Model - View - Controller

A Design Pattern

Hunter Chapman

Suggestions: PR @BootCoder

Why you care

- MVC gives your code a solid architecture that will soon feel like the back of your hand.
- Easily implemented pattern allowing for you to follow other Developers code and vice versa.

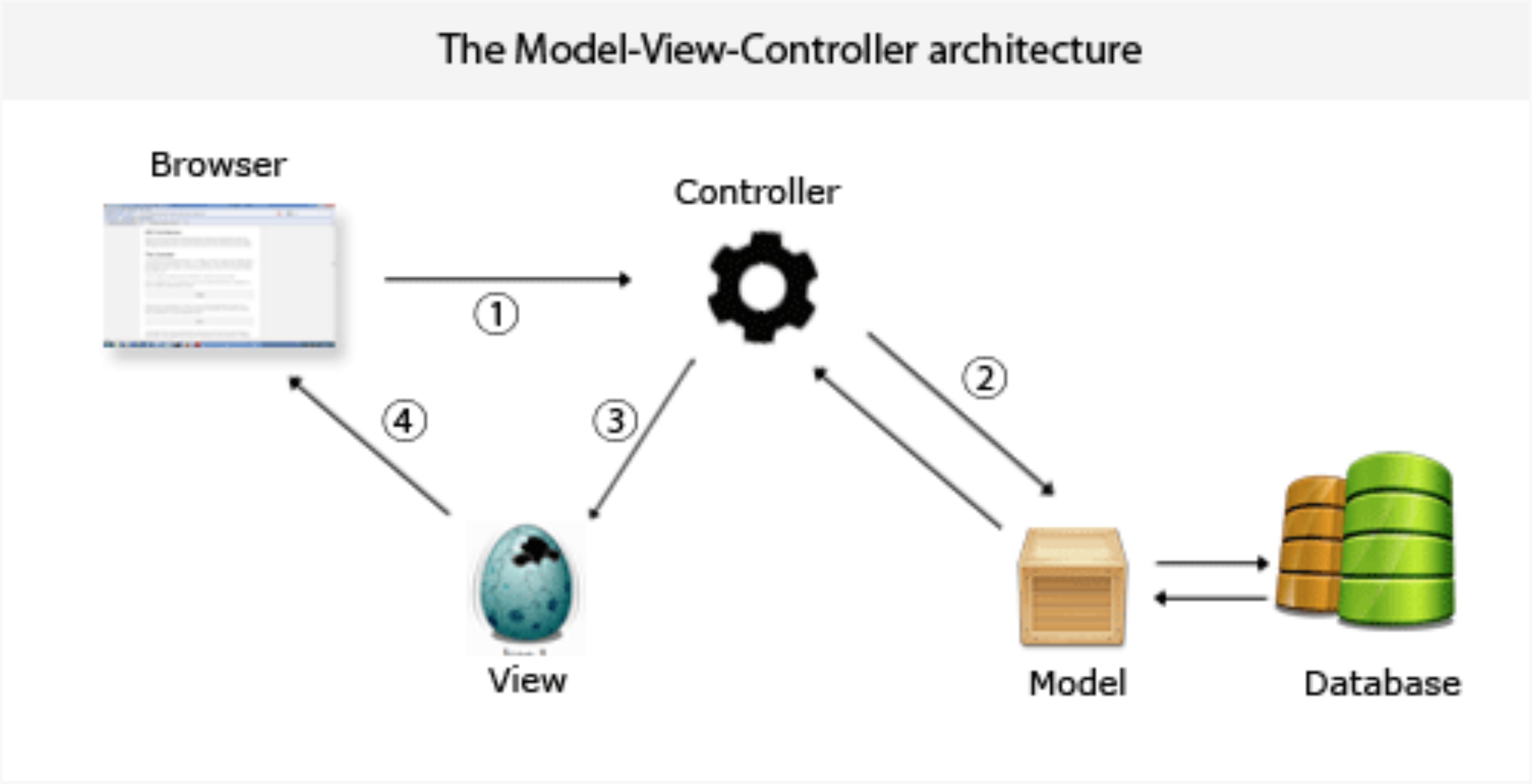
M-V-C Basically:

- MVC is not much different than the code you are already writing.
- Just a bunch of classes and modules put in specific locations!
- The critical change is following a pattern of organization which allows us to separate out responsibilities in a standardized way.

Model - View - Controller

- What is it?
- In short: MVC is a design pattern.
- In not so short: MVC is a software architectural pattern for implementing user interfaces. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.

M-V-C Summation:



Design Patterns

- A design pattern is a general reusable solution to a commonly occurring problem within a given context of software design.
- Get used to this term. It can have a transformational impact on how you think about the art of code.

Design Patterns

- Start thinking of everything you do as implementing some pattern.
- You're just following a blueprint, filling in the gaps as needed with as little distinction from the standard pattern as possible.

MVC Related Quandaries

Sure, you can write all the codes… Where does it go?

- Whose business is it to talk to users?
- What about displaying output?
- Should your classes all know how to parse user input?
- Where do heavy algorithms run?
- How do you separate 10,000 lines of code into a manageable structure?

MVC Restaurant

- Does your waiter make your food?
- Do they know all the recipes?
- Do they run the kitchen?
- What *is* a waiter responsible for?

Everyone has their own role.
If you want a well-run restaurant,
you need to divvy up the work.

WHAT ROLES DO WE NEED?

- The waiters handle customer interaction
- They take your order and deliver your food

- The chef combines ingredients to make the food
- They know the recipes and gather ingredients

- The kitchen manager oversees the whole process
- They manage the orders coming in, tell the cooks what they need to make, and ensure the orders are properly put together for delivery to the waiter

M-V-C Bits:

- The model (chef) handles all the data. Interacts with a DB, Web API or other data store, crunches the numbers and does the ‘heavy lifting’ in your application.

- The view (waiter) handles user interactions. It shows information to the user and gets user input

- The controller (manager) manages communication between the model and view. It runs literally everything!

M-V-C Bits:

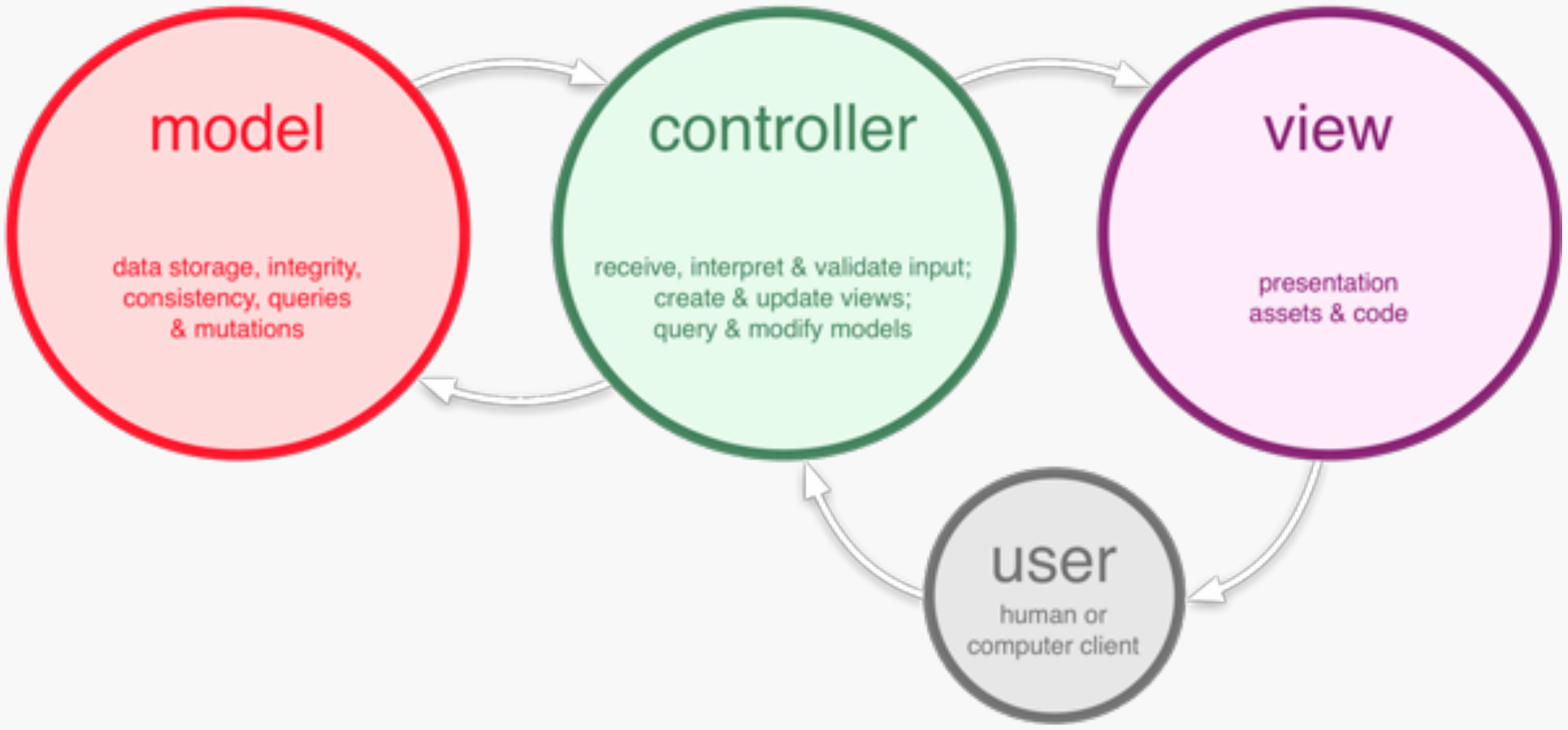
Everything passes through the Controller

It’s the central hub handling various functionalities within your application. It makes decisions based on the input received. Then acquires and packages the necessary information from the model, and passes it back out to the view.

Recurring Problem solved with M-V-C

- You have some data and business logic that a backend team takes care of.
- Your user interface is changing on a regular basis and rendered across multiple devices.
- You need to facilitate communications between them.

Model - View - Controller


model
data storage, integrity, consistency, queries & mutations
controller
receive, interpret & validate input; create & update views; query & modify models
view
presentation assets & code
user
human or computer client

M-V-C Design Pattern

Model

Controller

View

M-V-C Design Pattern

Model

Data &
Business logic

Controller

View

M-V-C Design Pattern

Model

Data &
Business logic

- classes
- modules
- databases

Controller

View

M-V-C Design Pattern

Model	Controller	View
Data & Business logic • classes • modules • databases		User interface

M-V-C Design Pattern

Model

Data &
Business logic

- classes
- modules
- databases

Controller

View

User interface

- get input
- display data
- HTML

M-V-C Design Pattern

Model

Data &
Business logic

- classes
- modules
- databases

Controller

Communicate

View

User interface

- get input
- display data
- HTML

M-V-C Design Pattern

Model

Data &
Business logic

- classes
- modules
- databases

Controller

Communicate

- parses input
- update model
- access model
- deliver data to the view

View

User interface

- get input
- display data from model
- HTML

M-V-C TODO's

How would TODO's look following the M-V-C pattern?

M-V-C TODO's

Model	Controller	View
Data & Business logic	Communicate	User interface

M-V-C TODO's

Model

Data &
Business logic

- List
- Task
- CSVParsing

Controller

Communicate

View

User interface

M-V-C TODO's

Model

Data &
Business logic

- List
- Task
- CSVParsing

Controller

Communicate

- TODO-Controller

View

User interface

M-V-C TODO's

Model

Data &
Business logic

- List
- Task
- CSVParsing

Controller

Communicate

- TODO-Controller

View

User interface

- Views
    - Options
    - Task List
    - Task View

Message Pattern

- All of programming is essentially passing messages between all the things.
- Start thinking of your methods/classes as a means of passing a message to a place.
- That place does a thing and passes some message back.

Message Pattern

Implementing the Message Pattern is much easier when methods can be relied upon to return a single message only.

This is called the Single Responsibility Design Pattern

Single Responsibility

- A design pattern whereby you structure your methods to carry out one specific task.
- This DOES NOT mean everything you write is a one liner.
- Will absolutely increase the modularity and reusability of your code base.

