

Alle Tests grün? Oh no!!!

Warum es manchmal gut ist, wenn ein Test rot wird.

About me

Birgit Kratz

- Freelancing IT Consultant
- Java-Backend
- More than 20 years experience
- Co-Organizer of Softwerkskammer in Düsseldorf and Köln (Cologne)
- Email: mail@birgitkratz.de
- Twitter: [@bikratz](https://twitter.com/bikratz)
- Github: <https://github.com/bkratz>
- Web: <https://www.birgitkratz.de>



Agenda

What is Mutation Testing and how does it work

Demo

Tips

First some questions

**Even with 100% code
coverage...**

**... can you tell how good and
reliable your tests are?**

“Program testing can be used to show the presence of bugs, but never to show their absence!”

— Edsger W. Dijkstra

**So how can we check
whether our tests are
good and reliable?**

Mutation Testing

How it works

How it works



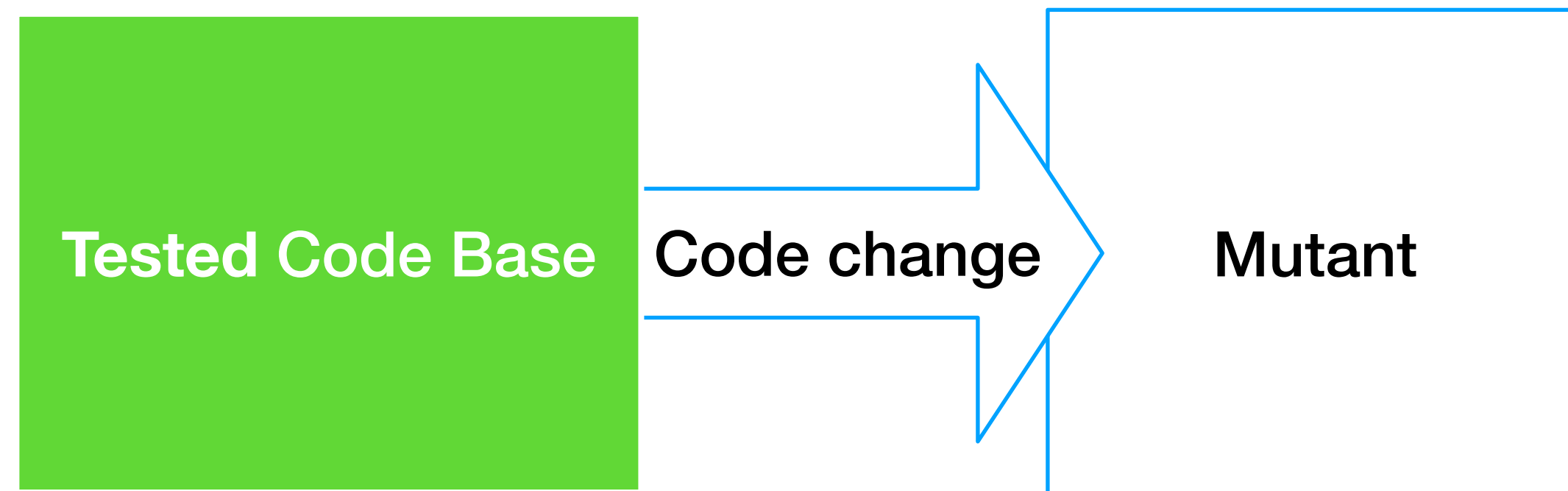
Tested Code Base

How it works

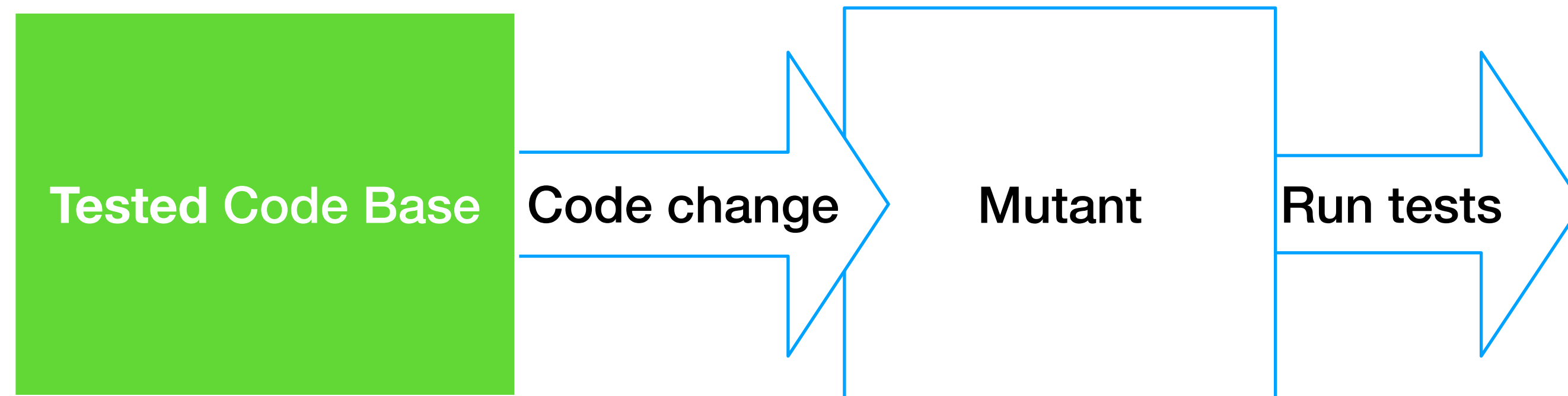


Tested Code Base

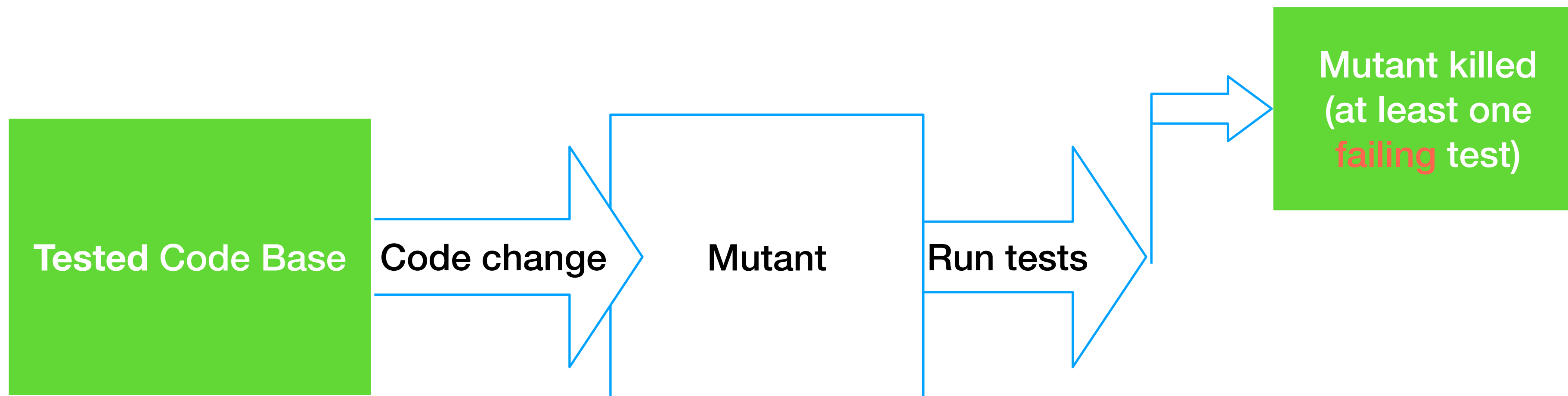
How it works



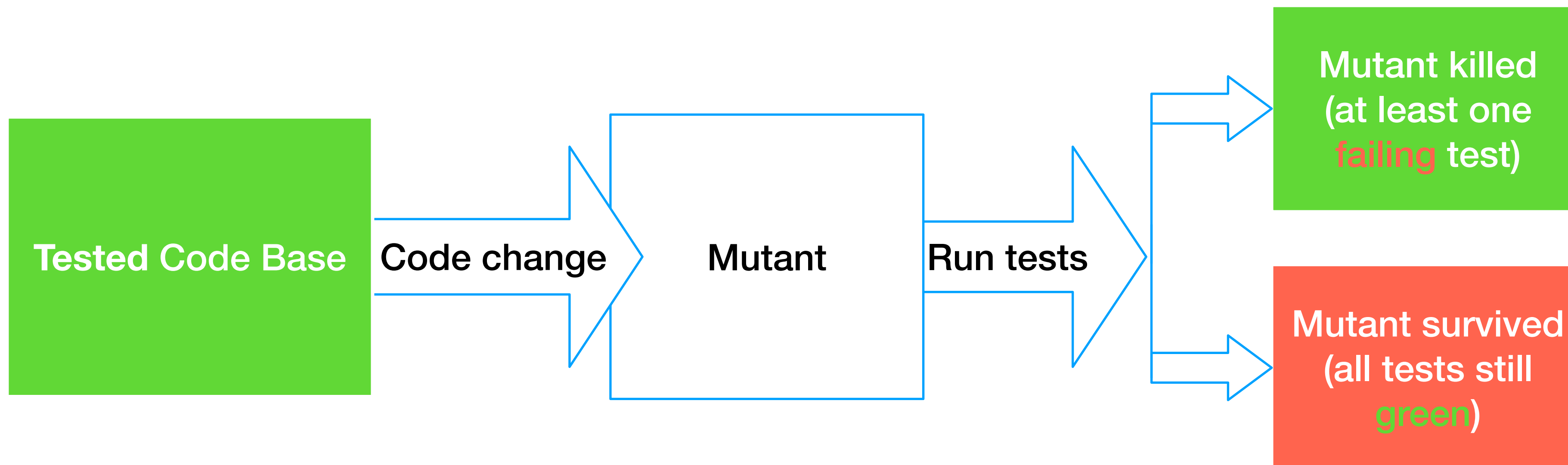
How it works



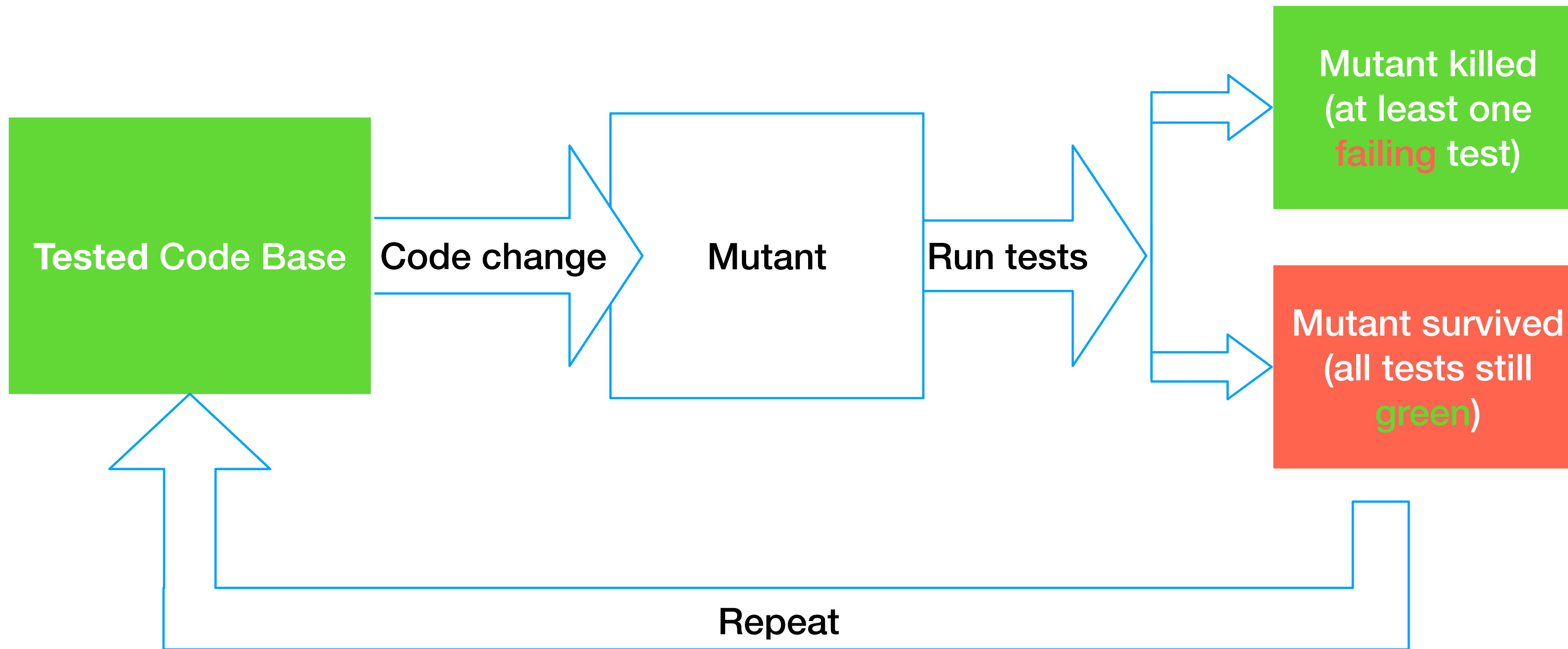
How it works



How it works



How it works



**Which kind of Mutants are
we talking about?**

Conditional Boundary Mutator

Original	Mutant
<	<=
<=	<
>	>=
>=	>

Negate Conditionals Mutator

Original	Mutant
==	!=
!=	==
>	<=
>=	<
<=	>
<	>=

Increment Mutator

Original	Mutant
<code>i++</code>	<code>i-</code>
<code>i-</code>	<code>i++</code>

Invert Negatives Mutator

inverts negation of integer and floating point numbers

Original	Mutant
<code>return -i</code>	<code>return i</code>

Math Mutator

Original	Mutant
+	-
*	/
&	
>>	<<
...	...

