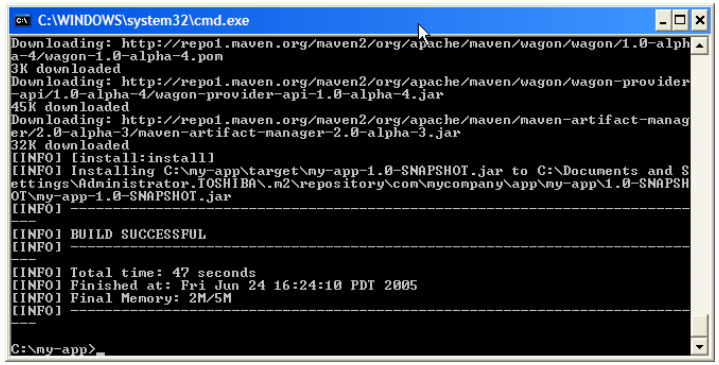
Maven

Runtime project management and comprehension tool

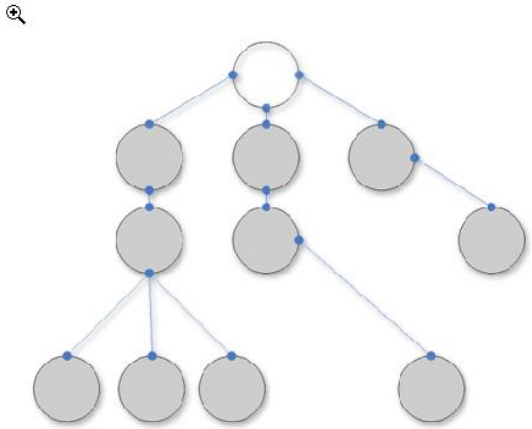
What is Maven?

A build tool

- Compile
- Execute tests



A dependency management tool



A documentation tool

Apply patterns to project build infrastructure

Maven is really a process of applying **patterns** to a build infrastructure in order to provide a coherent view of software projects.

Provides a way to help with managing:

- Builds
- Documentation
- Reporting
- Dependencies
- Software Configuration Management
- Releases

Objectives

- Make the development process visible or transparent
- Provide an easy way to see the health and status of a project
- Decreasing training time for new developers
- Bringing together the tools required in a uniform way
- Preventing inconsistent setups
- Providing a standard development infrastructure across projects
- Focus energy on writing applications

Benefits

- Standardization
- Fast and easy to set up a powerful build process
- Greater momentum vs. Ant – it is now becoming legacy and not moving fast ahead.
- Dependency management (automatic downloads)
- Project website generation, Javadoc
- Repository management
- Extensible architecture

Maven : vocabulaire

- Plugin
  - Extension of the basic application, with a set of goals
- Goal
  - Task proposed by a plugin allowing to launch a certain number of actions when it is invoked by `mvn plugin:goal`. Parameterized by `-Dparam=value`
- Maven lifecycle phase
  - Phase of the software development cycle, usually associated with goals and executed by `mvn phase`
- Artifact
  - Application whose development is managed via Maven
- POM
  - xml file describing the specificities of the project

