iOS 프로그래밍

Async Network Programming

코드스쿼드

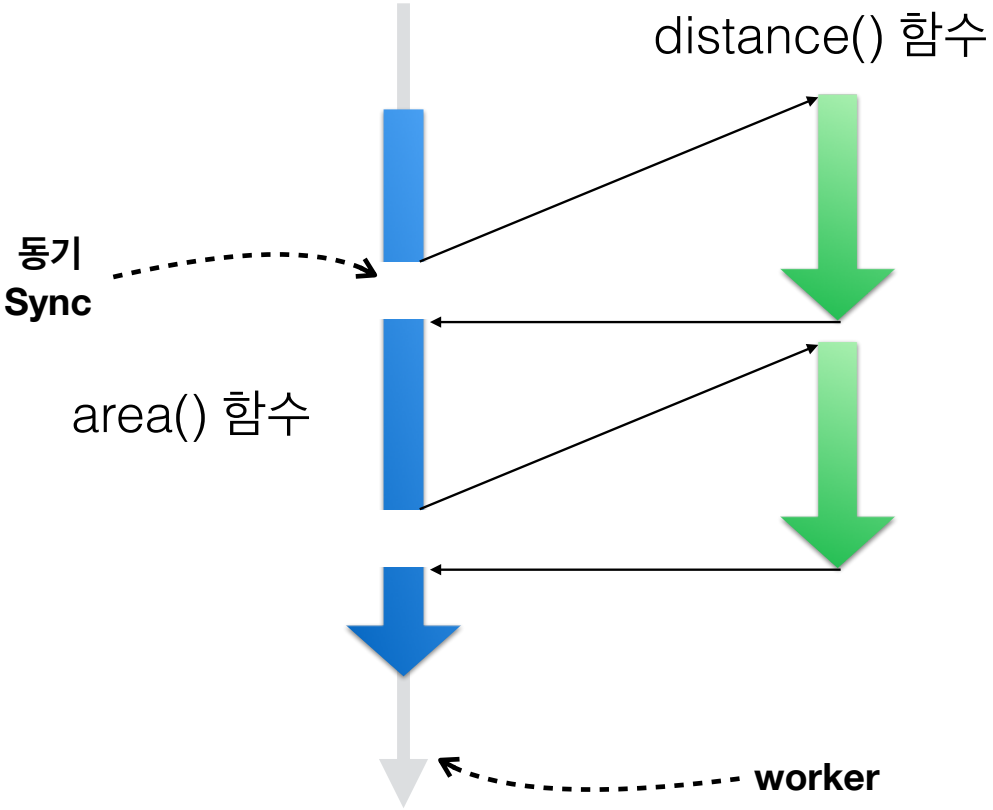
학습 목표

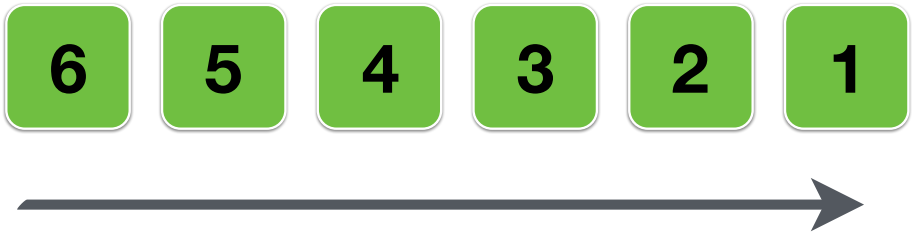
- Concurrent Programming
- HTTP Request 와 Response
- URLConnection 과 URLSession
- Reachability

Concurrent Programming

Parellel vs Concurrent

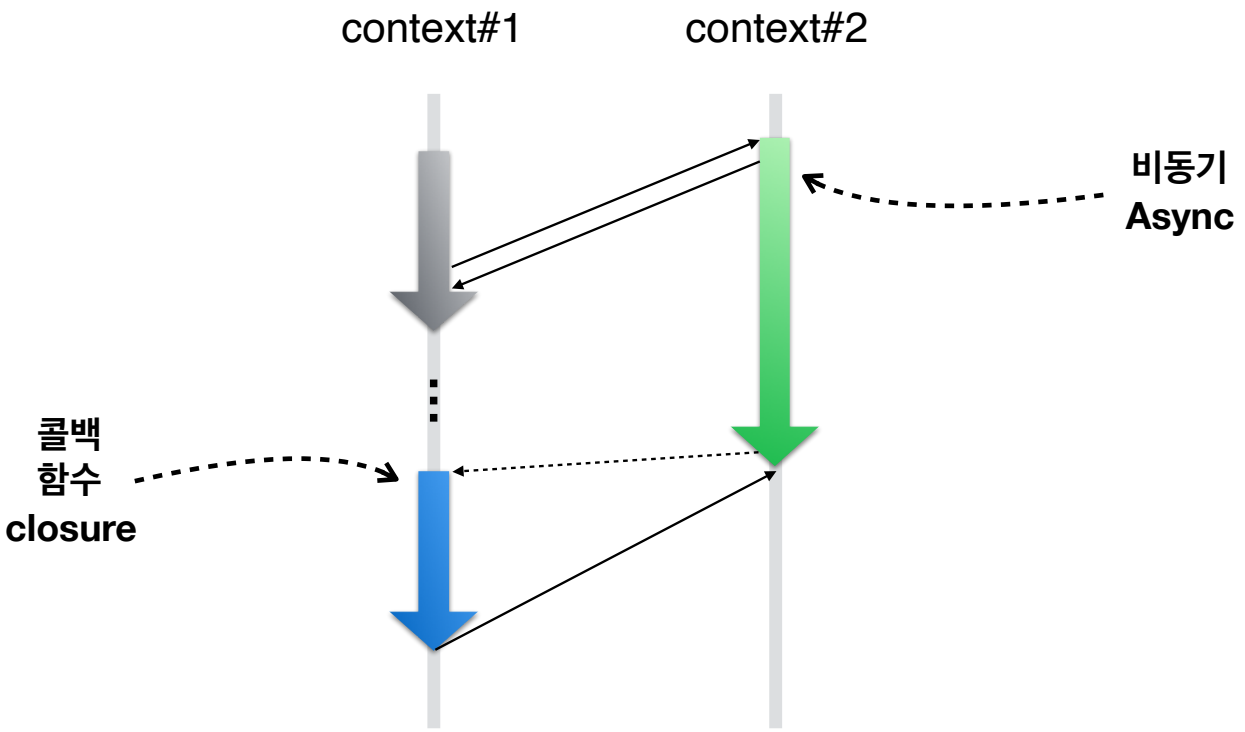
Routine - subroutine



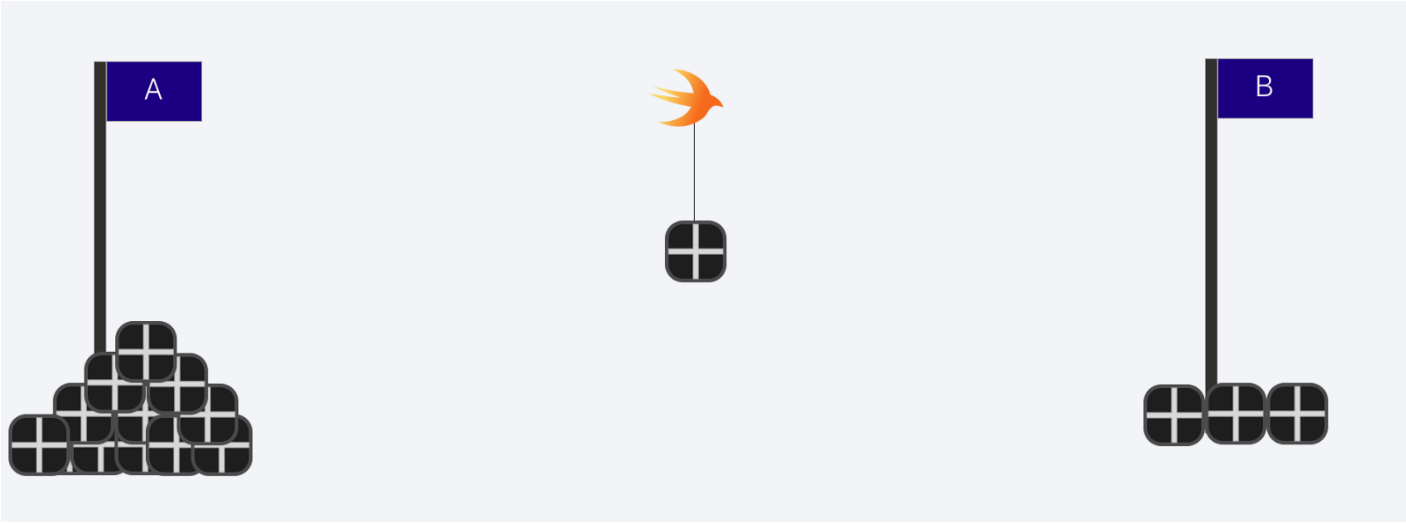


https://medium.com/flawless-app-stories/basics-of-parallel-programming-with-swift-93fee8425287

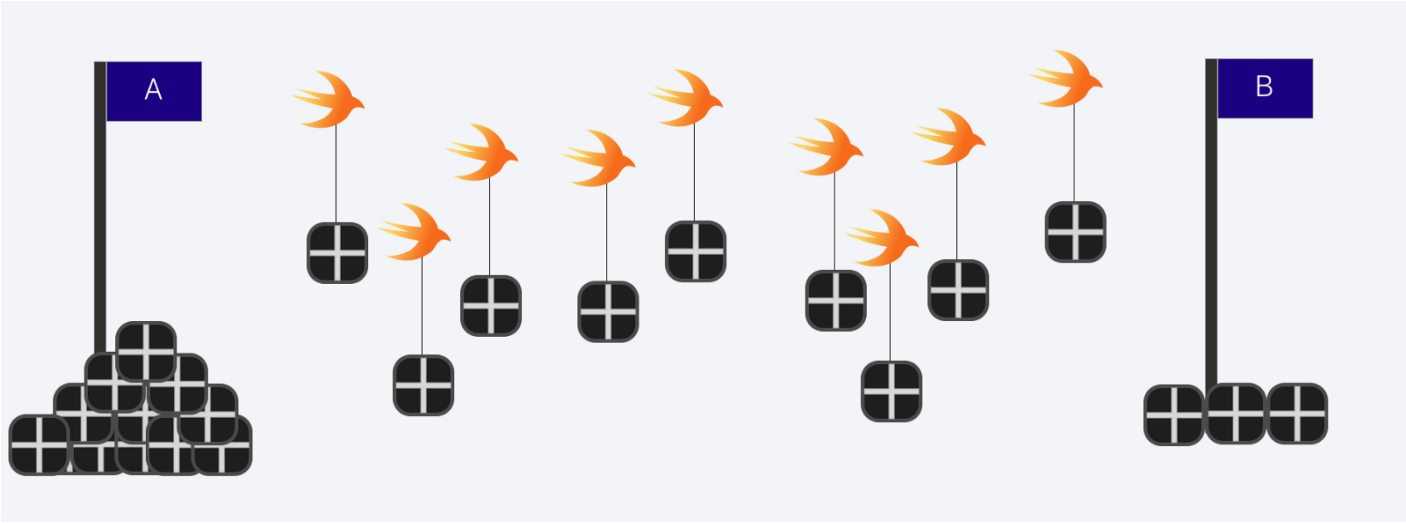
비동기 Asynchronous



Parallel vs Concurrent

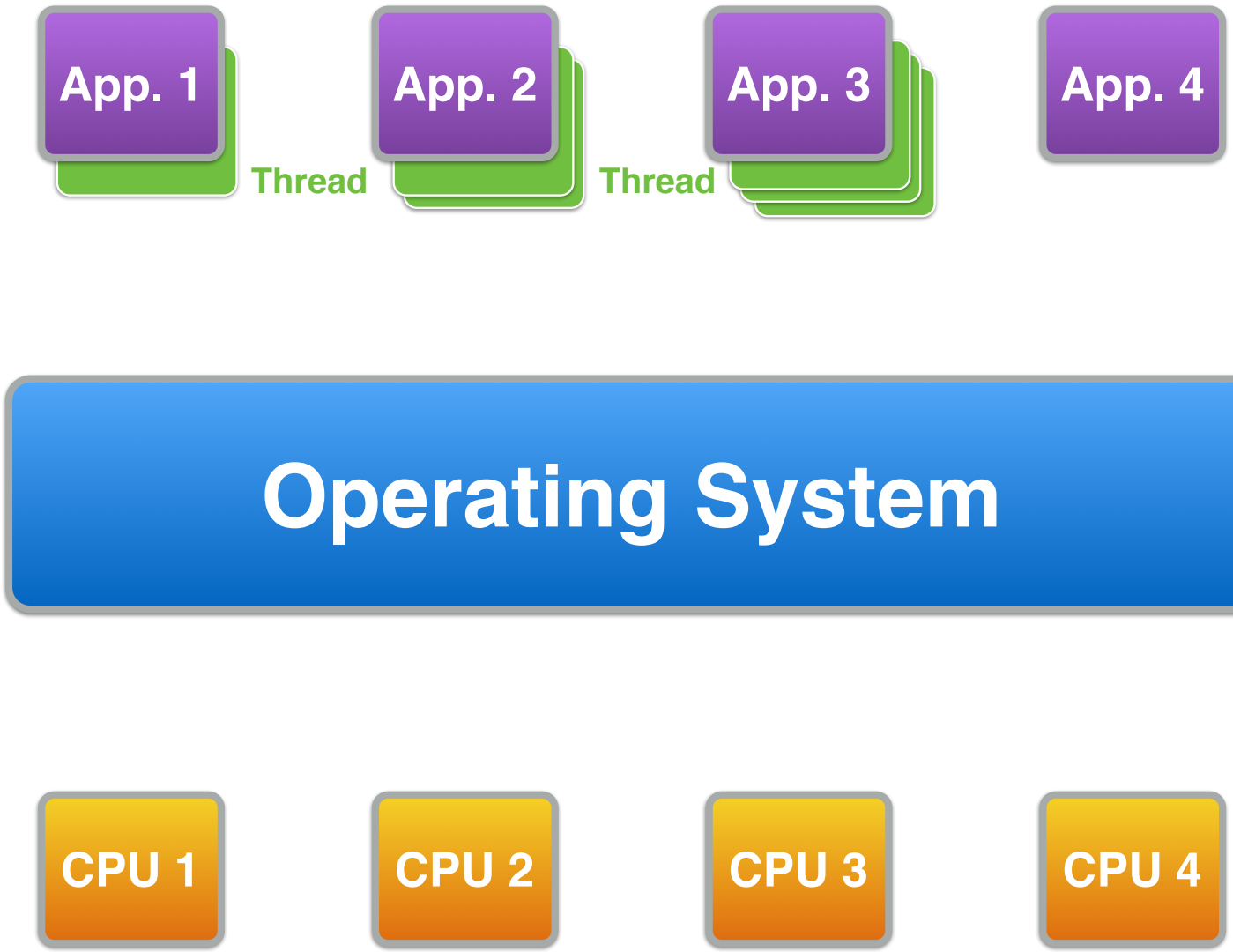


이미지 소스 https://medium.com/flawless-app-stories/basics-of-parallel-programming-with-swift-93fee8425287



