SUMMIT JBoss WORLD

PRESENTED BY RED HAT

LEARN. NETWORK.
EXPERIENCE OPEN SOURCE.

www.theredhatsummit.com

MAVEN BEST PRACTICES

John Casey
Senior Software Engineer, Red Hat
4 May 2011

Agenda

- Better Maven builds
  - composition:
    - projects
    - dependencies
  - distribution:
    - assemblies
    - javadocs
  - release
- Maven and JBoss

Why “Maven Best Practices”?

- Trip hazards for new users
  - many are conceptual
- New potential for trouble
  - network problems
  - other people’s bad ideas
- (Just) enough rope to hang yourself
- More than just creating jars and zips
  - requires a little more thought to get right

Review: Maven Design Goals

- Address both user and developer concerns
  - **ZERO** project-specific knowledge to run a build
  - minimal project-specific knowledge required to design a build
  - standard build lifecycle
    - controlled scaffold of build “verbs”
- Reuse build logic
- Network of related projects and artifacts

Better Projects

- Common problems with projects:
  - unnecessarily unique layout
  - monolithic projects may present usability problems

Using Maven's Defaults

- Most configurations have defaults
  - use them
- Avoid pain
  - unnecessary maintenance burden
  - potential for untested configurations
- Let go and let Maven

Default Project Layout

```
• |-- pom.xml
`-- src
    |-- main
    |   |-- java
    |   |   `-- org
    |   |       `-- myproj
    |   |           `-- SomethingCool.java
    |   `-- resources
    |       `-- log4j.properties
    `-- test
        |-- java
        |   `-- org
        |       `-- myproj
        |           `-- SomethingCoolTest.java
        `-- resources
            |-- log4j.properties
            `-- templates
                |-- email.vm
                |-- print.vm
                `-- screen.vm
```

Default Project Layout

```
|-- pom.xml
`-- src
    |-- main
    |   |-- java
    |   |   `-- org
    |   |       `-- myproj
    |   |           `-- SomethingCool.java
    |   `-- resources
    |       `-- log4j.properties
    `-- test
        |-- java
        |   `-- org
        |       `-- myproj
        |           `-- SomethingCoolTest.java
        `-- resources
            |-- log4j.properties
            `-- templates
                |-- email.vm
                |-- print.vm
                `-- screen.vm
```

Default Project Layout

```
• |-- pom.xml
`-- src
    |-- main
    |   |-- java
    |   |   `-- org
    |   |       `-- myproj
    |   |           `-- SomethingCool.java
    |   `-- resources
    |       `-- log4j.properties
    `-- test
        |-- java
        |   `-- org
        |       `-- myproj
        |           `-- SomethingCoolTest.java
        `-- resources
            |-- log4j.properties
            `-- templates
                |-- email.vm
                |-- print.vm
                `-- screen.vm
```

Default Project Layout

```
• |-- pom.xml
`-- src
    |-- main
    |   |-- java
    |   |   `-- org
    |   |       `-- myproj
    |   |           `-- SomethingCool.java
    |   `-- resources
    |       `-- log4j.properties
    `-- test
        |-- java
        |   `-- org
        |       `-- myproj
        |           `-- SomethingCoolTest.java
        `-- resources
            |-- log4j.properties
            `-- templates
                |-- email.vm
                |-- print.vm
                `-- screen.vm
```

Default Project Layout

```
• |-- pom.xml
`-- src
    |-- main
    |   |-- java
    |   |   `-- org
    |   |       `-- myproj
    |   |           `-- SomethingCool.java
    |   `-- resources
    |       `-- log4j.properties
    `-- test
        |-- java
        |   `-- org
        |       `-- myproj
        |           `-- SomethingCoolTest.java
        `-- resources
            |-- log4j.properties
            `-- templates
                |-- email.vm
                |-- print.vm
                `-- screen.vm
```

1:1 POM-to-Artifact Relationship

- POM contains dependency statement for your jar
- Goes on the classpath?
  - probably needs its own POM
- Assemblies can be gray areas...

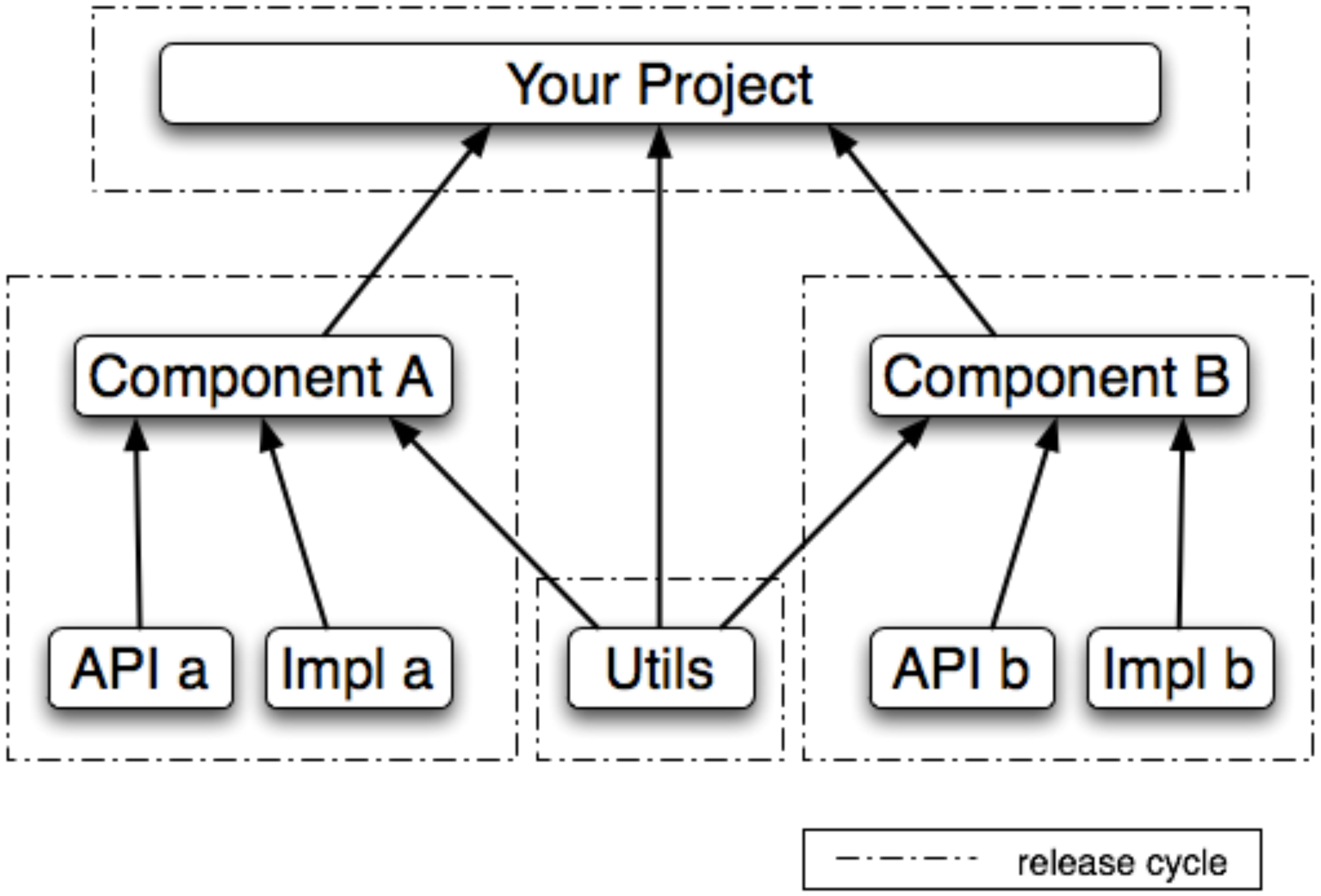
When to Modularize

- Generate multiple output jars from one build
- Separate API from implementation
- Publish functional subsets of project code

