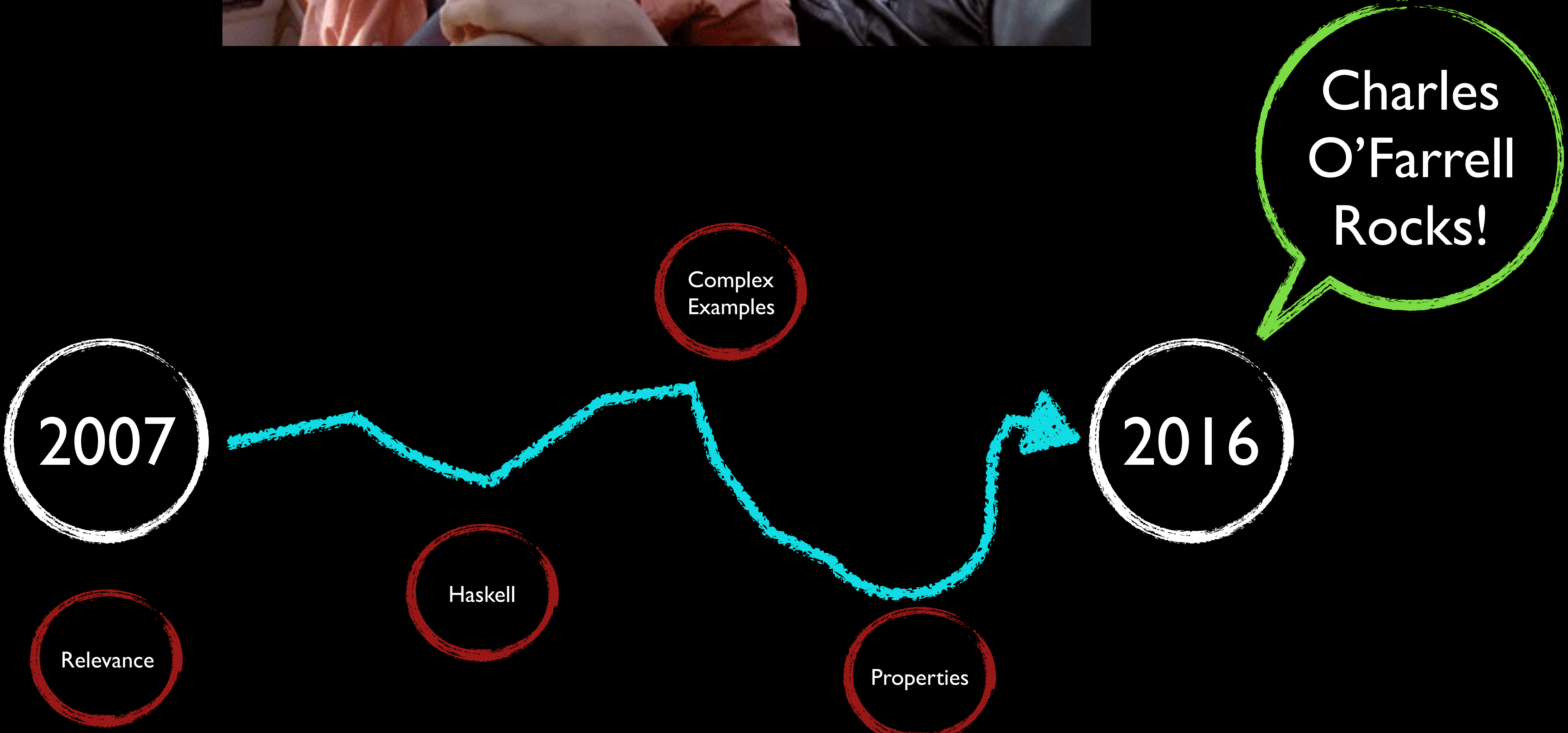
Intro to PBT



2007

Relevance

Haskell

Complex Examples

Properties