Common project metadata format

- 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>4.0.0</modelVersion>
  <groupId>org.codehaus.cargo</groupId>
  <artifactId>cargo-core-api-container</artifactId>
  <name>Cargo Core Container API</name>
  <version>0.7-SNAPSHOT</version>
  <packaging>jar</packaging>
  <dependencies/>
  <build/>
[…]
```

Minimal POM

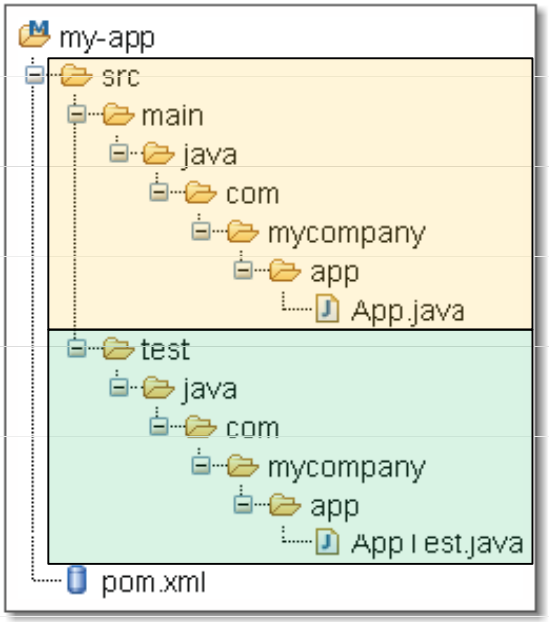
Use Inheritance

Standard directory organization

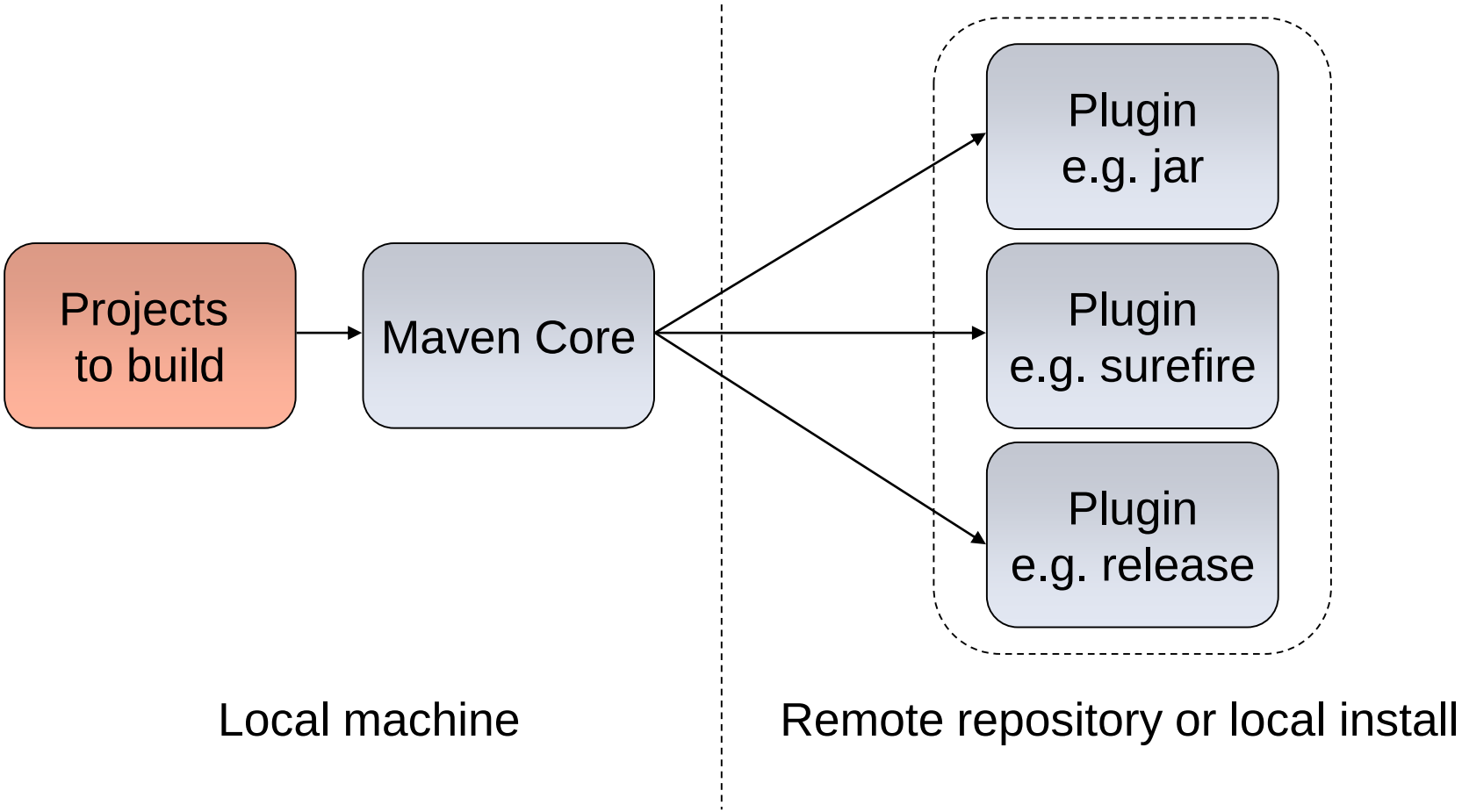
- Having a common directory layout would allow for users familiar with one Maven project to immediately feel at home in another Maven project.

Convention over configuration

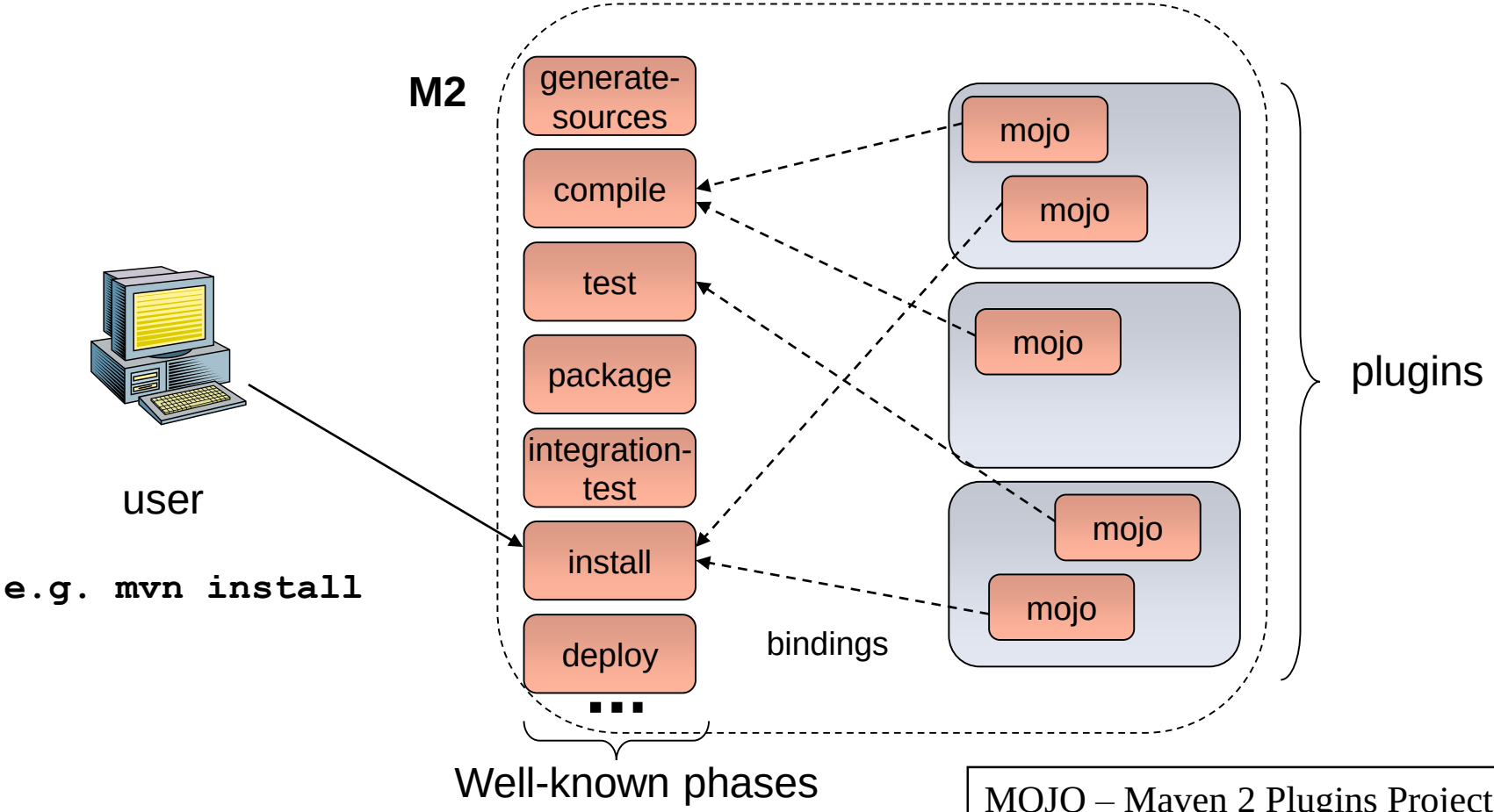
src/main/java	Application/Library sources
src/main/resources	Application/Library resources
src/main/filters	Resource filter files
src/main/assembly	Assembly descriptors
src/main/config	Configuration files
src/main/webapp	Web application sources
src/test/java	Test sources
src/test/resources	Test resources
src/test/filters	Test resource filter files
src/site	Site
LICENSE.txt	Project's license
README.txt	Project's readme



Maven Architecture



Common way to build applications



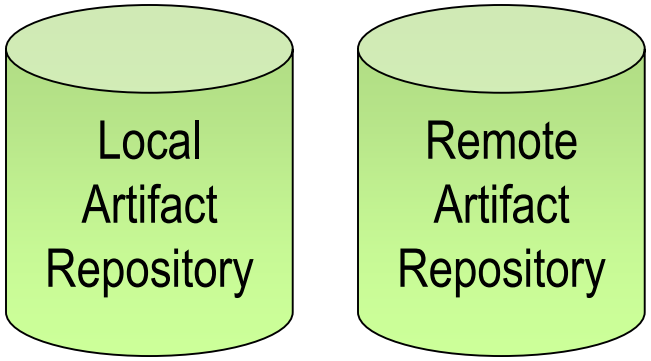
MOJO – Maven 2 Plugins Project
http://mojo.codehaus.org/

Plugins and goals examples

- Plugin compiler
  - For java code compilation
  - Goal: compile, tests-compile
- Plugin surefire
  - For test execution
  - Goal: test
- Plugin Jar
  - For packaging the project source code
  - Goal: jar creation
- ...

Artifact repositories (1/3)

- Used to store all kind of artifacts
  - JARs, EARs, WARs, NBMs, EJBs, ZIPs, plugins, …
- All project interactions go through the repository
  - No more relative paths!
  - Easy to share between team

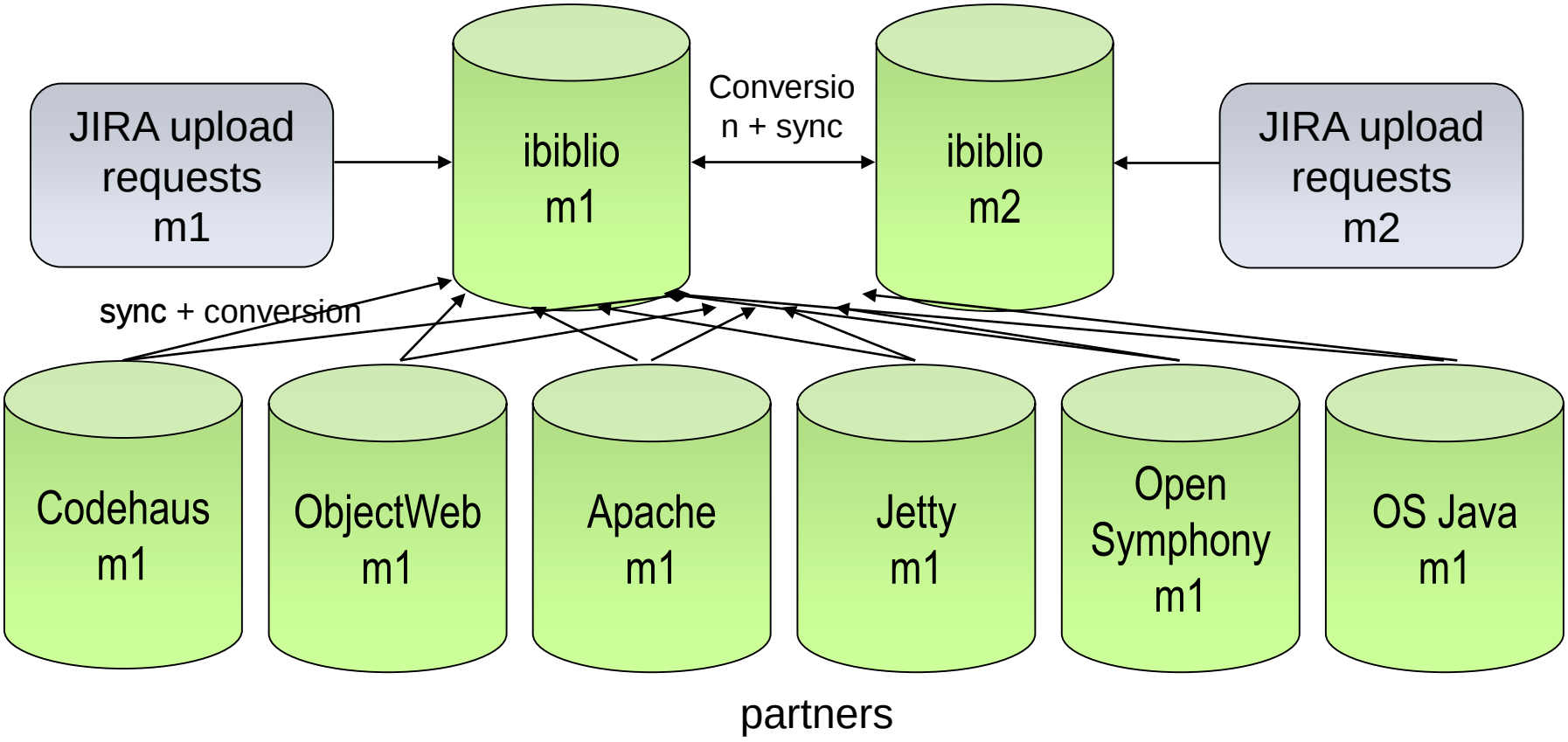


e.g. `http://ibiblio.org/maven2`

