

# Intro to PBT



EBT

PBT

ScalaCheck

Choosing Properties

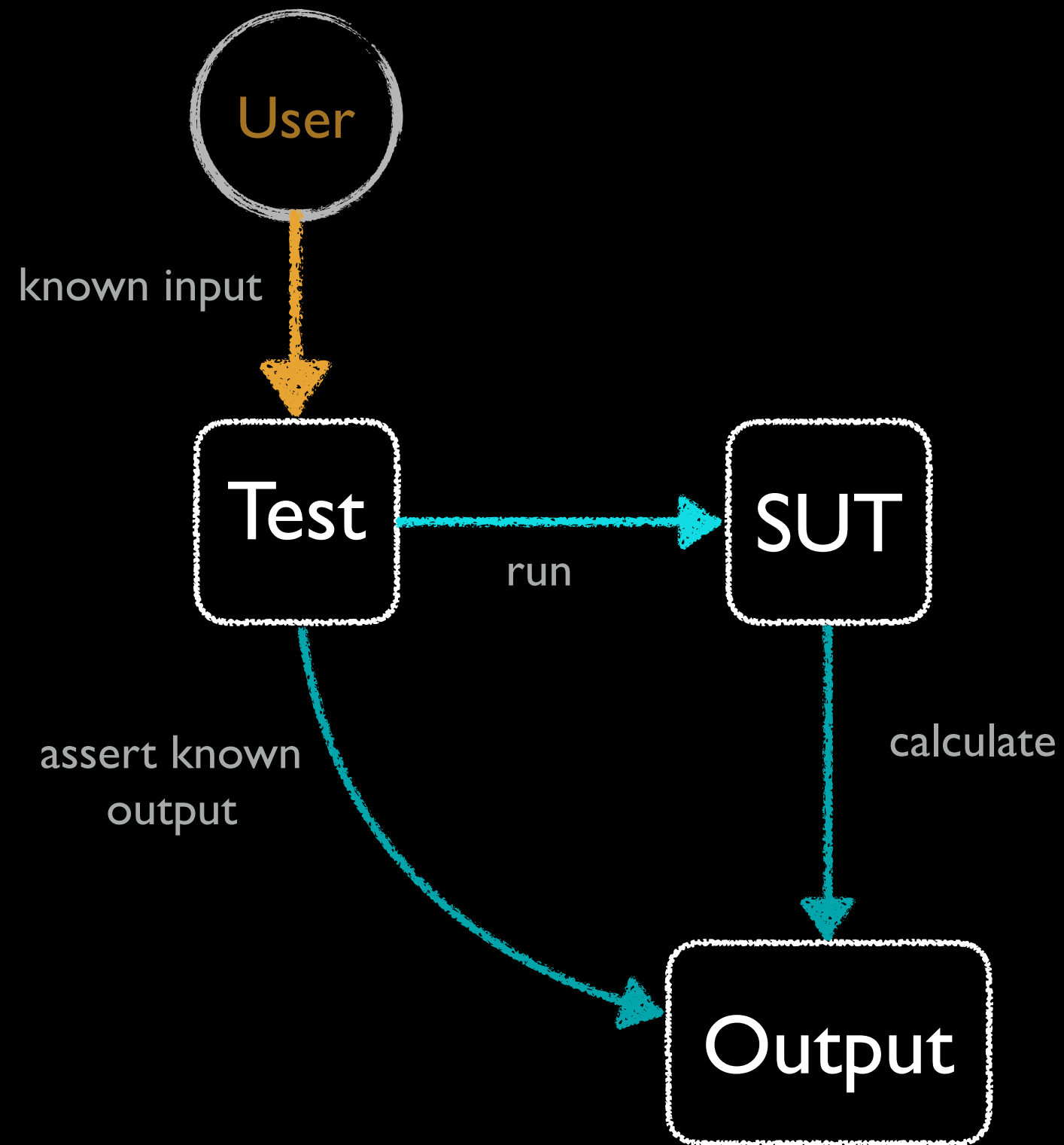
Examples

Summary

Testing shows the presence,  
not the absence of bugs

Dijkstra

# Example-Based Testing



known inputs

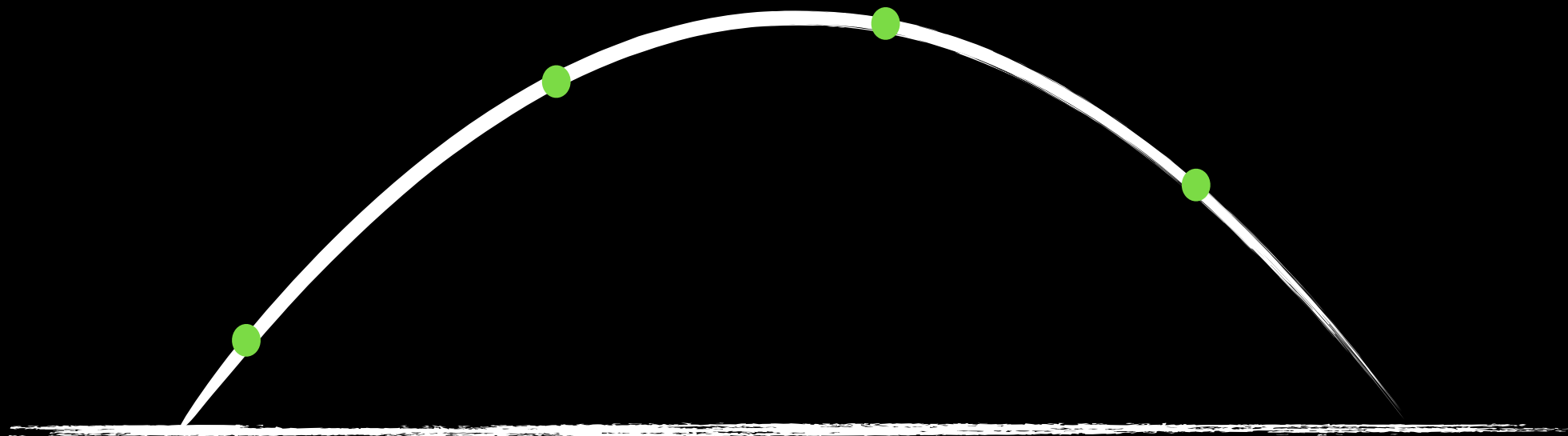
known output

add(**1**, **2**) should be **3**

The diagram illustrates the concept of known inputs and outputs for a function. It features the text "add(1, 2) should be 3" in white font on a black background. The numbers "1" and "2" are highlighted in red, and the number "3" is highlighted in green. Above the "1" and "2" is the text "known inputs" in white, with two red arrows pointing down to each number. Above the "3" is the text "known output" in white, with a red arrow pointing down to the number.



