Testing

Testing

Unit testing

Automated testing

Manual testing

Unit Testing

Why bother with unit tests?

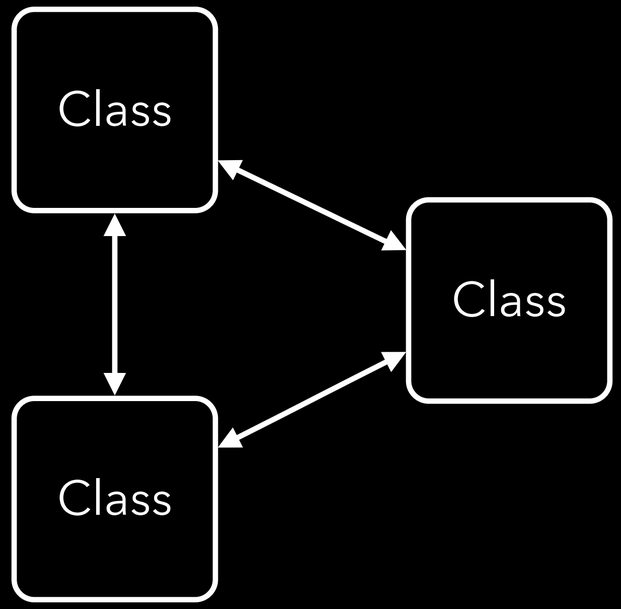
“Unit testing is teh suck, Urr.” - Wil Shipley

- There’s no substitute for manual testing
- You can’t test UI or performance
- iOS apps are simple
- There’s no time!

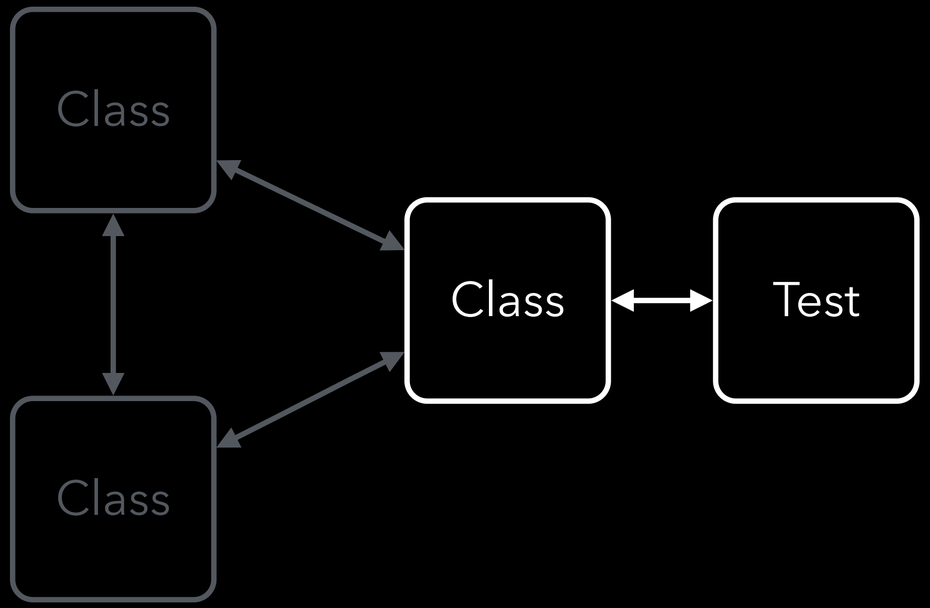
Why bother with unit tests?

