A physicist, an engineer and a programmer were in a car driving over a steep alpine pass when the brakes failed. The car was getting faster and faster, they were struggling to get round the corners and once or twice only the feeble crash barrier saved them from crashing down the side of the mountain. They were sure they were all going to die, when suddenly they spotted an escape lane. They pulled into the escape lane and came safely to a halt.

The physicist said "We need to model the friction in the brake pads and the resultant temperature rise, see if we can work out why they failed".

The engineer said "I think I've got a few wrenches in the back. I'll take a look and see if I can work out what's wrong".

The programmer said "Why don't we try again and see if it's reproducible?"

Section 8:
Model-View-Controller

Slides adapted from Alex Mariakakis

with material from Krysta Yousoufian, Kellen Donohue, and James Fogarty

MIDTERM REFLECTION – PROBLEM 2

Is this implementation correct given the invariant?

```
{{ P: 0 < n <= str.length, chars.length, lens.length }}
int runLengthEncode(String str, int n, char[] chars, int[] lens) {
    int count = 0;
    int index = 0;
    int i = -1;
    char prev = '\0';
{{ Inv: P and str[0..i-1]=chars[0]*lens[0]+…+chars[j]*lens[j] and
chars[0]!=chars[1], …, chars[j-1]!=chars[j] and (i = 0 or cur = str[i-1]) }}
    while (i < n) {
        i++;
        if (str.charAt(i) != prev && prev != '\0') {
            lens[index] = count;
            count = 0;
            index++;
        }
        prev = str.charAt(i);
        chars[index] = prev;
        lens[index] = 0;
        count++;
        if (i + 1 == n) lens[index] = count;
    }
}
```

NO! The code does not follow the loop invariant! This makes it impossible for other programmers to understand what your loop is doing because the loop invariant details how the code functions one way, but the implementation works completely differently.

MIDTERM REFLECTION – PROBLEM 2

```
{{ P: 0 < n <= str.length, chars.length, lens.length }}
int runLengthEncode(String str, int n, char[] chars, int[] lens) {
    int i = 0;
    int j = 0;
    int cur = '\0';
{{ Inv: P and str[0..i-1]=chars[0]*lens[0]+…+chars[j]*lens[j] and
chars[0]!=chars[1], …, chars[j-1]!=chars[j] and (i = 0 or cur = str[i-1]) }}
    while (i != n) {
        if (str.charAt(i) == cur) {
            lens[j] = lens[j] + 1;
        } else {
            j = j + 1;
            cur = str.charAt(i);
            chars[j] = cur;
            lens[j] = 1;
        }
        i = i + 1;
    }
}
```

Much better! To note some key differences here, the variables declared initially line up with the variables utilized in the loop invariant. Furthermore, when the variables are incremented, it corresponds and supports the ranges for what is asserted as true in the invariant for each iteration of the loop.

MVC

- The classic design pattern
- Used for data-driven user applications
- Such apps juggle several tasks:
  - Loading and storing the data – getting it in/out of storage on request
  - Constructing the user interface – what the user sees
  - Interpreting user actions – deciding whether to modify the UI or data
- These tasks are largely independent of each other (in theory)
- Model, view, and controller each get one task
- Typically view and controller tend to be more coupled together while model is standalone

MODEL

talks to data source to retrieve and store data

Which database table is the requested data stored in?

What SQL query will get me the data I need?

VIEW

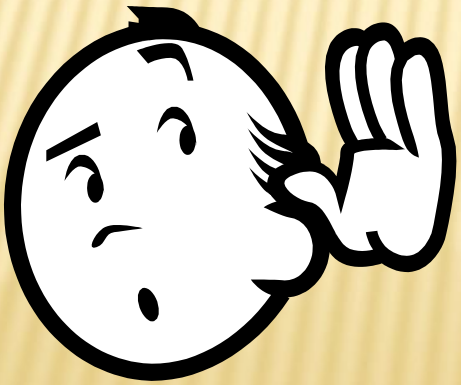
asks model for data and presents it in a user-friendly format

Would this text look better blue or red? In the bottom corner
or front and center?

Should these items go in a dropdown list or radio buttons?

