Hunting for hidden bugs: dealing with test flakiness in SKA control software

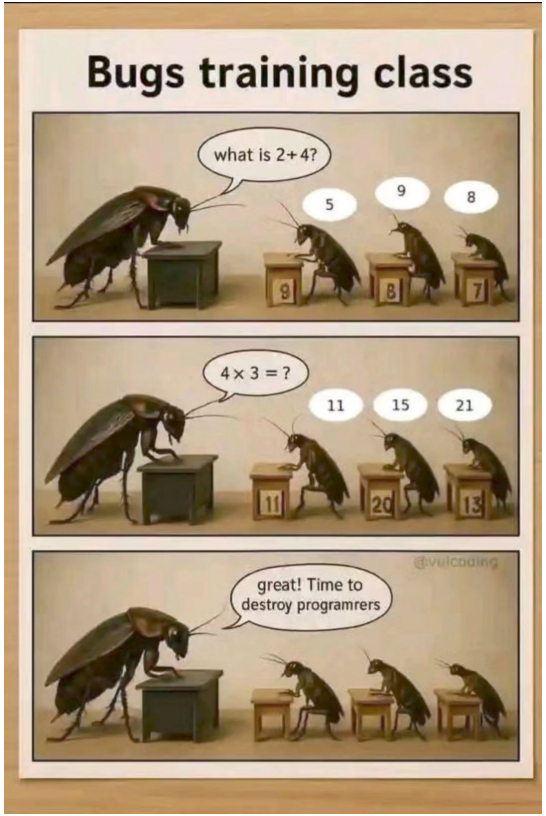
Gianluca Marotta, Carlo Baffa, Elisabetta Giani, Stefano Di Frischia, Ivana Novak, Martino Colciago, Emanuele Lena and Giorgio Brajnik

ICALEPCS 2025

Chicago, September 23, 2025

“Hunting for hidden bugs…”

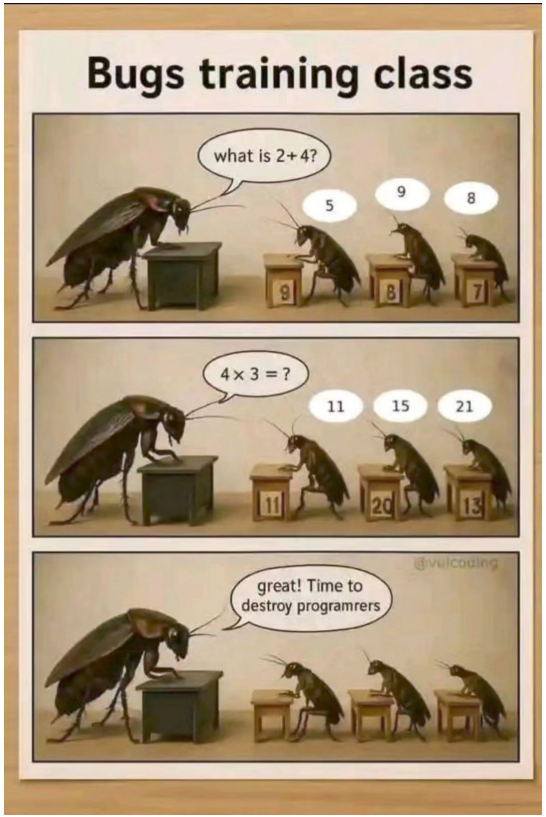
… what does it mean?



“Hunting for hidden bugs…”

… what does it mean?

1. All bugs are hidden



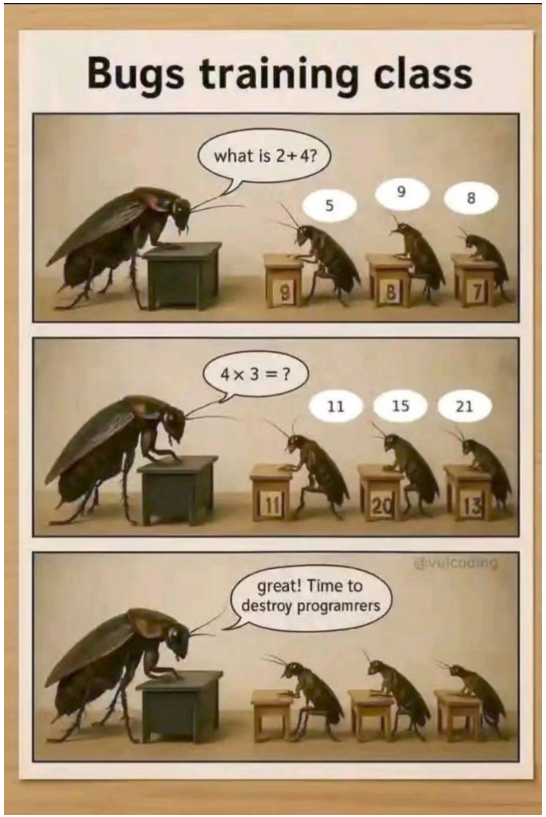
“Hunting for hidden bugs…”

… what does it mean?

1. All bugs are hidden
2. There is only <u>one way</u> to find out bugs

↓

running the code

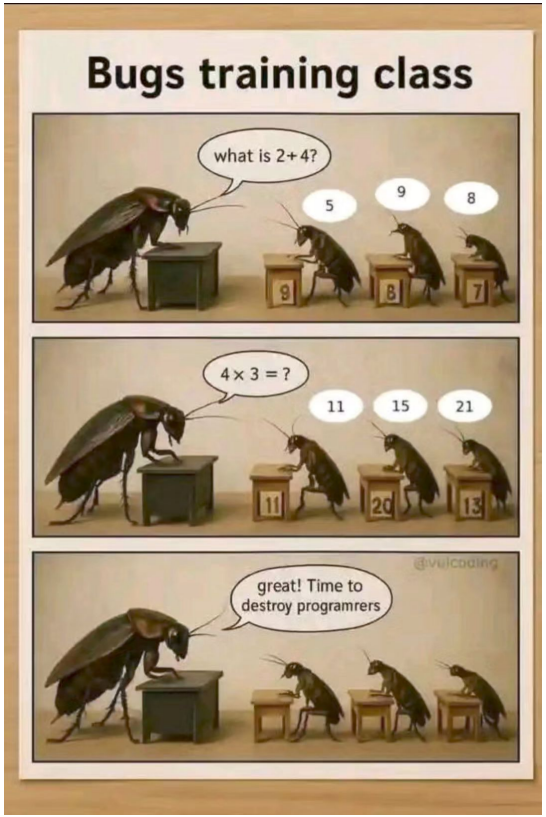


“Hunting for hidden bugs…”

… what does it mean?

1. All bugs are hidden
2. There is only <u>one way</u> to find out bugs

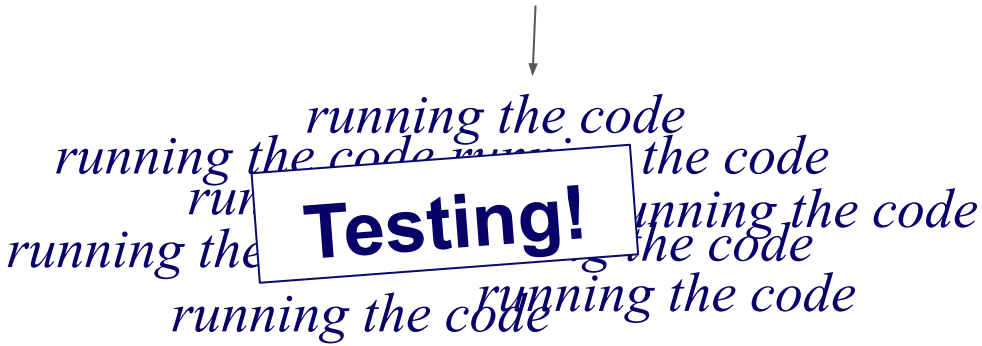
running the code
running the code running the code
running the code running the code
running the code running the code
running the code
running the code