```
<repositories>
  <repository>
    <id>maven2-snapshot</id>
    <releases>
      <enabled>true</enabled>
    </releases>
    <name>Maven Central Development Repository</name>
    <url>http://snapshots.maven.codehaus.org/maven2</url>
    <layout>legacy|default</layout>
  </repository>
</repositories>
```

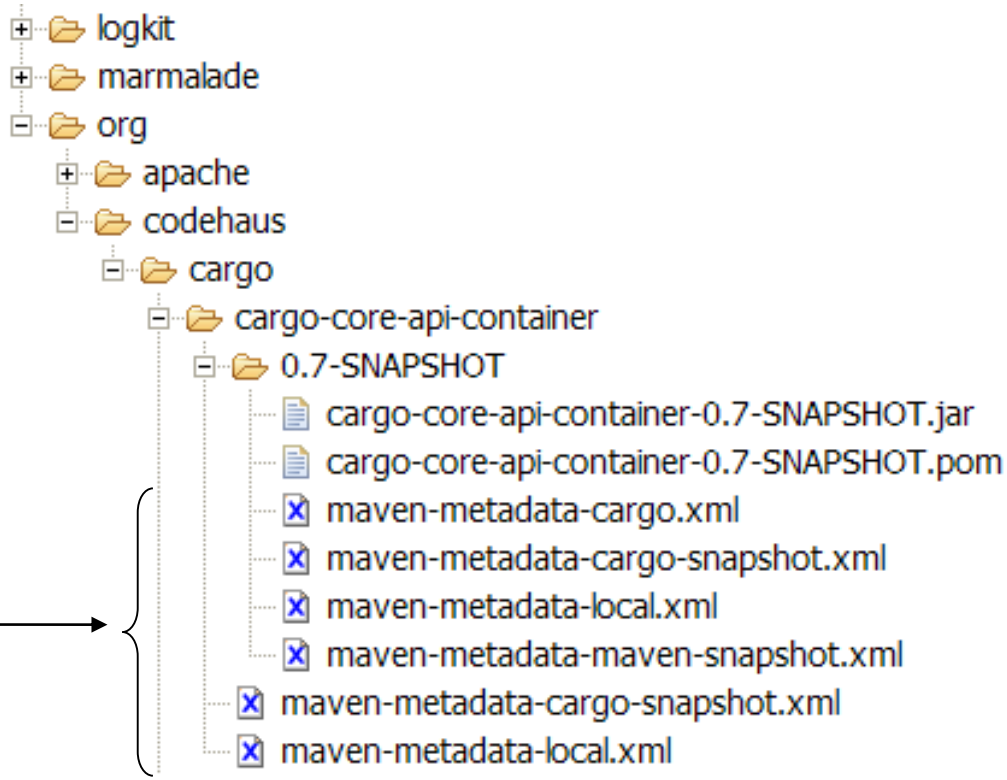
Artifact repositories (2/3)

- Some public remote repositories



Artifact repositories (3/3)

- Hierarchical structure
- Automatic plugin download
- Plugins are read directly from the repository
- Configurable strategies for checking the remote repositories for updates
  - Daily check by default for plugin and ranges updates
- Remote repositories contain Metadata information
  - Releases, latest, and more to come



POM example

