
Java is a trademark of Sun Microsystems, Inc.

JavaOne℠

# Getting Serious About Build Automation: Using Maven in the Real World

John Ferguson Smart
Wakaleo Consulting Ltd.

Email: john.smart@wakaleo.com
Twitter: wakaleo

Wakaleo Consulting
Optimizing your software development process

# Introduction

## Let's get serious about build automation!

> What we will cover today

# Introduction

## Let's get serious about build automation!

- What we will cover today
  - Get more out of your Maven builds!

# Introduction

## Let's get serious about build automation!

- What we will cover today
  - Get more out of your Maven builds!
    - ☑ Dependencies
      - Beyond the basics

# Introduction

## Let's get serious about build automation!

- What we will cover today
  - Get more out of your Maven builds!
    - ☑ Dependencies
      - Beyond the basics
    - ☑ Inheritance, aggregation and multi-module projects
      - More than just a pretty architecture

# Introduction

## Let's get serious about build automation!

- What we will cover today
  - Get more out of your Maven builds!
    - ☑ Dependencies
      - Beyond the basics
    - ☑ Inheritance, aggregation and multi-module projects
      - More than just a pretty architecture
    - ☑ Revving up your release process
      - Automating releases with Nexus, Hudson and the release plugin

# Dependencies, inheritance, and aggregation

## Fitting it all together

- Maven encourages modular design

# Dependencies, inheritance, and aggregation

## Fitting it all together

- Maven encourages modular design
  - Dependencies

# Dependencies, inheritance, and aggregation

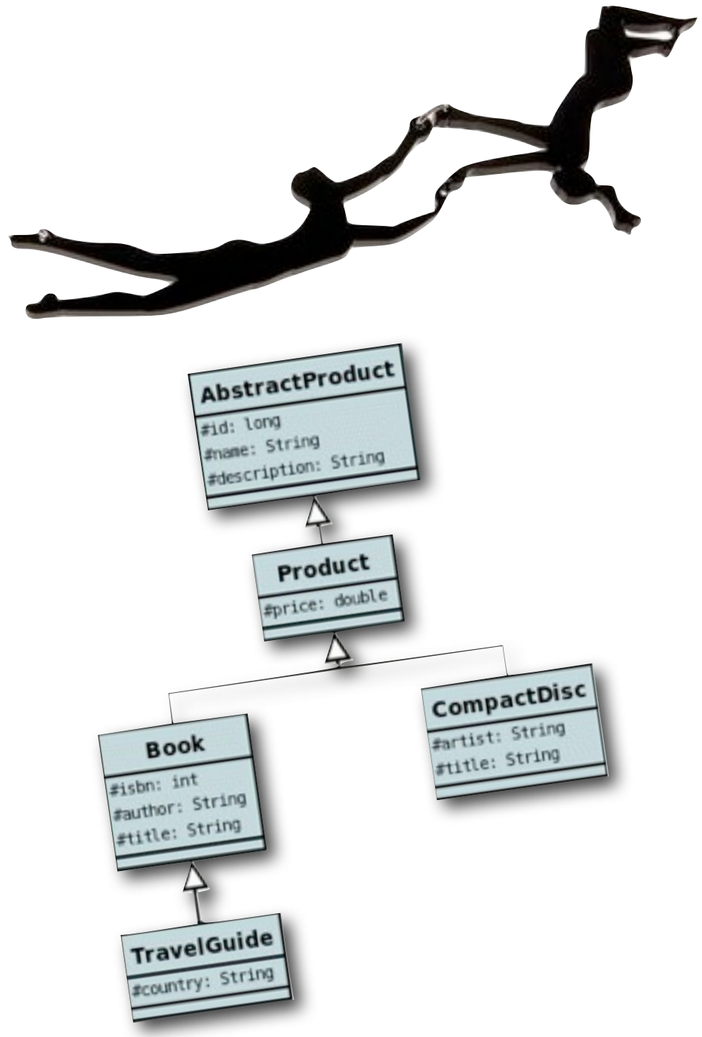
## Fitting it all together

- Maven encourages modular design
  - Dependencies
    - Define the relationships between modules

# Dependencies, inheritance, and aggregation

## Fitting it all together

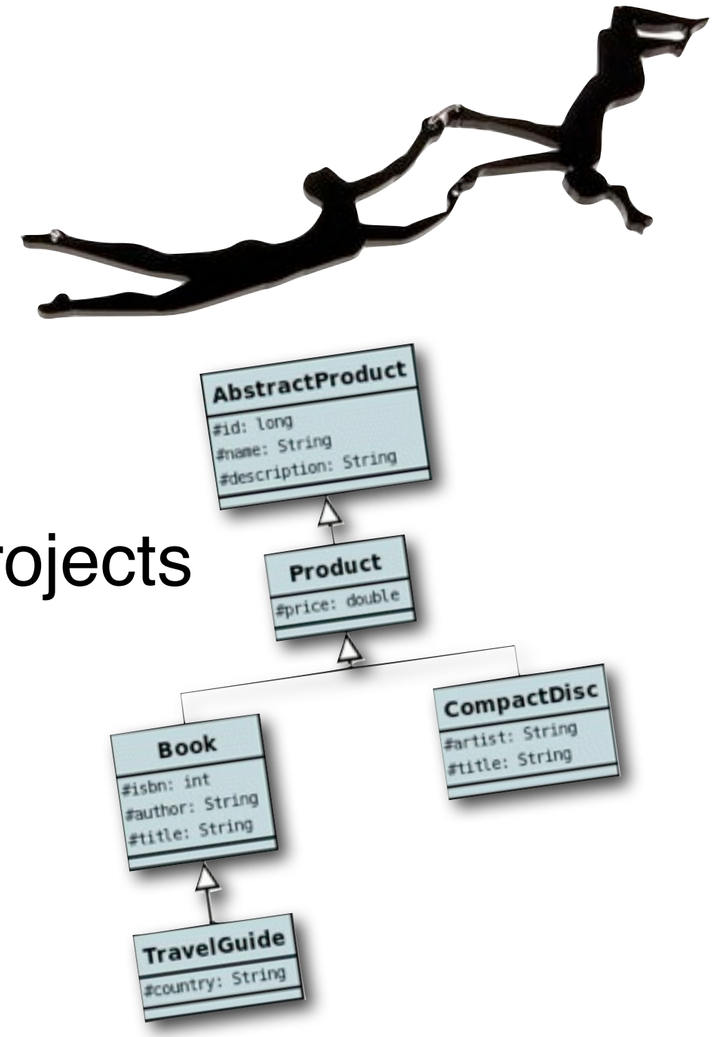
- Maven encourages modular design
  - Dependencies
    - Define the relationships between modules
  - Inheritance



# Dependencies, inheritance, and aggregation

## Fitting it all together

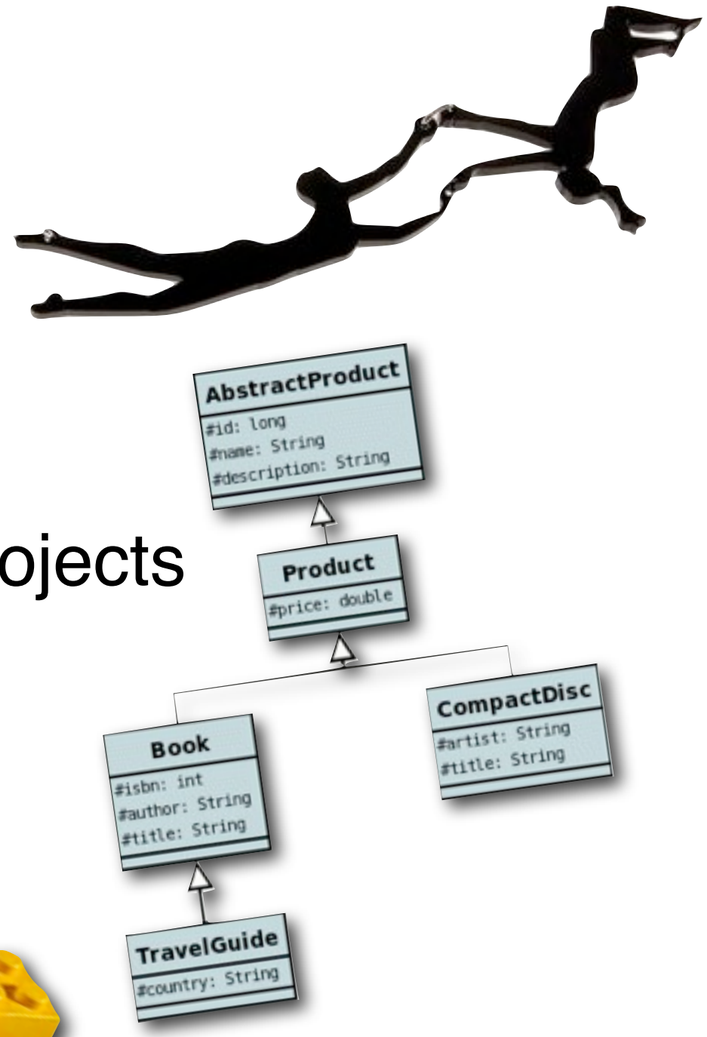
- Maven encourages modular design
  - Dependencies
    - Define the relationships between modules
  - Inheritance
    - Inherit properties and behavior from parent projects



# Dependencies, inheritance, and aggregation

## Fitting it all together

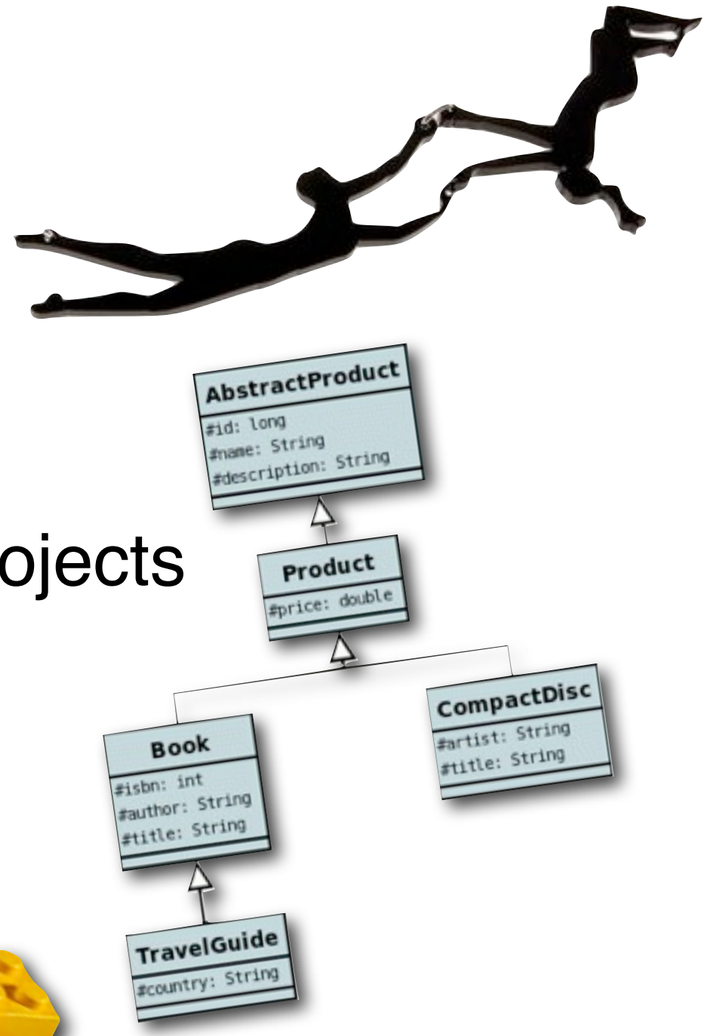
- Maven encourages modular design
  - Dependencies
    - Define the relationships between modules
  - Inheritance
    - Inherit properties and behavior from parent projects
  - Aggregation




# Dependencies, inheritance, and aggregation

## Fitting it all together

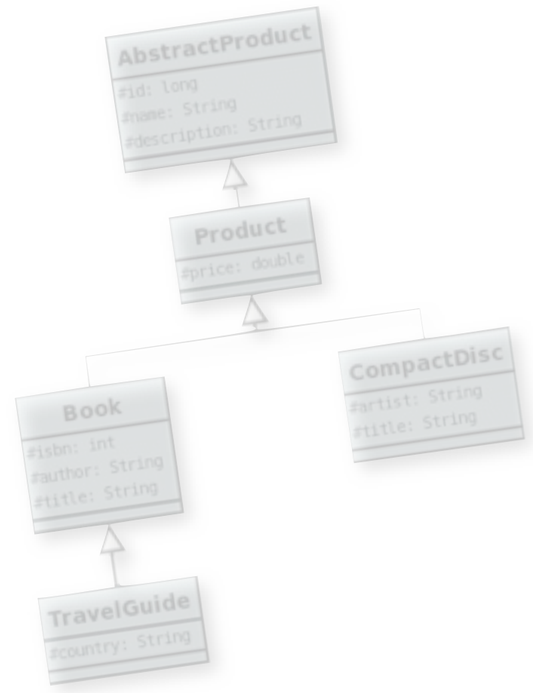
- Maven encourages modular design
  - Dependencies
    - Define the relationships between modules
  - Inheritance
    - Inherit properties and behavior from parent projects
  - Aggregation
    - Organize builds with multi-module projects




# Dependency Management

## If you only use one Maven feature, use this one!

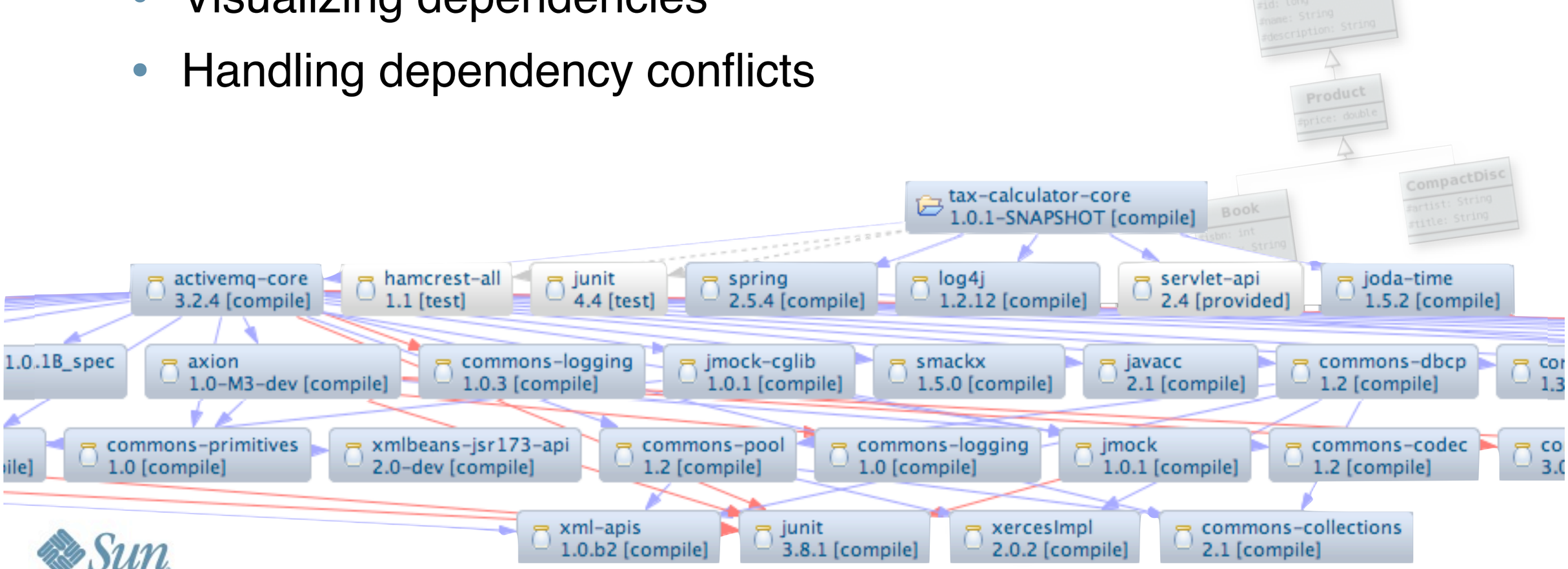
- Advanced Dependency Management in Maven
  - It's easier with Eclipse...
    - Adding new dependencies
    - Visualizing dependencies
  - Handling dependency conflicts



# Dependency Management

## If you only use one Maven feature, use this one!

- Advanced Dependency Management in Maven
  - It's easier with Eclipse...
    - Adding new dependencies
    - Visualizing dependencies
  - Handling dependency conflicts

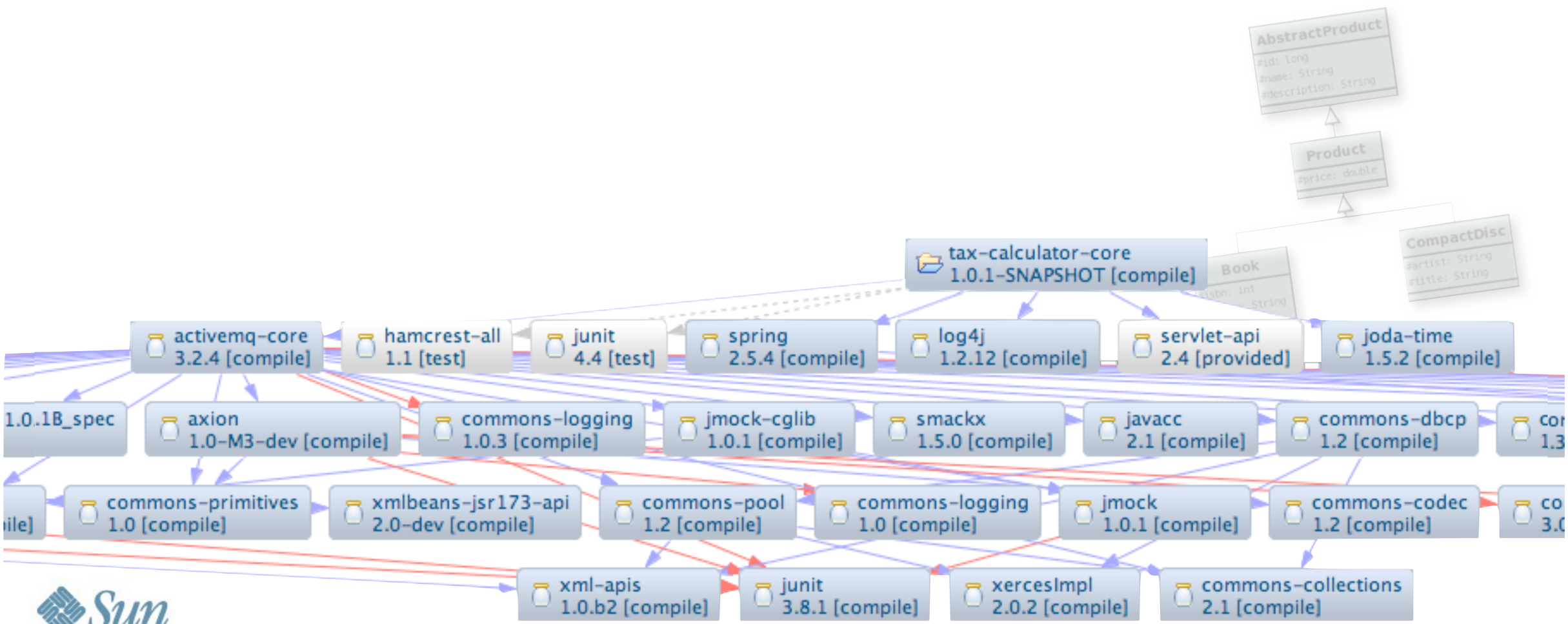



Sun microsystems

# Dependency Management

## If you only use one Maven feature, use this one!

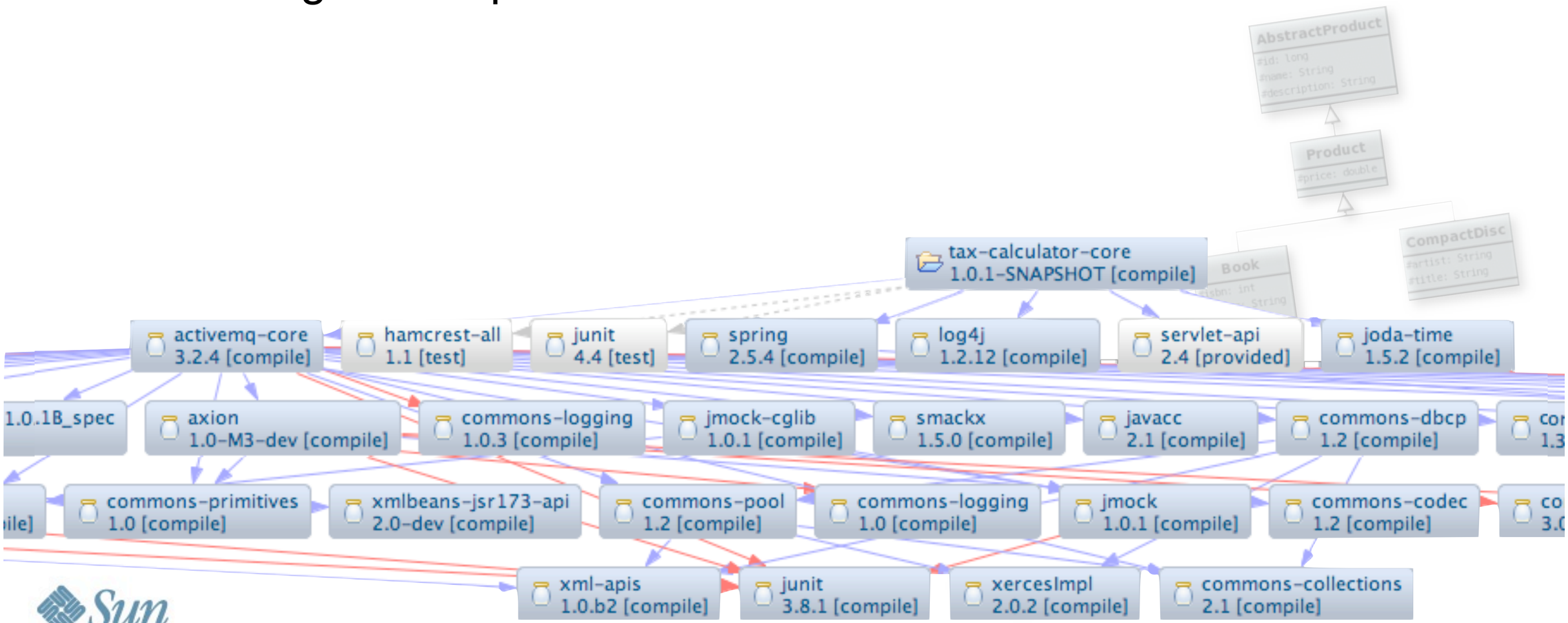
- Advanced Dependency Management in Maven
  - It's easier with Eclipse...



# Dependency Management

## If you only use one Maven feature, use this one!

- Advanced Dependency Management in Maven
  - It's easier with Eclipse...
    - Adding new dependencies

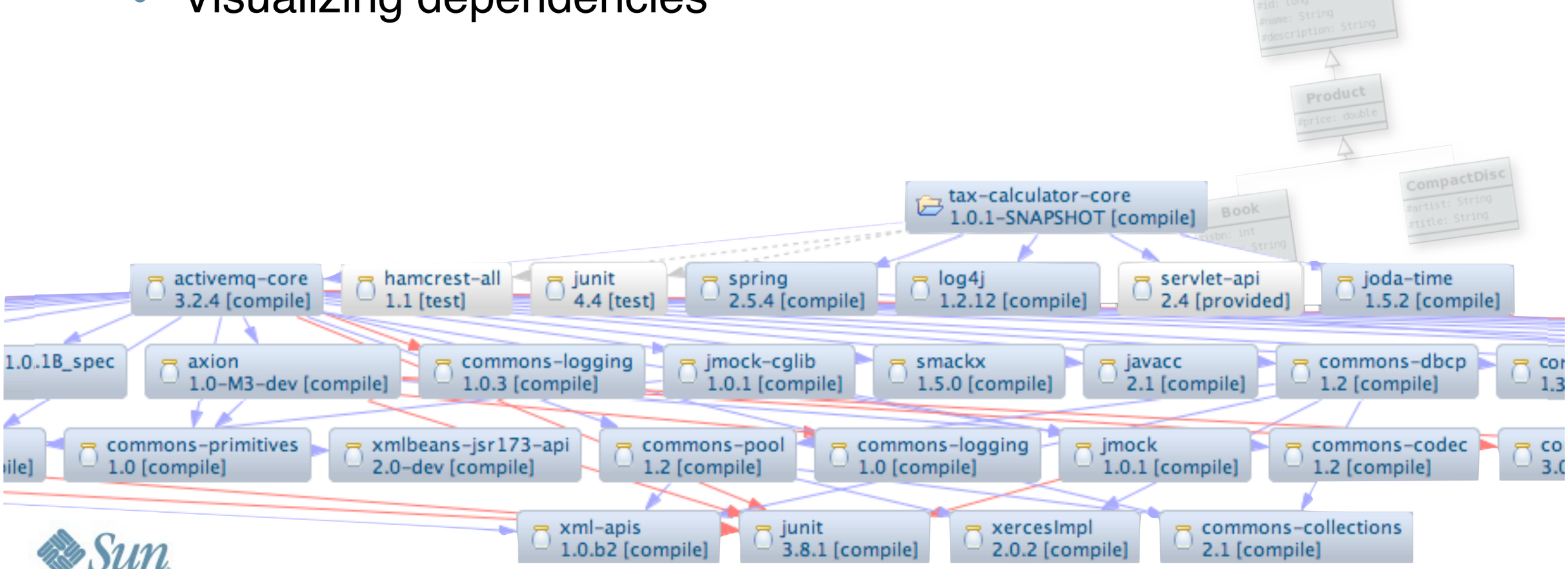



Sun microsystems

# Dependency Management

## If you only use one Maven feature, use this one!

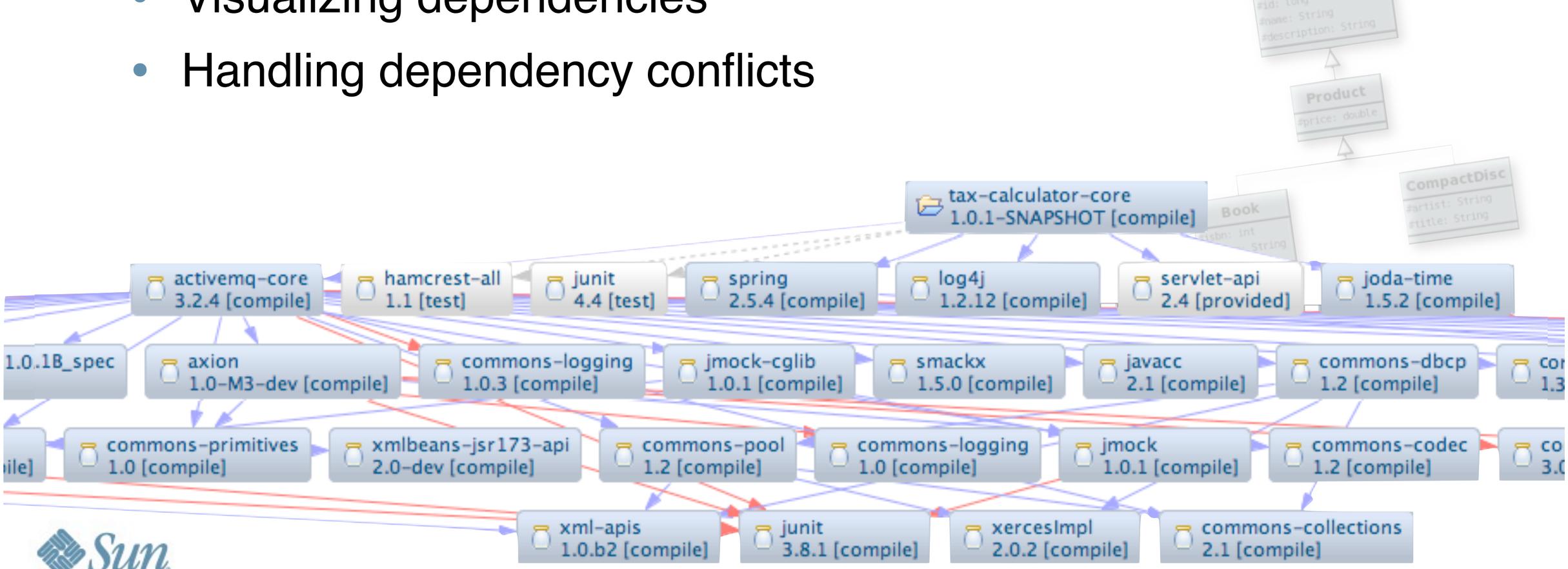
- Advanced Dependency Management in Maven
  - It's easier with Eclipse...
    - Adding new dependencies
    - Visualizing dependencies



# Dependency Management

## If you only use one Maven feature, use this one!

- Advanced Dependency Management in Maven
  - It's easier with Eclipse...
    - Adding new dependencies
    - Visualizing dependencies
  - Handling dependency conflicts




Sun microsystems

# Managing Dependencies in Eclipse

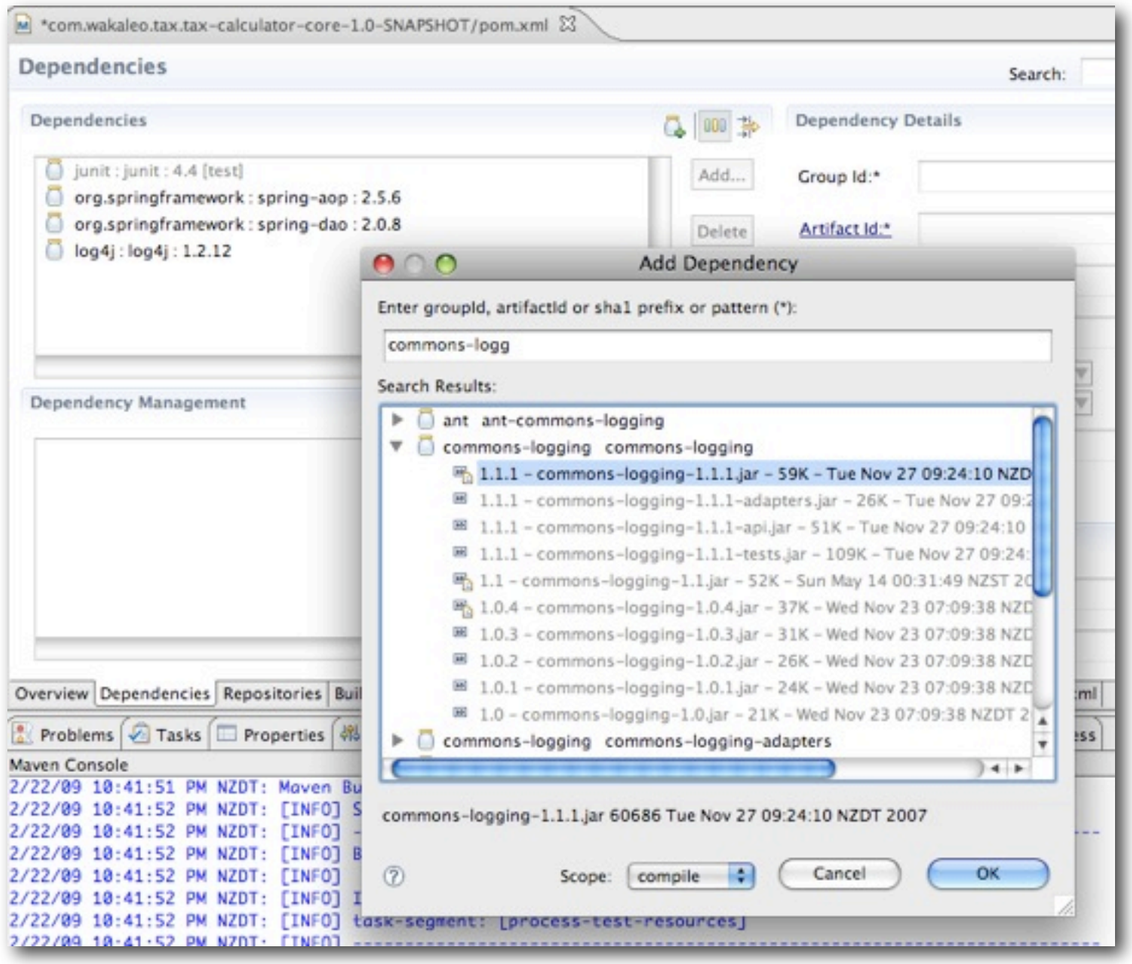
## The m2eclipse plugin

- Dependencies in Eclipse with m2eclipse

# Managing Dependencies in Eclipse

## The m2eclipse plugin

- Dependencies in Eclipse with m2eclipse
  - Adding new dependencies



# Managing Dependencies in Eclipse

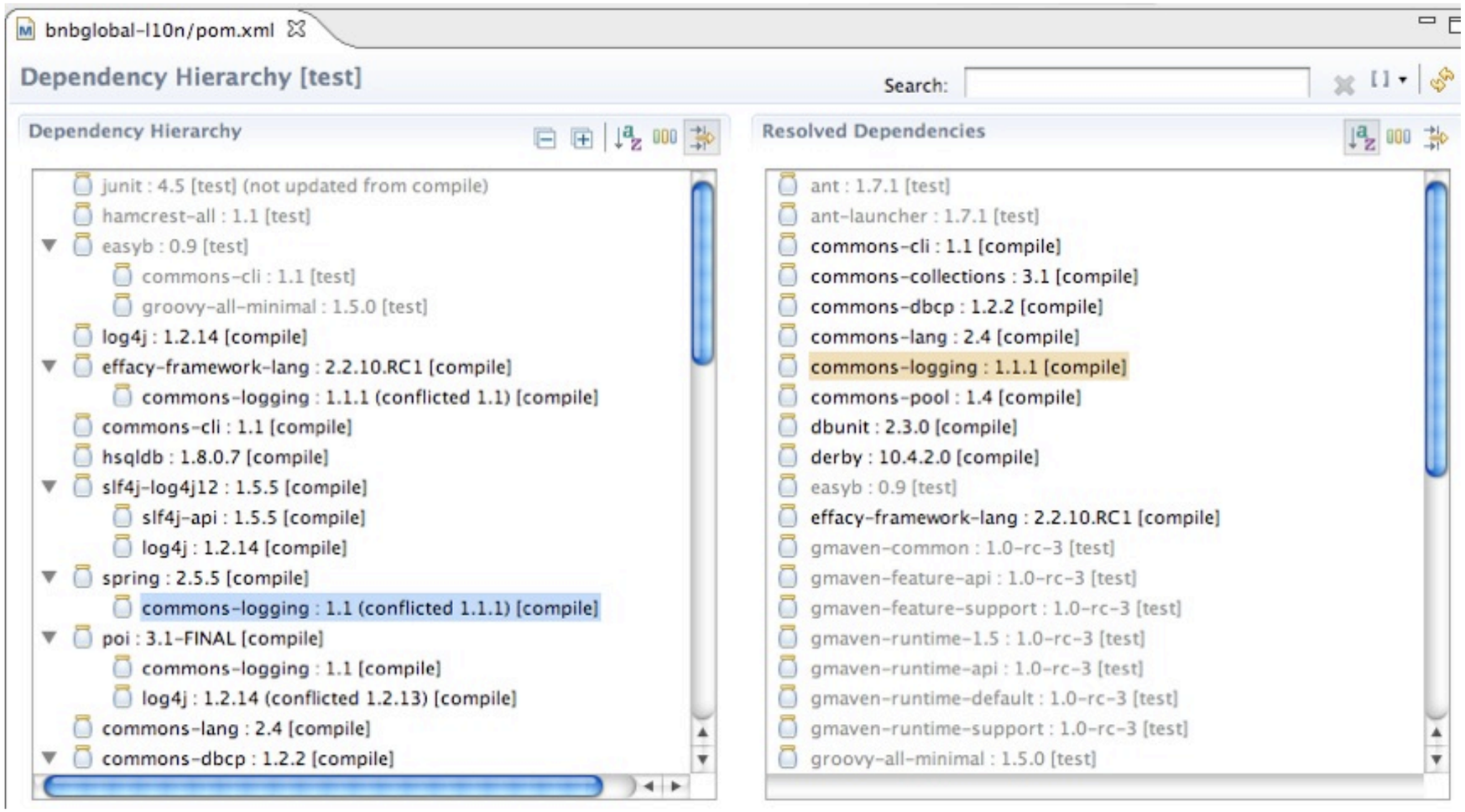
## The m2eclipse plugin

- Dependencies in Eclipse with m2eclipse

# Managing Dependencies in Eclipse

## The m2eclipse plugin

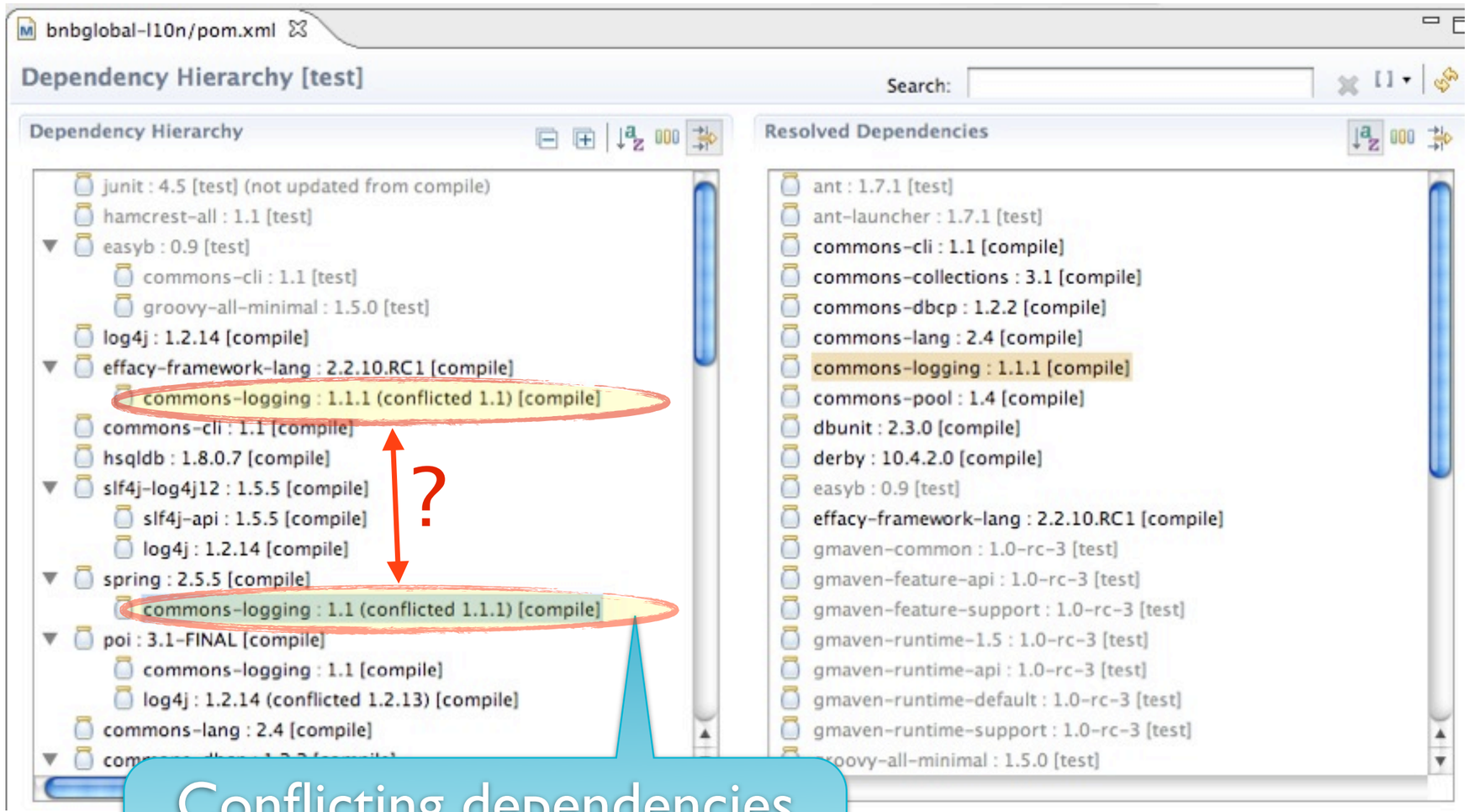
- Dependencies in Eclipse with m2eclipse
  - Visualizing the Dependency Hierarchy



# Managing Dependencies in Eclipse

## The m2eclipse plugin

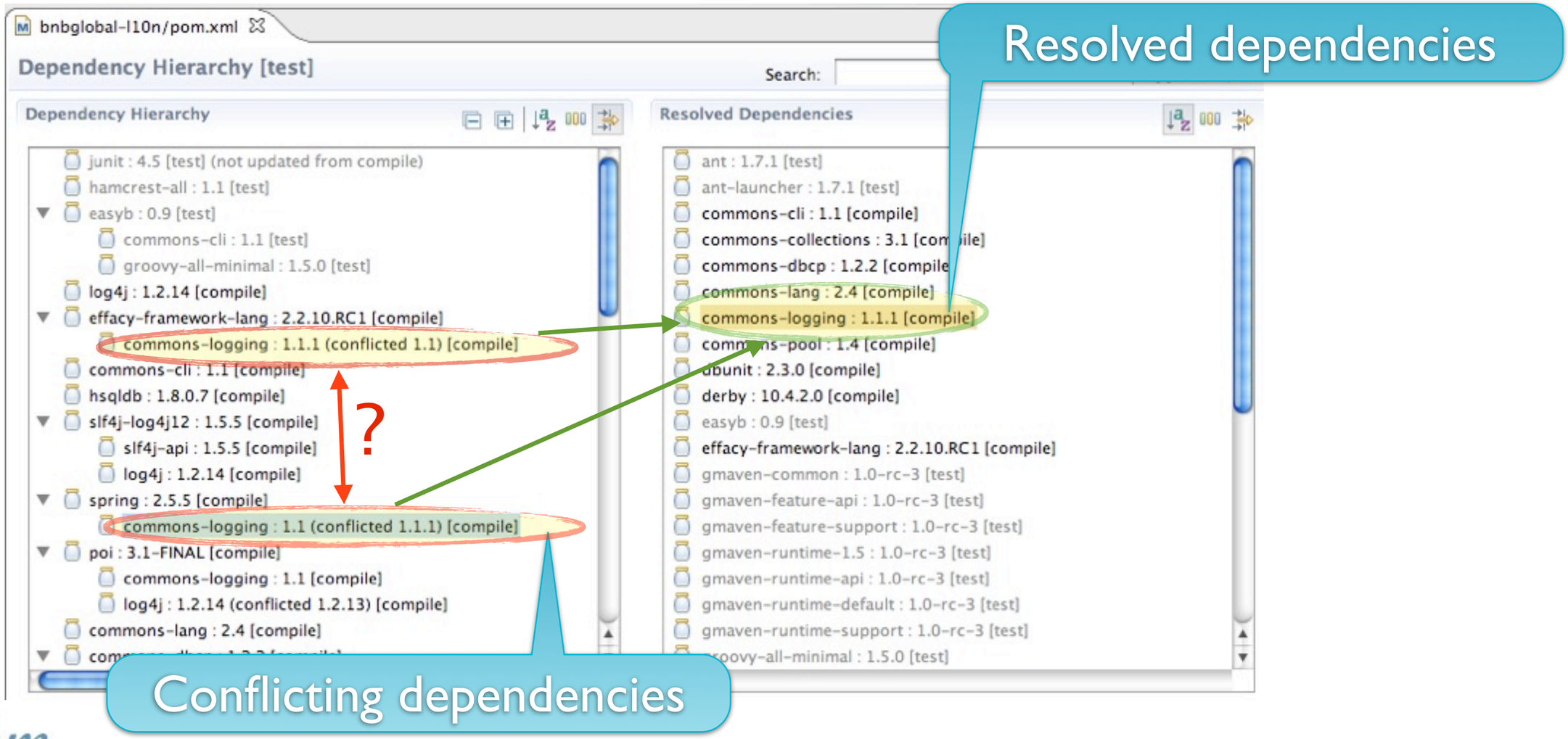
- Dependencies in Eclipse with m2eclipse
  - Visualizing the Dependency Hierarchy



# Managing Dependencies in Eclipse

## The m2eclipse plugin

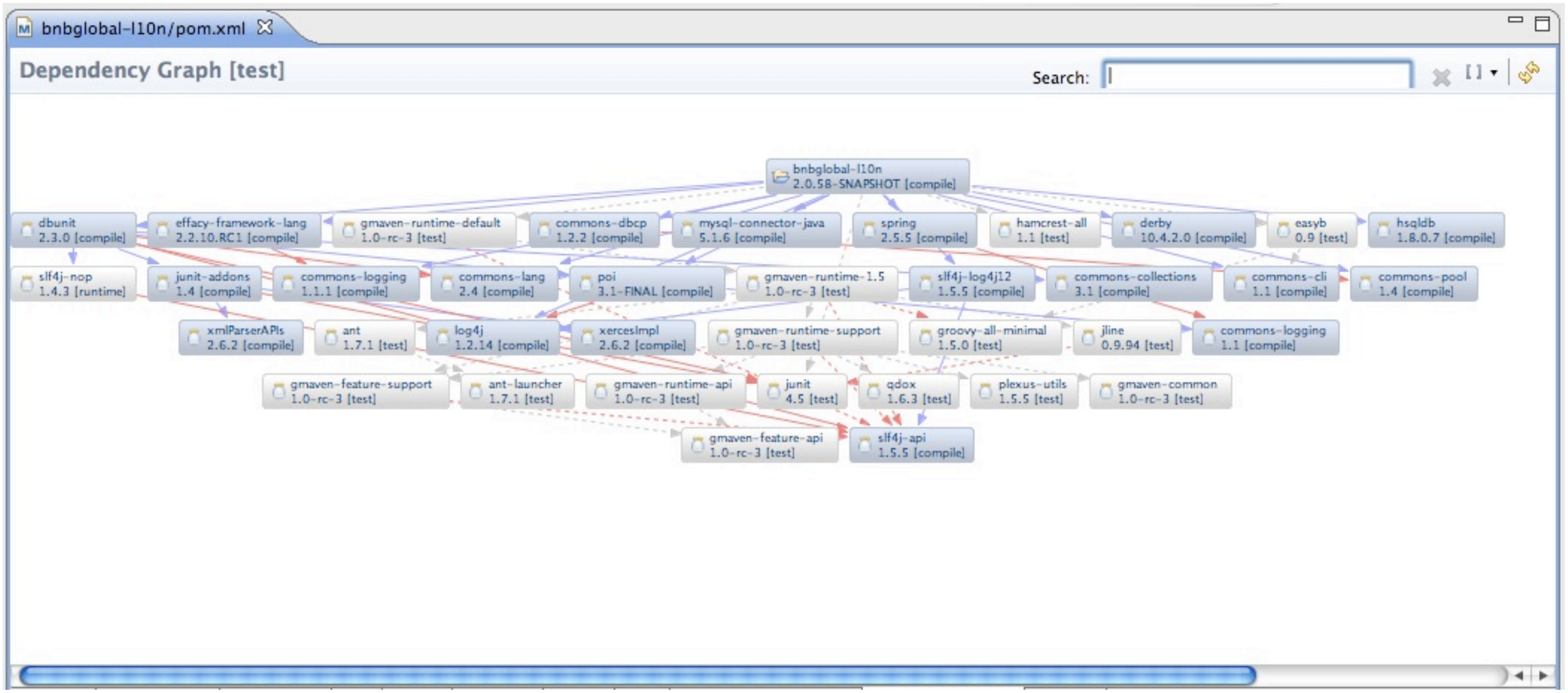
- Dependencies in Eclipse with m2eclipse
  - Visualizing the Dependency Hierarchy



# Managing Dependencies in Eclipse

## The m2eclipse plugin

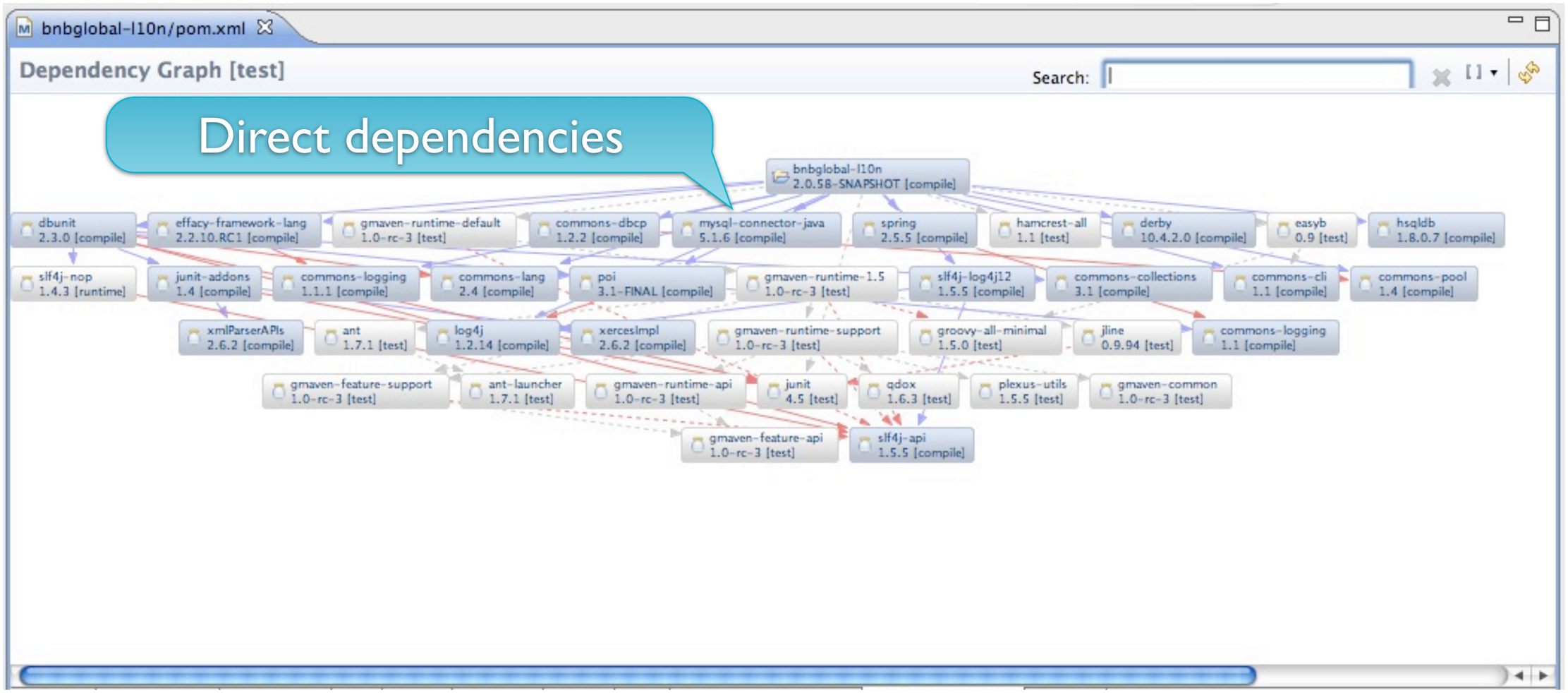
- Dependencies in Eclipse with m2eclipse
  - Visualizing the Dependency Graph



# Managing Dependencies in Eclipse

## The m2eclipse plugin

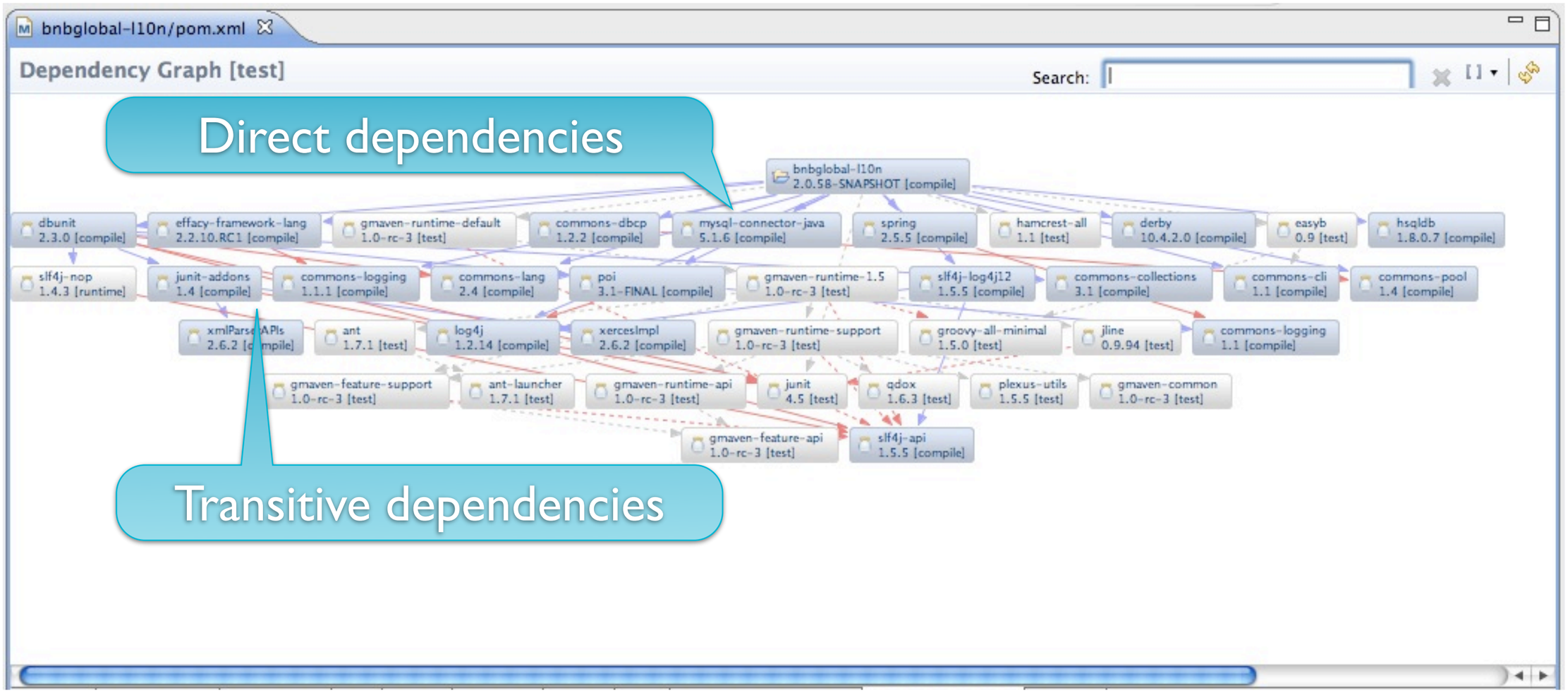
- Dependencies in Eclipse with m2eclipse
  - Visualizing the Dependency Graph



# Managing Dependencies in Eclipse

## The m2eclipse plugin

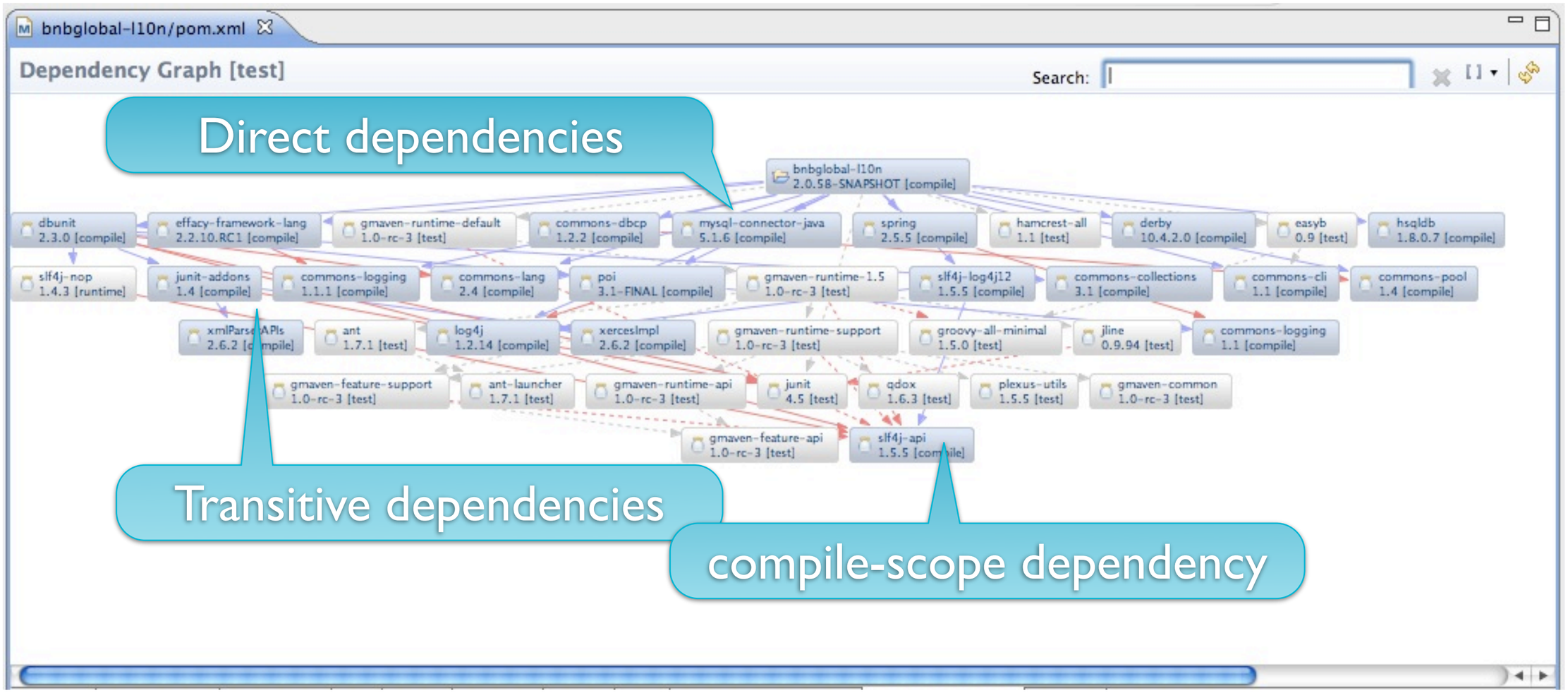
- Dependencies in Eclipse with m2eclipse
  - Visualizing the Dependency Graph



# Managing Dependencies in Eclipse

## The m2eclipse plugin

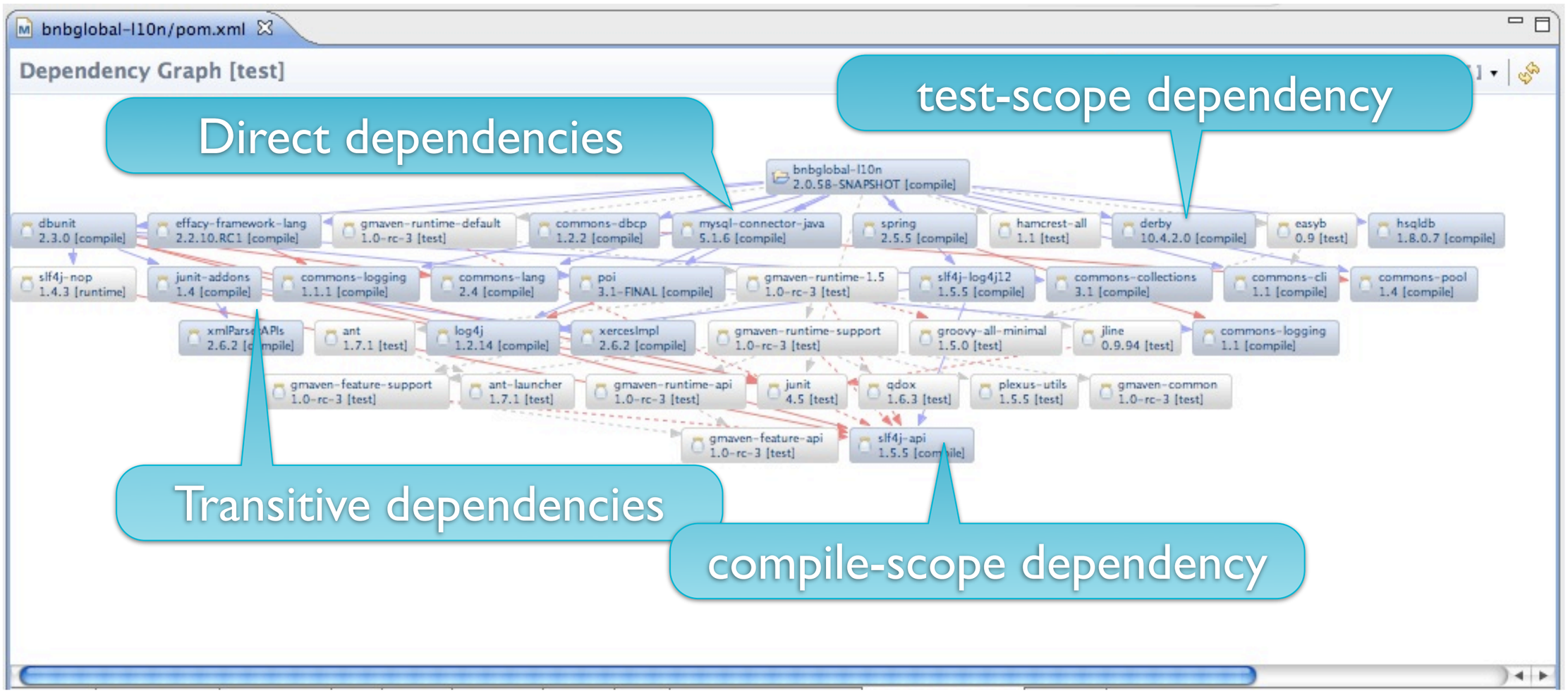
- Dependencies in Eclipse with m2eclipse
  - Visualizing the Dependency Graph



# Managing Dependencies in Eclipse

## The m2eclipse plugin

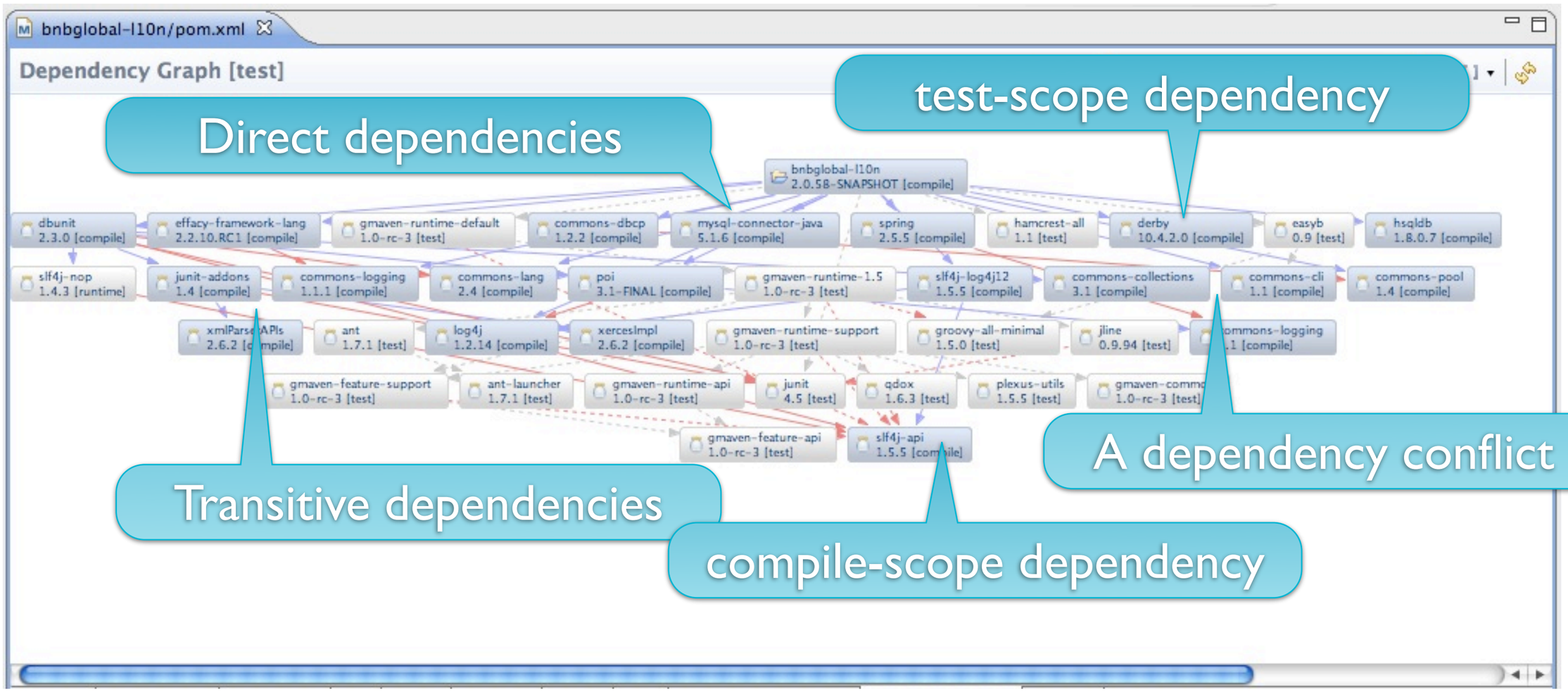
- Dependencies in Eclipse with m2eclipse
  - Visualizing the Dependency Graph



# Managing Dependencies in Eclipse

## The m2eclipse plugin

- Dependencies in Eclipse with m2eclipse
  - Visualizing the Dependency Graph



# Managing Dependency Conflicts

## When transitive dependencies aren't what you want

- Dependency Conflicts Happen

# Managing Dependency Conflicts

## When transitive dependencies aren't what you want

- Dependency Conflicts Happen
  - Maven's Golden Rule of Transitive Dependencies

# Managing Dependency Conflicts

## When transitive dependencies aren't what you want

- Dependency Conflicts Happen
  - Maven's Golden Rule of Transitive Dependencies
    - The closest dependency to your project wins

# Managing Dependency Conflicts

## When transitive dependencies aren't what you want

- Dependency Conflicts Happen
  - Maven's Golden Rule of Transitive Dependencies
    - The closest dependency to your project wins
  - What can go wrong?

# Managing Dependency Conflicts

## When transitive dependencies aren't what you want

- Dependency Conflicts Happen
  - Maven's Golden Rule of Transitive Dependencies
    - The closest dependency to your project wins
  - What can go wrong?
    - This dependency is not the one you want

# Managing Dependency Conflicts

## When transitive dependencies aren't what you want

- Dependency Conflicts Happen
  - Maven's Golden Rule of Transitive Dependencies
    - The closest dependency to your project wins
  - What can go wrong?
    - This dependency is not the one you want
  - How can I fix it

# Managing Dependency Conflicts

## When transitive dependencies aren't what you want

- Dependency Conflicts Happen
  - Maven's Golden Rule of Transitive Dependencies
    - The closest dependency to your project wins
  - What can go wrong?
    - This dependency is not the one you want
  - How can I fix it
    - Declare your own in your pom.xml file (or in a parent pom file)

# Managing Dependency Conflicts

## When transitive dependencies aren't what you want

- Dependency Conflicts Happen
  - Maven's Golden Rule of Transitive Dependencies
    - The closest dependency to your project wins
  - What can go wrong?
    - This dependency is not the one you want
  - How can I fix it
    - Declare your own in your pom.xml file (or in a parent pom file)
    - Exclude the dependency you don't want

# Managing Dependency Conflicts

## Example: excluding a dependency

- Excluding dependencies

# Managing Dependency Conflicts

## Example: excluding a dependency

- Excluding dependencies

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring</artifactId>
    <version>2.5.5</version>
    <exclusions>
     <exclusion>
      <groupId>javax.jms</groupId>
      <artifactId>jms<artifactId>
    </exclusion>
   </exclusions>
  </dependency>
  <dependency>
    <groupId>org.apache.geronimo.specs</groupId>
    <artifactId>geronimo-jms_1.1_spec</artifact>
    <version>1.1</version>
  </dependency>
<dependencies>
```

# Managing Dependency Conflicts

## Example: excluding a dependency

- Excluding dependencies

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring</artifactId>
    <version>2.5.5</version>
    <exclusions>
     <exclusion>
      <groupId>javax.jms</groupId>
      <artifactId>jms<artifactId>
     </exclusion>
    </exclusions>
  </dependency>
  <dependency>
    <groupId>org.apache.geronimo.specs</groupId>
    <artifactId>geronimo-jms_1.1_spec</artifact>
    <version>1.1</version>
  </dependency>
<dependencies>
```

Exclude the default JMS library

# Managing Dependency Conflicts

## Example: excluding a dependency

- Excluding dependencies

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring</artifactId>
    <version>2.5.5</version>
    <exclusions>
     <exclusion>
      <groupId>javax.jms</groupId>
      <artifactId>jms<artifactId>
     </exclusion>
    </exclusions>
  </dependency>
  <dependency>
    <groupId>org.apache.geronimo.specs</groupId>
    <artifactId>geronimo-jms_1.1_spec</artifact>
    <version>1.1</version>
  </dependency>
<dependencies>
```

Exclude the default JMS library

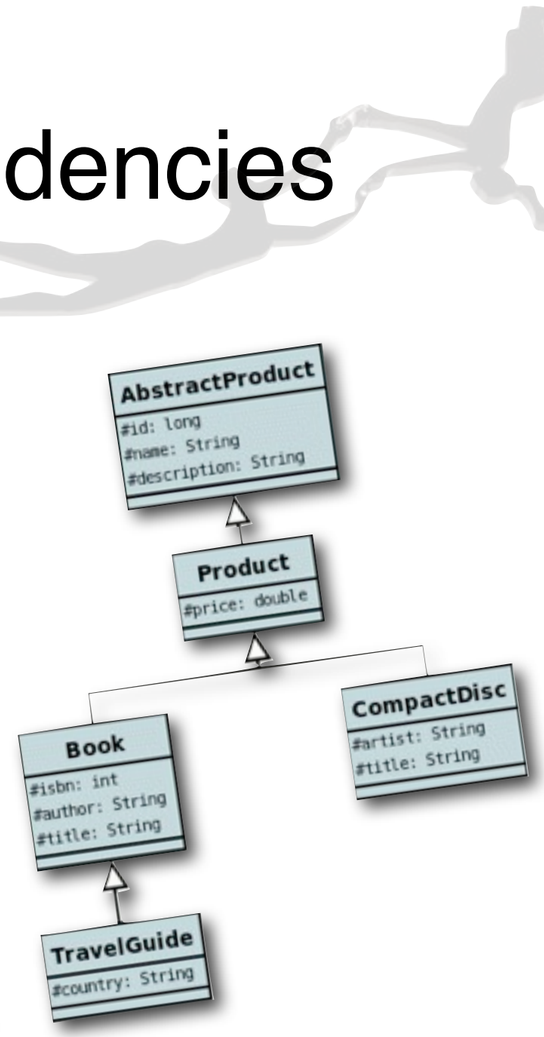
Use the Geronimo implementation instead

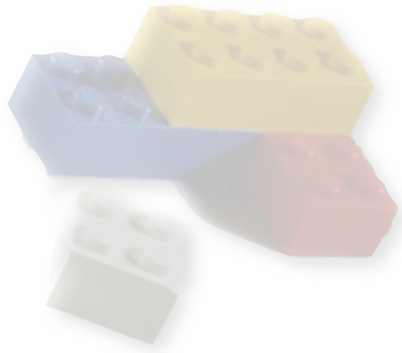
# Time for a demo!

# Project Inheritance

## Refactoring your builds

- Why use project inheritance?
  - Centralize shared behavior and dependencies
  - Centralize reporting
  - Simplify your pom files
  - Centralize pom file maintenance
  - Define organization-wide standards

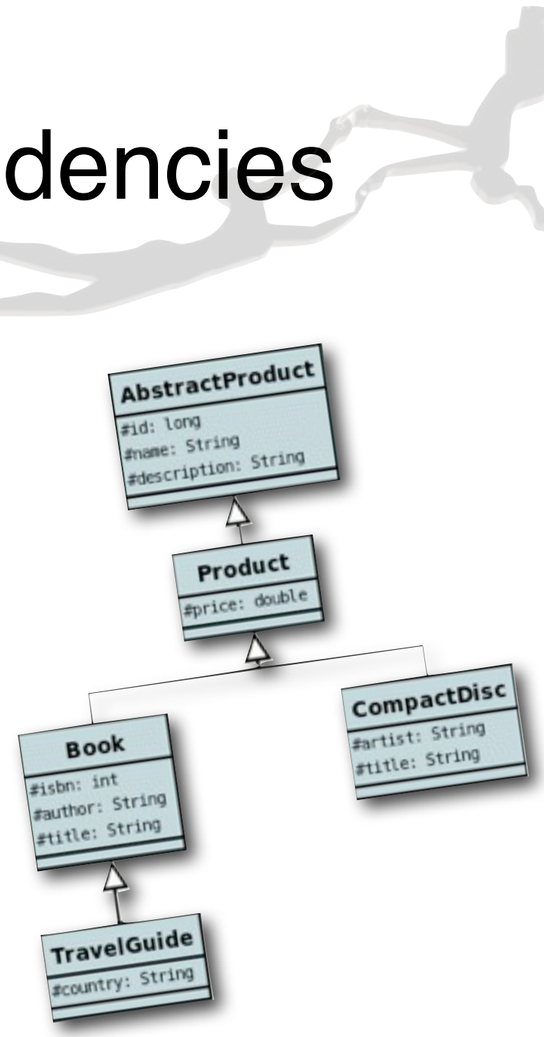


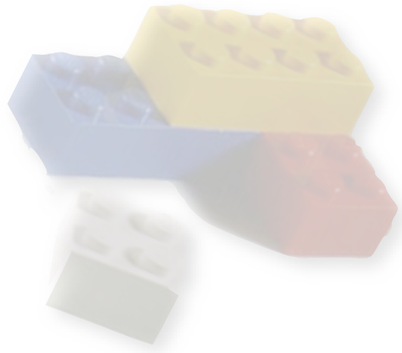



# Project Inheritance

## Refactoring your builds

- Why use project inheritance?
  - Centralize shared behavior and dependencies
  - Centralize reporting
  - Simplify your pom files
  - Centralize pom file maintenance
  - Define organization-wide standards

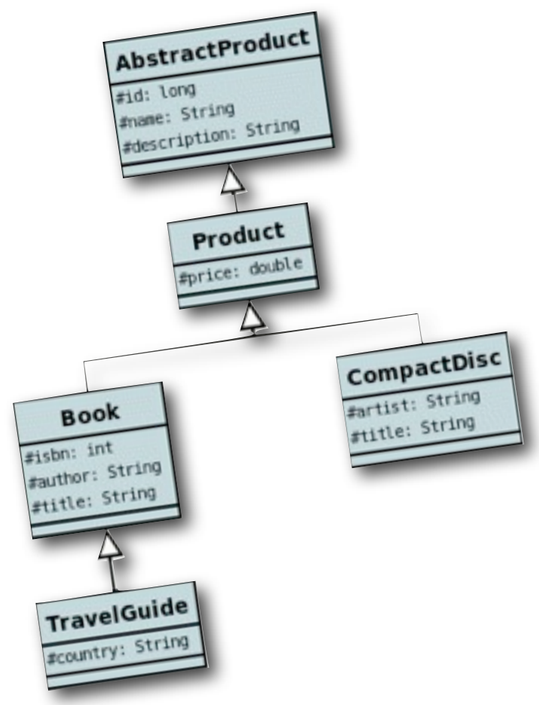


# Project Inheritance

## Refactoring your builds

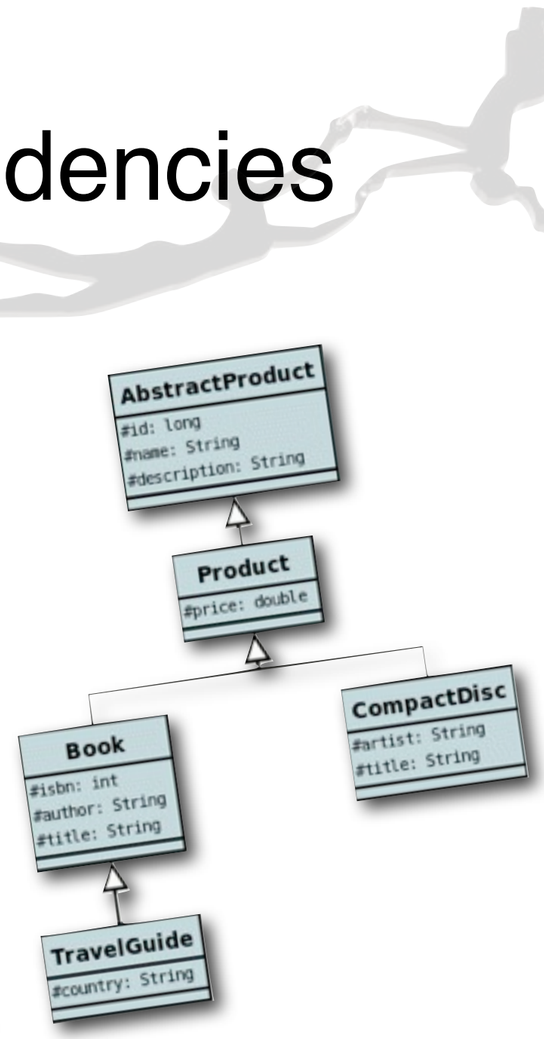
- Why use project inheritance?
  - Centralize shared behavior and dependencies



# Project Inheritance

## Refactoring your builds

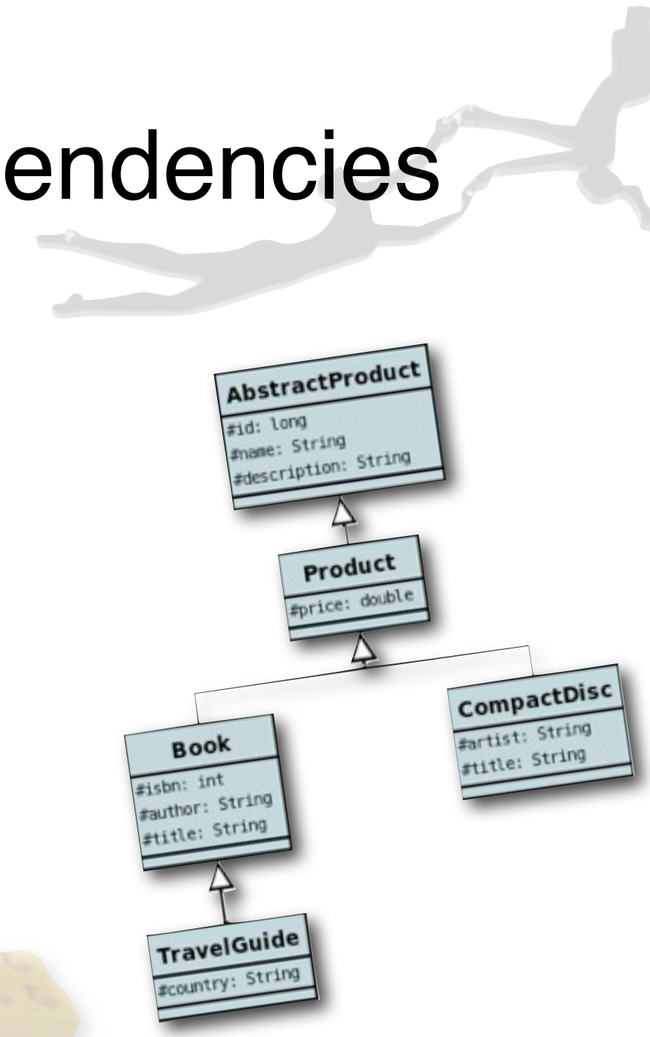
- Why use project inheritance?
  - Centralize shared behavior and dependencies
  - Centralize reporting

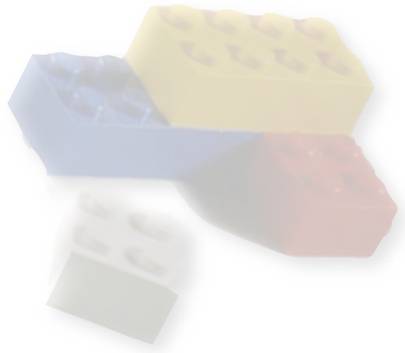


# Project Inheritance

## Refactoring your builds

- Why use project inheritance?
  - Centralize shared behavior and dependencies
  - Centralize reporting
  - Simplify your pom files

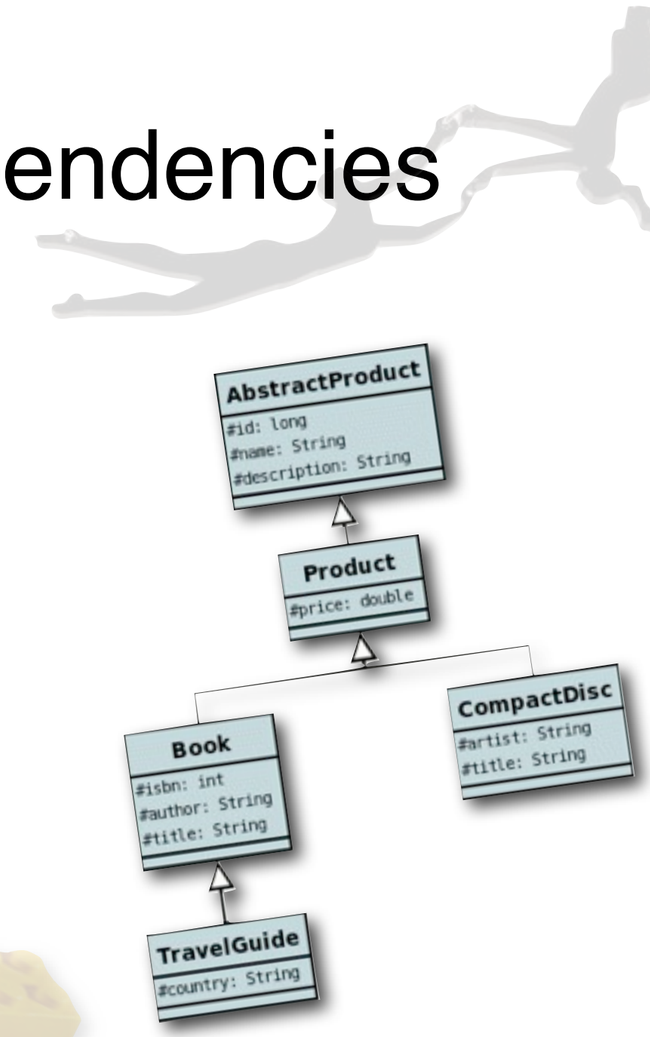


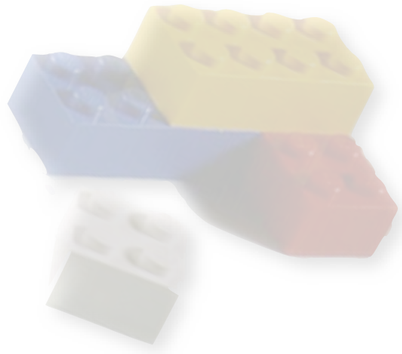



# Project Inheritance

## Refactoring your builds

- Why use project inheritance?
  - Centralize shared behavior and dependencies
  - Centralize reporting
  - Simplify your pom files
  - Centralize pom file maintenance

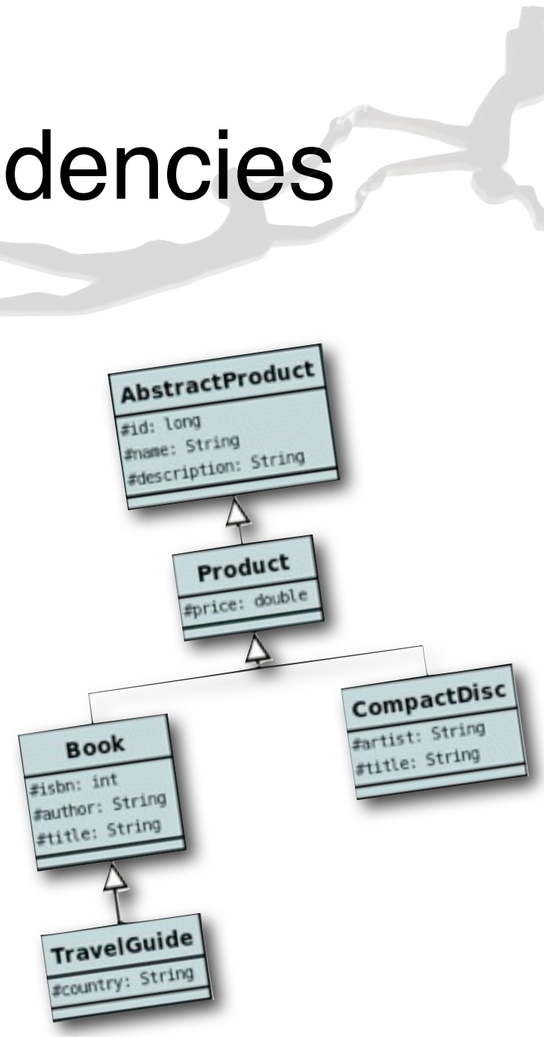


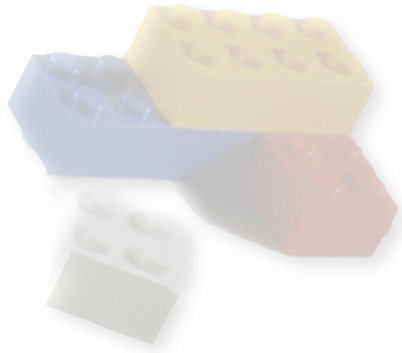



# Project Inheritance

## Refactoring your builds

- Why use project inheritance?
  - Centralize shared behavior and dependencies
  - Centralize reporting
  - Simplify your pom files
  - Centralize pom file maintenance
  - Define organization-wide standards



# Project Inheritance

## A sample project structure

- Inheritance in action

# Project Inheritance

## A sample project structure

- Inheritance in action

# Project Inheritance

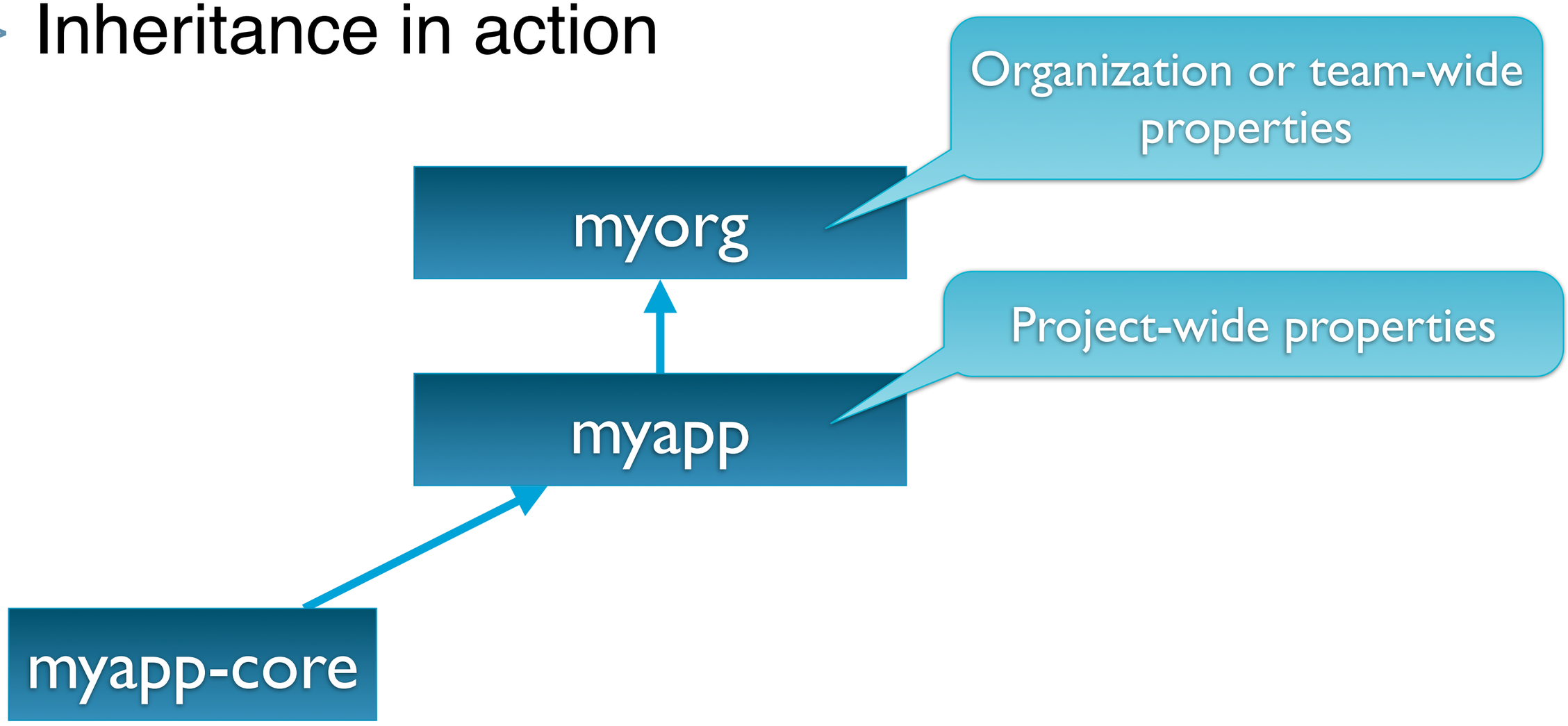
## A sample project structure

- Inheritance in action

# Project Inheritance

## A sample project structure

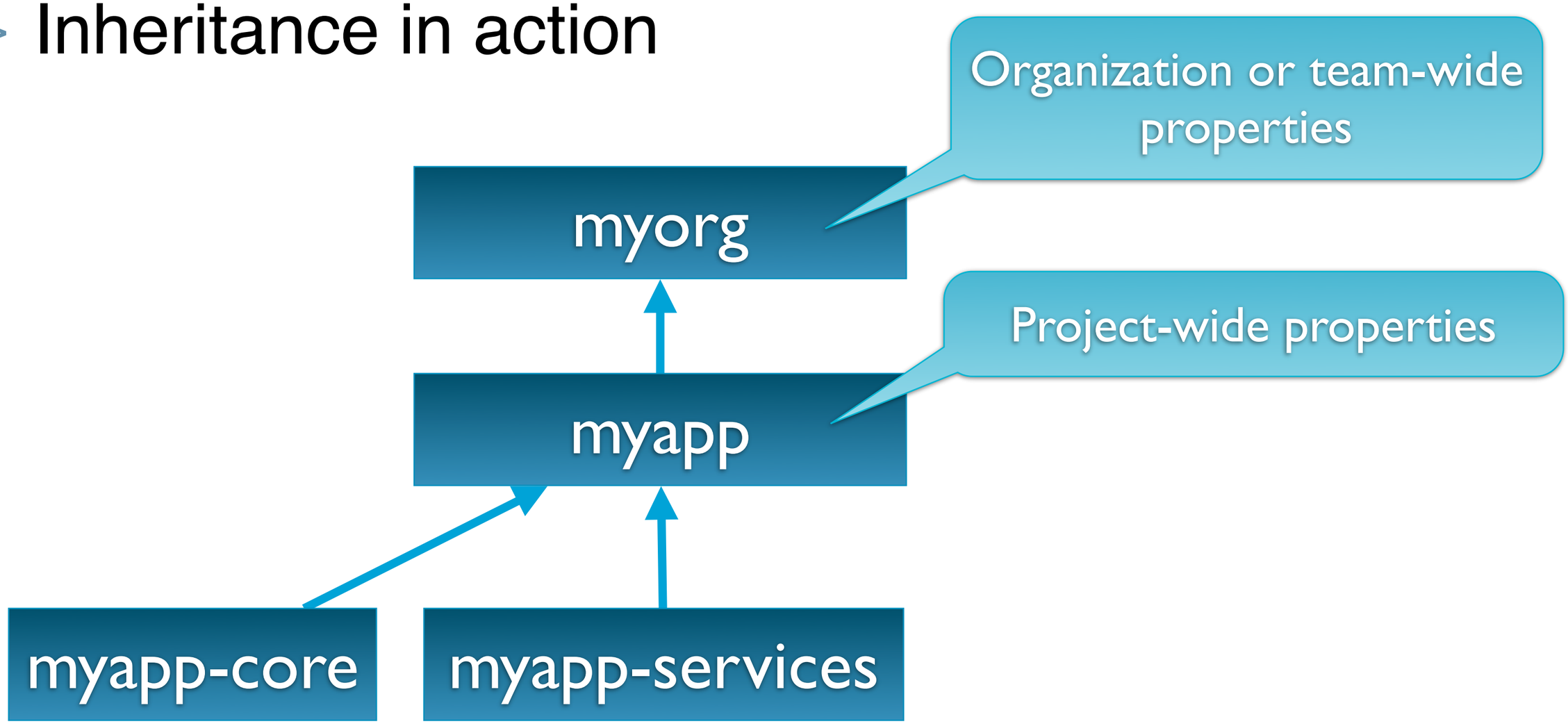
- Inheritance in action



# Project Inheritance

## A sample project structure

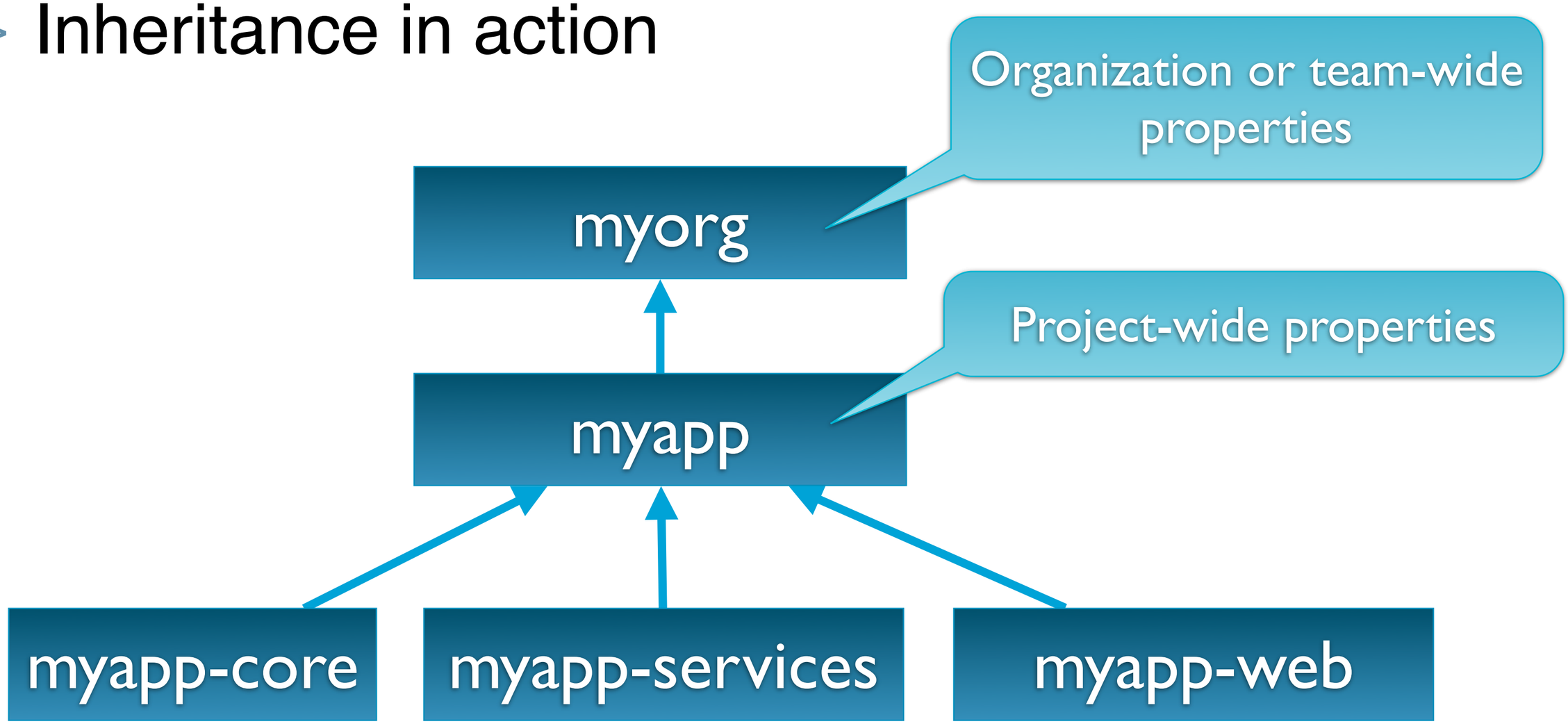
- Inheritance in action



# Project Inheritance

## A sample project structure

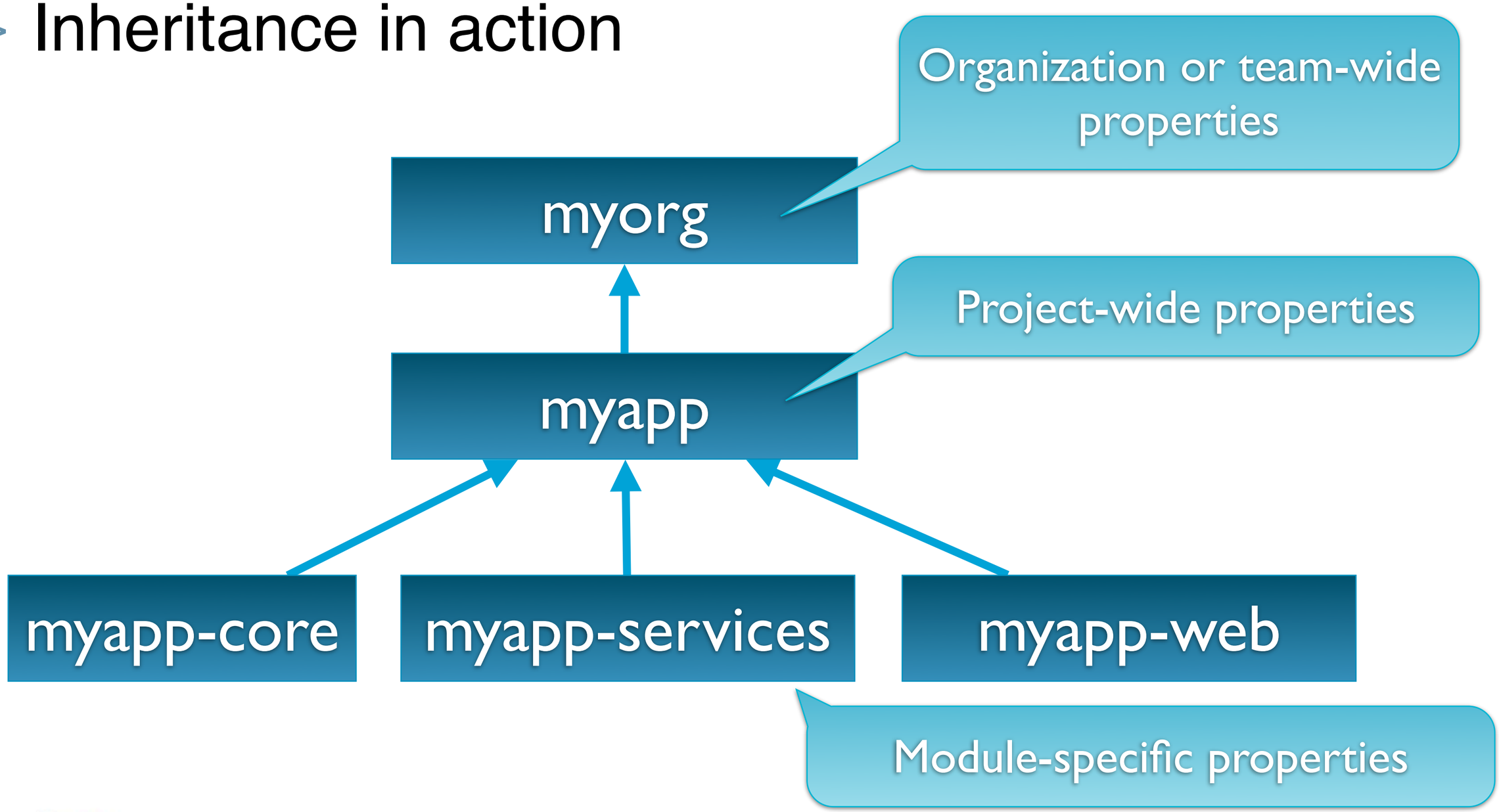
- Inheritance in action



# Project Inheritance

## A sample project structure

- Inheritance in action



# Project Inheritance

## Refactoring common dependencies

- Refactoring your dependencies

# Project Inheritance

## Refactoring common dependencies

- Refactoring your dependencies

