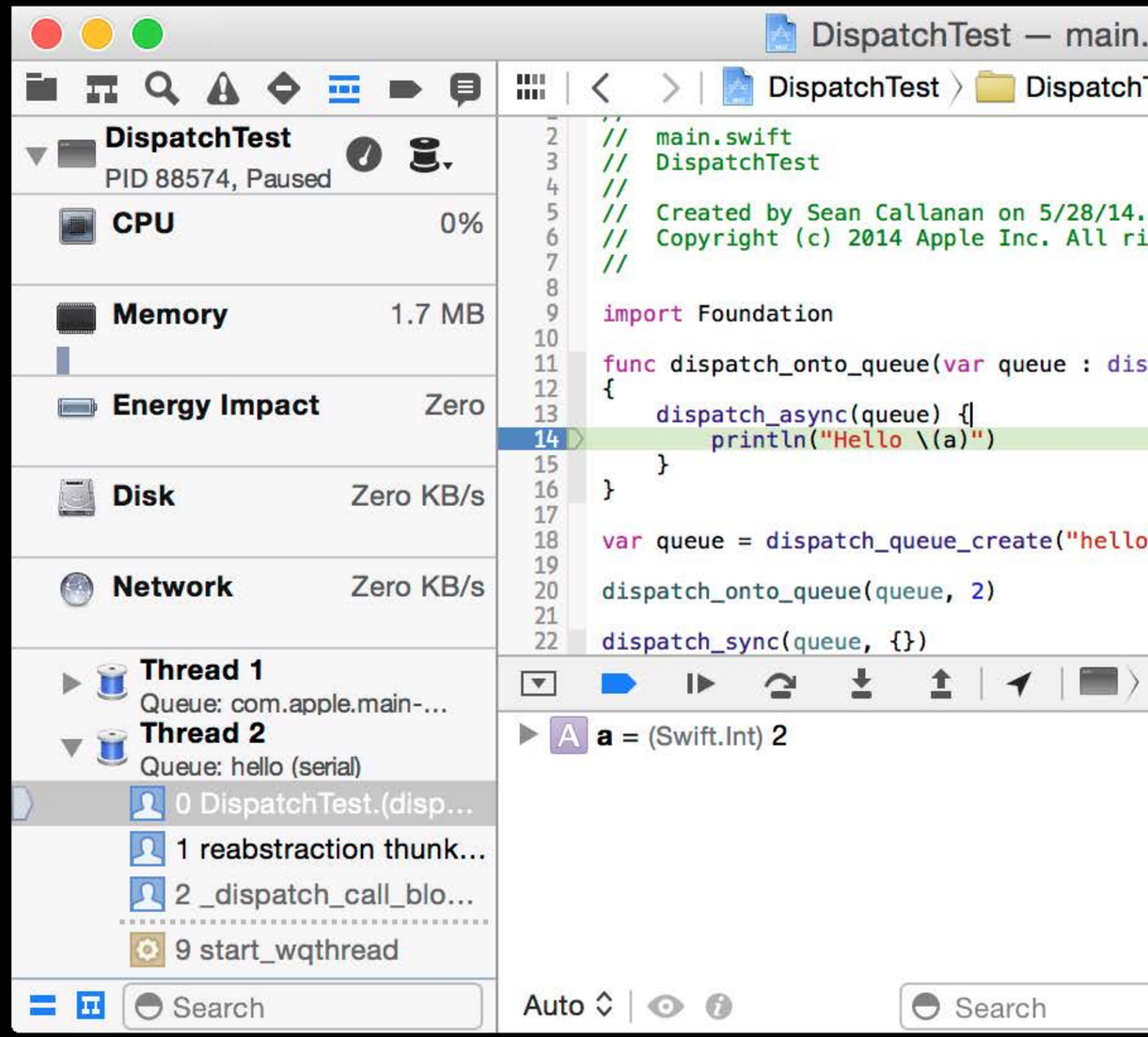
Tools

#WWDC14

Introduction to LLDB and the Swift REPL

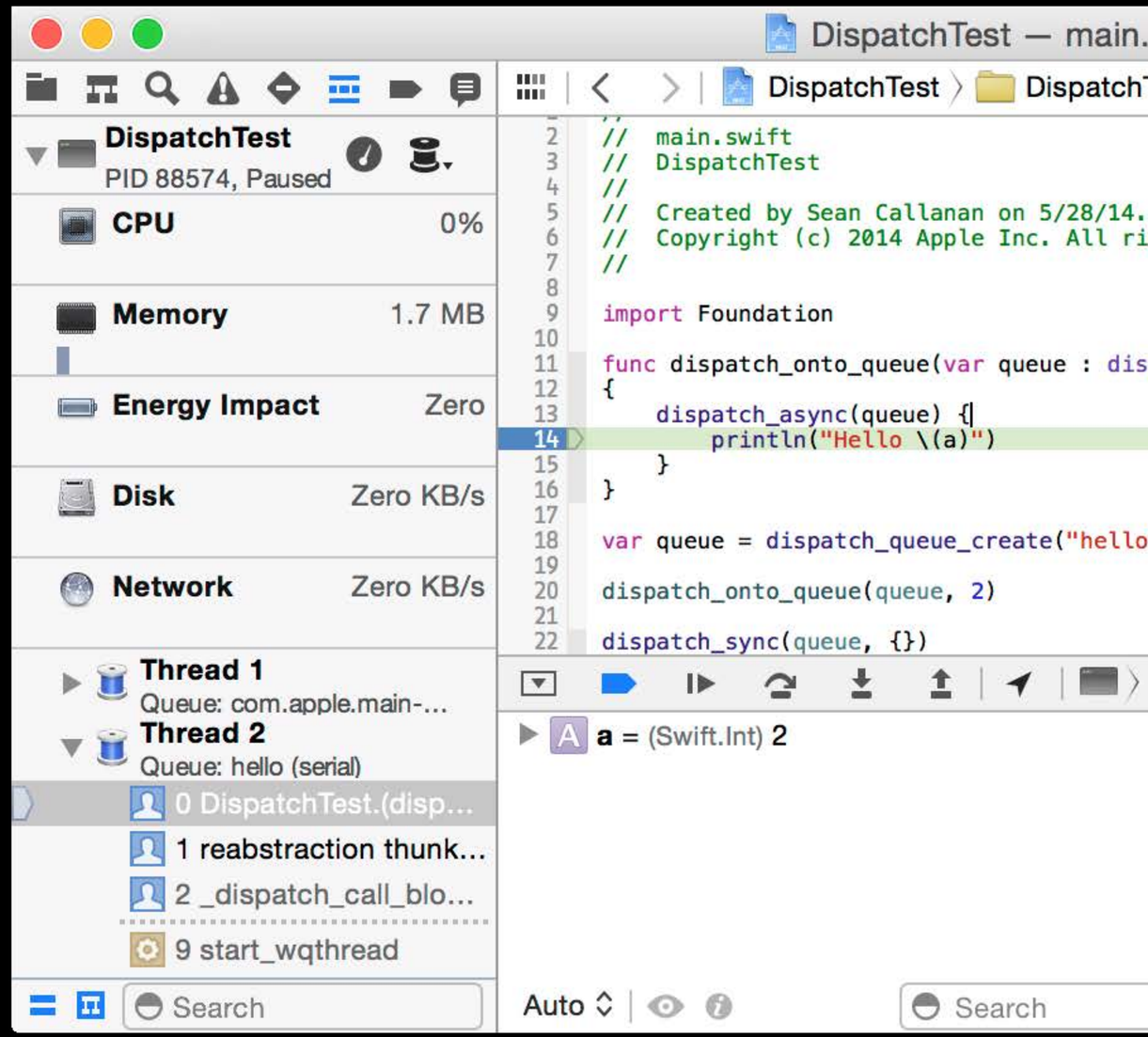
Session 409
Sean Callanan
Debugger Engineer

LLDB Is Better Than Ever with Swift



LLDB Is Better Than Ever with Swift

It helps you fix problems in your code!

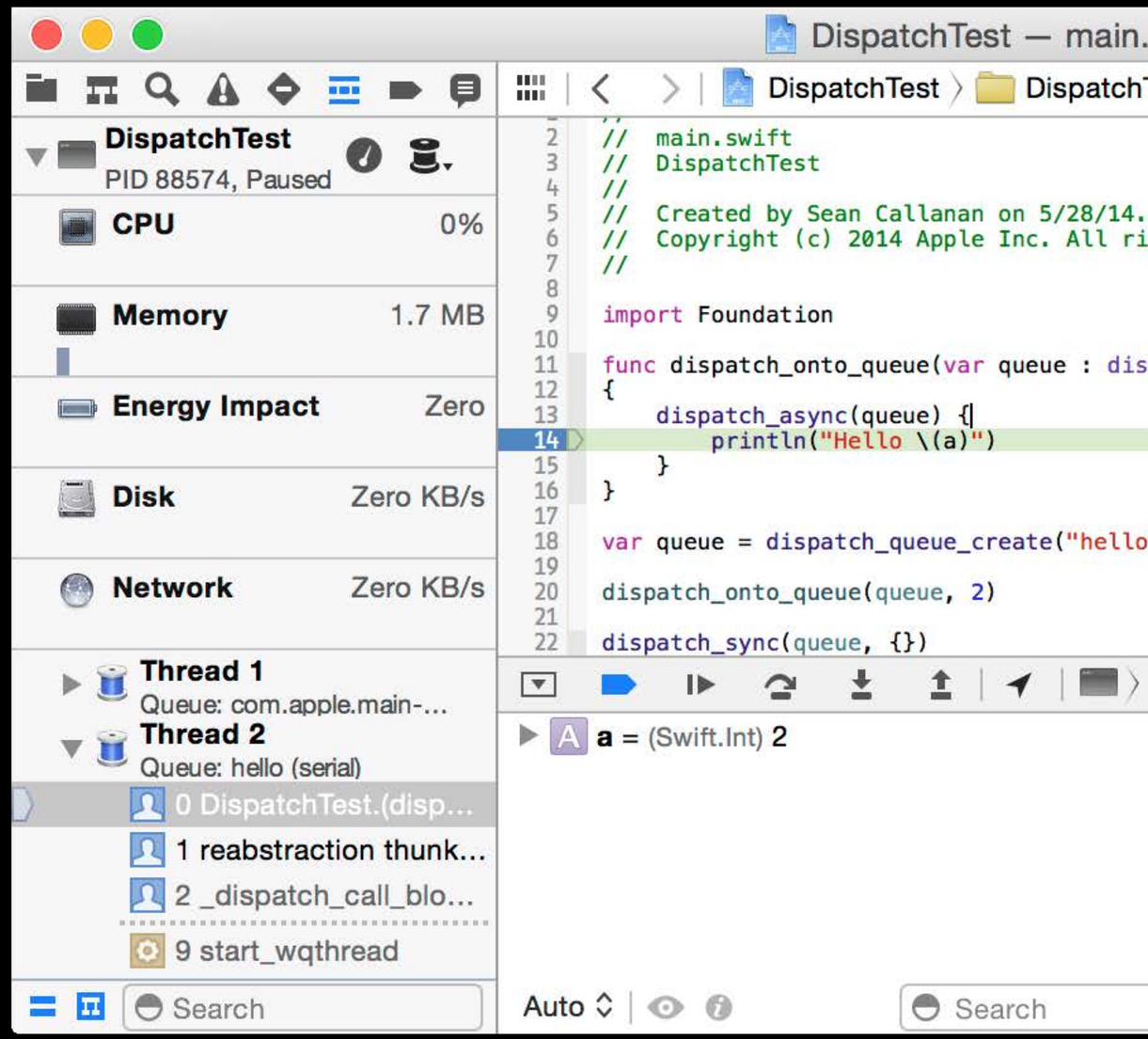


LLDB Is Better Than Ever with Swift

It helps you fix problems in your code!

LLDB has a variety of bug-fixing tools

- The stack diagnoses a stopped app
- Breakpoints stop an app when needed
- The `expr` command inspects data



LLDB Is Better Than Ever with Swift

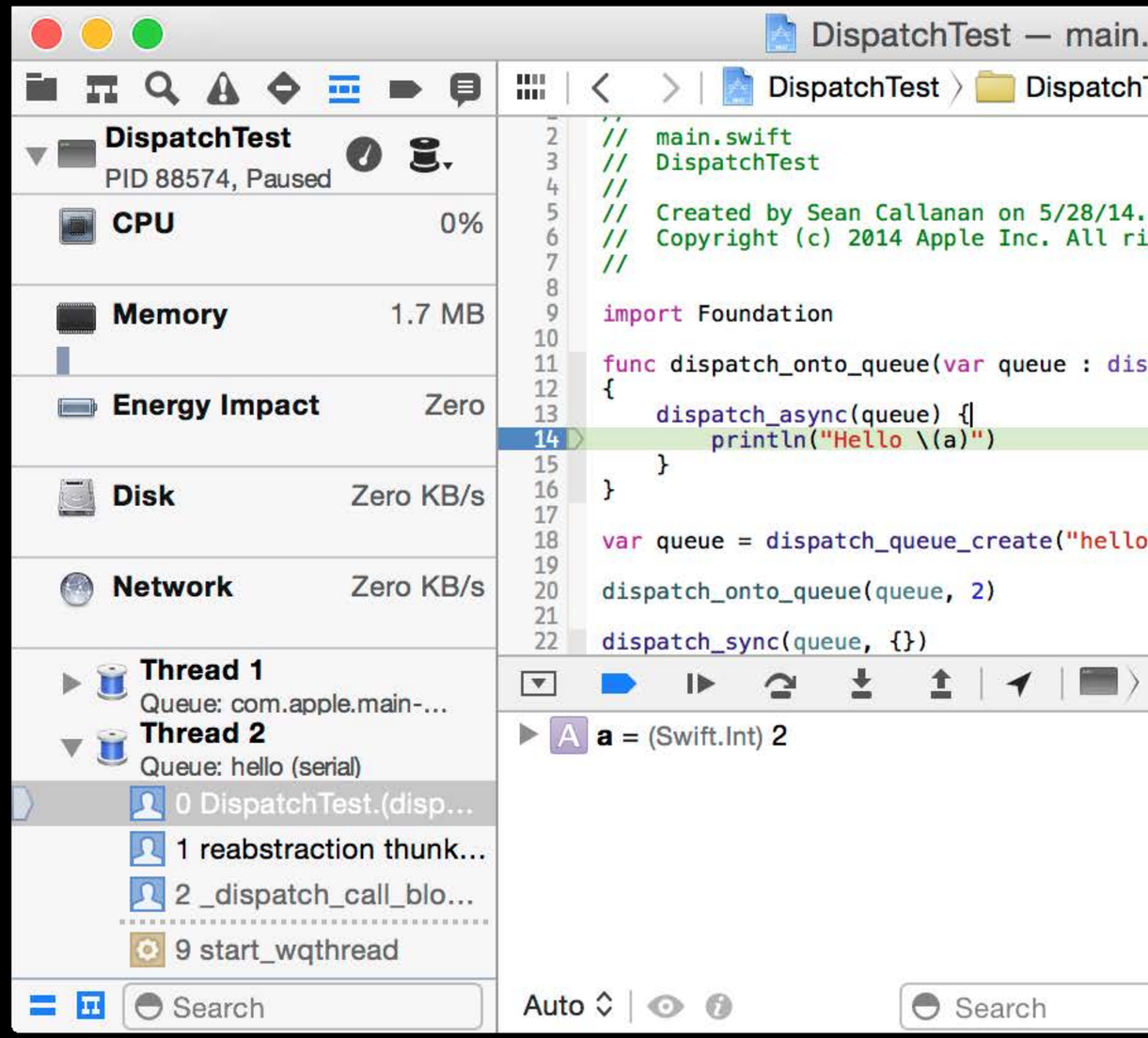
It helps you fix problems in your code!

