

# **CASE STUDY**

## **CONTINUOUS PERFORMANCE TESTING WITH MACHINE LEARNING**

# TaaS: Performance testing

## Description:

Some huge Icelandic bank, had a request for checking their application performance during their migration from archaic technology to .NET services. Application is to be integrated with a lot of third party system and consist of three layers. The test team was responsible not only for searching bugs and creating performance testing framework, but also for teaching client for performance testing skills

## Challenges:

- No documented knowledge about the system
- No special environment and no any test data
- No knowledge and expectation about performance testing
- No accesses
- No monitoring system

# Platforms / Technologies:

## Web-UI performance:

- *Python*
- *Selenium*
- *Robot*
- *Browser mob proxy*
- *Har-storage*
- *mongoDB*

## Backend performance:

- *Jmeter*
- *groovy*
- *R*
- *MsSql*
- *Shiny*
- *Dynatrace*
- *DotMemory*
- *DotTrace*

# TaaS: Performance testing

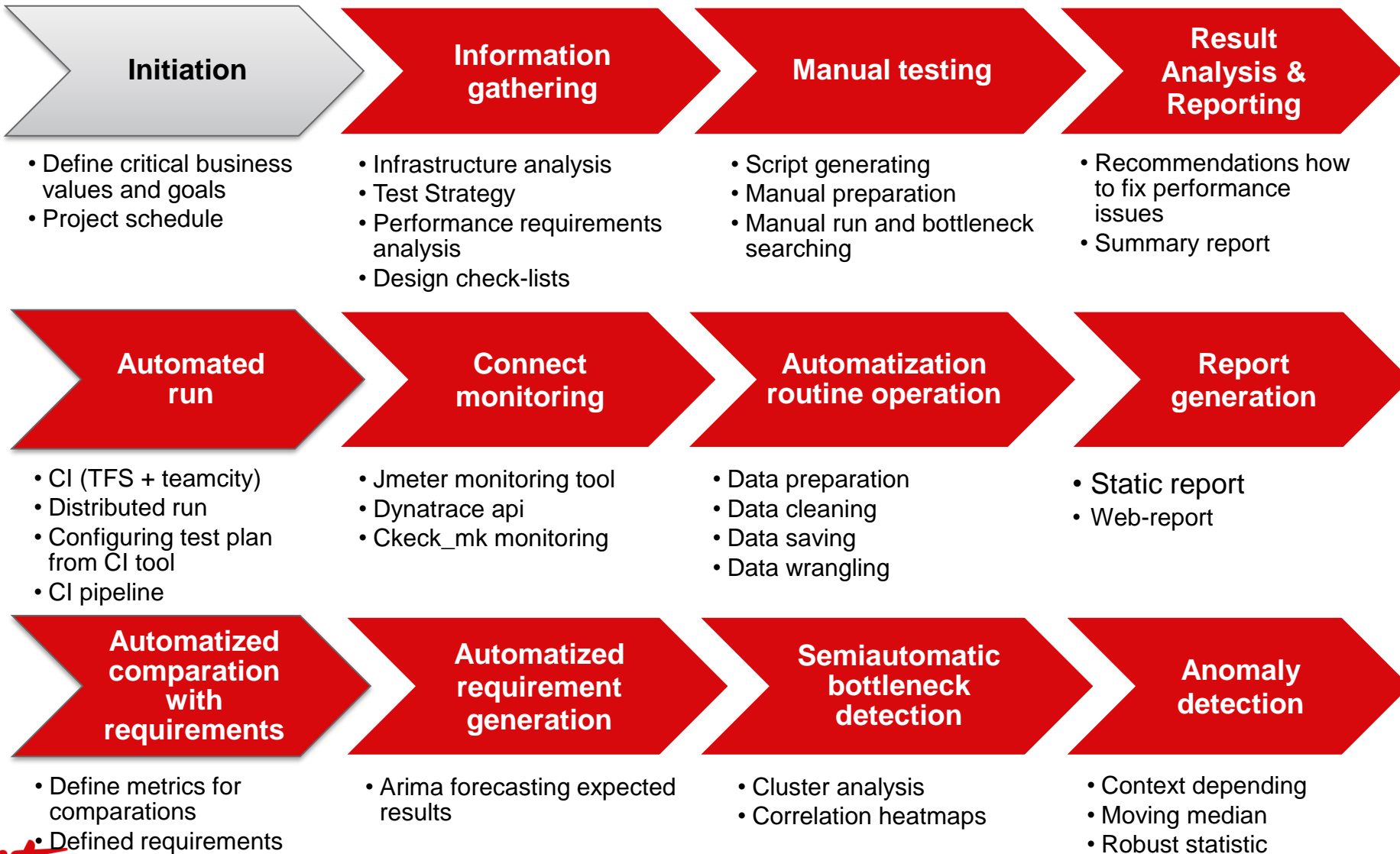
## Approach:

- Implementation of structured performance testing life cycle
- Integration of performance testing in continuous integration
- Automation of repetitive performance testing pipeline (smoke, load, scalability, stability, error detection)
- Maximal metric measurement with afterward analysis
- Maximal automatize of routine activities (data preparation and first analysis)
- Tight communication with onshore development team and customer teams

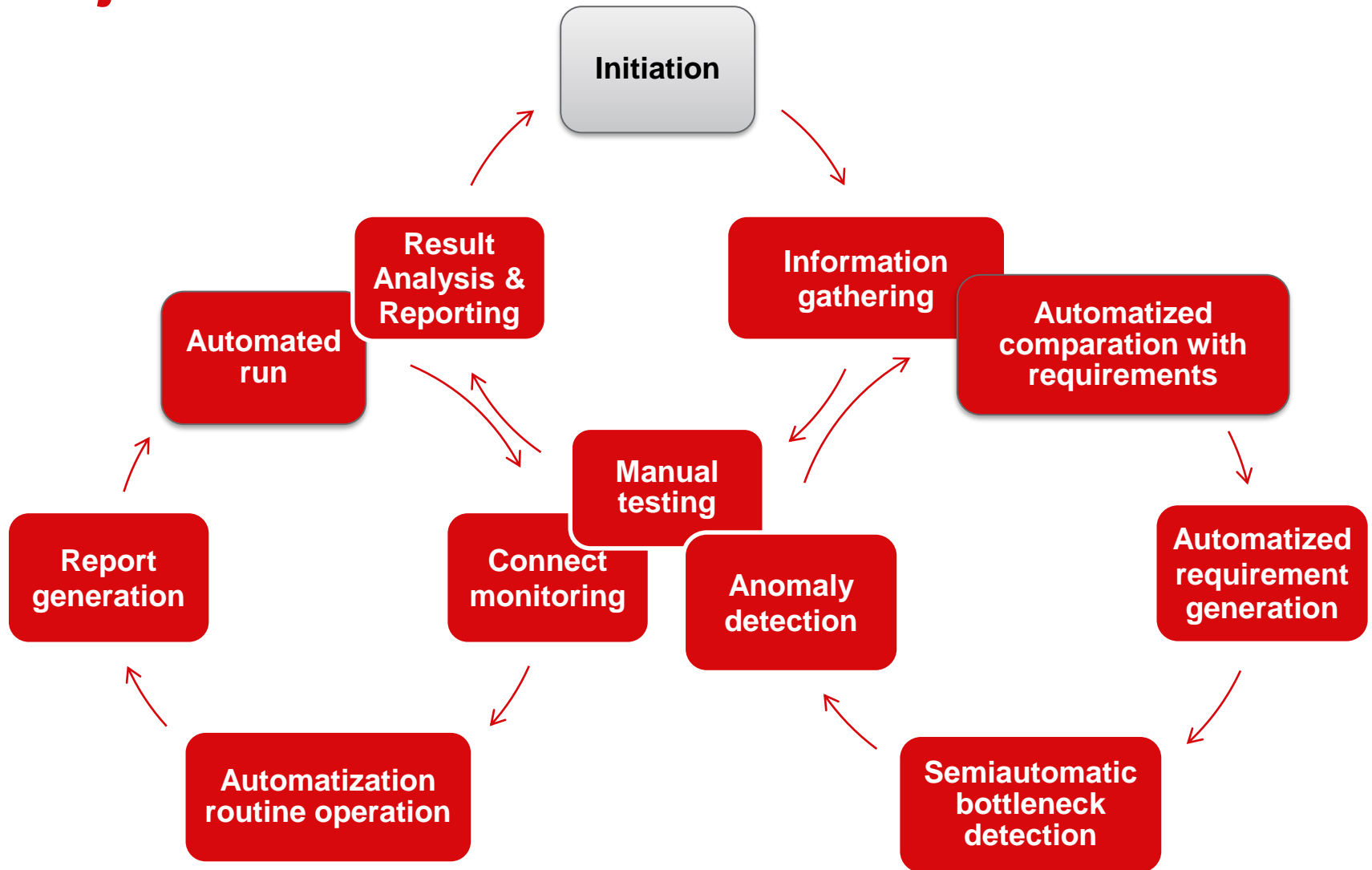
## Scope:

- Describe performance strategy for the bank
- Build solid performance testing framework
- Establish reliable and effective performance testing process which can guarantee reliability and speed of all types of banking operations
- Fast and transparent feedback regards connections quality with third-party systems.

