Portland iPhone Bootcamp

instructor: Bjorn Chambless
bjorn@builtlight.org

My approach to this boot-camp

- I've taught a number of Programming and Computer Science courses at the University level, but I've never done an intensive 24 hours in three days before.
- The choice of what to teach given such a format is an interesting problem.
- My hope is that you will come a way with a solid understanding of the fundamentals at the core of iOS development which will allow you to more easily build out your skills over the coming weeks and months.

My approach (continued)

- In an effort to keep everyone engaged I'm going to alternate between short slide presentations and short programming exercises.
- The programming exercises are meant to illustrate the concepts presented and give you experience applying them.
- Ask lots of questions and play around with the examples as we work them...then ask more questions.
- Things that don't work are often more educational than things that do. If something doesn't work and you don't understand why, ask about it.

Intro to iOS programming

The 3 things you need:

#1. An iOS device

#2. Development environment

Xcode

Objective-C

#3. Apple's Authorization

Developer registration

Certificates

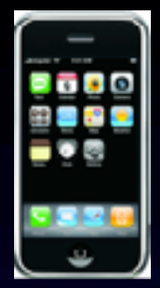
Provisioning profiles

iTunes Connect

iOS Devices

All Apple handheld devices use the same SDK

iPad



iPhone & iPod-touch

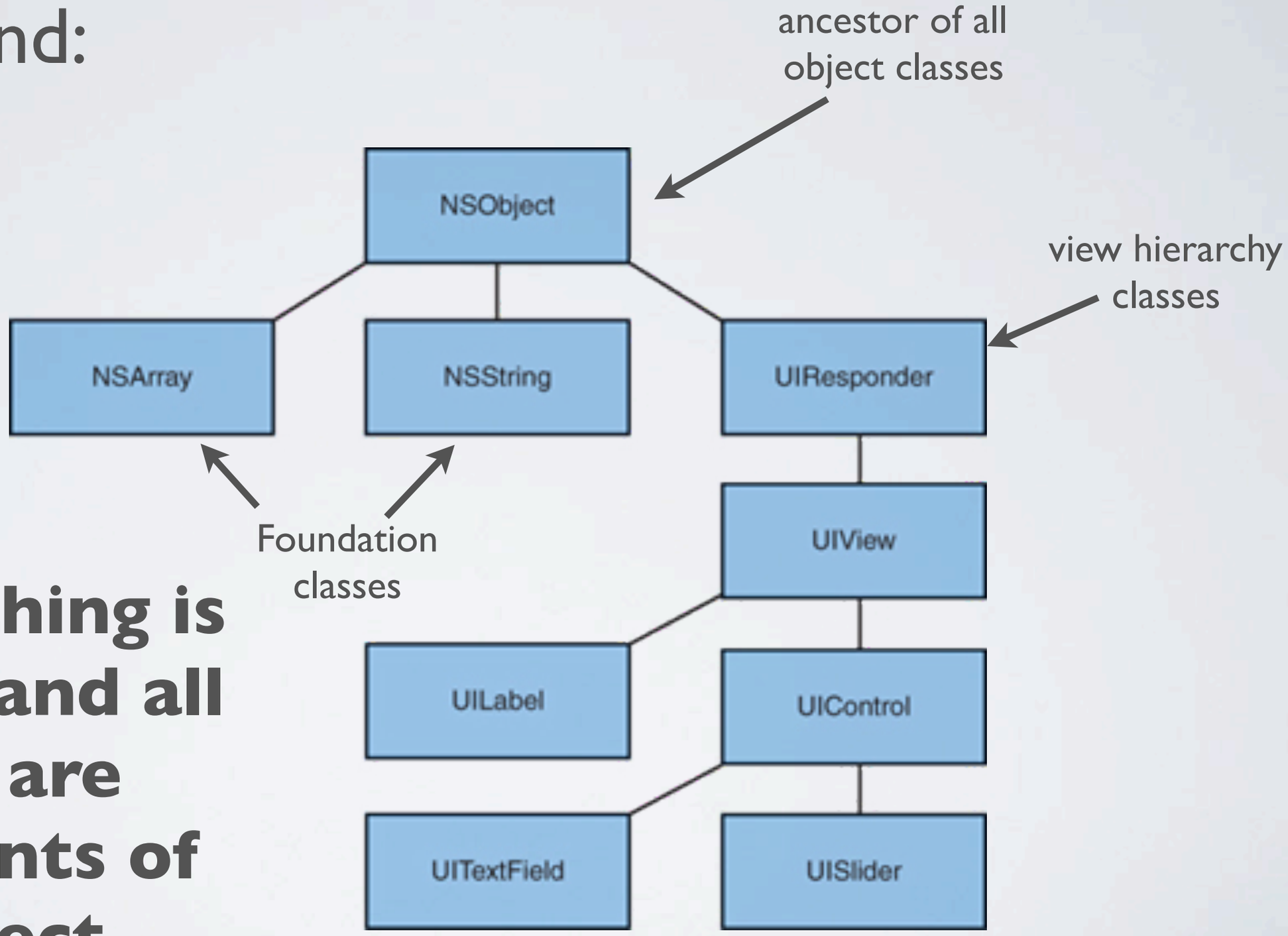
UIKit / Cocoa-touch

-Programming for the OS X desktop is similar, but not the same. Cocoa is not the same as Cocoa Touch.

-The basics language constructs(Foundation classes) are same, but as you move up the view class hierarchy, differences emerge.

For example: the basic view on iOS is UIView, while on the desktop it's NSView.

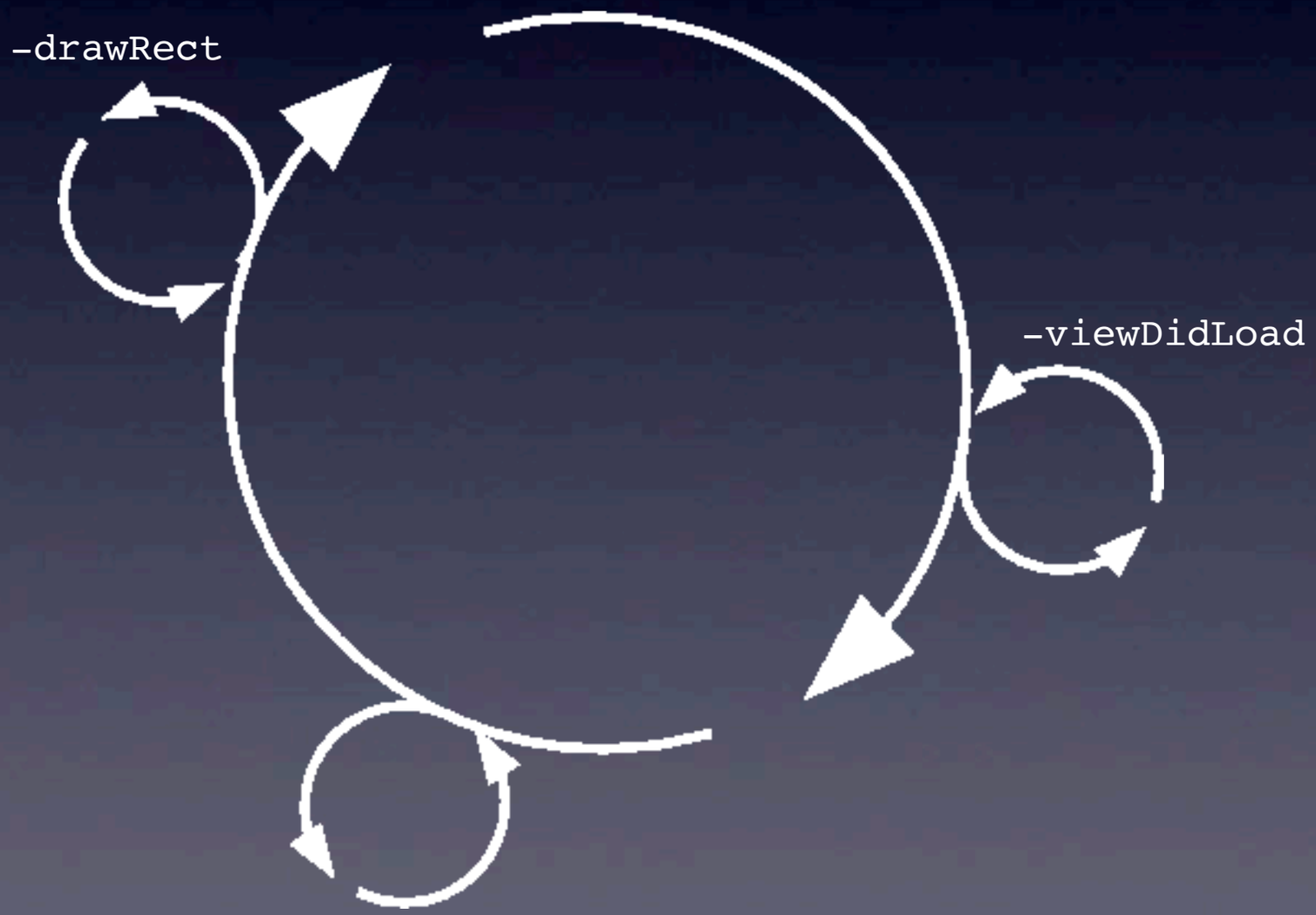
When developing for iOS there a few high-level concepts you should keep in mind:



