ITM103 iOS Application Development

Topic 10: Consuming Web Services

Objectives

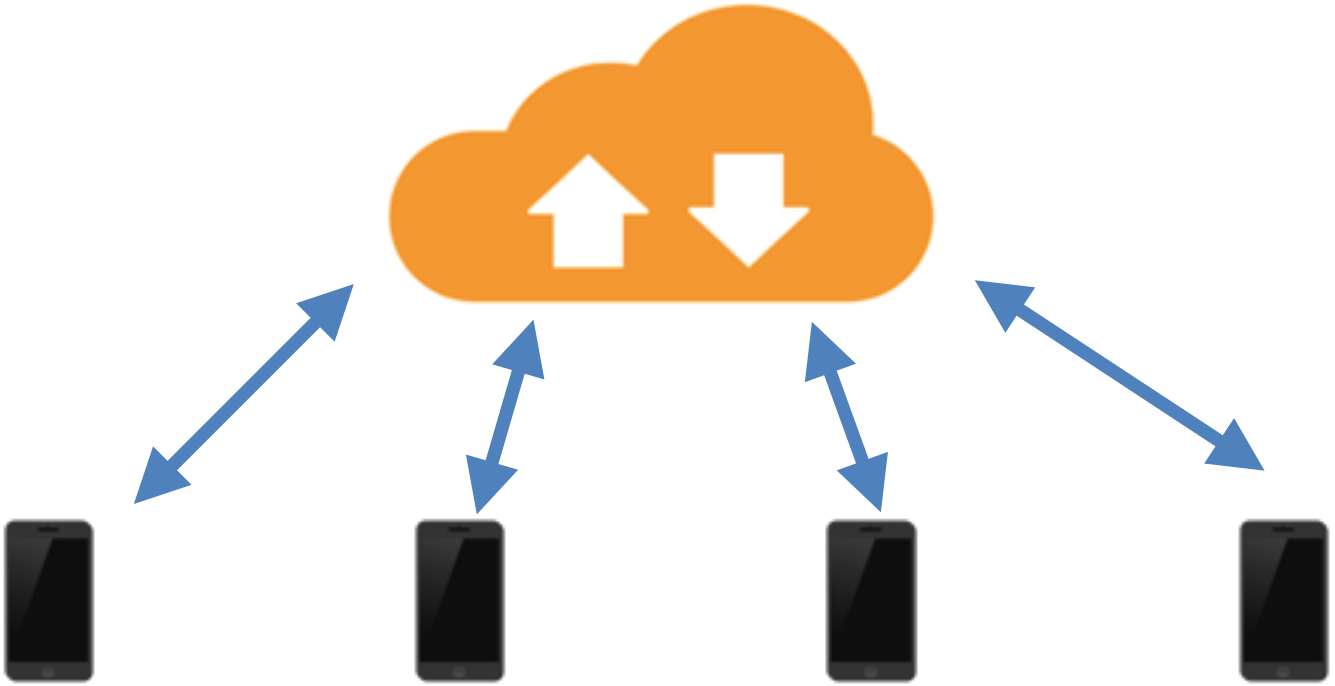
- By the end of the lesson, you will be able to:
  - Understand JSON framework
  - Fetch and Parse JSON feeds
  - Understand and develop applications using Grand Central Dispatch
  - Understand and develop applications using Cloud Persistent Storage

Consuming Web Services

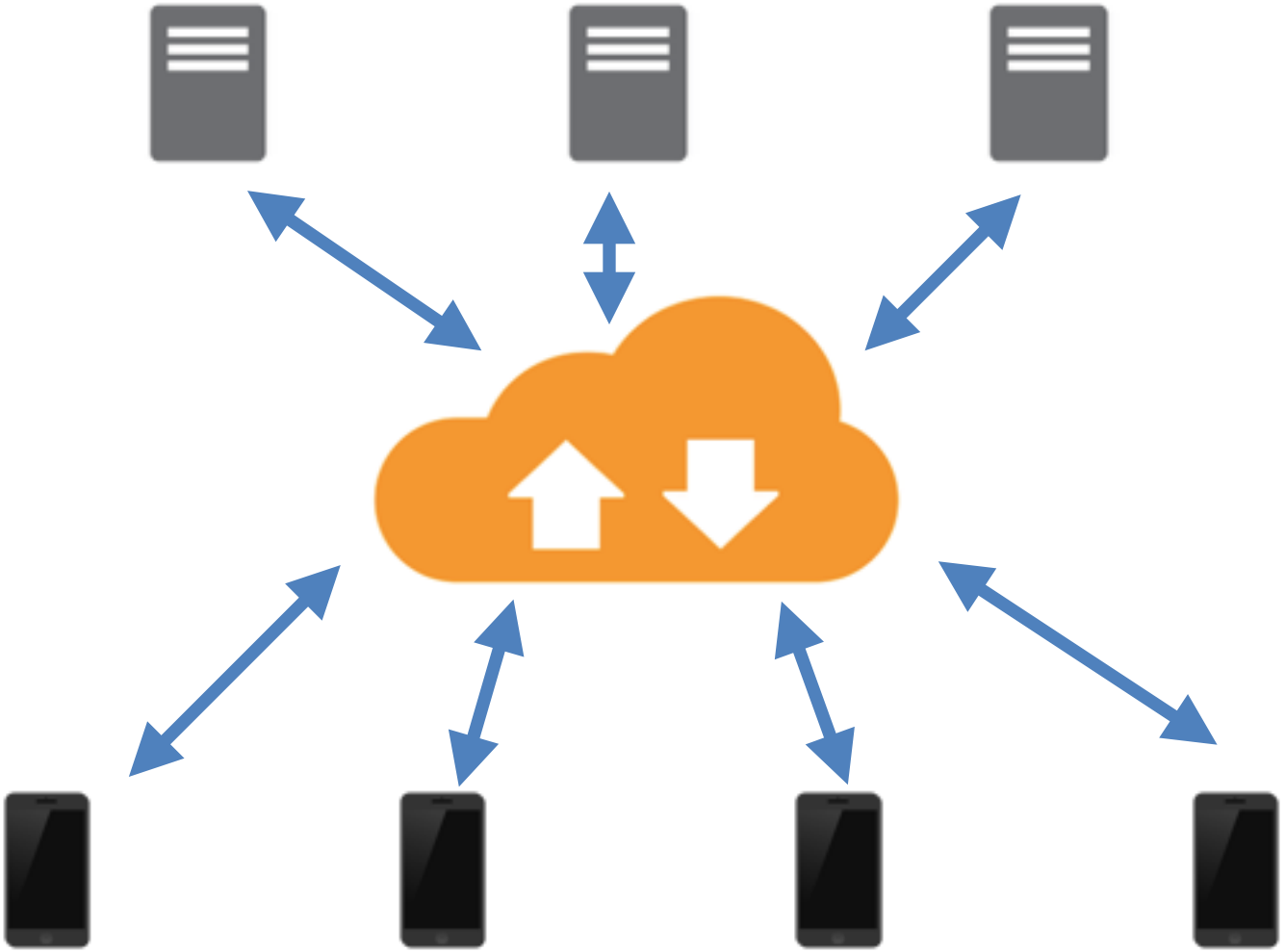
Why Web Services?

Why Web Services?

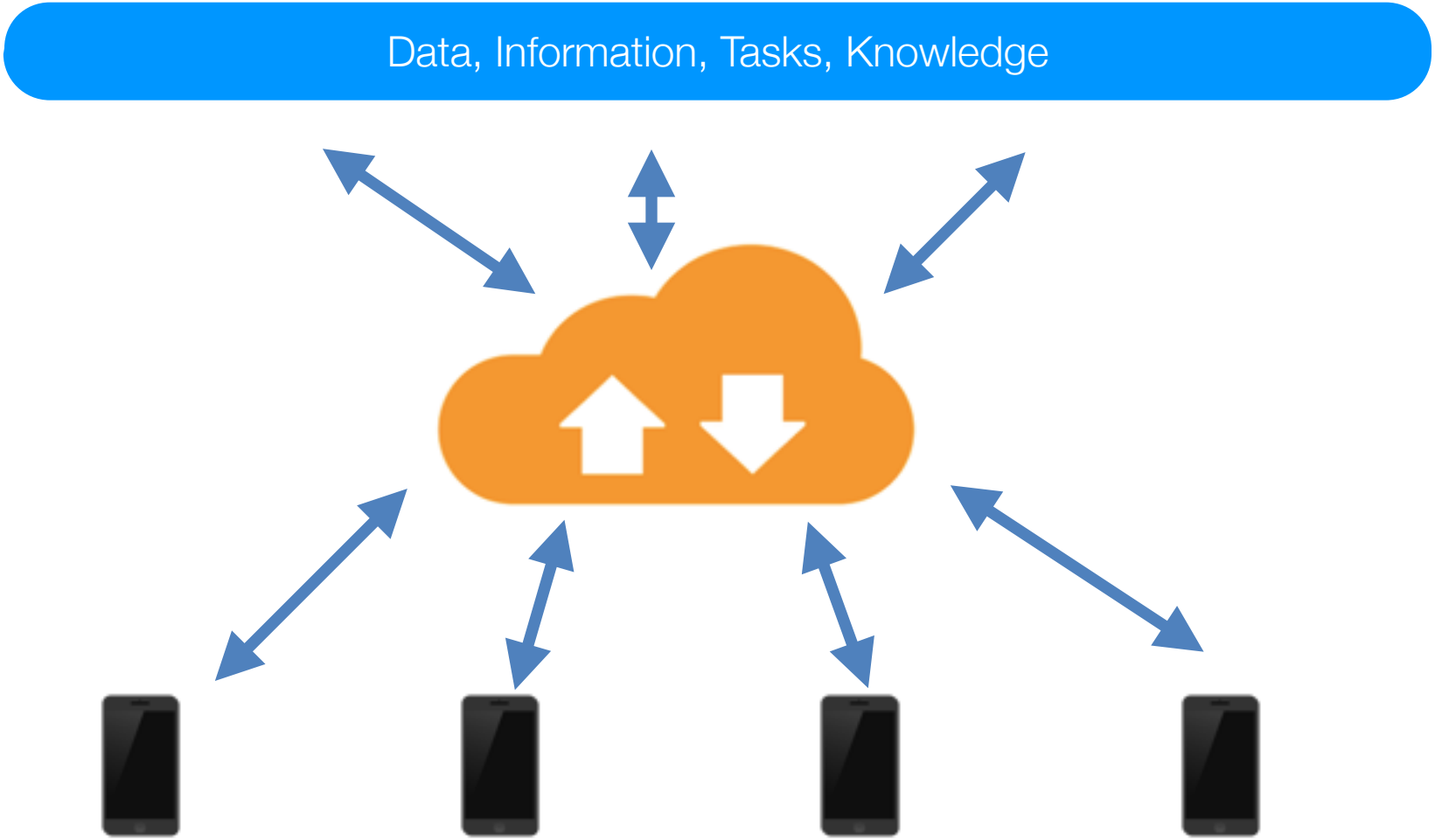
Why Web Services?



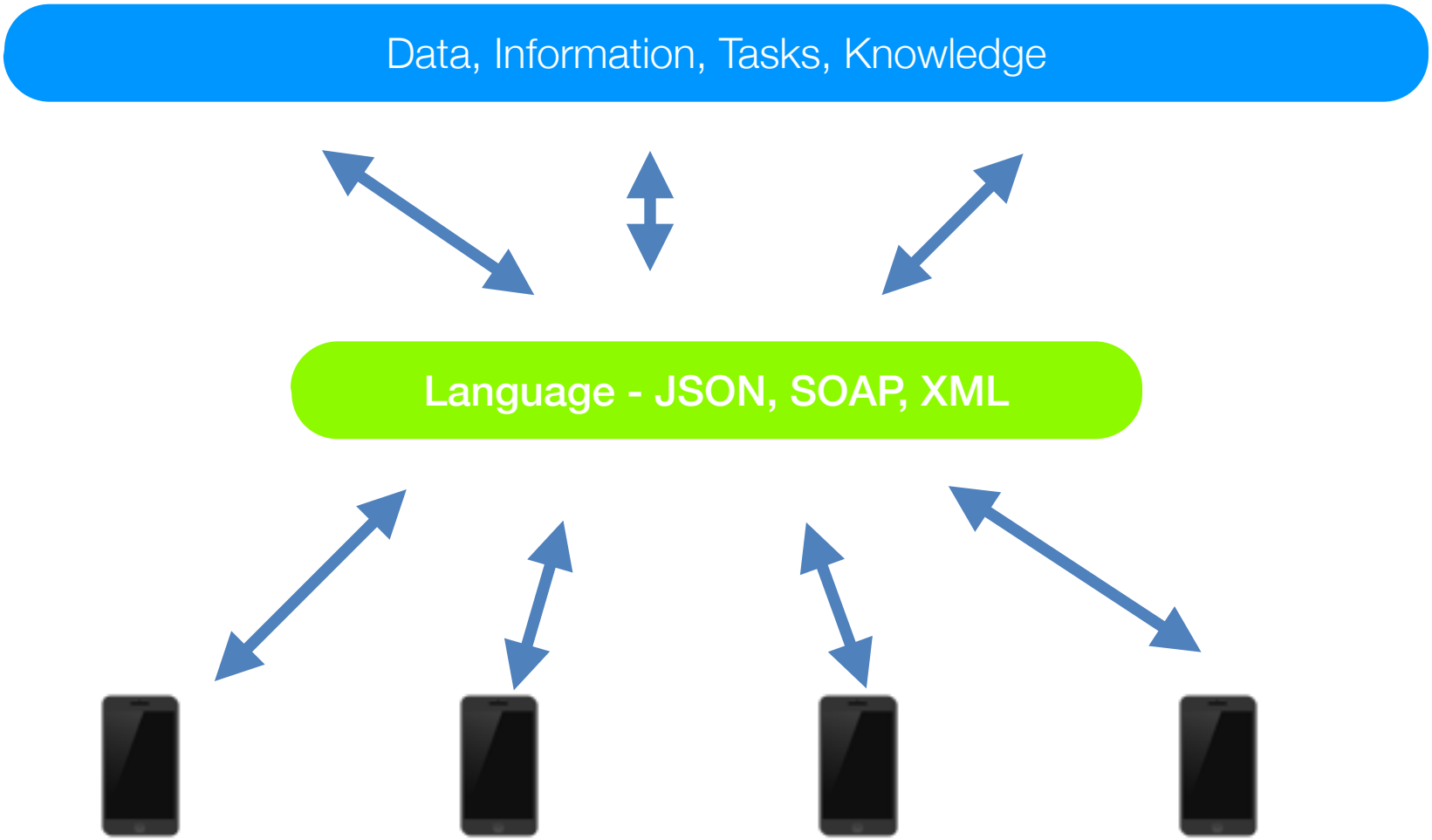
Why Web Services?



Why Web Services?



Why Web Services?



Consuming Web Services

Dealing with JSON

JSON

- **J**ava**S**cript **O**bject **N**otation

- It is a text-based, lightweight and easy way for storing and exchanging data.

- It's commonly used for representing structural data and data interchange in client-server application, an alternative to XML.

JSON - Example Format

```
{
    "title": "The Amazing Spider-man",
    "release_date": "03/07/2012",
    "director": "Marc Webb",
    "cast": [
        {
            "name": "Andrew Garfield",
            "character": "Peter Parker"
        },
        {
            "name": "Emma Stone",
            "character": "Gwen Stacy"
        },
        {
            "name": "Rhys Ifans",
            "character": "Dr. Curt Connors"
        }
    ]
}
```

JSONSerialization

- **Convert** Swift object to JSON data:
  - `data(withJSONObject: AnyObject, options) -> Data`

  - An object that may be converted to JSON must have the following properties:
    - The top level object is an Array or Dictionary.
    - All objects are instances of String, Number, Array, Dictionary, or nil.
    - All dictionary keys are instances of String.
    - Numbers are not NaN or infinity.

- https://developer.apple.com/reference/foundation/jsonserialization

JSONSerialization

- **Retrieve** JSON string into Swift object:
  - `jsonObject(with data: Data, options) -> Any`

  - The new Swift library returns **Any** as its type.
  - To simplify retrieving of data, we will develop a simple JSONDataAccessor wrapper (see Practical)

- https://developer.apple.com/reference/foundation/jsonserialization

JSONSerialization

- **Retrieve** JSON string into Swift object:

```
var data = str.data(using: String.Encoding.utf8)
let obj = JSONDataAccess(try JSONSerialization.jsonObject
    (with: data!, options: .mutableContainers))
```

- Examples:
  - `print (obj[0]?["test"]?.data)`
  - `print (obj[0]?["val"]?.data)`
  - `print (obj[0]?["arr"]?[0]?["x"]?.data)`
  - `print (obj[1]?["test"]?.data)`

Consuming Web Services

Uploading/Downloading over HTTP

URLSession, MutableURLRequest

- Provides the method for retrieving the contents of a URL.
- Provides a simple interface for:
  - asynchronously GETting and POSTing data from/to a web server.
  - upload / downloading of files to/from a web server.
- Supports HTTP, HTTPS, FTP, FILE, DATA protocols.
- URL sessions also support canceling, restarting or resuming, and suspending tasks.

- https://developer.apple.com/reference/foundation/urlsession

URLSession, MutableURLRequest

- MutableURLRequest
  - Use this to control whether you want to GET/POST to the server.

- URLSessions
  - Call the methods to consume web services **asynchronously**.
  - If you need to know when a request completes, you can call the methods with the completionHandler, passing in a closure to handle the completion.

URLSession, MutableURLRequest

```
// Create a new request
//
let request = MutableURLRequest(url: URL(string:"http://time.jsontest.com")!)
request.httpMethod = "GET"

// Creates a new singleton session
//
let session = URLSession.shared

// Create a task with a completion handler.
//
let task = session.dataTask(with: request as URLRequest,
    ...









 )

// Starts the task asynchronously
//
task.resume()
```

You can use the request to set the HTTPMethod and HTTPBody for POST data

URLSession, MutableURLRequest

```
// Create a new request
//
let request = MutableURLRequest(url: URL(string:"http://time.jsontest.com")!)
request.httpMethod = "GET"

// Creates a new singleton session
//
let session = URLSession.shared

// Create a task with a completion handler.
//
let task = session.dataTask(with: request as URLRequest,
    completionHandler: {data, response, error -> Void in
    do {
        let json = try JSONSerialization.jsonObject(with: data!,
            options: .mutableLeaves) as! NSDictionary


        let t = json["time"] as! String
        let d = json["date"] as! String
        print("Time: \(t)")
        print("Date: \(d)")

    }
    catch
    {
        print ("Unable to load JSON from URL")
    }
})

// Starts the task asynchronously
//
task.resume()
```

De-serialize JSON string into a dictionary

Closure:
This code is executed when the download is completed.

URLSession, MutableURLRequest

- URLSession methods
  - dataTask(url)
  - dataTask(request)
  - downloadTask(url)
  - downloadTask(request)
  - uploadTask(request, fromFile)
  - uploadTask(request, fromData)

Connects to a web service through its URL only, and retrieves data from server. Great for HTTP GETs.

