# MVC: the Model-View-Controller architectural pattern

Laura Farinetti
Dipartimento di Automatica e Informatica
Politecnico di Torino
laura.farinetti@polito.it

# Model-View-Controller

- MVC is an architectural pattern used in software development
- It's been around for several decades but has gained popularity recently thanks to some popular development frameworks such as Ruby on Rails
- Aim: to promote good programming practices and code reuse by separating a web application into three layers: data, presentation, and the interaction between the two
- By separating these elements from each other, one can be easily updated without affecting the others

# Model-View-Controller

- Developed in Xerox Parc, Palo Alto and implemented for the first time in Smalltalk-80
- Original objective: bridge the gap between the human user's mental model and the digital model that exists in the computer
- Used today in the most important software development frameworks
  - SmallTalk
  - Microsoft Foundation Classes (C++), .Net
  - Java (Struts, Swing, SpringMVC, Cocoon)
  - ActionScript
  - Pyton (Zope, Plone)
  - Ruby
  - PHP (Drupal, Joomla!)

# Traditional applications

- A web application collects data and action requests from users… elaborates/stores them… visualize the results
- Browser directly accesses page
  - Control is not centralized
  - No content/style separation
  - Easy and fast to produce
  - Difficult to maintain

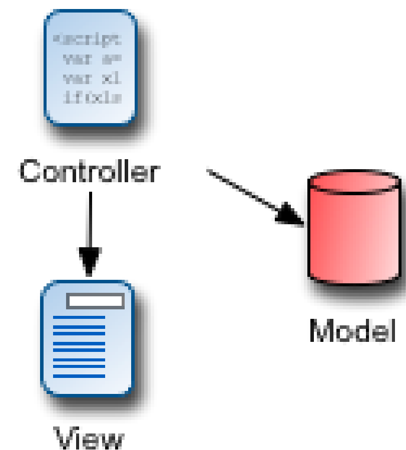
— Request →

← Response —

Script

# MVC Applications

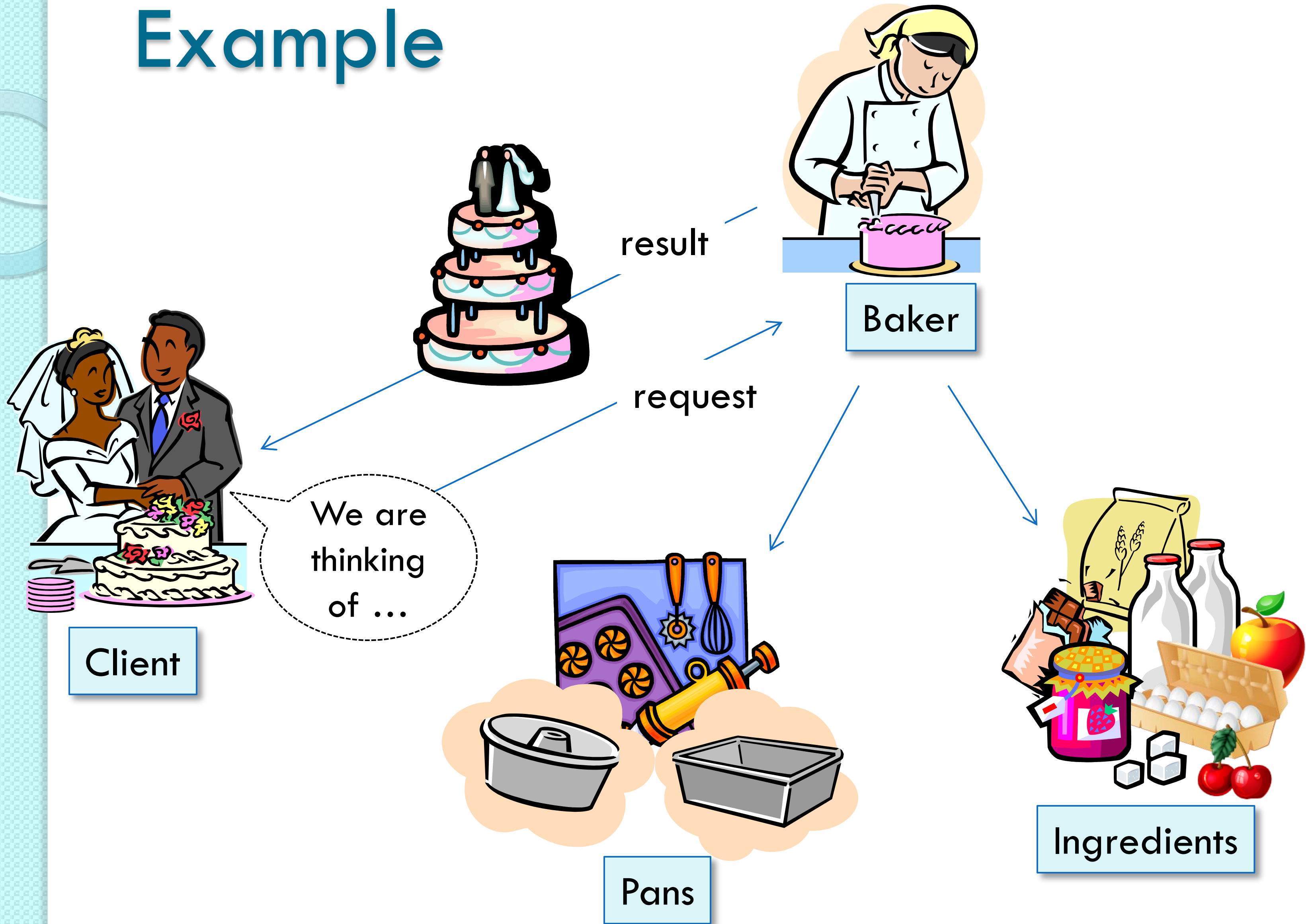
- A web application collects data and action requests from users… elaborates/stores them… visualize the results
- Browser accesses a "controller"
  - Control is centralized
  - Clean separation of content/style
  - More work to produce
  - Easier to maintain and expand



