AppDelegate
vs.
NSNotificationCenter

What they are, and some Pros and Cons

Kenji Hollis
March 03, 2009
WillowTree Consulting Group

What is a Delegate?

- An object that acts on events
- Generally named "ClassXYZDelegate"
- Allows objects to manipulate one-another without the need for inheritance
- Used by most Cocoa Touch classes
- Should be implemented using formal protocols
- Think of them as "callbacks", most of which are optional.

Target-Action

- Target: the delegate class that implements a method to call when an event is performed.
- Action: the method that responds to the target event.
- Used by UIKit. Example, UIAlertView: set the delegate (target), implement dismissal delegate and respond to the request (action).

Quick "How-To"

In your accessor class header:

Class .h file:

```
@interface Class {
    id parentDelegate;
}

- (id)delegate;
- (void)setDelegate:(id)newDelegate;
```

Quick "How-To"

Implement the code in the accessor class:

Class .m file:

```
- (id)delegate {
    return parentDelegate;
}

- (void)setDelegate:(id)newDelegate {
    parentDelegate = newDelegate;
}
```

Quick "How-To"

Calling a delegate function from the accessor:

Class .m file:

```
- (void)performEvent {
    if ([parentDelegate respondsToSelector:@selector(myEvent)]) {
        [parentDelegate myEvent];
    }
}
```

We use "respondsToSelector" to verify that the selector exists in the implementing delegate class. Note the "blind call" to the "myEvent" function.

Quick "How-To"

Alternate method:

Class .m file:

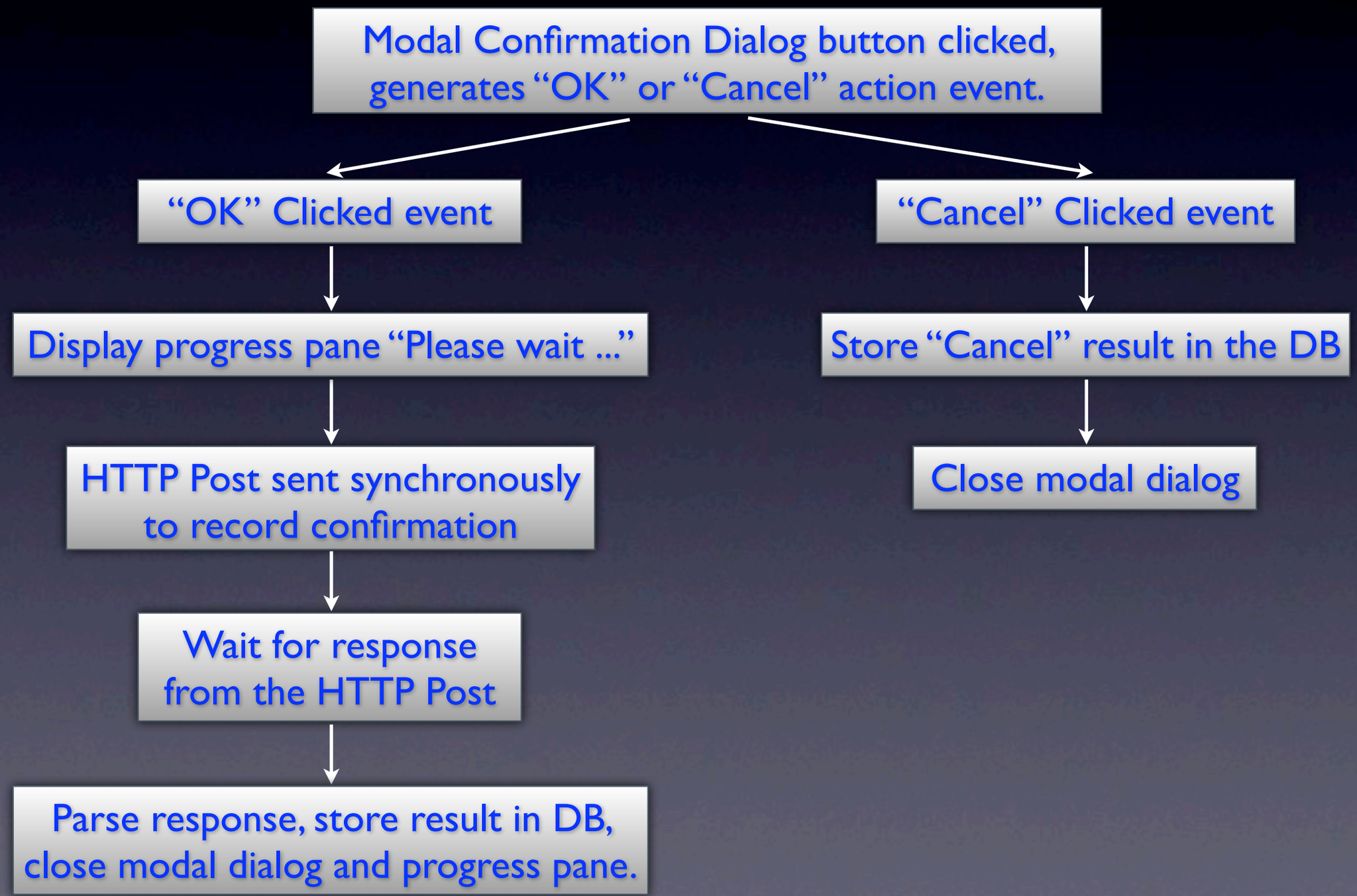
```
- (void)performEvent {
    if ([parentDelegate respondsToSelector:@selector(myEvent)]) {
        [parentDelegate performSelector:@selector(myEvent)];
    }
}
```

We are calling the selector by using "performSelector" instead of blindly calling the member. Here, "myEvent" is the action, "parentDelegate" is the target.

Delegate Use:

- A delegate is called after tapping a line in a UITableView, or tapping a button in a UIAlertView.
- Telling the application to switch to a different tab view.
- Responding to an application interruption.
- Running an operation that may block the application until complete (with a “Please wait” box)

An example delegation



What is an AppDelegate?

- Serves as the root controller of your application.
- Contains methods to handle application start, shutdown, and interruption.
- Contains the setup code for your application's main window.
- Generally knows how to control your application from the top level (ie. change tab bar views, save prefs, etc.)

AppDelegate Example

Delegate functions in an example AppDelegate:

```
- (void)applicationDidFinishLaunching:(UIApplication *)app {
    // ... Set up your code here ...
}

- (void)function_1 {
    // ... Do something in the app view controller ...
}

- (void)function_2 {
    // ... Manipulate the tab controller here ...
}

- (void)function_3 {
    // ... Miscellaneous call here ...
}

- (void)function_4;
- (void)function_5;

// ... and on ... and on ... and on...
```

Calling the AppDelegate members:

- First, get access to the AppDelegate by grabbing the

  [[UIApplication sharedApplication] delegate]

  object.
- Cast that to the application object.
- Call the delegate methods where appropriate.

Call the AppDelegate members:

```
- (void)alertView:(UIAlertView *)alertView
      didDismissWithButtonIndex:(NSInteger)button {

    // Retrieve the application delegate
    MyApplication *myApp = (MyApplication *)
                           [[UIApplication sharedApplication]
                           delegate];

    // Call the application delegate method.
    [myApp function_1];

}
```

Delegate Pros

- Function declarations must be strictly defined, or they will not be performed
- Delegate functions are almost always guaranteed to fire when the selector is called
- Delegates can be reused as often as the programmer sees fit, from several classes
- Often used in conjunction with modal operations, so they're easier to debug
- Can be called asynchronously.