CONTROLLER

listens for the user to change data or state in the UI, notifying the model or view accordingly



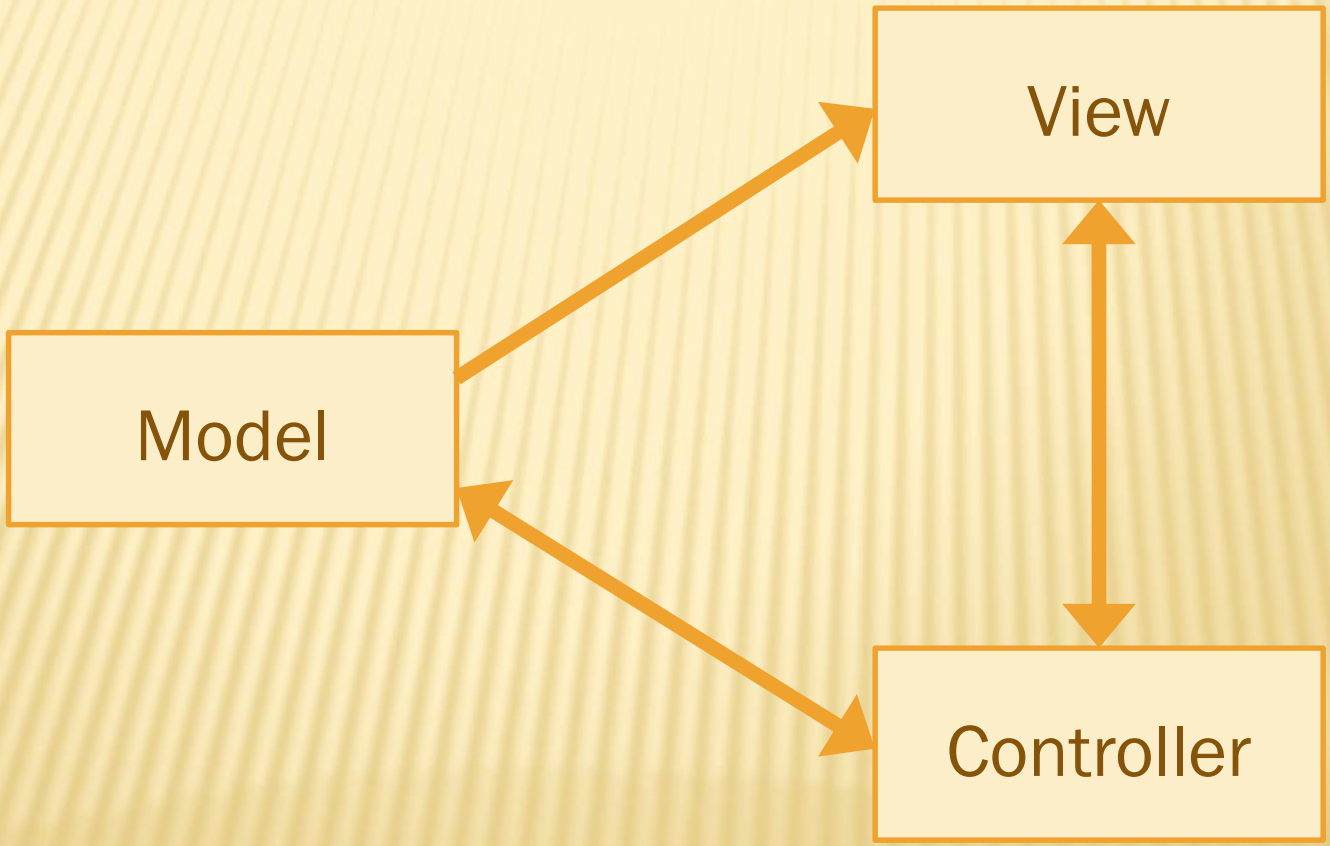
The user just clicked the “hide details” button. I better tell the view.

The user just changed the event details. I better let the model know to update the data.

BENEFITS OF MVC

- Organization of code
  - Maintainable, easy to find what you need
- Ease of development
  - Build and test components independently
- Flexibility
  - Swap out views for different presentations of the same data (ex: calendar daily, weekly, or monthly view)
  - Swap out models to change data storage without affecting user
- Modularity
  - Can test model independently and know any future bugs due to view or controller
  - View typically difficult to test – cannot simply write unit tests and must trial features manually

