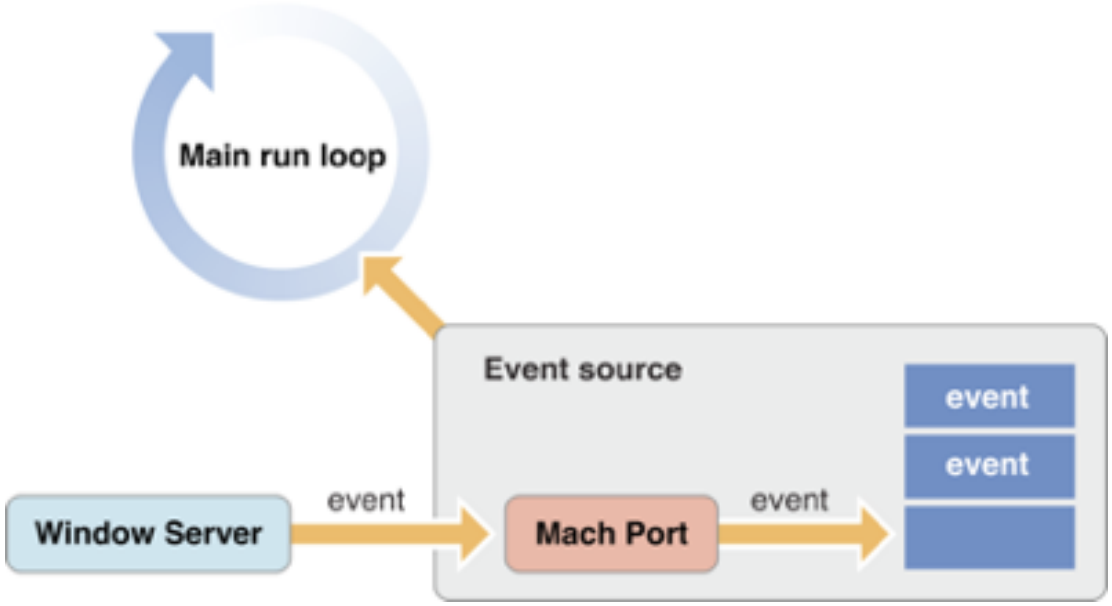
Events

- Events get filtered into a queue by MacOS X
- Some events never reach the application, like ?, or ?
- The active application processes events one at a time from the queue

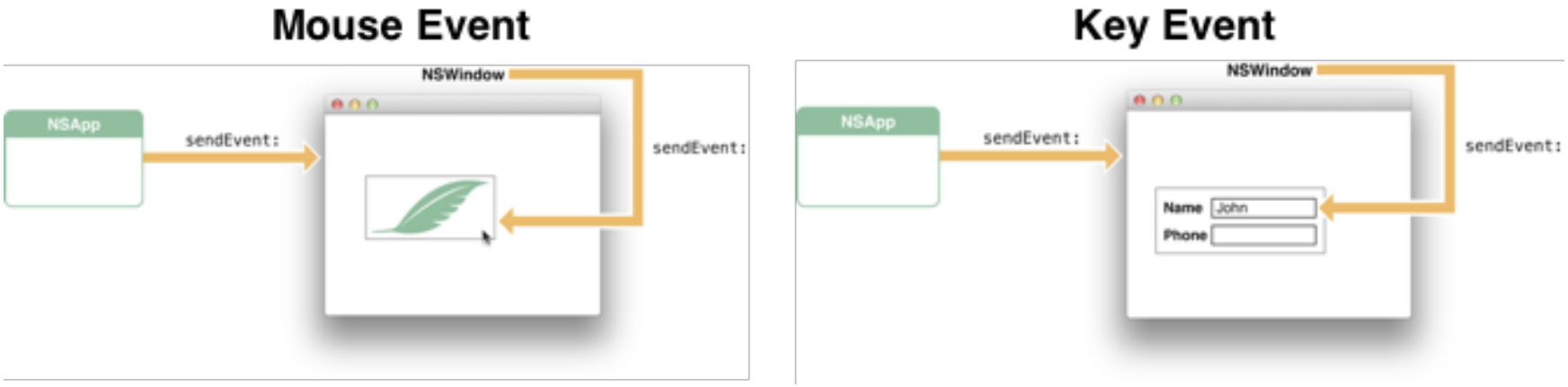


The event loop

- Events are encoded as NSEvent objects and put into a queue
  - The active application takes events from the queue and processes them
  - NSEvent contains information such as location, time, etc . . .