LLDB has a variety of bug-fixing tools

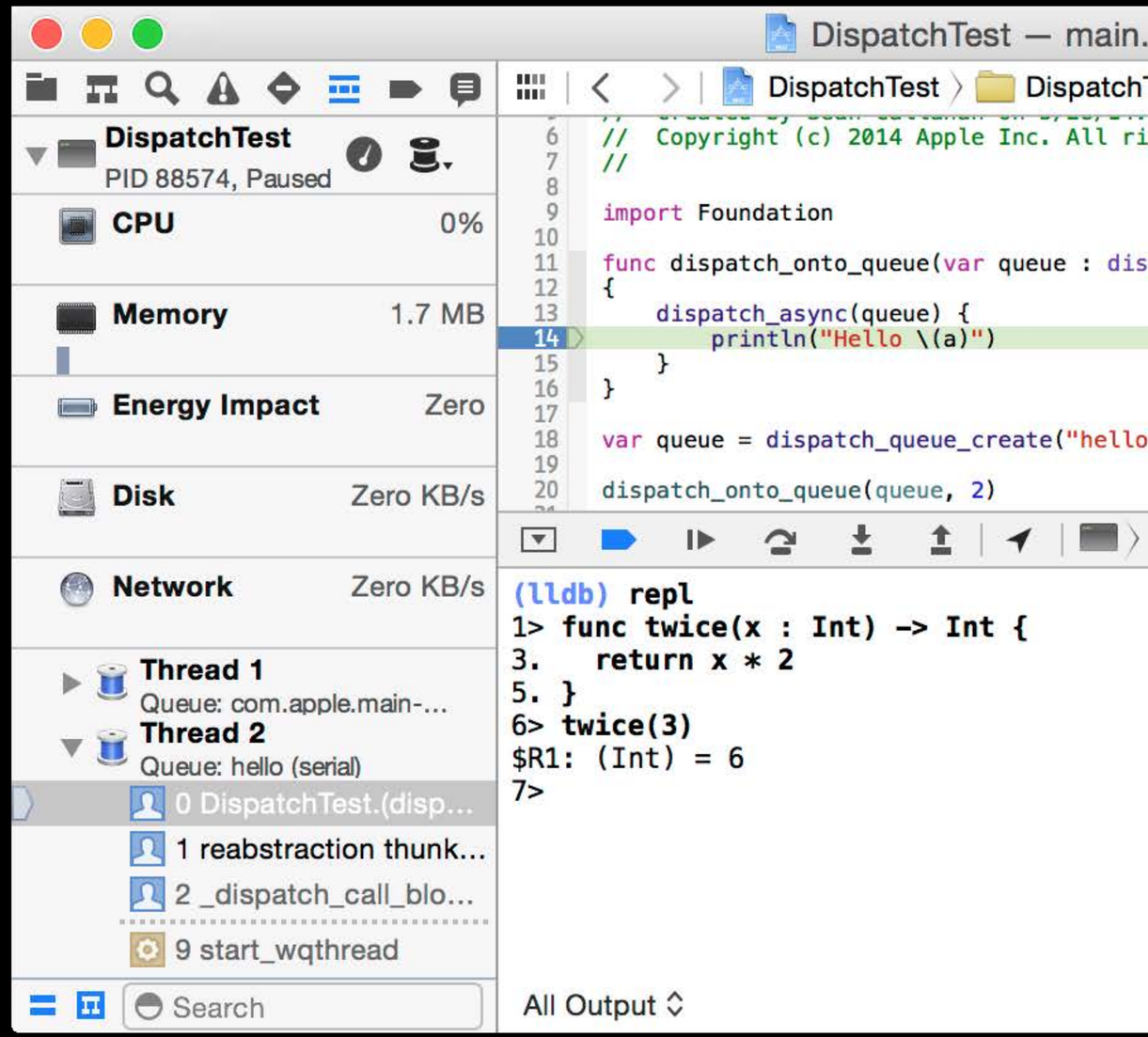
- The stack diagnoses a stopped app
- Breakpoints stop an app when needed
- The `expr` command inspects data

These tools work great with Swift!

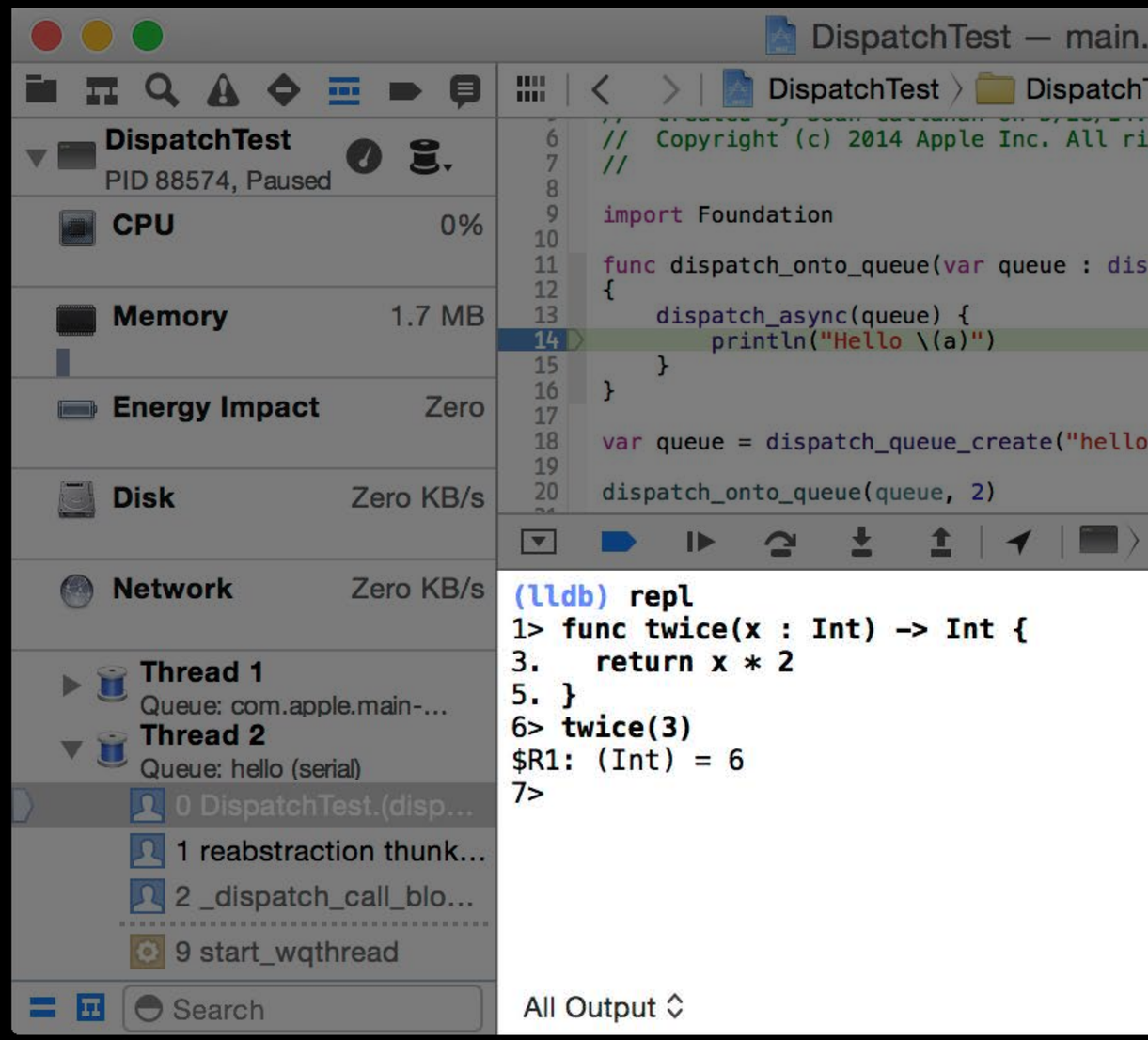
- Use them to find familiar bugs…
- …and some new ones!



…and Now There's a REPL, Too!

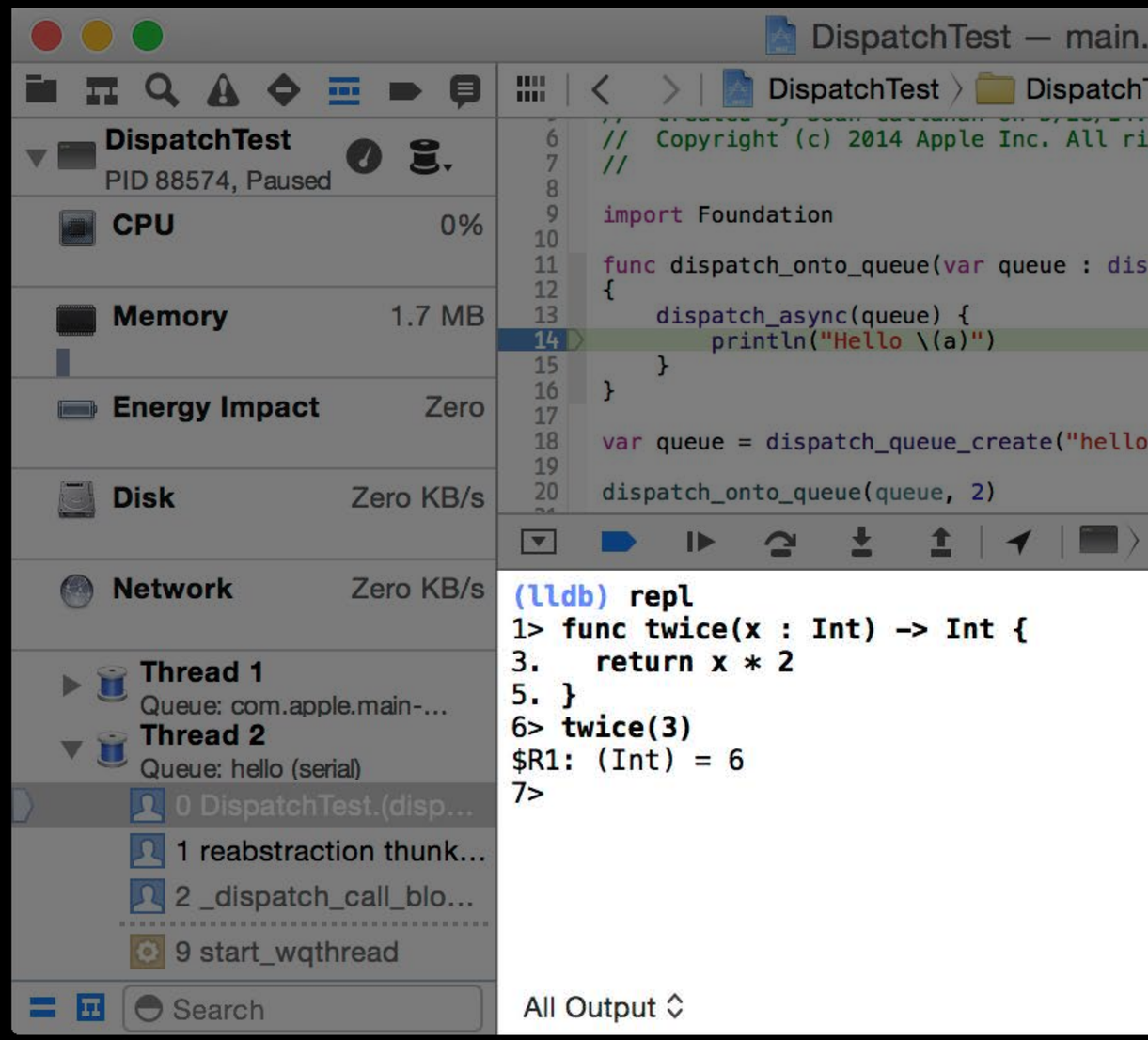


…and Now There's a REPL, Too!



…and Now There's a REPL, Too!

