A Partnership for Progress

Maven
Build Tool

Introduction to Maven

- Maven is a Java-based tool. The Apache group developed Maven to support developing and building multiple projects together, for publishing and deploying them, and finally, for generating the reports.
- **Maven aims to describe two important things:**
  1. How a software is built.
  2. The dependencies (jar files), plug-ins, and profiles that the project is associated in a standalone or a distributed environment.
- A more formal definition of Apache Maven is "*a project management tool, which encompasses a project object model, a set of standards, a project lifecycle, a dependency management system, and the logic for executing plugin goals at defined phases in a lifecycle*." When you use Maven, you describe your project using a well-defined project object model. Maven can then apply cross-cutting logic from a set of shared (or custom) plugins.

Advantages of Using Maven

- Dependencies.
- Convention over configuration.
- Multiple/repeated builds.
- Automation.
- Plugin management.
- Testing.
- Development process.
- Build status.
- Consistent setups.
- Standard and uniformed directory structure.

Core Concepts of Maven

- ❑ Project Object Model (POM) Files.
- ❑ Dependencies and repositories.
- ❑ Cycles, phases, and goals.
- ❑ Profiles.
- ❑ Plugins.

Installation of Maven

- ❑ Verify that your system has JDK installed. If it does not, install Java.
- ❑ Check to confirm that the Java Environment Variable is set. If is not, set the Java environment variable using this link: Install java and setting environment variable).
- ❑ Download Maven, Click here → (Link)
- ❑ Unpack your Maven zip at any place in your system (folder).
- ❑ Add Maven to the Environment variable (system variable environment and PATH environment variable).
  - ➢ We add both **M2_HOME** and **MAVEN_HOME** variables to the Windows environment.
- ❑ Open cmd and run **mvn -v** command.

LAB - Installation of Maven

Click here for [LAB - Install Maven](#) on Windows.

If you have technical questions while performing the practice assignment, ask your instructors for assistance.

Project Object Model

The easiest way to describe a Project Object Model (POM) in Maven is that POM is a core element. Maven projects consist of one configurable file called *pom.xml*. The pom.xml will always be located in the root directory of any maven project. This file represents the very basic and fundamental unit in Maven.

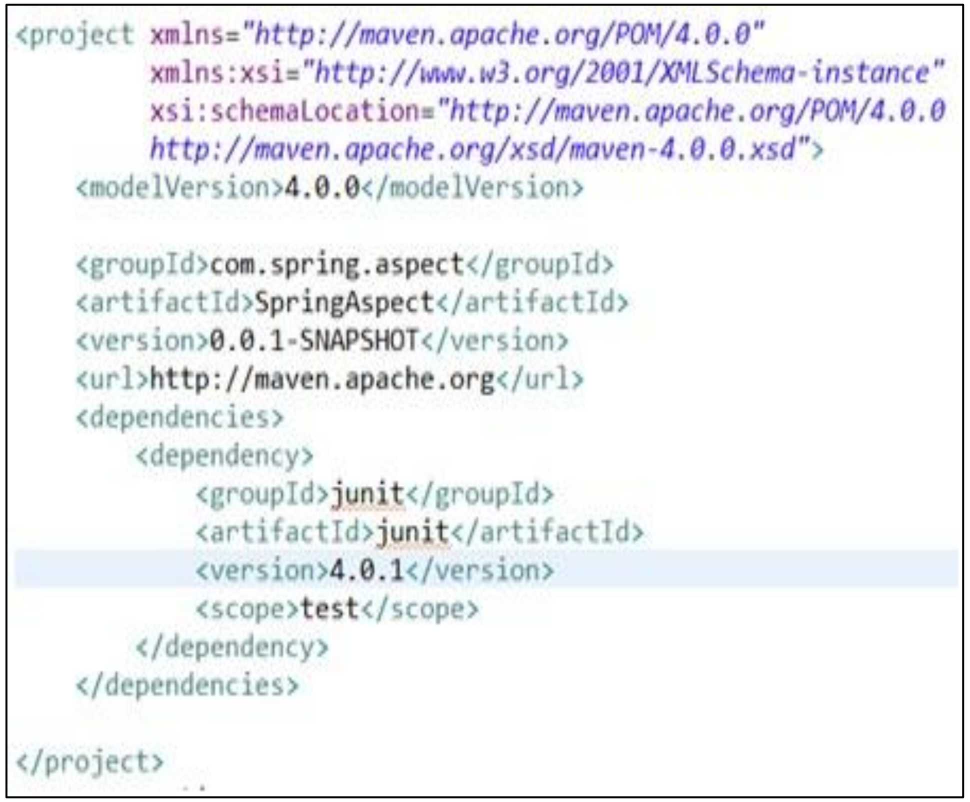
The **pom.xml** contains the information related to the project, which is built or to be built in. It contains all the necessary information about the configuration details, and dependencies and plugins included in the project. In simple words, it contains the details of the build lifecycle of a project.

