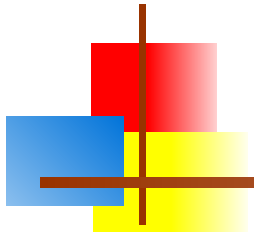
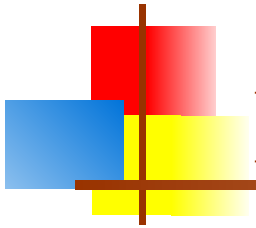
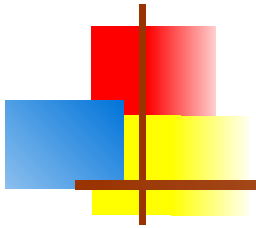


Model-View-Controller



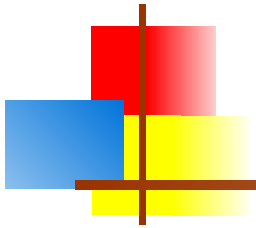
Design Patterns

- The hard problem in O-O programming is deciding what objects to have, and what their responsibilities are
- *Design Patterns* describe the higher-level organization of solutions to common problems
- Design patterns are a major topic in O-O design



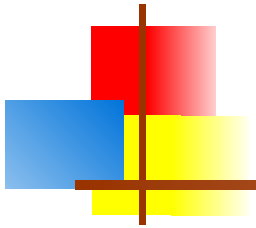
The MVC pattern

- Formulated in the late 1970s,
- A software architecture pattern built on the basis of keeping the presentation of data separate from the methods that interact with the data.
- A well-developed MVC system should allow a front-end developer and a back-end developer to work on the same system without interfering, sharing, or editing files either party is working on.



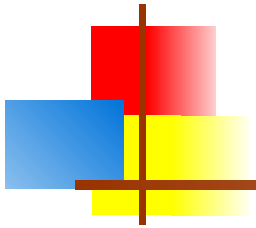
The MVC pattern

- Originally designed for personal computing
- It has been adapted and is widely used by web developers due to its emphasis on separation of concerns, and thus indirectly, reusable code.



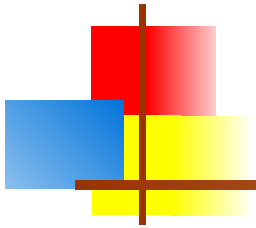
The MVC pattern

- MVC stands for Model-View-Controller
- At the core of MVC design pattern is separation of responsibility.
- The Model is the actual internal representation
- The View (or a View) is a way of looking at or displaying the model
- The Controller provides for user input and modification
- These three components are usually implemented as separate classes



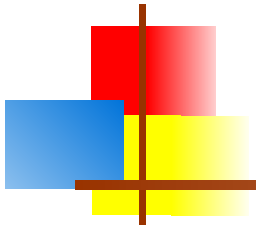
The Model

- The **model** deals with persistent storage, such as database, json file, xml file, etc.
- Doesn't matter where you have decided to store data, if you are planning on manipulating (create, read, update, delete - CRUD) that data, it should be done through the model.



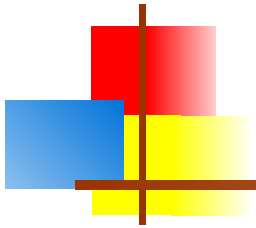
The Model

- Most programs are supposed to do work, not just be "another pretty face"
  - but there are some exceptions
  - useful programs existed long before GUIs
- The Model is the part that does the work--it *models* the actual problem being solved
- **The Model should be independent of both the Controller and the View**
  - **But it provides services (methods) for them to use**
- Independence gives flexibility, robustness



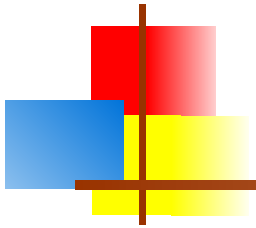
The Controller

- The **controller** is the think tank of our software.
- It maintains liaison between the view and the model.
- When should one view be invoked, when should a model be sent to fetch data, etc. - these kinds of shots are all called by the controller.



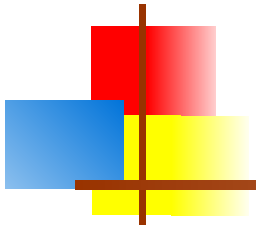
The Controller

- The Controller decides what the model is to do
- Often, the user is put in control by means of a GUI
  - in this case, the GUI and the Controller are often the same
- The Controller and the Model can almost always be separated (what to do versus how to do it)
- The design of the Controller depends on the Model
- The Model should *not* depend on the Controller



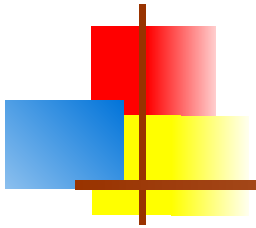
The View

- Typically, the user has to be able to see, or *view*, what the program is doing
- The View shows what the Model is doing
  - The View is a passive observer; it should not affect the model
- The Model should be independent of the View, but (but it can provide access methods)



Combining Controller and View

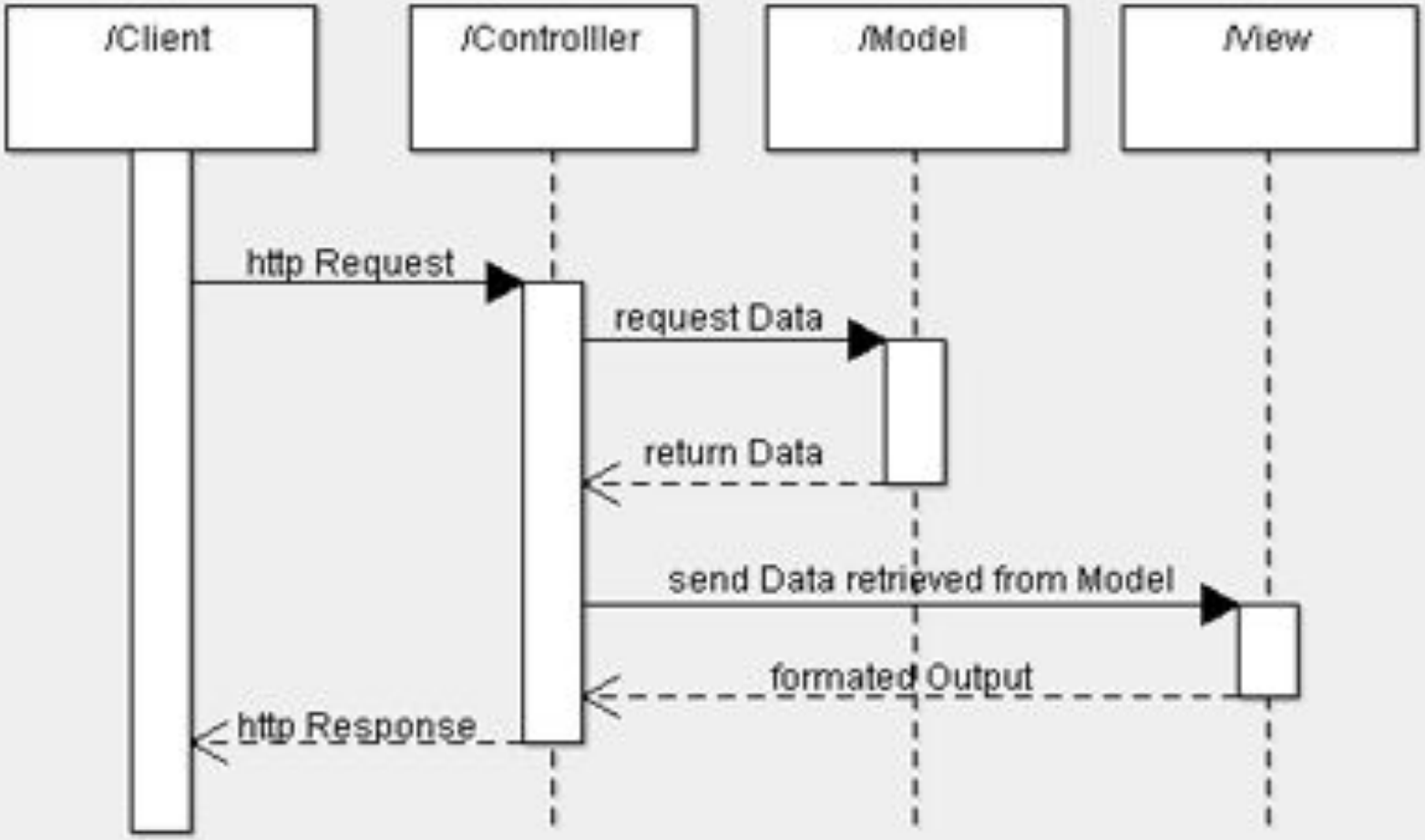
- Sometimes the Controller and View are combined, especially in small programs
- Combining the Controller and View is appropriate if they are very interdependent
- The Model should still be independent
- *Never* mix Model code with GUI code!



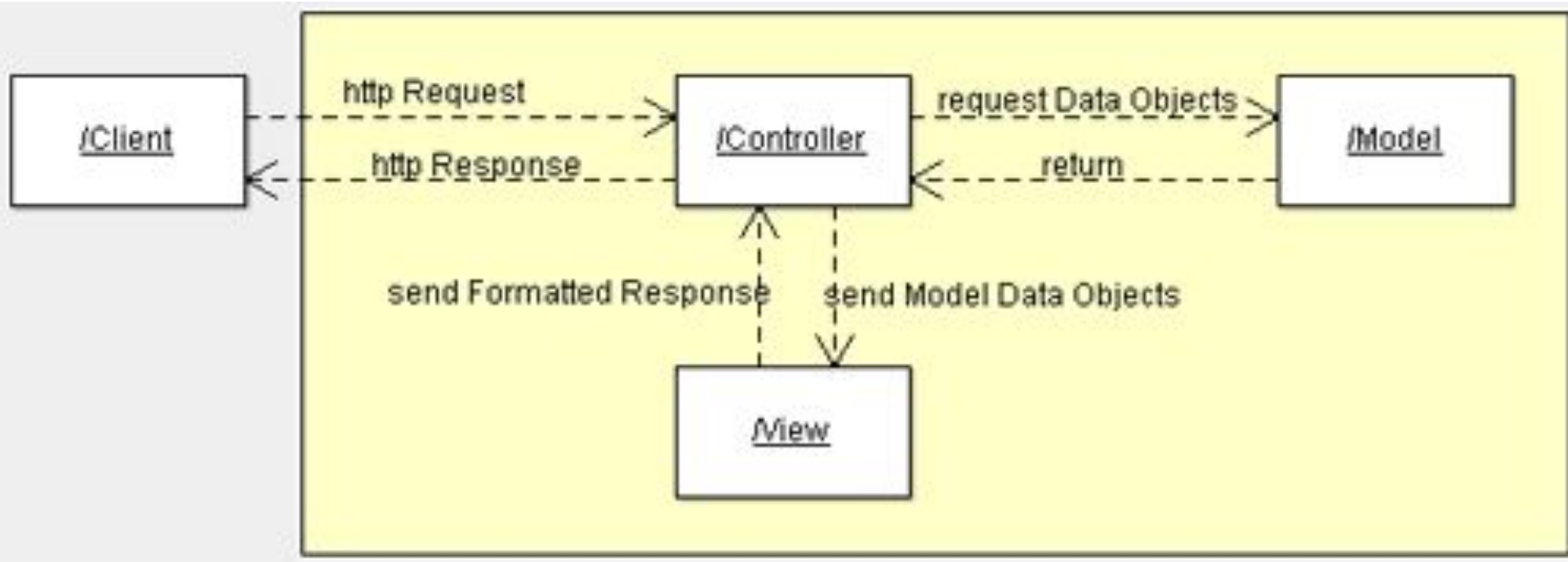
Separation of concerns

- As always, you want code independence
- The Model should not be contaminated with control code or display code
- The View should represent the Model as it really is, not some remembered status
- The Controller should *talk to* the Model and View
  - The Controller can set variables that the Model and View can read

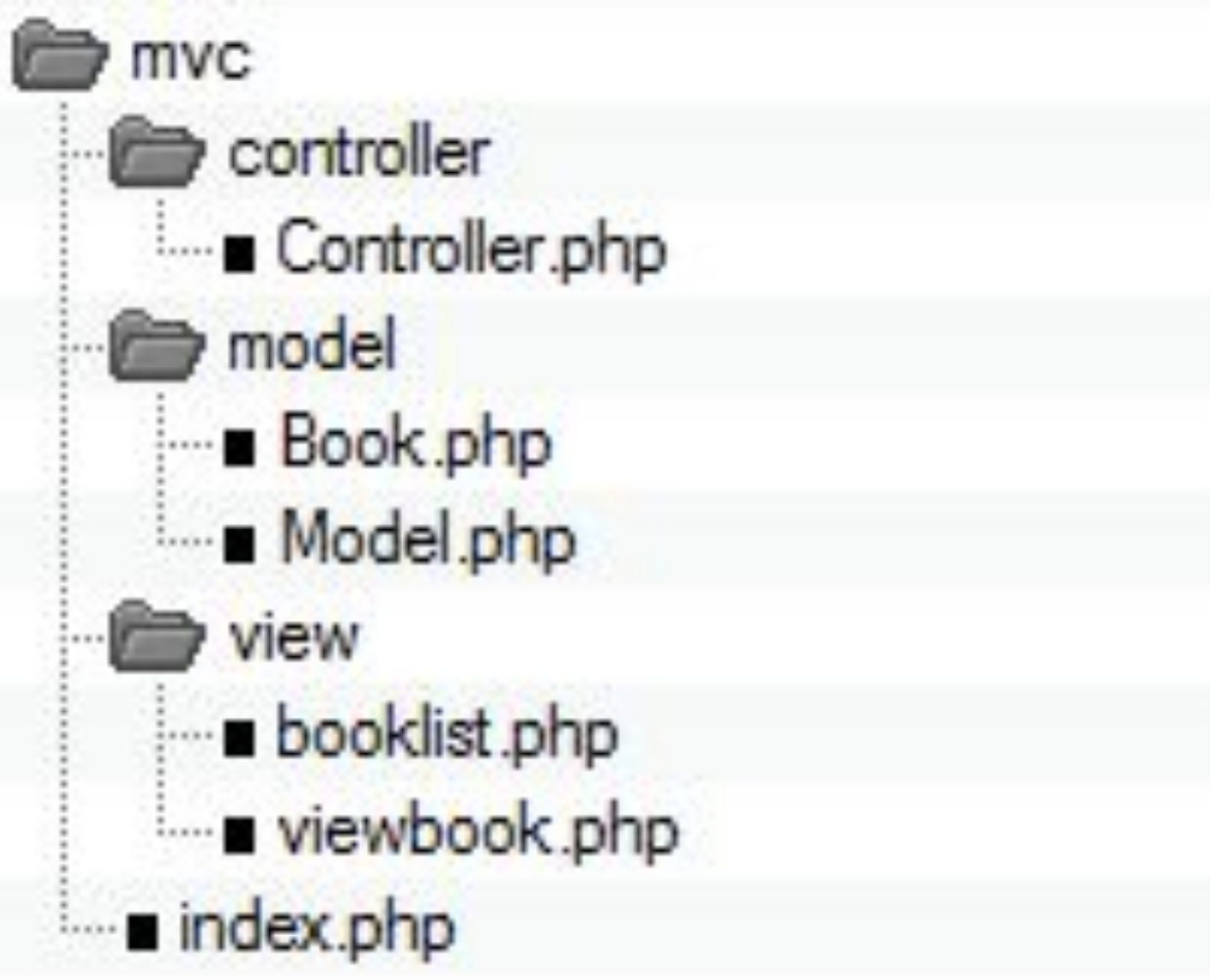
PHP & MVC


/Client
/Controlller
/Model
/View
http Request
request Data
return Data
send Data retrieved from Model
formated Output
http Response

PHP & MVC


/Client
http Request
http Response
/Controller
request Data Objects
return
/Model
send Formatted Response
send Model Data Objects
/View

PHP & MVC


mvc
controller
Controller.php
model
Book.php
Model.php
view
booklist.php
viewbook.php
index.php

Why do we need xampp?

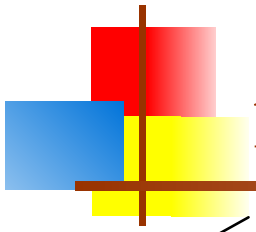
- XAMPP stands for Cross-Platform (X), Apache (A), MariaDB (M), PHP (P) and Perl (P).
- XAMPP is a free and open source cross-platform web server solution stack package.

Why do we need xampp?

- It is a simple, lightweight Apache distribution that makes it extremely easy for developers to create a local web server for testing and deployment purposes.
- XAMPP is also cross-platform, which means it works equally well on Linux, Mac and Windows.
- A development tool, to allow website designers and programmers to test their work on their own computers without any access to the Internet.

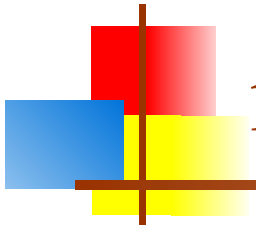
What is localhost?

- In computer networking talk, **localhost** refers to *"this computer"* or even more accurately "the computer I'm working on."
- On almost all networking systems, localhost uses the IP address 127.0.0.1.



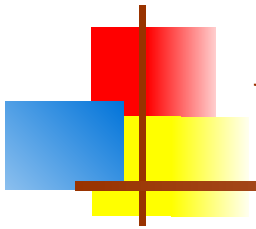
model/Book.php

```
<?php
class Book {
    public $title;
    public $author;
    public $description;

    public function __construct($title,
$author, $description)
    {
        $this->title = $title;
        $this->author = $author;
        $this->description =
$description;
    }
}
?>
```



model/Model.php

```
<?php
include_once("model/Book.php");
class Model {
    public function getBookList()
    {
    return array(
            "Teach Yourself C " => new Book("Teach Yourself C", "H. Schildt", "A boo
on the programming language C"),
            "Computer Networks" => new Book("Computer Networks", "Andrew S.
Tanenbaum", "A book on Computer networks."),
            "PHP for Dummies" => new Book("PHP for Dummies", "Some Smart
Guy", ""));
    }
    public function getBook($title)
    {
        // we use the previous function to get all the books
        // and then we return the requested one. in a real life scenario
//this will be done through a database select command
        $allBooks = $this->getBookList();
        return $allBooks[$title];
    }
}
?>
```

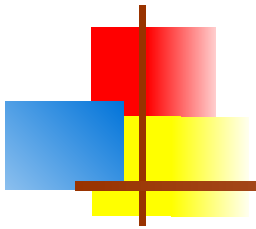


view/viewbook.php

```
<html>
<head></head>
<body>
    <?php

        echo 'Title:' . $book->title . '<br/>';
        echo 'Author:' . $book->author . '<br/>';
        echo 'Description:' . $book->description . '<br/>';

    ?>
</body>
</html>
```

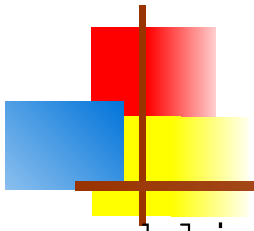
view/booklist.php

```
<html>
<head></head>
<body>
    <table>
        <tbody>

<tr><td>Title</td><td>Author</td><td>Description</td></tr>
        </tbody>
        <?php

            foreach ($books as $book)             {
                echo '<tr><td><a href="index.php?book=' .
                    $book->title . '">' . $book->title .
'</a></td><td>' .
                    $book->author . '</td><td>' .
$book->description . '</td></tr>';
            }
        ?>
    </table>
</body>
</html>
```



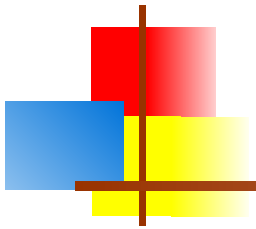
controller/Controller.php

```
<?php
include_once("model/Model.php");

class Controller {
	public $model;

	public function __construct()
	{
		$this->model = new Model();
	}
```



controller/Controller.php

```
 public function invoke()
{
          if (!isset($_GET['book']))
          {
               // no special book is requested, we'll show
a list of all available books
               $books = $this->model->getBookList();
               include 'view/booklist.php';
          }
          else
          {
               // show the requested book
               $book =
$this->model->getBook($_GET['book']);
               include 'view/viewbook.php';
          }
     }
}
?>
```



index.php

```
<?php

// All interaction goes through the index
and is forwarded
// directly to the controller

include_once("controller/Controller.php");

$controller = new Controller();
$controller->invoke();

?>
```