AngularJS

INTRODUCTION TO THE ANGULARJS FRAMEWORK

POLITECNICO
DI TORINO

e-Lite

AngularJS

- AngularJS is a very powerful JavaScript Framework for writing frontend web applications
- Used in Single Page Application (SPA) projects
- Inspired by the Model-View-Controller pattern
- Extends HTML DOM with additional attributes and makes it more responsive to user actions
- Open source, completely free, and used by thousands of developers around the world
- AngularJS version 1.0 was released in 2012
  - Miško Hevery, a Google employee, started to work with AngularJS in 2009
  - The idea turned out very well, and the project is now officially supported by Google

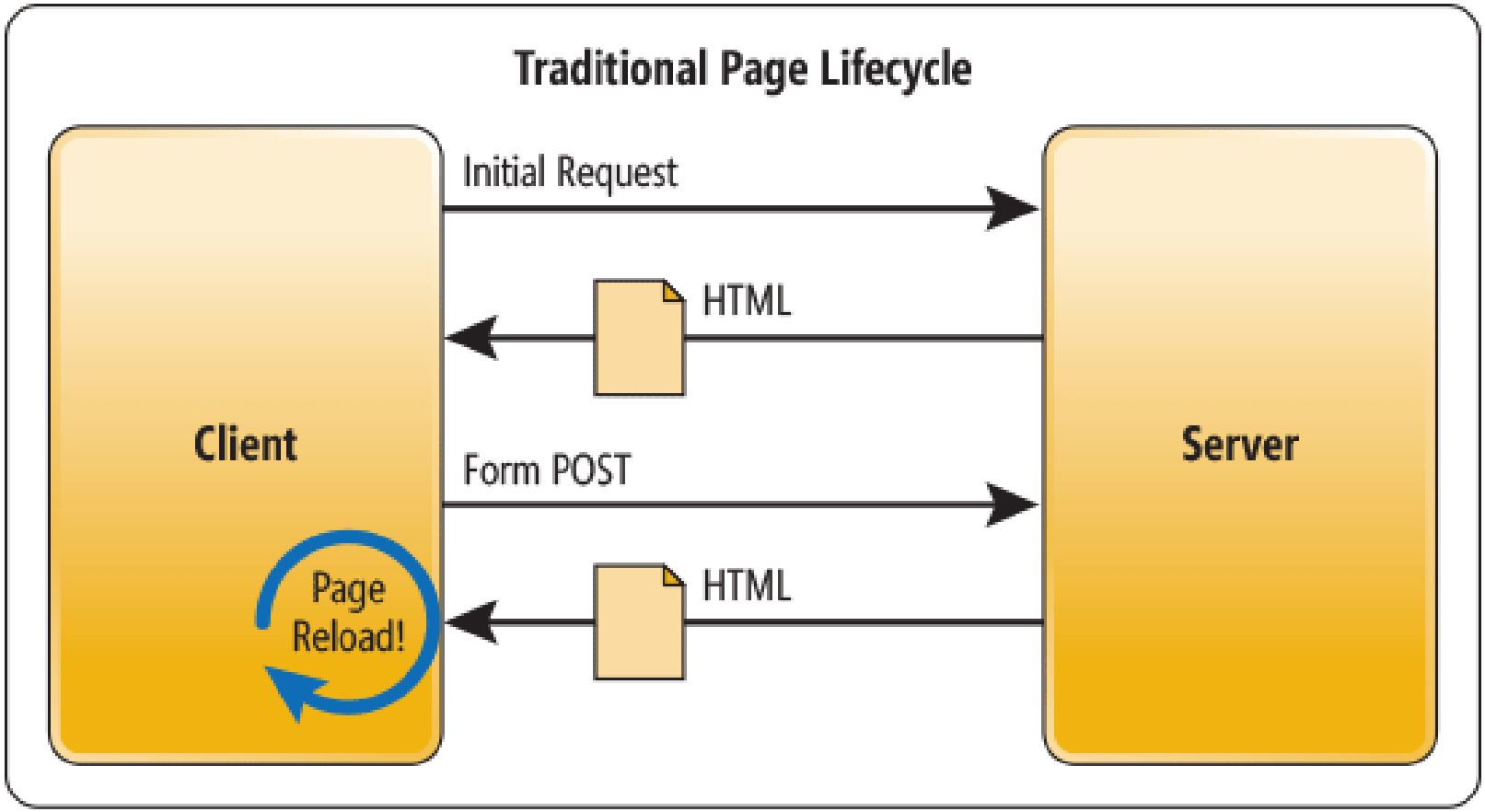
AngularJS

- AngularJS version 2 was released in September 2016
  - AngularJS 2 is not a version upgrade, but a complete rewrite
- AngularJS version 4 was announced in December 2016
- We will use version 1.x, to guarantee compatibility with other technologies
- Website: https://angularjs.org/
  - download version 1.x
- Included in our “starter kit”
  - https://github.com/SoNet-2017/starter-kit

