



Testing the tests with Mutalk in 2024

guillermo.polito@inria.fr

Guillermo Polito - ESUG'24



Quick About Me

guillermo.polito@inria.fr
@guillep



- **Now:** Researcher at Inria - Lille
- Pharo Contributor since ~2010
- **Keywords:** compilers, testing, test generation
- **Interests:** tooling, benchmarking, 日本語, board games, batman, concurrency

If any of that interests you, come talk to me!



Automated Tests

SetTest >> testAdd

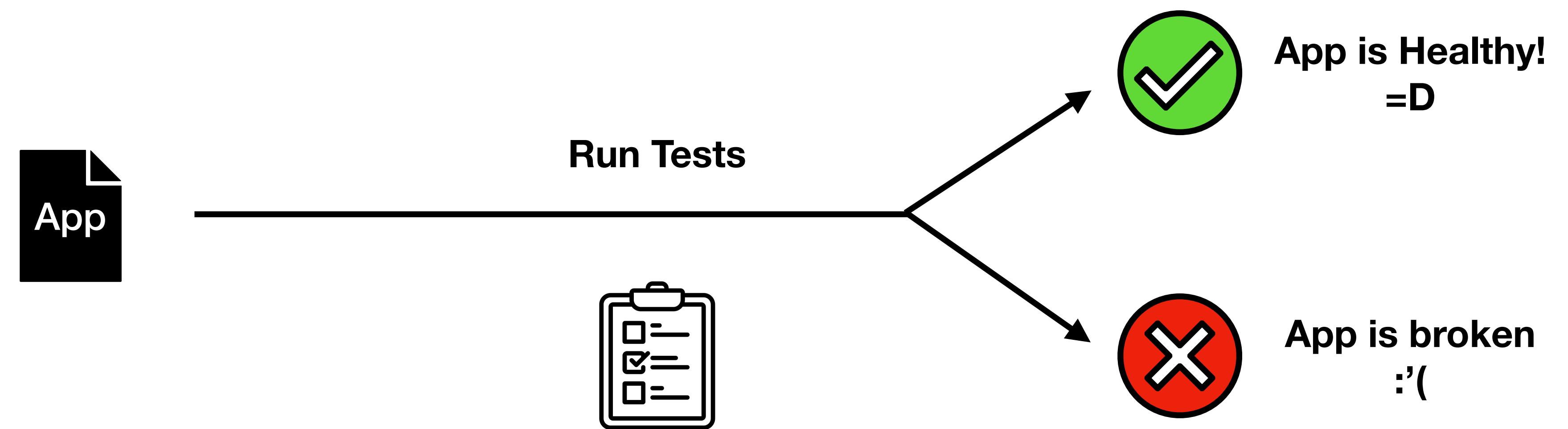
```
| aSet |  
"Context"  
aSet := Set new.
```

```
"Stimuli"  
aSet add: 5.  
aSet add: 5.
```

```
"Check"  
self assert: aSet size equals: 1.
```

*in this context
when this happens
then this should happen*

We love tests



What is a good test?

“A good test is a test that catches bugs”

- me

**WHO'S
WATCHING
THE
TESTS**

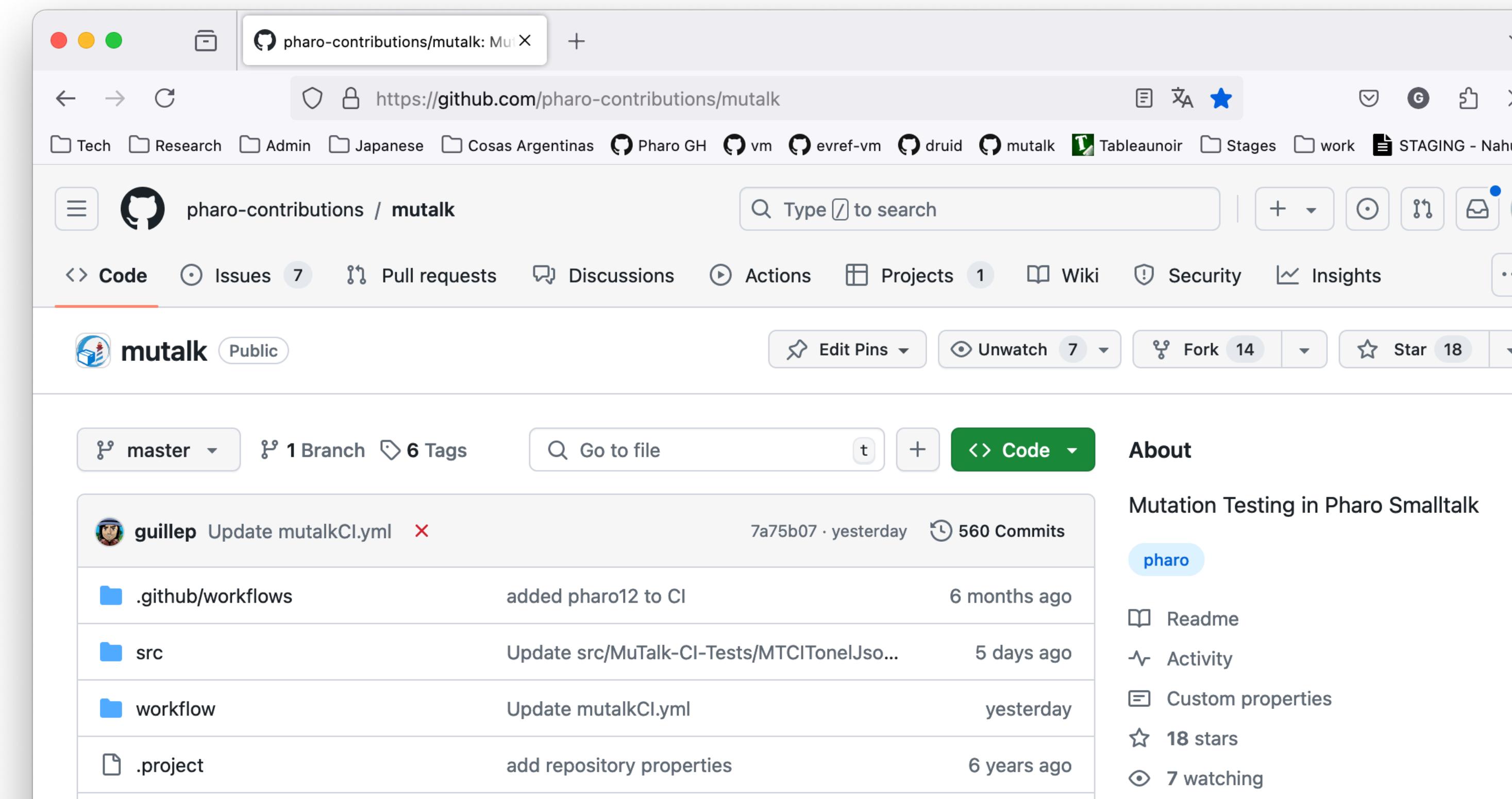
The poster has a yellow background. On the left side, there is large black text that reads "WHO'S" on the first line, "WATCH" on the second line, and "THE" on the third line. On the right side, there is also large black text that reads "TESTS" on the first line and "THE" on the second line. A thick black diagonal line runs from the top-left towards the bottom-right. There are red paint splatters at the top center, bottom center, and bottom right corner.



