G54MDP
Mobile Device Programming

Power, iOS

Rules of thumb

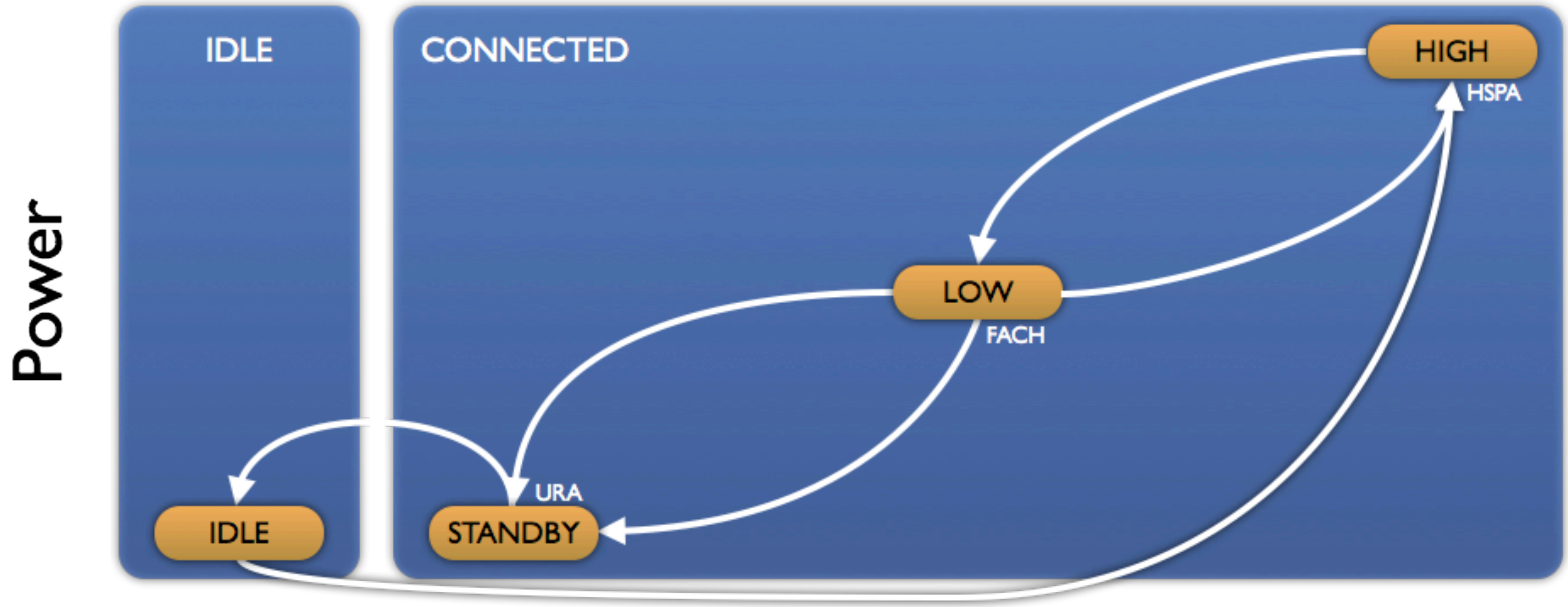
- Speed = Efficiency
  - The CPU runs at a certain rate
    - Instructions per second
  - The faster we can perform our work, the more time the CPU can idle
    - Idle at reduced power
    - More efficient use of instructions
  - The faster we can perform our work, the more quickly the CPU can go to sleep
    - Sleeping at reduced power consumption
- Waking up / Running services = Costs power
  - Assume we are not the only application in use
- Byproduct
  - A fast app feels more responsive
    - Users are less likely to use an app that is slow
    - Majority of apps are kept / uninstalled after first run

Other Chips

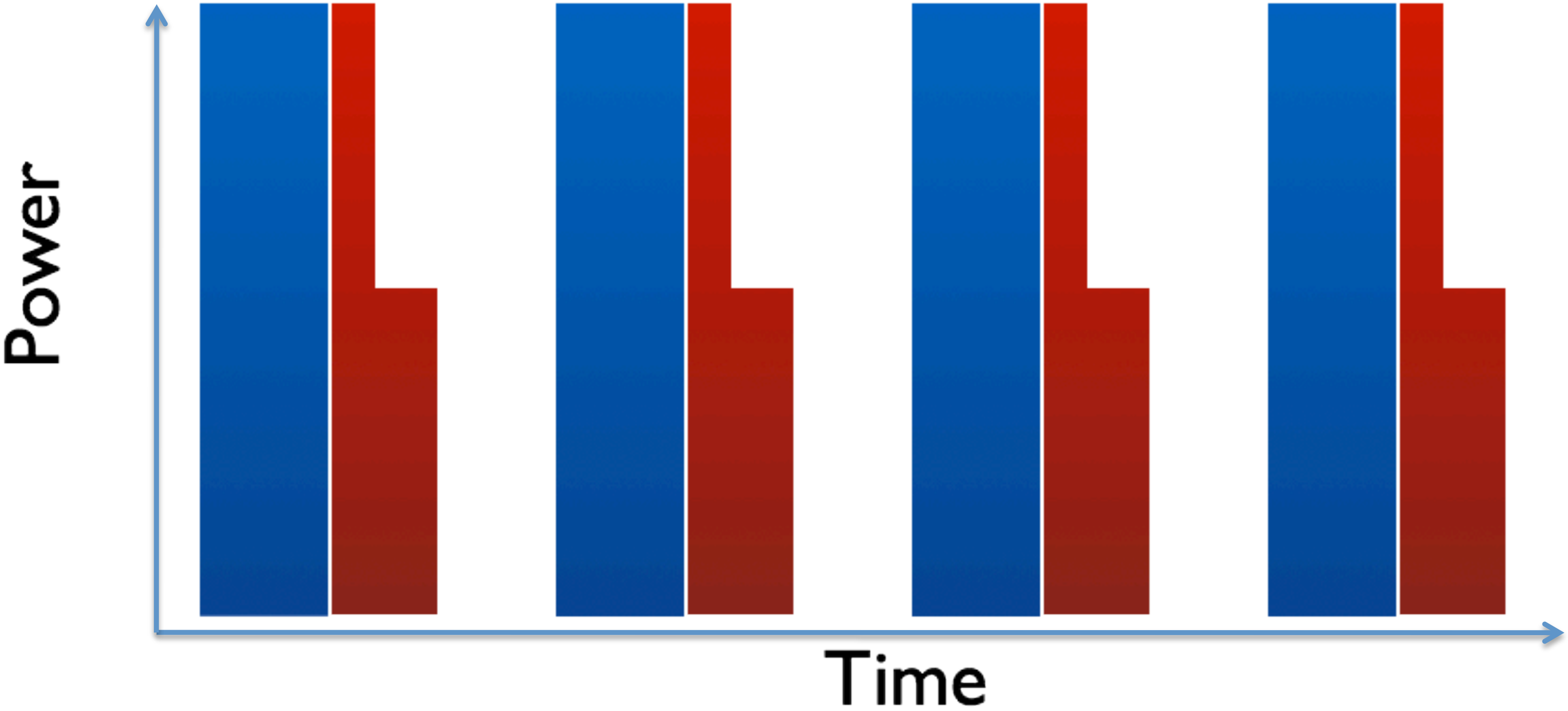
- CPU ~100mAh
- Other components use power too
  - Accelerometer
    - 10mA normal use – 80mA fastest / finest measurements
      - Choose most appropriate frequency
  - Location
    - Wifi basestations (~100m)
    - Cellphone tower triangulation (~500m – 3km)
    - GPS (~1-5m)
    - Select the most appropriate accuracy
      - GPS is very expensive in terms of battery usage, especially cold start
    - Register for updates appropriately
  - Radios (network connectivity, phone calls)

