MV* Design Patterns

Alexander Nelson

August 25, 2017

University of Arkansas - Department of Computer Science and Computer Engineering

Reminders

Course Mechanics

Course Webpage:

`you.uark.edu/ahnelson/cmpe-4623-mobile-programming/`

Syllabus is on the website.

Course Communication:

`https://csce4623-uark.slack.com/`

This slack channel is to be the primary mode of communication

Projects

Project 1 will go out Wednesday

Choose a project idea and team for the final project ASAP

First project report is due September 15th

Multitier Architectures

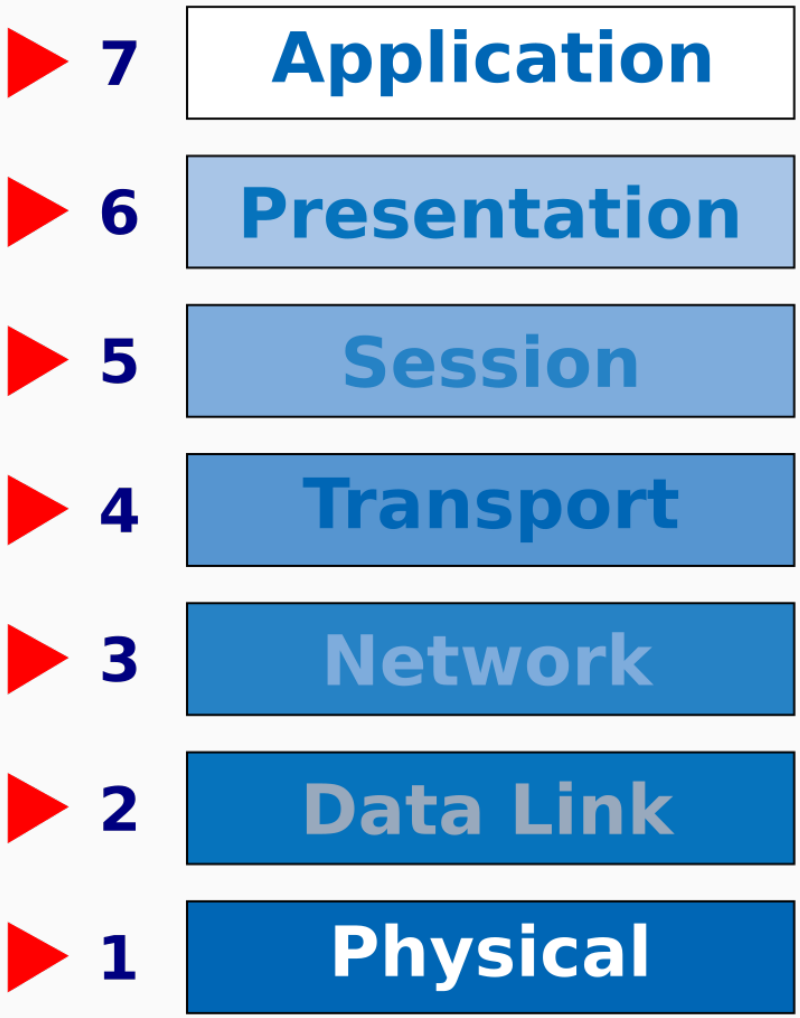
What is a multitier architecture?

Physical separation of data concerns

Examples:

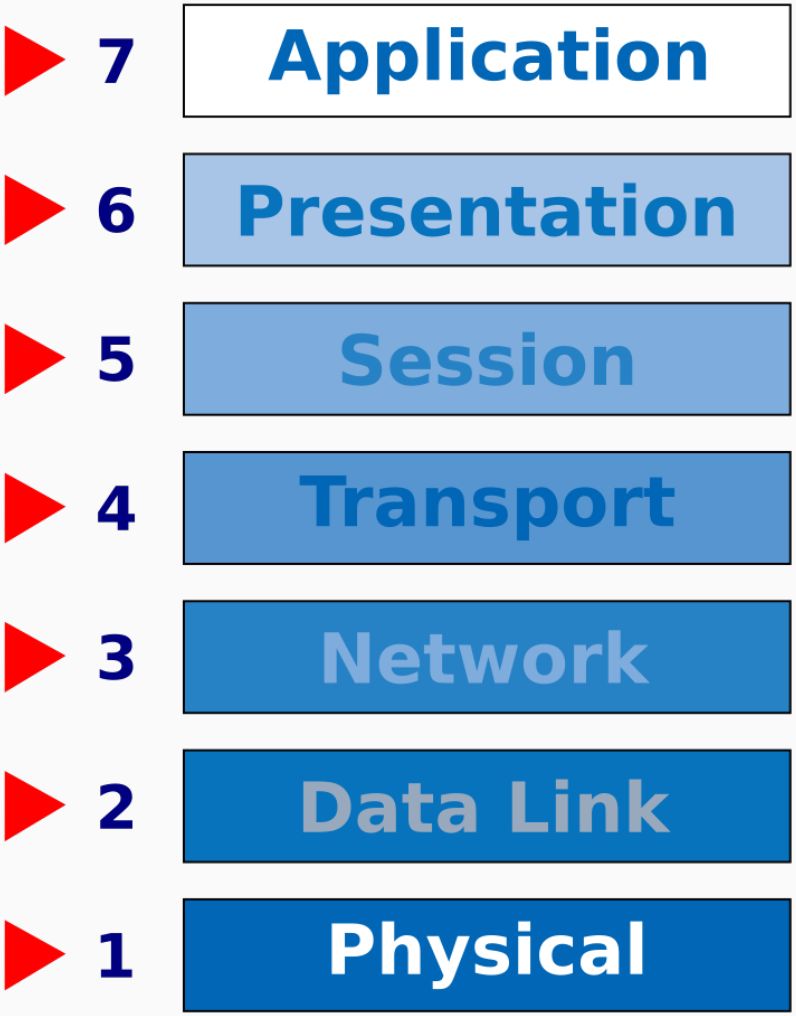
- Presentation (UI)
- Application Processing
- Data Management

Why split into layers?