Single Responsibility

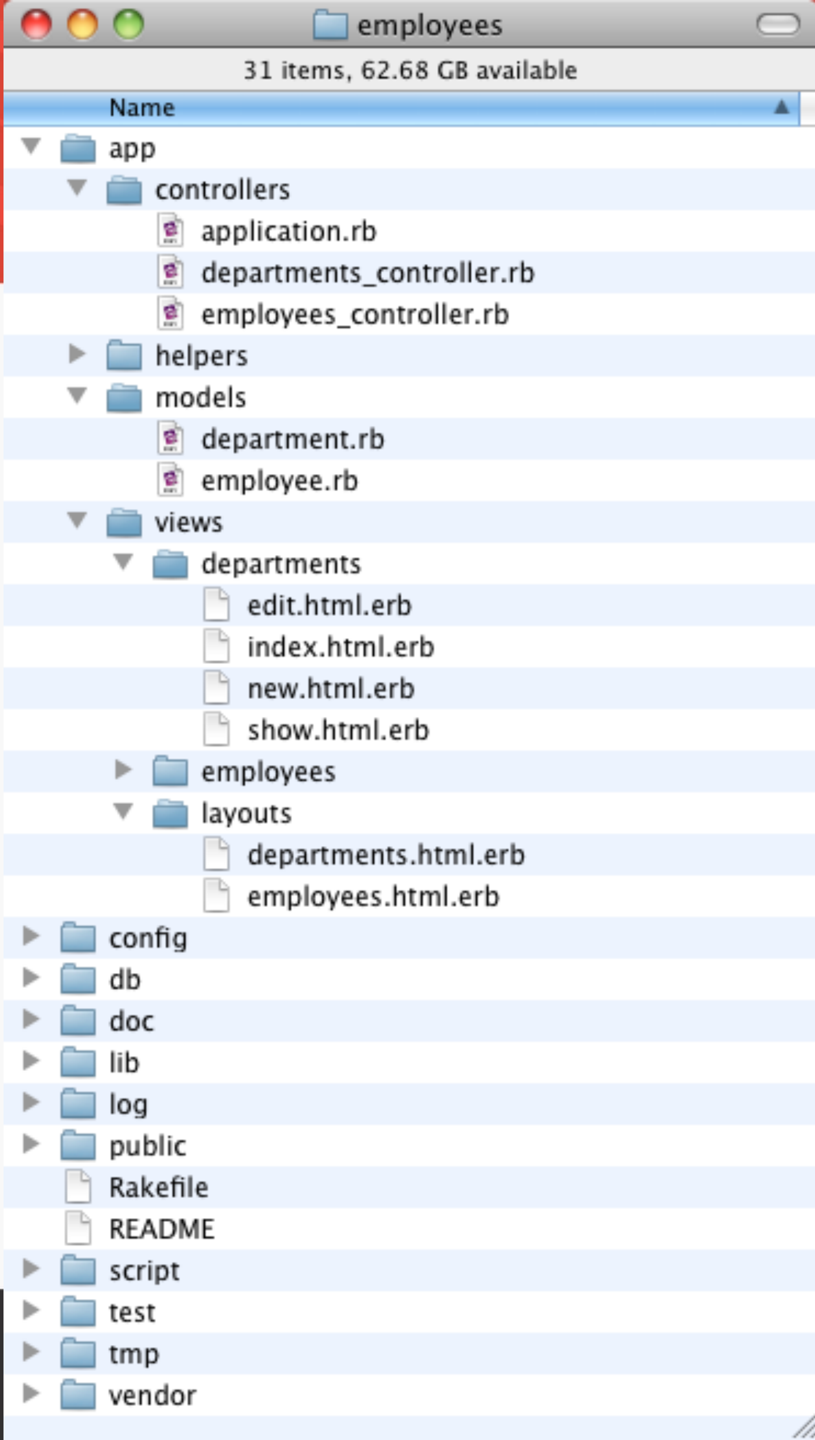
Couple good articles

http://jjbohn.info/blog/2014/07/28/single-responsibility-principle-a-solid-week/

https://robots.thoughtbot.com/back-to-basics-solid

Files

The vast majority of MVC apps will follow a file structure very similar to this.



```
employees
31 items, 62.68 GB available
Name
▼ app
    ▼ controllers
        application.rb
        departments_controller.rb
        employees_controller.rb
    ▶ helpers
    ▼ models
        department.rb
        employee.rb
    ▼ views
        ▼ departments
            edit.html.erb
            index.html.erb
            new.html.erb
            show.html.erb
        ▶ employees
        ▼ layouts
            departments.html.erb
            employees.html.erb
▶ config
▶ db
▶ doc
▶ lib
▶ log
▶ public
  Rakefile
  README
▶ script
▶ test
▶ tmp
▶ vendor
```

MVC in the Terminal

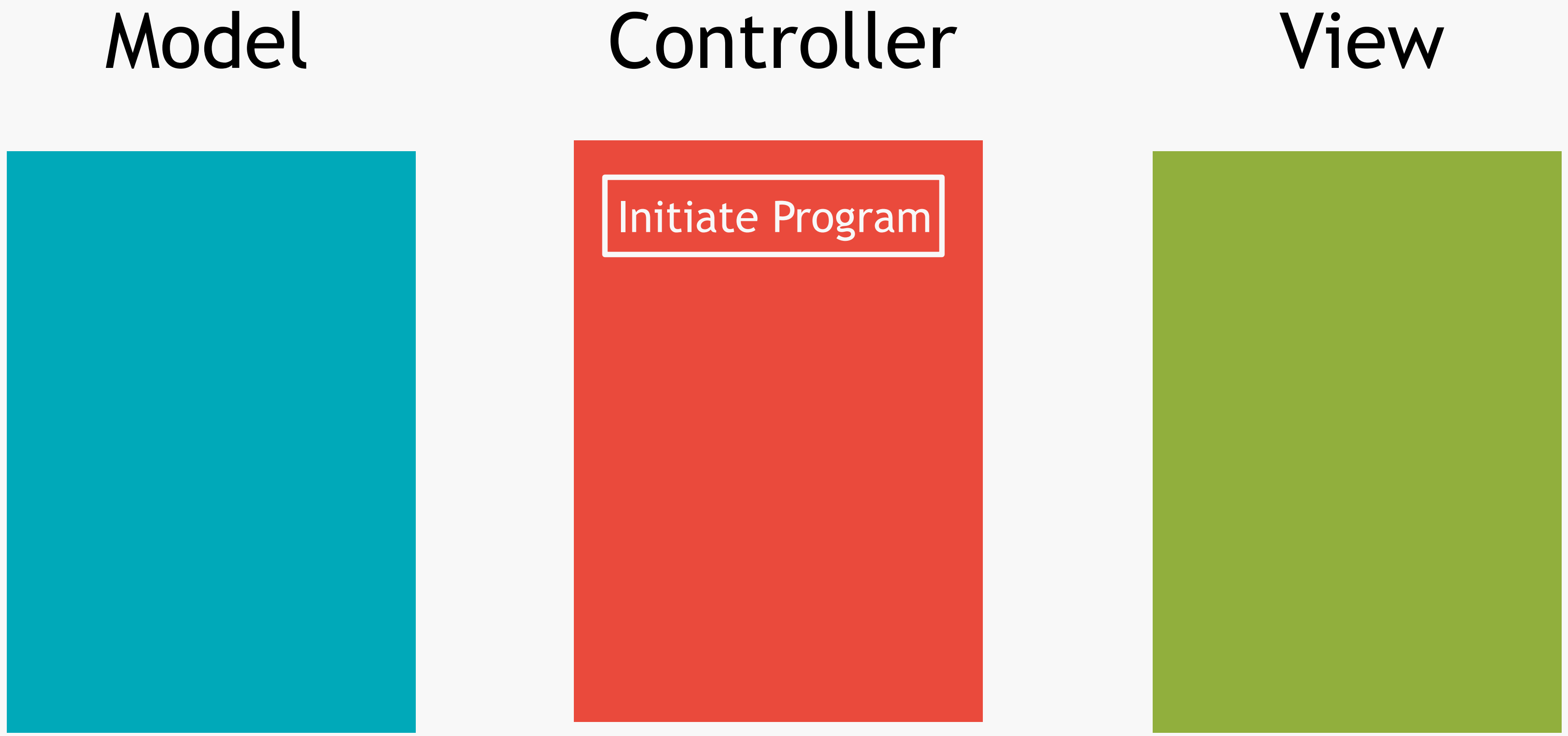
Spoiler Alert:

- MVC is kinda clunky to implement in the terminal. Not really, but a little… :-P
- View class can seem redundant since you can just puts out from where ever. (ps NOT LEGIT)
- Crucial to begin implementing this pattern now. These concepts needs to be rock solid before you hit the browser in phase 2.

