ARCHITECTURE

WIKIPEDIA

Architecture is both the process and the product of planning, designing, and constructing buildings or any other structures. Architectural works, in the material form of buildings, are often perceived as cultural symbols and as works of art. Historical civilisations are often identified with their surviving architectural achievements.

早上好

FINDING NEMO

@ANDYYHOPE

IOS HAS AN ARCHITECTURE PROBLEM

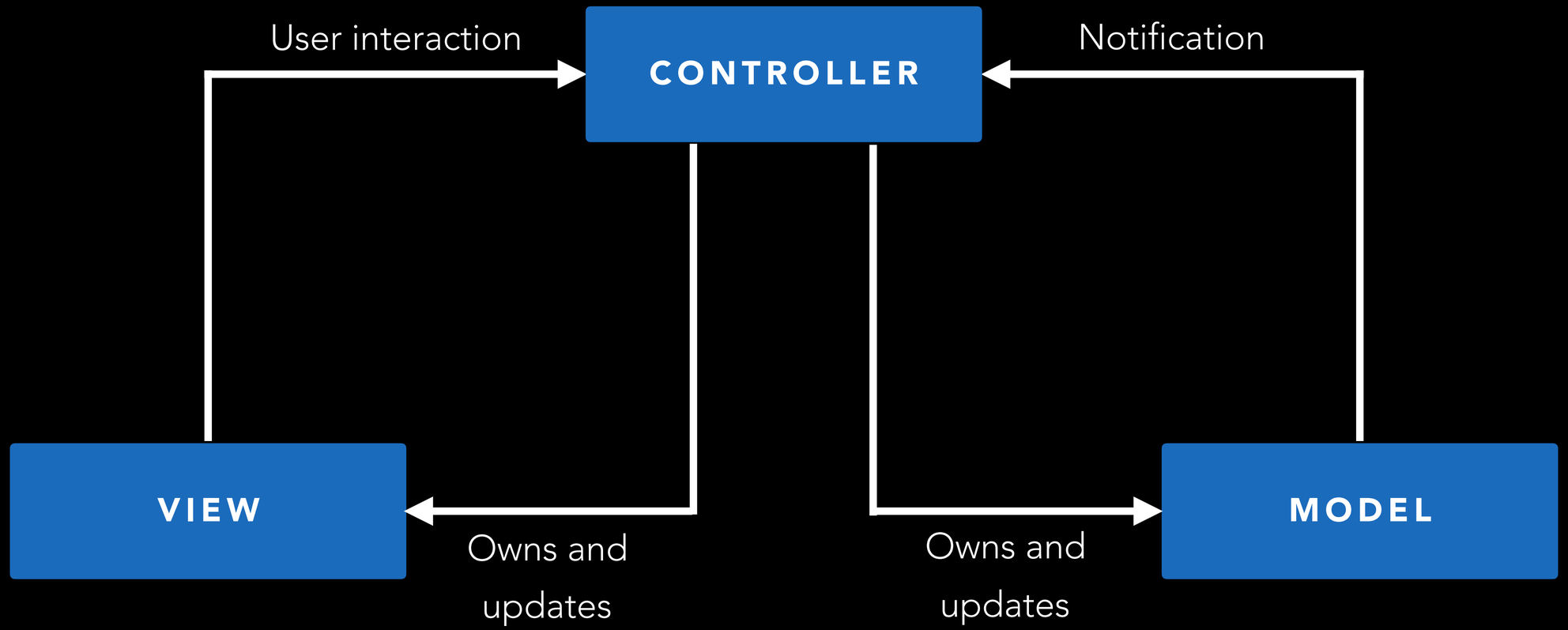
(KINDA...)

MVC: MODEL-VIEW-CONTROLLER

1970S

- Traditional architecture for GUI applications
- Used in different environments
  Java, ASP, C#, PHP, all the classics
- **Model** layer represents data
- **View** is what the user sees
- **Controller** is for View interaction or Model manipulation

MVC: MODEL-VIEW-CONTROLLER



MVC: MODEL-VIEW-CONTROLLER

THE GOOD

- Simple
- Easy to use and understand
- Officially supported by Apple frameworks

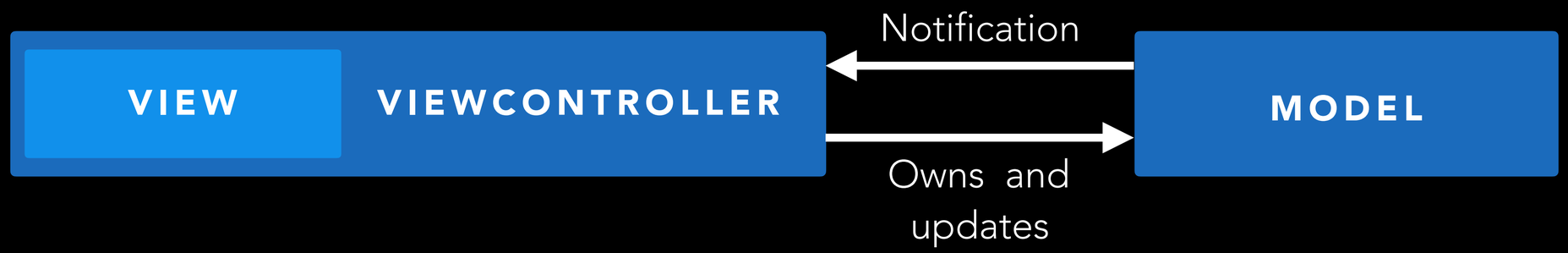
MVC: MODEL-VIEW-CONTROLLER

THE BAD

- Hasn’t aged well
- Data isn’t always readily available
- Logic has become more complex
- View and Controller are bound by UIKit (UIViewController)
- Difficult to write tests for
- Suffers from Massive View Controller
- Difficulties with scale

MVC: MODEL-VIEW-CONTROLLER

REALITY



MVC: MODEL-VIEW-CONTROLLER

SPIN-OFFS

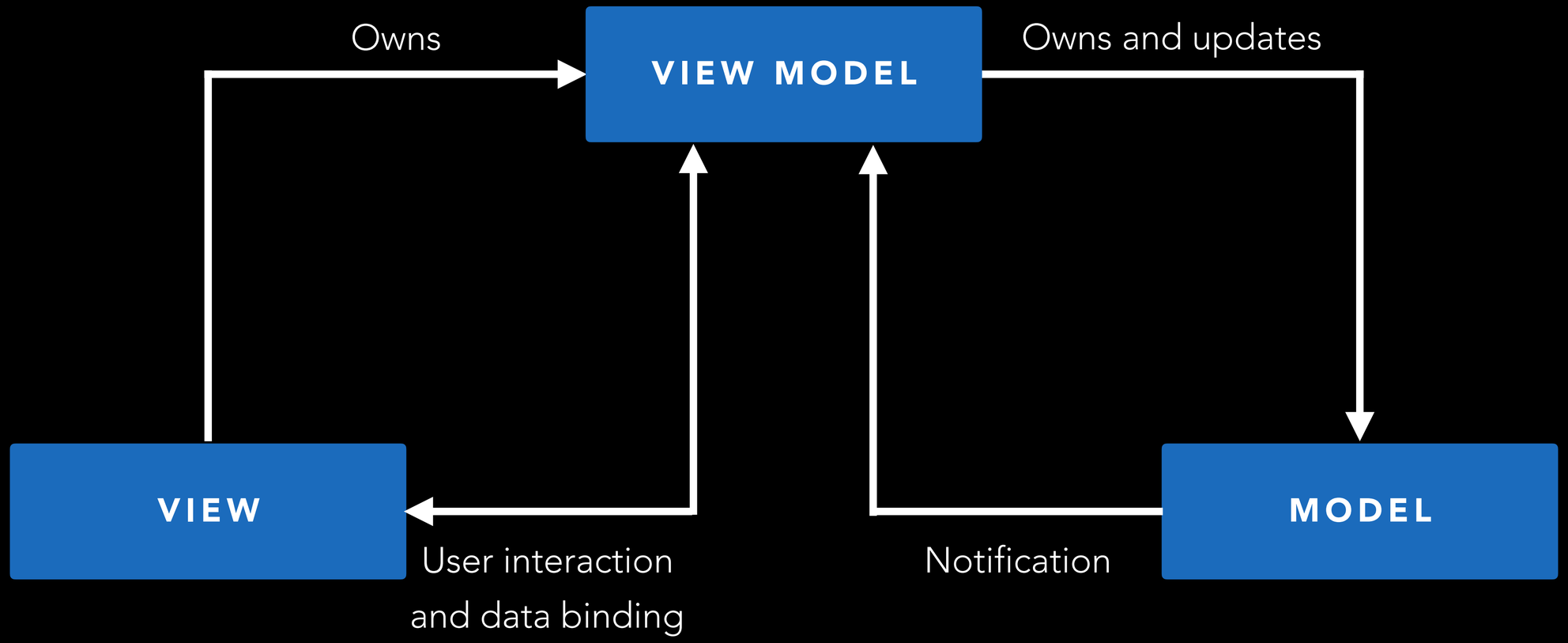
- Hierarchical Model-View-Controller
- Model-View-Adaptor
- Model-View-Presenter
- Model-View-ViewModel

MVVM: MODEL-VIEW-VIEWMODEL

2005

- Built for event-driven programming of user interfaces (2005)
- Designed by Microsoft
- **Model** represents business logic
- **View** represents what the user sees and interacts with
- **ViewModel** converts the Model for View representation

MVVM: MODEL-VIEW-VIEWMODEL



MVVM: MODEL-VIEW-VIEWMODEL

THE GOOD

- Reactive UI data binding
- Increased testability!
- First viable contender to MVC (within iOS)
- Post-MVC movement on iOS primarily started by Ash Furrow

MVVM: MODEL-VIEW-VIEWMODEL

THE BAD

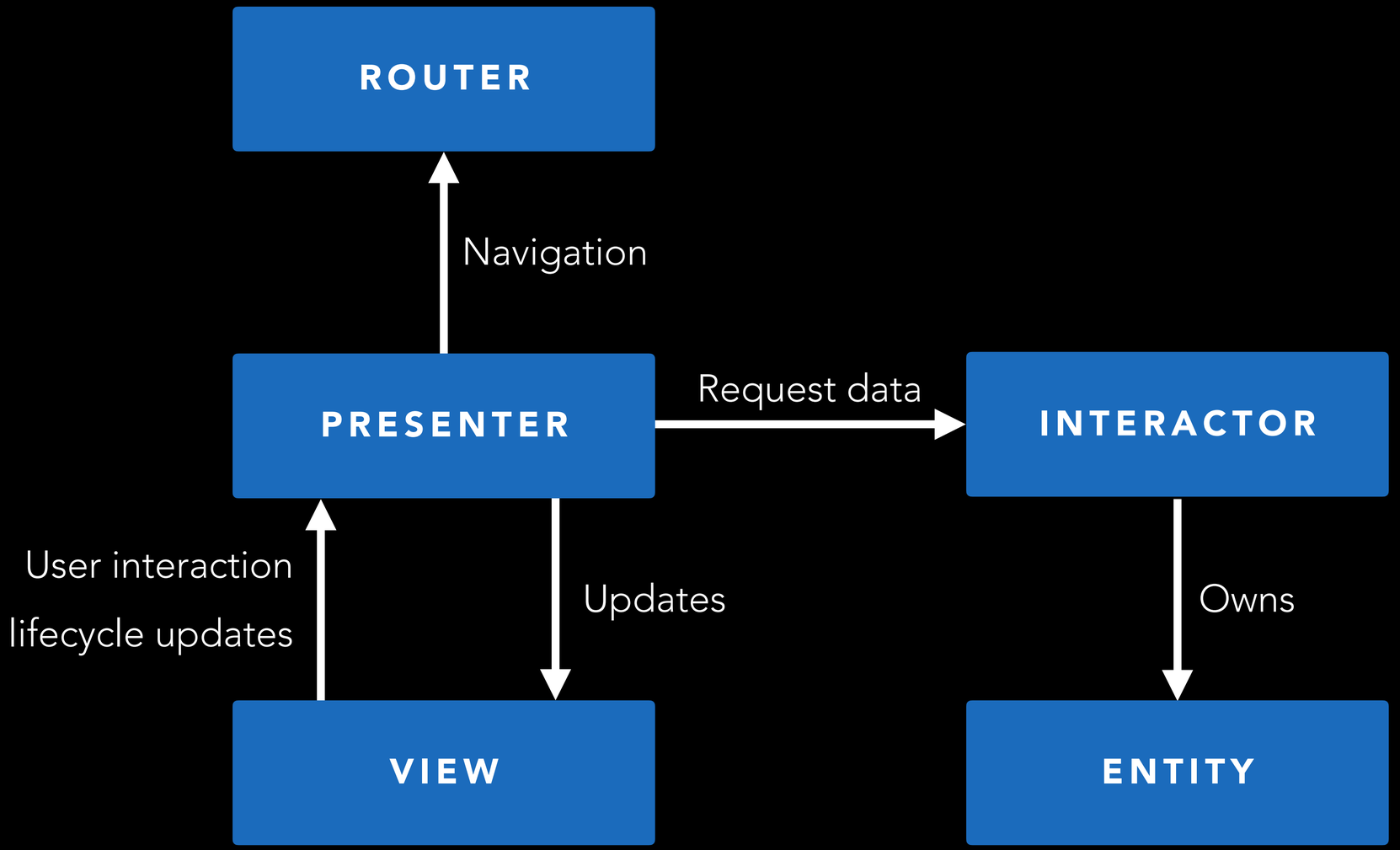
- *Very* idealogical
  - Community has different ideas on implementation
  - Some felt that the ViewModel shouldn't import UIKit
  - Some suggested putting network request logic inside the ViewModel
- Felt like we were trying to wedge it into iOS

VIPER

2012

- Derivative of Uncle Bob’s (Robert C. Martin) Clean architecture (2012)
- **View** presents user interface to user
- **Interactor** performs business logic
- **Presenter** works with Interactor and prepares the data
- **Entities** represents data models
- **Router** provides ability to move between scenes

VIPER


ROUTER
Navigation
PRESENTER
Request data
INTERACTOR
User interaction
lifecycle updates
Updates
Owns
VIEW
ENTITY

VIPER

THE GOOD

- Has a really cool name
- Follows *great* programming practices
- Well-defined separation of responsibility
- Modular
- Testable

VIPER

THE BAD

- Presenter is tightly coupled with View
- Spaghetti code
- Lots of classes
- Difficult to onboard engineers

GRAPH OF ARCHITECTURE

OVER-SIMPLIFIED

HOW DO WE DECIDE ON WHICH TO ADOPT?

