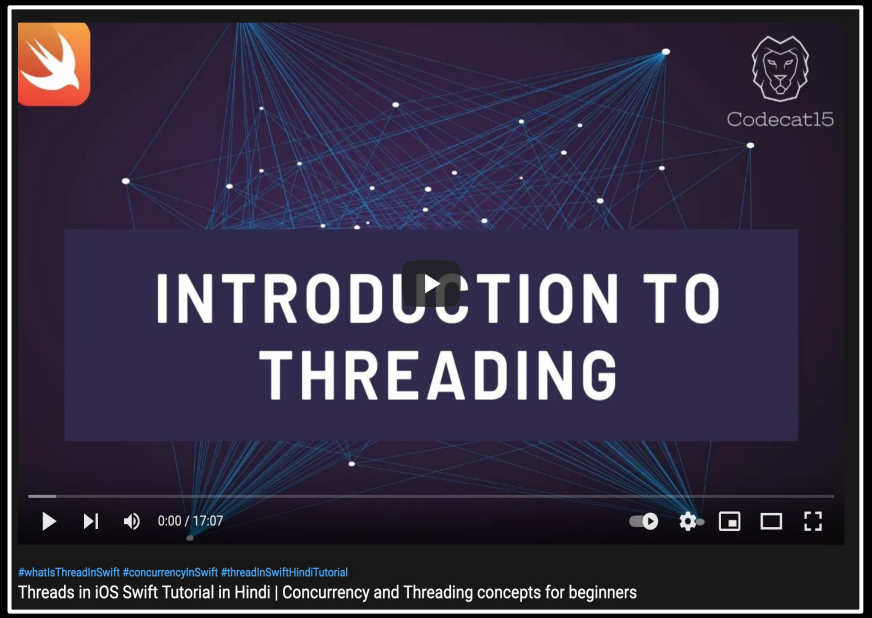
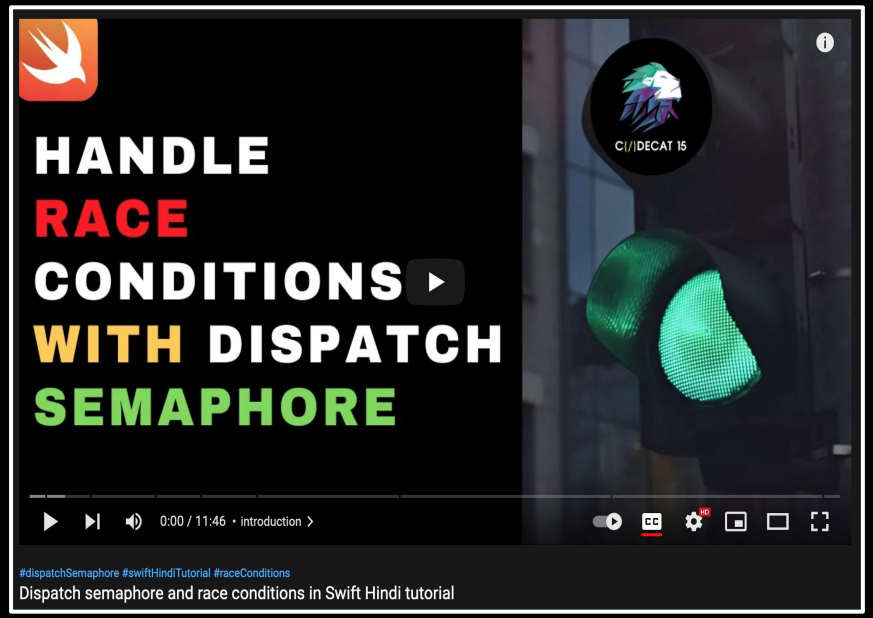
actor keyword