이미지 소스 https://medium.com/flawless-app-stories/basics-of-parallel-programming-with-swift-93fee8425287

Process vs. Thread


App. 1
Thread
App. 2
Thread
App. 3
App. 4
Operating System
CPU 1
CPU 2
CPU 3
CPU 4

NSThread class

Using these methods

+ (void)detachNewThreadSelector: (SEL) aSelector toTarget: (id) aTarget withObject: (id) anArgument;

- (void) aSelector: (id) anArgument;

```
- (void)doTimeConsumingOperation:(id)operation {
    id t = [[NSThread alloc] initWithTarget:self
                selector:@selector(runHelperThread:)
                object:operation];
    [t run];
    [t autorelease];
}
- (void)runHelperThread:(id)operation {
    NSAutoreleasePool *p = [NSAutoreleasePool new];
    [operation doOperation];
    [p release];
}
```

Perform Selector

NSObject methods

- performSelectorOnMainThread:withObject:waitUntilDone:
- performSelector:onThread:withObject:waitUntilDone:
- performSelector:withObject:afterDelay:
- performSelectorToBackground:withObject:

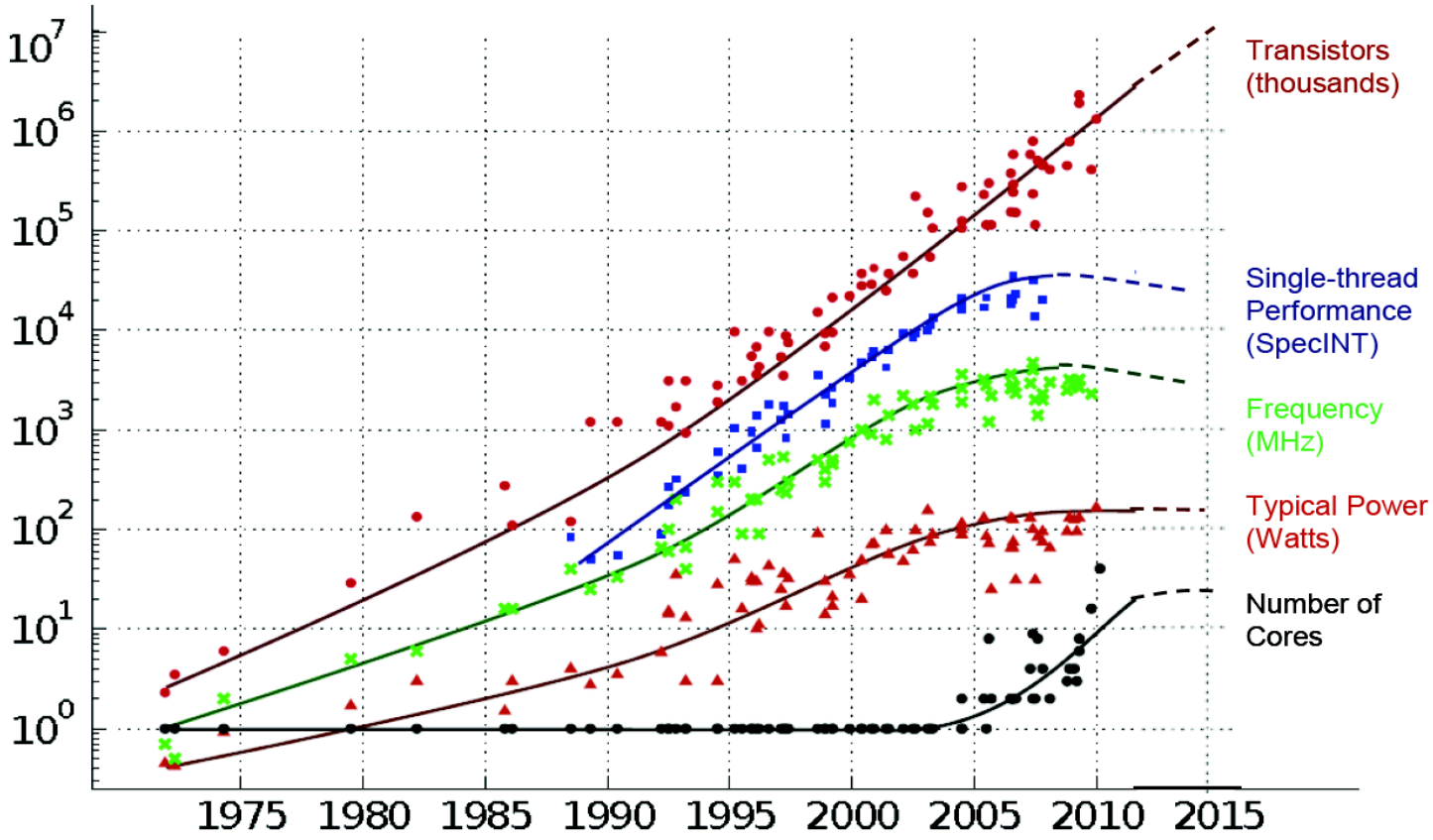
* Send messages between threads
* Wake up background threads
* Transfer data between threads

Concurrent Programming

Advent of these two

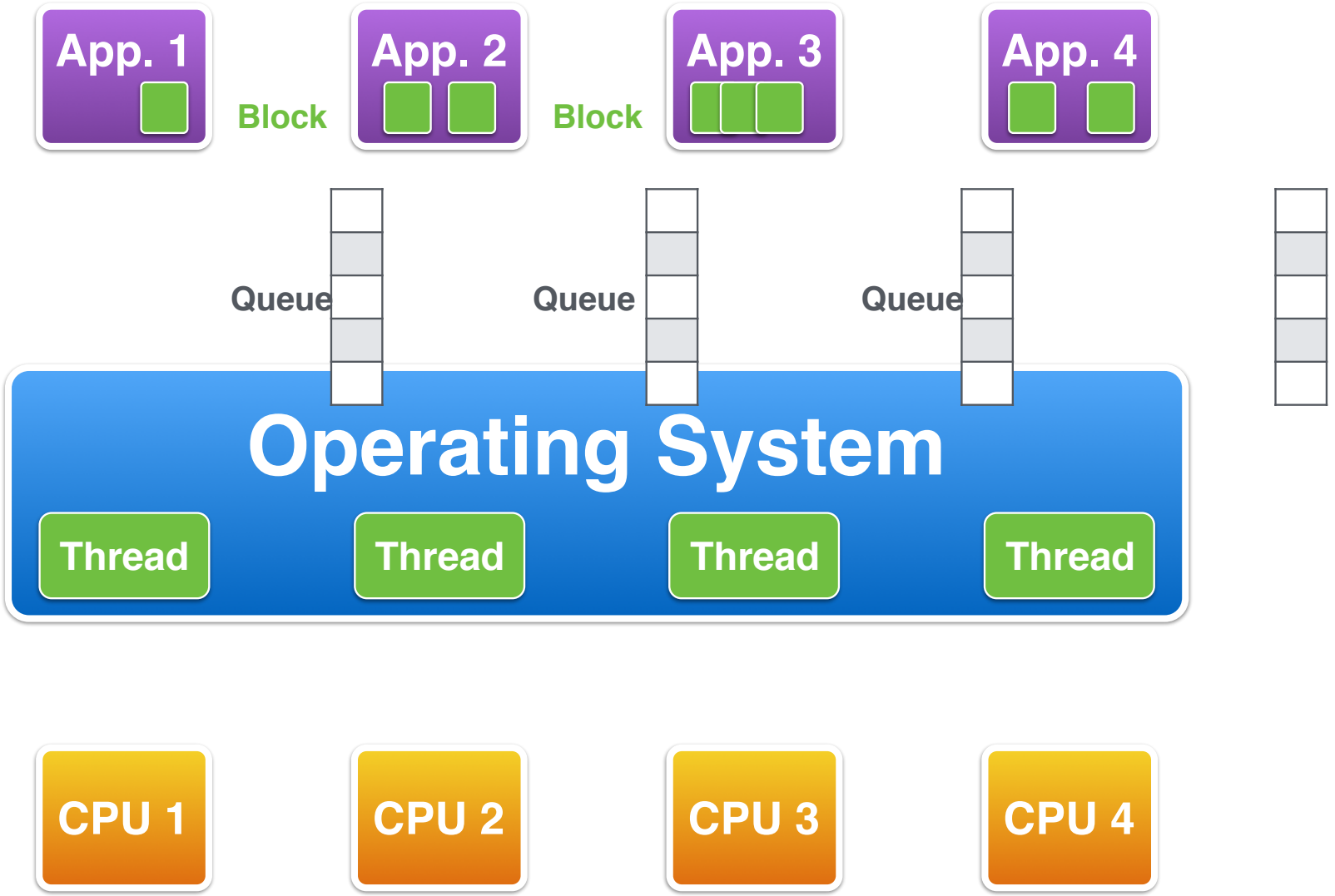
* **Benefit**
  * UI keeps responsive while processing
  * Faster operation by parallel execution
* **Cost**
  * Writing good multi-threaded code is EXTREMELY hard
  * General Problem of executing multiple tasks in a scalable way

35 YEARS OF MICROPROCESSOR TREND DATA

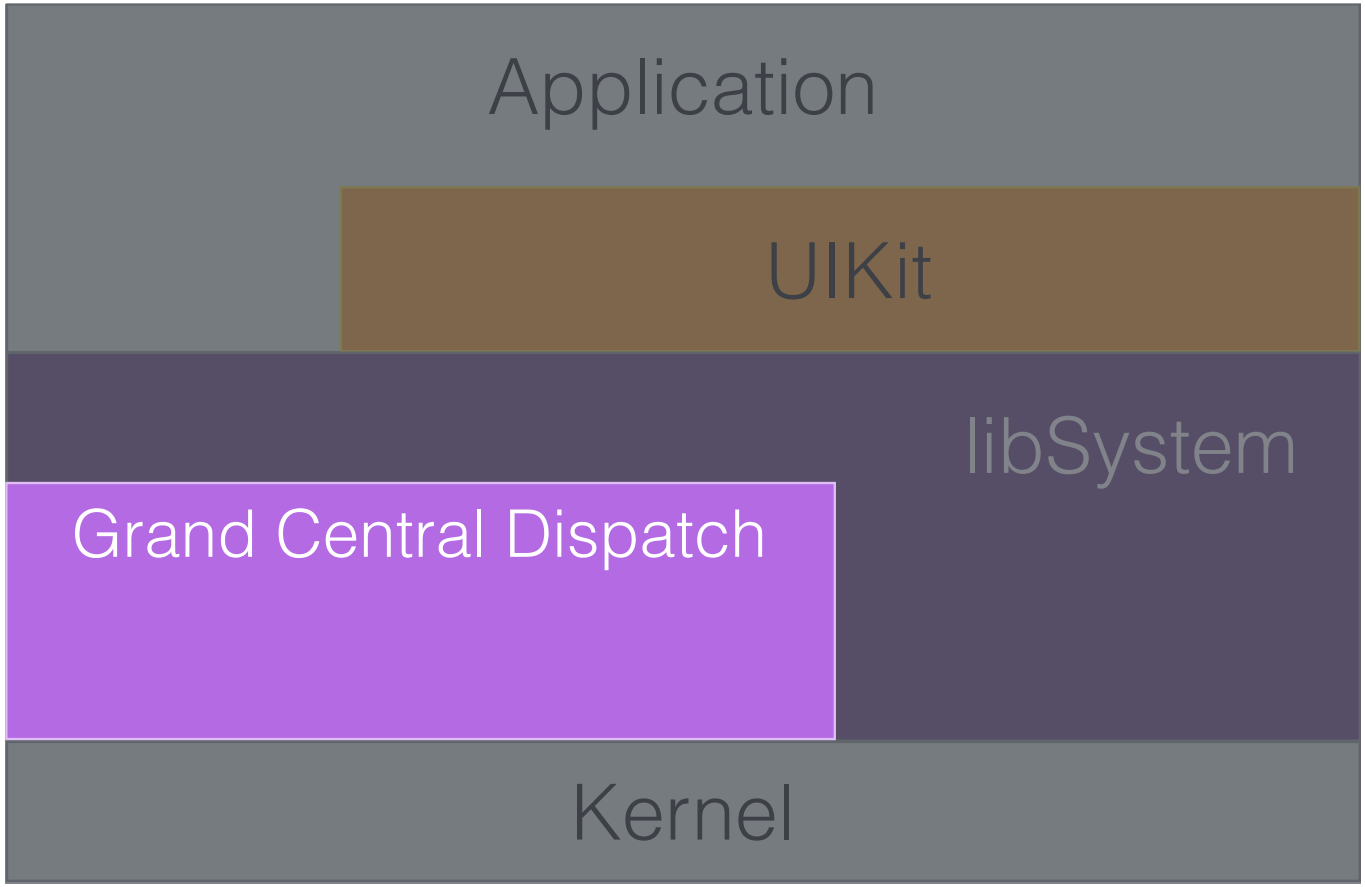


Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/


App. 1
Block
App. 2
Block
App. 3
App. 4
Queue
Queue
Queue
Operating System
Thread
Thread
Thread
Thread
CPU 1
CPU 2
CPU 3
CPU 4

