THE JAVA SOFTWARE MODEL

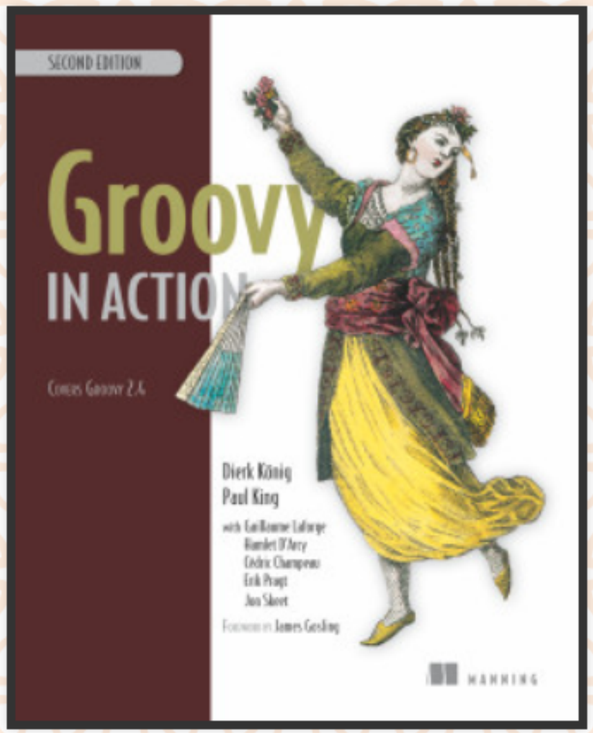
by Cédric Champeau (@CedricChampeau)

June 1st - 3rd 2016
Copenhagen, Denmark

WHO AM I

```
speaker {
    name 'Cédric Champeau'
    company 'Gradle Inc'
    oss 'Apache Groovy committer',
    successes (['Static type checker',
                'Static compilation',
                'Traits',
                'Markup template engine',
                'DSLs'])
        failures Stream.of(bugs),
        twitter '@CedricChampeau',
        github 'melix',
        extraDescription '''Groovy in Action 2 co-author
Misc OSS contribs (Gradle plugins, deck2pdf, jlangdetect, ...)'''
}
```