- Crashes
- Regressions
- Automation
- Helps you write the code

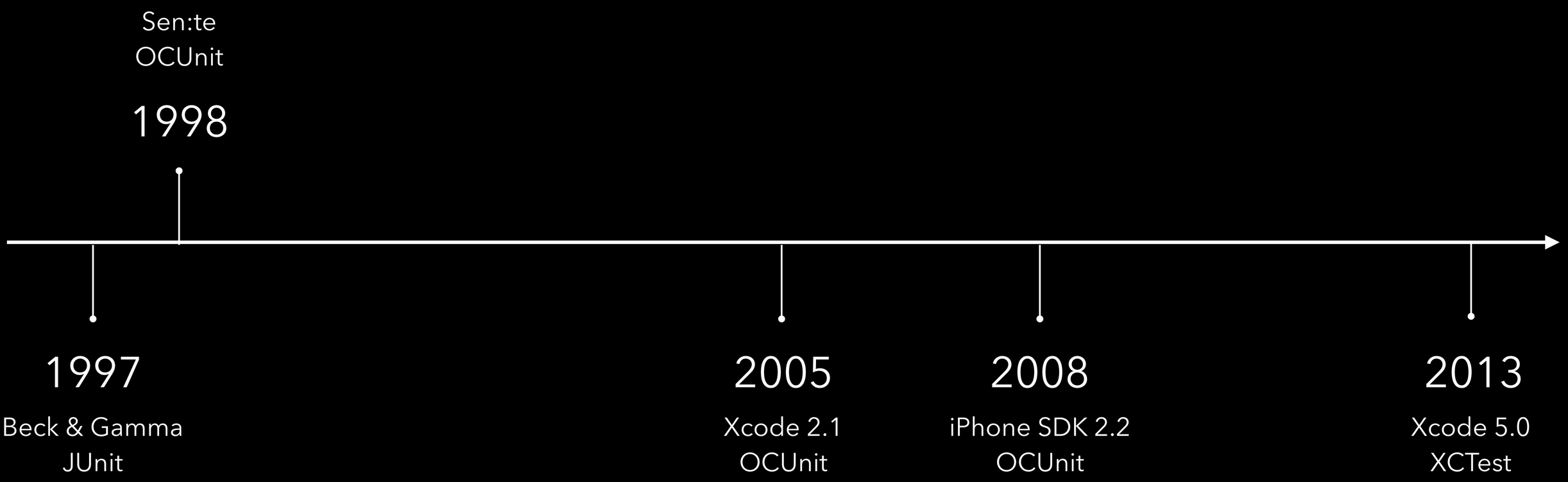
Unit Testing



Unit Testing



A (Very) Brief History



OCUnit

- Precursor to XCTest
- Nearly one-to-one correspondence with XCUnit assertions
- Still the best choice for iOS 6 compatibility
- Xcode migration tool

XCTest

xUnit for Objective-C

- XCTestCase subclasses
- test/setup/teardown methods
- Assertions
- Asynchronous testing coming soon (Xcode 6)

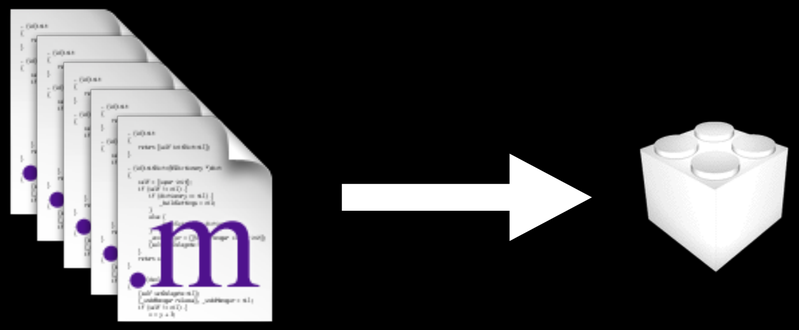
XCTestCase

SmarticleTests.m

```
@interface SmarticleTests : XCTestCase
@end

@implementation SmarticleTests

- (void)testMath
{
   XCTAssertTrue(1 + 1 == 2);
}

@end
```

- Combined interface/implementation
- Subclasses of XCTestCase
- Methods that start with test
- Assertions

XCTestCase



- Built into .xctest bundle
- Injected with test runner
- App is started
- applicationDidFinishLaunching:
- Each XCTestCase is instantiated
- Each test method is run

Demo: SmarticleTests

What to test

- Success-driven - The happy path (optionally TDD)
- Defect-driven - Every bug turns into a test
- Failure-driven - Expect the unexpected

Expecting the Unexpected

- Overflow, divide-by-zero, NaN
- Nil
- Empty collections
- Unexpected types in collections
- NSErrors
- Exceptions thrown

