iOS Dev Accelerator Week1 Day3

- AutoLayout
- Auto Sizing Cells
- SizeClasses
- UIActivityIndicator
- UINavigationController
- Git
- Queues (if we have time today, if not tomorrow)

Homework Review

Autolayout

- A constraint-based layout system for making user interfaces.
- AutoLayout works by you bossing it around.
- Specifically, you can tell it 2 things about every view in your interface:
    1. Size - How big the view is going to be
    2. Location - Where the object is going to be located in its super view
- Once told ‘the rules’, autolayout will enforce the rules you setup.
- These rules are setup using constraints.

Constraints

- Constraints are the fundamental building block of autolayout.
- Constraints contain rules for the layout of your interface's elements.
- You could give a 50 point height constraint to an imageView, which constrains that view to always have a 50 point height. Or you give it a constraint to always be 20 points from the bottom of its superview.
- Constraints can work together, but sometimes they conflict with other constraints.
- At runtime Autolayout considers all constraints, and then calculates the positions and sizes that bests satisfies all the constraints.

Constraints Priorities

- Constraints have a priority level. By default all constraints have the same priority level set, which is the maximum level of ‘required’
- Constraints with higher priority levels are satisfied before lower priority levels.
- You usually don’t have to alter the priority levels of constraints unless your interface is acting strangely or Xcode presents it to you as a solution.

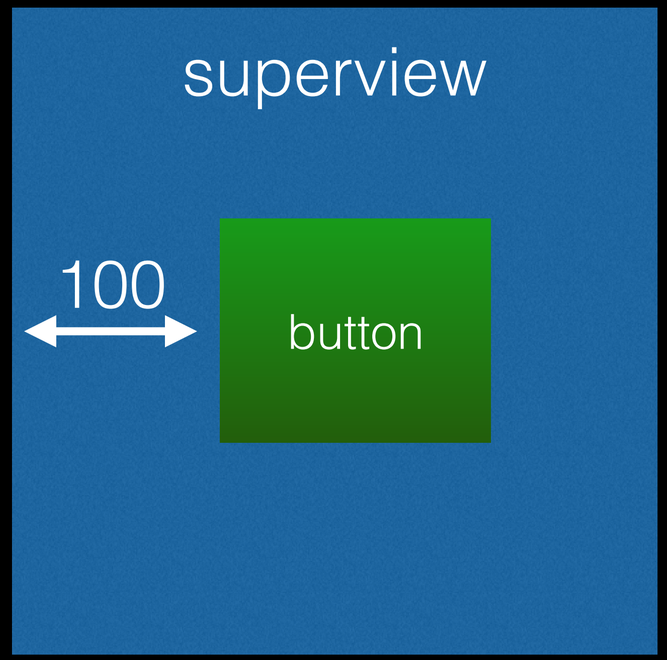
Demo

Attributes

- When you attach constraints to a view, you attach them using attributes.
- The attributes are: **left/leading, right/trailing, top, bottom, width, height, centerX, centerY.**
- Attributes tell the constraint what part of the view(s) they should be manipulating

Attributes

- So if you attach a constraint of 100 points from a button's left attribute to its container's left attribute, thats saying "I want the left side of this button to be 100 points over from its super view's left side"



Demo

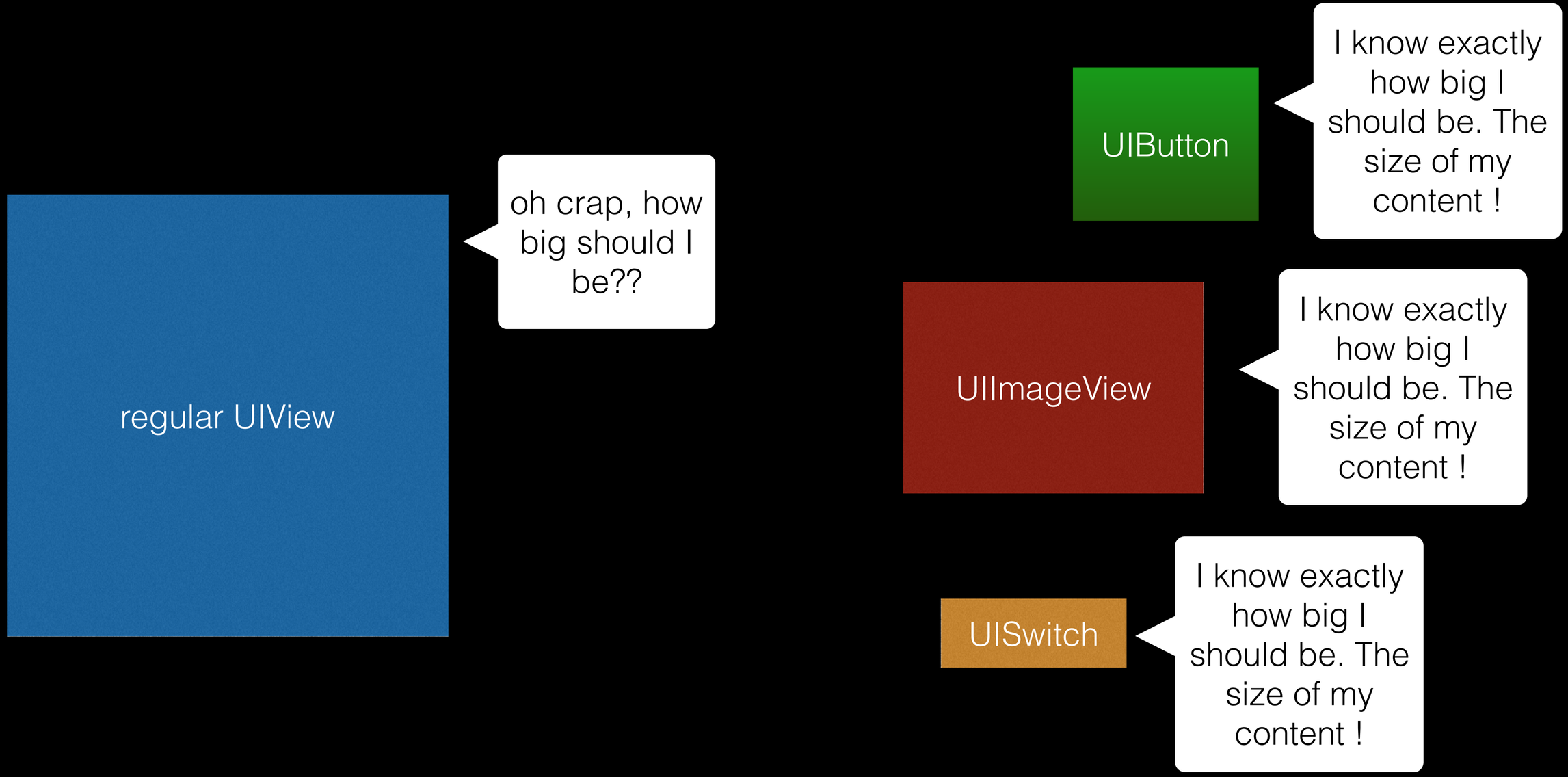
Storyboard and Autolayout

- Storyboard makes setting up autolayout pretty intuitive and painless, and even though you can setup autolayout completely in code, Apple strongly recommends doing it in storyboard.
- Xcode will let you build your app even if you have constraints that are conflicting and incorrect, but Apple says you should never ship an app like that.
- **When you drag a object onto your interface, it starts out with no constraints.**
- **Once you apply one constraint, that view needs to have constraints dictating its size AND location**

Intrinsic Content Size

- Intrinsic content size is the minimum size a view needs to display its content.
- Its available for certain UIView subclasses:
  - UIButton & UILabel: these views are as large as they need to display their full text
  - UIImageView: image views have a size big enough to display their entire image. This can change with its content mode.
- You will know a view has an intrinsic content size if autolayout doesn't require its size to be described with constraints.

Intrinsic Content Size



Demo

Hugging vs Resistance

- Every view that has an intrinsic content size, has attributes which define how constraints are built to show that size
- The first one is called content hugging priority, which dictates how much the view resists growing because of constraints.
- The second one is called compression resistance, which dictates how much the view resists shrinking because of constraints.
- Both of these categories have 2 values, one for horizontal and one for vertical.
- Whats nice is Xcode will usually tell you if you need to adjust those numbers.

Automatic TableView Row Height

Row Height

- Right now our table view gets it row height straight from the storyboard.
- Theres a datasource method we could implement, that requests the row height for every row that is displayed
- Prior to iOS 8, having dynamic cell height was a bit of a pain. You had to calculate the size of your text labels based on the font type, font size, and how many characters you had. You would do this in the data source method mentioned in the previous bullet

New way

- With iOS8, it is much easier:
  - Make sure you are using auto layout in your cell
  - Ensure the elements that are going to be dynamic and dictating the height don't have fixed height constraints
  - Modify the settings of the elements (for label set lines to 0, for text view disable scrolling)
  - Give your table view an estimatedRowHeight and then set its rowHeight to UITableViewAutomatic Dimensions

Demo

Size Classes

Size Classes

- “Size classes are traits assigned to a user interface element, like a screen or a view”
- There are only two types of size classes, Regular and Compact.
- Size classes, together with displayScale and userInterfaceIdiom (iPhone or iPad) make up a trait collection.
- Everything on screen has a trait collection, including the screen itself, and view controllers as well.
- The storyboard uses a view controller’s trait collection to figure out which layout should be currently displayed to the user.

Size Classes and Storyboard

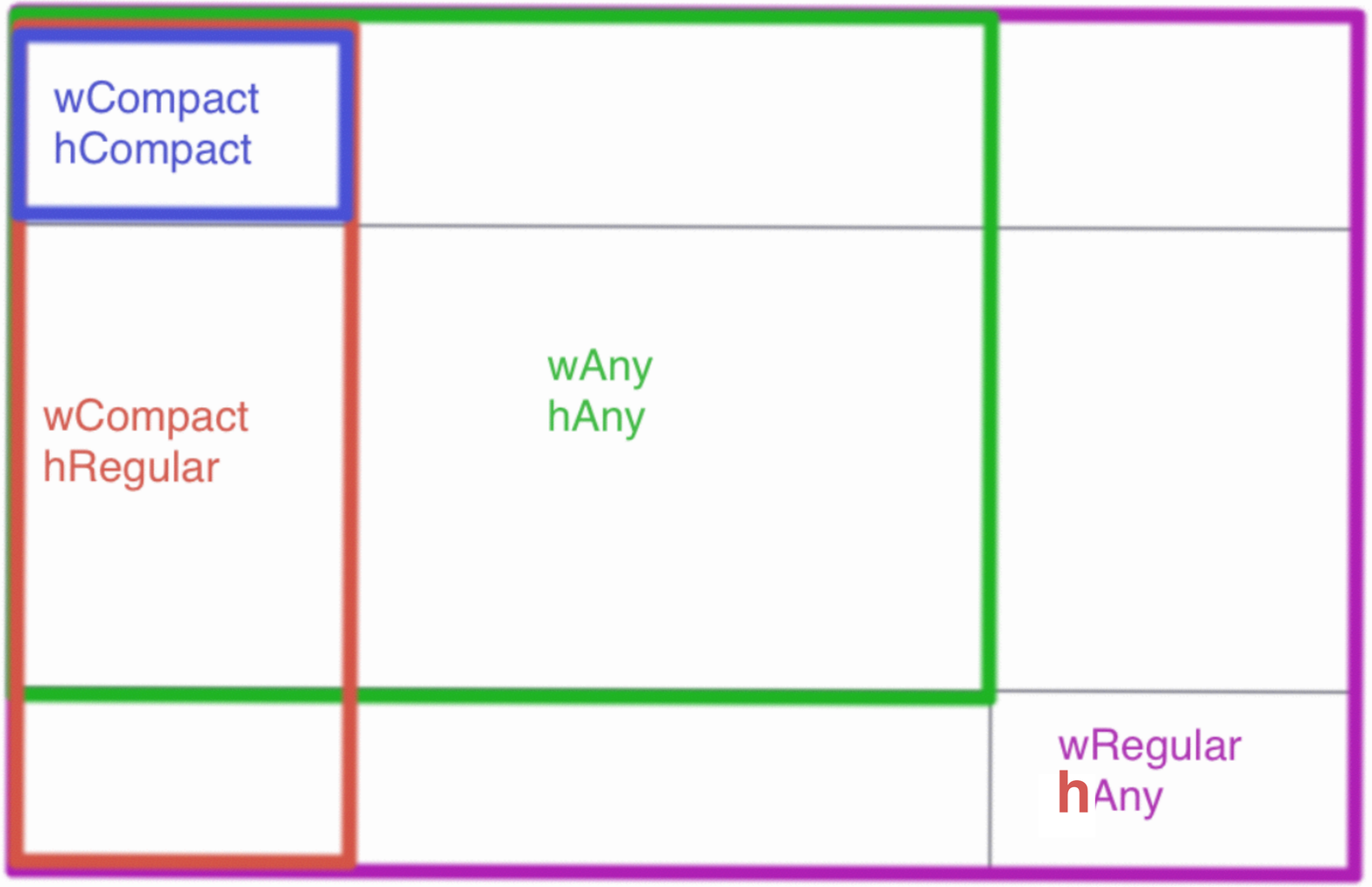
- Size classes allow you to have different constraints and layouts for each configuration on the storyboard.
- By default, every size class configuration will pull from the base configuration, which is wAny hAny.
- If you change your storyboard's configuration, certain changes you make will only apply when your app is running in that specific size class.

Size Classes

- Specifically, there are 4 things you can change in each configuration on your storyboard:

  1. constraint constants.
  2. font and font sizes
  3. turning constraints off
  4. turning view on and off

Size Classes



- iPad Landscape and Portrait
- Base configuration
- iPhone Portrait
- iPhone Landscape

Demo

UIActivityIndicator

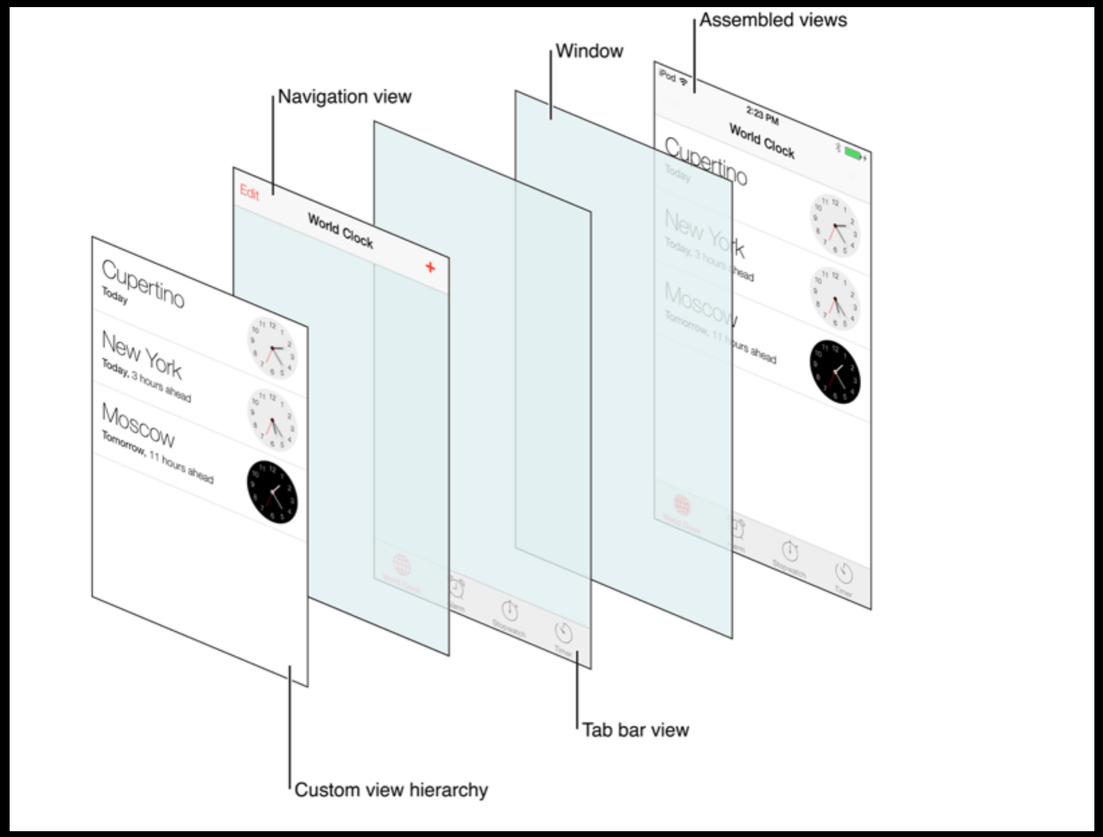
UIActivity Indicator

- An activity indicator shows that a task or process is progressing.
- An activity indicator does 2 things:
  - Spins while a task is progressing and disappears when task is complete
  - Doesn't allow user interaction
- Activity indicator assures the user their task or process hasn't stalled!

UIActivity Indicator Setup

- Drag it out from storyboard, place it in your view hierarchy.
- Should be placed at the bottom of the hierarchy, show it shows up on top
- Give it constraints just any other view (probably center of screen)
- Create an outlet for it
- Call startAnimating on it when you want to start showing progress to user, and stopAnimating when the task is complete
- It is your responsibility to hide/remove the indicator.

