Object Oriented Design in the Wild

@jessitron

c2.com/cgi/wiki?PrinciplesOfObjectOrientedDesign

Principles Of Object Oriented Design

A suite of eleven principles, conceived by people such as RobertCecilMartin, BertrandMeyer, BarbaraLiskov.

There are six papers that describe all of these principles. They can be found in the resources section of http://w
principles, the final papers cover the remaining two principles: "Granularity" and "Stability".

There are five principles of class design (aka SOLID):

- (SRP) The SingleResponsibilityPrinciple
- (OCP) The OpenClosedPrinciple
- (LSP) The LiskovSubstitutionPrinciple
- (ISP) The InterfaceSegregationPrinciple
- (DIP) The DependencyInversionPrinciple

There are three principles of package cohesion

- (REP) The ReuseReleaseEquivalencePrinciple
- (CCP) The CommonClosurePrinciple
- (CRP) The CommonReusePrinciple

There are three principles of package coupling

- (ADP) The AcyclicDependenciesPrinciple
- (SDP) The StableDependenciesPrinciple
- (SAP) The StableAbstractionsPrinciple

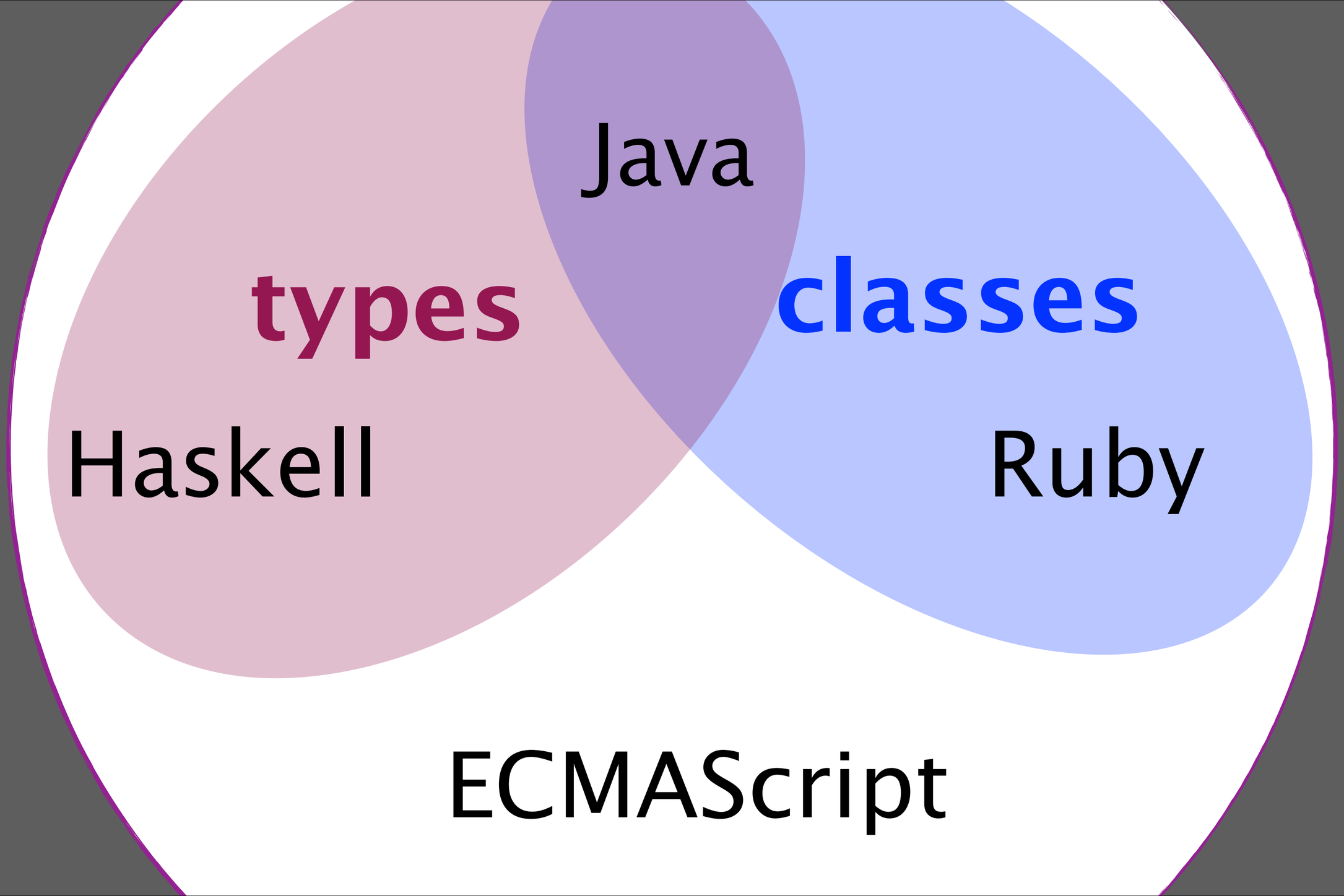
SOLID

+

package-level principles

+

DRY & YAGNI

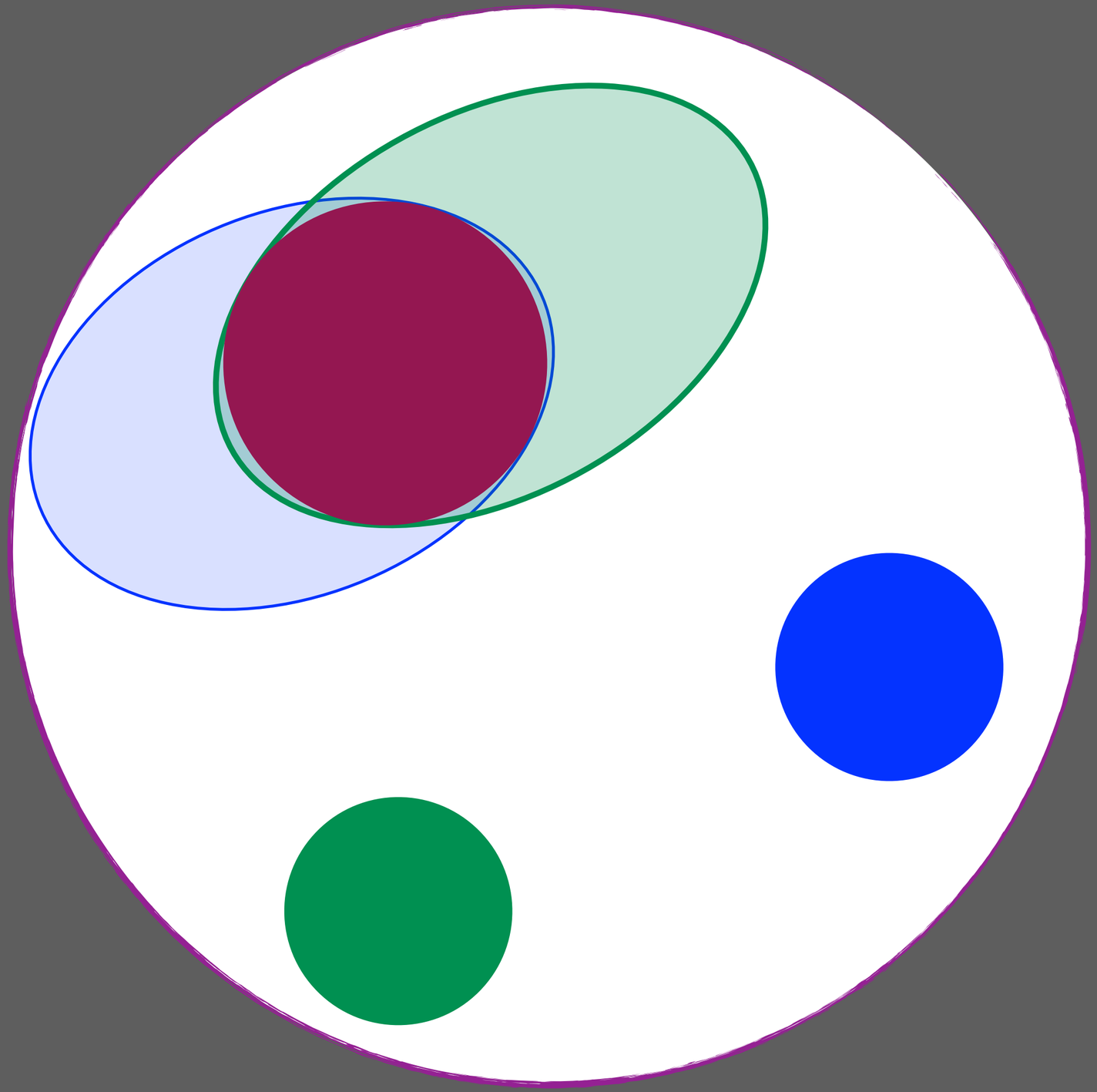

Java
types
classes
Haskell
Ruby
ECMAScript

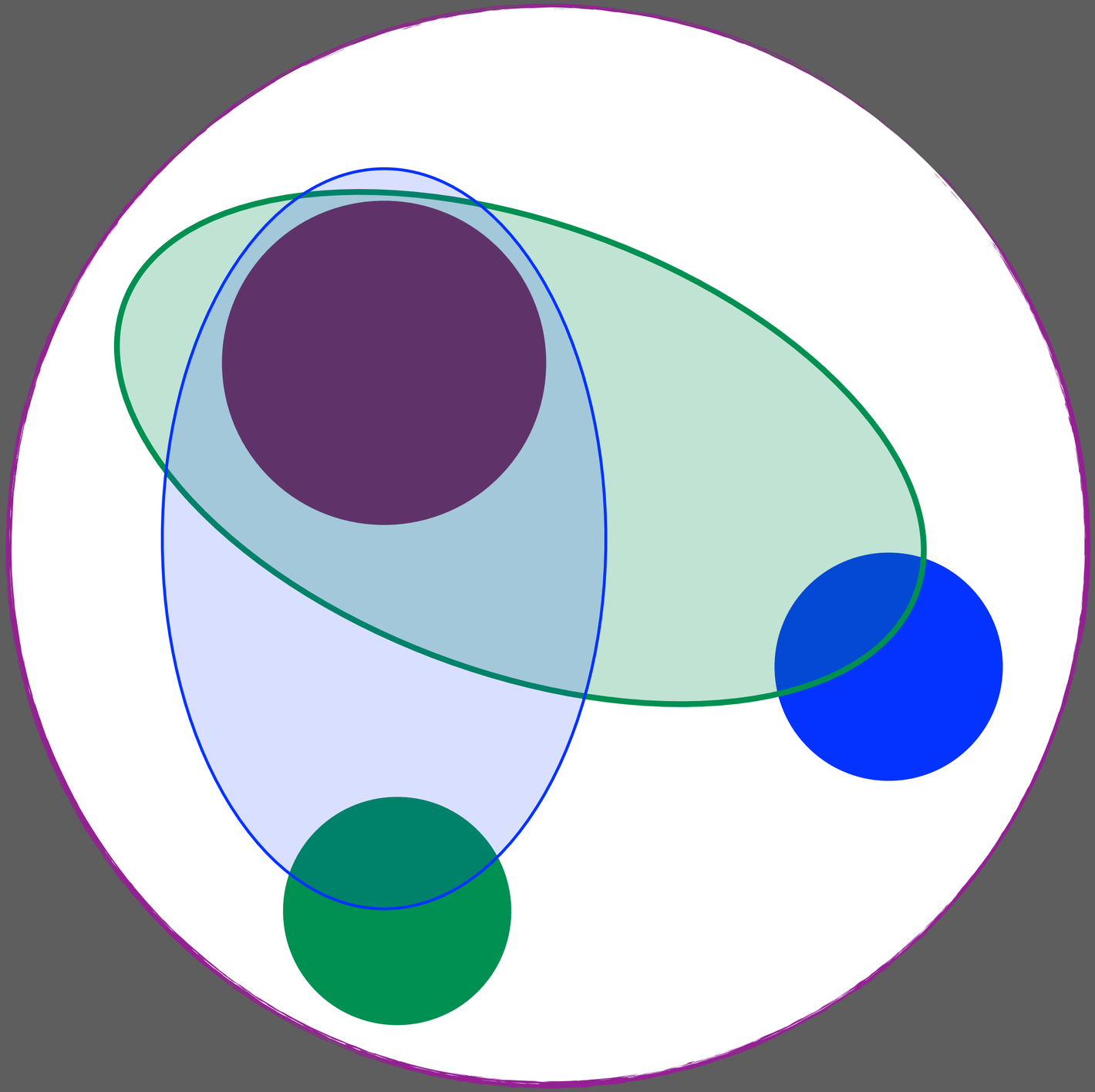
?

