UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

SENG365 Web Computing Architecture:
Week 7
Single page applications, Intro to Vue.js, Design patterns

Ben Adams
Course Coordinator
benjamin.adams@canterbury.ac.nz
310, Erskine Building

Main client-side technologies

The move toward Single Page Applications

- Users want responsiveness & interactivity; improved user experience
  - Compare user experience with native apps & stand alone apps

- Managing the interactions with a user is *much* more complex than managing communication with server
  - Think of all those events (e.g. `onClicks`) to handle

Single Page Applications (SPAs)

“Single page apps are distinguished by their ability to redraw any part of the UI without requiring a server round trip to retrieve HTML. This is achieved by separating the data from the presentation of data by having a model layer that handles data and a view layer that reads from the models.”

http://singlepageappbook.com/goal.html

Some features of Single Page Applications

Separate:

- **Data**
- '**Content**' & Presentation: HTML & CSS (and other resources)

Reduce communication with the server/s by:

- **Occasional download** of resources e.g. HTML, CSS and JavaScript
- Asynchronous 'background' fetching of data: **AJAX / XHR or web sockets**
- **Fetch data only** e.g. JSON
- Fetch data from different servers: **CORS**

'Page' navigation

- Client-side JavaScript handles the **routing** instead of the browser itself
  - Managing **page history**
- Routing within the SPA

Reconceive web application

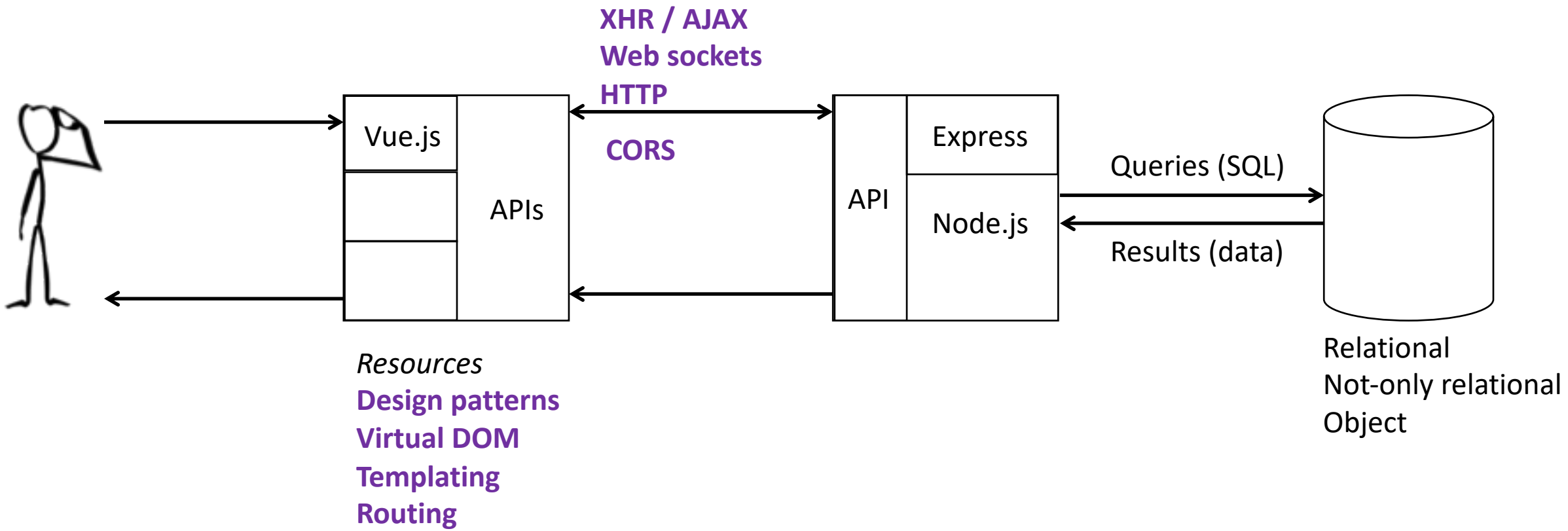
Re-balance workload across:

- Server-side application
  - e.g. Node.js
- Client-side application/
  - Front-end libraries and frameworks e.g. vue.js
- Communication between client & server
  - API-driven/specified
  - e.g. AJAX & JSON

Need to consider (for example):

- Manage application assets / resources
  - e.g. dependency management
- Application design and implementation
  - Modularisation
  - Design patterns
- **Templating**

The balance of work between client and server changes. And there are new technologies!



Vue.js

What is Vue.js?

- A progressive framework for building user interfaces;
- Designed to be incrementally adoptable; and
- Capable of powering sophisticated Single-Page Applications
  - when used in combination with modern tooling and supporting libraries.
- A (powerful) layer of abstraction for JavaScript

A simple example…

Why does this code seem odd, for HTML & JS?

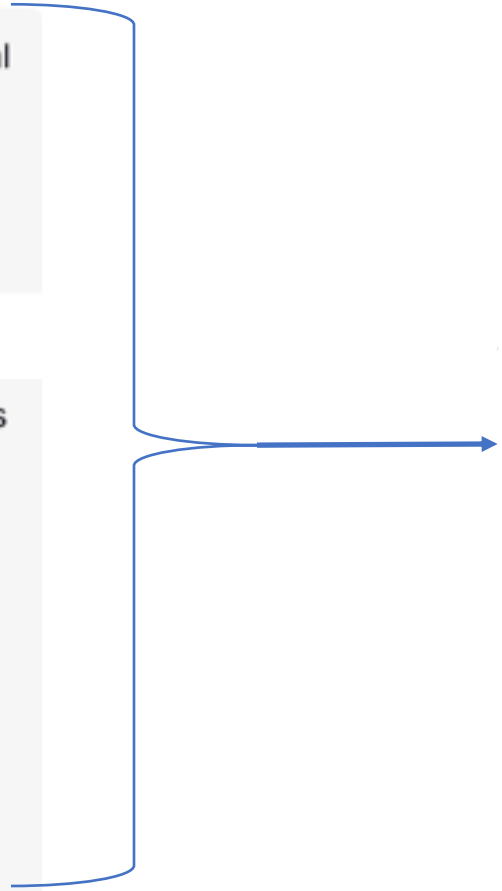
html

```html
<div id="counter">
  Counter: {{ counter }}
</div>
```

js

```js
const Counter = {
  data() {
    return {
      counter: 0
    }
  }
}

Vue.createApp(Counter).mount('#counter')
```



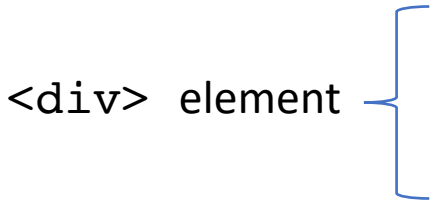
Rendered content

Counter: 0

https://v3.vuejs.org/guide/introduction.html

attribute id

attribute value




```html
<div id="counter">
  Counter: {{ counter }}
</div>
```

Declare HTML div element with content {{ message }} and with an id of “app”

templating
syntax: template for content

```js
const Counter = {
  data() {
    return {
      counter: 0
    }
  }
}

Vue.createApp(Counter).mount('#counter')
```

Create a new Vue object. The Vue object:

- Is mounted to an element with value ‘#counter’. This property ‘binds’ this Vue instance to the element with id of ‘counter’.
- A property called data() that returns a POJO with a programmer-defined name-value pair: name of counter, and a value of 0

Data binding

- Vue allows you to link data and the Document Object Model, a.k.a. DOM (via the virtual DOM)

- Once linked, things are then reactive
  - User changes the DOM (e.g. enters information), the data is updated;
  - JavaScript changes the data, the DOM is updated.

- You, as a programmer, don’t need to worry about:
  - Getters
  - Setters

Vue.js and HTML custom attributes

- Vue.js declares custom attributes, `v-*`, for example:
  - `v-bind`
  - `v-if`
  - `v-for`
  - `v-on:click`
- Custom attributes are known as *directives*
- A *directive* provides special reactive behaviour

```html
<div id="event-handling">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">Reverse Message</button>
</div>
```

```js
const EventHandling = {
  data() {
    return {
      message: 'Hello Vue.js!'
    }
  },
  methods: {
    reverseMessage() {
      this.message = this.message
        .split('')
        .reverse()
        .join('')
    }
  }
}

Vue.createApp(EventHandling).mount('#event-handling')
```

```html
<div id="event-handling">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">Reverse Message</button>
</div>
```

Directives are used to create reactive views, which update as the model changes.

```js
const EventHandling = {
  data() {
    return {
      message: 'Hello Vue.js!'
    }
  },
  methods: {
    reverseMessage() {
      this.message = this.message
        .split('')
        .reverse()
        .join('')
    }
  }
}

Vue.createApp(EventHandling).mount('#event-handling')
```

```html
<div id="event-handling">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">Reverse Message</button>
</div>
```

```js
const EventHandling = {
  data() {
    return {
      message: 'Hello Vue.js!'
    }
  },
  methods: {
    reverseMessage() {
      this.message = this.message
        .split('')
        .reverse()
        .join('')
    }
  }
}

Vue.createApp(EventHandling).mount('#event-handling')
```

