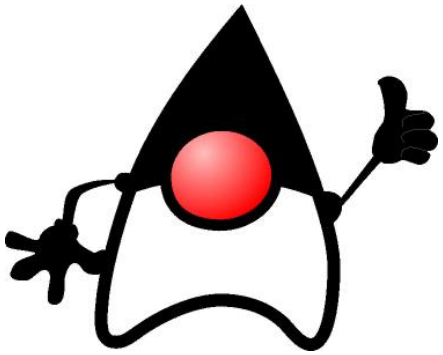
A Brief Introduction to Scala

S314144

Steven Reynolds

www.slreynolds.net

www.int.com



Dedicated to Dr. Augie Caponecchi

Scala

New-ish programming language for the JVM (and CLR)

Combines Functional programming with Object-Oriented

Static type system

Why Does Functional Programming Matter?

Why Does Functional Programming Matter?

Map-Reduce

Why Does Functional Programming Matter?

Map-Reduce

Google

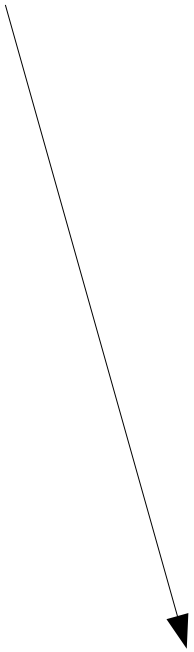
Why Does Functional Programming Matter?

Map-Reduce

Google

$150 Billion

Functional Programming



$150 Billion

Scala design goals

- Functional and object-oriented
- Practical
- Can call Scala ⇿ Java
- Powerful language – trust the programmer
- Powerful static type system that's easy to use

Functional Programming

Functional Programming

- Functions are first class citizens
- Pass functions as arguments and as return values
- Compact syntax to declare function literals
- Immutability: values don't change

Lambda Notation	Conventional Math Notation
λx.x + 2	f(x) = x + 2
(λx.x + 2) 3	f(3)

Alonzo Church

Functional Programming

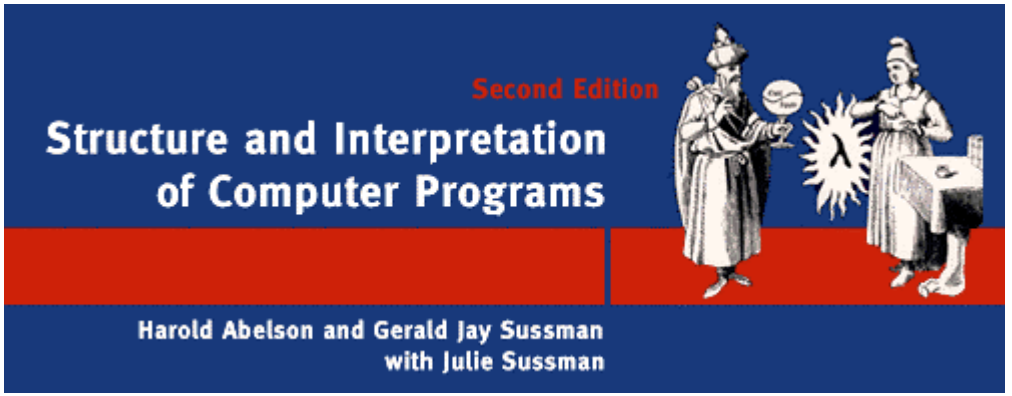
Advantages

- What was once true is always true - like math
- Reasoning and testing are simpler
- Nice for concurrency and distributed systems

Disadvantages

- Modular programming is sometimes harder
- Sometimes performance issues

These issues explored thoroughly in



http://mitpress.mit.edu/sicp/

Scala Approach

- Functions are first class citizens
- `val` – immutable

```
val x: Int = 7
```

- `var` – mutable

```
var y: String = “foo”
```

- Collections have immutable and mutable versions
- Steer to immutable, but make mutable equally easy