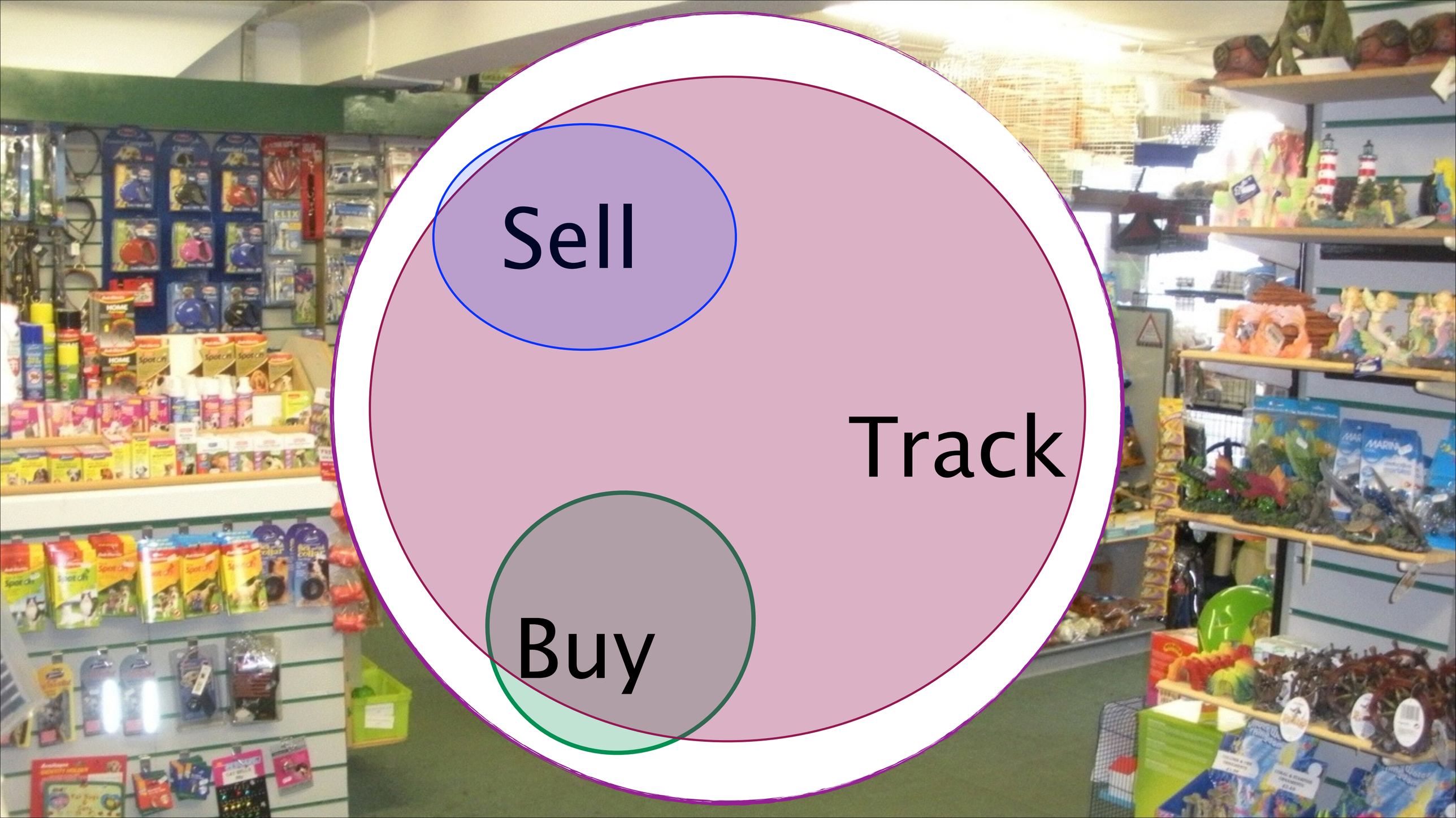
SOLID

Single Responsibility Principle

Don’t
Repeat
Yourself





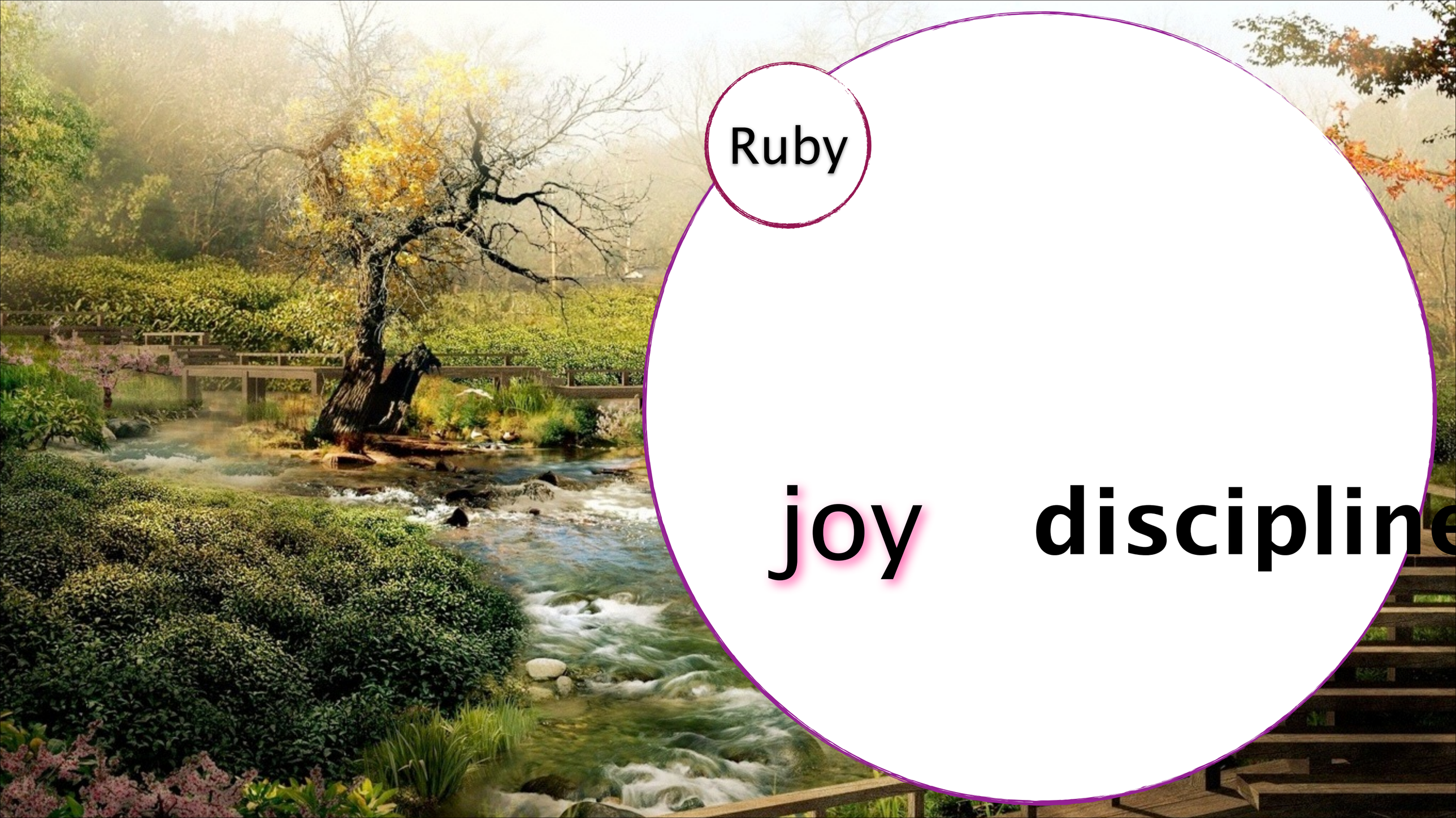

Sell
Track
Buy

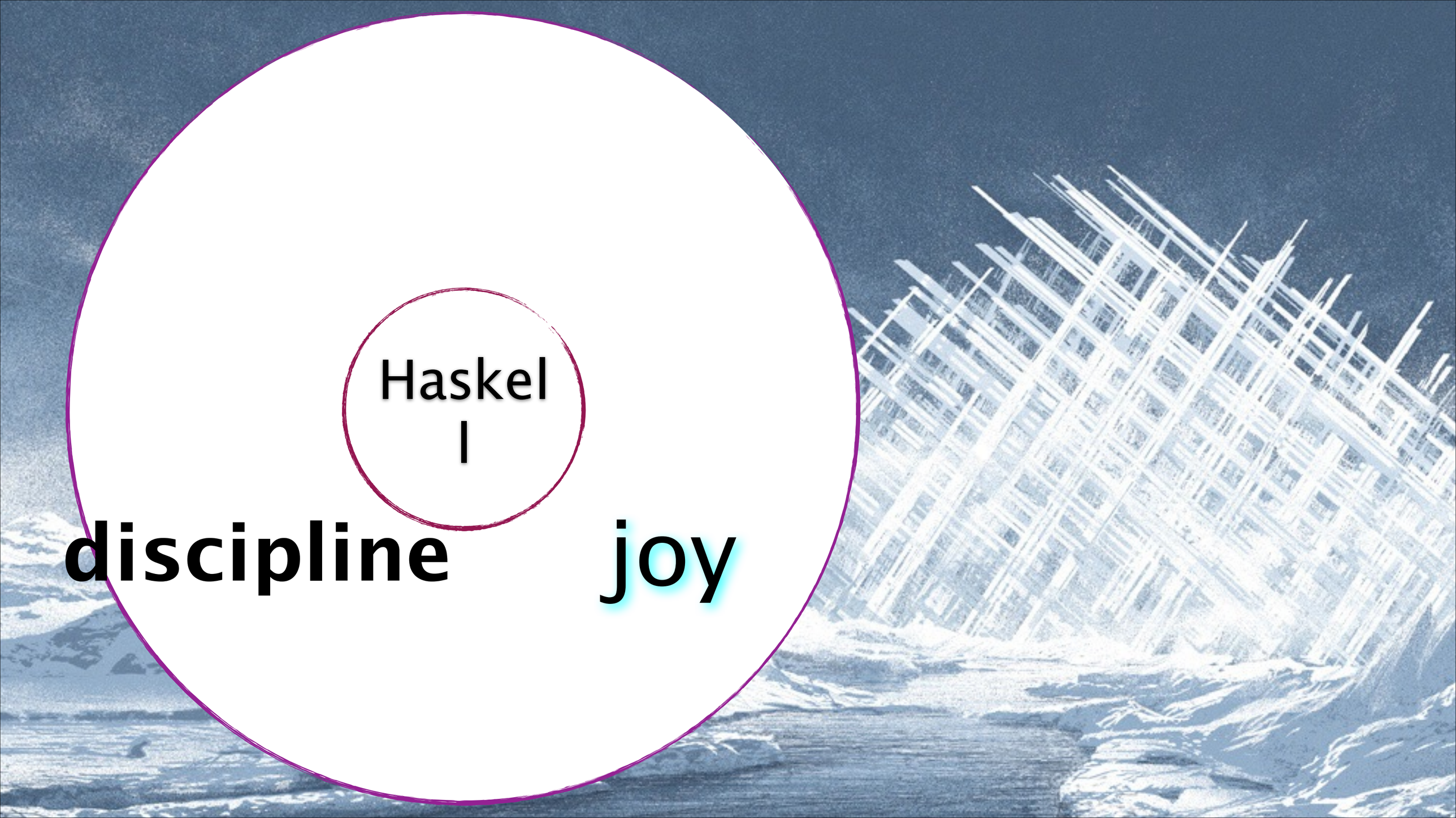
Java

```
class Good implements InvItem,
                      Purchase,
                      Sellable
{
  InventoryImpl inventory;
  PurchaseImpl  purchase;
  SellableImpl  sellable;
  ...
  }
```

Ruby

```
class Good
  include SellableItem
  include InventoryTracking
  include Purchase
end
```


Ruby
joy
discipline


Haskel
I
discipline
joy

Haskel
I

```
data Good = Good SaleInfo InvInfo PurchInfo
```

```
instance ReceiptItem Good where
  format Good s _ _ = desc s + price s
```

functions

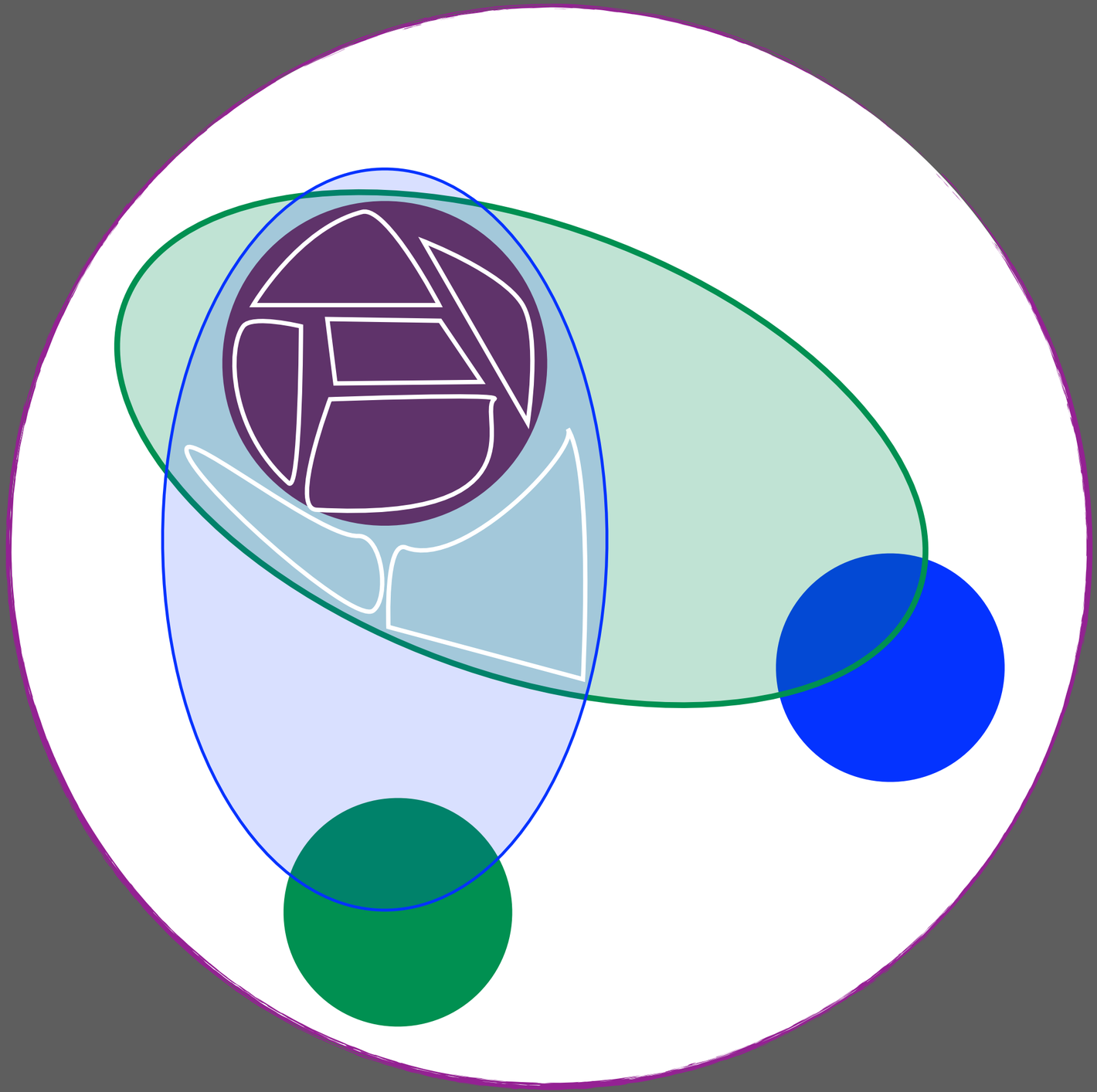
ECMA
Script

dictionaries

ECMA Script

```
var good = _.extend({}, invInfo,
                       saleInfo, purInfo)

{ name: “Canteen”,
  sku: 37671287,
  price: 1399,
  format: function() {...}
  cost: 715
}
```



