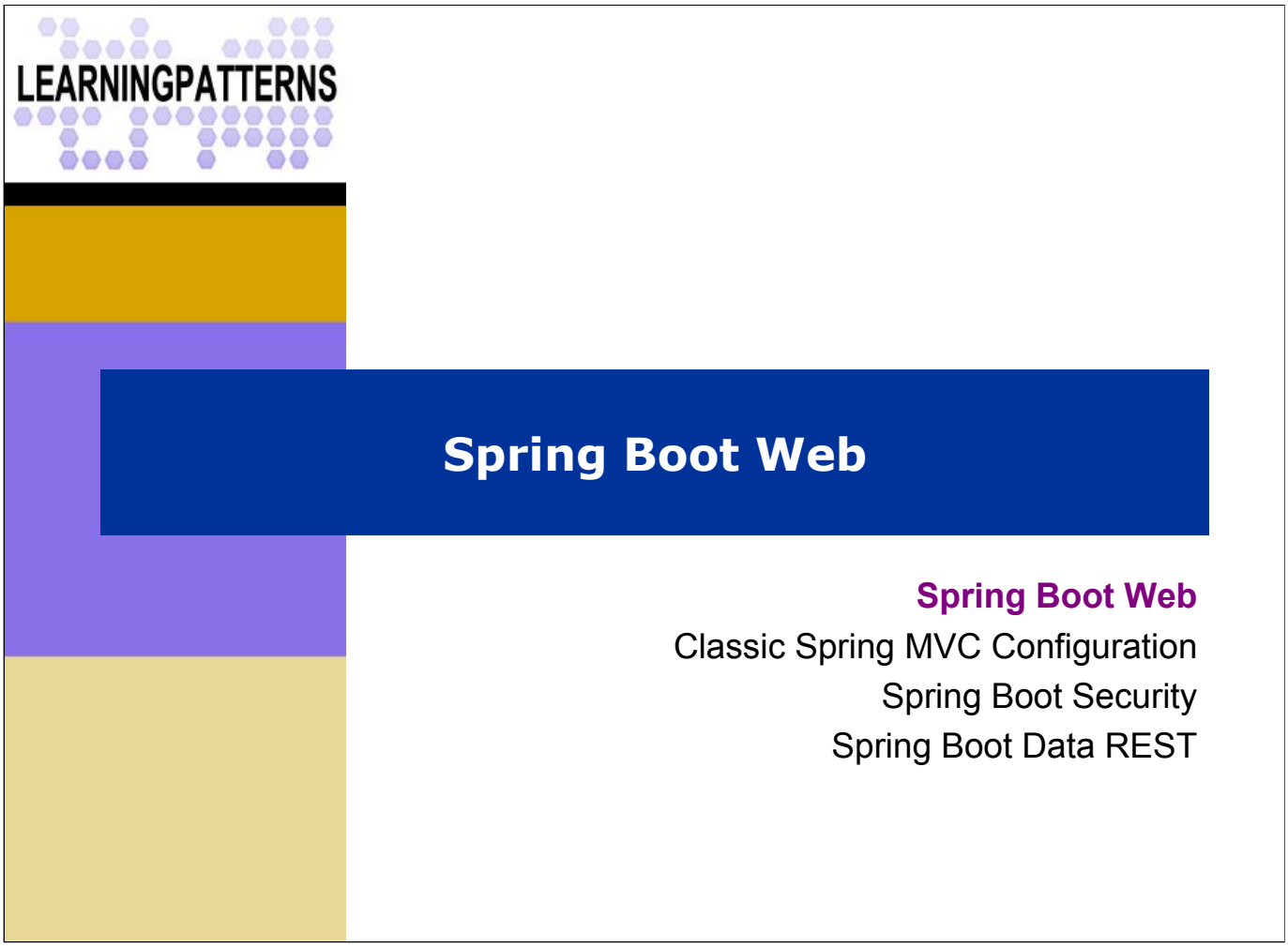


Notes:

Overview

- We've seen **`spring-boot-starter-web`**
  - And many of its capabilities and auto-configurations
- Here, we'll look at some more capabilities
  - Using **other servers** such as Jetty or Undertow
  - Other server configuration
  - Configuring deployment to **external Web server**

Notes:

Embedded Server Config Properties

- Many Boot properties to configure the **embedded** server
  - **`server.port/address`**: Server HTTP port / network address
  - **`server.servlet.context-path`**: Application context root
  - **`spring.mvc.servlet.path`**: Dispatcher servlet mapping (/)
  - **`server.tomcat.*`**: Many properties for Tomcat config
    - And similarly for **`.jetty.*`**, **`.netty.*`**, and **`.undertow.*`**
  - **`server.error.path`**: Error controller path (Default /error)
  - **Session settings**: Many settings
    - e.g. **`server.servlet.session.timeout`**: session timeout
- Lots of configuration available
  - Appendix A in the reference has a complete list of properties
  - Server properties are in the .A.11 section, and Spring Web properties are in the .A.9 section

Notes:

- For `server.port`, you can also do the following.
  - Use **`server.port=-1`** to switch off the HTTP endpoints, but still create a WebApplicationContext. This is sometimes useful for testing.
  - Use **`server.port=0`** to scan for a free port.

Other Web Support

- Spring Boot has a number of other web-related starters, including:
  - **`spring-boot-starter-jetty`**: Imports Jetty (Tomcat alternative)
  - **`spring-boot-starter-undertow`**: Imports Undertow (Tomcat alternative)
    - We've seen how to configure this
  - **`spring-boot-starter-websocket`**: WebSocket support
  - **`spring-boot-starter-hateoas`**: HATEOAS-based RESTful services

Notes:

- `HATEOAS`: Hypermedia as the Engine of Application State
  - Includes hypermedia links as part of a REST response.

Example: Using Undertow Embedded Server

- Just **add** the server dependency (a Boot starter)
  - And **exclude** the Tomcat dependency that's automatically pulled in by `spring-boot-starter-web`
  - We illustrate below
  - Boot will now use Undertow for the embedded server

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-undertow</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <exclusions>
    <exclusion>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-tomcat</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```


Notes:

Deploying to External Server

- Common to deploy apps to an external server
- Requirements for external server deploy include:
  - **Configure the Web app** for the servlet container
    - Done via a Servlet 3+ Java initializer (which is fairly standard for Spring Web apps)
    - Uses Spring's `WebApplicationInitializer` and variants of it
  - **Prevent deployment** of embedded server jars
    - Done in the POM
  - **Set up packaging** as a WAR
    - Done in the POM
  - We'll do this in the lab

Notes:

- When you use `spring-boot-starter-web` and you have `@RestController`, Spring Boot automatically registers the controller for you.
  - As well as the Spring MVC dispatcher servlet.
  - We'll show the non-Boot equivalent of this in the lab.

External Server - Web App Config

- The application class below now serves as a Web app initializer
  - By extending **`SpringBootServletInitializer`**
    - Package `org.springframework.boot.web.servlet.support`
  - The `configure()` callback is called as part of the Web app lifecycle (1)
    - **`SpringApplicationBuilder.sources`** adds config classes
    - `main()` not used at all with external server (2)
  - That's it - Spring Boot still configures everything else
    - Like the dispatcher servlet for Spring MVC/REST

```
@SpringBootApplication
public class HelloWebWorld extends SpringBootServletInitializer {

  @Override
  protected SpringApplicationBuilder configure(
                    SpringApplicationBuilder application) {
    return application.sources(HelloWebWorld.class);
  }

  public static void main(String[] args) {
    SpringApplication.run(HelloWebWorld.class, args);
  }
}
```


Notes:

(1) Note that the `configure()` method is a callback to give you a chance to configure the Spring container.

- It is called as part of the lifecycle of the Web app.
- The javadoc for this method states "*Configure the application. Normally all you would need to do is to add sources (e.g. config classes) because other settings have sensible defaults. You might choose (for instance) to add default command line arguments, or set an active Spring profile.*"

(2z) Using the above application class, you can both:

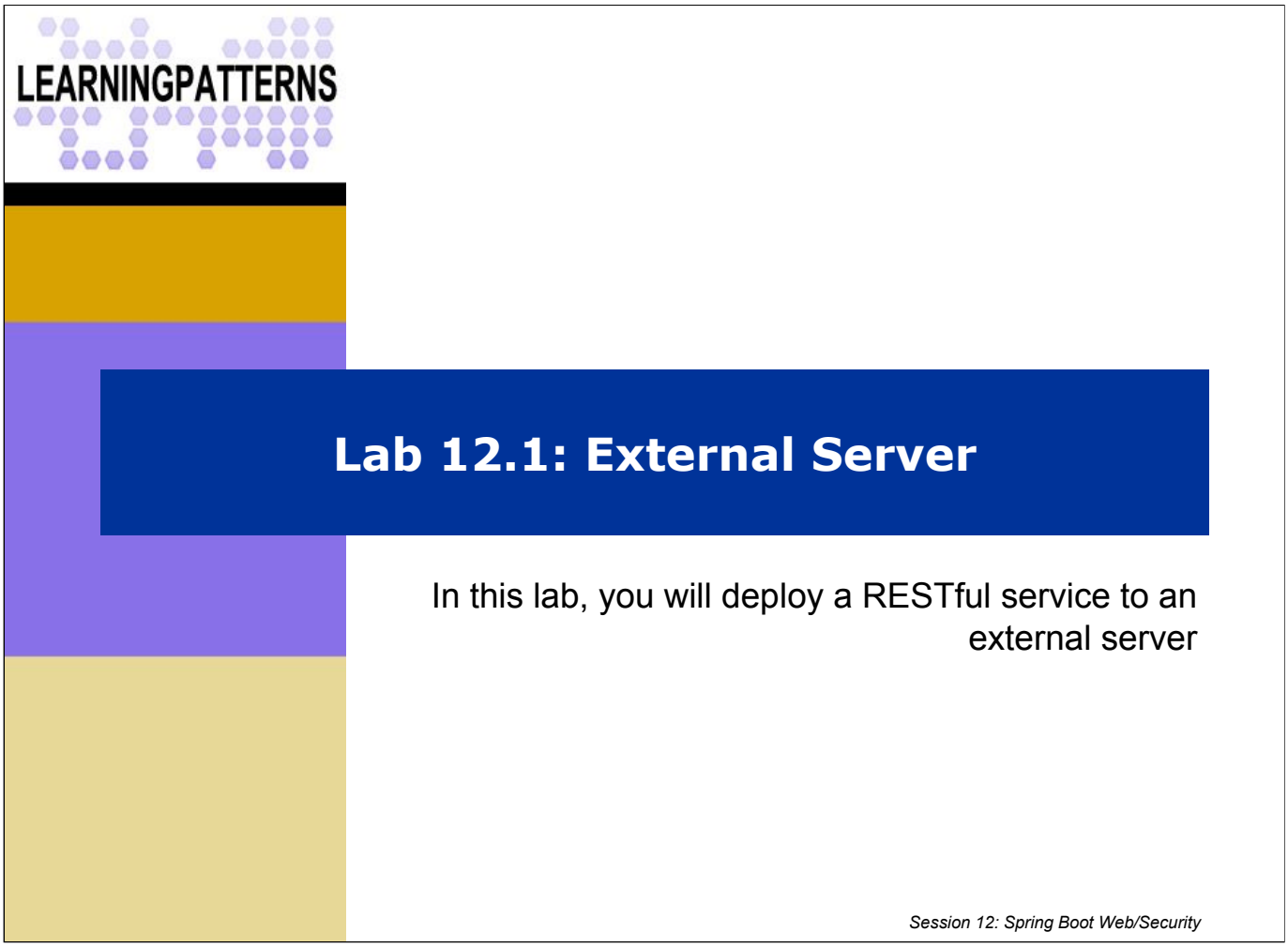
- Run it with the embedded server, by running `main()` as a standard Java app.
- Deploy it to an external server which will use the added web app initializer code.
- This is not required, but it is convenient if wanting to use both environments. If you're only using one environment (e.g. deploy to external server) then just leave out the other (e.g. leave out `main()`).

External Server Deployment - POM Details

- Must add two POM entries, as shown below
  - **<packaging>** configures **WAR** creation for the package
  - **<dependency>** specifies a scope of **provided** for the server
    - Tomcat in this case - so its jars are not included when a WAR is created
  - That's it - package it and you get a WAR without server jars

```
<project …>


  <packaging>war</packaging>

   <dependencies>

     <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-tomcat</artifactId>
       <scope>provided</scope>
     </dependency>

  <dependencies>
</project>
```


Notes:



Notes: