Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Maven Overview

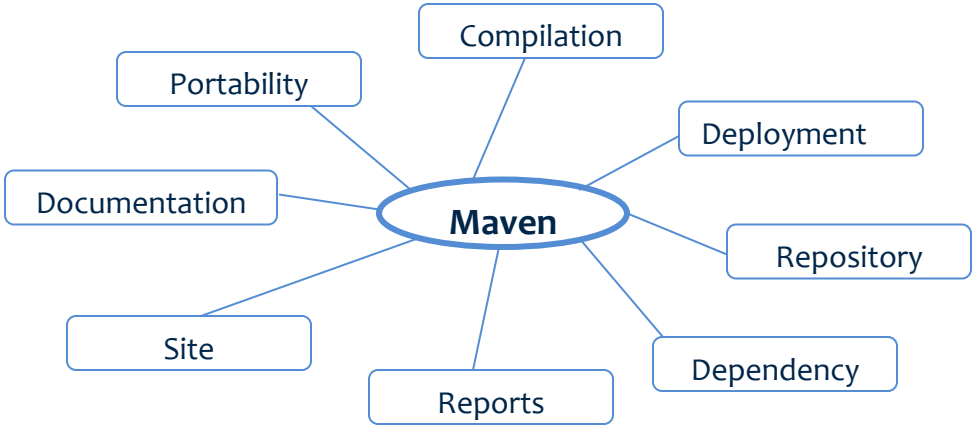
Lesson 01 Introduction

Lesson Objectives

- Maven Overview
  - What is Maven?
  - Comparison of Maven with Ant
  - Maven's Objective
- Maven's Principles
- Benefits of Maven
- Elements of Maven
- POM, Standard Directory Structure, Build Life Cycle,
- Plug-in, Dependency Management
- Resolving Dependency Conflicts, Repositories

1.1 Maven Overview

- Project Management and build tool hosted by Apache software foundation.
- Maven provides features such as:
  - Compilation
  - Deployment
  - Repository
  - Dependency
  - Reports
  - Site
  - Documentation
  - Portability



1.1 Maven Overview

Maven	Ant
Standard Naming Conventions	No Formal Conventions
Declarative	Procedural
Build Lifecycle	Own Lifecycle
Reusable plugins, repositories	Scripts are not reusable
Standard Directory Layout	No Standard Layouts followed
Reuse of Project Object Model file is easy	Reuse of build file is hard
Generates a website of project	Not possible to generate website
Dependencies downloaded automatically	Manually goal dependencies need to be set

1.1 Maven Overview

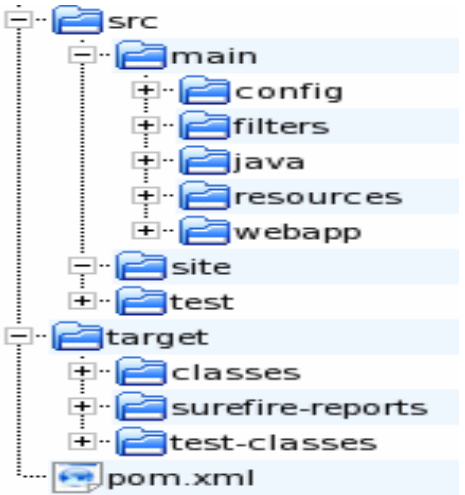
- Maven allows to comprehend the complete state of a development effort in the shortest period of time.
- To attain this goal, Maven deals with:
  - Making the build process easy
  - Providing a uniform build system
    - Familiarize with automatic project build makes to navigate through multiple projects easily.
  - Providing quality project information
    - Project information provided in POM file improves the reusability of resources.
  - Providing guidelines for best practices development
  - Allowing transparent migration to new features
    - Easy way of installing new or updated plugins

1.2 Maven Principles

- Maven provides following principles for creating a shared language:
  - Convention over Configuration
  - Declarative Execution
  - Reuse of build logic
  - Coherent Organization of dependencies
- These principles allows developers to communicate more effectively at higher level of abstraction.
- It also allows team members to get on with the important work of creating value at the application level.
- It improves the software development process.

1.2 Maven Principles

- **Facilitate a uniform build system that speeds up the development cycle.**
- **There are three primary conventions that Maven employs to promote a familiar development environment:**
  - Standard Project Layout
  - Standard Naming conventions
  - One Primary Output per Project
- **Maven will require almost zero effort, if convention is followed.**



1.2 Maven Principles

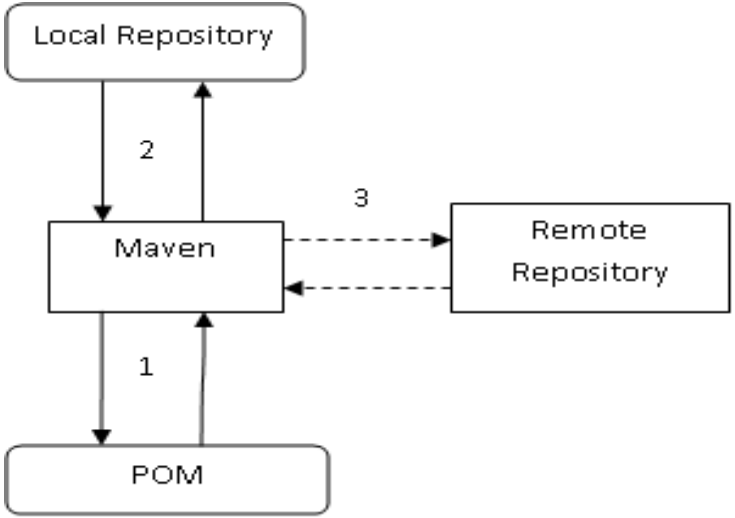
- Maven is driven in a declarative fashion using Project Object Model (POM) and specifically the plugin configurations contained in the POM.
- POM
  - POM consists of entire Project information in an xml document(pom.xml)
  - POM plays a vital role that drives maven execution as Model driven execution.
  - POM file can be inherited to reuse the project resources in another project.
- Build Life cycle
  - In maven, Build life cycle consists of series of phases where each phase can perform one or more actions.
  - Facilitates Automatic Build Process.

1.2 Maven Principles

- Plugins are the central feature of Maven that allow for the reuse of common build logic across multiple projects.
- Encapsulates build logic into coherent modules called plugins
- Project build can be customized by using existing or custom plugin.
- Everything is a plug-in
  - Compilation
  - Unit testing
  - Deployment
  - Documentation
- The execution of Maven's plugins is coordinated by Maven's build life cycle with instructions from Maven's POM

1.2 Maven Principles

- A dependency is a reference to an specific artifact that resides in a repository.
- Dependencies are requested in a declarative fashion with dependency's coordinates by looking up in repositories.
- There are two repositories available:
  - Local Repository
    - By default, Maven creates your local repository in ~/.m2/repository
  - Remote Repository
    - Default central Maven repository:
    - http://repo1.maven.org/maven2/



1.3 Benefits of Maven

- Encourages best practices
- Provides a uniform build system and consistent usage across all projects
- Provides dependency management including automatic updating, dependency closures (also known as transitive dependencies)
- Provides reuse of build logic
- Defines project standard directory layout
- Helps to reduce the duplication of dependent software libraries (jars) required to build an application
- Stores all the software libraries or artifacts in a remote stores called a Maven repositories

2.1 Project Object Model(POM)

- POM = Project Object Model = pom.xml
- Contains metadata about the Project
  - Location of directories, Developers/Contributors, Issue tracking system, Dependencies, Repositories to use, etc
- Example:

```
<project>
  <modelVersion>1.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>my-app</artifactId>
  <name>my maven app</name>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <dependencies/>
  <build/>
[…]
```

Minimal POM

2.1 Project Object Model(POM)

Declarative execution- POM model.
pom.xml

project-
Top level element
Of pom.xml

version-
Version of the generated
Artifact.
SNAPSHOT- project is
In a state of development

name-
Display name used for the
Project(Full name). Often used in
Generated Documentation, during
build
Process Of the project, or projects
that
use it As a dependency

```
<project>
    <modelVersion>1.0.0</modelVersion>
     <groupId>com.example</groupId>
     <artifactId>jmy-app</artifactId>
     <packaging>jar</packaging>
     <version>1.0-SNAPSHOT</version>
     <name>my maven app</name>
     <url>http://maven.apache.org</url>
     <description></description>
     <dependencies>
      <dependency>
       <groupId>junit</groupId>
       <artifactId>junit</artifactId>
       <version>3.8.1</version>
       <scope>test</scope>
     </dependency>
```

modelVersion-
Version of the object
Model this POM is using

groupId-
Unique identifier for
the project.