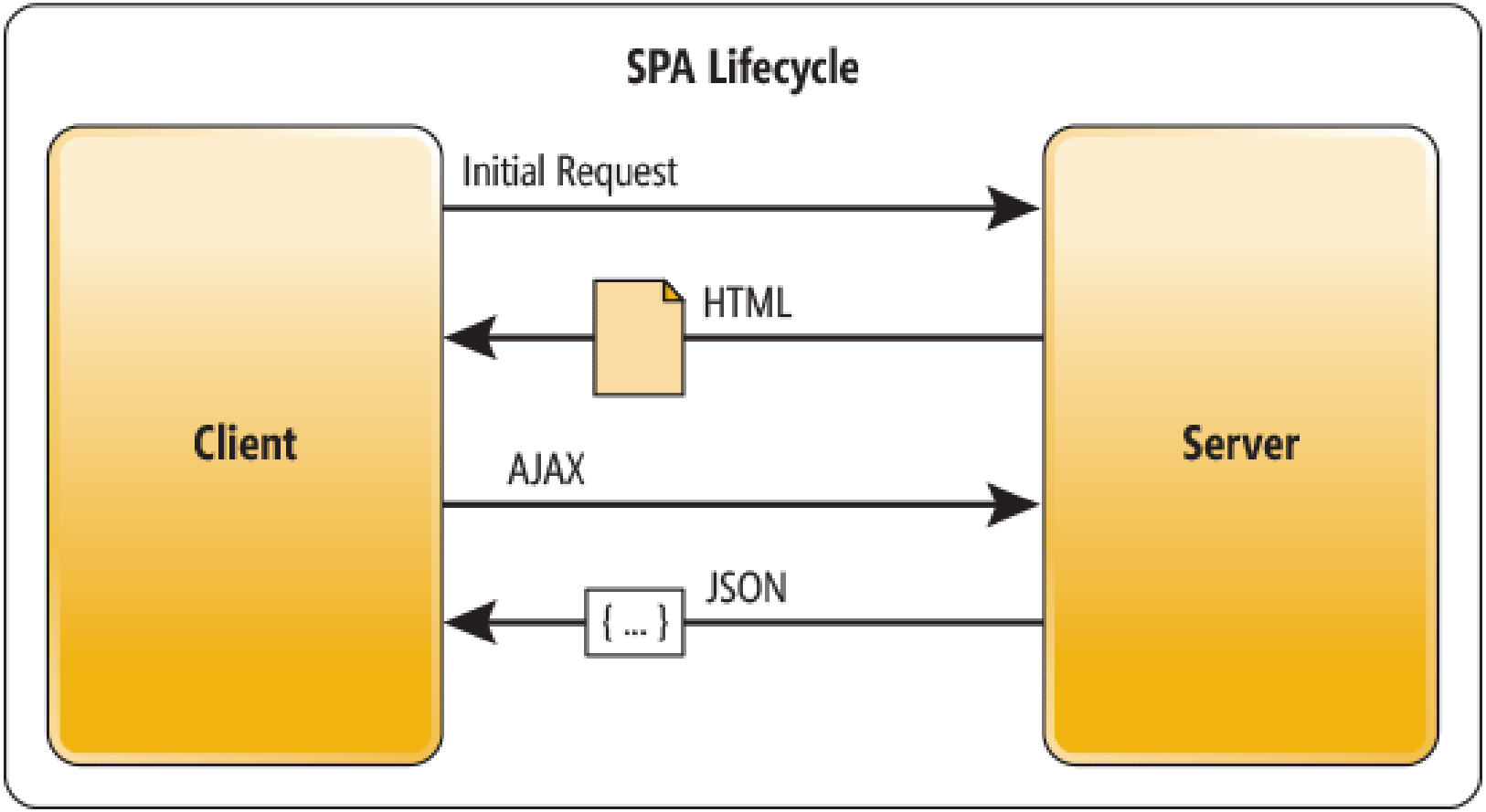
Single Page Applications

- A Single Page Application is a web application in which the majority of interactions are handled on the client without the need to reach a server
- Goal: to provide a more fluid user experience
- A traditional application force you to load everything again after each change
  - It's not very efficient on the bandwidth, especially in the mobile world
- A Single Page Application is an application in which there is a shell page and you load multiple views into that
- In a SPA you can load the initial content upfront and then the different views or the little kind of mini-web pages can be loaded on the fly and embedded into the shell

Traditional web application



Single Page Application



Model-View-Controller

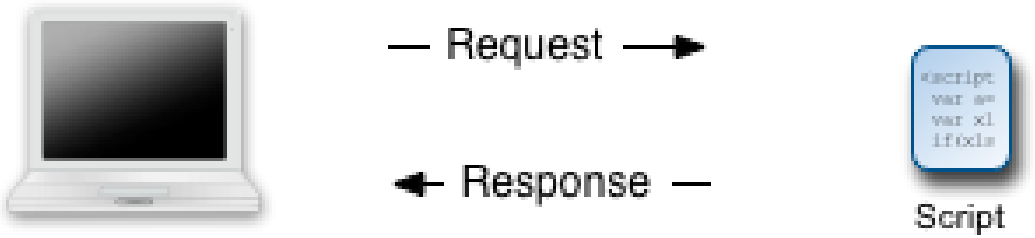
- MVC is an architectural pattern used in software development
- It's been around for several decades but has gained popularity recently thanks to some popular development frameworks
- Aim: to promote good programming practices and code reuse by separating a web application into three layers: data, presentation, and the interaction between the two
- By separating these elements from each other, each 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
  - AngularJS
  - SmallTalk
  - Microsoft Foundation Classes (C++), .Net
  - Java (Struts, Swing, SpringMVC, Cocoon)
  - ActionScript
  - Pyton (Zope, Plone)
  - Ruby
  - PHP (Drupal, Joomla!)

