iOS Dev Accelerator Week1 Day4

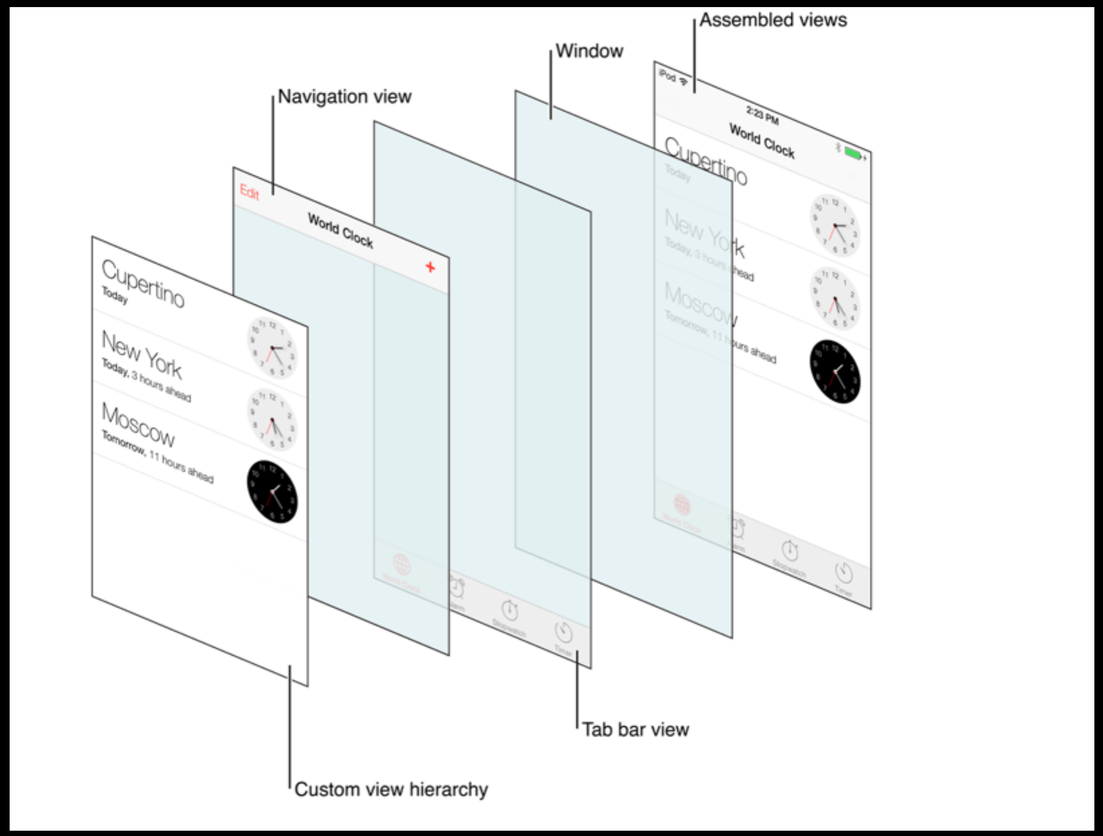
- Lazy Loading
- Navigation Controller
- TableView HeaderView
- Nib’s/Xib’s
- Technical Thursday: Stack Data Structure

Lazy Loading

Lazy Loading

- “Lazy Loading is a design pattern commonly used in programming to defer initialization of an object until the point at which it is needed”
- In iOS, lazy loading is often time used in collection views and table views for deferring the loading of resources used by cells until the time those resources are actually needed.
- Swift has introduced a lazy keyword that can make properties lazy, so they wont be instantiated until they are accessed.