artifactId-
Unique base name of the
Primary artifact being
Generated by the object.
Ex. My-app-1.0.jar

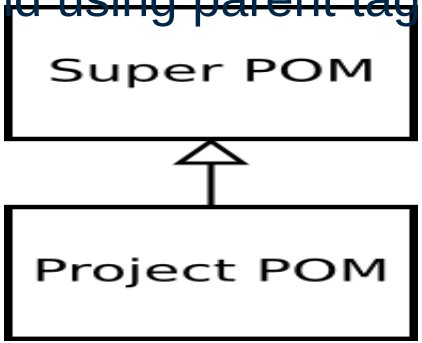
packaging-
Type of packaging to be
Used by this artifact.
Ex. JAR, WAR, or EAR

url-
This element indicates
Where the project's site
Can be found.

description-
Basic description of the
Project.

2.1 POM

- Share settings between projects/Modules using POM inheritance.
- Packaging of pom in parent and using parent tag in a child project.



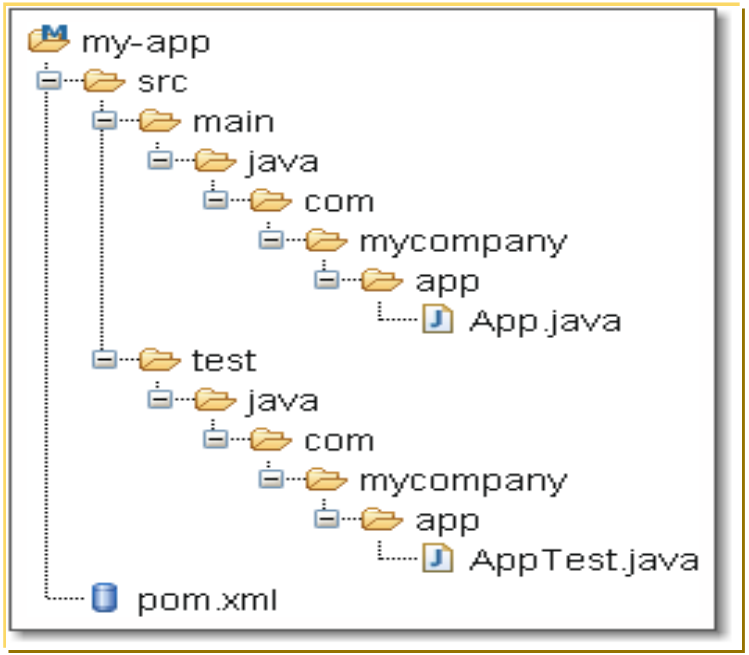
pom.xml for module1

```
<parent>
  <groupId>example</groupId>
  <artifactId>jars</artifactId>
  <version>1.0SNAPSHOT<version>
</parent>
```

- Reduces the amount of configuration done in "child" projects
- Helps to simplify the creation of multi module project.

2.2 Standard Directory Layout

- Having a common directory layout make the project easier to understand by other developers.
- It makes it easier to integrate plugins.
- Project home directory consists of POM(pom.xml) and two subdirectories initially:
  - src : contains all source code and
  - test: contains test source codes
- Target directory generated after the compilation of application sources.



Directory structure before project execution

2.2 Standard Directory Layout

Listing out few subdirectories in src directory

Directory name	Purpose
src/main/java	Contains the deliverable Java source code for the project.
src/main/resources	Contains the deliverable resources for the project, such as property files.
src/test/java	Contains the testing classes (JUnit or TestNG test cases, for example) for the project.
src/test/resources	Contains resources necessary for testing.
src/site	Contains files used to generate the Maven project website.