# Project Timeline



# Project Timeline in real life



# TaaS: Performance testing

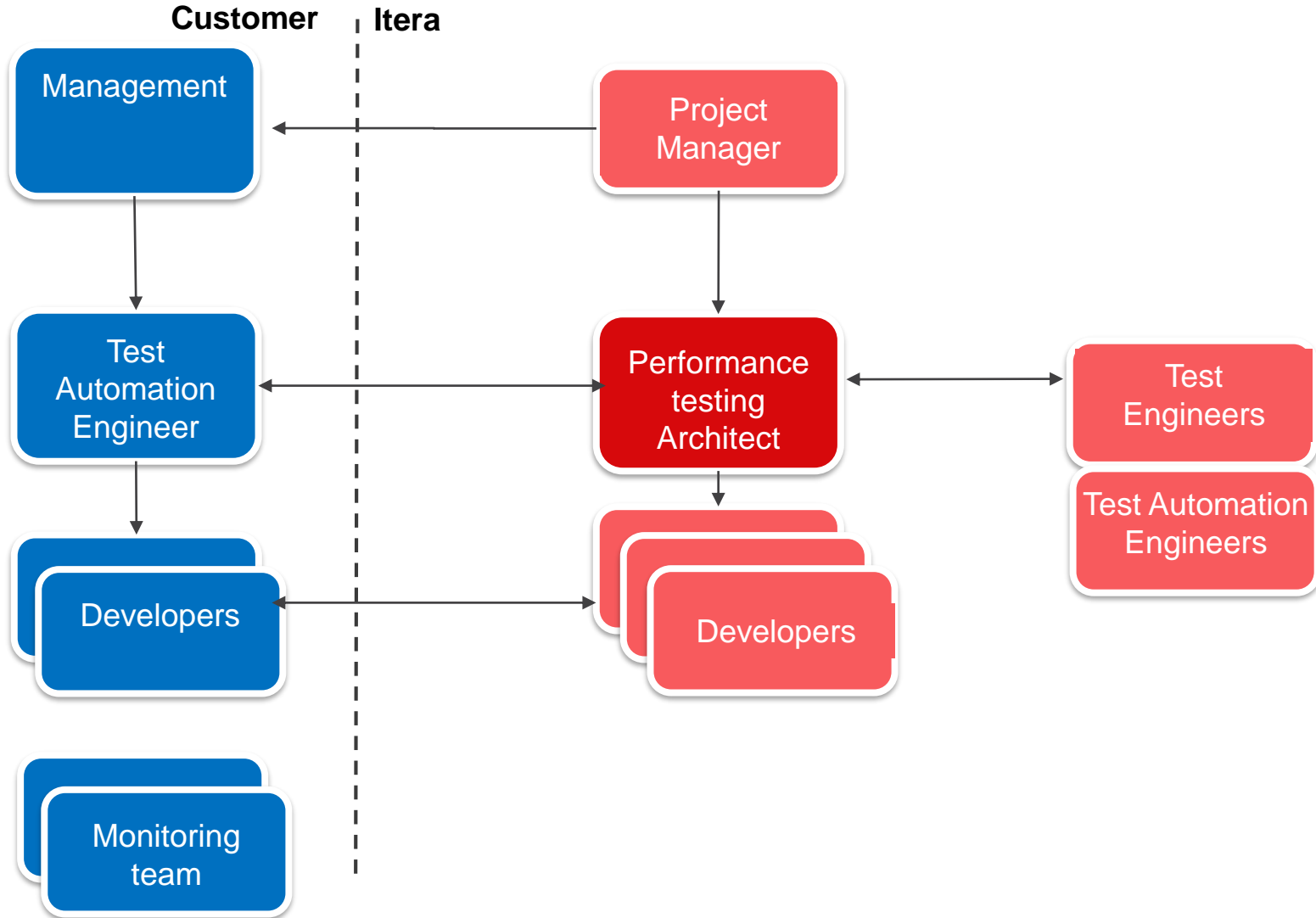
## Results:

- Performance testing framework
- Performance testing CI
- Automated analysis
- Automated report generation
- Semiautomatic bottleneck detection (using ML)
- Integration with monitoring system

## Deliverables:

- Test Strategy is introduced and followed
- Overall Test Scope is identified
- Performance testing framework (description and documentation)
- Scenarios and scripts automated (Smoke, load, scalability, stability)
- Customer issues analyzed (application, DB, infrastructure)
- Recommendation provided
- CI configured
- Daily ongoing automated reports and recommendation

