


# Apache Maven 4

## … is awesome

Chandra Guntur

May. 2024

Maven™



@CGuntur

# What is Apache Maven?

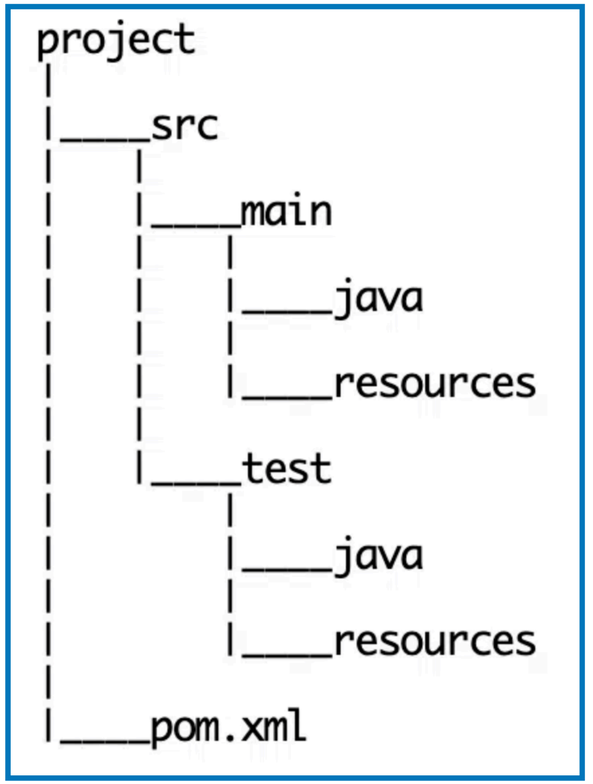
- a powerful build management tool
- primarily used to build Java projects
- developed in Java
- an open source project
- follows convention-over-configuration
- allows for other language projects to be built

# What is “build management”?

- compiling - compile code
- packaging executables - create a .jar, .war, .zip etc.
- testing - unit.functional/integration testing can be orchestrated
- generating documentation - javadoc/site generation capabilities
- generating metrics/reports - build metrics, test result reports
- deploying - push to repository or to server

# What is “convention-over-configuration”?

- a software design paradigm
- decreases the number of decisions needed from developer
- satisfies the principle of least astonishment
- provides sensible default “assumptions”
- promotes common structures and flow in a project
- an example: directory structure
- not just limited to directory structure
- instruction set for Apache Maven is via a POM



# What is a POM?

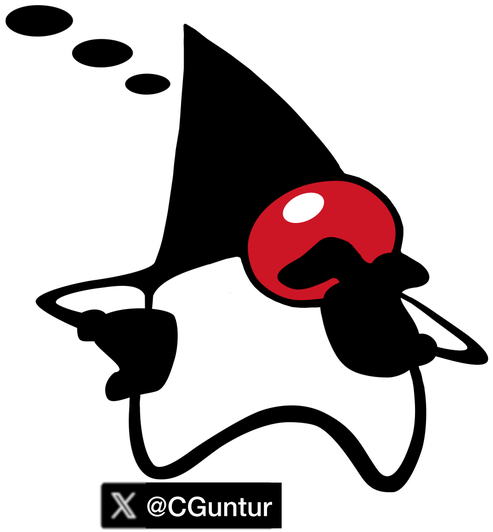
Maven™

# “POM” is …

A Pomeranian?


Pomegranate Juice?

One of the Pom-poms?



# “POM” is …

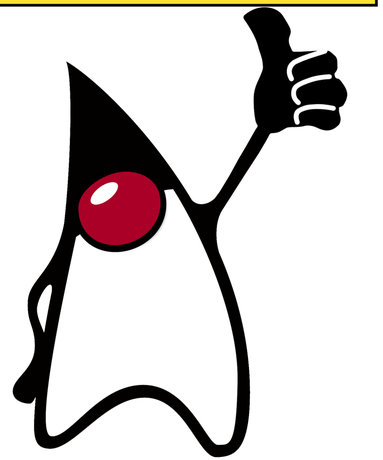
A pomeranian?


Pomegranate Juice?

One of the pom poms?

- POM lists:
  - dependencies
  - plugins
  - properties
  - inheritance details
  - profiles
  - ...

POM is Project Object Model



@CGuntur

# “POM” is …

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

<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>conf</groupId>
    <artifactId>kcdc2023</artifactId>
    <version>1.0.0</version>

    <name>kcdc2023</name>
    <packaging>pom</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>20</maven.compiler.source>
        <maven.compiler.target>20</maven.compiler.target>
    </properties>

    <modules>
        <module>kcdc-java8</module>
        <module>kcdc-java20</module>
    </modules>