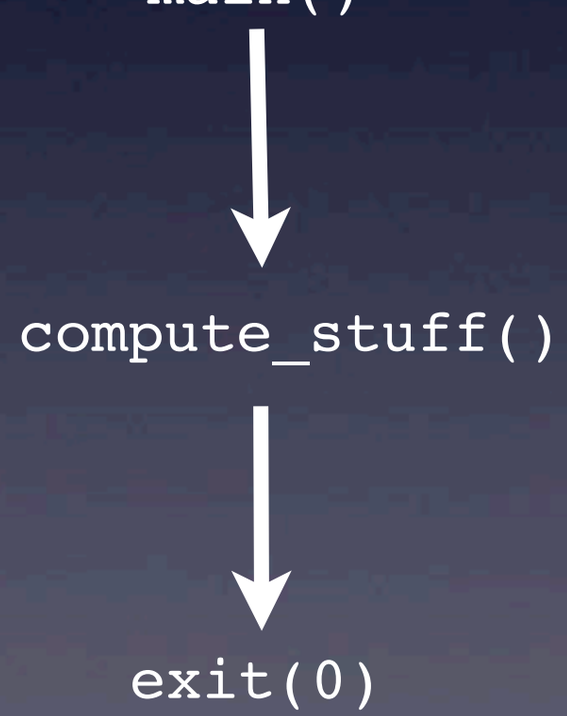
#1 - Everything is an object and all classes are descendants of NSObject