Where events go?



- The active application NSApp object issues a sendEvent: message to the active window
  - A mouse event is forwarded to the view where the mouse is pointing
  - A key event is forwarded to the window's "responder chain"

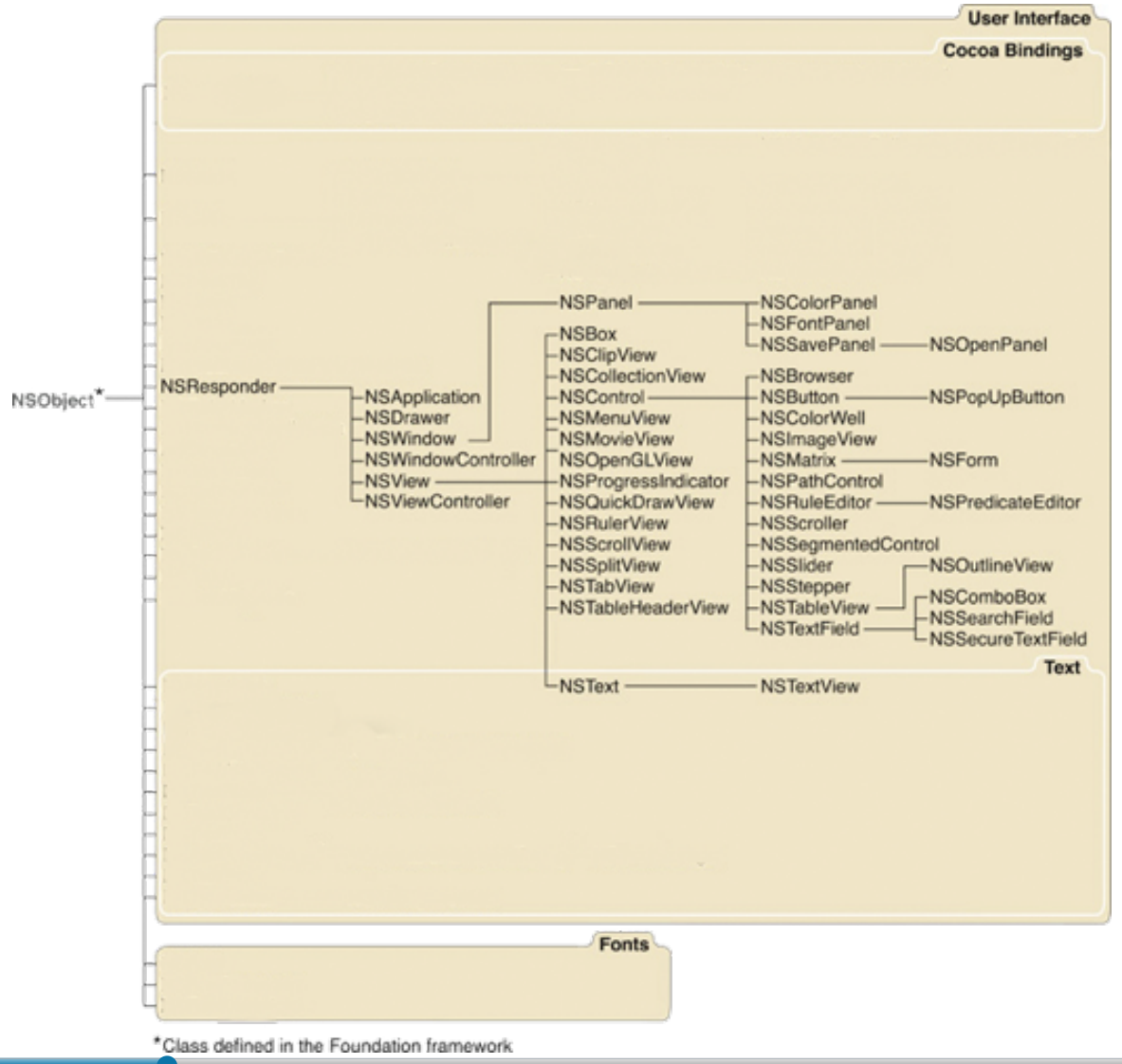
NSEvent

- Events are passed to handling methods as NSEvent objects
- For mouse events, information includes:
  - locationInWindow
  - modifierFlags (NSShiftKeyMask, NSControlKeyMask, . . .)
  - timestamp
  - window - window where event occurred
  - clickCount - number of mouse clicks
  - pressure - tablet
  - deltaX, deltaY - change in position
- For key events, information includes:
  - characters - an NSString object with typed keys
  - isARepeat -YES or NO regarding whether key is held down
  - keyCode - actual key that user pressed
  - modifierFlags - same as mouse modifierFlags

NSResponder

- . . . is an abstract class that dispatches received NSEvents to methods corresponding to various mouse and keyboard events
- All event-handling methods are declared in NSResponder. The following classes inherit from it:
  - NSWindow - so all windows
  - NSView - so almost everything that's in the window, including NSControl objects, such as buttons, etc.
  - NSApplication
  - . . . and even controllers: NSWindowController, NSViewController
- You can override NSResponder messages that you want to handle in your custom view

NSResponder Hierarchy

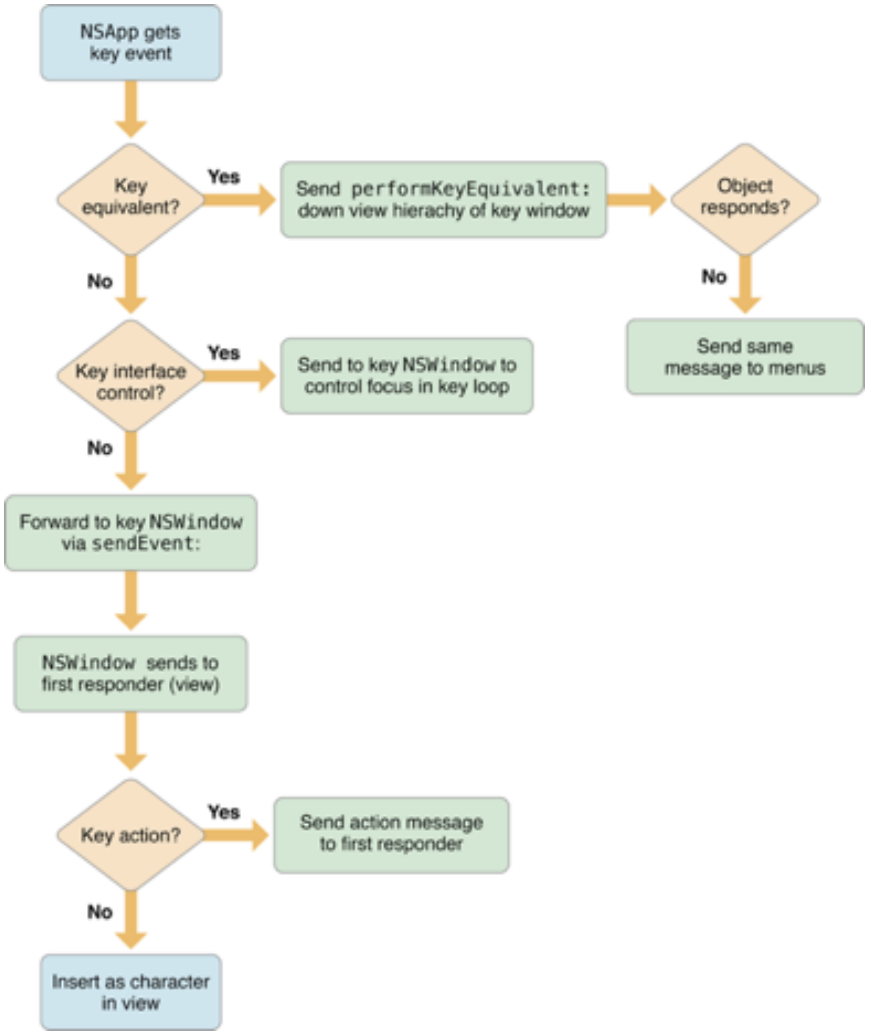


Responding to Mouse Events

- Methods: mouseDown, mouseUp, mouseDragged, mouseEntered, mouseExited, mouseMoved, scrollWheel
  - Note: certain rules apply to mouse events, e.g., mouse-up must be preceded by mouse-down, mouse-dragged must occur between mouse-down and mouse-up, etc.
- The methods get passed an argument of NSEvent type, which responds to methods that provide information about the mouse event:
  - clickCount - number of clicks that occurred
  - windowLocation - location where mouse was clicked
  - Methods to obtain scroll wheel information

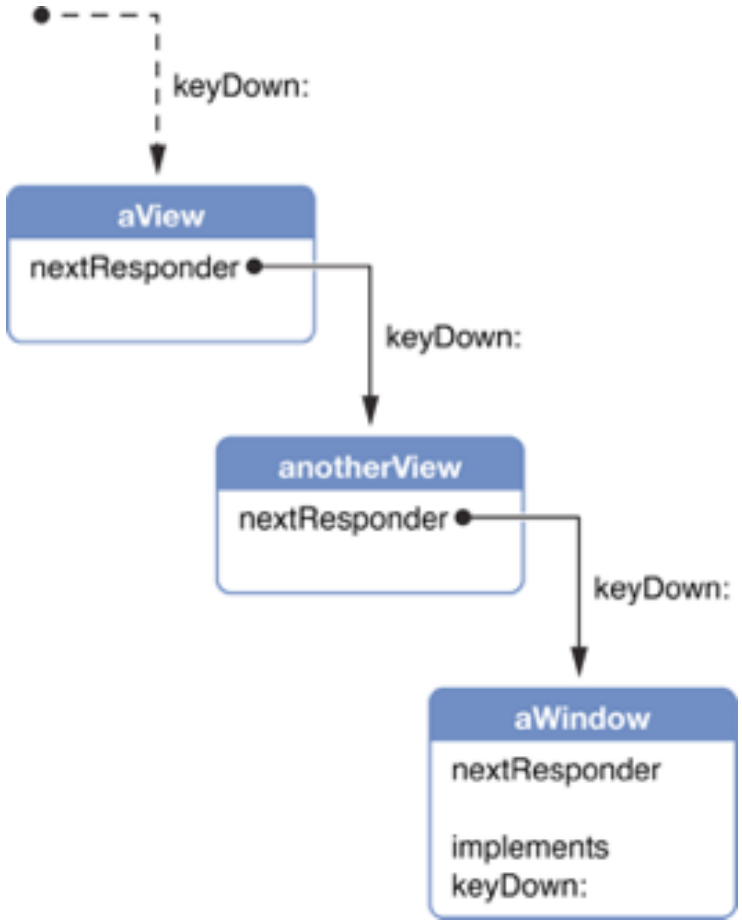
Responding to Key Events

- Key events are more complicated than mouse events, because they lack a target location
- Certain key combinations go straight to menu bar (e.g., command-Z), others to operating system (e.g., Mission Control)

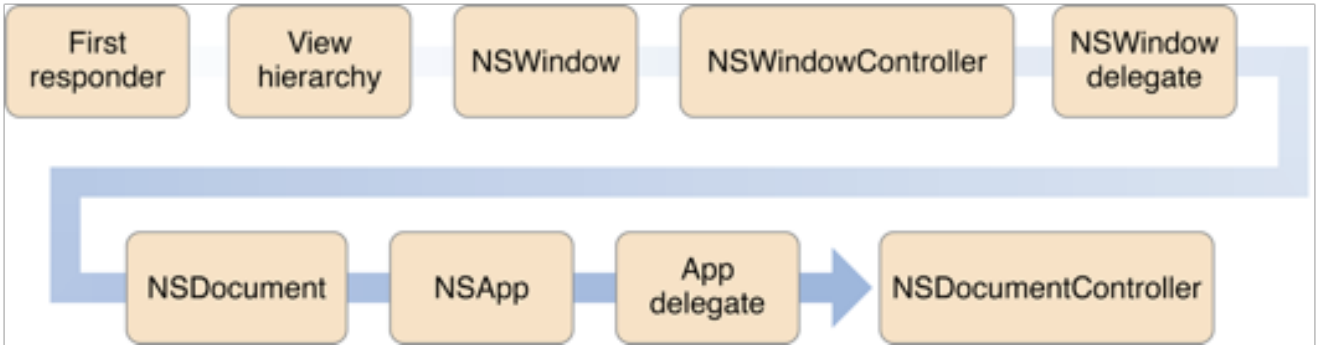


The Responder Chain

- Key events are processed using the responder chain
  - A message is initially passed to the first responder
  - If the first responder does not accept the message, it is passed to the nextResponder
- Responder chain implements the chain of responsibility design pattern
- The responder chain is implemented in NSResponder



