CO AND CONTRA VARIANCE: A WAR OF CONQUEST

Sebastián Ortega

jobandtalent

@_sortega

JARGON

> “A monad is just a monoid in the category of endofunctors, what's the problem?”

–James Iry, in his blog

“A monad is just a monoid in the category of endofunctors, what's the problem?”

–James Iry, in his blog

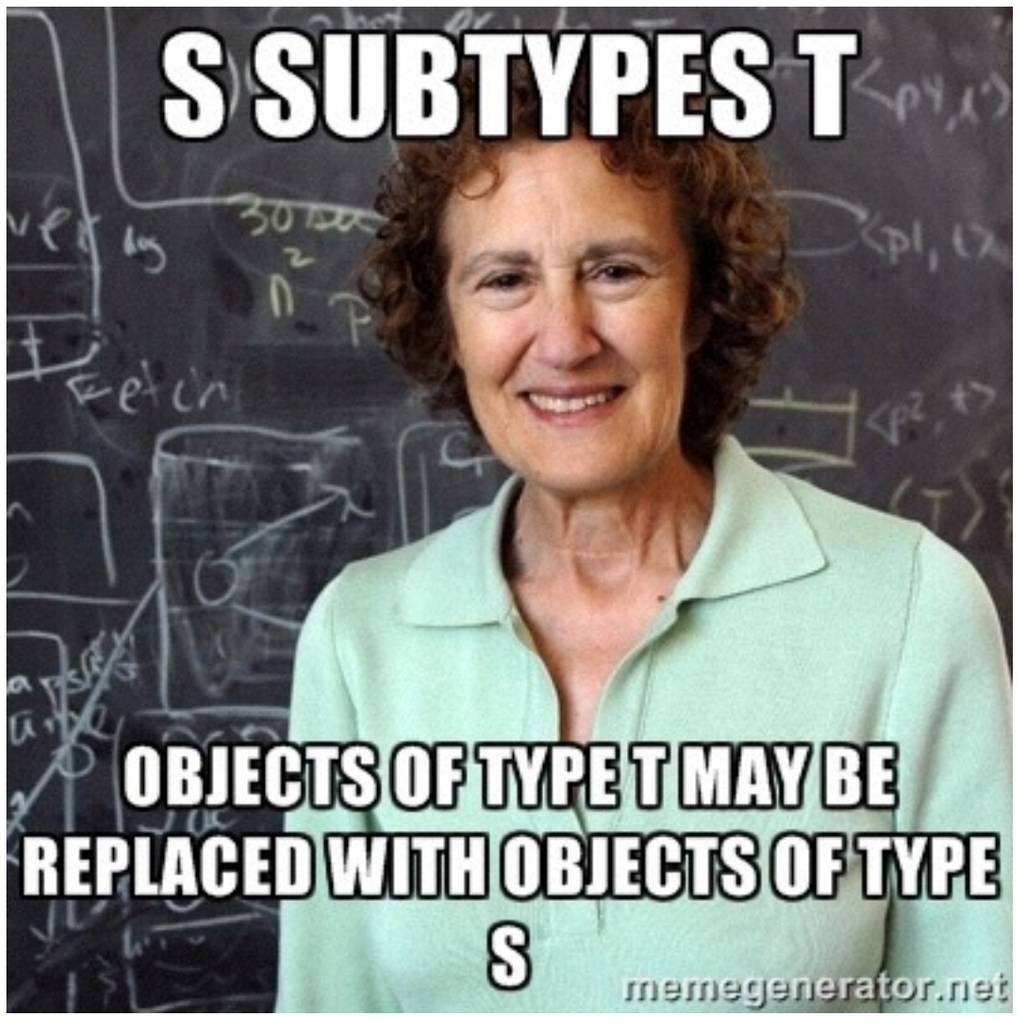
> By making type constructors covariant or contravariant instead of invariant, more programs will be accepted as well-typed.

SOLID PRINCIPLES

- Single responsibility principle
- Open-closed principle
- Liskov substitution principle
- Interface-segregation principle
- Dependency injection principle

SOLID PRINCIPLES

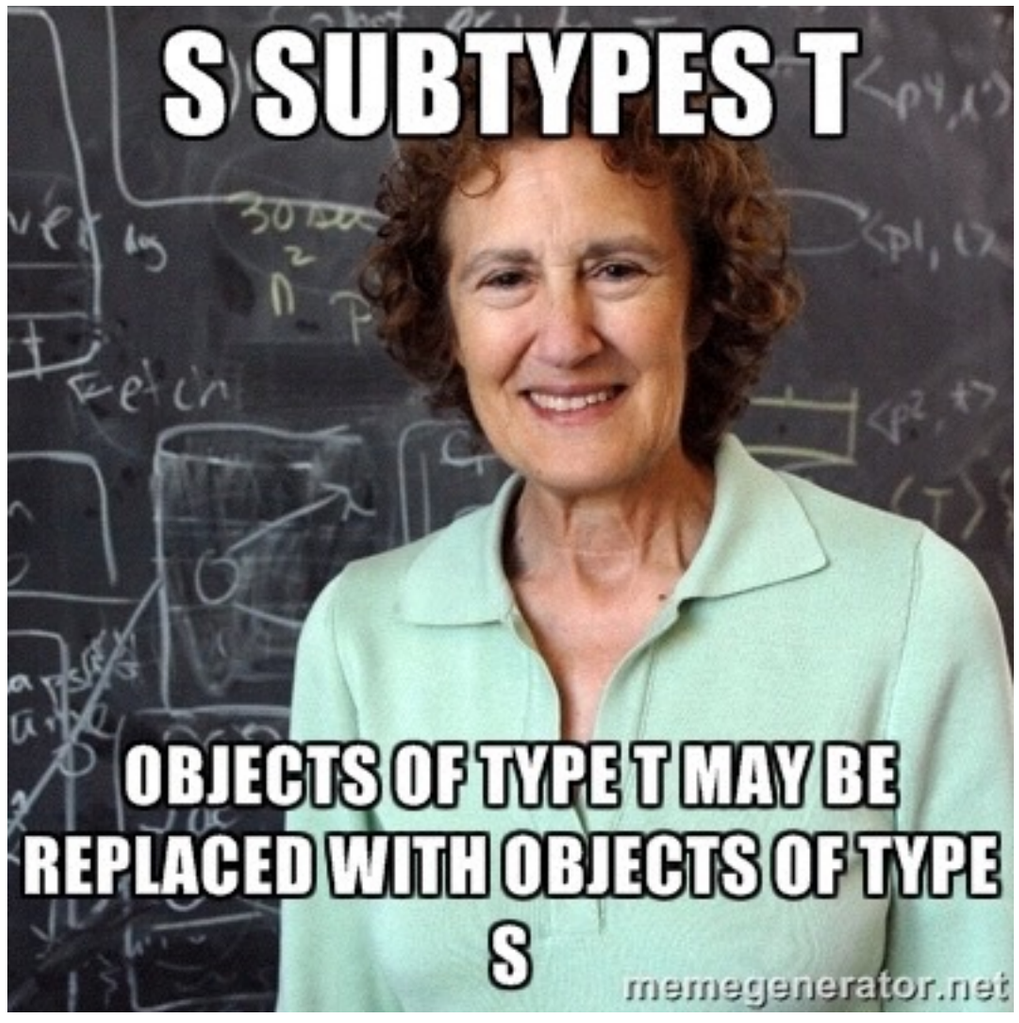
- Single responsibility principle
- Open-closed principle
- Liskov substitution principle
- Interface-segregation principle
- Dependency injection principle


S SUBTYPES T
OBJECTS OF TYPE T MAY BE REPLACED WITH OBJECTS OF TYPE S
memegenerator.net

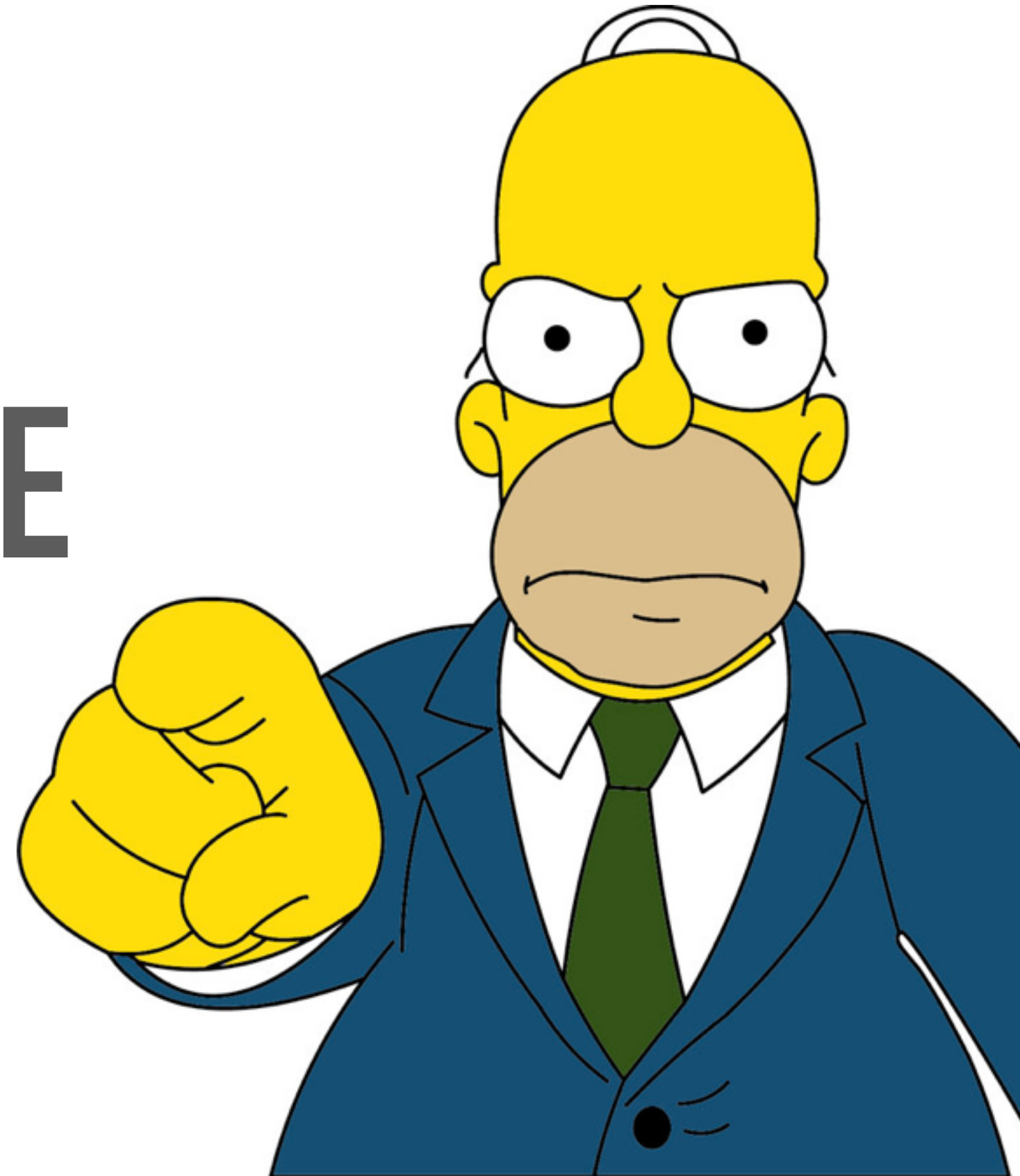
WHAT IT TAKES TO BE A SUBTYPE?

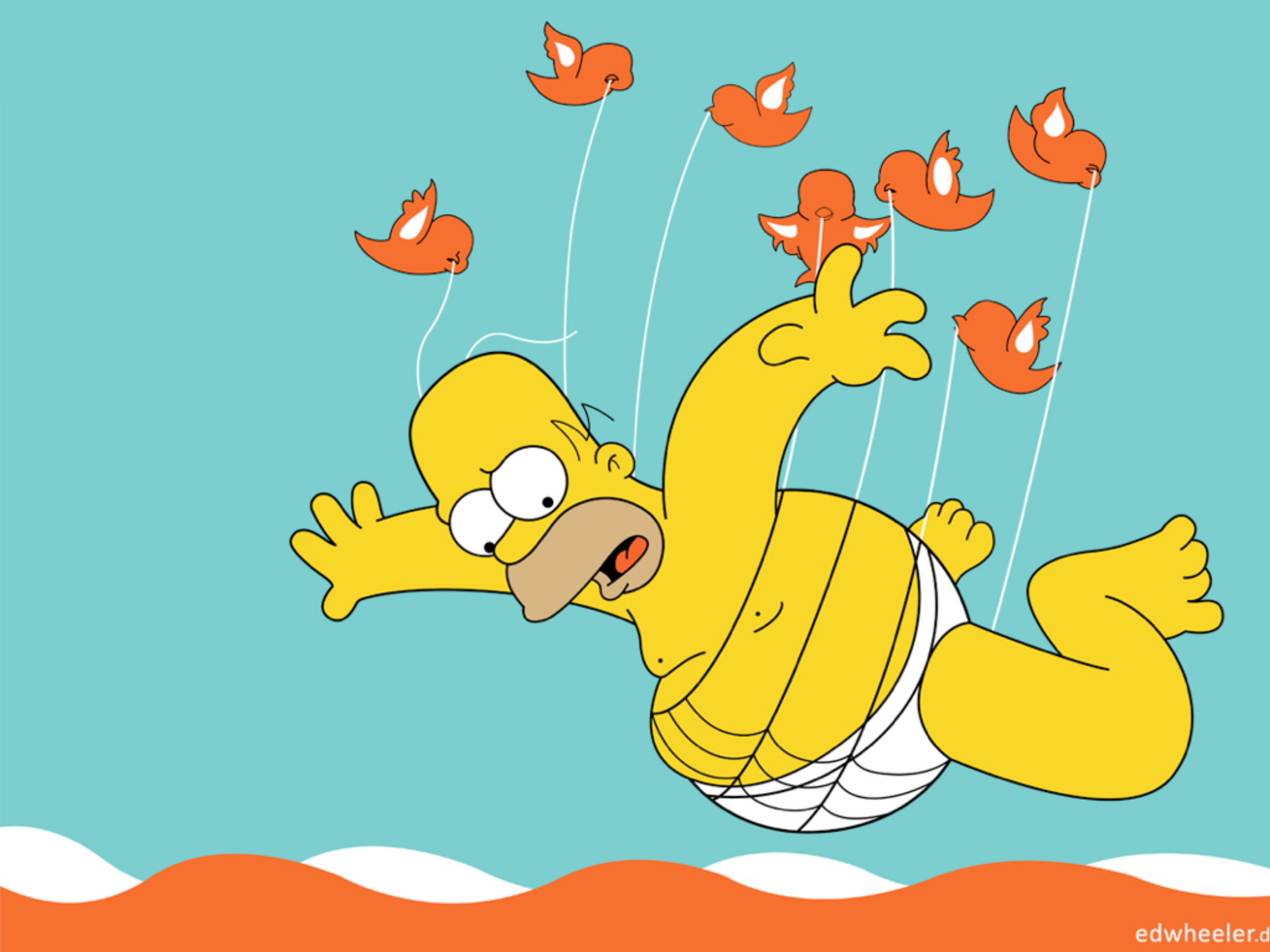
"Let $\Phi(x)$ be a property provable about objects x of type T. Then $\Phi(y)$ should be true for objects y of type S where S is a subtype of T."

