Optimising CI using Test Case Prioritisation

June 19, 2020

Pieter De Clercq

Promotors: prof. dr. Volckaert, prof. dr. ir. De Turck | Supervisors: Jasper Vaneessen, Dwight Kerkhove

Overview

1. Problem

Overview

1. Problem
2. Solutions

Overview

1. Problem
2. Solutions
3. Implementation

Overview

1. Problem
2. Solutions
3. Implementation
4. Results

Overview

1. Problem
2. Solutions
3. Implementation
4. Results

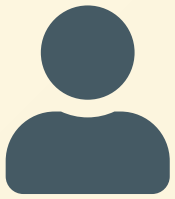
But first

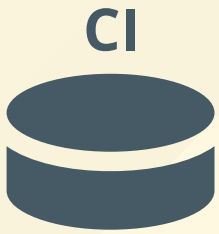
Just what is CI?

Continuous Integration

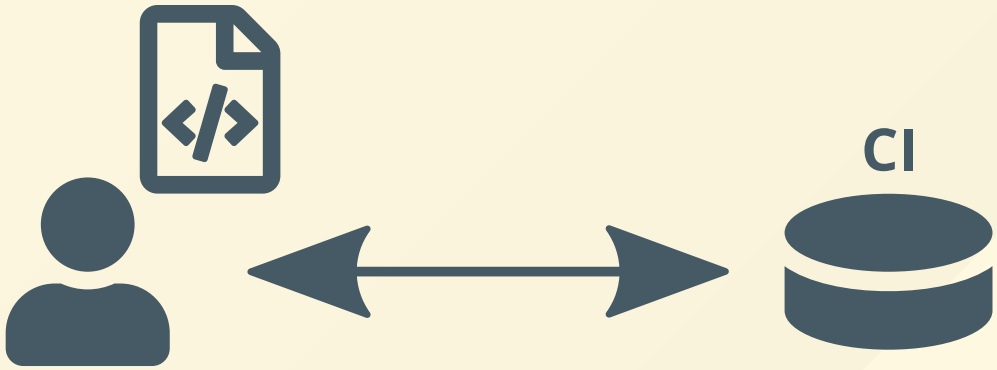
Example: Android app

Continuous Integration

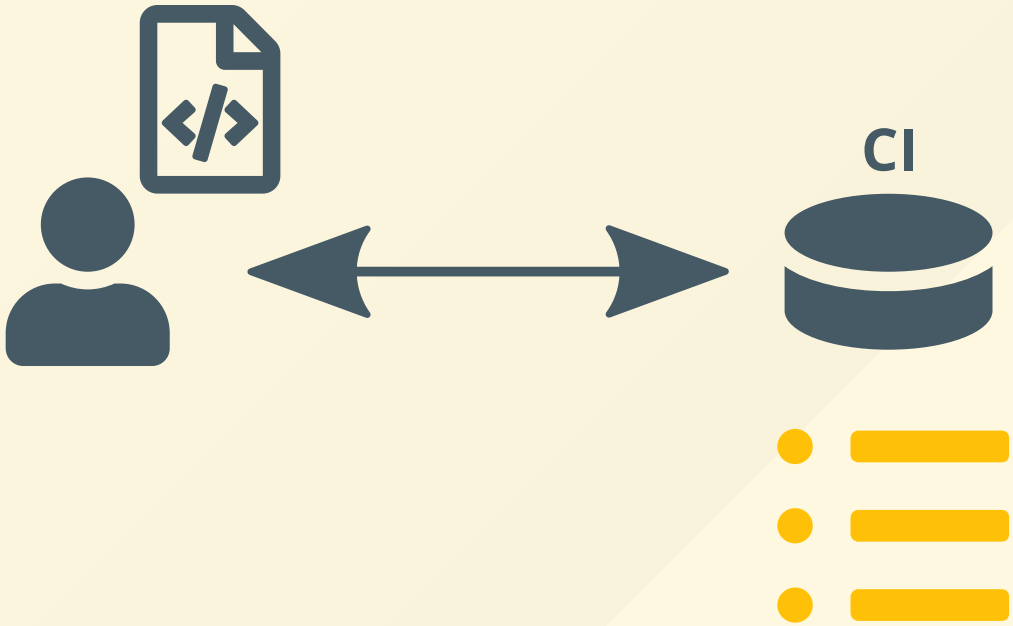




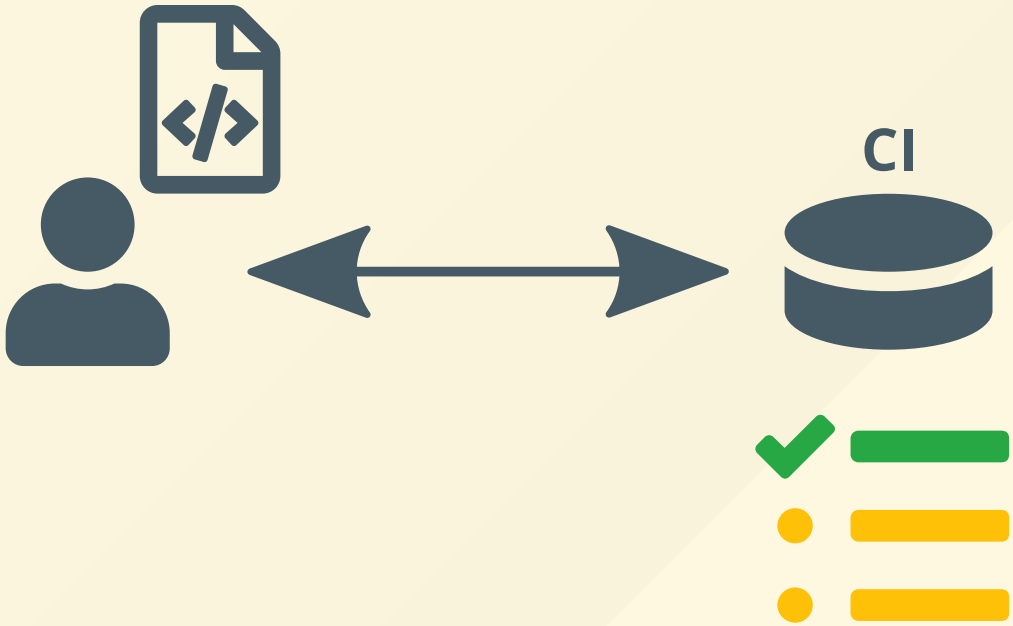
Continuous Integration



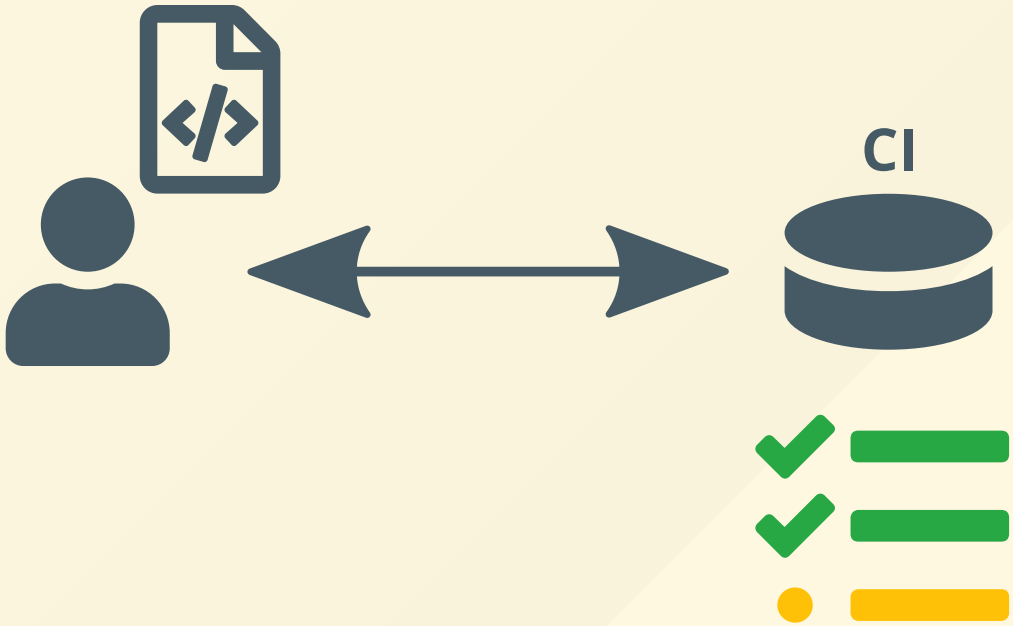
Continuous Integration



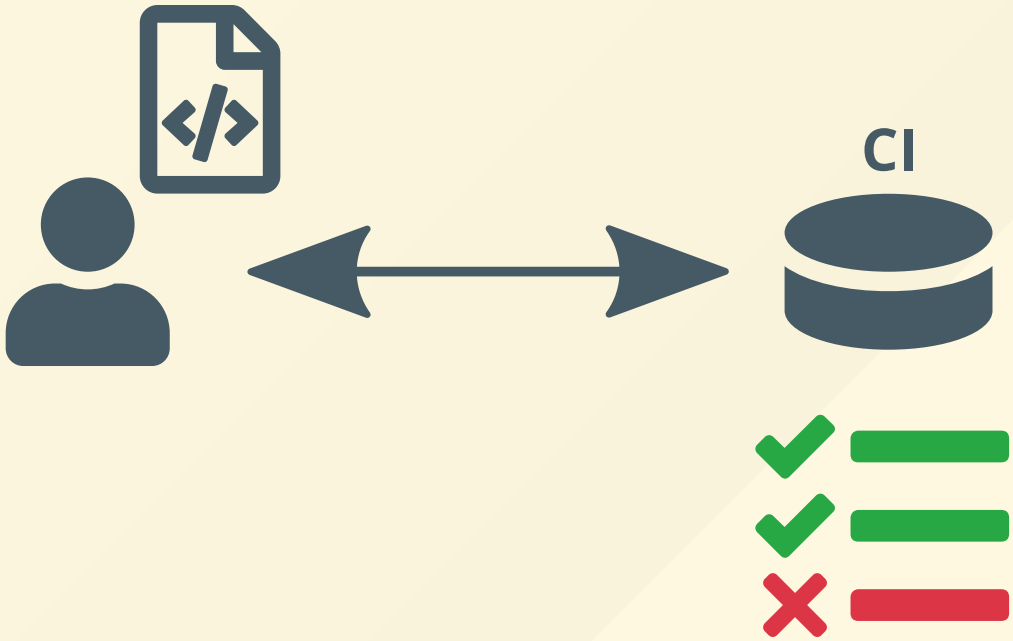
Continuous Integration



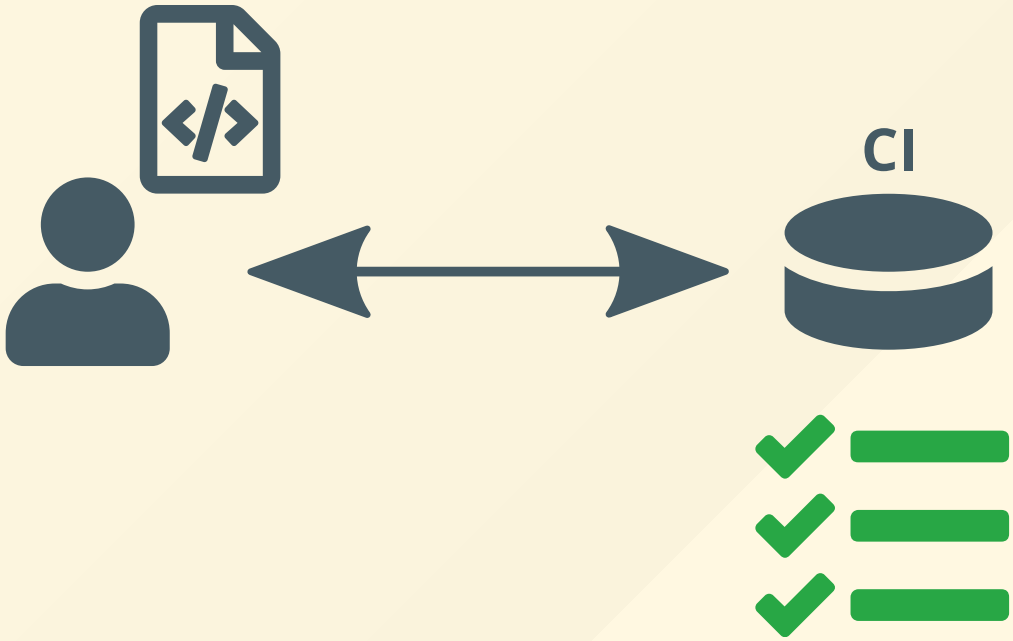
Continuous Integration



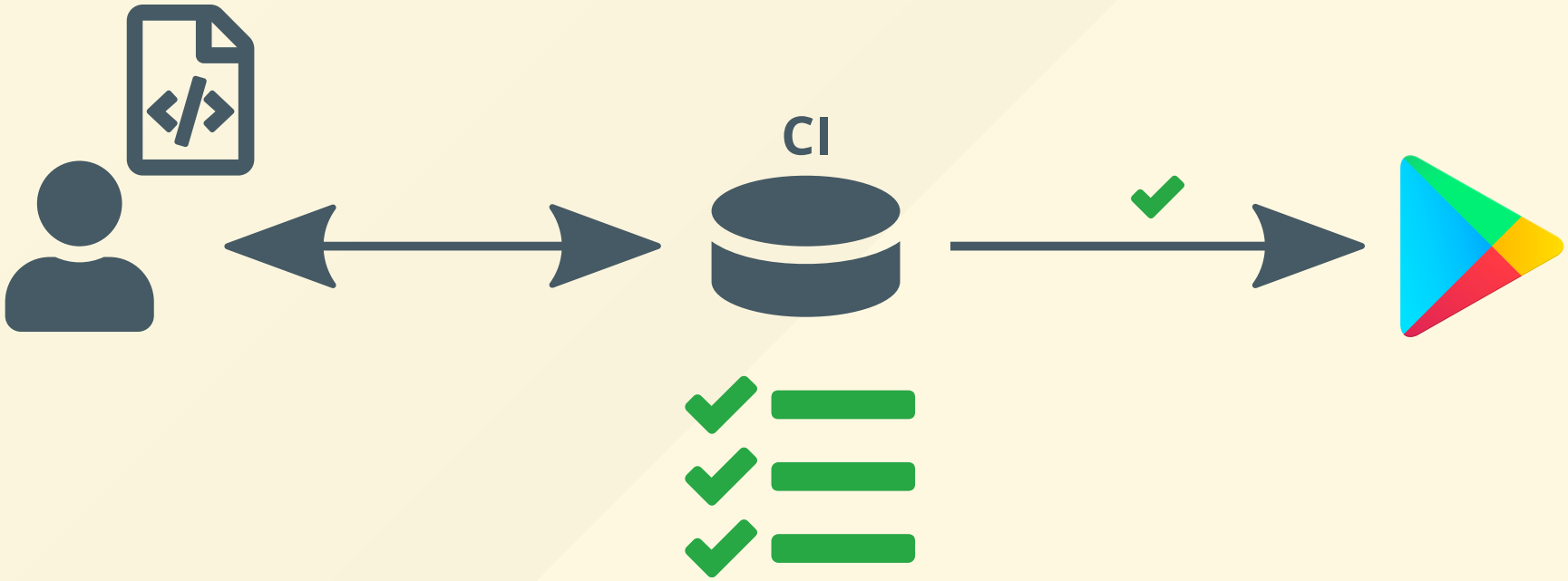
Continuous Integration



Continuous Integration



Continuous Integration



Problem?

Tests!