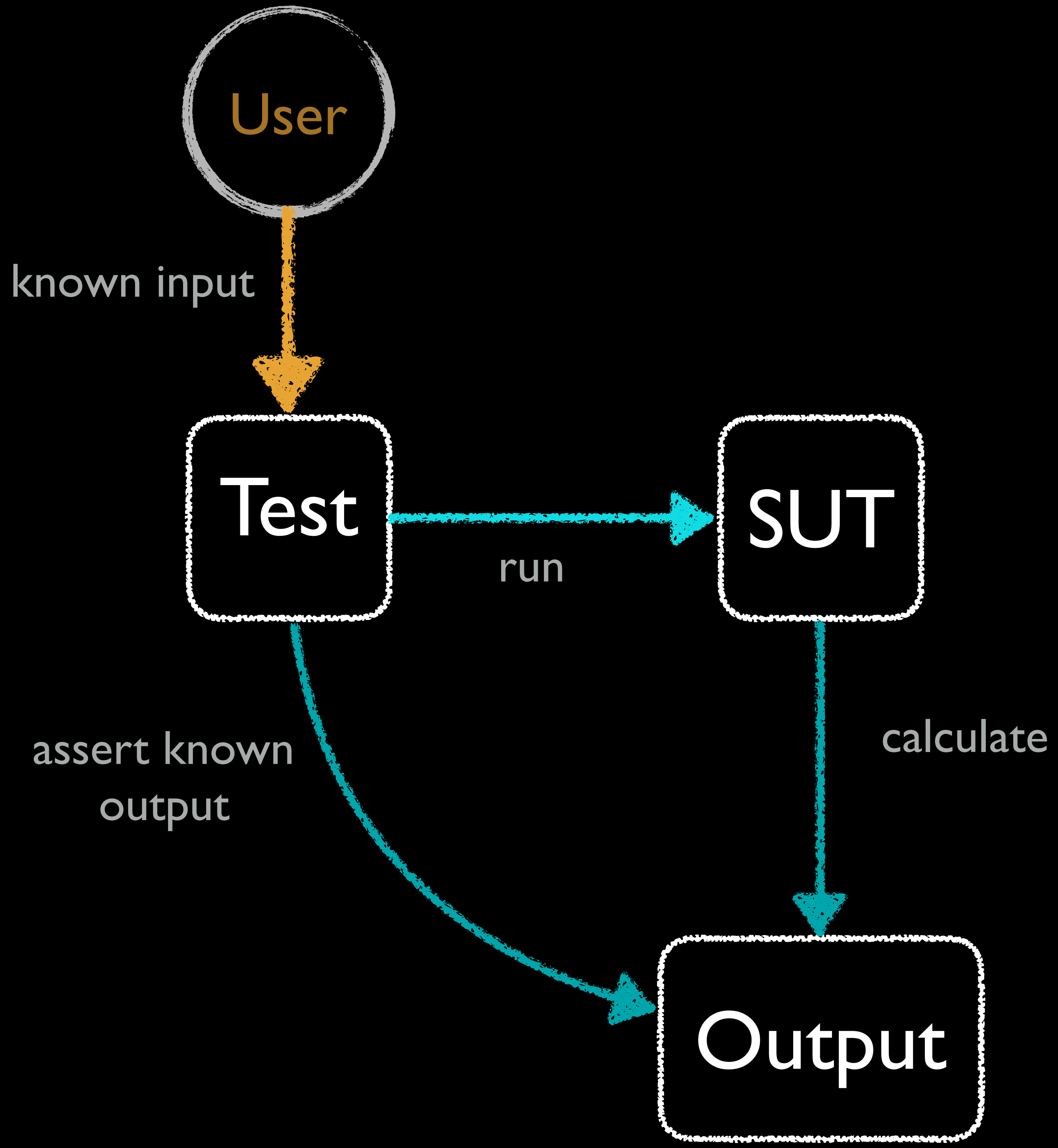
2016

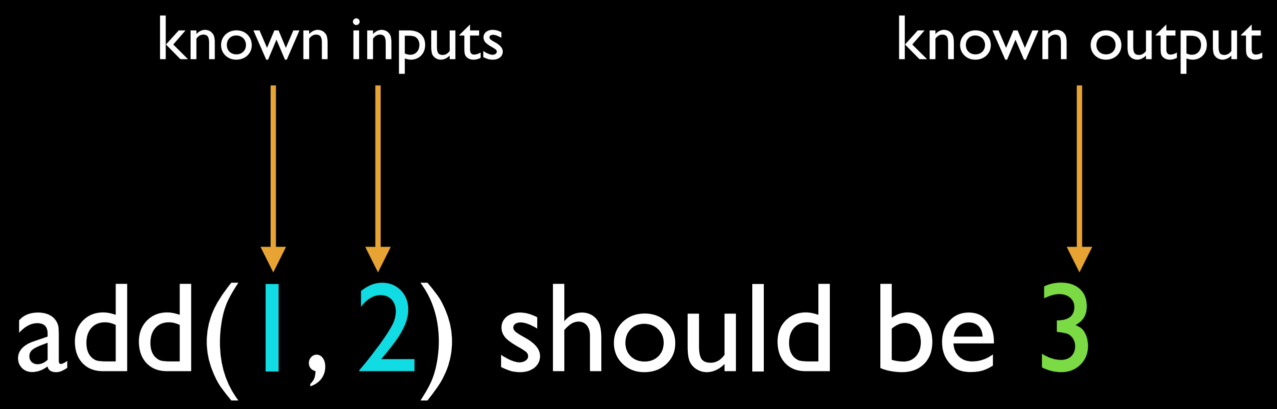
Charles O’Farrell Rocks!

