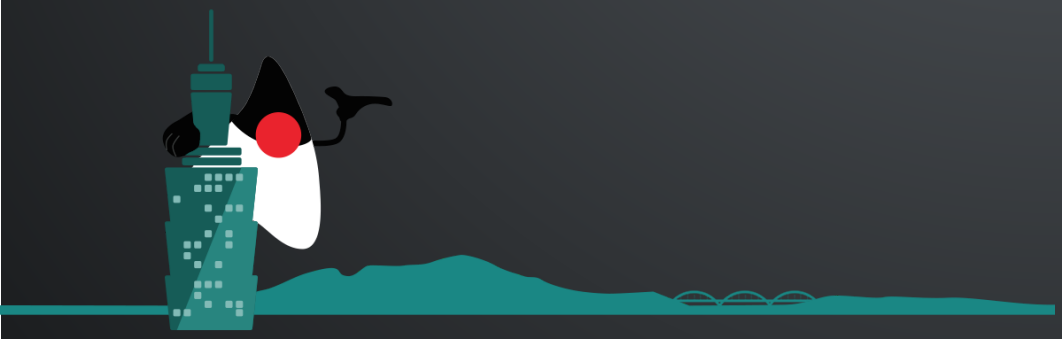
JCConf Taiwan 2014

MODERN DESIGN PATTERN



ABOUT ME

- *王文農 a.k.a Steven*
- ~~Softleader engineer~~, for build application and framework.
- Research interests are adjusted efficiency and effectiveness of algorithms.
- Play some of new frameworks/libraries and technology.

If you want to be a cool guy.

JOIN SOFTLEADER.

WHAT IS DESIGN PATTERN?

Descriptions of communicating objects and classes that are customized to solve a general design problem.

–Gamma, et. al

THE ONE CONSTANT IN SOFTWARE DEVELOPMENT:

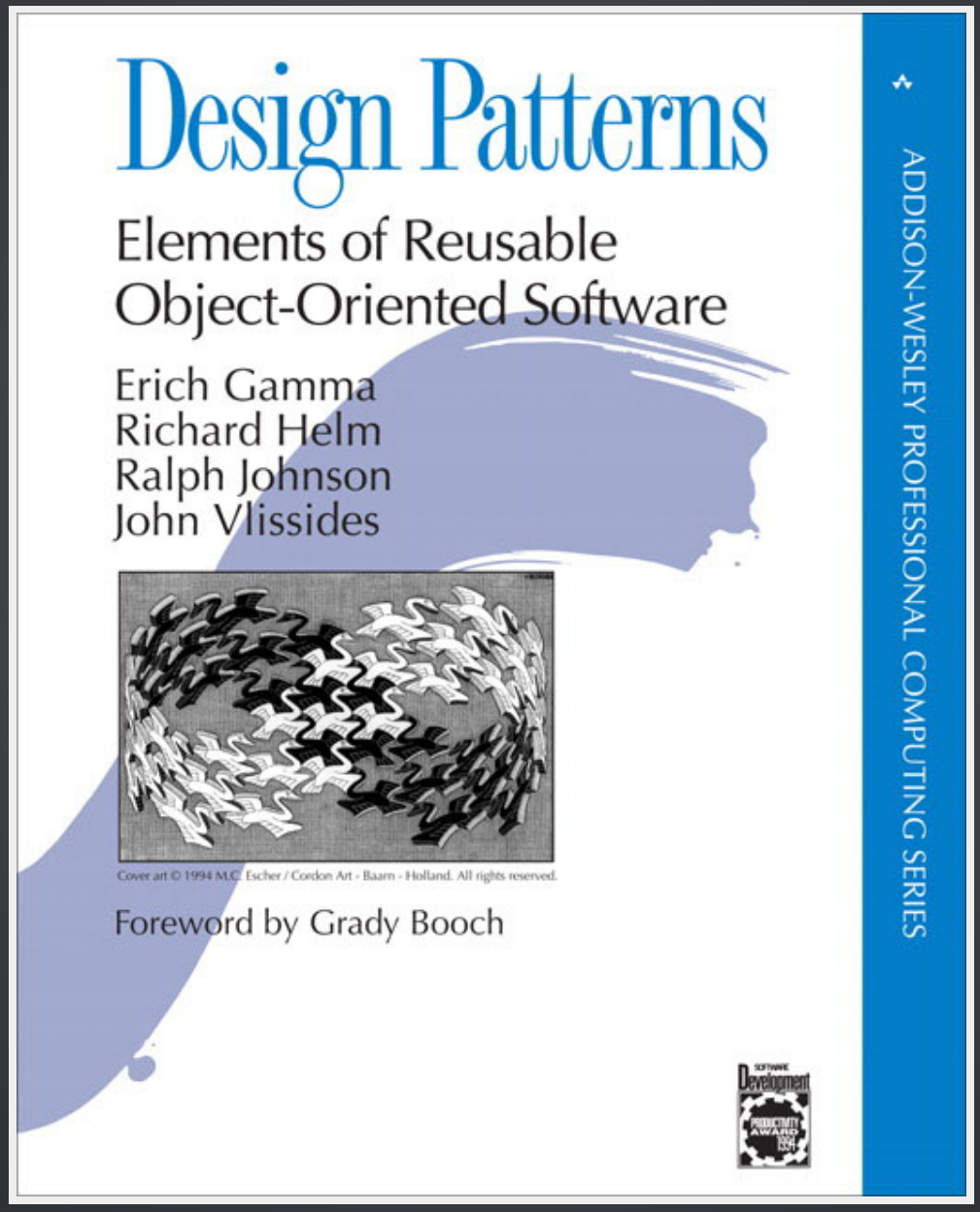
CHANGE!

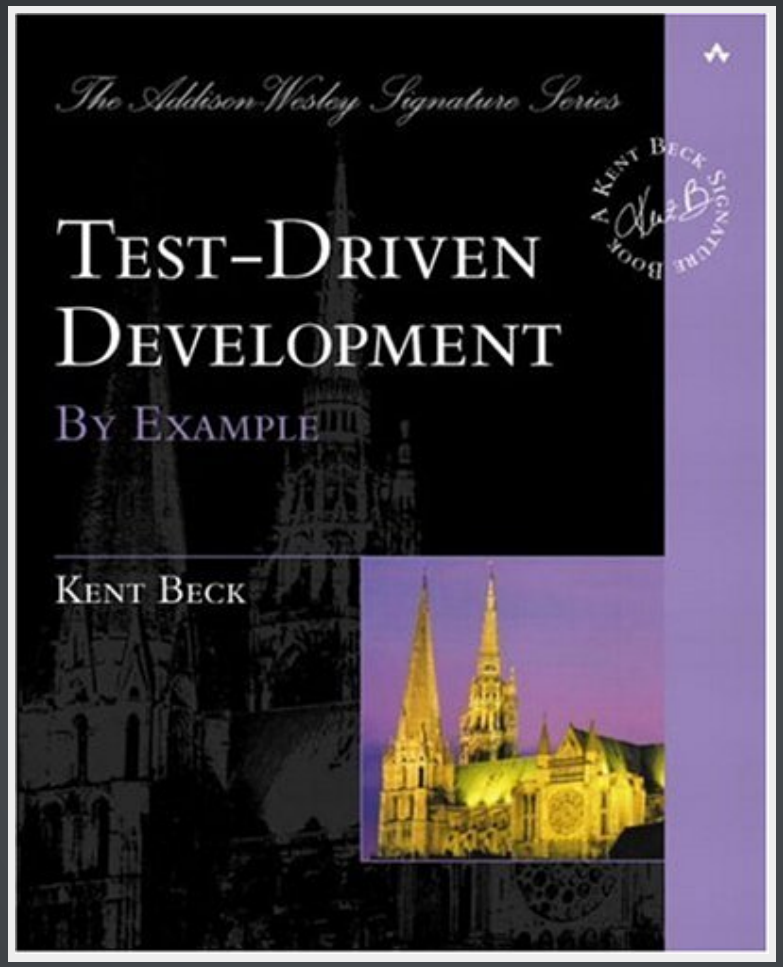
TIGHT COUPLING!