OSI Model

Why split into layers?



OSI Model

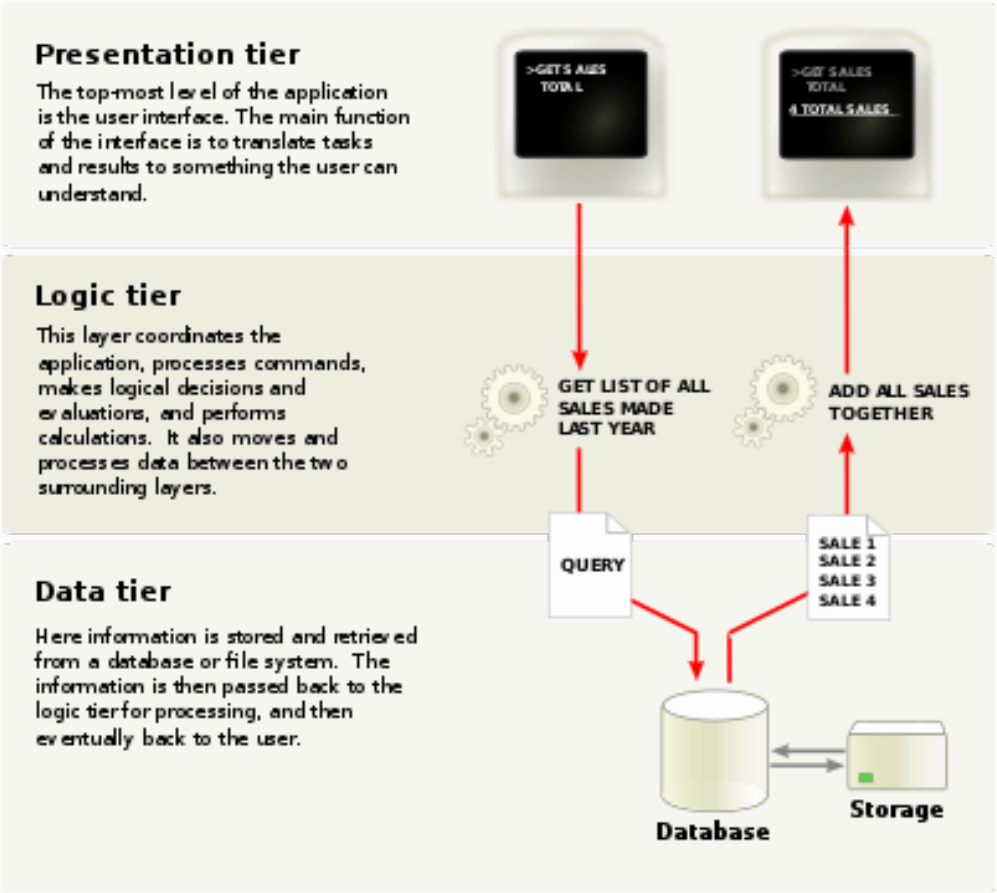
Separation of concerns!

A change to one layer can have no bearing on the rest of the model

e.g. Fiberoptic instead of Coax at the PHY layer

How does this apply to mobile?

Application designers often want separation of UI and logic!