Demo



Navigation Controllers

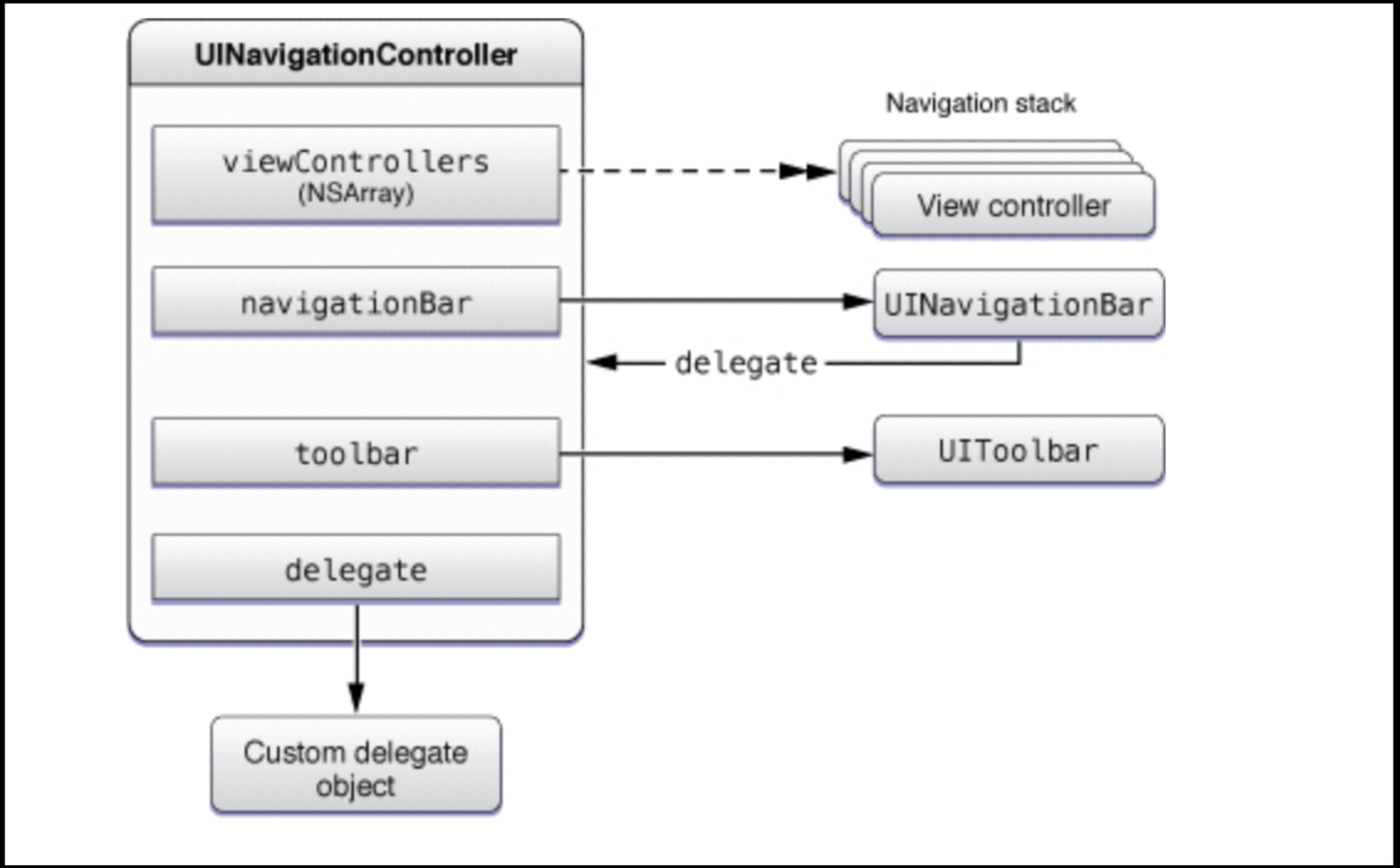
2 types of View Controllers

- Conceptually, there are two flavors of view controllers:
  - Content View Controllers: Present your app's content. Used to populate views with data from the model and response to user actions.
  - Container view controllers: Used to manage content view controllers. Container view controllers are the parents, and content view controllers are the children.