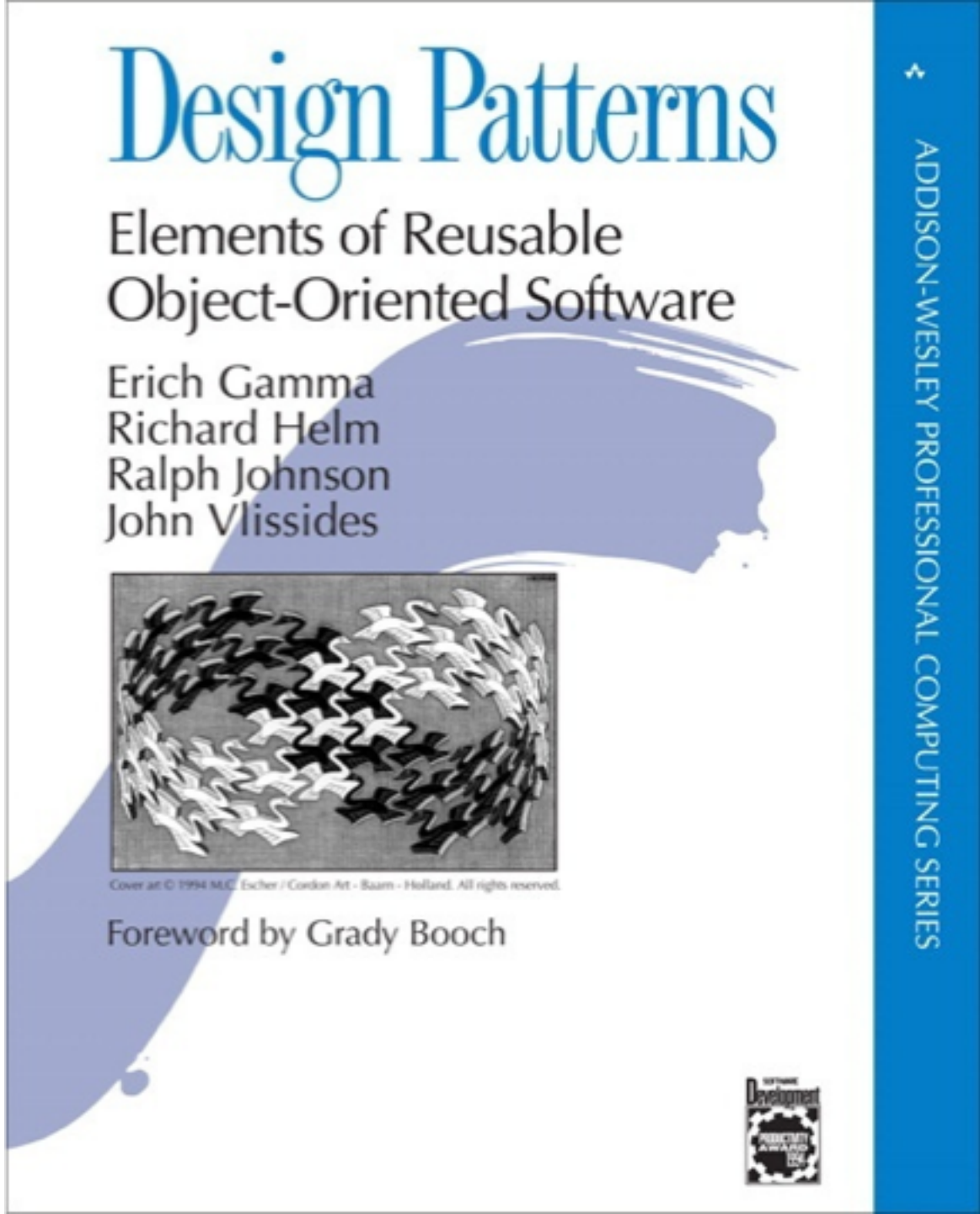
Use `this` to access the variables that make up your model.

Other features of Vue

- `computed`
  - A derived property
  - Has getters and (sometimes) setters
  - You don’t call a computed, you reference it
  - You don’t pass parameters to a computed
  - e.g. convert cents to dollars
- `method`
  - A traditional function/method of JavaScript
  - You must call it
  - You can pass parameters

- Vue lifecycle
- Watchers
- Components

Design patterns

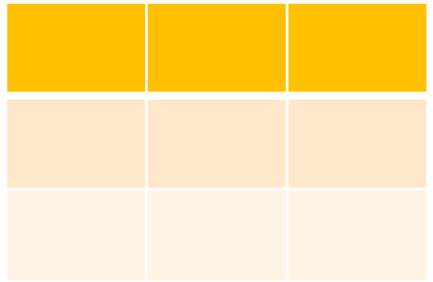


Background: *Separation of concerns*

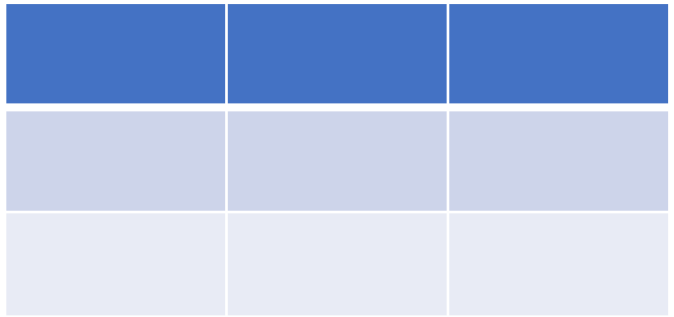
- A design principle for separating software code into distinct ‘sections’, such that each section addresses a separate concern.
- A strategy for handling complexity
  - Of the problem
  - Of the solution e.g. software code
- Examples of separation of concerns:
  - Object-oriented programming
    - Classes, objects, methods
  - Web computing
    - HTML: structure/organisation of content
    - CSS: presentation
    - JavaScript: functionality

Example: data and views of data

Visualisation of data



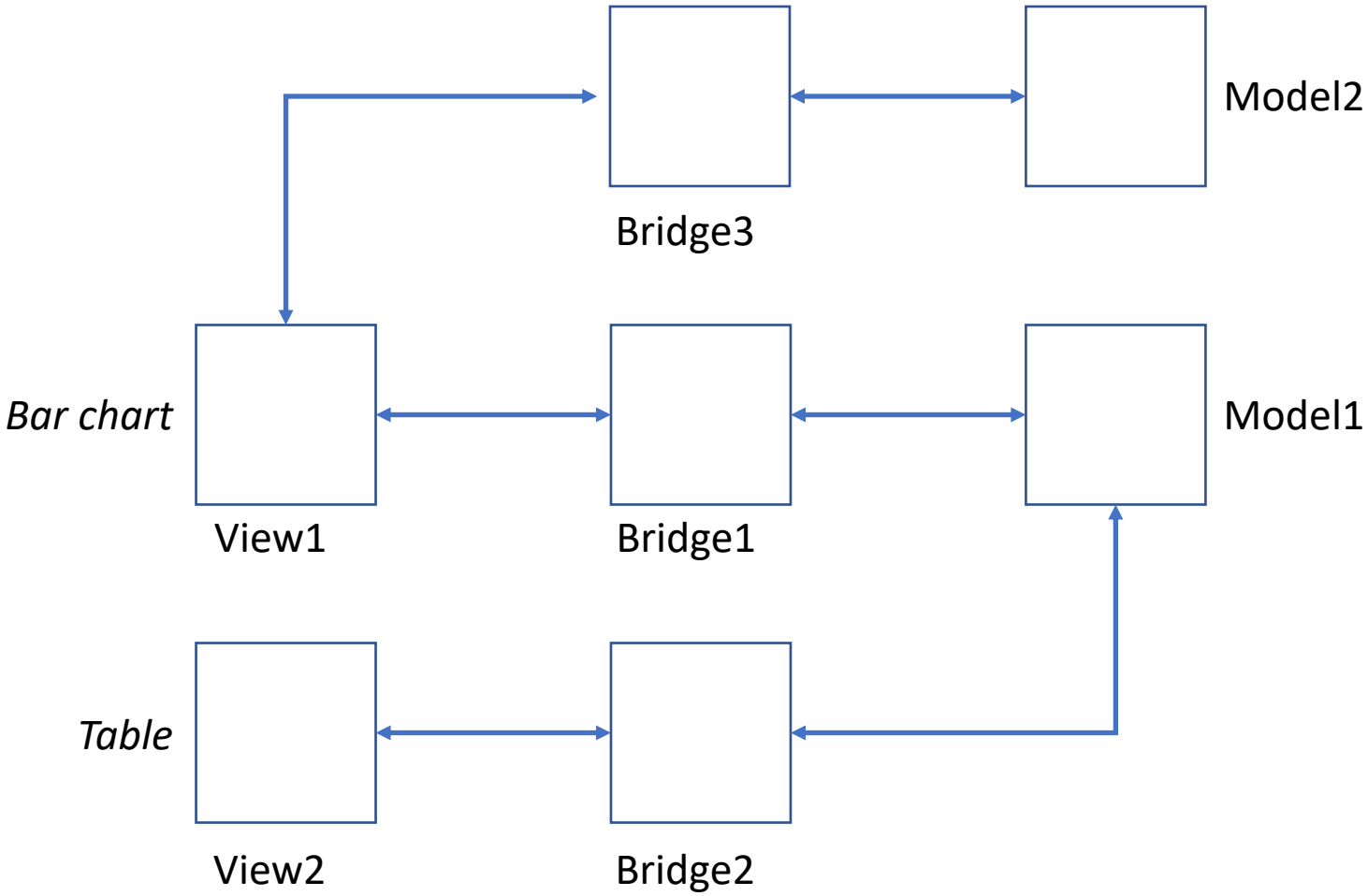
Data (models) e.g. array, object, NoSQL



Separation of concerns:

- **The management of the data e.g. CRUD**
- **The visualisation of the data**

A generic approach



- We want to display data to the user in different ways e.g. as a bar chart or as a table of data.
- We want to keep our data independent of the way it is presented.
- We may want to compute additional values depending on how and why we present data e.g. add a total in our table.
- We need some connection – some ‘bridge’ – between views and data.
- We can *reuse*:
  - models and
  - views.

