Thread Sanitizer and Static Analysis

Help with finding bugs in your code

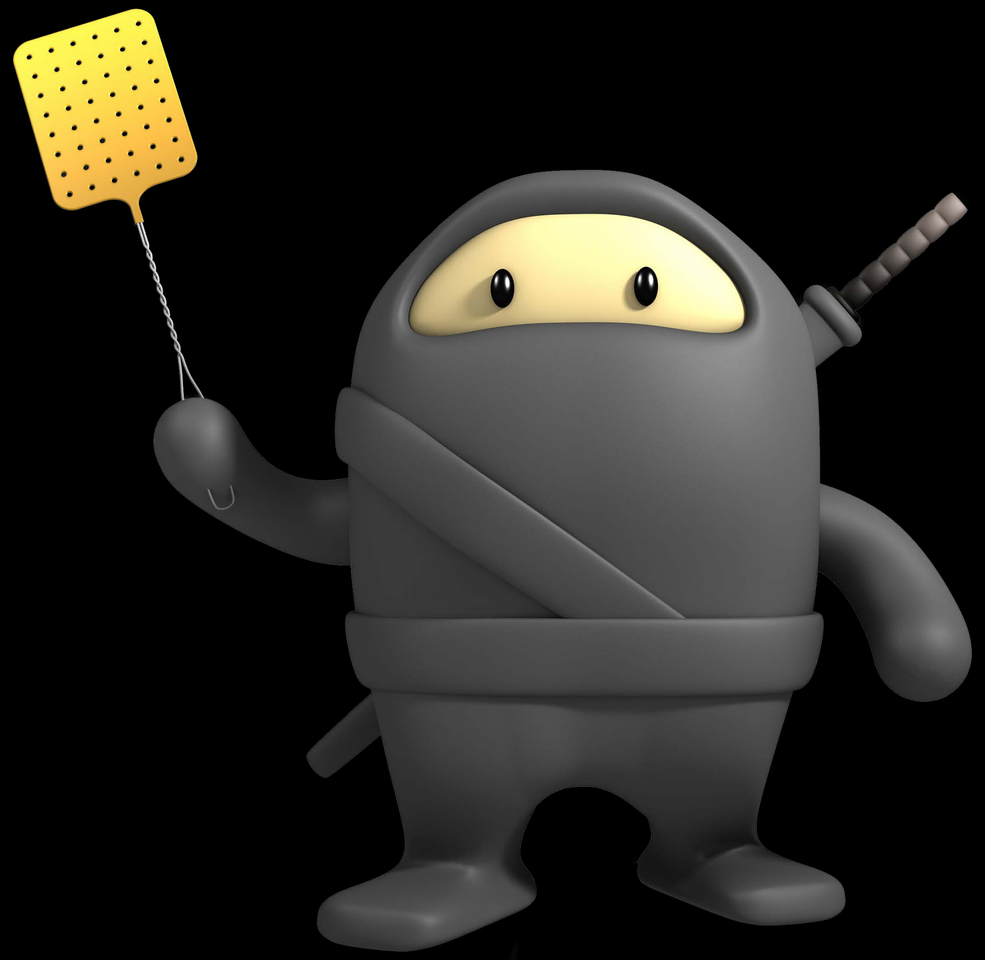
Session 412

Anna Zaks Manager, Program Analysis Team
Devin Coughlin Engineer, Program Analysis Team

What Is This Talk About?