```
<dependencies>
   <dependency...>
   <dependency...>
</dependencies>
```

Parent pom.xml

Mandatory dependencies

# Project Inheritance

## Refactoring common dependencies

- Refactoring your dependencies

```
<dependencies>
   <dependency...>
   <dependency...>
</dependencies>
```

Parent pom.xml

Mandatory dependencies

```
<dependencyManagement>
   <dependencies>
      <dependency...>
      <dependency...>
   </dependencies>
</dependencyManagement>
```

Parent pom.xml

Optional dependencies

# Project Inheritance

## Refactoring common dependencies

- Share dependencies across all projects

# Project Inheritance

## Refactoring common dependencies

- Share dependencies across all projects

Parent pom.xml

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.5</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

# Project Inheritance

## Refactoring common dependencies

- Share dependencies across all projects

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.5</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Parent pom.xml

Will be inherited by all child projects

# Project Inheritance

## Refactoring common dependencies

- Dependencies that are only used in some projects

# Project Inheritance

## Refactoring common dependencies

- Dependencies that are only used in some projects

Parent pom.xml

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.6</version>
    </dependency>
    <dependency>
      <groupId>postgres</groupId>
      <artifactId>postgres</artifactId>
      <version>7.3.2</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```

# Project Inheritance

## Refactoring common dependencies

- Dependencies that are only used in some projects

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.6</version>
    </dependency>
    <dependency>
      <groupId>postgres</groupId>
      <artifactId>postgres</artifactId>
      <version>7.3.2</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Parent pom.xml

Dependencies here will not be automatically inherited

# Project Inheritance

## Refactoring common dependencies

- Dependencies that are only used in some projects

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.6</version>
    </dependency>
    <dependency>
      <groupId>postgres</groupId>
      <artifactId>postgres</artifactId>
      <version>7.3.2</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Parent pom.xml

Dependencies here will not be automatically inherited

Project-wide version number declared here

# Project Inheritance

## Refactoring common dependencies

- Dependencies that are only used in some projects

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.6</version>
    </dependency>
    <dependency>
      <groupId>postgres</groupId>
      <artifa
      <versi
    </depend
  </dependen
</dependency
```

Parent pom.xml

Dependencies here will not be automatically inherited

Project-wide version number declared here

```
<dependencies>
  <dependency>
    <groupId>mysql</groupId>
     <artifactId>mysql-connector-java</artifactId>
  </dependency>
</dependencies>
```

Child pom.xml

# Project Inheritance

## Refactoring common dependencies

- Dependencies that are only used in some projects

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.6</version>
    </dependency>
    <dependency>
      <groupId>postgres</groupId>
      <artifactId>...</artifactId>
      <version>...</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Parent pom.xml

Dependencies here will not be automatically inherited

Project-wide version number declared here

Child projects must declare dependencies they need

```
<dependencies>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
  </dependency>
</dependencies>
```

Child pom.xml

# Project Inheritance

## Refactoring common dependencies

- Dependencies that are only used in some projects

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.6</version>
    </dependency>
    <dependency>
      <groupId>postgres</groupId>
```

Parent pom.xml

Dependencies here will not be automatically inherited

Project-wide version number declared here

Child projects must declare dependencies they need

But no version number is required

```
<dependencies>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
  </dependency>
</dependencies>
```

Child pom.xml

# Project Inheritance

## Refactoring common dependencies

- Importing DependencyManagement dependencies

# Project Inheritance

## Refactoring common dependencies

- Importing DependencyManagement dependencies

```
<groupId>myorg</groupId>
<artifactId>database-libraries</artifactId>
<versionId>1.0.0</versionId>
<packaging>pom</packaging>
...
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.6</version>
    </dependency>
    <dependency>
      <groupId>postgres</groupId>
      <artifactId>postgres</artifactId>
      <version>7.4.3</version>
    </dependency>
    ...
```

# Project Inheritance

## Refactoring common dependencies

- Importing DependencyManagement dependencies

```
<groupId>myorg</groupId>
<artifactId>database-libraries</artifactId>
<versionId>1.0.0</versionId>
<packaging>pom</packaging>
...
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.6</version>
    </dependency>
    <dependency>
      <groupId>postgres</groupId>
      <artifactId>postgres</artifactId>
      <version>7.4.3</version>
    </dependency>
    ...
```

Common dependencies are refactored into a separate project

# Project Inheritance

## Refactoring common dependencies

- Importing DependencyManagement dependencies

```
<groupId>myorg</groupId>
<artifactId>database-libraries</artifactId>
<versionId>1.0.0</versionId>
<packaging>pom</packaging>
...
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.6</version>
    </dependency>
    <dependency>
      <groupId>postgres</groupId>
      <artifactId>postgres</artifactId>
      <version>7.4.3</version>
    </dependency>
    ...
```

Common dependencies are refactored into a separate project

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>myorg</groupId>
      <artifactId>database-libraries</artifactId>
      <version>1.0.0</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
```

# Project Inheritance

## Refactoring common dependencies

- Importing DependencyManagement dependencies

```
<groupId>myorg</groupId>
<artifactId>database-libraries</artifactId>
<versionId>1.0.0</versionId>
<packaging>pom</packaging>
...
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.6</version>
    </dependency>
    <dependency>
      <groupId>postgres</groupId>
      <artifactId>postgres</artifactId>
      <version>7.4.3</version>
    </dependency>
    ...
```

Common dependencies are refactored into a separate project

Use the "import" scope to import DependencyManagement dependencies

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>myorg</groupId>
      <artifactId>database-libraries</artifactId>
      <version>1.0.0</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
```

# Project Inheritance

## Refactoring common build behavior

# Project Inheritance

## Refactoring common build behavior

- Organizing your build configuration

# Project Inheritance

## Refactoring common build behavior

- Organizing your build configuration
  - Use the <*plugins*> section for *mandatory* build configurations

# Project Inheritance

## Refactoring common build behavior

- Organizing your build configuration
  - Use the <*plugins*> section for *mandatory* build configurations
  - Use the <*pluginsManagement*> section for *optional* build configurations

# Project Inheritance

## Refactoring common build behavior

- Organizing your build configuration
  - Use the <*plugins*> section for *mandatory* build configurations
  - Use the <*pluginsManagement*> section for *optional* build configurations
  - The <*pluginsManagement*> plugins work for default lifecycle plugins

# Project Inheritance

## Refactoring common build behavior

- Build configuration shared across modules

# Project Inheritance

## Refactoring common build behavior

- Build configuration shared across modules

Parent pom.xml

```
<plugins>
  <plugin>
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-enforcer-plugin</artifactId>
     <version>1.0-beta-1</version>
     <executions>
         ...
         <configuration>
           <rules>
             <requireMavenVersion>
               <version>2.1.0</version>
             </requireMavenVersion>
           </rules>
         </configuration>
         ...
  </plugin>
</plugins>
```

# Project Inheritance

## Refactoring common build behavior

- Build configuration shared across modules

Will be used in all child projects

Parent pom.xml

```
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-enforcer-plugin</artifactId>
    <version>1.0-beta-1</version>
    <executions>
      ...
      <configuration>
        <rules>
          <requireMavenVersion>
            <version>2.1.0</version>
          </requireMavenVersion>
        </rules>
      </configuration>
      ...
  </plugin>
</plugins>
```

# Project Inheritance

## Refactoring common build behavior

- Build configuration shared across modules

# Project Inheritance

## Refactoring common build behavior

- Build configuration shared across modules

Parent pom.xml

```
<pluginManagement>
  <plugins>
    <plugin>
      <groupId>org.mortbay.jetty</groupId>
      <artifactId>maven-jetty-plugin</artifactId>
      <version>6.1.10</version>
      <configuration>
          ...
      </configuration>
    </plugin>
  </plugins>
</pluginManagement>
```

# Project Inheritance

## Refactoring common build behavior

- Build configuration shared across modules

Parent pom.xml

```
<pluginManagement>
  <plugins>
    <plugin>
      <groupId>org.mortbay.jetty</groupId>
      <artifactId>maven-jetty-plugin</artifactId>
      <version>6.1.10</version>
      <configuration>
        ...
      </configuration>
    </plugin>
  </plugins>
</pluginManagement>
```

Jetty configuration for all project modules

# Project Inheritance

## Refactoring common build behavior

- Build configuration shared across modules

Parent pom.xml

```
<pluginManagement>
  <plugins>
    <plugin>
      <groupId>org.mortbay.jetty</groupId>
      <artifactId>maven-jetty-plugin</artifactId>
      <version>6.1.10</version>
      <configuration>
        ...
      </configuration>
    </plugin>
  </plugins>
</pluginManagement>
```

Jetty configuration for all project modules

Child pom.xml

```
<plugins>
  <plugin>
    <groupId>org.mortbay.jetty</groupId>
    <artifactId>maven-jetty-plugin</artifactId>
  </plugin>
</plugins>
```

# Project Inheritance

## Refactoring common build behavior

- Build configuration shared across modules

Parent pom.xml

```
<pluginManagement>
  <plugins>
    <plugin>
      <groupId>org.mortbay.jetty</groupId>
      <artifactId>maven-jetty-plugin</artifactId>
      <version>6.1.10</version>
      <configuration>
        ...
      </configuration>
    </plugin>
  </plugins>
</pluginManagement>
```

Jetty configuration for all project modules

Child pom.xml

```
<plugins>
  <plugin>
    <groupId>org.mortbay.jetty</groupId>
    <artifactId>maven-jetty-plugin</artifactId>
  </plugin>
</plugins>
```

Modules using Jetty just declare the plugin

# Project Inheritance

## Refactoring common build behavior

- Build configuration shared across modules

# Project Inheritance

## Refactoring common build behavior

- Build configuration shared across modules

Parent pom.xml

```
<pluginManagement>
  <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>
</pluginManagement>
```

# Project Inheritance

## Refactoring common build behavior

- Build configuration shared across modules

Parent pom.xml

```
<pluginManagement>
  <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>
</pluginManagement>
```

Will be bound to the default lifecycle phase for all child project

# Multi-module projects

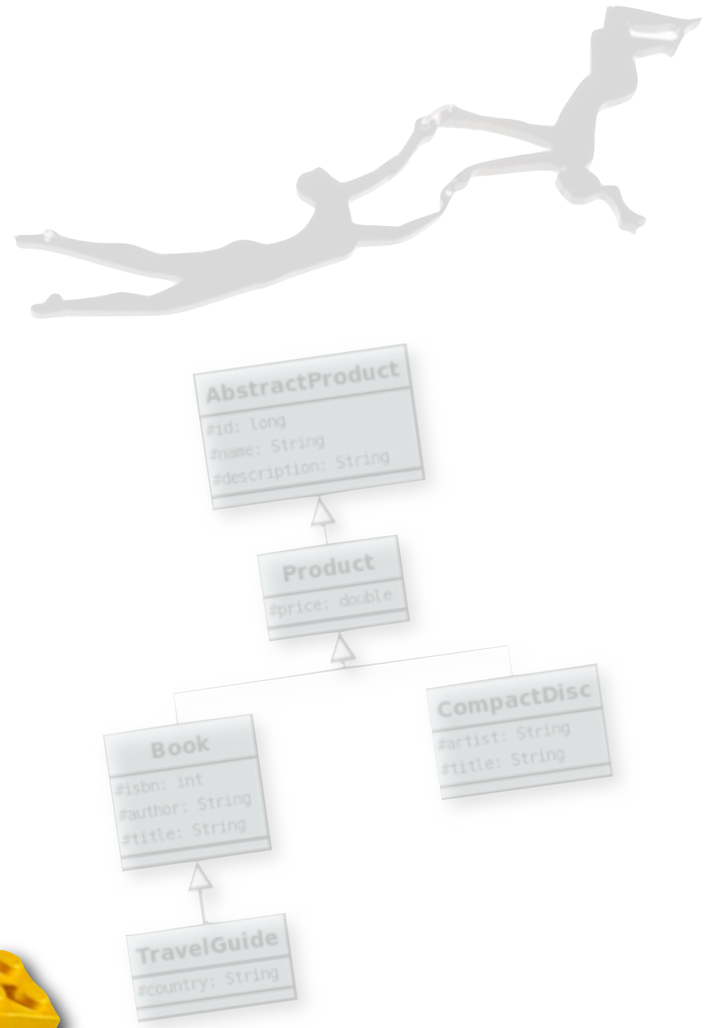
## More than just a pretty architecture

- Why use a modular architecture?


# Multi-module projects

## More than just a pretty architecture

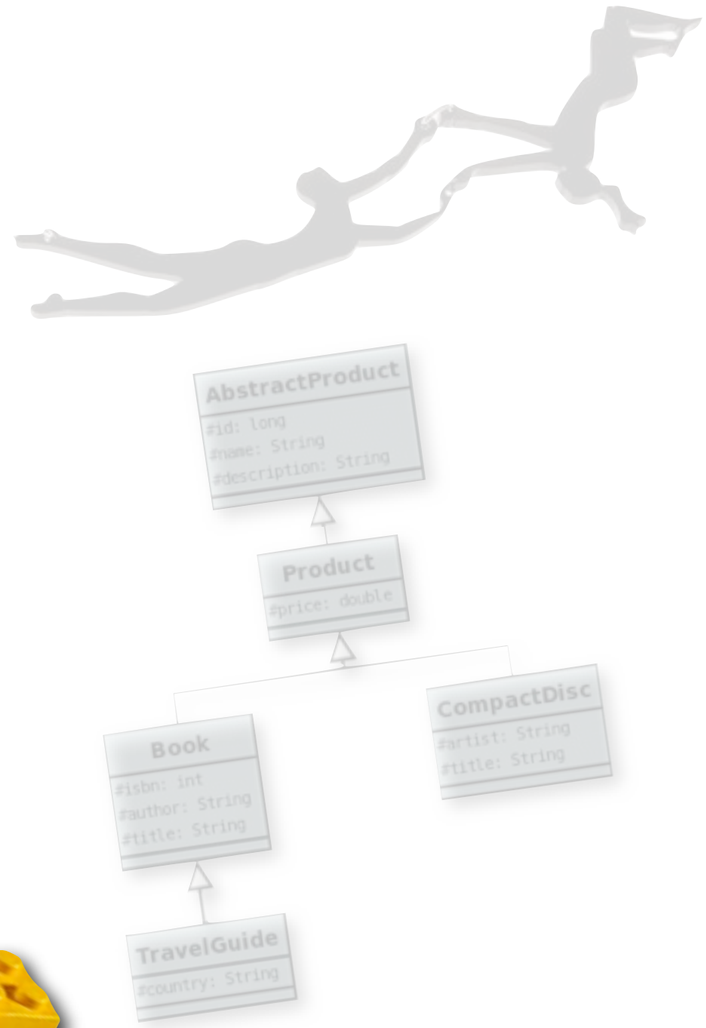
- Why use a modular architecture?
  - Separation of concerns




# Multi-module projects

## More than just a pretty architecture

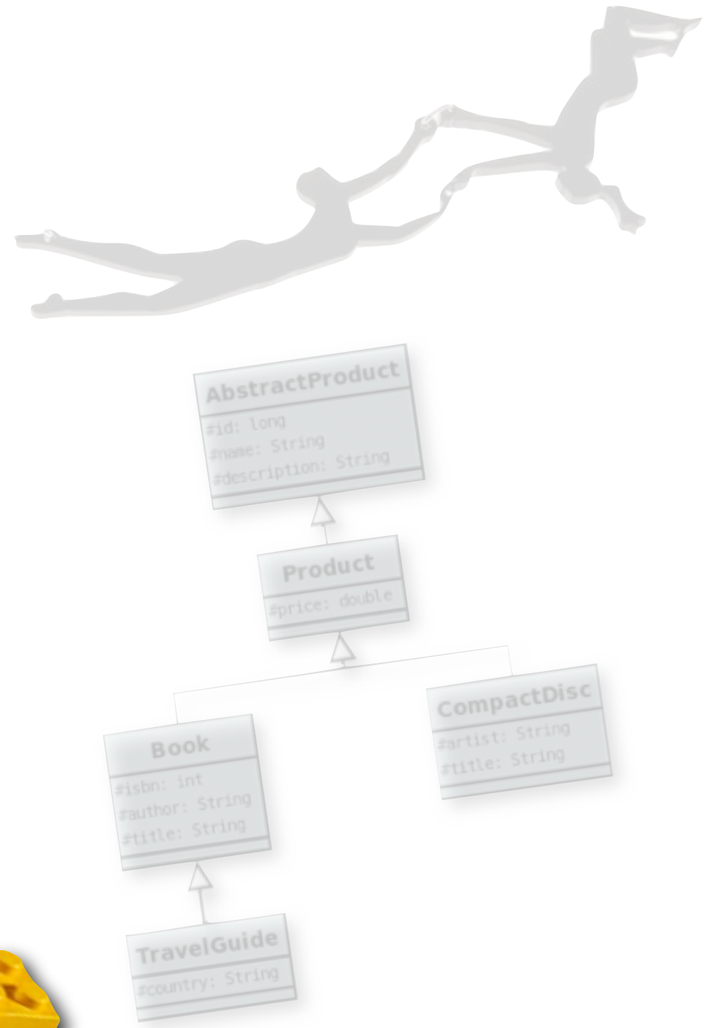
- Why use a modular architecture?
  - Separation of concerns
  - Easier to test




# Multi-module projects

## More than just a pretty architecture

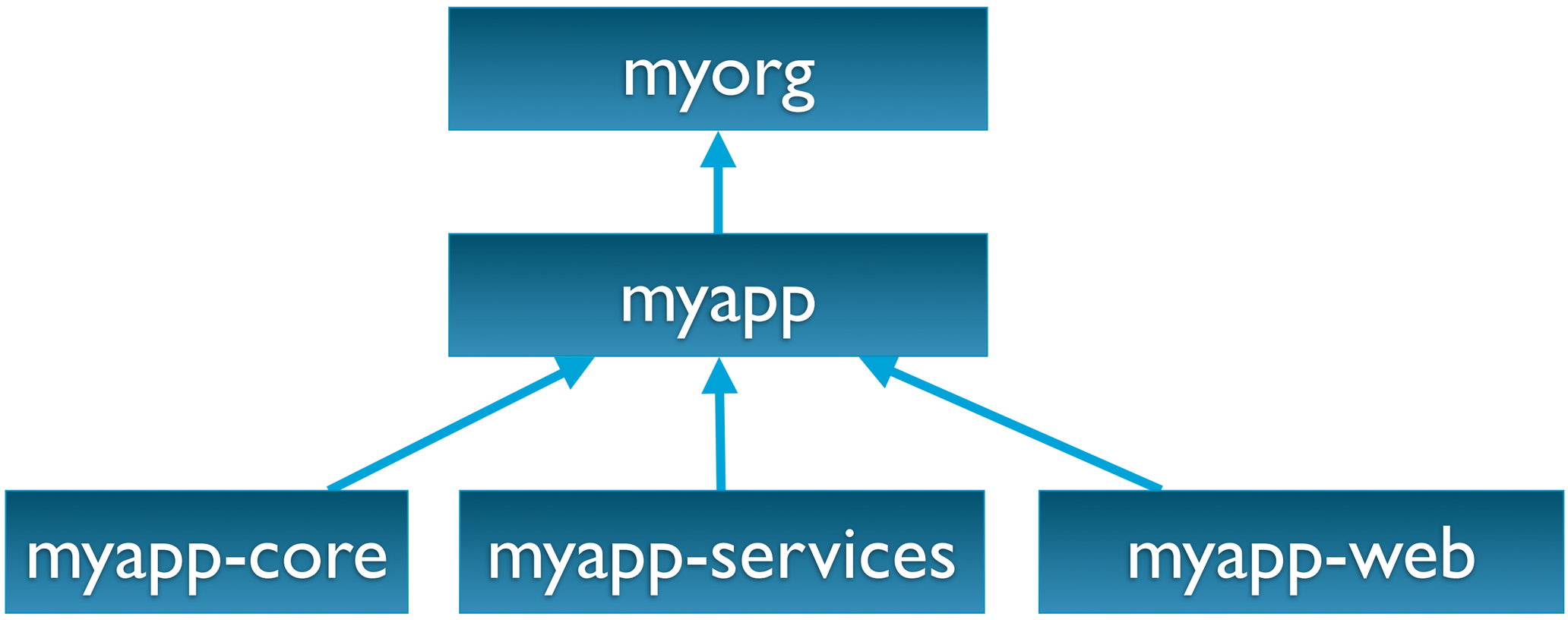
- Why use a modular architecture?
  - Separation of concerns
  - Easier to test
  - More flexible architecture




# Multi-module projects

## A sample project structure

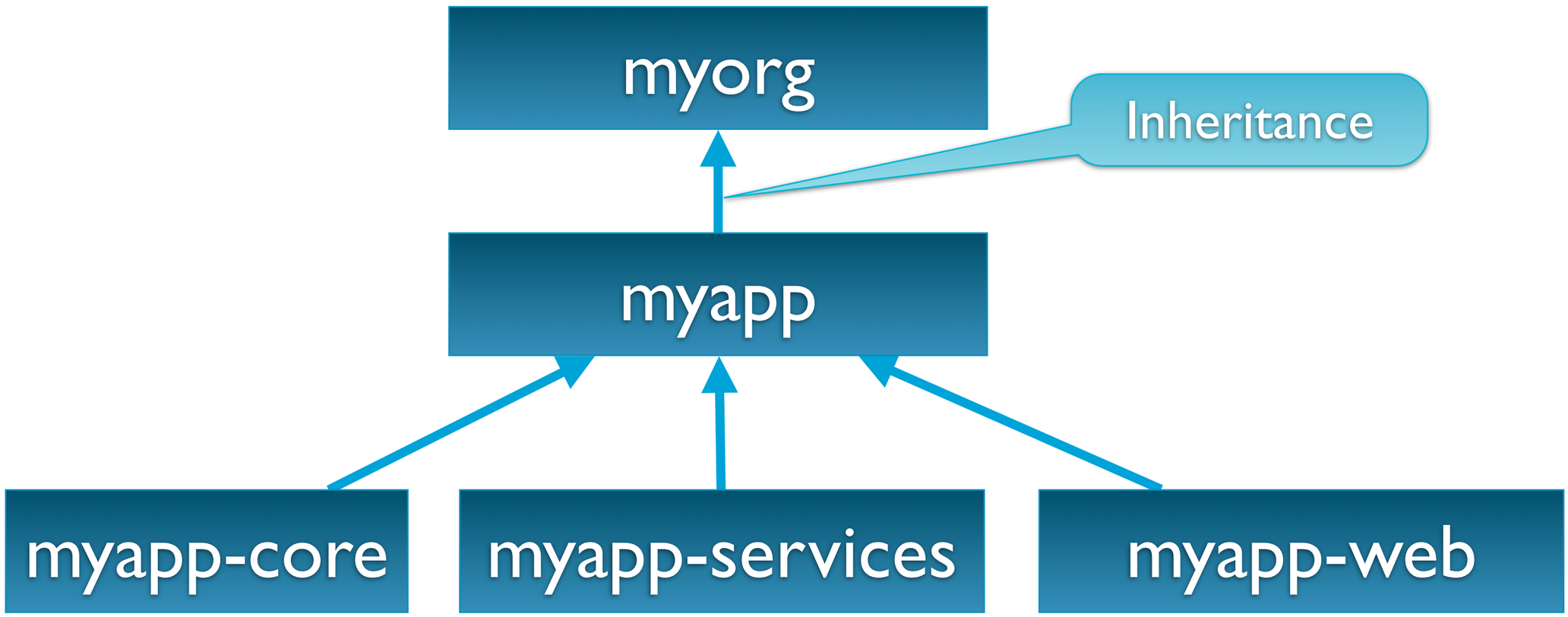
- Inheritance and Aggregation - the Dynamic Duo



# Multi-module projects

## A sample project structure

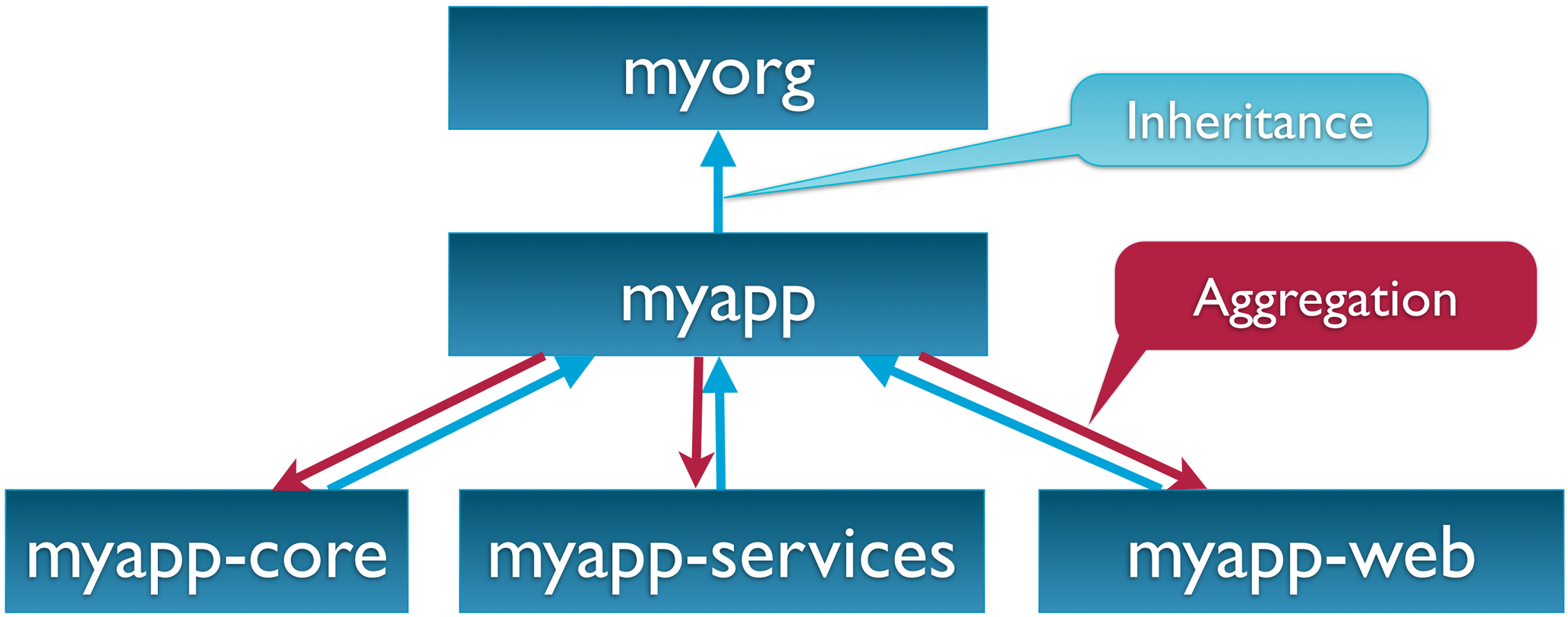
- Inheritance and Aggregation - the Dynamic Duo



# Multi-module projects

## A sample project structure

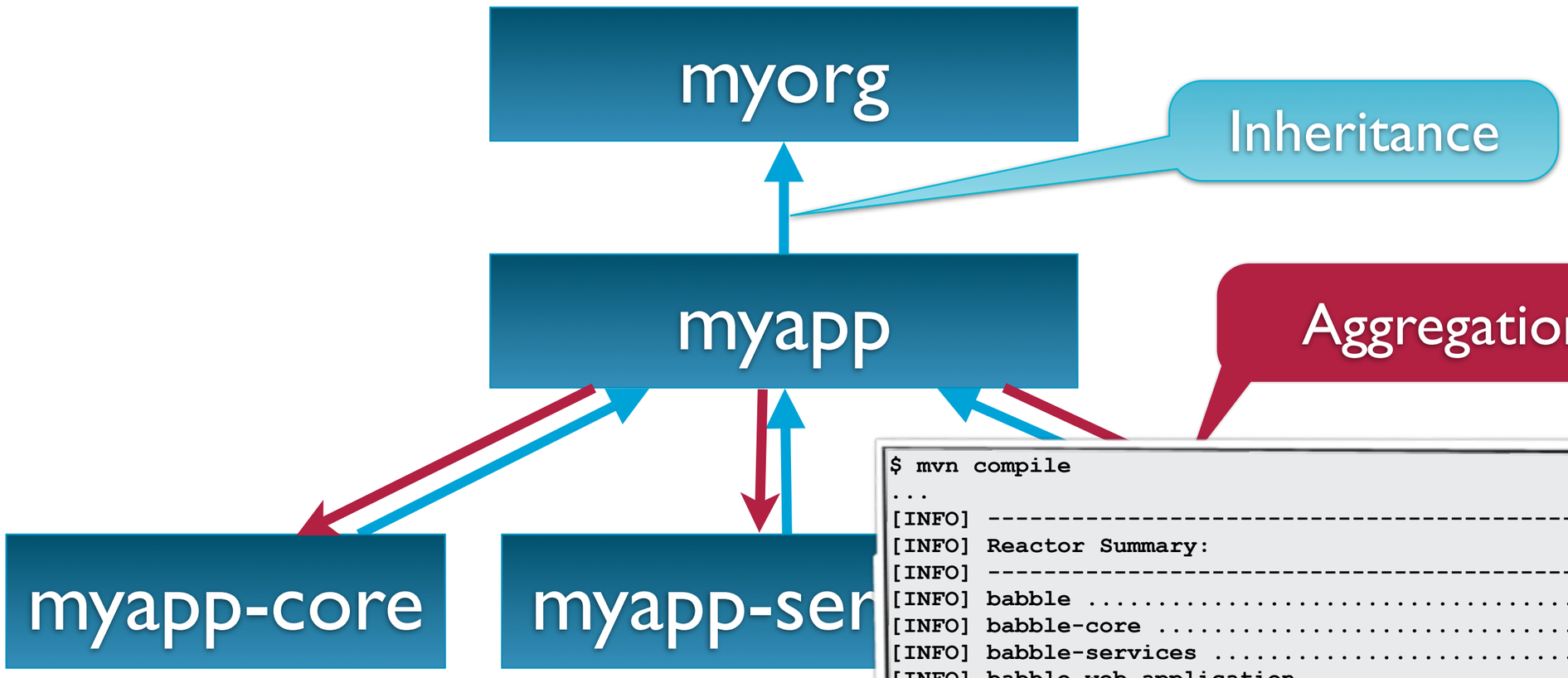
- Inheritance and Aggregation - the Dynamic Duo



# Multi-module projects

## A sample project structure

- Inheritance and Aggregation - the Dynamic Duo




```
$ mvn compile
...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO] ------------------------------------------------------------------------
[INFO] babble ..................................................... SUCCESS [2.362s]
[INFO] babble-core ................................................ SUCCESS [2.138s]
[INFO] babble-services ............................................ SUCCESS [0.221s]
[INFO] babble web application ..................................... SUCCESS [0.347s]
[INFO] babble-webtest ............................................. SUCCESS [0.722s]
[INFO] ------------------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

# Multi-module projects

## What's the buzz?

- What can multi-module projects do for you?

# Multi-module projects
## What's the buzz?

- What can multi-module projects do for you?
  - Coordinate your builds

# Multi-module projects

## What's the buzz?

- What can multi-module projects do for you?
  - Coordinate your builds
  - Stage your build process ("build pipelines")

# Multi-module projects

## What's the buzz?

- What can multi-module projects do for you?
  - Coordinate your builds
  - Stage your build process ("build pipelines")
  - Speed up your builds

# Multi-module projects
## What's the buzz?

- What can multi-module projects do for you?
  - Coordinate your builds
  - Stage your build process ("build pipelines")
  - Speed up your builds
  - Isolate and parallelize long-running tests

# Multi-module projects
## What's the buzz?

- What can multi-module projects do for you?
  - Coordinate your builds
  - Stage your build process ("build pipelines")
  - Speed up your builds
  - Isolate and parallelize long-running tests
  - ...

# Multi-module projects and the testing process

## Accelerating your integration tests

- An example - speed up your integration tests

# Multi-module projects and the testing process

## Accelerating your integration tests

- An example - speed up your integration tests
- Integration tests are slow!

# Multi-module projects and the testing process

## Accelerating your integration tests

- An example - speed up your integration tests
- Integration tests are slow!
  - Often time-consuming

# Multi-module projects and the testing process

## Accelerating your integration tests

- An example - speed up your integration tests
- Integration tests are slow!
  - Often time-consuming
    - web tests, production-scale database tests, performance tests,...

# Multi-module projects and the testing process

## Accelerating your integration tests

- An example - speed up your integration tests
- Integration tests are slow!
  - Often time-consuming
    - web tests, production-scale database tests, performance tests,...
  - Don't always need to be sequential

# Multi-module projects and the testing process

## Accelerating your integration tests

- An example - speed up your integration tests
- Integration tests are slow!
  - Often time-consuming
    - web tests, production-scale database tests, performance tests,...
  - Don't always need to be sequential
    - web tests on different browsers, unrelated database tests,...

# Multi-module projects and the testing process

## Accelerating your integration tests

- So how can I speed up my integration tests?

# Multi-module projects and the testing process

## Accelerating your integration tests

- So how can I speed up my integration tests?
  - Use modules!

