

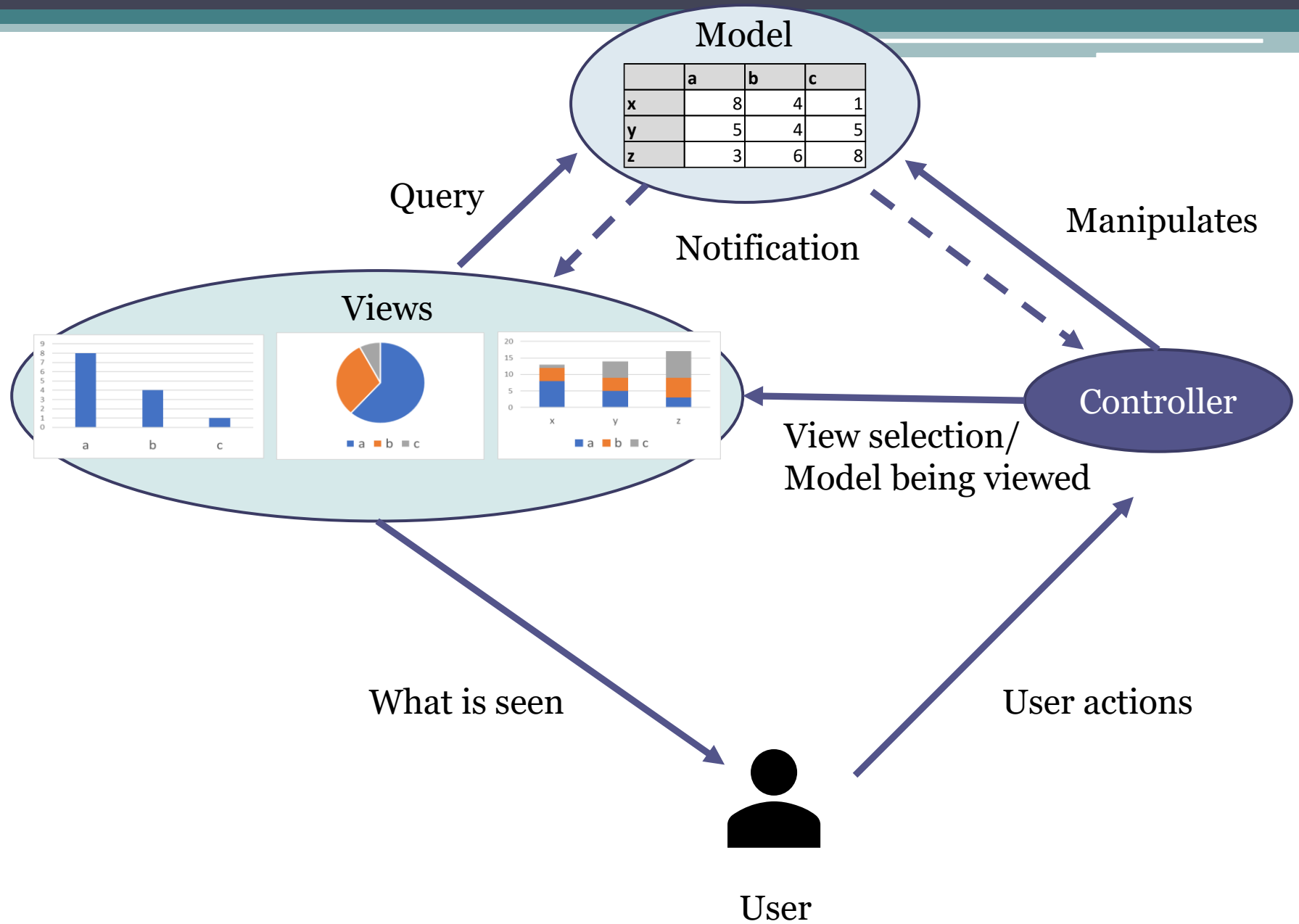
Design Patterns

Model View Controller Design Pattern

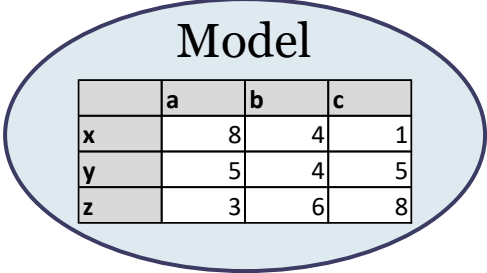
Dr. Williams
Central Connecticut State University