Mutalk: Mutation testing for Pharo

- Originally developed in Pharo 1.1 in Argentina (Chillo, Brunstein, Wilkinson)
- Presented at ESUG'09

- Pharo 9 to 12 !



Coverage vs Mutations



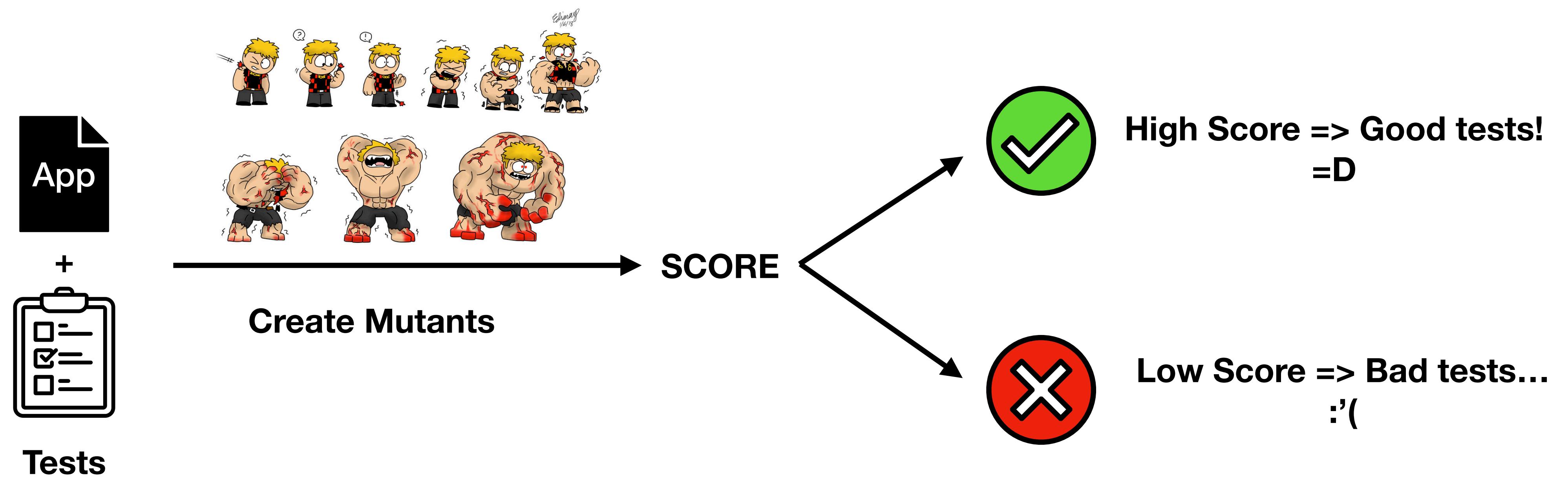
30%
Code
Coverage



4%
Mutation
Coverage

A screenshot of a GitHub repository named "Artefact". The repository has 30% code coverage and 4% mutation coverage. The repository page shows a list of files and their descriptions:

Mutation testing in a nutshell







These kind of mutant

- Introduce artificial bugs
- See if the test suite detects them
- What kind of bugs?



Competent developer hypothesis

- Developers are **capable people**
- Mistakes/bugs are **small details** easily overseen. E.g.,
 - Missing +/ - 1 in a loop
 - An inverted conditional
 - Signed/unsigned

