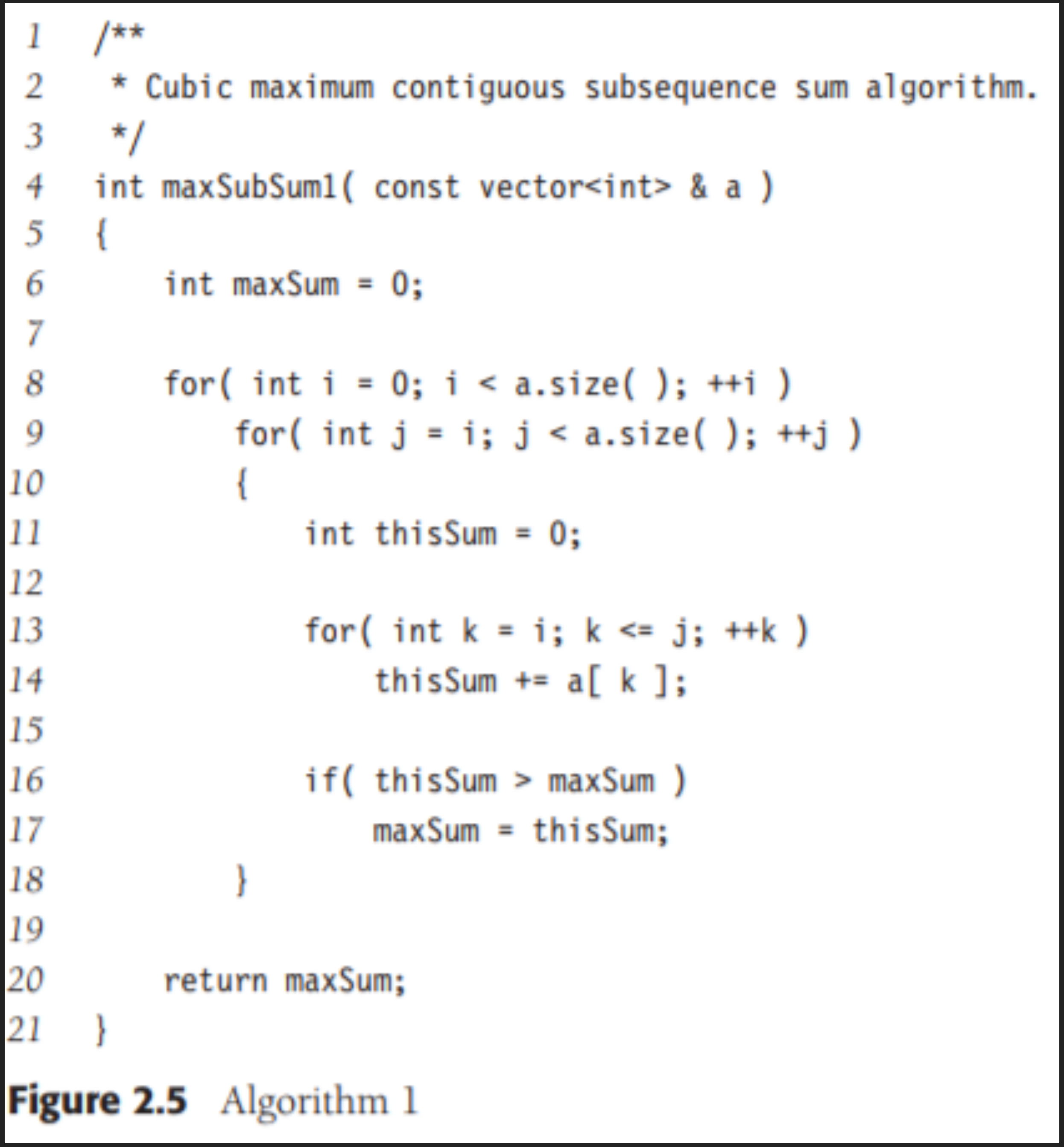
A.STEFANCHUK IT STEP

MVC

MAIN ARCHITECTURE PATTERN

WHAT IS ARCHITECTURE?

