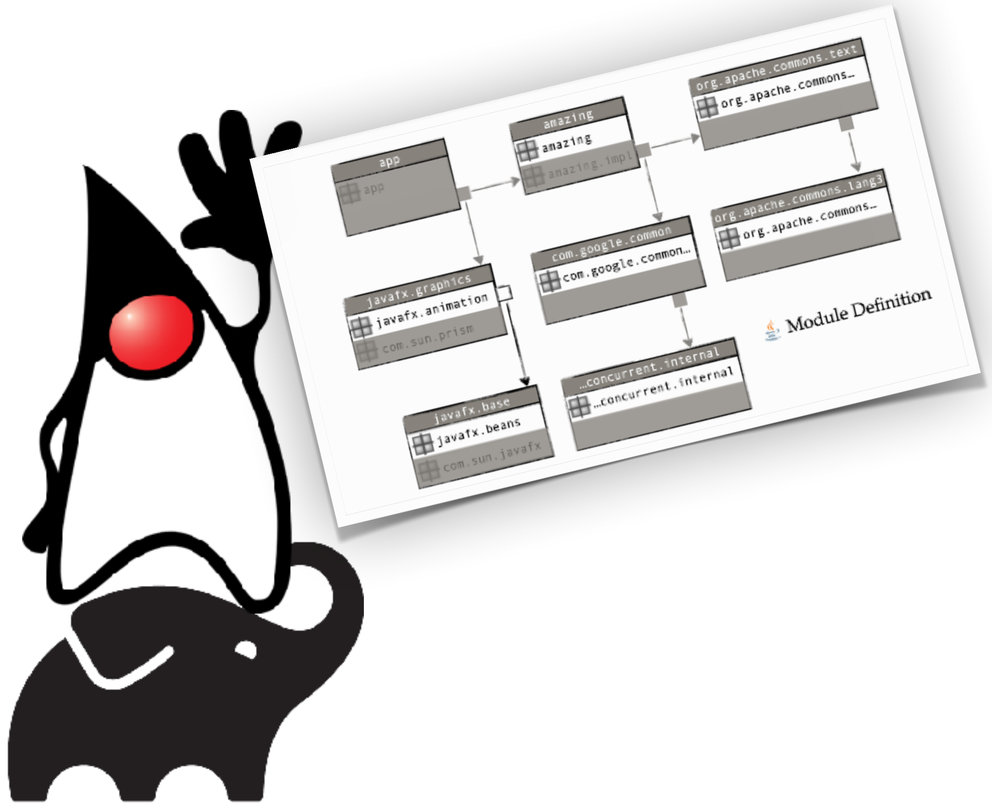
Combining Java Modules and Gradle for elegant project structures



Jendrik Johannes

github.com/jjohannes

JavaLand 2024

Problem

Accidental Complexity

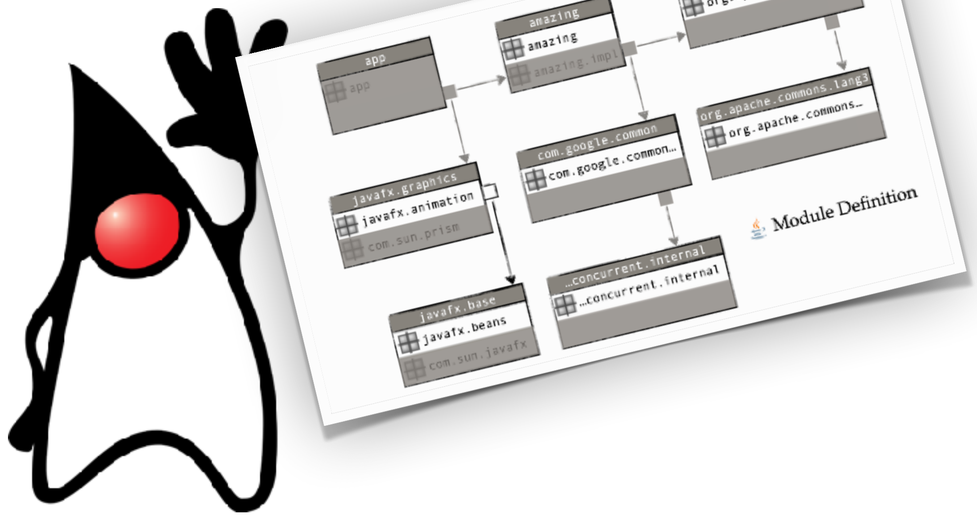
in

Build Configuration

and

Module Definition

(Architecture in Code)



In the Java language, a module is a set of packages designed for reuse.

In addition to .java files for classes and interfaces, each module has a source file called module-info.java which:

1. declares the **module's name**;
2. lists the **packages exported** by the module (to allow reuse by other modules);
3. lists other **modules required** by the module (to reuse their exported packages).

https://docs.oracle.com/en/java/javase/21/docs/specs/man/javac.html#:~:text=In%20the%20Java%20language%2C%20a,module

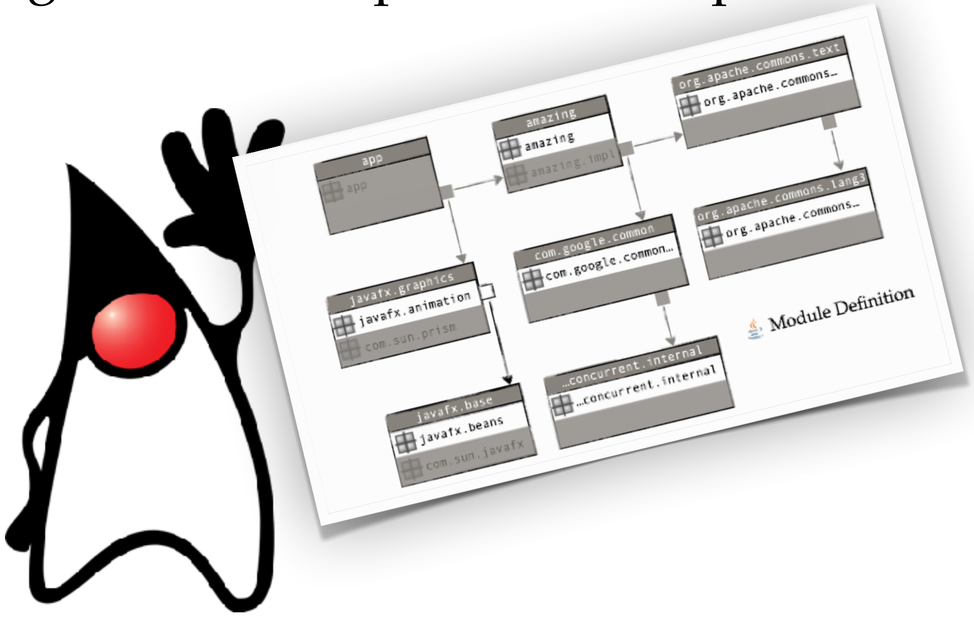
github.com/hibernate/hibernate-orm/blob/main/hibernate-core/hibernate-core.gradle

~300 Lines total
~20 Lines Module Definition

~300 Lines total
~100 Line Module Definition

```
29
30    dependencies {
31        api jakartaLibs.jpa
32        api jakartaLibs.jta
33
34        implementation libs.hcann
35        implementation libs.jandex
36        implementation libs.classmate
37        implementation libs.byteBuddy
38
39        implementation jakartaLibs.jaxbApi
40        implementation jakartaLibs.jaxb
41        implementation jakartaLibs.inject
42
43        implementation libs.antlrRuntime
44
```

github.com/apache/hadoop/blob/trunk/hadoop-common-project/hadoop-registry/pom.xml




```
29
30      <dependencies>
31
32        <dependency>
33          <groupId>org.slf4j</groupId>
34          <artifactId>slf4j-api</artifactId>
35        </dependency>
36
37        <dependency>
38          <groupId>org.apache.hadoop</groupId>
39          <artifactId>hadoop-auth</artifactId>
40        </dependency>
41
42        <dependency>
43          <groupId>org.apache.hadoop</groupId>
44          <artifactId>hadoop-annotations</artifactId>
45        </dependency>
46
```


Accidental Complexity

1. Mixing Module Definition and other concerns of the build

Essential Complexity

repo1.maven.org/maven2/org/apache/commons/commons-text/1.11.0/commons-text-1.11.0.pom

