MVC / HTTP

How (most of) the web works

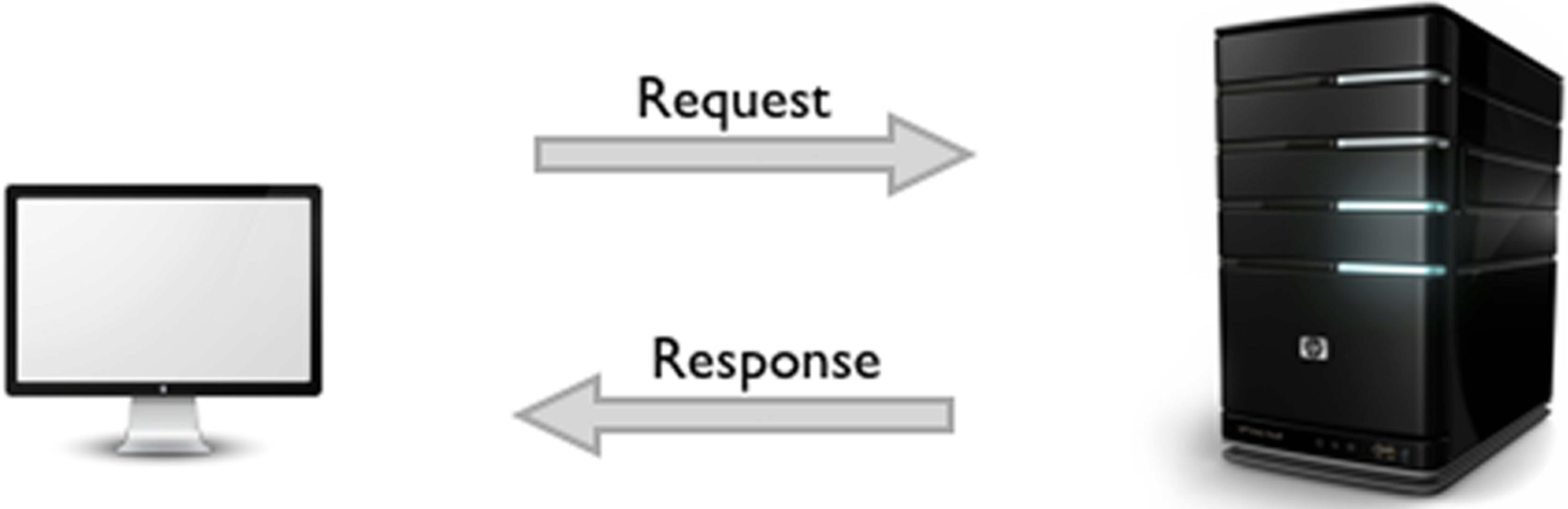
Model **V**iew **C**ontroller

- Software Architecture Pattern
- Promotes Separation of Concerns
- Popular in many web frameworks (not just Rails)
- Aligns with REST (more on this later)

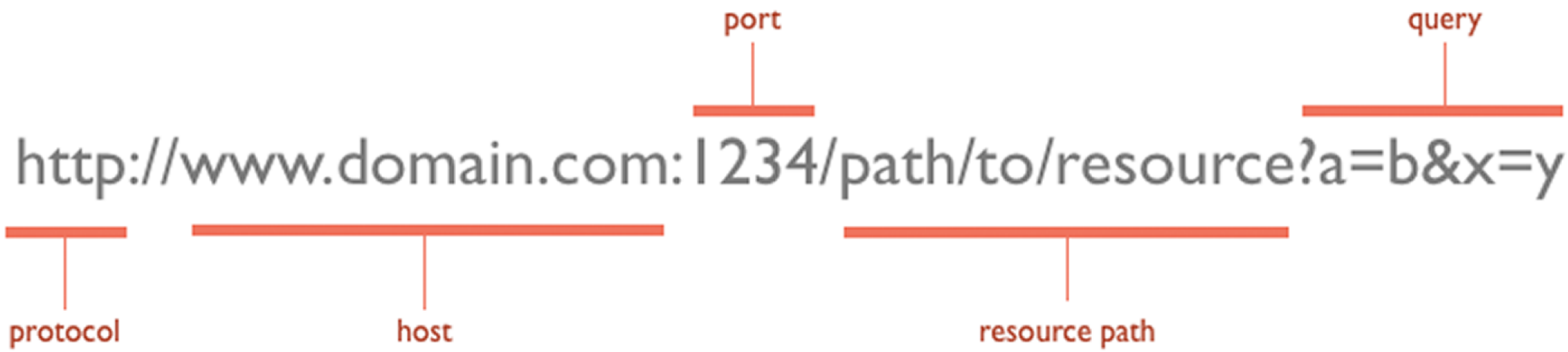
But first … HTTP

HTTP

- Powers the web … almost all of it
- Resources (pages, images, css, etc.) accessed by URL
- URL should always return the same thing … it’s a Resource Locator
- HTTP is *strictly* Request / Response
- HTTP is stateless - no link between different requests
- Except cookies & query string variables which can fake state


Request
Response

URLs

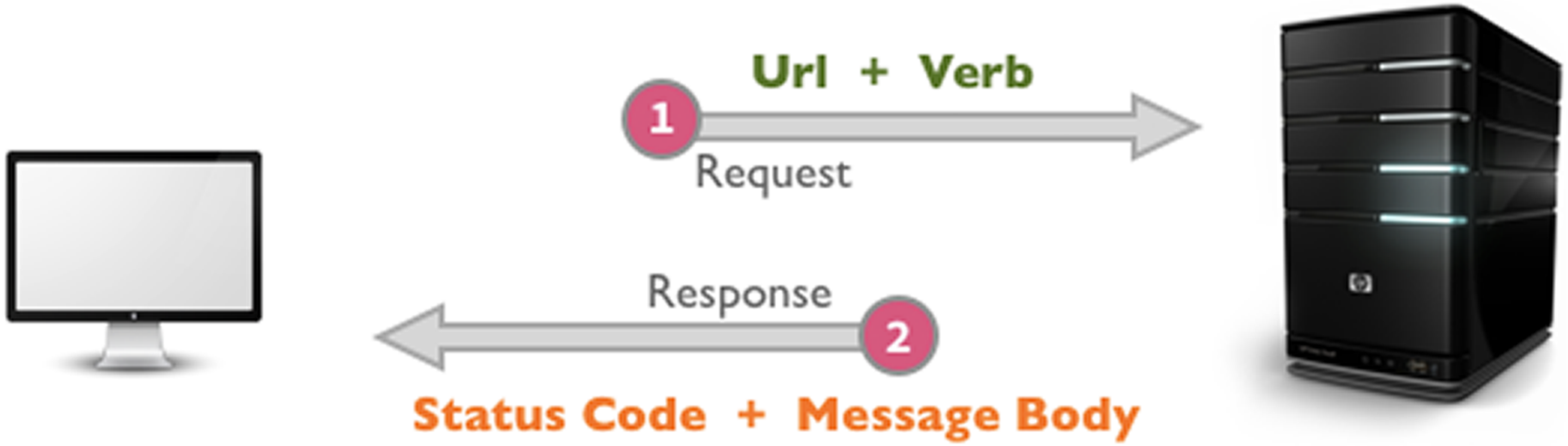

port
query
http://www.domain.com:1234/path/to/resource?a=b&x=y
protocol
host
resource path

More on URLs

- Protocol is http or https (depends on encryption)
- Host does ***not*** have to include www. despite what your aged relatives might insist
- Port is optional. Defaults to :80 for http and :443 for https
- Rails dev server runs on port :3000
- Only one server can listen on a host / port combo at a time

HTTP (continued)

- Requests are made to a server & to a Resource
  - Give me the thing at http://www.robdudley.com/about/
- Requests also have a verb that determines the type of action expected
  - GET the thing at http://www.robdudley.com/about/
- Responses are made up of:
  - Status Code
  - Message Body (often the thing you requested)


Url + Verb
1
Request
Response
2
Status Code + Message Body

HTTP (the last bit)

- Different verbs used for different things
  - GET = request something (i.e. get a page, image, stylesheet)
  - POST = create something (i.e. use this form to make a new product)
  - PUT / PATCH = update something
  - DELETE = … er … duh!
- There are other verbs (HEAD, OPTIONS, TRACE, CONNECT) but we don’t like these as they smell

HTTP (the actual last bit)

- HTTP isn’t just HTML
- Lots of different request & response types
    - HTML is text/html
    - JPG is image/jpeg
    - XML is application/xml (or text/xml)
    - JSON is application/json
- You can sometimes request responses in different formats

HTTP in the real world

- HTTP has 5 key verbs and a handful of others

- How many are available to you in HTML?

2

- HTML only supports GET & POST
- To use other verbs we need to either
    - Use JavaScript (XHR / AJAX)
    - Use a fake variable to specify the verb we meant

      e.g.

```
<input type="hidden" name="_method" value="PUT">
```

- In HTML we have 3 main vehicles for HTTP:
  - We can pass data via the query string (GET)

    /resource/?query=Fish&page=12
  - We can pass more data via a form (POST)
  - We can rule the world via JavaScript (everything)

The Life of an HTTP Request in an MVC Application

MVC - things to consider

- Traditional URLs are locations of actual things (files) on the server
- MVC applications may or may not serve actual things

  Instead they take the request and run it through some logic to work out what to return
- This makes them awesome and shiny…

1. The Request

- Requests are made to a URL
- Use one of the HTTP verbs
  (GET, POST, PUT/PATCH, DELETE)
- POST, PUT & PATCH can contain data
- Cookies & other “state” are sent with the request

2. Routing

- The Router tries to match a URL & a method
- Routes are defined in your app
- If a matching route is found, the request is passed to the controller
- If a route can’t be found then a 404 response is returned

3. Controller

- The Controller is a class that accepts a Request and associated data (if any)
- They get data from models and pass it into views
- The Controller *must* return a response (of some kind)
- Controllers should be “thin” (contain minimal logic)

4. Models

- Models are two things:
  - They map onto tables in your database (e.g. via “Active Record” or other ORM)
  - They can also contain non DB logic for organising other models or representing remote APIs (services)
- Models should be “fat” and handle the majority of your logic

5. Views

- Views can be rendered by a Controller
- They can have data passed into them
- They are a mix of HTML* and template code
- Views should not contain *much* logic

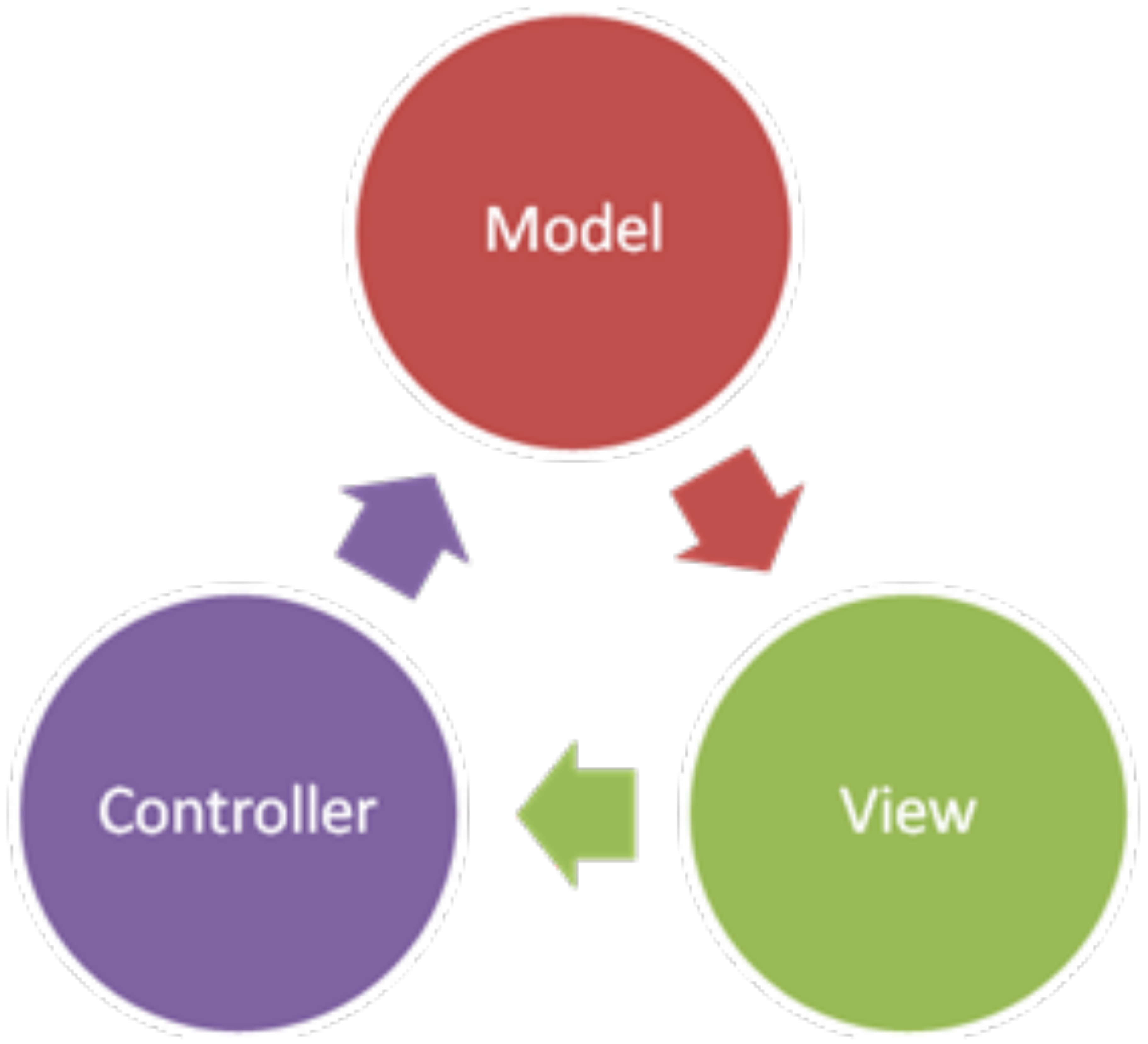
* views can also be JSON, XML, or pretty much anything else

6. The Response

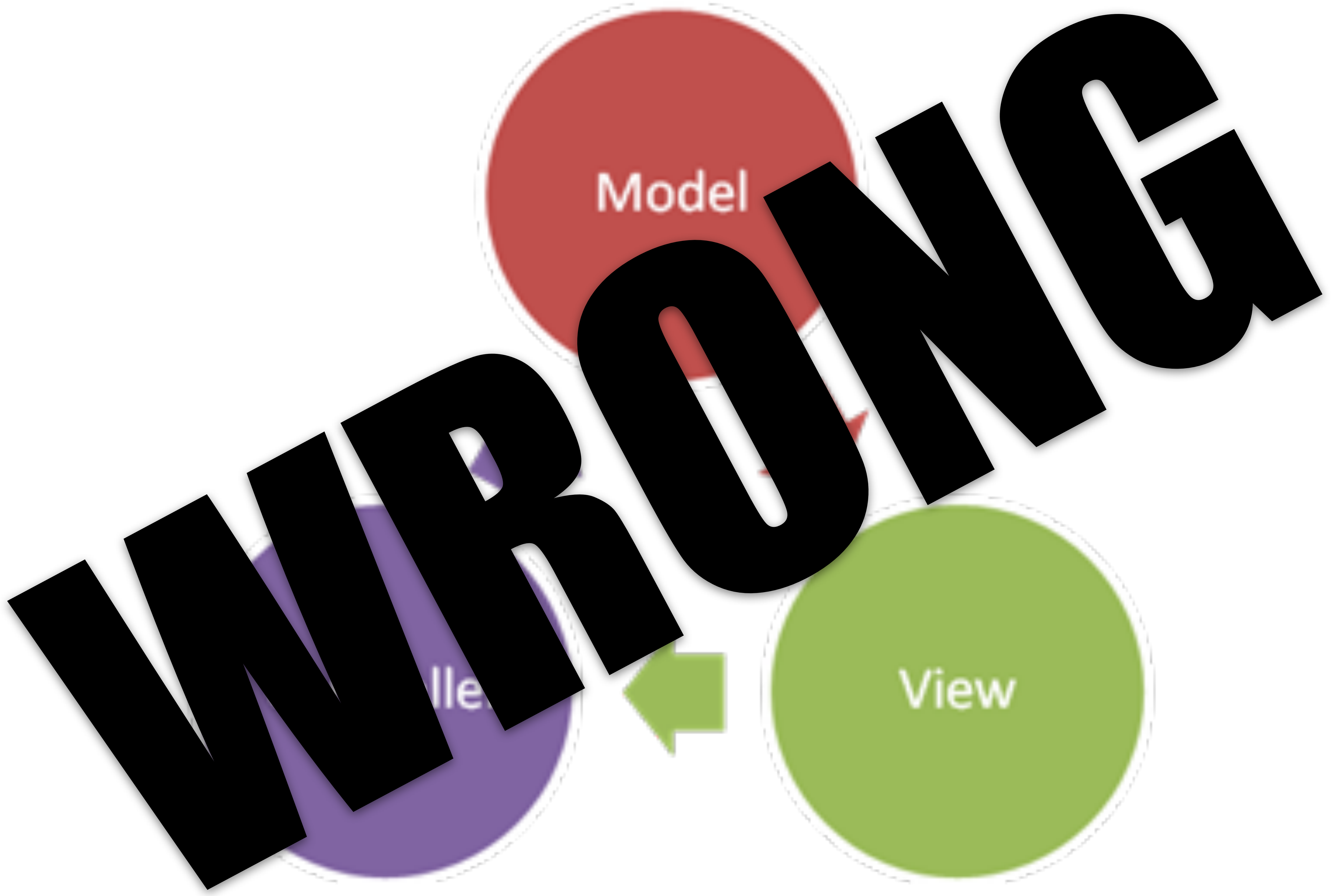
- The controller returns a response
- The response will have a HTTP response code

  (200 OK, 301 Redirect, 401 Need to Login, etc.)
- The response may have content (e.g. a view) or not

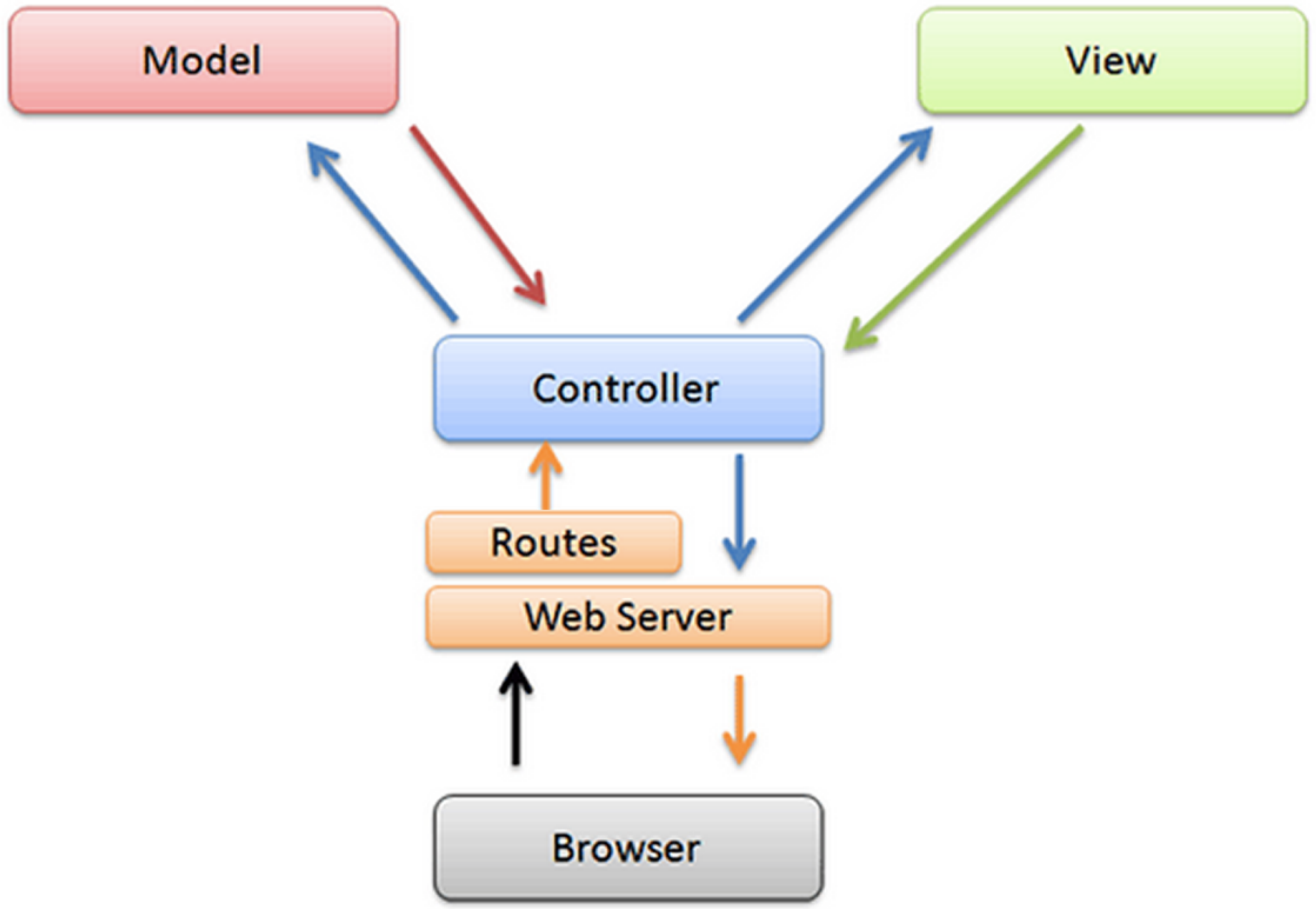
MVC Visualised



MVC Visualised



MVC Visualised



MVC - more things to consider

- Most MVC frameworks (Rails included) offer a bunch of other bits
    - Authentication built in (HTTP has 2 types of auth as well as the more common cookie or session based)
    - Middleware / Filters for passing the request through a series of tests or checks (see above)
    - Different response types (redirects, errors, etc.)
    - Different response formats

MVC Lab

Further Reading

A Beginner’s Guide to HTTP and REST
http://code.tutsplus.com/tutorials/a-beginners-guide-to-http-and-rest--net-16340

Intermediate Rails: Understanding Models, Views and Controllers
http://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/

Representational state transfer
https://en.wikipedia.org/wiki/Representational_state_transfer

What Are The Benefits of MVC?
http://blog.iandavis.com/2008/12/what-are-the-benefits-of-mvc/

Design Patterns: Elements of Reusable Object-Oriented Software (book)
http://amzn.to/1LbpLZ6