



Beginning iOS 10 Application Development

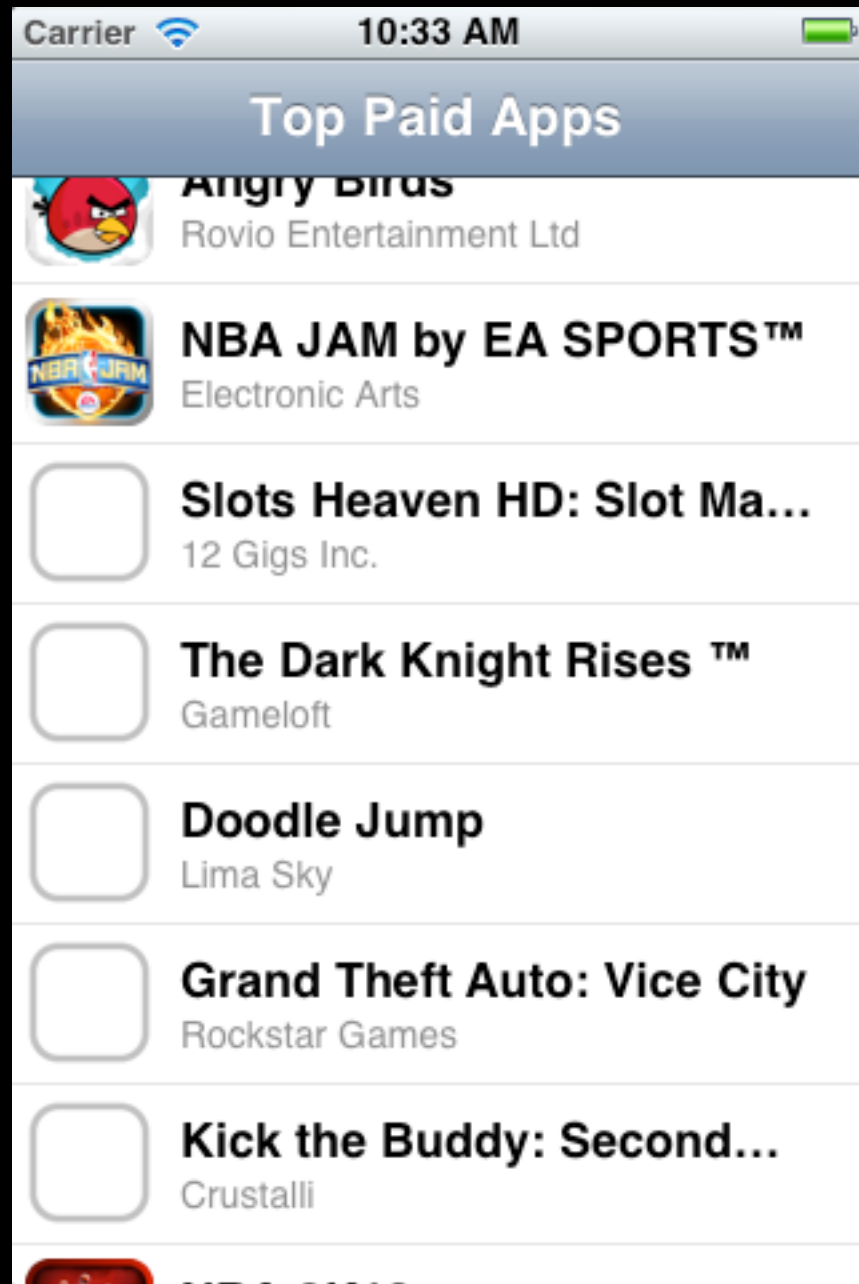
Concurrency & Networking

Yanping Zhao
Nov. 2016

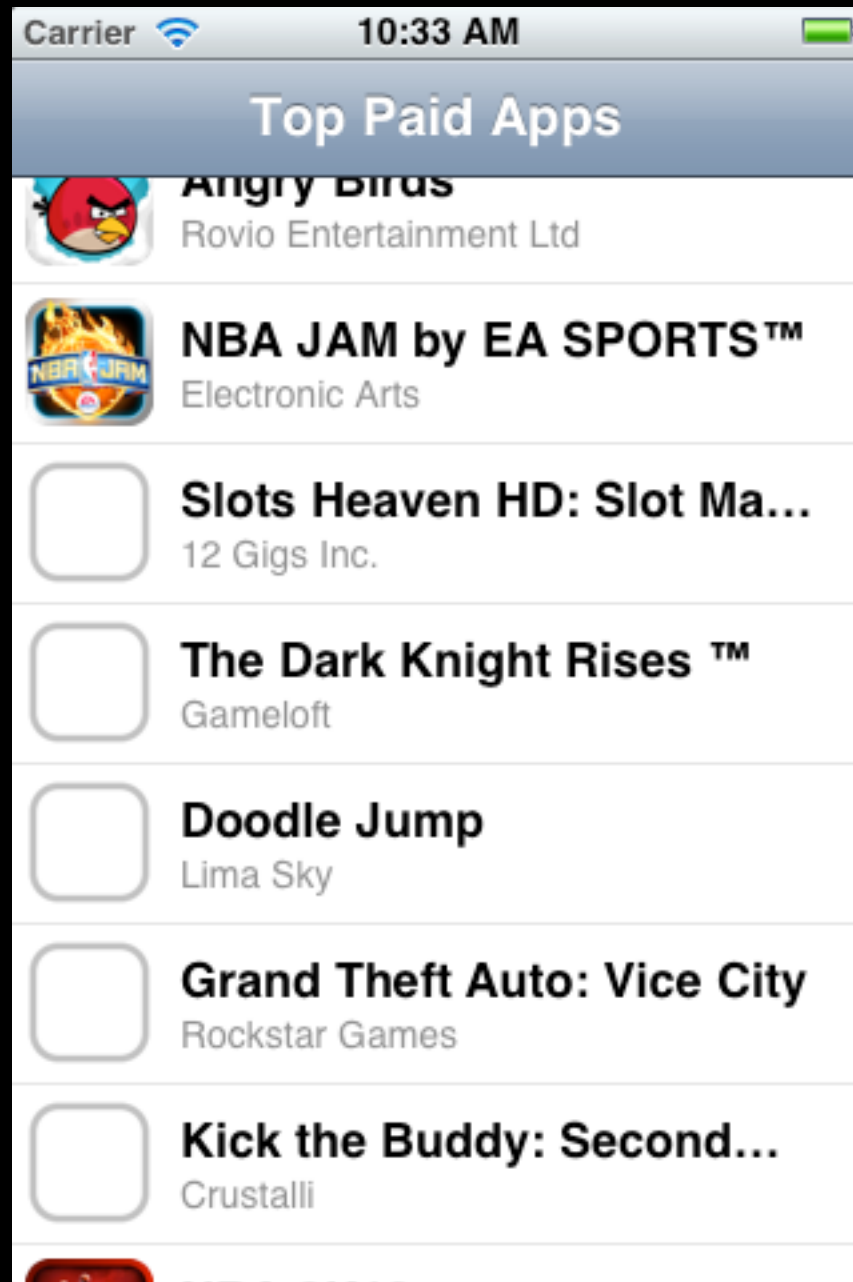
Networking in Model

Where to Do Networking?

