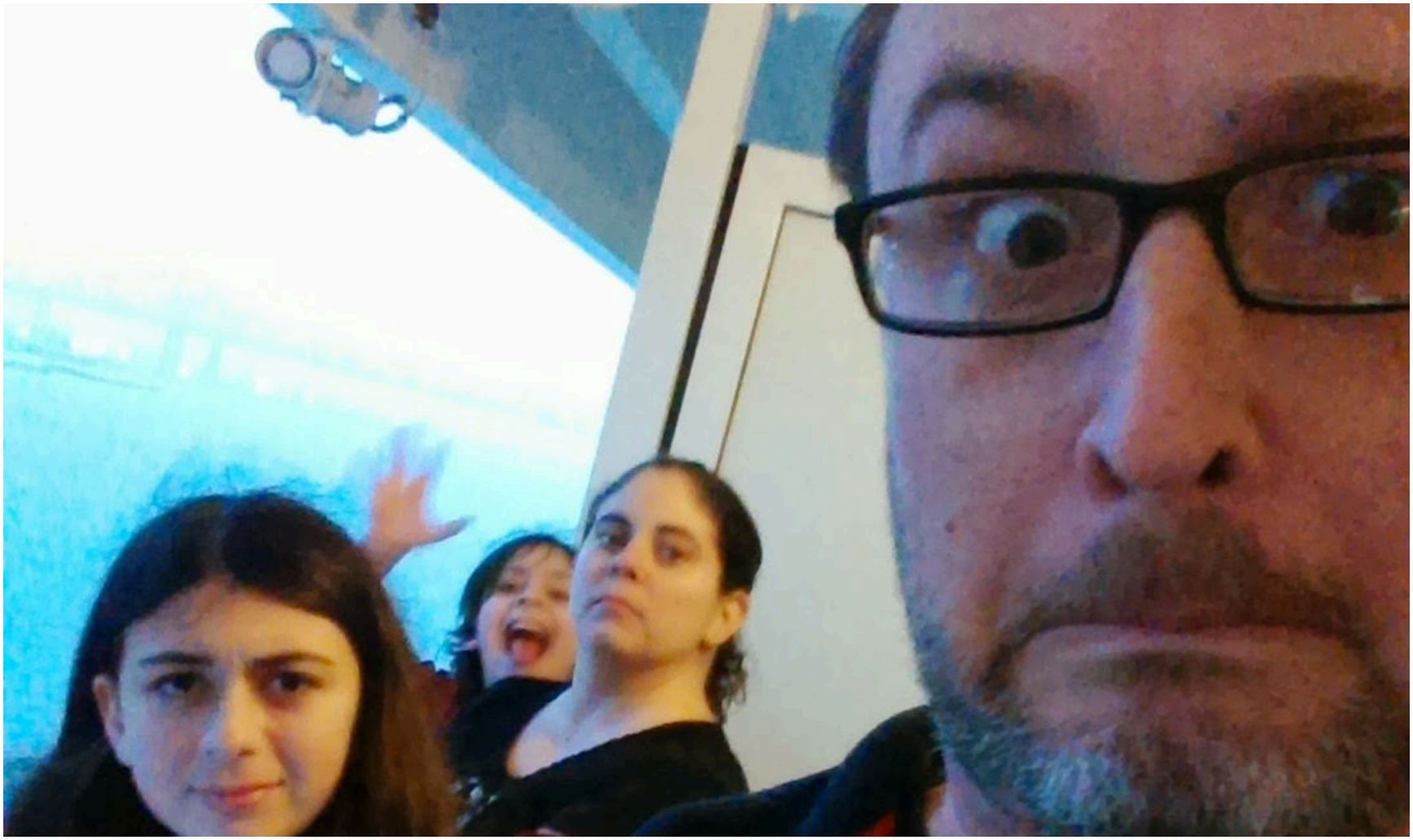
CODING
THE HAPPY MEAL

A little about me



A little bit about procedural code

1. Query database for list of program members
2. Loop through members
3. Check if member's settings allow for emails to be sent
4. If user is a man and emails are allowed
   - Query database and get email message for men

   If user is a woman and emails are allowed
   - Query database and get email message for women
5. Get member's name and, if there is a preferred name, use it otherwise use first name
6. Construct email
7. Connect to SMTP server and send email
8. Add member's email address to a list of Recipients
9. Query the database to update member's record that email was sent to them
10. Query the database of all related program managers
11. Loop through them
12. Send them an email with a list of all recipients
13. …

A little bit about procedural code

- Task based
- Sequential
- Many decision trees
  - If/Else statements
- Repetitive
- In the bigger picture, VERY repetitive.
  - Querying the database for the same information in multiple places
  - Making the same decisions in multiple places. Like using preferred name instead of first.
- Due to the specificity of the tasks being accomplished, the more complex the system becomes, the more difficult to change it.