2.3 Plug-in

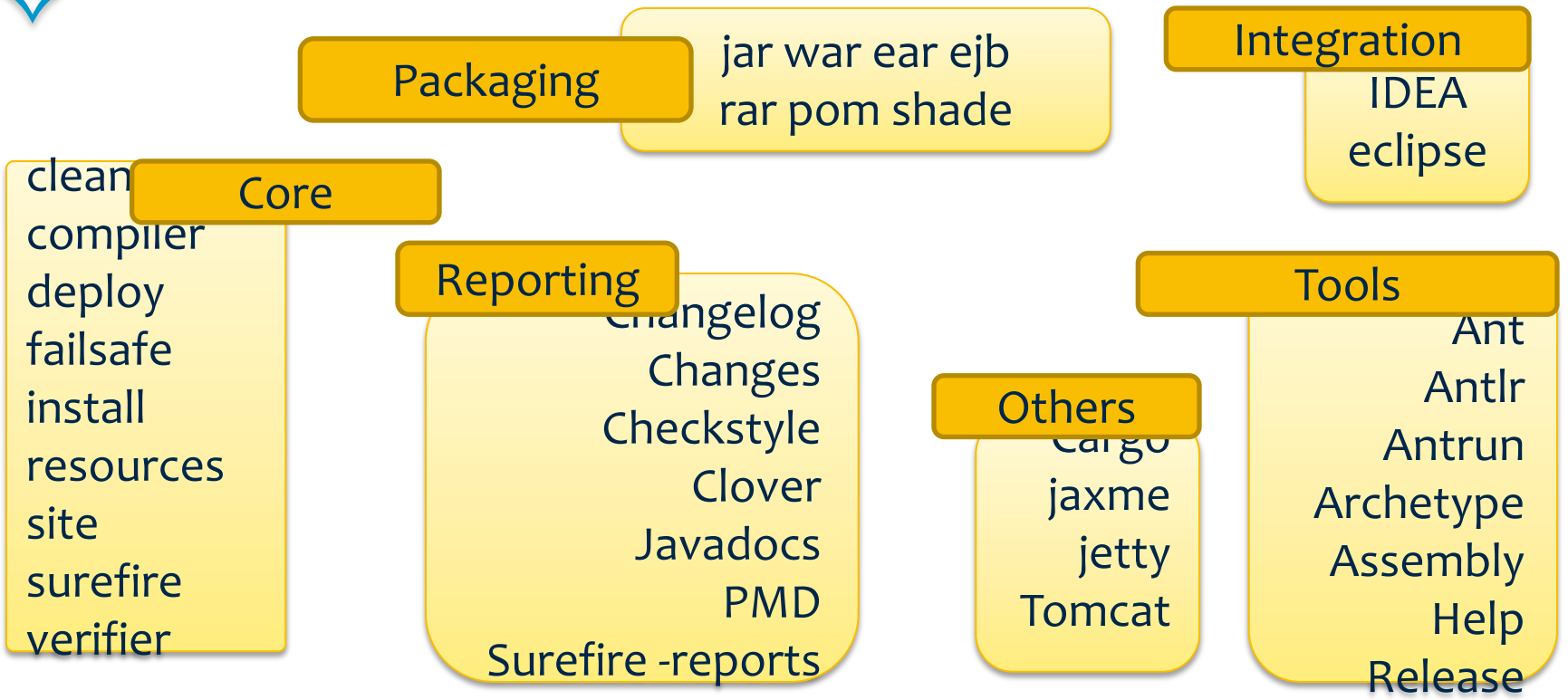
- Maven is built using a plugin-based architecture
- Each step in a lifecycle flow is called a phase. Zero or more plugin goals are bound to a phase.
- A plugin is a logical grouping and distribution (often a single JAR) of related goals, such as JARing.
- A goal, the most granular step in Maven, is a single executable task within a plugin.
- For example, discrete goals in the jar plugin include packaging the jar (jar:jar), signing the jar (jar:sign), and verifying the signature (jar:sign-

2.3 Plug-in

- A plugin provides a set of goals that can be executed using the following syntax:
  - mvn [plugin-name]:[goal-name]
- Plugins reduces the repetitive tasks involved in the programming.
- Plugins are configured in a <plugins>-section of a pom.xml file as shown below

```
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>2.0</version>
<configuration>
<source>1.5</source>
<target>1.5</target>
</configuration>
</plugin>
</plugins>
```

2.3 Plug-in



2.3 Plug-in

- Standard Plugin Configuration:
  - Build plugins will be executed during the build and they should be configured in the <build/> element from the POM.
  - Reporting plugins will be executed during the site generation and they should be configured in the <reporting/> element from the POM.
  - All plugins should have minimal required informations: groupId, artifactId and version
- Custom plugin can also be created and configured in pom.xml based on plugin coordinates.
- A mojo (build task) within a plug-in is executed when the Maven engine executes the corresponding phase on the build life cycle.
- Plug-in developers can flexibly associate one or more life-cycle phases with a plug-in.

