Writing **Swift** code with *great* testability

or, how ⌘ + U became my new ⌘ + R

@johnsundell

how ⌘ + U became my new ⌘ + R

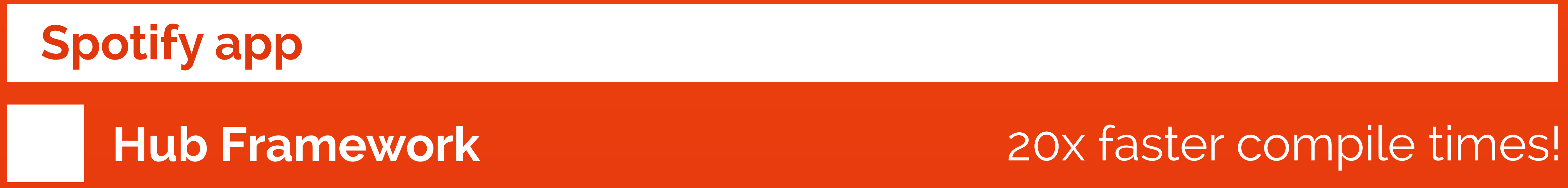
Unit tests are a **waste of time.** It's faster and easier to do manual testing, and focus my coding time on writing **actual** code.

Unbox, Wrap, Hub Framework, etc...

Integrating, re-building, manual testing

Bugs, regressions, unwanted behavior

how ⌘ + U became my new ⌘ + R



20x faster compile times!

🤖 Automating testing = Focus on coding

📋 Tests provide documentation of intent

Tests let me **move faster**, I don't have to constantly run the app, but can instead verify most of its working parts in isolation, and make **quick iterations**.

3 tips on how to work with unit testing in

1 Design your code for testability

What makes code easy to test?

- Unified Input/Output
- State is kept local
- Injected dependencies

```swift
class FileLoader {
    static let shared = FileLoader()
    private let cache = Cache()

    func file(named fileName: String) throws -> File {
        if let cachedFile = cache.file(named: fileName) {
            return cachedFile
        }

        let bundle = Bundle.main

        guard let url = bundle.url(forResource: fileName, withExtension: nil) else {
            throw NSError(domain: "com.johnsundell.myapp.fileLoader", code: 7, userInfo: nil)
        }

        let data = try Data(contentsOf: url)
        let file = File(data: data)

        cache.cache(file: file, name: fileName)

        return file
    }
}
```

Unified Input/Output

State is kept local

Injected dependencies

```
class FileLoader {
    static let shared = FileLoader()
    private let cache = Cache()

    func file(named fileName: String) throws -> File {
        if let cachedFile = cache.file(named: fileName) {
            return cachedFile
        }

        let bundle = Bundle.main

        guard let url = bundle.url(forResource: fileName, withExtension: nil) else {
            throw NSError(domain: "com.johnsundell.myapp.fileLoader", code: 7, userInfo: nil)
        }

        let data = try Data(contentsOf: url)
        let file = File(data: data)

        cache.cache(file: file, name: fileName)

        return file
    }
}
```

Unified Input/Output

State is kept local

Injected dependencies

```swift
class FileLoader {
    static let shared = FileLoader()
    private let cache = Cache()

    func file(named fileName: String) throws -> File {
        if let cachedFile = cache.file(named: fileName) {
            return cachedFile
        }

        let bundle = Bundle.main

        guard let url = bundle.url(forResource: fileName, withExtension: nil) else {
            throw NSError(domain: "com.johnsundell.myapp.fileLoader", code: 7, userInfo: nil)
        }

        let data = try Data(contentsOf: url)
        let file = File(data: data)

        cache.cache(file: file, name: fileName)

        return file
    }
}
```