OUR PRODUCT

PUNTERS APP

- Consumer focused
- News articles
- Statistical data
- Community platform for enthusiasts

- 130+ Screens
- Reused views
  - Interactive
  - Updated via web sockets
- Backend For Frontend (BFF) server architecture

CHOOSING AN ARCHITECTURE

REQUIREMENTS

- Testability
- Code reusability
- Easy to onboard
- Modular
- Expandable
- Compatible

PRINCIPLES

- iOS first and foremost
- Enforce unidirectional flow
- Utilise immutability
- Include reactive elements
- Embrace Swift’s functionality
- Strong guidelines

CONSUME

USING A PRE-EXISTING ARCHITECTURE

PROS

- Plenty of documentation available
- Officially sanctioned or community driven
- Environment integration
- Known limitations and edge cases

CONS

- Doesn't meet all requirements
- Retrofitted design
  - Layers of abstraction

CONSTRUCT

YOLO

PROS

- Potentially meet all requirements
- Design around an ideology
- Plenty of resources to learn from
- Ad-hoc solutions

CONS

- No documentation
- Errors can be time-consuming
- Requires a lot of thought and planning
- Onboarding other developers

WEIGHING THE DECISIONS

MVC

- Simple but not scaleable

MVVM

- Not well defined
- Too ambiguous

VIPER

- Rich in theory
- Poor in practice

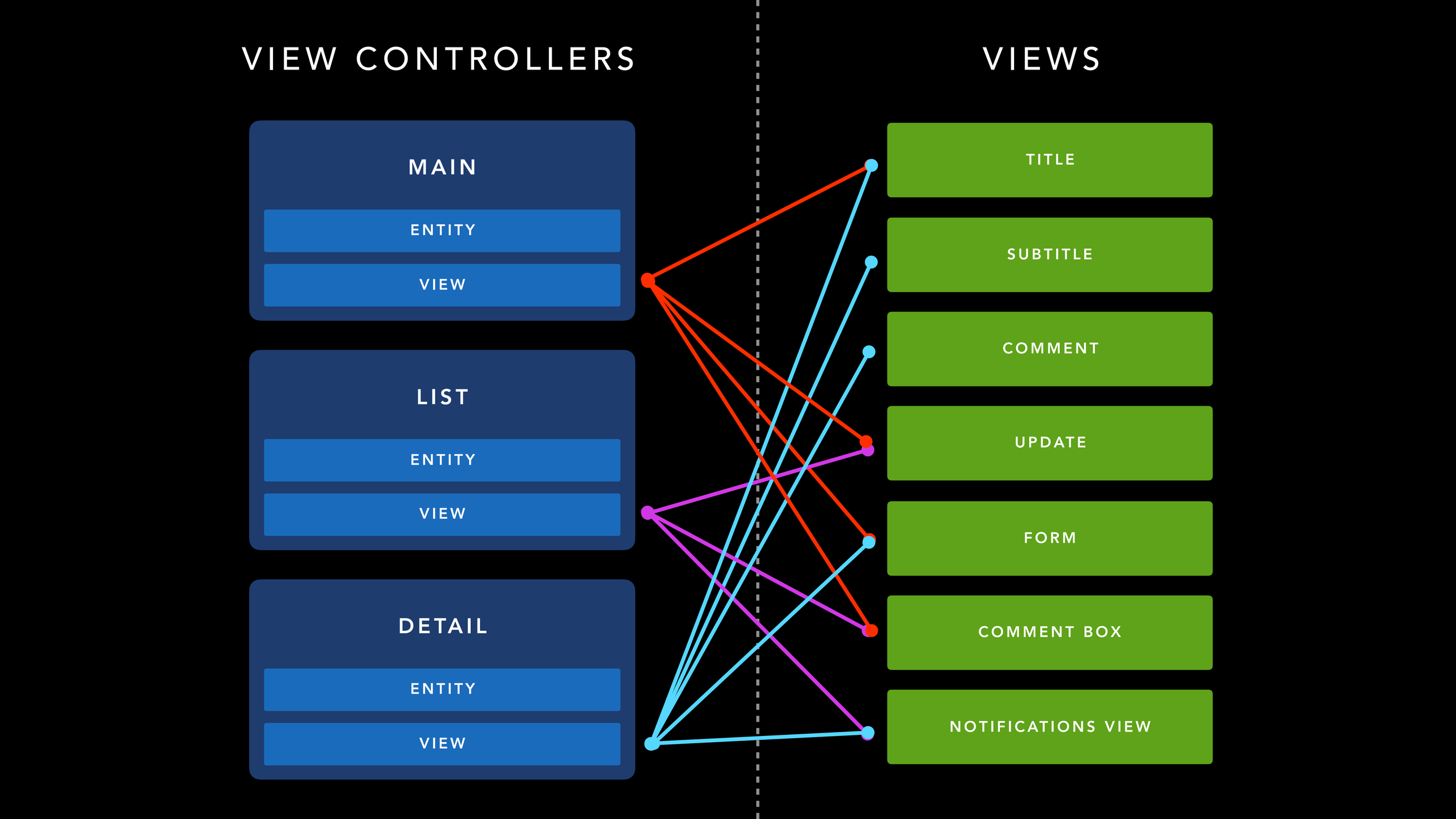
WE'RE GONNA CONSTRUCT

JUSTIFYING OUR DECISION

- We could afford the time to draft and prototype
- Foreknowledge of current and future requirements
- Understanding of edge cases
- Team backed the decision
- Hybridise architectures

DON'T REPEAT YOURSELF

WRITE YOUR CODE ONCE


VIEW CONTROLLERS
VIEWS
MAIN
ENTITY
VIEW
LIST
ENTITY
VIEW
DETAIL
ENTITY
VIEW
TITLE
SUBTITLE
COMMENT
UPDATE
FORM
COMMENT BOX
NOTIFICATIONS VIEW

WHERE DOES THE CODE RESIDE?

LOOKING AT FUNCTIONALITY CALL-SITES

VIEW CONTROLLER (MVC)

ISSUES DETAIL

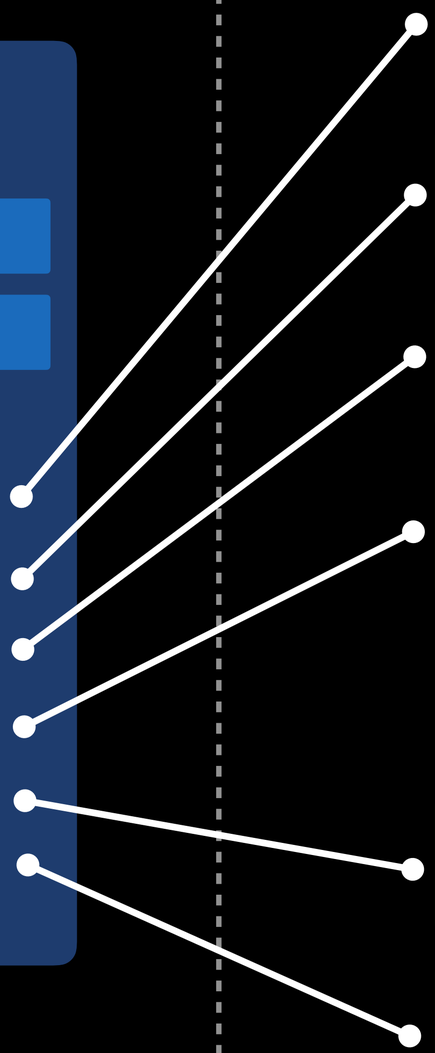
ENTITY

VIEW

```
fetchData()
titleText()
subtitleText()
commentText(forIndex: Int)
updateText(forIndex: Int)
submitComment(String)
subscribeNotifications(id: Int)
```

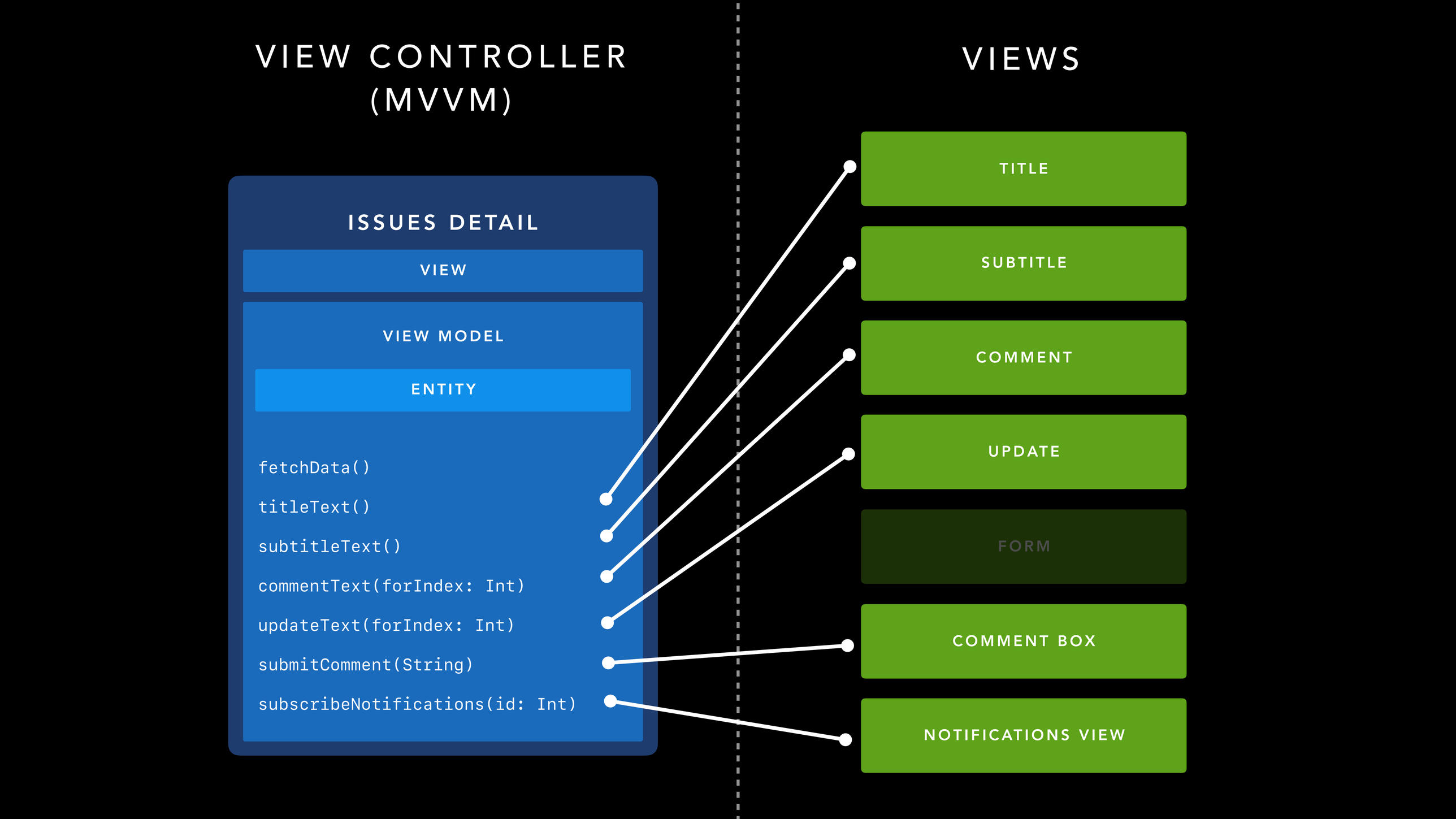
VIEWS

- TITLE
- SUBTITLE
- COMMENT
- UPDATE
- FORM
- COMMENT BOX
- NOTIFICATIONS VIEW



YUCKY

LET’S BREAK IT UP


VIEW CONTROLLER
(MVVM)
VIEWS
ISSUES DETAIL
VIEW
VIEW MODEL
ENTITY
fetchData()
titleText()
subtitleText()
commentText(forIndex: Int)
updateText(forIndex: Int)
submitComment(String)
subscribeNotifications(id: Int)
TITLE
SUBTITLE
COMMENT
UPDATE
FORM
COMMENT BOX
NOTIFICATIONS VIEW