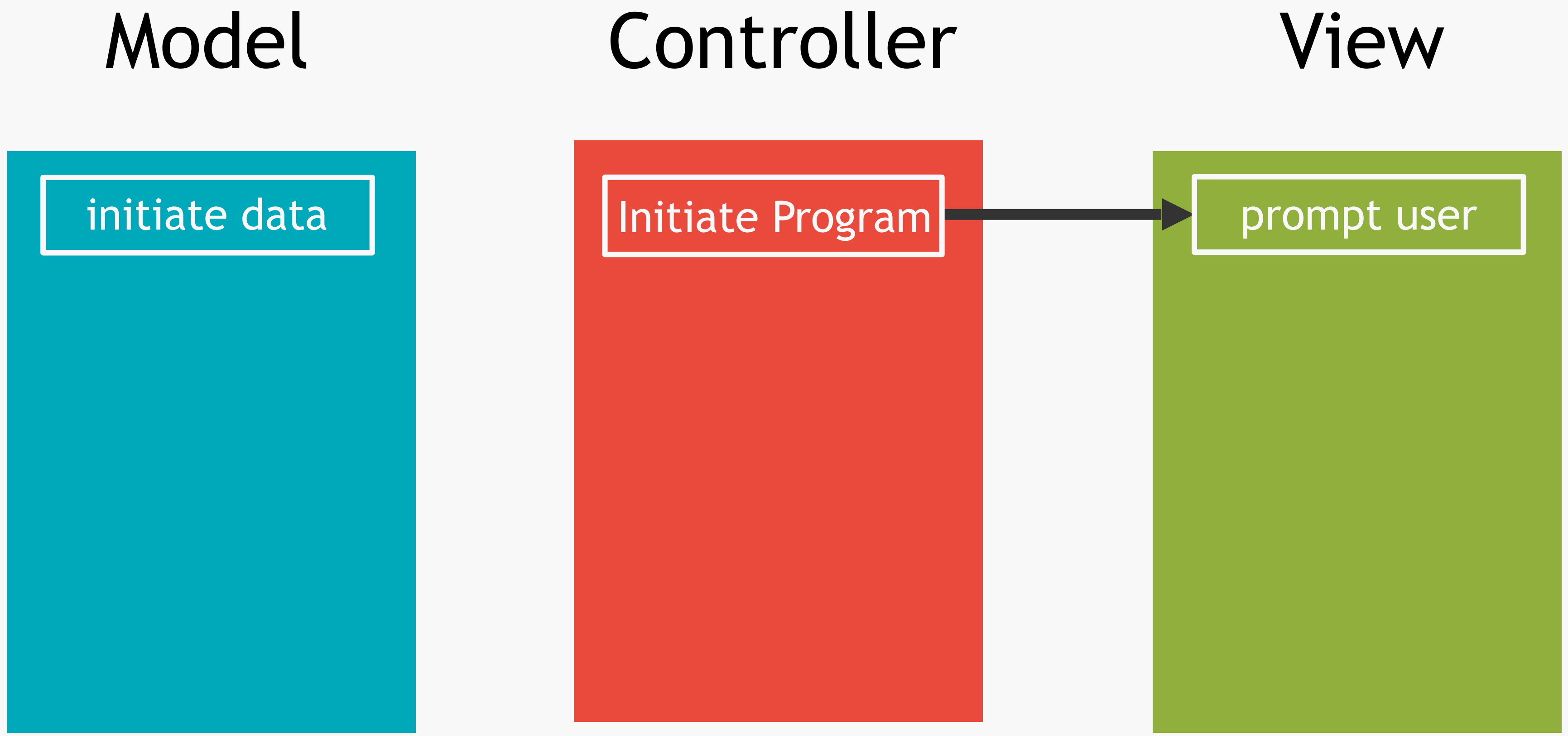
TODO’s: List

What does the
Single Responsibility Message Pattern
look like in MVC?

