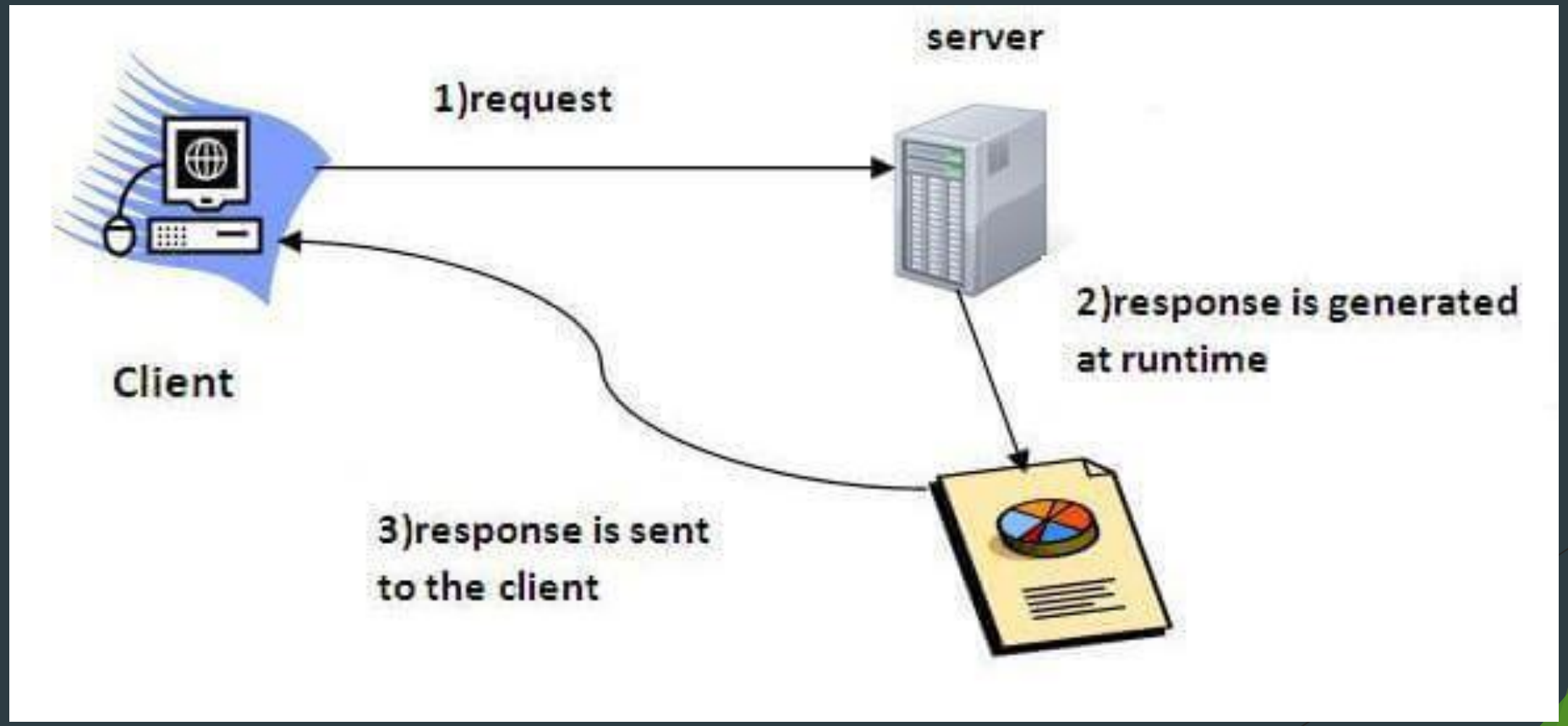
What is a web application?

- A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Spring, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

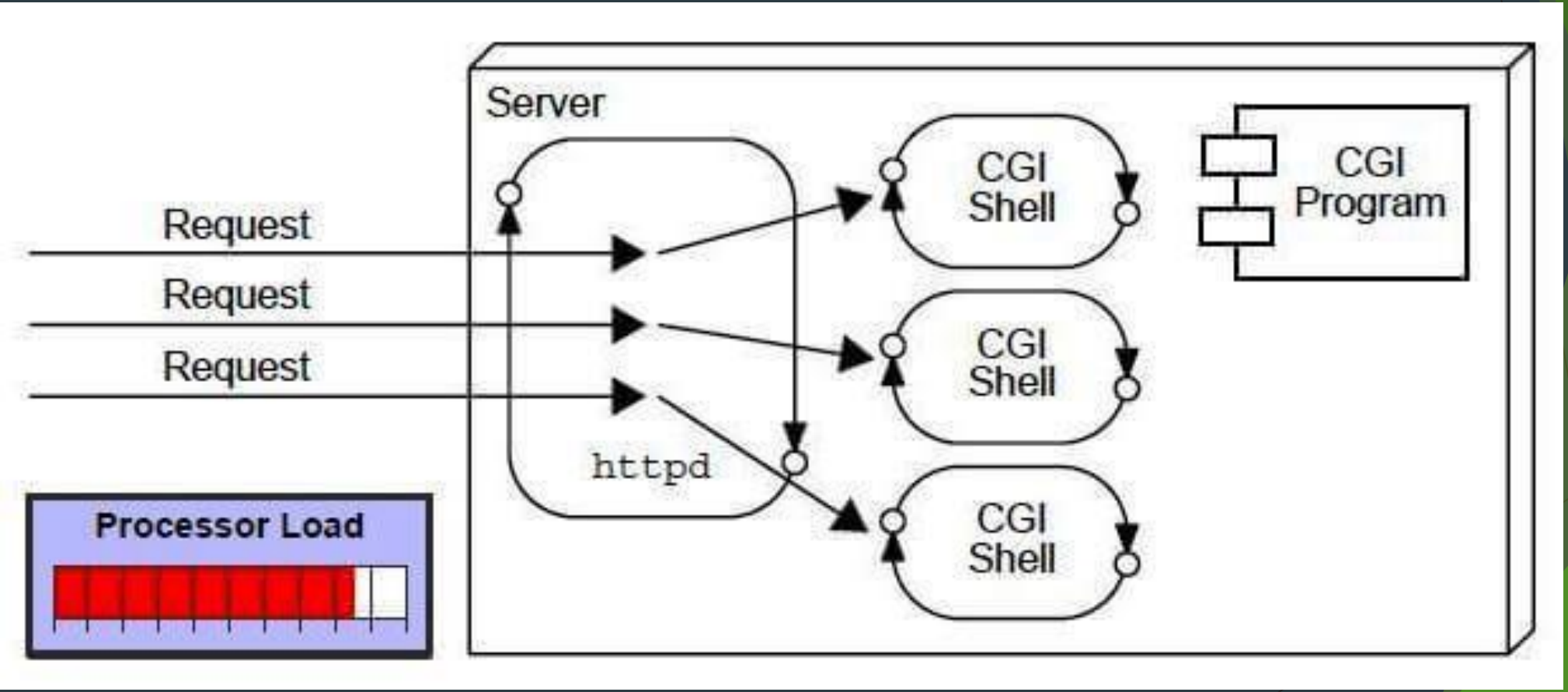


Actually, the Eclipse Foundation legally *had* to rename Java EE. That's because Oracle has the rights over the “Java” brand.
So to choose the new name, the community voted and picked: Jakarta EE. In a certain way, it's still JEE.

Version	Date
J2EE 1.2	December 1999
J2EE 1.3	September 2001
J2EE 1.4	November 2003
Java EE 5	May 2006
Java EE 6	December 2009
Java EE 7	April 2013
Java EE 8	August 2017
Jakarta EE	February 2018*

CGI (Common Gateway Interface)

- CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.

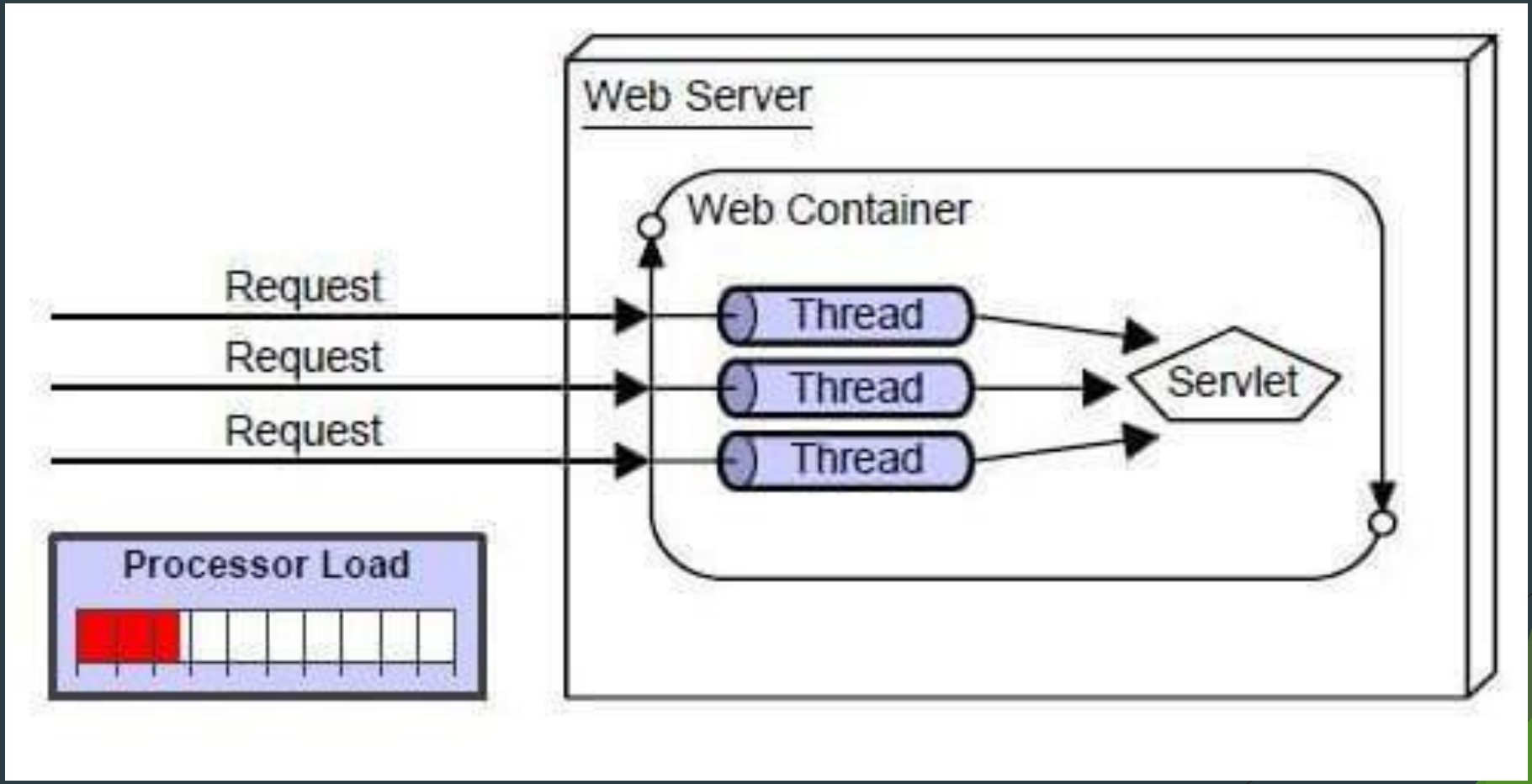


Disadvantages of CGI

- If the number of clients increases, it takes more time for sending the response.
- For each request, it starts a process, and the web server is limited to start processes.
- It uses platform dependent language e.g. C, C++, perl.

Servlet

- **Servlet** technology is used to create a web application (at server side and generates a dynamic web page).



Advantages of Servlet

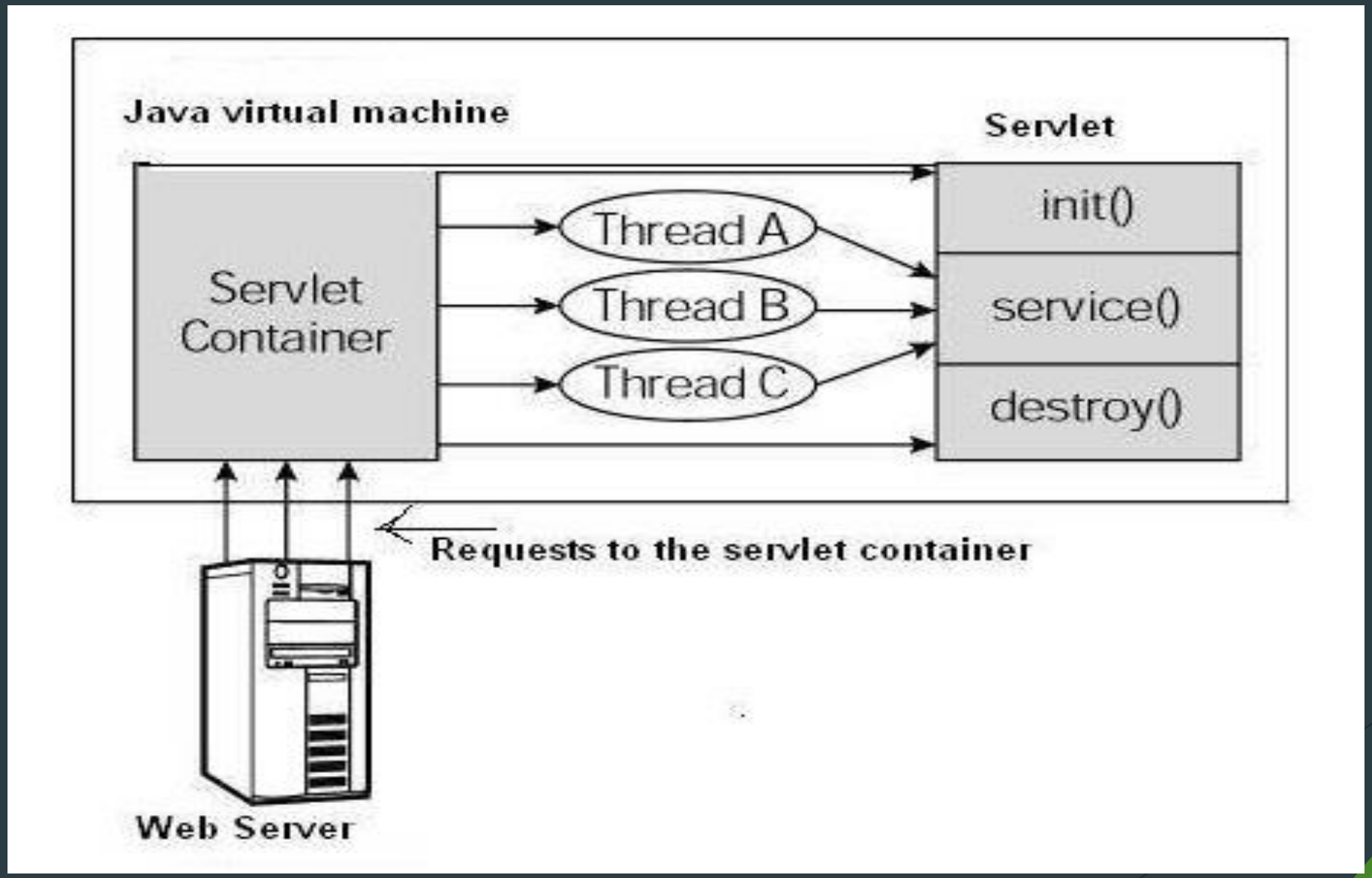
There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

- **Better performance:** because it creates a thread for each request, not process.
- **Portability:** because it uses Java language.
- **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
- **Secure:** because it uses java language.