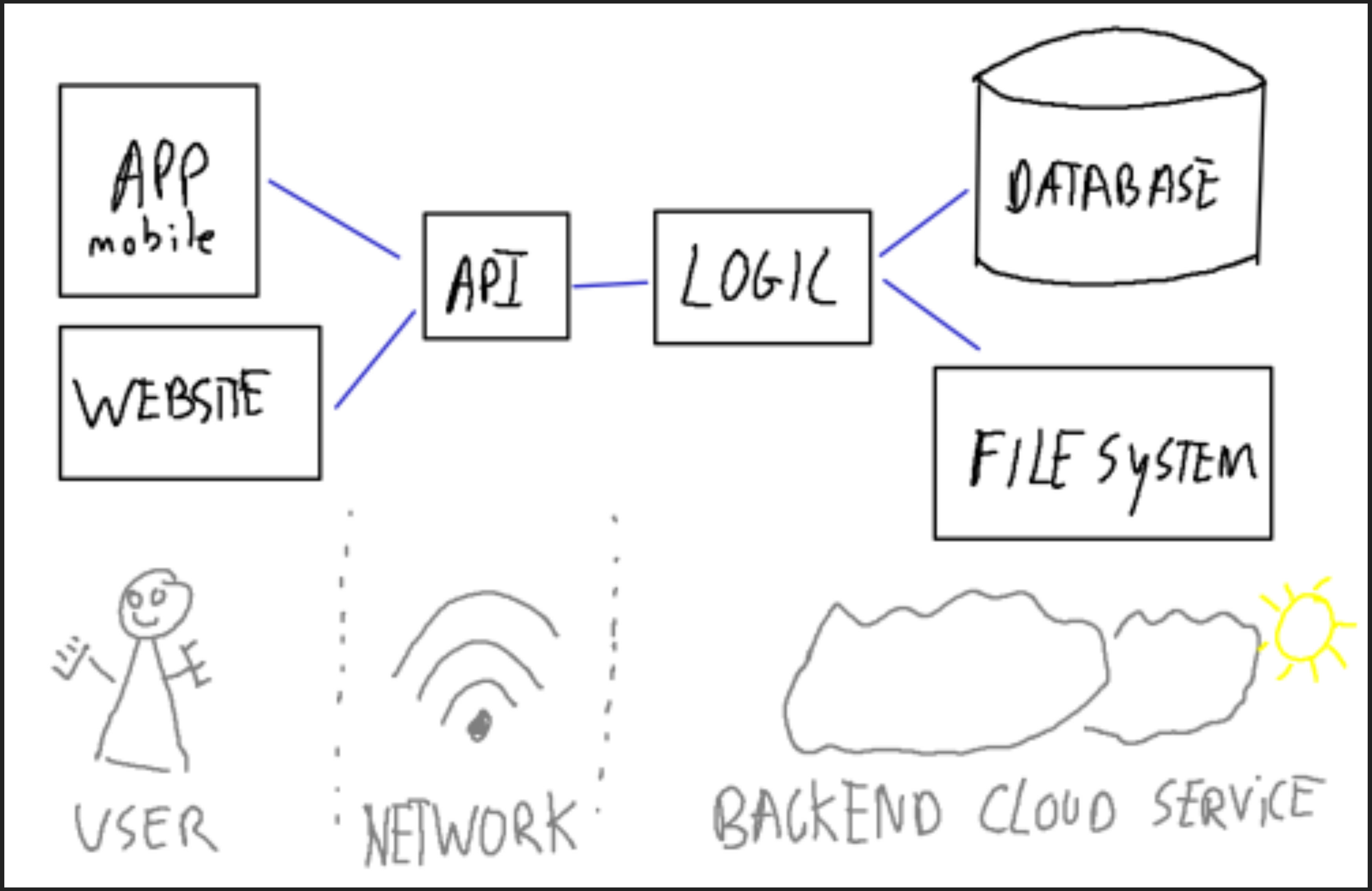
WE CAN'T SEE IT HERE



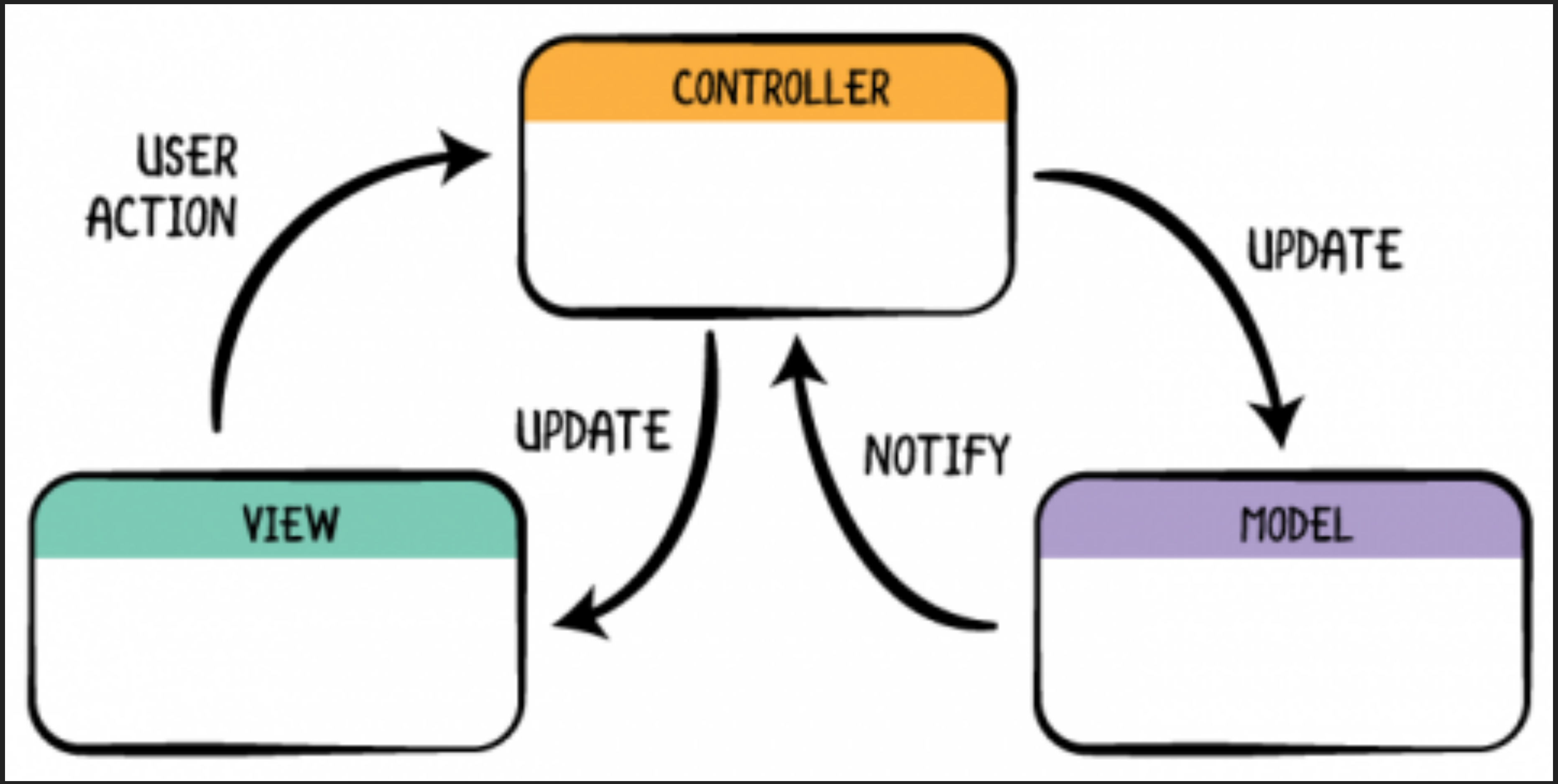
```
/**
 * Cubic maximum contiguous subsequence sum algorithm.
 */
int maxSubSum1( const vector<int> & a )
{
    int maxSum = 0;

    for( int i = 0; i < a.size( ); ++i )
        for( int j = i; j < a.size( ); ++j )
        {
            int thisSum = 0;

            for( int k = i; k <= j; ++k )
                thisSum += a[ k ];

            if( thisSum > maxSum )
                maxSum = thisSum;
        }

    return maxSum;
}
```

Figure 2.5 Algorithm 1

SYSTEM ARCHITECTURE



MOBILE APP ARCHITECTURE



MOBILE APP ARCHITECTURE

- It is how we build our application / system, which components do we use and how they are connected
- It’s application skeleton
- It’s like walls, bricks and foundation for real building

ARCHITECTURE PROPERTIES

- Components distribution (isolation + single responsibility)

ARCHITECTURE PROPERTIES

- Components distribution (isolation + single responsibility)
- Ease of use (other people can understand it work with it)

ARCHITECTURE PROPERTIES

- Components distribution (isolation + single responsibility)
- Ease of use (other people can understand it work with it)
- Consistent (same in different places)

ARCHITECTURE PROPERTIES

- Components distribution (isolation + single responsibility)
- Ease of use (other people can understand it work with it)
- Consistent (same in different places)
- No matter how many screens do we have or how complex those screens are -> it still can handle it (scalability)

ARCHITECTURE PROPERTIES

- Components distribution (isolation + single responsibility)
- Ease of use (other people can understand it work with it)
- Consistent (same in different places)
- No matter how many screens do we have or how complex those screens are -> it still can handle it (scalability)
- Maintainable - it's easy to add new features or edit existing ones

ARCHITECTURE PROPERTIES

- Components distribution (isolation + single responsibility)
- Ease of use (other people can understand it work with it)
- Consistent (same in different places)
- No matter how many screens do we have or how complex those screens are -> it still can handle it (scalability)
- Maintainable - it's easy to add new features or edit existing ones
- Code can be reused

ARCHITECTURE PROPERTIES

- Components distribution (isolation + single responsibility)
- Ease of use (other people can understand it work with it)
- Consistent (same in different places)
- No matter how many screens do we have or how complex those screens are -> it still can handle it (scalability)
- Maintainable - it’s easy to add new features or edit existing ones
- Code can be reused
- Code can be tested

MVC – THAT'S HOW YOU RULE THE PROJECT

MODEL

What your app is
(Business logic)

MVC – THAT'S HOW YOU RULE THE PROJECT

MODEL

What your app is
(Business logic)

- Model classes
- Networking (requests to server)
- Persistence (local saving of data)
- Parsing code (from JSON)
- Managers (for frameworks [Audio])
- Constants
- Helpers

MVC – THAT'S HOW YOU RULE THE PROJECT

MODEL

What your app is
(Business logic)

VIEW

How data is
displayed in app

(dumb objects)

MVC – THAT'S HOW YOU RULE THE PROJECT

- Layout
- Styling
- Animations
- Transitions
- Displaying of data!

VIEW

How data is displayed in app

(dumb objects)

MVC – THAT'S HOW YOU RULE THE PROJECT

CONTROLLER

Coordinator of those 2

MODEL

What your app is
(Business logic)

VIEW

How data is
displayed in app

(dumb objects)

MVC – THAT'S HOW YOU RULE THE PROJECT

CONTROLLER

Coordinator of those 2

- How to refresh data
- What screen to show next
- Transform data from model to view
- Handle user interactions

MVC – THAT'S HOW YOU RULE THE PROJECT

CONTROLLER

Coordinator of those 2

MODEL

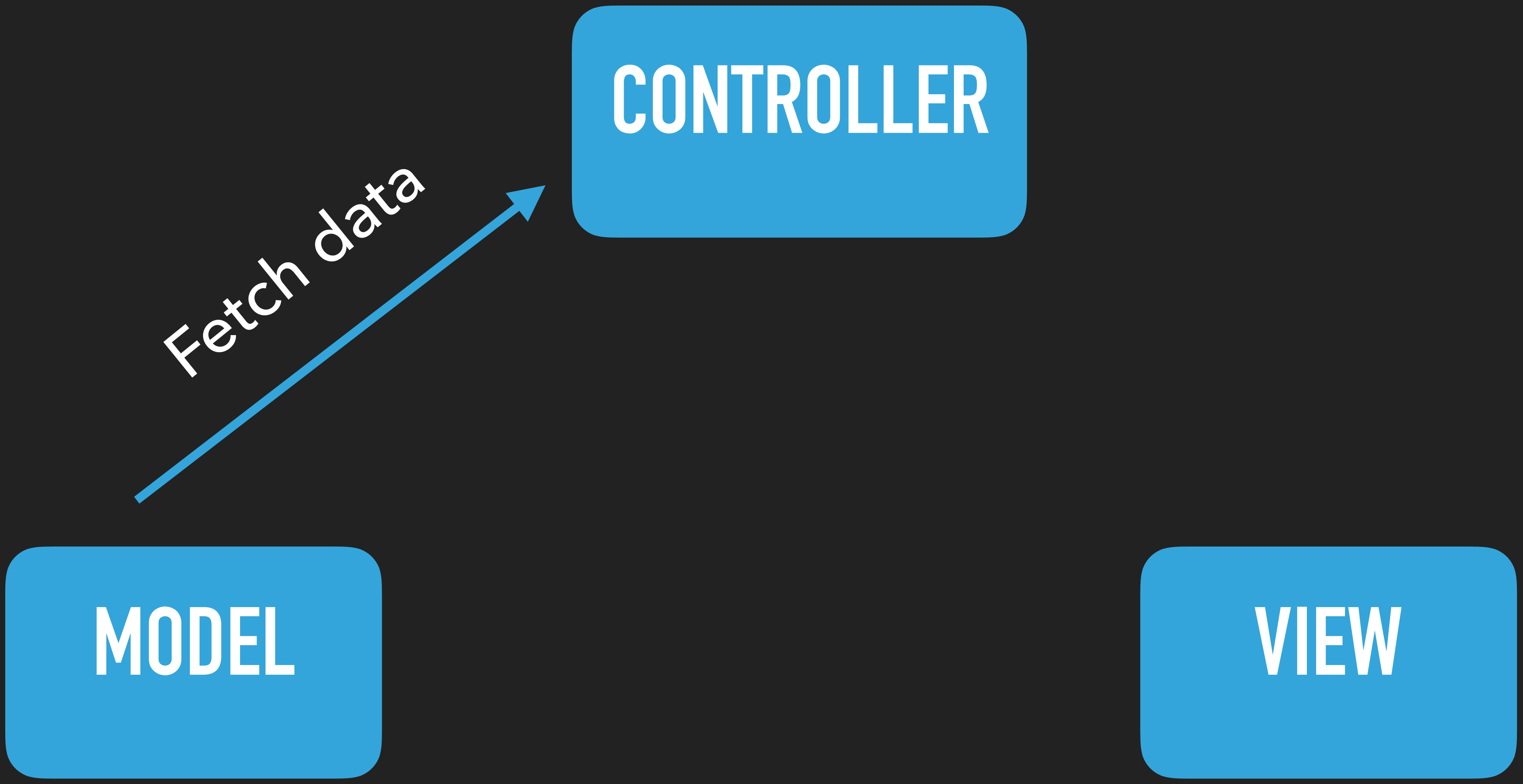
What your app is
(Business logic)