EBT
PBT
ScalaCheck
Choosing Properties
Examples
Summary

Testing shows the presence,
not the absence of bugs
Dijkstra

Example-Based Testing


User
known input
Test
run
SUT
calculate
assert known output
Output

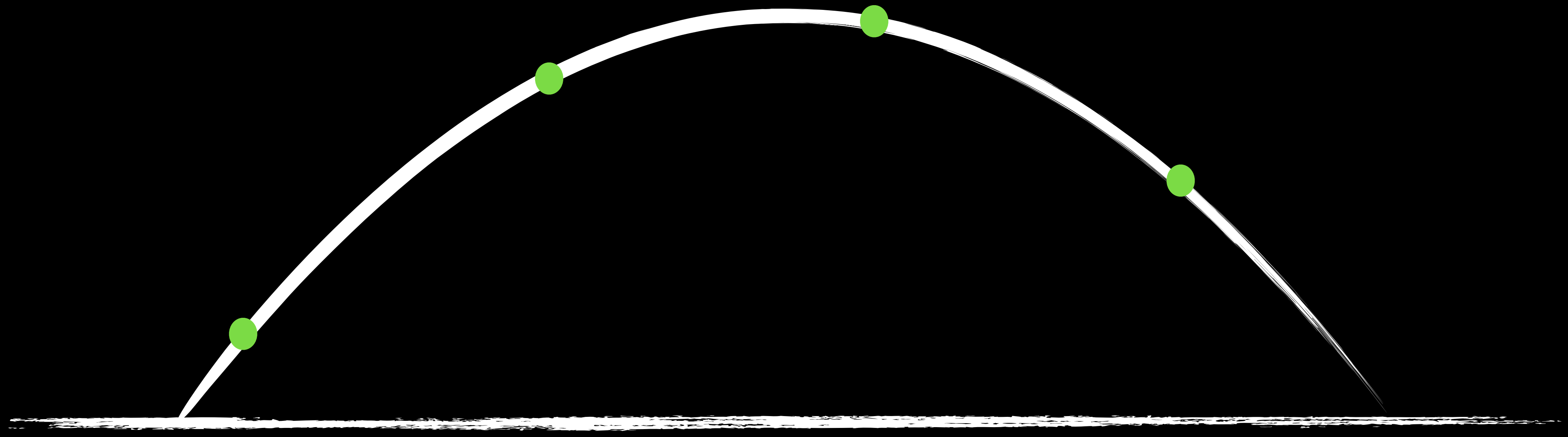


known inputs

known output

add(1, 2) should be 3