concurrency

locks

semaphore





Video link in description

C//DECAT 15

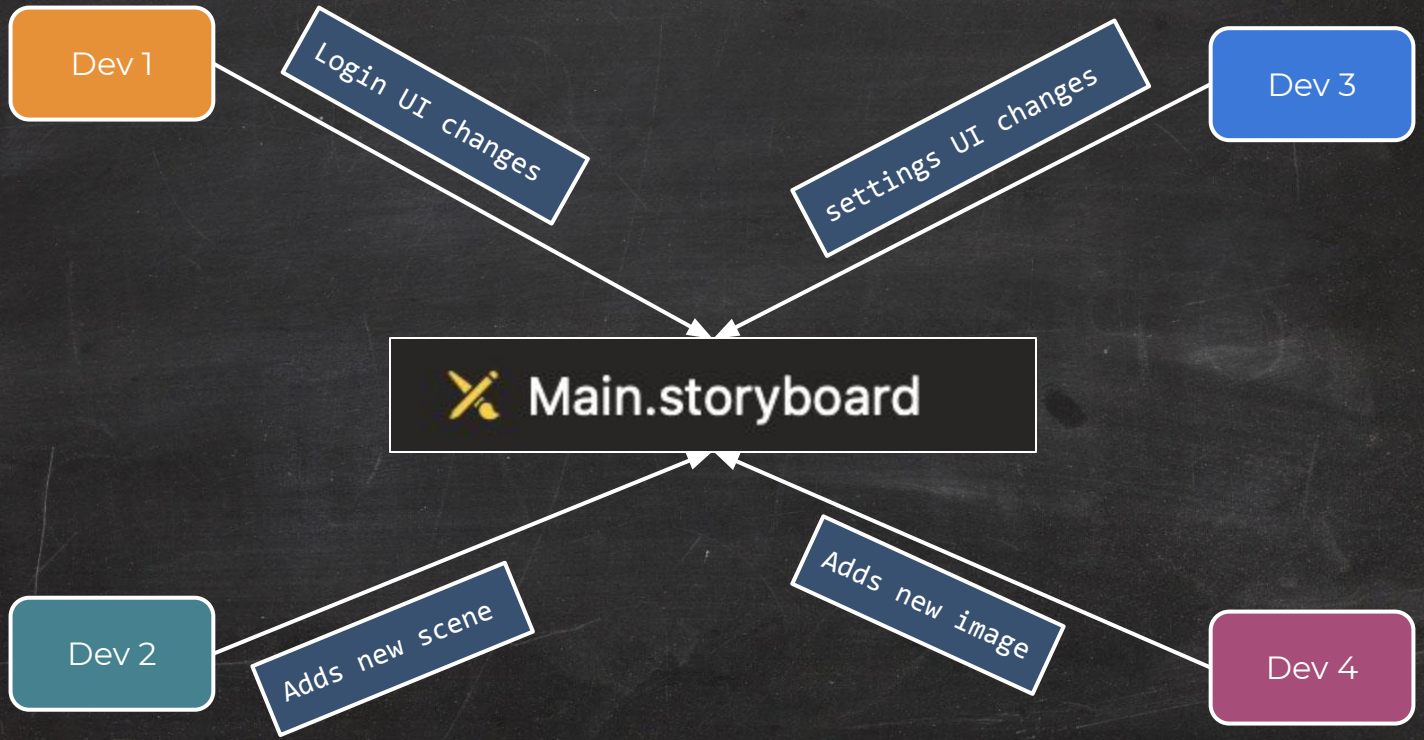
actor is used to handle data race in a concurrent environment

We know race condition but
what is data race?

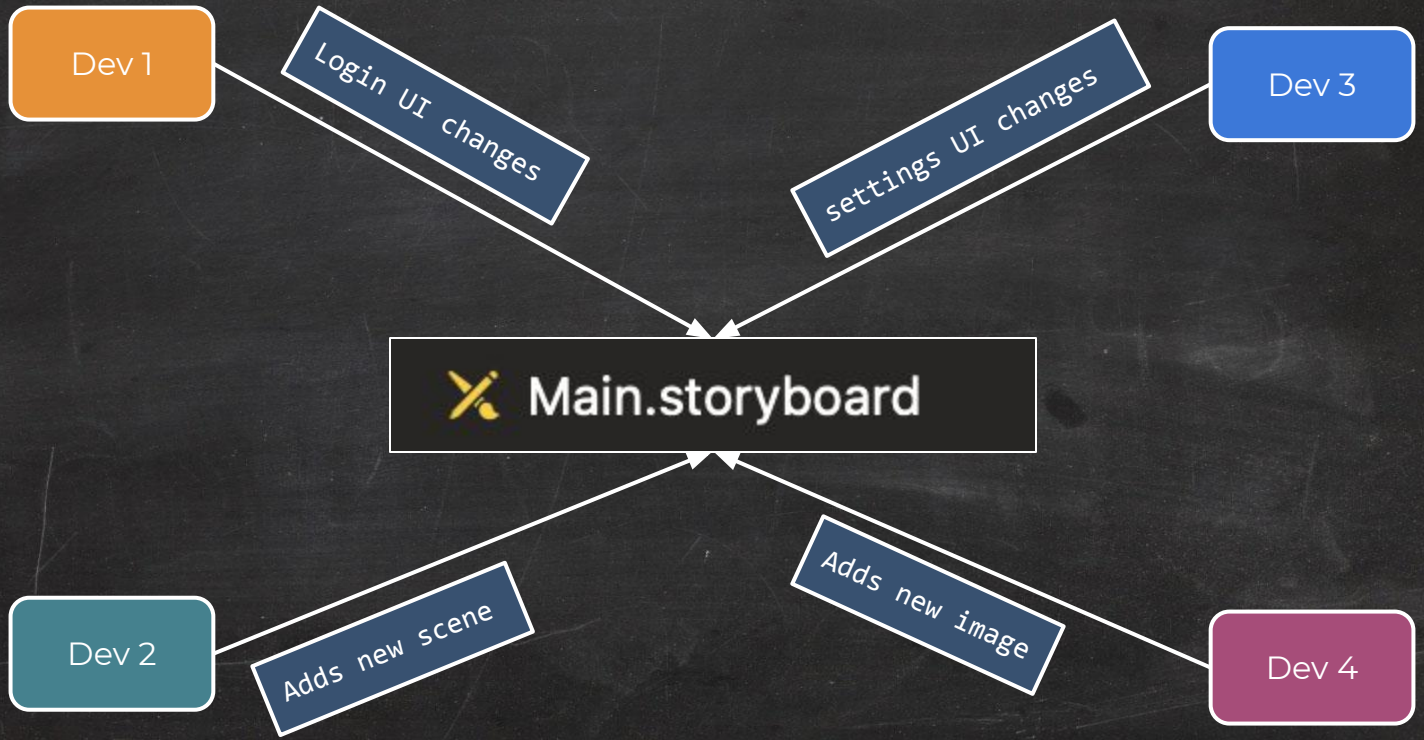
How is data race **different** from a race condition?