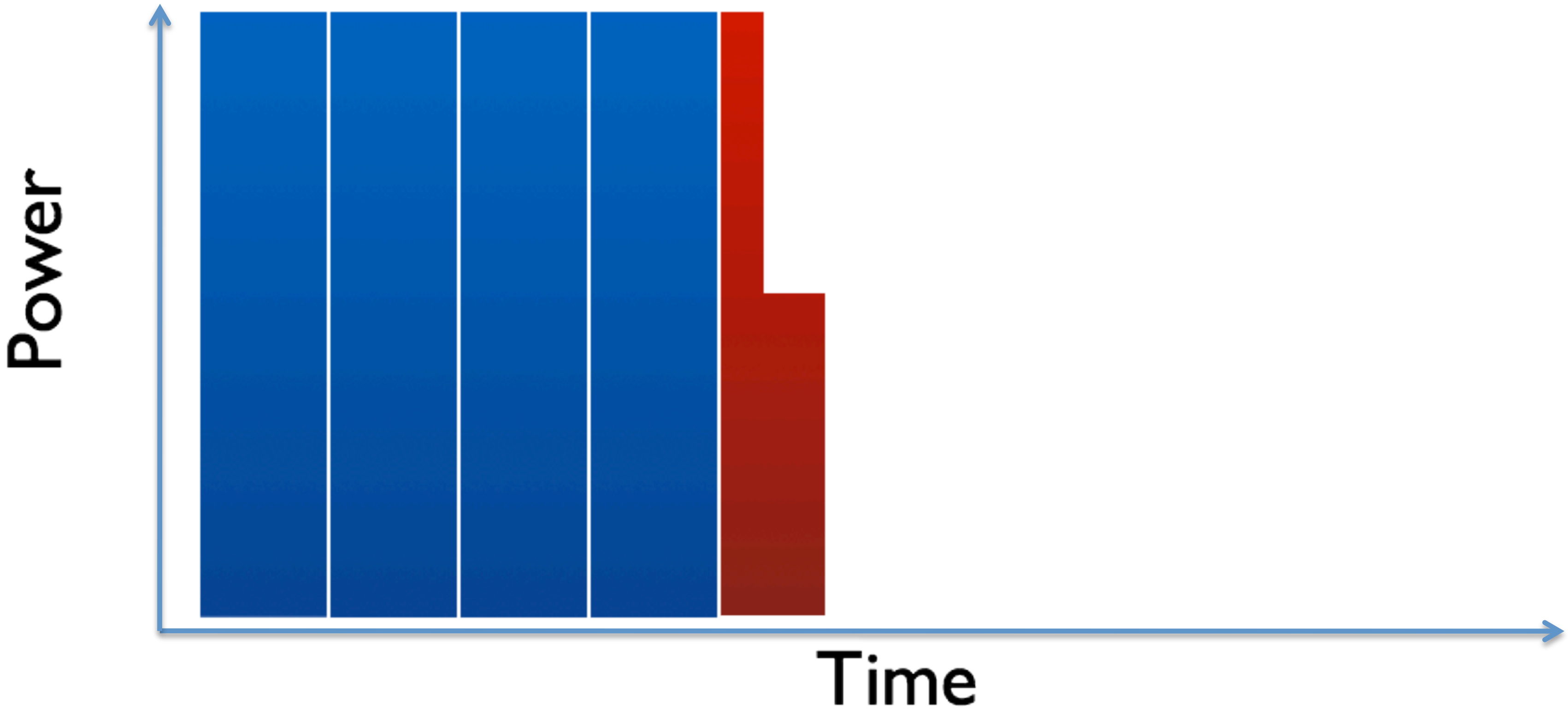
Radio / Network

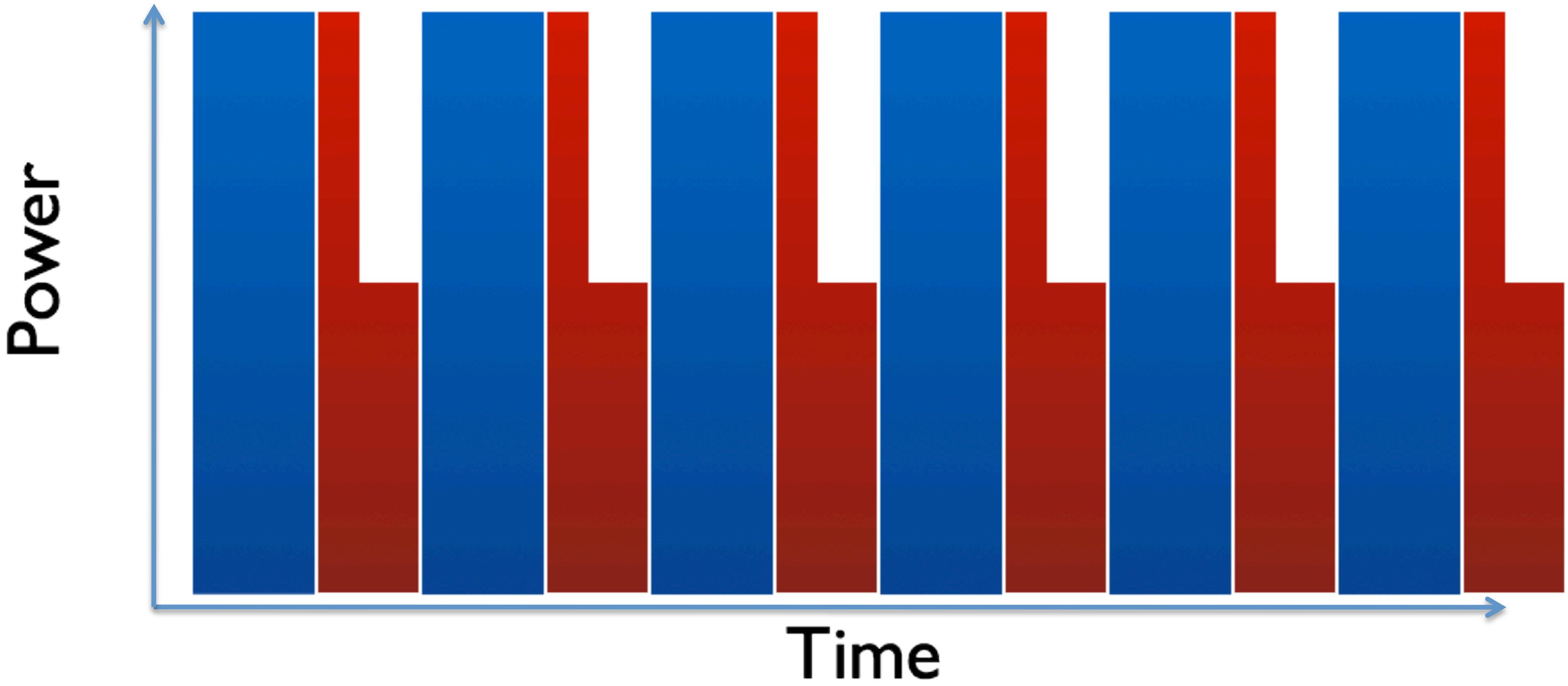
- 3G chip has a number of states
  - URA – Connected but not sending data
  - FACH – Half power, small amount of data
  - HSPA – Full power, dedicated channel
- Cost / time to transition upwards
  - Ramp up power, negotiate channel
- In high power radio state
  - Delay to transmit is shorter
  - Device stays in high state for a short period of time following communication
- Regular polling keeps the radio transition between states
  - Pay the battery cost even if we transfer nothing
    - Synchronize polling – inExactAlarms
    - Coalesce data into large chunks
    - Small transfers will only transition up to low / FACH power state (~256 – 512 bytes)
  - Be careful of reusing libraries
    - Were they designed for 3G, or do they assume Ethernet