- Views
- Controllers
- Model

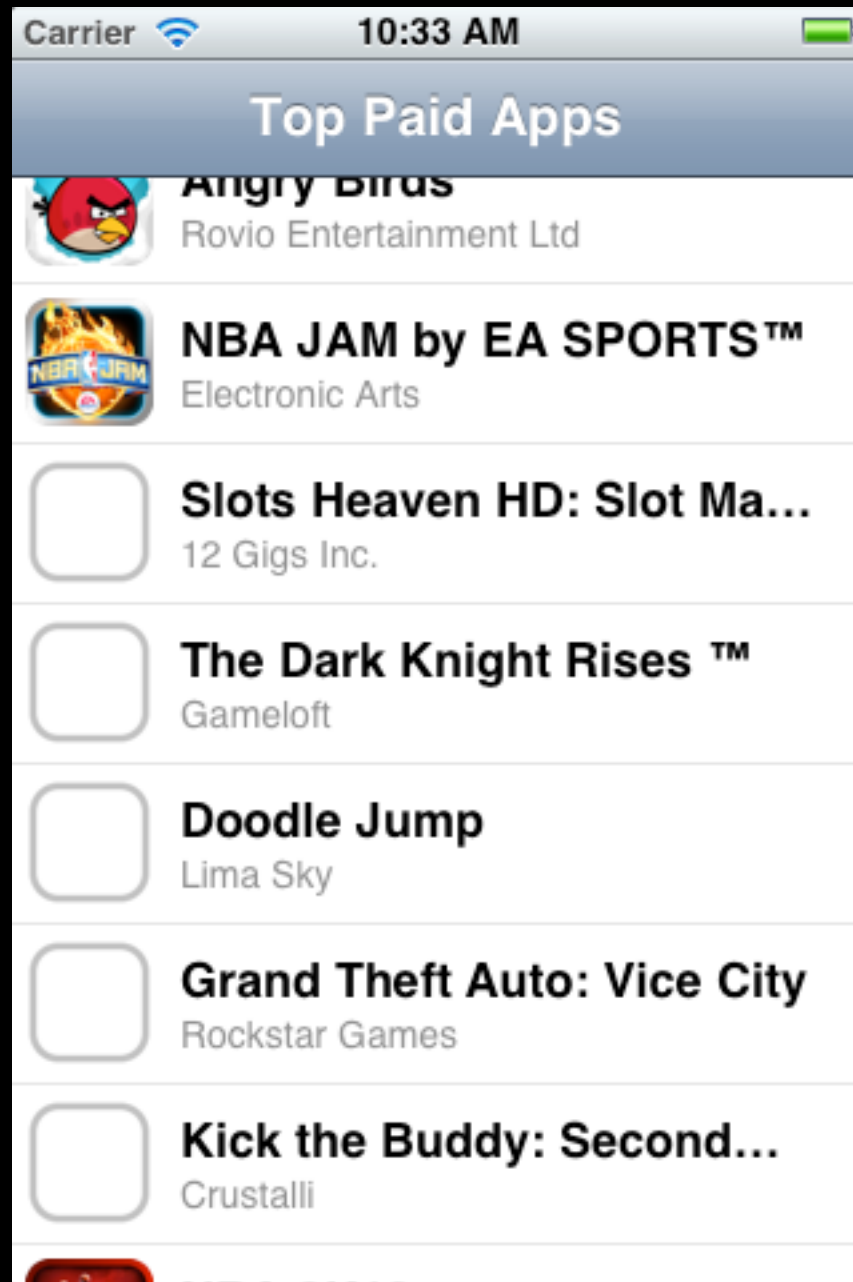


Where to Do Networking?



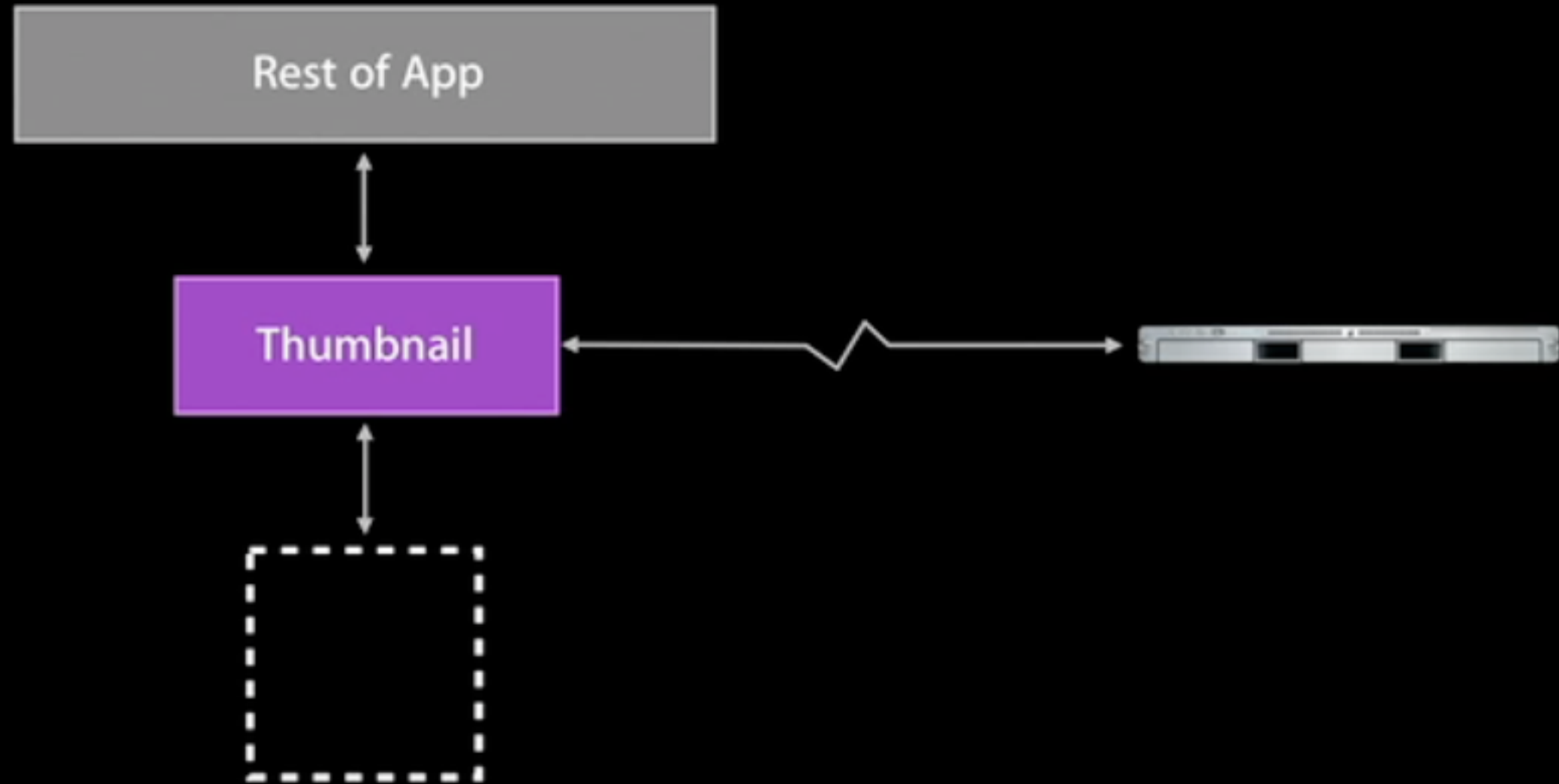
- ~~Views~~
- Controllers
- Model

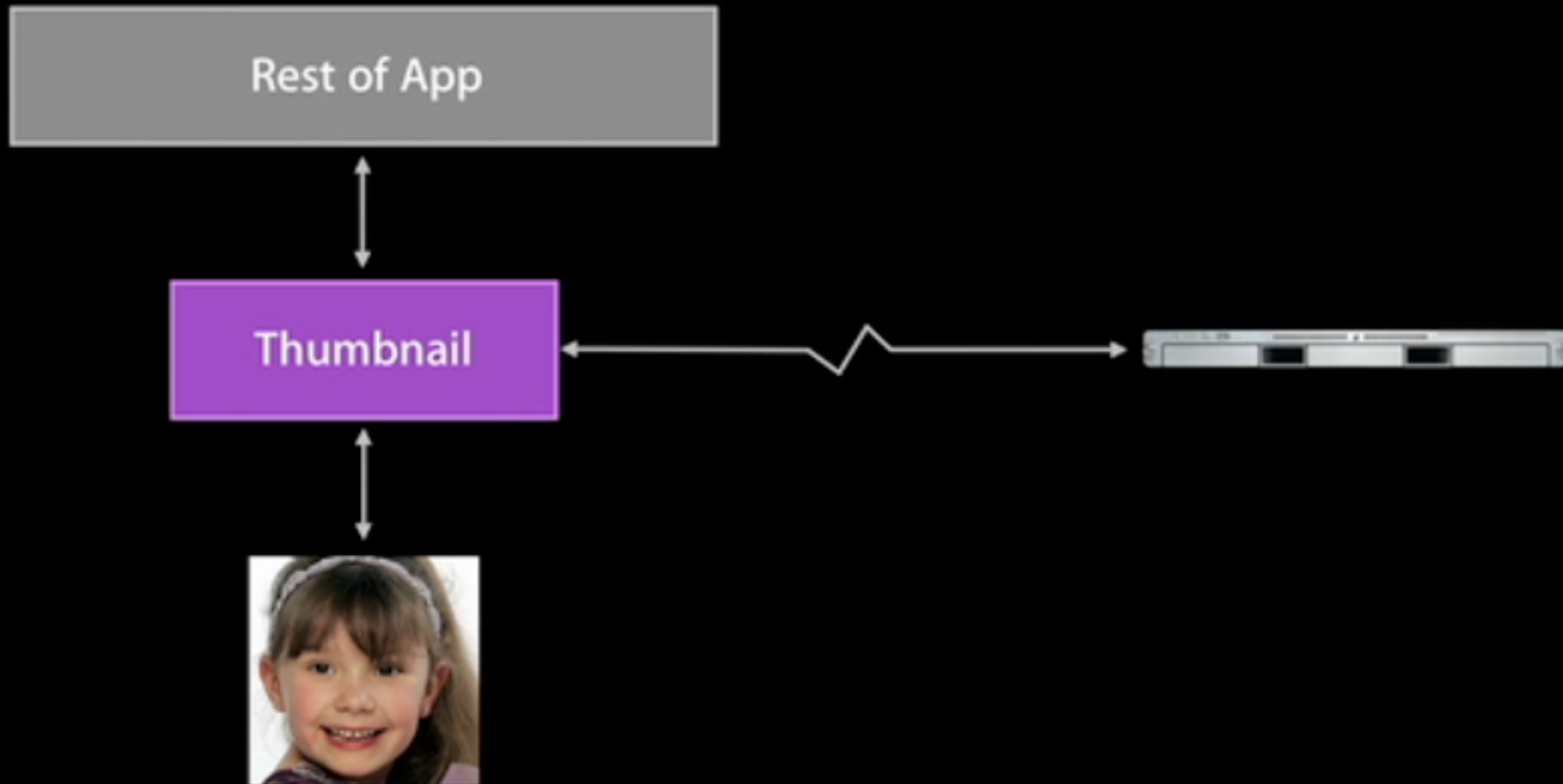
Where to Do Networking?