Delegate Cons

- If a delegate method is not defined properly, it will not be performed, or it may crash the app if called blindly.
- If a delegate method signature changes, it has to be changed in all classes that call it.
- Because most functions are called synchronously, they can tie up the app if expensive operations are performed.

AppDelegate Pros:

- Provide direct access to the application's view controllers and layout control.
- Provide simple, direct access to the root application.
- Controls how the application responds to interruptions, startup, and shutdown. This is provided to you by the UIApplicationDelegate protocol.

AppDelegate Overuse:

- AppDelegates tend to get filled up with too many member functions, so the AppDelegate turns into a giant mess.
- People often use the AppDelegate to get access to their view controller functions.
- Miscellaneous functions are often placed in the AppDelegate when they could be otherwise placed in a utility class.

How I've used Delegates

- Controlled application flow by switching tab views at the application top-level.
- Swapped views when the application changes from portrait to landscape mode.
- Initialized the application by copying a read-only database into a read/write directory.
- Preloaded data from a remote web call on application first load.
- The list goes on.

What is a Notification?

- A queued message with a payload
- Sent application-wide, not to a specific class
- Posted “Immediately”, asynchronously, or when convenient to the OS
- A single queued message can be received and processed by multiple observers
- Can be sent with no registered observers, and your application won’t crash!

Setting up the observer

This class receives an event message:

Class .m file registering the observer:

```
- (id)init ... {
      [[NSNotificationCenter defaultCenter]
        addObserver:self
        selector:@selector(selector1:)
        name:@”MY_SELECTOR_1”
        object:nil];
}

- (void)selector1:(NSNotification *)notification {
    NSObject *obj = [notification object];

    if ([obj isKindOfClass:[NSDictionary class]]) {
        // ... Do dictionary payload functionality here.
    } else {
        // ... Perform non-object payload functionality here.
    }
}
```

Sending the notification

This function sends the notification message:

Class .m file sending the notification:

```
- (void)myNotifier {
      [[NSNotificationCenter defaultCenter]
          postNotificationName:@”MY_SELECTOR_1”];

      NSDictionary *myDictionary = [NSDictionary
          dictionaryWithContentsOfFile:@”myFile.plist”];

      [[NSNotificationCenter defaultCenter]
          postNotificationName:@”MY_SELECTOR_1”
          object:myDictionary];
}
```

Don’t forget to deregister!

Class .m file registering the observer:

```
- (void)dealloc {
    [super dealloc];

    [[NSNotificationCenter defaultCenter]
      removeObserver:self
      name:@”MY_SELECTOR_1”
      object:nil];
}
```

Note: If you don’t de-register the observer, the class will receive multiple copies of the notification.

Example Notifications

- Sending an application-wide status change or refresh message
- Notifying the application when a network change occurs
- When network data or a packet is received and needs to be processed asynchronously
- Sending a message to more than one listener at a time