2.4 Build Life Cycle

- In Maven, process for building and distributing artifact is clearly defined in the form of life cycle.
- Each lifecycle contains phases in a specific order, and zero or more goals are attached to each phase.
- For example, the compile phase invokes a certain set of goals to compile a set of classes.
- Similarly phases are available for testing, installing artifacts,..
- There are three standard lifecycles in Maven
  - Clean
  - default (sometimes called build)
    - Handle project deployment
  - site

2.4 Build Life Cycle

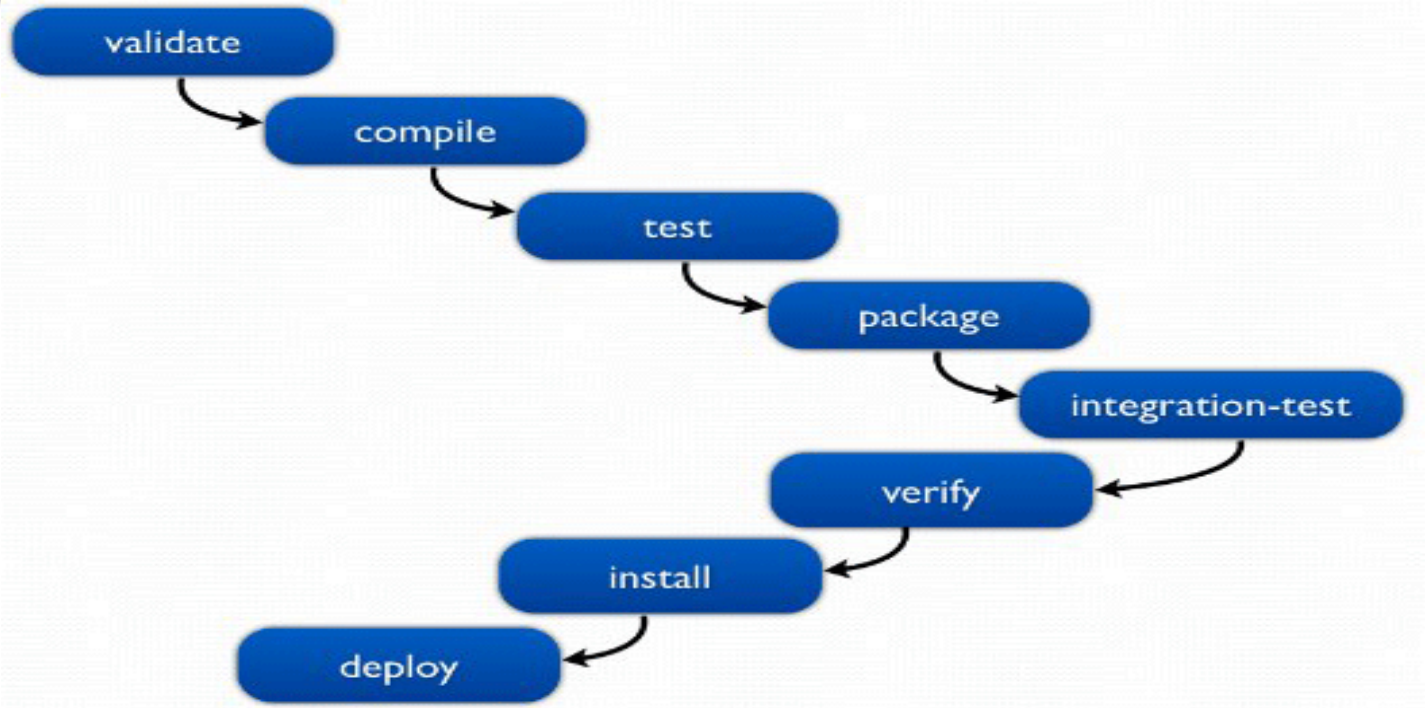
- clean lifecycle handles the cleaning of all project files generated by a previous build.
- Running mvn clean invokes the clean lifecycle

pre-clean	**executes processes needed prior to the actual project cleaning**
clean	**remove all files generated by the previous build**
post-clean	**executes processes needed to finalize the project cleaning**

2.4 Build Life Cycle

- The default lifecycle handles your project deployment.
- Some Key Phases in default life cycle are:
  - validate
  - compile
  - Test
  - Package
  - integration-test
  - Install
  - deploy

