Niko Schwarz
SCG, Uni Bern

# Phexample

Good examples expand on one another

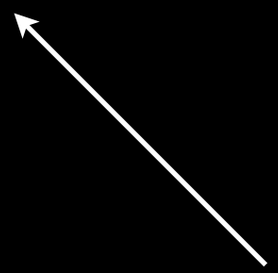
So should tests.

twitter.com/nes1983

testing framework

readable syntax

```
Stack isEmpty should be true.
```



This is real code!

JUnit:

```
assertEquals(
  stack.isEmpty(),
  true
);
```

Phexample:

```
stack isEmpty
should be
true
```

- Examples most useful when one example expands on the previous

Two examples of a stack that expand on one another:

# First example: push on a stack

Empty stack: 

# First example: push

```
StackExample>>shouldPush
    | stack |
    stack := Stack new
```

# First example: push

Push a 3:

# First example: push

```
StackExample>>shouldPush
    | stack |
    stack := Stack new.
    stack push: 3
```

# First example: push

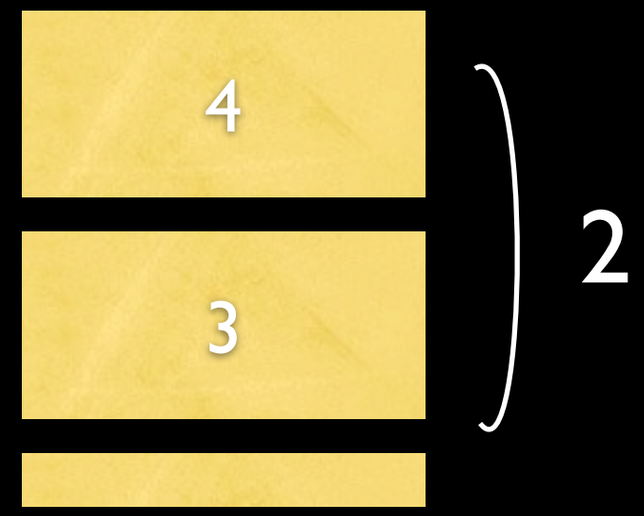
Push a 4:

# First example: push

```
StackExample>>shouldPush
    | Stack |
    stack := Stack new.
    stack push: 3; push: 4
```

# First example: push

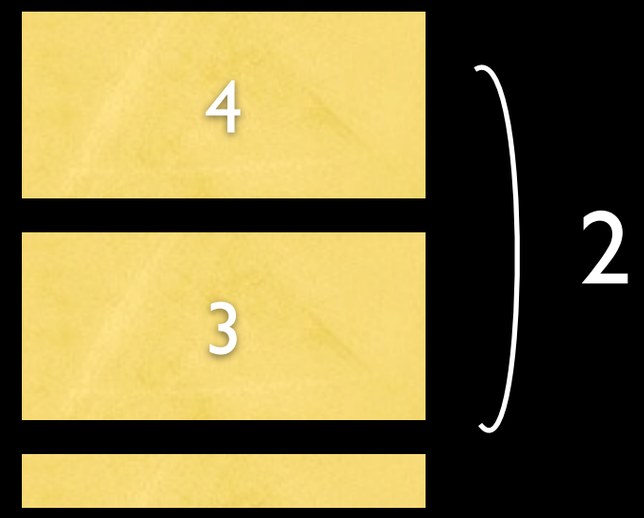
Now:




```
stack size should = 2
```

# First example: push

```
StackExample>>shouldPush
    | stack |
    stack := Stack new.
    stack push: 3; push: 4.
    stack size should = 2.
```



Make it expandable!

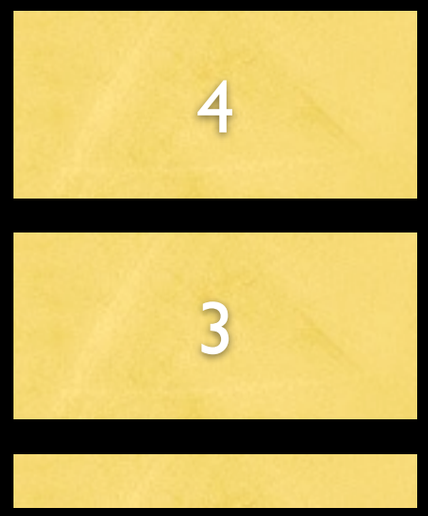
```
StackExample>>shouldPush
    | stack |
    stack := Stack new.
    stack push: 3; push: 4.
    stack size should = 2.
    ^ stack
```