Used for uploading of files. Uses HTTP POST/PUT.

- Each method above has a version with the completionHandler

Data

- A buffer that contains the bytes of data.

- You can also use this to retrieve data **synchronously** from a web server (using only GET).
  - Data(contentsOf: URL)
  - Data(contentsOf: URL, options)

- https://developer.apple.com/reference/foundation/data

Consuming Web Services

Multi-Threading

Multi-threading

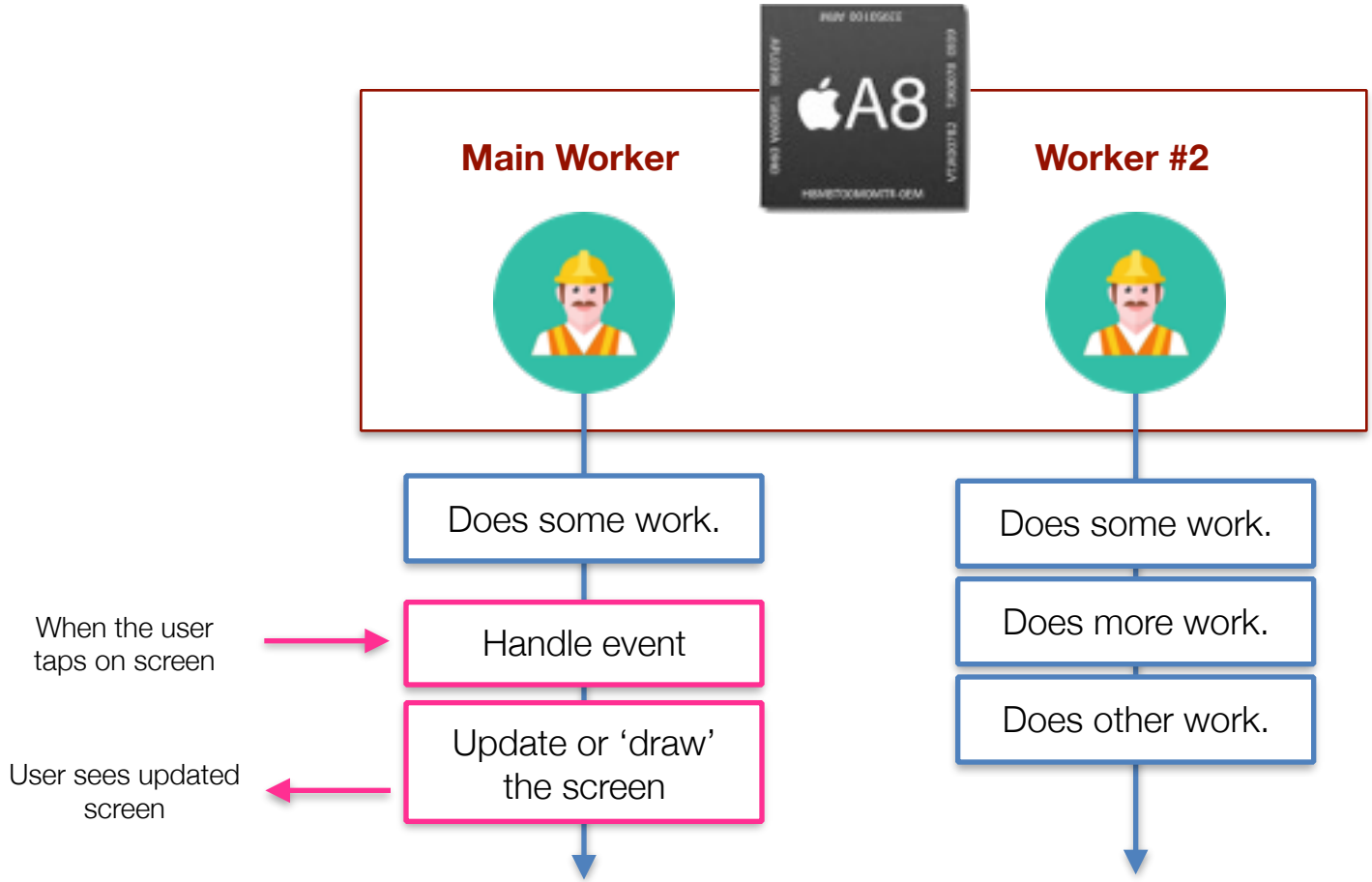
- Two modes of operation:
  - **Synchronous**

    Task executes in sequence.
  - **Asynchronous**

    Task executes in parallel.

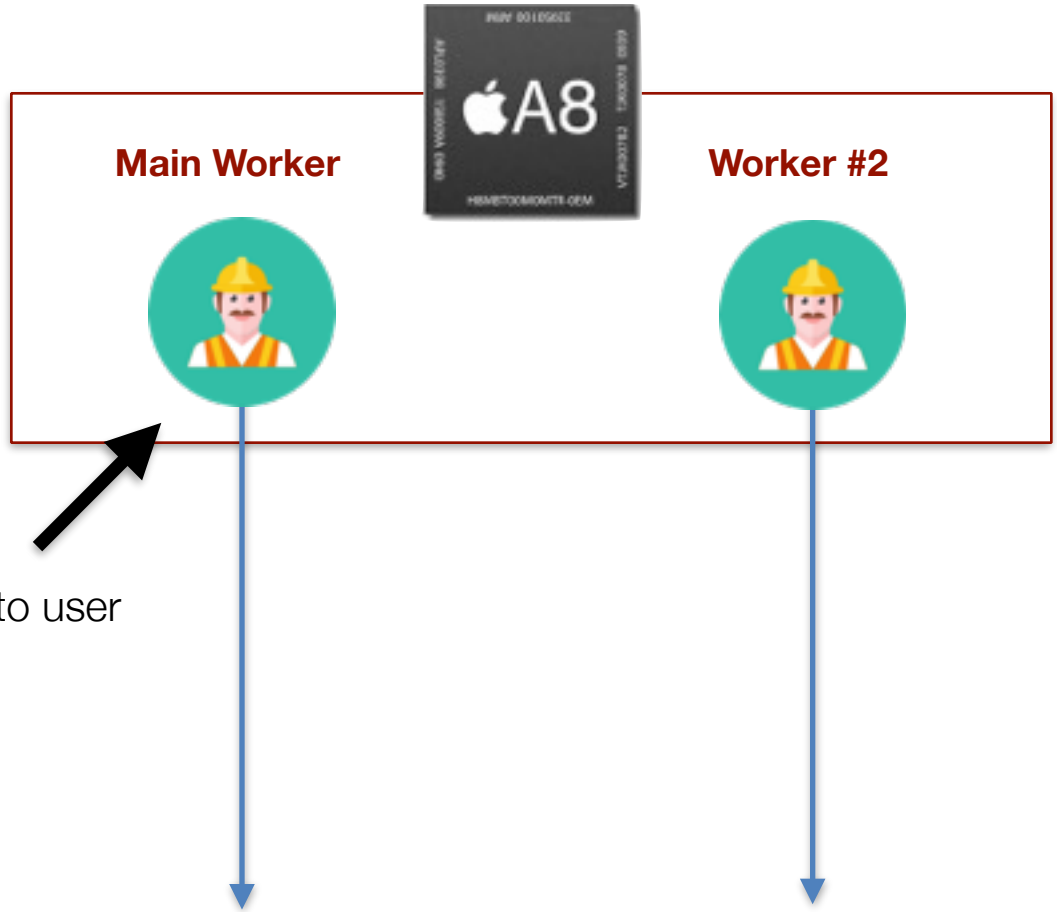
Multi-threading

- In any computer / device, the processor can run “threads”. Tasks execute in parallel.



Multi-threading

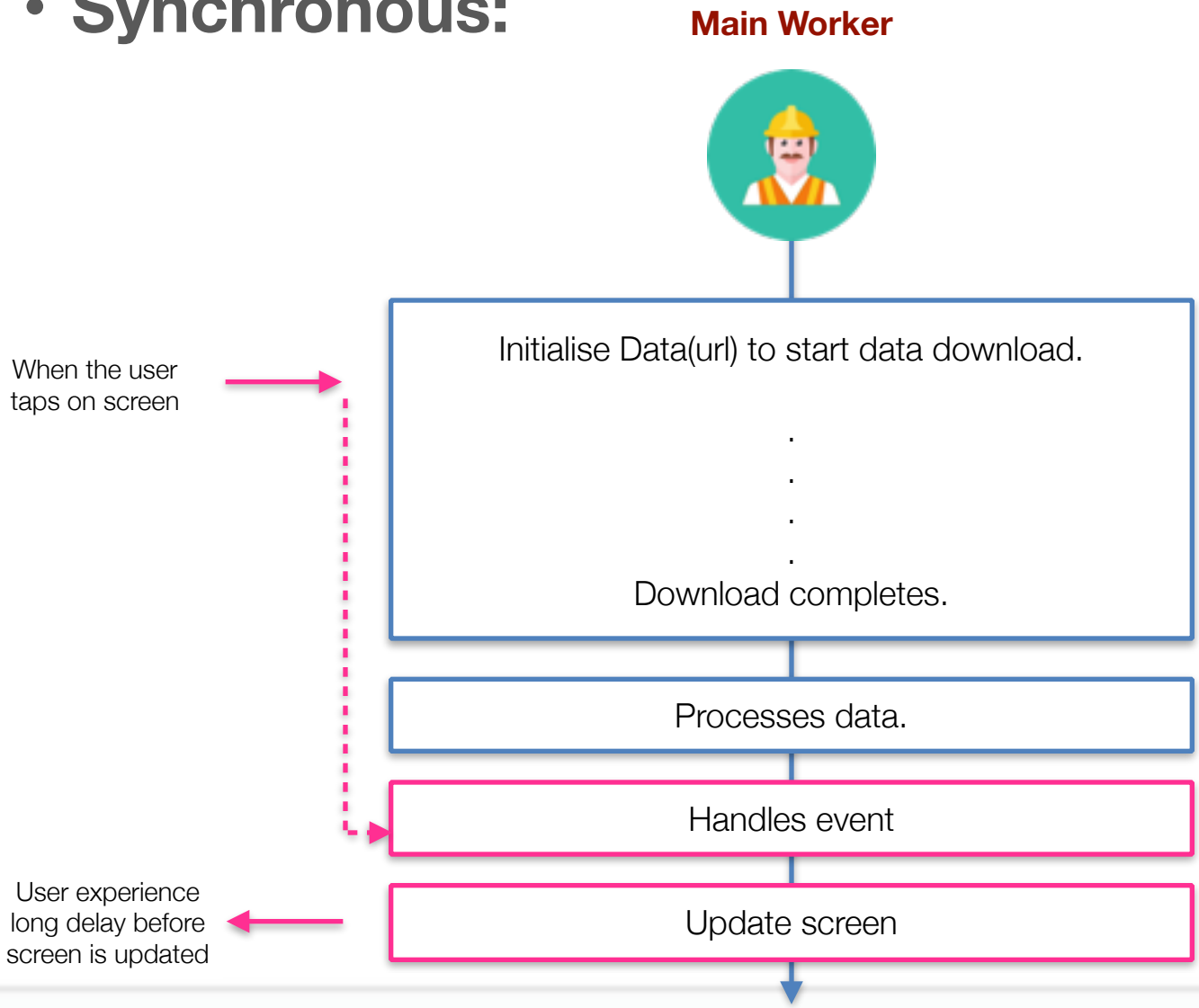
- In any computer / device, the processor can run "threads".