```
Example
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  ..
</project>
```

Identifier (unique) of the project

Type of project :
Jar, war, ear, bundle, ….

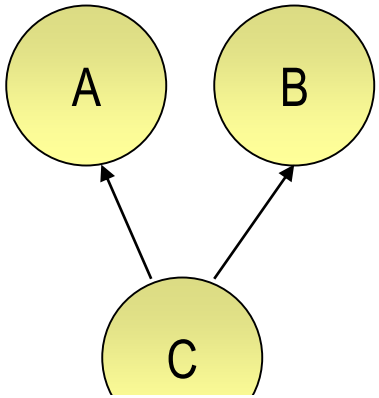
Declared dependencies

Declared dependency to junit version 3.8.1

Dependency scope w.r.t. life cycle (compile, runtime, test, …)

Dependency management (1/2)

- Maven uses binary dependencie



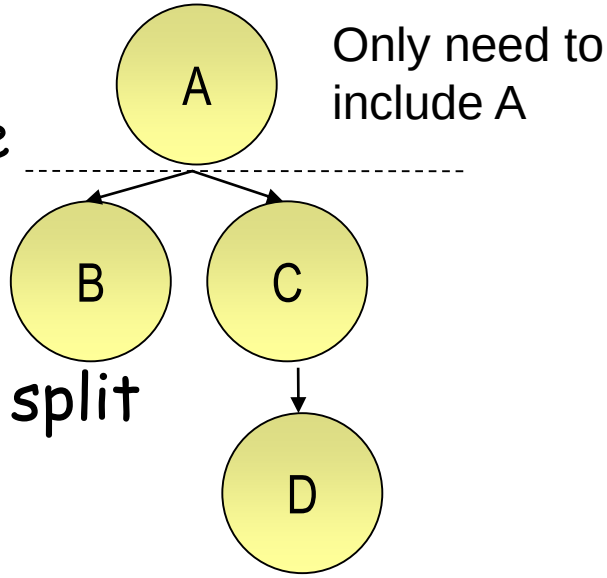