Navigation Controller

- A navigation controller is an example of a container view controller
- “A navigation controller manages a stack of view controllers to provide a drill down interface for hierarchal content”

Navigation Controller Anatomy



Creating a Navigation Controller

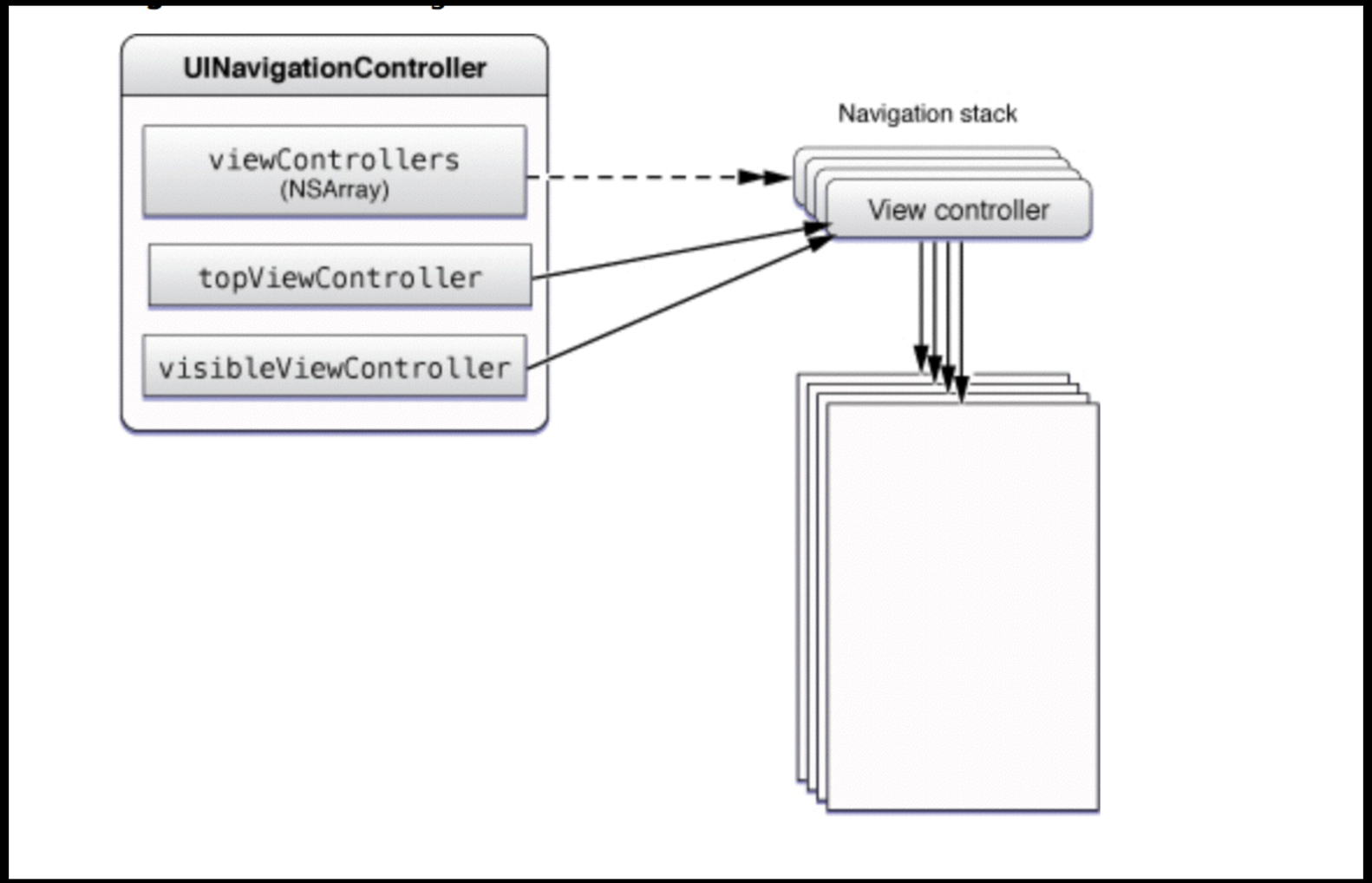
- 2 simple ways to get a navigation controller into your app:
    - Instantiate it in code. UINavigationController has 2 inits, one that takes in a view controller as its root view controller, and another that takes custom navigation bar and tool bar subclasses.
    - Embed it via storyboard.