    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.11.0</version>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.yaml</groupId>
                <artifactId>snakeyaml</artifactId>
                <version>2.0</version>
            </dependency>
            <dependency>
                <groupId>org.junit.jupiter</groupId>
                <artifactId>junit-jupiter</artifactId>
```

Appendix has POM structure details …

# Benefits/usage of Apache Maven?

# Benefits of using Apache Maven

- visibility - build logs as evidence
- reusability - builds can have composition & hierarchies
- verifiability - builds produce test evidence
- reproducibility - repeatable builds
- maintainability - management of builds is possible
- comprehensibility - ease of understanding the build process

# Apache Maven usage (1)

- create any auto-generated source code, if needed
- generate any documentation from source code
- compile source code, display any errors and warnings
- test the project running existing tests in source code

continued ...

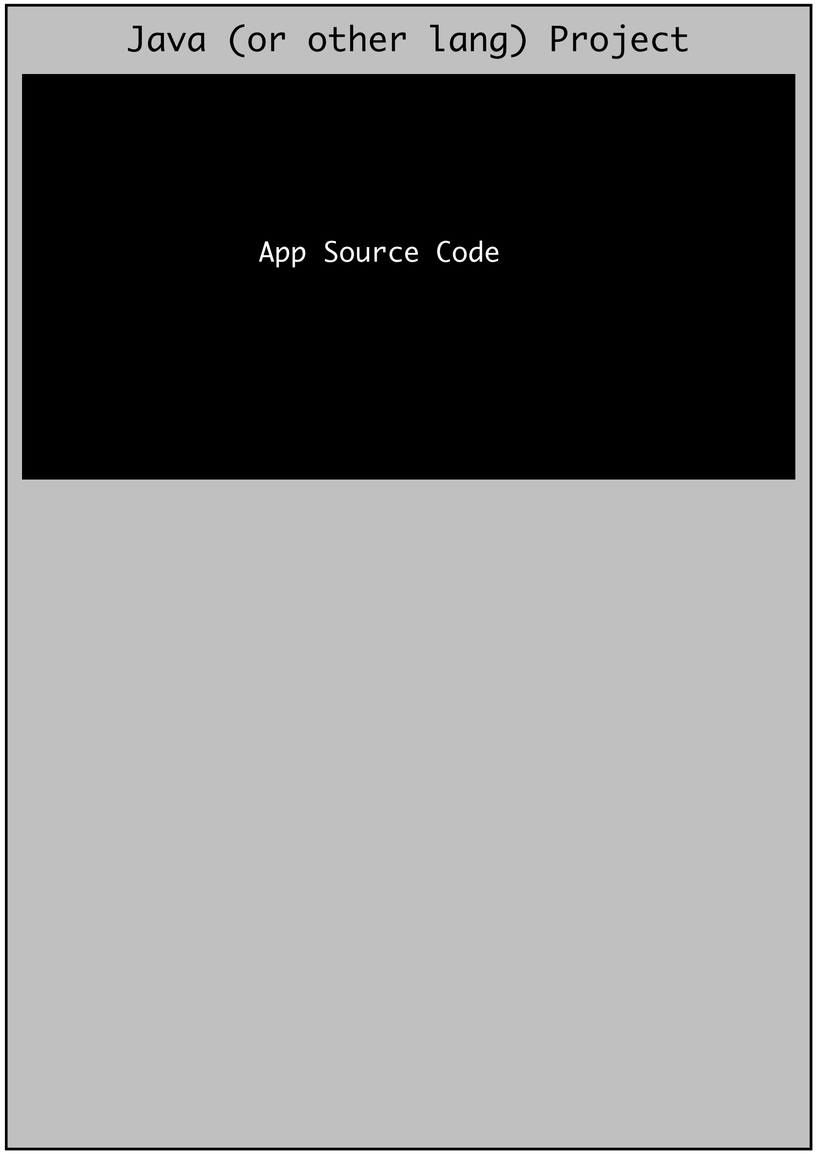
# Apache Maven usage (2)

- package compiled code into a .jar or .war or .zip etc.
- additionally, package source code into a .jar or .zip
- install the packaged code into a repository/server
- generate site reports and test evidence
- report a build as success or failure

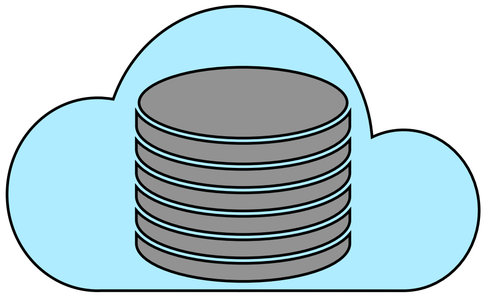
# How does Apache Maven work?

Maven™

# How does Apache Maven work?

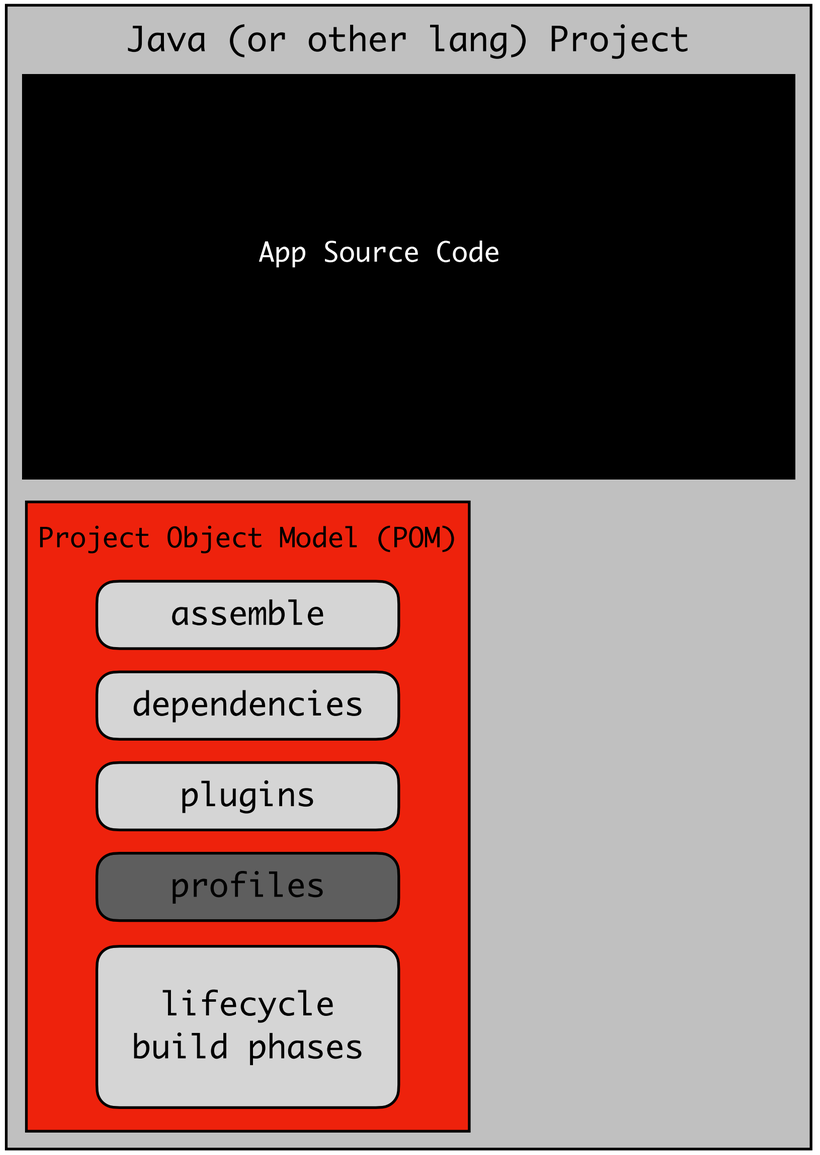



Watch for notes here



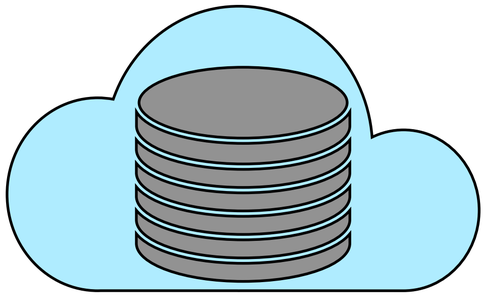
Internet Maven Repository

# How does Apache Maven work? (1)



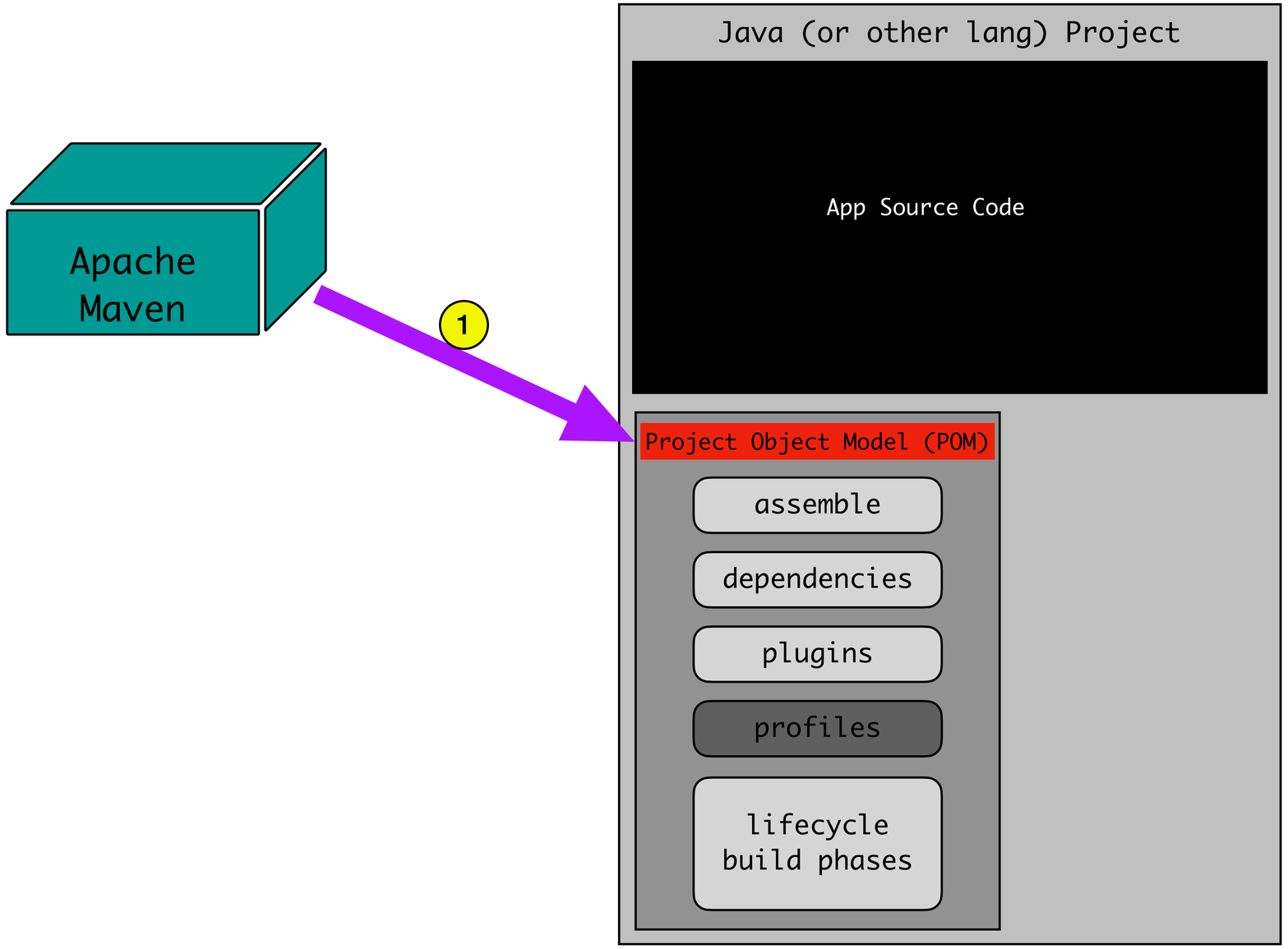

Notes

- Add a pom (typically a pom.xml file)
- Point settings.xml to repository
- Include dependencies & plugins
- Add properties & override configuration



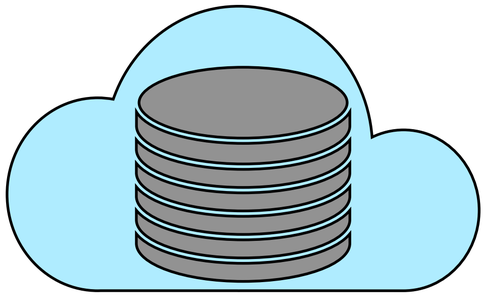
Internet Maven Repository

# How does Apache Maven work? (2)



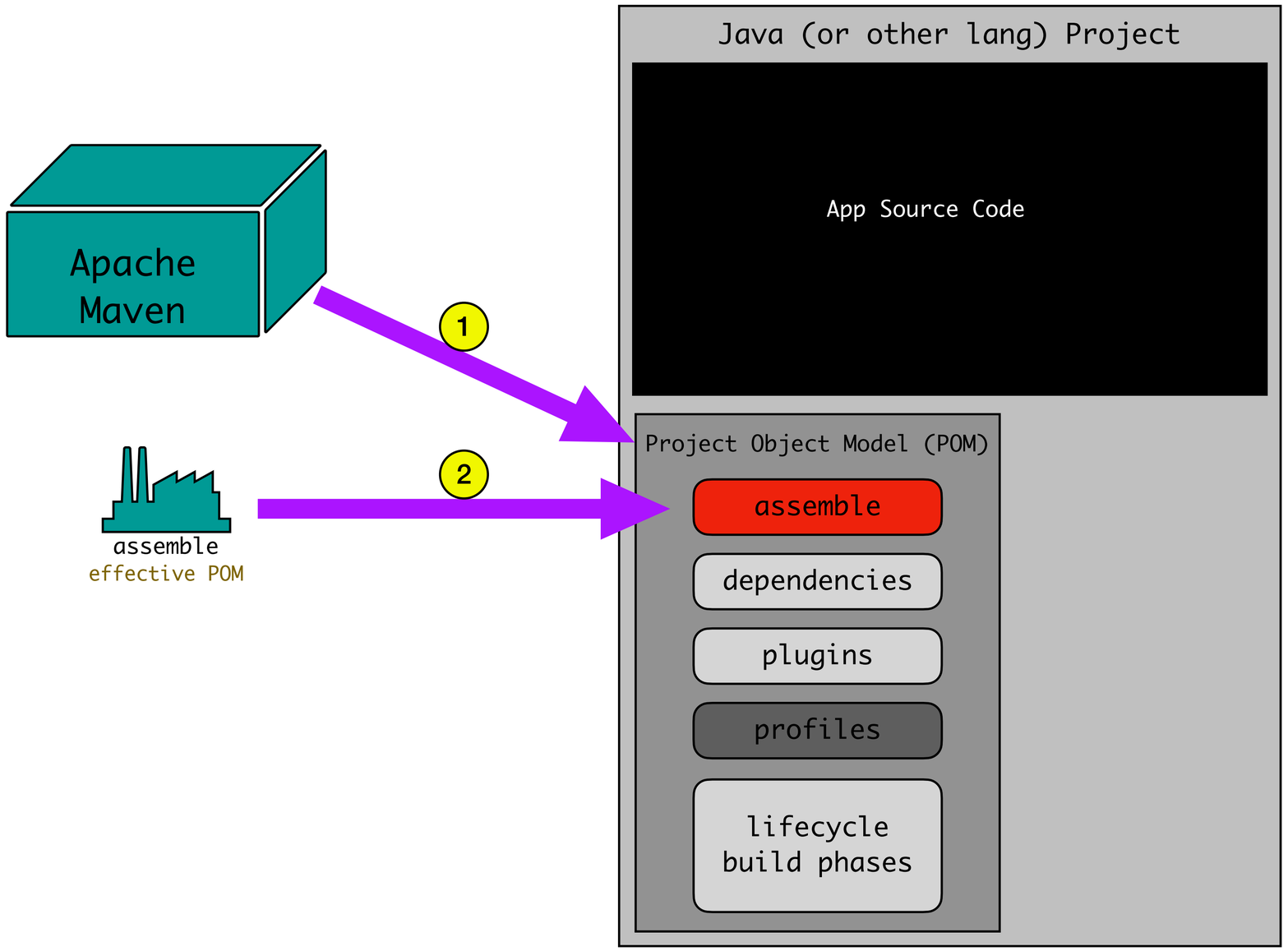

Notes

- Use Terminal or IDE
- Navigate to the project root
- List the phases (or phases:goals)
- Run Apache Maven with above



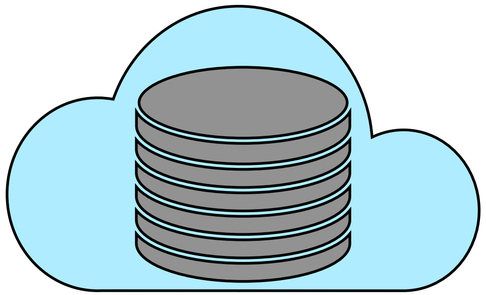
Internet Maven Repository

# How does Apache Maven work? (3)



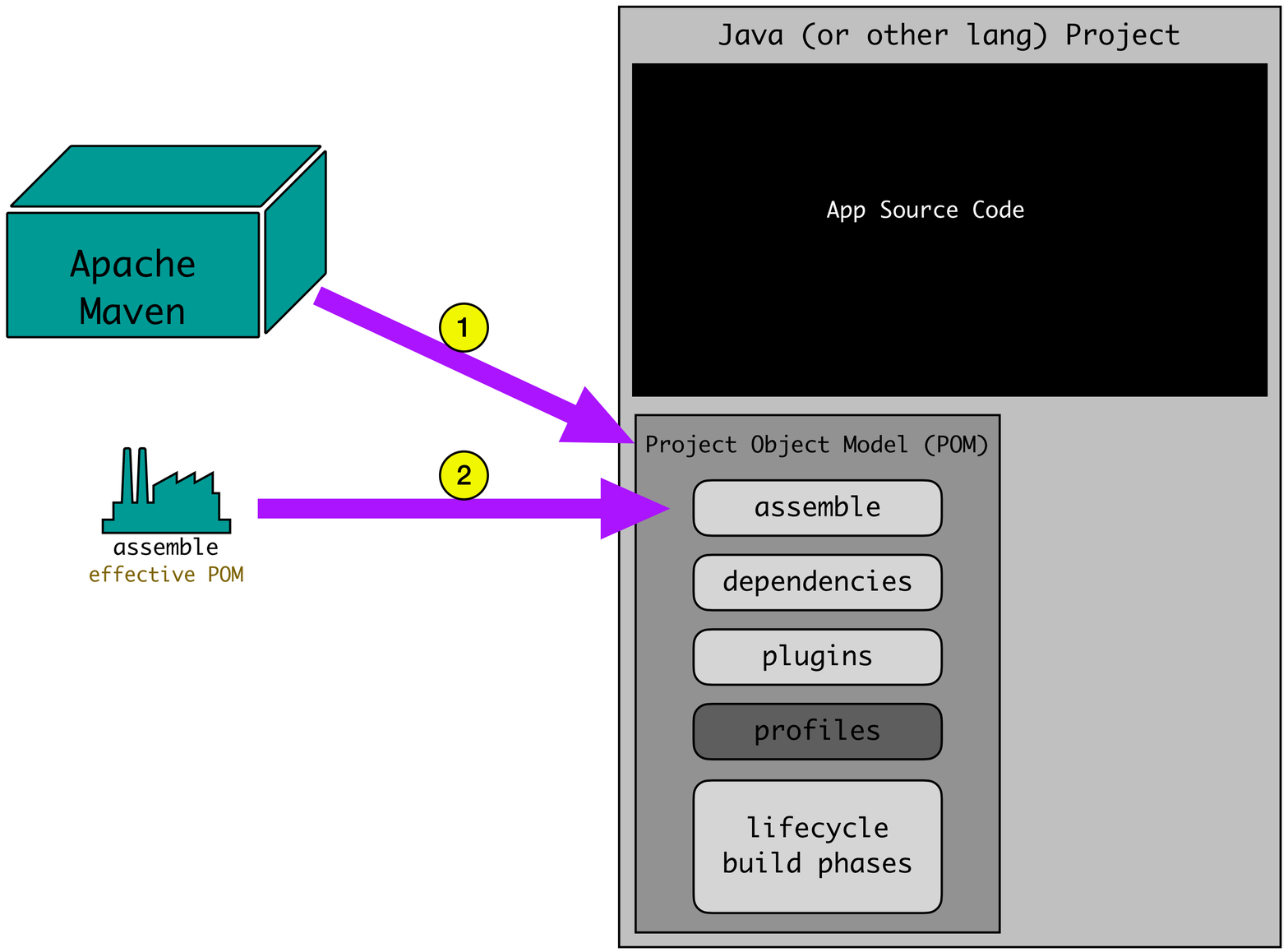

Notes

- Maven parses the POM
- Maven combines the below ...
  - Various settings
  - Maven defaults
  - POM overrides to defaults
- … to produce an “effective POM”
- Effective POM is read-only
- It is the full set of what Maven uses



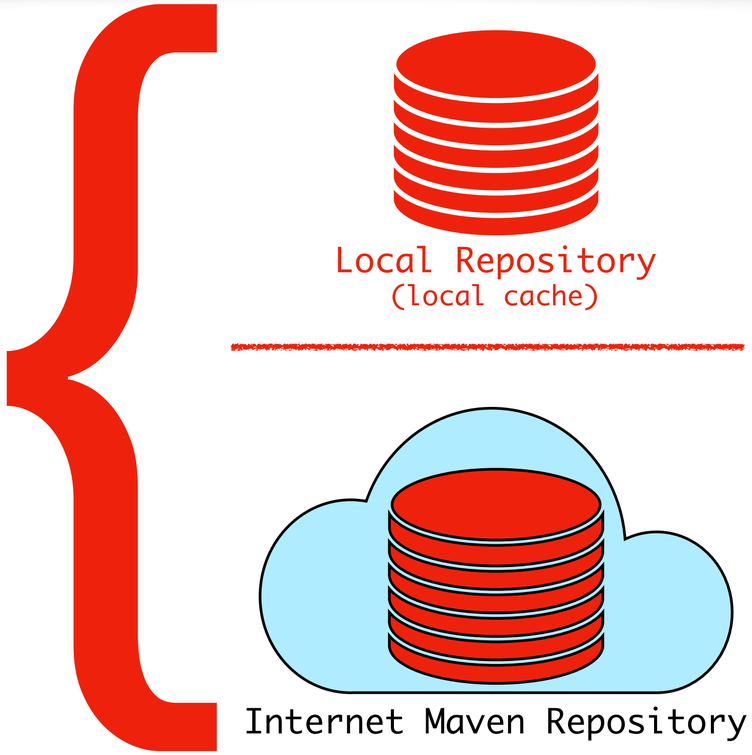
Internet Maven Repository

# How does Apache Maven work? (4)

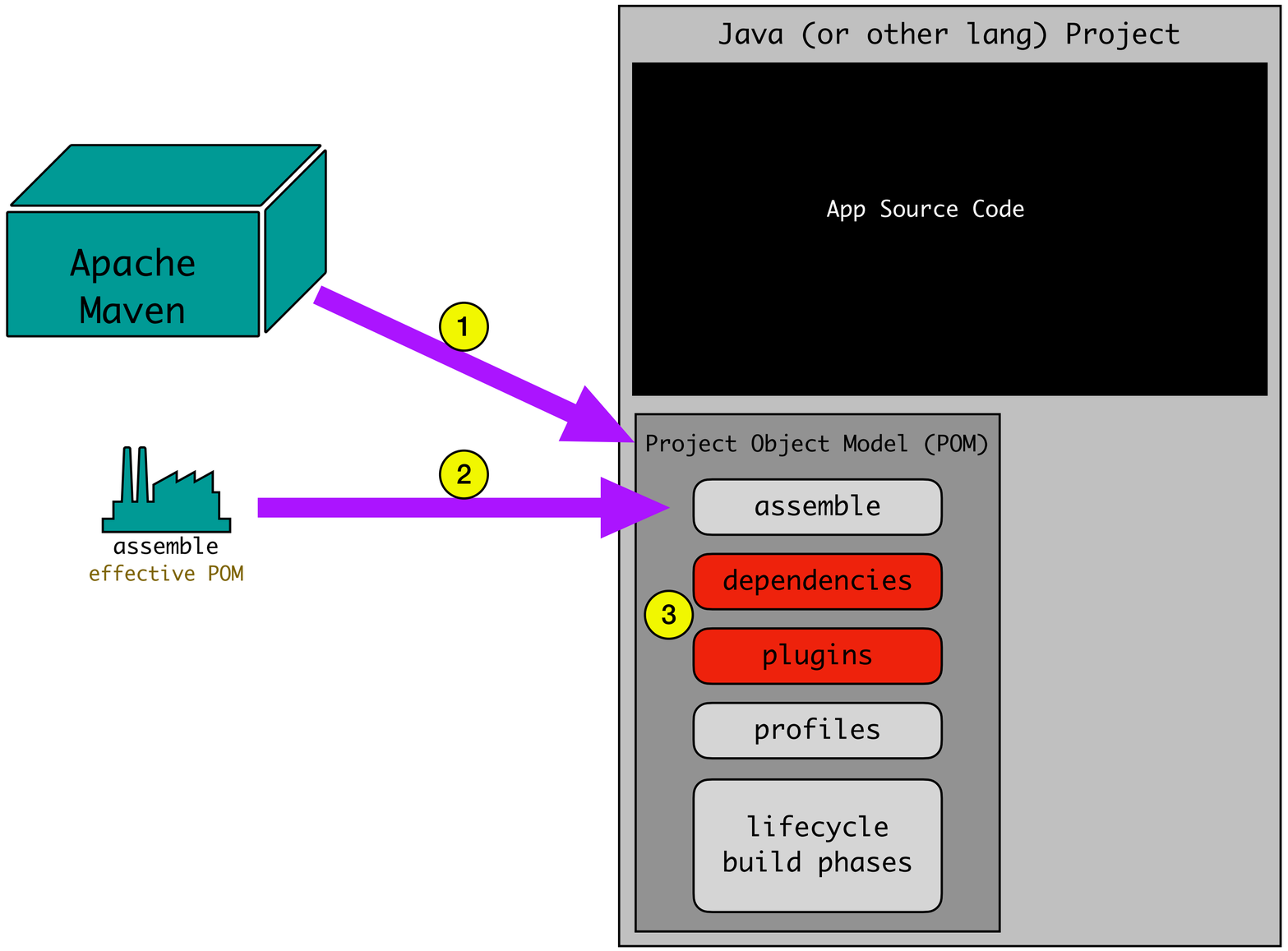



Notes

- If you previously ran a build
  - You may already have a local cache
  - Maven can use local cache
- If you don't have a local cache
  - Maven will use internet repository
  - Local cache will be created
  - Next run can depend on local cache

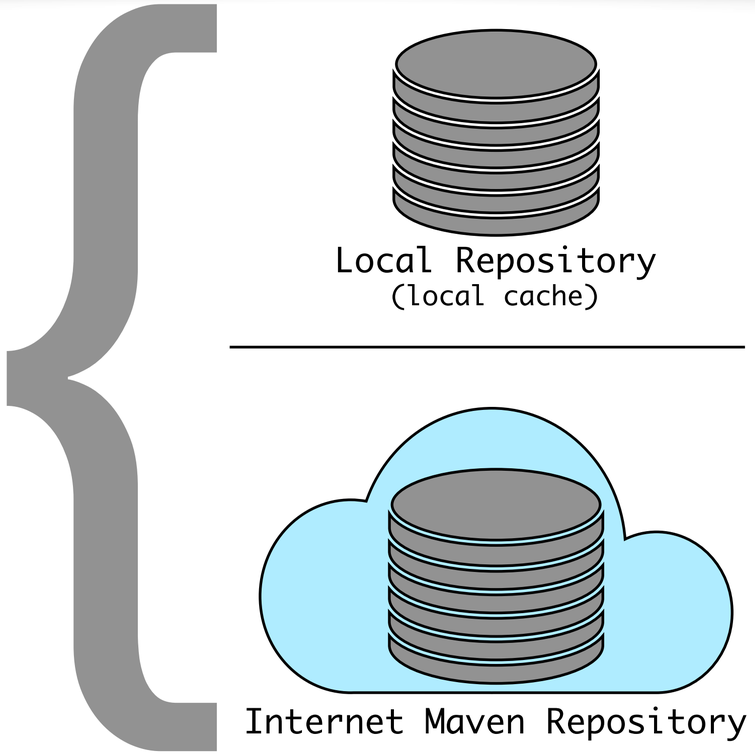


# How does Apache Maven work? (5)

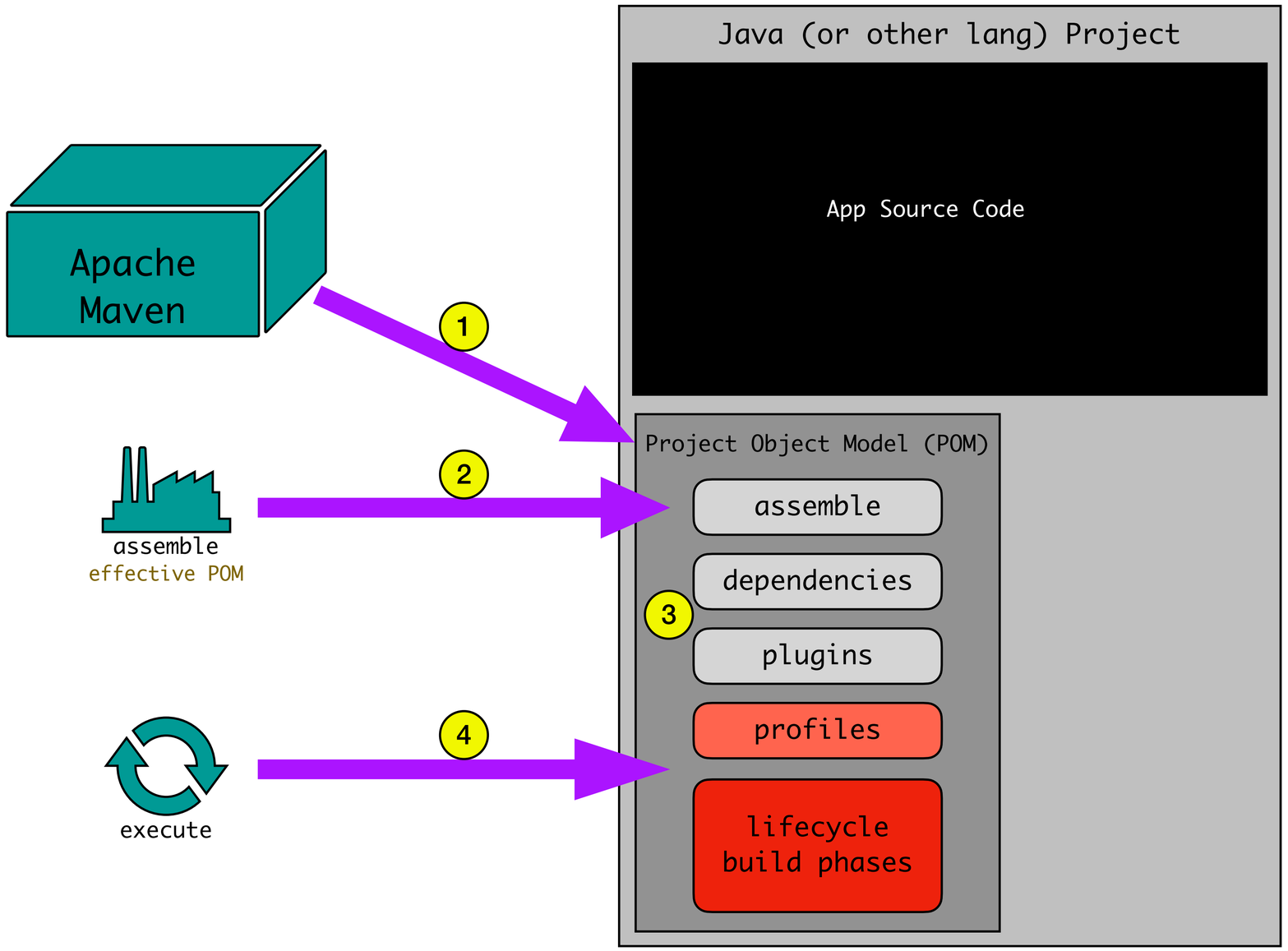



Notes

- Maven lists dependencies & plugins
- Includes all defaults for non-configured
- Creates a complete POM for building

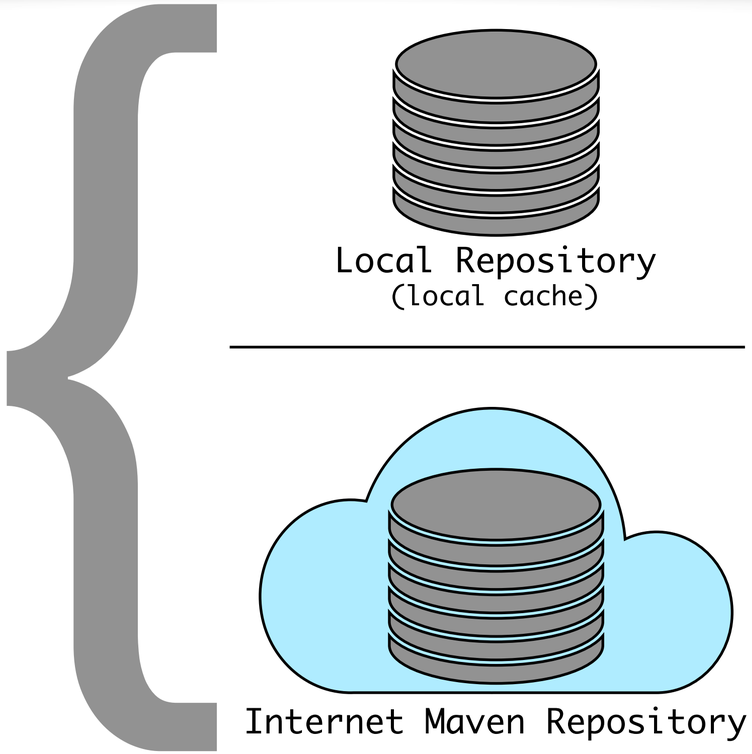


# How does Apache Maven work? (6)

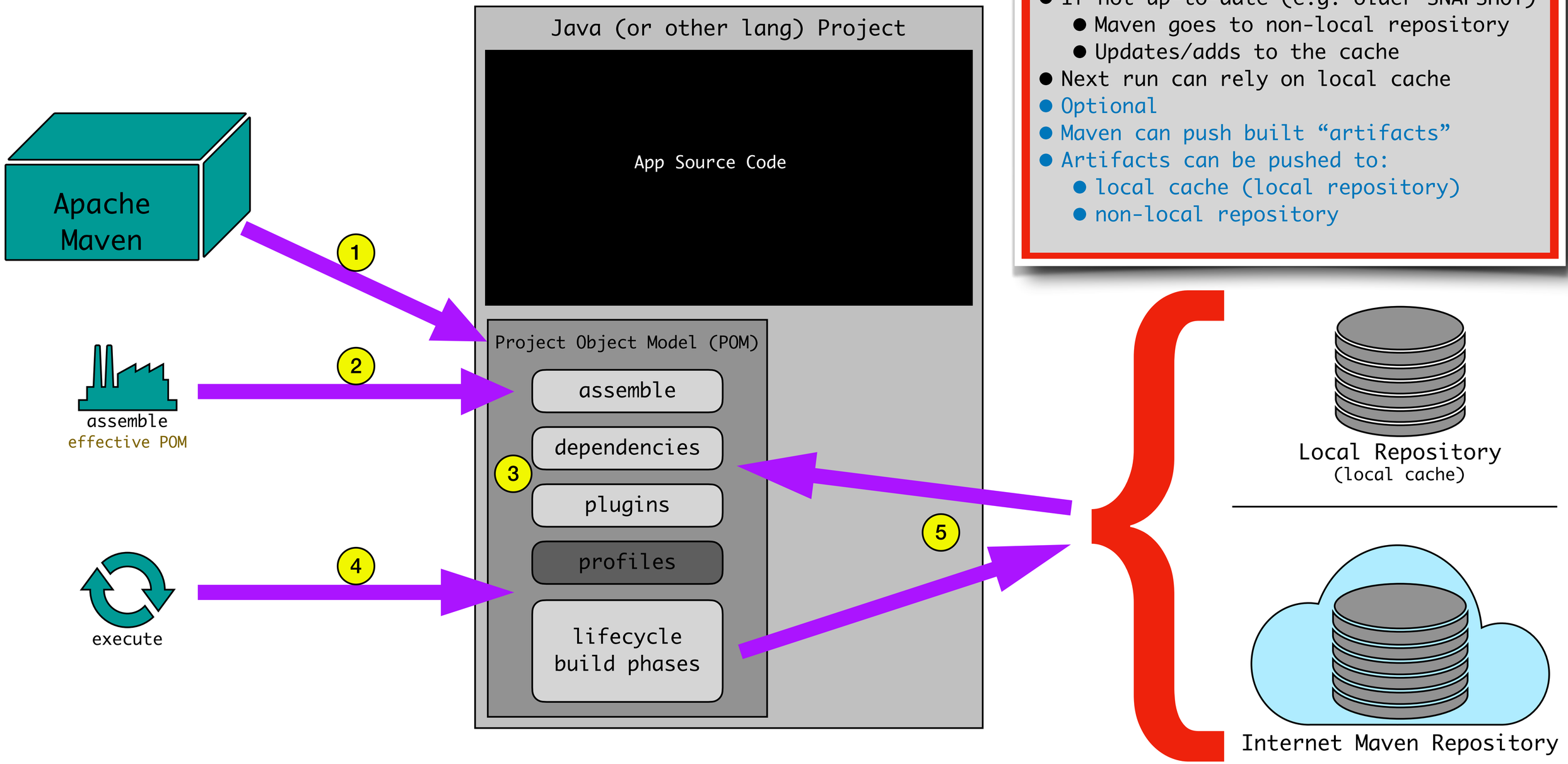



Notes

- Maven executes phases or phases:goals
- Profiles allow any or all of
  - alternate executions
  - alternate configurations
  - environment specificity



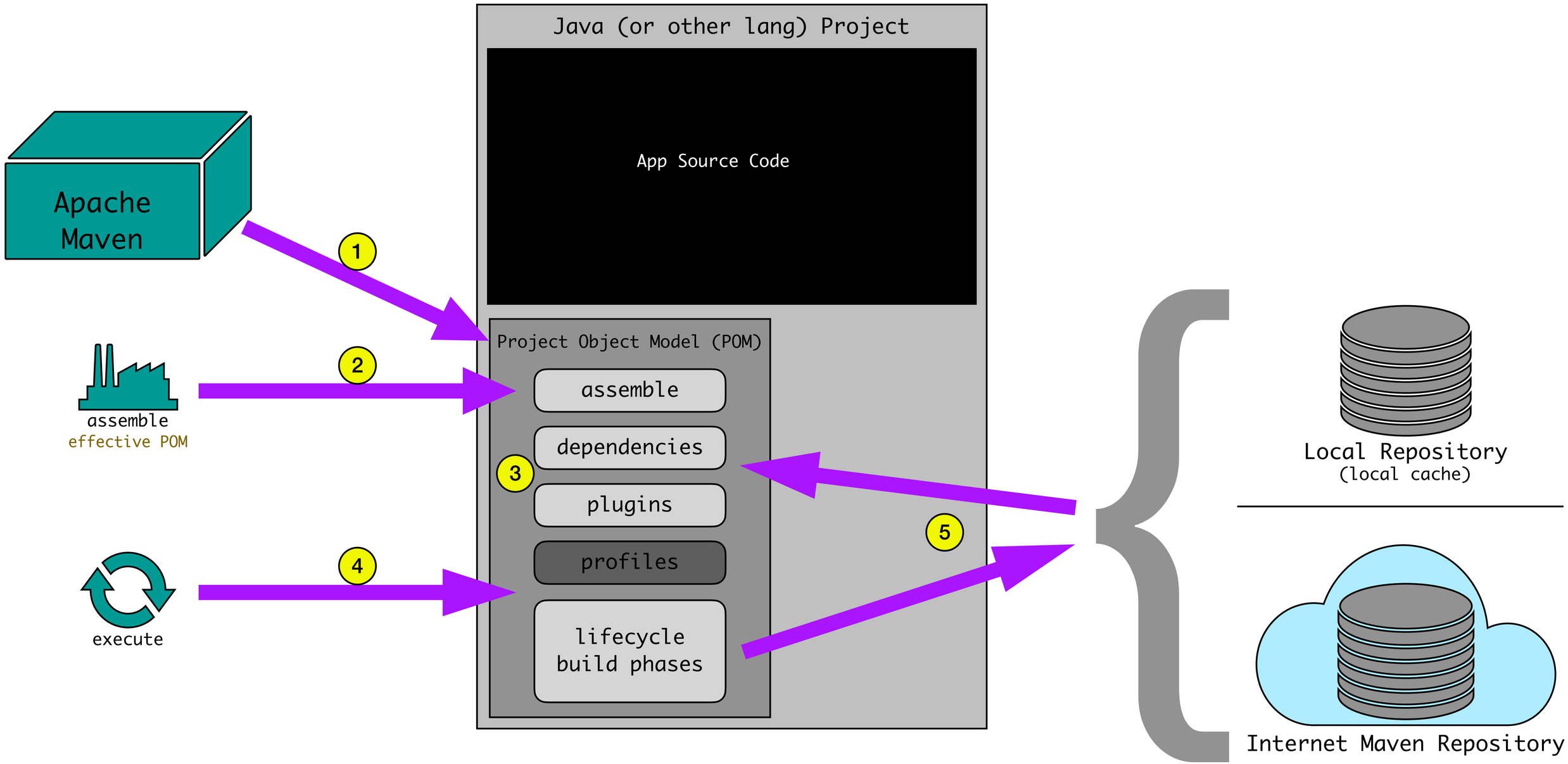
# How does Apache Maven work? (7)




Notes

