

Plan and incompatibility toward Embulk 🐟 v1.0

Dai Mikurube / July 9, 2020



This presentation is about

- Embulk is going to introduce lots of plugin incompatibility through `v0.9` \rightarrow `v0.10` \rightarrow `v0.11` \rightarrow `v1`.
- This presentation is to explain the plan
 - for users -- which versions to choose in production?
 - for plugin developers -- how to catch up?
 - inside the core -- why? (if time allows)

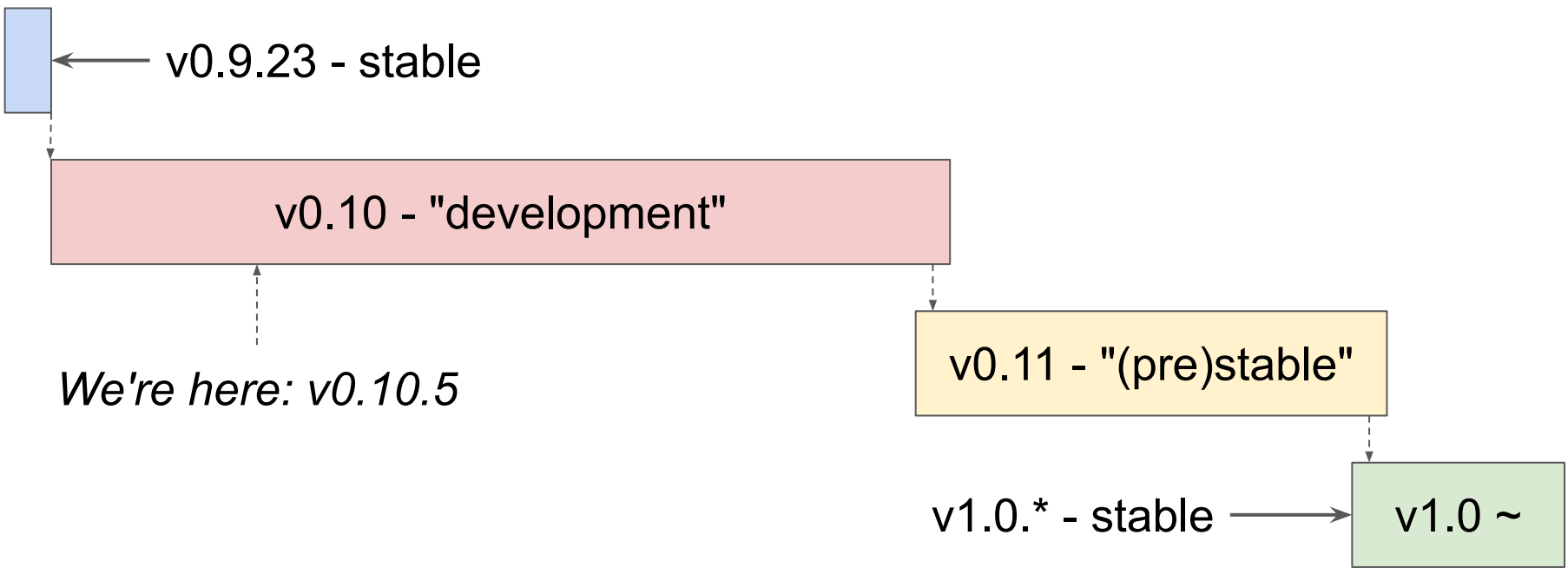
For Users
(everyone)





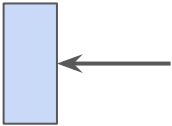
For Users
(everyone)

Planned versions



v0.9(.23)

For Users
(everyone)

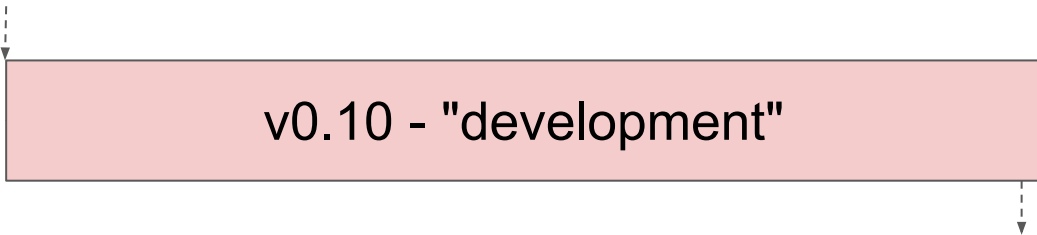
 ← v0.9.23 - stable

- Stable (as of July 2020)
- Only one production-ready version as of July 2020
- All latest plugins (as of July 2020) should work with it
- No more updates are expected in v0.9
 - Unless a backport is really required



For Users
(everyone)

v0.10



- "Development" unstable versions -- tries-and-errors
 - Not for your production! (TD uses it, though)
- Introducing several plugin incompatibility
- Deprecating several plugin API/SPI
- Plugins should be able to catch up to work both for the latest v0.9 (≡ v0.9.23) and the latest v0.10



For Users
(everyone)