Runtime Sanitizers

Static Analyzer



Runtime Sanitizers

Sanitizers

Find bugs at run time

Similar to Valgrind

Low overhead

Work with Swift 3 and C/C++/Objective-C

Integrated into Xcode IDE

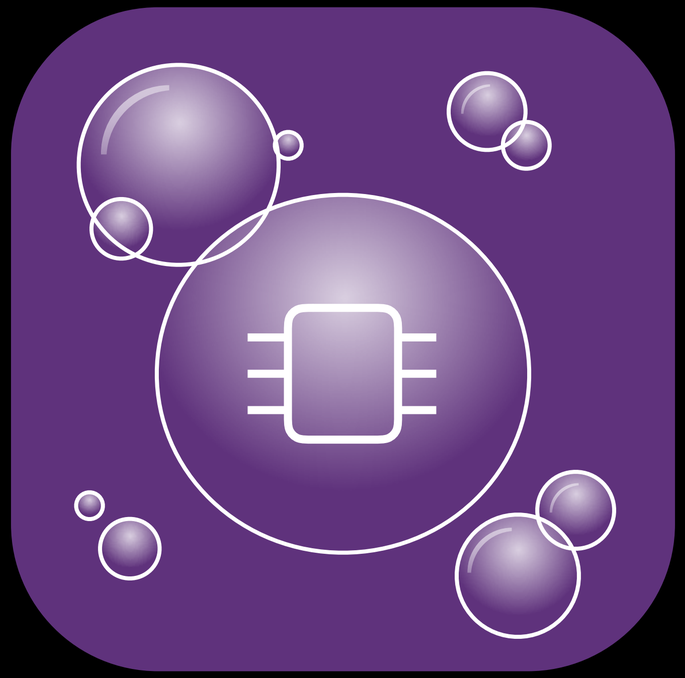
Address Sanitizer (ASan)

Introduced last year

Finds memory corruption issues

Effective at finding critical bugs

Now has full support for Swift



Threading Issues

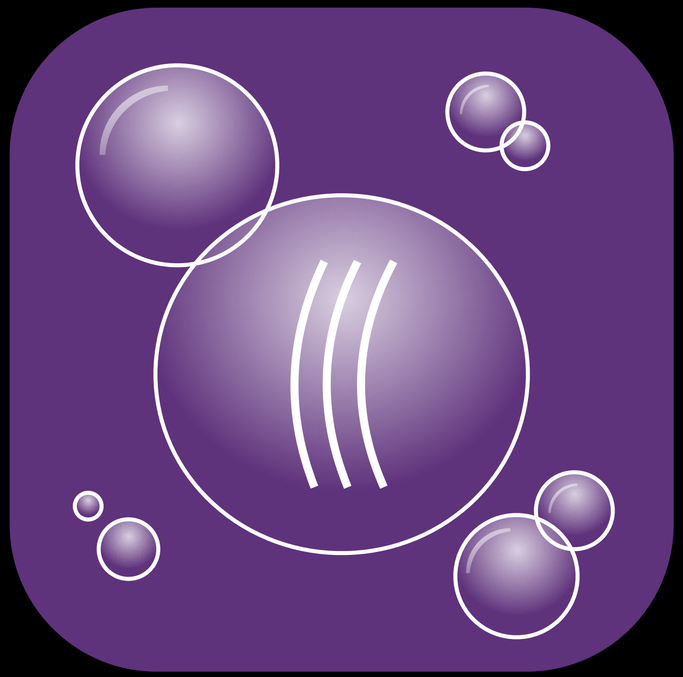
Hard to consistently reproduce

Difficult to debug

Lead to unpredictable results

Thread Sanitizer (TSan)

NEW



Thread Sanitizer (TSan)

NEW

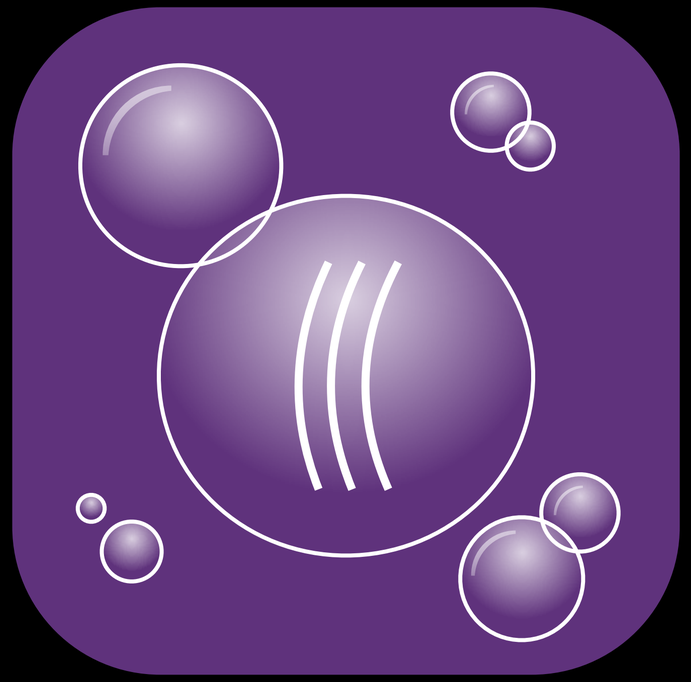
Use of uninitialized mutexes

Thread leaks (missing `pthread_join`)