VIEW

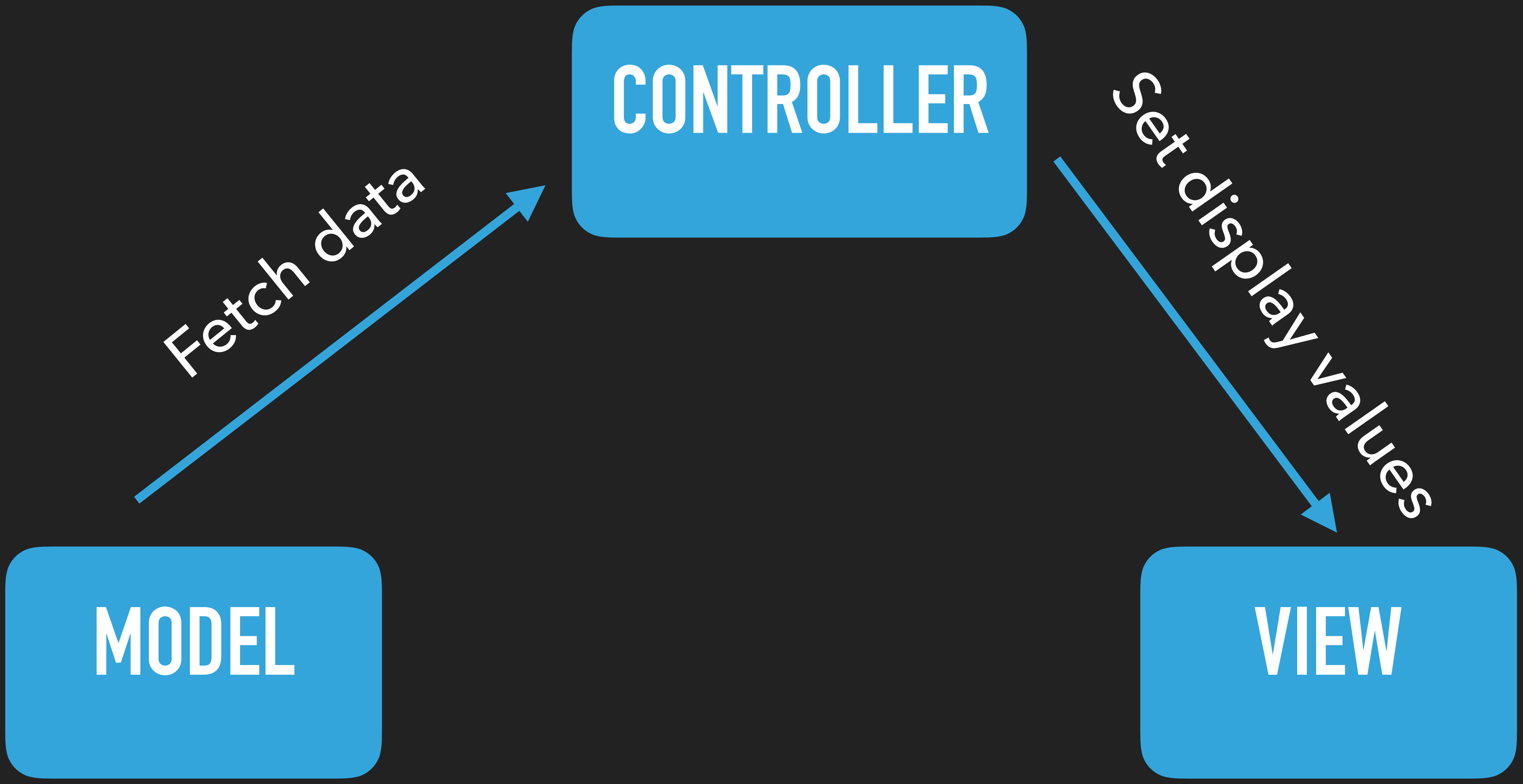
How data is
displayed in app

(dumb objects)