Maven scans through the entries pom.xml file. This will enable the Maven to read all the configurations made, build profiles defined, repositories configured, and all other important details, and then execute the tasks accordingly.

Configurations in the pom.xml file:

- ❑ Dependencies used in the projects (Jar files).
- ❑ Plugins used (report plugin).
- ❑ Project version.
- ❑ Developers involved in the project.
- ❑ Mailing list.
- ❑ Reporting.
- ❑ Build profiles.

Sample Pom.xml File



```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.spring.aspect</groupId>
    <artifactId>SpringAspect</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <url>http://maven.apache.org</url>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.0.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

</project>
```

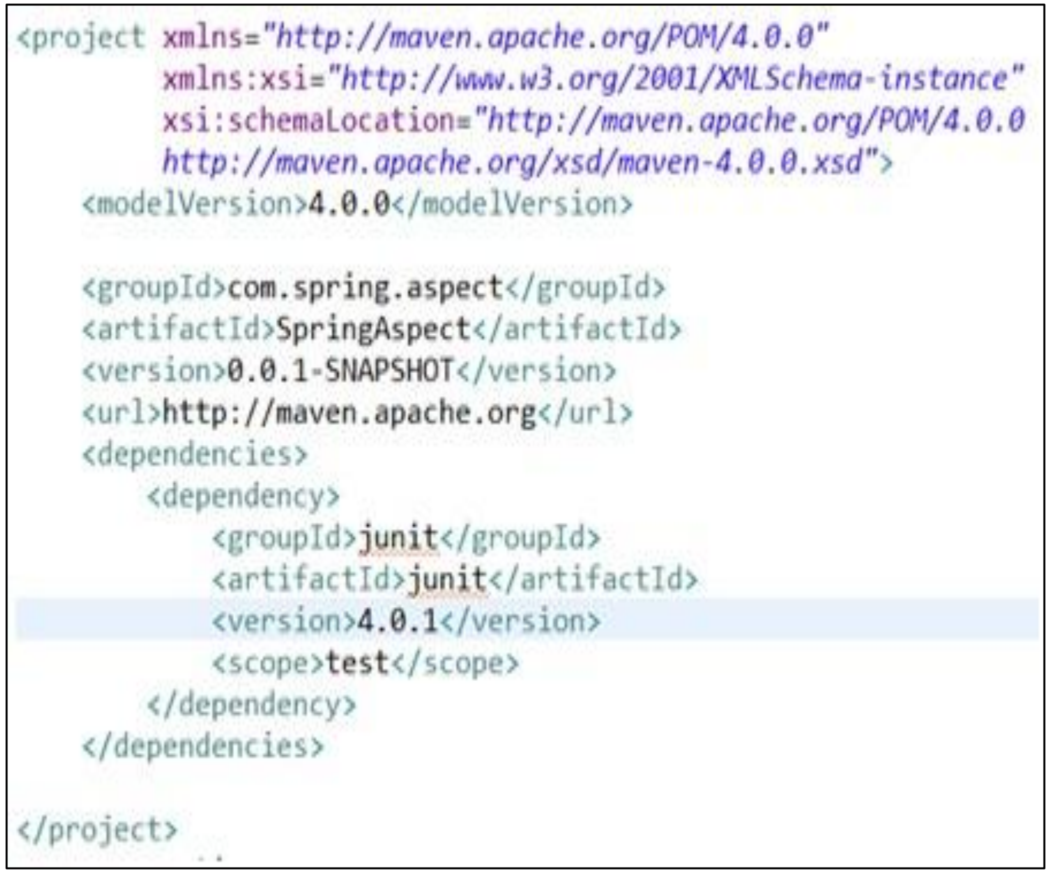
- **project** - the root element of the pom.xml file.
- **modelVersion** - the version of the POM model you are using.
- **groupId** - an id, uniquely identifying your project across all projects
- **artifactId** - defines the name of any project. The jar without version. If you created it, you can choose whatever name you want with lowercase letters and no strange symbols.
- **version** - used to derive the version of any project in order to classify the versions, and when the major changes/implementations are carried during the development phase of a project.