Assertions

XCTAssertNil

XCTAssertNotNil

XCTAssert

XCTAssertTrue

XCTAssertFalse

XCTAssertEqualObjects

XCTAssertNotEqualObjects

XCTAssertEqual

XCTAssertNotEqual

XCTAssertEqualWithAccuracy

XCTAssertNotEqualWithAccuracy

XCTAssertThrows

XCTAssertThrowsSpecific

XCTAssertThrowsSpecificNamed

XCTAssertNoThrow

XCTAssertNoThrowSpecificNamed

XCTAssertTrue/False

```
NSArray *array = nil;

XCTAssertFalse(array.count); // Pass

array = @[];

XCTAssertFalse(array.count); // Pass

/*…*/

NSDictionary *dictionary = @{
    @“array” : array
};
```

XCTAssertNotNil

```
NSArray *array = nil;

XCTAssertNotNil(array); // Fail

array = @[];

XCTAssertNotNil(array); // Pass

/*…*/

NSDictionary *dictionary = @{
    @“array” : array
};
```

XCTAssertEquals

For testing C scalars and primitives

```
NSArray *array = @[@"One", @"Two", @“Three"];

XCTAssertEquals(array.count, 3); // Fails, “3” not equal to “3”

XCTAssertEquals(array.count, 3U); // Works on 32-bit, not 64-bit

XCTAssertEquals(array.count, (NSUInteger)3); // Works everywhere

XCTAssert(array.count == 3); // Works everywhere
```

XCTAssertEqualObjects

Testing object equality

```
NSArray *array1 = @[@"One", @"Two", @“Three”];
NSArray *array2 = @[@"One", @"Two", @“Three”];

XCTAssertEquals(array1, array2); // Fails (“<0ad8…1000>”)
  is not equal to (“<04a8…1800>”)

XCTAssertEqualObjects(array1, array2); // Passes
```

Setup/Teardown

```
@implementation SmarticleTests

+ (void)setUp {
}

+ (void)tearDown {
}

- (void)setUp {
    [super setUp];
}

- (void)tearDown {
    [super tearDown];
}

- (void)testAddition {
    XCTAssertTrue(1 + 1 == 2);
}

- (void)testSubtraction {
    XCTAssertTrue(1 - 1 == 0);
}

@end
```

+ setup

  - setup
    - testAddition
  - teardown

  - setup
    - testSubtraction
  - teardown

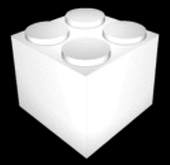
+ teardown

Setup/Teardown

- Any common test code
- Setup shims
  - Network connectivity
  - Database access
- Create fixtures
  - In memory
  - From files

Loading files

.xctest



.app

Test files are packaged in a separate bundle

```
// From within any test case
NSBundle *testBundle = [NSBundle bundleForClass:[self class]];
NSURL *json = [testBundle URLForResource:@"articles"
                           withExtension:@"json"];
```

Testing Private Interfaces

One option…

Don’t test private methods

Testing Private Interfaces

Otherwise, several options…

- Refactor private interface into separate classes
- Redeclare private interface in a test category
- Separate private interface header file

Separate Classes

Sandwich.h

```
@interface Sandwich : NSObject

- (void)initWithType:(SandwichType)type;

@end
```

Sandwich.m

```
#import “Sandwich.h”

#pragma mark - Private Interface

@interface Sandwich()

- (NSArray *)secretIngredientsForType:(SandwichType)type;

@end

@implementation Sandwich

#pragma mark - Private methods

@end
```

Separate Classes

Sandwich.h

```
@interface Sandwich : NSObject

- (void)initWithType:(SandwichType)type;

@end
```

Sandwich.m

```
#import “Sandwich.h”

#pragma mark - Private Interface

@interface Sandwich()


@end

@implementation Sandwich

#pragma mark - Private methods

@end
```

Separate Classes

Sandwich.h

```
@interface Sandwich : NSObject

- (void)initWithType:(SandwichType)type;

@end
```

Sandwich.m

```
#import “Sandwich.h”

#pragma mark - Private Interface

@interface Sandwich()

@property (nonatomic, strong) SecretRecipe *recipe;

@end

@implementation Sandwich

#pragma mark - Private methods

@end
```

Separate Classes

SecretRecipe.h

```
@interface SecretRecipe : NSObject

- (NSArray *)secretIngredientsForType:(SandwichType)type;

@end
```