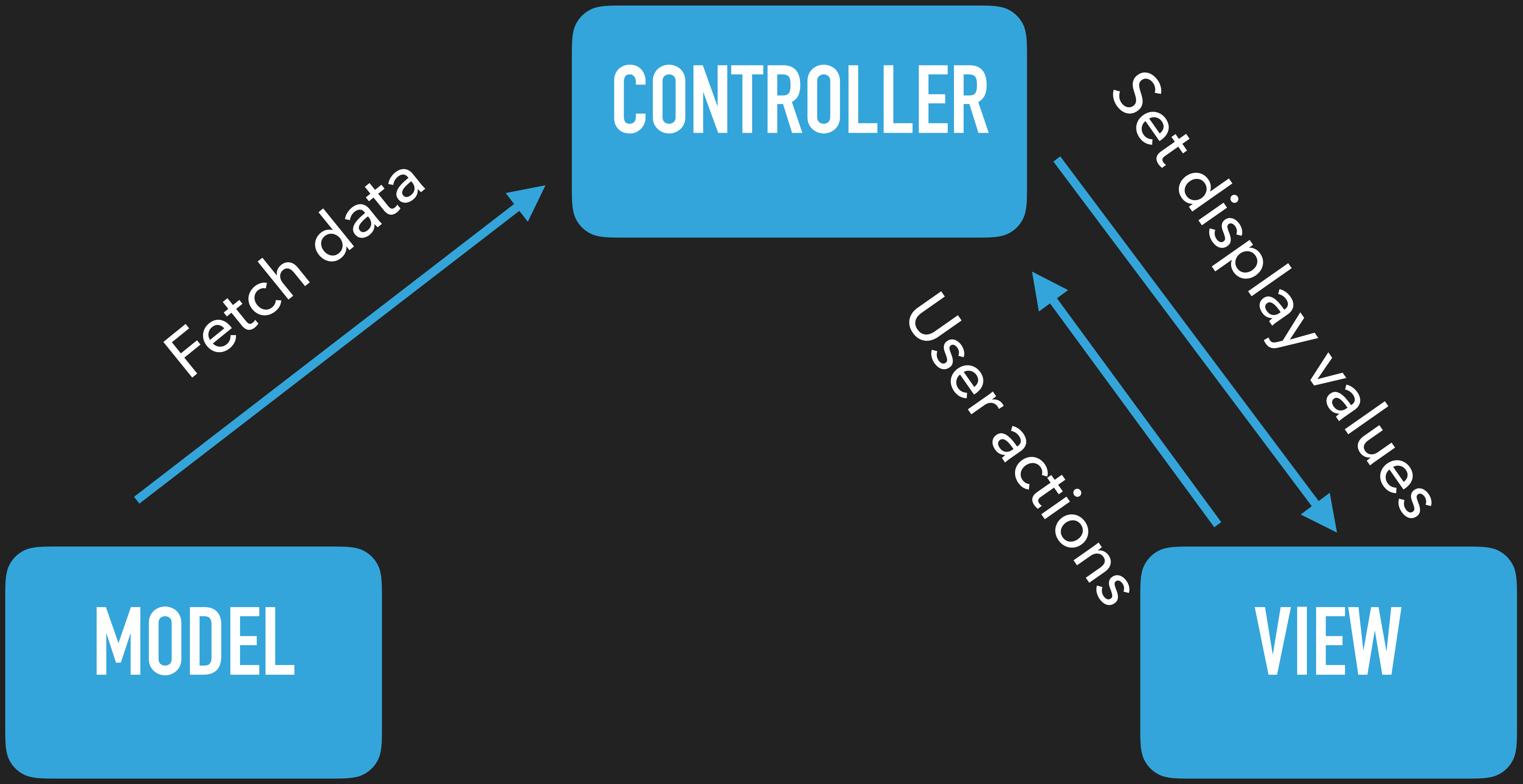
MVC – THAT'S HOW YOU RULE THE PROJECT



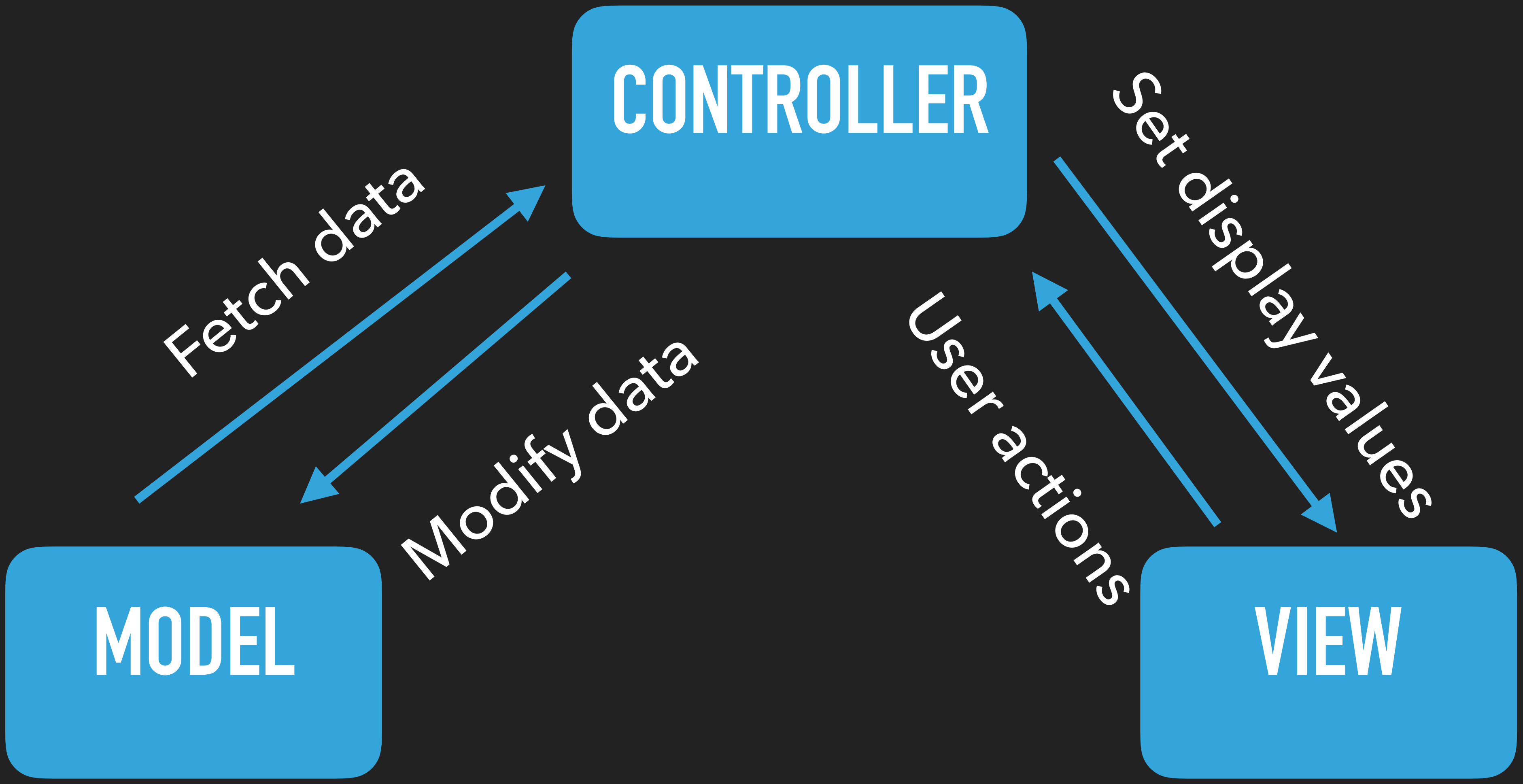
MVC – THAT'S HOW YOU RULE THE PROJECT



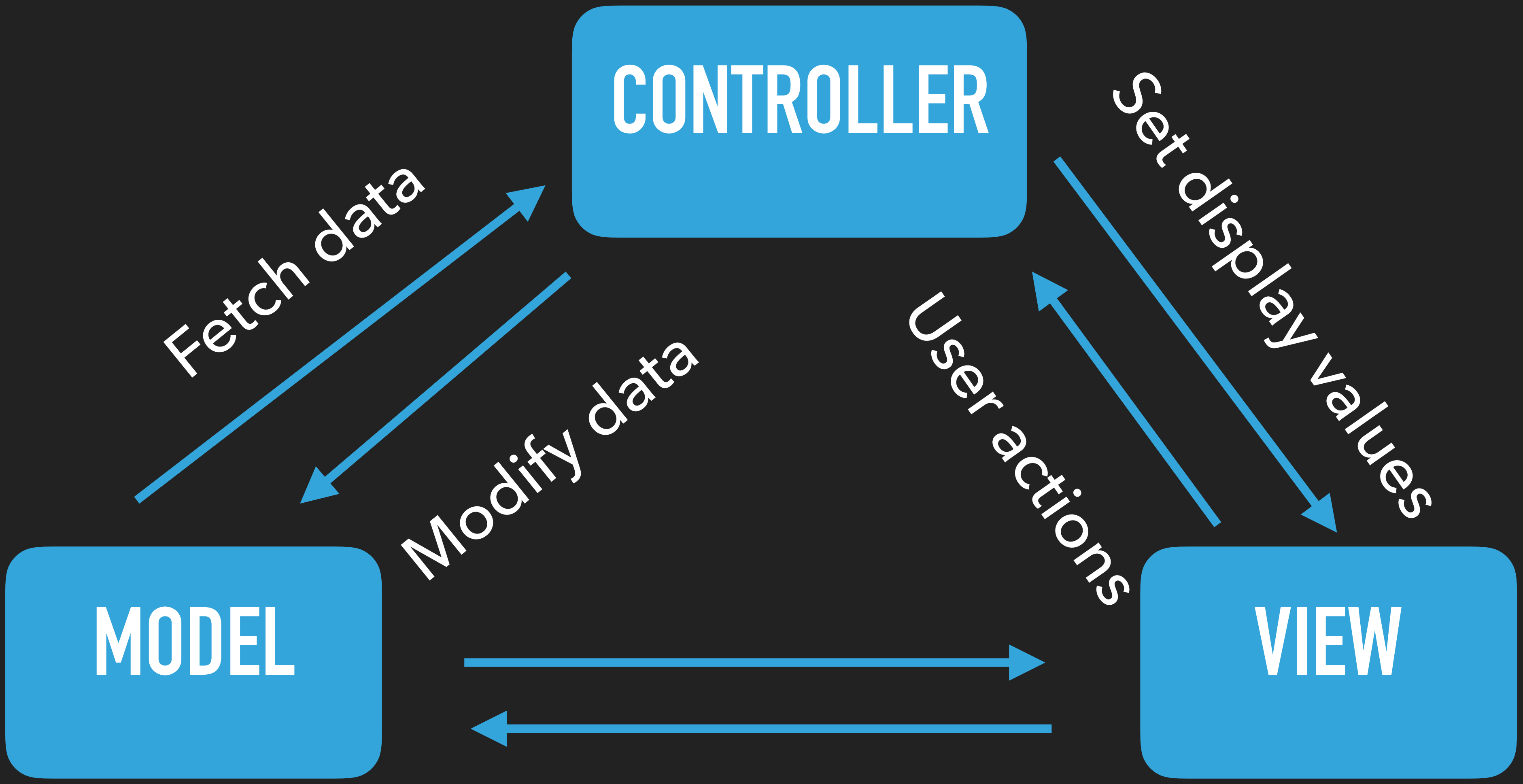
MVC – THAT'S HOW YOU RULE THE PROJECT



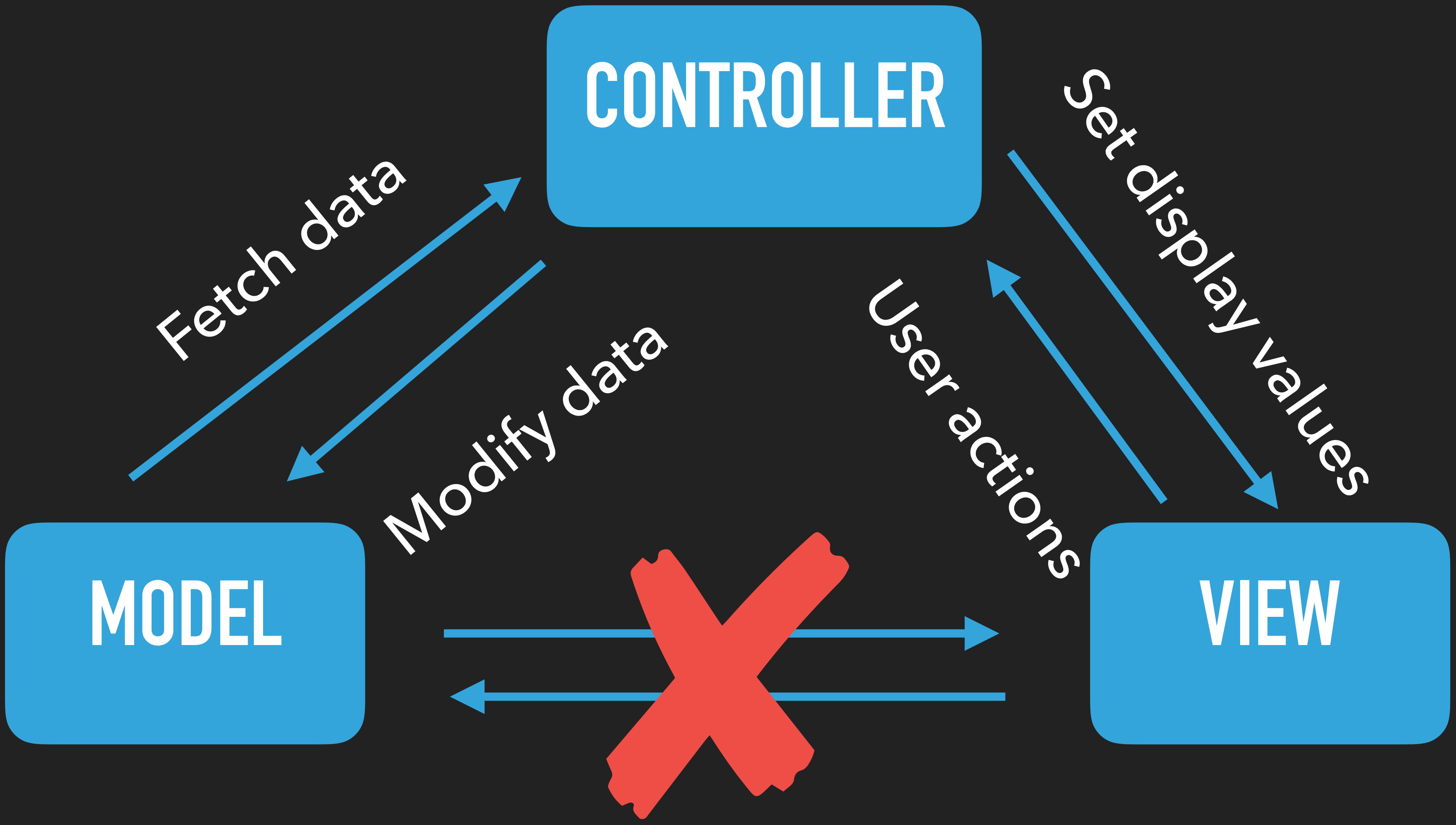
MVC – THAT'S HOW YOU RULE THE PROJECT



MVC – THAT'S HOW YOU RULE THE PROJECT

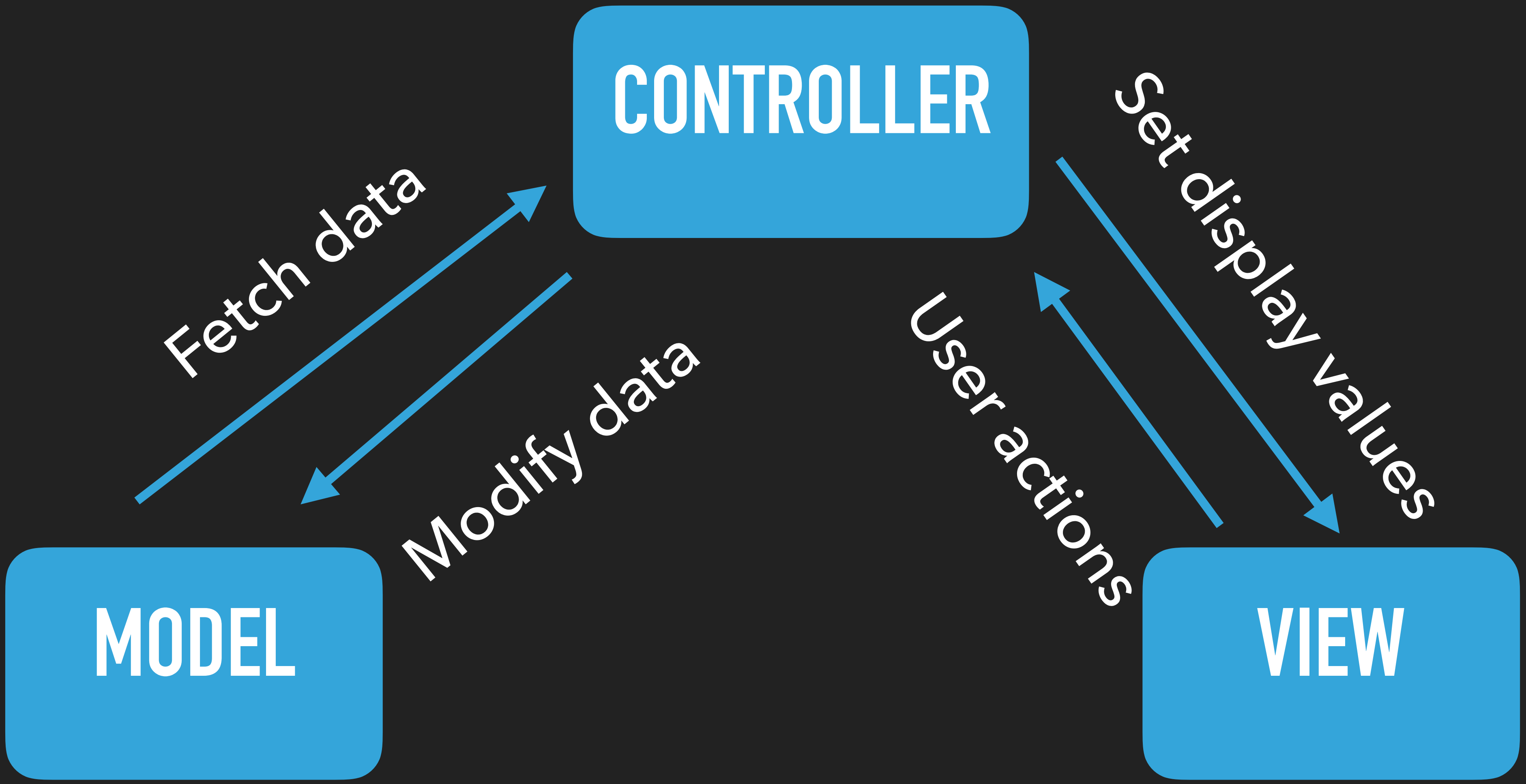


MVC – THAT'S HOW YOU RULE THE PROJECT



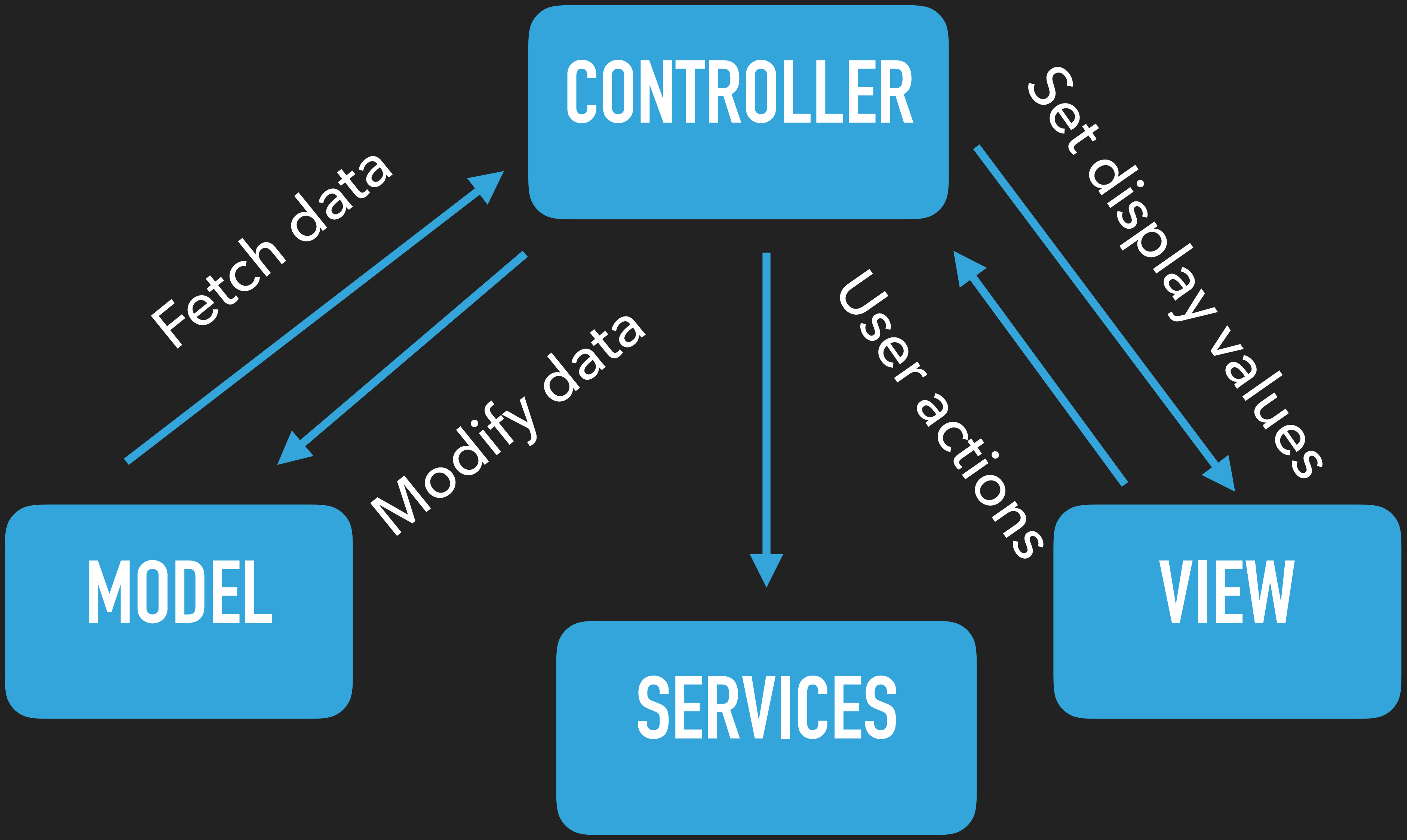
Such connections are not allowed!

