# Grand Central Dispatch

# Lecture plan

- Queues
- Methods
- Concurrent perform
- Work item
- Semaphore
- Dispatch group
- Dispatch barrier
- Dispatch source
- Target Queue Hierarchy
- Dispatch IO

# Queues

- Serial
- Concurrent

```
class QueueTest1 {

    private let serialQueue = DispatchQueue(label: "serialTest")

    private let concurrentQueue = DispatchQueue(label: "concurrentTest", attributes:
.concurrent)

}
```

# Queues

- Global
- Main

```
class QueueTest2 {

    private let globalQueue = DispatchQueue.global()

    private let mainQueue = DispatchQueue.main
}
```

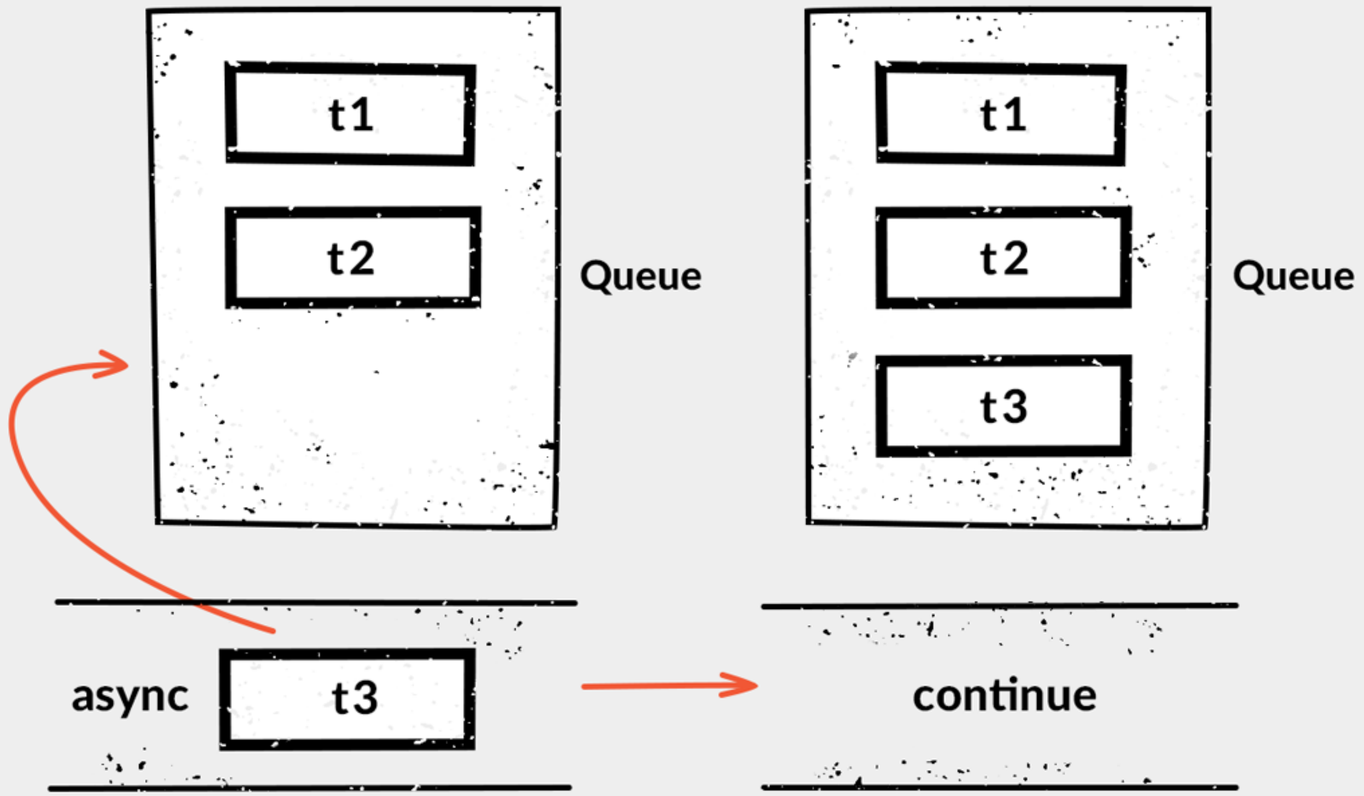
# Queues

```
class QueueTest {

    private let serialQueue = DispatchQueue(label: "serialTest")

    private let concurrentQueue = DispatchQueue(label: "concurrentTest", attributes:
.concurrent)

    private let globalQueue = DispatchQueue.global()

    private let mainQueue = DispatchQueue.main
}
```

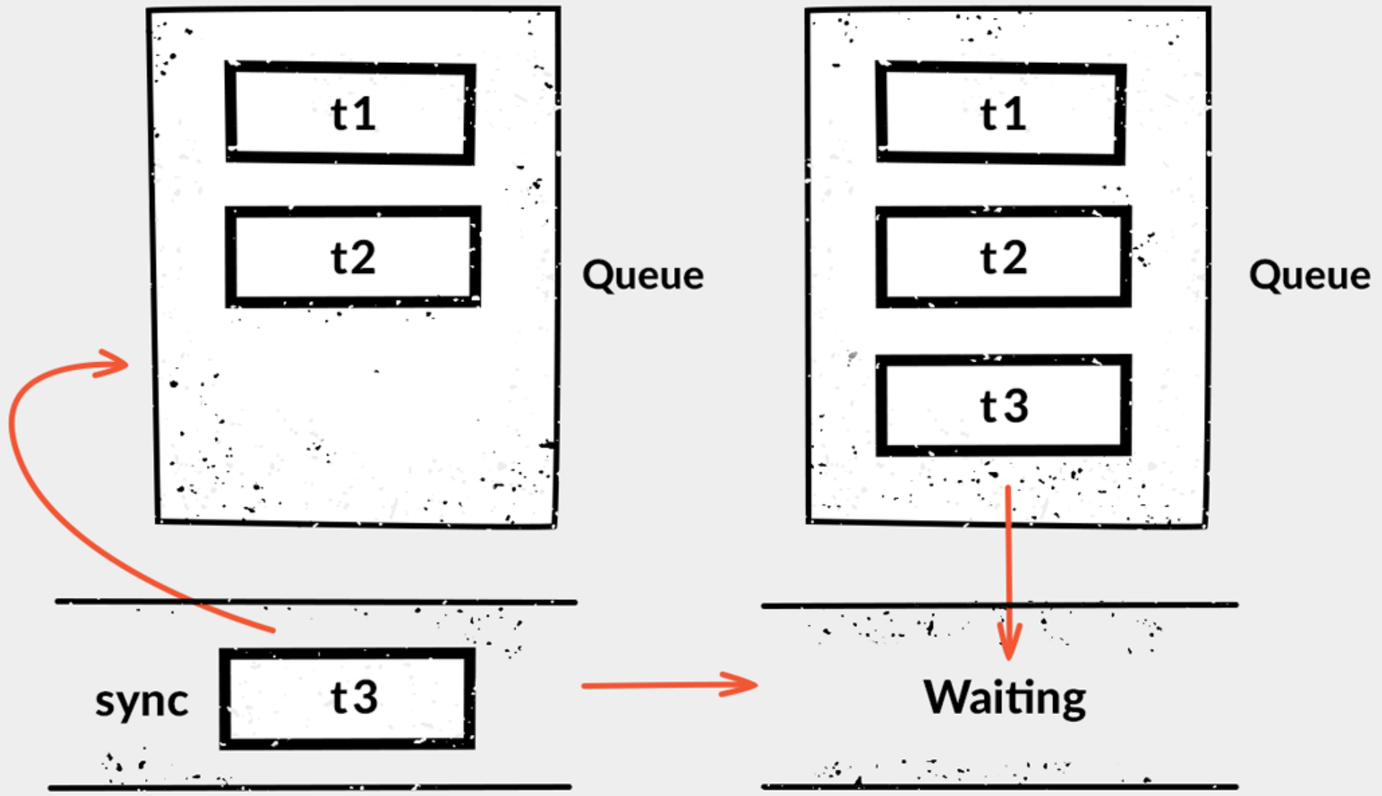
# Quality of service

```
class QosTest {
    private let backgroundQueue = DispatchQueue(label: "QosTest1", qos: .background)

    private let utilityQueue = DispatchQueue(label: "QosTest1", qos: .utility)

    private let userInitiatedQueue = DispatchQueue(label: "QosTest1", qos:
.userInitiated)

    private let userInteractiveQueue = DispatchQueue(label: "QosTest1", qos:
.userInteractive)

    //Global queue
    private let backgroundGlobalQueue = DispatchQueue.global(qos: .background)

    private let utilityGlobalQueue = DispatchQueue.global(qos: .utility)

    private let userInitiatedGlobalQueue = DispatchQueue.global(qos: .userInitiated)

    private let userInteractiveGlobalQueue = DispatchQueue.global(qos: .userInteractive)

    private let defaultGlobalQueue = DispatchQueue.global()
}
```

# Async vs Sync



# Async vs Sync



# Async vs Sync

```
class AsyncVsSyncTest1 {
    private let serialQueue = DispatchQueue(label:
"serialTest")

    func testSerial() {
        serialQueue.async {
            print("test1")
        }

        serialQueue.async {
            sleep(1)
            print("test2")
        }

        serialQueue.sync {
            print("test3")
        }

        serialQueue.sync {
            print("test4")
        }
    }
}
```

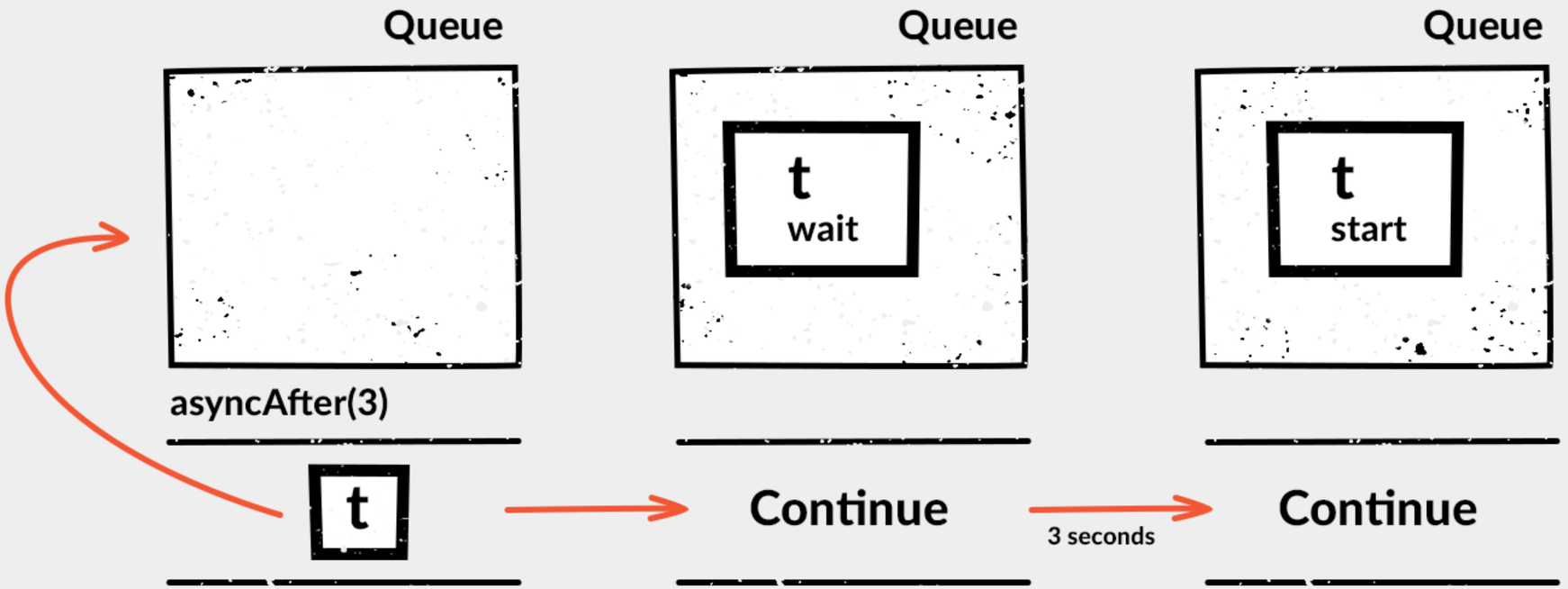
**Serial:**
**test1**
**test2**
**test3**
**test4**