Whenever it comes for executing a task for a project, Maven scans through the entries made in the pom.xml file, enabling Maven to read all of the configurations made, build profiles defined, repositories configured, and all other important details, and then execute the tasks accordingly.

Developers should ensure to define the list of elements which are known as *Maven coordinates* before defining a pom.xml file.

The first three of these (groupId:artifactId:version) combine to form the unique identifier, and are the mechanism by which you specify which versions of external libraries (e.g. JARs) your project will use.

Sample Pom.xml File (continued)



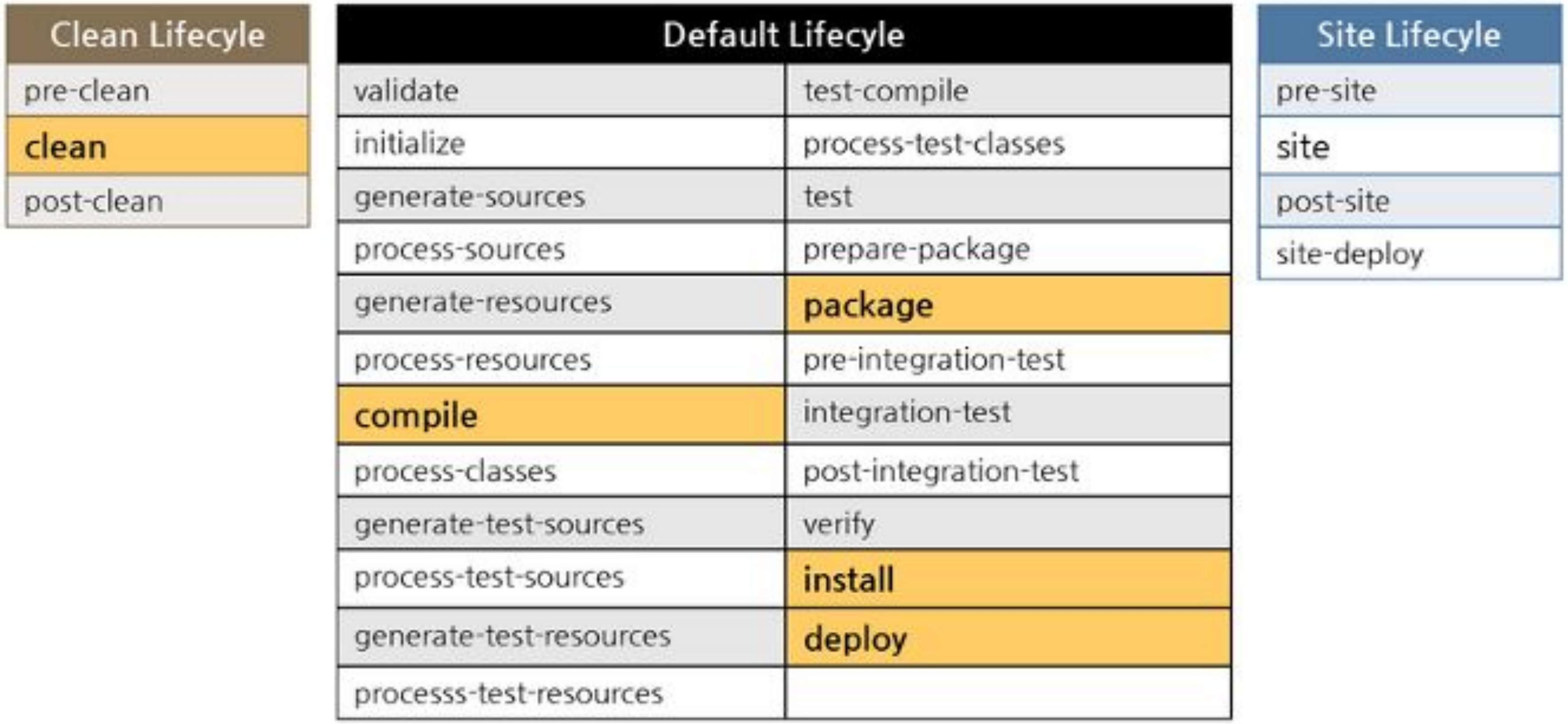
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.spring.aspect</groupId>
    <artifactId>SpringAspect</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <url>http://maven.apache.org</url>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.0.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