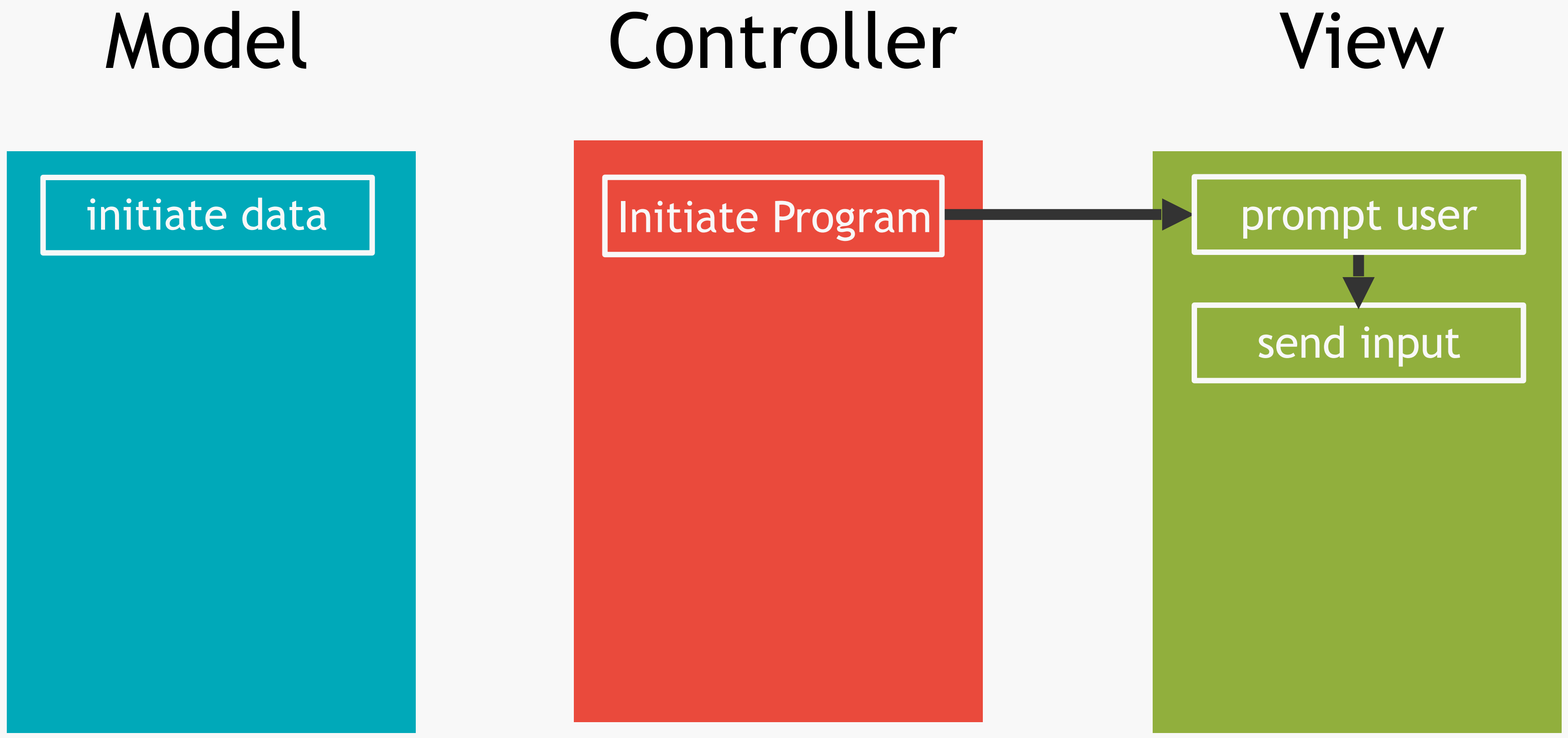
TODO’s: List



TODO's: List

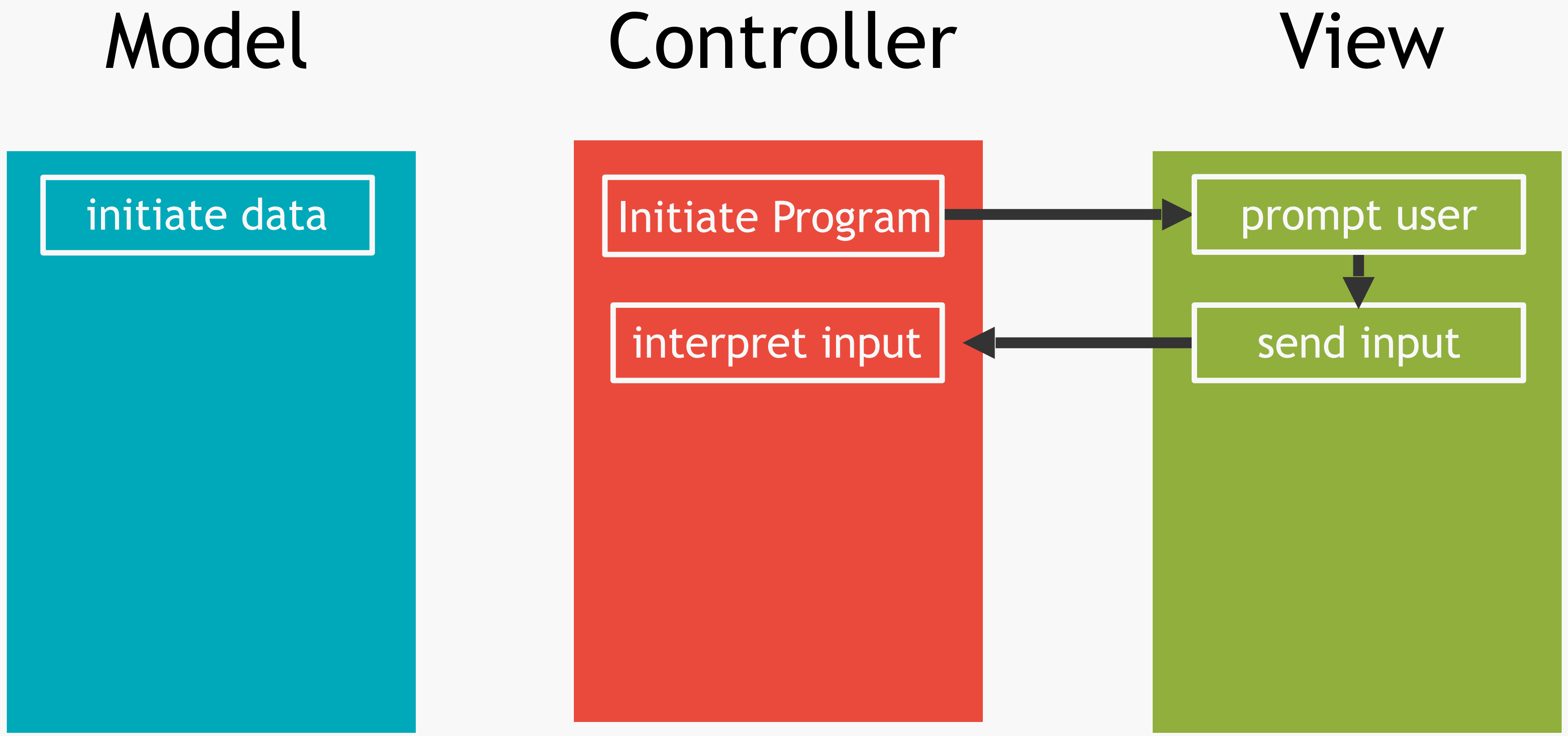

Model
Controller
View
initiate data
Initiate Program
prompt user

TODO's: List

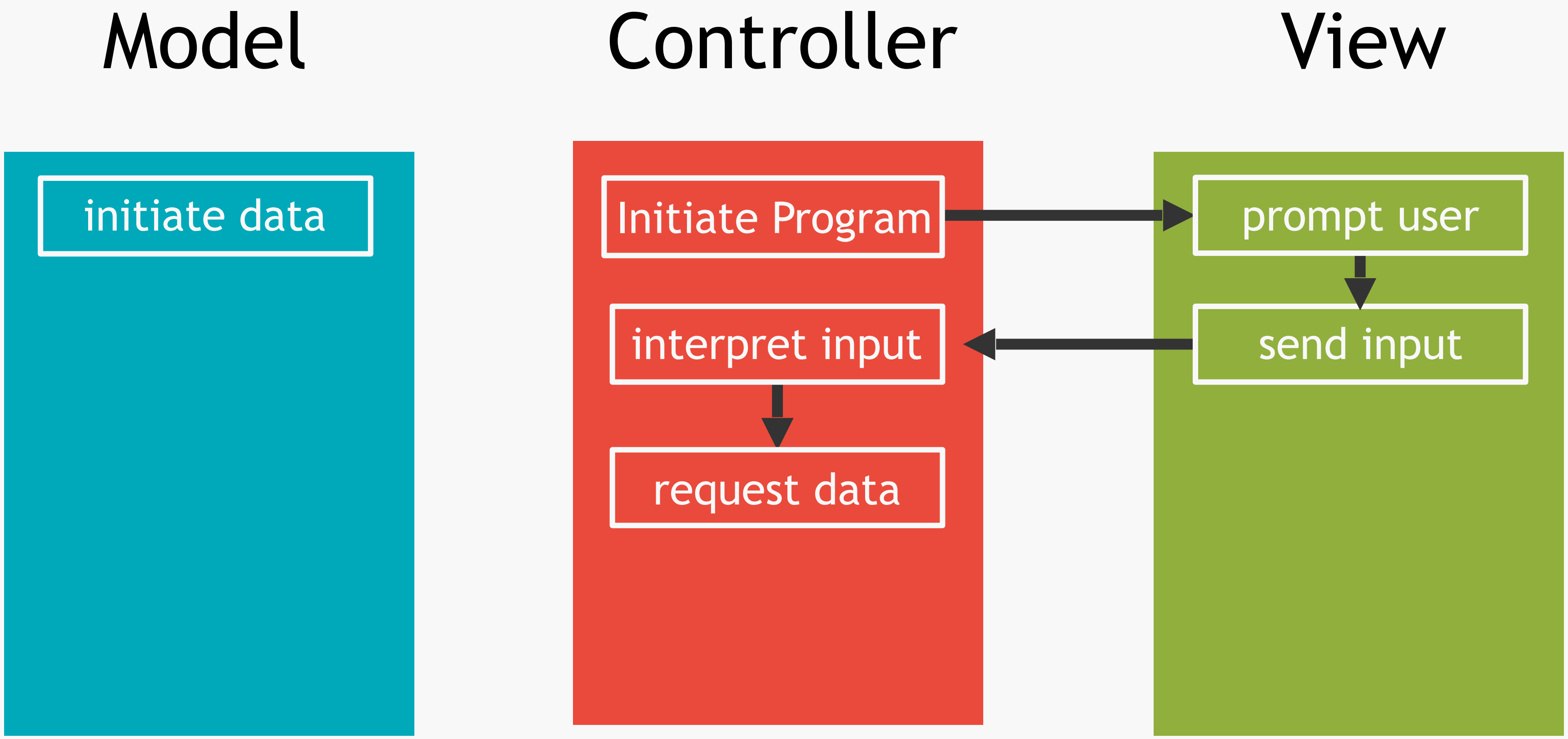