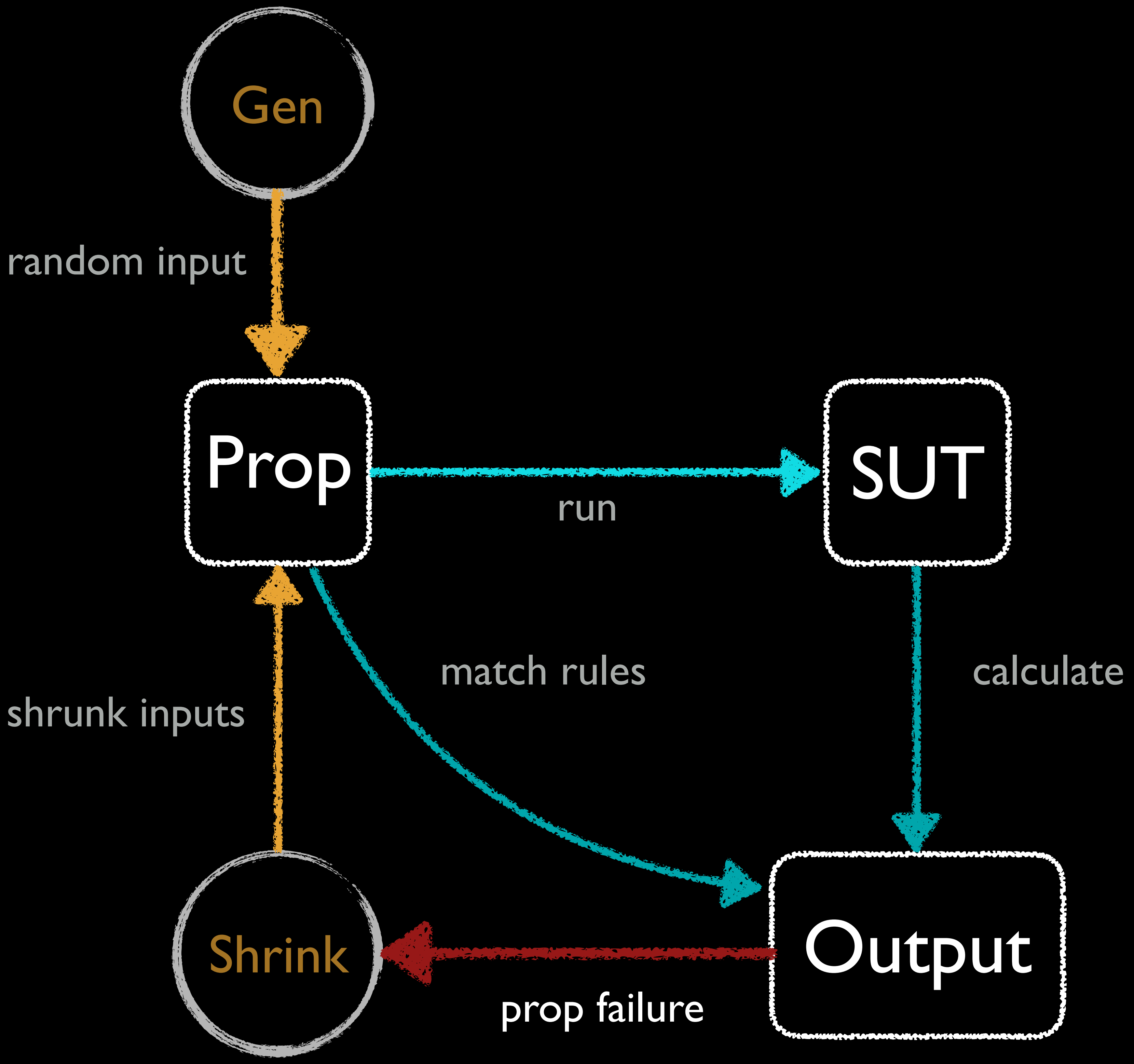
SENORGIF.COM



Input Sample

Property-Based Testing

x100



Gen

random input

Prop

run

SUT

match rules

calculate

shrunk inputs

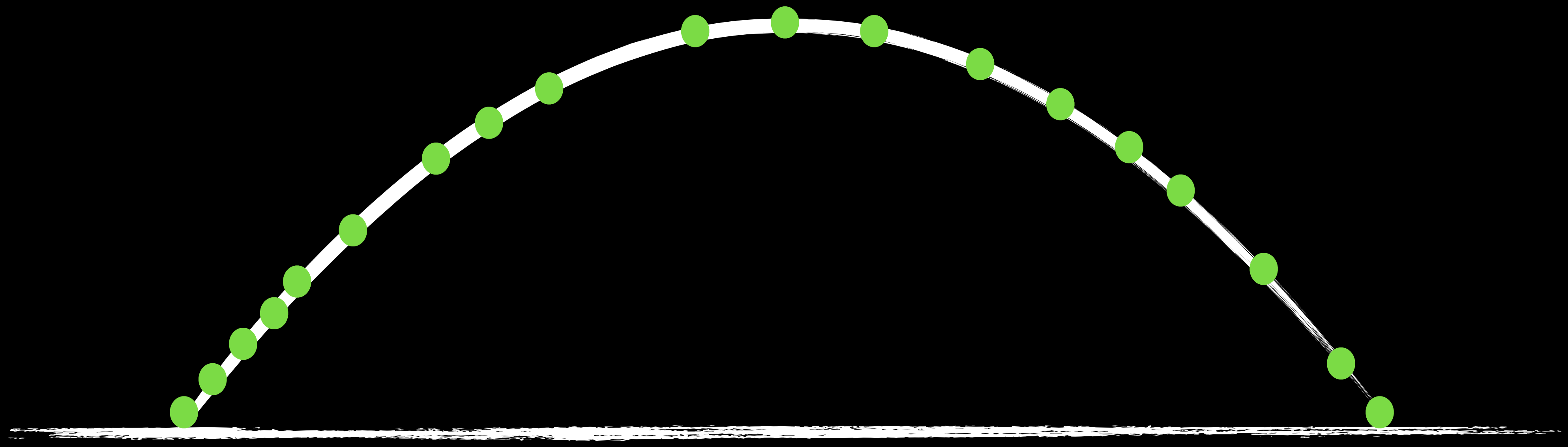
Shrink

prop failure

Output

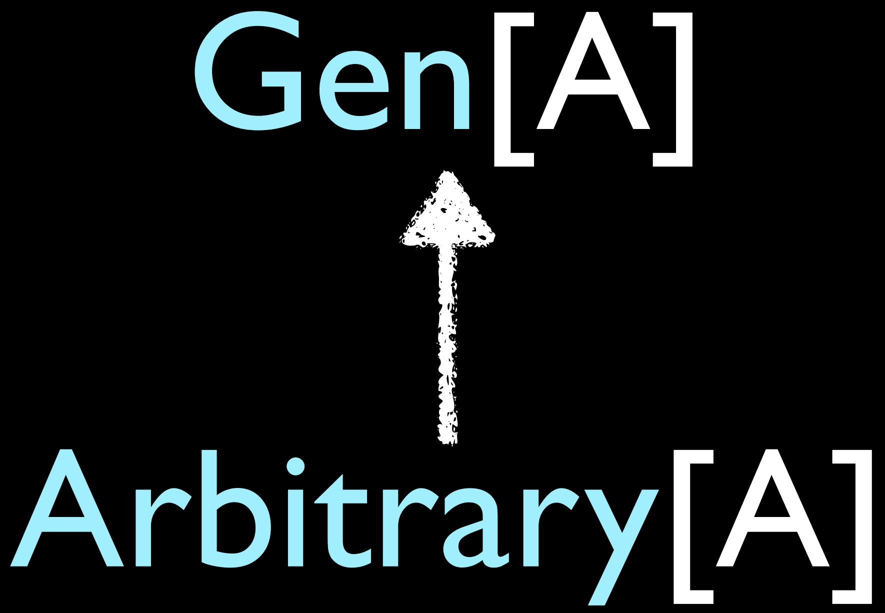
any inputs

```
add(x, y) == add(y, x)
```