The Responder Chain



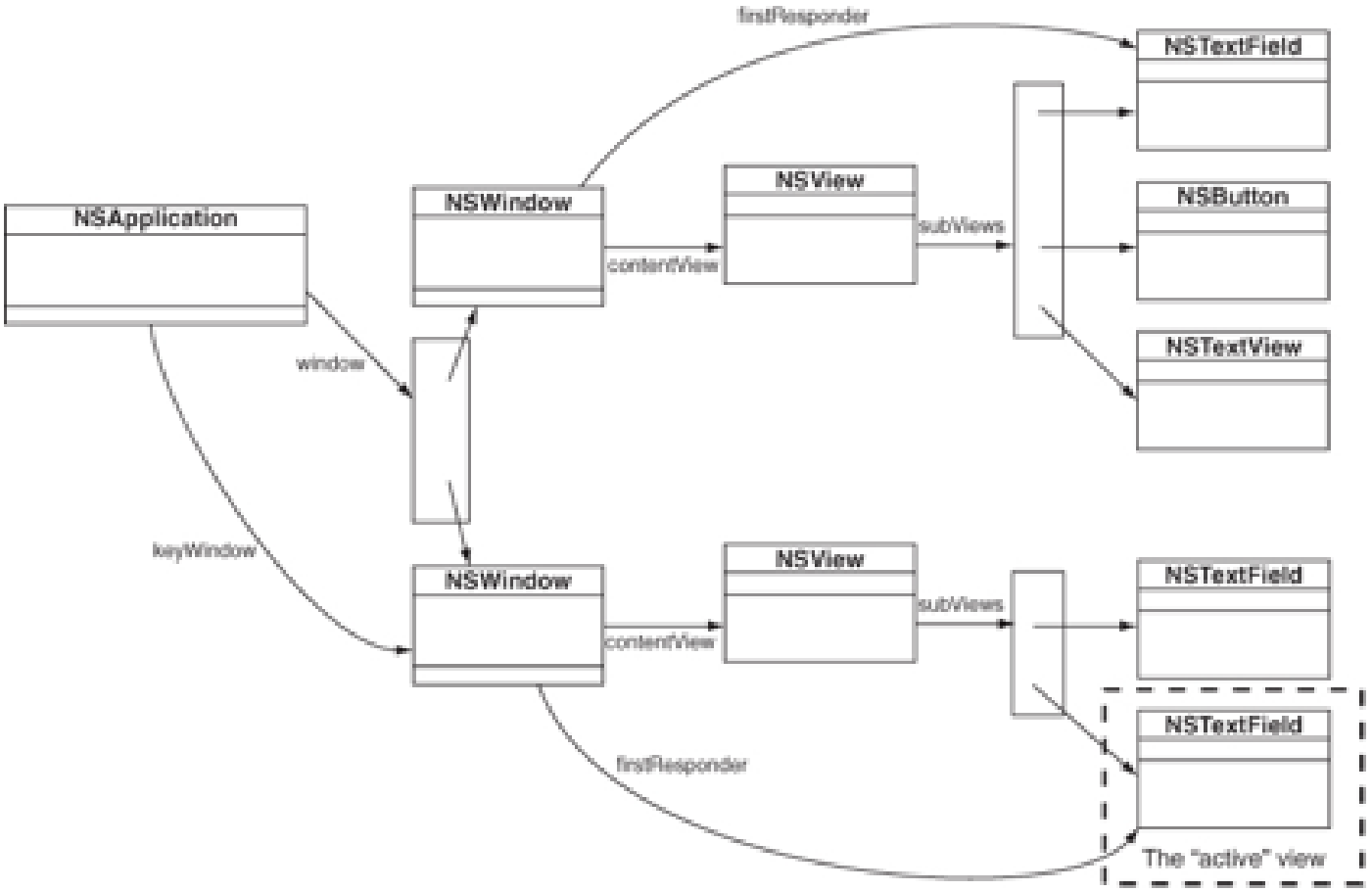
Window Types

- Assumption: in any application, even if it is multi-window, there is only one window with the user's focus. Therefore, the windows are categorised into the following types:
  - Main window - the window that the user is working in currently
  - Key window - the window that accepts user's input
  - Inactive window - all other windows
  - The main and key windows often are the same window - common exception are window panels for opening, saving, preferences, etc., which take input focus, but are not main windows
  - The NSApplication singleton has separate references to the main and key windows (although often these refer to the same window object)

First Responder
Window Change

- Each window has a first responder outlet
  - The active window's first responder gets key events
- The inner workings of NSApplication update its keyWindow outlet to correspond to the window currently selected by the user
  - A mouse click on an inactive window does not send an event to that window, it just changes that window to become the keyWindow