Technology Stack

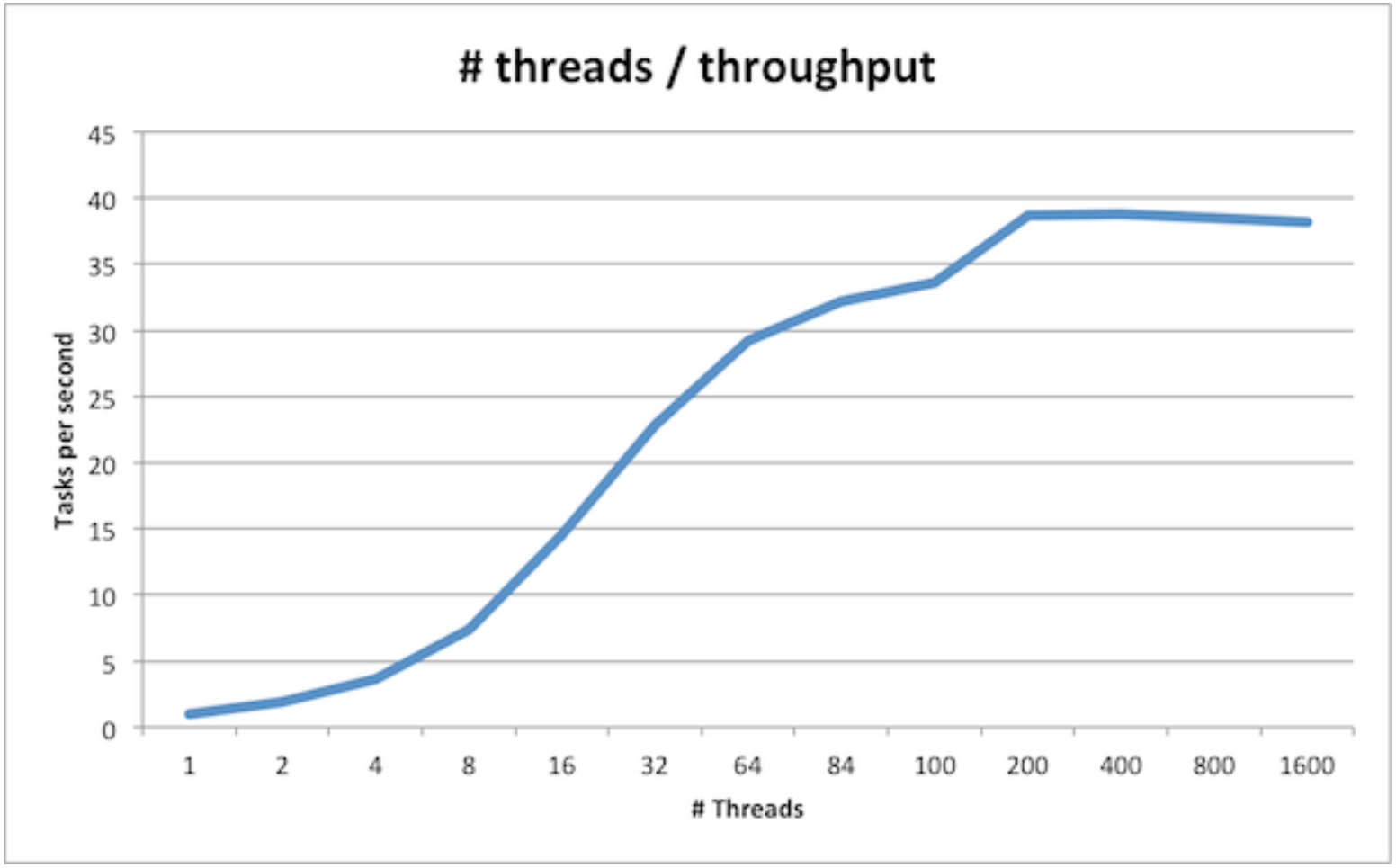


Grand Central Dispatch

GCD

Grand Central Dispatch

- OS handles managing threads and their execution
- Units of work are described as blocks
- queues are used to organize blocks based on how to be executed
- Benefits
  - Improved responsiveness
  - Dynamic scaling
  - Better processor utilisation
  - Cleaner code

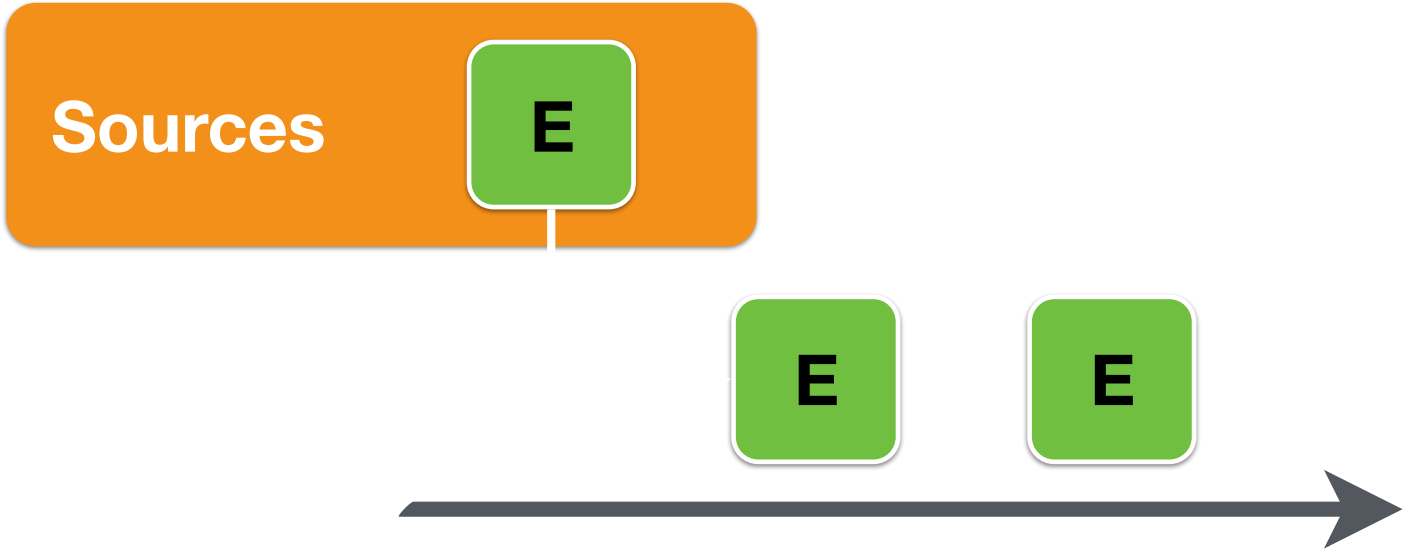

threads / throughput
Tasks per second
45
40
35
30
25
20
15
10
5
0
1
2
4
8
16
32
64
84
100
200
400
800
1600
Threads

GCD

Queues

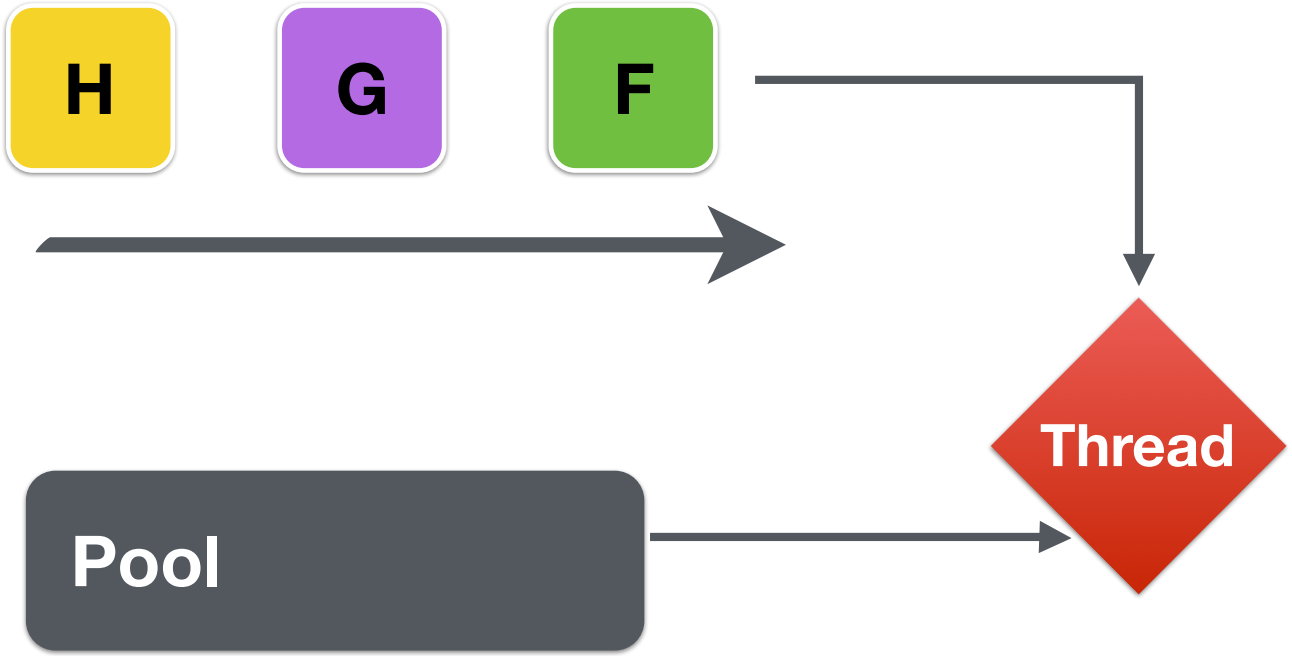
GCD

Event Sources



GCD

Thread Pool



Simplifying Your Code

GCD advantages

- **Efficiency**
  - More CPU cycles available for your code
- **Better metaphors**
  - Blocks are easy to use
  - Queues are inherently producer/consumer
- **Systemwide perspective**
  - Only the OS can balance unrelated subsystems

Simplifying Your Code with GCD

Compatibility

- Existing threading and synchronisation primitives are 100% compatible
- GCD threads are wrapped POSIX threads
  - Do not cancel, exit, kill, join, or detach GCD threads
- GCD reuses thread
  - Restore any per-thread state changed within al block

Thread

```
class SomeClass {
    var thread : Thread? = nil

    func doTimeConsumingOperation(operation : Any?) {
        thread = Thread(target: self,
                        selector: #selector(SomeClass.runHelper), object: operation)
        thread?.start()
    }

    @objc func runHelper(operation : Any?) {
        autoreleasepool { () in
            //operation.doOperation()
            print("Other thread is running...")
        }
    }
}

let some = SomeClass()
some.doTimeConsumingOperation(operation: nil)
```

Thread

```
func doTimeConsumingOperation(operation : Any?) {
    DispatchQueue.init(label: "jk.codesquad.operation").async {
        //operation.doOperation()
        print("Other thread is running...")
    }
}
```

GCD Advantages

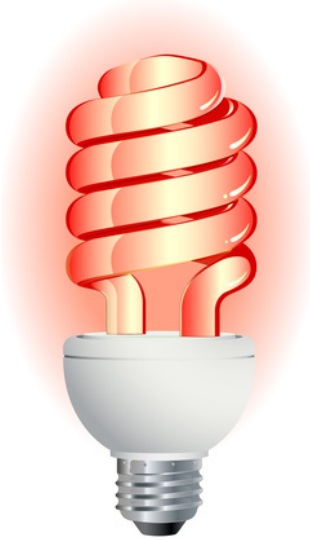
Convenient

- Less boilerplate
- No explicit thread management

GCD Advantages

Efficient

- Thread recycling
- Deferred based on availability



Locking

* Enforce mutually exclusive access to critical sections
* Serialize access to shared state between threads
* Ensure data integrity

Locking

```
var imageCache = [UIImage]()

func updateImageCache(_ image: UIImage) {
    let mutex = NSLock()
    mutex.lock()
    // Critical Section begin
    if self.imageCache.contains(image) {
        mutex.unlock()
        return
    }
    self.imageCache.append(image)
    mutex.unlock()
}
```

Locking

```
func updateImageCache(_ image: UIImage) {
    DispatchQueue.init(label: "jk.codesquad.imageCacheQueue").sync {
        // Critical Section begin
        if self.imageCache.contains(image) {
            return
        }
        self.imageCache.append(image)
    }
}
```

Locking

Deferred critical section

```
func updateImageCache(_ image: UIImage) {
    DispatchQueue.init(label: "jk.codesquad.imageCacheQueue").async {
        // Critical Section begin
        if self.imageCache.contains(image) {
            return
        }
        self.imageCache.append(image)
    }
}
```

Inter-Thread Communication

Performing selectors

```
- performSelector(onMainThread:#Selector, with: Any?, waitUntilDone: Bool)
- perform(aSelector:#Selector, on: Thread, with: Any?, waitUntilDone: Bool)
- perform(aSelector:#Selector, with: Any?, afterDelay: TimeInterval)
- performSelector(inBackground: Selector, with: Any)
```

Inter-Thread Communication

performSelector(onMainThread:with:waitUntilDone:)

//waitUntilDone: No

```
DispatchQueue.main.async {
    myObject.doSomething(foo, withData:bar)
}
```

//waitUntilDone: Yes

```
DispatchQueue.main.sync {
    myObject.doSomething(foo, withData:bar)
}
```

Inter-Thread Communication

perform(aSelector:with:afterDelay:)

```
DispatchQueue.main.asyncAfter(deadline: .now() + .milliseconds(500)) {
    myObject.doSomething(foo, withData:bar)
}
```

Inter-Thread Communication

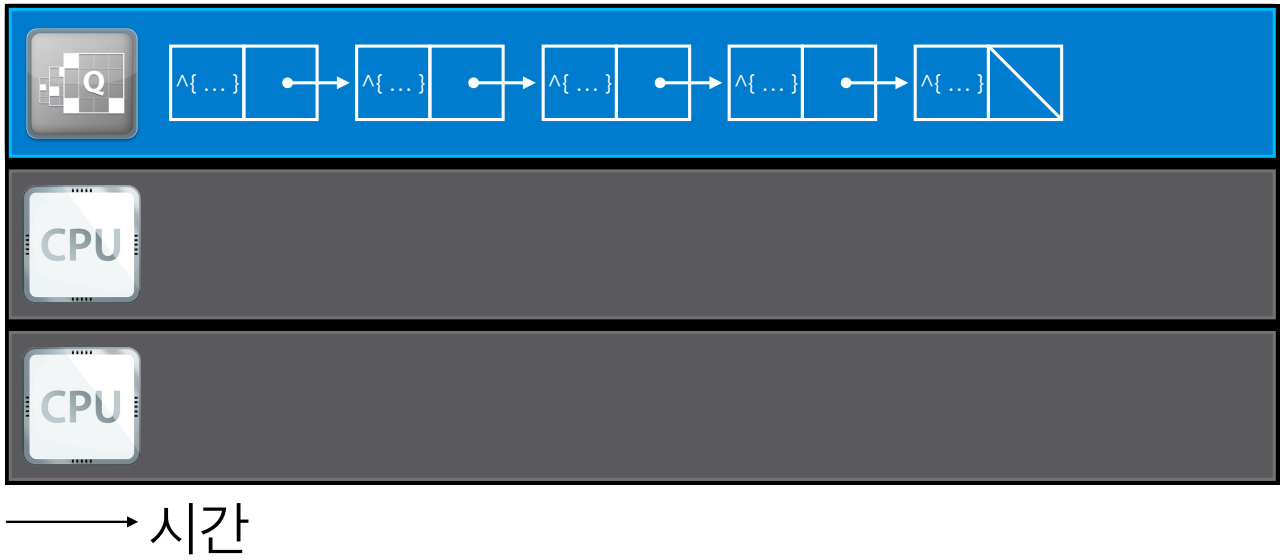
performSelector(inBackground:with:)

```
DispatchQueue.global().async {
    myObject.doSomething(foo, withData:bar)
}
```