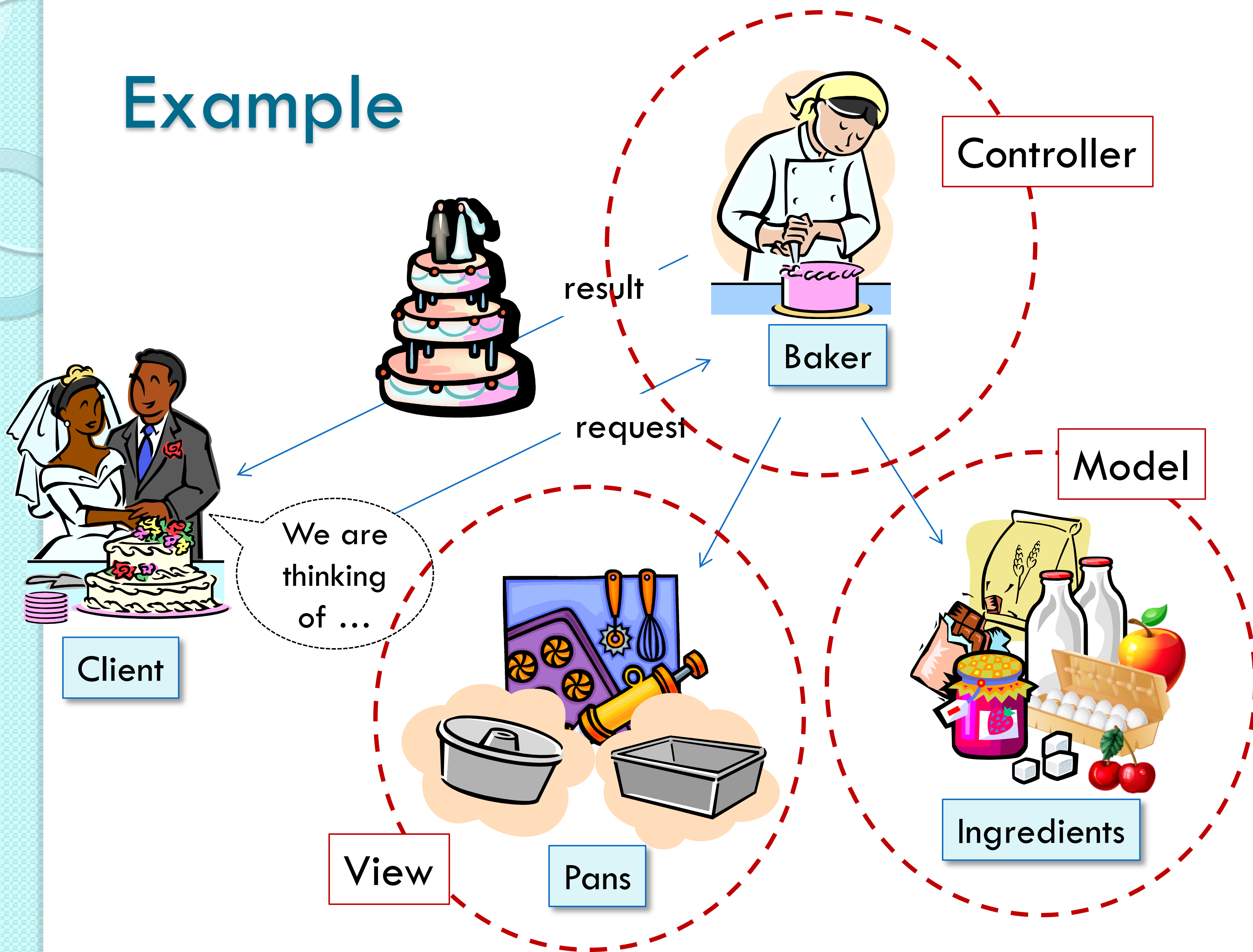
# MVC in short

- The Model represents the data
- The View represents the user interface (i.e. the web page)
- The Controller facilitates communication between the two

# Example



# Example



# The model

- The model represents the data in the application
- “Data” means the “things” in the application that can be abstracted, generally stored in a database
- In addition to defining the data that a “thing” contains, it’s also the model’s job to interact with the database where the actual data are stored, and to implement all logic relating to the creation, fetching, updating, and deleting, and other data manipulation
- The model is also built on top of an object-relational mapping (ORM), a system that connects the elements of the model object to the appropriate fields in the database
  - It automatically handles all interaction with the database, allowing the developer to avoid writing SQL

# The model

- The code in the model is often referred to as business logic
- Business logic is all the rules that define data and how to interact with it
- By isolating the business logic from the presentation layer, it is easier to write and maintain the logic for the application in a way that is both reusable and transportable to another framework without conflicting with the way the user interacts with it on the web page
- A common example of business logic is validation rules

# Example: a blog

- Blogs store posts in a database
- Model called “Post”
- Data
  - Post tells the application what type of data a post contains (usually a title, a date and some body text)
- Business logic
  - When a new blog post is made, the application developer wants to ensure that the post has been given a title
  - When the new post is submitted via web form, the model looks at the data it receives and checks if it conforms to any validation rules that apply
  - If there are any errors, e.g. an empty title field, the model rejects the data and sends an error back to the user
  - If data passes validation, the model opens a connection to the database and save a new post record, using its ORM

# Important

- The model is not the database
