The ideal module system ...

... and the harsh reality

Adam Warski
SoftwareMill

New languages

Are we focusing on the wrong problem?

About me

- During the day: coding @ SoftwareMill
- SoftwareMill: a great software house!
- Afternoon: playgrounds, Duplo, etc.
- Evening: blogging, open-source
  - Original author of Hibernate Envers
  - ElasticMQ, Veripacks, MacWire
  - http://www.warski.org

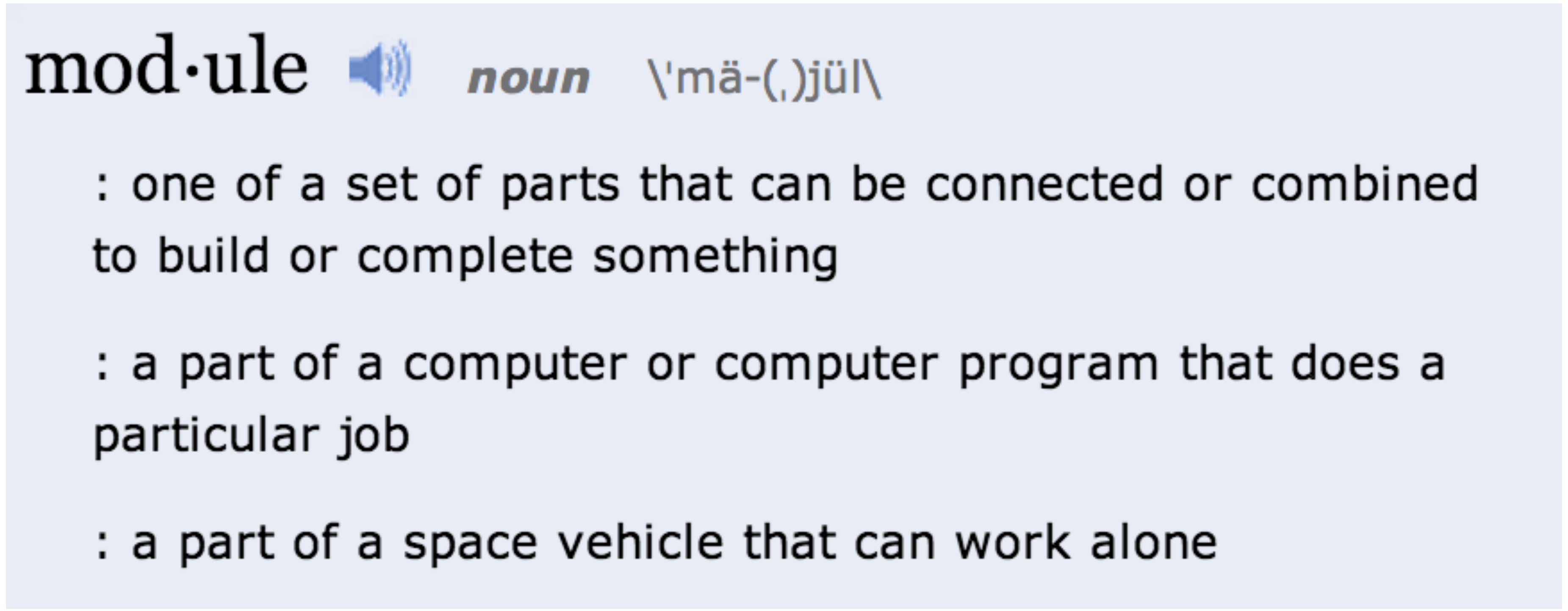
Plan for the next 50 minutes

- Ideal module system
- Veripacks
- Ceylon

What is a module?

- OSGi bundle?
- Jigsaw-something?
- Maven build module?
- Guice module?
- Ruby module?

What is a module?


mod·ule noun \'mä-(ˌ)jül\

: one of a set of parts that can be connected or combined to build or complete something

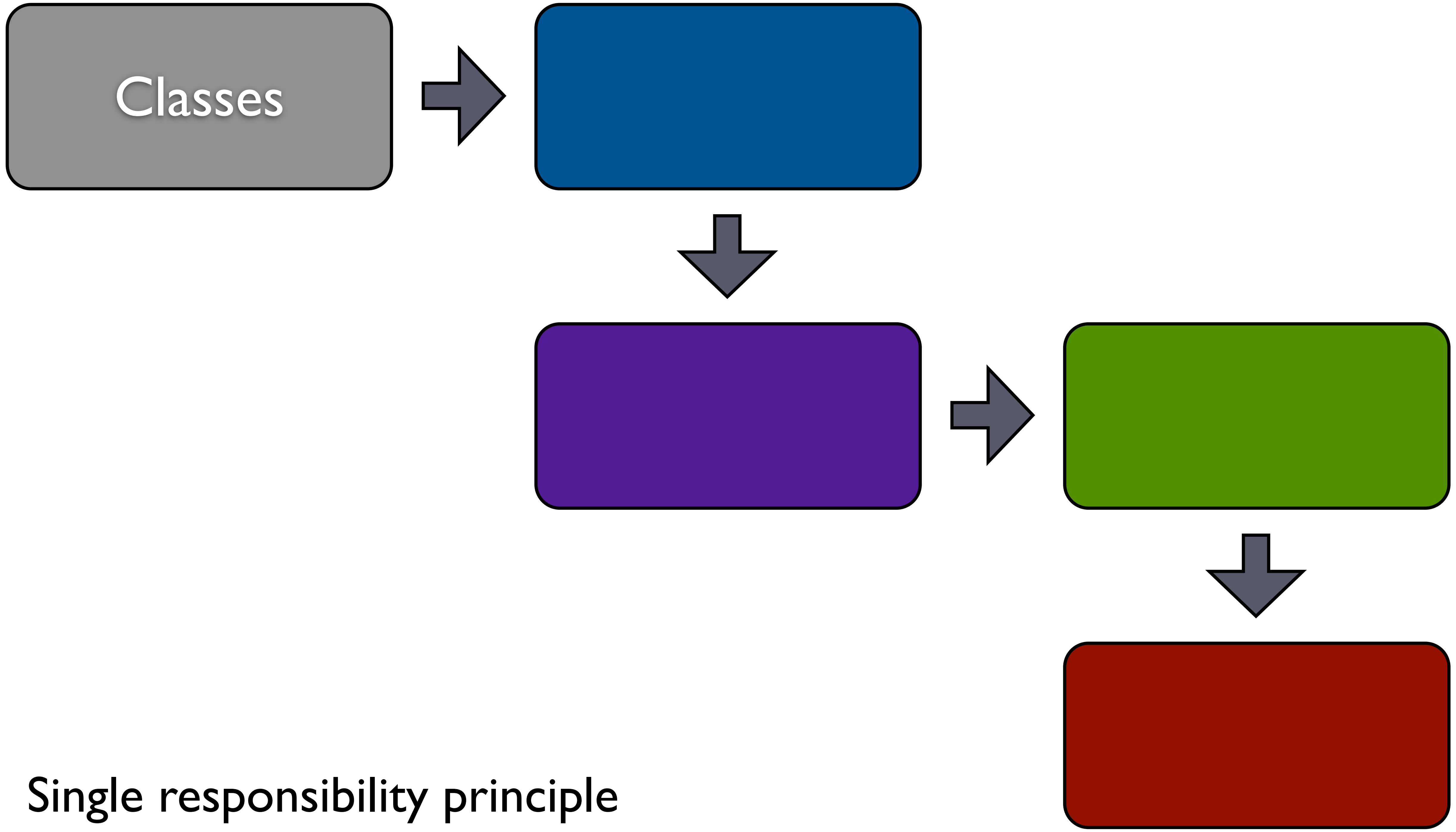
: a part of a computer or computer program that does a particular job

