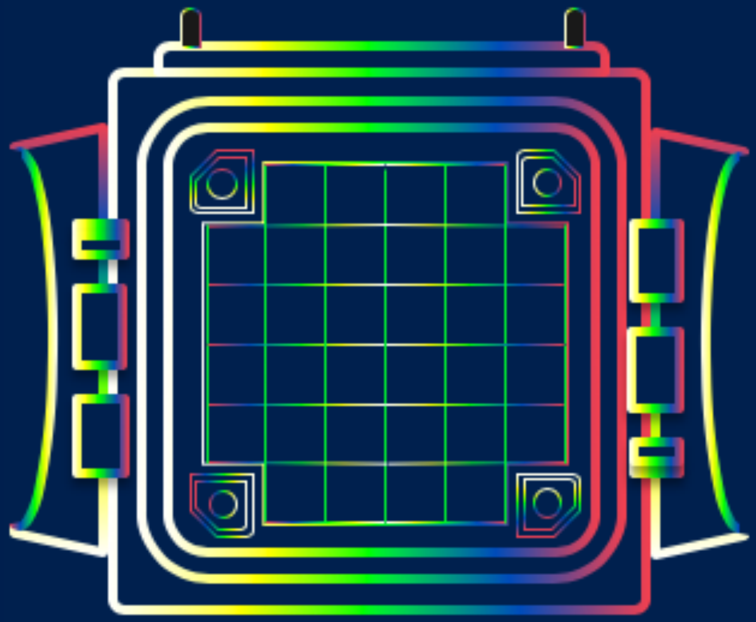


# SST Camera Software

## Overview of the design

Jason J. Watson

SST Software Meeting

February 19, 2024

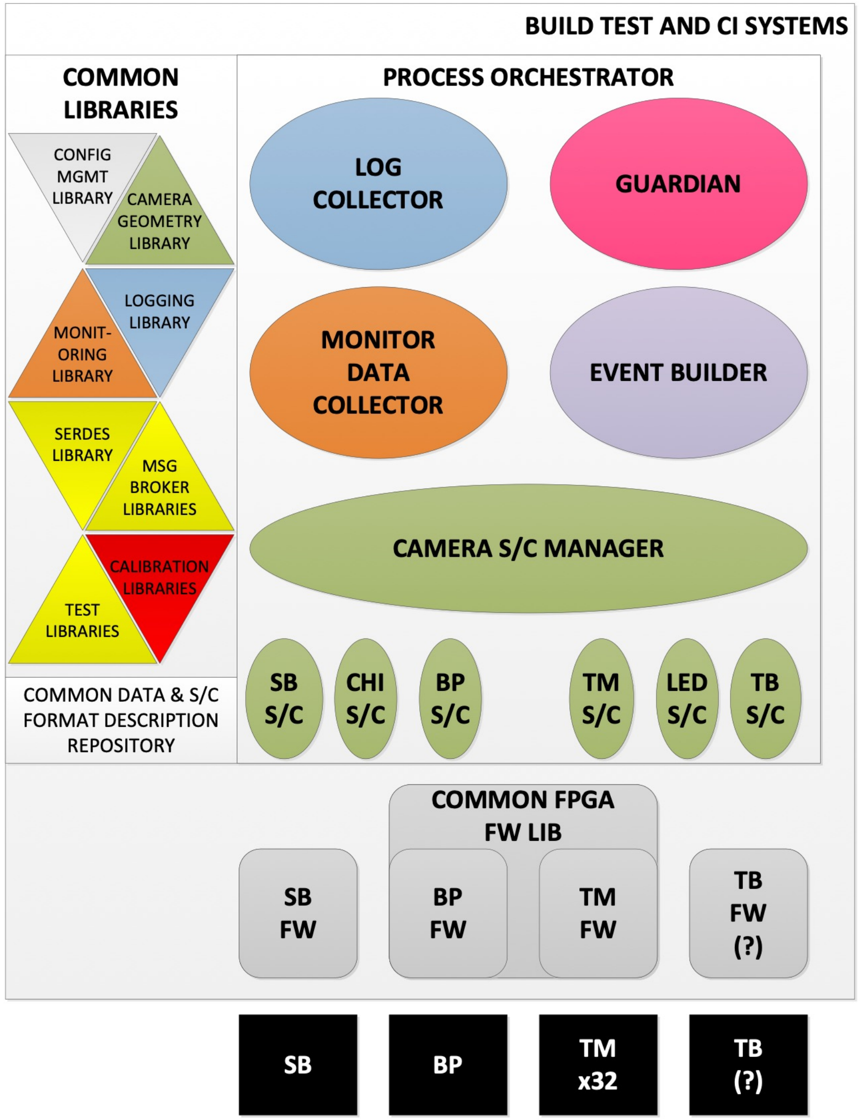
# Motivation of the SST Camera Control Software