#2 The code you write will be executing within a run loop

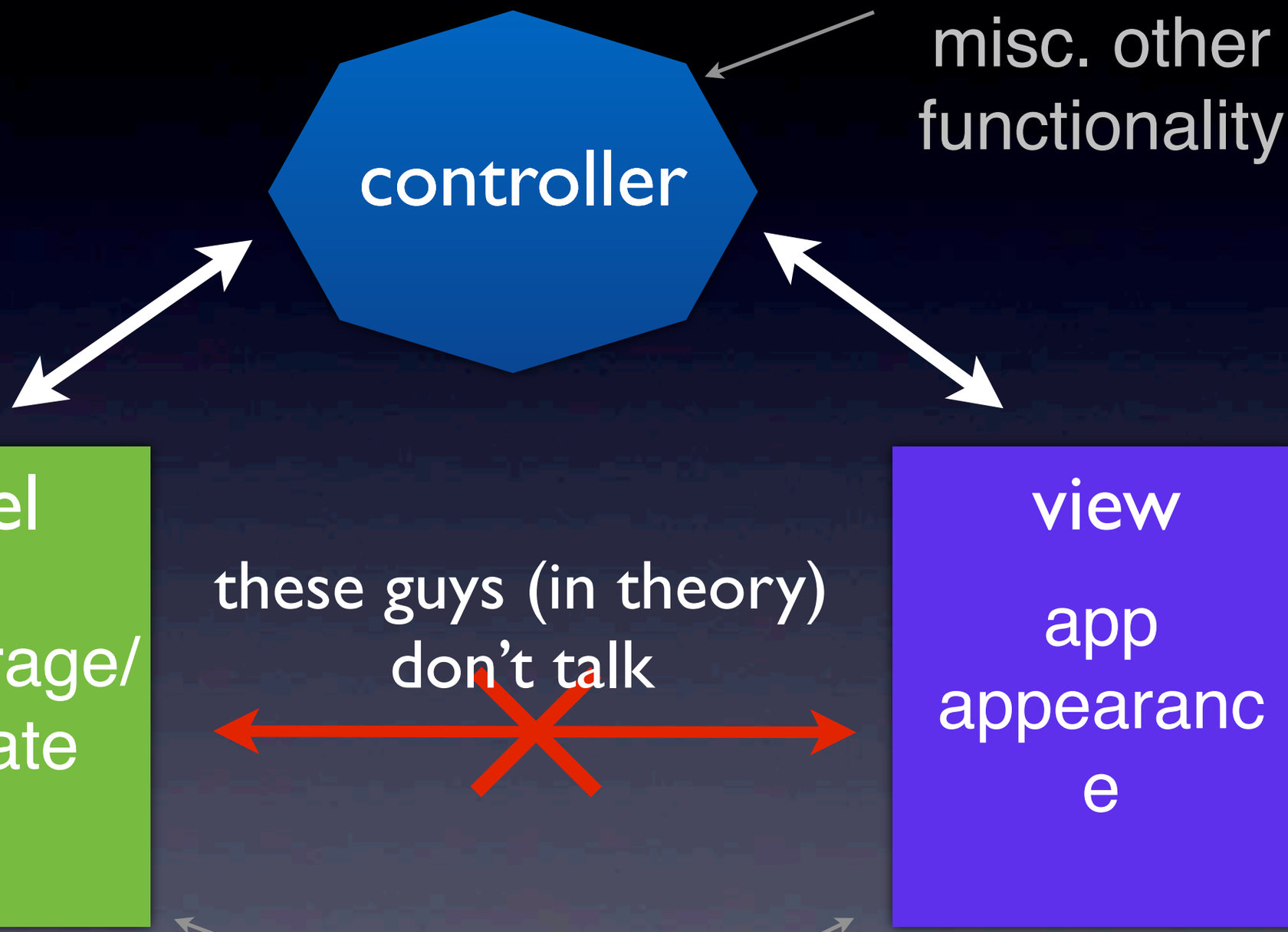
iOS Run Loop



Traditional/Theoretical Computing



#3 Model-View-Controller (MVC)



(#3 cont.) MVC is actually part of a larger theme in iOS programming: Decoupling

Decoupling is supported by a number of Objective-C patterns & features:

singleton classes

delegates responder chains

KVO notifications

} We will cover all these later

Now, a little bit of history explain some terms and put things in context...

Evolution of iOS/MacOS

1985  Macintosh launched

1986 Steve Jobs leaves Apple

starts 

selling computers ~1989

Apple becomes a sad and sickly creature

1990 - used by Tim Berners-Lee to create web server and browser

-John Carmack writes Wolfenstein3D & Doom on a NeXT machine

Jobs returns to Apple

Openstep renamed Cocoa

"Carbon" created to allow some backward compatibility

1997 Apple buys NeXT to replace MacOS (cooperative multi-tasking no longer cool)

2001 Mac OSX 10.0, first Darwin-based MacOS

NeXTSTEP

- Mach microkernel (experimental CMU technology)

