JAVA SPRING
CURS 2: SOLID, MVC

SOLID PRINCIPLES

Robert C Martin

Single Responsibility Principle

- No *God* classes, do one thing and do it well

- programmms easy to upgrade, easy to debug
- programms easy to expand

- Example: editing and printing, separate logic/presentation

- In Spring: **controller, service, repository** etc.

Open Closed Principle

- Software entities should be opend for extension, but closed for modification

- Use interfaces, inheritance, polymorphism
- Source code remains unchanged

- Example: change the lens, not the camera! *plug-ins*

Liskov Substitution Principle

- If S is a subtype of T, then objects of type T may be replaced with objects of type S (i.e., objects of type S may substitute objects of type T) without altering any of the desirable properties of the program (correctness, task performed, etc.)

- IsA Test

- Example: bird.fly( )

Interface Segregation Principle

- Implementing classes should not be forced to depend on methods that they do not use.
- use small, cohesive interfaces: ***role interfaces***
- Single responsability principle for interfaces

- Example: robot vs coffee maker

Dependency Inversion Principle

- High-level modules should not depend on low-level modules. Both should depend on abstractions
- Abstractions should not depend on details. Details should depend on abstractions

- Avoids tightly coupled classes => loosely coupled classes
- Example on/off button

- In Spring: Dependency Inversion decouples code so DI can be used

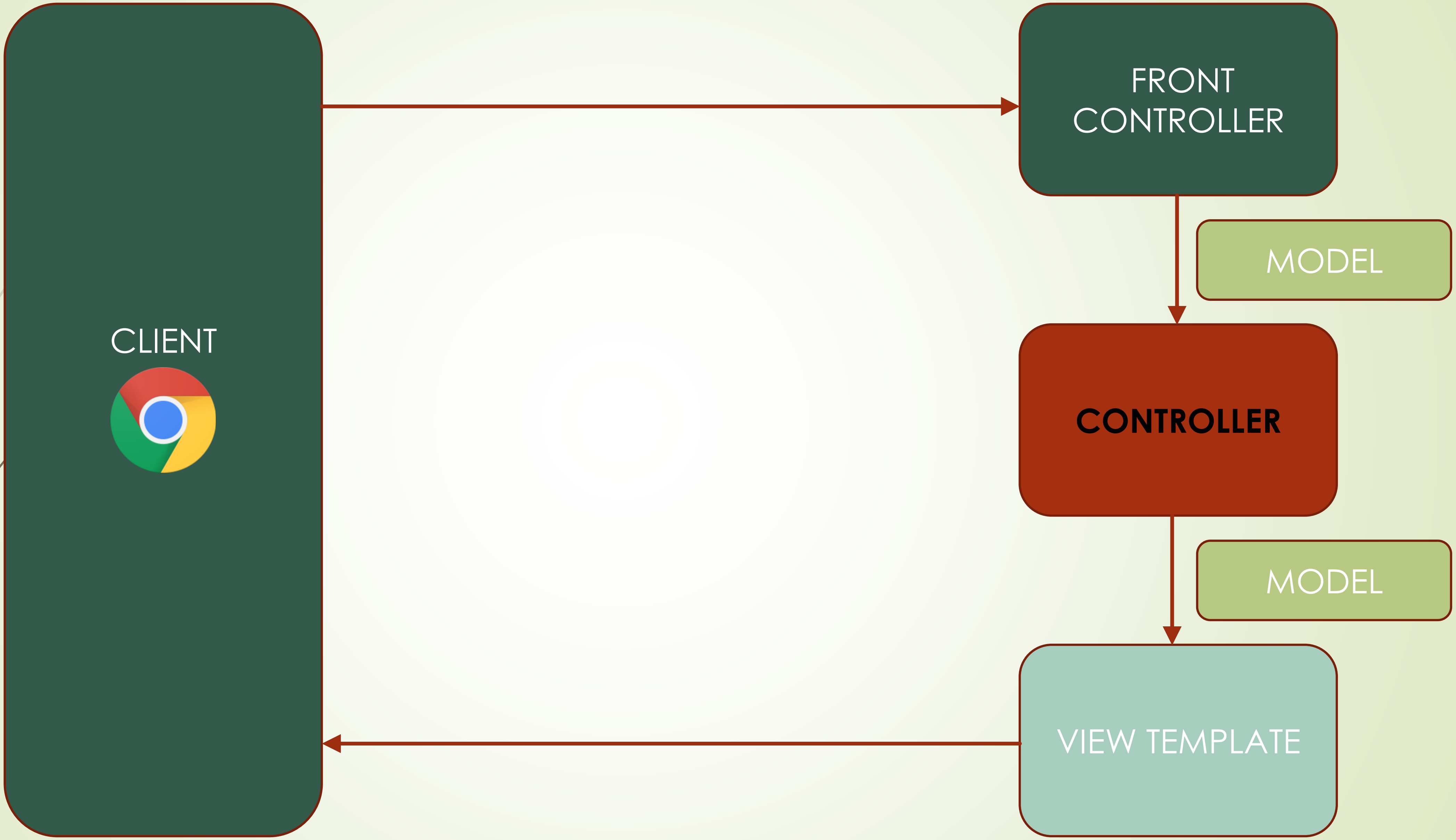
MVC Model View Controller

MVC Architectural pattern

- Reusable solution to a commonly occurring problem.
- May be implemented by different architecutres