– Barbara Liskov

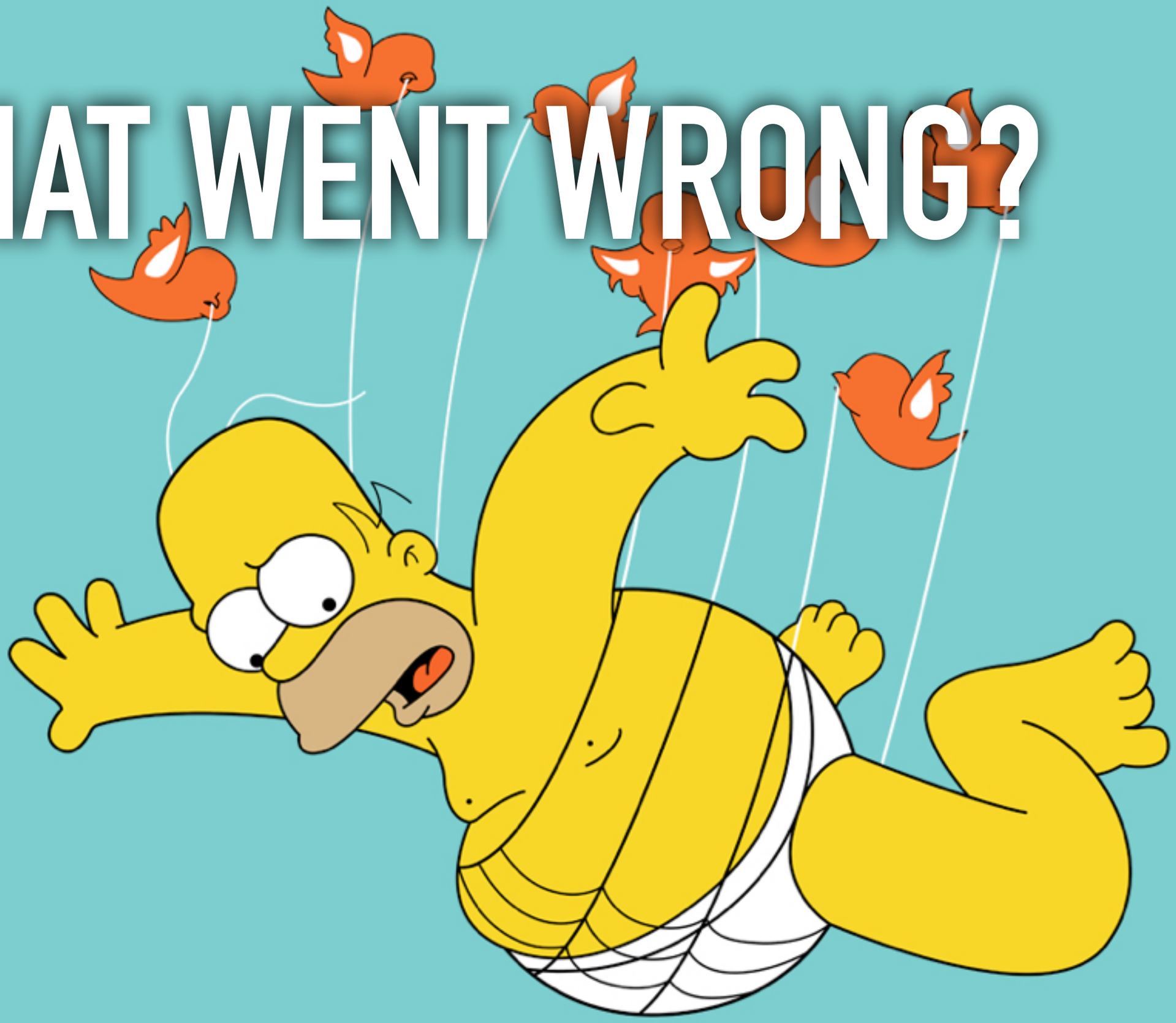


SHOW ME SOME CODE




edwheeler.d

WHAT WENT WRONG?



- “IS A” in the real world is about “being”
  - indeed, a square is a rectangle

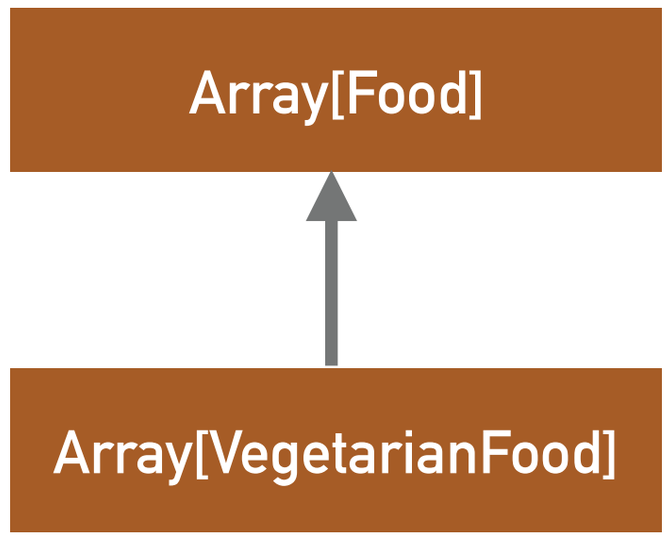
- “IS A” in the real world is about “being”
  - indeed, a square is a rectangle
- “IS A” in our code is about “behaving like”
  - a mutable square doesn’t behave like a rectangle

“

“To err is to be human but compilers never forget.”

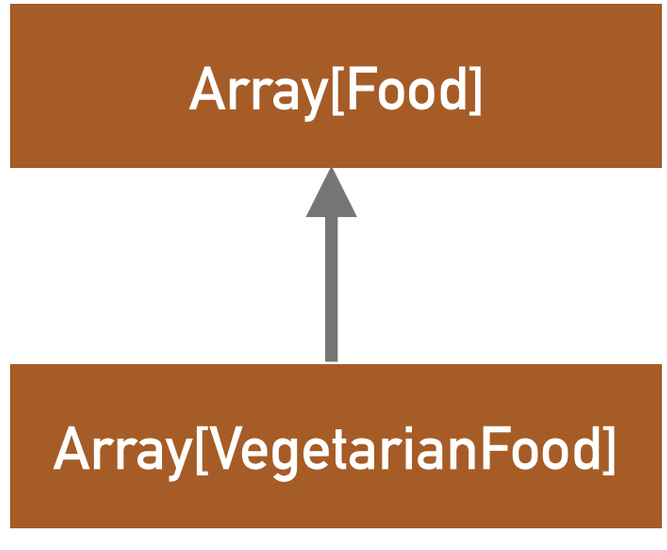
–The Rust Book

IS ARRAY[VEGETARIANFOOD] A SUBTYPE OF ARRAY[FOOD]?



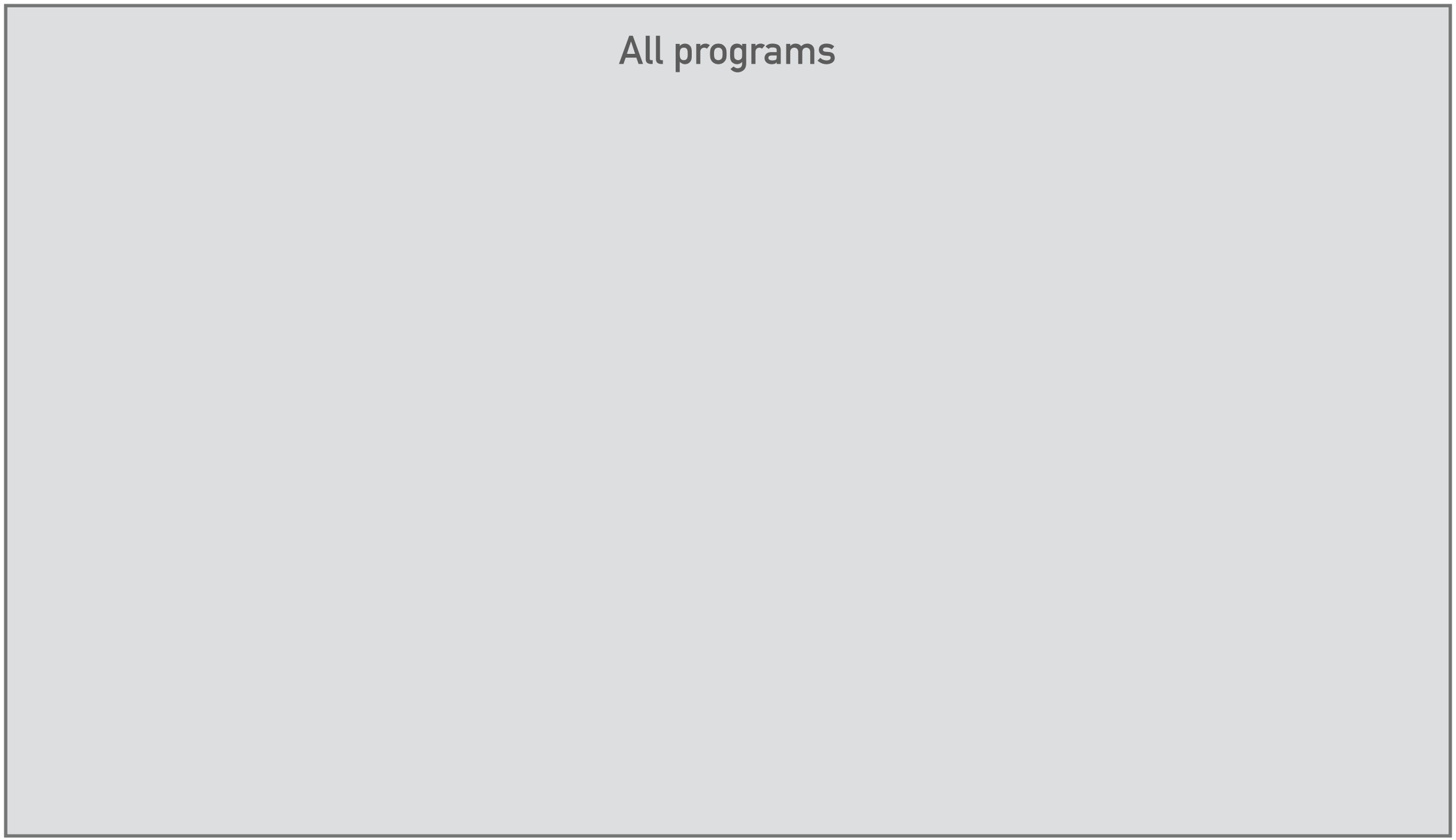
IS ARRAY[VEGETARIANFOOD] A SUBTYPE OF ARRAY[FOOD]?

NOPE

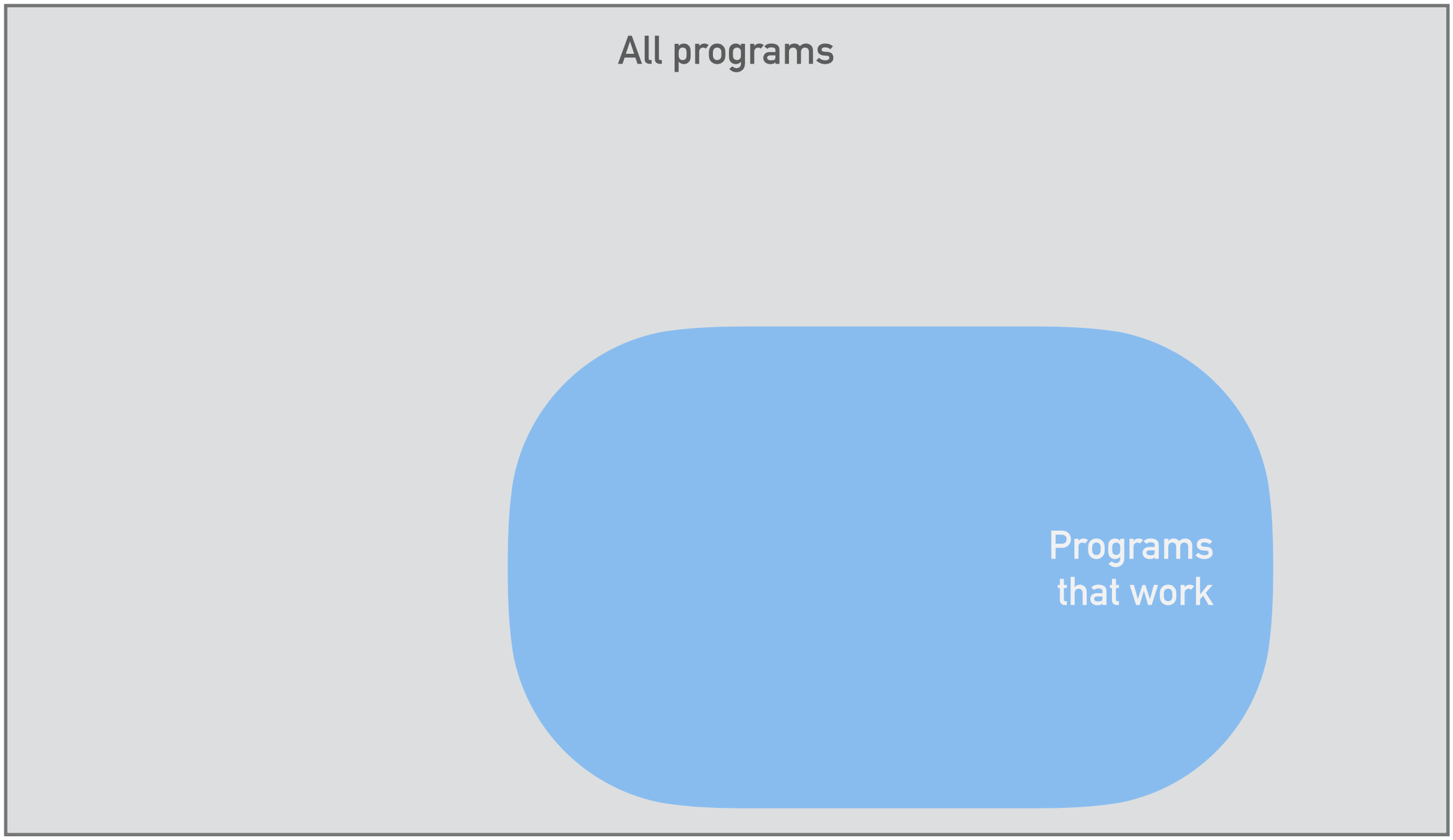


WHY?