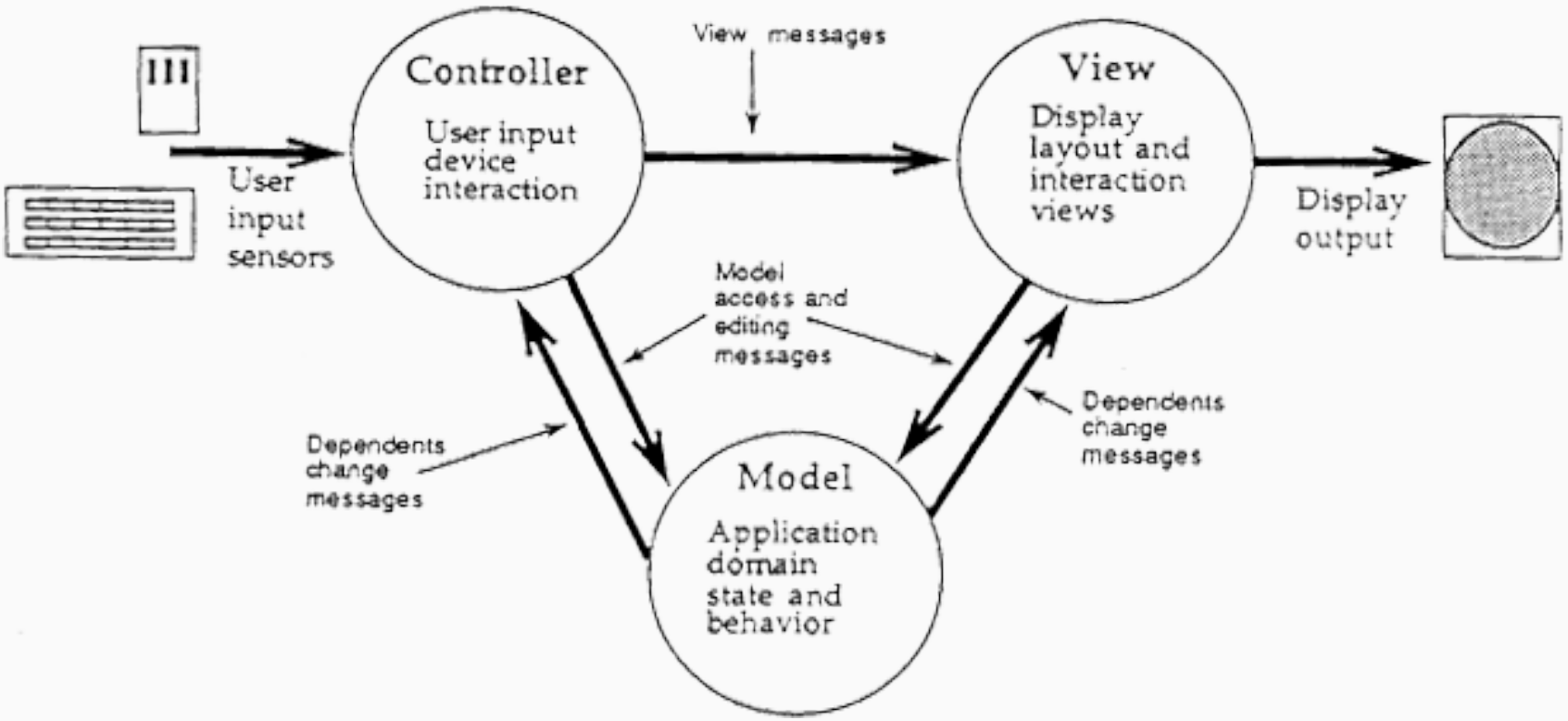


Three tier architecture

These software engineering abstractions relate to the MV* architectures that are common in mobile computing systems

Model View Controller (MVC)

Model View Controller



[1]

[1] Krasner 1988

Definitions

Model: Models are those components of the system application that actually do the work

View: Display aspects of the models

Controller: Used to send messages to the model, provide interface between model, views, and UI devices.

Models

Models enable encapsulation

Model encapsulates all data as well as methods to change them

- Can change the underlying data structures without having to recreate the model
- Can create multiple views and controllers without reimplementing the underlying model

Encapsulation enables scalability!

Models

Other advantages of models:

- Persistence – Model can pull data from a more permanent datastore (think local or cloud databases)
- Sharing – Models enable flexibility to share among multiple users (All manipulating same data instead of local copies)

Control

What does MVC control path look like?

- Views and controllers have exactly 1 model
- Model has 1 or more views and controllers associated with it
- Views and controllers need to know about model explicitly
- Models **should not know** about their views and controllers

Changes

Change in model is triggered by controller connecting user action to a message sent to the model.
Reflected in all views, not just the view associated with the controller which initiated the change

Dependencies

Views and controllers of a model are registered as dependents of the model

- Informed whenever aspect of the model has changed (subscriber?)
- Message passed, with parameters so that there are several types of model change messages
- View and controllers respond to the appropriate model changes

Interaction Cycle

User takes input action, and controller notifies model to change
Model carries out prescribed operations

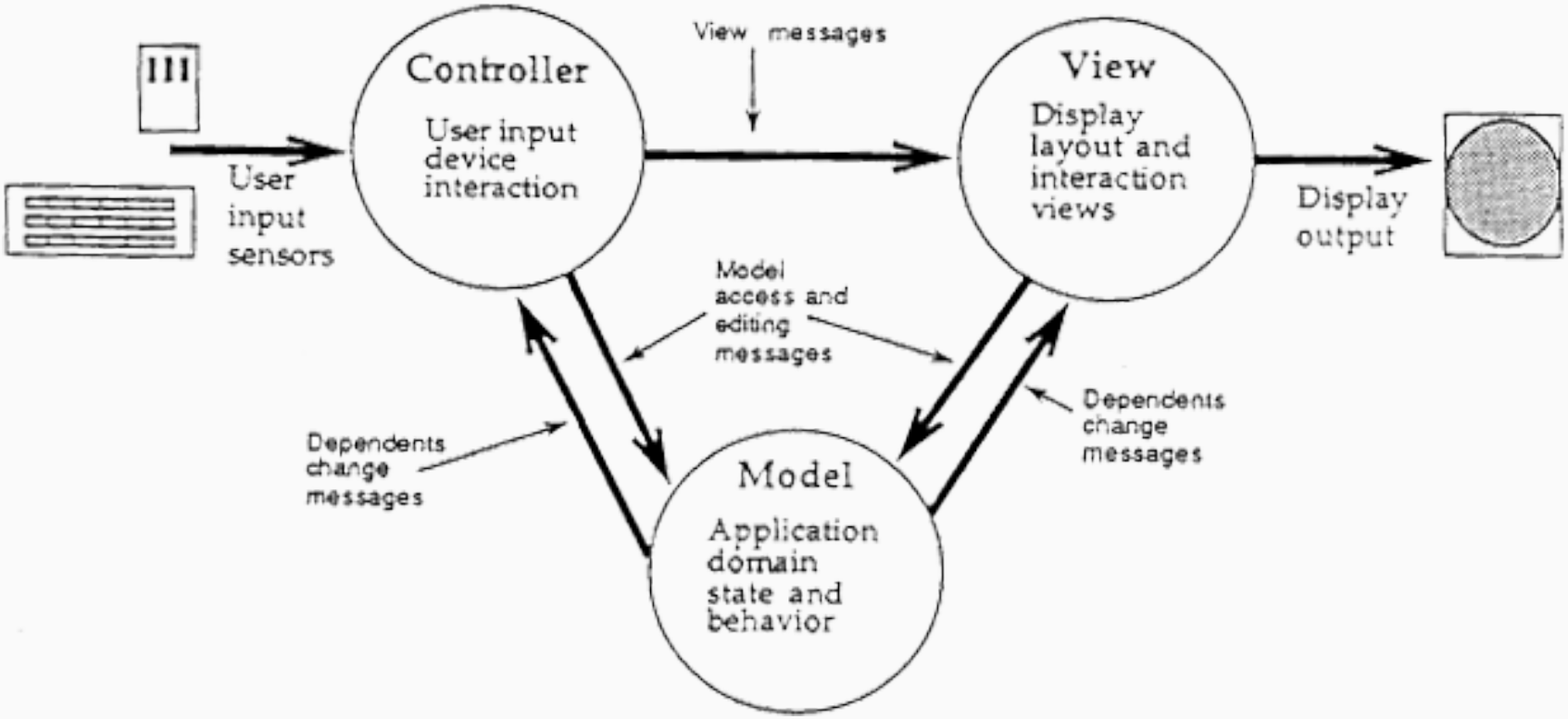
- possibly change state
- broadcasts to dependents that it has changed