GROOVY IN ACTION 2

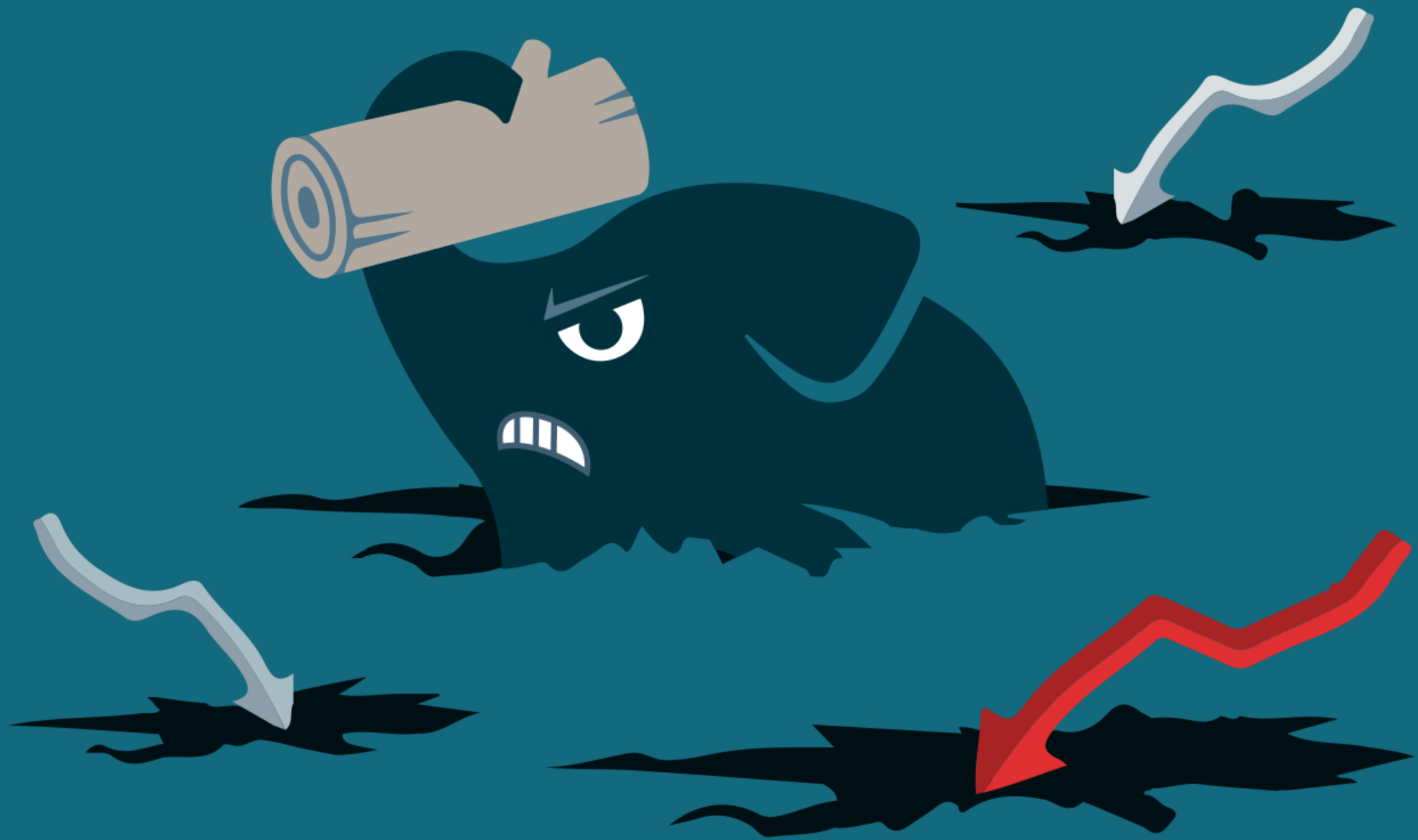


https://www.manning.com/books/groovy-in-action-second-edition

Coupon *ctwgr8conftw*

WHY GRADLE

End Broken Release Processes



End Bug Regressions

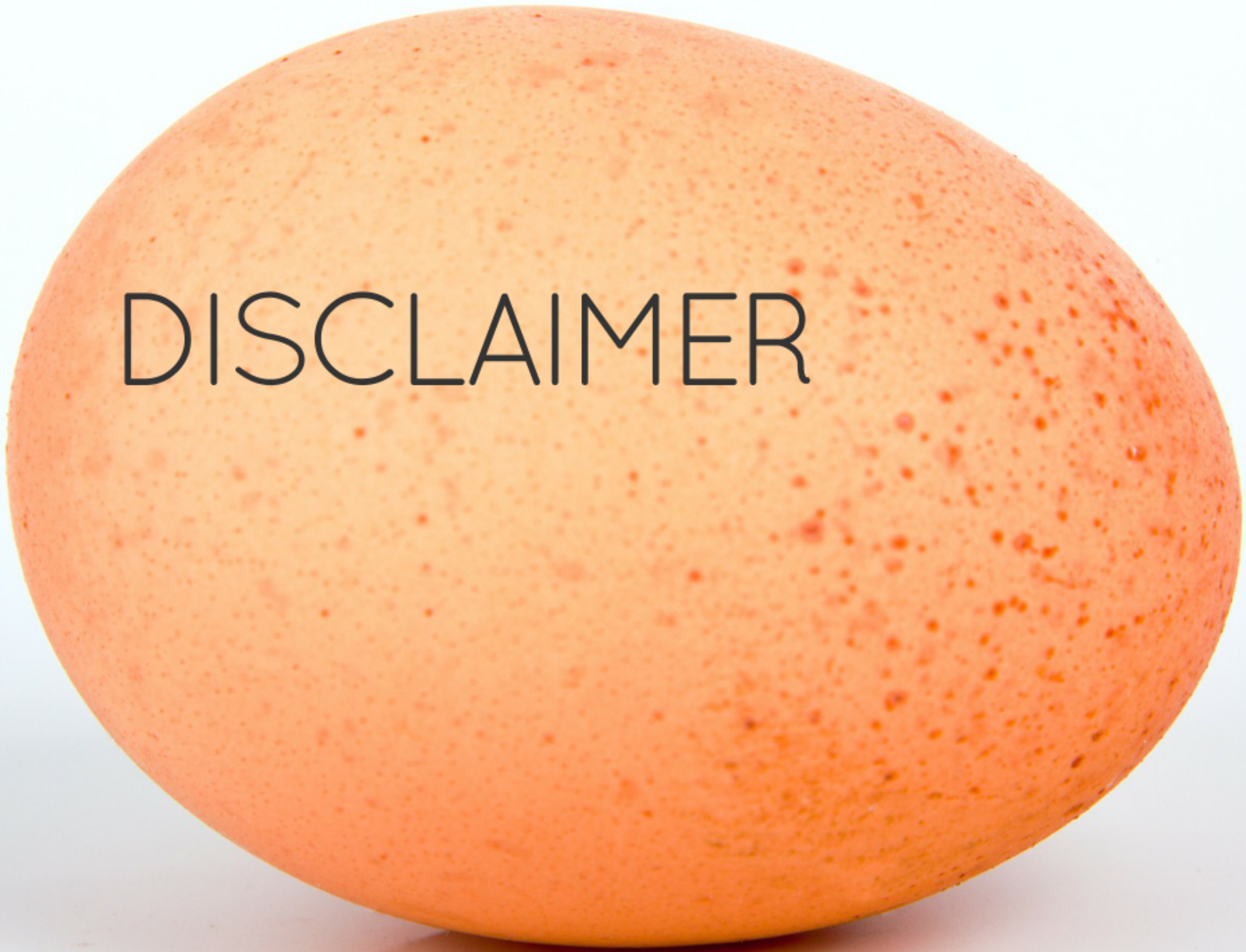
End Build Script Chaos

End Code Freeze

End Deathmarches

End Long Build Times

DISCLAIMER



CURRENT MODEL

- Convention based

```
apply plugin: 'application'
allprojects {
   apply plugin: 'java'
}
```

- Coupled
  - Determining which tasks need to be *executed* is easy
  - Determining which tasks need to be *configured* is hard
  - Configuring properly is hard (`afterEvaluate`…)

WHY A NEW MODEL

- Polyglot programming
- Multiple variants
- Scalability
- Continuous delivery
- Speed

EXAMPLE: MODELLING A JAVA LIBRARY

APPLY THE JAVA SOFTWARE MODEL PLUGINS

```
plugins {
    id 'jvm-component'
    id 'java-lang'
}
```

DECLARE A LIBRARY

```
model {
    components {
        main(JvmLibrarySpec)
    }
}
```

THE RULE ENGINE

- New way to write plugins
- New way to configure builds
- But likely to change soon
- More in the variant-aware dependency management talk

MODELLING A JAVA APPLICATION

APPLICATION WITH A DEPENDENT LIBRARY

```
model {
    components {
        app(JvmLibrarySpec)
        myLib(JvmLibrarySpec)
    }
}
```

PROJECT LAYOUT

```
src
 |-- app
 |     |-- java
 |-- myLib
       |-- java
```

DECLARE THE DEPENDENCY ONTO THE LIBRARY

```
model {
    components {
        app(JvmLibrarySpec) {
            sources {
                java {
                    dependencies {
                        library 'myLib'
                    }
                }
            }
        }
        myLib(JvmLibrarySpec)
    }
}
```

ADD A DEPENDENCY ONTO AN EXTERNAL LIBRARY

```
model {
    components {
        app(JvmLibrarySpec) {
            sources {
                java {
                    dependencies {
                        library 'myLib'
                        module 'org.ow2.asm:asm:5.0.4'
                    }
                }
            }
        }
        myLib(JvmLibrarySpec)
    }
}
```