- MVC divides an application in three interconnected parts
- Popular for designing web-appications

- Model: data structure and logic.
- View: data representation.
- Accepts requests and interacts with model and view.


CLIENT
FRONT CONTROLLER
MODEL
CONTROLLER
MODEL
VIEW TEMPLATE

Controller

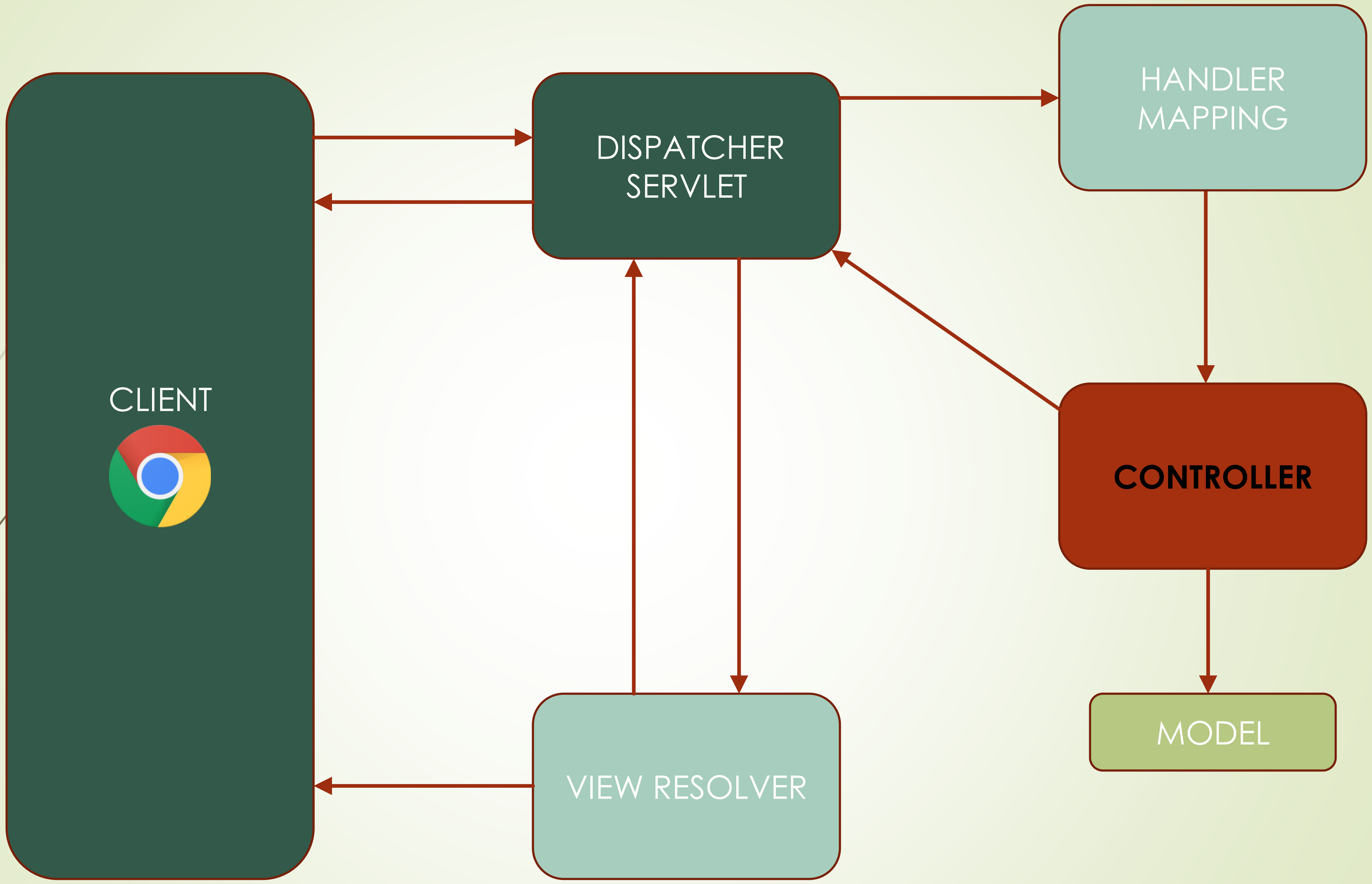
- Front controller design pattern: a single controller is responsible for directing incoming HttpRequests to all other controllers.
- Spring Front Controller: **DispatcherServlet** (extends Servlet)
- DispatcherServlet dispatches incoming HttpRequest to **handlers**