How is OOP different?

- Model the world you are interacting with
  - Member
  - Manager
  - Email
  - Settings
  - Email Service
- Build behavior into your objects outside of a specific set of tasks:

```
member.preferred_name #return preferred,first if not available
member.get_message #return gender specific message
email.send(message) #sends email via SMTP server
```

What is an Object

Super simple explanation

An object is a data construct that has attributes and behaviors.

Attributes are also known as properties. A car might be an object, it may have attributes of color (green) and top speed (125)

Behaviors are known as methods or functions. An object may have many methods. A car may have methods “stop” and “go”

Ruby Object

```ruby
class Soda < Drink
  def initialize(syrup, water)
    @syrup = syrup
    @water = water
    @calories = 100
  end
  def calories
    return @calories
  end
end
```

Object Domain

You are presented with a set of features or functions your web app needs to have.

- What are the real world objects that are involved?
- What are their properties?
- What are their behaviors?
- How do they relate to each other?

A good Object Domain will

- allow you to ask it questions and get back answers
- allow you to extend it, building in new features
- allow you to fix broken parts without affecting the overall system

Four Core Principles

Four Core Principles

- Encapsulation

Four Core Principles

- Encapsulation
- Inheritance

Four Core Principles

- Encapsulation
- Inheritance
- Composition

Four Core Principles

- Encapsulation
- Inheritance
- Composition
- Polymorphism

Encapsulation

- All data associated with an object is “private”
- Properties/Attributes are private
- Methods to access properties are public
- Methods can also be private
- You have to “ask” the object to get or change a property
- Core methods are called:
  - Accessors
  - Getters and Setters
  - Mutators

Inheritance

- Objects can inherit properties and behaviors from other objects
  - Sub Classes
  - Super Classes
- Sub classes “inherit” properties and behaviors from its super class
- Sub classes “overwrite” properties and methods that are specific to it

Inheritance

- Super Class: Car
  - Property: Top speed
  - Method: go(‘fast’)

  - Sub Class 1 : Clunker
    - “Type” of Car
    - Top Speed property is set to 25
    - It inherits a “go” method, but can only go 25mph
  - Sub Class 2: Sports Car
    - “Type” of Car
    - Top Speed property is set to 125
    - It inherits a “go” method, but it can go 125mph

Inheritance

“Is A”

- A clunker “is a” car
- A dolphin “is a” mammal
- A mammal “is an” animal
- A dolphin “is a” animal
- A Sprite “is a” drink

Composition

- Single Objects are composed of multiple other objects
- Is generally preferred over inheritance
- Allows for greater flexibility as systems become more complex
- Does not prevent inheritance and can extend a super classes
- Often, the nature of a composed object is determined more by the objects it is composed of rather than it's own properties or attributes

Composition

“Has a”

- A car “has a” engine
- A car “has” 4 wheels
- A human “has” opposable thumbs
- A happy meal “has a” drink
- A happy meal “has a” hamburger
- A happy meal “has a” small fries

Composition vs. Inheritance

Inheritance Example

Object: Car

- Sports Car is a car
- Clunker is a car
- Sports Car/Clunker “extends” the car super class
- The sub classes overwrite the top speed property

Composition vs. Inheritance

Composition Example

Object: Car

- Has a transmission
- Has four wheels
- Has an engine
  - Engine object has a top speed property

- Unlike inheritance, top speed is not determined by the car's top speed attribute but by the type of engine it is composed with.

Polymorphism

- Two or more objects of different classes respond to the same method
- Example: The Cut() Method
  - Doctors Cut
  - Hair Stylists Cut
  - Directors Cut
  - Serial Killers Cut
- In each case, the object “cuts”, but the action is different

The Happy Meal

Three Simple Items

Three Simple Items

Is it really that simple?

Three Simple Items

Three Simple Items

Three Simple Items

Three Simple Items

Three Simple Items

Three Simple Items

Composition

/(ﾟ°□°)ﾉ ┻━┻

Not so simple after all…

Composition

Composition allows for flexibility. The variety of different combinations of a “Happy Meal” object can be left to the objects that actually compose the meal.

Outside of the variety, a happy meal **IS** actually pretty simple!

- A Happy Meal has a drink
- A Happy Meal has a main course
- A Happy Meal has a side dish

Composition

Composition allows for flexibility. The variety of different combinations of a “Happy Meal” object can be left to the objects that actually compose the meal.

Outside of the variety, a happy meal IS actually pretty simple!

- A Happy Meal has a drink
- A Happy Meal has a main course
- A Happy Meal has a side dish
- A Happy Meal has a toy

Encapsulation

How many calories are in this happy meal?

Three Simple Items

Let's break this down into an Object Domain using all 4 principles of object oriented programming…

Inheritance: The Soft Drink

Inheritance: The Soft Drink

If many possible objects share the same characteristic or properties, then inheritance will allow you to consolidate all the shared characteristics in a “Super Class”.

A generic soft drink has several possible properties:

1. Carbonated Water
2. Syrup
3. Ice
4. Syrup to carbonated water ratio

Syrup is what differentiates all soft drinks.

The Soft Drink: Coke

A Coke inherits from “Soft Drink” and overwrites syrup property with “coke”

The Soft Drink: Coke

A Dr. Pepper inherits from “Soft Drink” and overwrites syrup property with “dr. pepper”

The Soft Drink: Coke

A Sprite Zero inherits from “Soft Drink” and overwrites syrup property with “sprite zero”

Inheritance: The Soft Drink

With all three of these soft drinks. The only difference is the syrup used.

Inheritance: Profit

McDonald's want to maximize profits by saving on syrup used in their soda fountains. It would be pretty tedious to go and alter every single different type of soda…

Inheritance: Profit

Remember these 4 properties?

1. Carbonated Water
2. Syrup
3. Ice
4. Syrup to carbonated water ratio

Inheritance: Profit

Remember these 4 properties?

1. Carbonated Water
2. Syrup
3. Ice
4. **Syrup to carbonated water ratio**

Inheritance: Profit

Remember these 4 properties?

1. Carbonated Water
2. Syrup
3. Ice
4. **Syrup to carbonated water ratio**

Update all syrup to water ratios for ALL soft drinks by updating the super class!

The Hamburger

The Hamburger

The Hamburger

A hamburger is composed of:

- Meat
- Bun / Bread
- A variety of condiments
  - Could have no condiments
  - Could have a ton of condiments

Side Dishes?

At this point. Fries are just fries. And apples are just apples. Just two different objects that neither inherit from another object or are composed of multiple objects.

The Happy Meal

This little happy meal has become pretty complex!!!

Happy meal is composed of:

1 Drink that inherits from “Soft Drink”. The drink will swap out the syrup to make it the correct type

1 Main meal that could be chicken nuggets OR a hamburger. If a hamburger, then the main meal is actually an object that is composed of several other objects. We don’t actually know how many.

1 Side Dish that is just a plain object. No inheritance or composition. Just a boring object.

The Happy Meal

But how many calories are in this happy meal?

Encapsulation!

Wouldn't it be nice if we could just ask the happy meal how many calories it is?

Sounds like a difficult question to answer!

Wouldn't it be nice if we could ask a soft drink how many calories it is?

Well that sounds easier. It's just the syrup.

Wouldn't it be nice to ask a hamburger how many calories it is and have the answer be immediately adjusted based on the number and types of condiments that were added?

Good grief…

Encapsulation: Soft Drinks

A Soft Drink object can respond to a method asking how many calories it has. The object will look up it's "diet" Boolean flag (true or false) and return either 140 or 0, if diet.

- Nothing but the Soft Drink object needs to know how this is determined.
- At a later date, a more exact calculation based on syrup and "syrup to water" ratios can be implemented.
- Any sub type of Soft Drink (sprite, coke, fanta) can respond to this method because it has been inherited.

Encapsulation: Side Dishes

Both Fries and Apples are independent objects that simple have a calories property and a *Getter* method implemented. In both of these cases, the number of calories is a static value that is returned

Encapsulation: The Hamburger

Encapsulation: Condiments

Condiments has become a new object used in constructing a happy meal.

Condiments are very simple objects that can also respond to a calories method.

Encapsulation: Hamburger

Hamburger also responds to calories. It determines total calories by calculating 150 for the bun and 200 for the all beef patty. Then, on top of the 350 calories, it will ask each and every condiment how many calories it is and then add the responding numbers

Encapsulation: Hamburger

350 Calories

+ 15 calories

+ 10 calories

+ 5 calories

+ 30 Calories

410 Total Calories

Encapsulation: Hamburger

The hamburger responds to a simple calories method. No other object knows how it is determining the number.

The hamburger, likewise, is asking each condiment how many calories.

It has no idea how the condiments are determining their number.

The Happy Meal

But how many calories are in this happy meal?

The Happy Meal

But how many calories are in this happy meal?

Ask the happy meal. It will tell you. You will have no idea how the number is determined But the Happy Meal…

1. Asks the drink how many calories
2. Asks the side dish how many calories
3. Asks the main meal how many calories
4. Add them together

What about Polymorphism?

What about Polymorphism?

We have been asking a lot of different types of objects how many calories they have…

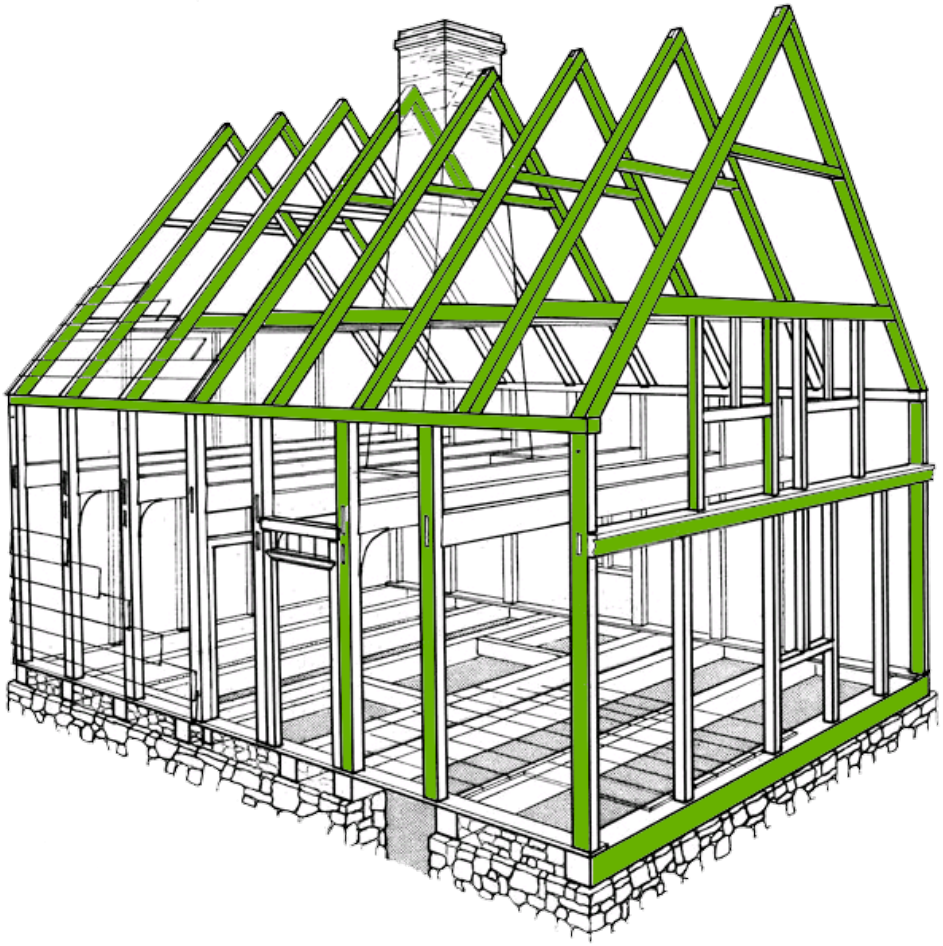
In code:

```
condiment.calories
sprite.calories
fries.calories
hamburger.calories
happy_meal.calories
```

What about Polymorphism?

What about Polymorphism?

Frameworks



What is MVC?

- Model View Controller
- Design pattern / Architecture
- Standard in the design of modern web applications
- Decouple Presentation from Data
- Separation of Concerns

What is a Framework

- …is an abstraction in which software providing generic functionality can be selectively changed by user code, thus providing application specific software. It is a collection of software libraries providing a defined application programming interface (API).

From wikipedia.org

What is a Framework

Allows designers and programmers to devote their time to meeting software requirements rather than dealing with the more standard low-level details of providing a working system, thereby reducing overall development time

From wikipedia.org

Why?

- Structured and consistent
- Alleviates common tasks
- Makes things easy
- Makes you productive
- Makes teamwork easier
- Makes applications easier to support
- Makes applications easier to extend

What is MVC?

- Model

What is MVC?

- Model
  - Your Data
  - Business Logic
  - Knows nothing about how the data will be presented to a user

What is MVC?

- Model
- View

What is MVC?

- Model
- View
  - Presentation Layer
  - HTML/CSS
  - XML/JSON
  - PDF
  - Has no idea where the data came from

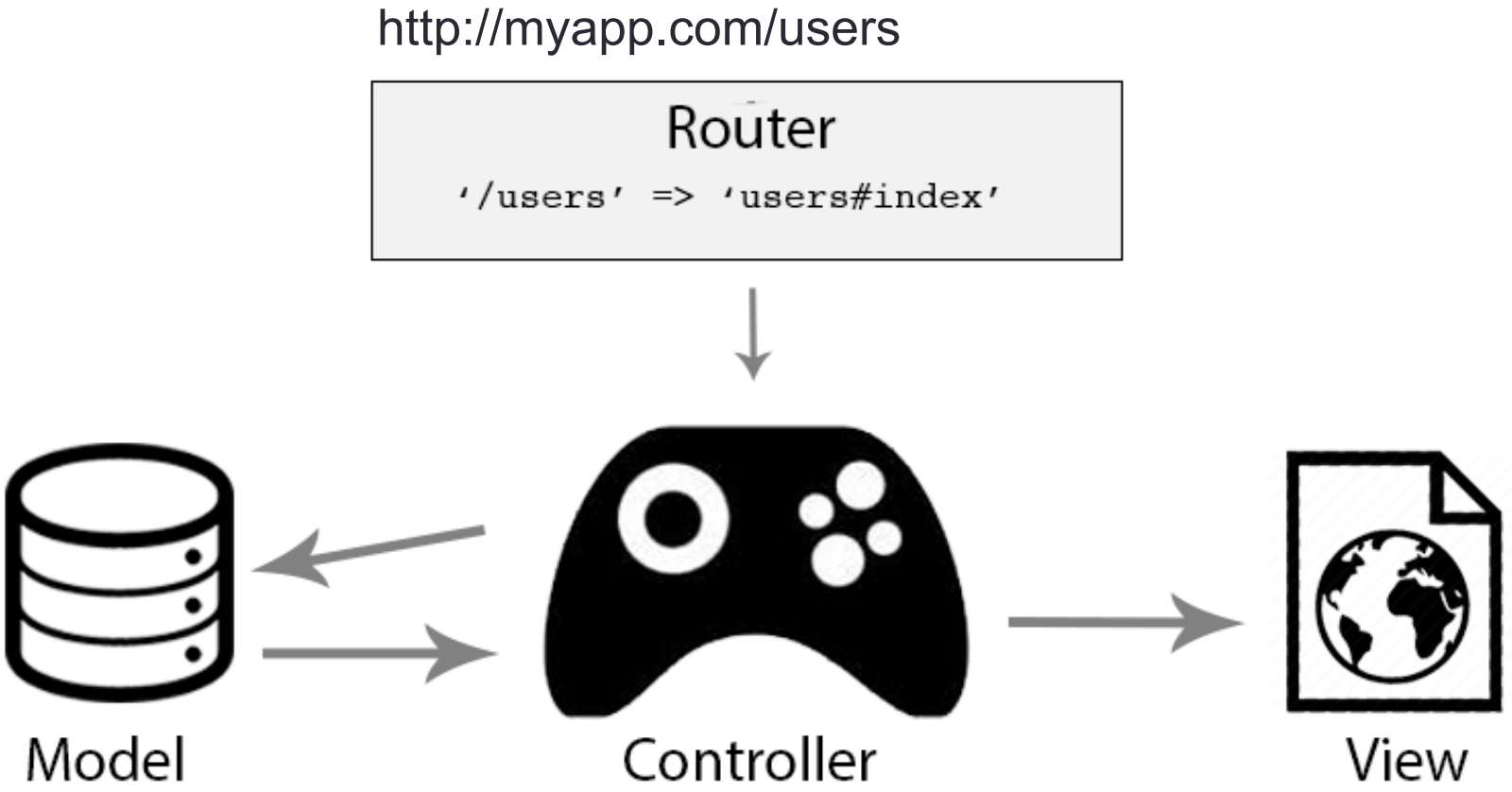
What is MVC?

- Model
- View
- Controller

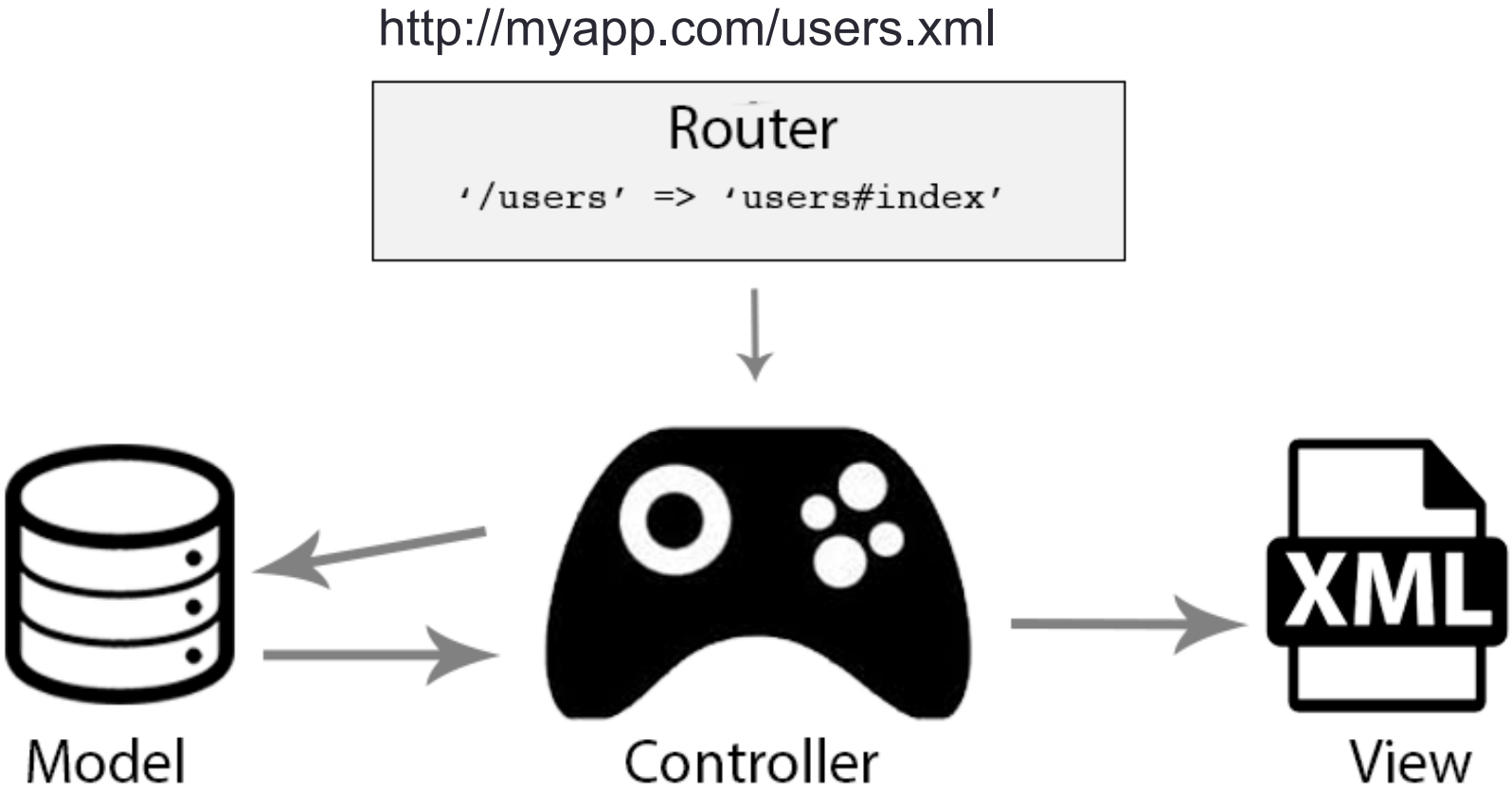
What is MVC?

- Model
- View
- Controller
  - Receives request from user
  - Talks to the model to get any needed data
  - Passes data to the view

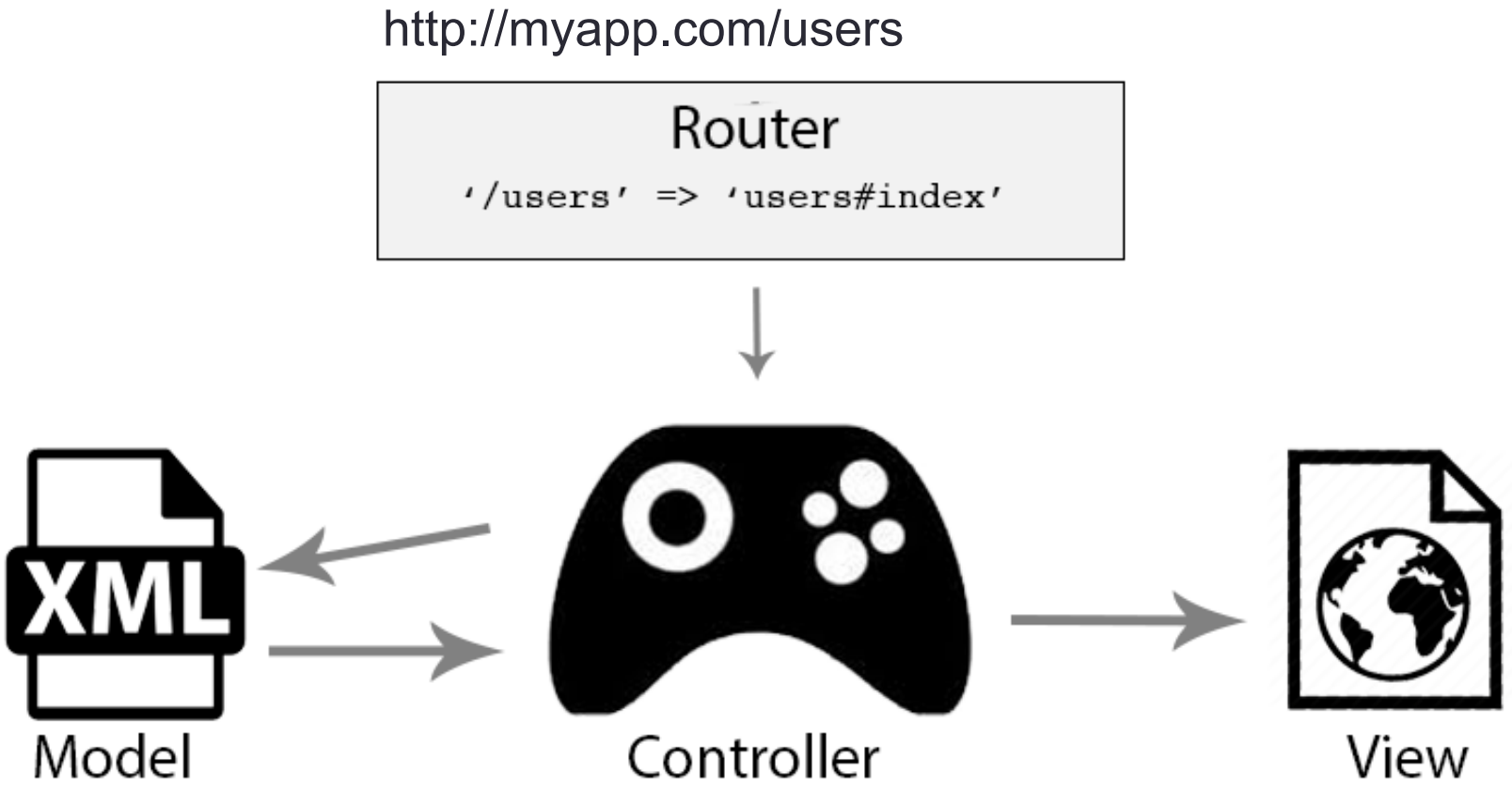
What is MVC?


http://myapp.com/users
Router
'/users' => 'users#index'
Model
Controller
View

What is MVC?


http://myapp.com/users.xml
Router
'/users' => 'users#index'
XML
Model
Controller
View

What is MVC?


http://myapp.com/users
Router
'/users' => 'users#index'
XML
Model
Controller
View

Happy Meals

- Happy Meals themselves
  - Objects
  - Composed of other Objects
  - Composed of other Objects using Inheritance
  - Composed of other Objects composed of other Objects

Happy Meals

- Happy Meals themselves
  - Objects
  - Composed of other Objects
  - Composed of other Objects using Inheritance
  - Composed of other Objects composed of other Objects
- What about ordering a Happy Meal?

Ordering a Happy Meal

Question:

How inefficient would it be if every person who worked at McDonalds took care of a single customer, one at a time?

1. Greet customer
2. Take order
3. Go in the back and cook food
4. Package food up
5. Take food to customer
6. Take money
7. Greet next customer

Ordering a Happy Meal

Question:

How inefficient would it be if every person who worked at McDonalds took care of a single customer, one at a time?

Instead, there are cashiers and cooks. There are managers.

They all have their specific role.

Cashiers do not need to learn to cook. Cooks do not need to learn to smile.

Controllers

Controllers

“Hi. I would like to order a happy meal.
Can I have it with a sprite, a hamburger
With ONLY pickles and ketchup.

And fries.”

Controllers

You do not care about who cooks your food.

You do not care (or try to avoid thinking about) how the food gets cooked.

You don’t talk to the cook. Your interaction is with the cashier.

Models

Models

What are these people doing?

Models

The cook does not care about you.

The cook doesn't even know you exist.

The cook talks to the cashier. And the cashier tells the cook what to make.

Cooks do not have to smile.

Views

This is what your order looks like

Views

And let’s be honest.

You don’t care about the cashier or the cook, do you?

All you care about is that your food gets to you and that it is super delicious.

Views

What about orders “To Go”?

Views

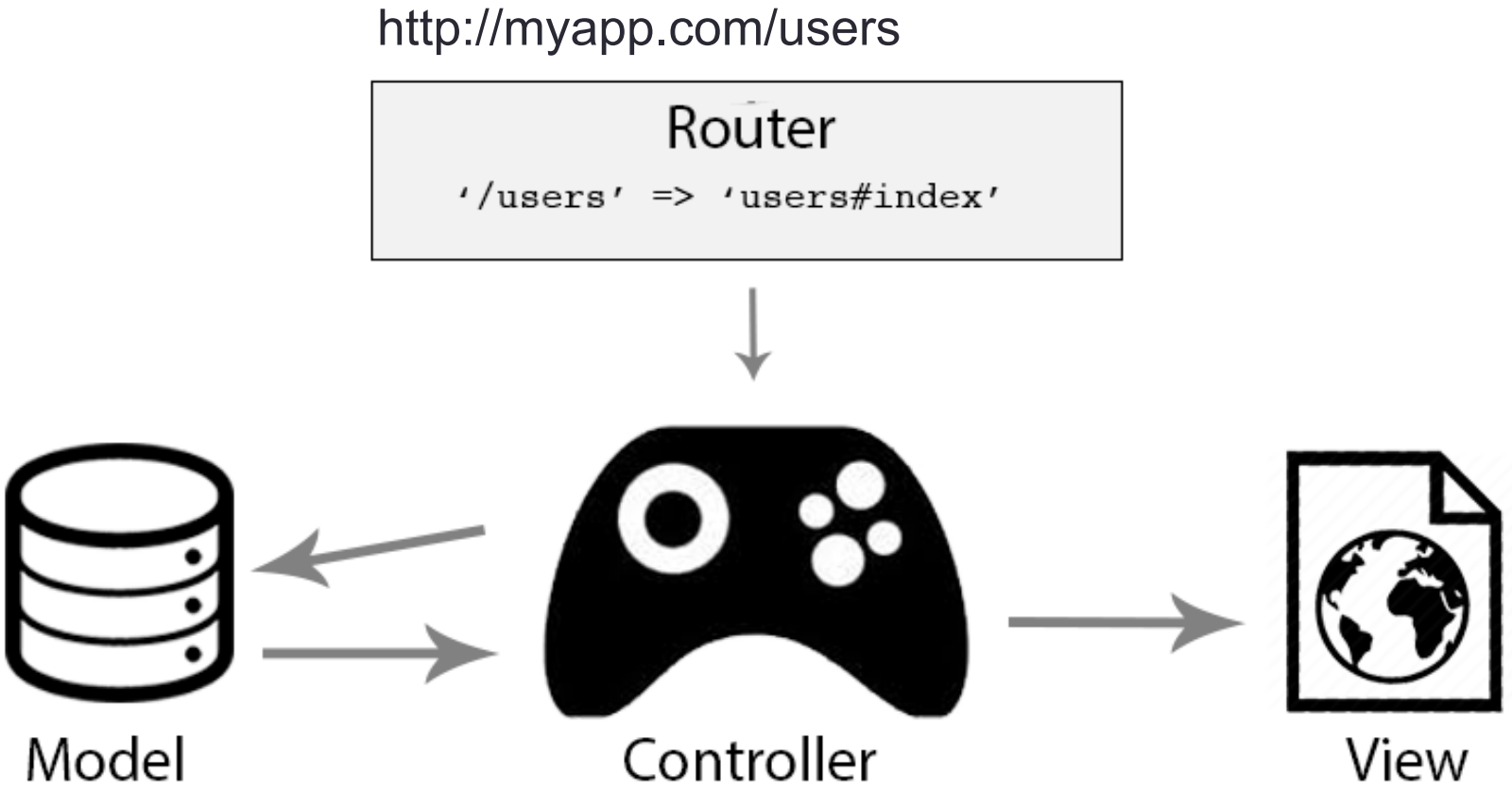
“To Go” orders only affect how the meal is packaged.

The ordering process is exactly the same. The communication between the cashier and the cook is exactly the same.

The cook still prepares the food the same way.

The cook still doesn’t care about you. You are still avoiding thinking about how your food has been made.

What is MVC?


http://myapp.com/users
Router
'/users' => 'users#index'
Model
Controller
View

API Access to your Web App

Your boss has a great idea. You need to build a web services layer. She wants you to allow API access to your app for third parties to integrate.

This is going to be great! But is it going to be major work? Are we going to have to rebuild our entire system?

API Access to your Web App

Questions?