Thousands of simple mutations

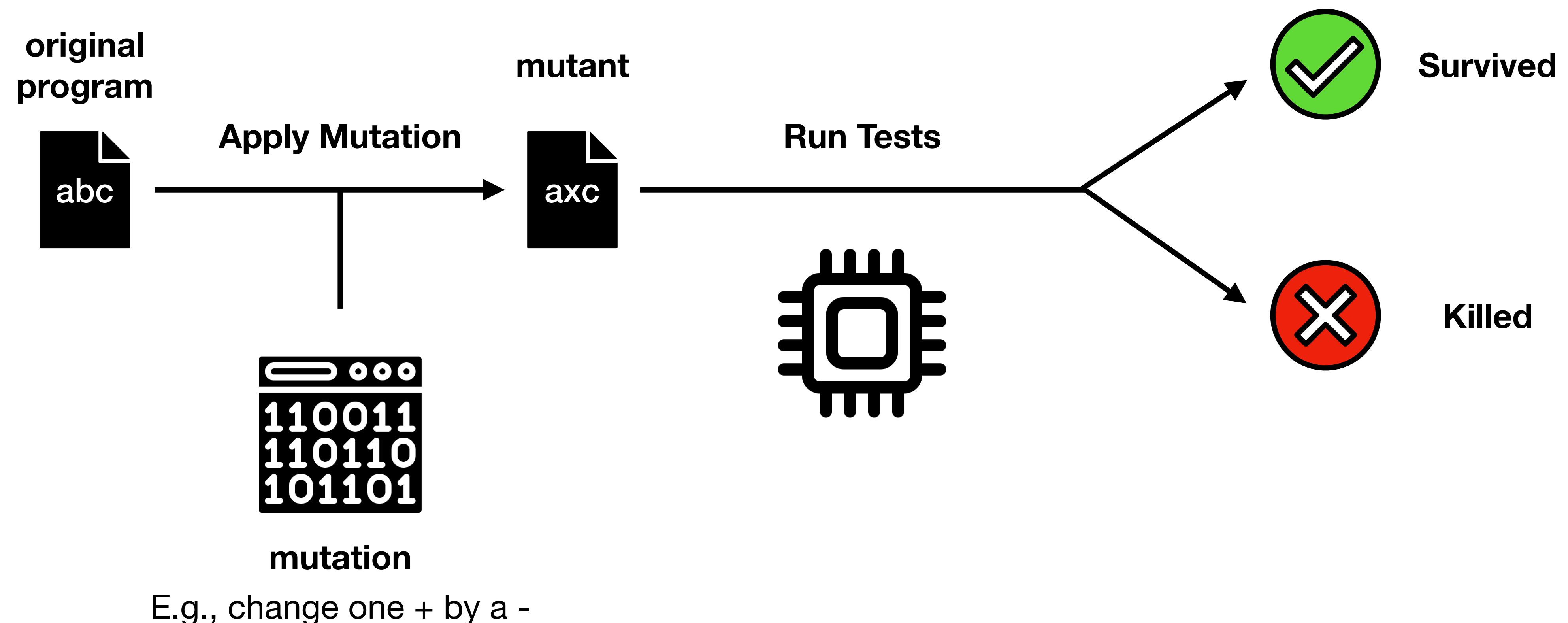
- Control flow bugs
- Arithmetic bugs
- Logic bugs
- Overflow bugs
- Typoses

The screenshot shows a mutation inspection tool window titled "Inspector on 3090 mutants, 1599 killed, 1491 alive...". The top navigation bar includes tabs for "Surviving Mutants" (which is selected), "Killed Mutants", "Terminated Mutants", "Excluded Tests", "Raw", "Breakpoints", and "Meta". Below the tabs, a list of 8 mutation results is displayed:

- 1 Replace #ifTrue: receiver with false in MicAnchorBlock>>#printOn:
- 2 Replace detect: block with [:each | true] when #detect:ifNone: in Class>>#documentExampleCode
- 3 Replace #ifFalse: receiver with false in MicInlineDelimiter class>>#initializeDelimiters
- 4 Replace #ifTrue:ifFalse: receiver with false in MicMicrodownTextualBuilder>>#mathblock:firstLineAssociations:withCaption:
- 5 Decrease a literal integer in Package>>#buildMicroDownUsing:withComment:
- 6 Replace #ifTrue: with #ifFalse: in BaselineOf class>>#buildMicroDownUsing:withComment:
- 7 Replace #= with #'~= in MicInlineDelimiter class>>#initializeDelimiters
- 8 Replace #or: with #and: in MicFileResourceReference>>#loadChildren

Below the list, two code snippets are shown side-by-side, illustrating the mutation. The left snippet shows the original code with a highlighted line: "each key = #caption". The right snippet shows the mutated code where the line has been changed to "false". Both snippets include context from surrounding code, such as "mathblock: aString firstLineAssociations: aCol withCaption: a" and "self raw: MathOpeningBlockMarkup. aCol".

Mutation Analysis



The insight

- Survived mutants were either

1. not covered

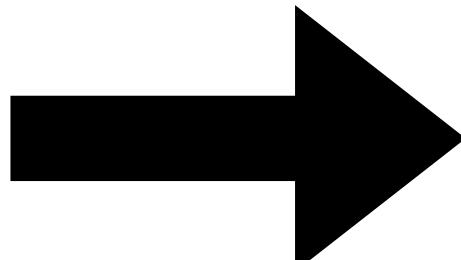


improve your tests!!

2. not asserted

3. or semantically equivalent.

E.g. $A+B=A-B$ if $B=0$ always



bias our results

Mutation Score

- Run each mutation independently
- Score:

$$\frac{\text{\#Killed}}{\text{\#Mutants}}$$

THE Problem of Mutation Analysis

Runtime = Time(tests) * #Mutants

Optimizing of Mutation Analysis

Runtime = Time(tests) * #Mutants

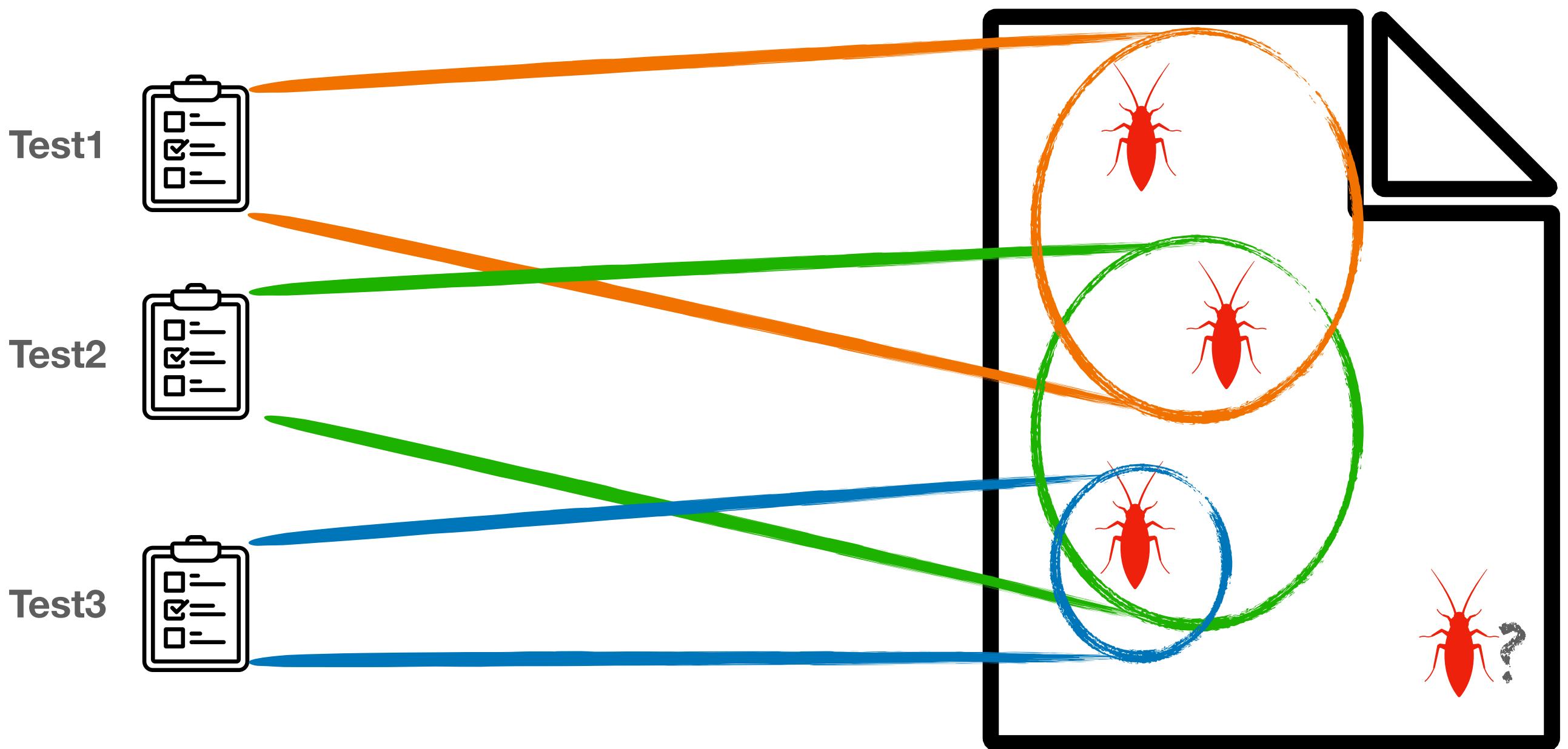
Can we run less tests?

Can we run less mutants?

Ideally... no impact on score!

