

Construction et gestion de développement avec Maven 3.0

Roozbeh SOLTANI
roozbehsltn@gmail.com



Maven : Kesako ?

- Un outil de construction d'application
 - Génère une application « déployable » à partir d'un code source
 - Compile Exécute des tests
 -
- Un outil de gestion de développement
 - Gère aussi
 - Documentation & Rapports
 - Site web
 - ...

Maven : choix sous-jacents

- Privilégier la standardisation à la liberté (*Convention over Configuration*)
 - Structure standard des répertoires d'une application
 - Cycle de développement standard d'une application
 - Maven se débrouille souvent tout seul !
 - Factoriser les efforts
 - Un dépôt global regroupant les ressources/bibliothèques communes
 - Des dépôts locaux
 - Un dépôt personnel (~/.m2)
 - Multiplier les possibilités
 - Une application légère
 - De nombreux plugins, chargés automatiquement au besoin
-
-

Maven : vocabulaire

- Plugin

Extension de l'application de base, proposant un ensemble de buts

- But (*Goal*)

Tâche proposée par un plugin permettant de lancer un certain nombre d'action lorsqu'il est invoqué par `mvn plugin:but`.

Paramétré par `-Dparam=valeur`

- Phase (*Maven Lifecycle Phase*)

Phase du cycle de développement d'un logiciel, généralement associée à des buts et exécutée par `mvn phase`

- Artefact (*Artifact*)

Application dont le développement est géré via Maven

- POM (Project Object Model)

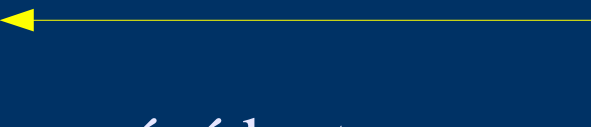
Fichier xml décrivant les spécificités du projets (par rapport à Build.xml, décrit non pas tout, mais juste le « non standard »)

Plugins et buts

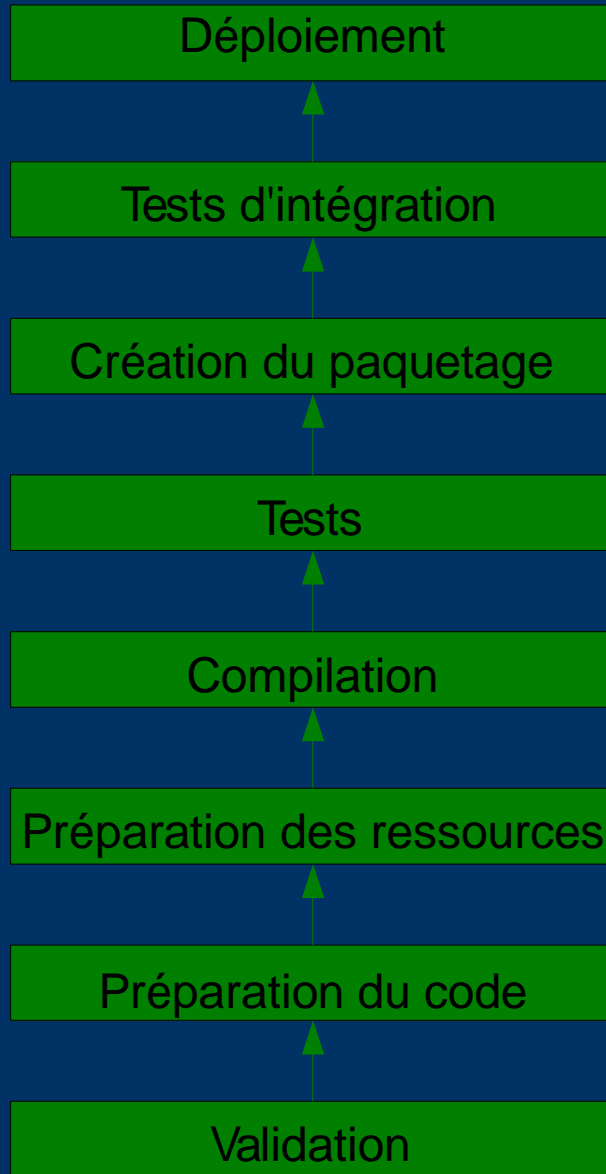
Exemples

- Plugin archetype
 - Pour créer de nouveaux projets standards
 - Buts : generate, create
 - Plugin compiler
 - Pour la compilation de code java
 - But : compile, test-compile
 - Plugin surefire
 - Pour l'exécution des tests
 - But : test
-
-