Unsafe calls in signal handlers (ex: `malloc`)

Unlock from wrong thread

Data races



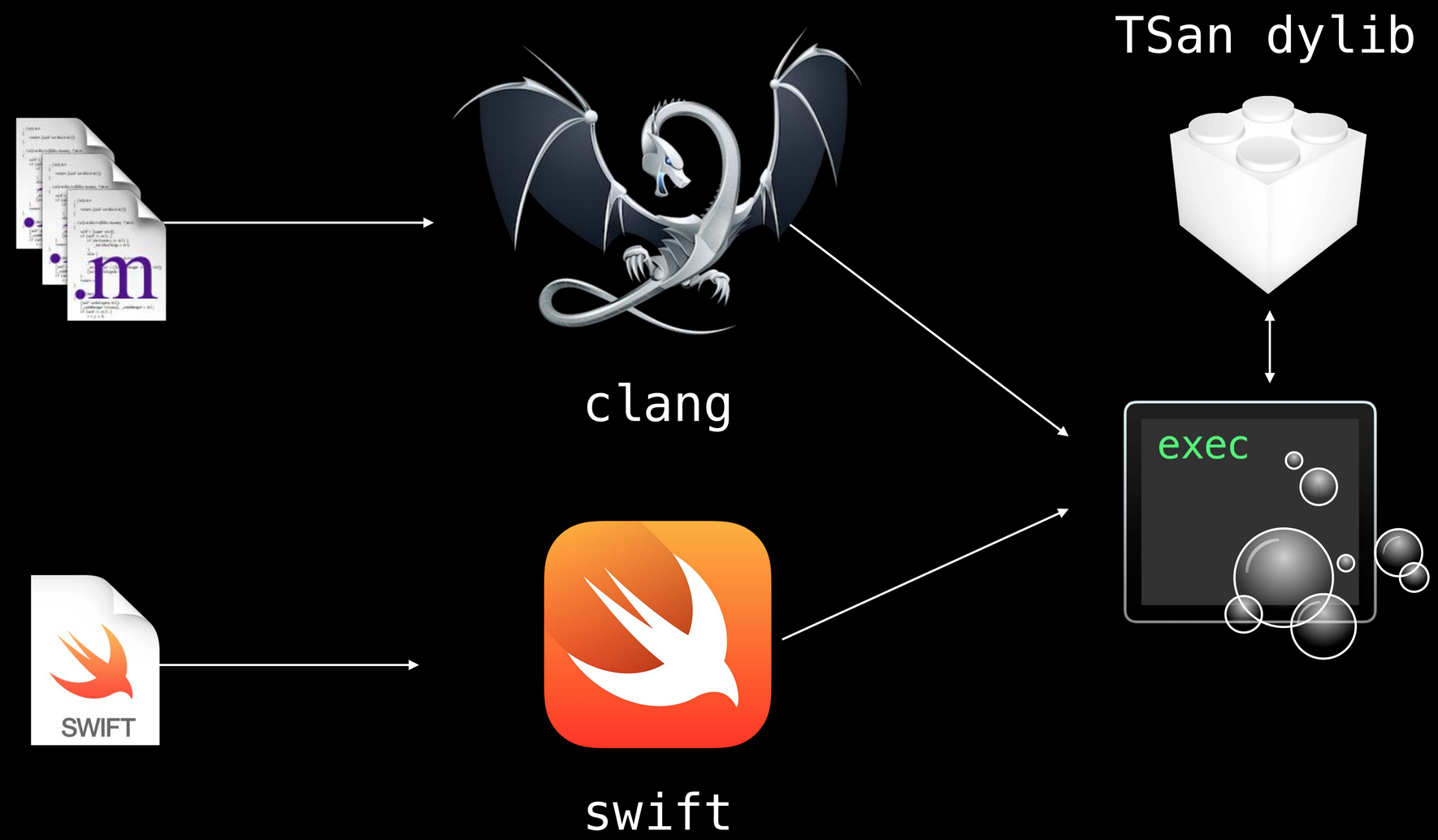
Demo

Thread Sanitizer in Xcode

Thread Sanitizer (TSan) in Xcode

1. Edit Scheme – Diagnostics tab
2. “Enable Thread Sanitizer” checkbox
3. Build and Run
4. View all of the generated runtime issues
5. Can choose to break on every issue