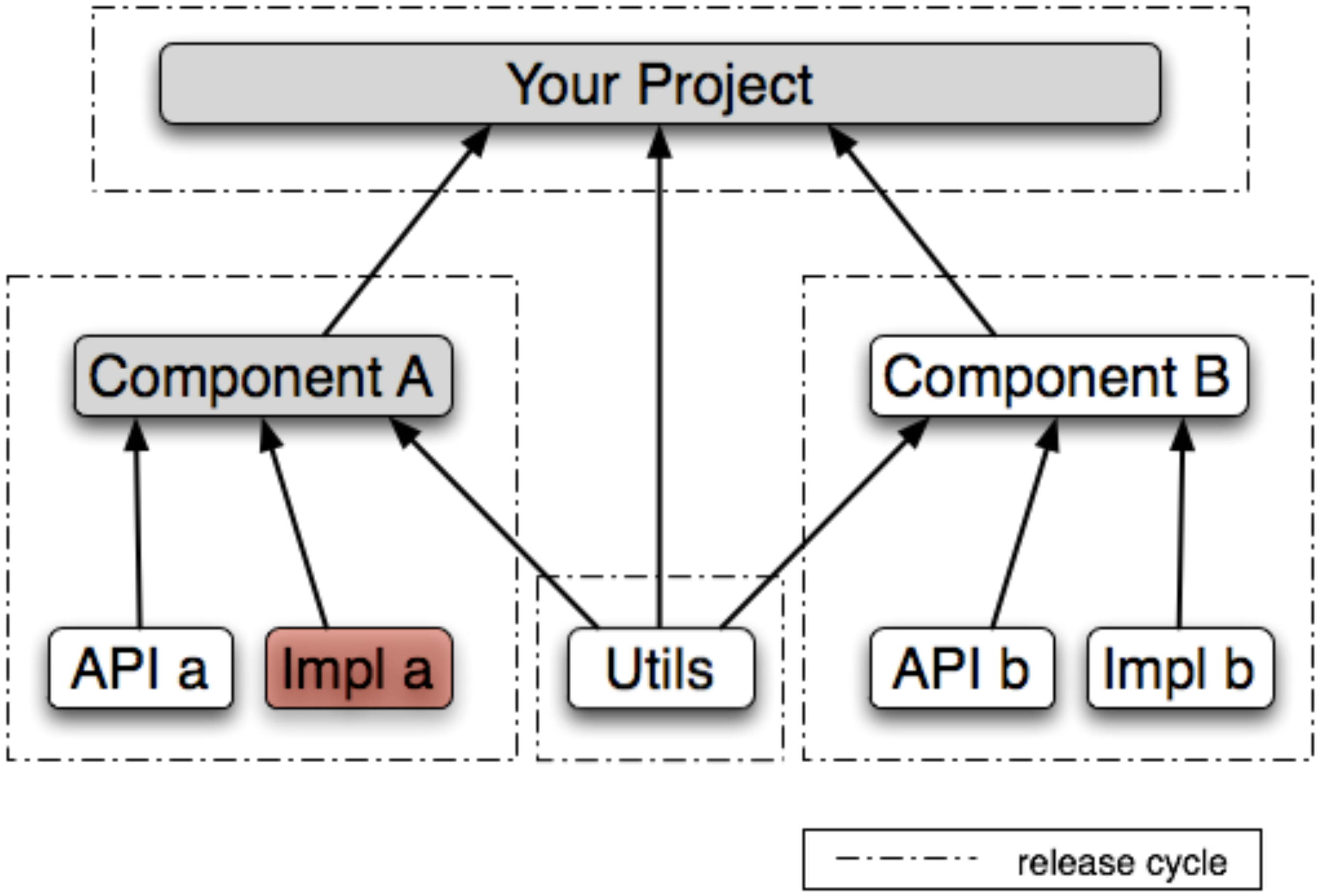
Better Dependencies

- Common problems with dependencies
  - misuse of -SNAPSHOT
  - misunderstanding of scope