# Second example: pop

Start with the previous stack:

# Second example: pop

```
StackExample>>shouldPop
    | stack |
    stack :=
      self given: #shouldPush
```



# Second example: pop

Pop the 4:

# Second example: pop

```
StackExample>>shouldPop
    | stack |
    stack :=
      self given: #shouldPush.
    stack pop.
```

# Second example: pop

Now:


```
stack size should = 1
```

# Second example: pop

```
StackExample>>shouldPop
    | stack |
    stack :=
      self given: #shouldPush.
    stack pop.
    stack size should = 1
```

# Second example: pop

```
StackExample>>shouldPop
    | stack |
    stack :=
      self given: #shouldPush.
    stack pop.
    stack size should = 1.
    ^ stack
```

# What would you do in JUnit?

push test would be about the same

but POP would differ!

```
void testPop() {
 Stack stack = new Stack();
 stack.push(3); stack.push(4);
 stack.pop();
 assertEquals(stack.size(),1);
}
```

```
void testPop() {
 Stack stack = new Stack();
 stack.push(3); stack.push(4);
 stack.pop();
 assertEquals(stack.size(),1);
}
```

Weren't we testing pop?

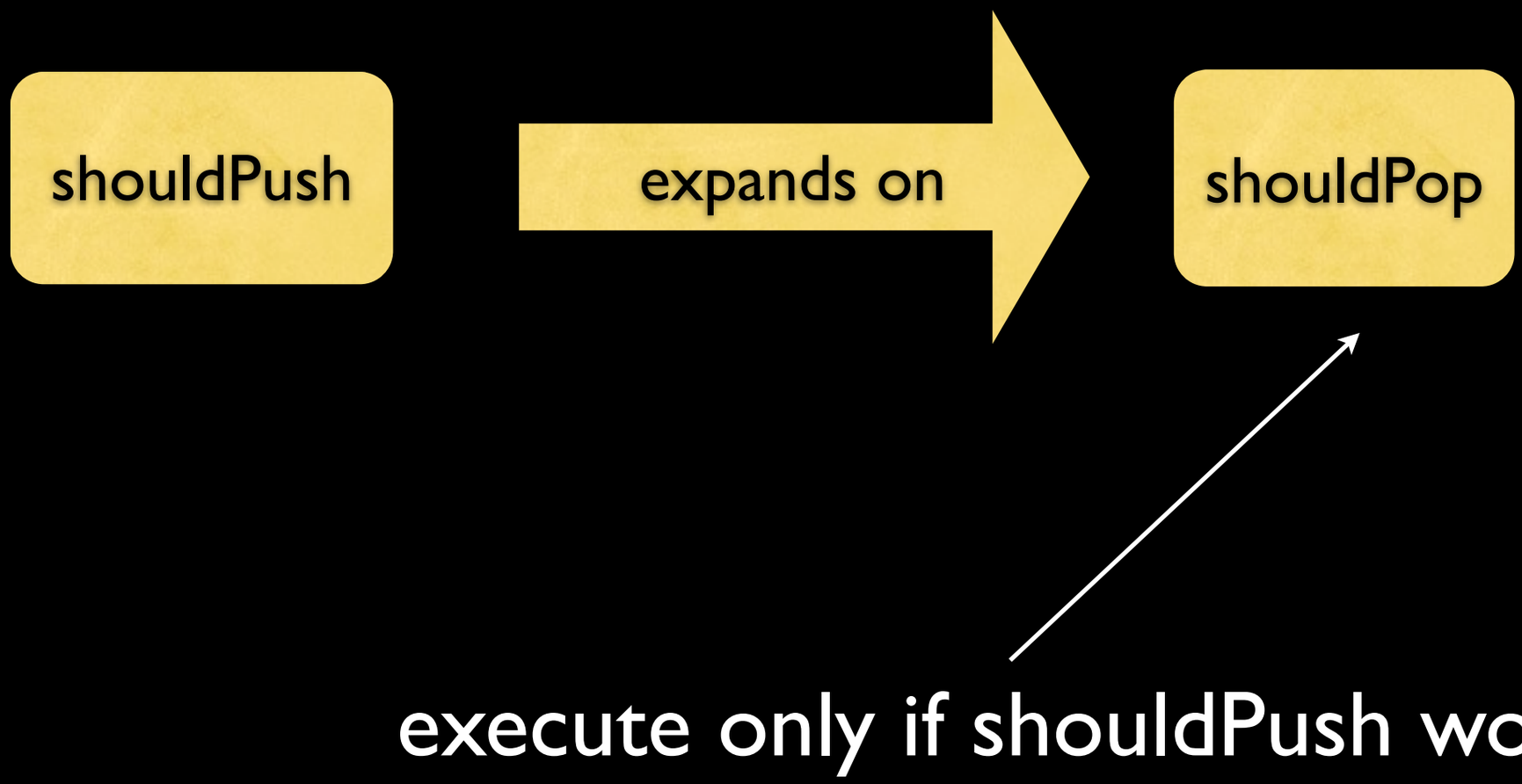
# Can only pop if there is something

Don't start all over.

Improves concreteness.