TSan Build Flow



Usage from Command-Line

Compile and link with TSan

```
$ clang -fsanitize=thread source.c -o executable
$ swiftc -sanitize=thread source.swift -o executable
$ xcodebuild -enableThreadSanitizer YES
```

Stop after the first error

```
$ TSAN_OPTIONS=halt_on_error=1 ./executable
```

Platform Support for TSan

64-bit macOS

64-bit iOS and tvOS simulators

No device support

No watchOS support

Fixing Data Races

Data Race

Multiple threads access the same memory without synchronization

At least one access is a write

May end up with any value or even memory corruption!

Reasons for Data Races

Can indicate a problem with the structure of your program

Often means that synchronization is missing

Data Race Example

```
var data: Int? = nil

func producer() {
    // More code here.
    data = 42
}
```

```
func consumer() {
    // More code here.
    print(data)
}
```

Data Race Example

```
var data: Int? = nil

func producer() {
    // More code here.
    data = 42
    dataIsAvailable = true
}
```

```
func consumer() {
    // More code here.
    while !dataIsAvailable {
        usleep(1000)
    }
    print(data)
}
```

Order is not guaranteed

Data Race Example

```
var data: Int? = nil

func producer() {
    // More code here.
    serialDispatchQueue.async {
        data = 42
    }
}
```

```
func consumer() {
    // More code here.
    serialDispatchQueue.sync {
        print(data)
    }
}
```

Data Race in Lazy Initialization Code

```
Singleton *getSingleton() {
    static Singleton *sharedInstance = nil;

    if (sharedInstance == nil) {
        sharedInstance = [[Singleton alloc] init];
    }

    return sharedInstance;
}
```

Both threads could be setting the value at the same time

Data Race in Lazy Initialization Code

```
Singleton *getSingleton() {
    static Singleton *sharedInstance = nil;

    if (sharedInstance == nil) {
        Singleton *localObject = [[Singleton alloc] init];

        // Only assign if sharedInstance is still nil.
        atomic_compare_and_set(&sharedInstance, nil, localObject);
    }

    return sharedInstance;
}
```

Use-after-free in ARC and object is leaked on a race in MRR

Unsynchronized read

Data Race in Lazy Initialization Code

```
Singleton *getSingleton() {
    static dispatch_once_t predicate;
    static Singleton *sharedInstance = nil;

    dispatch_once(&predicate, ^{
        sharedInstance = [[self alloc] init];
    });

    return sharedInstance;
}
```

Lazy Initialization in Swift

Global variables

```
var sharedInstance = Singleton()


func getSingleton() -> Singleton {
    return sharedInstance
}
```

Class constants

```
class Singleton  {
    static let sharedInstance = Singleton()
}
```

Choosing the Right Synchronization API

1. Use GCD

   Dispatch racy accesses to the same serial queue

2. Use pthread API, NSLock

   `pthread_mutex_lock()` to synchronize accesses

3. New `os_unfair_lock` (use instead of `OSSpinLock`)

4. Atomic operations

There is No Such Thing as a "Benign" Race

On some architectures (ex., x86) reads and writes are atomic

But even a "benign" race is undefined behavior in C

May cause issues with new compilers or architectures


FIX ALL THE
BUGS!

Behind the Scenes

Compiler Instruments Memory Accesses

```
*p = 'a';
```

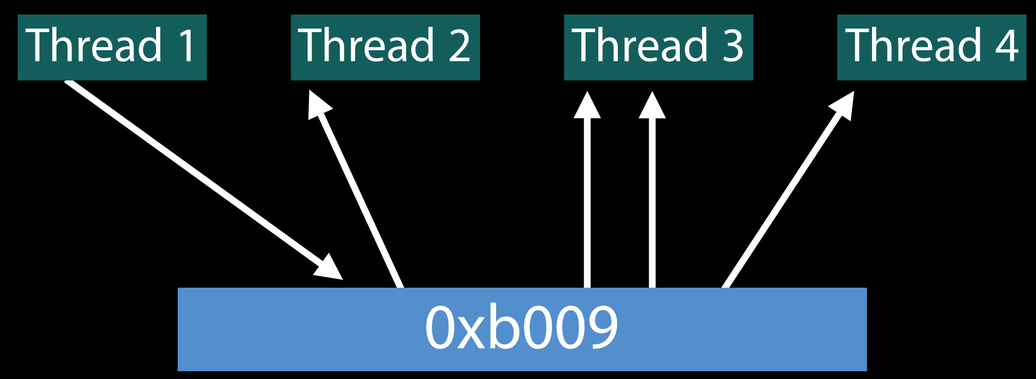
→

```
RecordAndCheckWrite(p);
*p = 'a';
```