Demo



Navigation Controllers

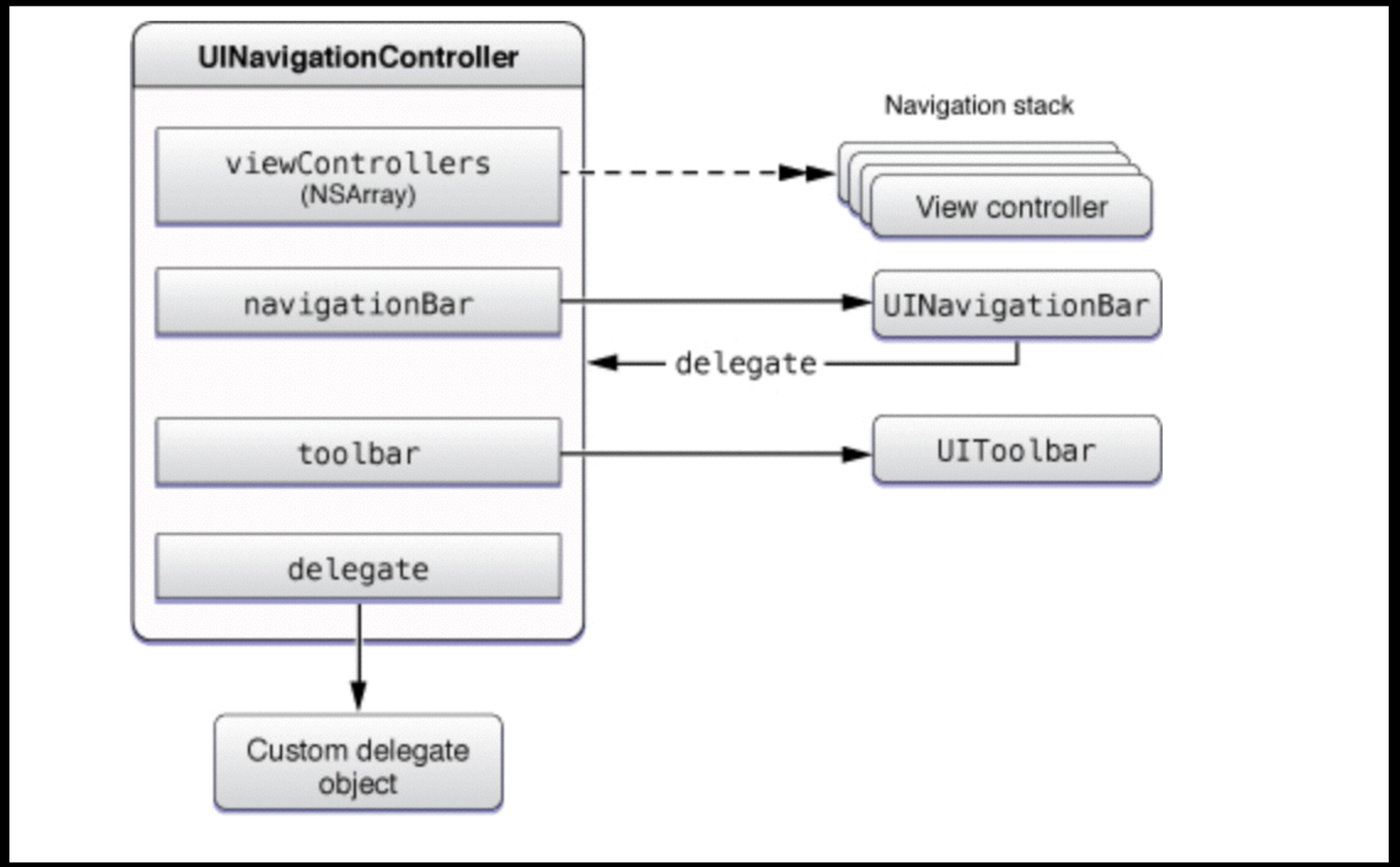
2 types of View Controllers

- 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 respond 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

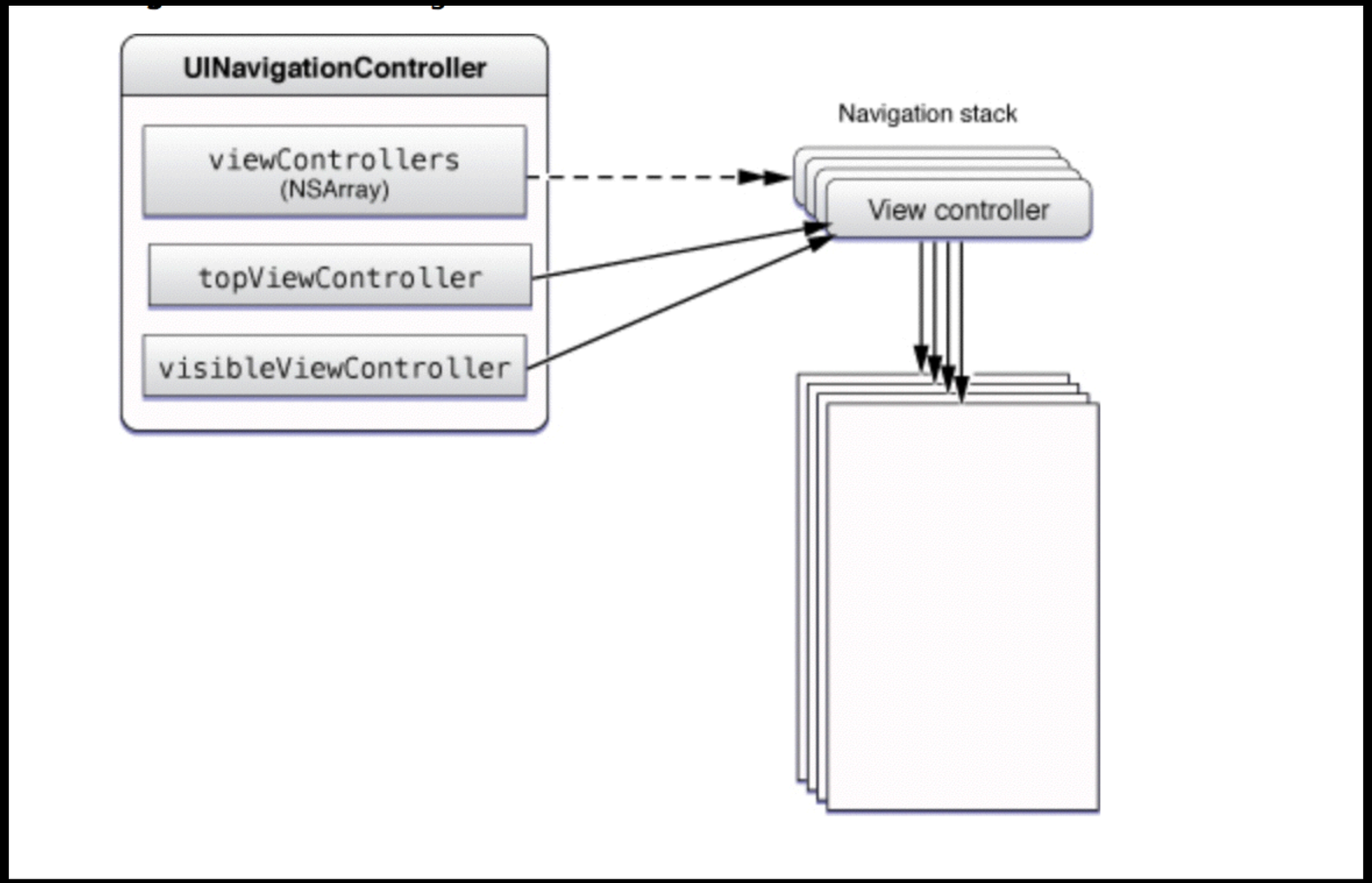
Accessing the navigation controller

- Every view controller has a property called navigationController.
- This property is reference to the navigation controller the view controller is currently in, **IF it is in a navigation controller**
- So you should always check if it exists first, before interacting with it (optional binding!)

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



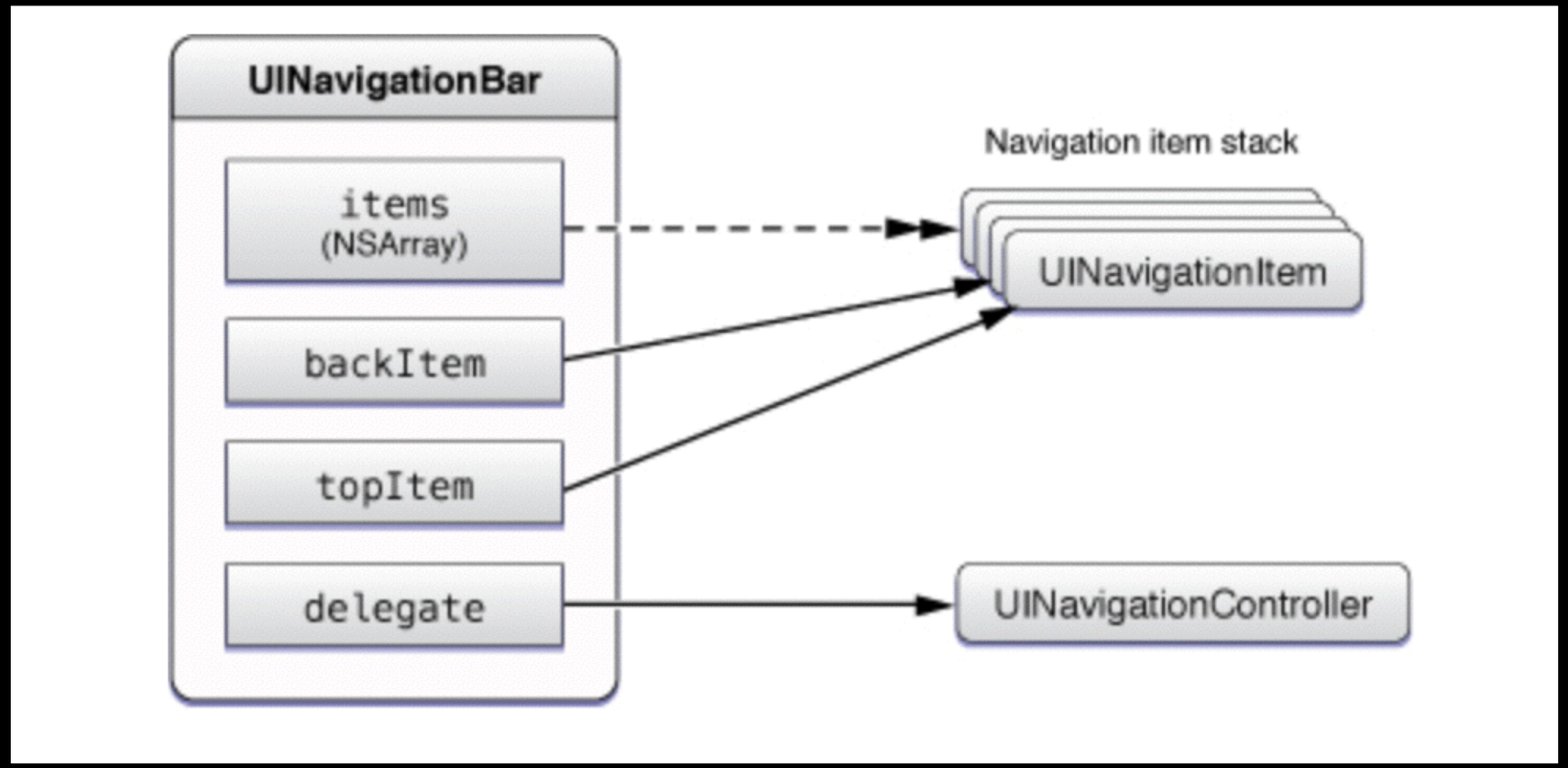
Demo

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 (rare)

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 managing navigation controller uses the navigation items of the topmost two view controllers to populate the navigation bar with content.
- 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 its corresponding navigation item
- The navigation bar has 3 positions for an item to fill: 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.

Demo

Altering the Nav Bar

- The Navigation Controller owns the navigation bar and is 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 on the nav controller

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

Demo

Git

What is Git?

- Git is an open source **version control** system.
- Git is an application you install on your computer.
- Git itself is not Github, Github is a hosting service for git repositories. Basically, Cloud.
- type git —version into terminal to see if you have git installed and which version you have.

What is version control?

- Version control is a system that keeps track of changes to a file or a group of files over time so you can retrieve specific versions of the file(s) later.
- Git is considered a distributed version control system (DVCS) because any clients who check out a repository has a fully mirrored repository with all prior file histories. They essentially have a full backup.
- So, if any server or client dies, they can just check out the repo from another client or server who they were collaborating with and receive the full restore.
- A repo, or repository, is simply a place where the history of your work is stored.

Git Basics

- There are really 2 ways to get a git project on your computer: import an existing directory into git, or cloning an existing repository from another server. We will talk about cloning later.
- git init - Creates a .git directory or reinitializes an existing one in the directory you are currently in. This turns the current directory into a git repository.
- After running git init, there will be a hidden .git file in the directory you are in. type ls -a to list all files including hidden files. You will see the .git file.

.git

- So what is this mysterious .git file?
- The .git file is actually just a directory.
- When you run git init, it creates this directory and writes a few files into it.
- These files define the configuration of the git, and the history of the project git is now managing.

Demo

git add

- It is important to understand that git add is a multi-purpose command:
  - used to begin tracking new files
  - to stage files
  - mark merge-conflicted files as resolved
- If you specify a directory instead of a single file with git add the commands adds all files in that directory recursively.
- git add -A runs the add command on all files in the repository. This is a great shortcut to use, vs running add on every file you need to start tracking or stage.

Tracking new files

- When you bring a new file or create a new file in a git repository, it is not tracked by git by default.
- You need to manually add it as a tracked file.
- To do this, use the git add command.
- This adds it to the staging area.

Demo

The 3 sections

- Git uses 3 sections for a git repository:

    1. **git directory** - where git stores the metadata and object database for your project. This is the most important part of git, and it's what is copied when you clone a repository from another computer/server. It is that hidden .git file.
    2. **working directory** - a single checkout of one version of the project. These files are pulled from the compressed database in the git directory and placed on disk for you to use. Any coding you do will be in the working directory. The regular project files you see in the folder represent the working directory.
    3. **staging area** - a simple file, generally contained in your git directory, that stores information about what will go into your next commit. Sometimes referred to as the index.

Staging Area

- Things are added to the staging area with the intent of eventually committing the.
- When you add a new to track it, it is added to the staging area.
- When you need to commit a modified file, you first need to add it to the staging area.
- The staging area is managed by the file called ‘index’ in your hidden .git directory

Working Directory

- Every file in your working directory has only 2 states - tracked and untracked.
- tracked means it was in the last snapshot, and untracked means it wasn’t.
- Once a file is tracked, it can then be one of 3 states: unmodified, modified, staged
- So when you first initialize a repository, nothing is being tracked. You will need to git add the files so they can be tracked by git.

Checking the status

- Use the git status command to determine the state of your files in the git project
- It also tells you which branch you are on (more on branches later)
- If you see the message “nothing to commit, working directory clean” that means all files in the directory are tracked and have no changes to commit.
- if you are ever feeling confused or lost while working with a git project, git status is your best friend

Demo

How are files actually stored?

- When the git add command is run on a file, it creates a ‘blob’ file in the .git/objects/ directory
- This blob file contains the compressed contents of the file itself
- The blob file’s name is derived from by hashing its contents.

Git Commits

- A git commit is synonymous with saving the state of your project in git.
- Every time you commit, git essentially takes a picture of what all of your files look like at that moment in time, and then stores a reference to that snapshot of each file.
- For the sake of efficiency, if any files have not had any changes made to them since the last commit, git does not store those files again. It just stores a reference to the previous identical file it already has stored in a previous snapshot.

Git Commits under the hood

- A git commit has 3 steps under the hood:
  - Creates a tree graph that represents the content of the current version being committed
  - The commit object is created
  - The current branch is set to point at the new commit

Commits are made locally

- A git commit is strictly a local operation.
- In fact, almost all operations in git are local, thats why its so much faster than previous version control systems.
- And since repositories have the entire history of the project, you can browse through the entire set of versions of all files while offline, without making any network calls.

git commit command

- Use the git commit once your staging area is set up the way you want it.
- Running just plane git commit will launch your text editor configured in your global git config.
- It does this because git wants a git commit message describing what this commit has changed.
- Alternatively, you can use the git commit -m “commit message” to enter your commit message inline.

Demo

The 3 states

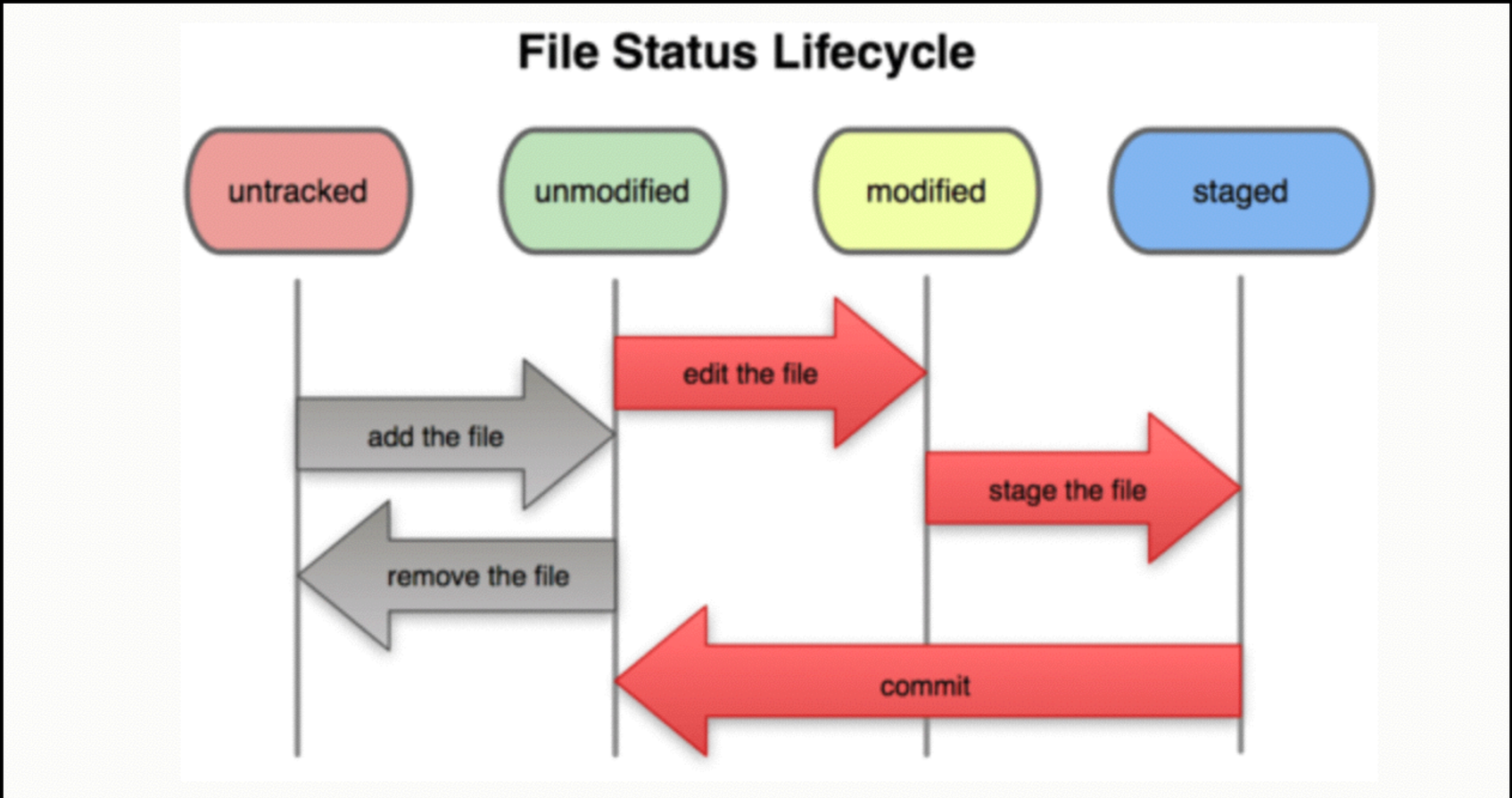
- Git has 3 main states your files can be in:

    1. Unmodified - data is safely stored in your local database. A tracked file without changes is in this state.
    2. Modified - you have changed the file but have not committed it to your database yet
    3. Staged - you have marked a file with changes to go into the next commit

Basic Git Workflow

1. You modify files in your working directory.
2. You stage the files, which adds snapshots of them in their new states, and adds them to your staging area.
3. You do a commit, which takes files as they are in the staging area and stores snapshot permanently to your git directory.

Git File Status Lifecycle

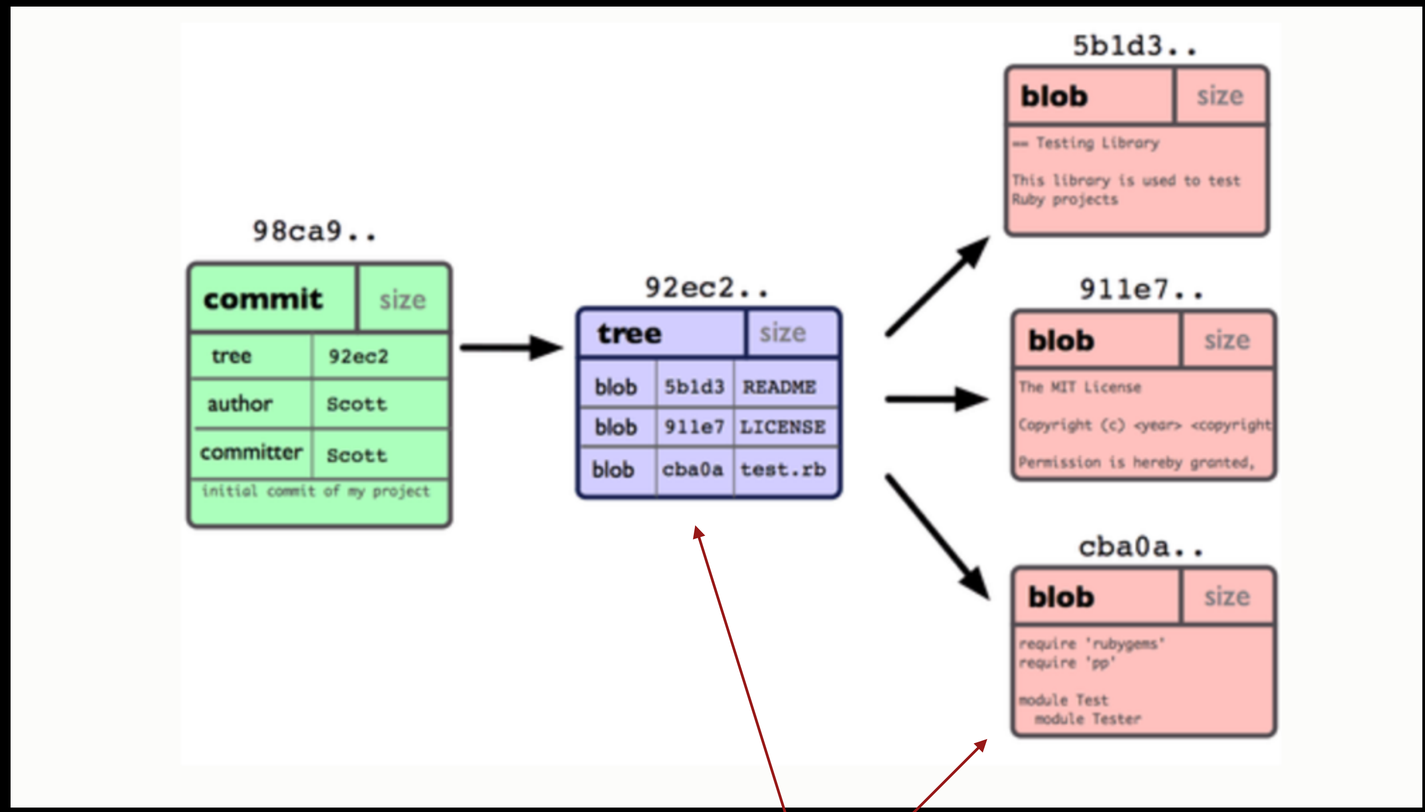


Git Branches

- To understand what a branch is, you need to further understand how git works.
- Every commit is actually a commit object.
- that commit object has a pointer to the snapshot of the staged files.
- it also has a pointer to its direct parent commit:
  - zero parent pointers if its the first commit
  - one parent pointer if its a regular commit
  - two parent pointers if its a result of a merge

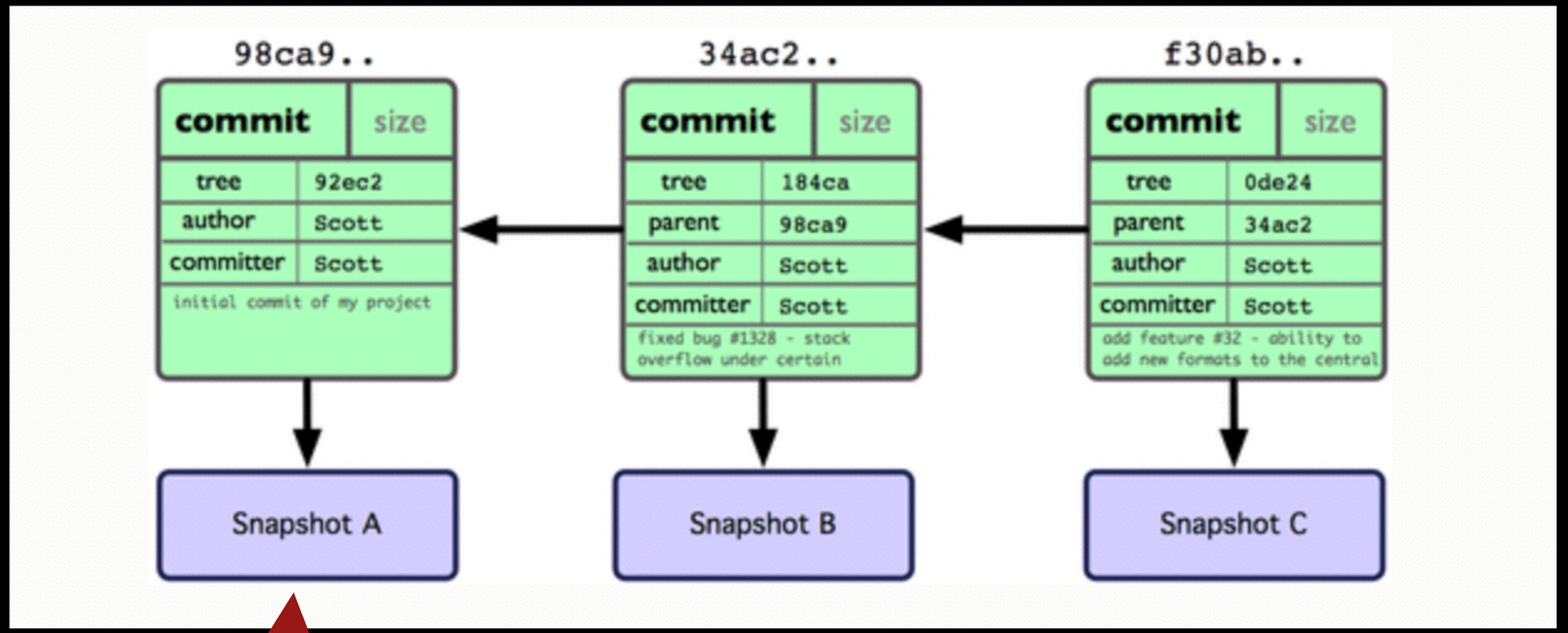
Anatomy of a Commit

- green: commit object containing metadata and pointer to tree
- purple: tree object lists the contents of the directory and specifies which files are stored as which blobs
- red: one blob file for every file in your project