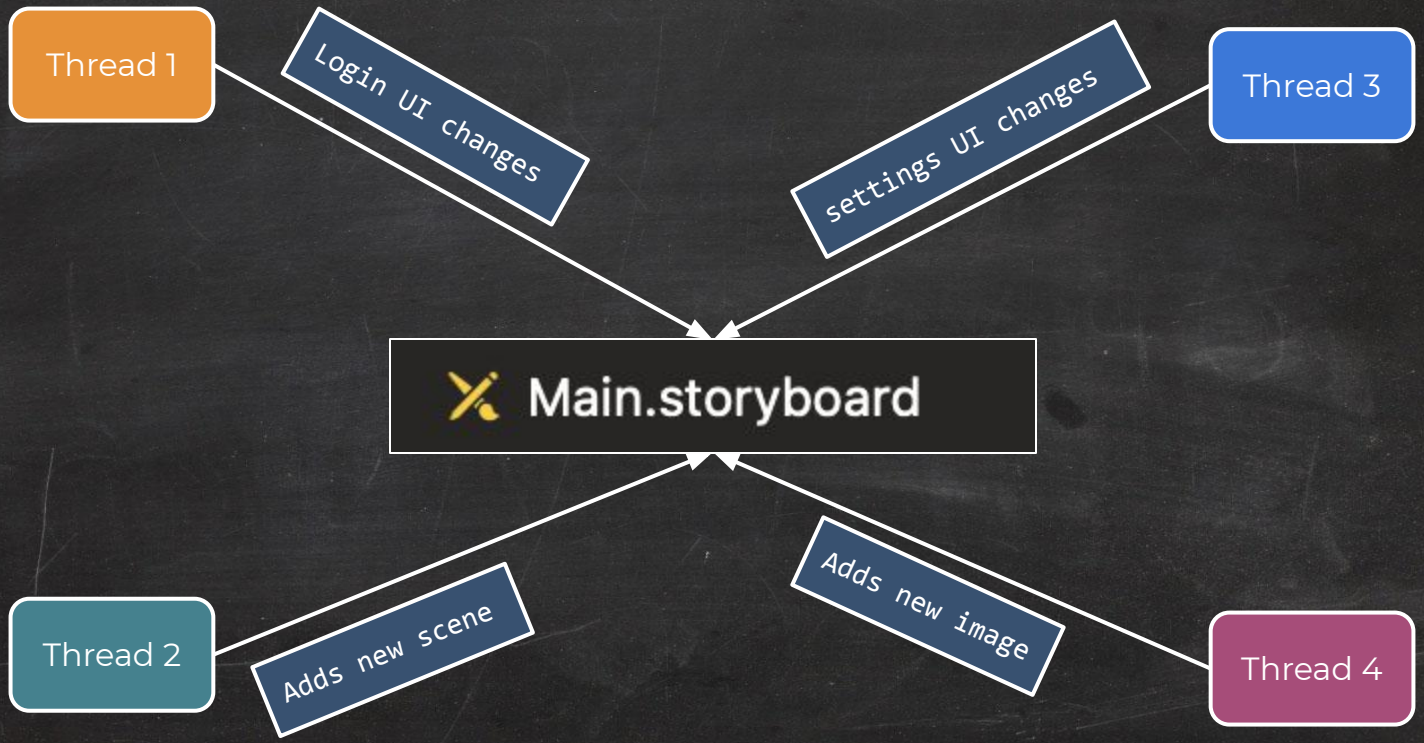
Data race and race conditions are
two different concepts