```
<dependencies>
  <dependency>
    <groupId>com.acme</groupId>
    <artifactId>B</artifactId>
    <version>[1.0,)</version>
    <scope>compile</scope>
  </dependency>
</dependencies>
```

« any version after 1.0 »

Range	Meaning
`(,1.0]`	Less than or equal to 1.0
`[1.2,1.3]`	Between 1.2 and 1.3 (inclusive)
`[1.0,2.0)`	Greater than or equal to 1.0, but less than 2.0
`[1.5,)`	Greater than or equal to 1.5
`(,1.1),(1.1,)`	Any version, except 1.1

Dependency management (2/2)

- Transitive dependencies
  - Possibility to exclude some depndencies
  - Need good metadata
  - Ideally projects should be split
- SNAPSHOT handling
  - Always get latest
- Automatic dependency updates
  - By default every day



Version numbering

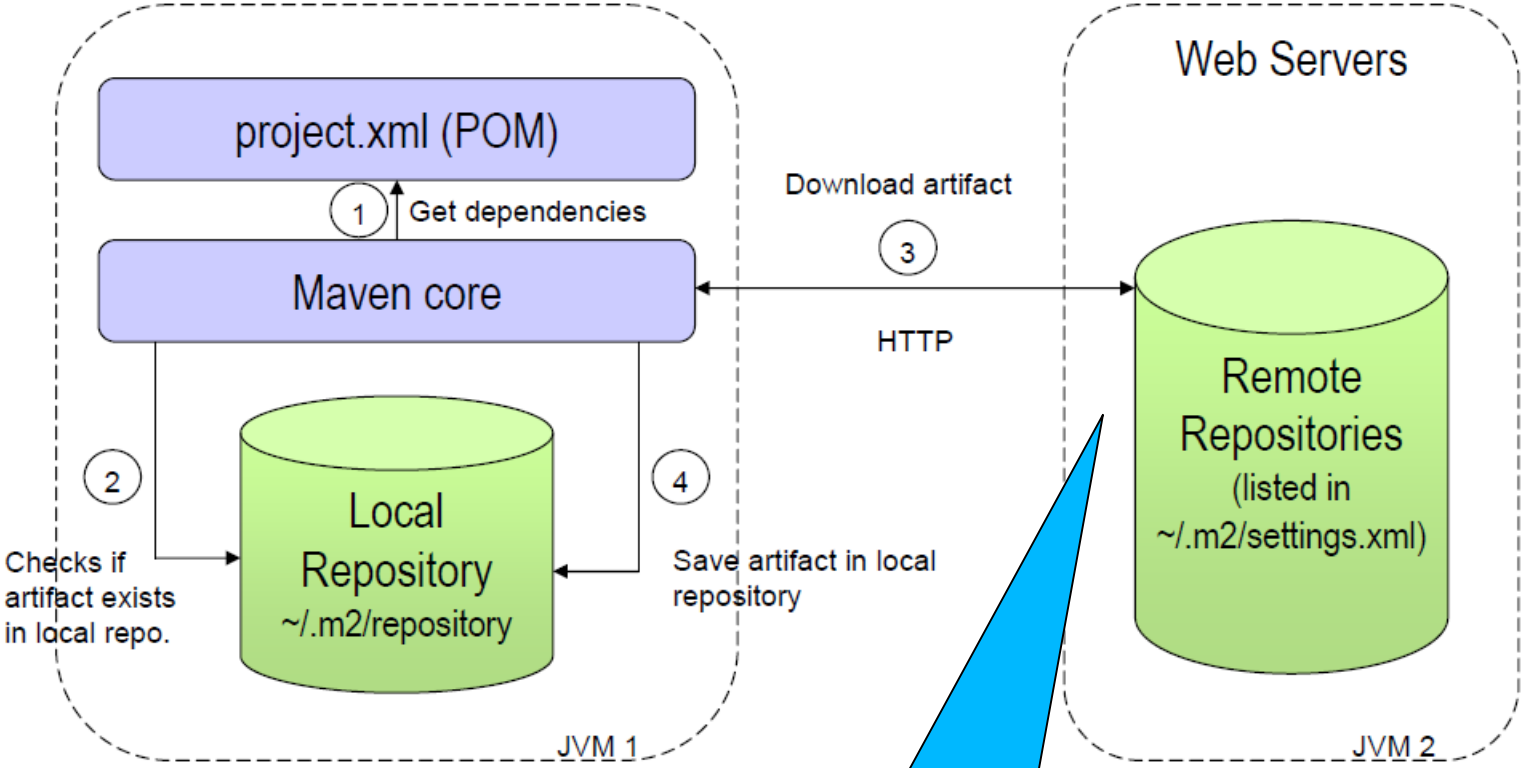
- <major>.<mini>[.<micro>][-<qualifier>[-<buildnumber>]]
  - Major: major changes
    - No guarantee for retro compatibility
  - Mini: new functionalities
    - Guarantee should be for retro compatibility
  - Micro: corrective changes (bug fixing)
- Qualifiers
  - SNAPSHOT (Maven) : version in evolution from most recent source files
  - alpha1 : version alpha (very instable and incomplete)
  - beta1, b1, b2 : version beta (instable)
  - rc1, rc2 : release candidate
  - m1, m2 : milestone
  - ea : early access
  - …
- Ordering
  - 1.1.1 < 1.1.2 < 1.2.2
  - 1.1.1-SNAPSHOT < 1.1.1
  - 1.1.1-alpha1 < 1.1.1-alpha2 < 1.1.1-b1 < 1.1.1-rc1 < 1.1.1-rc2 < 1.1.1

Installation and Setup

- Download Maven 3 from **`http://maven.apache.org/`**
- Add Maven's bin directory to PATH
- Ensure JAVA_HOME is set to SDK
- Run **`mvn -version`** to test install

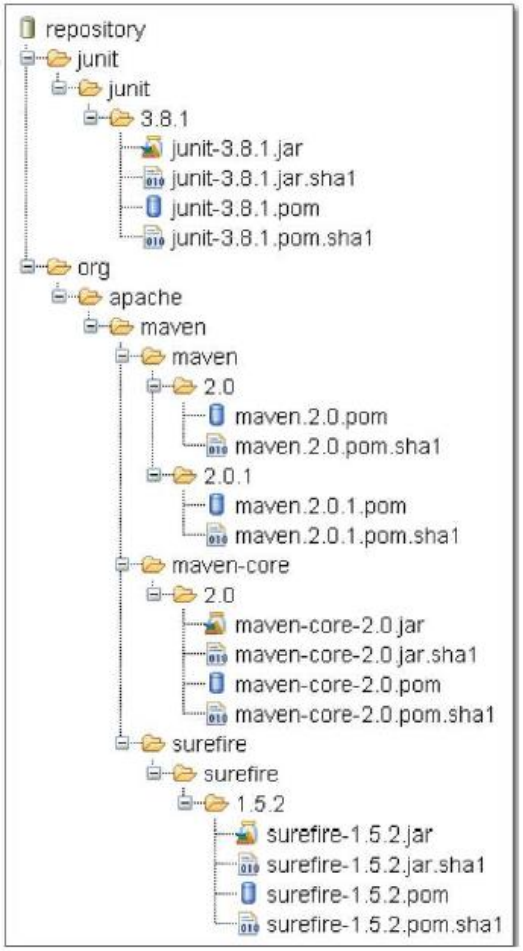
```
C:\Documents and Settings\alina>mvn -version
Maven version: 3.0.4
Java version: 1.6.0_30
```

Dependecies search



Maven central has more than ***200 000*** artifacts in 2018

Installing JARs to loc repository



- Sometimes you need to put some spe your local repository for use in your
- The JARs must be placed in the cor order for it to be correctly picked u
- To install a JAR in the local reposito following command:

```
mvn install:install-file -Dfile=<path-to-file> -DgroupId=<group-id> \
-DartifactId=<artifact-id> -Dversion=<version> -Dpackaging=jar
```

- Now can include dependency in pom.xml:

```
<dependency>
    <groupId><group-id></groupId>
    <artifactId><artifact-id></artifactId>
    <version><version></version>
</dependency>
```

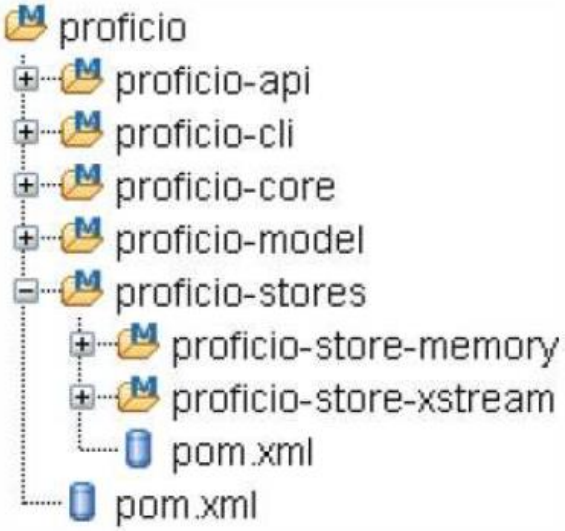
Overview of common Goals

- **clean** – clean the current project
- **validate** - validate the project is correct and all necessary information is available
- **compile** - compile the source code of the project
- **test** - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- **package** - take the compiled code and package it in its distributable format, such as a JAR
- **integration-test** - process and deploy the package if necessary into an environment where integration tests can be run
- **install** - install the package into the local repository, for use as a dependency in other projects locally
- **deploy** - done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects

Hierarchical organization

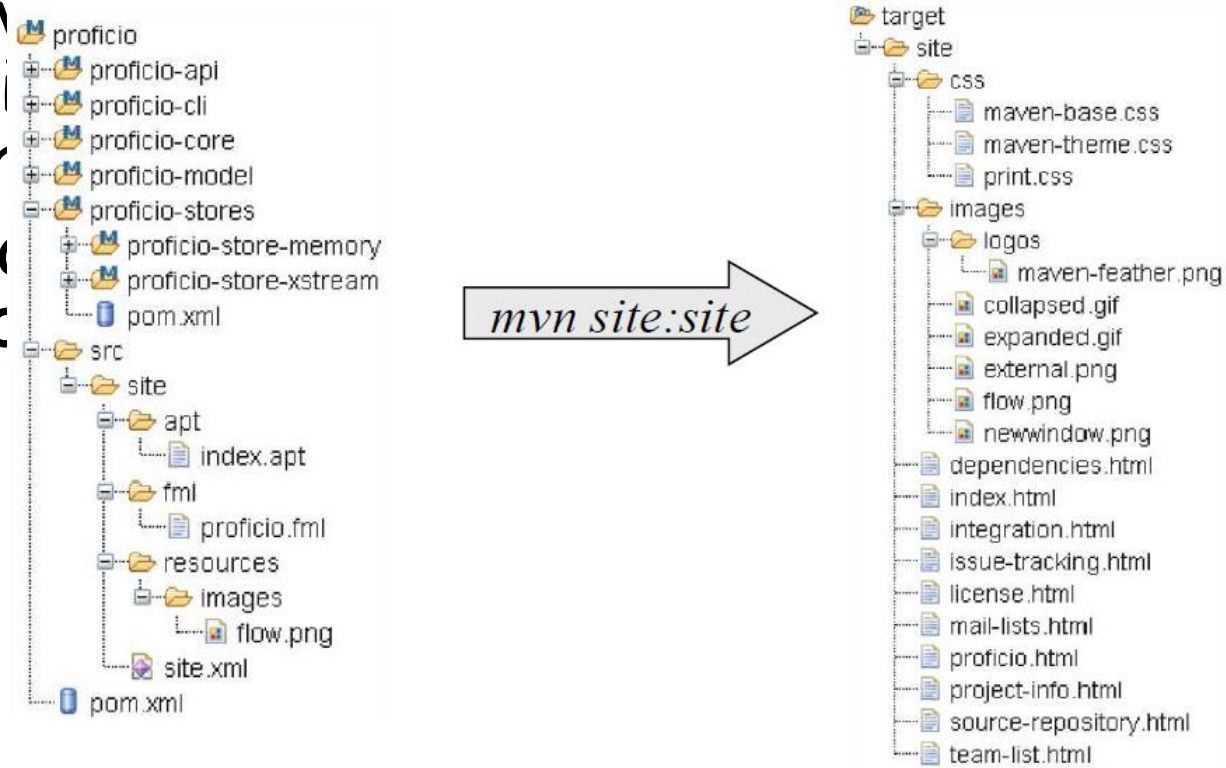
- You can organize your development in sub-projects (n levels)
- Create super POM per level
  - Group common plugins/goals per level
- Sub-projects(called modules) inherite the super POM

- <parent> … </parent>
- <modules>
  - <module> … </module>
  - …
- </modules>



Creating project website

- Run: mvn site
- Let the build run, it'll start downloading and creating things left and right
- Ev p
  wi le
  pr
- Ja
  co



More stuff

- Automatically generate reports, diagrams, and so on through Maven / the project site
- Internationalization – create different language project websites
- Create projects within projects (more pom.xml files inside sub dirs\jars), with different build stats and so on
- Maven can make .war files, EJBs, etc.

Using Maven Plugins

- Whenever you want to customise the build for a Maven project, this is done by adding or reconfiguring plugins
- For example, configure the Java compiler to allow JDK 5.0 sources
- Plugins in Maven 3.0 look much like a dependency

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

Using Maven Plugins

- Plugin = {<goal,MOJO>}
- MOJO= Maven POJO //@s dcoLet, attached to a phase