# Async vs Sync

```
class AsyncVsSyncTest2 {

    private let concurrentQueue = DispatchQueue.global()

    func testConcurrent() {

        concurrentQueue.async {
            print("test1")
        }

        concurrentQueue.async {
            print("test2")
        }

        concurrentQueue.sync {
            print("test3")
        }

        concurrentQueue.sync {
            print("test4")
        }
    }
}
```

**Concurrent:**
**test3 -> test4**

# Async after


Queue
Queue
Queue
t
wait
t
start
asyncAfter(3)
t
Continue
3 seconds
Continue

# Async after

```
class AsyncAfterTest {

    private let concurrentQueue = DispatchQueue(label: "AsyncAfterTest", attributes:
.concurrent)

    func test() {
        concurrentQueue.asyncAfter(deadline: .now() + 3, execute: {
            print("test")
        })
    }
}
```

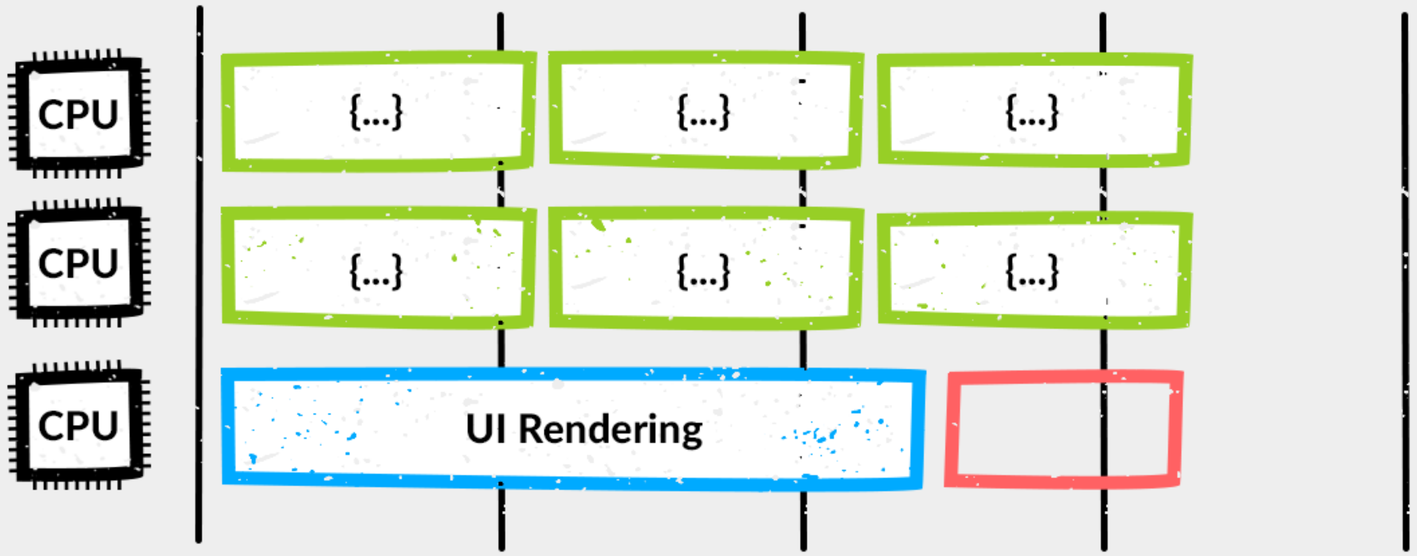
**...**
**3 seconds**
**...**
**test**

# Async after

```
class AsyncAfterTest2 {

    private let serialQueue = DispatchQueue(label: "StringAsyncAfterTest2")

    func test() {

        serialQueue.async {

            sleep(3)
            print("finish")
        }

        serialQueue.asyncAfter(deadline: .now() + 1, execute: {
            print("test")
        })
    }
}
```