Design pattern: Model View Controller (aka MVC)

- Behavioral pattern
- Motivation:
 - Often it makes sense to present multiple perspectives or views of the same model or even have the same view behave differently depending on the context
 - Goal of MVC is to separate these three aspects: view of data, model of data, and control of interaction of view and model to increase reusability of each of these areas
- Example:
 - Same business logic applies whether I am placing an order through a web site or a mobile application
 - Same view of data applies whether employee or manager, but actions available differ



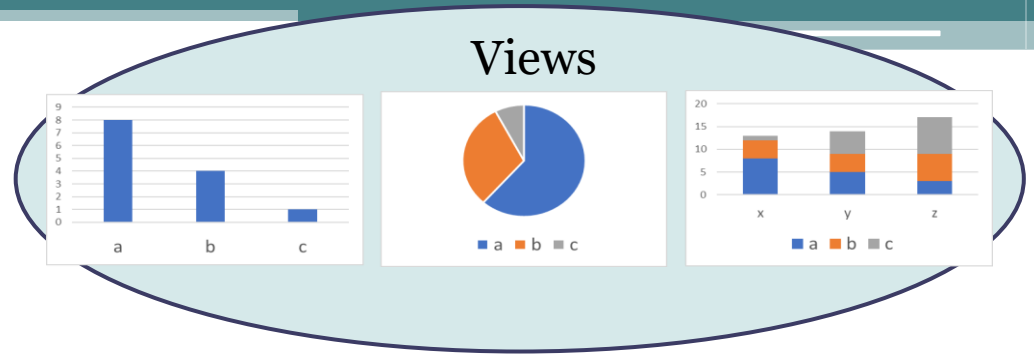
Model



	a	b	c
x	8	4	1
y	5	4	5
z	3	6	8

- Your underlying data
- Manages app data and state
- Not concerned with UI or presentation
- Often persists somewhere
 - Database or network store
- Same model should be reusable, unchanged in different interfaces

View



- What the user sees
- Present the model to the user
 - Desktop view/mobile view
 - Bar chart, pie chart, table
- Provides interface for user to manipulate data
- Does not store any data
- Easily reusable and configurable to display different data

Controller

Controller

- Glue that makes app unique
- Defines how to respond to events
- Intermediary between Model & View
- Updates the model when the user manipulates the view
- If model changes dictate a change in View, selects next view to present

Controller cont.

- Typically where the **app** logic lives
- **Not** the business logic
- In MVC this sometimes requires thought
 - Model makes business decisions
 - Controller what to display (which view) based on decisions
 - Sometimes this gets a little cloudy as we will see later
 - View how to display that perspective

Why use an MVC model

- Just like you can have “spaghetti” back-end programs you can have “spaghetti” UI apps (yuck!!)
 - Clear responsibilities make things easier to maintain
 - Avoid having one monster class that does everything
- Separating responsibilities leads to reusability
 - By minimizing dependencies, you can take model or view class already written and use it elsewhere
 - Think of ways to write less code

Communication and MVC

- How should objects communicate?
- Which objects know about one another?
- **Model**
 - Example: Customer class
- Not aware of views or controllers
 - Note this does not mean methods aren't created for use by views in the generic sense
- Typically the most reusable
- Either passive waiting to be queried or communicate generically using
 - Notifications

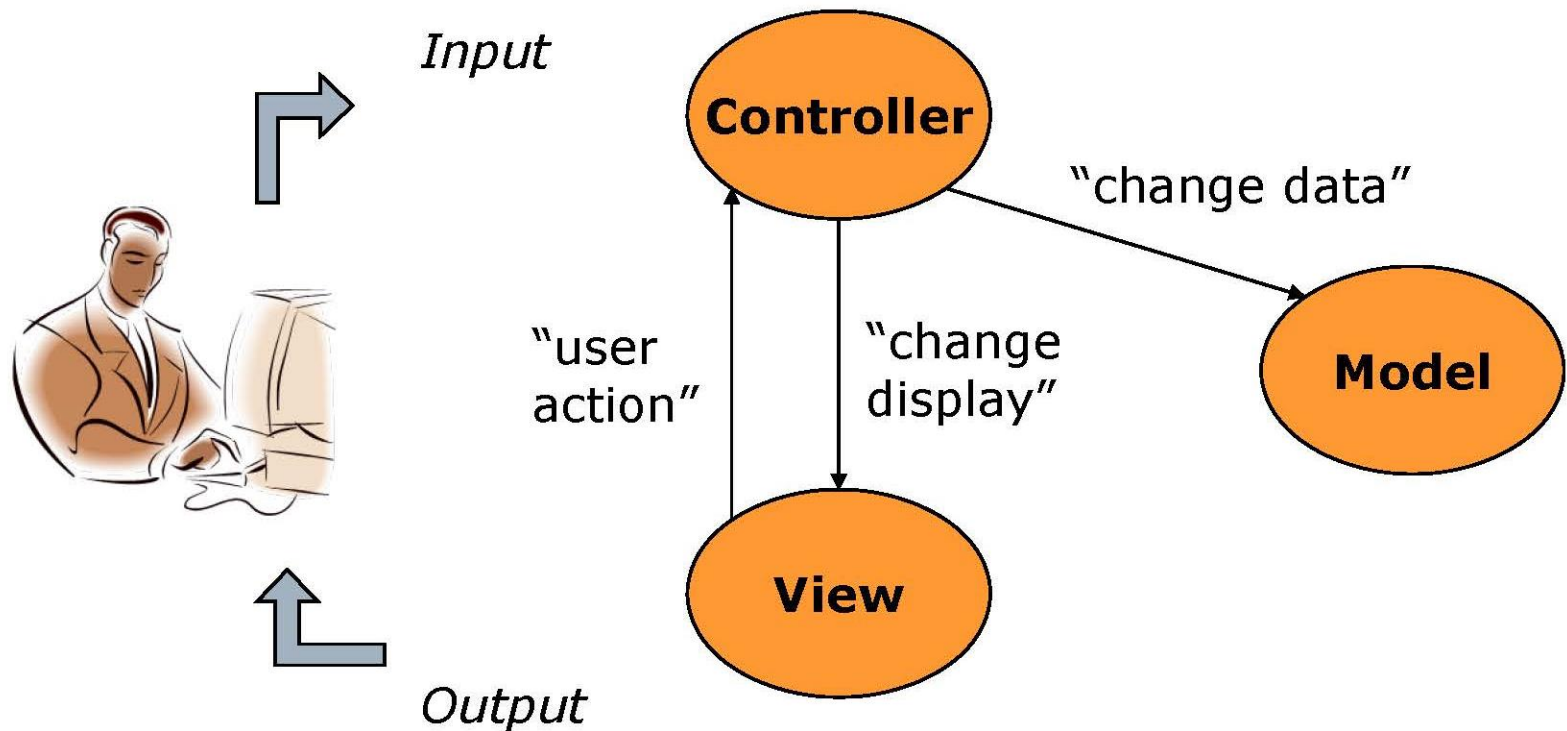
Communication and MVC

- How should objects communicate?
- Which objects know about one another?
- **View**
 - Example: Table view of customers
- Not aware of controller flow/controller context
- Also **tends to be reusable**
- Communicate with controller using
 - Target-action (i.e. click on UI passed appropriately to Controller for action)
 - Delegation

Communication and MVC

- How should objects communicate?
- Which objects know about one another?
- **Controller**
 - Knows about model and view objects
 - Knows and **manages application context** (brains of the application)
 - Manages relationships and data flow
 - Typically app-specific
 - **Rarely reusable**