Responsible for
1. Responding to user
2. Updating UI

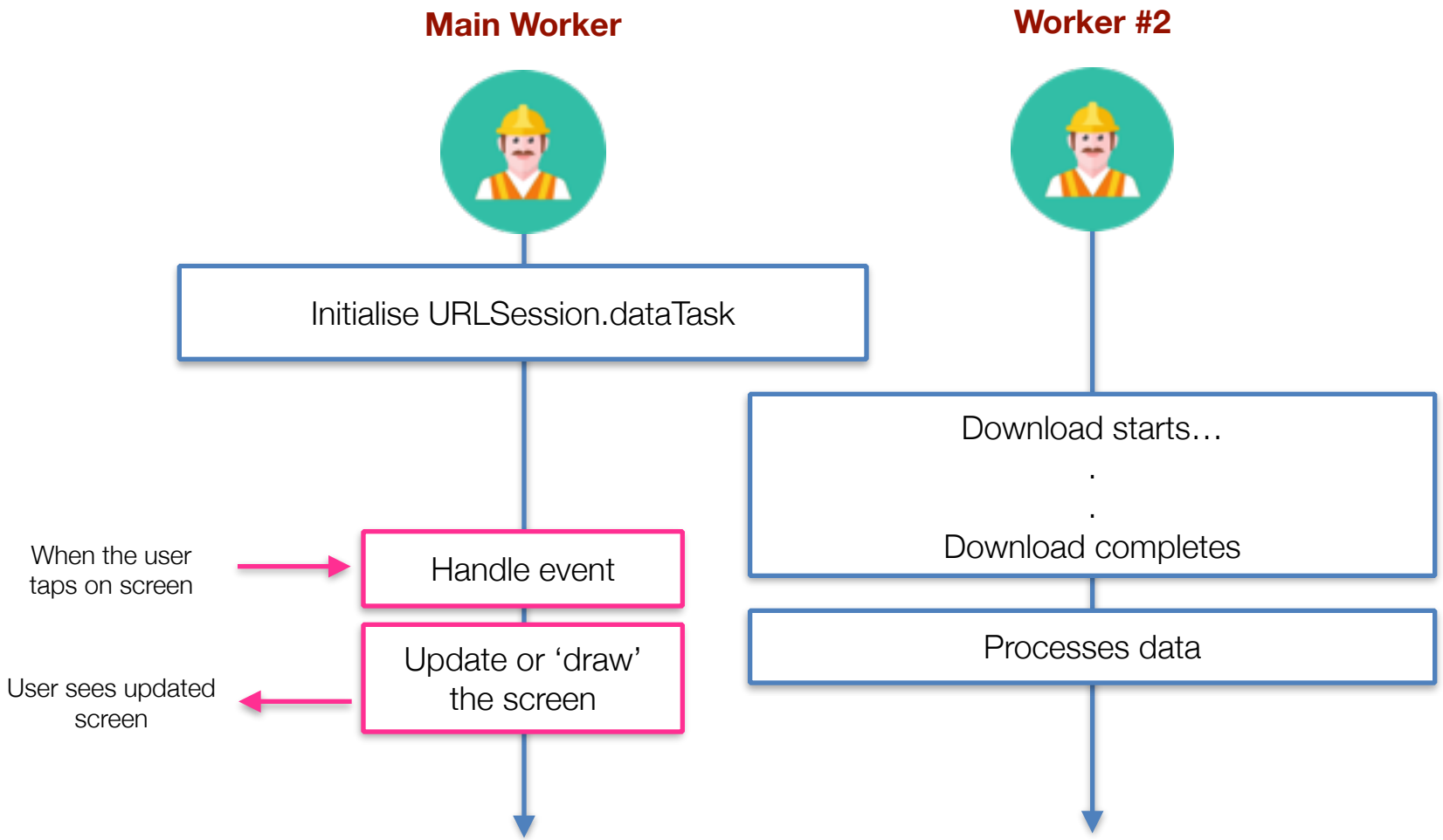
Multi-threading

- **Synchronous:**



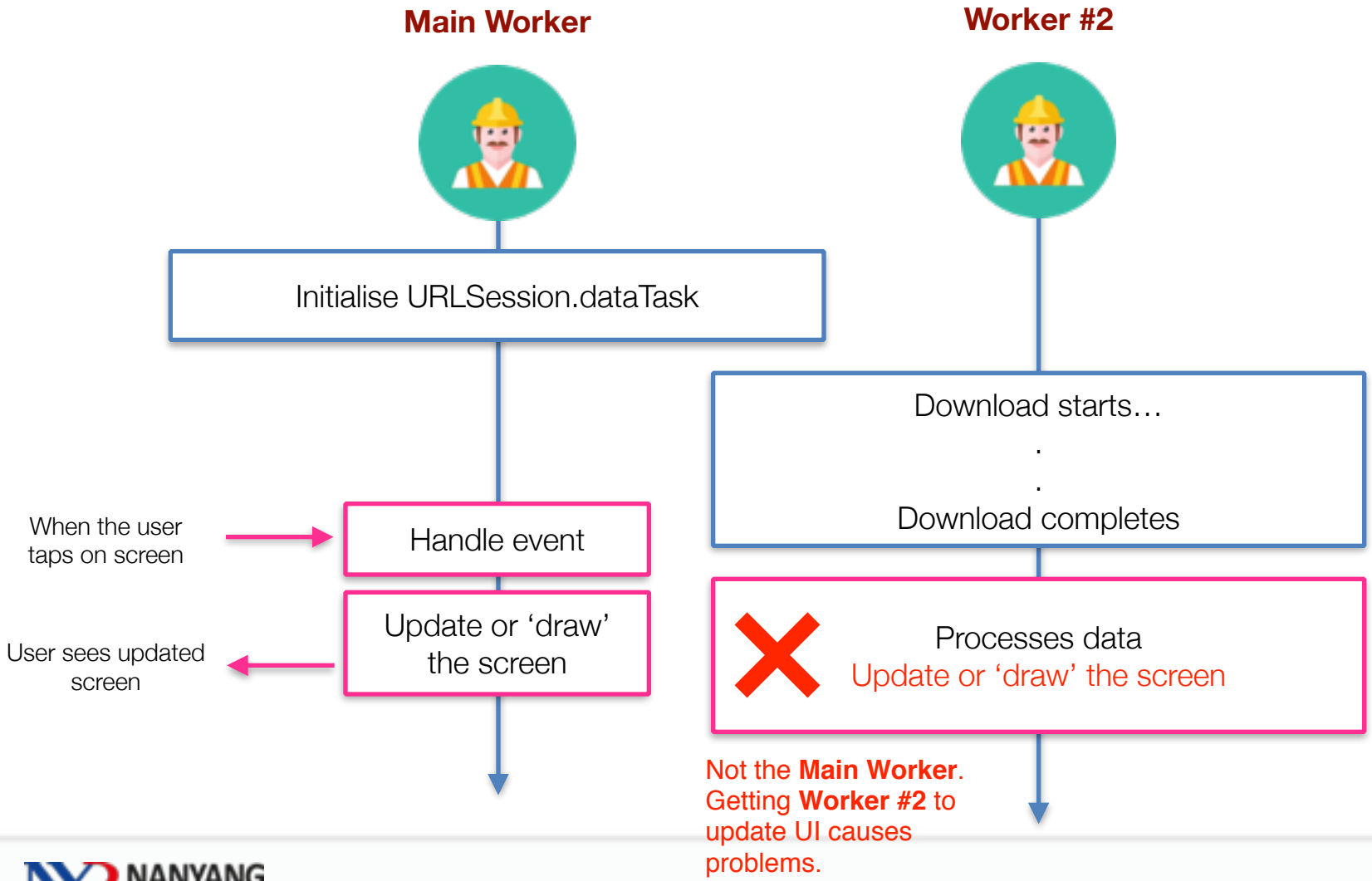
Multi-threading

- **Asynchronous:**



Multi-threading

- What if Worker #2 needs to update UI?



