Lecture 8

Lambdas and Streams

Previously, in cs2030..

```java
class Square implements Function<Integer, Integer>
{
  public Integer apply(Integer x) {
    return x*x;
  }
}

applyList(list, new Square());
applyList(list, new Function<Integer,Integer>() {
  public Integer apply(Integer x) {
    return x * x;
  }
});
```

we know applyList expects a Function<Integer,Integer>

```
applyList(list, new Function<Integer,Integer>() {
  public Integer apply(Integer x) {
    return x * x;
  }
});
```

only one method is abstract in Function

```
applyList(list, x -> {
    return x * x;
  }
});
```

```
applyList(list, x -> x * x);
```

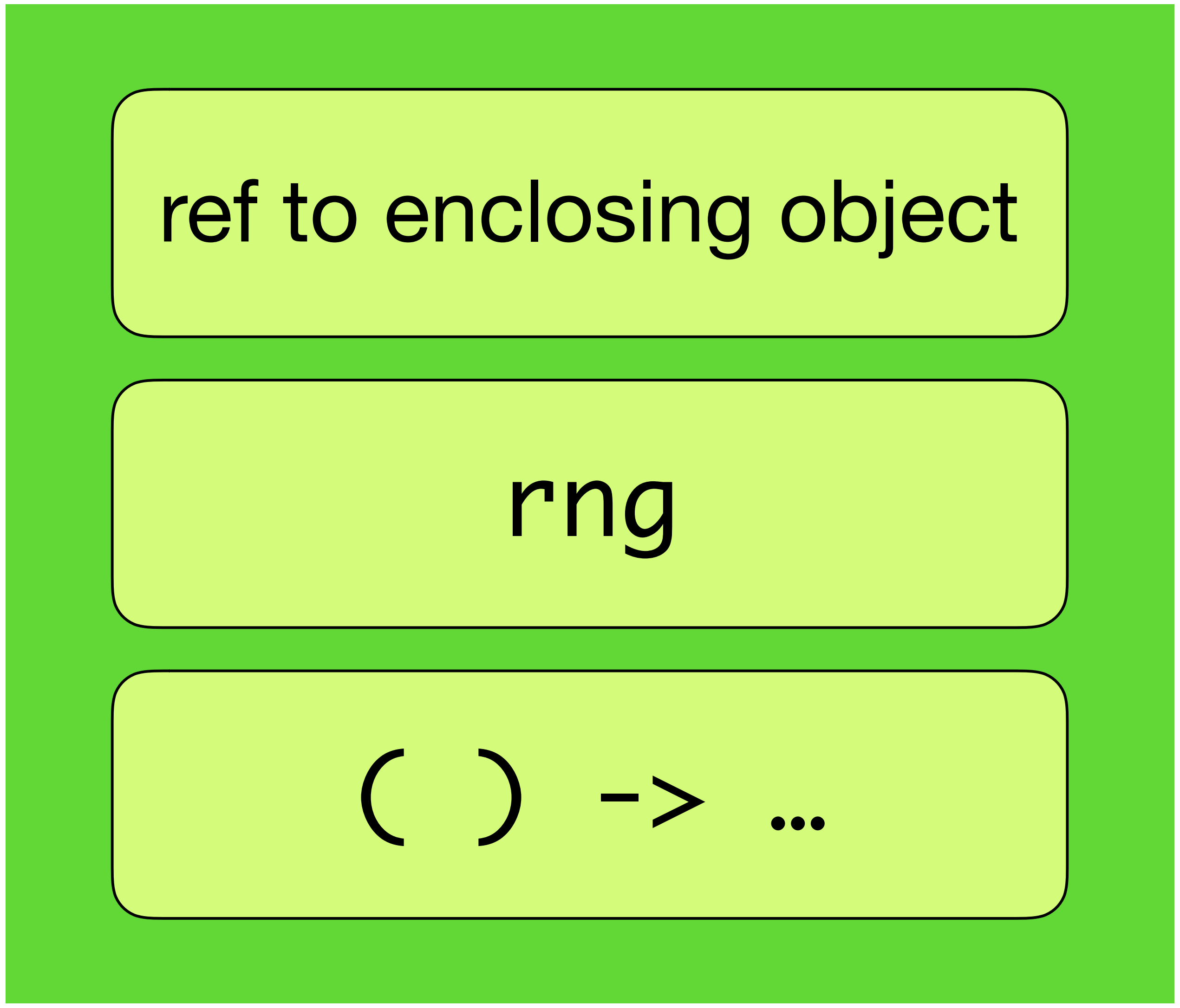
actually an anonymous class

Recap: an anonymous class can access:

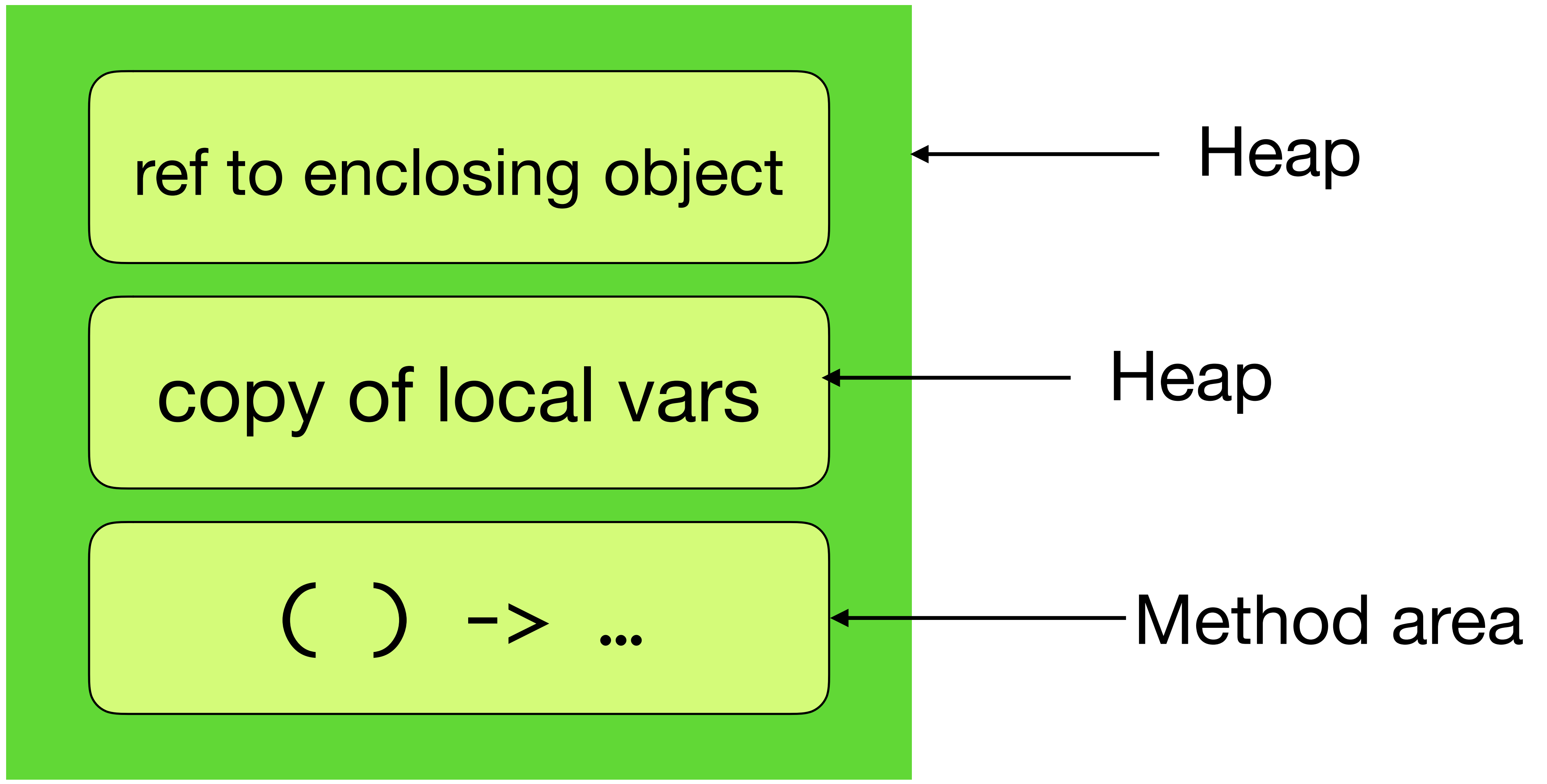
- final / eff. final local variables
- members of enclosing class

```
rng = new RandomGenerator(…)
Customer c = new Customer(
  () -> rng.genServiceTime()
);
```

Closure

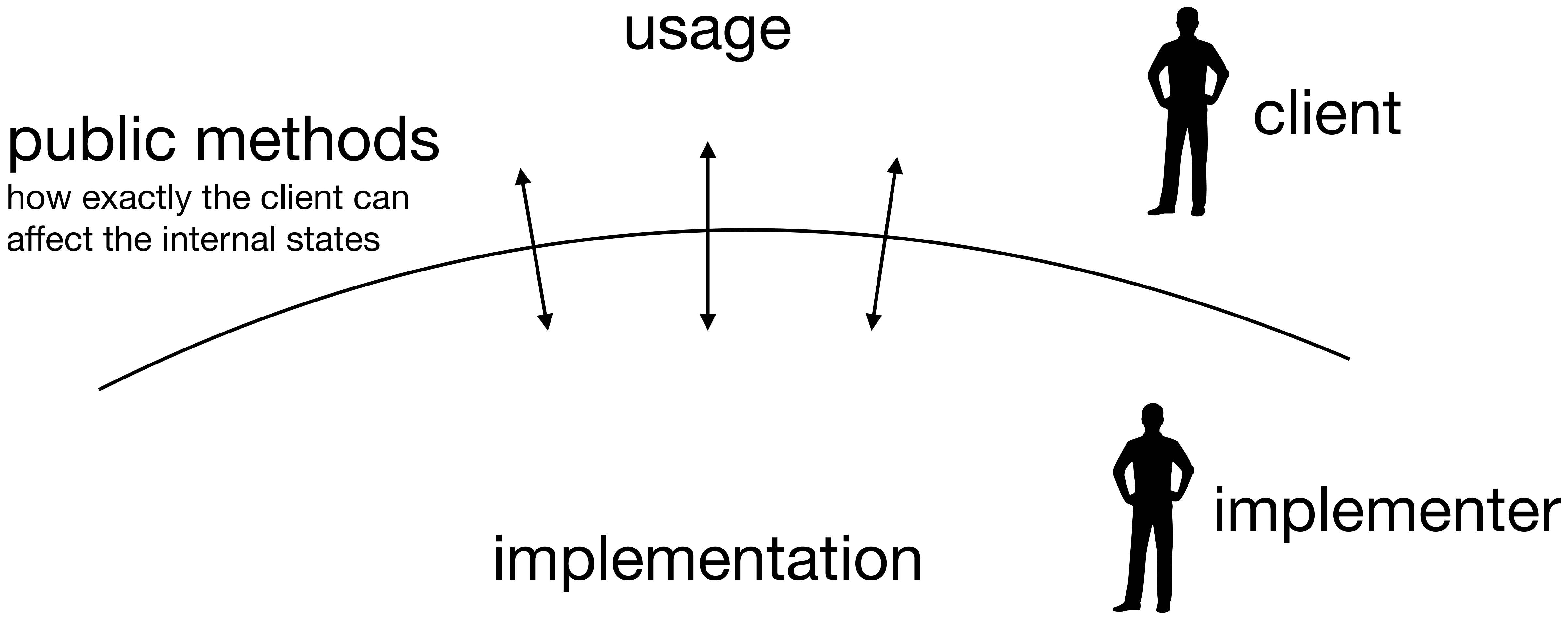


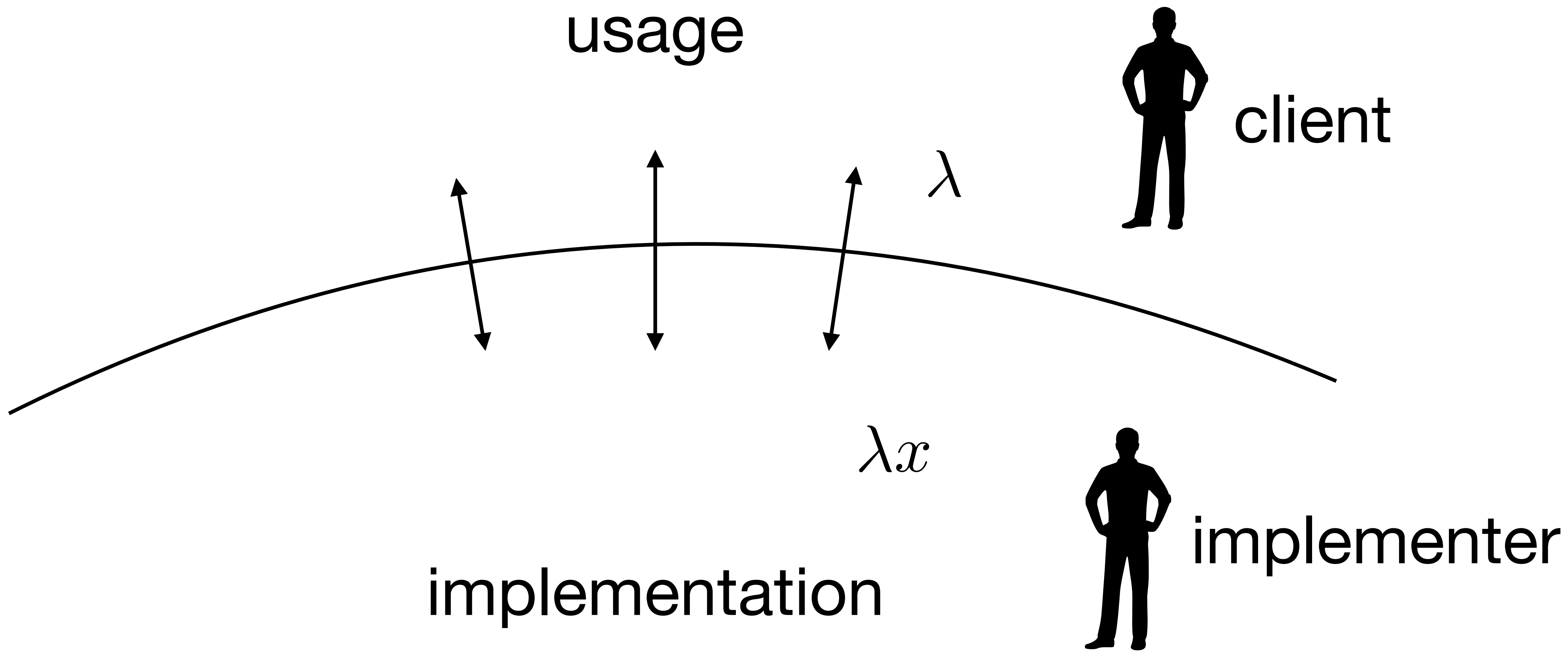
Memory Model

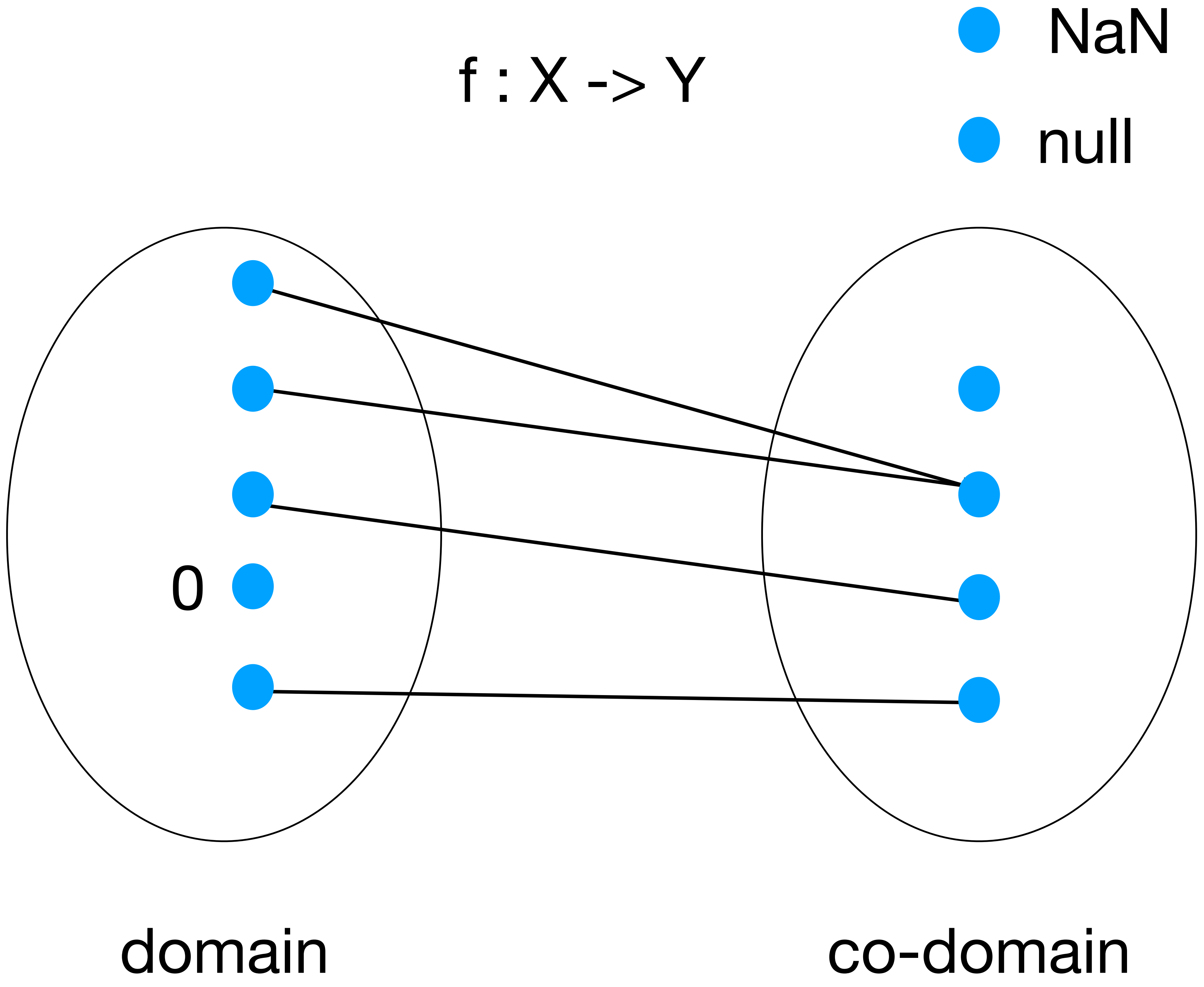


Function for Cross-Barrier Manipulation

Abstraction Barrier




usage
client
λ
λx
implementer
implementation

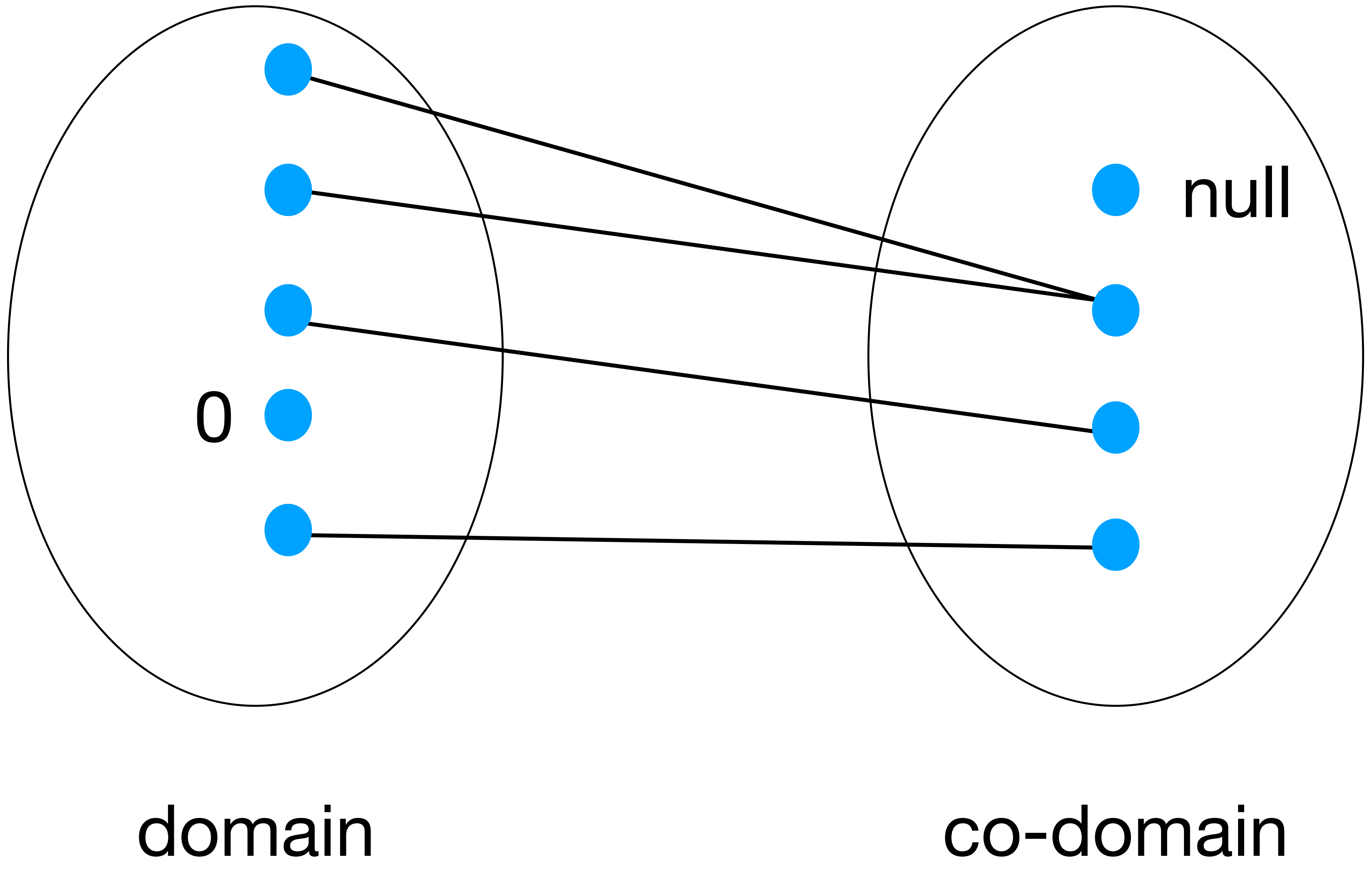

NaN
f : X -> Y
null
0
domain
co-domain

I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

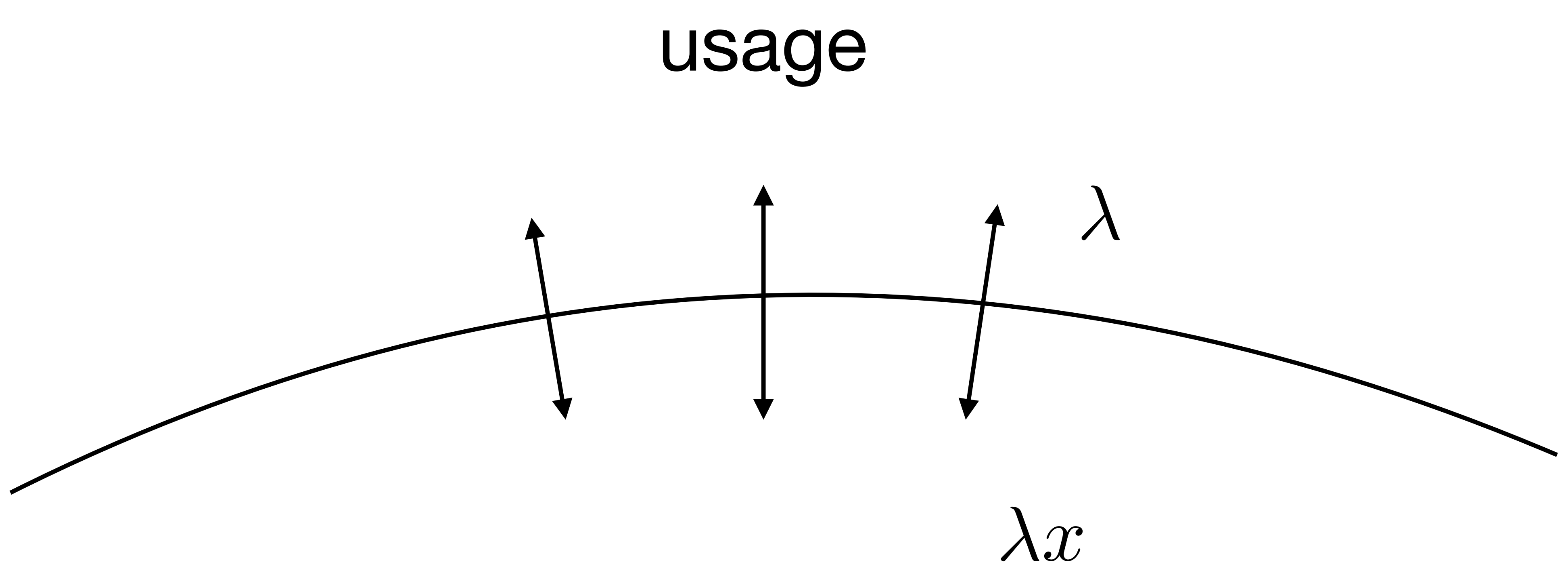
- Sir Tony Hoare

```
shop.findIdleServer()
    .serve(customer);
```

f : X -> Y



wrap a nullable reference in an `Optional` object



a reference or `null`

```
server = shop.findIdleServer();
if (server != null) {
  server.serve(customer);
}
```

```java
shop.findIdleServer()
    .ifPresent(s -> s.serve(customer))
```

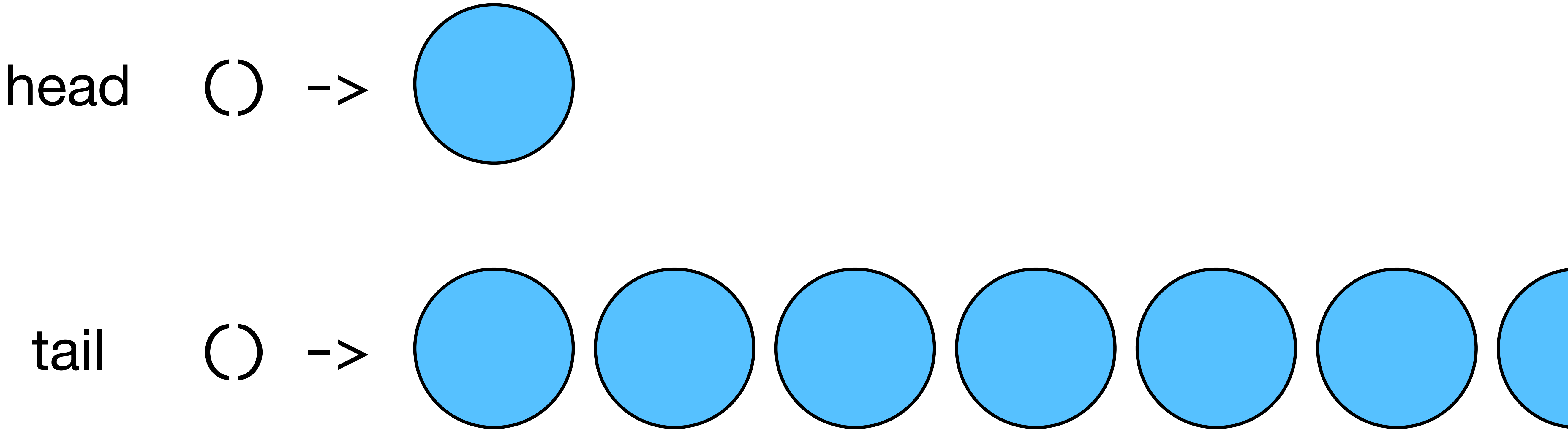
```
server = shop.findIdleServer();
if (server == null) {
  server = shop.findShortestQueue();
  if (server == null) {
    customer.leave();
  } else {
    server.serve(customer);
  }
}
```

```
shop.findIdleServer()
    .or(shop::findShortestQueue)
    .ifPresentOrElse(
        s -> s.serve(customer),
        customer::leave);
```

Function is Delayed Data

Infinite List:

A tale of two functions

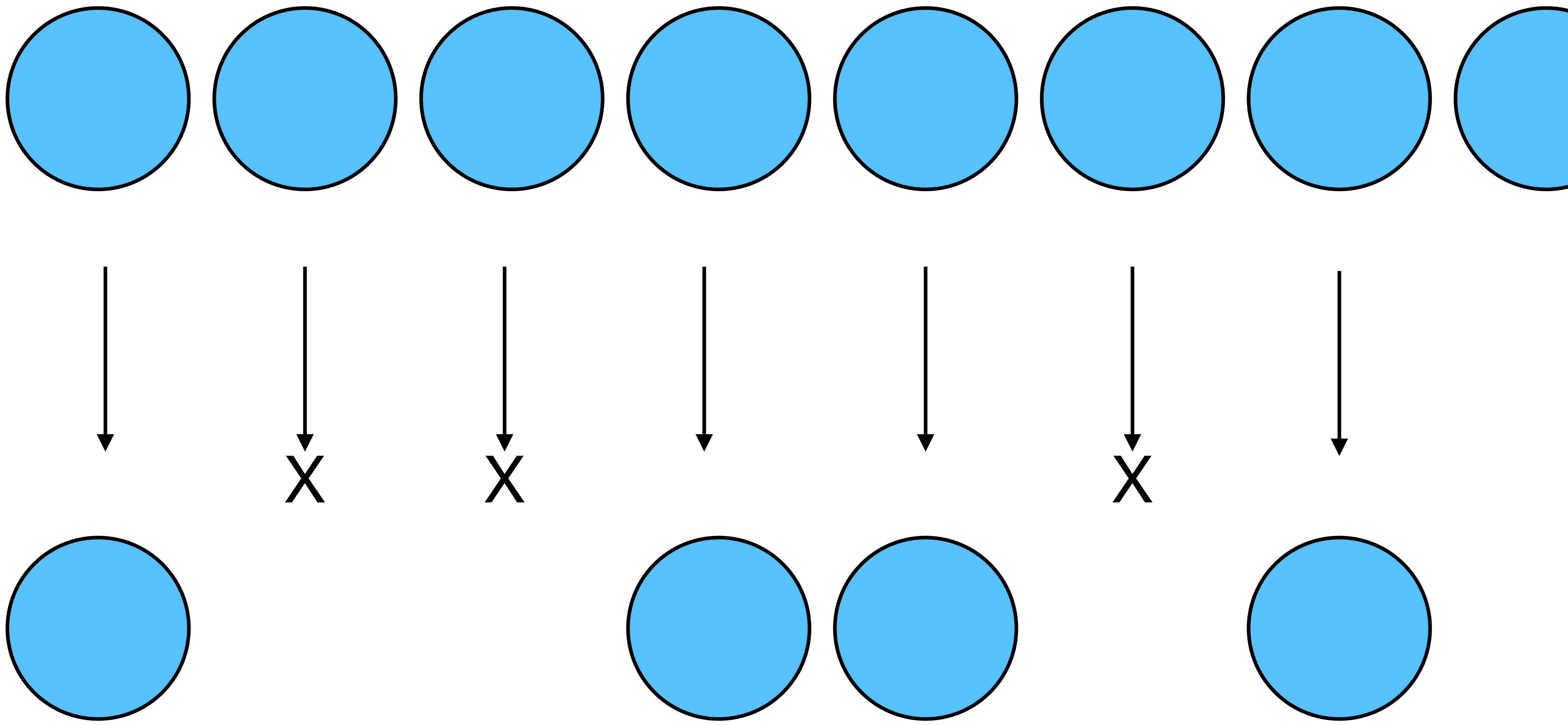

head () ->
tail () ->

Stream in Java 8

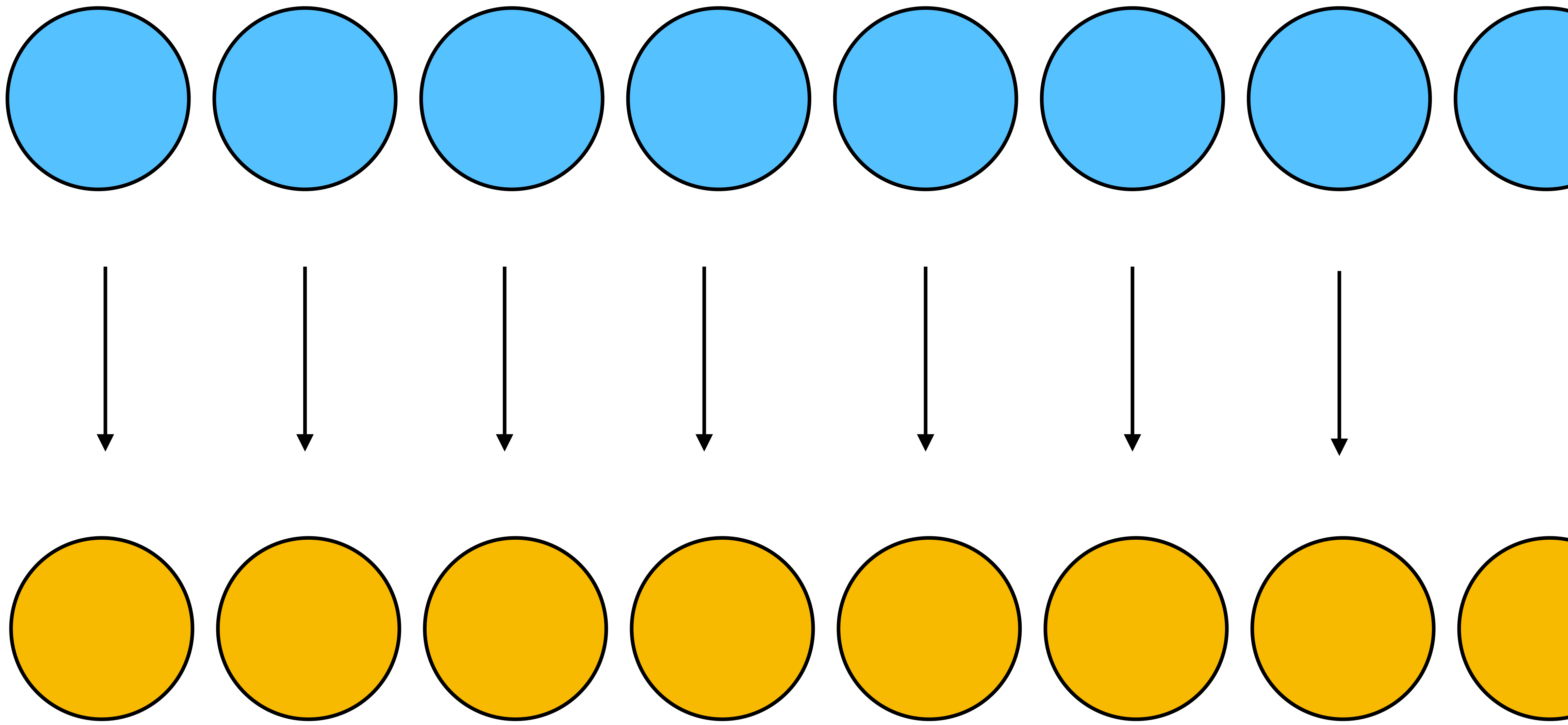
A Lazy (Possibly Infinite) List

and more..

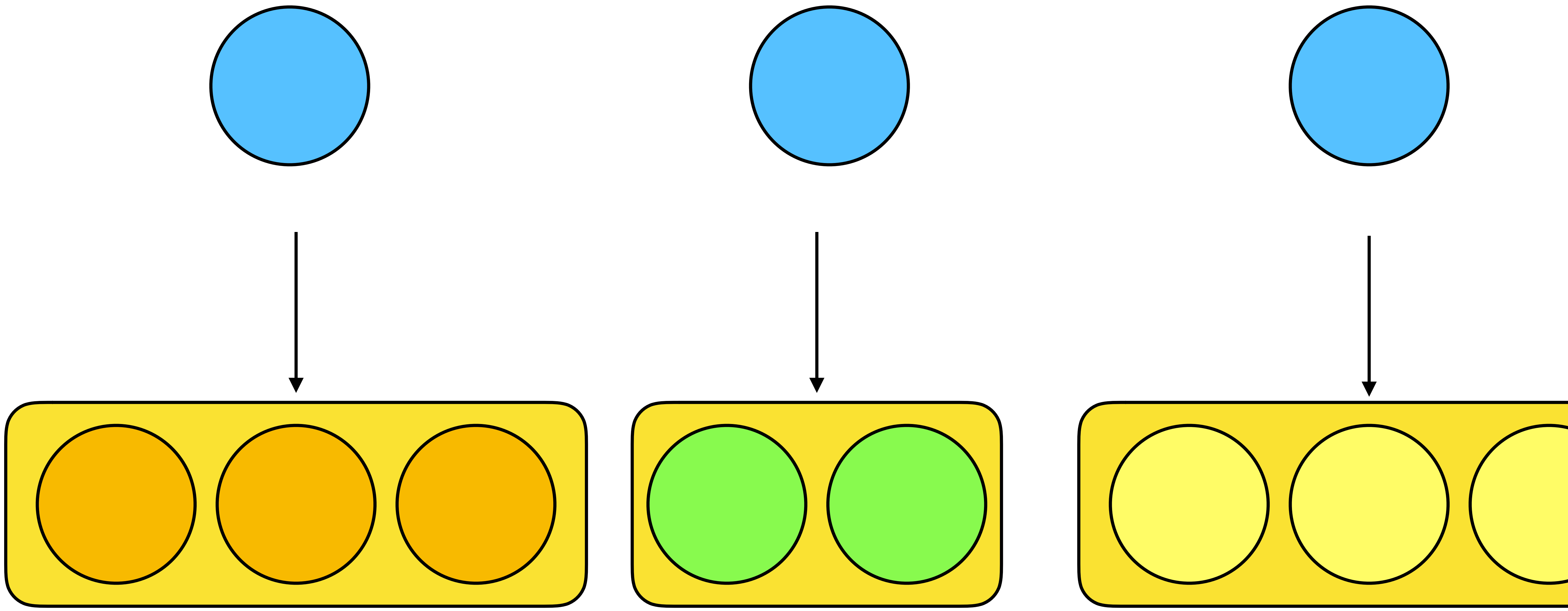
filter



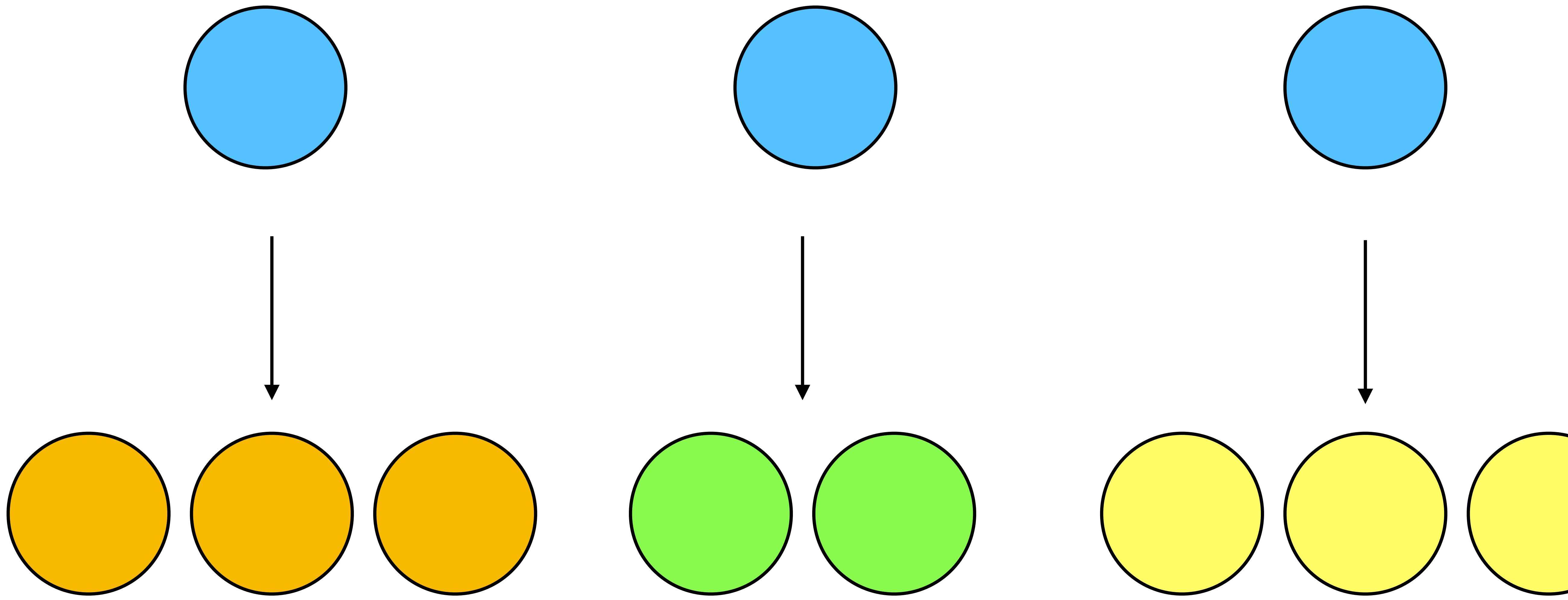
map



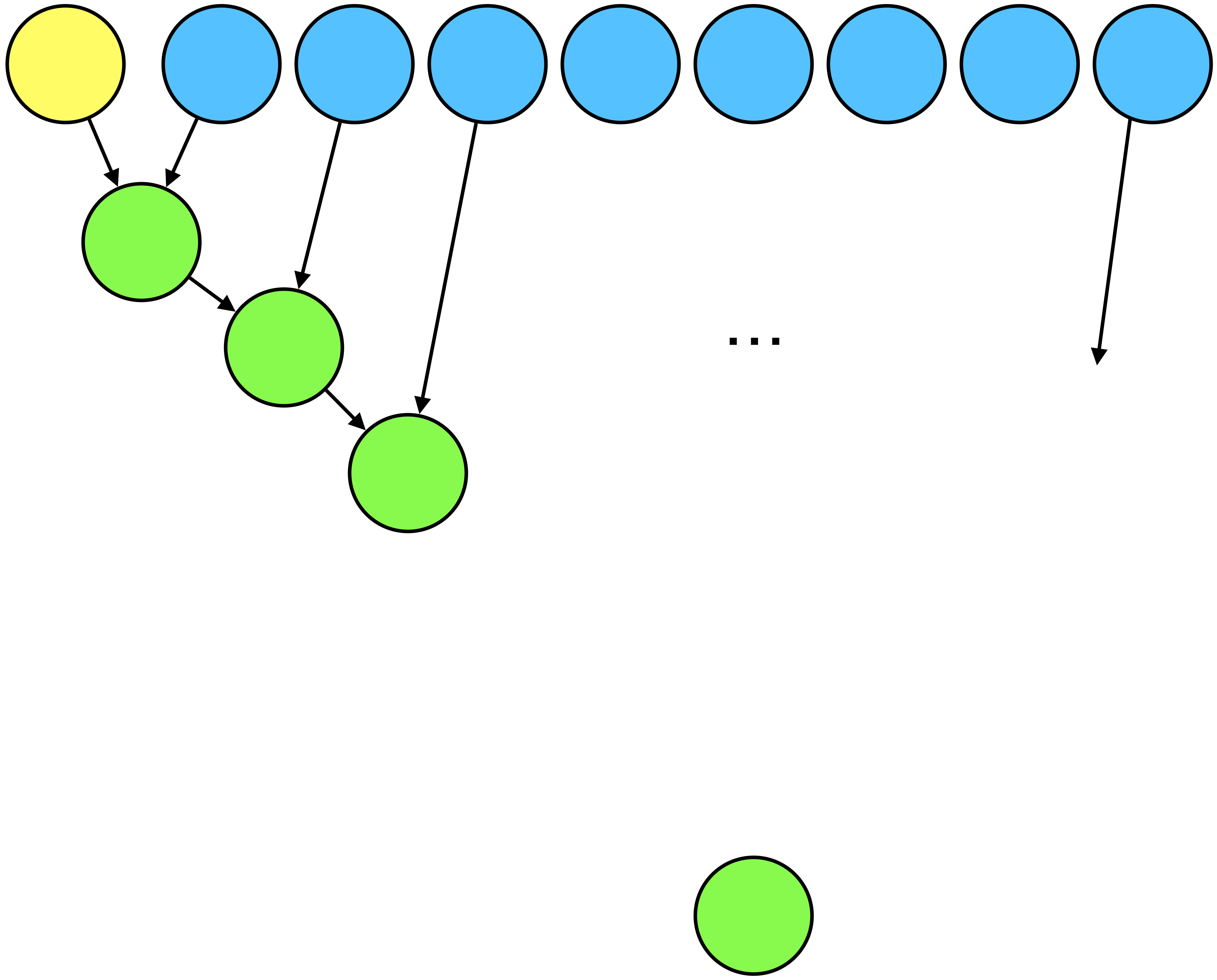
map



flatMap



reduce



```
boolean isPrime(int x) {
  for (int i = 2; i <= x-1; i++) {
    if (x % i == 0) {
      return false;
    }
  }
  return true;
}
```

```java
void fiveHundredPrimes() {
  int count = 0;
  int i = 2;
  while (count < 500) {
    if (isPrime(i)) {
      System.out.println(i);
      count++;
    }
    i++;
  }
}
```

```java
void fiveHundredPrime() {
    int count = 0;
    int i = 2;
    while (count < 500) {
        if (isPrime(i)) {
            System.out.println(i);
            count++;
        }
        i++;
    }
}
```

```java
void fiveHundredPrimes() {
  IntStream.iterate(2, x -> x+1)
      .filter(x -> isPrime(x))
      .limit(500)
      .forEach(System.out::println);
}
```