SOLID

Interface
Segregation
Principle

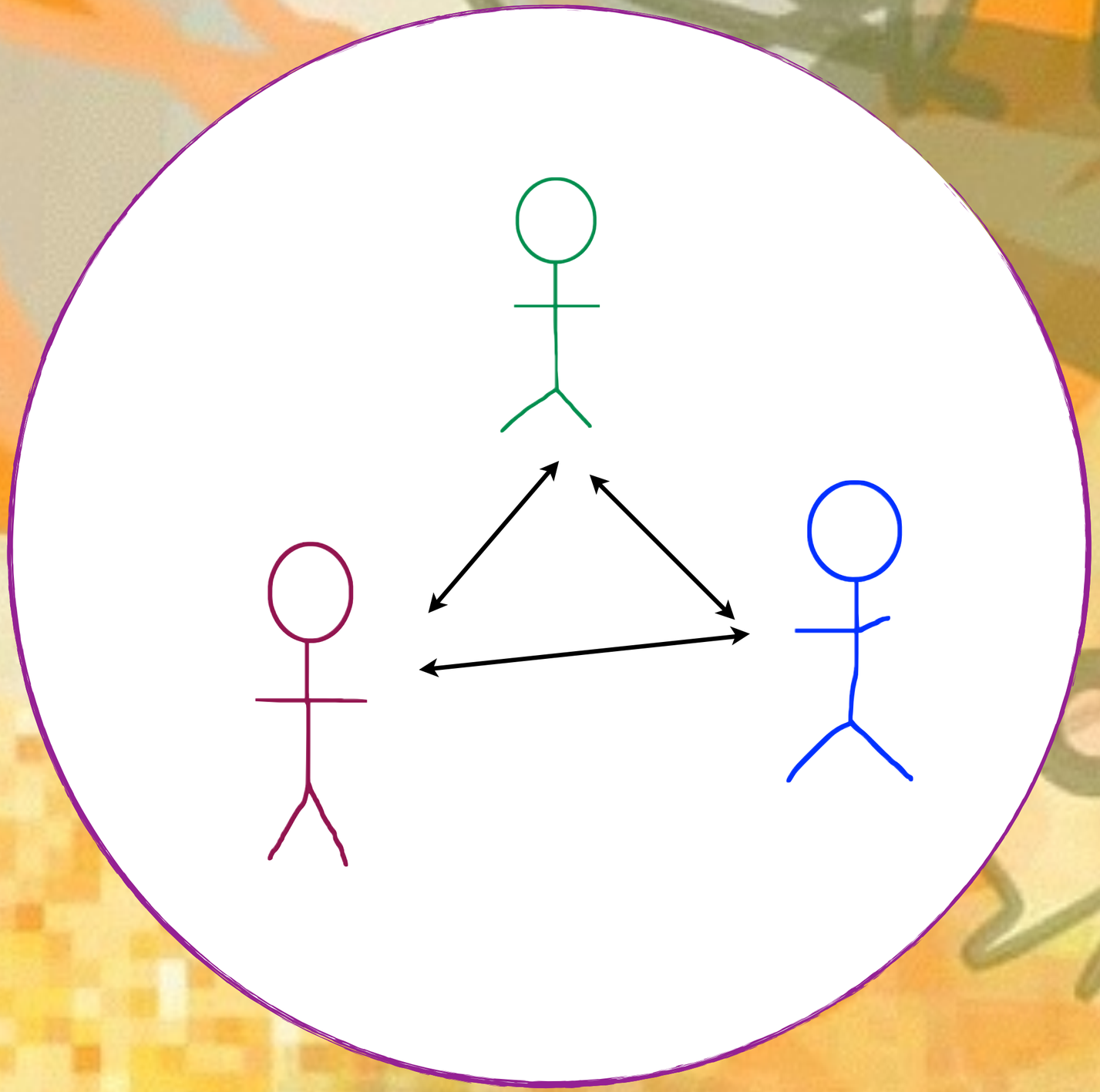
Java

```
String receiptLine(Sellable item)
```

ECMA Script

```
function receiptLine(item) {
 console.log("Cha-ching! profit "
        + item.price - item.cost)

 return format(item.desc,
              item.price)
 }
```



Haskel
l

```
print ::
(ReceiptItem a) => a -> String
```

SOLID

Liskov
Substitution
Principle

Java

```
List sneaky = new MyList()
...

if (sneaky.length == -1) {
   System.out.println("WAT!")
}
```

LSP

PLS

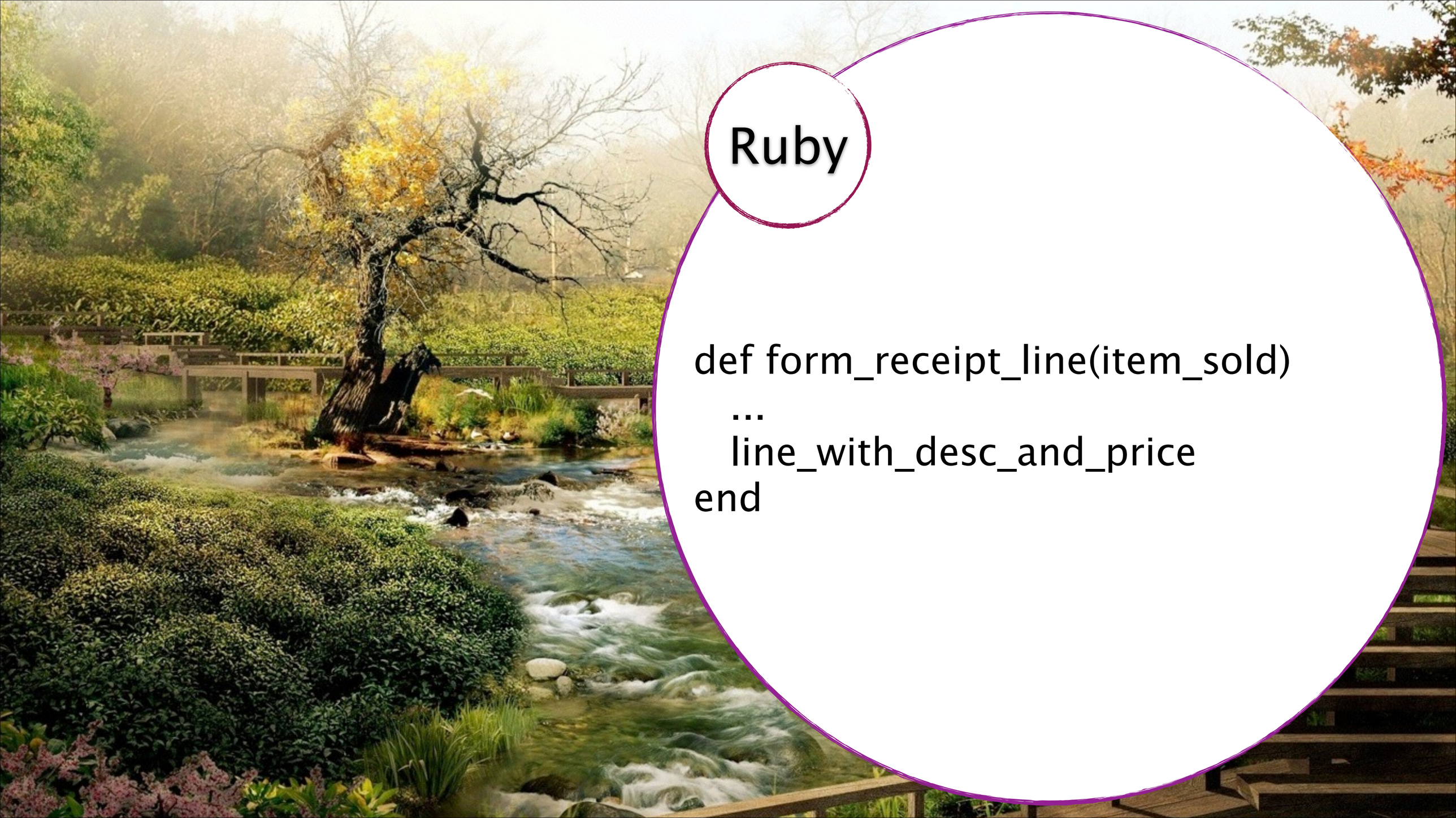
Principle of

Least

Surprise

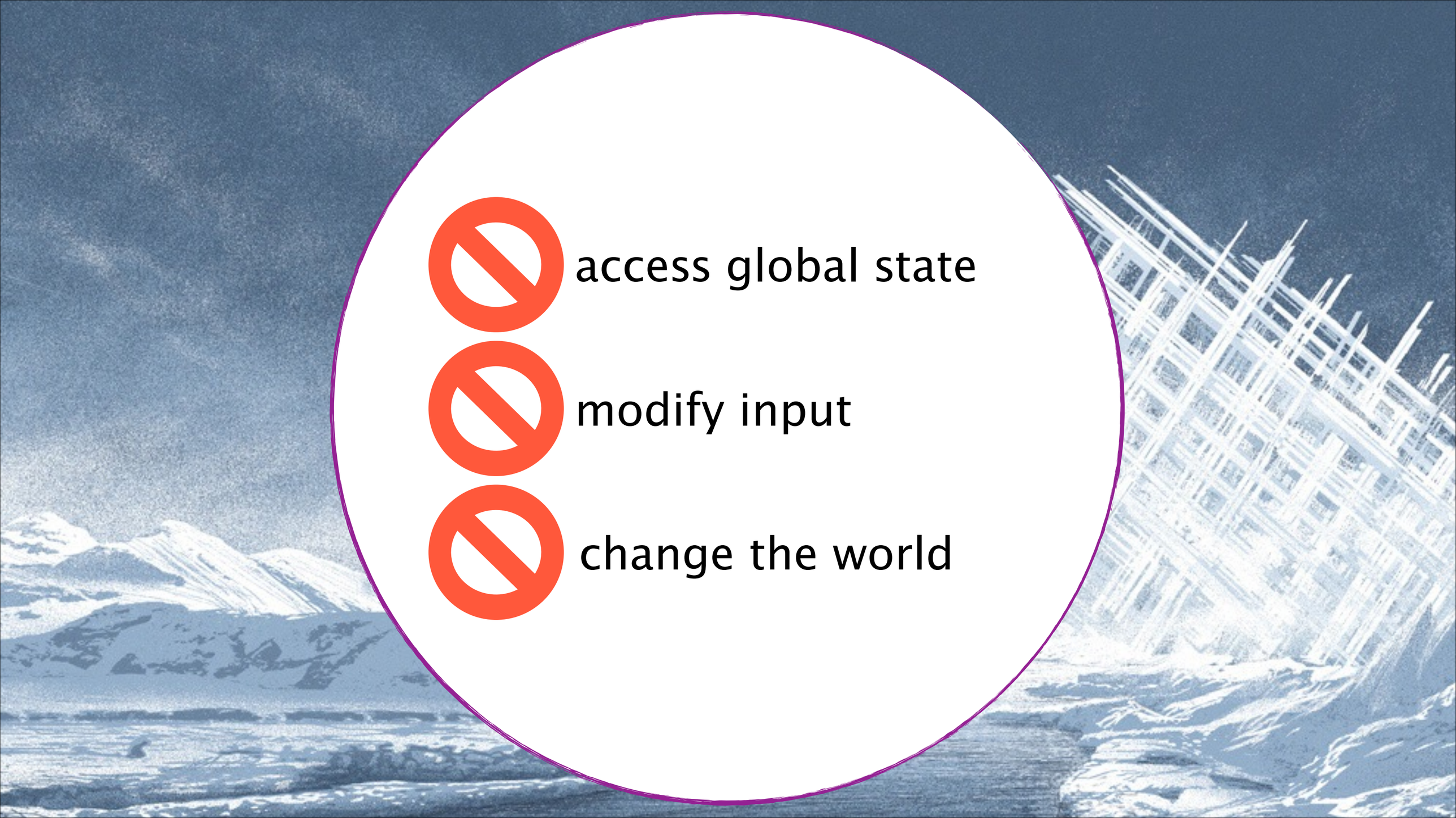
Ruby

```
class ImmutableList
  def map(f)
    ...
    ImmutableList.new(...)
  end
end
```