Notification Flow

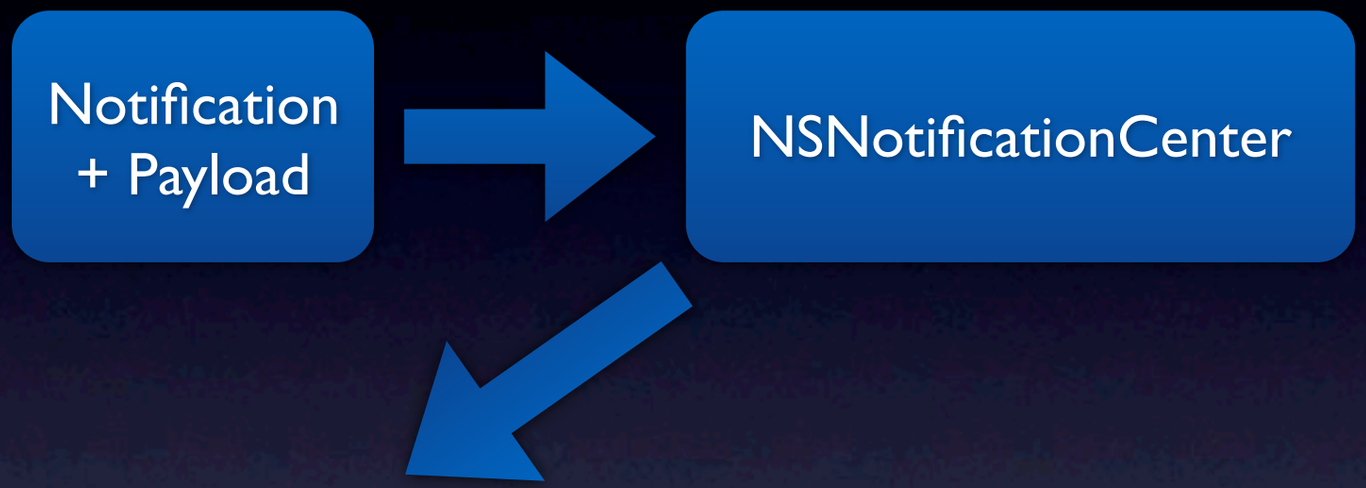
Notification Flow

Notification
+ Payload

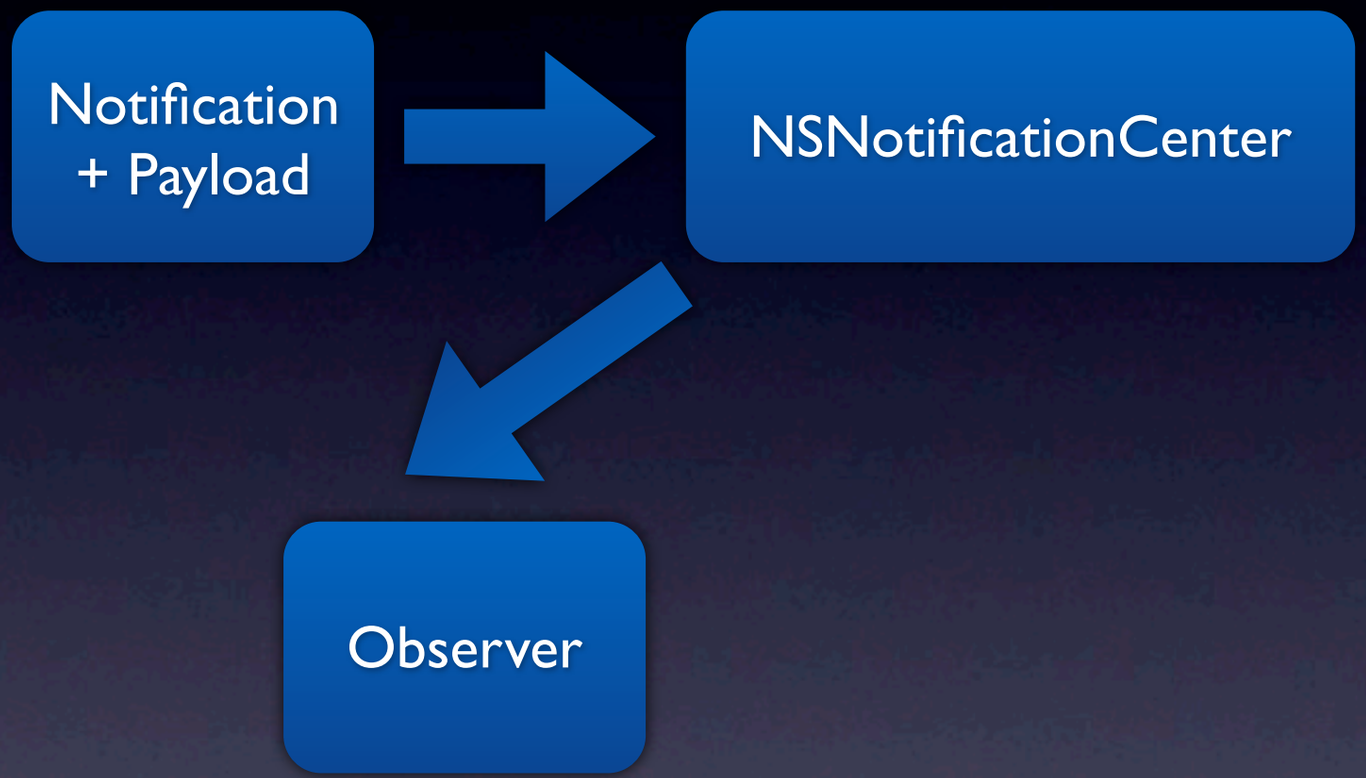
Notification Flow

Notification Flow

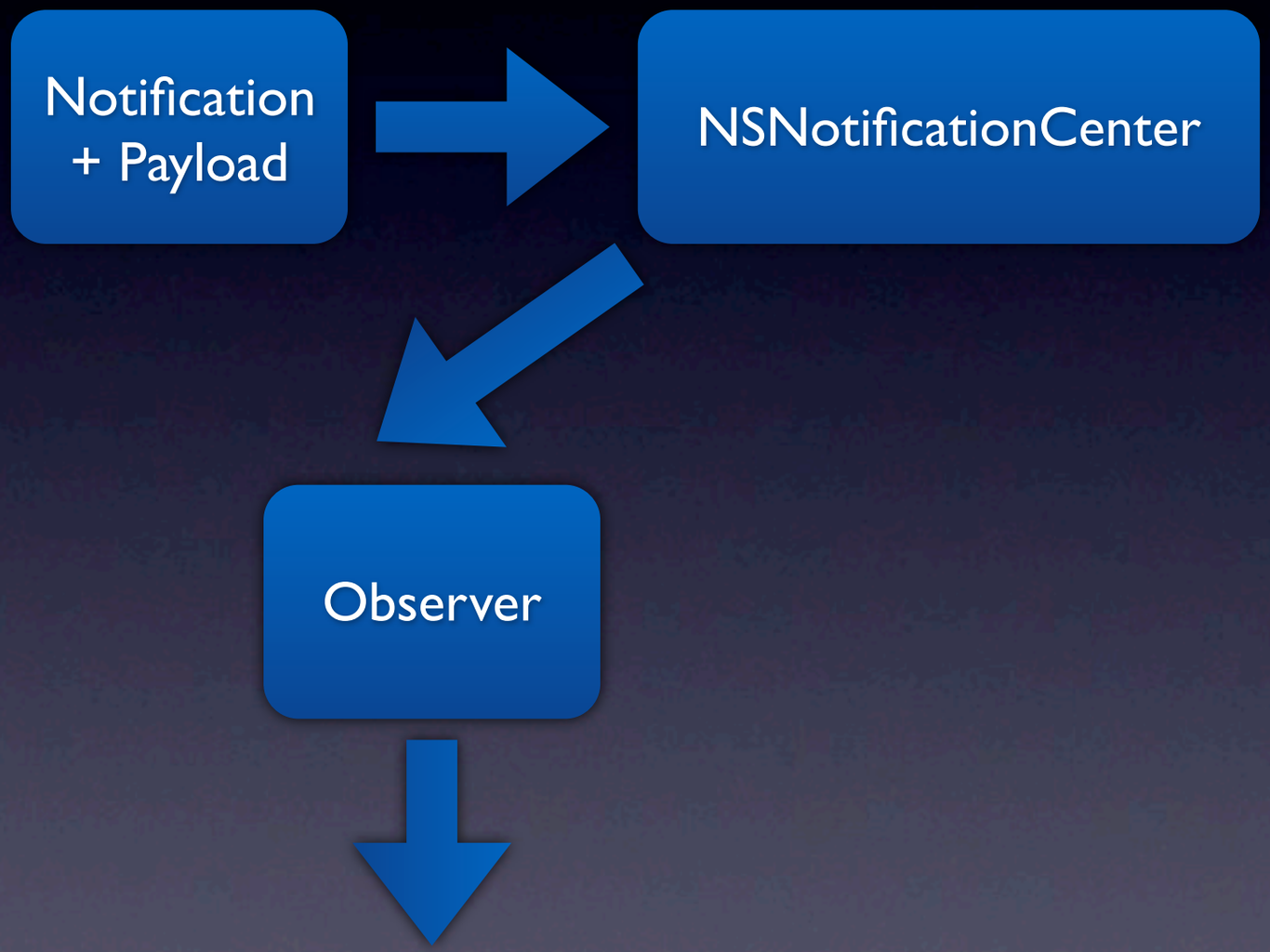
Notification Flow



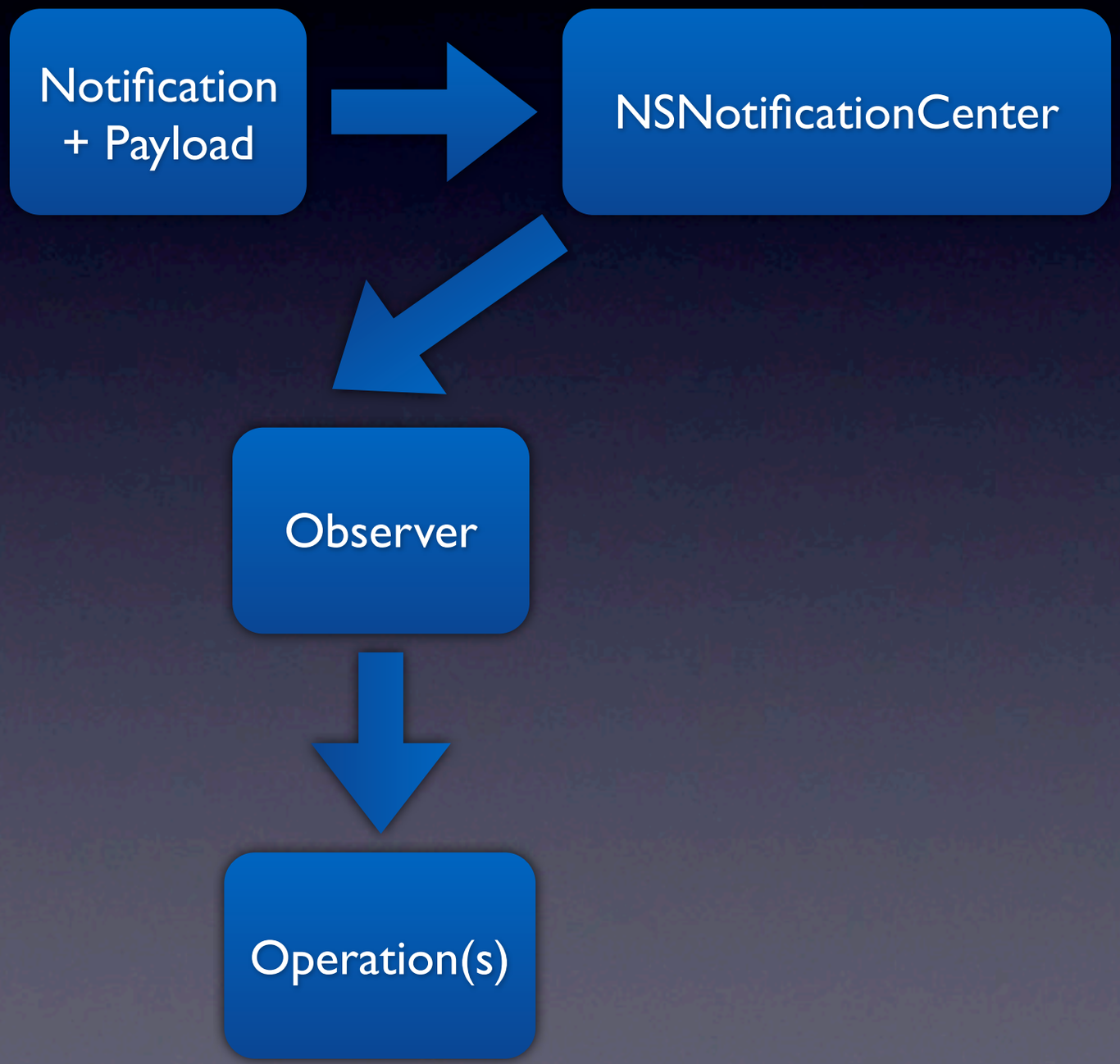
Notification Flow



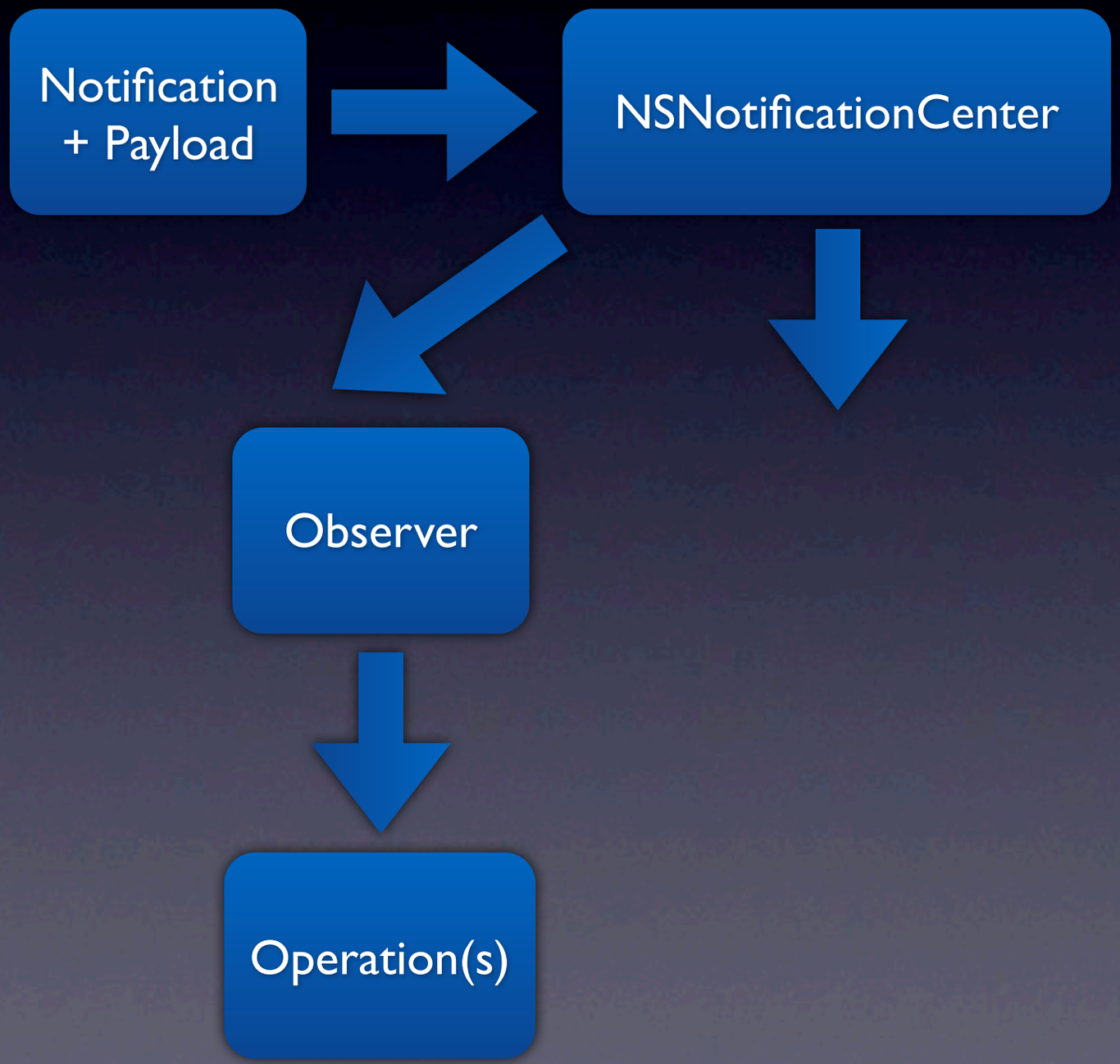
Notification Flow



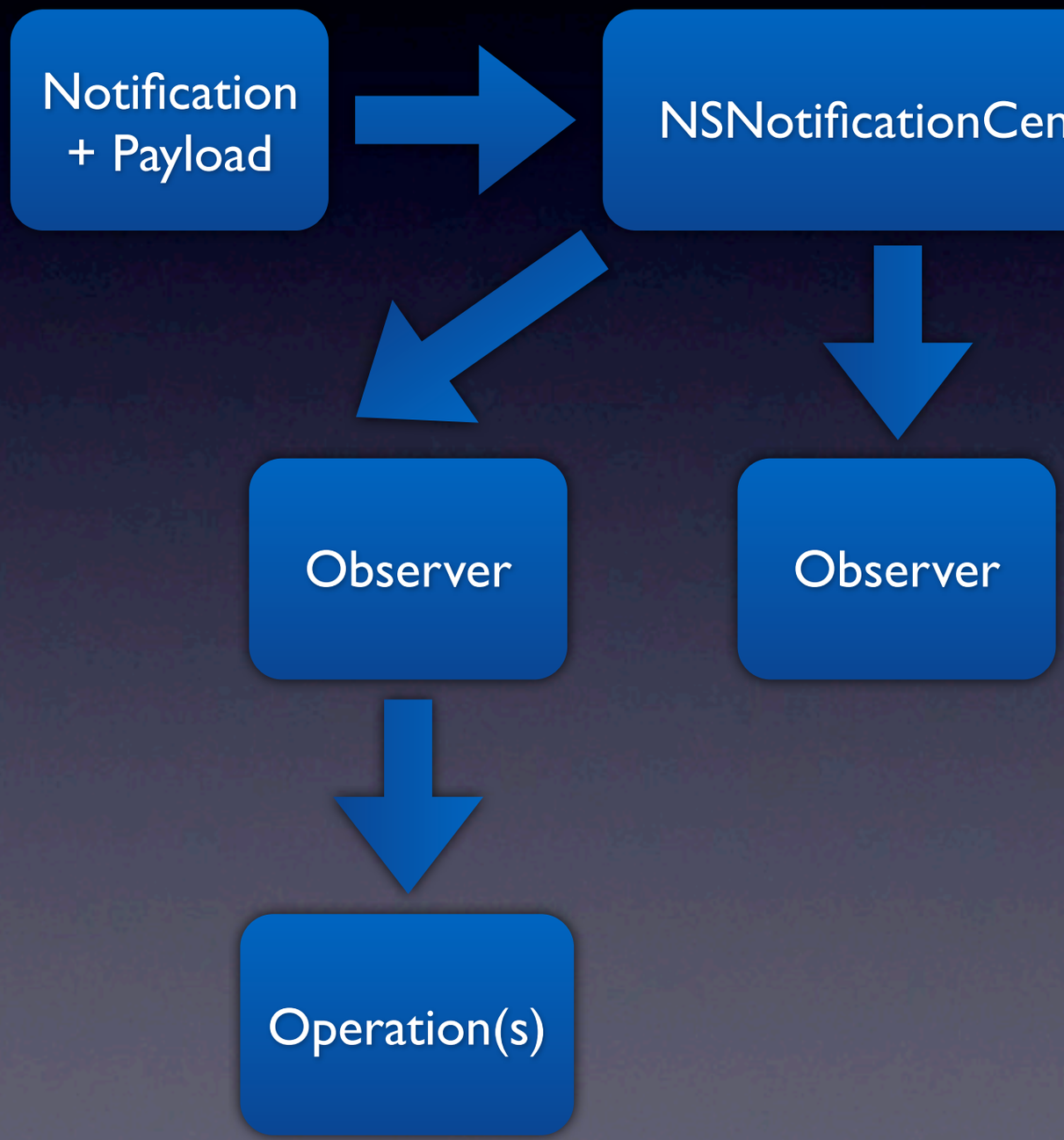
Notification Flow



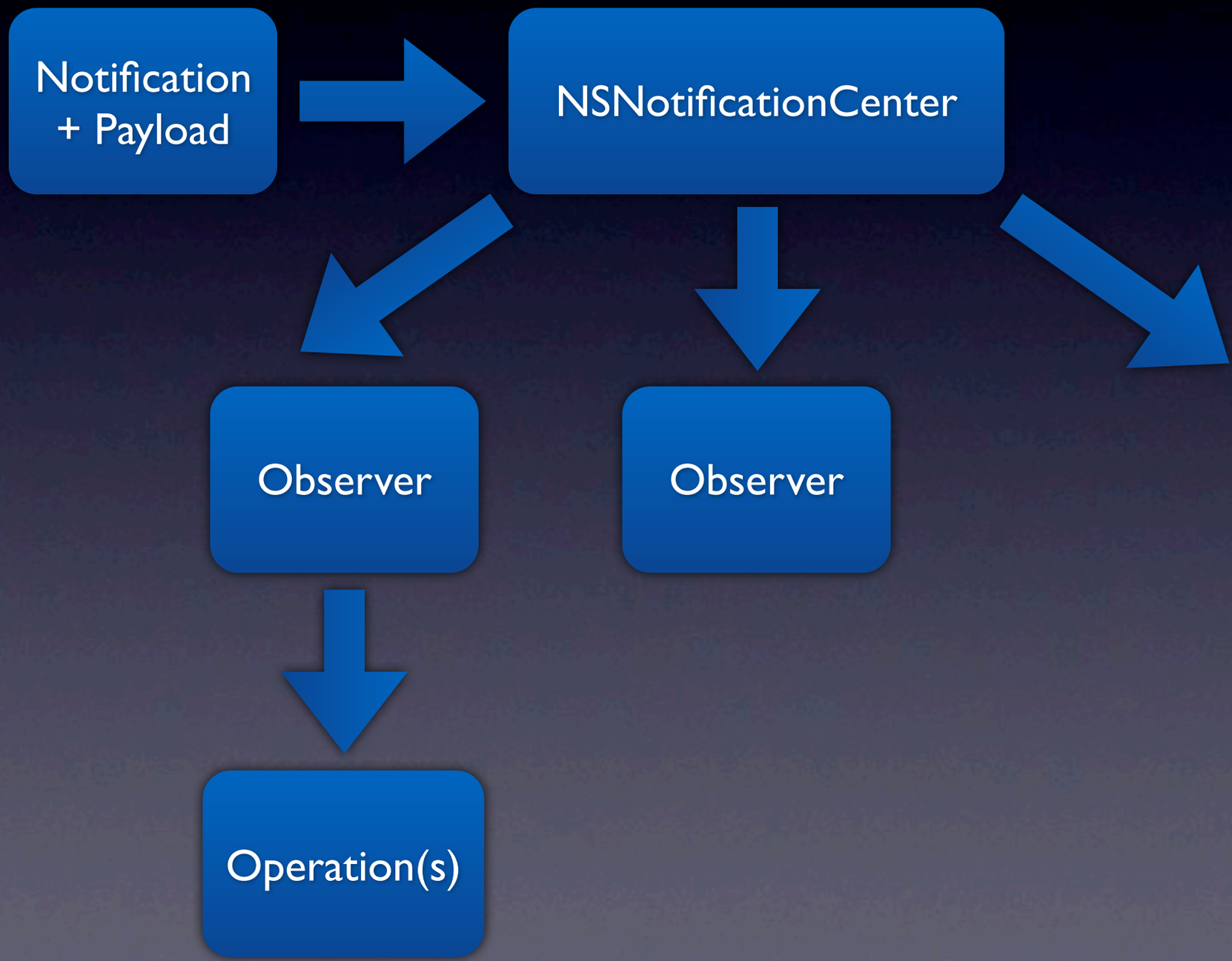
Notification Flow



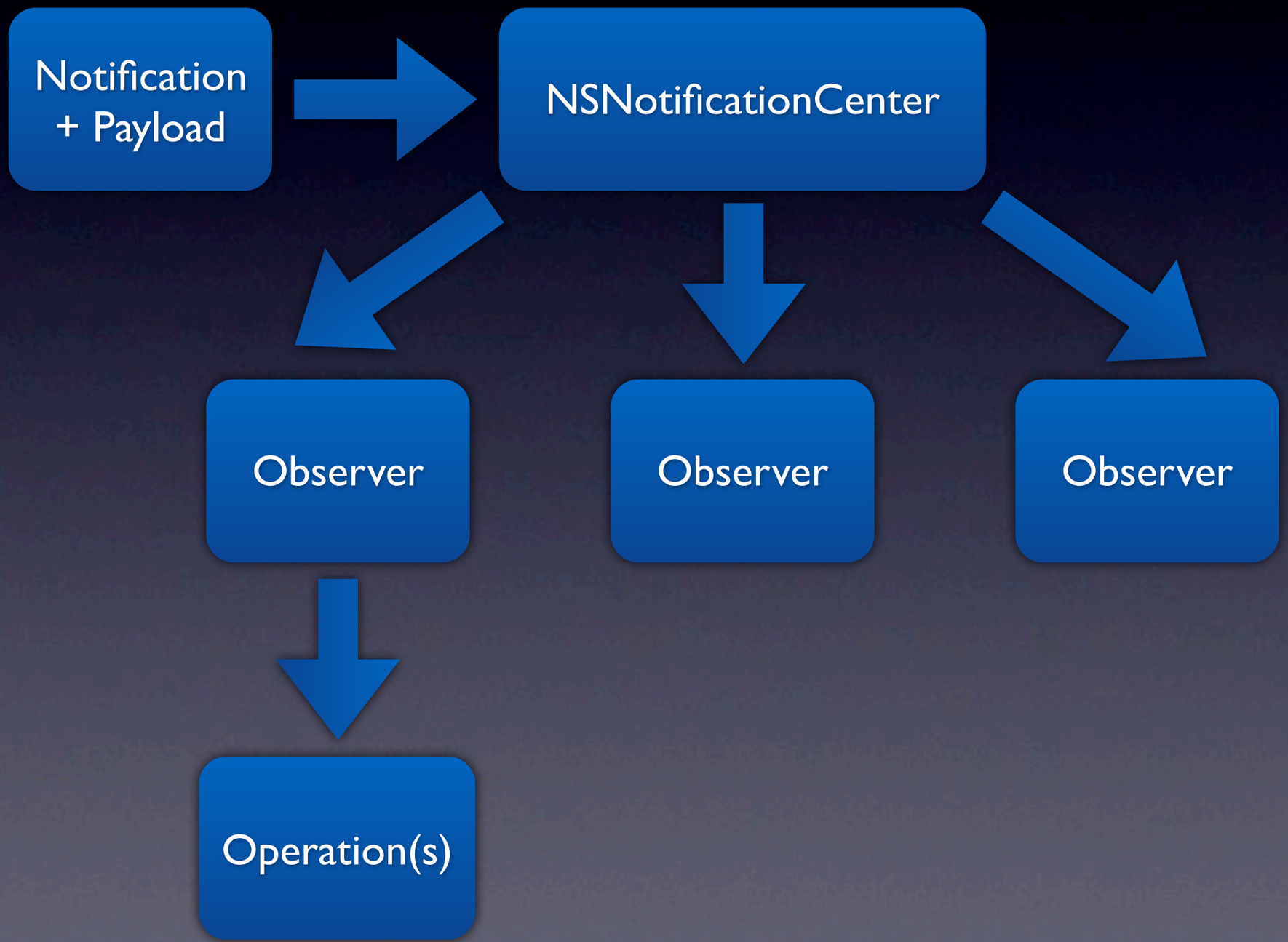
Notification Flow