- Lessons learned from the CHEC prototype were gathered
- A fresh revamp of the software was logical for the SST camera:
  - Account for the changes in hardware
  - Address the shortcomings identified during internal code reviews and usage in the field
  - Meet the requirements in quality and functionality required for CTA
  - Retain knowledge and positives from the previous software

# Architecture Components Overview

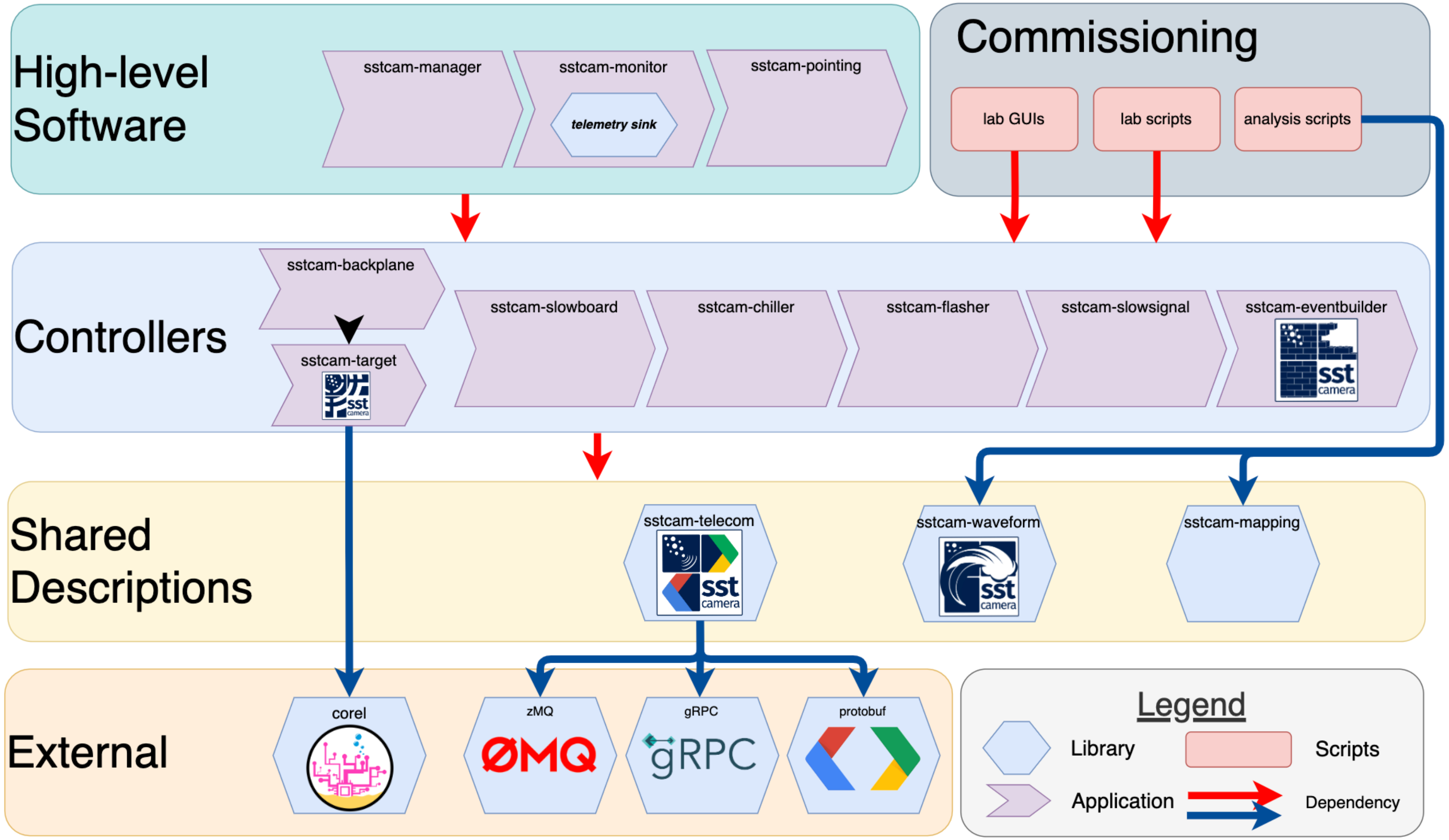
- **Operations modularised into discrete processes, each with a single responsibility**
- Each hardware item of the camera is controlled by its own process
- Collectors gather monitor and log information from other processes
- Manager implements the camera state machine, interfacing with the individual hardware control processes, and makes decisions based on the monitor information collected
- The event builder process handles the bulk waveform data at high performance
- Orchestrator oversees the initialisation and health of all processes
- Common libraries used as dependencies across the system to avoid duplication of shared features
  - messages (monitoring, logging, IPC, etc.), geometry/mapping, configuration, file IO

CAMERA SOFTWARE ARCHITECTURE
COMPONENTS VIEW



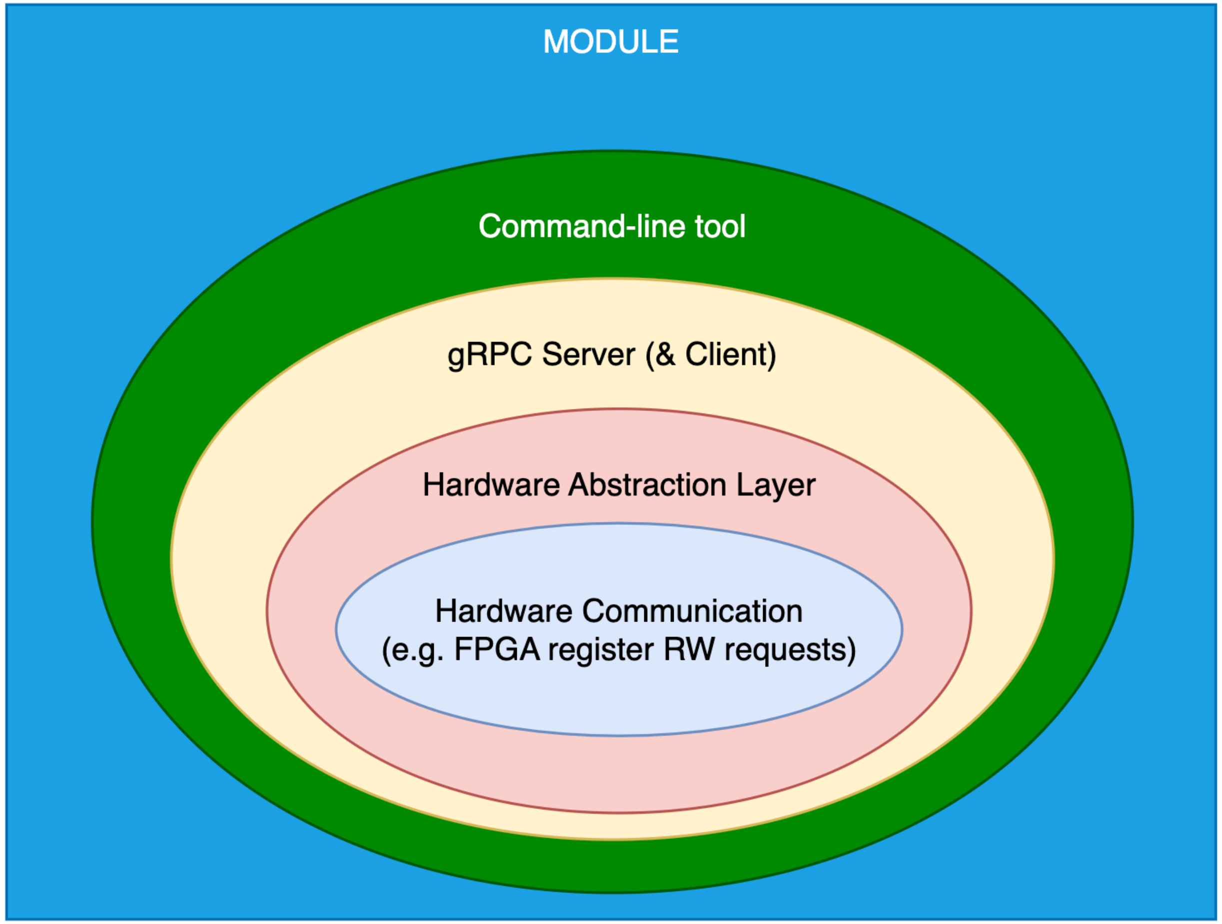
# Packages

- Processes are 1-to-1 with a package
- Three layers of packages:
  - **Shared Descriptions**: Libraries containing the descriptions of messages, serialised camera data and camera geometries
  - **Controllers**: Hardware drivers and control servers (per device)
  - **High-level Software**: Pilot multiple controllers and expose the interface to CTA (ACS)
- Dependencies between layers (bottom to top), not across layers
- Each controller can be installed individually