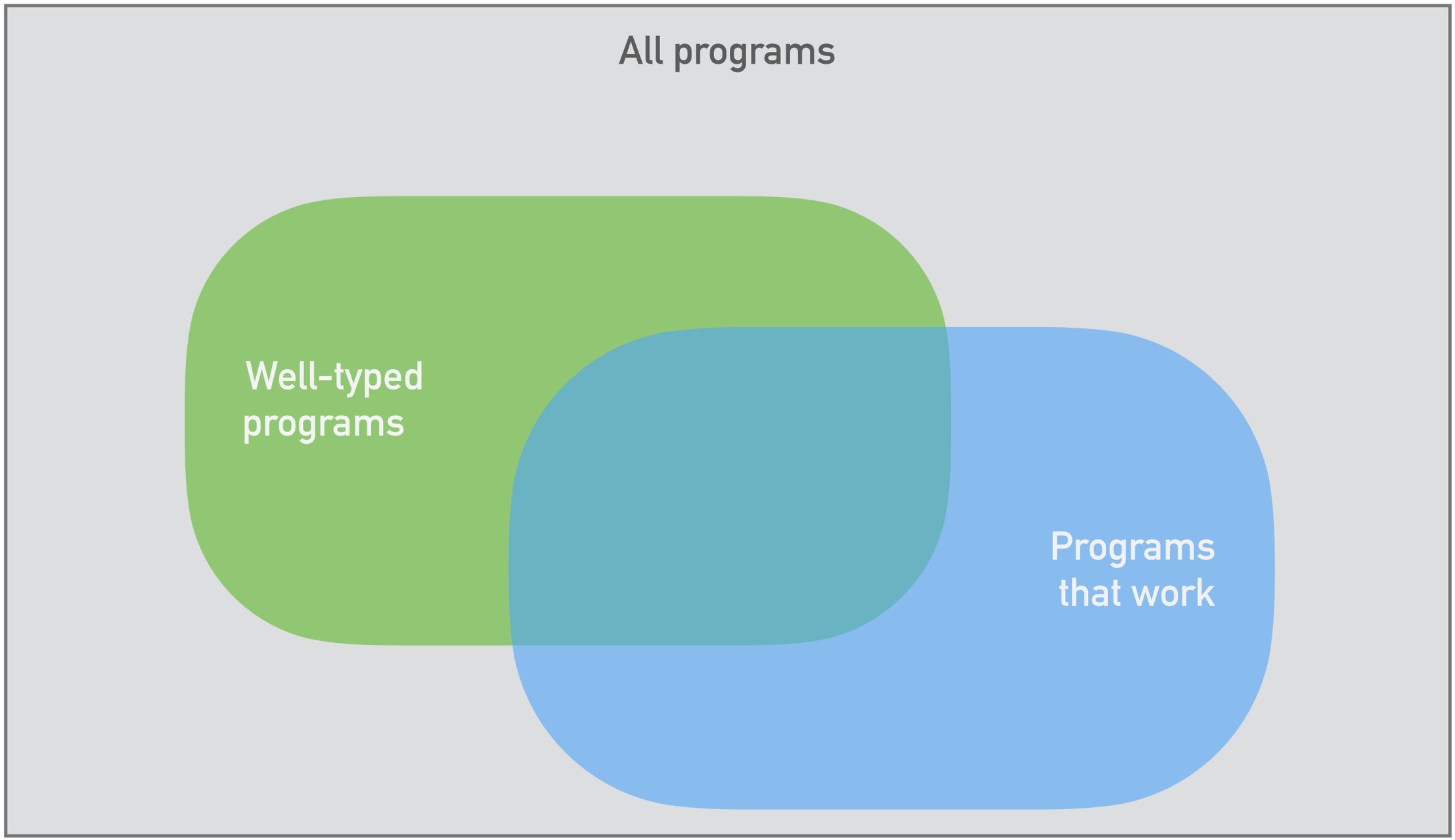
TYPE SYSTEM



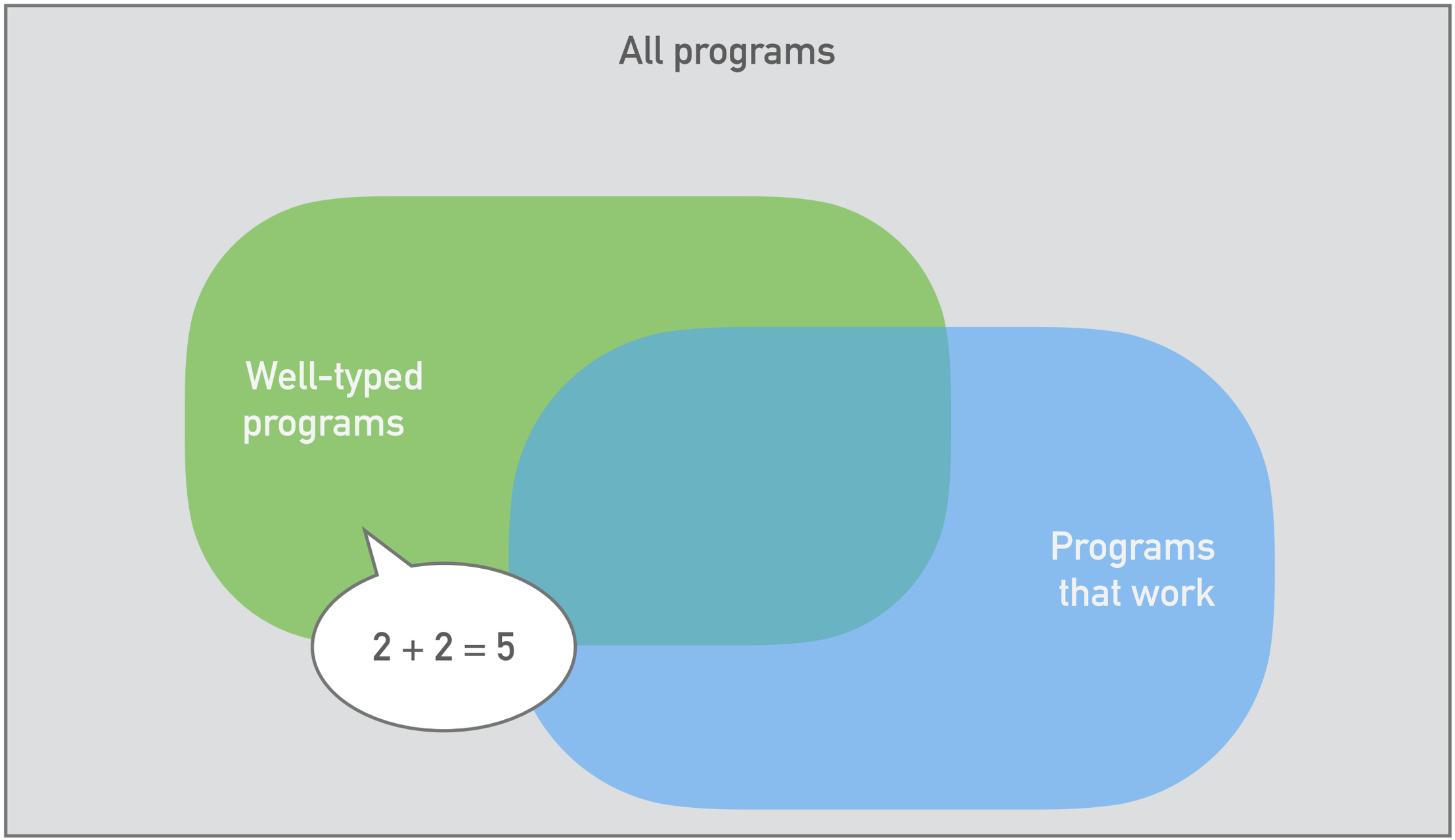
TYPE SYSTEM



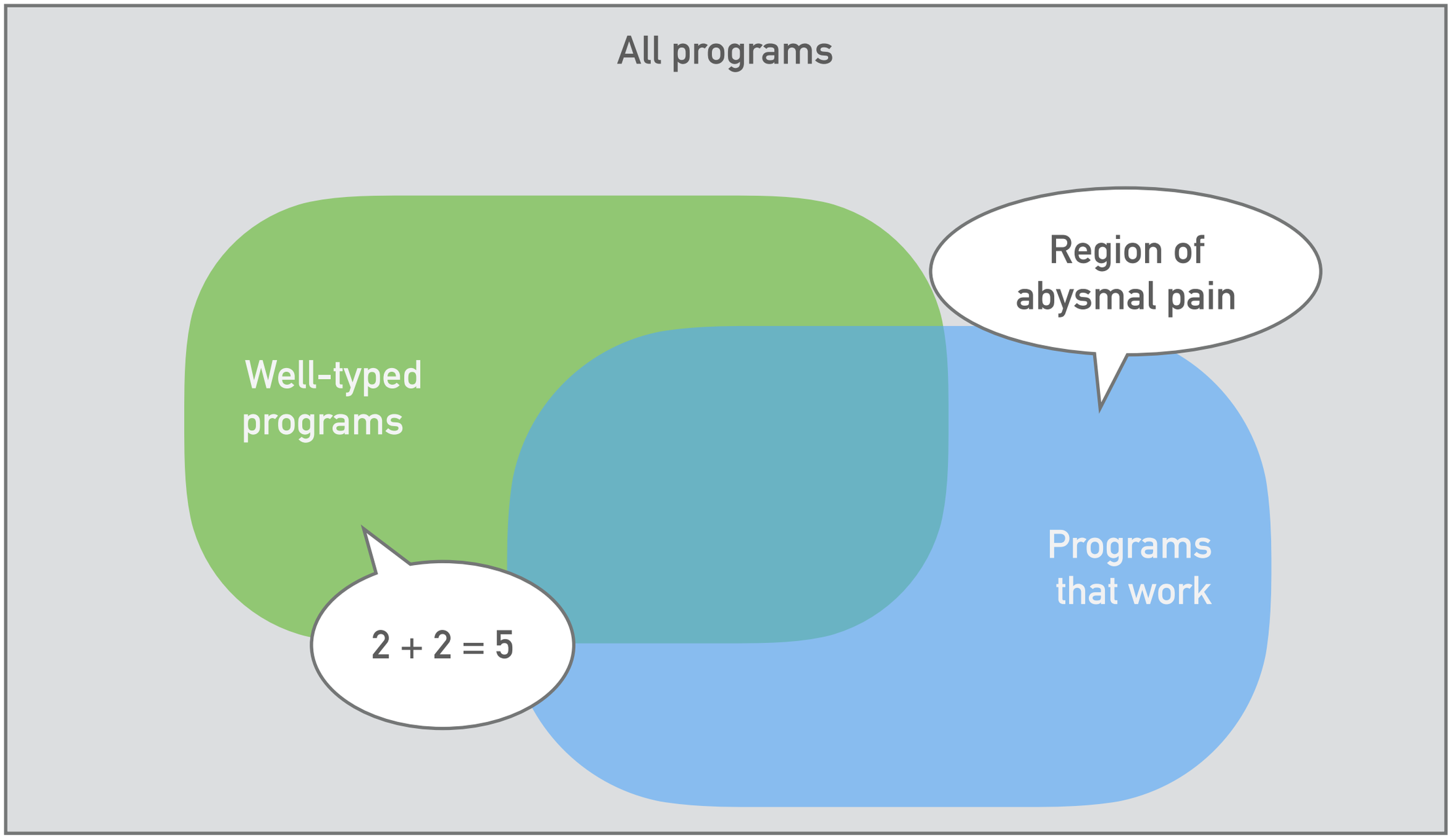
TYPE SYSTEM



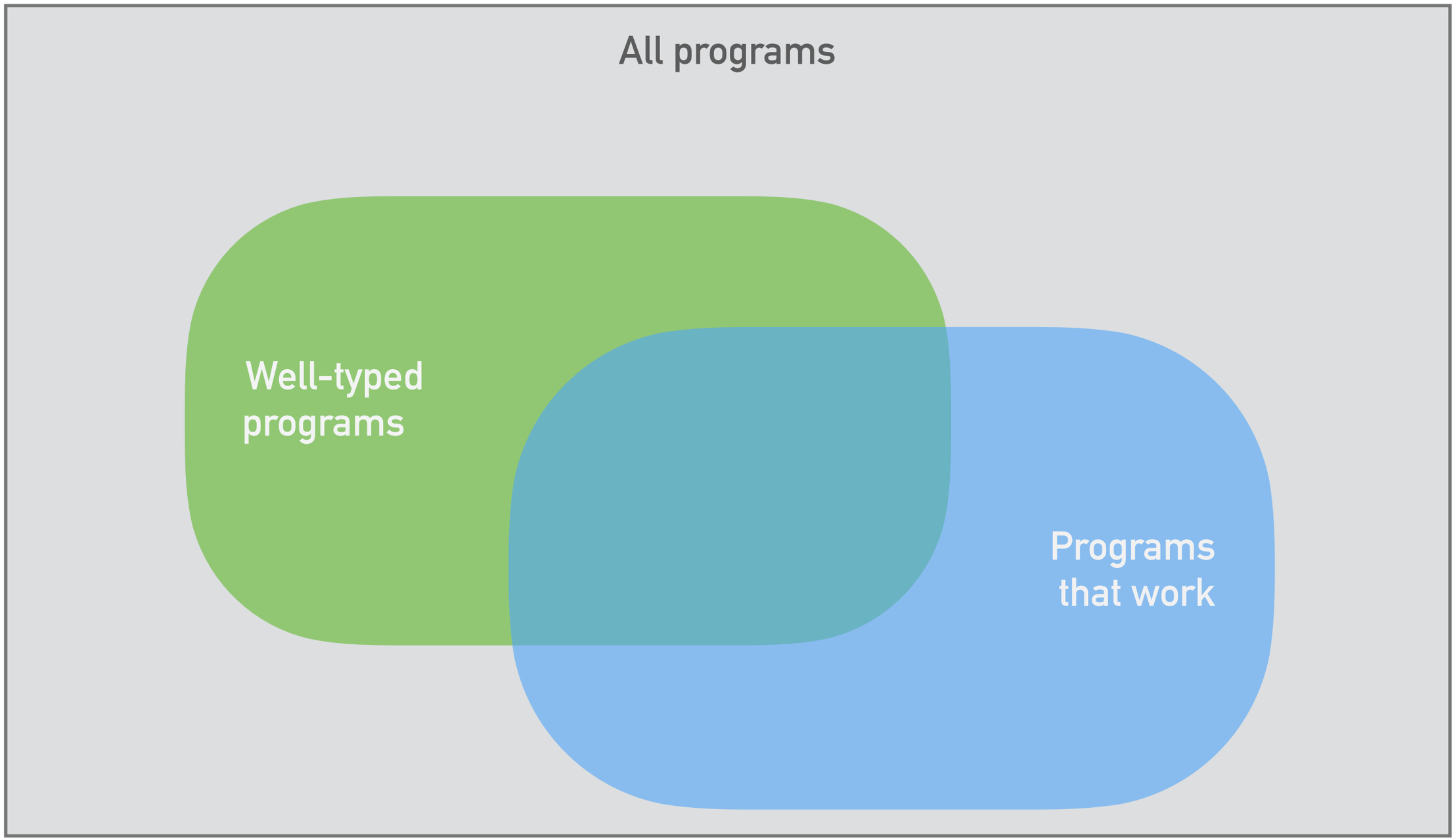
TYPE SYSTEM



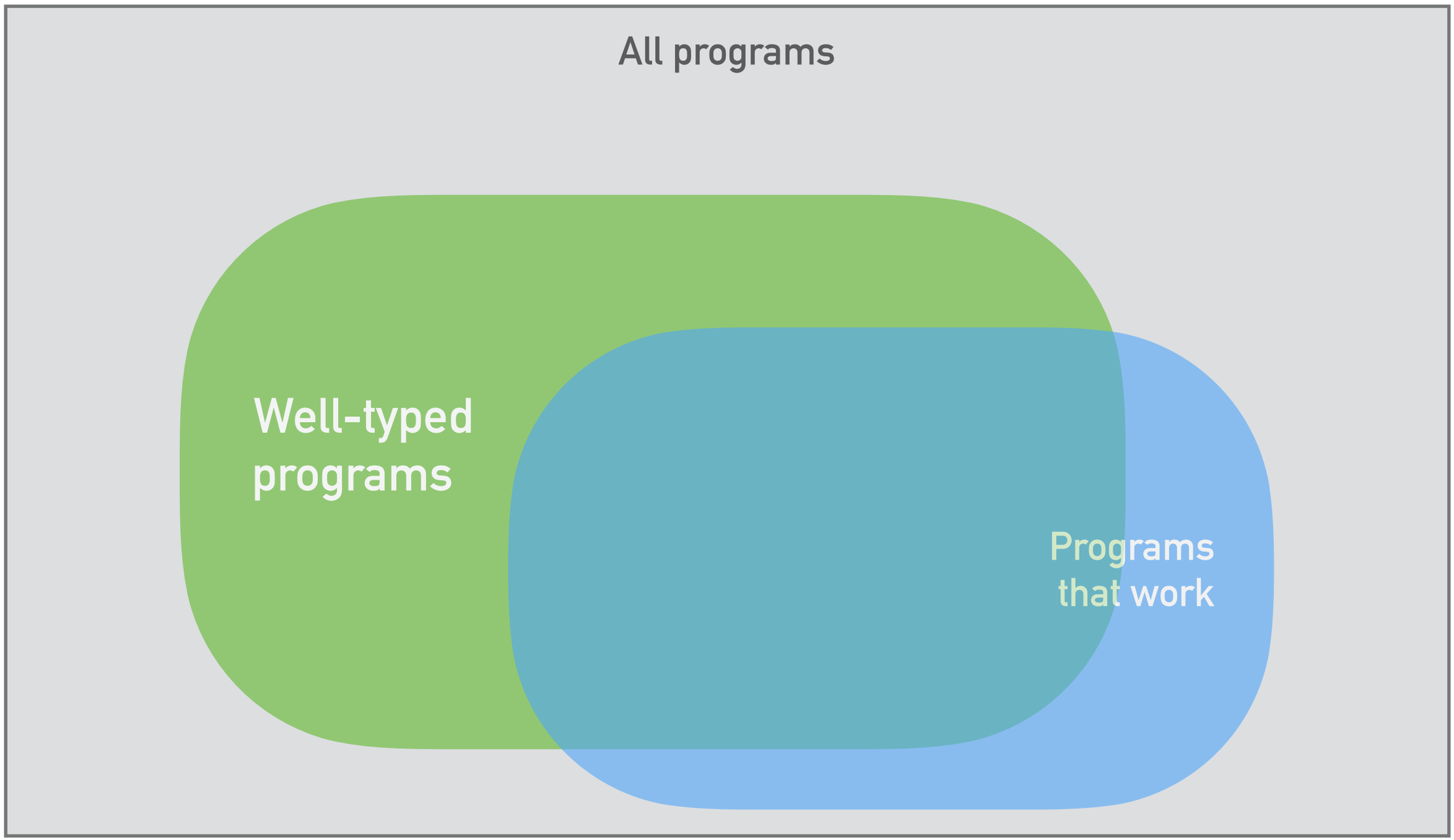
TYPE SYSTEM


All programs
Region of abysmal pain
Well-typed programs
Programs that work
2 + 2 = 5

BETTER TYPE SYSTEM



BETTER TYPE SYSTEM



WHAT'S IN A CONTRACT?

WHAT'S IN A CONTRACT?

Precondition	OK	Compiler friendly
Object has been initialized	y	y
Object has some property	y	n
Input arguments have some type	y	y
Input arguments have some property	y	n
Global state has some property	n	n

WHAT'S IN A CONTRACT?

Postcondition	OK	Compiler friendly
Returned value has a type	y	y
Returned value has a property	y	n
Object state has a property	y	n
Global state has some property	n	n

WHAT'S IN A CONTRACT? (II)

WHAT'S IN A CONTRACT? (II)

- We **must** respect the contract

WHAT'S IN A CONTRACT? (II)

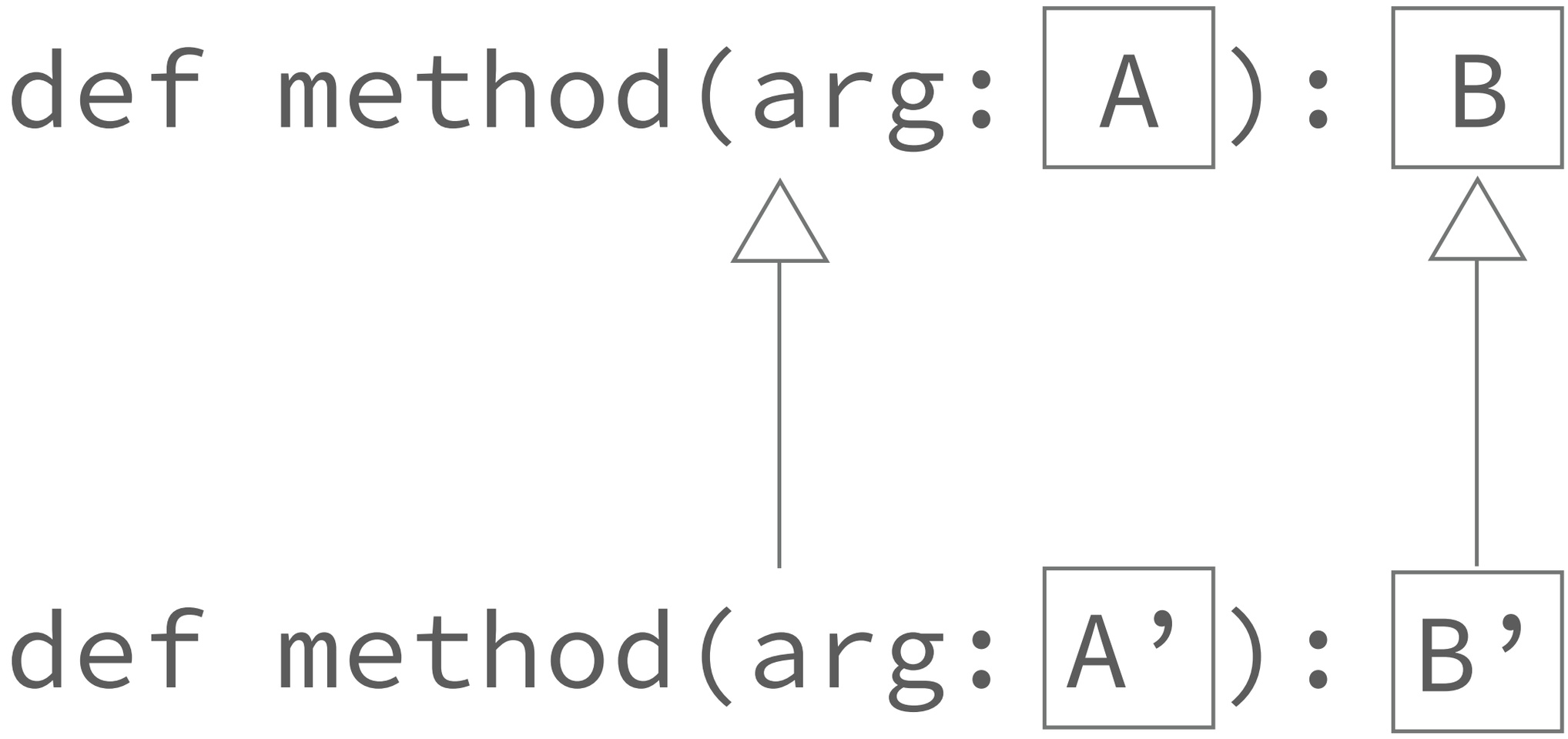
- We **must** respect the contract
- We can **constrain** the postconditions

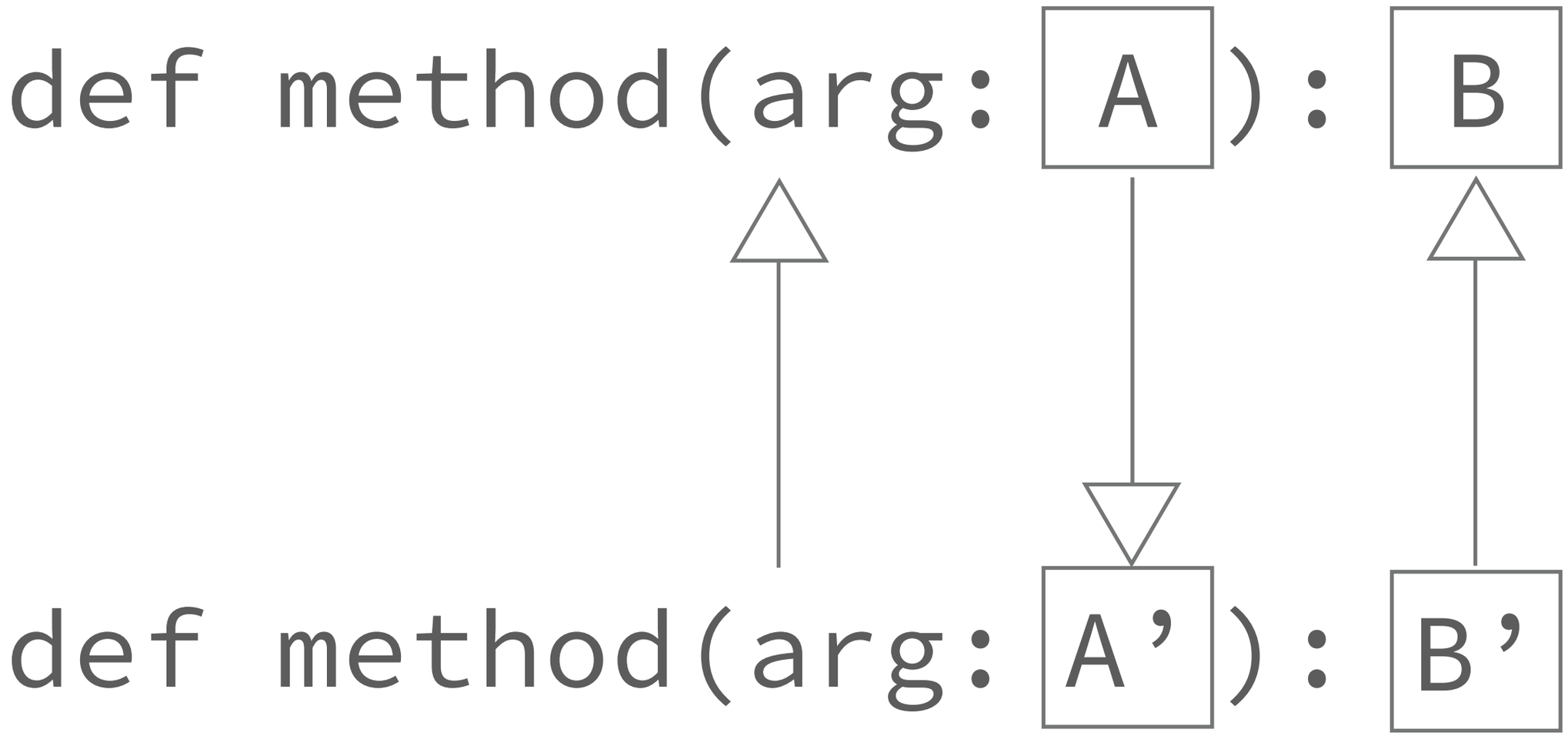
WHAT'S IN A CONTRACT? (II)

- We **must** respect the contract
- We can **constrain** the postconditions
- We can **relax** the preconditions

```
def method(arg: A): B
```

```
def method(arg: A'): B'
```


def method(arg: A): B
def method(arg: A’): B’

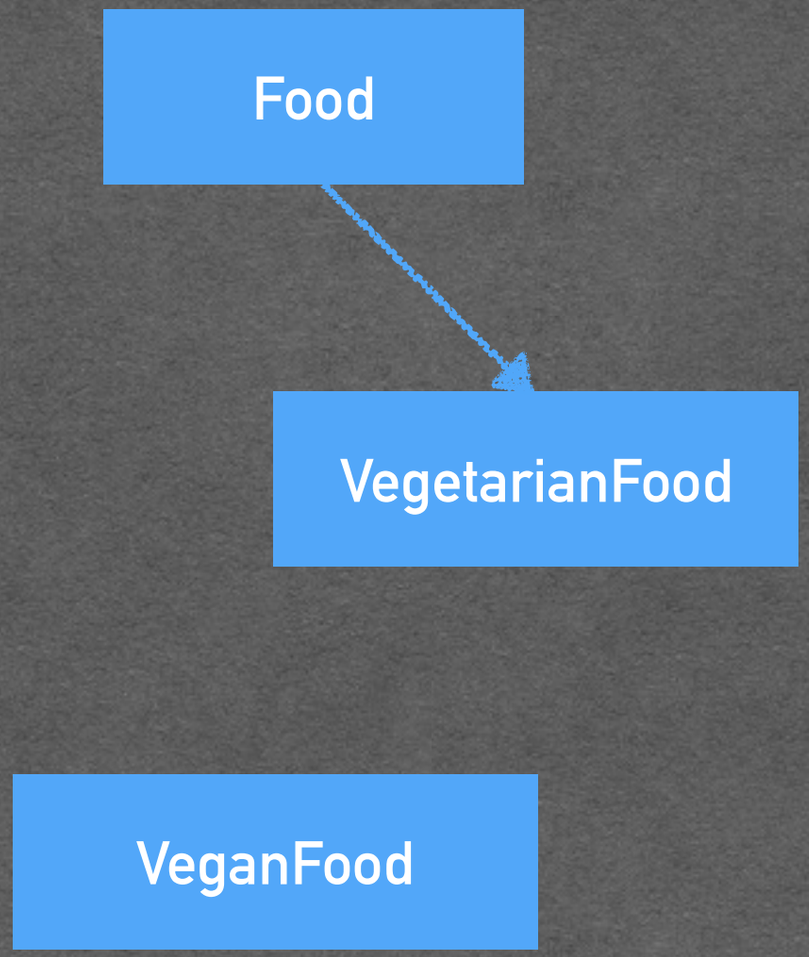

def method(arg: A): B
def method(arg: A’): B’