Consuming Web Services

Responsive Apps with Grand Central Dispatch (GCD)

Grand Central Dispatch (GCD)

- Enter **Grand Central Dispatch**
  - Apple's library that allows concurrent code execution.
  - improve your app's responsiveness by running part of your long-running tasks in background

    http://www.raywenderlich.com/79149/grand-central-dispatch-tutorial-swift-part-1

Example

Initialising

```
print ("Before download...")
```

Download data asynchronously

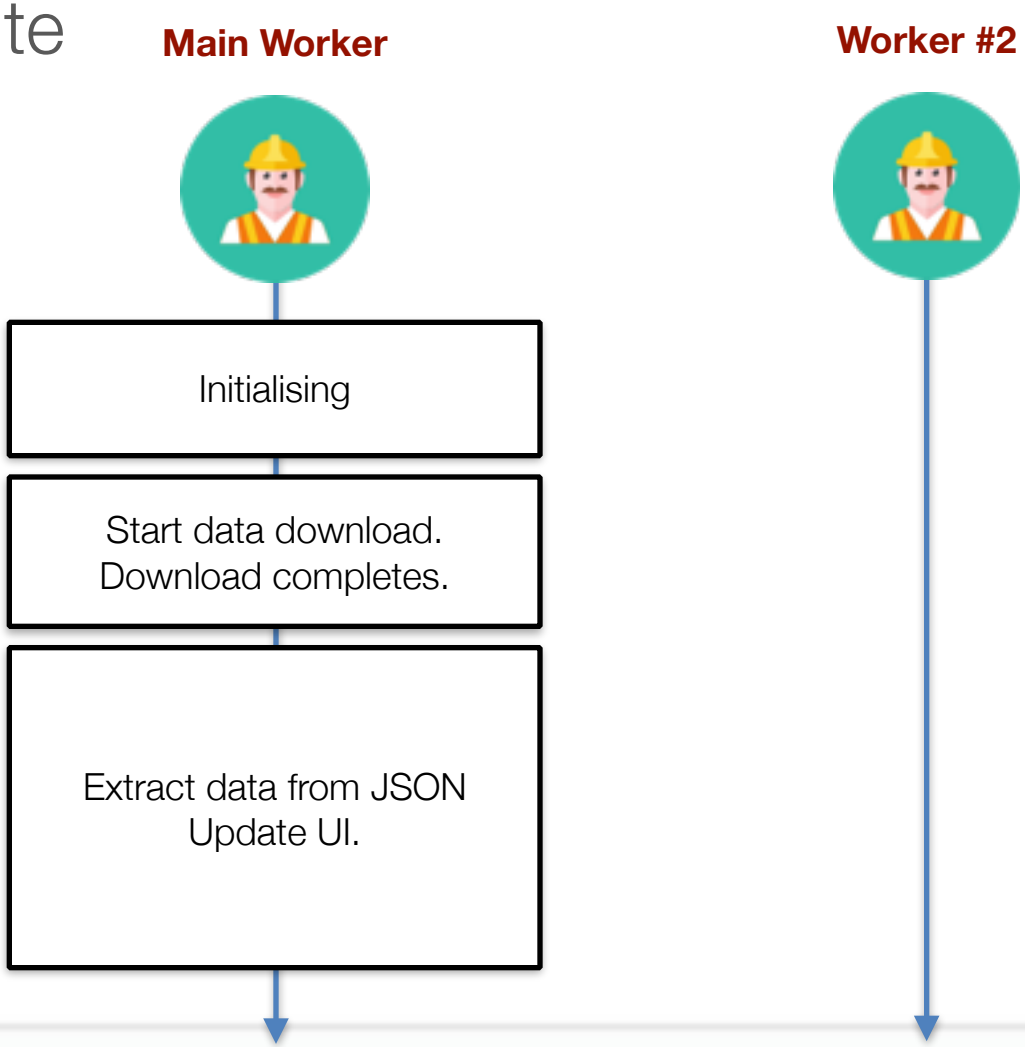
```
do {
    let data = try Data (contentsOf: googleUrl!)

    do {
        let json = JSONDataAccessor(try JSONSerialization.jsonObject(with: data,
            options:.mutableContainers))

        let entryCount = json["responseData"]["entries"].count
        for var i in 0 ..< entryCount
        {
            print (json["responseData"]["entries"][i]["title"].data as! String)
        }
    }
    catch {
        print ("Error parsing JSON from server.")
    }
}
catch {
    print ("Error retrieving data from server.")
}
```

Process downloaded data

```
print ("After download...")
```

Grand Central Dispatch (GCD)

- Any worker can dispatch code to another worker to execute



Example

```
print ("Before download...")

    do {
        let data = try Data (contentsOf: googleUrl!)


            do {
                let json = JSONDataAccessor(try JSONSerialization.jsonObject(with: data,
                    options:.mutableContainers))

                let entryCount = json["responseData"]["entries"].count
                for var i in 0 ..< entryCount
                {
                    print (json["responseData"]["entries"][i]["title"].data as! String)
                }
            }
            catch {
                print ("Error parsing JSON from server.")
            }

    }
    catch {
        print ("Error retrieving data from server.")
    }

print ("After download...")
```

Grand Central Dispatch (GCD)

Example

```
print ("Before dispatch...")
DispatchQueue.global(qos: .background).async {
    do {
        let data = try Data (contentsOf: googleUrl!)


            do {
                let json = JSONDataAccessor(try JSONSerialization.jsonObject(with: data,
                    options:.mutableContainers))

                let entryCount = json["responseData"]["entries"].count
                for var i in 0 ..< entryCount
                {
                    print (json["responseData"]["entries"][i]["title"].data as! String)
                }
            }
            catch {
                print ("Error parsing JSON from server.")
            }

    }
    catch {
        print ("Error retrieving data from server.")
    }
}
print ("After dispatch...")
```

Grand Central Dispatch (GCD)

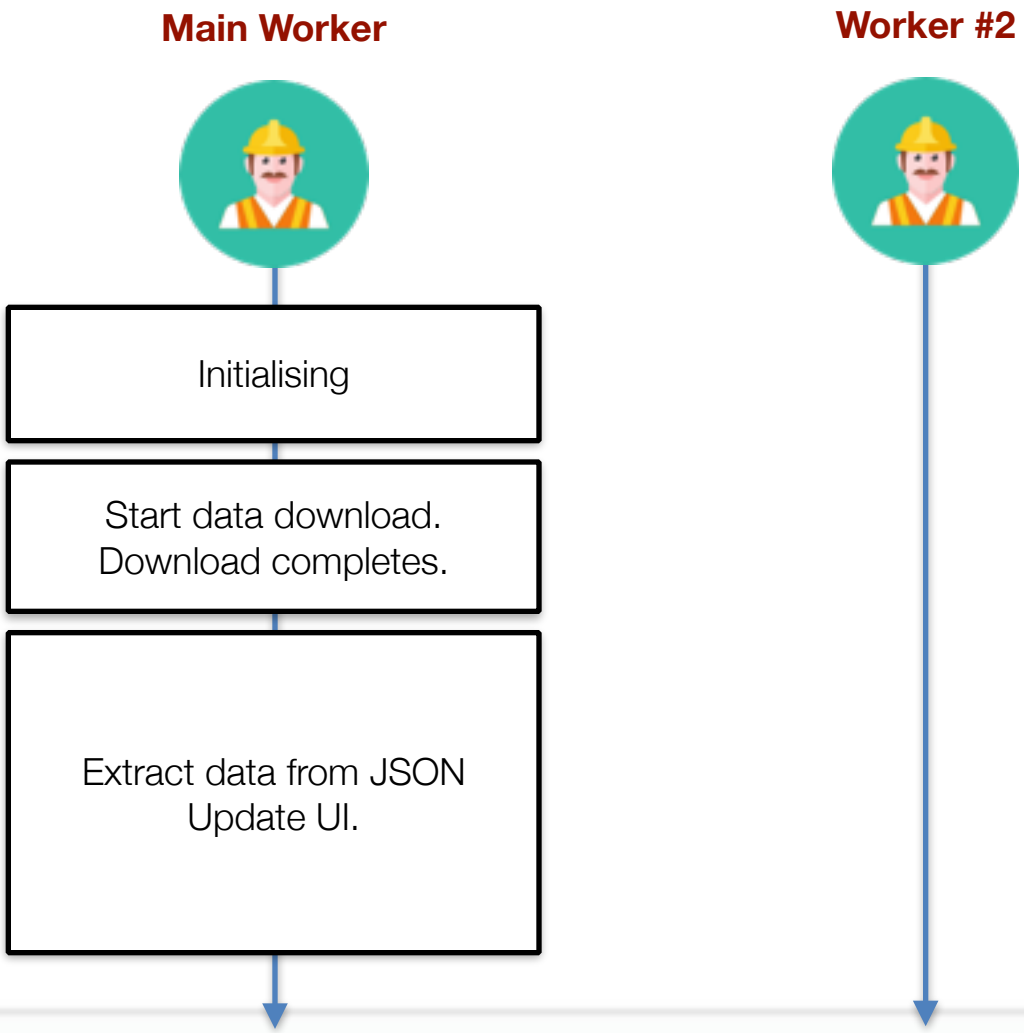
Example

```
print ("Before dispatch...")
DispatchQueue.global(qos: .background).async {
    do {
        let data = try Data (contentsOf: googleUrl!)


            do {
                let json = JSONDataAccessor(try JSONSerialization.jsonObject(with: data,
                    options:.mutableContainers))

                let entryCount = json["responseData"]["entries"].count
                for var i in 0 ..< entryCount
                {
                    print (json["responseData"]["entries"][i]["title"].data as! String)
                }
            }
            catch {
                print ("Error parsing JSON from server.")
            }

    }
    catch {
        print ("Error retrieving data from server.")
    }
}
print ("After dispatch...")
```

This whole code in blue now runs in the background!

