Building J2EE projects with Maven

Vincent Massol, 27th June 2003

Agenda

- What is Maven and how it works?
- Implementing a J2EE example
- Continuous Integration with Maven
- Extending Maven
- Tips and future directions

Agenda

- What is Maven and how it works?
- Implementing a J2EE example
- Continuous Integration with Maven
- Extending Maven
- Tips and future directions

What is Maven?

- A build tool one level above Ant
- Ant
  - Write the build script
  - Run the targets
- Maven
  - Describe the project and configure plugins
  - Run existing plugins

Project Object Model (POM)

```
<project>
  <id>junitbook-sampling</id>
  <name>JUnit in Action - Sampling JUnit</name>
  <currentVersion>1.0</currentVersion>
  <organization/>
  <inceptionYear>2002-2003</inceptionYear>
  <package>junitbook.sampling</package>
  <logo>/images/jia.jpg</logo>

  <description>[…]</description>
  <shortDescription>[…]</shortDescription>

  <url>http://sourceforge.net/projects/junitbook/</url>

  <developers/>
  <dependencies/>

  <build>
    <sourceDirectory>src/java</sourceDirectory>
    <unitTestSourceDirectory>src/test</unitTestSourceDirectory>
    <unitTest>
      <includes/>
      <excludes/>
    </unitTest>
  </build>

</project>
```

Maven Plugins

- ant
- antlr
- appserver
- ashkelon
- aspectj
- cactus
- castor
- changelog
- changes
- checkstyle
- clean
- clover
- codeswitcher
- console
- deploy
- developer-activity
- dist
- docbook
- ear
- eclipse
- ejb
- examples
- faq
- file-activity
- genapp
- gump
- hibernate
- html2xdoc
- idea
- j2ee
- jalopy
- jar
- java
- javadoc
- jboss
- jbuilder
- jdee
- jdepend
- jdeveloper
- jdiff
- jellydoc
- jnlp
- junit-report
- junitdoclet
- jxr
- latex
- latka
- license
- linkcheck
- native
- pdf
- perforce
- plexus
- plugin
- pmd
- pom
- release
- repository
- runner
- sea
- shell
- site
- statcvs
- struts
- summit
- tasklist
- test
- torque
- touchstone
- touchstone-partner
- uberjar
- vdoclet
- war
- was40
- webserver
- wizard
- word2html
- xdoc

… and more

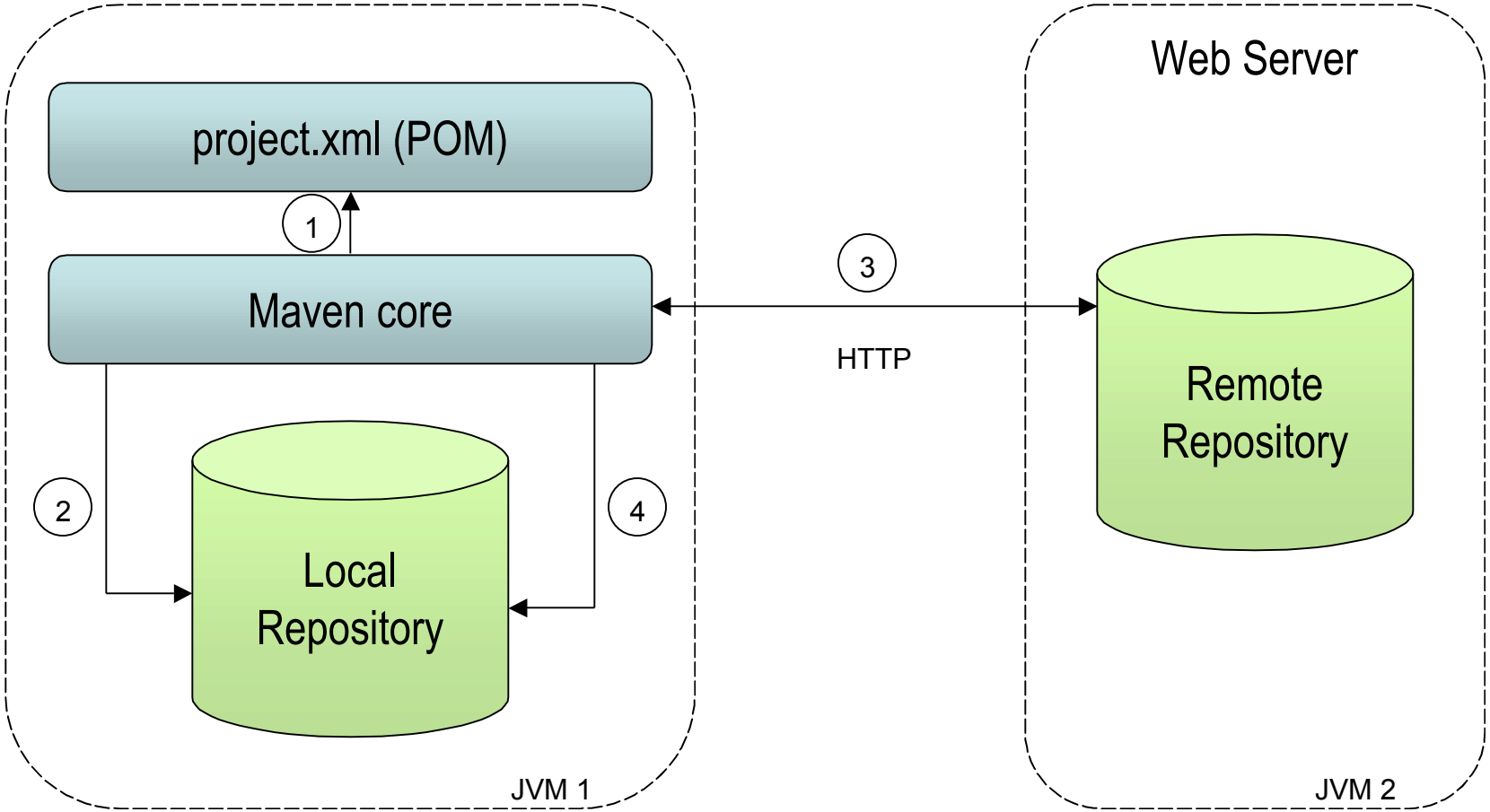
Maven reports (Site plugin)