The Read-Eval-Print-Loop is built on LLDB

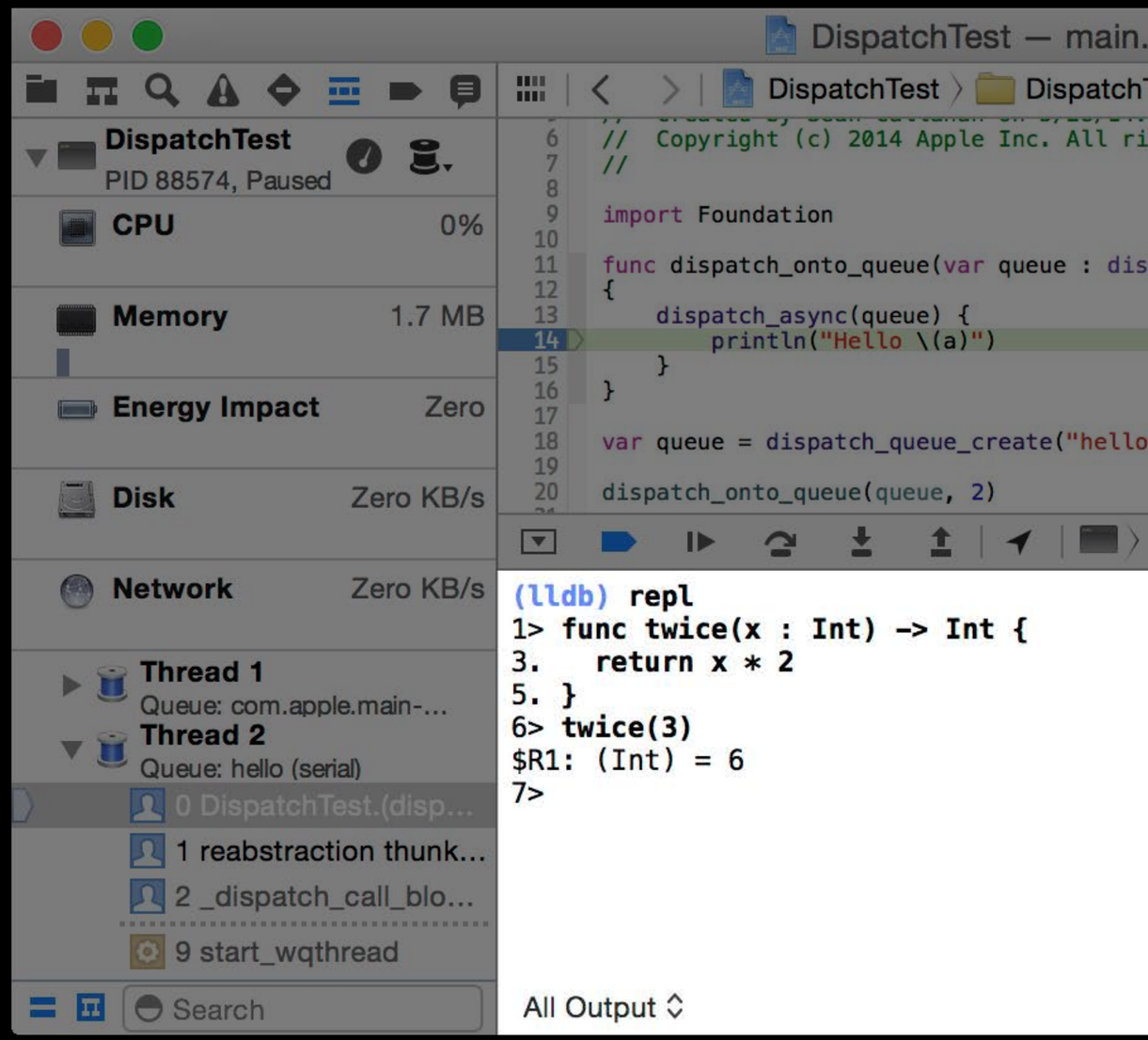


…and Now There's a REPL, Too!

The Read-Eval-Print-Loop is built on LLDB

You can use it anywhere

- (lldb) `repl` when your app is running
- `$ xcrun swift` for a clean slate



…and Now There's a REPL, Too!

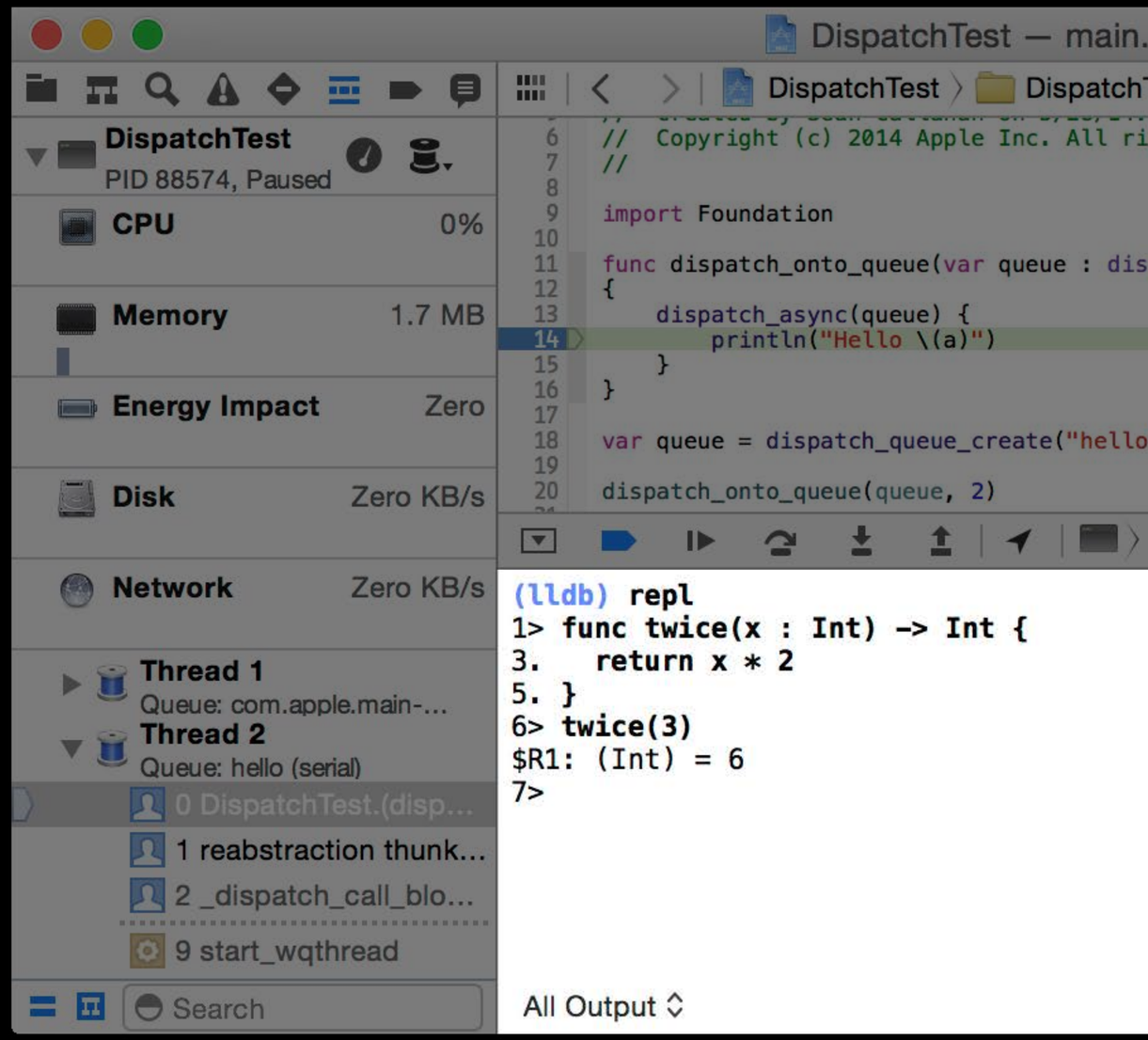
The Read-Eval-Print-Loop is built on LLDB

You can use it anywhere

- (lldb) `repl` when your app is running
- `$ xcrun swift` for a clean slate

We'll show you some of the ways to use it

- Test your app interactively
- Try out new code in your app



Where the REPL Fits In

Interactive development tools and their uses

The `expression` command	Call functions, edit data in scope
LLDB Python scripting	Create new debugger features
Playgrounds	Prototype from scratch
The LLDB REPL	Add code to an existing session