First Responder
Window Change



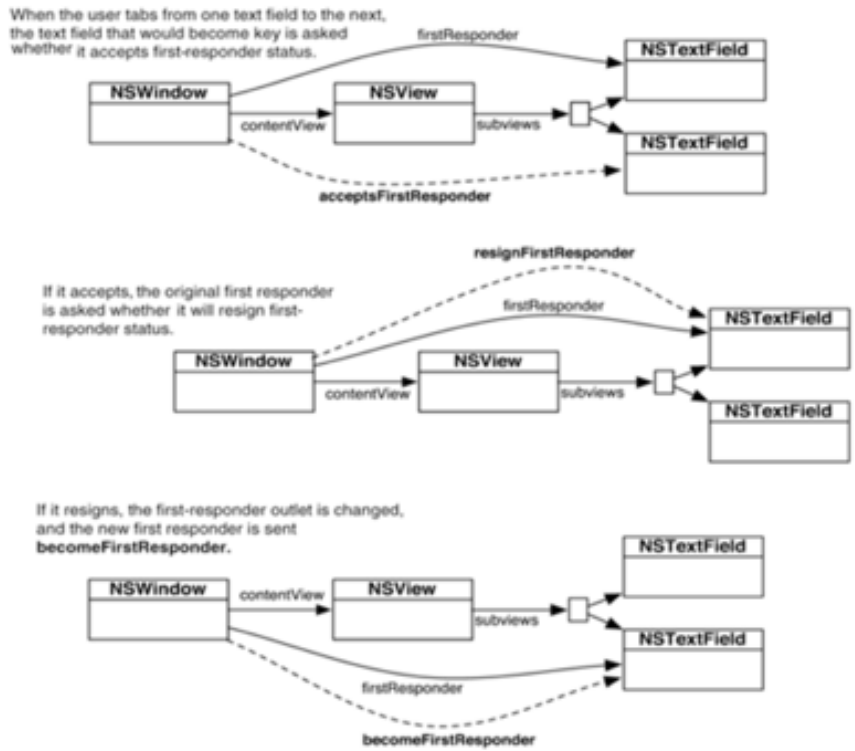
Initial First Responder

- What does the first responder outlet point to just after a window is created?
  - Initial first responder is the element that window's first responder points to by default
- You can set the initial first responder:

```
import Cocoa

class OurWindowController: NSWindowController {
    @IBOutlet weak var view1: NSView!
    @IBOutlet weak var view2: NSView!

    convenience init() {
        self.init()

        self.window?.initialFirstResponder = view1
    }
}
```

First Responder
View Change

- The inner workings of the NSWindow object updates its firstResponder outlet to reference the currently selected view
- The first responder for the application is the object pointed to by the firstResponder reference of the key Window



First Responder Placeholder

- In Interface Builder there's a placeholder object that stands in for an arbitrary first responder
- You can set custom action methods for this placeholder
- You can connect target actions from window elements to the first responder placeholder
- At run-time, when the window element is selected, it sends an action to the object that happens to be the current first responder