...
3 seconds
...
finish
test

# Concurrent perform

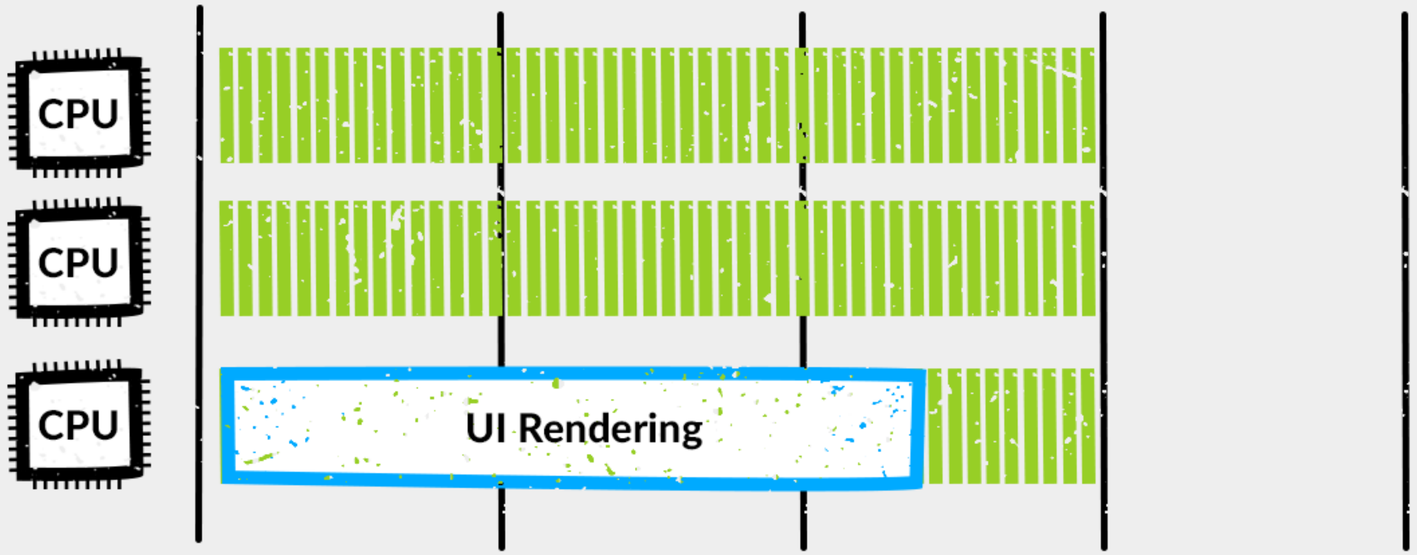


# Concurrent perform

```
class ConcurrentPerformTest {

    func test() {
        DispatchQueue.concurrentPerform(iterations: 3, execute: { i in
            print(i)
        })
    }
}
```

**0**
**1**
**2**

# Concurrent perform



# Concurrent perform

```
class ConcurrentPerformTest2 {

    func test() {
        DispatchQueue.concurrentPerform(iterations: 1000, execute: { i in
            print(i)
        })
    }
}
```

**0**
**2**
**1**
**...**
**999**

# Work item

```
class DispatchWorkItemTest1 {

    private let queue = DispatchQueue(label: "DispatchWorkItemTest1", attributes:
.concurrent)

    func testNotify() {

        let item = DispatchWorkItem {
            print("test")
        }

        item.notify(queue: DispatchQueue.main, execute: {
            print("finish")
        })
        queue.async(execute: item)
    }
}
```