Unified Input/Output

State is kept local

Injected dependencies

```swift
class FileLoader {
    static let shared = FileLoader()
    private let cache = Cache()

    func file(named fileName: String) throws -> File {
        if let cachedFile = cache.file(named: fileName) {
            return cachedFile
        }

        let bundle = Bundle.main

        guard let url = bundle.url(forResource: fileName, withExtension: nil) else {
            throw NSError(domain: "com.johnsundell.myapp.fileLoader", code: 7, userInfo: nil)
        }

        let data = try Data(contentsOf: url)
        let file = File(data: data)

        cache.cache(file: file, name: fileName)

        return file
    }
}
```

Unified Input/Output

State is kept local

Injected dependencies

```
enum FileLoaderError: Error {
    case invalidFileName(String)
    case invalidFileURL(URL)
}

class FileLoader {
    static let shared = FileLoader()
    private let cache = Cache()

    func file(named fileName: String) throws -> File {
        if let cachedFile = cache.file(named: fileName) {
            return cachedFile
        }

        let bundle = Bundle.main

        guard let url = bundle.url(forResource: fileName, withExtension: nil) else {
            throw FileLoaderError.invalidFileName(fileName)
        }

        do {
            let data = try Data(contentsOf: url)
            let file = File(data: data)

            cache.cache(file: file, name: fileName)

            return file
        } catch {
            throw FileLoaderError.invalidFileURL(url)
        }
    }
}
```

Dedicated error type

Unified error output

Unified Input/Output

State is kept local

Injected dependencies

```swift
enum FileLoaderError: Error {
    case invalidFileName(String)
    case invalidFileURL(URL)
}

class FileLoader {

    private let cache = Cache()

    func file(named fileName: String) throws -> File {
        if let cachedFile = cache.file(named: fileName) {
            return cachedFile
        }

        let bundle = Bundle.main

        guard let url = bundle.url(forResource: fileName, withExtension: nil) else {
            throw FileLoaderError.invalidFileName(fileName)
        }

        do {
            let data = try Data(contentsOf: url)
            let file = File(data: data)

            cache.cache(file: file, name: fileName)

            return file
        } catch {
            throw FileLoaderError.invalidFileURL(url)
        }
    }
}
```

🟩 Unified Input/Output

🟩 State is kept local

🟥 Injected dependencies

Unified Input/Output

State is kept local

Injected dependencies

```swift
class FileLoader {
    private let cache = Cache()
    private let bundle: Bundle

    init(cache: Cache = .init(), bundle: Bundle = .main) {
        self.cache = cache
        self.bundle = bundle
    }

    func file(named fileName: String) throws -> File {
        if let cachedFile = cache.file(named: fileName) {
            return cachedFile
        }



        guard let url = bundle.url(forResource: fileName, withExtension: nil) else {
            throw FileLoaderError.invalidFileName(fileName)
        }

        do {
            let data = try Data(contentsOf: url)
            let file = File(data: data)

            cache.cache(file: file, name: fileName)

            return file
        } catch {
            throw FileLoaderError.invalidFileURL(url)
        }
    }
}
```