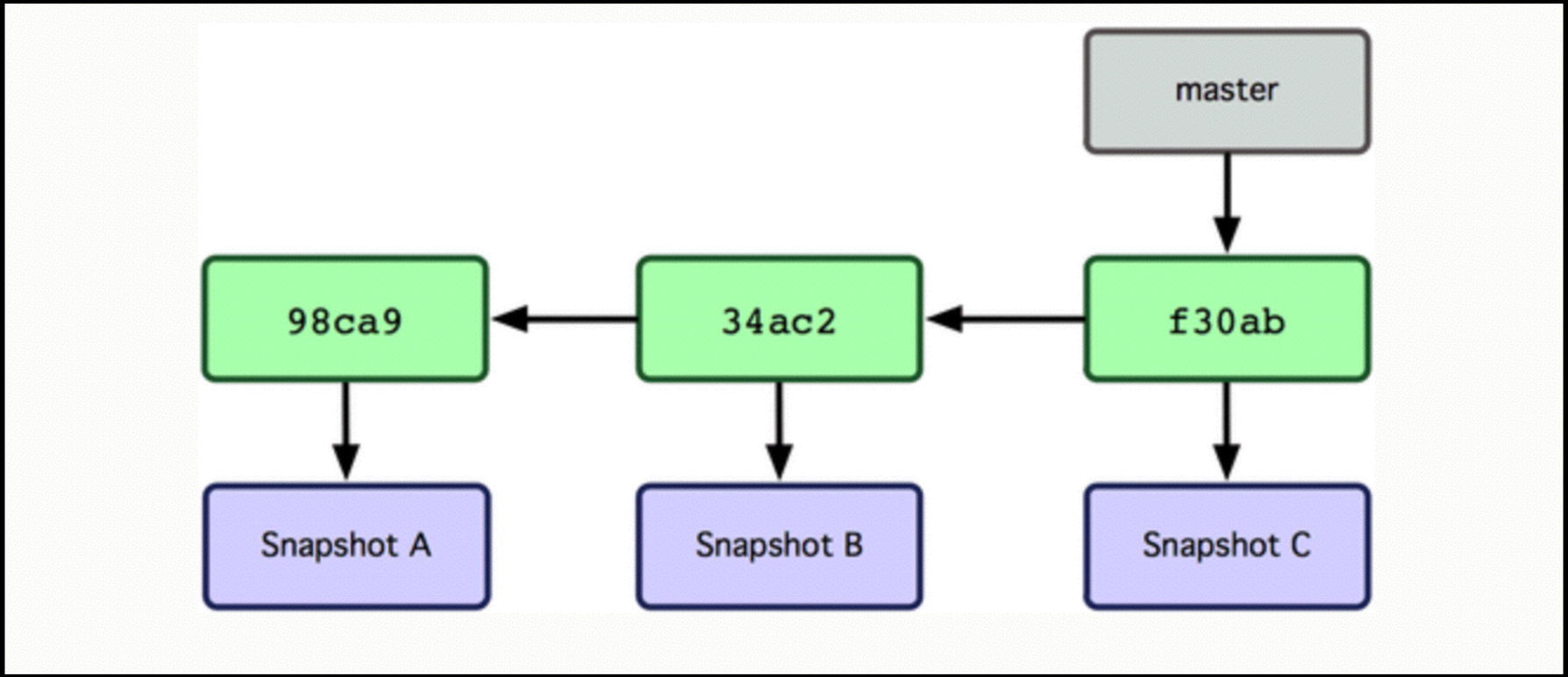
"Snapshot"

Multiple commits



initial commit

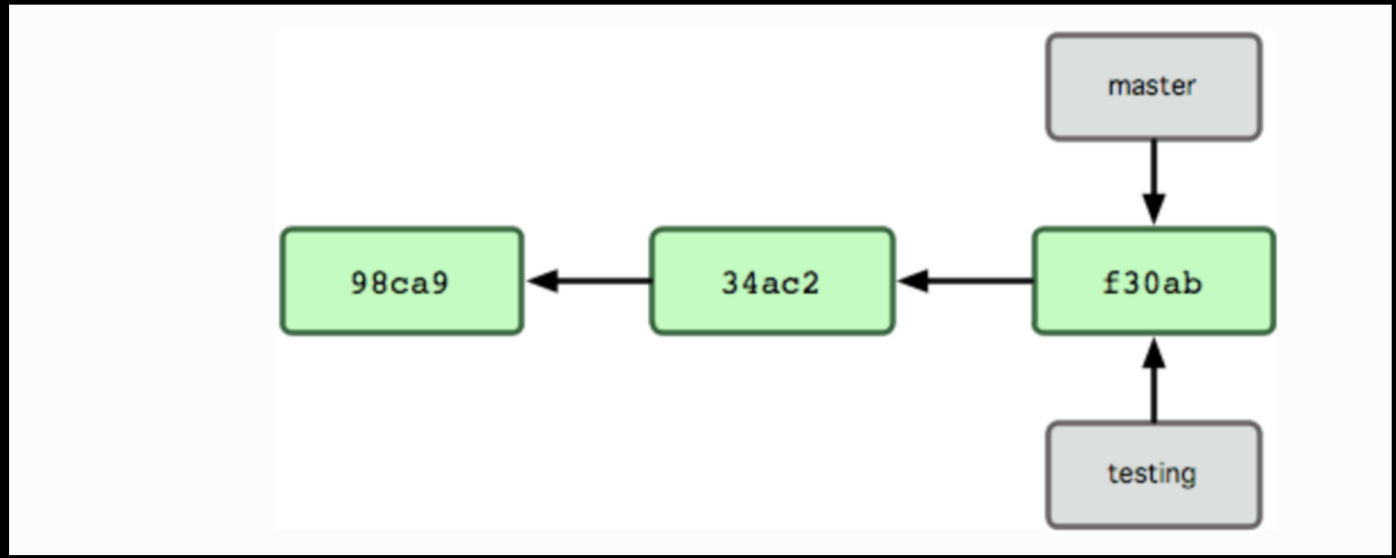
Branch == pointer



A branch is just a lightweight pointer to a commit

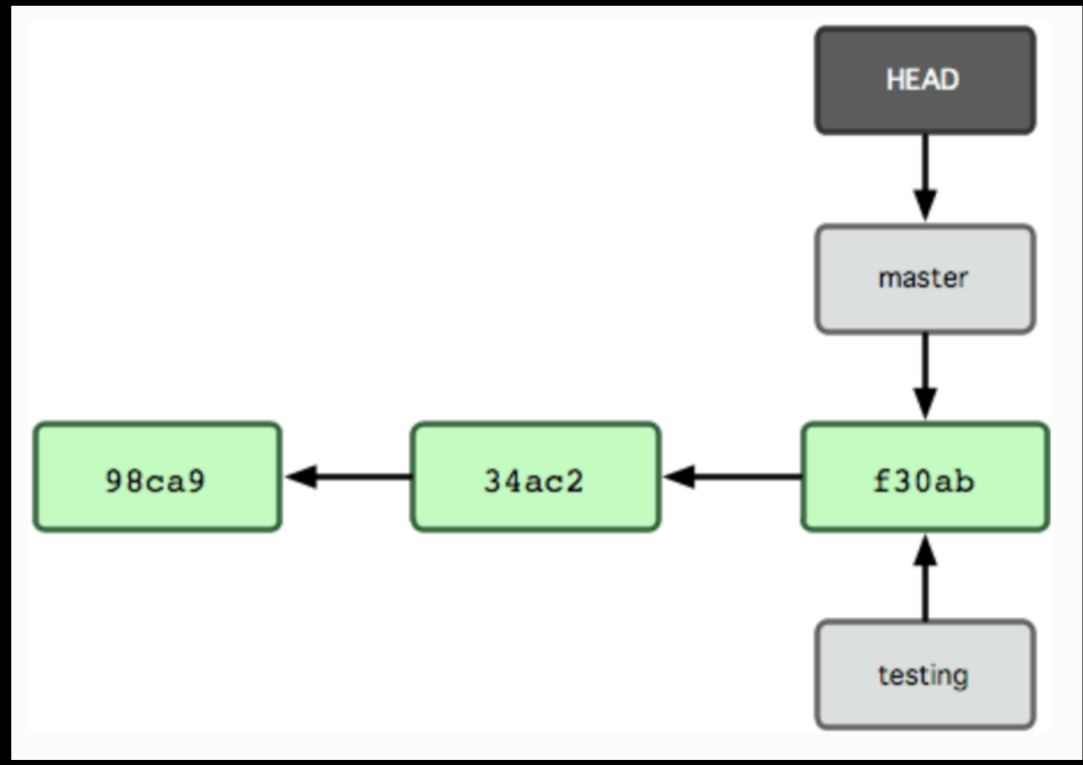
Creating a branch

- Use git branch <name> to create a new branch.
- Your new branch will point to the same commit of the branch you are currently on when you ran the that command