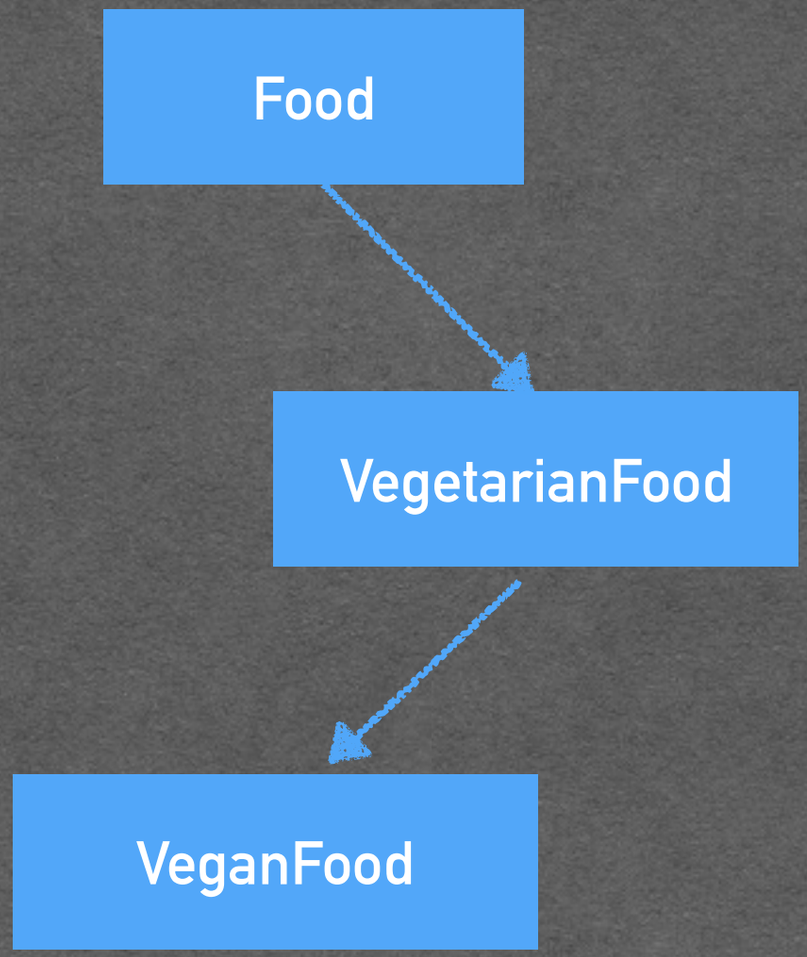
LET'S LISTEN TO THE CODE

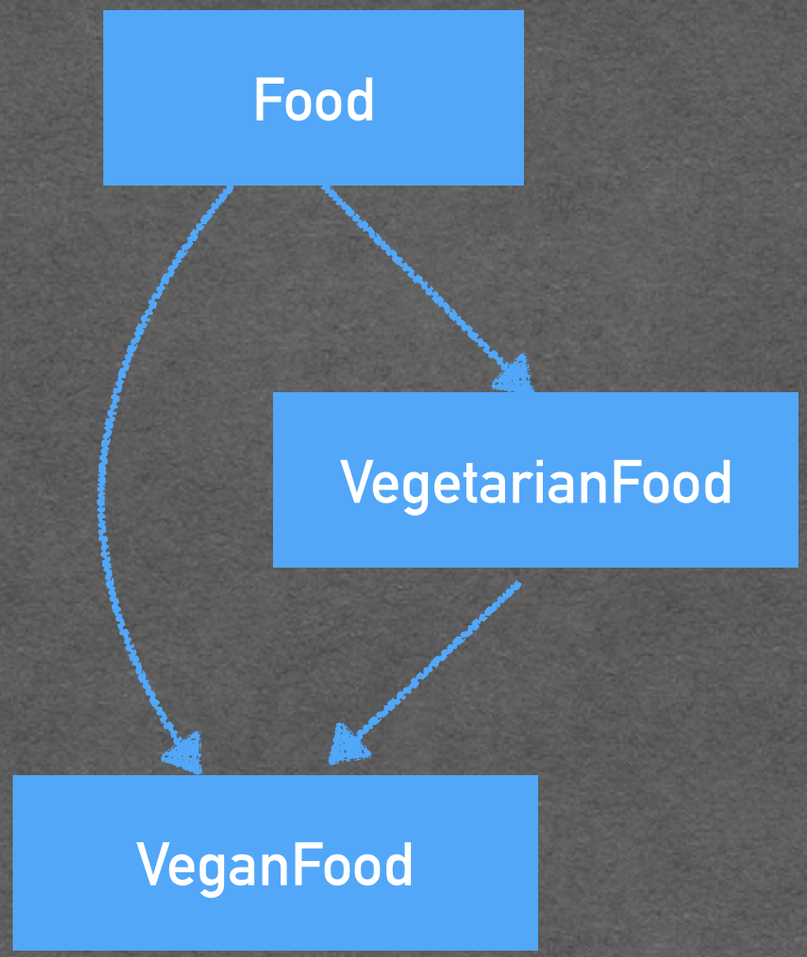
Food

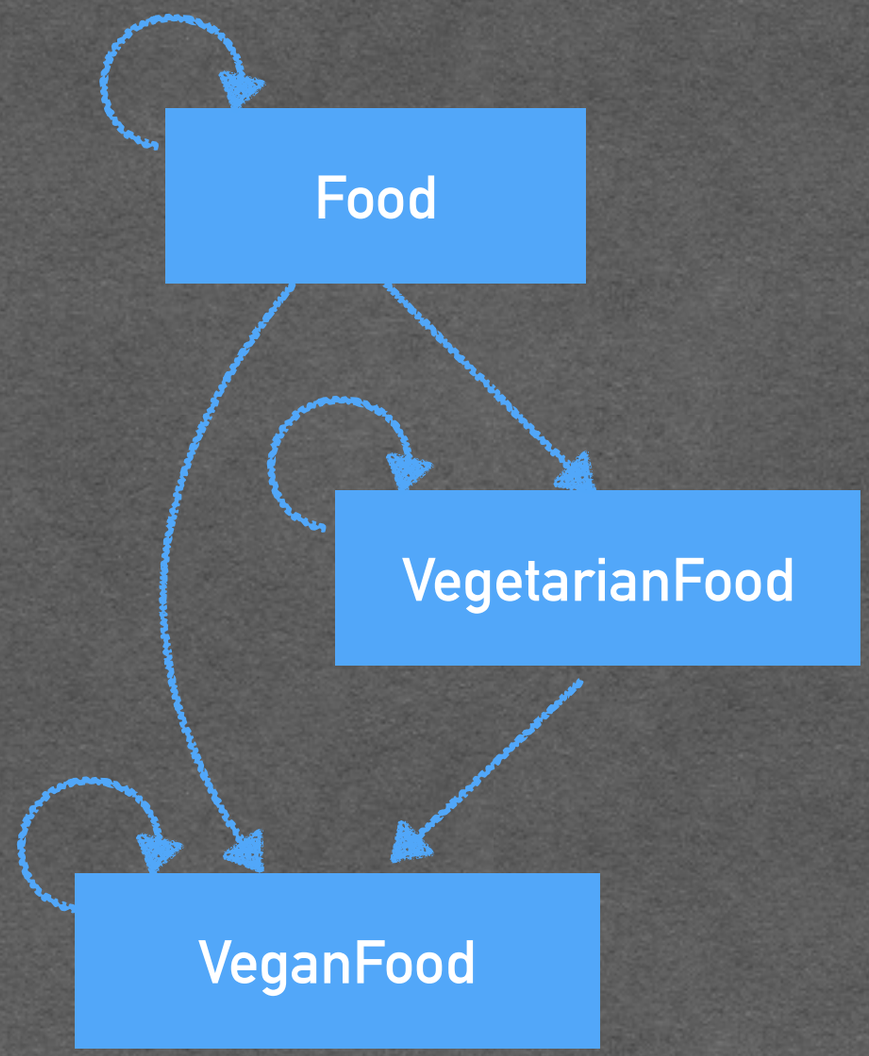
VegetarianFood

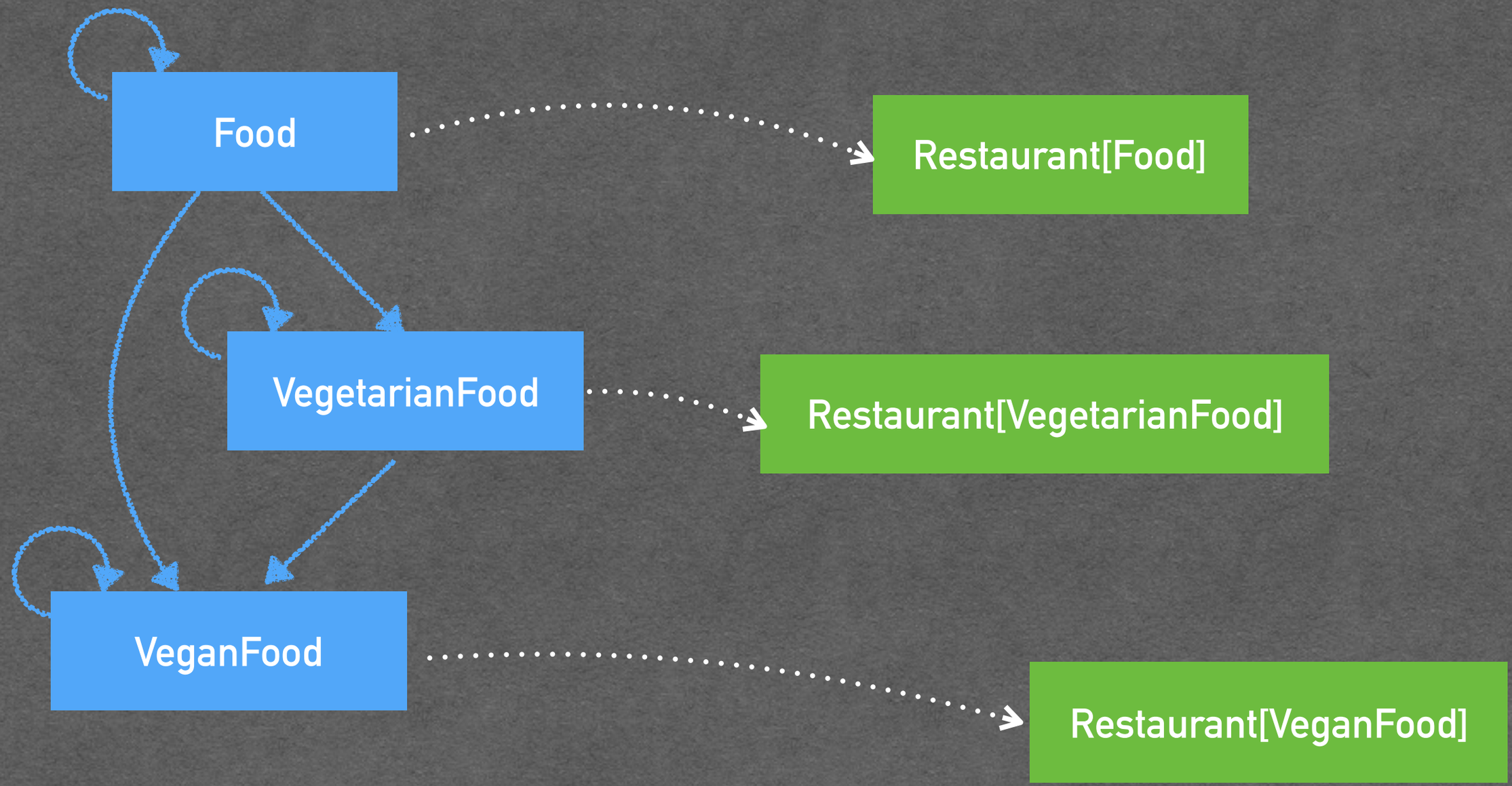
VeganFood

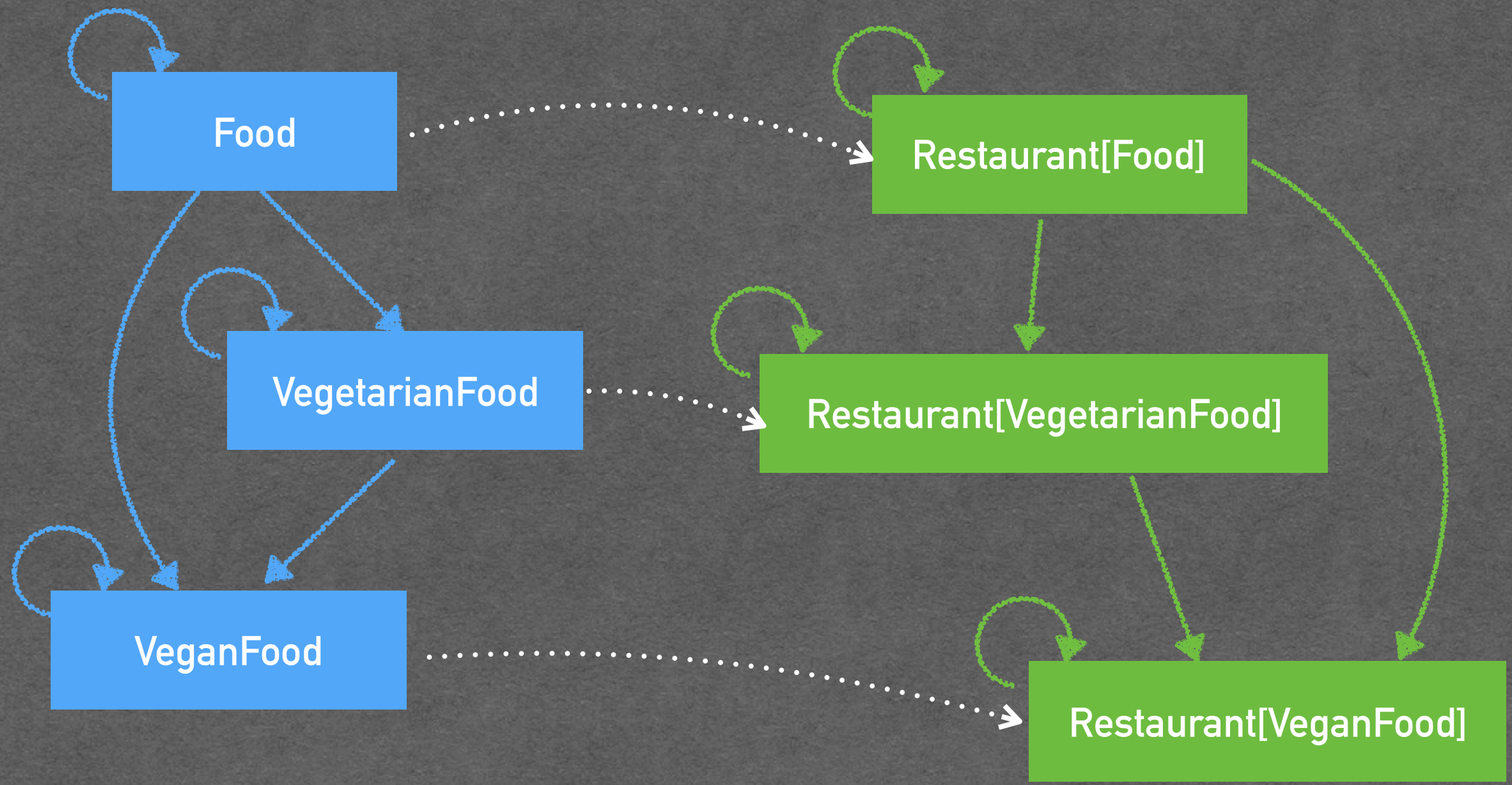

Food
VegetarianFood
VeganFood


Food
VegetarianFood
VeganFood

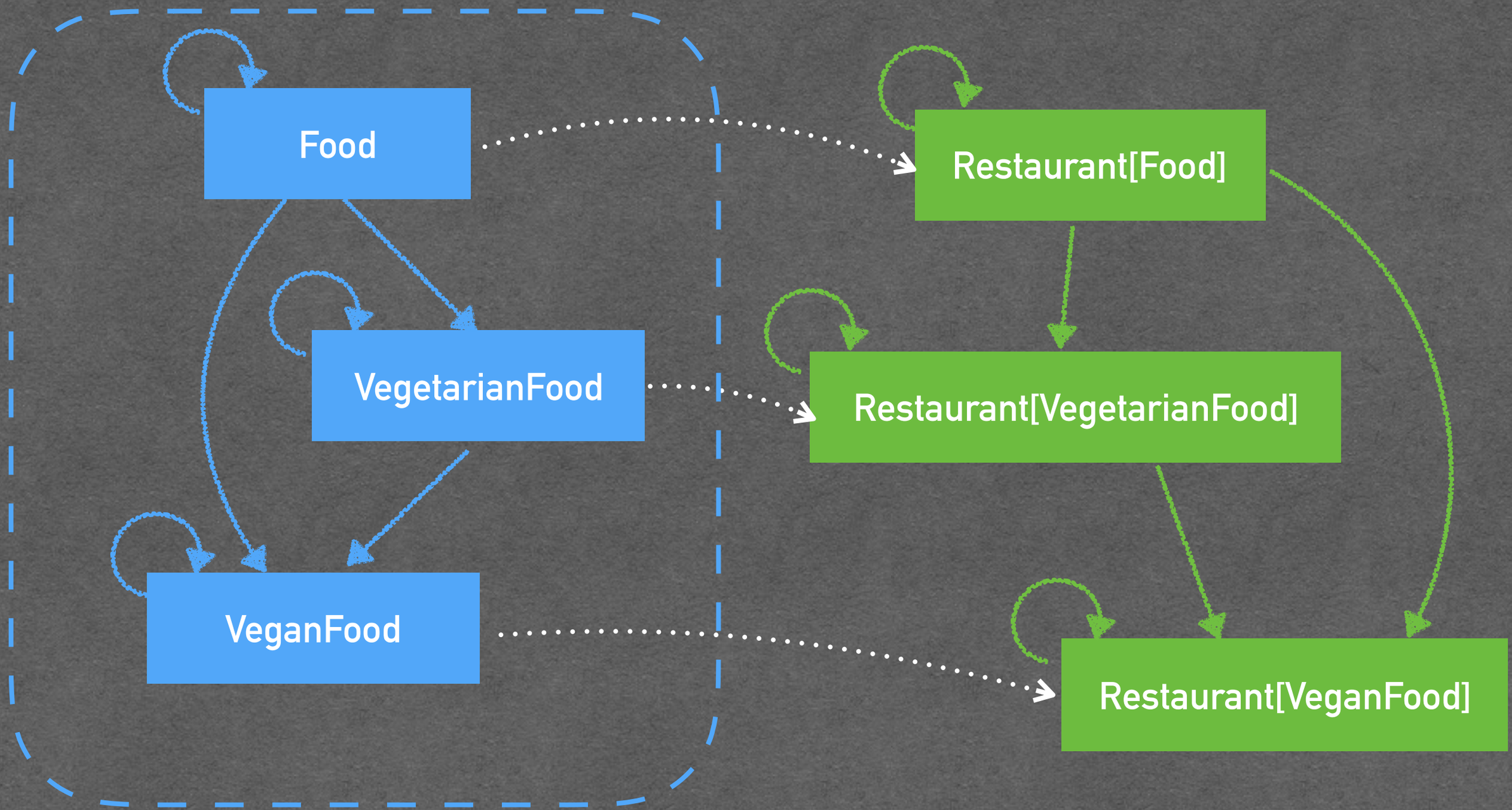

Food
VegetarianFood
VeganFood


Food
VegetarianFood
VeganFood

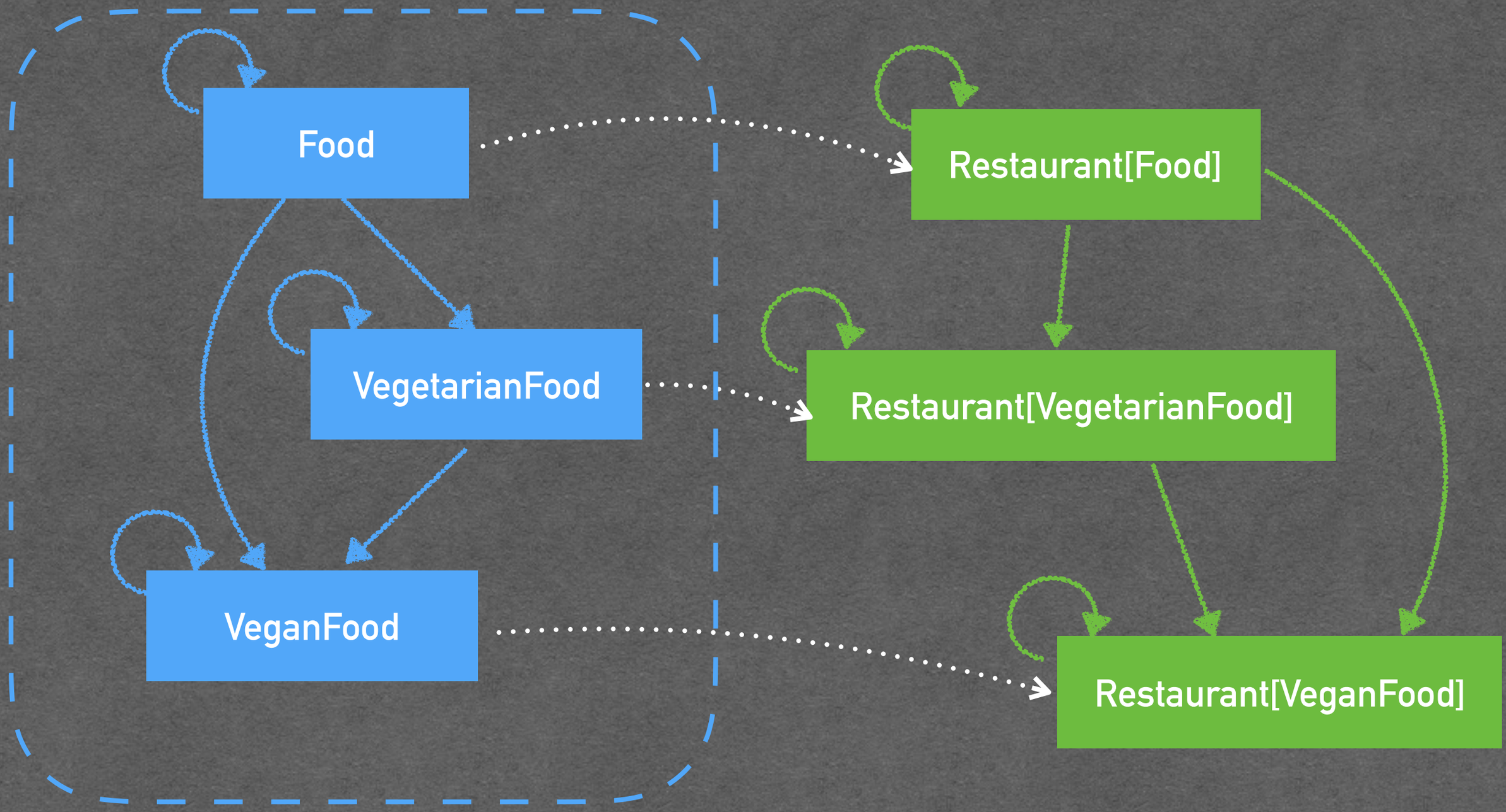

Food
Restaurant[Food]
VegetarianFood
Restaurant[VegetarianFood]
VeganFood
Restaurant[VeganFood]


Food
Restaurant[Food]
VegetarianFood
Restaurant[VegetarianFood]
VeganFood
Restaurant[VeganFood]

This is a category!

