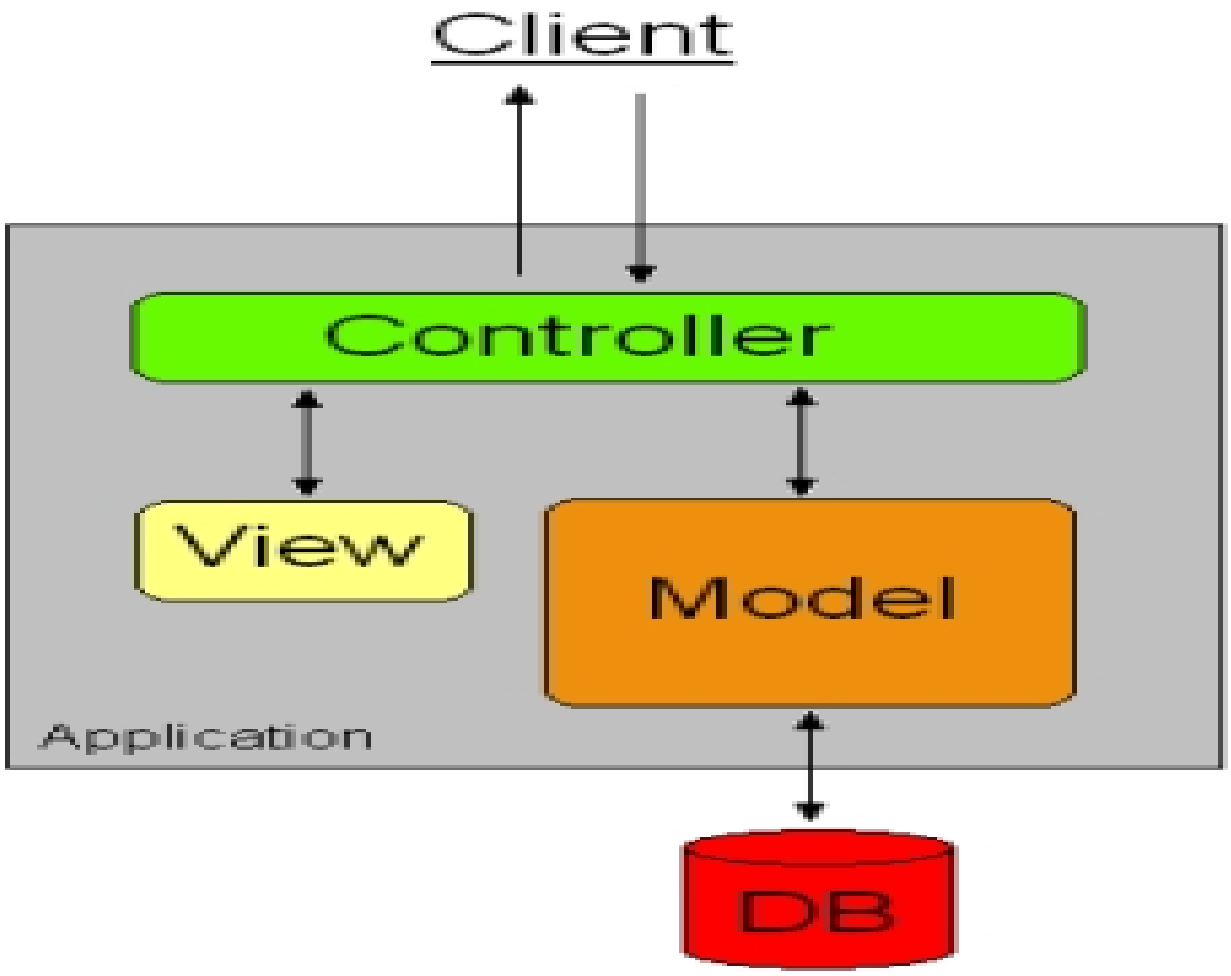
Model-View-Controller

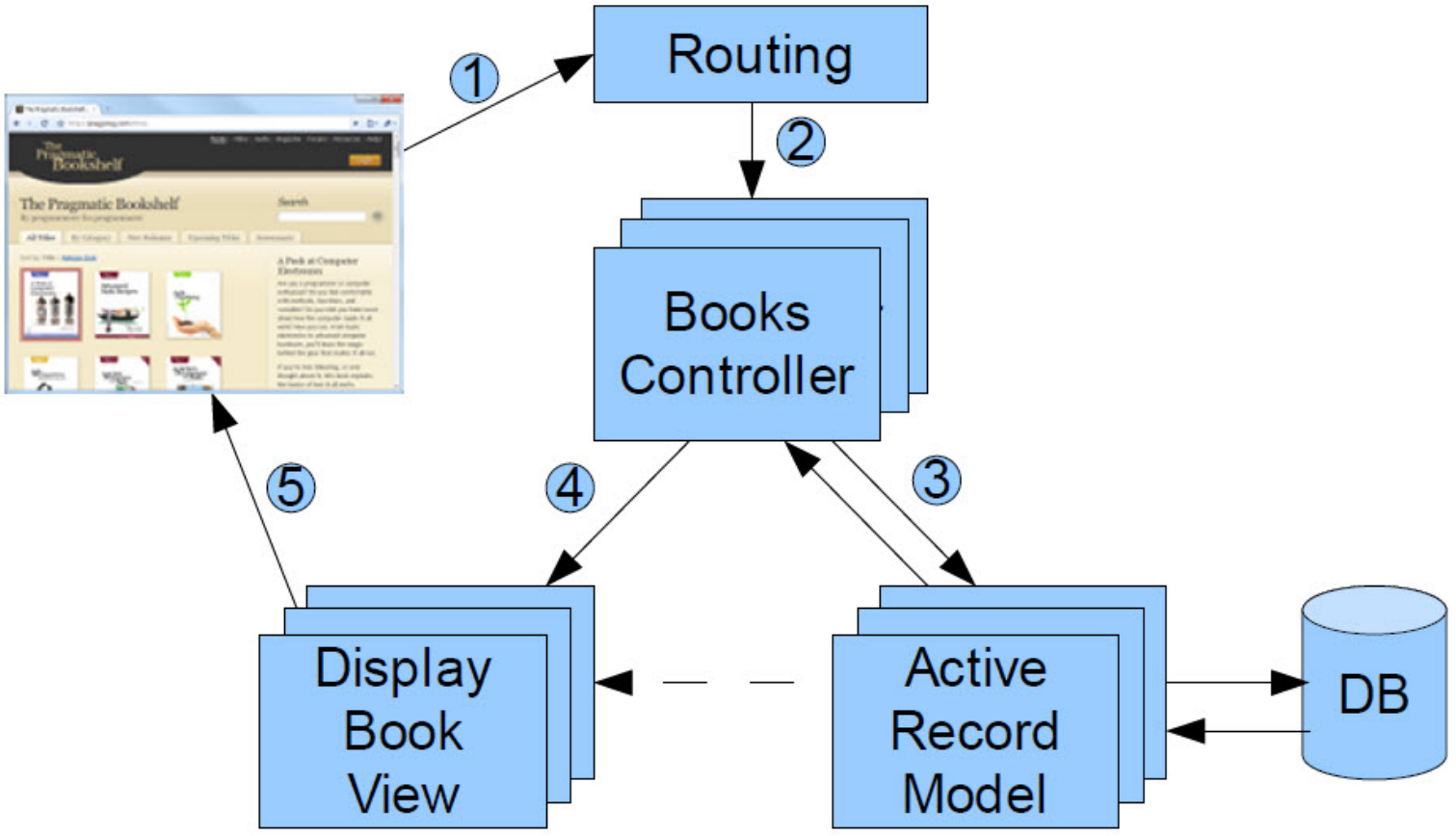
Model-View-Controller

- **MODEL–VIEW–CONTROLLER** (**MVC**) is a **SOFTWARE ARCHITECTURE**, considered as an **ARCHITECTURAL PATTERN** used in **SOFTWARE ENGINEERING**
- It isolates “**DOMAIN LOGIC**" (the **APPLICATION LOGIC** for the **USER**) from the **USER INTERFACE** (**INPUT** and **PRESENTATION)**, **TESTING** and **MAINTENANCE** of each **SEPARATION OF CONCERNS**
- **MODEL–VIEW–CONTROLLER** (**MVC**) pattern **CREATES APPLICATIONS** that separate the different aspects of the application (**INPUT LOGIC**, **BUSINESS LOGIC**, and **USER INTERFACE LOGIC**), while **PROVIDING** a **LOOSE COUPLING** between these elements

Model-View-Controller



Model-View-Controller



Model-View-Controller

Why use MVC?

- In the early days of the web (and even up to now), many web applications were written with all of the processing logic (business processing, routing, rendering) concentrated in a single location. This approach didn't scale well for some reasons:
- Maintenance was hard. Lack of structure meant that the dependencies between modules aren't clear. Changing one part of the system might affect another part of the system without a programmer knowing it.
- Debugging was hard. As all of the processing logic were contained in single files, a programmer hunting for the code that caused a bug would have to scan through a lot of code which has nothing to do with the bug itself before finding it.
- MVC addresses these problems. First off, the structure provides a degree of isolation between modules. Sure, changing a Model might still affect a lot of programs in the system, but at least the extent of the changes can be easily predicted.

Model-View-Controller

- AN ARCHITECTURAL PATTERN USED IN SOFTWARE ENGINEERING
- IT SEPARATES THE DATA (MODEL) AND THE USER INTERFACE (VIEW) CONCERNS
- CHANGES IN THE USER INTERFACE DO NOT AFFECT DATA HANDLING
- IT USES AN INTERMEDIATE COMPONENT CALLED THE CONTROLLER
- IT SEPARATES DATA ACCESS AND BUSINESS LOGIC FROM DATA PRESENTATION AND USER INTERACTION USING THE CONTROLLER
- THE LAYERS INVOLVED ARE
  - USER INTERFACE
  - INFORMATION
  - DATA ACCESS

Model-View-Controller

- MVC IS USED IN VARIOUS LANGUAGES LIKE
  - .NET
  - ASP.NET
  - WINDOWS FORM
  - JAVA
  - RUBY
  - PYTHON (DJANGO, TURBOGEARS)
  - PERL
  - PHP
  - COLDFUSION
- SUGGESTED IN WEB APPLICATIONS INFRASTRUCTURE

Model-View-Controller

- THE MODEL IS THE INFORMATION REPRESENTATION ON WHICH THE APPLCIATION OPERATES
  - THE MODEL CONTAINS PROCESSING LOGIC
  - ACCESS TO INFORMATION SOURCES
  - ALL APPLICATION SPECIFIC CONTENT
- THE VIEW RENDERS THE MODEL SUITABLE FOR INTERACTION
  - CONTAINS ALL INTERFACE SPECIFIC FUNCTIONS
  - ENABLER OF CONTENT PRESENTATION AND PROCESSING LOGIC
  - CONTAINS ALL PROCESSING FUNCTIONALITY
- THE CONTROLLER PROCESSES AND RESPONDS TO EVENTS LIKE USER ACTIONS
  - COORDINATES FLOW OF DATA BETWEEN THE MODEL AND VIEW

Model-View-Controller

- Though **MVC** comes in different flavors, **CONTROL FLOW** is generally as follows:

1. The **USER INTERACTS** with the **USER INTERFACE** in some way (for example, by **PRESSING** a **MOUSE BUTTON**).
2. The **CONTROLLER HANDLES** the **INPUT EVENT** from the **USER INTERFACE** , often via a registered **HANDLER** or **CALLBACK**, and **CONVERTS** the **EVENT** into an appropriate **USER ACTION**, **UNDERSTANDABLE** for the **MODEL**
3. The **CONTROLLER NOTIFIES** the **MODEL** of the **USER ACTION**, possibly resulting in a **CHANGE** in the **MODEL'S STATE**

Model-View-Controller

4. A **VIEW QUERIES** the **MODEL** in order to **GENERATE** an appropriate **USER INTERFACE** (for example the **VIEW LISTS** the **ENTITY'S CONTENT**)
5. The **VIEW GETS** its own **DATA** from the **MODEL**
6. In some implementations, the **CONTROLLER** may **ISSUE** a **GENERAL INSTRUCTION** to the **VIEW** to **RENDER ITSELF**
7. In others, the **VIEW** is automatically **NOTIFIED** by the **MODEL** of **CHANGES** in **STATE** that **REQUIRE** a **SCREEN UPDATE**
5. The **USER INTERFACE** waits for further **USER INTERACTIONS**, which **RESTARTS** the **CONTROL FLOW CYCLE**

Model-View-Controller

- The **MODEL MANAGES** the **BEHAVIOR** and **DATA** of the **APPLICATION DOMAIN**, **RESPONDS** to **REQUESTS** for **INFORMATION** about its **STATE** (usually from the **VIEW**), and **RESPONDS** to **INSTRUCTIONS** to **CHANGE STATE** (usually from the **CONTROLLER**)
- In **EVENT-DRIVEN** systems, the **MODEL NOTIFIES** observers (usually **VIEWS**) when the **INFORMATION CHANGES** so that they can **REACT**
- The **VIEW RENDERS** the **MODEL** into a **FORM** suitable for **INTERACTION**, typically a **USER INTERFACE** element