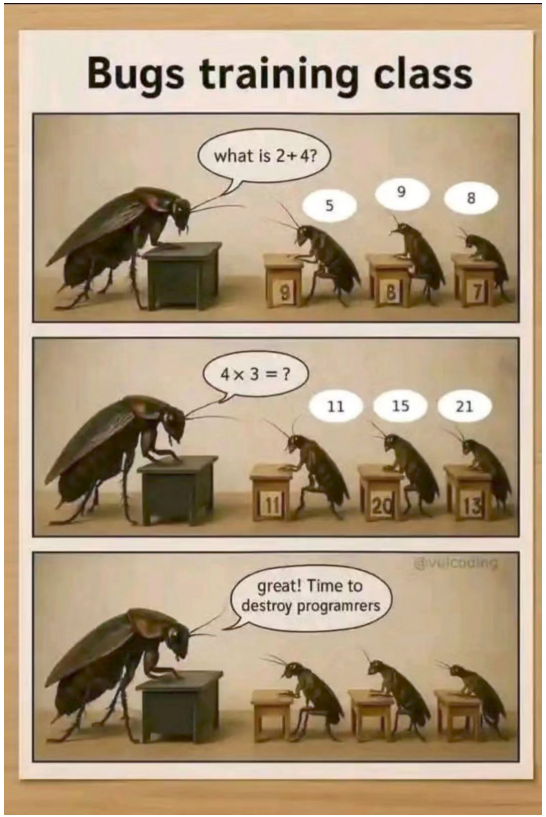


“Hunting for hidden bugs…”

… what does it mean?

1. All bugs are hidden
2. There is only <u>one way</u> to find out bugs





What… “testing”?

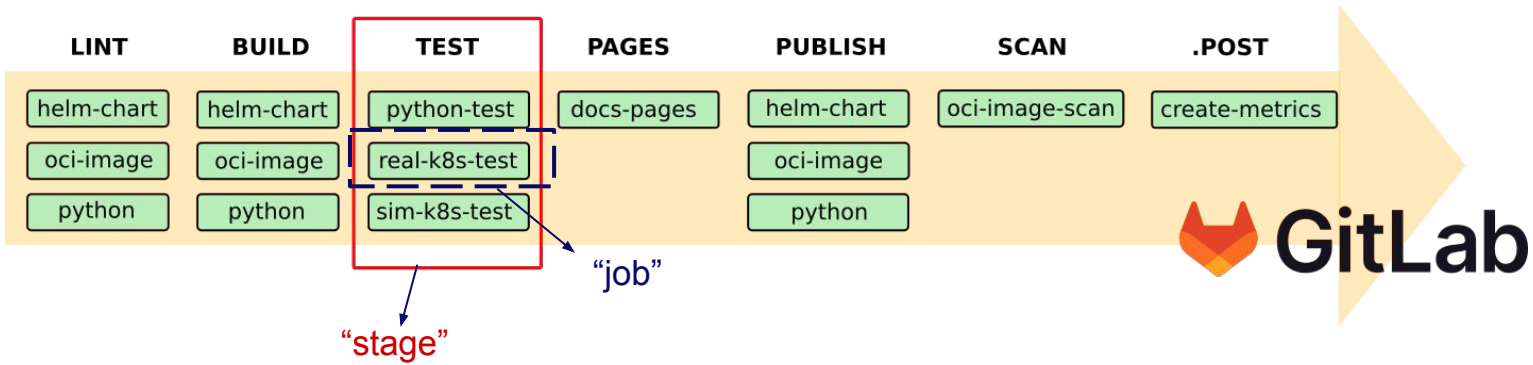
> “The purpose of testing is to increase **confidence** for stakeholders through **evidence**”
>
> Dan North “We need to talk about testing”

What… "testing"?

"The purpose of testing is to increase **confidence** for stakeholders through **evidence**"

Dan North "We need to talk about testing"

Developers ensure code quality via **automated tests** within *Continuous Integration/ Continuous Delivery (CI/CD)* pipelines



What if a test fails… sometimes?

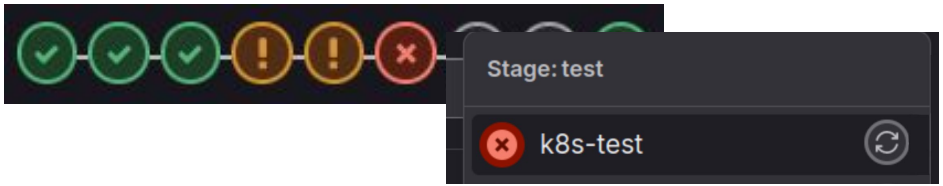
“**Regression testing** is performed at least every time code is committed on any branch in the source code repository. This should be ensured by the CI/CD pipeline”

SKAO Software Testing Policy and Strategy

What if a test fails… sometimes?

“**Regression testing** is performed at least every time code is committed on any branch in the source code repository. This should be ensured by the CI/CD pipeline”

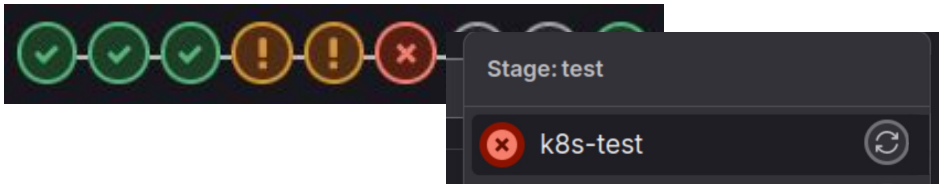
SKAO Software Testing Policy and Strategy



What if a test fails… sometimes?

“**Regression testing** is performed at least every time code is committed on any branch in the source code repository. This should be ensured by the CI/CD pipeline”

SKAO Software Testing Policy and Strategy



Two possible causes for “red pipeline”:

1) the change in the code disrupted an existing functionality
2) one or more tests has failed in a non-reproducible way

→ *“test **flakiness**”*

How to deal with test flakiness?

It seems that *few* solutions are possible:

1) block the development until the test has been fixed
2) log as bugs/failures and put in backlog
3) remove the flaky tests
4) retry until pipeline is green *(yes, we all do…)*

How to deal with test flakiness?

It seems that *few* solutions are possible:

1) ~~block the development until the test has been fixed~~
2) log as bugs/failures and put in backlog
3) remove the flaky tests
4) retry until pipeline is green *(yes, we all do…)*

"Flaky failures" are difficult to reproduce!

How to deal with test flakiness?

It seems that *few* solutions are possible:

1) ~~block the development until the test has been fixed~~
2) ~~log as bugs/failures and put in backlog~~ → "Flaky failures" are difficult to reproduce!
3) remove the flaky tests
4) retry until pipeline is green *(yes, we all do…)*

How to deal with test flakiness?

It seems that *few* solutions are possible:

1) ~~block the development until the test has been fixed~~
2) ~~log as bugs/failures and put in backlog~~ → "Flaky failures" are difficult to reproduce!
3) ~~remove the flaky tests~~ → This is *ignoring bad evidence!*
4) retry until pipeline is green *(yes, we all do…)*

How to deal with test flakiness?

It seems that *few* solutions are possible:

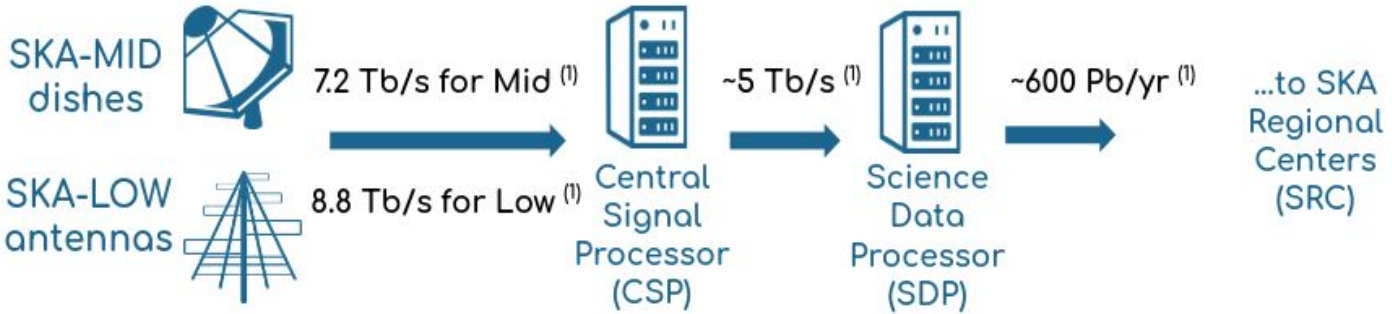
1) ~~block the development until the test has been fixed~~
2) ~~log as bugs/failures and put in backlog~~ → "Flaky failures" are difficult to reproduce!
3) ~~remove the flaky tests~~ → This is *ignoring bad evidence!*
4) retry until pipeline is green *(yes, we all do…)*

we decided to try a *new solution*…

collect and **analyze** the test results!