# Multi-module projects and the testing process

## Accelerating your integration tests

- So how can I speed up my integration tests?
  - Use modules!
  - Isolate your integration tests

# Multi-module projects and the testing process

## Accelerating your integration tests

- So how can I speed up my integration tests?
  - Use modules!
  - Isolate your integration tests
    - Use directory or naming conventions

# Multi-module projects and the testing process

## Accelerating your integration tests

- So how can I speed up my integration tests?
  - Use modules!
  - Isolate your integration tests
    - Use directory or naming conventions
  - Place them in separate modules

# Multi-module projects and the testing process

## Accelerating your integration tests

- So how can I speed up my integration tests?
  - Use modules!
  - Isolate your integration tests
    - Use directory or naming conventions
    - Place them in separate modules
    - Use profiles and/or properties

# Multi-module projects and the testing process

## Accelerating your integration tests

- So how can I speed up my integration tests?
  - Use modules!
  - Isolate your integration tests
    - Use directory or naming conventions
    - Place them in separate modules
    - Use profiles and/or properties
  - Don't rebuild, deploy!

# Multi-module projects and the testing process

## Accelerating your integration tests

- So how can I speed up my integration tests?
  - Use modules!
  - Isolate your integration tests
    - Use directory or naming conventions
    - Place them in separate modules
    - Use profiles and/or properties
- Don't rebuild, deploy!
  - Deploy to your repository, then reuse

# Multi-module projects and the testing process

## An example - specifying what integration tests to run

- Configurable integration tests in 3 easy steps

# Multi-module projects and the testing process

## An example - specifying what integration tests to run

- Configurable integration tests in 3 easy steps

```
<properties>
  <integration.tests>
  **/*IntegrationTest.java,**/*WebTest.java
  </integration.tests>
</properties>
```

# Multi-module projects and the testing process

## An example - specifying what integration tests to run

- Configurable integration tests in 3 easy steps

```
<properties>
  <integration.tests>
  **/*IntegrationTest.java,**/*WebTest.java
  </integration.tests>
</properties>
```

What tests are run by default

# Multi-module projects and the testing process

## An example - specifying what integration tests to run

- Configurable integration tests in 3 easy steps

```
<properties>
  <integration.tests>
  **/*IntegrationTest.java,**/*WebTest.java
  </integration.tests>
</properties>
```

What tests are run by default

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  ...
  <configuration>
    <includes>
      <include>${integration.tests}</include>
    </includes>
  </configuration>
  ...
</plugin>
```

# Multi-module projects and the testing process

## An example - specifying what integration tests to run

- Configurable integration tests in 3 easy steps

```
<properties>
  <integration.tests>
  **/*IntegrationTest.java,**/*WebTest.java
  </integration.tests>
</properties>
```

What tests are run by default

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  ...
  <configuration>
    <includes>
      <include>${integration.tests}</include>
    </includes>
  </configuration>
  ...
</plugin>
```

Only execute these tests

# Multi-module projects and the testing process

## An example - specifying what integration tests to run

- Configurable integration tests in 3 easy steps

```
<properties>
  <integration.tests>
  **/*IntegrationTest.java,**/*WebTest.java
  </integration.tests>
</properties>
```

What tests are run by default

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  ...
  <configuration>
    <includes>
      <include>${integration.tests}</include>
    </includes>
  </configuration>
  ...
</plugin>
```

Only execute these tests

```
$ mvn verify -Dintegration.tests=**/*SlowWebtest.java
```

# Multi-module projects and the testing process

## An example - specifying what integration tests to run

- Configurable integration tests in 3 easy steps

```
<properties>
  <integration.tests>
  **/*IntegrationTest.java,**/*WebTest.java
  </integration.tests>
</properties>
```

What tests are run by default

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  ...
  <configuration>
    <includes>
      <include>${integration.tests}</include>
    </includes>
  </configuration>
  ...
</plugin>
```

Only execute these tests

Override the default value in a profile or on the command line

```
$ mvn verify -Dintegration.tests=**/*SlowWebtest.java
```

# Multi-module projects and the testing process

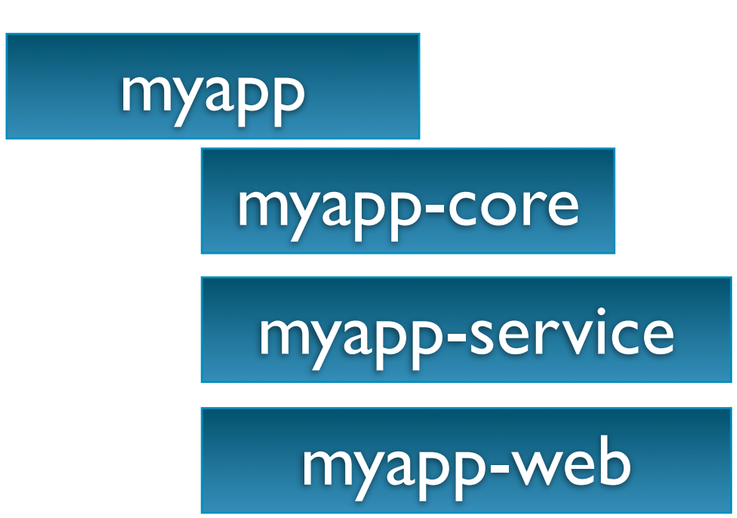
## Accelerating your integration tests

- Don’t rebuild, deploy!

# Multi-module projects and the testing process

## Accelerating your integration tests

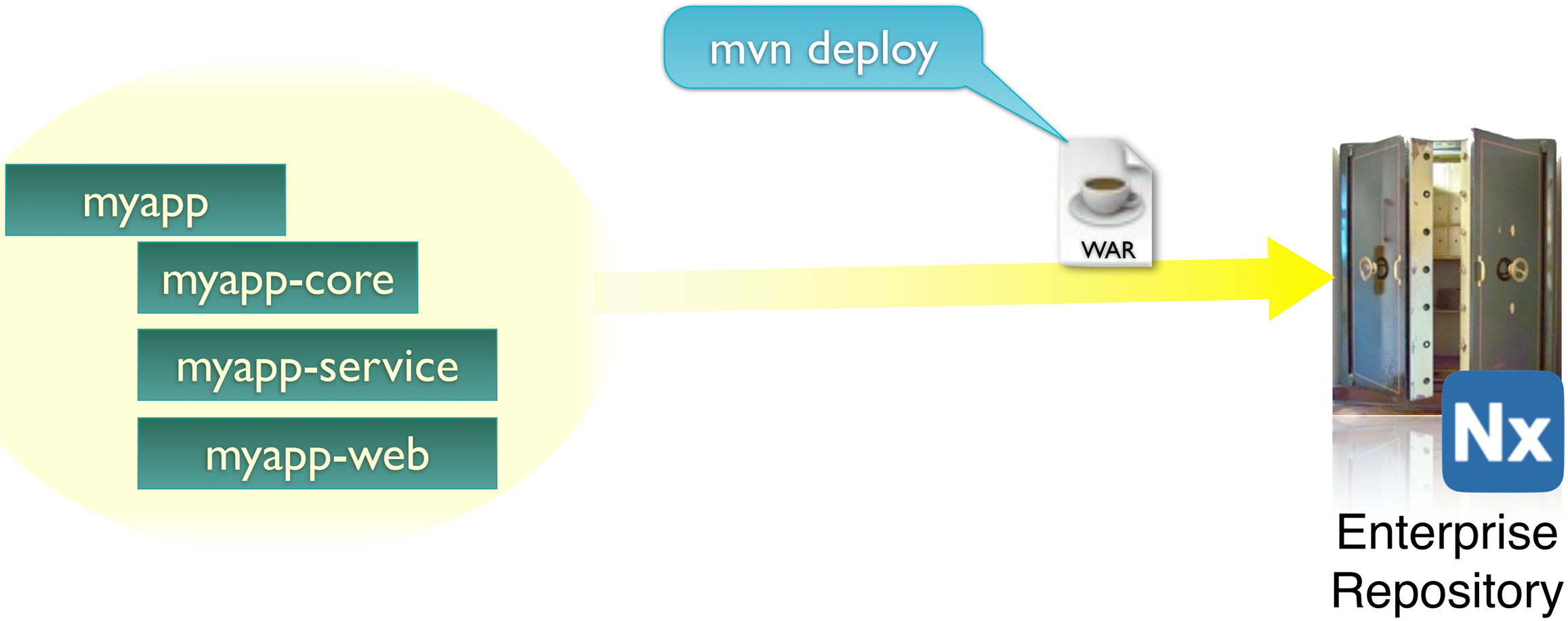
- Don’t rebuild, deploy!



# Multi-module projects and the testing process

## Accelerating your integration tests

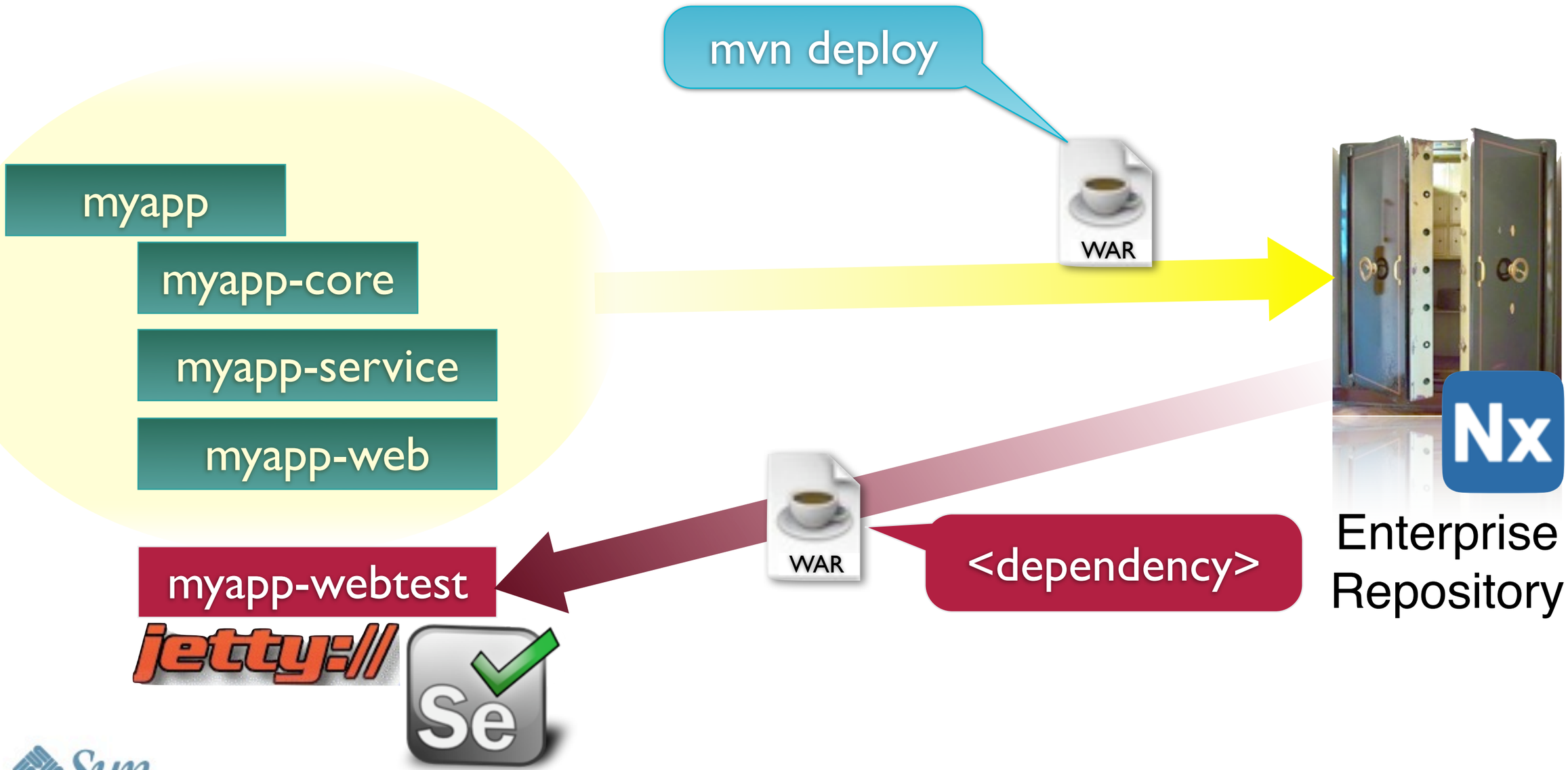
- Don’t rebuild, deploy!



# Multi-module projects and the testing process

## Accelerating your integration tests

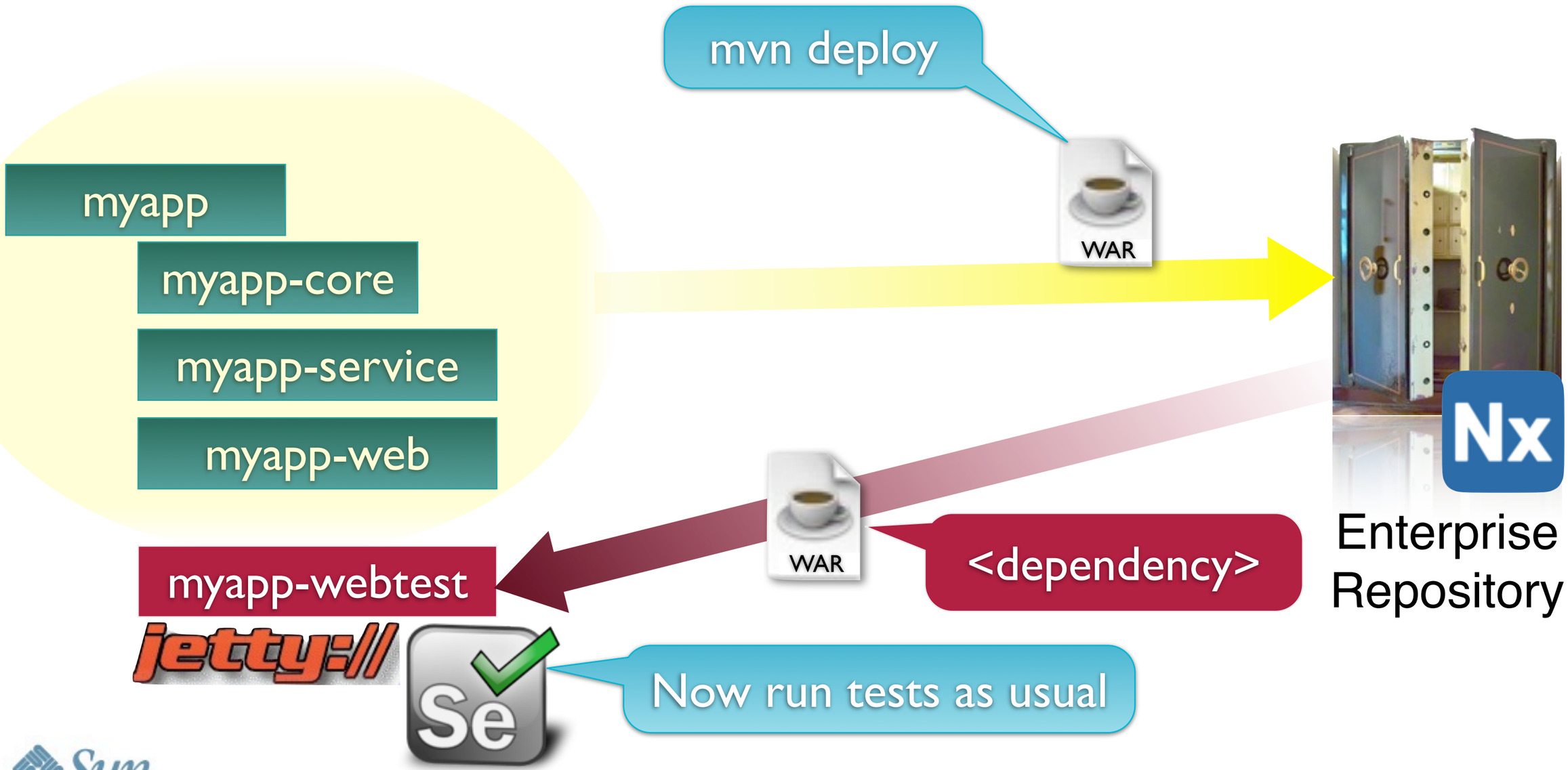
- Don’t rebuild, deploy!



# Multi-module projects and the testing process

## Accelerating your integration tests

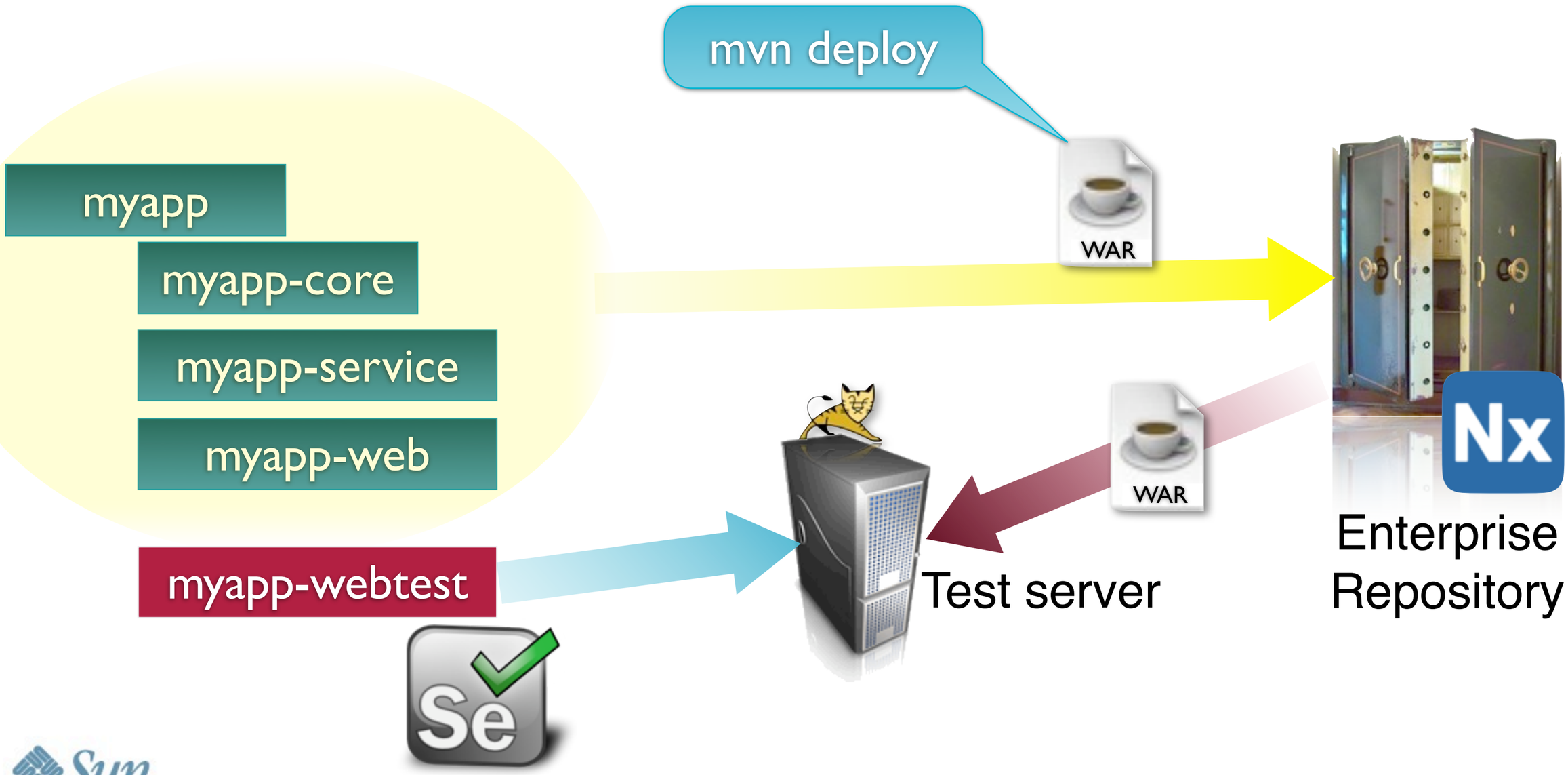
- Don’t rebuild, deploy!



# Multi-module projects and the testing process

## Accelerating your integration tests

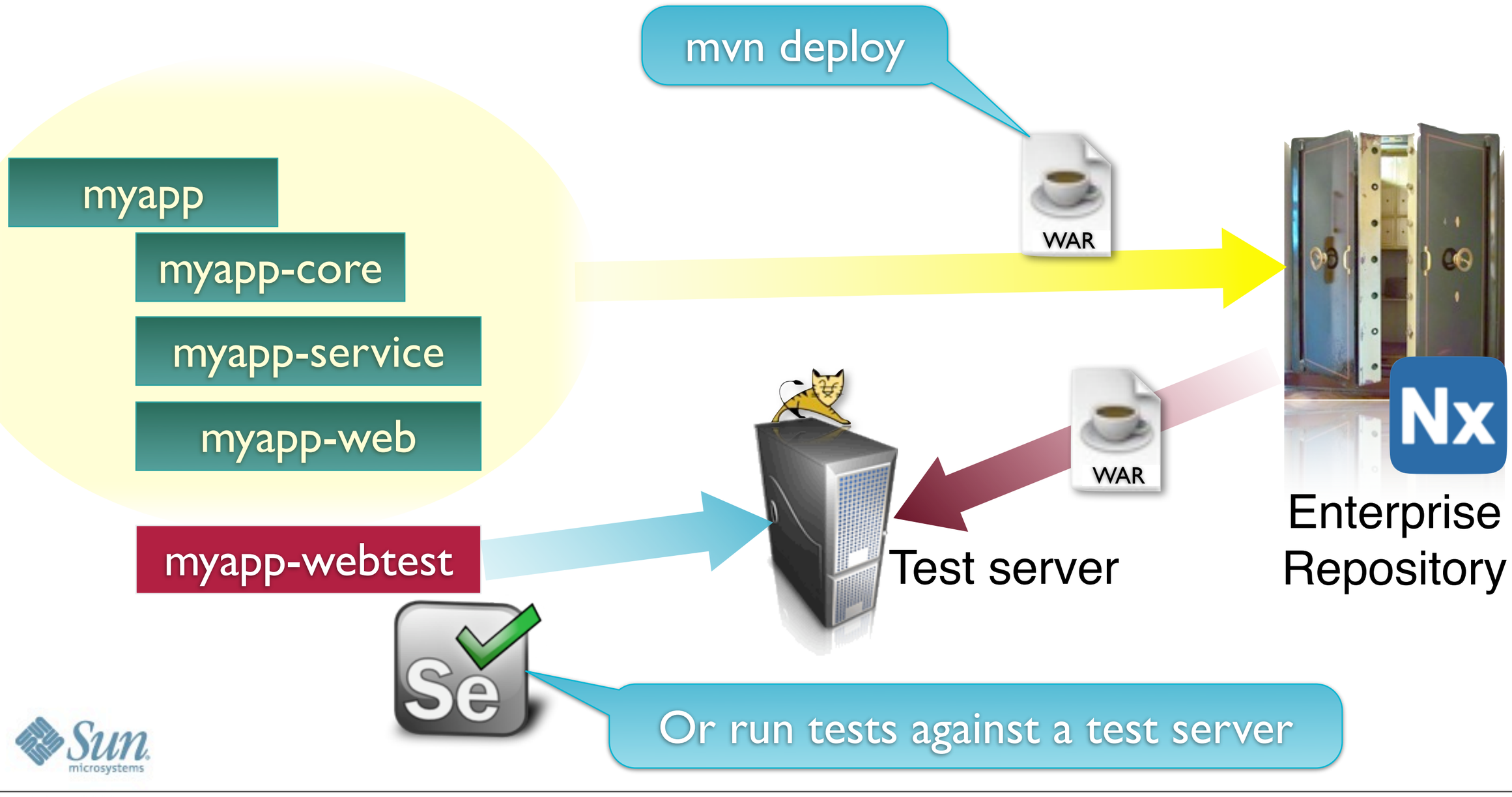
- Don’t rebuild, deploy!



# Multi-module projects and the testing process

## Accelerating your integration tests

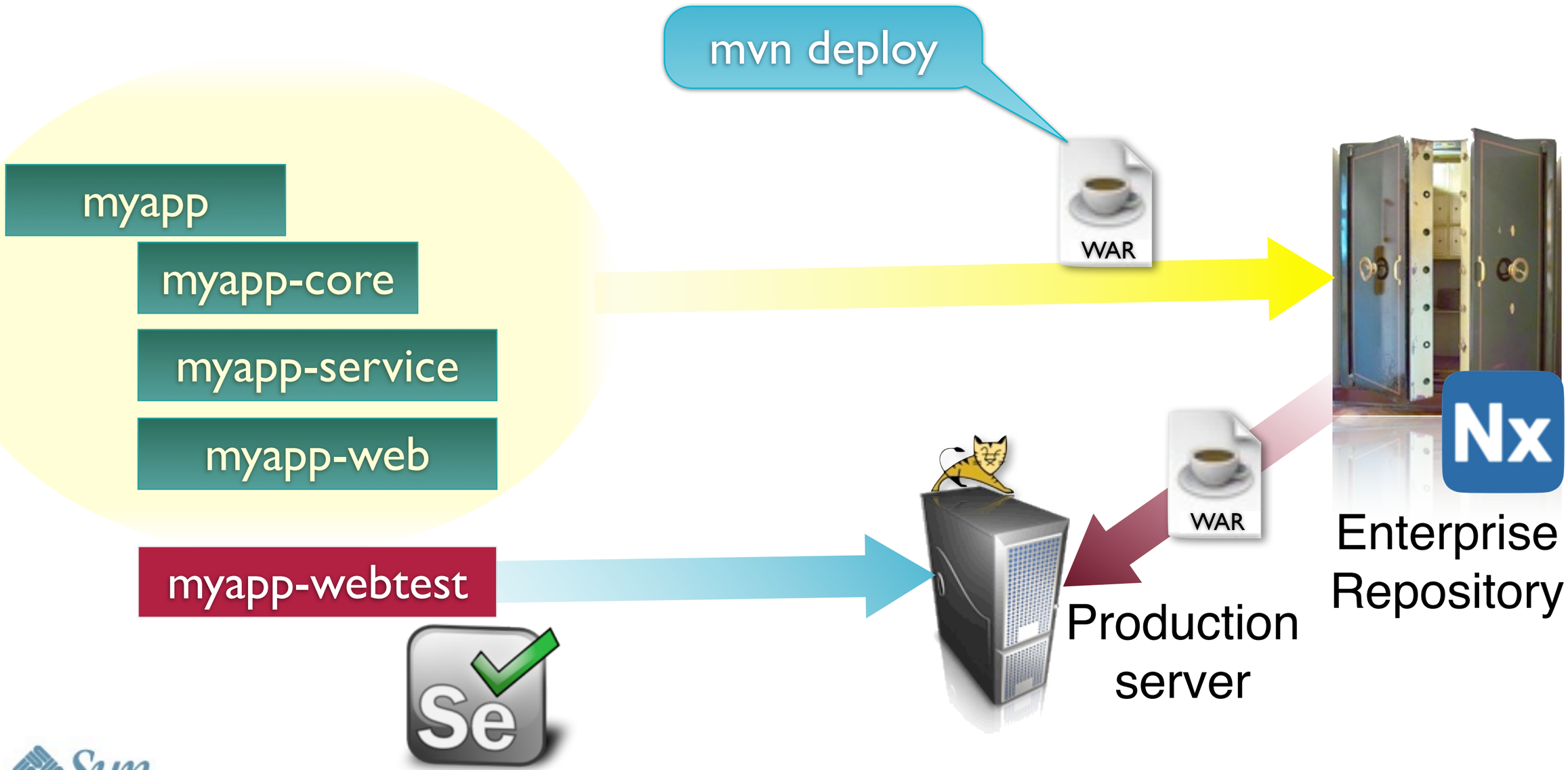
- Don’t rebuild, deploy!



# Multi-module projects and the testing process

## Accelerating your integration tests

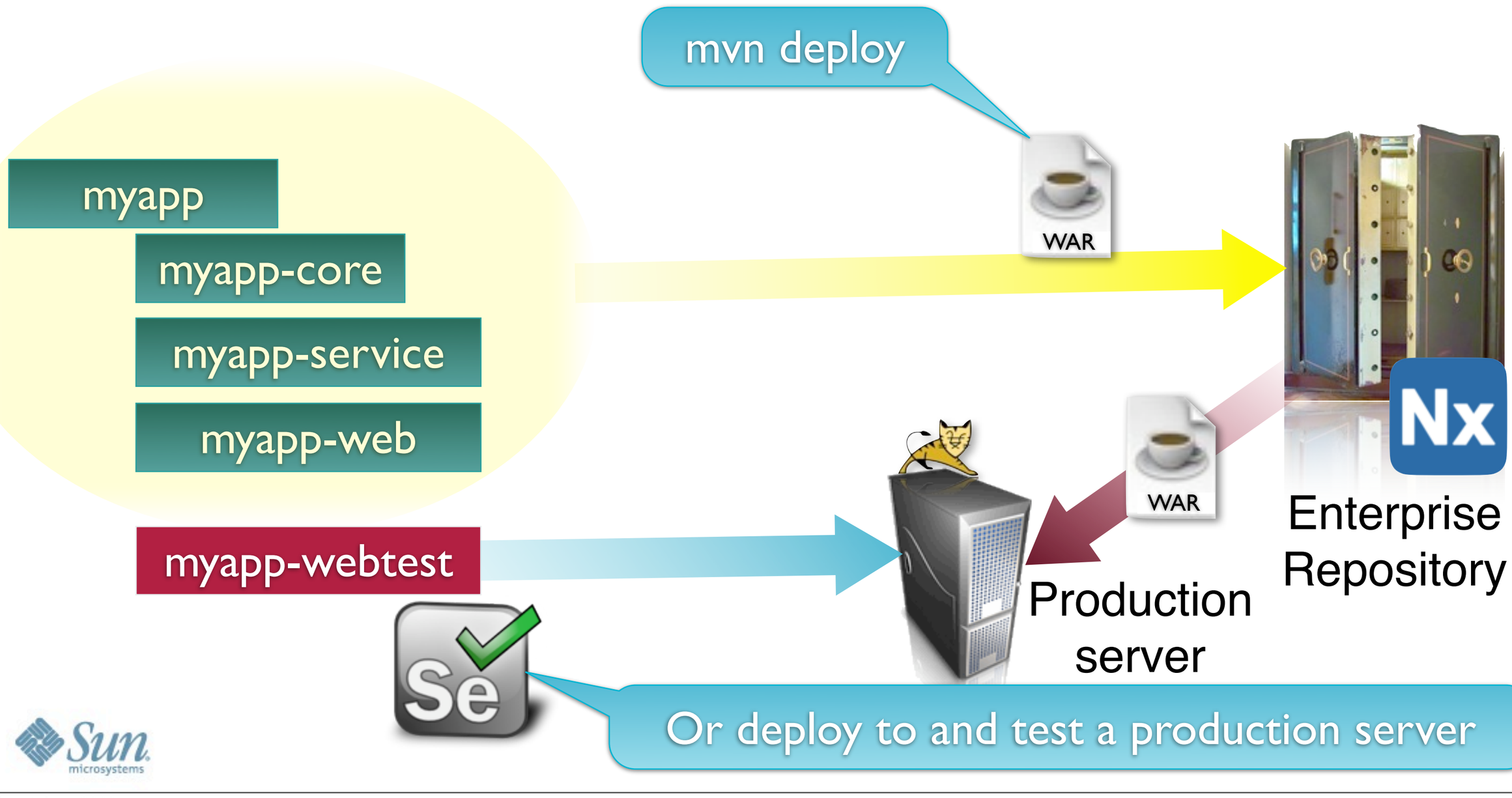
- Don’t rebuild, deploy!



# Multi-module projects and the testing process

## Accelerating your integration tests

- Don’t rebuild, deploy!



# Multi-module projects and the testing process

## An example - reusing a WAR file

- Reusing an existing WAR file in 3 easy steps!