Input Sample

ScalaCheck



Gen[A]

Arbitrary[A]

Sherman Brothers

Arbitrary[String]

ﾐﾘ嚽夵⍰蟵●銀霌煲⍰釷哰⍰隔畓授僐¦㌭Ձ璦瀎析ὤ㗥豪篟Lڜ䜖｢柹⍰⍰فخ⏬鑣걺尯◌୭令Ⴤ듷鄹霏ᄃ釗瞛흰

```
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
```

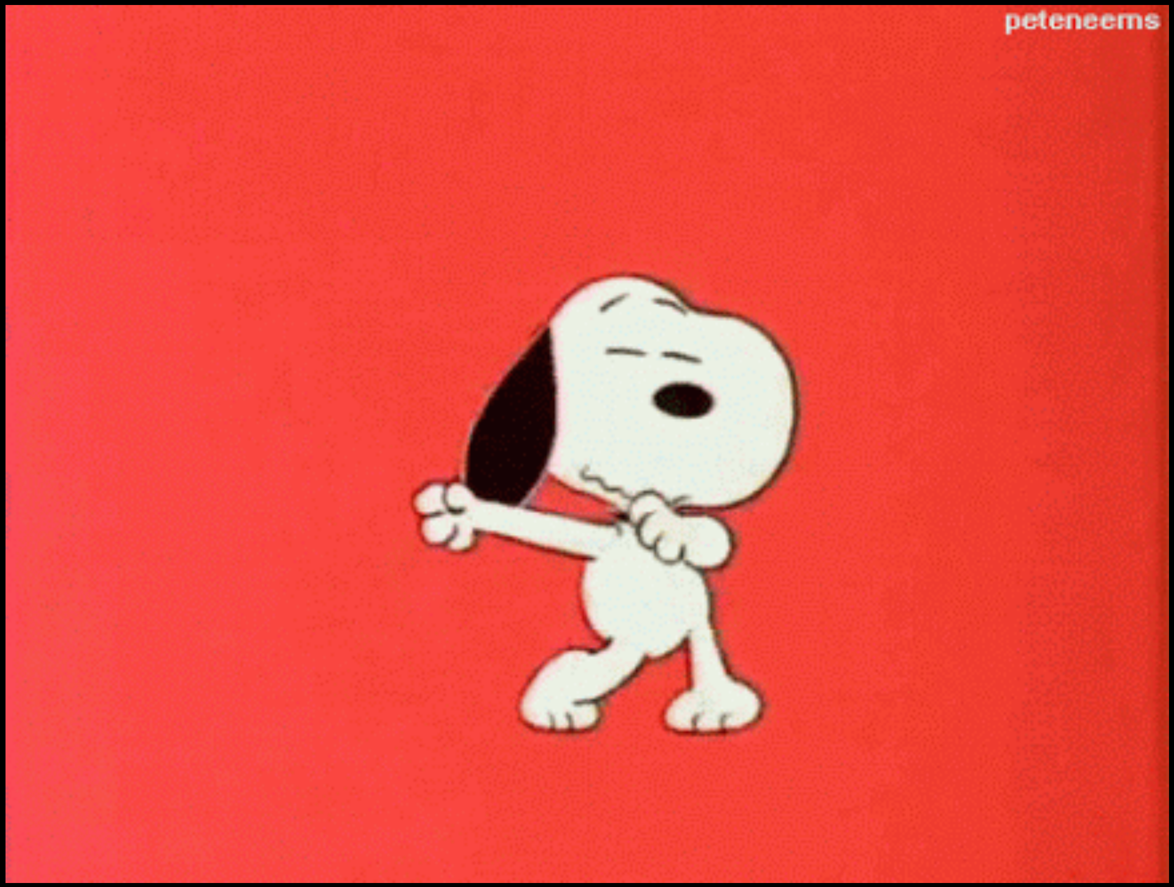
Choose[Byte], Choose[Short], Choose[Char], Choose[Int], Choose[Long], Choose[Float], Choose[Double],

```
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[A] {
  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[T1, P](g1: Gen[T1])
(T1 ⇒ P)
(implicit p: (P) ⇒ Prop,
s1: Shrink[T1],
pp1: (T1) ⇒ Pretty)
:Prop
```

```
forAll[T1, P](T1 ⇒ P)
(implicit p: (P) ⇒ Prop,
a1: Arbitrary[T1],
s1: Shrink[T1],
pp1: (T1) ⇒ Pretty)
:Prop
```

Choosing Properties

Mλth

Laws

Associativity	$(a+b)+c$	==	$a+(b+c)$
Commutativity	$(a+b)$	==	$(b+a)$
Identity	$(a+\varnothing)$	==	a
	$(\varnothing+a)$	==	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?