Servlets - Life Cycle

- The servlet is initialized by calling the **init()** method.
- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()** method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

Servlets - Life Cycle



Servlet Tutorials

- https://www.javatpoint.com/servlet-tutorial
- https://www.tutorialspoint.com/servlets/index.htm

Environment Setup

- Download and install JDK
- Setup Eclipse IDE
- Spring Framework Libraries

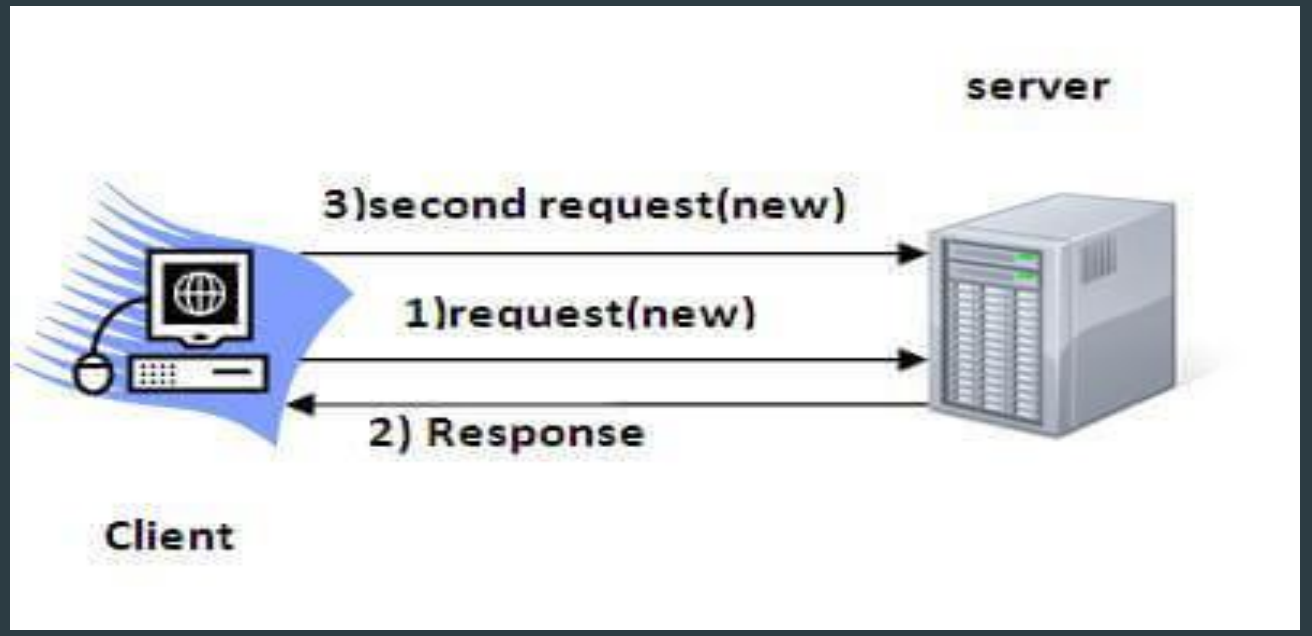
  https://repo.spring.io/ui/native/release/org/springframework/spring
- Download and install Tomcat

Servlet - Example

```java
/**
 * Servlet implementation class FirstServelt
 */
@WebServlet("/FirstServelt")
public class FirstServelt extends HttpServlet {

    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.print("<html><body>");
        out.print("<h3>Hello Servlet</h3>");
        out.print("</body></html>");
    }
```

Session

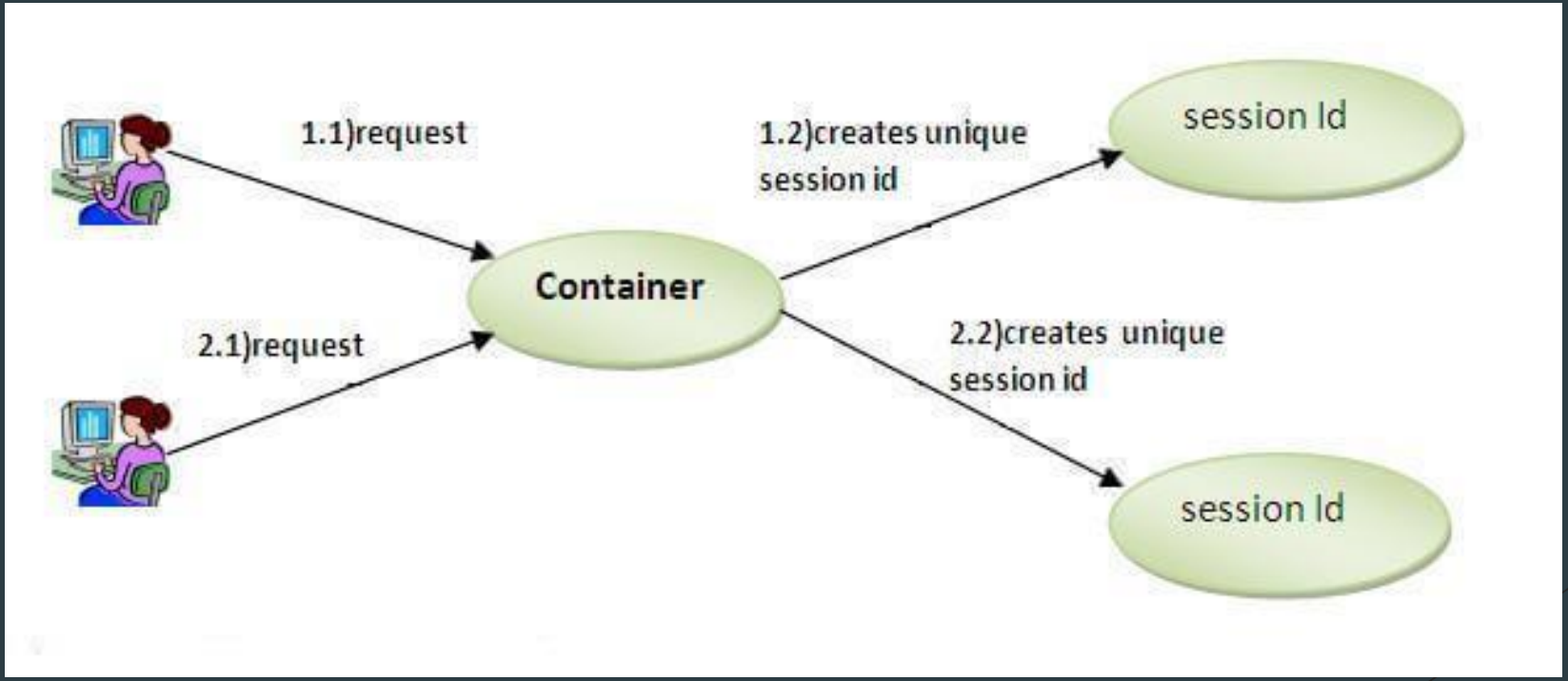
- **Session:** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.
- **Http protocol:** is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.



HttpSession interface

In such case, container creates a session id for each user. The container uses this id to identify the particular user. An object of HttpSession can be used to perform two tasks:

- bind objects
- view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



Cookies in Servlet

- **Cookies:** are text files stored on the client computer and they are kept for various information tracking purpose. Java Servlets transparently supports HTTP cookies.
- **There are three steps involved in identifying returning users:**
  - Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
  - Browser stores this information on local machine for future use.
  - When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.
- **Advantage of Cookies:**
  - Simplest technique of maintaining the state.
  - Cookies are maintained at client side.
- **Disadvantage of Cookies:**
  - It will not work if cookie is disabled from the browser.
  - Only textual information can be set in Cookie object.

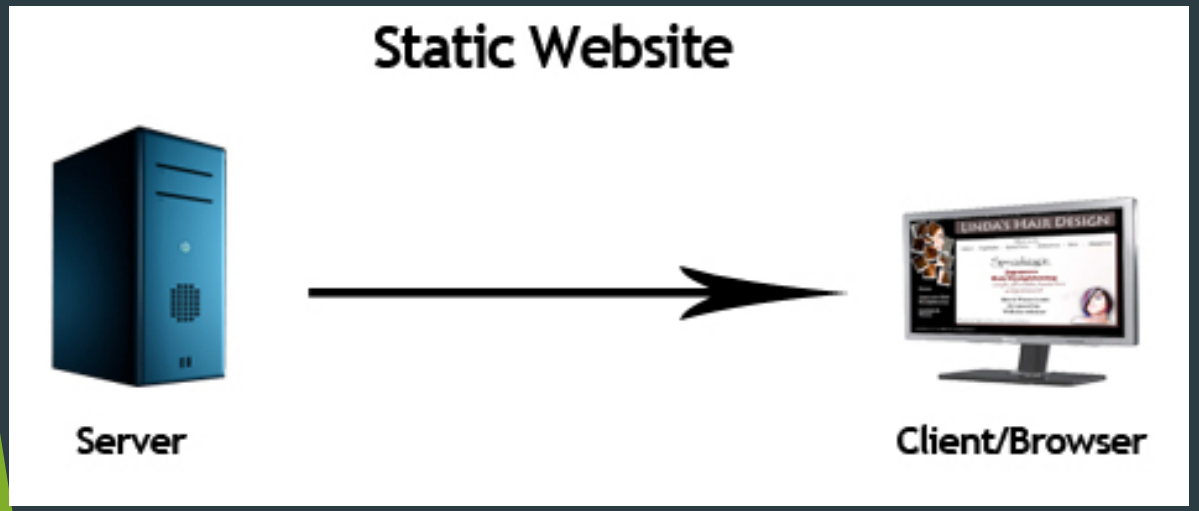
Difference between Cookies and Session

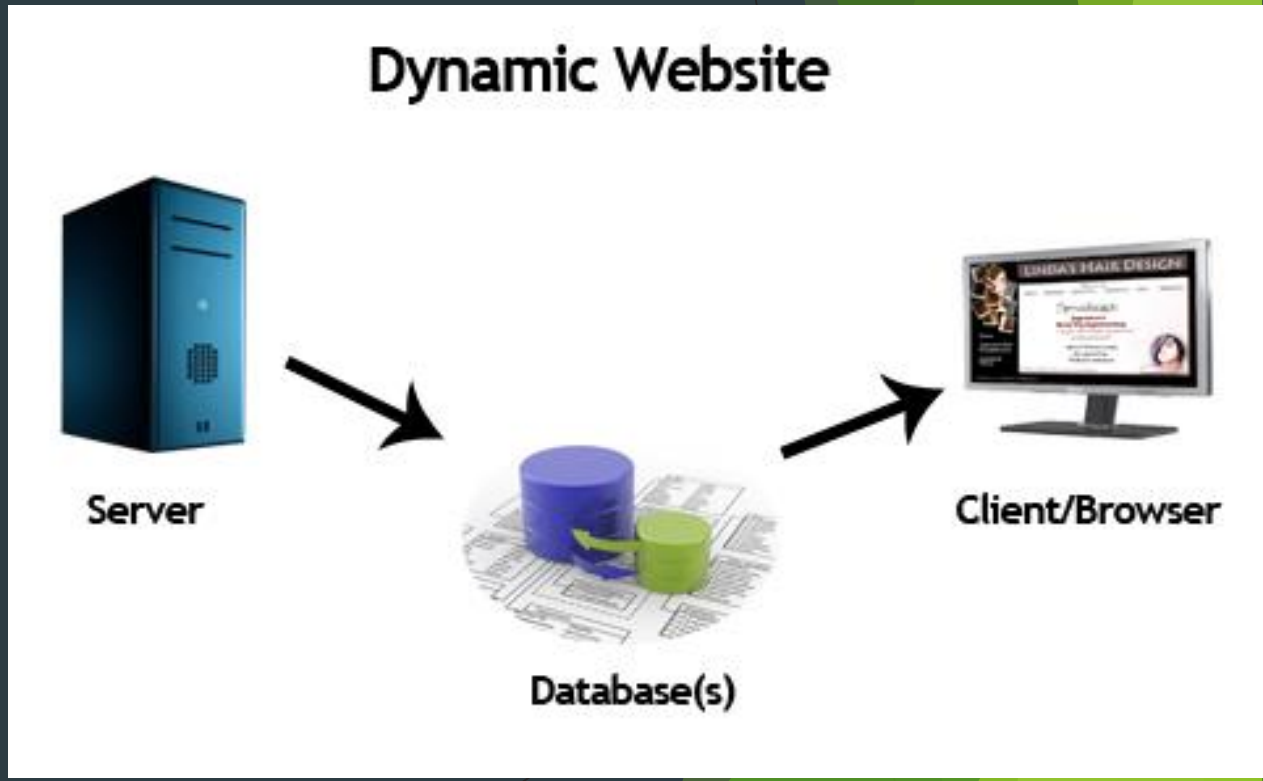
Session	Cookies
A session stores the variables and their values within a file in a temporary directory on the server.	Cookies are stored on the user's computer as a text file.
The session ends when the user logout from the application or closes his web browser.	Cookies end on the lifetime set by the user.
It can store an unlimited amount of data.	It can store only limited data.
We can store as much data as we want within a session, but there is a maximum memory limit, which a script can use at one time, and it is 128 MB.	The maximum size of the browser's cookies is 4 KB.
Sessions are more secured compared to cookies, as they save data in encrypted form.	Cookies are not secure, as data is stored in a text file, and if any unauthorized user gets access to our system, he can temper the data.

Static vs Dynamic website

Static Website	Dynamic Website
Prebuilt content is same every time the page is loaded.	Content is generated quickly and changes regularly.
It uses the **HTML** code for developing a website.	It uses the server side languages such as **PHP,SERVLET, JSP, and ASP.NET** etc. for developing a website.
It sends exactly the same response for every request.	It may generate different HTML for each of the request.
The content is only changed when someone publishes and updates the file (sends it to the web server).	The page contains "server-side" code which allows the server to generate the unique content when the page is loaded.
Flexibility is the main advantage of static website.	Content Management System (CMS) is the main advantage of dynamic website.

