TODAY

- Multi-Threading
  - Grand Central Dispatch
  - URLSessions
  - Dispatch Groups

MULTI-THREADING

- What for?
  - handling long-running activities
  - blocking activities
  - computationally expensive

- How? Grand Central Dispatch
  - most used are the queues, but also:
    - lock-based synchronization
    - barriers
    - etc.
DISPATCH QUEUES

- **Main Queue (serial)**
  - all UI activity here
  - no long-running, non-UI activity here
  - want UI to be responsive
    ```swift
    let mainQ = DispatchQueue.main
    ```

- **Global Queues (concurrent)**
  ```swift
  // high priority, short
  let backgroundQ = DispatchQueue.global(qos: .userInteractive)
    .userInitiated // high priority, longer
    .background // not user-created, slow
    .utility // system maintenance
  ```

- **Adding to queues**
  ```swift
  backgroundQ.async { <#code#> } // most common
  backgroundQ.sync { <#code#> }
  ```

SERIAL VS CONCURRENT

[Diagram showing serial and concurrent task execution]
SYNC VS ASYNC

- Synchronous (caller waits until finished)
  
  `DispatchQueue.sync(qos: .userInteractive)`

- Asynchronous (call returns immediately)

  `DispatchQueue.async(qos: .background)`

CUSTOM QUEUES

- Custom Queues
  - can be `serial` (single thread)

    `let serialQ = DispatchQueue(label: "MySerial")`

  - or `concurrent` (multiple threads)

    `let concurrentQ = DispatchQueue(label: "My Concurrent", attributes: .concurrent)`
GCD DETAILS

- Threads
  - faster execution
  - responsiveness
  - optimized resource consumption

- Dispatch queues (ordered sets of tasks)
  - main queue (serial, UI, short fast jobs) (DispatchQueue.main)
  - Six global queues (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 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!"
    }
}
```
SESSIONS AND URLs

• Also: URLSessions
  • Lets you fetch the contents of an http URL into a Data off the main queue!
  • To repeat: not on the main queue
  • Means we must dispatch back to the main queue for UI stuff

SESSIONS

• Use URLSession to create a session
  • groups many requests (think of tab in browser), caches…
  • configure with URLSessionConfiguration object
    • caching, cookies, credentials
• The URLSession is used to create URLSessionTask instances
  • fetch data to your app
  • upload and download files
  • just give it a URL
ASIDE: SCROLLVIEW COORDS

• Converting coords from scrollview context to content context
• Image scroller demo
  • w/ pic in assets
• Image scroller demo
  • pic accessed from URL
  • split screen VC
    • segues
• Loading from URLs

TRANSLATING

• If have a tap recognizer and want to center screen around it
  • must convert coordinates:
    • from the scrollview context
    • to the image (contentSize) context

```swift
@objc func tap(recog: UITapGestureRecognizer) {
    if recog.state == .ended {
        let location = recog.location(in: scrollView)
        let pt = imageView.convert(location, from: scrollView)
        let visible = imageView.convert(scrollView.bounds, from: scrollView)
        scrollView.contentOffset.x += (pt.x - visible.midX) * scrollView.zoomScale
        scrollView.contentOffset.y += (pt.y - visible.midY) * scrollView.zoomScale
    }
}
```
URL

• Local through (usually the “main”) bundle
  • URL = Bundle.main.URL(forContent: “pretty”, withExtension: “jpg”)

• Remote through normal URL
  • URL = URL(string: “http://kelehers.me/orion.jpg“)