IT MIGHT GET YOU INTO TROUBLE

BEWARE OF

GOF: 23


Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES


The Addison Wesley Signature Series
A Kent Beck Signature Book
Test-Driven Development
By Example
Kent Beck

MVC PATTERN

A PROBLEM OR A SOLUTION ?

Commonly reasons:

- Reduced Code complexity
- Code reuse
- Increased flexibility
- Decoupled code

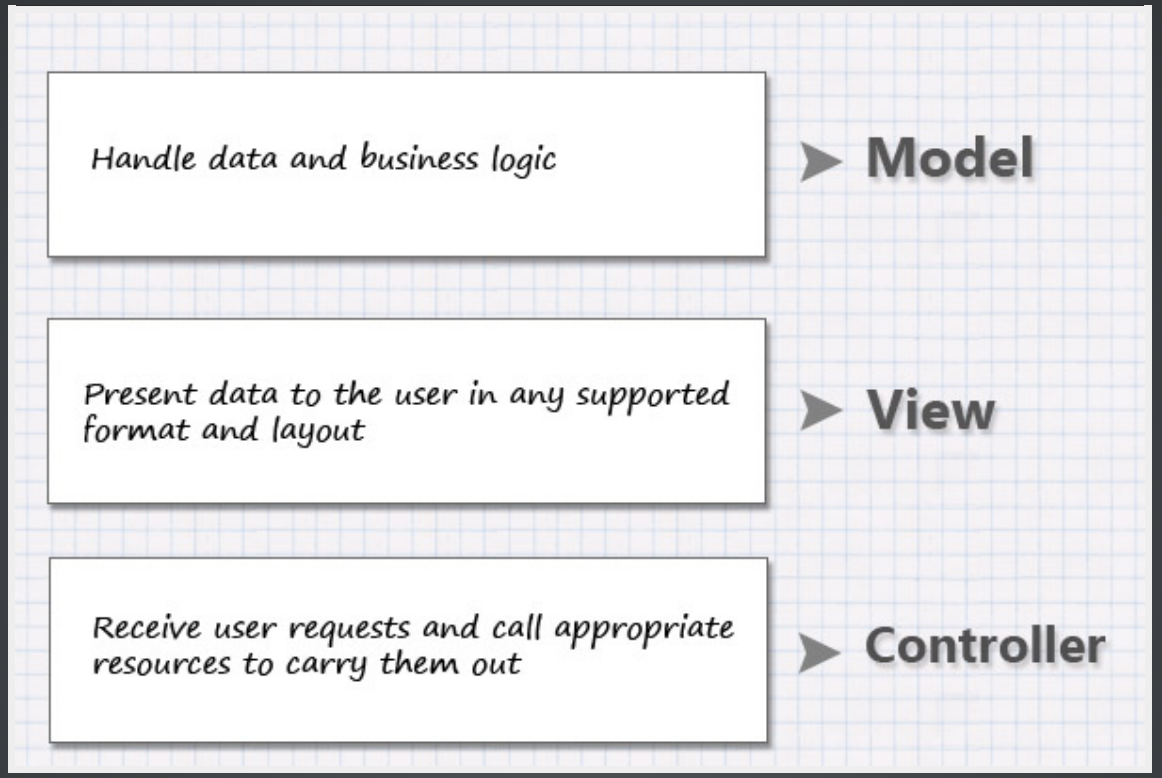
COOL, SOUNDS NICE, BUT

- Is it true ?
- Do all other patterns lack these cool things ?

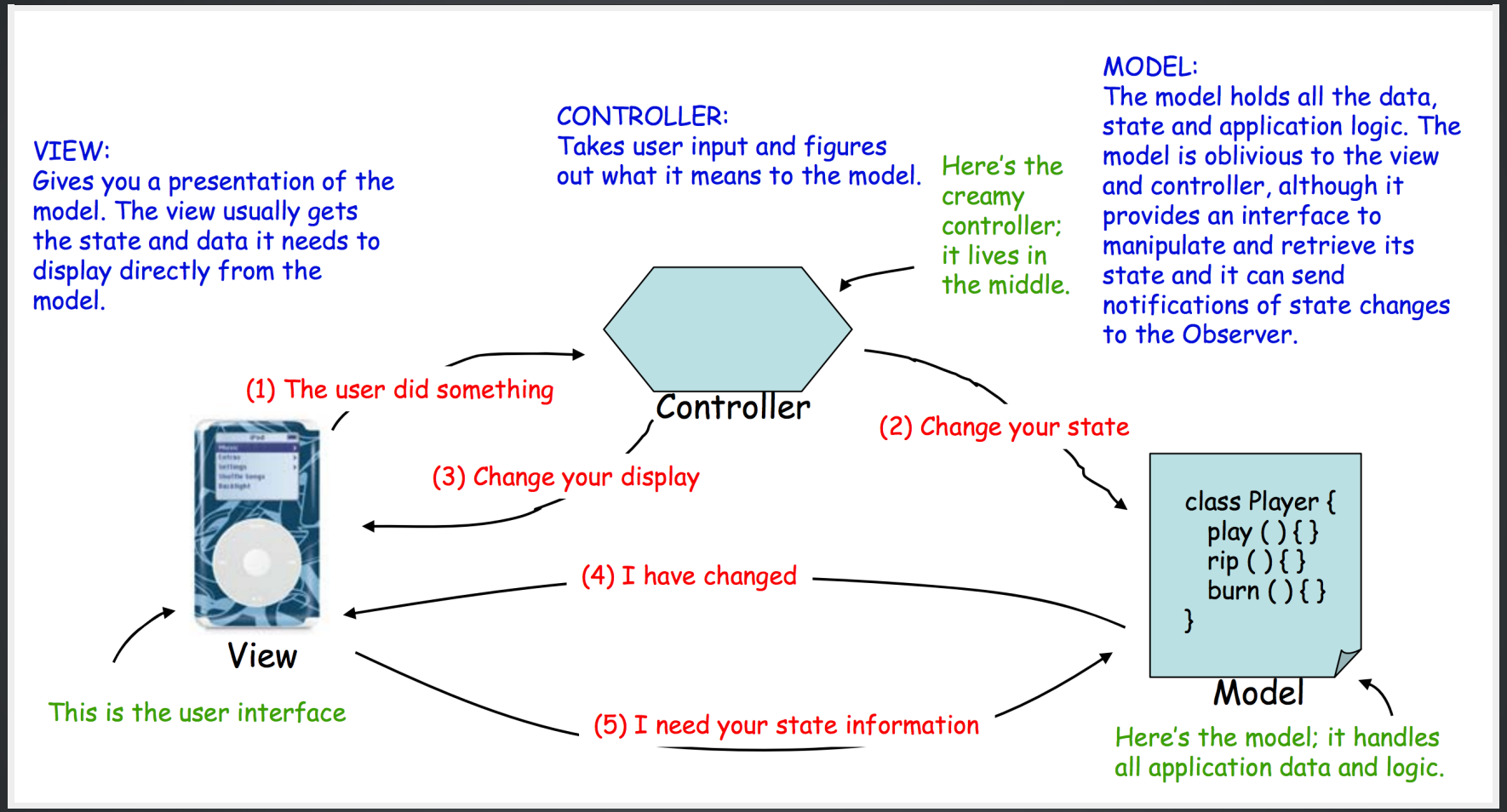
THE ANSWER IS NO!