INSPIRATION FROM IOS PATTERNS

UITABLEVIEWDATASOURCE PROTOCOL

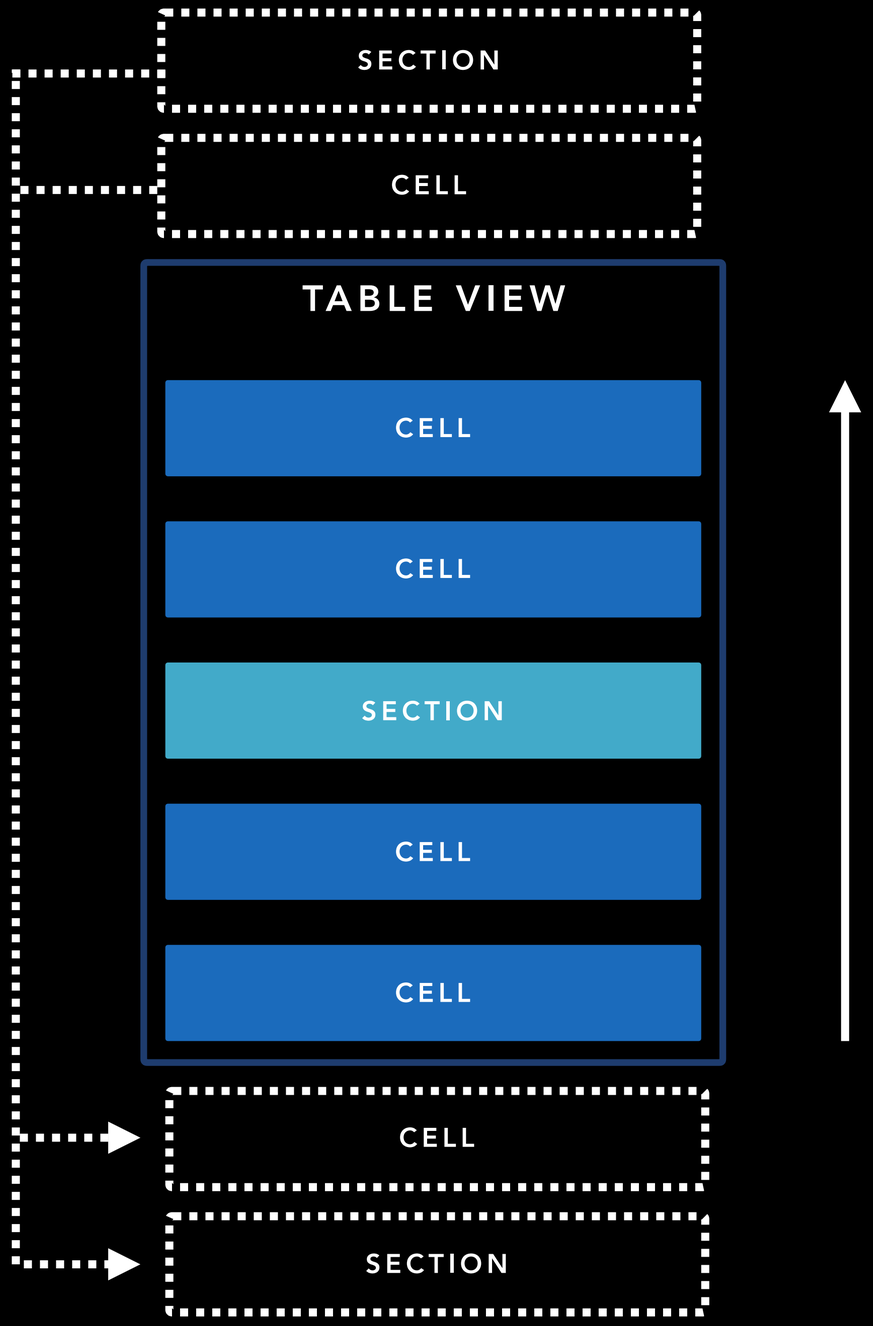
CRASH COURSE

UITABLEVIEW FUNCTIONALITY

IOS TABLE VIEWS 101

FLYWEIGHT PATTERN

- Designed for the original iPhone (2007)
- Highly efficient for memory
- Reuses cells and views as they scroll off-screen



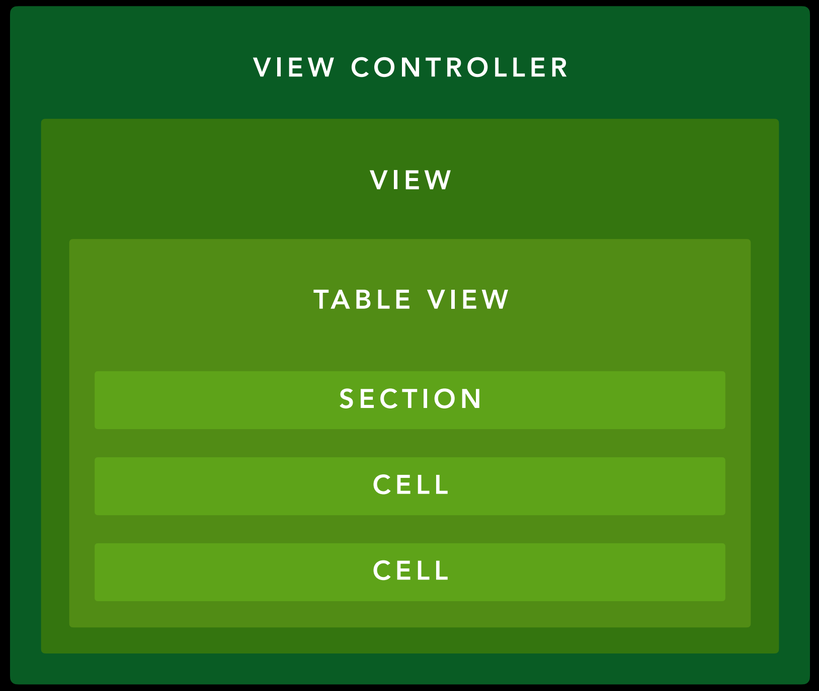
IOS TABLE VIEWS 101

```
numberOfSections(in tableView: UITableView) -> Int

tableView(UITableView, numberOfRowsInSection: Int) -> Int

tableView(UITableView, cellForRowAt: IndexPath) -> UITableViewCell

tableView(UITableView, viewForHeaderInSection: Int) -> UIView?
```



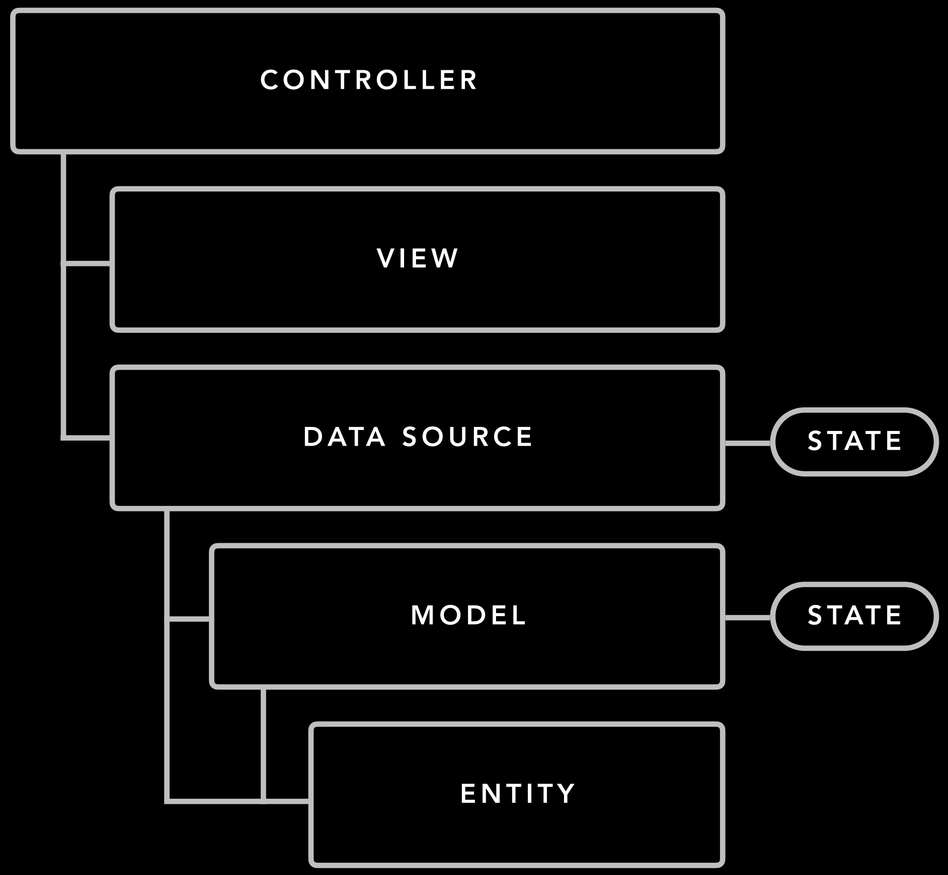
NEMO

IN THE BEGINNING

NEMO

DIVIDED RESPONSIBILITY

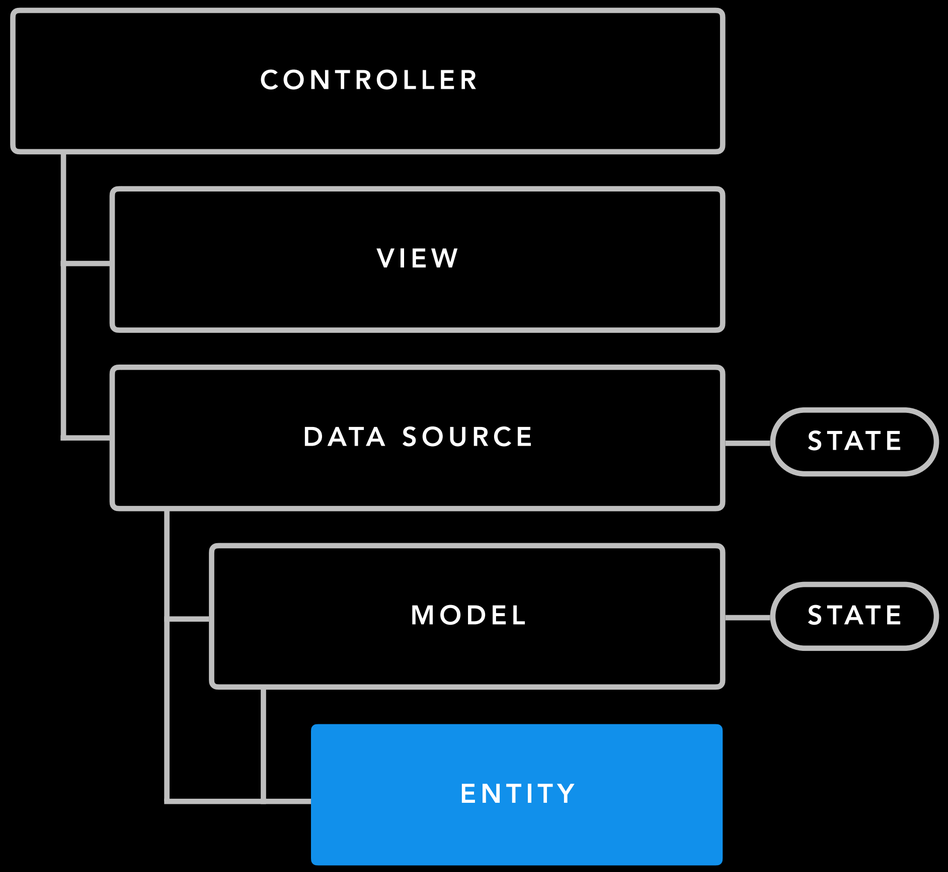
- Controller
- View
- DataSource
- Model (aka View Model)
- Entity
- State