Model
Controller
View
initiate data
Initiate Program
prompt user
send input

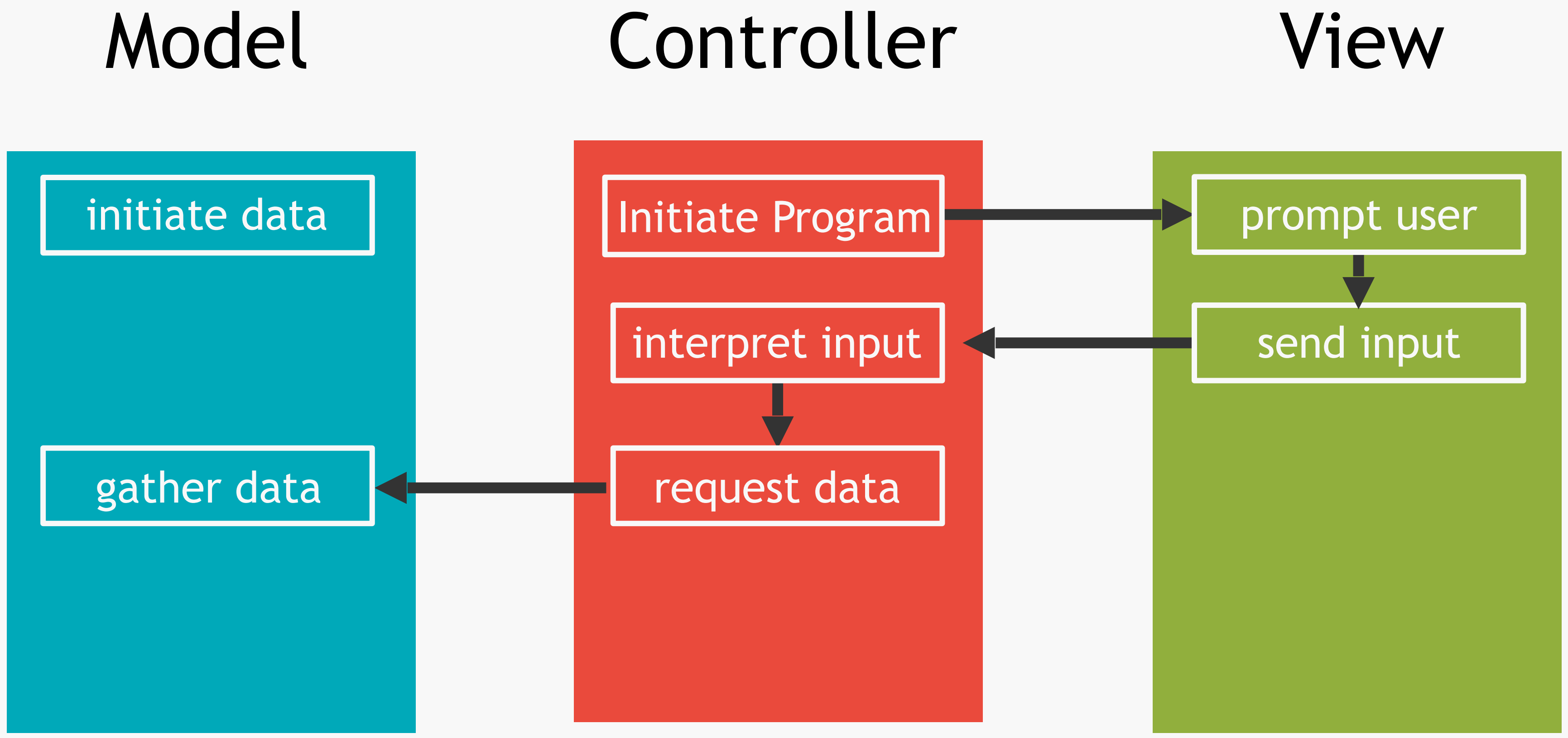
TODO's: List


Model
Controller
View
initiate data
Initiate Program
interpret input
prompt user
send input