- The model is an abstraction
  - of the data itself
  - of everything the application knows about what the data is and how it works
- It is considered good practice to put as much of the code as possible into the model

# The view

- The view is the presentation layer of the application, the user interface
- For the most part, the view is simply the HTML page
- Small bits of inline logic are included, e.g. simple loops to create tables
- The goal in creating a good view is to have as little logic as possible
  - A view should be simple enough that someone who only works with markup and doesn't program, like a designer, can work with it easily
  - Heavy logics is in the model and in the controller

# The view

- There is a different view for each different page in a MVC application
  - Web frameworks that use MVC usually offer a method of dividing the view into even smaller sections to further modularize code
- There are many elements of a single page that usually are in common with other pages on a site (logo and branding, navigation, footer text, …)
  - To keep from repeating all this code in every view, the view offers a layout, an HTML template that contains all the markup in common to multiple pages
  - When a page loads, the framework will take the specific view for that page and insert it into the overall layout

# The view

- Another common feature is what in Rails is called a “partial”
  - A very small chunk of reusable markup that can be included wherever it is needed
- Partials are another way to organize the code into small chunks, following the programming practice called Don’t Repeat Yourself (DRY), which is one of the core philosophies of rapid development frameworks like Rails

# Example: a blog

- Separate pages, and therefore separate views are:
  - the page for viewing all blog posts
  - the page for viewing a specific blog post
  - the page for adding a new blog post
  - the page for editing an existing blog post
- A partial might be used to contain the markup for an individual blog post
  - on the page that shows one specific entry, this partial will be used once
  - on the index page, where all recent posts are shown, the partial is called in a loop

# The controller

- The controller is the translator between the view and the model
- It receives requests from the view (the user), decides what to do, communicates with the model as necessary to send or retrieve data, and then prepares a response for the user to be delivered back to the view
- The controller is composed of methods that operate on a model
- When a user follows a link in the application, the request is sent through what is called the "dispatcher", which accesses the appropriate action in the appropriate controller

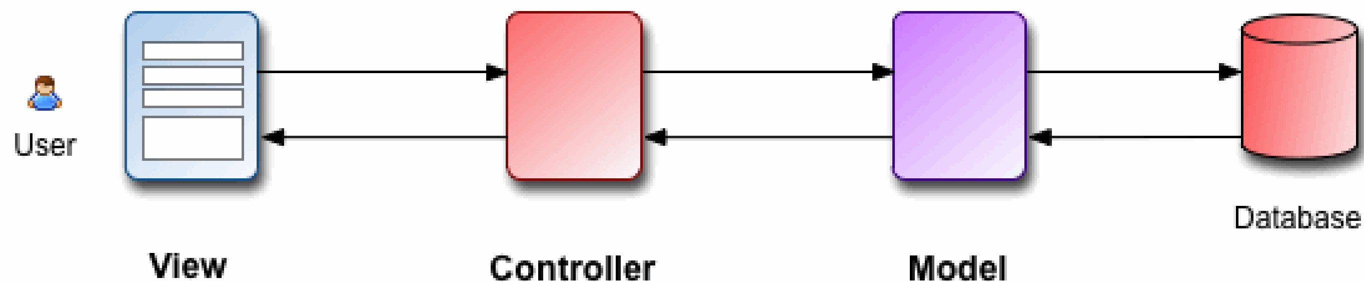
# Example: a blog

- The blog has actions for creating, viewing, editing, and deleting blog posts
- If a user visits a link to a single blog entry
  - the dispatcher calls the blog controller's show action
  - the controller then asks the model for the data for the blog post that the user is requesting
  - when the controller receives this data from the model, it will set variables with that data and pass it on to the view

# Important

- In best practice, the controller don't do any manipulation of data or user interface, it simply translates between the view and the model
  - It presents the model with requests for data that it can understand, and it provides the view with data that it knows how to format and present to the user

# Example: the big picture




- Example:
  - The user submits a form that adds a new blog post
  - The request is sent to the blog controller, which extracts the data submitted via the HTTP POST request and sends a message to the blog model to save a new post with this data
  - The model checks the data against its validation rules
  - Assuming it passes validation, the model stores the data for this new post in the database and tells the controller it was successful
  - The controller then sets a variable for the view indicating success
  - The view displays this message to the user back on the web page, and they know their new blog post has been successfully created
  - If, for some reason, validation of the data failed, the model alerts the controller of any errors, which would set a variable containing these errors for the view
  - The view would then present the original form along with the error messages for any fields that didn't validate