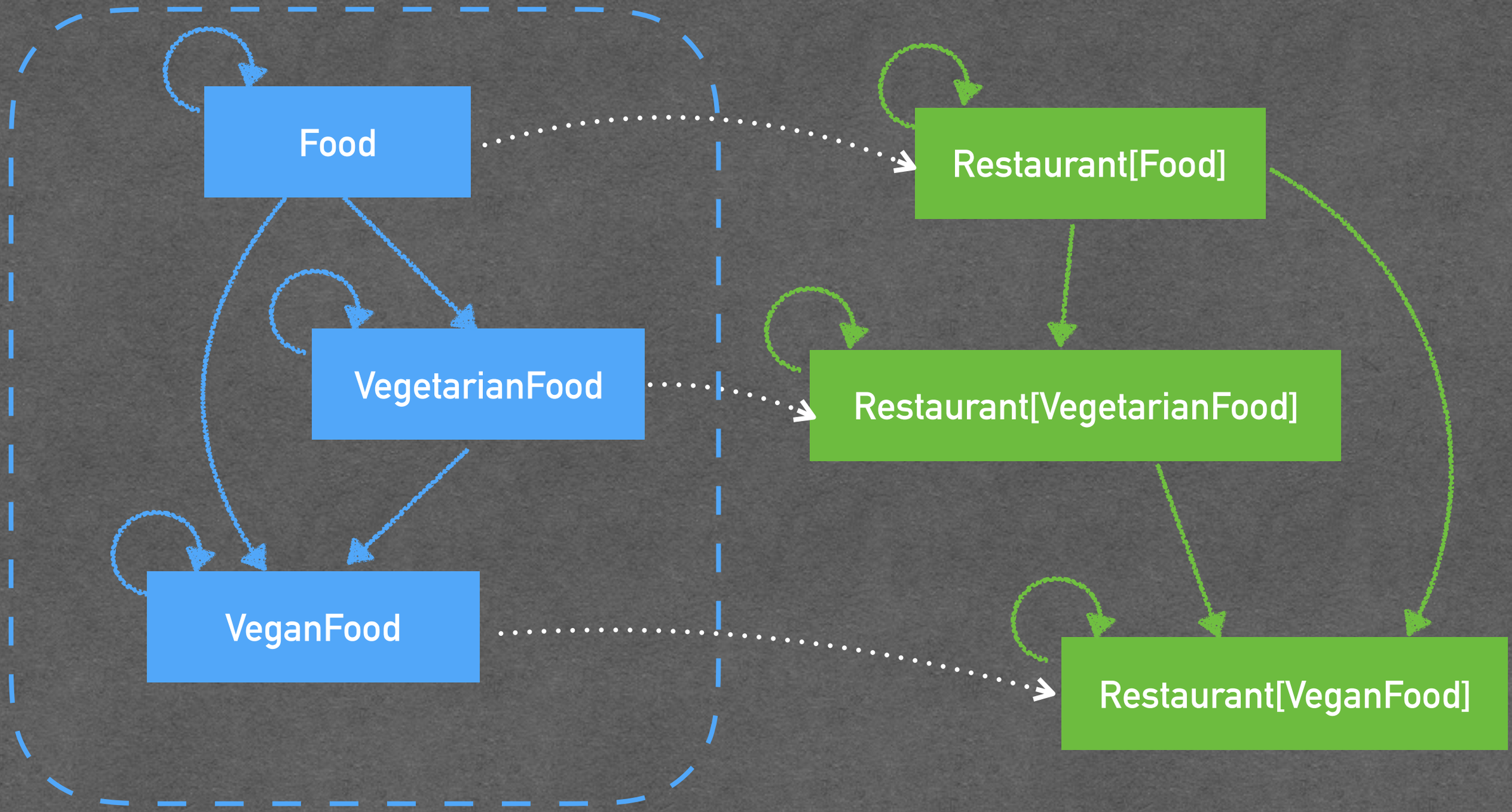


This is a category!



Type *'s are called objects*

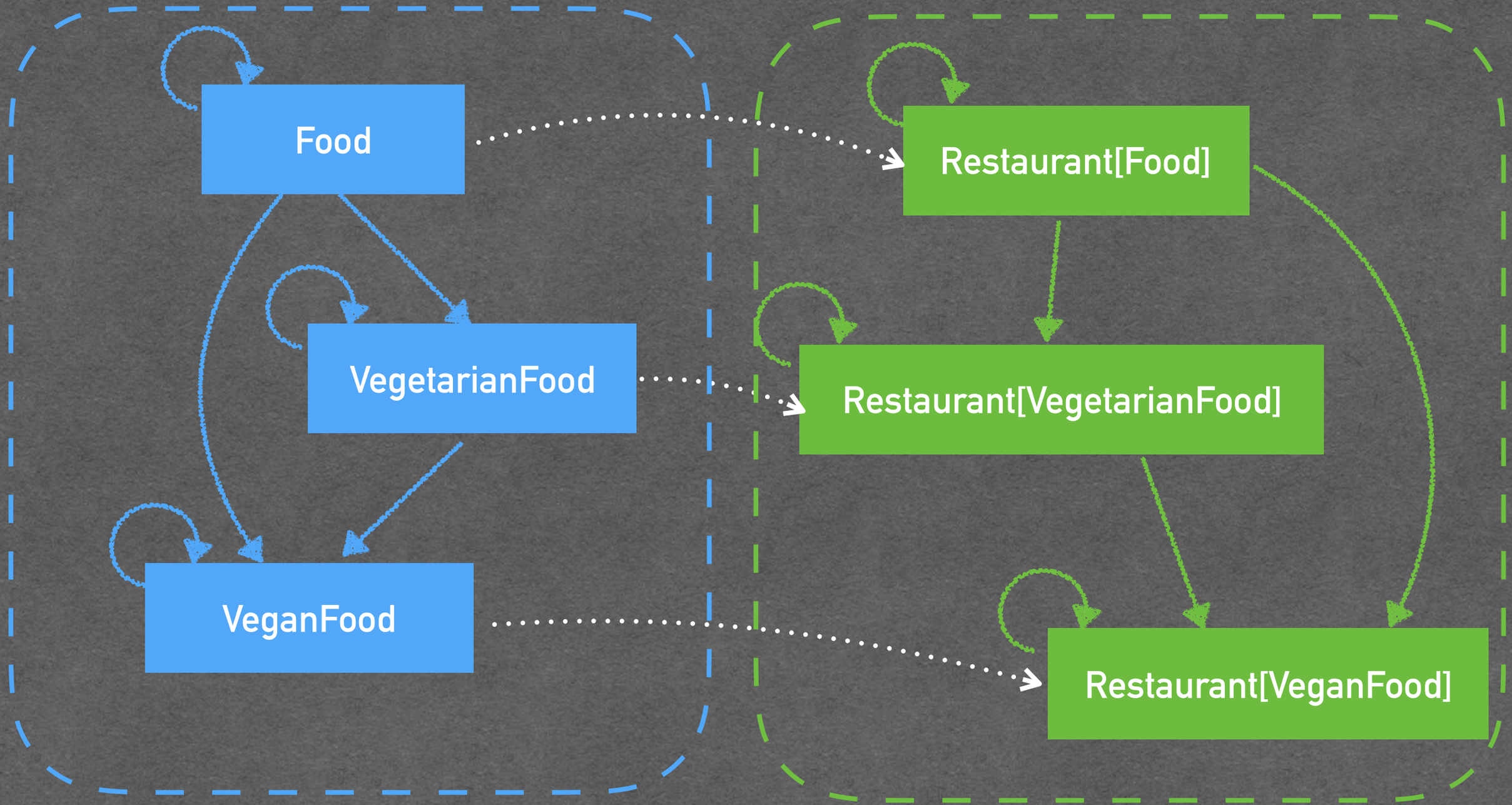
This is a category!



Type *'s are called objects*

→ *'s are called morphisms*

This is a category!



Type *'s are called objects*

's are called morphisms

Another category!

This is a category!

Restaurant[_] is covariant

Food

VegetarianFood

VeganFood

Restaurant[Food]

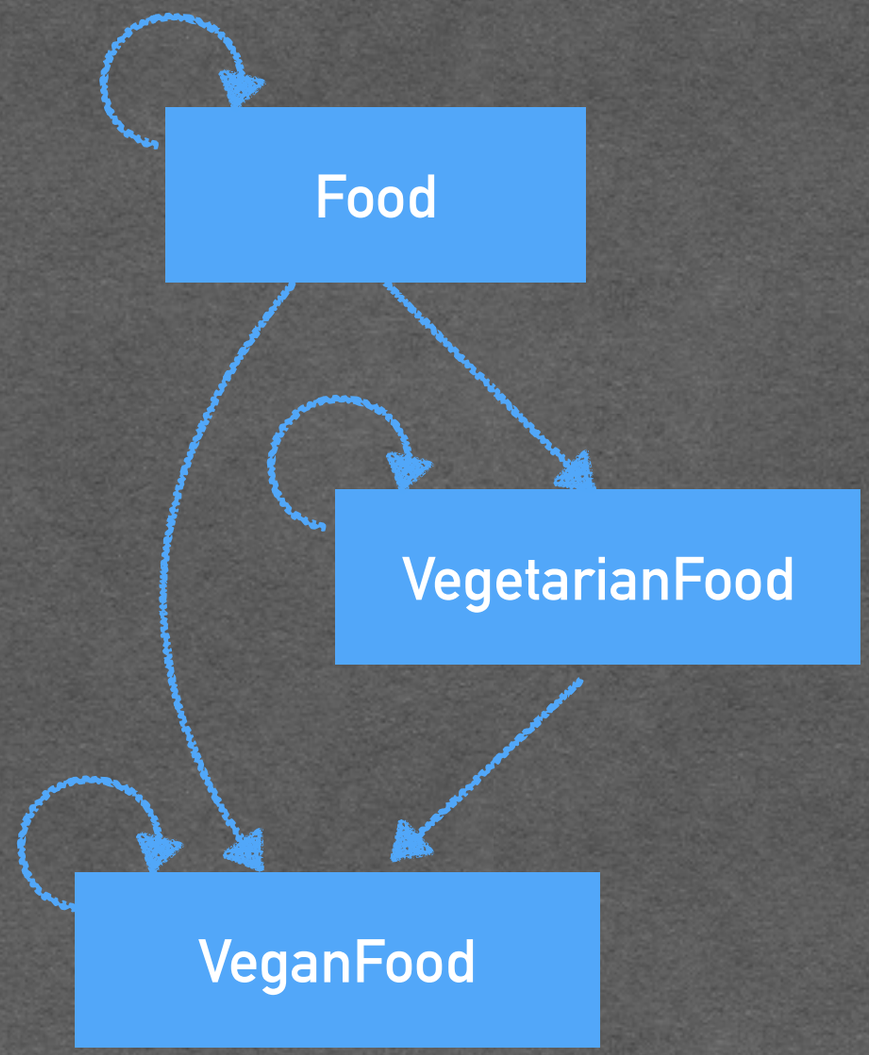
Restaurant[VegetarianFood]

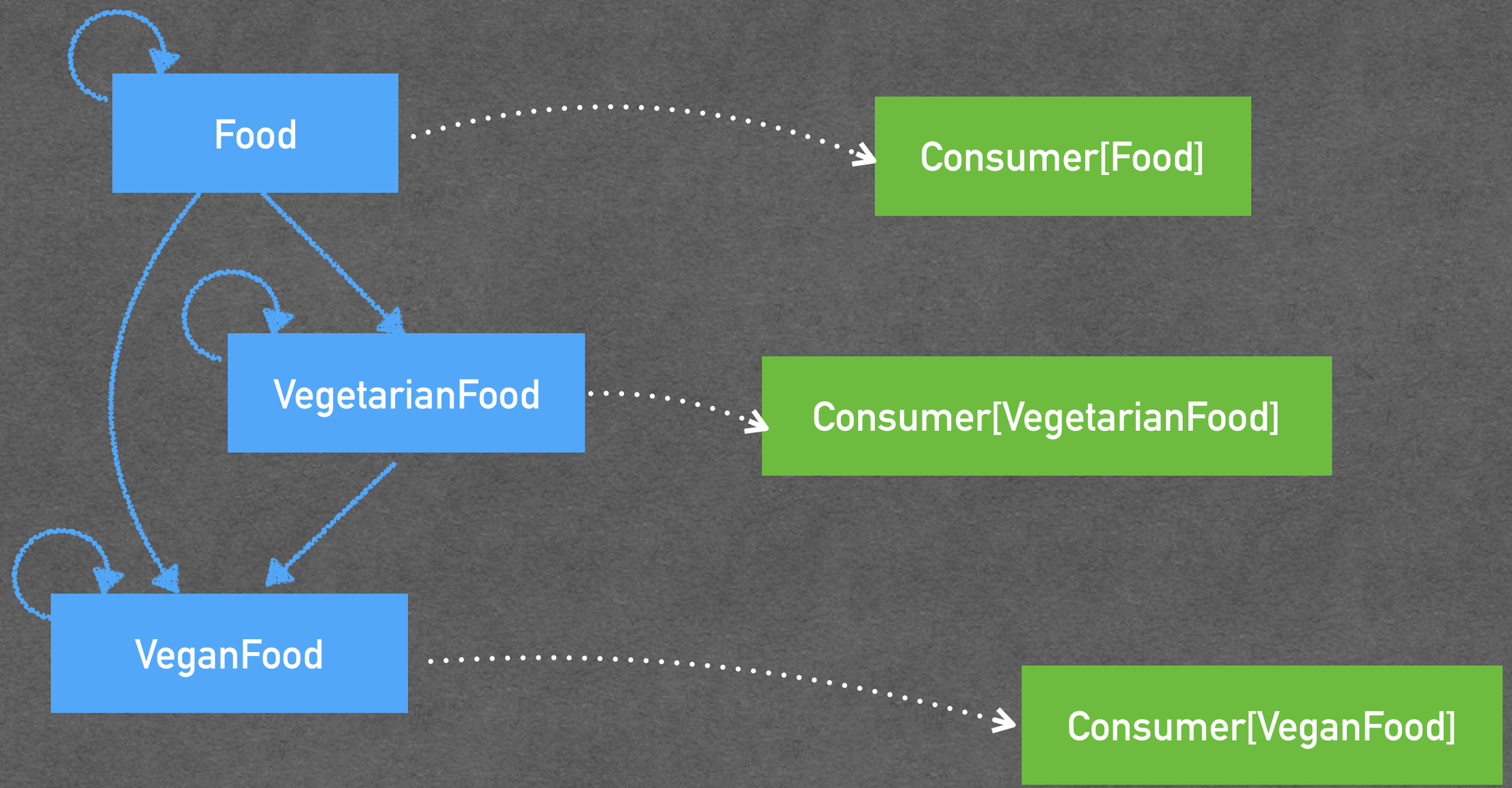
Restaurant[VeganFood]