For every access:

- Records the information about that access
- Checks if that access participates in a race

Read Shadow Stack Updates for Accesses



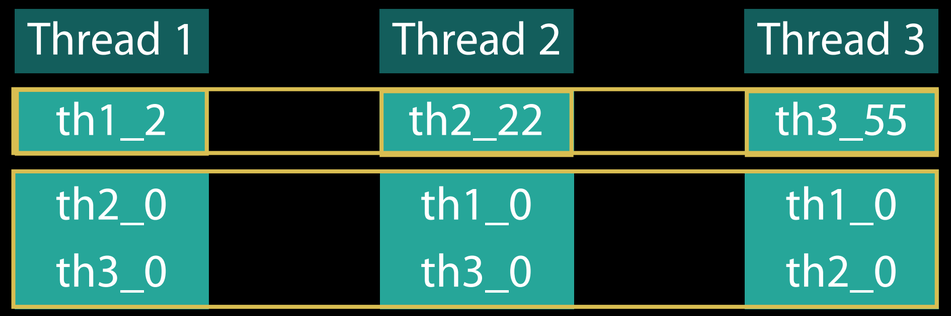
Application memory: 8 bytes

Shadow state:
Up to four 8 byte objects

Detecting a Race

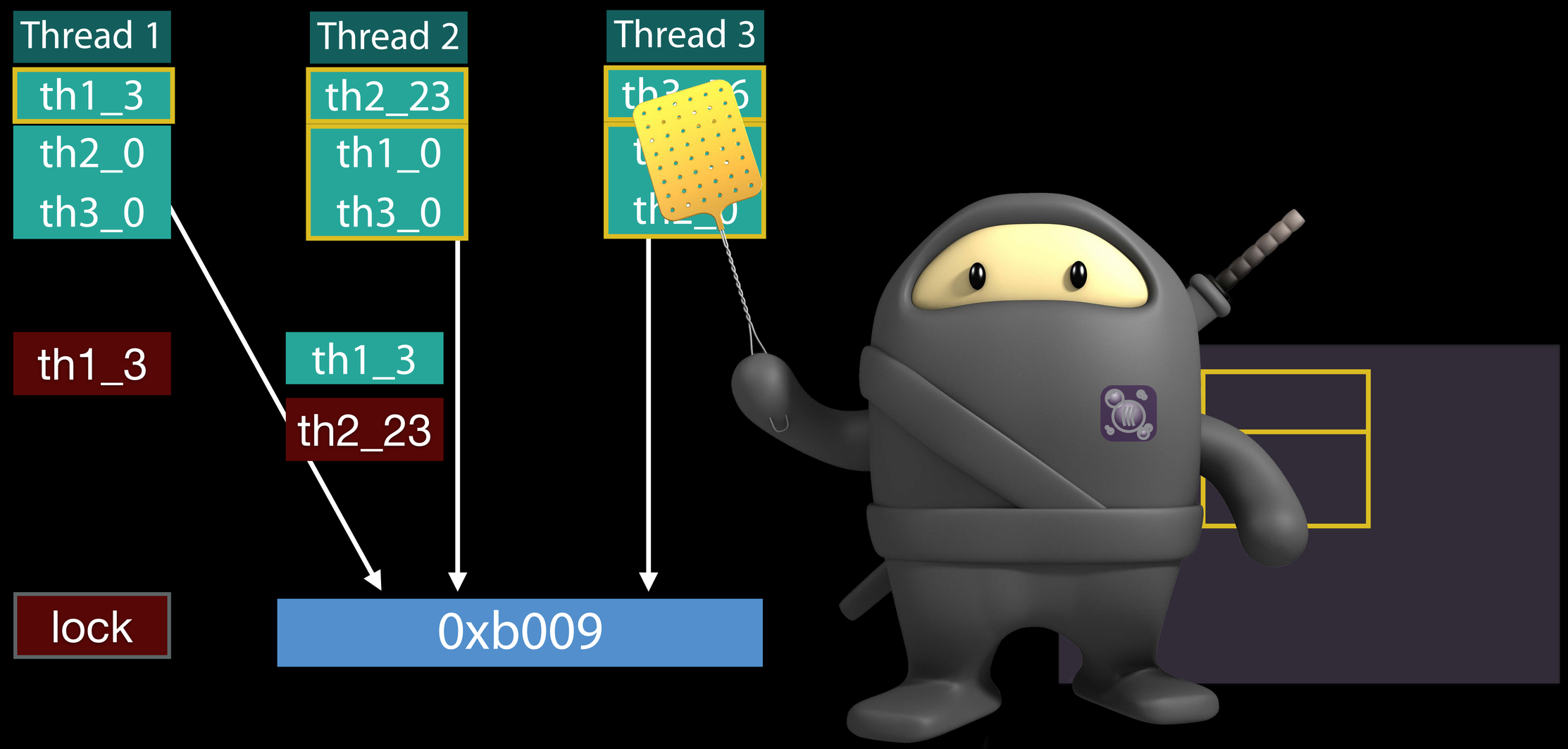
Every thread stores (in thread local):