Basic interactions in MVC



Mechanics of Basic MVC

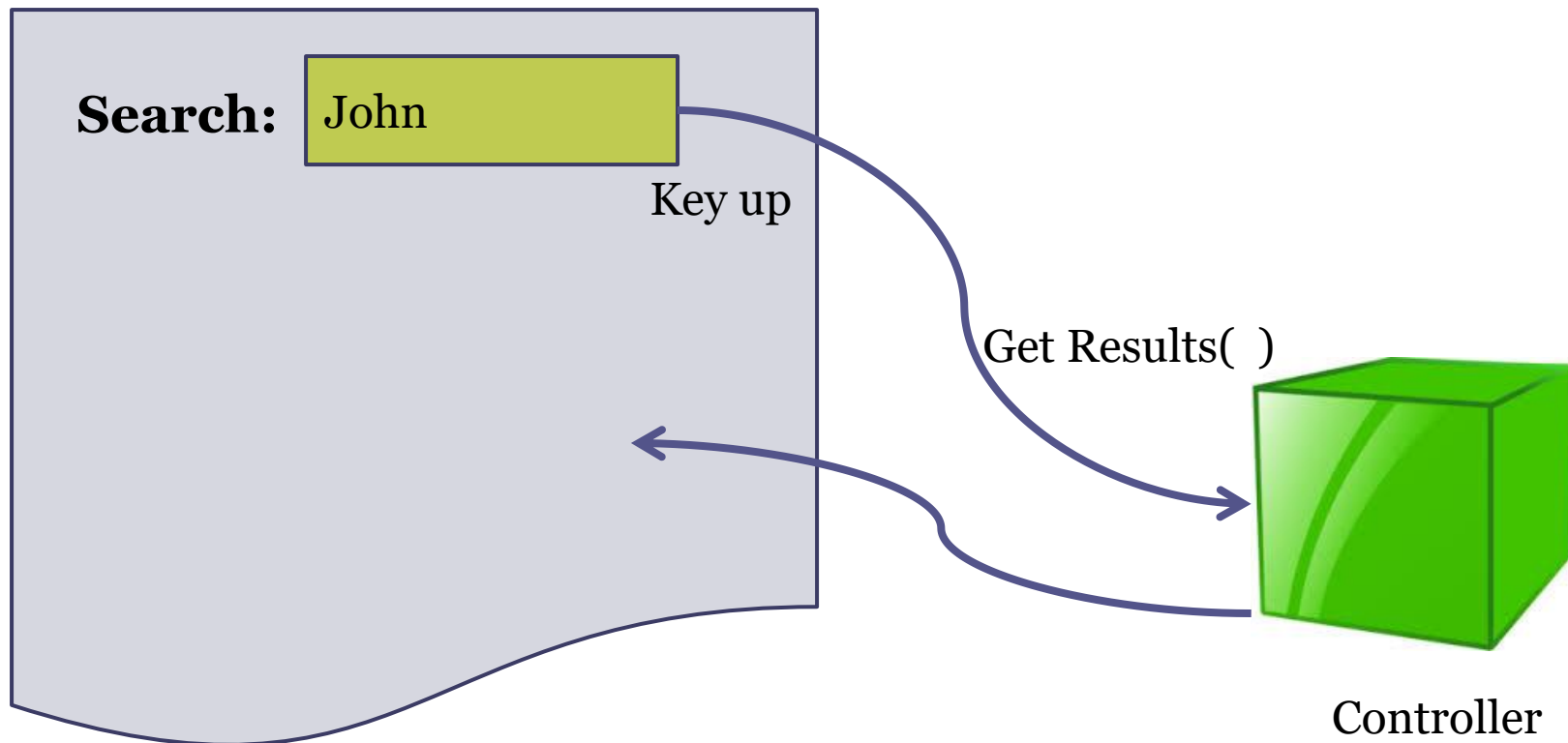
- Setup
 - Instantiate model
 - Instantiate view
- Execution
 - View recognizes event
 - View calls appropriate method on controller
 - Controller accesses model, possibly updating it
 - If model has been changed, view is updated (via the controller or via generic notification from model and view updating details of display)

View Controls - Events

- View objects that allows user to initiate some type of action
- Respond to variety of events
 - Value changed
 - Editing
 - Mouse over
 - Clicks