ENTITY

AKA DATA MODEL

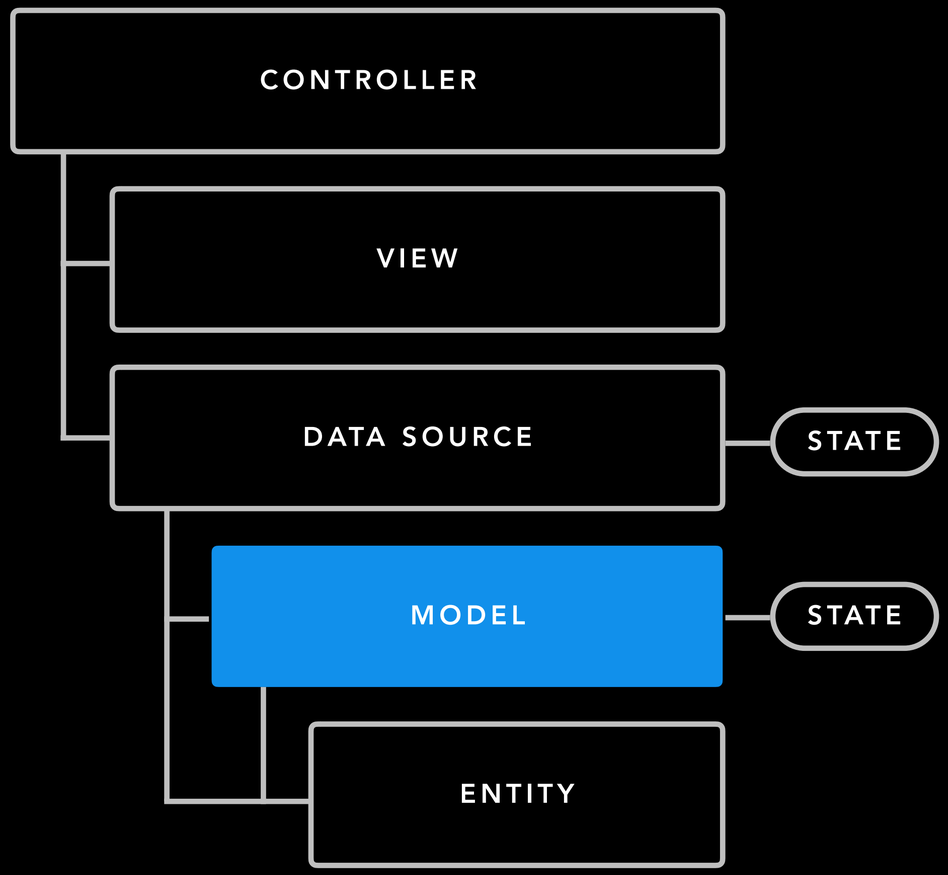
- Converts JSON data into type-safe code
- Doesn’t know about UIKit
- Is immutable
- May have several objects referencing it



MODEL

AKA VIEW MODEL

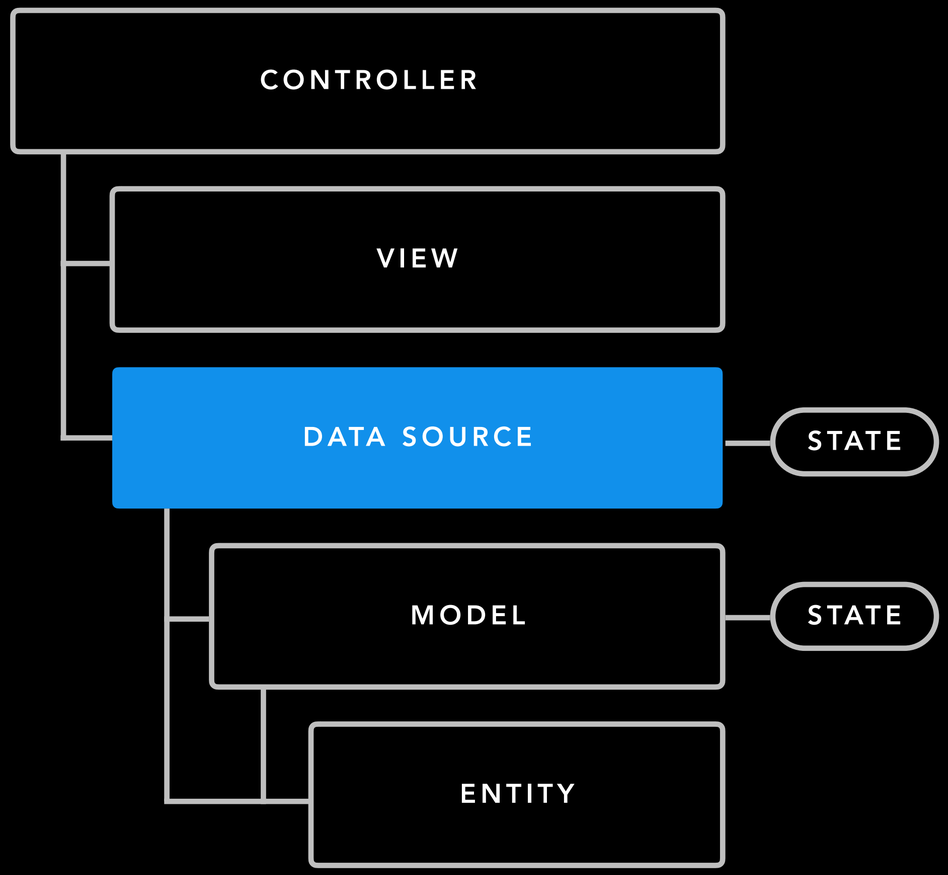
- Converts the Entity object for the user interface (UI).
- Manages State for views. ie, enabling and disabling UI.
- Manages logic for state
- Holds reference to the entity



DATASOURCE

SOURCE OF TRUTH FOR DATA

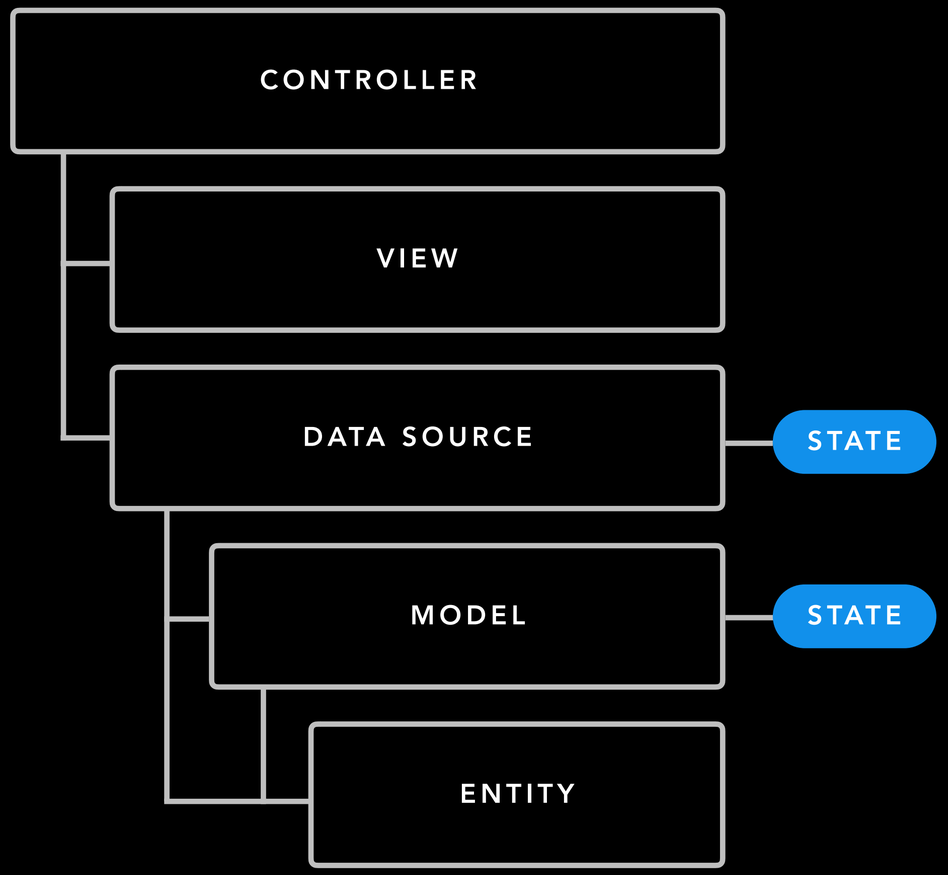
- Serves data to Controllers for consumption
- Holds reference to:
  Sub-Controllers, (View) Model and the Entity
- Handles network requests
- Manages State for network requests



STATE

AKA VIEW MODEL

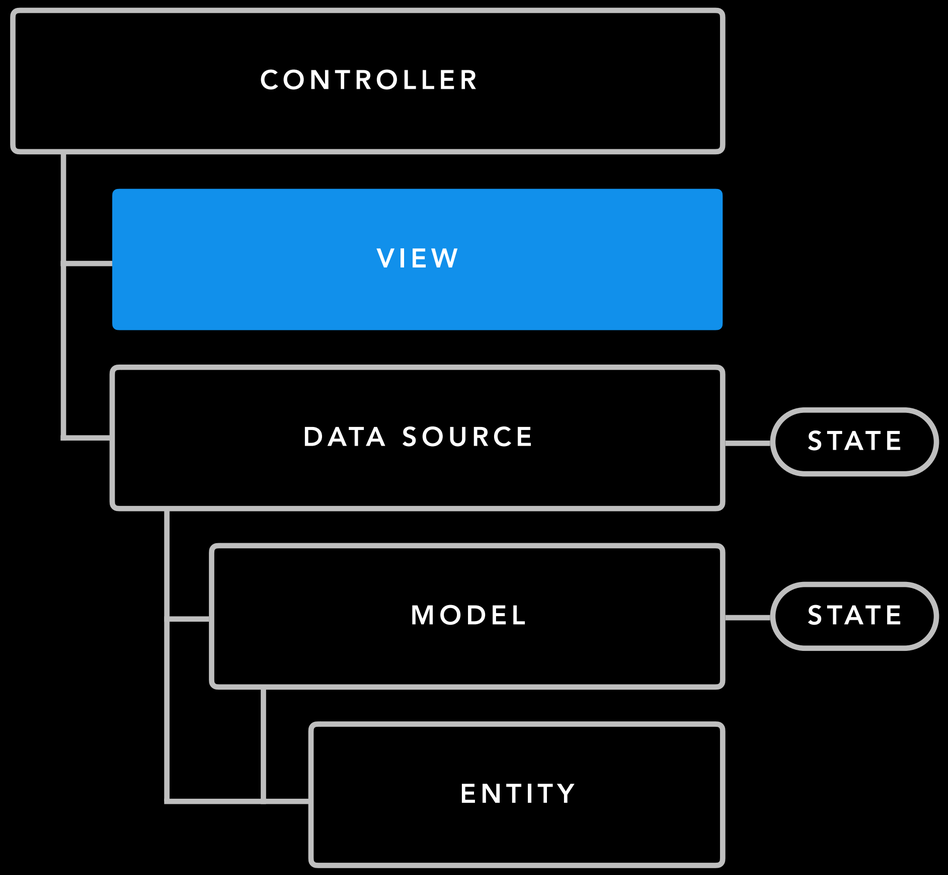
- Two different types of state: Network and View
- Network State (DataSource):
  Loading, Completed, Failed
- View State (Model):
  ie, Enabled, Disabled,



VIEW

NEMO ARCHITECTURE

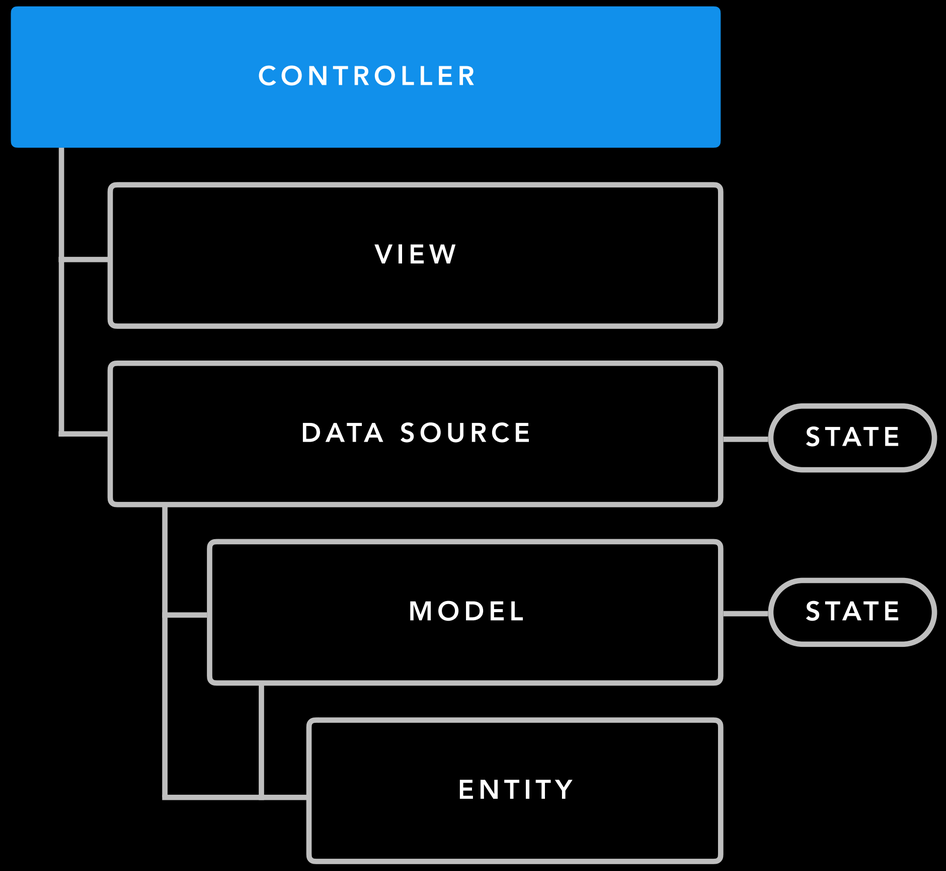
- Is what is presented to the user
- Has no context of how it's used
- Contains no logic
- Has no state