</project>
```

- **Dependencies** - elements used to define a list of dependencies of a project.
- **Dependency** - defines a dependency, and is used inside a dependencies tag; described by its groupId, artifactId, and version.
- **Name** - used to give a name to our Maven project.
- **Scope** - scope attribute is used to specify the visibility of a dependency, relative to the different lifecycle phases (build, test, runtime etc). Maven provides six scopes: compile, provide, runtime, test, system, and import.
- **Packaging** - used to package our project to output types such as JAR, WAR, etc.

Maven Build Lifecycle

- The sequence of steps defined in order to execute the tasks and goals of any Maven project is known as *build lifecycle* in Maven.
- Developers only need to learn a small set of commands to build any Maven project, and the POM will ensure that they get the results they desired.
- Maven comes with three built-in build lifecycles:
  1. **Clean** - this phase involves cleaning of the project (for a fresh build and deployment).
  2. **Default** - this phase handles the complete deployment of the project.
  3. **Site** - this phase handles generating the Java documentation of the project.

Maven Build Lifecycle (continued)



Clean Lifecycle
pre-clean
clean
post-clean

Default Lifecycle	
validate	test-compile
initialize	process-test-classes
generate-sources	test
process-sources	prepare-package
generate-resources	**package**
process-resources	pre-integration-test
compile	integration-test
process-classes	post-integration-test
generate-test-sources	verify
process-test-sources	**install**
generate-test-resources	**deploy**
processs-test-resources	

Site Lifecycle
pre-site
site
post-site
site-deploy

Dependencies and Repositories

- Dependencies are external Java libraries required for the project, and repositories are directories of packaged JAR files. The local repository is just a directory on your machine hard drive. If the dependencies are not found in the local Maven repository, Maven downloads them from a central Maven repository and puts them in your local repository.
- To download the required artifacts of the build and dependencies (JAR files) and other plugins, which are configured as part of any project, there should be a common place where all such artifacts are placed. This commonly shared area is called a **Repository**.
- In Maven, repositories are classified into three main categories:

  1. Local Repository.
  2. Remote Repository.
  3. Central Repository.

Local Repository

The repository resides in our local machine, which is cached from the remote/central repository downloads and is ready for the usage. The folder to hold/place all the dependencies in the local repository can be configured in the settings.xml file of the maven folder under the tag **<localRepository>.**

The default location of the local repository is **$HOME/.m2/repository.**

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">
    <localRepository>D:/m2repo</localRepository>
</settings>
```

Remote Repository

The remote repository, as the name suggests, resides in the remote server and can be accessed by using different file transfer protocols like **file:// or http://.** Remote repositories will be used for both downloading and uploading the dependencies and artifacts.

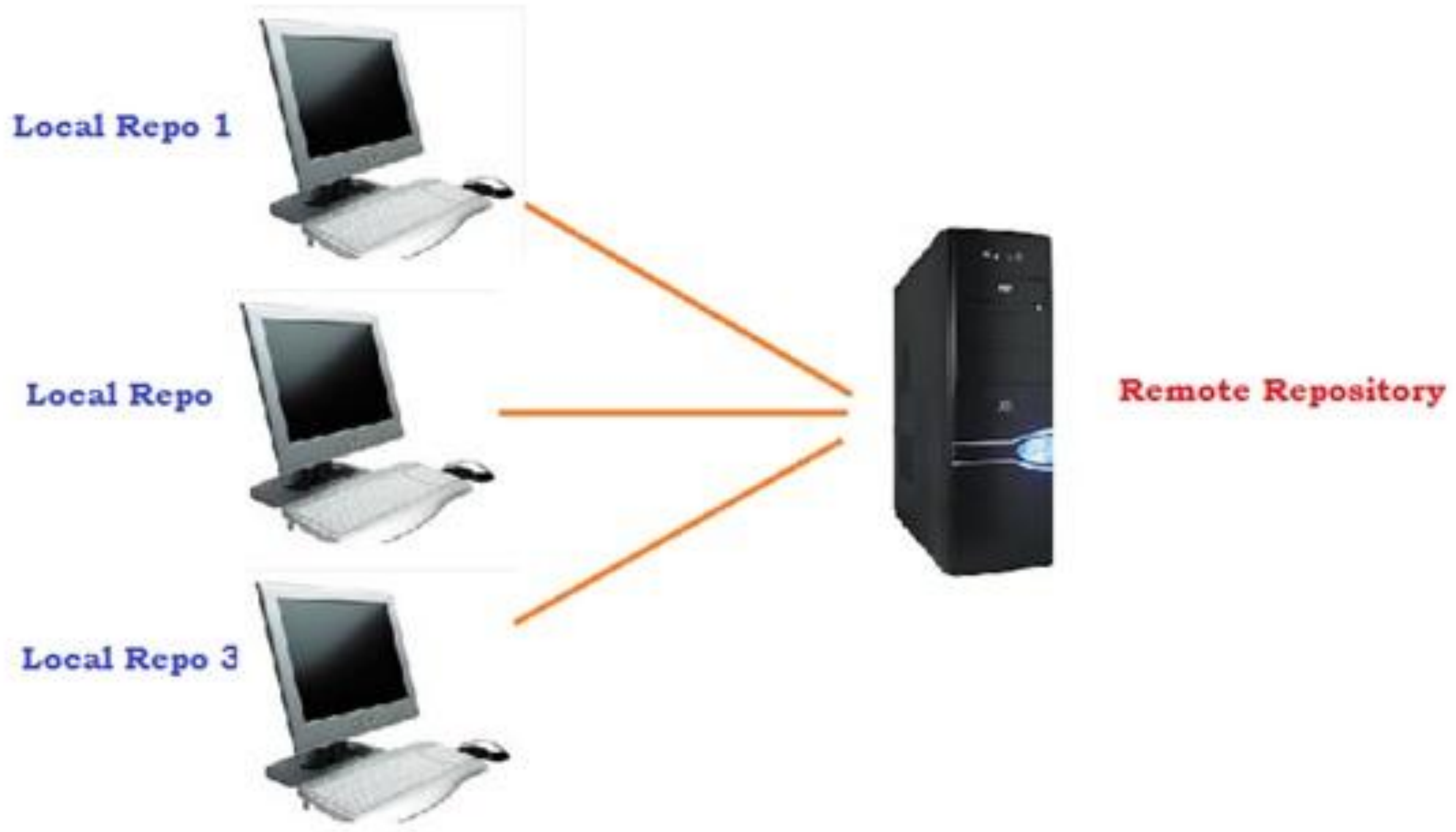
```
<repositories>
    <repository>
        <id>remote.repository</id> <url>http://download.ogrname.com/maven2/</url>
    </repository>
</repositories>
```

Central Repository

This is the repository provided by the Maven community. This repository contains large sets of commonly used/required libraries for any Java project. Basically, internet connection is required if developers want to make use of this central repository; but no configuration is required for accessing this central repository.

```
<repositories>
    <repository>
        <id>central.repository</id> <url>http://repo1.maven.org/maven2/</url>
    </repository>
</repositories>
```

Local and Remote/Central Repositories



Search for Dependencies

When Maven starts executing the building commands, it starts searching the dependencies as explained below:

- It scans through the local repositories for all of the configured dependencies. If found, it continues with further execution. If the configured dependencies are not found in the local repository, then it scans through the central repository.
- If the specified dependencies are found in the central repository, those dependencies are downloaded to the local repository for future reference and usage. If not found, Maven starts scanning into the remote repositories.
- If no remote repository has been configured, Maven will throw an exception saying, "not able to find the dependencies," and stops processing. If found, those dependencies are downloaded to the local repository for future reference and usage.