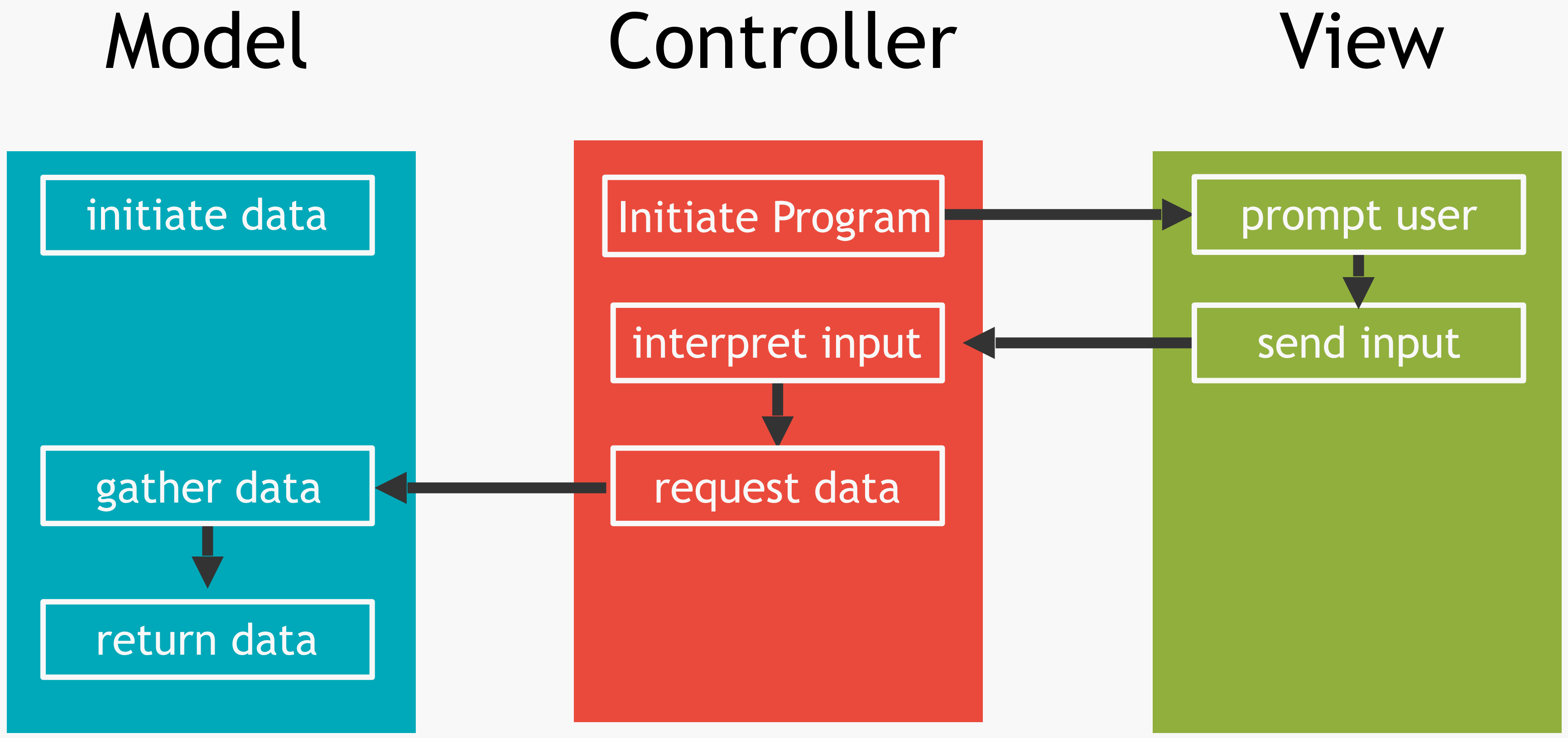
TODO's: List


Model
Controller
View
initiate data
Initiate Program
interpret input
request data
prompt user
send input

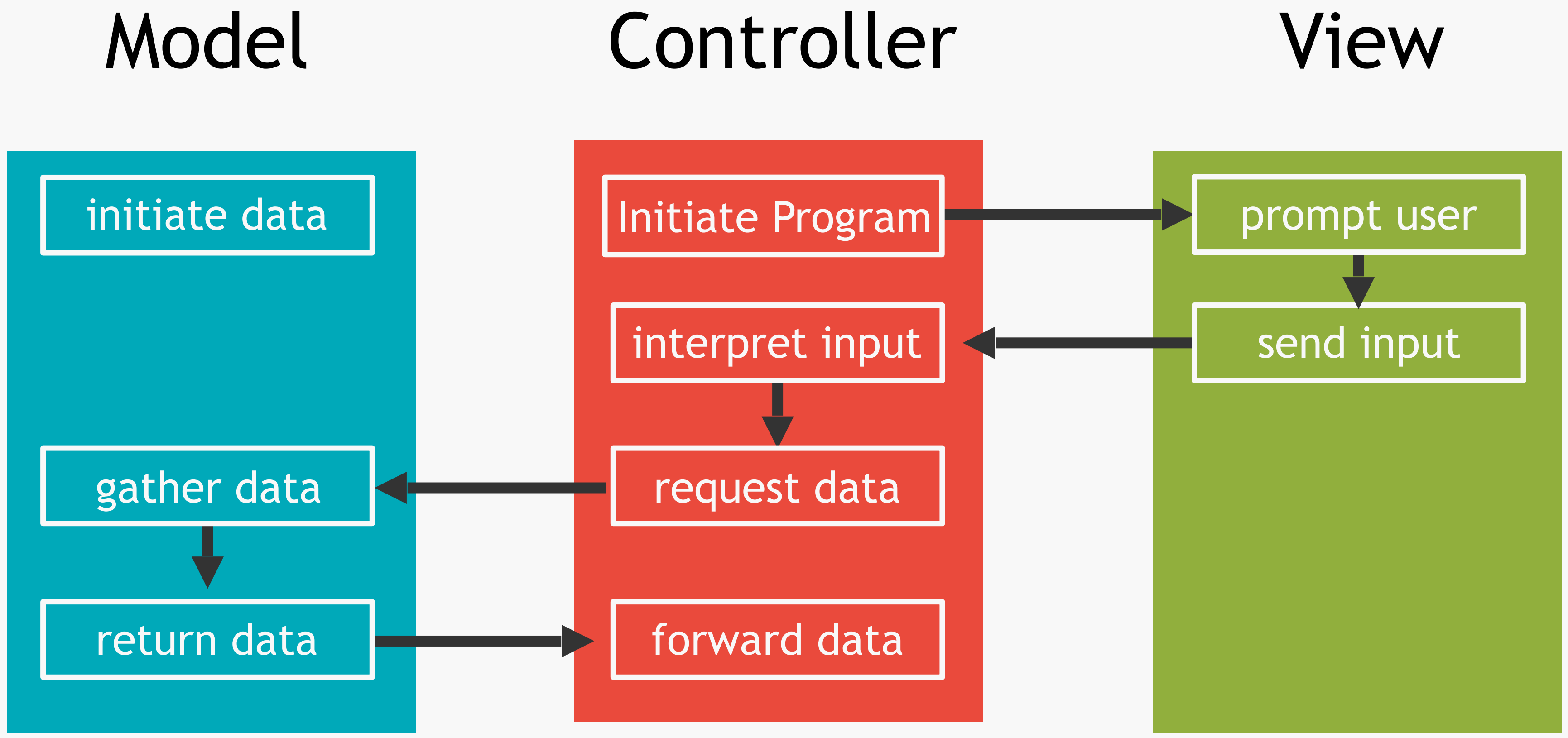
TODO's: List


Model
Controller
View
initiate data
Initiate Program
prompt user
interpret input
send input
gather data
request data