Queues

Serialization

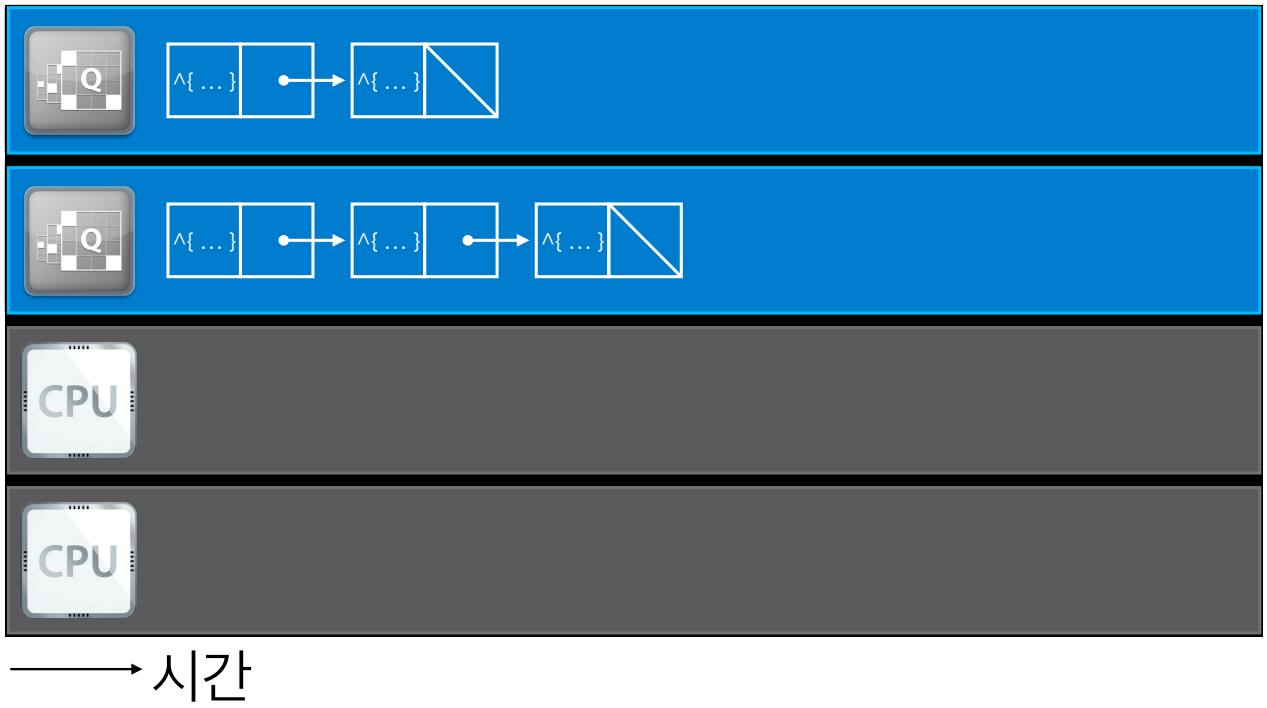
* Lightweight list of blocks
* Enqueue and dequeue are FIFO
* Serial queues execute blocks one at a time



Queues

Concurrency

* Concurrent queues execute multiple blocks at the same time
* Concurrently executed blocks may complete out of order
* Queues execute concurrently with respect to other queues



Queues

API

* Submitting blocks to queues

```
func async(group: DispatchGroup? = default,
      qos: DispatchQoS = default,
      flags: DispatchWorkItemFlags = default,
      execute work: @escaping @convention(block) () -> Swift.Void)

func sync(execute block: () -> Swift.Void)
```

* Submitting blocks later

```
func asyncAfter(deadline: DispatchTime,
       qos: DispatchQoS = default,
       flags: DispatchWorkItemFlags = default,
       execute work: @escaping @convention(block) () -> Swift.Void)
```

* Concurrently executing one block many times

```
class func concurrentPerform(iterations: Int, execute work: (Int) -> Swift.Void)
```

Queues

API

* Suspending and resuming execution

```
func suspend()
```

```
func resume()
```

* Managing queue lifetime

```
func retain()
```

```
func release()
```

Queues

유의할 점

* 일반적인 목적의 데이터 구조가 아님
* 흐름 제어를 위한 것!
* 큐에 블록을 한 번 추가하면 반드시 실행됨
* 동기화 API를 사용할 때는 조심할 것

Queues

Type

* Global queue

```
DispatchQueue.global()
```

* Main Queue

```
DispatchQueue.main
```

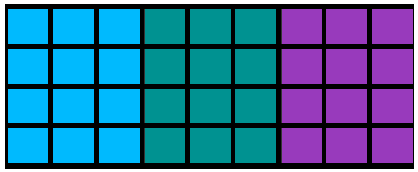
* Serial Queue

```
DispatchQueue.init(label:String)
```

GCD Design Patterns

One queue per task or subsystem

* Easy communication
  * dispatch_async()



* Queues are inherently producer/consumer
  * Blocks carry data between tasks

* Queues are lightweight and efficient
  * Automatic thread creation and recycling

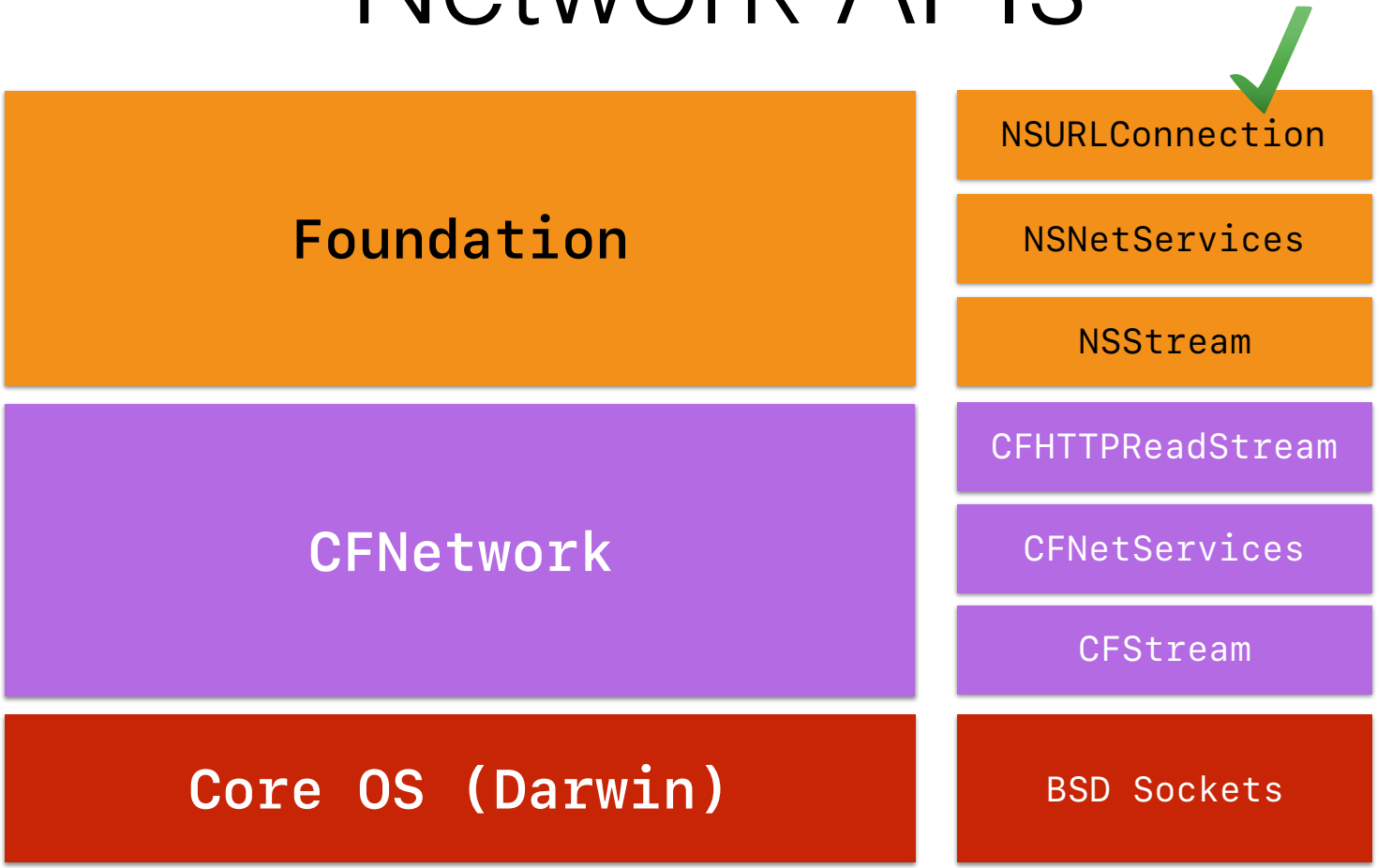
GCD Design Patterns

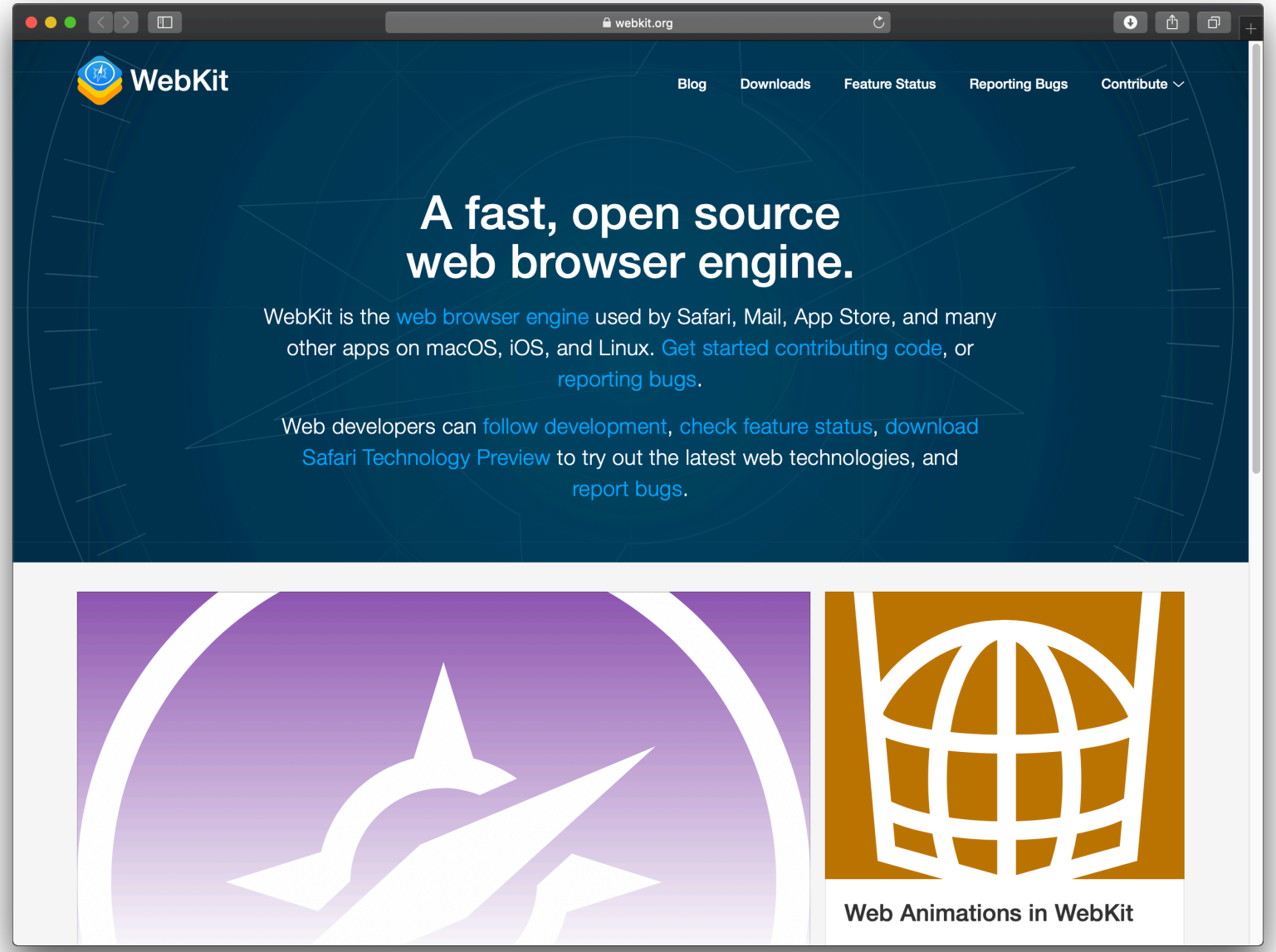
Low-level event notifications

* Similar approach to UI event-driven programming
* Don’t poll or block a thread waiting for external events
  * Waiting on a socket
  * Polling for directory changes
* Dispatch sources
  * Monitor external OS events
  * Respond on-demand

APIs for HTTP/HTTPs

Network APIs




webkit.org
WebKit
Blog
Downloads
Feature Status
Reporting Bugs
Contribute
A fast, open source web browser engine.
WebKit is the web browser engine used by Safari, Mail, App Store, and many other apps on macOS, iOS, and Linux. Get started contributing code, or reporting bugs.
Web developers can follow development, check feature status, download Safari Technology Preview to try out the latest web technologies, and report bugs.
Web Animations in WebKit

NSURLConnection

Best API for HTTP and HTTPS

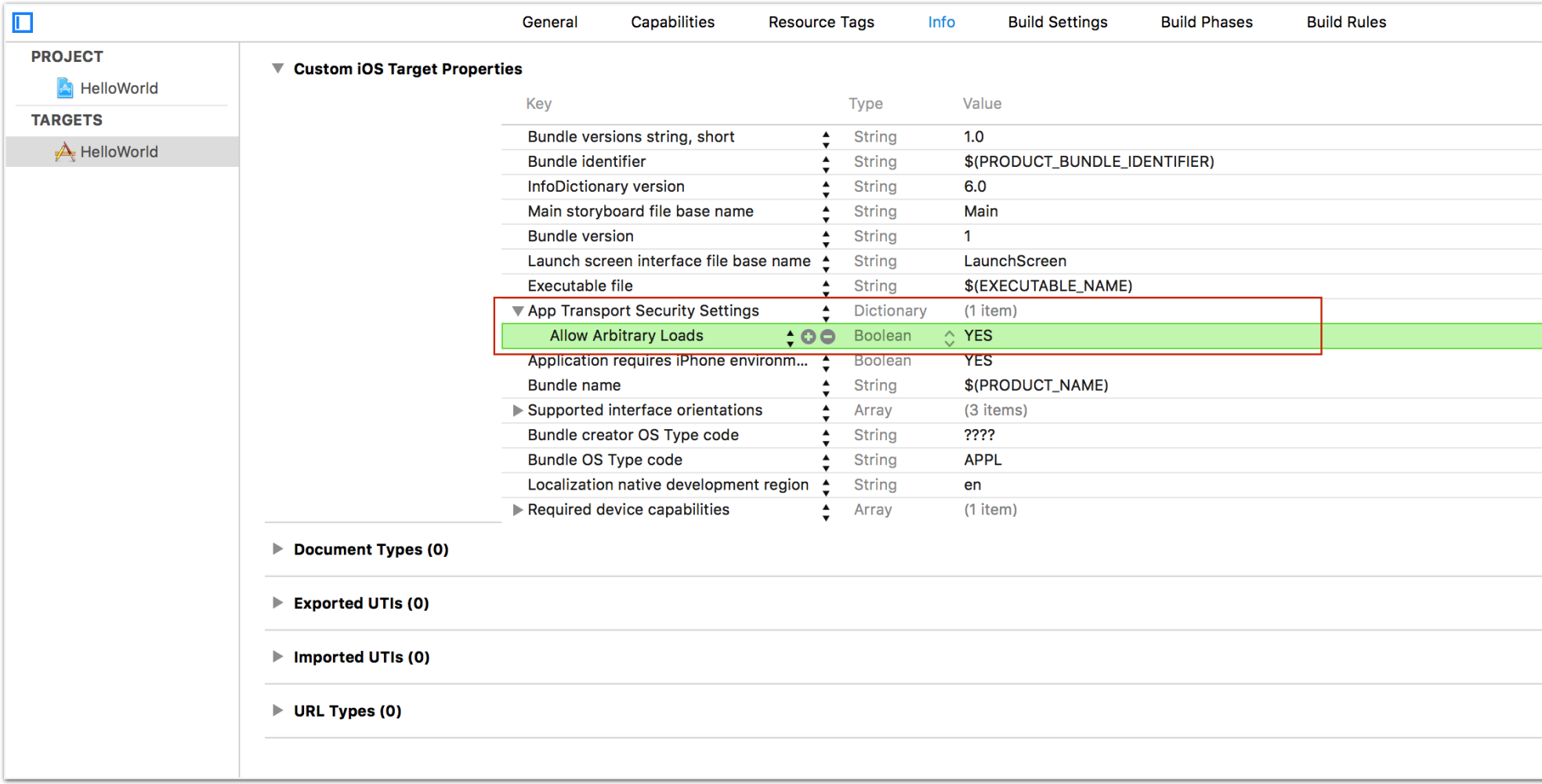
- Asynchronous event-based API
- 기능
  - Persistent connections (세션 연결 유지)
  - Pipelining (이어받기)
  - Authentication (인증)
  - Caching (URL 캐쉬)
  - Cookies (쿠키)
  - SOCKS and HTTP proxy (프록시)

NSURLConnection

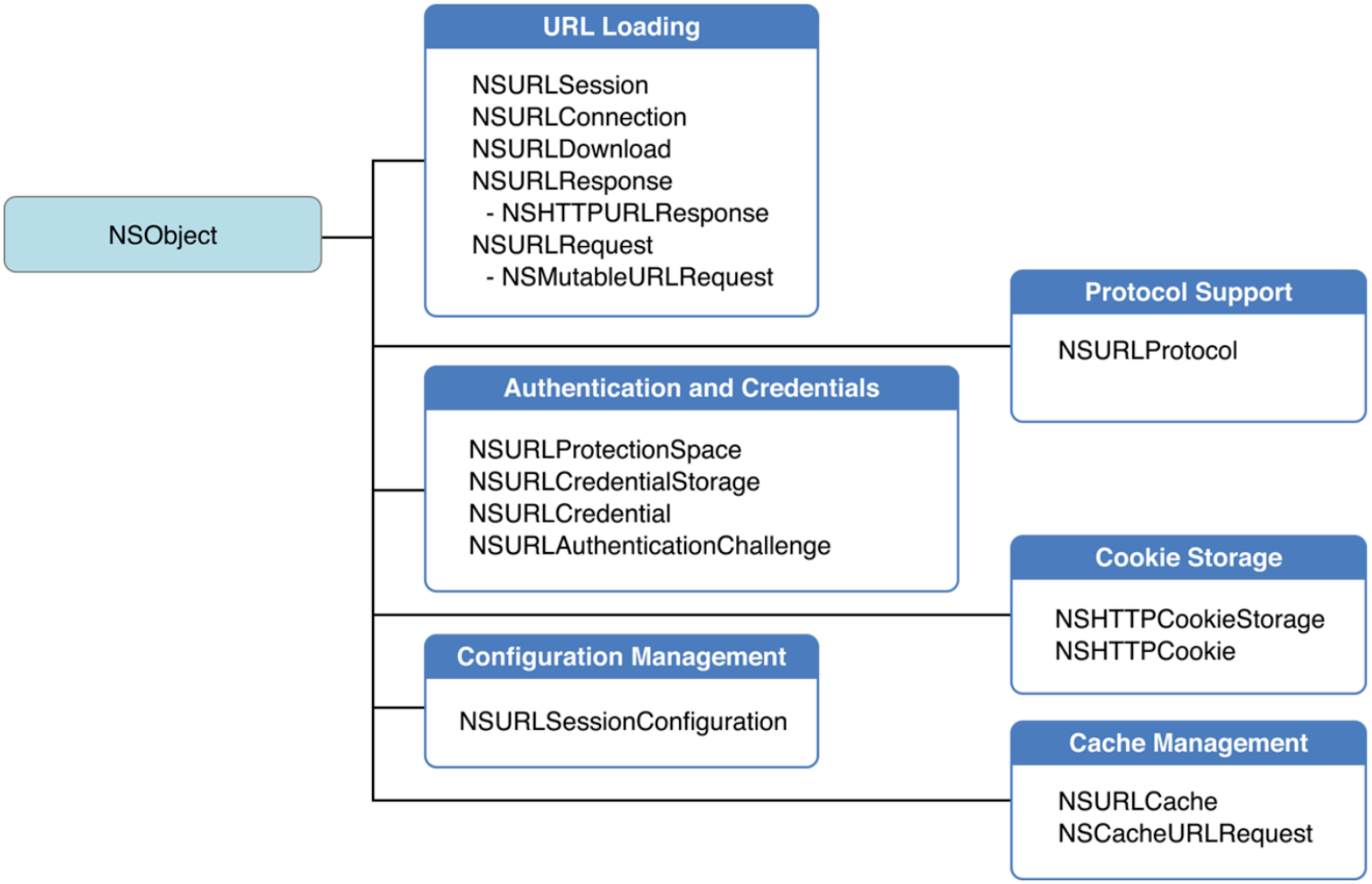
Life Cycle

1. Create NSURLRequest
2. Send request
3. Wait for response
4. Wait and receive for data
5. Finish or Error

App Transport Security Settings



URL Session Classes



NSURLSession

Session

Singleton shared session

+ 델리게이트 없이 간단한 비동기 요청

Default session

+ 기본 설정 세션. 커스텀 설정 가능

Ephemeral session

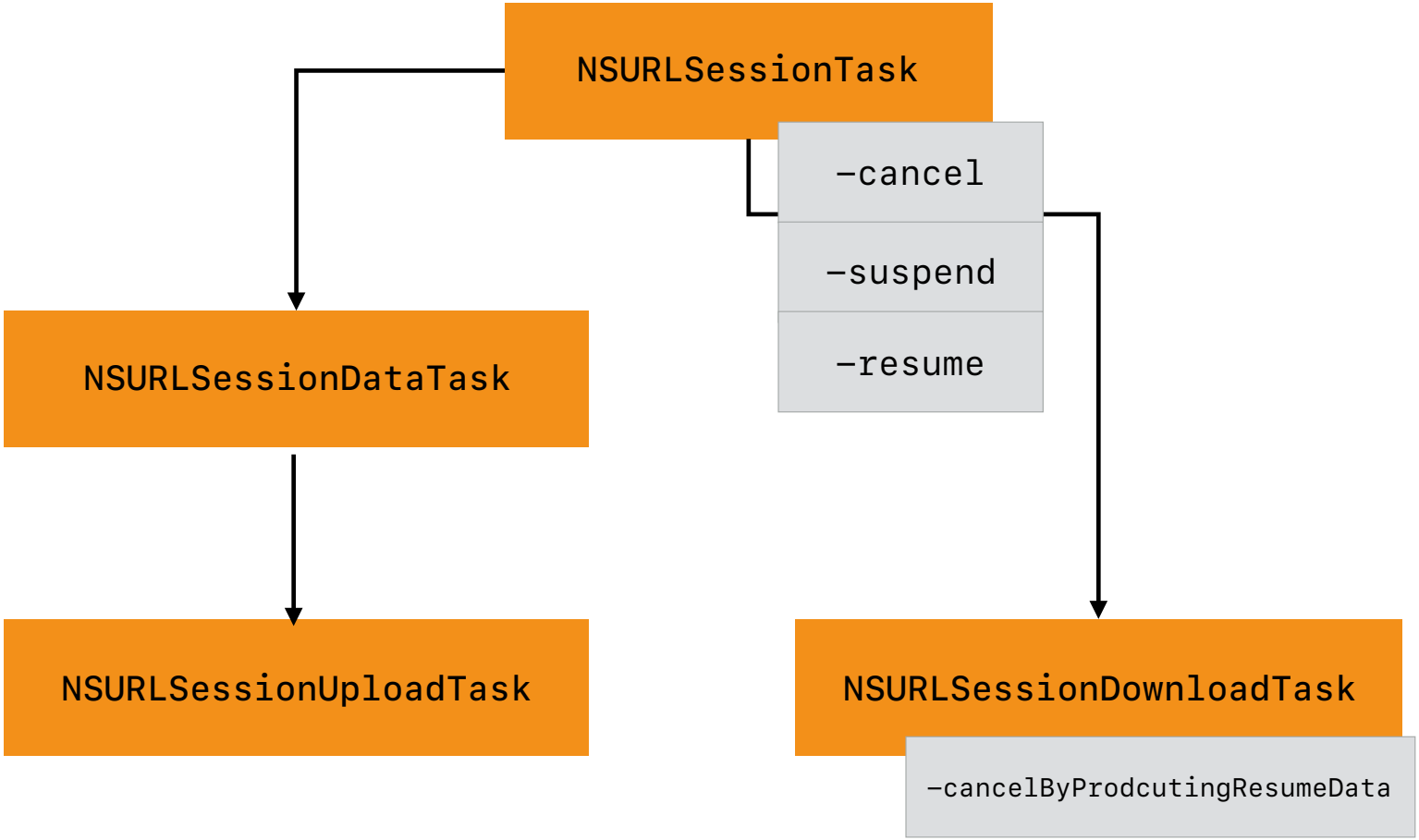
+ 델리게이트 없이 비공개(private) 세션

Background session

+ 백그라운드 동작을 위한 세션

NSURLSession

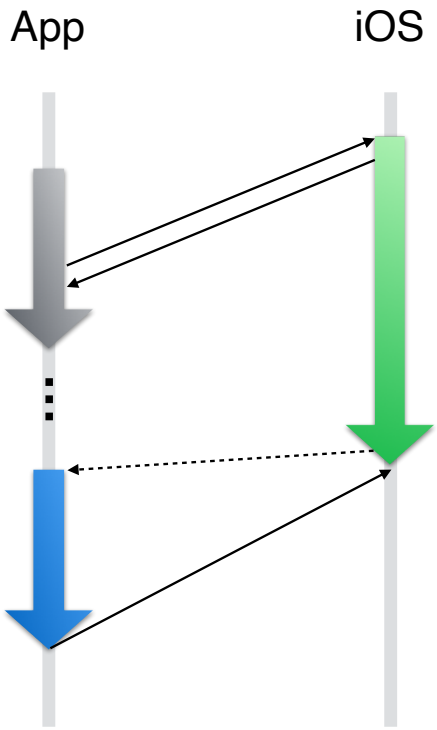
Tasks



NSURLSession

The Next Network APIs

1. Create a session configuration
2. Create a session (with delegate)
3. Create Task object with URL
4. Send request with closure
5. Wait for response (background)
6. Wait and receive for data
7. Finish or Error by closure



https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/URLLoadingSystem/URLLoadingSystem.html