Demo

Pushing and Popping

- A navigation controller uses a stack data structure to manage all of its children content view controllers.
- A stack is a pretty simple data structure. To add something to the stack, we push onto it. To take something off the stack, we pop.
- So to get a view controller on to the top of our navigation controller's stack, aka on screen, we can simply call pushViewController() on our navigation controller, and pass in the VC we want to push.
- And for taking a view controller off these stack, basically like pressing the back button, we call popViewController().
- There are also methods for popping to the root and popping to specific view controller in the stack.

Pushing and Popping



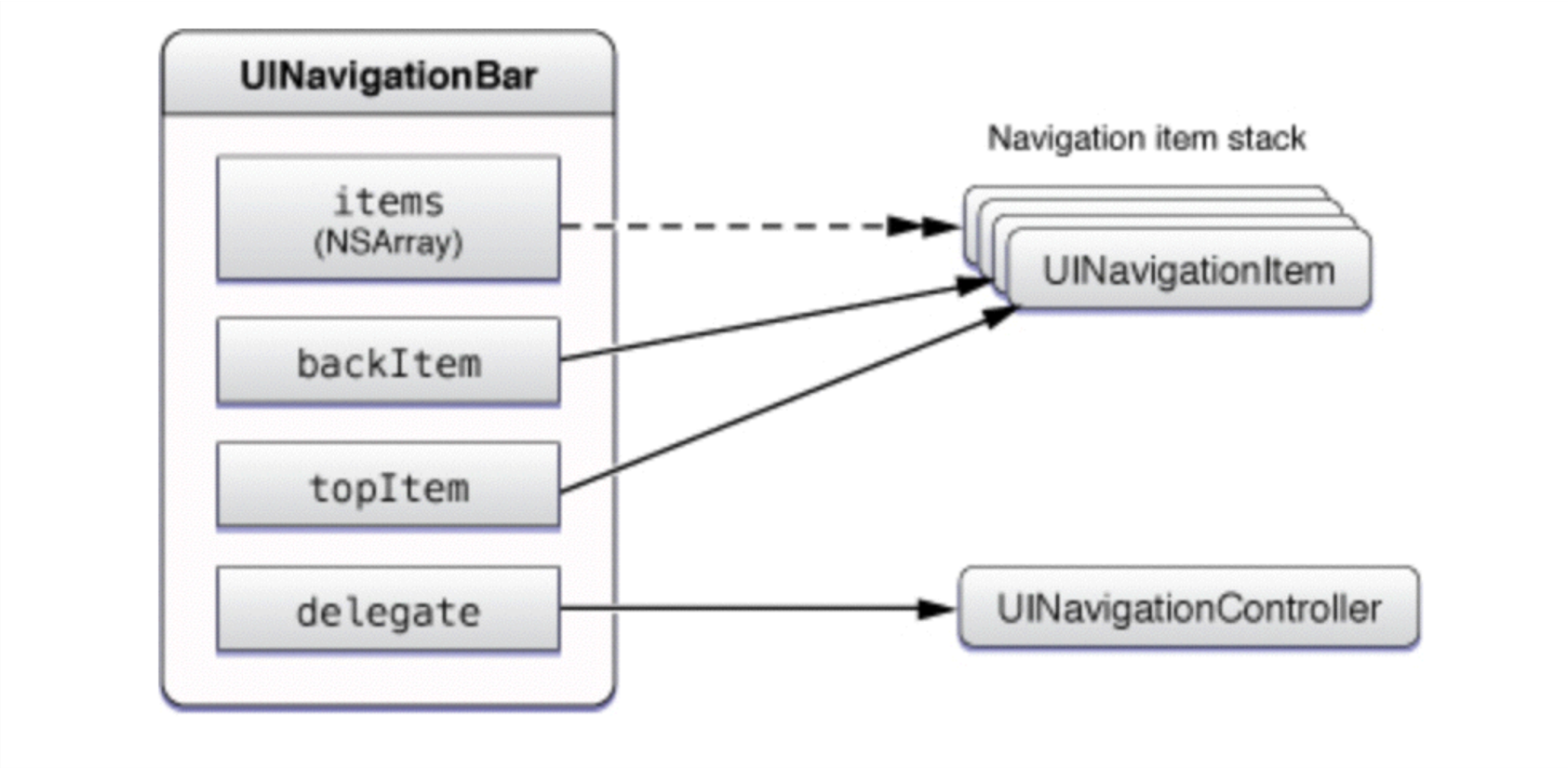
Demo

Navigation Bar

- The navigation bar of the navigation controller manages the controls of the navigation interface.
- The navigation controller takes most of the responsibility in creating and maintaining the navigation bar.
- You can also create UINavigationBar's as standalone view's and use them in your apps without using a navigation controller.