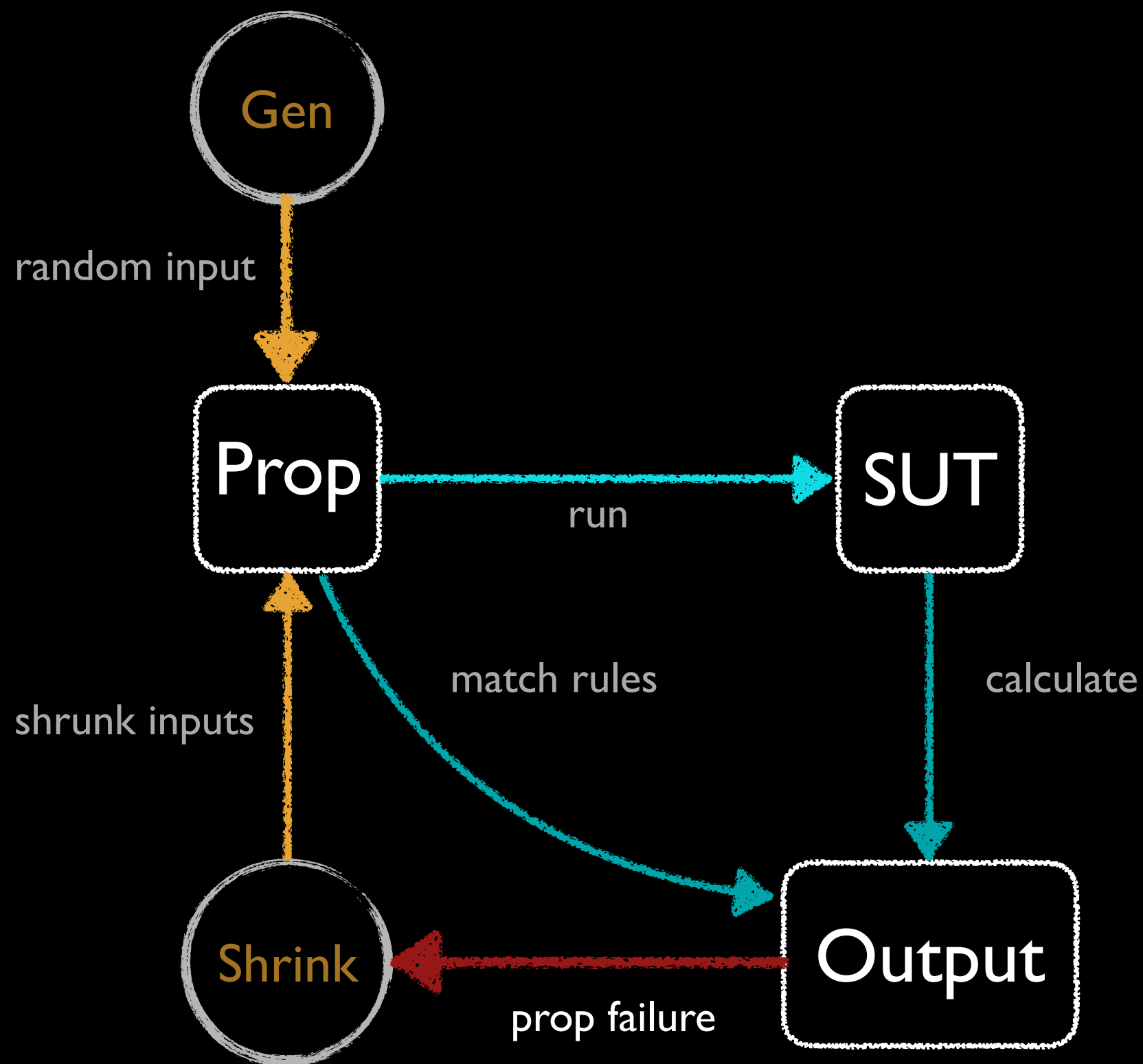




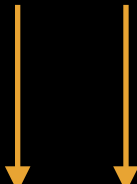
Input Sample

# Property-Based Testing

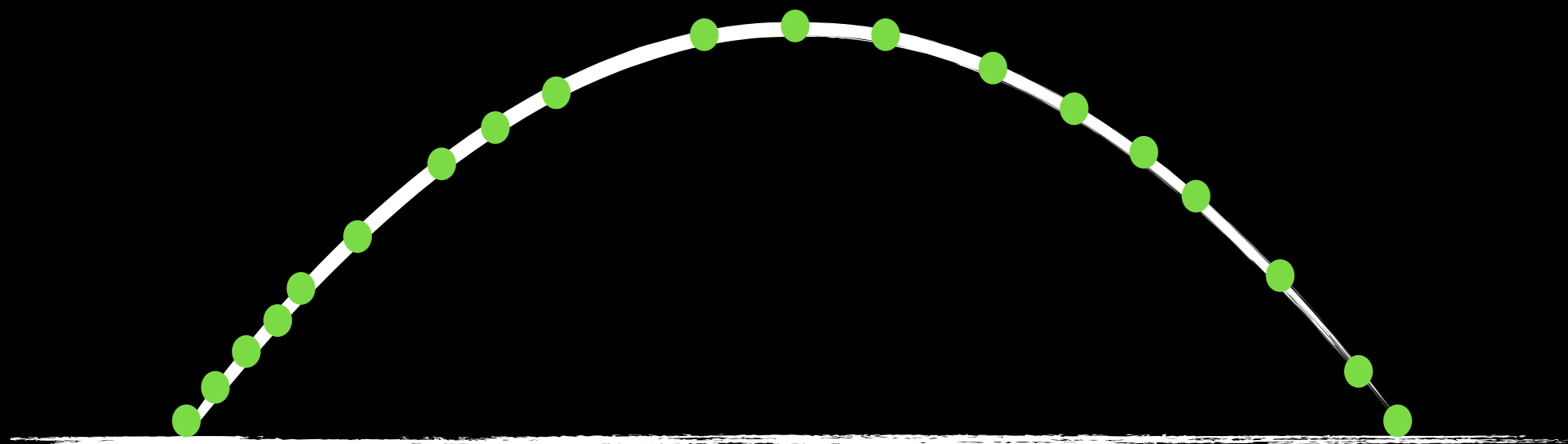
x100



any inputs


$$\text{add}(\textcolor{teal}{x}, \textcolor{teal}{y}) == \text{add}(\textcolor{teal}{y}, \textcolor{teal}{x})$$