Overview

Document	Description
Metrics	Report on source code metrics.
Checkstyle	Report on coding style conventions.
Change Log	Report on the source control changelog.
Developer Activity	Report on the amount of developer activity.
File Activity	Report on file activity.
Project License	Displays the primary license for the project.
JavaDocs	JavaDoc API documentation.
JavaDoc Report	Report on the generation of JavaDoc.
Source Xref	A set of browsable cross-referenced sources.
Test Xref	A set of browsable cross-referenced test sources.
Unit Tests	Report on the results of the unit tests.
Link Check Report	Report on the validity of all links in the documentation.
Task List	Report on tasks specified in the source code.

```
<project>
  [...]
  <reports>
    <report>maven-junit-report-plugin</report>
    <report>maven-checkstyle-plugin</report>
    [...]
  </reports>
</project>
```

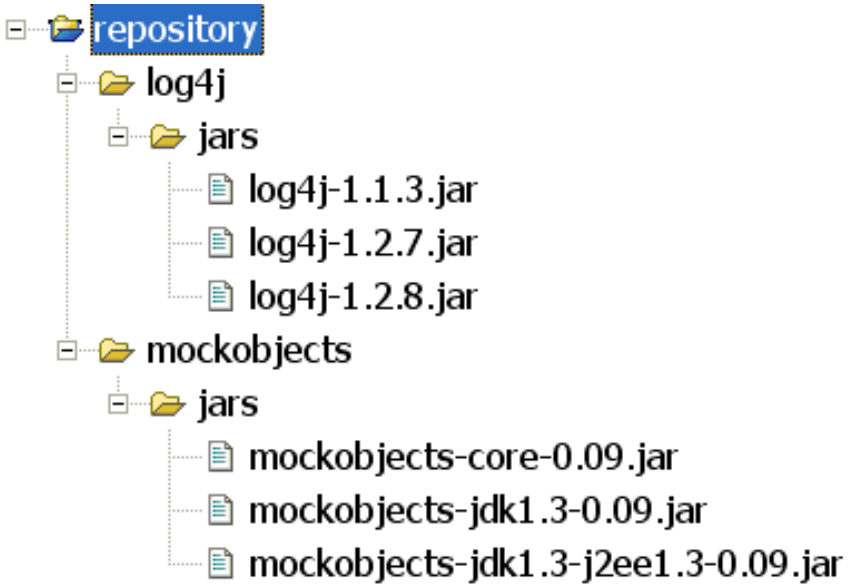
Maven Artifact Repositories (1/2)



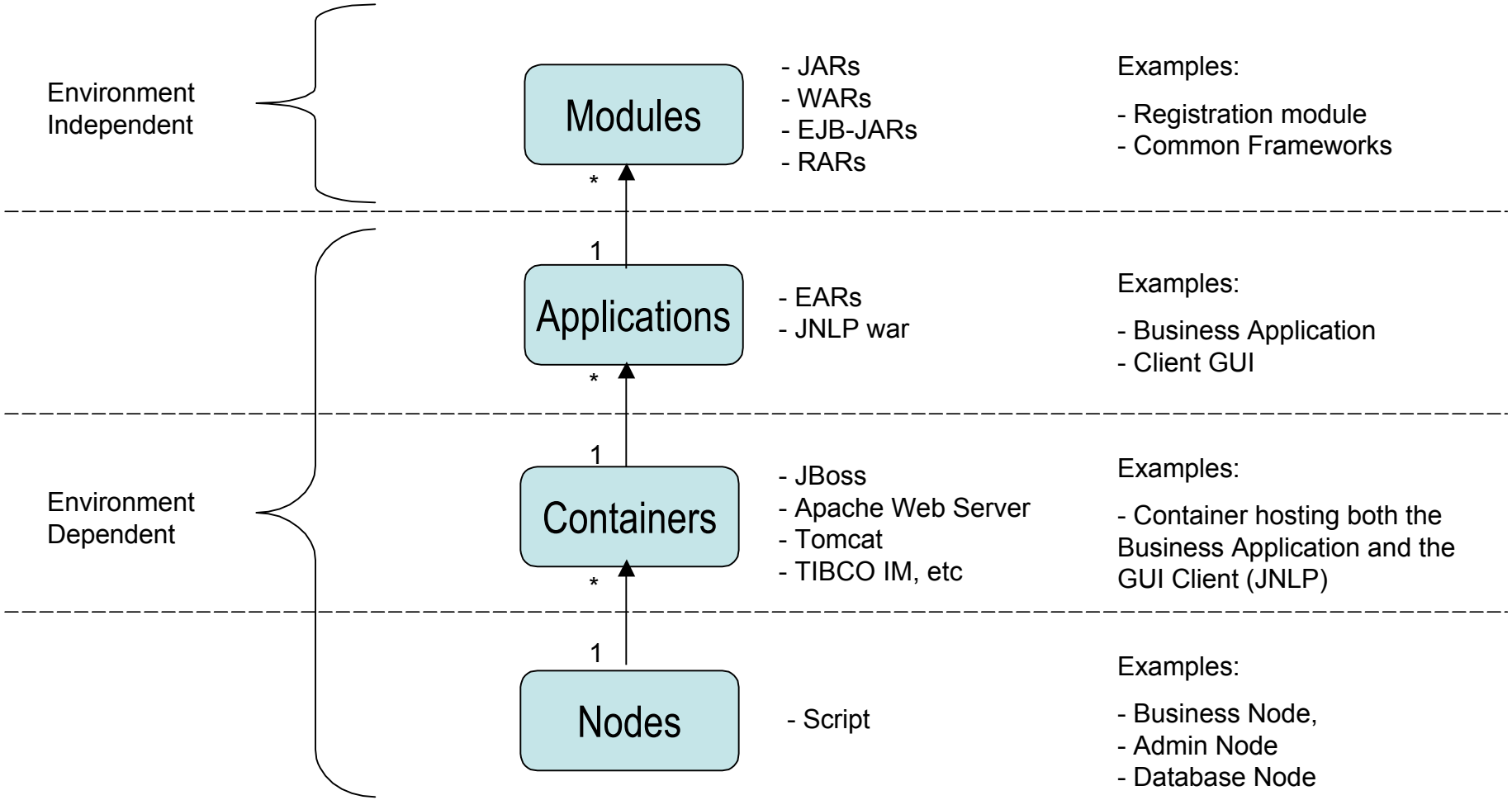
Maven Artifact Repositories (2/2)

```
<project>
  [...]
  <dependencies>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.8</version>
      <type>jar</type>
    </dependency>
    [...]
  </dependencies>
  [...]
</project>
```

```
%MAVEN_REPO%/
<groupId>/<type>s/<artifactId>-<version>.<type>
```



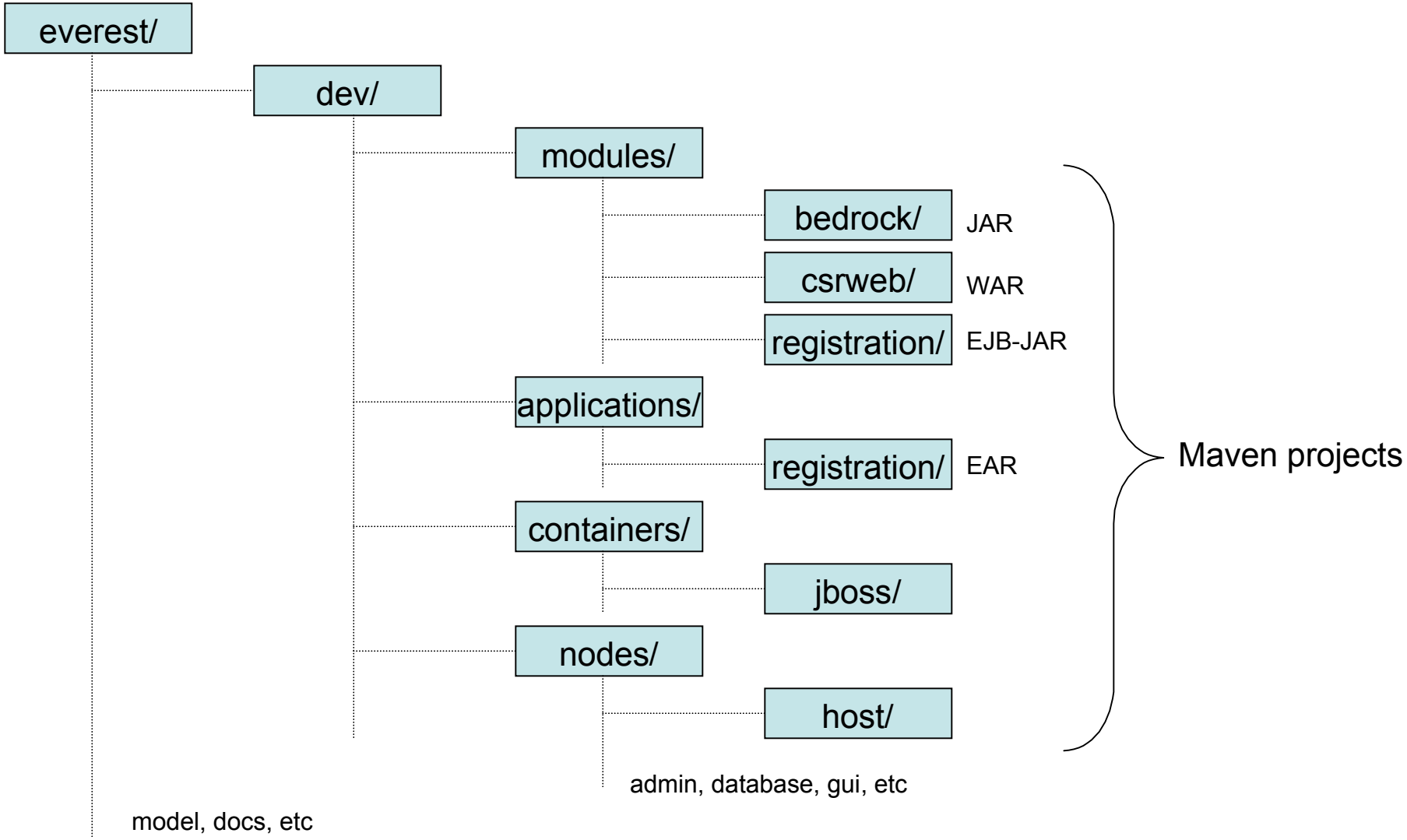
Project Organization



Agenda

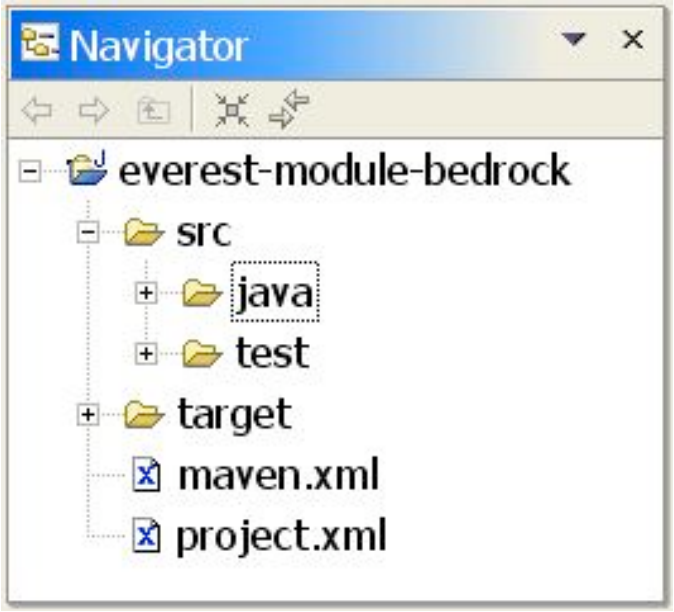
- What is Maven and how it works?
- Implementing a J2EE example
- Continuous Integration with Maven
- Extending Maven
- Tips and future directions

