Hochschule **Rosenheim**
University of Applied Sciences

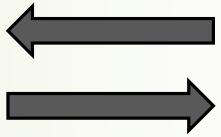
Object-oriented programming

Comparison between Go and Scala

Presentation
by

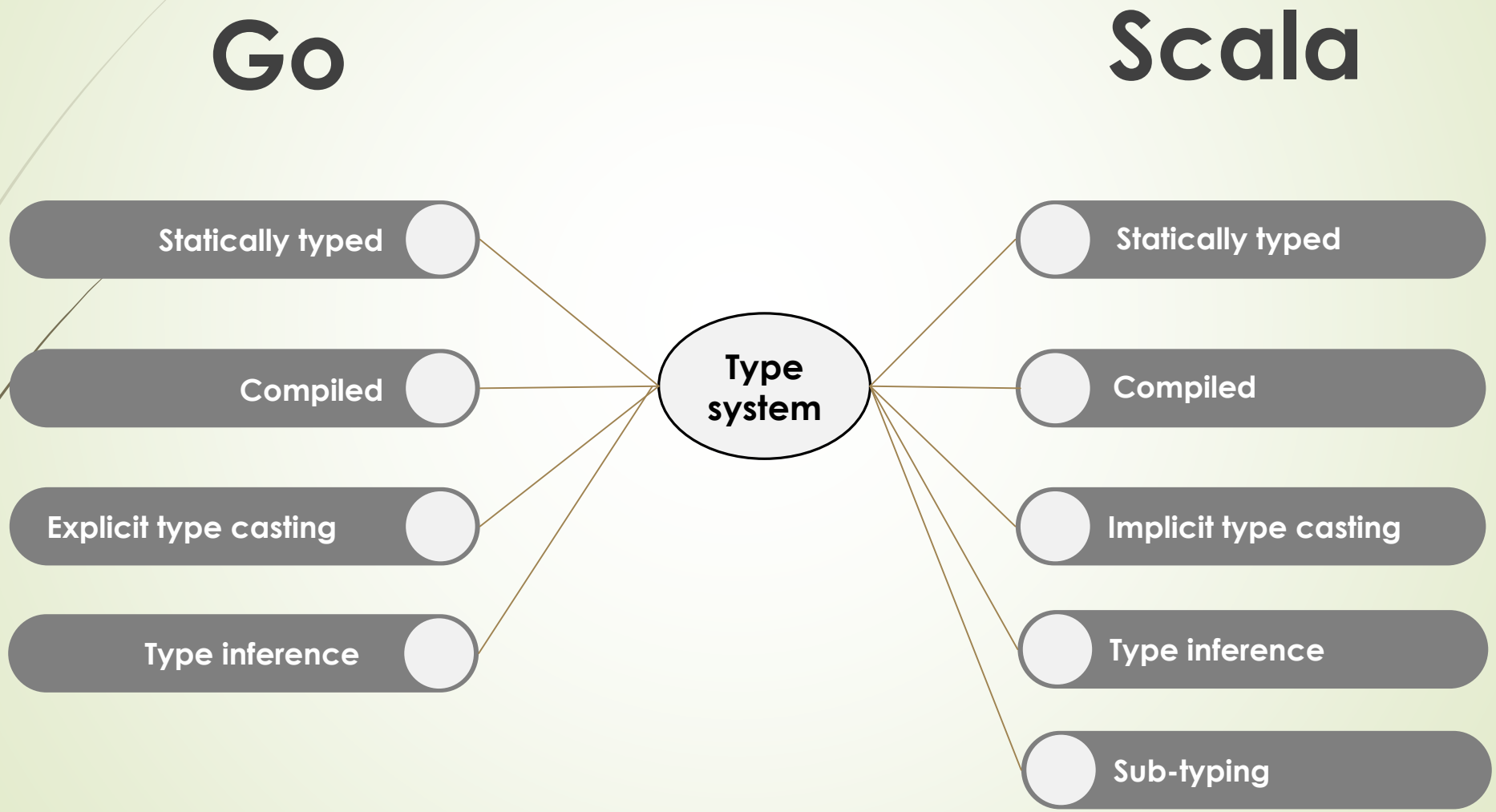
Tobias Djermester

Introduction

Go  **Scala**

1. Type System
2. Encapsulation
3. Inheritance
4. Polymorphism
5. Abstraction
6. OO in concurrent programming

Type system


Go
Scala
Statically typed
Compiled
Explicit type casting
Type inference
Type system
Statically typed
Compiled
Implicit type casting
Type inference
Sub-typing

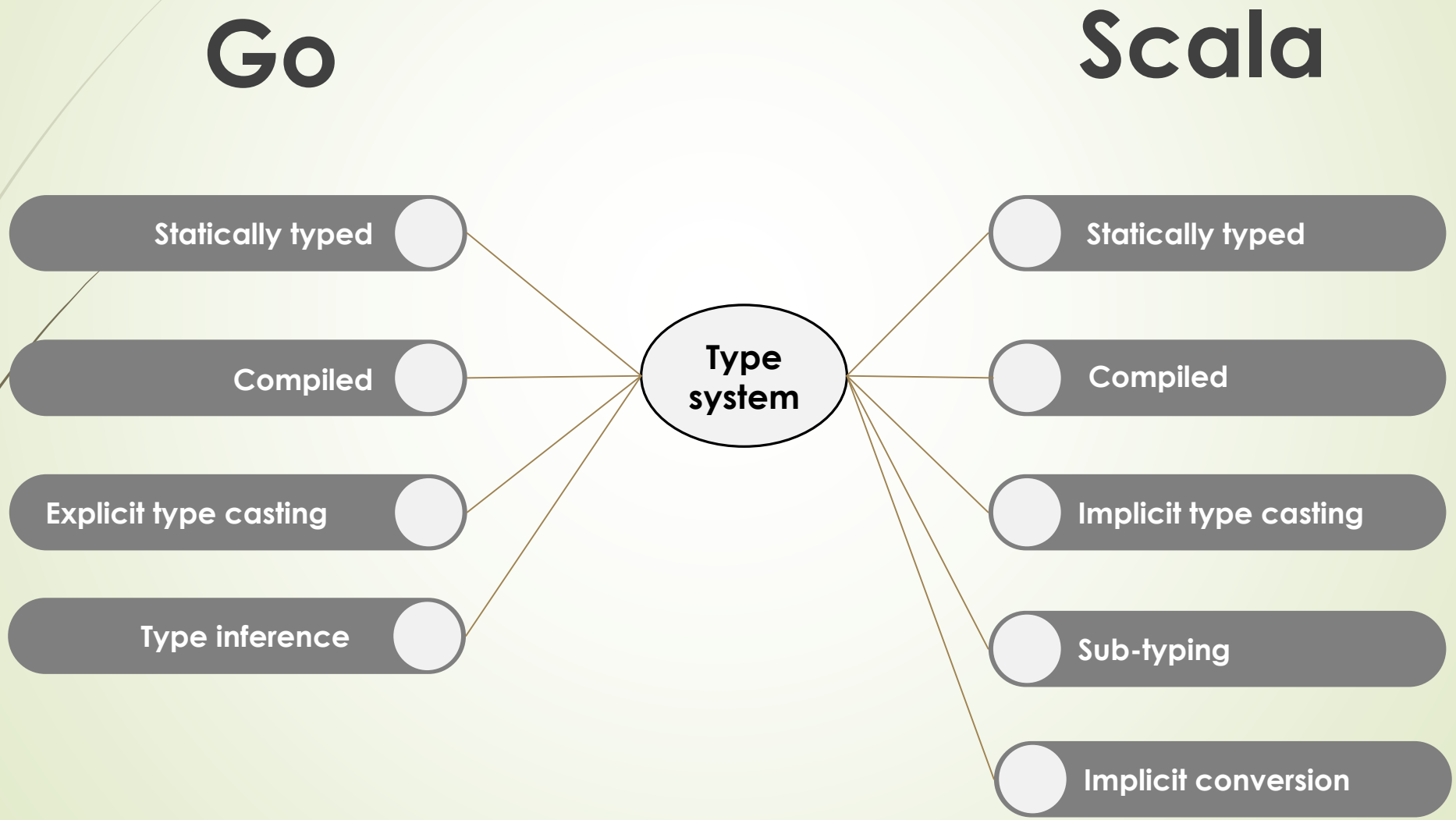
Type system

Sub-typing

```scala
val list1 = List(10, 'a')
// -> List[Int] = List(10, 97)

val list2 = List(20.2, 10)
// -> List[Double] = List(20.2, 10.0)

val list3 = List("Hello", 10, true)
// -> List[Any] = List(Hello, 10, true)
```

Scala

Type system


Go
Scala
Statically typed
Compiled
Explicit type casting
Type inference
Type system
Statically typed
Compiled
Implicit type casting
Sub-typing
Implicit conversion

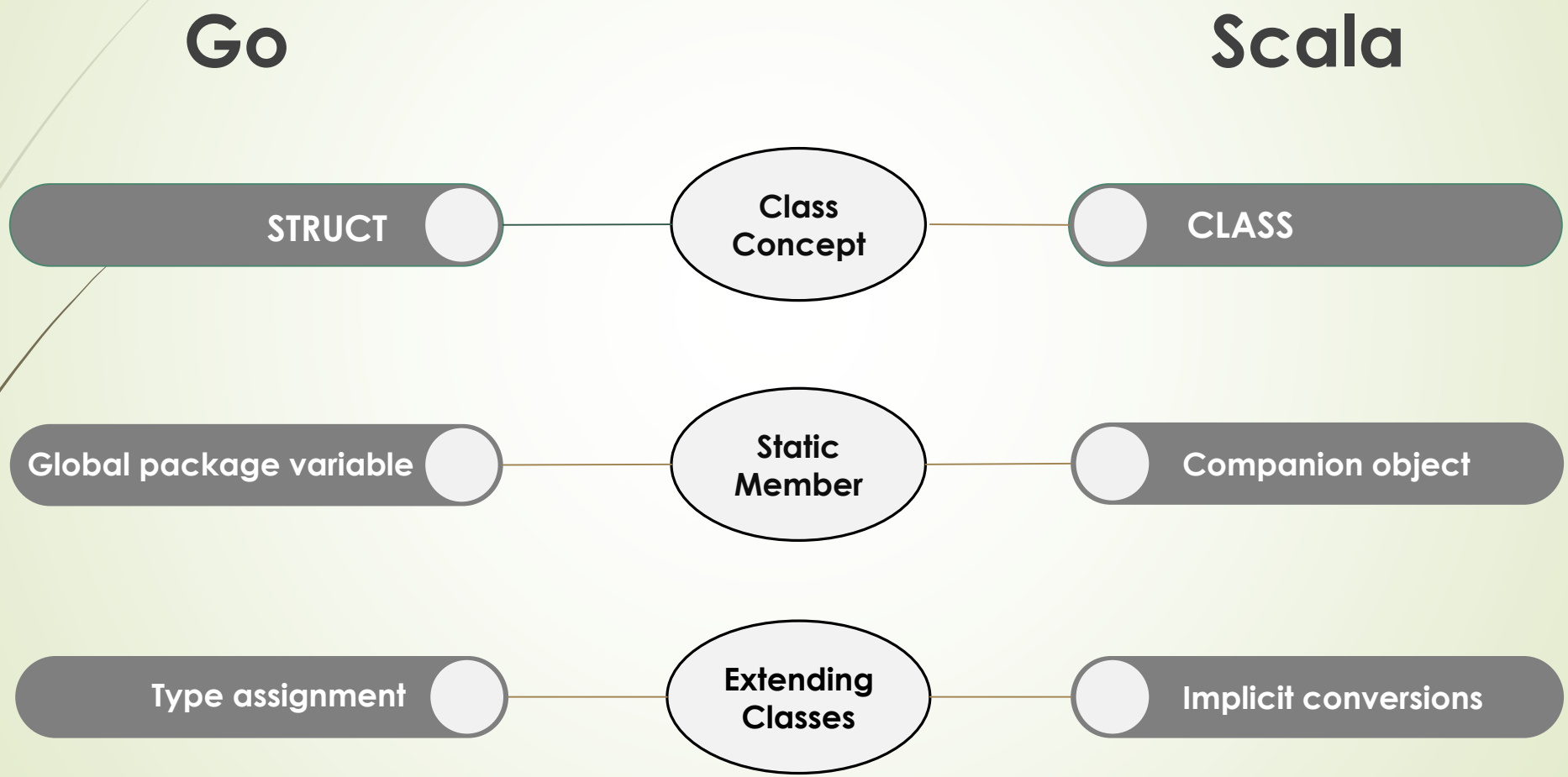
Type system

Implicit conversion

```scala
implicit def str2int(str:String):Int = Integer.parseInt(str)

def addTwo(a:Int, b:Int) :Int  = a + b
addTwo("123", 456)
```

Scala

Encapsulation



Encapsulation

Class concept

```go
type circle struct {
    radius  float64
}

func NewCircle(radius float64) *circle {
    c := new(circle)
    c.radius = radius
    return c
}

func (c *circle) calculateArea() float64 {
    area := math.Pi * math.Exp2(c.radius)
    return area
}
```

Go

Encapsulation

Companion object

```scala
class Circle(var radius: Double) {
  import Circle._

  circlesCount += 1
  def area: Double = calculateArea(radius)
}

object Circle {

  private var circlesCount = 0
  private def calculateArea(radius: Double): Double = Pi * pow(radius, 2.0)
  def getNumberOfCircles: Int = circlesCount
}

object CompanionObject {

  def main(args: Array[String]) : Unit  {

    val circle1 = new Circle( radius = 5.0)
    val circle2 = new Circle( radius = 5.0)
    circle2.area
    println(Circle.getNumberOfCircles)
  }
}
```

Scala

Encapsulation

Type assignment

```go
type Int int

func (i Int) Add(j Int) Int {
    return i + j
}

func main() {
    i := Int(5)
    j := Int(6)
    fmt.Println(i.Add(j))
    fmt.Println(i.Add(j) + 12)
}
```

Go

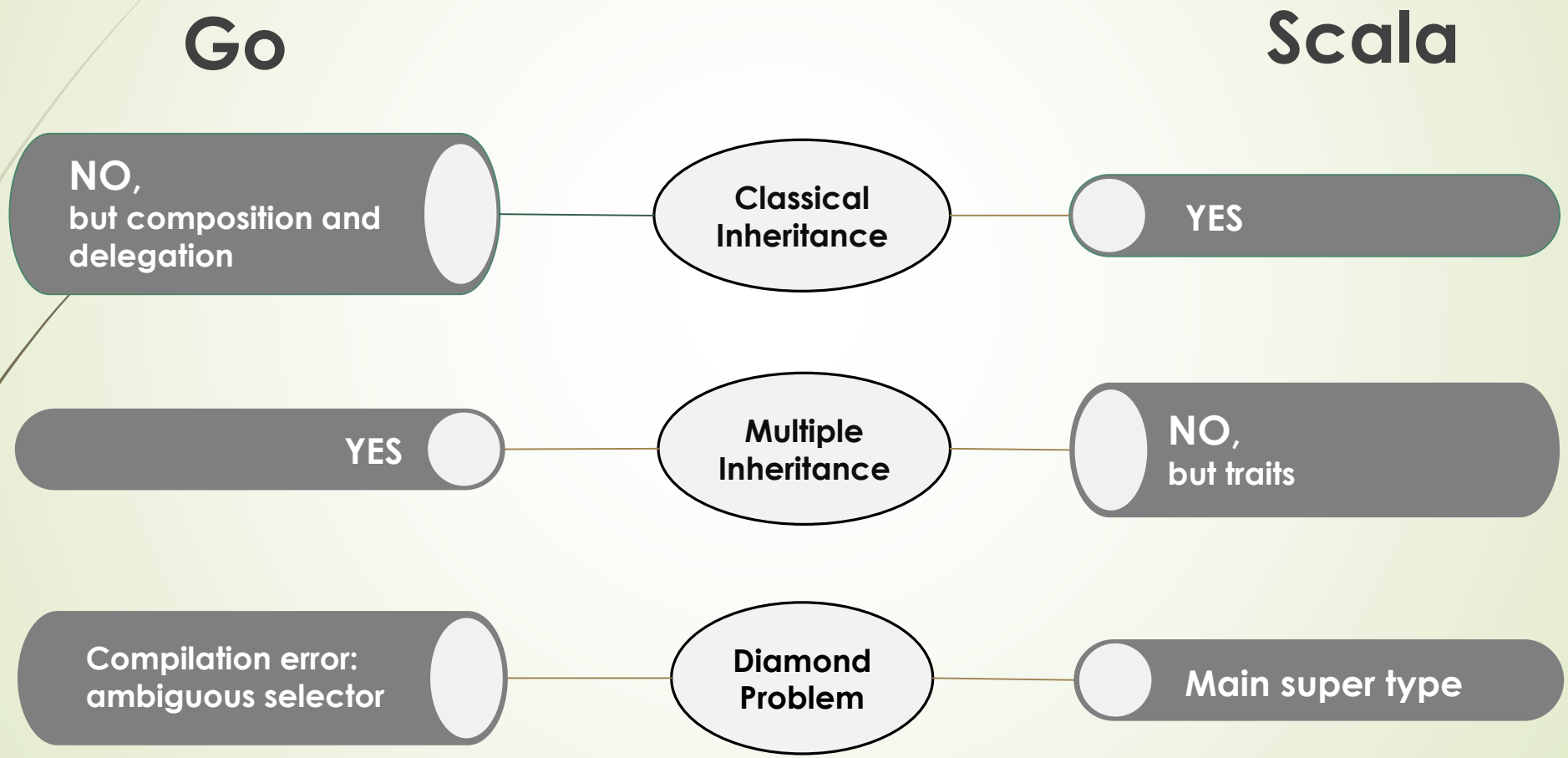
Encapsulation

Implicit conversion

```scala
class MyString(str:String) {
  def printSelf() {
    println(str)
  }
}

implicit def str2myString(str:String) : MyString = new MyString(str)

"Hello".printSelf()
```

Scala

Inheritance



Inheritance

Multiple inheritance

```
type geometricShape struct {
    id int
    description string
}

type drawing struct {
    color color.Color
    rotation float64
    description string
}

type circle struct {
    radius  float64
    geometricShape
    drawing
}

func main() {
    var c = NewCircle( radius: 15)
    c.color = color.Black
    c.id = 5
    c.geometricShape.description = "Geometric shape description"
    c.drawing.description = "Drawing description"
}
```

Diamond Problem

Go

Inheritance

Multiple inheritance

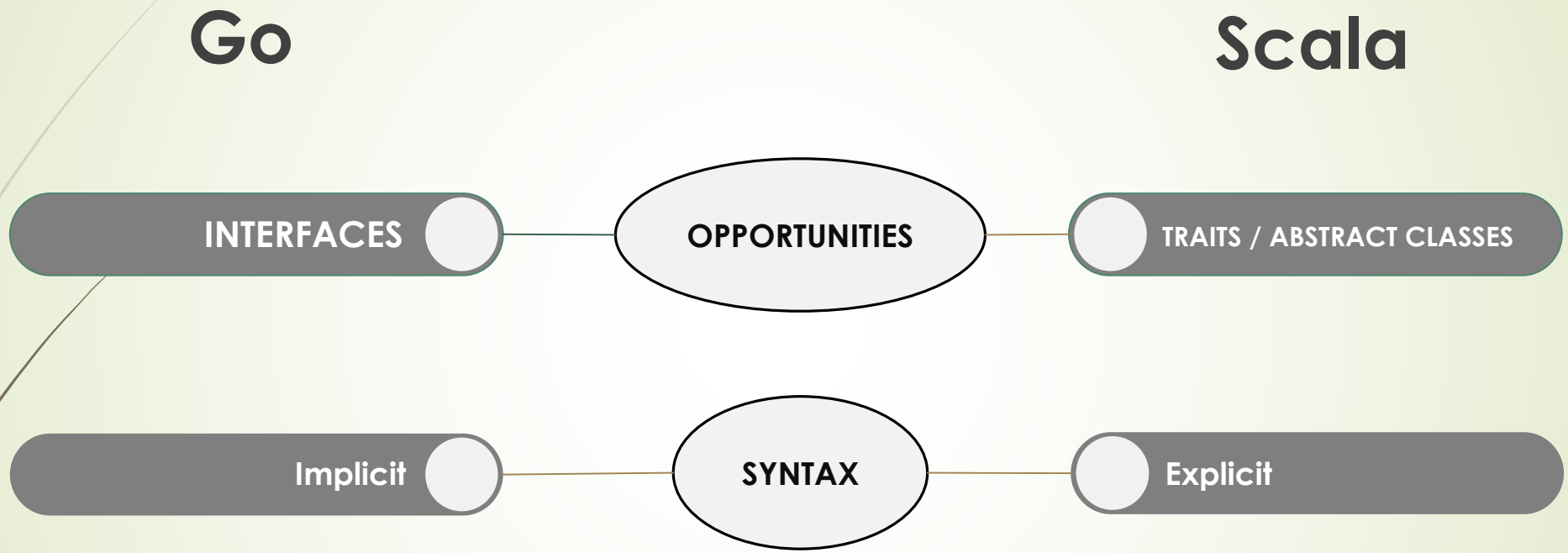
```scala
class Circle(override val id: Int, override val description: String, var radius: Double)
  extends GeometricShape(id, description) with Drawing {
}

class GeometricShape(val id: Int, val description: String) {
  def printDescription() {
    println(description)
  }
}

trait Drawing {
  var color: Color = Color.BLACK
  var rotation: Double = 0
  var description: String = "Default description"

  def printDescription() {
    println(description)
  }
}

object Main {
  def main(args: Array[String]) {
    val circle = new Circle( id = 1,  description = "description",  radius = 5.0)
    circle.color = Color.RED
    circle.printDescription()
  }
}
```

Diamond problem solution

Diamond problem

Scala

Polymorphism



Polymorphism

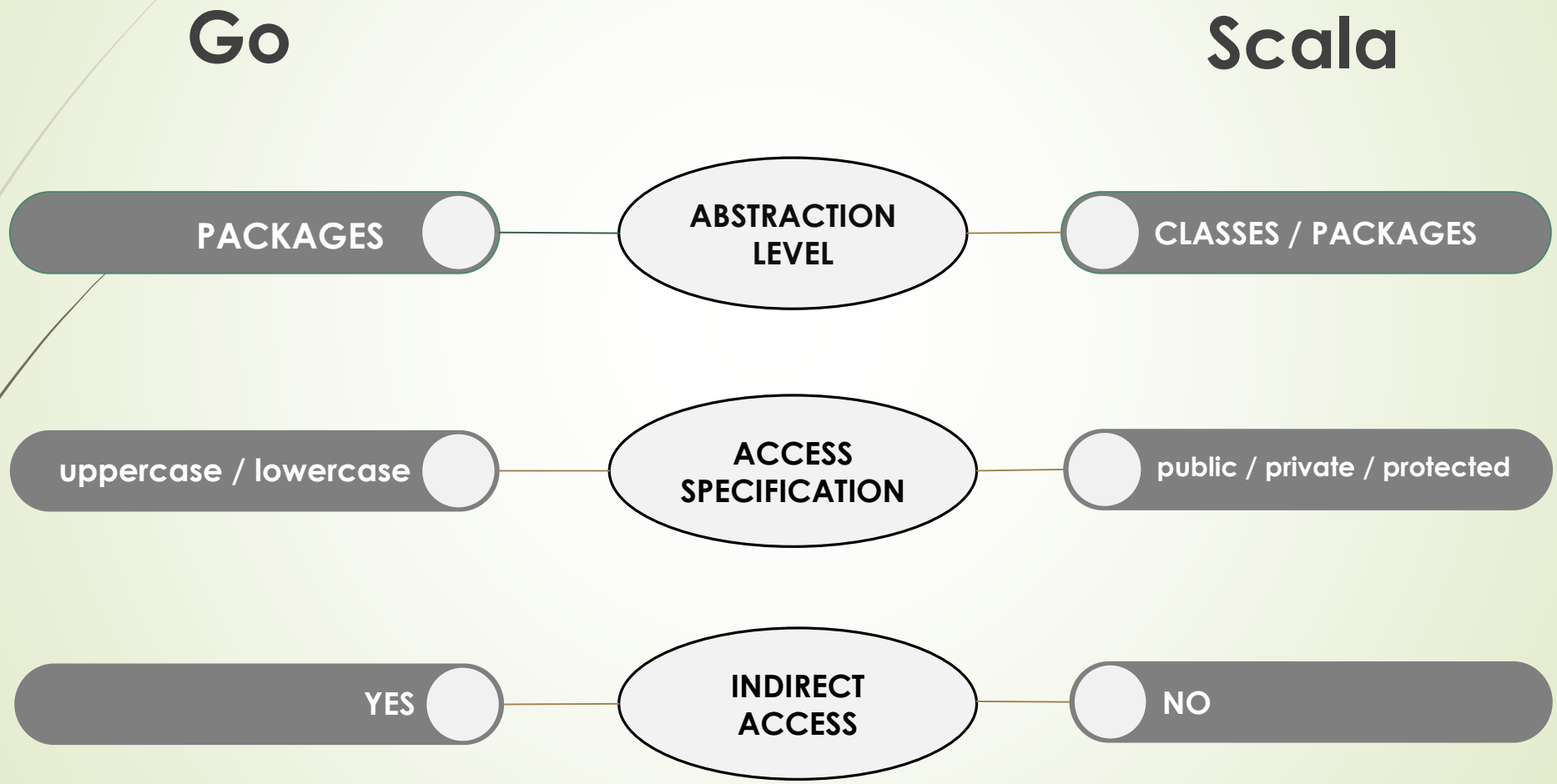
```go
type geometricShapeInterface interface {
    calculateArea() float64
}

type rectangle struct {
    length float64
    width float64
}

func (c *circle) calculateArea() float64 {
    area := math.Pi *  math.Pow(c.radius, y: 2)
    return area
}

func (r *rectangle) calculateArea() float64 {
    area := r.length * r.width
    return area
}

func main() {
    circle := &circle{ radius: 15}
    rectangle := &rectangle{ length: 10, width: 15}
    shapes := []geometricShapeInterface{circle, rectangle}
    var totalArea float64 = 0
    for _, shape := range shapes {
        totalArea += shape.calculateArea()
    }
}
```

Go

```scala
trait GeometricShapeInterface {
  def calculateArea(): Double
  def printDescription() {
    println("Default description")
  }
}

class Circle( var radius: Double)
  extends GeometricShapeInterface {
  def calculateArea(): Double = Pi * pow(radius, 2.0)
}

class Rectangle(var length: Double, var width: Double)
extends GeometricShapeInterface {
  def calculateArea(): Double = length * width
}

object Main {
  def main(args: Array[String]) {
    val circle = new Circle( radius = 10)
    val rectangle = new Rectangle( length = 10, width = 15)
    val shapes = Array(circle, rectangle)
    var totalArea = 0.0
    for ( shape <- shapes) {
      totalArea += shape.calculateArea()
    }
  }
}
```

Scala

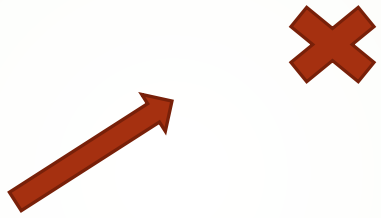
Abstraction



Abstraction

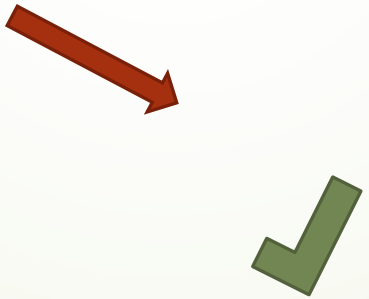
```go
package shapes

type geometricShape struct {
    Id int
    Description string
}

type Circle struct {
    geometricShape
    Radius  float64
}
```

Go



```go
package main

import (
    "./shapes"
)

func main() {

    circle := shapes.Circle{
        GeometricShape: shapes.GeometricShape{
            Id: 5,
            Description: "description of circle",
        },
        Radius: 10,
    }
    println(circle.Description)
}
```

Go



```go
package main

import (
    "./shapes"
)

func main() {

    circle := shapes.Circle{
        Radius: 10,
    }
    circle.Id = 5
    circle.Description = "description of circle"

    println(circle.Description)
}
```

Go

Object orientation in concurrent programming

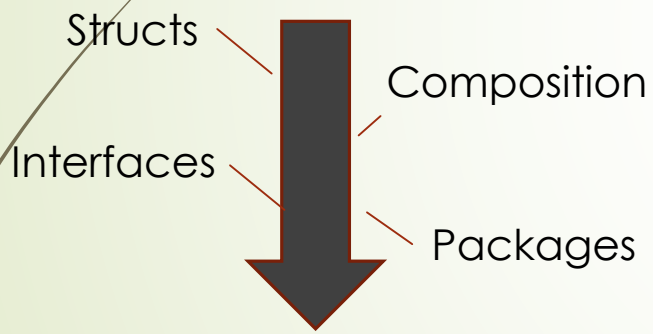
```
case class CalculateArea(radius: Double)

class CircleCalculator extends Actor {

  def receive : PartialFunction[Any, Unit] = {
    case CalculateArea(radius) => actorRef2Scala(sender) ! Pi * pow(radius, 2.0)(self)
}
}

def main(args: Array[String]) : Unit  {

  val system = ActorSystem("CalculatorActorSystem")
  val myCircleCalculator = system.actorOf(Props[CircleCalculator](ClassTag), name = "myCircleCalculator")

  implicit val timeout = Timeout(10 seconds)
  val future = ask(myCircleCalculator) ? CalculateArea(19)(timeout, sender)
  val result = Await.result(future, timeout.duration).asInstanceOf[Double]
  println(result)
}
```

In Go: Similar functionality by goroutines and channels

Conclusion

Go

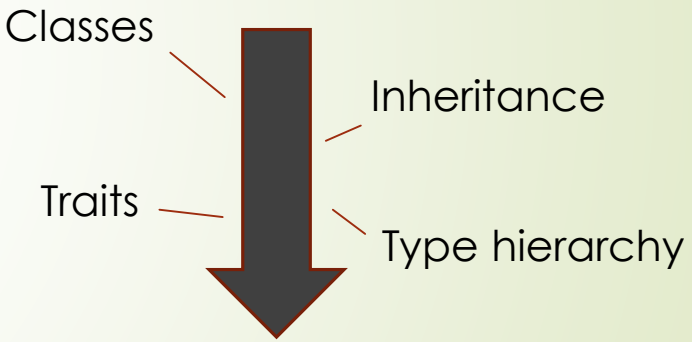
Less Object Oriented



Less opportunities to accomblish object orientation

Scala

More Object Oriented



More opportunities to accomblish object orientation