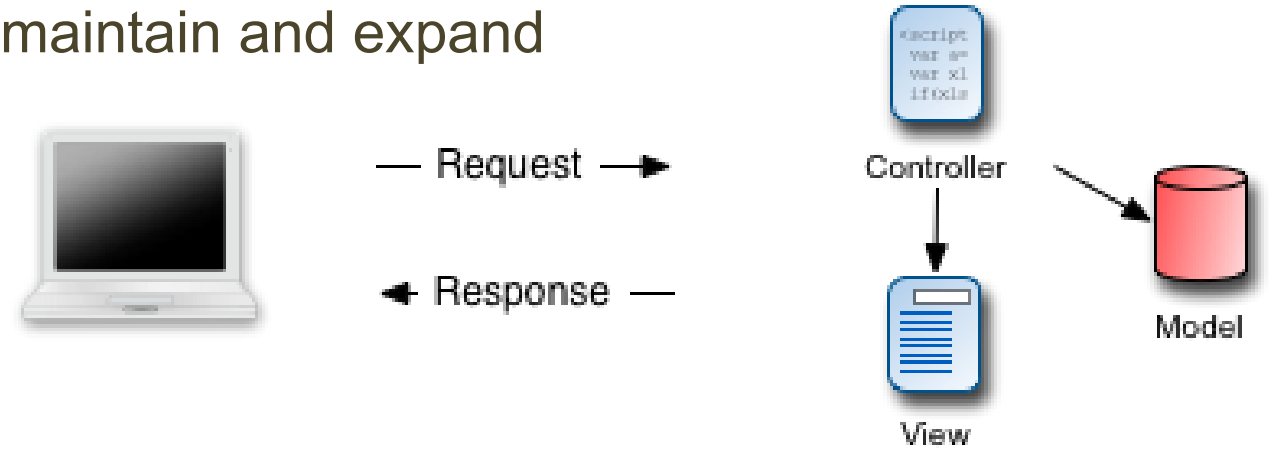
Traditional applications

- 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



MVC applications

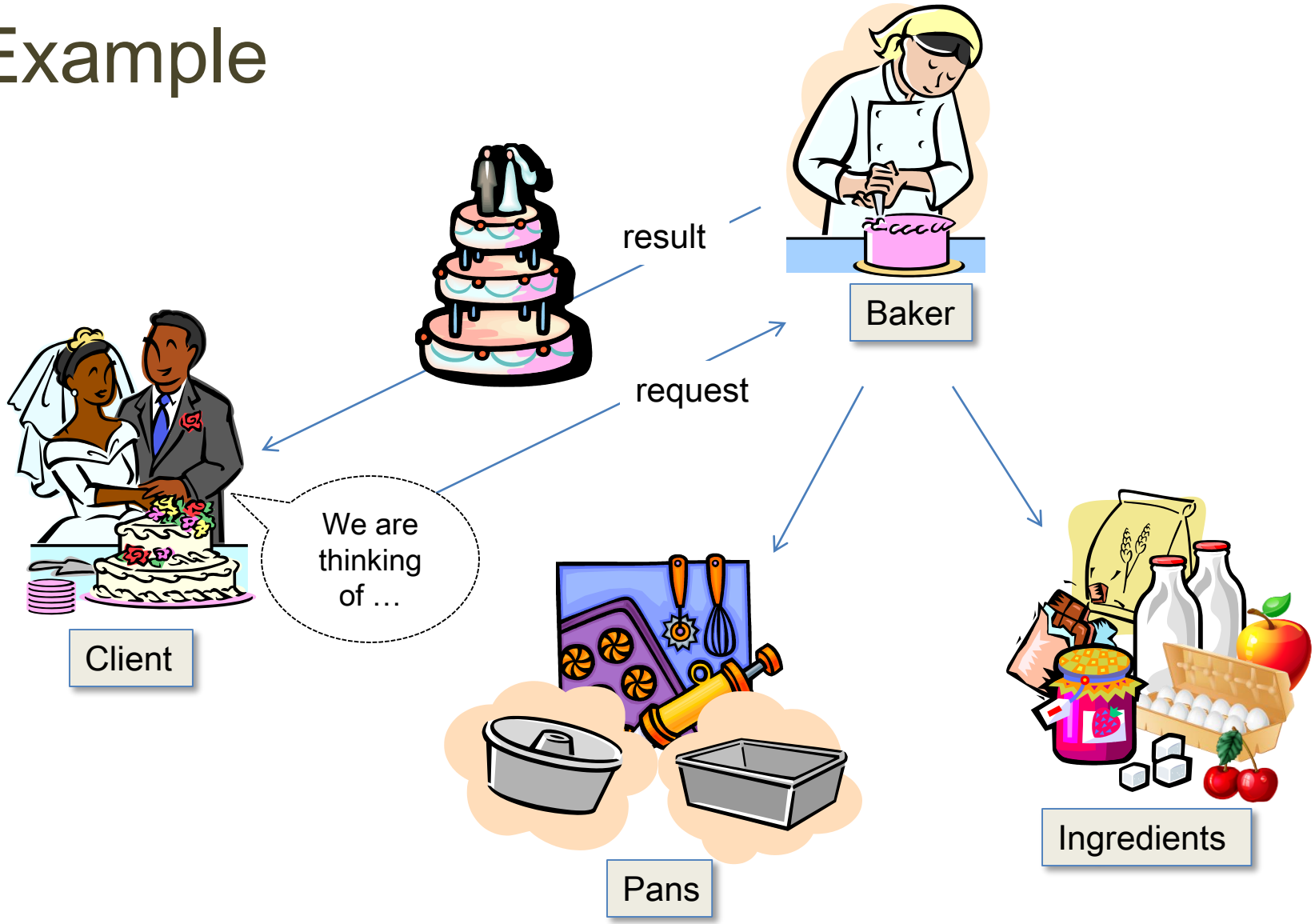
- A web application collects data and action requests from users… elaborates/stores them… visualize the results
- Browser accesses a “controller”
  - Control is centralized
  - Clean separation of content/style
  - More work to produce
  - Easier to maintain and expand



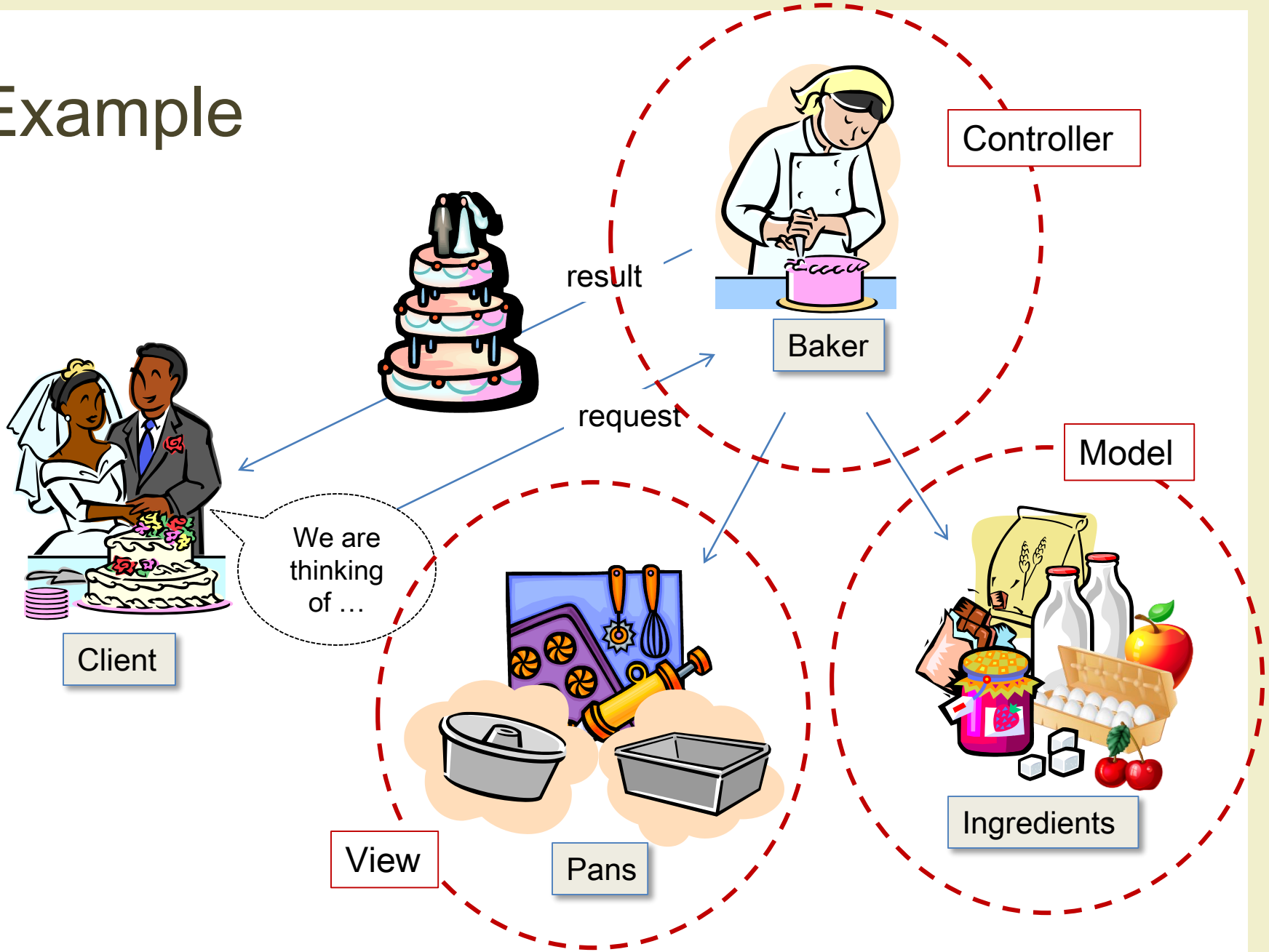
MVC in short

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

Example


result
request
We are thinking of …
Client
Baker
Pans
Ingredients

Example


Controller
Baker
result
request
Model
We are thinking of …
Client
View
Pans
Ingredients