: a part of a space vehicle that can work alone

Source: http://www.merriam-webster.com/dictionary/module

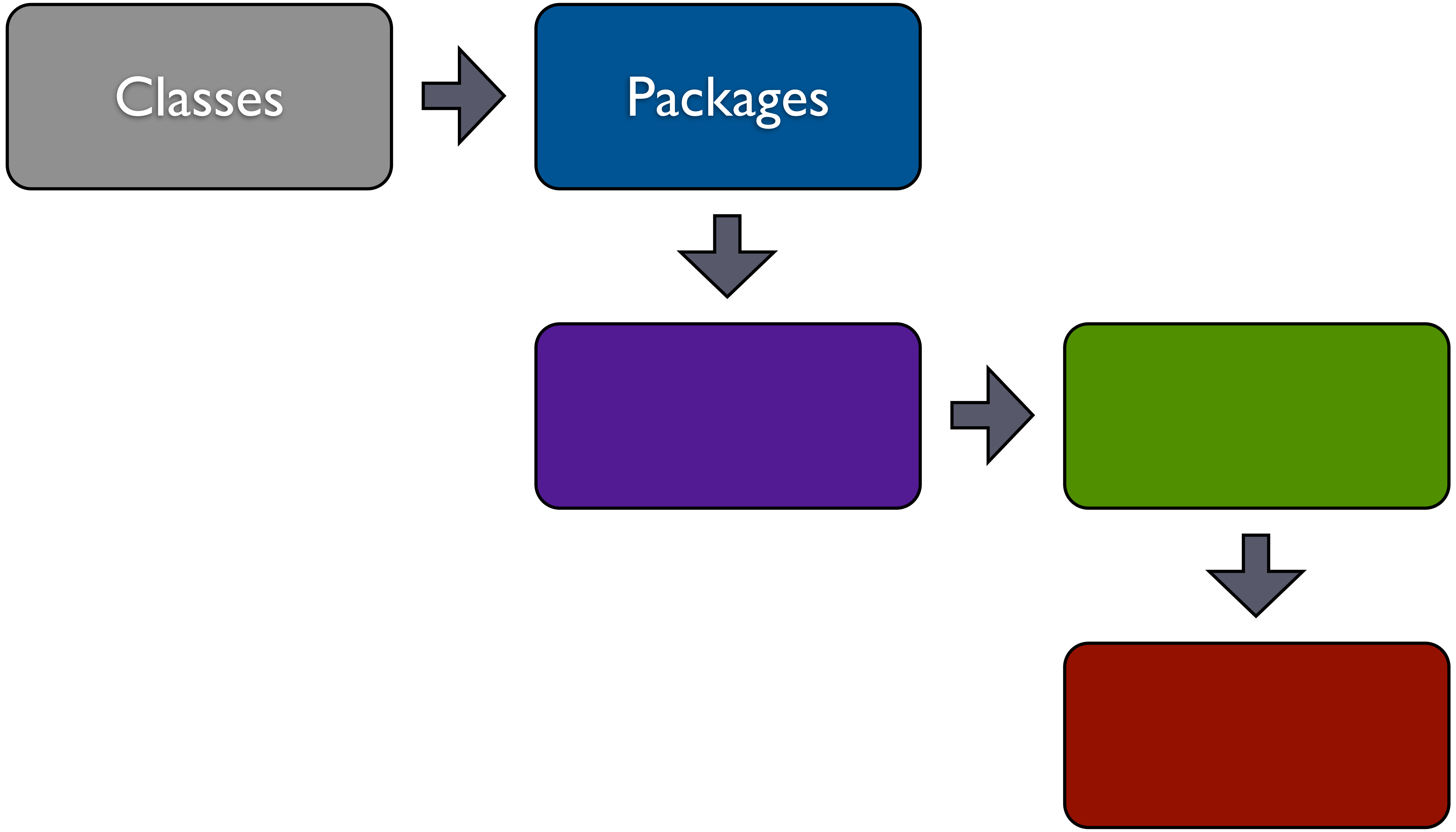
The Modularity Continuum

The Modularity Continuum

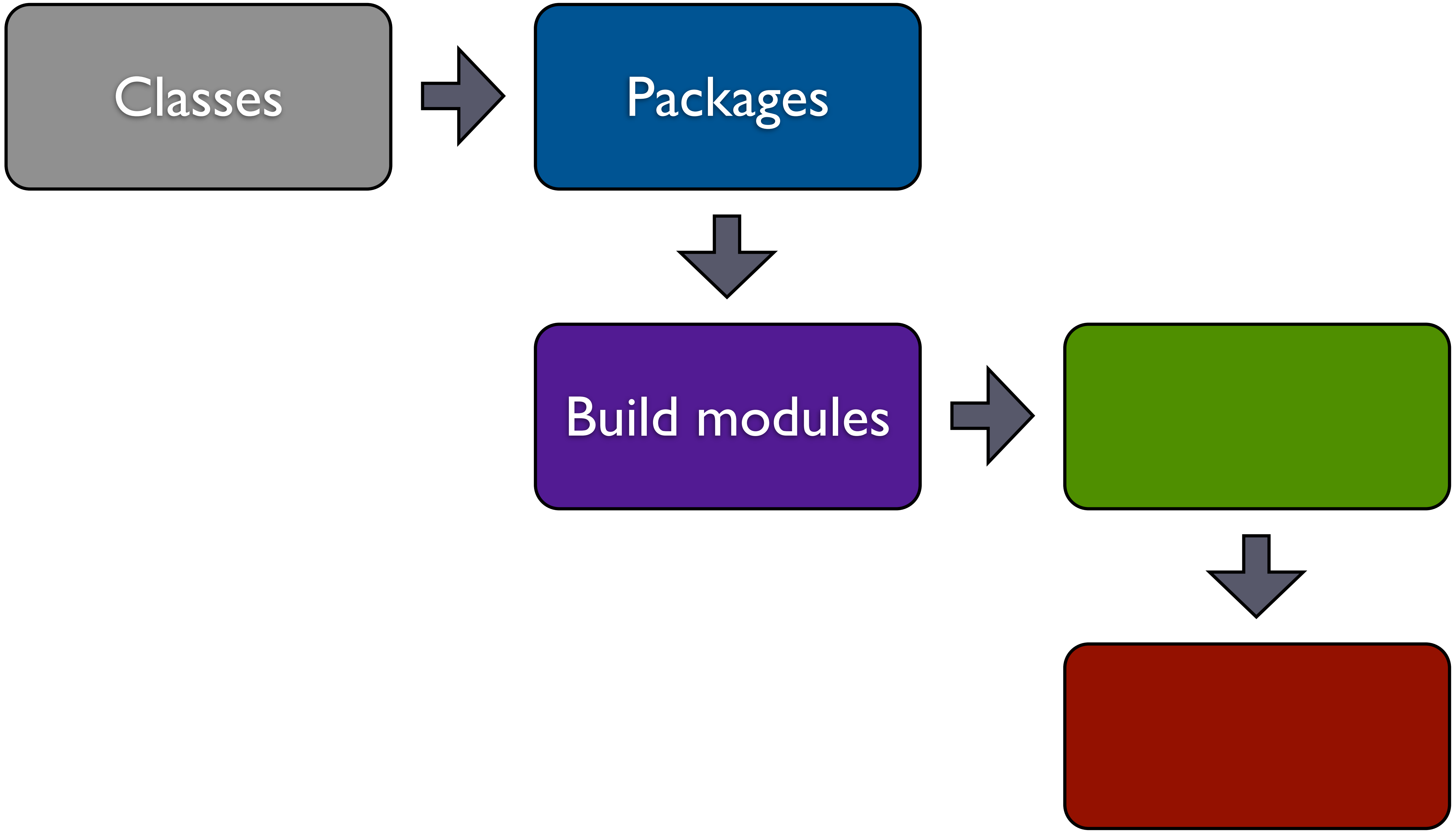


Single responsibility principle

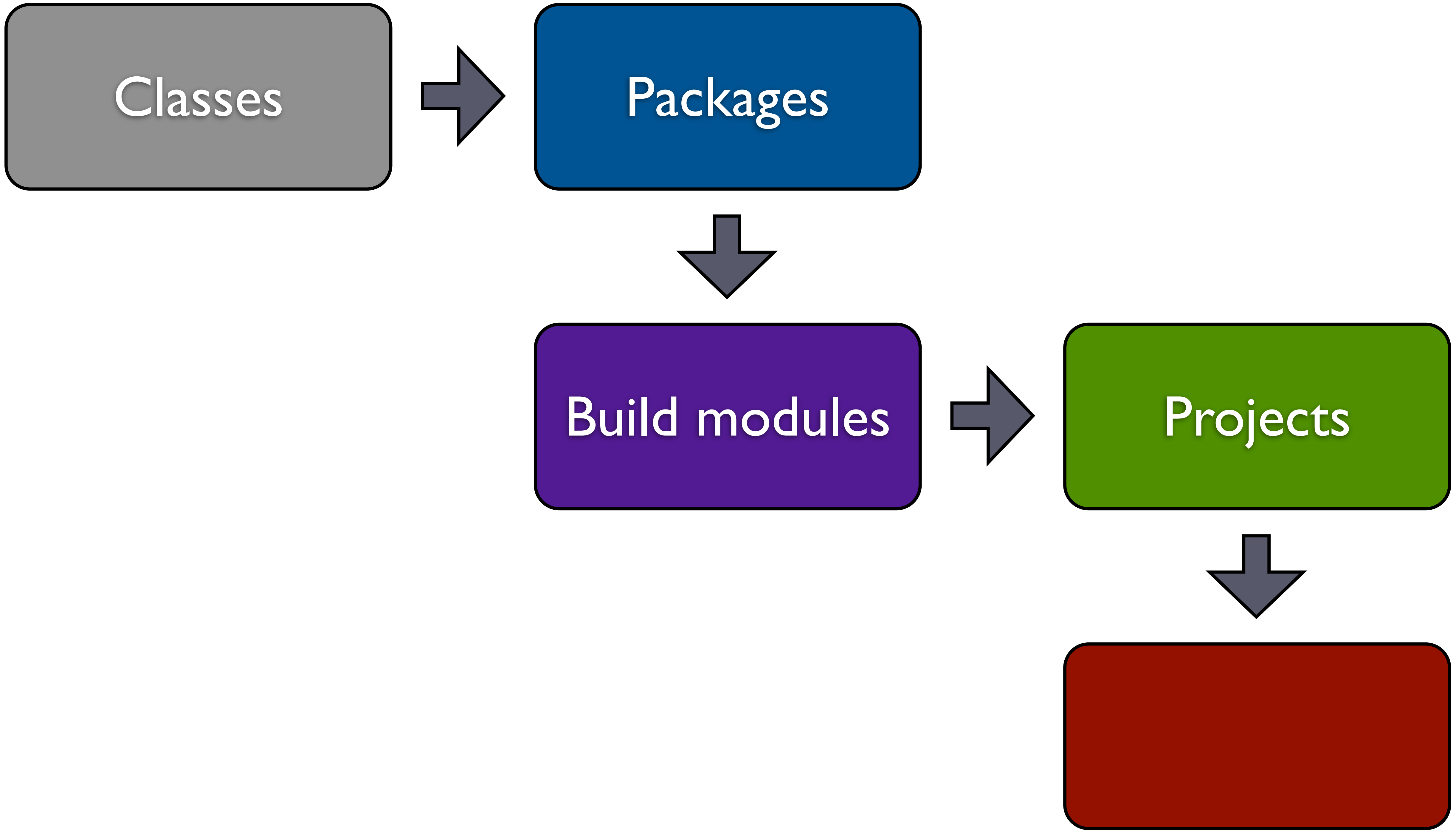