MVC – THAT'S HOW YOU RULE THE PROJECT

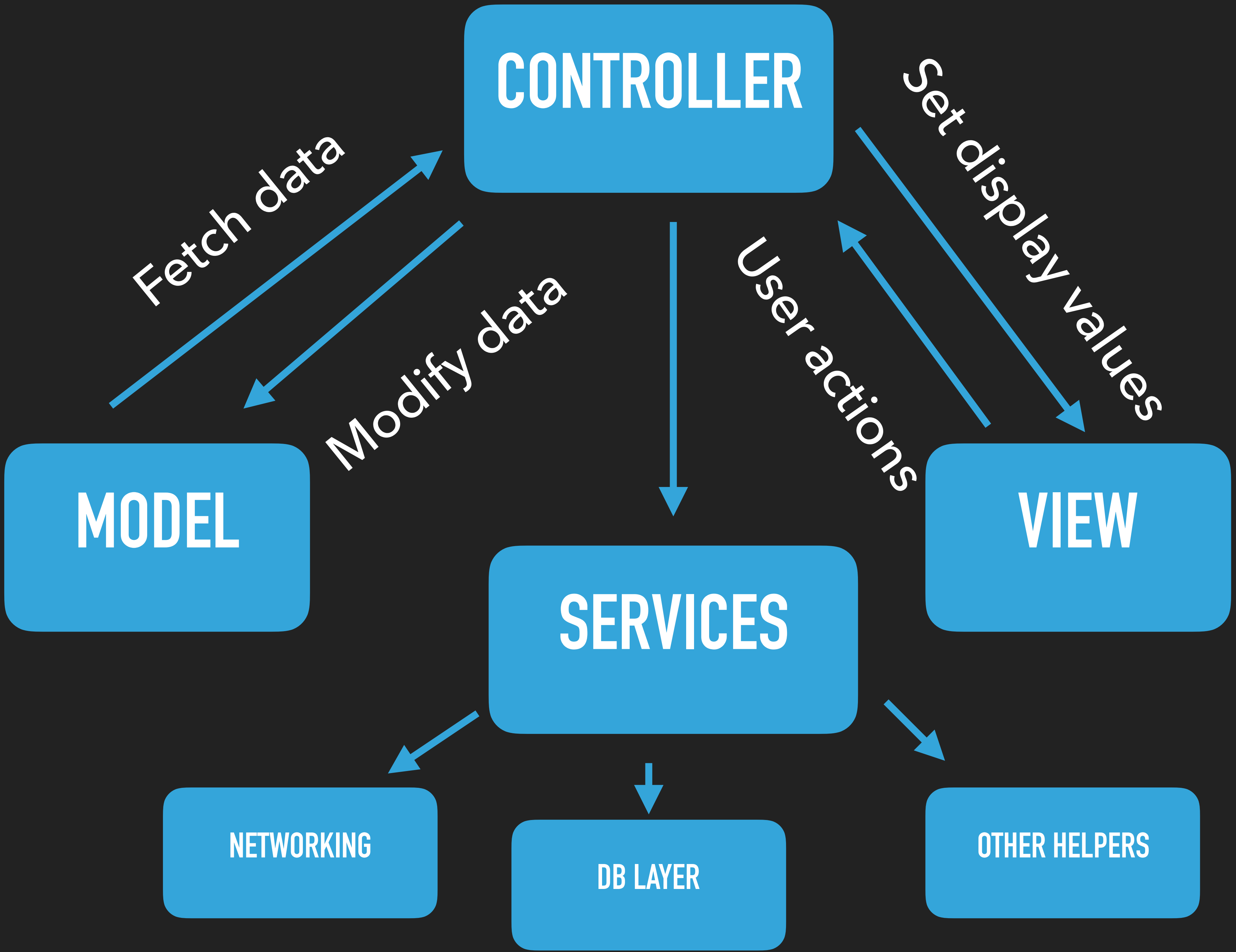


With time controller can become very big

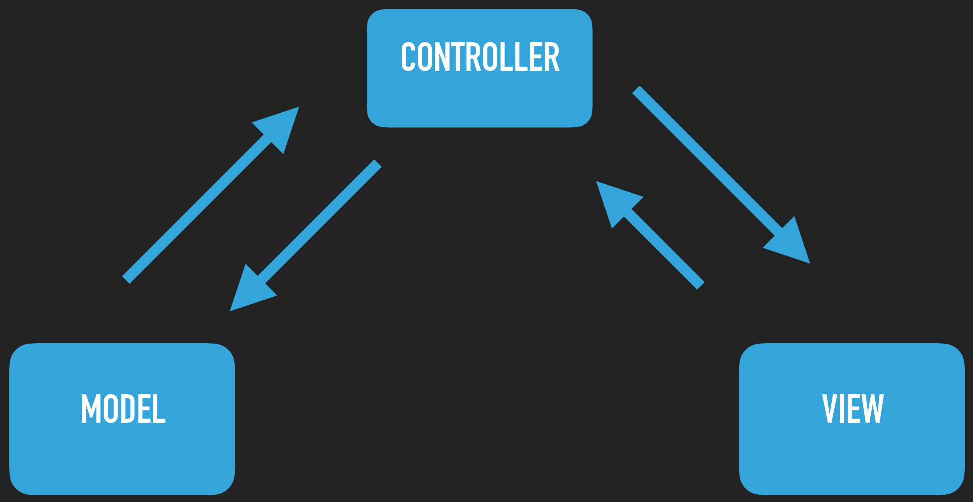
MVC – THAT'S HOW YOU RULE THE PROJECT

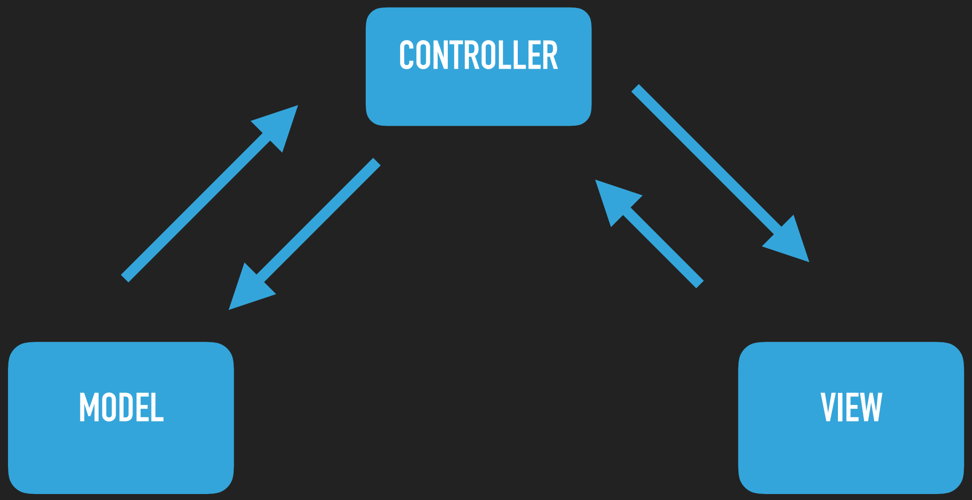


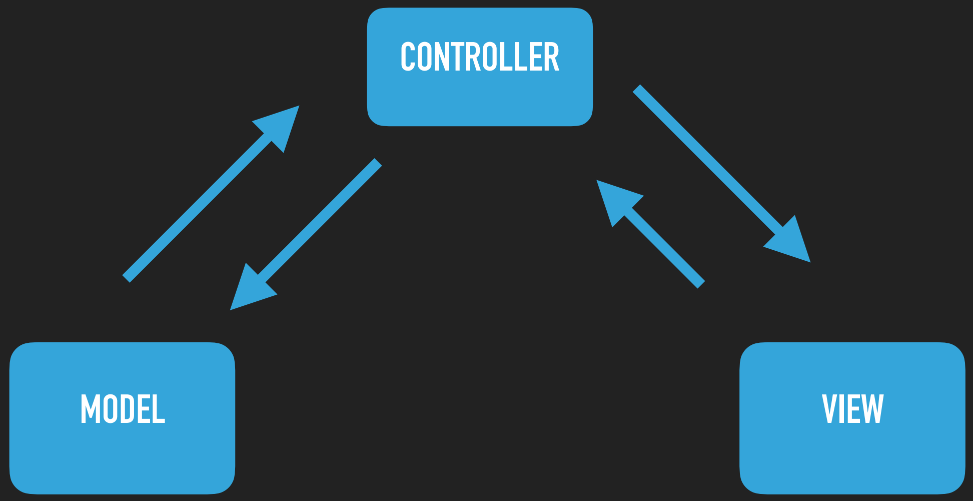
MVC – THAT'S HOW YOU RULE THE PROJECT



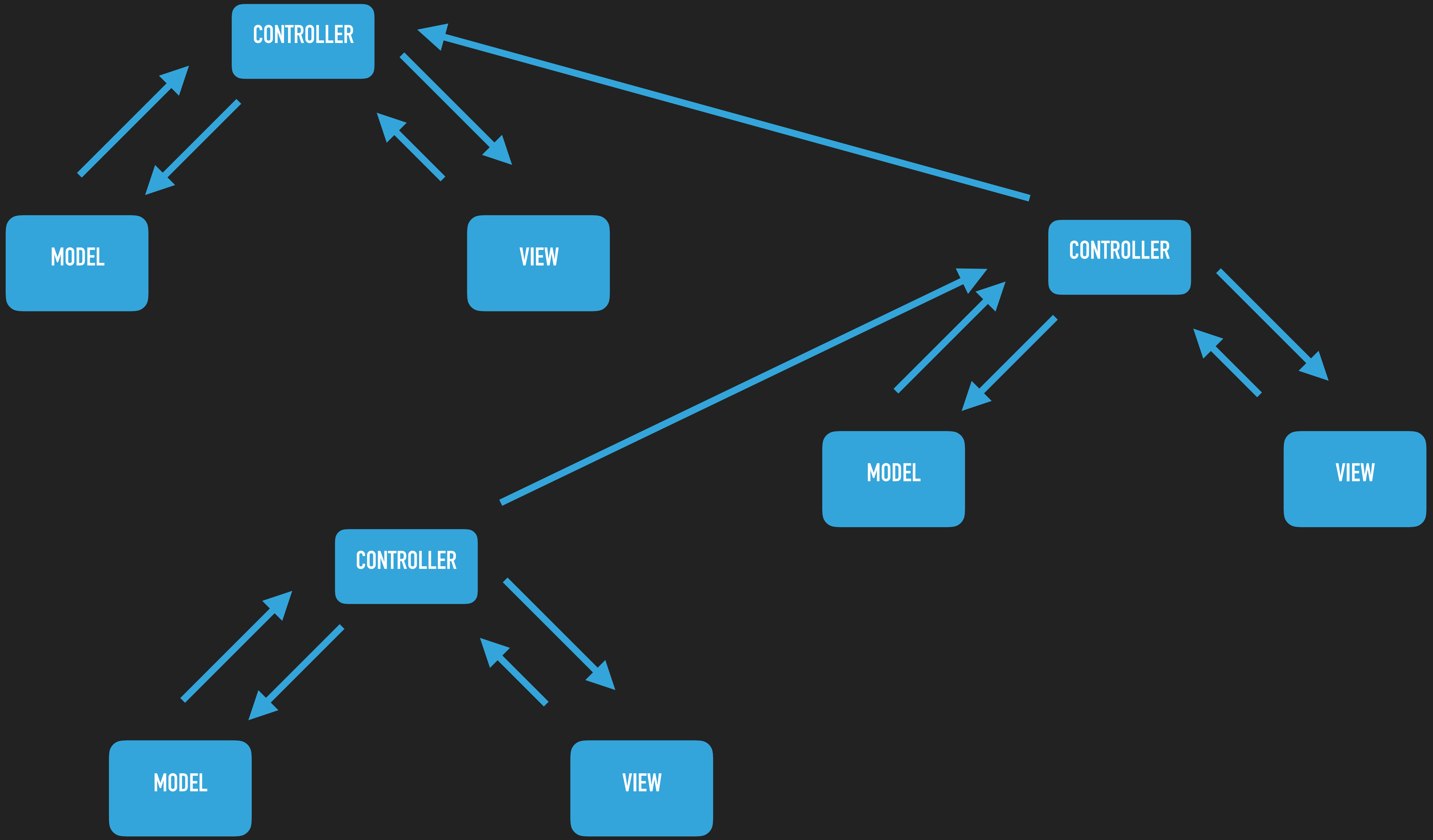
HOW IT WORKS FOR MANY SCREENS



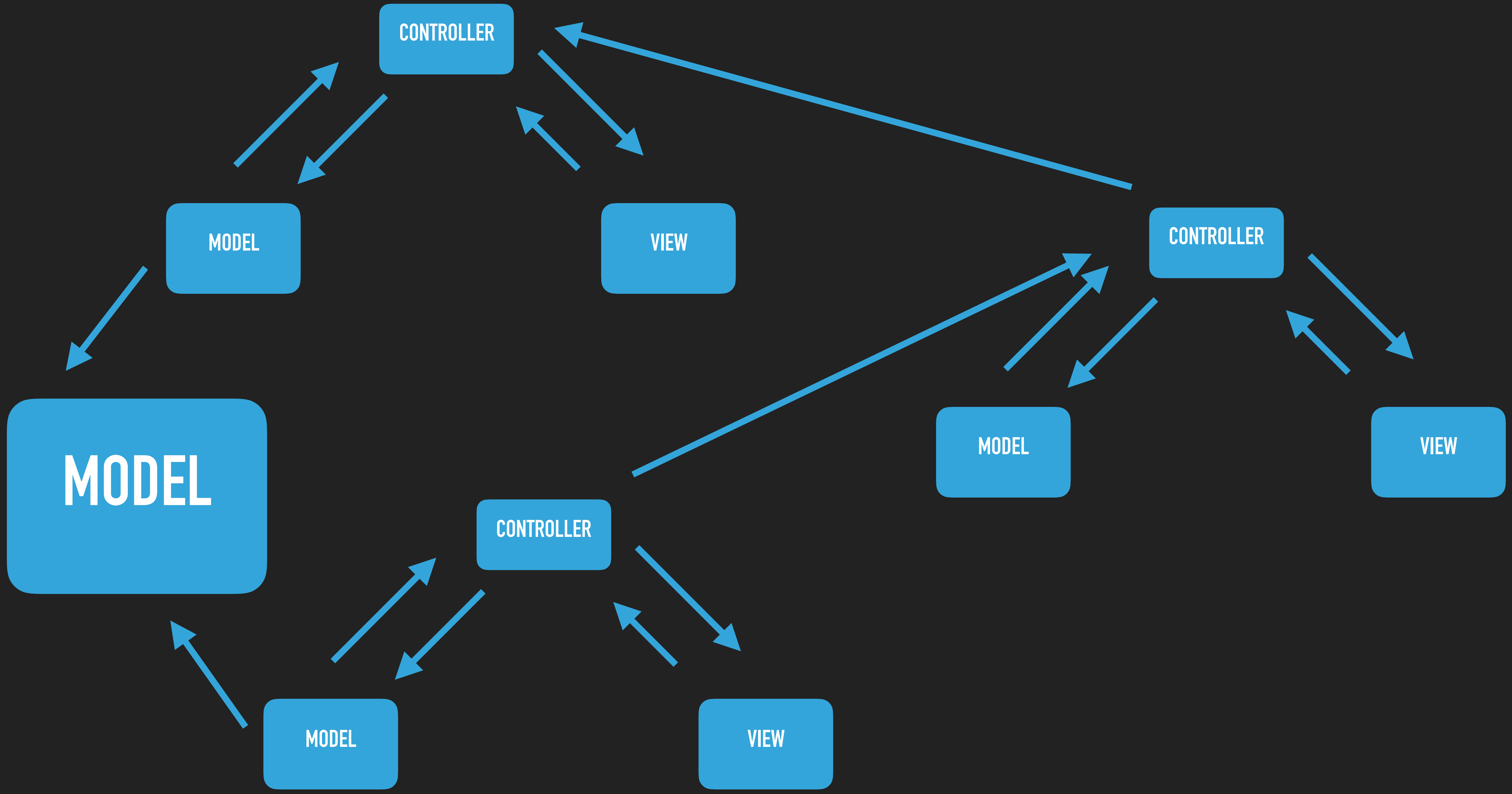




HOW IT WORKS FOR MANY SCREENS



HOW IT WORKS FOR MANY SCREENS



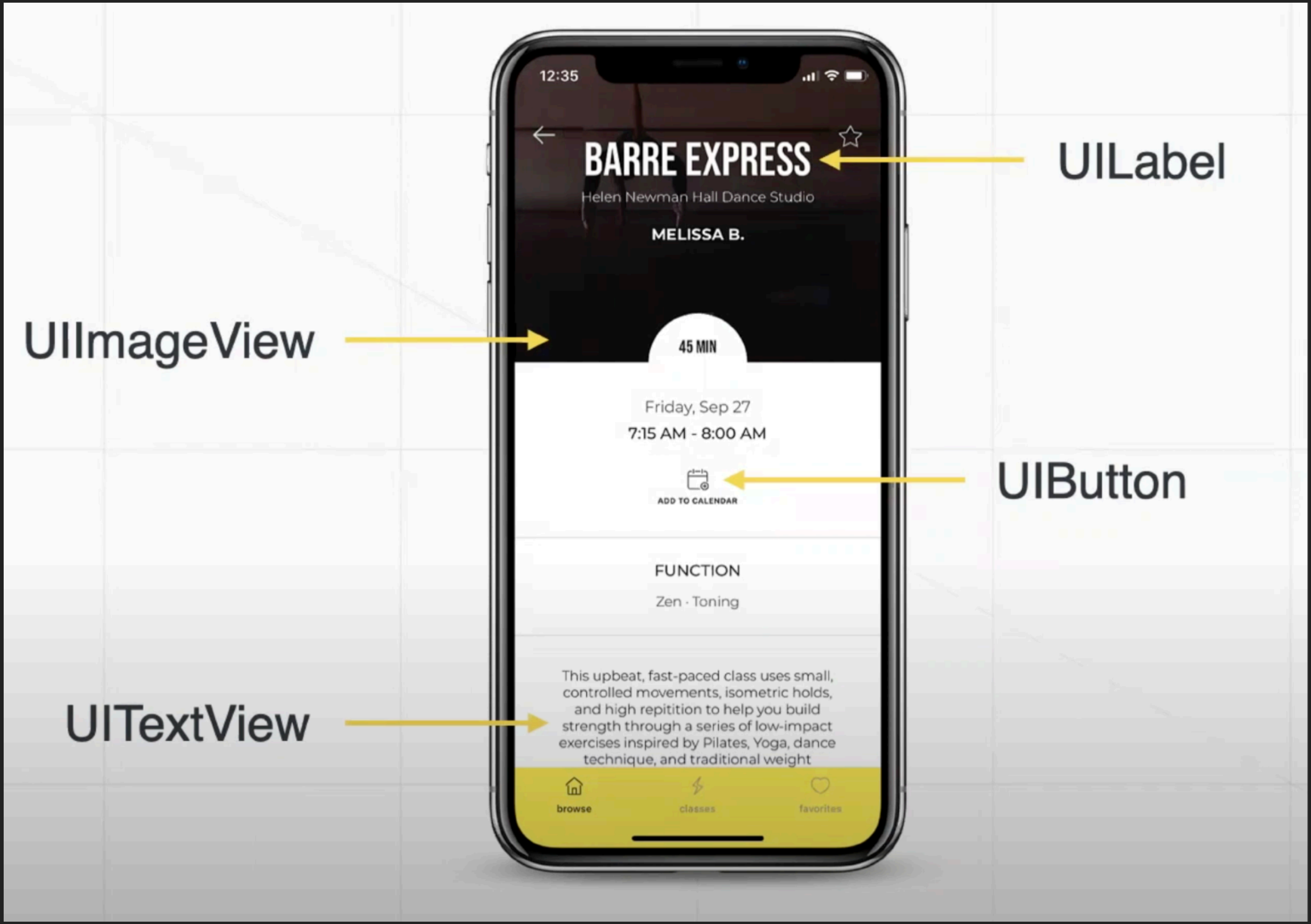
CORE OBJECTS - VIEW CONTROLLER

- UIViewController class
- Single screen in application
- Maintain and holds hierarchy of views and their subviews
- Every app has at least one view controller - it lives as root view controller in App Window property