- Maven uses local cache if up-to-date
- If not up-to-date (e.g. older SNAPSHOT)
  - Maven goes to non-local repository
  - Updates/adds to the cache
- Next run can rely on local cache
- Optional
- Maven can push built “artifacts”
- Artifacts can be pushed to:
  - local cache (local repository)
  - non-local repository

# How does Apache Maven work? (FINAL)

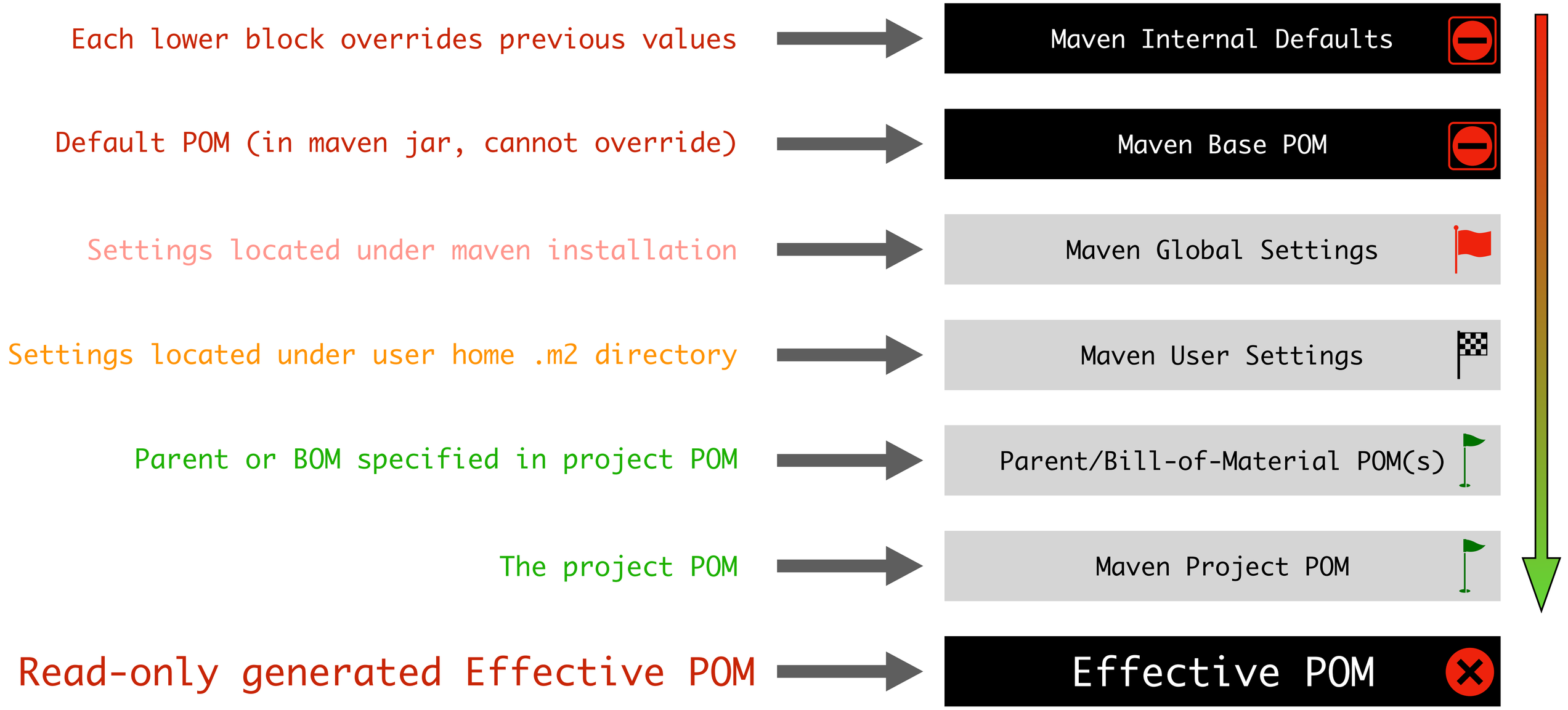


Effective POM?

# What is an “effective POM”?

- an assembly of execution steps, properties and profiles
- content that Maven can execute for the project
- exhaustive set of dependencies and plugins that can be used to build
- ready for execution by Maven executable

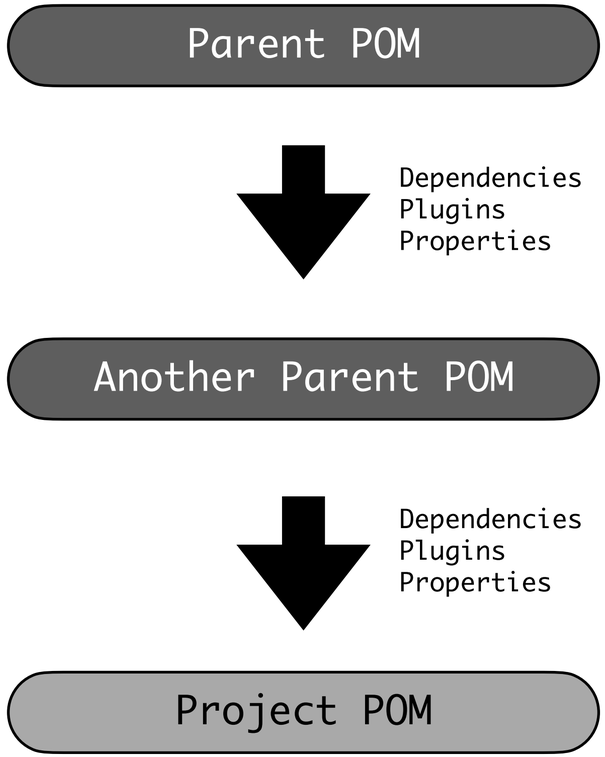
# How is the effective POM created?



# POM Hierarchies

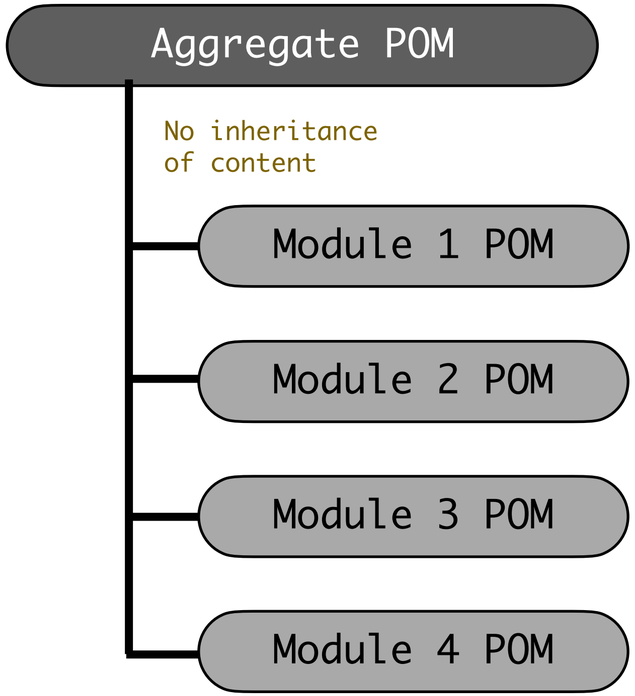
# POM Hierarchies

## Parentage



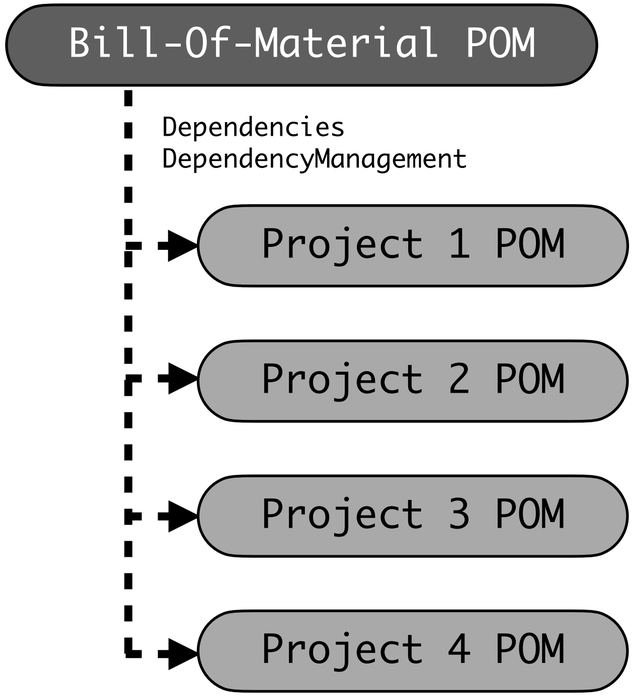

Project knows Parent
Parent does not know Project
Parent can aggregate Project

## Aggregation




Module does not know Aggregator
Module can set Aggregator as Parent
Aggregator knows Module

## Bill-Of-Materials




Project inherits/imports Bill-Of-Materials
Bill-Of-Materials does not know Project
Bill-Of-Materials can aggregate Project

# POM Hierarchies - example pom excerpts

## Parentage

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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
    <modelVersion>4.0.0</modelVersion>

    <artifactId>datetime-fetcher</artifactId>
    <version>1.0.0</version>

    <parent>
        <groupId>none.cvg.maven</groupId>
        <artifactId>maven-multi</artifactId>
        <version>1.0.0</version>
    </parent>

    <packaging>jar</packaging>

    <name>datetime-fetcher</name>

</project>
```

