CSC 471 / 371 Mobile Application Development for iOS

Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu
@DePaulSWEng

Multi-Threads & Grand Central Dispatch

Outline

- Multi-threads
- Grand Central Dispatch
- Dispatch queues
  - Main queue
  - Background queue
- Progress View & Activity indicator

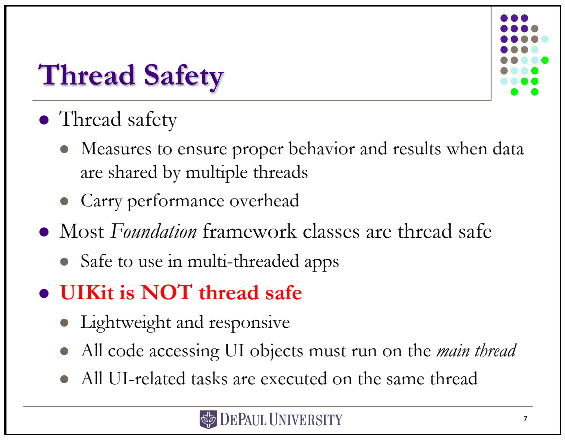
Multi-Threads

Threads

- Lightweight mechanism to support multiple *concurrent* execution paths of code within an app
  - iOS supports multi-threading
  - Thread execution is managed by the system
- Potential benefits
  - Improve perceived responsiveness
    - Perform time-consuming tasks in the background
  - Improve real-time performance on multicore systems.
    - iPhone 5/5s/6/6s: dual-core
    - iPad 2/3/4, iPad mini 2/3/4, iPad Pro: dual-core
    - iPad Air 2: triple-core

Manage Multiple Threads

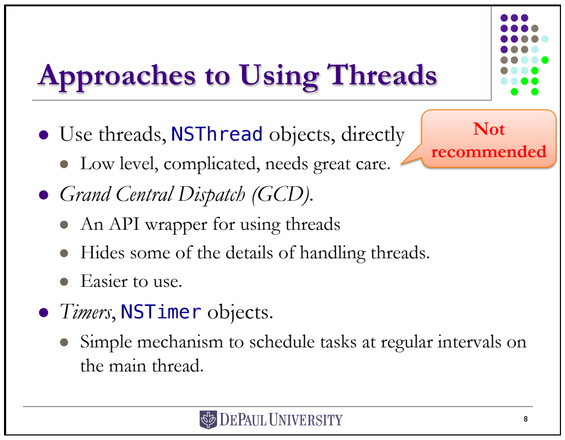
- Each app has at least one thread – the *main* thread
  - The main thread handles all UI related tasks
  - Additional threads can be created if needed
- Complications of multiple threads within an app
  - Shared memory space.
  - All threads have access to all the data in the app
  - Uncoordinated access of data may lead to **data corruption, erroneous results,** and **unpredictable behaviors**
  - Subtle bugs, *Heisenbugs,* – dead lock, race condition, etc.



Thread Safety

- Thread safety
  - Measures to ensure proper behavior and results when data are shared by multiple threads
  - Carry performance overhead
- Most *Foundation* framework classes are thread safe
  - Safe to use in multi-threaded apps
- **UIKit is NOT thread safe**
  - Lightweight and responsive
  - All code accessing UI objects must run on the *main thread*
  - All UI-related tasks are executed on the same thread

