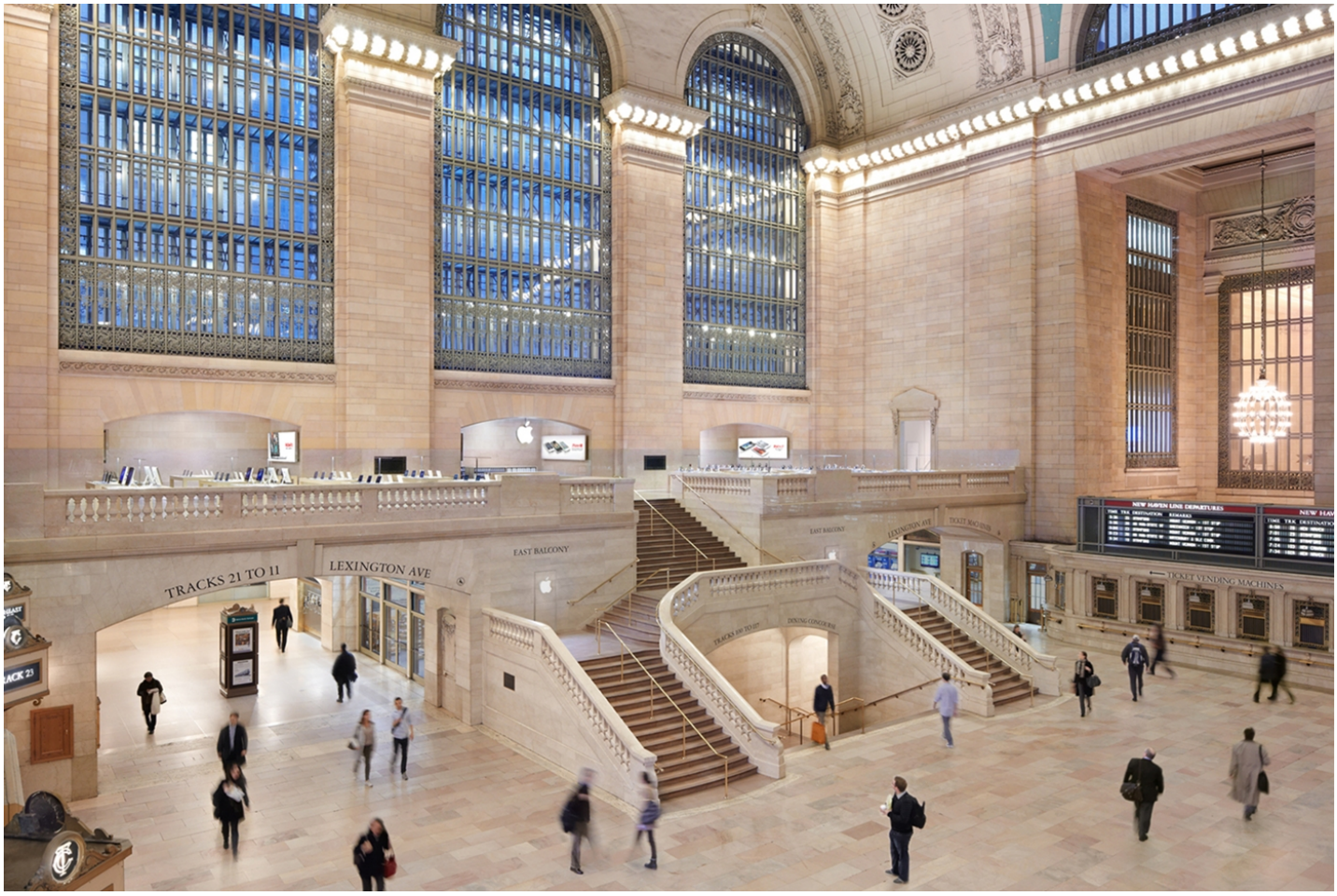
GCD - Grand Central Dispatch - Introduction



Agenda

- Introduction
- Dispatch Queue [Async, Sync]
- Dispatch Group
- QoS
- (!Dispatch Semaphores, !Dispatch Time, !Dispatch Sources)

Who

- Dispatch Framework
- Apple Inc, California
- FreeBSD libdispatch
- OpenSource clones for Windows under development [native Safari, iTunes Windows integration]

Why concurrency

- reduce load on main thread -> UI does not hang/block => getting work off your main thread
- work in asyncronous speed steps is possible
- different task priorities are possible
- more work, faster, more efficient
- timing

Where GCD Availability

- all Apple platforms, iOS [sinde 4], tvOS, macOS [since Snow Leopard], watchOS

What is GCD

- Framework [BSD subsystem, Core Foundation, Cocoa API]
- "Execute code concurrently on multicore hardware by submitting work [items] to dispatch queues managed by the system" ~https://developer.apple.com/reference/dispatch
- abstraction layer over threads [threadpools]
- what are the problems of other threading concepts?

How / GCD Fabric

- Dispatch Queue
    - submit items to the cue [in Swift these are closures]
    - GCD brings up a worker thread for this cue on its own and service that work for you
    - GCD suspends thread for you, when work is done
- Run Loop
    - are run by your own thread
- Main Loop [speciality]
    - has its own Main Run Loop
    - has its own Main Queue
- Quality of Service QoS

Dispatch Queues

- *serial*: work items executed one at a time

- *concurrent*: work items are dequeued in order, but run all at once and can finish in any order
- execute work items in FIFO order

Asynchronous Execution

- stack up itmes on the queue
- GCD will execute work step by step on own thread but work items are processed concurrently
- GCD will then suspend the thread on its own

Synchronous Execution

- stack up items on the queue from your own thread
- the program waits until execution finishes before the method call returns
- GCD will execute work and wait/block until the the read that submitted the item is ready before continuing with the work on the queue; controll is thereby passed over to the calling thread; controll will be given back to the worker thread when the work done
- GCD then reclaims the thread that it was working on

Controlling Concurrency

- Thread pool of GCD will limit concurrency (default size dependent on system and resources available)
- Worker threads that *block* can cause more worker threads to be spawed, therefore it is important to choose the right number of dispatch queues [thread explosion]
- one dispatch queues per subsystem of your application is recommended [i.e. networking, data transformation, database]

Concurrency Strategies

- Chaining vs. Grouping

Grouping Work together

- Dispatch Group

Synchronizing Between subsystems

- use subsystem serial queues for mutual exclusion
- use sync queue access to safely access properties from subsystems
- caution: "lock ordering" introduced between subsystems -> dead lock problem

Quality of Service QoS

- Prioritizing Work and Specifying Quality of Service
- provides explicit classification of work [i.e. userInteractive, .userInitiated, .utility, .background]
- indicates the developer intent
- affects execution properties of the work
- use `.async` to submit work with a specific QoS class
- GCD helps resolve priority inversions