Static vs Dynamic website





Web Terminology

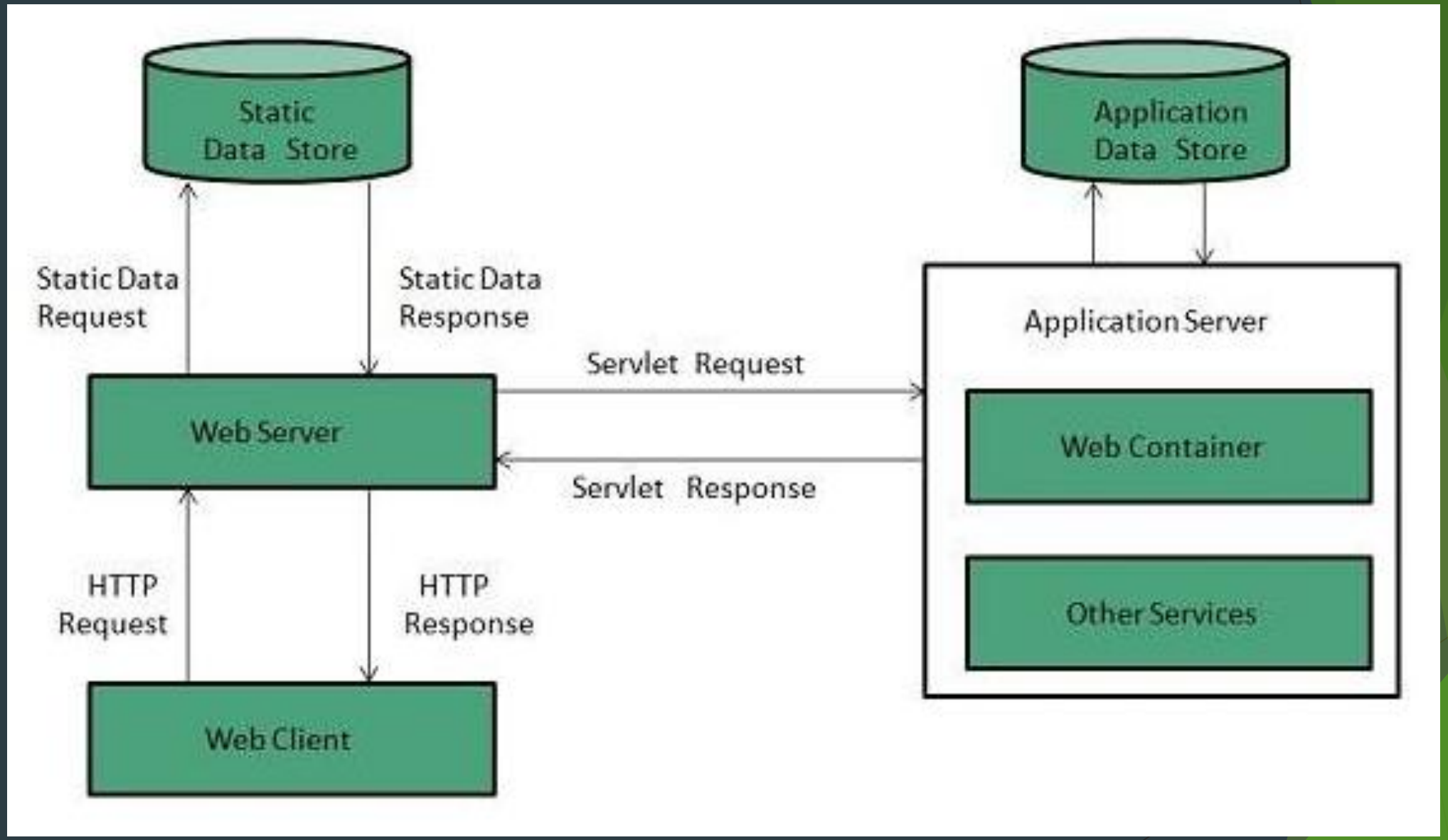
- **HTTP:** It is the data communication protocol used to establish communication between client and server.
- **HTTP Requests:** It is the request send by the computer to a web server that contains all sorts of potentially interesting information.
- **Container:** It is used in java for dynamically generating the web pages on the server side.
- **Content Type:** It is HTTP header that provides the description about what are you sending to the browser (text/html, text/plain, application/pdf, application/x-zip, images/jpeg, images/png).

Get vs. Post

GET	POST
1) In case of Get request, only **limited amount of data** can be sent because data is sent in header.	In case of post request, **large amount of data** can be sent because data is sent in body.
2) Get request is **not secured** because data is exposed in URL bar.	Post request is **secured** because data is not exposed in URL bar.
3) Get request **can be bookmarked.**	Post request **cannot be bookmarked.**
4) Get request is **idempotent** . It means second request will be ignored until response of first request is delivered	Post request is **non-idempotent.**
5) Get request is **more efficient** and used more than Post.	Post request is **less efficient** and used less than get.

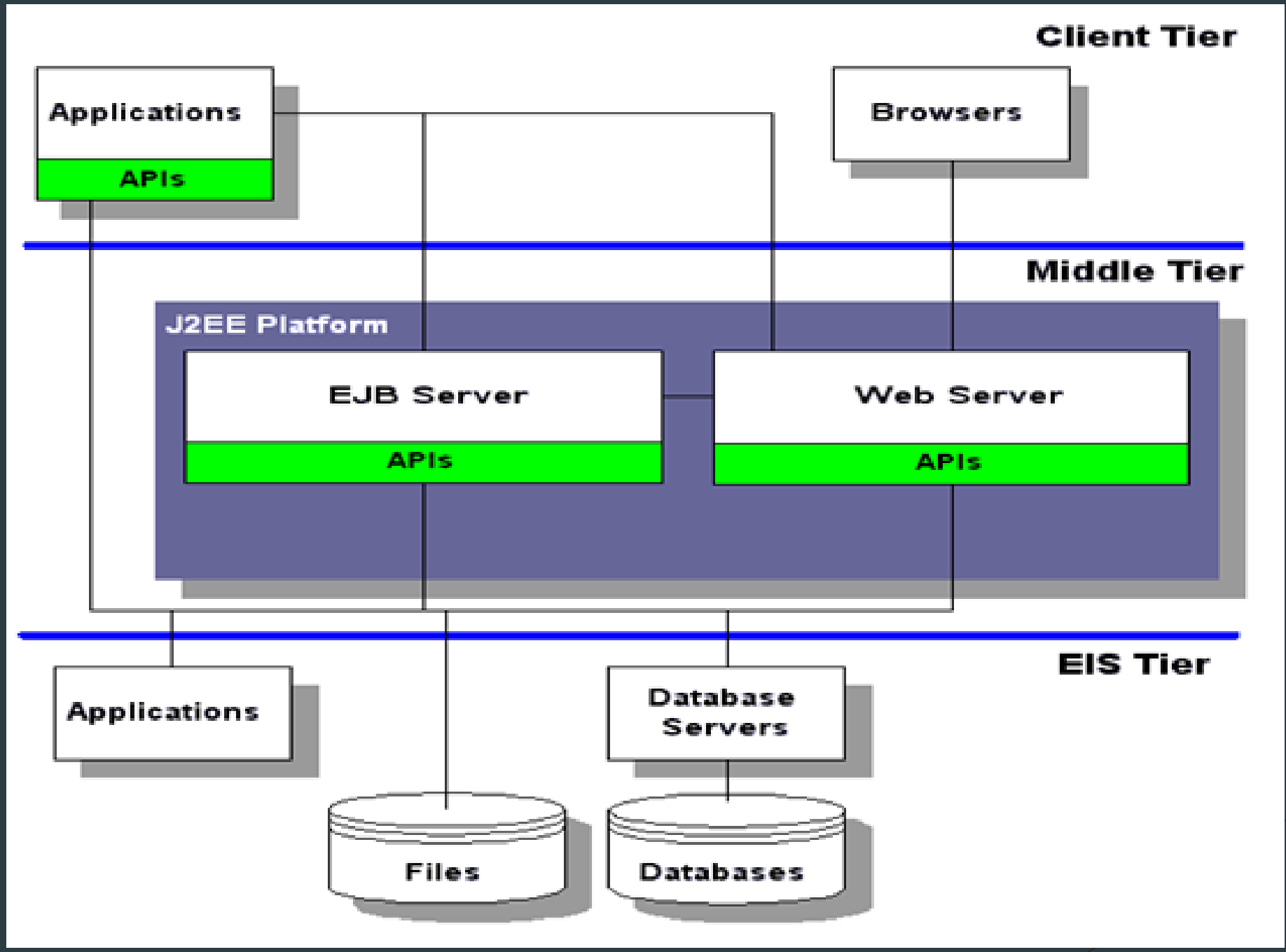
Web Server vs. Application Server

- **Web Server:** Web server contains only web or servlet container. It can be used for servlet, jsp, struts, jsf etc. It can't be used for EJB (**Apache Tomcat, Resin**).
- **Web Server Working:** It can respond to the client request in either of the following two possible ways:
  - Generating response by using the script and communicating with database.
  - Sending file to the client associated with the requested URL.
- **Important points:**
  - If the requested web page at the client side is not found, then web server will sends the HTTP response: Error 404 Not found.
  - When the web server searching the requested page if requested page is found then it will send to the client with an HTTP response.
  - If the client requests some other resources then web server will contact to application server and data is store for constructing the HTTP response.


Static
Data Store
Application
Data Store
Static Data
Request
Static Data
Response
Application Server
Servlet Request
Web Server
Web Container
Servlet Response
HTTP
Request
HTTP
Response
Other Services
Web Client

Application Server

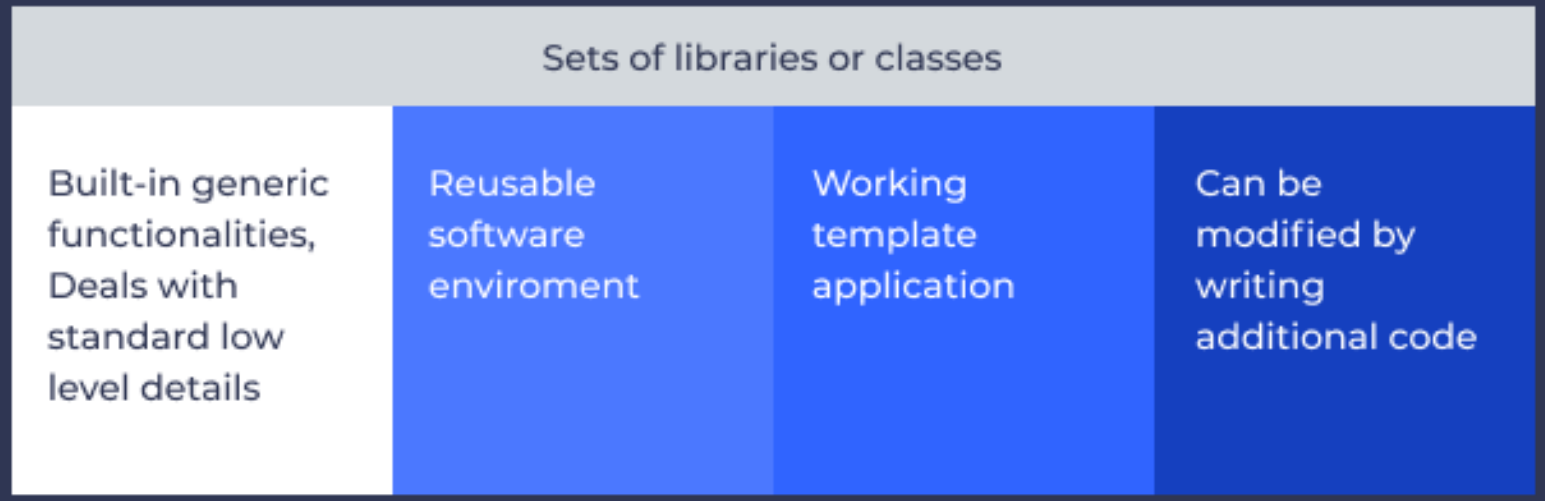
- **Application server:** contains Web and EJB containers. It can be used for servlet, jsp, struts, jsf, ejb etc. It is a component based product that lies in the middle-tier of a server centric architecture.
- **The Example of Application Servers are:**
  - **JBoss:** Open-source server from JBoss community.
  - **Glassfish:** Provided by Sun Microsystem. Now acquired by Oracle.
  - **Weblogic:** Provided by Oracle. It more secured.
  - **Websphere:** Provided by IBM.


Client Tier
Applications
APIs
Browsers
Middle Tier
J2EE Platform
EJB Server
APIs
Web Server
APIs
EIS Tier
Applications
Database Servers
Files
Databases

Difference Between JAR WAR and EAR

- **JAR file:** is a file with Java classes, associated metadata and resources such as text, images aggregated into one file *(Java Archive)*.
- **WAR file:** is a file that is used to distribute a collection of JAR files, JSP, Servlet, XML files, static web pages like HTML and other resources that constitute a web application. Thus, this explains the main difference between JAR and WAR Files *(Web Application Resource)*.
- **EAR file:** is a Java EE file that packs one or more modules into a single archive to deploy them on to an application server *(Enterprise Application Archive)*.

Software Framework



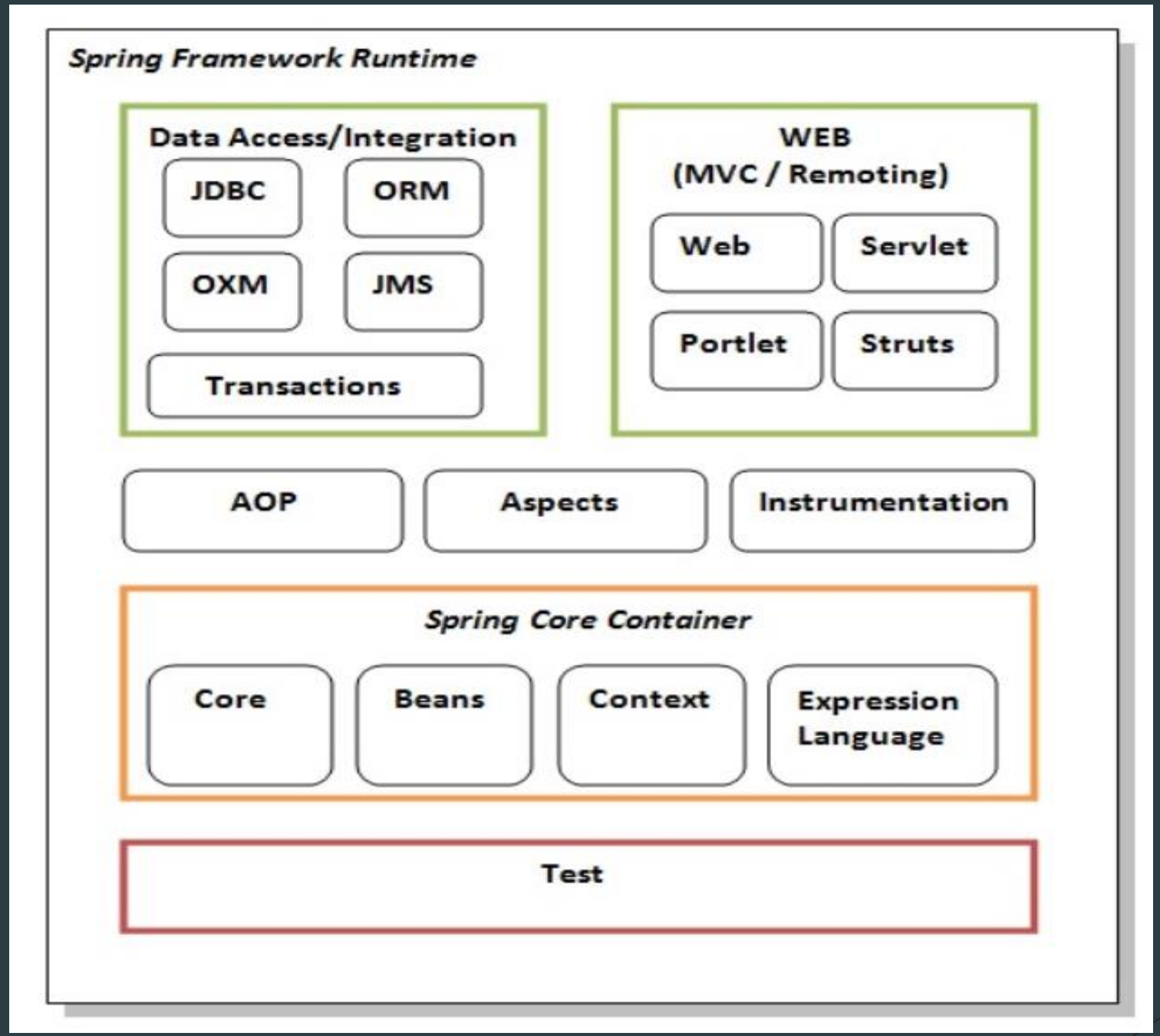
Spring Tutorial

- **Spring framework:** is an open source Java platform that provides comprehensive infrastructure support for developing robust Java applications very easily and very rapidly.
- **History:** It was developed by Rod Johnson in 2003. Spring framework makes the easy development of JavaEE application.
- **Benefits of using Spring Framework:**
  - **POJO Based:** not mandatory application server but you can using servlet container such as Tomcat
  - **Modular**
  - **Integration with existing frameworks:** such as Struts, Hibernate, JSP, JSF
  - **Testability**
  - **Web MVC**
  - **Central Exception Handling**
  - **Lightweight**
  - **Transaction management**

Spring Modules

Spring is modular, allowing you to pick and choose which modules are applicable to you, without having to bring in the rest.

