Concurrent Programming with Grand Central Dispatch

An overview of concepts, functions, and uses.

About Me

- Michael Tackes
- Started coding ~2011
- Codecademy in 2012
  - Forums moderator
- Madison College in 2013
  - Mobile/Web programs
  - Graduating by 2015

Sequential Programming

- One task at a time
- Executed in order
- Easy to reason about
- Easy to debug
- Fine for most tasks
- Very bad for some tasks

Problems with Sequential

- I/O operations
  - Disk operations
  - Network operations
  - Large files
  - Unreliable networks
- Complex computations
- Responsive user interactions

Concurrent Programming

- Multiple tasks at once!
- I/O in the background
- Complex computations in background
- Takes advantage of multi-core

Problems with Concurrent

- Multiple tasks at once
- No guaranteed execution order
- Difficult to reason about
- Difficult to debug
- Doesn't necessarily speed up code
- Generally difficult to write well

A programmer had a problem. He thought to himself, "I know, I'll solve it with threads!". has Now problems. two he

–Davidlohr Bueso (@davidlohr)

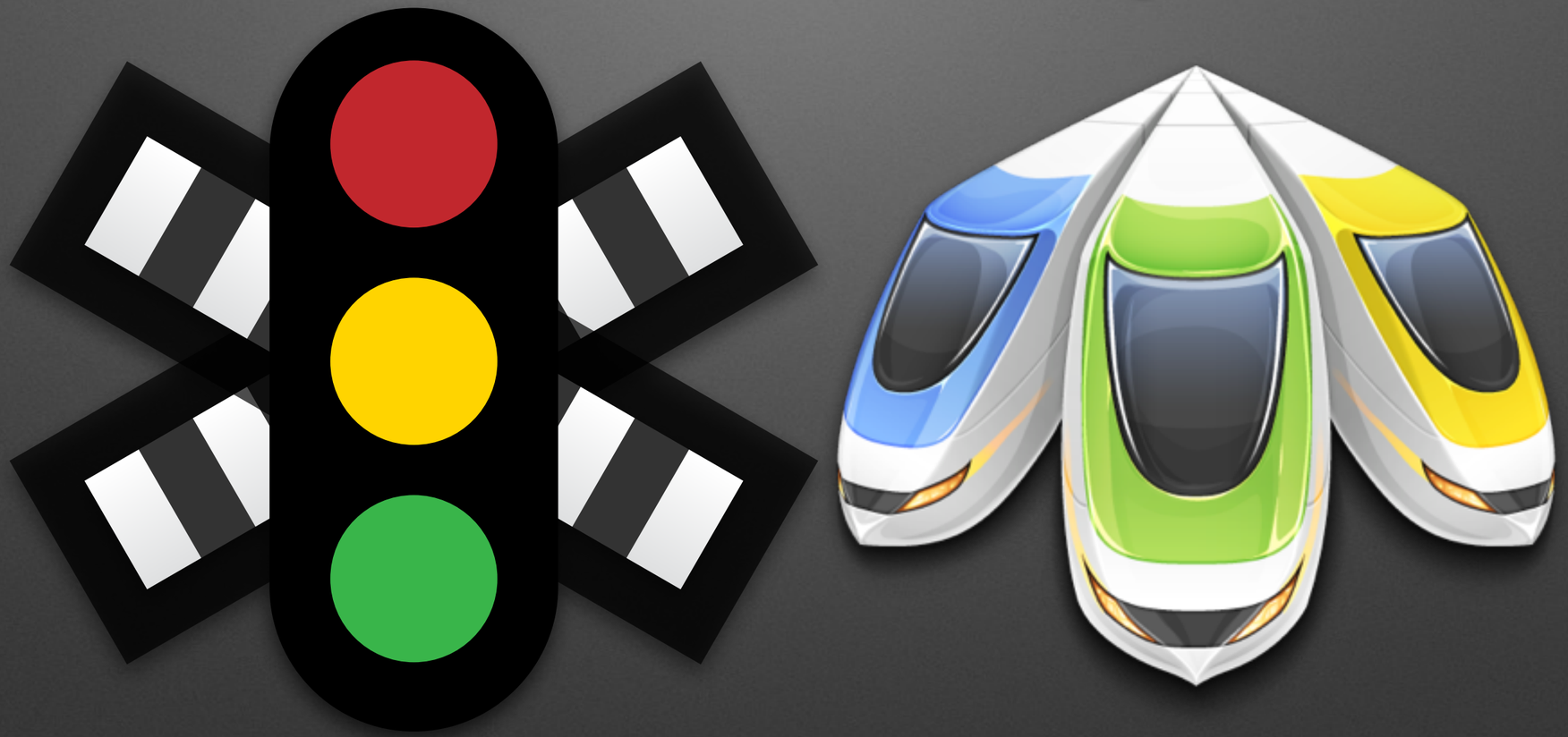
https://twitter.com/davidlohr/status/288786300067270656