Overview of Today's Session

Overview of Today's Session

Basic debugging survival skills

- Reading a stopped app
- Stopping an app at the right time

Overview of Today's Session

Basic debugging survival skills

- Reading a stopped app
- Stopping an app at the right time

REPL-enabled debugging workflows

- Validating existing code
- Trying out new code

Overview of Today's Session

Basic debugging survival skills

- Reading a stopped app
- Stopping an app at the right time

REPL-enabled debugging workflows

- Validating existing code
- Trying out new code

Summing up

Overview of Today's Session

Basic debugging survival skills

- Reading a stopped app
- Stopping an app at the right time

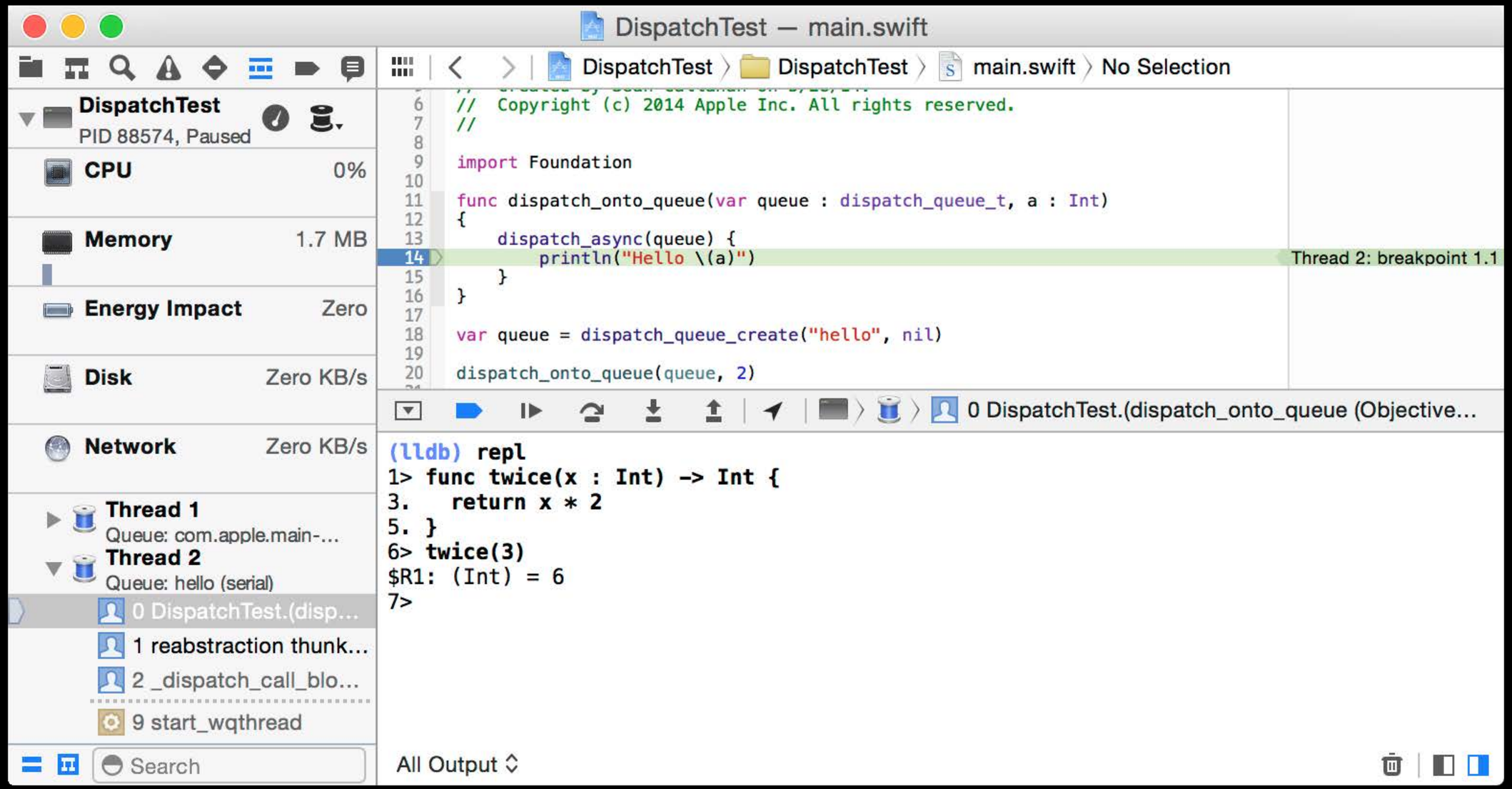
REPL-enabled debugging workflows

- Validating existing code
- Trying out new code

Summing up

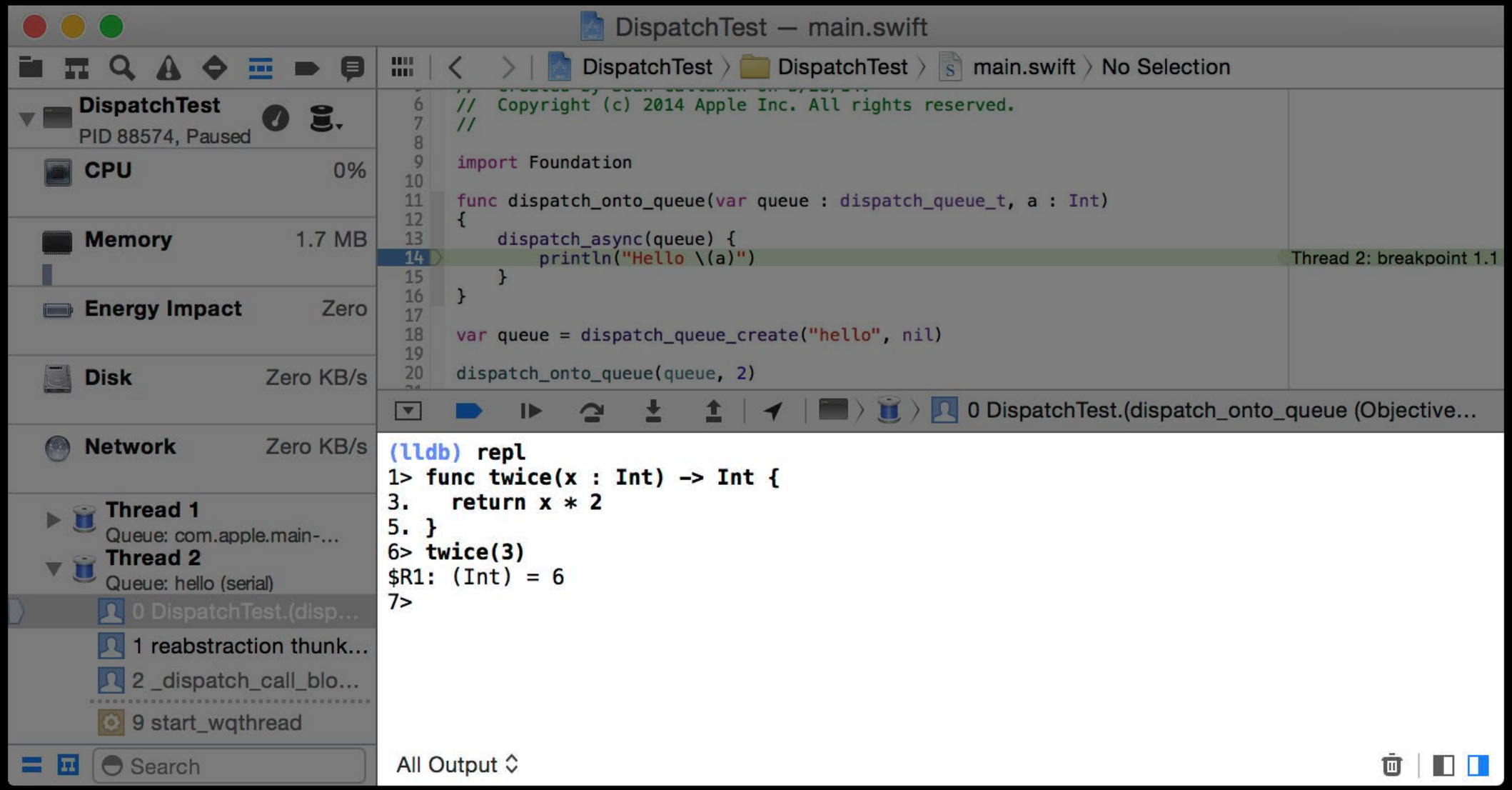
Basic Debugging Survival Skills

Interacting with LLDB



Basic Debugging Survival Skills

Interacting with LLDB



Basic Debugging Survival Skills

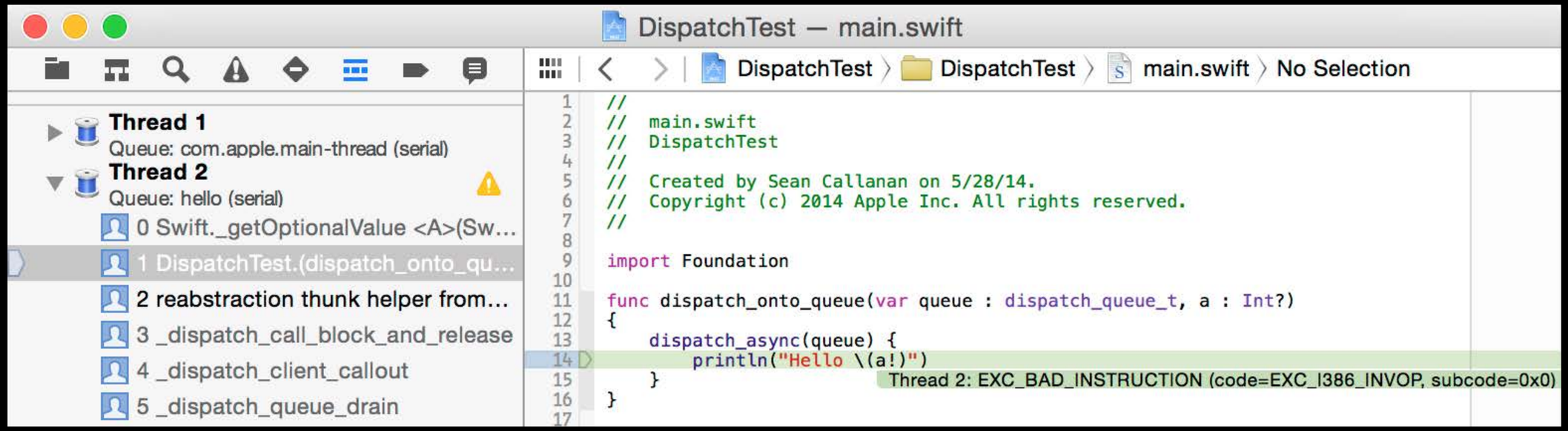
Interacting with LLDB

Debug — lldb — 80×24

```
(lldb) r
Process 78019 launched: '/Volumes/Data/WWDC 2014/DispatchTest/build/Debug/Dispat
chTest' (x86_64)
Process 78019 stopped
* thread #2: tid = 0x17f0a8, 0x0000000100001d2c DispatchTest`DispatchTest.(a=Swi
ft.Int at rdi) -> ()).(closure #1) + 28 at main.swift:14, queue = 'hello', stop
reason = breakpoint 1.1
    frame #0: 0x0000000100001d2c DispatchTest`DispatchTest.(a=Swift.Int at rdi)
-> ()).(closure #1) + 28 at main.swift:14
   11   func dispatch_onto_queue(var queue : dispatch_queue_t, a : Int)
   12   {
   13       dispatch_async(queue) {
-> 14           println("Hello \(a)")
   15       }
   16   }
   17
(lldb) repl
  1> func twice(x : Int) -> Int {
  2.   return x * 2
  3. }
  4> twice(3)
$R1: (Int) = 6
  5>
```

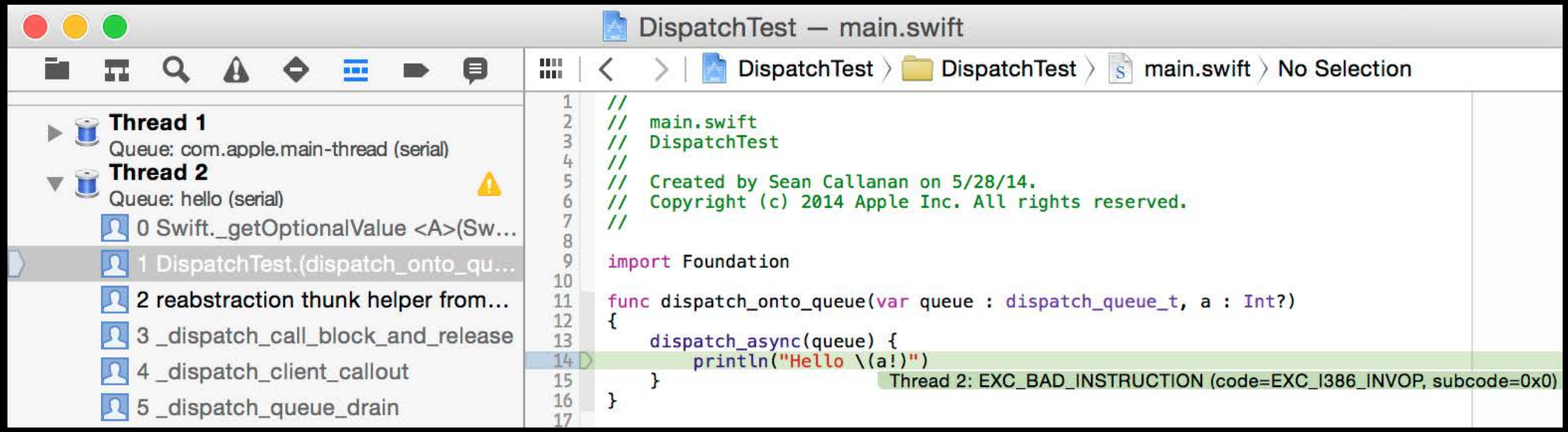
Reading a Stopped App

Questions to ask



Reading a Stopped App

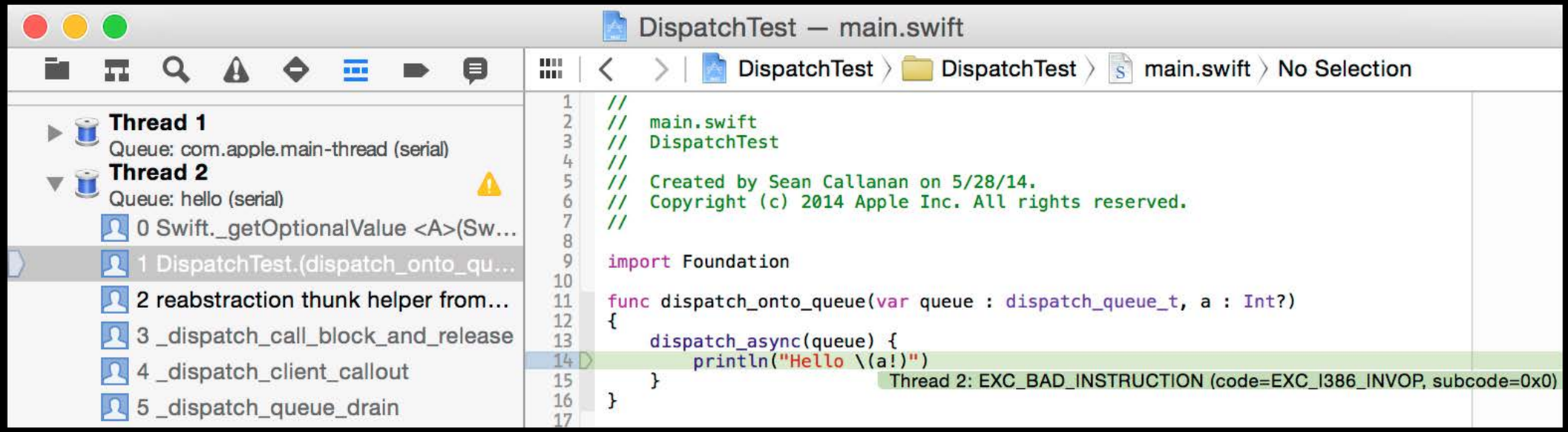
Questions to ask



What is the stop reason?

Reading a Stopped App

Questions to ask



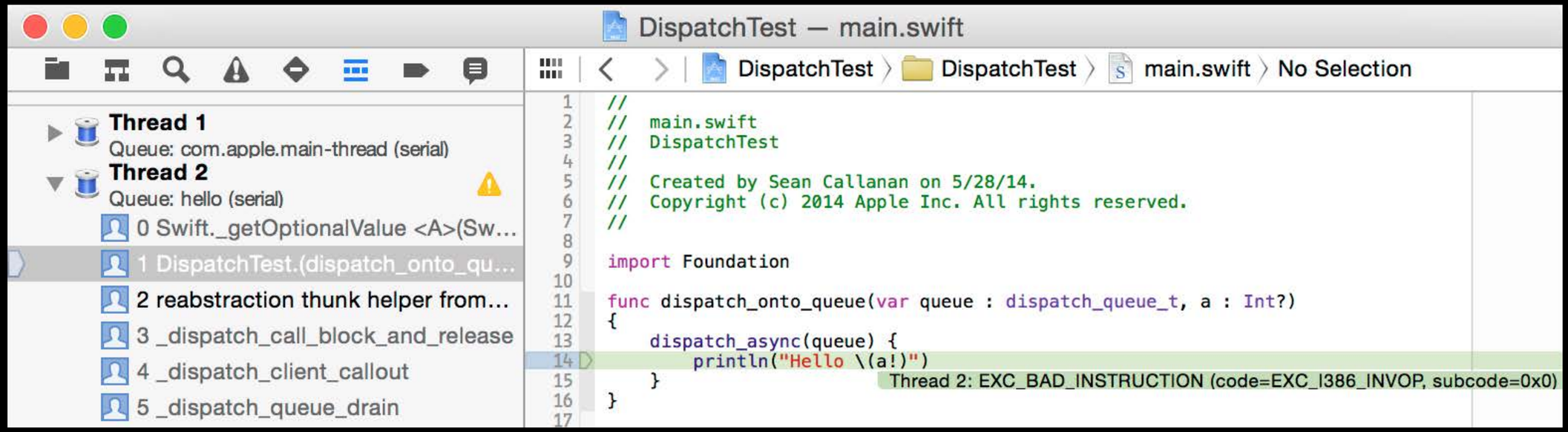
What is the stop reason?

How did this code get called?

- Look up the stack
- Find where things went bad

Reading a Stopped App

Questions to ask



What is the stop reason?

How did this code get called?

- Look up the stack
- Find where things went bad

What were the failure conditions?

- Find the problematic function
- Inspect variables to find bad data

Reading a Stopped App

Example one: The basics

What is the stop reason?

(lldb) **t i** thread info

```
* thread #1:
  tid = 0x91a7fa,
  0x100002208 accessing-nil`Swift._getOptionalValue <A>(Swift.Optional<A>) ->
A + 696 at accessing-nil.swift:0,
  queue = 'com.apple.main-thread',
  stop reason = EXC_BAD_INSTRUCTION (code=EXC_I386_INVOP, subcode=0x0)
```

Reading a Stopped App

Example one: The basics

What is the stop reason?

(lldb) **t i**

thread info

We're in standard library code

```
* thread #1:
  tid = 0x91a7fa,
    0x100002208 accessing-nil`Swift._getOptionalValue <A>(Swift.Optional<A>) ->
A + 696 at accessing-nil.swift:0,
  queue = 'com.apple.main-thread',
  stop reason = EXC_BAD_INSTRUCTION (code=EXC_I386_INVOP, subcode=0x0)
```

Reading a Stopped App

Example one: The basics

What is the stop reason?

(lldb) **t i**

thread info

We're in standard library code

```
* thread #1:
  tid = 0x91a7fa,
    0x100002208 accessing-nil`Swift._getOptionalValue <A>(Swift.Optional<A>) ->
A + 696 at accessing-nil.swift:0,
  queue = 'com.apple.main-thread',
  stop reason = EXC_BAD_INSTRUCTION (code=EXC_I386_INVOP, subcode=0x0)
```

Reading a Stopped App

Example one: The basics

What is the stop reason?

(lldb) **t i**

thread info

We're in standard library code

```
* thread #1:
  tid = 0x91a7fa,
  0x100002208 accessing-nil`Swift._getOptionalValue <A>(Swift.Optional<A>) ->
A + 696 at accessing-nil.swift:0,
  queue = 'com.apple.main-thread',
  stop reason = EXC_BAD_INSTRUCTION (code=EXC_I386_INVOP, subcode=0x0)
```

Bad instructions are typically used in assertions

Reading a Stopped App

Example one: The basics

What is the stop reason?