```
package sample.plugin;
import org.apache.maven.plugin.AbstractMojo;
import org.apache.maven.plugin.MojoExecutionException;
/**
 * Says "Hi" to the user.
 * @goal sayhi
 * @phase compile
 */
public class GreetingMojo extends AbstractMojo {
   /** The greeting to display.
    * @parameter alias="message" expression="Hello, world (from ${project.groupId}:${project.artifactId})" */
   private String greeting;

/** The classpath.
  * @parameter expression="${project.compileClasspathElements}"
  * @required
  * @readonly */
  private List classpathElements;

   public void execute() throws MojoExecutionException {
     getLog().info(greeting);
     getLog().info("Project classpath: " + classpathElements().toString().replace( ',', ':' ));
   } }
```

phase et but durant laquelle execute() est appelé

paramètre renseigné dans <configuration>

Integer, …, String, List, Properties, Map, Object, File, URL, …

paramètre issue du pom

Maven Plugins

- **AlmostPlainText**
- **Maven Axis**
- **Maven Cobertura**
- **Maven DB2**
- **Dbunit**
- **Debian Package**
- **Maven DotUml**
- **Doxygen**
- **Maven Files**
- **FindBugs**
- **Maven flash**
- **Help**
- **Maven IzPack**
- **Java Application**
- **Maven JAVANCSS**
- **Maven JAXB**
- **JUNITPP**
- **Kodo**
- **Maven Macker**
- **SDocBook**
- **Sourceforge**
- **Maven SpringGraph**
- **RPM Plugin**
- **Runtime Builder**
- **Strutsdoc**
- **Tasks**
- **Maven Transform**
- **Maven UberDist**
- **Maven Vignette**
- **WebSphere 4.0**
- **WebSphere 5 (5.0/5.1)**
- **Maven WebLogic**
- **Canoo WebTest**
- **Wiki**
- **Word to HTML**
- **XML Resume**
- **Maven DotUml**
- **Middlegen**
- **Maven News**

Archetypes

- For reuse, create archetypes that work as project templates with build settings, etc
- An archetype is a project, with its own pom.xml
- An archetype has a descriptor called archetype.xml
- Allows easy generation of Maven projects

Good things about Maven

- Standardization
- Reuse
- Dependency management
- Build lifecycle management
- Large existing repository
- IDE aware
- One directory layout
- A single way to define dependencies
- Setting up a project is really fast
- Transitive dependencies
- Common build structure
- Use of remote repository
- Web site generation
- Build best practices enforcement
- Automated build of application
- Works well with distributed teams
- All artifacts are versioned and are stored in a repository
- Build process is standardized for all projects
- A lot of goals are available
- It provides quality project information with generated site
- Easy to learn and use
- Makes the build process much easier at the project level
- Promotes modular design of code

References

- Maven Home
  http://maven.apache.org/
- Maven Getting Started Guide
  http://maven.apache.org/guides/getting-started/index.html
- Steps for creating a Maven-based Website
  http://www.javaworld.com/javaworld/jw-02-2006/jw-0227-maven_p.html
/
- Maven Integration for Eclipse
  http://m2eclipse.codehaus.org/

Time for TP

TP1.Maven.pdf