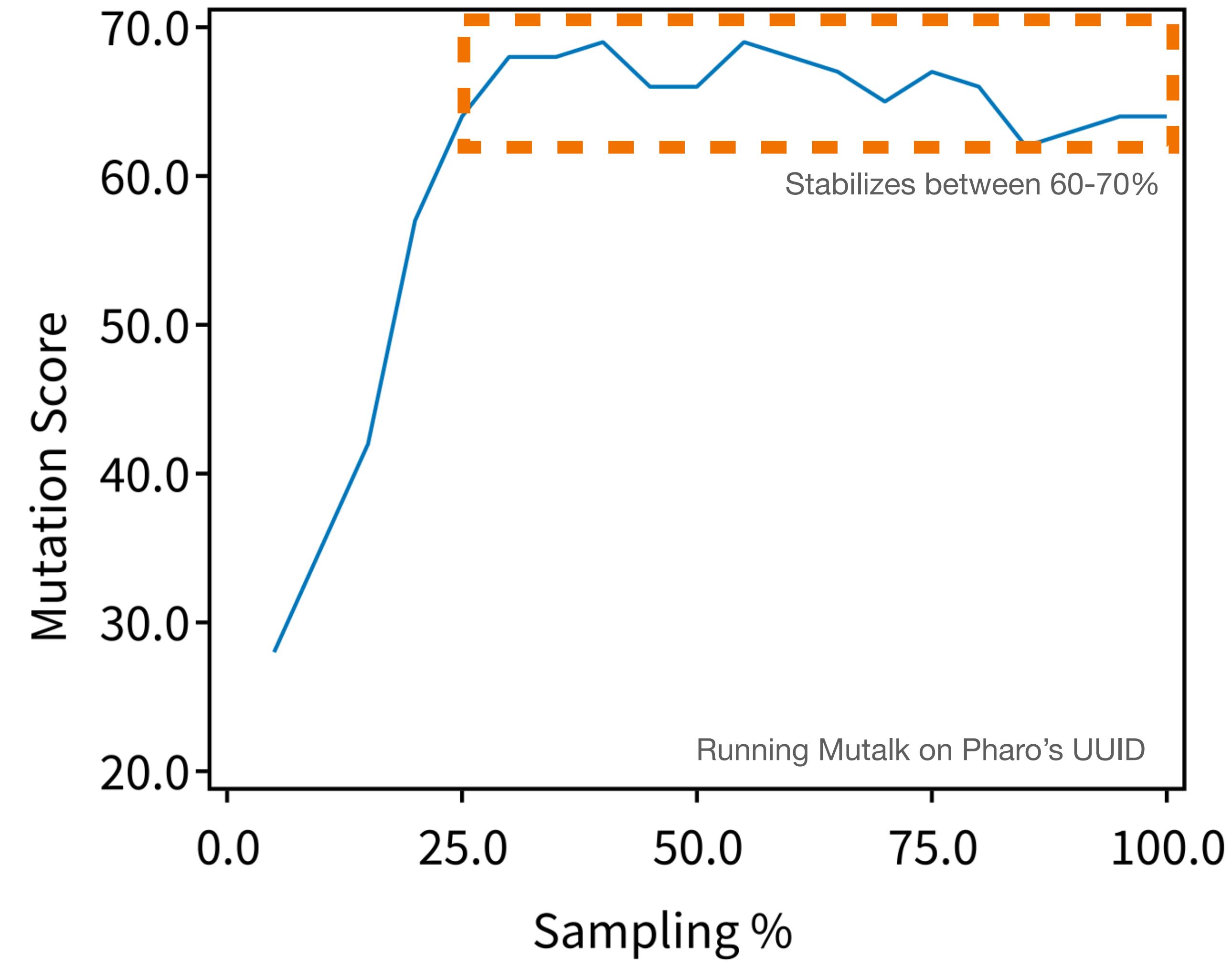
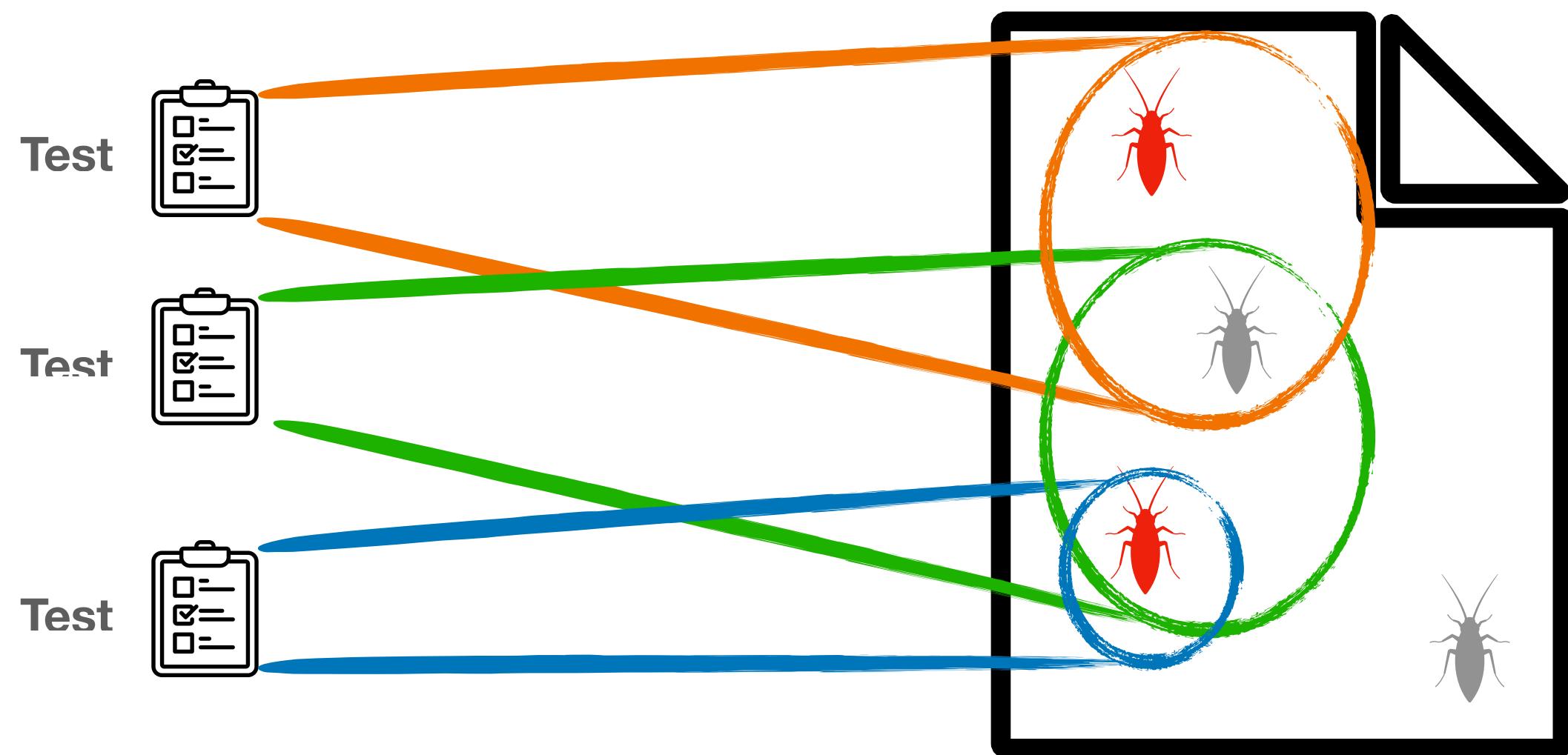
Selecting Mutants and Tests from Coverage

- Run *only* tests covering mutants
 - Only mutate covered code
-
- What if lots of mutants remain?



