Handling Errors in a Functional Way

Esteban Herrera
JAVA ARCHITECT

@eh3rrera www.eherrera.net

Side Effects

Throwing an exception doesn’t break the purity of a function

Catching an exception can do it

Nondeterministic Exceptions

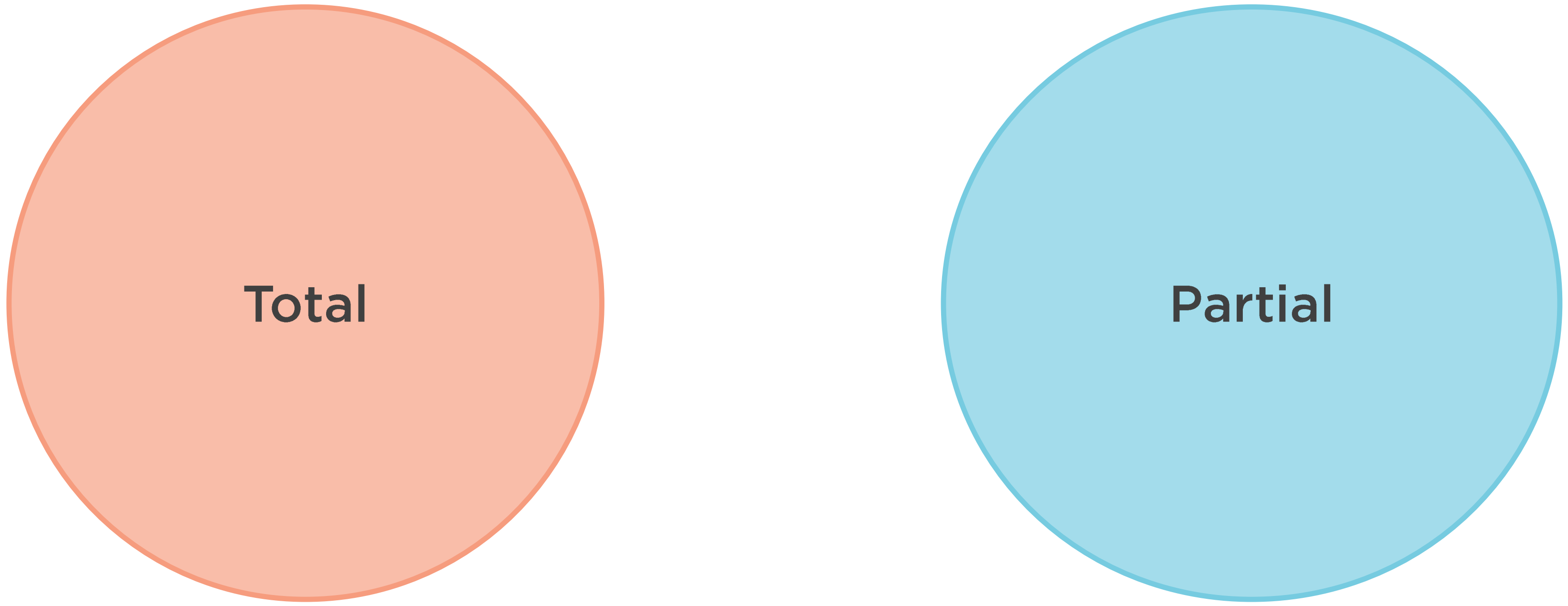
- If you catch them, you can return different values for the same input
- Languages that don’t enforce purity let you catch exceptions anywhere

Deterministic Exceptions

Exceptions thrown when a particular set of arguments is evaluated

The function is pure if always throws an exception for the same argument

Two Types of Functions



Total function

A function that is defined for all possible values of its input. It always terminates and returns a value.

Partial function

A function that is not defined for all possible input values, because in some cases, it may never return anything at all.

Partial functions

!=

Partially-applied functions

If null, an exception is thrown

```
Function<Integer, Integer>  f = a -> a / 2;
```

To Make a Function Total

Change the domain
(e.g. with a NonNullInteger type)

Change the codomain
(e.g. with an IntegerOrException type)

Functional programmers avoid exceptions by using total functions.

```
IntegerOrException operation(Integer i) {
    // ...
}
```

```
Integer operation(Integer i) throws Exception {
    // ...
}
```

Railway-oriented Programming

```
GiftRewardLoyaltyProgram loyaltyProgram =
                   lpRepository.getGiftRewardLoyaltyProgram();

loyaltyProgram.setNeededPoints(points);
loyaltyProgram.setProductId(productId);

lpRepository.save(loyaltyProgram);
```