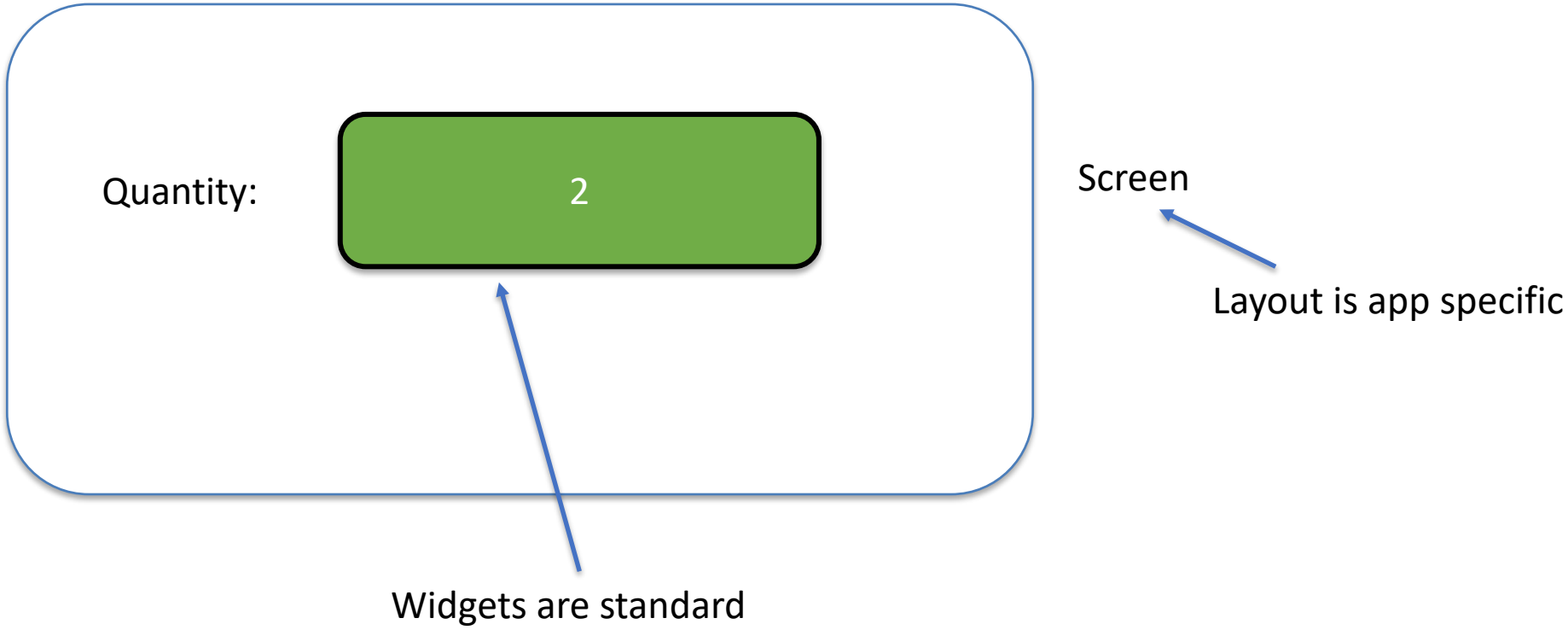
Variations on a theme of Model-View-{Bridge}*

MV…uh.. whatever?

- Model View Controller (MVC)
- Model View Adaptor (MVA)
- Model View Presenter (MVP)
- Model View ViewModel (MVVM)

Commentary

- (Some) disagreements on what *exactly* are these design patterns

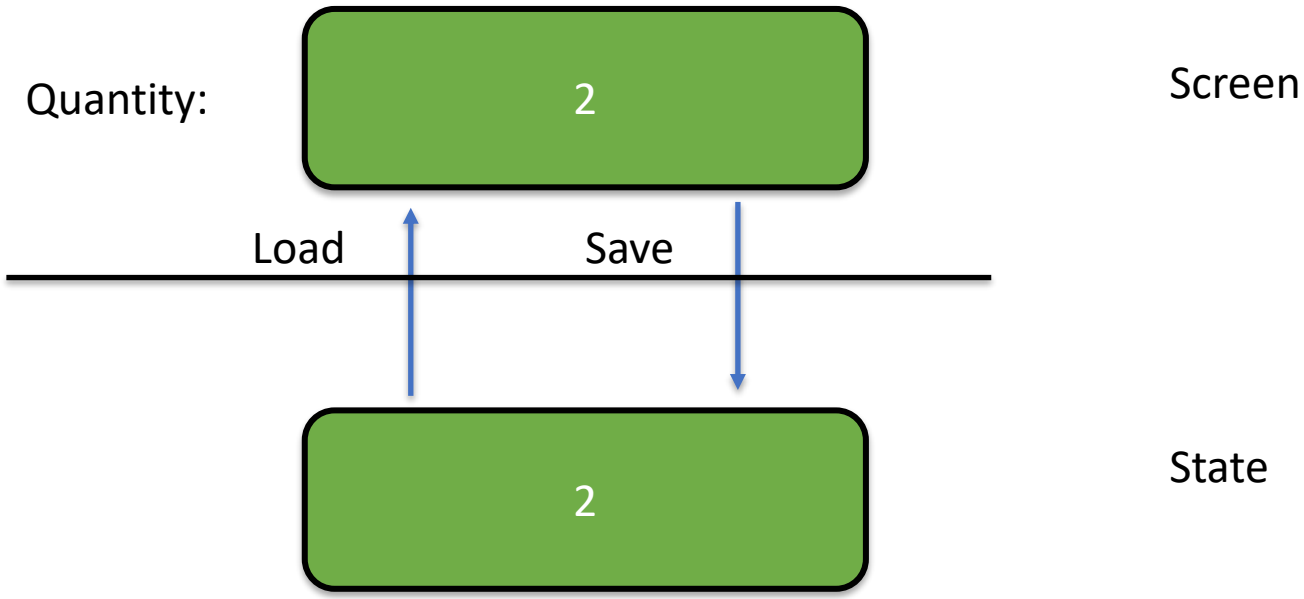


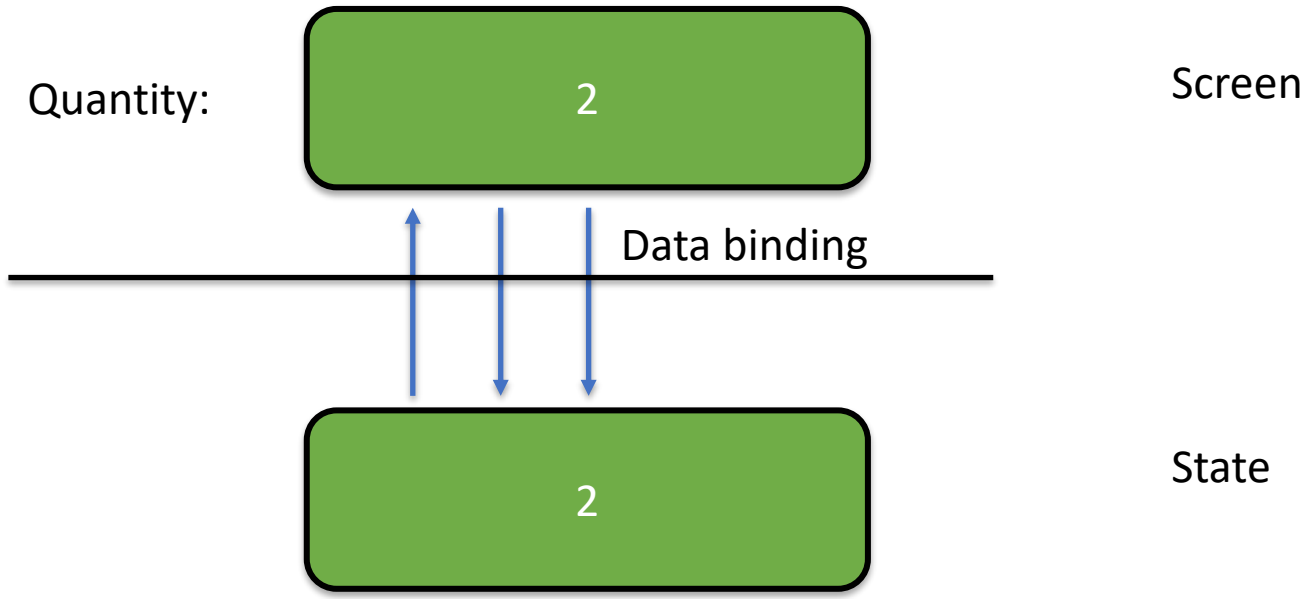
http://martinfowler.com/eaaDev/uiArchs.html

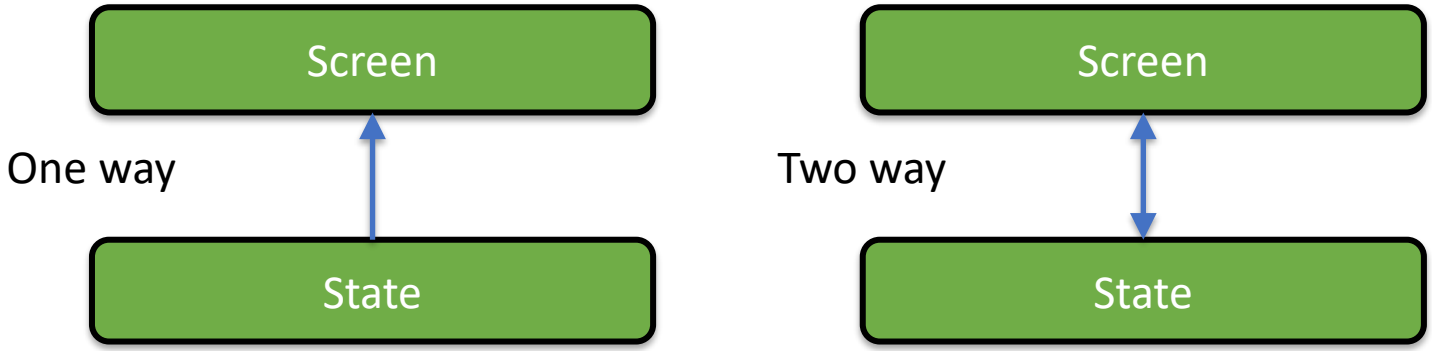
Quantity:

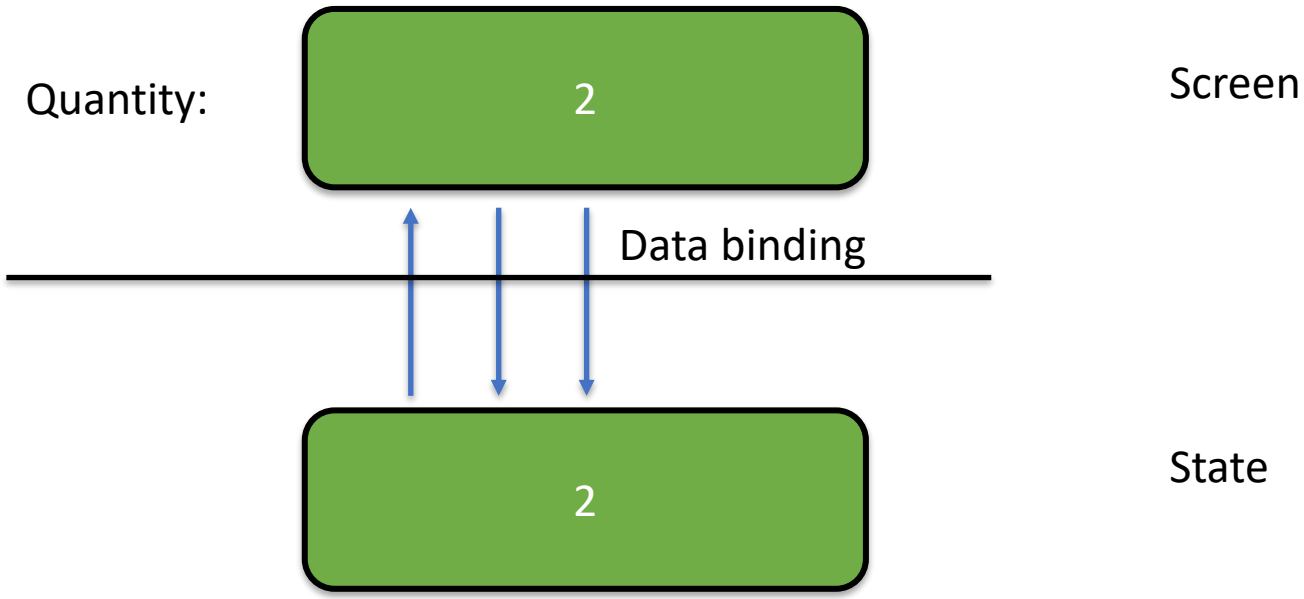
Screen

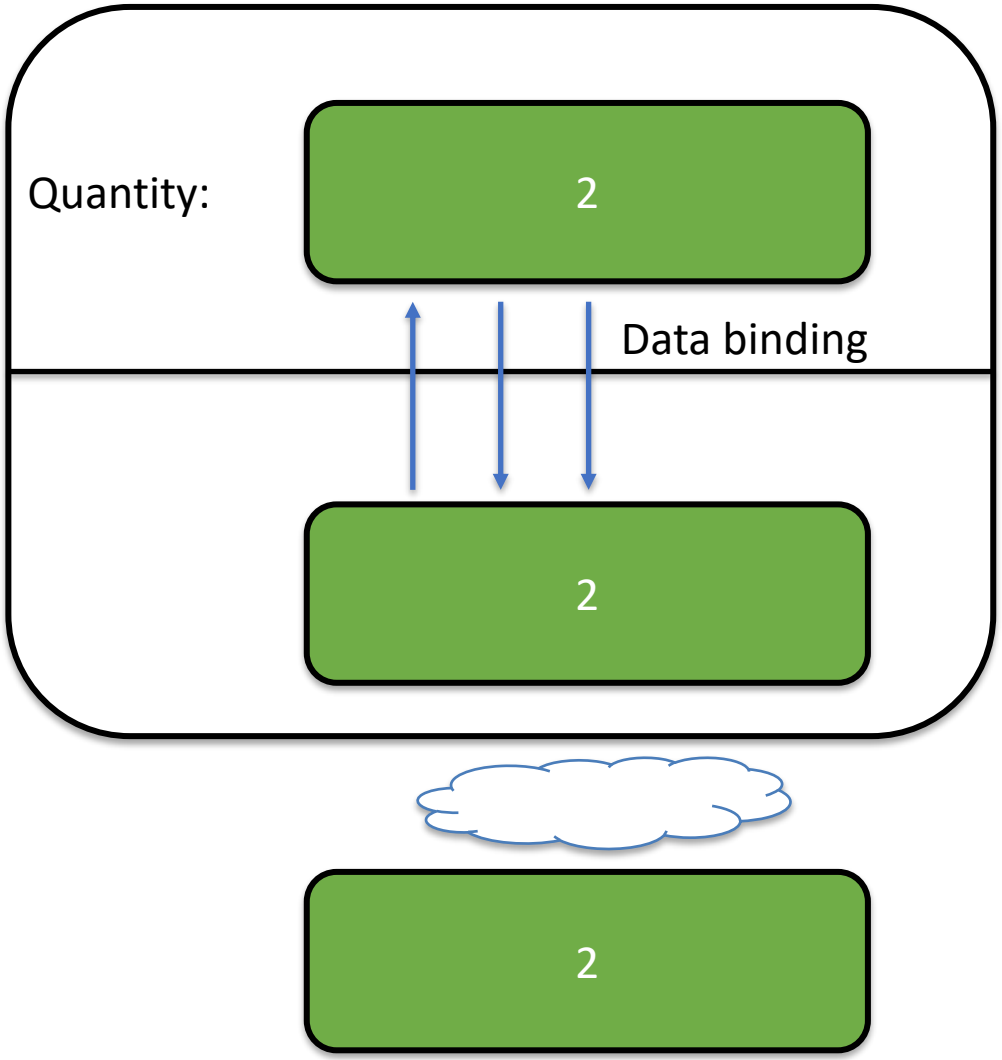
State


Quantity:
2
Screen
Load
Save
2
State


Quantity:
2
Screen
Data binding
2
State


Screen
One way
State
Screen
Two way
State

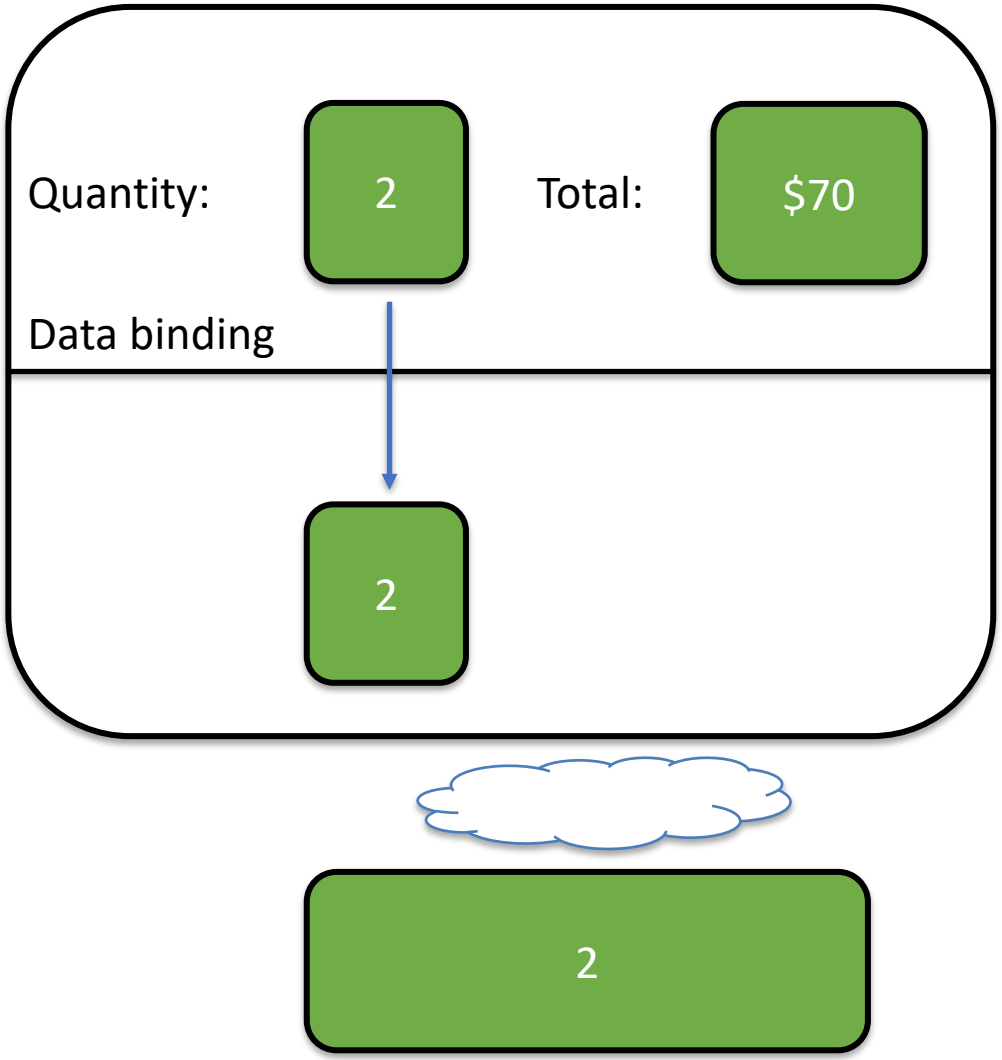

Quantity:
2
Screen
Data binding
2
State

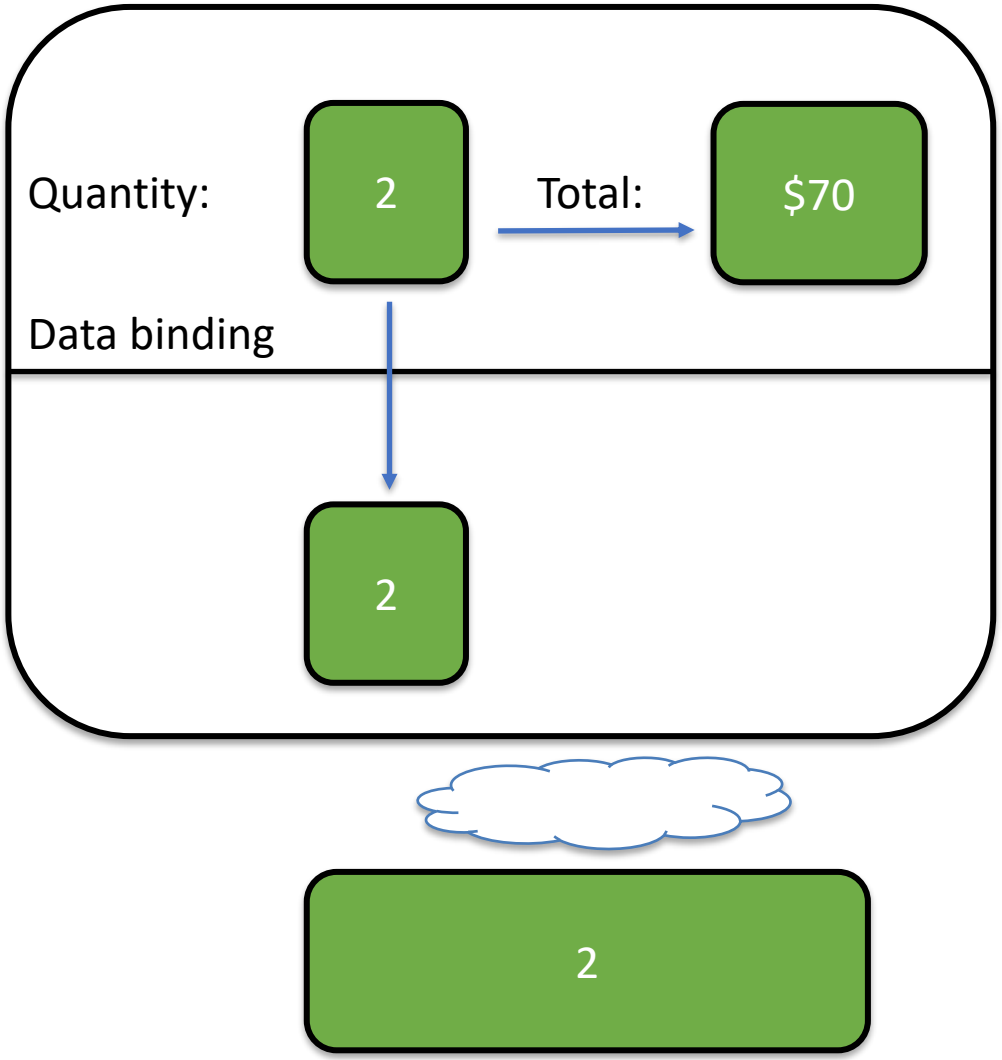