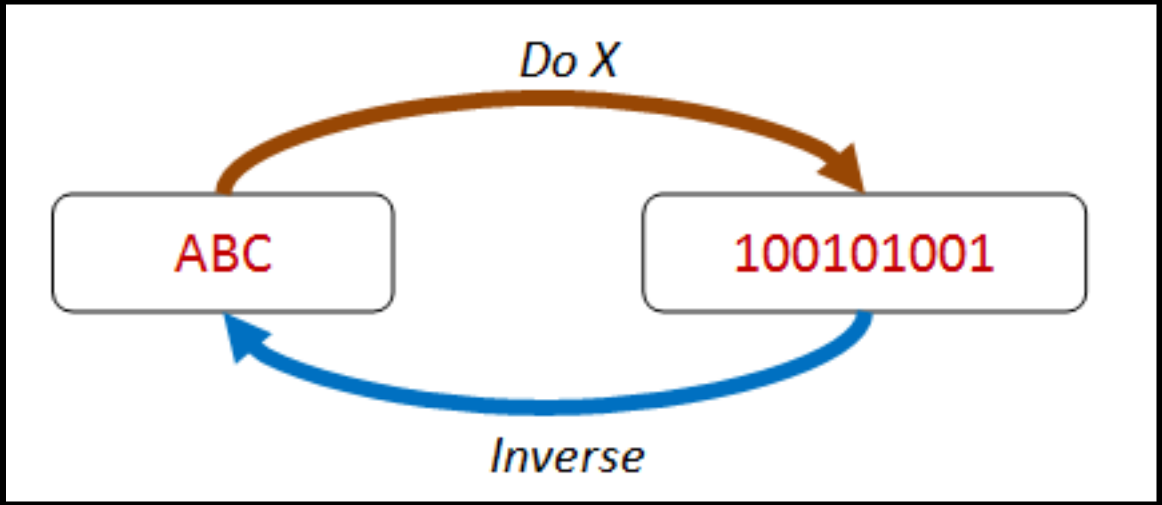
Patterns

Invariants

Length
Contents

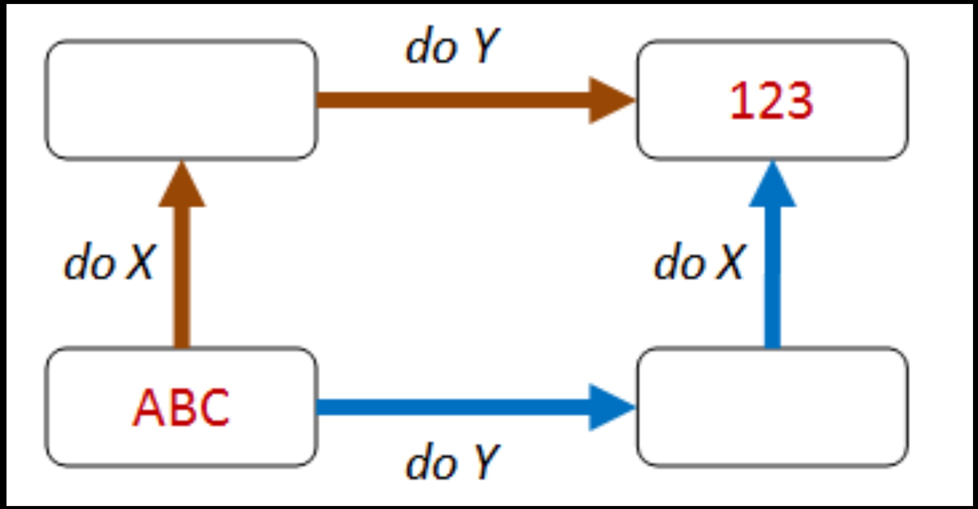
```
list.sorted.length == list.length
```