- Thread's own timestamp
- The timestamps for other threads that establish the points of synchronization
- Timestamps are incremented on every memory access



Thread 1	Thread 2	Thread 3
th1_2	th2_22	th3_55
th2_0 th3_0	th1_0 th3_0	th1_0 th2_0

Detecting Data Race: Check for Races



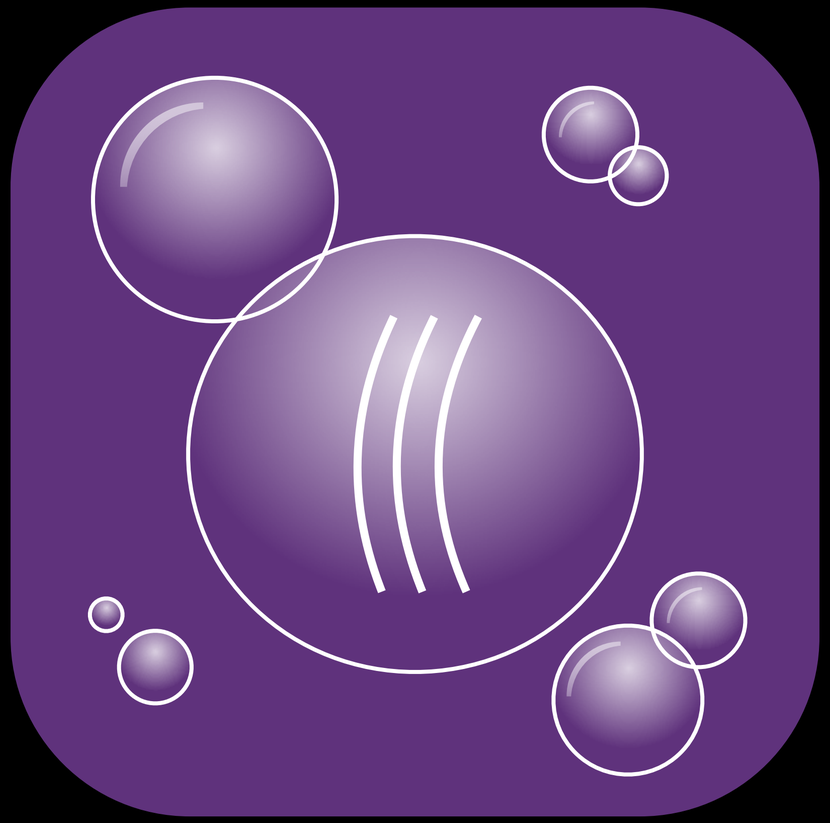
Thread Sanitizer

Timing does not matter

Can detect races even if they did not manifest during the particular run

The more run time coverage the better

Run all your tests with TSan!