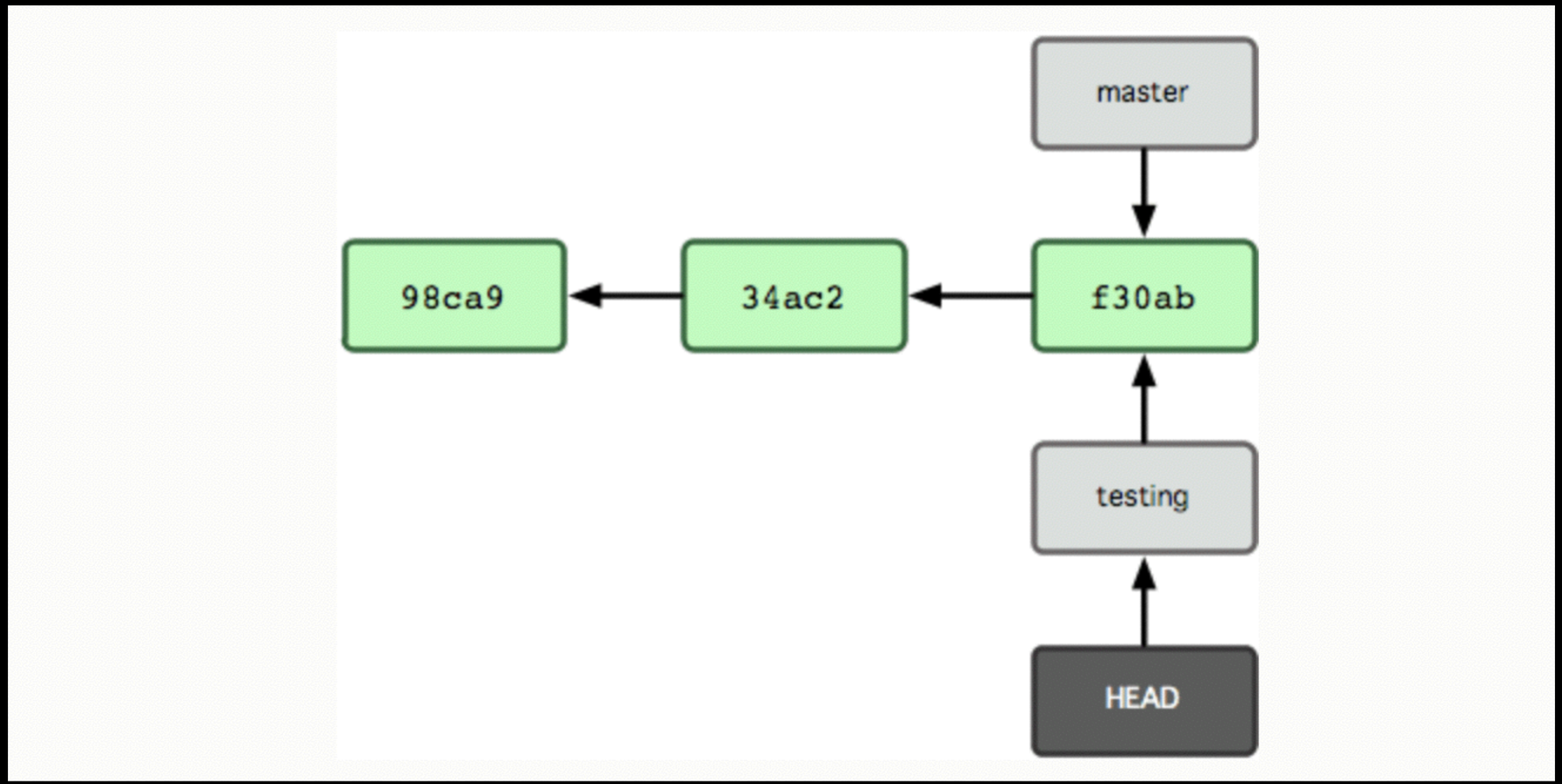
HEAD pointer

- Git keeps a special pointer called HEAD, which points to the current local branch you are on.
- Your working directory reflects the current commit your HEAD is pointing to.
- The branch command does not switch you to your new branch, it just creates it. So we are still on master here.



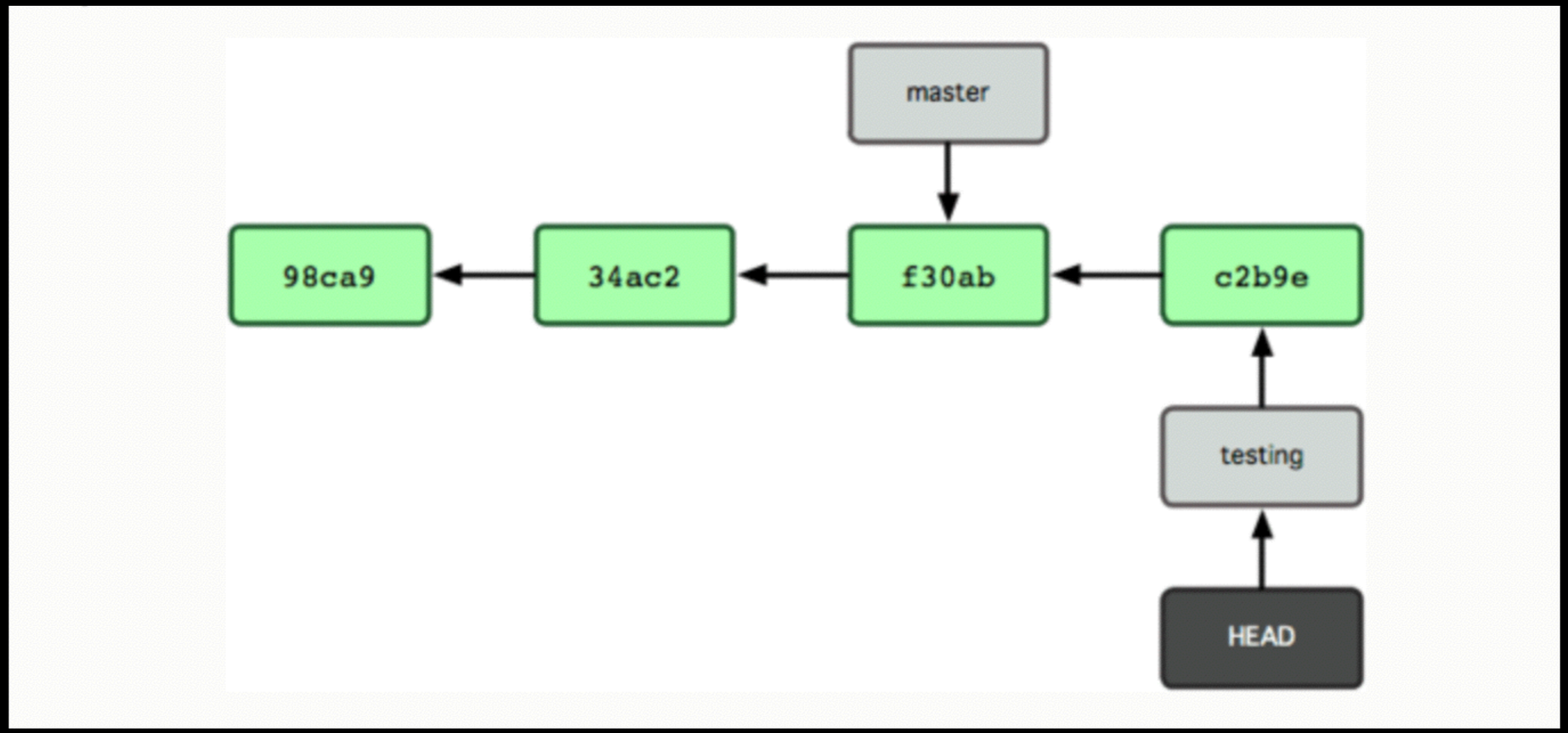
Git Checkout

- Running git checkout <branch name> switches HEAD to point to branch you specify