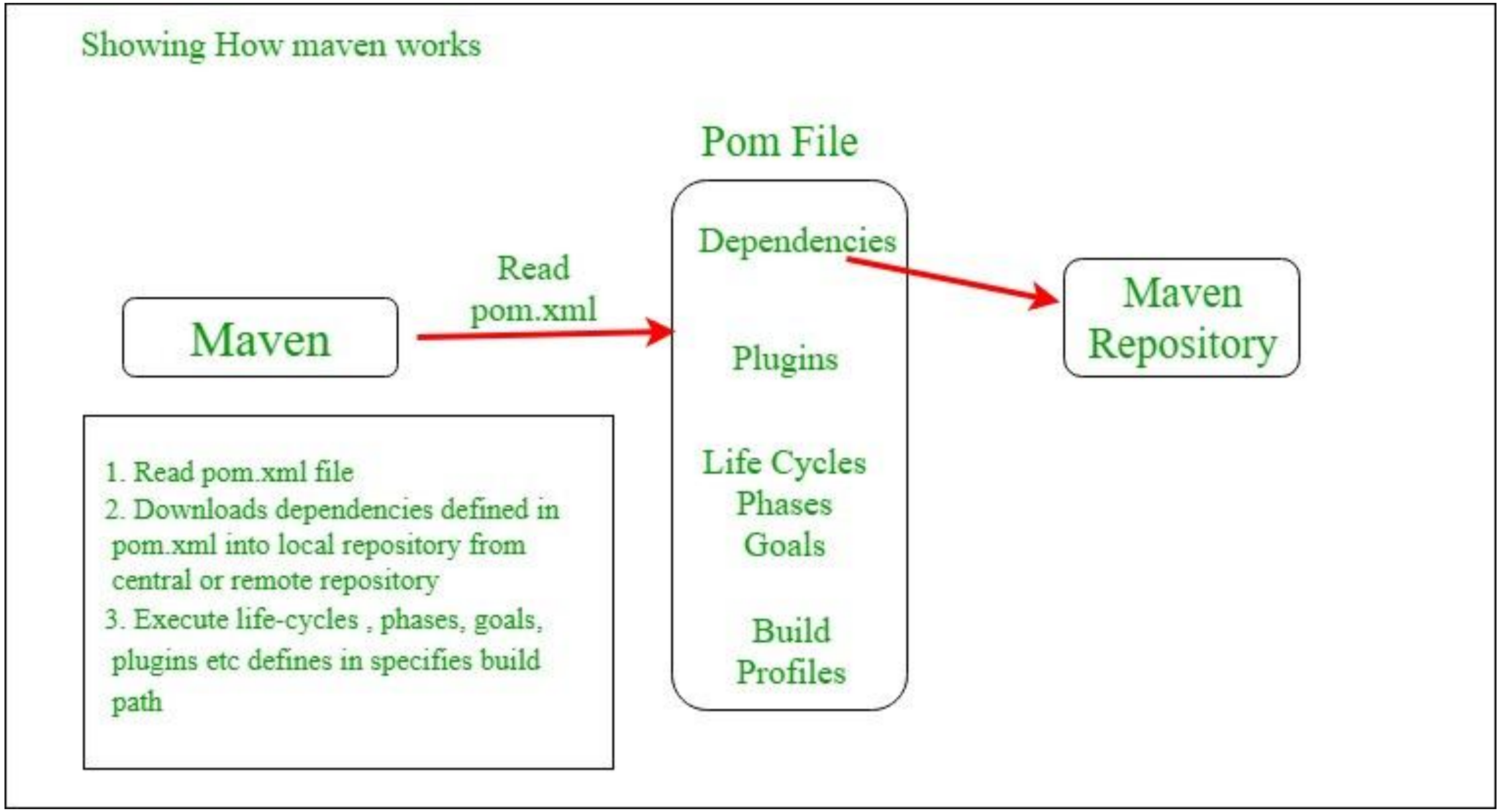
Maven Plugins

- Maven is actually a *plugin execution* framework, where every task is performed by plugins used to:
  - Create jar files.
  - Create war files.
  - Compile code files.
  - Test units of code.
  - Create project documentation.
  - Create project reports.
- A plugin generally provides a set of goals, which can be executed using the following syntax:

  mvn [plugin-name]:[goal-name]
- Example: A Java project can be compiled with the maven-compiler-plugin's compile-goal by running the following command:

  mvn compiler:compile

How Maven Works



1. Read pom.xml file
2. Downloads dependencies defined in pom.xml into local repository from central or remote repository
3. Execute life-cycles , phases, goals, plugins etc defines in specifies build path

Questions?