- Spring Core Container
- Data Access / Integration
- Web : Web, Web-Servlet, Web-Struts and Web-Portlet.
- AOP, Aspects and Instrumentation
- Test


Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
WEB
(MVC / Remoting)
Web
Servlet
Portlet
Struts
AOP
Aspects
Instrumentation
Spring Core Container
Core
Beans
Context
Expression Language
Test

IoC Container

- **The IoC container:** is responsible to instantiate, configure and assemble the objects. The IoC container gets information's from the XML file and works accordingly. The main tasks performed by IoC container are:
  - to instantiate the application class
  - to configure the object
  - to assemble the dependencies between the objects

- **There are two types of IoC containers:**
  - **BeanFactory**
  - **ApplicationContext**

Bean Factory Container

- This is the simplest container providing the basic support for DI and defined by the *org.springframework.beans.factory.BeanFactory* interface.
- There are a number of implementations of the *BeanFactory* interface that are come straight out-of-the-box with Spring. The most commonly used *BeanFactory* implementation is the **XmlBeanFactory** class. This container reads the configuration metadata from an XML file and uses it to create a fully configured system or application.
- The BeanFactory is usually preferred where the resources are limited like mobile devices or applet-based applications. Thus, use an *ApplicationContext* unless you have a good reason for not doing so.

Application Context Container

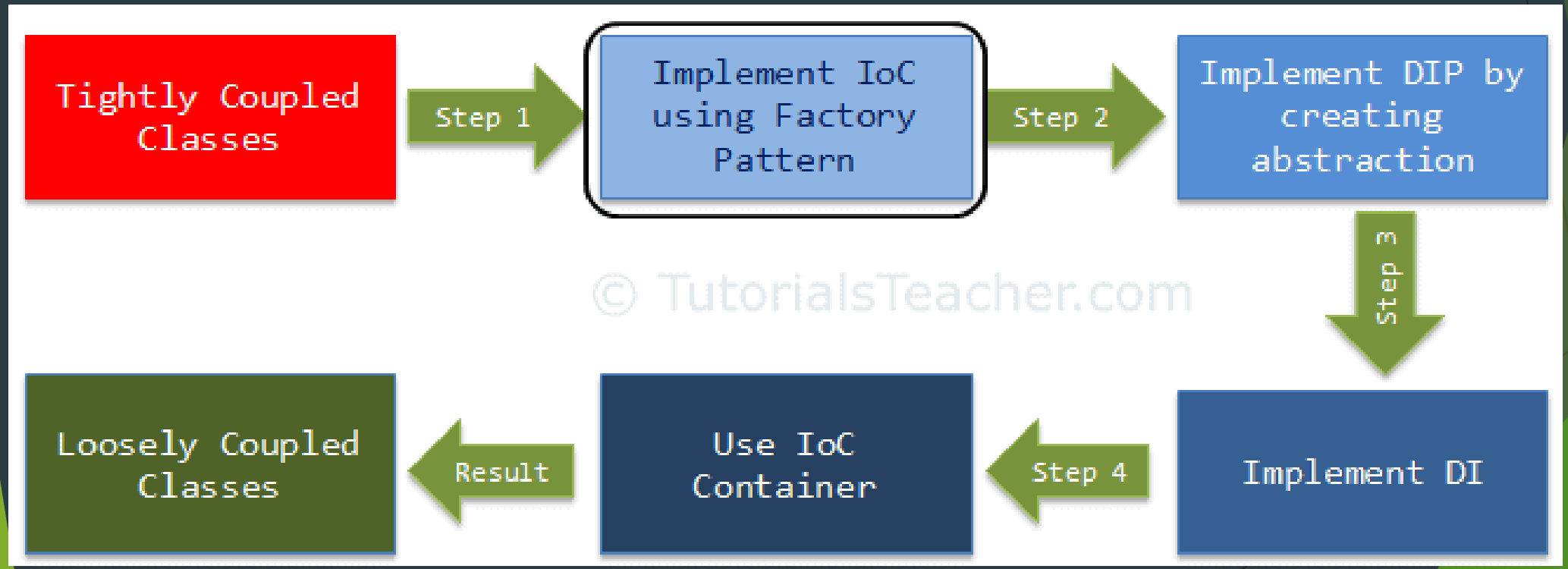
- **Application Context:** is Spring's advanced container. Similar to BeanFactory, it can load bean definitions, wire beans together, and dispense beans upon request. Additionally, it adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners. This container is defined by *org.springframework.context.ApplicationContext* interface.
- **The most commonly used ApplicationContext implementations are:**
  - ***FileSystemXmlApplicationContext*** – This container loads the definitions of the beans from an XML file. Here you need to provide the full path of the XML bean configuration file to the constructor.
  - ***ClassPathXmlApplicationContext*** – This container loads the definitions of the beans from an XML file. Here you do not need to provide the full path of the XML file but you need to set CLASSPATH properly because this container will look like bean configuration XML file in CLASSPATH.
  - ***WebXmlApplicationContext*** – This container loads the XML file with definitions of all beans from within a web application.

BeanFactory vs ApplicationContext

- The *org.springframework.beans.factory.**BeanFactory*** and the *org.springframework.context.**ApplicationContext*** interfaces acts as the IoC container.
- The ApplicationContext interface is built on top of the BeanFactory interface. It adds some extra functionality than BeanFactory such as simple integration with Spring's AOP, message resource handling (for I18N), event propagation, application layer specific context (e.g. WebApplicationContext) for web application. So it is better to use ApplicationContext than BeanFactory.

Inversion of Control

- Inversion of Control (IoC) is a design principle, it is used to invert different kinds of controls in object-oriented design to achieve loose coupling. Here, controls refer to any additional responsibilities a class has, other than its main responsibility. This include control over the flow of an application, and control over the flow of an object creation or dependent object creation and binding.



Dependency Injection

- **Dependency Injection (DI):** is a design pattern used to implement IoC. It allows the creation of dependent objects outside of a class and provides those objects to a class through different ways. Using DI, we move the creation and binding of the dependent objects outside of the class that depends on them.
  - Easy to manage
  - Testing the application
  - Loosely coupled

- **Dependency Lookup:** is an approach where we get the resource after demand.
  - By new object
  - Or static method

- **Problems of Dependency Lookup:**
  - **Tight coupling:** If resource is changed, we need to perform a lot of modification in the code
  - **Not easy for testing**

Dependency Injection

- **Two ways to perform Dependency Injection in Spring framework:**
  - By Constructor
  - By Setter method

- **Difference between constructor and setter:**
  - Partial dependency: can be injected using setter injection but it is not possible by constructor. Suppose there are 3 properties in a class, having 3 arg constructor and setters methods. In such case, if you want to pass information for only one property, it is possible by setter method only.
  - Overriding: Setter injection overrides the constructor injection. If we use both constructor and setter injection, IOC container will use the setter injection.
  - Changes: We can easily change the value by setter injection. It doesn't create a new bean instance always like constructor. So setter injection is flexible than constructor injection.

Spring - Bean Definition

- A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container. These beans are created with the configuration metadata that you supply to the container.

Spring - Bean Scopes

- **Singleton:** the container creates a single instance of that bean, all requests for that bean name will return the same object from, which is cached (**default**).
- **Prototype:** A bean return a different instance every time it is requested from the container.

- ***Web aware:-*** *there are three additional scopes that are only available in a web-aware application context.*
- **Request:** This scopes a bean definition to an HTTP request.
- **Session:** This scopes a bean definition to an HTTP session.
- **Global-session:** This scopes a bean definition to a global HTTP session.

Bean Life Cycle

- When a bean is instantiated, it may be required to perform some initialization to get it into a usable state. Similarly, when the bean is no longer required and is removed from the container, some cleanup may be required.
- To define setup and teardown for a bean, we simply declare the <bean> with **initmethod** and/or **destroy-method** parameters. The init-method attribute specifies a method that is to be called on the bean immediately upon instantiation. Similarly, destroymethod specifies a method that is called just before a bean is removed from the container.
- **Initialization callbacks**: *org.springframework.beans.factory.InitializingBean interface specifies a single method*
- **Destruction callbacks:** *The org.springframework.beans.factory.DisposableBean interface specifies a single method*

BeanPostProcessor interface

- The **BeanPostProcessor** interface defines callback methods that you can implement to provide your own instantiation logic, dependency-resolution logic, etc. You can also implement some custom logic after the Spring container finishes instantiating, configuring, and initializing a bean by plugging in one or more BeanPostProcessor implementations.

Configuration Type

- XML Based Configuration
- Annotation Based Configuration
- Java Based Configuration

Annotation Based Configuration

- Starting from Spring 2.5 it became possible to configure the dependency injection using **annotations**.
- Annotation injection is performed before XML injection. Thus, the latter configuration will override the former for properties wired through both approaches.
- Annotation wiring is not turned on in the Spring container by default. So, before we can use annotation-based wiring.
  - @Required: applies to bean property setter methods.
  - @Autowired: apply to bean property setter methods, non-setter methods, constructor and properties.
  - @Qualifier: along with @Autowired can be used to remove the confusion by specifiying which exact bean will be wired.
  - JSR-250 Annotation: as a Java Specification Request, has the objective to define a set of annotations that address common semantic concepts and therefore can be used by many Java EE and Java SE components (PostConstruct, PreDestroy, Resources).

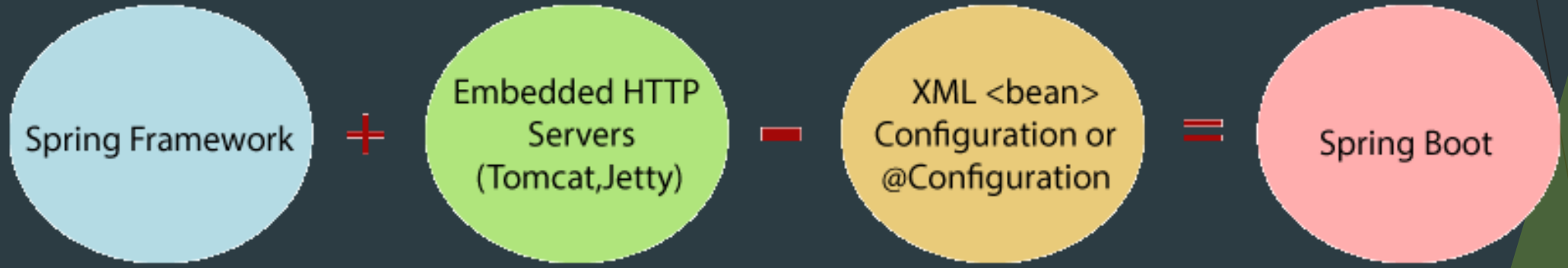
Java Based Configuration

- Java-based configuration option enables you to write most of your Spring configuration without XML
- Annotating a class with the **@Configuration** indicates that the class can be used by the Spring IoC container as a source of bean definitions.
  The **@Bean** annotation tells Spring that a method annotated with @Bean will return an object that should be registered as a bean in the Spring application context.

- https://docs.spring.io/spring-framework/docs/5.0.0.M1/spring-framework-reference/pdf/spring-framework-reference.pdf
- https://www.tutorialspoint.com/spring/index.htm
- https://www.javatpoint.com/spring-tutorial

Spring Boot

- Spring Boot is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based applications.



Advantages of Spring Boot

- It creates **stand-alone** Spring applications that can be started using Java **-jar**.
- It tests web applications easily with the help of different **Embedded** HTTP servers such as **Tomcat**, **Jetty**, etc. We don't need to deploy WAR files.
- It provides opinionated '**starter**' POMs to simplify our Maven configuration.
- It provides **production-ready** features such as **metrics**, **health checks**, and **externalized configuration**.
- There is no requirement for **XML** configuration.
- It offers a **CLI** tool for developing and testing the Spring Boot application.
- It offers the number of **plug-ins**.
- It **increases** **productivity** and reduces development time.

Limitations of Spring Boot:-

- Spring Boot can use dependencies that are not going to be used in the application. These dependencies increase the size of the application.

Prerequisite of Spring Boot:-

- Java 1.8
- Maven 3.0+
- Spring Framework 5.0.0.BUILD-SNAPSHOT
- An IDE (Spring Tool Suite) is recommended.

Create Spring Boot Project

- **Spring Initializr:-** is a web-based tool provided by the Pivotal Web Service. With the help of **Spring Initializr**, we can easily generate the structure of the Spring Boot Project *(https://start.spring.io/)*.
- **Spring Tool Suite (STS) IDE :-**
- **Spring Boot CLI :-**

@SpringBootApplication

A single @SpringBootApplication annotation is used to enable the following annotations:

- **@EnableAutoConfiguration:** It enables the Spring Boot auto-configuration mechanism.
- **@ComponentScan:** It scans the package where the application is located.
- **@Configuration:** It allows us to register extra beans in the context or import additional configuration classes.

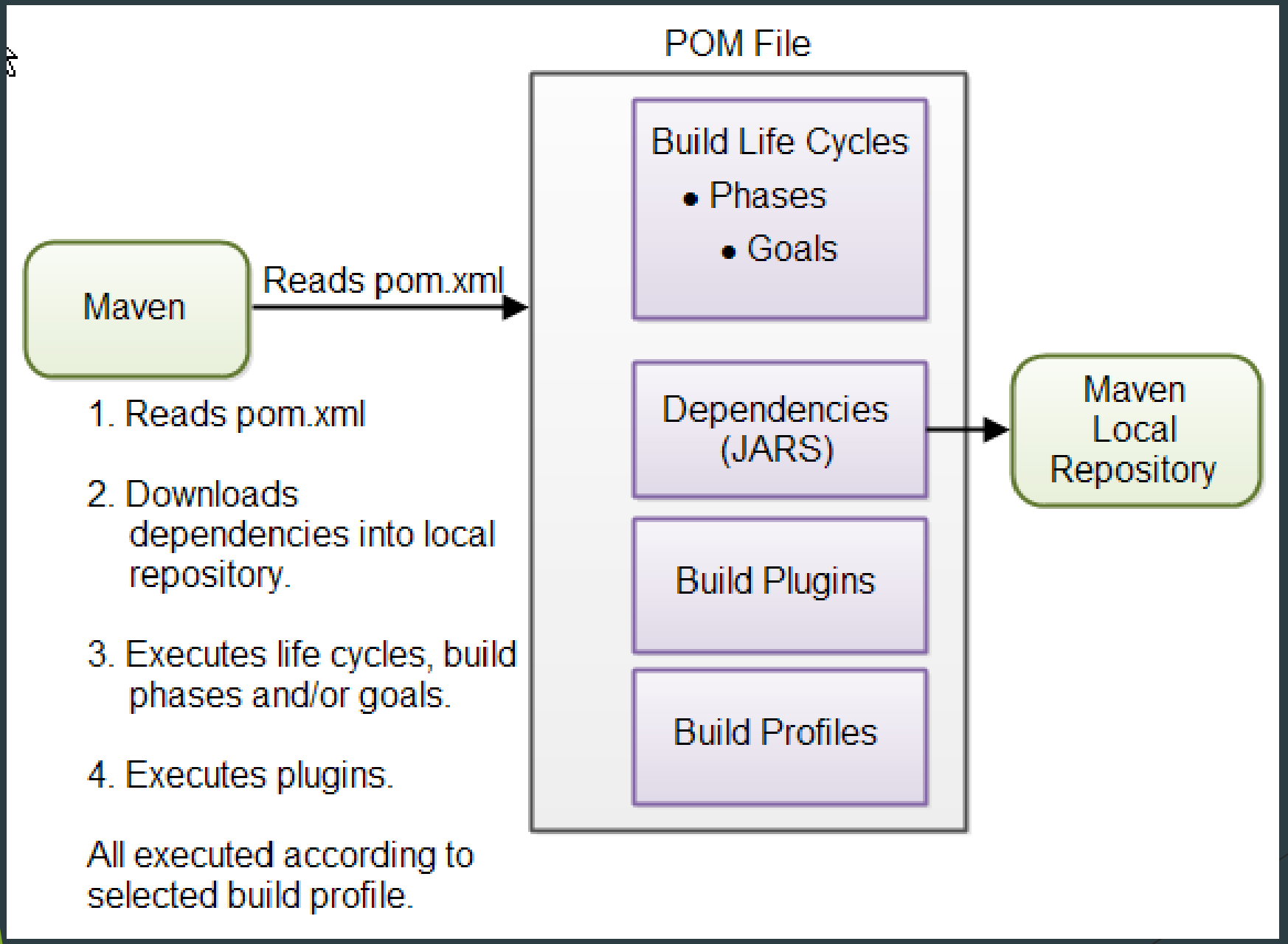
Maven Overview

- Maven is a powerful *project management tool* that is based on POM (project object model). It is used for projects build, dependency and documentation.
- It simplifies the build process like ANT. But it is too much advanced than ANT.
- **Ant** and **Maven** both are build tools provided by Apache, but maven is Declarative, Framework, Reusable, more preferred than ANT
- http://maven.apache.org/