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <scope>compile</scope>
  <version>3.13.0</version>
</dependency>
```

should be:

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <scope>runtime</scope>
  <version>3.13.0</version>
</dependency>
```


Accidental Complexity

1. Mixing Module Definition and other concerns of the build
2. Different Module visibility at compile time and runtime (misunderstood / bad tool support)

Essential Complexity

- Different Module visibility at compile time and runtime (embrace this feature / good tool support)

repo1.maven.org/maven2/org/apache/commons/commons-text/1.11.0/commons-text-1.11.0.pom

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <version>3.13.0</version>
</dependency>
```

repo1.maven.org/maven2/org/apache/commons/commons-compress/1.26.1/commons-compress-1.26.1.pom

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <version>3.14.0</version>
</dependency>
```


Accidental Complexity

1. Mixing Module Definition and other concerns of the build
2. Different Module visibility at compile time and runtime (misunderstood / bad tool support)
3. Multiple Versions and Variants of a 3rd party Module are in conflict (misunderstood / bad tool support)

Essential Complexity

- Different Module visibility at compile time and runtime (embrace this feature / good tool support)
- Multiple Versions and Variants of a 3rd party Module are in conflict (accept / good tool support)

repo1.maven.org/maven2/org/openjfx/javafx-graphics/21.0.2/

```
org/openjfx/javafx-graphics/21.0.2/
├── javafx-graphics-21.0.2-linux.jar
├── javafx-graphics-21.0.2-mac.jar
└── javafx-graphics-21.0.2-win.jar
```

Accidental Complexity

1. Mixing Module Definition and other concerns of the build
2. Different Module visibility at compile time and runtime (misunderstood / bad tool support)
3. Multiple Versions and Variants of a 3rd party Module are in conflict (misunderstood / bad tool support)
4. Parts of Java software are OS/Arch specific – at minimum the VM itself (misunderstood / bad tool support)

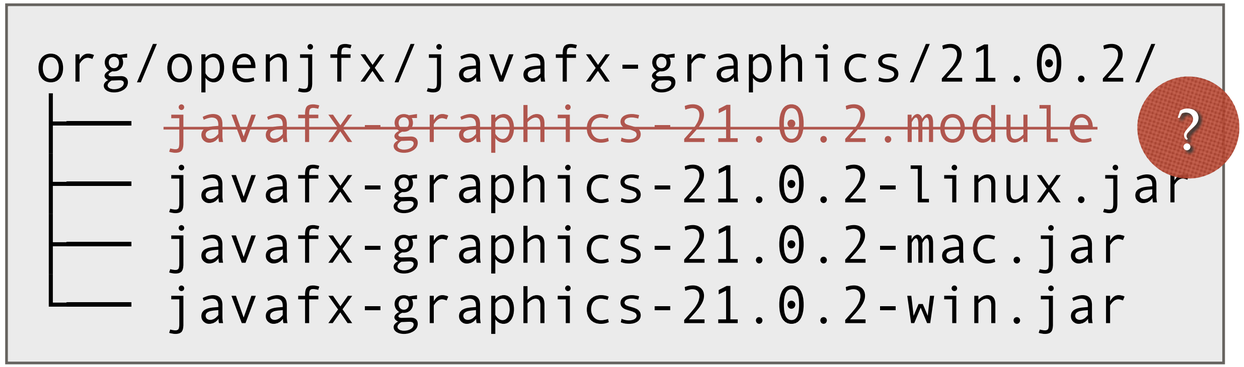
Essential Complexity

- Different Module visibility at compile time and runtime (embrace this feature / good tool support)
- Multiple Versions and Variants of a 3rd party Module are in conflict (accept / good tool support)
- Parts of Java software are OS/Arch specific (accept / good tool support)

repo1.maven.org/maven2/org/apache/commons/commons-text/1.11.0/commons-text-1.11.0.pom

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <scope>compile</scope>
  <version>3.13.0</version>
</dependency>
```

repo1.maven.org/maven2/org/openjfx/javafx-graphics/21.0.2/




Accidental Complexity

1. Mixing Module Definition and other concerns of the build
2. Different Module visibility at compile time and runtime (misunderstood / bad tool support)
3. Multiple Versions and Variants of a 3rd party Module are in conflict (misunderstood / bad tool support)
4. Parts of Java software are OS/Arch specific – at minimum the VM itself (misunderstood / bad tool support)
5. Incomplete or wrong 3rd party Module definitions (misunderstood by authors / bad tool support)

Essential Complexity

- Different Module visibility at compile time and runtime (embrace this feature / good tool support)
- Multiple Versions and Variants of a 3rd party Module are in conflict (accept / good tool support)
- Parts of Java software are OS/Arch specific (accept / good tool support)

Accidental Complexity

1. Mixing Module Definition and other concerns of the build
2. Different Module visibility at compile time and runtime (misunderstood / bad tool support)

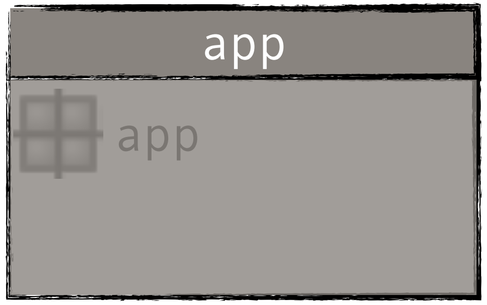
Essential Complexity

- Different Module visibility at compile time and runtime (embrace this feature / good tool support)

How to use the Java Module System to address this?

DEMO

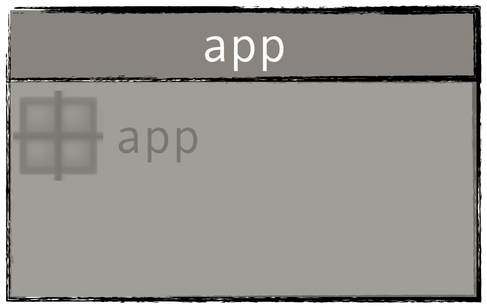
a pure Java Modules project



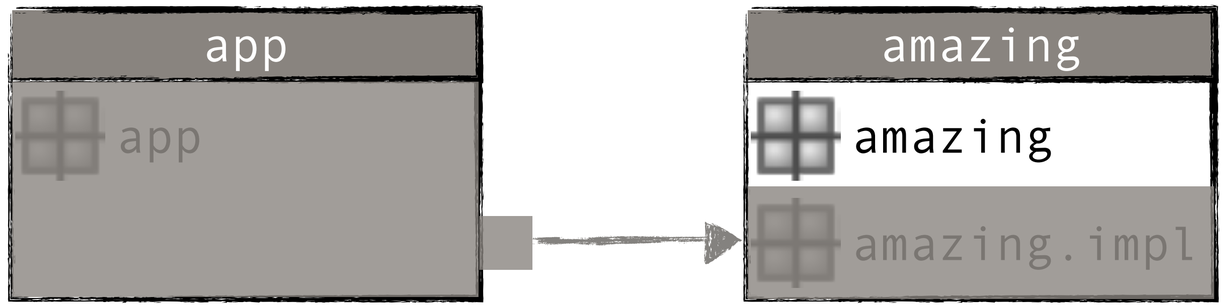