Side effects of data race

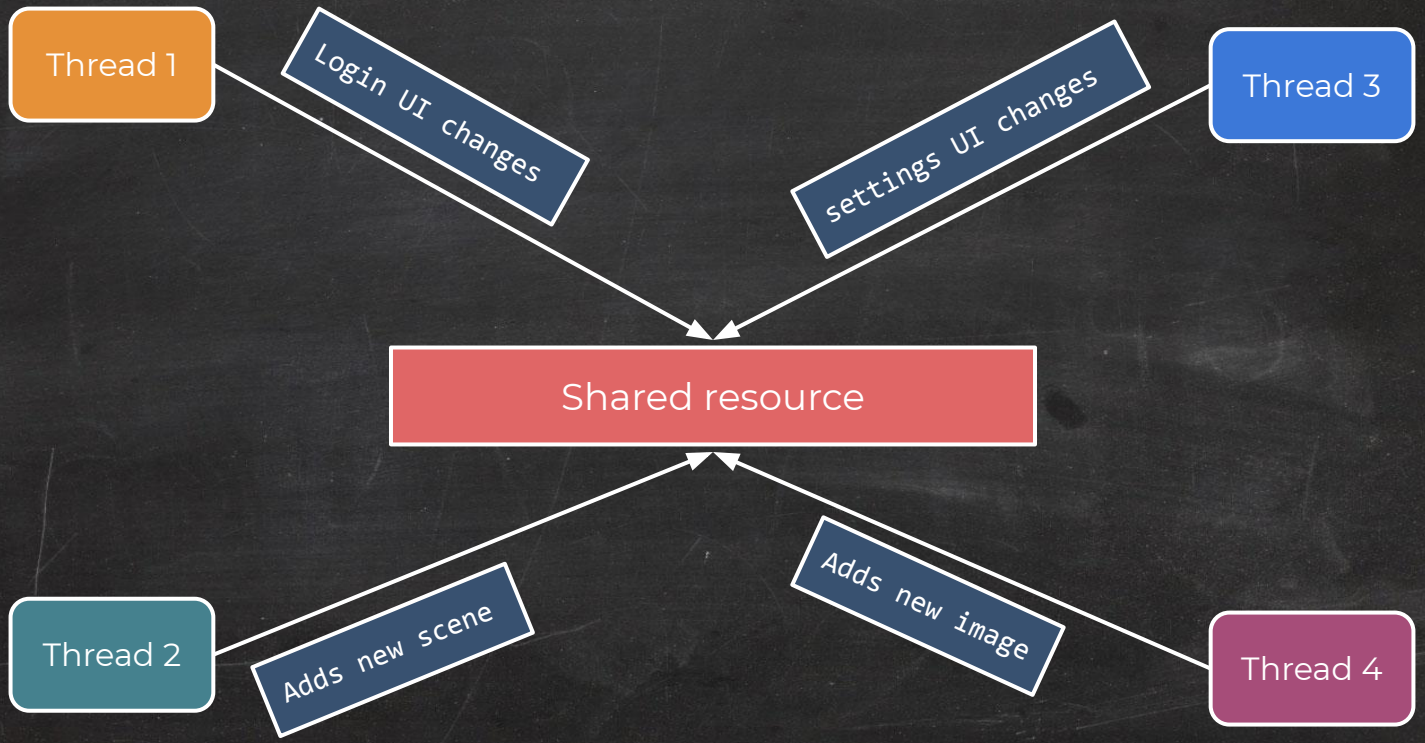

Dev 1
Login UI changes
settings UI changes
Dev 3
Main.storyboard
Dev 2
Adds new scene
Adds new image
Dev 4

MERGE CONFLICT


Dev 1
Login UI changes
settings UI changes
Dev 3
Main.storyboard
Dev 2
Adds new scene
Adds new image
Dev 4


Thread 1
Login UI changes
settings UI changes
Thread 3
Main.storyboard
Thread 2
Adds new scene
Adds new image
Thread 4

C(/)DECAT 15


Thread 1
Login UI changes
settings UI changes
Thread 3
Shared resource
Thread 2
Adds new scene
Adds new image
Thread 4

Unexpected output or behaviour

Data race

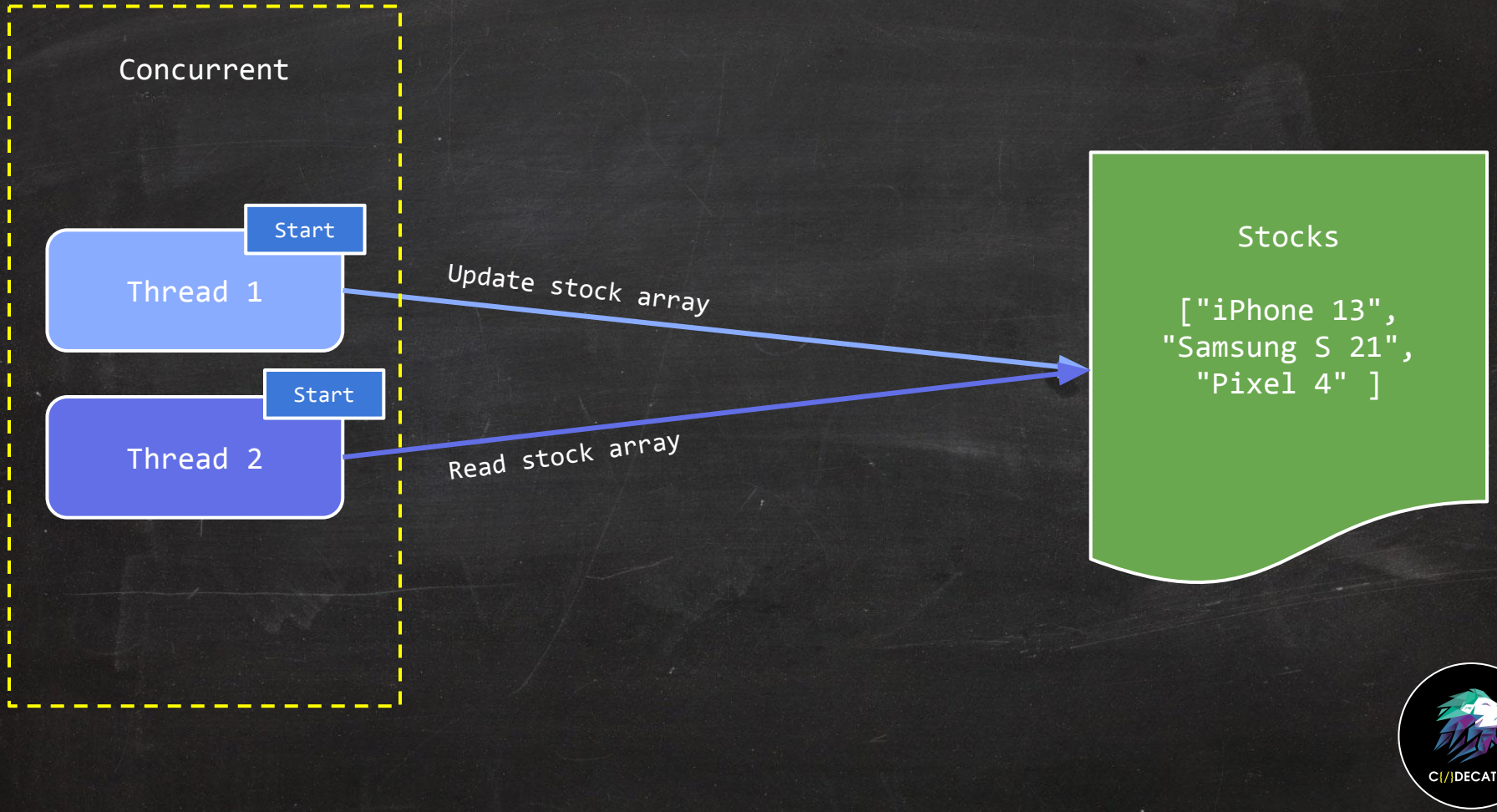
1. Multiple threads access shared resource or shared memory location in a concurrent environment
2. Without any locks or checks
3. Modify the shared resource or shared memory location