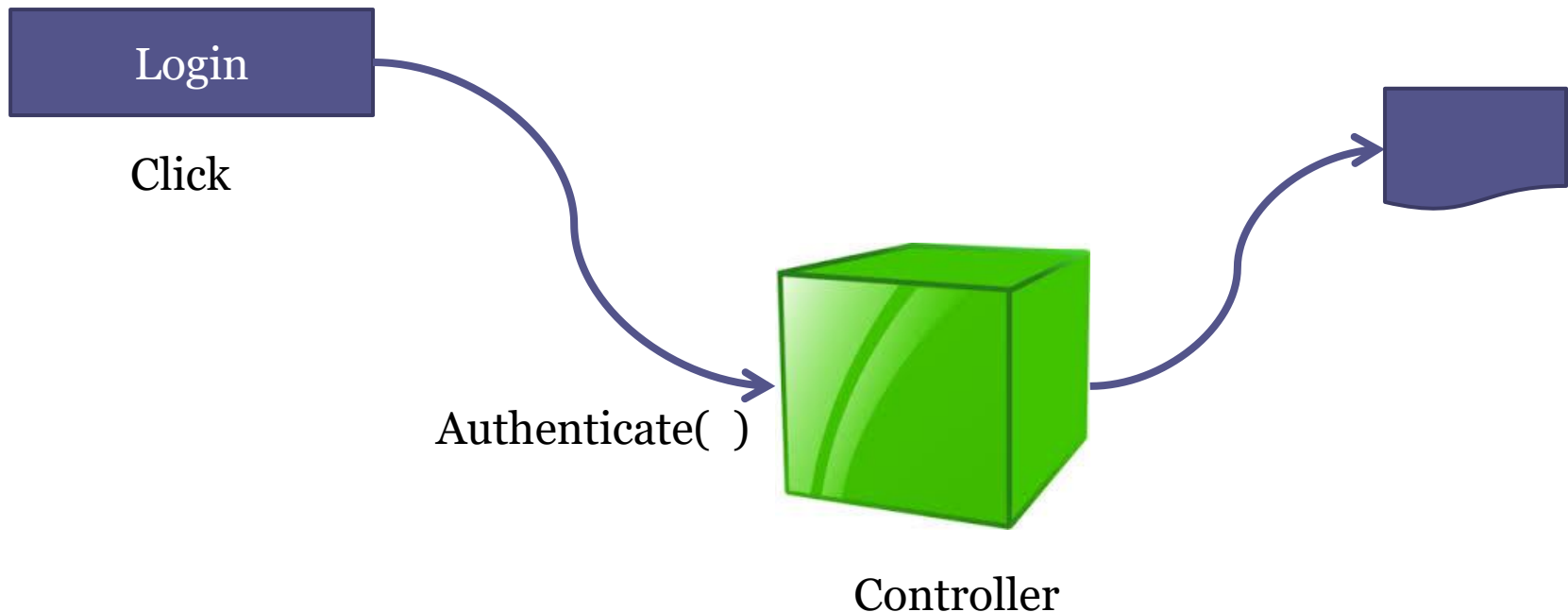
Controls - Delegation

- When an event occurs, controller called to return and/or update content



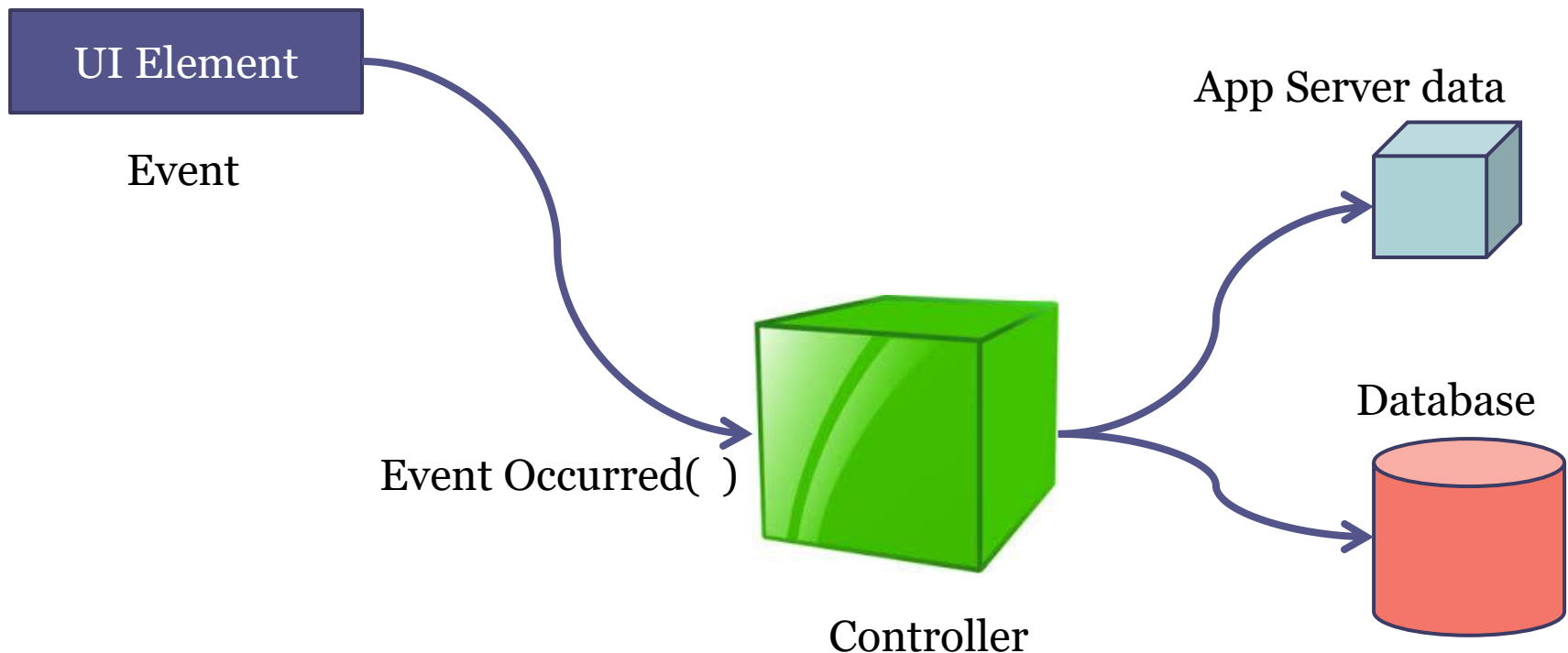
Controls - Target/Action

- When an event occurs, an action is invoked on the controller to decide based on context what view to display next