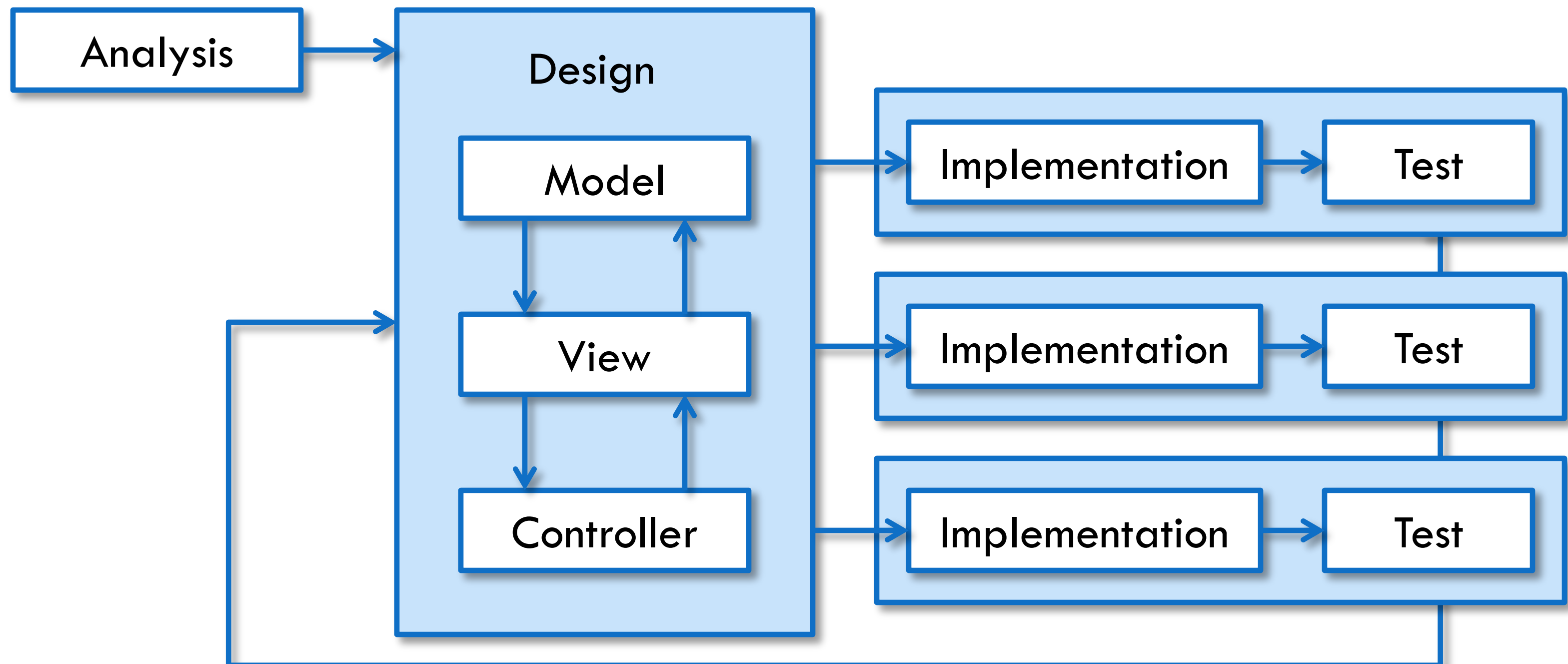
Jimmy Cuadra, "An Introduction to MVC"

# MVC advantages

- Focus separation
  - Model centralizes business logic: information designer
  - View centralizes display logic: visual designer
  - Controller centralizes application flow: interaction designer
- Clean separation of content/style
  - Multi-device systems: same model, different views and controls
  - Creative design: different views, adaptable to different styles or contexts
- Allows multiple people to work on different parts
- Easier testing