Tests

Tests

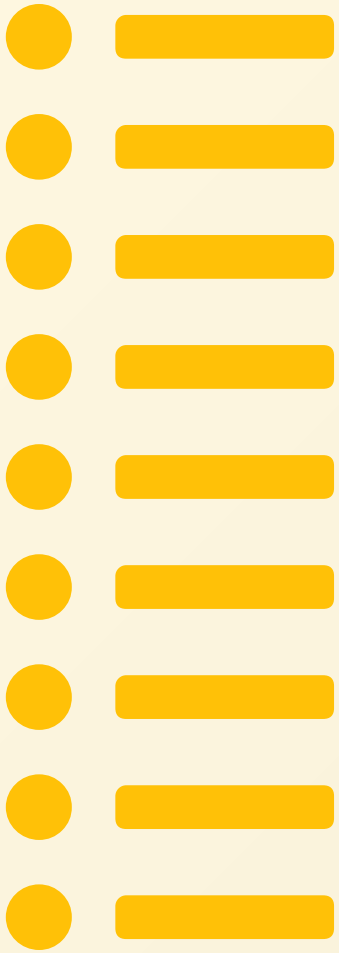
Tests

Solutions

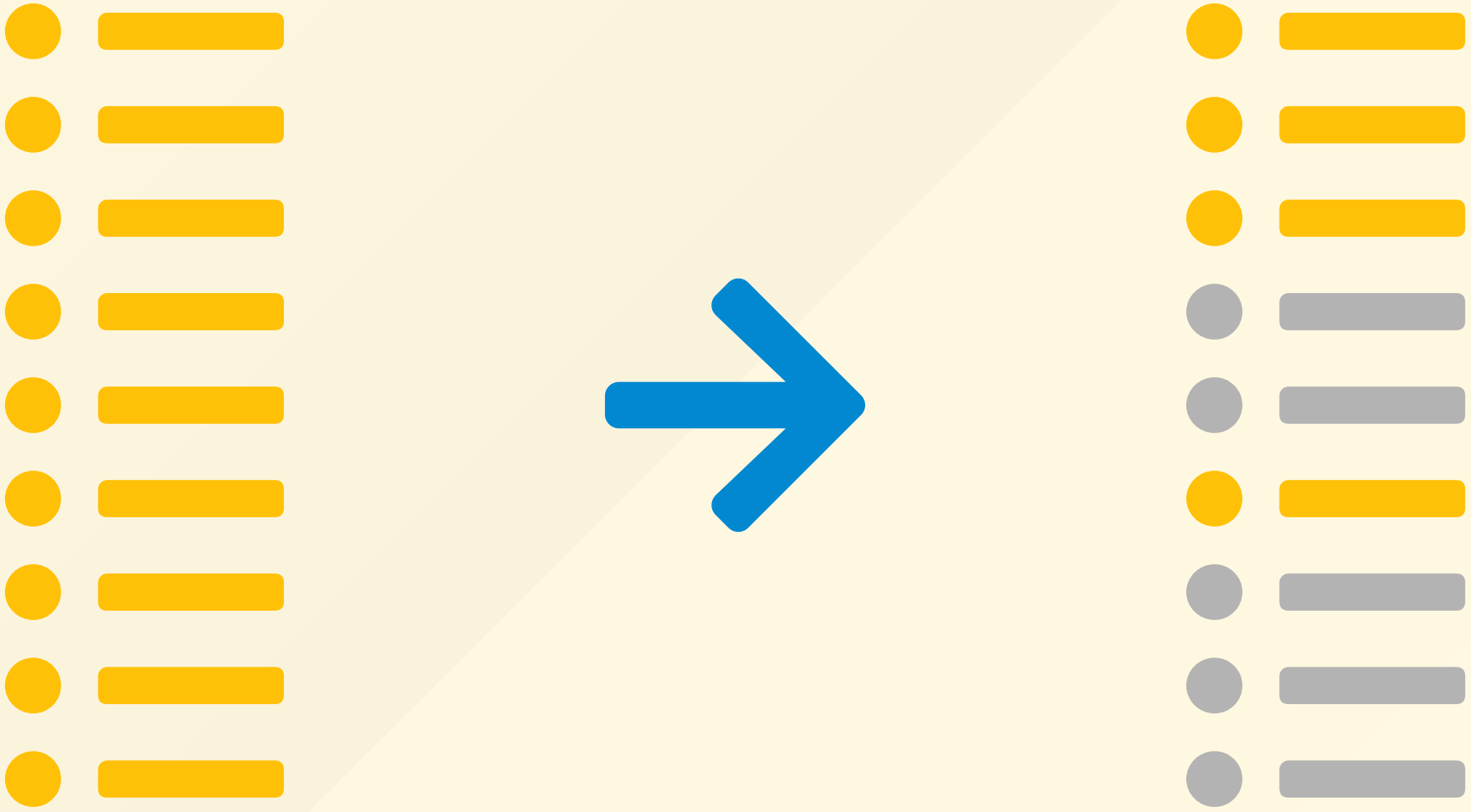
Solutions

Test Case Selection

Solutions / Test Case Selection



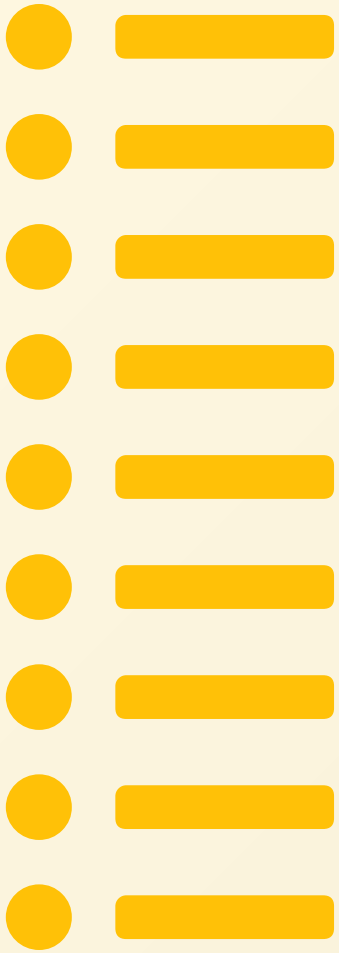
Solutions / Test Case Selection



Solutions

Test Suite Minimisation

Solutions / Test Suite Minimisation

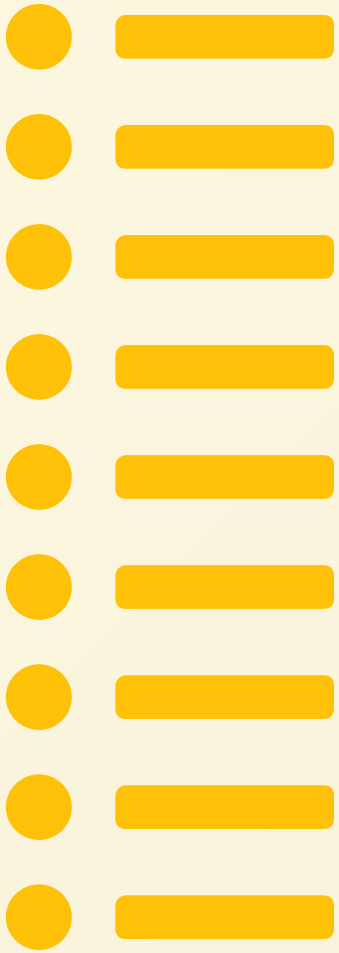


Solutions / Test Suite Minimisation

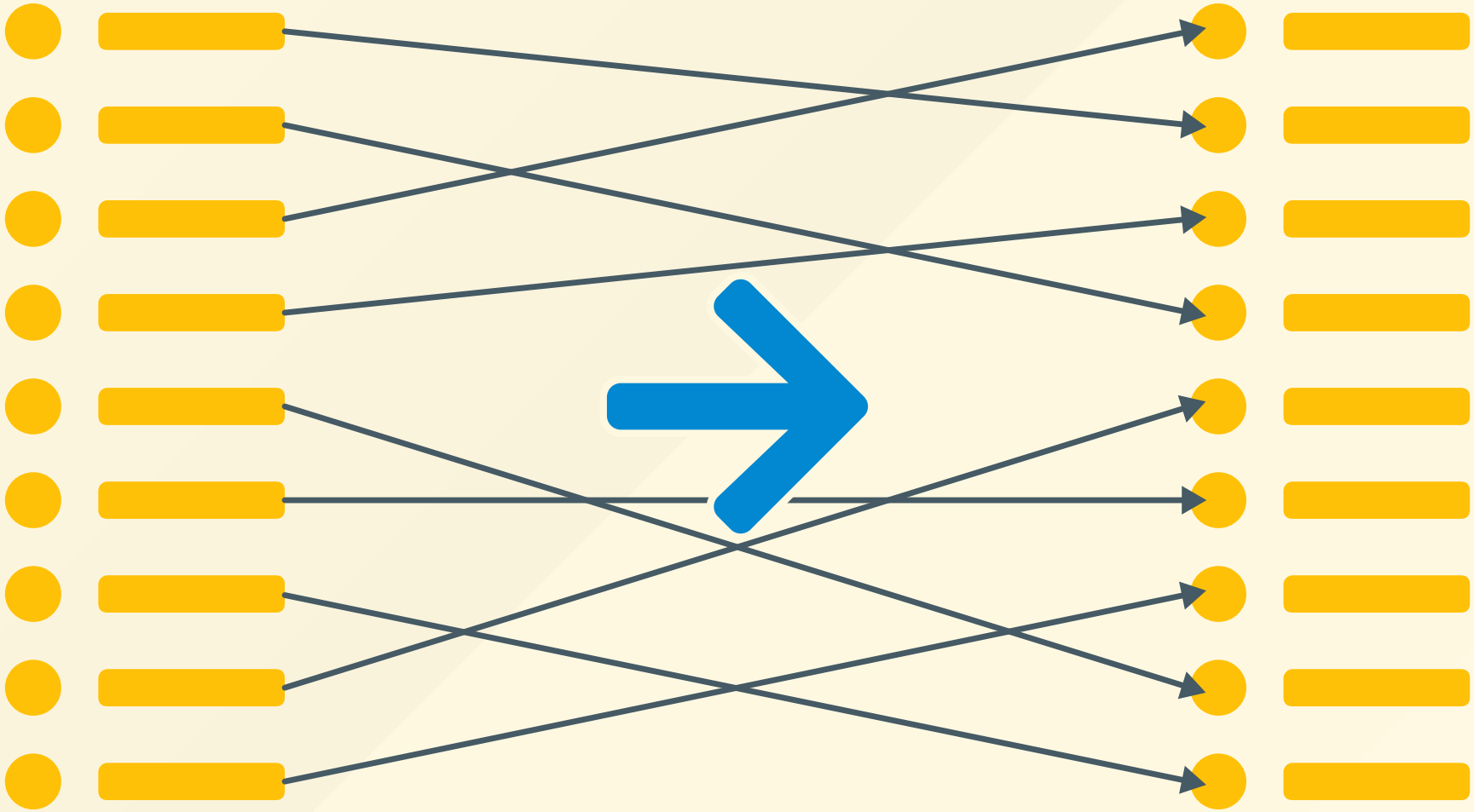
Solutions

Test Case Prioritisation

Solutions / Test Case Prioritisation



Solutions / Test Case Prioritisation



So.. problem solved!

..right?