Sample Everest project



Bedrock – A simple jar project

- `project.xml`
  - Maven POM
- `project.properties` (optional)
  - Plugin configurations
- `build.properties` (optional)
  - Configuration specific to local machine
- `maven.xml` (optional)
  - Custom build goals/tasks
- `target/`
  - Temp directory generated by Maven
- `xdocs/` (optional)
  - Project documentation written in XML format
- `src/java/`
  - Runtime java classes
- `src/test/`
  - JUnit tests
- `src/conf/` (optional)
  - Configuration files

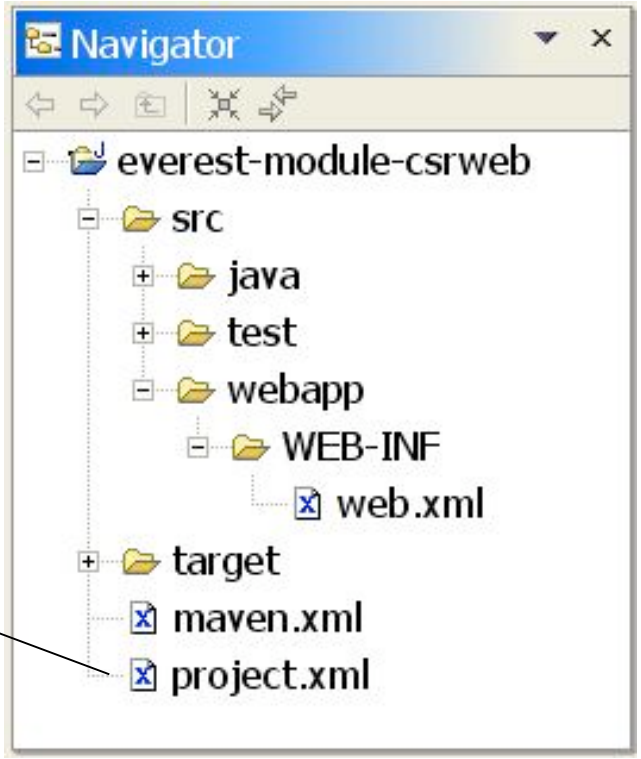


To execute: "`maven jar`"

CsrWeb – A simple WAR project

- `src/webapp`
  - Webapp content
    - ✓ web.xml
    - ✓ JSPs and other web resources

```
<dependency>
  <groupId>everest</groupId>
  <artifactId>everest-module-bedrock</artifactId>
  <version>1.0-SNAPSHOT</version>
  <properties>
    <war.bundle>true</war.bundle>
  </properties>
</dependency>
```

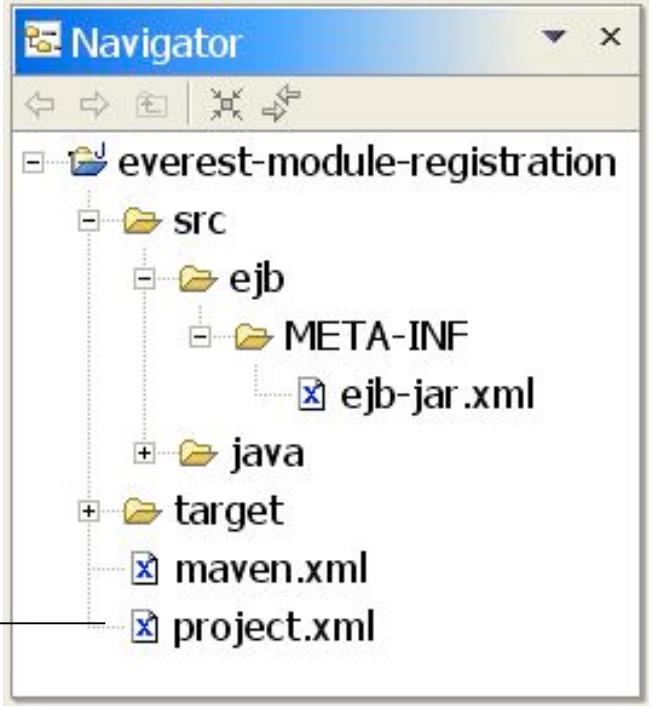


To execute: "`maven war`"

Registration – A simple EJB-JAR project

- `src/ejb`
  - EJB descriptors
- Some limitations
  - Does not run container-dependant ejb compiler for containers that needs stubs
    - ✓ Needs a preGoal

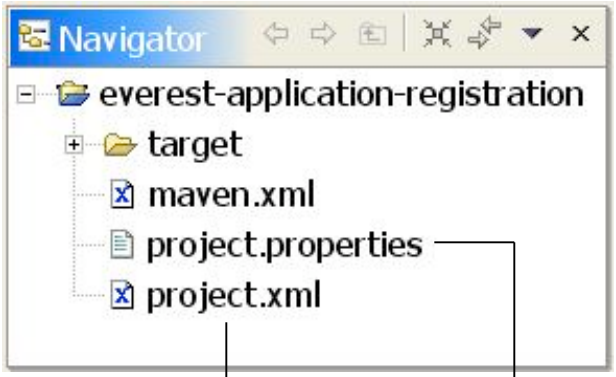
```
<dependency>
  <groupId>everest</groupId>
  <artifactId>everest-module-bedrock</artifactId>
  <version>1.0-SNAPSHOT</version>
  <properties>
    <ejb.manifest.classpath>true
                  </ejb.manifest.classpath>
  </properties>
</dependency>
```



To execute: "`maven ejb`"

Registration – A simple EAR project

- `src/application` (optional)
  - Location where to put
    - ✓ `META-INF/application.xml`
    - ✓ Any file that goes in `META-INF/`




```
<dependencies>
  <dependency>
    <groupId>everest</groupId>
    <artifactId>[...]csrweb</artifactId>
    <version>1.0-SNAPSHOT</version>
    <type>war</type>
    <properties>
      <ear.bundle>true</ear.bundle>
    </properties>
  </dependency>
[...]
```

```
maven.ear.appxml.generate=true
maven.ear.src=
  ${maven.build.dir}/application
```

To execute: "`maven ear`"

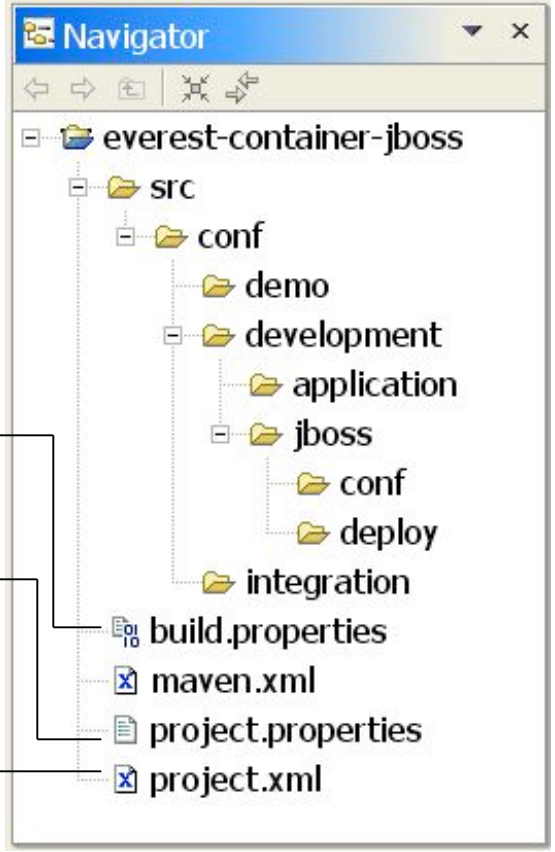
Simple container project with JBoss

- `src/conf/<environment name>`
  - Configuration files for a given environment
    - ✓ development, demo, integration, etc
  - Contains application configuration files
  - Contains JBoss configuration files
- The container project generates a zip
  - Contains a JBoss Server configuration

```
maven.jboss.home = C:/apps/jboss-3.2.1
env.name=development
```

```
maven.jboss.conf.dir = src/conf/${env.name}/jboss/conf
maven.jboss.deploy.dir = src/conf/${env.name}/jboss/deploy
maven.jboss.appconf.dir = src/conf/${env.name}/application
```

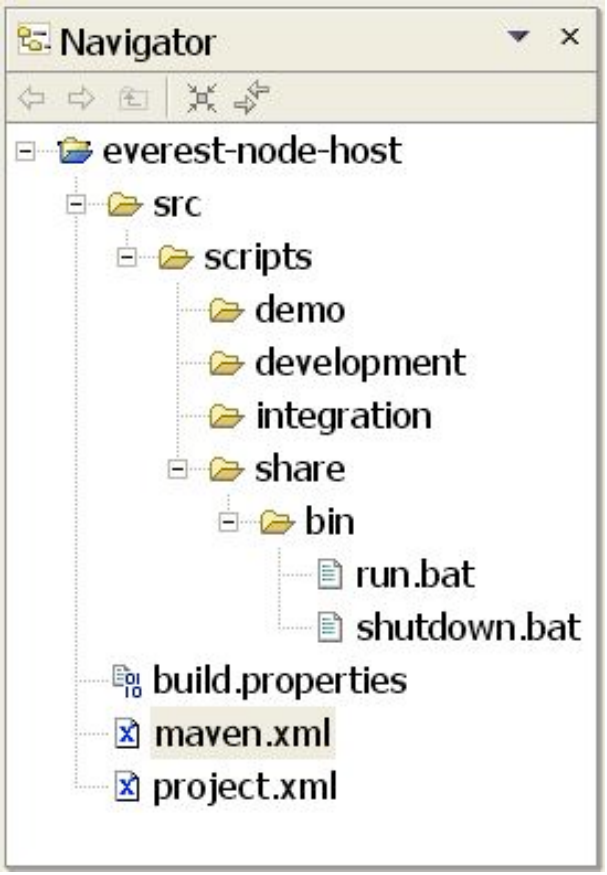
```
<dependency>[...]
  <artifactId>everest-application-registration</artifactId>
  <type>ear</type>
  <properties>
    <jboss.bundle>true</jboss.bundle>[...]
```



To execute: "`maven jboss:dist`"

A simple Host node project

- `src/scripts/`
  - Node script files
    - ✓ Start/stop containers
    - ✓ Other operation tasks
- The node project generates a zip
  - Contains a ready made set of containers, configured for the correct environment
- Custom made maven.xml script