2.4 Build Life Cycle



2.4 Build Life Cycle

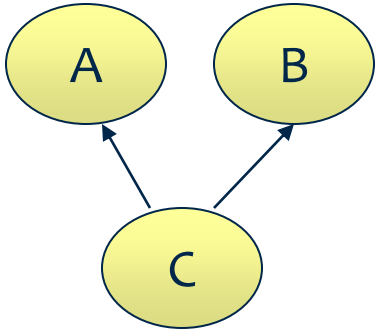
- Site lifecycle handles the creation of your project's site documentation.
- You can generate a site from a Maven project by running the following command:
  - mvn site

pre-site	executes processes needed prior to the actual project site generation
site	generates the project's site documentation
post-site	executes processes needed to finalize the site generation, and to prepare for site deployment
site-deploy	deploys the generated site documentation to the specified web server

2.5 Dependency Management

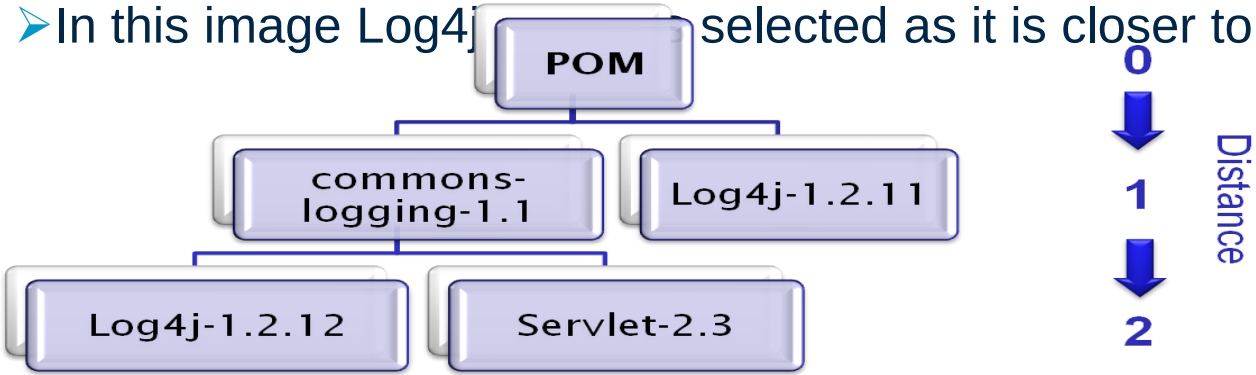
- The dependency management is a mechanism for centralizing dependency information.
- In Maven, Dependencies are defined in the POM.

```
<project ...>
 ...
 <dependencies>
  <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>3.8.1</version>
   <scope>test</scope>
  </dependency>
 </dependencies>
</project>
```



2.6 Resolving Dependency Conflicts

- Conflicts arise in Maven when the same dependency (Ex. Log4j) of different version is identified in dependency graph.
- While resolving such conflicts Maven traverses the dependency in a top down manner and selects the version “nearest” to the top of the tree.
- For an Example, looking for Log4j-1.2.12 dependency in a dependency graph as shown below.
- In this image Log4j selected as it is closer to the root of the tree.



2.6 Resolving Dependency Conflicts

- Ways to resolve dependency conflicts:
  - To manually resolve conflicts, remove the incorrect version from the tree, or override both with the correct version.
    - Identify the source of the incorrect version by running Maven with the -X flag
    - Once the path to the version has been identified, exclude the dependency from the graph by adding an exclusion to the dependency.
  - Explicit specification of dependency version in pom.xml file

....

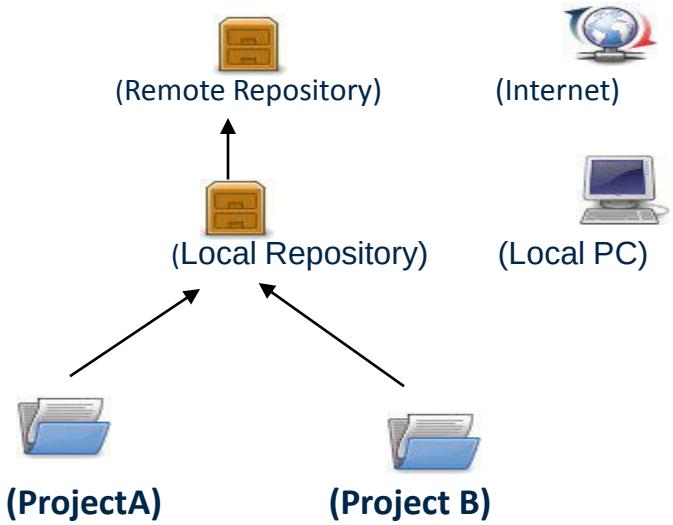
```
<dependencies>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
             <version>1.2.12</version>
        </dependency>
</dependencies>
```

....