Well yes, but actually no

State of the art

State of the art

Java

State of the art

Java

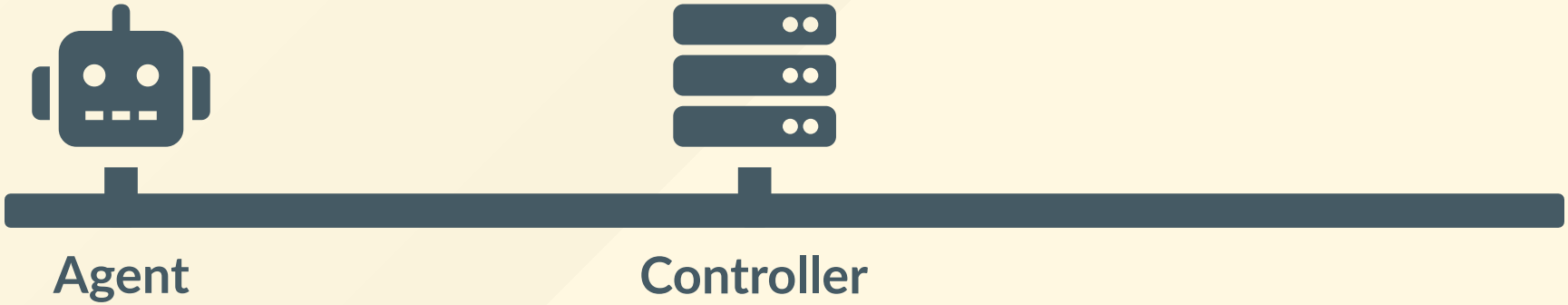
Other languages

Implementation

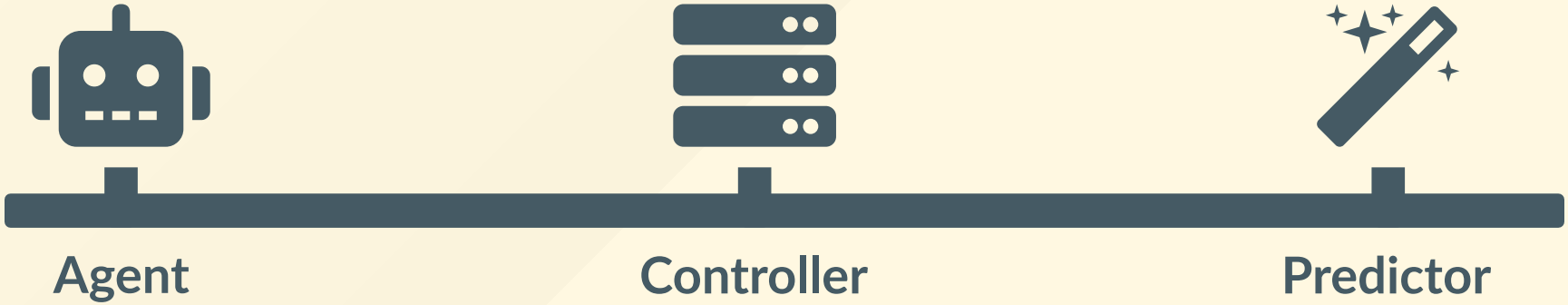
Implementation

Implementation

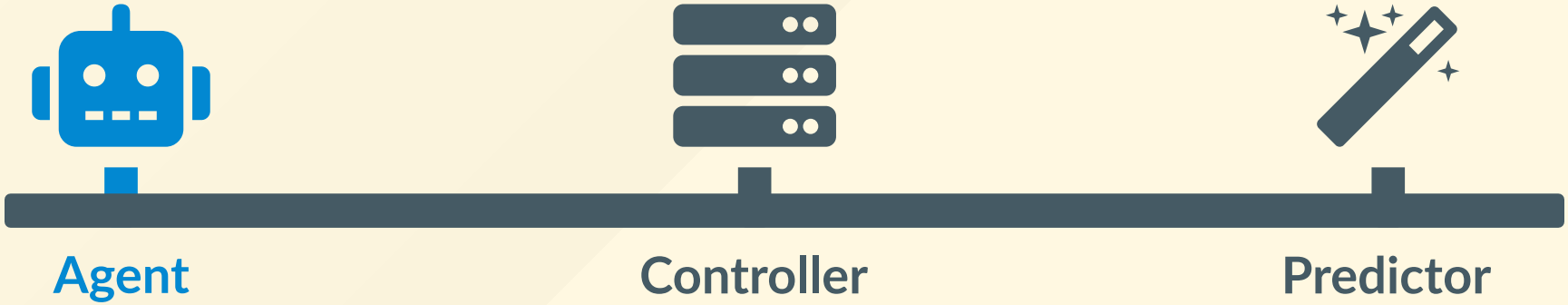
Implementation



Implementation



Implementation



Implementation / Agent

Execute tests

Implementation / Agent

Execute tests

Feedback

Implementation

Implementation / Controller

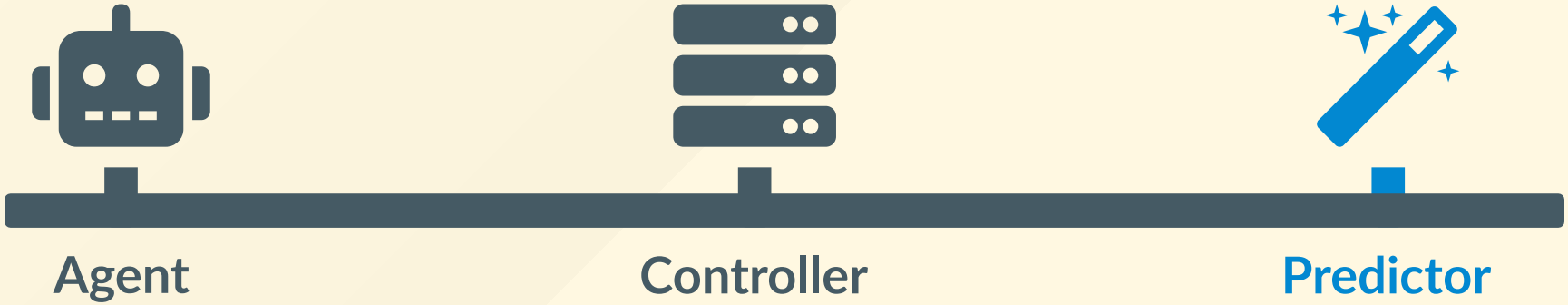
Routing

Implementation / Controller

Routing

Analysis

Implementation

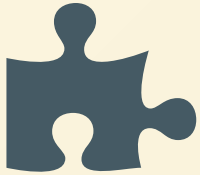


Implementation / Predictor

Determine order

Implementation / Predictor

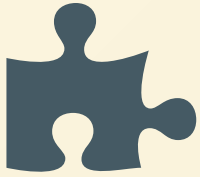
Determine order



10 algorithms

Implementation / Predictor

Determine order