```
module app {}
```


```
module amazing {}
```

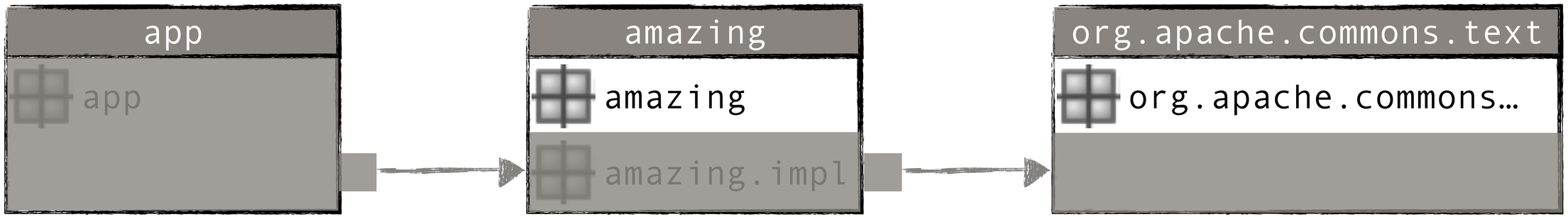

```
module amazing {
  exports amazing;
}
```




```
module app {
}
```

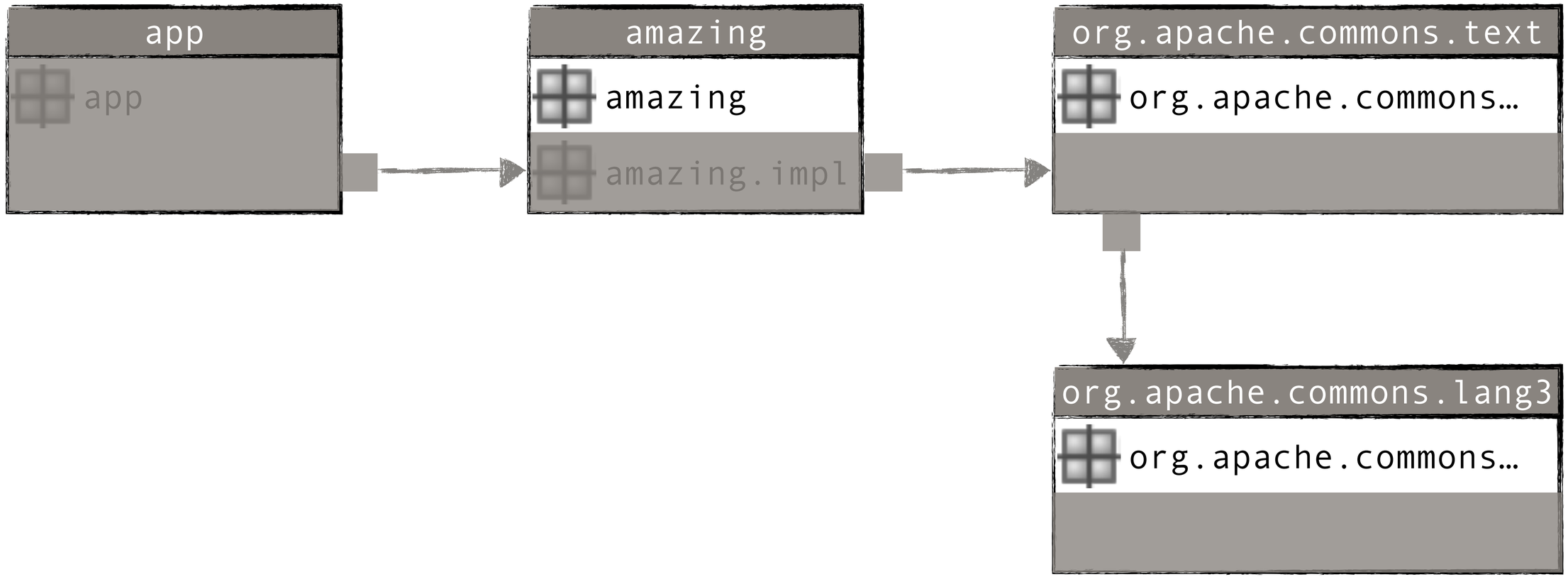



```
module app {
  requires amazing;
}
```

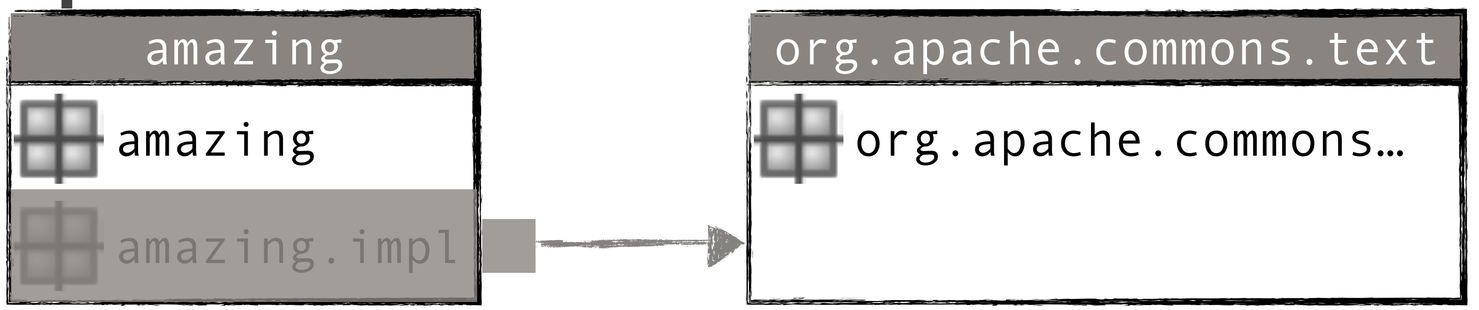



```
module amazing {
  requires org.apache.commons.text;

  exports amazing;
}
```


app
app
amazing
amazing
amazing.impl
org.apache.commons.text
org.apache.commons…
org.apache.commons.lang3
org.apache.commons…

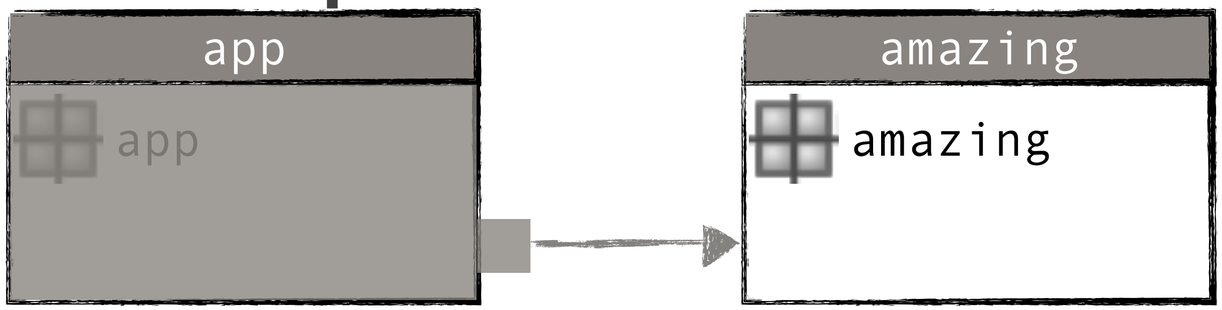
```
javac amazing/**/*.java module-info.java --module-path commons-text.jar
```




```
module amazing {
  requires org.apache.commons.text;

  exports amazing;
}
```

```
javac -d out app/App.java module-info.java --module-path ../amazing/out
```

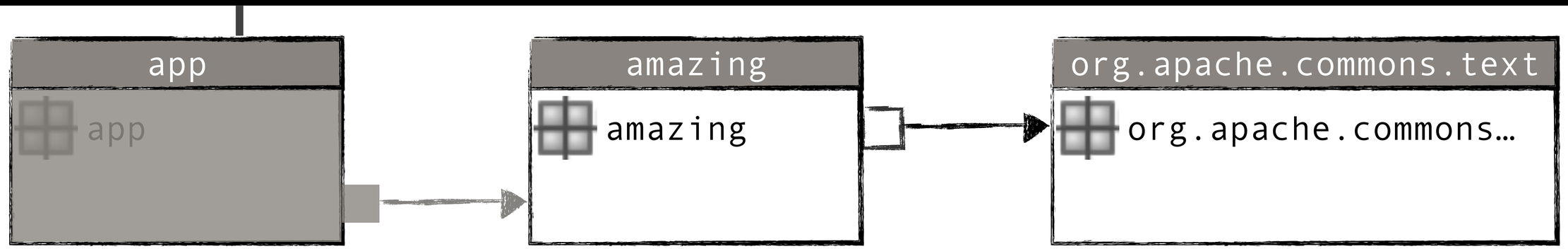