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

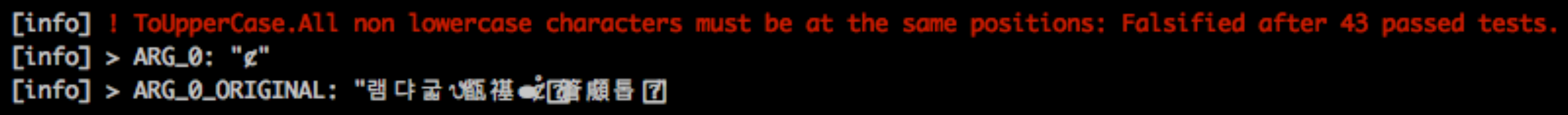
http://bit.ly/2o6DKsy

Examples

Addition

ToAsciiUPPERCase

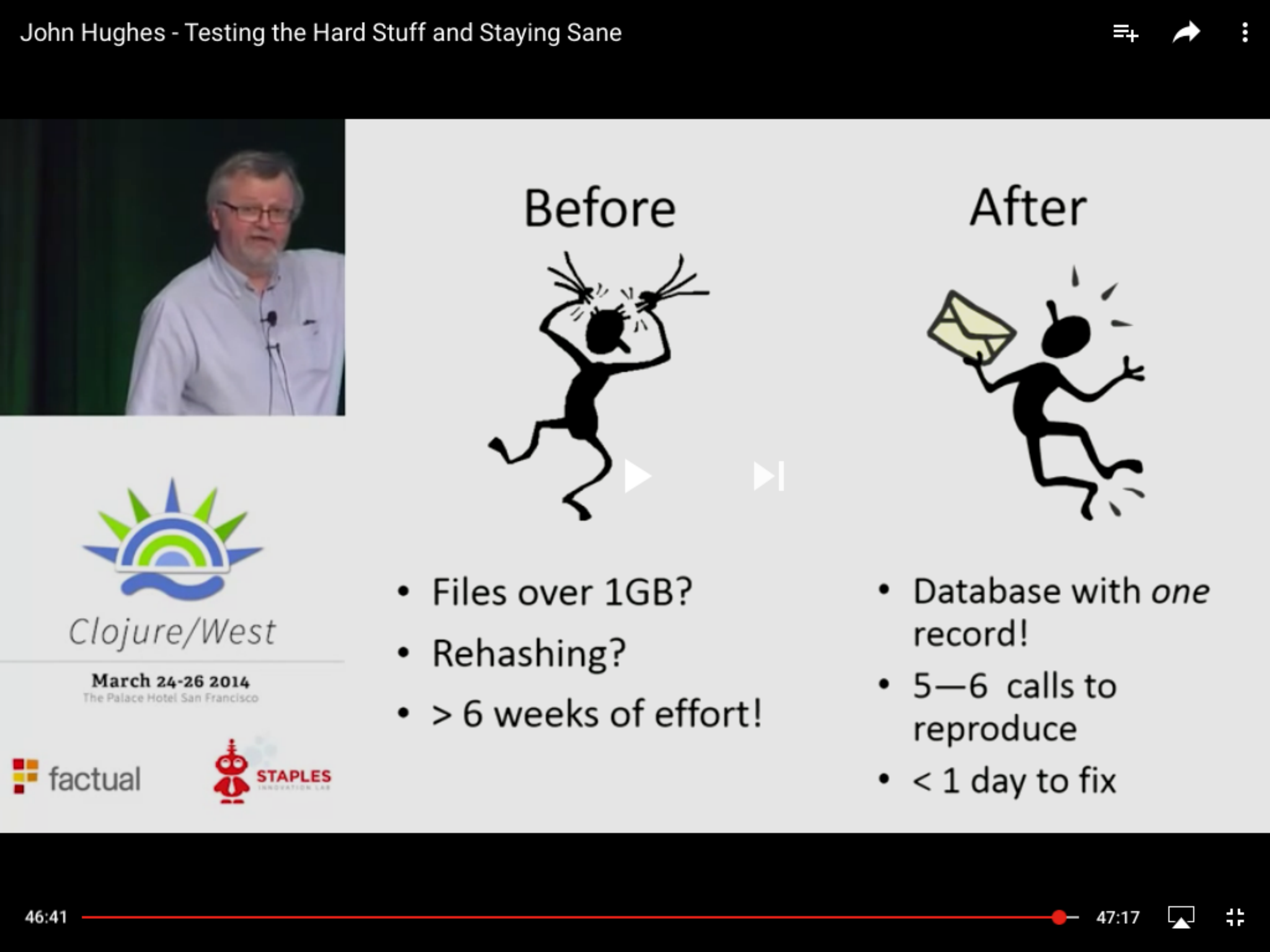
PBT’s got your Back



```
[info] ! ToUpperCase.All non lowercase characters must be at the same positions: Falsified after 43 passed tests.
[info] > ARG_0: "ȼ"
[info] > ARG_0_ORIGINAL: "램다굶ↄ甂禥●ȼ⍰齎顚릅⍰
```