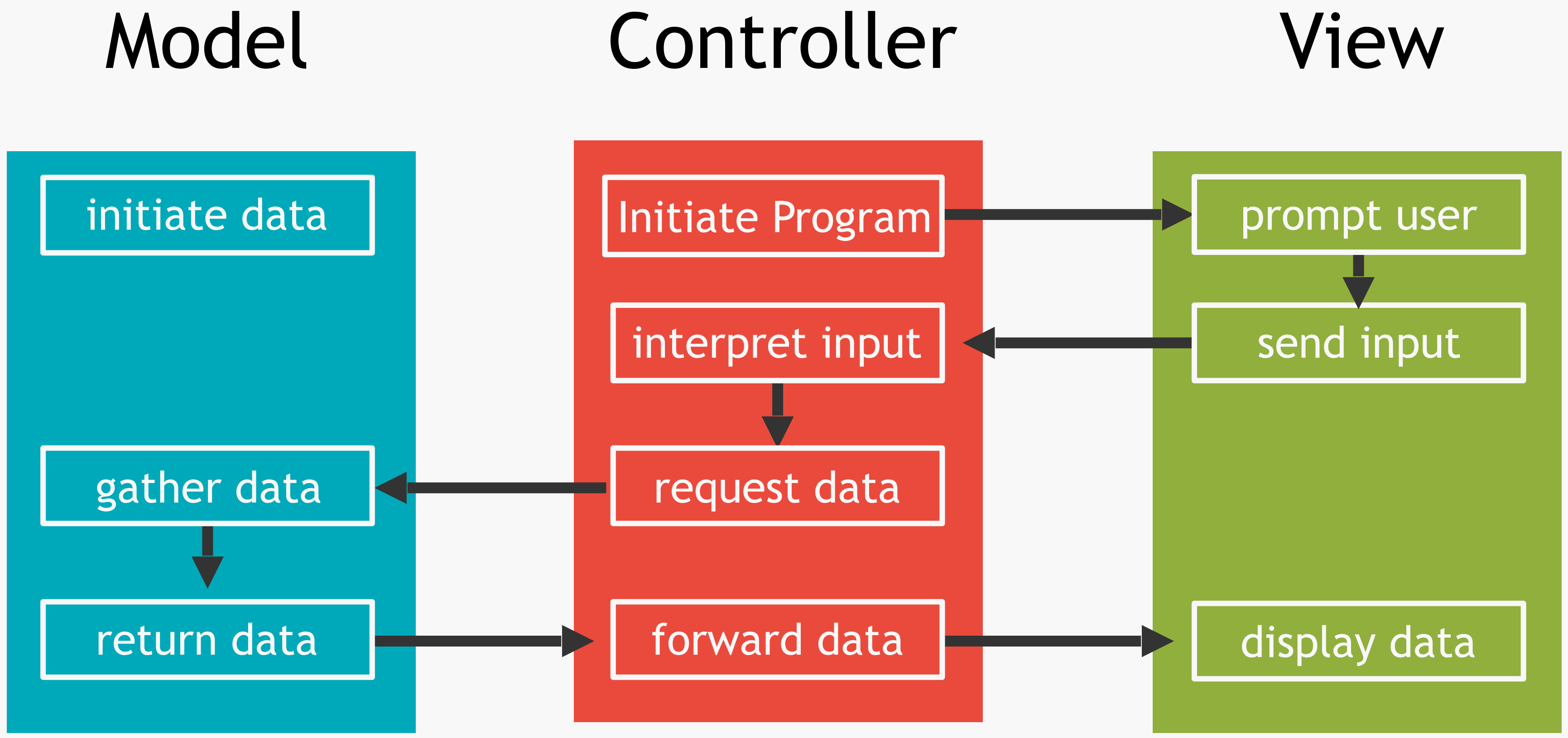
TODO's: List


Model
Controller
View
initiate data
Initiate Program
prompt user
interpret input
send input
gather data
request data
return data

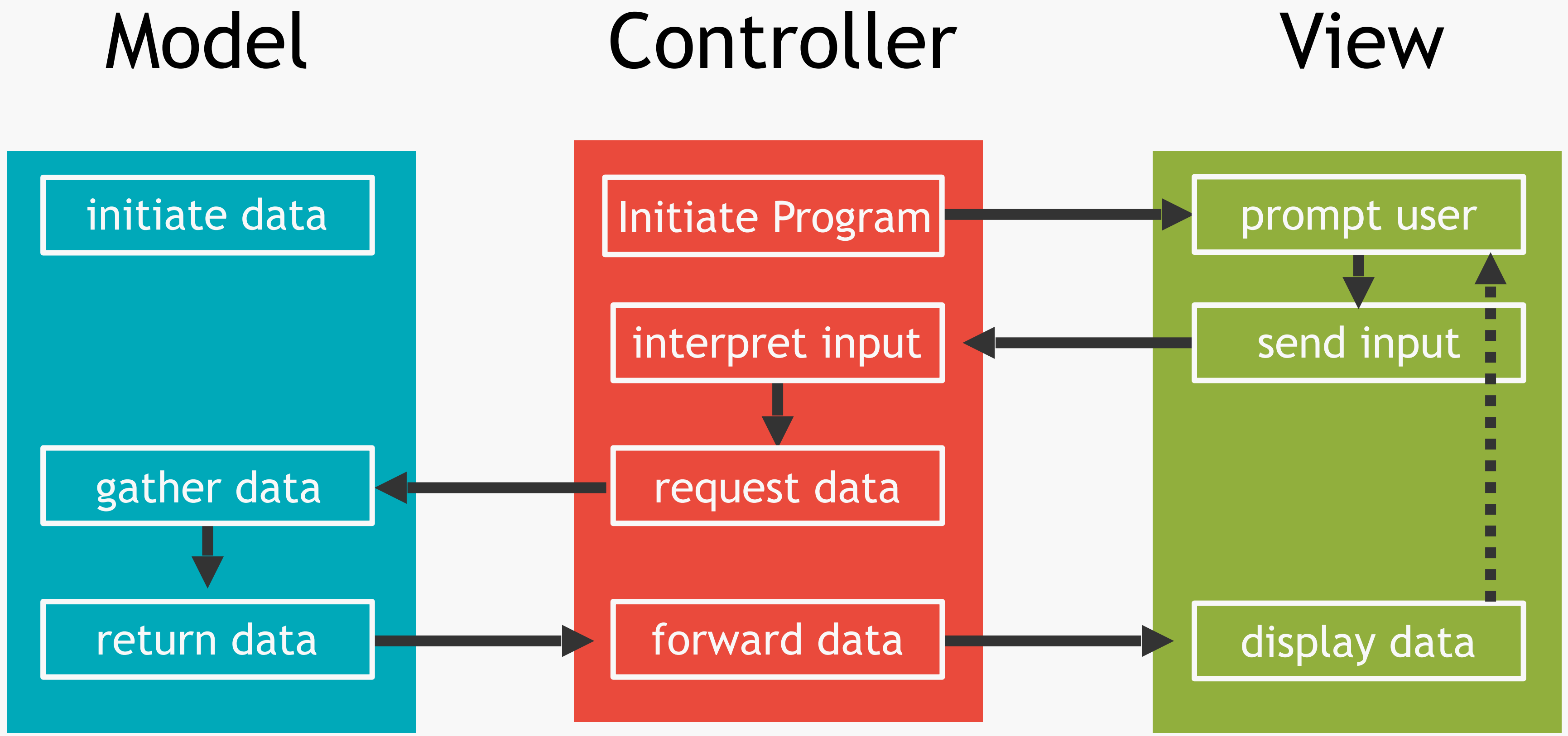
TODO's: List


Model
Controller
View
initiate data
Initiate Program
prompt user
interpret input
send input
gather data
request data
return data
forward data

TODO's: List


Model
Controller
View
initiate data
Initiate Program
prompt user
send input
interpret input
request data
gather data
return data
forward data
display data

Everything you need to visualize MVC:



FIN: