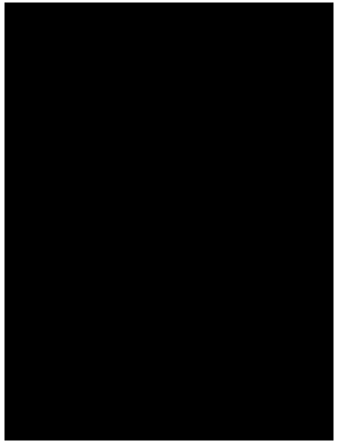
How Flipkart scrolls at 60fps



THE TEAM

The problem

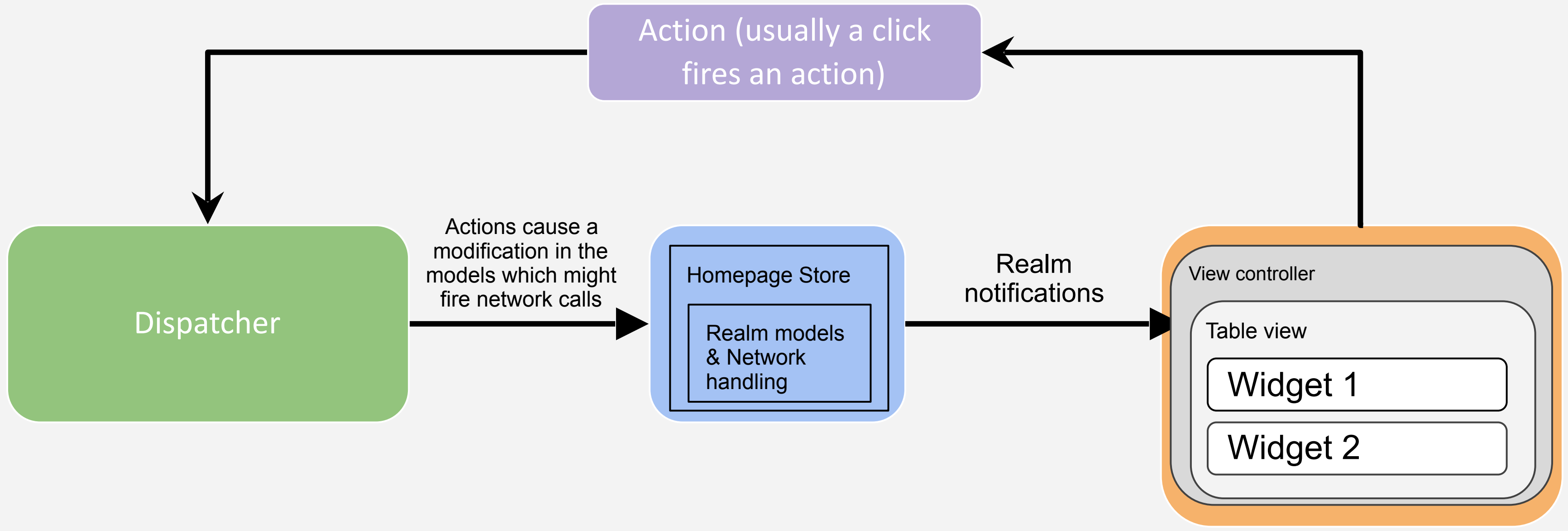
1. Home Page load time was high
    - Unoptimized store/data layer
    - High Cache (DB) fetch time

1. Home Page scroll was janky
    - Rendering time for UI blocks (widgets) was high
        - Caused mostly by complex view hierarchy
    - Tracking on main thread, Swizzling of methods

The problem

1. Home Page load time was high
    - ~~Unoptimized store/data layer~~ Switched from MVC to Flux(ish) + cache first strategy
    - ~~High Cache (DB) fetch time~~ Switched to Realm from CoreData

Flux : In our context


Action (usually a click fires an action)
Dispatcher
Actions cause a modification in the models which might fire network calls
Homepage Store
Realm models & Network handling
Realm notifications
View controller
Table view
Widget 1
Widget 2

The Benefits

- Flux pattern:
  - Unidirectional flow of data
  - UI as a function of states
  - Only stores can mutate states
  - UI can only fire actions
- Realm:
  - 3X faster than core data
  - Threading errors are caught early and predictably
  - Provides change notifications at object level
  - Fits flux paradigm perfectly
  - The speed benefit made us use it as if it’s an in-memory DB

The problem

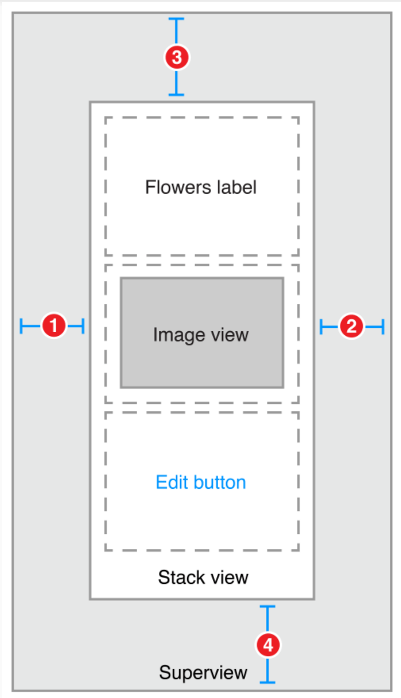
1. Home Page load time was high
    - Unoptimized store/data layer
    - High Cache (DB) fetch time