Multithreaded programming - theory and practice.

-Alexander Ilyushin (@chronum)

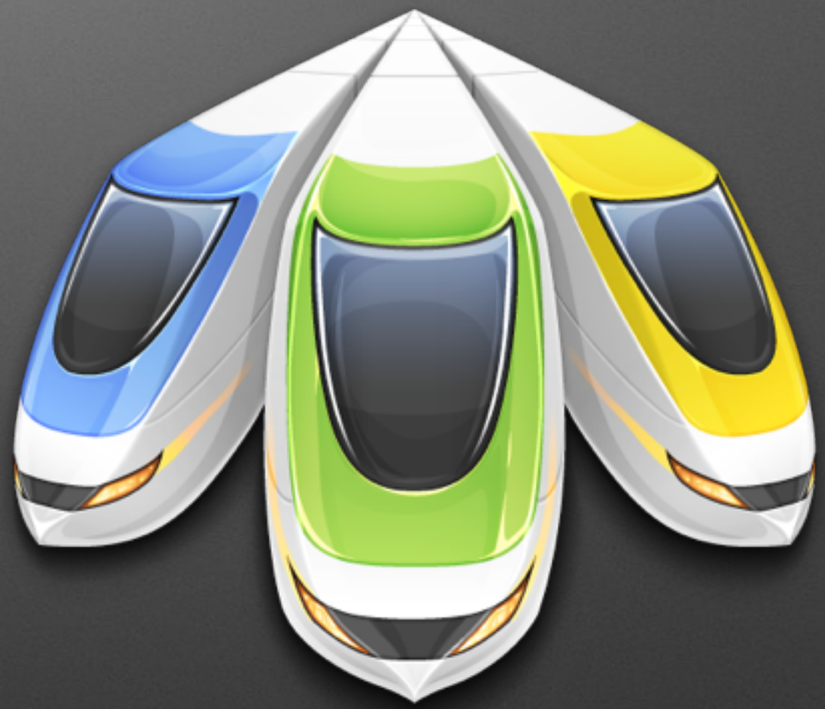
https://twitter.com/chronum/status/540437976103550976/

Grand Central Dispatch



Grand Central Dispatch

- Introduced in OS X in 2009
  - 10.6 - Snow Leopard
  - "Featureless" release
- Introduced in iOS in 2010
  - iOS 4.0 and above
  - A5 or higher for multicore



GCD High Level Features

- Multiple tasks at once
- Some guarantee about execution order
- Easier to reason about (queue model)
- Better debugging with Xcode
- Less overhead than NSOperation
- Pretty easy to use!

```
func userButtonPress() {
  longRunningTask() // Will freeze UI!
  updateDisplay()
}
```

```
func userButtonPress() {
  dispatch_async(dispatch_get_global_queue(QOS_CLASS_DEFAULT, 0)) {
    longRunningTask()
    dispatch_async(dispatch_get_main_queue()) {
      updateDisplay()
    }
  }
}
```

Responsiveness Example: Kitten Loader

“Write your application as usual, but if there's any part of its operation that can reasonably be expected to take more than a few seconds to complete, then for the love of Zarzycki, get it off the main thread!”