Project knows Parent
Parent can aggregate Project

## Aggregation

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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
    <modelVersion>4.0.0</modelVersion>

    <groupId>none.cvg.maven</groupId>
    <artifactId>maven-multi</artifactId>
    <version>1.0.0</version>

    <packaging>pom</packaging>

    <name>maven-multi</name>

    <modules>
        <module>datetime-fetcher</module>
        <module>greeting</module>
        <module>greeting-generator</module>
    </modules>

    <build>
        <pluginManagement>
            <plugins>
                <!-- Enforce the plugin version, rather than rely
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.8.1</version>
                </plugin>
                <!-- Enforce the plugin version, rather than rely
```

Module can set Aggregator as Parent
Aggregator knows Module

## Bill-Of-Materials

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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://ma
    <modelVersion>4.0.0</modelVersion>

    <groupId>none.cvg.maven</groupId>
    <artifactId>maven-spring-bom</artifactId>
    <version>1.0.0</version>

    <packaging>jar</packaging>

    <name>maven-spring-bom</name>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-framework-bom</artifactId>
                <version>${spring-framework-bom.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>


    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
```

Project inherits/imports Bill-Of-Materials
Bill-Of-Materials does not know Project

Maven 3.x …

# Current state of Apache Maven 3

- Apache Maven is versatile, comprehensive and a great fit for corporations!
- Some challenges exist:
  - Apache Maven 3: It is painful to maintain versions in multi-module projects
  - Apache Maven 3: Is no implicit way to separate build info from consumer info
  - Apache Maven 3: Handle versions of (dependency) sibling modules is a chore!
  - Apache Maven 3: We developers are lazy! We use default versions for plugins
  - Apache Maven 3: Creating a Bill-of-Materials POM is not easy!

# Maintaining child module versions in Apache Maven 3

- No built-in support to propagate new version to child modules
- Codehaus offers the flatten-maven-plugin
  - The flatten-maven-plugin has issues around profile interpolation
  - It requires including the third-party plugin

    https://maven.apache.org/maven-ci-friendly.html
- Outbrain offers a ci-friendly-flatten-maven-plugin
  - This plugin solves the issue with profile interpolation
  - It requires including the third-party plugin

    https://github.com/outbrain/ci-friendly-flatten-maven-plugin
- Both plugins use a revision property to set version values

# Build info versus consumption info in Apache Maven 3

- POM contains build information - which is not relevant to a consumer!
- Building an artifact produces a .pom file which contains irrelevant info
- Once again both third party plugins come in handy:
  - Codehaus : flatten-maven-plugin
  - Outbrain : ci-friendly-flatten-maven-plugin

# Versions of (dependency) sibling modules in Apache Maven 3

- Some multi-module projects have sibling dependencies
- Managing versions of such can require using the project.version property
- Limits testing to
  - either using most recent version of all siblings
  - or hardcoding versions during testing

# Staying current on plugin versions in Apache Maven 3

- Apache Maven provides default versions of some plugins to use.
- Best practice is to always set exact versions of plugins
- In Apache Maven 3, there is no way to identify if a default version is in use.

# Creating a Bill-of-Materials POM in Apache Maven 3

- Apache Maven has no “bom” packaging.

Let’s see some code!

Maven 4.x …

https://github.com/c-guntur/maven4

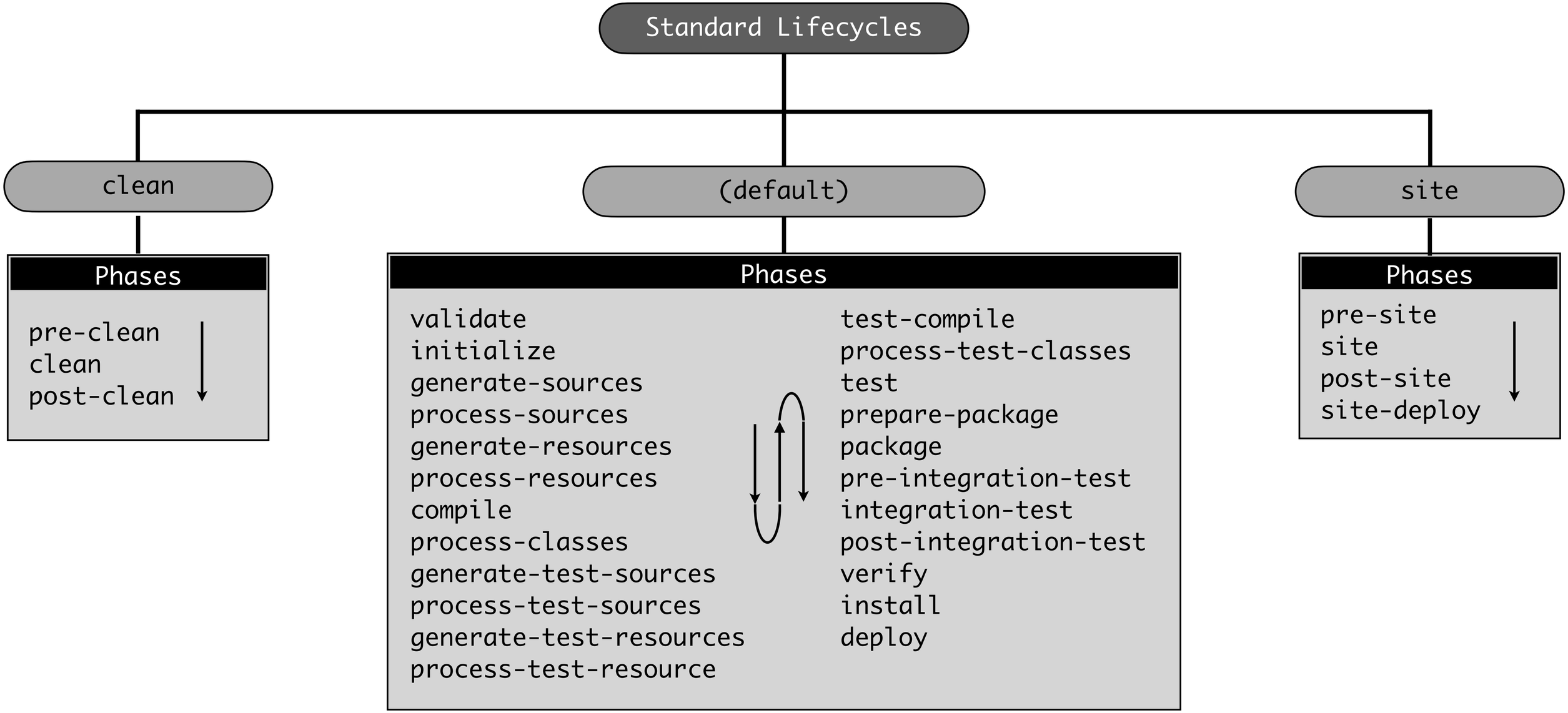
# Current state of Apache Maven 3

- Apache Maven is versatile, comprehensive and a great fit for corporations!
- Some challenges exist:
  - Apache Maven 3: It is painful to maintain versions in multi-module projects
    Apache Maven 4: Version support is awesome!
  - Apache Maven 3: Is no implicit way to separate build info from consumer info
    Apache Maven 4: 2 separate POM files are generated!
  - Apache Maven 3: Handle versions of (dependency) sibling modules is a chore!
    Apache Maven 4: Implicit versions for (dependency) sibling modules
  - Apache Maven 3: We developers are lazy! We use default versions for plugins
    Apache Maven 4: Maven issues warnings when plugin versions are not declared
  - Apache Maven 3: Creating a Bill-of-Materials POM is not easy!
    Apache Maven 4: Introduces the “bom” packaging option