DEPAUL UNIVERSITY 7

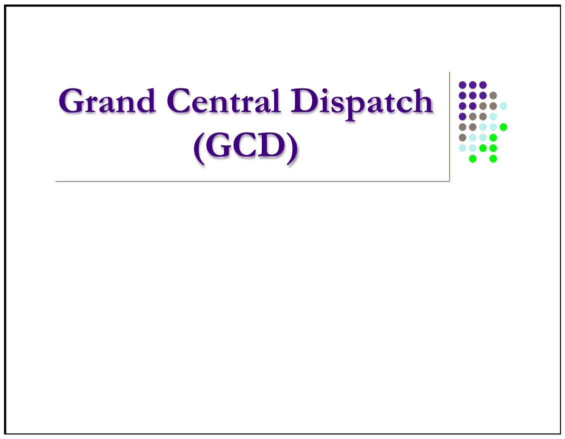


Approaches to Using Threads

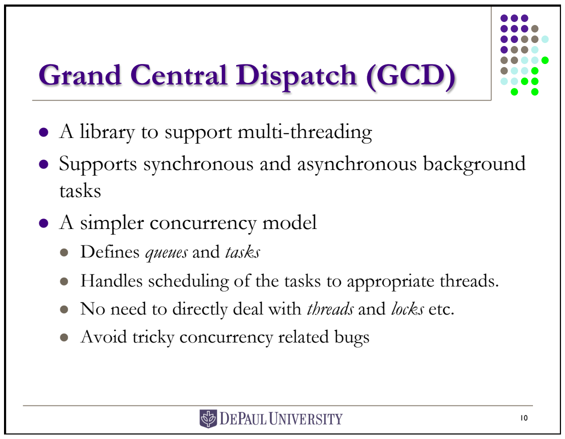
- Use threads, `NSThread` objects, directly
  - Low level, complicated, needs great care.
- *Grand Central Dispatch (GCD).*
  - An API wrapper for using threads
  - Hides some of the details of handling threads.
  - Easier to use.
- *Timers*, `NSTimer` objects.
  - Simple mechanism to schedule tasks at regular intervals on the main thread.

Not recommended

DEPAUL UNIVERSITY 8



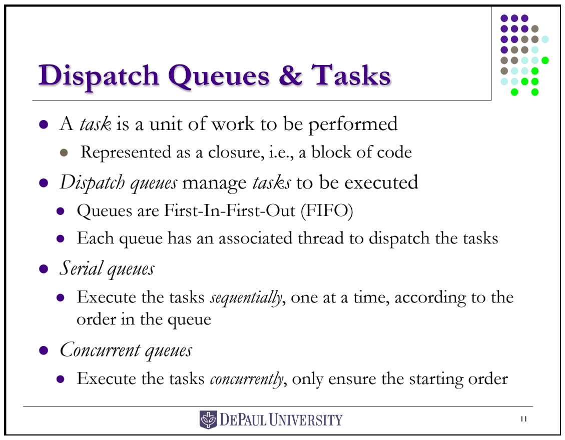
Grand Central Dispatch (GCD)



Grand Central Dispatch (GCD)

- A library to support multi-threading
- Supports synchronous and asynchronous background tasks
- A simpler concurrency model
  - Defines *queues* and *tasks*
  - Handles scheduling of the tasks to appropriate threads.
  - No need to directly deal with *threads* and *locks* etc.
  - Avoid tricky concurrency related bugs

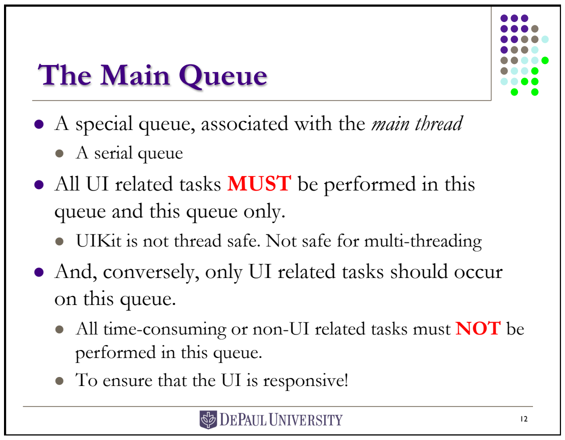
DEPAUL UNIVERSITY 10



Dispatch Queues & Tasks

- A *task* is a unit of work to be performed
  - Represented as a closure, i.e., a block of code
- *Dispatch queues* manage *tasks* to be executed
  - Queues are First-In-First-Out (FIFO)
  - Each queue has an associated thread to dispatch the tasks
- *Serial queues*
  - Execute the tasks *sequentially*, one at a time, according to the order in the queue
- *Concurrent queues*
  - Execute the tasks *concurrently*, only ensure the starting order

DEPAUL UNIVERSITY 11



The Main Queue

- A special queue, associated with the *main thread*
  - A serial queue
- All UI related tasks **MUST** be performed in this queue and this queue only.
  - UIKit is not thread safe. Not safe for multi-threading
- And, conversely, only UI related tasks should occur on this queue.
  - All time-consuming or non-UI related tasks must **NOT** be performed in this queue.
  - To ensure that the UI is responsive!