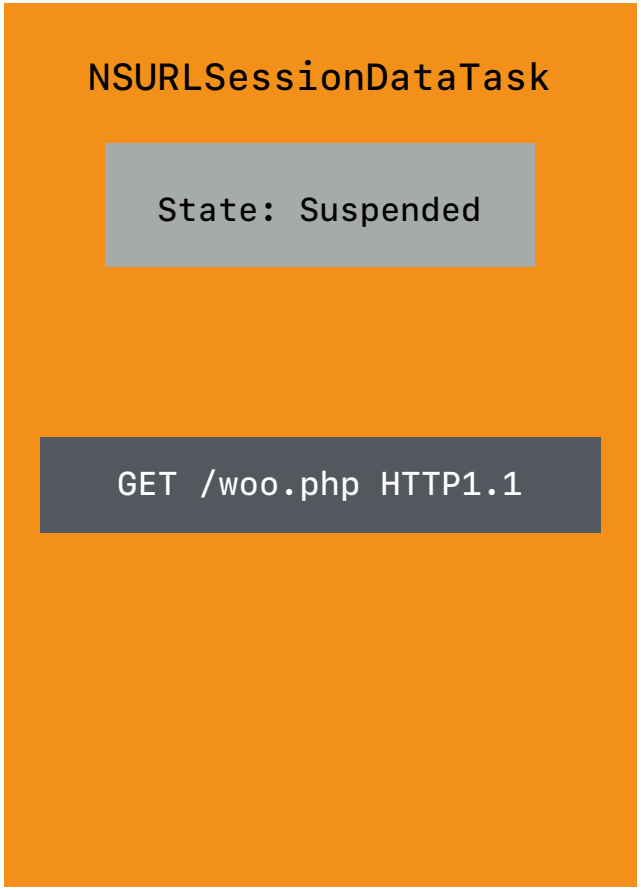
NSURLSession DataTask 예제

```
func dataTask(with url: URL, completionHandler: (Data?, URLResponse?, Error?)
-> Void) -> URLSessionDataTask
```

```
URLSession(configuration: URLSessionConfiguration.default).dataTask(with:
    URL(string: "http://apple.com")!) {
        (data, response, error) in
        var resultHTML = String(data: data!, encoding: String.Encoding.utf8)
        print(resultHTML)

   }.resume()
```