Impact to shared resource due to data race

1. Unstable state
2. Corrupt memory
3. Unexpected behavior
4. Crash

Data race are hard to debug

Xcode setting to identify data race during development


Concurrent
Start
Thread 1
Start
Thread 2
Update stock array
Read stock array
Stocks
["iPhone 13",
"Samsung S 21",
"Pixel 4"]

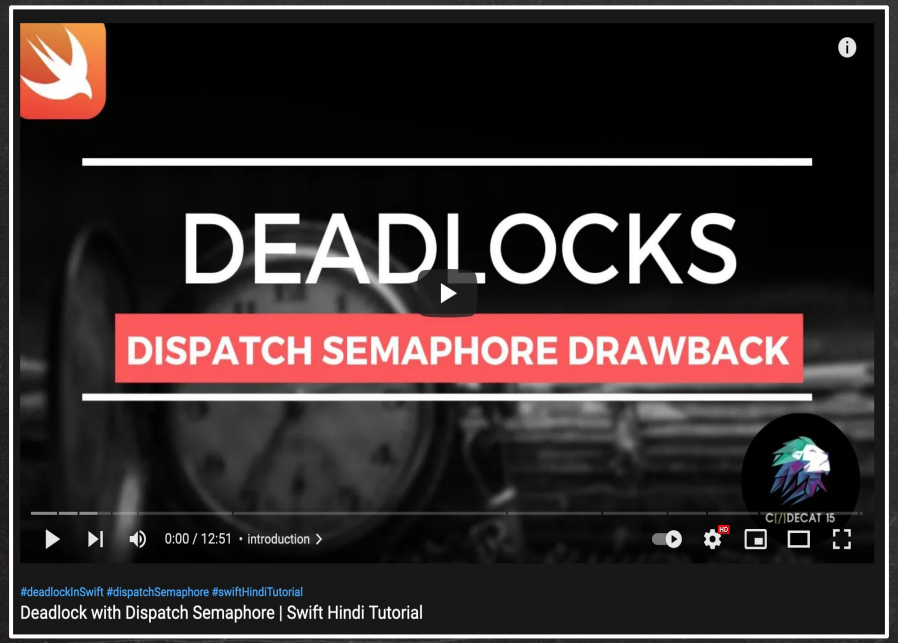
Dispatch Semaphore & Dispatch barriers

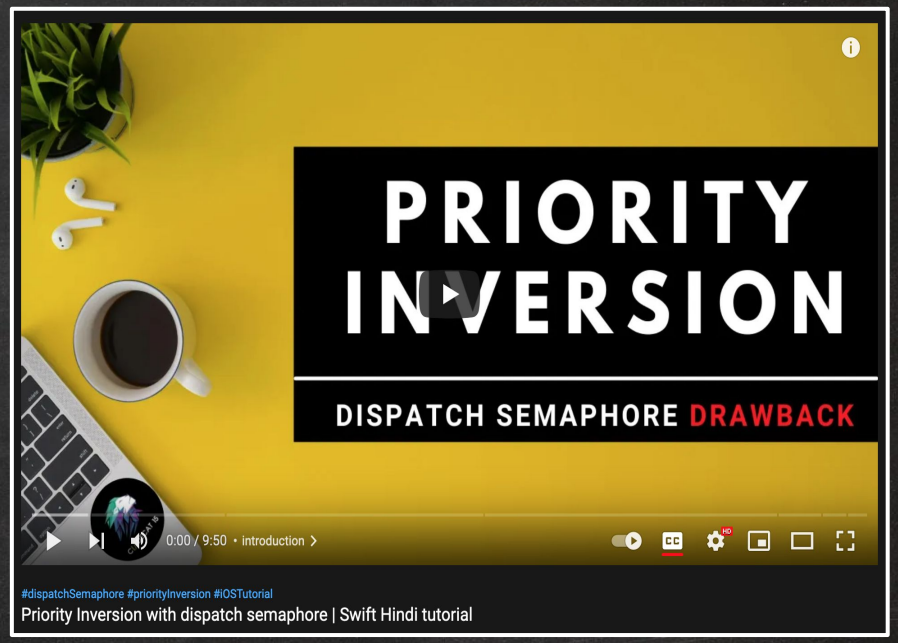
If dispatch barriers and semaphores can handle data race

Why did apple introduce the actor keyword?

Incomplete or half isolation of the shared resource

Deadlock and priority inversion





Video link in description

C[/]DECAT 15

actor addresses the half isolation issue

Actor model

From Wikipedia, the free encyclopedia

The **actor model** in computer science is a mathematical model of concurrent computation that treats *actor* as the universal primitive of concurrent computation. In response to a message it receives, an actor can: make local decisions, create more actors, send more messages, and determine how to respond to the next message received. Actors may modify their own private state, but can only affect each other indirectly through messaging (removing the need for lock-based synchronization).

The actor model originated in 1973.[1] It has been used both as a framework for a theoretical understanding of computation and as the theoretical basis for several practical implementations of concurrent systems. The relationship of the model to other work is discussed in actor model and process calculi.

Source: https://en.wikipedia.org/wiki/Actor_model

Using actor keyword we can protect shared resource from data race

With actor keyword it’s the responsibility of the framework to isolate changes to the shared resource or memory

Maybe internally actor is using some kind of locking mechanism?

Actor model

From Wikipedia, the free encyclopedia

The **actor model** in computer science is a mathematical model of concurrent computation that treats *actor* as the universal primitive of concurrent computation. In response to a message it receives, an actor can: make local decisions, create more actors, send more messages, and determine how to respond to the next message received. Actors may modify their own private state, but can only affect each other indirectly through messaging (removing the need for lock-based synchronization).

The actor model originated in 1973.[1] It has been used both as a framework for a theoretical understanding of computation and as the theoretical basis for several practical implementations of concurrent systems. The relationship of the model to other work is discussed in actor model and process calculi.

Source: https://en.wikipedia.org/wiki/Actor_model

Actor

1. Reference type just like class
2. Use actor keyword to declare an actor
3. Actor can have properties, functions and generics

```swift
class BankAccount {

    var accountBalance : Double = 30000 // mutable property

    let bankName : String = "CodeCat15 Bank" // non-mutable (constant) property

    func withDrawAmount(amount: Double) {
        guard accountBalance > amount else {return}
        accountBalance -= amount
    }

    func displayBankName() {
        print(bankName)
    }
}
```

```swift
actor BankAccount {

    var accountBalance : Double = 30000 // mutable property

    let bankName : String = "CodeCat15 Bank" // non-mutable (constant) property

    func withDrawAmount(amount: Double) {
        guard accountBalance > amount else {return}
        accountBalance -= amount
    }

    func displayBankName() {
        print(bankName)
    }
}
```

What's the difference between class and actor?

Inheritance

```
actor TransactionCharges {}

actor BankAccount : TransactionCharges {

        public var accountBalance : Double = 30000 // mutable property

        let bankName : String = "CodeCat15 Bank" // non-mutable (constant) property
```

Actor types do not support inheritance

An actor cannot inherit from other actor

Alternatives considered

Actor inheritance

Earlier pitches and the first reviewed version of this proposal allowed actor inheritance. Actor inheritance followed the rules of class inheritance, albeit with specific additional rules required to maintain actor isolation:

- An actor could not inherit from a class, and vice-versa.
- An overriding declaration must not be more isolated than the overridden declaration.

Subsequent review discussion determined that the conceptual cost of actor inheritance outweighed its usefulness, so it has been removed from this proposal. The form that actor inheritance would take in the language is well-understand from prior iterations of this proposal and its implementation, so this feature could be re-introduced at a later time.

Source: https://github.com/apple/swift-evolution/blob/main/proposals/0306-actors.md#actor-isolation

```swift
protocol TransactionCharges {}

actor BankAccount : TransactionCharges {

        public var accountBalance : Double = 30000 // mutable property

        let bankName : String = "CodeCat15 Bank" // non-mutable (constant) property
```

An actor conforms to protocol

Empty protocol should never be used in code, this is just for demonstration purpose

Documentation › ··· › Concurrency › Actor

Language: Swift API Changes: Show

Protocol

Actor

Common protocol to which all actors conform.

Declaration

```
protocol Actor : AnyObject, Sendable
```

Overview

The `Actor` protocol generalizes over all actor types. Actor types implicitly conform to this protocol.