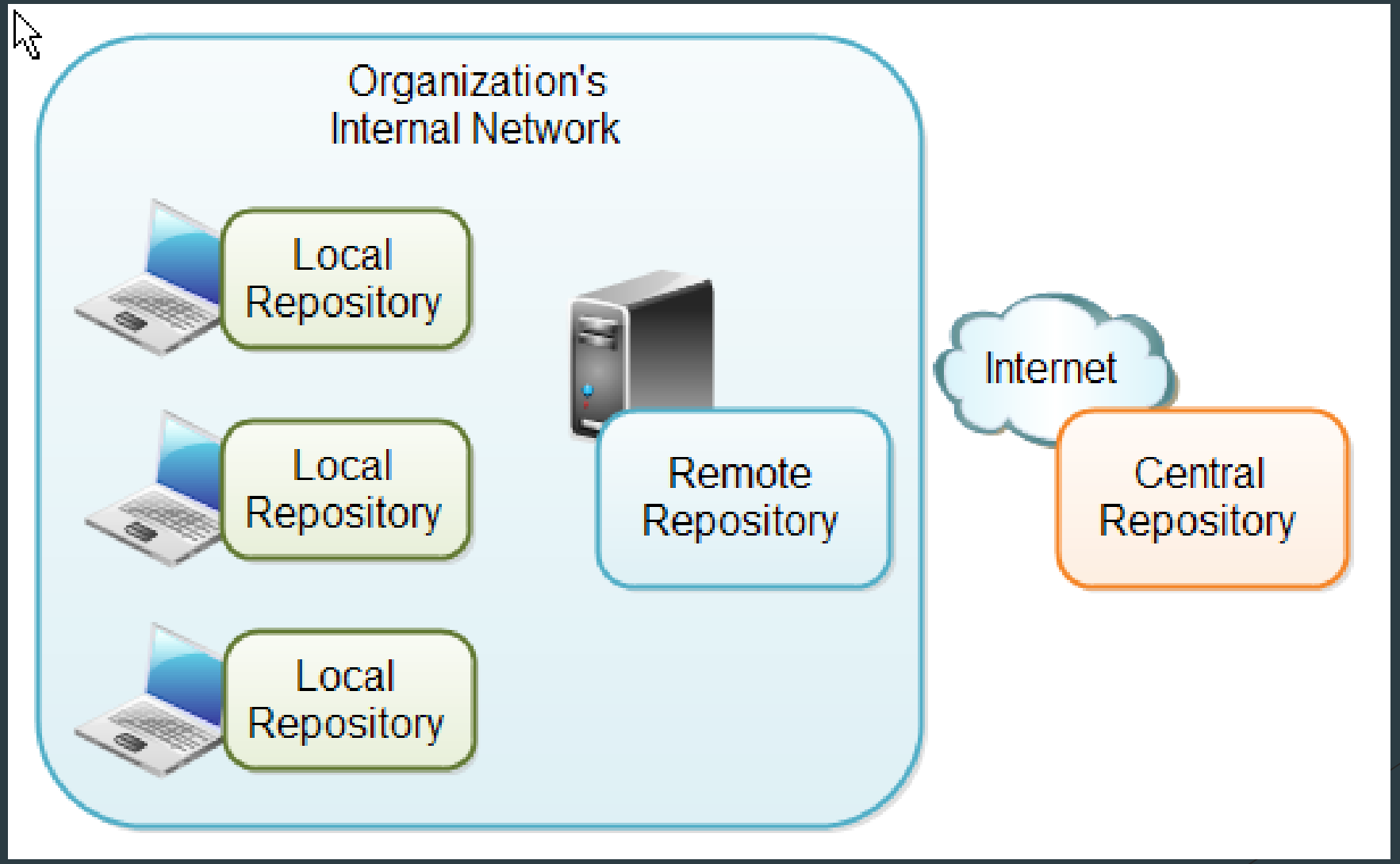
What is a Build Tool?

A build tool is a tool that automates everything related to building the software project. Building a software project typically includes one or more of these activities:

- Generating source code (if auto-generated code is used in the project).
- Generating documentation from the source code.
- Compiling source code.
- Packaging compiled code into JAR files or ZIP files.
- Installing the packaged code on a server, in a repository or somewhere else.


POM File
Build Life Cycles
• Phases
• Goals
Maven
Reads pom.xml
Dependencies (JARS)
Maven Local Repository
Build Plugins
Build Profiles
1. Reads pom.xml
2. Downloads dependencies into local repository.
3. Executes life cycles, build phases and/or goals.
4. Executes plugins.
All executed according to selected build profile.

Maven Repositories



Spring Boot Dependency Management

- Spring Boot manages dependencies and configuration automatically. Each release of Spring Boot provides a list of dependencies that it supports. The list of dependencies is available as a part of the **Bills of Materials** (spring-boot-dependencies) that can be used with **Maven**.
- we don't need to specify the version of the dependencies in our configuration. Spring Boot manages itself. Spring Boot upgrades all dependencies automatically in a consistent way when we update the Spring Boot version.
- **Advantages of Dependency Management:**
  - It provides the centralization of dependency information by specifying the Spring Boot version in one place. It helps when we switch from one version to another.
  - It avoids mismatch of different versions of Spring Boot libraries.

Application Properties

- Spring Boot comes with a built-in mechanism for application configuration using a file called **application.properties**.
- Spring Boot provides various properties that can be configured in the **application.properties** file.
- The properties have default values.
- Spring Boot also allows us to define our own property if required.
- The application.properties file allows us to run an application in a **different environment.**
- Spring Boot provides another file to configure the properties is called **yml** file, Instead of using the application.properties file.
- There are **sixteen** categories of Spring Boot Property:-
  - Core Properties
  - Cache Properties
  - Mail Properties
  - ……

Starters

- **Spring Boot Starters** are the **dependency descriptors.**
- **Spring Boot** provides a number of **starters** that allow us to add jars in the classpath. Spring Boot built-in **starters** make development easier and rapid.
- All the starters follow a similar naming pattern: **spring-boot-starter-*.**
- Spring Boot provides the following application starters under the **org.springframework.boot** group.

spring-boot-starter-web	It is used for building the web application, including RESTful applications using Spring MVC. It uses Tomcat as the default embedded container.
spring-boot-starter-thymeleaf	It is used to build MVC web applications using Thymeleaf views.
spring-boot-starter-mail	It is used to support Java Mail and Spring Framework's email sending.

Spring Boot Starter Parent

- The spring-boot-starter-parent is provides default configurations for our applications. It is used internally by all dependencies.
- Parent Poms allow us to manage the following things for multiple child projects and modules:
  - **Configuration:** It allows us to maintain consistency of Java Version and other related properties.
  - **Dependency Management:** It controls the versions of dependencies to avoid conflict.
  - Source encoding
  - Default Java Version
  - Resource filtering
  - Default plugin configuration.

Application Runner

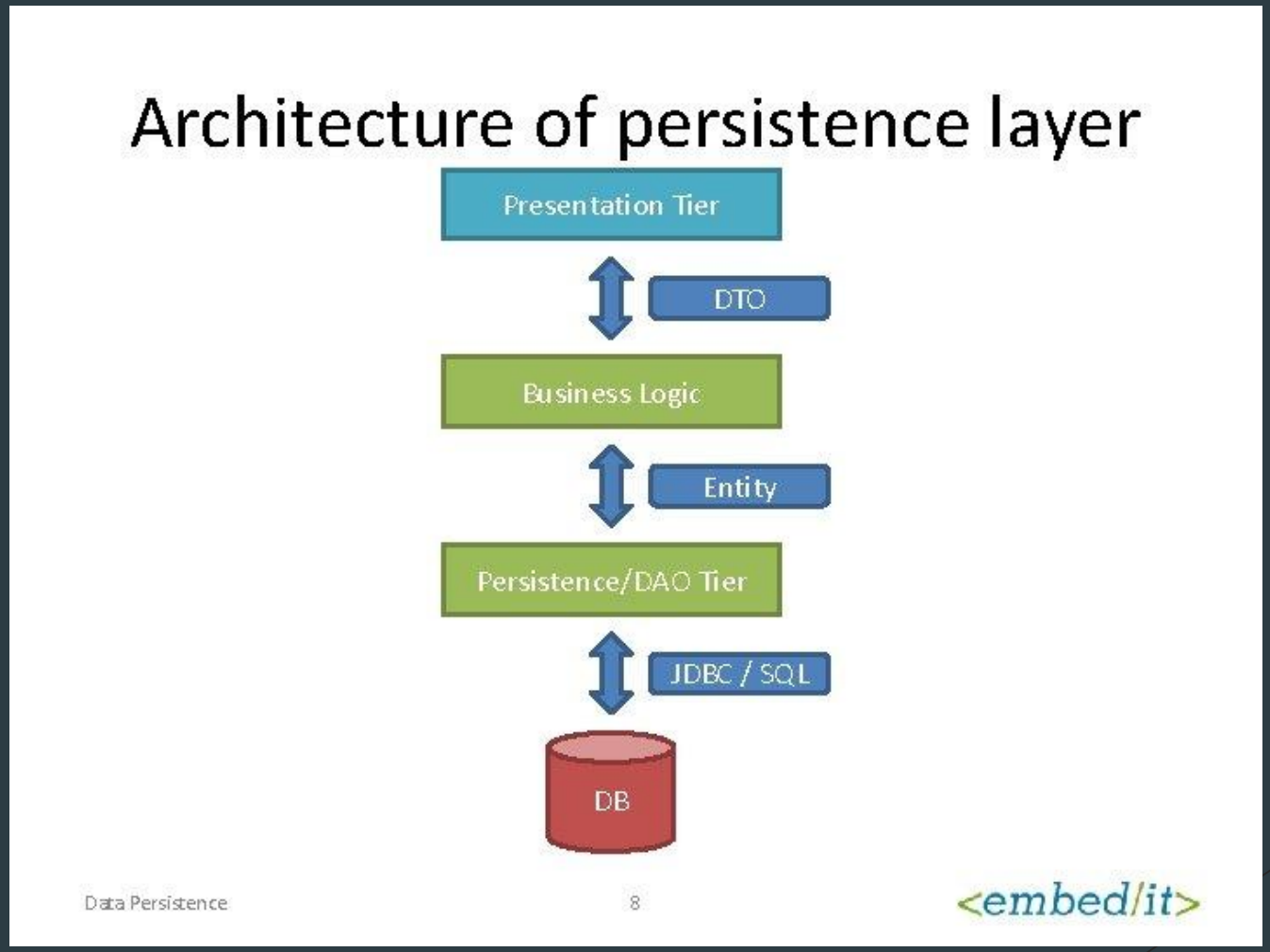
- Spring Boot provides two interfaces, CommandLineRunner and ApplicationRunner, to run specific pieces of code when an application is fully started. These interfaces get called just before run() once SpringApplication completes.

Session Content (4)

- Persistence Layer
- Spring Boot JDBC
- JDBC Connection Pool
- HikariCP
- Dependency Injection Review

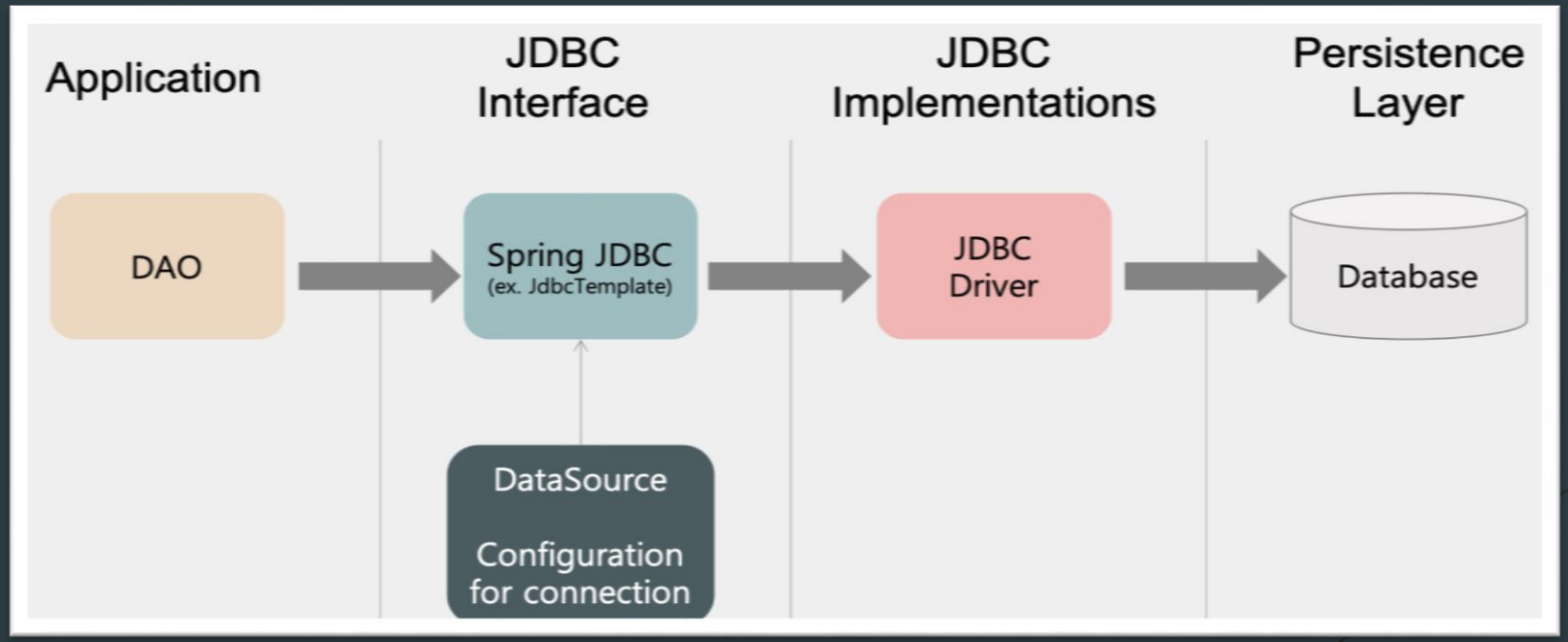
Persistence layers

- Any software layer that makes it easier for a program to persist its state is generically called a persistence layer. most persistence layers will not achieve persistence directly but will use an underlying database management system.