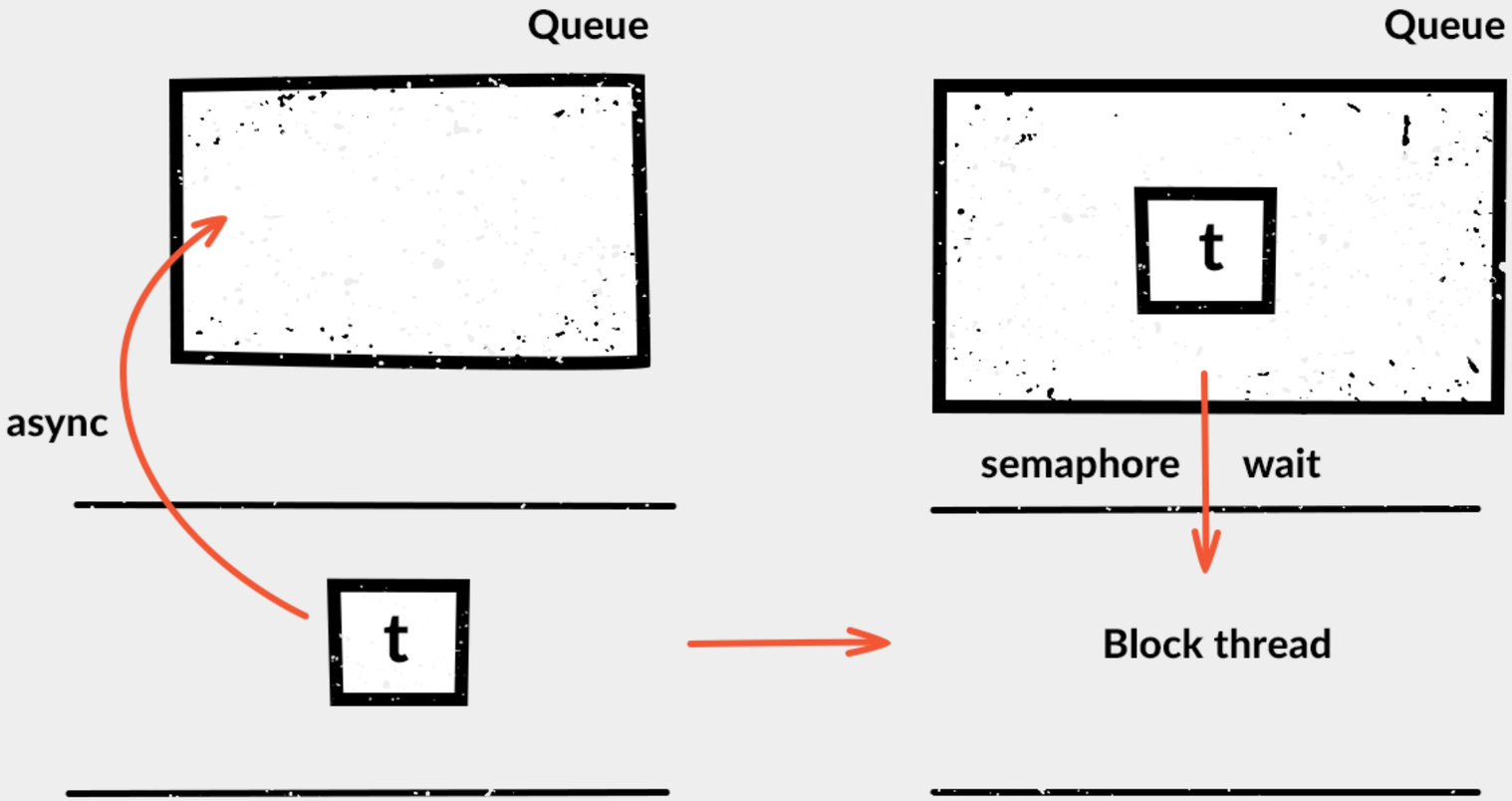
**test**
**finish**

# Work item

```
class DispatchWorkItemTest2 {

    private let queue = DispatchQueue(label: "DispatchWorkItemTest2")

    func testCancel() {

        queue.async {
            sleep(1)
            print("test1")
        }
        queue.async {
            sleep(1)
            print("test2")
        }

        let item = DispatchWorkItem {
            print("test")
        }
        queue.async(execute: item)

        item.cancel()
    }
}
```

**test1**
**...**
**1 second**
**...**
**test2**

# Semaphore



# Semaphore

```
class SemaphoreTest {

    private let semaphore = DispatchSemaphore(value: 0)

    func test() {

        DispatchQueue.global().async {

            sleep(3)
            print("1")
            self.semaphore.signal()
        }
        semaphore.wait()
        print("2")
    }
}
```