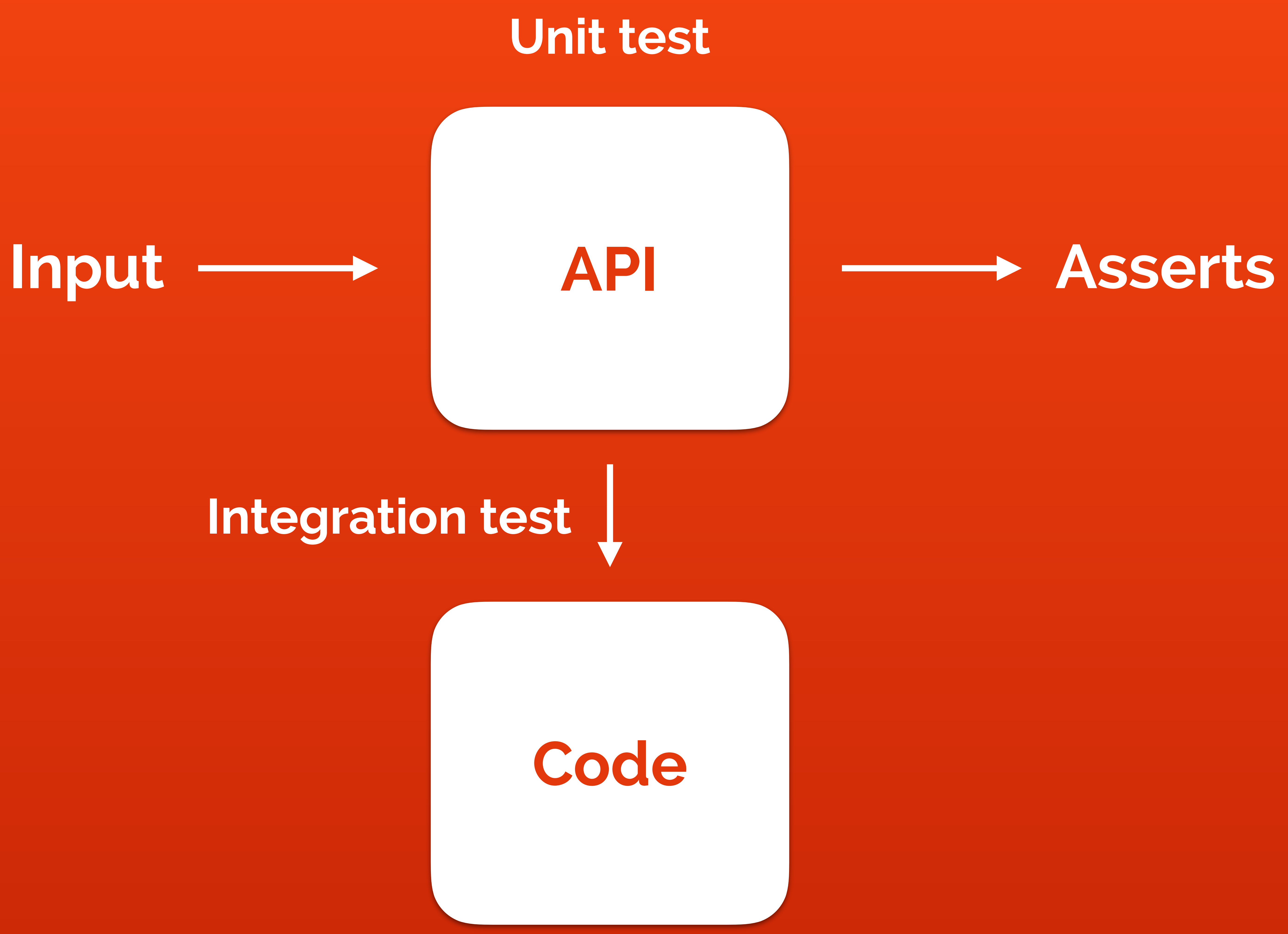
Dependency injection (with defaults)

Using injected dependencies

Let's write a test! 🚀

2 Use access control to create clear API boundaries

2 Use access control to create clear API boundaries



2 Use access control to create clear API boundaries

`private` `fileprivate` `internal` `public` `open`

2

Use access control to create clear API boundaries

```
public class SendMessageViewController: UIViewController {
    public var recipients: [User]
    public var title: String
    public var message: String
    public var recipientsPicker: UserPickerView?
    public var titleTextField: UITextField?
    public var messageTextField: UITextField?
}
```

2

Use access control to create clear API boundaries

```swift
public class SendMessageViewController: UIViewController {
    private var recipients: [User]
    private var title: String
    private var message: String
    private var recipientsPicker: UserPickerView?
    private var titleTextField: UITextField?
    private var messageTextField: UITextField?
}
```