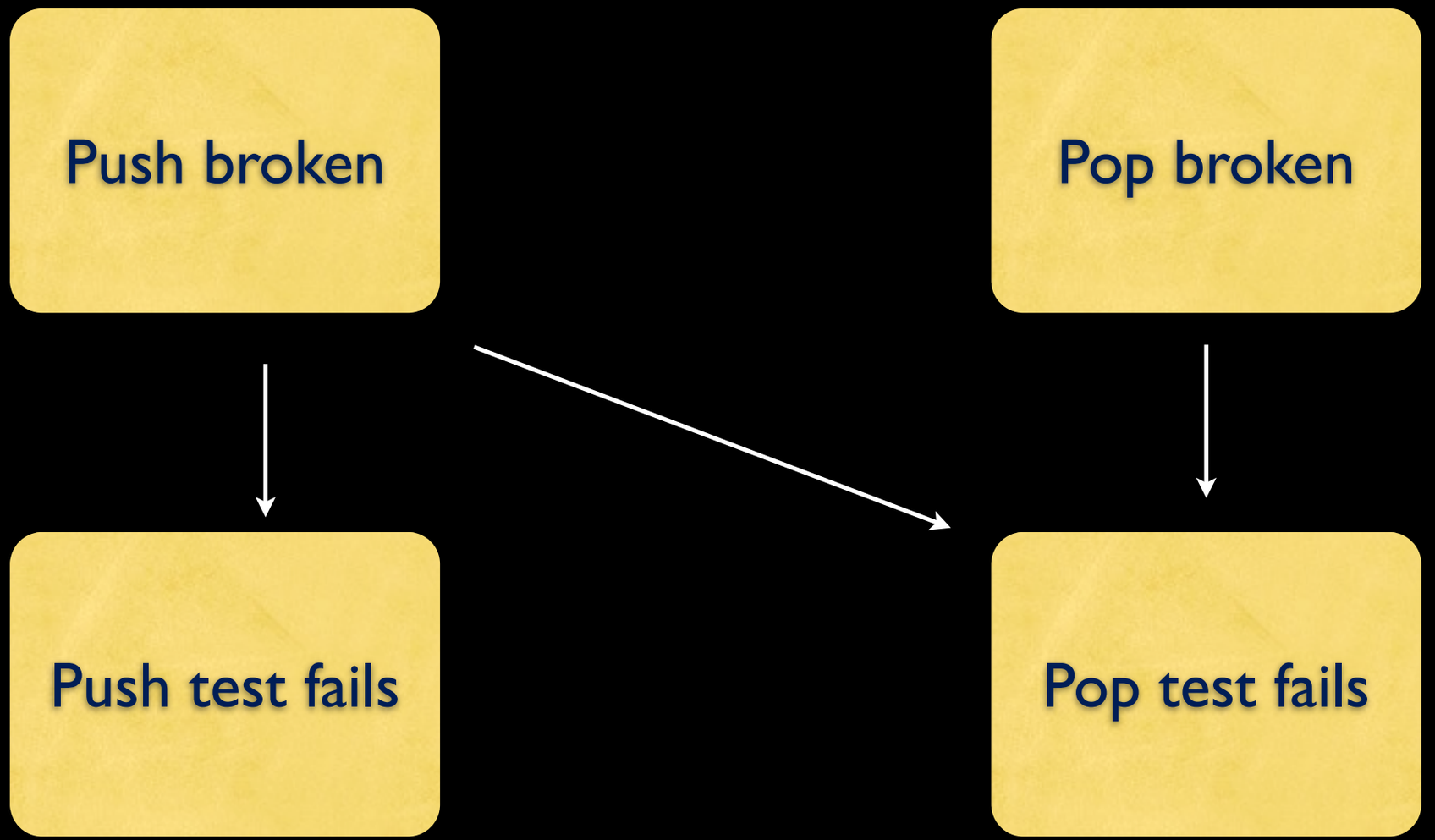
# Won brevity


shouldPush
expands on
shouldPop

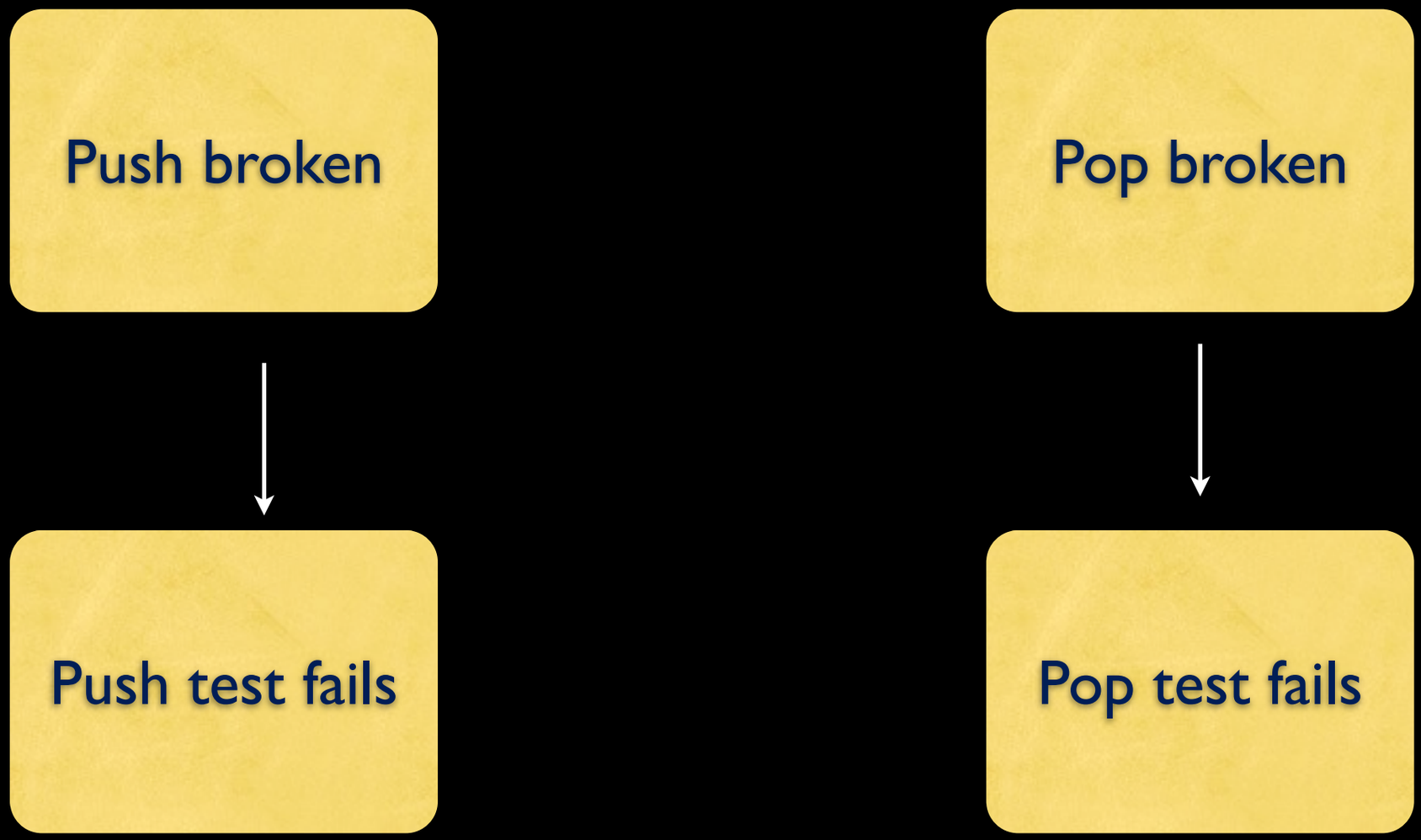

shouldPush
expands on
shouldPop
execute only if shouldPush works

Effect:

# JUnit



# Phexample



improved **locality**

# Summary

# good syntax

(almost like human language!)

