INAF
ISTITUTO NAZIONALE
DI ASTROFISICA

SKAO

TETIS
TEchnologies for Telescopes and
Instrument control Software

USC VIII

SKA Observatory Management and Control Software: the INAF contribution

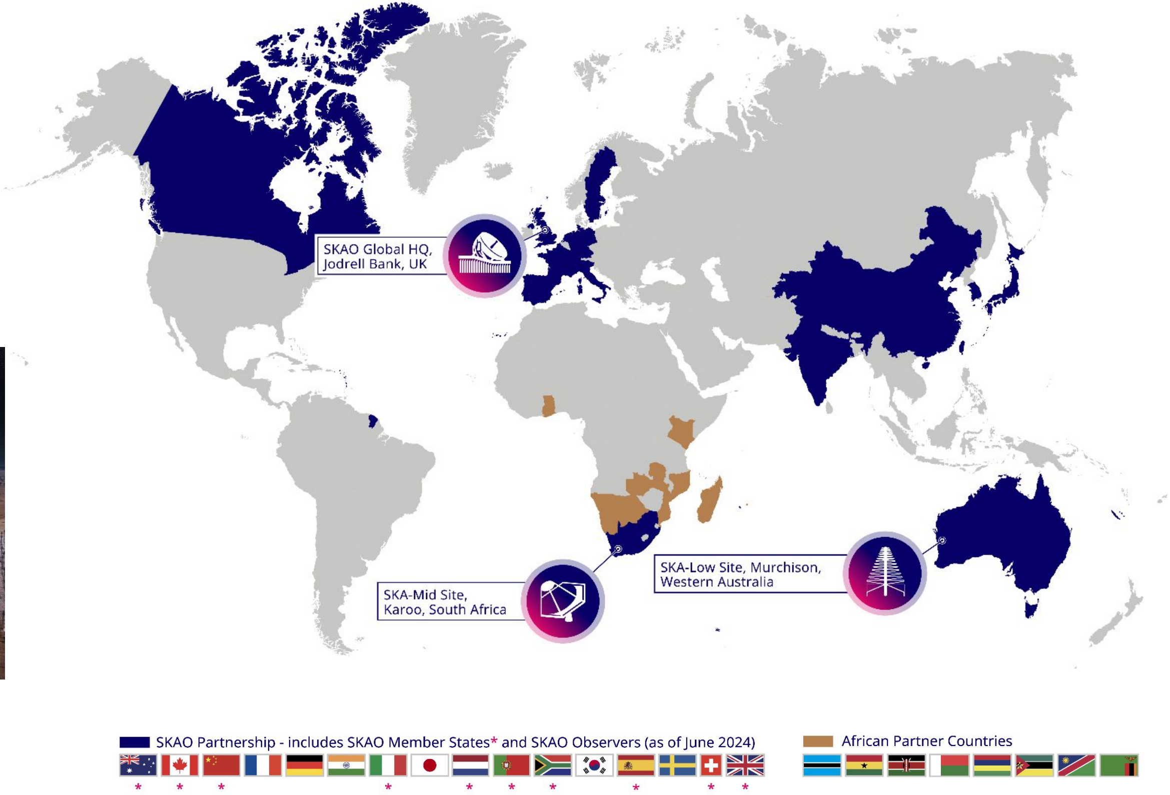
Gianluca Marotta, Valentina Alberti, Carlo Baffa, Matteo Canzari, Matteo Di Carlo,
Stefano Di Frischia, Elisabetta Giani, Teresa Pulvirenti and Mauro Dolci

USCVIII - General Assembly

Galzignano, 16 October 2024

The SKA Project

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


SKAO Global HQ, Jodrell Bank, UK

SKA-Mid Site, Karoo, South Africa

SKA-Low Site, Murchison, Western Australia

■ SKAO Partnership - includes SKAO Member States* and SKAO Observers (as of June 2024)

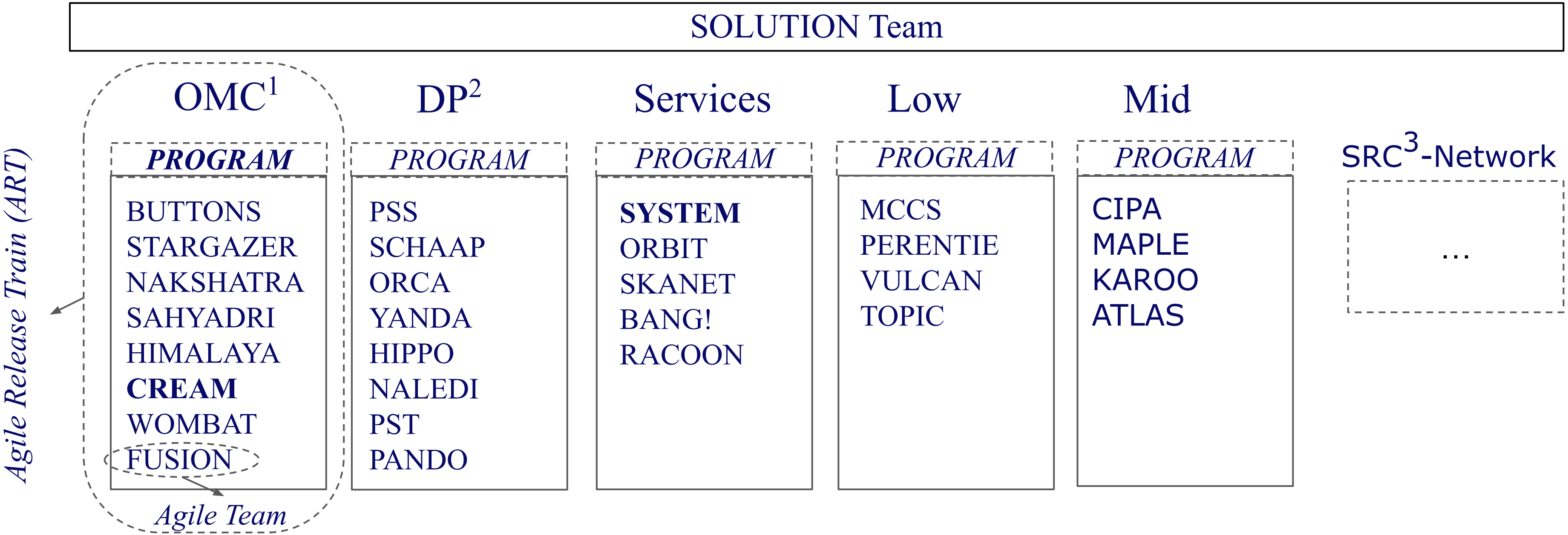
■ African Partner Countries

The SKA Project: the software engineering group

The SKA Project: the software engineering group

- more than 200 people involved from 15 different countries
- organized with Scaled Agile Framework (SAFe)

SAFe® PROVIDED BY SCALED AGILE

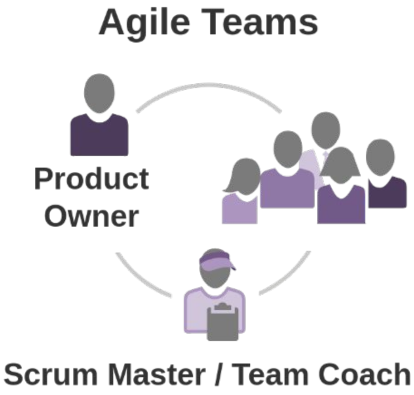


SOLUTION Team

OMC[1]	DP[2]	Services	Low	Mid	SRC[3]-Network
PROGRAM	*PROGRAM*	*PROGRAM*	*PROGRAM*	*PROGRAM*	
BUTTONS	PSS	**SYSTEM**	MCCS	CIPA	...
STARGAZER	SCHAAP	ORBIT	PERENTIE	MAPLE	
NAKSHATRA	ORCA	SKANET	VULCAN	KAROO	
SAHYADRI	YANDA	BANG!	TOPIC	ATLAS	
HIMALAYA	HIPPO	RACOON			
CREAM	NALEDI				
WOMBAT	PST				
FUSION	PANDO				

[1] *Observatory Management and Control* [2] *Data Processing* [2] *SKA Regional Centers*

The Agile Team



© Scaled Agile, Inc.

SAFe® PROVIDED BY SCALED AGILE

- each Agile Team is responsible for one or more SW Product;
- code is developed iteratively;
- Team plans the work within the ART[1] every 3 months (*Program Iteration - PI*);
- Team revise its PI plan every 2 weeks (*sprint*);
- the *Scrum Master* “helps implement and maintain Agile practices, [...] optimizes and improves team performance”[2]
- the *Product Owner* “contributes to the Vision and roadmap [...] and prioritize the team’s work”[2].

! more about it: **tomorrow**’s training session on “Agile Framework” conducted by Valentina Alberti and Matteo Di Carlo (**h14:00**)

[1] *Agile Release Train* [2] *https://scaledagileframework.com/agile-teams/*

The INAF Group

Program Team

OATs	Valentina Alberti	ex CREAM Scrum Master -> UI/UX[1] Expert (OMC Program)	
OAA	Carlo Baffa	CREAM Product Owner	
	Elisabetta Giani	CREAM Developer	CSP[2] Local Monitoring and Control
	Gianluca Marotta	CREAM Developer	
OAAb	Stefano Di Frischia	CREAM Developer	
	Matteo Canzari	CREAM Developer	TARANTA TANGO ON WEB
	Matteo Di Carlo	SYSTEM Developer	System Infrastructure
	Mauro Dolci	Scientific Responsible	
	Teresa Pulvirenti	Contract Management	

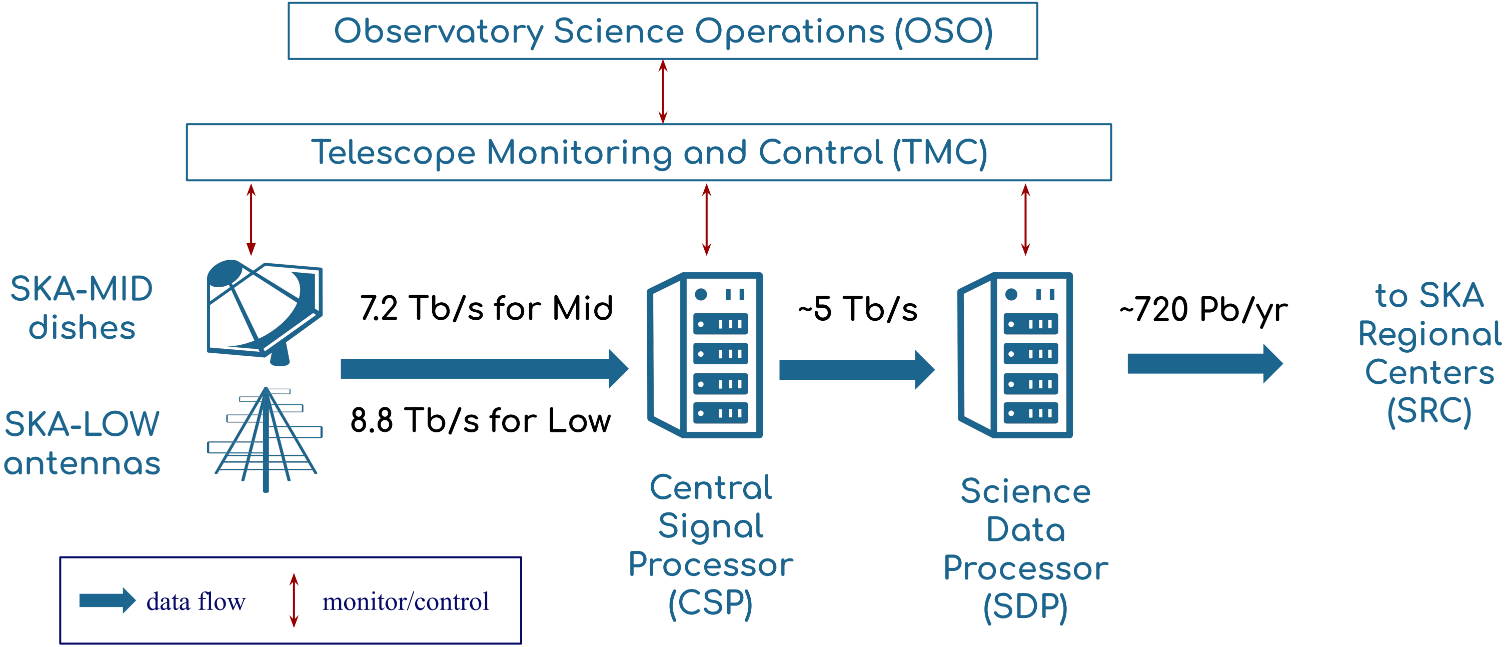
N.B. CREAM and SYSTEM are composed by more people coming from other countries and companies/institutions

[1] User Interfaces/User eXperience

[2] Central Signal Processor

The SKA Data Flow and Control System

… a very simplified view



Observatory Science Operations (OSO)

Telescope Monitoring and Control (TMC)

SKA-MID dishes

SKA-LOW antennas

7.2 Tb/s for Mid

8.8 Tb/s for Low

Central Signal Processor (CSP)

~5 Tb/s

Science Data Processor (SDP)

~720 Pb/yr

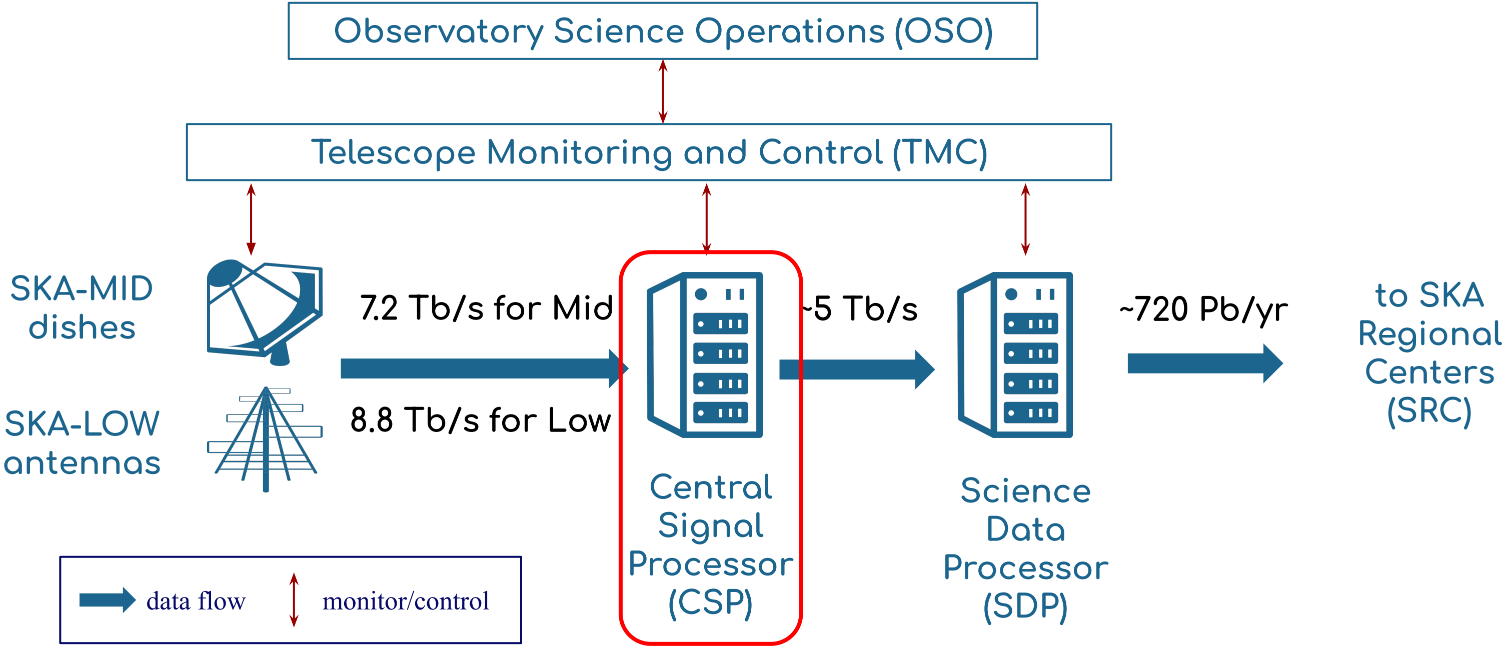
to SKA Regional Centers (SRC)

data flow

monitor/control

The SKA Data Flow and Control System

… a very simplified view



Observatory Science Operations (OSO)

Telescope Monitoring and Control (TMC)

SKA-MID dishes

SKA-LOW antennas

7.2 Tb/s for Mid

8.8 Tb/s for Low

Central Signal Processor (CSP)

~5 Tb/s

Science Data Processor (SDP)

~720 Pb/yr

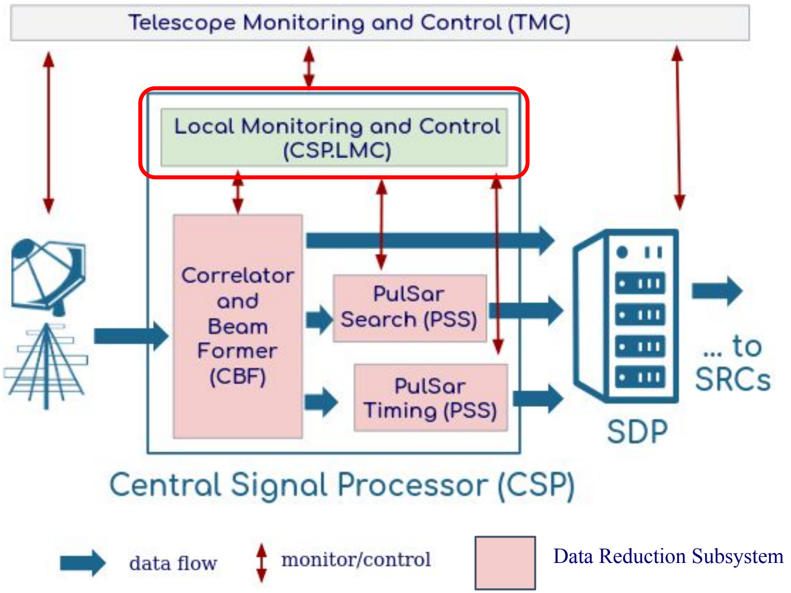
to SKA Regional Centers (SRC)

data flow

monitor/control

… let's zoom in!

The Central Signal Processor

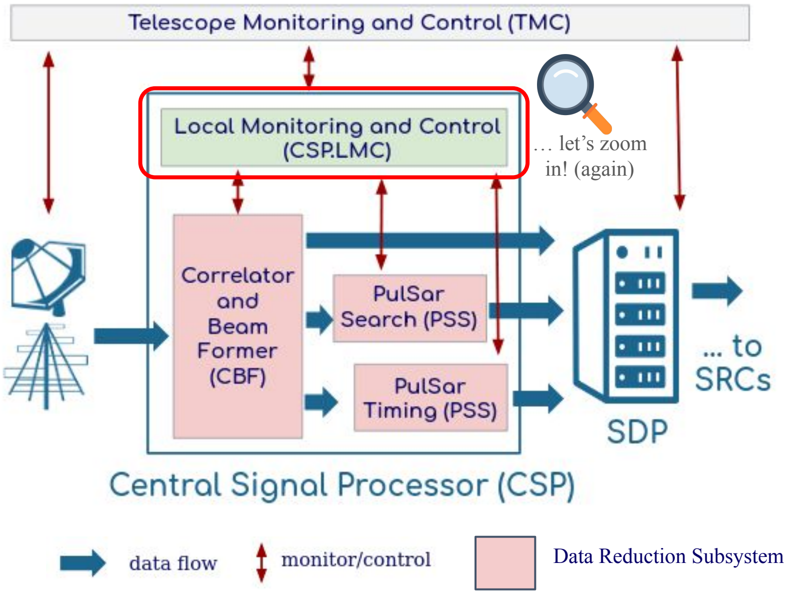

Telescope Monitoring and Control (TMC)
Local Monitoring and Control (CSP.LMC)
Correlator and Beam Former (CBF)
PulSar Search (PSS)
PulSar Timing (PSS)
SDP
... to SRCs
Central Signal Processor (CSP)
data flow
monitor/control
Data Reduction Subsystem

CSP is composed of three main subsystems:

- the **Correlator and Beam Former (CBF),** to creates the visibilities and the data beams;
- the **Pulsar Search (PSS)**, to perform an all-sky pulsar search survey;
- the **Pulsar Timing (PST),** to measures the frequency of the pulsar candidates

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

The Central Signal Processor


Telescope Monitoring and Control (TMC)
Local Monitoring and Control (CSP.LMC)
… let's zoom in! (again)
Correlator and Beam Former (CBF)
PulSar Search (PSS)
PulSar Timing (PSS)
SDP
... to SRCs
Central Signal Processor (CSP)
data flow
monitor/control
Data Reduction Subsystem

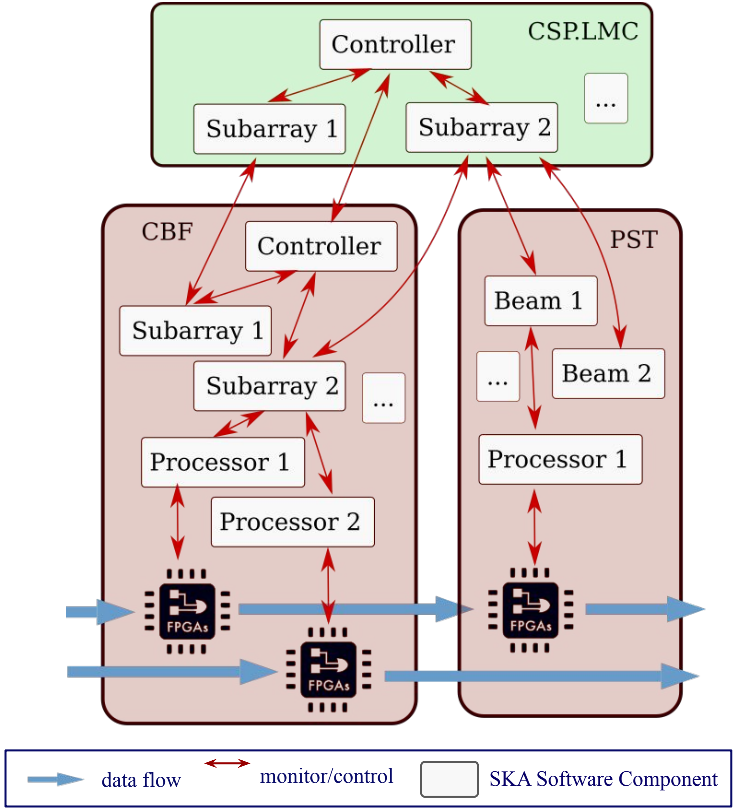
CSP is composed of three main subsystems:

- the **Correlator and Beam Former (CBF),** to creates the visibilities and the data beams;
- the **Pulsar Search (PSS)**, to perform an all-sky pulsar search survey;
- the **Pulsar Timing (PST),** to measures the frequency of the pulsar candidates

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

The CSP Local Monitoring and Control (CSP.LMC)

… a very simplified view

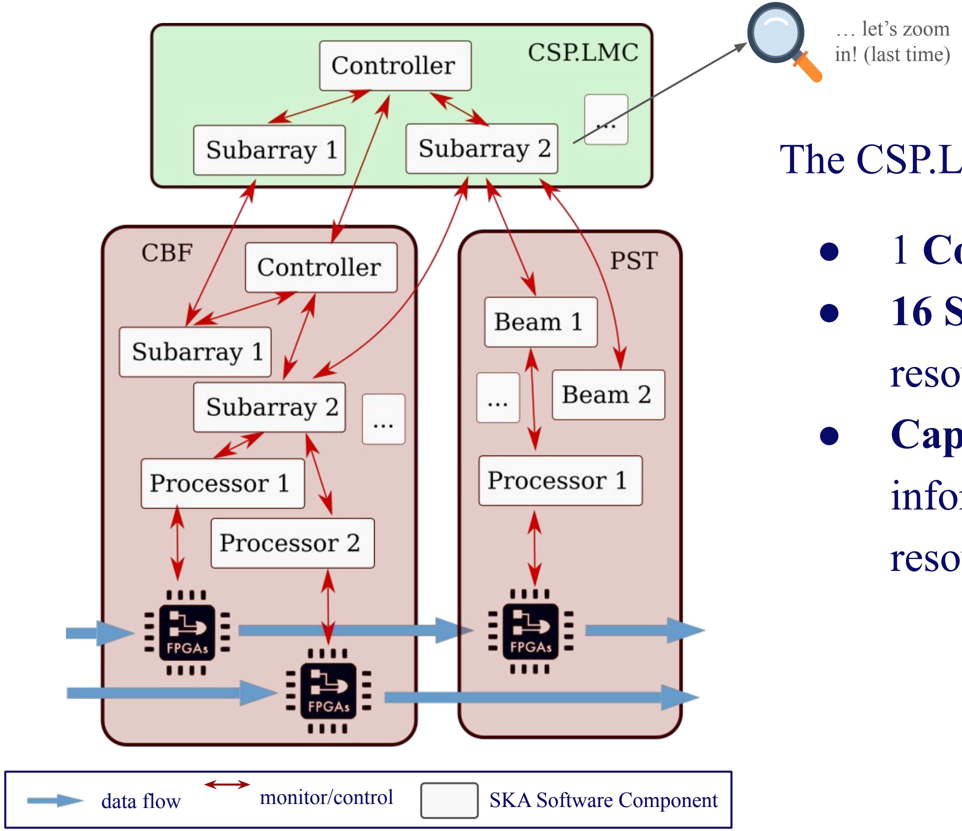


The CSP.LMC is composed by:

- 1 **Controller,** i.e. the primary point of access for CSP
- **16 Subarray**, representing subsets of the telescope resources that can be used for one observation
- **Capability** devices, apt to monitor and report to TMC information and statistical data about specific CSP resources (e.g. CBF processors, PST Beams)

The CSP Local Monitoring and Control (CSP.LMC)

… a very simplified view



… let's zoom in! (last time)

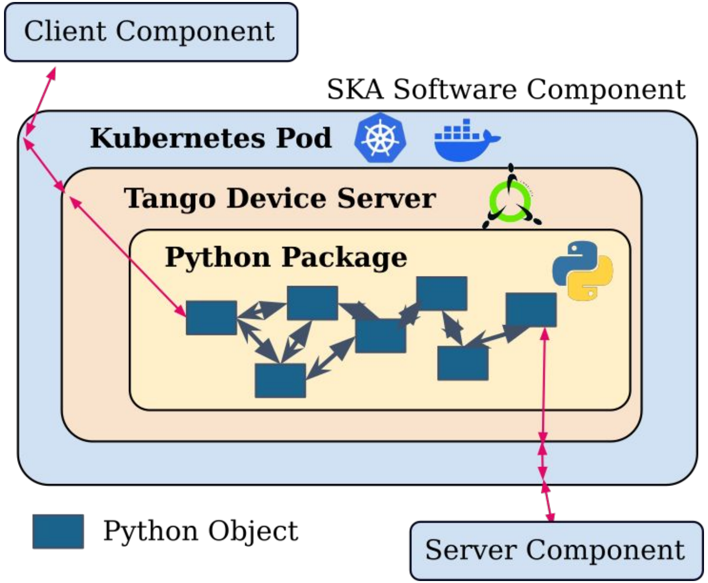
The CSP.LMC is composed by:

- 1 **Controller,** i.e. the primary point of access for CSP
- **16 Subarray**, representing subsets of the telescope resources that can be used for one observation
- **Capability** devices, apt to monitor and report to TMC information and statistical data about specific CSP resources (e.g. CBF processors, PST Beams)

CSP Local Monitoring and Control

The SKA Software Component

Software is **executed by** a **Tango Device Server** process, that runs in a **Docker container** that is **orchestrated** in a cluster **by Kubernetes**



TANGO: “a free open source device-oriented controls toolkit for controlling any kind of hardware or software and building SCADA[1] systems”[2].

both SKAO and INAF are consortium members!

docker: “is an open platform for developing, shipping, and running applications [...] enables you to separate your applications from your infrastructure[3]”

kubernetes: “is a portable, extensible, open source platform for managing containerized workloads and services …[4]”

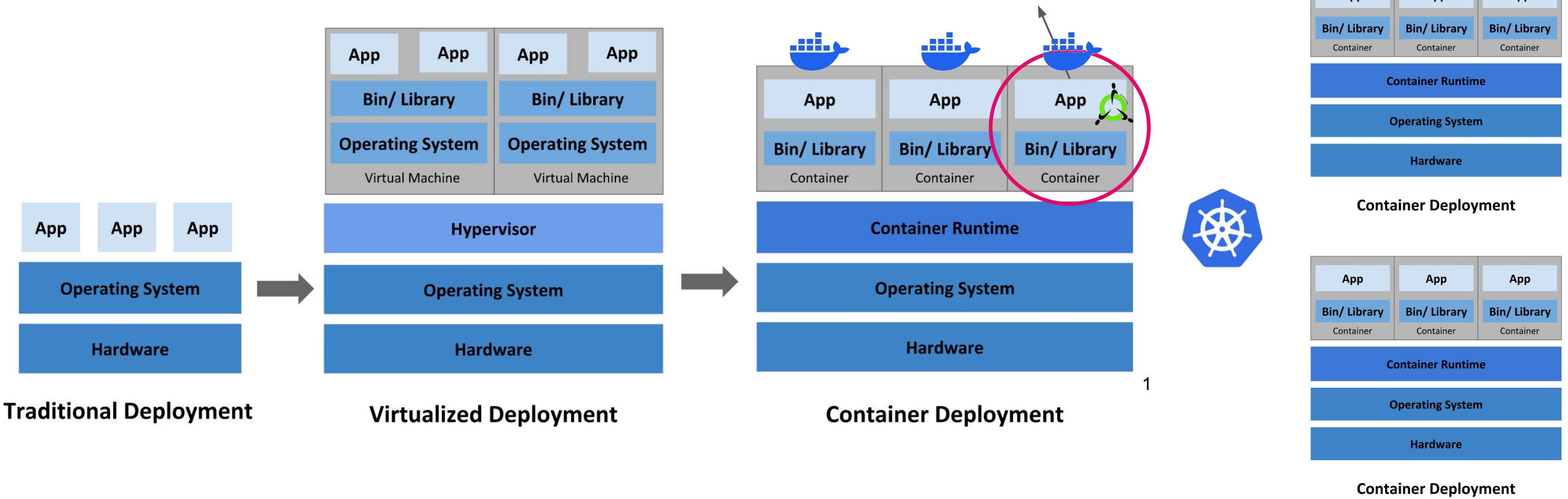
* for simplicity only one server/client is reported

[1] Supervisory Control And Data Acquisition [2] https://www.tango-controls.org/ [3] https://docs.docker.com [4] https://kubernetes.io/docs

Docker and Kubernetes

…a small digression

SKA Software Component



Kubernetes:

- establishes the network between containers,
- allows shared storage,
- manage secrets,
- distributes workloads in cluster
- is self-healing,
- ensure scalability

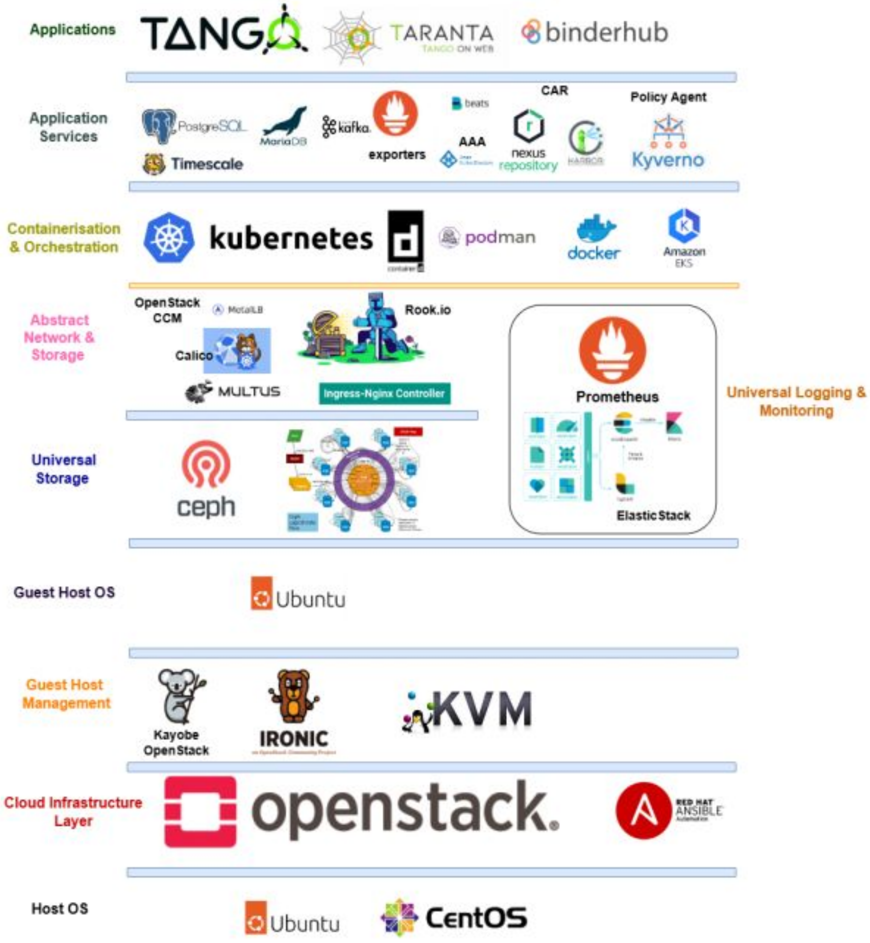
[1] https://kubernetes.io/docs

The System Infrastructure

SKA Control Software is meant to be deployed with a large use of **cloud-native technologies**

The infrastructure is developed and maintained by System Team

Tools for *deploying*, *testing* and *monitoring* are provided to developers



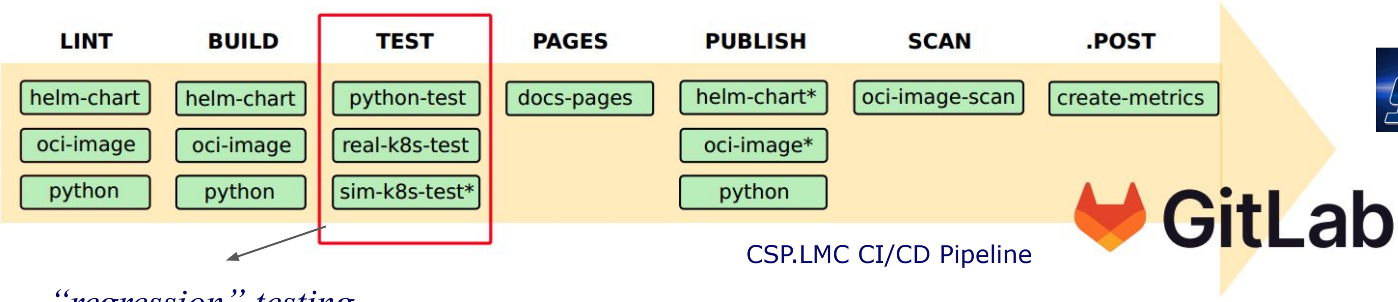
M.Di Carlo et al, “Monitoring the SKA Infrastructure for CICD”, *Proceedings of the 19th ICALEPCS* (2023)
M.Di Carlo et al, “SKA Tango Operator”, *Proceedings of the 19th ICALEPCS* (2023)

Continuous Integration/Continuous Delivery/Deployment

Continuous Integration/Continuous Delivery/Deployment (CI/CD) refers to development practices:

- single source repository for each component;
- automated build;
- ˆautomated testing;
- ˆ**every commit** should build on an integration machine (with tango/kubernetes)

CI/CD practices are ensured by the use of **Gitlab pipelines,** based on System Team templates.



LINT	BUILD	TEST	PAGES	PUBLISH	SCAN	.POST
helm-chart	helm-chart	python-test	docs-pages	helm-chart*	oci-image-scan	create-metrics
oci-image	oci-image	real-k8s-test		oci-image*		
python	python	sim-k8s-test*		python		

CSP.LMC CI/CD Pipeline

"regression" testing

M.Di Carlo et al, "CI-CD practices at SKA", Software and Cyberinfrastructure for Astronomy VII; 1218903 (2022)

Testing the SKA Software

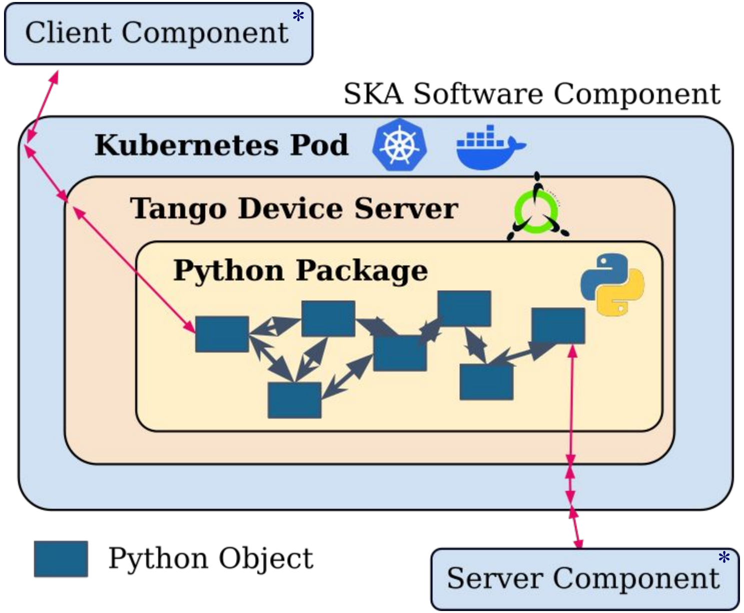
“[Testing is] the process [...] to determine that they [software products] **satisfy specified requirements**, to demonstrate that **they are fit for purpose** and to **detect bugs**.”[1]

- Individual teams are responsible for a specific software component quality and testing strategy
- Verification Tests based on requirements are done by AIV teams
- A *Testing Community of Practice* gather developers from different teams to share knowledge and practices

Testing can represent a **considerable amount of time** in developing a SKA Software Component (about 50% in CSP.LMC experience)

[1] SKAO Software Testing Policy and Strategy - https://developer.skao.int/

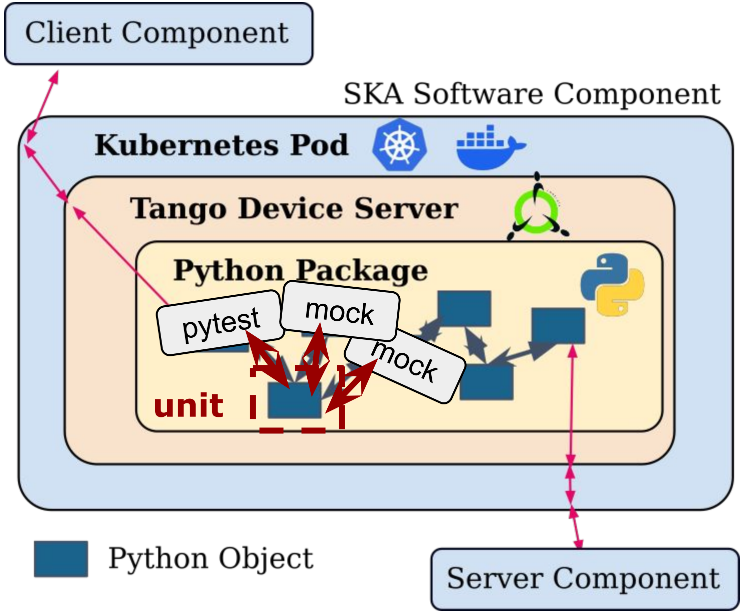
Testing strategy of CSP.LMC



* for simplicity only one server/client is reported

G.Marotta et al, “Strategy and Tools to Test Software in the SKA Project: The CSP. LMC Case”, *Proceedings of the 19th ICALEPCS* (2023)

Testing strategy of CSP.LMC



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

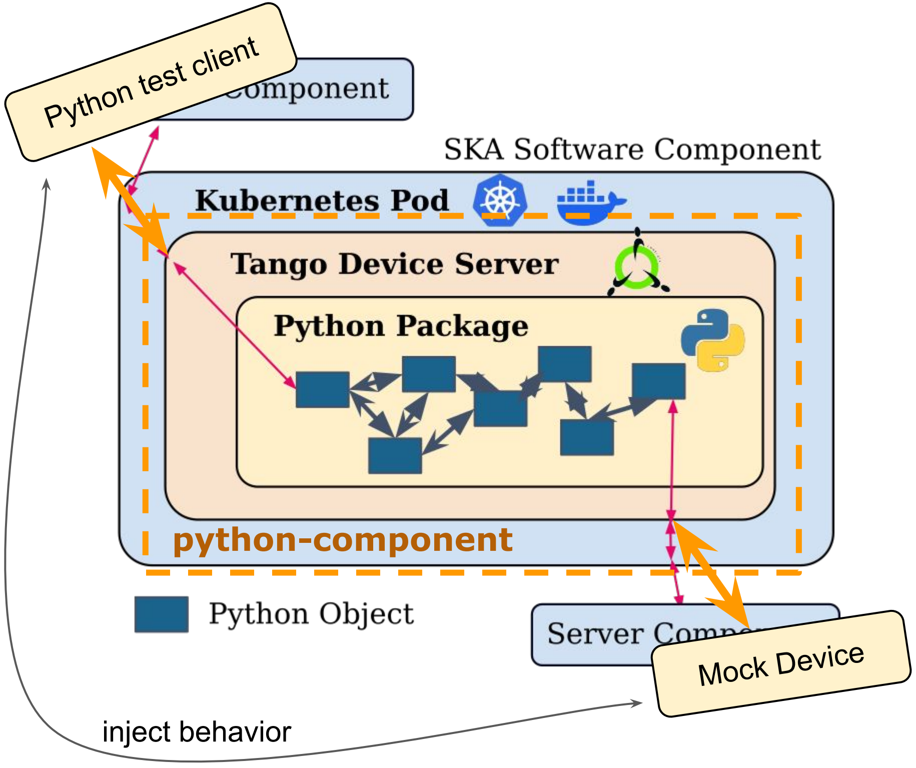
- *unit* tests

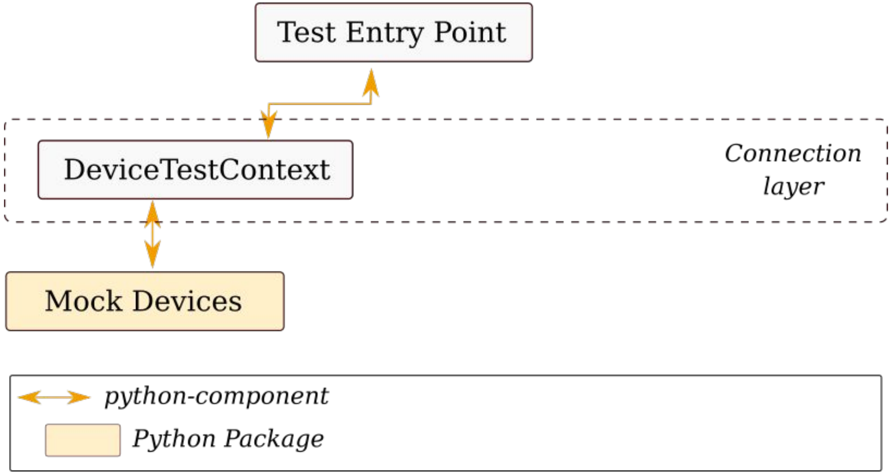
G.Marotta et al, “Strategy and Tools to Test Software in the SKA Project: The CSP. LMC Case”, *Proceedings of the 19th ICALEPCS* (2023)

Testing strategy of CSP.LMC

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

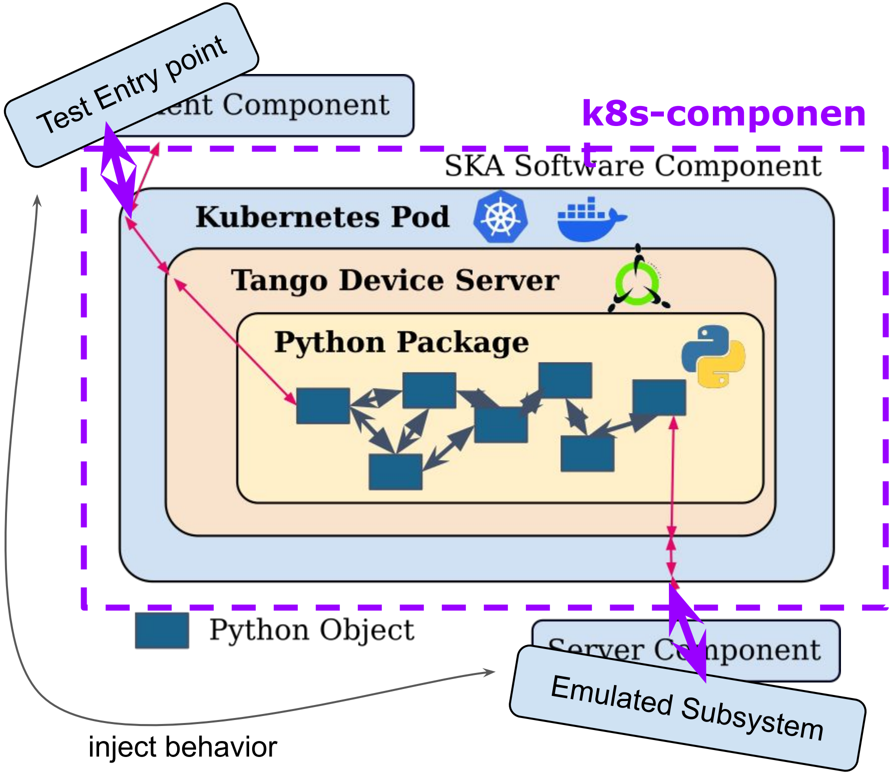
- *unit* tests
- *python-component* tests





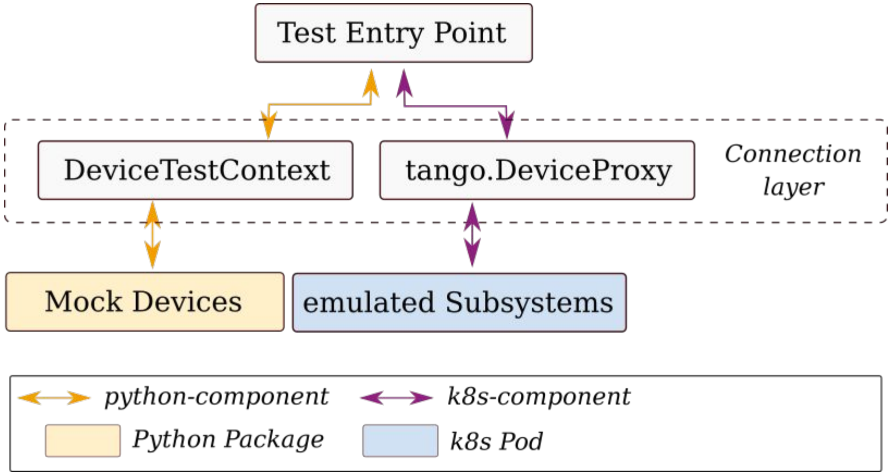
G.Marotta et al, “Strategy and Tools to Test Software in the SKA Project: The CSP. LMC Case”, *Proceedings of the 19th ICALEPCS* (2023)

Testing strategy of CSP.LMC



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

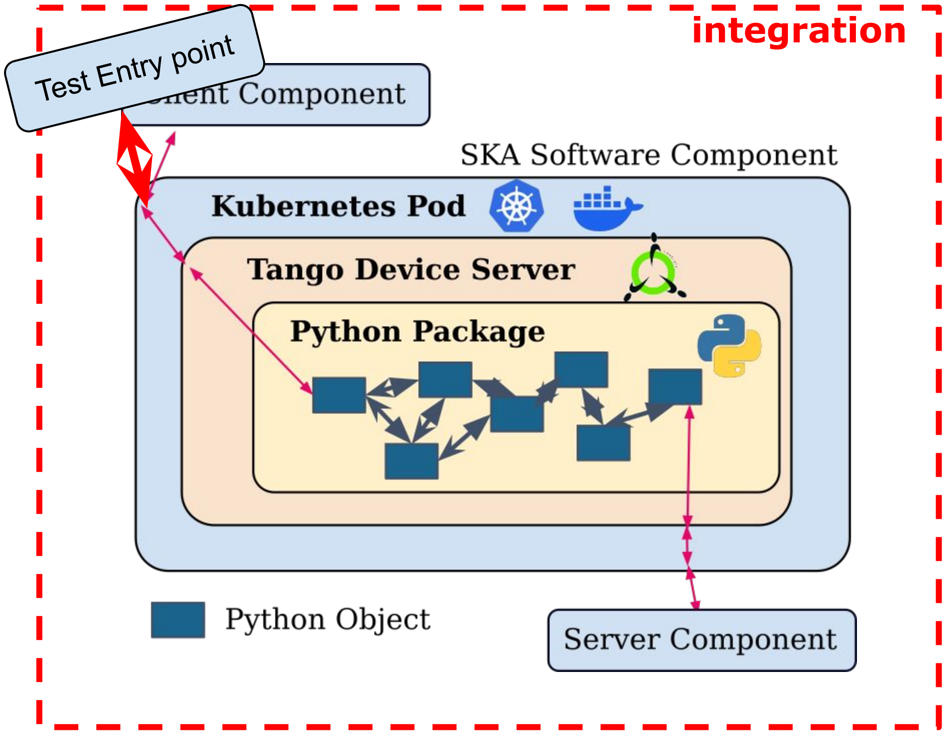
- *unit* tests
- *python-component* tests
- *k8s-component* tests



G.Marotta et al, "Strategy and Tools to Test Software in the SKA Project: The CSP. LMC Case", *Proceedings of the 19th ICALEPCS* (2023)

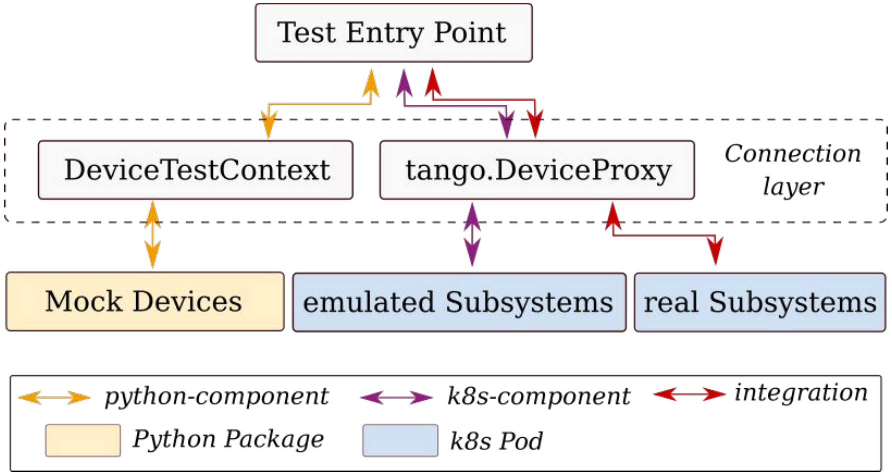
Testing strategy of CSP.LMC

CSP Local Monitoring and Control



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

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



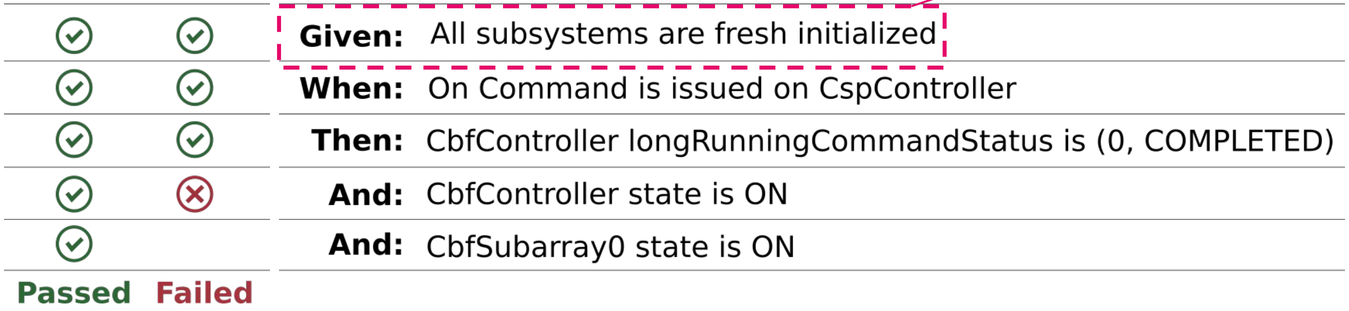
G.Marotta et al, "Strategy and Tools to Test Software in the SKA Project: The CSP. LMC Case", *Proceedings of the 19th ICALEPCS* (2023)

BDD testing

“The purpose of testing is to **increase confidence** for **stakeholders** through **evidence**”[1]

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

“step”



✓	✓	**Given:**	All subsystems are fresh initialized
✓	✓	**When:**	On Command is issued on CspController
✓	✓	**Then:**	CbfController longRunningCommandStatus is (0, COMPLETED)
✓	✗	**And:**	CbfController state is ON
✓		**And:**	CbfSubarray0 state is ON
Passed	**Failed**		

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

Gherkin files can be used as **living documentation**

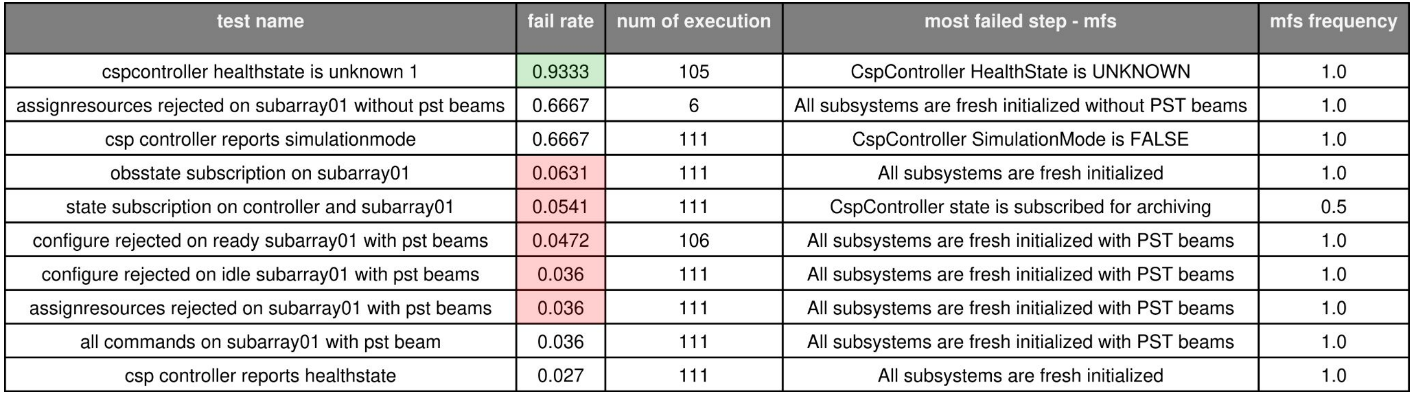
“regression” testing

A single failing test let the CI/CD pipeline to fail and prevent any MR to be effective!

[1] Dan North “We need to talk about testing”- dannorth.net

“Test flakiness” and data mining

Tests **randomly fail** during CI/CD pipeline execution → *“test flakiness”*



test name	fail rate	num of execution	most failed step - mfs	mfs frequency
cspcontroller healthstate is unknown 1	0.9333	105	CspController HealthState is UNKNOWN	1.0
assignresources rejected on subarray01 without pst beams	0.6667	6	All subsystems are fresh initialized without PST beams	1.0
csp controller reports simulationmode	0.6667	111	CspController SimulationMode is FALSE	1.0
obsstate subscription on subarray01	0.0631	111	All subsystems are fresh initialized	1.0
state subscription on controller and subarray01	0.0541	111	CspController state is subscribed for archiving	0.5
configure rejected on ready subarray01 with pst beams	0.0472	106	All subsystems are fresh initialized with PST beams	1.0
configure rejected on idle subarray01 with pst beams	0.036	111	All subsystems are fresh initialized with PST beams	1.0
assignresources rejected on subarray01 with pst beams	0.036	111	All subsystems are fresh initialized with PST beams	1.0
all commands on subarray01 with pst beam	0.036	111	All subsystems are fresh initialized with PST beams	1.0
csp controller reports healthstate	0.027	111	All subsystems are fresh initialized	1.0

an example of test statistics: the 10 most failing tests

data mining on test result helps in providing:

- better metrics on test quality;
- hints on “deeply hidden” bugs

❓ *Predictive maintenance* also for software?

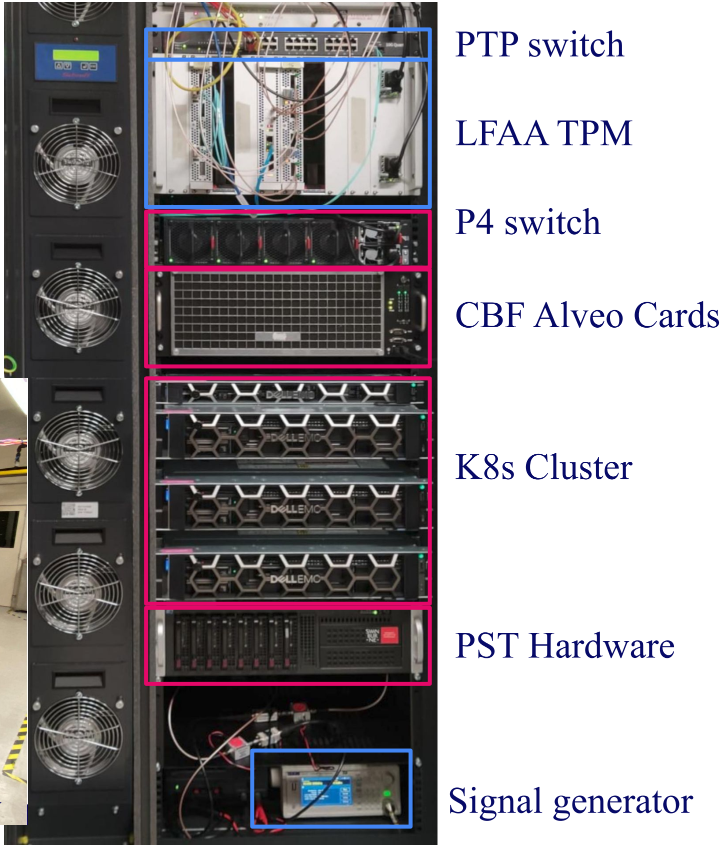
G.Marotta et al., “Enhancing SKA software testing through data mining strategies”, *Software and Cyberinfrastructure for Astronomy VIII; 131010E* (2024)

Testing the SKA Software

Tests with hardware are performed remotely and integrated in CI/CD pipeline.

Testing means also to be aligned with the other Teams!

PSI-LOW Facility - CSIRO @ Sydney



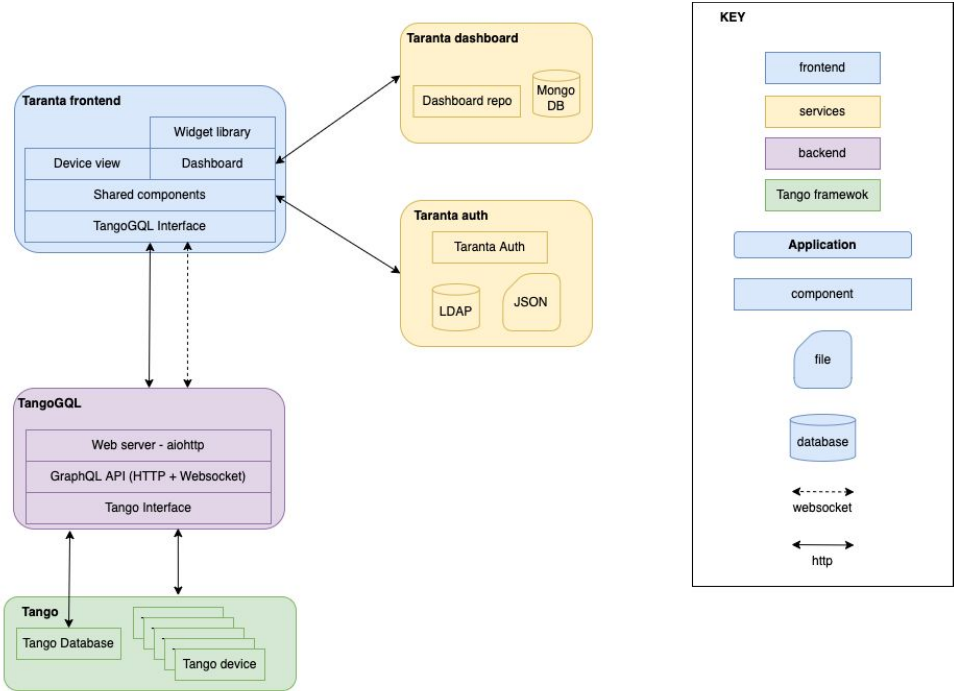
Taranta: a web UI for Tango

TARANTA
TANGO ON WEB

https://webjive.readthedocs.io

Taranta offers a **no-code**, **web-based** approach for creating dashboards that integrate multiple Tango devices.

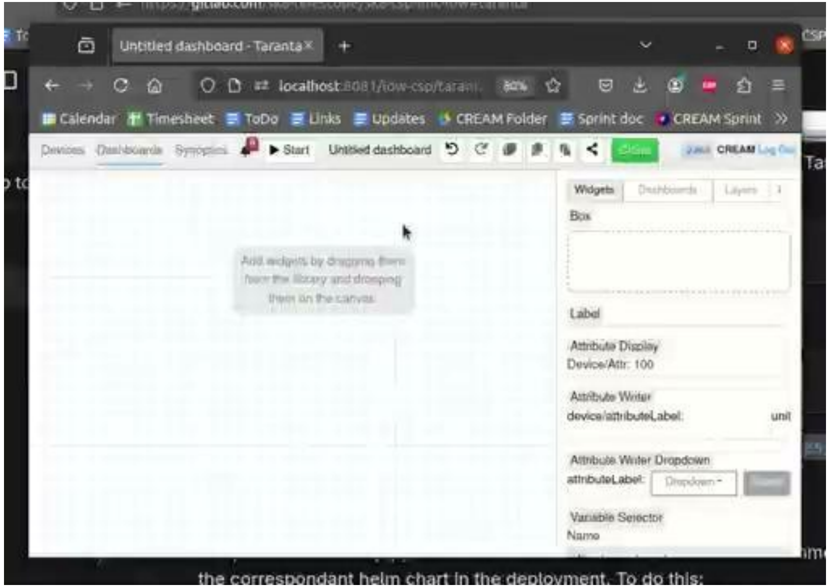
Taranta is developed by CREAM team in collaboration with MAX IV institute



M. Canzari et al., "How Taranta provides tools to build user interfaces for TANGO devices in the SKA integration environment without writing a line of code" *Software and Cyberinfrastructure for Astronomy VII*; 121890W (2022)

Taranta: a web UI for Tango

Taranta is largely used by teams for creating **engineering dashboards**



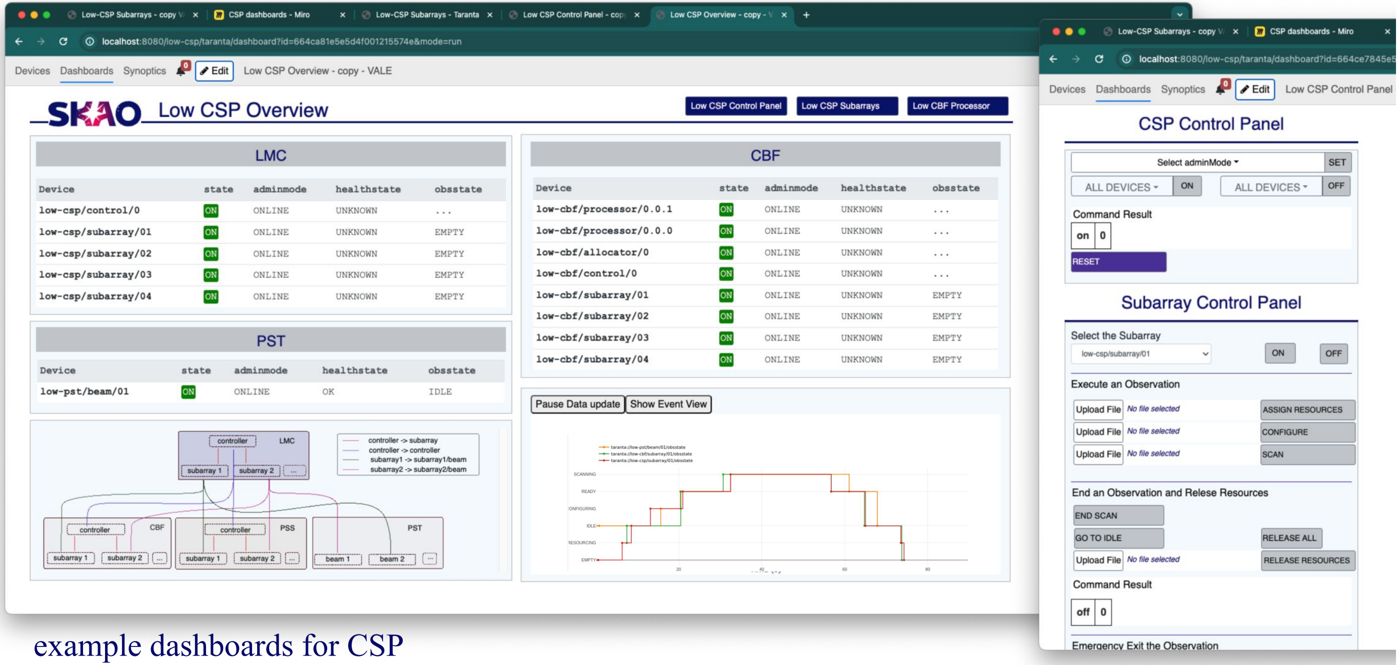
Add widgets by dragging them from the library and dropping them on the canvas

the correspondant helm chart in the deployment. To do this:

- simple to use!
- new functionalities based on user feedback
- possibility to create vectorial and synoptic dashboards with Inkscape

Taranta: a web UI for Tango

Taranta stands as a flexible and scalable candidate for future adoption in the **SKA control room**



Low CSP Overview

LMC

Device	state	adminmode	healthstate	obsstate
low-csp/control/0	ON	ONLINE	UNKNOWN	...
low-csp/subarray/01	ON	ONLINE	UNKNOWN	EMPTY
low-csp/subarray/02	ON	ONLINE	UNKNOWN	EMPTY
low-csp/subarray/03	ON	ONLINE	UNKNOWN	EMPTY
low-csp/subarray/04	ON	ONLINE	UNKNOWN	EMPTY

PST

Device	state	adminmode	healthstate	obsstate
low-pst/beam/01	ON	ONLINE	OK	IDLE

CBF

Device	state	adminmode	healthstate	obsstate
low-cbf/processor/0.0.1	ON	ONLINE	UNKNOWN	...
low-cbf/processor/0.0.0	ON	ONLINE	UNKNOWN	...
low-cbf/allocator/0	ON	ONLINE	UNKNOWN	...
low-cbf/control/0	ON	ONLINE	UNKNOWN	...
low-cbf/subarray/01	ON	ONLINE	UNKNOWN	EMPTY
low-cbf/subarray/02	ON	ONLINE	UNKNOWN	EMPTY
low-cbf/subarray/03	ON	ONLINE	UNKNOWN	EMPTY
low-cbf/subarray/04	ON	ONLINE	UNKNOWN	EMPTY

CSP Control Panel

Command Result

on	0

Subarray Control Panel

Select the Subarray: low-csp/subarray/01

Execute an Observation

End an Observation and Relese Resources

Command Result

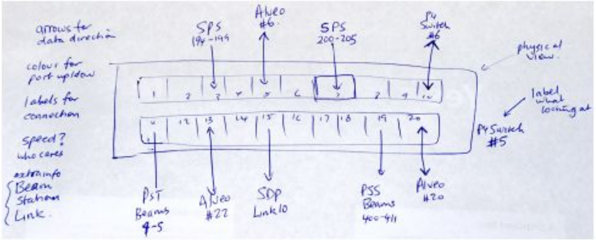
off	0

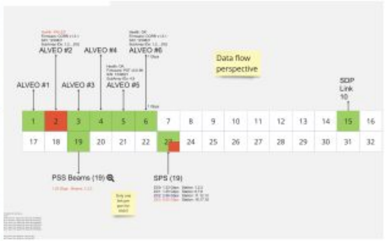
Emergency Exit the Observation

example dashboards for CSP

Creating engineering UIs: lean UX

The creation of User Interfaces for LOW-CSP has been experimented as a cross team collaboration using a **LeanUX approach**



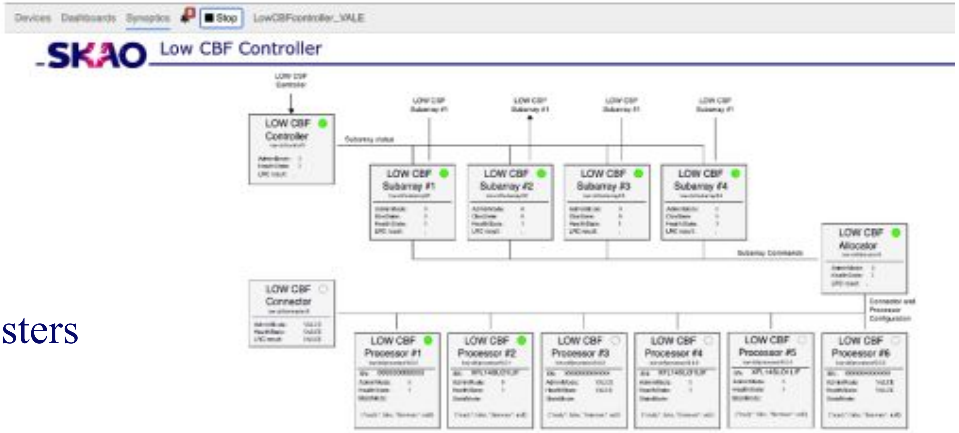


CREAM TEAM

perentie SKA Low Correlator & Beamformer

TOPIC TEAM

Taranta Users are both developer and testers



V.Alberti et al., "Leverage collaborative design techniques to develop a unified interface for the SKA central signal processor", *Software and Cyberinfrastructure for Astronomy VIII;* 131014I (2024)

Thank you for your attention!

About Tango Controls

"Tango Controls is an object oriented, *distributed control system framework*."

- it is built around the concept of **devices,** that run into device servers
    - each device has *state machine*, *commands* , *pipes* and *attributes*
- each Tango system has a centralised **database** that:
    - stores configuration data to start up device servers;
    - acts as a name server storing the dynamic network addresses;
- uses CORBA (syncronous) and ZMQ (asyncronous) to communicate between device server and clients;
- kernel written in C++;
- can be programmed in C++, **Python** or Java.