Availability

iOS 13.0+

iPadOS 13.0+

macOS 10.15+

Mac Catalyst 15.0+

tvOS 13.0+

watchOS 6.0+

Xcode 13.0+

Framework

Swift Standard Library

Source: https://developer.apple.com/documentation/swift/actor

Class

MainActor

A singleton actor whose executor is equivalent to the main dispatch queue.

Declaration

```
@globalActor final actor MainActor
```

Availability

iOS 13.0+

iPadOS 13.0+

macOS 10.15+

Mac Catalyst 15.0+

tvOS 13.0+

watchOS 6.0+

Xcode 13.0+

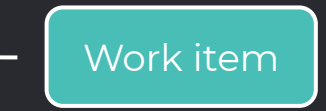
Source: https://developer.apple.com/documentation/swift/mainactor

CODECAT15 EXAMPLE

DispatchQueue *async* is like the delivery person

```swift
let queue = DispatchQueue(label: "queue", attributes: .concurrent)

queue.async {
    // code
}
```



Documentation > Dispatch > DispatchQueue > async(execute:)

Instance Method

async(execute:)

Schedules a work item for immediate execution, and returns immediately.

Declaration

```
func async(execute workItem: DispatchWorkItem)
```

Source: https://developer.apple.com/documentation/dispatch/dispatchqueue/2016103-async

Cash on delivery payment mode

Declaration

```
@frozen struct Task<Success, Failure> where Failure : Error
```

Overview

When you create an instance of `Task`, you provide a closure that contains the work for that task to perform. Tasks can start running immediately after creation; you don't explicitly start or schedule them. After creating a task, you use the instance to interact with it — for example, to wait for it to complete or to cancel it. It's not a programming error to discard a reference to a task without waiting for that task to finish or canceling it. A task runs regardless of whether you keep a reference to it. However, if you discard the reference to a task, you give up the ability to wait for that task's result or cancel the task.

Source: https://developer.apple.com/documentation/swift/task/

Data race
VS
Race condition

Data race

Multiple threads access the resource in a concurrent environment

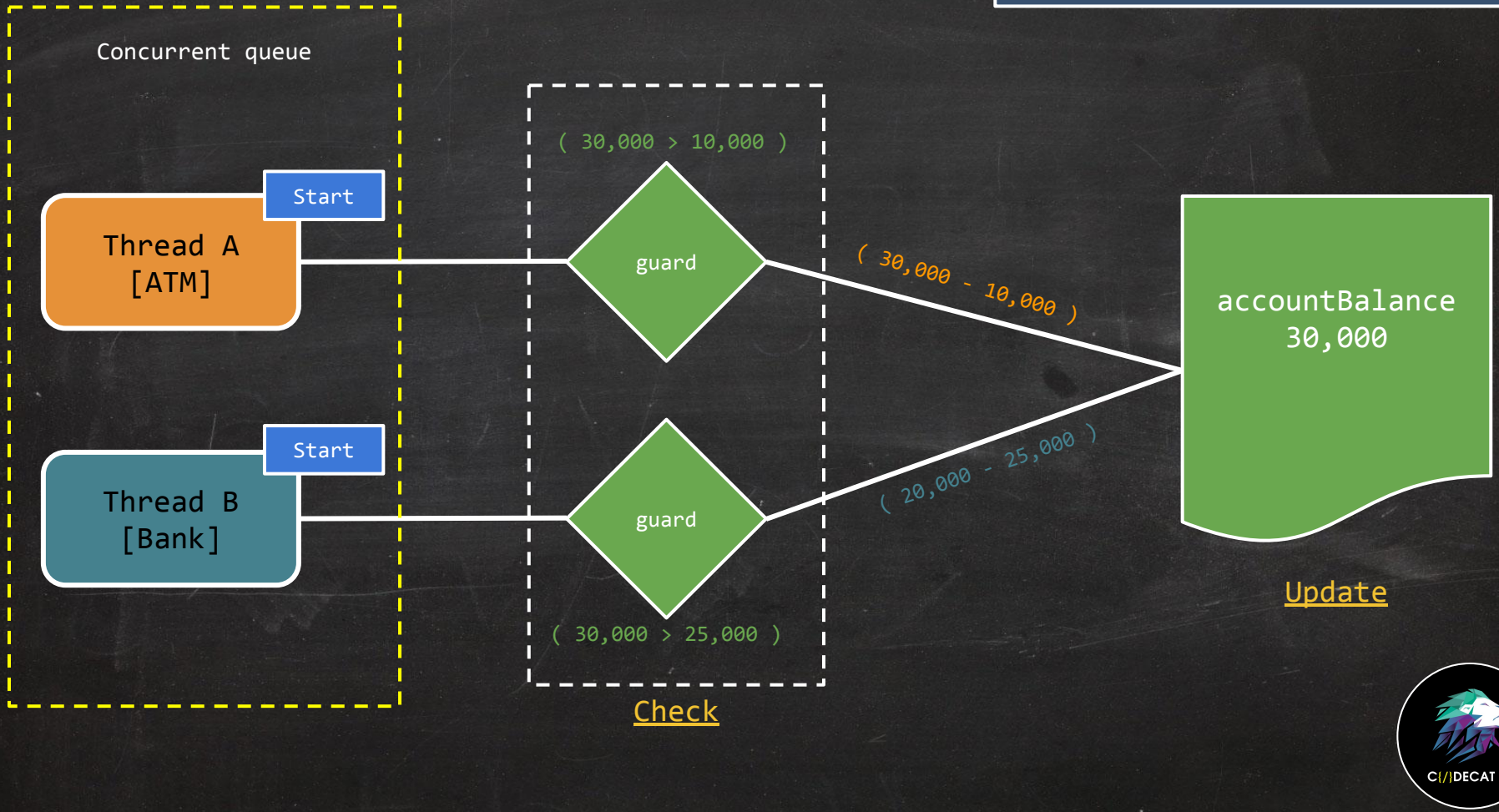
There are no checks or locks before the resource is accessed

