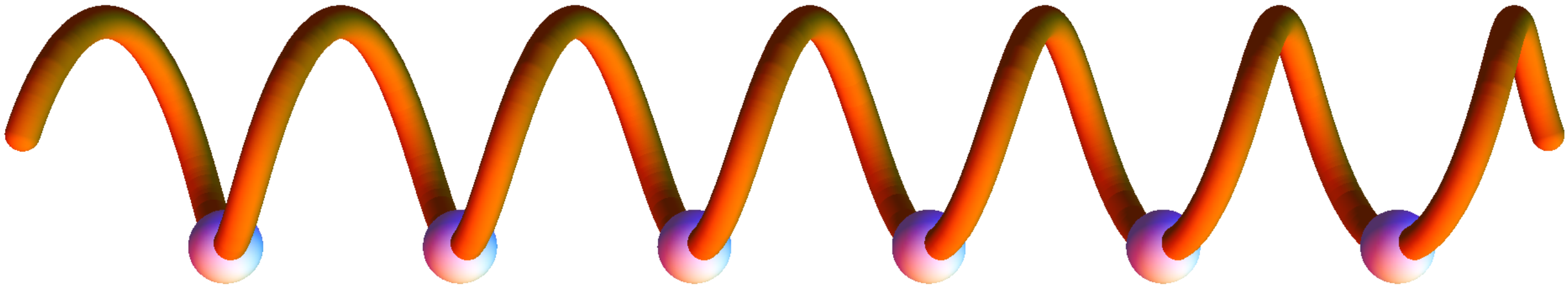
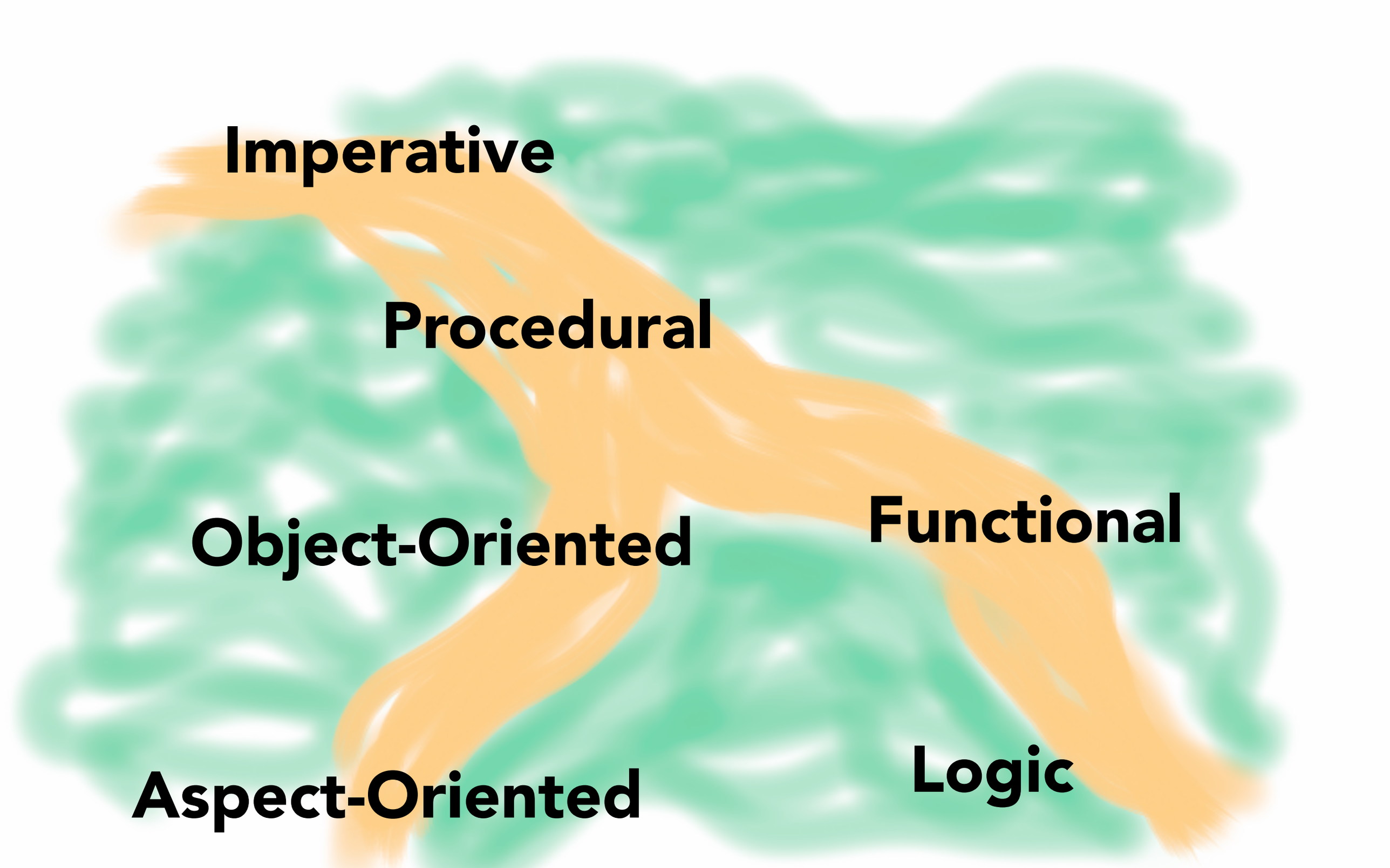


Functional principles for object-oriented development

@jessitron

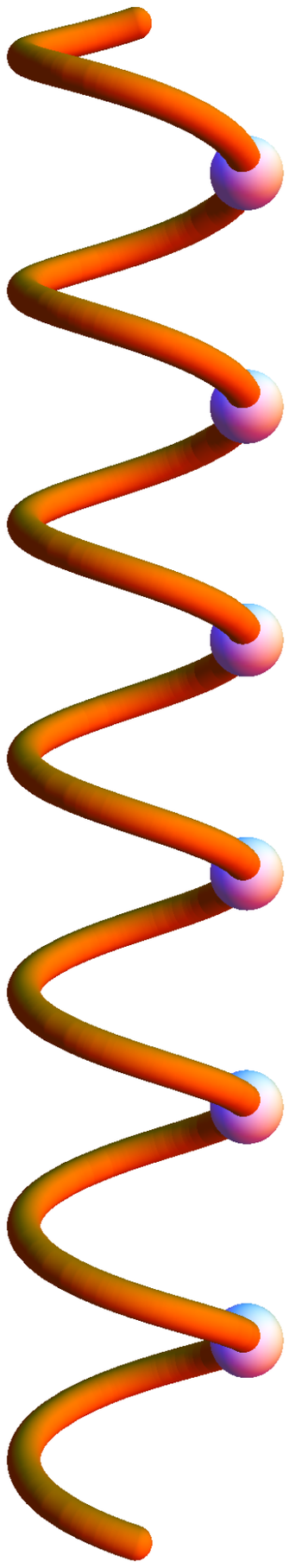

Imperative
Procedural
Object-Oriented
Functional
Aspect-Oriented
Logic

coridrew
@coridrew

Following

@jessitron Thanks! Every new tool & paradigm seems to help me at work in a surprisingly recursive & backward-compatible way =D