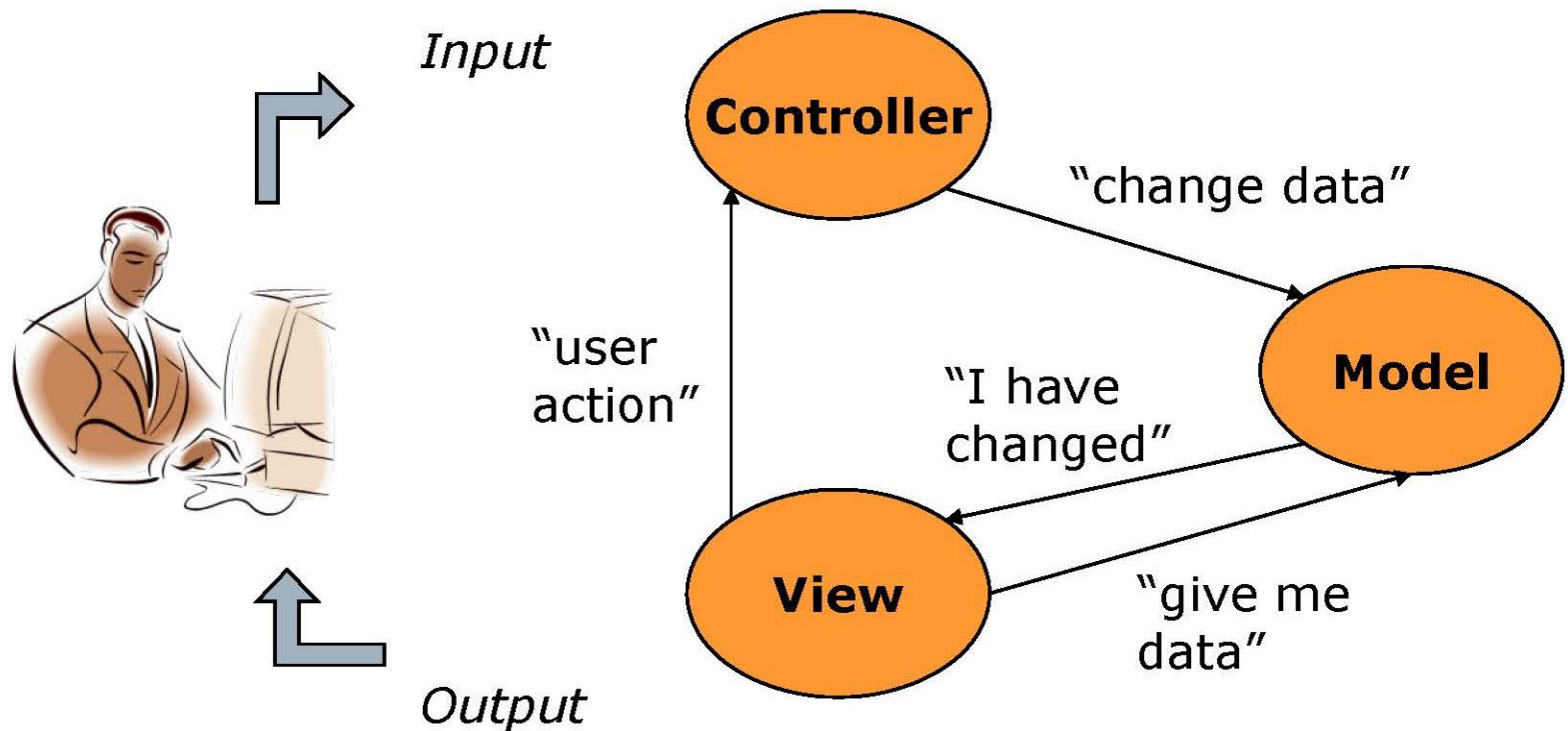


Controls - Persisting

- When an event occurs, controller responsible for persisting if necessary



Extended Interactions in MVC



Running example - car sales application

- Basics
 - Enter car information
 - Make
 - Owner information
 - Mileage
 - Based on information determine application flow
 - Context driven
 - Business driven

MVC - Controller

- Controller is responsible for the **tracking context, flow of the application** and **persistence**
 - Designed with three primary goals
 - Separate flow from the UI View itself
 - Integrate request context with model (glue between the two)
 - Simplify controller development for common cases (KISS)

Business logic

Business Logic - *encodes the real-world business rules that determine how data can be created, displayed, stored, and changed*

- *Business logic rules do not belong on controller*

Car sale example

- Example rule
 - The rules on selling a car vary based on state, in order to be sold
 - Cars sold in NY and CA must have additional forms completed for joint ownership
 - Cars sold in CA must have a special emissions check completed
 - *In actual business case there are over 50 such variations based on state*
 - **This is business logic**

Business logic and simple navigation

- Rather than having controller determine...
- Have model determine if business rules are met
- Have control decide flow if they are
- Logically
 - On submit
 - Is model valid? (`car.valid()`)
 - YES – Go to next page
 - NO – Go someplace else

Car sale example...

- State rules – business – let model decide
- Note – the information that needs to be considered for the business rules is business rules
 - Example options on a particular form vary by state
 - Rather than have a different UI View for each state have conditional elements controlled by model
- Example – Model knows which states matter