(lldb) **t i**

thread info

We're in standard library code

```
* thread #1:
  tid = 0x91a7fa,
  0x100002208 accessing-nil`Swift._getOptionalValue <A>(Swift.Optional<A>) ->
A + 696 at accessing-nil.swift:0,
  queue = 'com.apple.main-thread',
  stop reason = EXC_BAD_INSTRUCTION (code=EXC_I386_INVOP, subcode=0x0)
```

Bad instructions are typically used in assertions

This means that an assertion failed in the standard library!

Reading a Stopped App

Example one: The basics

How did this code get called?

(lldb) **bt** | thread backtrace |

```
…
  * frame #0: 0x100002208 accessing-nil`Swift._getOptionalValue
<A>(Swift.Optional<A>) -> A + 696 at accessing-nil.swift:0
    frame #1: 0x100001b02 accessing-nil`top_level_code + 114 at accessing-
nil.swift:6
    frame #2: 0x100001c8a accessing-nil`main + 42 at accessing-nil.swift:0
    frame #3: 0x7fff873995c9 libdyld.dylib`start + 1
```

Reading a Stopped App

Example one: The basics

How did this code get called?

(lldb) **bt** thread backtrace

```
...
  * frame #0: 0x100002208 accessing-nil`Swift._getOptionalValue
<A>(Swift.Optional<A>) -> A + 696 at accessing-nil.swift:0
    frame #1: 0x100001b02 accessing-nil`top_level_code + 114 at accessing-
nil.swift:6
    frame #2: 0x100001c8a accessing-nil`main + 42 at accessing-nil.swift:0
    frame #3: 0x7fff873995c9 libdyld.dylib`start + 1
```

Reading a Stopped App

Example one: The basics

How did this code get called?

(lldb) **bt** thread backtrace

```
...
  * frame #0: 0x100002208 accessing-nil`Swift._getOptionalValue
<A>(Swift.Optional<A>) -> A + 696 at accessing-nil.swift:0
    frame #1: 0x100001b02 accessing-nil`top_level_code + 114 at accessing-
nil.swift:6
    frame #2: 0x100001c8a accessing-nil`main + 42 at accessing-nil.swift:0
    frame #3: 0x7fff873995c9 libdyld.dylib`start + 1
```

Our code called directly into the standard library

Reading a Stopped App

Example one: The basics

What were the failure conditions?

(lldb) **f 1** | frame select 1 |

```
frame #1: 0x100001b02 accessing-nil`top_level_code + 114 at accessing-
nil.swift:6
   3  }
   4
   5  var foo : AnyObject? = ReturnsNil()
-> 6  var bar : AnyObject = foo!
```

Reading a Stopped App

Example one: The basics

What were the failure conditions?

(lldb) **f 1** | frame select 1 |

```
frame #1: 0x100001b02 accessing-nil`top_level_code + 114 at accessing-
nil.swift:6
   3  }
   4
   5  var foo : AnyObject? = ReturnsNil()
-> 6  var bar : AnyObject = foo!
```

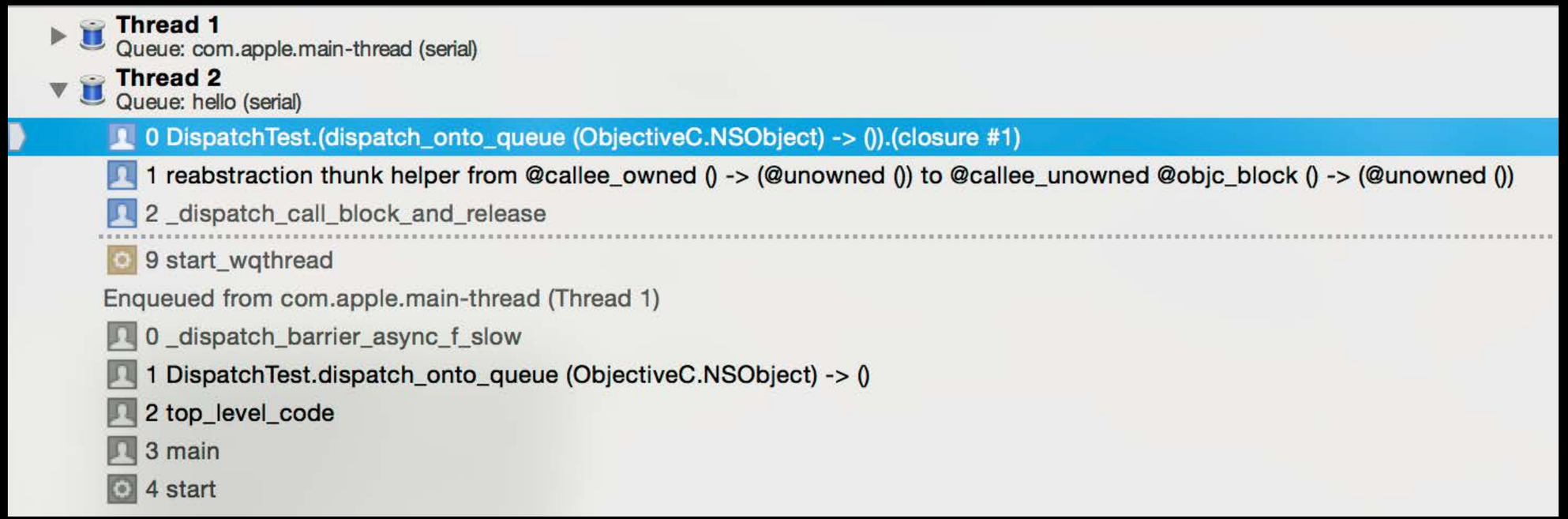
(lldb) **p foo** | expression foo |

```
(AnyObject?) $R1 = None
```

Reading a Stopped App

Stacks and libdispatch

If you're using Xcode, then you can now see where blocks got dispatched!



Reading a Stopped App

Example two: More Swift assertions

What is the stop reason? How did the code get called?

(lldb) **bt**

```
* … stop reason = EXC_BAD_INSTRUCTION (code=EXC_I386_INVOP, subcode=0x0)
  * frame #0: 0x10003cb57
libswift_stdlib_core.dylib`Swift.Array.subscript.getter (Swift.Int) -> A +
167
    frame #1: 0x100001047 out-of-bounds`main.FindElement <A>($return_value=T?
at 0x00007fff5fbffb18, array=T[] at 0x00007fff5fbffb40,
matches=<unavailable>) -> Swift.Bool) -> Swift.Optional<A> + 679 at out-of-
bounds.swift:3
    frame #2: 0x100000d9a out-of-bounds`top_level_code + 362 at out-of-
bounds.swift:11
    frame #3: 0x10000136a out-of-bounds`main + 42 at out-of-bounds.swift:0
    frame #4: 0x7fff873995c9 libdyld.dylib`start + 1
```

Reading a Stopped App

Example two: More Swift assertions

What is the stop reason? How did the code get called?

(lldb) **bt**

```
* … stop reason = EXC_BAD_INSTRUCTION (code=EXC_I386_INVOP, subcode=0x0)
  * frame #0: 0x10003cb57
libswift_stdlib_core.dylib`Swift.Array.subscript.getter (Swift.Int) -> A +
167
    frame #1: 0x100001047 out-of-bounds`main.FindElement <A>($return_value=T?
at 0x00007fff5fbffb18, array=T[] at 0x00007fff5fbffb40,
matches=<unavailable>) -> Swift.Bool) -> Swift.Optional<A> + 679 at out-of-
bounds.swift:3
    frame #2: 0x100000d9a out-of-bounds`top_level_code + 362 at out-of-
bounds.swift:11
    frame #3: 0x10000136a out-of-bounds`main + 42 at out-of-bounds.swift:0
    frame #4: 0x7fff873995c9 libdyld.dylib`start + 1
```

Reading a Stopped App

Example two: More Swift assertions

What is the stop reason? How did the code get called?

(lldb) **bt**

```
* … stop reason = EXC_BAD_INSTRUCTION (code=EXC_I386_INVOP, subcode=0x0)
  * frame #0: 0x10003cb57
libswift_stdlib_core.dylib`Swift.Array.subscript.getter (Swift.Int) -> A +
167
    frame #1: 0x100001047 out-of-bounds`main.FindElement <A>($return_value=T?
at 0x00007fff5fbffb18, array=T[] at 0x00007fff5fbffb40,
matches=<unavailable>) -> Swift.Bool) -> Swift.Optional<A> + 679 at out-of-
bounds.swift:3
    frame #2: 0x100000d9a out-of-bounds`top_level_code + 362 at out-of-
bounds.swift:11
    frame #3: 0x10000136a out-of-bounds`main + 42 at out-of-bounds.swift:0
    frame #4: 0x7fff873995c9 libdyld.dylib`start + 1
```

Reading a Stopped App

Example two: More Swift assertions

What is the stop reason? How did the code get called?

(lldb) **bt**

```
* … stop reason = EXC_BAD_INSTRUCTION (code=EXC_I386_INVOP, subcode=0x0)
  * frame #0: 0x10003cb57
libswift_stdlib_core.dylib`Swift.Array.subscript.getter (Swift.Int) -> A +
167
    frame #1: 0x100001047 out-of-bounds`main.FindElement <A>($return_value=T?
at 0x00007fff5fbffb18, array=T[] at 0x00007fff5fbffb40,
matches=<unavailable>) -> Swift.Bool) -> Swift.Optional<A> + 679 at out-of-
bounds.swift:3
    frame #2: 0x100000d9a out-of-bounds`top_level_code + 362 at out-of-
bounds.swift:11
    frame #3: 0x10000136a out-of-bounds`main + 42 at out-of-bounds.swift:0
    frame #4: 0x7fff873995c9 libdyld.dylib`start + 1
```

Reading a Stopped App

Example two: More Swift assertions

What were the failure conditions?

(lldb) **f 1**

```
   1 func FindElement<T>(var array : Array<T>, var match : T -> Bool) -> T? {
   2   for index in 0...array.count {
-> 3       if (match(array[index])) {
   4         return array[index]
   5       }
   6   }
   7   return nil
```

Reading a Stopped App

Example two: More Swift assertions

What were the failure conditions?

(lldb) **f 1**

```
   1 func FindElement<T>(var array : Array<T>, var match : T -> Bool) -> T? {
   2   for index in 0...array.count {
-> 3       if (match(array[index])) {
   4         return array[index]
   5       }
   6   }
   7   return nil
```

(lldb) **p index**

```
(RangeGenerator<Int>) $R1 = (startIndex = 4, endIndex = 4)
```

Reading a Stopped App

Example two: More Swift assertions

What were the failure conditions?

(lldb) **f 1**

```
   1 func FindElement<T>(var array : Array<T>, var match : T -> Bool) -> T? {
   2   for index in 0...array.count {
-> 3       if (match(array[index])) {
   4         return array[index]
   5       }
   6   }
   7   return nil
```

(lldb) **p index**

```
(RangeGenerator<Int>) $R1 = (startIndex = 4, endIndex = 4)
```

(lldb) **p array.count**

```
(Int) $R2 = 3
```

Reading a Stopped App

Example two: More Swift assertions

What were the failure conditions?

(lldb) **f 1**

```
   1 func FindElement<T>(var array : Array<T>, var match : T -> Bool) -> T? {
   2   for index in 0...array.count {
-> 3       if (match(array[index])) {
   4         return array[index]
   5       }
   6   }
   7   return nil
```

Use `...` to avoid the last index

Or just use `for element in array`

(lldb) **p index**

```
(RangeGenerator<Int>) $R1 = (startIndex = 4, endIndex = 4)
```

(lldb) **p array.count**

```
(Int) $R2 = 3
```

Reading a Stopped App

Example three: Exceptions in Objective-C

(lldb) **bt**

```
* thread #1: tid = (…), 0x7fff8995337a libsystem_kernel.dylib`__pthread_kill + 10, queue =
'com.apple.main-thread', stop reason = signal SIGABRT
* frame #0: 0x7fff8995337a libsystem_kernel.dylib`__pthread_kill + 10
  frame #1: 0x7fff876699bb libsystem_pthread.dylib`pthread_kill + 90
  frame #2: 0x7fff876345ff libsystem_c.dylib`abort + 129
  frame #3: 0x7fff95203b31 libc++abi.dylib`abort_message + 257
  frame #4: 0x7fff9522b9d1 libc++abi.dylib`default_terminate_handler() + 267
  frame #5: 0x7fff8f49350d libobjc.A.dylib`_objc_terminate() + 103
  frame #6: 0x7fff952290a1 libc++abi.dylib`std::__terminate(void (*)()) + 8
  frame #7: 0x7fff95228b2c libc++abi.dylib`__cxa_throw + 121
  frame #8: 0x7fff8f48f6a7 libobjc.A.dylib`objc_exception_throw + 341
  frame #9: 0x7fff93616cce CoreFoundation`-[__NSArrayI objectAtIndex:] + 190
  frame #10: 0x100001baf nsarray`nsarray.FindElement <A>(…)
  frame #11: 0x100001855 nsarray`top_level_code + 693 at nsarray.swift:15
  frame #12: 0x10000225a nsarray`main + 42 at nsarray.swift:0
  frame #13: 0x7fff873995c9 libdyld.dylib`start + 1
```

Reading a Stopped App

Example three: Exceptions in Objective-C

(lldb) **bt**

```
* thread #1: tid = (…), 0x7fff8995337a libsystem_kernel.dylib`__pthread_kill + 10, queue =
'com.apple.main-thread', stop reason = signal SIGABRT
* frame #0: 0x7fff8995337a libsystem_kernel.dylib`__pthread_kill + 10
  frame #1: 0x7fff876699bb libsystem_pthread.dylib`pthread_kill + 90
  frame #2: 0x7fff876345ff libsystem_c.dylib`abort + 129
  frame #3: 0x7fff95203b31 libc++abi.dylib`abort_message + 257
  frame #4: 0x7fff9522b9d1 libc++abi.dylib`default_terminate_handler() + 267
  frame #5: 0x7fff8f49350d libobjc.A.dylib`_objc_terminate() + 103
  frame #6: 0x7fff952290a1 libc++abi.dylib`std::__terminate(void (*)()) + 8
  frame #7: 0x7fff95228b2c libc++abi.dylib`__cxa_throw + 121
  frame #8: 0x7fff8f48f6a7 libobjc.A.dylib`objc_exception_throw + 341
  frame #9: 0x7fff93616cce CoreFoundation`-[__NSArrayI objectAtIndex:] + 190
  frame #10: 0x100001baf nsarray`nsarray.FindElement <A>(…)
  frame #11: 0x100001855 nsarray`top_level_code + 693 at nsarray.swift:15
  frame #12: 0x10000225a nsarray`main + 42 at nsarray.swift:0
  frame #13: 0x7fff873995c9 libdyld.dylib`start + 1
```

Reading a Stopped App

Example three: Exceptions in Objective-C

(lldb) **bt**

```
* thread #1: tid = (…), 0x7fff8995337a libsystem_kernel.dylib`__pthread_kill + 10, queue =
'com.apple.main-thread', stop reason = signal SIGABRT
* frame #0: 0x7fff8995337a libsystem_kernel.dylib`__pthread_kill + 10
  frame #1: 0x7fff876699bb libsystem_pthread.dylib`pthread_kill + 90
  frame #2: 0x7fff876345ff libsystem_c.dylib`abort + 129
  frame #3: 0x7fff95203b31 libc++abi.dylib`abort_message + 257
  frame #4: 0x7fff9522b9d1 libc++abi.dylib`default_terminate_handler() + 267
  frame #5: 0x7fff8f49350d libobjc.A.dylib`_objc_terminate() + 103
  frame #6: 0x7fff952290a1 libc++abi.dylib`std::__terminate(void (*)()) + 8
  frame #7: 0x7fff95228b2c libc++abi.dylib`__cxa_throw + 121
  frame #8: 0x7fff8f48f6a7 libobjc.A.dylib`objc_exception_throw + 341
  frame #9: 0x7fff93616cce CoreFoundation`-[__NSArrayI objectAtIndex:] + 190
  frame #10: 0x100001baf nsarray`nsarray.FindElement <A>(…)
  frame #11: 0x100001855 nsarray`top_level_code + 693 at nsarray.swift:15
  frame #12: 0x10000225a nsarray`main + 42 at nsarray.swift:0
  frame #13: 0x7fff873995c9 libdyld.dylib`start + 1
```

Reading a Stopped App

Example three: Exceptions in Objective-C

(lldb) **bt**

```
* thread #1: tid = (…), 0x7fff8995337a libsystem_kernel.dylib`__pthread_kill + 10, queue =
'com.apple.main-thread', stop reason = signal SIGABRT
* frame #0: 0x7fff8995337a libsystem_kernel.dylib`__pthread_kill + 10
  frame #1: 0x7fff876699bb libsystem_pthread.dylib`pthread_kill + 90
  frame #2: 0x7fff876345ff libsystem_c.dylib`abort + 129
  frame #3: 0x7fff95203b31 libc++abi.dylib`abort_message + 257
  frame #4: 0x7fff9522b9d1 libc++abi.dylib`default_terminate_handler() + 267
  frame #5: 0x7fff8f49350d libobjc.A.dylib`_objc_terminate() + 103
  frame #6: 0x7fff952290a1 libc++abi.dylib`std::__terminate(void (*)()) + 8
  frame #7: 0x7fff95228b2c libc++abi.dylib`__cxa_throw + 121
  frame #8: 0x7fff8f48f6a7 libobjc.A.dylib`objc_exception_throw + 341
  frame #9: 0x7fff93616cce CoreFoundation`-[__NSArrayI objectAtIndex:] + 190
  frame #10: 0x100001baf nsarray`nsarray.FindElement <A>(…)
  frame #11: 0x100001855 nsarray`top_level_code + 693 at nsarray.swift:15
  frame #12: 0x10000225a nsarray`main + 42 at nsarray.swift:0
  frame #13: 0x7fff873995c9 libdyld.dylib`start + 1
```

Reading a Stopped App

Example three: Exceptions in Objective-C

(lldb) **f 10**

```
  2
  3 func FindElement<T>(var array : NSArray, var matches : T -> Bool) -> T? {
  4   for index in 0...array.count {
->5       if let elementAsT = array[index] as? T {
  6         if matches(elementAsT) {
  7           return elementAsT
  8         }
```

Reading a Stopped App

Example three: Exceptions in Objective-C

(lldb) **f 10**

```
  2
  3 func FindElement<T>(var array : NSArray, var matches : T -> Bool) -> T? {
  4   for index in 0...array.count {
->5       if let elementAsT = array[index] as? T {
  6         if matches(elementAsT) {
  7           return elementAsT
  8         }
```

Reading a Stopped App

Example four: The more things change…

(lldb) **bt**

```
* thread #1: tid = (…), 0x1001ef7cd libswift_stdlib_core.dylib`initializeWithCopy value
witness for Swift.Character + 13, queue = 'com.apple.main-thread', stop reason =
EXC_BAD_ACCESS (code=1, address=0x0)
* frame #0: 0x1001ef7cd libswift_stdlib_core.dylib`
                  initializeWithCopy value witness for Swift.Character + 13
  frame #1: 0x100001e9a null-pointer`
                  Swift.UnsafePointer.memory.getter : A + 26
  frame #2: 0x100001966 null-pointer`
                  Swift.UnsafePointer.subscript.getter (Swift.Int) -> A + 54
  frame #3: 0x10000167b null-pointer`
                  top_level_code + 299 at null-pointer.swift:4
  frame #4: 0x10000171a
                  null-pointer`main + 42 at null-pointer.swift:0
```

Reading a Stopped App

Example four: The more things change…

(lldb) **bt**

```
* thread #1: tid = (…), 0x1001ef7cd libswift_stdlib_core.dylib`initializeWithCopy value
witness for Swift.Character + 13, queue = 'com.apple.main-thread', stop reason =
EXC_BAD_ACCESS (code=1, address=0x0)
* frame #0: 0x1001ef7cd libswift_stdlib_core.dylib`
                  initializeWithCopy value witness for Swift.Character + 13
  frame #1: 0x100001e9a null-pointer`
                  Swift.UnsafePointer.memory.getter : A + 26
  frame #2: 0x100001966 null-pointer`
                  Swift.UnsafePointer.subscript.getter (Swift.Int) -> A + 54
  frame #3: 0x10000167b null-pointer`
                  top_level_code + 299 at null-pointer.swift:4
  frame #4: 0x10000171a
                  null-pointer`main + 42 at null-pointer.swift:0
```

Reading a Stopped App

Example four: The more things change…

(lldb) **bt**

```
* thread #1: tid = (…), 0x1001ef7cd libswift_stdlib_core.dylib`initializeWithCopy value
witness for Swift.Character + 13, queue = 'com.apple.main-thread', stop reason =
EXC_BAD_ACCESS (code=1, address=0x0)
* frame #0: 0x1001ef7cd libswift_stdlib_core.dylib`
                    initializeWithCopy value witness for Swift.Character + 13
  frame #1: 0x100001e9a null-pointer`
                    Swift.UnsafePointer.memory.getter : A + 26
  frame #2: 0x100001966 null-pointer`
                    Swift.UnsafePointer.subscript.getter (Swift.Int) -> A + 54
  frame #3: 0x10000167b null-pointer`
                    top_level_code + 299 at null-pointer.swift:4
  frame #4: 0x10000171a
                    null-pointer`main + 42 at null-pointer.swift:0
```

Reading a Stopped App

Example four: The more things change…

(lldb) **f 3**

```
frame #3: 0x000000010000167b null-pointer`top_level_code + 299 at null-
pointer.swift:4
   1 import Foundation
   2 let empty_stuff =
       NSData(contentsOfURL:NSURL(string:"http://non.existent.web.site.com"))
   3 let chars = UnsafePointer<Character>(empty_stuff.bytes)
->4 let char = chars[0]
   5 println(char)
```

Reading a Stopped App

Example four: The more things change…

(lldb) **f 3**

```
frame #3: 0x000000010000167b null-pointer`top_level_code + 299 at null-
pointer.swift:4
   1 import Foundation
   2 let empty_stuff =
         NSData(contentsOfURL:NSURL(string:"http://non.existent.web.site.com"))
   3 let chars = UnsafePointer<Character>(empty_stuff.bytes)
->4 let char = chars[0]
   5 println(char)
```

(lldb) **p chars**

```
(UnsafePointer<Character>) $R1 = (value = Builtin.RawPointer =
0x0000000000000000)
```

Reading a Stopped App

Example four: The more things change…

(lldb) **f 3**

```
frame #3: 0x000000010000167b null-pointer`top_level_code + 299 at null-
pointer.swift:4
   1 import Foundation
   2 let empty_stuff =
         NSData(contentsOfURL:NSURL(string:"http://non.existent.web.site.com"))
   3 let chars = UnsafePointer<Character>(empty_stuff.bytes)
->4 let char = chars[0]
   5 println(char)
```

(lldb) **p chars**

```
(UnsafePointer<Character>) $R1 = (value = Builtin.RawPointer =
0x0000000000000000)
```

Swift code is safe by default, but be careful with APIs that use unsafe pointers!

Reading a Stopped App

Summing up

The stop reason tells you what happened

`EXC_BAD_INSTRUCTION`	Assertion
`SIGABRT`	Exception (usually Objective-C)
`EXC_BAD_ACCESS`	Memory error

The stack tells you how it happened

The `expression` command helps you inspect variables

Overview of Today's Session

Basic debugging survival skills

- Reading a stopped app
- Stopping an app at the right time

REPL-enabled debugging workflows

- Validating existing code
- Trying out new code

Summing up

Stopping an App at the Right Time

Breakpoint concepts

Stopping an App at the Right Time

Breakpoint concepts

Use a breakpoint if any of these are true

- The problem isn’t a crash
- The cause of a crash isn’t on the stack
- You want to step through failing code

Stopping an App at the Right Time

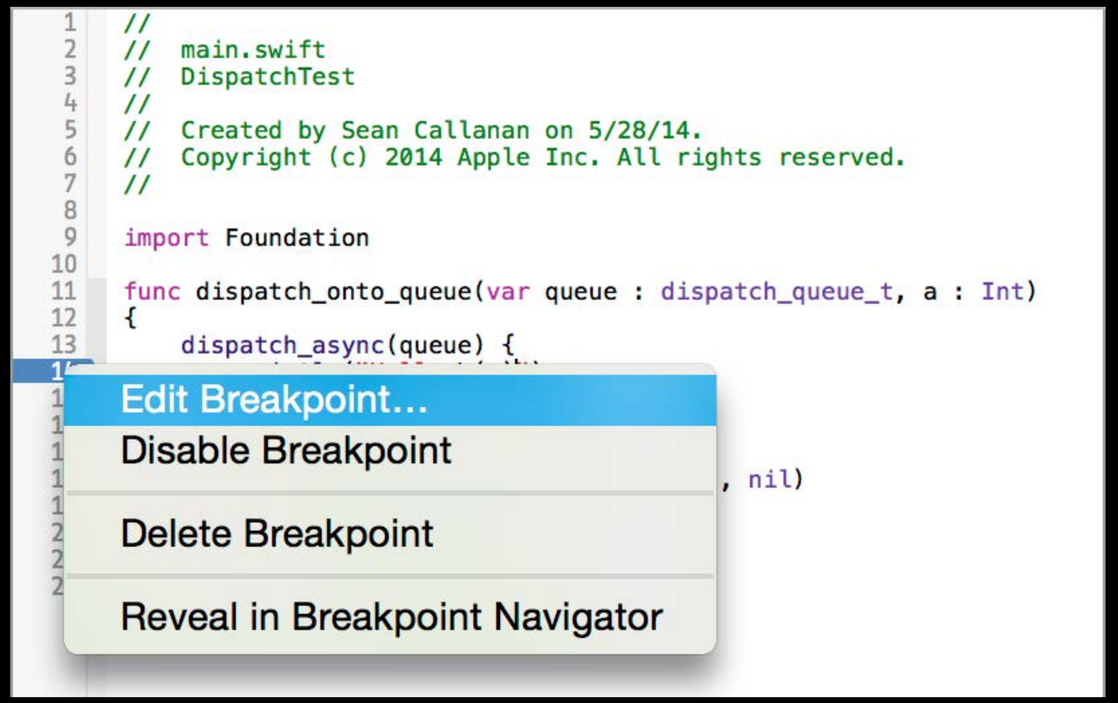
Breakpoint concepts

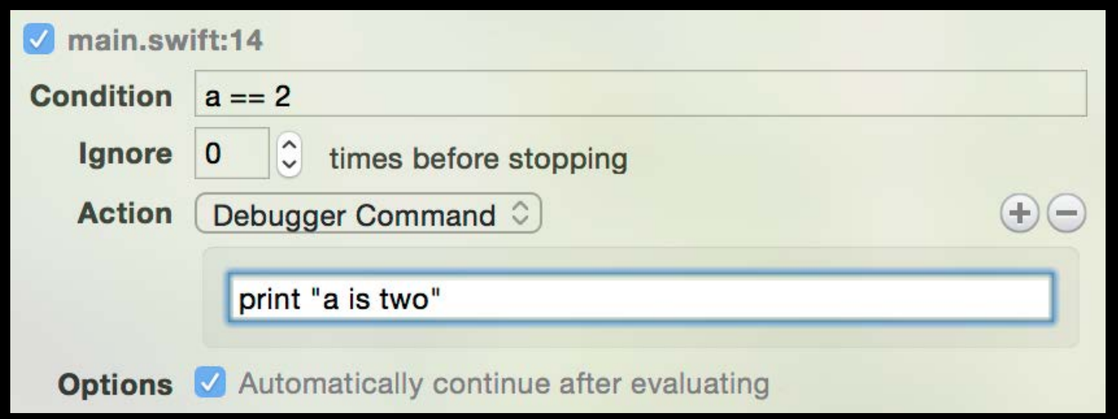
Use a breakpoint if any of these are true

- The problem isn't a crash
- The cause of a crash isn't on the stack
- You want to step through failing code

Important parts of a breakpoint

- Specification of where to stop
- Locations matching your specification
- Condition (optional)—Stop only if true
- Actions (optional) to perform when hit





Stopping an App at the Right Time

Example one: File-and-line breakpoints

```
1 var overdraftFee : UInt = 10
2 class Account {
3   var valueInCents : Int = 0
4   func withdraw(var amountInCents:UInt) {
5     if (Int(amountInCents) > self.valueInCents) {
6       valueInCents -= Int(overdraftFee)
7     }
8     valueInCents -= Int(amountInCents)
9   }
10  func deposit(var amountInCents:UInt) {
11    valueInCents += Int(amountInCents)
12  }
13 }
```

Stopping an App at the Right Time

Example one: File-and-line breakpoints

```
1  var overdraftFee : UInt = 10
2  class Account {
3    var valueInCents : Int = 0
4    func withdraw(var amountInCents:UInt) {
5      if (Int(amountInCents) > self.valueInCents) {
6        valueInCents -= Int(overdraftFee)
7      }
8      valueInCents -= Int(amountInCents)
9    }
10   func deposit(var amountInCents:UInt) {
11     valueInCents += Int(amountInCents)
12   }
13 }
```

Stopping an App at the Right Time

Example one: File-and-line breakpoints

```
1  var overdraftFee : UInt = 10
2  class Account {
3    var valueInCents : Int = 0
4    func withdraw(var amountInCents:UInt) {
5      if (Int(amountInCents) > self.valueInCents) {
6        valueInCents -= Int(overdraftFee)
7      }
8      valueInCents -= Int(amountInCents)
9    }
10   func deposit(var amountInCents:UInt) {
11     valueInCents += Int(amountInCents)
12   }
13 }
```

```
(lldb) b Account.swift:6
breakpoint set --file Account.swift --line 6
```

Stopping an App at the Right Time

Example one: File-and-line breakpoints

(lldb) **br l** breakpoint list

```
Current breakpoints:
1: file = 'file-and-line.swift', line = 6, locations = 1
  1.1: where = file-and-line`main.Account.withdraw (main.Account)(Swift.UInt)
-> () + 79 at file-and-line.swift:6, address = file-and-line
[0x0000000100000123f], unresolved, hit count = 0
```

Stopping an App at the Right Time

Example one: File-and-line breakpoints

Specification

(lldb) **br l** breakpoint list

```
Current breakpoints:
1: file = 'file-and-line.swift', line = 6, locations = 1
  1.1: where = file-and-line`main.Account.withdraw (main.Account)(Swift.UInt)
  -> () + 79 at file-and-line.swift:6, address = file-and-line
  [0x0000000100000123f], unresolved, hit count = 0
```

Stopping an App at the Right Time

Example one: File-and-line breakpoints

Specification

(lldb) **br l** breakpoint list

```
Current breakpoints:
1: file = 'file-and-line.swift', line = 6, locations = 1
  1.1: where = file-and-line`main.Account.withdraw (main.Account)(Swift.UInt)
    -> () + 79 at file-and-line.swift:6, address = file-and-line
    [0x000000010000123f], unresolved, hit count = 0
```

Location

Stopping an App at the Right Time

Example one: File-and-line breakpoints

Specification

(lldb) **br l** | breakpoint list |

```
Current breakpoints:
1: file = 'file-and-line.swift', line = 6, locations = 1
  1.1: where = file-and-line`main.Account.withdraw (main.Account)(Swift.UInt)
    -> () + 79 at file-and-line.swift:6, address = file-and-line
    [0x000000010000123f], unresolved, hit count = 0
```

Location

If you want to stop hitting the breakpoint for a while, you can disable the breakpoint

(lldb) **br dis 1** | breakpoint disable 1 |

Stopping an App at the Right Time

Example two: Symbolic breakpoints

```
1 func timestwo(var a : Int) -> Int {
2   return a * 2
3 }
4 func timestwo(var a : Double) -> Double {
5   return a * 2.0
6 }
7 func timestwo(var a : String) -> String {
8   return a + a
9 }
```

Stopping an App at the Right Time

Example two: Symbolic breakpoints

```
1 func timestwo(var a : Int) -> Int {
2   return a * 2
3 }
4 func timestwo(var a : Double) -> Double {
5   return a * 2.0
6 }
7 func timestwo(var a : String) -> String {
8   return a + a
9 }
```

(lldb) **b timestwo**

```
Breakpoint 1: 3 locations.
```

Stopping an App at the Right Time

Example two: Symbolic breakpoints

(lldb) **br l**

```
Current breakpoints:
1: name = 'timestwo', locations = 3
  1.1: where = symbolic`symbolic.timestwo (Swift.Int) -> Swift.Int + 8 at
symbolic.swift:2, address = symbolic[0x0000000100001548], unresolved, hit
count = 0
  1.2: where = symbolic`symbolic.timestwo (Swift.Double) -> Swift.Double + 24
at symbolic.swift:5, address = symbolic[0x0000000100001578], unresolved, hit
count = 0
  1.3: where = symbolic`symbolic.timestwo (Swift.String) -> Swift.String + 20
at symbolic.swift:8, address = symbolic[0x0000000100001594], unresolved, hit
count = 0
```

Stopping an App at the Right Time

Example two: Symbolic breakpoints

(lldb) **br l**

```
Current breakpoints:
1: name = 'timestwo', locations = 3
  1.1: where = symbolic`symbolic.timestwo (Swift.Int) -> Swift.Int + 8 at
symbolic.swift:2, address = symbolic[0x0000000100001548], unresolved, hit
count = 0
  1.2: where = symbolic`symbolic.timestwo (Swift.Double) -> Swift.Double + 24
at symbolic.swift:5, address = symbolic[0x0000000100001578], unresolved, hit
count = 0
  1.3: where = symbolic`symbolic.timestwo (Swift.String) -> Swift.String + 20
at symbolic.swift:8, address = symbolic[0x0000000100001594], unresolved, hit
count = 0
```

Stopping an App at the Right Time

Example two: Symbolic breakpoints

(lldb) **br l**

```
Current breakpoints:
1: name = 'timestwo', locations = 3
  1.1: where = symbolic`symbolic.timestwo (Swift.Int) -> Swift.Int + 8 at
symbolic.swift:2, address = symbolic[0x0000000100001548], unresolved, hit
count = 0
  1.2: where = symbolic`symbolic.timestwo (Swift.Double) -> Swift.Double + 24
at symbolic.swift:5, address = symbolic[0x0000000100001578], unresolved, hit
count = 0
  1.3: where = symbolic`symbolic.timestwo (Swift.String) -> Swift.String + 20
at symbolic.swift:8, address = symbolic[0x0000000100001594], unresolved, hit
count = 0
```

Stopping an App at the Right Time

Example two: Symbolic breakpoints

(lldb) **br l**

```
Current breakpoints:
1: name = 'timestwo', locations = 3
  1.1: where = symbolic`symbolic.timestwo (Swift.Int) -> Swift.Int + 8 at
symbolic.swift:2, address = symbolic[0x0000000100001548], unresolved, hit
count = 0
  1.2: where = symbolic`symbolic.timestwo (Swift.Double) -> Swift.Double + 24
at symbolic.swift:5, address = symbolic[0x0000000100001578], unresolved, hit
count = 0
  1.3: where = symbolic`symbolic.timestwo (Swift.String) -> Swift.String + 20
at symbolic.swift:8, address = symbolic[0x0000000100001594], unresolved, hit
count = 0
```

If you only care about strings, you can disable the other locations

(lldb) `breakpoint disable 1.1 1.2`

Stopping an App at the Right Time

Example two: Symbolic breakpoints

```
(lldb) br s -r timestwo.*String
breakpoint set --func-regex timestwo.*String
```

Stopping an App at the Right Time

Example two: Symbolic breakpoints

(lldb) **br s -r timestwo.*String**
breakpoint set --func-regex timestwo.*String

```
Breakpoint 2: where = symbolic`symbolic.timestwo (Swift.String) ->
Swift.String + 20 at symbolic.swift:8, address = 0x0000000100001594
```

Stopping an App at the Right Time

Example two: Symbolic breakpoints

```
(lldb) br s -r timestwo.*String
       breakpoint set --func-regex timestwo.*String

Breakpoint 2: where = symbolic`symbolic.timestwo (Swift.String) ->
Swift.String + 20 at symbolic.swift:8, address = 0x0000000100001594
```

You can use regular-expression breakpoints for many types of categories

- Methods of a class: `-r Account\\.`
- Functions in a module: `-r main\\.`

Stopping an App at the Right Time

Example two: Symbolic breakpoints

(lldb) **br s -r timestwo.*String**
breakpoint set --func-regex timestwo.*String

```
Breakpoint 2: where = symbolic`symbolic.timestwo (Swift.String) ->
Swift.String + 20 at symbolic.swift:8, address = 0x0000000100001594
```

You can use regular-expression breakpoints for many types of categories:

- Methods of a class: `-r Account\\.`
- Functions in a module: `-r main\\.`

Stopping an App at the Right Time

Example three: Smart breakpoints

```
1 class Account {
2   var valueInCents : Int = 0
3   func withdraw(var amountInCents:UInt) {
4     valueInCents -= Int(amountInCents)
5   }
6   func deposit(var amountInCents:UInt) {
7     valueInCents += Int(amountInCents)
8   }
9 }
```

Stopping an App at the Right Time

Example three: Smart breakpoints

```
1 class Account {
2   var valueInCents : Int = 0
3   func withdraw(var amountInCents:UInt) {
4     valueInCents -= Int(amountInCents)
5   }
6   func deposit(var amountInCents:UInt) {
7     valueInCents += Int(amountInCents)
8   }
9 }
```

Stopping an App at the Right Time

Example three: Smart breakpoints

```
1 class Account {
2   var valueInCents : Int = 0
3   func withdraw(var amountInCents:UInt) {
4     valueInCents -= Int(amountInCents)
5   }
6   func deposit(var amountInCents:UInt) {
7     valueInCents += Int(amountInCents)
8   }
9 }
```

(lldb) **b Account.swift:4**

Stopping an App at the Right Time

Example three: Smart breakpoints

```
1 class Account {
2   var valueInCents : Int = 0
3   func withdraw(var amountInCents:UInt) {
4     valueInCents -= Int(amountInCents)
5   }
6   func deposit(var amountInCents:UInt) {
7     valueInCents += Int(amountInCents)
8   }
9 }
```

(lldb) **b Account.swift:4**

Where would we charge an overdraft fee?

Stopping an App at the Right Time

Example three: Smart breakpoints

```
1 class Account {
2   var valueInCents : Int = 0
3   func withdraw(var amountInCents:UInt) {
4     valueInCents -= Int(amountInCents)
5   }
6   func deposit(var amountInCents:UInt) {
7     valueInCents += Int(amountInCents)
8   }
9 }
```

(lldb) **b Account.swift:4**

Where would we charge an overdraft fee?

(lldb) **br m -c "valueInCents < Int(amountInCents)"**

breakpoint modify --condition "…"

Stopping an App at the Right Time

Example three: Smart breakpoints

```
1 class Account {
2   var valueInCents : Int = 0
3   func withdraw(var amountInCents:UInt) {
4     valueInCents -= Int(amountInCents)
5   }
6   func deposit(var amountInCents:UInt) {
7     valueInCents += Int(amountInCents)
8   }
9 }
```

```
(lldb) b Account.swift:4
(lldb) br co a          breakpoint command add
Enter your debugger command(s).  Type 'DONE' to end.
```

Stopping an App at the Right Time

Example three: Smart breakpoints

```
1 class Account {
2   var valueInCents : Int = 0
3   func withdraw(var amountInCents:UInt) {
4     valueInCents -= Int(amountInCents)
5   }
6   func deposit(var amountInCents:UInt) {
7     valueInCents += Int(amountInCents)
8   }
9 }
```

```
(lldb) b Account.swift:4
(lldb) br co a          breakpoint command add
Enter your debugger command(s).  Type 'DONE' to end.
> p valueInCents
```

Stopping an App at the Right Time

Example three: Smart breakpoints

```
1 class Account {
2   var valueInCents : Int = 0
3   func withdraw(var amountInCents:UInt) {
4     valueInCents -= Int(amountInCents)
5   }
6   func deposit(var amountInCents:UInt) {
7     valueInCents += Int(amountInCents)
8   }
9 }
```

```
(lldb) b Account.swift:4
(lldb) br co a
```

breakpoint command add

```
Enter your debugger command(s).  Type 'DONE' to end.
> p valueInCents
> continue
```

Stopping an App at the Right Time

Example three: Smart breakpoints

```
1 class Account {
2   var valueInCents : Int = 0
3   func withdraw(var amountInCents:UInt) {
4     valueInCents -= Int(amountInCents)
5   }
6   func deposit(var amountInCents:UInt) {
7     valueInCents += Int(amountInCents)
8   }
9 }
```

```
(lldb) b Account.swift:4
(lldb) br co a          breakpoint command add
Enter your debugger command(s).  Type 'DONE' to end.
> p valueInCents
> continue
> DONE
```

Stopping an App at the Right Time

Example three: Smart breakpoints

```
(lldb) run
```

Stopping an App at the Right Time

Example three: Smart breakpoints

```
(lldb) run
(Int) $R1 = 30
Process 9257 resuming
Command #2 'continue' continued the target.
(Int) $R2 = 15
Process 9257 resuming
Command #2 'continue' continued the target.
(Int) $R3 = 10
Process 9257 resuming
Command #2 'continue' continued the target.
(Int) $R4 = -5
Process 9257 resuming
Command #2 'continue' continued the target.
```

Stopping an App at the Right Time

Smart breakpoints in Swift and Objective-C

You must set separate conditions and actions for Objective-C and Swift locations

```
(lldb) br m —c “a[3] == 2” 3.1
```

Stopping an App at the Right Time

Smart breakpoints in Swift and Objective-C

You must set separate conditions and actions for Objective-C and Swift locations

(lldb) **br m —c "a[3] == 2" 3.1**

(lldb) **br m —c "[a[3] isEqual:@2]" 3.2**

Stopping an App at the Right Time

Smart breakpoints in Swift and Objective-C

You must set separate conditions and actions for Objective-C and Swift locations

(lldb) **br m —c "a[3] == 2" 3.1**

(lldb) **br m —c "[a[3] isEqual:@2]" 3.2**

How to tell breakpoint locations apart

Objective-C Function	`1.1: where = test`main + 126 (...)`
Objective-C Method	`2.1: where = test`-[MyString initWithNSString:] + 24 (...)`
Swift Function	`3.1: where = test`test.ConformsDirectly.protocol_func (a.ConformsDirectly)(Swift.Int) -> Swift.Int + 35 (...)`

Stopping an App at the Right Time

Summing up

Breakpoints are a powerful way to stop your program

You can filter based on:

Source location	`--file --line`
Function name	`--name`
Module or class	`--func-regex`
Variable values	`breakpoint modify --condition`

You can even set automated actions so you don’t have to stop!