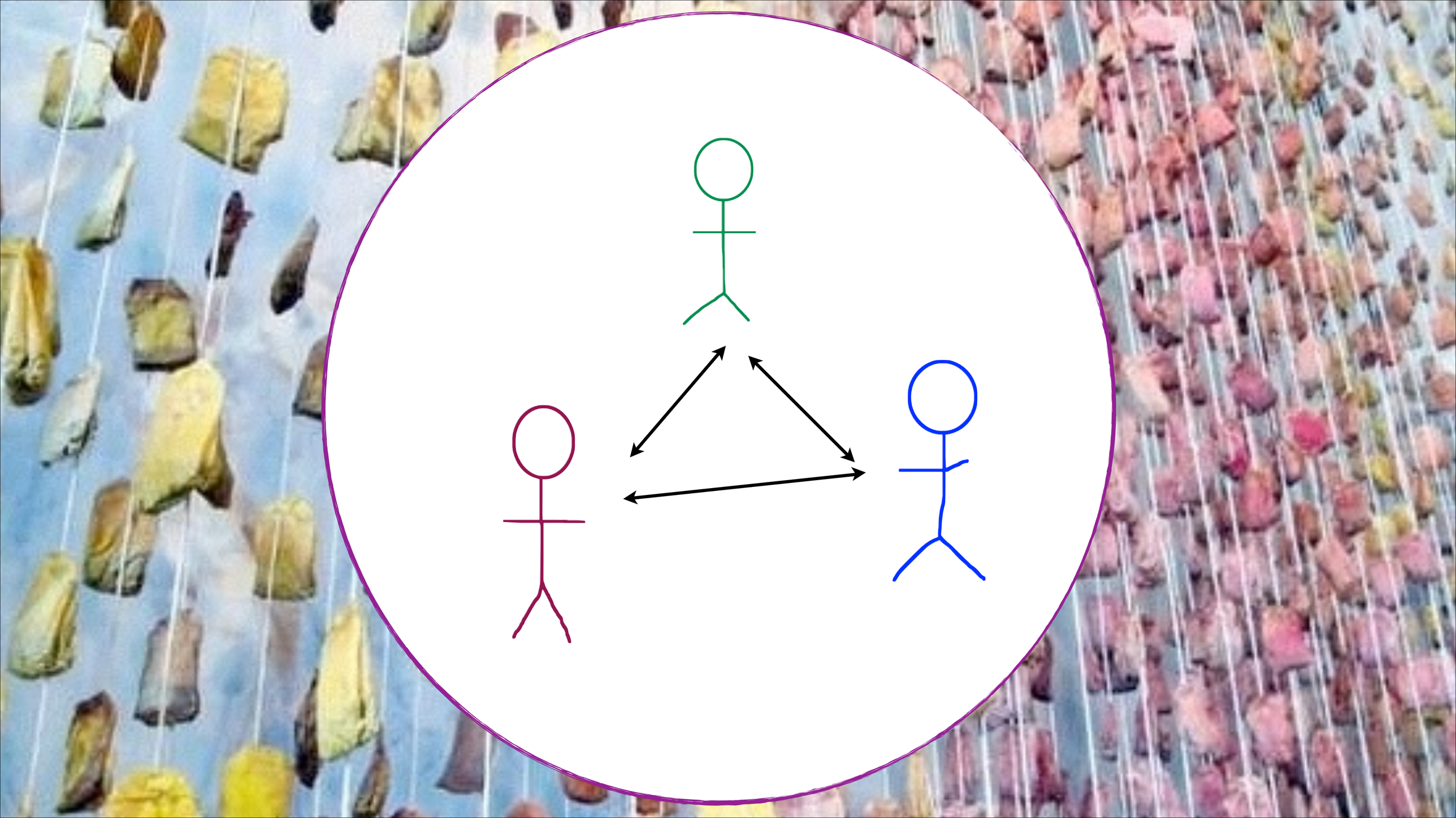


Ruby

```
def form_receipt_line(item_sold)
  ...
  line_with_desc_and_price
end
```

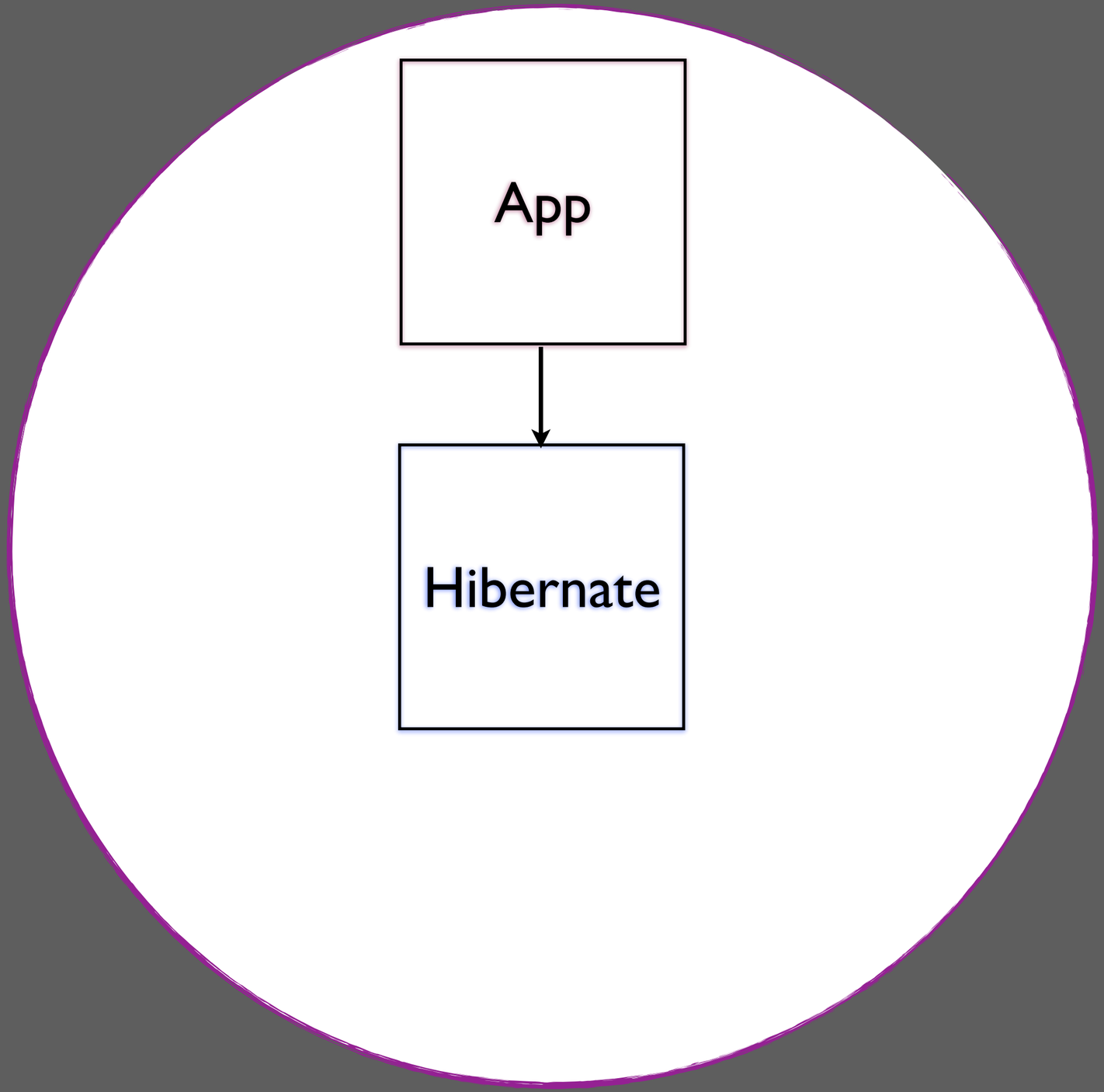

access global state
modify input
change the world

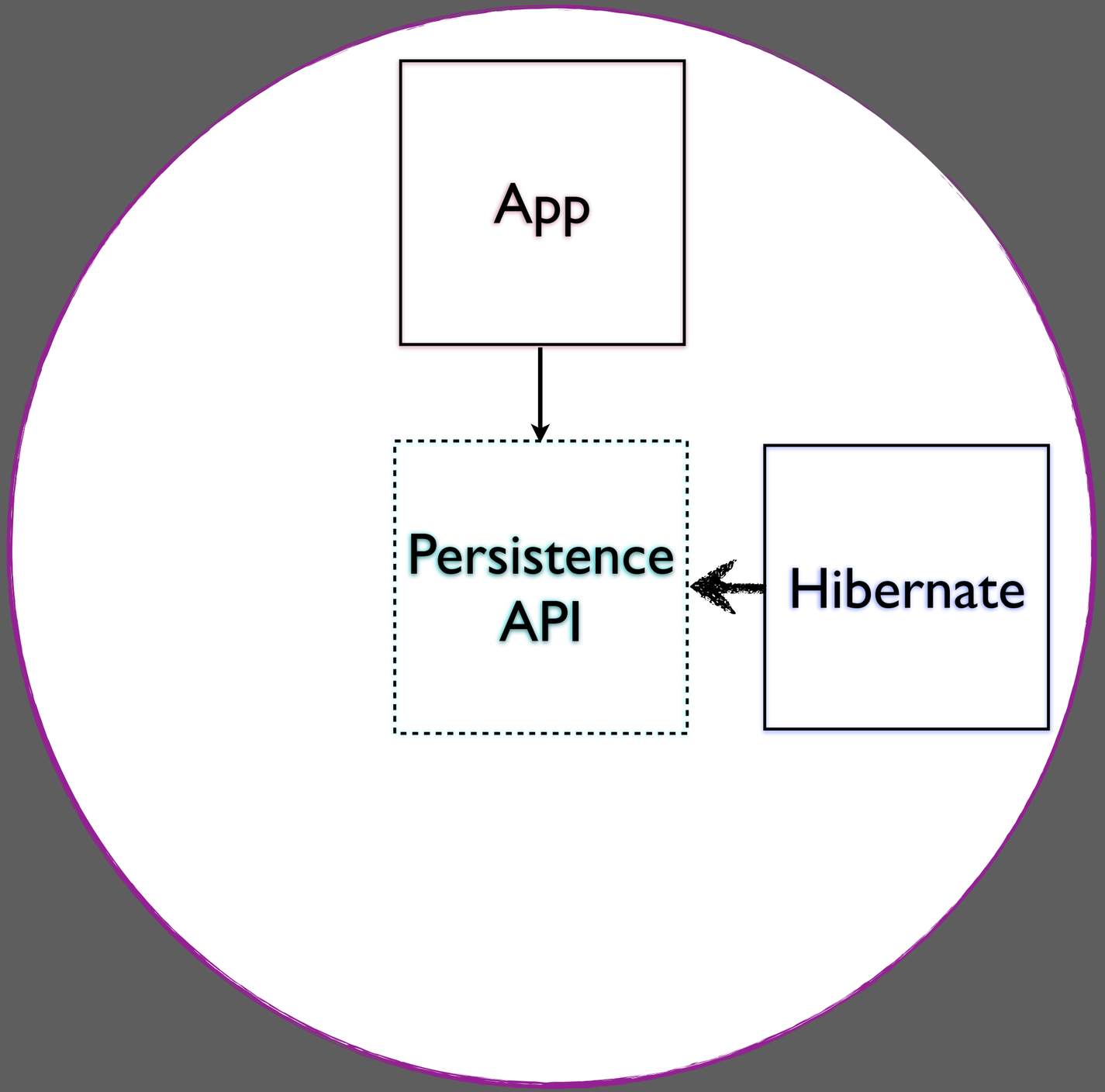
violate expectations

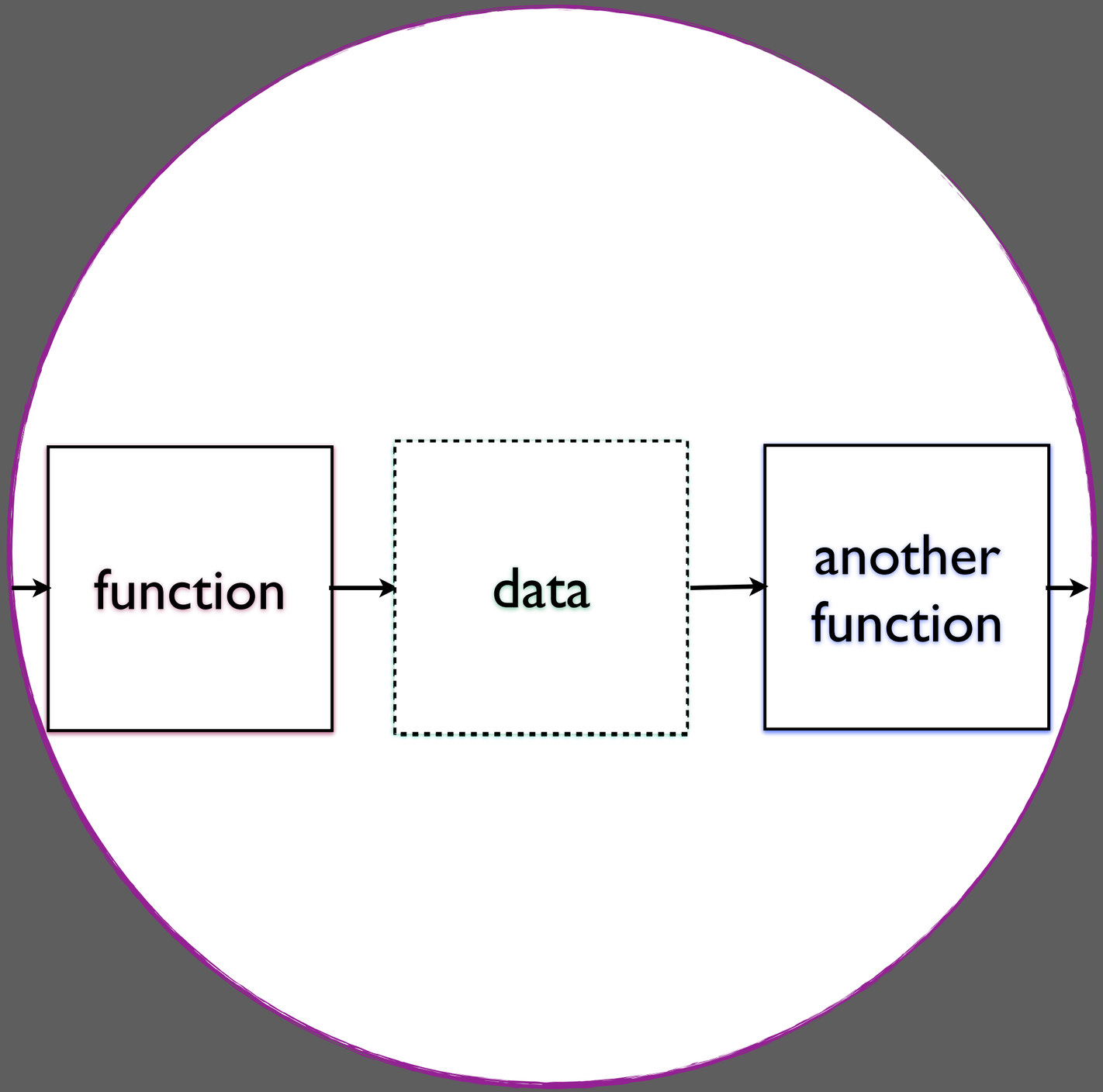


SOLI**D**

Dependency
Inversion
Principle


App
Hibernate


App
Persistence API
Hibernate


function
data
another function

ECMA Script

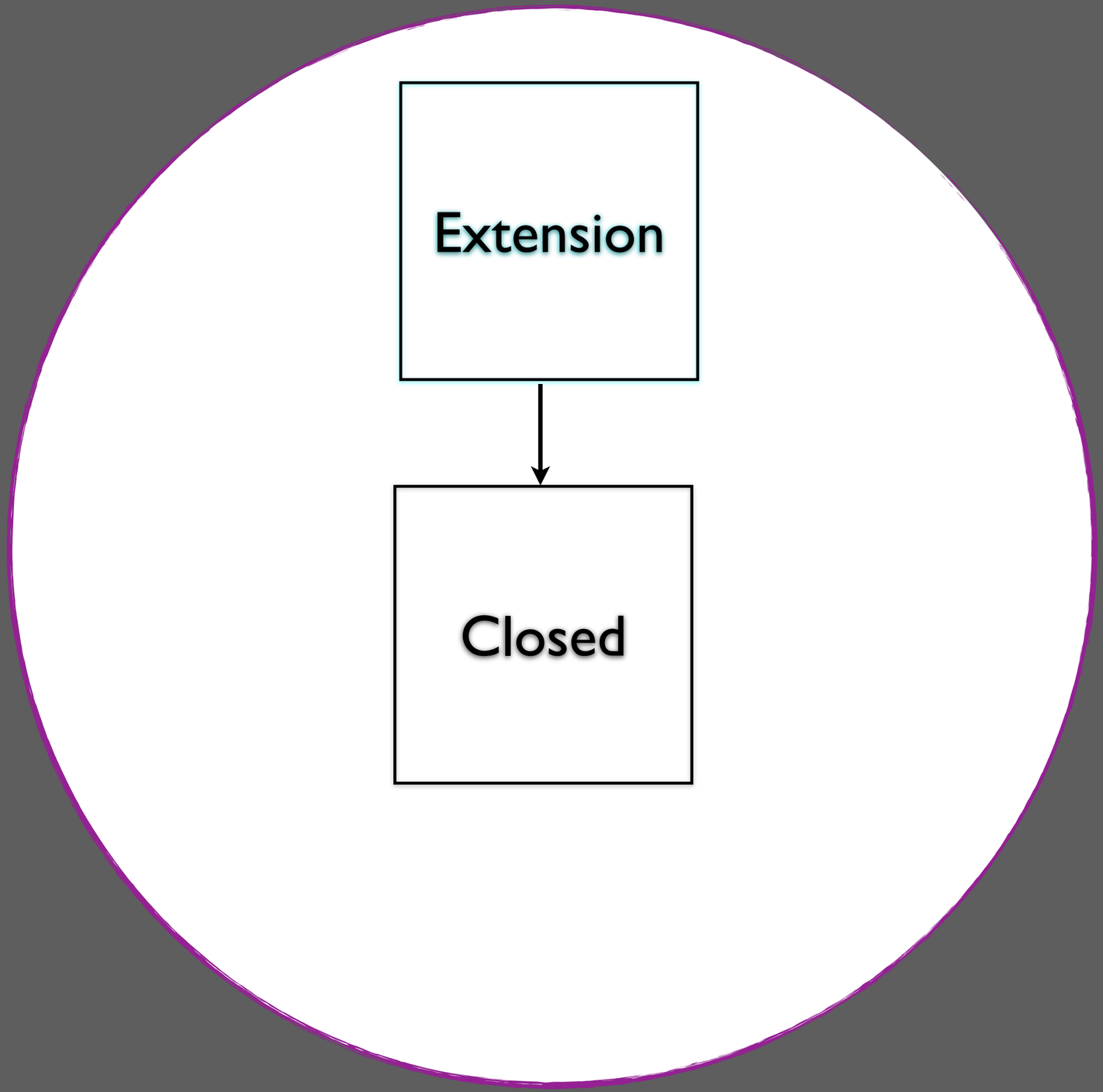
```
var items = _.map(codes, findItem)
var lines = _.map(items,
                  formatReceiptLine)
```

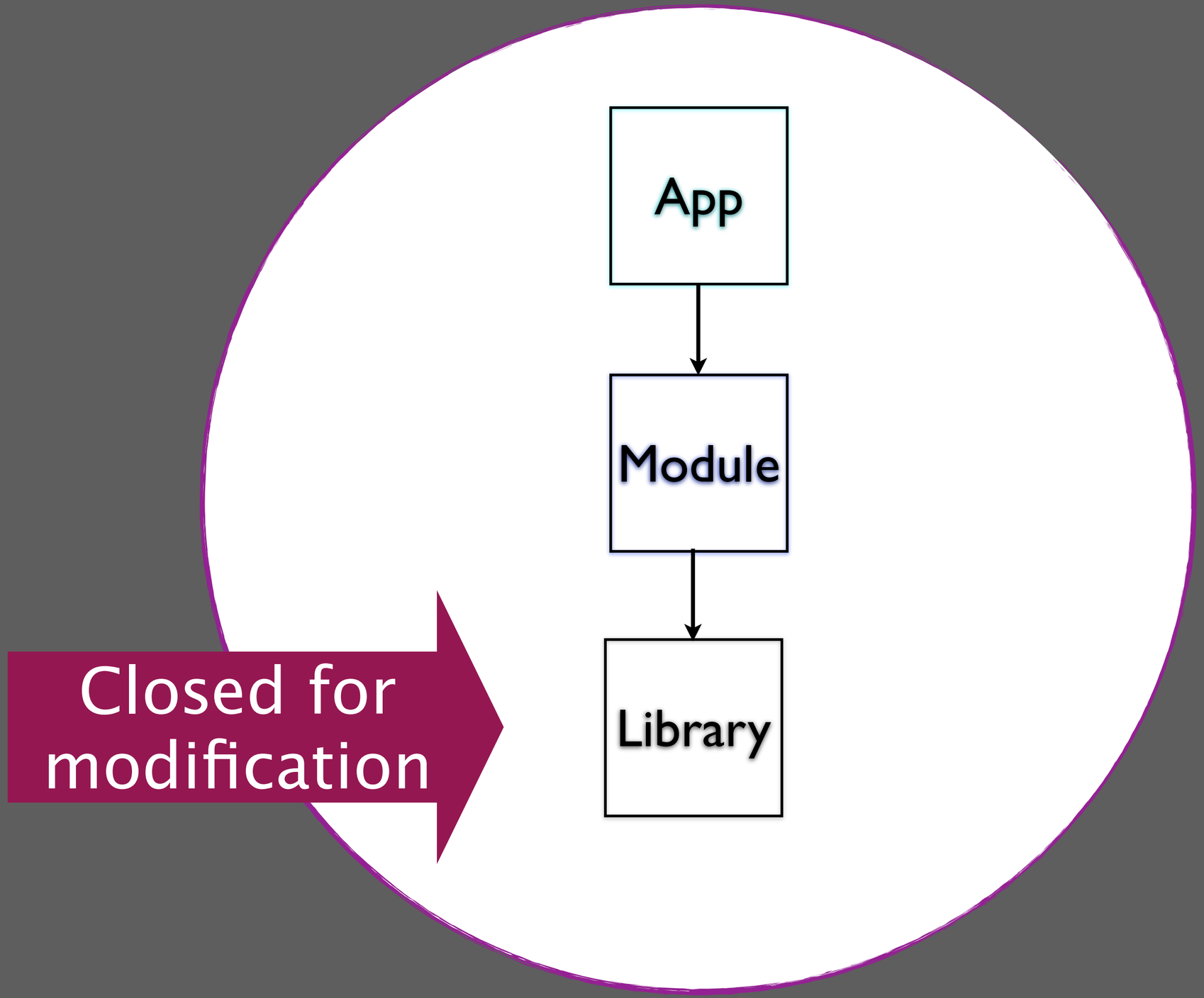
ksh

```
> cat data.csv | grep 42 | cut -f2 | sort
```