Race condition

Multiple threads access the resource in a concurrent environment

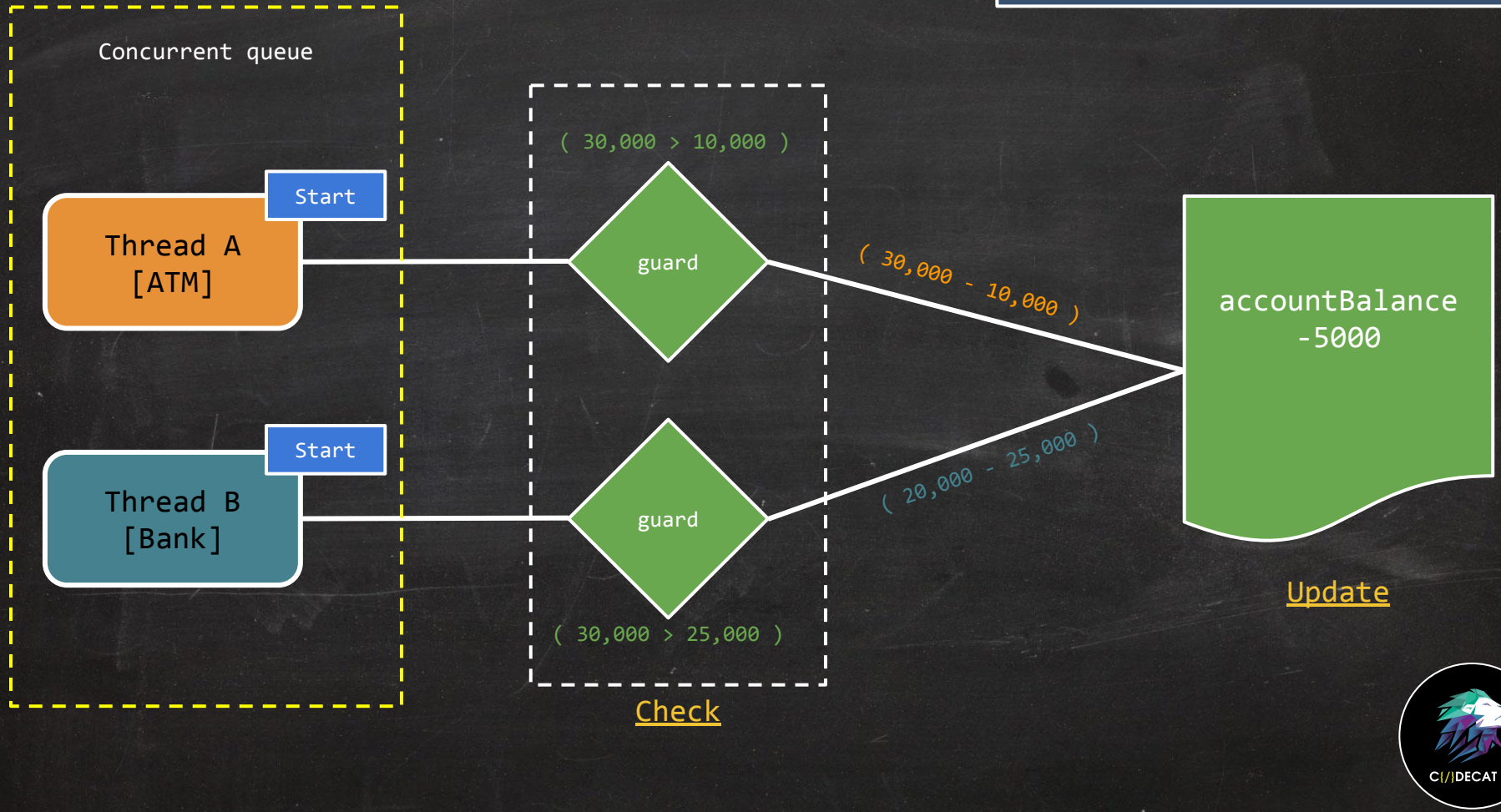
Follows check and update mechanism, it checks for a condition or validation before the resource is updated.

From semaphore video



C{/}DECAT 15

From semaphore video



C{/}DECAT 15

Do mention your feedback and questions in the comments

SUBSCRIBE

 @codecat15 |  @codecat15 |  @codecat15