- There can be child view controllers - they don’t take whole screen
- View controller has lifecycle - viewDidLoad(), viewWillAppear(), viewWillDisappear() and others

CORE OBJECTS - VIEW

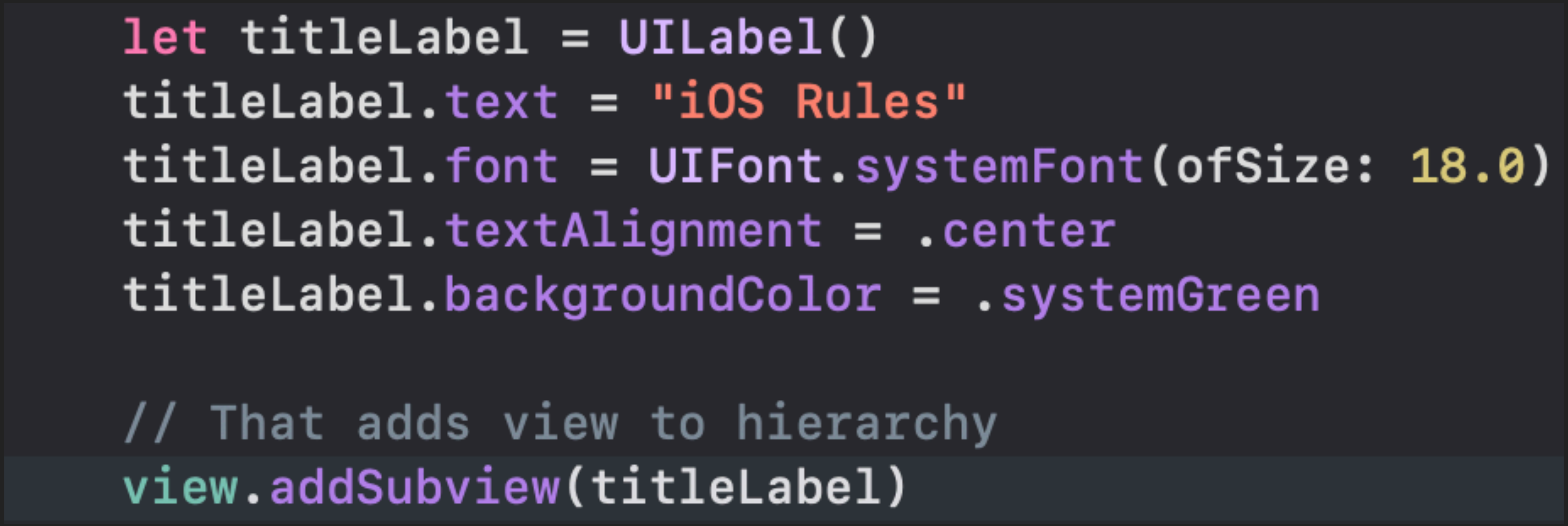
- UIView class
- Object drawn into the screen
- Can contain other views - subviews
- Manage nothing - just displays data and sends signals when user interacts with them
- Can have custom drawing inside thanks to Layer
- Subclasses - UILabel, UIButton, UIImageView, UITextField

CORE OBJECTS – VIEW



This upbeat, fast-paced class uses small, controlled movements, isometric holds, and high repitition to help you build strength through a series of low-impact exercises inspired by Pilates, Yoga, dance technique, and traditional weight

CORE OBJECTS – VIEW



```swift
let titleLabel = UILabel()
titleLabel.text = "iOS Rules"
titleLabel.font = UIFont.systemFont(ofSize: 18.0)
titleLabel.textAlignment = .center
titleLabel.backgroundColor = .systemGreen

// That adds view to hierarchy
view.addSubview(titleLabel)
```

That's how do you work with views in code - just setting different values to properties

COMMON FOLDERS IN YOUR PROJECT

- Model - all models structs (like user, delivery, post)
- View - storyboard files, separate views
- Controller - each screen can have it's own folder with controller + helper classes
- DB layer - classes for saving data in app
- Networking layer - classes for making HTTP requests
- Utils - helper classes that can be reusable
- Constants - all the "magic" numbers in project
- Extensions - in Swift we have them a lot (Int+Extension)
- Recourses - all images, music, etc.
- Supporting files - settings files and others