Random Sampling of Mutants

- Lots on research on it
- Aggressively reduces run time

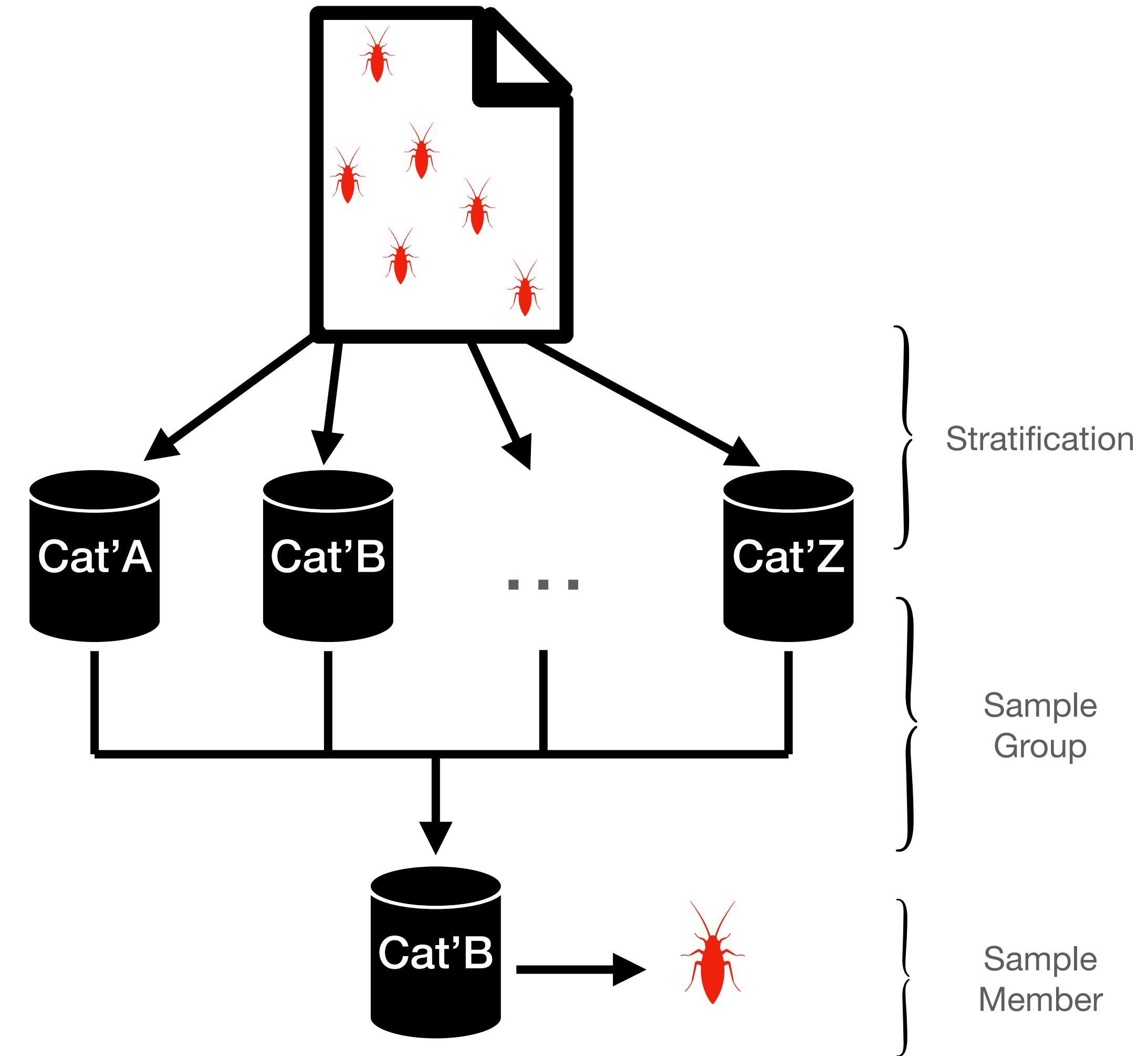


Guiding Random Sampling of Mutants

- Stratified sampling with different strategies
 - per class, method, ...