Views can inquire about new state

- Update display if needed

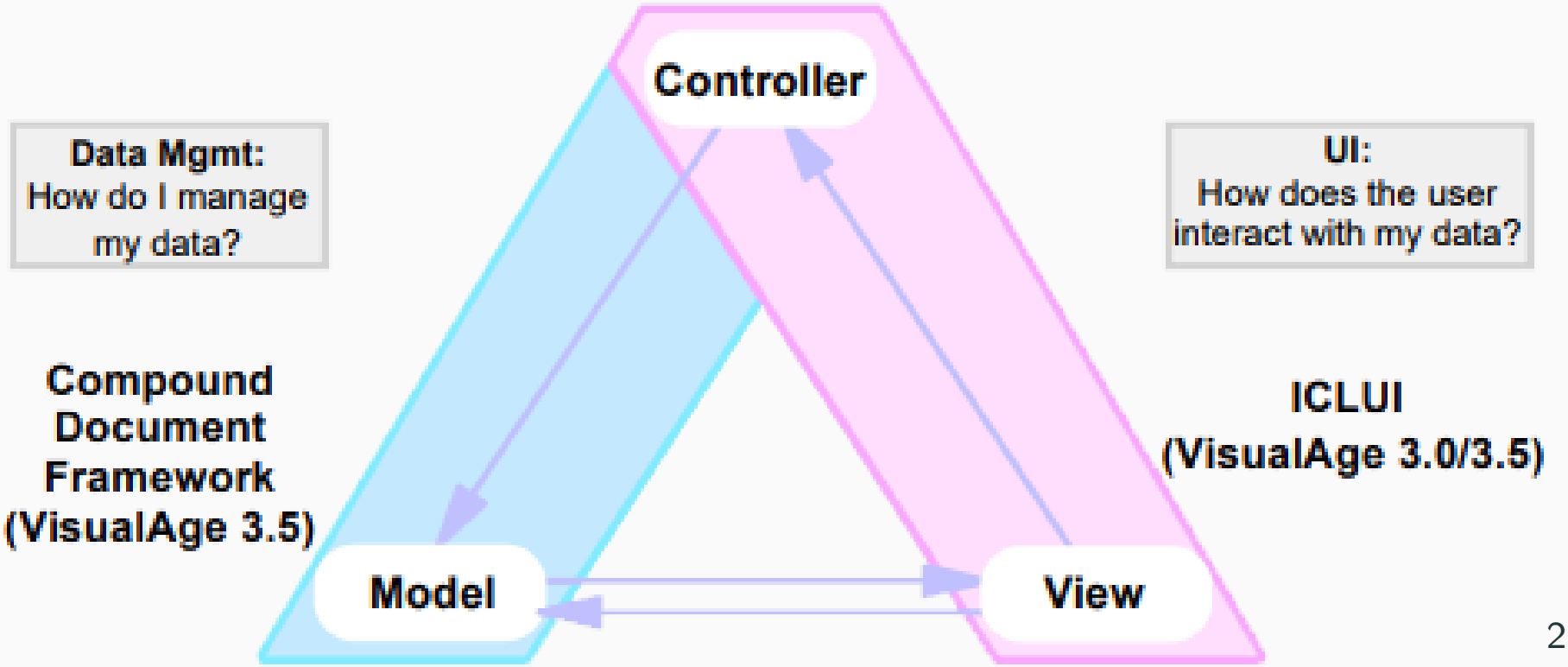
Controllers change their method of interaction depending on new state of the mode

Model View Controller



Model View Presenter (MVP)

Model View Presenter



[2] Potel 1996

Goals

Two fundamental concepts to a program:

1. Data Management
2. User Interface

These two concepts embody two basic design questions:

1. How do I manage my data?
2. How does my user interact with my data?

Goals

Two fundamental concepts to a program:

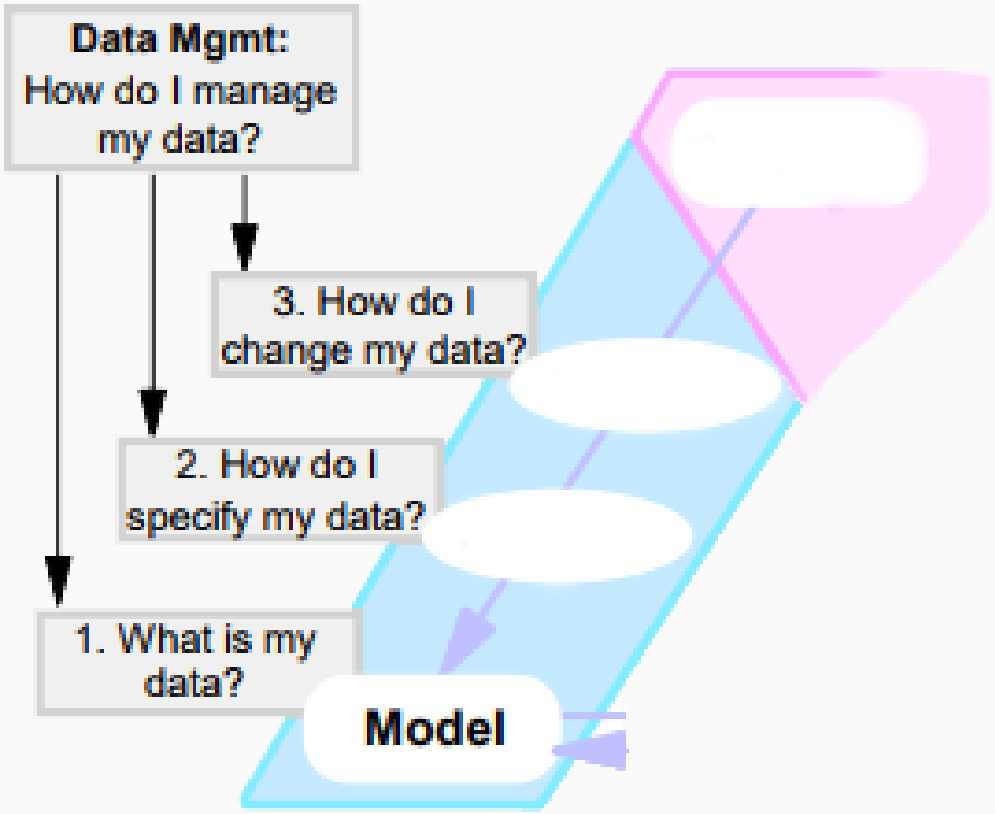
1. Data Management
2. User Interface

These two concepts embody two basic design questions:

1. How do I manage my data?
2. How does my user interact with my data?

MVP addresses these by formalizing a distinction between the model and the viewcontroller (now called presentation)

Data Management Questions



Data Management Questions

What is my data?

This question is the same as the model in MVC

How do I specify my data?

Selections

How do I change my data?

Commands

Selections and Commands

Selections are operations which allow the ability to choose a certain set from the data

Or, more formally...

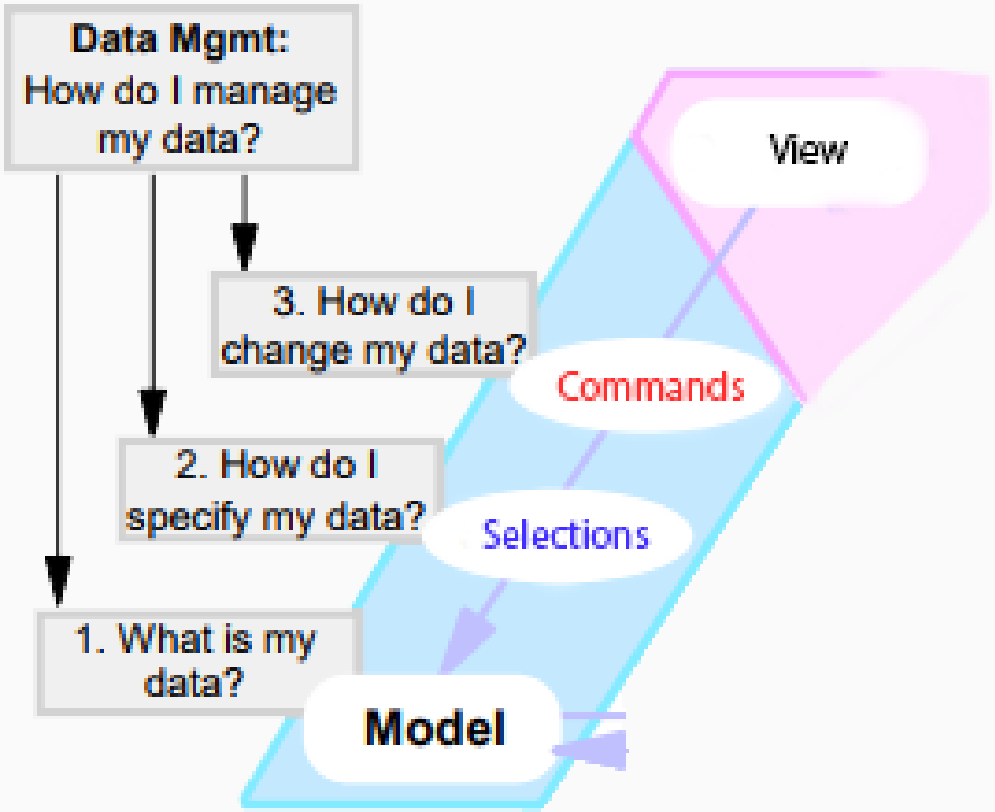
Selection $S \subset D$ where D is the set of all data