- Objective-C used for development

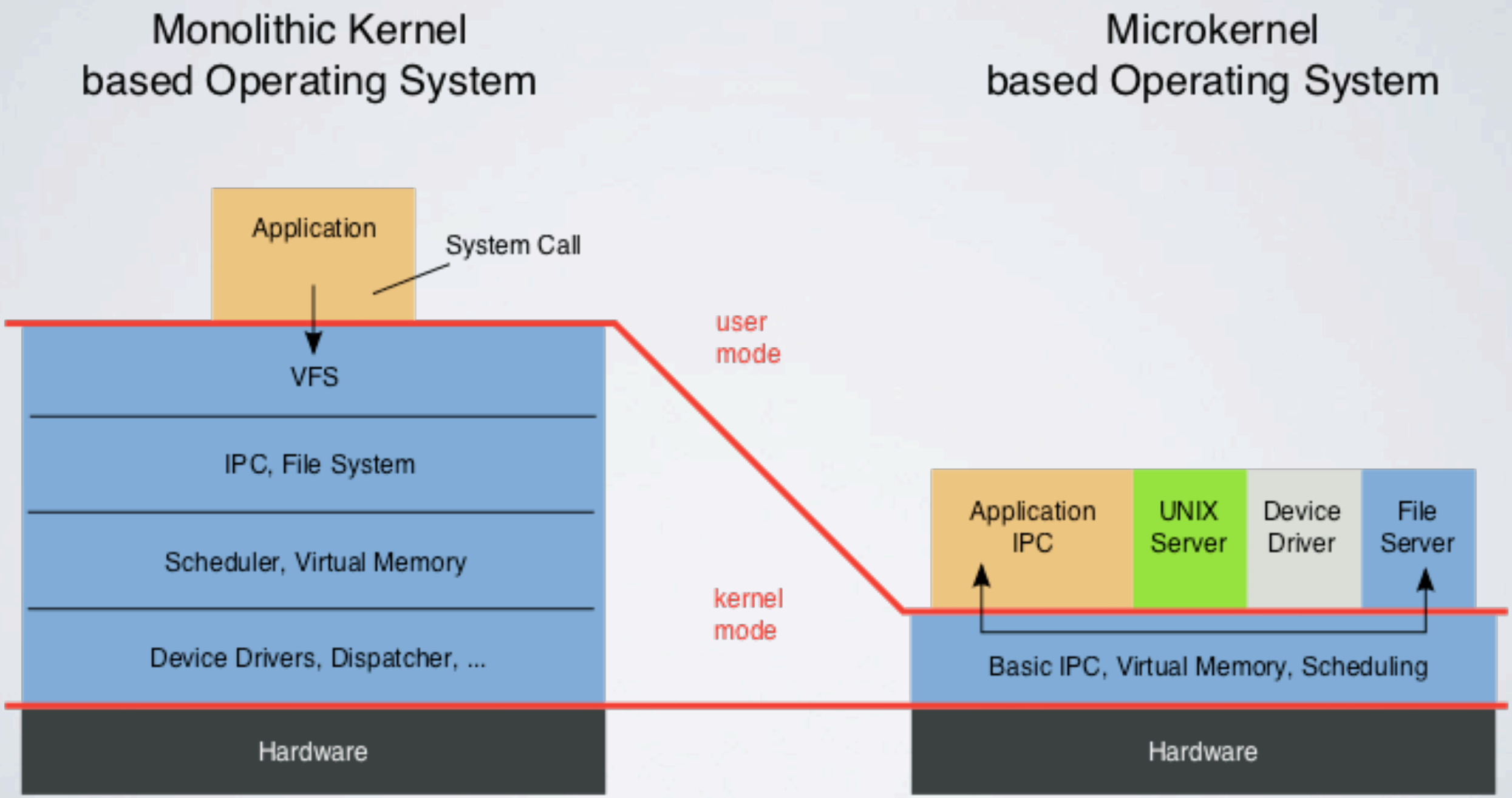
- Application Bundles

- Vector based display (Postscript)

this will influence Quartz/Core Graphics

You occasionally hear the terms “micro-kernel” and “monolithic-kernel” used when talking about an Apple OS

In a nutshell, this is what it means:

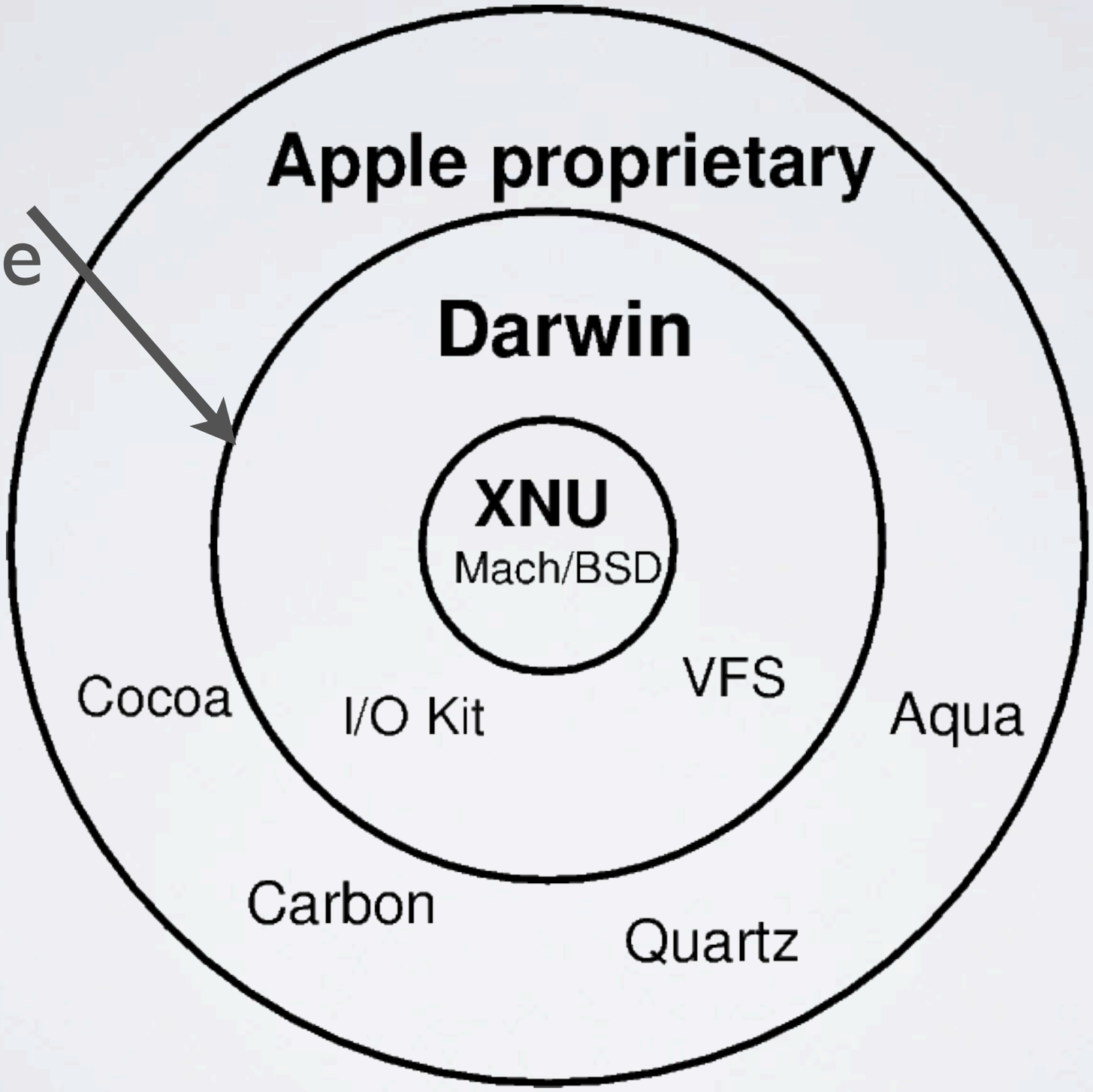


More context: The Darwin Kernel
(at the core of both OSX and iOS)

Hexley

Darwin is open-source

MacPorts
Fink
Homebrew



The Evolution of Objective-C

-Developed by Brad Cox & Tom Love in the early 80s in an effort to add features of SmallTalk to C.

-In 1988 NeXT licensed Objective-C, extended GCC to support it and developed AppKit & Foundation

AppKit		*UIKit*
NSApplication	→	*UIApplication*
NSWindow	→	*UIWindow*
NSView	→	*UIView*
NSResponder	→	*UIResponder*

MacOSX only

iOS only

Foundation

NSObject

NSString

NSValue

NSNumber

Getting into writing code...

Day 1 (today)

-Knowledge of objective-C is the most important thing you can take away from this course, so initially we are going to spend a lot of time on it.

-Xcode is a large and complex development environment with so many bells, whistles, auto-generated code and code generation options that it can initially be more distracting than helpful.

-Before we jump into Xcode, we're going to start working with Xcode's compiler: clang

-Once we have a handle on Objective-C and the Foundation classes, we'll start moving into Xcode.