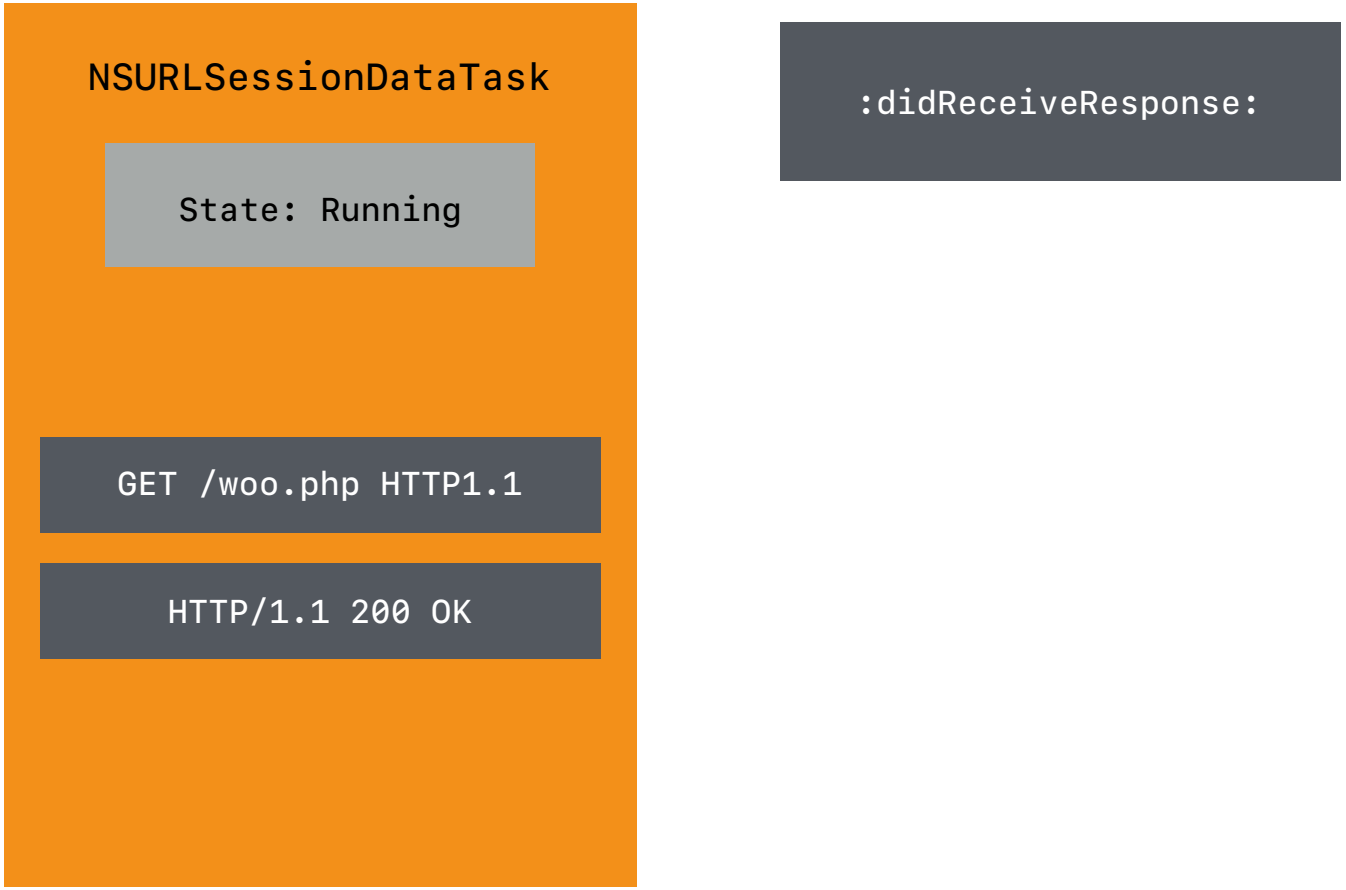
NSURLSession

Data task



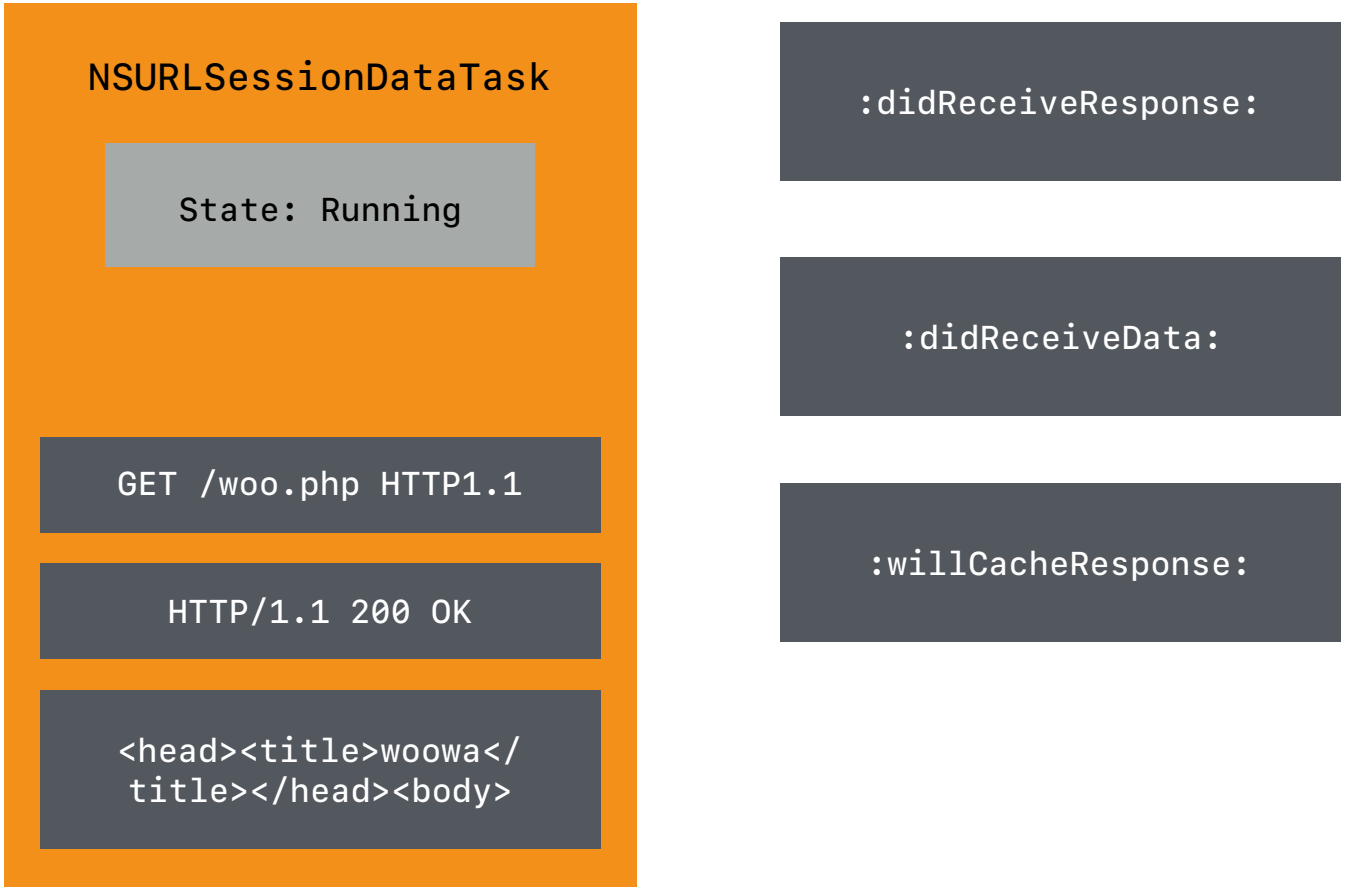
NSURLSession

Data task



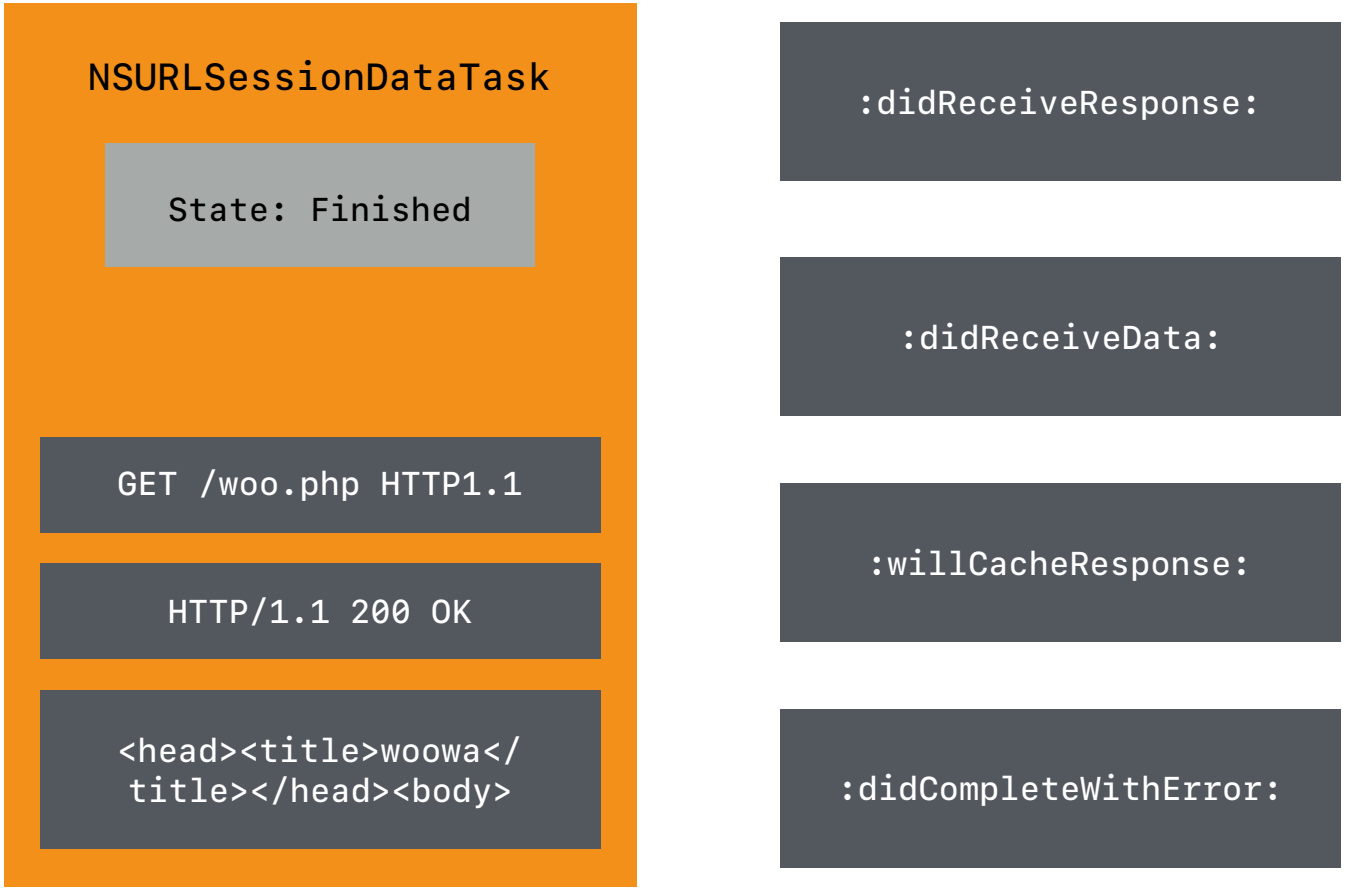
NSURLSession

Data task



NSURLSession

Data task

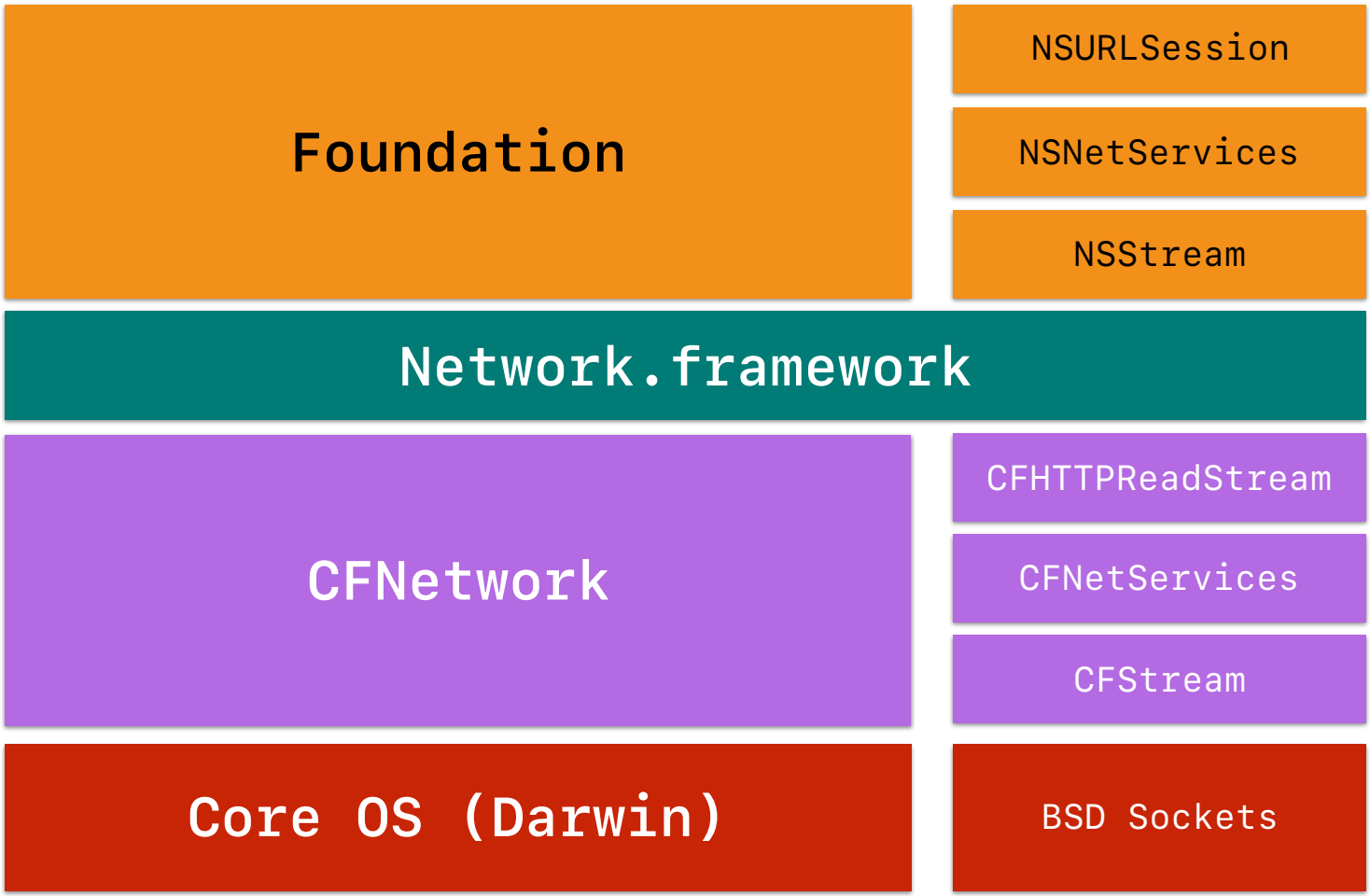


Background Transfer

앱 멈춘후에도 백그라운드 전송 지원

- 이벤트 처리를 위한 델리게이트 필수
- HTTP / HTTPS 프로토콜만
- 리다이렉트 지원
- 업로드는 파일만 가능

Network APIs



Network.framework

Connection Setup

```
// Create an outbound connection
import Network
let connection = NWConnection(host: "mail.example.com", port: .imaps, using: .tls)

connection.stateUpdateHandler = { (newState) in
    switch(newState) {
    case .ready:
        // Handle connection established
    case .waiting(let error):
        // Handle connection waiting for network
    case .failed(let error):
        // Handle fatal connection error
    default:
        break
    }
}

connection.start(queue: myQueue)
```

출처: WWDC

Network.framework

Listener

```
// UDP Bonjour listener

do {
    if let listener = try NWListener(parameters: .udp) {

        // Advertise a Bonjour service
        listener.service = NWListener.Service(type: "_camera._udp")

        listener.newConnectionHandler = { (newConnection) in
            // Handle inbound connections
            newConnection.start(queue: myQueue)
        }

        listener.start(queue: myQueue)
    }
} catch {
    // Handle listener creation error
}
```

출처: WWDC

Network.framework

Send

```
// Send a single frame
func sendFrame(_ connection: NWConnection, frame: Data) {

    // The .contentProcessed completion provides sender-side back-pressure
    connection.send(content: frame, completion: .contentProcessed { (sendError) in

        if let sendError = sendError {
            // Handle error in sending

        } else {
            // Send has been processed, send the next frame
            let nextFrame = generateNextFrame()
            sendFrame(connection, frame: nextFrame)
        }
    })
}
```

출처: WWDC

Network.framework

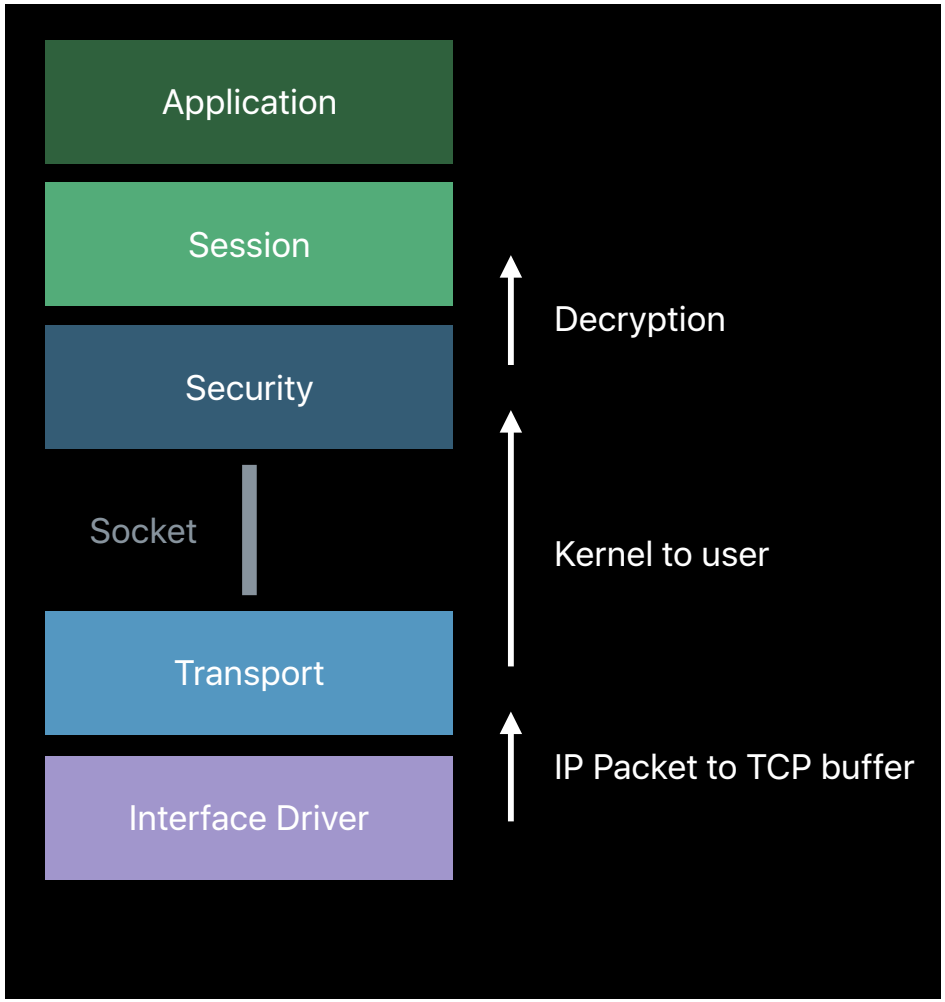
Receive

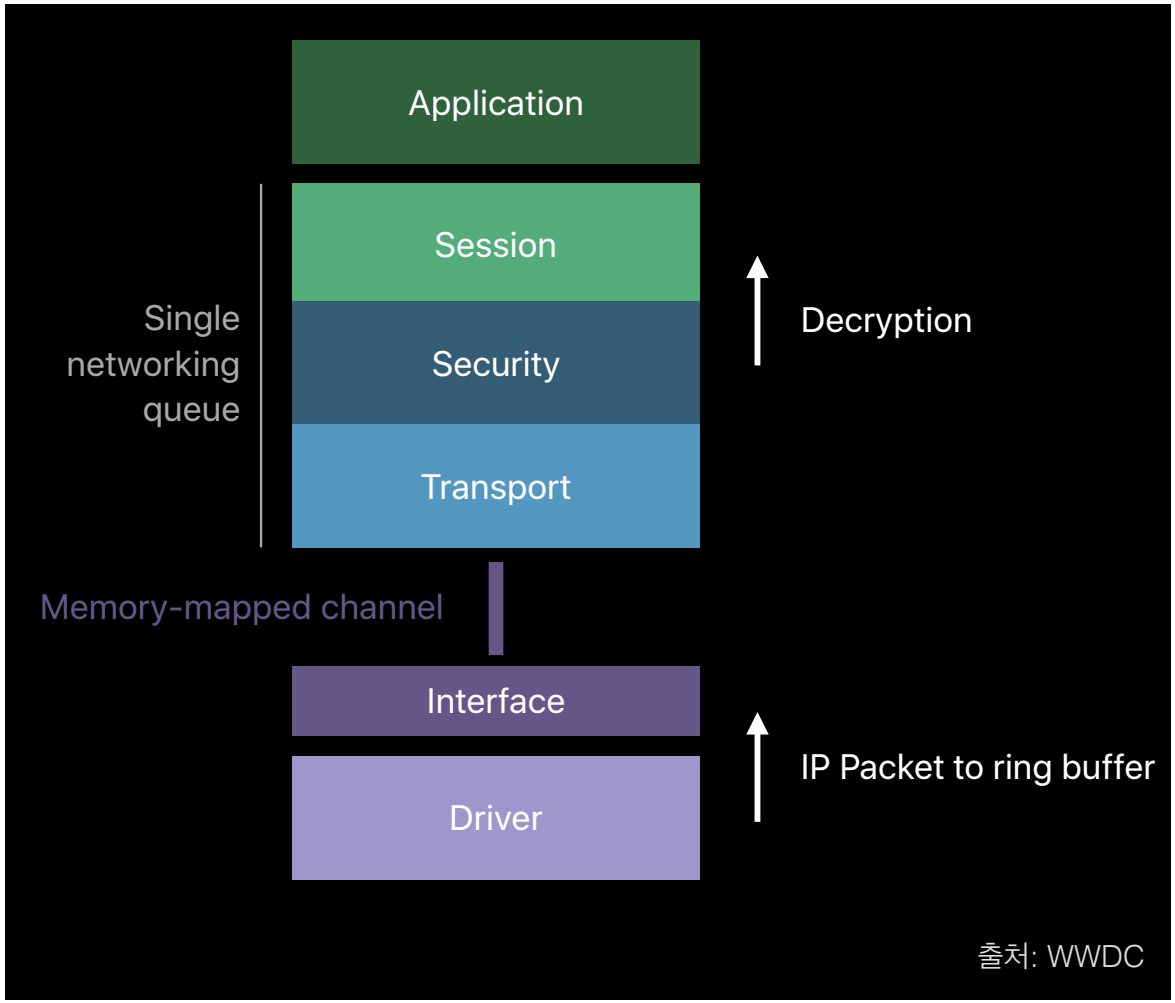
```
// Read one header from the connection
func readHeader(connection: NWConnection) {
    // Read exactly the length of the header
    let headerLength: Int = 10
    connection.receive(minimumIncompleteLength: headerLength, maximumLength: headerLength)
            { (content, contentContext, isComplete, error) in
        if let error = error {
            // Handle error in reading

        } else {
            // Parse out body length
            readBody(connection, bodyLength: bodyLength)
        }
    }
}

// Follow the same pattern as readHeader() to read exactly the body length
func readBody(_ connection: NWConnection, bodyLength: Int) { ... }
```

출처: WWDC

Network.framework

User-Space





ALAMOFIRE

Elegant Networking in Swift

https://github.com/Alamofire/Alamofire

Chainable Request / Response Methods
URL / JSON / plist Parameter Encoding
Upload File / Data / Stream / MultipartFormData
Download File using Request or Resume Data
Authentication with URLCredential
HTTP Response Validation
Upload and Download Progress Closures with Progress
cURL Command Output
Dynamically Adapt and Retry Requests
TLS Certificate and Public Key Pinning
Network Reachability
Comprehensive Unit and Integration Test Coverage
Complete Documentation

서버에 GET 요청하기

새로운 GET 버튼을 추가하고 [GET] 버튼을 누르면,

http://public.codesquad.kr/jk/storeapp/main.json

주소에서 JSON 데이터를 받아오는 모델 클래스를 만드세요.

응답을 받으면 Notification을 보내세요.