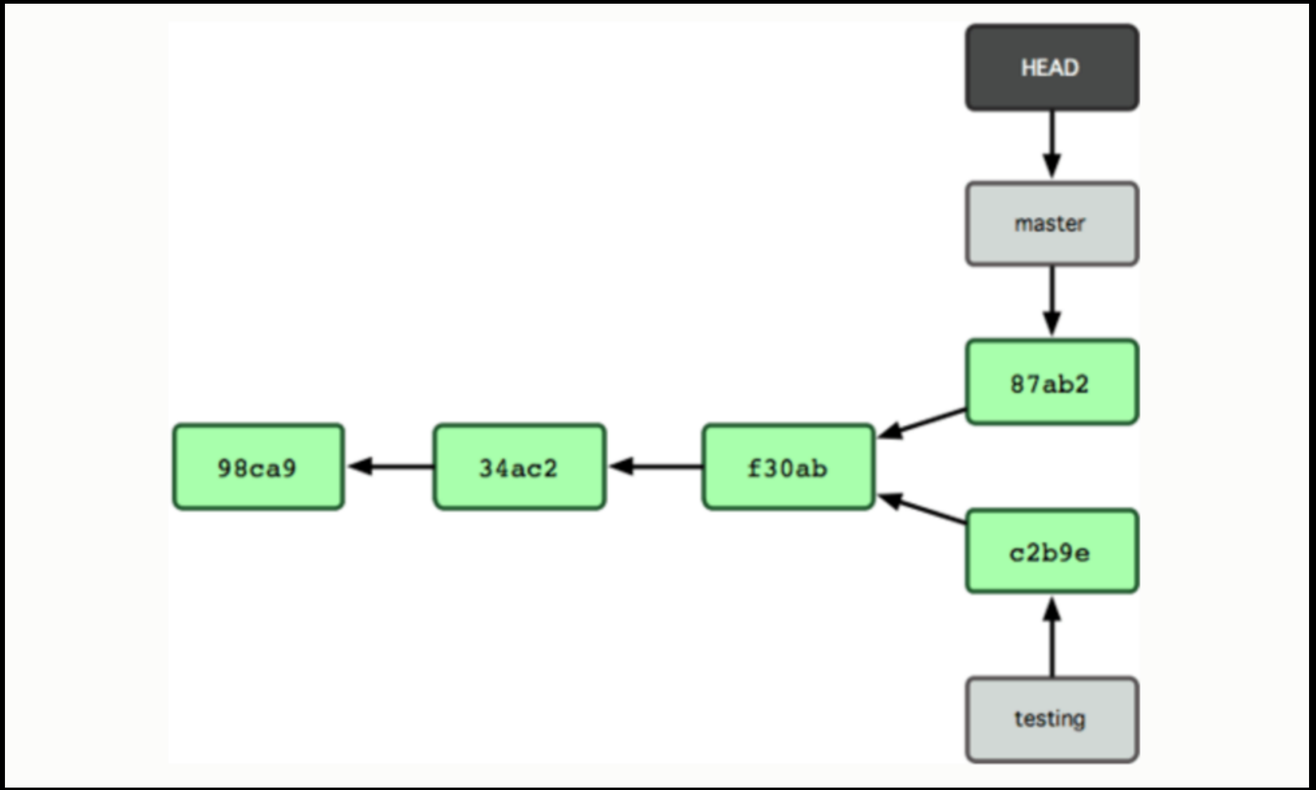
Moving forward

- If you commit now in your testing branch, testing moves forward while master stays put.



Fork in the road

- Now if you checkout master and make commits, we have a divergence.



demo

GitHub

- Github is a repository web-based hosting service.
- Github hosts people's repositories, and those repositories can be public or private.
- The repositories on Github are just like the repositories that you have on your own machine, except Github's web application has additional features that makes working in a team much easier.

- https://help.github.com/articles/set-up-git/

Remotes

- Remote repositories are versions of your project that are hosted on the Internet or network somewhere.
- So when you have a repository on Github, it is considered a remote repository.
- When you import a directory into a git project with git init, eventually you may want to create a remote repository on Github and push to it. This gives you a backup in the cloud, and also lets your work be seen by others.
- Or you can clone an already existing remote repository (your own or someone elses) to get a local copy of the remote repository on your machine.

GitHub and Git

- Github and your local git install have 2 ways of communicating:
  - https (recommended)
  - ssh
- Both of these forms of communication require authentication.
- For https, it is recommended you use a credential helper so you don't have to enter in your credentials every time you interact with a remote repository.
- For SSH, you can generate SSH keys on your computer and then register those SSH keys to your Github account.
- Lets all follow the steps at https://help.github.com/articles/set-up-git/ together to get our github & git properly setup.

demo

Git Remote Commands

- git remote -v shows you all the remotes you have configured for your local repository on your machine
- use git remote add <nickname> <url> to add a remote repo
- after committing, use git push <nickname> <branch> to push your committed changes to your remote
- use git pull to automatically fetch and merge a remote branch into your current local branch

Demo

Git Clone

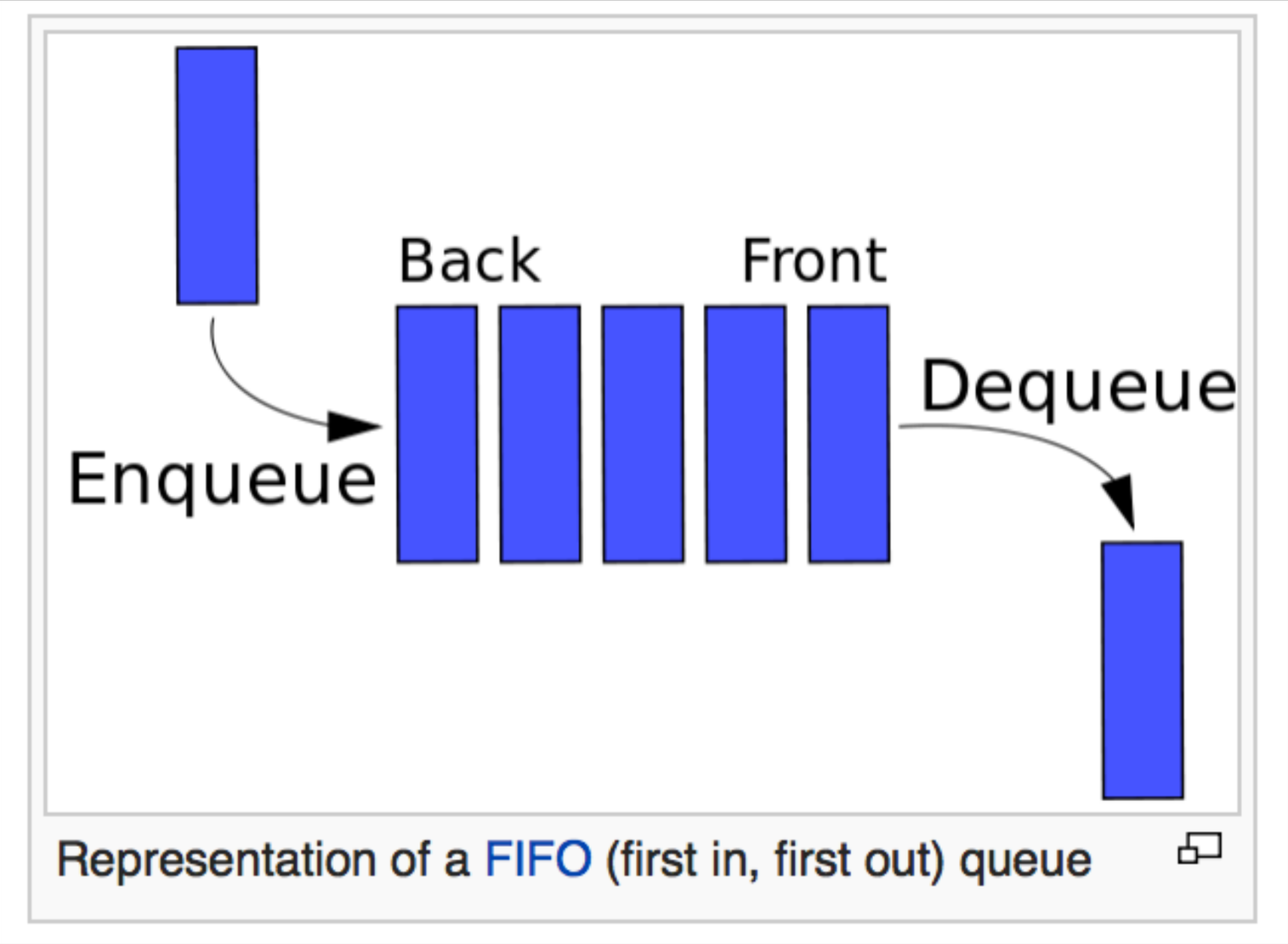
- The git clone command clones a repository into a newly created directory, and creates remote tracking branches for each branch in the cloned repository (view them with git branch -r)
- **A remote branch is slightly different from a local branch.** The remote branches are just references to the state of branches on your remote repository. You don't move them yourself, they move automatically whenever you do any network communication with your remote.
- But the git clone command sets up remote tracking branches, which are local branches that have a direct relationship to the remote branches. You can push and pull from those tracking branches and they will automatically know which remote branch to work with.
- Cloning a repository gives you the complete history of that original repository.
- Cloning also automatically creates a remote that points to the repository you cloned from. Hooray!

Queue Data Structure

Queues

- From Wikipedia: “In computer science, a queue is a particular kind of data structure in which entities in the collection are kept in order and the principal operations on the collection are addition of entities to the rear (known as enqueue) and removal of entities from the front position (known as dequeue)”

Queue Visual



Representation of a FIFO (first in, first out) queue

What are Queues used for?

- A queue is great for any situation where you need to **efficiently maintain a First-In-First-Out order of some grouping of entities.**
- Servers use queues as a fair policy when responding to requests. They might be getting 10,000's requests a second, and they respond to those requests in a first come first serve fashion using a queue.
- CPU's use queues to properly schedule operations to run based on priority

Implementing a Queue

- Implementing a queue is pretty similar to a stack
- Can be implemented using an array or linked list (linked lists are in week 3)
- Just need to implement dequeue (removing) and enqueue (adding)
- Queues can have a peek operation as well (sometimes called front), which simply returns the front most entity.

Demo