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 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, i.e. the user interface
- For the most part, the view is simply an 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

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

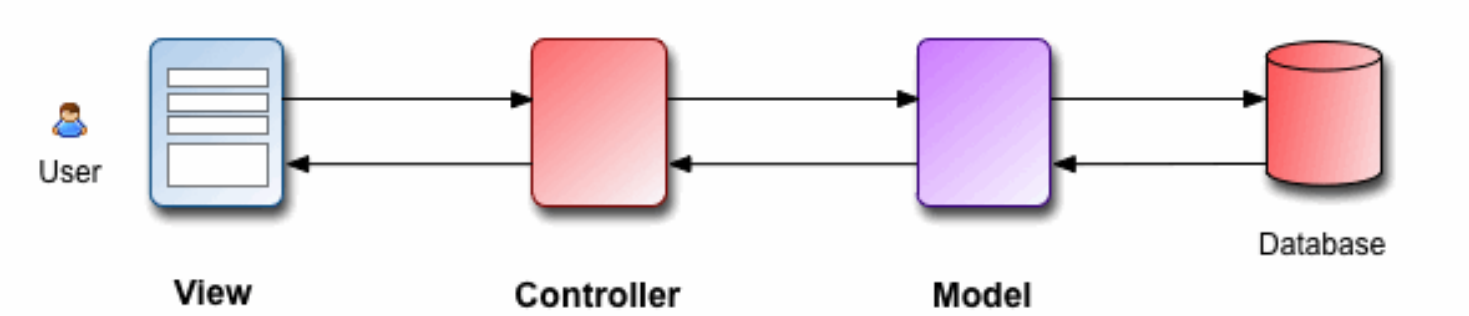
Example: a blog

- 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: a blog



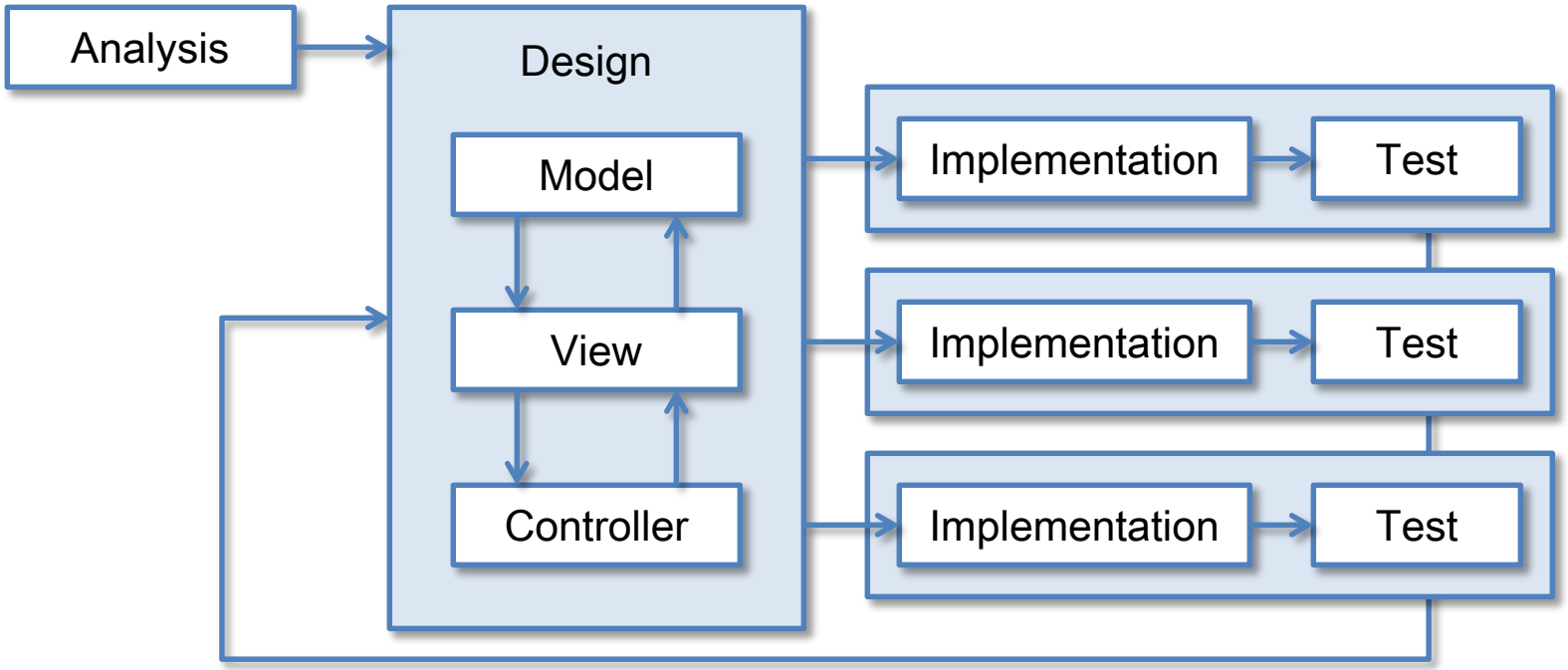
- The user submits a form that adds a new blog post
- The request is sent to the blog controller, which extracts the data submitted via the HTTP POST request and sends a message to the blog model to save a new post with this data
- The model checks the data against its validation rules
- Assuming it 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