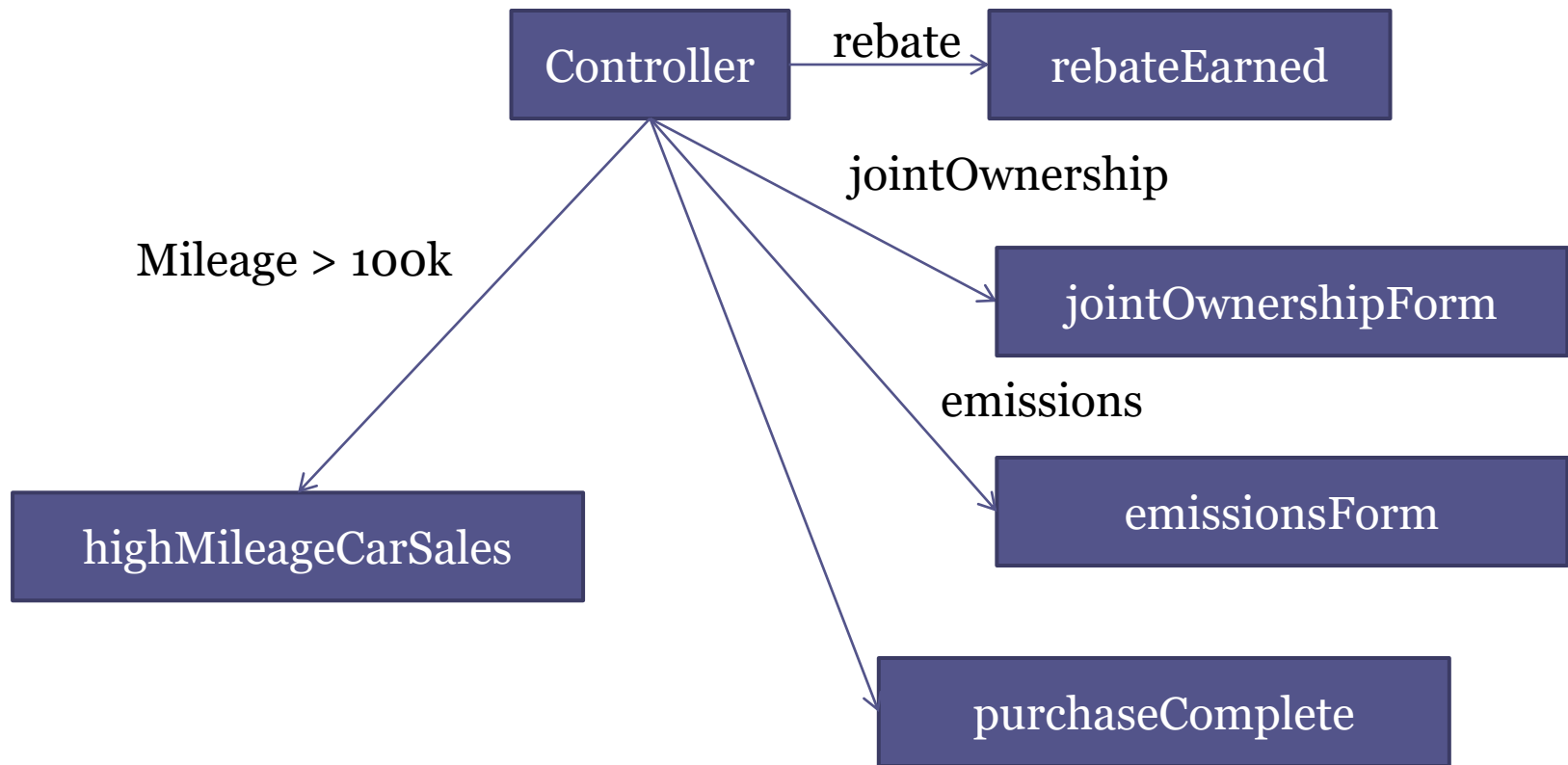
Car sale example -rules

- Enhancing flow
 - Logic results in more than yes/no
 - Controller direct flow based on model
 - Additional context criteria
- Additional rule
 - If car has over 100k miles send to another reseller
 - Collection of useful information
 - Questionable – Not intrinsic to business logic, conditional flow may be good solution

Car sale example

- Complex control
 - Flow dictated by view context
 - Flow dictated by model context
 - **Persistence**

Navigation -complex flow driven by controller business logic in model car sales



Design MVC

- Bank is developing online mortgage application. Basic flow collect personal data, financial data, and home data then display approval/denial. 3 additional possible financial screens dependent on loan status. For home info screen some options state specific
 - Identify potential model
 - Identify psuedo methods on model
 - Identify views
 - Identify controllers
 - How implemented
- Describe points of interaction