Navigation Bar Anatomy

- A navigation bar has pretty similar setup as the navigation controller:



UINavigationItem

- UINavigationItem provides the content that the navigation bar displays. It is a wrapper object that manages the buttons and views to display in a navigation bar.
- The navigation bar keeps a stack of all the items, in the exact same order as the navigation controller keeps track of its child content view controllers.
- Each View controller has a property that points to the navigation item that corresponds to the stack of nag items the nav controller keeps.
- The navigation bar has 3 positions for items: left, center, and right.

UINavigationItem positions

- Left: usually reserved for the back button, but you can replace it with whatever view you want by setting the navigation bar's leftBarButtonItem property.
- Center: Displays the title of the currently displayed view controller.
- Right: Empty by default, is typically used to place buttons that fire off actions.

Altering the Nav Bar

- The Navigation Controller owns the navigation bar and its very protective of it.
- You can’t modify its bounds,frame, or alpha values directly.
- The properties you can modify are barStyle, translucent, and tintColor.
- To hide the navigation bar, call the method setNavigationBarHidden(animated:)

More hiding properties

- hideBarsOnTap
- hideBarsOnSwipe
- hidesBarsWhenVerticallyCompact
- hidesBarsWhenKeyboardAppears
- barHideOnTapGestureRecognizer
- barHideOnSwipeGestureRecognizer

Demo

TableViewHeaderView

The tableView's header view

- In addition to header views for each section, you can also have an accessory view that sits onto of the table itself.
- It's just a regular UIView
- Can be set in code by creating the view and setting it to the tableView's tableHeaderView property.
- Set in storyboard by just dragging a view on to the tableview.

Demo

Nib’s/Xib’s

Nib's/Xib's

- Nibs, or Xibs since Xcode 3.1, allowed developers to create interfaces graphically instead of programatically.
- Storyboard are now the best way to create your interface, but nibs still serve a few purposes:
  - Create a single isolated view controller layout
  - Create the layout of a non-view controller related view (like a tableview cell!)
  - Work better with git

How Nib's work

- Interface objects are what you add to a nib file to layout your interface.
- At runtime, the interface objects are instantiated by the system and inserted into your code.

Nib Lifecycle

1. The system loads contents of the nib file into memory
2. the nib's object graph is unarchived, with our old friend NSKeyedUnarchiver
3. All connections are reestablished (outlets and actions)
4. awakeFromNib is sent to all appropriate objects