Screen

Session state

Record state

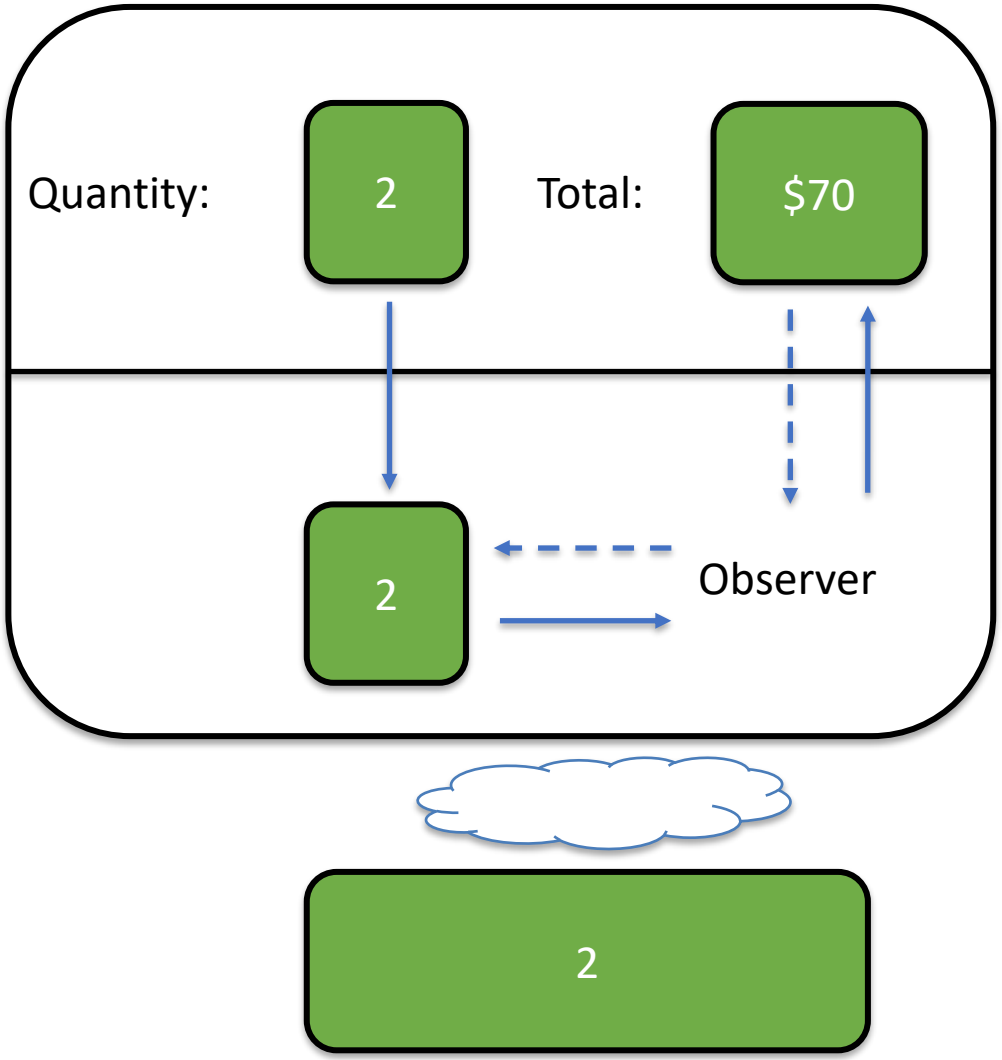

Quantity:
2
Total:
$70
Data binding
2
2
Screen
Session state
Record state



Screen

Session state

Record state



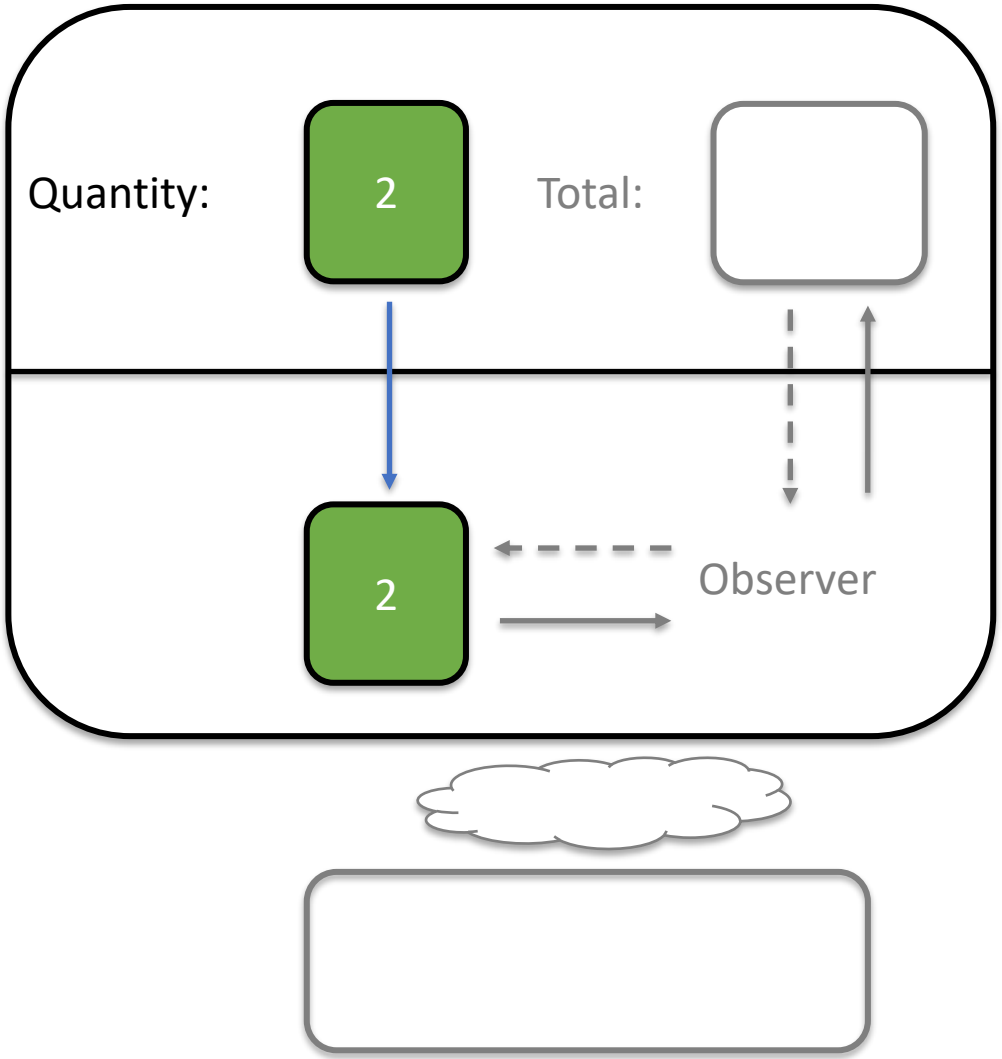
Screen = Presentation

Specific to the UI
Independent of Domain

Session state = Model

Independent of UI
Specific to the Domain

Record state = Data



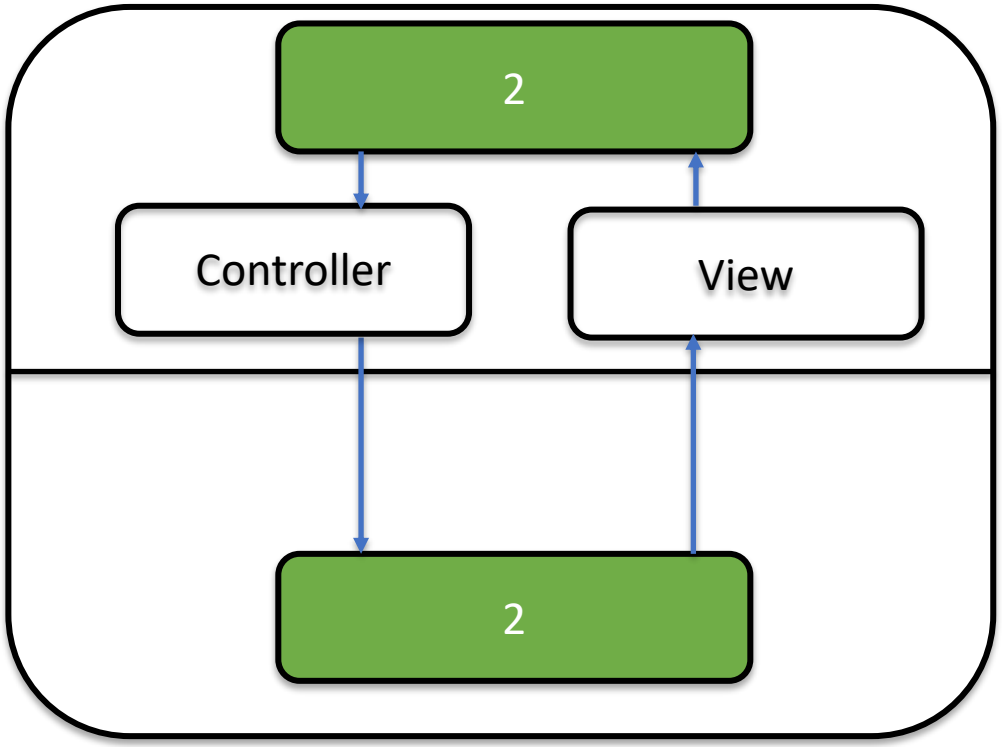
Screen = Presentation

Specific to the UI
Independent of Domain

Session state = Model

Independent of UI
Specific to the Domain

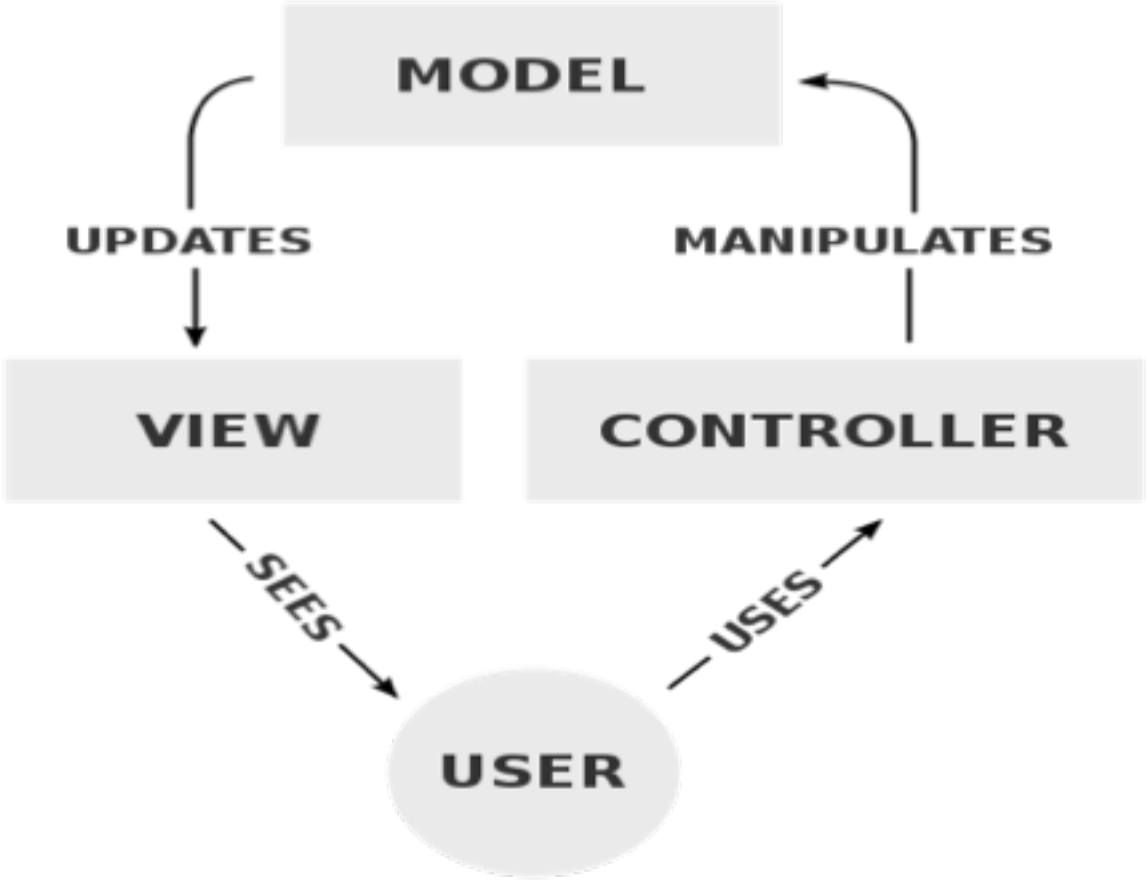
Record state = Data



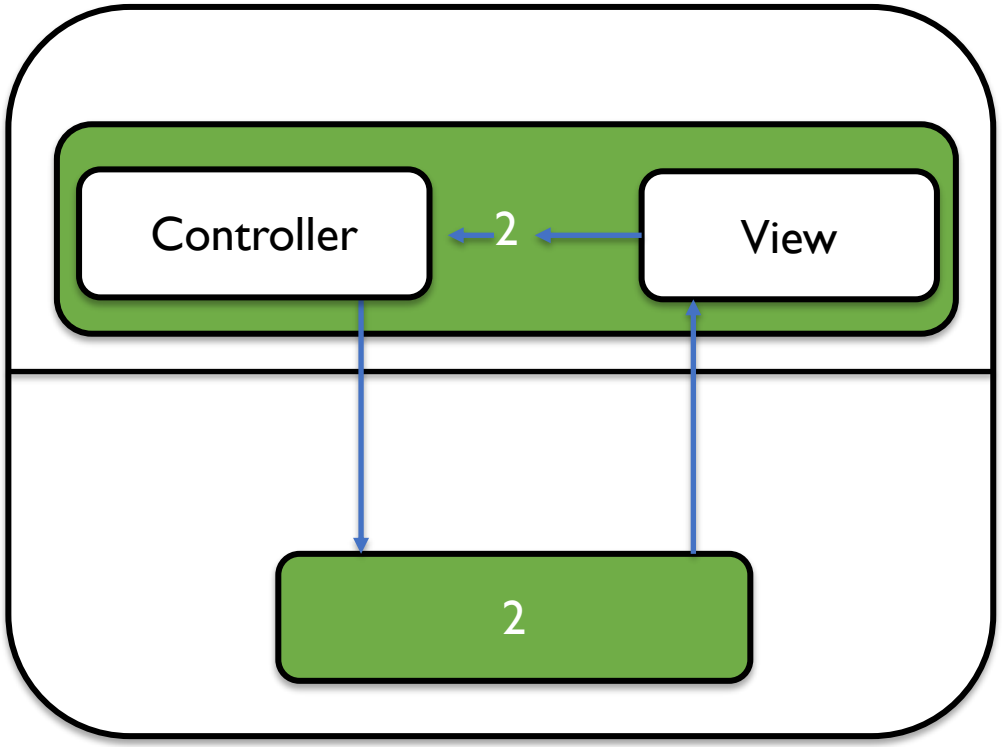
Model

MVC:

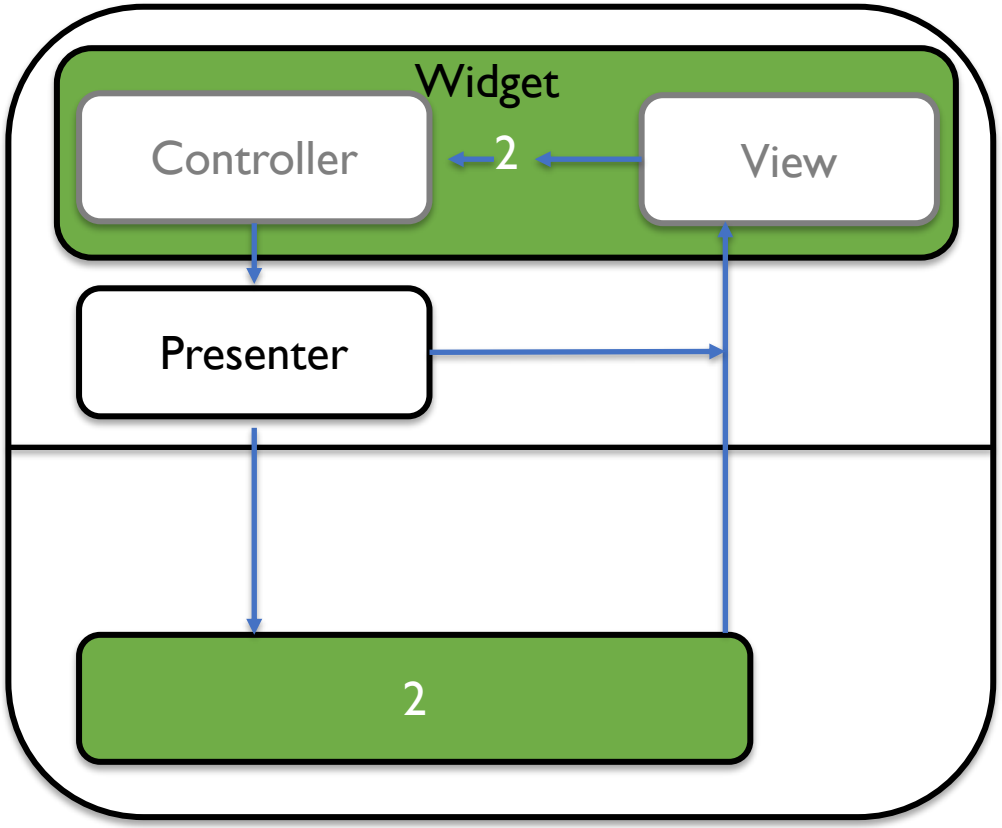
+ Multiple views
+ Synchronized views
+ Pluggable views and controllers
+ Exchangeable look and feel



- Complexity
- Can have lots of updates/notifications
- Links between controller and view
- Coupling of controller/view and model
- Mix of platform-dependent/independent code within controller and view (porting)

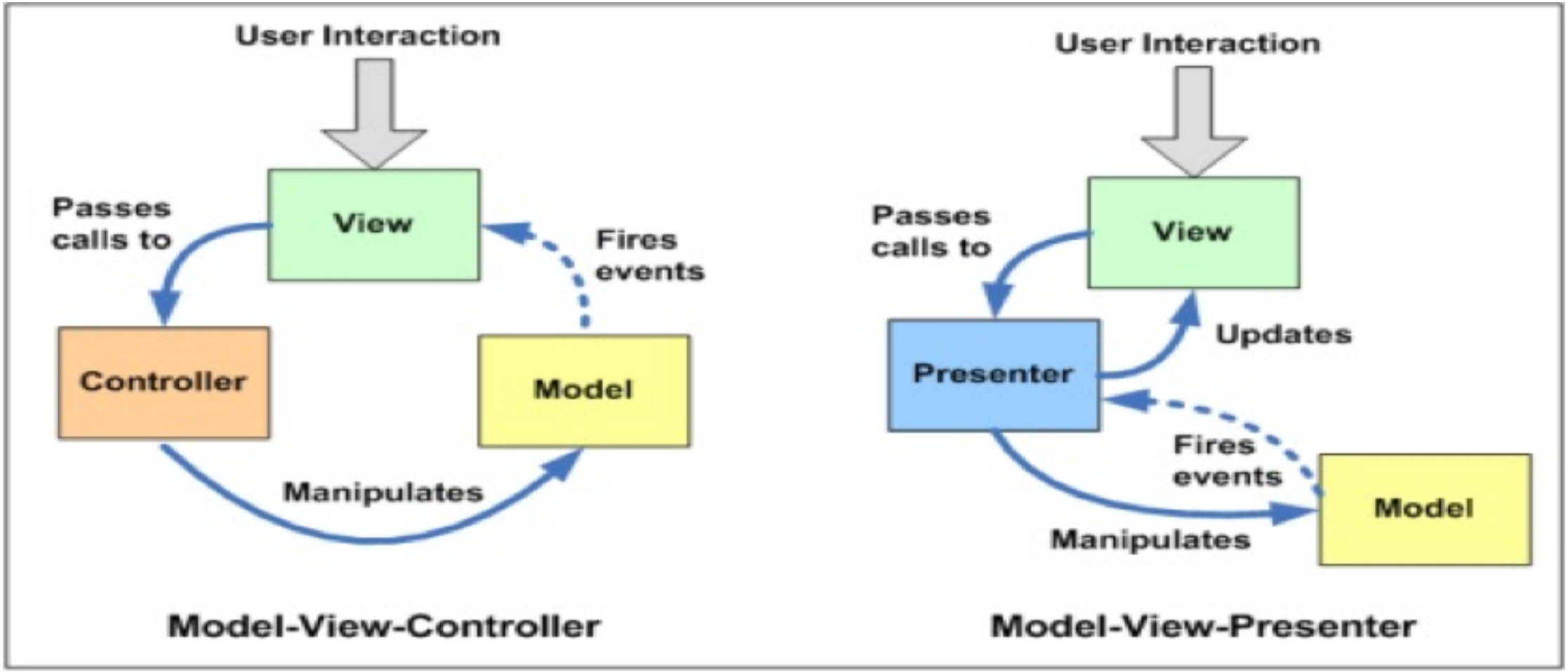


Model



Model

MVC & MVP



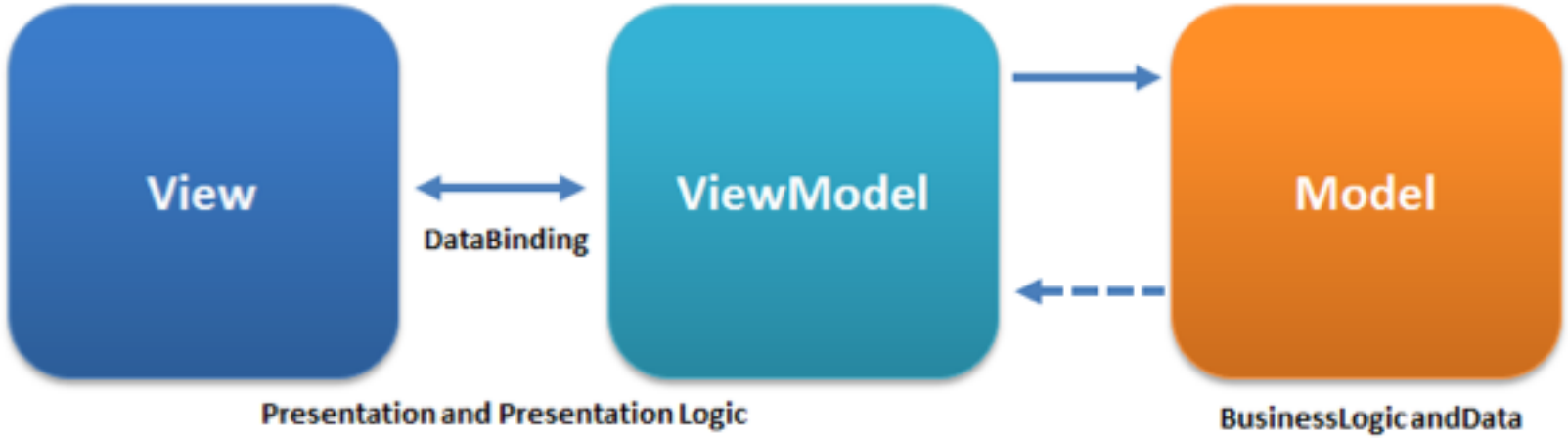
http://joel.inpointform.net/software-development/mvvm-vs-mvp-vs-mvc-the-differences-explained/

MVC versus MVP

- **MVC** is Model-View-Controller
- **MVP** is Model-View-Presenter
- Much hot air expended in defining / comparing / contrasting these.
  - **MVC**: view is stateless with not much logic. Renders a representation of model(s) when called by controller or triggered by model. Gets data directly from model.
  - **MVP**: view can be completely isolated from model and rendering data from presenter, or can be a MVC view, or somewhere in between
- But many variants.
- Categorisation not very important; you mostly just use whatever tools the framework gives you.
- The term MV* may be safer to avoid arguments!