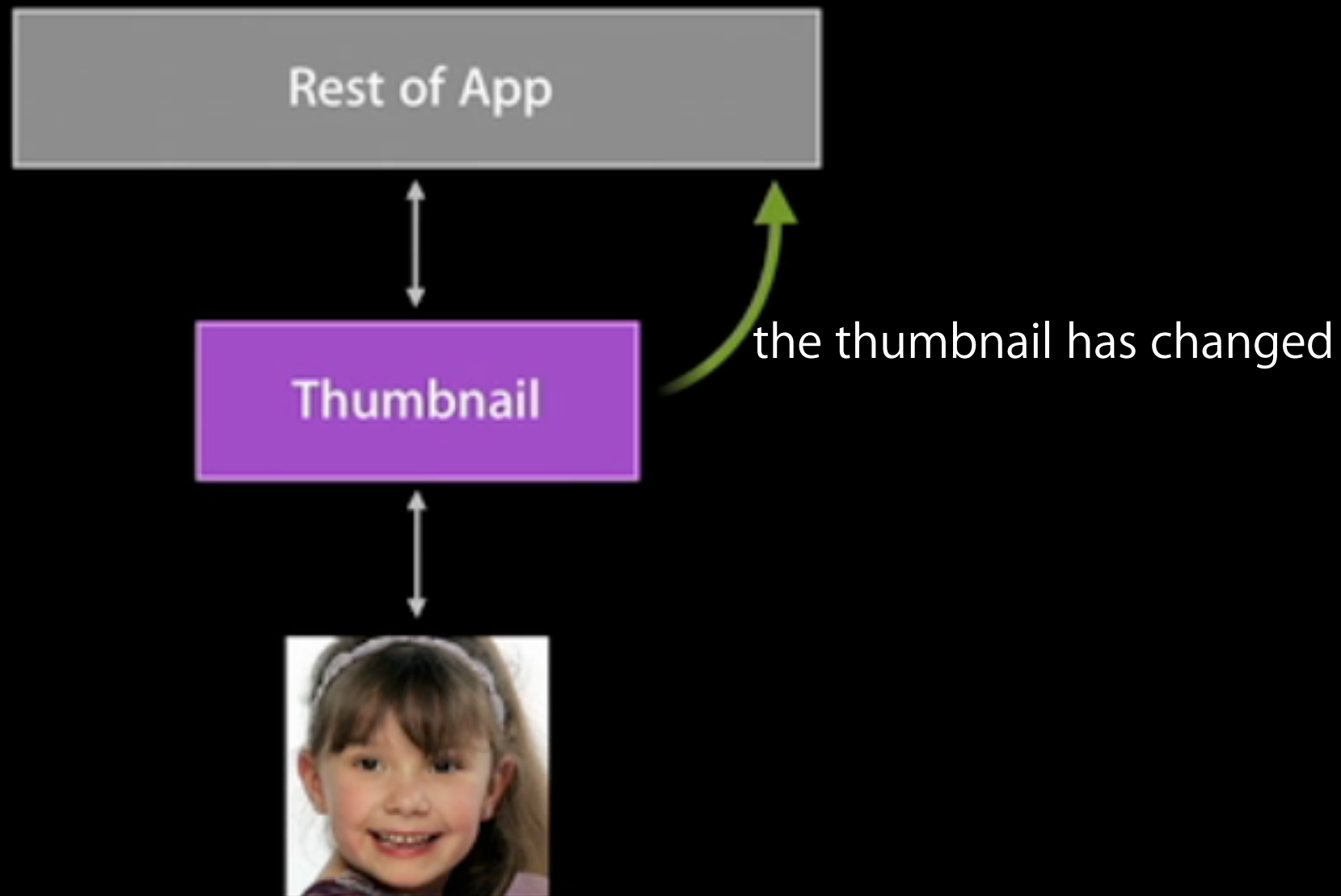
- ~~Views~~
- ~~Controllers~~
- Model

Model Example

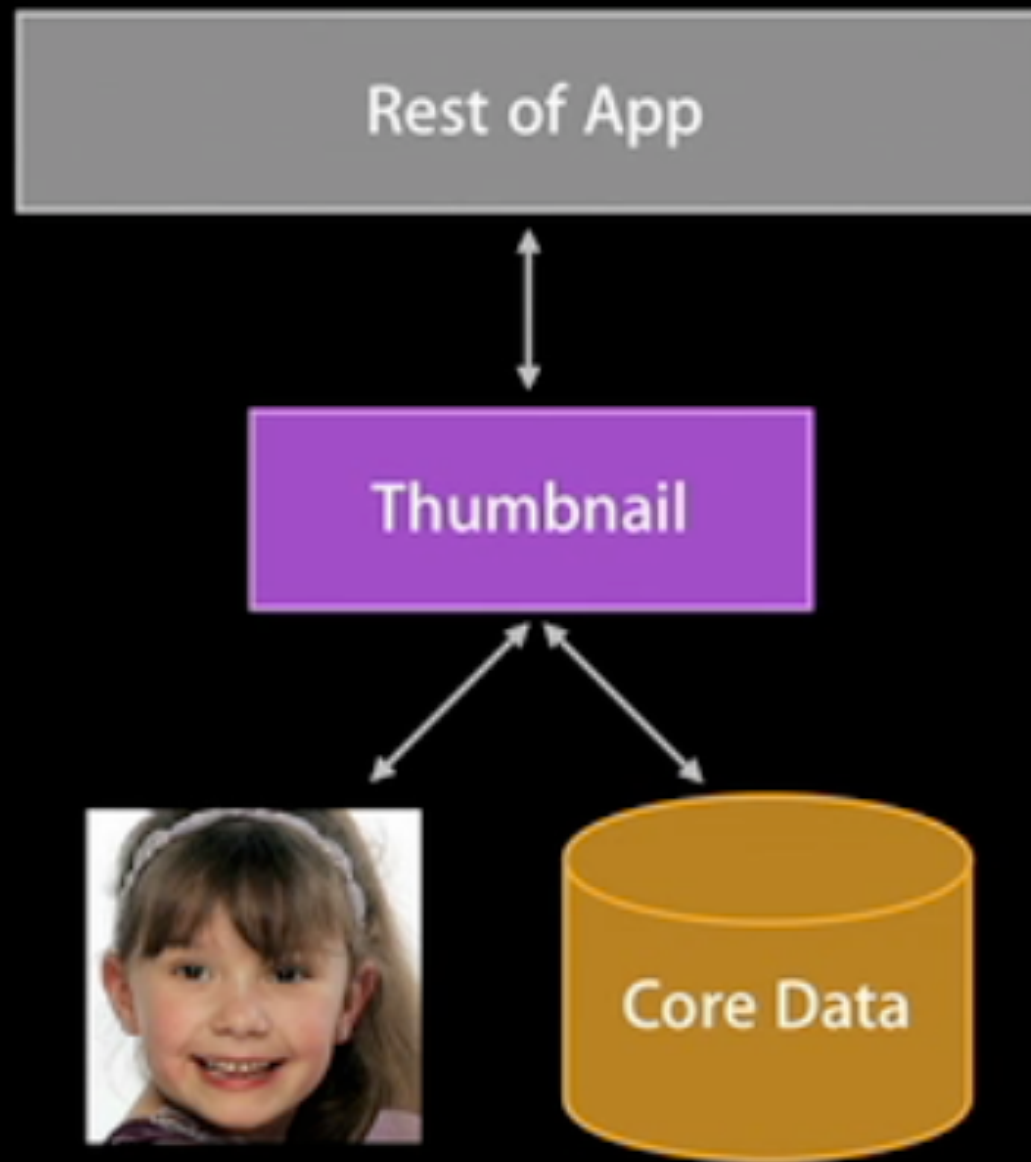




Model Example



Model Example



Notification Mechanisms

Mechanism	1:N	Granularity
Delegation	NO	Fine
NSNotification	YES	Coarse
Key-Value Observing	YES	Fine

Networking in Model

Advantages

- Isolates networking
- Persistence
- External changes
- Testing

Asynchronous Networking Design Pattern

Asynchronous Design Pattern

- The main thread -- handles user interactions
- CPU intensive code or blocking code -- other threads
- Call-Callback pattern -- interactions with UI should all happen on main thread

Asynchronous Design Pattern

Main
Thread

Touch Event

User Event

Main Queue



The diagram illustrates the Asynchronous Design Pattern. It features a large orange rectangle representing the 'Main Queue'. To the left of this rectangle, the text 'Main Thread' is displayed. Inside the orange rectangle, at the top left, are two smaller green rectangles labeled 'Touch Event' and 'User Event'. Below these green rectangles, the text 'Main Queue' is written in white. This visualizes how events are queued and processed asynchronously by the main thread.

Asynchronous Design Pattern

Main
Thread

Touch Event

User Event

Main Queue

User-defined Queue

Asynchronous Design Pattern

Main
Thread

Touch Event

User Event

Main Queue

User Block {...}

User-defined Queue

Asynchronous Design Pattern

Main
Thread

Touch Event

User Event

Main Queue

User Block {...}

User Block {...}

User-defined Queue

Asynchronous Design Pattern

Main
Thread

Touch Event

User Event

Main Queue

User Block {...}

User Block {...}

User-defined Queue

CPU1

Asynchronous Design Pattern

Main
Thread

Main Queue

User Event

User Block {...}

User Block {...}

User-defined Queue

Touch Event

Asynchronous Design Pattern

Main
Thread

Main Queue

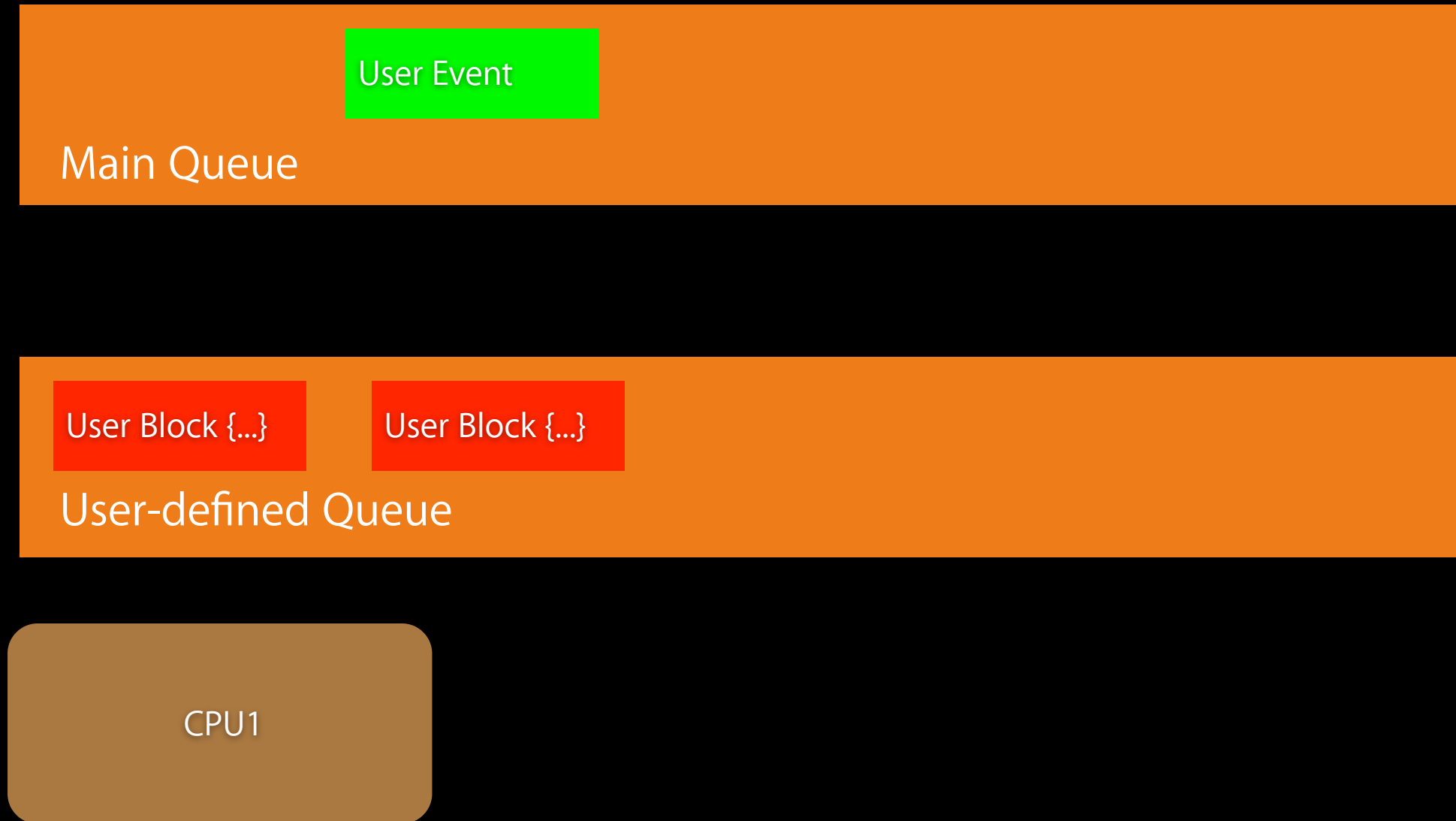
User Event

User Block {...}

User Block {...}

User-defined Queue

CPU1



Asynchronous Design Pattern

Main
Thread

Main Queue

User Event

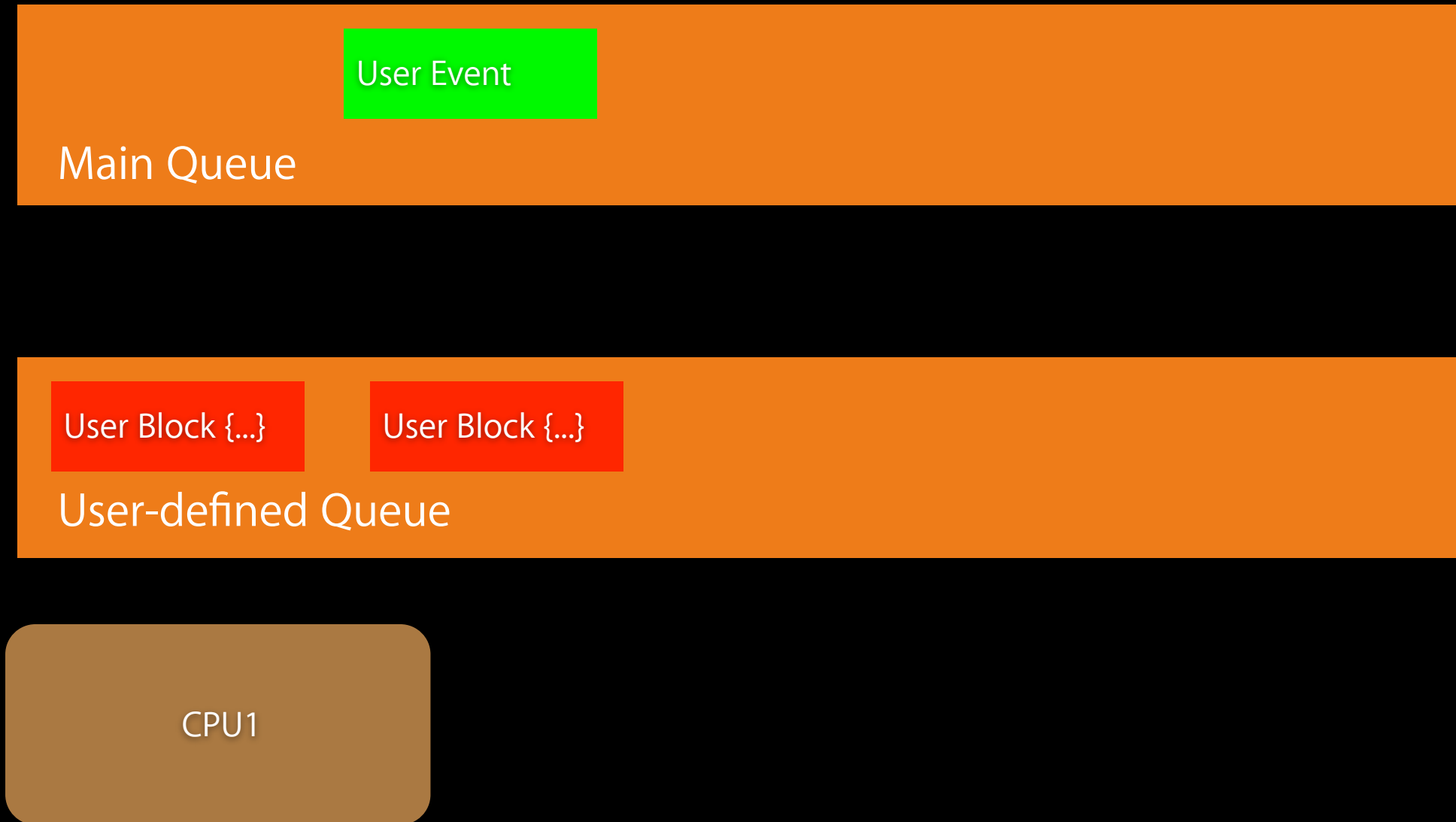
User Block {...}

User Block {...}

User-defined Queue

Worker
Thread1

CPU1



Asynchronous Design Pattern

Main
Thread

Main Queue

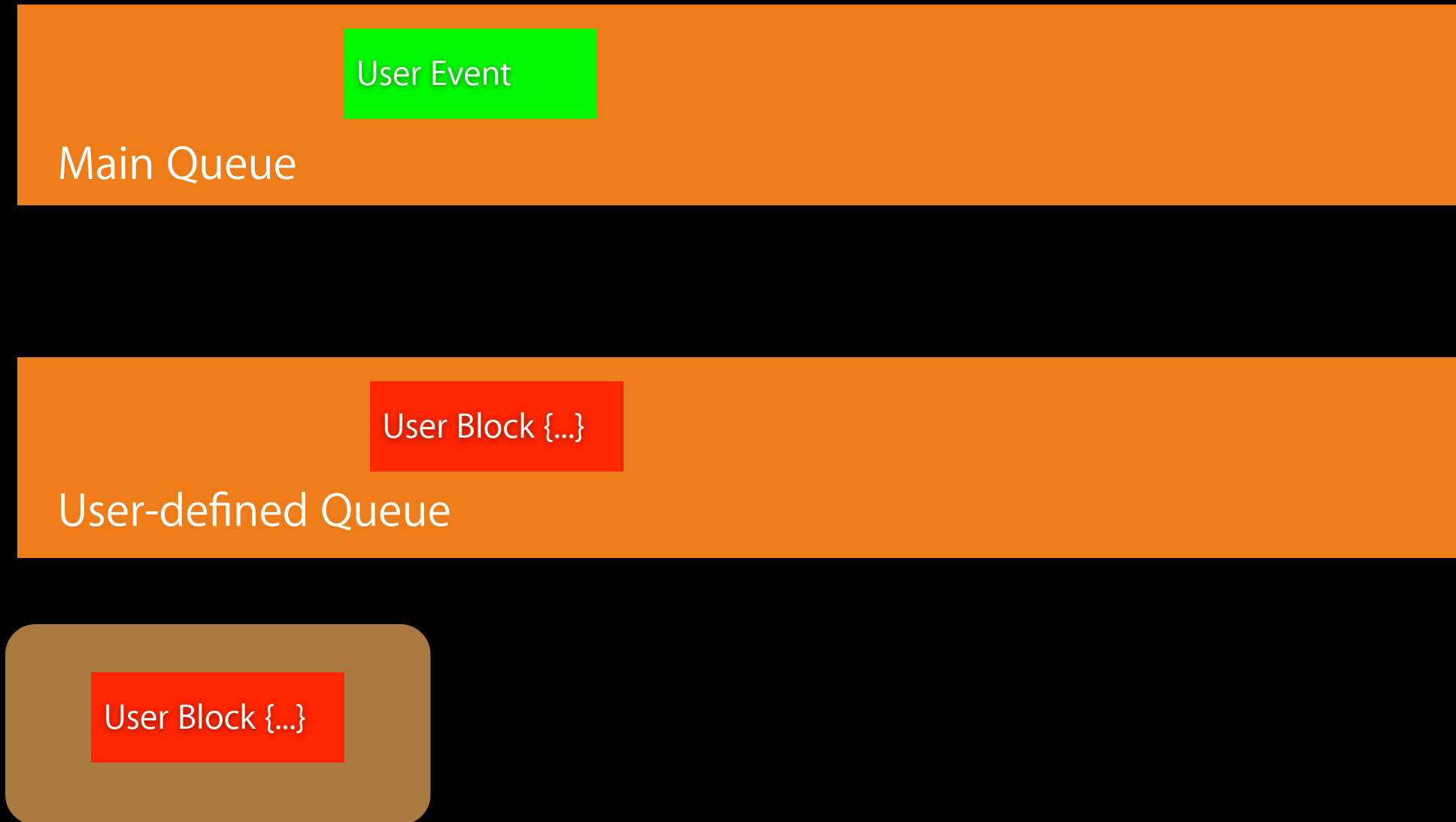
User Event

User-defined Queue

User Block {...}

Worker
Thread1

User Block {...}



Asynchronous Design Pattern

Main
Thread

Main Queue

User Event

User-defined Queue

User Block {...}

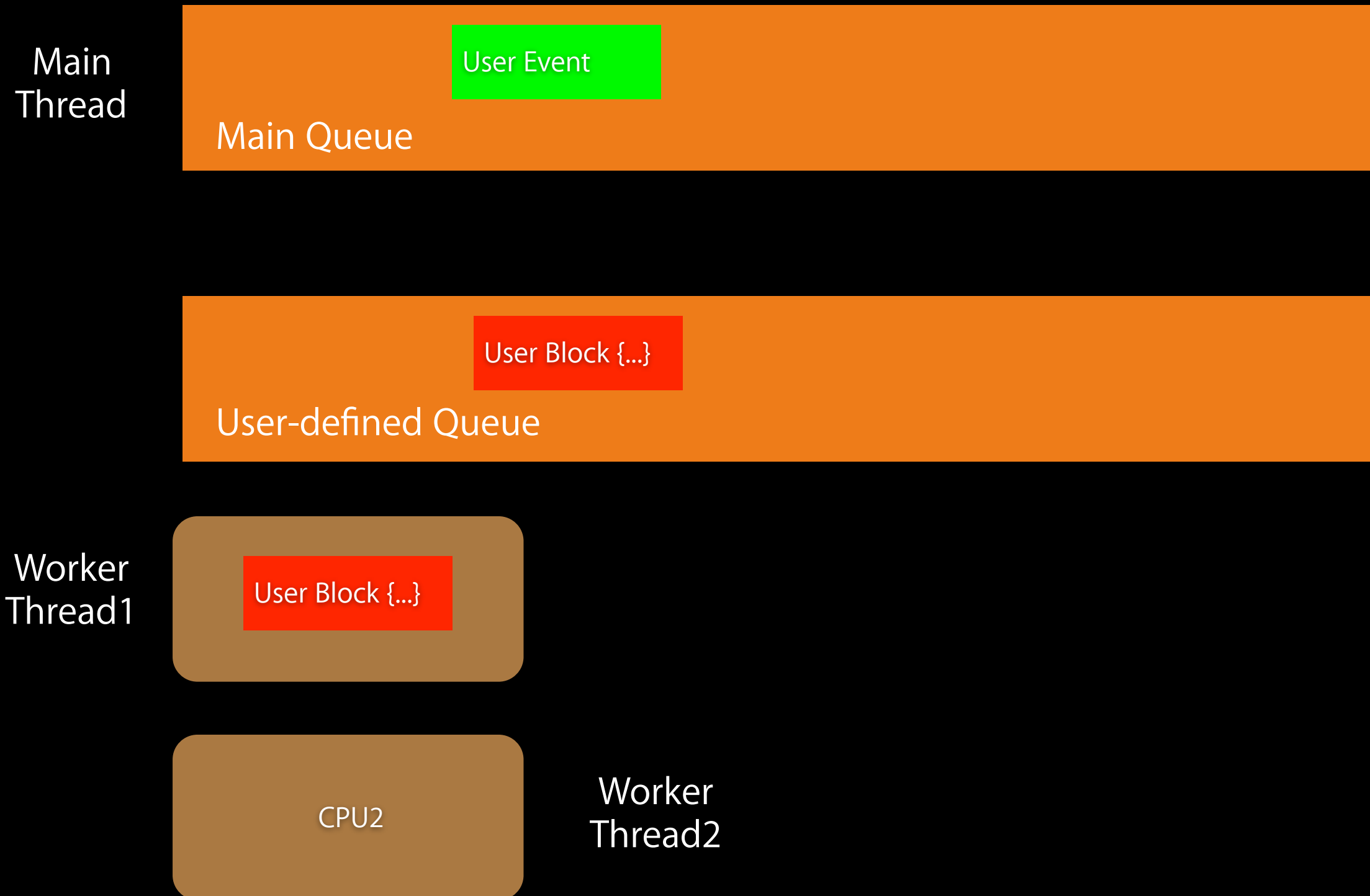
Worker
Thread1

User Block {...}

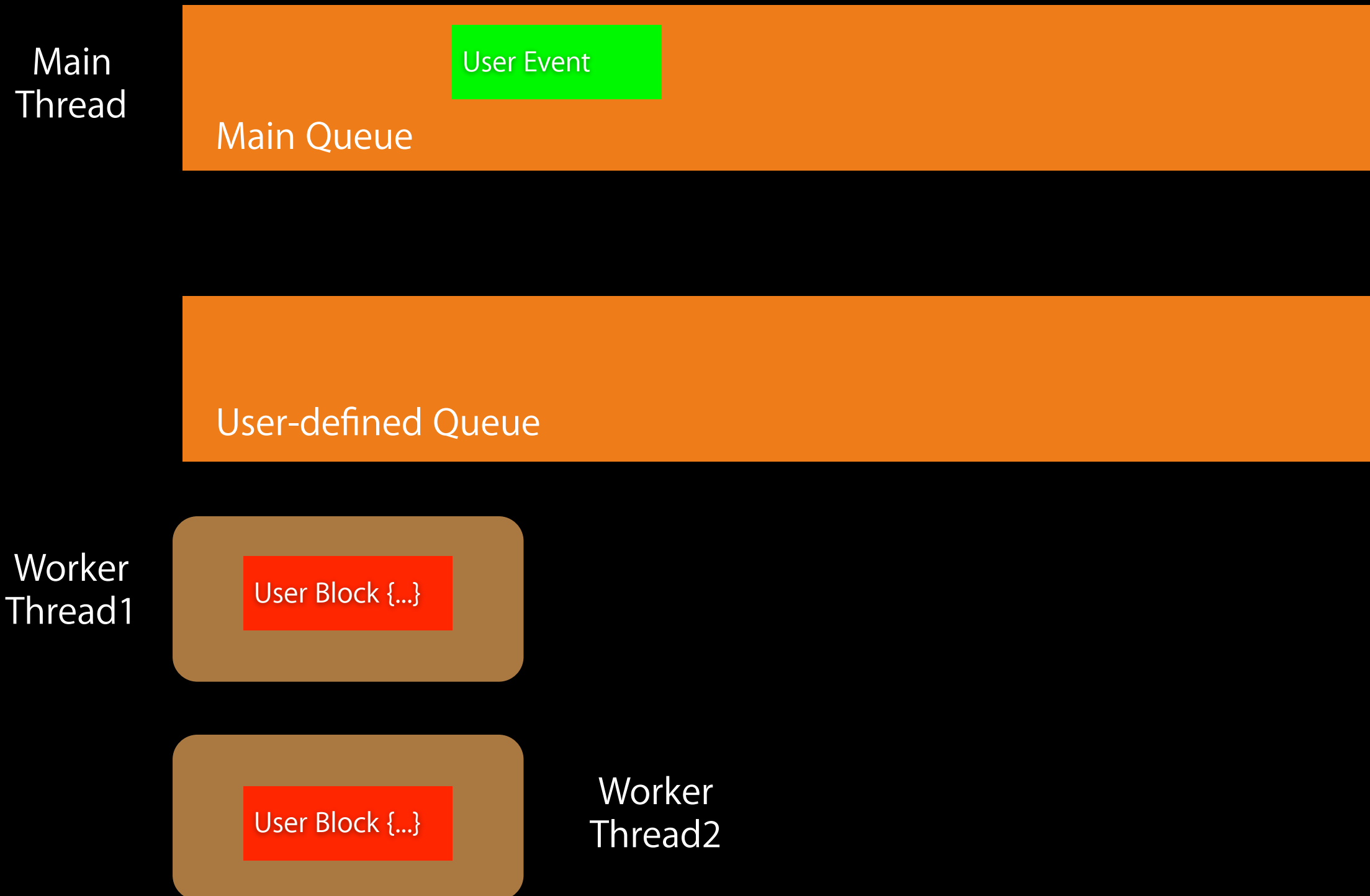
CPU2



Asynchronous Design Pattern



Asynchronous Design Pattern



Asynchronous Design Pattern

Main
Thread

Main Queue

User Event

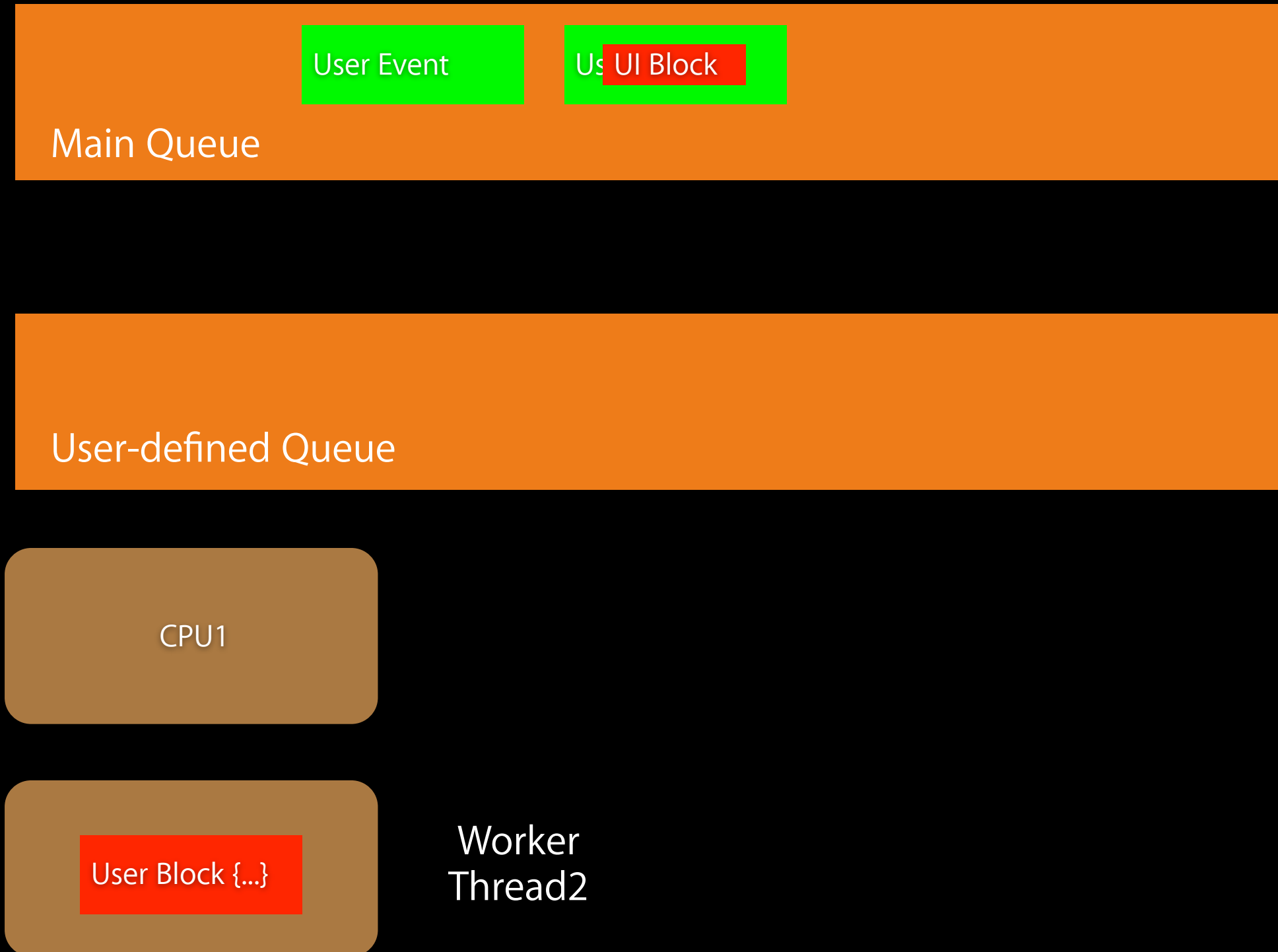
User UI Block

User-defined Queue

CPU1

User Block {...}

Worker
Thread2



Asynchronous Design Pattern

Main
Thread

Main Queue

User Event

User UI Block

User-defined Queue

CPU1

CPU2



Custom Queues

- NSOperationQueue
 - Objective-C API
 - in the Foundation framework
- Grand Central Dispatch Queues
 - C API
 - lower-level APIs than NSOperationQueue

Operation Queues

- First-in/first out thread-safe queues to execute Operations
- Higher-level APIs
- `OperationQueue.main` — the queue running on main threads

Grand Central Dispatch (GCD) Queues

- First-in/first out thread-safe queues to execute user tasks
- Serial queues
 - execute one task at a time on a distinct thread
 - created by the user
- Concurrent queues
 - execute multiple tasks concurrently on distinct threads
 - System-defined global queues: `DispatchQueue.global(qos)`
- The main dispatch queue
 - a global serial queue that executes tasks on the main thread
 - `DispatchQueue.main`


User Tasks -- Dispatch Blocks

- No arguments
- No return value

$\wedge\{\dots\}$

Asynchronous Networking with GCD

```
DispatchQueue.global(qos: .userInitiated).async() {  
    let imageData = try Data(contentsOf: url)  
    print("received data for image \(self.imageName)")  
  
    DispatchQueue.main.async {  
        self.ImageView.image = UIImage(data:imageData)  
    }  
}
```