Thread Sanitizer

Finding Bugs with Static Analysis

Find Bugs Without Running Code

Does not require running code (unlike sanitizers)

Great at catching hard to reproduce edge-case bugs

Supported only for C, C++, and Objective-C

Localizability

Instance Cleanup

Nullability

Missing Localizability

Startling for Users

Demo

Clang Static Analyzer in Xcode

Clang Static Analyzer in Xcode

1. Product > Analyze or
   Product > Analyze "SingleFile.m"
2. View in Issue Navigator
3. Explore Issue Path

Find Missing Localizability

Find unlocalized user-facing string:

```
[button setTitle:@“Cancel”];
```

User-facing text should use localized string macro

Find missing localization context comment:

```
NSString *t = NSLocalizedString(@“Cancel”, nil);
```

Localized string macro should include a non-empty comment for translators

Enable Checks in Build Settings

Static Analyzer - Generic Issues

Setting	MyProject
Dead Stores	Yes
Improper Memory Management	Yes - $(CLANG_ANALYZER_MALLOC)
Missing Localizability	**Yes**
Missing Localization Context Comment	**Yes**
Misuse of 'nonnull'	Yes
Misuse of Grand Central Dispatch	Yes

Checking -dealloc in Manual Retain/Release

Do Not Release ‘assign’ Properties

Release of synthesized ivar in -dealloc is over-release:

```
@property(assign) id delegate;

-(void)dealloc {
    [_delegate release];
    The '_delegate' ivar was synthesized for an assign property but was released in 'dealloc'

    [super dealloc];
}
```

Do Release 'retain/copy' Properties

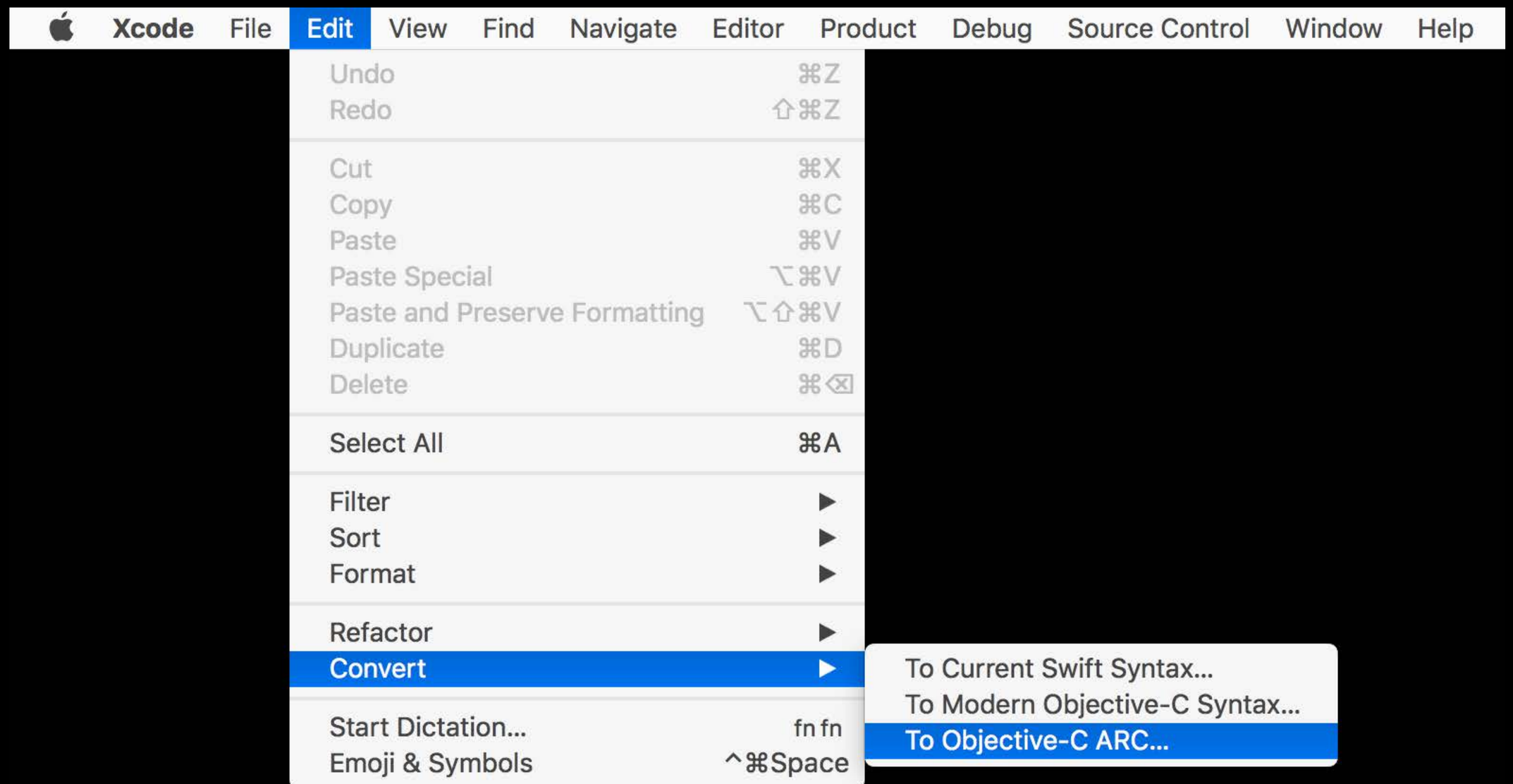
Leak if no release of ivar for retain/copy property in -dealloc:

```
@property(assign) id delegate;
@property(copy) NSString *title;
-(void)dealloc {

    The '_title' ivar was copied by a synthesized property but not released

    [super dealloc];
}
```