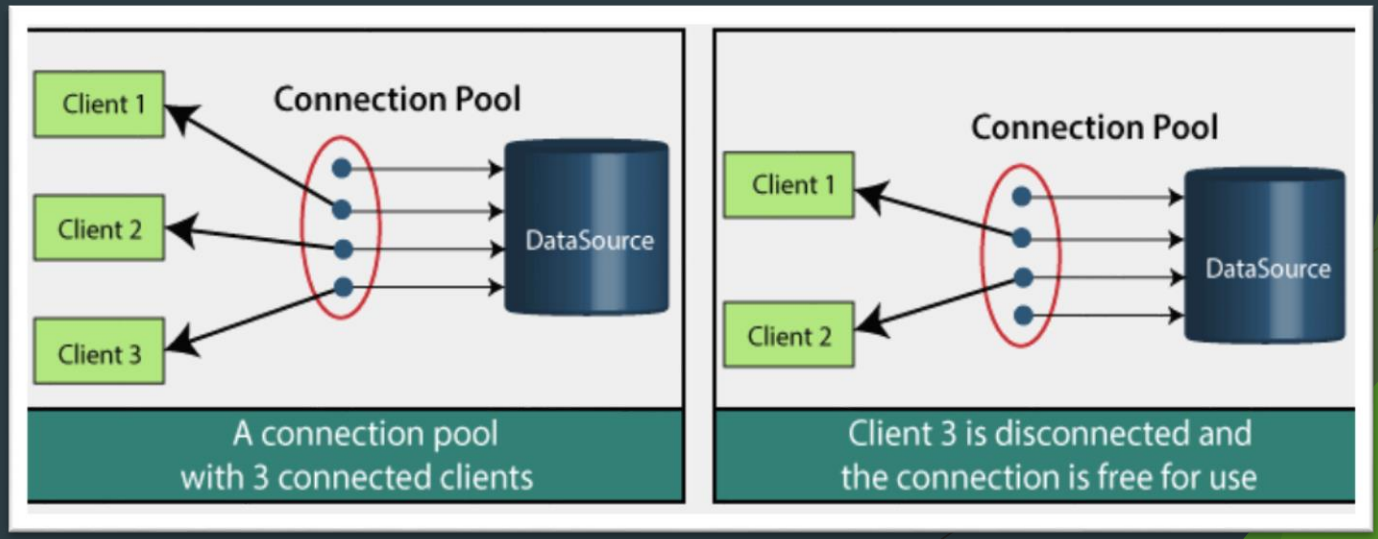
Spring Boot JDBC

- **Spring Boot JDBC** provides starter and libraries for connecting an application with JDBC.
- In Spring Boot JDBC, the database related beans such as **DataSource, JdbcTemplate,** and **NamedParameterJdbcTemplate** auto-configures and created during the startup. We can autowire these classes if we want to use it.
- In **application.properties** file, we configure **DataSource** and **connection pooling**.



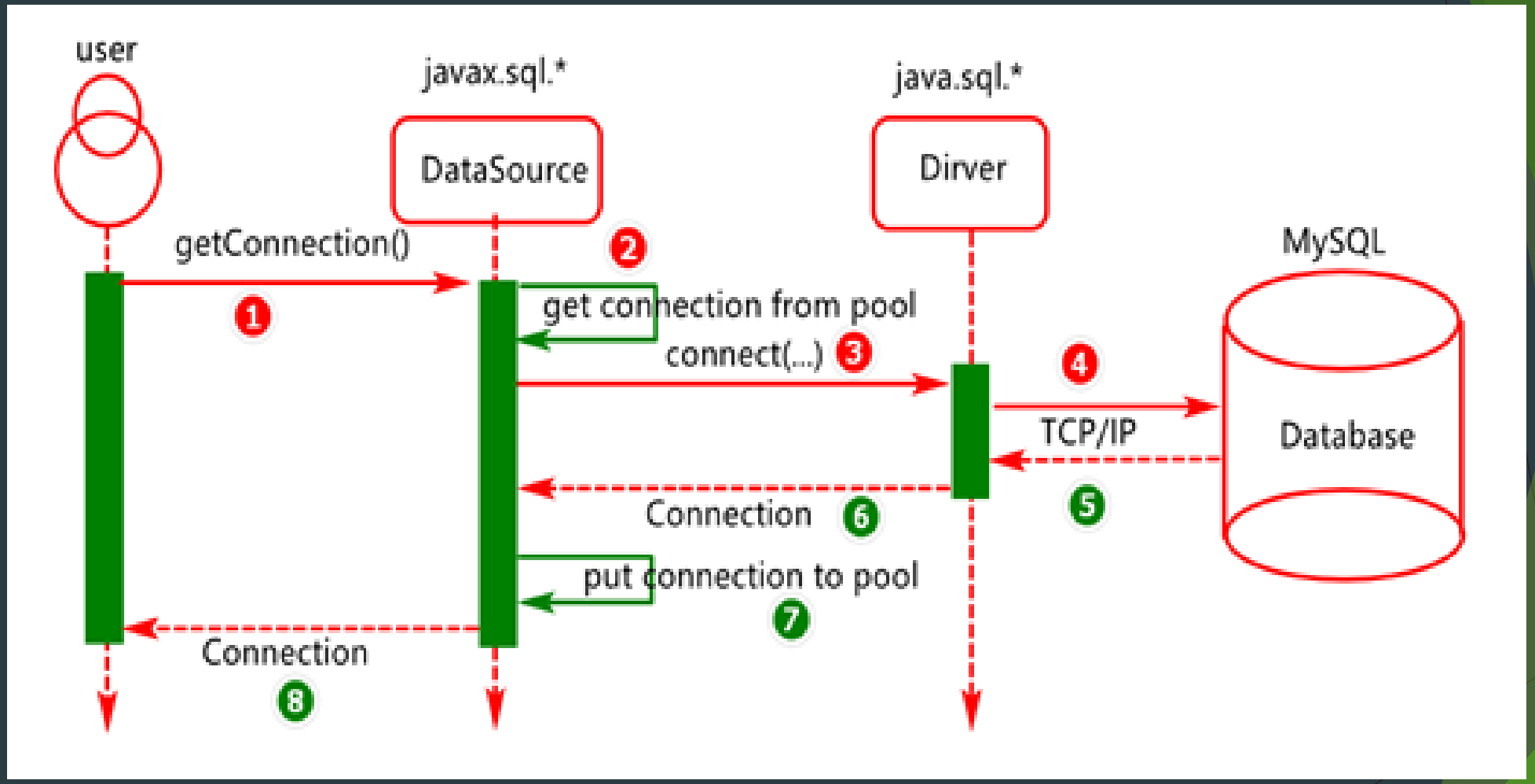
JDBC Connection Pooling

- **JDBC connection pooling** is a mechanism that manages **multiple** database connection requests. In other words, it facilitates connection reuse, a memory cache of database connections, called a **connection pool.** A connection pooling module maintains it as a layer on top of any standard JDBC driver product.
- It increases the speed of data access and reduces the number of database connections for an application. It also improves the performance of an application. Connection pool performs the following tasks:
  - Manage available connection
  - Allocate new connection
  - Close connection



HikariCP

- HikariCP is a JDBC DataSource implementation that provides a connection pooling mechanism.
- It provides enterprise-ready features and better performance.
- The default connection pool in Spring Boot 2 is **HikariCP**.
- If the HikariCP is present on the classpath, the Spring Boot automatically configures it.
- If the HikariCP is not found on the classpath, Spring Boot looks for the **Tomcat JDBC Connection Pool.** If it is on the classpath Spring Boot, pick it up.
- If both the above options are not available, Spring Boot chooses **Apache Commons DBCP2** as the JDBC connection pool.
- We can also configure a connection pool manually, by exclude **HikariCP** dependency and add the **tomcat-jdbc** dependency in the pom.xml file.


user
javax.sql.*
java.sql.*
DataSource
Dirver
MySQL
getConnection()
1
2
get connection from pool
connect(...)
3
4
TCP/IP
Database
5
Connection
6
put connection to pool
7
Connection
8

Session Content (5)

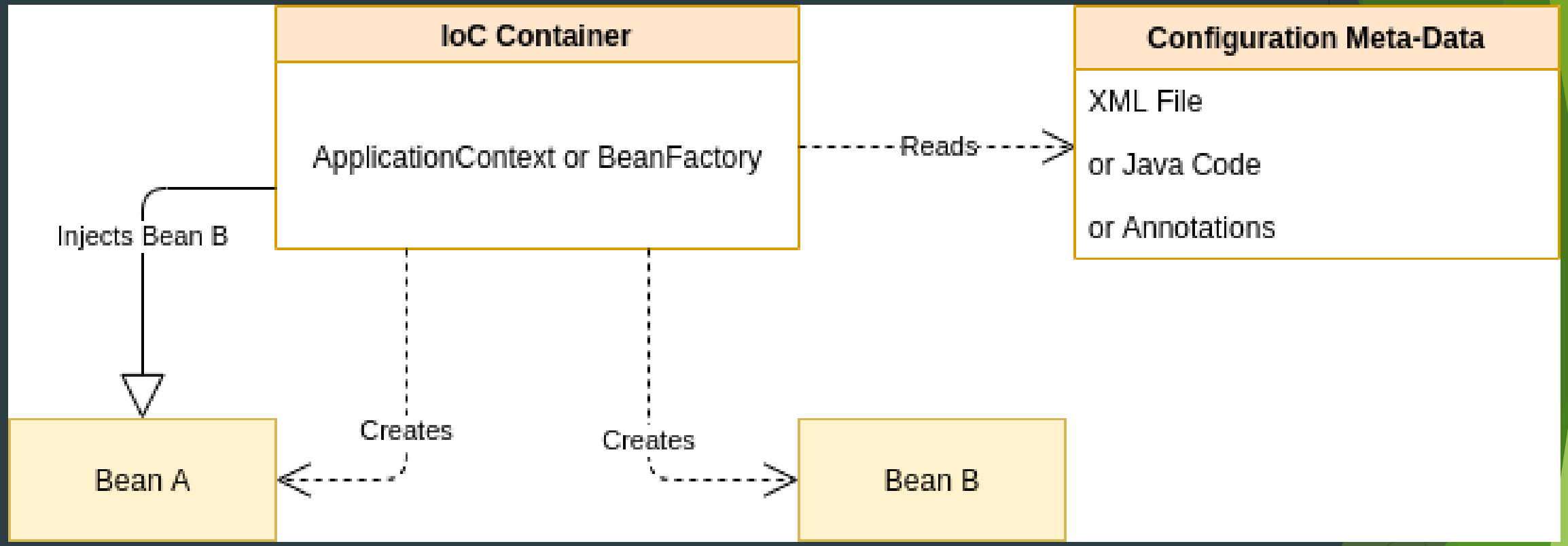
- Dependency Injection Review
- Spring Boot Data JDBC
- ORM (Object Relational Mapping)
- RESTful web service

Spring Boot Annotations

- Spring Boot Annotations is a form of metadata that provides data about a program. In other words, annotations are used to provide supplemental information about a program. It is not a part of the application that we develop. It does not have a direct effect on the operation of the code they annotate. It does not change the action of the compiled program.

Some Annotations

- **@Autowired:** It is used to autowire spring bean on setter methods, instance variable, and constructor. When we use @Autowired annotation, the spring container auto-wires the bean by matching data-type.
- **@Component:** It is a class-level annotation. It is used to mark a Java class as a bean. A Java class annotated with **@Component** is found during the classpath. The Spring Framework pick it up and configure it in the application context as a **Spring Bean**.
- **@ComponentScan:** It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.
- **@Configuration:** It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions.
- **@Bean:** It is a method-level annotation. It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed by Spring Container.
- **@Service:** It is a class-level annotation. It tells the Spring that class contains the **business logic**.
- **@Repository:** It is a class-level annotation. The repository is a **DAOs** (Data Access Object) that access the database directly. The repository does all the operations related to the database.
- **@Primary:** to give higher preference to a bean when there are multiple beans of the same type.


IoC Container
ApplicationContext or BeanFactory
Configuration Meta-Data
XML File
or Java Code
or Annotations
Reads
Injects Bean B
Creates
Creates
Bean A
Bean B

Spring Data JDBC

- **The Spring Data JDBC project belongs to Spring Data family and provides abstractions for the JDBC based Data Access Layer**. It provides easy to use Object Relational Mapping (ORM) framework to work with databases. That means, Spring Data JDBC supports using entity objects and repositories. However, it reduces a lot of complexities which are introduced by JPA .
  - Name of the POJO is same as that of table. Otherwise, it uses @Table annotation to refer to the actual table name.
  - The POJO has a primary key and that is annotated as @Id.
  - All the persistable fields in the POJO have same naming as that of database table columns. Else, we can use @Column annotation to provide column name.
  - @Id field in POJO is null – The POJO is inserted as a new record in table with next auto incrementing value.
  - @Id field in POJO is not null – the operation is considered to be an UPDATE and it throws exception if the given primary key is not find in existing records.

ORM

- The **O**bject-**R**elational **M**apping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C#, etc.
- **ORM Consists:-**
  - An API to perform basic CRUD operations on objects of persistent classes.
  - A language or API to specify queries that refer to classes and properties of classes.
  - A configurable facility for specifying mapping metadata.
  - A technique to interact with transactional objects to perform dirty checking, lazy association fetching, and other optimization functions.
- **Advantages:-**
  - Hides details of SQL queries from OO logic.
  - No need to deal with the database implementation.
  - Entities based on business concepts rather than database structure.
  - Transaction management and automatic key generation.
  - Fast development of application.
- **Java ORM Frameworks:-**
  - Hibernate, TopLink, EclipseLink, MyBatis, OpenJPA

Session Content (6)

- Spring Boot Starter Web
- Spring Boot Embedded Web Server
- **MVC Architecture**
- Web Services
- **RESTful web service**

Spring Boot Starter Web

- There are two important features of spring-boot-starter-web:
  - It is compatible for web development
  - Auto configuration
- Starter of Spring web uses Spring MVC, REST and Tomcat as a default embedded server.
- The spring-boot-starter-web auto-configures the following things that are required for the web development:
  - Dispatcher Servlet: responsible for correctly coordinating the *HttpRequests* to their right handlers.
  - Error Page
  - Web JARs for managing the static dependencies
  - Embedded servlet container
- Spring-boot-starter-web contains the following:
  - spring-boot-starter
  - jackson
  - spring-core
  - spring-mvc
  - spring-boot-starter-tomcat

Spring Boot Embedded Web Server

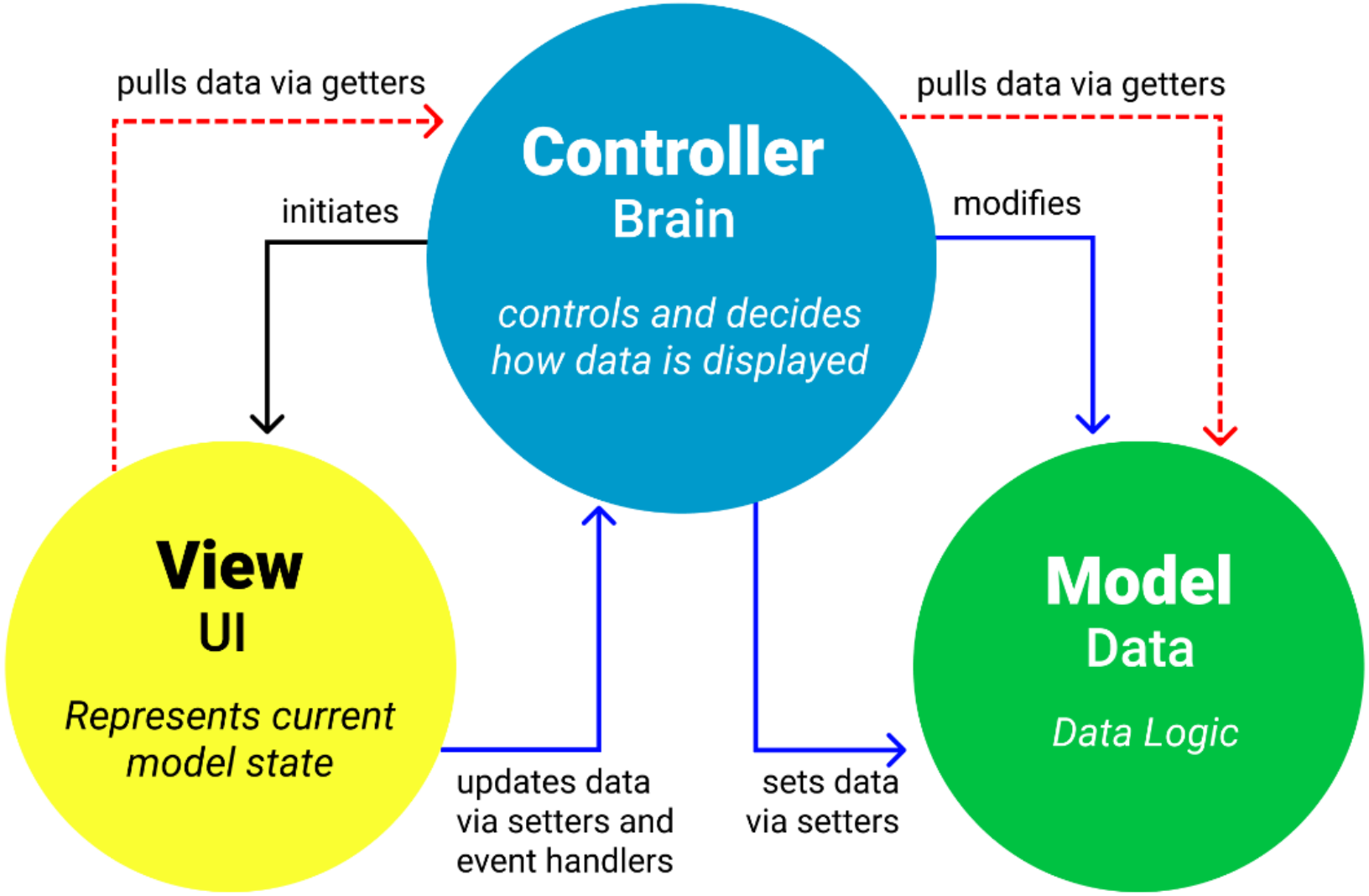
- Each Spring Boot application includes an embedded server. Embedded server is embedded as a part of deployable application. The advantage of embedded server is, we do not require pre-installed server in the environment.
- Default embedded server is **Tomcat**. Spring Boot also supports another two embedded servers:
  - Jetty Server
  - Undertow Server
- Jetty Server: It is an HTTP server and Servlet container that has the capability of serving static and dynamic content.
- If we want to add the Jetty server in the application, we need to add the **spring-boot-starter-jetty** dependency and exclude tomcat in our pom.xml file.
- Undertow: It is also an embedded web server like Jetty. It is written in Java and manage and sponsored by JBoss.
- If we want to add the Undertow server in the application, we need to add the **spring-boot-starter-undertow** dependency and exclude tomcat in our pom.xml file.
- We can also customize the behavior of the Undertow server by using the **application.properties** file.

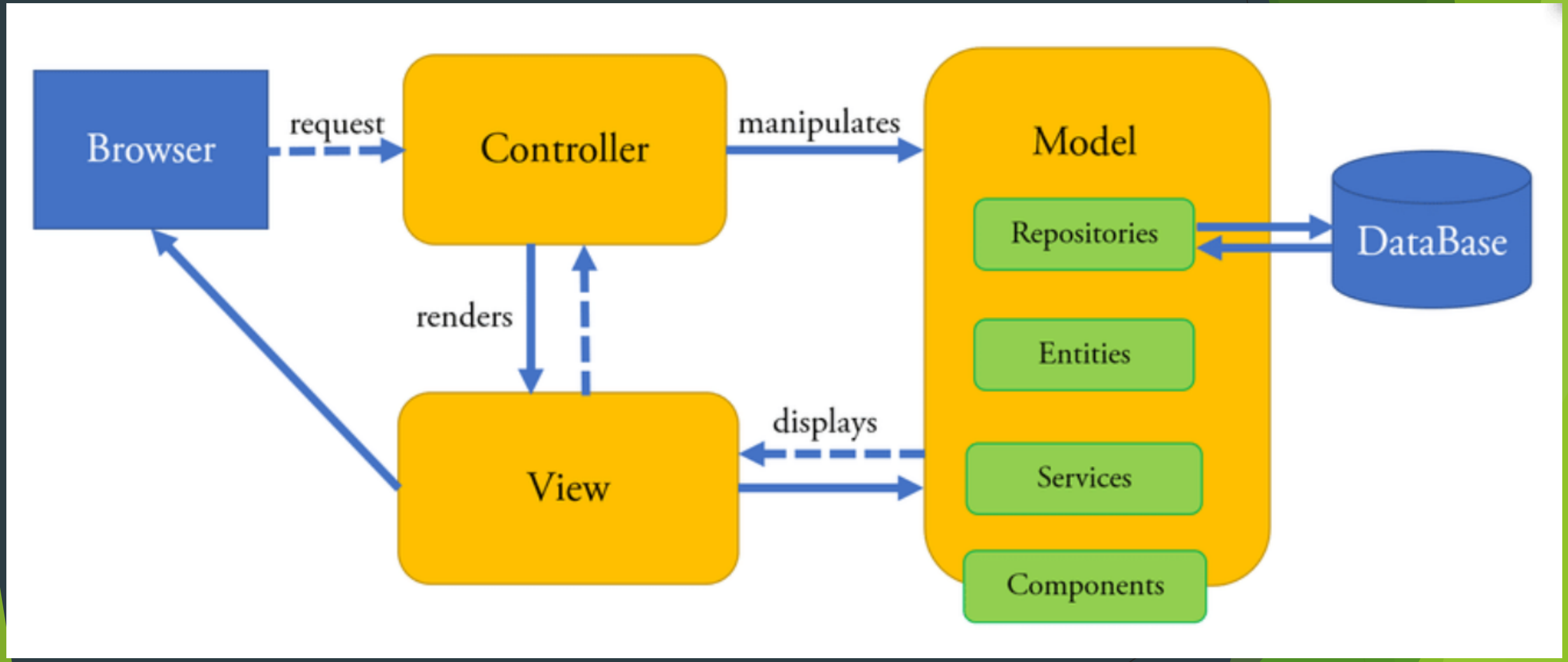
Web Services vs. Web Apps

- **Web application**
  - Provides service directly to user

- **Web service or (Web API)**
  - Provides service to other programs

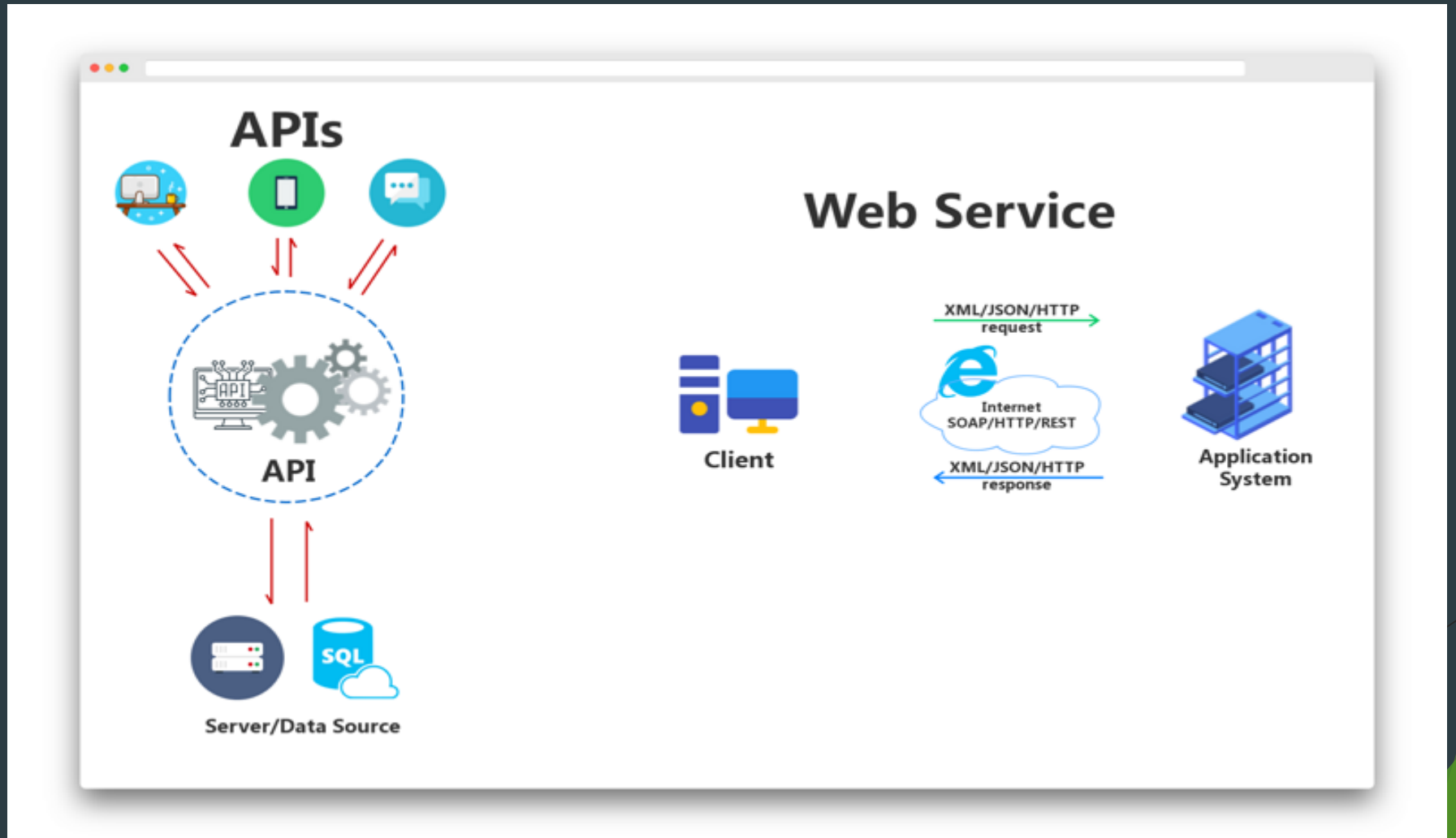
MVC Architecture Pattern

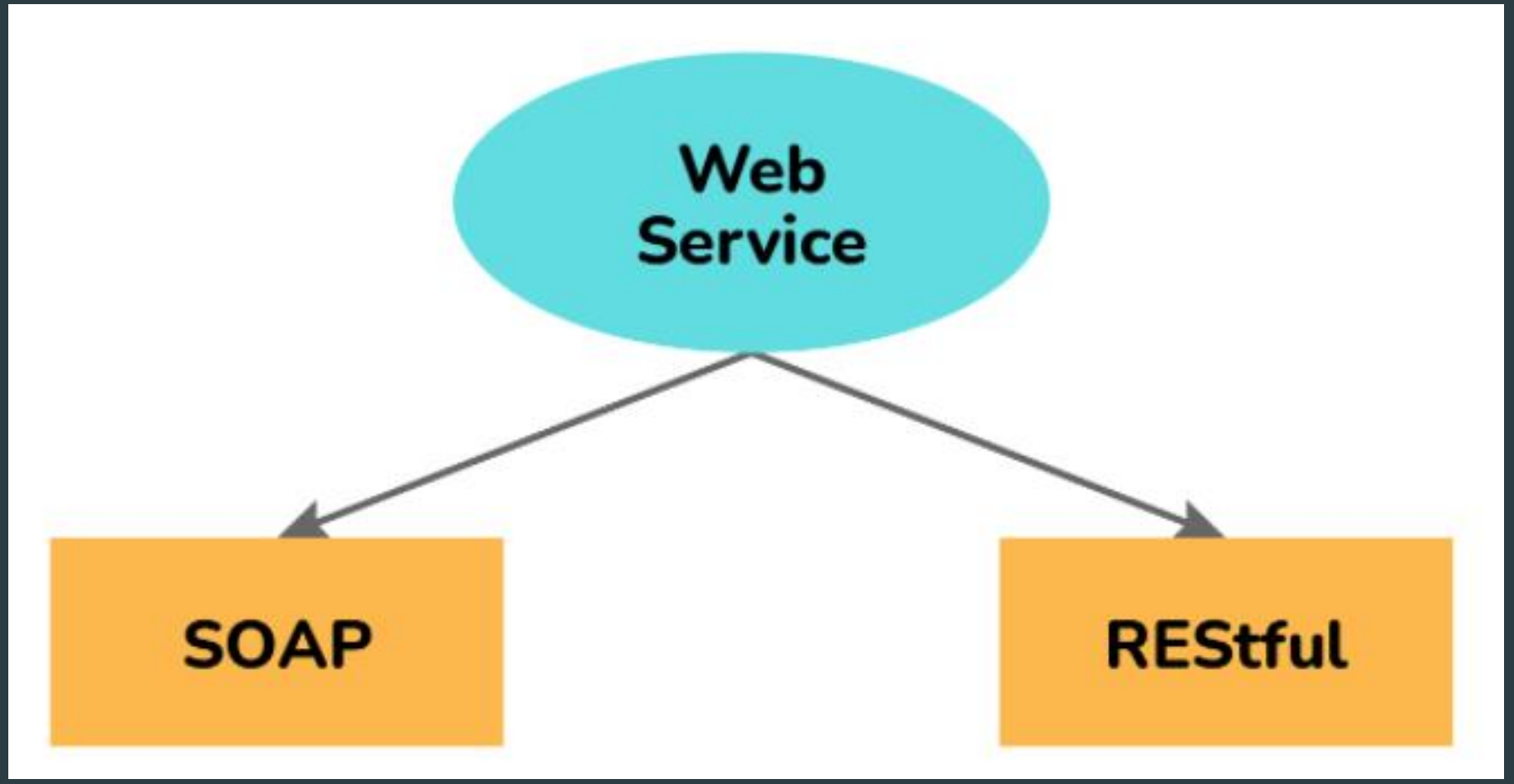



Browser
request
Controller
manipulates
Model
Repositories
DataBase
renders
Entities
displays
View
Services
Components

Web Services

- Web service is a technology to communicate one programming language with another. For example, java programming language can interact with PHP and .NET by using web services.




Web
Service
SOAP
RESTful

RESTful web services.

- REST stands for REpresentational State Transfer.
- REST is an architectural style not a protocol.
- **RESTful Advantage:-**
  - **Fast:** RESTful Web Services are fast because there is no strict specification like SOAP. It consumes less bandwidth and resource.
  - **Language and Platform independent:** RESTful web services can be written in any programming language and executed in any platform.
- **RESTful Methods:-**
  - **GET:** This method helps in offering read-only access for the resources.
  - **POST:** This method is implemented for creating a new resource.
  - **PUT:** This method is implemented for updating an existing resource or creating a fresh one.
  - **DELETE:** This method is implemented for removing a resource.