update UI

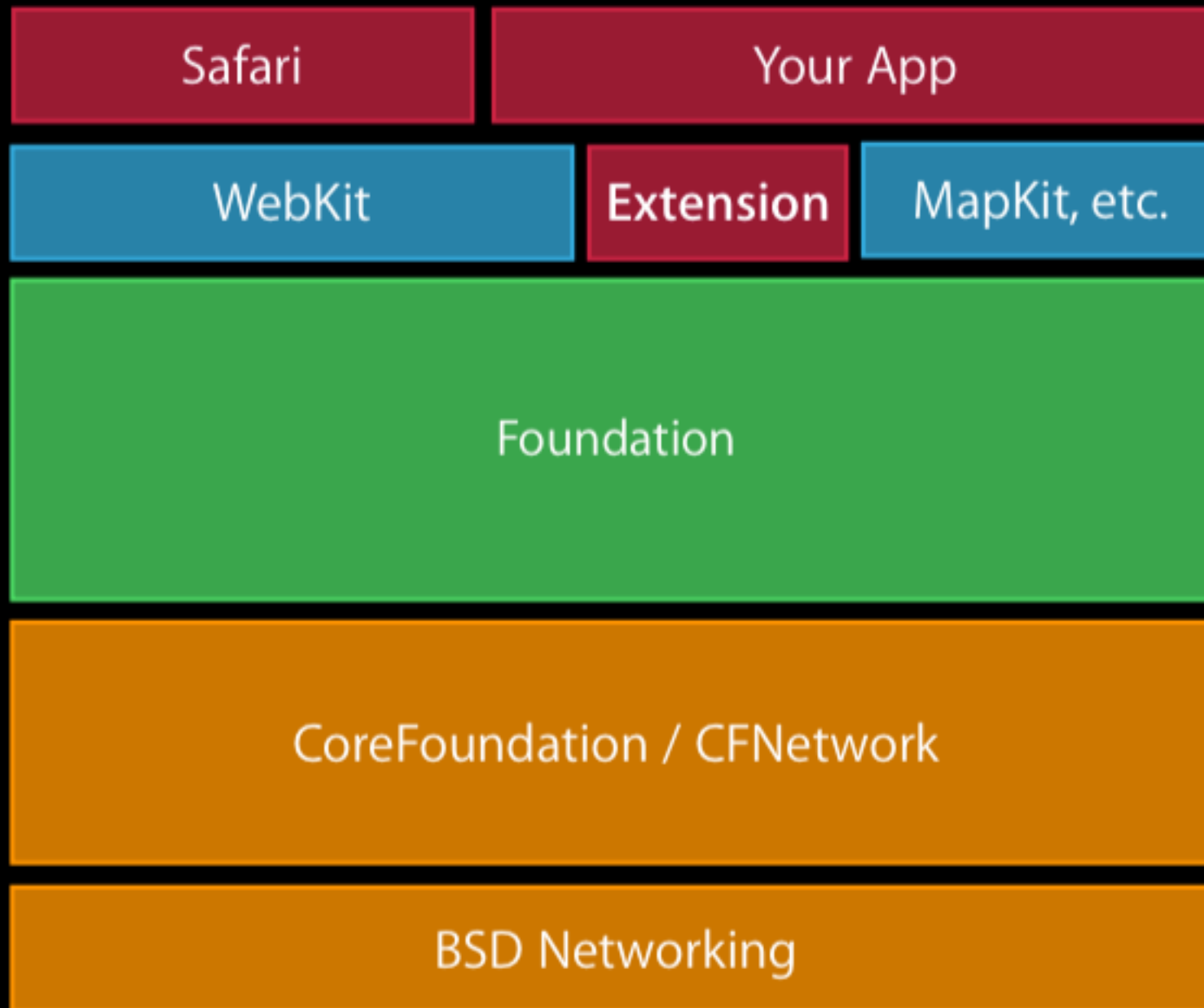
Networking APIs

Synchronous Networking APIs

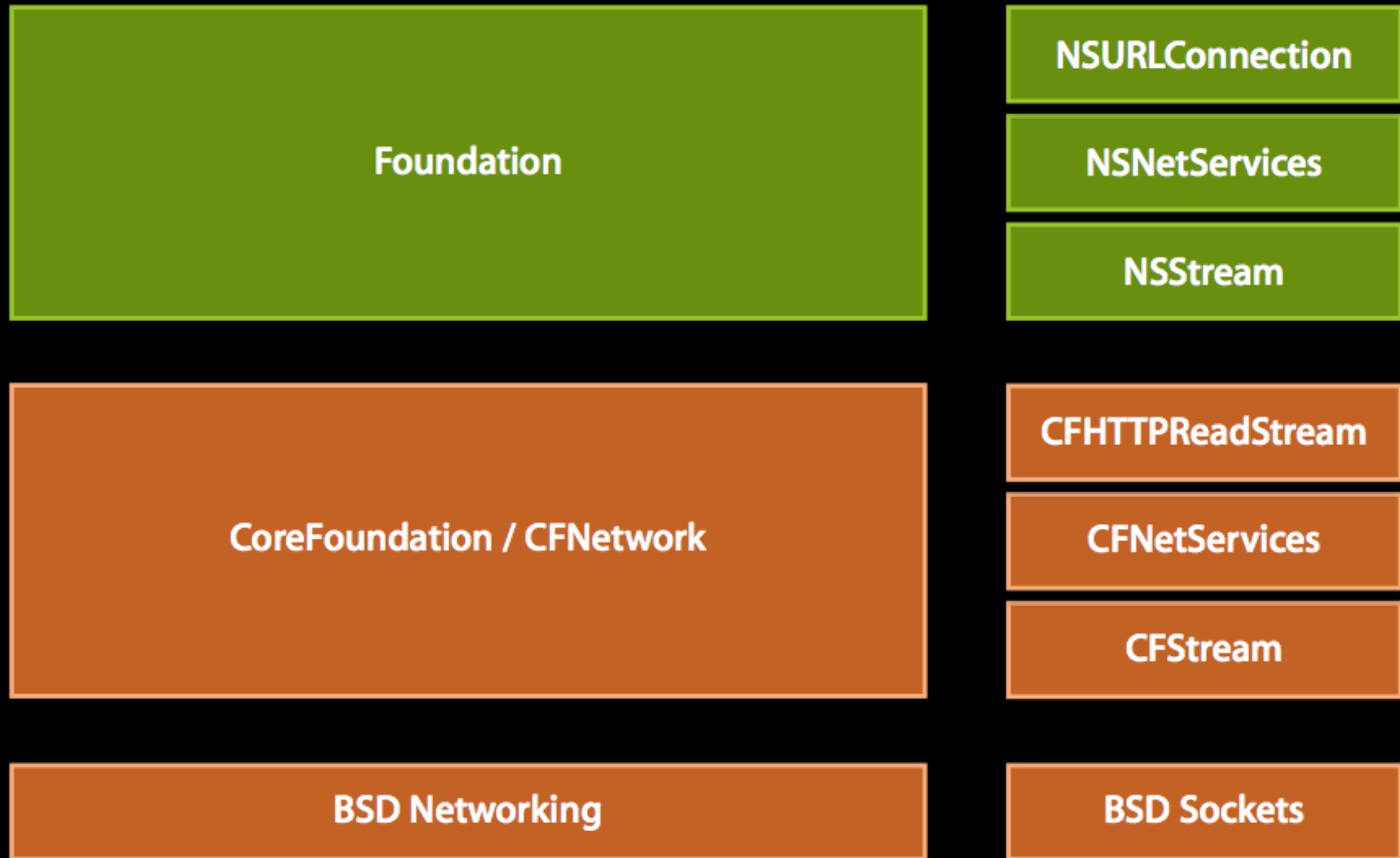
- `Data(contentOfURL)` — synchronously download the content at the URL to a `Data` object
- `String(contentOfURL)` — synchronously download the content at the URL to a `String` object
- Asynchronous networking with synchronous APIs

```
DispatchQueue.global(qos: .userInitiated).async() {  
    let imageData = try Data(contentsOf: url)  
    print("received data for image \(self.imageName)")  
}
```