# Performance team integration





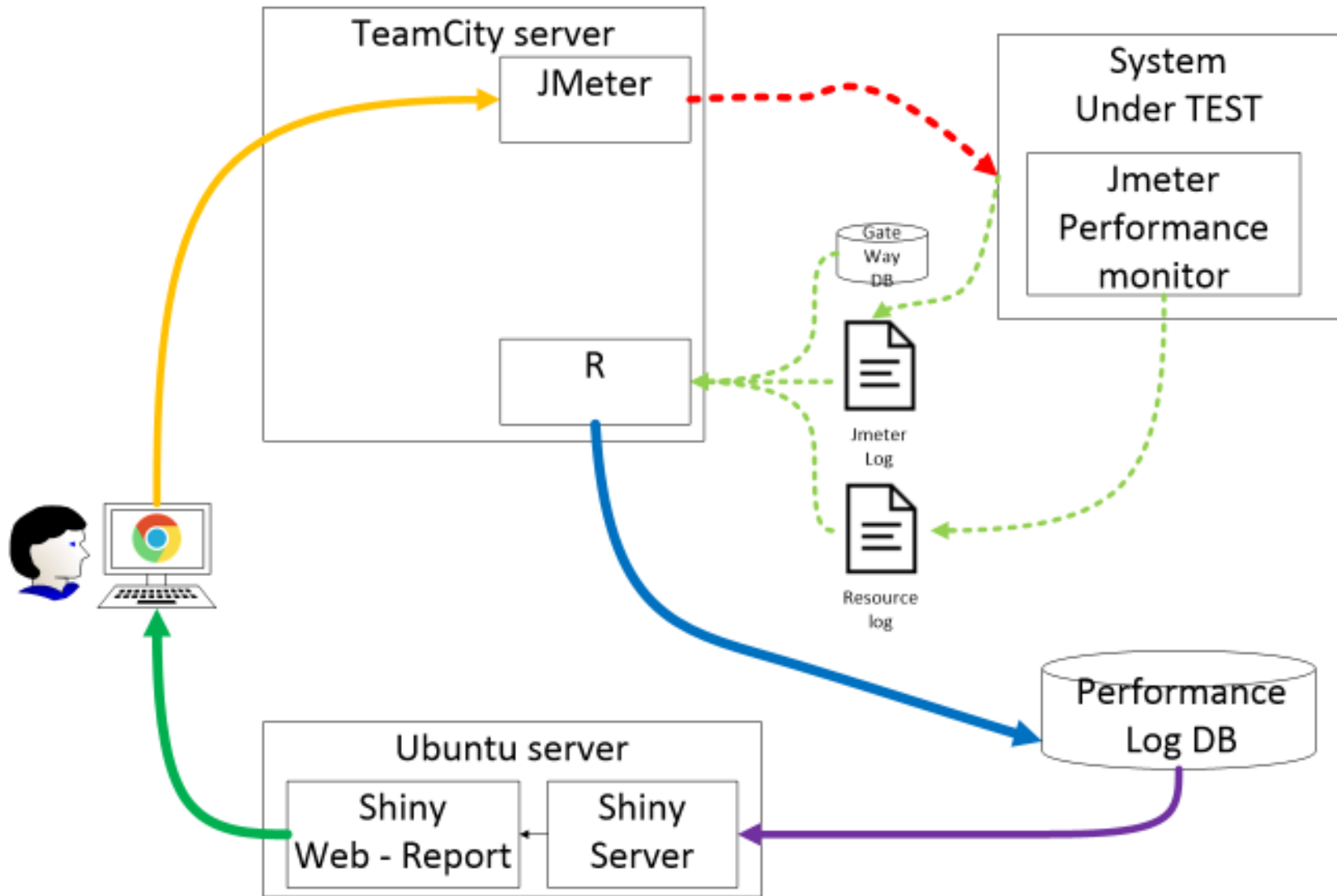
**Continuous  
PT**

**Report  
Generation**

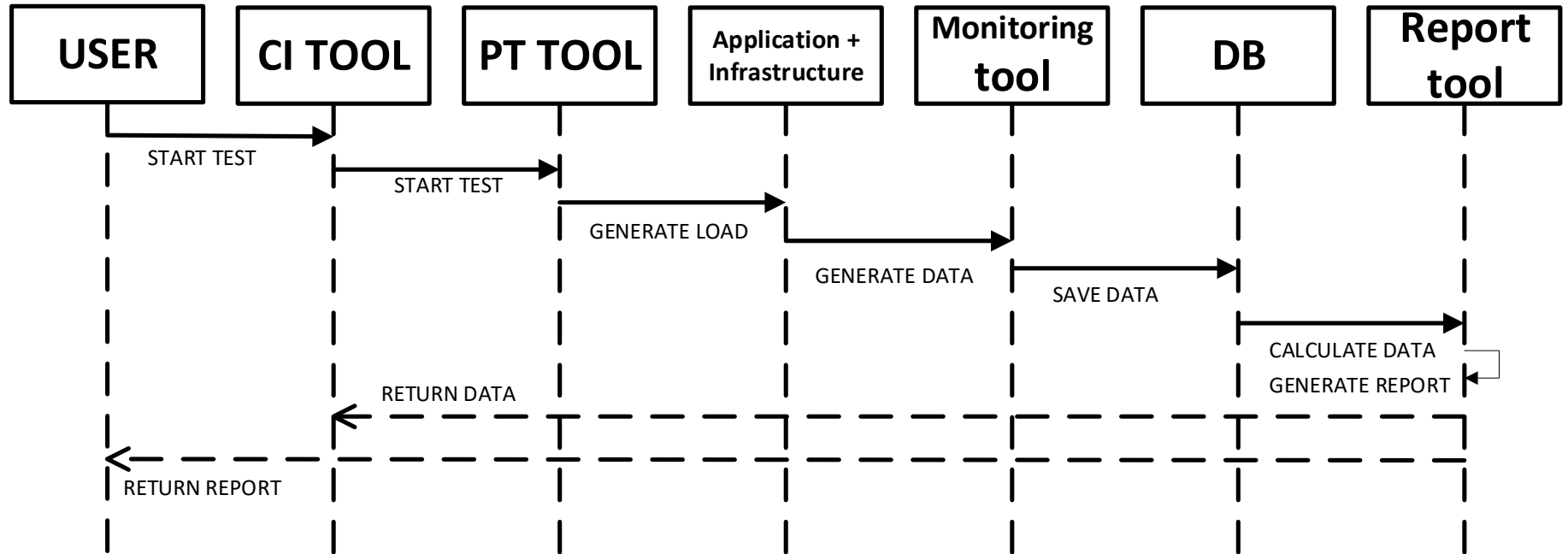
**Machine  
learning in  
PT**

**Recommendation  
for bottleneck  
detection**

**PT fairy ~~fails~~ tales**



# Continuous performance testing



# Collected data

## Response times:

- Client
- Servers
- DB
- Gateways

## Resources usage

- CPU
- RAM
- Network
- Disk IO

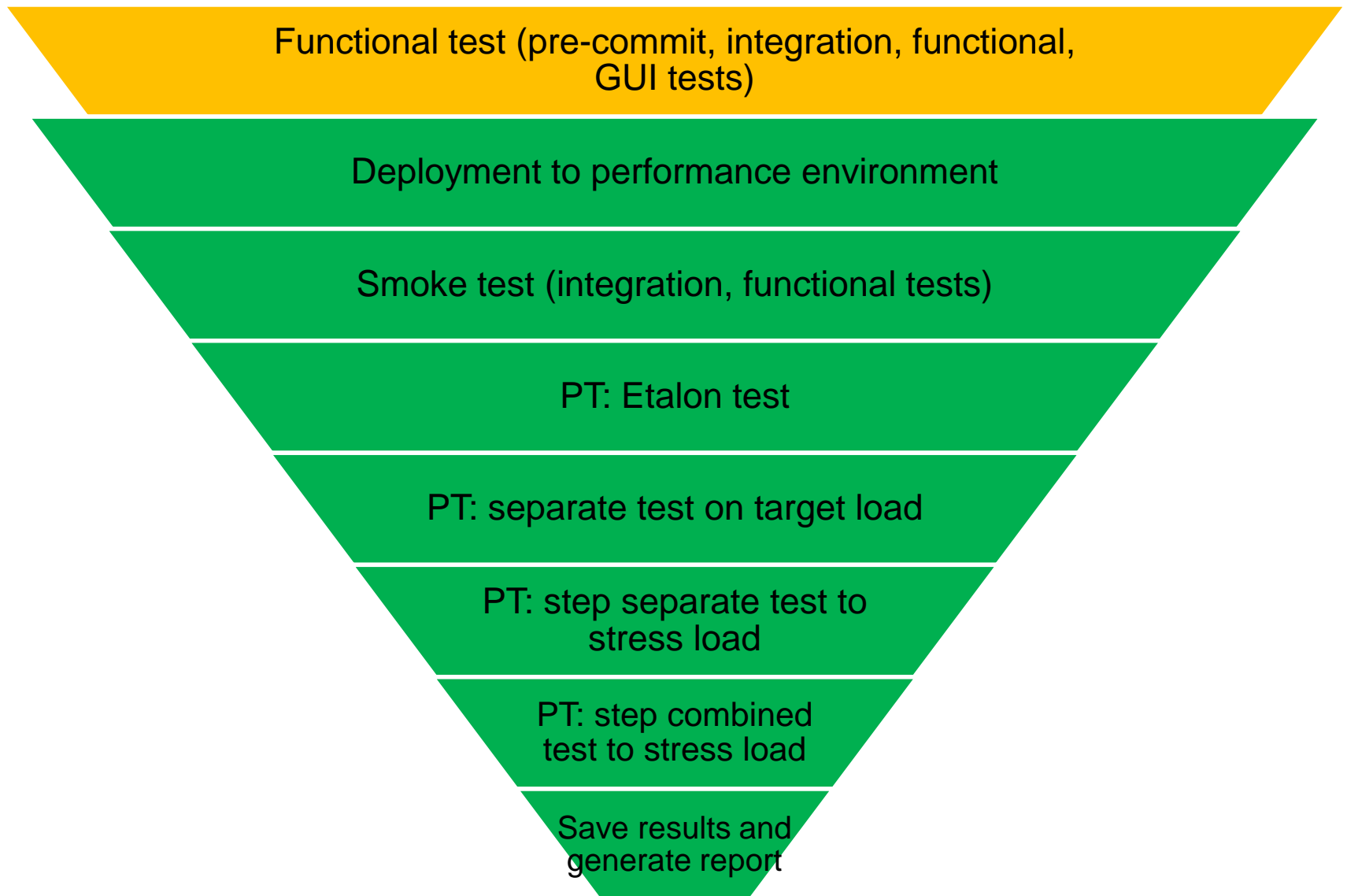
## Request phases:

- DNS lookup
- Connecting
- Sending
- Waiting
- Receiving

# Continuous performance testing – descriptio

- User interact and configure everything from CI tool (web-ui)
- CI tool interact with all elements via it's agents and instruments API
- Everything possible to configure from CI tool (Load, duration, versions, type of testing, report and log level)
- CI tool start load tools and load tools generate necessary load in one agent or distributed load, which also configured from CI tool
- CI tool gather all necessary information afterward performance test
- CI tool run analytical and report generator scripts
- CI tool save all information and logs (raw, cleaned, aggregated) into DB
- CI save all aggregated reports into artifacts
- User may interact with report tool vi web-ui
- Report tool should support filtering, comparing by many parameters (load, date, ifrastructure)

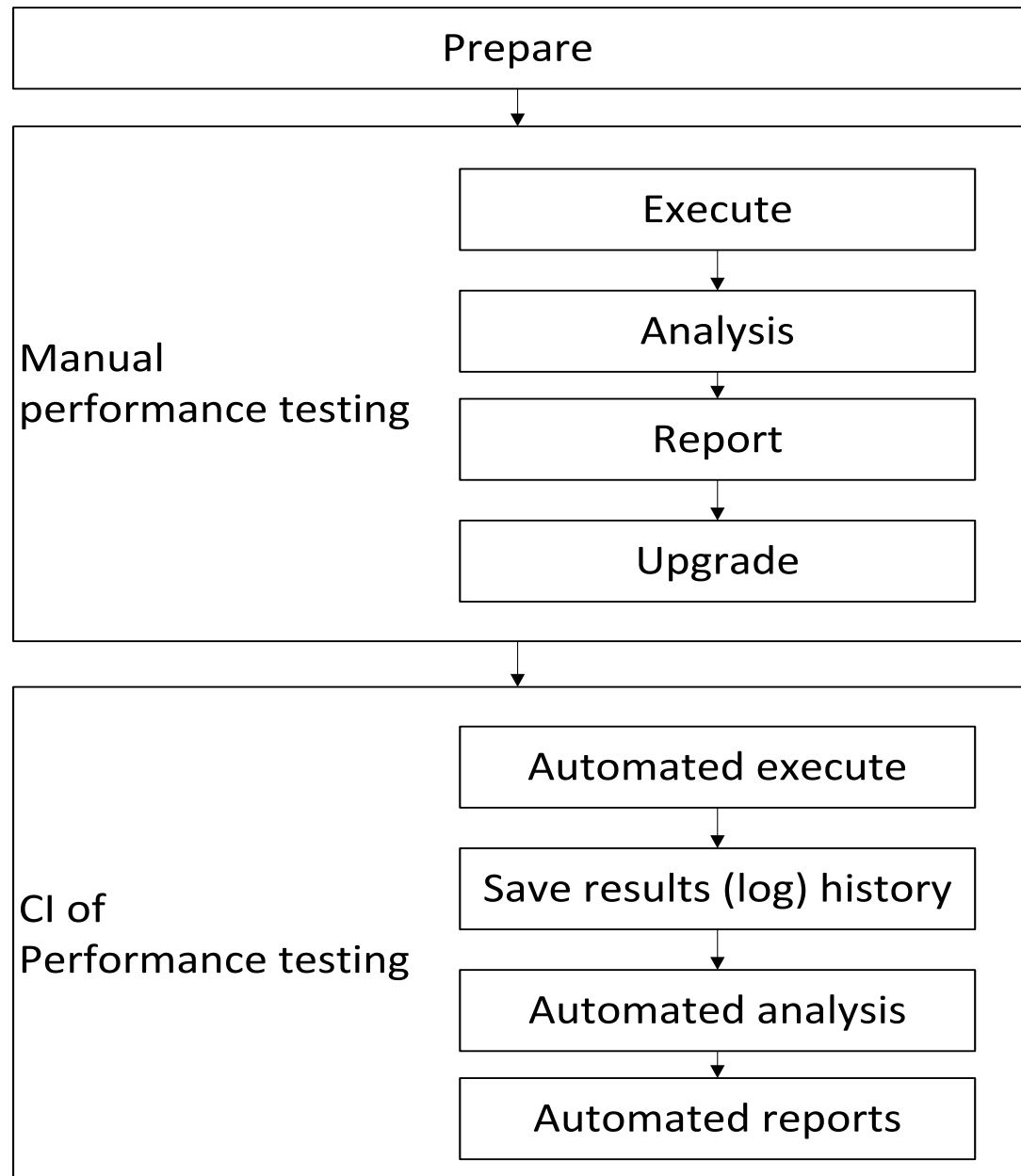
# Continuous performance testing



# Performance testing pipeline description

- Each steps should based on information from successful previous step
- Successful functional tests is precondition for start performance testing (PT takes too much time for idle run)
- Smoke test check availability of functionality and make warm up
- Smoke tests duration should non be more than minimal time execution (e.g. 5 minutes)
- Etalon test gather information about measurements in perfect conditions (e.g. 1 RPS, no more than open connection etc.). Most of comparation will be done with etalon conditions
- Separate tests shows how system works without other impact. It gives possibility compare and exclude additional impact in future tests.
- Target load gives possibility check system under expected load.
- But overload during stress tests gives possibility get overload point and detects systems thresholds
- Saving all information gives possibility to detect bottlenecks and not to waste time for rerun all tests

# Evolution Continuous performance testing





# Evolution Continuous performance testing

- Hardcoded scripts
- Manual executions
- Manual (eye analysis)
- Manual checks
- Manual report generators
- Configured scripts
- Configured runs
- Automatic executions
- Configured tests
- Results saving
- Automatic results analysis
- Automatic report generation
- Automatic anomaly detection
- Automatic recommendation generation

**Continuous  
PT**


**Report  
Generation**

**Machine  
learning in  
PT**

**Recommendation  
for bottleneck  
detection**



**PT fairy ~~fails~~ tales**

# Report generation

Shiny report 

Aggregate report   Day report   Resource Utilization   Scenarios Report   Aggregate info about subrequests   Detail timings for subOperations

miniGacker info by Guid   miniGacker info by ScenarioId   More ▾

**Numbers of threads**

**Date input**

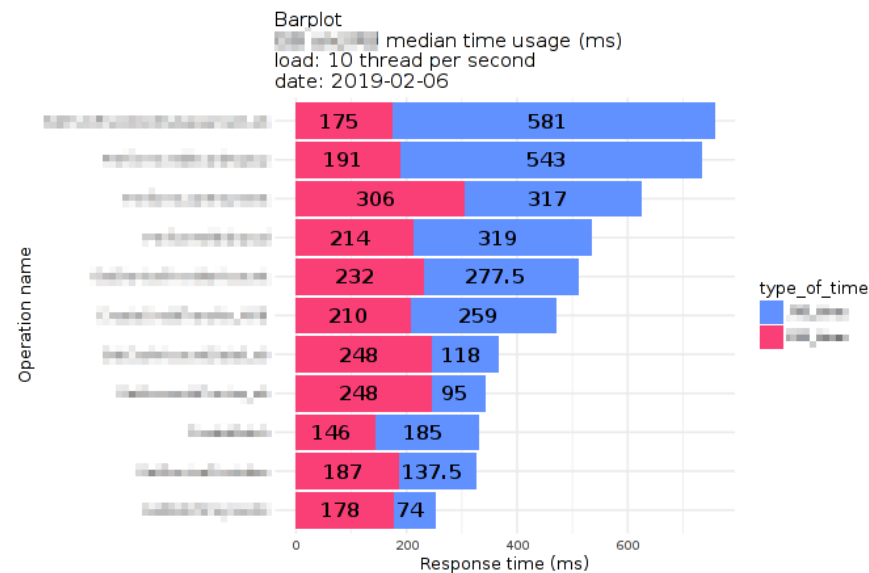
☒ Show only success responses:

[Dates for runs](#)   
 [Operations aggregated table](#)   
 [Target Times](#)

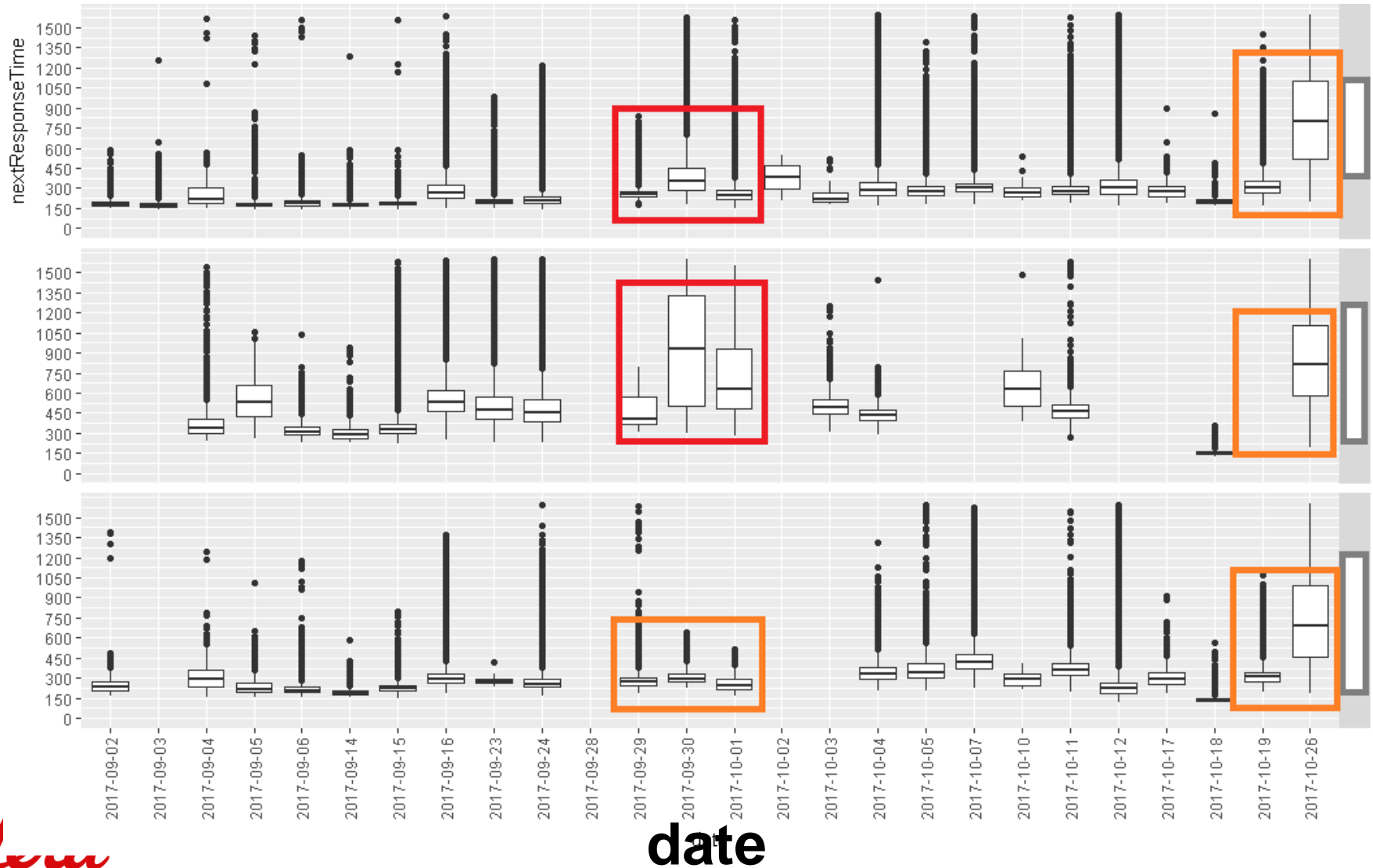
[Aggregated Table with !\[\]\(4729e517bc6a7cd81c8025b9646574fb\_img.jpg\) time](#)   
 [Aggregated Plots with !\[\]\(90a2fb2f2c617b26262139ae4159c0a0\_img.jpg\) time](#)   
 [MinMaxGuid's table](#)

[Daily errors table](#)   
 [Complex BoxPlots](#)

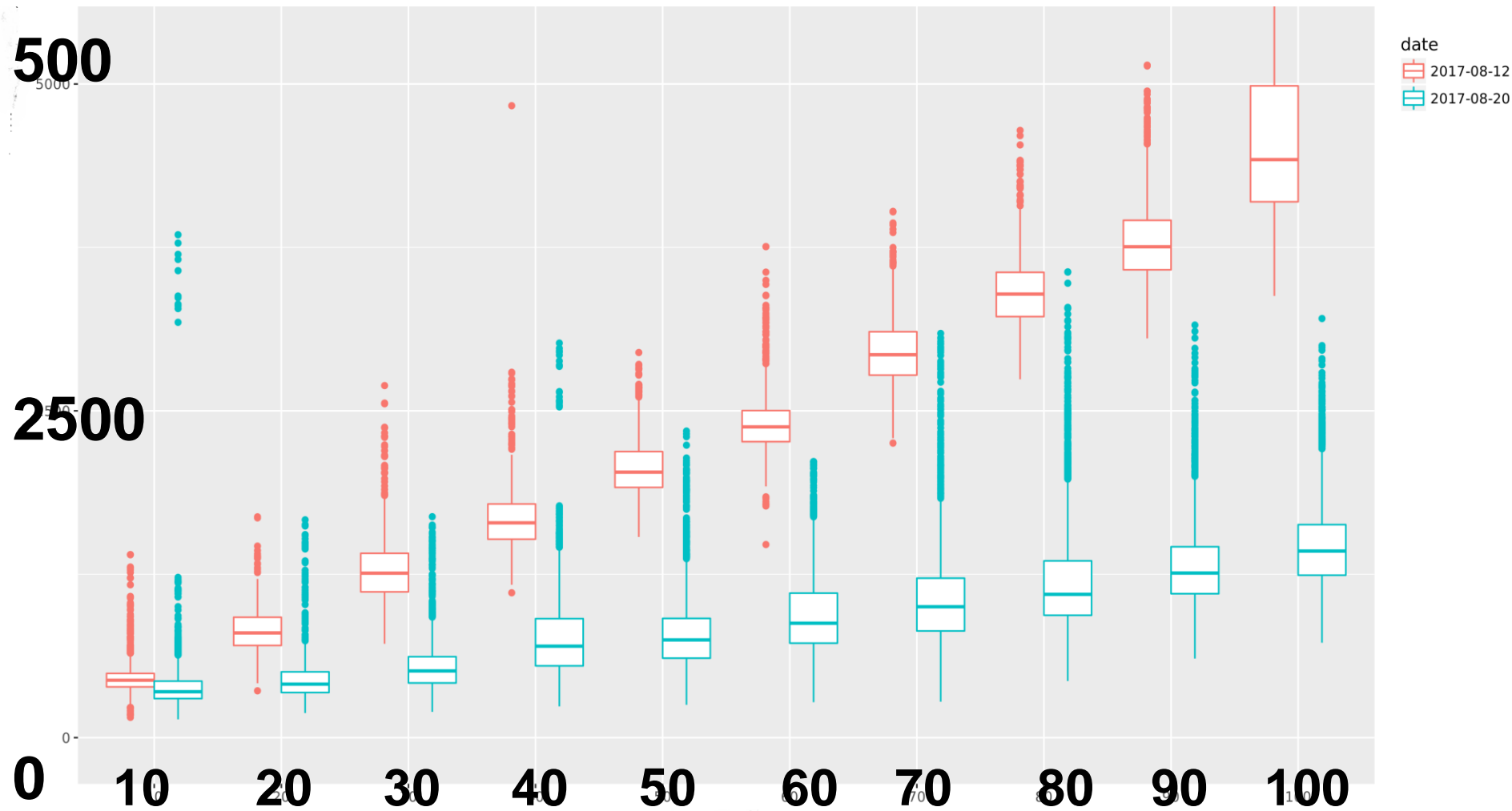
## Ratio between and times



# Report generation



# Report generation



# Report generation

- Based on typical log files reports should be divided on next parts:
- Summary report – show only things which go bad
- Aggregated report -- describe all measurements with all statistics and graphs
- Interactive report – should give full access to all historical data with possibility to compare, filter, split and unite different data types and measurements.
  - It should be some application with visualization of data
  - Should provide access to all history of data
  - Should provide access for all involved person
  - Should provide export raw data for analysis in external tools
  - Should be possibility to compare results and measurement between dates, loads, layers and others....