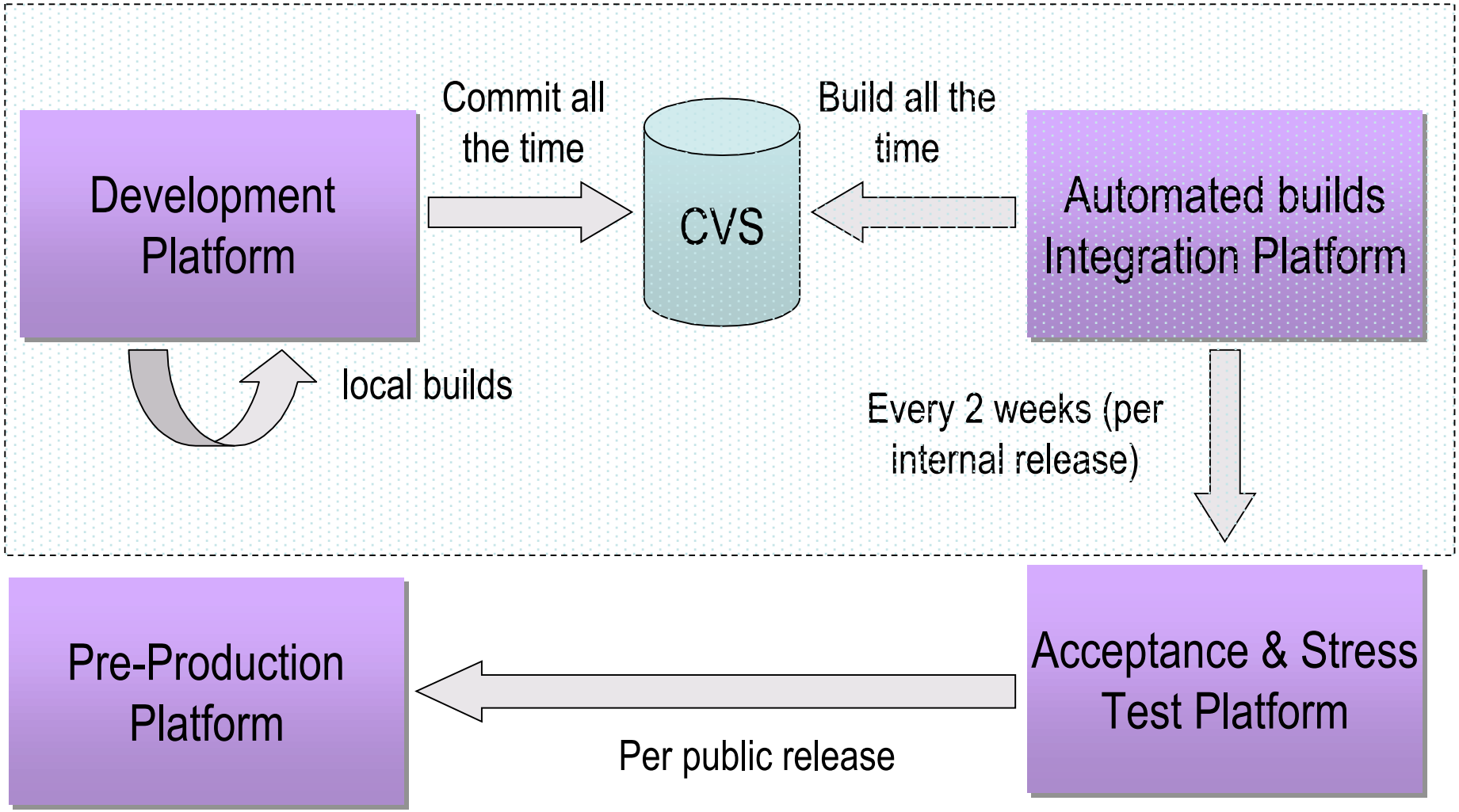


To execute: "`maven everest:dist`"

Agenda

- What is Maven and how it works?
- Implementing a J2EE example
- Continuous Integration with Maven
- Extending Maven
- Tips and future directions