NSControl

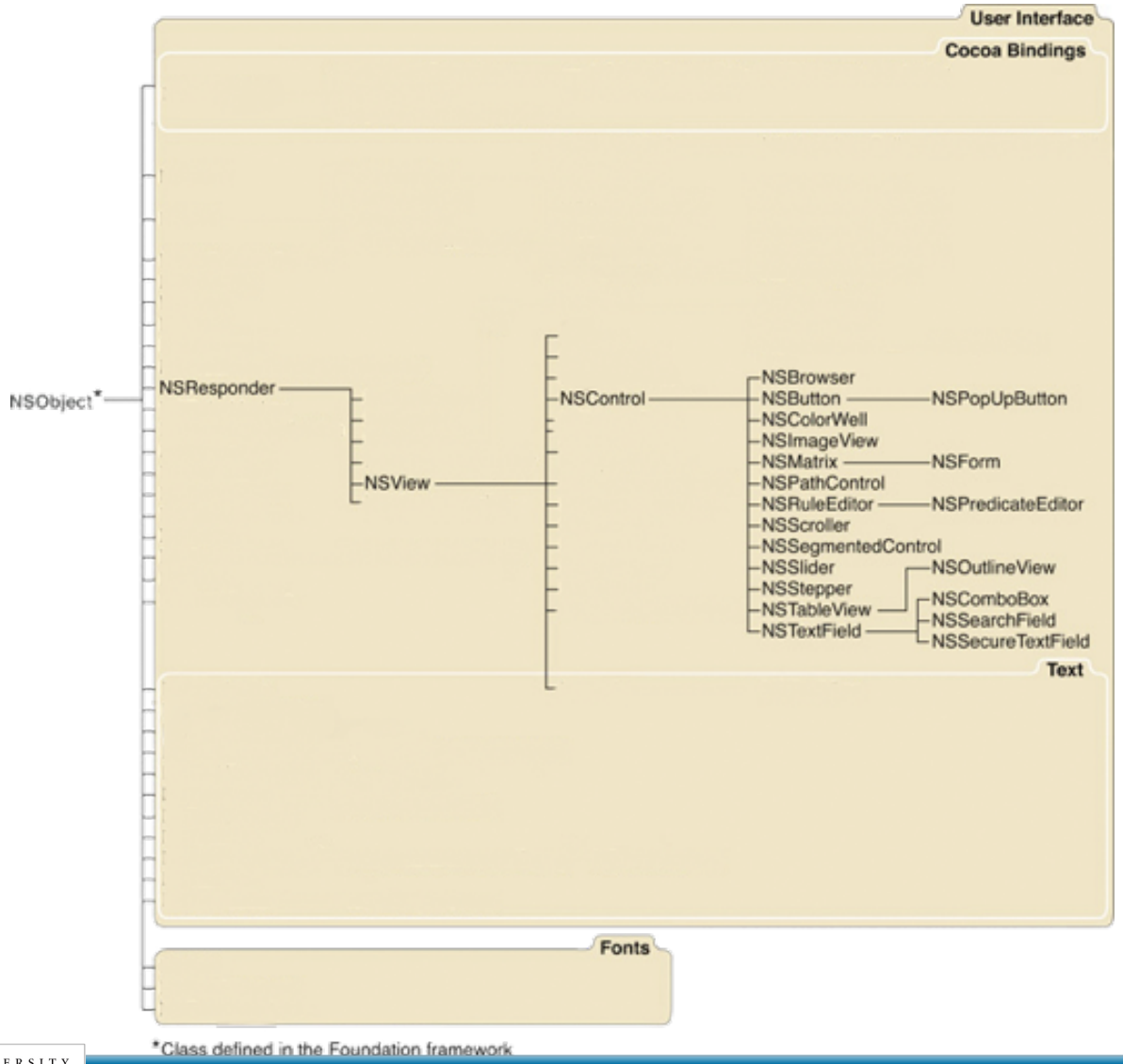
- Object that inherits from NSResponder
- It contains an NSCell object, which has a reference to target object and action selector
- Buttons, check boxes, and many other UI controls inherit from NSControl
  - Each different control overrides selected NSResponder methods to redirect the event to the action selector of the target object (if target and action have been set)
  - Example: NSButton class overrides mouseDown event invoking the action method of its corresponding target
- Target and action values can be set through Interface Builder
  - We have seen how to do this in Lecture 15/16

NSControl

- Target and action values can also be set programmatically
- This example modifies the ViewController of a storyboard
- myB is an NSButton without an IBAction set in the Xcode IB

```
import Cocoa
class ViewController: NSViewController {
    @IBOutlet weak var label: NSTextField!
    @IBOutlet weak var myB: NSButton!
    var count:Int = 0
    override func viewDidLoad() {
        super.viewDidLoad()
        myB.target = self
        myB.action = #selector(ViewController.clickAction(sender:))
    }
    @IBAction func clickAction(sender: AnyObject) {
        count += 1
        label.stringValue = "Click count \(count)"
    }
}
```

NSControl Hierarchy



Things To Watch out for?!