10 algorithms

Extensible

```
# Generate a random order.
def predict(test_cases, coverage, results, duration):
        return shuffle(test_cases)
```

Implementation / Alpha-algorithm

Implementation / Alpha-algorithm

1. Unstable, affected test cases (by duration)

Implementation / Alpha-algorithm

1. Unstable, affected test cases (by duration)
2. Affected test cases (by duration)

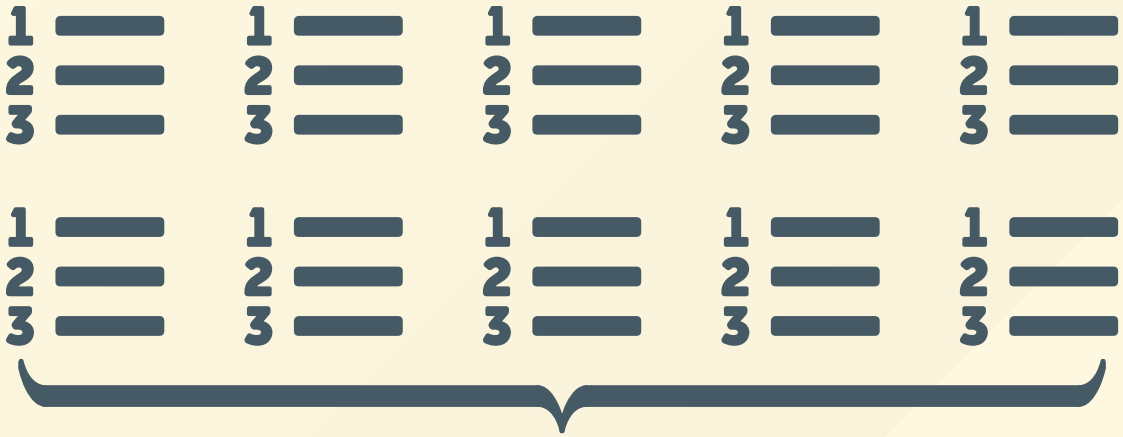
Implementation / Alpha-algorithm

1. Unstable, affected test cases (by duration)
2. Affected test cases (by duration)
3. Test cases based on added coverage

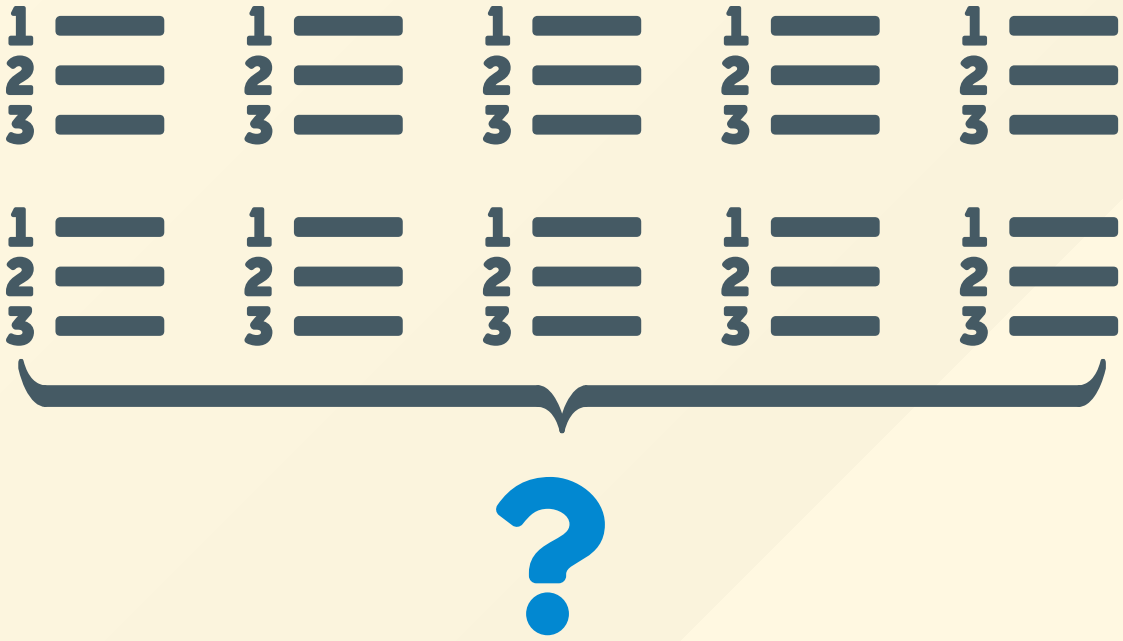
Implementation / Alpha-algorithm

1. Unstable, affected test cases (by duration)
2. Affected test cases (by duration)
3. Test cases based on added coverage
4. Other test cases `[redunant]`