- Doesn't solve the code *complexity* problem
- Doesn't solve the code *reuse* or *no-flexibility* problem
- Doesn't guarantee *decoupled* code

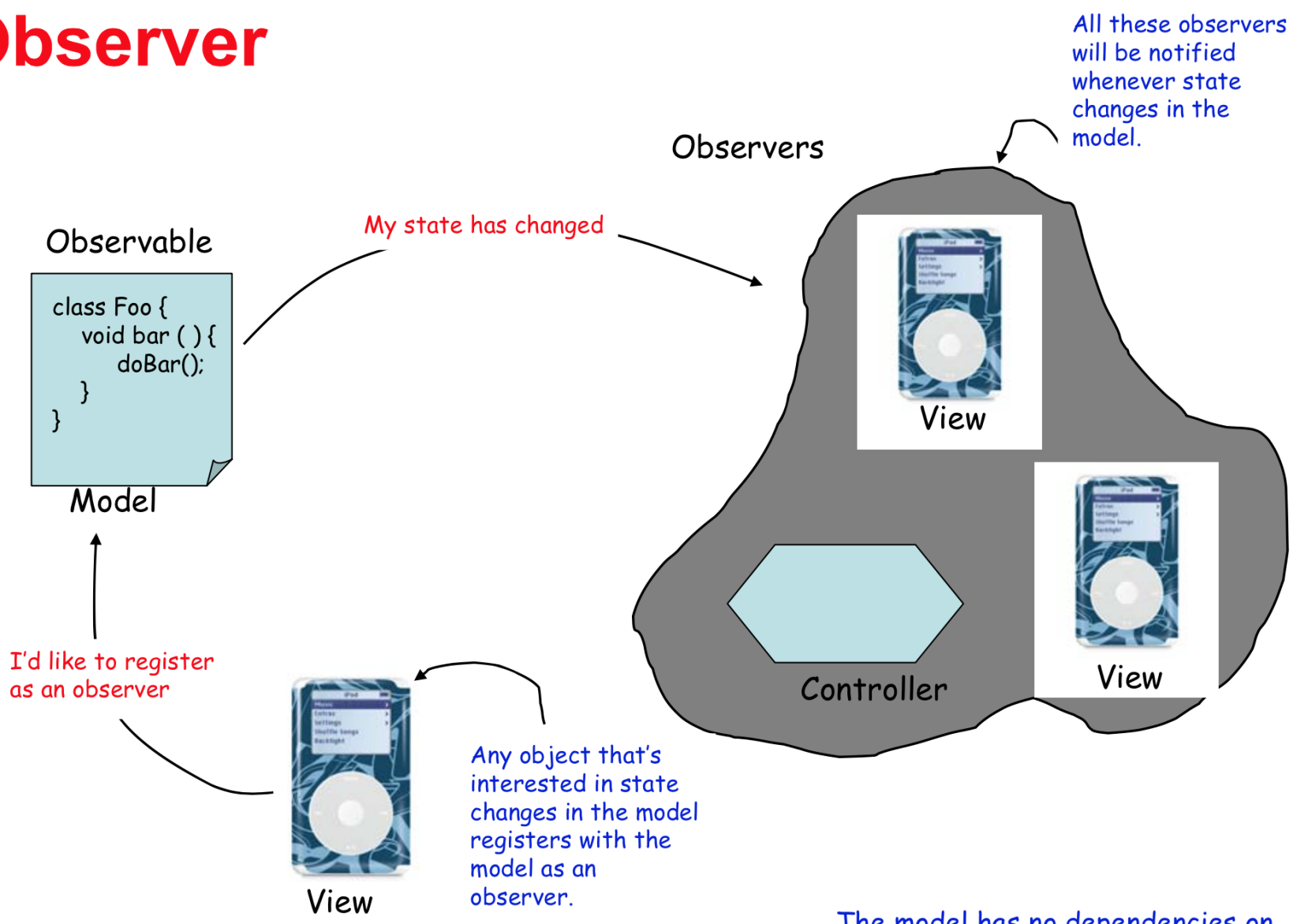
MVC COMPONENTS



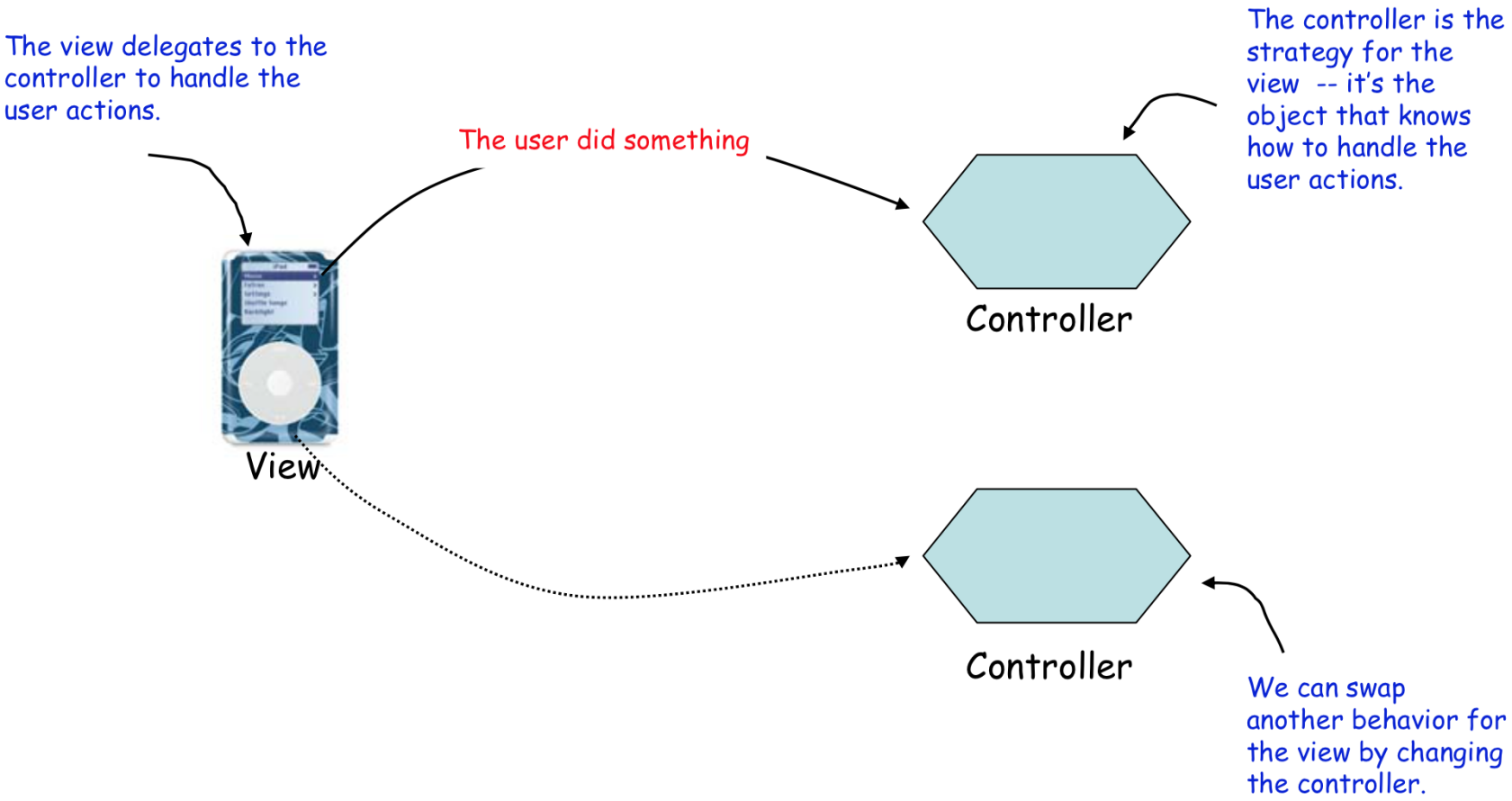
HOW MVC WORKS



Observer

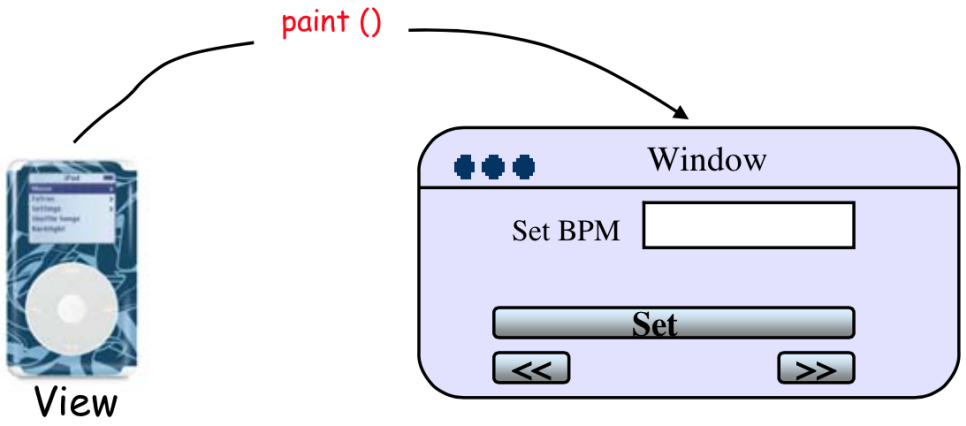

All these observers will be notified whenever state changes in the model.
Observers
Observable
My state has changed
class Foo {
void bar ( ) {
doBar();
}
}
Model
View
Controller
View
I'd like to register as an observer
View
Any object that's interested in state changes in the model registers with the model as an observer.
The model has no dependencies on viewers or controllers!