```
module app {
  requires amazing;
}
```

```
module amazing {
  requires org.apache.commons.text;

  exports amazing;
}
```

```
javac -d out app/App.java module-info.java --module-path ../amazing/out:commons-text.jar
```

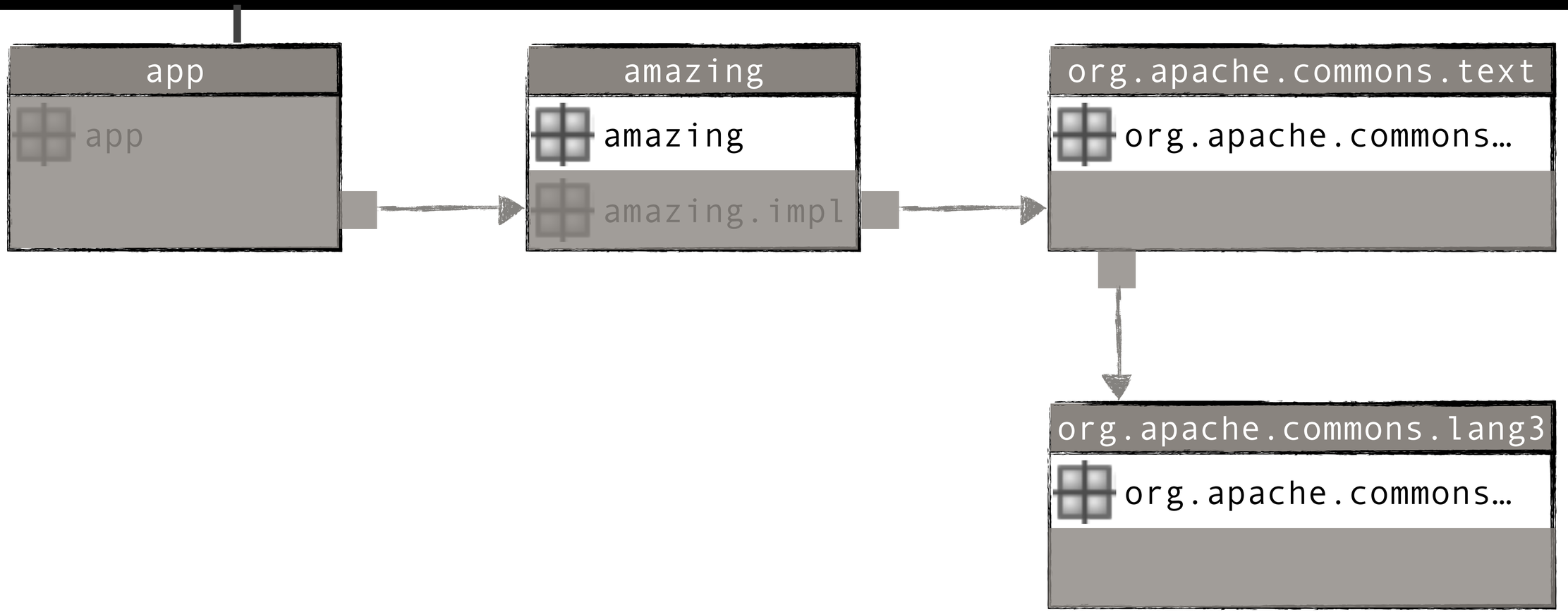



```
module app {
  requires amazing;
}
```

```
module amazing {
  requires transitive org.apache.commons.text;

  exports amazing;
}
```

```
java --module-path app/out:amazing/out:commons-text.jar:commons-lang3.jar --module app/app.App
```



Accidental Complexity

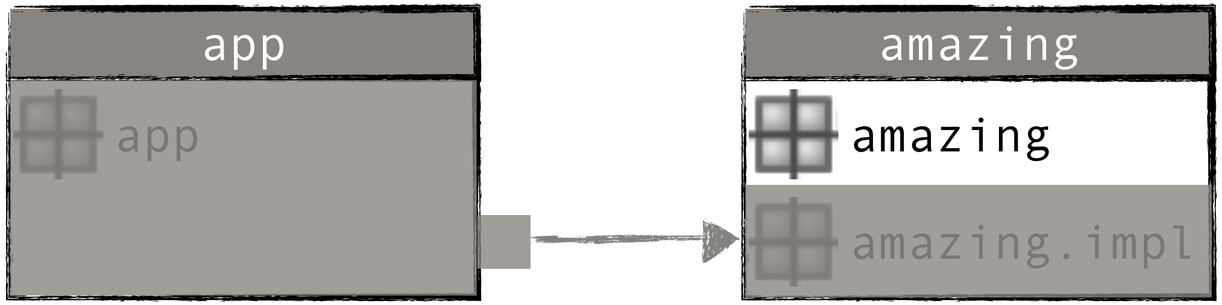
1. Mixing Module Definition and other c
2. Differe
compile time and runtime (misunderstood / bad tool support)

Module Definition

Essential Complexity

- Different Module visibility at compile time and runtime (embrace this feature / good tool support)

Use **module-info.java** for isolated Module Definitions in standard Java syntax




Accidental Complexity

1. ~~Mixing M~~
   ~~other cor~~
2. ~~Different~~
   ~~compile t~~
   ~~(misunderstood / bad tool support)~~
3. Multiple Versions and Variants of a 3rd party Module are in conflict (misunderstood / bad tool support)
4. Parts of Java software are OS/Arch specific – at minimum the VM itself (misunderstood / bad tool support)

Essential Complexity

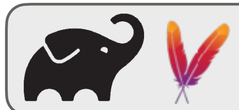
lity at
me (embrace
this feature / good tool support)

- Multiple Versions and Variants of a 3rd party Module are in conflict (accept / good tool support)
- Parts of Java software are OS/Arch specific (accept / good tool support)

How to use Gradle's Dependency Management to address this?

DEMO

add a Gradle-based build system



Accidental Complexity

1. Mixing Module Definition and other c
2. Differe compile time and runtime (misunderstood / bad tool support)
3. Multiple Versions and Variants of a 3rd party Module are in conflict (misu
4. Parts c specific – at minimum the VM itself (misunderstood / bad tool support)

Module Definition

Module Discovery

Essential Complexity

- Different Module visibility at compile time and runtime (embrace this feature / good tool support)
- Multiple Versions and Variants of a 3rd party Module are in conflict (accept / good tool support)
- Parts of Java software are OS/Arch specific (accept / good tool support)

Accidental Complexity

1. Mixing Module Definition and other c
2. Differe compile time and runtime (misunderstood / bad tool support)
3. Multiple Versions and Variants of a 3rd party Module are in conflict (misu
4. Parts c specific – at minimum the VM itself (misunderstood / bad tool support)
5. Incomplete or wrong 3rd party Module definitions (misunderstood by authors / bad tool support)

Module Definition

Module Discovery

Essential Complexity

- Different Module visibility at compile time and runtime (embrace this feature / good tool support)
- Multiple Versions and Variants of a 3rd party Module are in conflict (accept / good tool support)
- Parts of Java software are OS/Arch specific (accept / good tool support)

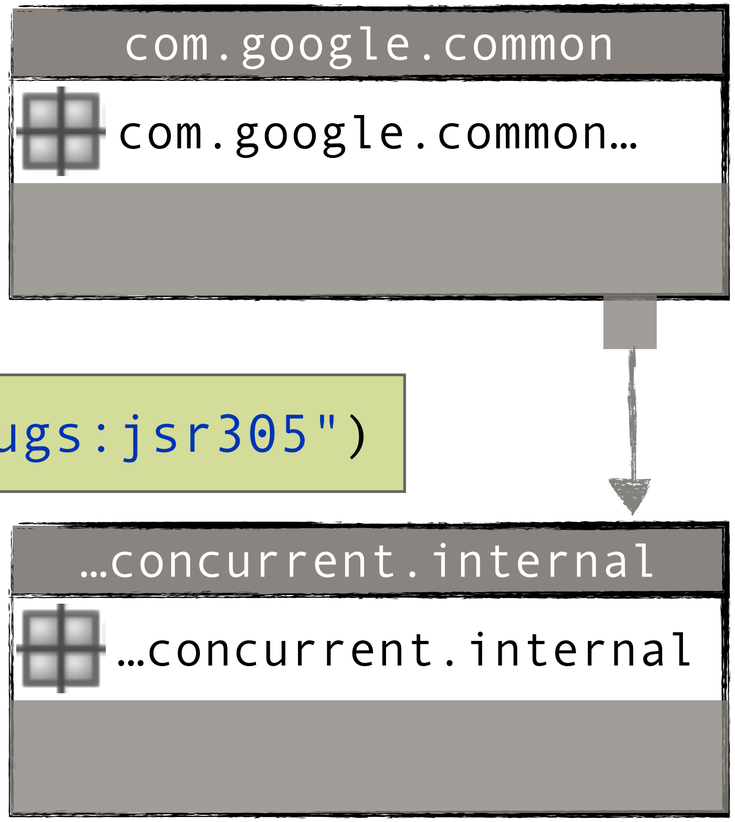
DEMO

add (and patch) more 3rd party Modules