Demo

Technical Thursdays: Stacks and Queues

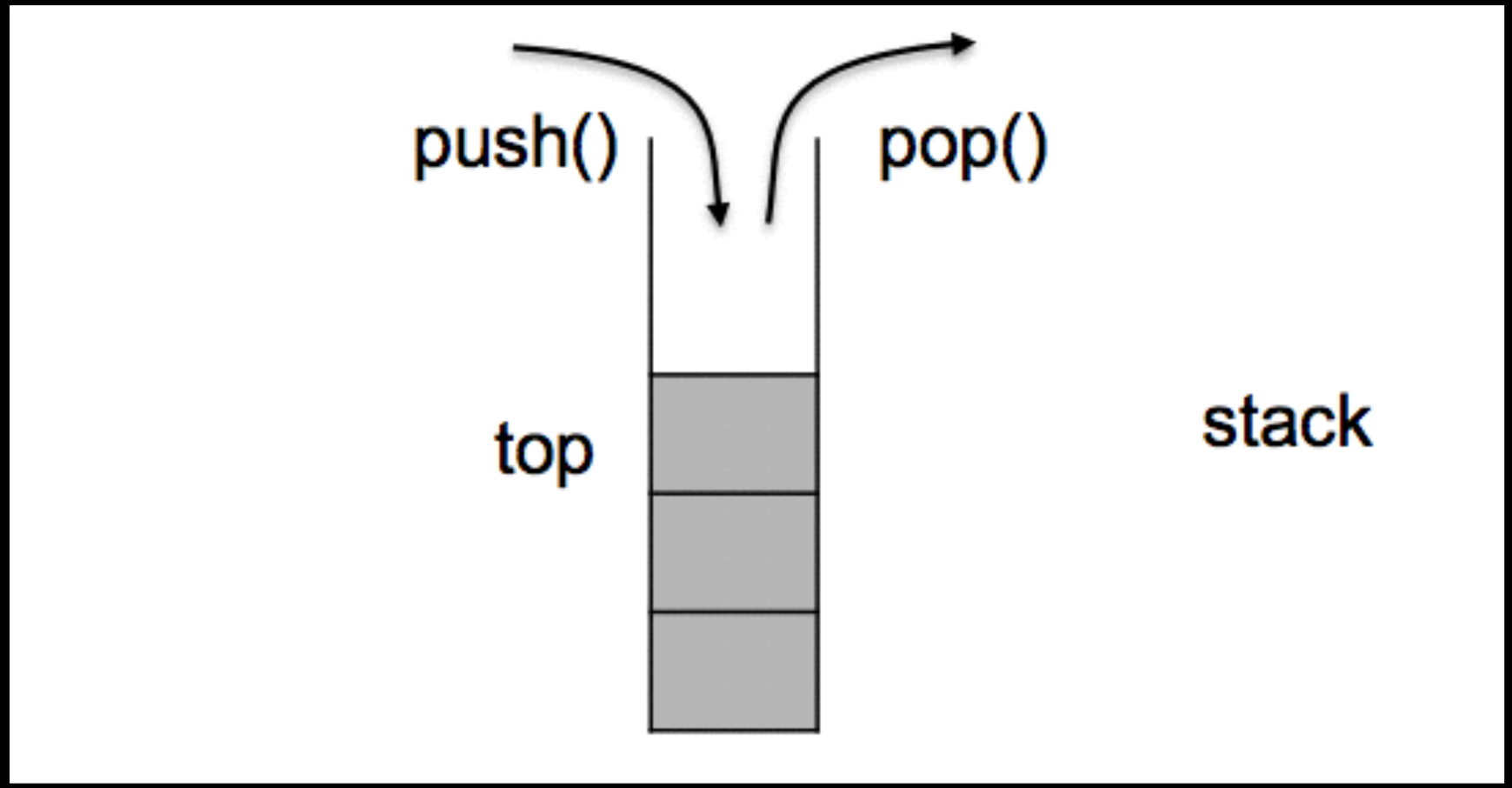
Stack Data structure

- “The stack data structure is one of the most important data structures in computer science” -Wikipedia (so you know its legit)
- Used in navigation controllers!
- And also in pretty much every programming language ever.

Stack Metaphor

- To understand a stack, think of a deck of playing cards that is face down.
- We can easily access the card that is on top.
- There are 2 things we can do to access the card on top:
  - peek at it, but leave it on the deck
  - pop it off the deck. When we pop something off a stack, we are taking it off the stack
- When you want to put something onto the stack, we call it pushing onto the stack
- A stack is considered LIFO. Last in, first out. This means the last thing we added to the deck (pushed) is the first thing that gets taken off (popped)

Stack Visual



Call stack

- “In computer science, a call stack is a stack data structure that stores information about the active subroutines of a computer program.”
- In most high level programming languages, the implementation of the call stack is abstracted away for us, but its still valuable to know what it is and how it works.
- When a function/method is called, it is pushed onto the stack. Any local variables are created and stored on the stack as well. If the called function calls another function, that 2nd function is pushed onto the stack as well. This keeps happening until all the functions have returned and they are all popped off the stack.