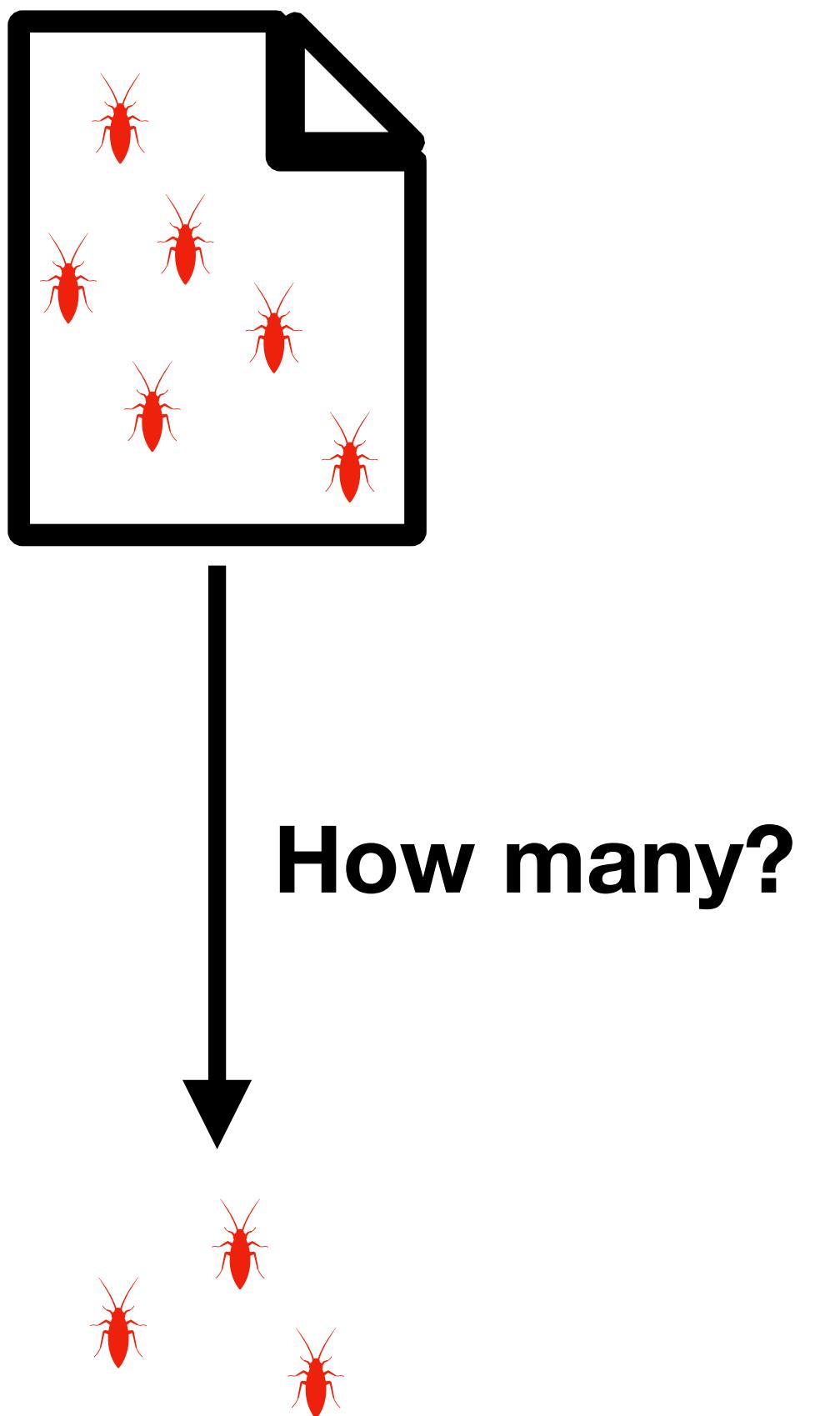
1. Taking a sample:

1. Take a group at random
2. Take a member from the group



When to Stop Sampling: Budgets!

- Fixed budgets: # or % of mutants
 - Too many: No win in runtime!
 - Too few: Unrealistic mutation score...
- We need a practical solution:
 - **Time budget!**



Github actions integration

- Github actions
- + Microdown
- Configure budgets, different run modes...

```
name: MutalkCI

on:
  pull_request_target:
    branches: [ main ]
  paths-ignore:
    - 'README.md'
    - 'resources/**'

jobs:
  mutation_testing:
    uses: pharo-contributions/mutalk/.github/workflows/mutalk-workflow.yml@v2.5.0
```

github-actions bot commented 3 days ago

Mutation results

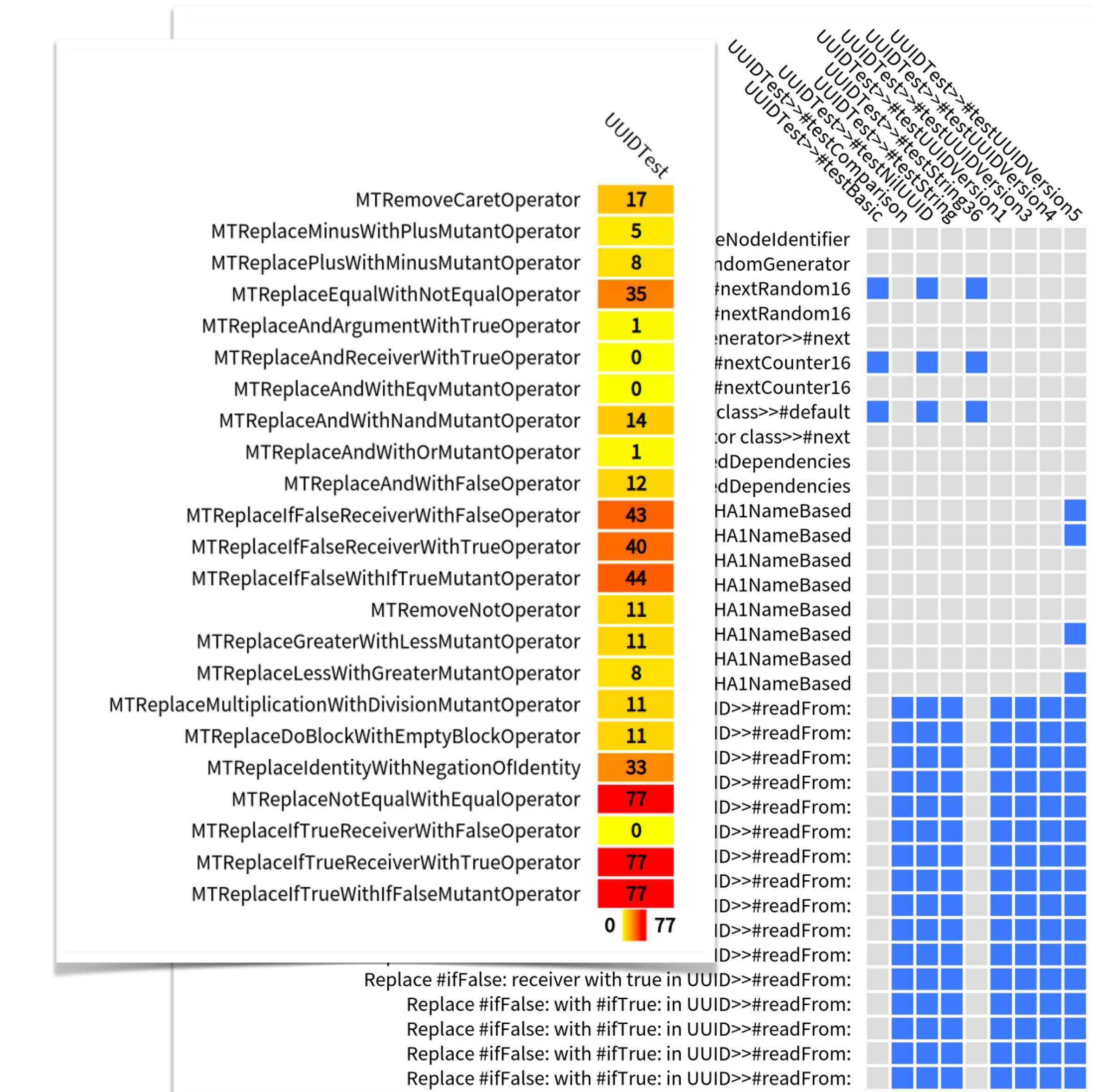
- Commit: [82aace5](#)