What are we testing?

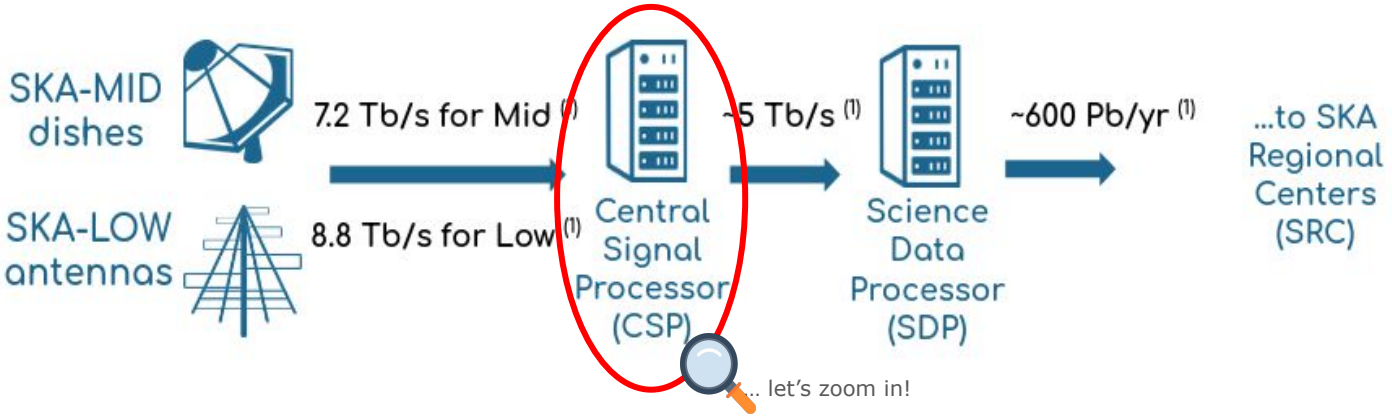
The Square Kilometer Array (SKA) is an international effort to construct the two *world's biggest radio telescopes*.



The case study: *Local Monitoring and Control* for the *Central Signal Processor* (CSP.LMC)

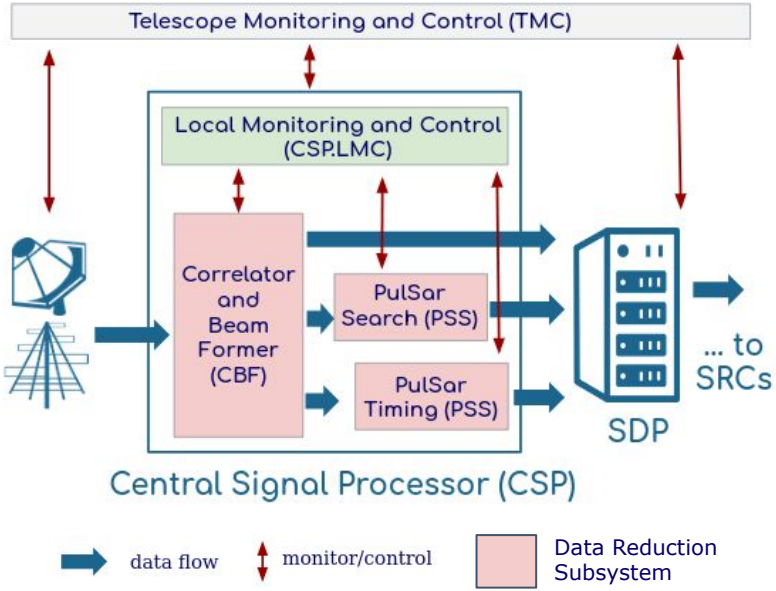
What are we testing?

The Square Kilometer Array (SKA) is an international effort to construct the two *world's biggest radio telescopes*.



The case study: *Local Monitoring and Control* for the *Central Signal Processor* (CSP.LMC)

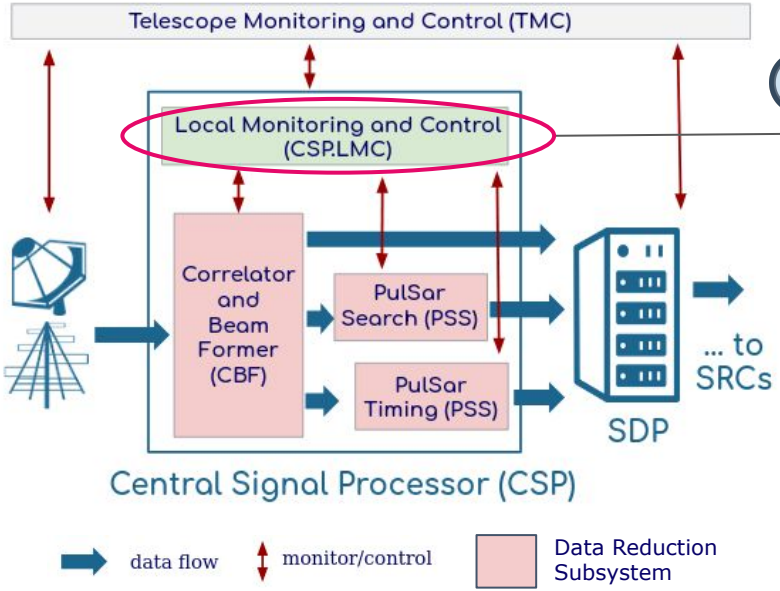
What are we testing?

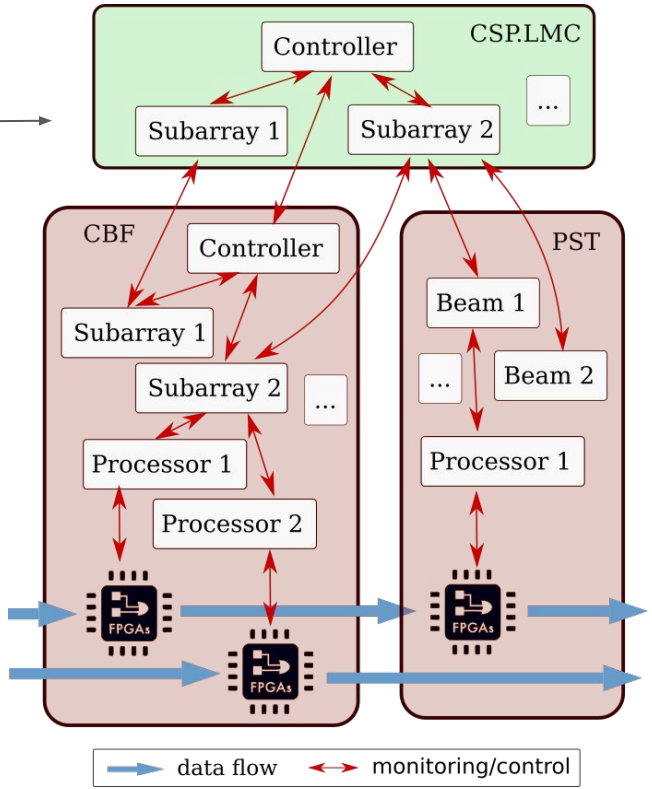


CSP.LMC provides the *interface* to TMC *without exposing CSP internal complexity.*

What are we testing?