# Multi-module projects and the testing process

## An example - reusing a WAR file

- Reusing an existing WAR file in 3 easy steps!

```
<dependency>
  <groupId>myorg.myapp</groupId>
  <artifactId>myapp-web</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <type>war</type>
  <scope>runtime</scope>
</dependency>
```

# Multi-module projects and the testing process

## An example - reusing a WAR file

- Reusing an existing WAR file in 3 easy steps!

```
<dependency>
  <groupId>myorg.myapp</groupId>
  <artifactId>myapp-web</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <type>war</type>
  <scope>runtime</scope>
</dependency>
```

Add a snapshot dependency to your WAR

# Multi-module projects and the testing process

## An example - reusing a WAR file

- Reusing an existing WAR file in 3 easy steps!

```
<dependency>
  <groupId>myorg.myapp</groupId>
  <artifactId>myapp-web</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <ty
  <sc
</dep
```

```
<plugin>
  <artifactId>maven-war-plugin</artifactId>
  <configuration>
    <overlays>
      <overlay>
        <groupId>myorg.myapp</groupId>
        <artifactId>myapp-web</artifactId>
      </overlay>
    </overlays>
  </configuration>
</plugin>
```

Add a snapshot dependency to your WAR

# Multi-module projects and the testing process

## An example - reusing a WAR file

- Reusing an existing WAR file in 3 easy steps!

```
<dependency>
  <groupId>myorg.myapp</groupId>
  <artifactId>myapp-web</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <ty
  <sc
</dep
```

```
<plugin>
  <artifactId>maven-war-plugin</artifactId>
  <configuration>
    <overlays>
      <overlay>
        <groupId>myorg.myapp</groupId>
        <artifactId>myapp-web</artifactId>
      </overlay>
    </overlays>
  </configuration>
</plugin>
```

Add a snapshot dependency to your WAR

Use overlays to include the dependent WAR into this project

# Multi-module projects and the testing process

## An example - reusing a WAR file

- Reusing an existing WAR file in 3 easy steps!

```
<dependency>
  <groupId>myorg.myapp</groupId>
  <artifactId>myapp-web</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <ty
  <sc
</dep
```

Add a snapshot dependency to your WAR

```
<plugin>
  <artifactId>maven-war-plugin</artifactId>
  <configuration>
    <overlays>
      <overlay>
        <groupId>myorg.myapp</groupId>
        <artifactId>myapp-web</artifactId>
      </
  </co
</plug
```

Use overlays to include the dependent WAR into this project

```
<plugin>
  <groupId>org.mortbay.jetty</groupId>
  <artifactId>maven-jetty-plugin</artifactId>
  <executions>
    <execution>
      <id>start-jetty</id>
      <phase>pre-integration-test</phase>
      <goals>
        <goal>run-war</goal>
      </goals>
```

# Multi-module projects and the testing process

## An example - reusing a WAR file

- Reusing an existing WAR file in 3 easy steps!

```
<dependency>
  <groupId>myorg.myapp</groupId>
  <artifactId>myapp-web</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <ty
  <sc
</dep
```

Add a snapshot dependency to your WAR

```
<plugin>
  <artifactId>maven-war-plugin</artifactId>
  <configuration>
    <overlays>
      <overlay>
        <groupId>myorg.myapp</groupId>
        <artifactId>myapp-web</artifactId>
      </
  </co
</plug
```

Use overlays to include the dependent WAR into this project

```
<plugin>
  <groupId>org.mortbay.jetty</groupId>
  <artifactId>maven-jetty-plugin</artifactId>
  <executions>
    <execution>
      <id>start-jetty</id>
      <phase>pre-integration-test</phase>
      <goals>
        <goal>run-war</goal>
      </goals>
```

Run jetty against the WAR file

# Multi-module projects and the testing process

## Staging your builds

- Split your builds into distinct stages


Enterprise Repository

# Multi-module projects and the testing process

## Staging your builds

- Split your builds into distinct stages
  - Faster notification


Enterprise Repository

# Multi-module projects and the testing process

## Staging your builds

- Split your builds into distinct stages
  - Faster notification
  - Parallel builds where possible


Enterprise Repository

# Multi-module projects and the testing process

## Staging your builds

- Split your builds into distinct stages
  - Faster notification
  - Parallel builds where possible

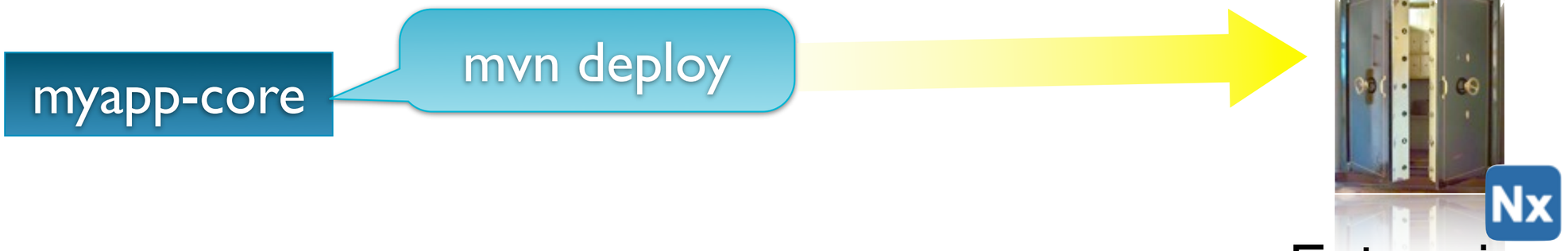
myapp-core


Enterprise Repository

# Multi-module projects and the testing process

## Staging your builds

- Split your builds into distinct stages
  - Faster notification
  - Parallel builds where possible



# Multi-module projects and the testing process

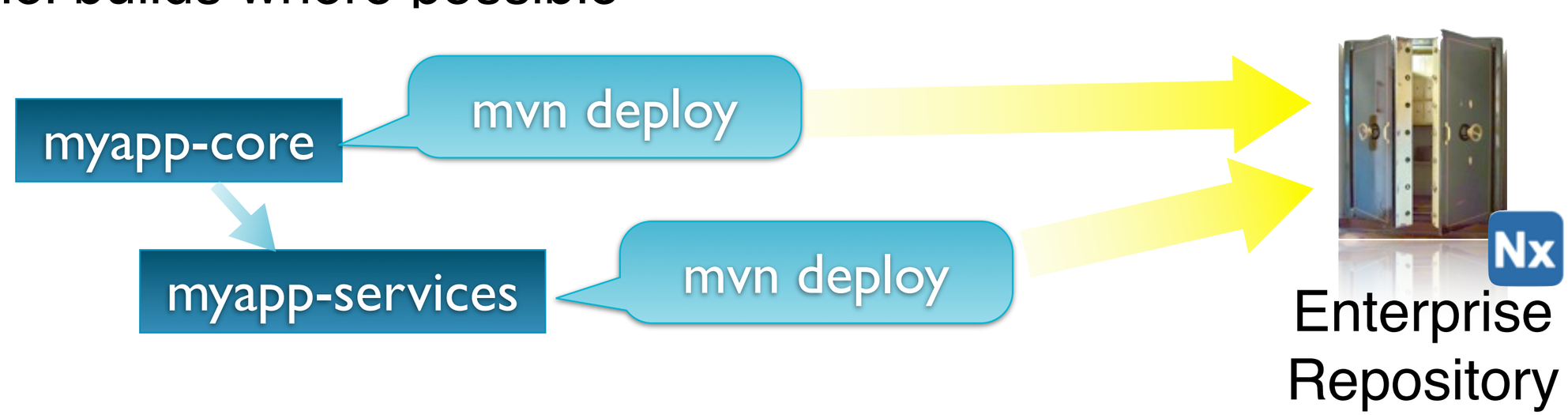
## Staging your builds

- Split your builds into distinct stages
  - Faster notification
  - Parallel builds where possible

# Multi-module projects and the testing process

## Staging your builds

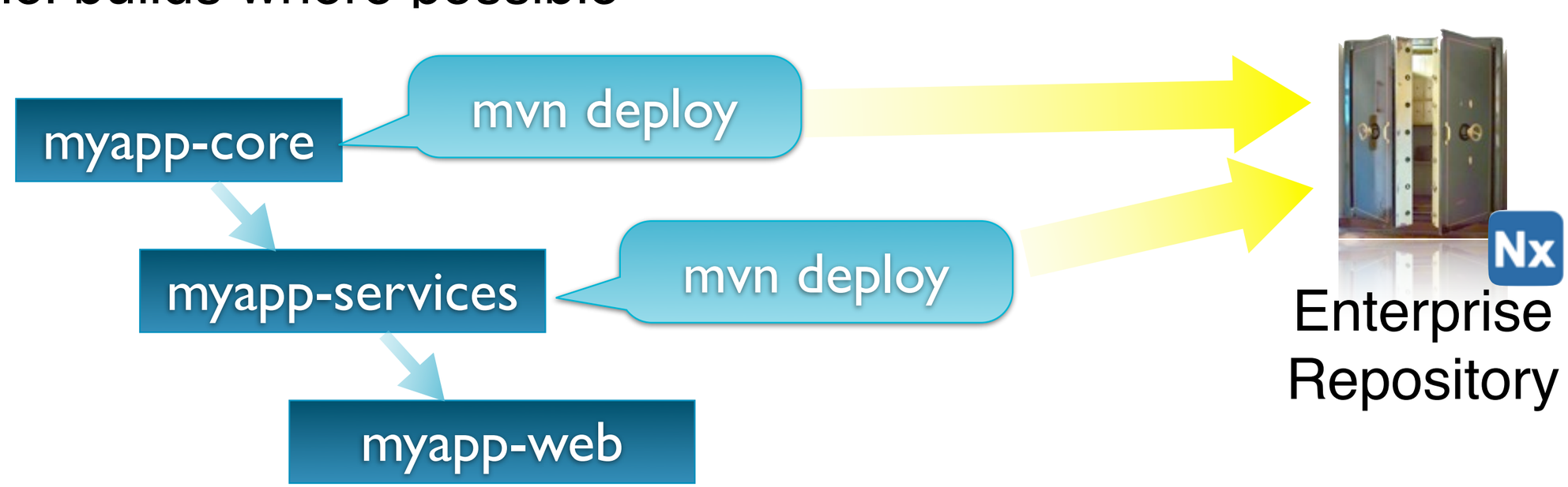
- Split your builds into distinct stages
  - Faster notification
  - Parallel builds where possible



# Multi-module projects and the testing process

## Staging your builds

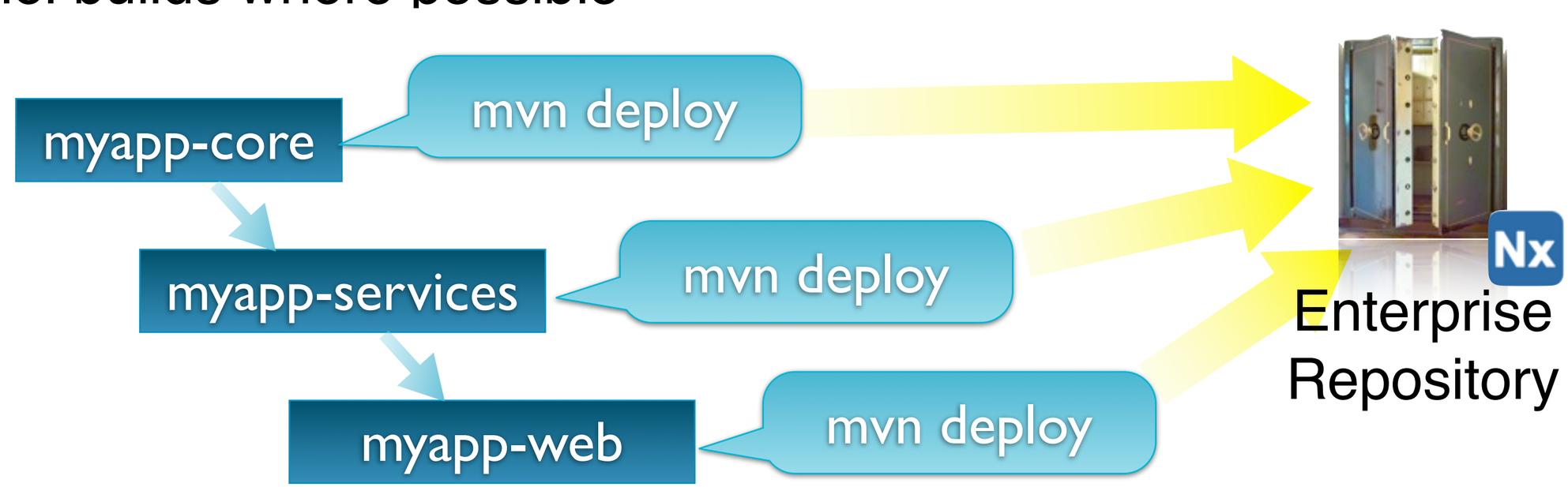
- Split your builds into distinct stages
  - Faster notification
  - Parallel builds where possible



# Multi-module projects and the testing process

## Staging your builds

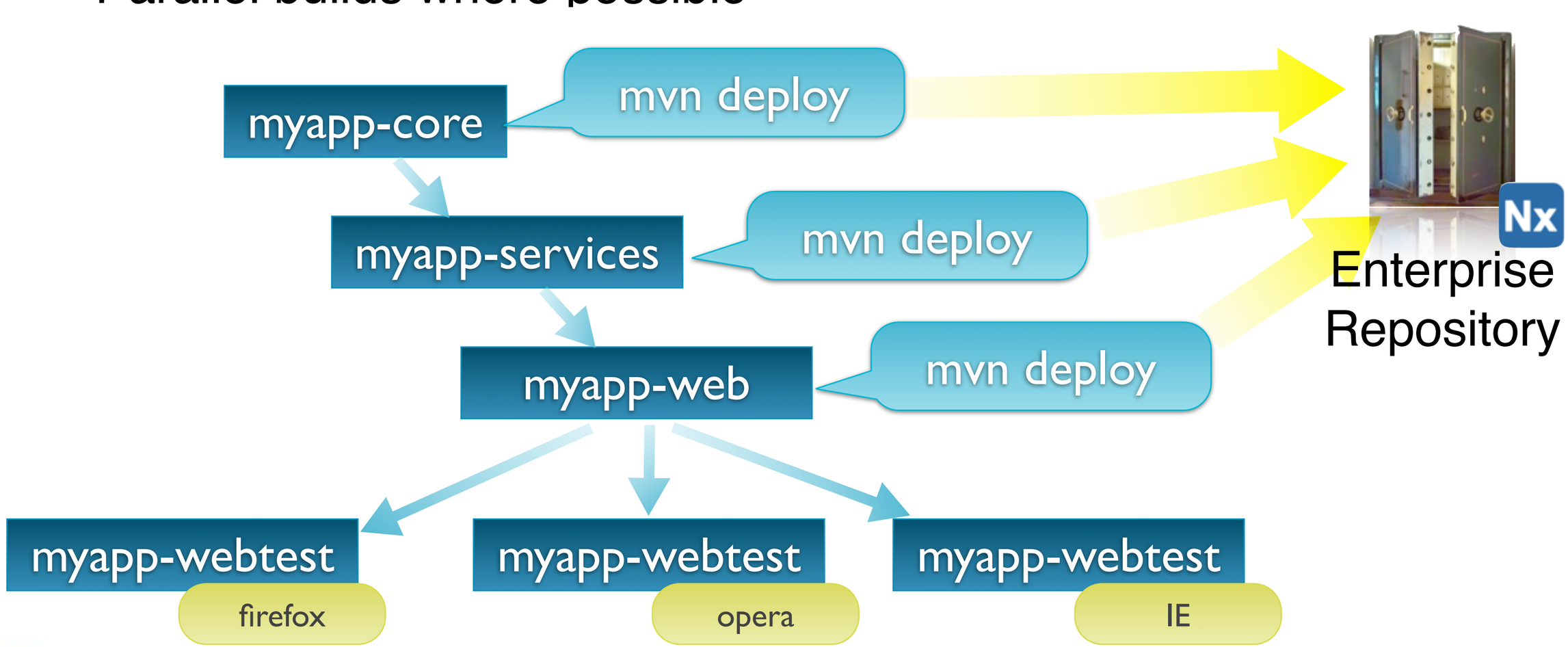
- Split your builds into distinct stages
  - Faster notification
  - Parallel builds where possible



# Multi-module projects and the testing process

## Staging your builds

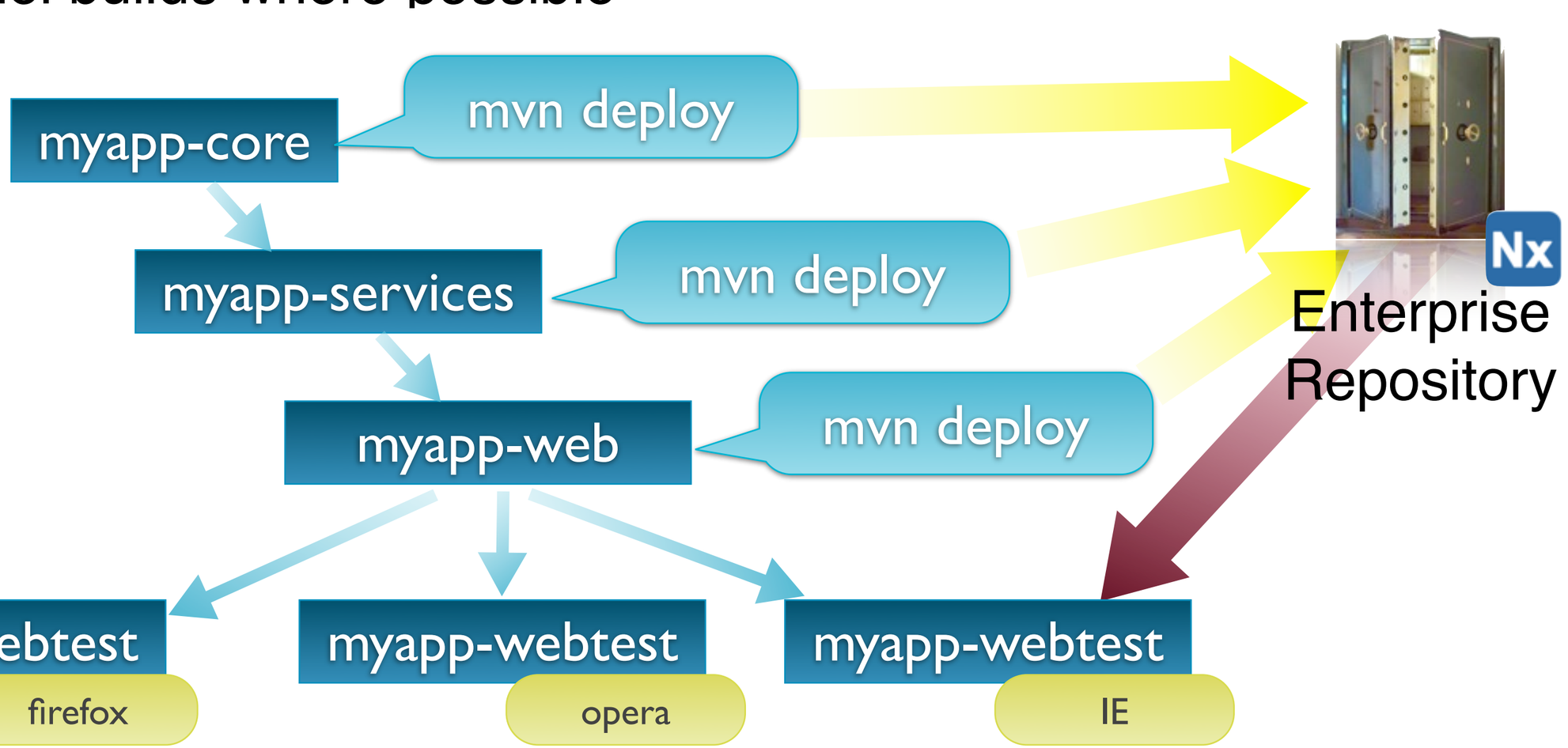
- Split your builds into distinct stages
  - Faster notification
  - Parallel builds where possible



# Multi-module projects and the testing process

## Staging your builds

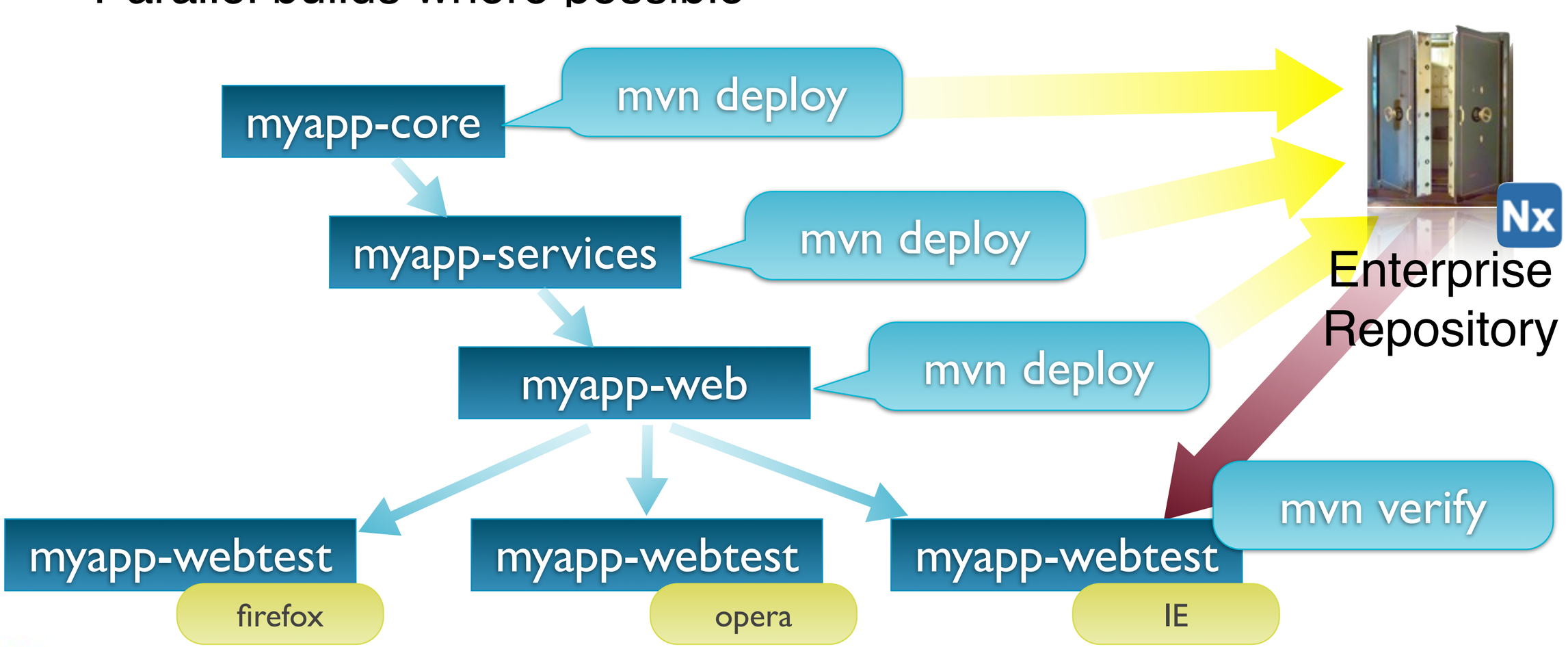
- Split your builds into distinct stages
  - Faster notification
  - Parallel builds where possible



# Multi-module projects and the testing process

## Staging your builds

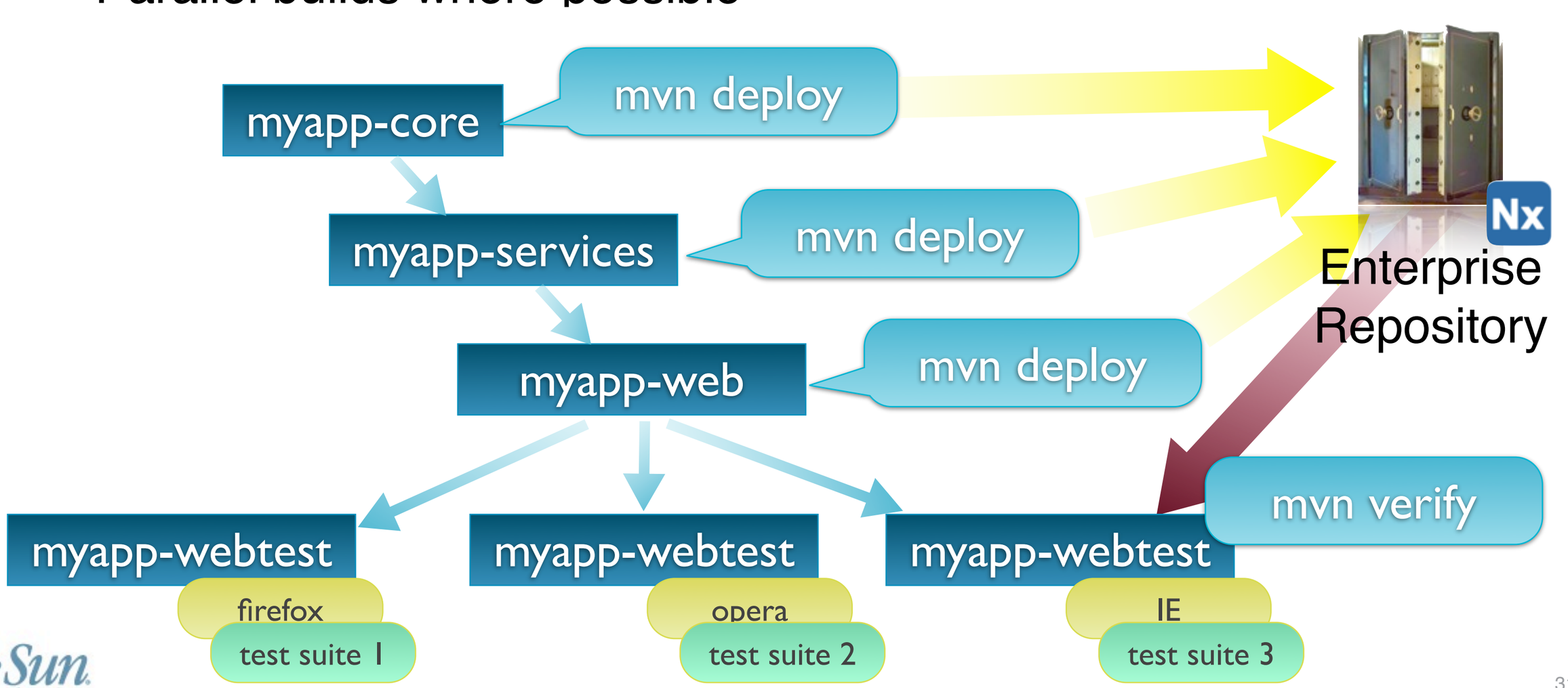
- Split your builds into distinct stages
  - Faster notification
  - Parallel builds where possible



# Multi-module projects and the testing process

## Staging your builds

- Split your builds into distinct stages
  - Faster notification
  - Parallel builds where possible



# Revving up your release process

## Automated releases

- A typical Maven project ecosystem

myapp

myapp-core

myapp-service

myapp-web


Enterprise Repository

CI build server

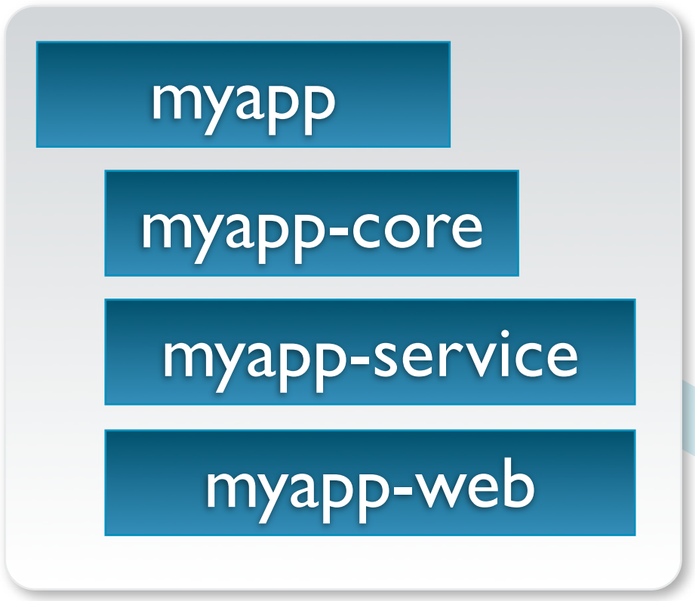
SCM server

SUBVERSION

# Revving up your release process

## Automated releases

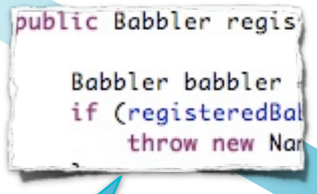
- A typical Maven project ecosystem





Enterprise Repository

CI build server




Source code changes

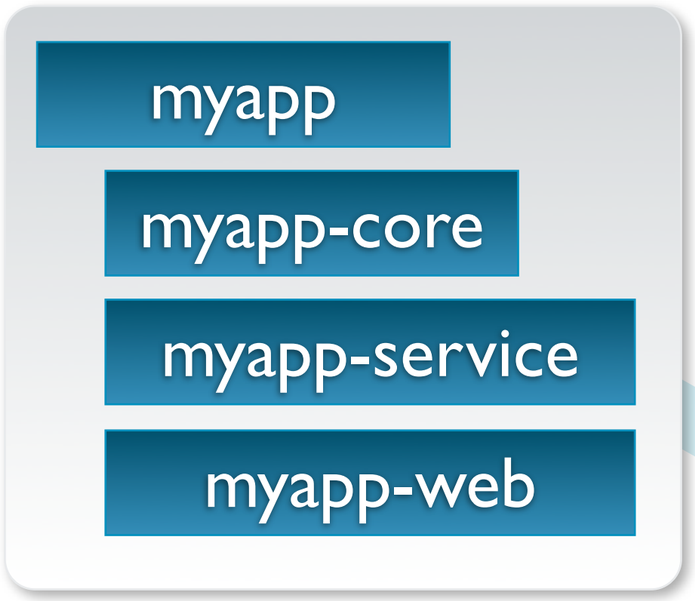
SCM server

SUBVERSION

# Revving up your release process

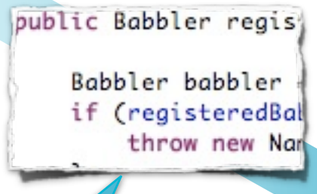
## Automated releases

- A typical Maven project ecosystem





Enterprise Repository




Source code changes

SCM server

SUBVERSION

CI build server

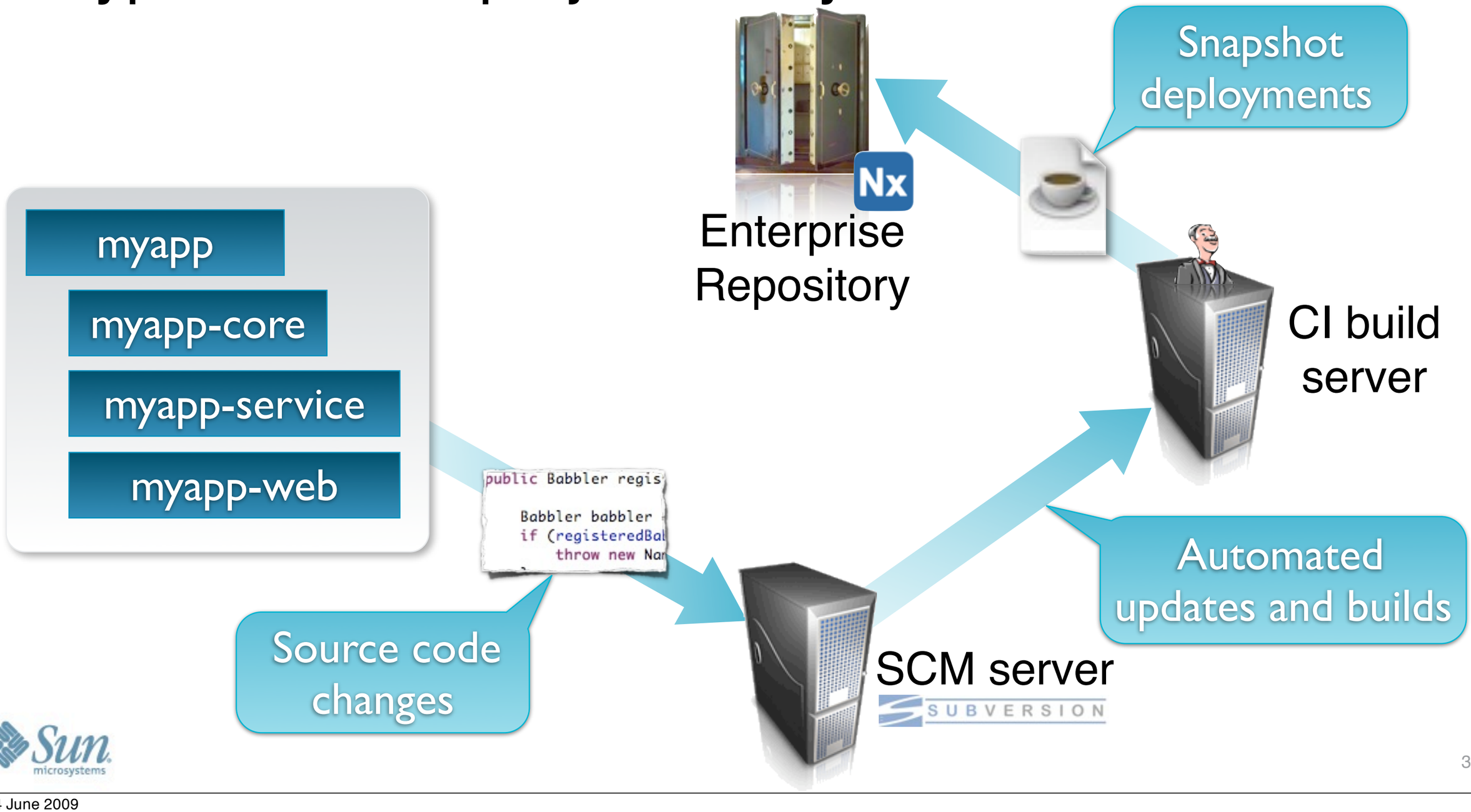
Automated updates and builds

Sun microsystems

# Revving up your release process

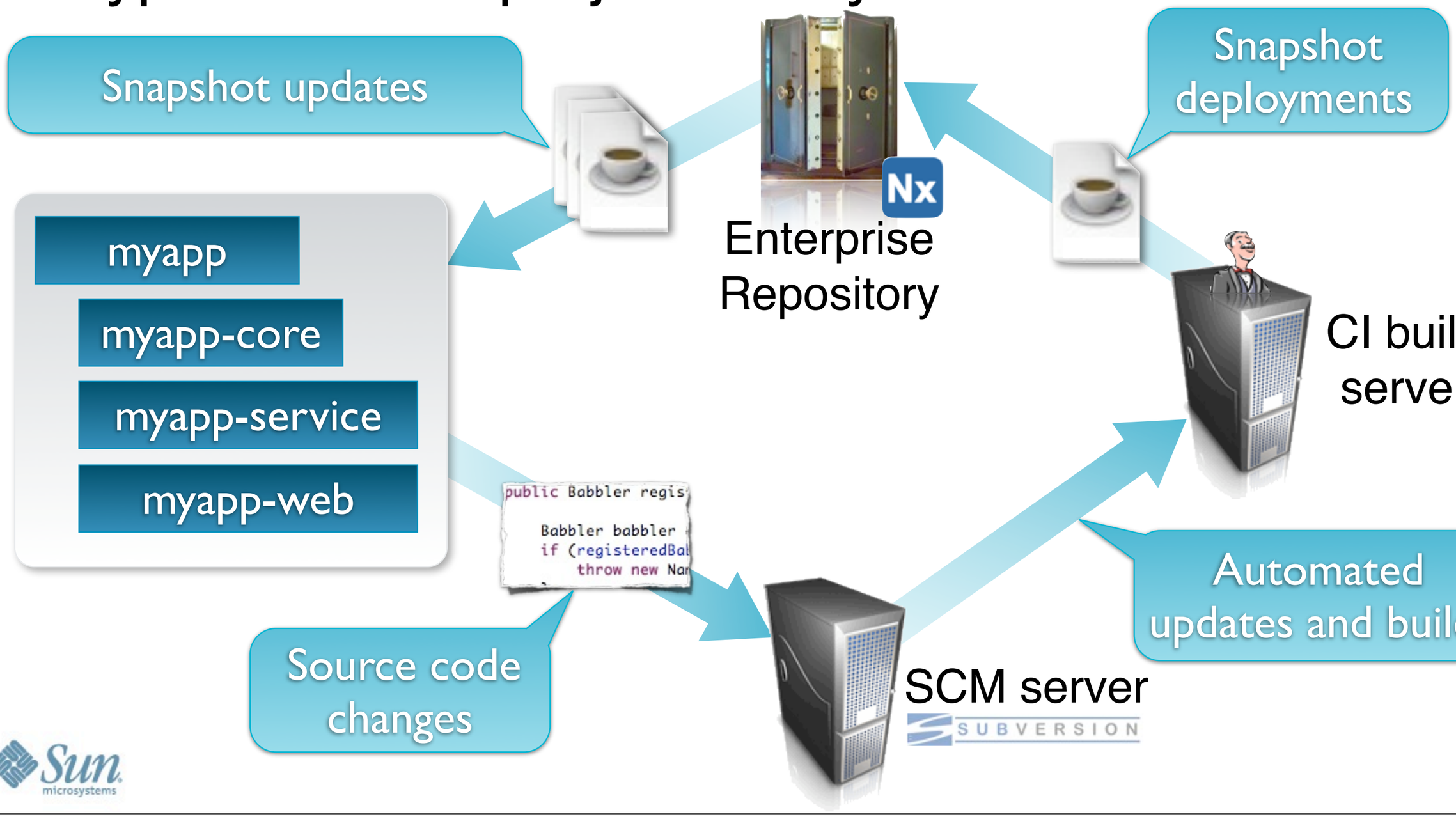
## Automated releases

- A typical Maven project ecosystem



# Revving up your release process
## Automated releases

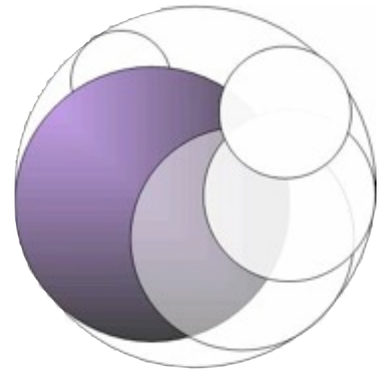
- A typical Maven project ecosystem



# Revving up your release process

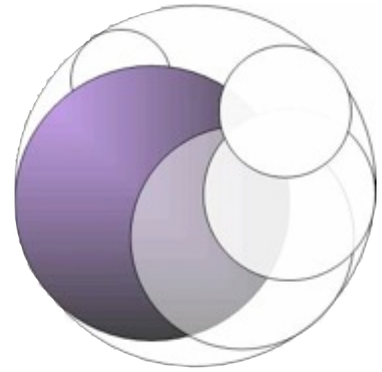
## Automated releases

- The Maven Release Cycle



# Revving up your release process
## Automated releases

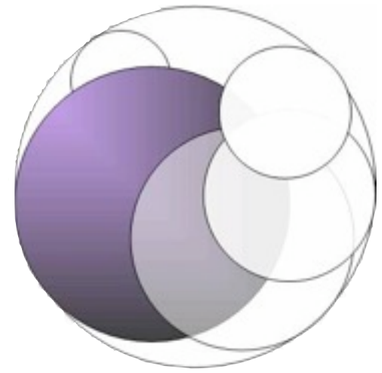
- The Maven Release Cycle
  - Develop against snapshot versions



```
<groupId>myorg.myapp</groupId>
<artifactId>myapp-web</artifactId>
<version>1.0.0-SNAPSHOT</version>
```

# Revving up your release process
## Automated releases

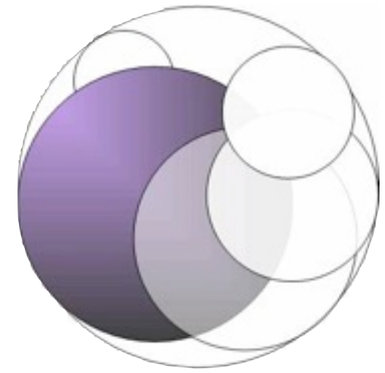
- The Maven Release Cycle
  - Develop against snapshot versions
    - Each release produces a new time-stamped artifact



```
<groupId>myorg.myapp</groupId>
<artifactId>myapp-web</artifactId>
<version>1.0.0-SNAPSHOT</version>
```

# Revving up your release process
## Automated releases

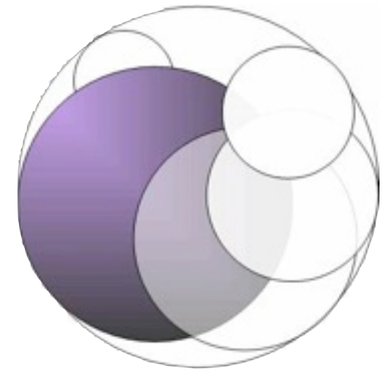
- The Maven Release Cycle
  - Develop against snapshot versions
    - Each release produces a new time-stamped artifact
    - Snapshot versions are identified by the SNAPSHOT keyword



```
<groupId>myorg.myapp</groupId>
<artifactId>myapp-web</artifactId>
<version>1.0.0-SNAPSHOT</version>
```

# Revving up your release process
## Automated releases

- The Maven Release Cycle
  - Develop against snapshot versions
    - Each release produces a new time-stamped artifact
    - Snapshot versions are identified by the SNAPSHOT keyword
  - Release stable versions

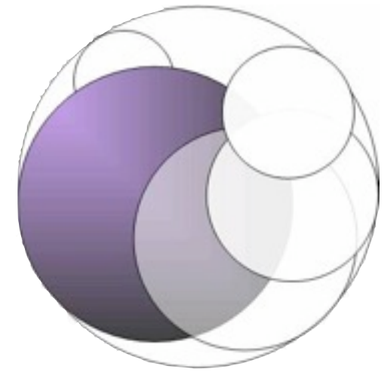


```
<groupId>myorg.myapp</groupId>
<artifactId>myapp-web</artifactId>
<version>1.0.0-SNAPSHOT</version>
```

```
<groupId>myorg.myapp</groupId>
<artifactId>myapp-web</artifactId>
<version>1.0.0</version>
```

# Revving up your release process
## Automated releases

- The Maven Release Cycle
  - Develop against snapshot versions
    - Each release produces a new time-stamped artifact
    - Snapshot versions are identified by the SNAPSHOT keyword
  - Release stable versions
    - Stable, tested release

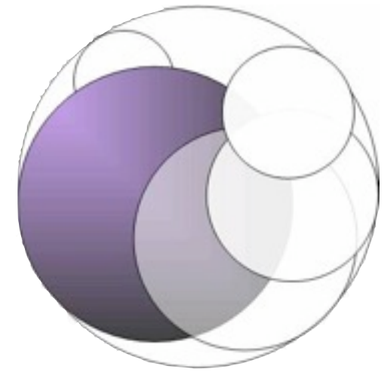


```
<groupId>myorg.myapp</groupId>
<artifactId>myapp-web</artifactId>
<version>1.0.0-SNAPSHOT</version>
```

```
<groupId>myorg.myapp</groupId>
<artifactId>myapp-web</artifactId>
<version>1.0.0</version>
```

# Revving up your release process
## Automated releases

- The Maven Release Cycle
  - Develop against snapshot versions
    - Each release produces a new time-stamped artifact
    - Snapshot versions are identified by the SNAPSHOT keyword
  - Release stable versions
    - Stable, tested release
    - Each artifact is unique



```
<groupId>myorg.myapp</groupId>
<artifactId>myapp-web</artifactId>
<version>1.0.0-SNAPSHOT</version>
```

```
<groupId>myorg.myapp</groupId>
<artifactId>myapp-web</artifactId>
<version>1.0.0</version>
```

# Revving up your release process

## Automated releases

- The Maven Release Cycle

# Revving up your release process
## Automated releases

- The Maven Release Cycle
  - Deploying a release involves many tasks

# Revving up your release process

## Automated releases

- The Maven Release Cycle
  - Deploying a release involves many tasks
    - Commit all current changes to version control

# Revving up your release process
## Automated releases

- The Maven Release Cycle
  - Deploying a release involves many tasks
    - Commit all current changes to version control
    - Upgrade the version number(s) and commit the changes

# Revving up your release process
## Automated releases

- The Maven Release Cycle
  - Deploying a release involves many tasks
    - Commit all current changes to version control
    - Upgrade the version number(s) and commit the changes
    - Tag the release version

# Revving up your release process
## Automated releases

- The Maven Release Cycle
  - Deploying a release involves many tasks
    - Commit all current changes to version control
    - Upgrade the version number(s) and commit the changes
    - Tag the release version
    - Build and deploy this version

# Revving up your release process
## Automated releases

- The Maven Release Cycle
  - Deploying a release involves many tasks
    - Commit all current changes to version control
    - Upgrade the version number(s) and commit the changes
    - Tag the release version
    - Build and deploy this version
    - Update the version number and commit the changes

# Revving up your release process

## Automated releases

- The Maven Release Plugin

# Revving up your release process
## Automated releases

- The Maven Release Plugin
  - Automates the release process

# Revving up your release process
## Automated releases

- The Maven Release Plugin
  - Automates the release process
  - You will need:

# Revving up your release process

## Automated releases

- The Maven Release Plugin
  - Automates the release process
  - You will need:
    - A valid SCM configuration

```
<scm>
  <connection>scm:git:git://github.com/wakaleo/babble.git</connection>
  <developerConnection>scm:git:git://github.com/wakaleo/babble.git
  </developerConnection>
</scm>
```

# Revving up your release process

## Automated releases

- The Maven Release Plugin
  - Automates the release process
  - You will need:
    - A valid SCM configuration
    - The maven-release-plugin

```
<scm>
  <connection>scm:git:git://github.com/wakaleo/babble.git</connection>
  <developerConnection>scm:git:git://github.com/wakaleo/babble.git
  </developerConnection>
</scm>
```

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-release-plugin</artifactId>
  <configuration>
    <preparationGoals>clean verify install</preparationGoals>
  </configuration>
</plugin>
```

Sun microsystems

# Revving up your release process
## Automated releases

- The Maven Release Plugin
  - Automates the release process
  - You will need:
    - A valid SCM configuration
    - The maven-release-plugin

```
<scm>
  <connection>scm:git:git://github.com/wakaleo/babble.git</connection>
  <developerConnection>scm:git:git://github.com/wakaleo/babble.git
  </developerConnection>
</scm>
```

Need this for multi-module projects

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-release-plugin</artifactId>
  <configuration>
    <preparationGoals>clean verify install</preparationGoals>
  </configuration>
</plugin>
```

Sun microsystems

# Revving up your release process
## Automated releases

- Using the Maven Release Plugin


  - Audits your code...
    - Check that there is no uncommitted code
    - Check that there are no snapshot dependencies
    - Prompts for release tag, branch and new version numbers

# Revving up your release process
## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:prepare
```

  - Audits your code...
    - Check that there is no uncommitted code
    - Check that there are no snapshot dependencies
    - Prompts for release tag, branch and new version numbers

# Revving up your release process

## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:prepare
```

  - Audits your code...

# Revving up your release process

## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:prepare
```

  - Audits your code...
    - Check that there is no uncommitted code

# Revving up your release process
## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:prepare
```

  - Audits your code...
    - Check that there is no uncommitted code
    - Check that there are no snapshot dependencies

# Revving up your release process
## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:prepare
```

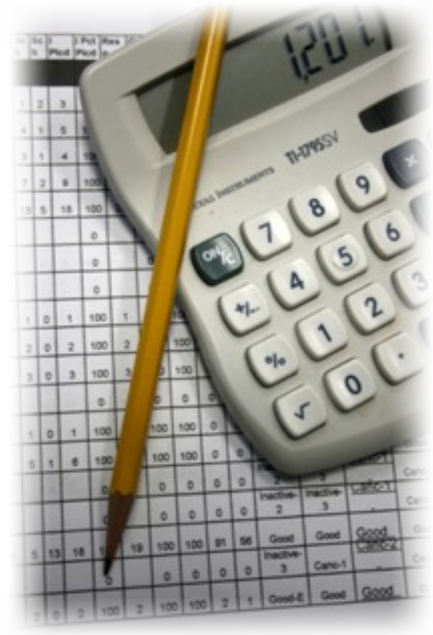
  - Audits your code...
    - Check that there is no uncommitted code
    - Check that there are no snapshot dependencies
    - Prompts for release tag, branch and new version numbers

# Revving up your release process

## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:prepare
```



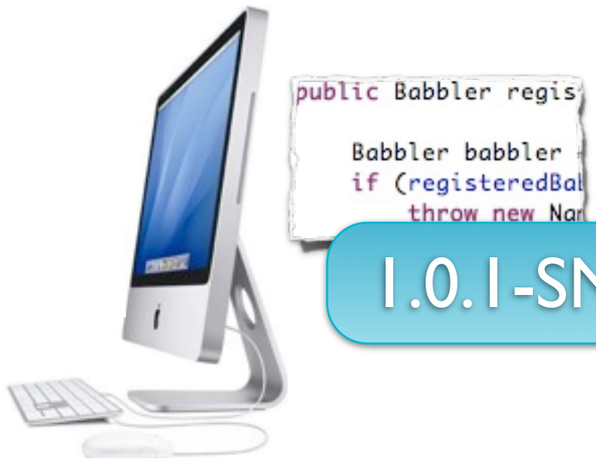
# Revving up your release process

## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:prepare
```

  - ...then does the bookkeeping



```
public Babbler regis

    Babbler babbler
    if (registeredBa
        throw new Na
```

1.0.1-SNAPSHOT

SCM server

SUBVERSION

Sun microsystems

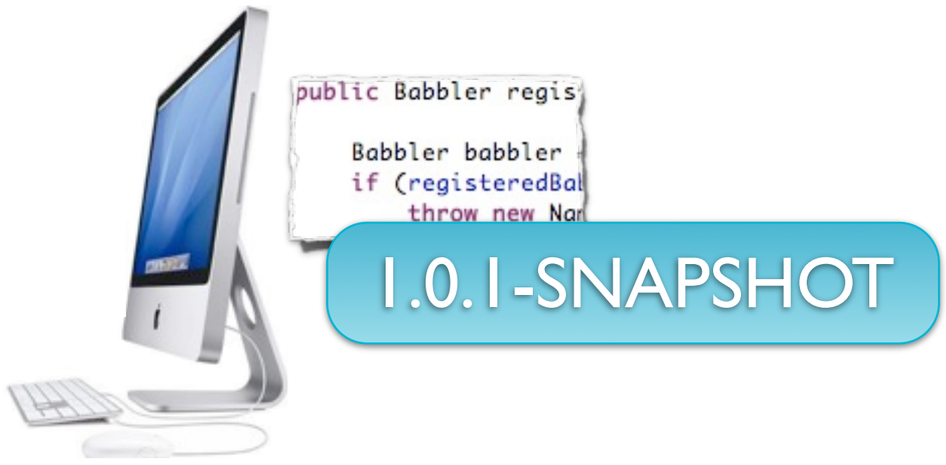
# Revving up your release process

## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:prepare
```

  - ...then does the bookkeeping
    - Updates version numbers to release versions




SCM server

SUBVERSION

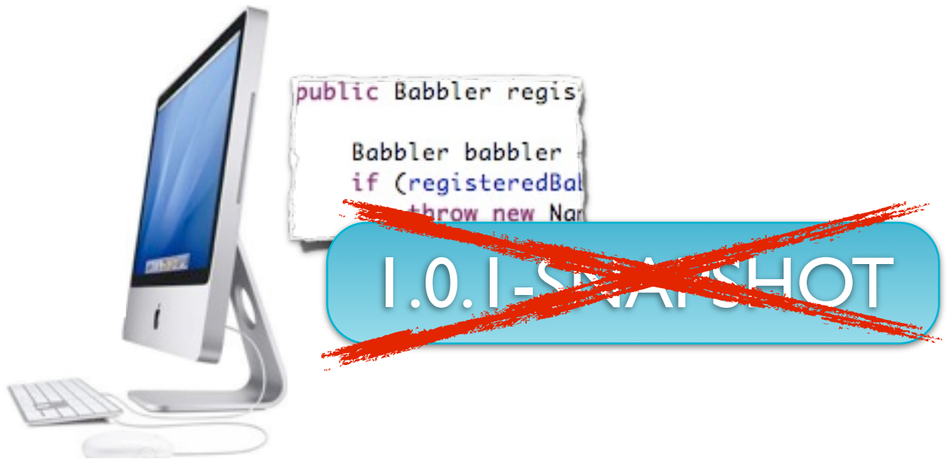
# Revving up your release process

## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:prepare
```

  - ...then does the bookkeeping
    - Updates version numbers to release versions




SCM server

SUBVERSION

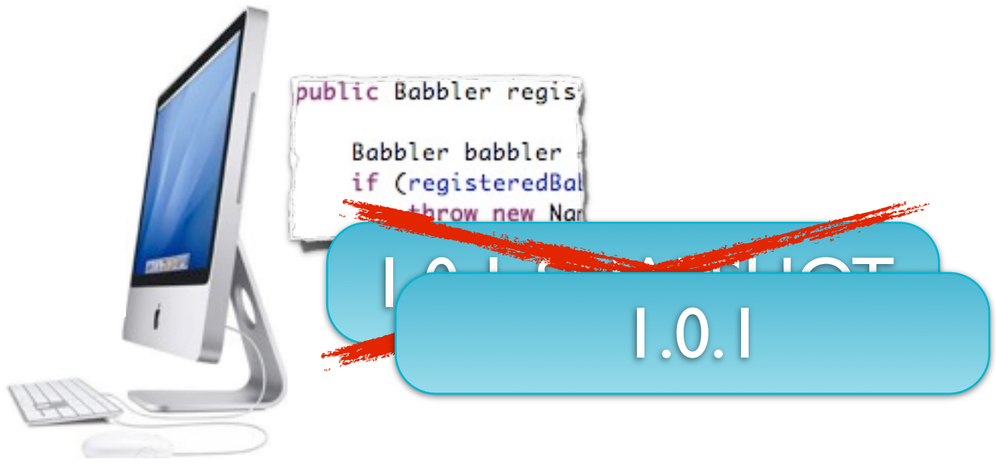
# Revving up your release process

## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:prepare
```

  - ...then does the bookkeeping
    - Updates version numbers to release versions




SCM server

SUBVERSION

Sun microsystems

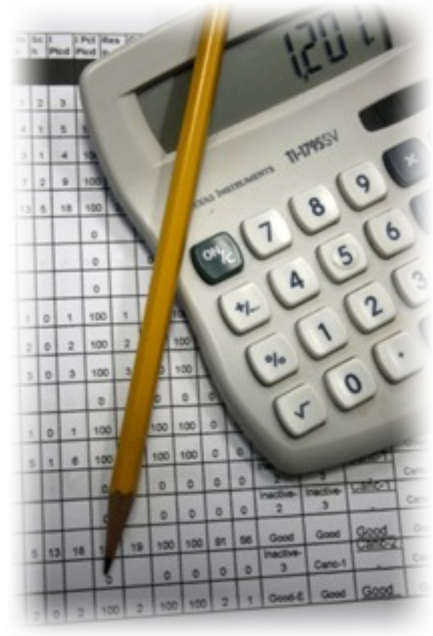
# Revving up your release process

## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:prepare
```

  - ...then does the bookkeeping
    - Updates version numbers to release versions
    - Creates a new release tag in SCM



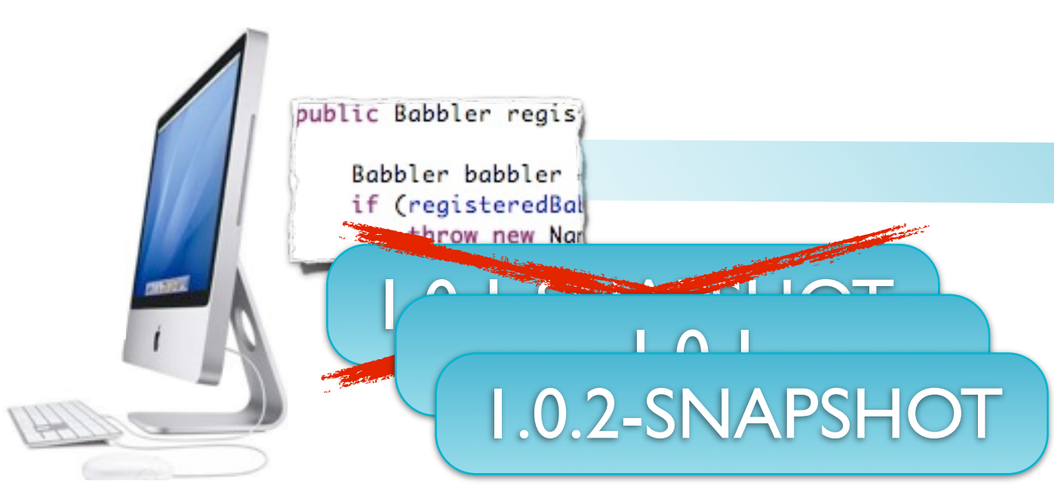
# Revving up your release process

## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:prepare
```

  - ...then does the bookkeeping
    - Updates version numbers to release versions
    - Creates a new release tag in SCM
    - Updates version numbers to next SNAPSHOT versions





SCM server

SUBVERSION

Sun microsystems

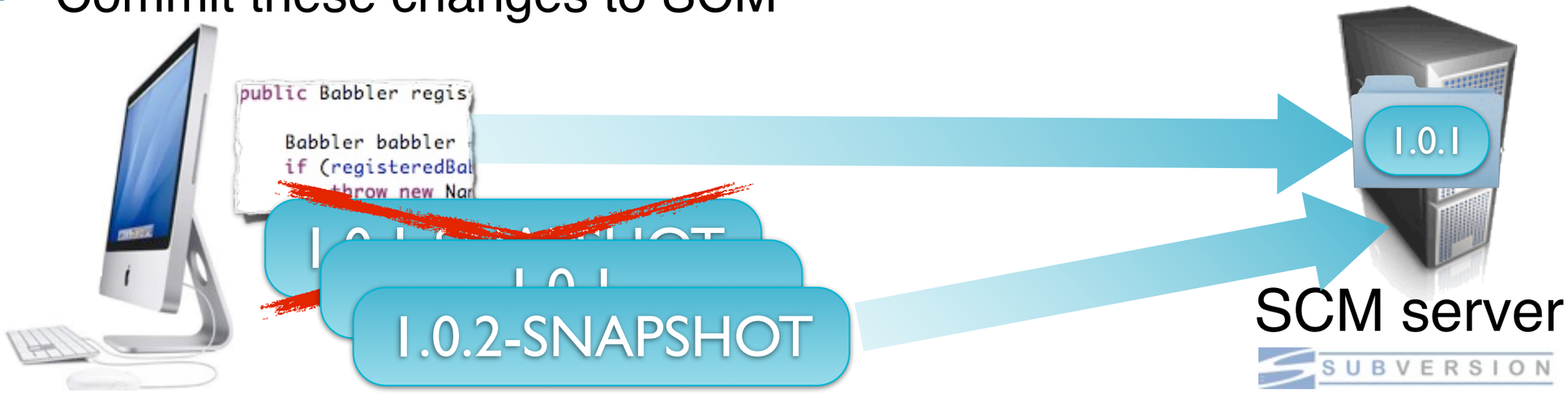
# Revving up your release process

## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:prepare
```

  - ...then does the bookkeeping
    - Updates version numbers to release versions
    - Creates a new release tag in SCM
    - Updates version numbers to next SNAPSHOT versions
    - Commit these changes to SCM




Sun microsystems

# Revving up your release process
## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:prepare -B
```

No questions asked

  - ...then does the bookkeeping
    - Updates version numbers to release versions
    - Creates a new release tag in SCM
    - Updates version numbers to next SNAPSHOT versions
    - Commit these changes to SCM


Sun microsystems

# Revving up your release process

## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:perform
```

SCM server

SUBVERSION


Enterprise Repository

Sun microsystems

# Revving up your release process

## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:perform
```

  - Do the real work

SCM server

SUBVERSION


Enterprise Repository

Sun microsystems

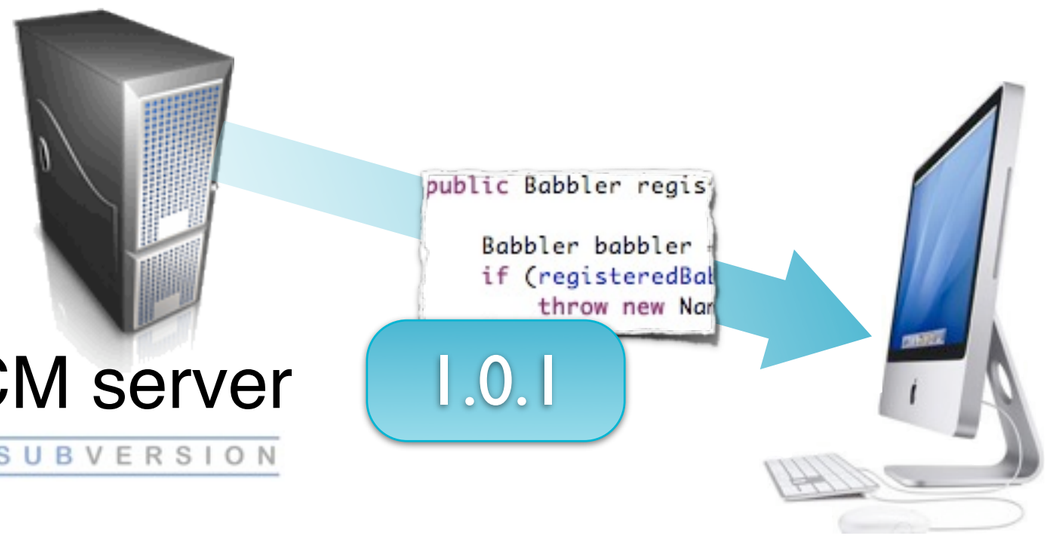
# Revving up your release process

## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:perform
```

  - Do the real work
    - Checkout a release version


SCM server

SUBVERSION


Enterprise Repository

Sun microsystems

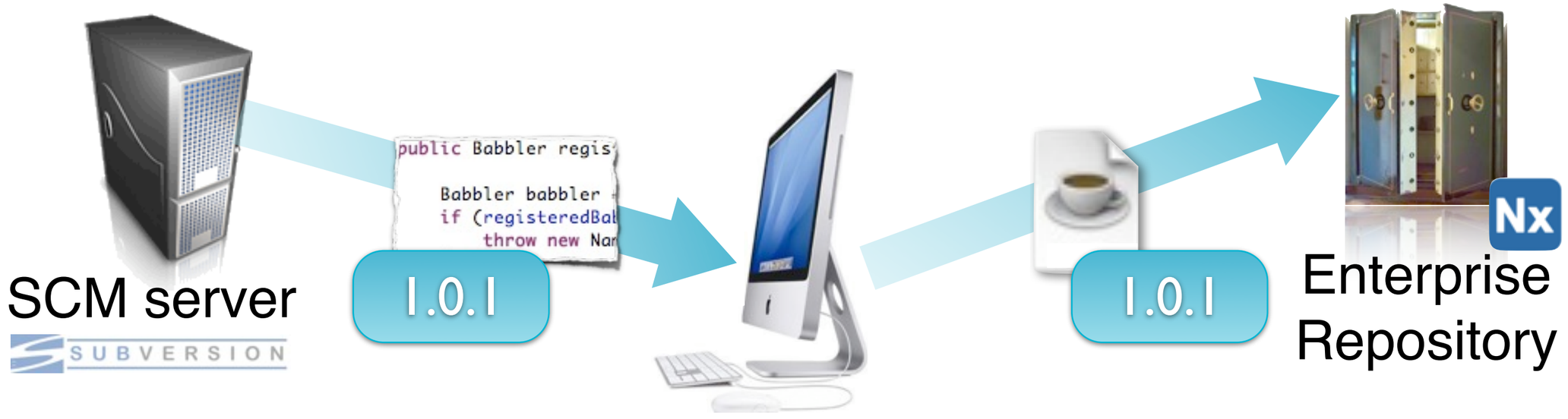
# Revving up your release process

## Automated releases

- Using the Maven Release Plugin