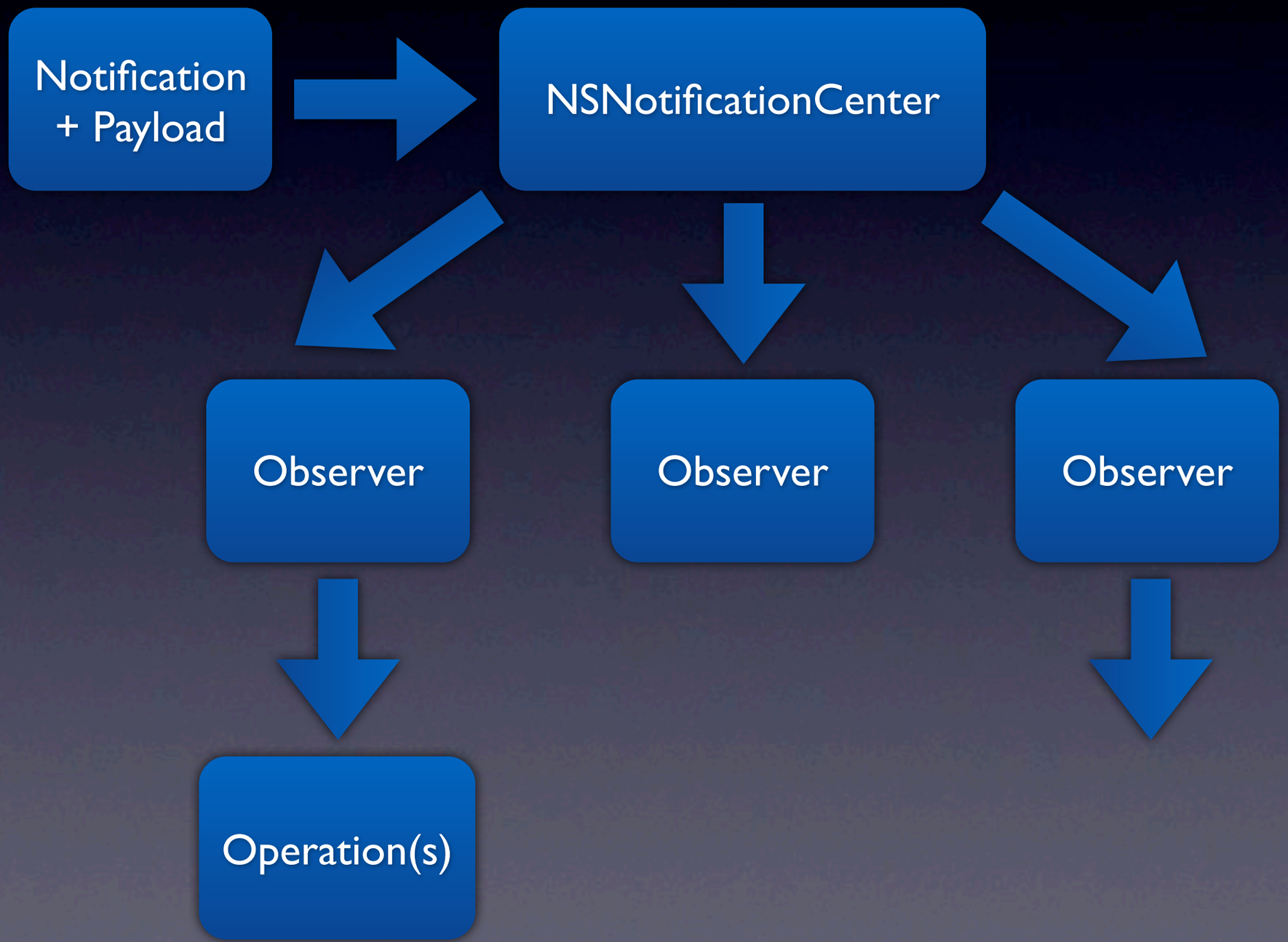
Notification Flow



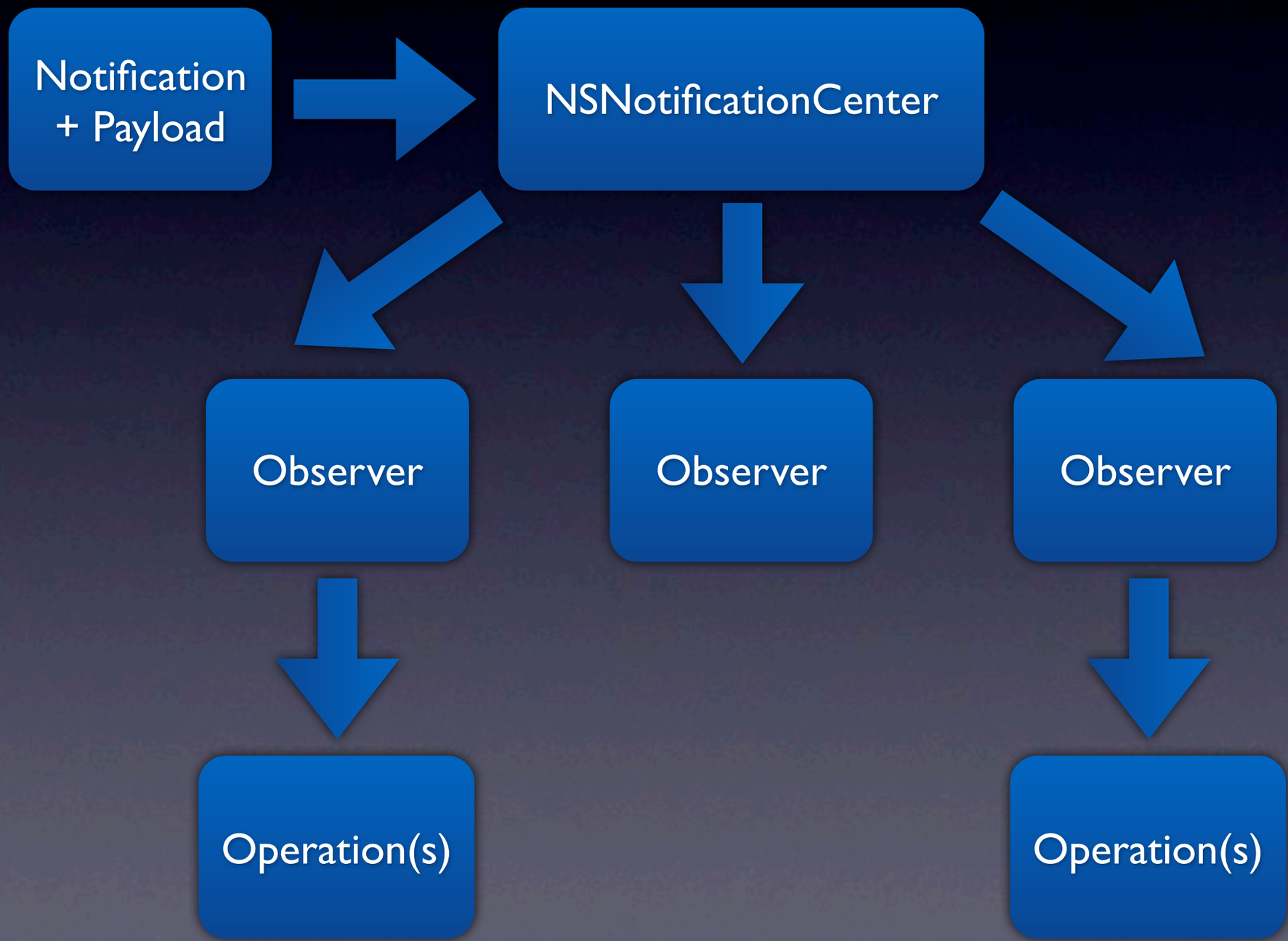
Notification Flow



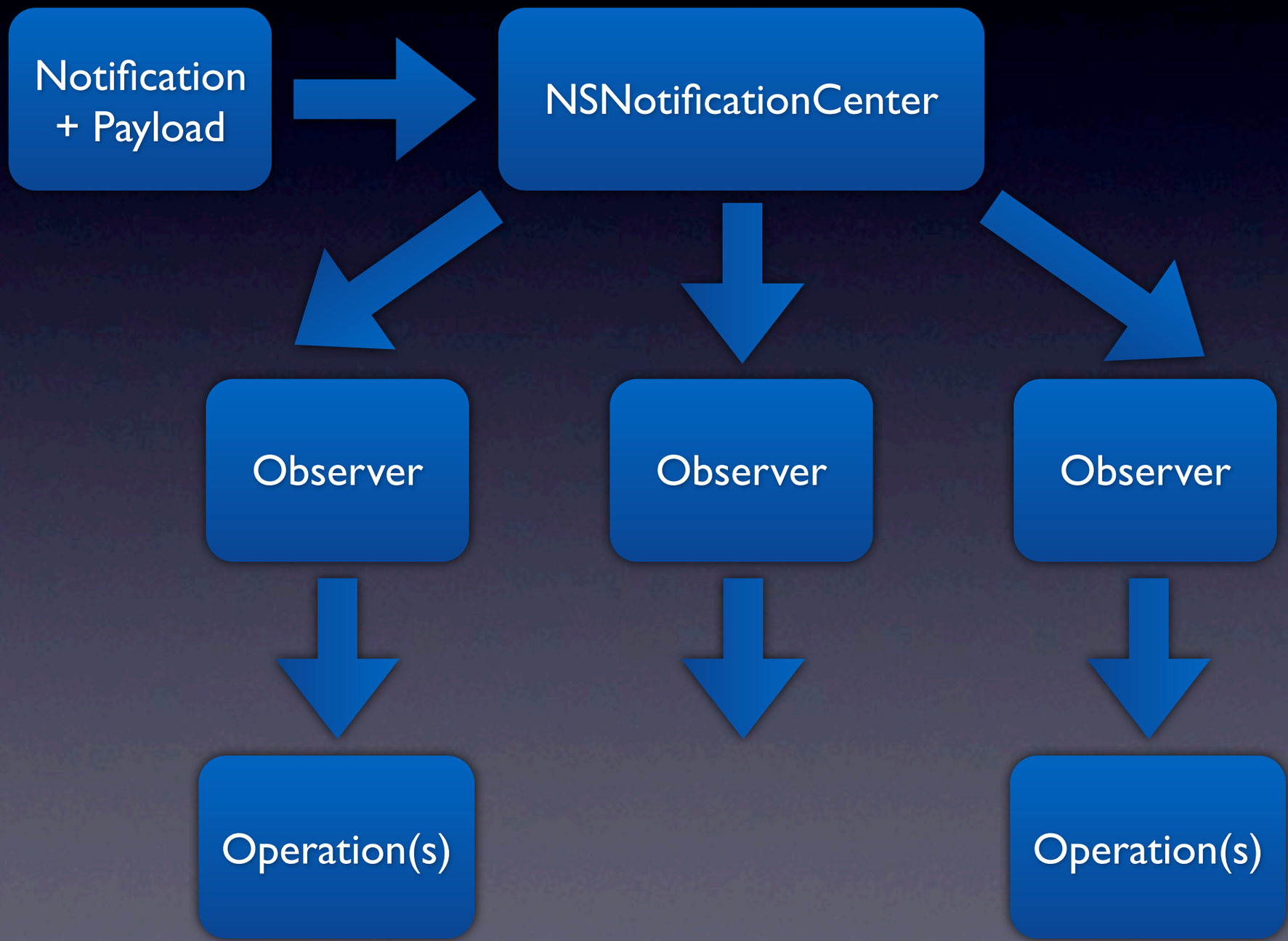
Notification Flow



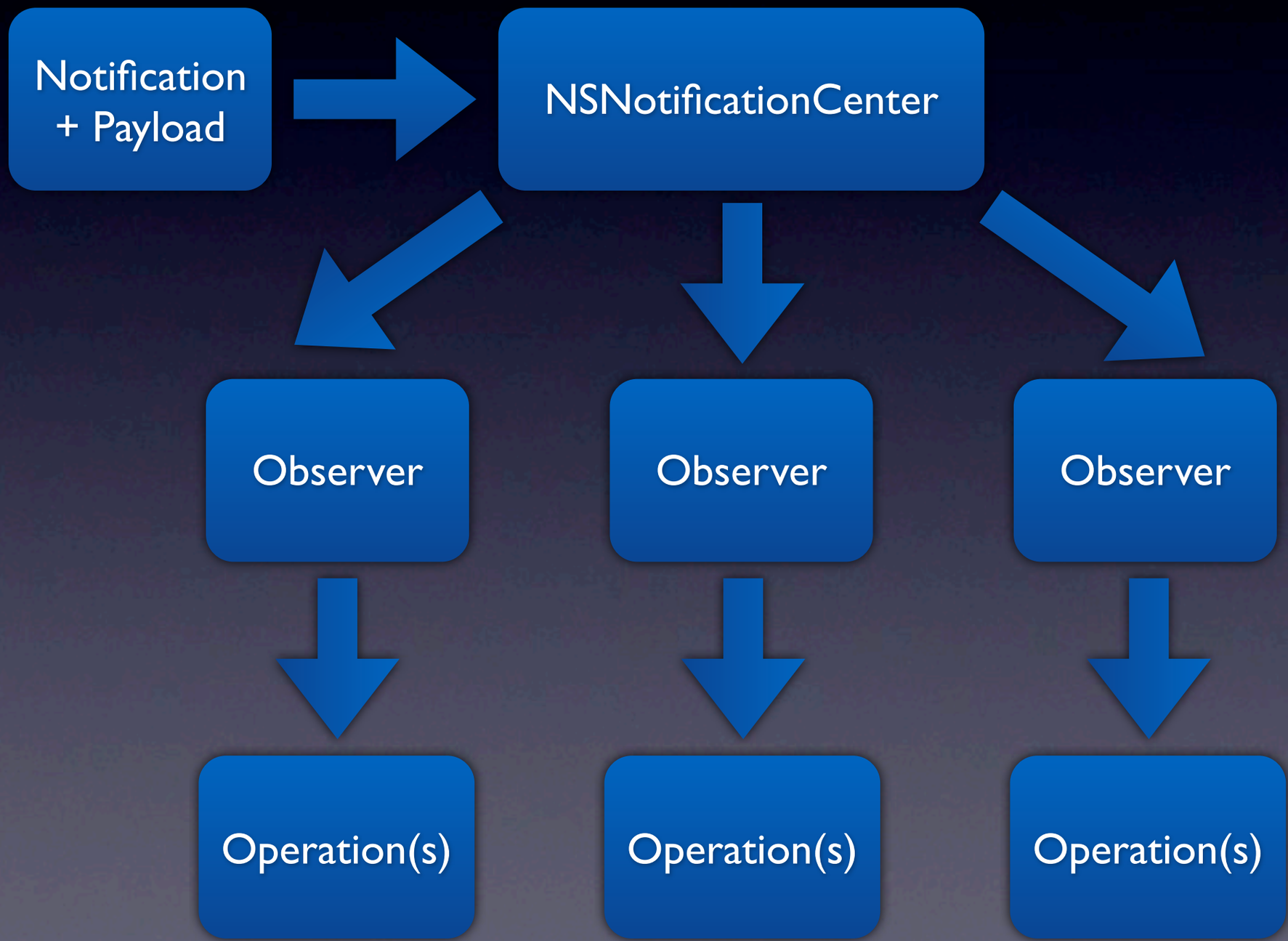
Notification Flow



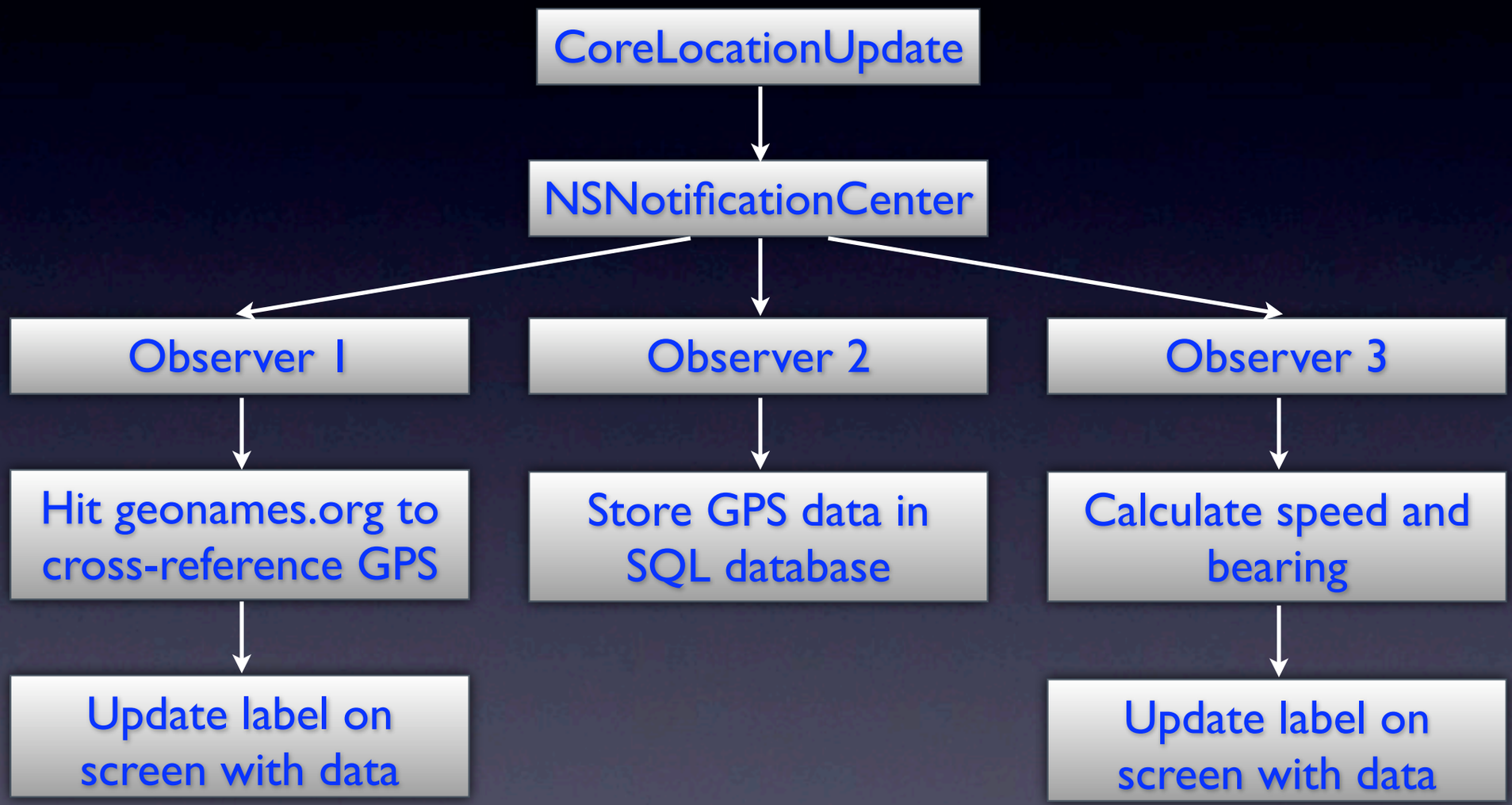
Notification Flow



Notification Flow



A real-world example



All of these operations happen asynchronously.

Notification Pros

- "Fire and Forget"
- A single notification can be processed by multiple listeners