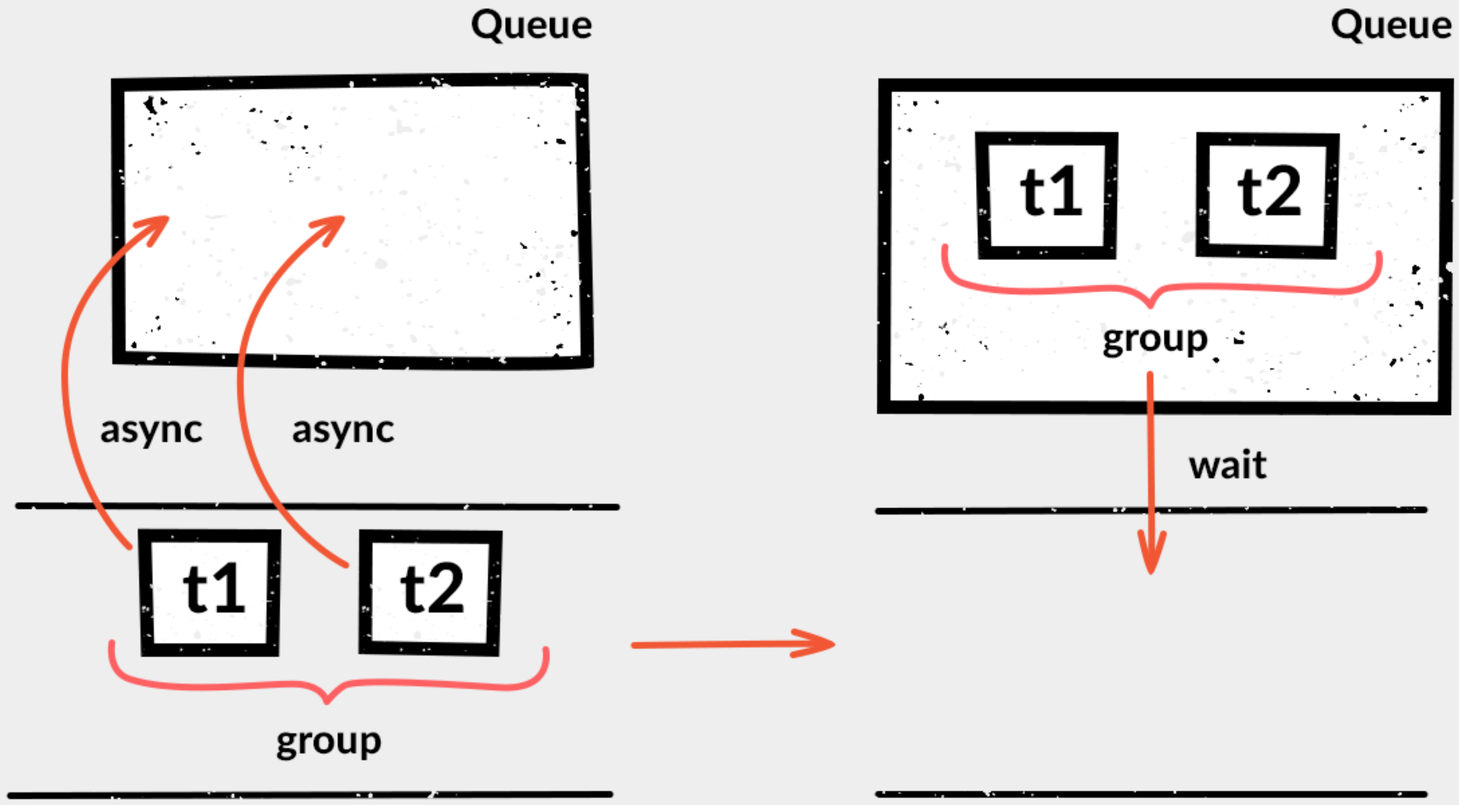
**1**
**2**

# Semaphore

```
class SemaphoreTest2 {
    private let semaphore = DispatchSemaphore(value: 2)

    func doWork() {

        semaphore.wait()
        print("test")
        sleep(3) //Do something
        semaphore.signal()
    }

    func test() {

        DispatchQueue.global().async {
            self.doWork()
        }
        DispatchQueue.global().async {
            self.doWork()
        }
        DispatchQueue.global().async {
            self.doWork()
        }
    }
}
```

**test**
**test**
**...**
**3 seconds**
**...**
**test**

# Dispatch group



# Dispatch group

```
class DispatchGroupTest1 {
    private let group = DispatchGroup()
    private let queue = DispatchQueue(label:
"DispatchGroupTest1", attributes: .concurrent)

    func testNotify() {

        queue.async(group: group) {

            sleep(1)
            print("1")
        }
        queue.async(group: group) {

            sleep(2)
            print("2")
        }
        group.notify(queue: DispatchQueue.main) {
            print("finish all")
        }
    }
}
```

**1**
**2**
**...**
**2 seconds**
**...**
**finish all**

# Dispatch group

```
class DispatchGroupTest2 {
    private let group = DispatchGroup()
    private let queue = DispatchQueue(label:
"DispatchGroupTest2", attributes: .concurrent)

    func testWait() {
        group.enter()
        queue.async {
            sleep(1)
            print("1")
            self.group.leave()
        }
        group.enter()
        queue.async {
            sleep(2)
            print("2")
            self.group.leave()
        }
        group.wait()
        print("finsh all")
    }
}
```

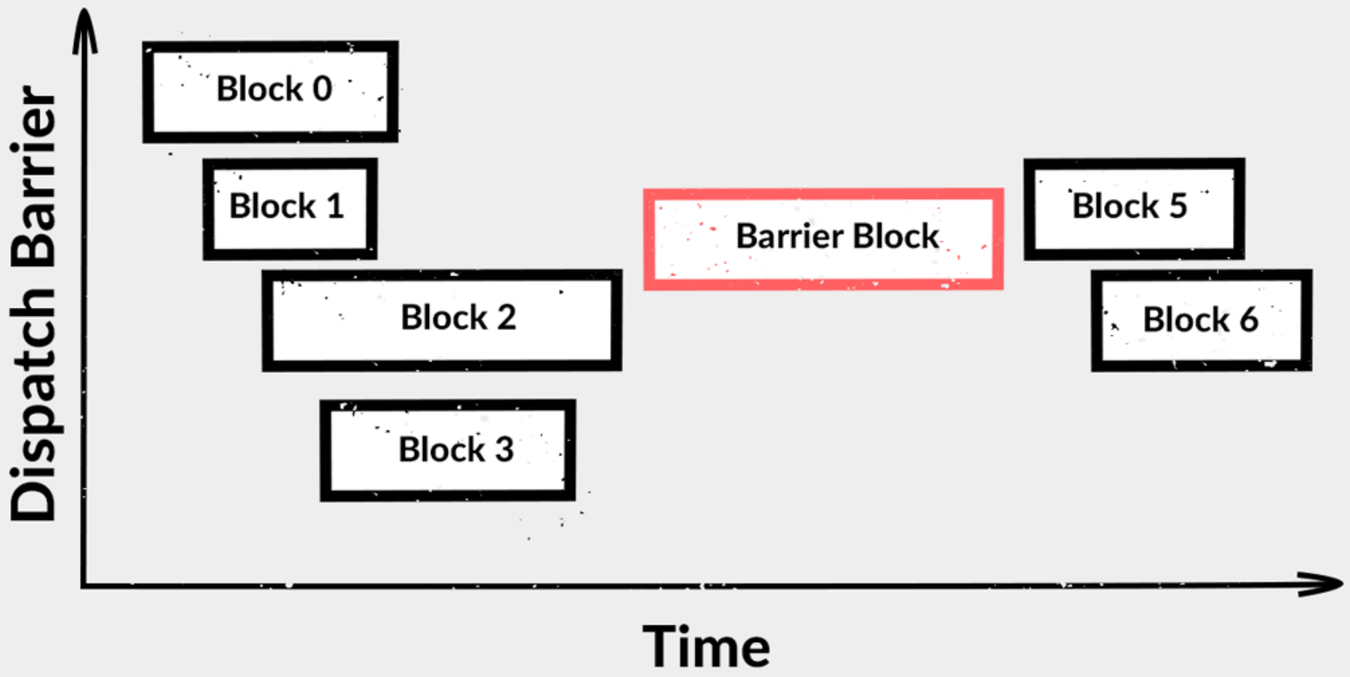
**...**
**1 second**
**...**
**1**
**...**
**1 second**
**...**
**2**
**finish all**