Many More

Void Method Call Mutator - removes calls to void methods

Empty Returns Mutator - replaces return values with an 'empty' value

False Returns Mutator - always returns false for a primitive boolean return value

True Returns Mutator - always returns true for a primitive boolean return value

Null Returns Mutator - replaces return values with null

Primitive Returns Mutator - replaces int, short, long, char, float and double return values with 0

Constructor Call Mutator - replaces constructor calls with null values

still more...

**What kind of problems can
be detected?**

**Poorly chosen or missing
test data**

Ambiguities in code base
Logical errors

Missing test coverage

**What kind of problems can
not be solved?**

Equivalent Mutation

The mutants in this set cannot be killed because they are equivalent to the original program. No possible test input exists that can distinguish their behaviour from that of the original program.

Original

```
1 int i = 2;  
2 if ( i >= 1 ) {  
3     return "foo";  
4 }
```

Mutant

```
1 int i = 2;  
2 if ( i > 1 ) {  
3     return "foo";  
4 }
```

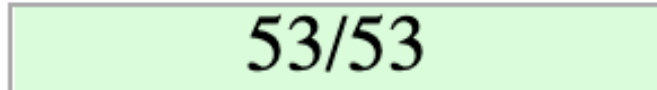
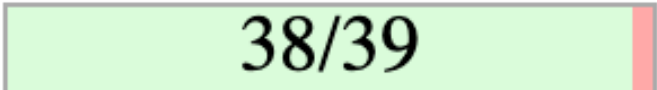
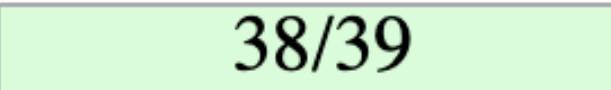
Stubborn Mutation

The mutants in this set can be killed. Each stubborn mutant does have a test input that distinguishes its behaviour from that of the original program. However, none of these distinguishing test inputs has yet been found.

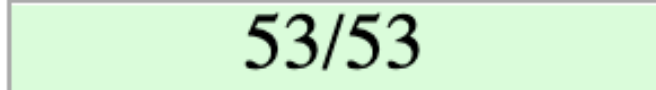
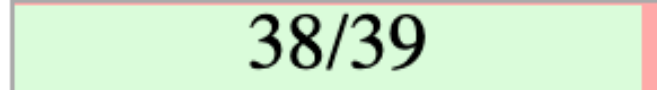
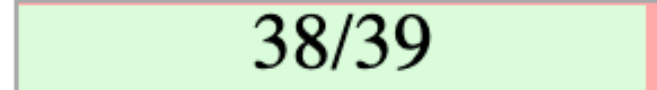
Report Example

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
1	100% 	97% 	97% 

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
de.birgitkratz.aoc2021	1	100% 	97% 	97% 

DEMO with Java and PIT

(<https://github.com/hcoles/pitest>)

Disadvantages of Mutation testing

- Can be **very** time consuming
- Cannot detect/avoid equivalent mutations, since the resulting mutant behaves in exactly the same way as the original
- Not usable for Black Box Testing

Cost of Mutation Testing

Let's assume we have:

- a code base with 300 Java classes
- 10 test cases for each class
- on average, each test case requires 0.2 seconds for its execution
- the total test suite execution costs $300 * 10 * 0,2 = \mathbf{600 \text{ seconds}}$ (10 minutes)

Let's assume we have, on average, 20 mutants per each class.

The total cost of mutation analysis is $300 * 10 * 0,2 * 20 = \mathbf{12000 \text{ seconds}}$ (3h 20 min)

How to reduce this cost?

**Reduce number of used
Mutations**

**Reduce number of Classes to
apply Mutation Testing**

Incremental Analysis

Extreme Mutation Strategy

Article: Will My Tests Tell Me If I Break This Code?

<https://arxiv.org/pdf/1611.07163.pdf>

Implementierung für PIT: [pit-descartes](#)

Mutation Test Tools

<https://github.com/theofidry/awesome-mutation-testing>

Try it!

- ✓ Try it again
- ✓ Start small
- ✓ Write more tests
- ✓ Get familiar with reported issues and how to solve them
- ✓ Configure it to your needs
- ✓ Start with critical components
- ✓ Don't use all Mutators all the time
- ✓ Integrate into CI pipeline
- ✓ Don't strive for 100%

Questions?

Thank you

Sample code:

<https://github.com/bkratz/AdventOfCode-2021/tree/main/day03-java>

<https://github.com/bkratz/MutationTesting-aoc2021-d03-akaritakai>

<https://github.com/bkratz/MutationTesting-aoc2021-d03-jerchende>

- Email: mail@birgitkratz.de
- Twitter: @bikratz
- Github: <https://github.com/bkratz>