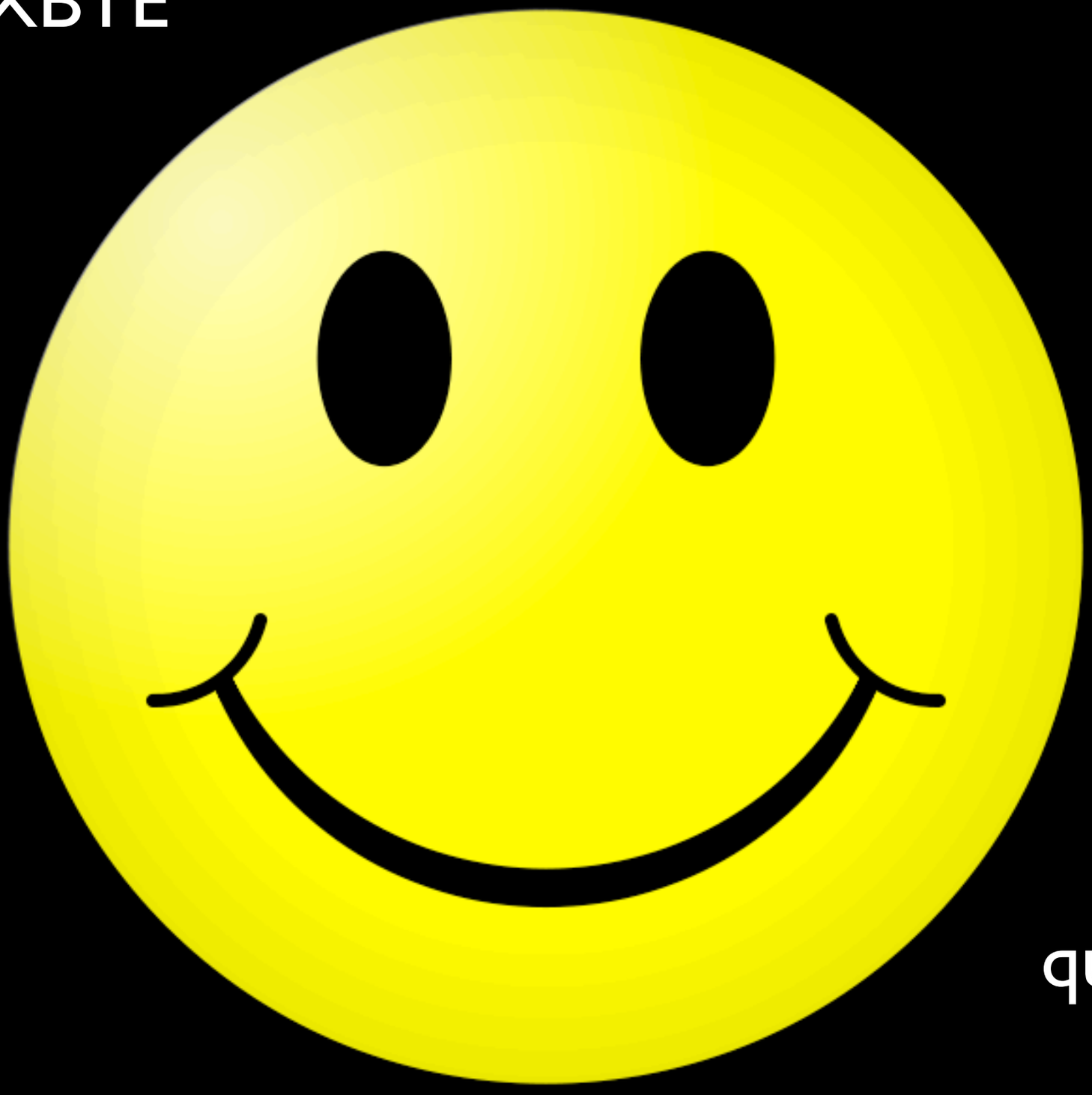
improves **brevity**

(reduce tests to the interesting part)

good locality

(One bug leads to one failing examples!)

KTHXBYE



questions?

These slides are available under the GFDL