# Dispatch group

```
class DispatchGroupTest3 {
    private let group = DispatchGroup()
    private let queue1 = DispatchQueue(label: "DispatchGroupTest3",
attributes: .concurrent)
    private let queue2 = DispatchQueue(label: "DispatchGroupTest3_serial")

    func test() {
        queue1.async(group: group) {
            sleep(1)
            print("1")
        }
        queue2.async(group: group) {
            sleep(2)
            print("2")
        }
        queue2.async(group: group) {
            sleep(3)
            print("3")
        }
        group.notify(queue: DispatchQueue.main) {
            print("finish all")
        }
    }
}
```

**...**
**1 second**
**...**
**1**
**...**
**1 second**
**...**
**2**
**...**
**3 seconds**
**...**
**3**
**finish all**

# Dispatch barrier



# Dispatch barrier

```
class DispatchBarrierTest {

    private let queue = DispatchQueue(label: "DispatchBarrierTest", attributes:
.concurrent)

    private var internalTest: Int = 0

    func setTest(_ test: Int) {

        queue.async(flags: .barrier) {
            self.internalTest = test
        }
    }

    func test() -> Int{

        var tmp: Int = 0
        queue.sync {
            tmp = self.internalTest
        }
        return tmp
    }
}
```

# Dispatch source

- Timer dispatch source
- Signal dispatch source
- Descriptor dispatch source
- Process dispatch source

# Dispatch source

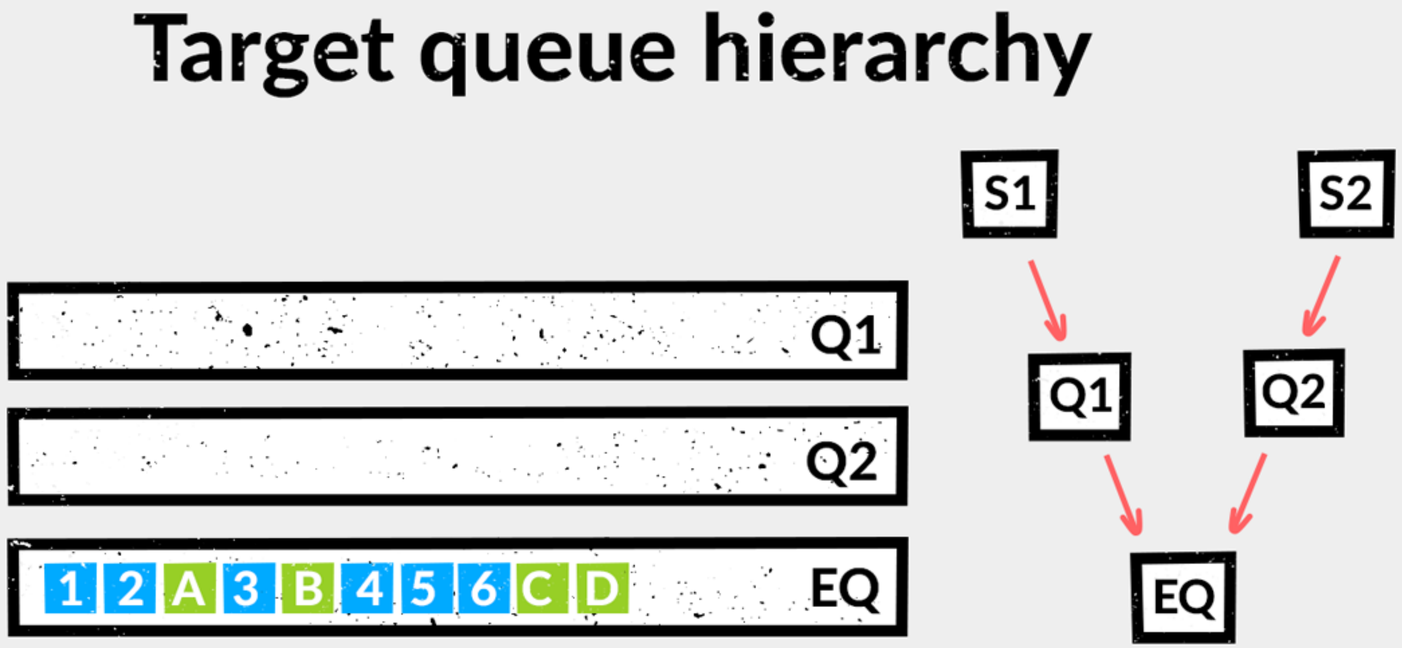
```
class DispatchSourceTest1 {

    private let source = DispatchSource.makeTimerSource()

    func test() {

        source.setEventHandler {
            print("test")
        }
        source.schedule(deadline: .now(), repeating: 5)
        source.activate()
    }
}
```

**test**
**...**
**5 seconds**
**...**
**test**
**....**

# Dispatch source

```
class DispatchSourceTest2 {

    private let source = DispatchSource.makeUserDataAddSource(queue: DispatchQueue.main)

    init() {

        source.setEventHandler {
            print(self.source.data)
        }
        source.activate()
    }

    func test() {

        DispatchQueue.global().async {
            self.source.add(data: 10)
        }
    }
}
```