Data rate / resources / lower latency

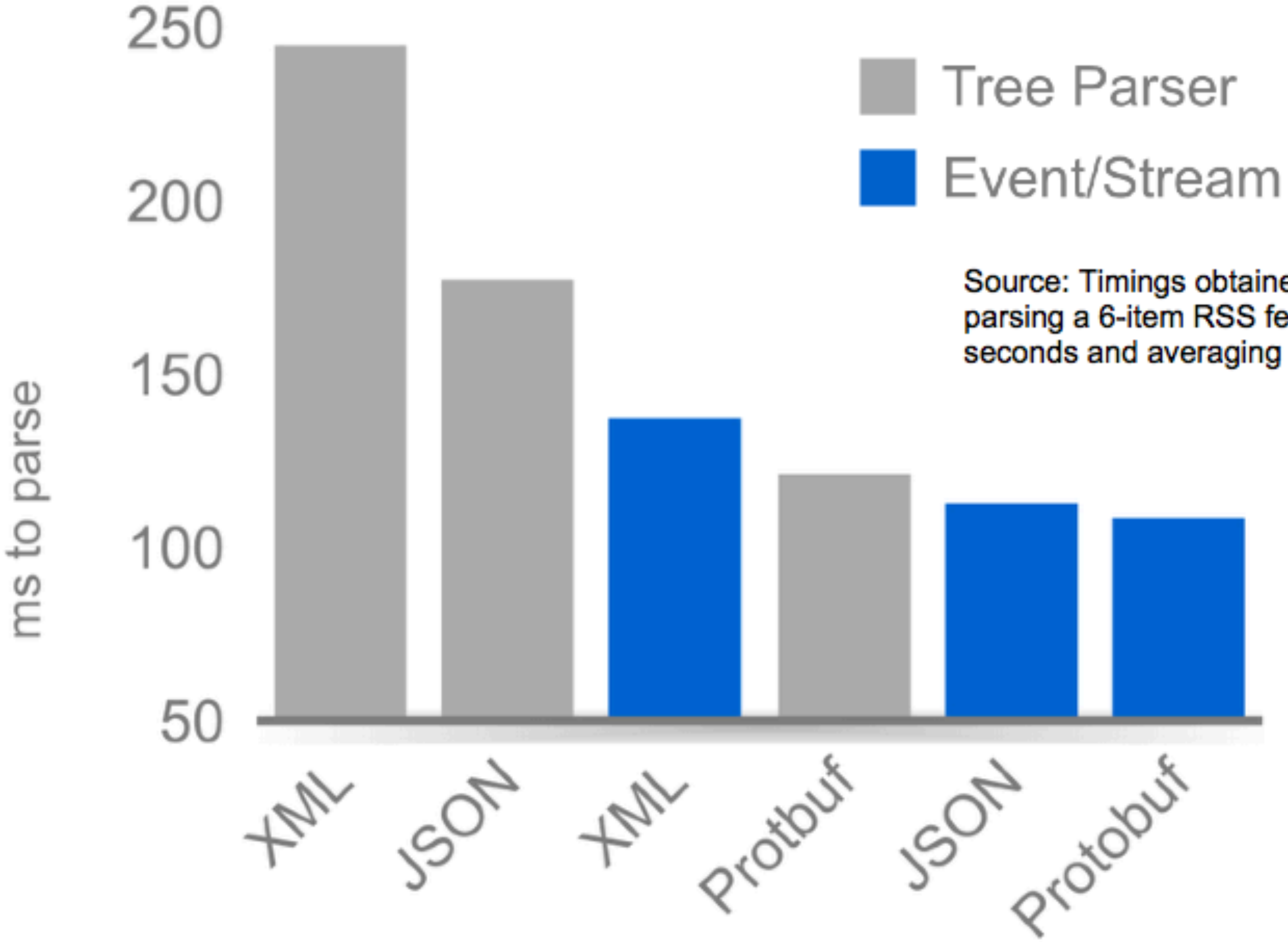

Power
Time


Power
Time


Power
Time

Data Transfer

- Battery cost per byte
  - Radio usage, CPU usage
  - Minimise the amount of data transferred
- Reduce signal-to-noise ratio
  - How much of the data describes the structure and not the data?
  - XML is bulkier than JSON
  - JSON is bulkier than binary
- Use Gzip compression where possible
  - Decompressor is native code
    - *Cost* to decompress is less than *cost* to send uncompressed
- Consider time taken to parse



Source: Timings obtained by downloading and parsing a 6-item RSS feed repeatedly for 60 seconds and averaging results.

References

- http://www.google.com/events/io/2009/sessions/CodingLifeBatteryLife.html
- http://developer.sonymobile.com/2010/08/23/android-tutorial-reducing-power-consumption-of-connected-apps/
- http://www.slideshare.net/EricssonLabs/droidcon-understanding-smartphone-traffic

iOS

- OS of the iPhone/iPad/iPod Touch
  - Originally called iPhoneOS
  - Based (heavily) on MacOS X
- App support added in v2 — 2008
- Closed Source
  - Tools, deployment, app ecosystem controlled by Apple
- Apps can only be installed from an App Store
  - Cryptographically signed
  - Apple runs iTunes App Store
  - Approves all apps available from it
  - It is possible to set up an internal to enterprise app store

iOS Development

- Needs an Intel Mac
- Enables development in an emulator
- Installation / device deployment requires a developer licence
  - $99/year
- XCode is the primary development environment for iOS
  - IDE supporting both code, and interface development
  - Encourages visual ways of linking code to UI
  - Long history back to NeXTStep
- Visual coding
  - Link methods to events
  - Define object variables
  - Storyboarding

iOS Apps

- Written in Objective-C (ObjC)
  - Using the Cocoa Touch UI framework
  - Can also use C/C++ libraries
  - Compiles to native code
    - Not interpreted/JITted as on Android
- iOS uses Objective-C as its main language
  - Extension of C to add support for OO
  - Developed around the same time as C++

Objective C

- Smalltalk heritage means it is very OO
  - Uses features perhaps unfamiliar to Java/C++ users
    - Message passing
    - Categories
    - Protocols
- ObjC's syntax is probably the biggest stumbling block
  - Originally implemented via a preprocessor to a C compiler
  - The syntax designed not to clash with C

Objective C

- Class definition split into
  - header file (.h)
  - source file (.m)
- Header file contains the interface definition and member variables
  - Cf class declaration
- Source file contains the implementation

ObjC Class Interface

```
@interface classname : superclassname
{
   int mVariable;
}
+ (void)classMethod1;
+ (int)classMethod2:(int)varName1;
- (void)instanceMethod1:
   (int)varName1(int)varName2;
@end
```

ObjC Messages

- Programming based on **message passing** between objects
  - Rather than calling methods directly
  - Send messages to an object to call a method
- Target is resolved at runtime
  - Not compile time
  - Receiving object interprets the message
- An object is not guaranteed to respond to a message
  - Raises an exception
  - Send messages to a collection of objects
    - Only some may be expected to respond
  - Objects do not have to be defined at compile time
  - Can **forward** messages to other objects
    - Delegation

ObjC Categories

- Adding methods to a class at runtime
  - Without the need to recompile / access to source code
    - Cf "Monkey patching" in Ruby, but by design
  - Define a **category** that specifies new methods to add to an existing class
- Replace existing methods
- Add new functionality
  - E.g. add a spellchecker to a TextEdit component

ObjC Protocols

- Multiple-Inheritance
  - Via specification rather than implementation
- Informal
  - Ad-hoc, specified via documentation
  - A list of methods that a class can opt to implement
    - If implemented, change the behaviour of the class in the specified manner
  - E.g. TextEdit inspects a delegate for an auto-complete method, calls the method if it is available
- Formal
  - Similar to interfaces in Java
  - Compiler ensures a class implements all methods specified in the protocol
    - Or detectable at runtime

