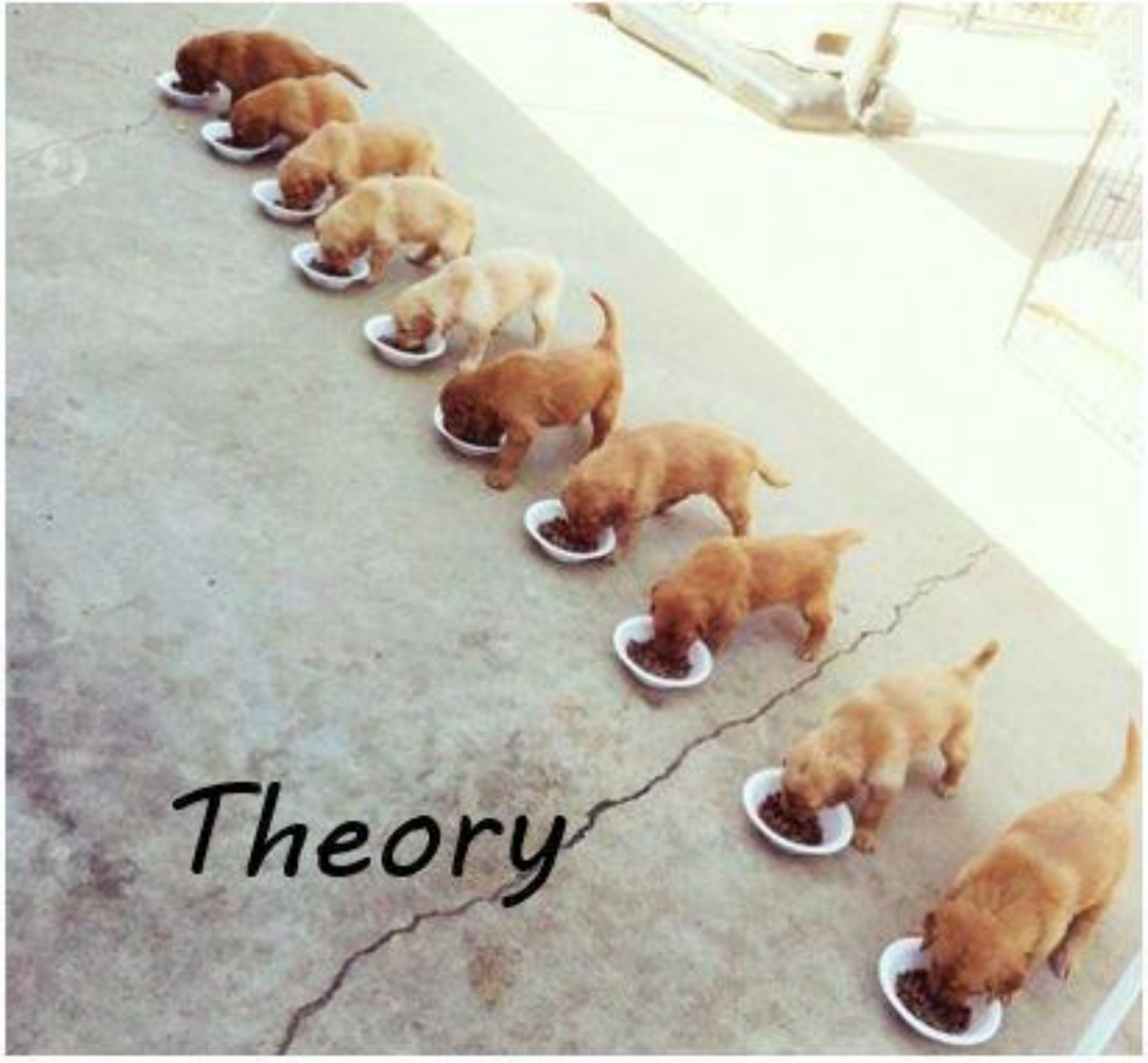
GCD

(**G**rand **C**entral **D**ispatch)

Multithreaded programming



Filas

- FIFO
- Blocos/closures
- Threads são detalhes de implementação
- 3 tipos
  - Main
  - Concorrente
  - Serial

Main Queue

- Thread principal
- UI sempre deve ser feita nela
- Se estiver sobrecarregada, vai ser visível para o usuário

```
let queue = dispatch_get_main_queue()

dispatch_async(queue) {
    // work
}
```

Concurrent Queues

- Tarefas são iniciadas na ordem FIFO
- Mas podem ser executadas ao mesmo tempo
- Não há garantia de ordem de término
- Criadas pelo sistema ou por você

```
// DISPATCH_QUEUE_PRIORITY_BACKGROUND
// DISPATCH_QUEUE_PRIORITY_DEFAULT
// DISPATCH_QUEUE_PRIORITY_HIGH
// DISPATCH_QUEUE_PRIORITY_LOW

let priority = DISPATCH_QUEUE_PRIORITY_HIGH
let queue = dispatch_get_global_queue(priority, 0)

let label = "com.movile.next.exemplo"
let createdQueue = dispatch_queue_create(label,
                                         DISPATCH_QUEUE_CONCURRENT)
```

Serial Queues

- FIFO
- Uma tarefa por vez
- Criada sempre por você

```
let label = "com.movile.next.exemplo.serial"
let createdQueue = dispatch_queue_create(label,
                                         DISPATCH_QUEUE_SERIAL)
```

Como iniciar tarefas?

```
let queue = dispatch_get_main_queue()

dispatch_async(queue) {
    // work
}

// DEADLOCK se a fila atual == queue
dispatch_sync(queue) {
    // work
}
```

```
let delayTime = dispatch_time(DISPATCH_TIME_NOW,
                              Int64(1 * Double(NSEC_PER_SEC)))

dispatch_after(delayTime, queue)) {
    // work after 1s
}
```

dispatch_once

```
+ (instancetype)sharedInstance {
    static dispatch_once_t once;
    static id sharedInstance;
    dispatch_once(&once, ^{
        sharedInstance = [[self alloc] init];
    });
    return sharedInstance;
}

// Em Swift, é só usar `static let`
static let sharedInstance = MyClass()
```

Semáforos

```
// ESSE EXEMPLO TRAVA A MAIN THREAD. NUNCA FAÇAM ISSO!!!

let semaphore = dispatch_semaphore_create(0)

let client = TraktHTTPClient()
client.getPopularShows { _ in
    dispatch_semaphore_signal(semaphore)
}

dispatch_semaphore_wait(semaphore, DISPATCH_TIME_FOREVER)
```

Dispatch Groups

- Avisa quando várias tarefas terminam
- Sincronizar chamadas de API, por exemplo

http://commandshift.co.uk/blog/2014/03/19/using-dispatch-groups-to-wait-for-multiple-web-services/

```
let group = dispatch_group_create()
let client = TraktHTTPClient()

dispatch_group_enter(group)
client.getPopularShows { _ in
    dispatch_group_leave(group)
}

dispatch_group_enter(group)
client.getRecentShows { _ in
    dispatch_group_leave(group)
}

// existe também o dispatch_group_wait
dispatch_group_notify(group, dispatch_get_main_queue()) {
    // work...
}
```