Input Sample

ScalaCheck

Gen[A]



Arbitrary[A]





# Arbitrary[String]

ミリ噴奈[?]蠟●銀霄煲[?]鈦啐[?]隔耑授徻!飢瓊濊析Ω馱豪笛Lش醯「柁[?][?]فخ▼鑣𪛗旆◌𪛗令ㄣ耑霏c釧𪛗𪛗

`posNum[T](implicit Numeric[T]): Gen[T]`

`posNum[Int]`

25, 6, 56, 19, 9, 86, 94, 8, 20, 68

`frequency[T]((Int, Gen[T])*): Gen[T]`

```
frequency(  
  2 -> Gen.choose('A', 'Z'),  
  3 -> Gen.choose(1, 10)  
)
```

3, C, 8, 2, 9, D, 6, U, S, 4

`choose[T](min:T, max:T)`  
`(implicit Choose[T]):`  
`Gen[T]`

`choose('a', 'z')`

`d, u, f, z, b, m, f, z, f, m`

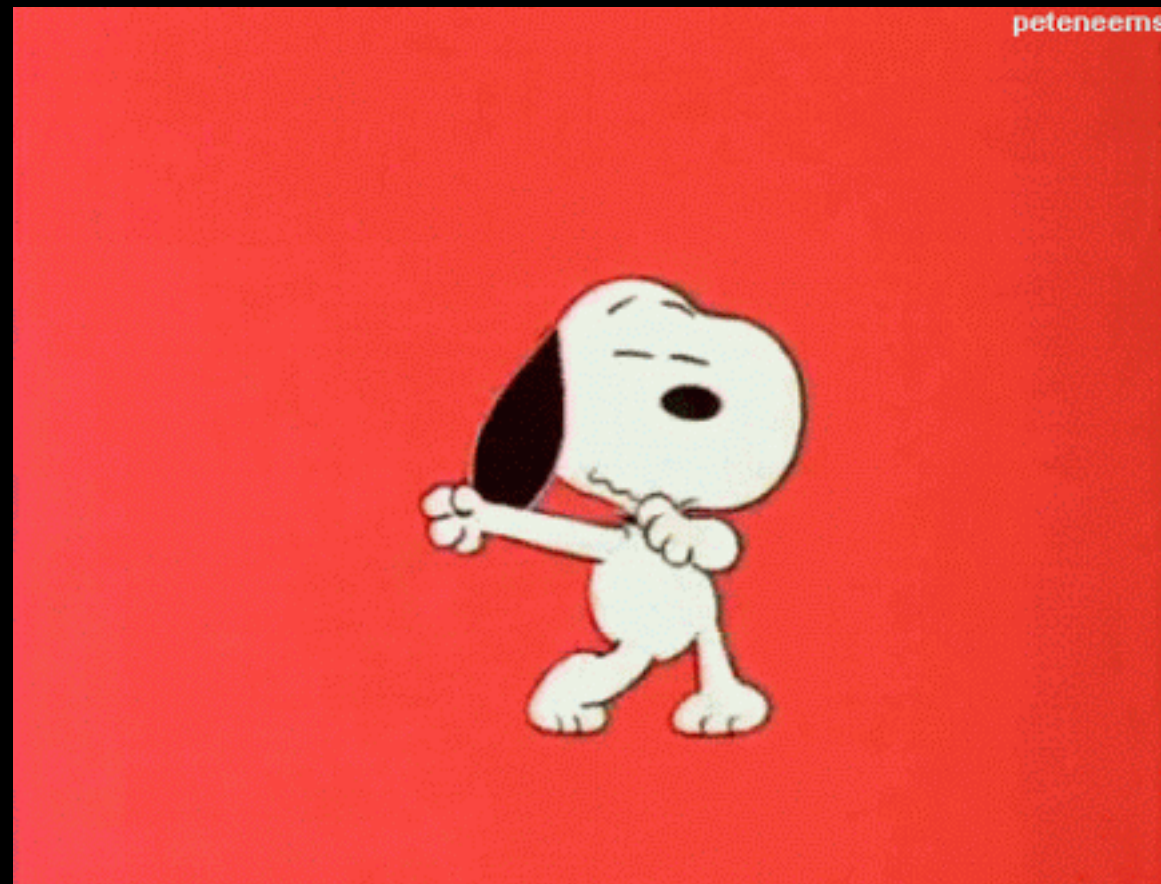
`Choose[Byte], Choose[Short], Choose[Char], Choose[Int], Choose[Long], Choose[Float],`  
`Choose[Double],`

