Maven

Modul 14
Maven

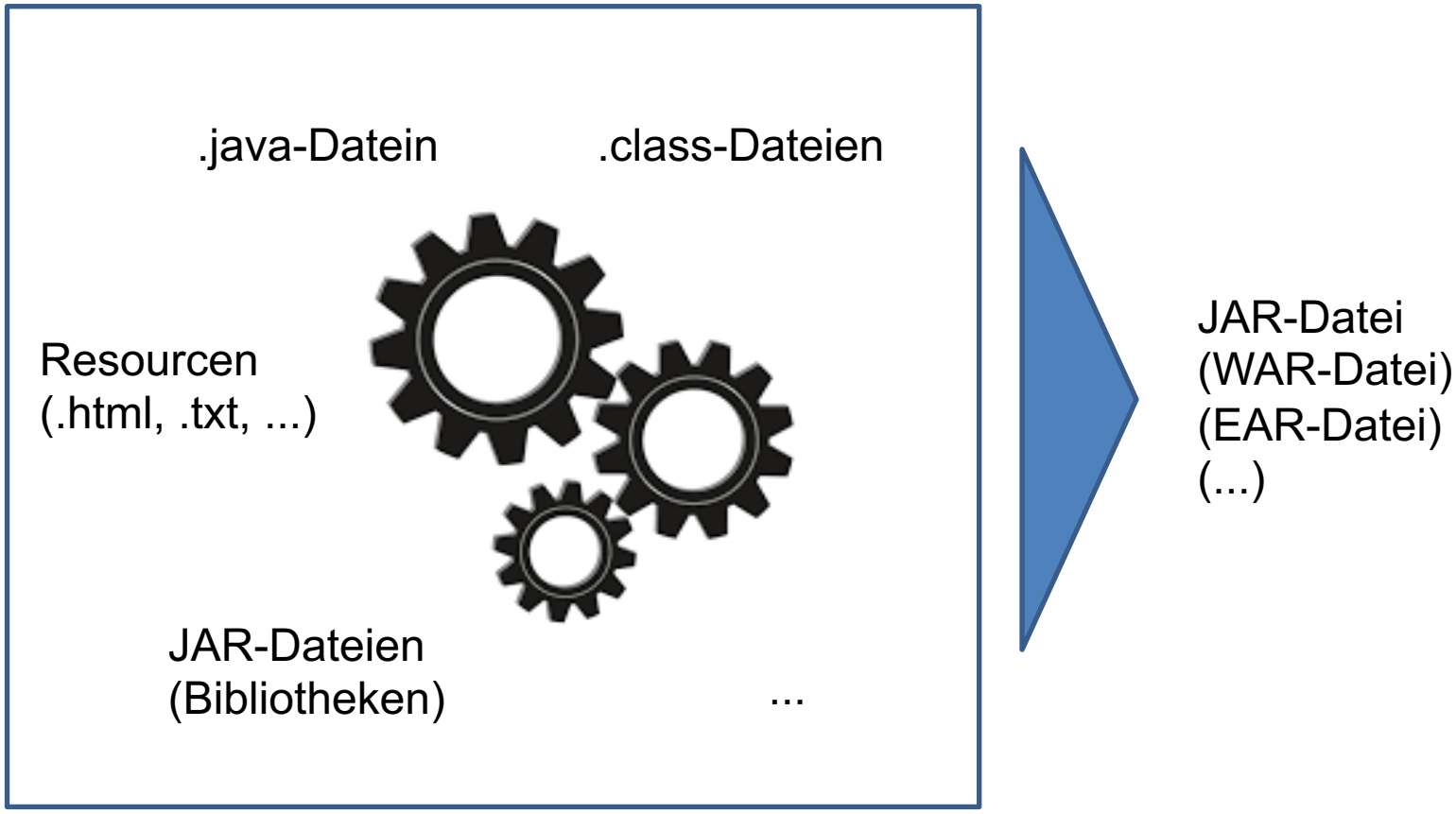
Command Line:

```
C:\> mvn
```

Aktuelle Verison:

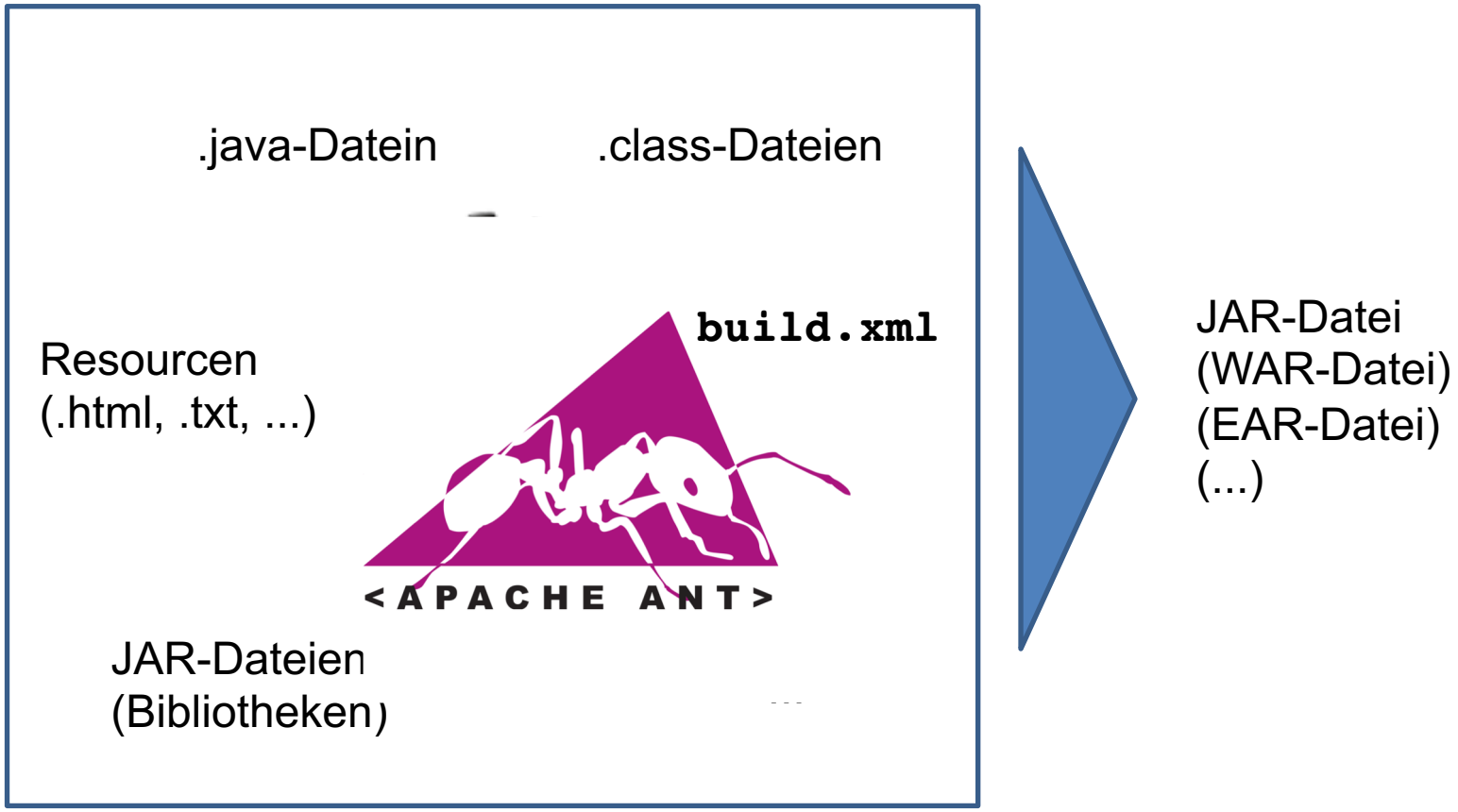
```
maven 3.6.0
```

Build Management System für Java



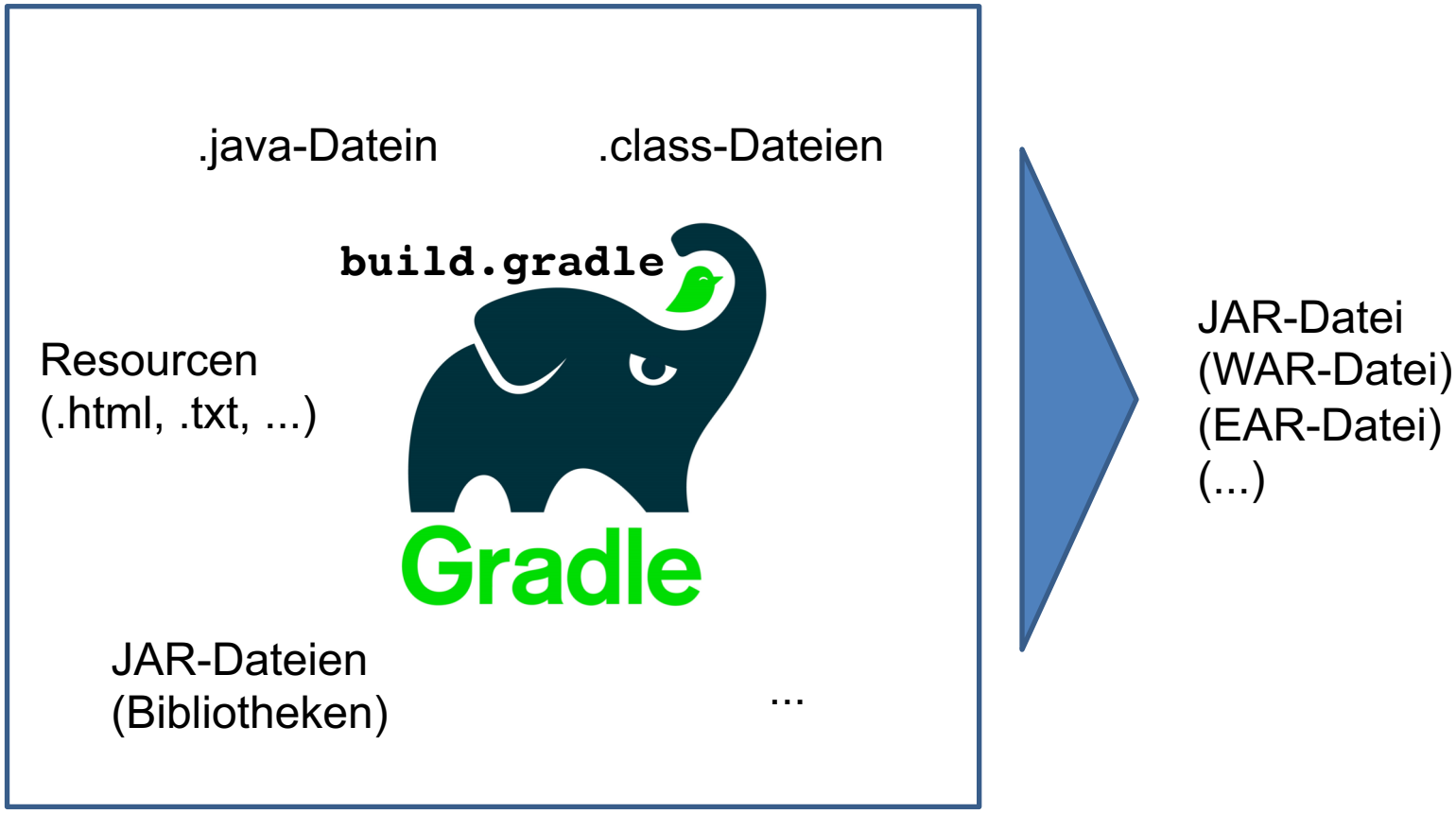
Modul 14
Was ist Maven?

Build Management System für Java



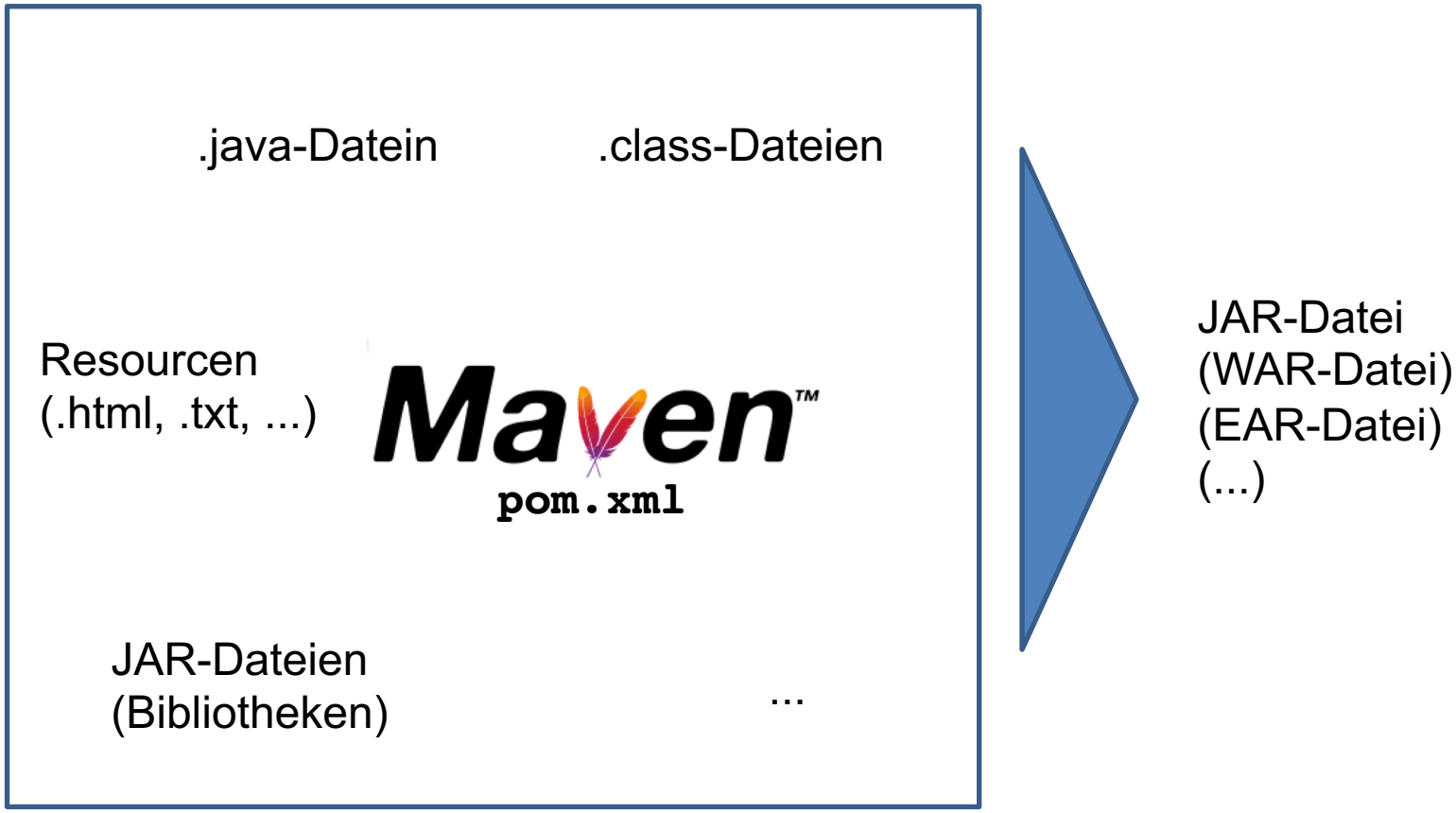
Modul 14
Was ist Maven?

Build Management System für Java



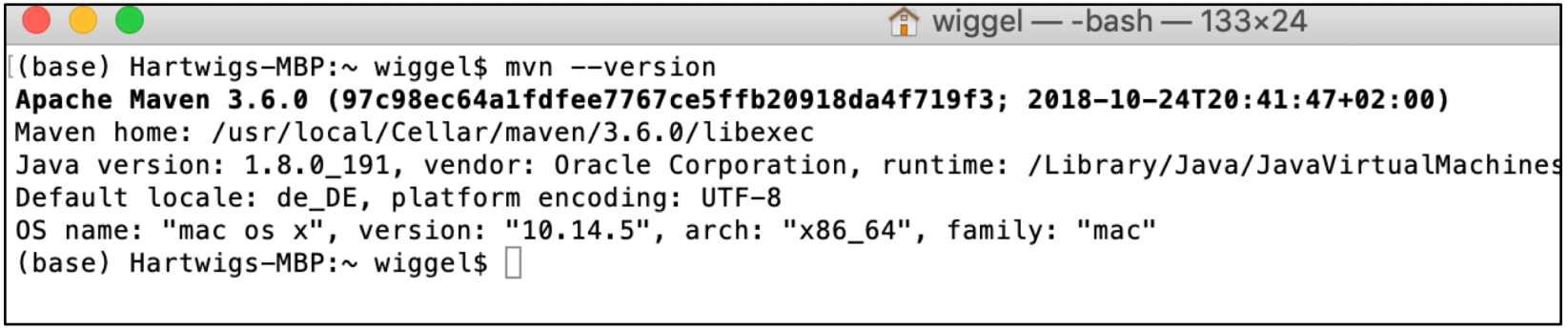
Modul 14
Was ist Maven?

Build Management System für Java



Modul 14
Maven Installation

```
unzip apache-maven-3.6.1-bin.zip
```

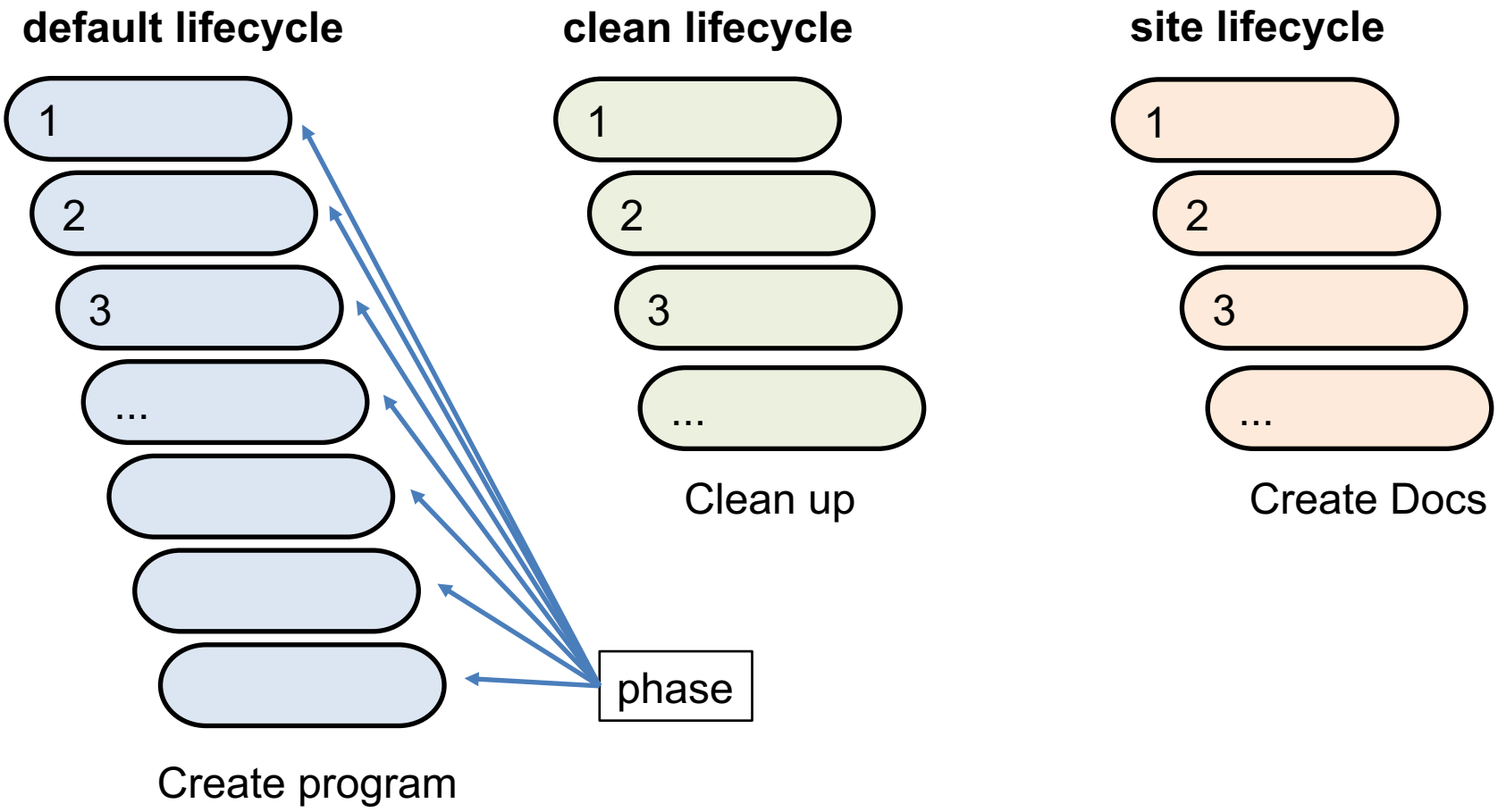


Command-Line Tool

```
C:\> mvn --version
```

Modul 14
Maven core concept

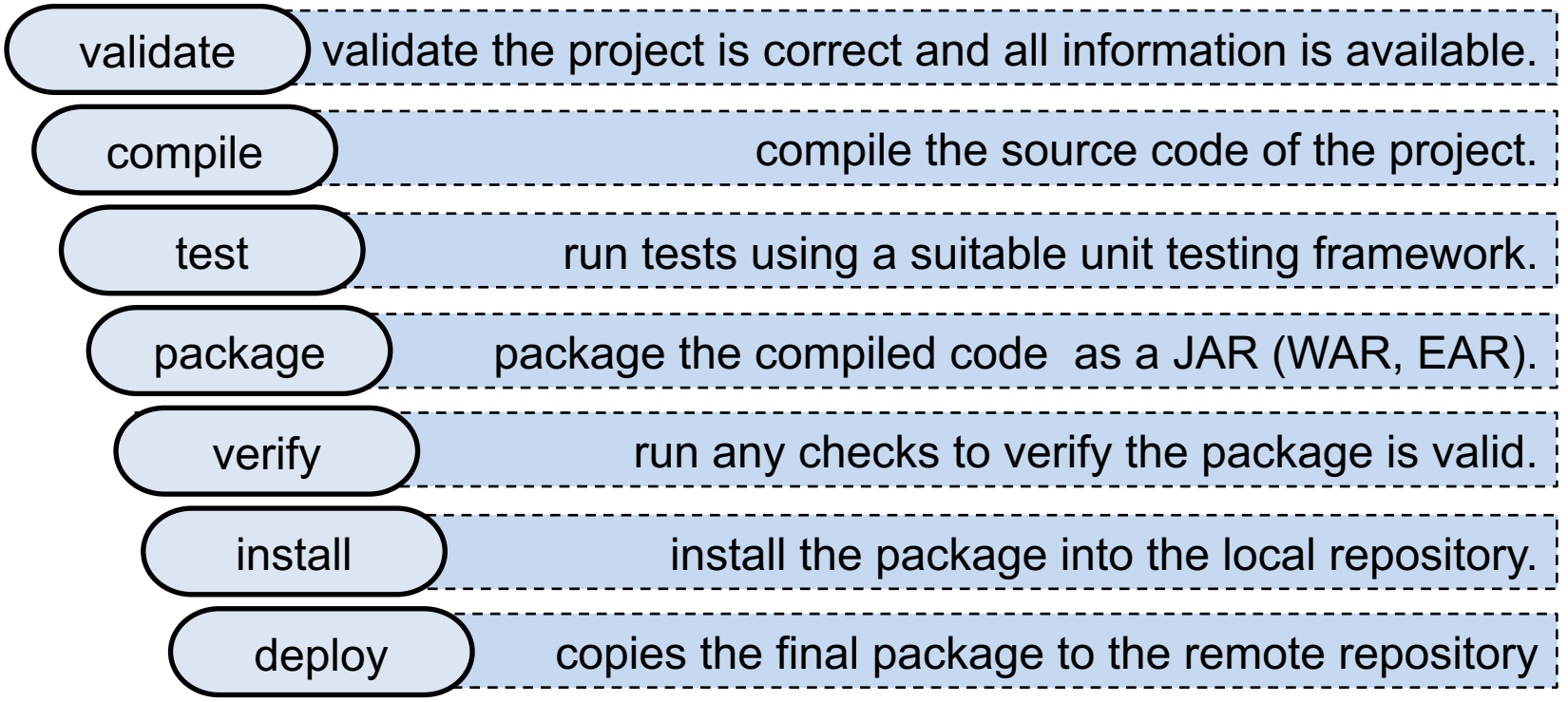
Three build-in lifecycles



Modul 14
Maven default lifecycle

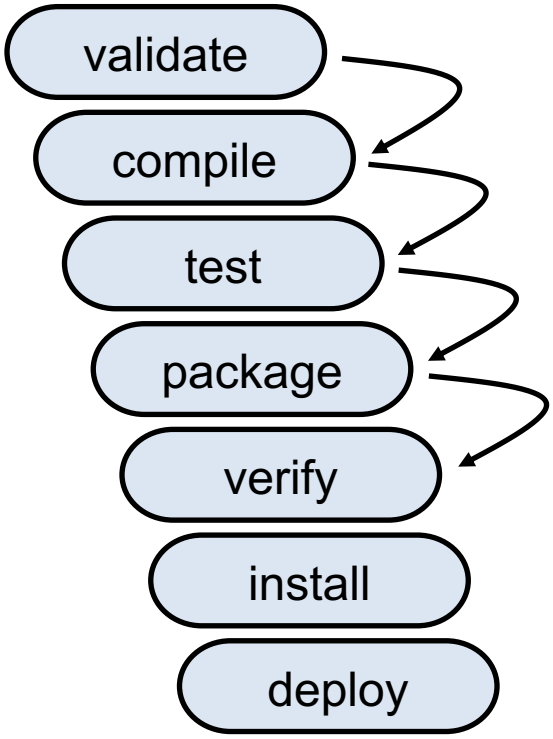
The core lifecycle consists of 7 phases. The full lifecycle consists of 22 phases.

default lifecycle (core)



"Execute" a lifecycle phase

default lifecycle (core)



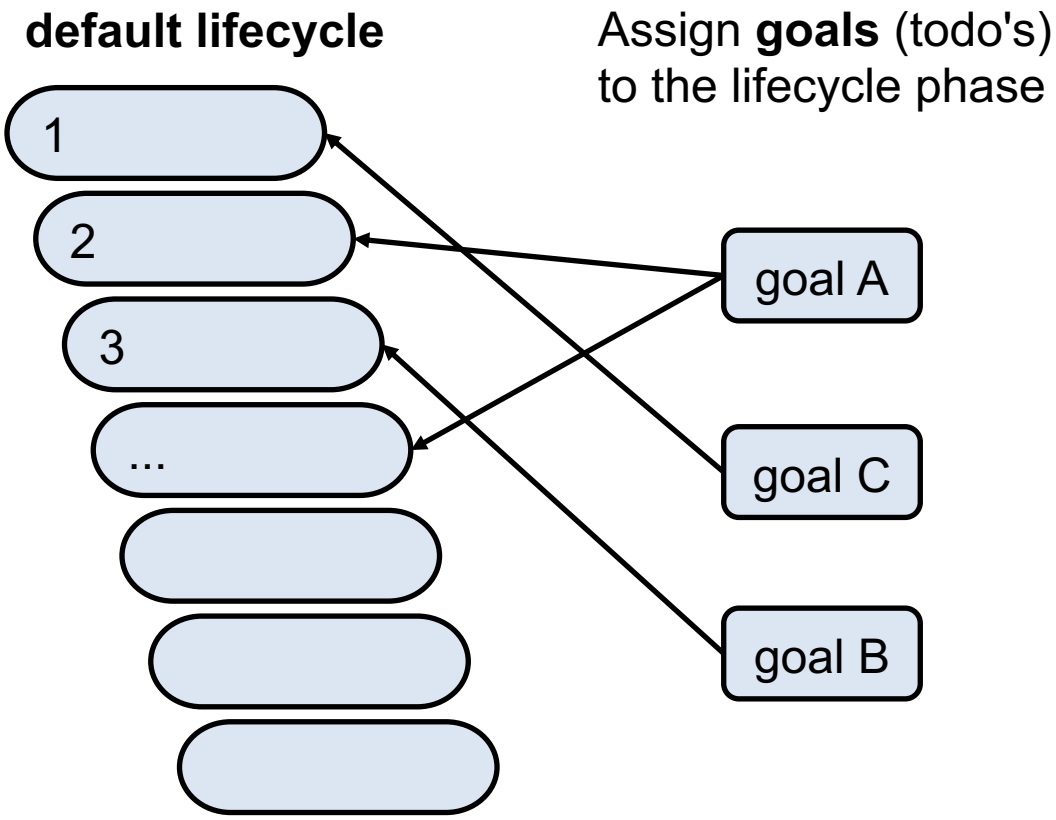
We can advise maven to execute the lifecycle phases up to a certain phase, e.g.:

```
c:\> mvn verify
```

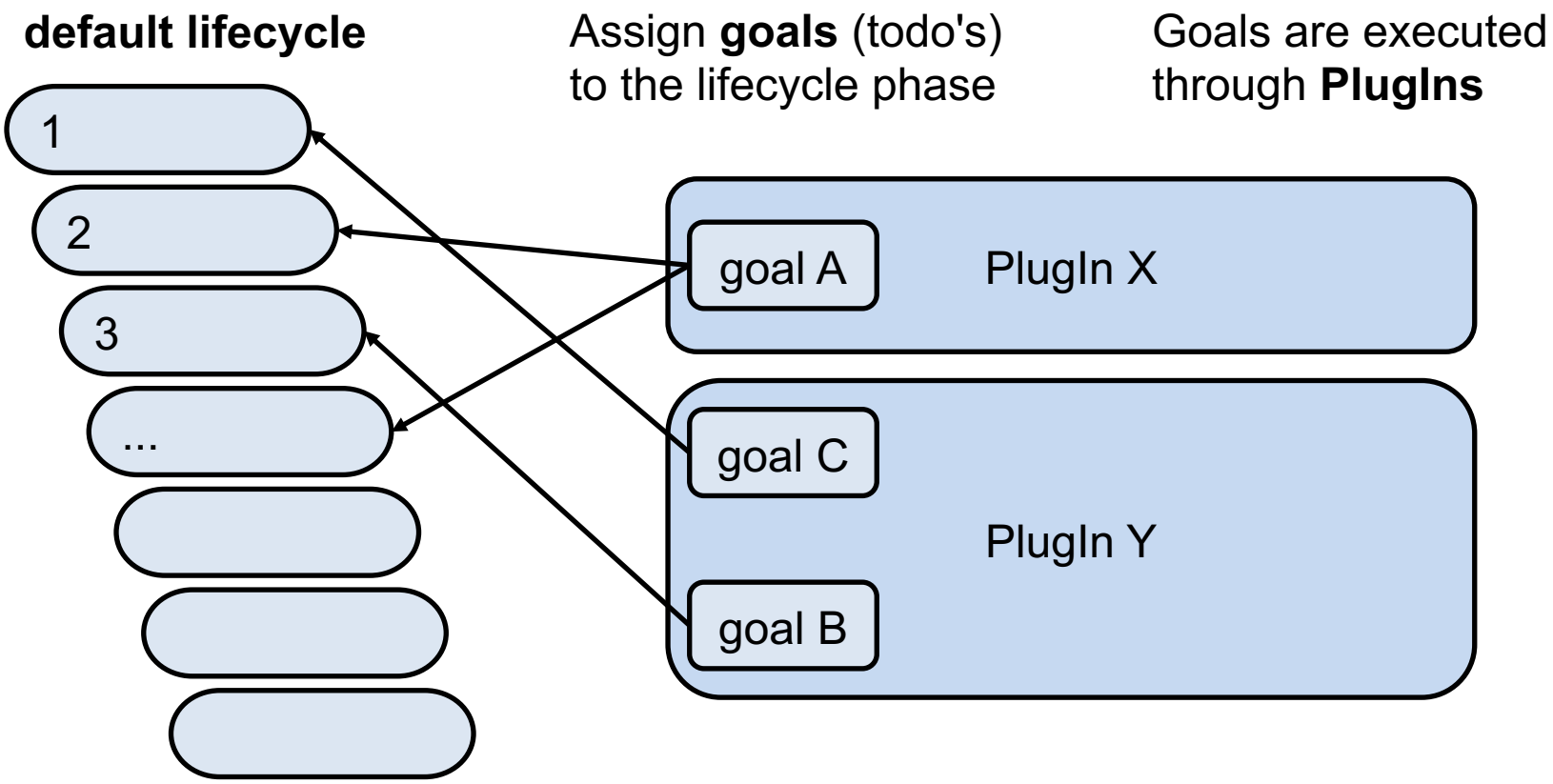
All phases up to the specified phase are executed.

You can't execute a single phase (except the first one).

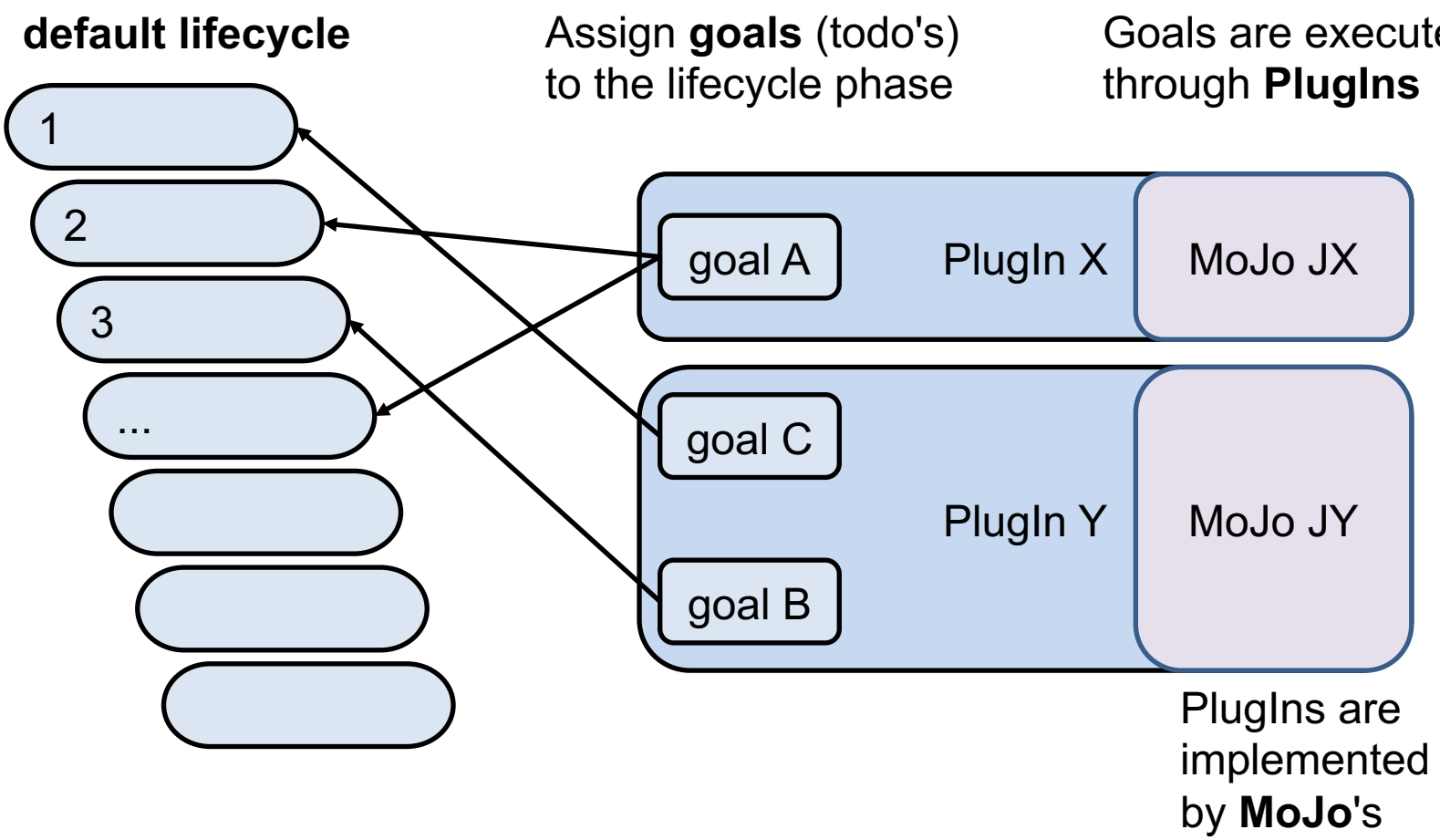
Goals


default lifecycle
Assign **goals** (todo's)
to the lifecycle phase
1
2
3
...
goal A
goal C
goal B

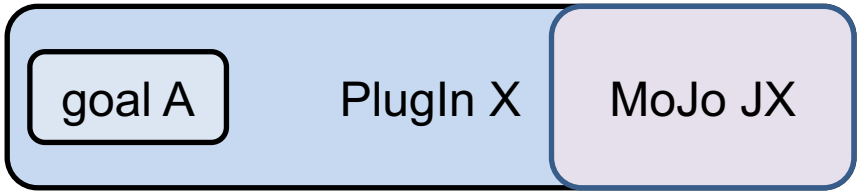
Goals


default lifecycle
Assign goals (todo's) to the lifecycle phase
Goals are executed through PlugIns
1
2
3
...
goal A
PlugIn X
goal C
PlugIn Y
goal B

Goals

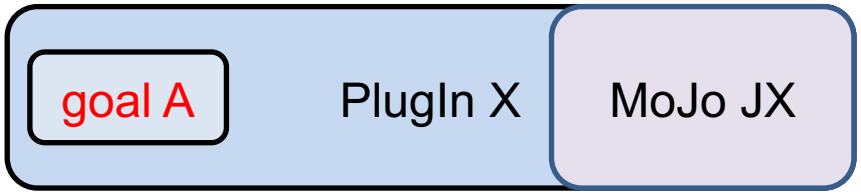

default lifecycle
Assign goals (todo's) to the lifecycle phase
Goals are executed through PlugIns
1
2
3
...
goal A
PlugIn X
MoJo JX
goal C
goal B
PlugIn Y
MoJo JY
PlugIns are implemented by MoJo's

Modul 14
Maven PlugIns



Each Mojo must specify at least one **goal, but it can be more.**

```
@Mojo( name = "goalA")
public class MojoJX extends AbstractMojo
{
  public void execute() throws MojoExecutionException
  {
    getLog().info( "Hello, world." );
  }
}
```



Each Mojo must specify at least one **goal, but it can be more.**

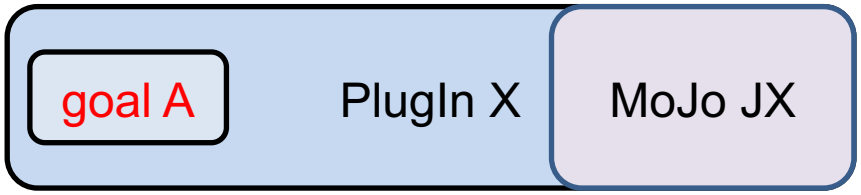
You never call the MoJo directly, you call the **PlugIn** with its **goal**.

A PlugIn is identified by **groupId:artifactId:version**

```
<plugins>
  <plugin>
   <groupId>sample.plugin</groupId>
     <artifactId>pluginx-maven-plugin</artifactId>
     <version>1.0-SNAPSHOT</version>
  </plugin>
</plugins>
```

```
C:\> mvn sample.plugin:pluginx-maven-plugin:
         1.0-SNAPSHOT:goalA
```

Modul 14
Maven PlugIns



Each Mojo must specify at least one **goal, but it can be more.**

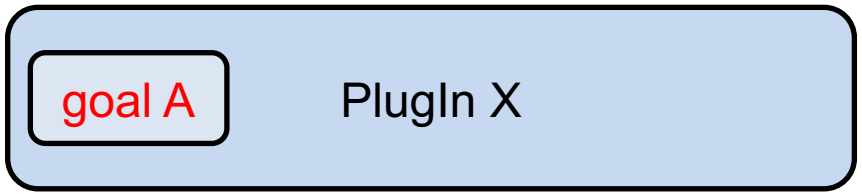
You never call the MoJo directly, you call the **PlugIn** with its **goal**.

A PlugIn is identified by **groupId:artifactId:version**

```
<plugins>
  <plugin>
   <groupId>sample.plugin</groupId>
     <artifactId>pluginx-maven-plugin</artifactId>
     <version>1.0-SNAPSHOT</version>
  </plugin>
</plugins>
```

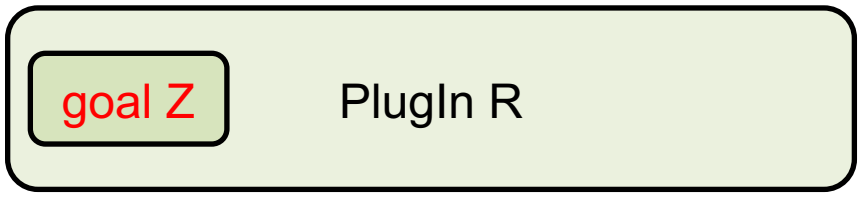
```
C:\> mvn sample.plugin:pluginx-maven-plugin:
         1.0-SNAPSHOT:goalA
```

Build plugins



Build plugins will be executed during the default-lifecycle and they should be configured in the `<build/>` element.

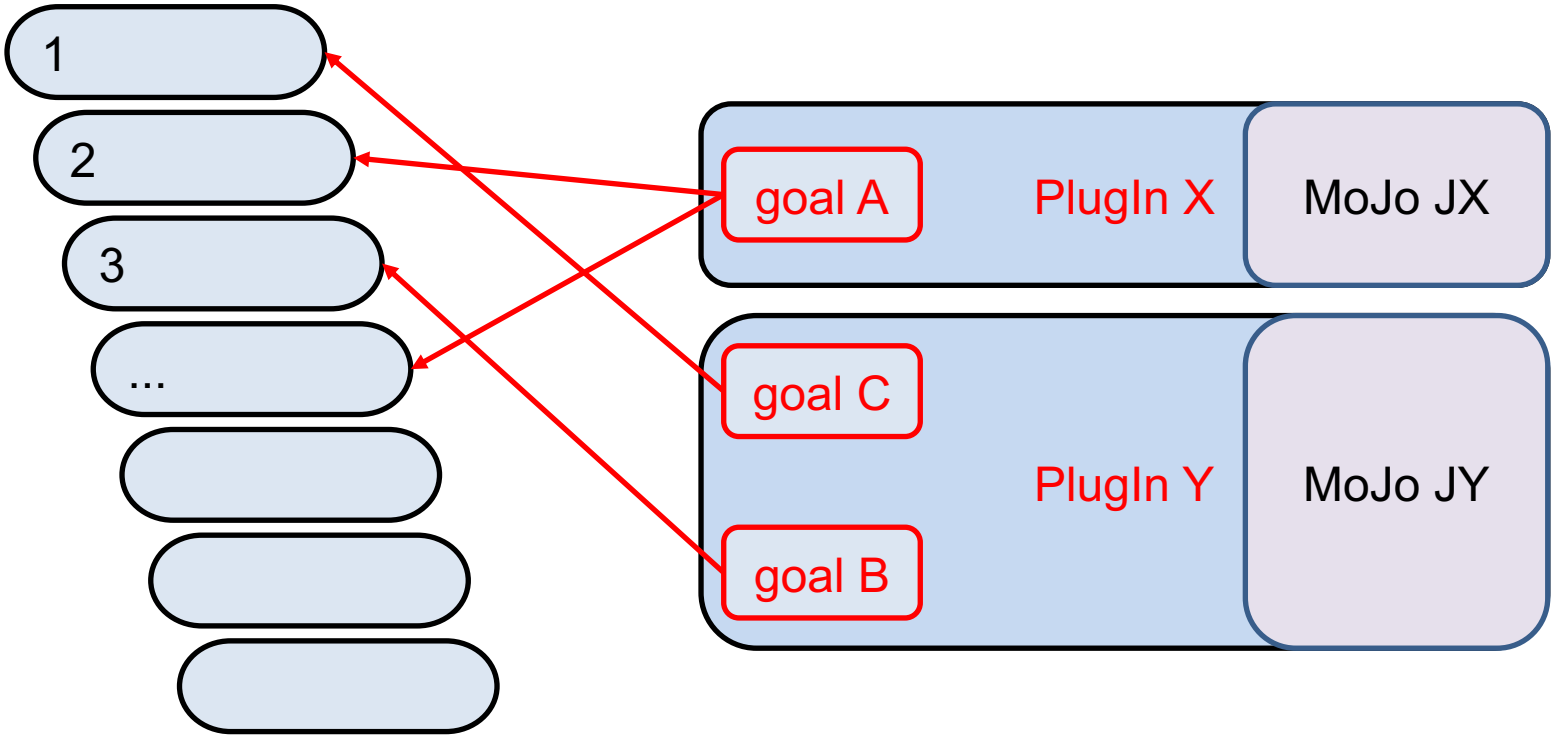
Reporting plugins



Reporting plugins will be executed during the site lifecycle and they should be configured in the `<reporting/>` element.

All plugins should have minimal required information:
groupId, **a**rtifactId and **v**ersion (**GAV** == Maven Coordinates)

The **`pom.xml`** file is the core of a project's configuration.



Modul 14
Maven - pom.xml

The **`pom.xml`** file is the core of a project's configuration.

It defines:

- Which **PlugIns** are used

- Which **PlugIn** with which **goal** is assigned to which **phase** of the lifecycle

- Which **Dependancies** (Java-Libraries) are needed for the build (later)

- ... reporting, etc.

The **`pom.xml`** file is the core of a project's configuration.

It defines:

- Which **PlugIns** are used

- Which **PlugIn** with which **goal** is assigned to which **phase** of the lifecycle

- Which **dependencys** (Java-Libraries) are needed for the build (later)

- ... reporting, etc.

Modul 14
Example of pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns=http://maven.apache.org/POM/4.0.0 ...>

<modelVersion>4.0.0</modelVersion>                    Version of pom-file

<groupId>com.hsoffenburg</groupId>
<artifactId>M14_Empty001</artifactId>                 GAV of users artefact
<version>1.0-SNAPSHOT</version>

 <build>
   <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>build-helper-maven-plugin</artifactId>   PlugIn
      <version>3.0.0</version>                             GAV
    </plugin>
   </plugins>
 </build>

</project>
```

Modul 14
Example of pom.xml

Get the PlugIns goals:

```
mvn help:describe -Dplugin=org.codehaus.mojo:build-
helper-maven-plugin -Ddetail=true
```

Run the goal 'help'

```
mvn org.codehaus.mojo:build-helper-maven-plugin:3.0.0:help
```

Shortcut per convention (prefix resolution for artefact ID):
`maven-${prefix}-plugin` or **`${prefix}-maven-plugin`**
can be used: **`${prefix}`**.
For often used artefacts, the GroupID and Version is preset in settings.xml and can be omitted.

Run the goal 'help'

```
mvn build-helper:help
```

The **`pom.xml`** file is the core of a project's configuration.

It defines:

- Which **PlugIns** are used

- Which **PlugIn** with which **goal** is assigned to which **phase** of the lifecycle

- Which **dependencys** (Java-Libraries) are needed for the build (later)

- ... reporting, etc.

Normally, Build-**PlugIns** (Goals of Build-PlugIns) are assigned to be **executed** during a certain lifecycle **phase**.

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>build-helper-maven-plugin</artifactId>
  <version>3.0.0</version>

    <executions>
      <execution>
        <id>execution1</id>
        <phase>validate</phase>
        <goals>
          <goal>help</goal>
        </goals>
      </execution>
    </executions>

</plugin>
```

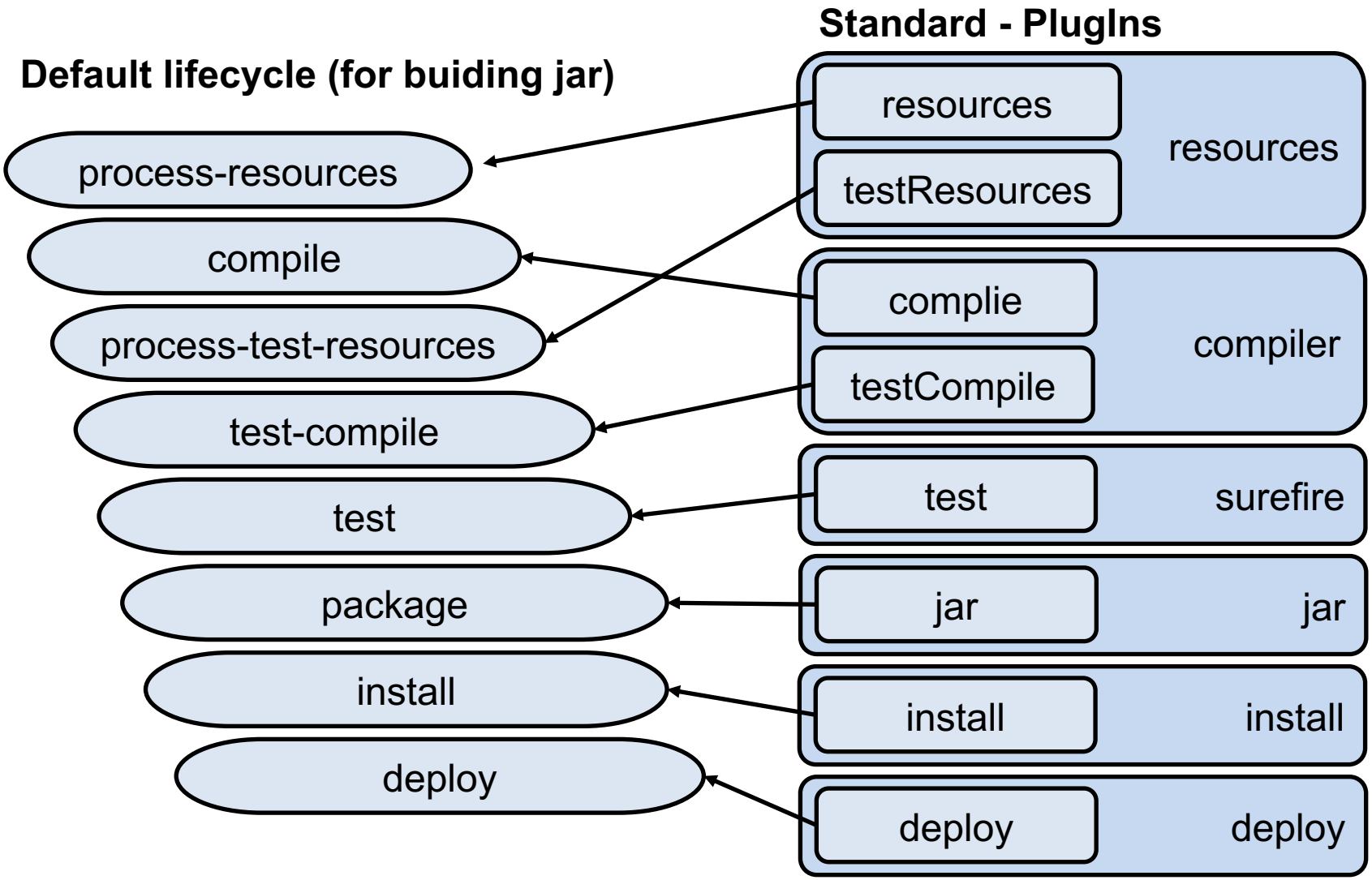
Which PlugIn ...

...in which phase to execute ...

... with which goal

```
C:\> mvn validate
```

Modul 14
Standard PlugIn assignment for building a JAR



Modul 14
Standard PlugIn assignment for building a JAR

Example: Add the resources-goal for phase process-resources

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-resources-plugin</artifactId>
    <version>3.1.0</version>
    <executions>
       <execution>
          <phase>process-resources</phase>
          <goals>
             <goal>resources</goal>
          </goals>
        </execution>
     </executions>

</plugin>
```

Modul 14
Standard PlugIn assignment for building a JAR

Example: Add the resources-goal for phase process-resources

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-resources-plugin</artifactId>
    <version>3.1.0</version>

</plugin>
```

This is default setting of the PlugIn

Modul 14
Standard PlugIn assignment for building a JAR

Example: Add the compile-goal for phase complie

```
<plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-compiler-plugin</artifactId>
 <version>3.8.0</version>
   <configuration>
    <source>1.8</source>
    <target>1.8</target>
   </configuration>
   <executions>
     <execution>
     <phase>compile</phase>
     <goals>
        <goal>compile</goal>
     </goals>
    </execution>
   </executions>
</plugin>
```

Modul 14
Standard PlugIn assignment for building a JAR

Example: Add the compile-goal for phase complie

```
<plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-compiler-plugin</artifactId>
 <version>3.8.0</version>
   <configuration>
    <source>1.8</source>
    <target>1.8</target>
   </configuration>
</plugin>
```

This is default setting of the PlugIn

Modul 14
Standard PlugIn assignment for building a JAR

```
<build>
  <plugins>
     <plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-resources-plugin</artifactId>
        <version>3.1.0</version>
     </plugin>

     <plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-compiler-plugin</artifactId>
       <version>3.8.0</version>
       <configuration>
          <source>1.8</source>
          <target>1.8</target>
       </configuration>
     </plugin>

     ...

  <plugins>
<build>
```

Modul 14
Standard PlugIn assignment for building a JAR

All 8 goals becomes **default** setting if you use the tag

```
<packaging>jar</packaging>
```

Modul 14
Standard PlugIn assignment for building a JAR

Complete pom.xml for JAR

```
<?xml version="1.0" encoding="UTF-8"?>
<project ...>

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.hsoffenburg</groupId>
    <artifactId>M14_Empty001</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>          //Default
    <build>
        <plugins>

        </plugins>
    </build>
</project>
```

Modul 14
Standard PlugIn assignment for building a JAR

Complete pom.xml for JAR

```
<?xml version="1.0" encoding="UTF-8"?>
<project ...>

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.hsoffenburg</groupId>
    <artifactId>M14_Empty001</artifactId>
    <version>1.0-SNAPSHOT</version>

</project>
```

Use *properties* for Plugin configuration

```
<properties>
  <maven.compiler.source>1.8</maven.compiler.source>
</properties>

 ... ${maven.compiler.source} ...  //access property
```

Modul 14
Standard PlugIn assignment for building a JAR

But …

```
wiggel$ java -jar ./target/M14_Empty001-1.0-SNAPSHOT.jar

Fehler: Hauptklasse jar konnte nicht gefunden oder
geladen werden
```

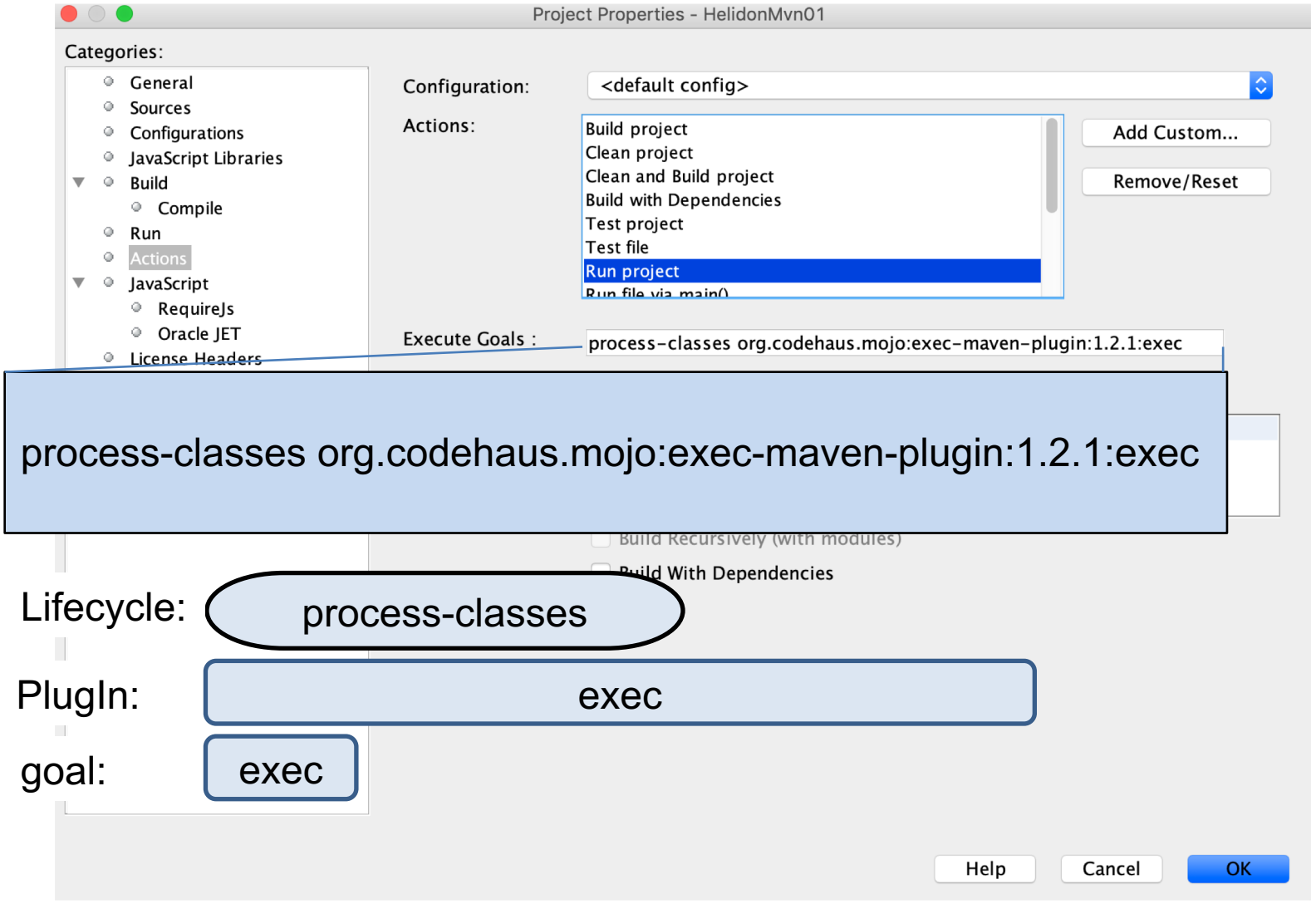
…, so add Main-Class in Manifest-Tag of jar-PlugIn:

```
 <plugin>
    <artifactId>maven-jar-plugin</artifactId>
    <version>3.1.2</version>
    <configuration>
      <archive>
        <manifest>
          <mainClass>hsog.m14_empty001.HalloW</mainClass>
        </manifest>
      </archive>
    </configuration>
</plugin>
```

```
C:\> java -jar ./target/M14_Empty001-1.0-SNAPSHOT.jar
```

Maven – IDE Integration

Maven – IDE Integration



process-classes org.codehaus.mojo:exec-maven-plugin:1.2.1:exec

Lifecycle: process-classes

PlugIn: exec

goal: exec

Maven – Simulate IDE integration by hand (CLI)

Start via exec-maven-plugin

```
 <plugin>
   <groupId>org.codehaus.mojo</groupId>
   <artifactId>exec-maven-plugin</artifactId>
   <version>1.6.0</version>
      <executions>
        <execution>
           <goals>
             <goal>java</goal>
           </goals>
        </execution>
      </executions>
     <configuration>
       <mainClass>m14_empty001.HalloW</mainClass>
     </configuration>
</plugin>
```

```
c:\> mvn exec:java
```

Maven
Dependencies

Maven Dependencies

With *ant* download all required libraries by hand

```
<dependency>
<groupId>io.helidon.microprofile.bundles</groupId>
<artifactId>helidon-microprofile-1.2</artifactId>
</dependency>
```

- io.helidon.health » helidon-health-checks
- io.helidon.microprofile » helidon-microprofile-fault-tolerance
- io.helidon.microprofile.bundles » helidon-microprofile-1.1
- io.helidon.microprofile.health » helidon-microprofile-health
- io.helidon.microprofile.jwt » helidon-microprofile-jwt-auth-cdi
- io.helidon.microprofile.metrics » helidon-microprofile-metrics

Transitive Dependencies

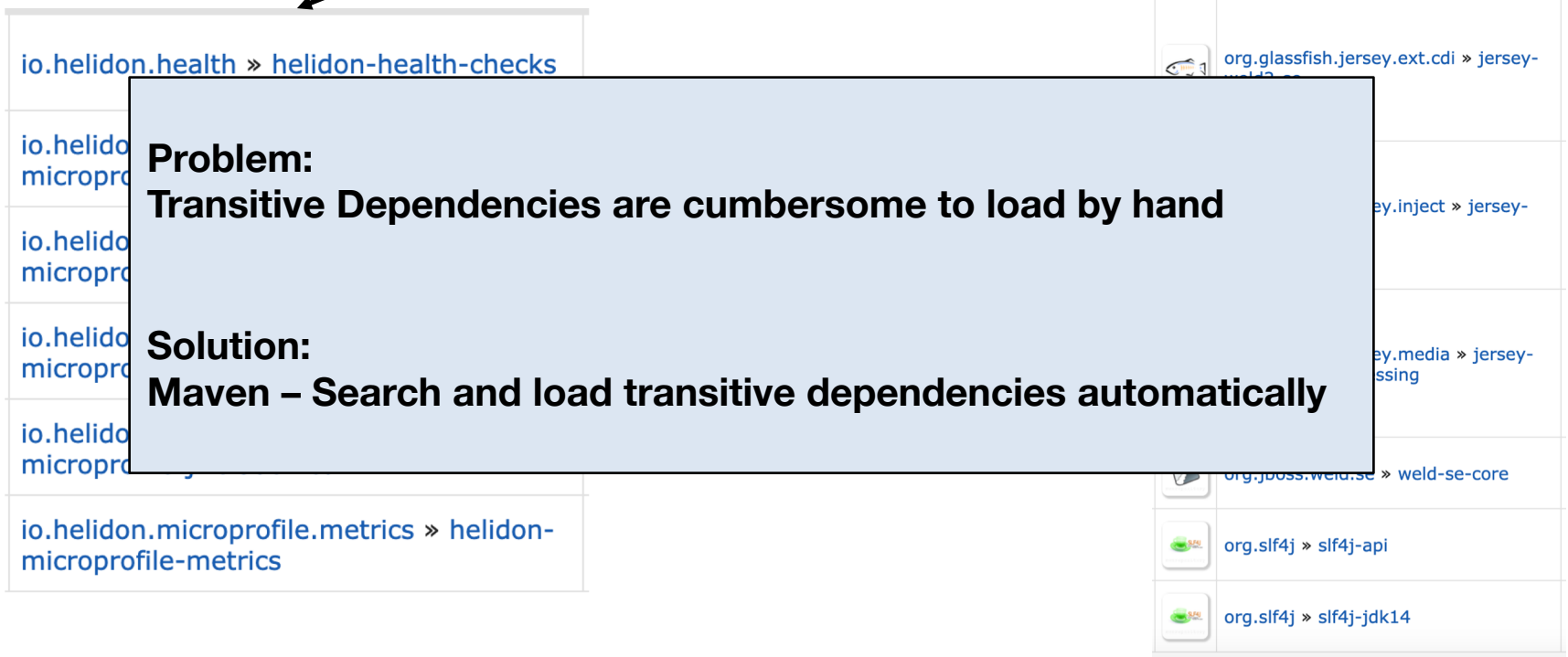
- io.helidon.microprofile.config » helidon-microprofile-config
- io.helidon.microprofile.config » helidon-microprofile-config-cdi
- io.helidon.microprofile.server » helidon-microprofile-server

- org.glassfish » javax.json
- org.glassfish.jersey.ext.cdi » jersey-weld2-se
- org.glassfish.jersey.inject » jersey-hk2
- org.glassfish.jersey.media » jersey-media-json-processing
- org.jboss.weld.se » weld-se-core
- org.slf4j » slf4j-api
- org.slf4j » slf4j-jdk14

Maven Dependencies

With *ant* download all required libraries by hand

```
<dependency>
<groupId>io.helidon.microprofile.bundles</groupId>
<artifactId>helidon-microprofile-1.2</artifactId>
</dependency>
```



Problem:
Transitive Dependencies are cumbersome to load by hand

Solution:
Maven – Search and load transitive dependencies automatically

Dependencies
pom.xml

The **`pom.xml`** file is the core of a project's configuration.

It defines:

- Which **PlugIns** are used

- Which **PlugIn** with which **goal** is assigned to which **phase** of the lifecycle

- Which **dependencies** (Java-Libraries) are needed for the build (now)

- ... reporting, etc.

Dependencies
pom.xml

List required libraries in pom.xml.

```
</project>
   ...
   <dependencies>
     <dependency>
        <groupId>io.helidon.microprofile.bundles</groupId>
        <artifactId>helidon-microprofile-1.2</artifactId>
        <version>1.0.2</version>
      </dependency>
    </dependencies>
</project>
```

Maven automatically searches and loads the libs:

Search `helidon-microprofile-1.2` in Maven local repository.

Search `helidon-microprofile-1.2` in Maven central repository.

Search `helidon-microprofile-1.2` in Maven remote repository (if remote repository defined in pom.xml).

Dependencies
Example pom.xml for using helidon 1.2

```
<?xml version="1.0" encoding="UTF-8 …">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.hsoffenburg</groupId>
    <artifactId>HelidonMvn01</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <dependencies>
        <dependency>
         <groupId>io.helidon.microprofile.bundles</groupId>
            <artifactId>helidon-microprofile-1.2</artifactId>
            <version>1.0.2</version>
        </dependency>
    </dependencies>
</project>
```

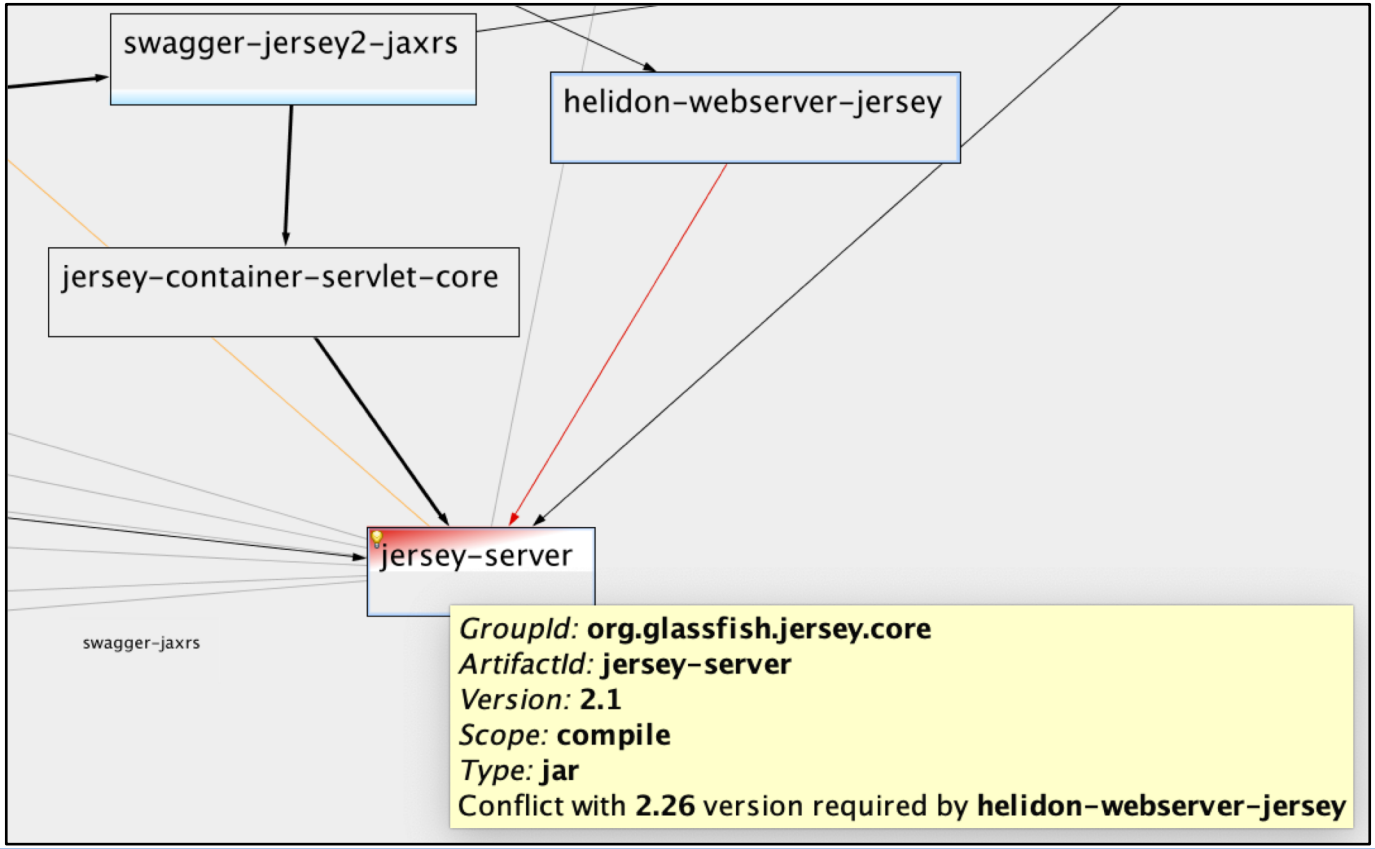
Dependencies
Example pom.xml for using helidon 1.2

Adding further dependencies is easy …

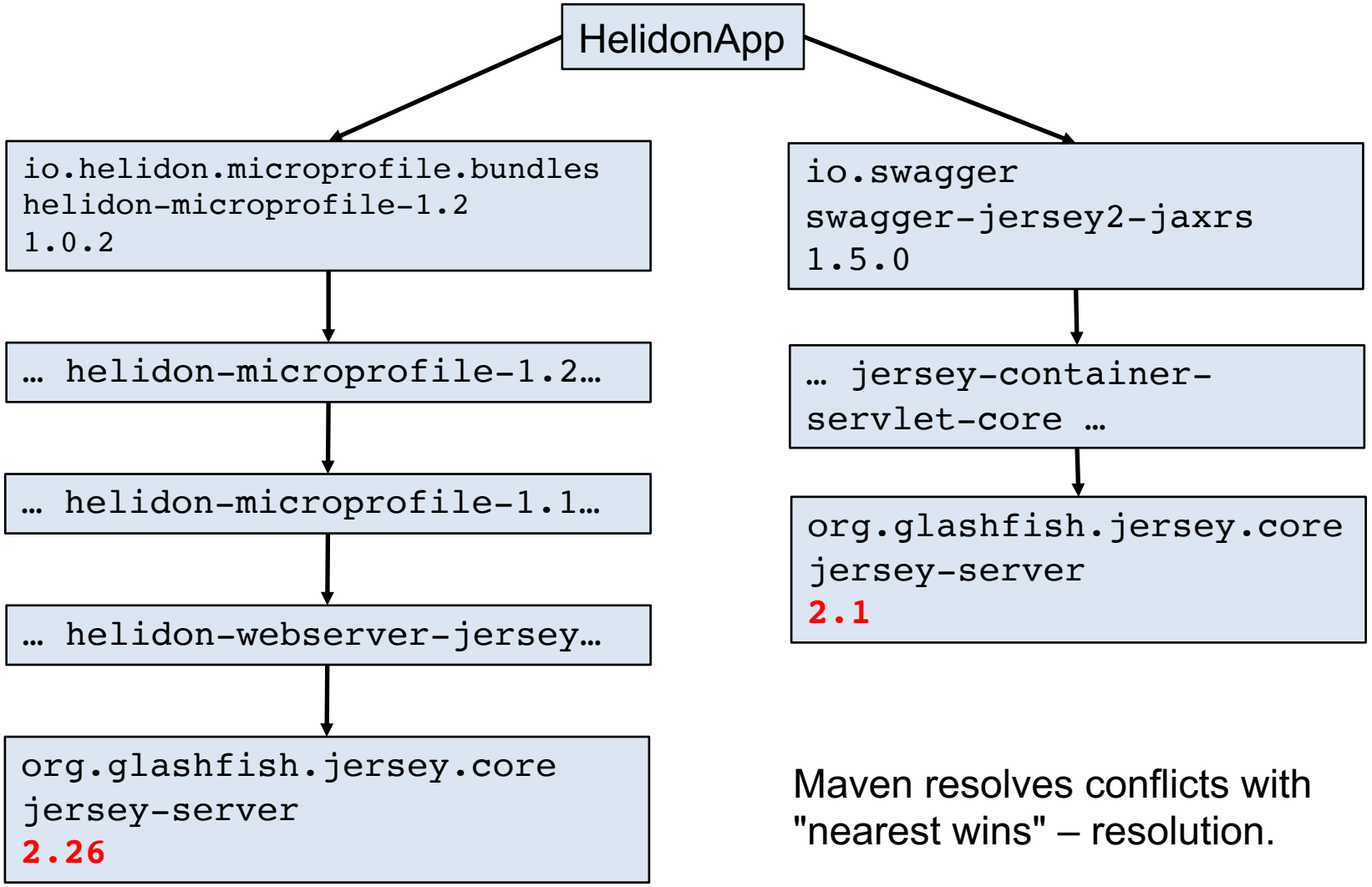
```
<?xml version="1.0" encoding="UTF-8 …">
    ...
    <dependencies>
        <dependency>
         <groupId>io.helidon.microprofile.bundles</groupId>
            <artifactId>helidon-microprofile-1.2</artifactId>
            <version>1.0.2</version>
        </dependency>

         <dependency>
            <groupId>io.swagger</groupId>
            <artifactId>swagger-jersey2-jaxrs</artifactId>
            <version>1.5.0</version>
        </dependency>

    </dependencies>
</project>
```

Dependencies
Example pom.xml for using helidon 1.2

Adding further dependencies is easy … but in this example, we get an (runtime) error! Using the dependency graph shows a version conflict between to artefacts:

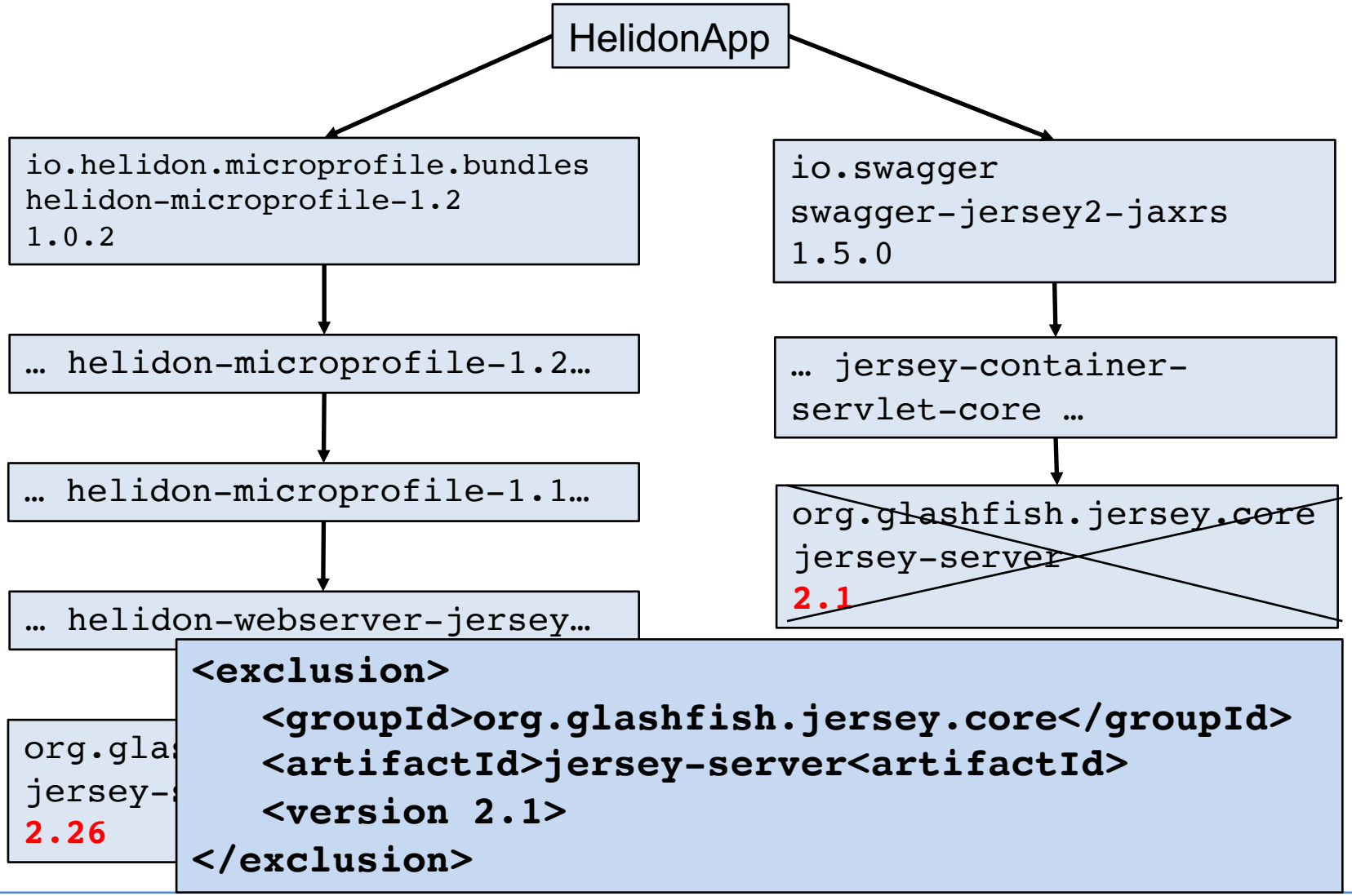


Dependencies
Version conflicts

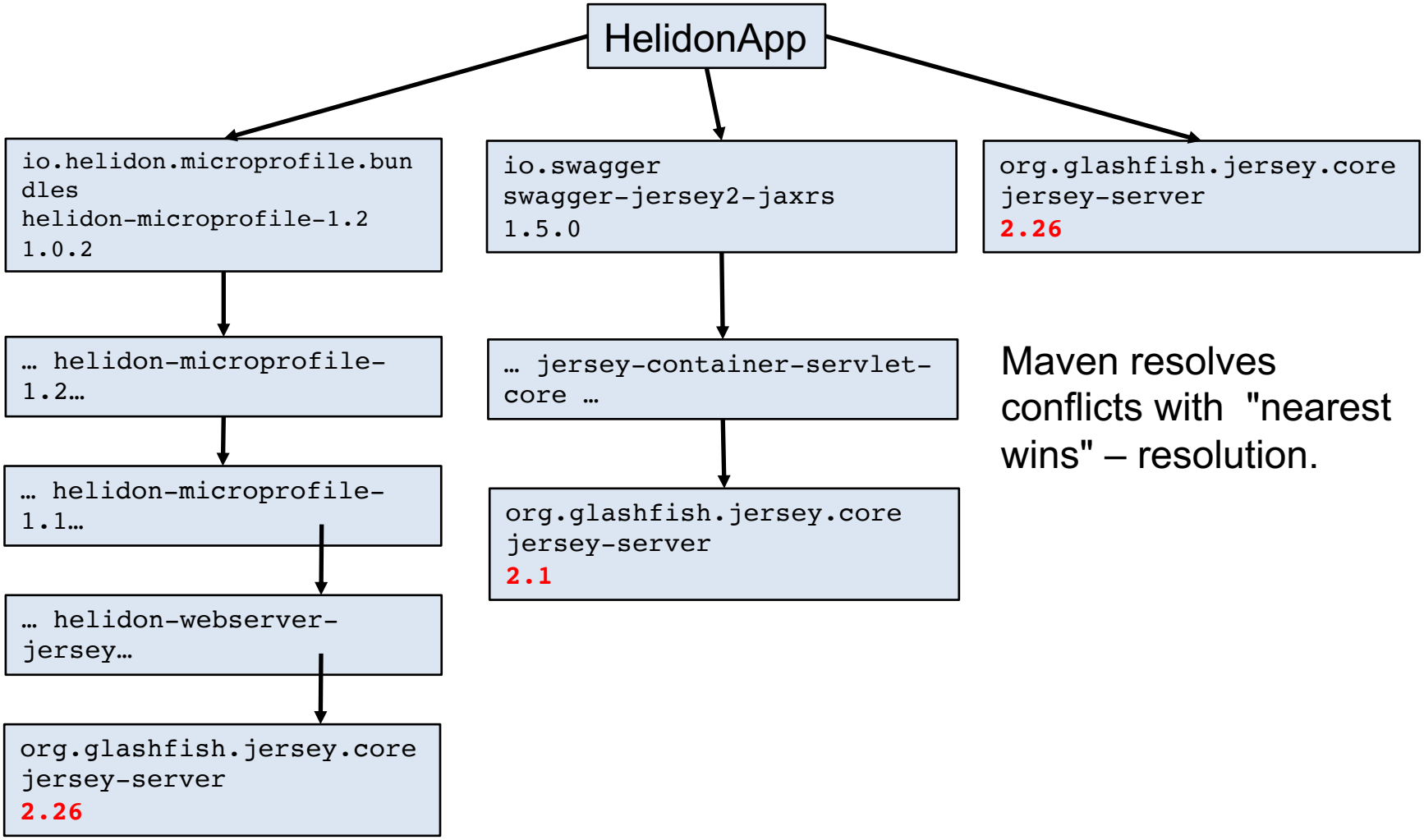


Maven resolves conflicts with "nearest wins" – resolution.

Dependencies
Solution 1 – Exclude Artefact

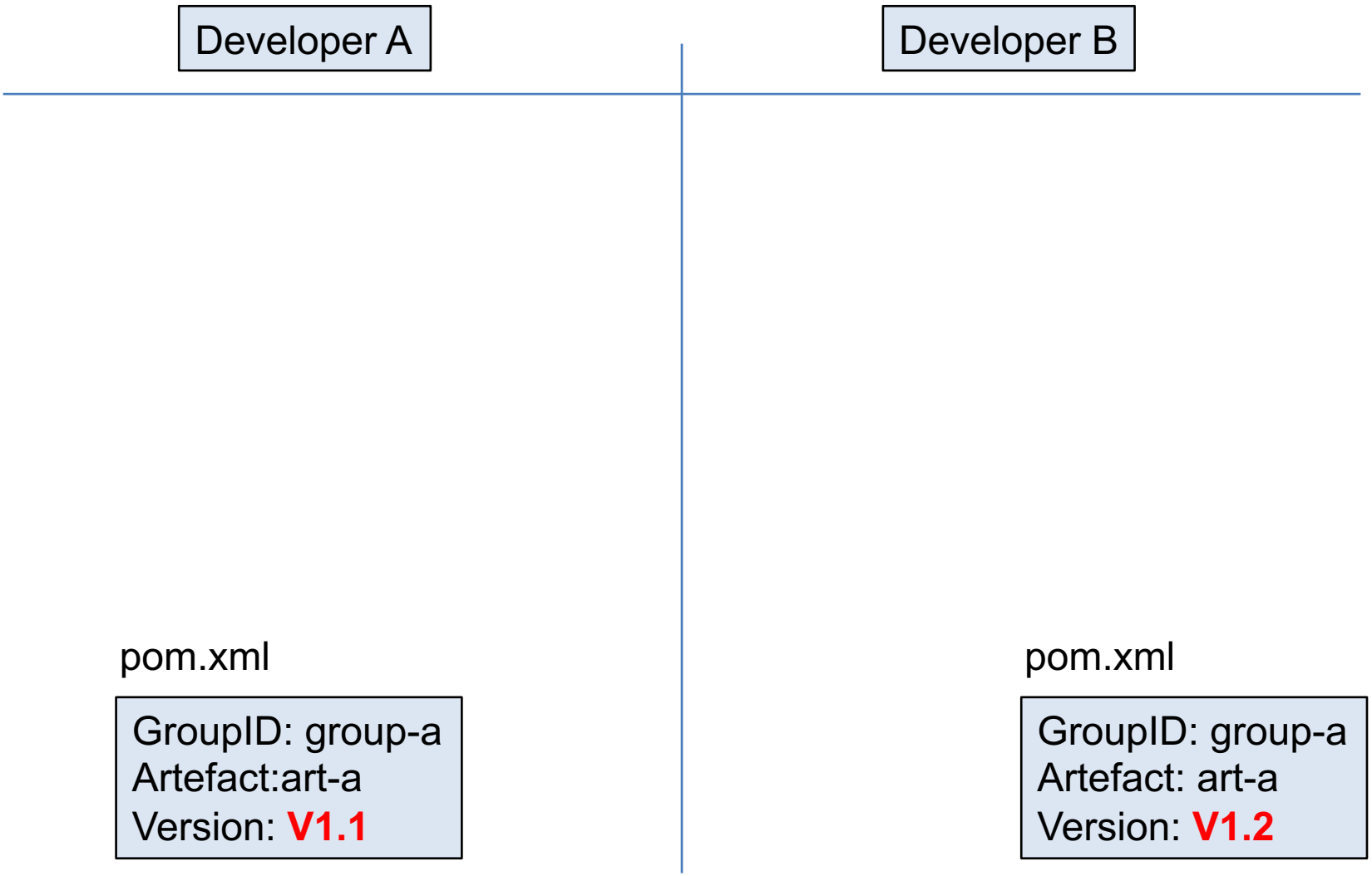


Dependencies
Solution 2 – Include as a direct dependency

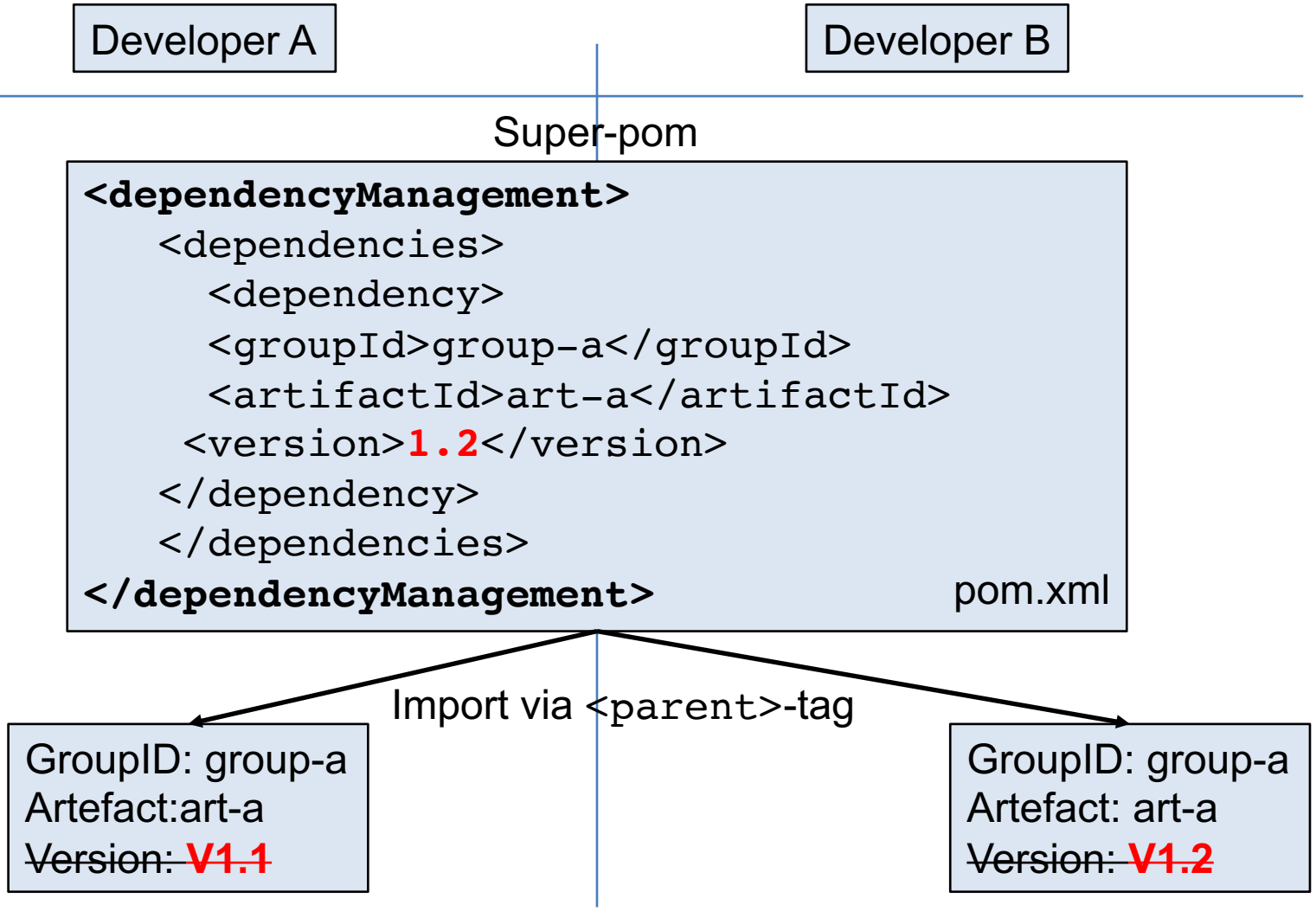


Maven resolves conflicts with "nearest wins" – resolution.

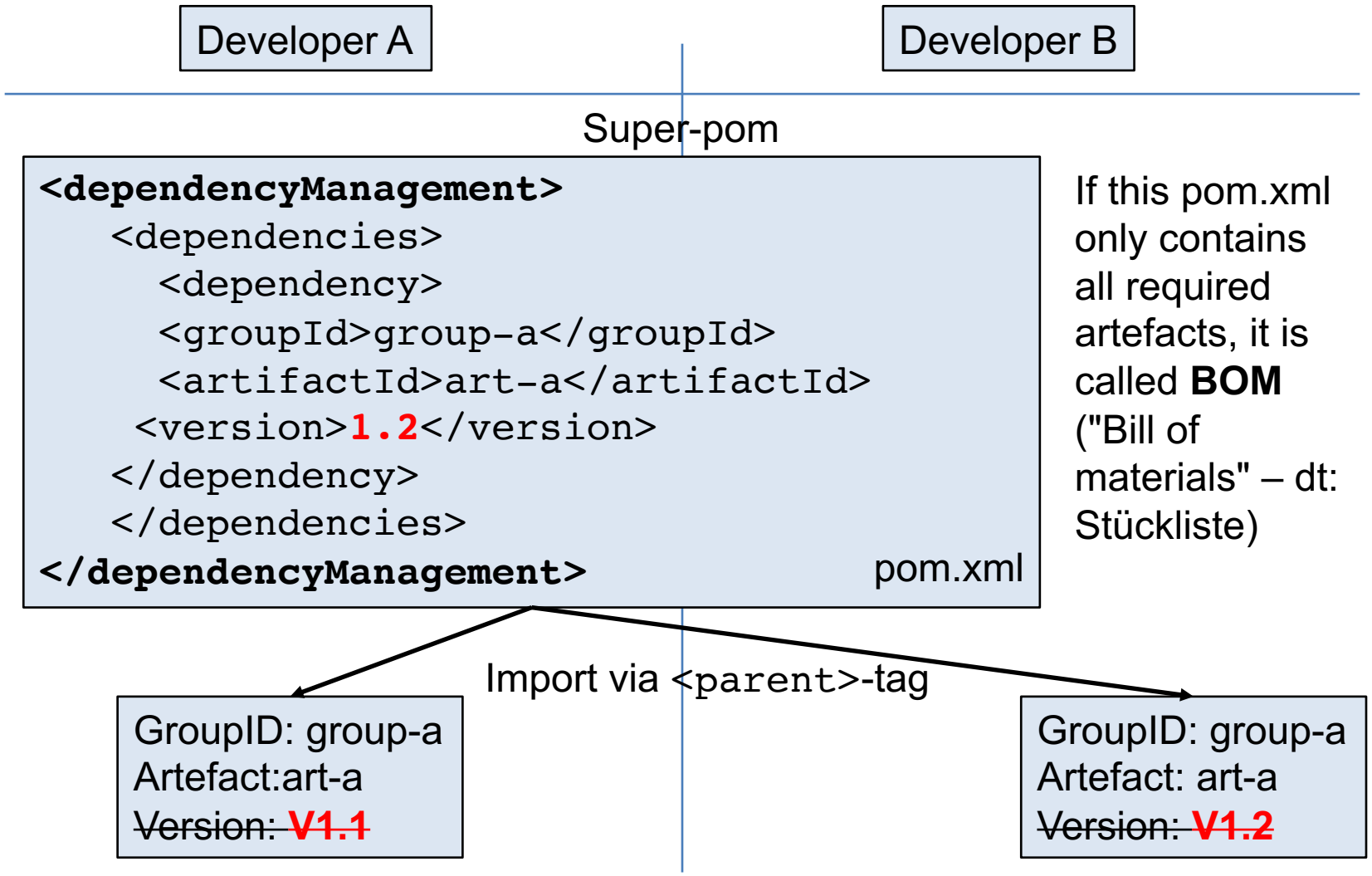
Avoiding Version Conflicts
Dependency Management



Avoiding Version Conflicts
Dependency Management



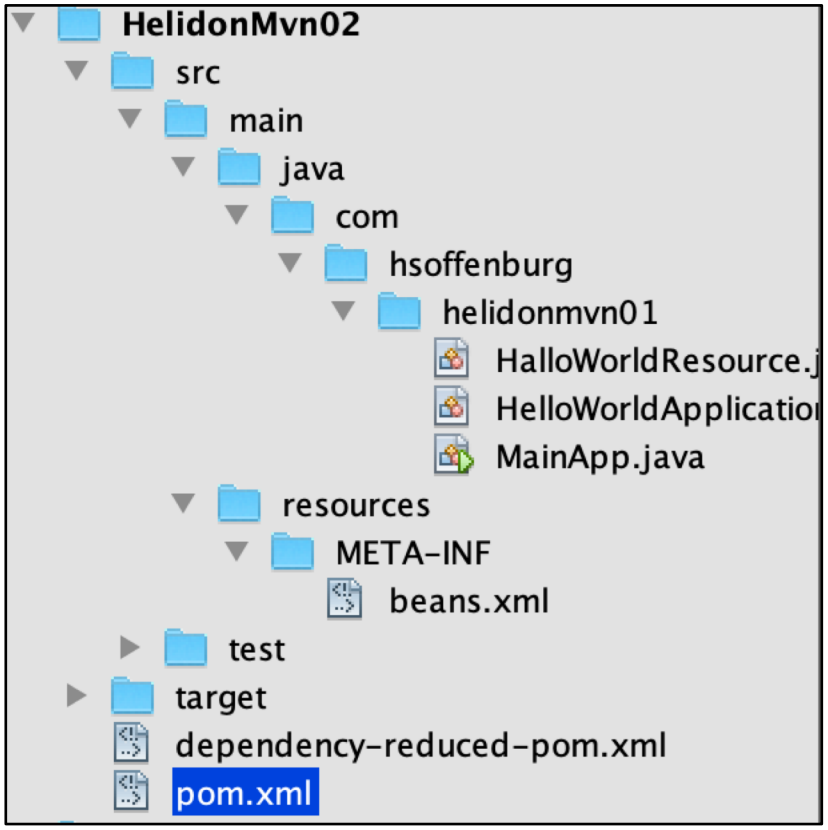
Avoiding Version Conflicts
BOM-file



Maven Archetype

Maven Archetype

For developing java-projects, we need a certain structure of folders and a couple of files.



For generating this folder structure, we can use a pre-defined template: a *maven archetype*.

This template can be customized (i.e. groupID, artefactID, etc.)

To generate the structure, use the PlugIn *archetype* with the goal *generate*:

```
mvn archetype:generate …
```

Maven Archetype for simple JAR

```
archetype:generate
    -DgroupId={project-packaging}
    -DartifactId={project-name}
    -DarchetypeArtifactId={maven-template}
    -DinteractiveMode=false
```

```
mvn archetype:generate
      -DgroupId=com.mkyong.hashing
      -DartifactId=java-project
      -DarchetypeArtifactId=maven-archetype-quickstart
      -DinteractiveMode=false
```

Appendix

List Table of PlugIns, ExecutionIDs and Phases

```
<plugin>
        <groupId>fr.jcgay.maven.plugins</groupId>
        <artifactId>buildplan-maven-plugin</artifactId>
        <version>1.3</version>
</plugin>
```

```
c:\> mvn buildplan:list-phase
```

Appendix

Disable default phase of default goal

```
<plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
        <executions>
           <execution>
            <id>default-compile</id>
            <phase>none</phase>
           </execution>
        </executions>
</plugin>
```

Appendix

Execute main() in jar

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>exec-maven-plugin</artifactId>
  <version>1.6.0</version>
     <executions>
        <execution>
         <goals>
           <goal>java</goal>
          </goals>
         </execution>
       </executions>
       <configuration>
         <mainClass>hsog.m14_empty001.HalloW</mainClass>
       </configuration>
 </plugin>
```

Appendix
Find / check dependancies with enforcer:enforce

```
<plugin> <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-enforcer-plugin</artifactId>
    <version>1.4.1</version>
    <executions>
      <execution>
        <id>default-cli</id>
        <phase>dont-execute</phase>
        <configuration>
          <rules>
            <dependencyConvergence/>
          </rules>
        </configuration>
        <goals>
          <goal>enforce</goal>
        </goals>
      </execution>
    </executions>
</plugin>
```

Appendix
Speed up Weld with jandex

```
<plugin>
    <groupId>org.jboss.jandex</groupId>
    <artifactId>jandex-maven-plugin</artifactId>
    <version>1.0.5</version>
    <executions>
        <execution>
            <id>make-index</id>
            <goals>
                <goal>jandex</goal>
            </goals>
            <phase>process-classes</phase>
        </execution>
    </executions>
</plugin>
```

Appendix
Generate executable fat-jar with capsulate:build

```
<plugin>
    <groupId>com.github.chrisdchristo</groupId>
    <artifactId>capsule-maven-plugin</artifactId>
    <executions>
        <execution>
            <goals>
                <goal>build</goal>
            </goals>
            <configuration>
              <appClass>helidonmvn01.MainApp</appClass>
                <type>thin</type>
            </configuration>
        </execution>
    </executions>
</plugin>
```