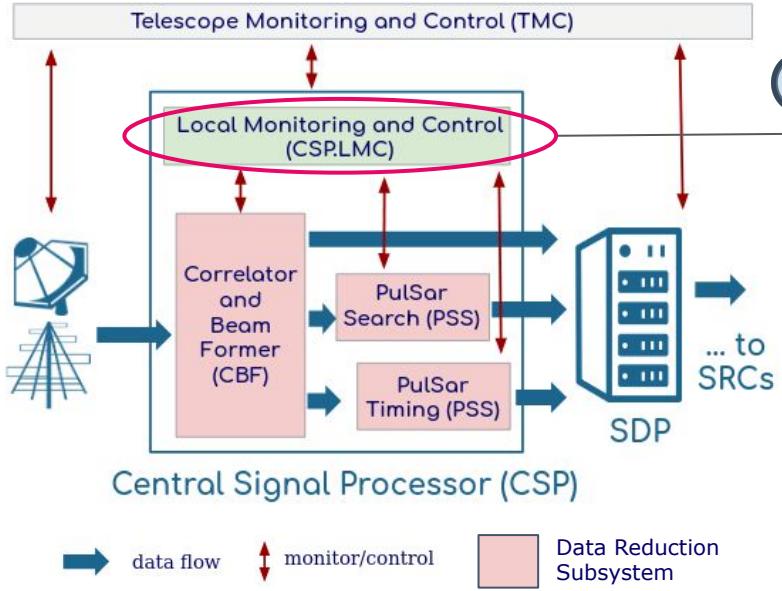
A *very simplified* view of the internal structure…





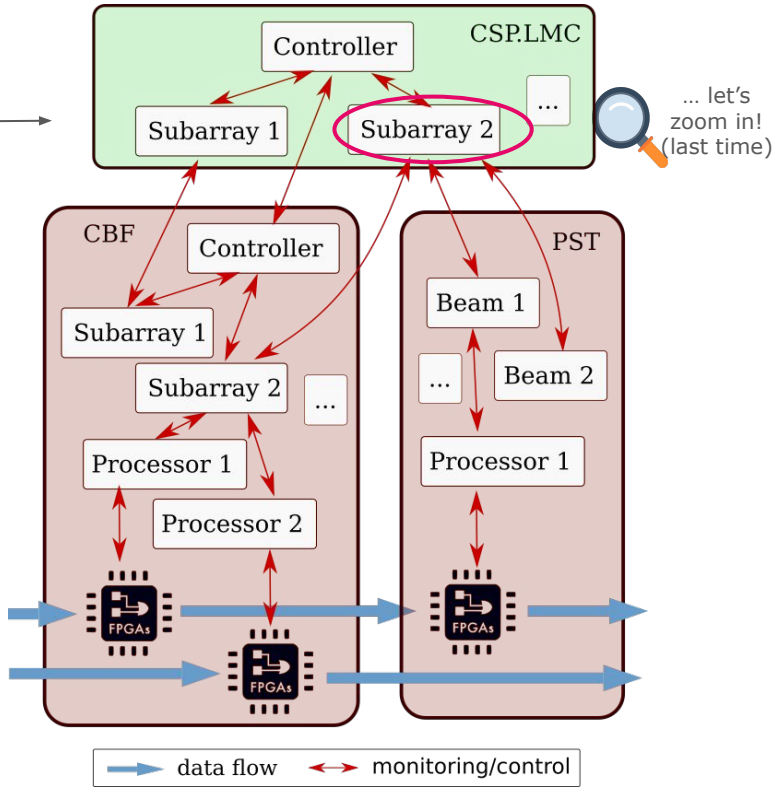
CSP.LMC provides the *interface* to TMC *without exposing CSP internal complexity.*

What are we testing?



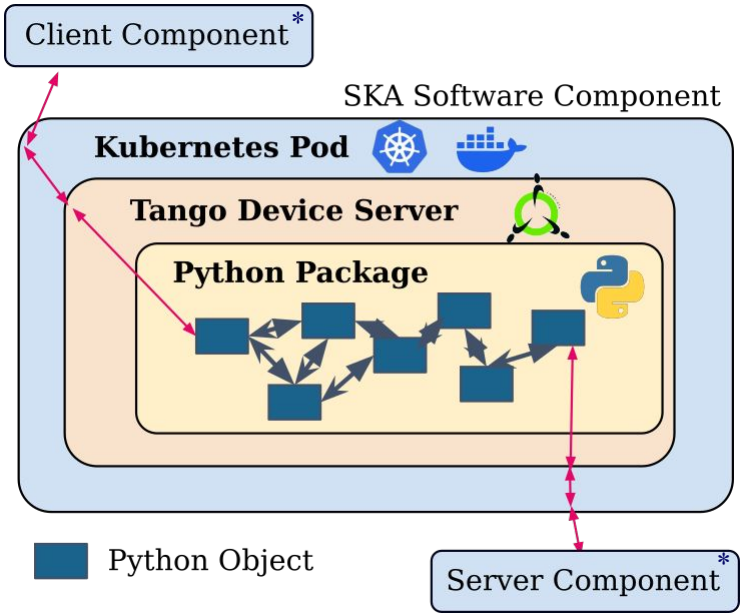
... let's zoom in! (again)

A *very simplified* view of the internal structure…



CSP.LMC provides the *interface* to TMC *without exposing CSP internal complexity.*

What are we testing?

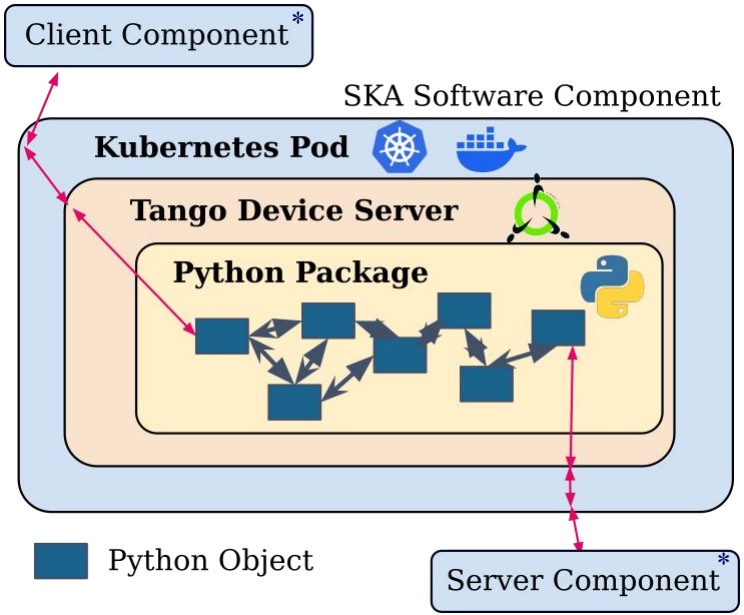


SKA Software component are multi-layered:

- **TANGO devices** written in **Python**
- **containerized** and deployed with **Kubernetes**

* for simplicity only one server/client is reported

What are we testing?



SKA Software component are multi-layered:

- **TANGO devices** written in **Python**
- **containerized** and deployed with **Kubernetes**

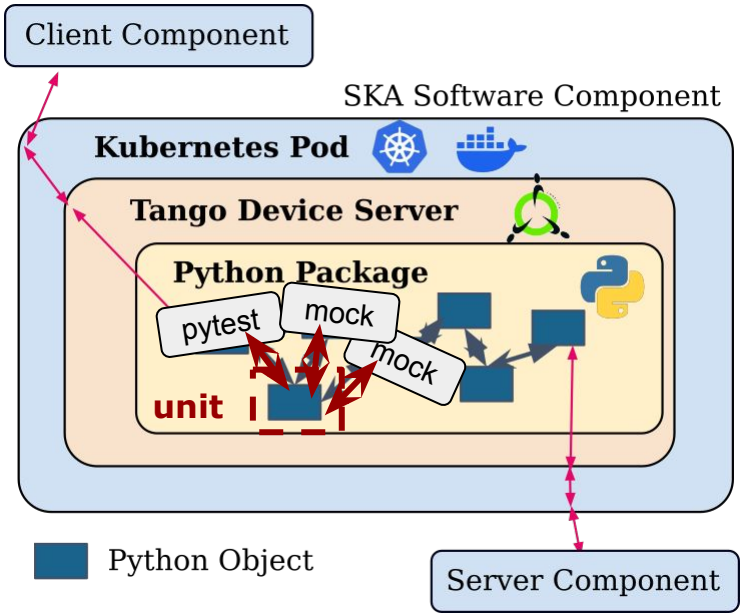
Could this structure be a source of *non-determinism* and *test flakiness?*

* for simplicity only one server/client is reported

How are we testing?

