OK, we're switching Gradle. What's next?

Pasha Finkelshteyn

Congratulations!

What you already know:

Congratulations!

What you already know:

1. You will (finally!) write in
   - Yay, autocompletion!!!

Congratulations!

What you already know:

1. You will (finally!) write in
   - Yay, autocompletion!!!
2. You'll finally get rid of </>

Congratulations!

What you already know:

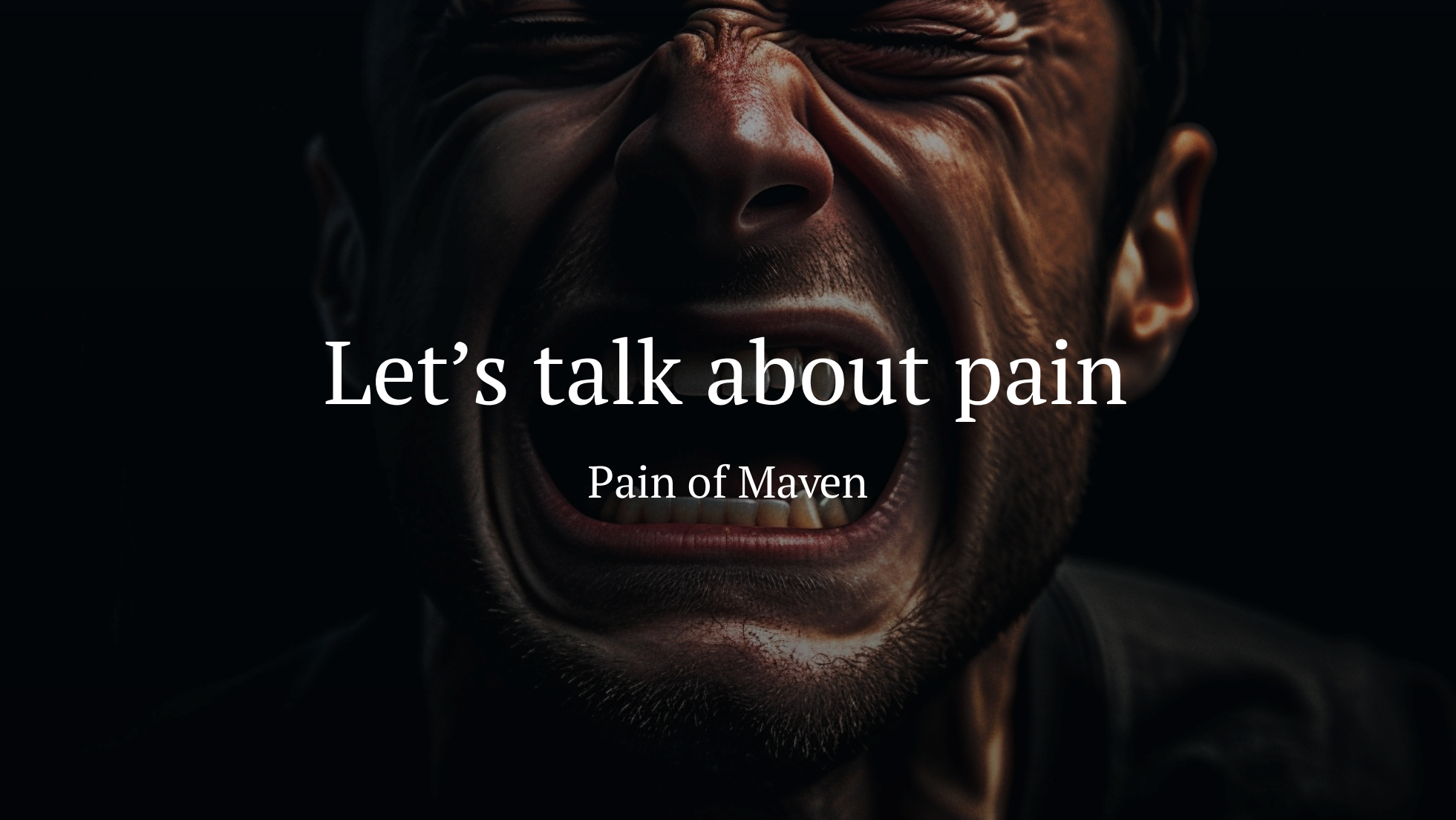
1. You will (finally!) write in
   - Yay, autocompletion!!!
2. You'll finally get rid of </>

This talk is about less obvious things.

And your concerns.