Controller

- RequestMappingHandlerAdapter suports @RequestMapping annotated classes/methods
- HttpRequestHandlerAdapter suports HttpRequestHandler
- **SimpleControllerHandlerAdapter** suports Controllers specified with **@Controller** and **@RestController**.
- @RestController defines **@ResponseBody** by default

Controller

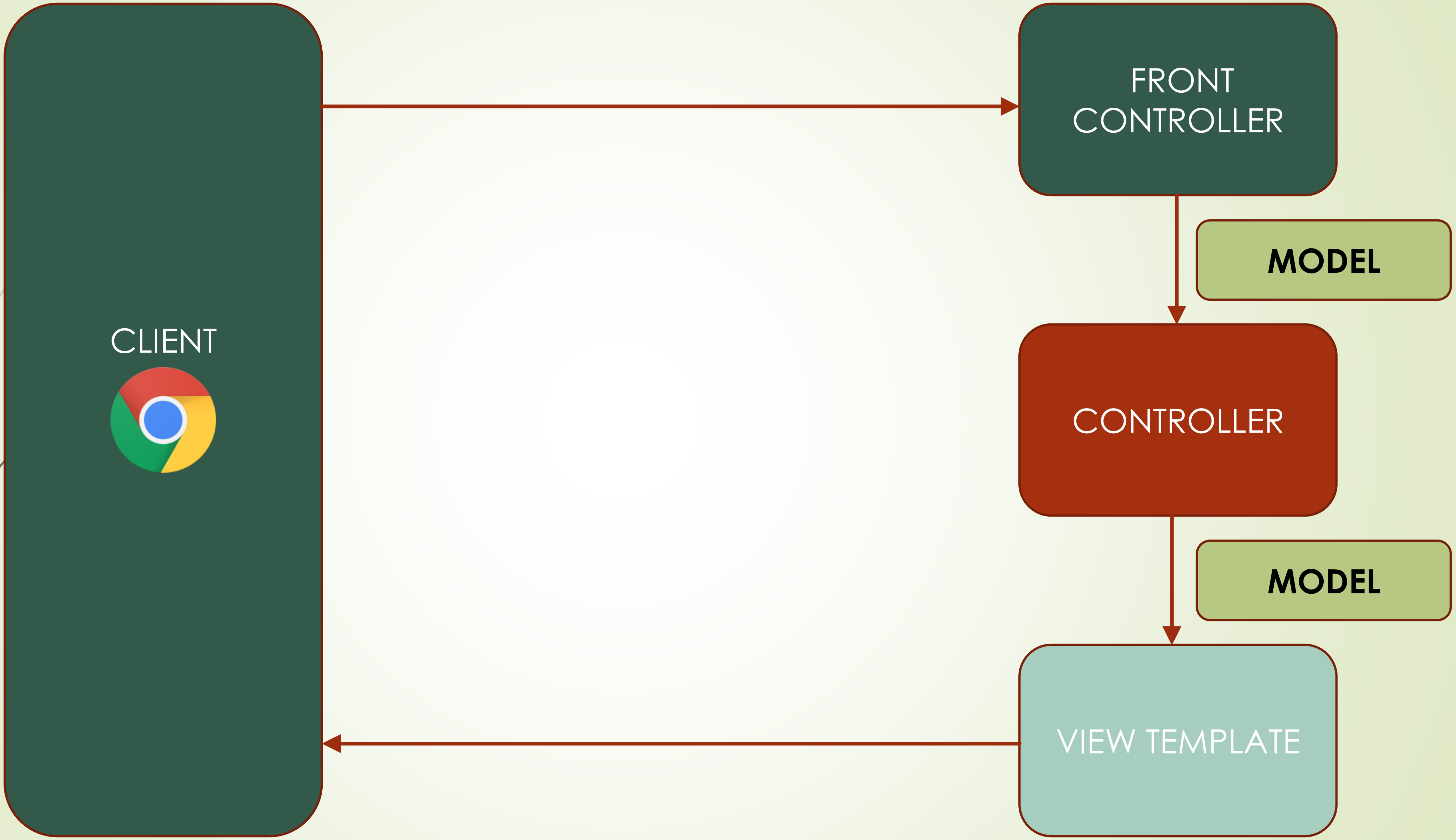
- @RequestMapping specifies an endpoint available within the WebApplicationContext
- HttpServletRequest – request object with parameters, headers etc.
- @ResponseBody -- returned object is automatically serialized into JSON.
- @RequestParam – extracts parameter from request
- @PathVariable – extracts data from template URI request


HANDLER MAPPING
DISPATCHER SERVLET
CLIENT
CONTROLLER
MODEL
VIEW RESOLVER

View

- ViewResolver determines the types of views served by the dispatcher and from where they are served.