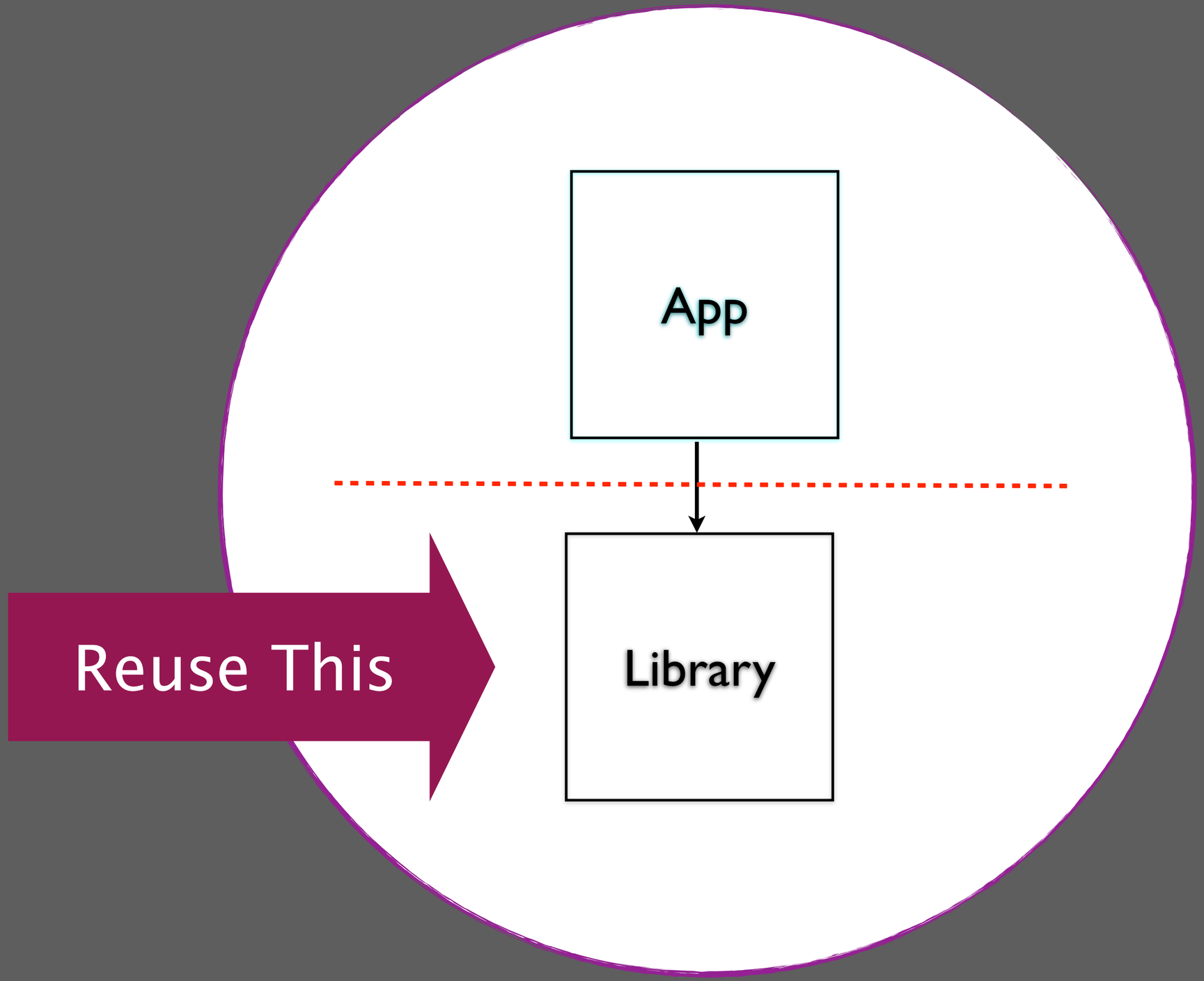
SOLID

Open/Closed Principle

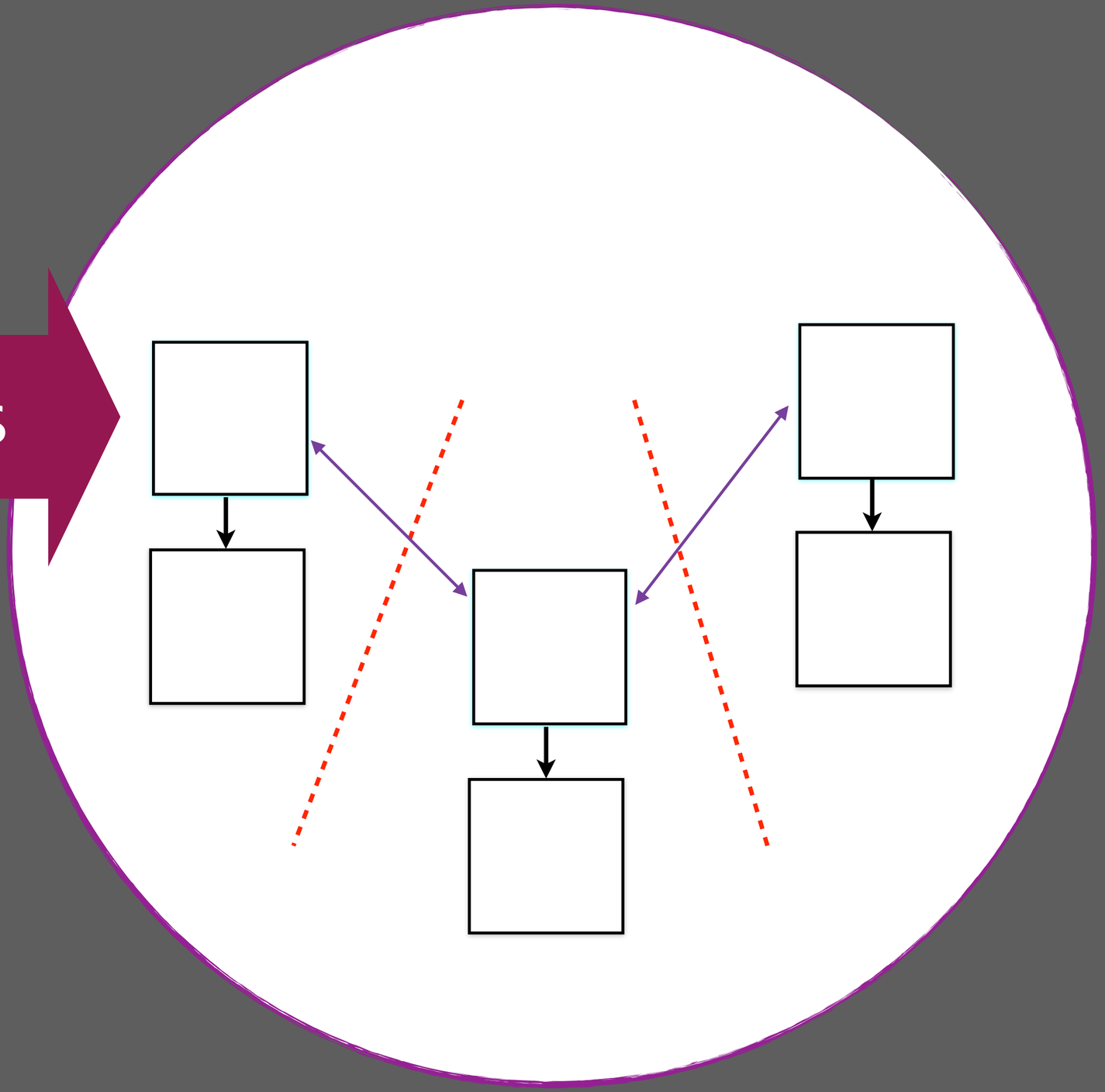

Extension
Closed


App
Module
Library
Closed for modification

Reuse-Release Equivalency Principle


App
Library
Reuse This

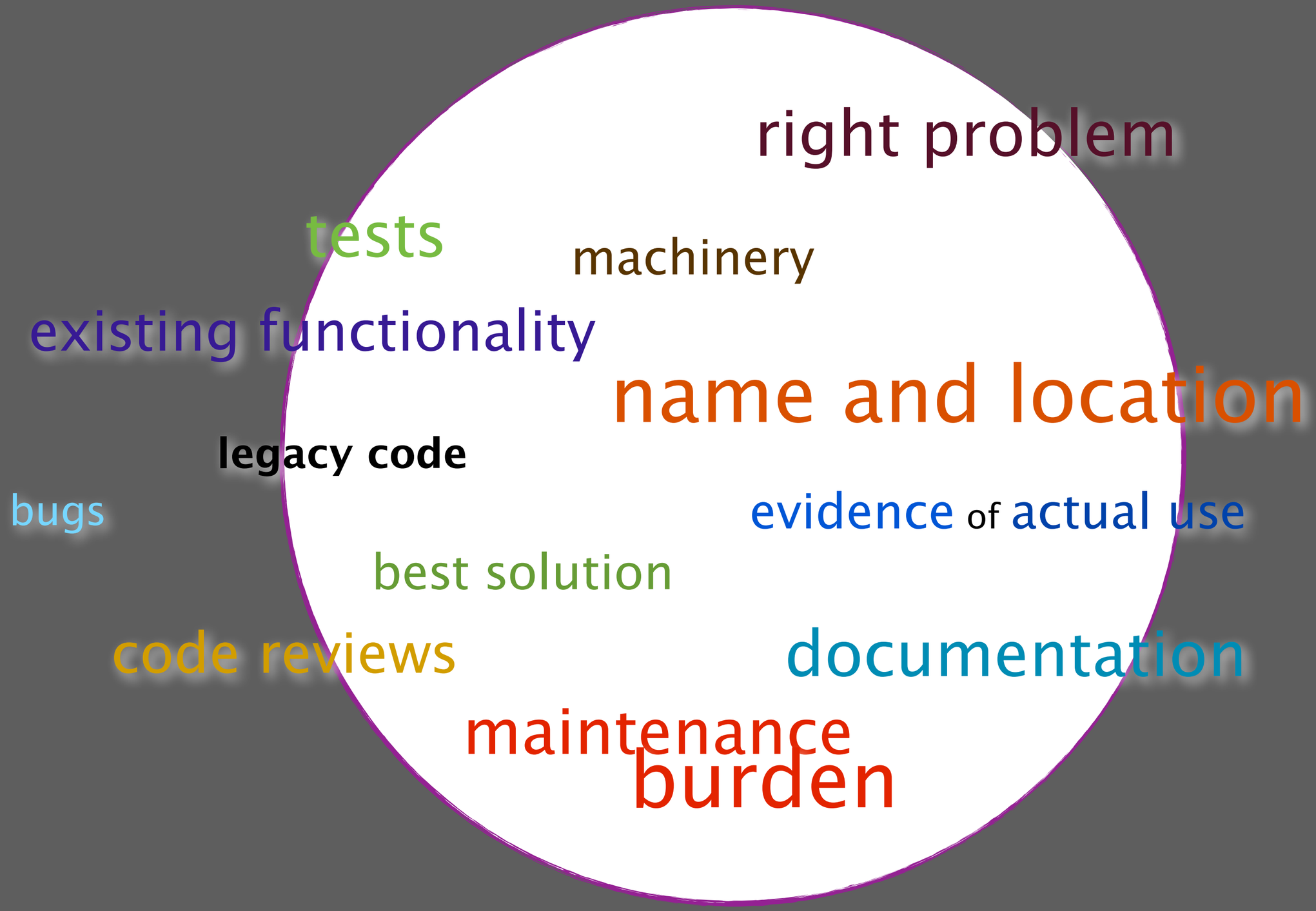
Reuse This



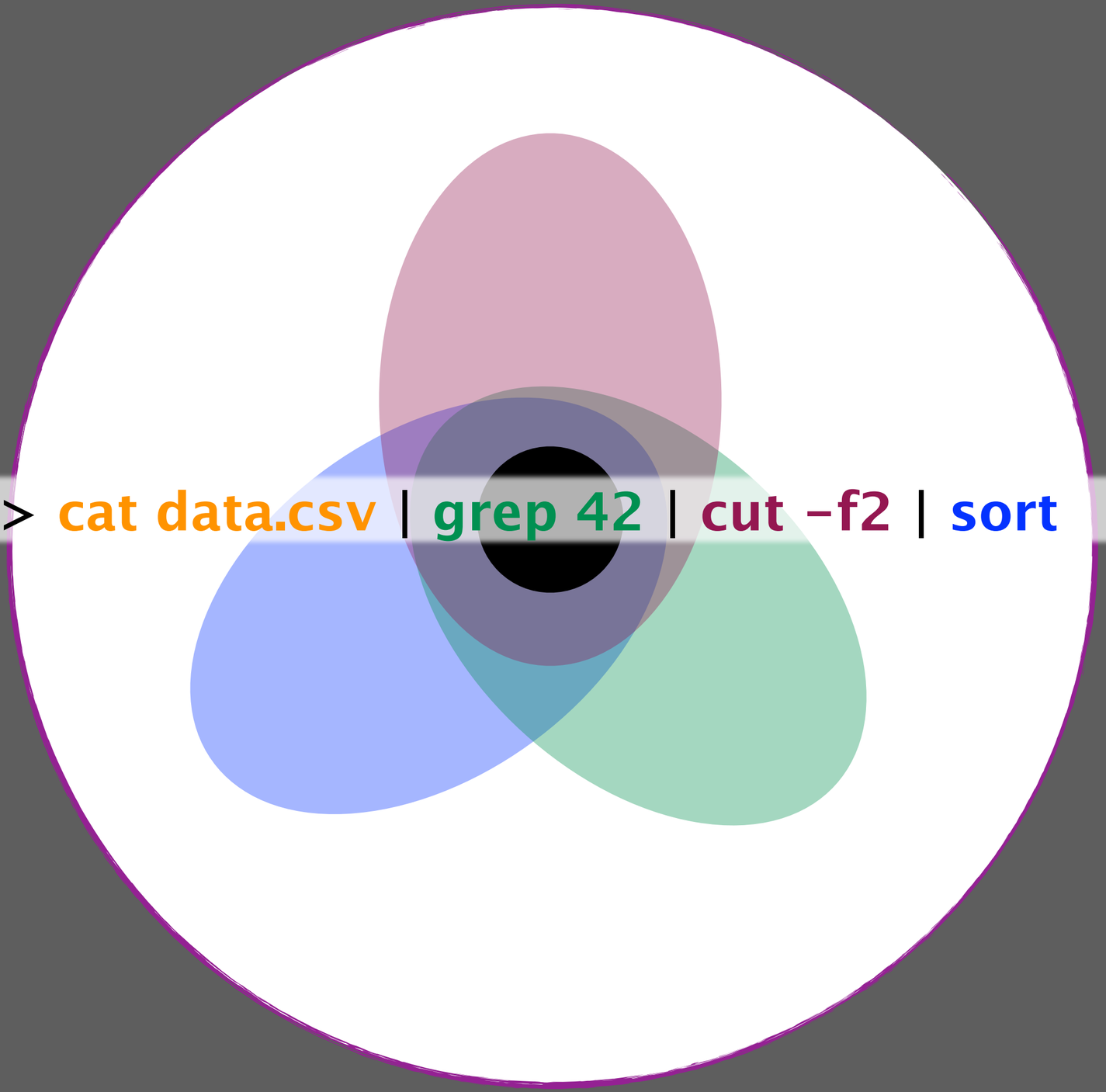
Planning for reuse is premature optimization

Donald Knuth

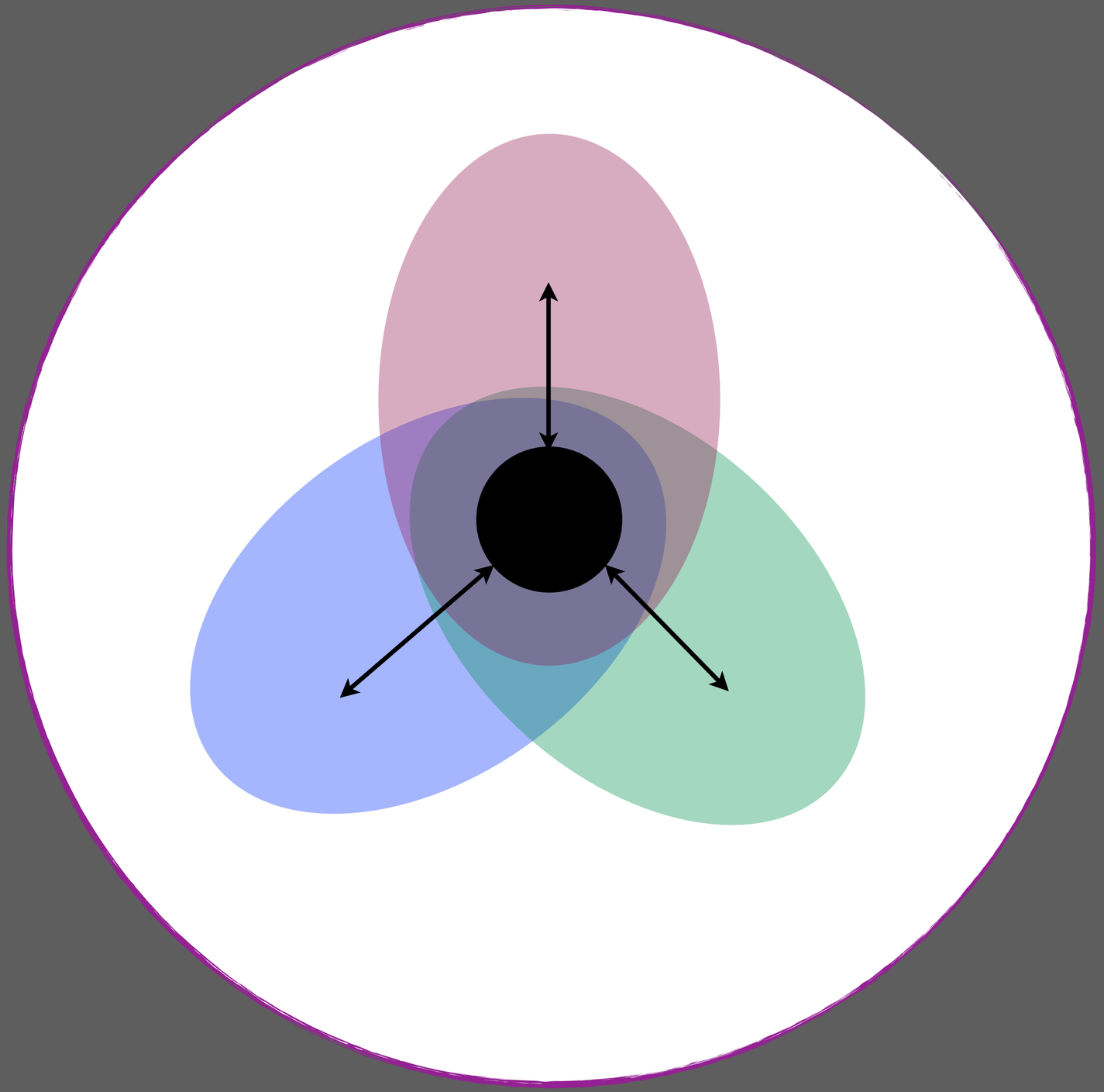
Premature optimization
is the root of all evil


right problem
tests
machinery
existing functionality
name and location
legacy code
bugs
evidence of actual use
best solution
code reviews
documentation
maintenance
burden