1
FAVORITE



Data In, Data Out

Verbs Are People Too

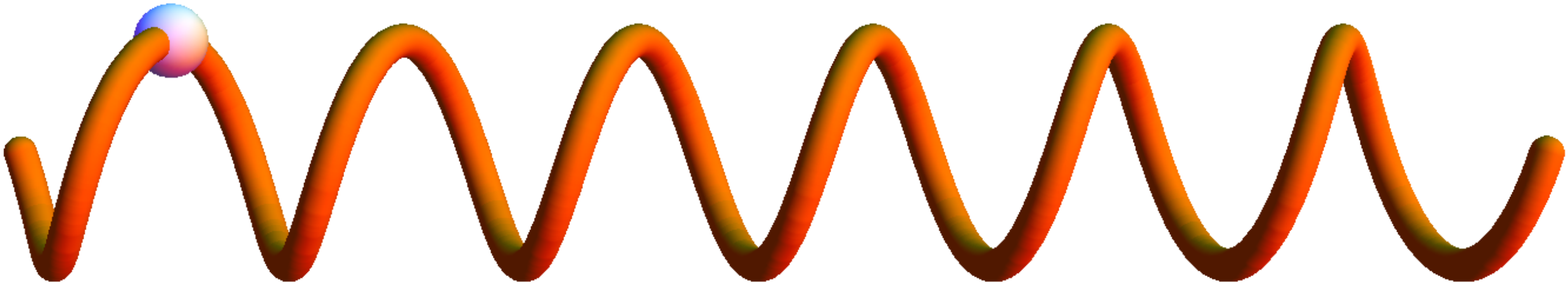
Immutability

Specific Typing

Declarative Style

Lazy Evaluation

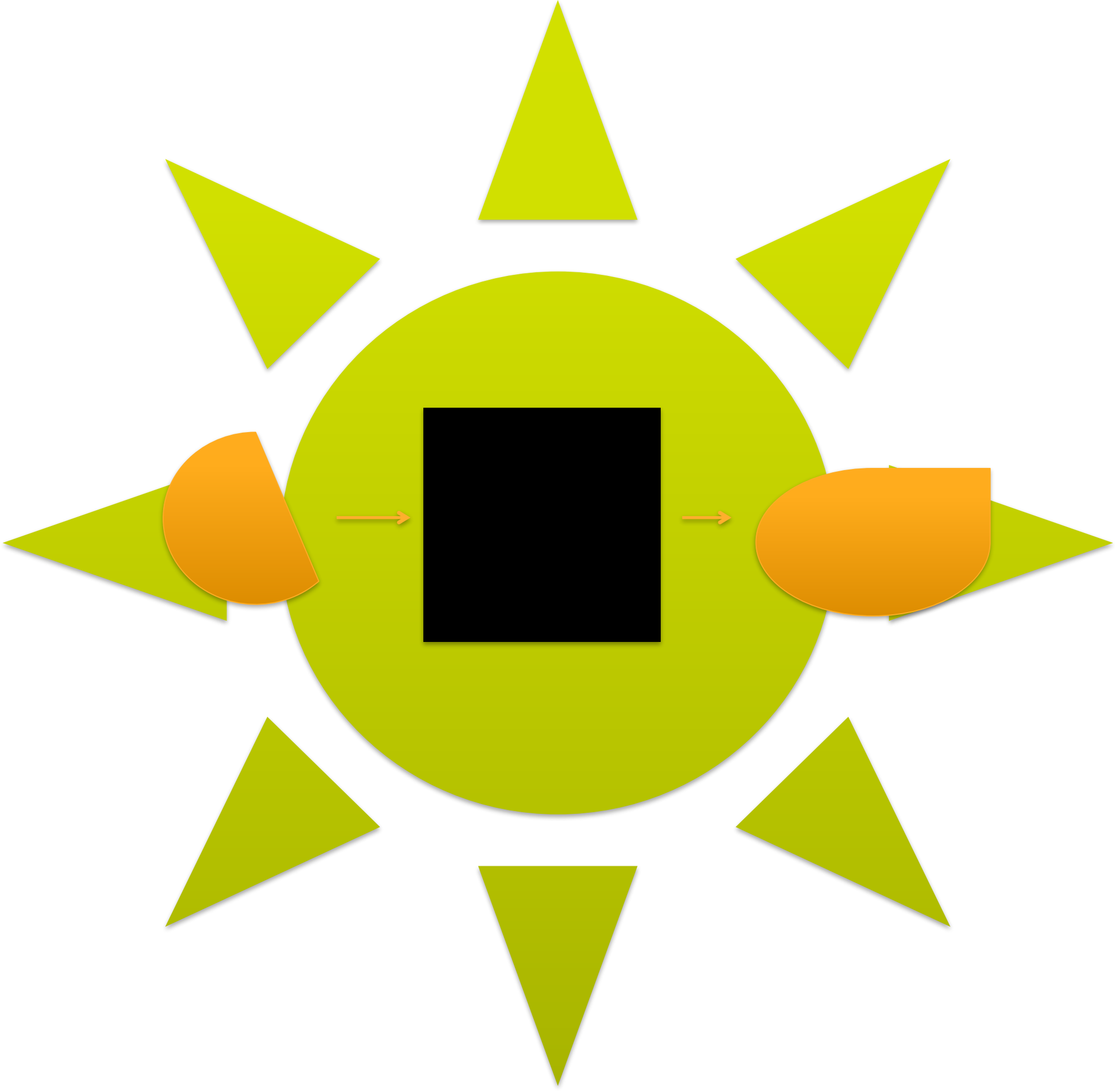
Data In, Data Out



access global state

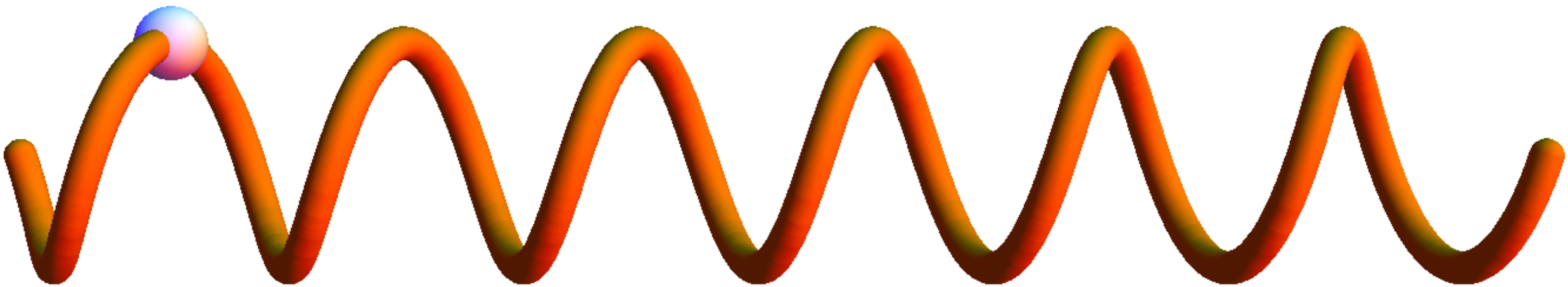
modify input

change the world

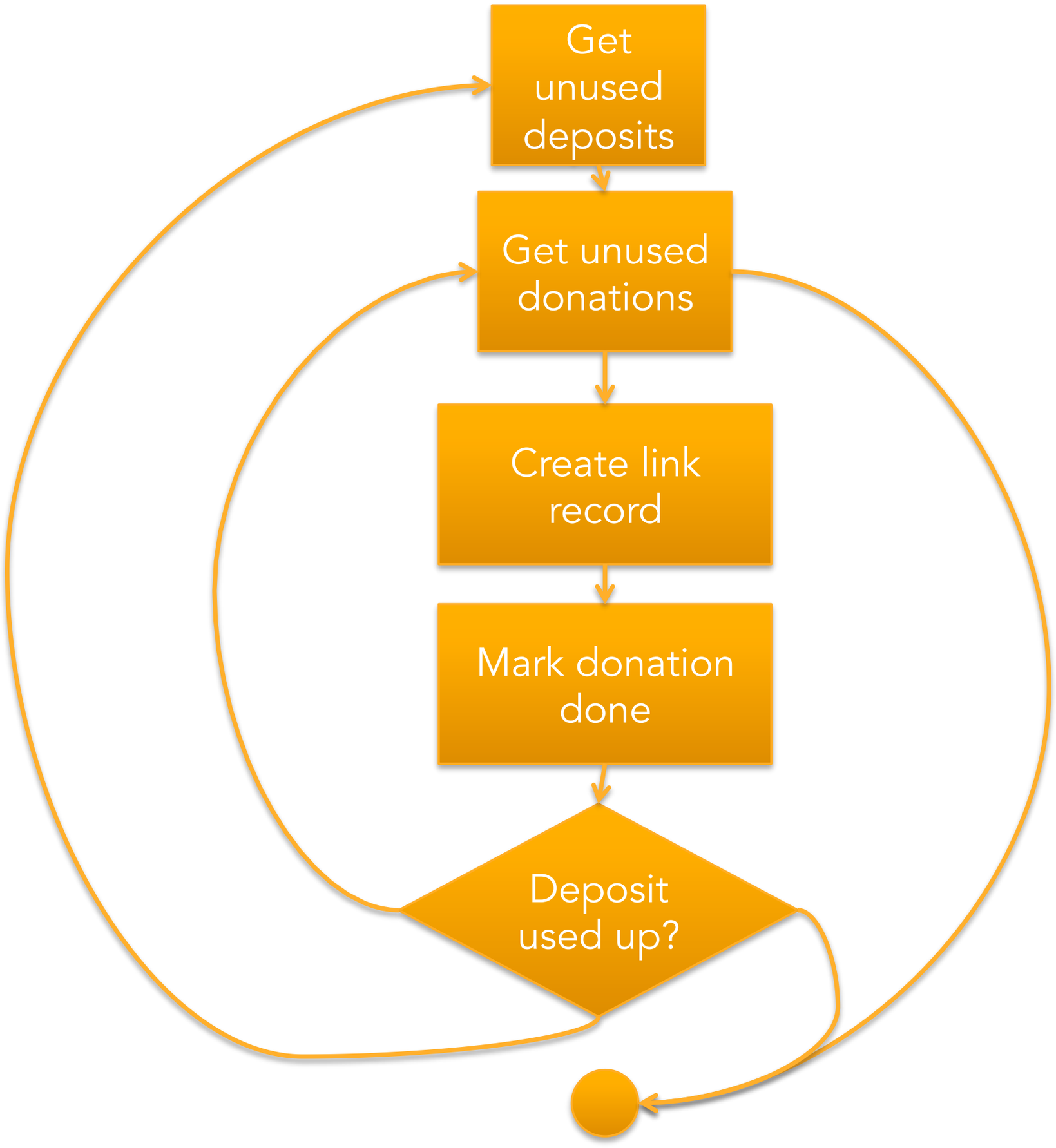


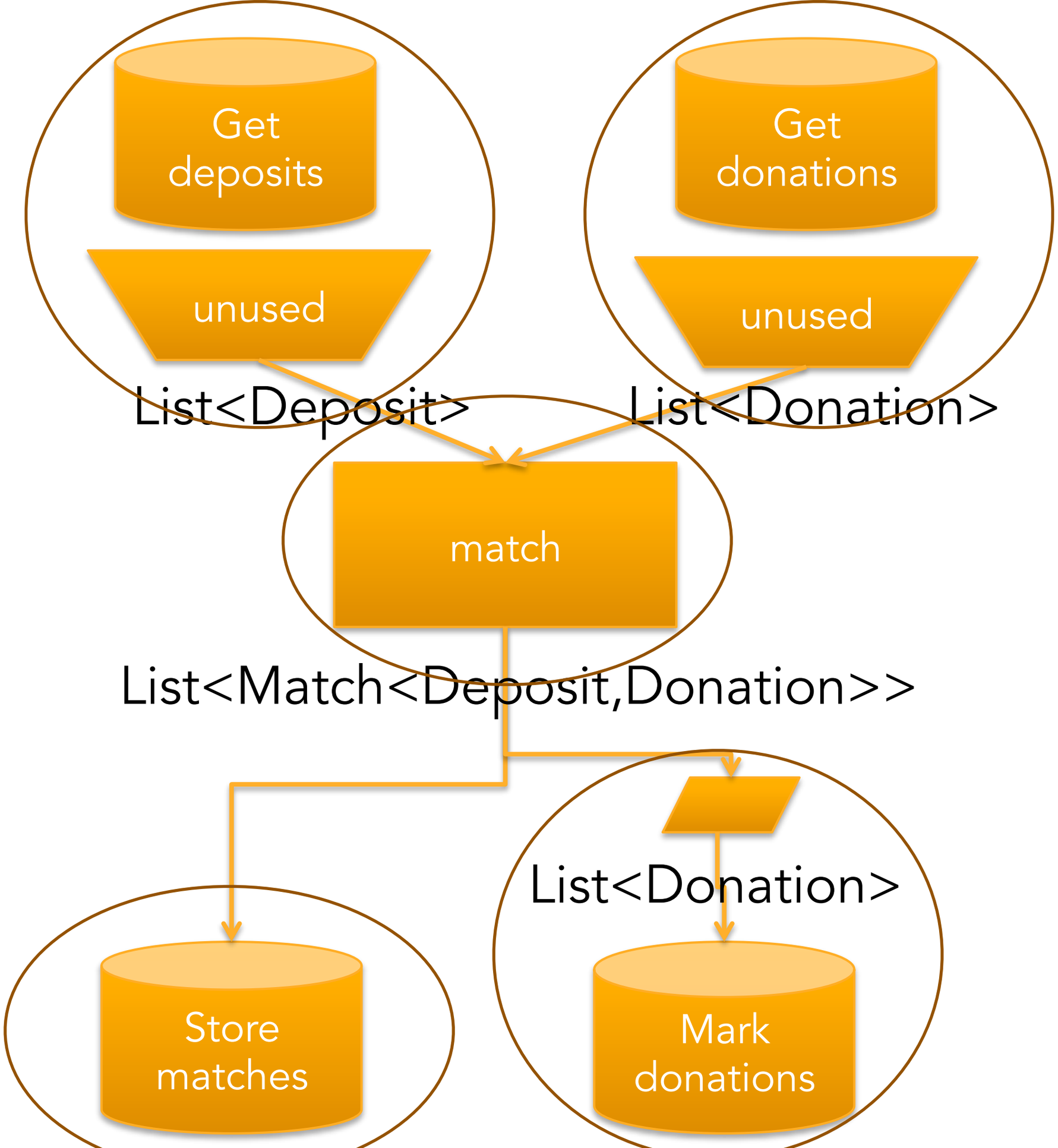
Testable

Easier to understand



the flow of data


Get unused deposits
Get unused donations
Create link record
Mark donation done
Deposit used up?


Get deposits
unused
Get donations
unused
List<Deposit>
List<Donation>
match
List<Match<Deposit,Donation>>
List<Donation>
Store matches
Mark donations

Problem easier, because we know each step of the way of data.

```
String password;
String email;

private boolean invalidPassword() {
    return !password.contains(email);
}
```

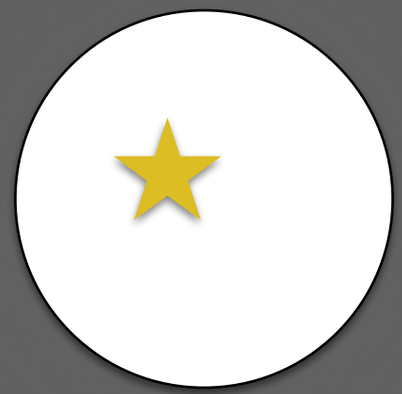
```
private boolean invalidPassword(
                                String password,
                                String email) {
    return !password.contains(email);
}
```

(Email, Password) -> boolean

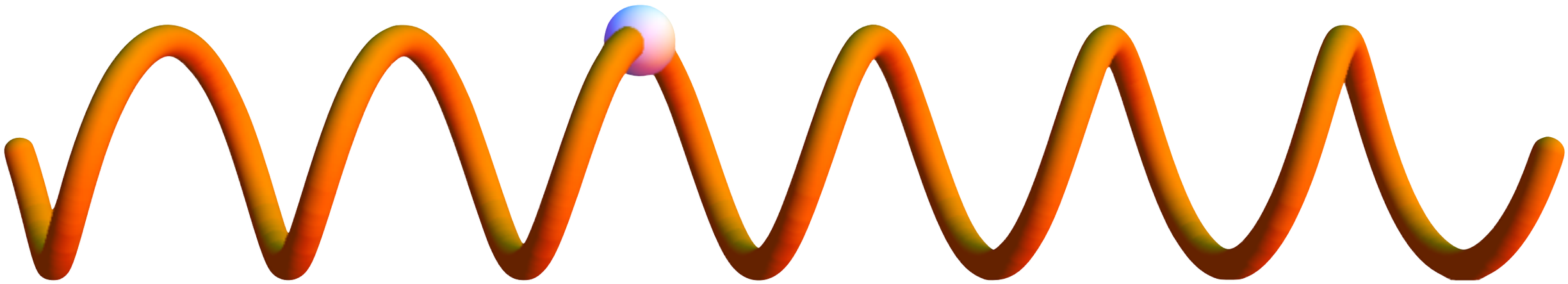
```
Response CreateAccount(UserDbService svc,
                       Request req,
                       Config  cfg) {
  …
  Account acc = constructAccount(…)
  AccountInsertResult r = svc.insert(acc)
  return respond(r);
}
```

Encapsulation

Isolation



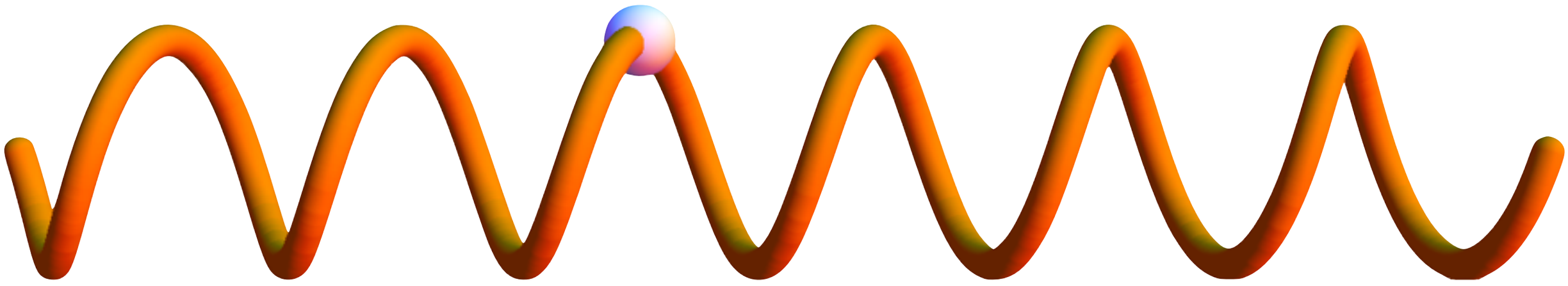
Verbs Are People Too



OnClick()

Command

Strategy



Java

```
class CostInflation implements
FunctionOverTime
{
    public float valueAt(int t) {
        return // some calculated value;
    }
}
```

Scala

```
val costInflation = Variable (
    “Cost Inflation”, t => Math.pow(1.15,t))
```

```
val seasonalVolume = Array(0.95,0.99,1.01,…)

val seasonalAdjustment = Variable(“Season”,
      t => seasonalVolume(t))
```

C#

```
private readonly Func<int, double> calc;

public Func<int, double> ValueAt
{
    get { return calc; }
}
```

C#

```
var inflation = new Variable(
                        “Inflation”,
                        t => Math.Pow(1.15,t));

inflation.ValueAt(3);
```

Java

```
Variable inflation = new Variable(
                         “Inflation”,
                         t -> Math.Pow(1.15,t));

inflation.valueAt(3);
```

Java 8

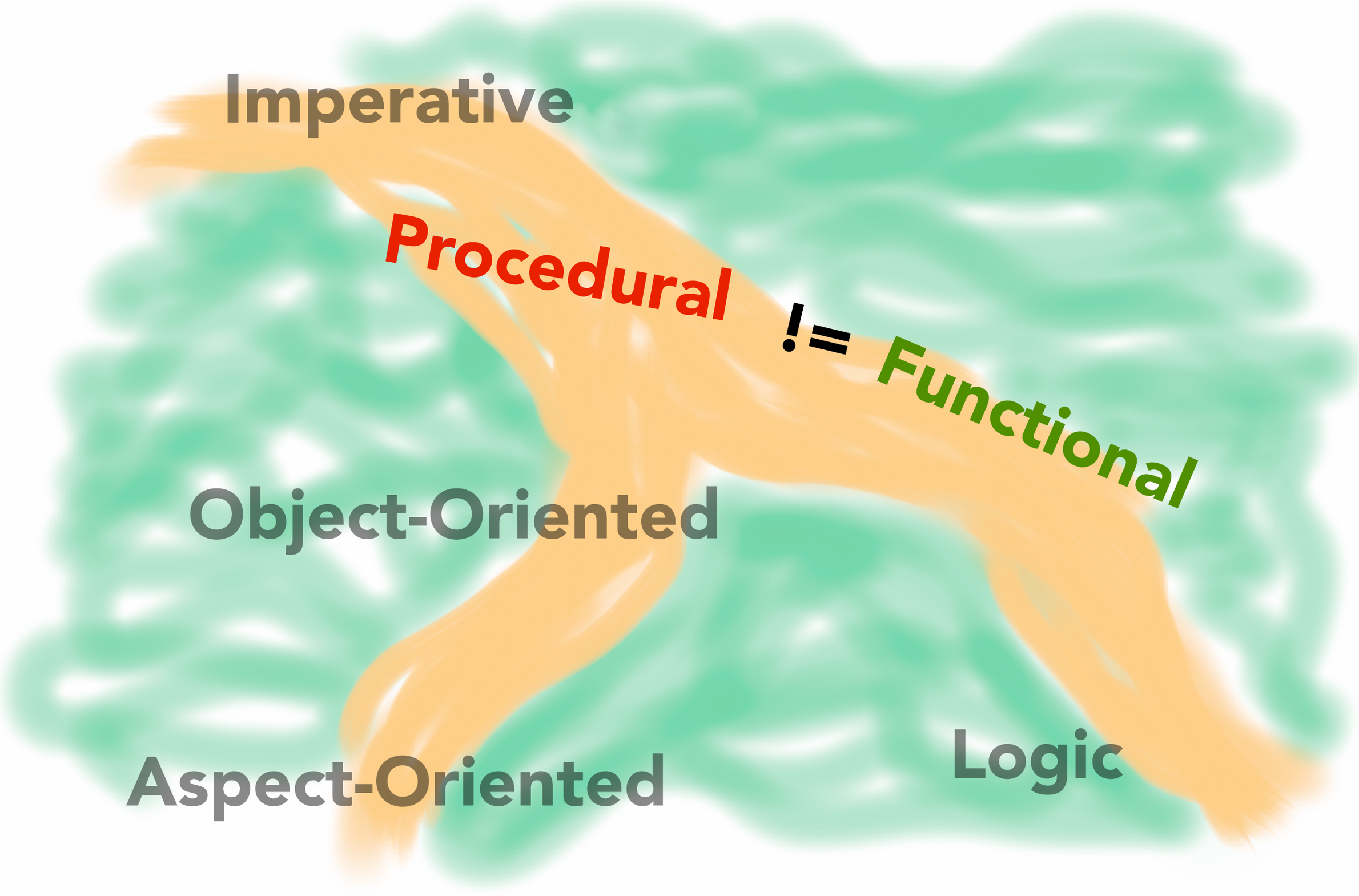
```
Variable inflation = new Variable(
                         “Inflation”,
                         t -> Math.Pow(1.15,t));

inflation.valueAt(3);
```

Java 8

```
class Variable implements FunctionOverTime
{
    private final Function<Int, Double> vals;
    …
    public float valueAt(int t) {
        return vals.apply(t);
    }
}
```

Scala

```
Int -> Double
```


Imperative
Procedural
!=
Functional
Object-Oriented
Aspect-Oriented
Logic

Java 6

```
Variable inflation = new Variable(“Inflation”,
    new Function<Integer, Double>() {
       @Override
       public Double apply(Integer input)
       {
          return Math.pow(1.15, input);
       }
});
```

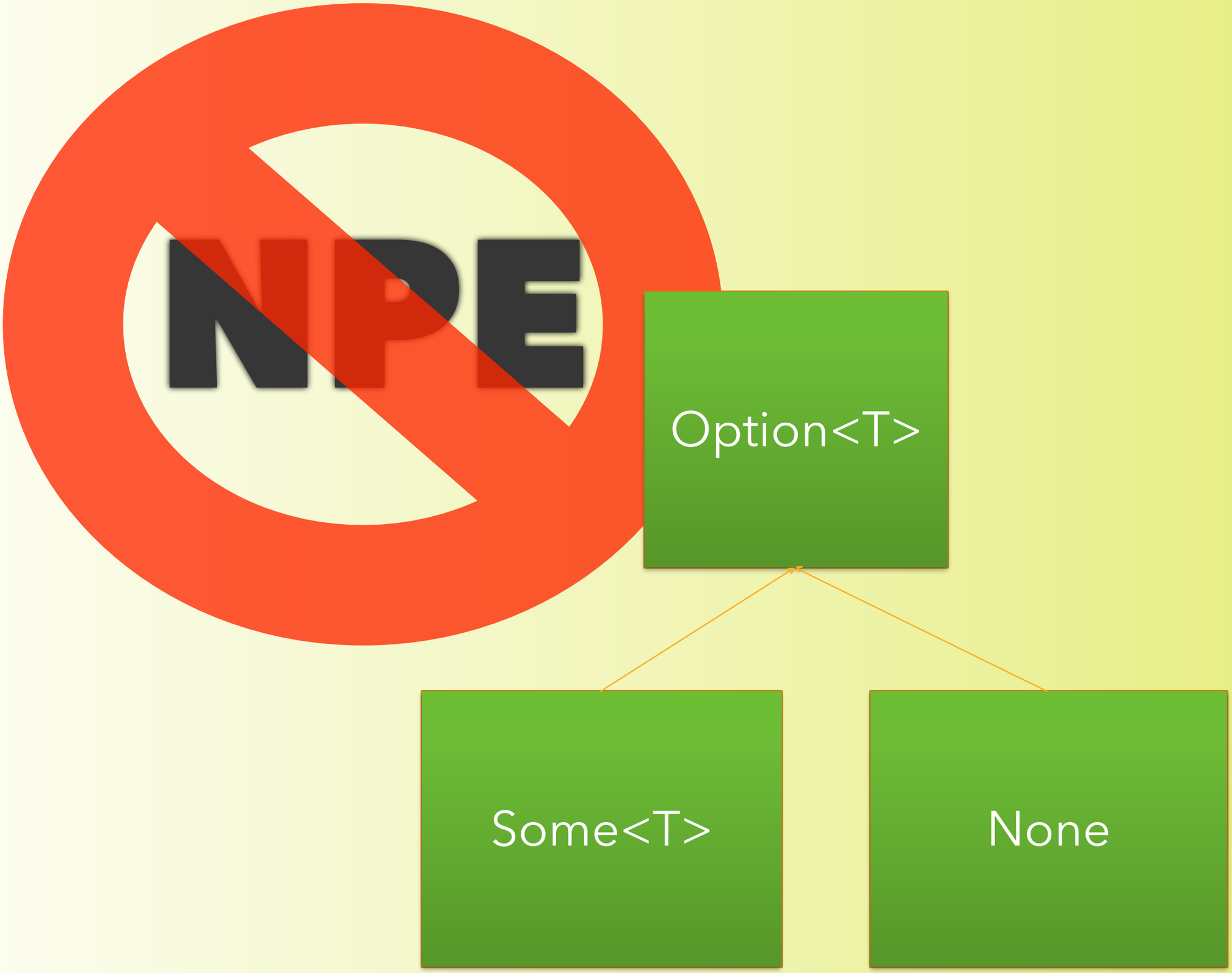
OnClick()

release ( )

Idempotence


OPEN
CLOSE
1
2

this talk is brought to you by… the Option type!

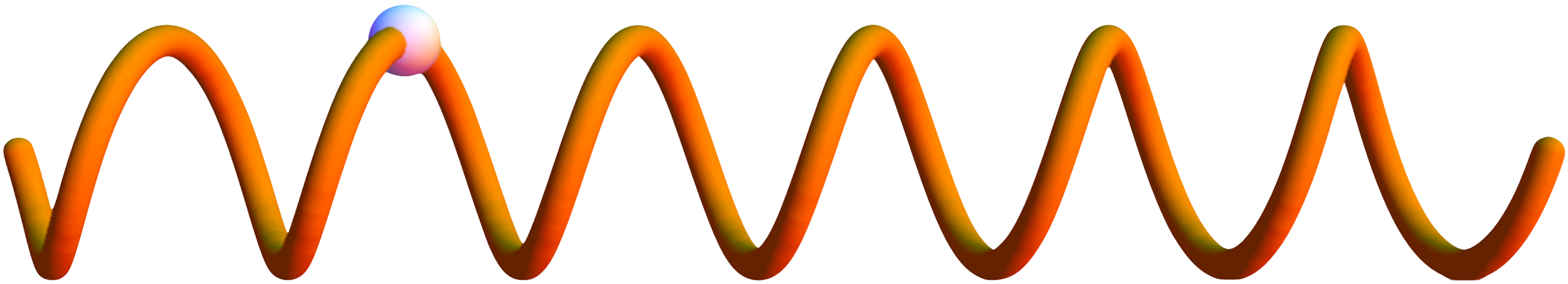


FSharpOption<T>

Optional<T>

… because null is not a valid object reference.

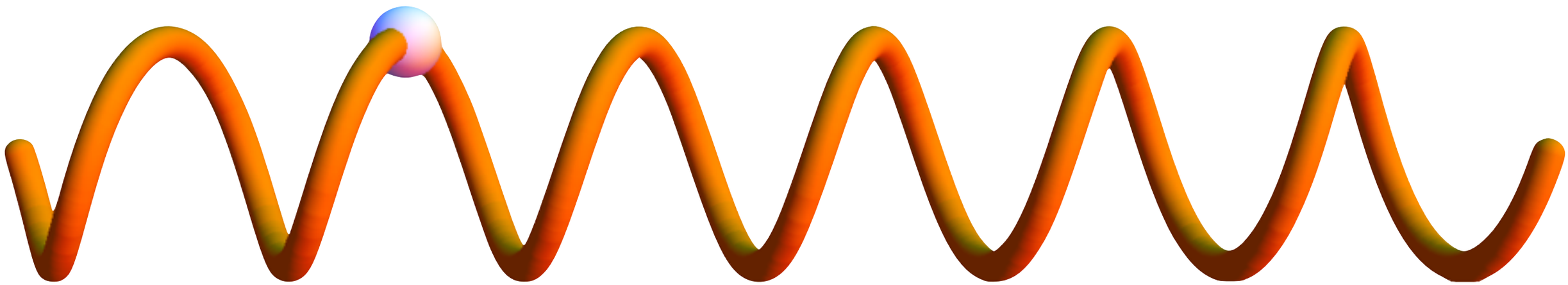
Immutability



String

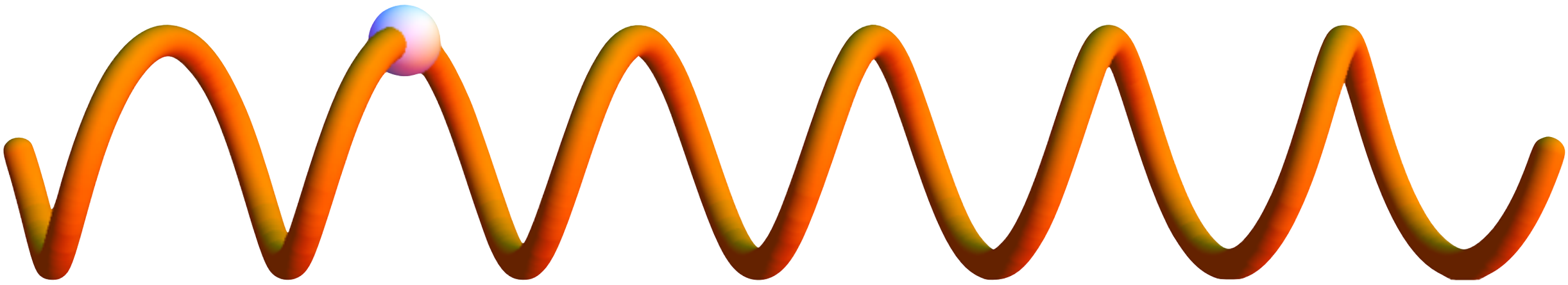
Effective Java

Effective C#



Concurrency

fewer possibilities



Scala

```
val qty = 4   // immutable

var n = 2     // mutable
```

F#

```
let qty = 4                  // immutable
let mutable n = 2 // mutable
n = 4                              // false
n <- 4                 // destructive update
```

C#: easy

```
public struct Address {
    private readonly string city;

    public Address(string city) : this() {
        this.city = city;
    }

    public string City {
        get { return city; }
    }
...}
```

Java: easy

```
public class Address {
    public final String city;

    public Address(String city) {
        this.city = city
    }

...}
```

Java: defensive copy

```
private final ImmutableList<Phone> phones;

public Customer (Iterable<Phone> phones) {
  this.phones = ImmutableList.copyOf(phones);
}
```

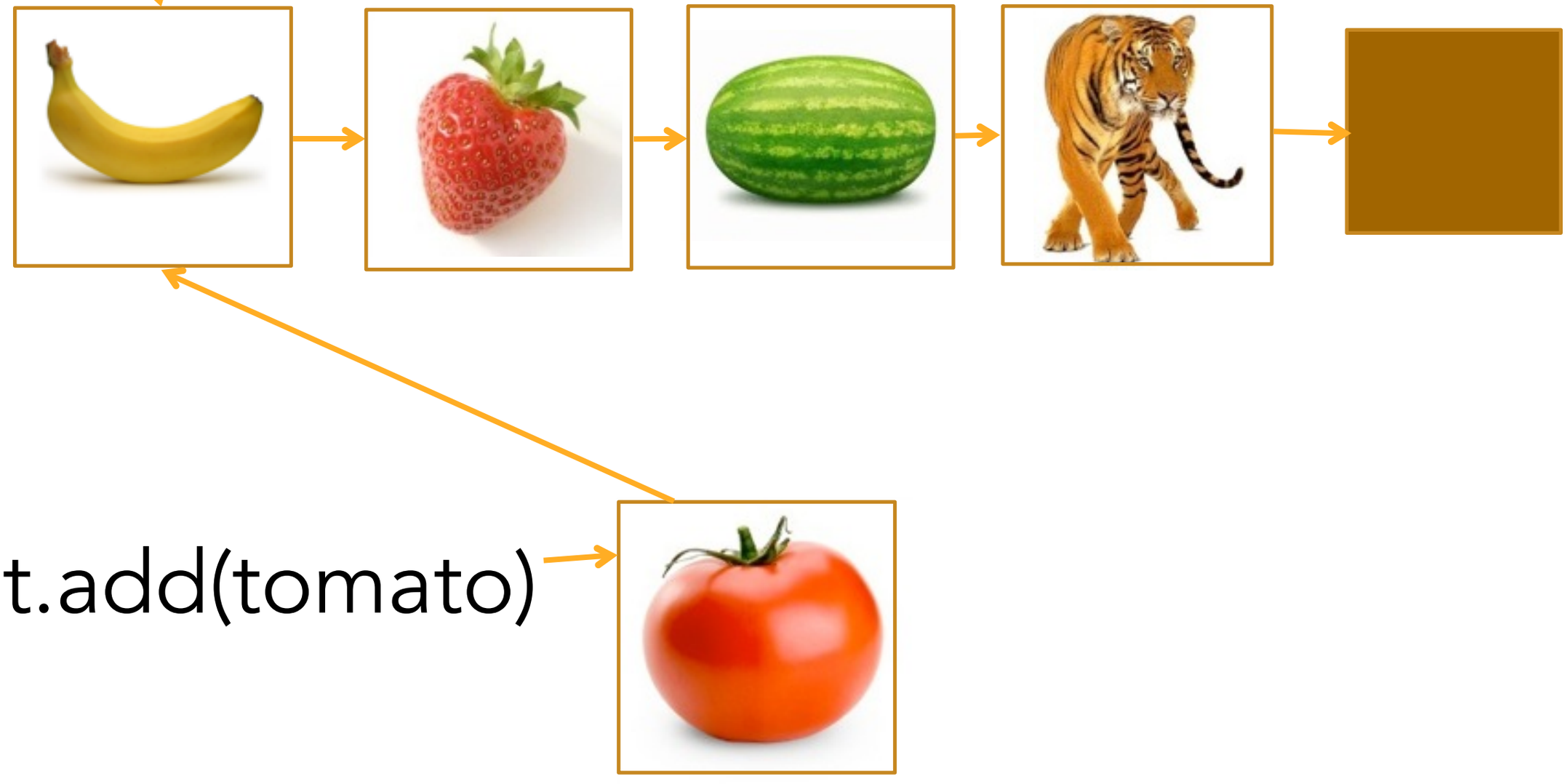
Java: copy on mod

```
public Customer addPhone(Phone newPhone) {
    Iterable<Phone> morePhones =
                ImmutableList.builder()
                    .addAll(phones)
                    .add(newPhone).build();
    return new Customer(morePhones);
}
```

F#: copy on mod

```
member this.AddPhone (newPhone : Phone) {
    new Customer(newPhone :: phones)
}
```

fruit



fruit.add(tomato)

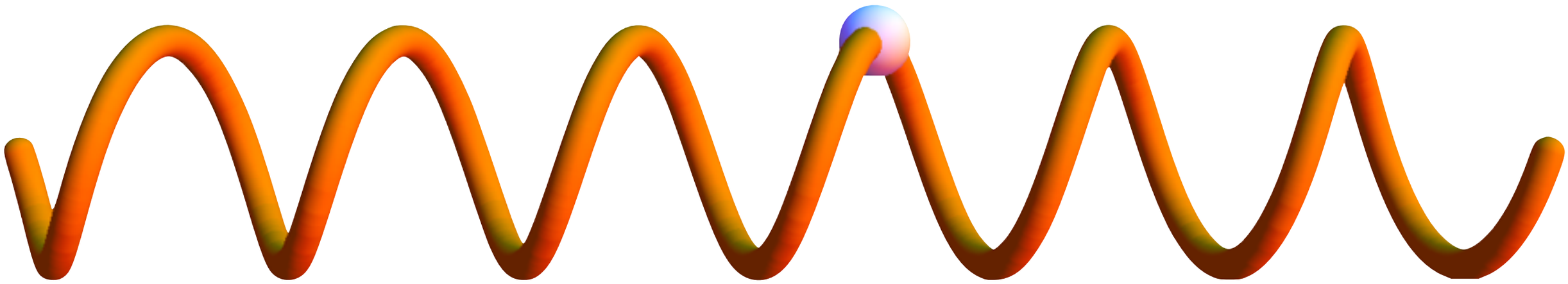
C#: shallow copy

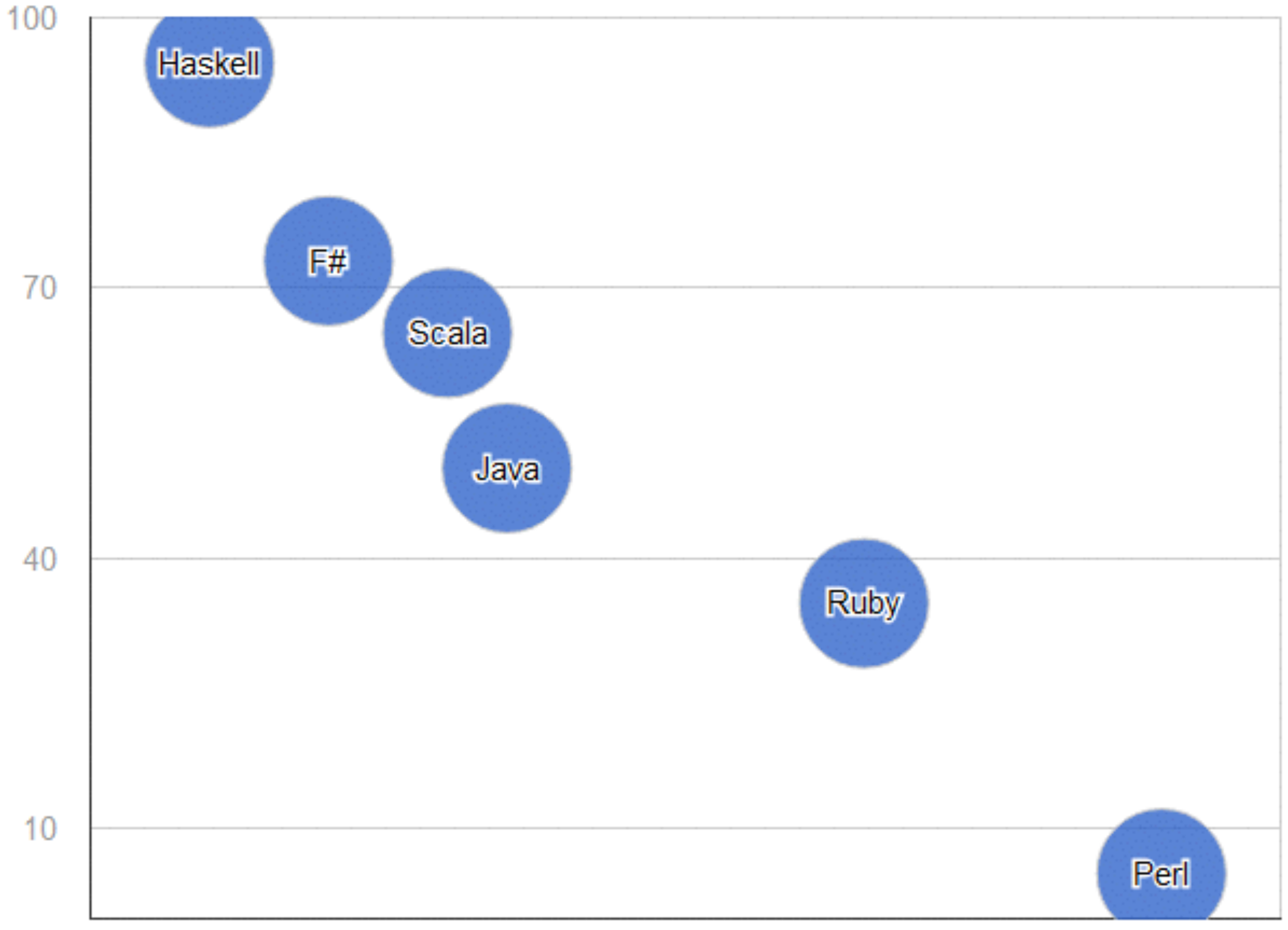
```
public Customer AddPhone(Phone newPhone) {
    IEnumerable<Phone> morePhones =
                // create list
    return new Customer(morePhones,
                        name,
                        address,
                        birthday ,
                        cousins);
}
```

persistent data structure

persistent data structure

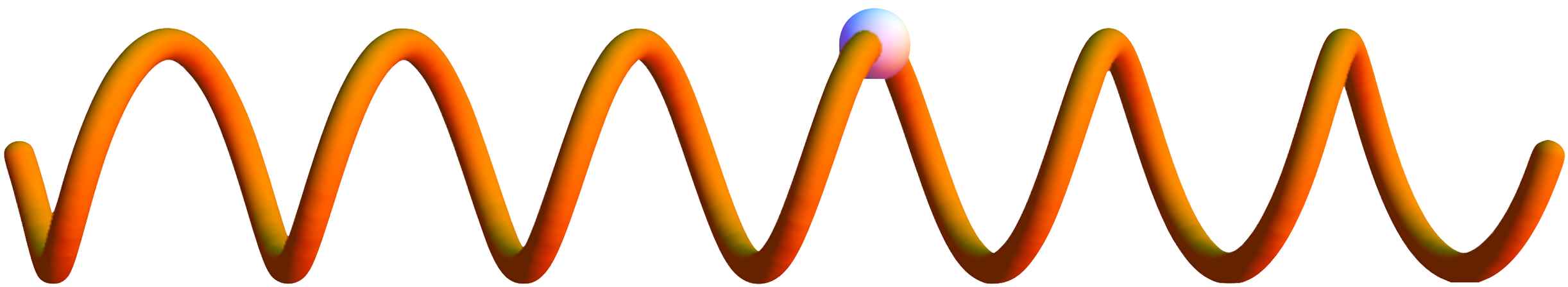
Specific Typing




100
70
40
10
Haskell
F#
Scala
Java
Ruby
Perl

expressiveness

catch errors early



Domain-Driven DESIGN

Tackling Complexity in the Heart of Software

Eric Evans

Foreword by Martin Fowler

The beginning of wisdom is to call things by their right names.