# Target queue hierarchy

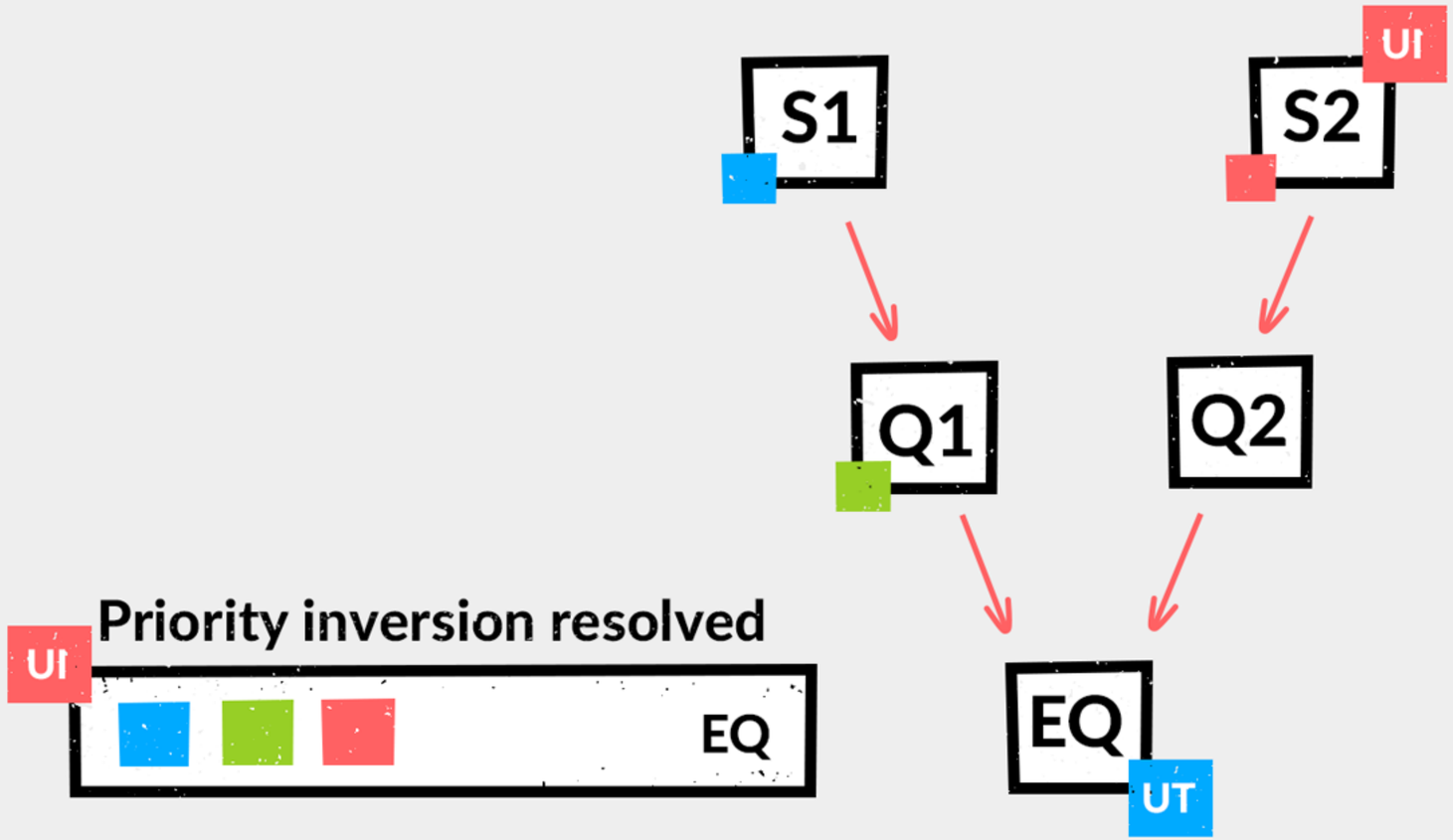


# Target queue hierarchy

```
class TargetQueueHierarchyTest1 {
    private let targetQueue = DispatchQueue(label: "TargetQueue")

    func test() {
        let queue1 = DispatchQueue(label: "Queue1", target: targetQueue)
        let dispatchSource1 = DispatchSource.makeTimerSource(queue: queue1)
        dispatchSource1.setEventHandler {
            print("test1")
        }
        dispatchSource1.activate()

        let queue2 = DispatchQueue(label: "Queue2", target: targetQueue)
        let dispatchSource2 = DispatchSource.makeTimerSource(queue: queue2)
        dispatchSource2.setEventHandler {
            print("test2")
        }
        dispatchSource2.activate()
    }
}
```

Target queue hierarchy

# QoS and Target queue hierarchy



# Target queue hierarchy

```
class TargetQueueHierarchyTest2 {
    private let targetQueue = DispatchQueue(label: "TargetQueue", qos: .utility)

    func test() {

        let queue1 = DispatchQueue(label: "Queue1", target: targetQueue)
        let dispatchSource1 = DispatchSource.makeTimerSource(queue: queue1)
        dispatchSource1.setEventHandler(qos: .userInitiated) {
            print("test1")
        }
        dispatchSource1.activate()

        let queue2 = DispatchQueue(label: "Queue2", target: targetQueue)
        let dispatchSource2 = DispatchSource.makeTimerSource(queue: queue2)
        dispatchSource2.setEventHandler {
            print("test2")
        }
        dispatchSource2.activate()
    }
}
```

# Dispatch IO

```
class GcdChannelTest {
    private let queue = DispatchQueue(label: "GcdChannelTest", attributes:
.concurrent)
    private var channel: DispatchIO?

    func test() {

        guard let filePath = Bundle.main.path(forResource: "test", ofType: "") else {
return }
        channel = DispatchIO(__type: DispatchIO.StreamType.stream.rawValue, path:
filePath, oflag: O_RDONLY, mode: 0, queue: DispatchQueue.global(), handler: { error in
            //Handle error
        })
        channel?.read(offset: 0, length: Int.max, queue: queue) { (done, data, error)
in
            if data != nil {
                //Handle data
            }
            if error != 0 {
                //Handle error
            }
        }
    }
}
```