–John Siracusa
Ars Technica review of OS 10.6

http://arstechnica.com/apple/2009/08/mac-os-x-10-6/13/

GCD Building Blocks

- Dispatch queues
- General dispatch (sync/async)
- Dispatch apply
- Groups
- Barriers
- Semaphores

Dispatch Queues

- Lists of tasks to be executed
- Tasks in a given queue will begin in the same order they were added (FIFO)
- Tasks won't necessarily finish execution in the same order
- Two types of queues
  - Serial
  - Concurrent

Serial Queues

- One task at a time
- New tasks start only when a previous task on the same queue ends
- Apple provides one serial queue (main)
  - The default queue for (almost) all of your code
  - UI updates must happen here!

Concurrent Queues

- Multiple tasks at once
- New tasks start whenever the system thinks they should start
- Apple provides several queues
  - Quality of Service (QoS)
  - Priority

Creating Queues

```
func dispatch_queue_create(
                        label: String,
                    attr: dispatch_queue_attr_t!
                              ) -> dispatch_queue_t!

dispatch_queue_attr_t: DISPATCH_QUEUE_SERIAL (nil)
                       DISPATCH_QUEUE_CONCURRENT

let counterQueue =
     dispatch_queue_create("com.mtackes.counter", nil)

dispatch_release(counterQueue) // Not necessary! (ARC)
```

Getting Queues

```
func dispatch_get_global_queue(
                    identifier: qos_class_t / Int,
                         flags: UInt (0)
                                ) -> dispatch_queue_t

func dispatch_get_main_queue() -> dispatch_queue_t!

// Deprecated
func dispatch_get_current_queue() -> dispatch_queue_t!
```

Getting Queues

```
// OS X 10.10, iOS 8.0
QOS_CLASS_USER_INTERACTIVE
QOS_CLASS_USER_INITIATED
QOS_CLASS_DEFAULT      // System
QOS_CLASS_UTILITY
QOS_CLASS_BACKGROUND
QOS_CLASS_UNSPECIFIED // System

// Older versions
DISPATCH_QUEUE_PRIORITY_HIGH        // USER_INITIATED
DISPATCH_QUEUE_PRIORITY_DEFAULT     // DEFAULT
DISPATCH_QUEUE_PRIORITY_LOW         // UTILITY
DISPATCH_QUEUE_PRIORITY_BACKGROUND // BACKGROUND
```

Getting Queues

```
let utilityQueue =
        dispatch_get_global_queue(QOS_CLASS_UTILITY, 0)

let lowPriorityQueue =
        dispatch_get_global_queue(
                            DISPATCH_QUEUE_PRIORITY_LOW, 0)

let mainQueue = dispatch_get_main_queue()
```

Basic Dispatch Functions

- Send a task (closure or block) to a specified queue to be completed
- `dispatch_sync` - waits for the dispatched task to complete before returning
- `dispatch_async` - returns immediately and lets the task complete sometime later

Basic Dispatch Functions

- `dispatch_sync`
  - Can create deadlocks! Take care when using on serial queues
  - Good for getting values back without an extra dispatch or very quick tasks
- `dispatch_async`
  - No guarantee when it will complete
  - Great for long running processes or many tasks sent to concurrent queues

Dispatching Tasks

```
func dispatch_sync(queue: dispatch_queue_t!,
                   block: dispatch_block_t!)

func dispatch_async(queue: dispatch_queue_t!,
                    block: dispatch_block_t!)

typealias dispatch_block_t = () -> Void
```

Dispatching Tasks

```
var thing = 0
dispatch_sync(aQueue) {
  thing = getSomeValue()
}
println(thing) // Will display returned value


var thing = 0
dispatch_async(aQueue) {
  thing = getSomeValue()
}
println(thing) // Will still be 0
```

Dispatching Tasks

```
// Do not do this from the main queue!
// It will cause deadlock!

dispatch_sync(dispatch_get_main_queue()) {
  // Anything here
}
```

Demo App

- Dollar Words
  - Take a list of words
  - A = 1, B = 2 … Z = 26
  - Words whose letters add up to 100 are "dollar" words
- Add some concurrency!