연습 문제

JSON 데이터에 있는 이미지 다운로드 URL에 해당하는

Download Task 방식으로 Image 파일들을 다운받으세요

저장할 때는 앱 번들 Cache 디렉터리에 저장하세요

Mobility and Cost

Mobility

Challenges

Computers fit in pockets

Multiple interfaces

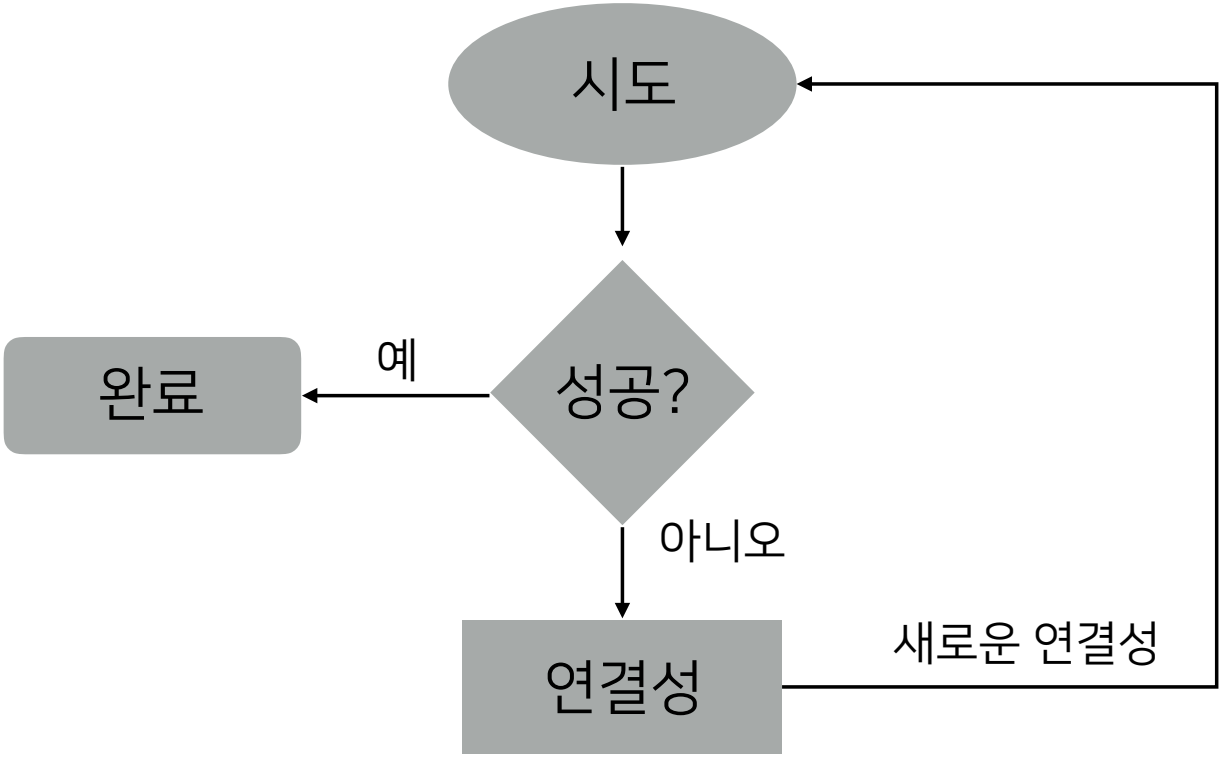
* Ethernet
* Wi-Fi
* Cellular

Changing environment

* Train through tunnel
* Arriving home to Wi-Fi

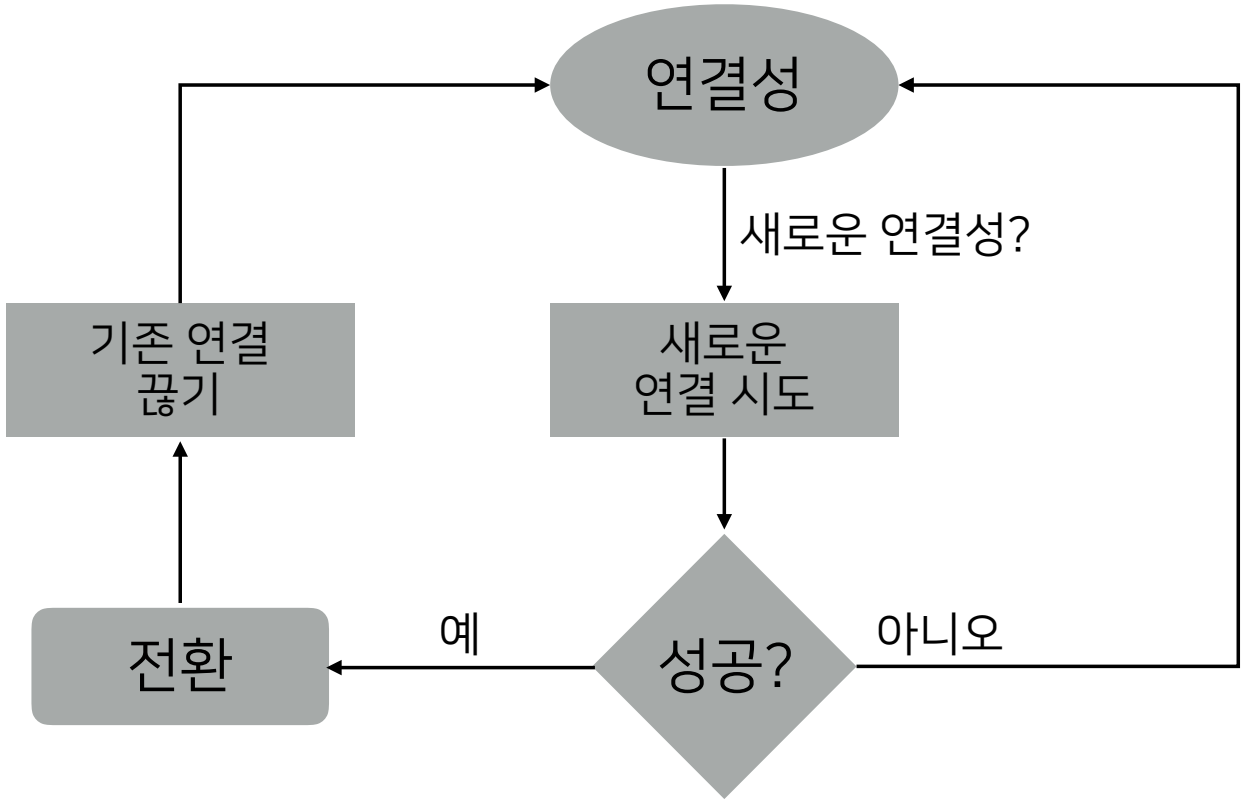
Mobility

Connecting



Mobility

Connected



비용 (Cost)

Cellular

- Power Cost
- Money Cost

Wi-Fi

- Power Cost
- Money Cost(?)

비용 (Cost)

해결방법

Money

- Cache data
- Fetch appropriately sized resources
- Fetch only what is necessary

Power

- Fetch in bursts

Reachability 클래스로 인터넷 연결 여부를 판단하세요

Reachability.m 또는 Alamofire 를 프로젝트에 추가하세요.

인터넷에 연결됐는지 판단해서 연결된 경우,

화면의 가장자리(border)를 UIColor.green 으로 표시하고

연결이 안되어 있는 경우 border를 UIColor.red 로 표시하세요.

미션

네트워크 모델 객체를 합치기

Network Model

https://h3rb9c0ugl.execute-api.ap-northeast-2.amazonaws.com/develop/baminchan/main

https://h3rb9c0ugl.execute-api.ap-northeast-2.amazonaws.com/develop/baminchan/soup

https://h3rb9c0ugl.execute-api.ap-northeast-2.amazonaws.com/develop/baminchan/side

* 위 URL 에서 JSON을 받아서 Serialization 하고 Array로 만드는 함수를 만드세요.
* 다운로드가 완료되면 Array를 Notification에 담아서 POST

Main ViewController

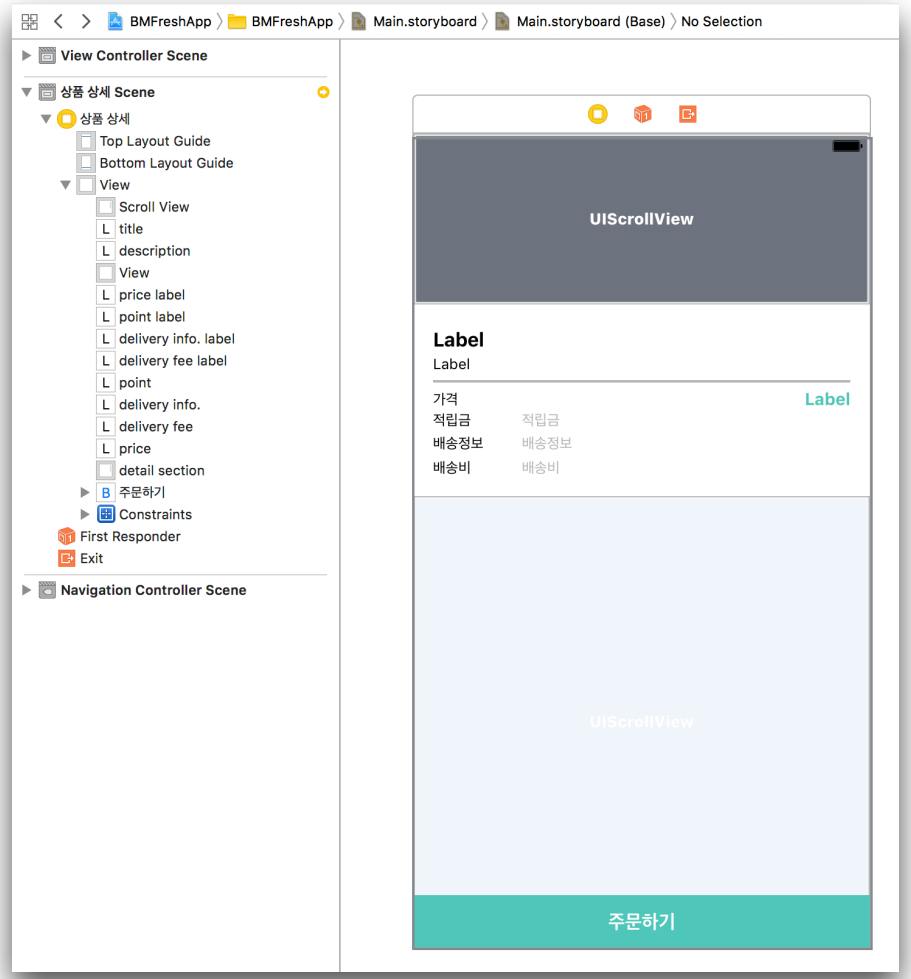
* viewDidLoad에서 Observer를 등록해놓고
* Network Model에 3가지 데이터에 대해 요청하세요.
* Notification을 모두 받으면 TableView를 그리세요.
* 음식 메뉴를 나타내는 클래스를 설계하고 만드세요.

Detail ViewController

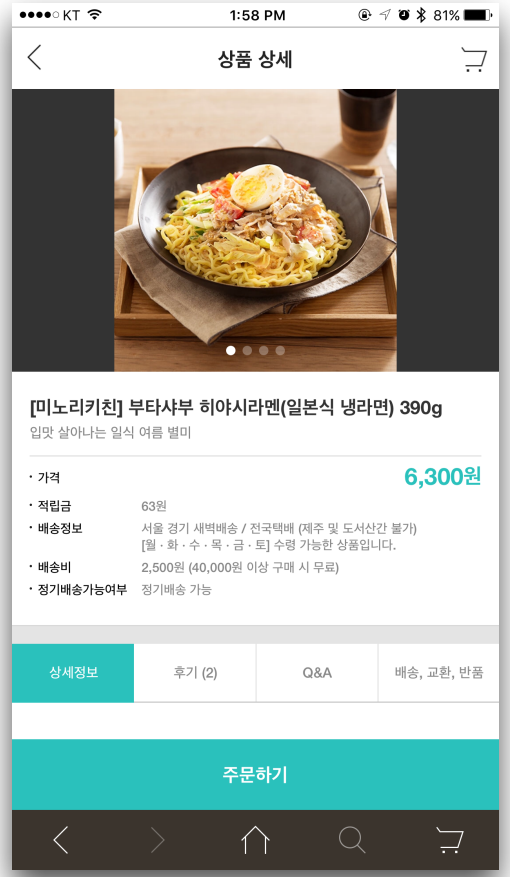
`https://h3rb9c0ugl.execute-api.ap-northeast-2.amazonaws.com/develop/baminchan/detail/{hash}`

- ViewController 를 Navigation Controller로 embed 하세요.
- cell을 선택하면 상세 화면을 보이도록 만드세요.
- 상세 화면(DetailViewController)에 선택한 cell의 detail_hash 값을 전달하세요.
- 위와 같은 URL 형식으로 요청하고 받은 JSON 데이터를 Serialization 하고 배열로 지정하세요.
- 상세 화면 디자인은 다음 슬라이드를 참고하세요.
- [주문하기] 버튼을 누르면, 슬랙으로 “누가-얼마짜리-메뉴” 주문을 POST 요청으로 보내세요. 그리고 이전 화면으로 돌아갑니다.

ProductDetailViewController



실제 앱 화면



Detail JSON

{“hash":"HBDEF",
“data":{"top_image":"https://cdn.bmf.kr/_data/product/HBDEF/
6ef14155afc5b47e8c9efd762f7a6096.jpg",
“thumb_images":["https://cdn.bmf.kr/_data/product/HBDEF/
6ef14155afc5b47e8c9efd762f7a6096.jpg","https://cdn.bmf.kr/_data/product/HBDEF/
e30bd6de9340fc05db3cd1d1329b2c56.jpg","https://cdn.bmf.kr/_data/product/HBDEF/
4cce011a4a352c22cd399a60271b4921.jpg","https://cdn.bmf.kr/_data/product/HBDEF/
8744504ff3bc315f901dca1f26fe63a1.jpg"],
"product_description":"일본인의 소울푸드! 한국인도 좋아하는 소고기덮밥”,
“point":"52원",
"delivery_info":"서울 경기 새벽배송 / 전국택배 (제주 및 도서산간 불가) [월 · 화 · 수 · 목 · 금 · 토]
수령 가능한 상품입니다.”,
"delivery_fee":"2,500원 (40,000원 이상 구매 시 무료)”,
“prices":["6,500원","5,200원"],
"detail_section":["https://cdn.bmf.kr/_data/product/HBDEF/
2c62efce07c96be700f317b60c537c6e.jpg","https://cdn.bmf.kr/_data/product/HBDEF/
768afd17faa8bf3461b8160ba0aa26bf.jpg","https://cdn.bmf.kr/_data/product/HBDEF/
72f1049b047f65f42a267d5bbd1e6204.jpg","https://cdn.bmf.kr/_data/product/HBDEF/
b2b3d0d2107ab91b16e0eb804cd84bc9.jpg","https://cdn.bmf.kr/_data/product/HBDEF/
9c2c53b40a11b79c90549a058c2da4b7.jpg","https://cdn.bmf.kr/_data/product/HBDEF/
2450219a4686d9d6d579fc04020929b4.jpg","https://cdn.bmf.kr/_data/product/HBDEF/
a8c434715709fe855f3ea1554ec362b6.jpg","https://cdn.bmf.kr/_data/product/HBDEF/
95816f09d3294641f2e0feacaa739991.jpg","https://cdn.bmf.kr/_data/product/HBDEF/
70b0c77d3ef5cdd6269588685bbefe43.jpg","https://cdn.bmf.kr/_data/product/HBDEF/
c0dd6887c9d9368604fc70d7fc3c4598.jpg","https://cdn.bmf.kr/_data/product/HBDEF/
4971475295545ec336c9479fabb25364.jpg"]}}

Detail ViewController 상세

* 최상위 View 커스텀 클래스를 UIScrollView로 지정하세요.
* ScrollView ContentSize에 대해 찾아보고,
* 전체 높이를 계산해서 스크롤되도록 값을 지정하세요.
* 상단 ScrollView 에 thumb_images 항목의 이미지들을 Page 형태로 추가하세요.
* 좌우로 페이지 넘기듯이 넘어가도록 만드세요.
* 하단에는 detail_section 항목의 이미지들을 코드로 이어서 붙이세요.