SecretRecipe.m

```
#import “SecretRecipe.m”

#pragma mark - Private Interface

@interface SecretRecipe()

@end

@implementation SecretReceip

#pragma mark - Private methods

@end
```

Separate Classes

Positives

- No funny business
- Encourages smaller classes

Negatives

- Requires a change in the code to accomodate testing
- Too many classes can be an organizational pain

Test Category

Sandwich.h

```
@interface Sandwich : NSObject

- (void)initWithType:(SandwichType)type;

@end
```

Sandwich.m

```
#import “Sandwich.h”

#pragma mark - Private Interface

@interface Sandwich()

- (NSArray *)secretIngredientsForType:(SandwichType)type;

@end

@implementation Sandwich

#pragma mark - Private methods

@end
```

Test Category

SandwichTests.m

```
#import “Sandwich.h”

@interface Sandwich()
- (NSArray *)secretIngredientsForType:(SandwichType)type;
@end

@interface SandwichTests : XCTestCase
@end

@implementation SandwichTests

- (void)testForSecretIngredients
{
    Sandwich *s = [[Sandwich alloc] init];
    NSArray *ingredients = [s secretIngredientsForType:Rueben];
    XCTAssertNotNil(ingredients);
}

@end
```

Test Category

Positives

- Test private methods, for real
- Organized

Negatives

- Compiler won’t help you if you get it wrong
- Method signature duplication

Private Interface Header

Sandwich.h

```
@interface Sandwich : NSObject

- (void)initWithType:(SandwichType)type;

@end
```

Sandwich.m

```
#import “Sandwich.h”

#pragma mark - Private Interface

@interface Sandwich()

- (NSArray *)secretIngredientsForType:(SandwichType)type;

@end

@implementation Sandwich

#pragma mark - Private methods

@end
```

Private Interface Header

Sandwich.h

```
@interface Sandwich : NSObject

- (void)initWithType:(SandwichType)type;

@end
```

Sandwich+Private.h

```
@interface Sandwich()

- (NSArray *)secretIngredientsForType:(SandwichType)type;

@end
```

Sandwich.m

```
#import “Sandwich.h”
#import “Sandwich+Private.h”

@implementation Sandwich

#pragma mark - Private methods

@end
```

Private Interface Header

SandwichTests.m

```
#import “Sandwich.h”
#import “Sandwich+Private.h”

@interface SandwichTests : XCTestCase
@end

@implementation SandwichTests

- (void)testForSecretIngredients
{
    Sandwich *s = [[Sandwich alloc] init];
    NSArray *ingredients = [s secretIngredientsForType:Rueben];
    XCTAssertNotNil(ingredients);
}

@end
```

Private Interface Header

Positives

- Test private methods, for reals
- No duplication of method signatures

Negatives

- Two header files = more switching/tabbing between files

Dynamic Language Features

- performSelector:withObject:

- valueForKey:

- NSInvocation

What’s missing?

- Mocks and stubs
- Expectations
- Concurrency
- Elegance?

OCMock

- Since 2005, the defacto standard mocking library
- Mocks and stubs
- Class method mocks
- Expectations
- Swizzling
- Others include OCMockito, Kiwi

OCMock

```
id mock = [OCMockObject mockForClass:[Sandwich class]];

[[mock expect] secretIngredientsForType:Club];

[mock verify];  // Fails unless secretIngredientsForType was called
```