- Notifications can be posted to the queue with different "posting styles" or priorities
- Notifications can be coalesced so they can be sent in pairs or alphabetically
- Almost always used asynchronously.

Notification Cons

- If you don't de-register a handler, you could process a notification message multiple times. (Typically, de-registration is done in the dealloc member function.)
- Sending asynchronous notifications can be hard to debug, like threads.
- The order in which notifications are received by registered listeners is not guaranteed.

How I've used Notifications

- Wrote an application-wide CoreLocation wrapper that notifies when location range changes (remember the real-world example?)
- Sent notifications to update multiple views when a database change occurred.
- Sent notifications when XML lines of data are parsed one-at-a-time.

Delegate + Notification Scenario:

- SQL Insert/Update service listens for notifications, with a payload containing insert or update SQL data.
- When the SQLUpdate call completes, it fires off a new notification indicating new data has been populated.
- A Notification is received, and the affected UITableView then calls “refreshData” and updates itself with the new SQL data, using delegates to populate and refresh itself.

Notification Extras:

- You can enqueue notifications using “enqueueNotification” for finer notification control.
- You can create your own NSNotificationCenter queue for even finer control.
- You can pair notifications up alphabetically or by object for “in order” event firing.

Things to think about...

- Cocoa touch uses a large combination of delegates and notification calls.
- You can see lots of **fun things** the UI does by using Categories on the NSNotificationCenter class.
- If you end up using NSNotificationCenter in your app, try experimenting with the "enqueueNotification" function for finer grained control.

A Quick Demo!!!

- Simple demo that shows NSNotificationCenter being overridden with Categories, so we can see those "fun things"

Thank you!

- khollis@willowtreeconsulting.com
- http://www.bitgatemobile.com/
- Twitter: @KenjiHollis
- kenji.hollis on Skype
- 303-521-1864

- Questions or comments? Speak up!!