```
$ mvn release:perform
```

  - Do the real work
    - Checkout a release version
    - Build, test and deploy to the Enterprise repository

# Revving up your release process

## One-click automated releases

- Automating the Maven Release Plugin

# Revving up your release process

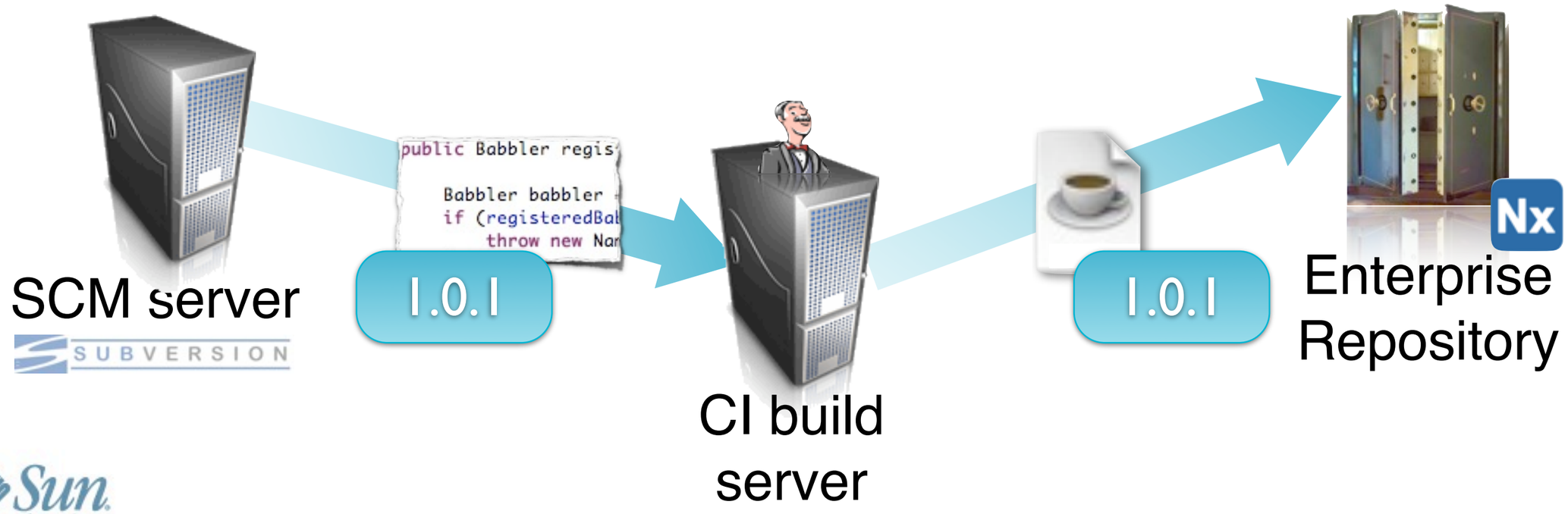
## One-click automated releases

- Automating the Maven Release Plugin
  - Don’t use the maven-release-plugin on your own machine

# Revving up your release process

## One-click automated releases

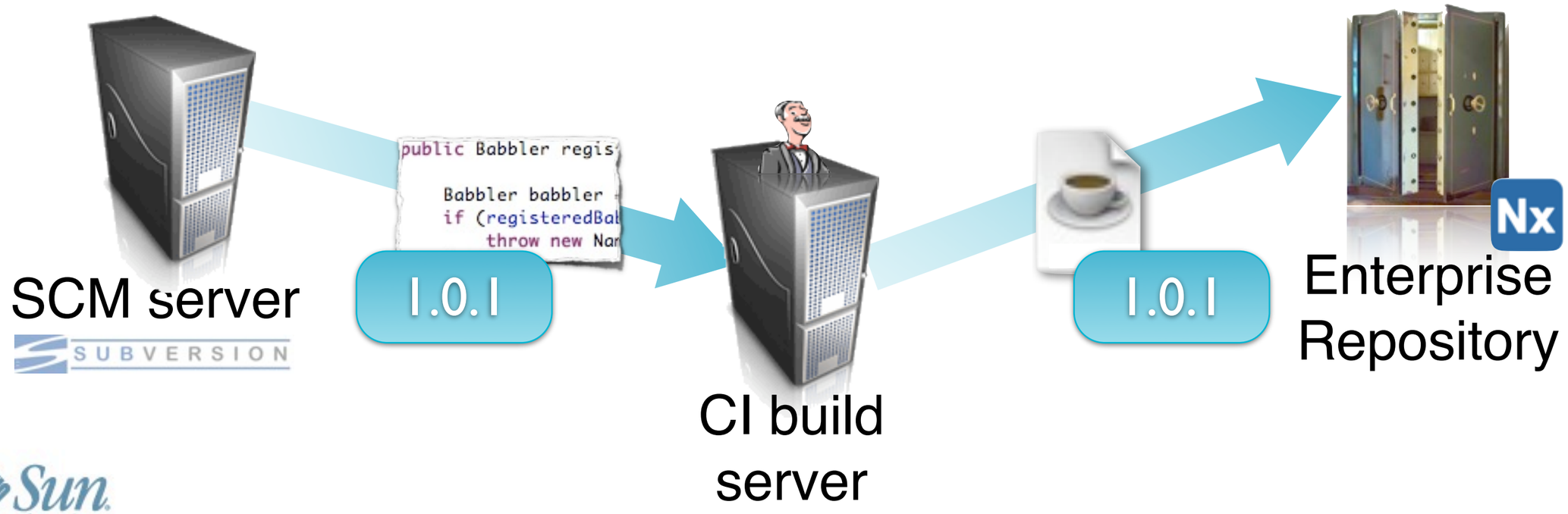
- Automating the Maven Release Plugin
  - Don’t use the maven-release-plugin on your own machine
  - Run it on a build server instead



# Revving up your release process

## One-click automated releases

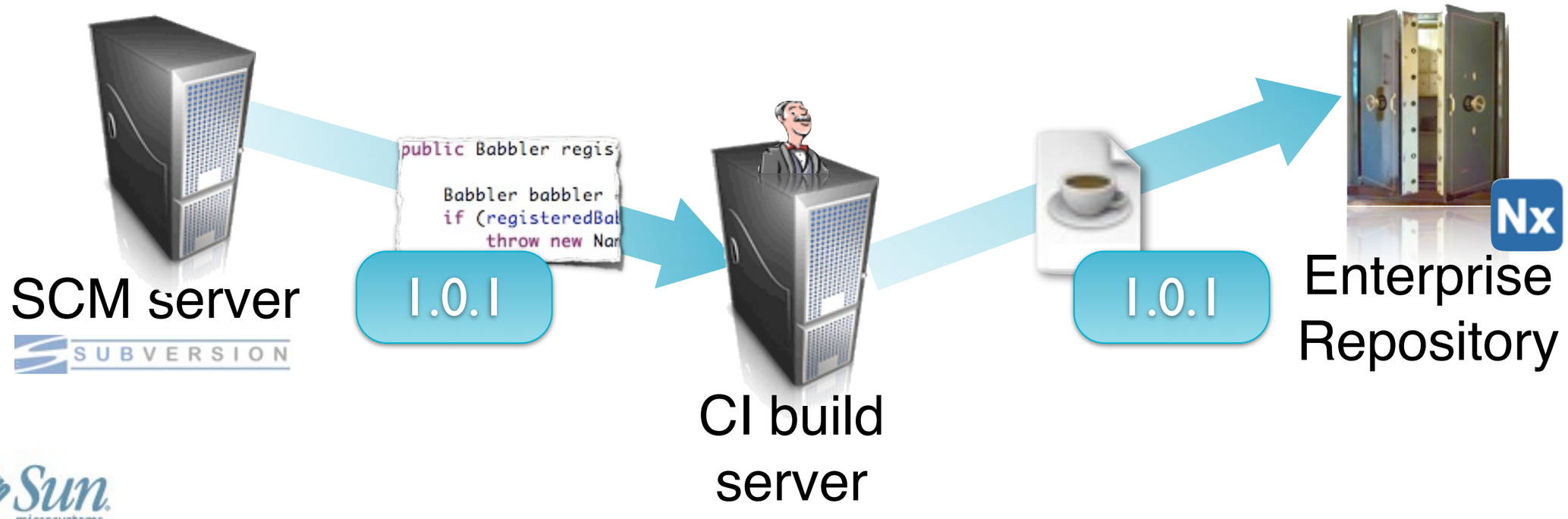
- Automating the Maven Release Plugin
  - Don’t use the maven-release-plugin on your own machine
  - Run it on a build server instead
    - Accessible to all



# Revving up your release process

## One-click automated releases

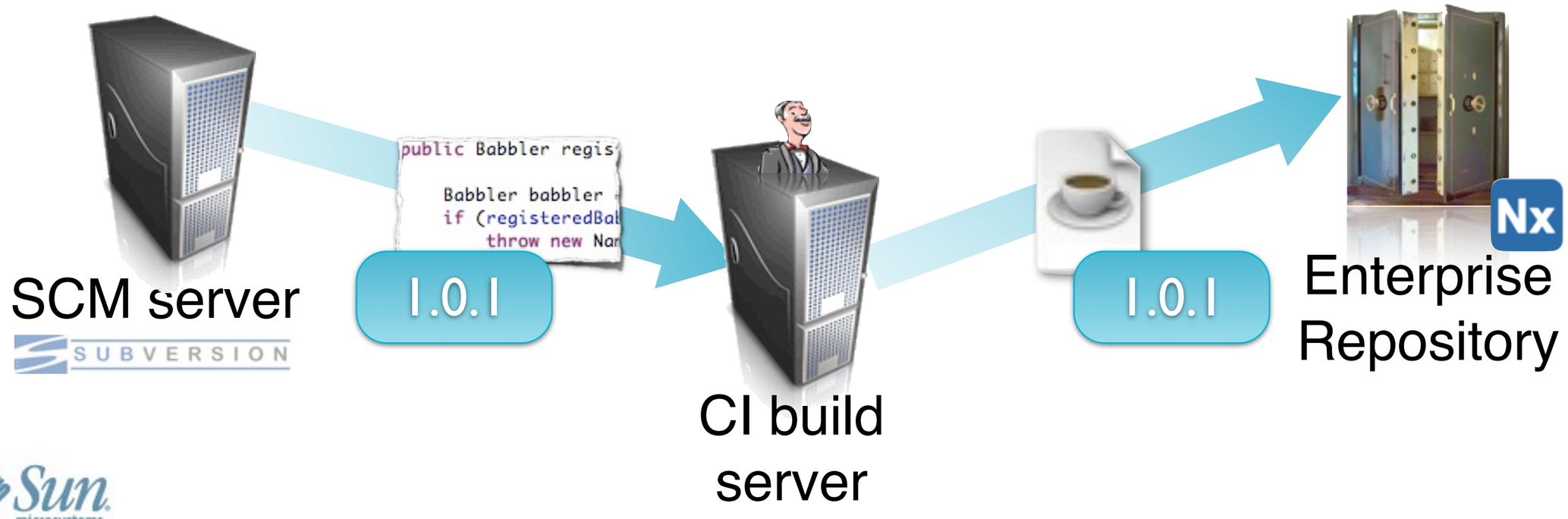
- Automating the Maven Release Plugin
  - Don't use the maven-release-plugin on your own machine
  - Run it on a build server instead
    - Accessible to all
    - Reference environment



# Revving up your release process

## One-click automated releases

- Automating the Maven Release Plugin
  - Don't use the maven-release-plugin on your own machine
  - Run it on a build server instead
    - Accessible to all
    - Reference environment
    - No risk of local code changes



# Doing up your deployment process

## One-click automated deployments

- Automating the deployment process


Enterprise Repository

Sun microsystems

# Doing up your deployment process

## One-click automated deployments

- Automating the deployment process
  - Don’t rebuild, reuse!


Enterprise Repository

Sun microsystems

# Doing up your deployment process

## One-click automated deployments

- Automating the deployment process
  - Don’t rebuild, reuse!
    - Download application binaries from the Enterprise repository

# Doing up your deployment process

## One-click automated deployments

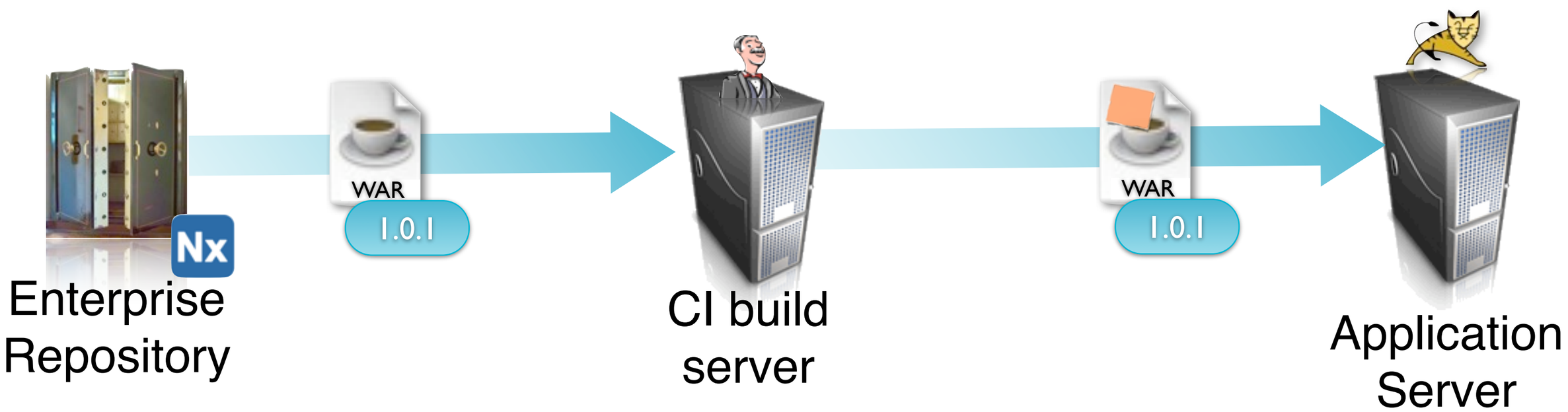
- Automating the deployment process
  - Don’t rebuild, reuse!
    - Download application binaries from the Enterprise repository
    - Patch if necessary with target environment configuration


Sun microsystems

# Doing up your deployment process

## One-click automated deployments

- Automating the deployment process
  - Don’t rebuild, reuse!
    - Download application binaries from the Enterprise repository
    - Patch if necessary with target environment configuration
    - Deploy to target environment



# Doing up your deployment process

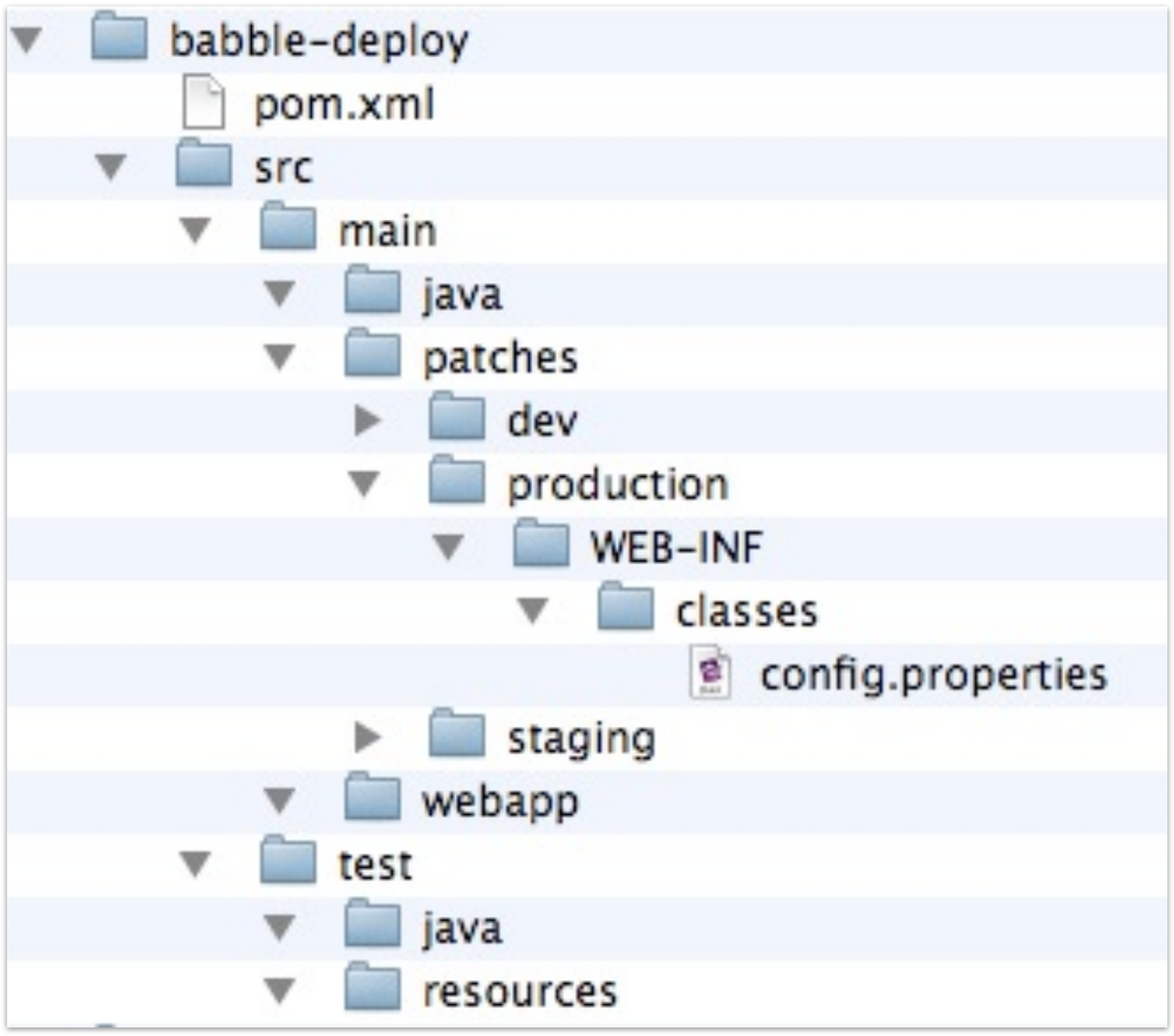
## Example - deploying a web application

- Deploying a web application in 5 easy steps!

# Doing up your deployment process

## Example - deploying a web application

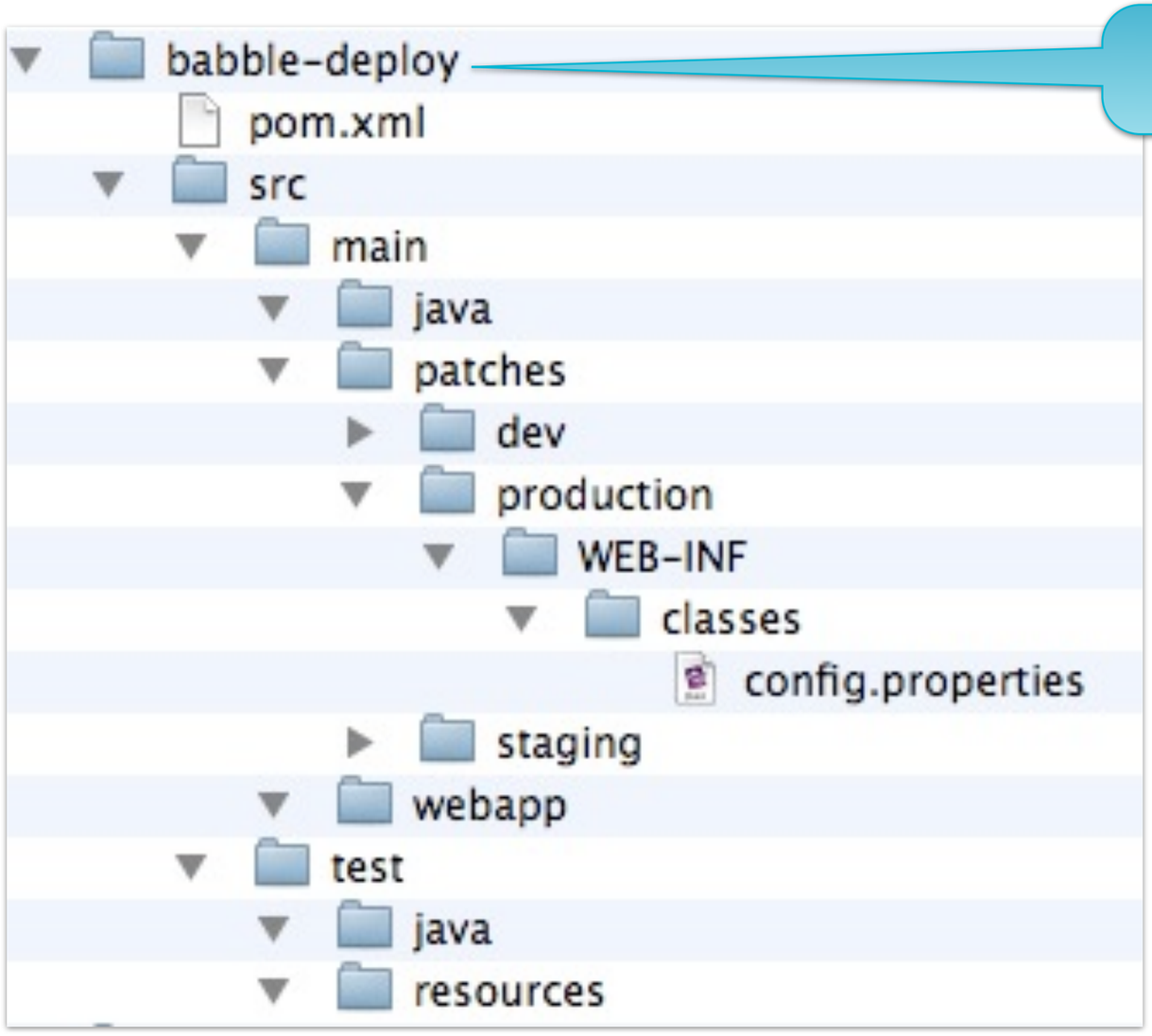
- Deploying a web application in 5 easy steps!



# Doing up your deployment process

## Example - deploying a web application

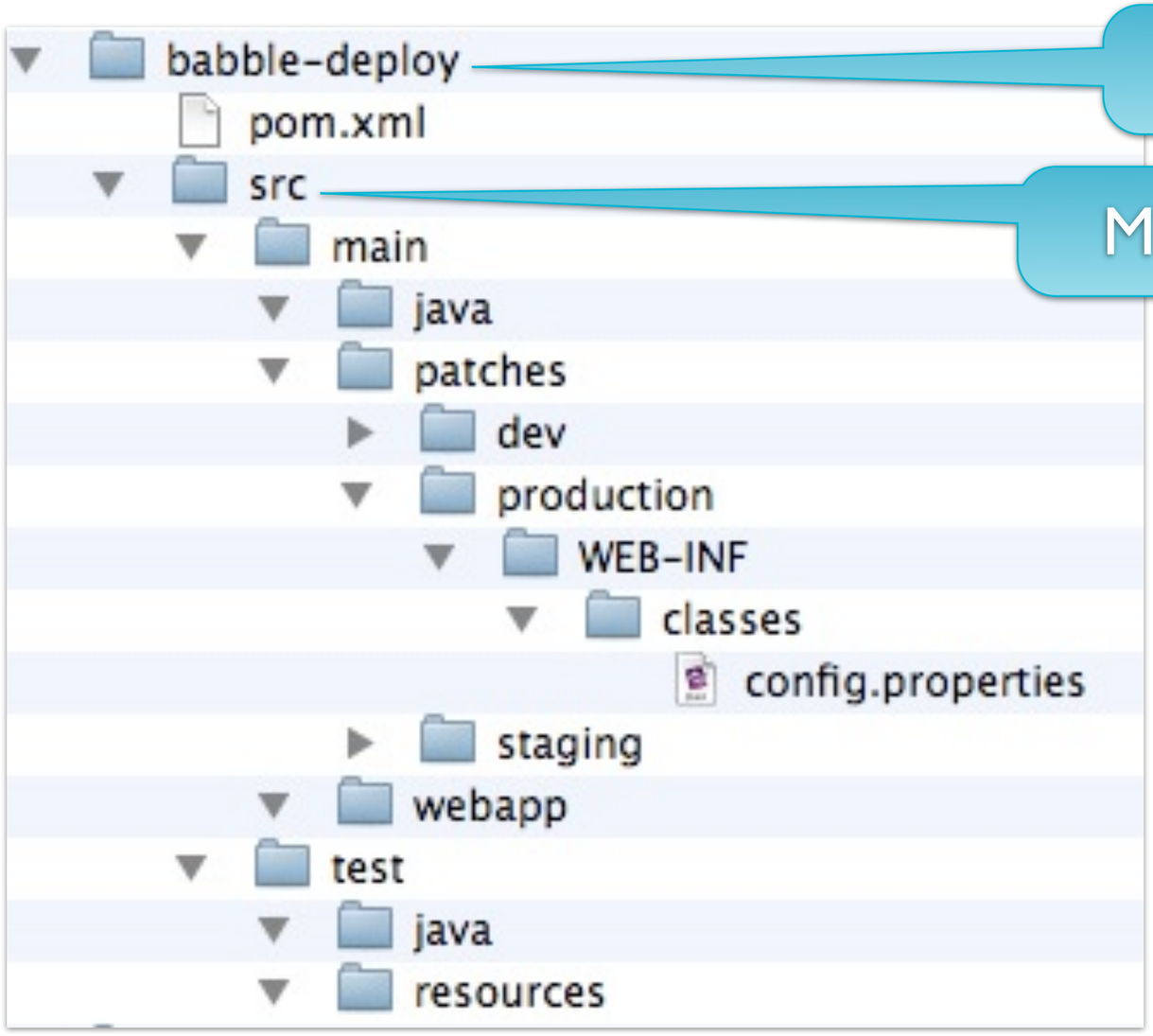
- Deploying a web application in 5 easy steps!



# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!



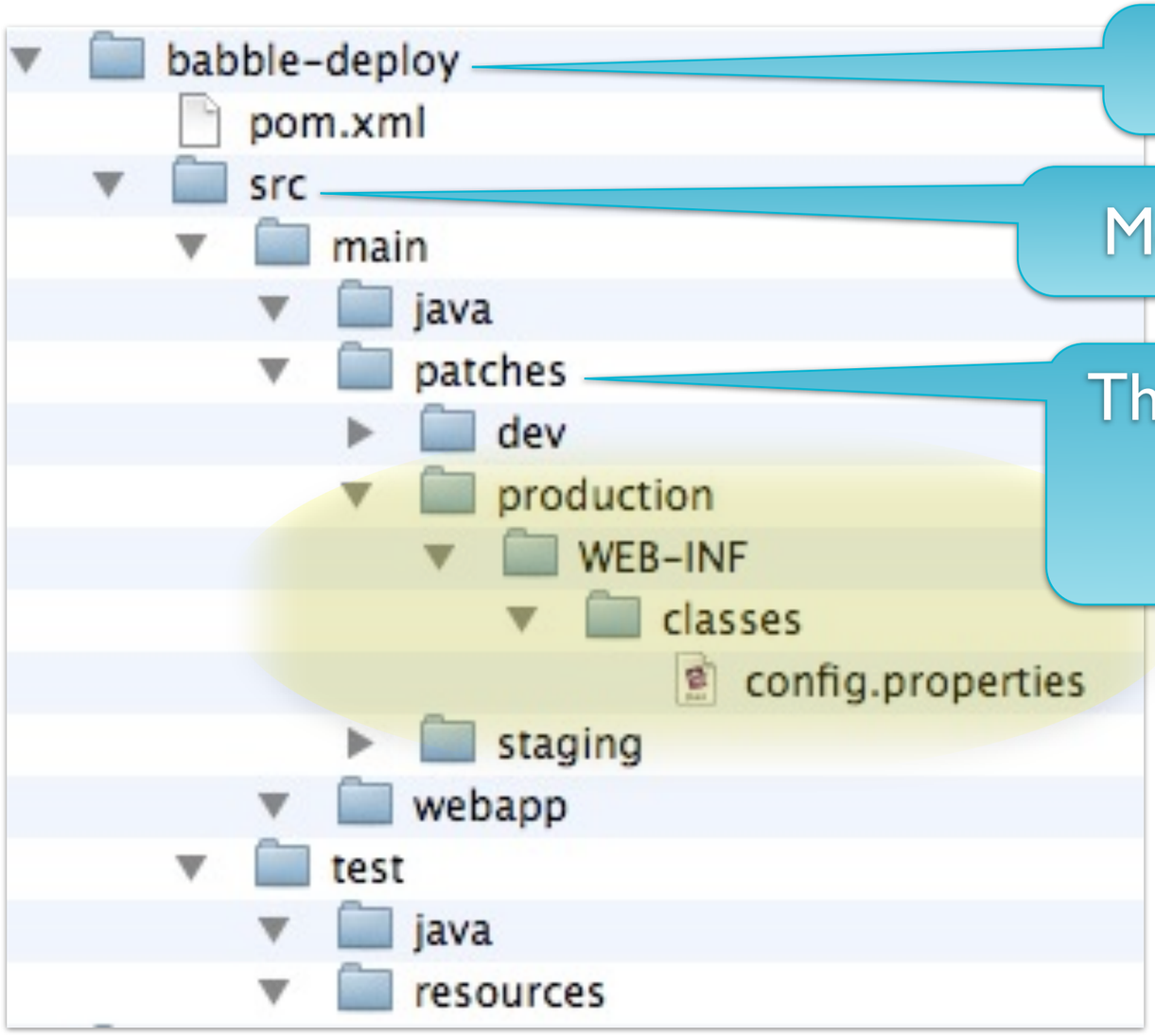

Create a deployment project

Most of the directories are empty

# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!




Create a deployment project

Most of the directories are empty

These are the files that will patch the web application for different environments

# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!

# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!

```
<dependency>
  <groupId>myorg.myapp</groupId>
  <artifactId>myapp-web</artifactId>
  <version>${webapp.version}</version>
  <type>war</type>
</dependency>
...
<properties>
  <webapp.version>${project.version}</webapp.version>
</properties>
```

# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!

```
<dependency>
  <groupId>myorg.myapp</groupId>
  <artifactId>myapp-web</artifactId>
  <version>${webapp.version}</version>
  <type>war</type>
</dependency>
...
<properties>
  <webapp.version>${project.version}</webapp.version>
</properties>
```

Add a parameterized dependency to your WAR

# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!

```
<dependency>
  <groupId>myorg.myapp</groupId>
  <artifactId>myapp-web</artifactId>
  <version>${webapp.version}</version>
  <type>war</type>
</dependency>
...
<properties>
  <webapp.version>${project.version}</webapp.version>
</properties>
```

Add a parameterized dependency to your WAR

Provide a sensible default target version

# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!

# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!

```
<plugin>
  <artifactId>maven-war-plugin</artifactId>
  <configuration>
    <warName>myapp-web</warName>
    <overlays>
      <overlay>
        <groupId>myorg.myapp</groupId>
        <artifactId>myapp-web</artifactId>
        <excludes>
          <exclude>WEB-INF/classes/config.properties</exclude>
        </excludes>
      </overlay>
      ...
```

# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!

```
<plugin>
  <artifactId>maven-war-plugin</artifactId>
  <configuration>
    <warName>myapp-web</warName>
    <overlays>
      <overlay>
        <groupId>myorg.myapp</groupId>
        <artifactId>myapp-web</artifactId>
        <excludes>
          <exclude>WEB-INF/classes/config.properties</exclude>
        </excludes>
      </overlay>
      ...
```

Incorporate the web application, but exclude the files to be patched

# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!

# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!

```
<plugin>
  <artifactId>maven-war-plugin</artifactId>
  <configuration>
    ...
    <webResources>
      <resource>
        <directory>${patch.path}</directory>
      </resource>
    </webResources>
  </configuration>
</plugin>
...
```

# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!

```
<plugin>
  <artifactId>maven-war-plugin</artifactId>
  <configuration>
    ...
    <webResources>
      <resource>
        <directory>${patch.path}</directory>
      </resource>
    </webResources>
  </configuration>
</plugin>
...
```

Include patched files in the final bundle

# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!

# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!

```
<profile>
  <id>dev</id>
  <properties>
    <patch.path>src/main/patches/dev</patch.path>
  </properties>
</profile>
```

# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!

```
<profile>
  <id>dev</id>
  <properties>
    <patch.path>src/main/patches/dev</patch.path>
  </properties>
</profile>
```

The exact patch files is defined in profiles

# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!

```
<profile>
  <id>dev</id>
  <properties>
    <patch.path>src/main/patches/dev</patch.path>
  </properties>
</profile>
```

The exact patch files is defined in profiles

```
<profile>
  <id>staging</id>
  <properties>
    <patch.path>src/main/patches/staging</patch.path>
    <webapp.version>${target.version}</webapp.version>
  </properties>
</profile>
```

# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!

```
<profile>
  <id>dev</id>
  <properties>
    <patch.path>src/main/patches/dev</patch.path>
  </properties>
</profile>
```

The exact patch files is defined in profiles

```
<profile>
  <id>staging</id>
  <properties>
    <patch.path>src/main/patches/staging</patch.path>
    <webapp.version>${target.version}</webapp.version>
  </properties>
</profile>
```

For staging and production, you can also override the version being deployed

# Doing up your deployment process

## Example - deploying a web application

- Deploying a web application in 5 easy steps!

```
<profile>
  <id>dev</id>
  <properties>
    <patch.path>src/main/patches/dev</patch.path>
  </properties>
</profile>
```

The exact patch files is defined in profiles

```
<profile>
  <id>staging</id>
  <properties>
    <patch.path>src/main/patches/staging</patch.path>
    <webapp.version>${target.version}</webapp.version>
  </properties>
</profile>
```

For staging and production, you can also override the version being deployed

```
$ mvn package -Dtarget.version=1.0.5 -Pstaging
```

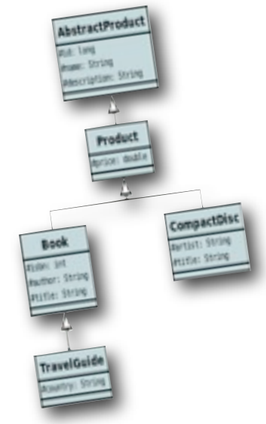
# Conclusion

## What have we learnt?

# Conclusion

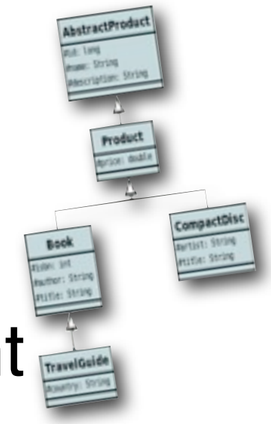
## What have we learnt?

- Refactor your pom files



# Conclusion

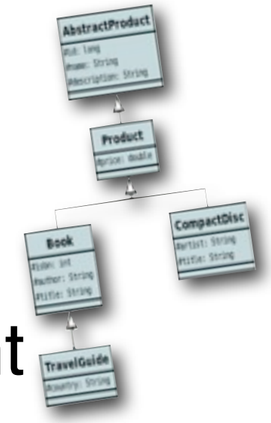
## What have we learnt?




- Refactor your pom files
  - Use tools like parent projects and dependency management to make your pom files cleaner and more maintainable

# Conclusion

## What have we learnt?



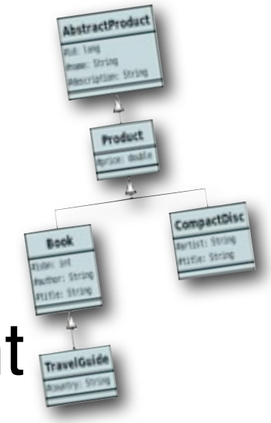

- Refactor your pom files
  - Use tools like parent projects and dependency management to make your pom files cleaner and more maintainable

- Multi-module projects rule!

CI build server

# Conclusion

## What have we learnt?



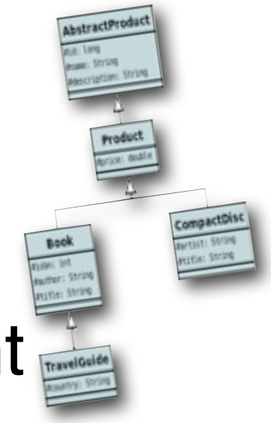

- Refactor your pom files
  - Use tools like parent projects and dependency management to make your pom files cleaner and more maintainable

- Multi-module projects rule!
  - Using multi-module projects to improve your architecture and speed up your builds

CI build server

# Conclusion

## What have we learnt?




- Refactor your pom files
  - Use tools like parent projects and dependency management to make your pom files cleaner and more maintainable

- Multi-module projects rule!
  - Using multi-module projects to improve your architecture and speed up your builds

CI build server

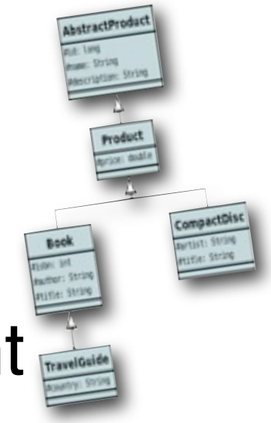
- Nexus as a hub of developer activity


Enterprise Repository

Sun microsystems

# Conclusion

## What have we learnt?




- Refactor your pom files
  - Use tools like parent projects and dependency management to make your pom files cleaner and more maintainable

- Multi-module projects rule!
  - Using multi-module projects to improve your architecture and speed up your builds

CI build server

- Nexus as a hub of developer activity
  - Using Maven, Nexus and Hudson to implement one-click releases and deployments


Enterprise Repository

Sun microsystems


# JavaOne℠

# Thank You

John Ferguson Smart
Email: john.smart@wakaleo.com
Web: http://www.wakaleo.com
Twitter: wakaleo