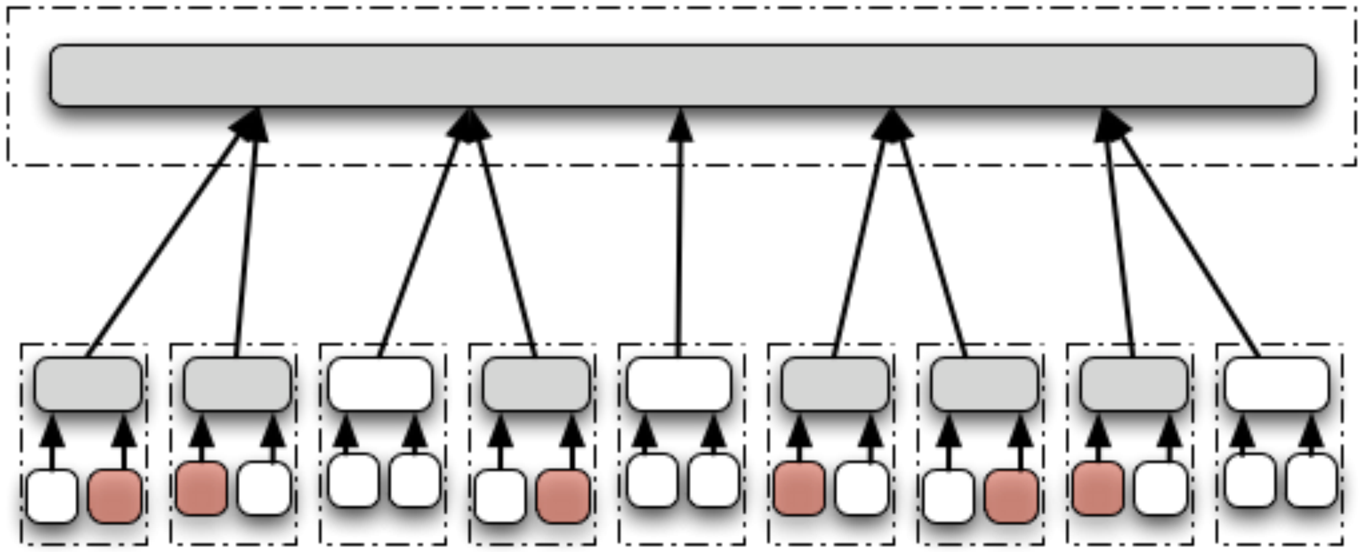
Building on Shifting Sand



Building on Shifting Sand



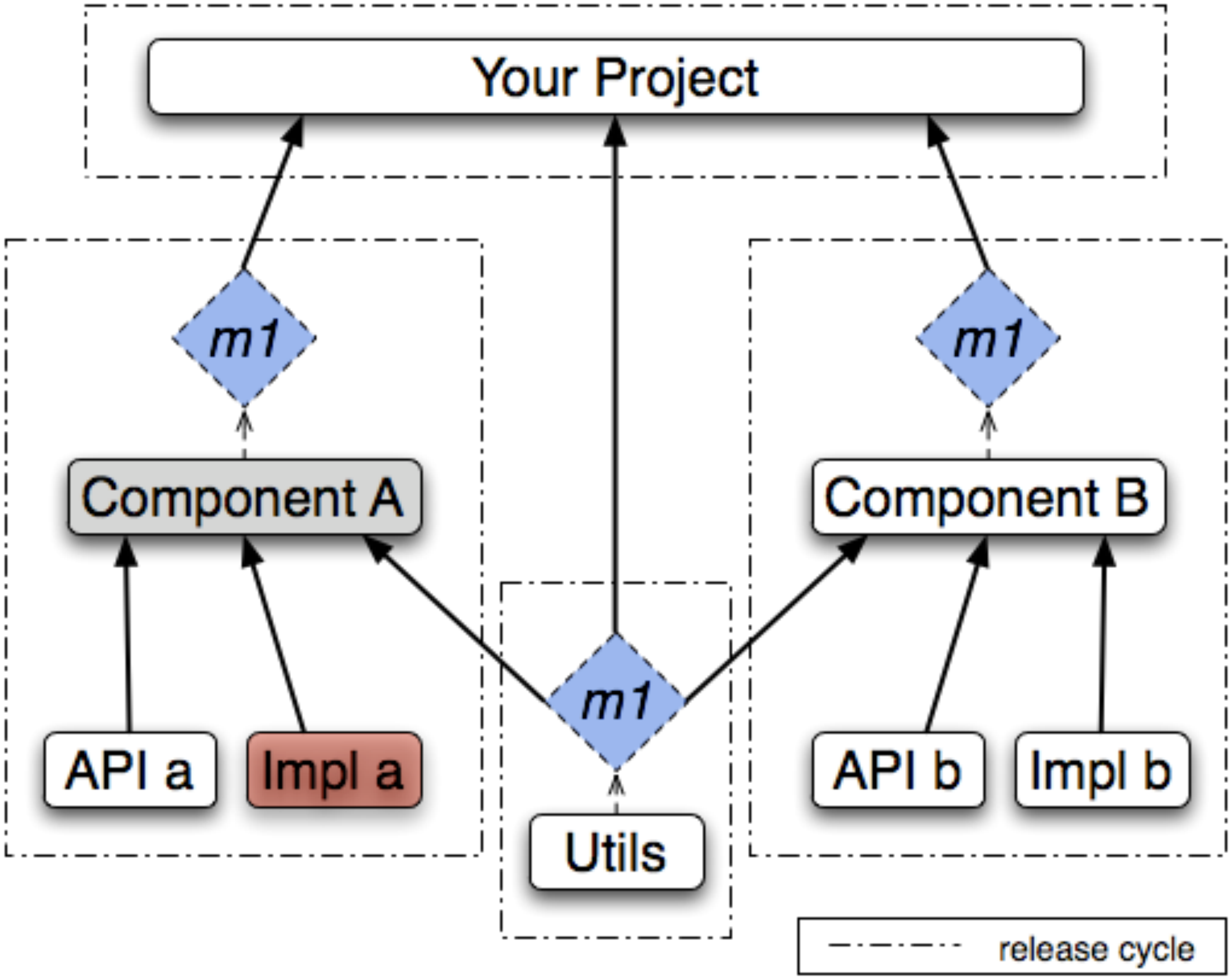
Building on Shifting Sand



Snapshot Dependencies

- What does a snapshot version imply?
- What are you committing your developers to?
- How much volatility are you prepared to handle?
- How do you limit volatility?

Building on (Less) Shifting Sand



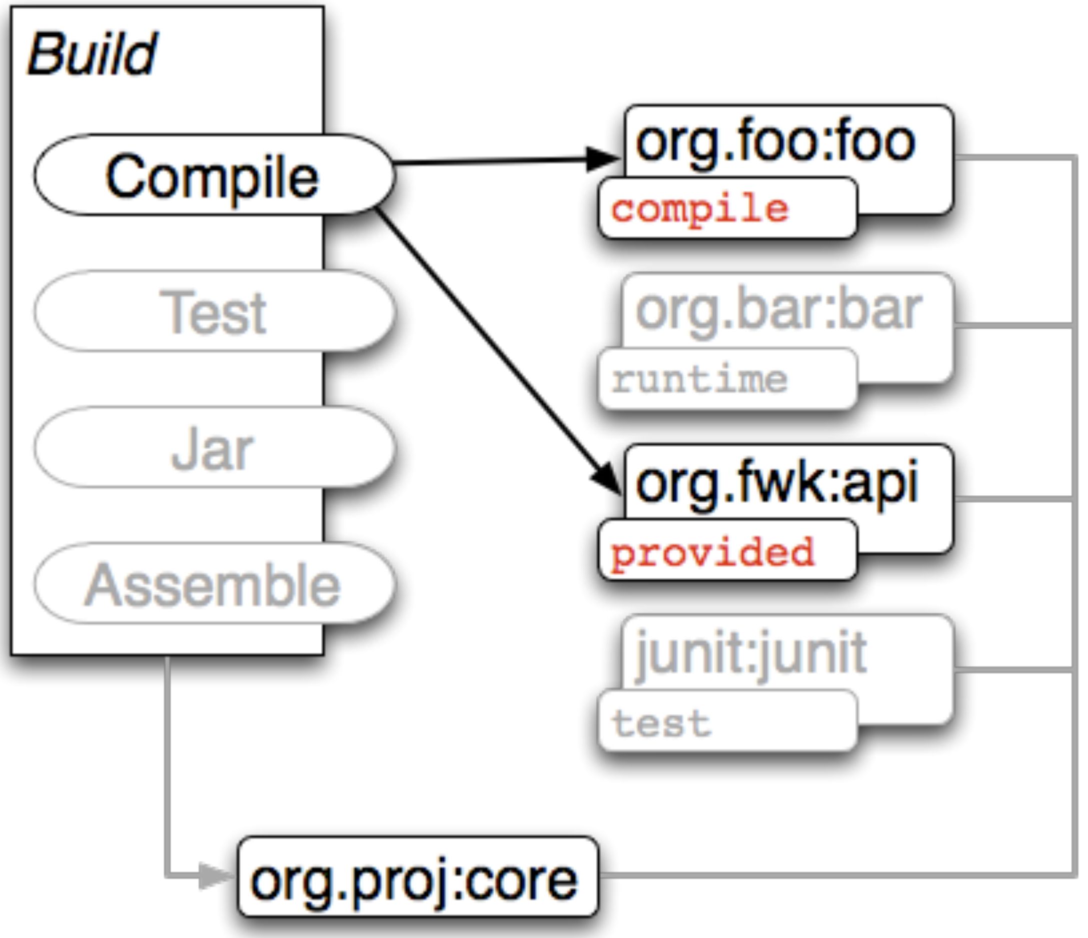
Dependency Scoping

- Two considerations:
  - How is it used in the build?
  - Do users need it?
- What does each scope mean?

```
compile
runtime
provided
test
```