Tests are performed with a *multi-level strategy:*

- *unit* tests

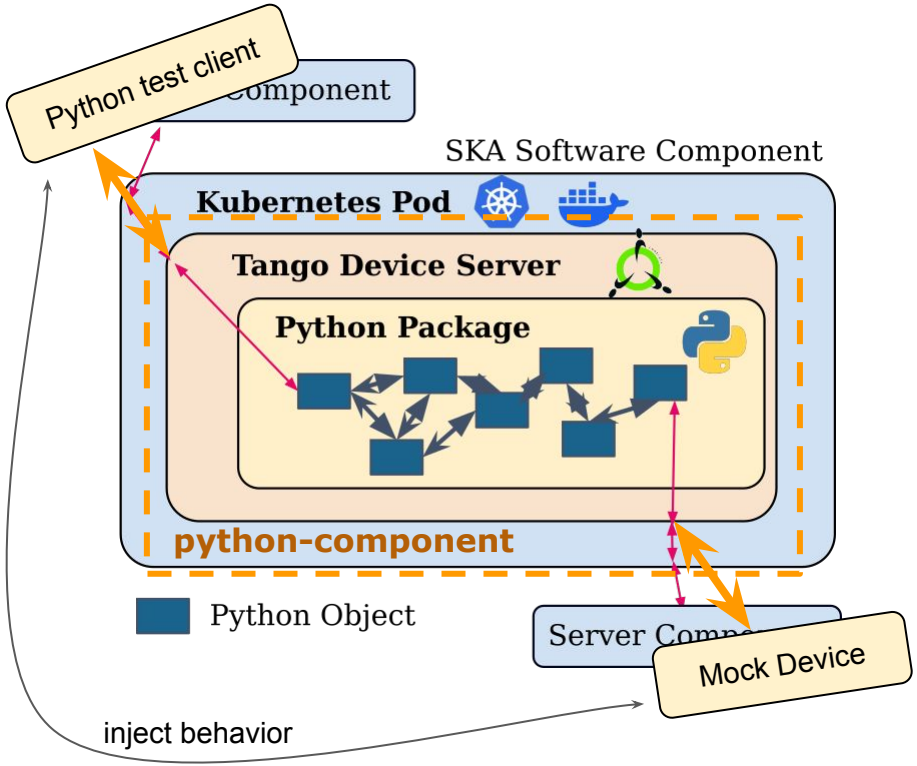


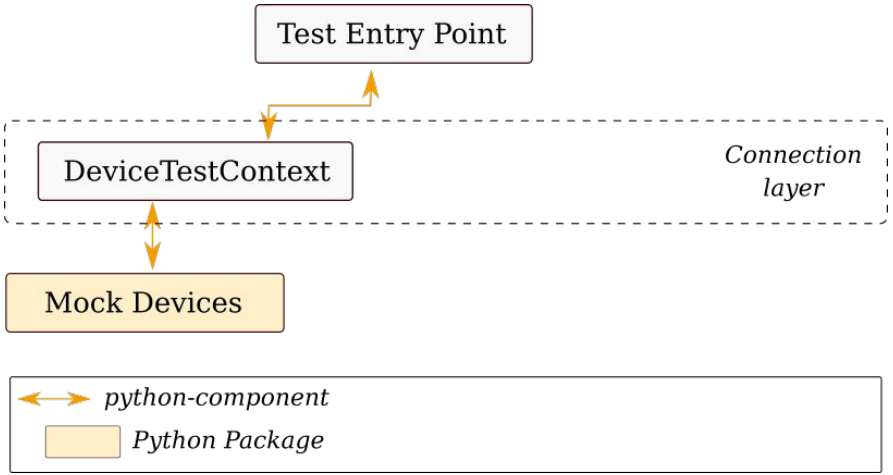
How are we testing?

The CSP.LMC Testing Strategy

Tests are performed with a *multi-level strategy:*

- *unit* tests
- *python-component* tests



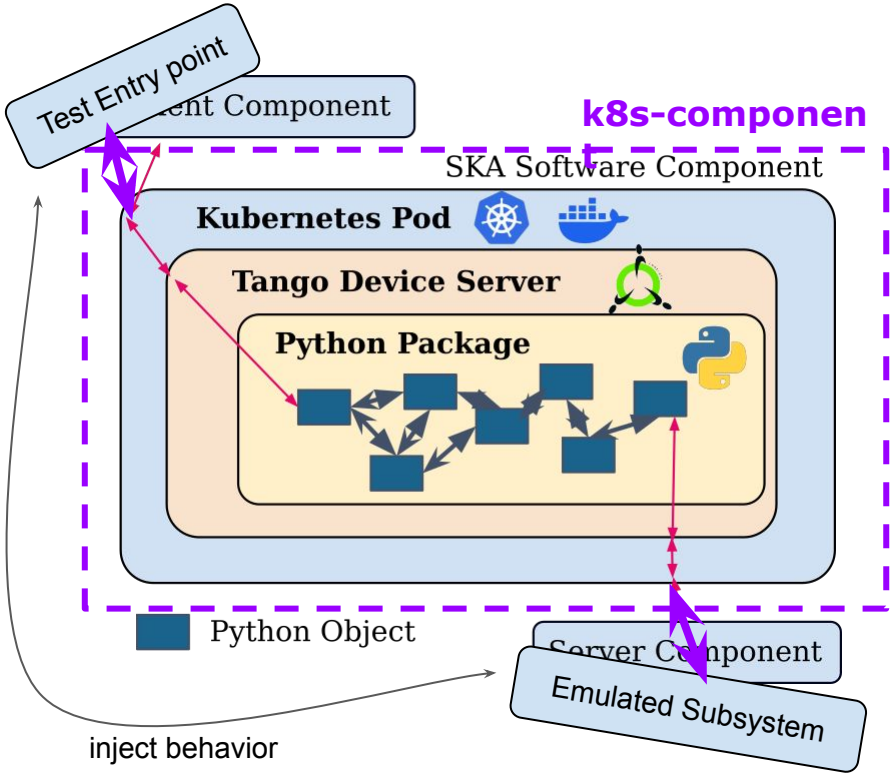


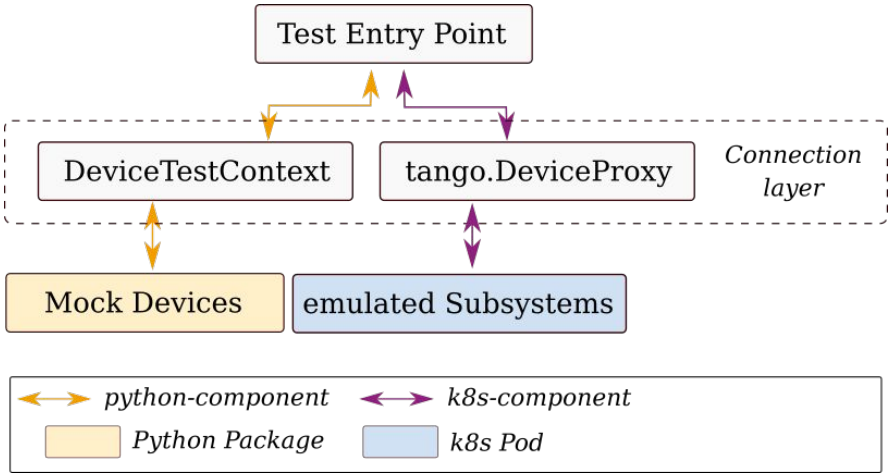
How are we testing?

The CSP.LMC Testing Strategy

Tests are performed with a *multi-level strategy:*

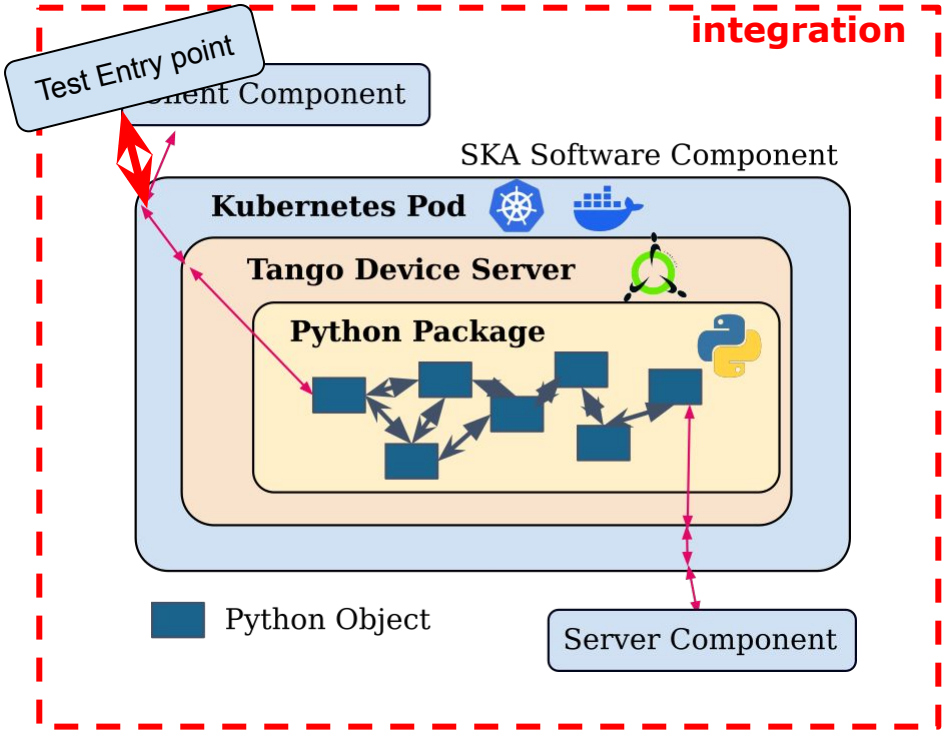
- *unit* tests
- *python-component* tests
- *k8s-component* tests





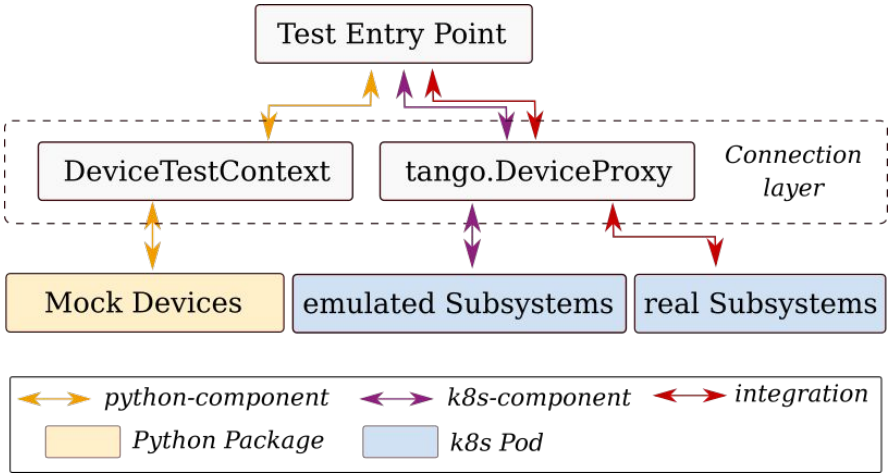
How are we testing?

The CSP.LMC Testing Strategy



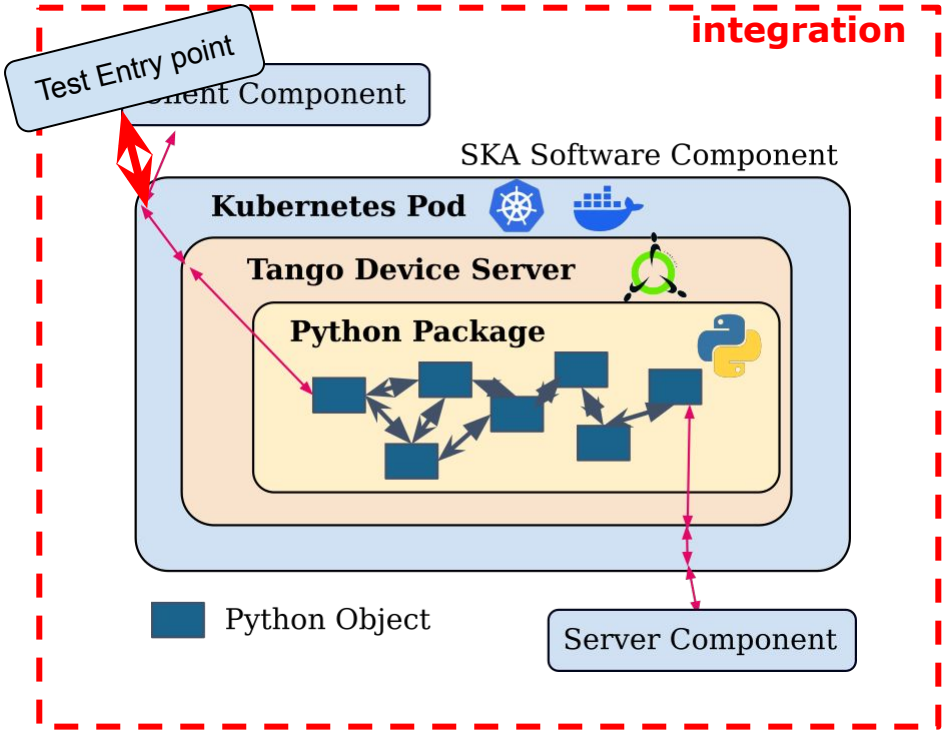
Tests are performed with a *multi-level strategy:*

- *unit* tests
- *python-component* tests
- *k8s-component* tests
- *integration* tests



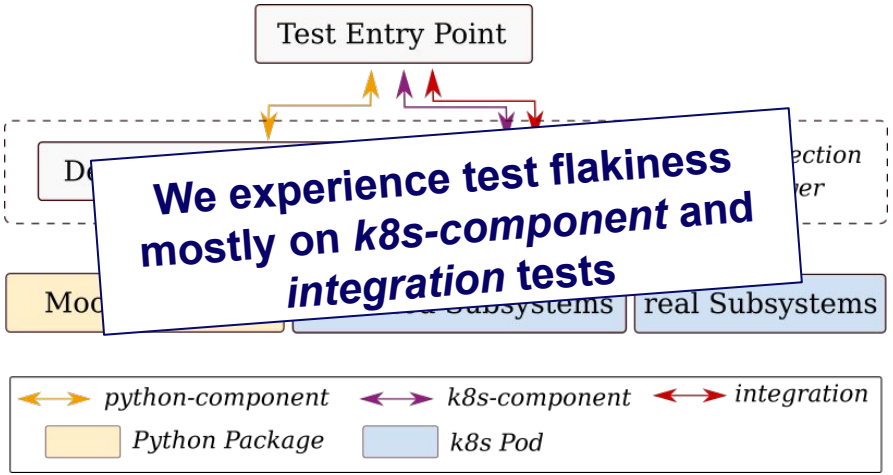
How are we testing?