Commands are operations on the set of data

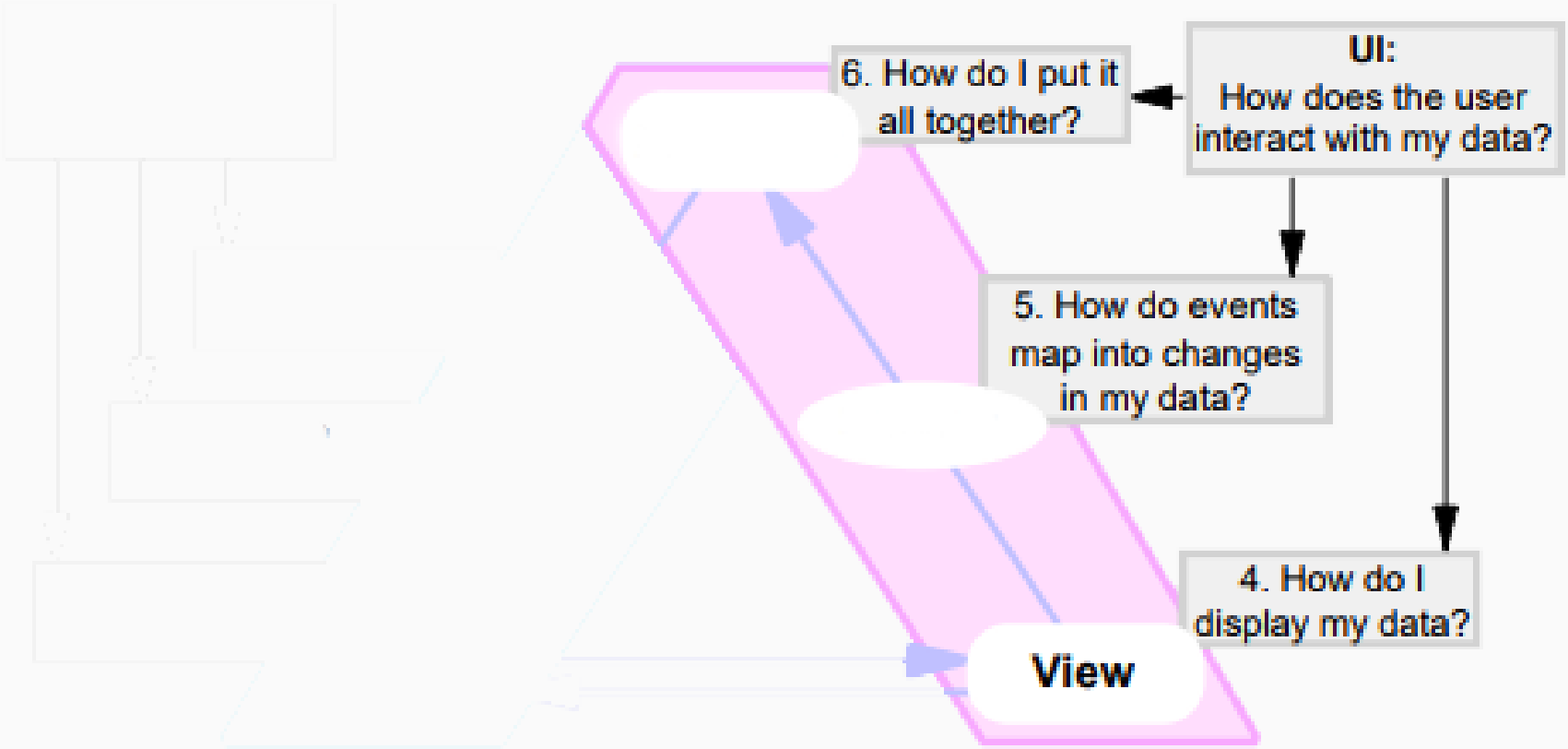
Or, more formally

Command C acts on **Selection** S, yielding **Selection** S'

Selections and Commands



User Interface Questions



User Interface Questions

How do I display my data?

This is the same as the View from MVC

How do events map into changes in my data?

Interactors

How do I put it all together?