Continuous Integration Development

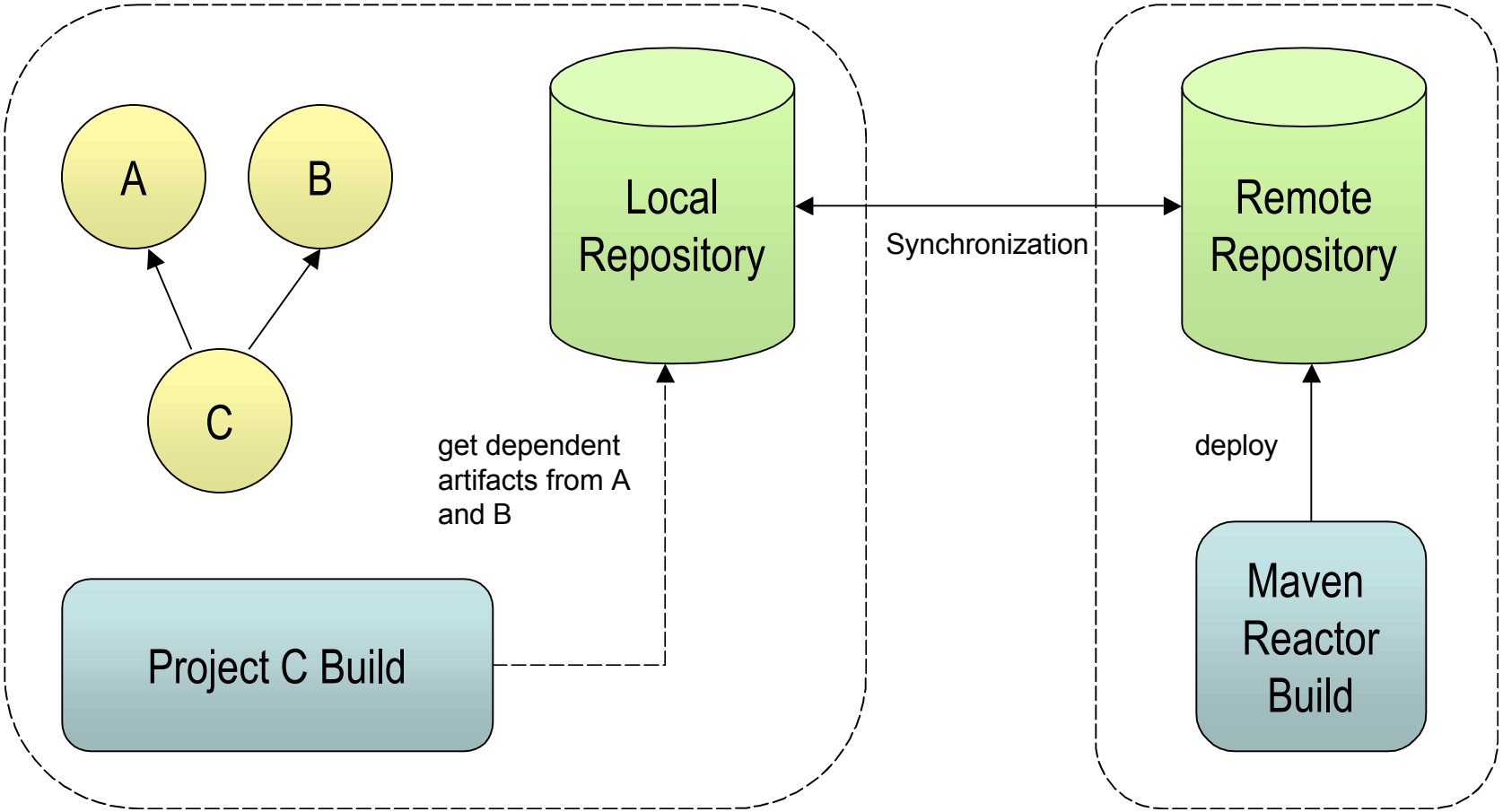


- Maven Reactor

Maven.xml

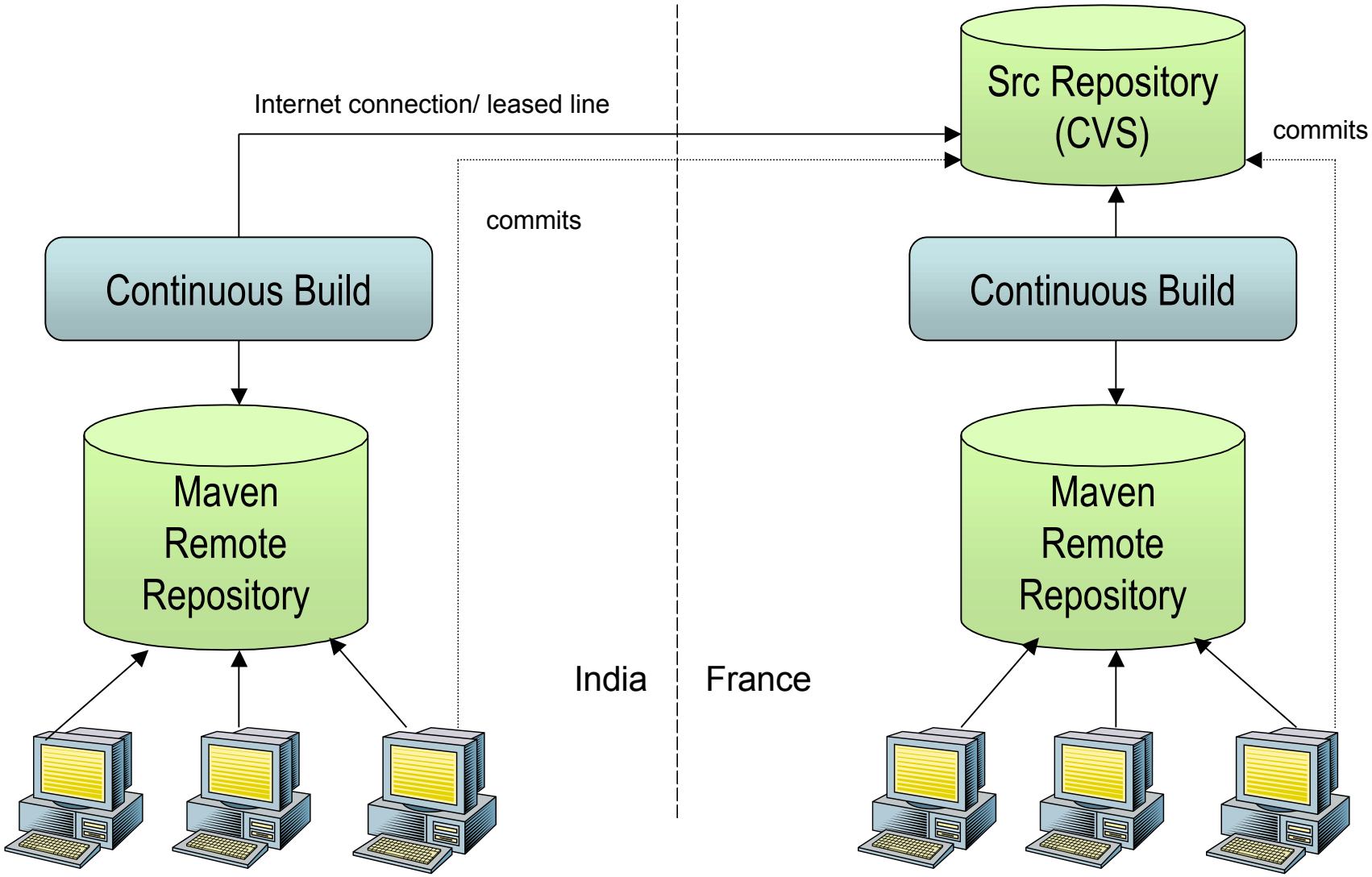
```
<goal name="everest:dist-all">
   <maven:reactor
       basedir="${basedir}"
       includes="**/${pattern}/**/project.xml"
       excludes="project.xml"
       goals="everest:dist"
       banner="Building"
       ignoreFailures="false"/>
 </goal>
```