Here is what I know about your company:

- You already use Gradle for your Android projects
- You have one Java monorepo
- You have small Java satellite repositories
- You current primary build system is Maven (with antrun mixins)
- You have your own plugins



Let’s talk about pain

Pain of Maven

Current pains

- Dependency conflict resolution is *crazy* complicated
- Lifecycle is not flexible enough
- Different build systems for different languages
- Slow builds of monorepo
- Complex dependencies between modules

Dependency conflict resolution in Maven

```
A
├── B
│   └── C
│       └── D 2.0
└── E
    └── D 1.0
```

How will the dependency be resolved?

1. Exception
2. 1.0
3. 2.0
4. Depends on the code

Dependency conflict resolution in Maven

```
A
├── B
│   └── C
│       └── D 2.0
└── E
    └── D 1.0
```

How will the dependency be resolved?

1. Exception
2. 1.0
3. 2.0
4. Depends on the code

Dependency conflict resolution in Maven

```
A
├── B
│   └── C
│       └── D 2.0
└── E
    └── D 1.0
```

How will the dependency be resolved?

1. Exception
2. 1.0
3. 2.0
4. Depends on the code

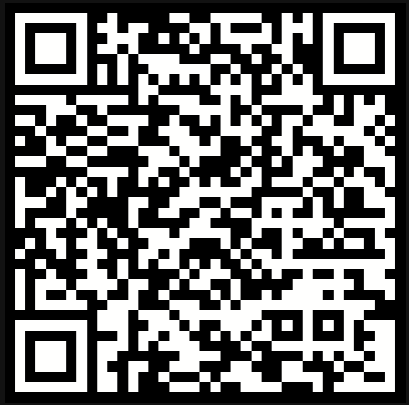
🤯 What are the chances that `A` won't work?

Not the case with Gradle!

Gradle dependency resolution

- Gradle will select the *highest* one
- Gradle is semver-aware
- Rich version declarations helps to make the best decision possible

Rich version declaration ↓



Maven Lifecycle

Maven Lifecycle

Is it flexible?

On which phase do you obfuscate your `jar` file? `pre-integration-test`? `post-integration-test`?

Not the case with Gradle!

Gradle lifecycle

You define the Gradle Lifecycle!

```
project("project-a") {
    tasks.register("taskX") {
        dependsOn(":project-b:taskY")
        doLast {
            println("taskX")
        }
    }
}
```

Gradle lifecycle

You define the Gradle Lifecycle!

```
project("project-a") {
    tasks.register("taskX") {
        dependsOn(":project-b:taskY")
        doLast {
            println("taskX")
        }
    }
}
```

Gradle lifecycle

You define the Gradle Lifecycle!

```
project("project-a") {
    tasks.register("taskX") {
        dependsOn(":project-b:taskY")
        doLast {
            println("taskX")
        }
    }
}
```

Gradle lifecycle

You define the Gradle Lifecycle!

```
1  project("project-a") {
2      tasks.register("taskX") {
3          dependsOn(":project-b:taskY")
4          doLast {
5              println("taskX")
6          }
7      }
8  }
```

Gradle lifecycle

You define the Gradle Lifecycle!

```
project("project-a") {
    tasks.register("taskX") {
        dependsOn(":project-b:taskY")
        doLast {
            println("taskX")
        }
    }
}
```

Gradle lifecycle

You define the Gradle Lifecycle!

```
project("project-a") {
    tasks.register("taskX") {
        dependsOn(":project-b:taskY")
        doLast {
            println("taskX")
        }
    }
}
```

Gradle lifecycle

You define the Gradle Lifecycle!

```
project("project-a") {
    tasks.register("taskX") {
        dependsOn(":project-b:taskY")
        doLast {
            println("taskX")
        }
    }
}
```

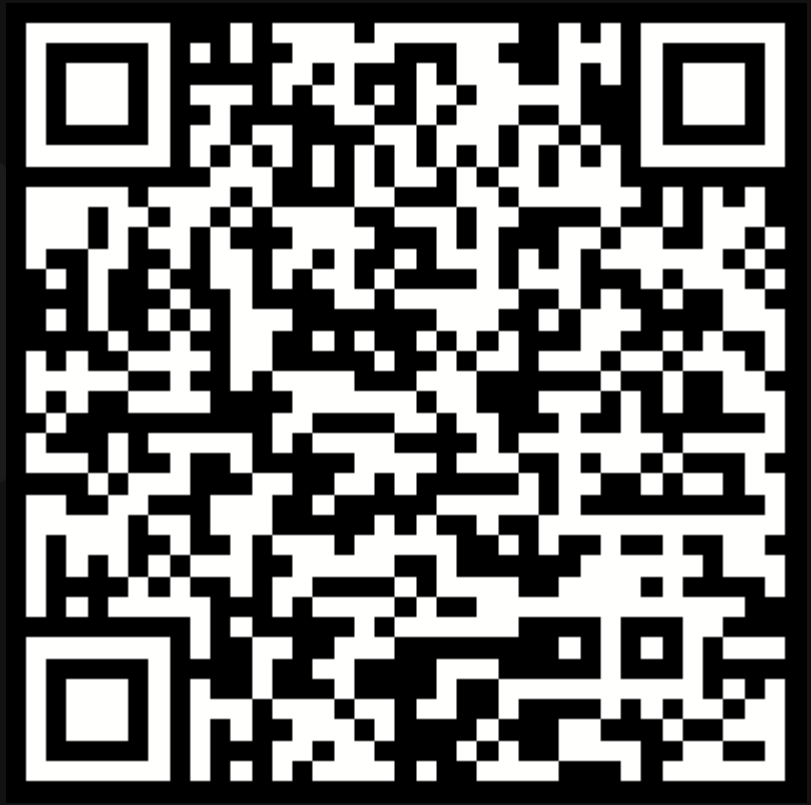
And you can define graph as complex as you need!

Gradle lifecycle

You define the Gradle Lifecycle!

```
project("project-a") {
    tasks.register("taskX") {
        dependsOn(":project-b:taskY")
        doLast {
            println("taskX")
        }
    }
}
```

And you can define graph as complex as you need!



Slow build of the monorepo

Incremental builds to the rescue!

Slow build of the monorepo

Incremental builds to the rescue!

You can use the maven-incremental build plugin, if your project has hundreds of modules. It saves lot of time.

Share Edit Follow Flag

edited Jan 31, 2016 at 13:22

asgs

3,918 ● 6 ● 39 ● 54

answered Jan 19, 2012 at 6:41

Pradeep Fernando

659 ● 3 ● 8

Slow build of the monorepo

Incremental builds to the rescue!

You can use the maven-incremental build plugin, if your project has hundreds of modules. It saves lot of time.

Share Edit Follow Flag

edited Jan 31, 2016 at 13:22

asgs

3,918 ● 6 ● 39 ● 54

answered Jan 19, 2012 at 6:41

Pradeep Fernando

659 ● 3 ● 8

Java.net

Not the case with Gradle!

Gradle speeds things up

Gradle speeds things up

1. Gradle daemon

Gradle speeds things up

1. Gradle daemon
2. Gradle incremental build (`org.gradle.caching=true`)

Gradle speeds things up

1. Gradle daemon
2. Gradle incremental build (`org.gradle.caching=true`)
3. Multithreading OOTB

Gradle speeds things up

1. Gradle daemon
2. Gradle incremental build (`org.gradle.caching=true`)
3. Multithreading OOTB
4. Configuration caching (`org.gradle.configuration-cache=true`)

Gradle speeds things up

1. Gradle daemon
2. Gradle incremental build (`org.gradle.caching=true`)
3. Multithreading OOTB
4. Configuration caching (`org.gradle.configuration-cache=true`)
5. Lazy task configuration

Gradle speeds things up

1. Gradle daemon
2. Gradle incremental build (`org.gradle.caching=true`)
3. Multithreading OOTB
4. Configuration caching (`org.gradle.configuration-cache=true`)
5. Lazy task configuration
6. Gradle enterprise helps with anomaly detection and understanding

Maven inter-module dependencies

What if you have a library?

I know, any big organization has their own libraries and this is cool!

But what happens when you library depends on, say, HTTP client?

It will be available to all the clients of the library!

Not the case with Gradle!

Api dependencies in Gradle

Module is just another flavor of library

- `api` dependencies will be visible to those who depend on a library
- `implementation` dependencies won't be

Just change preferred HTTP client without breaking others!

```
1   api(project(":core"))
```

This ↑ is how dependency on other module looks

Profiles

I had a `profile` hell!

For Tomcat I had a crazy profiles to download the correct `tcnative` library.

```
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-antrun-plugin</artifactId>
<executions>
  <execution>
    <phase>initialize</phase>
    <configuration>
      <exportAntProperties>true</exportAntProperties>
      <target>
        <condition property="tcnative.classifier" value="${os.detected.classifier}-fedora" else="${os.detected.classi
          <isset property="os.detected.release.fedora"/>
        </condition>
      </target>
    </configuration>
    <goals>
      <goal>run</goal>
    </goals>
  </execution>
</executions>
```

Not the case with Gradle!

Just use `if`s

You have all the JVM's power at your disposal

Migration

Migration

Basic scenario

```
gradle init
```

Migration

Basic scenario

```
gradle init
```

- Will generate basic `build.gradle`
- With scopes defined as accurate as possible

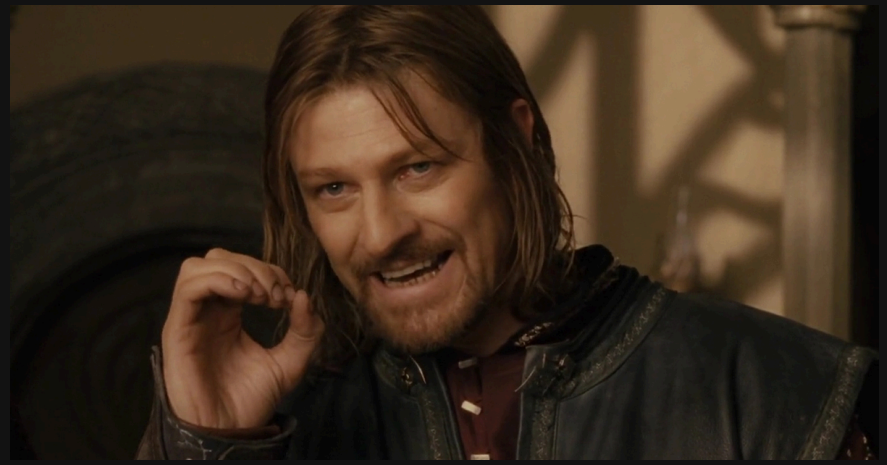
Migration

Better scenario

1. Create a Build Scan for Maven with Gradle Enterprise
2. Learn how to compare artifacts
3. `gradle init`
4. Create a Build Scan for Gradle
5. Compare results until match

But what about our plugins?

Plugin migration

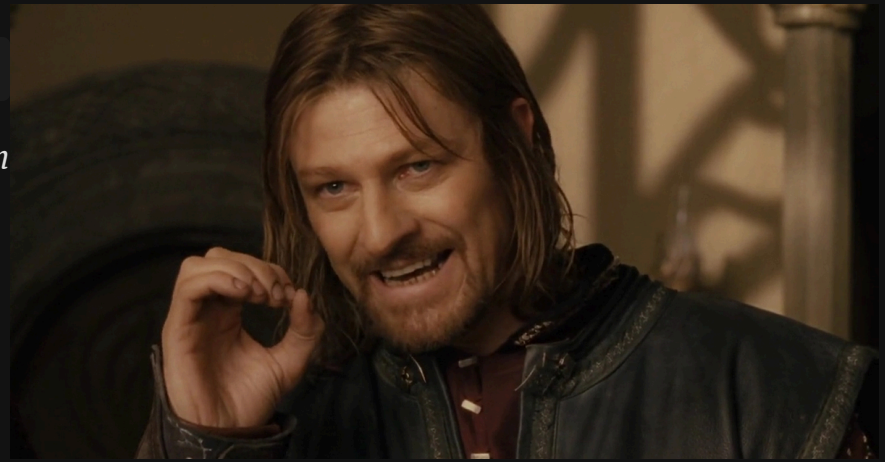


Plugin migration

Bad news:

> One does not simply reuse Maven plugins in Gradle

Ben Franklin

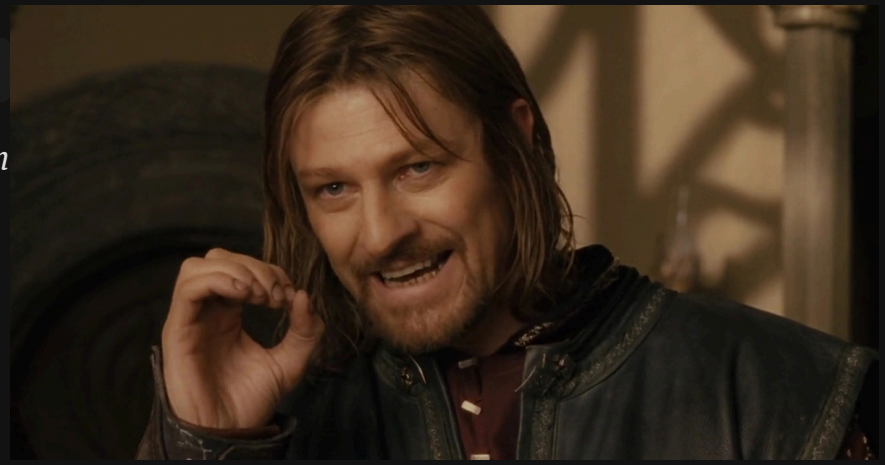


Plugin migration

Bad news:

> One does not simply reuse Maven plugins in Gradle

Ben Franklin



Good news:

1. Plugin ecosystem is **mammoth** 🦣.
   Probably somebody already did what you need
2. Writing plugins for Gradle is order of magnitude easier
3. We have awesome docs
4. We can help you!

Summary

What might gradle do for you?

1. Make your builds faster
2. Resolve conflicts better
3. Write in a *normal* ™ programming language
4. Give you more flexibility

Do you have issues with Maven?

Hopefully it’s not the case with Gradle!

Thank you!

It’s time for your questions!