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

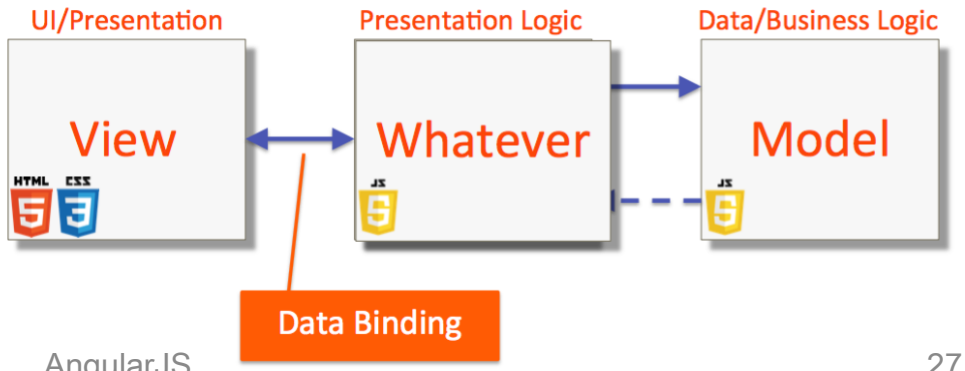
- Iterative process



- In Angular 1, HTML markup is the View, Controller is the Controller and the Service (when it used to retrieve data) is the Model

MVC and AngularJS

- The MVC pattern concepts are somehow abstract
  - thepattern may have different implementations depending on the language, platform, and purpose of the application
- One of the authors says that AngularJS adopts a Model-View-Whatever (MVW or MV*) pattern
  - Regardless of the name, the most important benefit is that the framework provides a clear separation between the application layers, providing modularity, flexibility, and testability



References

- Official site
  - https://angularjs.org/
- AngularJS tutorials
  - https://www.w3schools.com/angular/default.asp
  - https://www.tutorialspoint.com/angularjs/
  - http://www.html.it/guide/guida-angularjs/

License

- This work is licensed under the Creative Commons “Attribution-NonCommercial-ShareAlike Unported (CC BY-NC-SA 3,0)” License.
- You are free:
  - to Share - to copy, distribute and transmit the work
  - to Remix - to adapt the work
- Under the following conditions:
  - Attribution - You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
  - Noncommercial - You may not use this work for commercial purposes.
  - Share Alike - If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.
- To view a copy of this license, visit http://creativecommons.org/license/by-nc-sa/3.0/