2 Use access control to create clear API boundaries

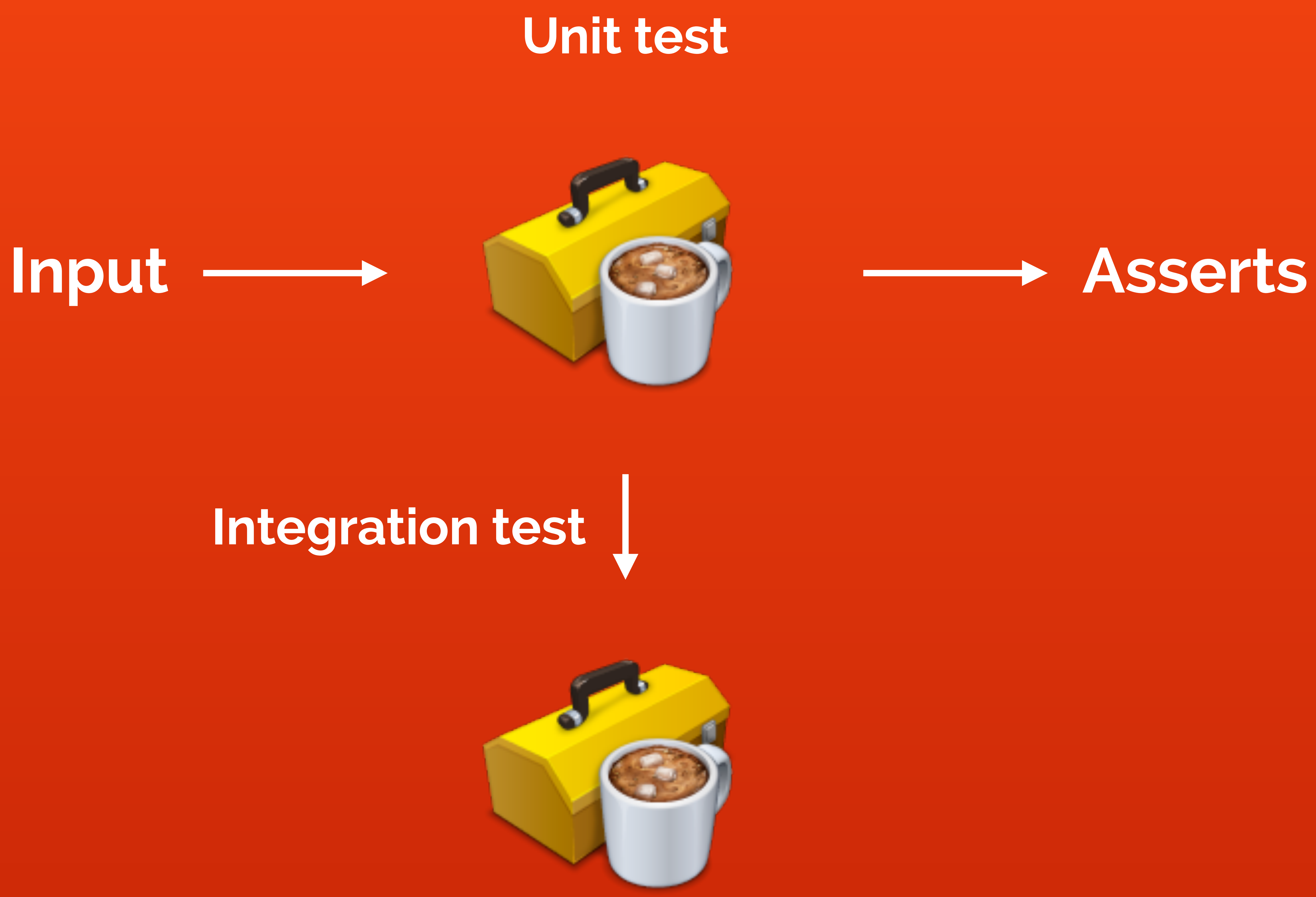
```
public class SendMessageViewController: UIViewController {
    private var recipients: [User]
    private var title: String
    private var message: String
    private var recipientsPicker: UserPickerView?
    private var titleTextField: UITextField?
    private var messageTextField: UITextField?

    func update(recepients: [User]? = nil, title: String? = nil, message: String? = nil) {
        if let recepients = recepients {
            self.recipients = recepients
        }

        // Same for title & message
    }
}
```