**Continuous  
PT**

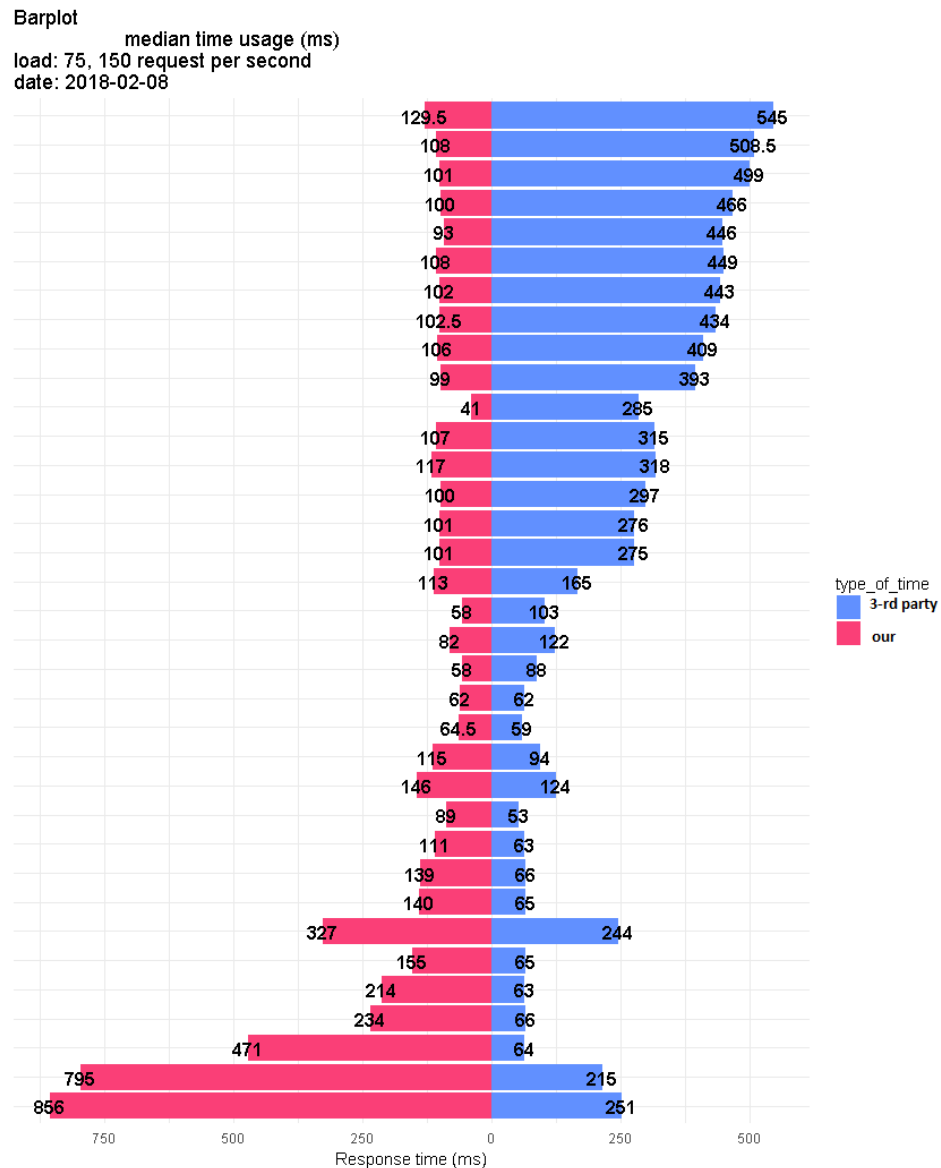
**Report  
Generation**

**Machine  
learning in  
PT**

**Recommendation  
for bottleneck  
detection**

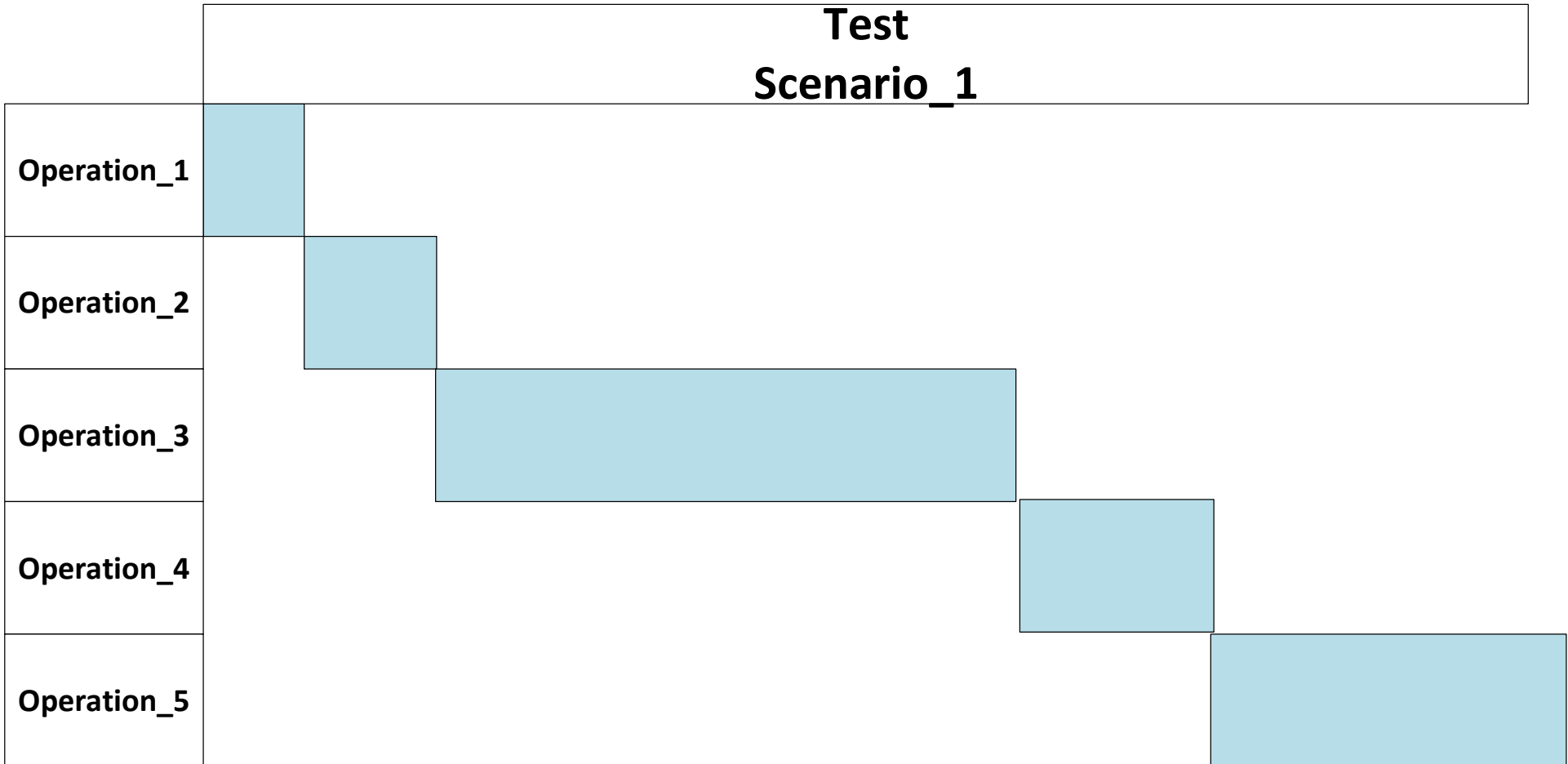
**PT fairy ~~fails~~ tales**

# Recommendation for bottleneck detections

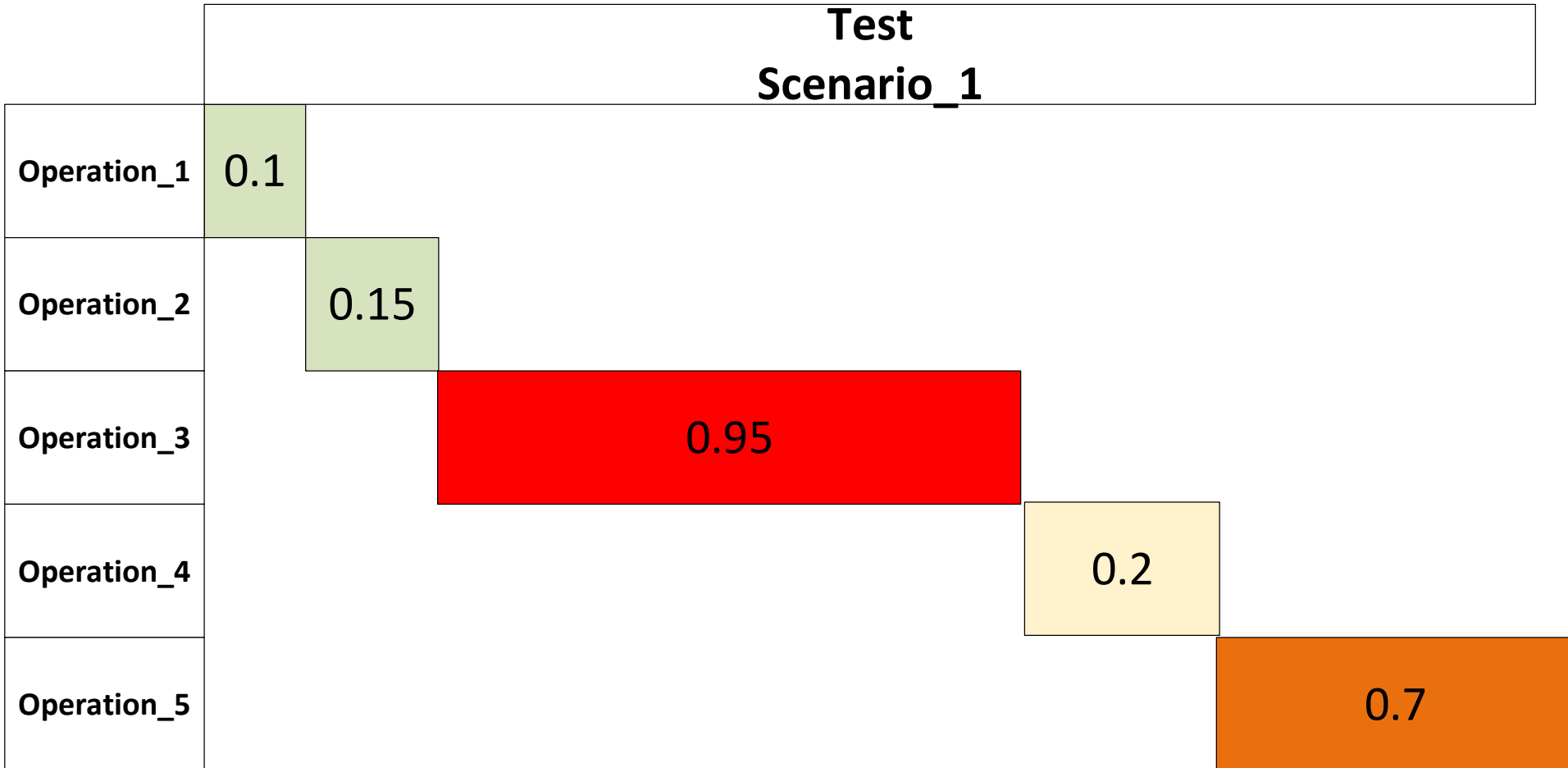




# Recommendation for bottleneck detections



# Recommendation for bottleneck detections



# Recommendation for bottleneck detections

|             | Test<br>Scenario_1 |
|-------------|--------------------|
| Operation_1 | 0.1                |
| Operation_2 | 0.15               |
| Operation_3 | 0.95               |
| Operation_4 | 0.2                |
| Operation_5 | 0.7                |

# Recommendation for bottleneck detections

|             | Test<br>Scenario_1 | Test<br>Scenario_2 | Test<br>Scenario_3 | Test<br>Scenario_4 | Test<br>Scenario_5 |
|-------------|--------------------|--------------------|--------------------|--------------------|--------------------|
| Operation_1 | 0.1                | 0.99               | 0.2                | 0.99               | 0.3                |
| Operation_2 | 0.15               | 0.9                | 0.25               | 0.1                | 0.2                |
| Operation_3 | 0.95               | 0.1                | 0.95               | 0.3                | 0.86               |
| Operation_4 | 0.2                | 0.5                | 0.75               | 0.4                | 0.7                |
| Operation_5 | 0.7                | 0.1                | 0.99               | 0.77               | 0.25               |

## Old build

|                       | REST<br>Response<br>time | SOAP<br>Response<br>time | SQL<br>Response<br>time | CPU<br>usage | RAM<br>usage |
|-----------------------|--------------------------|--------------------------|-------------------------|--------------|--------------|
| REST<br>Response time |                          | <b>0.75</b>              | 0.6                     | <b>0.7</b>   | 0.3          |
| SOAP<br>Response time |                          |                          | <b>0.8</b>              | <b>0.78</b>  | 0.2          |
| SQL<br>Response time  |                          |                          |                         | 0.4          | 0.2          |
| CPU<br>usage          |                          |                          |                         |              | 0.32         |
| RAM<br>usage          |                          |                          |                         |              |              |