- setPrefix(“WEB-INF/views”)
- setSuffix(“html”)


CLIENT
FRONT CONTROLLER
MODEL
CONTROLLER
MODEL
VIEW TEMPLATE

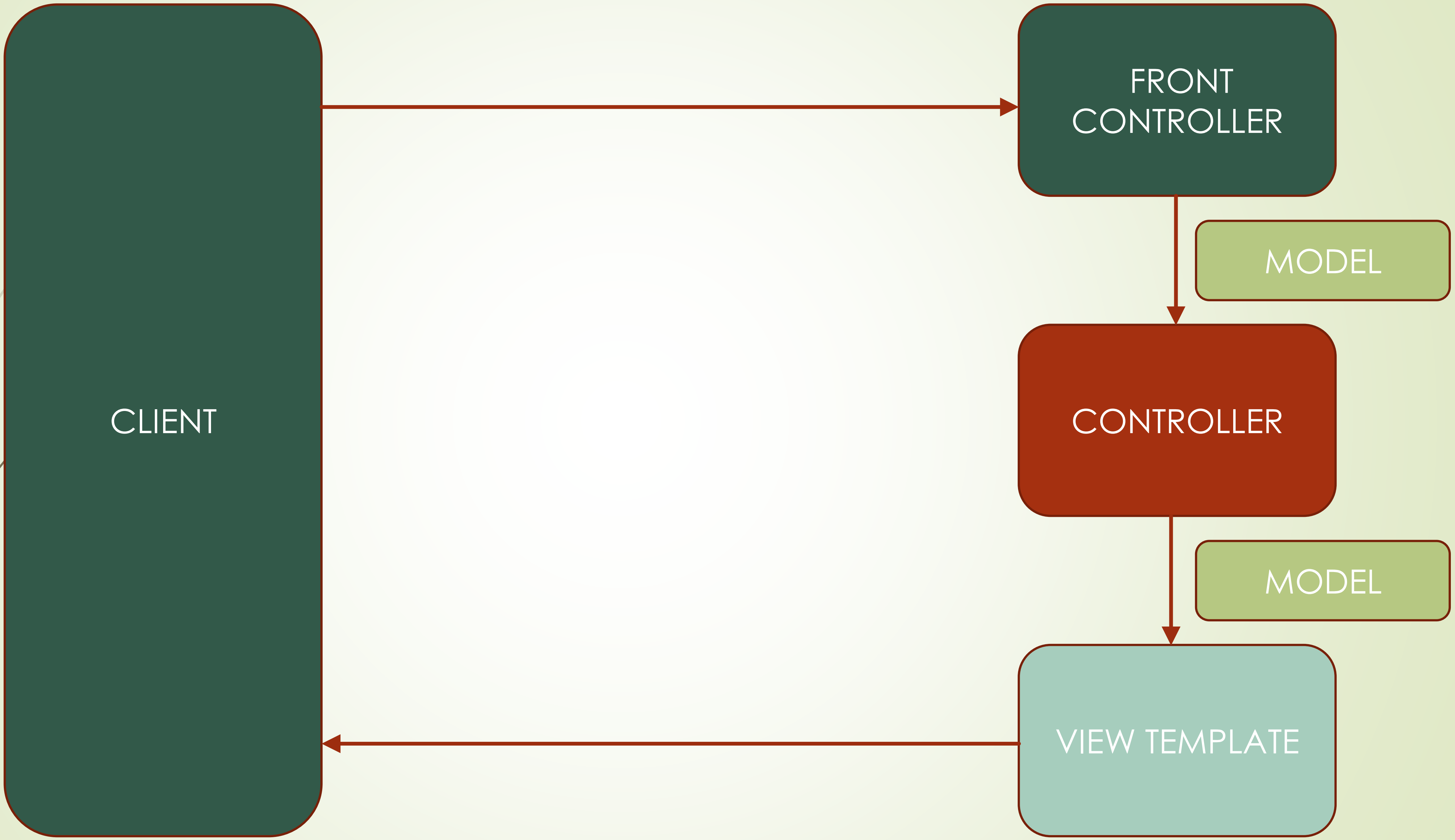
Model

- Container for application data
- Initially Model parameters of @RequestMapping methods are empty
- Object (String, List, Object etc) may be added in the model by using addAttribute method

```
model.addAttribute("hello", new Hello());
```

- Objects can be accesed in views.

```
${hello}
```


CLIENT
FRONT CONTROLLER
MODEL
CONTROLLER
MODEL
VIEW TEMPLATE

Bibliografie

- https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html
- https://www.baeldung.com/spring-mvc-handler-adapters