Single API entry point

2 Use access control to create clear API boundaries

Frameworks

2 Use access control to create clear API boundaries



2

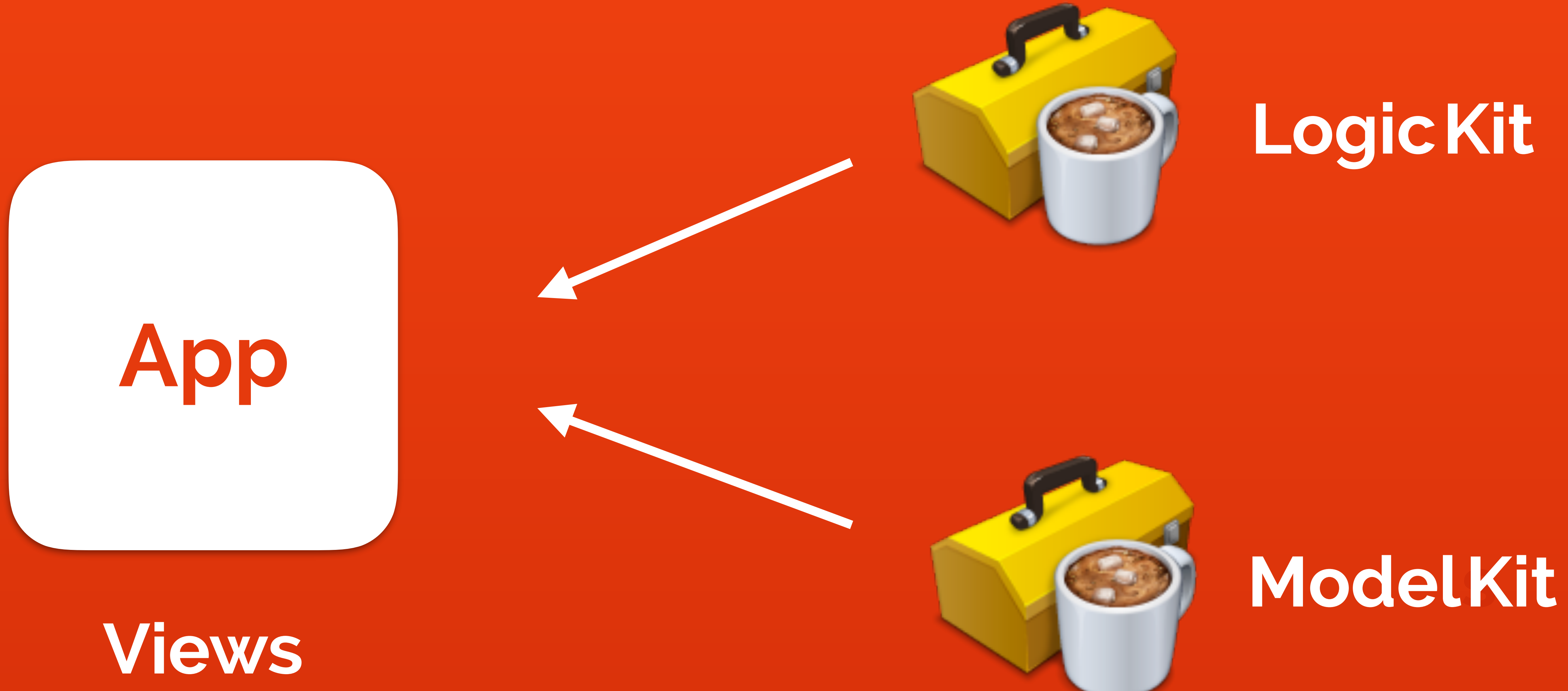
Use access control to create clear API boundaries

Models

Views

Logic

2 Use access control to create clear API boundaries



```
($ brew install swiftplate)
```

3 Avoid mocks to avoid getting tied down into implementation details

3 Avoid mocks to avoid getting tied down into implementation details

👻 Mocks are “fake” objects that are used in tests to be able to assert that certain things happen as expected.