```
com/google/guava/guava/33.1.0-jre/
├── guava-33.1.0-jre.module
└── guava-33.1.0-jre.jar
    └── module-info.class
```


```
"dependencies": [{
  "group": "com.google.guava",
  "module": "failureaccess",
  "version": { "requires": “1.0.2" },
  …
}]
```

```
com/google/guava/guava/33.1.0-jre/
├── guava-33.1.0-jre.module
└── guava-33.1.0-jre.jar
    └── module-info.class
```




```
PATCH: removeDependency("com.google.code.findbugs:jsr305")
```

```
"dependencies": [{
  "group": "com.google.code.findbugs",
  "module": "jsr305",
  "version": { "requires": "3.0.2" },
  …
}]
```

```
PATCH: module("com.google.guava:guava", "com.google.common")
```

```
module com.google.common {
  requires com.google.common.util.concurrent.internal;

  exports com.google.common.annotations;
  ...
}
```

Accidental Complexity

1. Mixing Module Definition and other c
2. Differe compile time and runtime (misunderstood / bad tool support)
3. Multiple Versions and Variants of a 3rd party Module are in conflict (misu
4. Parts c specific – at minimum the VM itself (misunderstood / bad tool support)
5. Incom Modu by author / bad tool support)

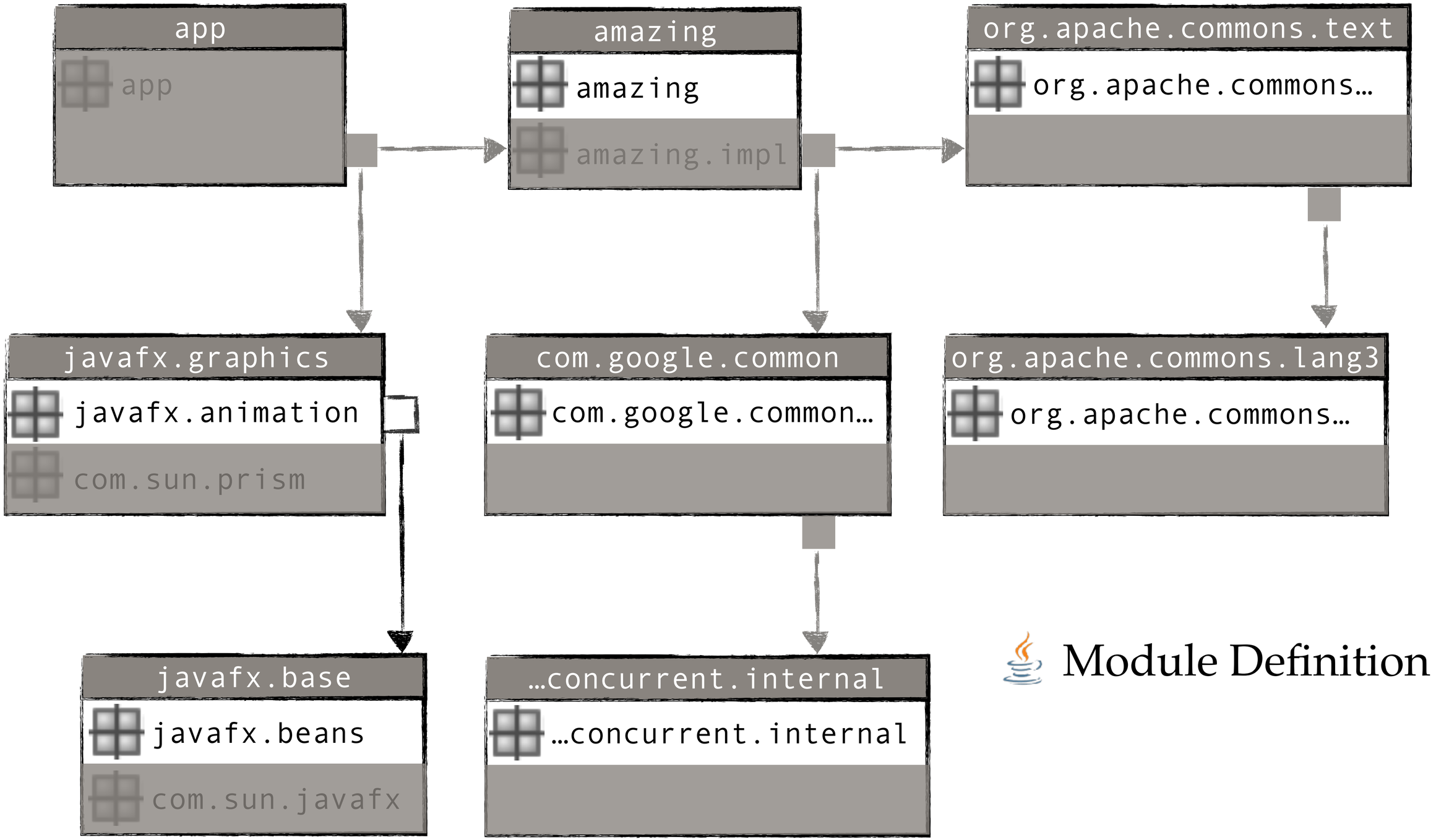
Module Definition

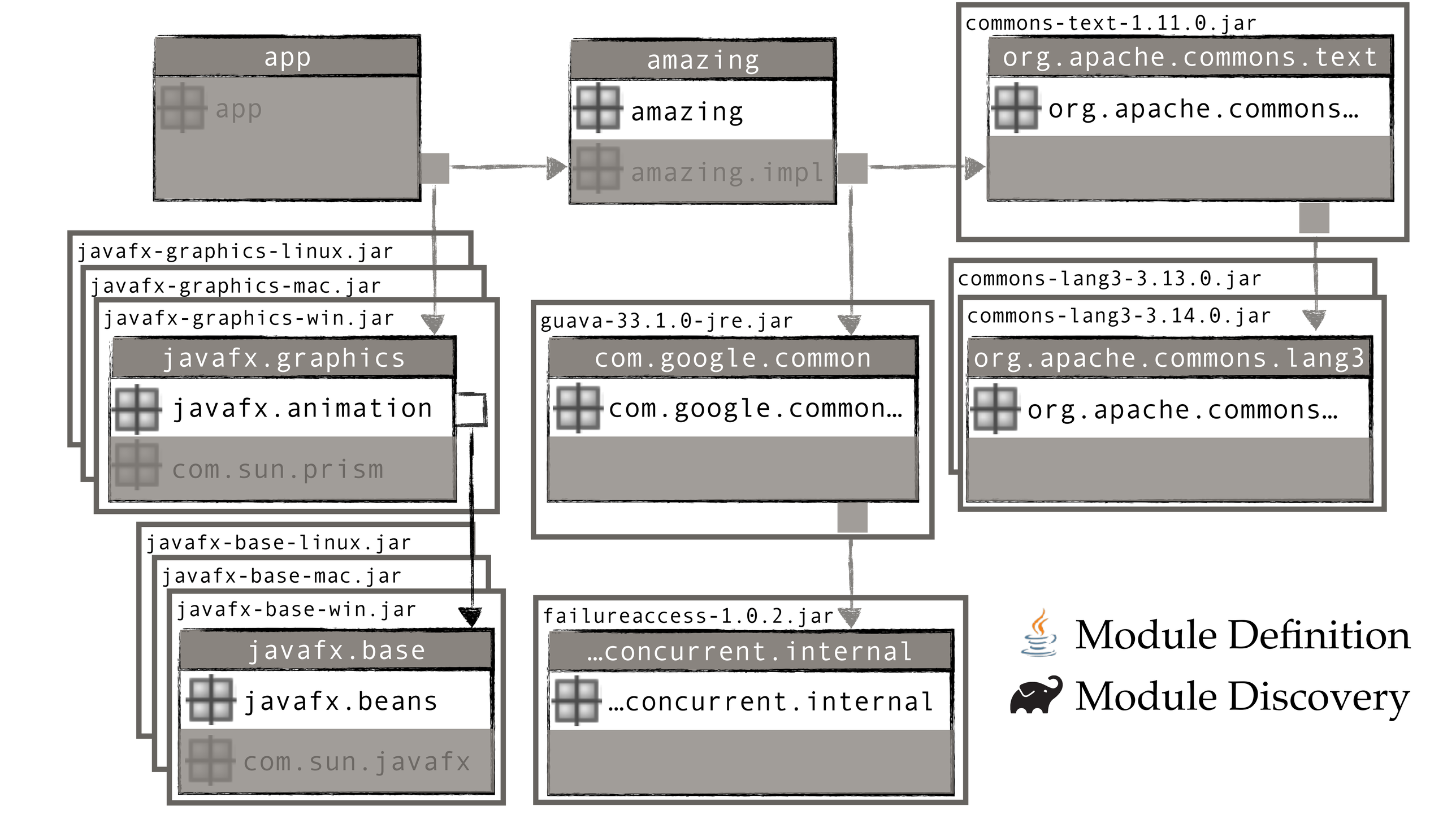
Module Discovery

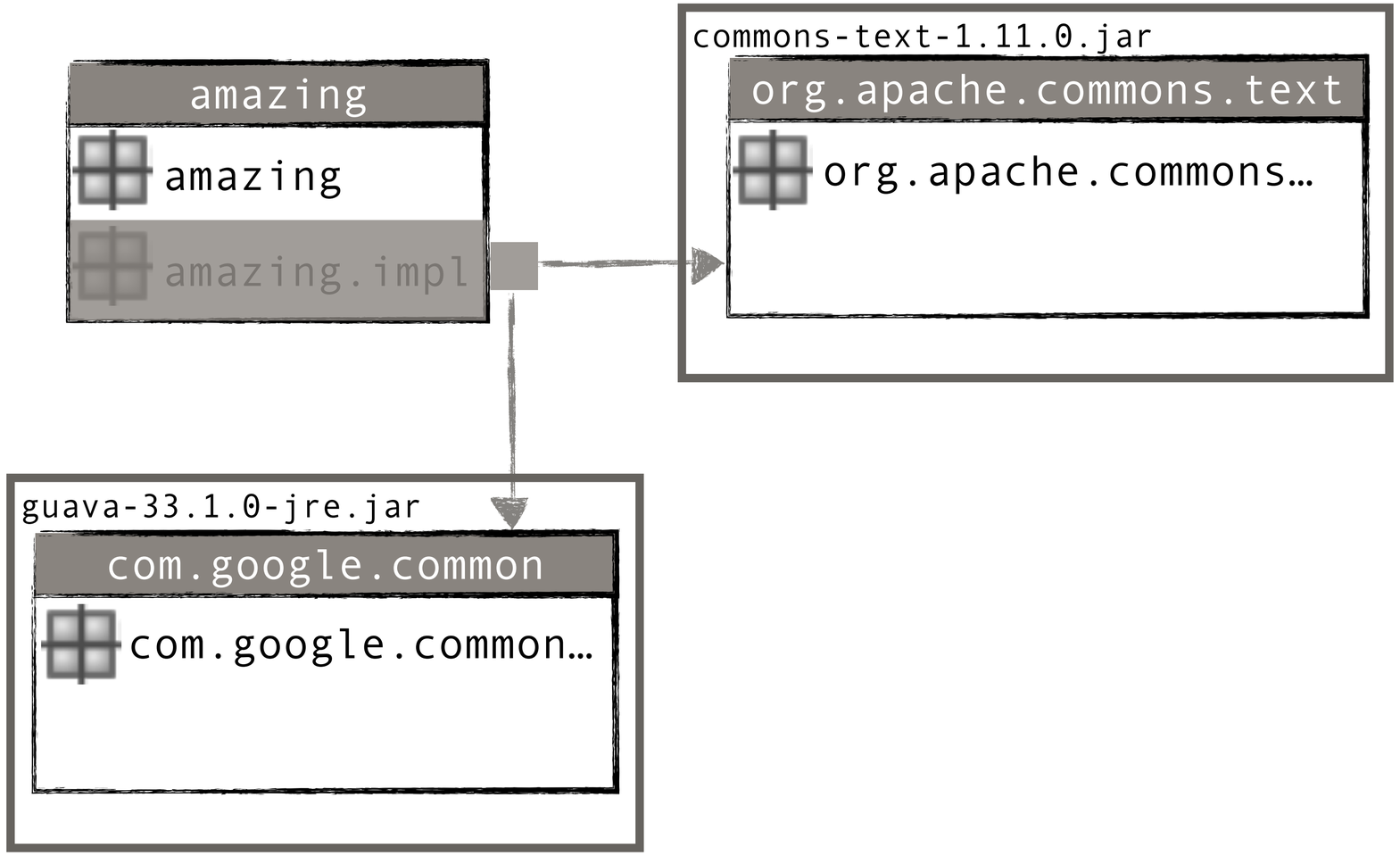
Module Patching

Essential Complexity