How Embulk and plugins would update

When a plugin is updated to catch up with the latest `v0.10`, the latest version of the plugin should work with :

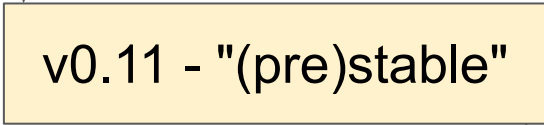
- The latest `v0.9` (= `v0.9.23` if no backports)
- The latest `v0.10`

`v0.10.*` is developed to satisfy the condition for plugins to work both with the latest `v0.9` and the latest `v0.10`

v0.11



For Users
(everyone)



v0.11 - "(pre)stable"

- "(Pre-)stable" versions -- should be production-ready
- `v0.11.0` will be identical with the last `v0.10`
- `v0.11.0` defines **v1-ready** API/SPI
- `v0.11.1+` removes items deprecated in `v0.10.*`
 - Many of legacy plugins (for `v0.9`) stop working (TBD)

v1.0

v1.0.* - stable

For Users
(everyone)

v1.0 ~

- Stable!
- **v1.0.0** will be identical with the last **v0.11**
 - Released when **v0.11.*** settles down, and gets confirmed



For Users
(everyone)

Plugin would / wouldn't work with vX.Y.Z ?

No catch-up

Catch-up in v0.10

v1-ready (v0.11+)

✓ v0.9.23

✓ v0.9.23 (or latest)

? v0.9.23

? v0.10.*

? v0.10.*

? v0.10.*

? the la(te)st v0.10

✓ the la(te)st v0.10

✓ the la(te)st v0.10

||

||

||

? v0.11.0

✓ v0.11.0

✓ v0.11.0

✗ v0.11.1+

? v0.11.1+

✓ v0.11.1+

✗ v1.0

? v1.0

✓ v1.0



For Users
(everyone)

Other decided incompatibility for every user

- JRuby is to be optional (not embedded in the core)

```
# TBD  
embulk -X jruby=file:///.../jruby-complete-9.1.15.0.jar run ??? .yml
```

- Bundler and Liquid: users to install by themselves? (TBD)

-
- **selfupdate** never finds the latest

```
embulk selfupdate          # It does not work since v0.10.0.  
embulk selfupdate X.Y.Z   # Specific version is always needed.
```

- **selfupdate** from **v0.9** does not update to **v0.10**

For
Developers





For
Developers

Two choices for plugin development

- Catch up accordingly with the latest Embulk `v0.10.*`
 - We (manage to) do it for github.com/embulk plugins
 - TD members may post Pull Requests for some other plugins
 - Especially for plugins that TD uses (Thanks!)
 - Helpful for the Embulk core team
 - Feedbacks would be reflected to `v0.11` and `v1`
- Catch up with v1-ready API/SPI when `v0.11.0` is out
 - One-shot catch-up



For
Developers

0) Ruby plugins

TBD

- Sorry, but the situation is complicated with Ruby plugins
- I guess most of the existing Ruby plugins would work as-is
 - Not guaranteed... we may see some unavoidable incompatibility
- Tests -- not working fine
 - Helps are welcome



For
Developers

1) gradle-embulk-plugins

- <https://github.com/embulk/gradle-embulk-plugins>
 - Gradle plugin to build Java-based Embulk plugins
- Recommended to apply for all plugins **NOW**
 - Either catch-up accordingly `v0.10.*`, or once `v0.11.0`
 - It helps building and releasing pure-Java plugins (Maven)
 - It enables checking plugin's dependencies explicitly
 - By Gradle "dependency lock"



For
Developers

1) gradle-embulk-plugins

```
plugins {  
    id "java"  
    id "checkstyle"  
    id "jacoco"  
}
```

```
version = "X.Y.Z"
```

```
plugins {  
    id "java"  
    id "checkstyle"  
    id "jacoco"  
    id "maven-publish"  
    id "org.embulk.embulk-plugins"  
        version "0.4.1"  
}
```

```
group = "<your Maven group name>"
```

```
// group = "io.github.your-user-name"
```

```
//           is typical
```

```
// group = "org.embulk"
```

```
//           is only for github.com/embulk
```

```
version = "X.Y.Z"
```

```
description "<'description' of Gem>"
```




For
Developers

1) gradle-embulk-plugins

```
configurations { provided }

dependencies {
    compile "org.embulk:embulk-core:.."
    provided "org.embulk:embulk-core:.."

    // May conflict with embulk-core.
    compile "...:~:~:~:~:~:"
}

task classpath(...) {
    from (configurations.runtime
        - configurations.provided
        + files(jar.archivePath))
    into "classpath"
}
```

```
dependencies {
    compileOnly "org.embulk:embulk-core:.."

    // Explicit conflict handling.
    compile(...:~:~:~:~:~) {
        exclude group: "...", module: "..."
    }
}
```



For
Developers

1) gradle-embulk-plugins

```
$ ./gradlew build
```

```
...
```

```
===== WARNING =====  
Following "runtime" dependencies are included also in "compileOnly" dependencies.
```

```
"com.fasterxml...:jackson-core:2.6.7" // <= Conflicting with embulk-core's dependencies.
```