Scala Type System

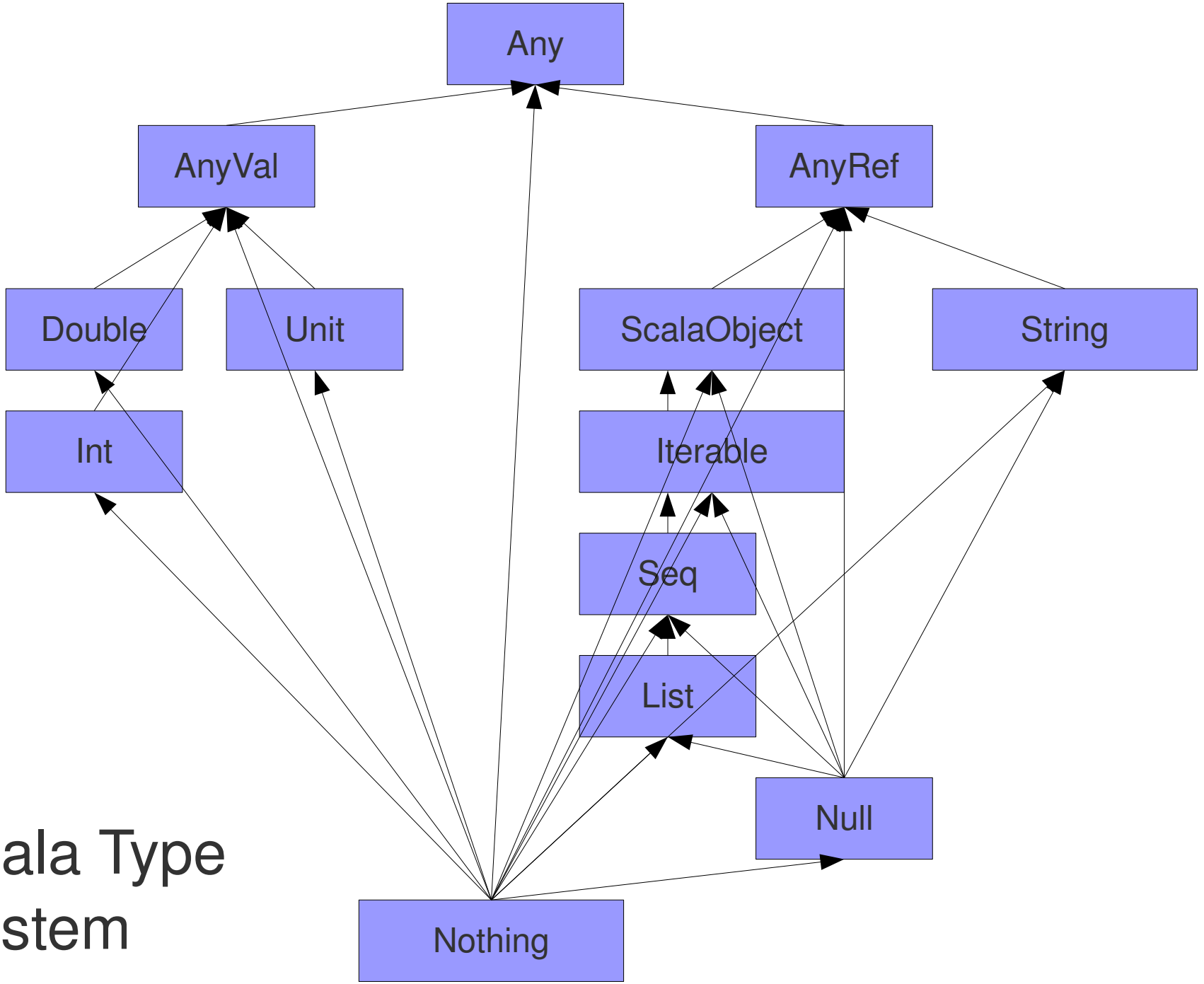
Scala Type system

- Every statement has a value
- Type inferencing

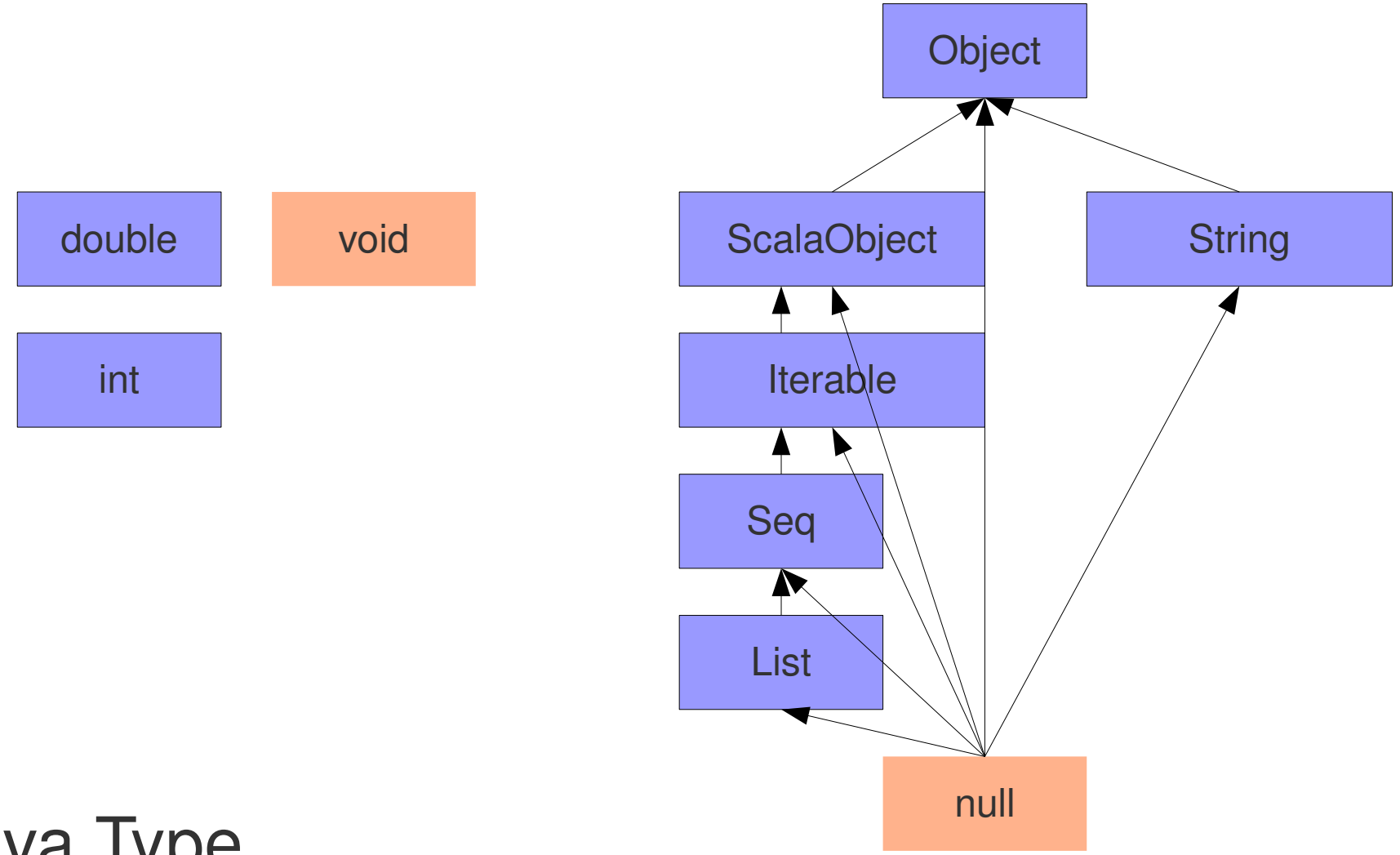
```
val s = “Some String”
```

- Generics with *all* types supported

```
var li = List[Int]()
```



Scala Type System



Java Type System

a lattice is

a partially ordered set in which any two elements have

- a unique least upper bound (join) and
- a unique greatest lower bound (meet).

Source: Wikipedia

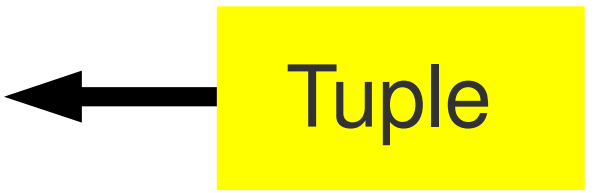
.... Great

What can I do with it?

```
def error(message: String): Nothing =
  throw new RuntimeException(message)
```

You can use error anywhere

```
def roots(a: Double, b: Double, c:
Double):(Double,Double) {
  dscr = b*b – 4*a*c
  if (dscr < 0 )
    error(“no complex math yet”)
  else
    ((-b – sqrt(dscr))/2*a,
     (-b + sqrt(dscr))/2*a)
}
```



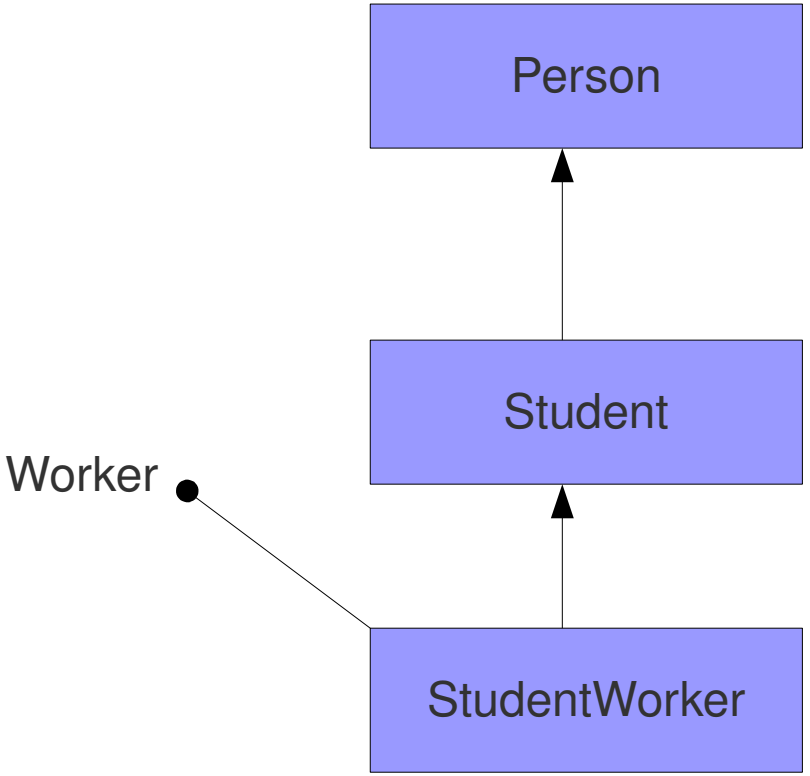
Inheritance in Scala

Inheritance

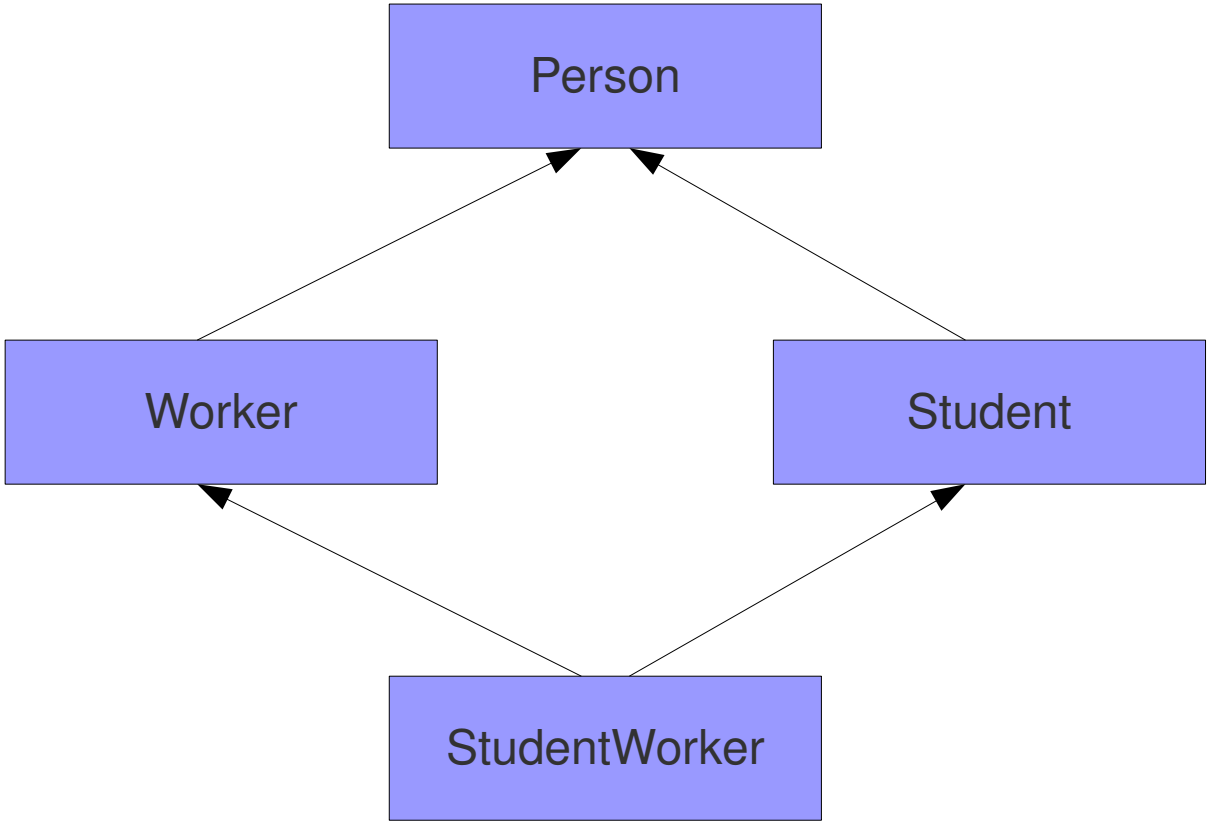
- C++ has multiple inheritance
- Java has single inheritance
- Scala has single inheritance with mixins (traits)

Origin of mixins?