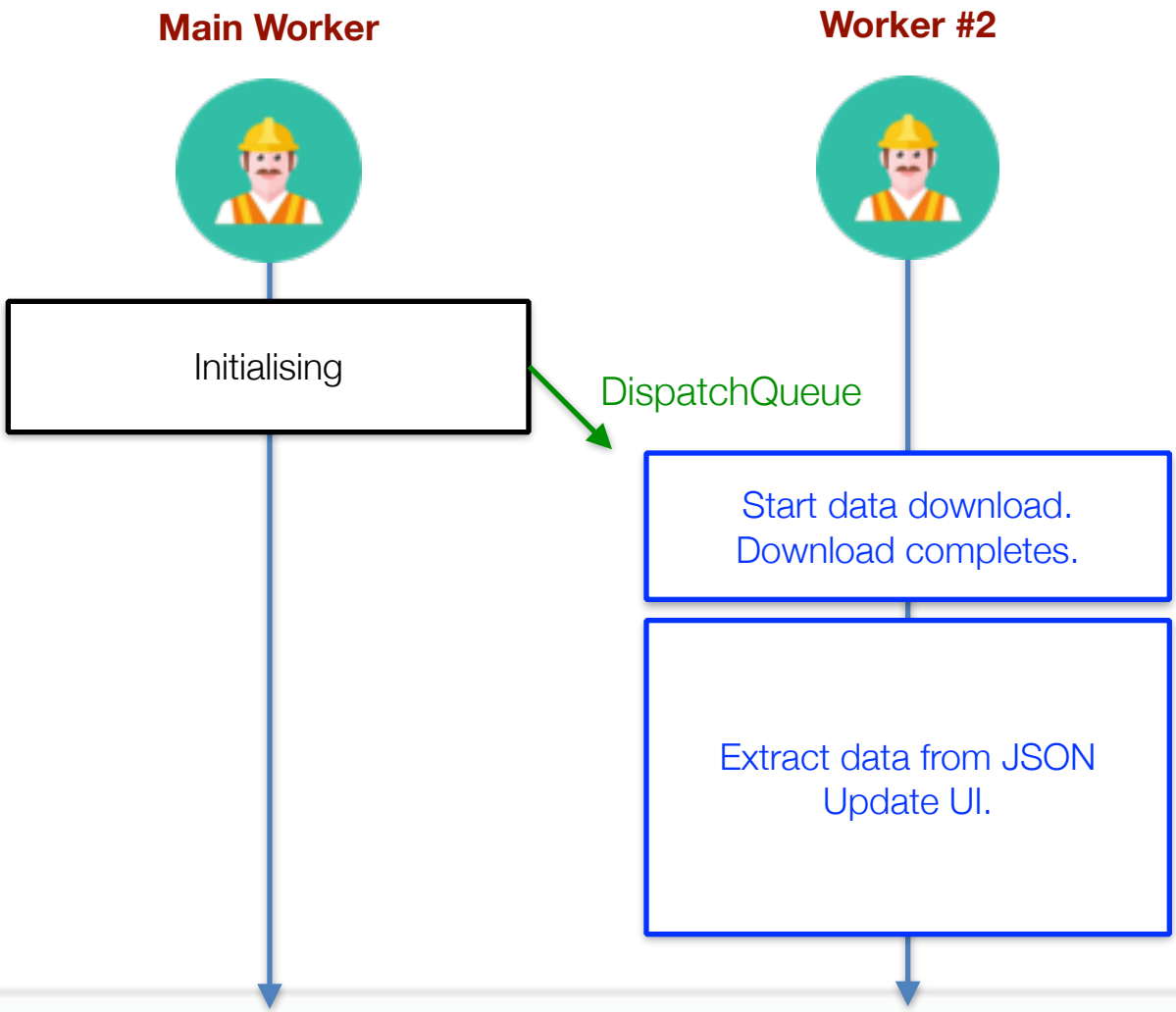
Grand Central Dispatch (GCD)

- Dispatch long running task to another worker



Grand Central Dispatch (GCD)

- Dispatch long running task to another worker



Grand Central Dispatch (GCD)

Example

```
print ("Before dispatch...")
DispatchQueue.global(qos: .background).async {
    do {
        let data = try Data (contentsOf: googleUrl!)


            do {
                let json = JSONDataAccessor(try JSONSerialization.jsonObject(with: data,
                    options:.mutableContainers))

                let entryCount = json["responseData"]["entries"].count
                for var i in 0 ..< entryCount
                {
                    print (json["responseData"]["entries"][i]["title"].data as! String)
                }
            }
            catch {
                print ("Error parsing JSON from server.")
            }

    }
    catch {
        print ("Error retrieving data from server.")
    }
}
print ("After dispatch...")
```

This whole code in blue now runs in the background!

Grand Central Dispatch (GCD)

Example

```
print ("Before dispatch...")
DispatchQueue.global(qos: .background).async {
    do {
        let data = try Data (contentsOf: googleUrl!)

        DispatchQueue.main.async {
            do {
                let json = JSONDataAccessor(try JSONSerialization.jsonObject(with: data,
                    options:.mutableContainers))

                let entryCount = json["responseData"]["entries"].count
                for var i in 0 ..< entryCount
                {
                    print (json["responseData"]["entries"][i]["title"].data as! String)
                }
            }
            catch {
                print ("Error parsing JSON from server.")
            }
        }
    }
    catch {
        print ("Error retrieving data from server.")
    }
}
print ("After dispatch...")
```

Grand Central Dispatch (GCD)

Example

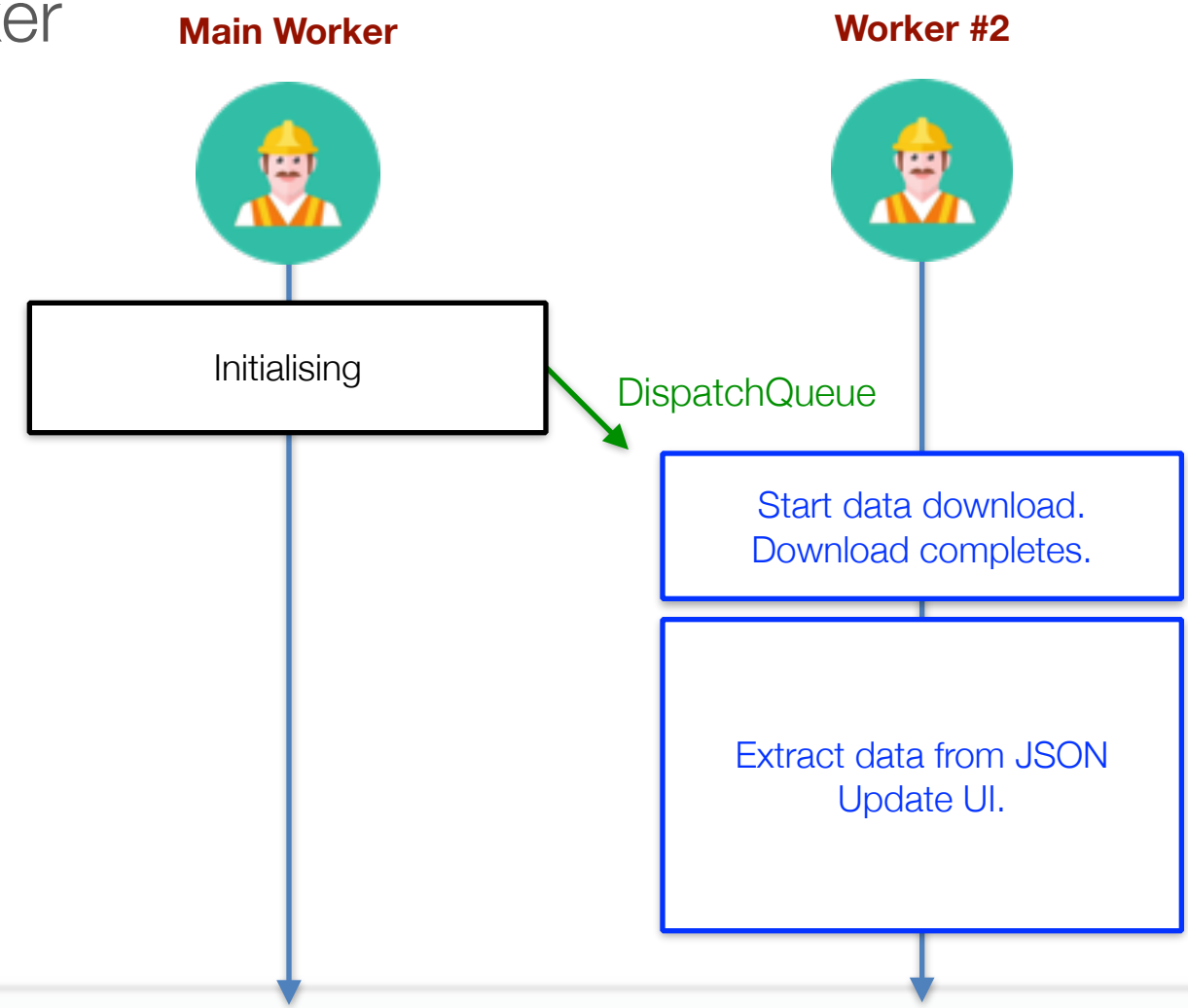
```
print ("Before dispatch...")
DispatchQueue.global(qos: .background).async {
    do {
        let data = try Data (contentsOf: googleUrl!)

        DispatchQueue.main.async {
            do {
                let json = JSONDataAccessor(try JSONSerialization.jsonObject(with: data,
                    options:.mutableContainers))

                let entryCount = json["responseData"]["entries"].count
                for var i in 0 ..< entryCount
                {
                    print (json["responseData"]["entries"][i]["title"].data as! String)
                }
            }
            catch {
                print ("Error parsing JSON from server.")
            }
        }
    }
    catch {
        print ("Error retrieving data from server.")
    }
}
print ("After dispatch...")
```

Only this blue part runs in the background

Now this black part runs in the UI thread, after the download is complete!