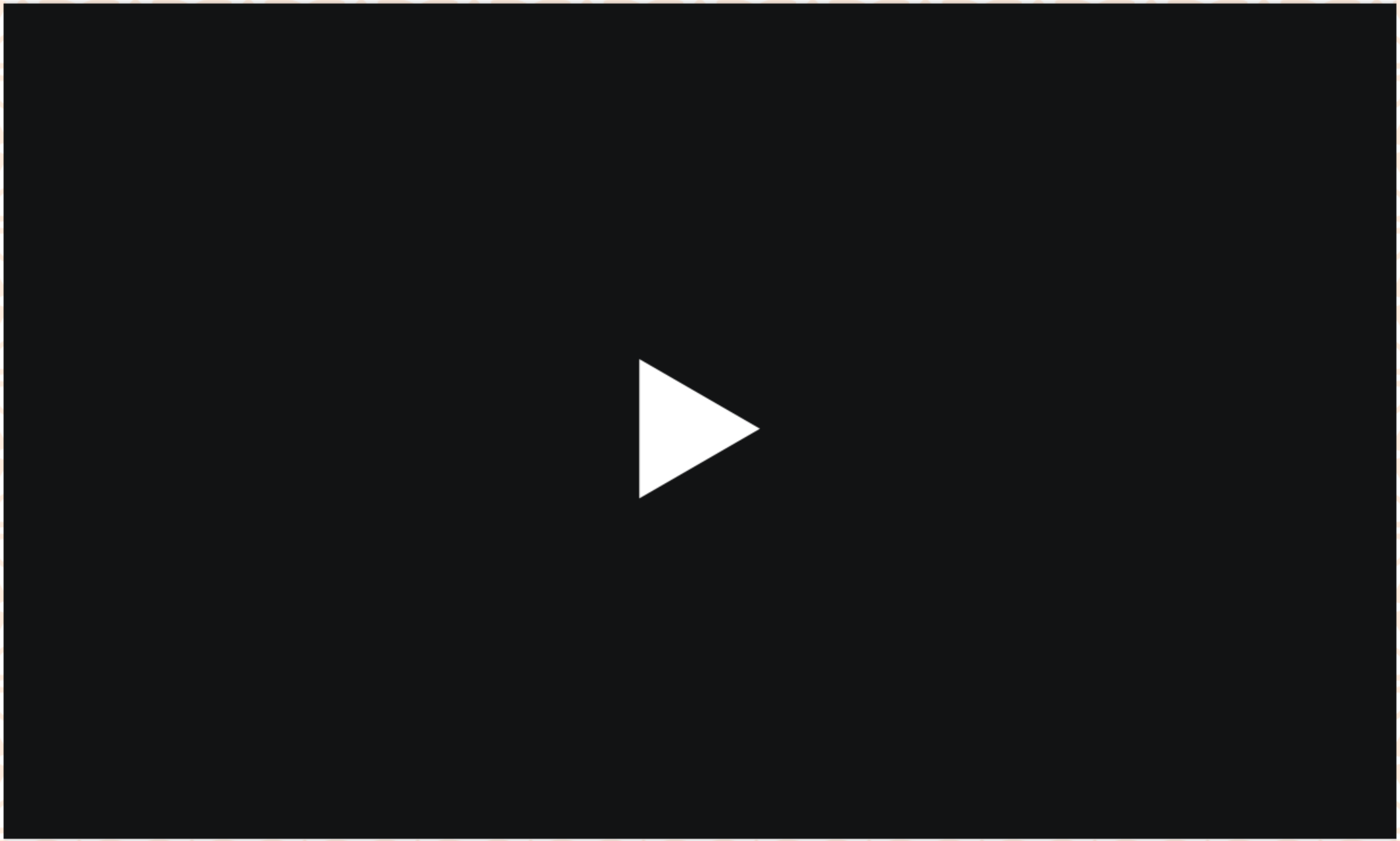
COMPONENT LEVEL DEPENDENCIES

```
model {
    components {
        app(JvmLibrarySpec) {
            dependencies {
                library 'myLib'
                module 'org.ow2.asm:asm:5.0.4'
            }
        }
        myLib(JvmLibrarySpec)
    }
}
```

API VS IMPLEMENTATION

```
model {
    components {
        myLib(JvmLibrarySpec) {
            api {
                exports 'com.acme.mylib' // not recursive!
            }
        }
    }
}
```

IN ACTION



00:00

API VS IMPLEMENTATION BENEFITS

- Strong encapsulation
  - No more private API leakage
    - Prepare for Jigsaw today!
- Compile avoidance
  - If private API changes
  - If public API changes in ABI compatible way

API DEPENDENCIES

```
model {
    components {
        myLib(JvmLibrarySpec) {
            api {
                dependencies {
                    library 'com.google.guava:guava:17.0'
                }
            }
        }
    }
}
```

TARGET PLATFORMS

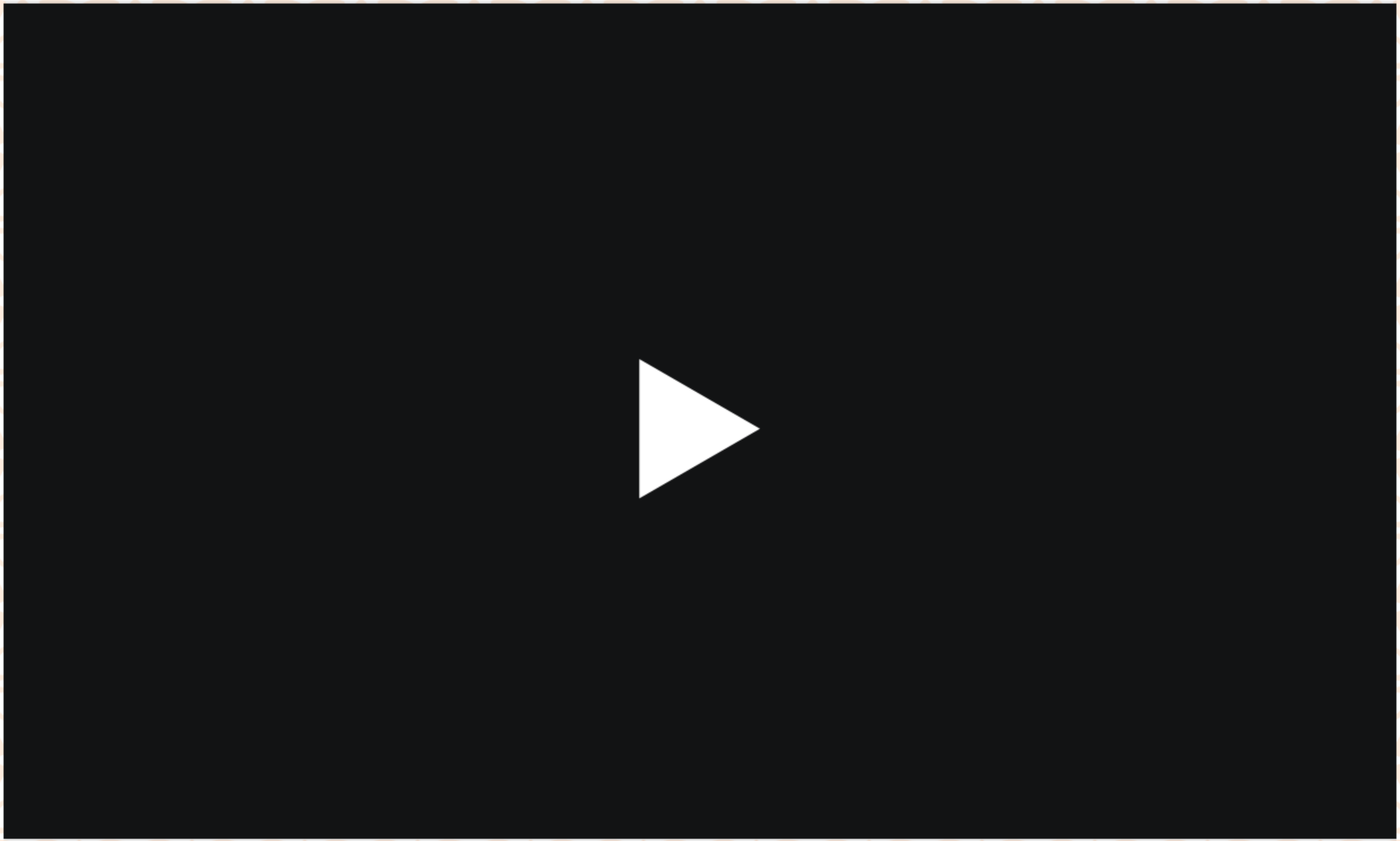
DECLARING TARGET PLATFORMS

```
model {
    components {
        myLib(JvmLibrarySpec) {
            targetPlatform 'java7'
            targetPlatform 'java8'
        }
    }
}
```

BINARIES

- `myLib.jar` for Java 7
- `myLib.jar` for Java 8
- Each one can have its own source sets/dependencies.

VARIANT AWARE



00:00


TESTING

JUNIT

- So far only `junit` is supported
- Adding more frameworks should be easy

```
plugins {
    id 'jvm-component'
    id 'java-lang'
    id 'junit-test-suite'
}
```

DECLARING A TEST SUITE

```
model {
    components {
        main(JvmLibrarySpec)
    }
    testSuites {
        test(JUnitTestSuiteSpec) {
            jUnitVersion '4.12'
            testing $.components.main
        }
    }
}
```

TEST SUITES

- Have JUnit version specified as a first-class model element
- Could support multiple target versions of JUnit
- Have their own target platforms
- Target a specific component

FUTURE WORK

TOOLCHAIN SUPPORT

DECLARING JDKS

```
model {
    javaInstallations {
        openJdk6(LocalJava) {
            path '/usr/lib/jvm/jdk1.6.0-amd64'
        }
        oracleJre7(LocalJava) {
            path '/usr/lib/jvm/jre1.7.0'
        }
        ibmJdk8(LocalJava) {
            path '/usr/lib/jvm/jdk1.8.0'
        }
    }
}
```

USING JDKS

- Automatic detection
- Automatic selection of toolchain
- Test on various platforms

MORE MODELS!

```
model {
    myPlugin(GradlePlugin) {
        targetGradleVersions '2.14', '3.0'
        // ...
    }
}
```

MORE MODELS!

```
model {
    microservice(SpringBootApplication) {
        springBootVersion '1.3.5'
        // ...
    }
}
```

MORE MODELS!

```
model {
    dockerImage(DockerImage) {
        from 'alpine:3.2'
        // ...
    }
}
```

BUT...

WE'RE HIRING!

http://gradle.org/gradle-jobs/

THANK YOU!

- Slides and code : https://github.com/melix/gr8conf2016/intro-software-model
- Follow me: @CedricChampeau

ATTRIBUTIONS

- Tests: https://www.flickr.com/photos/otacke/12221514614
- Q&A: https://www.flickr.com/photos/oberazzi/318947873

EXTRA TIME: THE RULE ENGINE

MANAGED TYPES

```
@Managed
interface ImageComponent extends LibrarySpec {
    String getTitle()
    void setTitle(String title)
    List<String> getSizes()
    void setSizes(List<String> sizes)
}
```

RULES

```
class MyImageRenderingPlugin extends RuleSource {
    @ComponentType
    void registerComponent(TypeBuilder<ImageComponent> builder) {
    }

    @ComponentType
    void registerBinary(TypeBuilder<ImageBinary> builder) {
    }
...
```

BINARIES

```
@Managed
interface ImageBinary extends BinarySpec {
    String getTitle()
    void setTitle(String title)
    String getSize()
    void setSize(String size)
}
```

CREATING BINARIES

```
@ComponentBinaries
void createBinariesForBinaryComponent(ModelMap<ImageBinary> binarie
    library.sizes.each { fontSize ->
        binaries.create(fontSize) {
            size = fontSize;
            title = library.title
        }
    }
}

@BinaryTasks
void createRenderingTasks(ModelMap<Task> tasks, ImageBinary binary)
    tasks.create(binary.tasks.taskName("render", "svg"), RenderSvg)
        it.content = binary.title;
        it.fontSize = binary.size;
        it.outputFile = new File(it.project.buildDir, "renderedSvg/
    }
```

GENERIC RULES

```
@Mutate
public void registerPlatformResolver(PlatformResolvers platformR
    platformResolvers.register(new JavaPlatformResolver());
}

@Model
JavaInstallationProbe javaInstallationProbe(ServiceRegistry serv
    return serviceRegistry.get(JavaInstallationProbe.class);
}
```