TODAY

• Task Dispatching

• Task Groups

• Semaphores

• Operations
RECALL

- Threads
  - faster execution
  - responsiveness
  - optimized resource consumption
- Dispatch queues
  - main queue (serial, UI) \((\text{DispatchQueue.main})\)
  - Six global queues \((\text{DispatchQueue.global(qos: ...)})\)
- QOS
  - .userInteractive (ex: animations)
  - .userInitiated (ex: read a document or a database and update)
  - .utility (ex: networking or continuous data feeds)
  - .background (user should know be aware)
  - .default and .unspecified

ADDING TASKS TO QUEUES

- careful of \textit{retain} cycles
- careful of which tasks go where

```swift
DispatchQueue.global(qos: .utility).async {
  [weak self] in
  guard let self = self else {
    return
  }
  // Perform your work here
  // ... 
  // Switch back to the main queue to
  // update your UI
  DispatchQueue.main.async {
    self.textLabel.text = "New articles available!"
  }
}
```
DISPATCH GROUPS

- Track completion of group of tasks:

```swift
let group = DispatchGroup()
someQueue.async(group: group) { ... your work ... }
someQueue.async(group: group) { ... more work .... }
someOtherQueue.async(group: group) { ... other work ... }
group.notify(queue: DispatchQueue.main) { [weak self] in
    self?.textLabel.text = "All jobs have completed"
}

// or
if group.wait(timeout: .now() + 60) == .timedOut {
    print("The jobs didn't finish in 60 seconds")
}
```

- Group tasks can be in multiple queues
- (A-)Synchronous notification at completion
- Can add more tasks up until none outstanding

Demo
WRAPPING ASYNC METHODS

- What if your async method calls another async method?

```swift
queue.dispatch(group: group) {

    someAsyncMethod {
        // Perform your work here,
        // count goes back to 1 once complete
    }
}
```

- What if your async method calls another async method?

```swift
queue.dispatch(group: group) {
    // count is 1
    group.enter()
    // count is 2
    someAsyncMethod {
        defer {
            group.leave()
        }
        // Perform your work here,
        // count goes back to 1 once complete
    }
}
```
WRAPPING ASYNC METHODS

• Might want to have `someAsyncMethod()` call group updates directly.

```swift
func myAsyncAdd(lhs: Int, rhs: Int, completion: @escaping (Int) -> Void) {
    // Lots of cool code here
    completion(lhs + rhs)
}
func myAsyncAddForGroups(group: DispatchQueue, lhs: Int, rhs: Int,
                          completion: @escaping (Int) -> Void) {
    group.enter()
    myAsyncAdd(first: first, second: second) { result in
        defer { group.leave() }
        completion(result)
    }
}
```

• `@escaping` means closure is allowed to escape

SEMAPHOPHORES

• Assume downloading a lot of data from the network
  • dispatch queues to offload work
  • dispatch groups for completion notification
  • but what if only want four simultaneous downloads?

• Straightforward semaphores
  • create w/ number of resources
  • wait on it to get access
RACE CONDITIONS

- Threads 1 and 2 both execute:
  - `count += 1`

- Very difficult to debug ("Heisenbug")
COULD SOLVE W/ A SERIAL Q

```swift
private let threadSafeCountQueue = DispatchQueue(label: "...")
private var _count = 0
public var count: Int {
    get {
        return threadSafeCountQueue.sync {
            _count
        }
    }
    set {
        threadSafeCountQueue.sync {
            _count = newValue
        }
    }
}
```

- queues serial by default

OR A DISPATCH BARRIER

```swift
private let threadSafeCountQueue = DispatchQueue(label: "...", attributes: .concurrent)
private var _count = 0
public var count: Int {
    get {
        return threadSafeCountQueue.sync {
            return _count
        }
    }
    set {
        threadSafeCountQueue.async(flags: .barrier) {
            unowned self in
            self._count = newValue
        }
    }
}
```
DEADLOCK

- With semaphores:
  - Thread 1 acquires A, waits to get B
  - Thread 2 acquires B, waits to get A

- How to prevent?
  - Always possible? Why or why not?

- Most common in iOS w/ .sync() against current dispatch queue

PRIORITY INVERSION

- When low-priority job gets higher priority than high-priority job...

- Dispatch queues have different QOS levels (basically priorities)
  - tasks added w/ explicit QOS levels
  - system adapts queue QOS to highest job QOS
  - low-priority tasks on same queue might end up running w/ high priority

- Happens if resources shared among high-/low-priority queues
  - and low priority queue gets the resource first
PRIORITY INVERSION

Demo
ALTERNATIVE TO GCD: OPERATIONS

- Advantages:
  - inter-operation dependencies
  - clean way to pass results from one to next (func. prog.)
  - reusability
  - cancelling
  - KVO notifications

- Details
  - on top of GCD
  - object-oriented
  - Operation is abstract class
    - must be subclassed

OPERATION QUEUES

- BlockOperation works by:
  - subclassing an Operation
  - adding a closure
  - adding an array of closures

- Useful operation queue methods / properties
  - .maxConcurrentOperationCount = 4
  - .cancelAllOperations()
  - .waitUntilAllOperationsAreFinished()
  - etc.
OPERATIONS

- BlockOperation is a convenience subclass of Operation
- .start()

```swift
let operation = BlockOperation {
    print("2 + 3 = \(2 + 3)")
}
operation.start()

or

let sentence = "Draymond Green is back, baby!"
let wordOperation = BlockOperation()
for word in sentence.split(separator: " ") {
    wordOperation.addExecutionBlock {
        print(word)
    }
}
wordOperation.start()
```

Demo
DEPENDENCIES

- Using GCD

```swift
let network = NetworkClass()
network.onDownloaded { raw in
    guard let raw = raw else { return }
    let decrypt = DecryptClass(raw)
decrypt.onDecrypted { decrypted in
        guard let decrypted = decrypted else { return }
        let tilt = ReadClass(decrypted)
tilt.onRead { read in
            guard let read = read else { return }
        }
    }
}
```

- Using dependencies

```swift
let networkOp = NetworkImageOperation()
let decryptOp = DecryptOperation()
let readOp = ReadOperation()
decryptOp.addDependency(networkOp)
readOp.addDependency(decryptOp)
```

DEADLOCKS!

![Diagram showing no deadlock and deadlock warning](image-url)
Demo