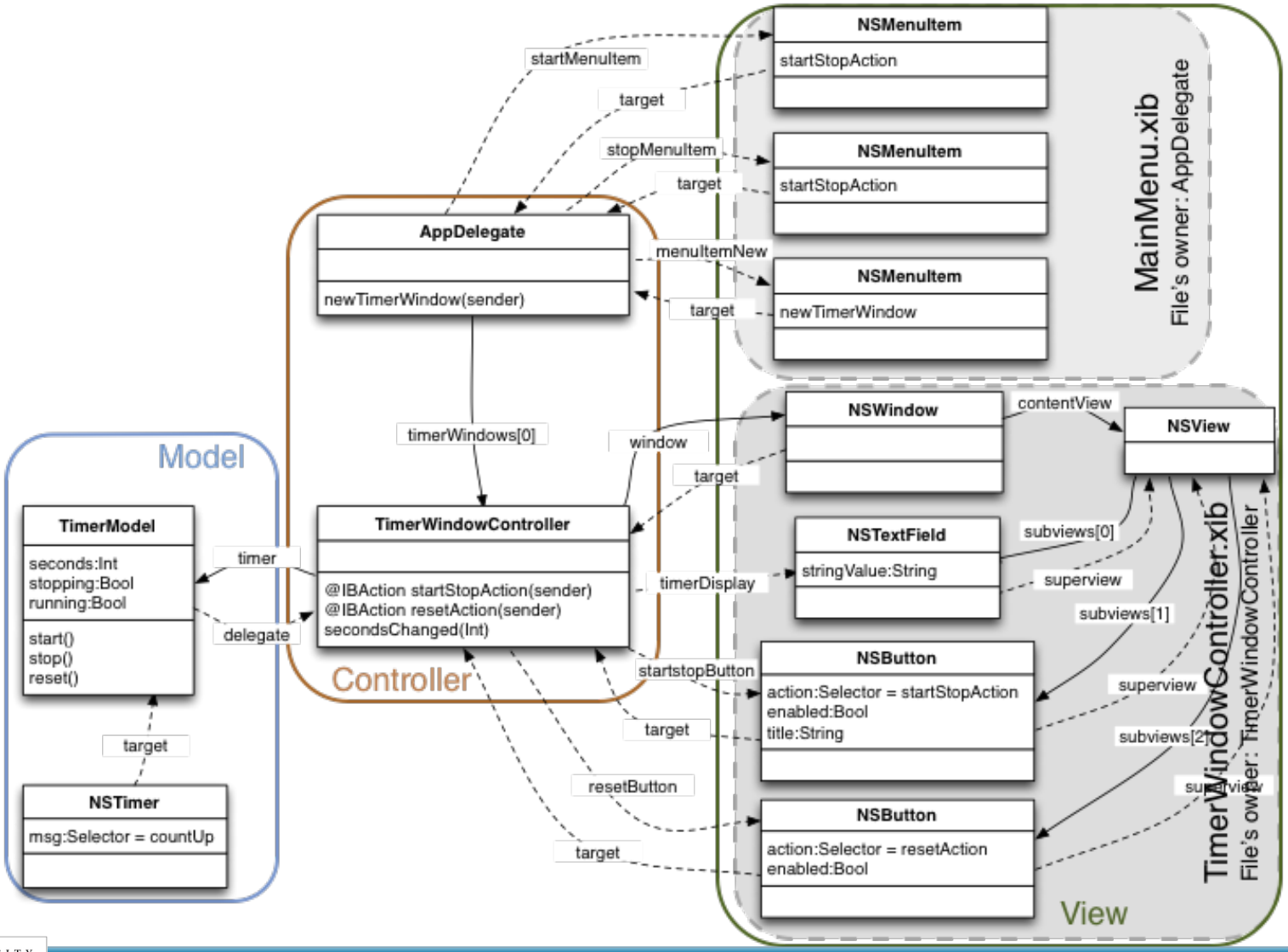
- The pre-made objects in the Cocoa's object library that inherit from NSResponder override various methods to implement specific behaviour for the type of interface they represent
- Example1:
  - NSButton (which is an NSControl), overrides mouseDown: method to execute an action of the corresponding target
  - If you extend NSButton and override mouseDown: method, you may disable the target action functionality
- Example2:
  - NSView becomeFirstResponder: returns NO by default
  - Override method in your custom view to return YES if you want the view to become a part of the responder chain

Summary

We examined how Cocoa applications process events and introduced some concepts that make dealing with events in your program easier

- NSEvent - object representing an event with all information about it
- NSResponder - an abstract class that sorts out the events it receives into more readable methods corresponding to mouse and keyboard events
- NSControl - an abstract class that contains a reference to a target and action, which can be triggered after an event
- First responder - reference to an object in a window that has focus
- First responder placeholder - object in Interface Builder with custom actions to which controls can be connected: at runtime, whenever the corresponding control event triggers an action, it gets sent to the target that happens to be the current first responder

Timer App
Multi Window



Timer App
First Responder