Buts et phases

- Exécution d'un but
 - Exécution du but seulement
- Exécution d'une phase 
 - Exécution de la phase précédente
 - Exécution du but associé

Etapes du cycle de vie Maven par défaut



Déploiement du code en local ou sur site distant

Préparation et exécution des tests d'intégration

Génération du paquetage à déployer

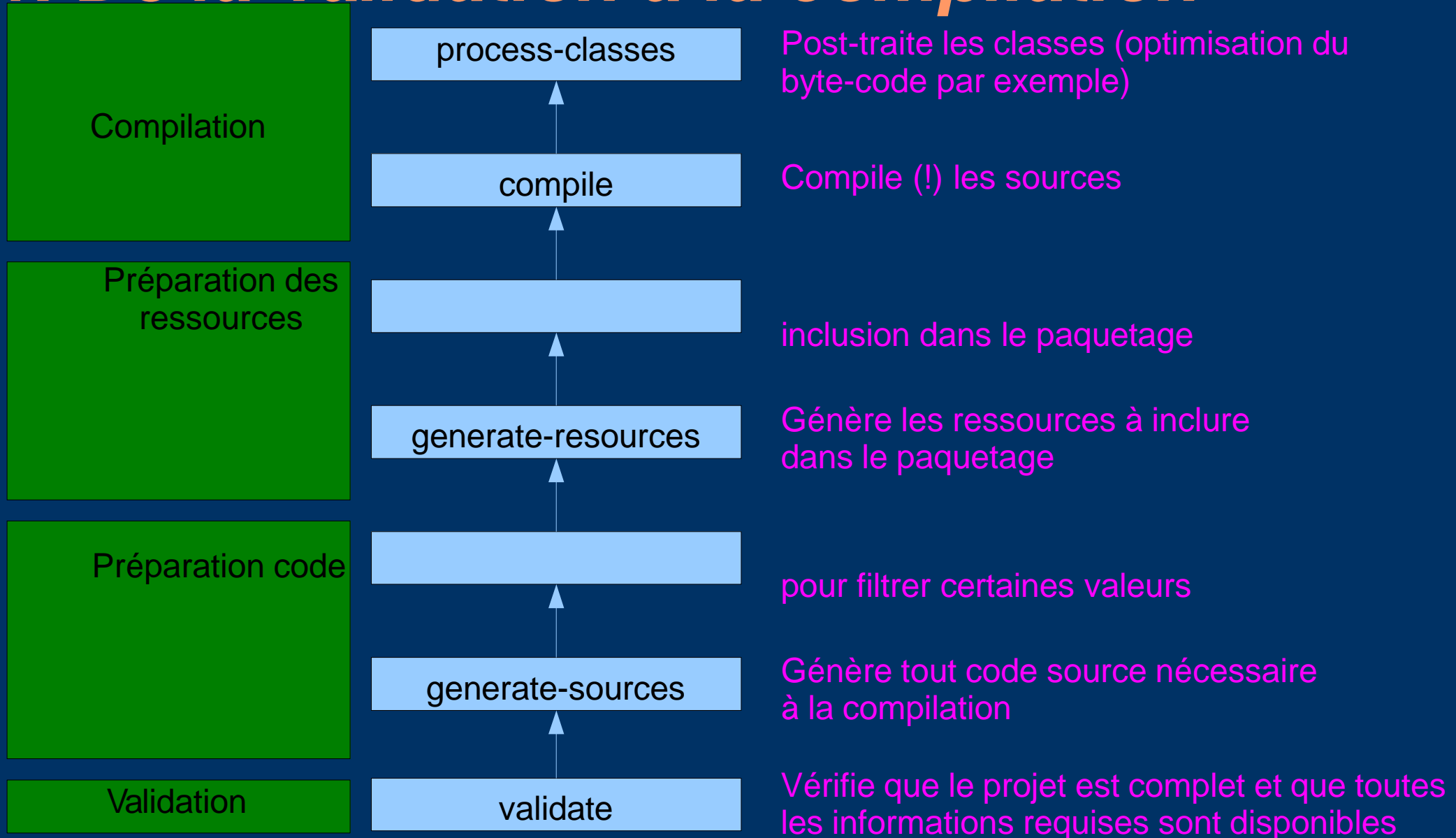
Préparation, compilation et exécution des tests