Dollar Words

Basic Dispatch

Dispatch Barriers

- Sync/Async
  - `dispatch_barrier_sync`
  - `dispatch_barrier_async`
- Only work on custom concurrent queues
- Stop any other tasks from executing while it is running
- Other tasks resume after barriers finish

Dispatch After

- `dispatch_after`
  - Time
  - Queue
  - Closure
- Submits the closure to the queue after the specified time
- No guarantee that it will run at that time; it will only be *submitted* to run

Dispatch Apply

- An automatically dispatched 'for' loop
- Dispatches a closure to a queue a specified number of times
- Passes the current iteration number as a parameter to the closure
- No easy way to find when all queued blocks have completed

Dispatch Apply

```
func dispatch_apply(iterations: UInt,
                    queue: dispatch_queue_t!,
                    block: ((UInt) -> Void)!)

dispatch_apply(10, aQueue) { (count) in
  // count will be 0..<10
}

dispatch_apply(UInt(array.count), aQueue) {
  // do something with array[Int($0)]
}
```

Dollar Words

Dispatch Apply

Dispatch Groups

- Used to group tasks together
- Let you run a task once the whole group has completed
- Really a counter of outstanding tasks
  - Dispatches increase the counter
  - Completions decrease the counter
  - The group is "done" when the counter hits zero

Creating Groups

```
func dispatch_group_create() -> dispatch_group_t!
typealias dispatch_group_t = NSObject // Plus magic

let group = dispatch_group_create()
```

Automatic Group Counting

```
func dispatch_group_async(group: dispatch_group_t!,
                          queue: dispatch_queue_t!,
                          block: dispatch_block_t!)

dispatch_group_async(myGroup, aQueue) {
  // async code
}
```

Manual Group Counting

```
// Increments the group counter
func dispatch_group_enter(group: dispatch_group_t!)

// Decrements the group counter
func dispatch_group_leave(group: dispatch_group_t!)

// Make sure to balance these calls!
```

Group Completion Actions

- Two versions
  - `dispatch_group_wait` (sync)
    - Pauses the thread
    - Lets you specify a timeout
  - `dispatch_group_notify` (async)
    - No timeout
    - Specify a closure to run

Dispatch Group Wait

```
// Returns 0 for success, 1 for failure
func dispatch_group_wait(group: dispatch_group_t!,
                         timeout: dispatch_time_t) -> Int
```

```
if dispatch_group_wait(group, dispatch_time(…)) == 0 {
  // Group completed before timeout
} else {
  // Timeout completed before group
}

// Special times
DISPATCH_TIME_FOREVER
DISPATCH_TIME_NOW
```

Dispatch Group Notify

```
func dispatch_group_notify(group: dispatch_group_t!,
                           queue: dispatch_queue_t!,
                           block: dispatch_block_t!)

dispatch_group_notify(myGroup, queue) {
  // Completion code
}
```

Dispatch Groups

- A single `dispatch_notify` will only execute once
- There can be multiple notifies/waits depending on a single group
- If a group is empty when a notify or wait is made, they execute immediately
- Extremely powerful
- The easiest way to handle the completion of a number of tasks

Dollar Words

Dispatch Groups

More GCD

- Semaphores
- Dispatch sources
- Queue-specific context data
- Dispatch time
- In-depth concurrent queue creation

References

- Ray Wenderlich - GCD Tutorials
  http://www.raywenderlich.com/79149/
  http://www.raywenderlich.com/79150/
- Apple Docs
  https://developer.apple.com/library/ios/documentation/Performance/Reference/GCD_libdispatch_Ref/
- Wikipedia - general concurrent computing

Links

- @MichaelTackes on Twitter
- github.com/mtackes
- tackes.net - still working on it

- KittenLoader: github.com/mtackes/KittenLoader/
- DollarWords: github.com/mtackes/DollarWords/