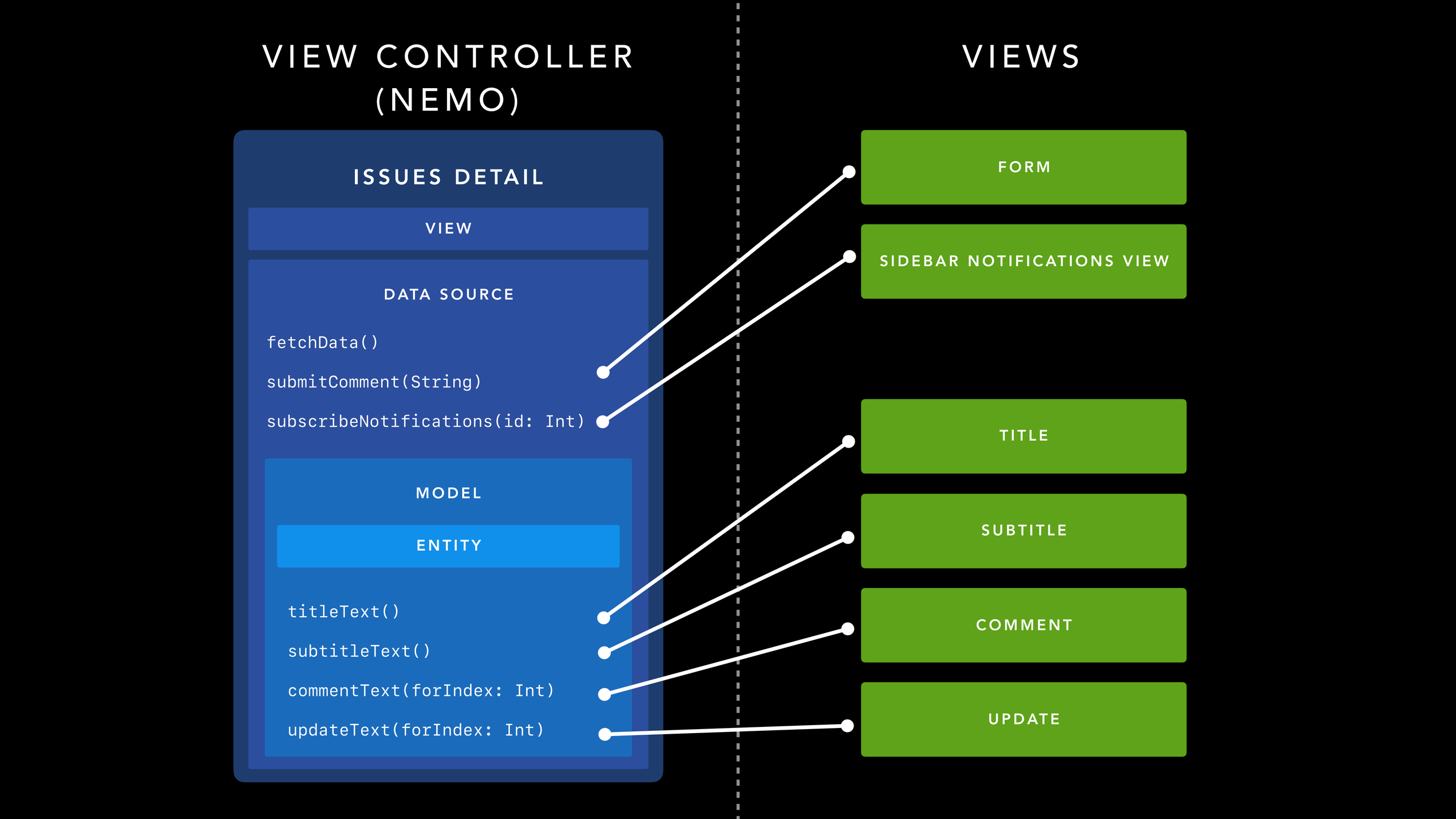


CONTROLLER

NEMO ARCHITECTURE

- Used to presents rows or cells of information to the user.
- Holds reference to the DataSource
- Has access to the Model, Entity and all States
- Has no State
- Passes Views to Sub-controllers
- Comes in three flavours:
  View Controller, Section Controller or Cell Controller



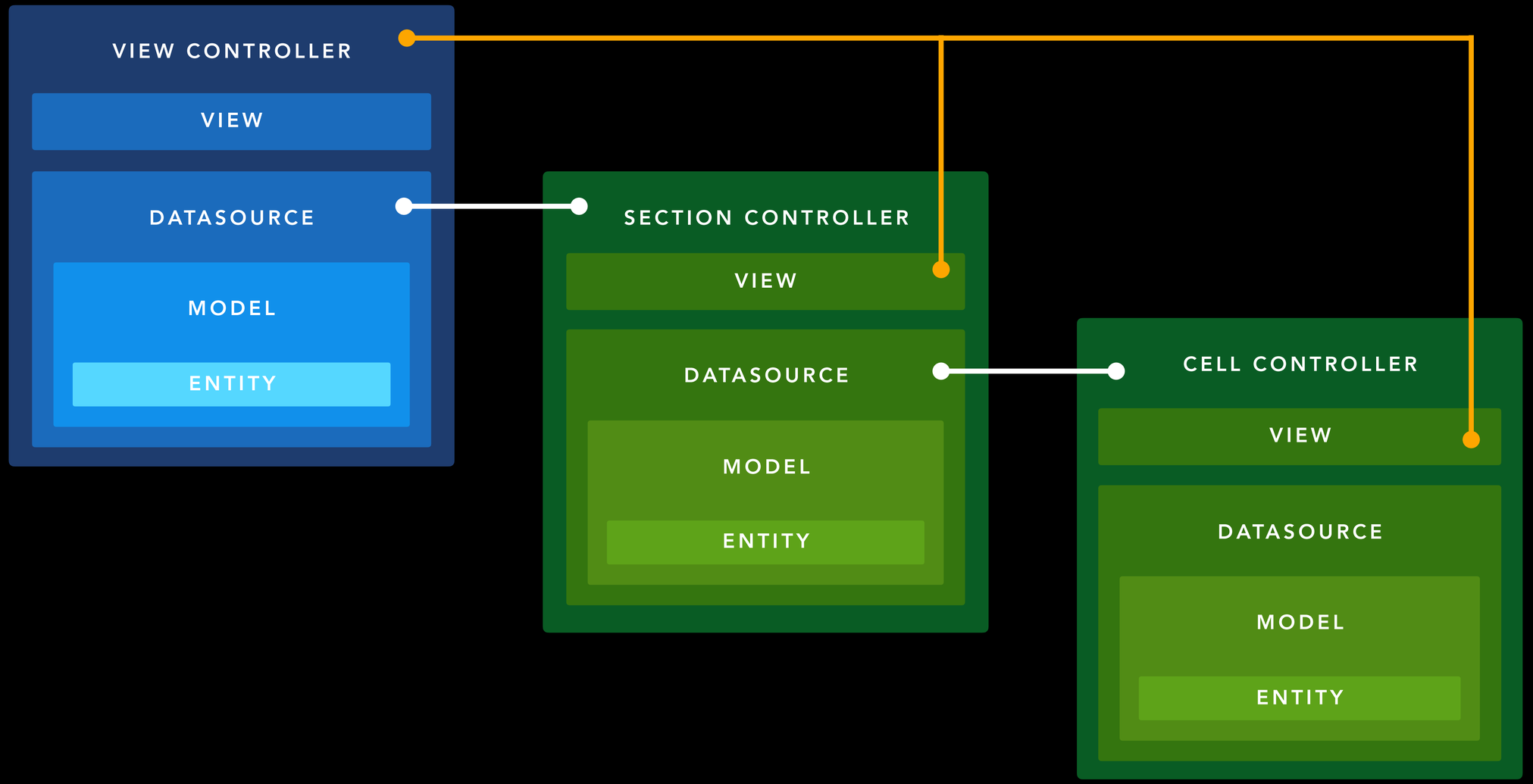

VIEW CONTROLLER (NEMO)
VIEWS
ISSUES DETAIL
VIEW
DATA SOURCE
fetchData()
submitComment(String)
subscribeNotifications(id: Int)
MODEL
ENTITY
titleText()
subtitleText()
commentText(forIndex: Int)
updateText(forIndex: Int)
FORM
SIDEBAR NOTIFICATIONS VIEW
TITLE
SUBTITLE
COMMENT
UPDATE

WE'RE ONTO SOMETHING

MOVING A STEP FURTHER

CONTROLLER

PROMOTIONS



VIEW CONTROLLER

FUNCTIONALITY SEPARATION

View Controller

```
prepare()
navigate(to: Destination)
```

Data Source <StateManageable>

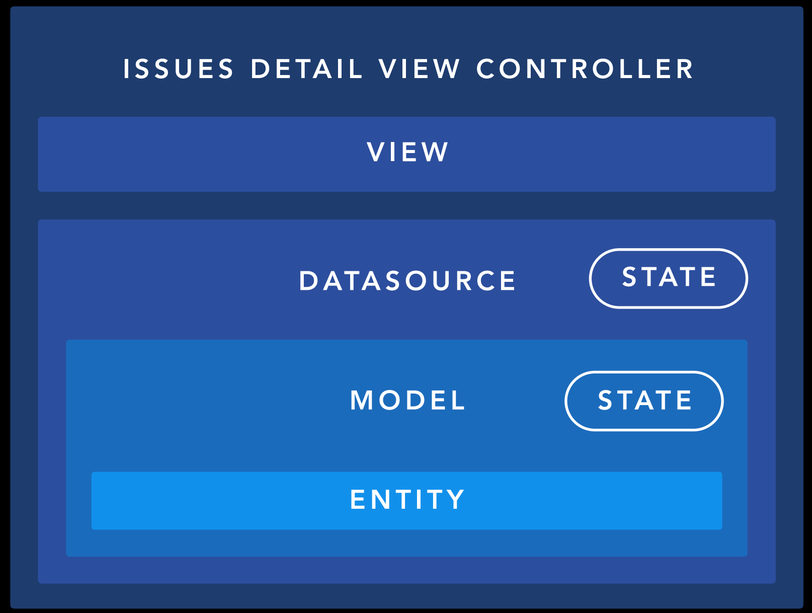
```
State {.loading, .completed, .failed}
requestData()
sectionController(forIndex index: Int)
  -> SectionController
cellController(for index: IndexPath)
  -> CellController
```

Model <ViewStateManageable>

```
ViewState {.light, .dark}
setStyle(_ style: ViewState)
```

View

```
updateStyle(for state: ViewState)
```



SECTION CONTROLLER

FUNCTIONALITY SEPARATION

Section Controller

```
init(SectionEntity)
prepare(_ view: UIView)
```

Data Source

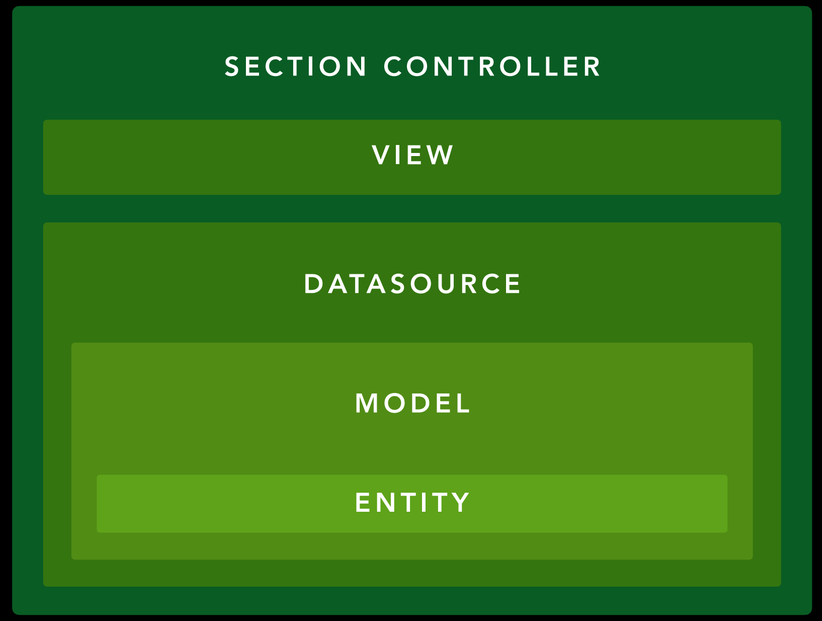
```
cellController(forIndex index: Int)
  -> CellController
```