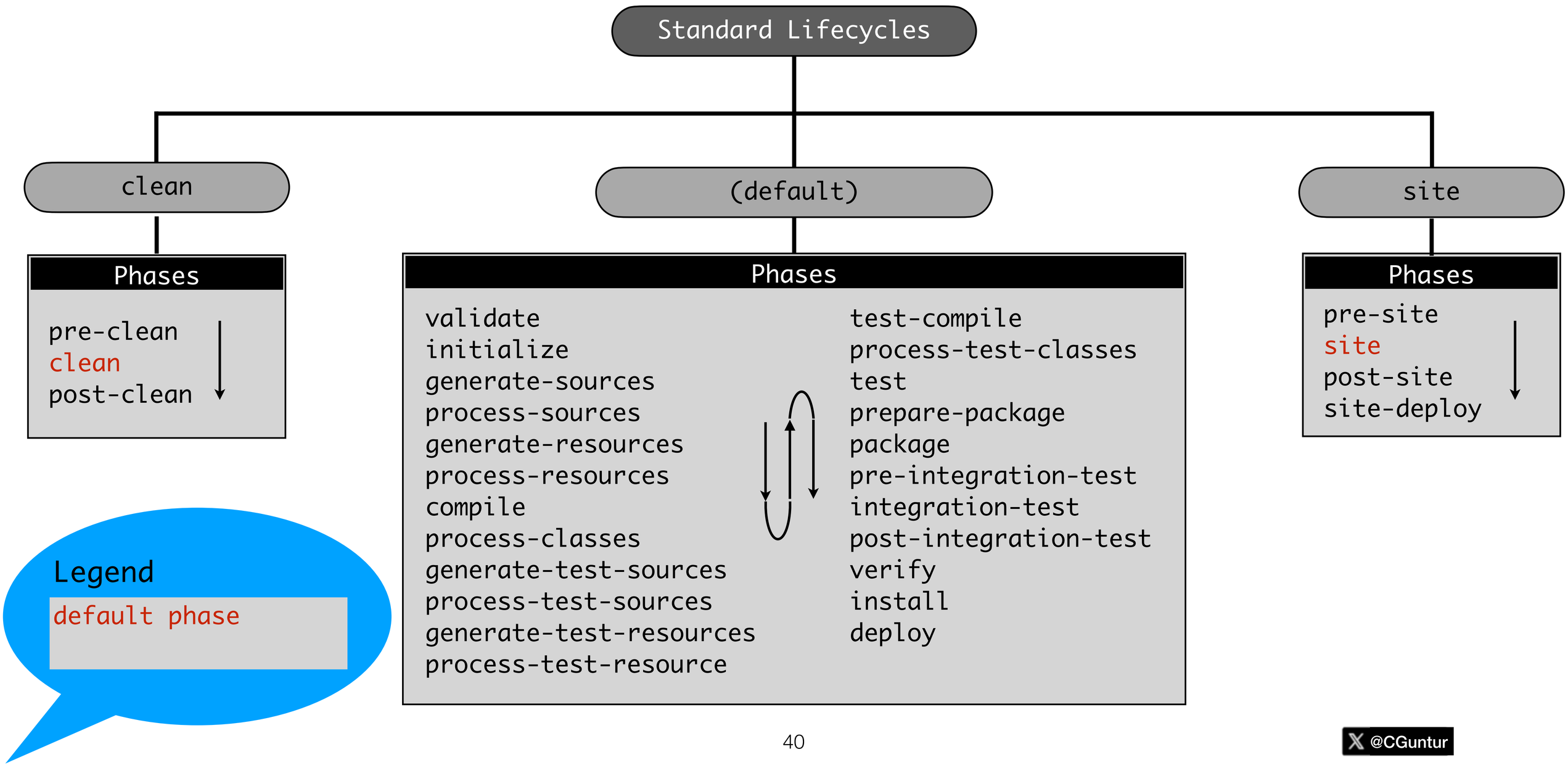
# Under the hood

Maven™

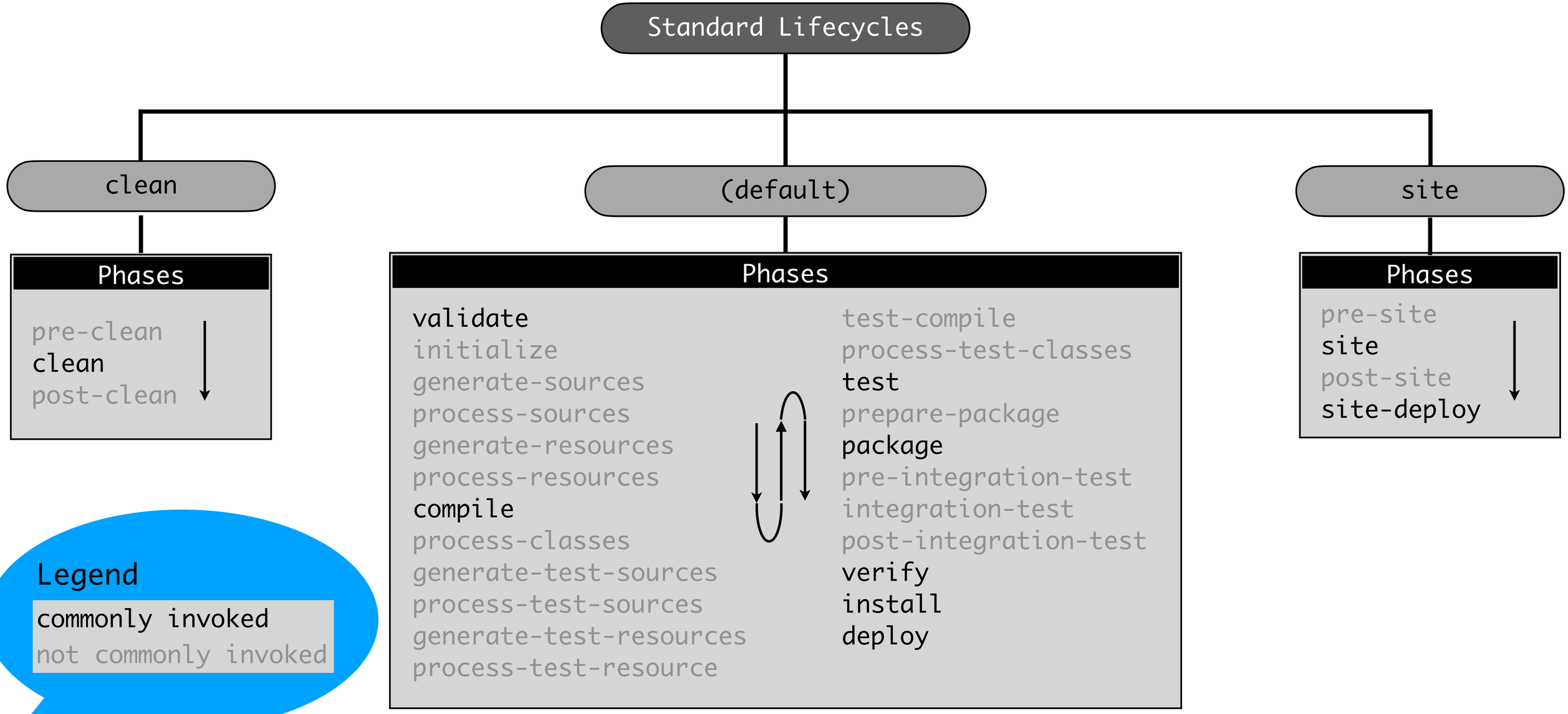
# Lifecycles in Maven



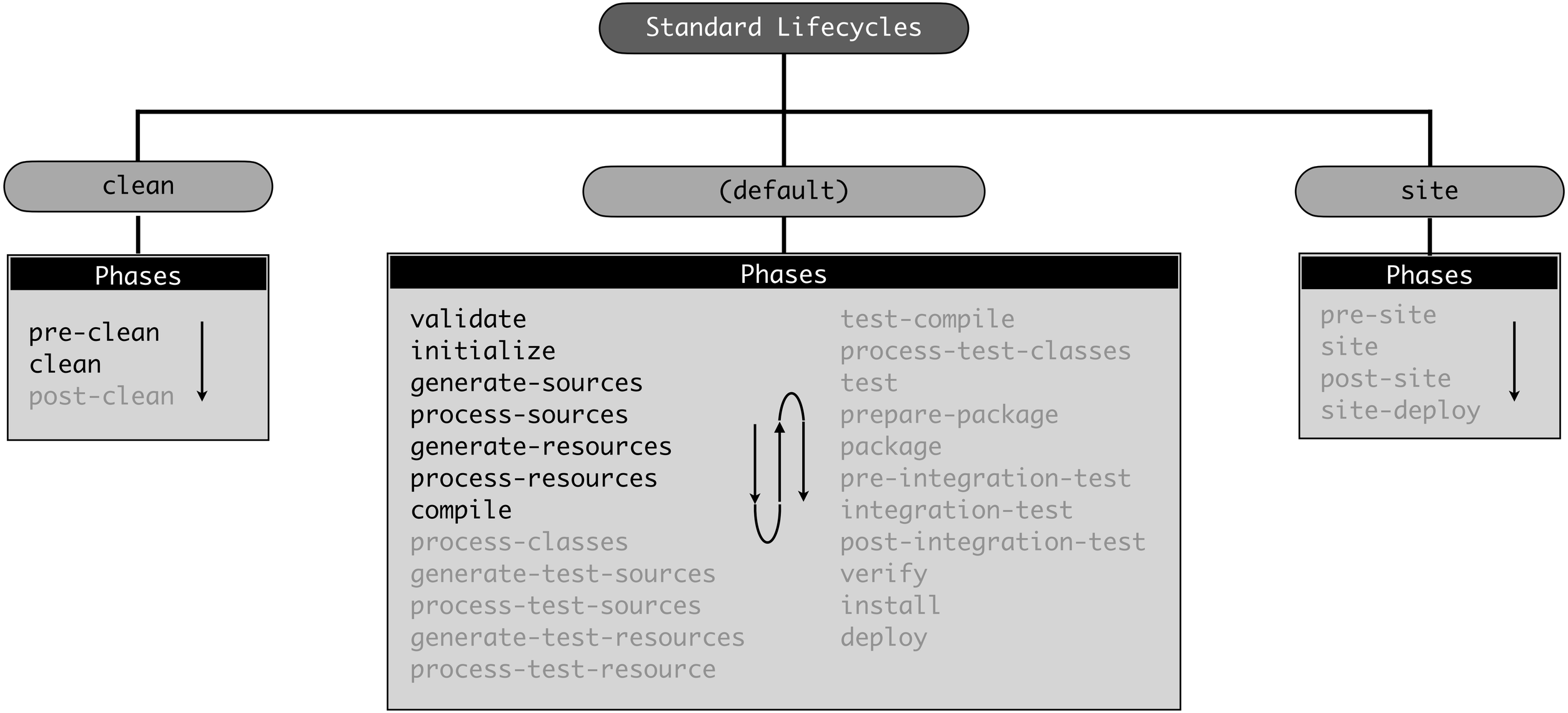
# Lifecycles in Maven - default phases



# Lifecycles in Maven - common phases used



# Lifecycles in Maven - calling “mvn clean compile”



# Lifecycles: Under the hood

- command: `mvn clean compile`
  - first calls `clean` (default phase of `clean` lifecycle is `clean`):

    `pre-clean`

    `clean`
  - then invokes the following phases of `default` lifecycle:

    `validate`

    `initialize`

    `generate-sources`

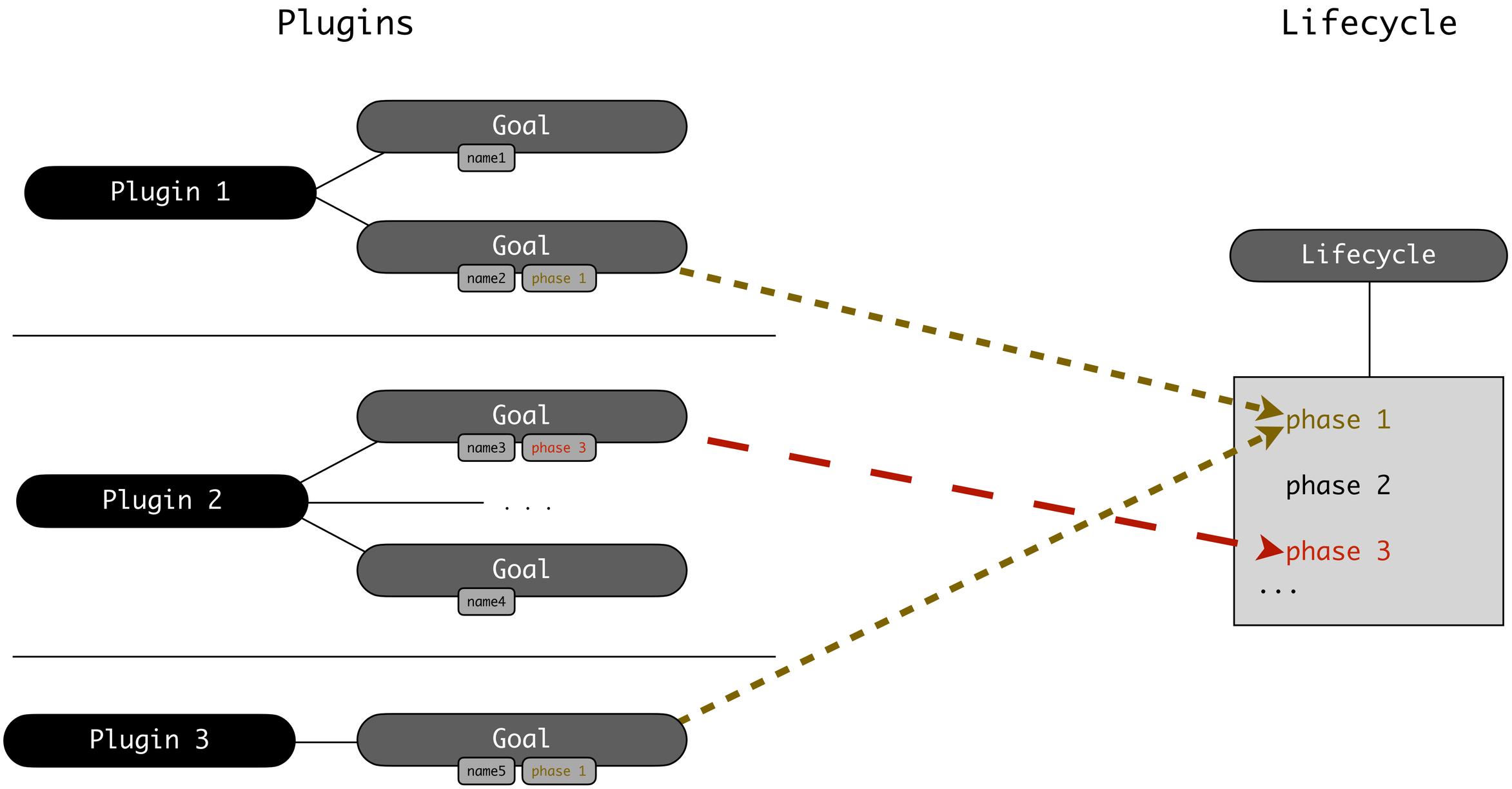
    `process-sources`

    `generate-resources`

    `process-resources`

    `compile`

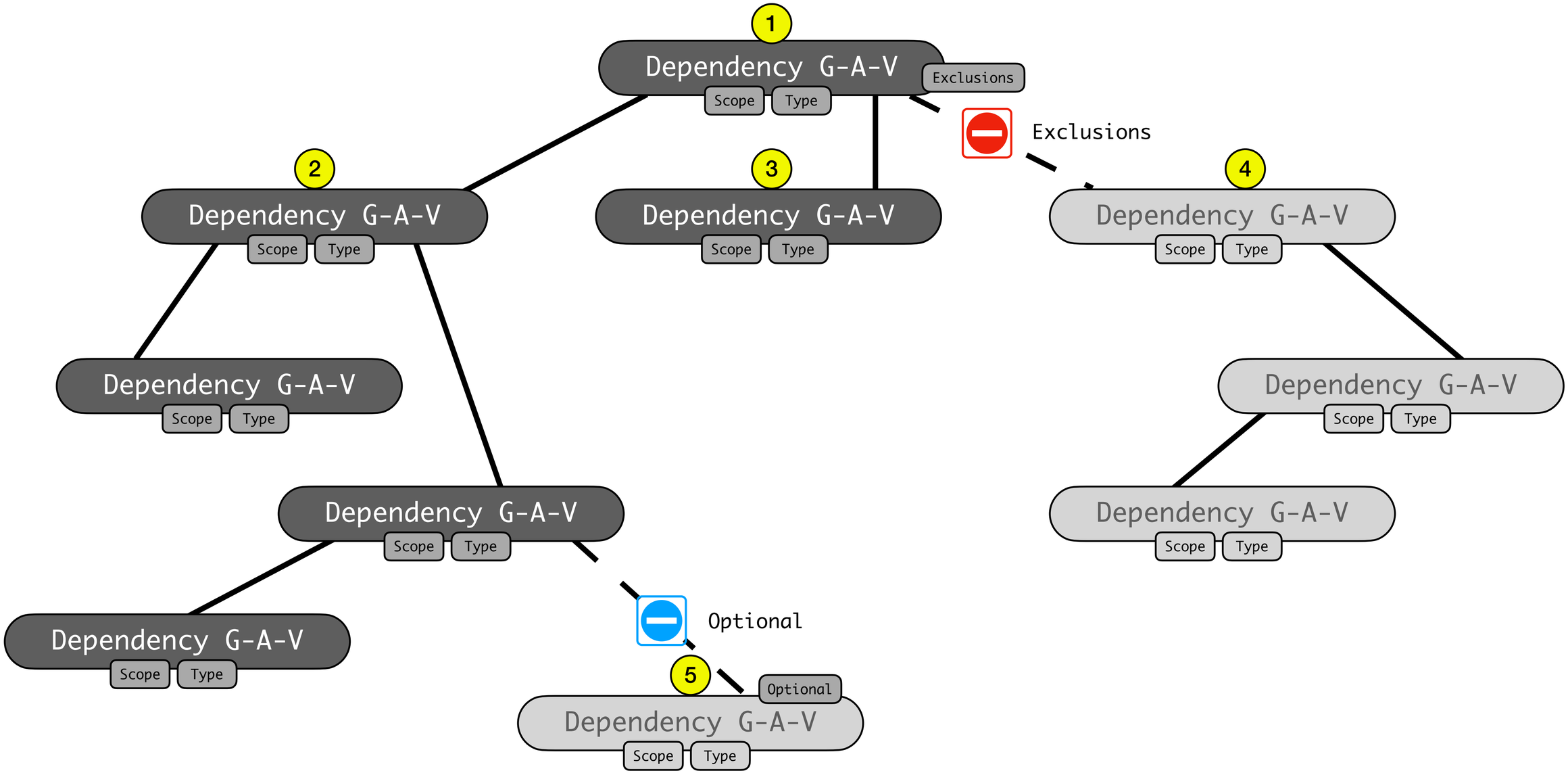
# Plugins



# Plugins: Under the hood

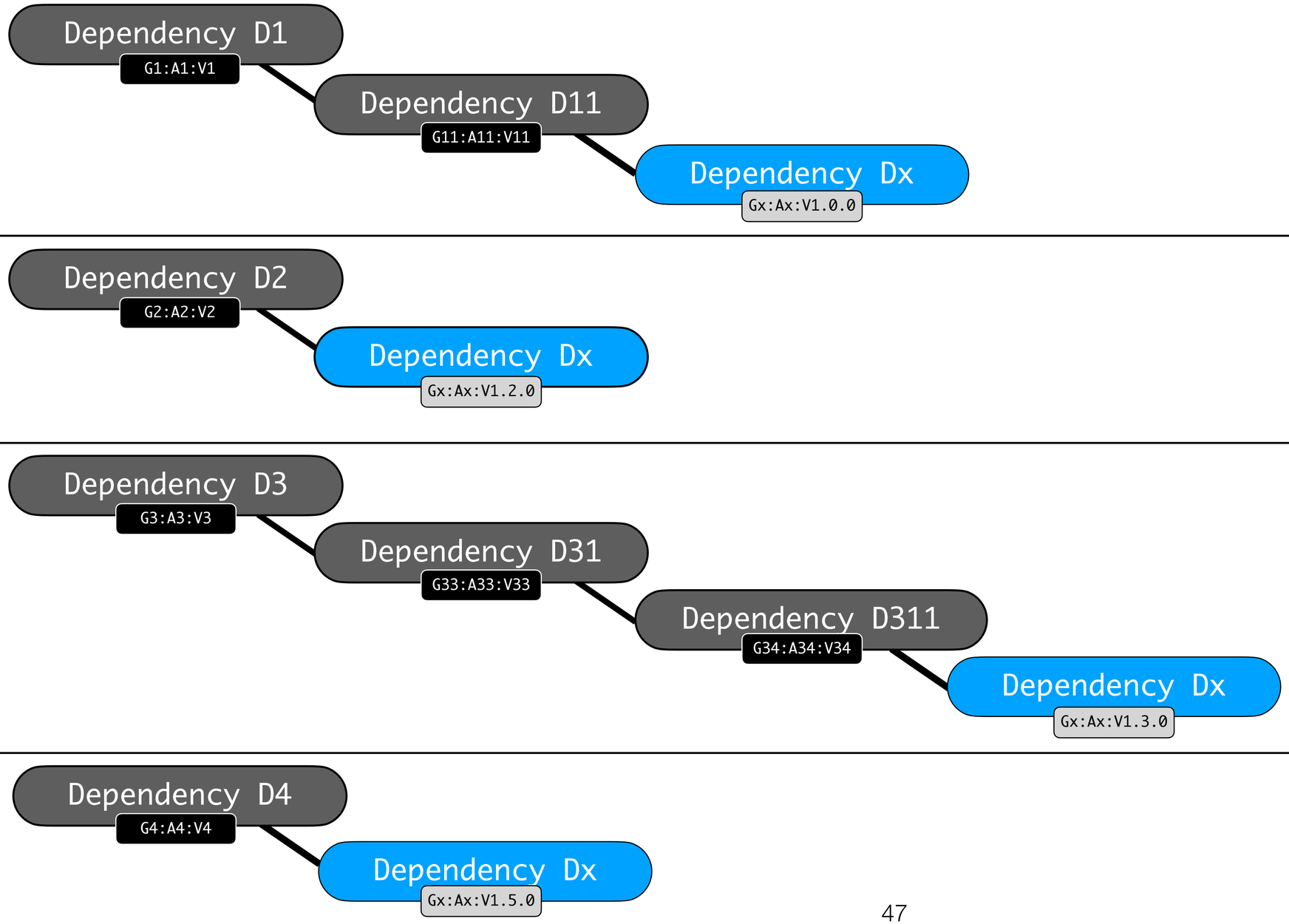
- a lifecycle phase by itself does not perform any task
- tasks are performed by plugins
- plugins allow phases to carry out responsibilities
- plugins declare goals
- a plugin goal defines a specific task towards building/managing the project
- a plugin goal may be bound to zero or more lifecycle phases
- invoking a phase invokes plugin goals linked to the phase
- in addition to calling lifecycles or their phases, possible to invoke a plugin goal
- default bindings of lifecycle phase to plugin goal may depend on packaging
  - package phase for ejb will call ejb goal from the maven-ejb-plugin
  - package phase for jar will call jar goal from the maven-jar-plugin
  - generate-resources phase for ear will call generate-application-xml goal from the maven-ear-plugin

# Dependencies



# Dependencies

## Dependency Graph



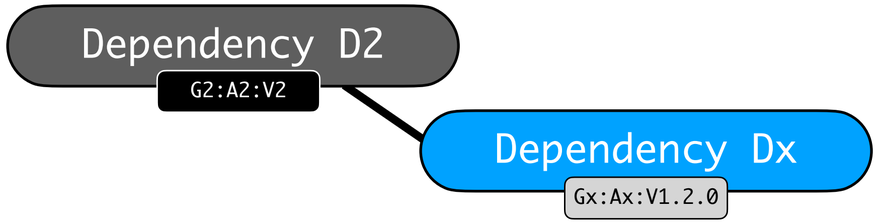

## Selection

# Dependencies

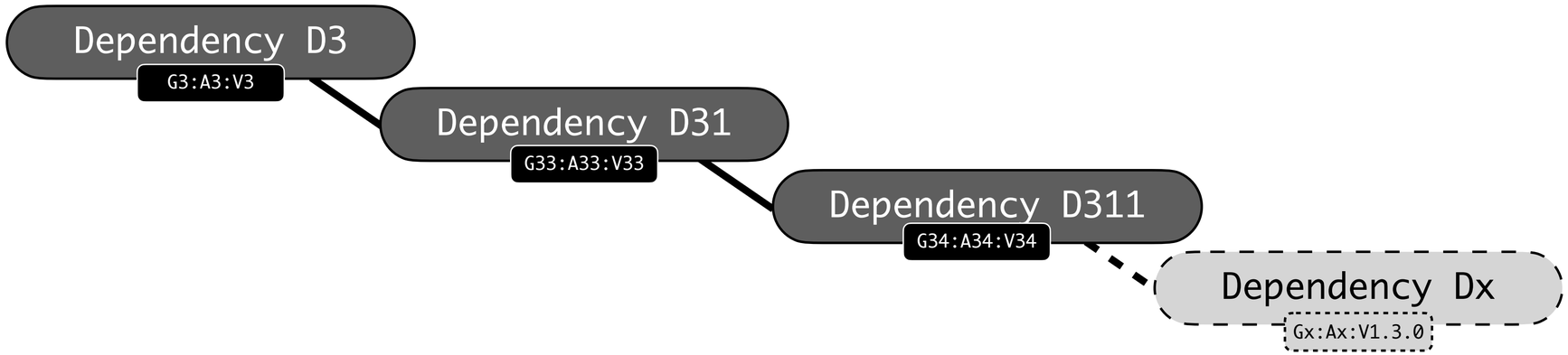
Dependency Graph

Selection

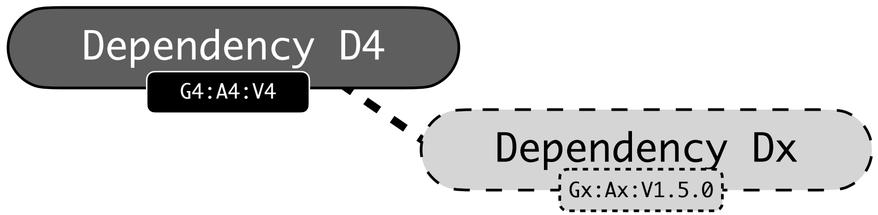

Omitted v1.0.0
Not nearest in depth




Selected v1.2.0
Nearest in depth
First in resolution path




Omitted v1.3.0
Not nearest in depth



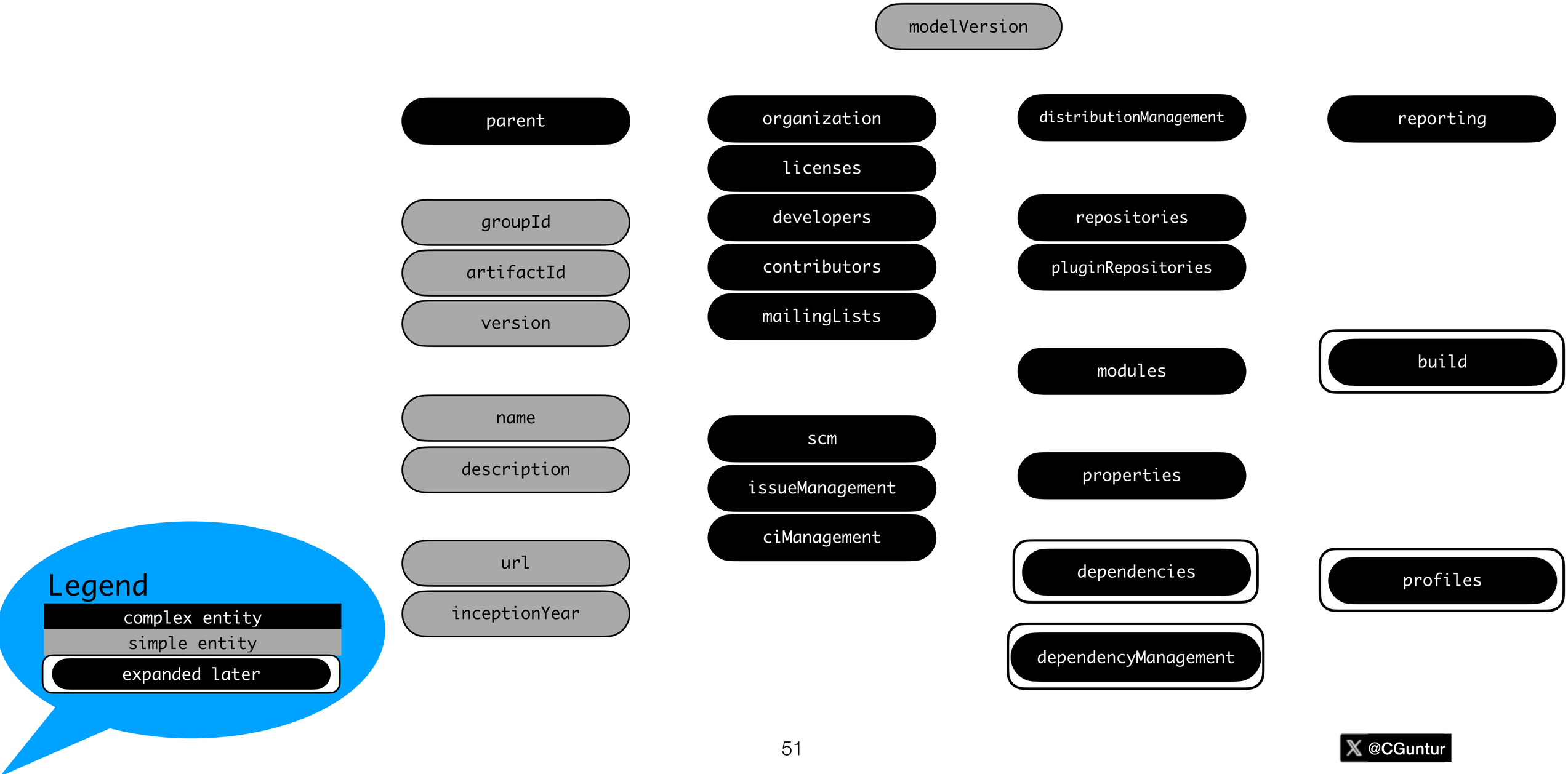

Omitted v1.5.0
Not first in resolution path

There is a lot more …
That's all we have time for!