Strategy



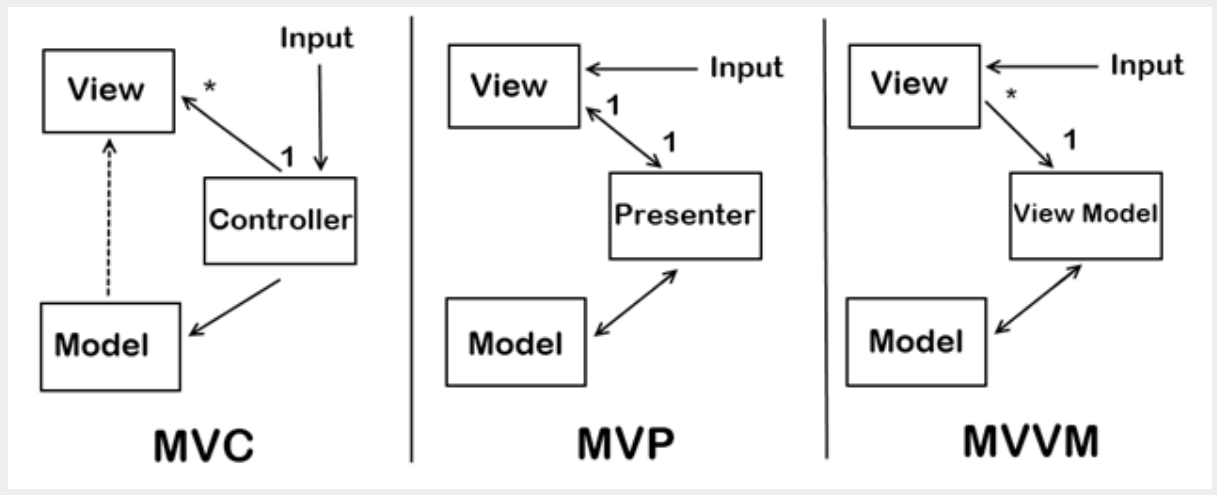
The view only worries about presentation, the controller worries about translating user input to actions on the model.

Composite



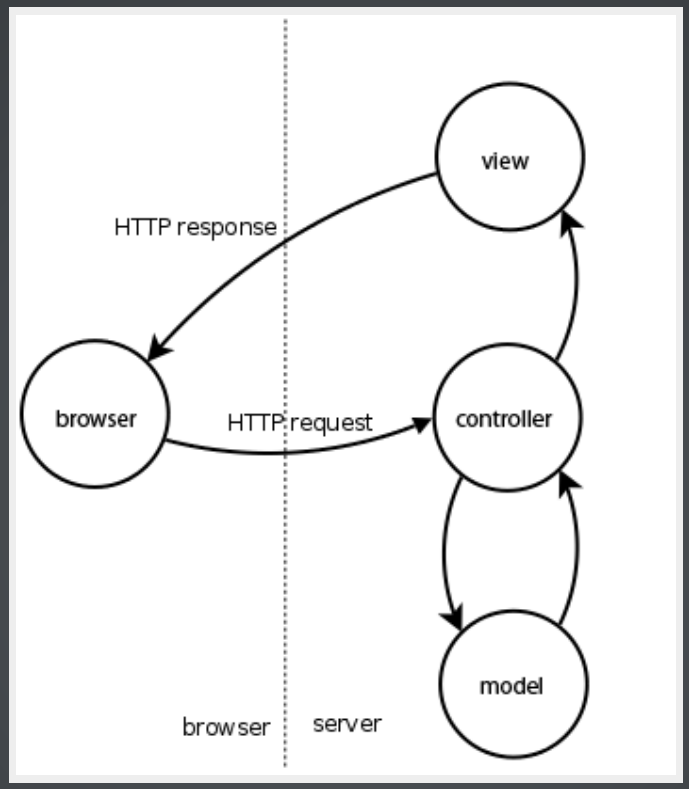
The view is a composite of GUI components (labels, buttons, text entry, etc.). The top level component contains other components, which contain other components, and so on until you get to the leaf nodes.

POOR MVC GIVE BIRTH TO



MVC DEMO WITH SERVLET

MODEL-VIEW-CONTROLLER



MODEL

```
public class Model {

    private String name;

    public void setName(String name) {
        // business logic
        this.name = name;
    }

    // for view access
    public String getName() {
        return name;
    }
}
```

VIEW

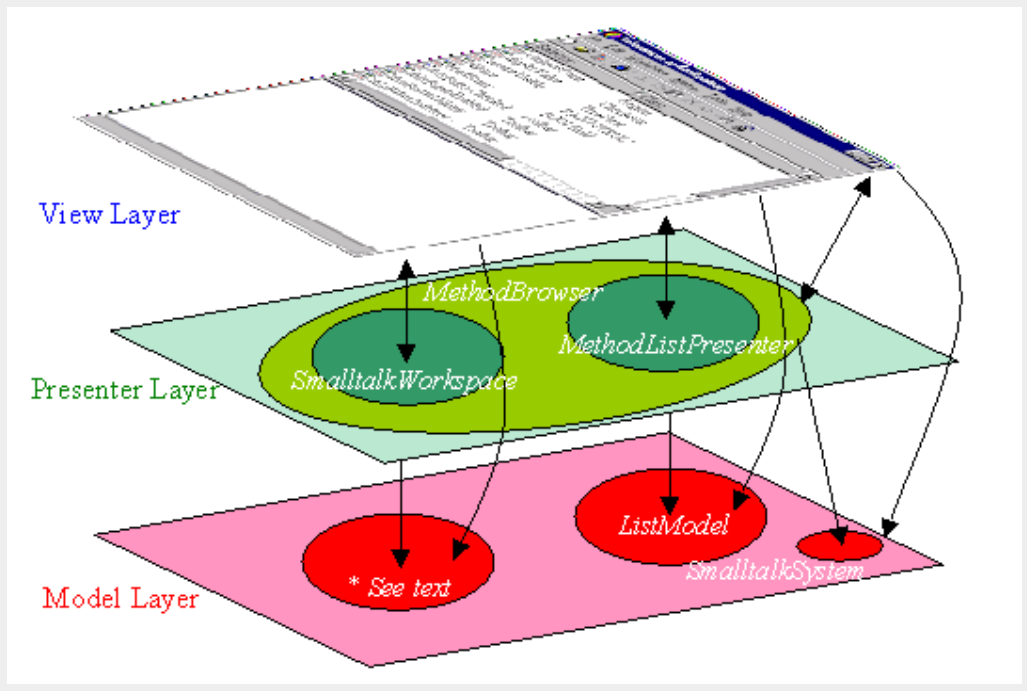
```
<%-- receive data --%>
<h1>Hello, ${model.name}</h1>

<form action="controller" method="post">
    <label for="name">My name is </label>
    <input type="text" id="name" name="name" value="${model.name}" />
    <button>Submit</button>
</form>
```

CONTROLLER

```
@WebServlet(value = "/controller")
public class Controller extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) th
        // accepting request
        String name = req.getParameter("name");

        // validate request
        if (name.isEmpty()) {
            name = "Guest";
        }

        // use Model to do business logic
        Model model = new Model();
        model.setName(name);

        // binding model to view
```