眉間一字寬，萬里眼中明

F#

```
[<Measure>] type cents
```

```
let price = 599<cents>
```

Scala

```
type FirstName = String          // type alias

Customer(name: FirstName, home: EmailAddress)
```

Scala

```
case class FirstName(value : String)
```

```
let friend = FirstName(“Joel”);
```

Java

```
public User(FirstName name, EmailAddress login)
```

```
public class FirstName {
   public final String stringValue;

   public FirstName(final String value) {
      this.stringValue = value;
   }

   public String toString() {...}
   public boolean equals() {...}
   public int hashCode() {...}
}
```

```
time => Cost
time => Volume
(Volume, Cost) => Expenditure
```

∴

```
time => Expenditure
```

$$A \Rightarrow B$$

$$B \Rightarrow C$$

$$\therefore$$

$$A \Rightarrow C$$

IF THIS THEN THAT

```
time => Cost
time => Volume
(Volume, Cost) => Expenditure
```

∴

```
time => Expenditure
```

State everything you need

State only what you need

Scala

```
List [+A]

def indexOf [B >: A] (elem: B): Int

def sameElements(that: GenIterable[A]): Boolean
```

C#: weak

```
public boolean Valid(Customer cust)
{
    EmailAddress email = cust.EmailAddress;
    // exercise business logic
    return true;
}
```

C#

```
public Validity Valid(IHasEmailAddress any)
{
   EmailAddress email = anything.EmailAddress;
   // exercise business logic
   return Valid;
}
```

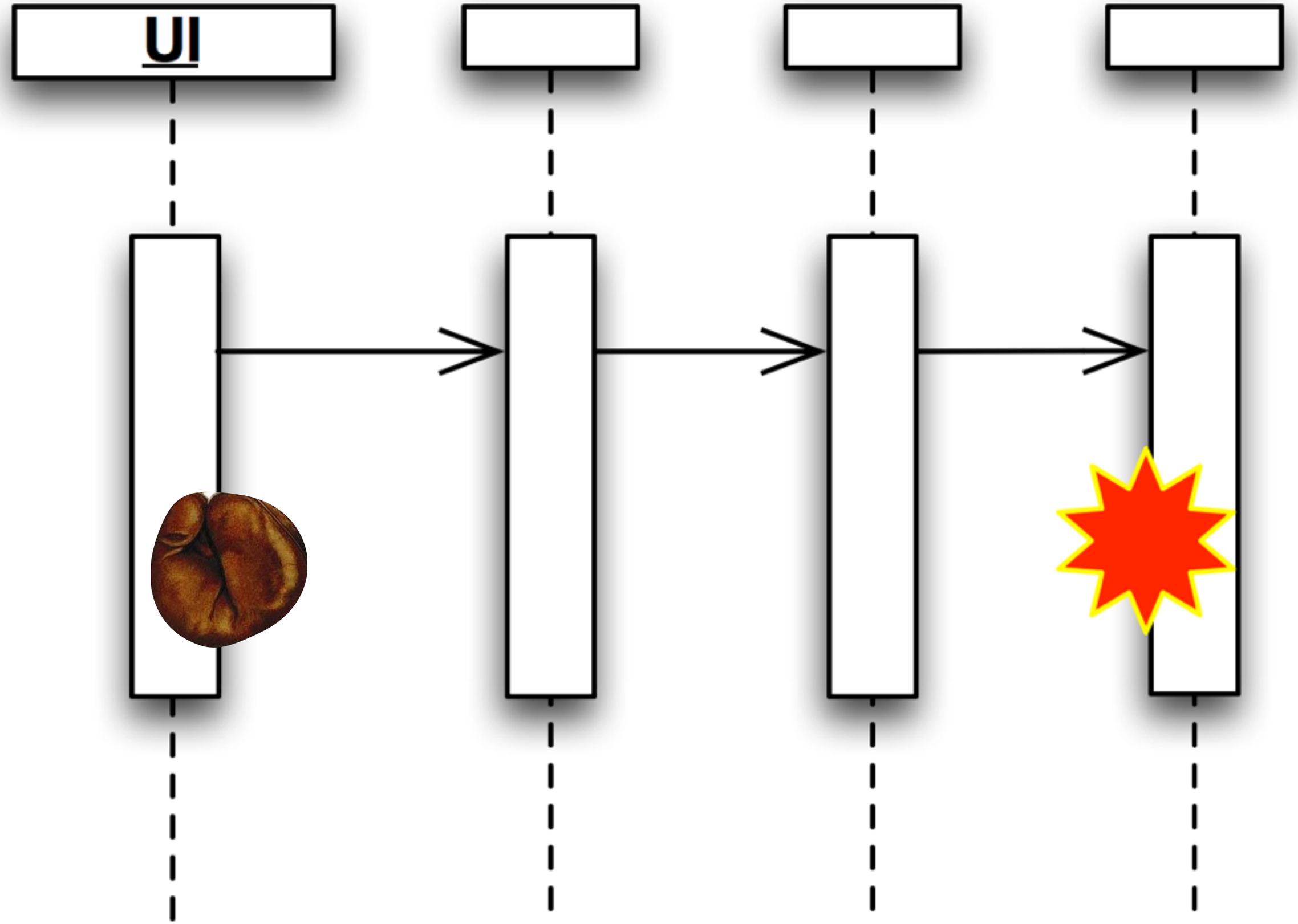
```
interface IHasEmailAddress {
   string EmailAddress {get; }
 }
```

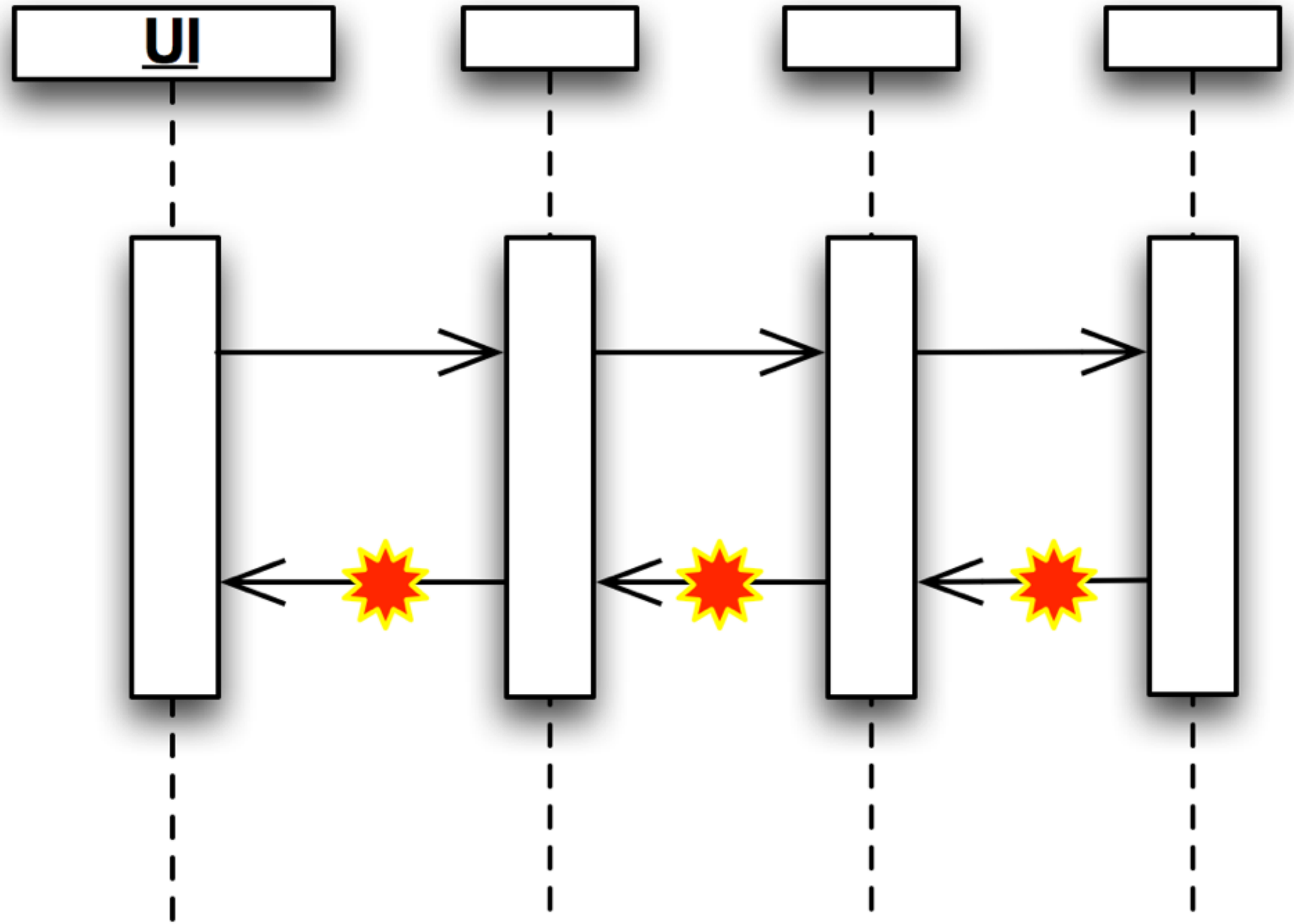
```
IHasEmailAddress => Validity
```

```
(UserDBService, AccountRequest) => Account
```

```
(UserDBService, AccountRequest) =>
    Either[AccountCreationFailure,Account]
```

Errors are data too

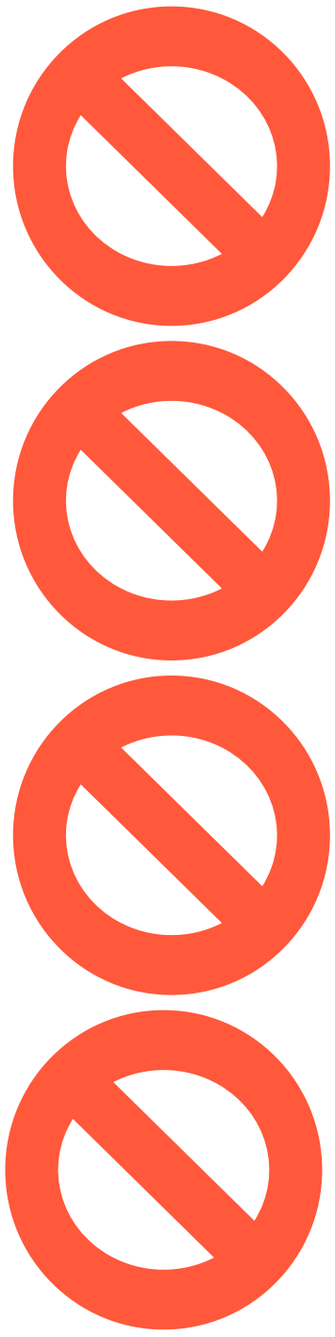

UI


UI

access global state

modify input

change the world



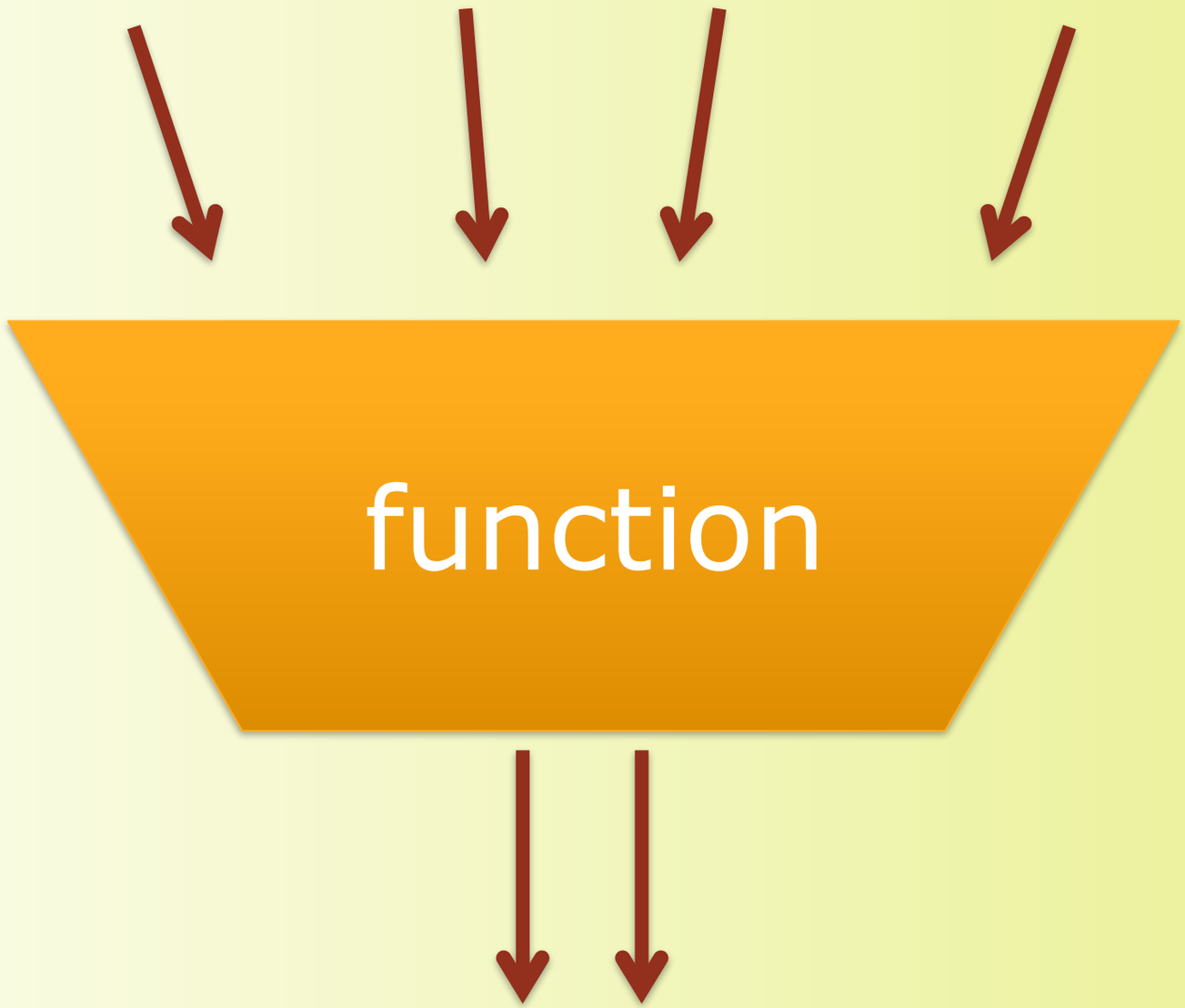
access global state

modify input

change the world

interrupt execution flow

this talk is brought to you by… the Tuple type!

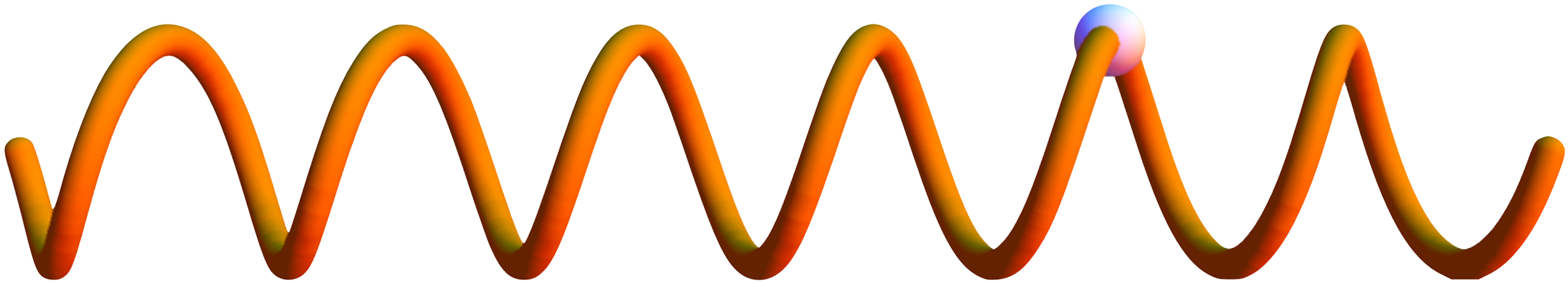



```
new Tuple<string,int>(“You win!”, 1000000)
```

Tuple<T1,T2,...>

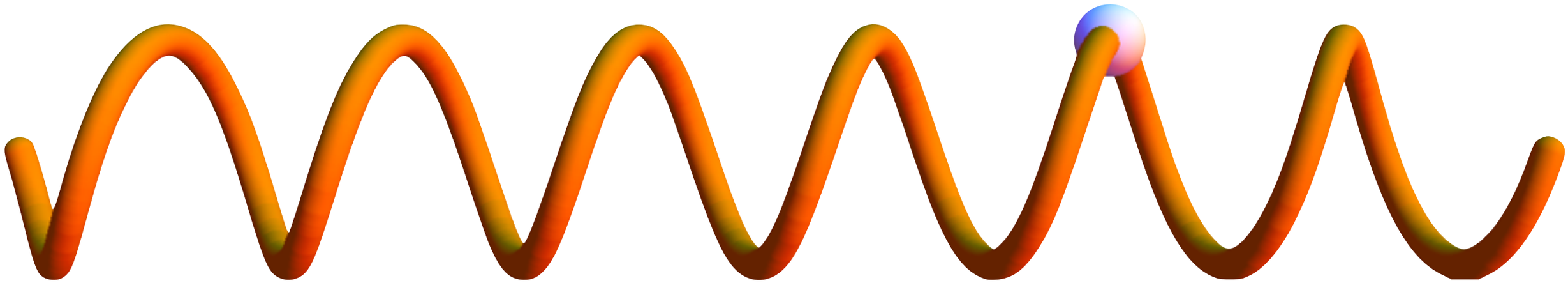
When one return value is not enough!

Declarative Style



say what you're doing,

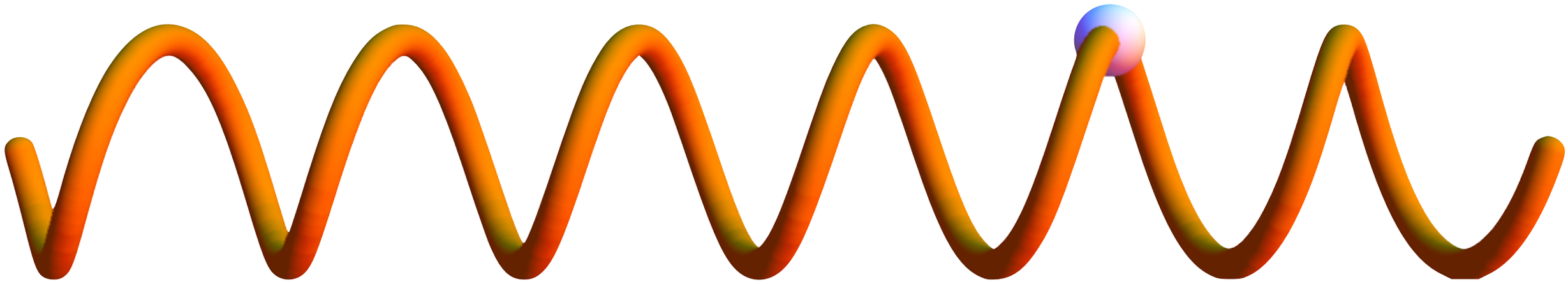
not how you're doing it



```
select ROLE_NAME,
UPDATE_DATE
from USER_ROLES
where USER_ID = :userId
```

readable code

smaller pieces



familiar != readable

Java

```
public List<String> findBugReports(List<String> lines)
{
    List<String> output = new LinkedList();
      for(String s : lines) {
         if(s.startsWith(“BUG”)) {
            output.add(s);
         }
      }
    return output;
}
```

Scala

```
lines.filter( _.startsWith(“BUG”))
```

```
lines.par.filter( _.startsWith(“BUG”))
```

Java 8

```
lines.stream
  .filter( _.startsWith(“BUG”))
  .collect(Collectors.toList)
```

Java 6

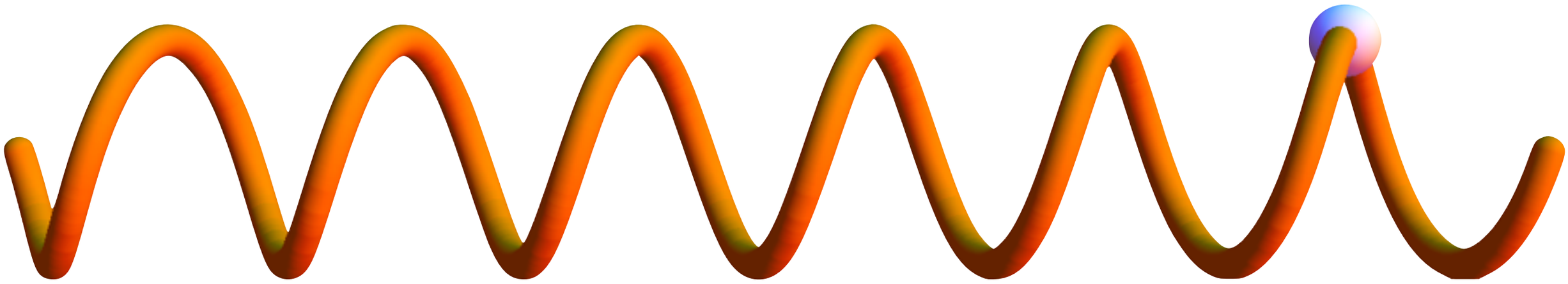
```
filter(list, startsWithBug);
```

```
final Predicate<String> startsWithBug =
    new Predicate<String>() {
      public boolean apply(String s) {
        return s.startsWith("BUG");
      }
    };
```

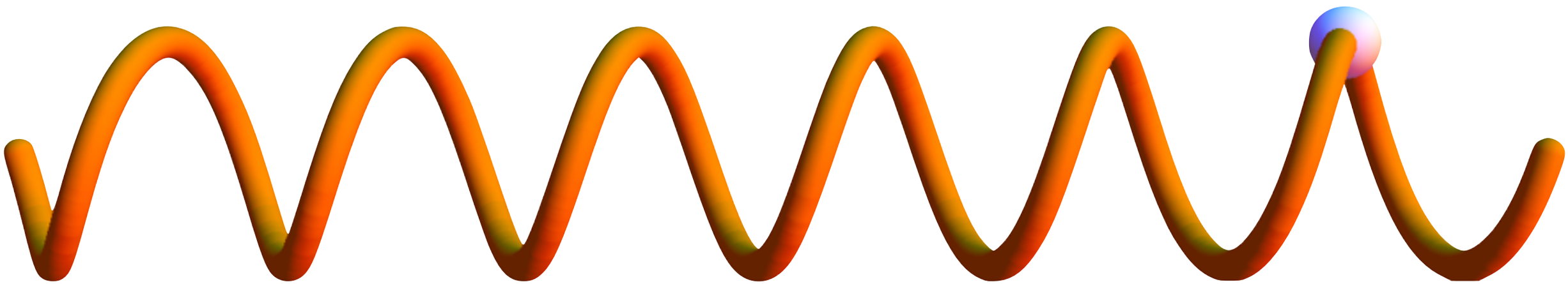
C#

```
lines.Where(s => s.StartsWith(“BUG”)).ToList
```

Lazy Evaluation



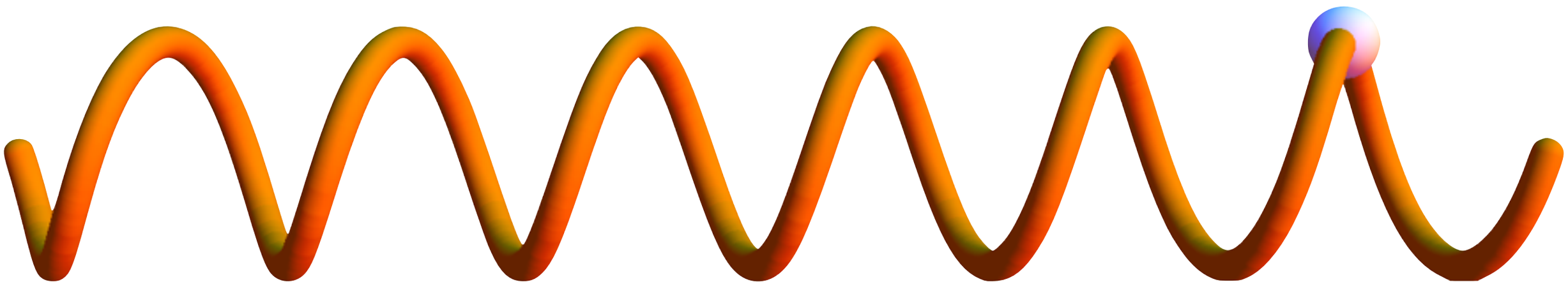
delay evaluation until the last responsible moment



save work

separate “what to do”

from “when to stop”



Java

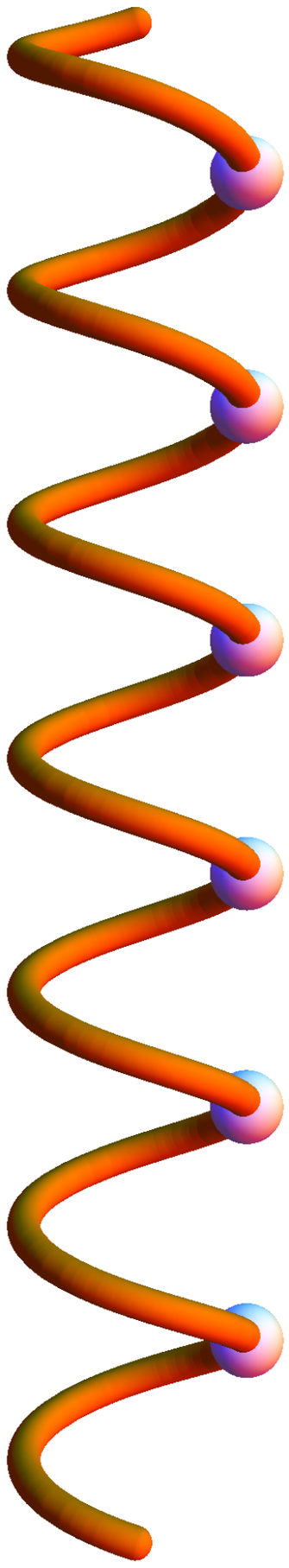
```
int bugCount = 0;
String nextLine = file.readLine();
while (bugCount < 40) {
    if (nextLine.startsWith("BUG")) {
        String[] words = nextLine.split(" ");
        report("Saw "+words[0]+" on "+words[1]);
        bugCount++;
    }
    waitUntilFileHasMoreData(file);
    nextLine = file.readLine();
}
```

Java 6

```
for (String s :
 FluentIterable.of(new RandomFileIterable(br))
    .filter(STARTS_WITH_BUG_PREDICATE)
    .transform(TRANSFORM_BUG_FUNCTION)
    .limit(40)
    .asImmutableList()) {
       report(s);
}
```

C#

```
IEnumerable<string> ReadLines(StreamReader r)
{
    while (true) {
        WaitUntilFileHasMoreData(r);
        yield return r.ReadLine();
    }
}
```



Data In, Data Out

Verbs Are People Too

Immutability

Specific Typing

Declarative Style

Lazy Evaluation

Alistair Cockburn's
Oath of Non-Allegiance

I promise not to exclude
from consideration any idea
based on its source, but to consider ideas
across schools and heritages
in order to find the ones that best suit the
current situation.

Optional<T>

Tuple<T>

Titanium Sponsors

AJi

VALOREM CONSULTING
WWW.VALOREMCONSULTING.COM
REGISTER TO WIN AN XBOX!

PAIGE TECHNOLOGIES
INTELLIGENT PAIRING. PERPETUAL SUCCESS.

AdventureTech
www.adventuretechgroup.com

Platinum Sponsors

ADAPTIVE SOLUTIONS GROUP

DST SYSTEMS

Balance Innovations

Sprint

epiQ SYSTEMS

DevExpress

jack henry & ASSOCIATES INC.

O'REILLY

Kauffman LABS for Enterprise Creation

FREIGHTQUOTE

Gold Sponsors

Cerner

CENTRIQ TRAINING

Bradford & Galt CONSULTING SERVICES

ADVANTAGE TECH

KEYHOLE SOFTWARE

ComponentOne a division of GrapeCity

dsi

LRS Consulting Services

GARMIN

JetBRAINS

Microsoft

MULTI SERVICE INNOVATION WHERE IT MATTERS

NETCHEMIA Transforming the way education works

OAKWOOD

stackify

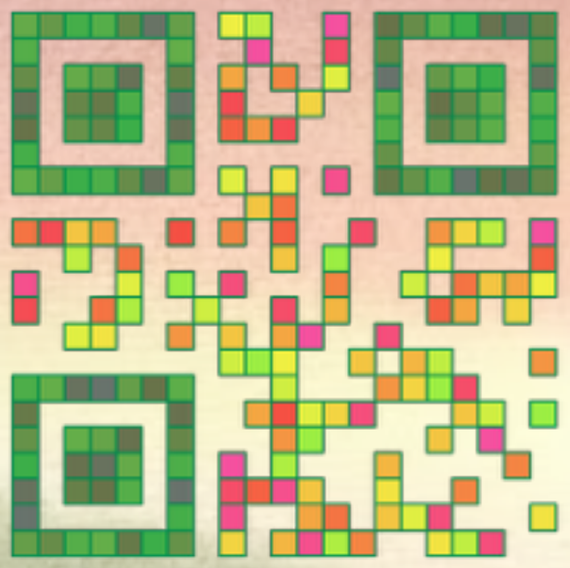
KU EDWARDS CAMPUS The University of Kansas

perceptivesoftware from Lexmark

TEKsystems

twilio CLOUD COMMUNICATIONS

UNITEDLEX



Jessica Kerr

blog.jessitron.com

@jessitron

github.com/jessitron/fp4ood

Man, the living creature, the creating individual, is always more important than any established style or system. – Bruce Lee