View Model

```
titleText() -> String
backgroundColor() -> UIColor
```

View

```
setTitle(_title: String)
setBackgroundColor(_ color: UIColor)
```



CELL CONTROLLER

FUNCTIONALITY SEPARATION

Cell Controller

```
init(CellEntity)
prepareBindings(for: UIView)
```

Data Source <StateManageable>

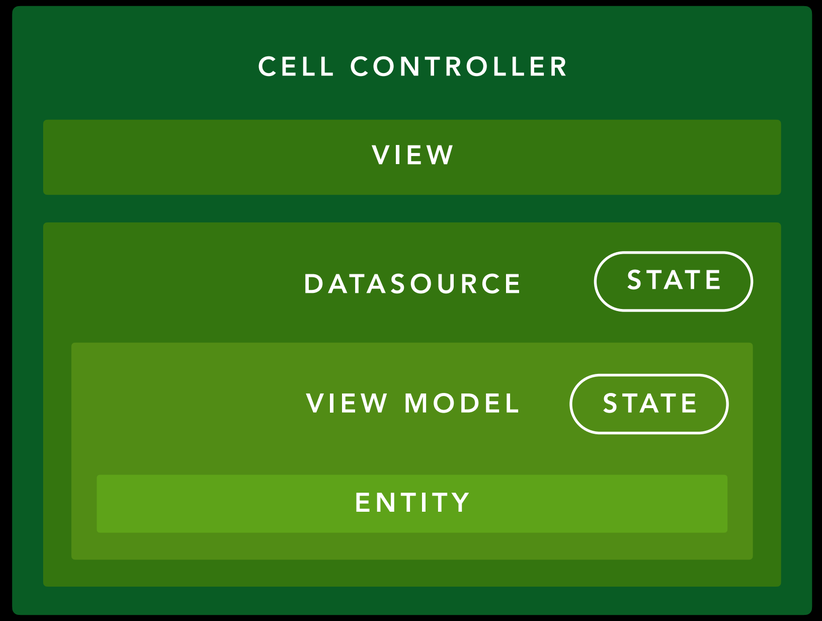
```
State {.loading, .completed, .failed}
updateFormat(Format, for: Range)
submitComment(String)
```

View Model <ViewStateManageable>

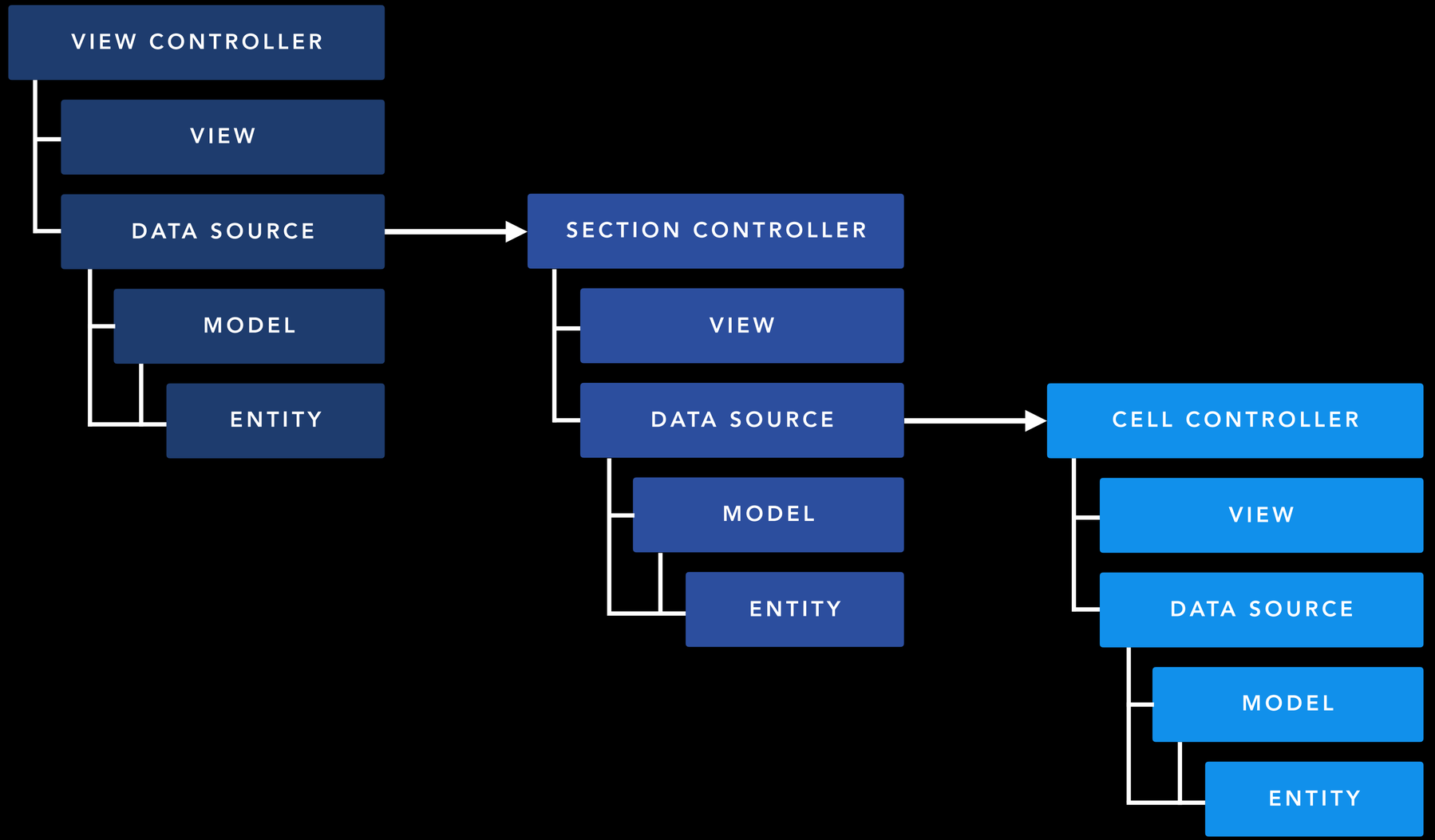
```
ViewState {.editing, .preview}
buttonTitle()
buttonColor()
switch(to: ViewState)
```

View

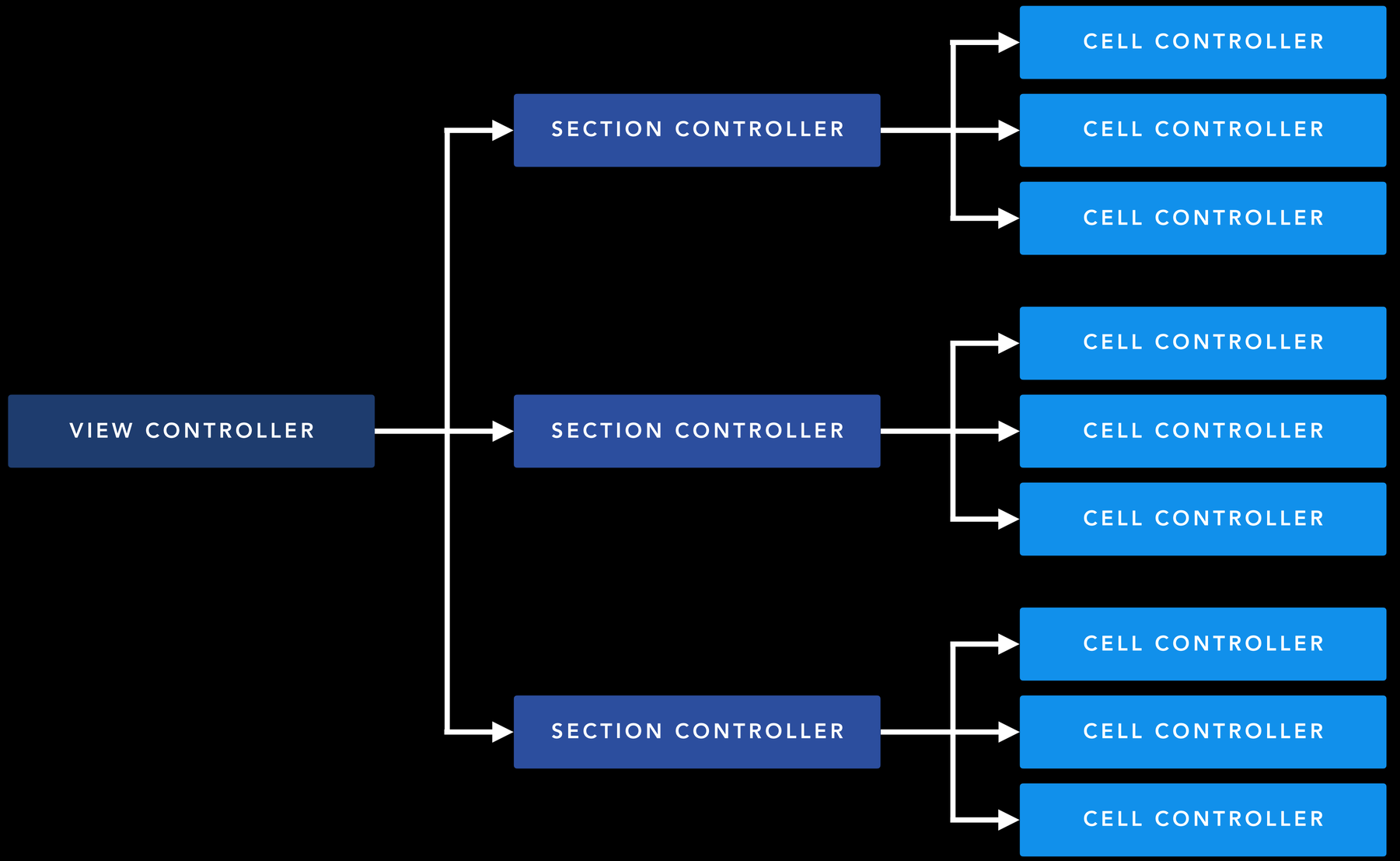
```
setBackgroundColor(UIColor)
update(to: ViewState)
```