DEPAUL UNIVERSITY 12

The Main Dispatch Queues

- The main queue

```
let mainQ: dispatch_queue_t = dispatch_get_main_queue()
let mainQ: NSOperationQueue = NSOperationQueue.mainQueue()
```

OO API

- Dispatch a task *asynchronously* on a different queue

```
dispatch_async(queue) { code }
```

  - Enqueue a task for *asynchronous* execution
  - Returns immediately
    - Move work off the main thread
    - Deferred execution of tasks
    - Automatic concurrency

Perform UI Related Tasks Safely

- To off load some tasks to other queues
- Dispatch a task from the main queue to another

```
dispatch_async(notTheMainQueue) {
    Some time consuming tasks, no UI access
    dispatch_async(dispatch_get_main_queue()) {
        Update UI with the results
    }
}
```

Update to UI must be executed on the main queue

Other System Queue

- System provides several *concurrent* queues
- Support different levels of priorities

`QOS_CLASS_USER_INTERACTIVE`	*quick and high priority*
`QOS_CLASS_USER_INITIATED`	*high priority, might take time*
`QOS_CLASS_UTILITY`	*long running*
`QOS_CLASS_BACKGROUND`	*user not concerned with this (prefetching, etc.)*

- Get the system queue with a given priority

```
let qos = Int(<one of the priorities above>.rawValue)
let queue = dispatch_get_global_queue(qos, 0)
```

Create Serial Queues

- You can create your own serial queue if needed

```
let serialQ = dispatch_queue_create("name",
                                     DISPATCH_QUEUE_SERIAL)
```

- Need the tasks to be executed sequentially
  - Downloading a bunch of things things from a certain website but you don't want to deluge that website, so you queue the requests up serially
  - The things you are doing depend on each other in a serial fashion

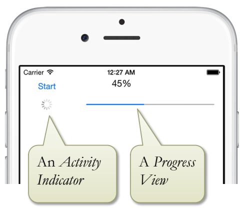
Dispatch with a Delay

Convert the delay to nanoseconds

```
let delayInSeconds = 5.0
let delay = Int64(delayInSeconds*Double(NSEC_PER_SEC))
let dispatchTime = dispatch_time(DISPATCH_TIME_NOW, delay)
dispatch_after(dispatchTime, dispatch_get_main_queue()) {
    Do something in the future on the main queue
}
```

The Progress View Demo App

- Simulating a time consuming task triggered by the "*Start*" button.
- An animated *Activity Indicator,* while the task is in progress
  - Select "*Hidden when stopped*"
- A *Progress View* showing the progress of the task



The Outlet

- Connect outlets to the *Activity Indicator*, the *Progress View* and the *Label* in the view

```
class ViewController: UIViewController {

    @IBOutlet weak var indicator: UIActivityIndicatorView!

    @IBOutlet weak var progress: UIProgressView!

    @IBOutlet weak var label: UILabel!

    …
}
```

DEPAUL UNIVERSITY 19

The Action

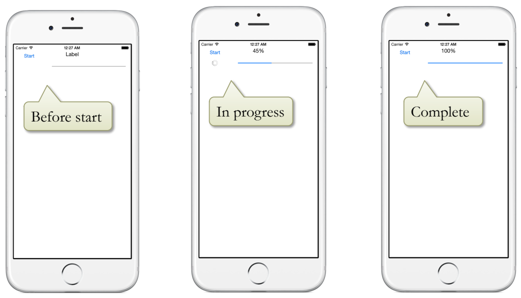
```
@IBAction func start(sender: UIButton) {
    indicator.startAnimating()
    let queue = dispatch_get_global_queue(
                    Int(QOS_CLASS_BACKGROUND.rawValue), 0)
    dispatch_async(queue) {
        for i in 0 ... 100 {
            dispatch_async(dispatch_get_main_queue()) {
                self.progress.progress = Float(i) / 100
                self.label.text = "\(i)%"
                if (i == 100) {
                    self.indicator.stopAnimating()
                }
            }
            usleep(100_000) // microseconds
        }
    }
}
```

The background queue

Update UI

Sleep for 0.1s. Do work in the background

Run the Progress View Demo



DEPAUL UNIVERSITY 21

Sample Code

- Progress View.zip

DEPAUL UNIVERSITY 22

Next …

- View animation & transition
- 2-D graphics drawing

✧ iOS is a trademark of Apple Inc.

DEPAUL UNIVERSITY 23