Overview of Today's Session

Basic debugging survival skills

- Reading a stopped app
- Stopping an app at the right time

REPL-enabled debugging workflows

- Validating existing code
- Trying out new code

Summing up

REPL Concepts

The REPL and the LLDB command line

REPL Concepts

The REPL and the LLDB command line

You can launch the REPL from a shell with an empty target

```
$ xcrun swift
```

REPL Concepts

The REPL and the LLDB command line

You can launch the REPL from a shell with an empty target

```
$ xcrun swift
```

You can break into the REPL whenever your program is stopped

```
(lldb) repl
  1>
```

REPL Concepts

The REPL and the LLDB command line

You can launch the REPL from a shell with an empty target

```
$ xcrun swift
```

You can break into the REPL whenever your program is stopped

```
(lldb) repl
  1>
```

You can break into LLDB from the REPL

```
  1> :
(lldb)
```

REPL Concepts

The REPL and the LLDB command line

You can launch the REPL from a shell with an empty target

```
$ xcrun swift
```

You can break into the REPL whenever your program is stopped

```
(lldb) repl
  1>
```

You can break into LLDB from the REPL

```
  1> :
(lldb)
```

You can issue LLDB commands directly from the REPL

```
  1> :help
```

Validating Existing Code

Example one: Array partition

```
func partition(input: Int[]) -> (Int[], Int[]) {…}
```

Validating Existing Code

Example one: Array partition

```
func partition(input: Int[]) -> (Int[], Int[]) {…}
```

(lldb) **repl**

Validating Existing Code

Example one: Array partition

```
func partition(input: Int[]) -> (Int[], Int[]) {…}
```

(lldb) **repl**

```
 1> partition([3,4,5])
```

Validating Existing Code

Example one: Array partition

```
func partition(input: Int[]) -> (Int[], Int[]) {…}
```

(lldb) **repl**

```
  1> partition([3,4,5])
$R1: (Int[], Int[]) = {
  0 = size=1 {
    [0] = 3
  }
  1 = size=2 {
    [0] = 4
    [1] = 5
  }
}
```

Validating Existing Code

Example one: Array partition

```
2>  func ispartition(l: Int[], r: Int[]) -> Bool {
3.    for le in l {
4.      for re in r {
5.        if le > re { return false }
6.      }
7.    }
8.    return true
9.  }
```

Validating Existing Code

Example one: Array partition

```
2>  func ispartition(l: Int[], r: Int[]) -> Bool {
3.    for le in l {
4.      for re in r {
5.        if le > re { return false }
6.      }
7.    }
8.    return true
9.  }

10>  ispartition(partition([6,2,1,5,3,8]))
```

Validating Existing Code

Example one: Array partition

```
2>  func ispartition(l: Int[], r: Int[]) -> Bool {
3.    for le in l {
4.      for re in r {
5.        if le > re { return false }
6.      }
7.    }
8.    return true
9.  }

10>  ispartition(partition([6,2,1,5,3,8]))
$R2: Bool = true
```

Validating Existing Code

Example two: Sorting

```
func mysort(input: Int[]) -> Int[] {…}
```

Validating Existing Code

Example two: Sorting

```
func mysort(input: Int[]) -> Int[] {…}
(lldb) repl
```

Validating Existing Code

Example two: Sorting

```
func mysort(input: Int[]) -> Int[] {…}
(lldb) repl
 1> mysort([5,1,4,2,3])
$R1: (Int[]) = size=5 {
  [0] = 1
  [1] = 2
  [2] = 3
  [3] = 4
  [4] = 5
}
```

Validating Existing Code

Example two: Sorting

```
2>  func issorted(a: Int[]) -> Bool {
3.    var last : Int? = nil
4.    for ae in a {
5.      if (last && (last! > ae)) { return false }
6.      last = ae
7.    }
8.    return true
9.  }
```

Validating Existing Code

Example two: Sorting

```
2>  func issorted(a: Int[]) -> Bool {
3.    var last : Int? = nil
4.    for ae in a {
5.      if (last && (last! > ae)) { return false }
6.      last = ae
7.    }
8.    return true
9.  }

10>  issorted(mysort([6,2,1,5,3,8]))
```

Validating Existing Code

Example two: Sorting

```
2>  func issorted(a: Int[]) -> Bool {
3.    var last : Int? = nil
4.    for ae in a {
5.      if (last && (last! > ae)) { return false }
6.      last = ae
7.    }
8.    return true
9.  }

10>  issorted(mysort([6,2,1,5,3,8]))
$R2: Bool = true
```

Overview of Today's Session

Basic debugging survival skills

- Reading a stopped app
- Stopping an app at the right time

REPL-enabled debugging workflows

- Validating existing code
- Trying out new code

Summing up

REPL Concepts

The REPL adds global code

Global variables

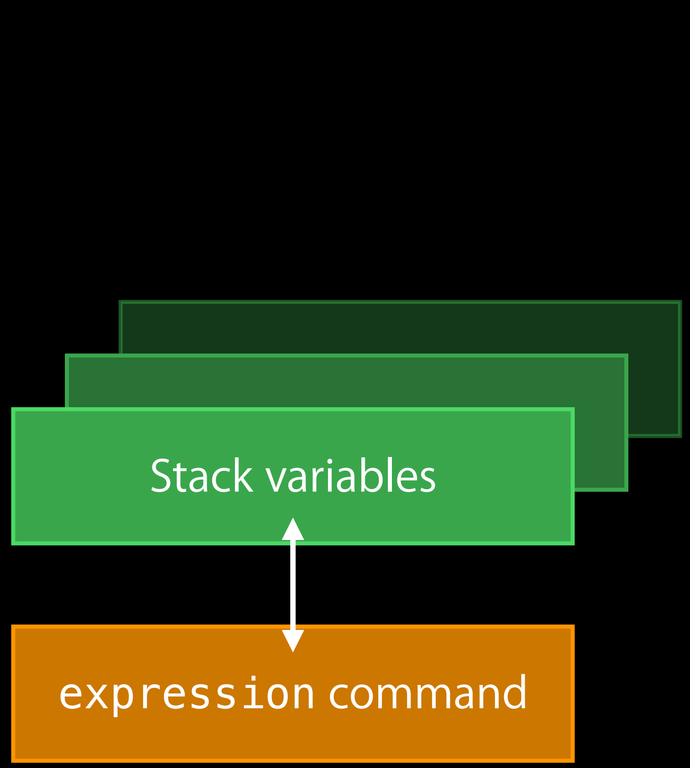
Classes

Functions

Stack variables

REPL Concepts

The REPL adds global code



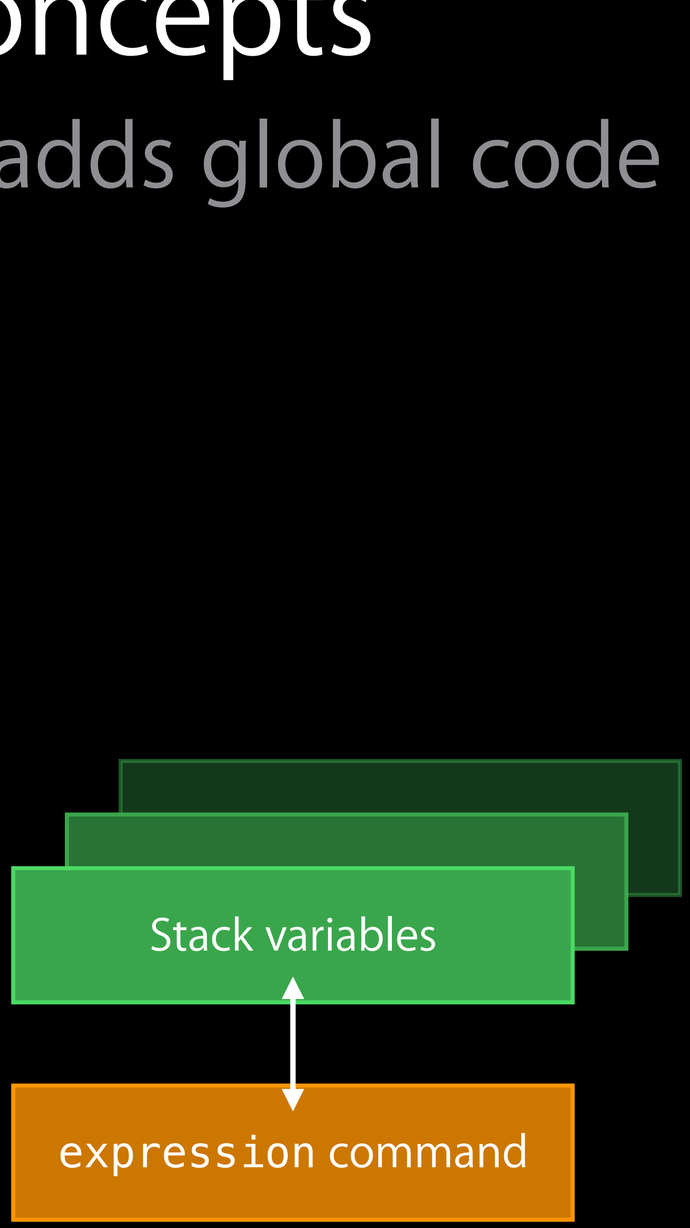
Global variables

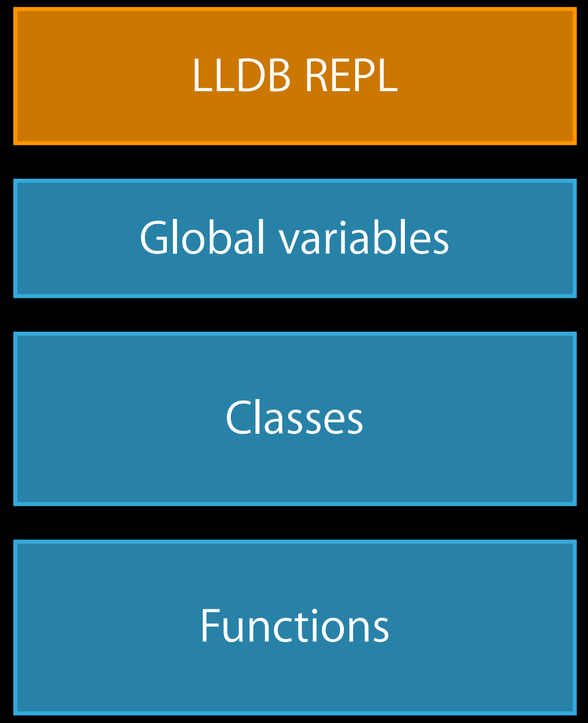
Classes

Functions

REPL Concepts

The REPL adds global code





Trying Out New Code

Example one: Dispatching on a queue

```
import Foundation

var queue : dispatch_queue_t =
  dispatch_queue_create("my_queue", nil)
```

Trying Out New Code

Example one: Dispatching on a queue

```
import Foundation

var queue : dispatch_queue_t =
  dispatch_queue_create("my_queue", nil)
```

```
(lldb) repl
 1> dispatch_sync(queue) {println(“world")}
```

Trying Out New Code

Example one: Dispatching on a queue

```
import Foundation

var queue : dispatch_queue_t =
  dispatch_queue_create("my_queue", nil)
```

```
(lldb) repl
 1> dispatch_sync(queue) {println(“world")}
world
 2>
```

Trying Out New Code

Example two: Implementing a protocol

```
protocol Doublable { func twice () -> Self }
extension Int : Doublable {
    func twice () -> Int { return 2 * self }
}
func fourtimes<T:Doublable>(a : T) -> T {
    return a.twice().twice()
}
```

Trying Out New Code

Example two: Implementing a protocol

```
protocol Doublable { func twice () -> Self }
extension Int : Doublable {
    func twice () -> Int { return 2 * self }
}
func fourtimes<T:Doublable>(a : T) -> T {
    return a.twice().twice()
}
(lldb) repl
```

Trying Out New Code

Example two: Implementing a protocol

```
protocol Doublable { func twice () -> Self }
extension Int : Doublable {
    func twice () -> Int { return 2 * self }
}
func fourtimes<T:Doublable>(a : T) -> T {
    return a.twice().twice()
}
```

(lldb) **repl**

```
 1> extension String : Doublable {
 2.   func twice() -> String { return self + self }
 3. }
```

Trying Out New Code

Example two: Implementing a protocol

```
protocol Doublable { func twice () -> Self }
extension Int : Doublable {
    func twice () -> Int { return 2 * self }
}
func fourtimes<T:Doublable>(a : T) -> T {
    return a.twice().twice()
}
```

(lldb) **repl**

```
 1> extension String : Doublable {
 2.   func twice() -> String { return self + self }
 3. }
 4> println(fourtimes("three"))
```

Trying Out New Code

Example two: Implementing a protocol

```
protocol Doublable { func twice () -> Self }
extension Int : Doublable {
    func twice () -> Int { return 2 * self }
}
func fourtimes<T:Doublable>(a : T) -> T {
    return a.twice().twice()
}
```

(lldb) **repl**

```
 1> extension String : Doublable {
 2.   func twice() -> String { return self + self }
 3. }
 4> println(fourtimes("three"))
threethreethreethree
```

Overview of Today's Session

Basic debugging survival skills

- Reading a stopped app
- Stopping an app at the right time

REPL-enabled debugging workflows

- Validating existing code
- Trying out new code

Summing up

Summary

LLDB provides tools to diagnose bugs in your program

- Stop reasons and the stack tell you what happened and how
- The print command tells you why it happened
- Breakpoints stop your program when you want to stop

With the REPL you can debug your program in Swift

- Validate your existing code with ad-hoc unit tests
- Add new functionality to your program on the fly

More Information

Dave DeLong
Developer Tools Evangelist
delong@apple.com

Documentation
LLDB Quick Start Guide
http://developer.apple.com

Apple Developer Forums
http://devforums.apple.com

Related Sessions

• Debugging in Xcode 6	Marina	Wednesday 10:15AM
• Advanced Swift Debugging in LLDB	Mission	Friday 9:00AM
• Advanced Debugging with LLDB	Online	WWDC 2013

WWDC14