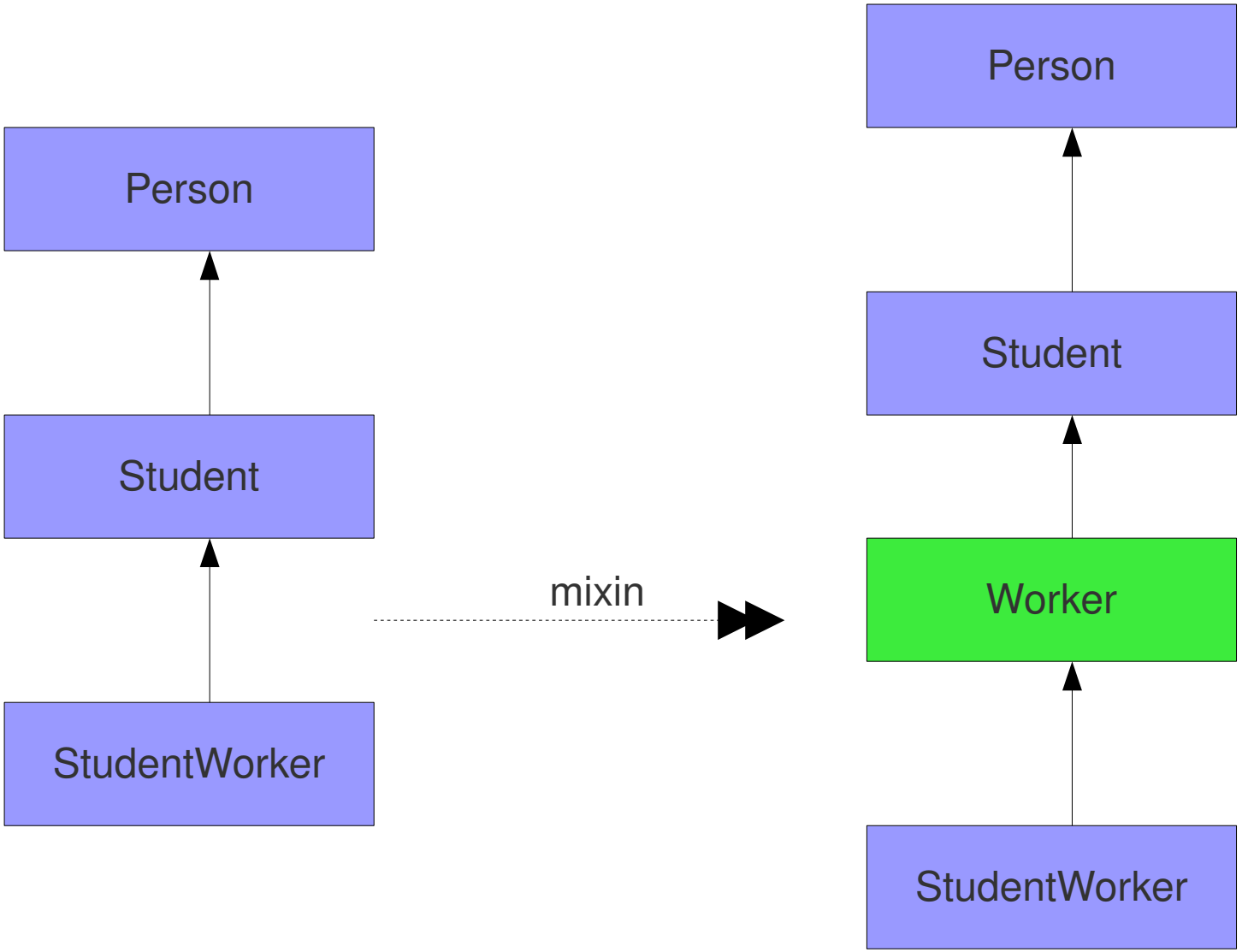
- Lisp flavors
- Steve's ice cream



Single Inheritance (Java)



Multiple Inheritance
(C++)


Person
Student
StudentWorker
mixin
Person
Student
Worker
StudentWorker

Mixin (Scala)

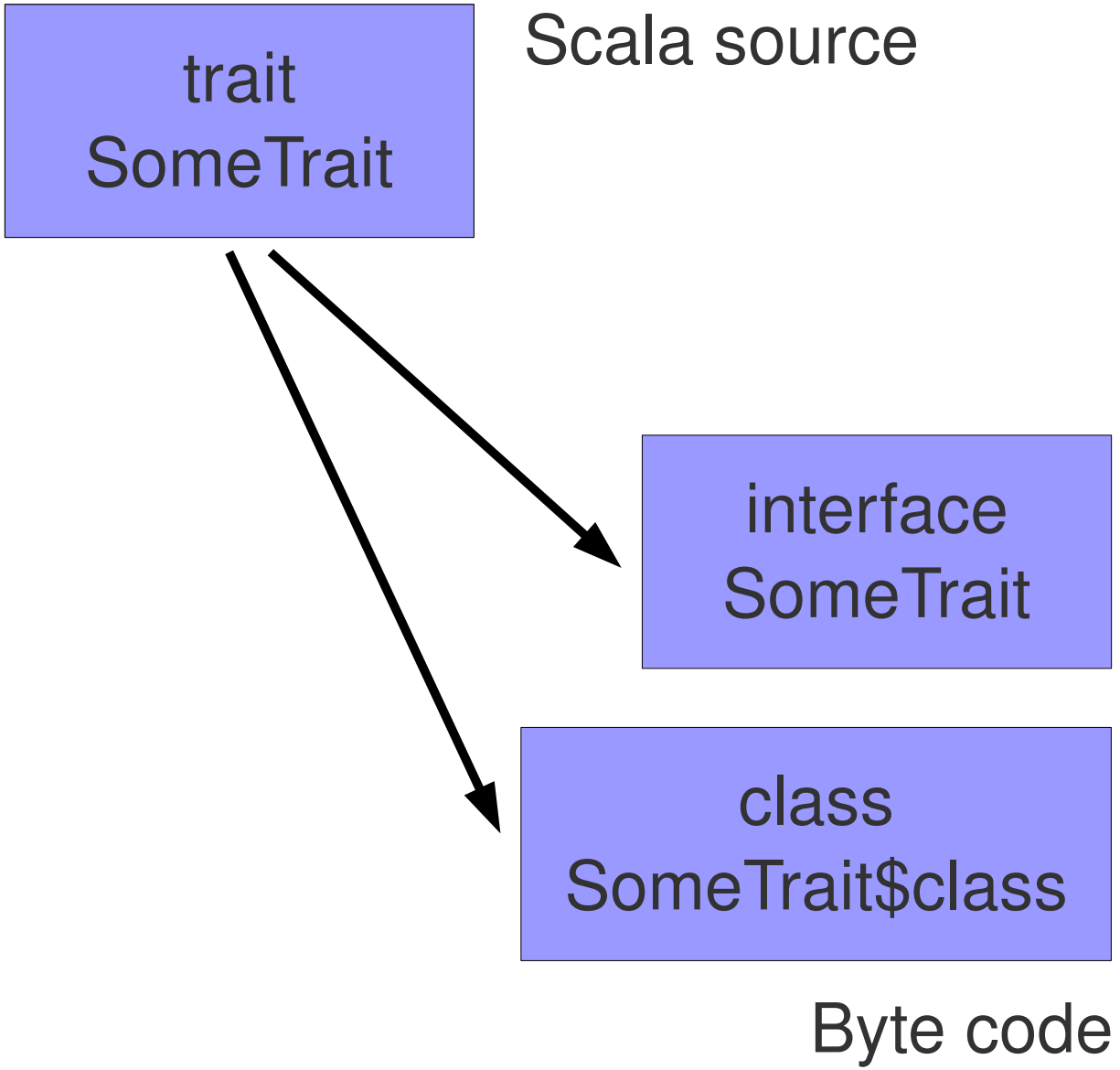
Mixins

- defines a delta relative to the hierarchy where the mixin is inserted
- does define behavior, and state
- cannot be understood until you know where it's inserted

Class

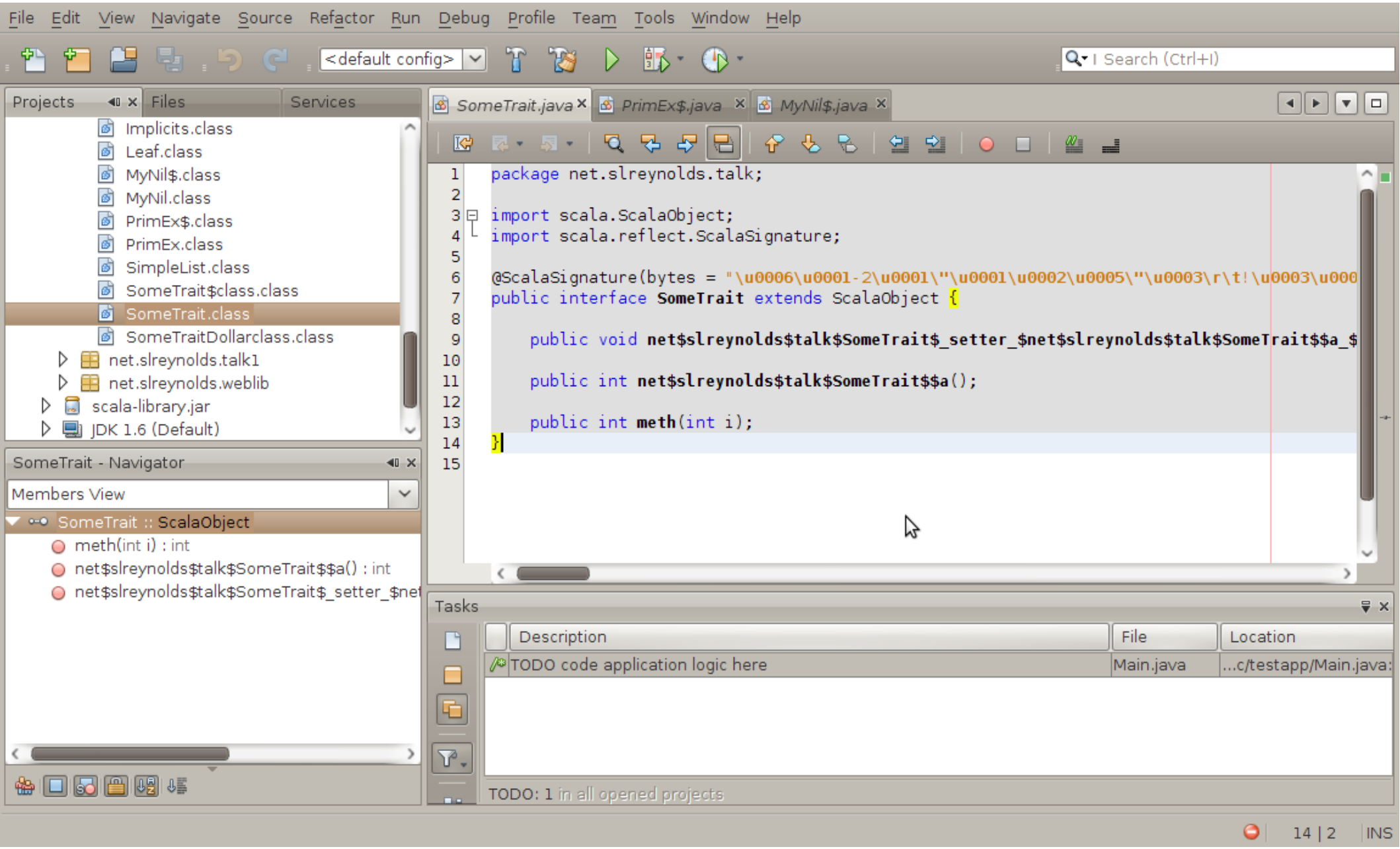
- is completely understood when you write the class
- is absolute

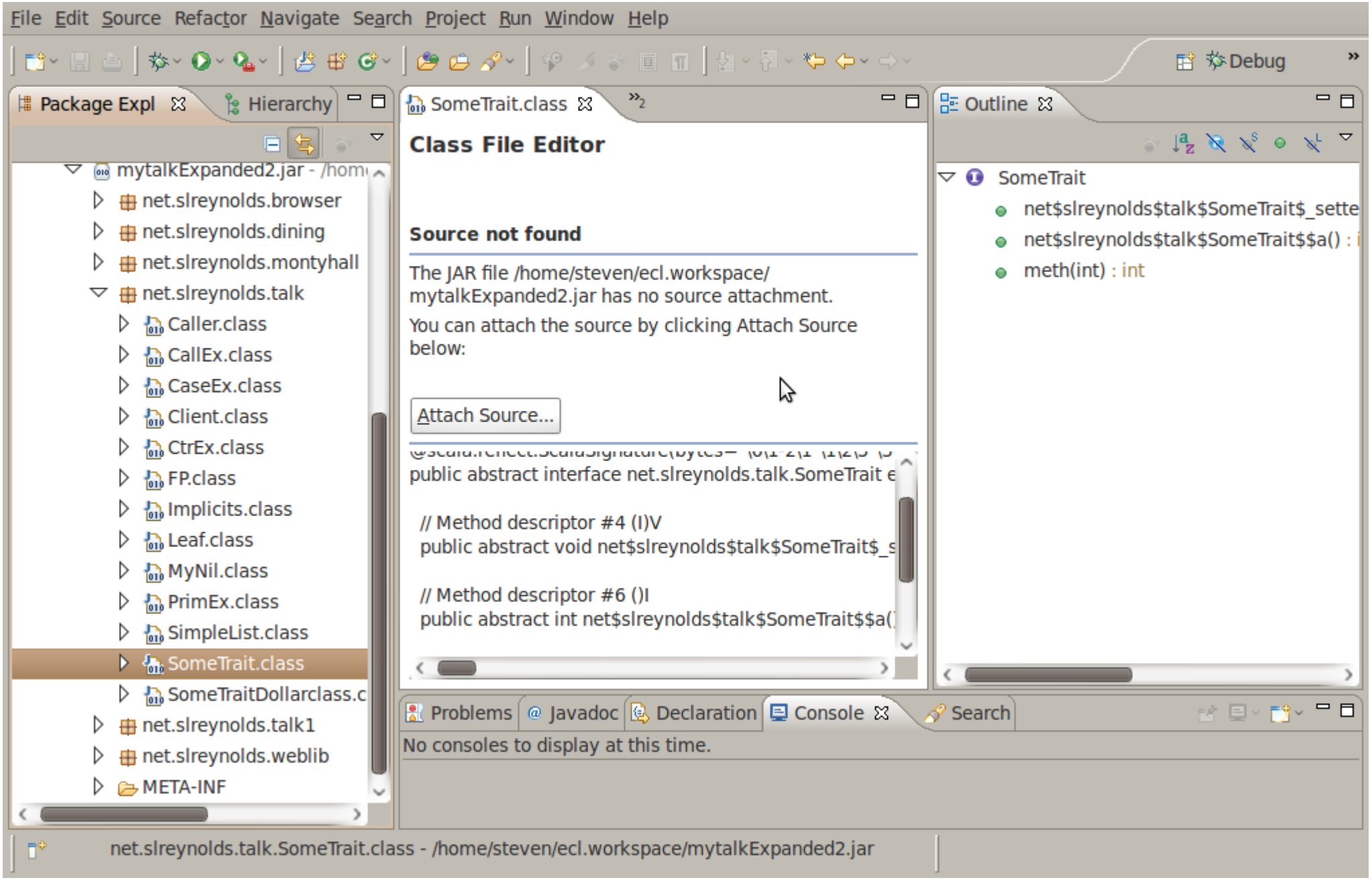
Under The Hood...


trait
SomeTrait
Scala source
interface
SomeTrait
class
SomeTrait$class
Byte code

If you want to see details...

1. Compile scala to class files and make a jar
2. Create a plain Java project, add scala jar file to this project
   - Source API view – NetBeans 6.9
   - Bytecode view – Eclipse


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Search (Ctrl+I)
Projects
Files
Services
Implicits.class
Leaf.class
MyNil$.class
MyNil.class
PrimEx$.class
PrimEx.class
SimpleList.class
SomeTrait$class.class
SomeTrait.class
SomeTraitDollarclass.class
net.slreynolds.talk1
net.slreynolds.weblib
scala-library.jar
JDK 1.6 (Default)
SomeTrait - Navigator
Members View
SomeTrait :: ScalaObject
meth(int i) : int
net$slreynolds$talk$SomeTrait$$a() : int
net$slreynolds$talk$SomeTrait$_setter_$net
SomeTrait.java
PrimEx$.java
MyNil$.java
1 package net.slreynolds.talk;
2
3 import scala.ScalaObject;
4 import scala.reflect.ScalaSignature;
5
6 @ScalaSignature(bytes = "\u0006\u0001-2\u0001\"\u0001\u0002\u0005\"\u0003\r\t!\u0003\u000
7 public interface SomeTrait extends ScalaObject {
8
9 public void net$slreynolds$talk$SomeTrait$_setter_netslreynolds$talk$SomeTrait$$a_$
10
11 public int net$slreynolds$talk$SomeTrait$$a();
12
13 public int meth(int i);
14 }
15
Tasks
Description
File
Location
TODO code application logic here
Main.java
...c/testapp/Main.java:
TODO: 1 in all opened projects
14 | 2
INS


File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Package Expl
Hierarchy
mytalkExpanded2.jar - /hom
net.slreynolds.browser
net.slreynolds.dining
net.slreynolds.montyhall
net.slreynolds.talk
Caller.class
CallEx.class
CaseEx.class
Client.class
CtrEx.class
FP.class
Implicits.class
Leaf.class
MyNil.class
PrimEx.class
SimpleList.class
SomeTrait.class
SomeTraitDollarclass.c
net.slreynolds.talk1
net.slreynolds.weblib
META-INF
SomeTrait.class
Class File Editor
Source not found
The JAR file /home/steven/ecl.workspace/ mytalkExpanded2.jar has no source attachment.
You can attach the source by clicking Attach Source below:
Attach Source...
public abstract interface net.slreynolds.talk.SomeTrait e
// Method descriptor #4 (I)V
public abstract void net$slreynolds$talk$SomeTrait$_s
// Method descriptor #6 ()I
public abstract int net$slreynolds$talk$SomeTrait$$a(
Outline
SomeTrait
net$slreynolds$talk$SomeTrait$_sette
net$slreynolds$talk$SomeTrait$$a() : i
meth(int) : int
Problems
Javadoc
Declaration
Console
Search
No consoles to display at this time.
net.slreynolds.talk.SomeTrait.class - /home/steven/ecl.workspace/mytalkExpanded2.jar

Aside – Compiler

- The scala compiler is an active participant – type inferencing, and code generation
- Compiler generates code
- Use scalac -Xprint:typer to see this output

Evaluation Order

Delayed Evaluation

By-name parameters

```
def byNameAssert(predicate: => Boolean) =
  if (assertionsEnabled && !predicate)
    throw new AssertionError

byNameAssert(5 > 3)
```

Lazy Initialization

- Variables can be initialized lazily

```
lazy val x = { println(“initializing x”);
“done” }
```

Are initialized

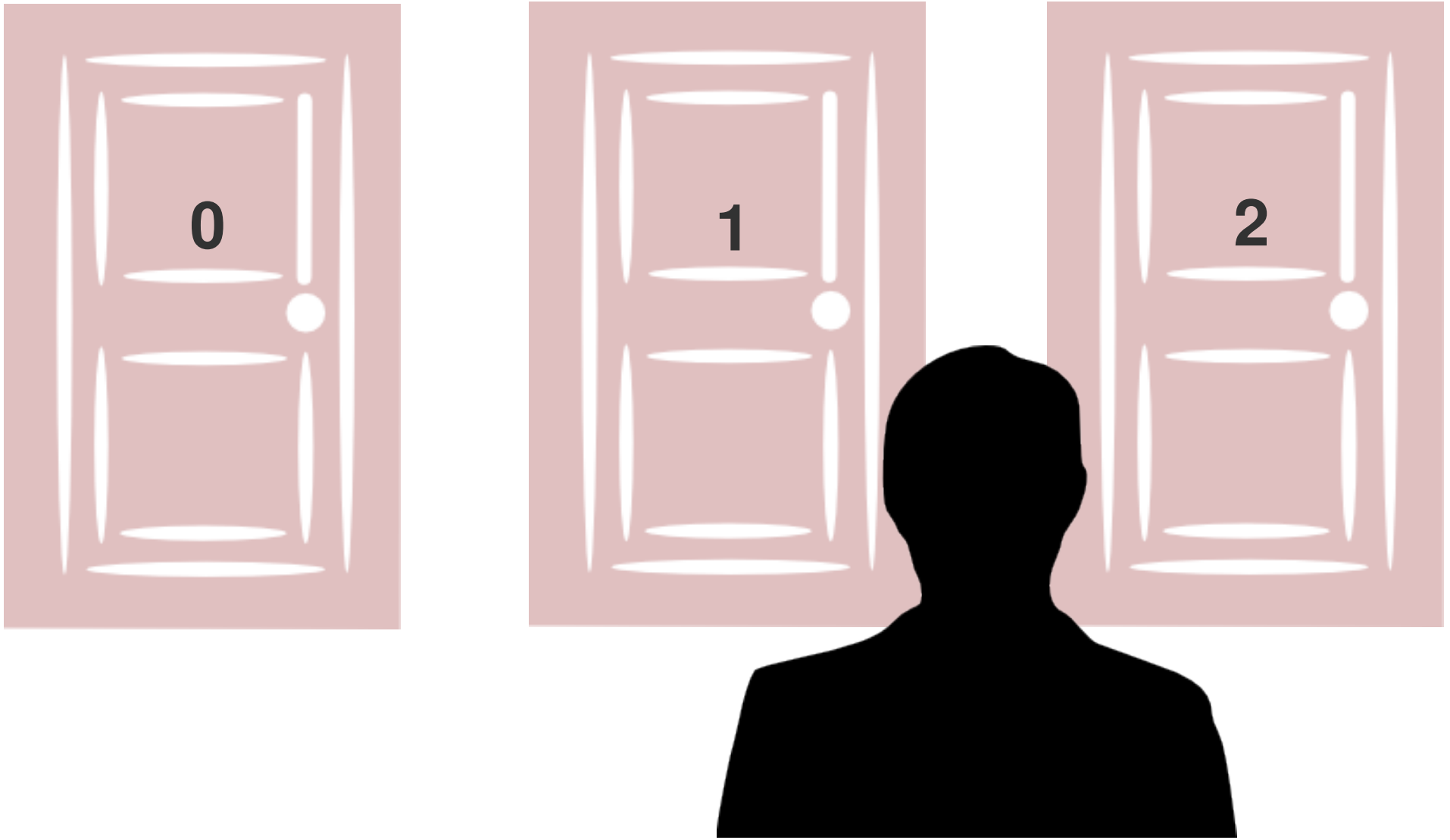
- at most once
- when they are needed

Matching

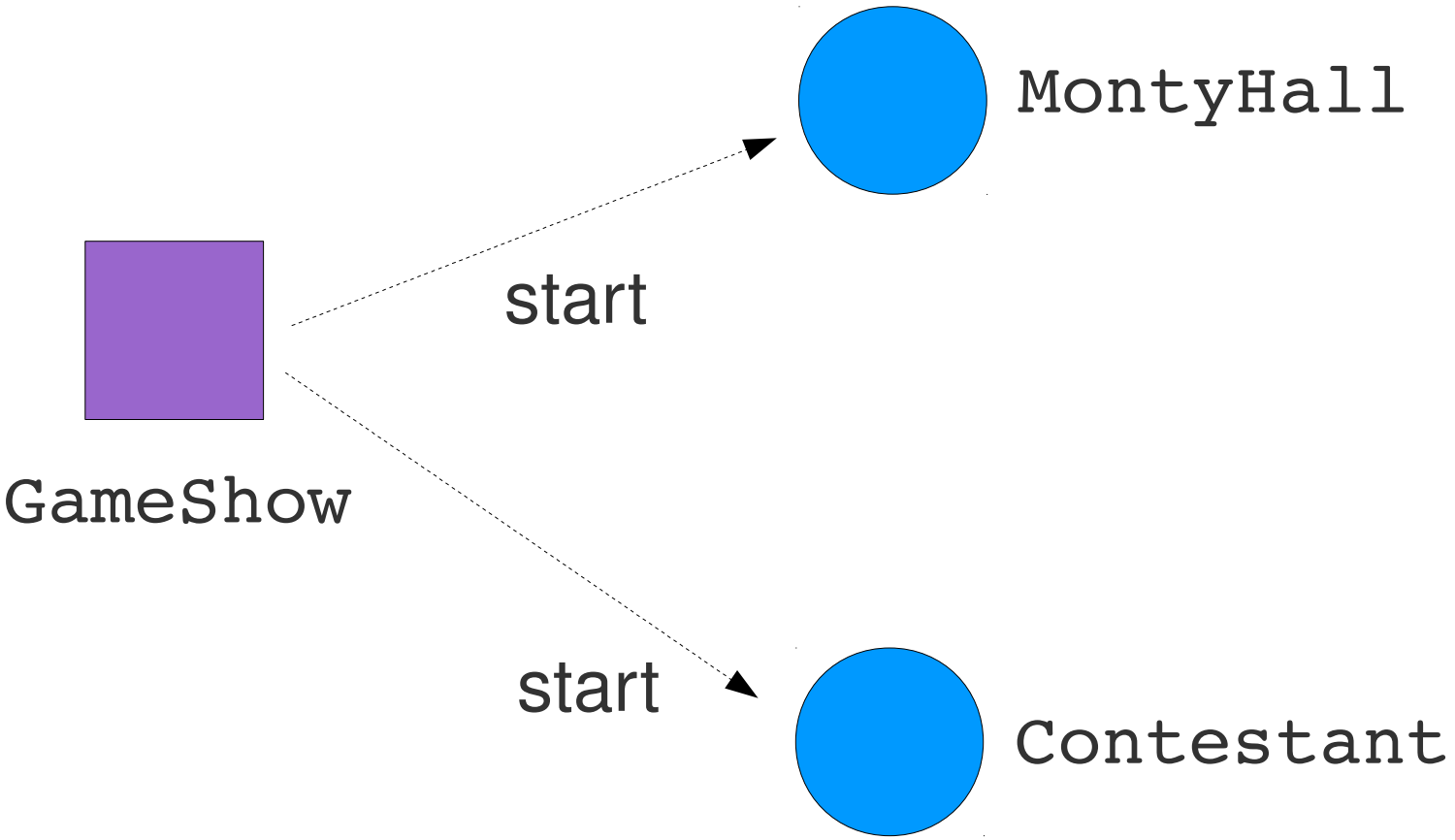
Example: `DogsEx.scala`

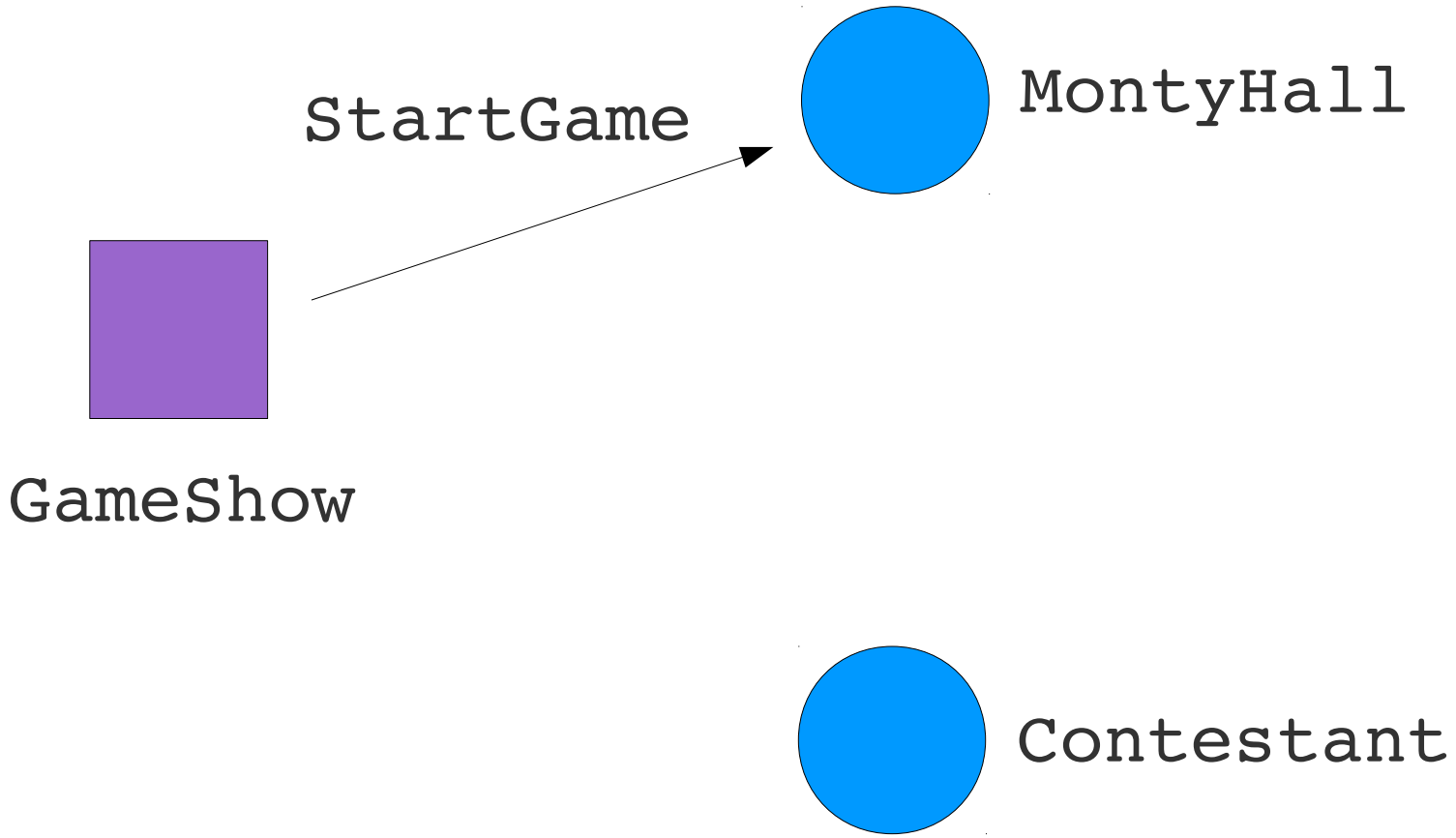
Actors

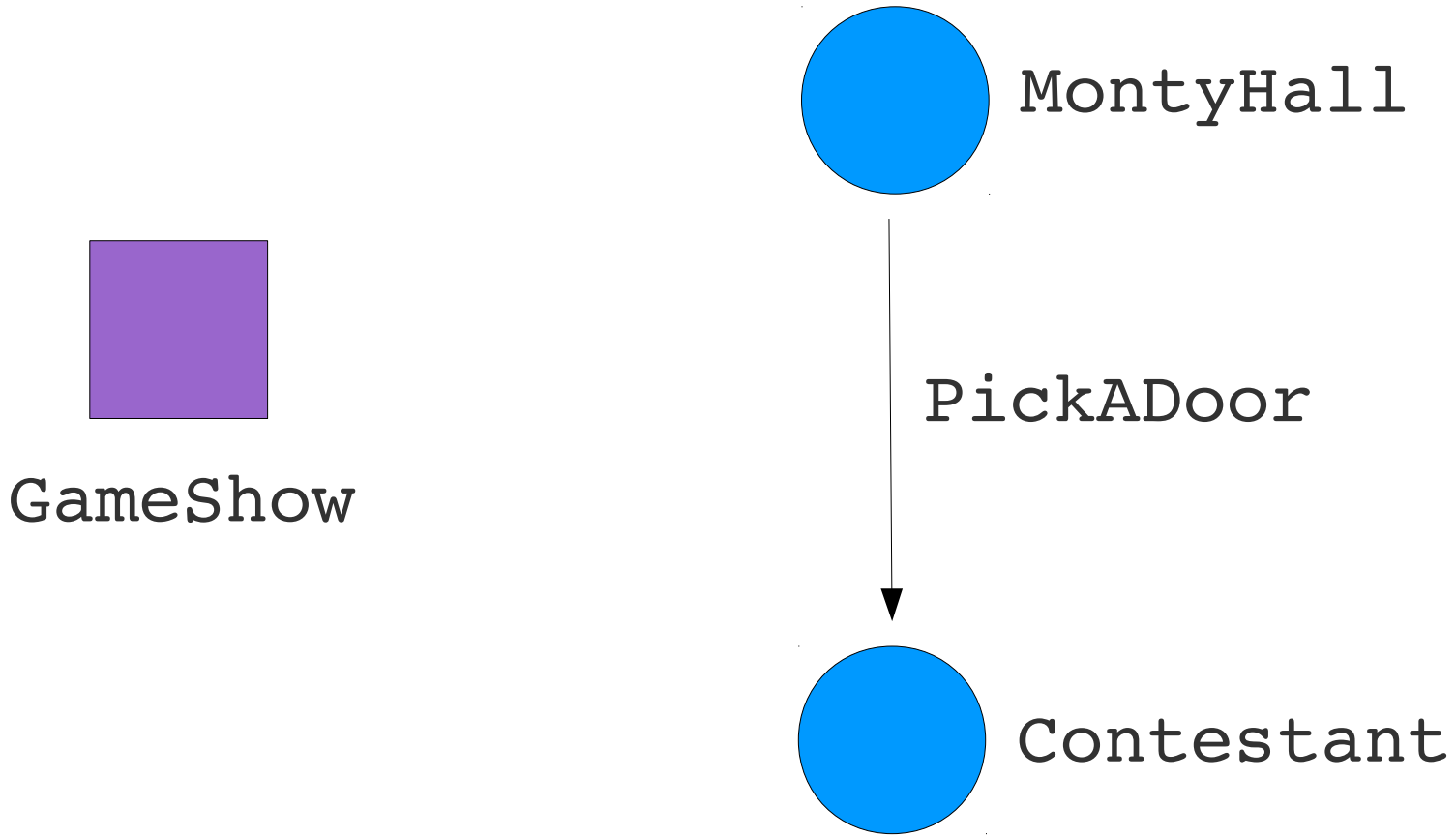
Let's Make a Deal

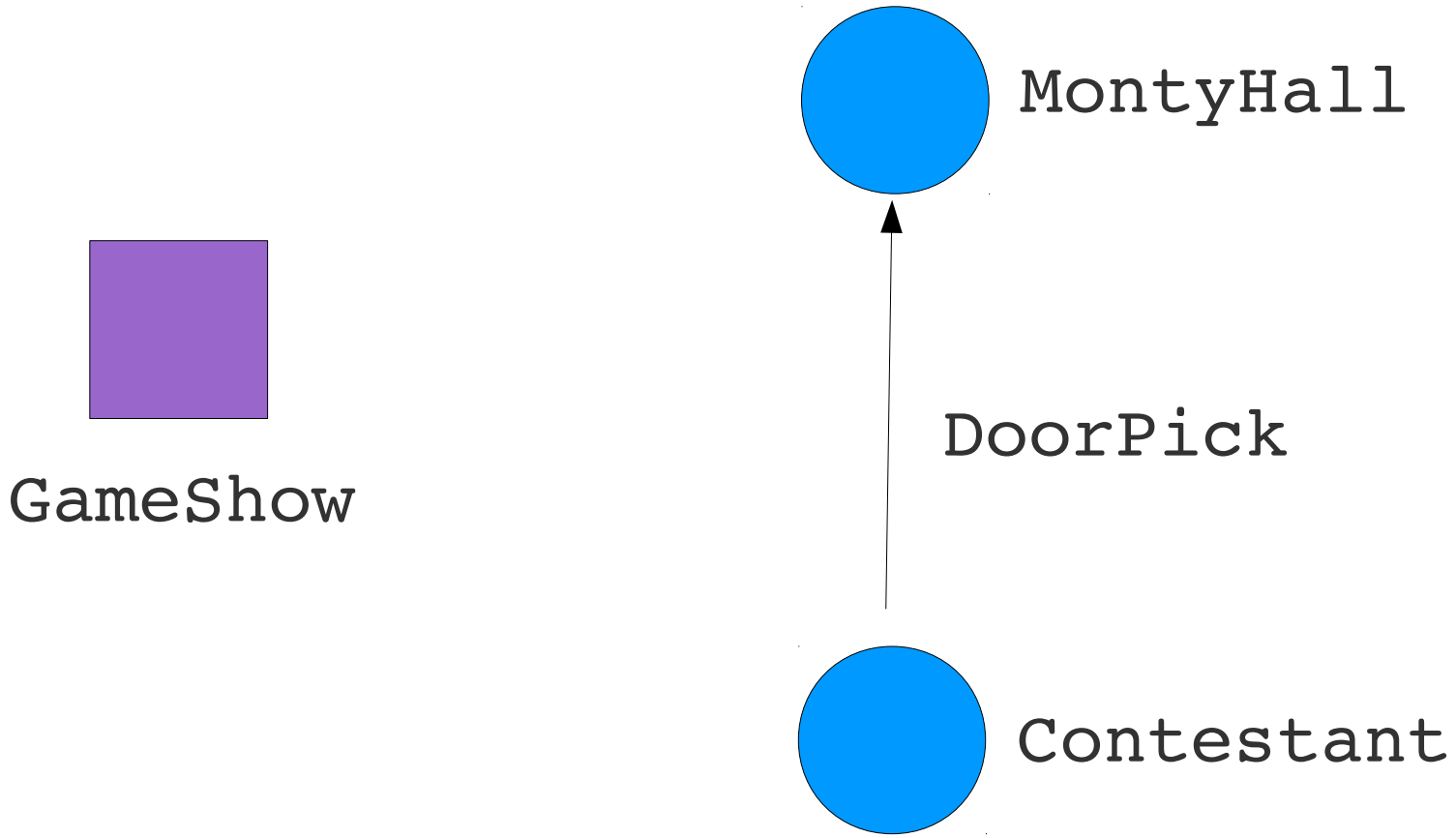


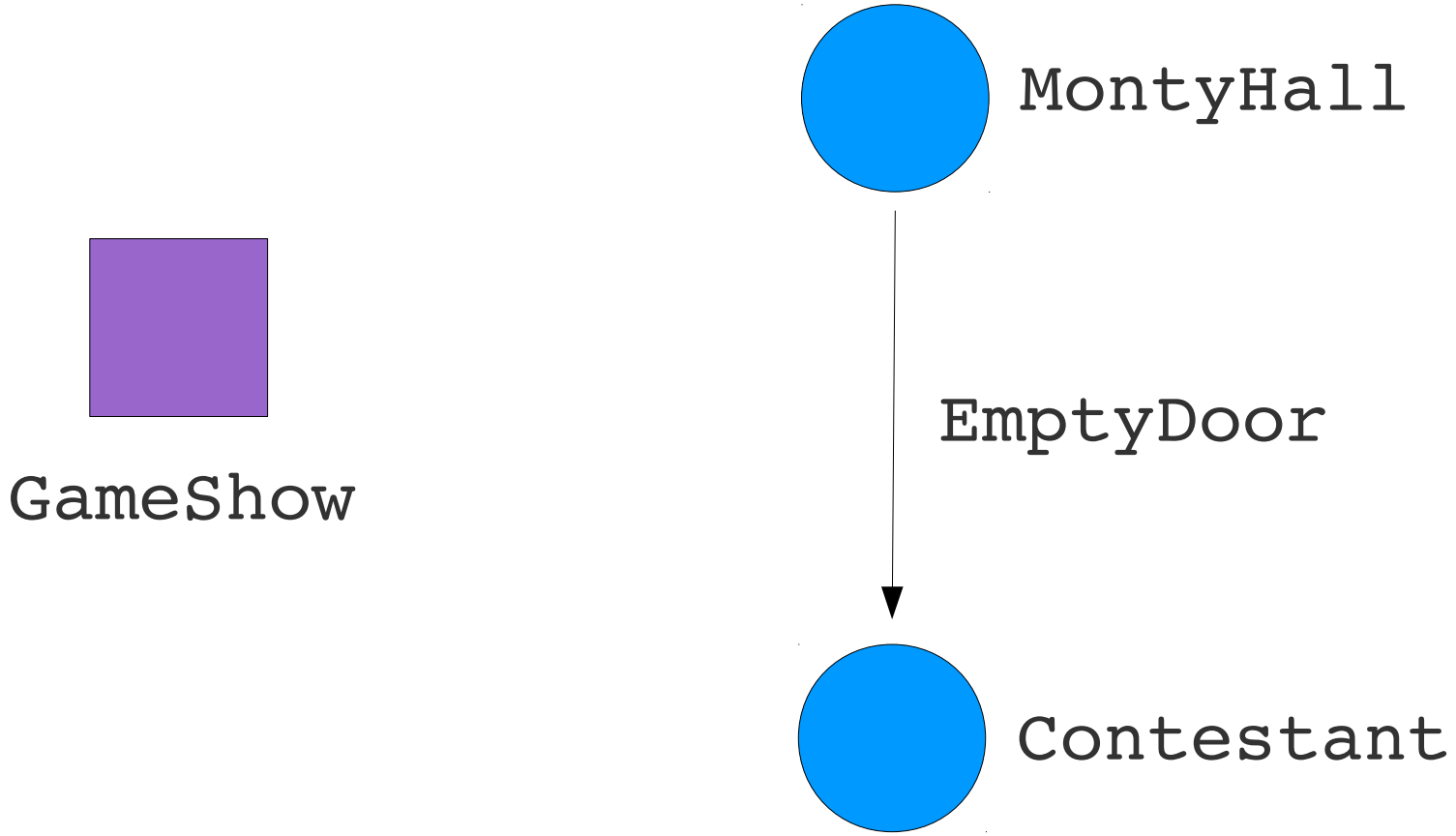
Host: Monty Hall

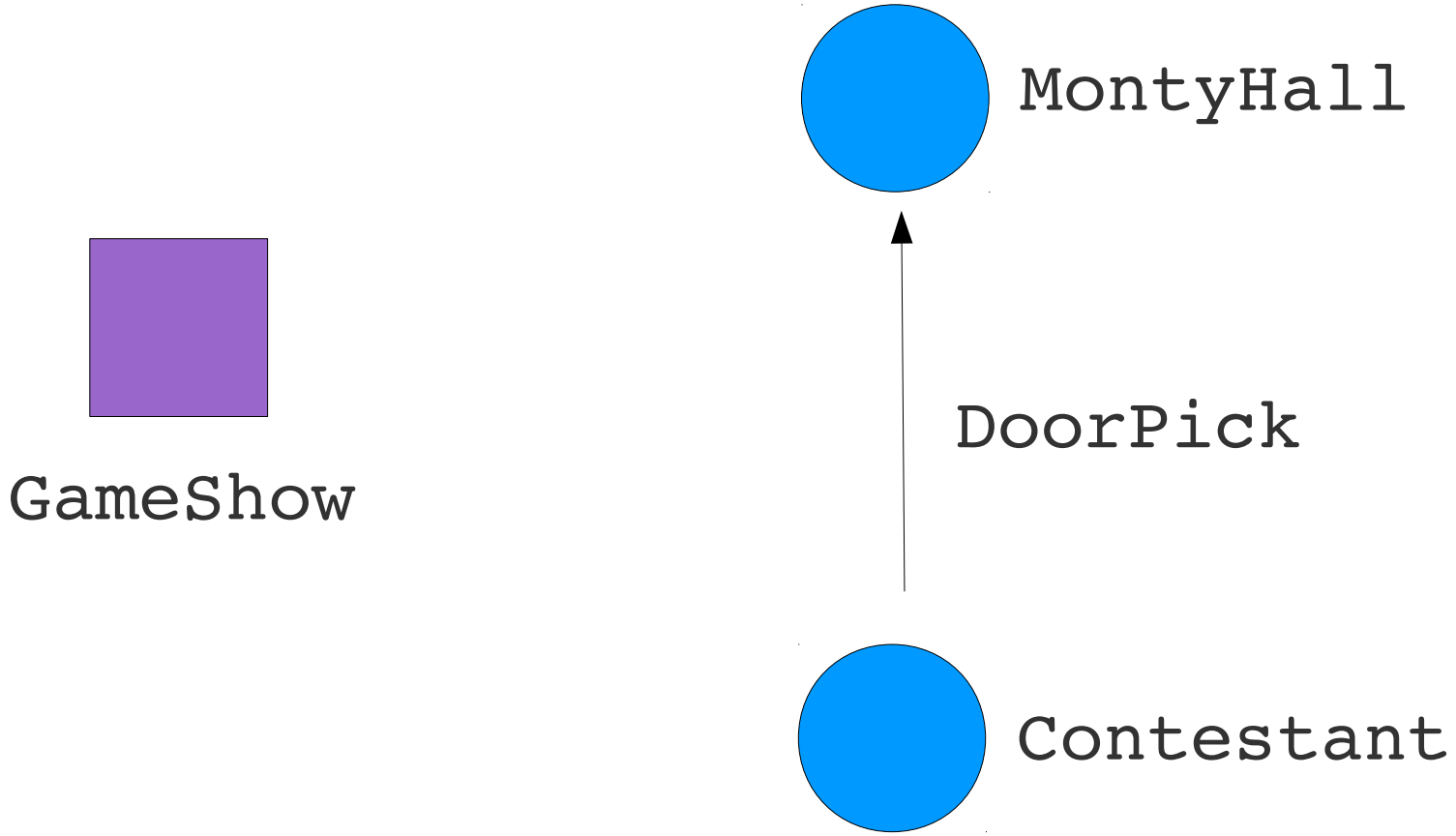

MontyHall
start
GameShow
start
Contestant

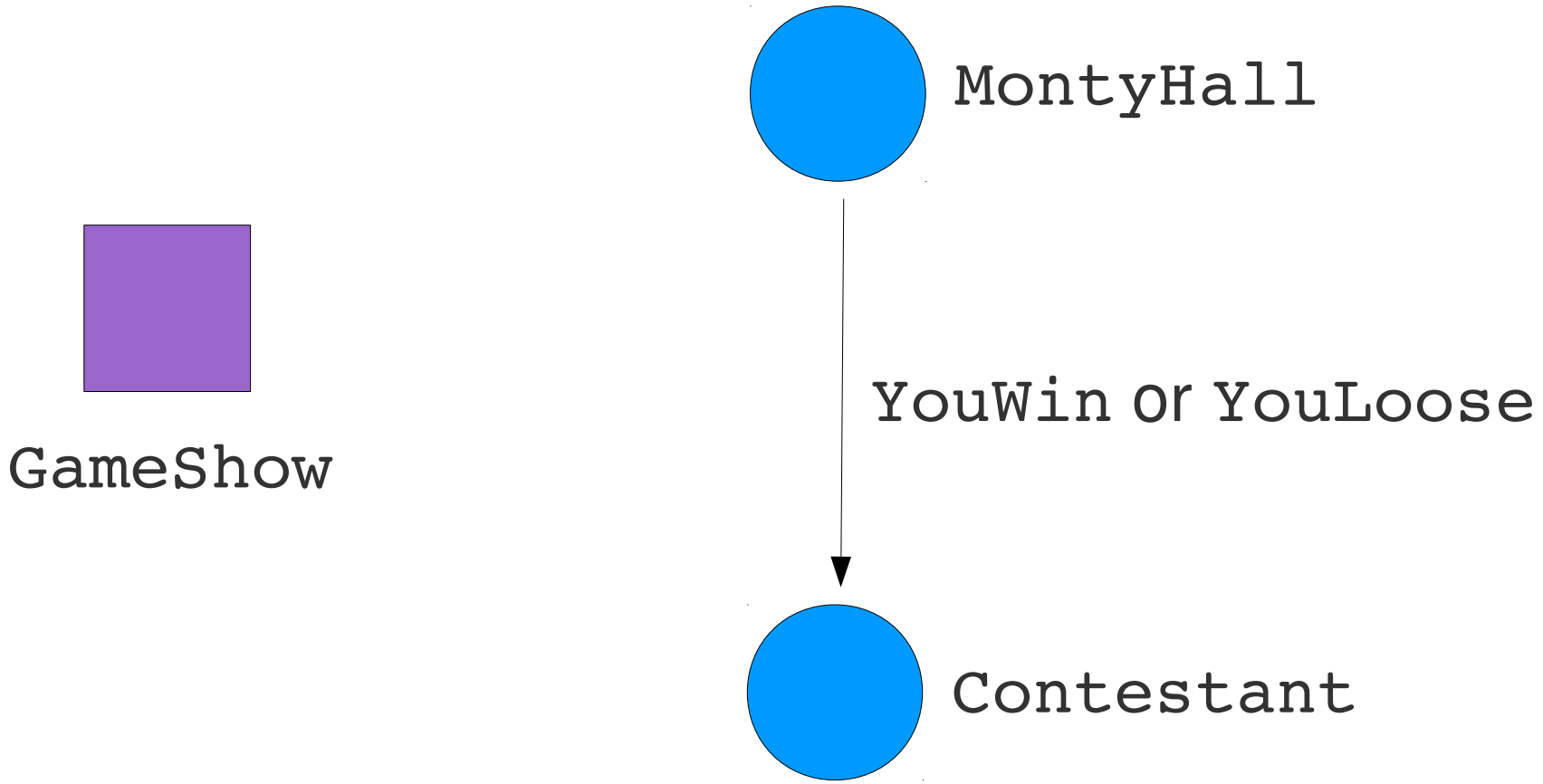

StartGame
MontyHall
GameShow
Contestant


MontyHall
PickADoor
GameShow
Contestant


MontyHall
DoorPick
GameShow
Contestant


MontyHall
EmptyDoor
GameShow
Contestant


MontyHall
DoorPick
GameShow
Contestant


MontyHall
YouWin or YouLoose
GameShow
Contestant

Actors

- Send message to an actor

  otherActor ! *message*

- Messages should be immutable
- Either one actor per thread (`receive`) or actors share threads (`react`)
- `react` Actors should not block

Actors

- Design goal: Event programming (`react`) without Inversion of Control

  ⇒ Complications

- `react` does not return
- Use functions defined on `Actor`: `loop`, `loopWhile`, and `exit`
- Or invoke `act()` from inside `act()`

Actors

Benefits from Actors

- Don't worry about “safe publication”
- Concurrency is explicit in the method calls to `Actor` – easier to analyze

However

- The messages form a protocol
- Protocols can have flaws

  The Clayton tunnel train crash 1861: 21 dead, 176 injured

Scala

Functional programming on the Java Platform

- Immutability
- Functions are first class
- Matching, tuples

You can choose: immutable or mutable

Strong static type system that's easy to use

Thank you for attending this talk

Thanks to the Scala Team at EPFL for creating Scala.

From http://www.scala-lang.org/node/89

> Martin Odersky, Iulian Dragos, Gilles Dubochet, Philipp Haller, Lukas Rytz, Ingo Maier, Antonio Cunei, Stepan Koltsov, Adriaan Moors, Miles Sabin, Geoffrey Washburn, Stéphane Micheloud, Lex Spoon, Sean Mc Dirmid, Burak Emir, Nikolay Mihaylov, Philippe Altherr, Vincent Cremet, Michel Schinz, Erik Stenman and Matthias Zenger