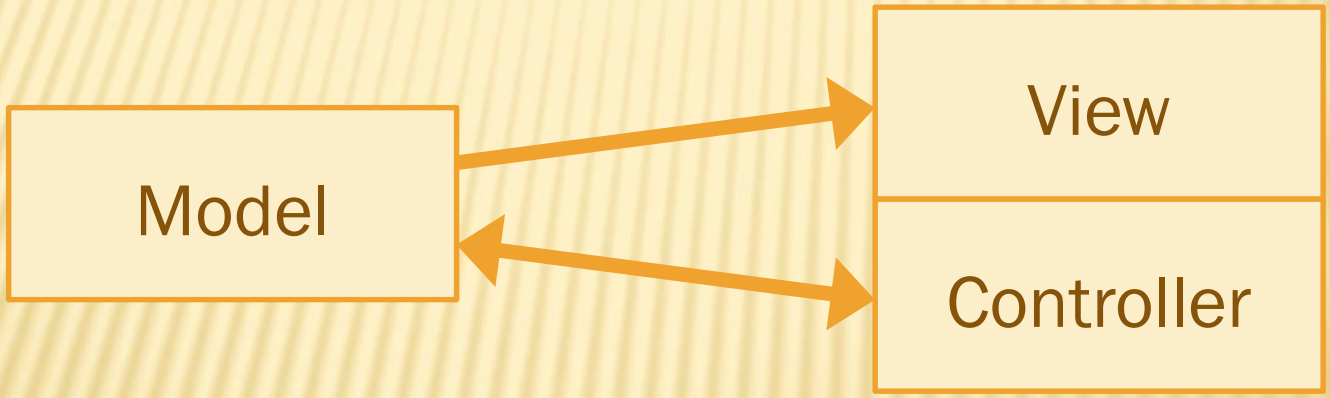
MVC FLOW IN THEORY



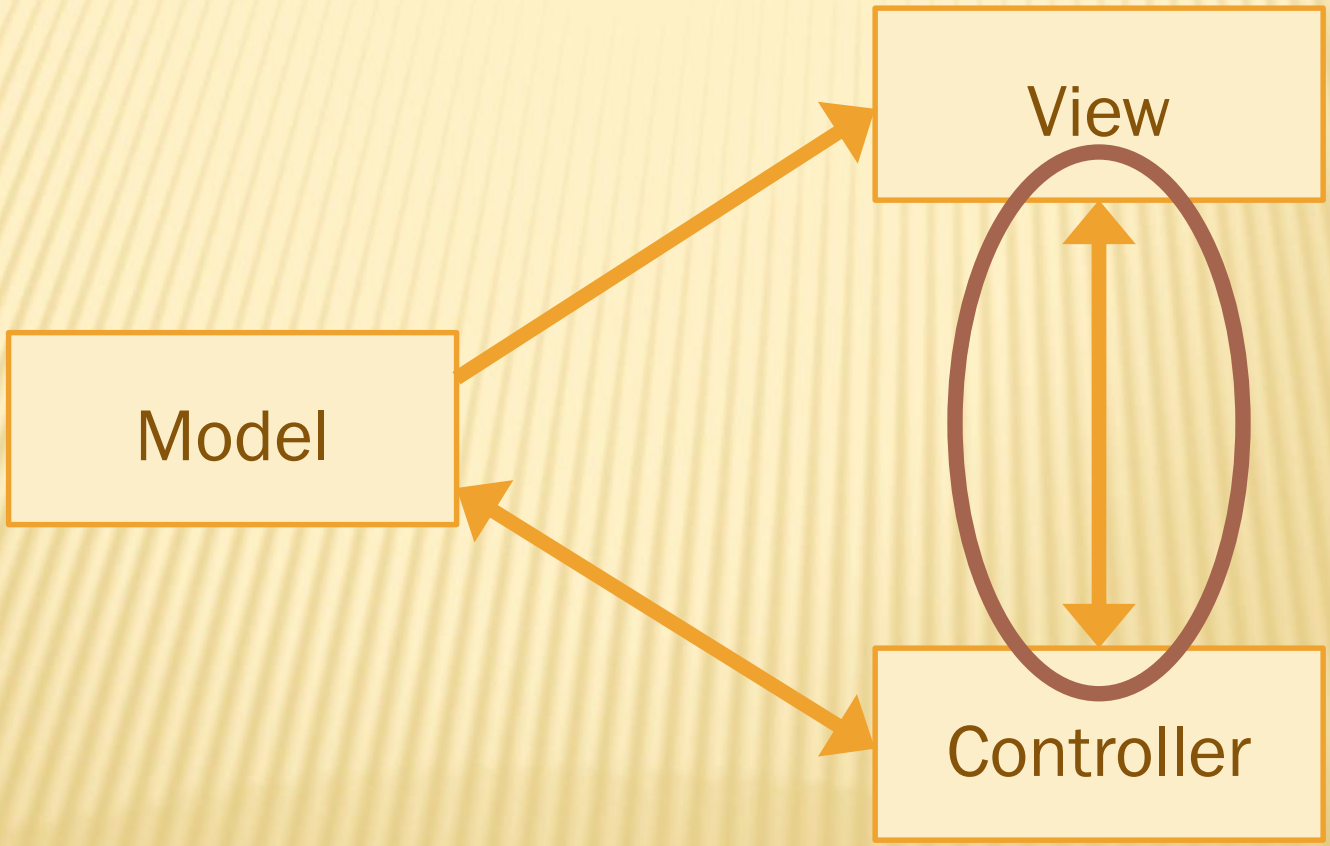
MVC FLOW

- In theory…
  - Pattern of behavior in response to inputs (controller) are independent of visual geometry (view)
  - Controller contacts view to interpret what input events should mean in the context of the view
- In practice…
  - View and controller are so intertwined that they almost always occur in matched pairs (ex: command line interface)
  - Many architectures combine the two
  - Other architecture names similar to MVC include: Model-view-adapter, Model-view-presenter, Model-view-viewmodel
  - Notice how the model and view are always distinguished – no one knows how to completely decouple controller!

MVC FLOW IN PRACTICE



PUSH VS. PULL



PUSH VS. PULL ARCHITECTURE

- Push architecture
  - As soon as the model changes, it notifies all of the views
- Pull architecture
  - When a view needs to be updated, it asks the model for new data

PUSH VS. PULL ARCHITECTURE

- Advantages for push
  - Guaranteed to have latest data in case something goes wrong later on
- Advantages for pull
  - Avoid unnecessary updates, not nearly as intensive on the view

MVC EXAMPLE – TRAFFIC SIGNAL

TRAFFIC SIGNAL – MVC

Component	Model	View	Controller
Detect cars waiting to enter intersection			
Traffic lights to direct car traffic			
Regulate valid traffic movements			
Manual override for particular lights			
Detect pedestrians waiting to cross			
Pedestrian signals to direct pedestrians			
External timer which triggers changes at set interval			

TRAFFIC SIGNAL – MVC

Component	Model	View	Controller
Detect cars waiting to enter intersection			X
Traffic lights to direct car traffic		X	
Regulate valid traffic movements	X		
Manual override for particular lights			X
Detect pedestrians waiting to cross			X
Pedestrian signals to direct pedestrians		X	
External timer which triggers changes at set interval			X

TRAFFIC SIGNAL

- **Model**
  - Stores current state of traffic flow
    - Knows current direction of traffic
    - Capable of skipping a light cycle
  - Stores whether there are cars and/or pedestrians waiting
- **View**
  - Conveys information to cars and pedestrians in a specific direction
- **Controller**
  - Aware of model's current direction
  - Triggers methods to notify model that state should change

TRAFFIC SIGNAL CODE

- **Model**
  - TrafficModel – keeps track of which lights should be on and off
- **View**
  - CarLight – shows relevant state of TrafficModel to cars
  - PedestrianLight – shows relevant state of TrafficModel to pedestrians
- **Controller**
  - PedestrianButton – notifies TrafficModel that there is a pedestrian waiting
  - CarDetector – notifies TrafficModel that there is a car waiting
  - LightSwitch – enables or disables the light
  - Timer – regulates time in some way, possibly to skip cycles

HW8 OVERVIEW

- Apply your generic graph & Dijkstra's to campus map data
- Given a list of buildings and walking paths
- Produce routes from one building to another on the walking paths

HW8 DATA FORMAT

- List of buildings (abbreviation, name, loc in pixels)

```
BAG Bagley Hall (East Entrance) 1914.5103,1708.8816
BGR By George 1671.5499,1258.4333
```

- List of paths (endpoint 1, endpoint 2, dist in feet)

```
1903.7201,1952.4322
        1906.1864,1939.0633: 26.583482327919597
        1897.9472,1960.0194: 20.597253035175832
        1915.7143,1956.5: 26.68364745009741
2337.0143,806.8278
        2346.3446,817.55768: 29.685363221542797
        2321.6193,788.16714: 49.5110360968527
        2316.4876,813.59229: 44.65826043418031
```

- (0,0) is in the upper left

MVC IN HW8

- **Model** stores graph, performs Dijkstra's
- **View** shows results to users in text format
- **Controller** takes user commands and uses view to show results
- **View** and **Controller** will change in HW9, but **Model** will stay the same
  - Yet another example of how the Model is portable to different interfaces, but the view and controller are coupled cannot be transferred to different UI

MVC IN ANDROID HW9

View – Android device application screen (including campus map image, display of button, and list of building abbreviations)

Controller – Pressing button makes text appear on screen “BAG”

Model – List populated with building abbreviations draws on buildings stored in campus graph

- Notice how tightly coupled the controller and view are here, but our same campus model is easily incorporated!