A More Robust Approach

Validate points

Get the product

Validate product

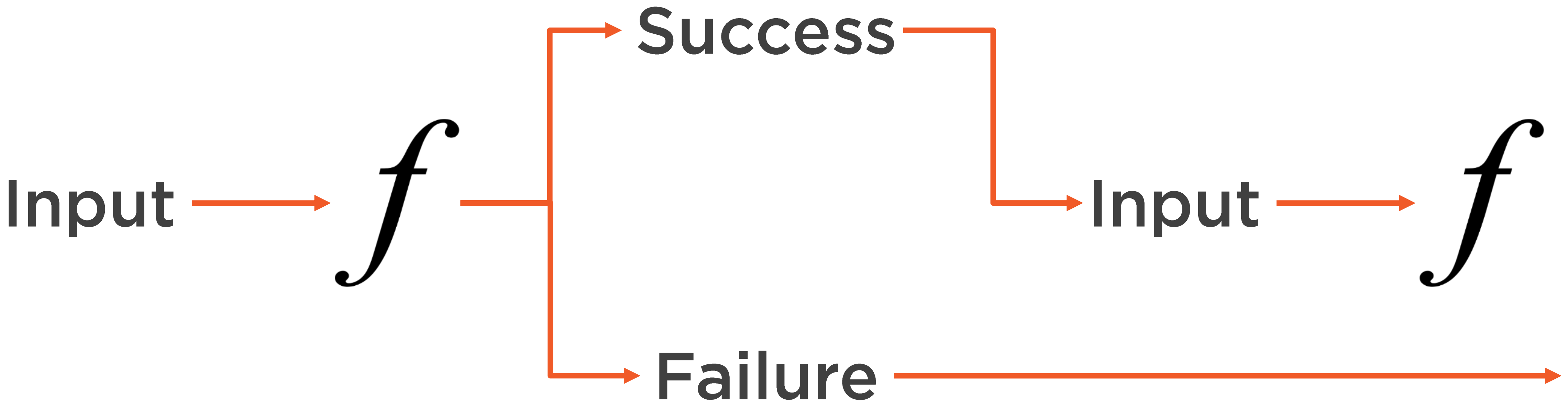
Update points and product

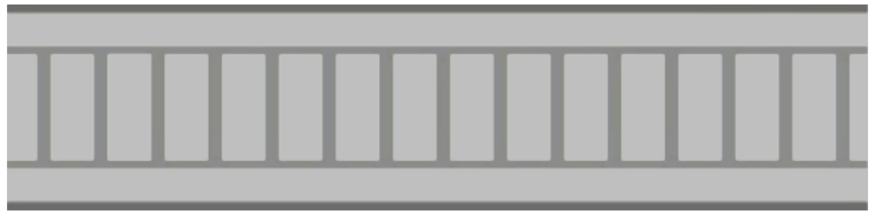
Save this to the database

```
GiftRewardLoyaltyProgram loyaltyProgram =
                   lpRepository.getGiftRewardLoyaltyProgram();

loyaltyProgram.setNeededPoints(points);
loyaltyProgram.setProductId(productId);

lpRepository.save(loyaltyProgram);
```

```java
if ( !isNumberOfPointsValid(newLoyaltyProgram) ) {
    throw new RuntimeException("Invalid points");
}
if ( !isProductValid(newLoyaltyProgram) ) {
    throw new RuntimeException("Invalid product");
}
GiftRewardLoyaltyProgram lp = lpRepository.getGiftRewardLoyaltyProgram();
if ( lp == null ) {
    throw new RuntimeException("The loyalty program was not found");
}
lp.setNeededPoints(newLoyaltyProgram.getNeededPoints());
lp.setProductId(newLoyaltyProgram.getProductId());
try {
    lpRepository.save(lp);
} catch( Exception e ) {
    throw new RuntimeException("Error when saving to the database");
}
```

```
if valid points
    return Success
else
    return Failure("Invalid points")
```

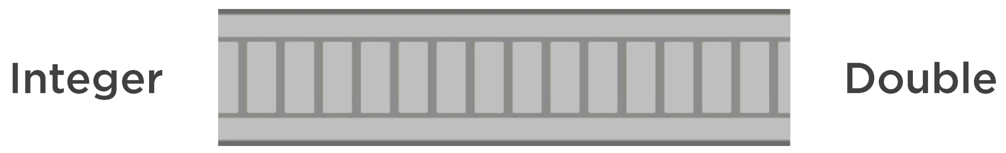

Input
f
Success
Failure
Input
f

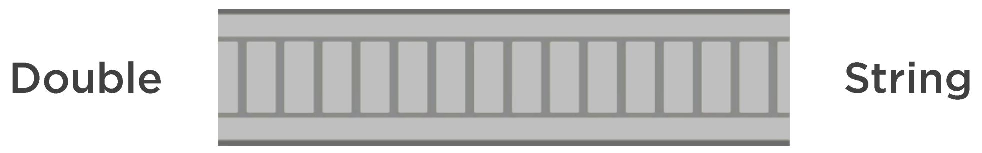
Integer  Double

$$f$$

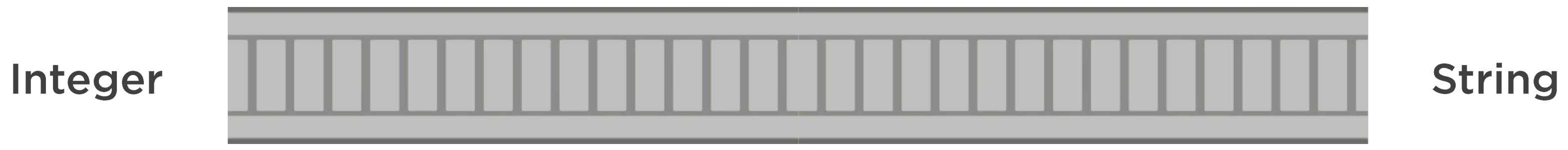
$$\text{Integer} \longrightarrow \text{Double}$$

Composition

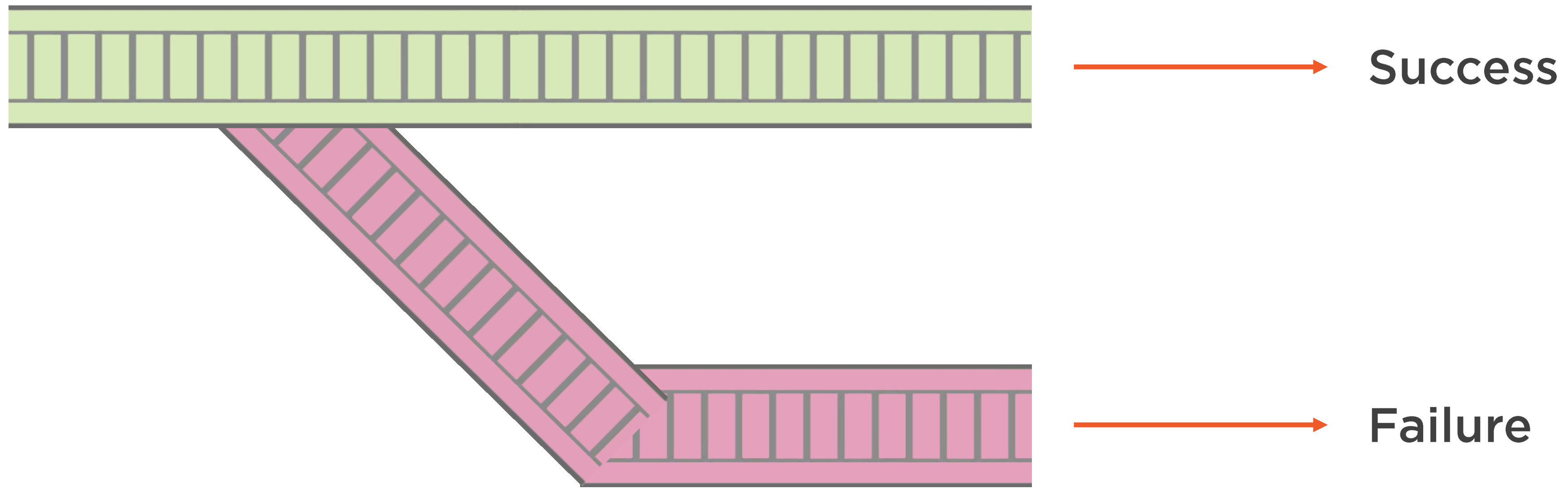




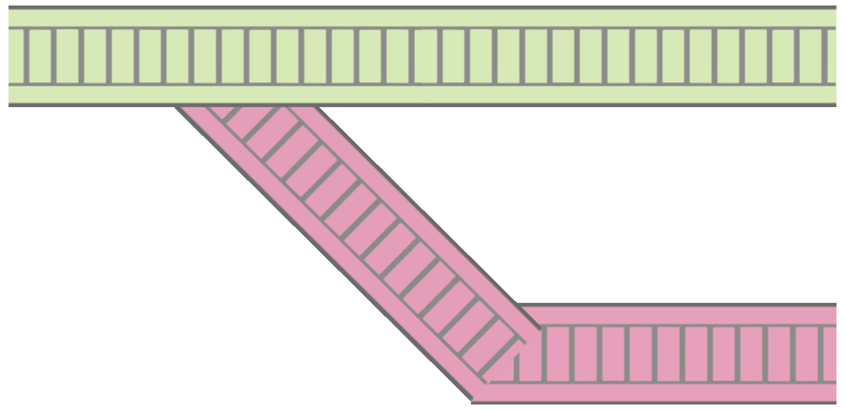
Composition

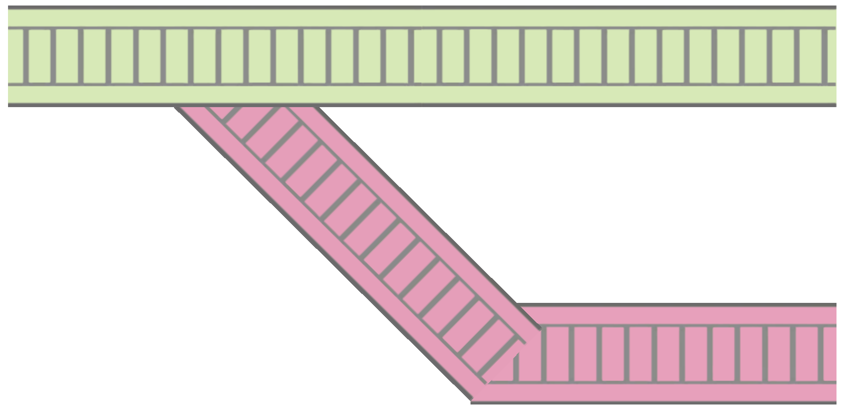