MVVM

Model/View/ViewModel



- ViewModel is just the data currently required by the view
  - In a web context, Model may be on server, View and ViewModel on client
- *Data binding* synchronises view and viewModel bidirectionally
  - Uses lower-level "hidden" mechanism

http://en.wikipedia.org/wiki/File:MVVMPattern.png

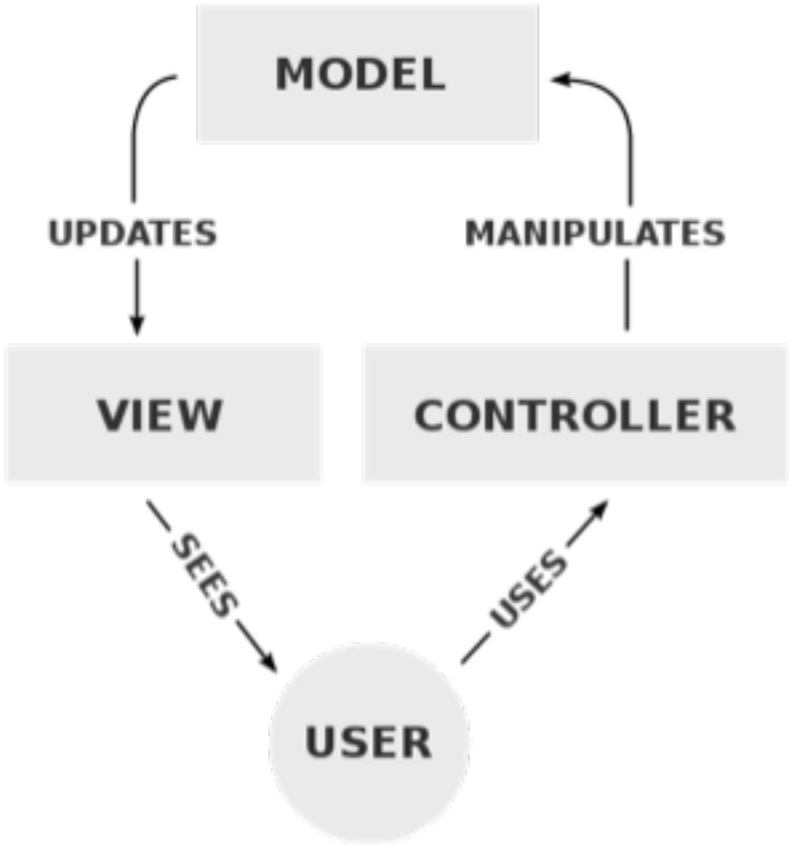
- Developing web applications is challenging
- One strategy to development, and to integration, is to ‘divide and conquer’
- Separate out concerns
  - 3-tier architecture of browser, server and data store

- Separate out concerns
  - MV* as a design concept
  - Differences of opinion on what the M, the V and the * were
  - Differences on how M, V and * interact with each other
  - Looked at MVC, MVP, MVVM and MVW

MVC: the case of the Wikipedia definition

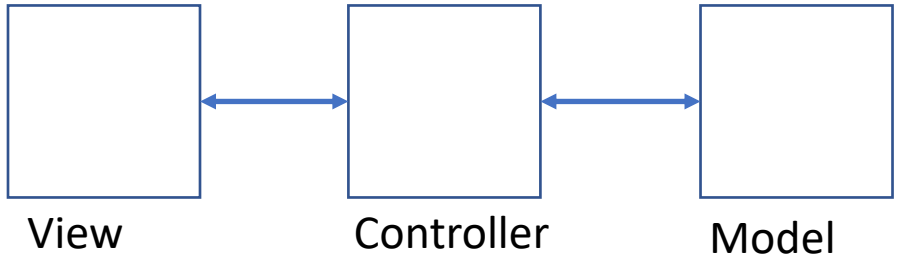
- **Model**
  - **Stores data** that is retrieved according to commands from the controller and displayed in the view.
- **View**
  - Generates new output to the user based on changes in the model.
- **Controller**
  - Send commands to the model to update the model's state (e.g., editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g., scrolling through a document, movement of document)

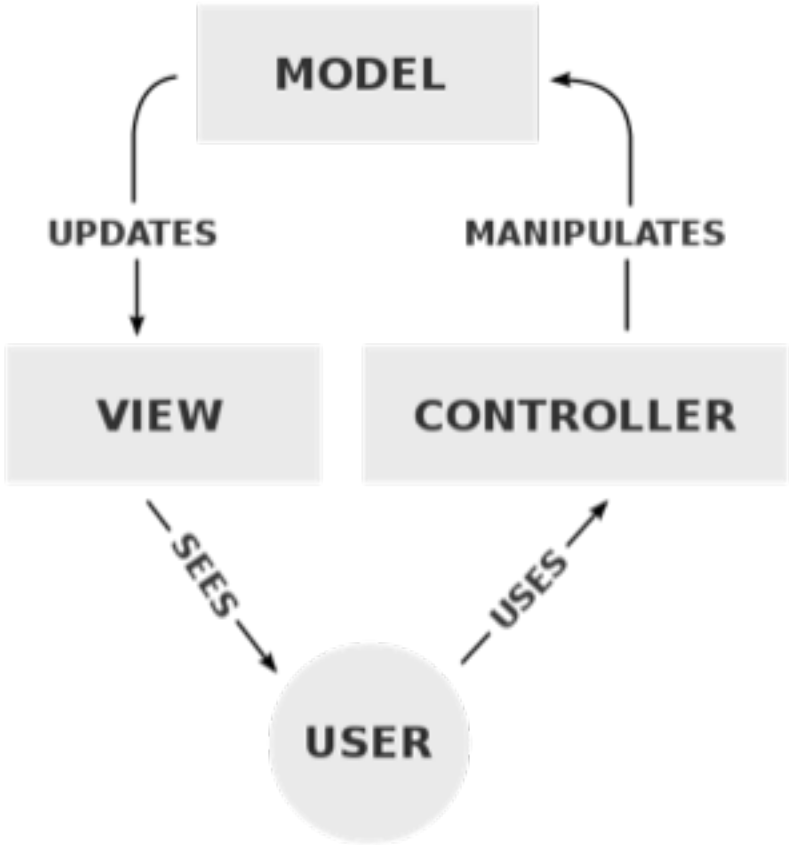
?



https://en.wikipedia.org/wiki/Model-view-controller

MVC: a comparison of models





https://en.wikipedia.org/wiki/Model-view-controller

Model View ViewModel (MVVM)

Components of Model View ViewModel (MVVM)

Model

- Either to a domain model, which represents real state content (an object-oriented approach), or to a data access layer, which represents content (a data-centric approach).

View

- The view is the structure, layout, and appearance of what a user sees on the screen (HTML & CSS?)

ViewModel

- An abstraction of the view exposing public properties and commands. Instead of the controller of the MVC pattern, or the presenter of the MVP pattern, MVVM has a binder.
- In the ViewModel, the binder mediates communication between the view and the data binder. The view model has been described as a state of the data in the model.

Binder

- Declarative data and command-binding are implicit in the MVVM pattern.

How does this apply to Vue.js?

“Technically, Vue.js is focused on the **ViewModel** layer of the MVVM pattern. It connects the **View** and the **Model** via two way data bindings. Actual DOM manipulations and output formatting are abstracted away into **Directives** and **Filters**.”

https://012.vuejs.org/guide/

Labs

Term 2 (6 weeks)

- Week 7: Lab 4
- Week 8: Lab 5
- Week 9: Lab 6
- Week 10 & 11: assignment support
- Week 12: **compulsory** lab
  - Assignment 2 testing

Assessment

The assessment

- Assignment 1 (25%)
- Mid-semester test (20%)
- Assignment 2 (25%)
  - Compulsory lab in final week
  - **No extension**
- Exam (30%; 2hrs)

Assignment 1

- We used a full test suite of 80 tests based on the API spec.
- Most people did really well!
  - Mean 77.1%, Median 88.7%
- Often people lost marks by simply missing small parts of the API spec, e.g.,
  - Not handling all image types jpeg, png, and gif
  - Missing key information in a request
  - *If you feel like the mark you got does not reflect what you were getting on the test server, then please email me.*
- If you do not have a mark on learn it means you code did not run on the test server (e.g., did not compile or did not connect to DB). You might get partial credit if you email me and come to my office to show me what you've done.
- If you want a full report of how your server did, then email me.

Mid-term test, mean 69.7, median 73.3

Line #	Code
1	`/* jshint esversion: 6 */`
2	`"use strict";`
3	`...`
4	`const reviewer = {`
5	`  id: '',`
6	`  firstName: '',`
7	`  familyName: '',`
8	`  updateId: function (anId) {`
9	(Complete for Q17)
10	`  ...},` (Complete for Q17)
11	`  updateFirstName: function (aName) {`
12	`  ...},`
13	`  updateFamilyName: function (aName) {`
14	`  ...},`
15	`  convertToJSON: function () {`
16	`  ...},`
	`  convertToString: function () {`
	`  return (this.id + ' ' + this.firstName + ' '`
	`  + this.familyName);`
17	`  }`
18	`};`
19	
20	`reviewer.updateId('12345')`
	`  .updateFirstName('Bobbie')`
	`  .updateFamilyName('Firmino');`
21	
22	`reviewer.id = '54321';`
23	
24	`console.log(reviewer.convertToJSON());`
25	`...`
26	`let returnString = reviewer.convertToString;`
27	`returnString();`

Assignment 2: web client

Assignment 2

- Assignment Briefing is up on Learn
- User stories are up on Learn
- Implement User stories
- Reference server from Assignment 1 remains online for you to use.
- Source code for Reference server has been made available on eng-git.
- You will continue to use your MySQL account
- **Make sure you've done up to lab 6**

- How will this be assessed?
  - Students will exercise each others' anonymized client-side apps
  - This will take place in the labs in the last week of term
  - Example scripts for testing online

Example fragment from 2017's test script

1. Can the application be run?		YES	NO
2. Project views			
Precondition: not logged in, viewing a list of sample projects (you might need to navigate to get to this point)			
Steps	**Expected**	**Pass**	**Fail**
1. Page or scroll to see all the sample projects	Should be able to see at least 12 projects (perhaps after scrolling or paging)		
2. Find a search box, and search for `farm`	Two projects: `Let's Raise the Roof (Farm)!` and `The Farmery` should be shown in the project view		
3. Select project `The Farmery` for detailed view	Extra information for project should be shown including rewards, progress towards goal, backers and pledges		
	Totals		
Comments			

UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

SENG365 Web Computing Architecture:
Week 7
Single page applications, Intro to Vue.js, Design patterns

Ben Adams
Course Coordinator
benjamin.adams@canterbury.ac.nz
310, Erskine Building