# The design process

- Iterative process



# The Ruby on Rails MVC framework

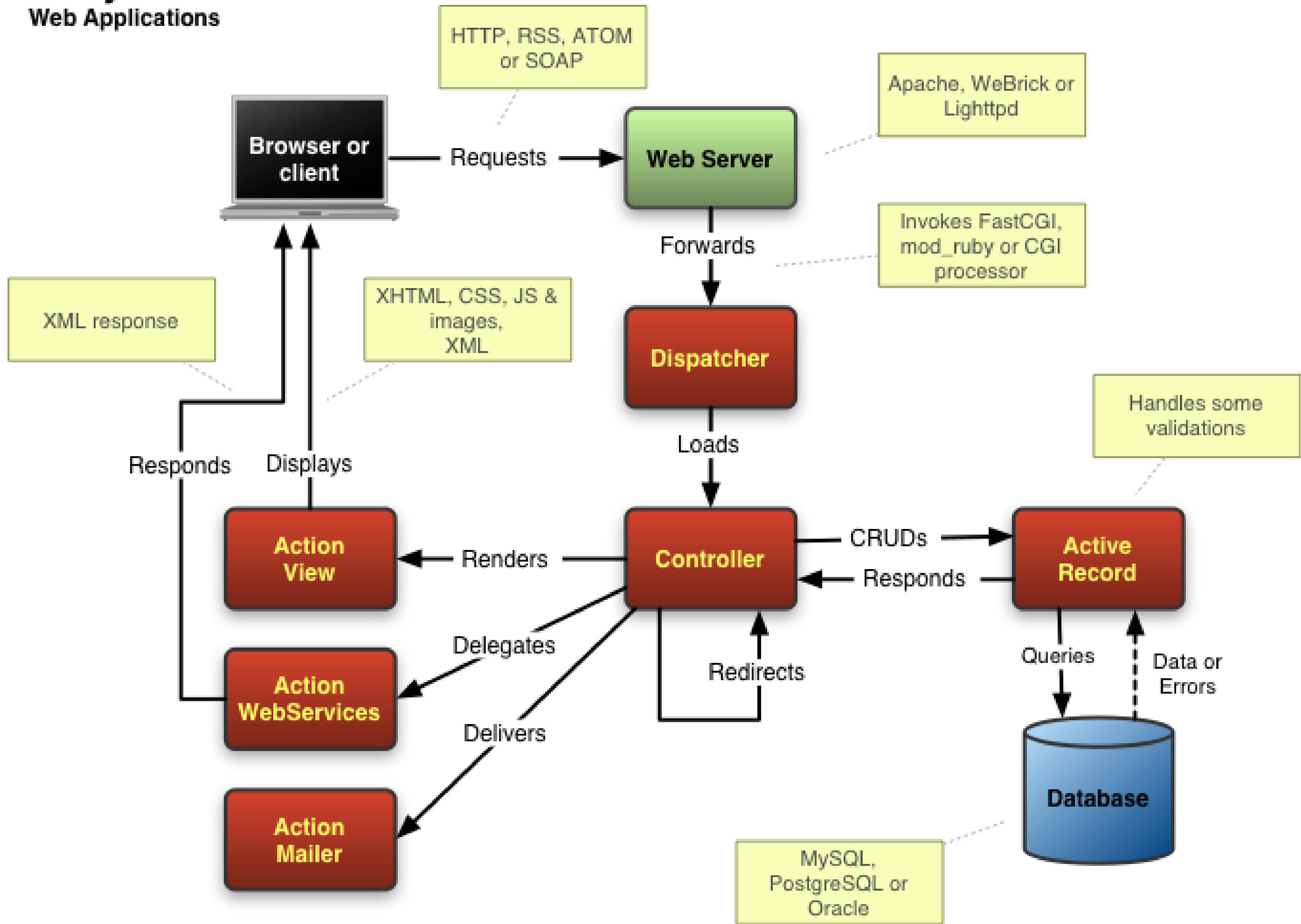
- Rails is a MVC framework
- ActiveRecord: the Model
  - Maintains the relationship between Object and Database and handles validation, association, transactions, and more
  - This subsystem is implemented in ActiveRecord library which provides an interface and binding between the tables in a relational database and the Ruby program code that manipulates database records
  - Ruby method names are automatically generated from the field names of database tables, and so on

# The Ruby on Rails MVC framework

- ActionView: the View
  - A presentation of data in a specific format, triggered by a controller's decision to present the data
  - Script-based templating systems like JSP, ASP, PHP and very easy to integrate with AJAX technology
  - This subsystem is implemented in ActionView library which is an Embedded Ruby (ERb) based system for defining presentation templates for data presentation
  - Every Web connection to a Rails application results in the displaying of a view

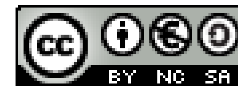
# The Ruby on Rails MVC framework

- ActionController: the Controller
  - The facility within the application that directs traffic, on the one hand querying the models for specific data, and on the other hand organizing that data (searching, sorting, massaging it) into a form that fits the needs of a given view
  - This subsystem is implemented in ActionController which is a data broker sitting between ActiveRecord (the database interface) and ActionView (the presentation engine)




http://wiki.rubyonrails.org/

# Licenza d’uso




- Queste diapositive sono distribuite con licenza Creative Commons “Attribuzione - Non commerciale - Condividi allo stesso modo 2.5 Italia (CC BY-NC-SA 2.5)”
- Sei libero:
  - di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest’opera
  - di modificare quest'opera

- Alle seguenti condizioni:
  - **Attribuzione** — Devi attribuire la paternità dell’opera agli autori originali e in modo tale da non suggerire che essi avallino te o il modo in cui tu usi l'opera.
  - **Non commerciale** — Non puoi usare quest’opera per fini commerciali.
  - **Condividi allo stesso modo** — Se alteri o trasformi quest’opera, o se la usi per crearne un’altra, puoi distribuire l’opera risultante solo con una licenza identica o equivalente a questa.

- http://creativecommons.org/licenses/by-nc-sa/2.5/it/