REA Robot

dets


John Hughes - Testing the Hard Stuff and Staying Sane

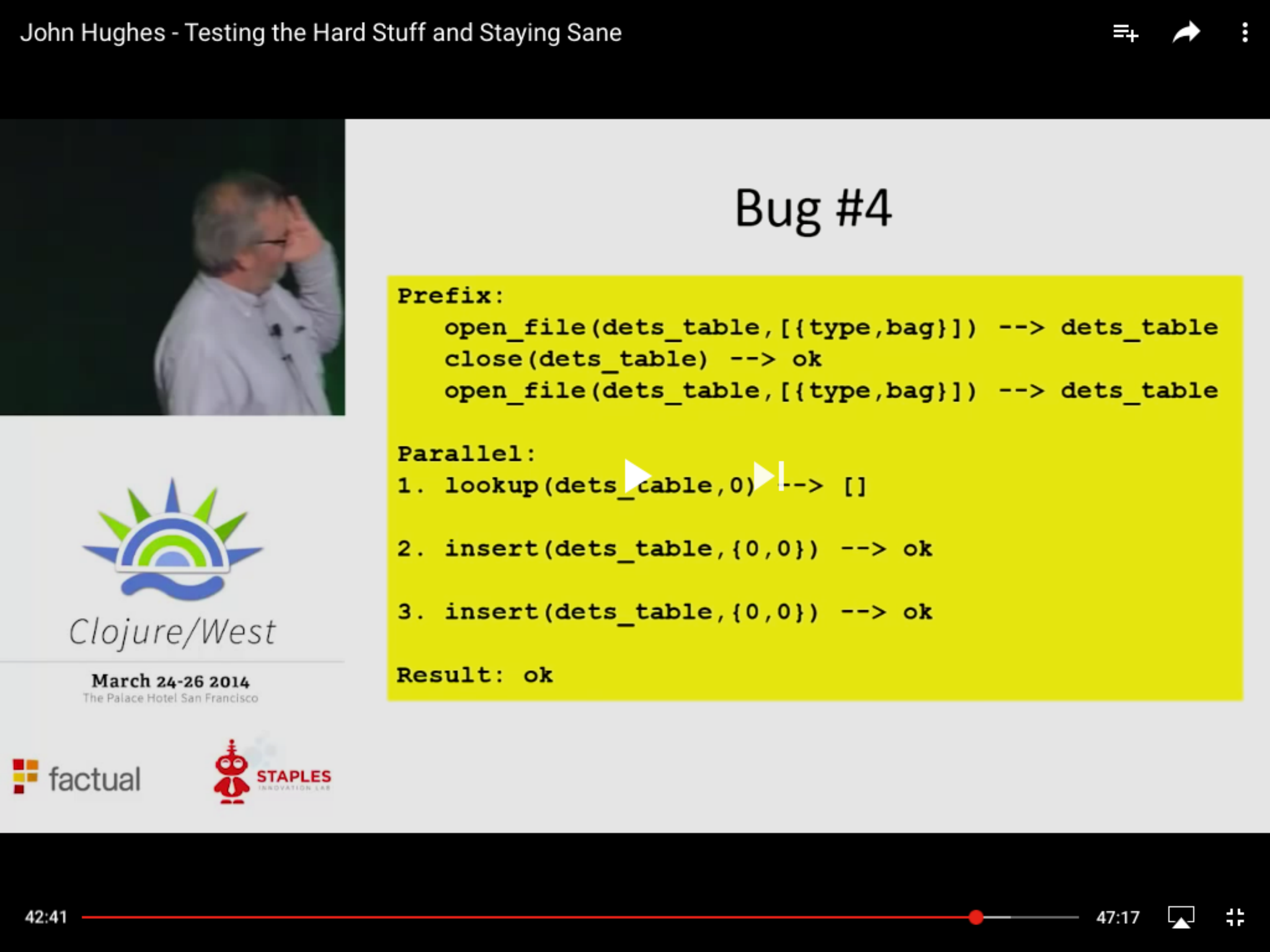
Before

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

After

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

Clojure/West
March 24-26 2014
The Palace Hotel San Francisco
factual
STAPLES
46:41
47:17


John Hughes - Testing the Hard Stuff and Staying Sane
Clojure/West
March 24-26 2014
The Palace Hotel San Francisco
factual
STAPLES
INNOVATION LAB
42:41
47:17

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
```

Summary

EBT

Easy to understand
Quick to implement
Good for implementations with few combinations
Needed for regression

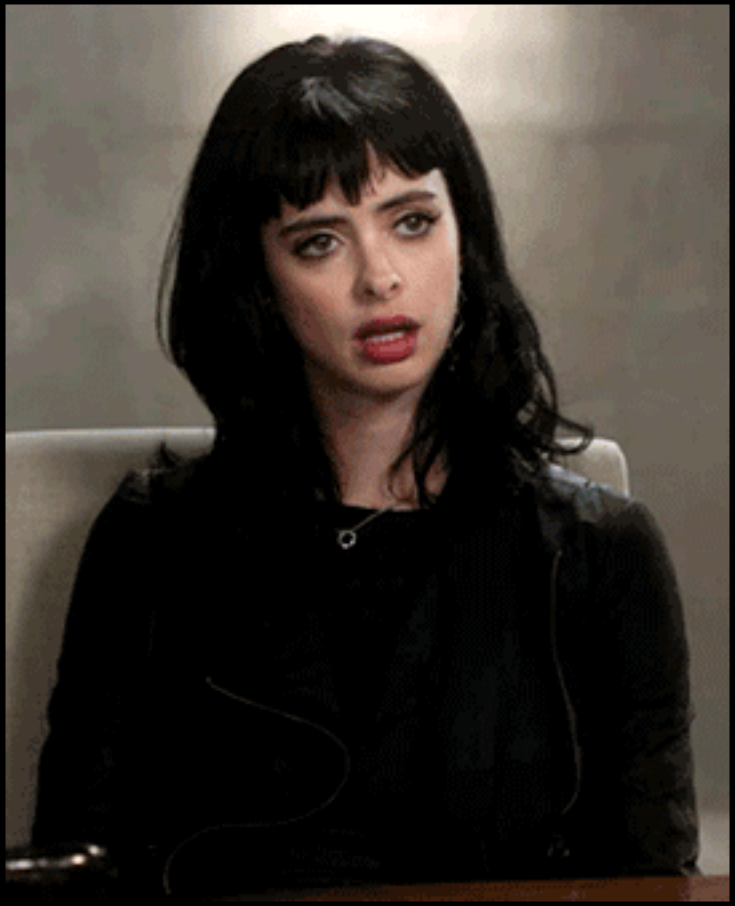
Limited by developers imagination
Hard to test complex implementations
Boring to write

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

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!