## New build

|                       | REST<br>Response<br>time | SOAP<br>Response<br>time | SQL<br>Response<br>time | CPU<br>usage | RAM<br>usage |
|-----------------------|--------------------------|--------------------------|-------------------------|--------------|--------------|
| REST<br>Response time |                          | <b>0.73</b>              | 0.6                     | <b>0.7</b>   | 0.3          |
| SOAP<br>Response time |                          |                          | <b>0.9</b>              | <b>0.58</b>  | 0.2          |
| SQL<br>Response time  |                          |                          |                         | 0.4          | 0.5          |
| CPU<br>usage          |                          |                          |                         |              | 0.32         |
| RAM<br>usage          |                          |                          |                         |              |              |

## Difference

|                       | REST<br>Response<br>time | SOAP<br>Response<br>time | SQL<br>Response<br>time | CPU<br>usage | RAM<br>usage |
|-----------------------|--------------------------|--------------------------|-------------------------|--------------|--------------|
| REST<br>Response time |                          | <b>0.02</b>              | 0                       | <b>0</b>     | 0            |
| SOAP<br>Response time |                          |                          | <b>-0.1</b>             | <b>0.2</b>   | 0            |
| SQL<br>Response time  |                          |                          |                         | 0            | <b>-0.3</b>  |
| CPU<br>usage          |                          |                          |                         |              | 0            |
| RAM<br>usage          |                          |                          |                         |              |              |

# Recommendation for bottleneck detections

- To decrease time for bottleneck detection it is necessary gather information about all layers and all resources.
- Next it is necessary build correlation matrix predictor and predicted variables.
- Next step it is visualize it in heat map – and it shows what impact more on dependent variable.
- It is easiest way to drop all unimportant variables in variety of variables.
- For detecting what was changes with previous releases it just needed count differences between previous matrix and newest