RESTful Standard Actions

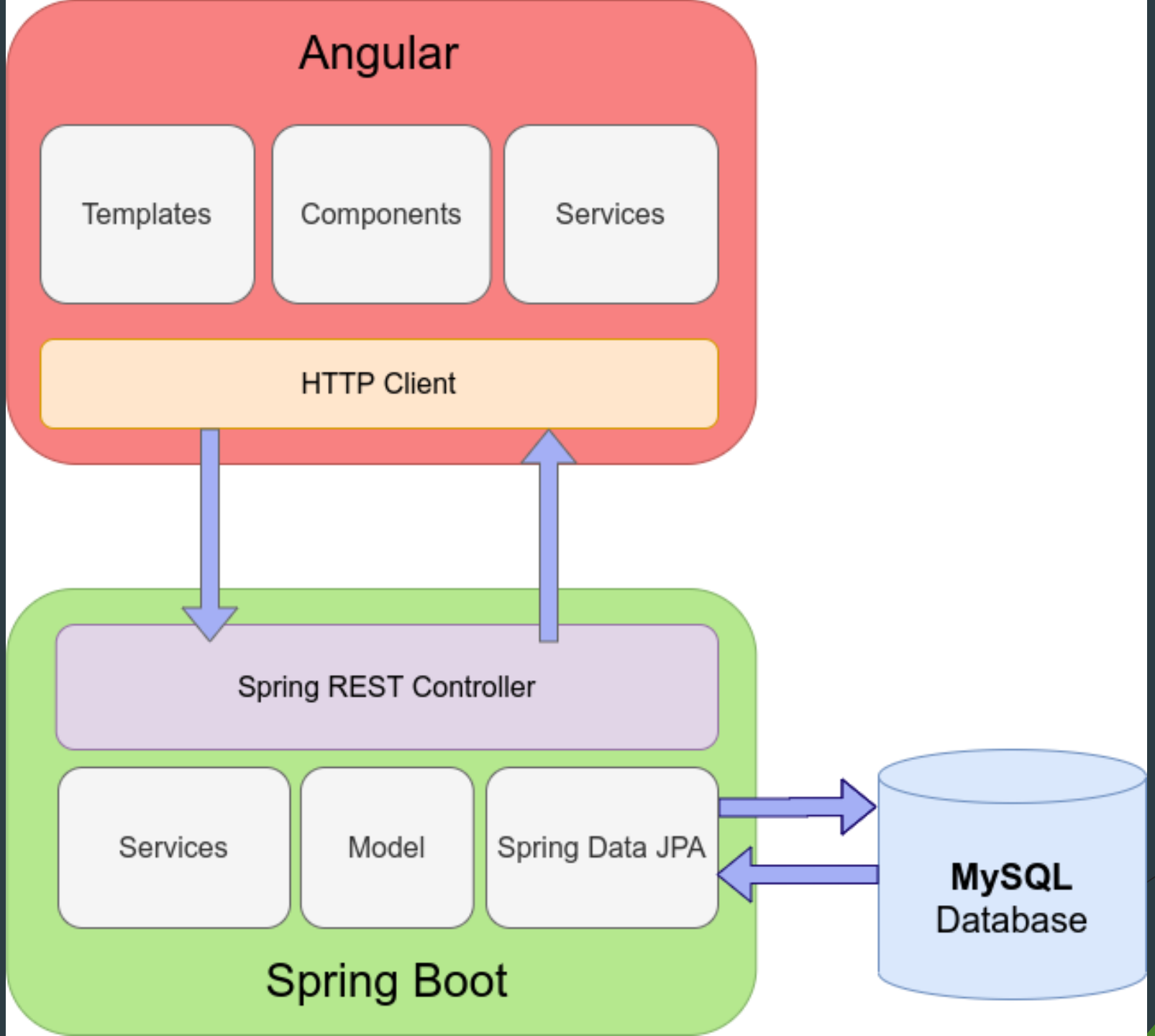
- **POST /users:** It creates a user.
- **GET /users/{id}:** It retrieves the detail of a user.
- **GET /users:** It retrieves the detail of all users.
- **DELETE /users:** It deletes all users.
- **DELETE /users/{id}:** It deletes a user.
- **GET /users/{id}/posts/post_id:** It retrieve the detail of a specific post.
- **POST / users/{id}/ posts:** It creates a post of the user.

Standard status code

- **404:** RESOURCE NOT FOUND
- **200:** SUCCESS
- **201:** CREATED
- **401:** UNAUTHORIZED
- **500:** SERVER ERROR

RESTful Service Constraints

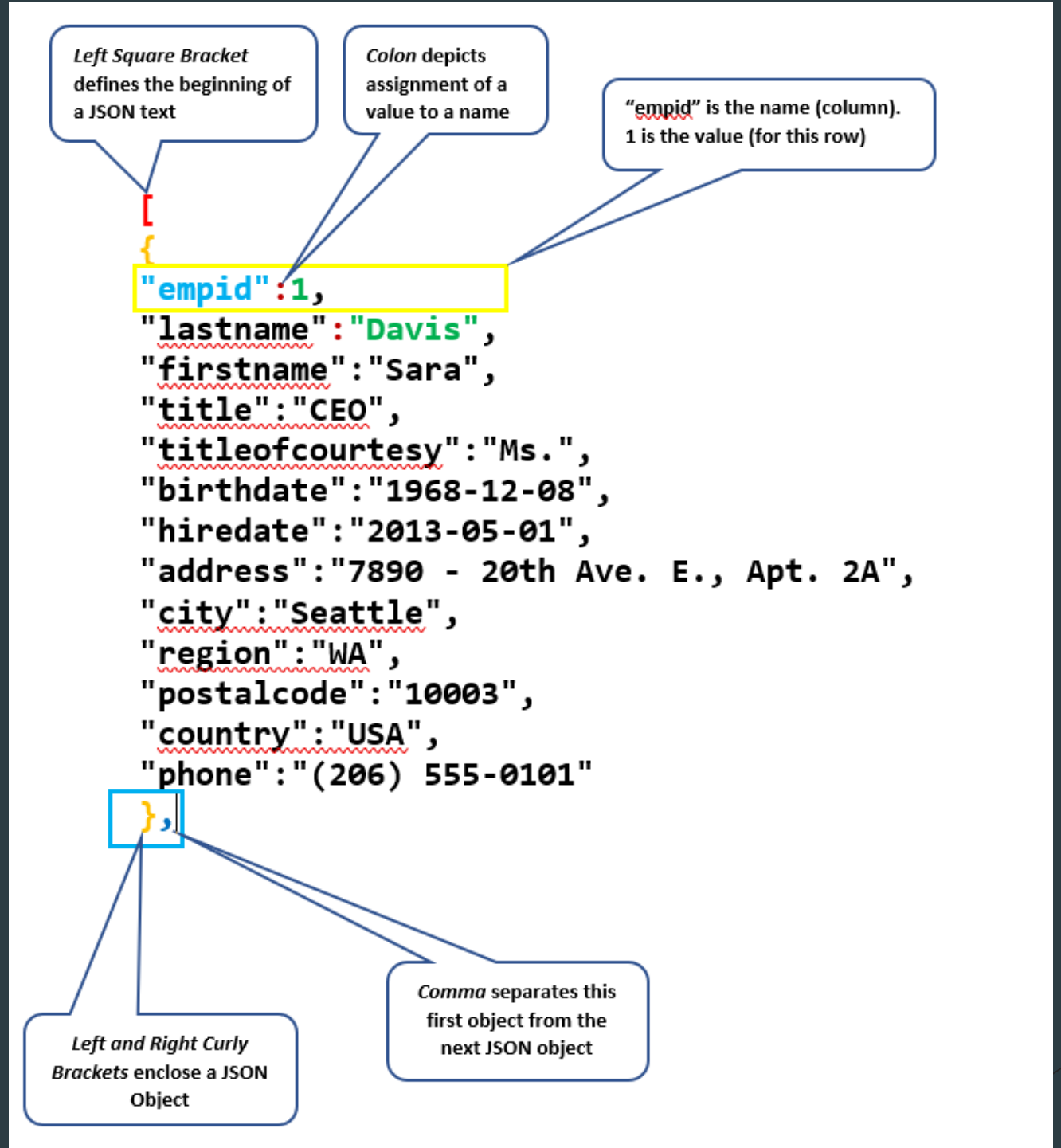
- There must be a service **producer** and service **consumer**.
- The service is **stateless**.
- The service result must be **cacheable**.
- The interface is **uniform** and exposing resources.
- The service should assume a **layered** architecture.


Angular
Templates
Components
Services
HTTP Client
Spring REST Controller
Services
Model
Spring Data JPA
Spring Boot
MySQL
Database

JSON

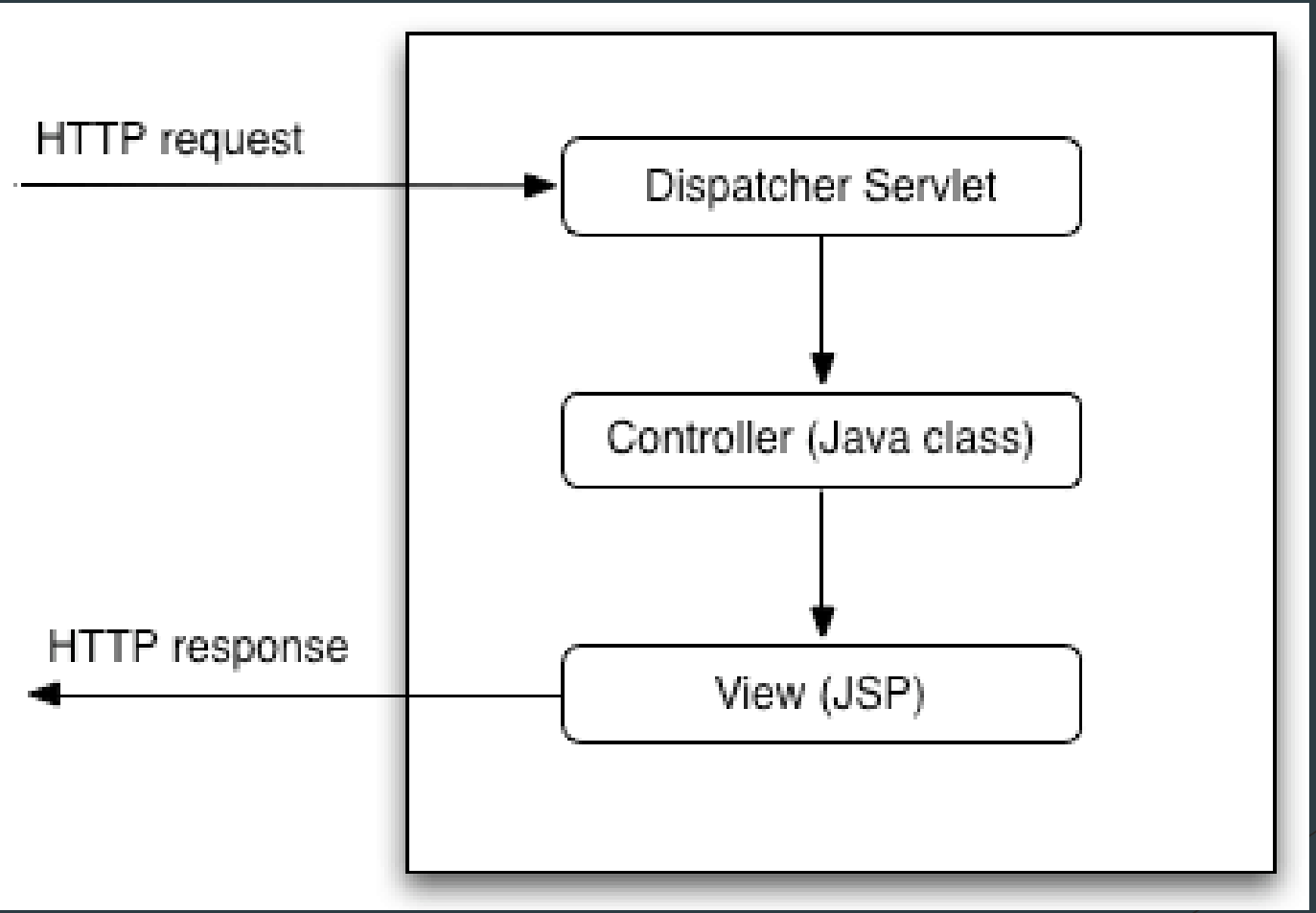
- **JSON** or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange. Conventions used by JSON are known to programmers, which include C, C++, Java, Python, Perl, etc.

- **Characteristics of JSON:**
  - JSON is easy to read and write.
  - It is a lightweight text-based interchange format.
  - JSON is language independent.
  - The filename extension is **.json**.


Left Square Bracket defines the beginning of a JSON text
Colon depicts assignment of a value to a name
"empid" is the name (column). 1 is the value (for this row)
[
{
"empid":1,
"lastname":"Davis",
"firstname":"Sara",
"title":"CEO",
"titleofcourtesy":"Ms.",
"birthdate":"1968-12-08",
"hiredate":"2013-05-01",
"address":"7890 - 20th Ave. E., Apt. 2A",
"city":"Seattle",
"region":"WA",
"postalcode":"10003",
"country":"USA",
"phone":"(206) 555-0101"
},
Left and Right Curly Brackets enclose a JSON Object
Comma separates this first object from the next JSON object

Dispatcher Servlet

- In Spring MVC all incoming requests go through a single servlet is called **Dispatcher Servlet**
- A single servlet receives all the request and transfers them to all other components of the application.



Jackson

- Jackson is a simple java based library to serialize java objects to JSON and vice versa.

Spring MVC and REST Annotations

- **@RequestMapping:** It is used to map the **web requests**. It has many optional elements like **consumes, header, method, name, params, path, produces**, and **value**. We use it with the class as well as the method.
- **@GetMapping:** It maps the **HTTP GET** requests on the specific handler method. It is used to create a web service endpoint that **fetches** It is used instead of using: **@RequestMapping(method = RequestMethod.GET)**
- **@PostMapping:** It maps the **HTTP POST** requests on the specific handler method. It is used to create a web service endpoint that **creates** It is used instead of using: **@RequestMapping(method = RequestMethod.POST)**
- **@PutMapping:** It maps the **HTTP PUT** requests on the specific handler method. It is used to create a web service endpoint that **creates** or **updates** It is used instead of using: **@RequestMapping(method = RequestMethod.PUT)**
- **@DeleteMapping:** It maps the **HTTP DELETE** requests on the specific handler method. It is used to create a web service endpoint that **deletes** a resource. It is used instead of using: **@RequestMapping(method = RequestMethod.DELETE)**

- **@RestController:** It can be considered as a combination of **@Controller** and **@ResponseBody** annotations. The @RestController annotation is itself annotated with the @ResponseBody annotation. It eliminates the need for annotating each method with @ResponseBody.
- **@RequestBody:** It is used to **bind** HTTP request with an object in a method parameter. Internally it uses **HTTP MessageConverters** to convert the body of the request. When we annotate a method parameter with **@RequestBody,** the Spring framework binds the incoming HTTP request body to that parameter.
- **@ResponseBody:** It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.
- **@RequestHeader:** It is used to get the details about the HTTP request headers. We use this annotation as a **method parameter**. The optional elements of the annotation are **name, required, value, defaultValue.** For each detail in the header, we should specify separate annotations. We can use it multiple time in a method
- **@RequestAttribute:** It binds a method parameter to request attribute. It provides convenient access to the request attributes from a controller method. With the help of @RequestAttribute annotation, we can access objects that are populated on the server-side.