Compilation du code

Vérification de la complétude du projet

Phases du cycle de vie Maven par défaut

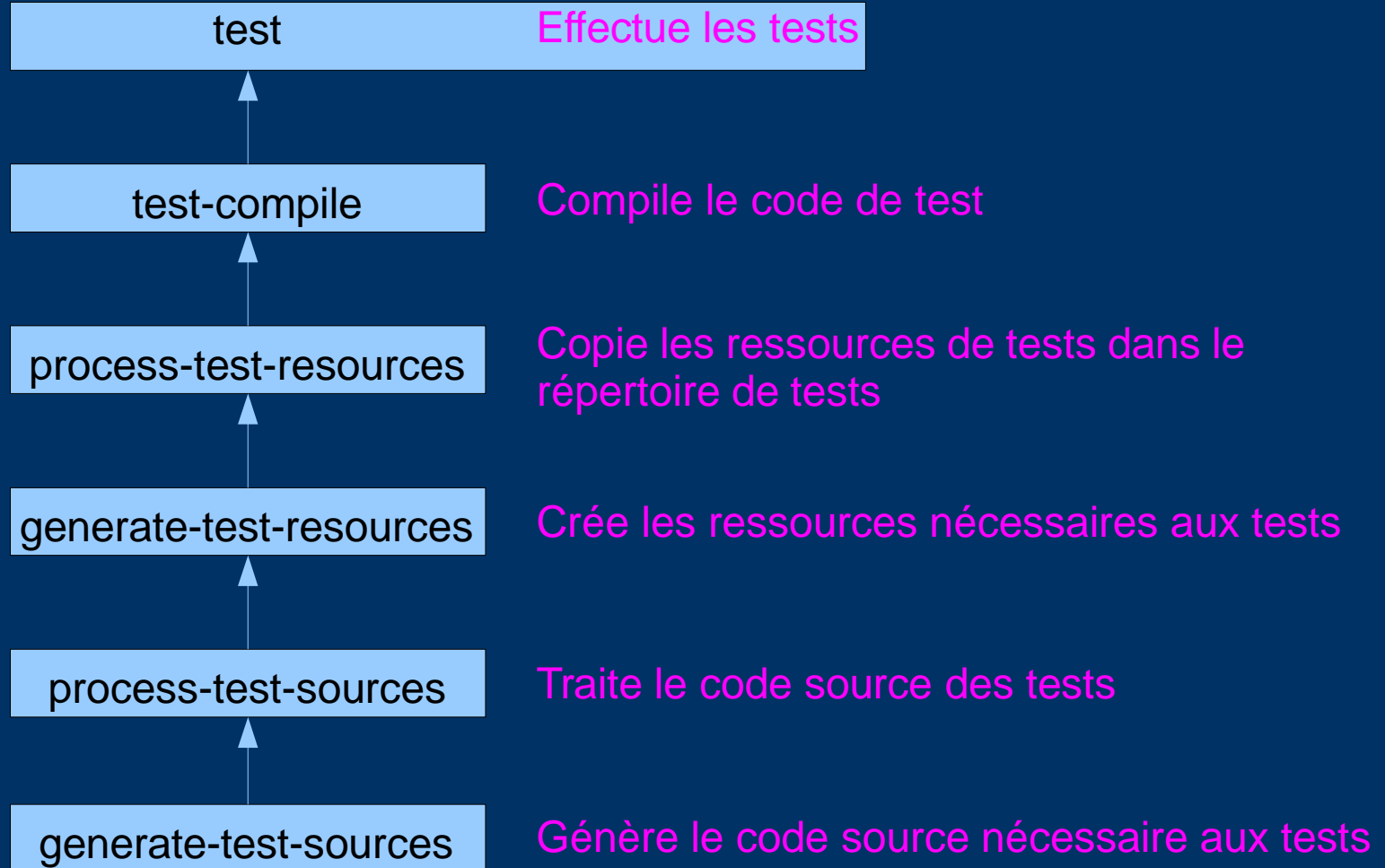
1. De la Validation à la compilation



Phases du cycle de vie Maven par défaut

2. Tests

Tests



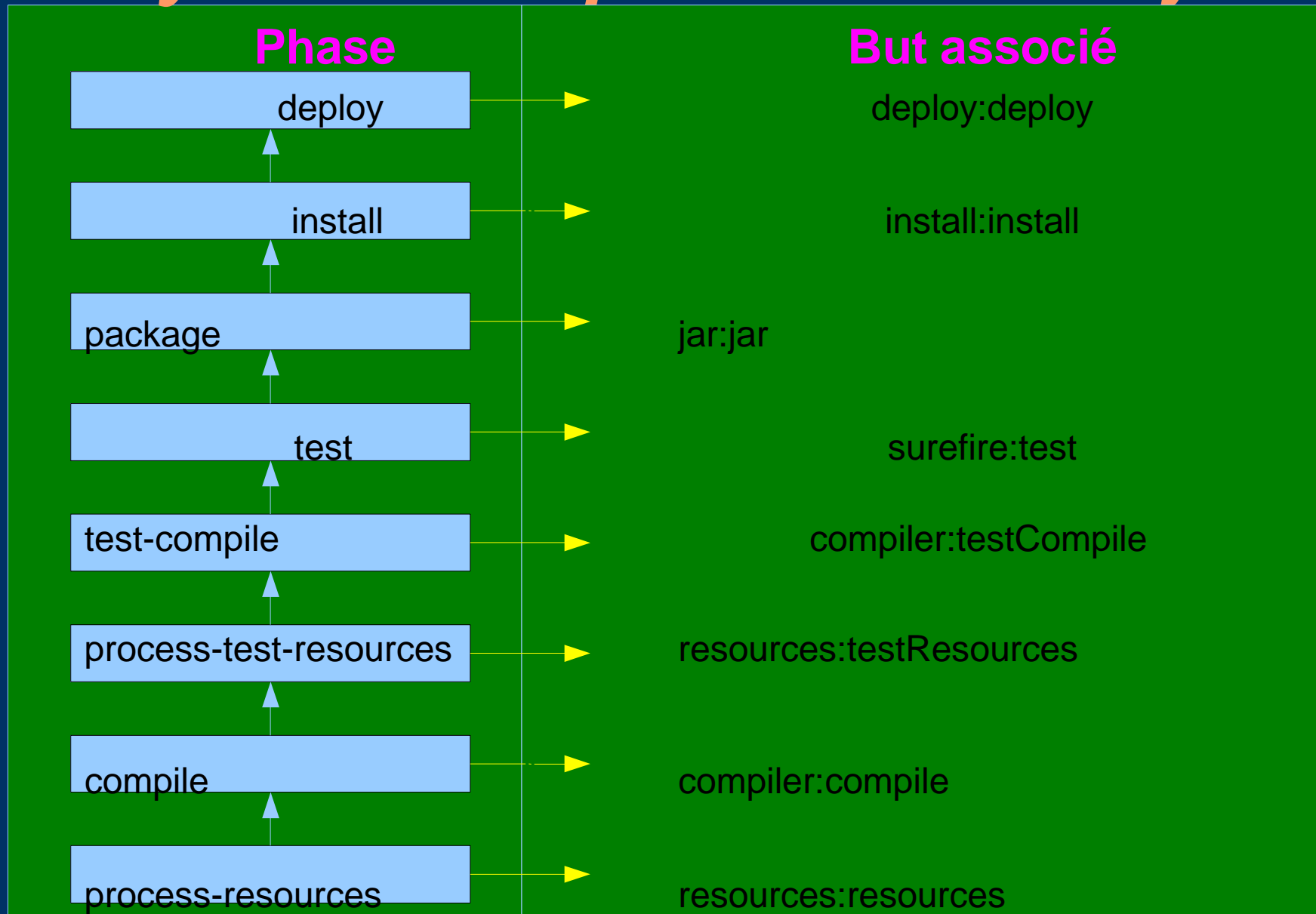
Phases du cycle de vie Maven par défaut

3. Du packaging au déploiement

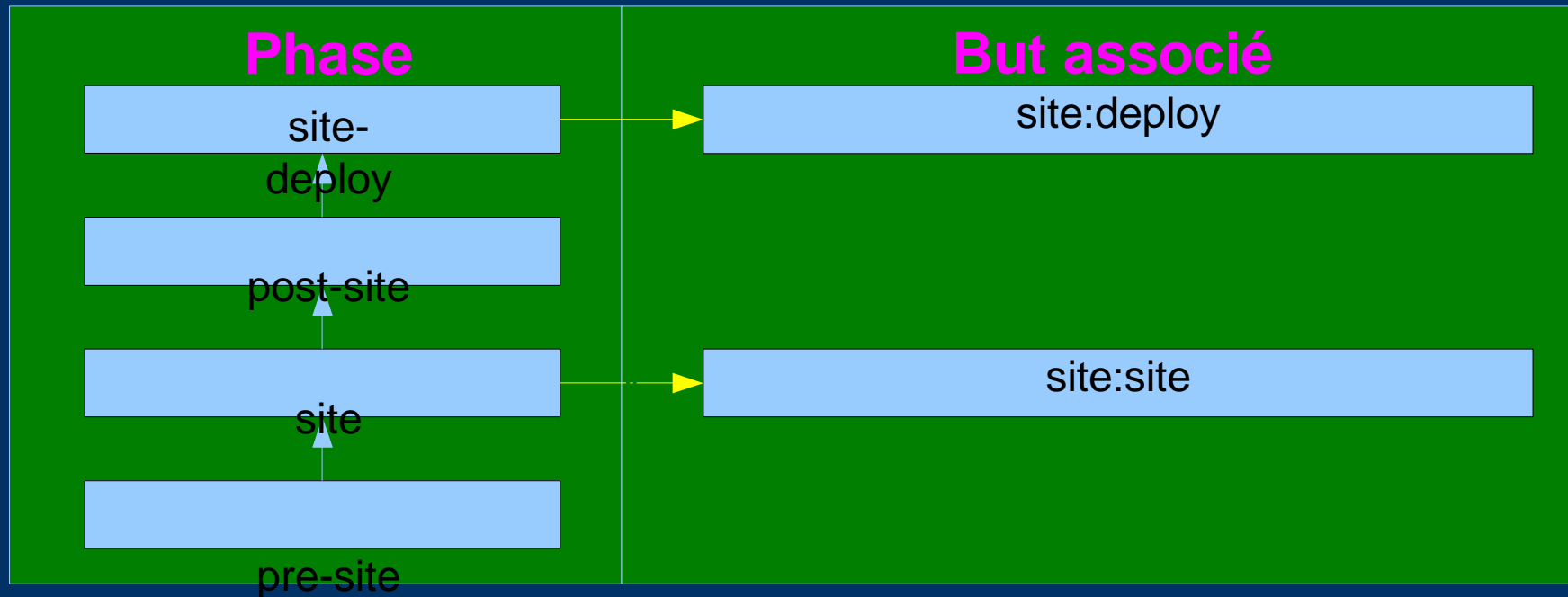


Cas particulier