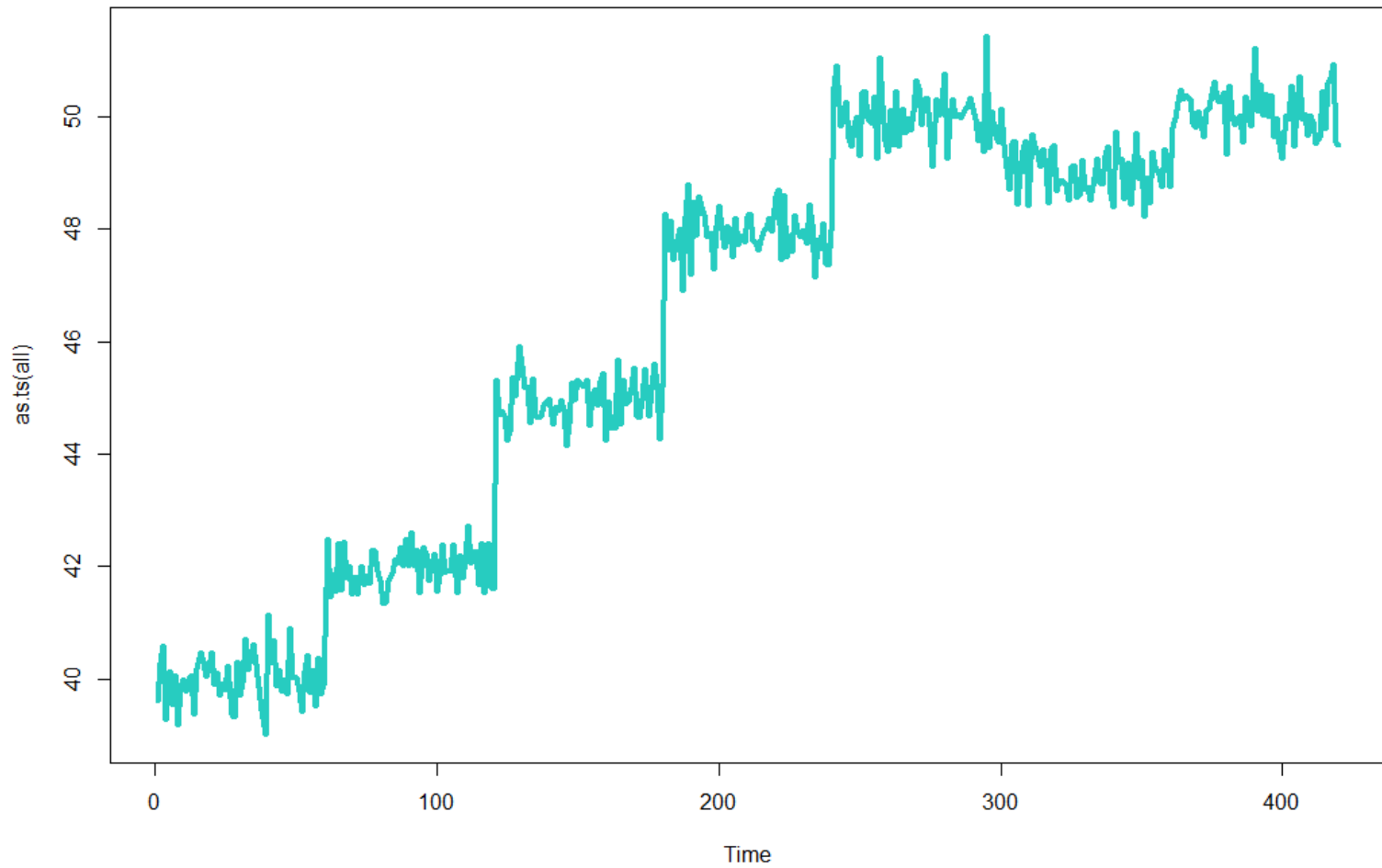
**Continuous  
PT**

**Report  
Generation**

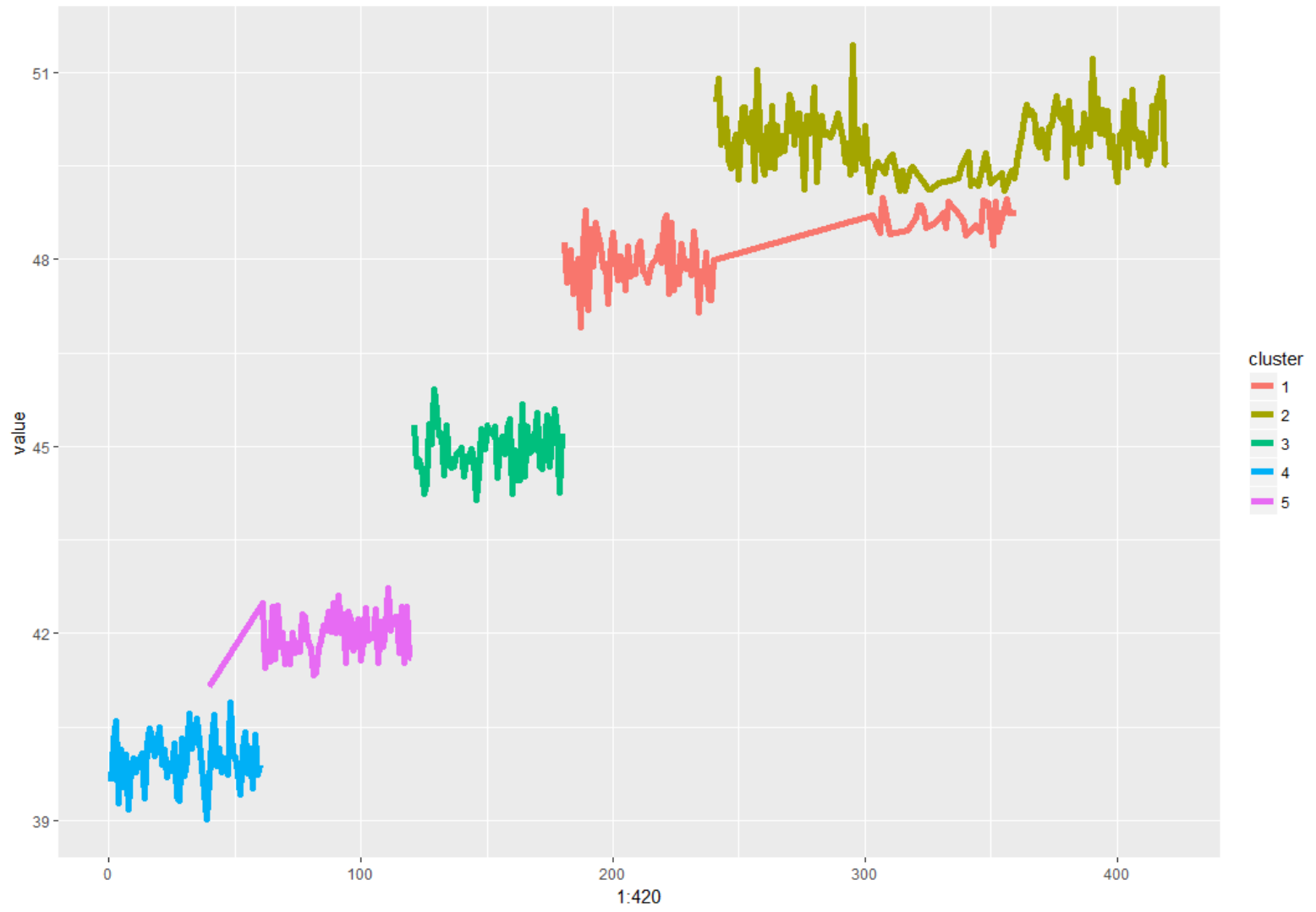
**Machine  
learning in  
PT**

**Recommendation  
for bottleneck  
detection**

**PT fairy ~~fails~~ tales**







# Machine learning in performance testing

- Using clustering in timeseries it is possible to detect bottlenecks-plateau, if plateau of resources usage appears before that biggest load in performance tests

**Continuous  
PT**

**Report  
Generation**

**Machine  
learning in  
PT**

**Recommendation  
for bottleneck  
detection**

**PT fairy ~~fails~~ tales**

# Performance testing fairy fails tales

- **Mailing from STAGE environment to real users**
- **Limit connections for STAGE environments**
- **Run performance test on TEST environments**

# Questions



# Presentation Here