Mutants	Number of Mutants
Alive	161
Killed	834
Terminated	0

Mutation score: 83

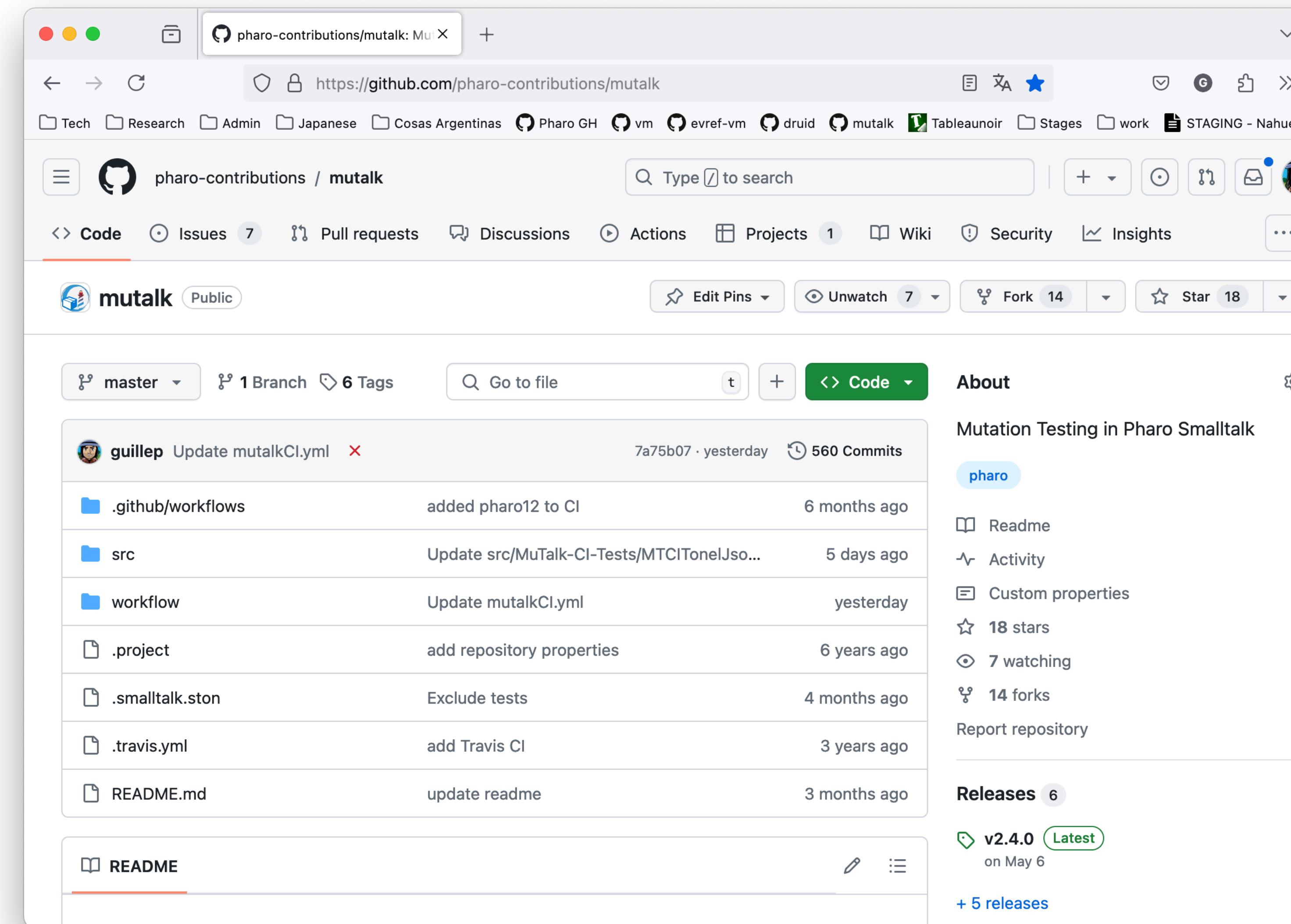
Mutation Matrix and Heatmap

- **Fine-tune for your project**
- **Advanced mutation analysis**
- Identify
 - **Trivial mutations:** easy to kill
 - **Redundant tests and mutants**
 - **Randomization strategy**



Talk to me

- If you want to **use it!**
- If you want to **contribute!**
- You need **specific operators**



Mutalk 2024

- Working from **Pharo9** to **Pharo12 (13?)**
- **Practical:** Budgets, random selection and CI
- **New** features: test filters, logging, new operators
- **Fixes, improvements, cleanups**
- Thanks to all people contributing over the years!

Coverage vs Mutation

- Average has 100% coverage
- What if we mistake the / by a *?

```
CollectionTest >> testAverage  
    self assert: #(1 2 3) average
```

```
Collection >> average  
    ^ self sum / self size
```