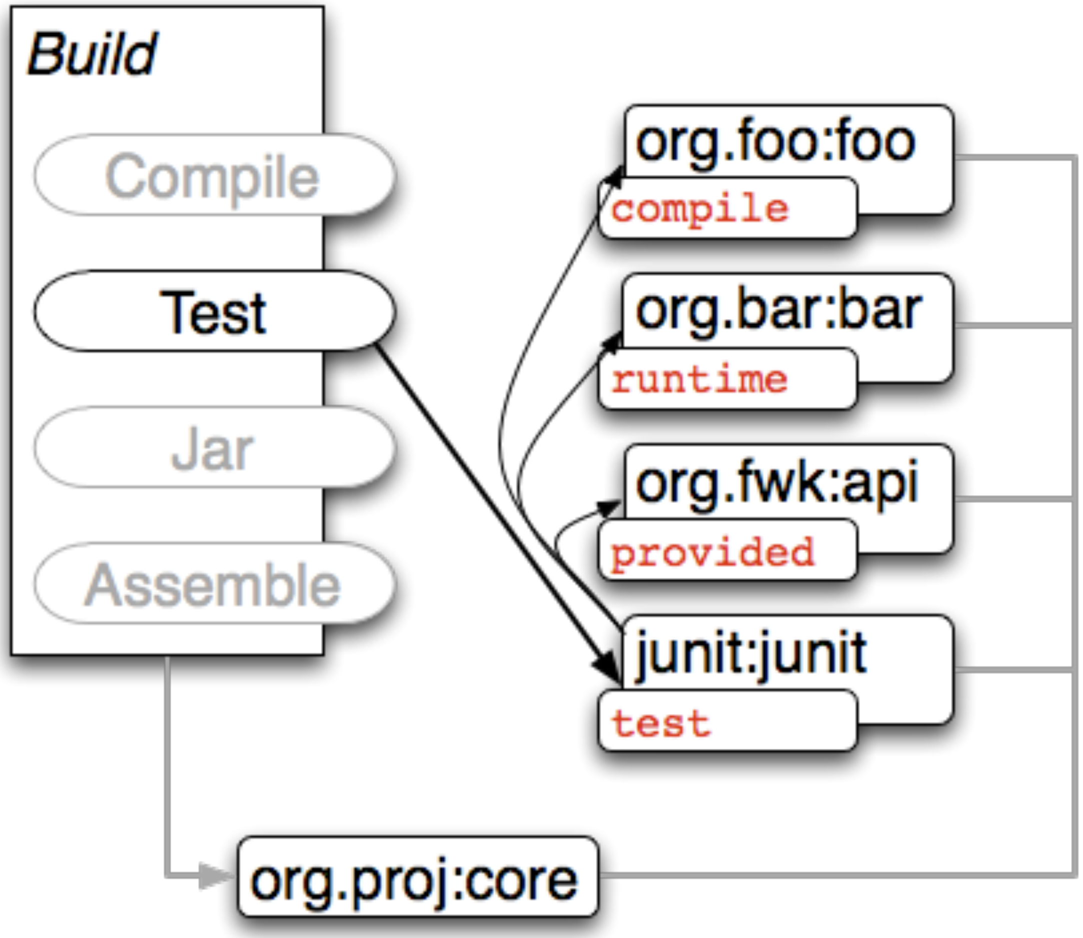
Dependency Scoping

Compiling:



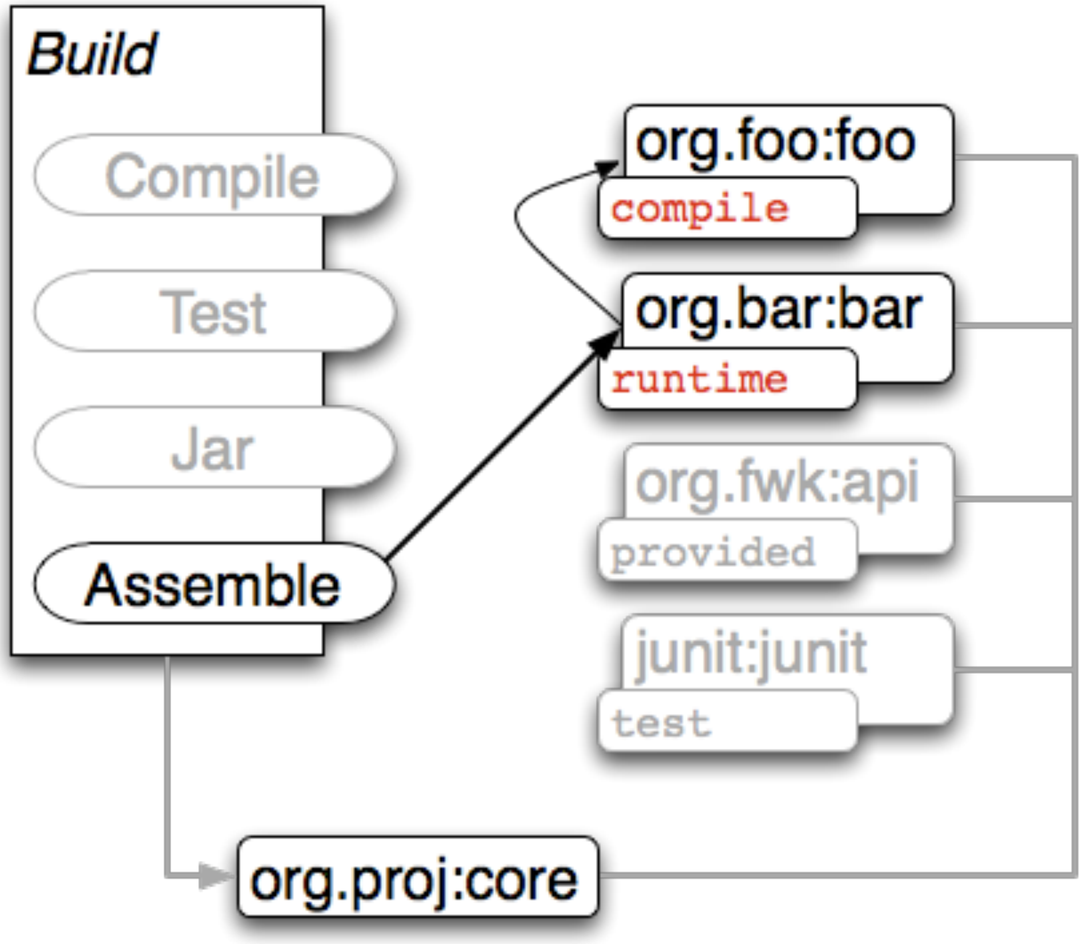
Dependency Scoping

Testing:



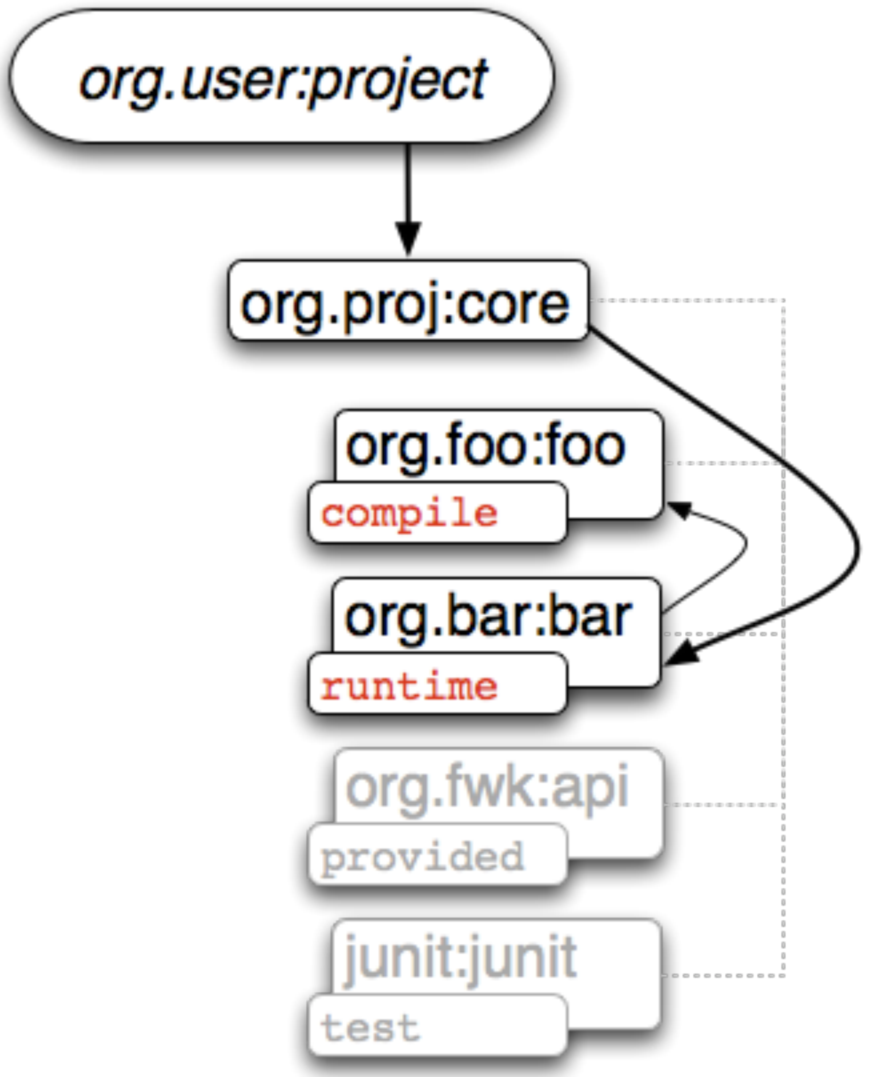
Dependency Scoping

"Running":



Dependency Scoping

What the user sees:



Questions So Far?

- Next Up:
  - Improving project distributions

Better Assemblies

- Common problems aggregating multi-module projects...
  - use a distribution module assembly at the top
  - use **ONLY** `assembly:single`

Better Assemblies

- Add dependencies on the other child modules
  - sorts the distro to the end of the build order

```
<dependencies>
  <dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>child1</artifactId>
    <version>${project.version}</version>
  </dependency>
  [...]
</dependencies>
```

Better Assemblies

- ...Then, use special `moduleSets` to aggregate:

```
<assembly>
  <id>distro</id>
  <formats>
    <format>zip</format>
  </formats>
  <moduleSets>
    <moduleSet>
      <useAllReactorProjects>true</useAllReactorProjects>
      <binaries>
        <outputDirectory>lib</outputDirectory>
      </binaries>
    </moduleSet>
  </moduleSets>
</assembly>
```

Better Assemblies

- ...Or better yet, use `dependencySets`:

```
<assembly>
  <id>distro</id>
  <formats>
    <format>zip</format>
  </formats>
  <dependencySets>
    <dependencySet>
      <outputDirectory>lib</outputDirectory>
    </dependencySet>
  </dependencySets>
</assembly>
```

Better JavaDocs

- Aggregating javadocs can be a mess...
  - recombining javadoc jars from modules is ugly
  - `javadoc:aggregate` goal considered harmful
  - **ANSWER:** use the distribution dependencies instead!