2.7 Repositories

- Repositories store a collection of artifacts used by Maven during dependency resolution for a project.
- An artifact is a resource generated by maven project usually bundled as a JAR, WAR, EAR, or other code-bundling type.
- For an example, junit.jar is an artifact.
- An artifact in repositories can be uniquely identified using coordinates:
  - The group ID
  - The artifact ID
  - The version
- Maven has two types of repositories:
  - Local
  - Remote

2.7 Repositories



Remote repository:

– Provides software artifacts (dependencies) for download

– E.g. repo1.maven.org houses Maven's central repository

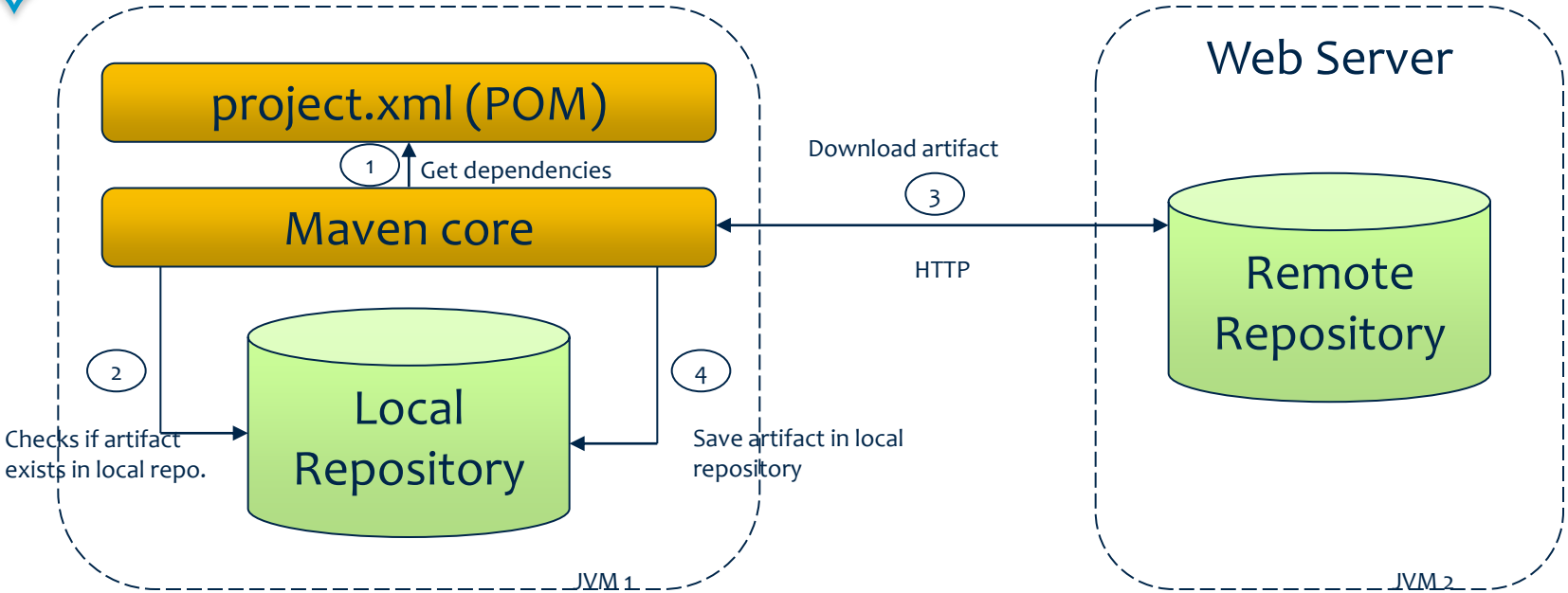
Local repository:

– Copy on local computer which is a cache of the remote downloads

– May contain project-local build artifacts as well

– Located in USER_HOME/.m2/repository

2.7 Repositories



Lesson Summary

- Maven Overview
  - What is Maven?
  - Comparison of Maven with Ant
  - Maven's Objective
- Maven's Principles
- Benefits of Maven
- Elements of Maven
- POM, Standard Directory Structure, Build Life Cycle,
- Plug-in, Dependency Management
- Resolving Dependency Conflicts, Repositories