`option[T](Gen[T]): Gen[Option[T]]`

`option(Gen.posNum[Int])`

`None, Some(2), None, Some(67), None, None, None, Some(97), Some(3), None`

NameGenerator



There are **many** more

<http://bit.ly/2oxLFLV>



Can be used with EBT

implicitly[Gen[T]].sample.get

```
Shrink[T] {  
  def shrink(x:T): Stream[T]  
}
```

```
implicitly[Shrink[Int]].shrink(100)  
List(50, -50, 25, -25, 12, -12, 6, -6, 3, -3, 1, -1, 0)
```

# [U]niversally Quantified Properties

=> Prop

=> Boolean

`forall[T I, P](gI: Gen[T I])  
 (T I  $\Rightarrow$  P)`

`(implicit p: (P)  $\Rightarrow$  Prop,`

`sI: Shrink[T I],`

`ppI: (T I)  $\Rightarrow$  Pretty)`

`:Prop`

**forAll**[T I, P](T I  $\Rightarrow$  P)

(implicit p: (P)  $\Rightarrow$  Prop,

a I: Arbitrary[T I],

s I: Shrink[T I],

pp I: (T I)  $\Rightarrow$  Pretty)

**:Prop**

# Choosing Properties

Math





# Laws

Associativity	$(a+b)+c$	$==$	$a+(b+c)$
Commutativity	$(a+b)$	$==$	$(b+a)$
Identity	$(a+\emptyset)$	$==$	$a$
	$(\emptyset+a)$	$==$	$a$
Distribution	$x(a+b)$	$==$	$xa+xb$

?ing



What does it **do**?

How is this **similar** to ...?

How is this **different** from ...?

Can I verify this **without** duplicating the CUT?

What will make it **fail**?

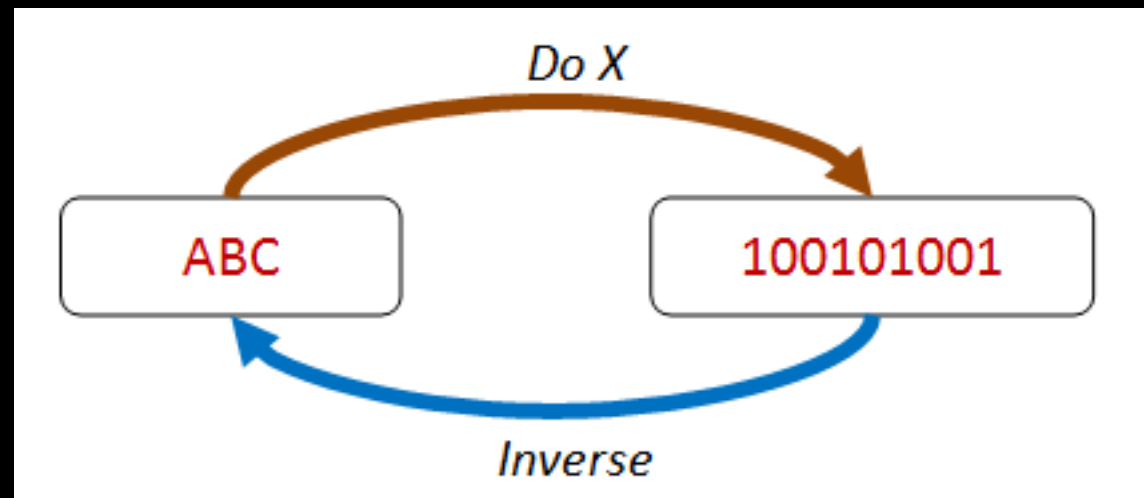
# Patterns

# Invariants

Length  
Contents

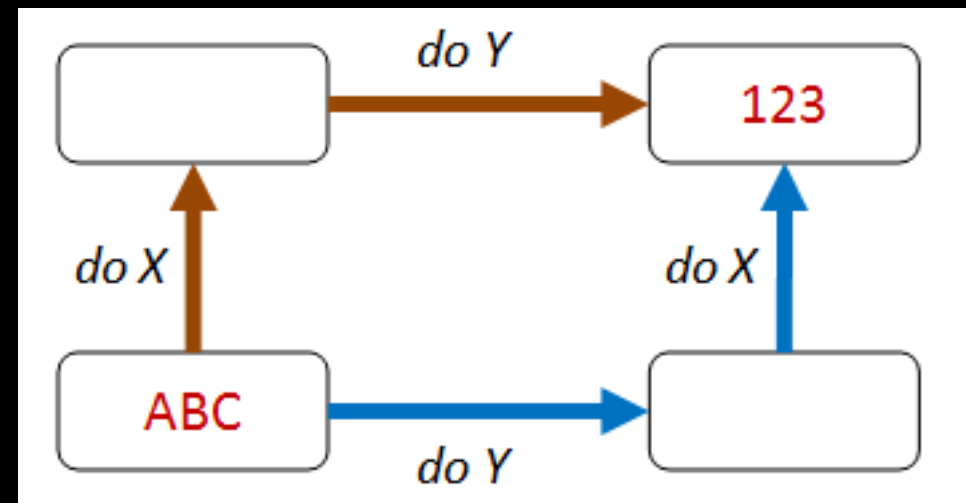
`list.sorted.length == list.length`

# Round-tripping



`json.parse.toJson == json`

# Different Order Same Result



`list.map(_ + 1).sorted == list.sorted.map(_ + 1)`



# Compose Methods

```
list2.reverse ++ list1.reverse  
==  
(list1 ++ list2).reverse
```

# Test Oracle

Verify against another implementation

multithreaded result == single-threaded result  
jsonLibX result == jsonLibY result



# There are others

<http://bit.ly/2o6DKsy>

# Examples

# Addition

ToAsciiUPPERCase

# PBT's got your Back

```
[info] ! ToUpperCase.All non lowercase characters must be at the same positions: Falsified after 43 passed tests.  
[info] > ARG_0: "꺆"  
[info] > ARG_0_ORIGINAL: "램 다 굶 1龜襪 2[?]簞 順틀 [?]"
```

REA Robot



dets



Before



After



- Files over 1GB?
- Rehashing?
- > 6 weeks of effort!

- Database with *one* record!
- 5—6 calls to reproduce
- < 1 day to fix



Clojure/West

March 24-26 2014

The Palace Hotel San Francisco





## Bug #4

### Prefix:

```
open_file(dets_table, [{type, bag}]) --> dets_table  
close(dets_table) --> ok  
open_file(dets_table, [{type, bag}]) --> dets_table
```

### Parallel:

1. `lookup(dets_table, 0)` --> `[]`
2. `insert(dets_table, {0, 0})` --> `ok`
3. `insert(dets_table, {0, 0})` --> `ok`

**Result:** `ok`



Clojure/West

March 24-26 2014

The Palace Hotel San Francisco



# Summary

# EBT

Easy to understand

Quick to implement

Good for implementations with few combinations

Needed for regression

---

Limited by developers imagination

Hard to test complex implementations

Boring to write

# PBT

Edge cases for free

Hundreds of tests

Reusable Generators

More thinking involved

Good for complex implementations

---

Requires investment in learning techniques

More thinking involved

Have to write Generators/Shrinkers

Not good for regression



EBT + PBT = WIN





# EBT

Basic cases

Regression (bugs)

# PBT

Edge cases



# Links

[The lazy programmer's guide to writing 1000's of tests - Scott Wlaschin](#)

[I Dream of Gen'ning - Kelsey Gilmore-Innis](#)

[Practical Property-Based Testing - Charles O'Farrell](#)

[Property-Based Testing for Better Code - Jessica Kerr](#)

[Testing the Hard Stuff and Staying Sane - John Hughes](#)

Thank You!

---