OCMock

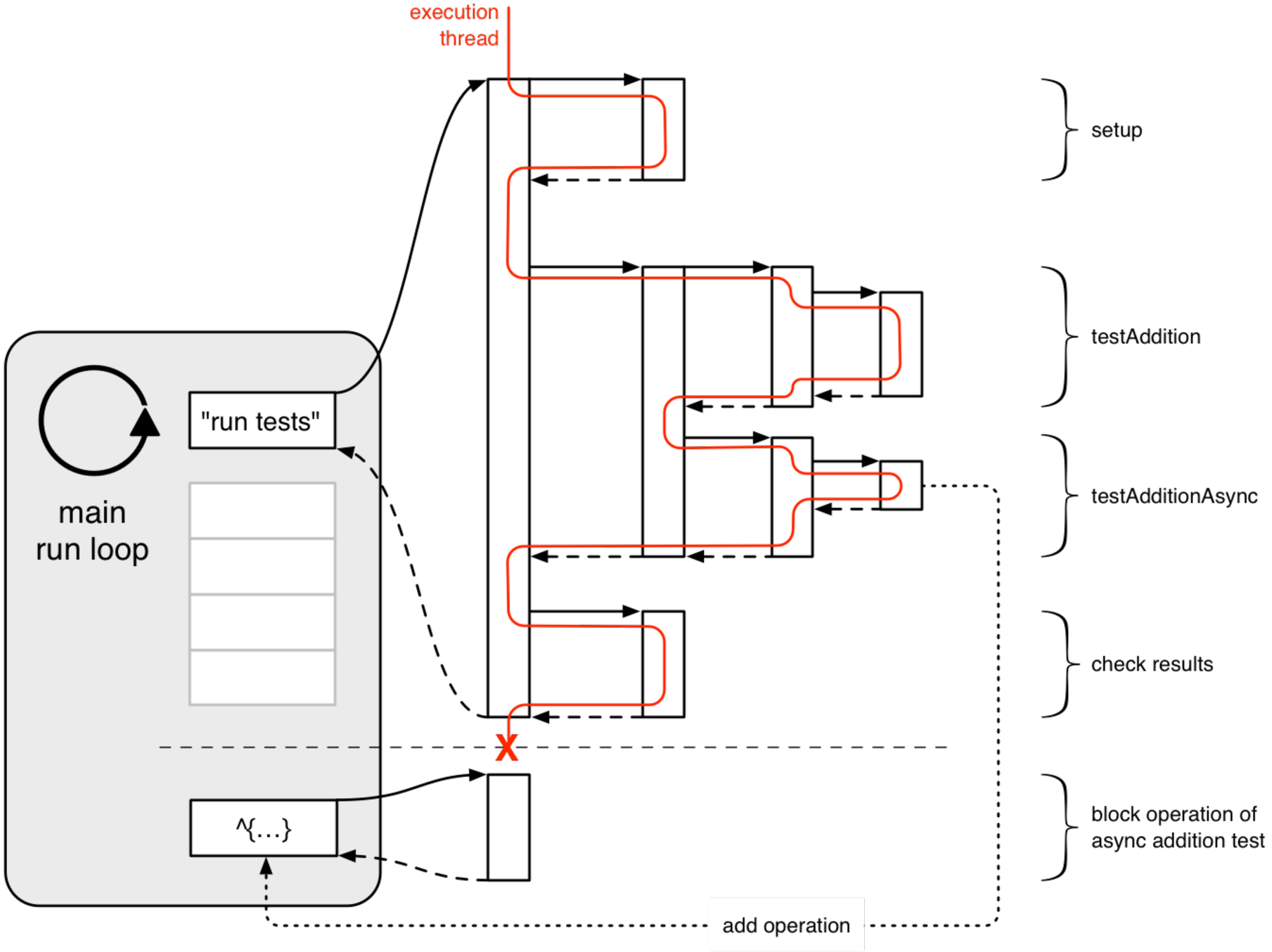
```
id mockRecipe = [OCMockObject stubForClass:[SecretRecipe class]];

NSArray *clubIngredients = @[];

[[[mockRecipe stub] andReturn:clubIngredients] secretIngredientsForType:Club];

// Now, calls that use mockRecipe behave predictably
```

OCMock

```
id mock = [OCMockObject observerMock];

[[NSNotificationCenter defaultCenter] addMockObserver:mock
                                                 name:SandwichNotification
                                               object:nil];

[[mock expect] notificationWithName:SandwichNotification object:[OCMArg any]];

// Make calls that generate notifications

[mock verify];
```

Concurrency

- Tests run on the main thread on the same run loop
- Can’t evaluate results of asynchronous operations
- By the time your operation finishes, the test is over


execution
thread
"run tests"
main
run loop
^{...}
X
setup
testAddition
testAdditionAsync
check results
block operation of
async addition test
add operation

Pause the test

- GCD dispatch semaphores, with dispatch_wait()
- NSRunLoop runMode:beforeDate:
- Libraries - Kiwi, SenTestingKitAsync, others (Xcode 6)

Demo: ArticlesProviderTests

Lab 6.1

ArticleCellTests