SIMPLIFIED HIERARCHY



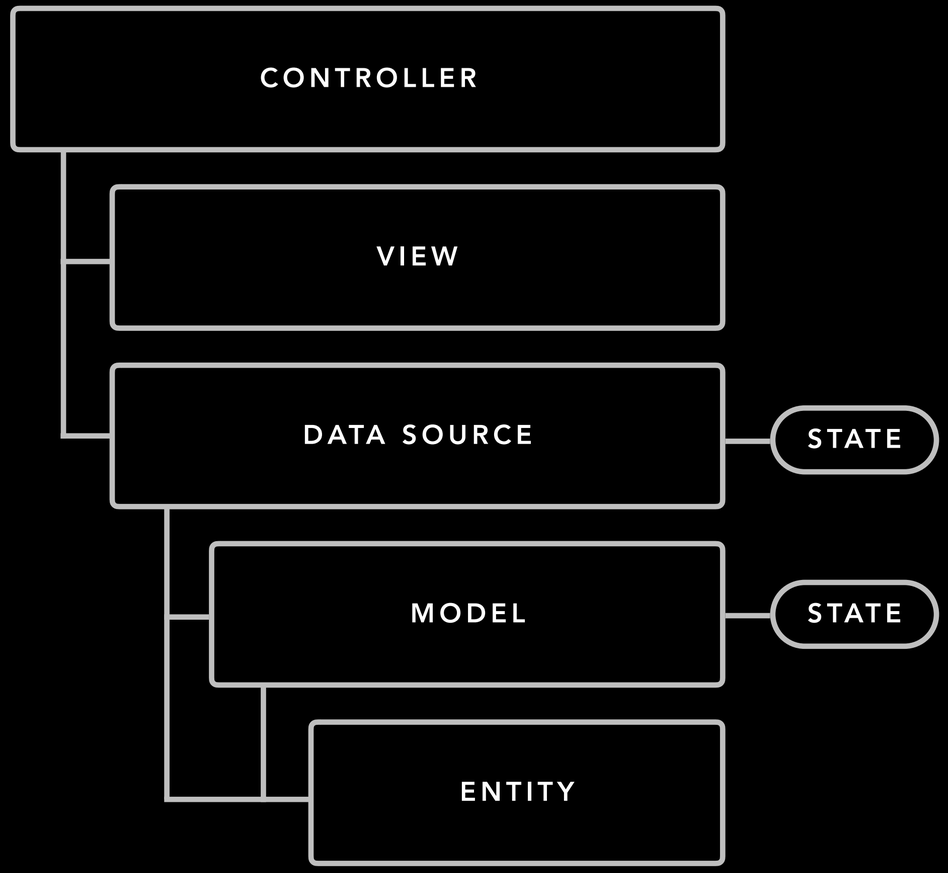
CONTROLLER HIERARCHY



BENEFITS

NEMO

- Portability
- Extensibility
- Modularity
- Dynamic user interface



DEMO

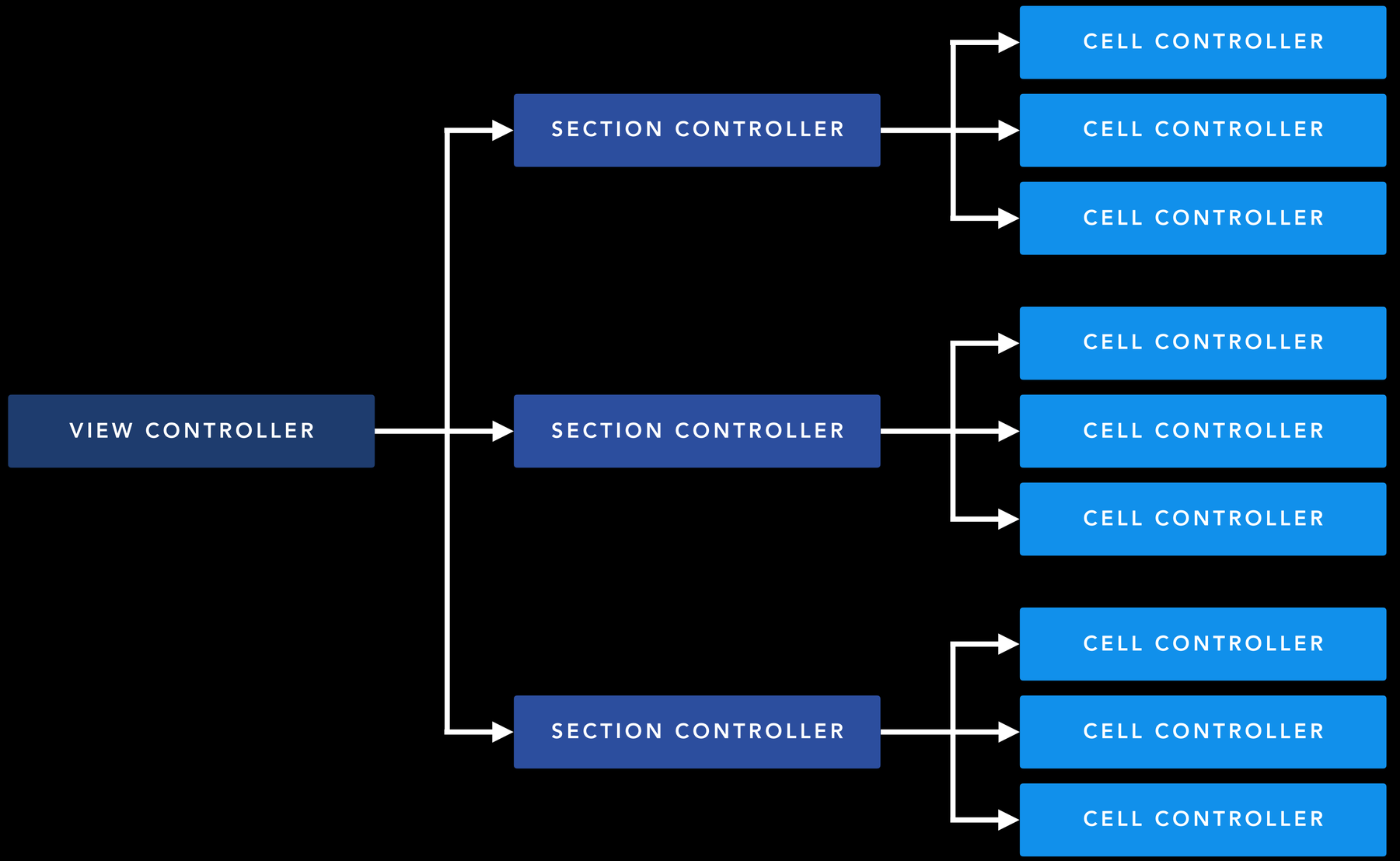
LET’S SEE IT IN ACTION

DEMO NOTES

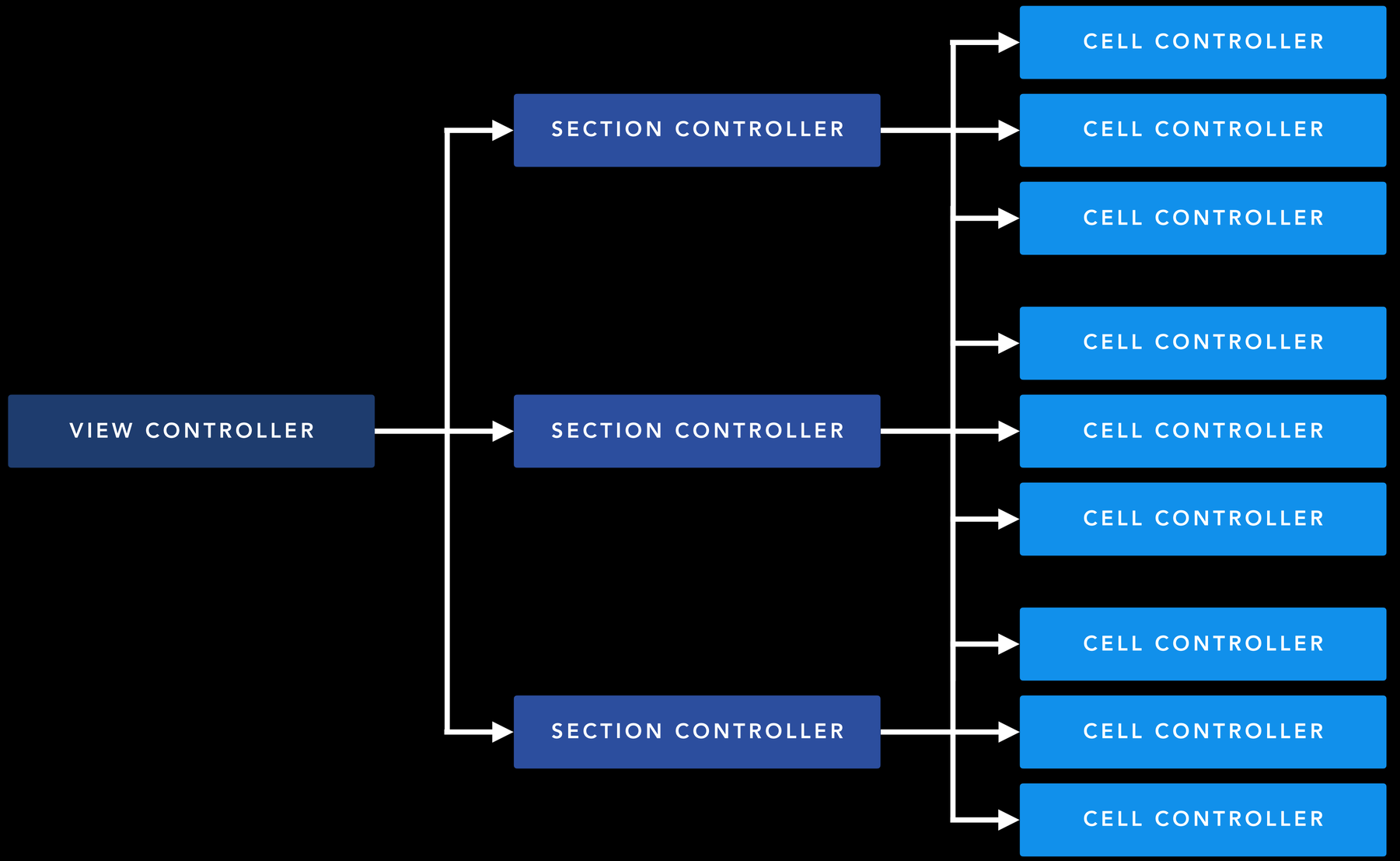
NEMO

- Delegates were used, but Reactive programming would make it much easier to code. ReactiveSwift, RxSwift, etc…
- Typecasing enums blog post: https://medium.com/swift-programming/swift-typecasing-3cd156c323e
- Only one view controller was used as an example of functionality and modularity
- Source: https://github.com/andyyhope/nemo