Update to Automated Reference Counting



Nullability Violations

Nullability Annotations

Indicate whether a value is expected to be nil:

```
@interface CLLocation : NSObject
@property(readonly, nonnull) NSDate *timestamp;
@property(readonly, nullable) CLFloor *floor;
```

Why Annotate Nullability?

New programming model communicates expectation to callers

Violation can cause crashes or other unexpected behavior

Swift enforces model in type system with optionals

```
class CLLocation : NSObject {
    public var timestamp: NSDate { get }
    public var floor: CLFloor? { get }
}
```

Finding Nullability Violations

Particularly useful in mixed Swift/Objective-C projects

Logical problem in code

Incorrect annotations

Violation: Branching with nil Default

```
- (nonnull NSString *)shortDescription {
    NSString *name = nil;

    if (self.cityName)
        name = self.cityName;
    if (self.countryName)
        name = self.countryName;

    return name;
}
```

Null is returned from a method that is expected to return a non-null value

Violation: Branching with nil Default

```
- (nonnull NSString *)shortDescription {
    NSString *name = NSLocalizedString(@“Earth”, @“The planet”);

    if (self.cityName)
        name = self.cityName;
    if (self.countryName)
        name = self.countryName;

    return name;
}
```

Violation: Incorrect Annotation

```
NS_ASSUME_NONNULL_BEGIN
@property(readonly) PressureData *pressure;
NS_ASSUME_NONNULL_END

- (PressureData *)pressure {
    if ([self hasBarometer])
        return [self measurePressure];

    return nil;
}
```

Null is returned from a method that is expected to return a non-null value

Violation: Incorrect Annotation

```
NS_ASSUME_NONNULL_BEGIN
@property(readonly, nullable) PressureData *pressure;
NS_ASSUME_NONNULL_END

- (PressureData *)pressure {
    if ([self hasBarometer])
        return [self measurePressure];

    return nil;
}
```

Nullability of Your API is a Contract

Do not change just to silence the analyzer

Do carefully consider nullability of API

Suppress with Cast

Return nil defensively for backwards compatibility:

```
- (NSString * _Nonnull)stringAtIndex:(int) index {
    if (index < 0 || index >= _count)
        return (NSString * _Nonnull)nil;
    …
}
```

Static Analyzer

Wrapping Up

These Tools Find Real Bugs!

Address Sanitizer and Thread Sanitizer

Clang Static Analyzer

Use on your code!

More Information

https://developer.apple.com/wwdc16/412

Related Sessions

Internationalization Best Practices	Mission	Tuesday 9:00AM
Visual Debugging with Xcode	Presidio	Wednesday 4:00PM
Debugging Tips and Tricks	Pacific Heights	Friday 1:40PM
Using Time Profiler in Instruments	Nob Hill	Friday 3:00PM
Concurrent Programming with GCD in Swift 3	Pacific Heights	Friday 4:00PM

Labs

Thread Sanitizer, Static Analysis, and LLVM Compiler Lab	Developer Tools Lab B	Thursday 12:00PM
Thread Sanitizer, Static Analysis, and LLVM Compiler Lab	Developer Tools Lab C	Friday 3:00PM
LLVM Compiler, Objective-C, and C++ Lab	Developer Tools Lab C	Friday 4:30PM
GCD Lab	Frameworks Lab D	Friday 5:00PM

WWDC16