call stack demo

Implementing a stack

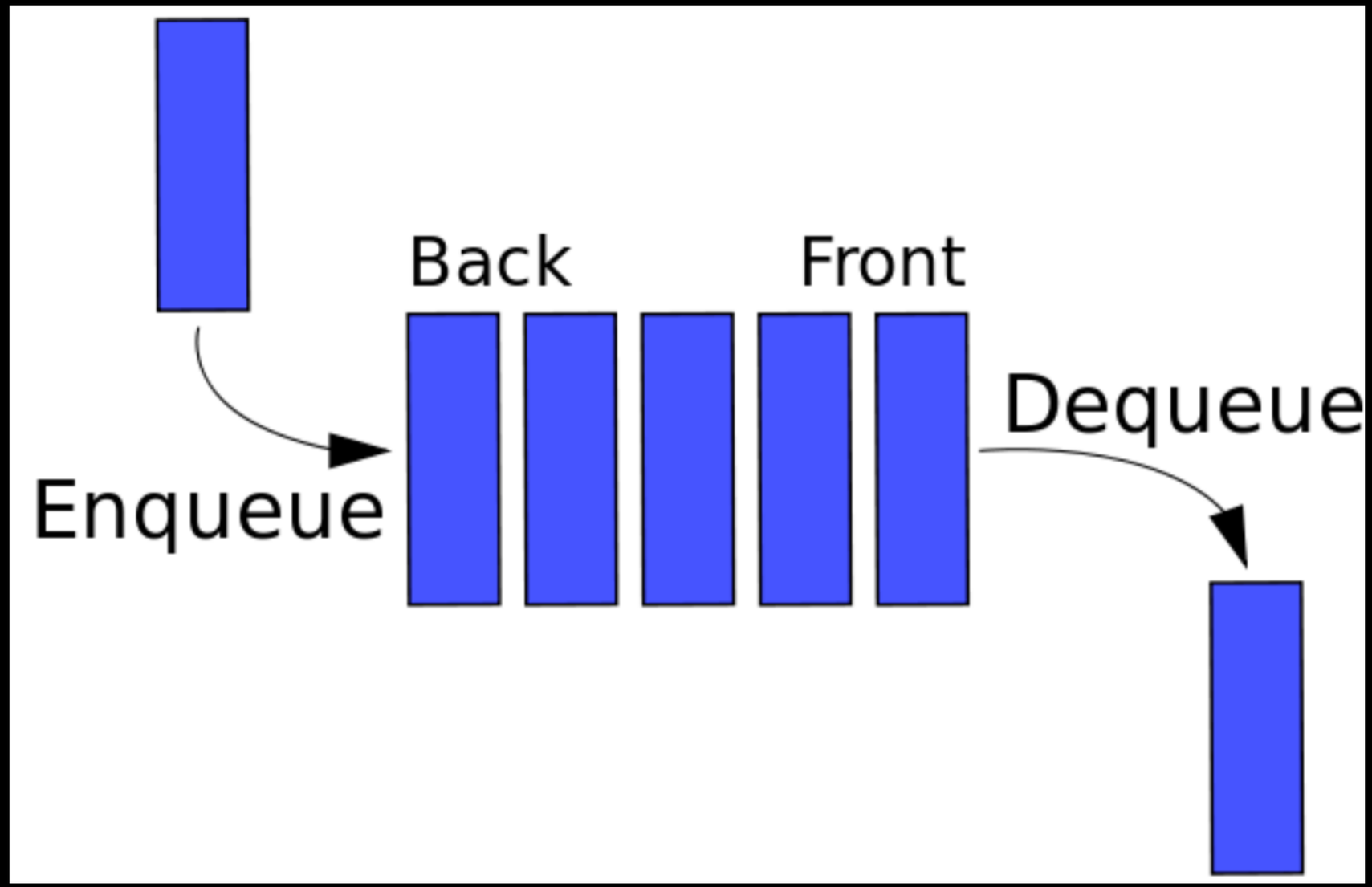
- There are two easy ways to implement a stack: using an array and using a linked list.
- Since we haven't learned linked lists yet, we will focus on the array way for now.

stack implementation demo

Queue Data structure

- A queue data structure is similar to a stack, except it is FIFO (First in, First Out).
- The terminology is also different. Theres no pushing and popping. This time its called enqueuing for adding something to the queue, and dequeuing for taking something out of the queue.
- Peek is still used to simply observe the value at the front of the queue.

Queue Visual



queue implementation demo