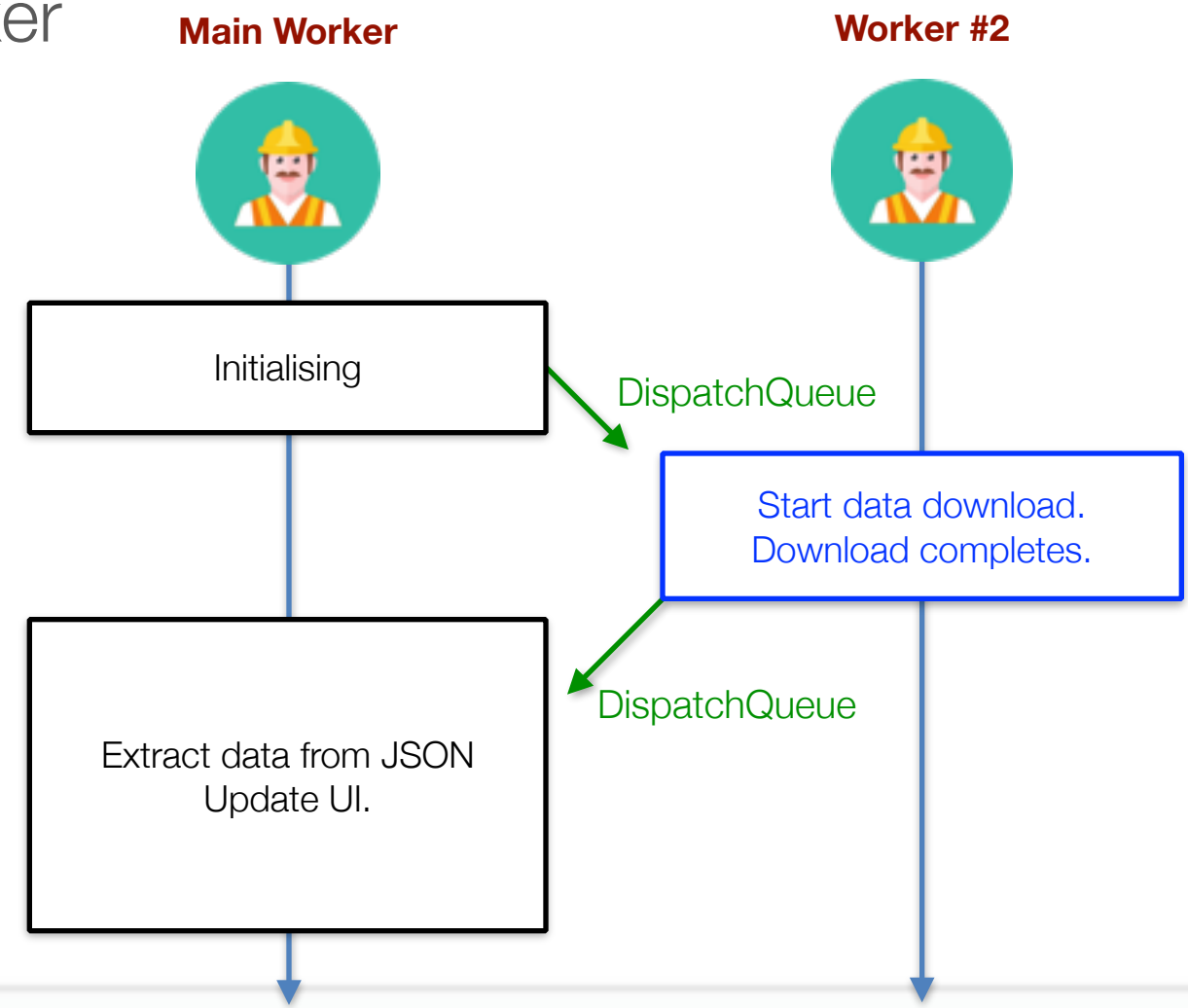
Grand Central Dispatch (GCD)

- The other worker dispatches UI task to Main Worker



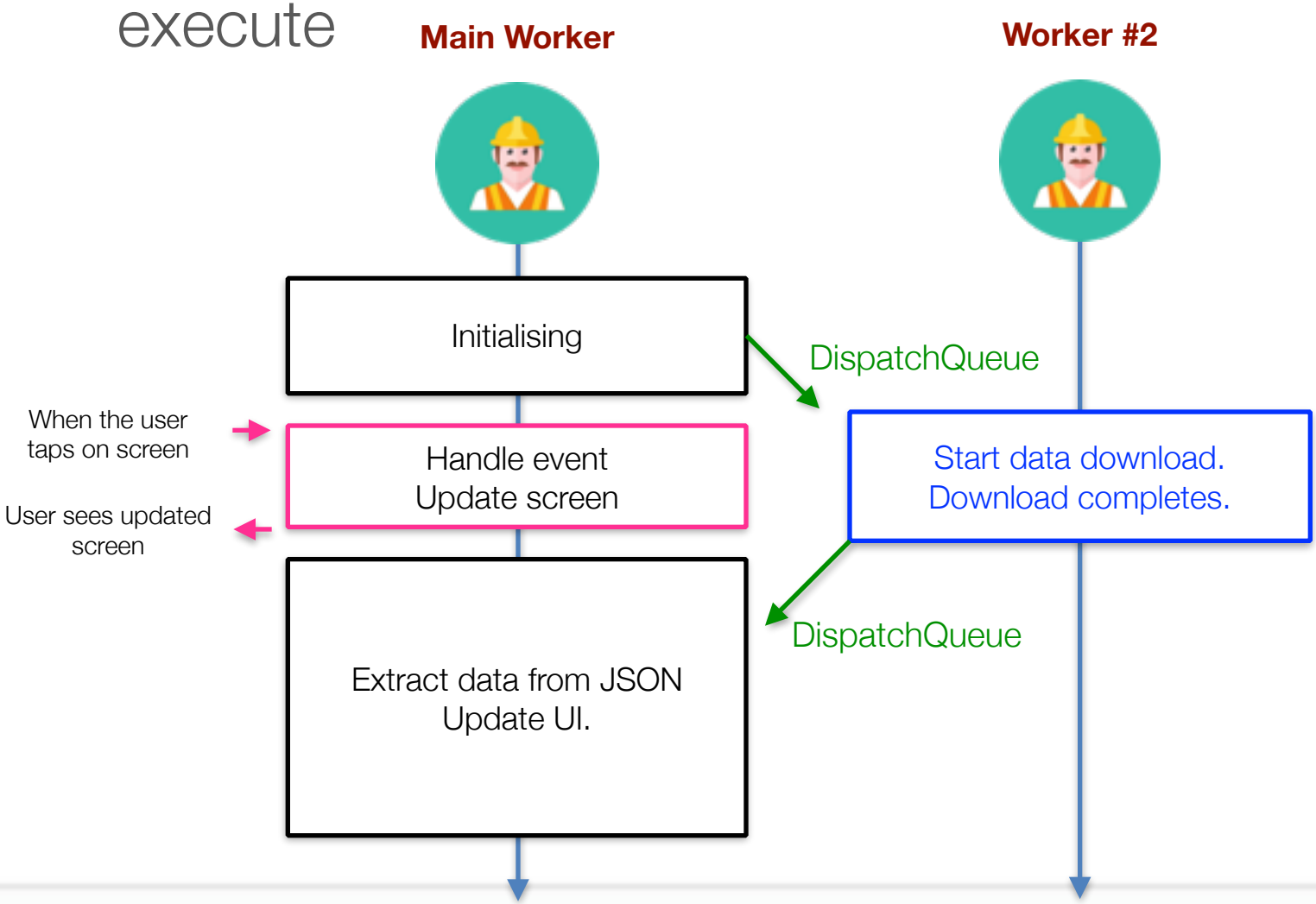
Grand Central Dispatch (GCD)

- The other worker dispatches UI task to Main Worker



Grand Central Dispatch (GCD)

- Any worker can dispatch code to another worker to execute



Grand Central Dispatch (GCD)

Example

```
print ("Before dispatch...")
DispatchQueue.global(qos: .background).async {
    do {
        let data = try Data (contentsOf: googleUrl!)

        DispatchQueue.main.async {
            do {
                let json = JSONDataAccessor(try JSONSerialization.jsonObject(with: data,
                    options:.mutableContainers))

                let entryCount = json["responseData"]["entries"].count
                for var i in 0 ..< entryCount
                {
                    print (json["responseData"]["entries"][i]["title"].data as! String)
                }
            }
            catch {
                print ("Error parsing JSON from server.")
            }
        }
    }
    catch {
        print ("Error retrieving data from server.")
    }
}
print ("After dispatch...")
```

Only this download part runs in the background

Now this part runs in the UI thread, after the download is complete!

Grand Central Dispatch (GCD)

- Easier, because you do not need to:
  - write code to create new threads,
  - synchronise the sequential code across multiple threads

- You can use GCD to run long running code asynchronously, not necessarily loading from web services. For example:
  - Complex mathematical computations
  - Long running queries on SQLite
  - AI for board games

Grand Central Dispatch (GCD)

- References:
  - https://developer.apple.com/reference/dispatch

Consuming Web Services

Cloud Persistent Storage Using Firebase

Firebase

- A database stored in the cloud.
  - Hosted service.
  - Fast to build.
  - Cross-platform.
  - Authenticate with Facebook, Twitter, Google, etc.

firebase.google.com

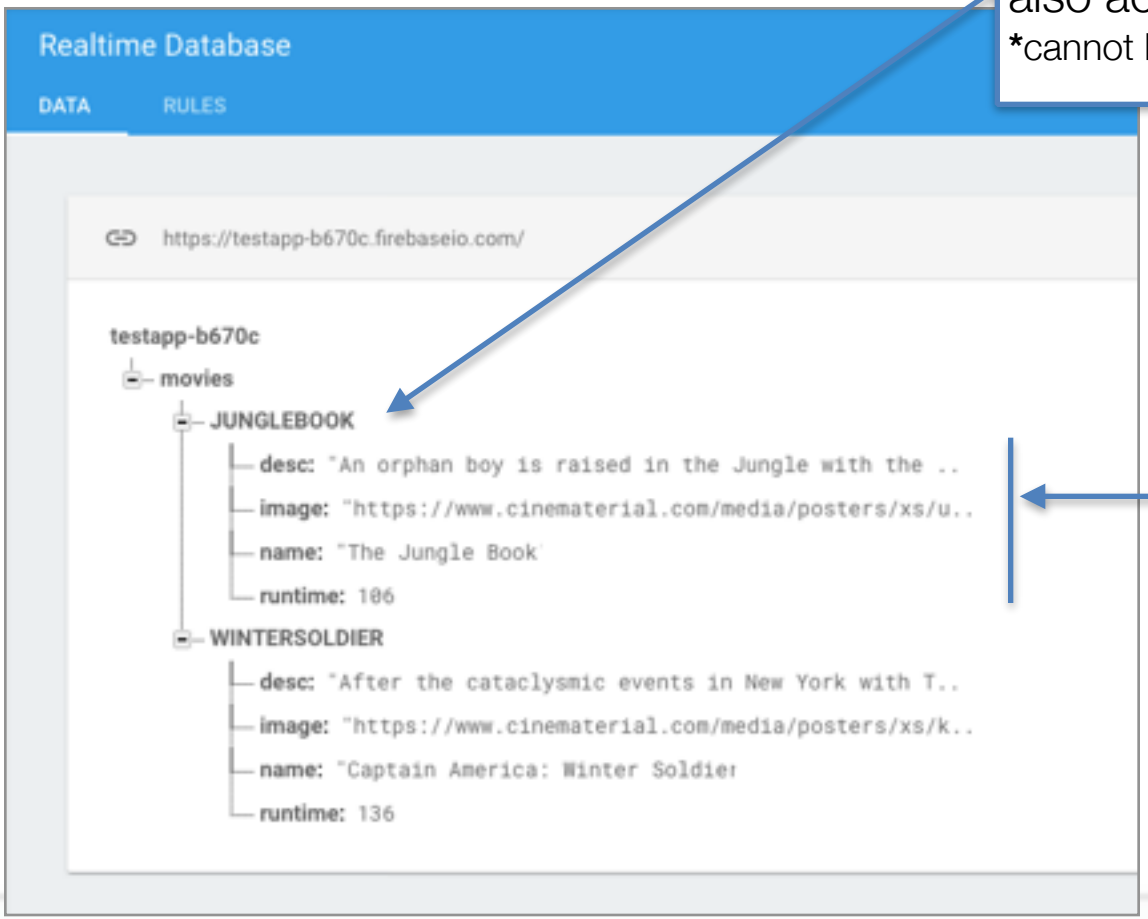
Firebase

- Pre-requisites
  - You need:
    - Xcode 8.0 or above
    - Xcode project with the Bundle ID
    - For Cloud Messaging:
      - A physical iOS device
      - APNs certificate with Push Notifications enabled

    https://firebase.google.com/docs/ios/setup

Firebase - Basic Functions

- Data format - JSON-style tree structure
- Like folders. No “tables”!



the record’s **key**
also acts as a folder name
*cannot be modified once created

the record’s **fields**

Firebase - Basic Functions

- Save data (example)

eg.: https://myapp2016-1.firebase.io/movies/JUNGLEBOOK/

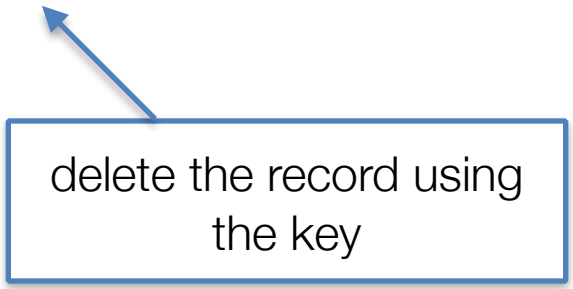
```
self.ref.child("movies/\(movie.movieID)/").setValue(
    [
    "name" : movie.movieName,
    "desc" : movie.movieDesc,
    "runtime" : movie.runtime,
    "image" : movie.imagePath])
```

saving the fields

Firebase - Basic Functions

- Delete data (example)

```
self.ref.child("movies/\(movie.movieID)/").
    removeValue()
```



Firebase - Basic Functions

- Query data (example)

load **all** records in movies **asynchronously**

```
let movieRef = self.ref.child("movies/")

movieRef.observeSingleEventOfType(.Value,

    ...

    )
```

Firebase - Basic Functions

- Query data (example)

```
let movieRef = self.ref.child("movies/")

movieRef.observeSingleEventOfType(.Value,
    withBlock:
     { (snapshot) in

         for record in snapshot.children
         {
             print ("\(record.key!): " +
                 "\(record.value!["name"]) " +
                 "\(record.value!["runtime"])");
         }
})
```

when done, loop and print each record

Firebase - Basic Functions

- Sort data (example)

```
let movieRef = self.ref.child("movies/")

movieRef
```

Let's move this here for better readability.

```
    .observeSingleEventOfType(.Value,
    withBlock:
     { (snapshot) in
         for record in snapshot.children
         {
             print ("\(record.key!): " +
                 "\(record.value!!["name"]) " +
                 "\(record.value!!["runtime"])");
         }
    })
```

Firebase - Basic Functions

- Sort data (example)

```
let movieRef = self.ref.child("movies/")

movieRef
    .queryOrderedByChild("runtime")


    .observeSingleEventOfType(.Value,
    withBlock:
     { (snapshot) in
         for record in snapshot.children
         {
             print ("\(record.key!): " +
                 "\(record.value!!["name"]) " +
                 "\(record.value!!["runtime"])");
         }
    })
```

sorts all movies by runtime (ascending order)

Firebase - Basic Functions

- Filter data (example)

```
let movieRef = self.ref.child("movies/")

movieRef
    .queryOrderedByChild("runtime")
    .queryEqualToValue(95)


    .observeSingleEventOfType(.Value,
    withBlock:
     { (snapshot) in
         for record in snapshot.children
         {
             print ("\(record.key!): " +
                 "\(record.value!!["name"]) " +
                 "\(record.value!!["runtime"])");
         }
    })
```

picks movies whose runtime = 95

Firebase - Basic Functions

- Filter data (example)

```
let movieRef = self.ref.child("movies/")

movieRef
    .queryOrderedByChild("runtime")
    .queryStartingAtValue(95)
    .queryEndingAtValue(110)

    .observeSingleEventOfType(.Value,
    withBlock:
     { (snapshot) in
         for record in snapshot.children
         {
             print ("\(record.key!): " +
                 "\(record.value!!["name"]) " +
                 "\(record.value!!["runtime"])");
         }
    })
```

picks movies whose runtime >= 95 and <= 110

Firebase - Basic Functions

- Filter data (example)

```
let movieRef = self.ref.child("movies/")

movieRef
    .queryOrderedByChild("runtime")
    .queryStartingAtValue(95)
    .queryEndingAtValue(110)
    .queryLimitedToFirst(10)
    .observeSingleEventOfType(.Value,
    withBlock:
     { (snapshot) in
         for record in snapshot.children
         {
             print ("\(record.key!): " +
                 "\(record.value!!["name"]) " +
                 "\(record.value!!["runtime"])");
         }
    })
```

limits the number of records to first 10.

You can also use **queryLimitedToLast**

Firebase

- Limitations:
  - Filtering on 1 field at any one time.
  - Sorting only in ascending order
  - Using only the following conditions:
    - equalTo: =
    - startingAtValue: >=
    - endingAtValue: <=
  - Only limit results to:
    - the first *n*, or
    - the last *n*

Firebase

- Good for:
  - Real-time data used in games / geospatial apps
  - Real-time chronological data such as news / blogs chats / comments / diaries / notes
  - Social media sharing

- Not so good for:
  - Analytics (of your data)
  - Free-text searching engine
    (due to limitations in the off-the-shelf query APIs)

Firebase - Basic Functions

- More on Firebase iOS API:
  - Get Started

    https://firebase.google.com/docs/database/ios/start

  - Saving / Retrieve Data

    https://firebase.google.com/docs/database/ios/save-data

    https://firebase.google.com/docs/database/ios/retrieve-data

  - API Reference
  - https://firebase.google.com/docs/reference/ios/firebasedatabase/

  - Examples

    https://github.com/firebase/quickstart-ios

Summary

- Understand JSON framework
- Fetch and Parse JSON feeds
- URLSession
- Grand Central Dispatch
- Cloud Persistent Storage