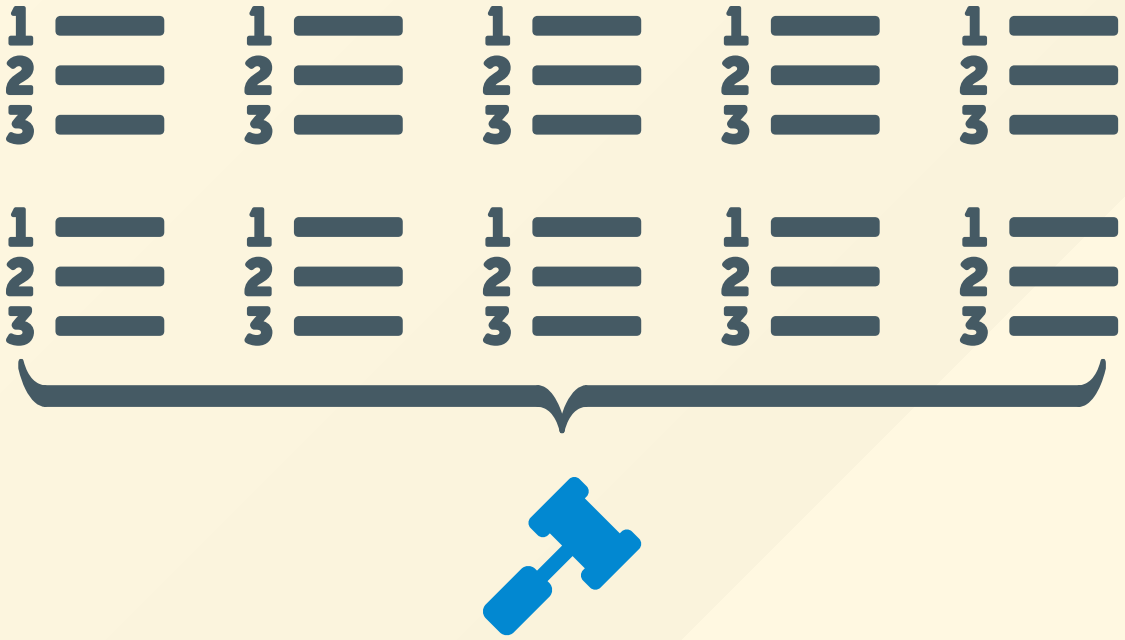
Implementation / Meta predictor



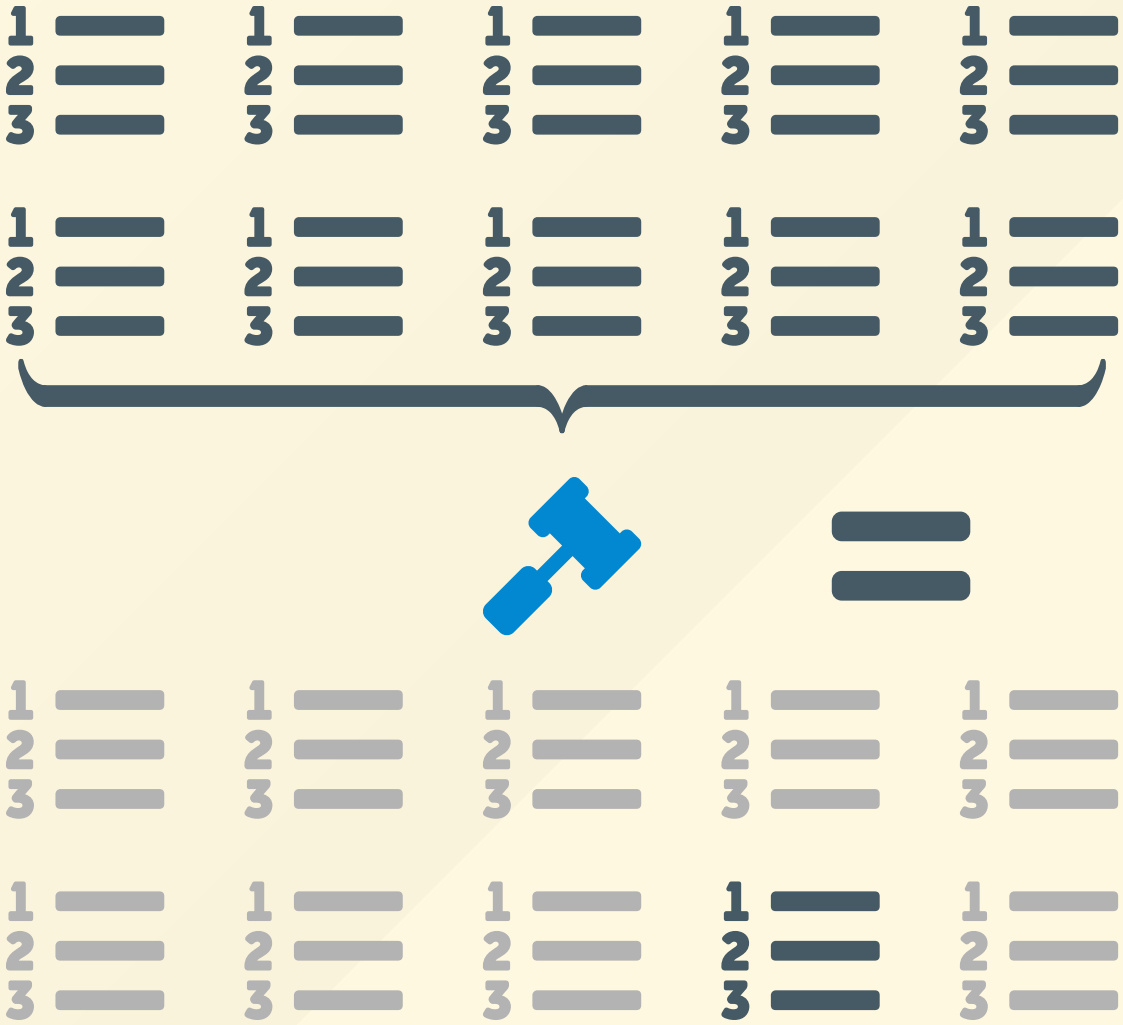
Implementation / Meta predictor



Implementation / Meta predictor



Implementation / Meta predictor

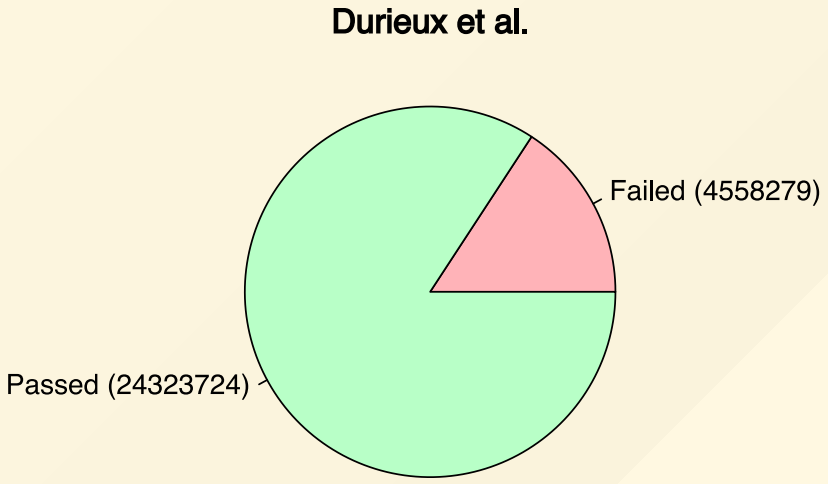


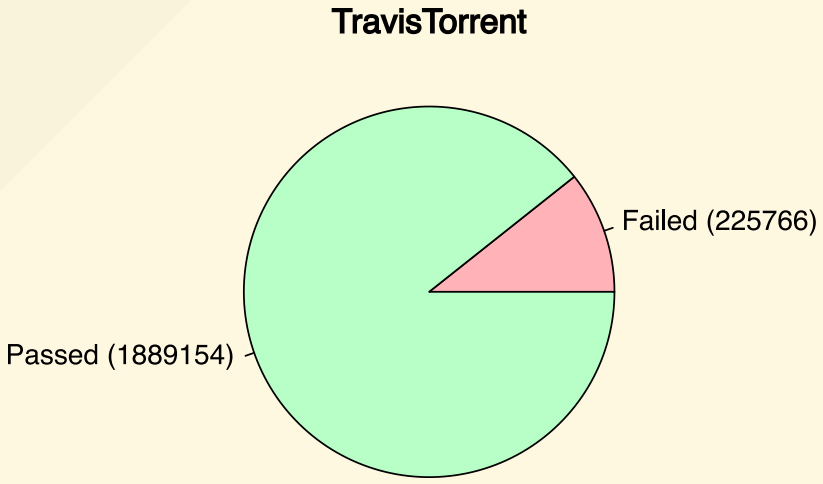
ALGORITHM	SCORE
Alpha	20
Greedy	10
HGS	-3