3

Avoid mocks to avoid getting tied down into implementation details

```
// Objective-C (using OCMockito)
NSBundle *bundle = mock([NSBundle class]);
[given([bundle pathForResource:anything() ofType:anything()]) willReturn:@“path”];

FileLoader *fileLoader = [[FileLoader alloc] initWithBundle:bundle];
XCTAssertNotNil([fileLoader fileNamed:@"file"]);
```

3 Avoid mocks to avoid getting tied down into implementation details

```
// Swift (partial mocking)
class MockBundle: Bundle {
    var mockPath: String?

    override func path(forResource name: String?, ofType ext: String?) -> String? {
        return mockPath
    }
}

let bundle = MockBundle()
bundle.mockPath = "path"

let fileLoader = FileLoader(bundle: bundle)
XCTAssertNotNil(fileLoader.file(named: "file"))
```

3

Avoid mocks to avoid getting tied down into implementation details

```
// Swift (no mocking)
let bundle = Bundle(for: type(of: self))
let fileLoader = FileLoader(bundle: bundle)
XCTAssertNotNil(fileLoader.file(named: "file"))
```

3

Avoid mocks to avoid getting tied down into implementation details

```swift
class ImageLoader {
    func loadImage(named imageName: String) -> UIImage? {
        return UIImage(named: imageName)
    }
}


class ImageViewController: UIViewController {
    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        imageView.image = imageLoader.loadImage(named: imageName)
    }
}
```

3 Avoid mocks to avoid getting tied down into implementation details

```swift
// Test using mocking
class ImageViewControllerTests: XCTestCase {
    func testImageLoadedOnViewWillAppear() {
        class MockImageLoader: ImageLoader {
            private(set) var loadedImageNames = [String]()

            override func loadImage(named name: String) -> UIImage {
                loadedImageNames.append(name)
                return UIImage()
            }
        }

        let imageLoader = MockImageLoader()
        let vc = ImageViewController(imageLoader: imageLoader)
        vc.imageName = "image"
        vc.viewWillAppear(false)
        XCTAssertEqual(imageLoader.loadedImageNames, ["image"])
    }
}
```

Manually implemented, partial mock

Capture loaded image names

Asserting that an image was loaded by verifying what our mock captured

3 Avoid mocks to avoid getting tied down into implementation details

```swift
class ImageLoader {
    private let preloadedImages: [String : UIImage]

    init(preloadedImages: [String : UIImage] = [:]) {
        self.preloadedImages = preloadedImages
    }

    func loadImage(named imageName: String) -> UIImage? {
        if let preloadedImage = preloadedImages[imageName] {
            return preloadedImage
        }

        return UIImage(named: imageName)
    }
}
```

← Optionally enable preloaded images to be injected

↑ Use preloaded image if any exists

3

Avoid mocks to avoid getting tied down into implementation details

```
// Test without mocking
class ImageViewControllerTests: XCTestCase {
    func testImageLoadedOnViewWillAppear() {
        let image = UIImage()
        let imageLoader = ImageLoader(images: ["image" : image])
        let vc = ImageViewController(imageLoader: imageLoader)
        vc.imageName = "image"
        vc.viewWillAppear(false)
        XCTAssertEqual(vc.image, image)
    }
}
```

Inject image

Compare against actually rendered image, instead of relying on mock capturing

However, sometimes you do need mocks, so let's make it easy to use them! 👍

To summarize

1. Design your code for testability
2. Use access control to create clear API boundaries
3. Avoid mocks to avoid getting tied down into implementation details

No tests? 😱

No problem! 😀

Just start somewhere 👍

Set goals for test coverage 🚀

@johnsundell

/johnsundell