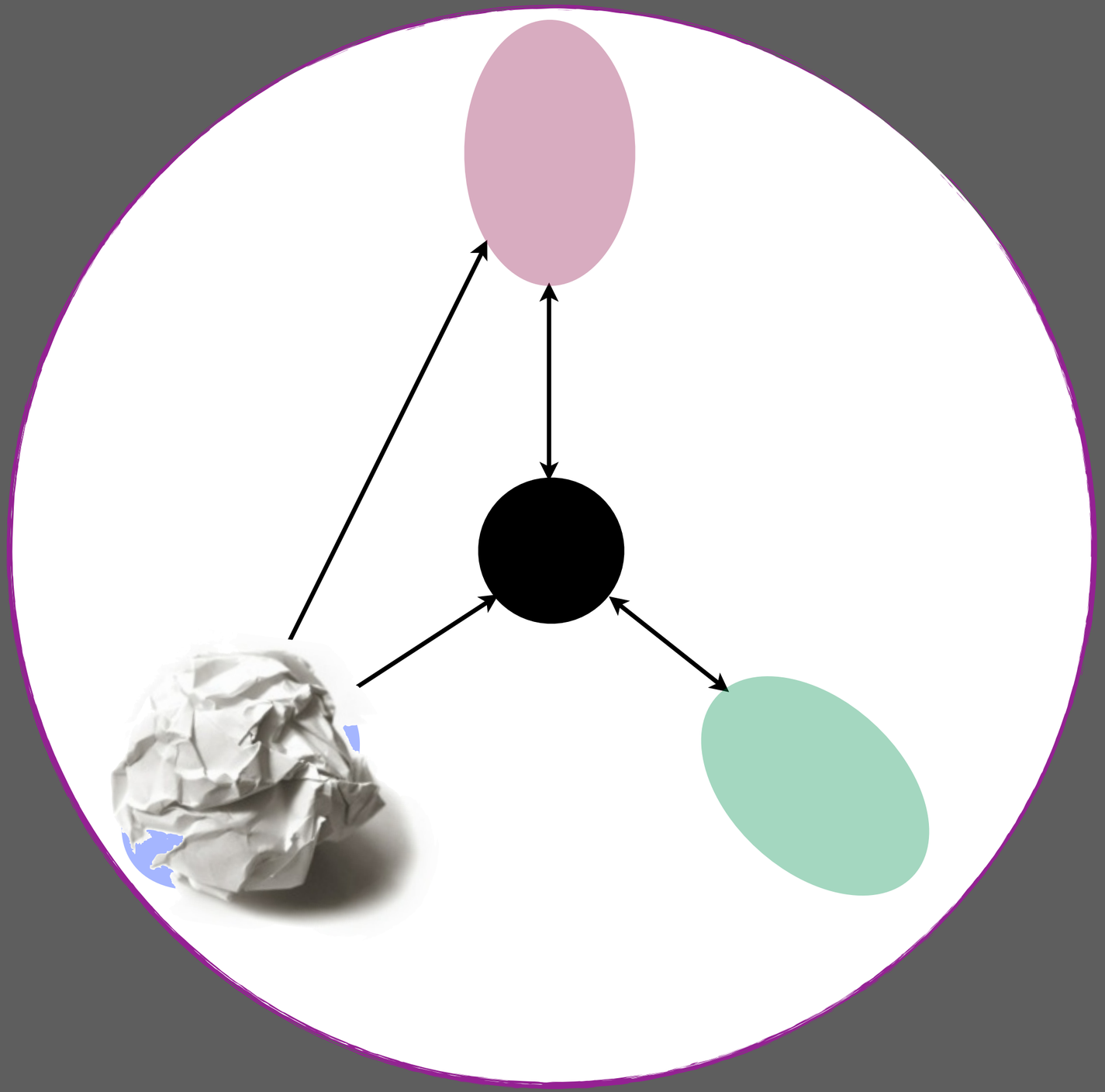
tests
name and location
backwards compatibility
best solution
bugs
code reviews
maintenance
burden



```
> cat data.csv | grep 42 | cut -f2 | sort
```