- Different Module visibility at compile time and runtime (embrace this feature / good tool support)
- Multiple Versions and Variants of a 3rd party Module are in conflict (accept reality / good tool support)
- Parts of Java software are OS/Arch specific (accept reality / good tool support)


app
app
amazing
amazing
amazing.impl
org.apache.commons.text
org.apache.commons…
javafx.graphics
javafx.animation
com.sun.prism
com.google.common
com.google.common…
org.apache.commons.lang3
org.apache.commons…
javafx.base
javafx.beans
com.sun.javafx
…concurrent.internal
…concurrent.internal
Module Definition

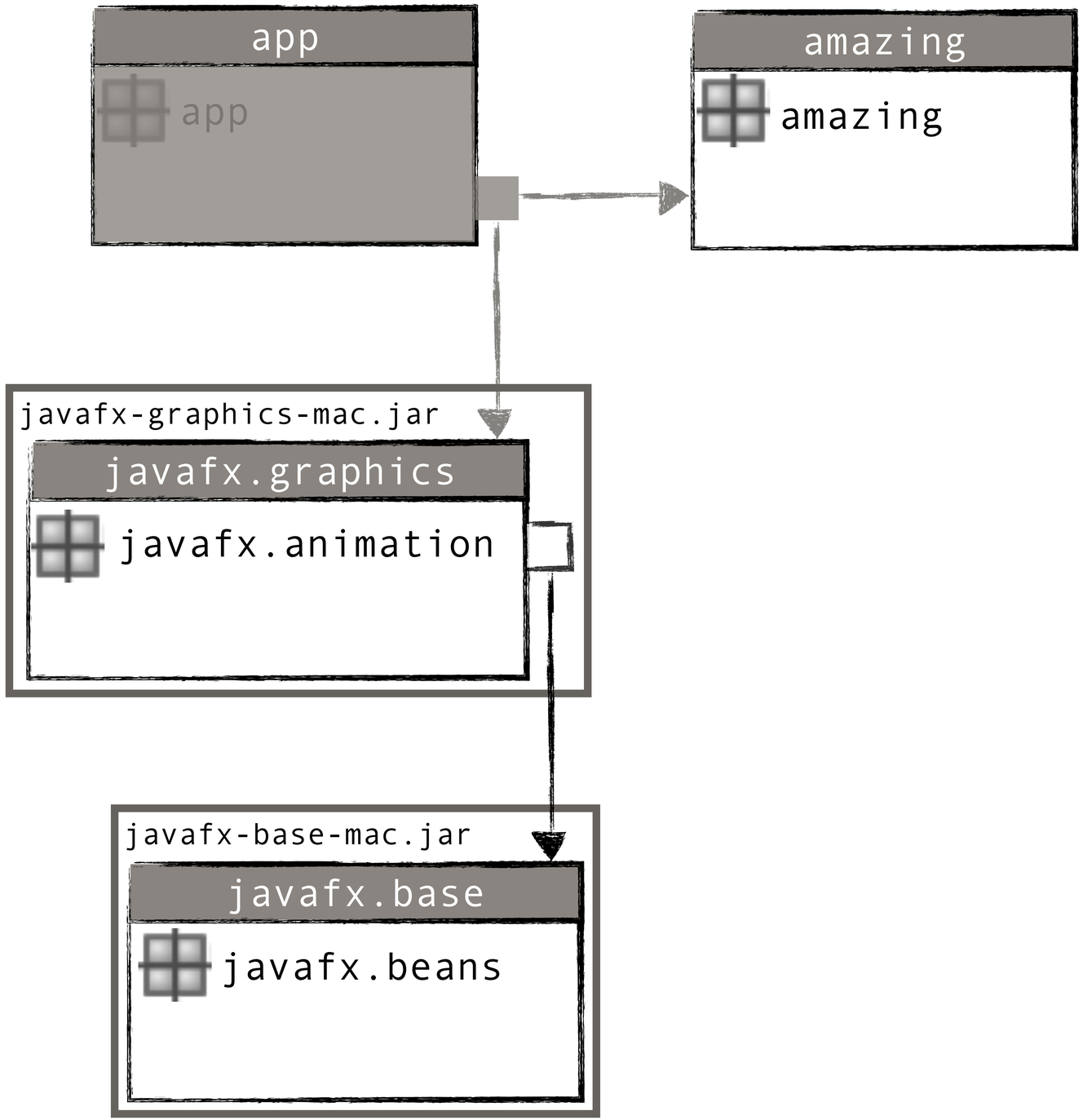

app
app
amazing
amazing
amazing.impl
commons-text-1.11.0.jar
org.apache.commons.text
org.apache.commons…
javafx-graphics-linux.jar
javafx-graphics-mac.jar
javafx-graphics-win.jar
javafx.graphics
javafx.animation
com.sun.prism
guava-33.1.0-jre.jar
com.google.common
com.google.common…
commons-lang3-3.13.0.jar
commons-lang3-3.14.0.jar
org.apache.commons.lang3
org.apache.commons…
javafx-base-linux.jar
javafx-base-mac.jar
javafx-base-win.jar
javafx.base
javafx.beans
com.sun.javafx
failureaccess-1.0.2.jar
…concurrent.internal
…concurrent.internal
Module Definition
Module Discovery



Module Definition

Module Discovery

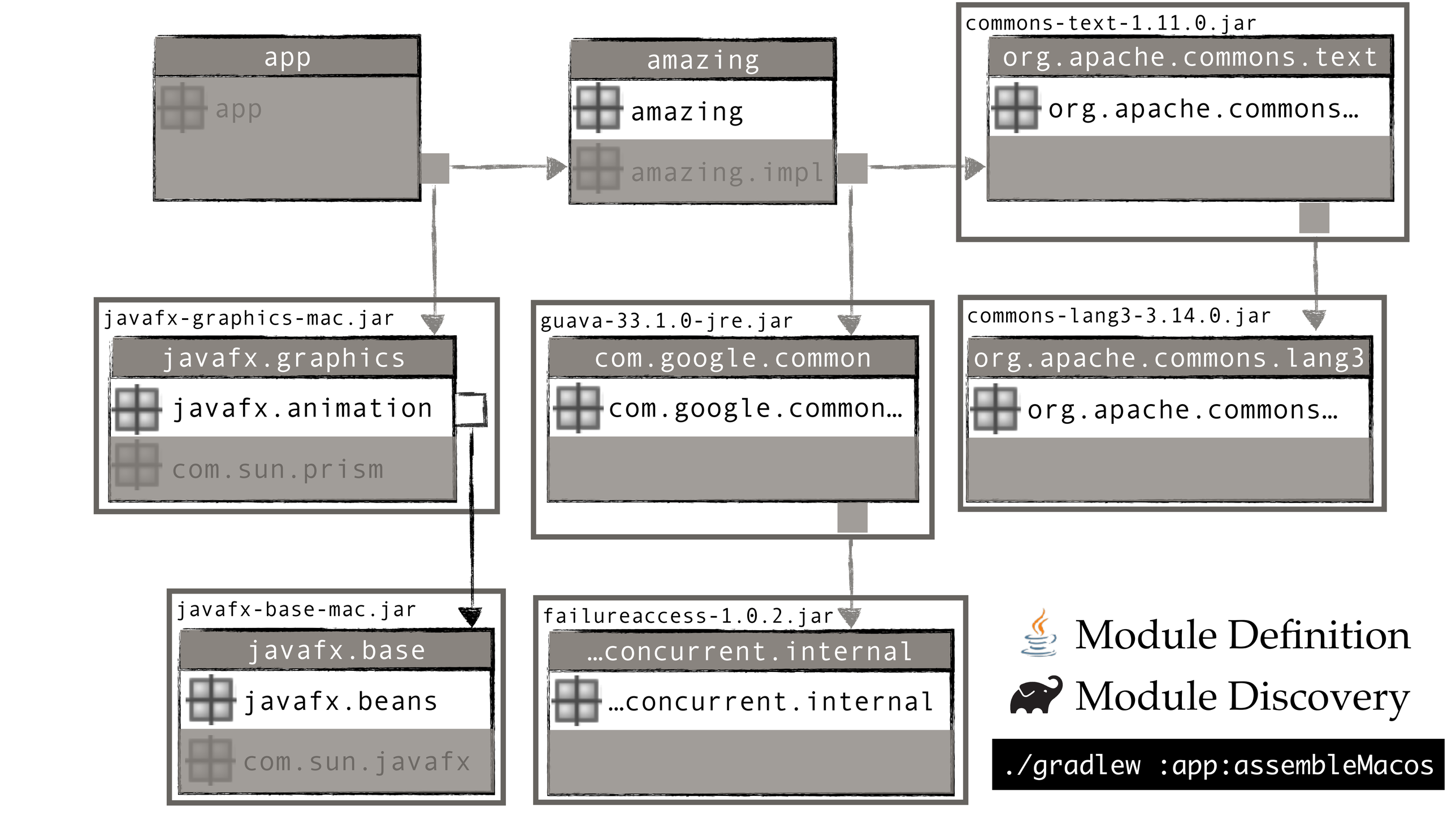
```
./gradlew :amazing:compileJava
```

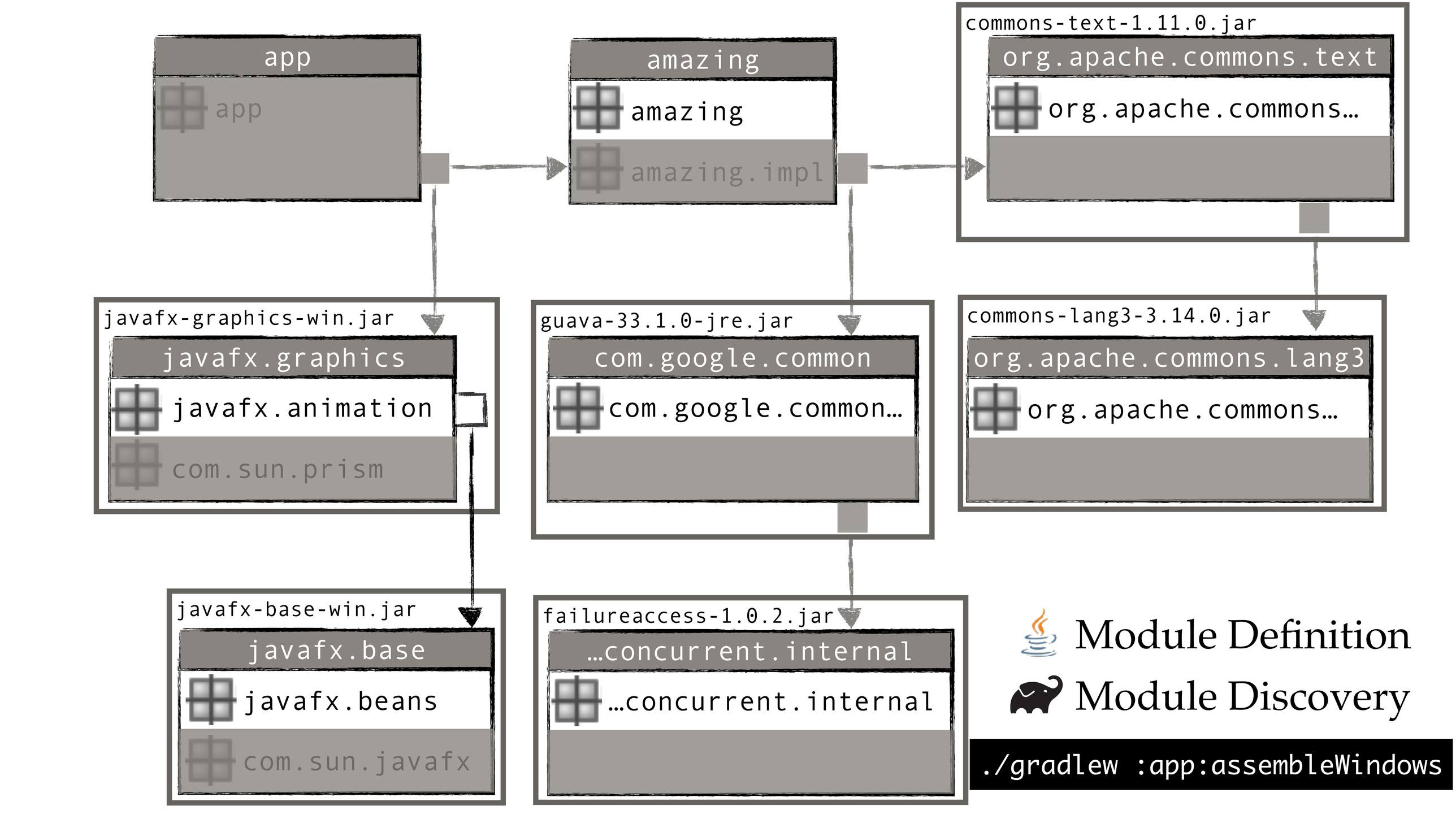


Module Definition

Module Discovery

```
./gradlew :app:compileJava
```


app
app
amazing
amazing
amazing.impl
commons-text-1.11.0.jar
org.apache.commons.text
org.apache.commons…
javafx-graphics-mac.jar
javafx.graphics
javafx.animation
com.sun.prism
guava-33.1.0-jre.jar
com.google.common
com.google.common…
commons-lang3-3.14.0.jar
org.apache.commons.lang3
org.apache.commons…
javafx-base-mac.jar
javafx.base
javafx.beans
com.sun.javafx
failureaccess-1.0.2.jar
…concurrent.internal
…concurrent.internal
Module Definition
Module Discovery
./gradlew :app:assembleMacos


app
app
amazing
amazing
amazing.impl
commons-text-1.11.0.jar
org.apache.commons.text
org.apache.commons…
javafx-graphics-win.jar
javafx.graphics
javafx.animation
com.sun.prism
guava-33.1.0-jre.jar
com.google.common
com.google.common…
commons-lang3-3.14.0.jar
org.apache.commons.lang3
org.apache.commons…
javafx-base-win.jar
javafx.base
javafx.beans
com.sun.javafx
failureaccess-1.0.2.jar
…concurrent.internal
…concurrent.internal
Module Definition
Module Discovery
./gradlew :app:assembleWindows

But in the real world … ?

Real World example: Hedera Services

github.com/hashgraph/hedera-services/blob/develop/hedera-node/hedera-consensus-service-impl/src/main/java/module-info.java

a module-info.java

github.com/hashgraph/hedera-services/blob/develop/hedera-node/hedera-consensus-service-impl/build.gradle.kts

a minimalistic build.gradle

github.com/hashgraph/hedera-services/blob/develop/hedera-dependency-versions/build.gradle.kts

versions

github.com/hashgraph/hedera-services/blob/develop/build-logic/project-plugins/src/main/kotlin/com.hedera.hashgraph.jpms-modules.gradle.kts

3rd party patching

github.com/hashgraph/hedera-services/tree/develop/build-logic

other build concerns

What if I can’t use the
Java Module System
for reason … ?

Module Definition

Java's module-info

- Module naming
- Module dependencies
  - Module private (requires)
  - Re-exported (req transitive)
  - Compile-only (req static)
- Module private packages

Module Discovery

Gradle Dependency Management

- Discover and build Modules from source on demand
- Discover Modules in Repositories by Name and Version
- Detect and handle Version and Variant conflicts
- Patch Module Metadata before handing it to Java tools

Module Definition

Java's module-info (build time only)

- Module naming
- Module dependencies
  - Module private (requires)
  - Re-exported (req transitive)
  - Compile-only (req static)
- ~~Module private packages~~

```
module amazing {
  requires org.apache.commons.text;
}
```

Module Discovery

Gradle Dependency Management

- Discover and build Modules from source on demand
- Discover Modules in Repositories by Name and Version
- Detect and handle Version and Variant conflicts
- Patch Module Metadata before handing it to Java tools

Module Definition

Gradle's build.gradle `dependencies`

- Module naming
- Module dependencies
  - Module private (implementation)
  - Re-exported (api)
  - Compile-only (compileOnly)
- ~~Module private packages~~

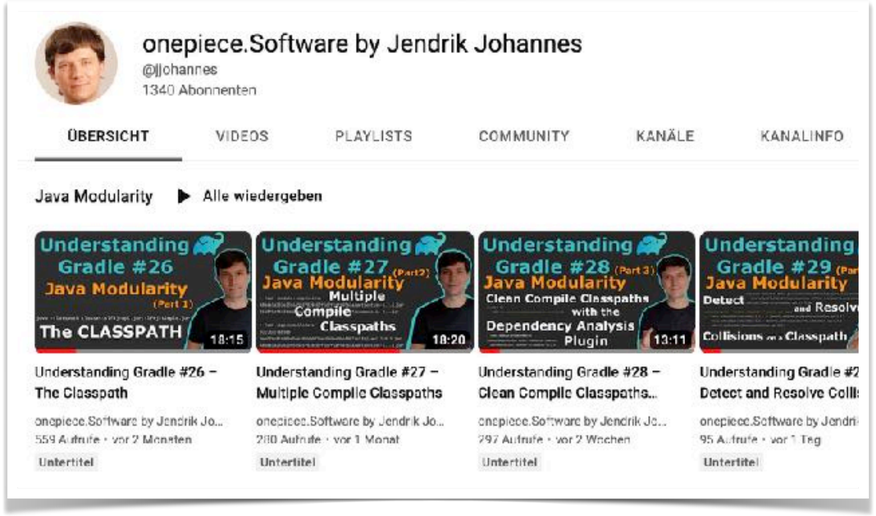
Module Discovery

Gradle Dependency Management

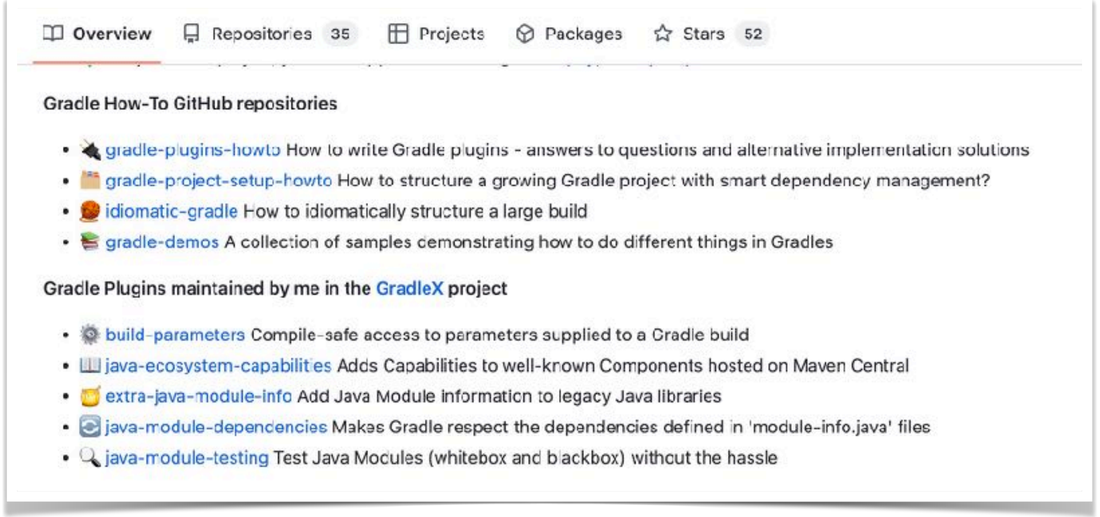
- Discover and build Modules from source on demand
- Discover Modules in Repositories by Name and Version
- Detect and handle Version and Variant conflicts
- Patch Module Metadata before handing it to Java tools

```
dependencies {
  implementation("org.apache.commons:commons-text")
}
```

slides / example / links
github.com/jjohannes/java-module-system



youtube.com/@jjohannes



github.com/jjohannes

stay in touch
mastodon.social/@jendrik
linkedin.com/in/jendrikjohannes

training and consulting
onepiece.software

GradleX Project
gradlex.org