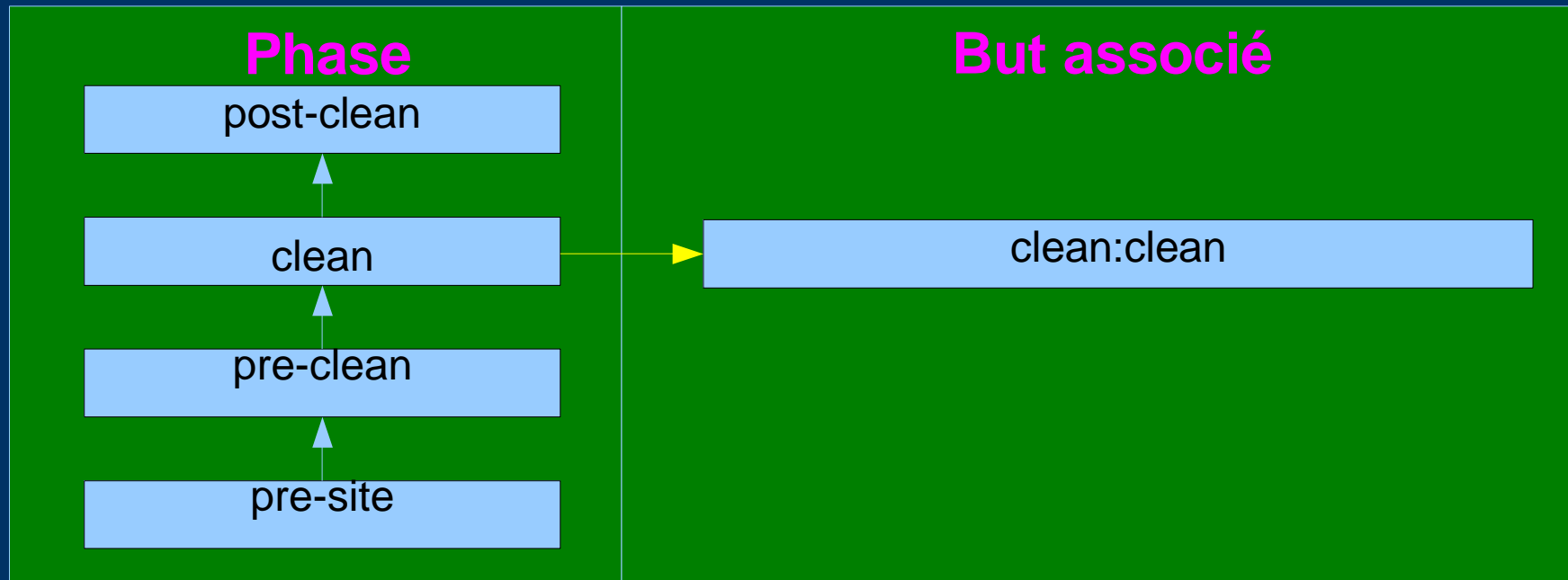
Cycle de vie pour un fichier jar



Phases du cycle de vie Maven pour le site

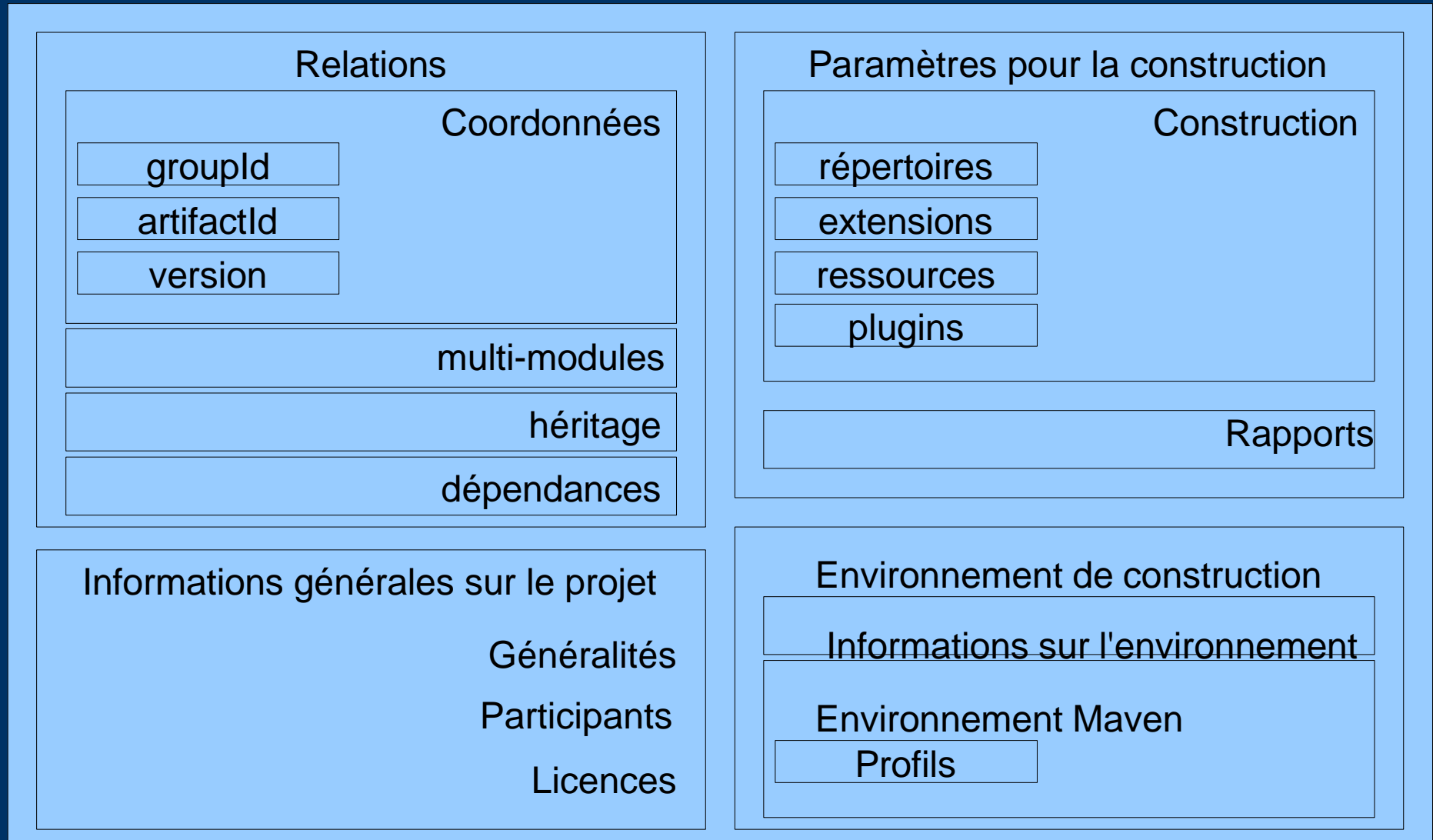


Phases du cycle de vie Maven pour le nettoyage



Project Object Model

Structure générale



Project Object Model

Un premier exemple

```
<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/maven-
v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>edu.mermet</groupId>
<artifactId>EssaiProjetMaven</artifactId>
<packaging>jar</packaging>
<version>1.0-SNAPSHOT</version>
<name>EssaiProjetMaven</name>
<url>http://maven.apache.org</url>
```

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

Coordonnées maven

- Identifiant
 - GroupId
 - En général, le nom du domaine à l'envers
 - ArtifactId
 - Ce que l'on veut
 - Version
 - `majorVersion.minorVersion.incrementalVersion-qualifier`
 - Permet de spécifier une version minimale lors d'une dépendance avec order `<num,num,num,alpha>`
 - Si contient SNAPSHOT, remplacé par dateHeureUTC lors de la génération du package
 - Non importé par défaut
- Packaging
 - Type de packaging à produire (jar, war, ear, etc.)

Structure des répertoires

Répertoire Du Projet

- pom.xml
- src
 - main
 - java
 - resources
 - test
 - java
 - resources
- target
 - classes
 - test-classes

Créer un projet Java depuis les dernières versions

- Maintenant, une version supérieure à java 1.5 est recommandée. Plusieurs solutions :
 - Rajouter les lignes suivantes dans le POM généré avec l'archetype maven-archetype-quickstart :

```
<maven.compiler.source>1.8</maven.compiler.source>  
<maven.compiler.target>1.8</maven.compiler.target>
```
 - Utiliser un autre archetype comme :
`com.github.ngeor:archetype-quickstart-jdk8`

Structure d'un projet nouvellement créé

- Commande

Mvn archetype:generate -DgroupId=edu.mermet
-DartifactId=exemple -Dversion=1.0-SNAPSHOT

- Structure

- exemple

- pom.xml
 - src

- main

- java

- edu

- mermet

- App.java

- test

- java

- edu

- mermet

- AppTest.java



Fichiers de base générés : pom.xml

```
<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/P
OM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>edu.mermet</groupId>
<artifactId>exemple</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>

<name>exemple</name>
<url>http://maven.apache.org</url>
```

```
<properties>
<project.build.sourceEncoding>UTF-8
</project.build.sourceEncoding>
</properties>

<dependencies>

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

</project>
```

Profils

- Principe

Ensemble nommé de configurations dans le pom.xml

- Utilisation

- Permettre de tester des configurations différentes (sur sélection manuelle)
- Permettre une configuration automatique selon la plateforme d'exécution de Maven



Profil nommé avec sélection manuelle

```
<profiles>
<profile>
<id>production</id>
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<configuration>
<debug>>false</debug>
<optimize>>true</optimize>
</configuration>
</plugin>
</plugins>
</build>
</profile>
</profiles>
```

Profil activé automatiquement

```
<profiles>
<profile>
    <id>dev</id>
    <activation>

<jdk>1.5</jdk>
<os>
<name>Windows XP</name>
<family>Windows</family>
<arch>x86</arch>
<version>5.1.2600</version>
</os>
<property>
<name>customProperty</name>
<value>BLUE</value>
</property>
<file>
<exists>file2.properties</exists>
<missing>file1.properties</missing>
</file>
    </activation>

...
</profile>
</profiles>
```

Profil activé par défaut

```
<profiles>
  <profile>
    <id>dev</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    ...
  </profile>
</profiles>
```


Héritage entre POMs

- Principe

- Tout « pom » hérite d'un autre
- POM racine : « super-POM »
- Définition d'un héritage

```
<parent>
```

```
<groupId>groupe</groupId>
```

```
<artifactId>projet</artifactId>
```

```
<version>version</version>
```

```
</parent>
```

- Propriété

- Transitivité
 - On hérite de toutes les propriétés du super-POM, mais on peut les redéfinir
-
-

Quelques configurations pratiques

1. Junit dernière version

```
<dependency>
```

```
  <groupId>junit</groupId>
```

```
  <artifactId>junit</artifactId>
```

```
  <version>4.12</version>
```

```
  <scope>test</scope>
```

```
</dependency>
```

Quelques configurations pratiques

2. Java 8

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Quelques configurations pratiques

3. Jar exécutable

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <version>2.4</version>
            <configuration>
                <archive>
                    <manifest>
                        <mainClass>edu.mermet.App</mainClass>
                    </manifest>
                </archive>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Tests unitaires et Tests d'intégration

- Principe général
 - Les tests unitaires sont lancés par surefire lors de la phase « test »
 - Les tests d'intégration sont lancés par failsafe lors de la phase « integration-test », préalable à la phase « verify »

Distinguer les tests d'intégration

- De base
 - Les tests unitaires sont dans des fichiers :
 - `**/Test*.java`
 - `**/*Test.java`
 - les tests d'intégration sont dans des fichiers :
 - `**/IT*.java`
 - `**/*IT.java`
- Sinon
 - Configurable (voir doc.)

Configuration du plugin failsafe

```
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-failsafe-plugin</artifactId>
<version>2.19.1</version>
<executions>
    <execution>
        <goals>

<goal>integration-test</goal>
    <goal>verify</goal>
        </goals>
    </execution>
</executions>
</plugin>
</plugins>
</build>
```

Intégration d'un rapport au site web

- Configuration

```
<reporting>
```

```
<plugins>
```

```
<plugin>
```

```
<groupId>org.apache.maven.plugins</groupId>
```

```
<artifactId>maven-surfire-report-plugin</artifactId>
```

```
<version>2.19.1</version>
```

```
</plugin>
```

```
< /plugins>
```

```
</reporting>
```

- Exécution

```
mvn site
```