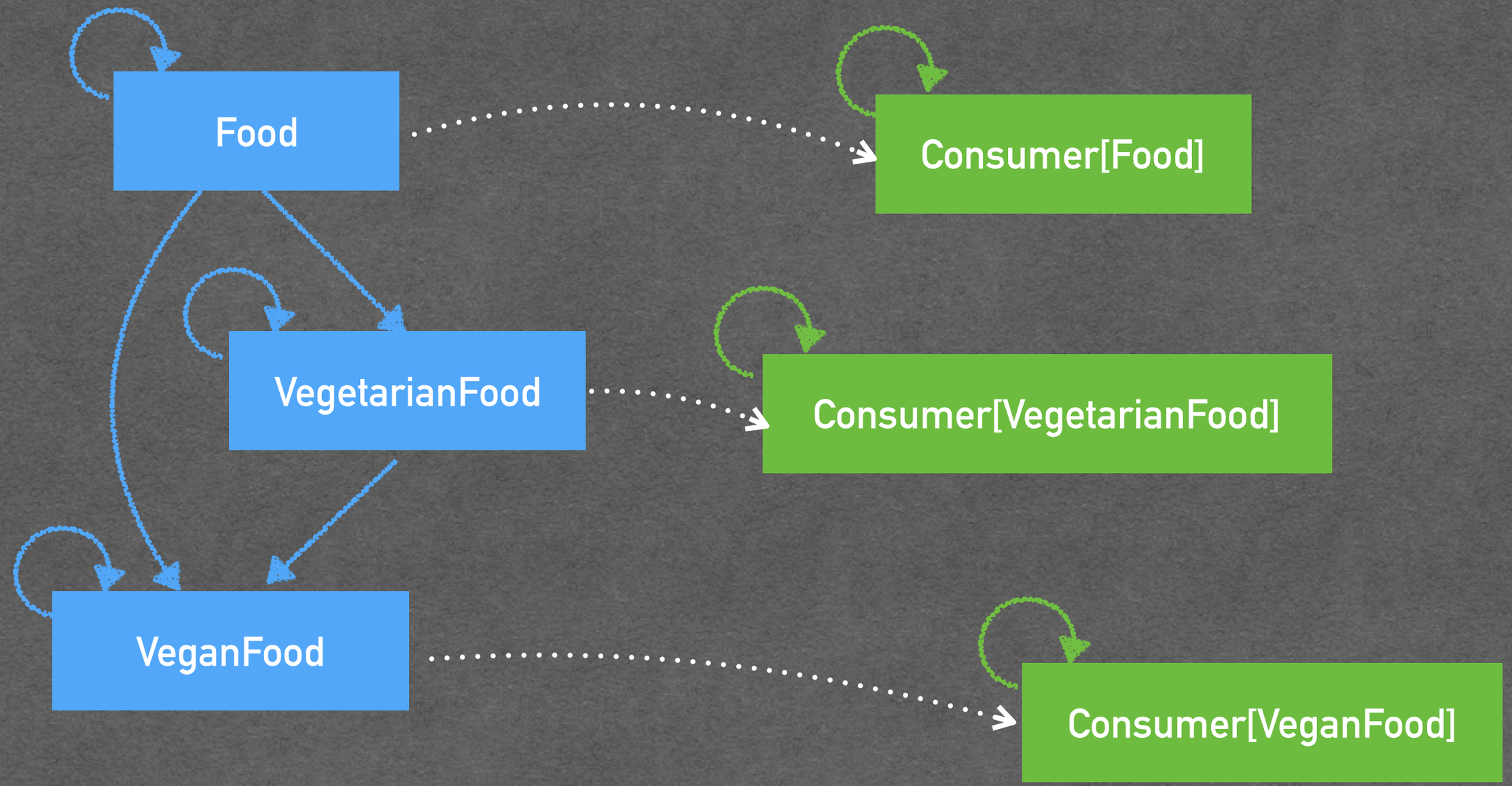
Type *'s are called objects*

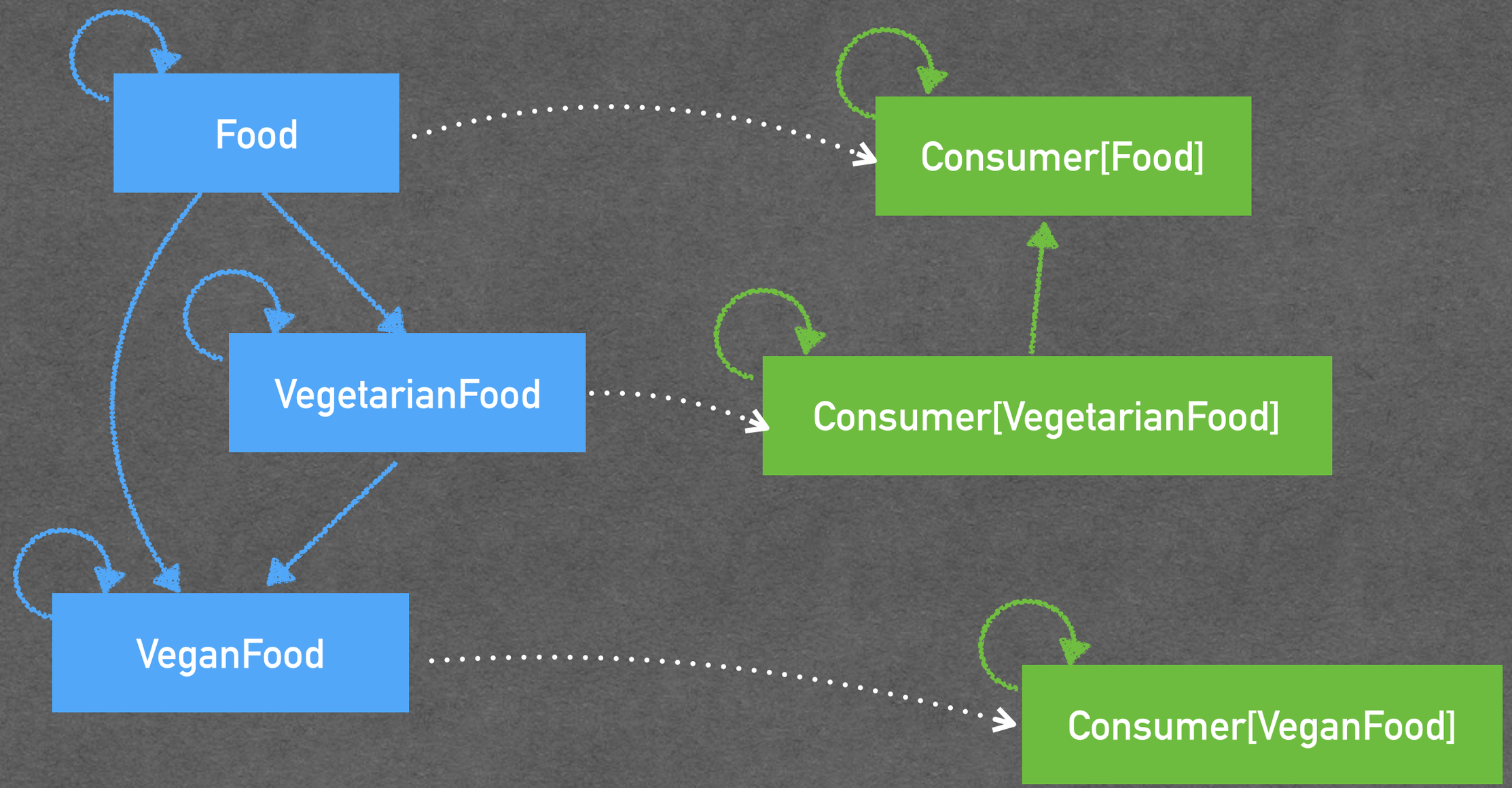
*'s are called morphisms

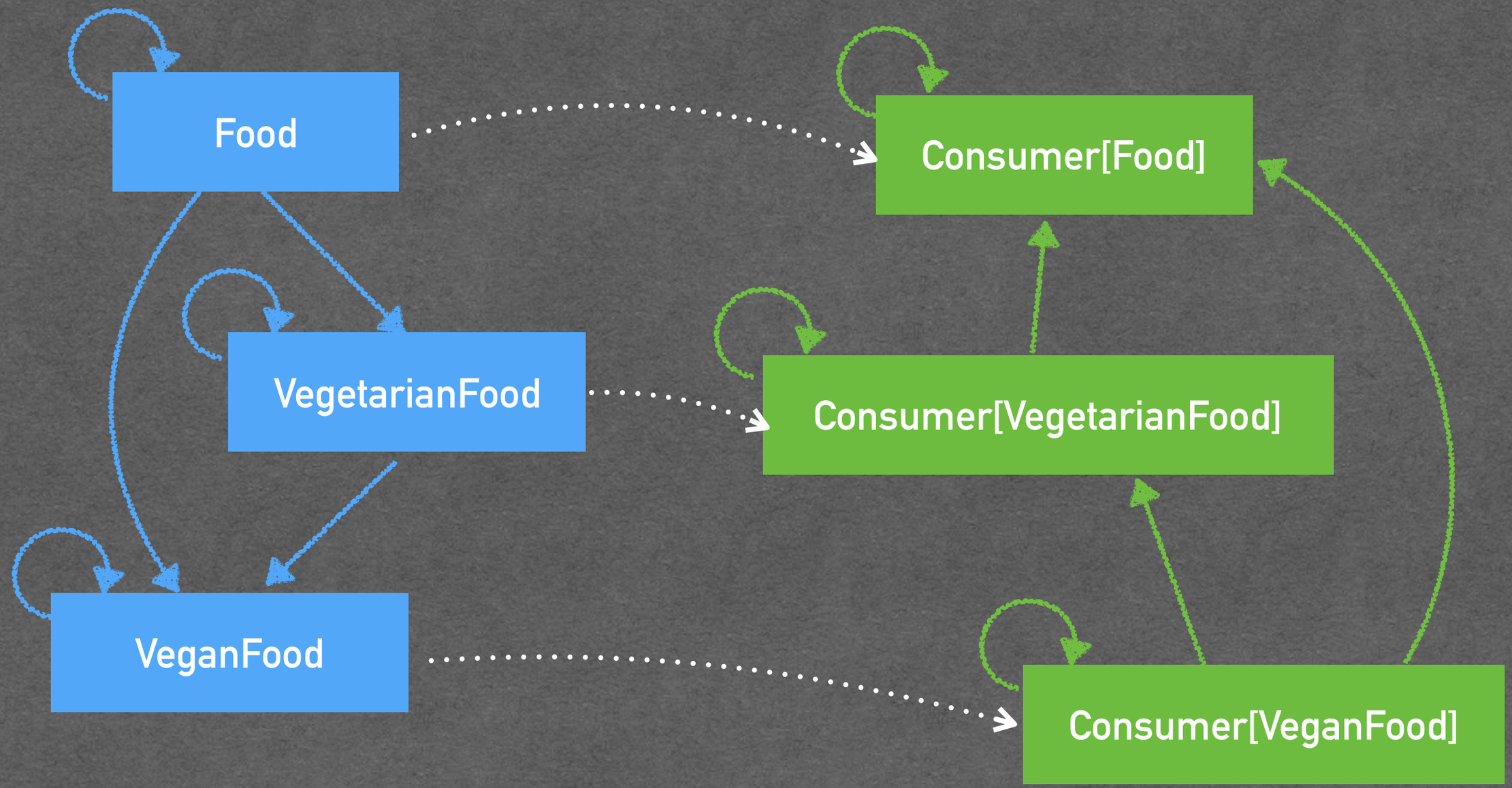
Another category!


Food
VegetarianFood
VeganFood

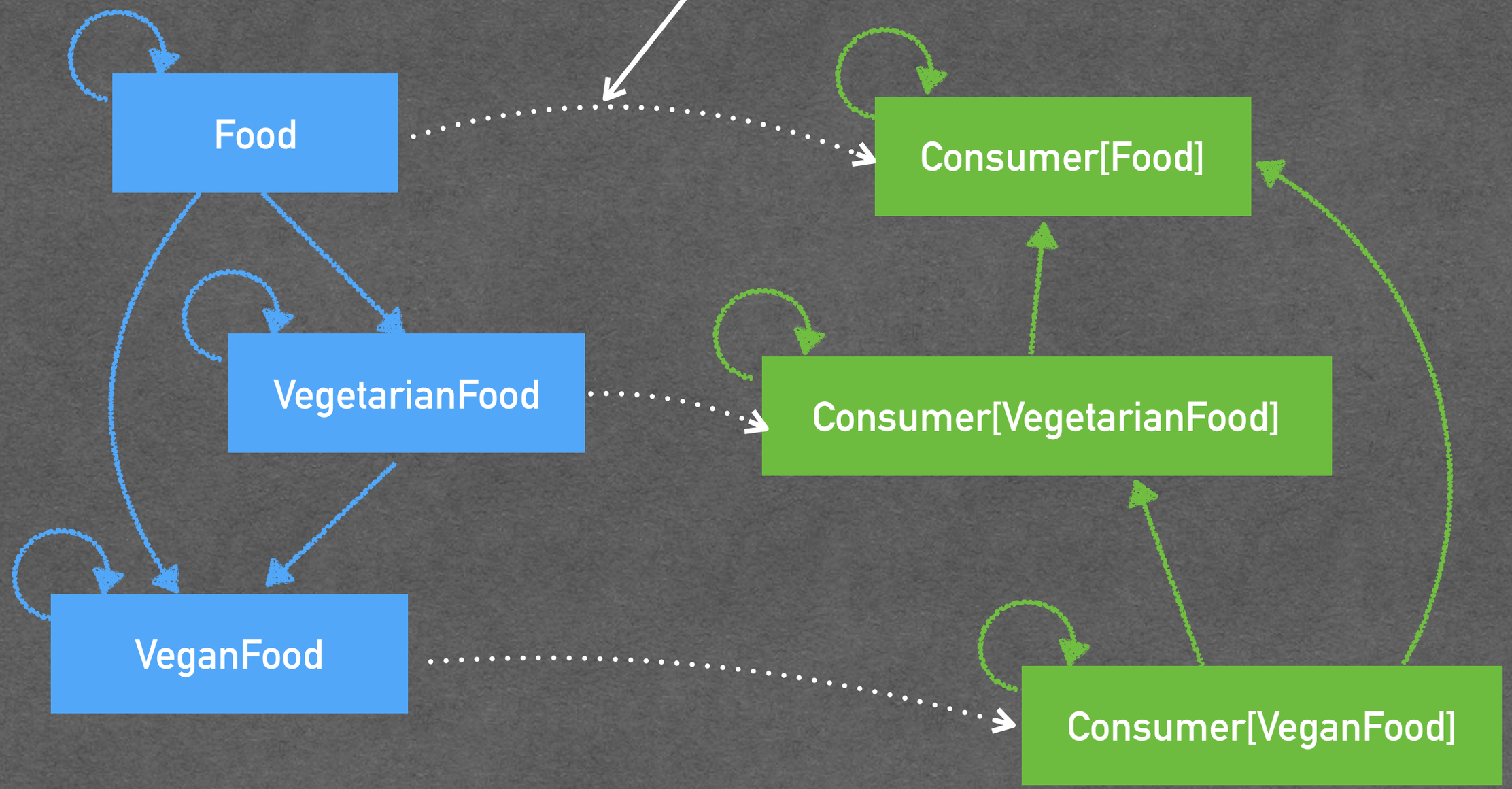

Food
Consumer[Food]
VegetarianFood
Consumer[VegetarianFood]
VeganFood
Consumer[VeganFood]


Food
Consumer[Food]
VegetarianFood
Consumer[VegetarianFood]
VeganFood
Consumer[VeganFood]

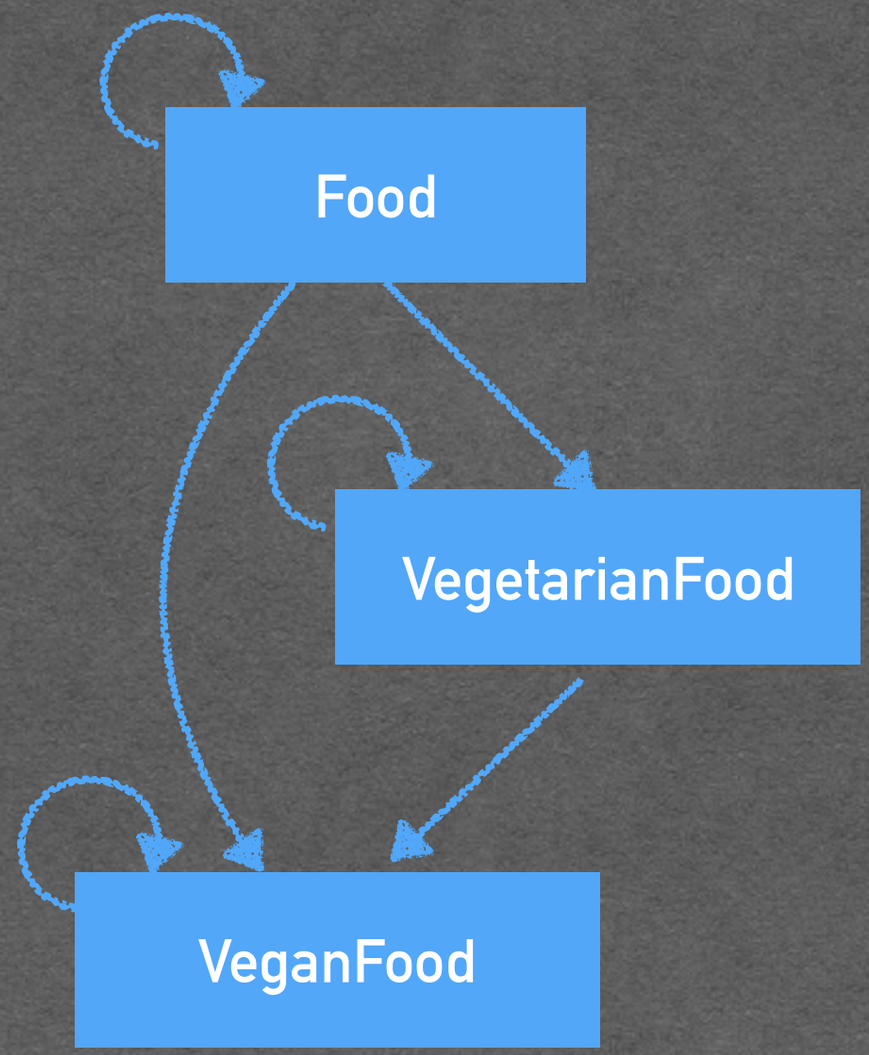

Food
VegetarianFood
VeganFood
Consumer[Food]
Consumer[VegetarianFood]
Consumer[VeganFood]