Tell Don’t Ask



Encapsulation

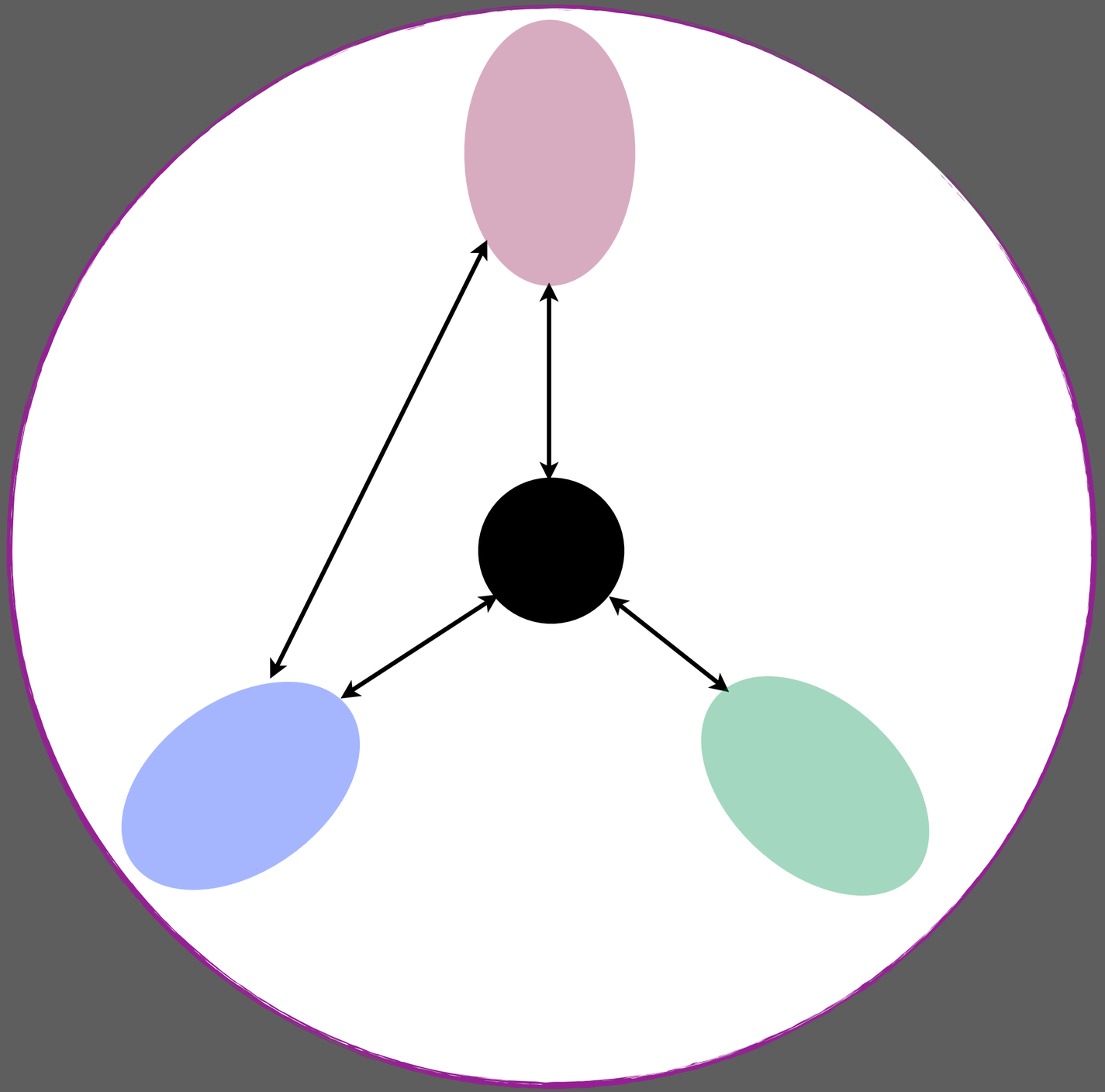
JS

Haskell

Ruby

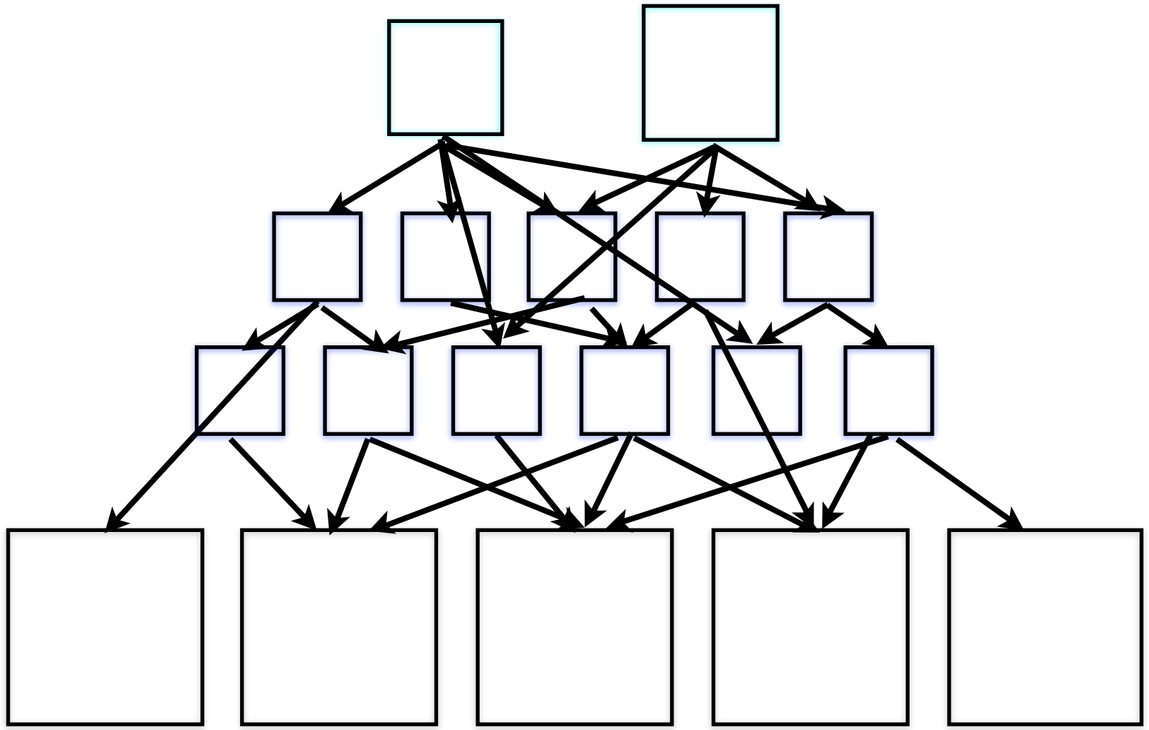
Java

Encapsulation



DRY

DRY



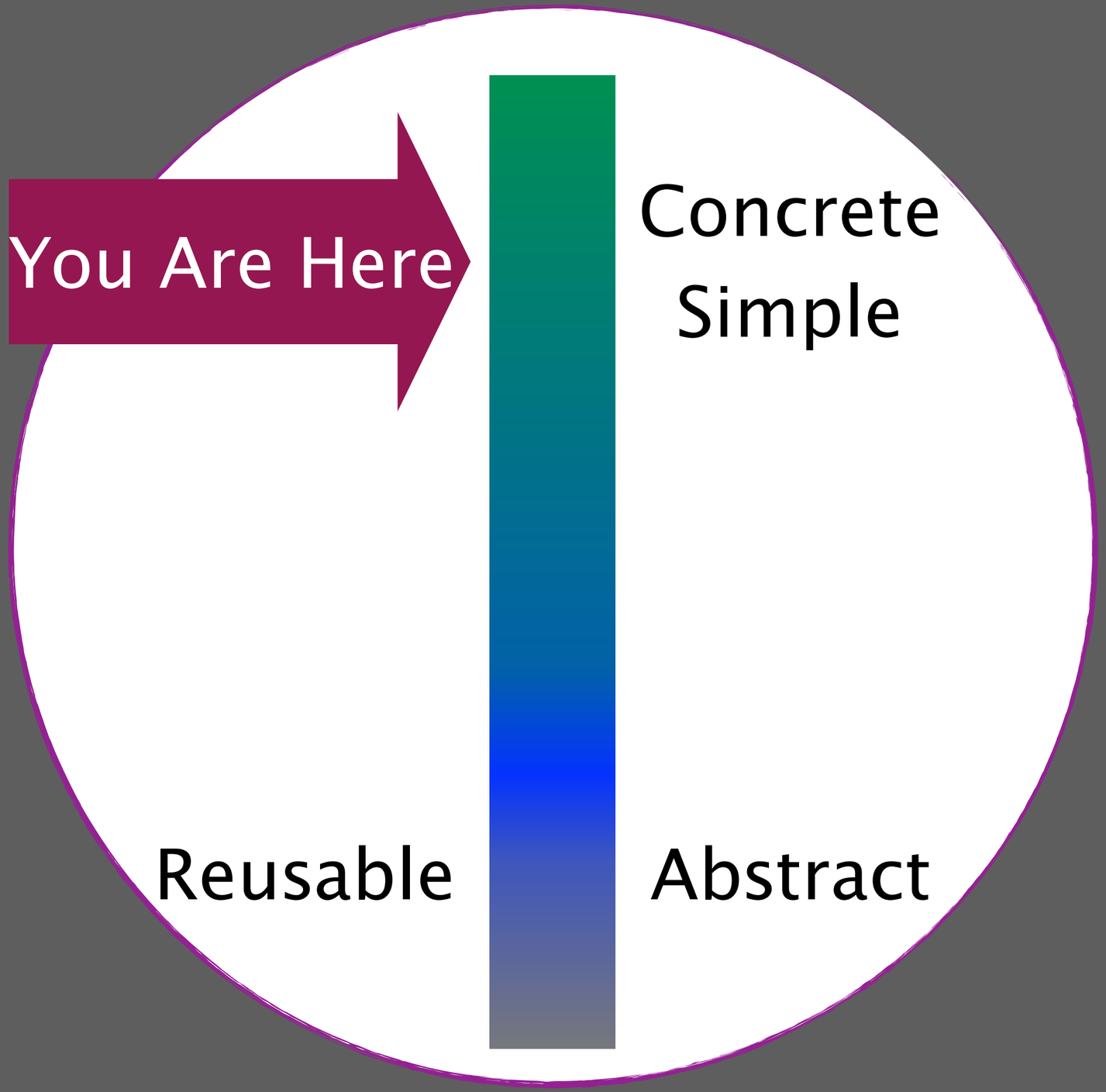
getTaxRate

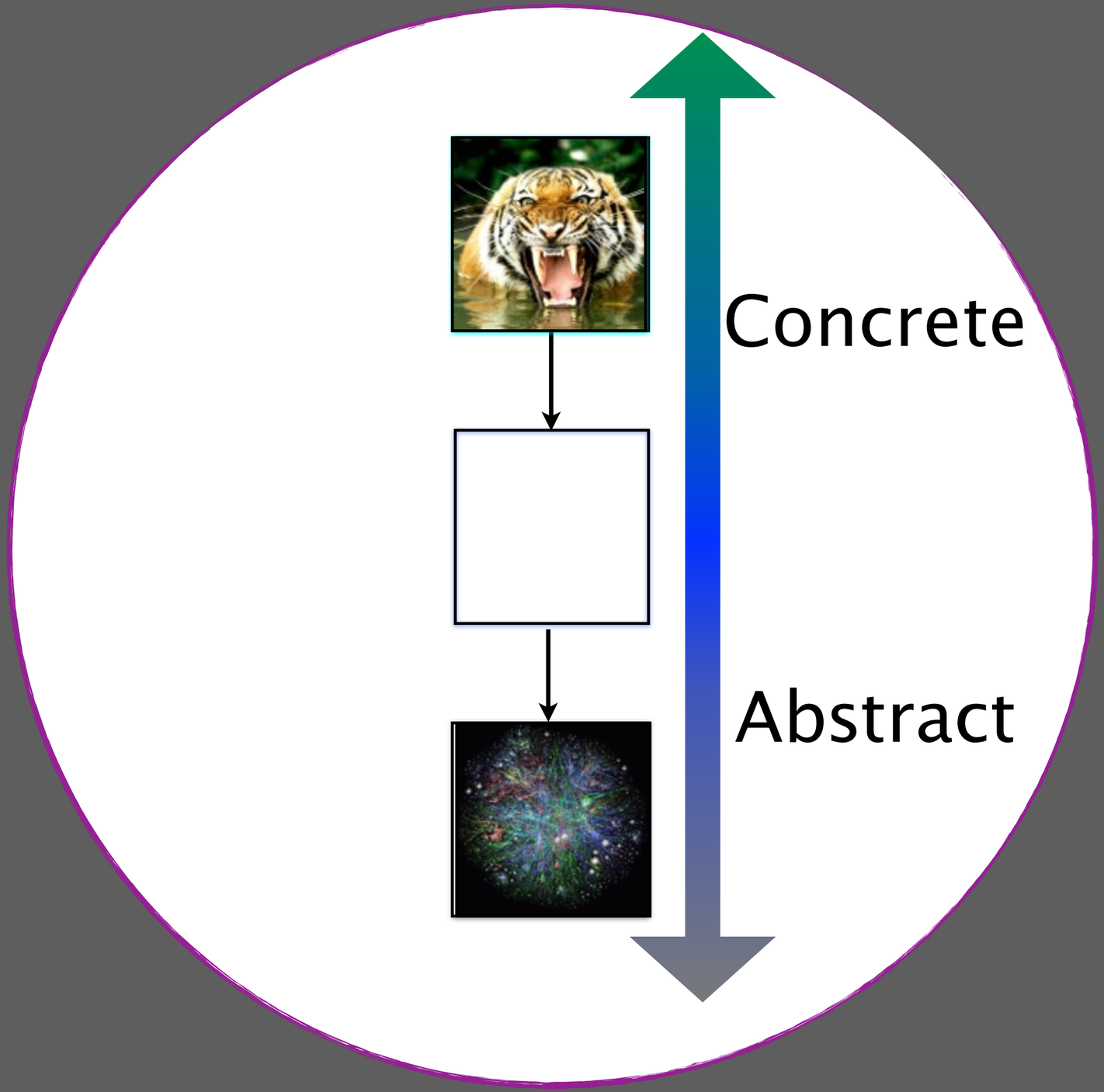
customerRewards

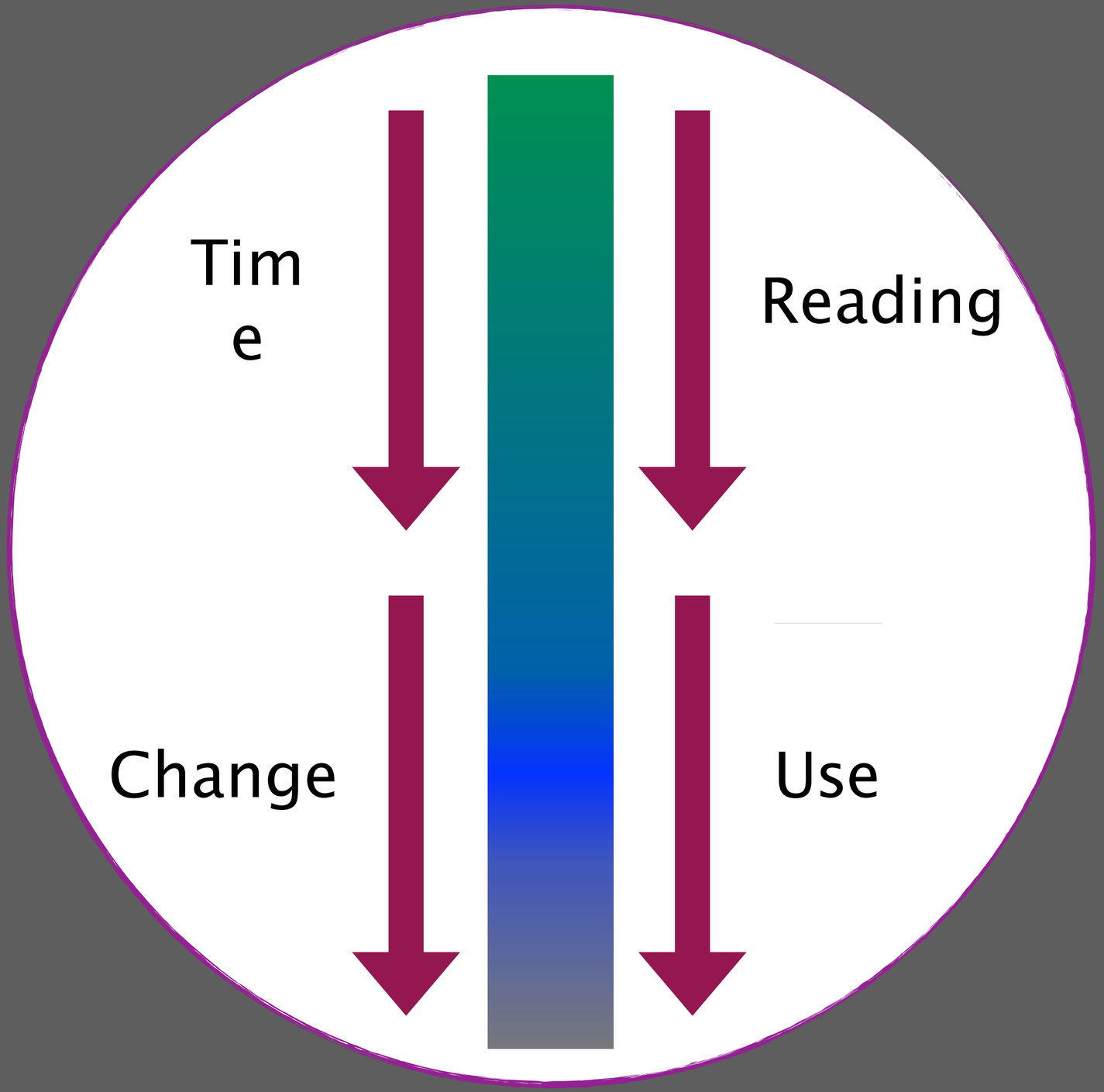
capitalize

formatDate

1. Make it work
2. Make it simple
3. Make it clever


You Are Here
Concrete
Simple
Reusable
Abstract


Concrete
Abstract


Tim
e
Reading
Change
Use

You
Ain't
Gonna
Need
It

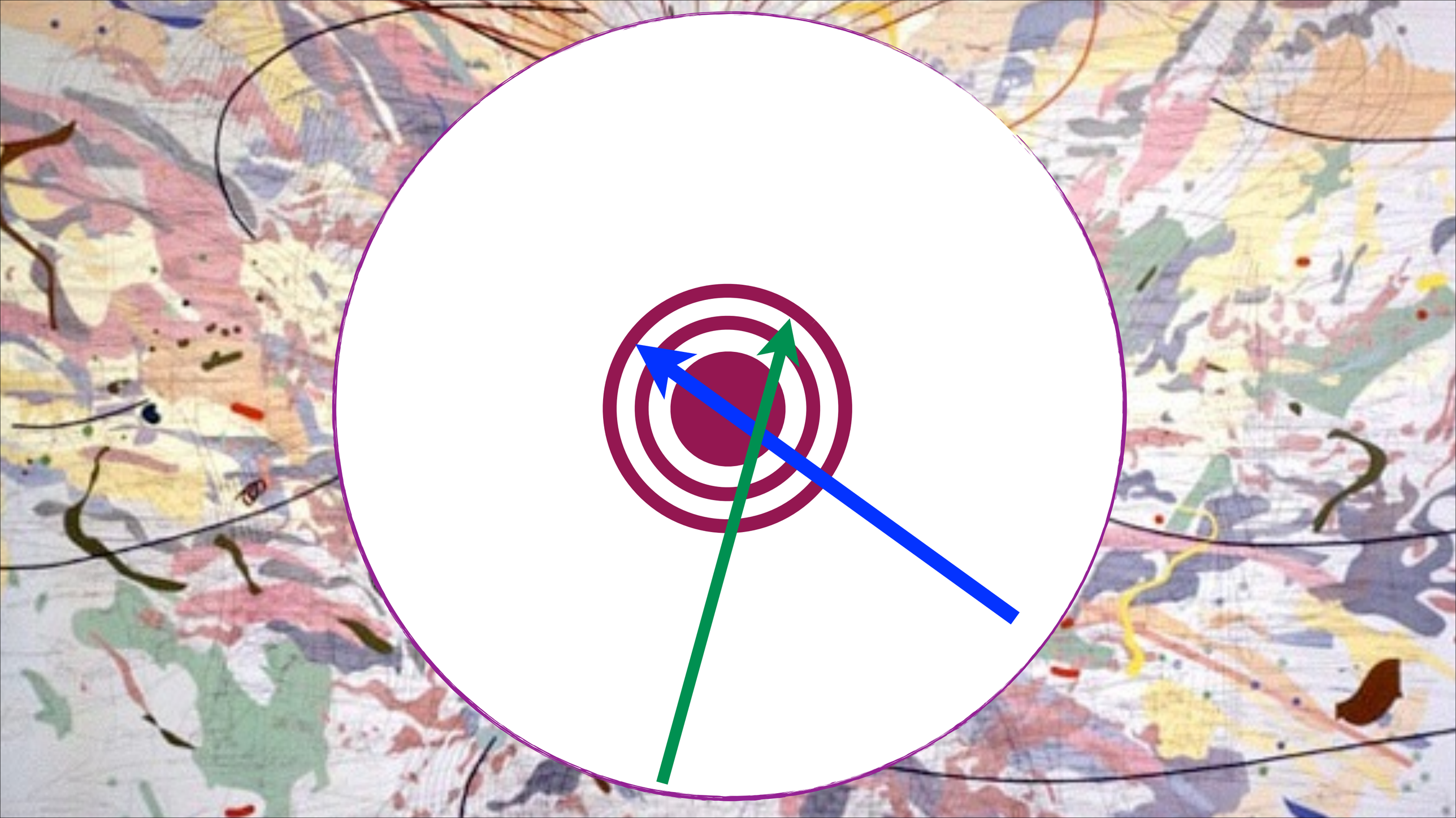
SOLID

+

Reuse/Release

+

DRY<-YAGNI



Small

Clear intent

older = better

Freedom

@jessitron