Presenter

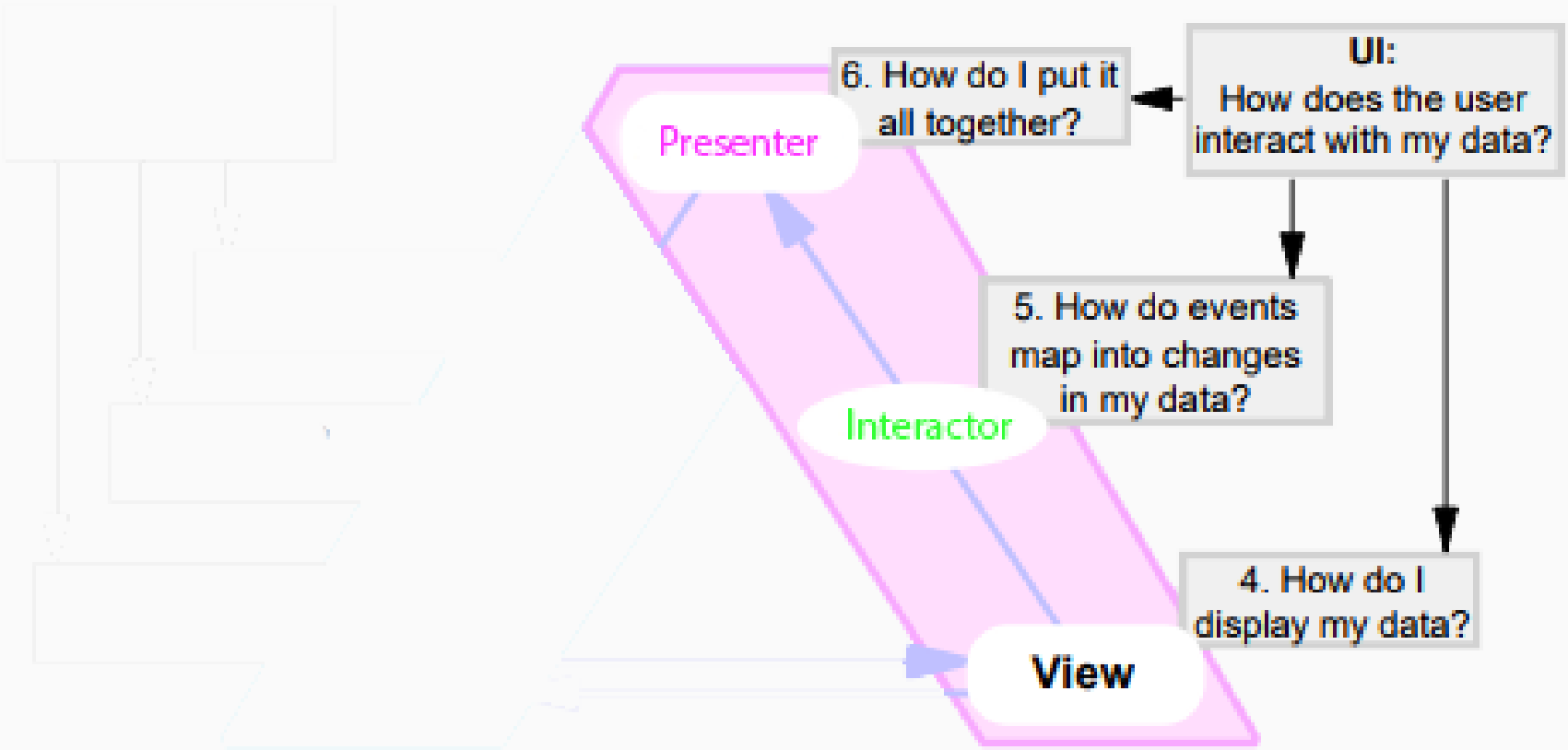
Interactors and Presenters

Interactors account for user interactions
e.g. Mouse tracking, clicking, menu selection, keyboard, specification of a **selection**, etc...

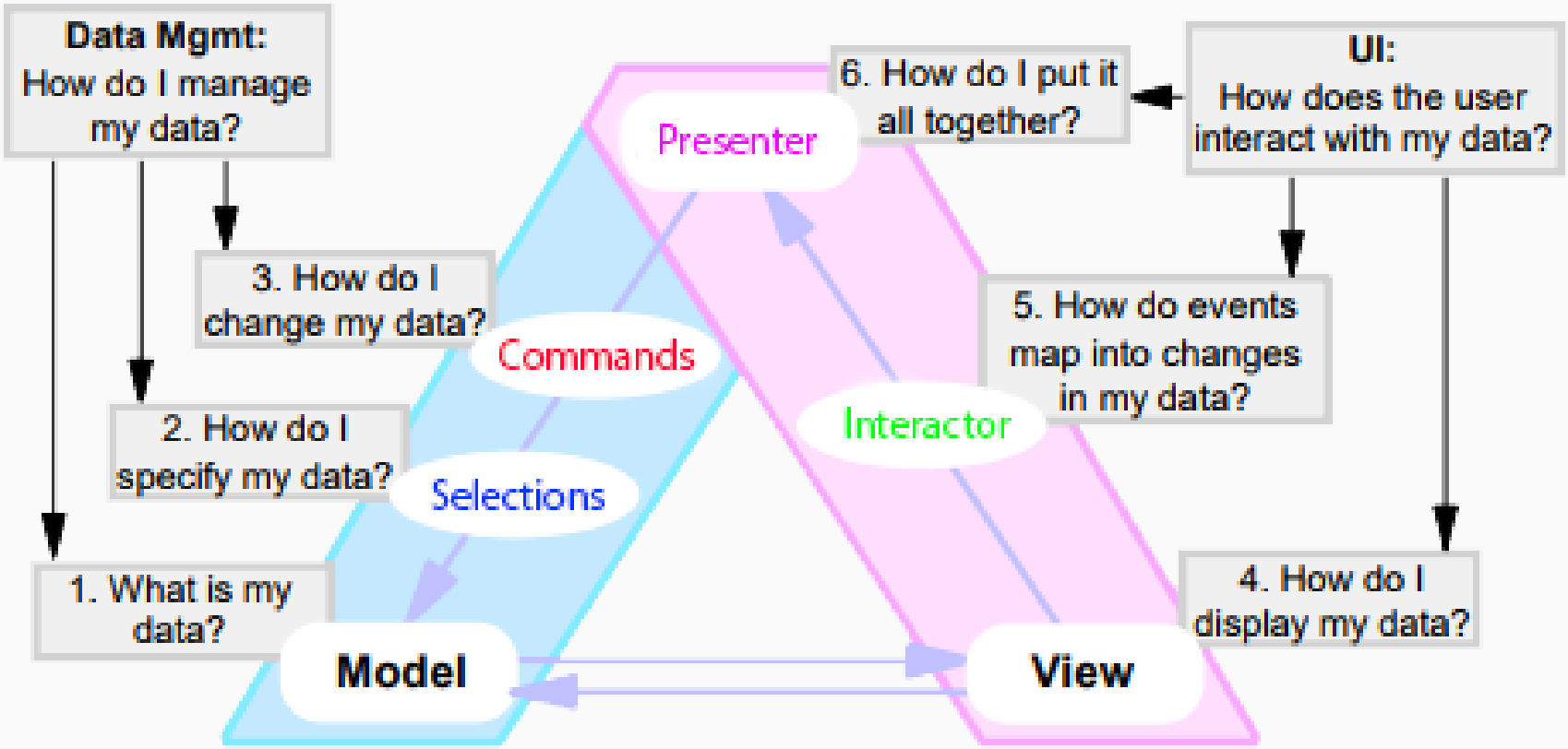
The **Presenter** is similar to the Controller of MVC, but with added functionality
The **Presenter** is responsible for:

- The traditional main/event loop
- Create models, selections, commands, views, and interactors
- Enable business logic that directs what happens and when