"compileOnly" dependencies are used to represent Embulk's core to be "provided" at runtime. They should be excluded from "compile" or "runtime" dependencies like the example below.

```
dependencies {  
    compile("org.glassfish.jersey.core:jersey-client:2.25.1") {  
        exclude group: "javax.inject", module: "javax.inject"  
    }  
}
```

```
...
```



For
Developers

1) gradle-embulk-plugins

```
task gem(type: JRubyExec, ...) {
  jrubyArgs "-S"
  script "gem"
  scriptArgs "build", ".../build/gemspec"
}

task gemspec { doLast {
  file('build').mkdirs()
  file('build/gemspec').write($/
Gem::Specification.new do |spec|
  spec.name      = "${project.name}"
  spec.version  = "${project.version}"
  spec.author   = ["... .."]
  spec.summary  = %[...]
  ...
end
-/$)
} }
```

```
embulkPlugin { // <== lib/embulk/...rb
  mainClass = "..."
  category = "input"
  type = "..."
}

gem {
  authors = [ "... .." ]
  email = [ "..." ]
  summary = "..."
  homepage = "..."
  licenses = [ "..." ]
}

gemPush {
  host = "https://rubygems.org"
}

// gemspec and .rb are auto-generated.
```



For
Developers

1) gradle-embulk-plugins

```
publishing {
  publications {
    // Publish it with "publishEmbulkPluginMavenPublicationToMavenRepository".
    embulkPluginMaven(MavenPublication) {
      from components.java // Must be "components.java".
    }
  }
  repositories {
    maven {
      // Any Maven repository you want to release to!
      name = "mavenCentral"
      url "https://oss.sonatype.org/service/local/staging/deploy/maven2"
      credentials {
        username = project.hasProperty("ossrhUsername") ? ossrhUsername : ""
        password = project.hasProperty("ossrhPassword") ? ossrhPassword : ""
      }
    }
  }
}
```



1) gradle-embulk-plugins

```
$ ./gradlew dependencies --write-locks
...

$ cat gradle/dependency-locks/embulkPluginRuntime.lockfile

# This is a Gradle generated file for dependency locking.
# Manual edits can break the build and are not advised.
# This file is expected to be part of source control.
com.fasterxml.jackson.core:jackson-annotations:2.6.7
com.fasterxml.jackson.core:jackson-core:2.6.7
com.fasterxml.jackson.core:jackson-databind:2.6.7
com.fasterxml.jackson.datatype:jackson-datatype-jdk8:2.6.7
com.jcraft:jsch:0.1.55
...

$ git add gradle/dependency-locks/embulkPluginRuntime.lockfile
```



For
Developers

2) Dependencies - source of incompatibility

If your Java plugin is (directly or indirectly) using

- Jackson, Guava, Apache Commons Lang 3, javax.validation
 - Include them explicitly in your plugin's dependencies
 - Choose **versions 100% same with embulk-core** for now
 - `gradle-embulk-plugins` will warn, but you can ignore it
 - You can mark "ignored" explicitly in `embulkPlugin { }`
- JRuby, Joda-Time, Logback, Guice
 - Stop using them -- find an alternative, or give up using it
 - Ex. Joda-Time → `java.time` classes



For
Developers

2) Dependencies - source of incompatibility

```
embulkPlugin {
  mainClass = "..."/>
  category = "input"
  type = "..."/>
  ignoreConflicts = [
    [ group: "com.fasterxml.jackson.core", module: "jackson-annotations" ],
    [ group: "com.fasterxml.jackson.core", module: "jackson-core" ],
    ...
  ]
}
```

```
===== WARNING =====
Following "runtime" dependencies are included also in "compileOnly" dependencies.
```

```
[[IGNORED] "com.fasterxml.jackson.core:jackson-annotations:2.6.7" ]
[[IGNORED] "com.fasterxml.jackson.core:jackson-core:2.6.7" ]
...
```



For
Developers

2) Dependencies - source of incompatibility

- Dependencies of **embulk-core** is now visible from plugins
 - Jackson, Guava, Joda-Time, JRuby, ...
- Some of them could be passed from the core to plugins

```
interface PluginTask extends Task {  
    @Config("example")  
    JsonNode getExample();  
}
```

```
public ConfigDiff transaction(  
    ConfigSource config, ...) {  
    ObjectNode object = config.getObjectNode();  
}
```