ObjC Message Declaration

-(void)buttonClick:(id)sender atPoint:(NSPoint)point;

- Types are placed in brackets before the parameter
- Return value is at the beginning
- Objects are always pointers (e.g. DAPageView*) or the generic id (also a pointer)
- Parameters are always explictly named
  - Even when calling
- Message name includes the name of all parameters
  - So this message would be called buttonClick:atPoint:
  - Means source code is very readable

ObjC Message Sending

- ObjcC’s message dispatch is probably the oddest part
  - Lots of square brackets
- To call the method buttonClick

```
[anObject buttonClick:self
   atPoint:NSMakePoint(100.0, 100.0)];
```

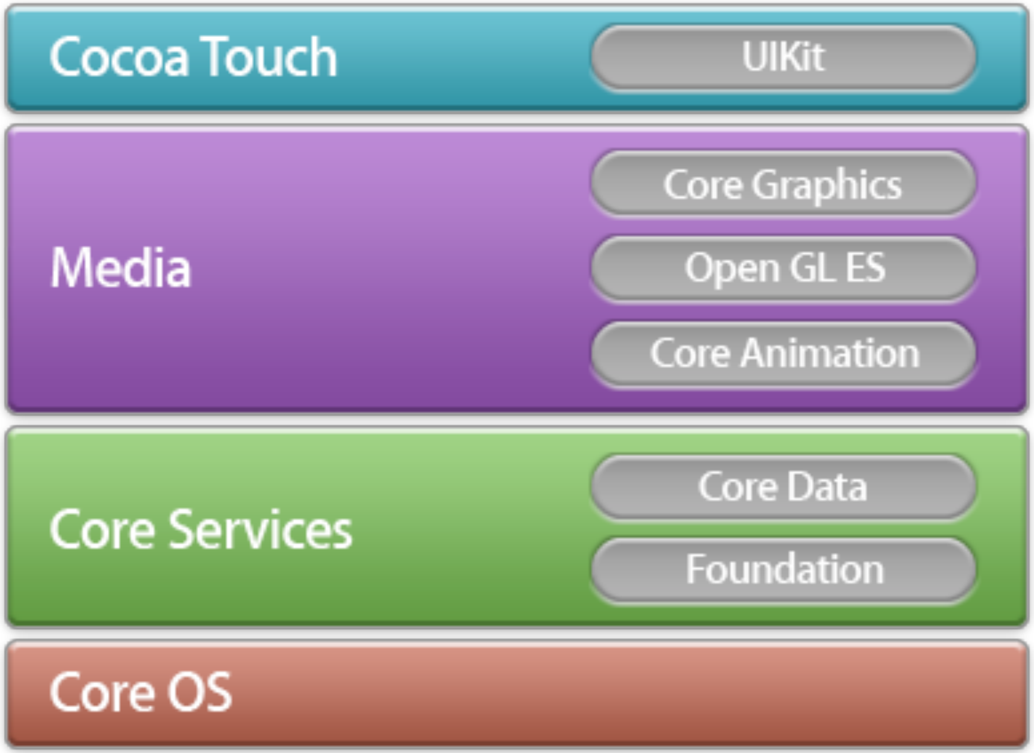
iOS Memory Management

- No garbage collection in iOS
- ObjC uses reference counting
  - Send retain message whenever you copy a pointer
  - Send release when the pointer goes out of scope
  - Object destroys itself when nothing points at it
- Memory management is fiddly
  - Can lead to strange crashes
  - Some support for automatic reference counting in the compiler

iOS Frameworks

- iOS comes with several frameworks that can help us with development
  - Foundation framework provides support for strings, files, collections etc
  - Other Frameworks provide support for Audio, video, animation, location etc
  - At the top is the UI framework, CocoaTouch
    - Widgets, buttons, views
- iOS is very much an evolution of PC GUI programming into the mobile space
  - Particularly MacOS X GUI programming
  - Almost every class in CocoaTouch has an equivalent in OS X
  - Vs Android major components