Railway Switches

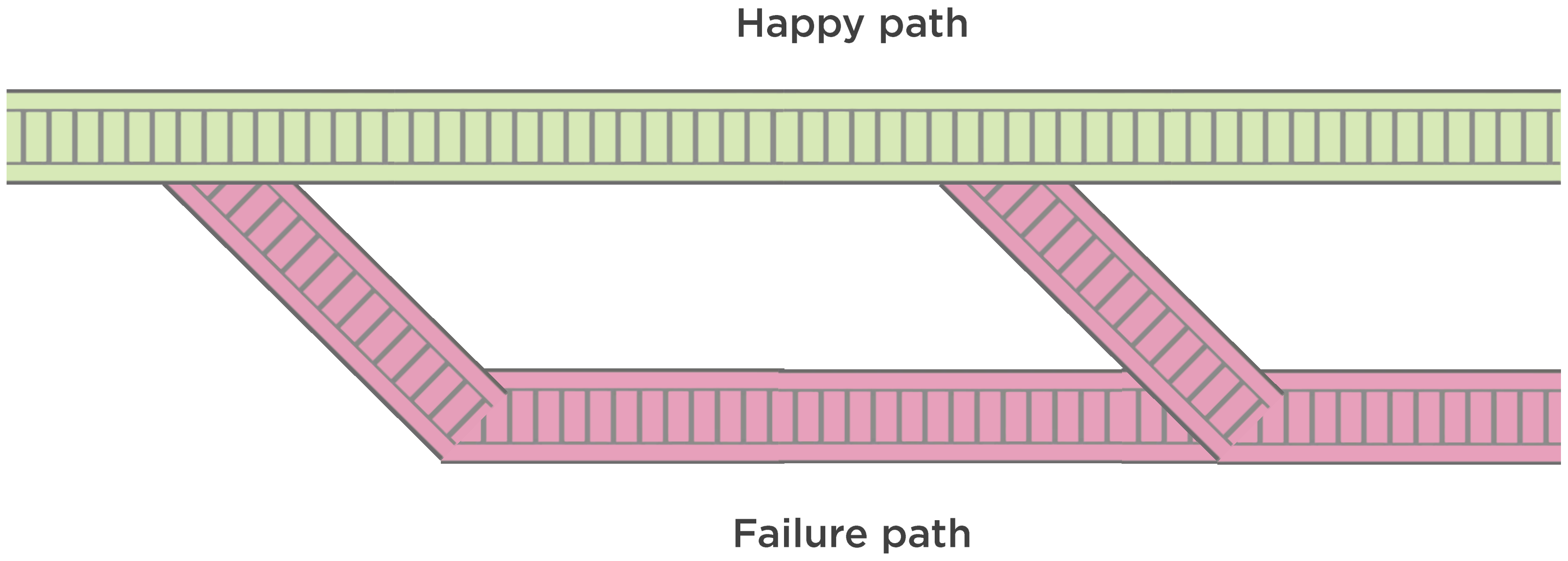


Railway Switches

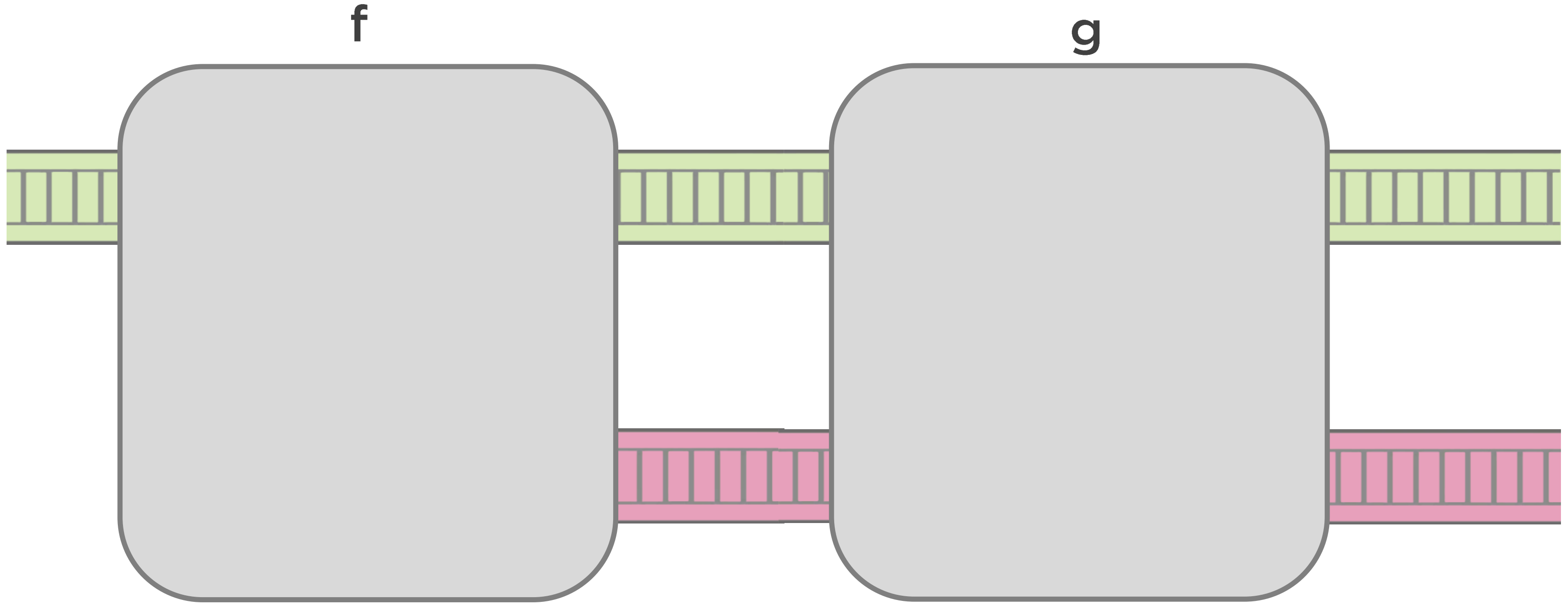