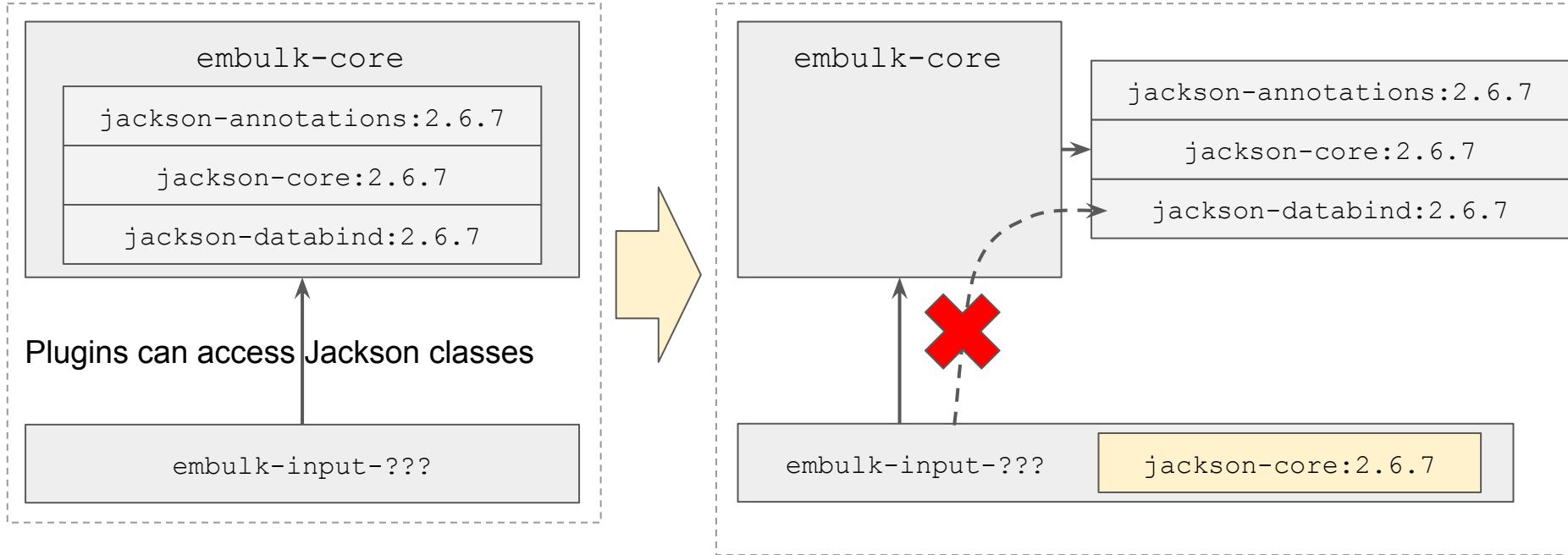
- It has caused problems between the core and plugins
 - Conflicts, broken when upgrading a library, ...



For
Developers

2) Dependencies - source of incompatibility

- **embulk-core** will keep using those dependencies internally, but they will be **hidden** from plugins during **v0.10**





For
Developers

2) Dependencies - source of incompatibility

- Dependencies below will stay visible from plugins:
 - `slf4j-api` (no logger implementation -- Logback)
 - `javax.inject` (not Guice)
 - `msgpack-core` -- TBD: maybe only its model classes



For
Developers

3) Prepare for Java 9+ (TBD: target → 11?)

- [JEP 320](#): `javax.xml` (& more) is removed from JRE since 11
- `embulk-core` will **NOT** provide it in place of JRE
 - A plugin using `javax.xml` will need to include them by itself

```
dependencies {  
    ...  
  
    compile "javax.activation:javax.activation-api:..."  
    compile "javax.xml.bind:jaxb-api:..."  
    compile "com.sun.xml.bind:jaxb-core:..."  
    compile "com.sun.xml.bind:jaxb-impl:..."  
}
```



For
Developers

4) Depend only on `embulk-api/spi`

- Plugins would **NOT** depend on `embulk-core`, but :
 - [embulk-api](#) (Started v0.10.1 ~ work-in-progress)
 - `embulk-spi` (To start v0.10.6 or later)
- Moving API/SPI from `embulk-core` to them
- `embulk-api/spi` will be *Documented*
- Others remaining in `embulk-core` will be *Undocumented*

```
dependencies {  
  compileOnly "...:embulk-core:..."  
}
```

```
dependencies {  
  compileOnly "...:embulk-api:..."  
  compileOnly "...:embulk-spi:..."  
}
```

5) Core features exported to libraries

Along with `embulk-api/spi`, utility classes for plugins are exported out of `embulk-core`, into external libraries

- `TimestampFormatter / TimestampParser`
 - ⇒ [embulk-util-timestamp](#) ([Javadoc](#))
- `ConfigSource#loadConfig / TaskSource#loadTask`
 - ⇒ [embulk-util-config](#) ([Javadoc](#))
- ... to be continued

6) Miscellaneous deprecation

- `Exec.getLogger` → `org.slf4j.LoggerFactory.getLogger`
- `Guava Optional` → `java.util.Optional`
- `Guava Throwables` → `throw RuntimeException` (or else)
 - <https://github.com/google/guava/wiki/Why-we-deprecated-Throwables.propagate>
- `Timestamp` → `java.time.Instant`
 - `embulk-util-timestamp` handles `java.time.Instant`
 - `Timestamp` would remain in some interfaces, though
- `@ConfigInject` → Use `Exec.get???` () instead
- `ModelManager` → Build your own `ObjectMapper`



For
Developers

Join Slack `embulk-dev`

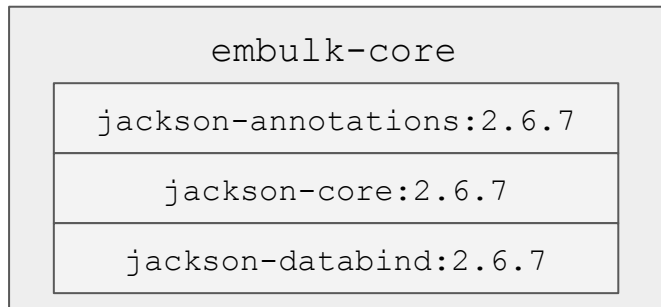
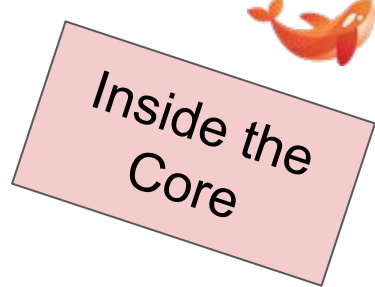
- **No user support**
 - Only for plugin (or core) developers
- Announcement, discussion, and Q&A for `v0.10+` catch-ups
- To join :
 - Leave a message at: <https://github.com/embulk/embulk/issues/1222>
 - Nice to leave a note about your plugins
 - I'll invite your GitHub email address (in commits)
 - Or, contact `@dmikurube` directly in some way

Appendix

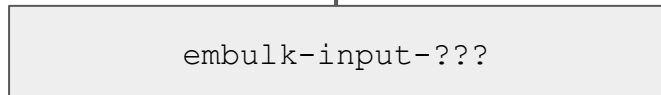
Inside the Core



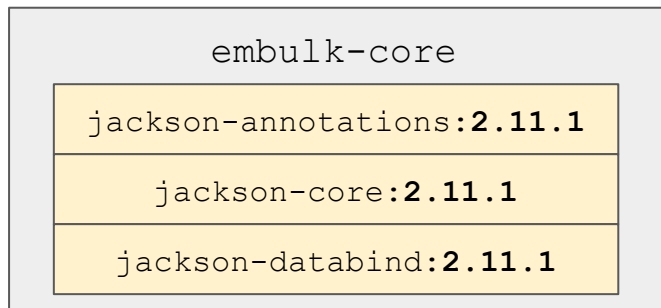
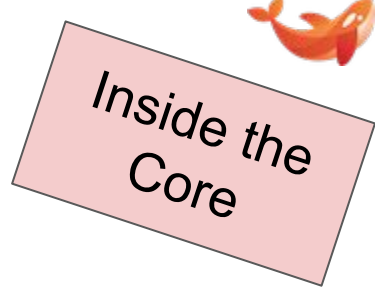
What's the problem? (Trivial case)



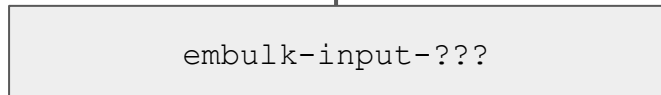
Using Jackson expecting 2.6.7



What's the problem? (Trivial case)



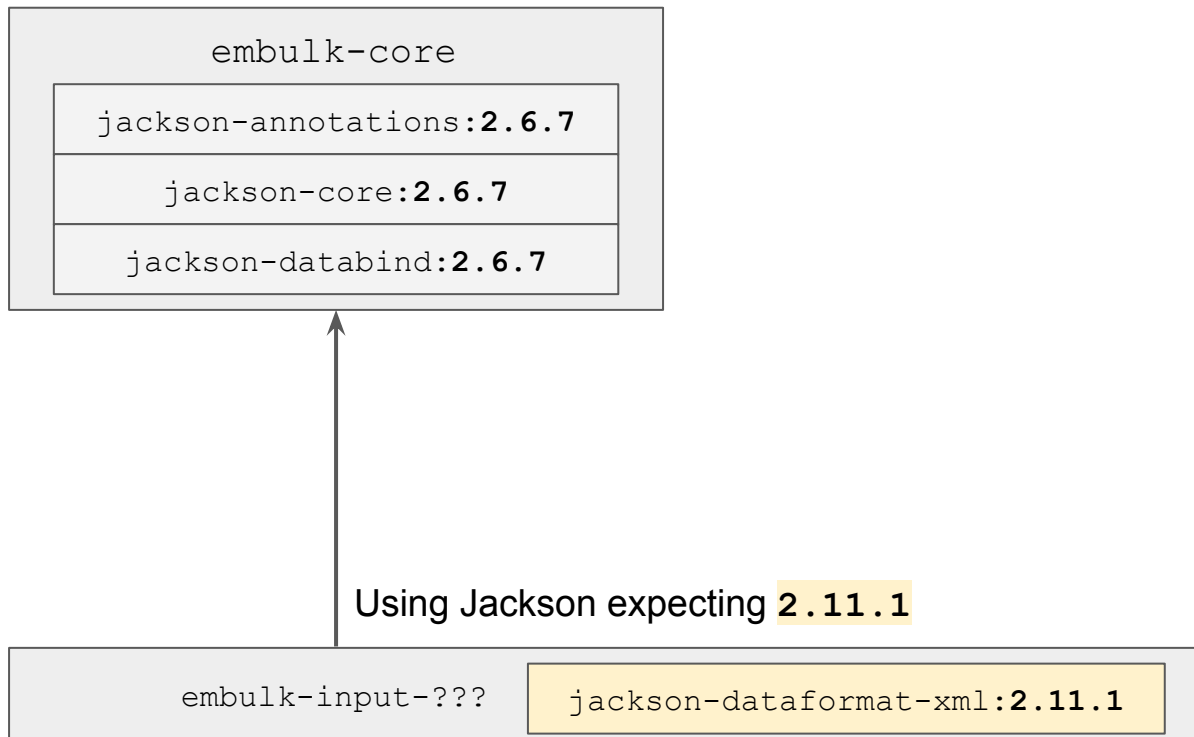
Using Jackson expecting 2.6.7



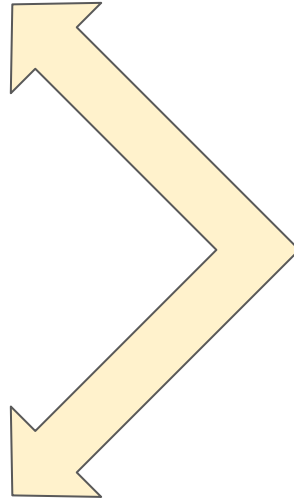
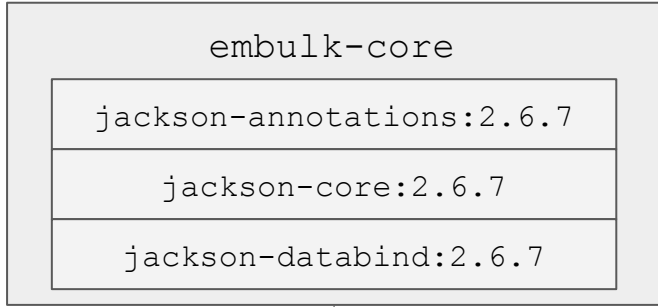
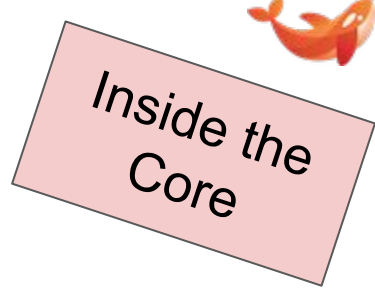


Inside the Core

What's the problem? (Trivial case)

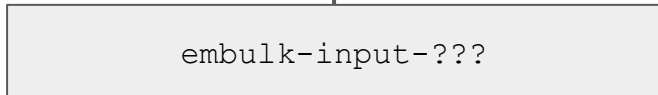


What's the problem? (Trivial case)



Standoff (deadlock)

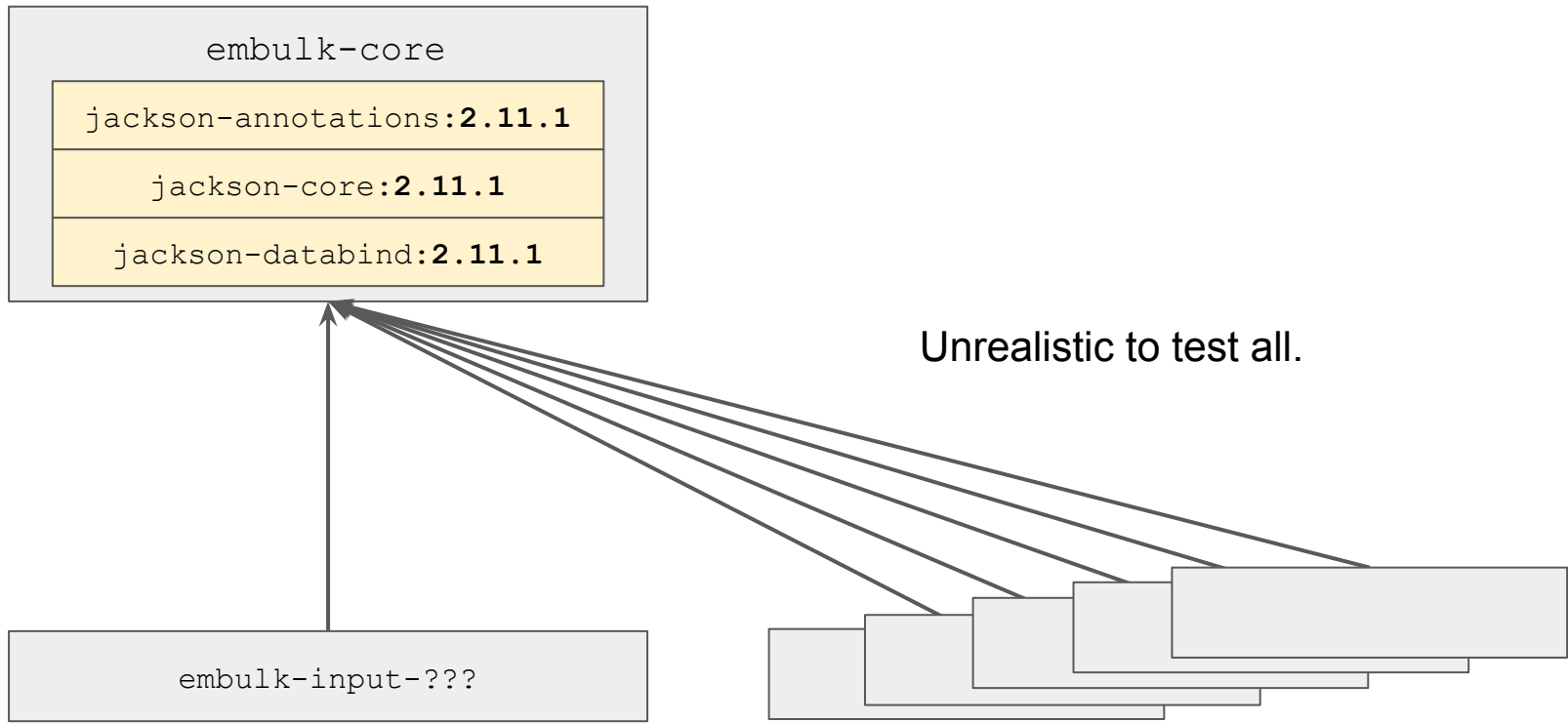
Using Jackson expecting 2.6.7





Inside the Core

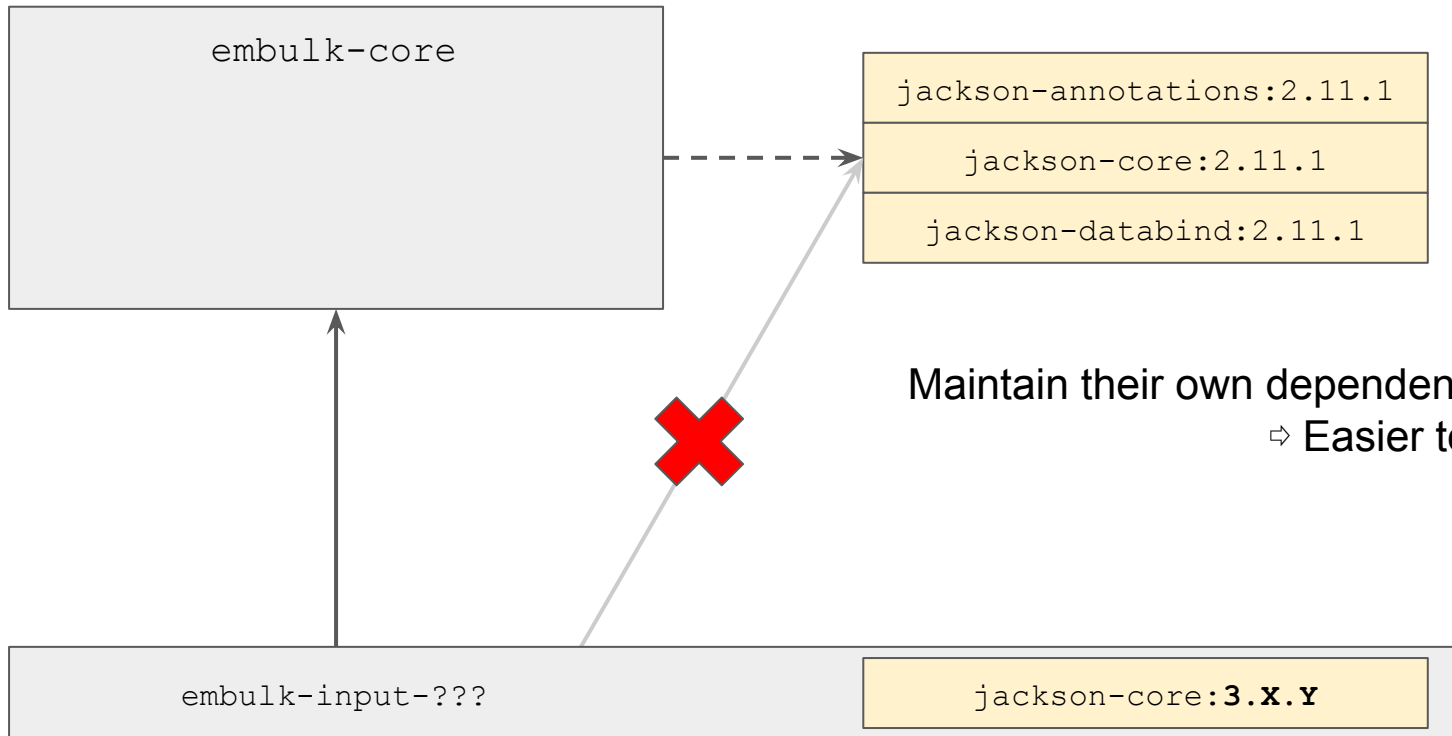
What's the problem? (Trivial case)



Unrealistic to test all.

What's wanted to achieve

Inside the Core

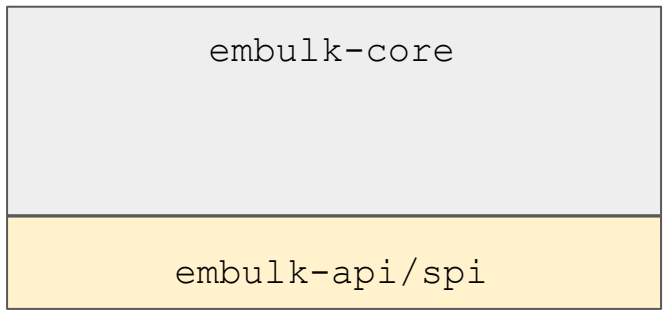


Maintain their own dependencies.

⇒ Easier to test each.

What's wanted to achieve

Inside the Core



Loose coupling.

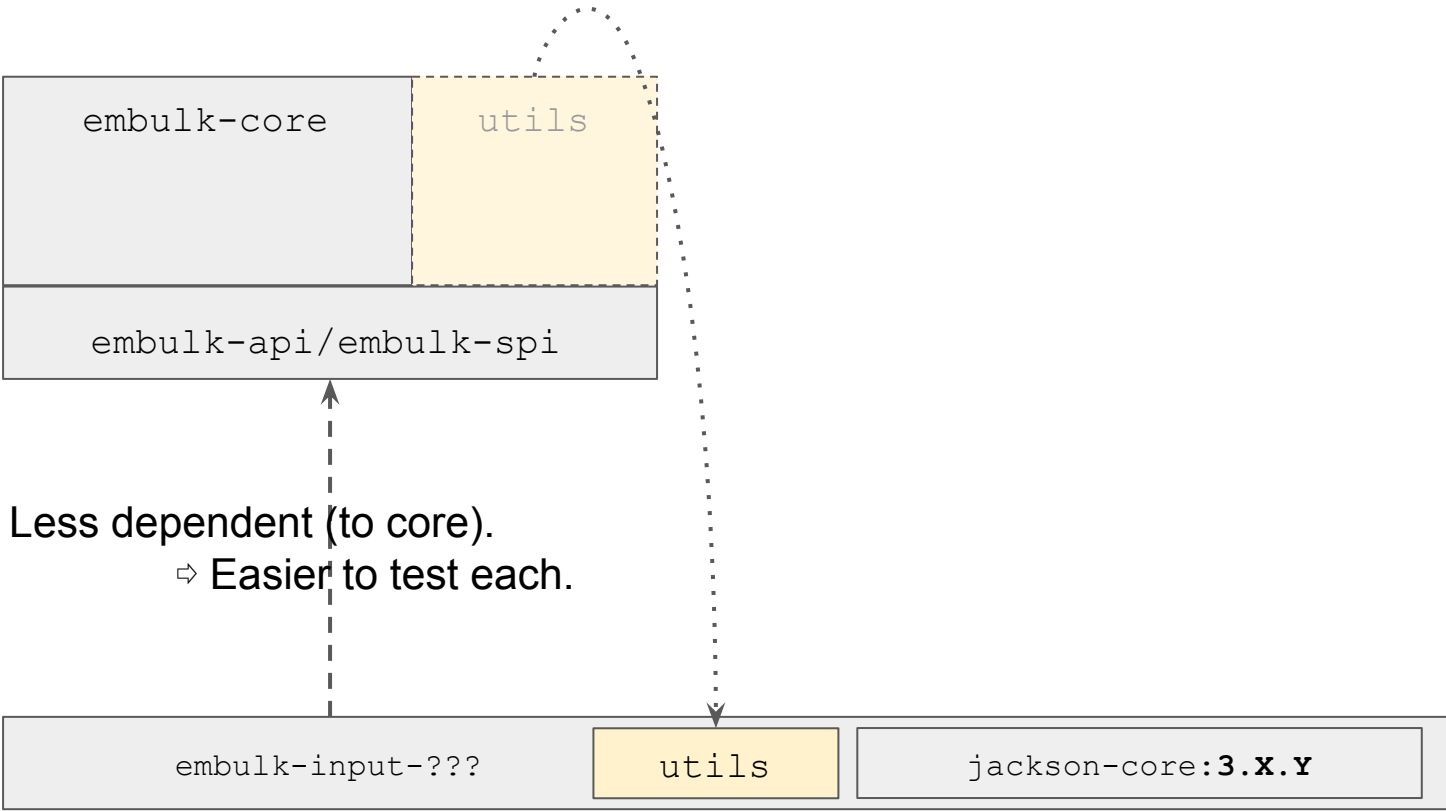
⇒ Easier to test each.





Inside the Core

What's wanted to achieve



Less dependent (to core).
⇒ Easier to test each.

What's wanted to achieve

Inside the Core