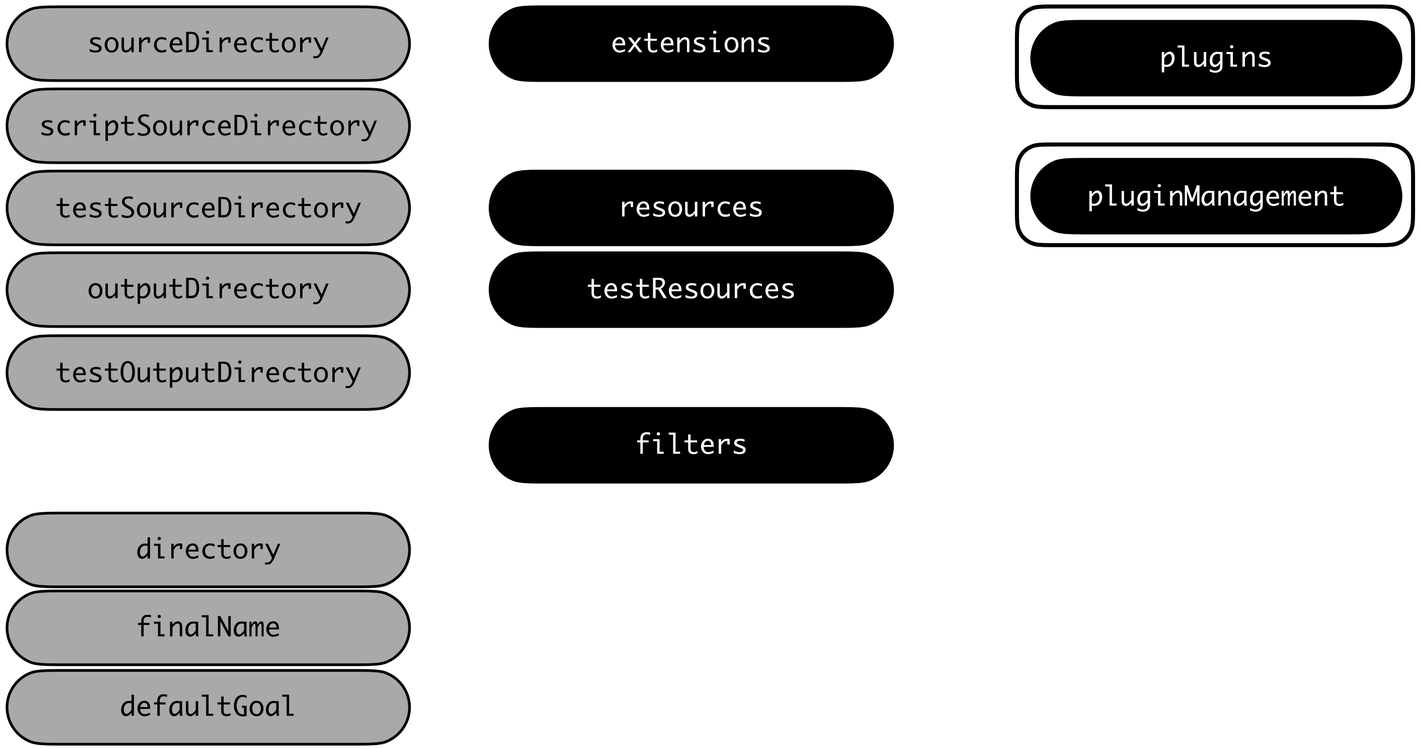
# Questions?

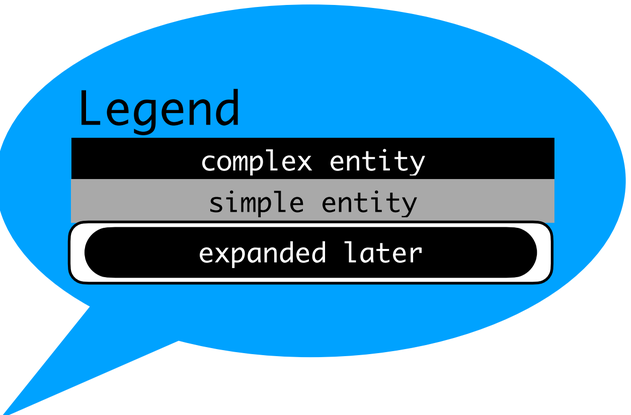
# Appendix

# Maven POM Reference - project

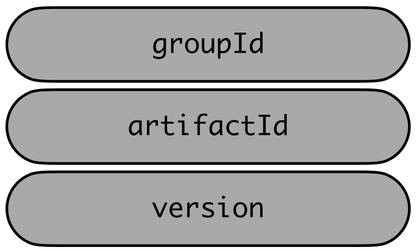


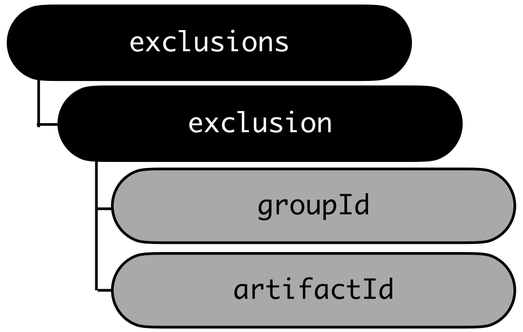
# Maven POM Reference - build

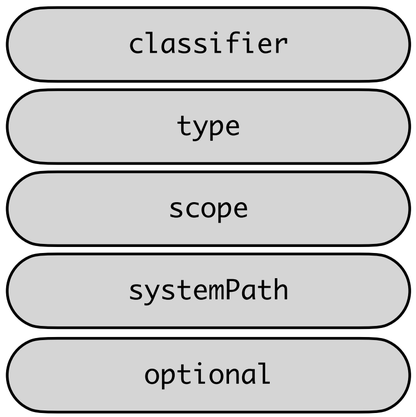


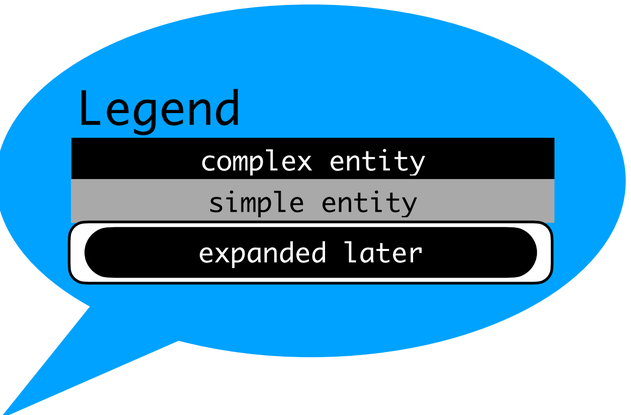



# Maven POM Reference - [dependencyManagement/] dependencies/dependency

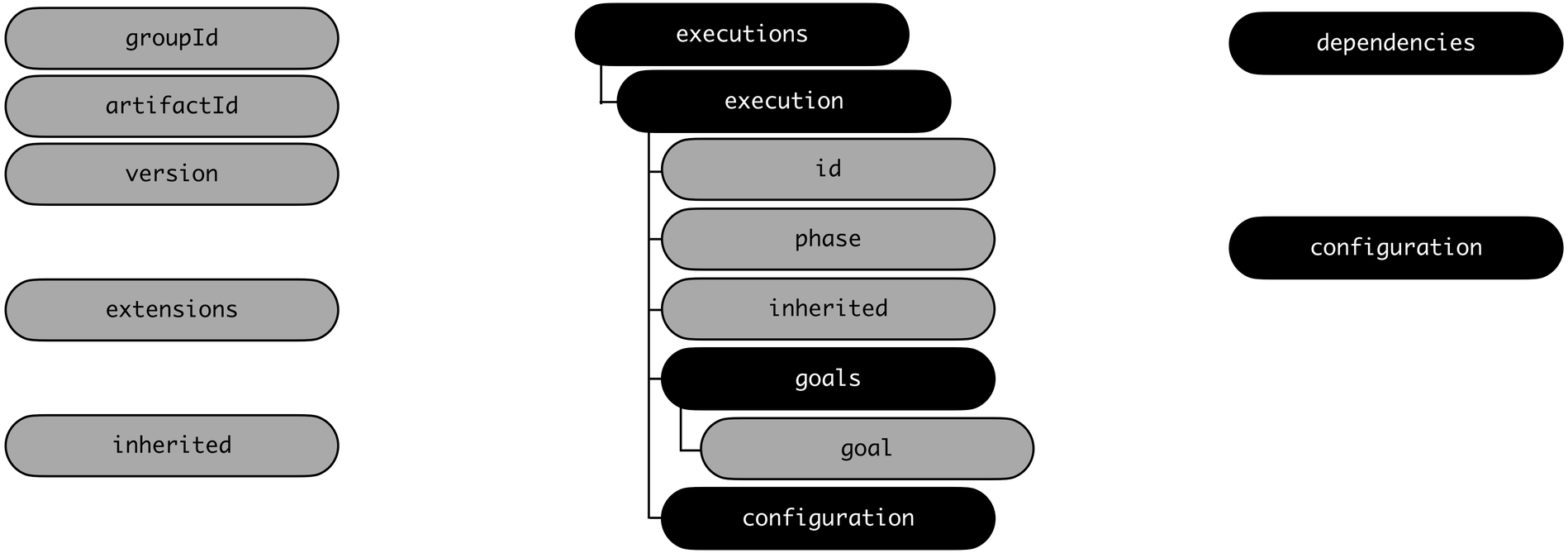


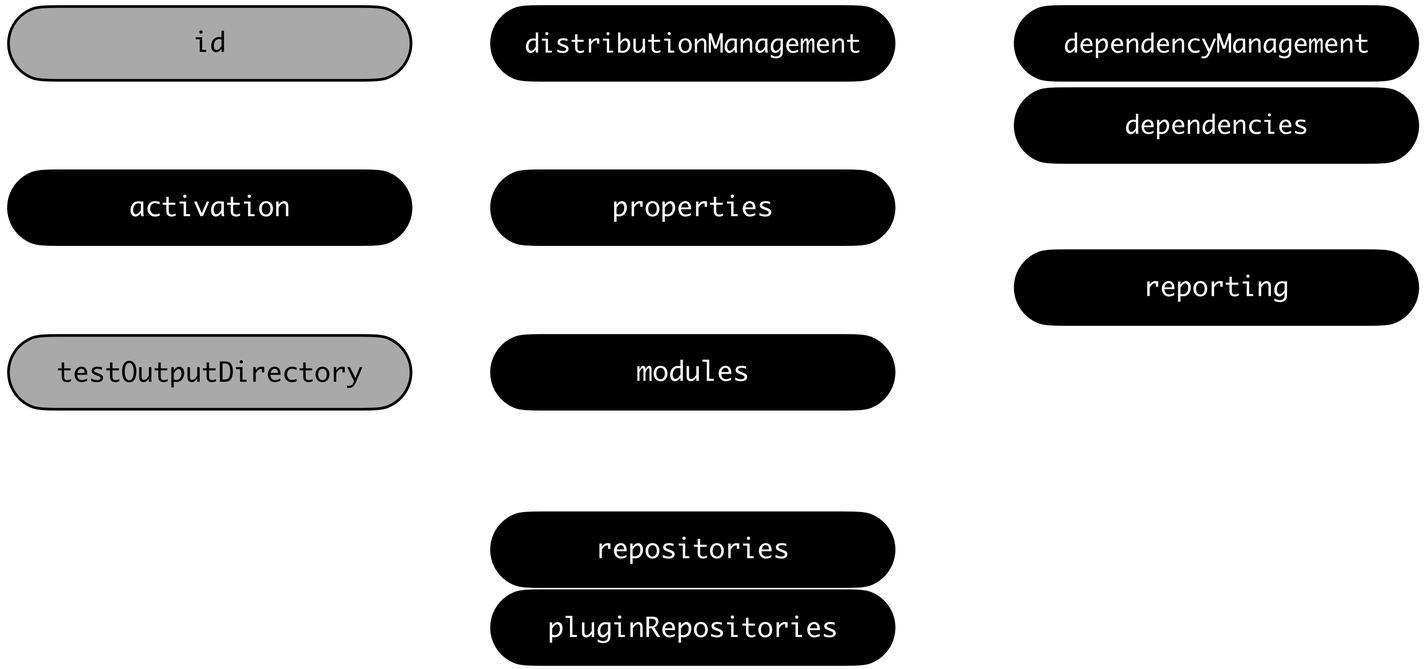



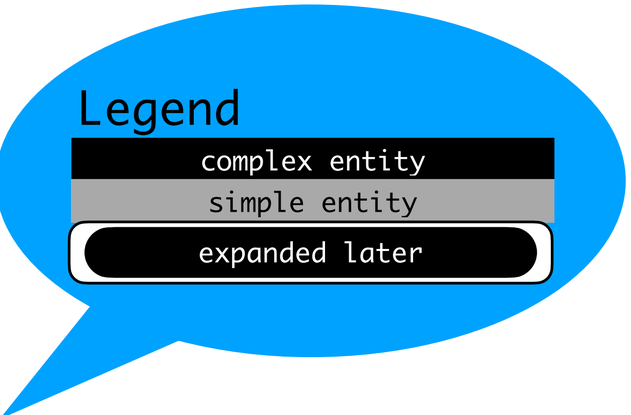



# Maven POM Reference - [pluginManagement/] plugins/plugin



# Maven POM Reference - profiles/profile



# Configuring Apache Maven

## Environment Variables

- JAVA_HOME
- M2_HOME
- PATH
- MAVEN_OPTS

## .mvn Config Files

- jvm.config
- mvn.config

## XML Configurations

- <Maven Installation>/conf/settings.xml
- <User Home>/.m2/settings.xml
- <Project Basedir>/.mvn/extensions.xml
- <User Home>/.m2/toolchains.xml
- <Project Basedir>/toolchains.xml

# Understanding versioning strategies