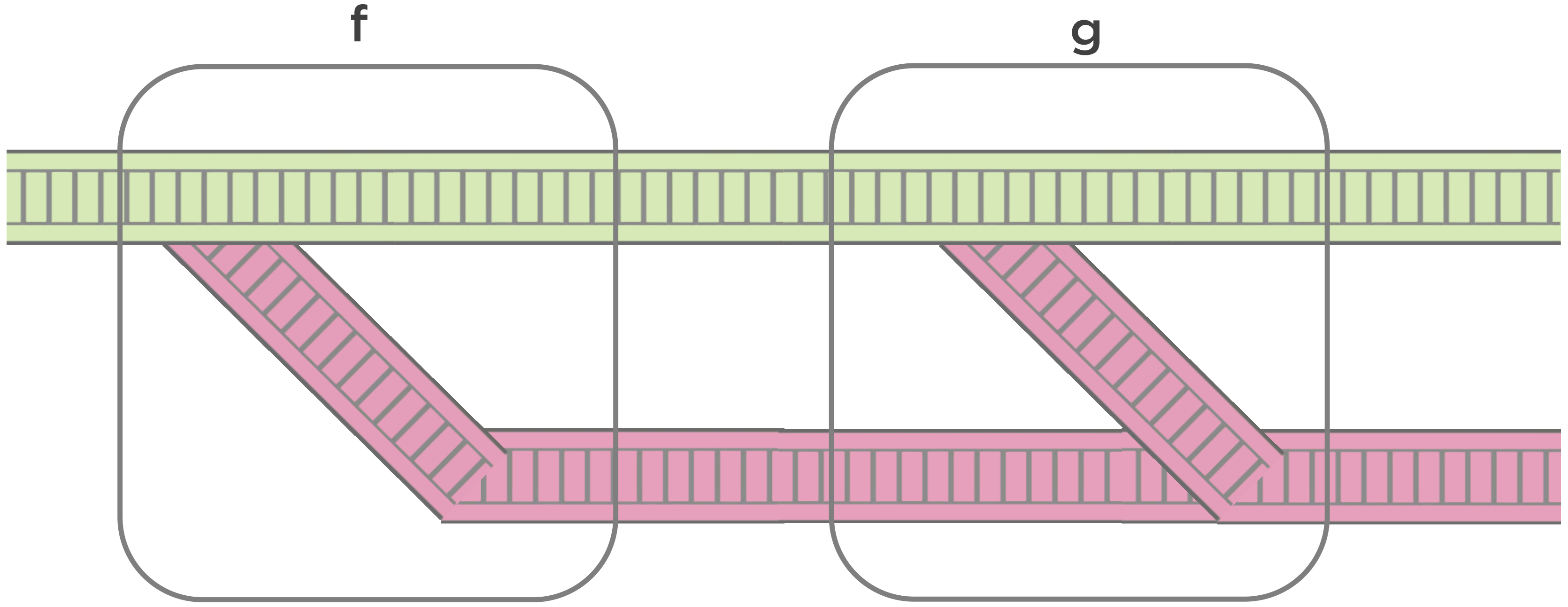
Railway Switches



Railway Switches

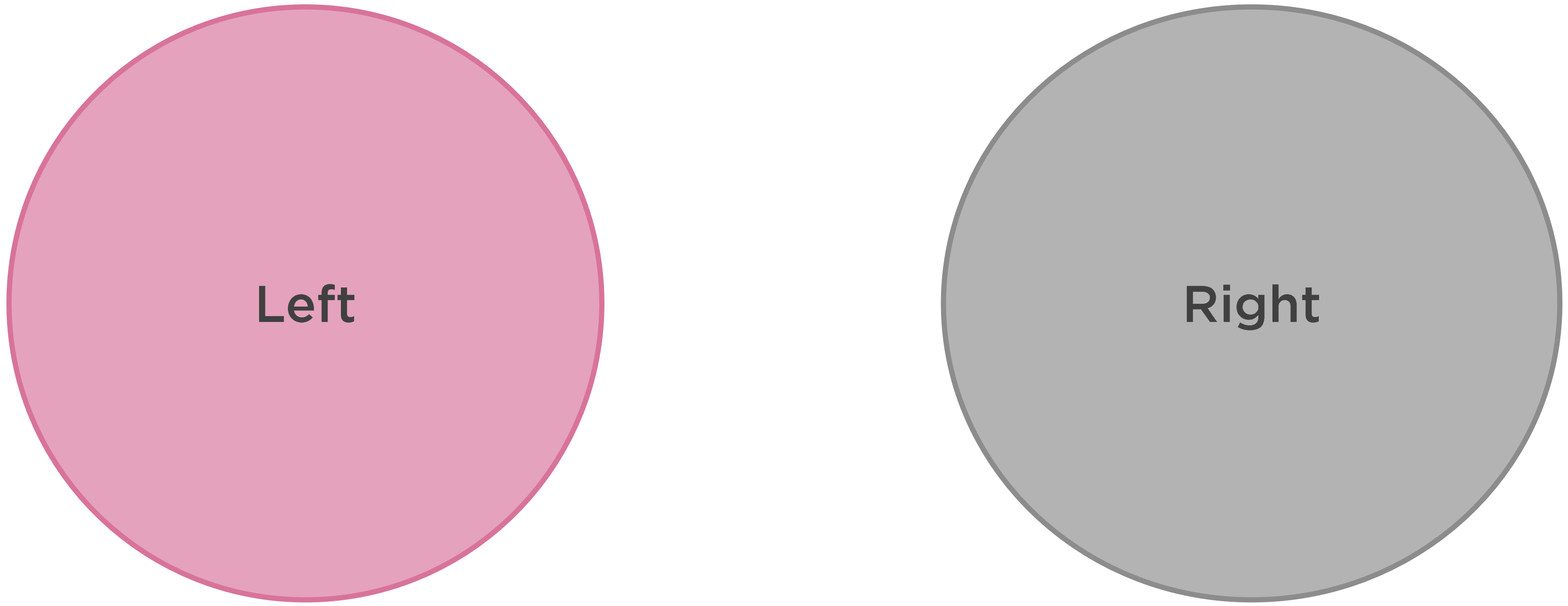


Railway Switches