The Modularity Continuum



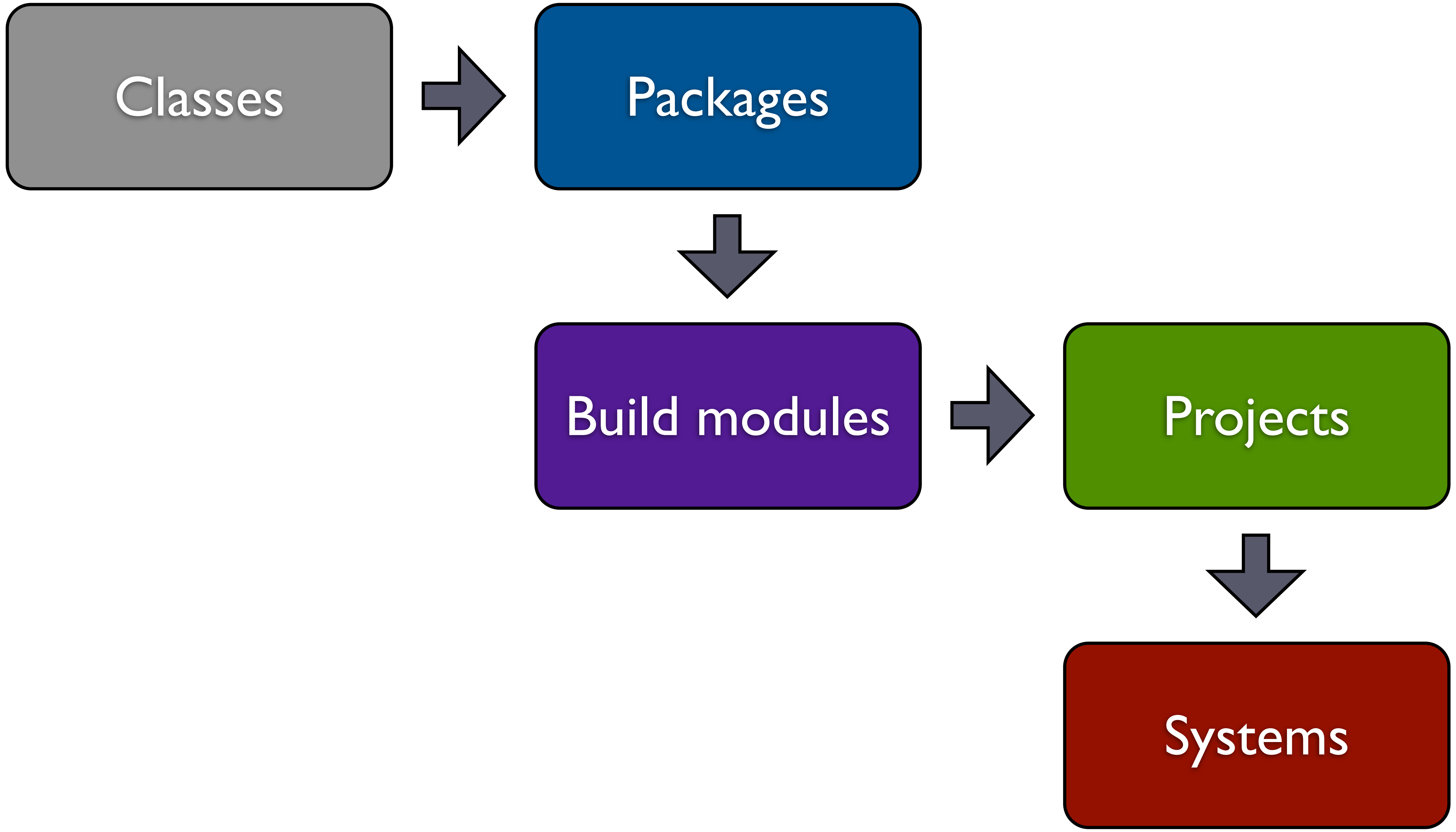
The Modularity Continuum



The Modularity Continuum



The Modularity Continuum



Ideal module system

Group code

- Comprehensible chunks of code
- Specific (single) responsibility
- Isolation
- Namespacing

Reusable

- Across one system
- Across multiple systems