Setting up Maven in a Corporate environment

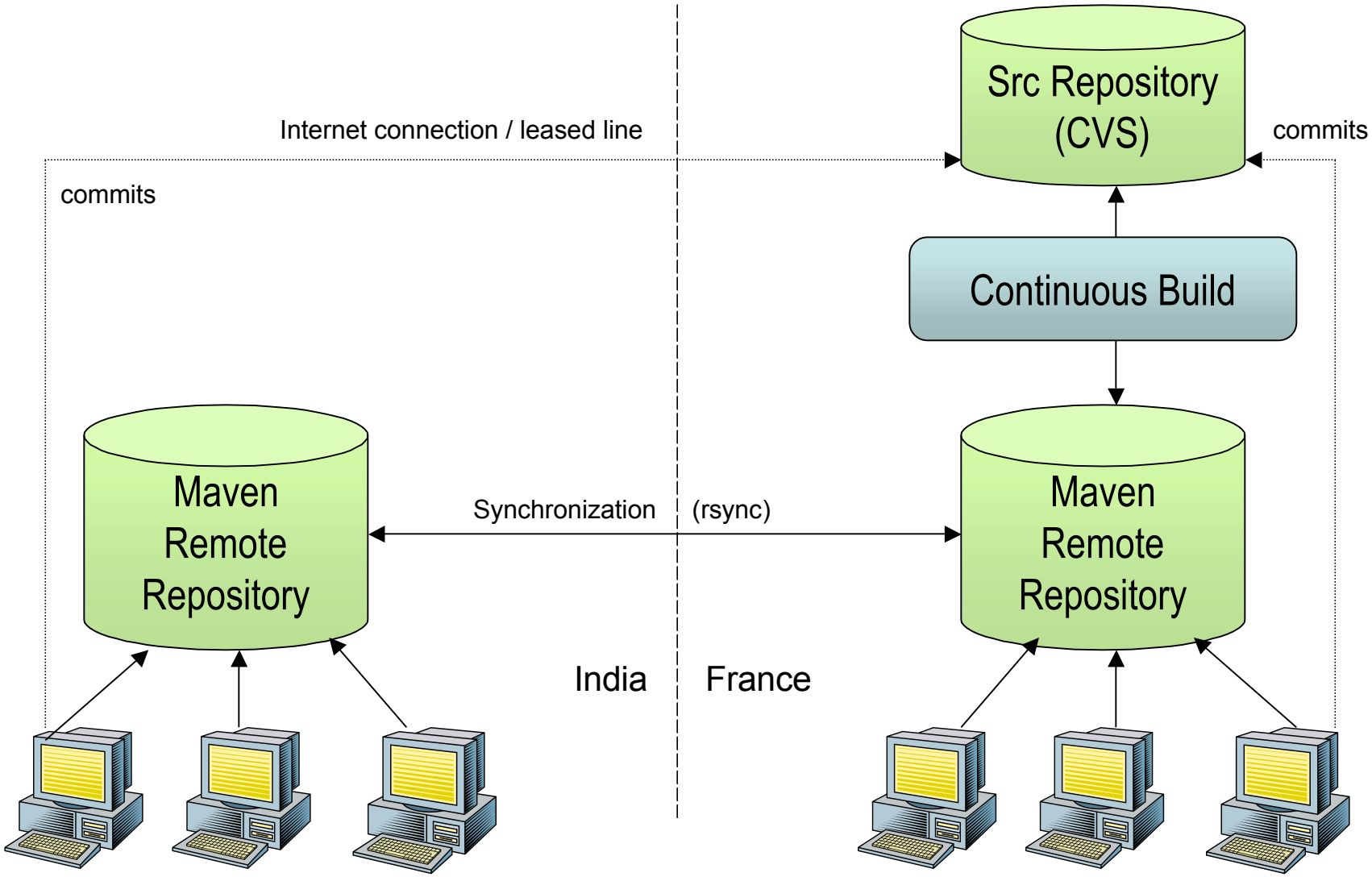


Note: Store SNAPSHOT artifacts in Remote repositories

Multi-location setup (1/2)



Multi-location setup (2/2)



Agenda

- What is Maven and how it works?
- Implementing a J2EE example
- Continuous Integration with Maven
- Extending Maven
- Tips and future directions

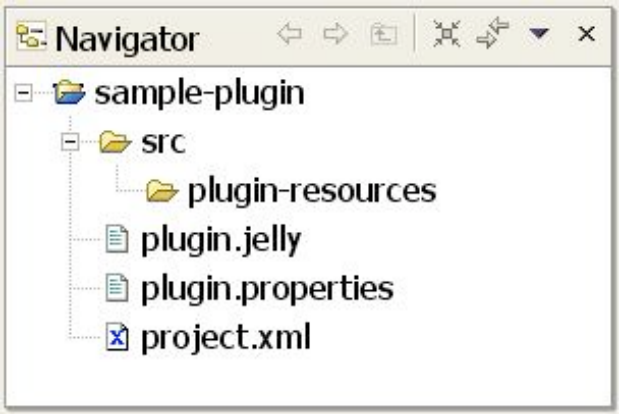
Extending Maven

- First level of customization
  - project.properties/build.properties
    - ✓ For plugin configuration
- Second level of customization
  - create a maven.xml file
    - ✓ Containing new goals
    - ✓ Intercepting existing goals with <preGoal/> and <postGoal/>
  - factorize maven.xml by using a common inherited project
- Third level of customization
  - create plugins to share common build logic

Writing a Maven plugin

plugin.jelly

```
<?xml version="1.0"?>

<project
  xmlns:j="jelly:core"
  xmlns:ant="jelly:ant">

  <goal name="hello"
    description="Send nice message">

    <ant:echo>Hello world!</ant:echo>

  </goal>

</project>
```



To build and deploy: "`maven plugin:install`"

Agenda

- What is Maven and how it works?
- Implementing a J2EE example
- Continuous Integration with Maven
- Extending Maven
- Tips and future directions

Maven tips

- Do not create too many subprojects
  - It is so easy to create a project with Maven…
  - We had 1500+ projects, brought back to 150+
  - A project = a public interface
    - ✓ The more you have the less stable you are
- Build every few hours
- Need strong commitment from Management
  - To tell that successful builds is all-important
- Use Snapshot jars
- Define external jars in top level inherited project
  - So that all projects use the same version
  - Better: use common project.properties
    - ✓ But not supported yet by Maven (soon)
- Share project specific build logic through
  - Top level maven.xml
  - Project-specific Plugins

Maven pros and cons

- Pros
  - Quick to set up
  - Best development practice enforcer
  - Benefit from numerous plugins
  - Nice features
    - ✓ Dependencies handling
    - ✓ Easy to set up a Continuous integration process
- Cons
  - Not mature yet
    - ✓ Little documentation (and not always up to date)
    - ✓ Lots of little details missing (but progressing quickly)
    - ✓ Needs someone to monitor/participate to the Mailing List

Future Maven

- Based on Avalon
  - Using the Plexus container
  - Everything is a component
    - ✓ Core Maven: Plugin Manager, etc
    - ✓ Plugins
- A plugin can be implemented
  - In Java code
  - Using Jelly
  - Using any scripting language
- The CLI is only one way of calling Maven core
- IDE-friendly

Q&A