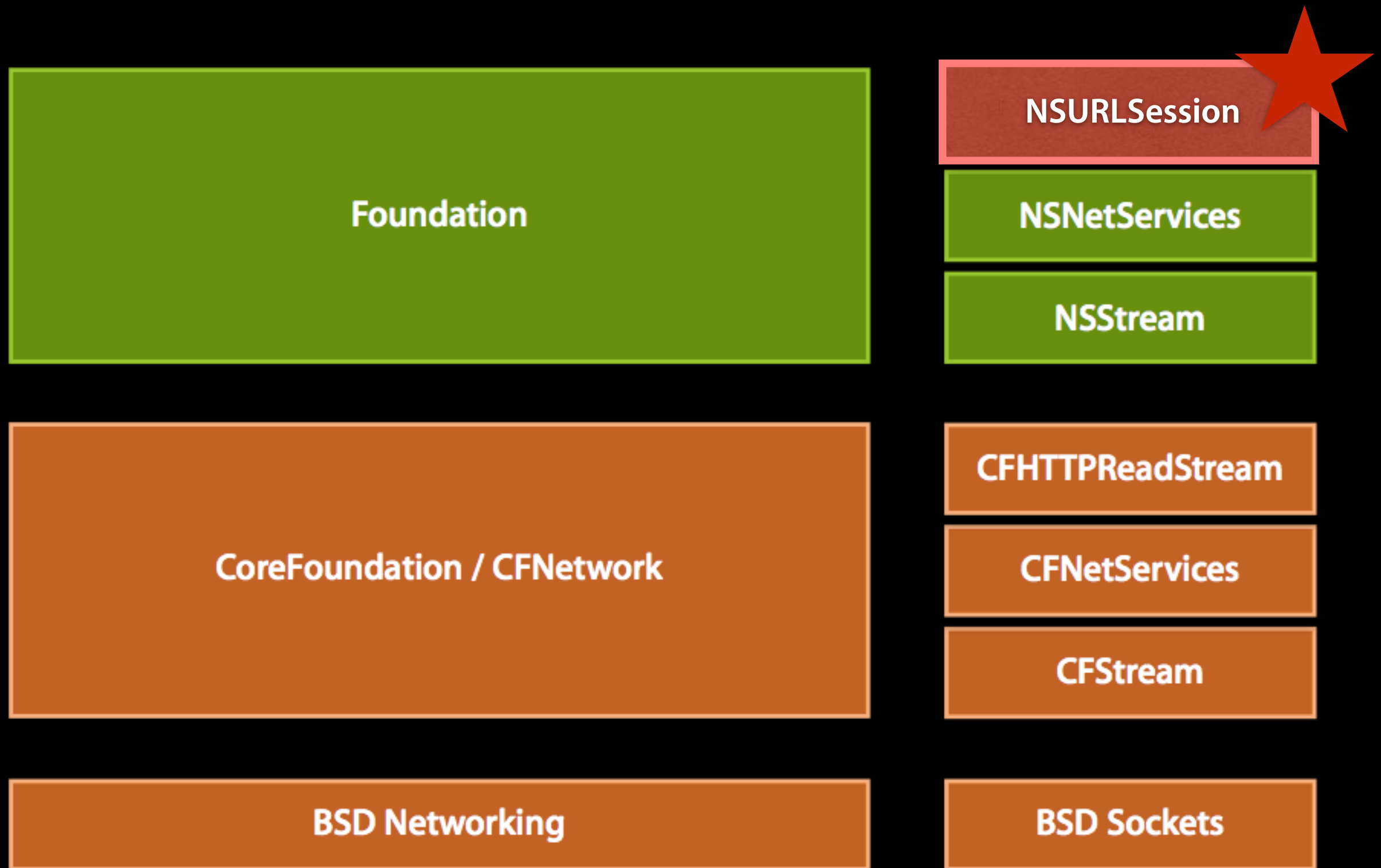
Foundation Networking



Foundation Networking

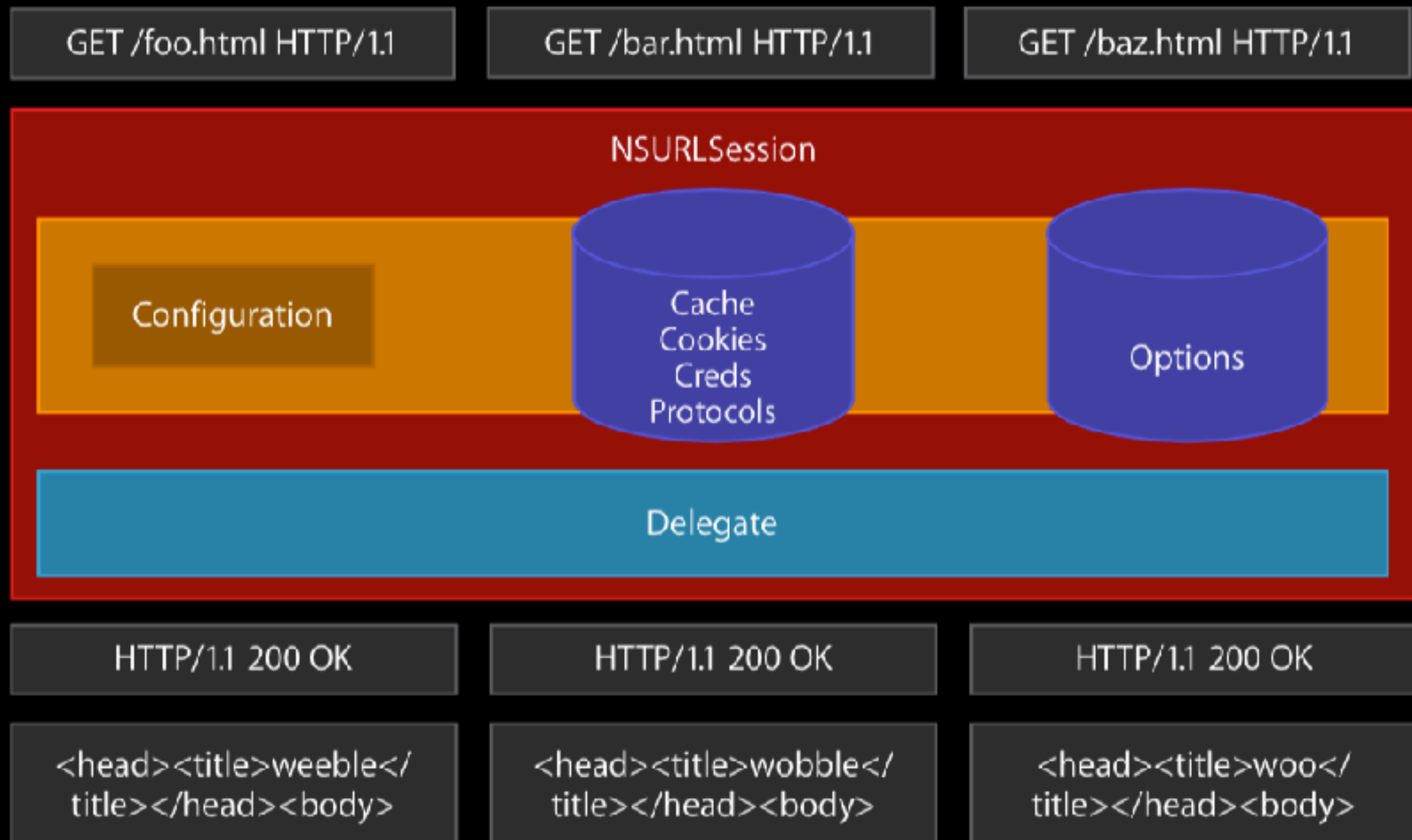


Foundation Networking



NSURLSession

- A complete suite of networking APIs for uploading and downloading content via HTTP/HTTPS



URLSession Types

- A singleton *shared session* — `URLSession.shared()`
 - Simply asynchronous requests
 - No configuration, no delegate
- *Default session*
 - Obtaining data incrementally using a delegate
 - Session data: caches, cookies, credentials are on disk
 - Customizable
 - `URLSessionConfiguration.default()`

URLSession Types

- *Ephemeral session*
 - similar to the default configuration
 - All session-related data is stored in memory — “private” session
- Background session
 - Upload/download tasks in the background when the app is not running

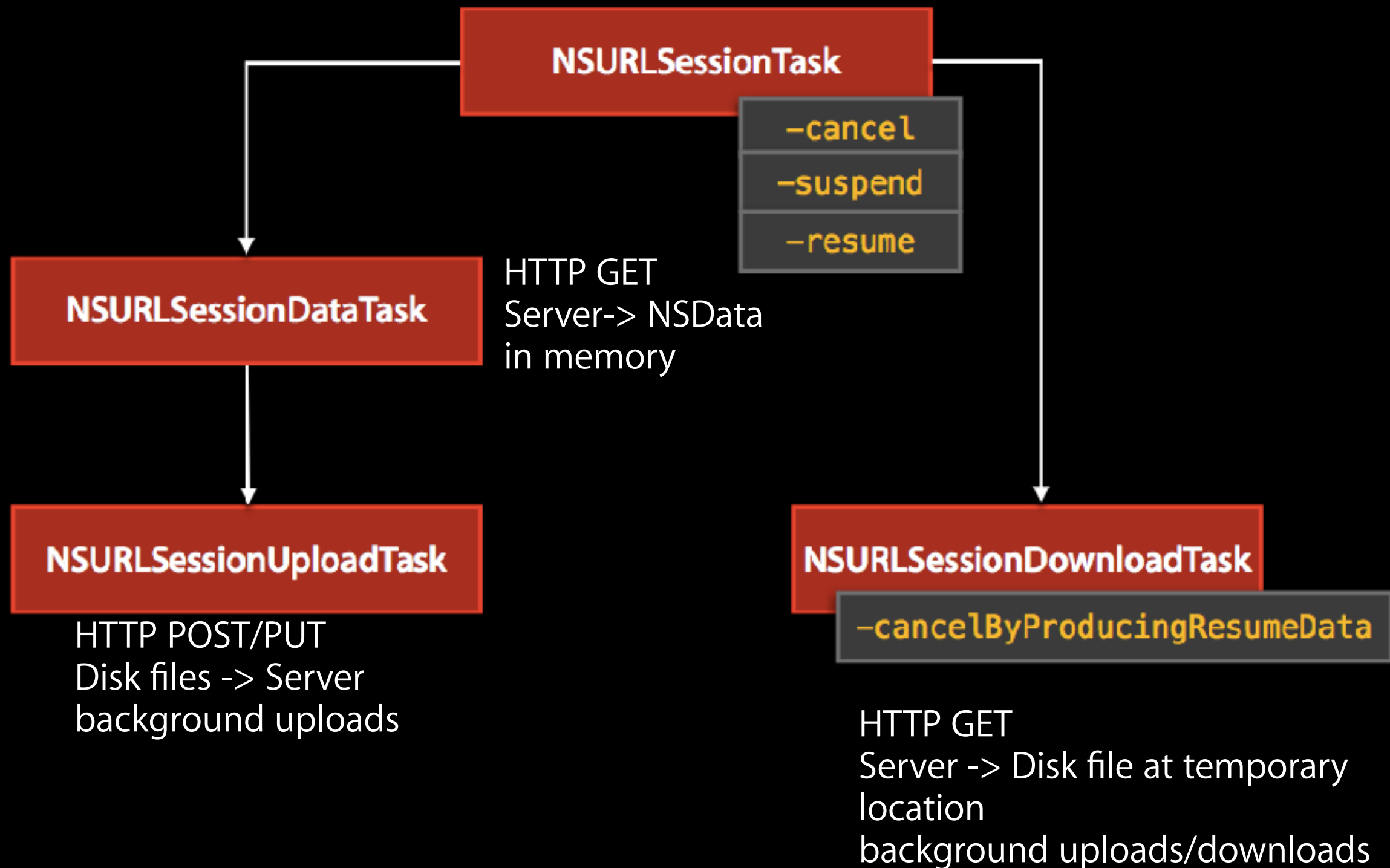
URLSession Configuration

- Defines connection behavior: max. # of simultaneous connections, whether to allow connections over a cellular network etc.
- Shared by all the tasks within a given URL session

URLSession Tasks

- Optionally upload data to a server
- Retrieve data from the server
 - NSData objects in memory
 - files in a temporary disk location

NSURLSessionTask



URLSession is Asynchronous

- Via a completion handler when a task finishes either successfully or with an error
- Calling methods on a delegate the app sets upon session creation

Using an URL Session

- Create a session configuration
- Create a session, specifying a configuration object and, optionally, a delegate
- Create task objects each representing a request
 - Each task starts out in a suspended state
 - Calls *resume()* to begin the task
- Invalidate a session by calling
 - `invalidateAndCancel()`: cancel outstanding tasks
 - `finishTasksAndInvalidate()`: allow outstanding tasks to finish before invalidating the object