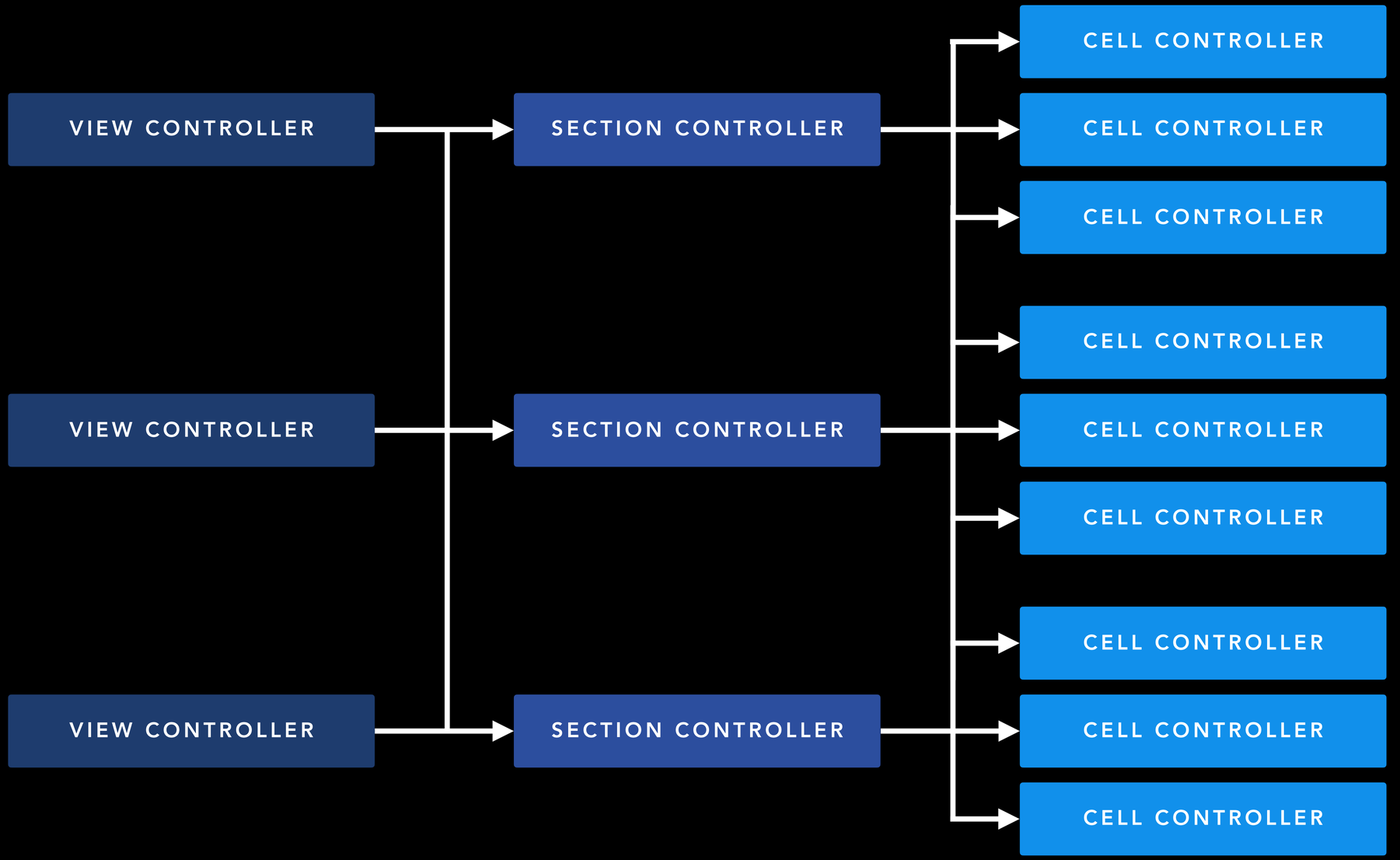
CONTROLLER HIERARCHY



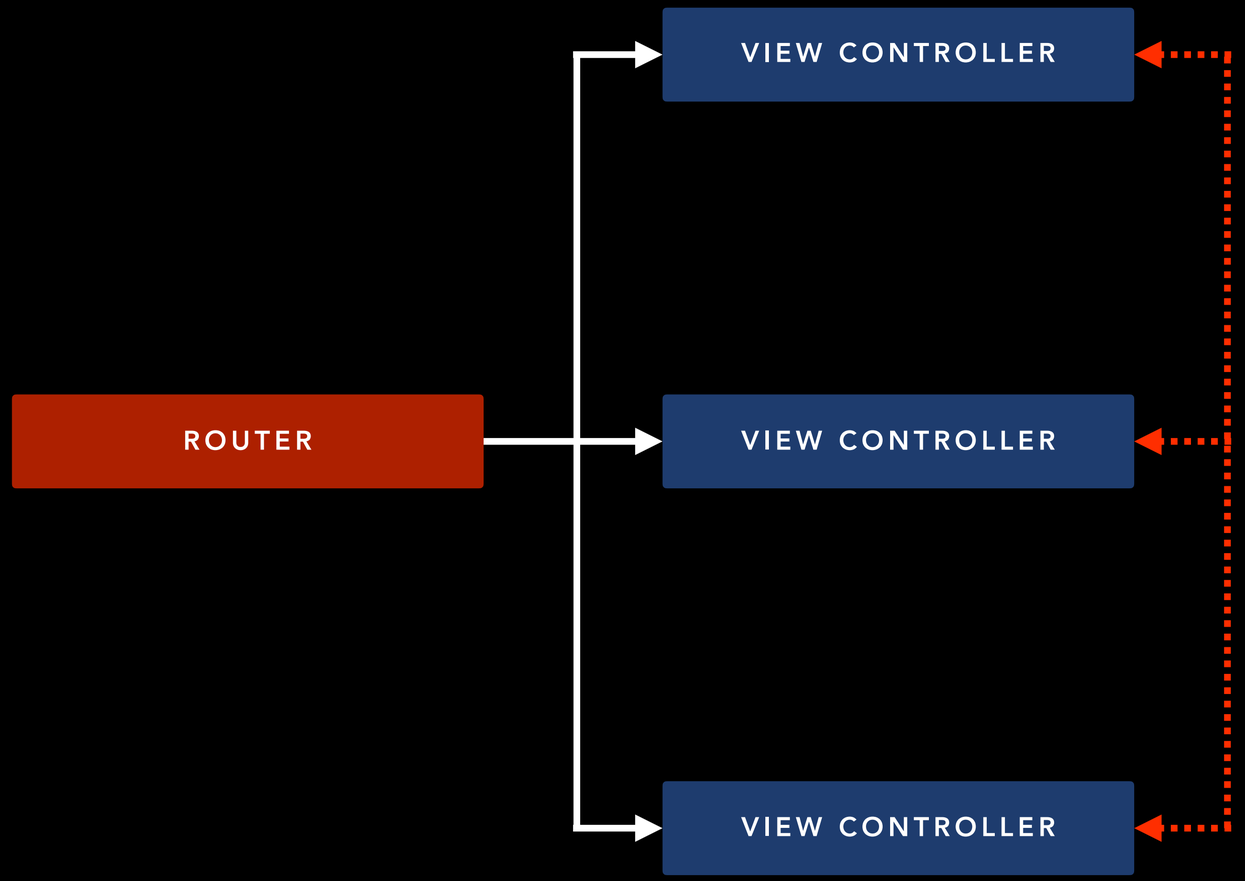
CELL CONTROLLER PORTABILITY



CONTROLLER PORTABILITY



ROUTER COMPATIBILITY

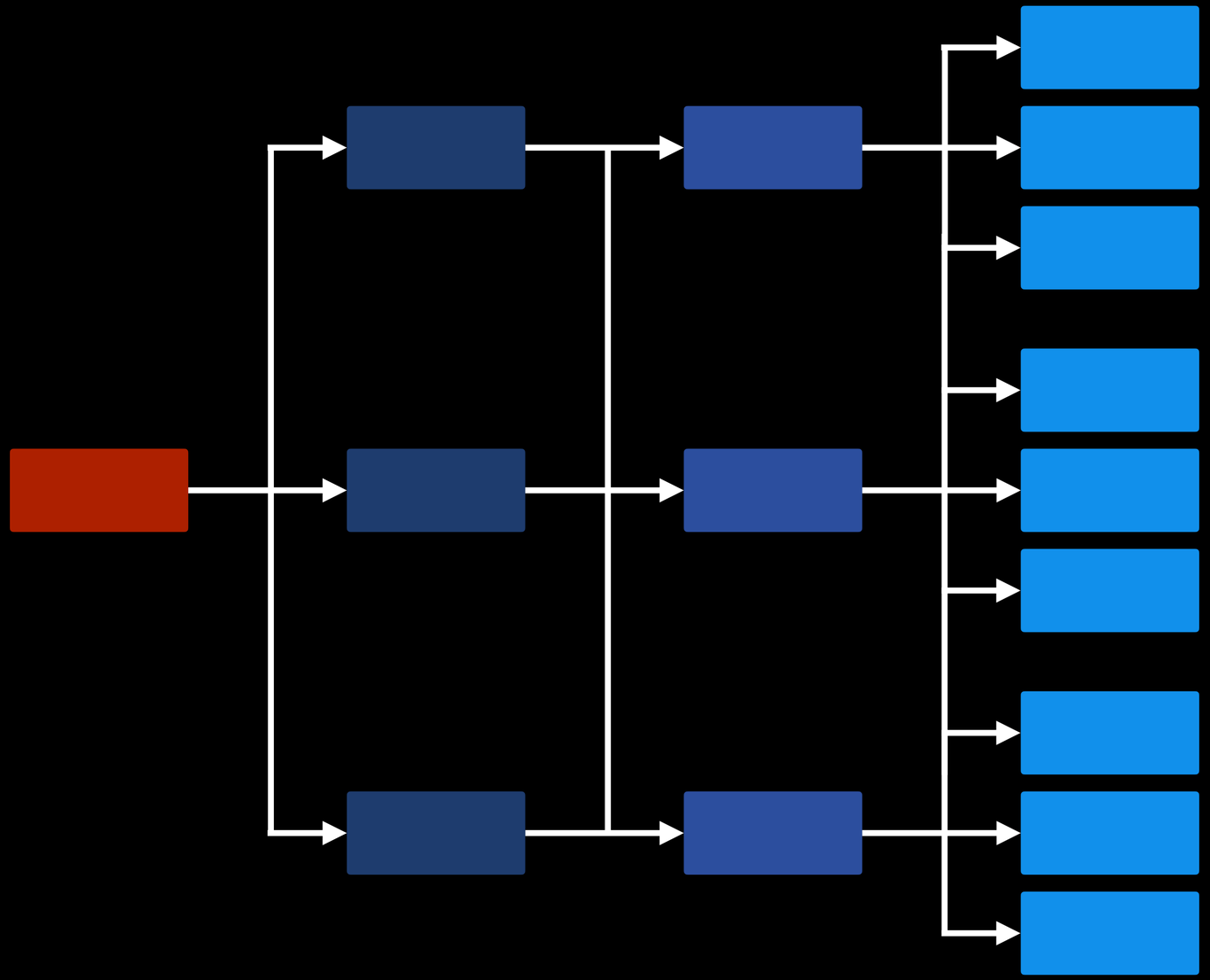


DID WE ACHIEVE OUR GOAL?

REFLECTING ON OUR TARGET

REQUIREMENTS

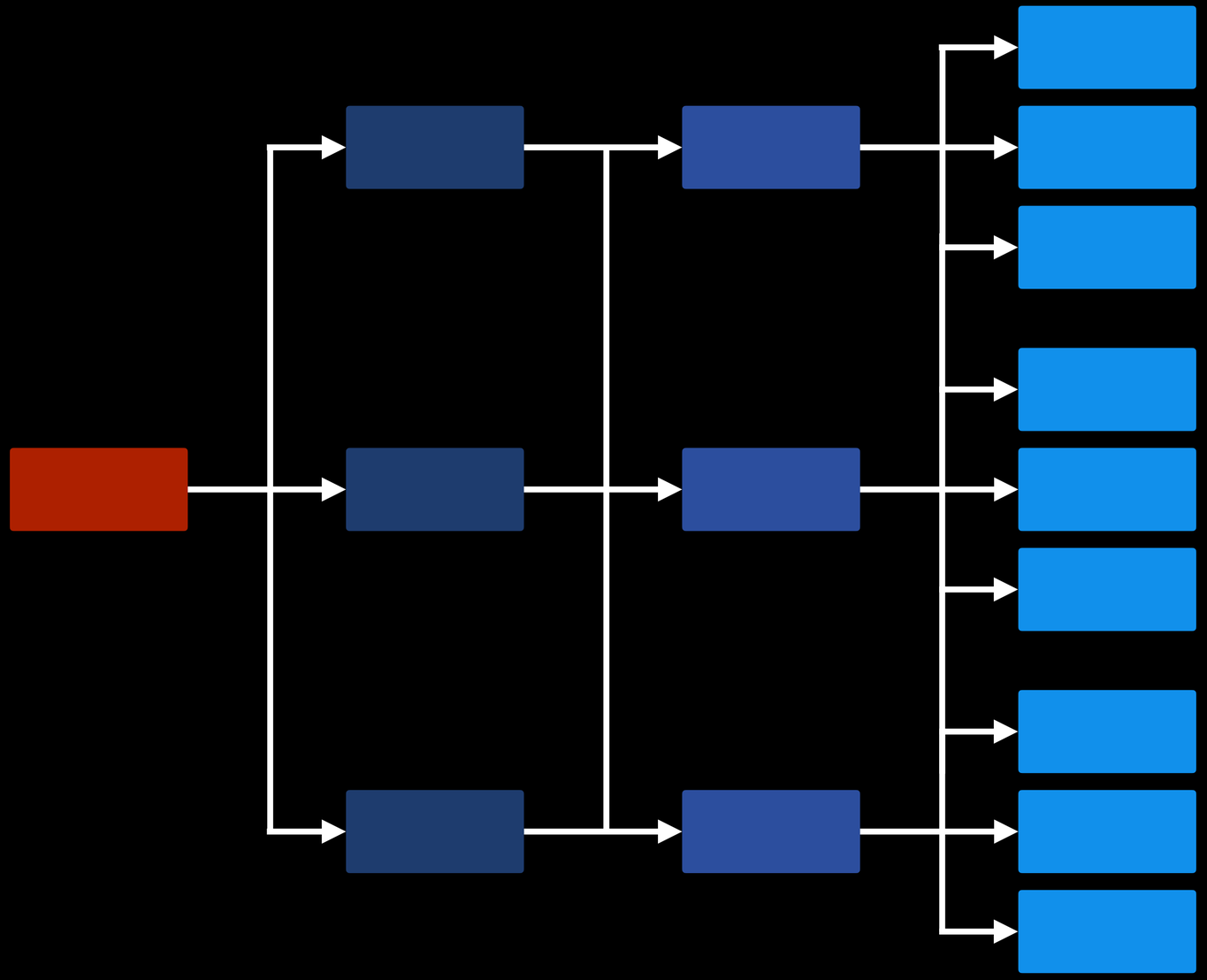
- Testability
- Code reusability
- Easy to onboard
- Modular
- Expandable
- Compatible



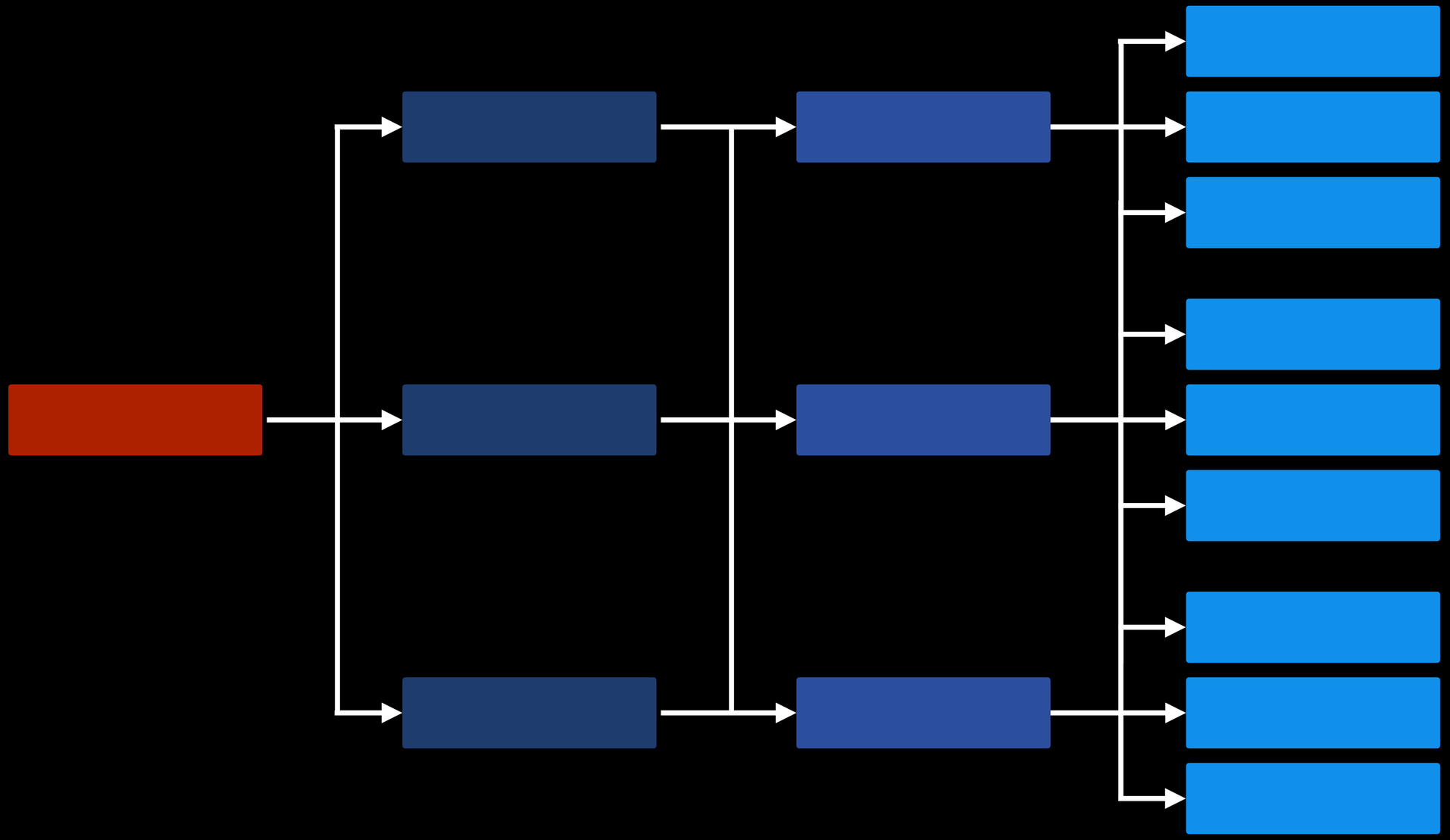
REFLECTING ON OUR TARGET

PRINCIPLES

- iOS first and foremost
- Enforce unidirectional flow
- Utilise immutability
- Include reactive elements
- Embrace Swift's functionality
- Strong guidelines



NEAT AND MODULAR



NEAT & MODULAR

NEAT & MODULAR

NEMO

FINDING NEMO

@ANDYYHOPE

谢 谢