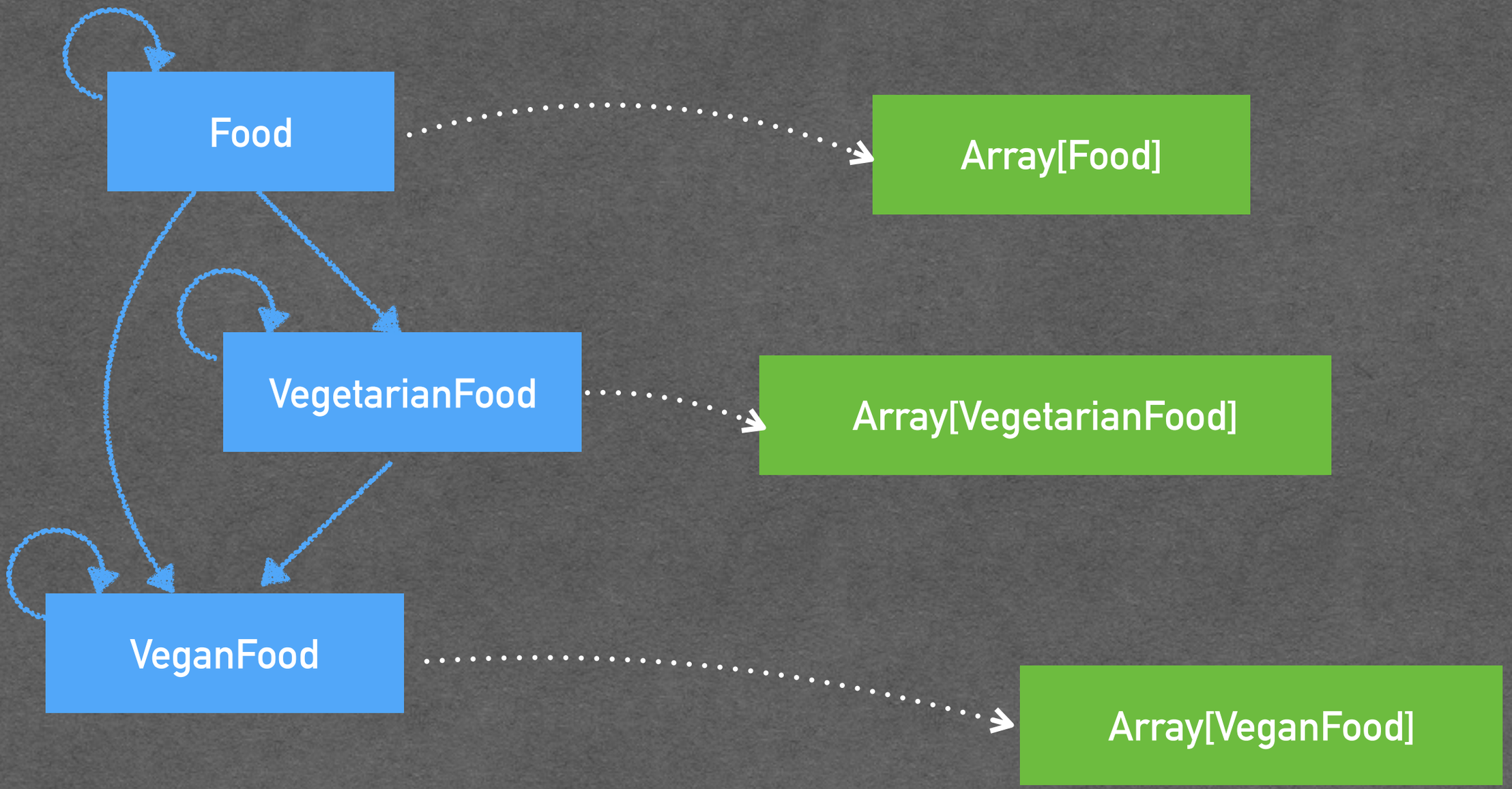

Food
VegetarianFood
VeganFood
Consumer[Food]
Consumer[VegetarianFood]
Consumer[VeganFood]

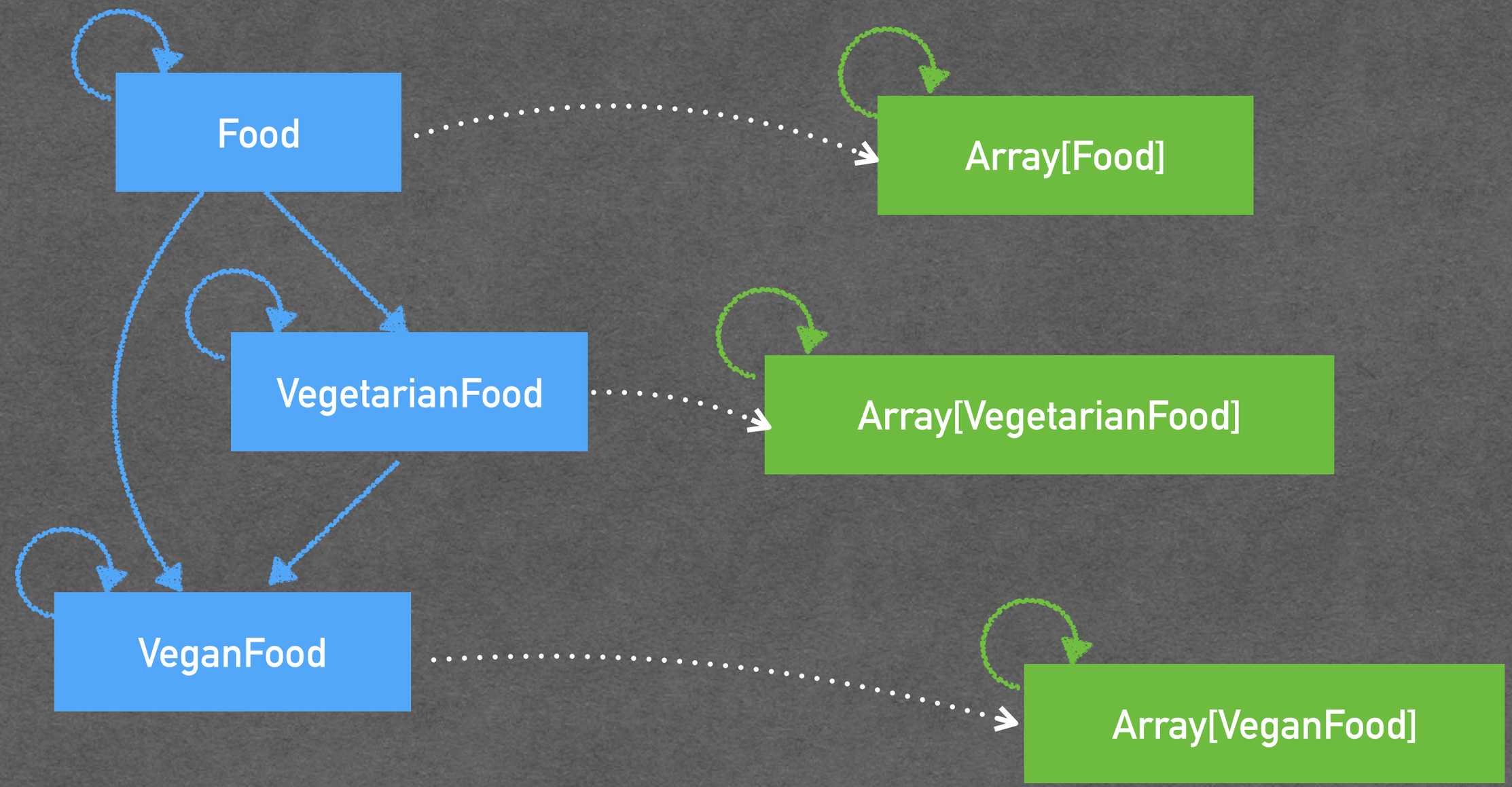
Consumer[_] is contravariant



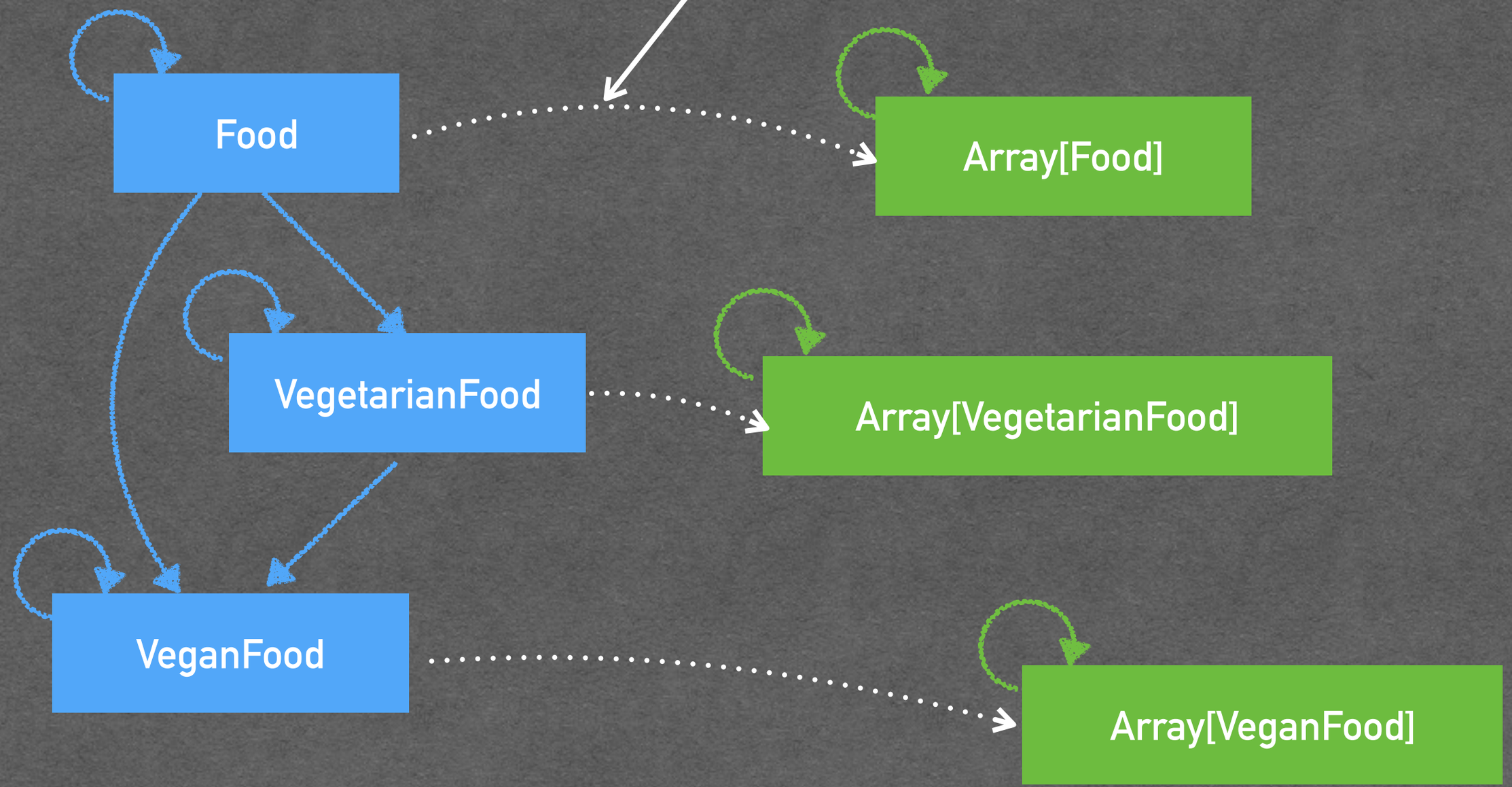
The dual category!


Food
VegetarianFood
VeganFood


Food
Array[Food]
VegetarianFood
Array[VegetarianFood]
VeganFood
Array[VeganFood]


Food
Array[Food]
VegetarianFood
Array[VegetarianFood]
VeganFood
Array[VeganFood]

Array[_] is invariant



Array[_] is invariant

Food

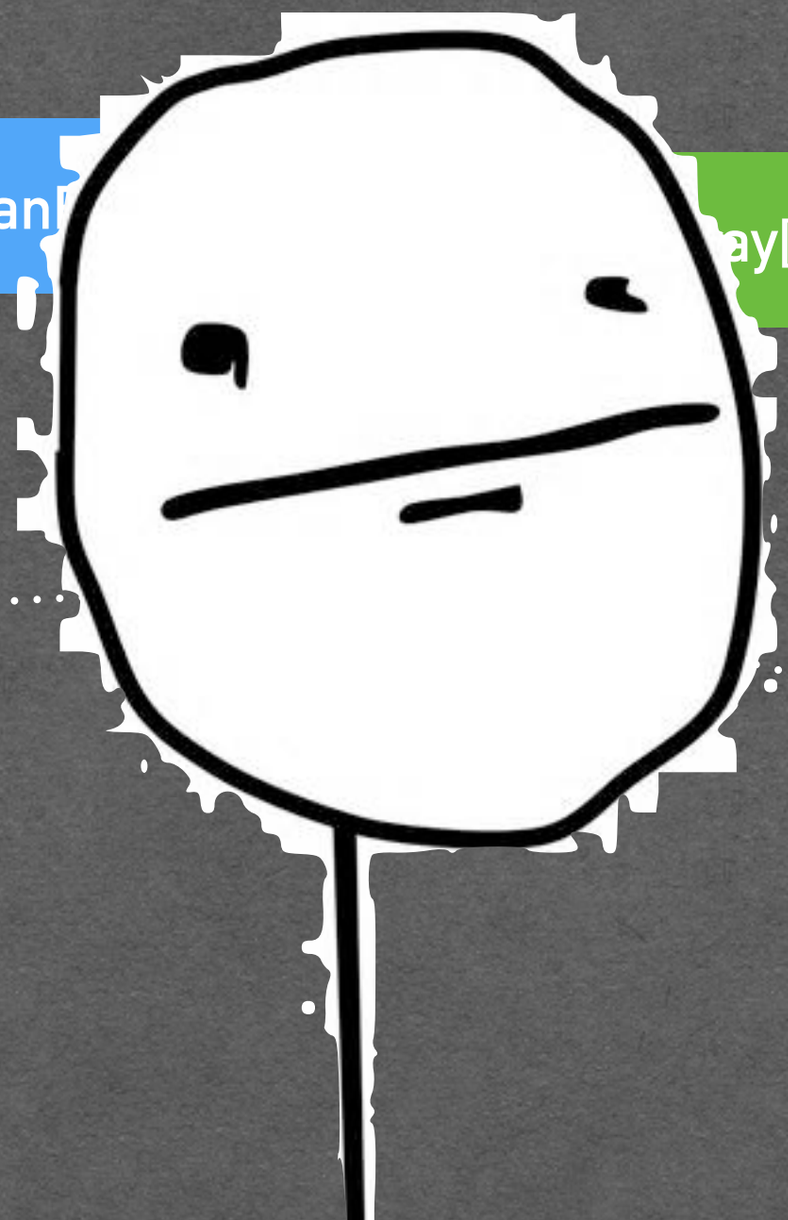
Vegetarian

VeganFood

Array[Food]

ay[VegetarianFood]

Array[VeganFood]



> By making type constructors covariant or contravariant instead of invariant, more programs will be accepted as well-typed.

NO
SWEDEN
LIVONIA
POLOTSK
SMOLENSK
LADIMIR
ENGLAND
SCOTLAND
DENMARK
POLAND
HOLY ROMAN EMPIRE
FRANCE
BRYANSK
KIEV
PEREYASLAVL
HUNGARY
BULGARIA
LEÓN
CASTILLE
ARAGON
PORTUGAL
SICILY
ALMOHAD


Congratulations!
You have conquered the provinces of Covariant and Contravariant Functors
ENGLAND
SCOTLAND
DENMARK
SWEDEN
NORWAY
LIVONIA
POLOTSK
SMOLENSK
BRYANSK
KIEV
PEREYASLAVL
LADIMIR
LEÓN
CASTILLE
PORTUGAL
ALMOHAD
SICILY
141
16 May,

CO/CONTRA VARIANCE EVERYWHERE!

- Collections library:
  Seq[+A], Iterable[+A]
- A=>B desugars to
  Function1[-A, +B]
- PartialFunction[-A, +B].
- PartiallyOrdered[+A]
- Scalacheck generators,
  Gen[+T]
- …

Q/A

- Incomplete and mostly wrong history of programming languages. http://james-iry.blogspot.com.es/2009/05/brief-incomplete-and-mostly-wrong.html
- (Liskov, 99) “Behavioral Subtyping Using Invariants and Constraints”, Barbara Liskov & Jeannette Wing
- Uncle Bob on LSP. http://www.objectmentor.com/resources/articles/lsp.pdf
- The Rust book. http://doc.rust-lang.org/book/
- Scape from the Ivory Tower, presentation by Simon Peyton Jones. https://yow.eventer.com/events/1004/talks/1054
- Presentation by Philip Walder: “Propositions as types”. https://www.youtube.com/watch?v=IOiZatlZtGU

- Blog post, “Covariance and Contravariance” http://typeinference.com/typing/2015/10/29/covariance_and_contravaciance.html
- Celular automaton as in “A new kind of science” by Stephen Wolfram. http://wolframscience.com/thebook.html