iOS Frameworks

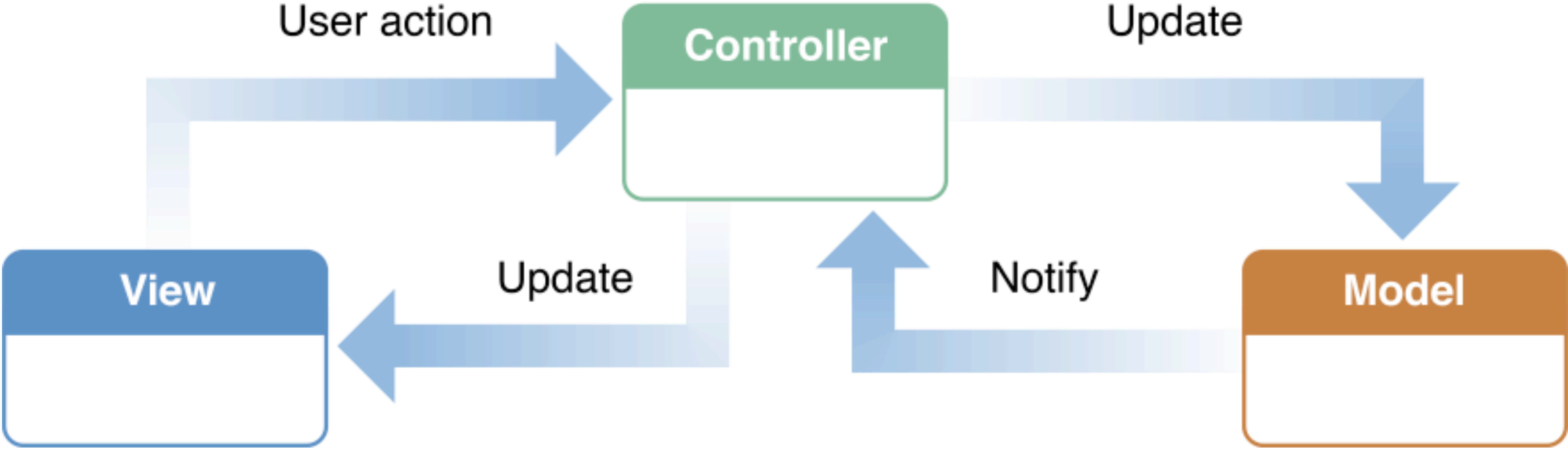


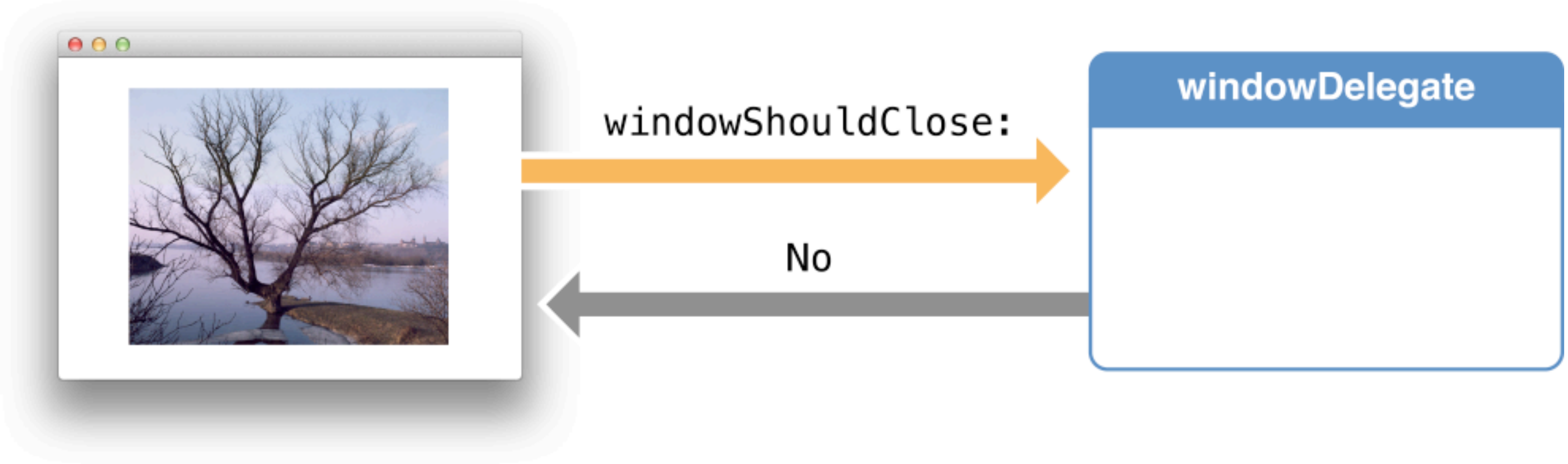
Design Patterns

- iOS / Cocoa framework strongly suggest use of certain design patterns
  - **Model View Controller**
  - Delegation
  - Protocols
  - Notification
  - Target-Action
  - Key-Value Observation

Model View Controller

- Divide objects into three types
- Model
  - What the application **is**, but not how it is displayed
  - A contact in an address book
- Controller
  - **How** the model is presented to and manipulated by the user
  - Add / read / modify a contact
- View
  - Drawing things on the screen
  - Render a text view containing the contact
- MVC design pattern determines how these components should communicate
  - The model and view are typically decoupled