MVP DEMO WITH VAADIN

MODEL-VIEW-PRESENTER



MODEL

```
public class Model {

    private String name;

    public void setName(String name) {
        // business logic
        this.name = name;
    }

    // for presenter access
    public String getName() {
        return name;
    }
}
```

VIEW

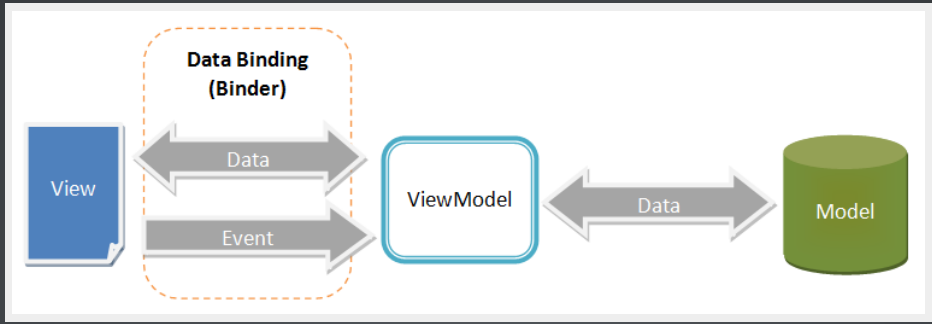
```
public class View extends CustomComponent {

    private Label display = new Label("Hello, Guest");
    private TextField name = new TextField("My name is ");
    private Button button = new Button("Submit");

    private Presenter presenter;

    public View() {
        // initial layout and presenter
        initialLayout();
        presenter = new Presenter(this);
    }

    protected void initialLayout() {
        CustomLayout layout = new CustomLayout("view");
        layout.addComponent(display, "display");
        layout.addComponent(name, "name");
```

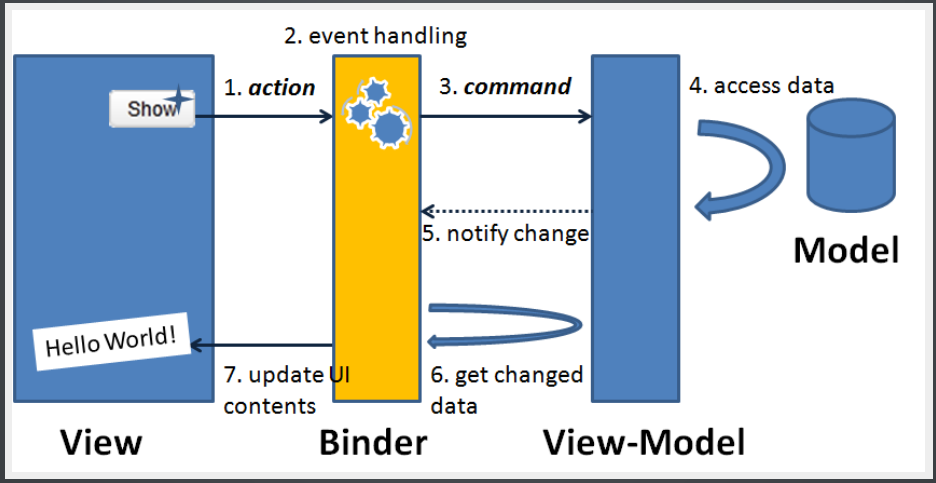
PRESENTER

```
public class Presenter {

    private View view;
    private Model model;

    public Presenter(View view) {
        this.view = view;
        this.model = new Model();
    }

    public void changeDisplay() {
        String name = view.getInput().getValue();

        // do some logic, eg: validate
        if (name.isEmpty()) {
            name = "Guest";
        }
```

MVVM DEMO WITH ZK

MODEL-VIEW-VIEWMODEL





MODEL

```
public class Model {

    private String name;

    public void setName(String name) {
        // business logic
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

VIEW

```
<zk>
    <window title="MVVM Example" border="normal" width="100%"
        apply="org.zkoss.bind.BindComposer"
        viewModel="@id('vm') @init('tw.jcconf.example.mvvm.vm.ViewModel')">
        <hbox>
            Hello, <label value="@load(vm.model.name)" />
        </hbox>
        <hbox>
            <label value="My name is " /><textbox value="@bind(vm.name)" />
            <button onClick="@command('changeName')" label="Submit" />
        </hbox>
    </window>
</zk>
```

VIEWMODEL

```
public class ViewModel {

    private String name;
    private Model model;

    public ViewModel() {
        model = new Model();
        model.setName(name = "Guest");
    }

    // event handler
    @Command
    @NotifyChange("model")
    public void changeName() throws Exception {
        model.setName(name);
    }

    // ViewModel provider getter & setter
```

RESTFUL PATTERN

An architectural style that abstracts the architectural elements within a distributed hypermedia system.

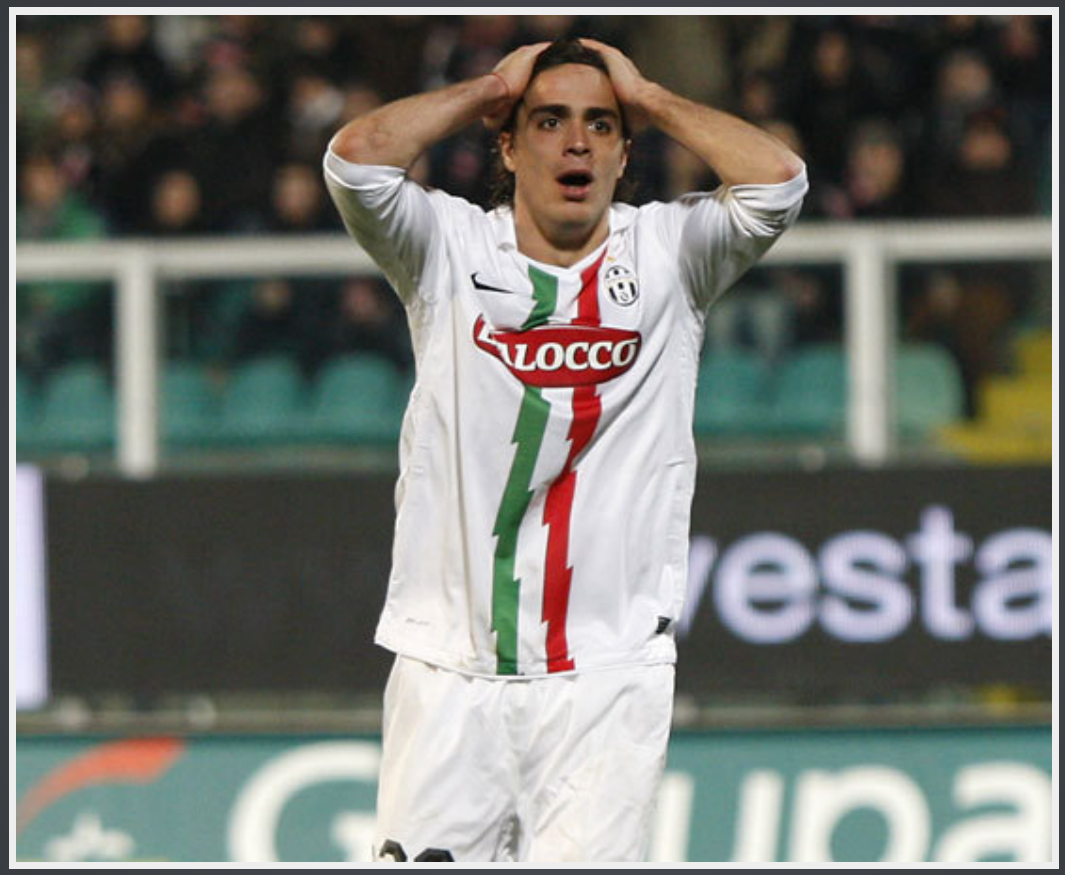
–Wikipedia

THE MEAS IS

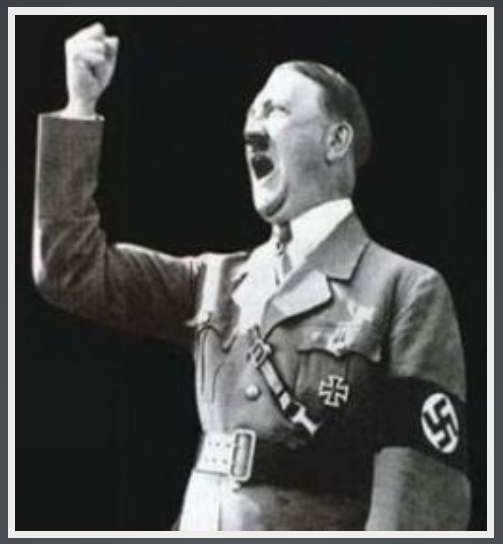
- *Use HTTP methods explicitly.*
- *Be stateless.*
- *Expose directory structure-like URIs.*
- *Transfer XML, JavaScript Object Notation (JSON), or both.*

–IBM's developerWorks website

WHAT IS REST ?



HTTP ITSELF IS REST IMELEMENTATION



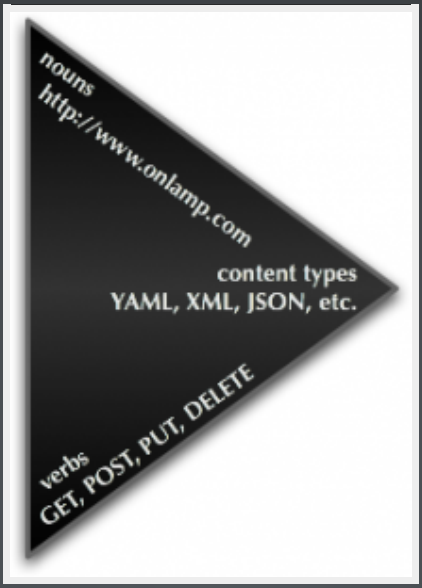
HOW TO USE ?

User Action	HTTP Method
Create	POST/PUT
Read (Retrieve)	GET
Update (Modify)	PUT/PATCH
Delete (Destroy)	DELETE

LOOK ALIKE

User Action	SQL
Create	INSERT
Read (Retrieve)	SELECT
Update (Modify)	UPDATE
Delete (Destroy)	DELETE

FOLLOW HTTP TO DESIGN RESTFUL WEB SERVICE



- Nouns
  Define the URL for your resources
- Verbs
  Choose the right HTTP method
- Content Types
  Choose content-type which you want to return


Java
2011
紅色魅力，活力Java！
JavaTwo
RESTful Services Using JAX-RS and Jersey
蘇國鈞/Monster Su
ORACLE

« ‹ › »

CREATE

POST *http://domain/books*

```
{
    "isbn": "123456789",
    "title": "Modern Design Pattern",
    "author": "Steven Wang",
    "publisher": "JCConf TW",
    "year": "2014"
}
```

HTTP/1.1 *200* | *401*

```
{
    "id": "9876543"
}
```

READ

GET *http://domain/books/123456789*

or

GET *http://domain/books?isbn=123456789*

HTTP/1.1 *200 | 404*

```
[{
    "id": "9876543",
    "isbn": "123456789",
    "title": "Modern Design Pattern",
    "author": "Steven Wang",
    "publisher": "JCConf TW",
    "year": "2014"
}]
```

UPDATE

PUT *http://domain/books/9876543*

or

PUT *http://domain/books*

```
{
    ["id": "9876543",]
    "isbn": "123456789",,
    "title": "Modern Design Pattern",
    "author": "Steven Wang",
    "publisher": "JCConf TW",
    "year": "2014"
}
```

HTTP/1.1 *200 | 401 | 404*

DELETE

DELETE *http://domain/books/9876543*

or

DELETE *http://domain/books?isbn=9876543*

HTTP/1.1 *200| 401 |404*

RESTFUL DEMO WITH SPRING

```java
@RestController
@RequestMapping("/books")
public class BooksResource {

    @Autowired
    BookService service;

    @RequestMapping(consumes = "application/json", produces = "application/jso
    @ResponseBody
    public List<Book> getAll() {
        return service.getAll();
    }

    @RequestMapping(value = "/{isbn}", consumes = "application/json", produces
    @ResponseBody
    public Book getByIsbn(@PathVariable("isbn") String isbn) {
        return service.getBookByIsbn(isbn);
    }
```

CQRS PATTERN

WHAT IS CQRS ?

CQRS is simply the creation of two objects where there was previously only one. The separation occurs based upon whether the methods are a command or a query (the same definition that is used by Meyer in Command and Query Separation, a command is any method that mutates state and a query is any method that returns a value).

–Wikipedia

Say what?

PUT ANOTHER WAY...

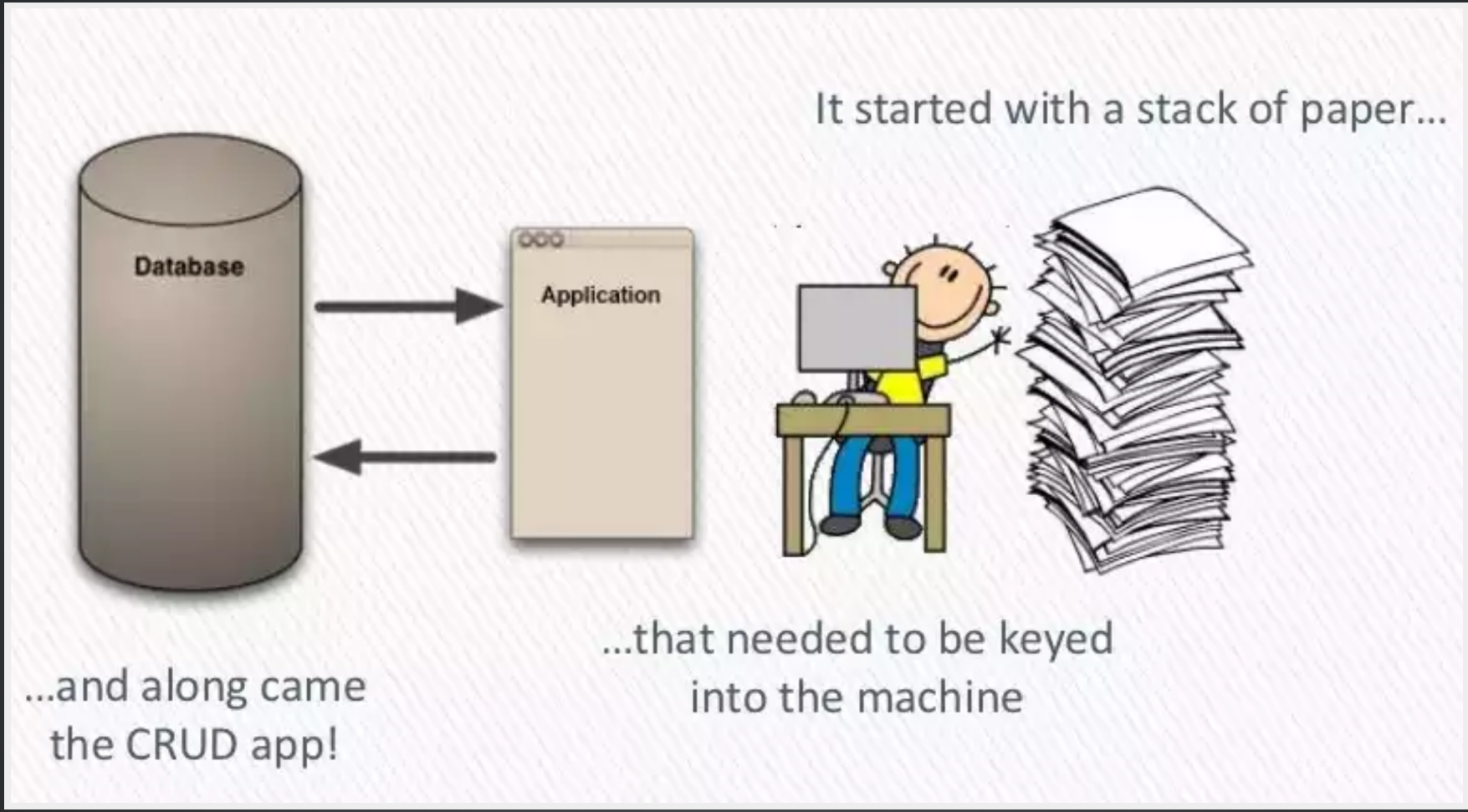
Command/Query Responsibility Segregation (CQRS) is the idea that you can use a different model to update information than the model you use to read information.

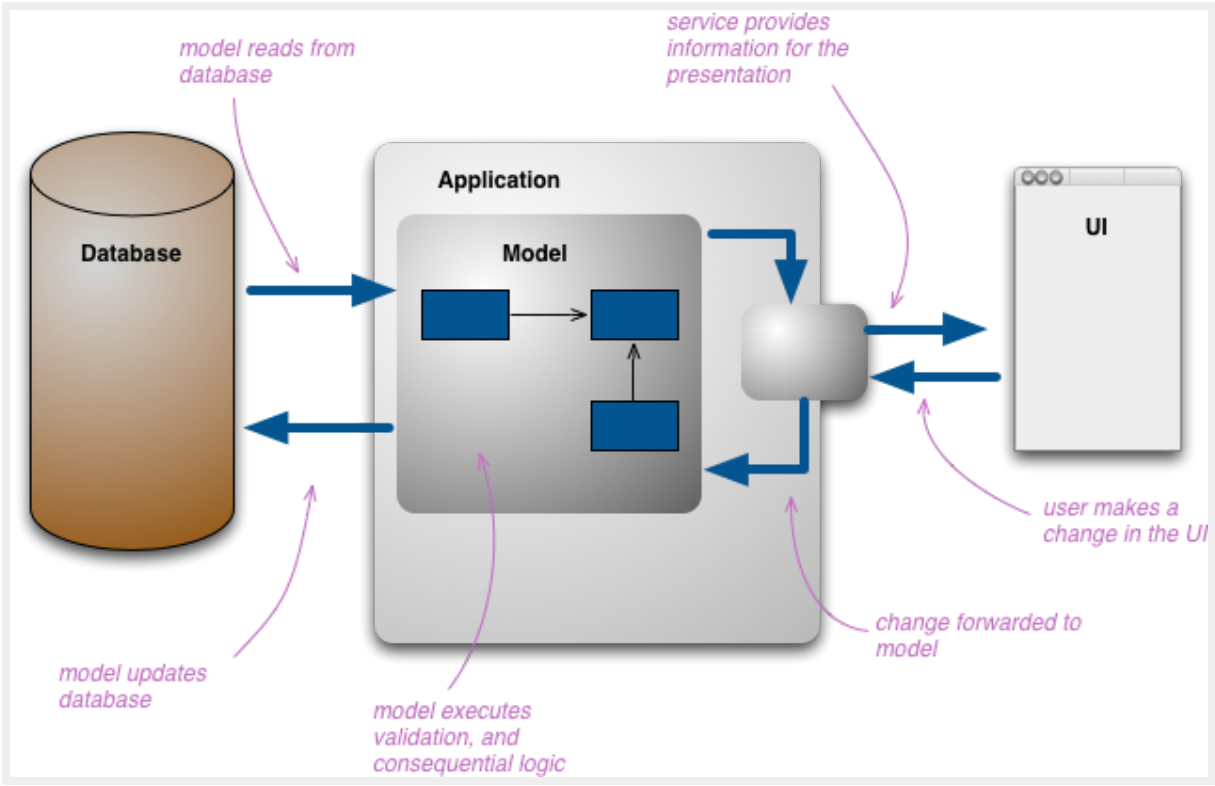
IN THIS CONTEXT,

- Commands = Writes
- Queries = Reads

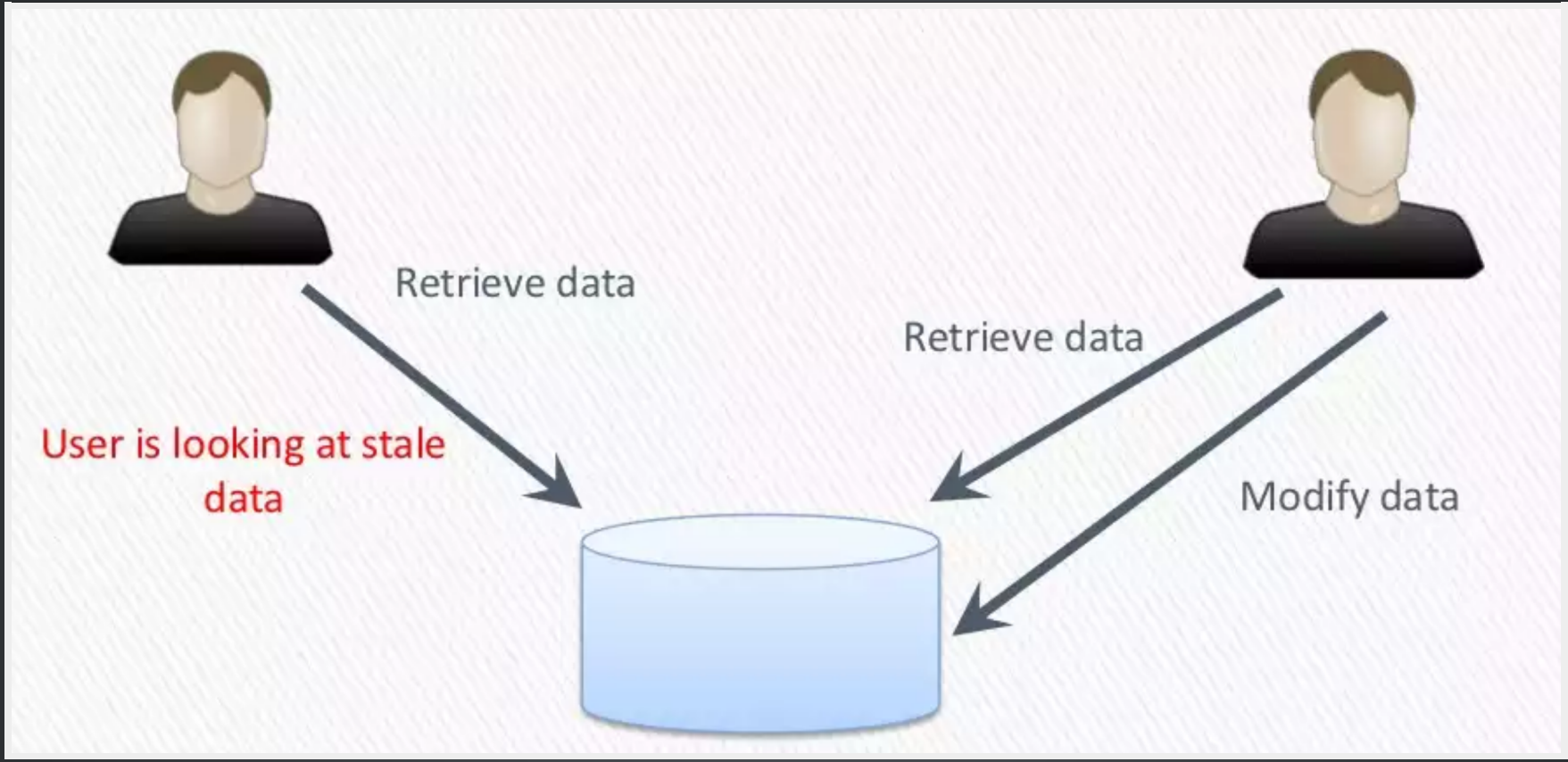
OK, BUT WHY ?

WHAT IS CQRS NEED ?

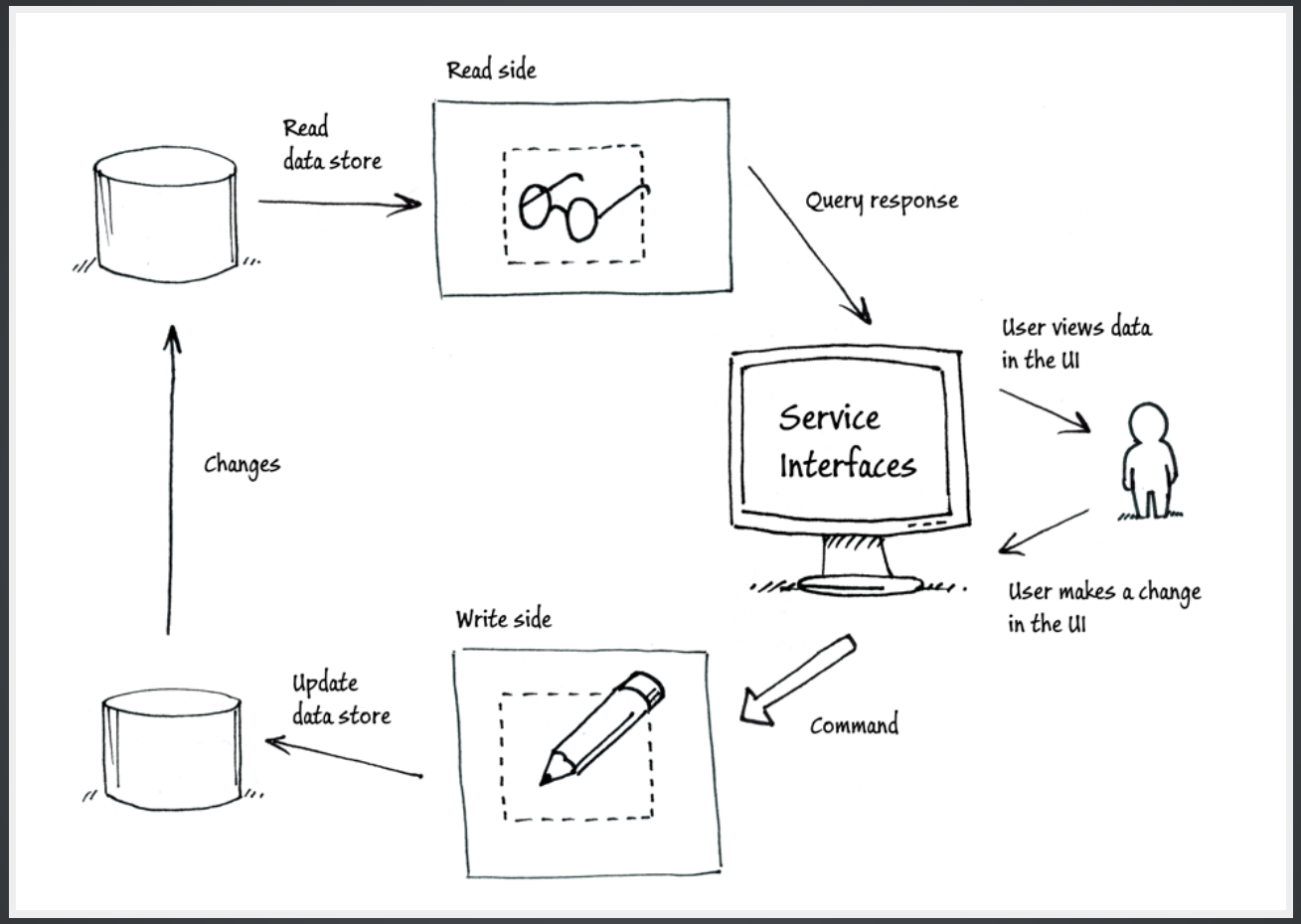



model reads from database
service provides information for the presentation
Application
Model
Database
UI
user makes a change in the UI
change forwarded to model
model updates database
model executes validation, and consequential logic

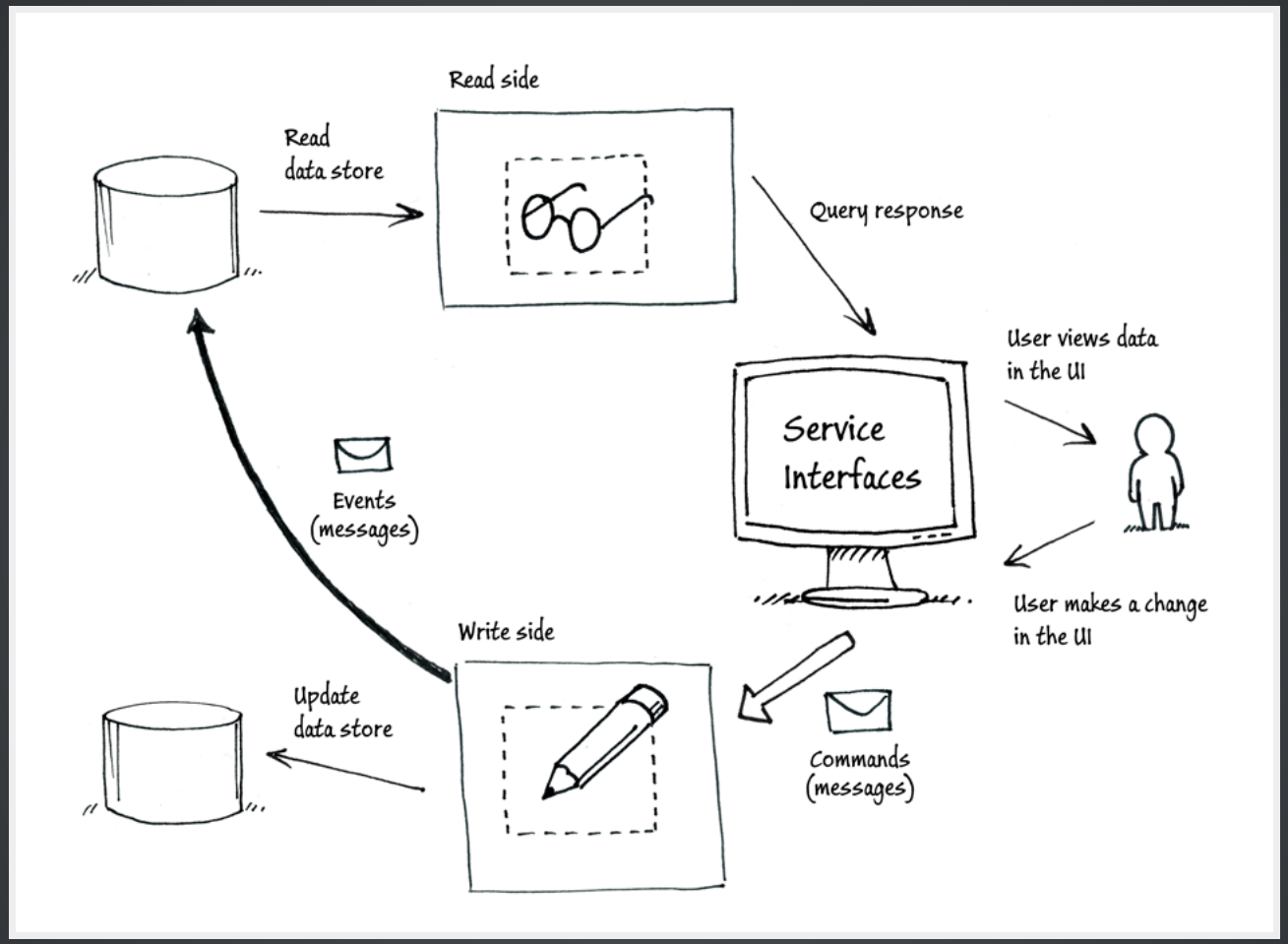
MULTI-USER SYSTEM



TYPICAL APPLICATION OF THE CQRS PATTERN



COMMANDS AND EVENTS IN THE CQRS PATTERN



WHEN TO AVOID CQRS ?

> *So, when should you avoid CQRS?*
>
> *The answer is most of the time...*

–Udi Dahan

OK, SOME SUGGESTIONS

- Large team.
- Difficult business logic.
- Scalability matters.
- Your top people can work on domain logic.

FINALLY

PATTERNS *DON'T* SOLVE PROBLEMS,
THEY *HELP* US


Q&A