- Industry standards

Abstraction

- Hide implementation details
- Decide what is accessible to who
- Not only what, but also **how** (wiring)

Wassily Kandinsky, Accent on Rose, 1926

Interfaces

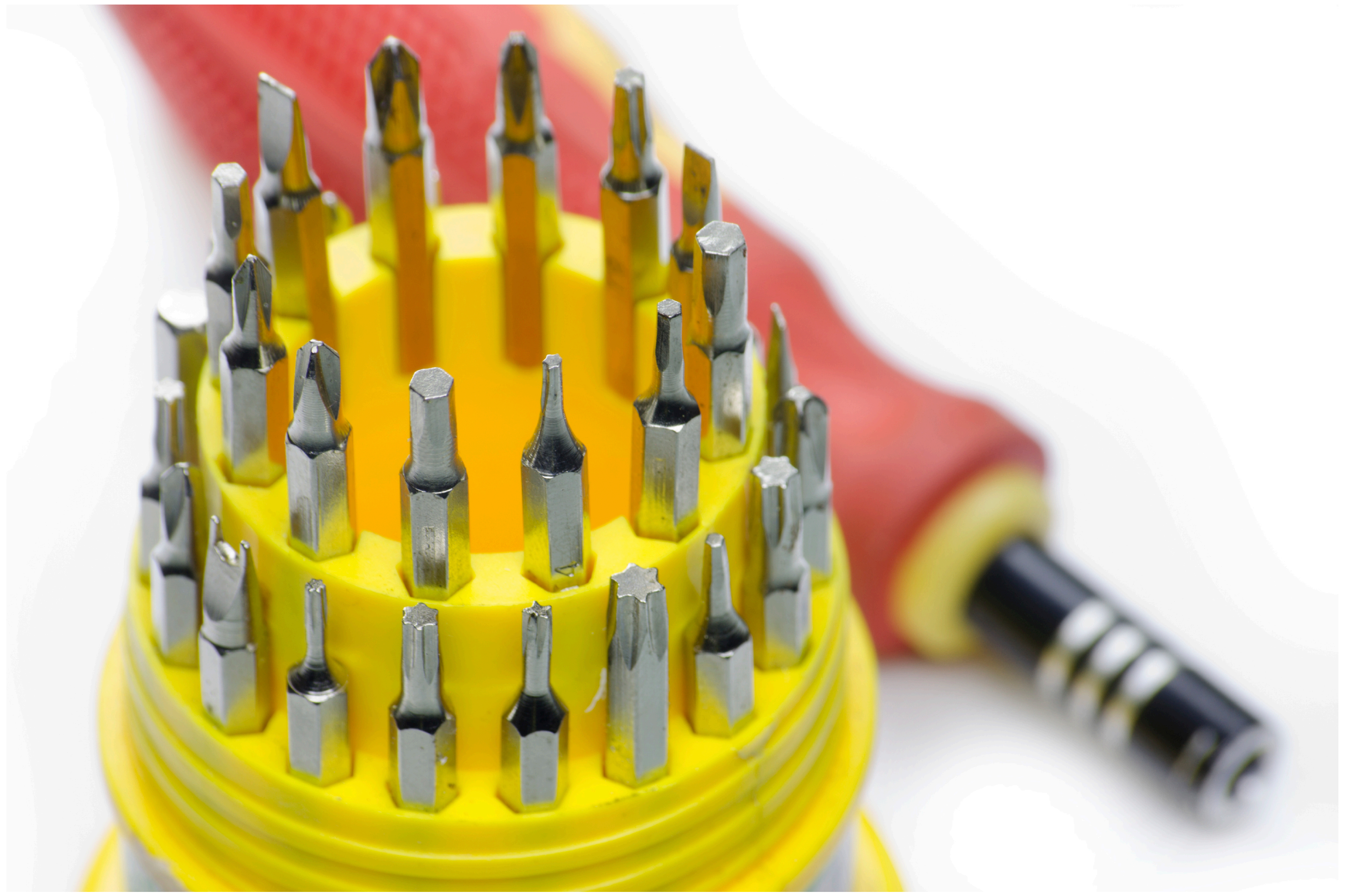
- Multiple implementations
- A way to define the interface & data structures

Composability

- Create big modules from smaller ones
- Hierarchical
- Scalable

Replaceable

- Swap implementations
- Run time/load time/build time



Meta

- Dependencies
- Versioning
- Specify & verify

Requirements

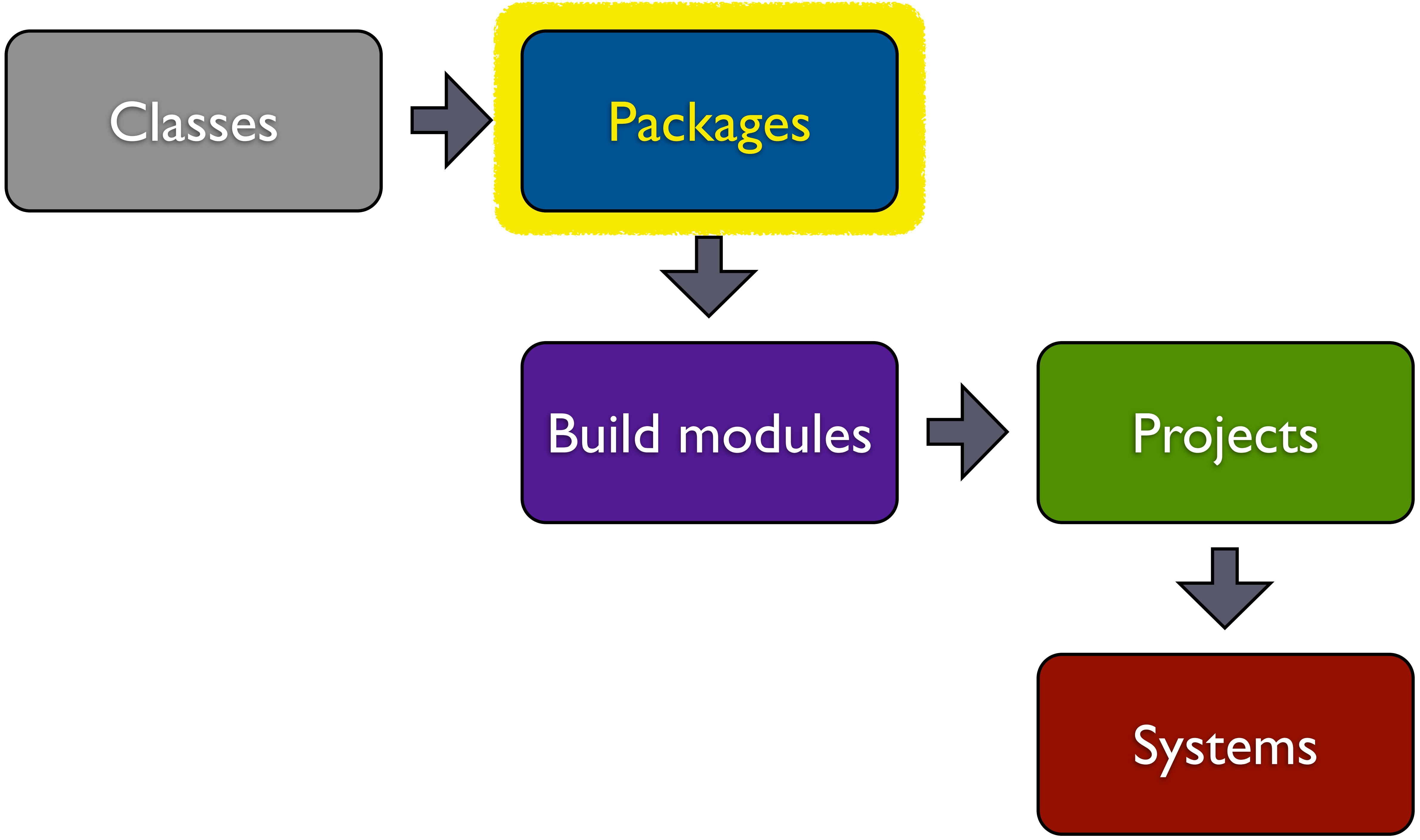
- Group code
- Reusable
- Abstraction
- Interfaces
- Composability
- Replaceable
- Meta

Requirements

- So ... what now?
- Let's create a new revolutionary language™!

Packages & Veripacks

Packages



Packages

- Namespacing
- Simple string identifiers
- Parent-child relations?
  - `com.company.myapp.finance`
  - `com.company.myapp.finance.invoicing`

All classes are equal?

- “By default” classes are public
- Which class is the main one?
- What’s the responsibility of the package?

One public class per package

- Make only one class public
- Other: `package-protected`
- Clearly visible:
  - what is the responsibility of the package
  - what's the main entry point

Growing the concept

- Support from JVM/Java ends here
- What if the functionality is big?
  - extract a sub-package
  - now the classes in the sub-package must be public

Enter Veripacks

- Specify which classes are exported from a **package hierarchy**
- Respect package parent-child relationships
- Allow exporting classes & child packages

- Using annotations
- **Veri**fy **Pack**age **S**pecifications

Enter Veripacks

```
package foo.bar.p1 {
  @Export
  class A { ... }

  class B { ... }
}

package foo.bar.p1.sub_p1 {
  class C { ... }
}
```

```
package foo.bar.p2 {
  class Test {
    // ok
    new A()

    // illegal
    new B()

    // illegal
    new C()
  }
}
```

Veripacks: exporting

- By default: export all
- Export a class
- Export a child package
- ... or any mix

- Transitive!

Running Veripacks

```
public void testPackageSpecifications() {
   VeripacksBuilder
      .build()
      .verify("foo.bar") // root package to check
      .throwIfNotOk()
}
```

Veripacks: importing

- Also transitive
  - to sub-packages

- Specify that a package needs to be imported
  - **@Import**
  - **@RequiresImport**

- Importing 3rd party libraries

3rd party library import example

```
// src/main/scala/org/veripacks/reader/package-info.java

@Import("org.objectweb")
package org.veripacks.reader;

import org.veripacks.Import;
```

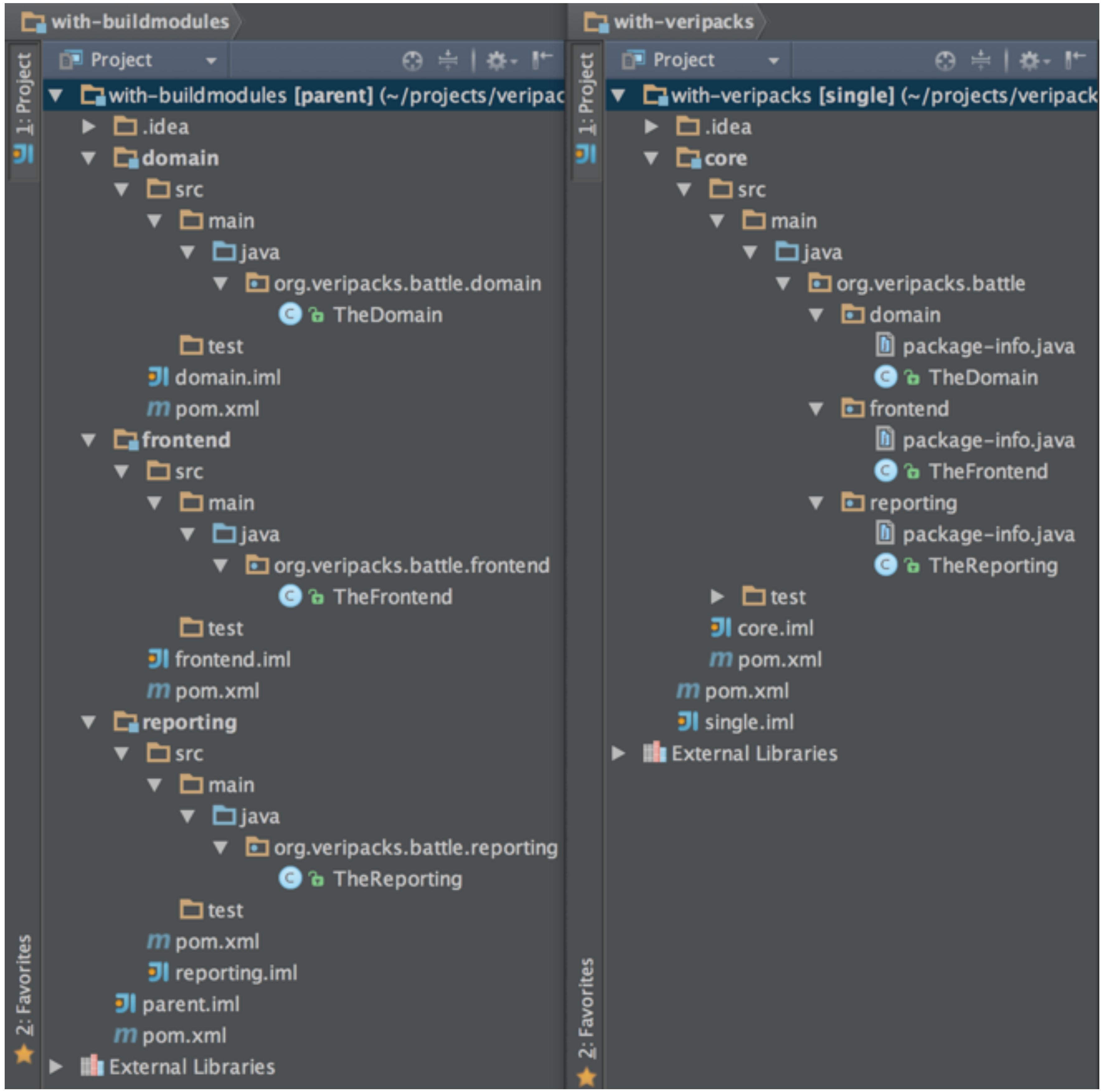
```
VeripacksBuilder
    .requireImportOf("org.objectweb")
    .build
    .verify("org.veripacks")
    .throwIfNotOk()
```

Replacing build modules?

- Why do we create build modules?
  - isolate parts of code
  - api/impl split
  - adding a 3rd party lib to a part of code
  - group code with similar functionality
  - statically check inter-module dependencies

Build modules are heavy

- Maven: elaborate xml
- Separate directory structure
- Hard to extract a common part
- Additional thing to name



```
with-buildmodules
Project
▼ with-buildmodules [parent] (~/projects/veripac
  ▶ .idea
  ▼ domain
    ▼ src
      ▼ main
        ▼ java
          ▼ org.veripacks.battle.domain
              TheDomain
        test
      domain.iml
      pom.xml
  ▼ frontend
    ▼ src
      ▼ main
        ▼ java
          ▼ org.veripacks.battle.frontend
              TheFrontend
        test
      frontend.iml
      pom.xml
  ▼ reporting
    ▼ src
      ▼ main
        ▼ java
          ▼ org.veripacks.battle.reporting
              TheReporting
        test
      pom.xml
      reporting.iml
    parent.iml
    pom.xml
▶ External Libraries
```

```
with-veripacks
Project
▼ with-veripacks [single] (~/projects/veripack
  ▶ .idea
  ▼ core
    ▼ src
      ▼ main
        ▼ java
          ▼ org.veripacks.battle
            ▼ domain
                package-info.java
                TheDomain
            ▼ frontend
                package-info.java
                TheFrontend
            ▼ reporting
                package-info.java
                TheReporting
      ▶ test
      core.iml
      pom.xml
    pom.xml
    single.iml
▶ External Libraries
```

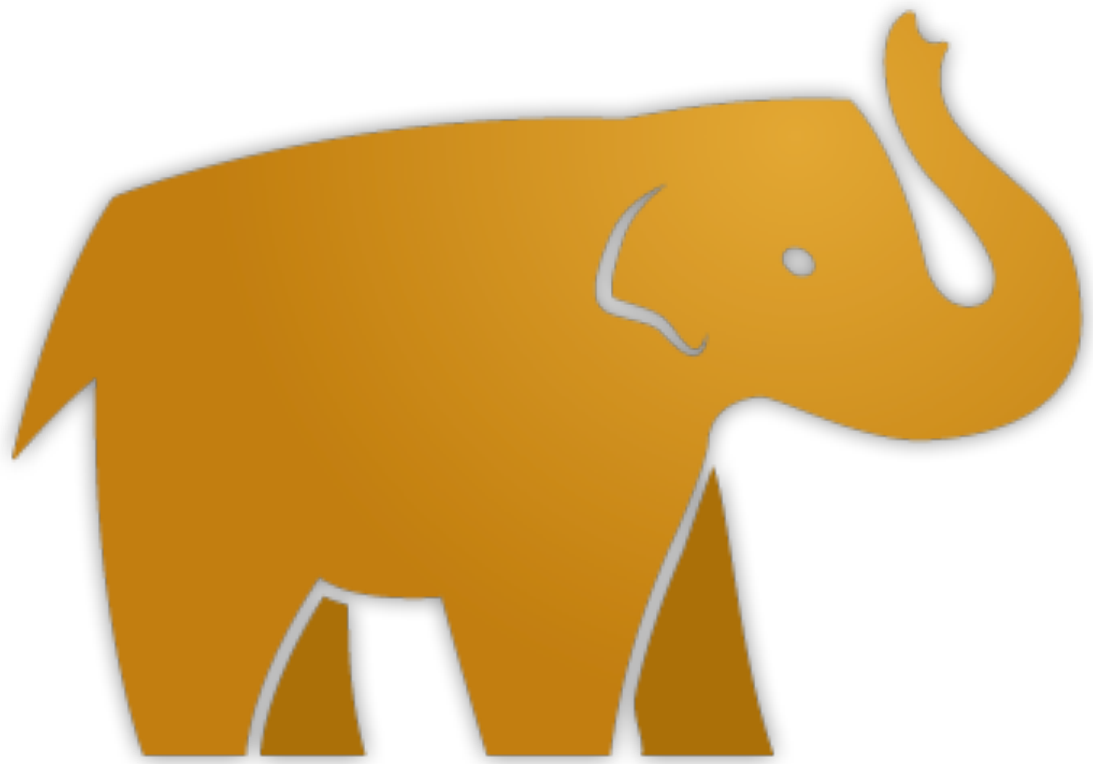
Other benefits

- No need to think when functionality is “big enough” to create a module
- Test code sharing
- Refactoring, easy to:
  - introduce a module
  - remove a module
  - rename a module

Packages+Veripacks as modules?

- *Group code*
- Partially *reusable*
- *Abstraction*: yes (`@Export`)
- *Interfaces*: no
- *Composability*: partial (transitivity)
- *Replaceable*: no
- *Meta*: dependencies yes (`@Import`), versioning no

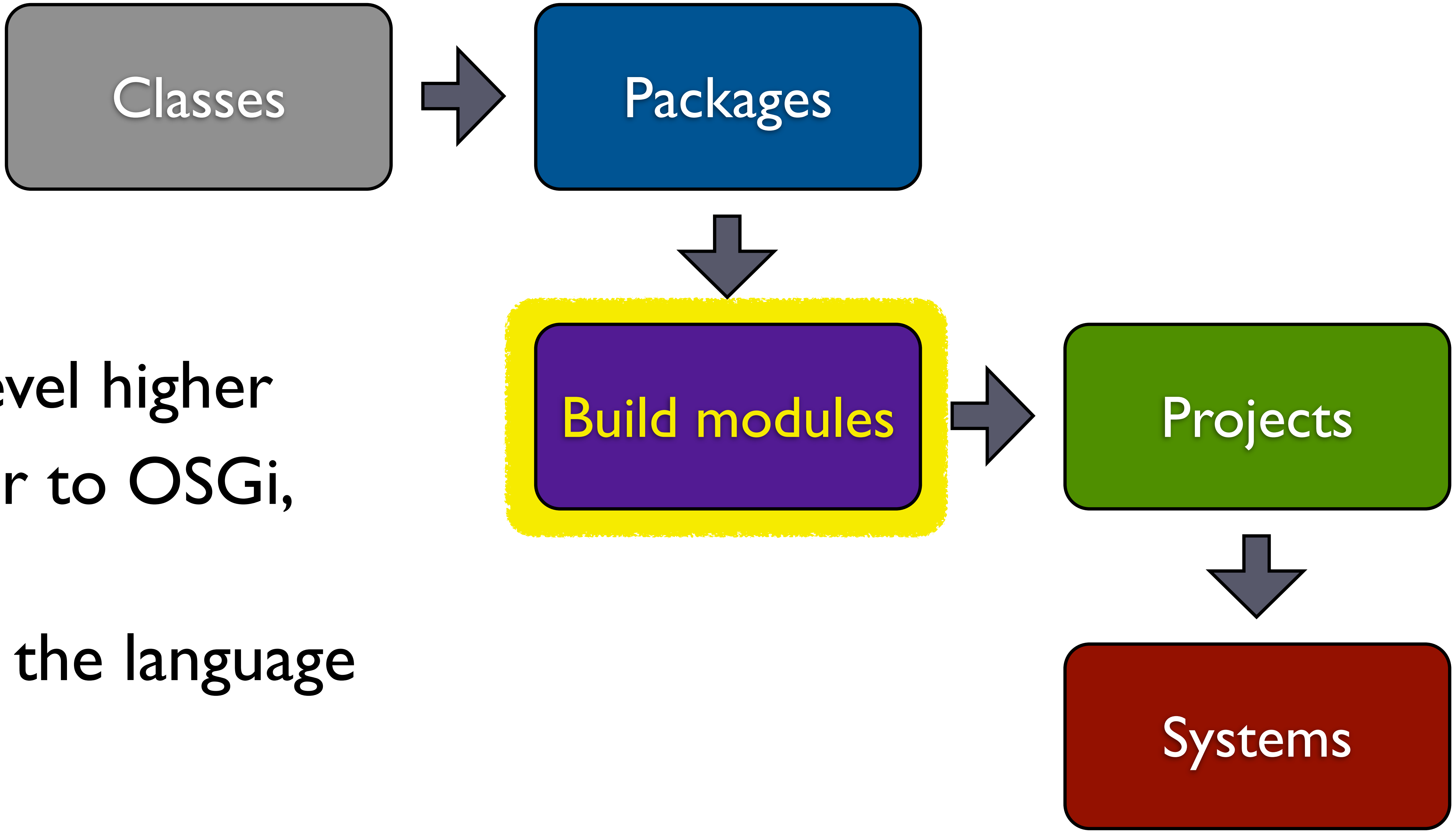
Ceylon



What is Ceylon?

- "A language for writing large programs in teams"
- Cross-platform (Java/JS)
- Statically typed (union, intersection types, ...)
- OO/FP mix
- Typesafe metaprogramming
- 1.0 with an IDE available: http://ceylon-lang.org/
- ... and **modularity**

Ceylon modules



- Moving a level higher
- A bit similar to OSGi, Jigsaw
- But part of the language

Sharing

- In Ceylon things can be `shared`, or not
- Things:
  - program elements (classes, class members)
  - packages

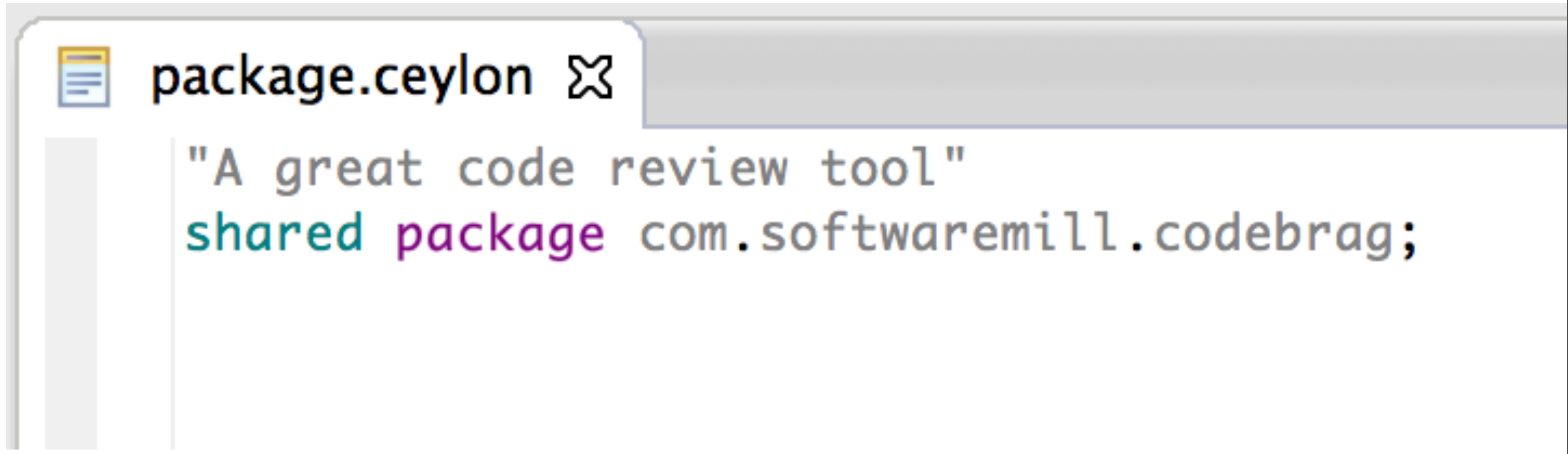
Modularity concepts in Ceylon

- 3 basic concepts:
  - modules
  - packages
  - classes

- Classes mostly similar as in Java
  - (from the modularity perspective)

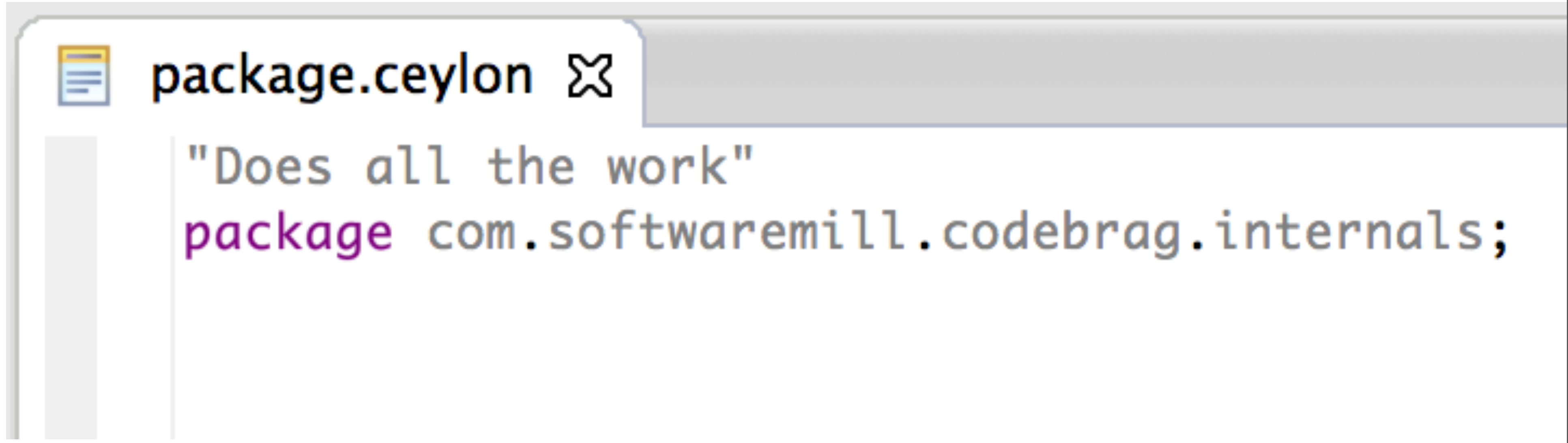
Packages in Ceylon

- Separate file with meta-data
- Annotations
  - sharing
  - comments



package.ceylon

```
"A great code review tool"
shared package com.softwaremill.codebrag;
```

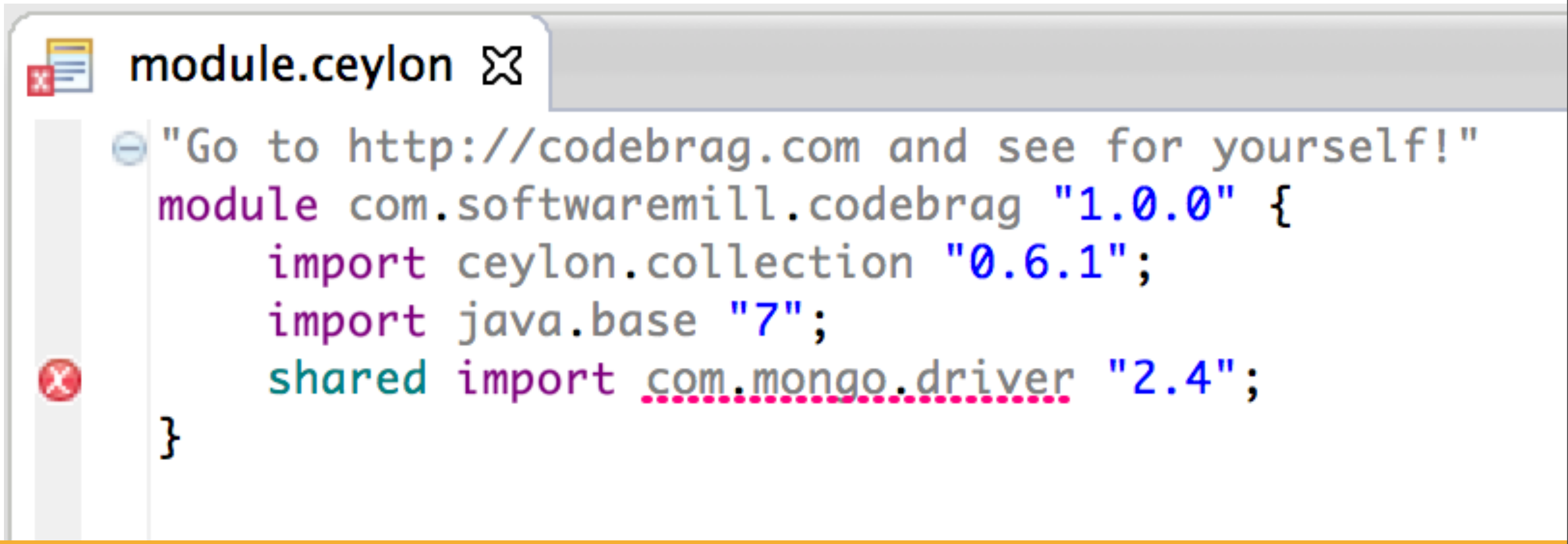


package.ceylon

```
"Does all the work"
package com.softwaremill.codebrag.internals;
```

Modules in Ceylon

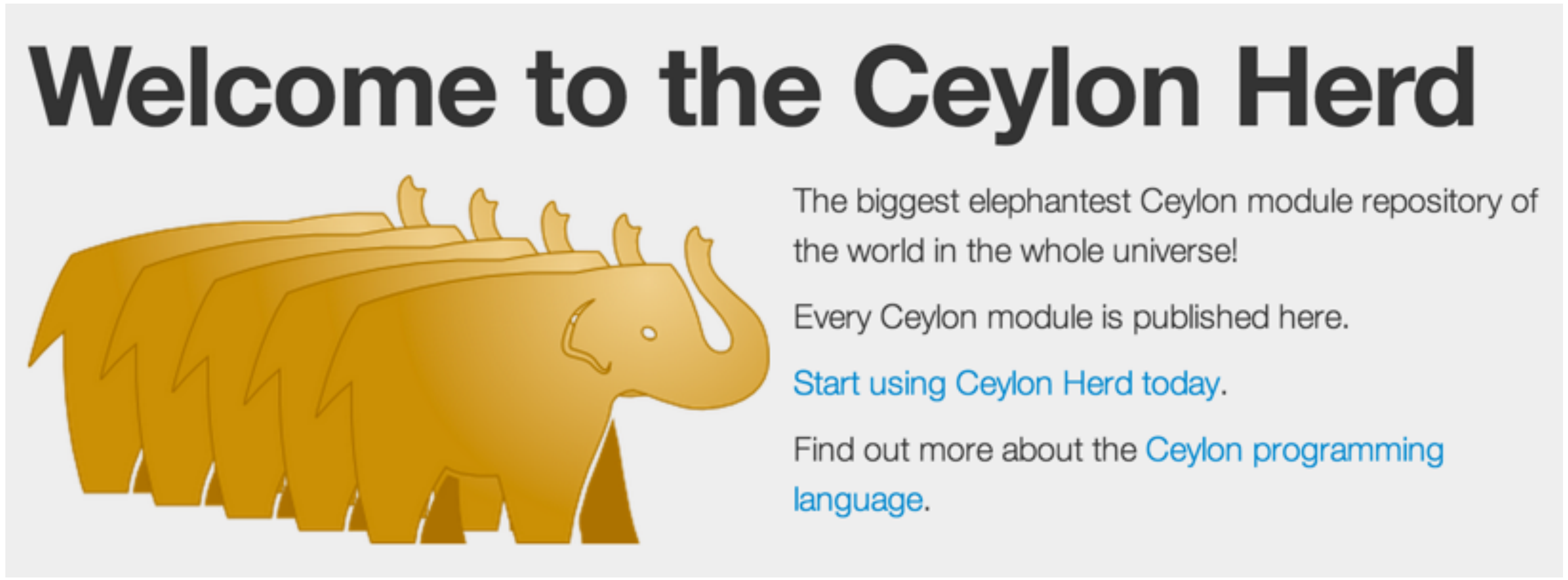
- Bundles a set of packages into a `.car` archive
- Package names: prefix of the module name
- Import other modules


module.ceylon

```
"Go to http://codebrag.com and see for yourself!"
module com.softwaremill.codebrag "1.0.0" {
    import ceylon.collection "0.6.1";
    import java.base "7";
    shared import com.mongo.driver "2.4";
}
```

There's more!

- Runnable modules
- Local/remote repositories used:
  - during the build
  - when running



Welcome to the Ceylon Herd

The biggest elephantest Ceylon module repository of the world in the whole universe!

Every Ceylon module is published here.

Start using Ceylon Herd today.

Find out more about the Ceylon programming language.

There's more!

- Run-time component
  - isolated class-loaders
  - based on JBoss Modules

Modules in Ceylon

- *Group code*
- *Reusable*: yes
- *Abstraction*: yes (**`shared`**)
- *Interfaces*: no
- *Composability*: partial
- Replaceable: partial
- *Meta*: yes (both for packages and modules)
- Run-time

Summing up

- Modules come in different flavors & sizes
- How many explicit module types do we want?
  - **scalability**
  - small, but specialized?
  - from very big to very small, general?
- Which requirements should which types meet?
- Challenge for the Next Big Language?

Links

- http://github.com/adamw/veripacks
- http://ceylon-lang.org
- http://warski.org

Thank you; Come & get a sticker

http://codebrag.com/devoxx/

Party!