1. Home Page scroll was janky
    - Rendering time for UI blocks (widgets) was high
        - Caused mostly by complex view hierarchy
    - Tracking on main thread, Swizzling of methods

The UI layer : First attempt : Autolayout

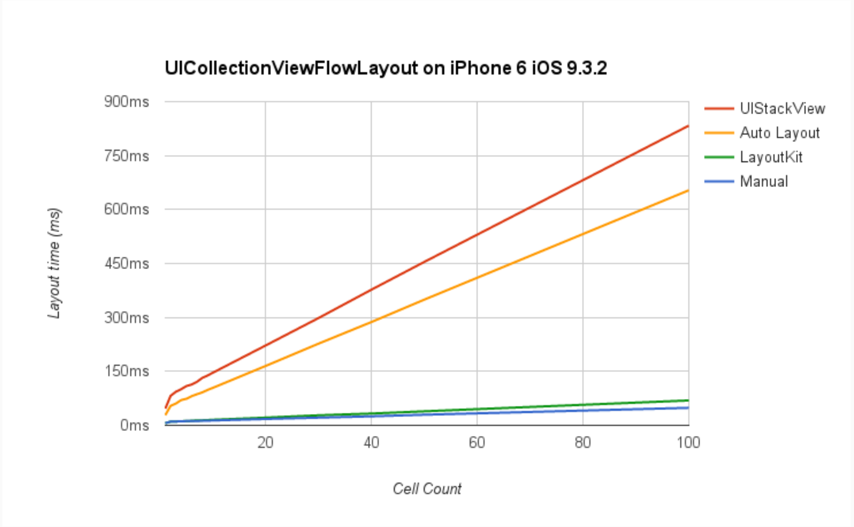
- Used apple's own auto layout system
- It internally uses Cassowary constraint solver
- Slow to build due to constraint breakages
- Slow to run since the equation solver is at runtime

Result : Janky Scrolling



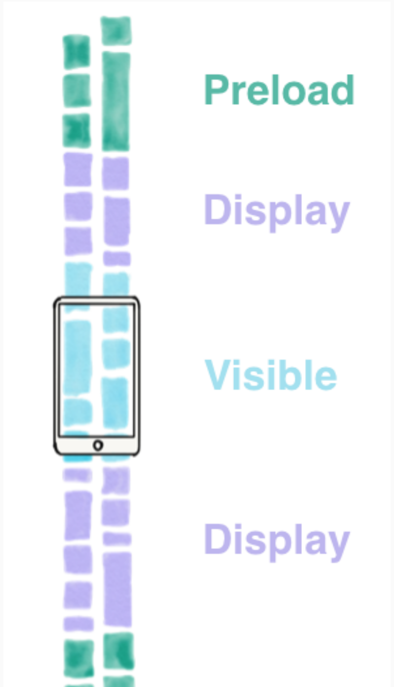
The UI layer : Second attempt : LayoutKit

- Used LinkedIn's LayoutKit library
- Android like way of building layout
- Code written in Swift 3.0
- Easy to build
- Faster than auto layout
- But still suffered from frame drops



The UI layer : Final attempt : AsyncDisplayKit

- Facebook's ADK library (renamed to texture)
- Code compatible with Swift 3.0
- Easy to build
- Faster than layoutkit and autolayout
- Performs layout and draw in background thread using "Nodes"
- Full 60 frames per second.
- Supports UI as function of states
- Intelligent preloading, Multiplex Images & view flattening



Video demo

AsyncDisplayKit
(No frame drops)
iPhone 5C scroll performance

Developer benefits

- Easier to write new widgets.
- Scroll performance is guaranteed to remain at 60fps.
- Widget UI will scale for different container sizes and multi column layouts easily.
- Image prefetching and request cancellation out of the box.
- Image multiplexing out of the box.
- Realm & ADK have very strict Assertions. More debug build crashes. Less production crashes.

Customer benefits

- Cached version of the page is served instantly, even if offline. No empty screen.
- Widgets which expired are reloaded in a smooth transition (ADK magic!)
- Less time spent on page load due to faster parsing. Average session length is shorter

Business benefits

Old **New**

Transactions per visit (in %)	+1.52% (6 bps)
Units per visit (in %)	+1.02% (5 bps)
Revenue per visit	+1.1%

Crashfree sessions for refactored app are 99.8% (as compared to 99.4% for earlier versions)

!(All is well)

1. We depend on ADK/Texture/Realm to fix crashes in their SDK which are affecting our app
2. There are things like height resize during call or hotspot which get taken care of automatically if you write correct autolayout constraints
3. Realm requires all your model classes to be inherited from their class and all params written in a certain way.
4. Unlike autolayouts, there isn’t a designer available for ADK/Texture