The Either and Try Types

Either Type



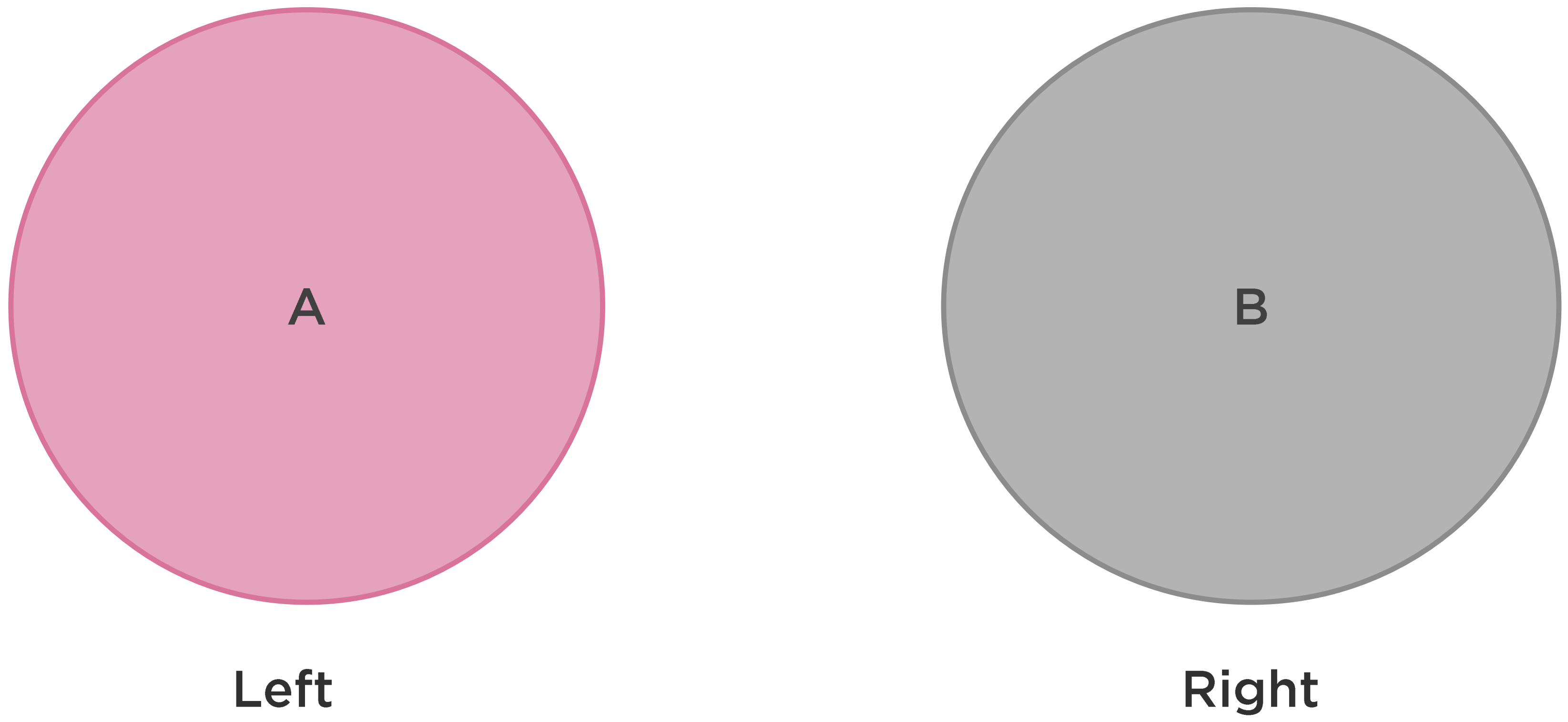
Contains **either** A or B

```
Either<A, B>
```