Better JavaDocs

- First, in non-distribution modules:
  - maven-source-plugin: `jar`
  - maven-javadoc-plugin: `resource-bundle`

Non-Distro Module Configuration

- Bundle the sources for dependency modules:

```
<plugin>
  <artifactId>maven-source-plugin</artifactId>
  <version>2.1.2</version>
  <executions>
    <execution>
      <id>sources</id>
      <goals>
        <goal>jar</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Non-Distro Module Configuration

Bundle javadoc resources for dependency modules:

(in `src/main/javadoc/` by default)

```
<plugin>
  <artifactId>maven-javadoc-plugin</artifactId>
  <version>2.7</version>
  <executions>
    <execution>
      <id>javadoc-bundle</id>
      <goals>
        <goal>resource-bundle</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Better JavaDocs

- Now, in the distribution module:
  - include dependencies for aggregation
  - enable `includeDependencySources`
  - use `javadoc:jar`

Distribution Module Configuration

- Now, aggregate those dependencies!

```
<plugin>
  <artifactId>maven-javadoc-plugin</artifactId>
  <version>2.7</version>
  <executions>
    <execution>
      <id>aggregated-javadocs</id>
      <goals>
        <goal>jar</goal>
      </goals>
      <configuration>
        <includeDependencySources>true</includeDependencySources>
[...]
```

Questions So Far?

- Next Up:
  - Better releases
  - JBoss dependencies in Maven builds

Better Releases

- Release workflow
  - `mvn clean release:prepare release:perform`
  - consider extra steps for better verification
- Requirements
- Common mistakes

Release Plugin Workflow

- `release:prepare`
  - "test" build
  - update version (`1.0`)
  - commit / tag
  - update version (`1.1-SNAPSHOT`)
  - commit

Release Plugin Workflow

- `release:perform`
  - checkout
  - “release” build

Better Releases

- Extra steps to consider:
  - staging
  - verifying
  - promoting
- Some repository managers support this “extended” release process

Release Plugin Requirements

- Single-pass build is critical!
- Using a release profile?
  - consider testing the profile before releasing.
- Required POM sections...

Better Releases

Enable project deployment:

```
<distributionManagement>
  <repository>
    <id>deployment</id>
    <name>Deployment Repository</name>
    <url>http://repo.myco.com/deployment/path</url>
  </repository>
</distributionManagement>
```

Better Releases

Enable SCM tagging:

```
<scm>
    <connection>
      scm:git:http://github.com/jdcasey/myproj.git
    </connection>

    <developerConnection>
      scm:git:git@github.com:jdcasey/myproj.git
    </developerConnection>

    <url>
      http://github.com/jdcasey/myproj
    </url>
</scm>
```

Better Releases

- Avoid exotic deployment transports
  - deploying to SVN is **NOT** a good idea
- Avoid strange configurations
  - Maven calls Ant, which calls Maven...
  - Maven calls Maven

Maven and JBoss

- JBoss Nexus instance

  http://repository.jboss.org/nexus/content/groups/public/

- JBoss community BOMs:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.as</groupId>
      <artifactId>jboss-as-parent</artifactId>
      <version>7.0.0.Beta2</version>
      <scope>import</scope>
      <type>pom</type>
    </dependency>
[...]
```

- Yes, but what about JBoss **products**?

Maven and EAP

- Maven repository ZIP archive for EAP 5.1.x
- One BOM to rule them all...

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.jboss.eap</groupId>
      <artifactId>eap-bom</artifactId>
      <version>5.1.0</version>
      <scope>import</scope>
      <type>pom</type>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Maven and EAP

- Depend on EAP client APIs via `jbossall-client`:

```
<dependencies>
  <dependency>
    <groupId>org.jboss.jbossas</groupId>
    <artifactId>jbossall-client</artifactId>
  </dependency>
</dependencies>
```

Maven and EAP

- Thar be monsters!
  - 5.1.x repository is “reconstructed”
  - POMs lack transitive dependency metadata
- EAP 6
  - native Maven builds
  - high-quality Maven repository will be a side effect

Questions?

LIKE US ON FACEBOOK
www.facebook.com/redhatinc

FOLLOW US ON TWITTER
www.twitter.com/redhatsummit

TWEET ABOUT IT
#redhat

READ THE BLOG
summitblog.redhat.com

GIVE US FEEDBACK
www.redhat.com/summit/survey