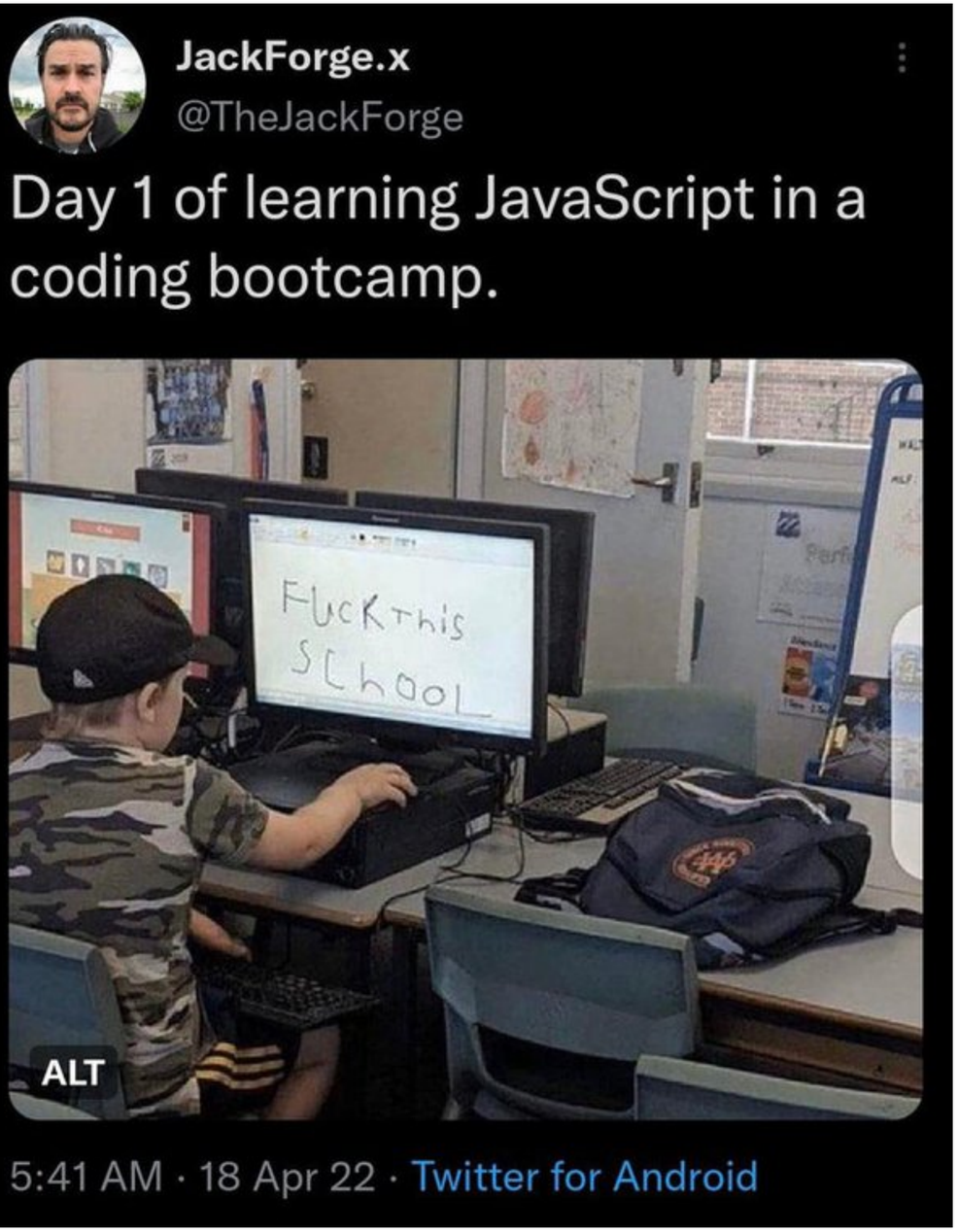
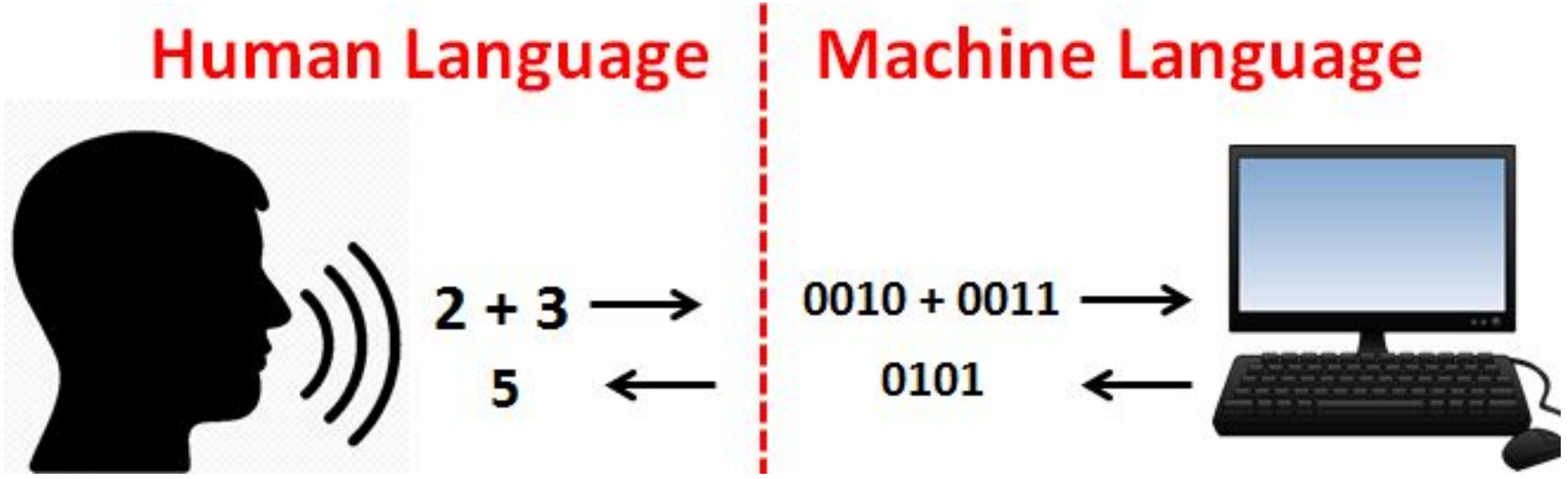

JackForge.x
@TheJackForge
Day 1 of learning JavaScript in a coding bootcamp.
FUCK THIS SCHOOL
ALT
5:41 AM · 18 Apr 22 · Twitter for Android

Programming = give the computer tasks to perform


Human Language
Machine Language
2 + 3
5
0010 + 0011
0101

High level language

Machine code

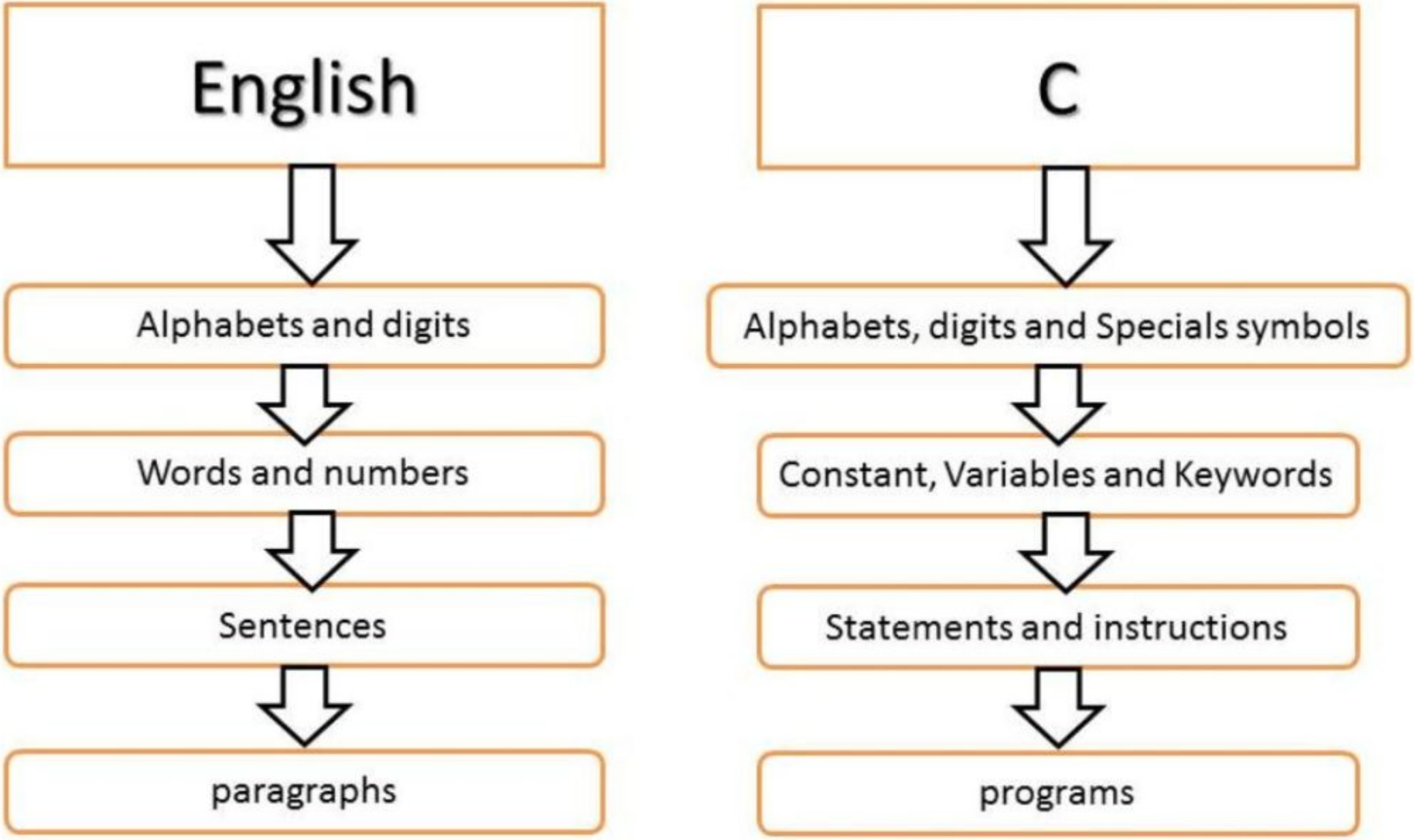
Easy for programmer to understand

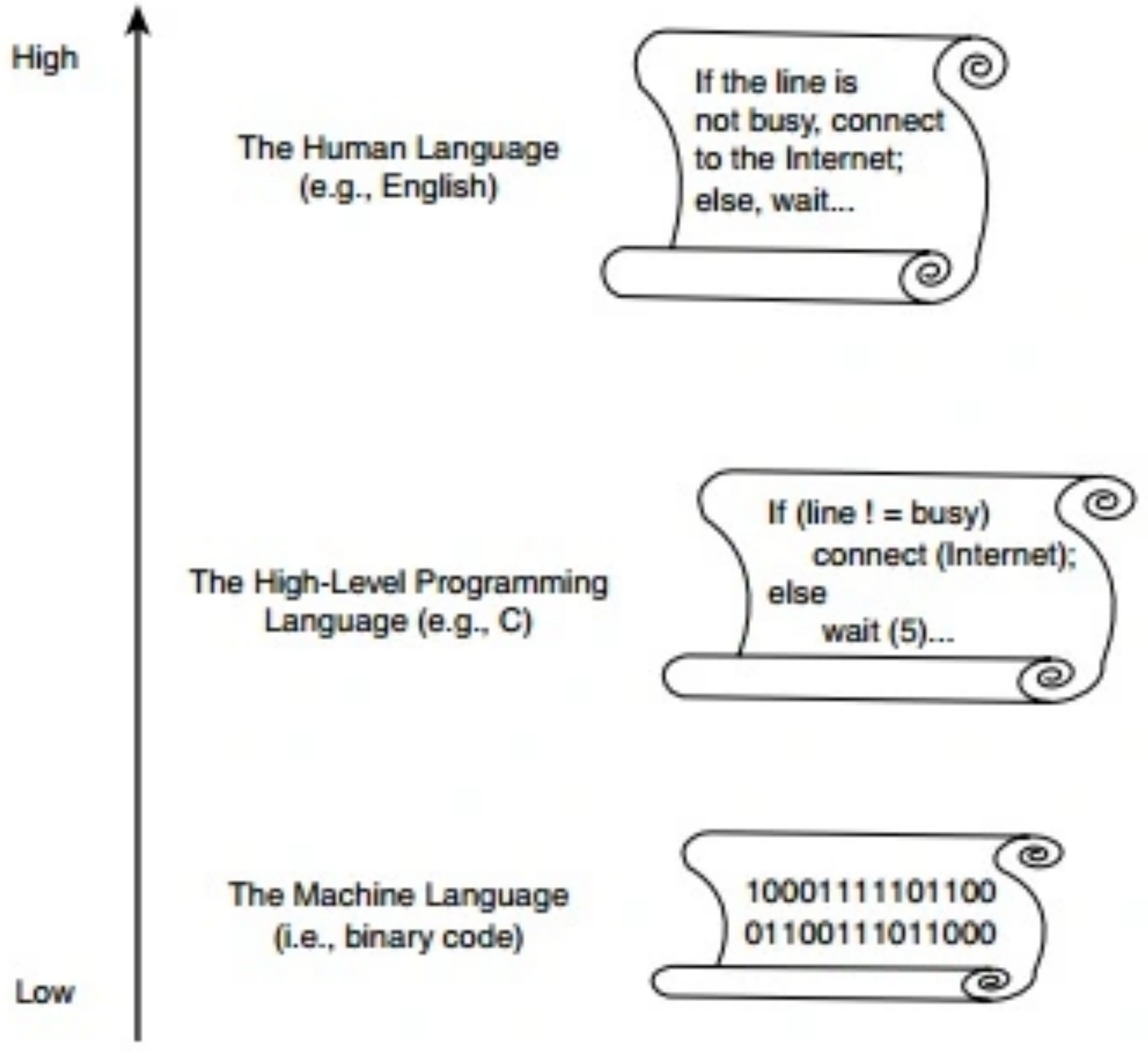
Contains English words

Translator program

The computer's own language

Binary numbers All 1s and 0s

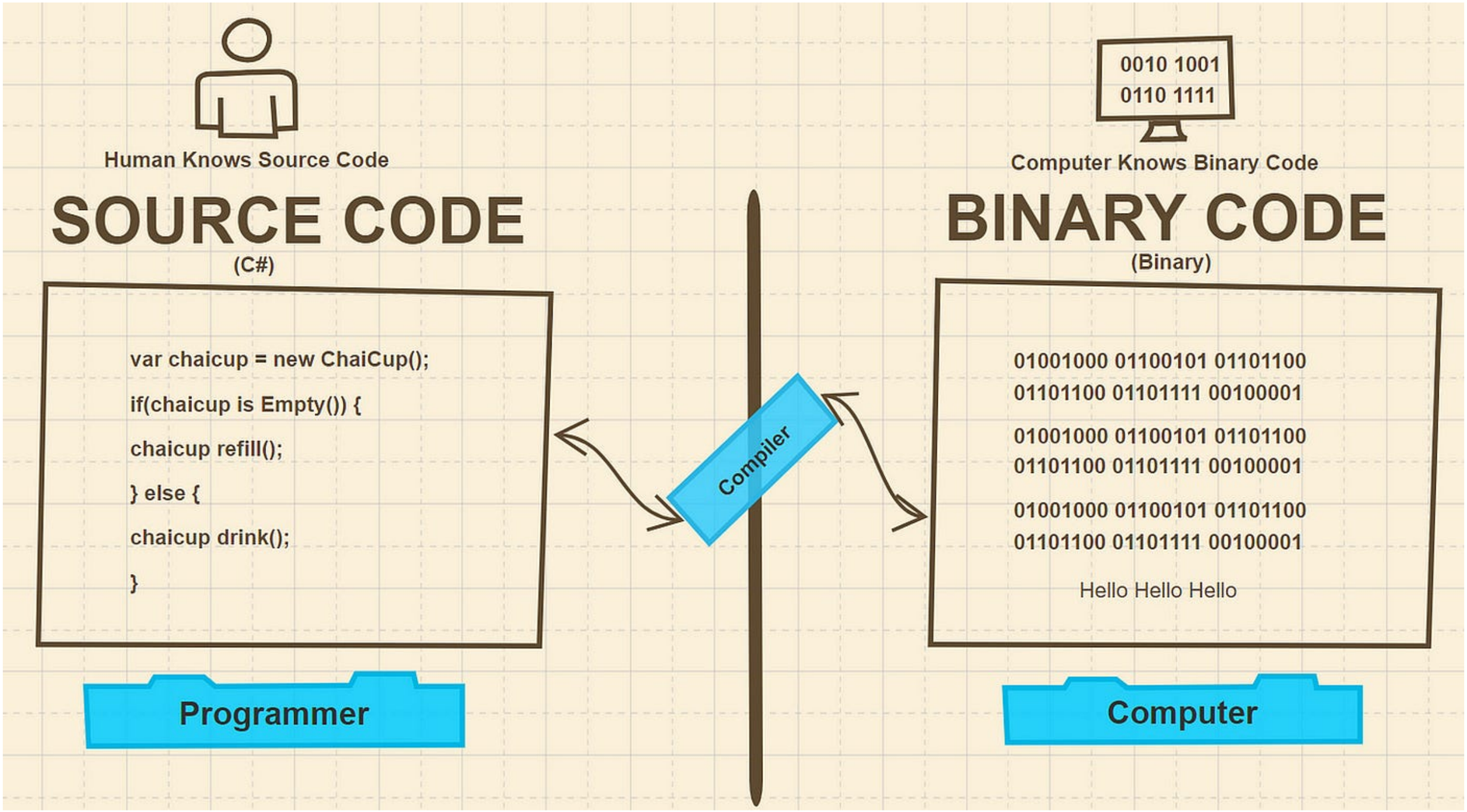

English
Alphabets and digits
Words and numbers
Sentences
paragraphs
C
Alphabets, digits and Specials symbols
Constant, Variables and Keywords
Statements and instructions
programs


High
The Human Language
(e.g., English)
If the line is
not busy, connect
to the Internet;
else, wait...
The High-Level Programming
Language (e.g., C)
If (line ! = busy)
connect (Internet);
else
wait (5)...
The Machine Language
(i.e., binary code)
10001111101100
01100111011000
Low


COMPUTER
PROGRAMMER
COMPILER/INTERPRETER
imgflip.com

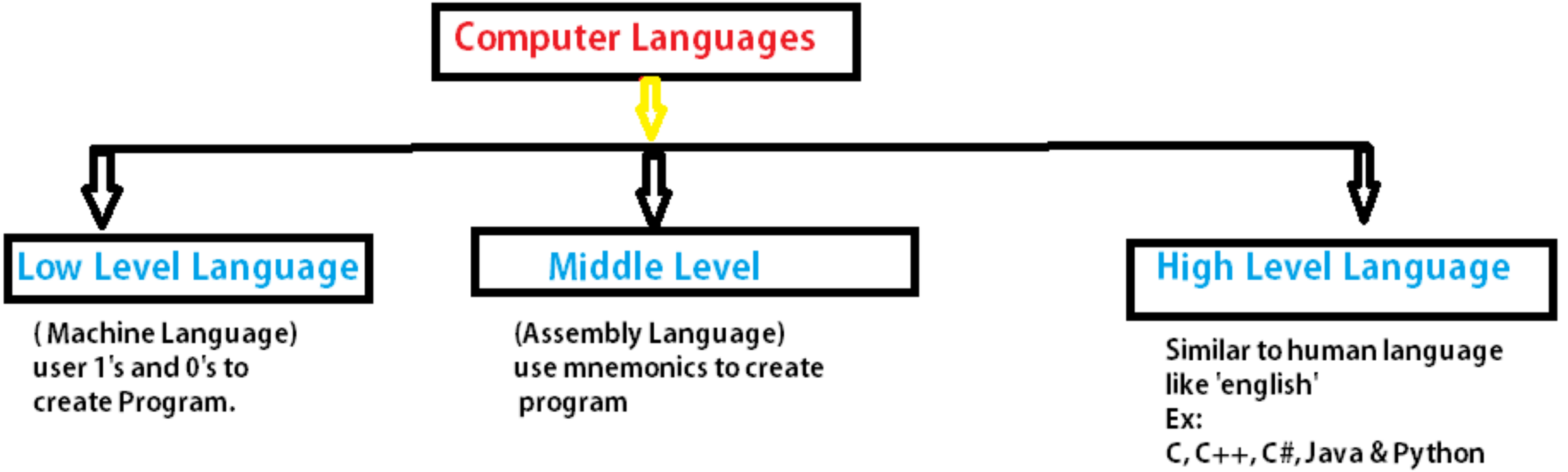
How Compiler Works

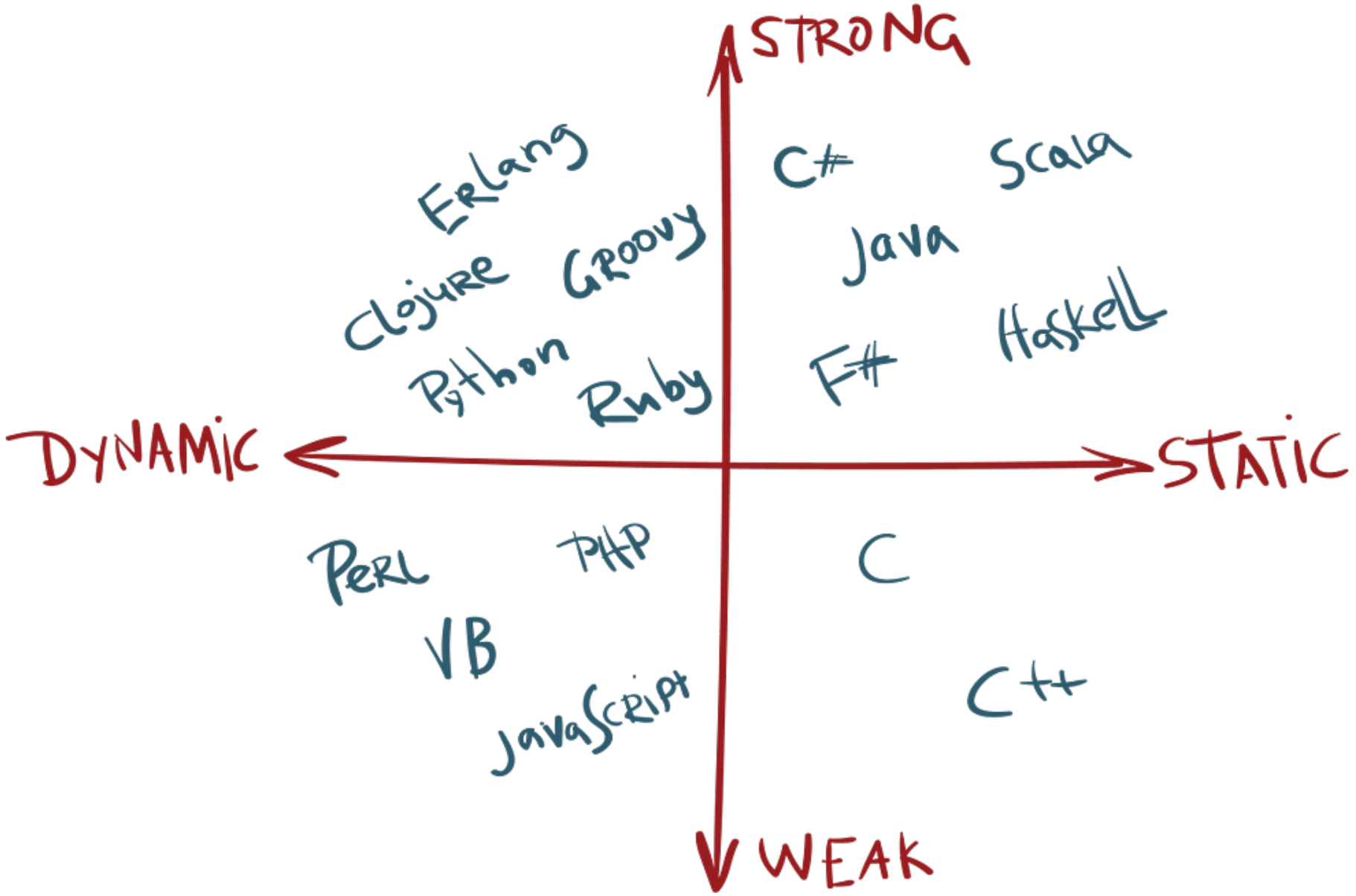
How Interpreter Works

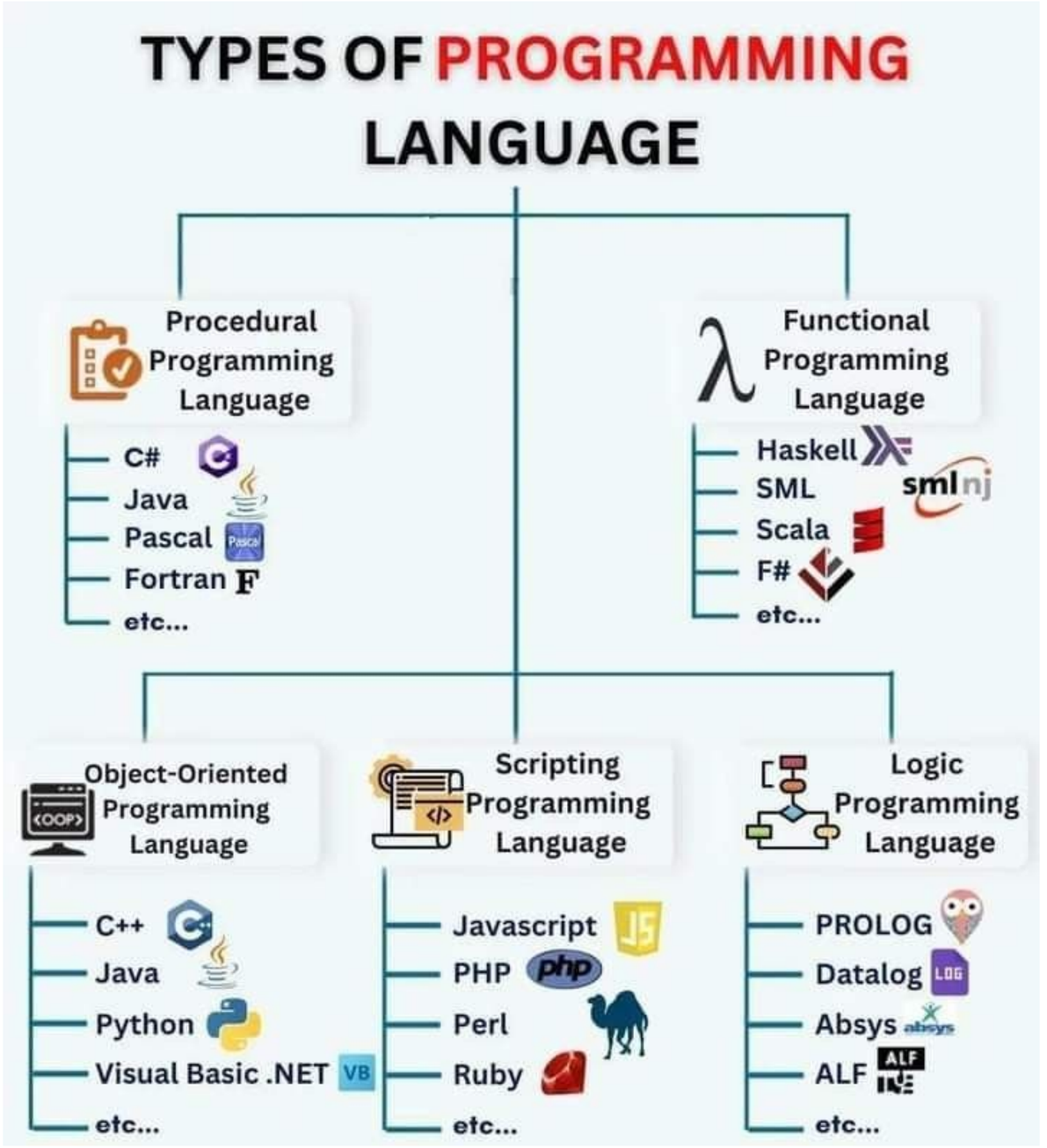

Human Knows Source Code
SOURCE CODE
(C#)
var chaicup = new ChaiCup();
if(chaicup is Empty()) {
chaicup refill();
} else {
chaicup drink();
}
Programmer
Compiler
0010 1001
0110 1111
Computer Knows Binary Code
BINARY CODE
(Binary)
01001000 01100101 01101100
01101100 01101111 00100001
01001000 01100101 01101100
01101100 01101111 00100001
01001000 01100101 01101100
01101100 01101111 00100001
Hello Hello Hello
Computer

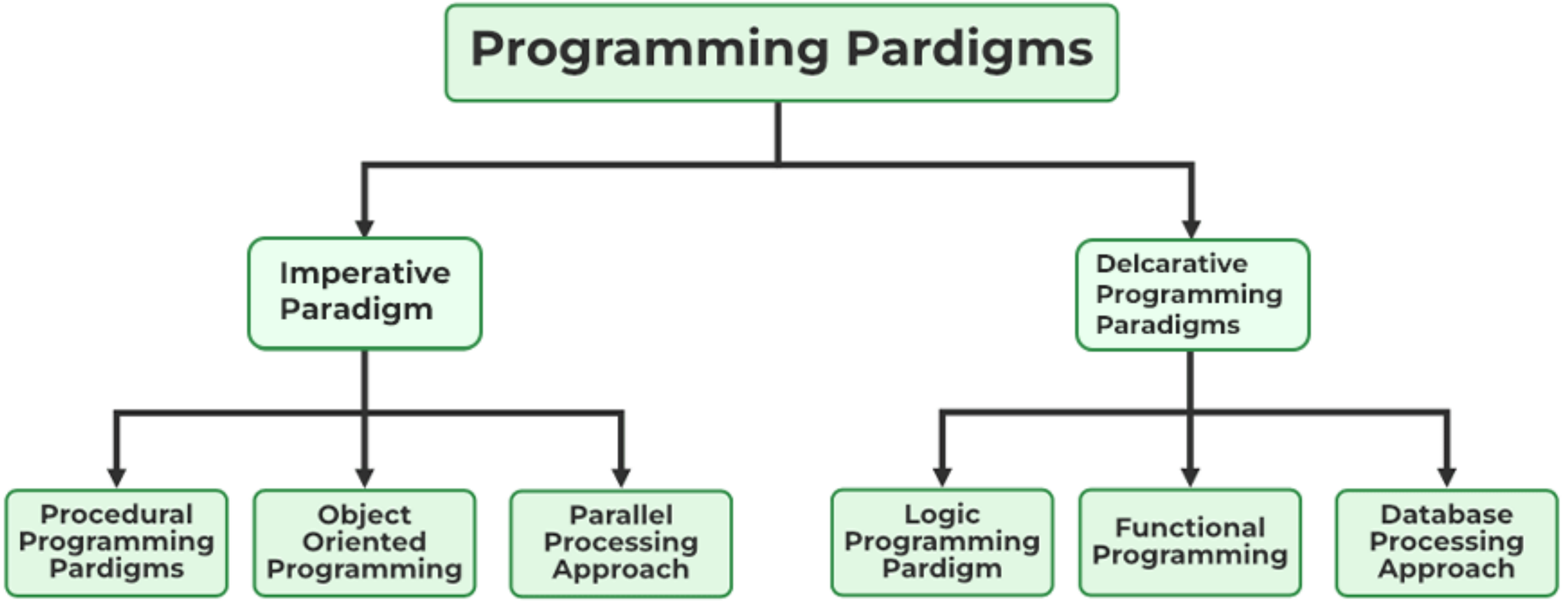
A [edit]

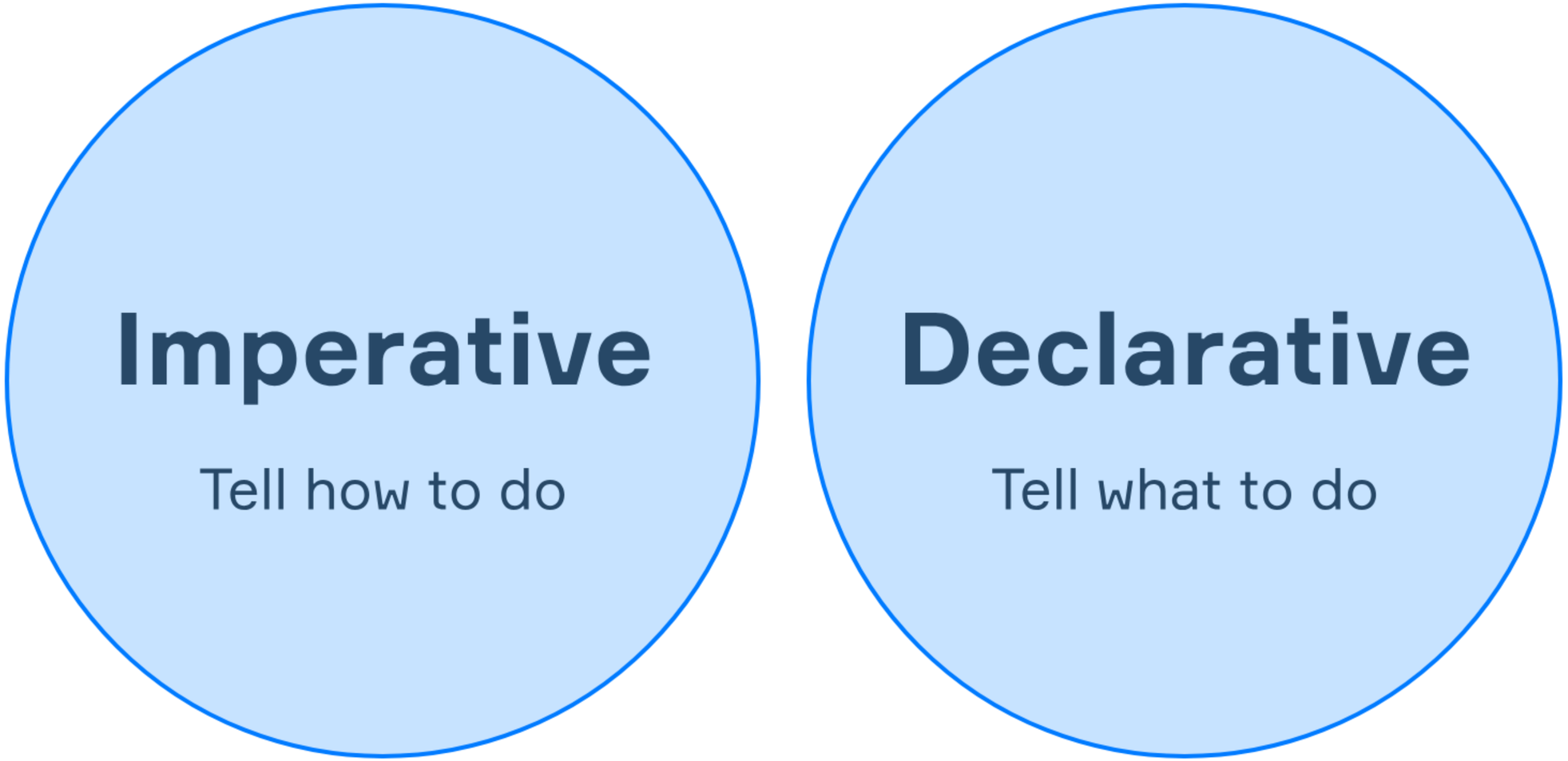
- A# .NET
- A# (Axiom)
- A-0 System
- A+
- A++
- ABAP
- ABC
- ABC ALGOL
- ABLE
- ABSET
- ABSYS
- ACC
- Accent
- Ace DASL
- ACL2
- ACT-III
- Action!
- ActionScript
- Ada
- Adenine
- Agda
- Agilent VEE
- Agora
- AIMMS
- Alef
- ALF
- ALGOL 58
- ALGOL 60
- ALGOL 68
- ALGOL W
- Alice
- Alma-0
- AmbientTalk
- Amiga E
- AMOS
- AMPL
- Apex (Salesforce.com)
- APL
- App Inventor for Android's visual block language
- AppleScript
- Arc
- ARexx
- Argus
- AspectJ
- Assembly language
- ATS
- Ateji PX
- AutoHotkey
- Autocoder
- AutoIt
- AutoLISP / Visual LISP
- Averest
- AWK
- Axum


Computer Languages
Low Level Language
(Machine Language)
user 1's and 0's to
create Program.
Middle Level
(Assembly Language)
use mnemonics to create
program
High Level Language
Similar to human language
like 'english'
Ex:
C, C++, C#, Java & Python


STRONG
WEAK
DYNAMIC
STATIC
Erlang
Clojure
Groovy
Python
Ruby
C#
Scala
Java
F#
Haskell
Perl
PHP
VB
JavaScript
C
C++


TYPES OF PROGRAMMING LANGUAGE
Procedural Programming Language
C#
Java
Pascal
Fortran
etc...
Functional Programming Language
Haskell
SML
Scala
F#
etc...
Object-Oriented Programming Language
C++
Java
Python
Visual Basic .NET
etc...
Scripting Programming Language
Javascript
PHP
Perl
Ruby
etc...
Logic Programming Language
PROLOG
Datalog
Absys
ALF
etc...


Programming Pardigms
Imperative Paradigm
Delcarative Programming Paradigms
Procedural Programming Pardigms
Object Oriented Programming
Parallel Processing Approach
Logic Programming Pardigm
Functional Programming
Database Processing Approach

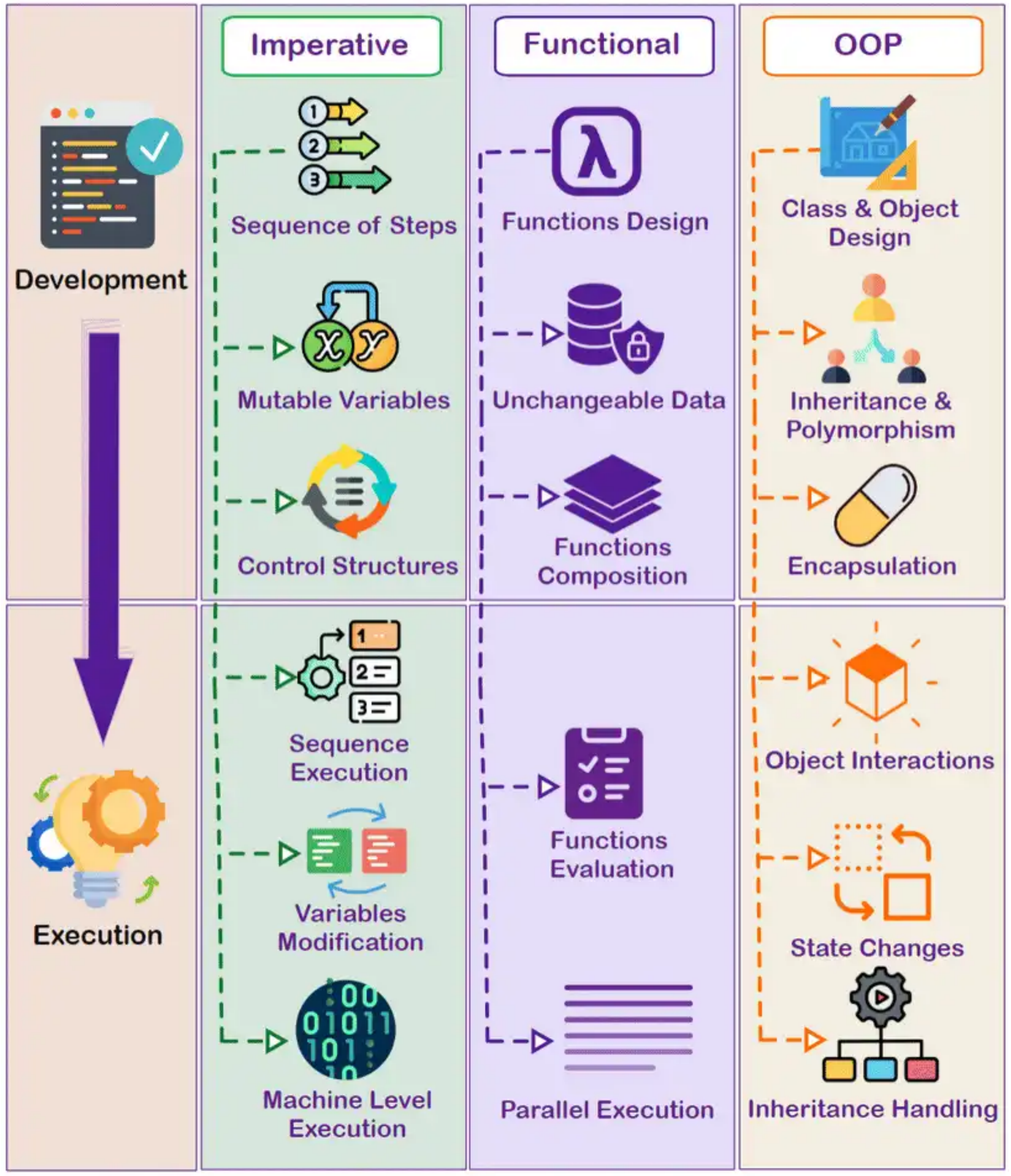

Imperative
Tell how to do
Declarative
Tell what to do

```swift
// Imperative Programming

let array = [1, 2, 3, 4, 5, 6]

var evenNumbers: [Int] = []

for i in 0..<array.count {
    if array[i] % 2 == 0 {
        evenNumbers.append(array[i])
    }
}
```

```swift
// Declarative

let evenNumbers2 = array.filter { $0 % 2 == 0 }
```

Imperative vs Functional vs Object-oriented PROGRAMMING

blog.bytebytego.com



Feature	Functional Programming	Object-Oriented Programming
Primary focus	Functions	Objects
Data	Immutable	Mutable
Programming model	Declarative	Imperative
Parallel programming	Well-supported	Not as well-supported

Export to Sheets

OO pattern/principle	FP pattern/principle
• Single Responsibility Principle	• Functions
• Open/Closed principle	• Functions
• Dependency Inversion Principle	• Functions, also
• Interface Segregation Principle	• Functions
• Factory pattern	• Yes, functions
• Strategy pattern	• Oh my, functions again!
• Decorator pattern	• Functions
• Visitor pattern	• Functions □

FUNCTIONS

TOTAL
(NOT PARTIAL)

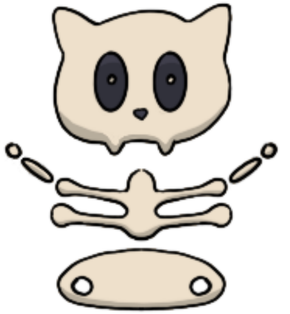
DETERMINISTIC
(NO RANDOMNESS)

PURE
(NO SIDE EFFECT)

NO MUTATION

NO NULL

NO REFLECTION



NO EXCEPTION

Totalité

```
def validate(user: User): String = {
  val trimmed = user.name.trim
  if(trimmed.isEmpty) throw new Exception("Name is empty!")
  else trimmed
}
```

```
def validate(user: User): Option[String] =
  user.name.trim match {
    case "" => None
    case trimmed => Some(trimmed)
  }
```

toujours renvoyer un résultat “exploitable”, peu importe les paramètres fournis

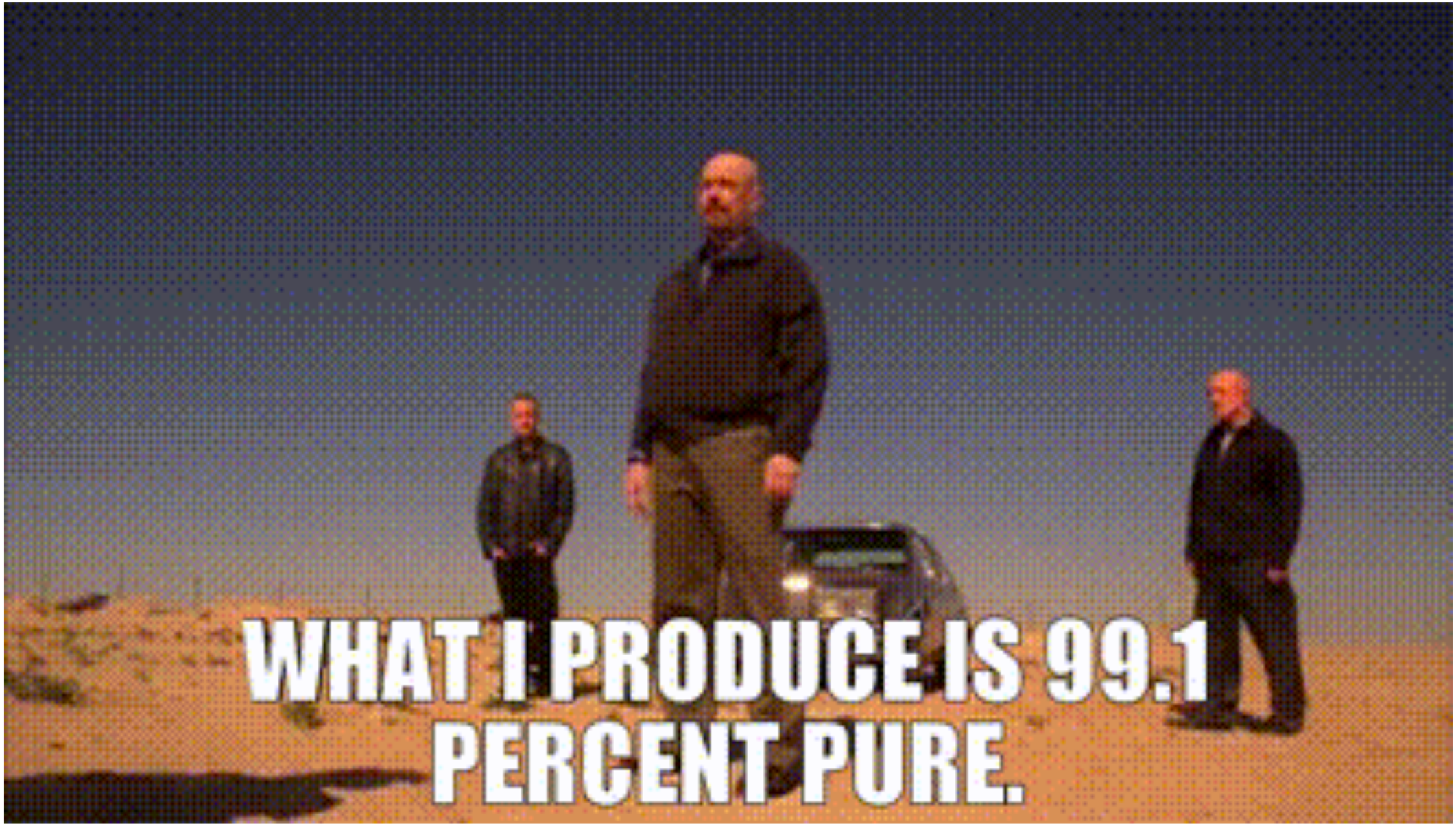
Déterminisme

```
def foo(): Int = Random.nextInt()
```

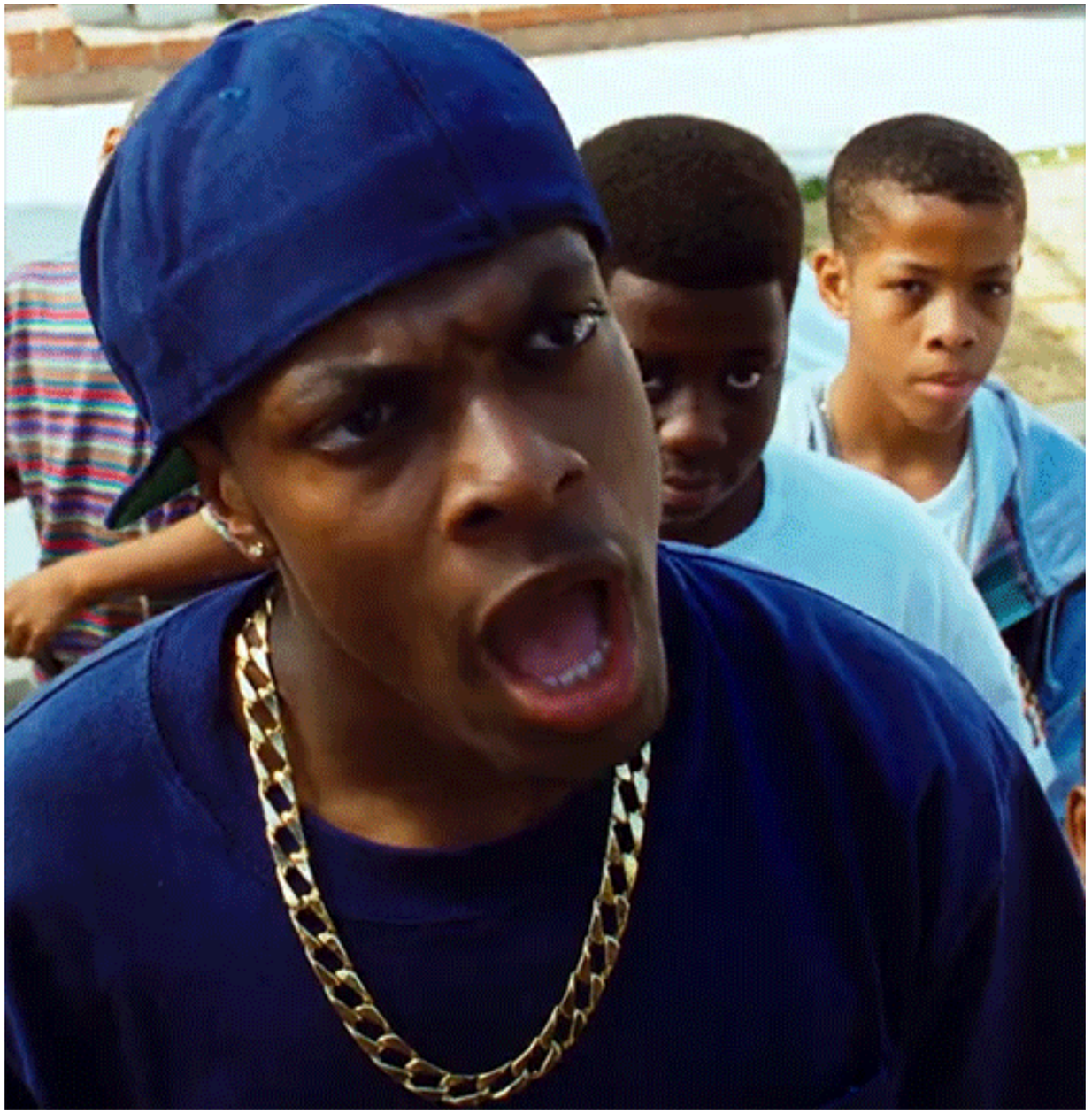
```
def foo(seed: Int): Int = new Random(seed).nextInt()
```

renvoyer à chaque fois le même résultat pour les mêmes paramètres

Pureté



Une fonction est pure si elle respecte la transparence référentielle



Transparence référentielle: une expression (comme un appel à une fonction) peut être remplacée par son résultat sans changer le comportement du programme

```scala
var x = 5
def incrementX(): Int = {
  x += 1
  x
}
```

pas TR car le résultat dépend de la

variable x extérieure à la fonction

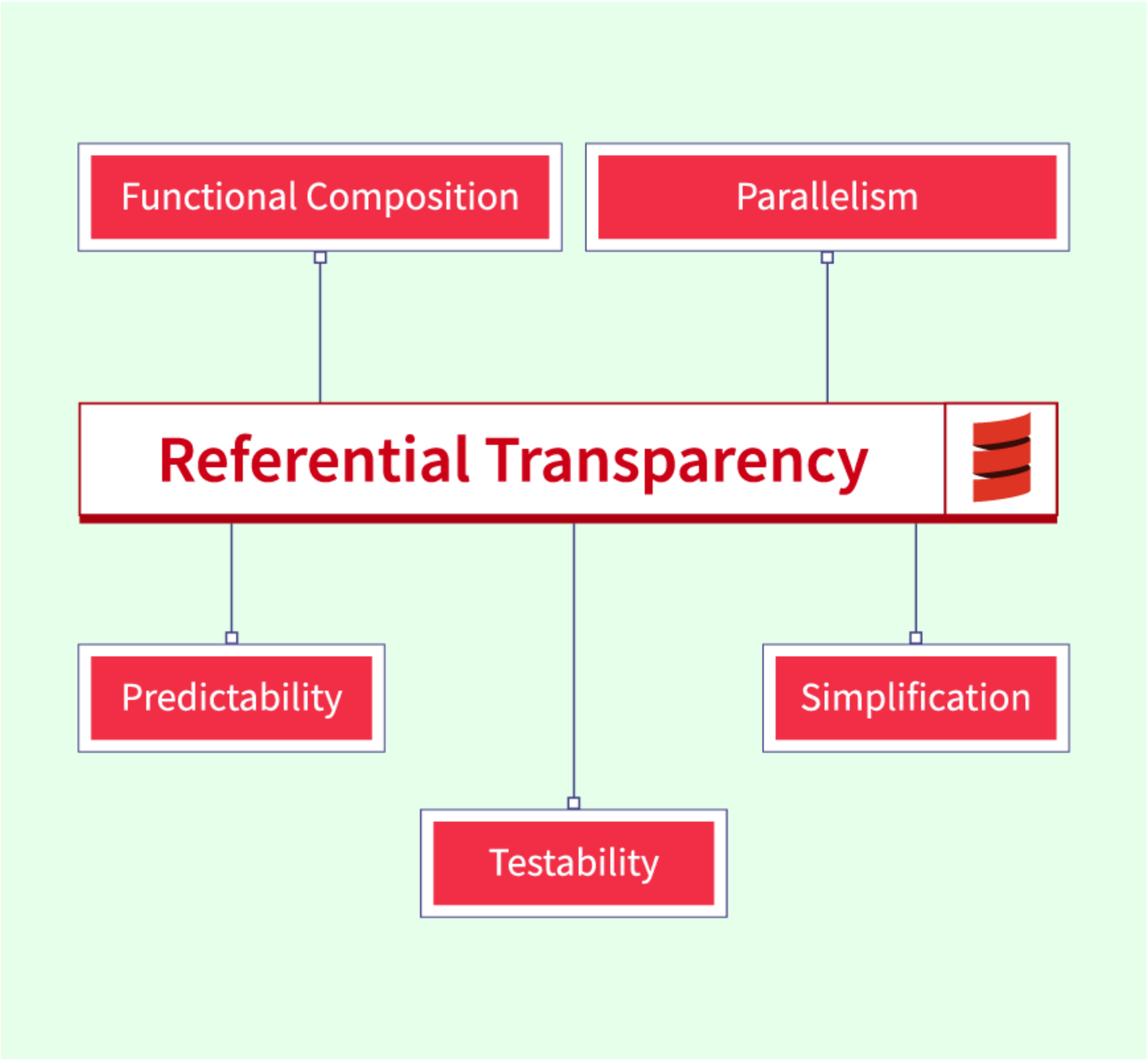
=> mutabilité

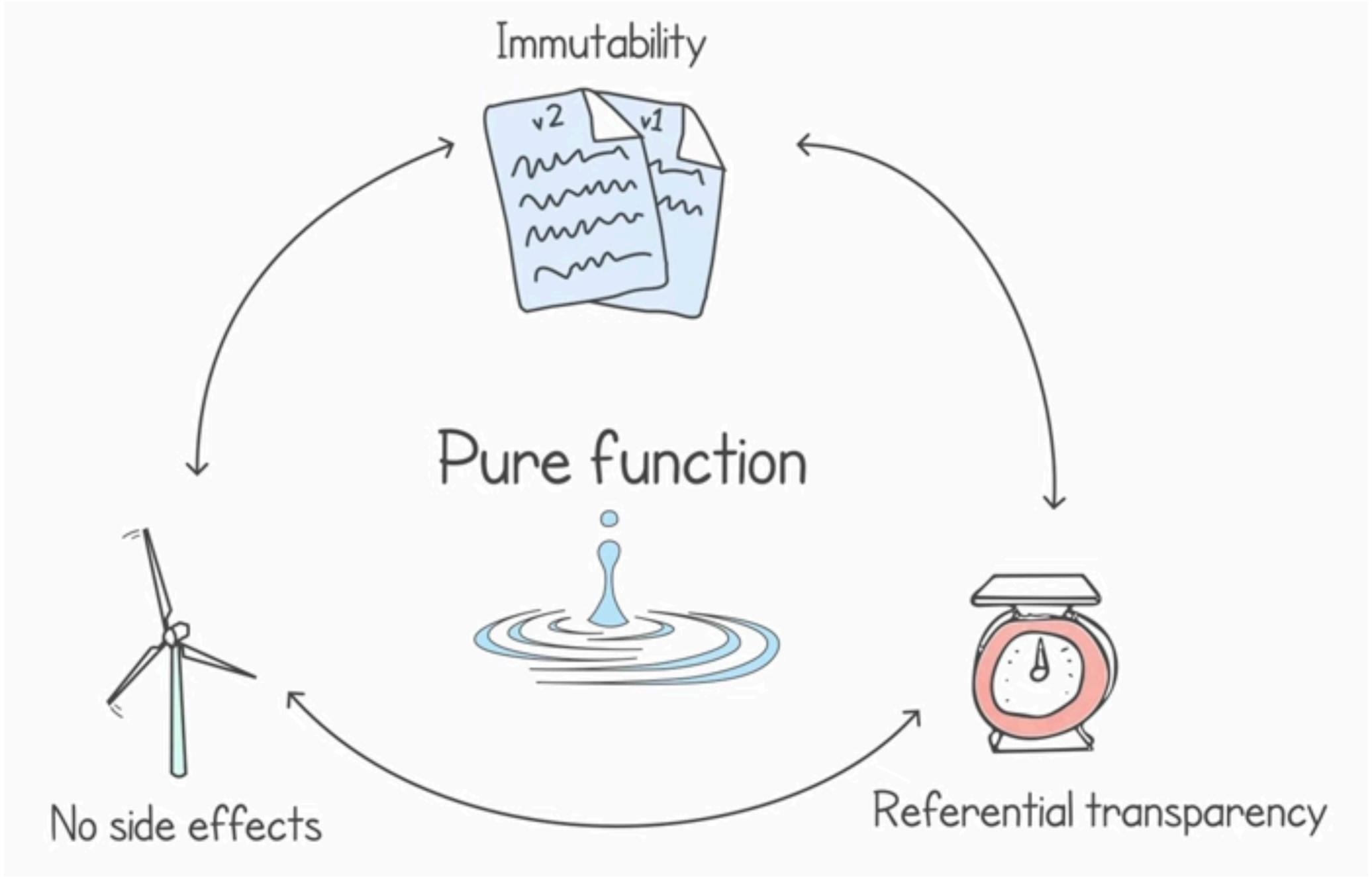
```scala
def printAndIncrement(x: Int): Int = {
  println(s"Value: $x")
  x + 1
}
```

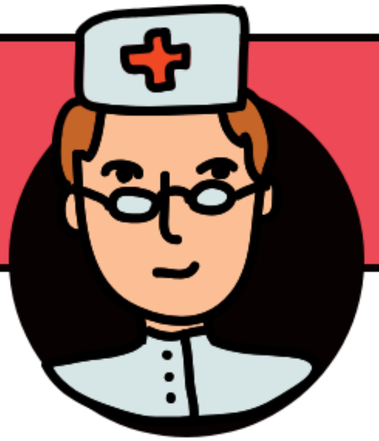
pas TR car affichage

en console

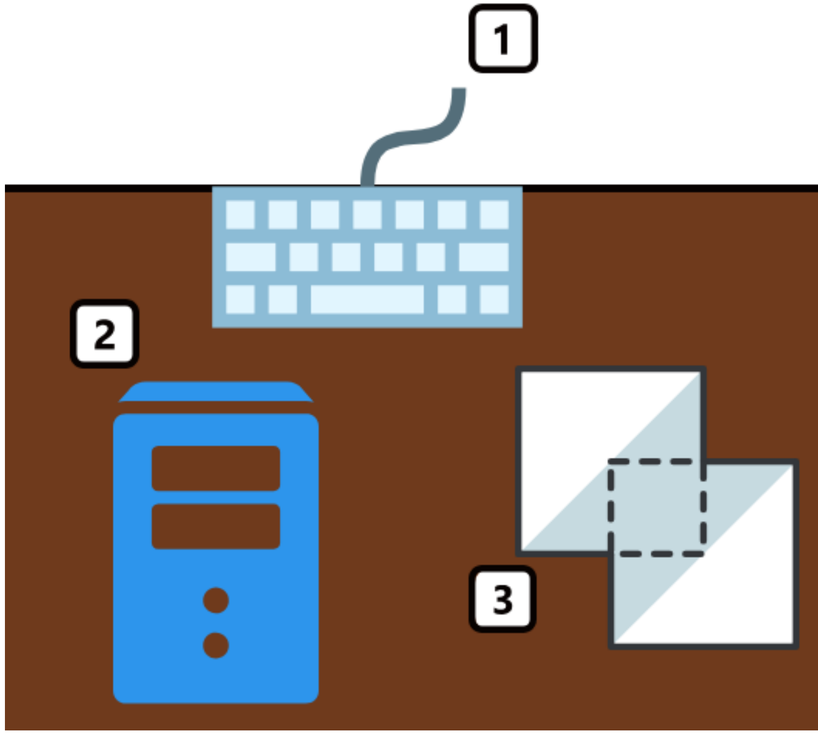
=> effet de bord


Functional Composition
Parallelism
Referential Transparency
Predictability
Testability
Simplification


Immutability
v2
v1
Pure function
No side effects
Referential transparency



3 WAYS TO AVOID SIDE-EFFECTS



1 Leave Your Input Alone

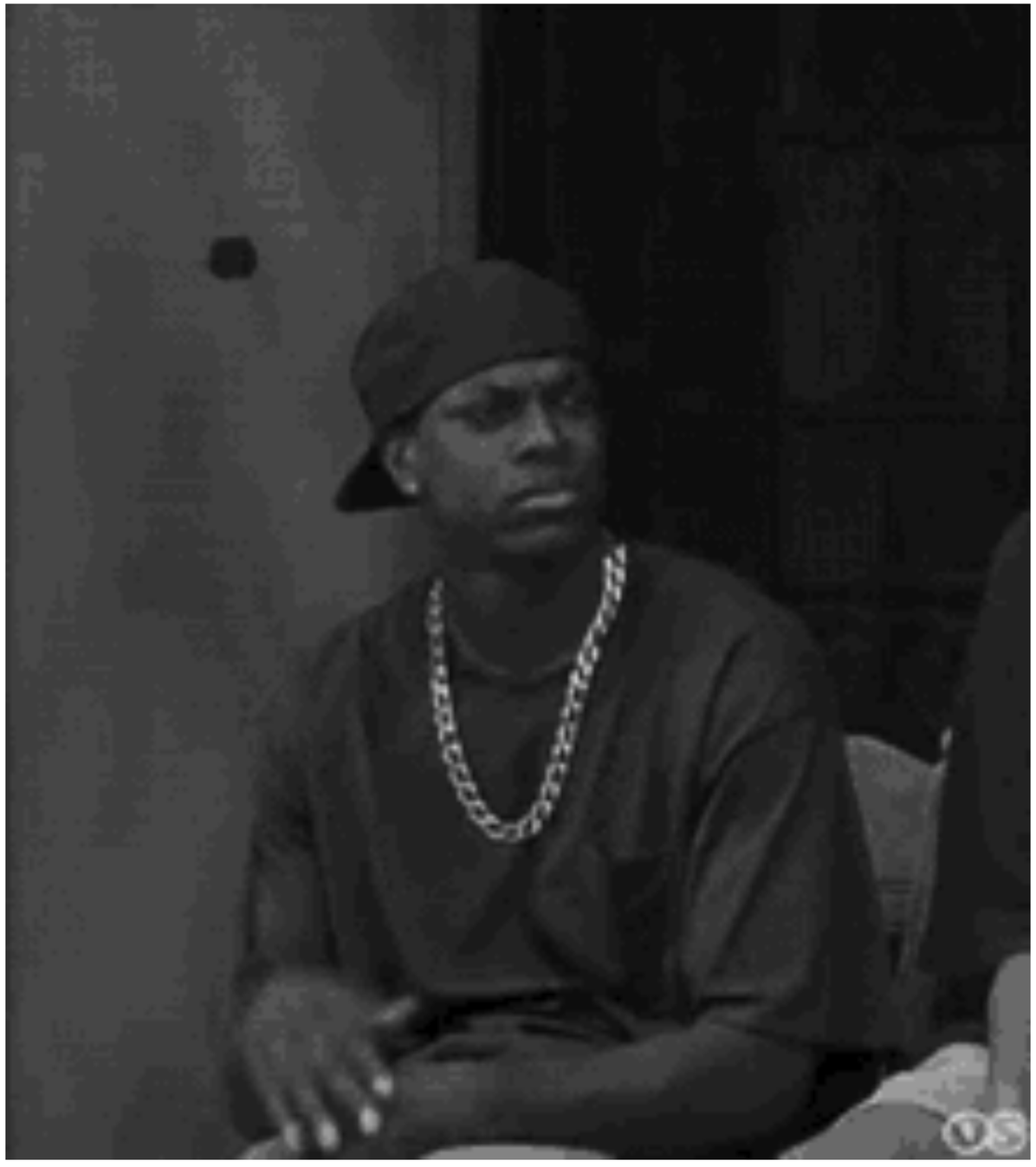
Changing your function's input can lead to unexpected behaviors in a large system. You never know when something changed, or who did it. Instead, consider **cloning your input** and modifying *that*.

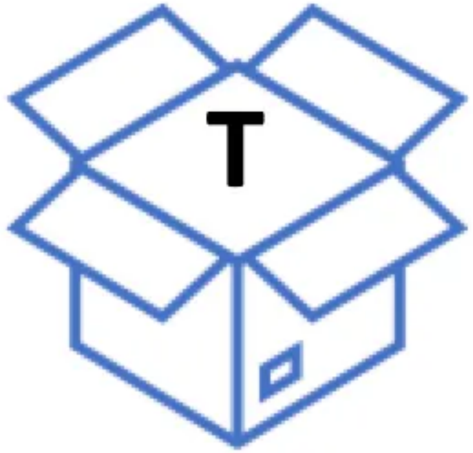
2 Don't Make HTTP Calls Or Write To Disk

Mixing data manipulation logic with network/filesystem calls hurts testability and parallel processing. Encapsulate those effects in their own modules.

3 Be Transparent

"Referential transparency" means your function can be substituted with its output and not affect your program's behavior. If a function has side-effects, swapping it out can break the program.





Option[T]

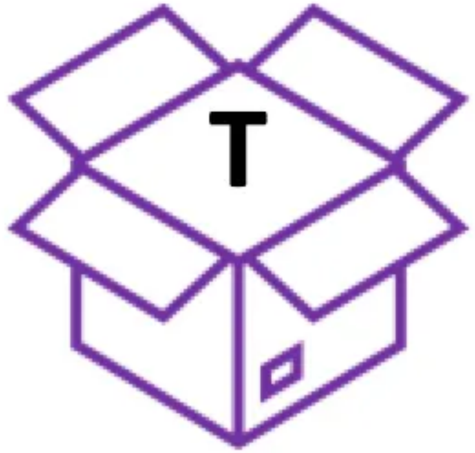
Some[T] or

None

Try[T]

Success[T] or

Failure[E]



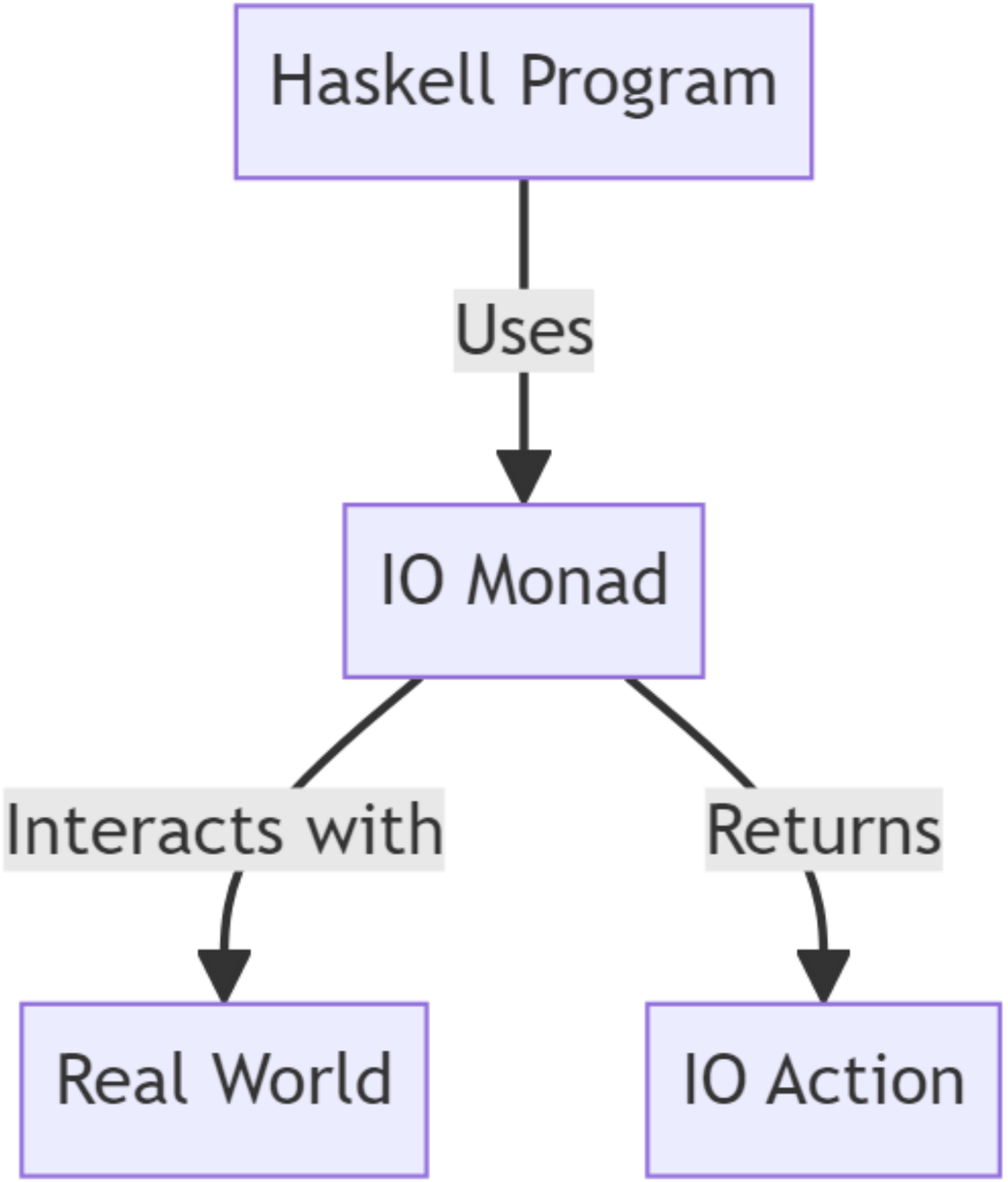
Either[S,T]

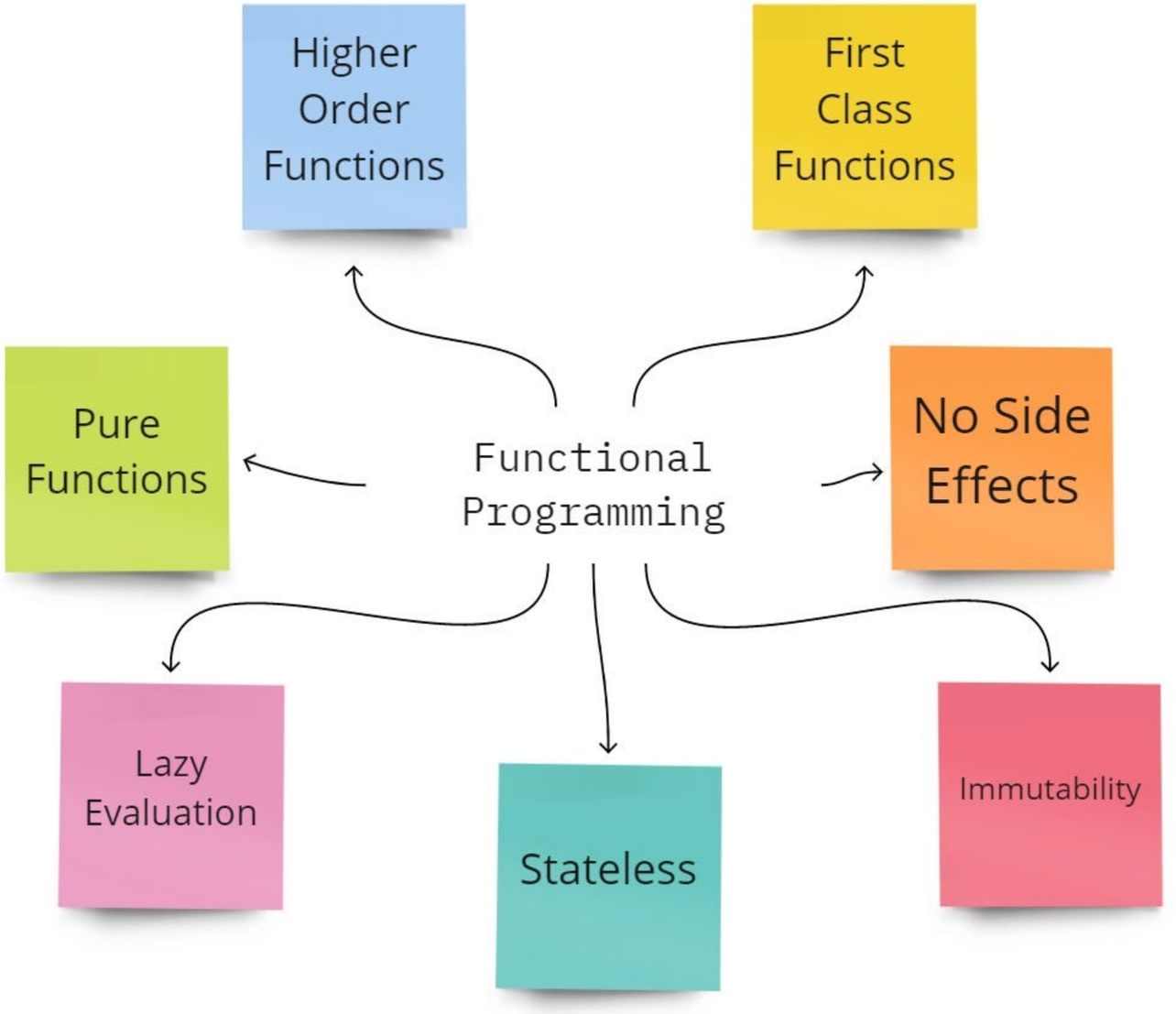
Right[T] or

Left[S]

```scala
def foo: IO[Int] = IO { Random.nextInt() }
```

TR car IO n’exécute pas l’action, mais renvoie juste une description de celle-ci. L’action est décalée à plus tard


Haskell Program
Uses
IO Monad
Interacts with
Returns
Real World
IO Action


Higher Order Functions
First Class Functions
Pure Functions
Functional Programming
No Side Effects
Lazy Evaluation
Stateless
Immutability


ŠKODA
SCALA
CZ 4SM 0466

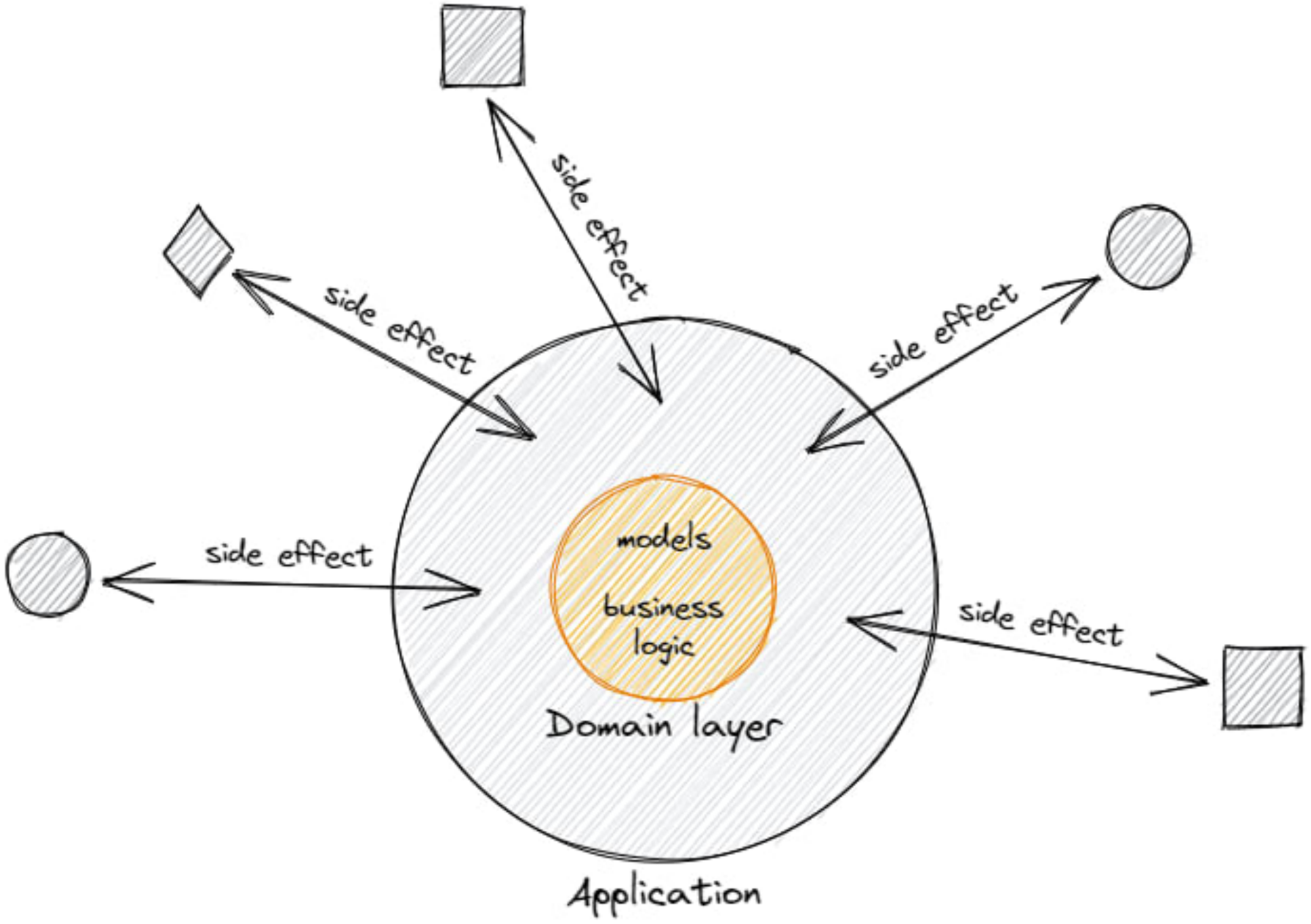

Object-Oriented
Programming
Functional
Programming

Scala:

- multi-paradigme (objet et fonctionnel)
- statiquement typé
- basé sur la JVM


Alright team. We need to decide which approach we will use to build our latest app.
Functional Programming!
OOP!
Why not apply both when necessary?
imgflip.com

1. **Encapsulation et transparence référentielle:**
   avoir des classes avec des champs immuables
2. **Des méthodes pures dans les classes:**
   avoir des classes avec des méthodes qui sont des fonctions pures
3. **Patterns de la programmation fonctionnelle:**
   possibilité d'utiliser des fonctions de 1er ordre, du pattern matching et de l'immutabilité Scala d'obtenir du code orienté object et respectant la transparence référentielle
4. **Interaction avec du code impur:**
   encapsuler le code impur dans une monade (type IO), et les exécuter le plus tard possible aux bornes du programme


side effect
side effect
side effect
side effect
side effect
side effect
models
business
logic
Domain layer
Application