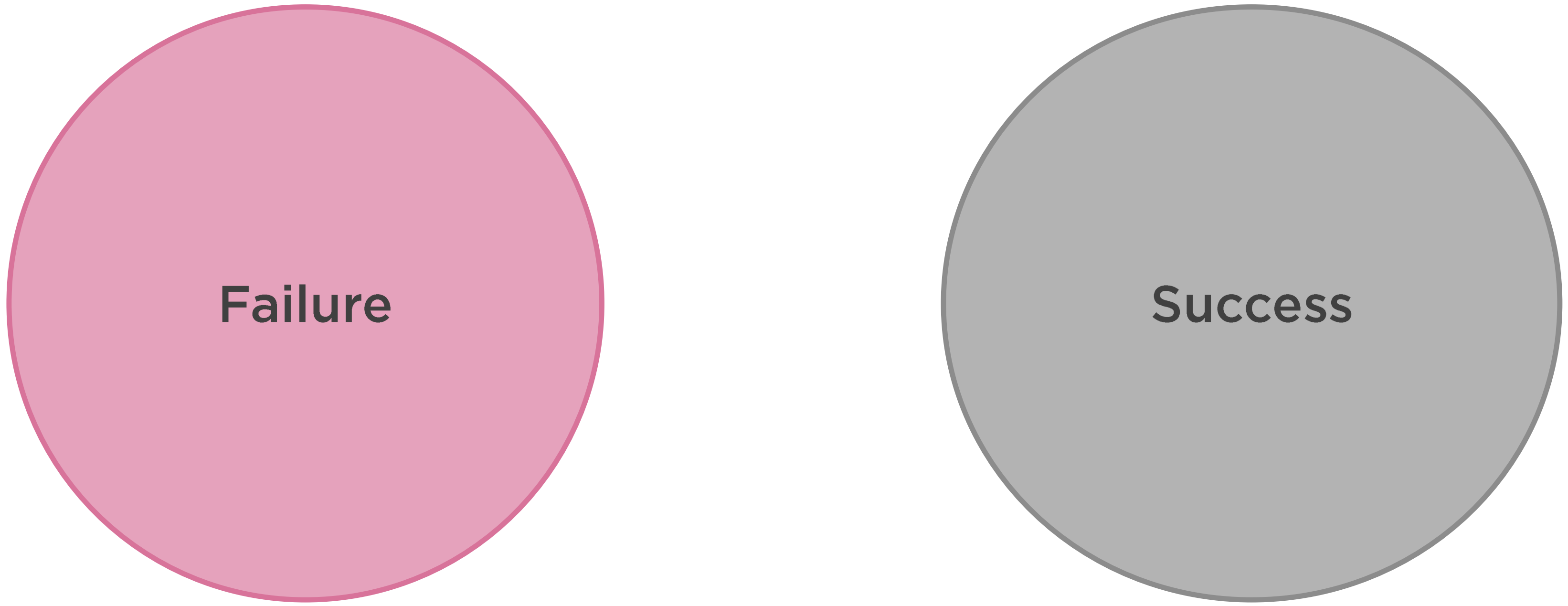
```
Tuple<A, B>
```

Contains **both** A and B

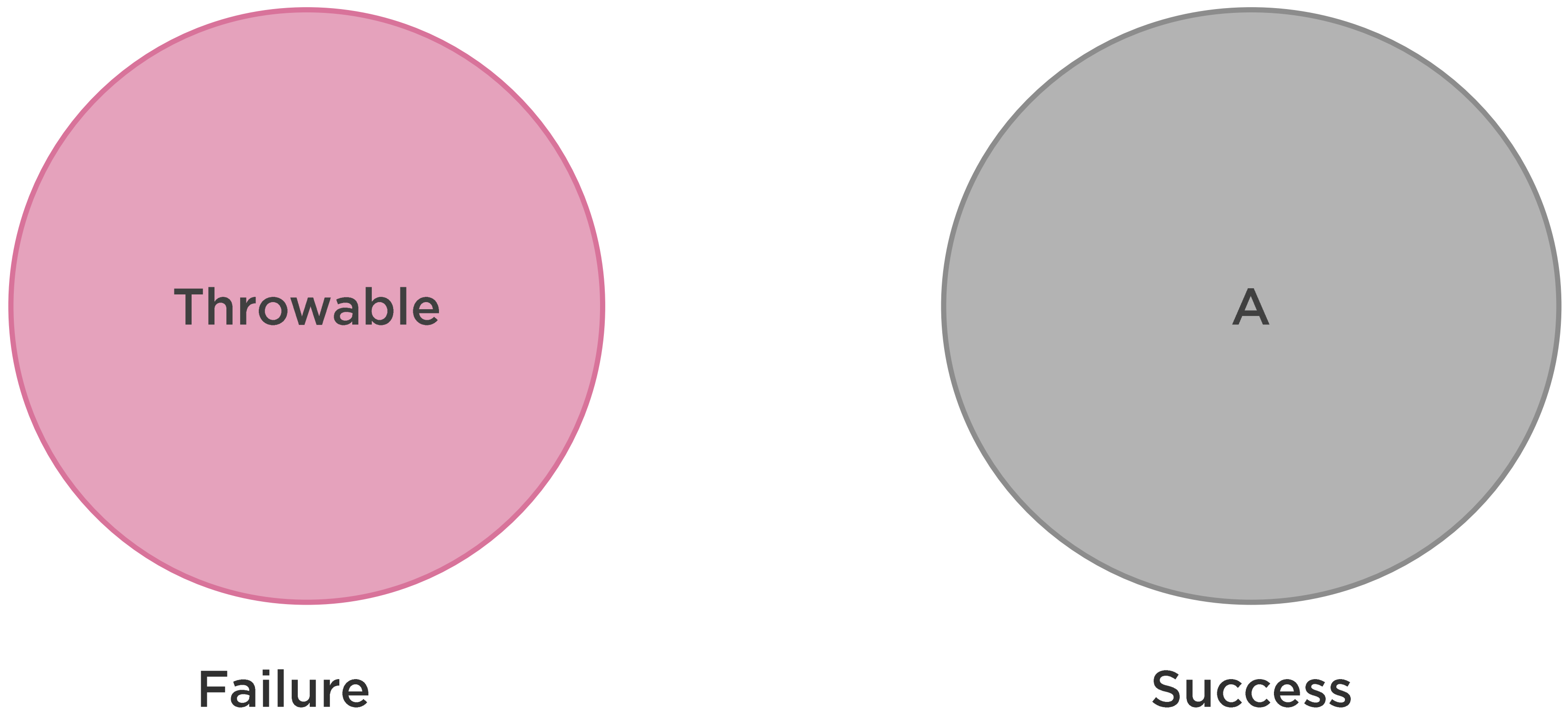
Either<A, B>



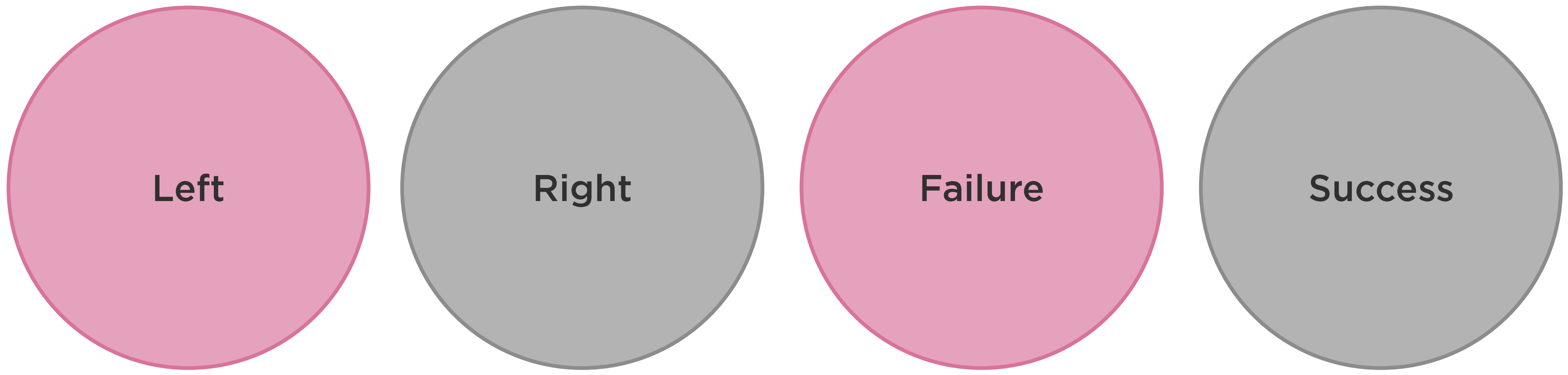
Try Type



Try<A>



Subtypes



Pass the error without doing anything

Pass the error without doing anything

The way to use these types is not by retrieving the value, but by composing functionality.

Implementing the Result Type

Using the Result Type

Things to Remember

Two types of functions

- Total functions
- Partial functions

To turn a partial function into a total function

- Change the domain, by creating a new type that groups all possible input values
- Or change the codomain, by creating a new type that groups all possible output values

Things to Remember

Railway-oriented programming

- A railway track represents a function
- And we can compose two functions by joining tracks
- But railways have switches for directing trains onto a different track
- These switches are represented by the success or failure outputs of the functions

Things to Remember

Either type

- Left
- Right

Try type

- Failure
- Success

Things to Remember

Extend the implemented Result type

- Get the exception from Failure
  - getError
  - orElseThrow
- onSuccess or onFailure

In the Next Module

Building container types