The CSP.LMC Testing Strategy



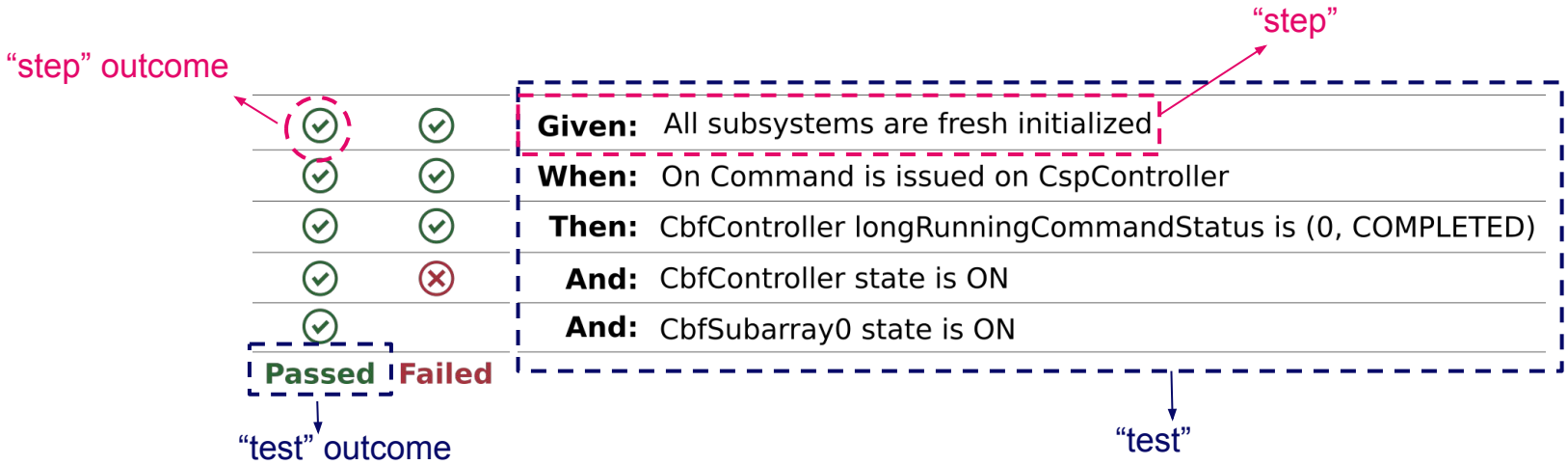
Tests are performed with a *multi-level strategy:*

- *unit* tests
- *python-component* tests
- *k8s-component* tests
- *integration* tests



How are we testing?

Integration and *component* tests follow the *Behaviour Driven Development (BDD)* approach. They are written in the *Gherkin* language.



Each "step" is translated to a specific Python function and can be utilised in different tests

How are we testing?

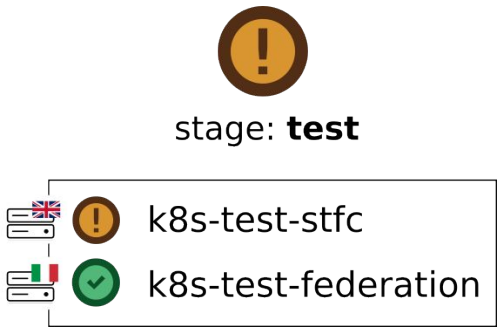
A real example of CSP.LMC Gherkin

decorators for internal SKA tracking

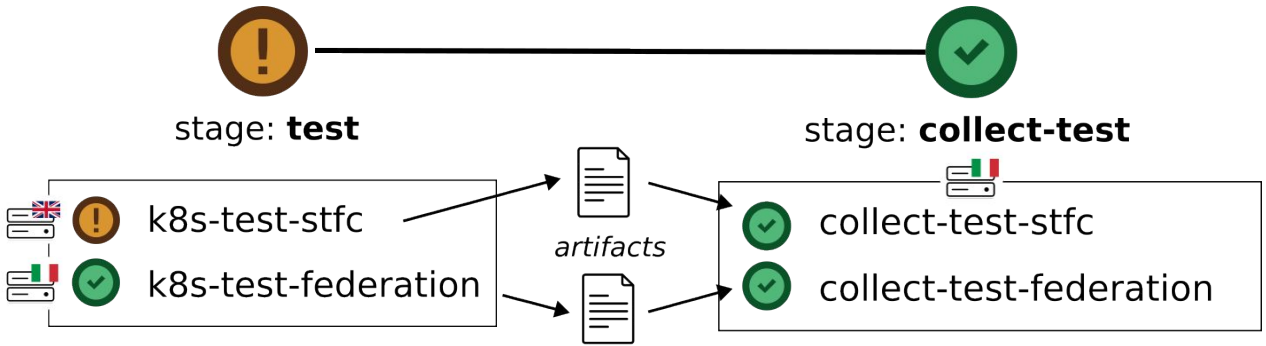
```
@XTP-77030 @L2-4884 @L2-4883 @L2-4844
  Scenario Outline: CSP.LMC Subarray is successfully configured (with PST Beam 01) and
transition to READY
      Given CspSubarray<sub_id> has assigned PstBeam01 and ObsState IDLE
      When Configure command is executed WITH SUCCESS on CspSubarray <sub_id>
      Then CspSubarray<sub_id> ObsState is READY
      And CspSubarray<sub_id> ObsMode is (PULSAR_TIMING)
      And CbfSubarray<sub_id> ObsState is READY
      And PstBeam01 ObsState is READY

  Examples:
      | sub_id |
      | 01     |
      | 02     |
```

"Examples" are used to parametrize the scenario

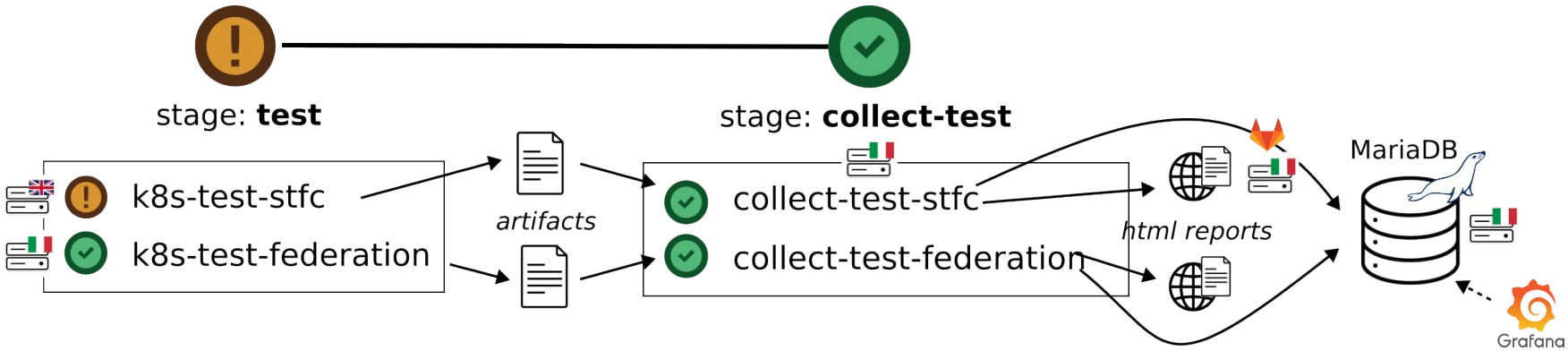
Collecting tests result using Gitlab CI/CD



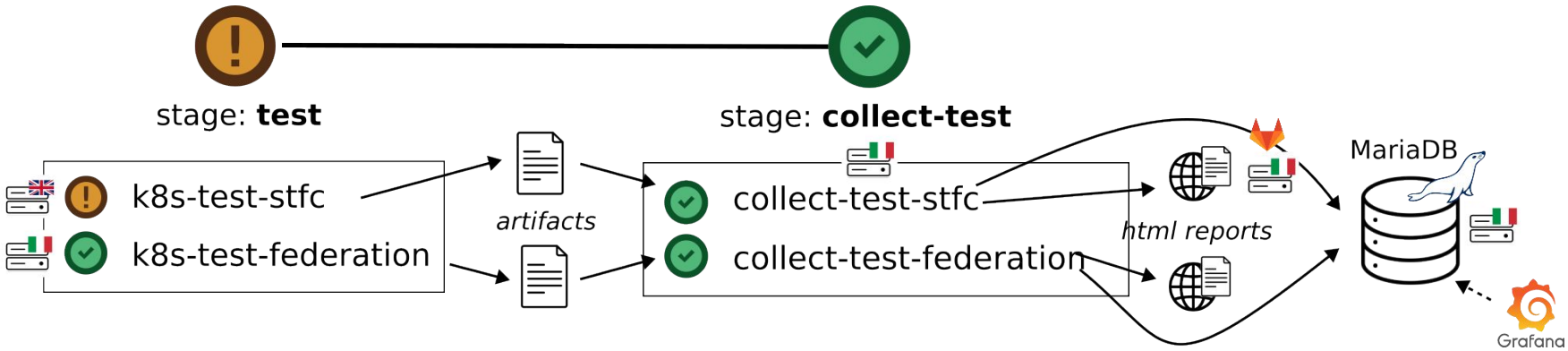
Collecting tests result using Gitlab CI/CD



Collecting tests result using Gitlab CI/CD



Collecting tests result using Gitlab CI/CD



- **Periodic pipelines** every 6 hours (4 times a day)
- Only **two stages** in periodic pipelines (no build – **only released versions**)
- k8s–tests run on **two** similar **facilities**
- Jobs in stage “**test**” are **allowed to fail**
- Artifacts coming from test execution are **parsed** and **collected** by stage **“collect-test”***

Database structure 1/2

Pipeline jobs

job_id	pipeline_id	facility	job_time	subsystem_type
1040503	187283746	STFC	2025-09-08 18:03:32	real
…	…	…	…	…

Product versions

job_id	product_name	version
1040503	ska-csp-lmc-low	1.0.1
1040503	ska-low-cbf	1.0.3
1040503	ska-pst	1.0.0
…	…	…

Test executions

id	job_id	test_name	test_start_time	setup_outcome	setup_duration
98	1040503	test_on_command	2025-09-08 18:03:32	passed	0.0012
…	…	…	…	…	…

teardown_outcome	teardown_duration	test_duration	test_outcome	error_traceback
passed	0.865	2,52	passed	…
…	…	…	…	…

Database structure 1/2

Pipeline jobs

job_id	pipeline_id	facility	job_time	subsystem_type
1040503	187283746	STFC	2025-09-08 18:03:32	real
…	…	…	…	…

Product versions

job_id	product_name	version
1040503	ska-csp-lmc-low	1.0.1
1040503	ska-low-cbf	1.0.3
1040503	ska-pst	1.0.0
…	…	…

Test executions

id	job_id	test_name	test_start_time	setup_outcome	setup_duration
98	1040503	test_on_command	2025-09-08 18:03:32	passed	0.0012
…	…	…	…	…	…

teardown_outcome	teardown_duration	test_duration	test_outcome	error_traceback
passed	0.865	2,52	passed	…
…	…	…	…	…

Database structure 2/2

Test executions

id	job_id	test_name	test_start_time	setup_outcome	setup_duration	…
98	1040503	test_on_command	2025-09-08 18:03:32	passed	0.0012	…
…	…	…	…	…	…	…

Test steps

id	test_id	step_index	step_keyword	step_name	step_outcome	step_duration
489	98	0	Given	CSP is initialized	passed	0.604
…	…	…	…	…	…	

Test logs

id	test_id	log_index	time	log_level	code_source	thread	message
768	98	0	2025-09-08 18:03:32	INFO	base_wrapper.py:135	MainThread	low-csp/subarray/01 available …
…	…	…	…	…	…		

Database structure 2/2

Test executions

id	job_id	test_name	test_start_time	setup_outcome	setup_duration	…
98	1040503	test_on_command	2025-09-08 18:03:32	passed	0.0012	…
…	…	…	…	…	…	…

Test steps

id	test_id	step_index	step_keyword	step_name	step_outcome	step_duration
489	98	0	Given	CSP is initialized	passed	0.604
…	…	…	…	…	…	

Test logs

id	test_id	log_index	time	log_level	code_source	thread	message
768	98	0	2025-09-08 18:03:32	INFO	base_wrapper.py:135	MainThread	low-csp/subarray/01 available …
…	…	…	…	…	…		

The Global Fail Rate

The Global Fail Rate (GFR) is *a* **metric** that reports, for each version, the **combined quality** of **control software** and **testware.**

$$GFR = \frac{\#\ tests\ failed}{\#\ tests\ executed}$$

(LOW) CSP.LMC version	GFR
0.13.0	23.3 %
0.14.0	6.7 %
1.0.1	**0.86%**

(MID) CSP.LMC version	GFR
1.1.0	**5.99 %**

Addressing test flakiness - the hunt for hidden bugs

The best approach is to do the exploration of the DB as a **collective effort** during **Explore Test DB sessions**

- To be planned every 2/3 weeks
- Iterative process to improve the collecting of data results
- Items in backlog to lower the GFR

we take advantage of **Grafana dashboards** and **HTML Reports**

Grafana Dashboard - the hunt for hidden bugs

db: ska_csp_lmc_mid_test_records | scenario: csplmc subarray and fsp capability reports correct attributes after configuration 1 and 2 fsp no pstbeams

example: 01-"Configure CBF 2ndFSP ver5.json"-'mid csp cbf/fsp/02'-2-("ON",)-("UNKNOWN",)-"CORR", "CORR", "CORR", "CORR", "PST BF", "PST BF", "PST BE", "ON", "DISABLE", "DIS...

csp_version: 1.1.0 | step_name: | test_id: 27700

Global Fail Rate over CSP.LMC versions -- click on link for details

CSP.LMC version	total_tests	failed_tests	Global Fail Rate (%)
1.1.0	20605	1329	6.45

Facility Fail Rate over CSP.LMC versions -- click on link for details

CSP.LMC version	facility	total_tests	failed_tests	Global Fail Rate (%)
1.1.0	INAF-Federation	11440	374	3.27
1.1.0	STFC	9165	955	10.4

Fail Rate over test sessions

seq_id	job_id	Global Fail Rate (%)	go_to_report
1	10415614384	6.15	http://10.17.17.13:310...
2	10415891958	3.08	http://10.17.17.13:310...

Most Failed Scenario Outline -- click on link to show failures per examples

scenario_outline	failed_count
csplmc subarray and fsp capability reports c...	166
csplmc subarray successfully release all reso...	90
csplmc subarray and vcc capability reports c...	84

Most Failed Scenario examples (per selected Scenario Outline)

scenario_example	failed_count
01-"Configure CBF 2ndFSP ver5.json"-'mid c...	57
01-"Configure CBF vis 5.json"-'mid csp cbf/f...	55

Most Failed Steps (per selected Scenario)

step_name	failed_count
CspSubarray01 has assigned no PstBeams a...	50
Configure command is executed with SUCCE...	6

Absolute Most Failed Steps

step_name	failed_count
CspSubarray01 has been configured with no ...	243

HTML Reports - the hunt for hidden bugs

CSP.LMC MID Test Session Report

Test Session Summary

Test Job Id:	10892166623
Facility:	STFC
Execution Time:	2025-08-02 04:11:47
Total Executed Tests:	65
Total Failures:	28
Local Fail Rate:	43.077
Global Fail Rate:	5.860

System Under Test

Product Name	Version
ska-csp-lmc-mid	1.1.0
ska-mid-cbf-tdc-mcs	1.3.0
ska-pst	1.0.1

Pod Logs Explorer

Select Pod: ds-fsp-fsp-01-0-logs.html

Go to Logs

Show All Tests | Go to Benchmark report | Go to Campaign report

Failed Tests

Scenario	Example	Start Time	Outcome	Duration	Setup	Setup Time	Teardown	Teardown Time	Details	Error Traceback
csplmc successfully send abort command to subarray after scan start	01	2025-08-02 04:15:32	failed	17.394	passed	0.0033	passed	30.0131	Logs Steps	Info
csplmc successfully send restart command after an abort operation	01	2025-08-02 04:16:20	failed	57.326	passed	0.00309	failed	0.00947	Logs Steps	Info

Conclusions and future work

At the *present*:

- the “hunts” are based on a **data mining approach**
- **CSP.LMC as test case**, to be possibly extended to other SKA components
- **collaborative explore db sessions** with UIs (Grafana/Reports) proved valuable
- **GFR as key indicator** of combined code & test quality

Conclusions and future work

At the *present*:

- the “hunts” are based on a **data mining approach**
- **CSP.LMC as test case**, to be possibly extended to other SKA components
- **collaborative explore db sessions** with UIs (Grafana/Reports) proved valuable
- **GFR as key indicator** of combined code & test quality

In the *future*:

- integrate **resource consumption** (CPU/Memory/Network) into collection
- more extensive **code benchmarking**
- **new tools** for analysis (Jupyter/Pandas/ML?)

Thank you for your attention!

ICALEPCS 2025

Chicago, September 23, 2025

For further information, questions, feedback, collaborations, please don't hesitate to contact:

gianluca.marotta@inaf.it

Any questions?

INAF ISTITUTO NAZIONALE DI ASTROFISICA

COSYLAB Advancing humanity. Engineering remarkable.

IDS Design Coaching for Excellence

SKAO