Interactors and Presenters

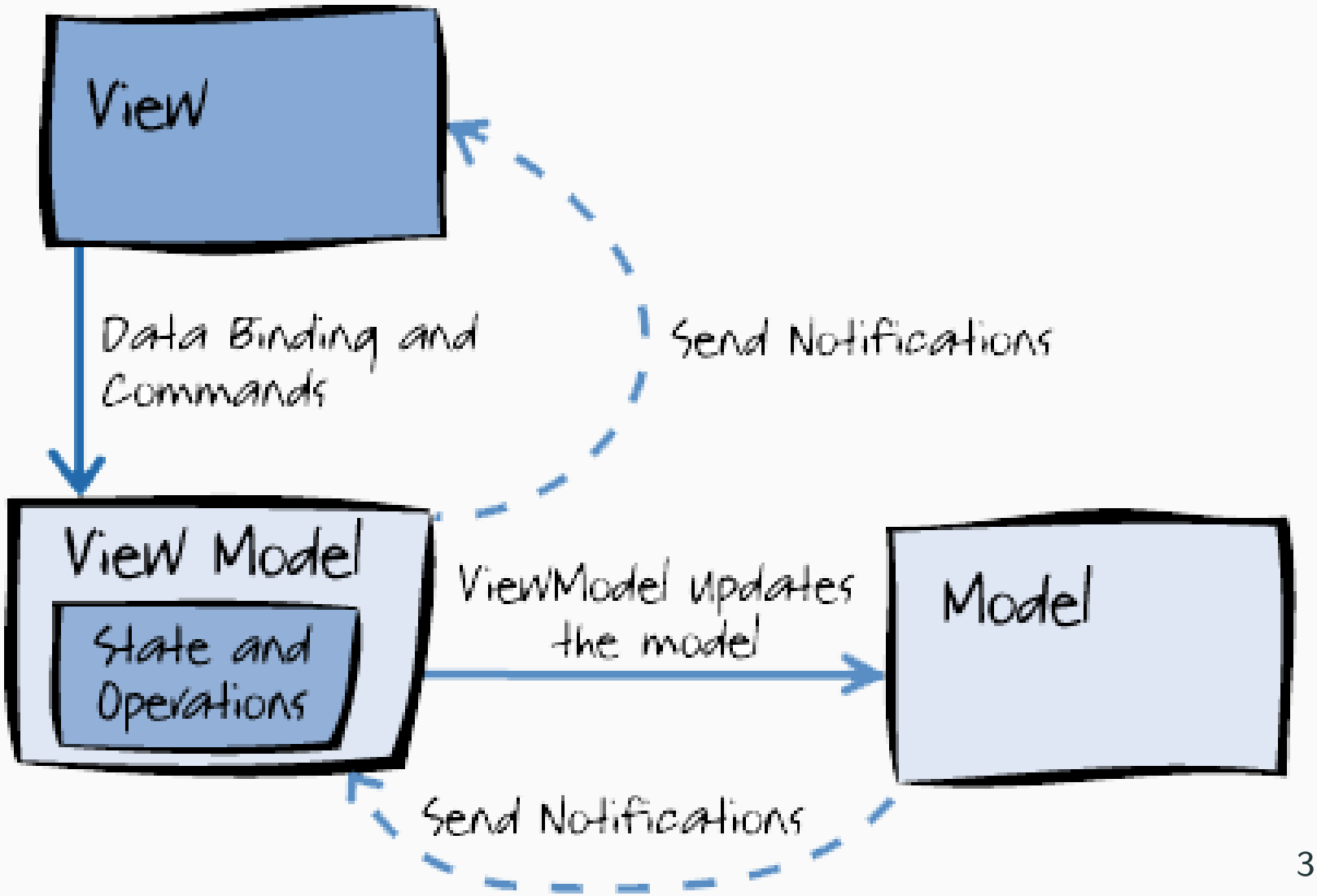


MVP - Put all together



Model View ViewModel (MVVM)

Model View ViewModel



[3] https://msdn.microsoft.com/en-us/library/hh848246.aspx

View

Responsible for defining structure, layout, and appearance of what user sees

Ideally defined purely with XAML (or XML)

Limited code which **does not contain business logic**

Model

Implementation of applications domain model

Includes business and validation logic

Includes repositories, business objects, data transfer objects, etc...

ViewModel

Intermediary between view and model

Responsible for handling view logic

Interacts with model by invoking methods in model classes

Provides data from model in a form that the view can use

Benefits

Developers and designers can work more independently and concurrently on components

- Designers can concentrate on the view using sample data
- Developers can work on the view model and model components

Create unit tests for the view model and the model without using the view

Benefits

Easy to redesign the UI of the application without touching the code

- View is implemented entirely in XML
- New versions of the view should work with an existing view model

Enables backwards compatibility

- Often changing existing models is risky. ViewModel acts as an adapter to views

What pattern should I use?

What pattern should I use?

There are lots of opinions

https://academy.realm.io/posts/eric-maxwell-mvc-mvp-and-mvvm-on-android/

Short Answer:

Any are sufficient for this class

None of the projects (except maybe the final) are big enough to necessitate separation of concerns

However...

MVC is being replaced by MVVM and MVP in many live dev scenarios

What pattern should I use?

There are lots of opinions

https://academy.realm.io/posts/eric-maxwell-mvc-mvp-and-mvvm-on-android/

Short Answer:

Any are sufficient for this class

None of the projects (except maybe the final) are big enough to necessitate separation of concerns

However...

MVC is being replaced by MVVM and MVP in many live dev scenarios

What pattern should I use?

There are lots of opinions

https://academy.realm.io/posts/eric-maxwell-mvc-mvp-and-mvvm-on-android/

Short Answer:

Any are sufficient for this class

None of the projects (except maybe the final) are big enough to necessitate separation of concerns

However...

MVC is being replaced by MVVM and MVP in many live dev scenarios

You should know is how to separate concerns in code with hierarchies

Readings

Reading

MVC - Krasner (1988)

MVP - Potel (1996)

Google Samples MVP and MVVM **(Just know that this exists)**

Read these by Monday, and be prepared to discuss