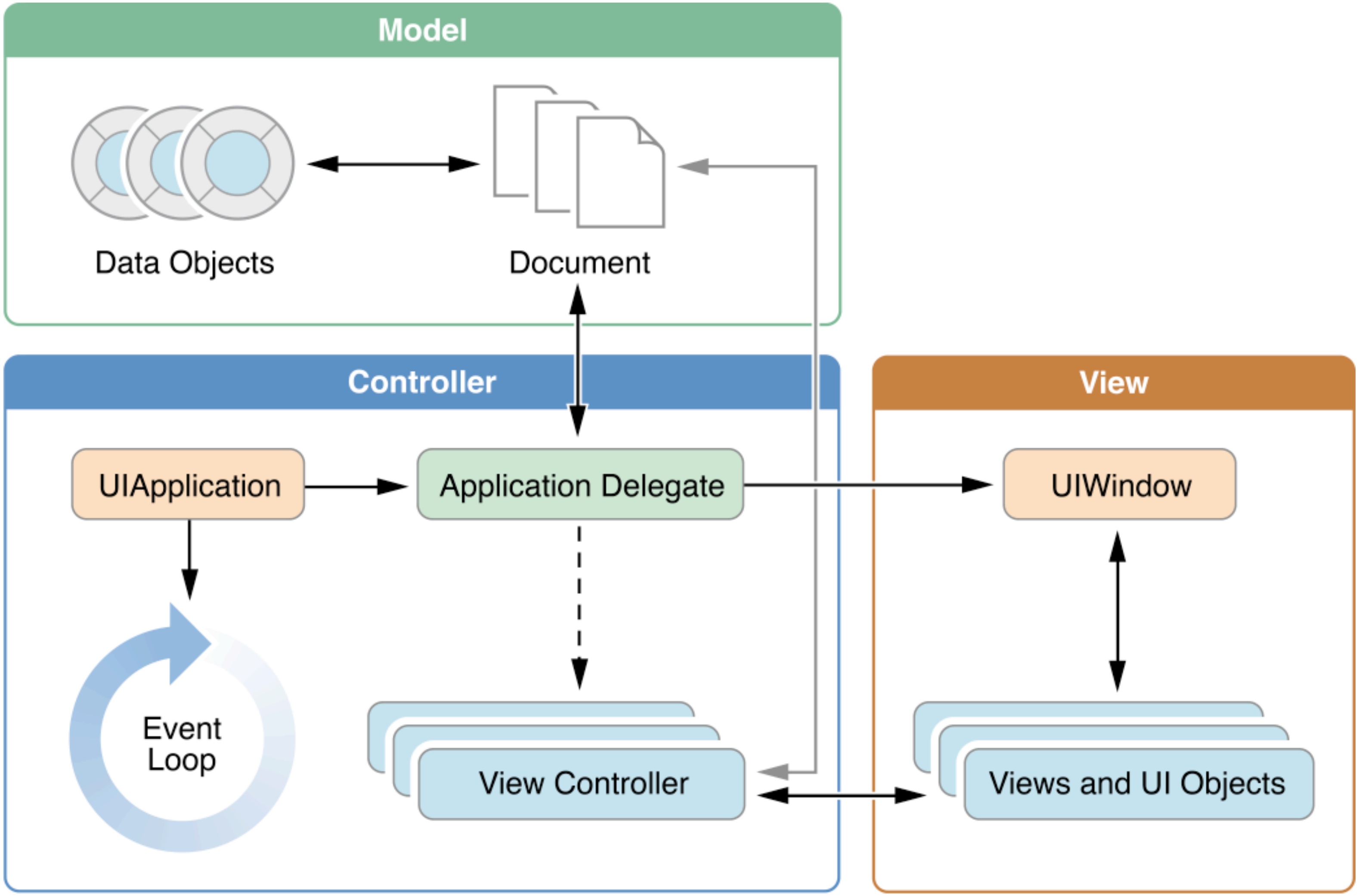

User action
Controller
Update
View
Update
Notify
Model



windowShouldClose:

No

windowDelegate



Model

Data Objects

Document

Controller

UIApplication

Application Delegate

Event Loop

View Controller

View

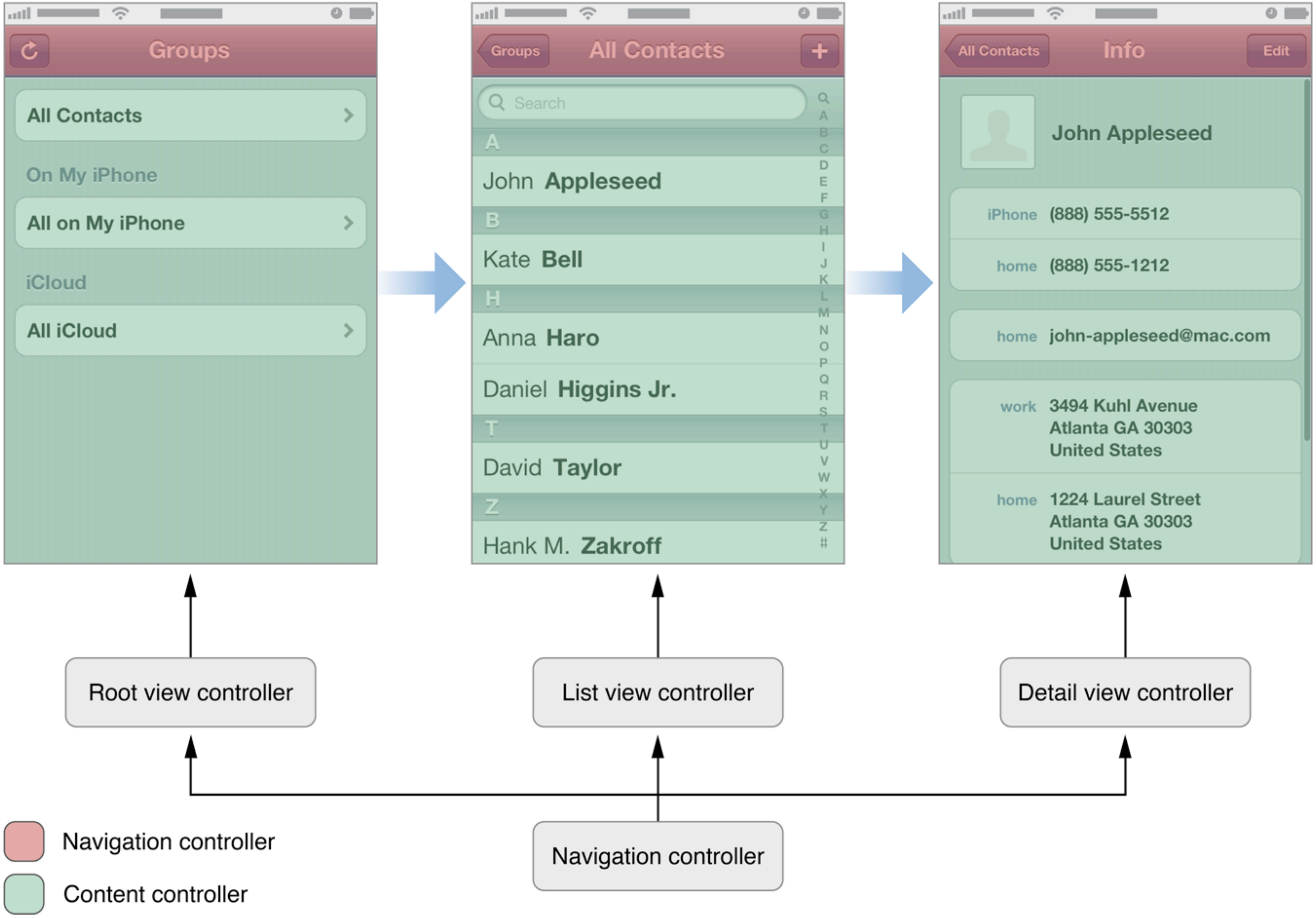
UIWindow

Views and UI Objects

🟩 Custom Objects

🟧 System Objects

🟦 Either system or custom objects



Root view controller

List view controller

Detail view controller

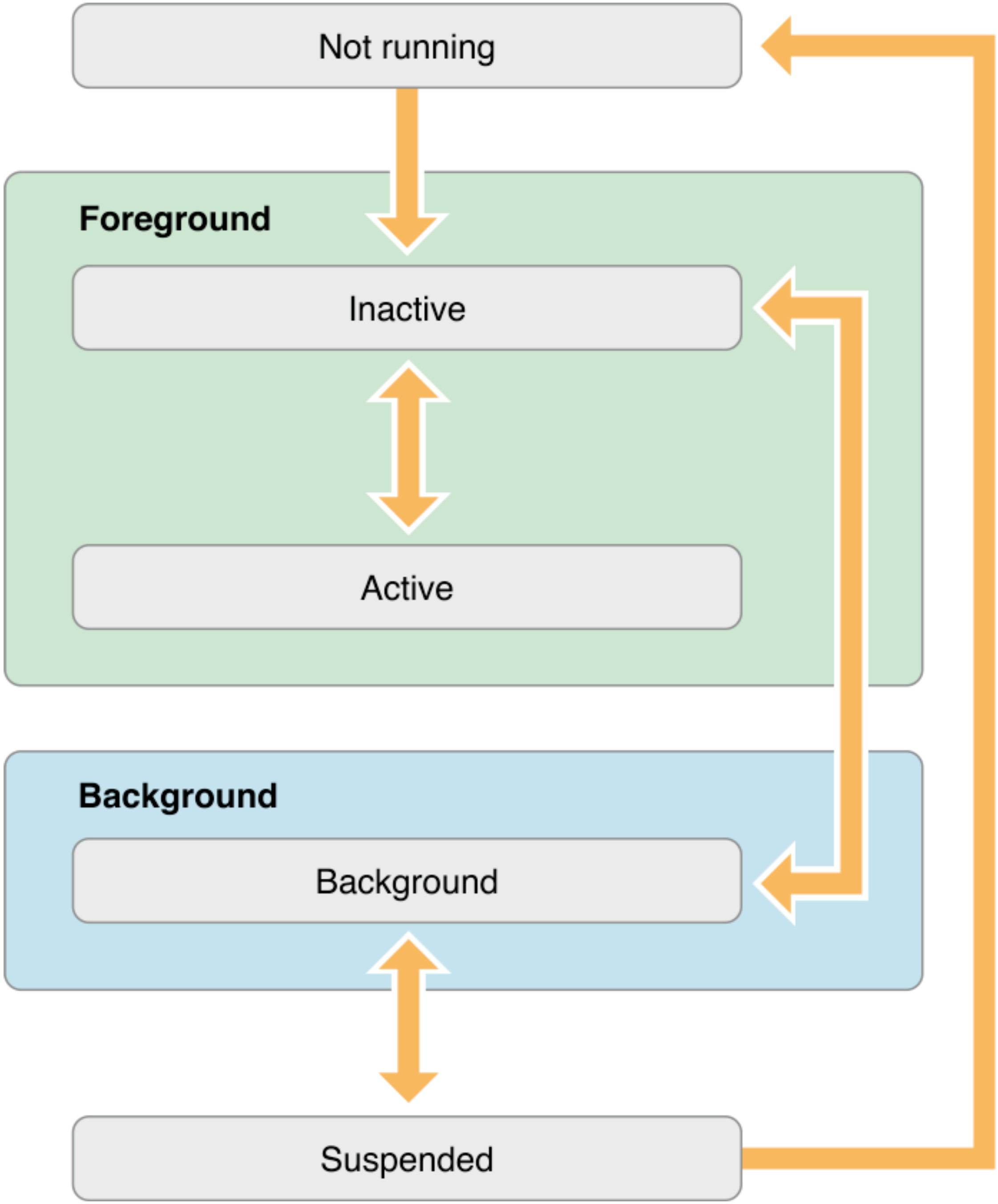
Navigation controller

🟥 Navigation controller

🟩 Content controller

iOS Lifecycle

- Analogous to Android lifecycle
  - Only one application in the foreground / visible at any one time
- A main loop processes events for the application
- An app can be a number of significant states
  - Active – foreground
  - Inactive – foreground but interrupted
    - By a phonecall, notification etc
  - Background – can remain in this state to perform long running tasks
    - Analogous to Services
  - Suspended
    - Main loop no longer running, potentially killed by the operating system
- iOS 3.2 and earlier
  - No support for suspended / background states
    - No long running tasks


Not running
Foreground
Inactive
Active
Background
Background
Suspended

App Store

- “We will reject Apps for any content or behavior that we believe is over the line. What line, you ask? Well, as a Supreme Court Justice once said, "I'll know it when I see it". And we think that you will also know it when you cross it.”
- Pre-moderation
  - Apple approves all applications in advance
  - Vs Android – publish then revoke
- A long list of guidelines as to what is appropriate
  - Correct use of interface components
  - Substantial content

App Store Restrictions

- **2.5** Apps that use non-public APIs will be rejected
- **2.8** Apps that install or launch other executable code will be rejected
- **2.10** iPhone Apps must also run on iPad without modification, at iPhone resolution, and at 2X iPhone 3GS resolution
- **2.16** Multitasking Apps may only use background services for their intended purposes: VoIP, audio playback, location, task completion, local notifications, etc.
- **2.17** Apps that browse the web must use the iOS WebKit framework and WebKit Javascript
- **13.2** Apps that rapidly drain the device's battery or generate excessive heat will be rejected

References

- http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html
- http://developer.apple.com/library/ios/#referencelibrary/GettingStarted/RoadMapiOS/chapters/DesignPatterns.html
- https://developer.apple.com/appstore/resources/approval/guidelines.html