A physicist, an engineer and a programmer were in a car driving over a steep alpine pass when the brakes failed. Their car raced down the mountain, and only a conveniently-placed escape lane saved them.

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

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
- Model, view, and controller each get one task

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

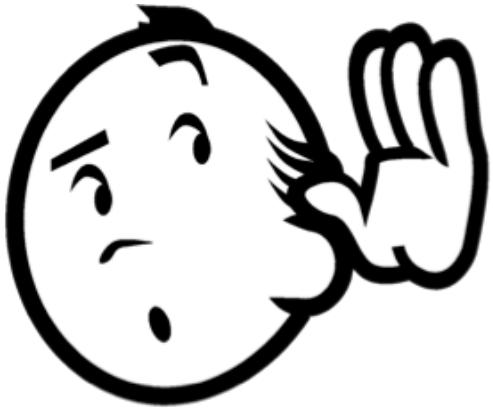
asks model for data and presents it in a user-friendly format

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



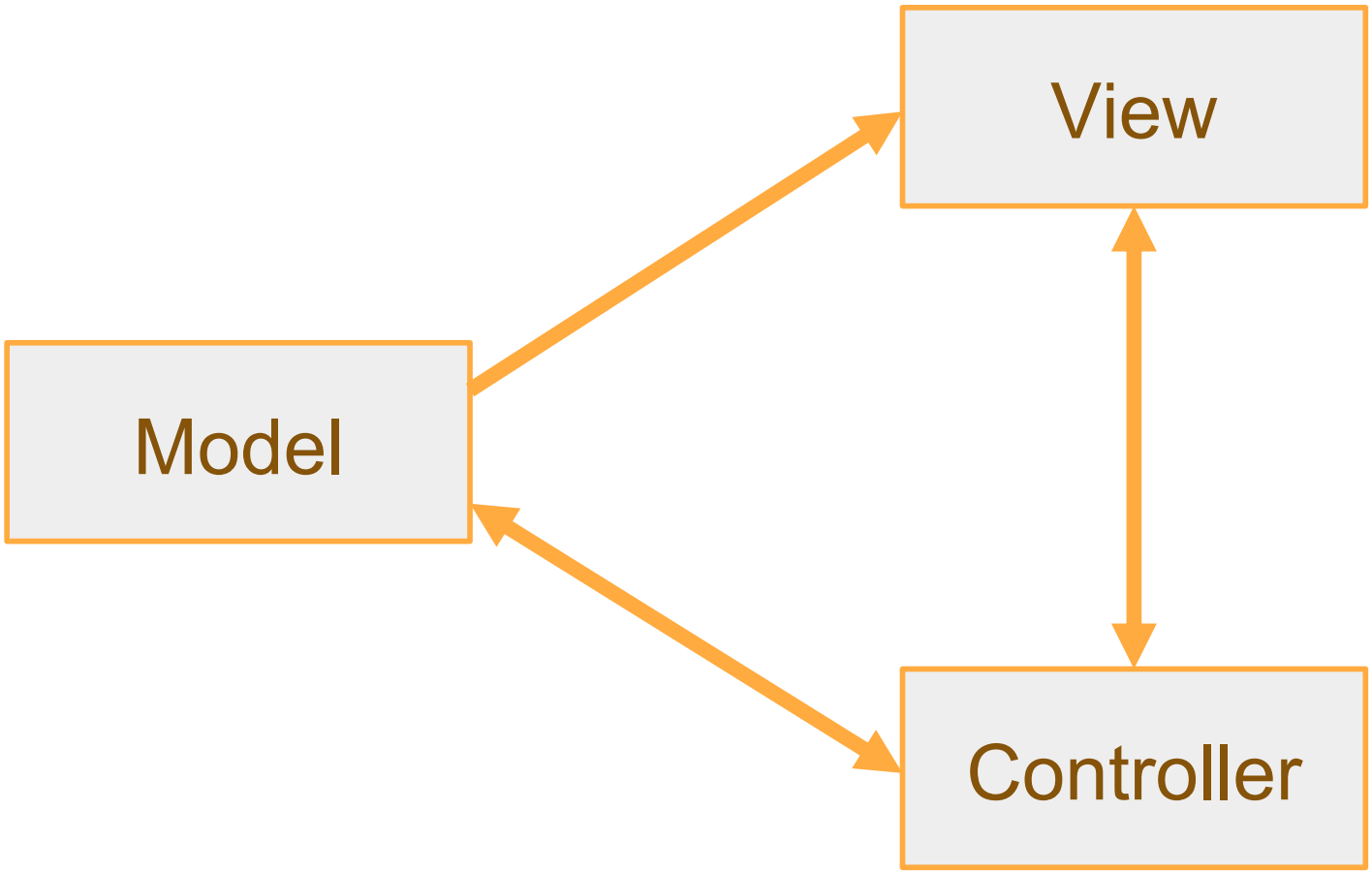
The user just clicked the “hide details” button. I better tell the view.

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

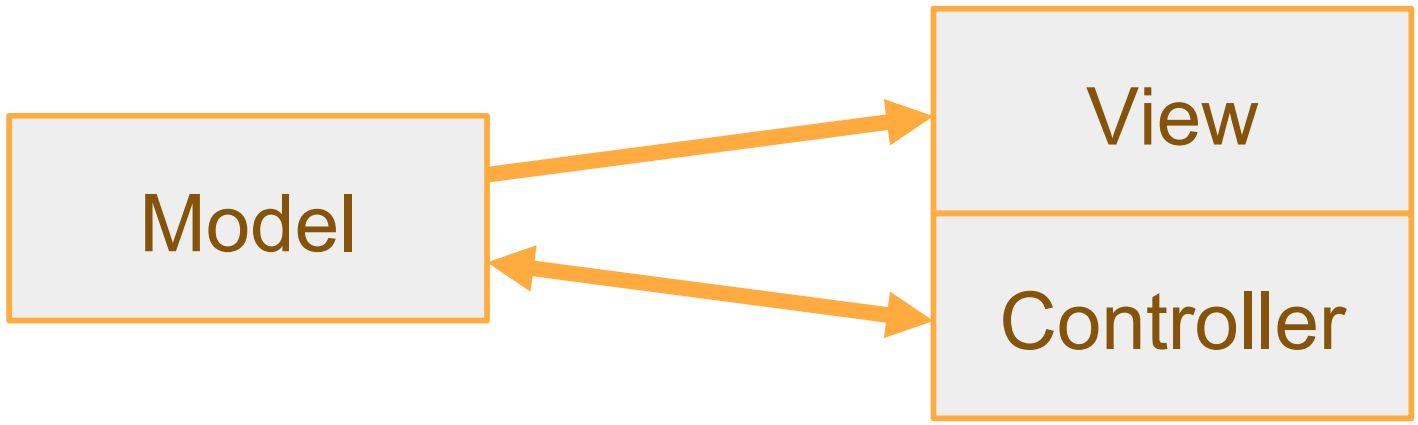
MVC FLOW IN THEORY



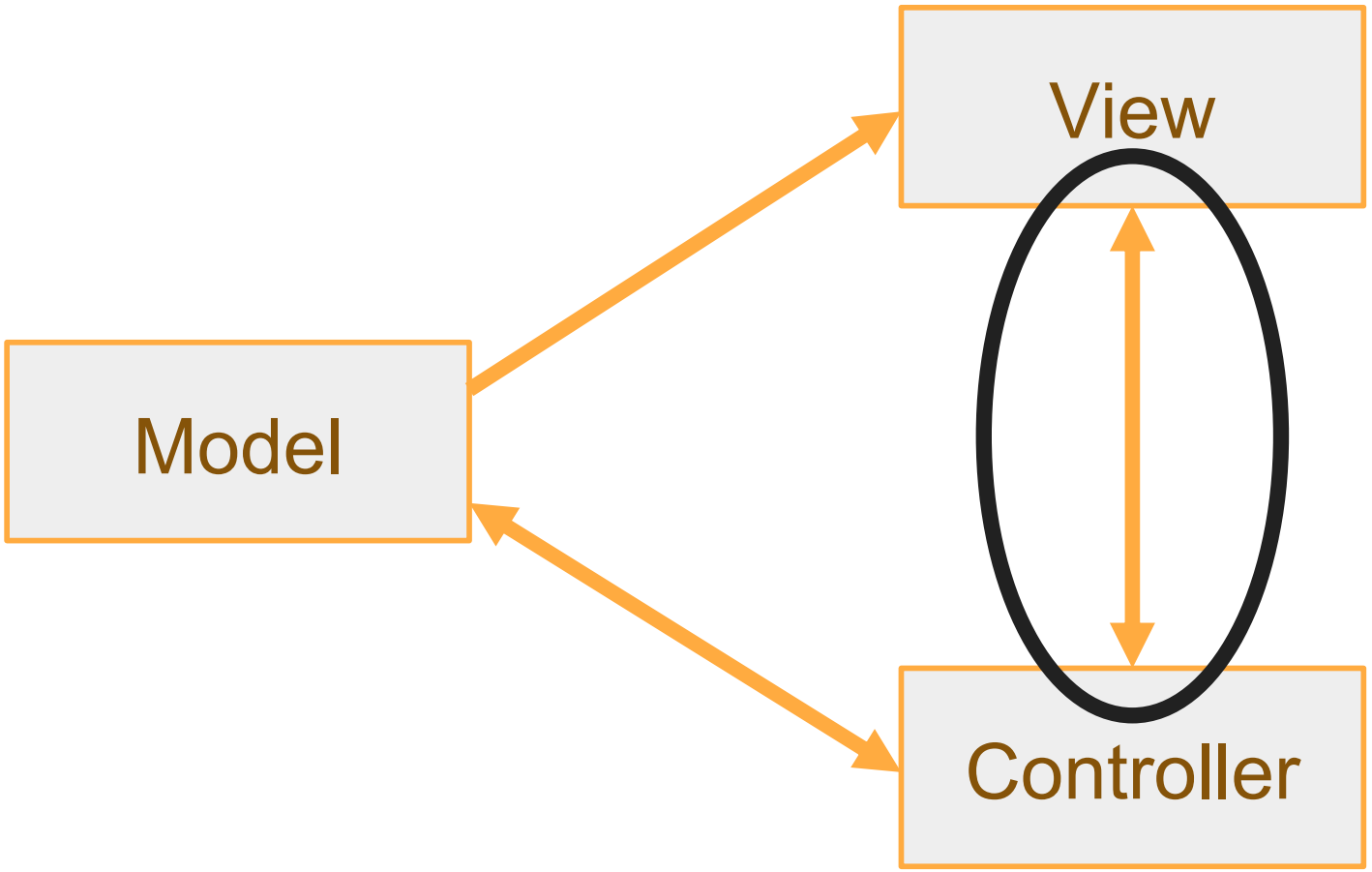
MVC FLOW

- 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

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		X	X
Traffic lights to direct car traffic	X		
Decide to change the light's status			X
Manual override for particular lights			X
Detect pedestrians waiting to cross		X	
Pedestrian signals to direct pedestrians			
External timer which triggers changes at set interval			X

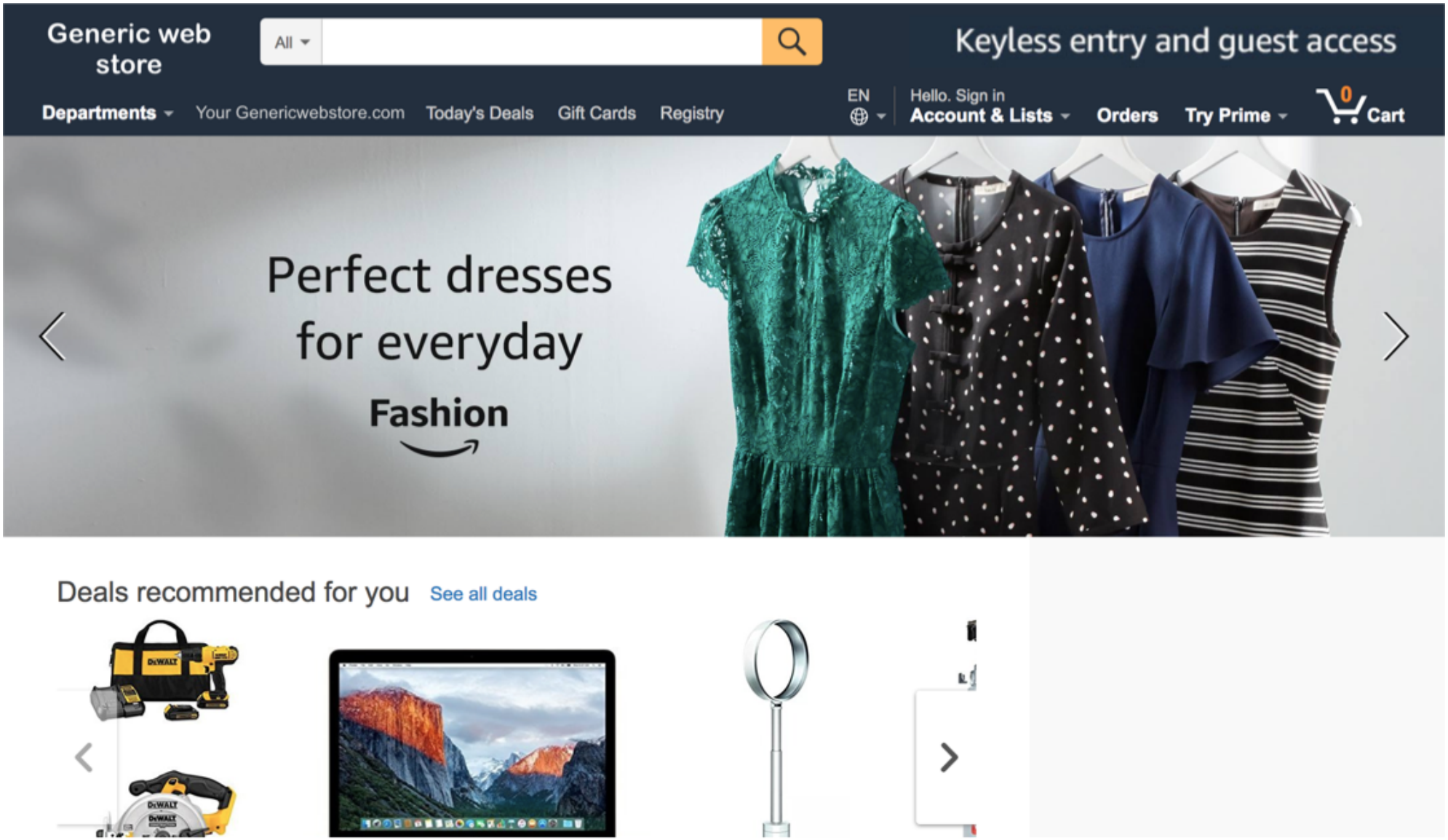
TRAFFIC SIGNAL

- **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

MVC EXAMPLE – WEB STORE



WEB STORE – MVC

Component	Model	View	Controller
Update user's shopping cart			
Display price/details of a product			
Storage of product/inventory details			
Purchase items in shopping cart			
Record of customer transactions			
User sign-in			
Authenticate user sign-in attempt			
Check user credentials			

WEB STORE – MVC

Component	Model	View	Controller
Update user's shopping cart			X
Display price/details of a product		X	
Storage of product/inventory details	X		
Purchase items in shopping cart			X
Record of customer transactions	X		
User sign-in		X	
Authenticate user sign-in attempt			X
Check user credentials	X		

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