Results

Results

RQ1: Failure probability

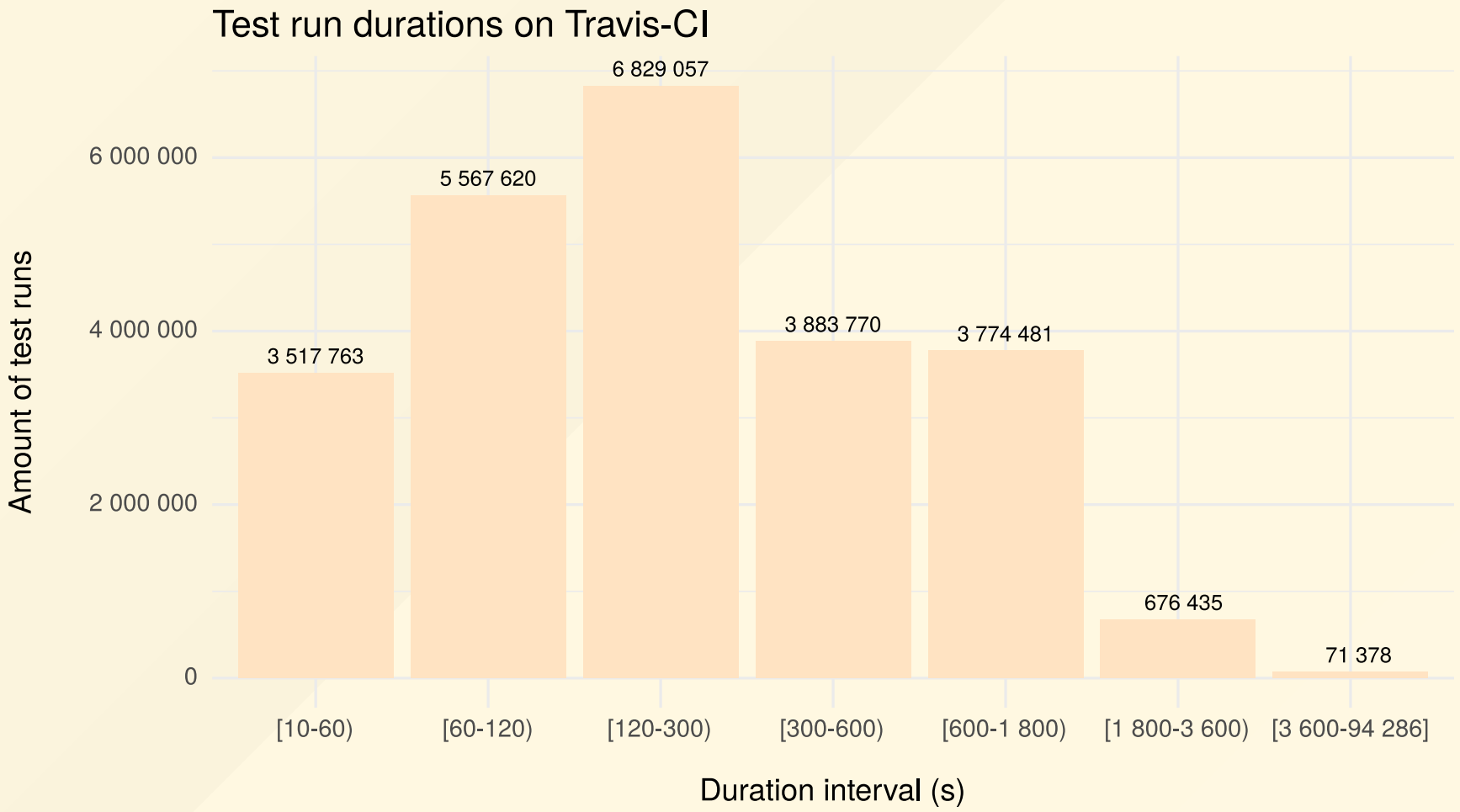




11% - 19%

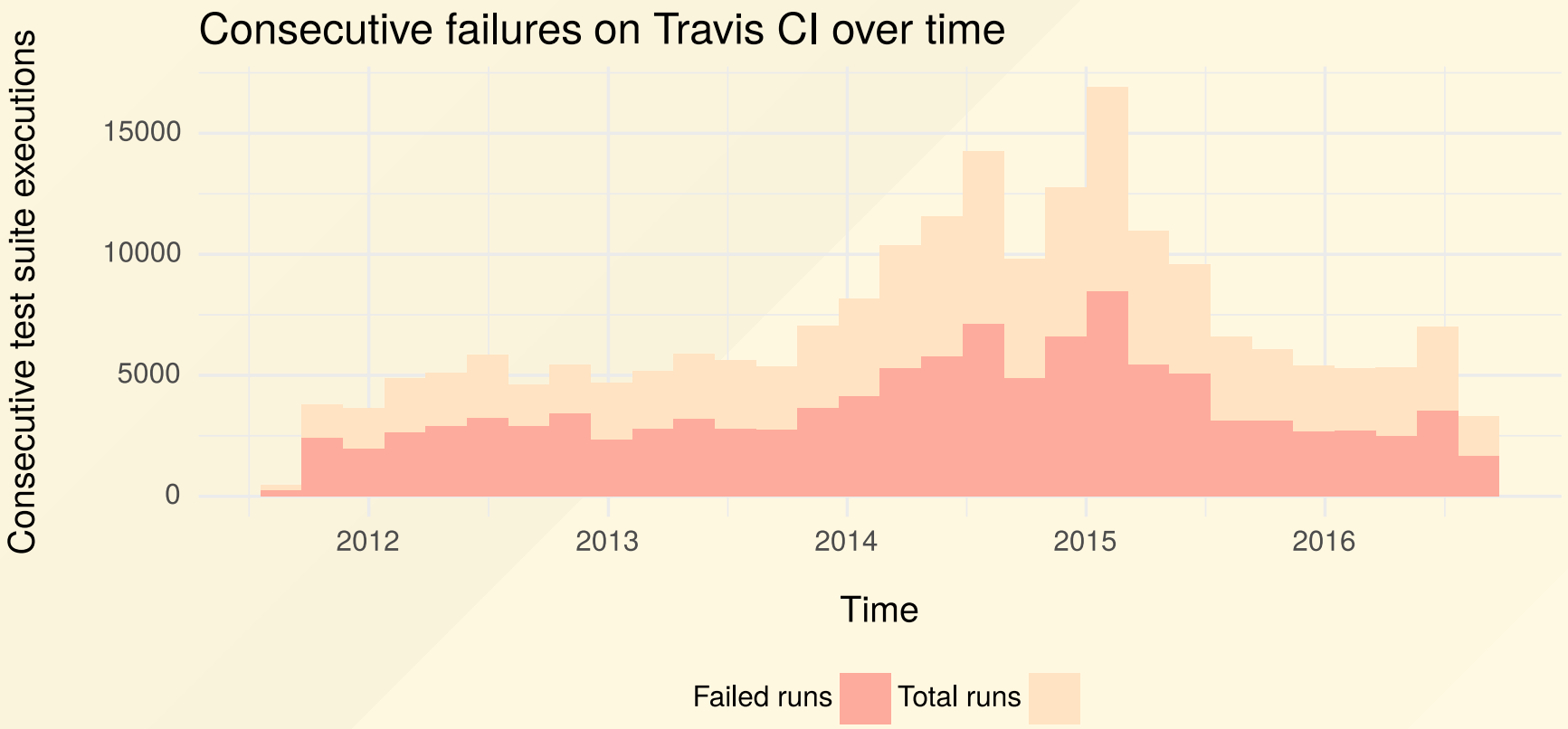
Results

RQ2: Average test run duration



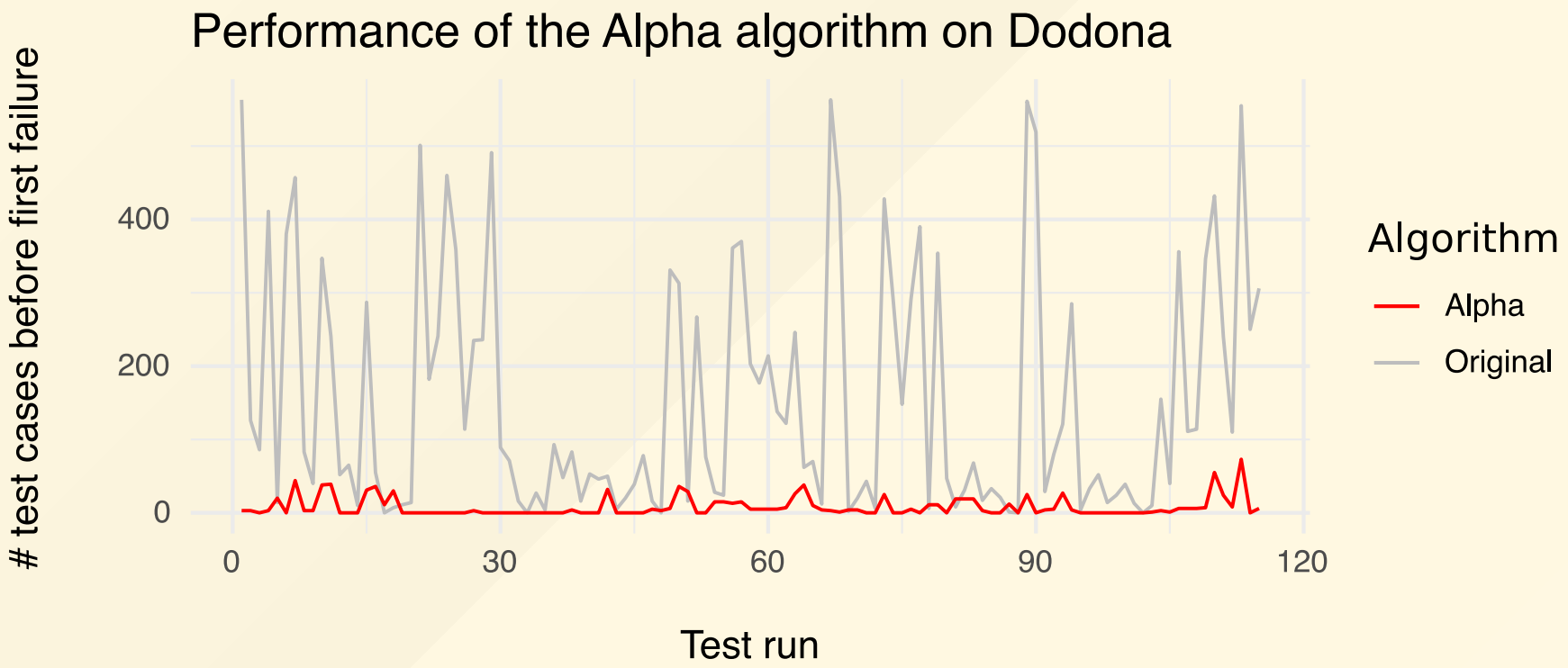
Results

RQ3: Consecutive failure probability



Results

RQ4: Performance on Dodona (Tests)

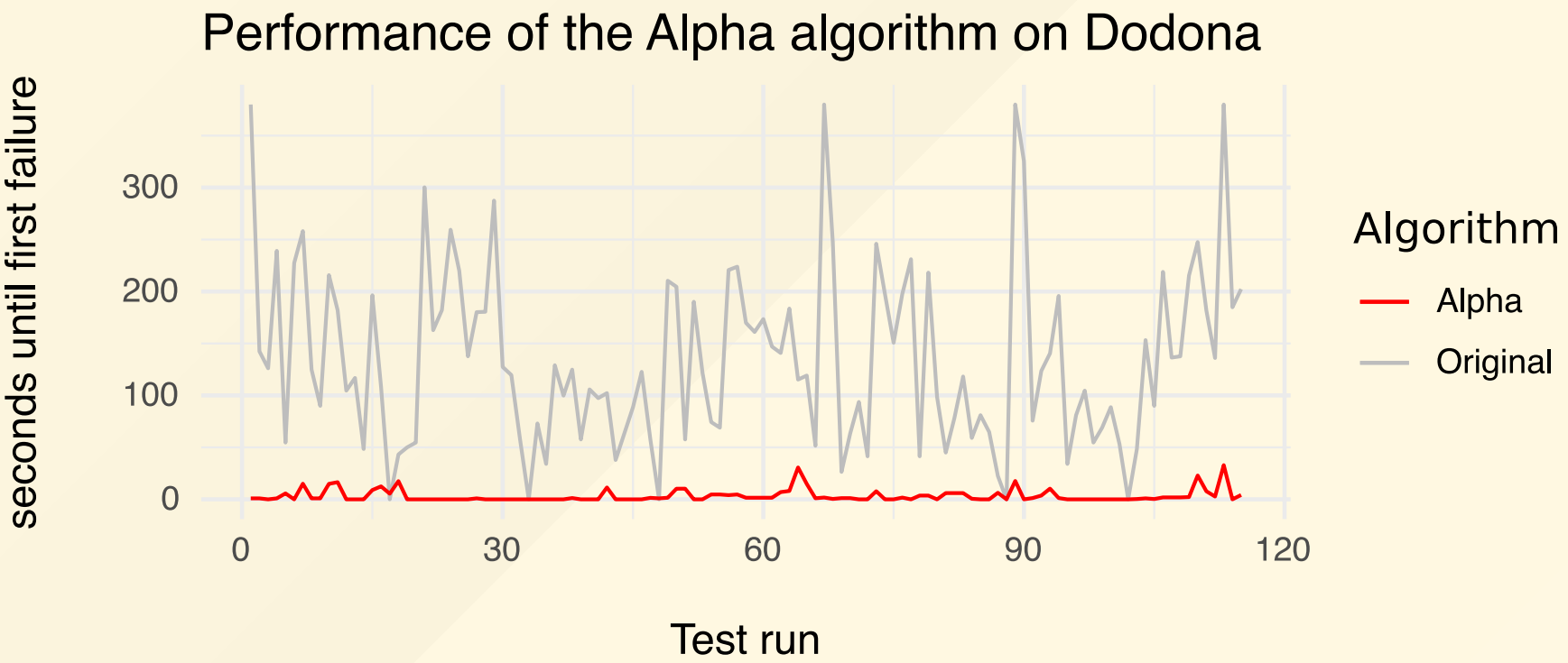


 # test cases: < 25x

until first observed failure

Results

RQ4: Performance on Dodona (Duration)



duration: < 40x

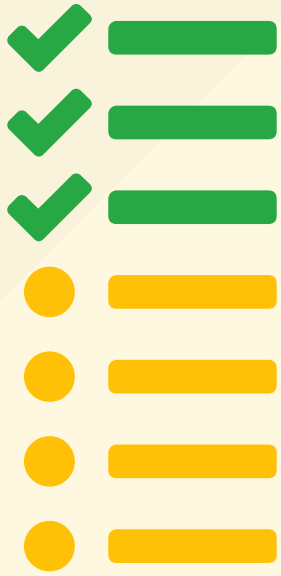
until first observed failure

Demo

Wrapping up

Conclusion

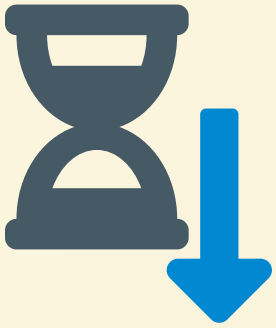
Conclusion



Conclusion

Test Case Prioritisation

Conclusion



Waiting time

Productivity

Questions?

References

- Slides created using Marp.
- Icons are property of FontAwesome.