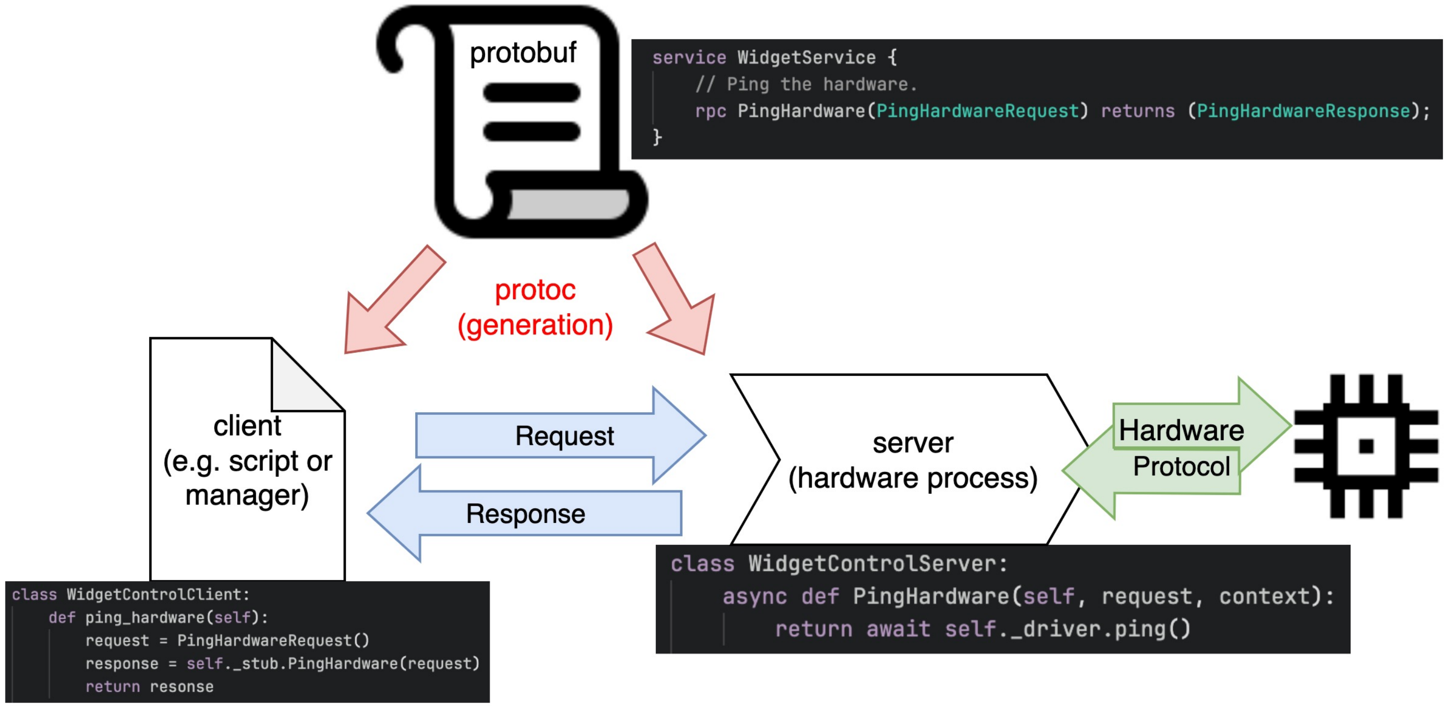
# General Design of the Controllers

- Hardware abstraction layer scheme:
  - Via the gRPC interface, expose all functionalities of the hardware device
  - Users are not required to have the low-level knowledge of how those operations are achieved with the device
- Same layout is adopted for all camera hardware, minimising learning effort for each controller
- Each controller is accompanied with a “mock” of the hardware, implemented in software
- The mock facilitates development and testing of the higher level software in the absence of the hardware



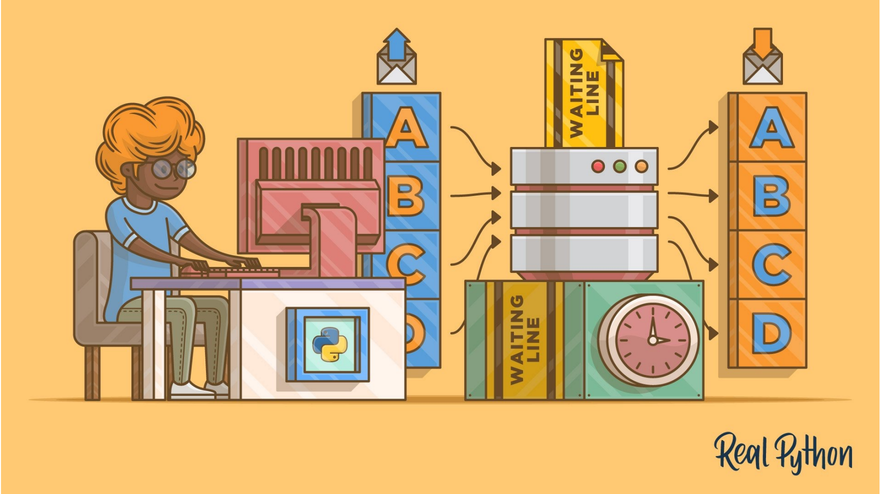
# Inter-Process Communication

- IPC within the SST camera software can be separated into three categories:
  - Internal **Remote Procedure Calls (RPC) -** request a process operation from another process
  - Internal telemetry gathering, to monitor the status of software and hardware in the system
  - Receiving of CTA control requests and sending of camera data to CTA (ACS)
- The technology we chose for **RPC** is gRPC - language agnostic, using protobuf as a common IDL
- Each package is responsible for implementing the server described in the protobuf files
- Any process with knowledge from the protobufs can then make requests to the other servers



# Language & Frameworks

- Python is chosen for its short development cycle time and rich ecosystem
- Asyncio (Python standard library) is used for concurrency within a process
  - Asyncio is designed for IO-dominated operations (e.g. hardware or remote communication)
  - Single threaded event loop, allowing CPU to switch to another task when waiting on IO
  - Switch points are defined **explicitly** (async/await)
- Where high performance is needed (i.e. only the event builder), then C++ wrapped with pybind11
- Trivial pip installation with scikit-build (glue between setuptools and Cmake)
- Language versions are Python ≥3.9, and C++ 11.

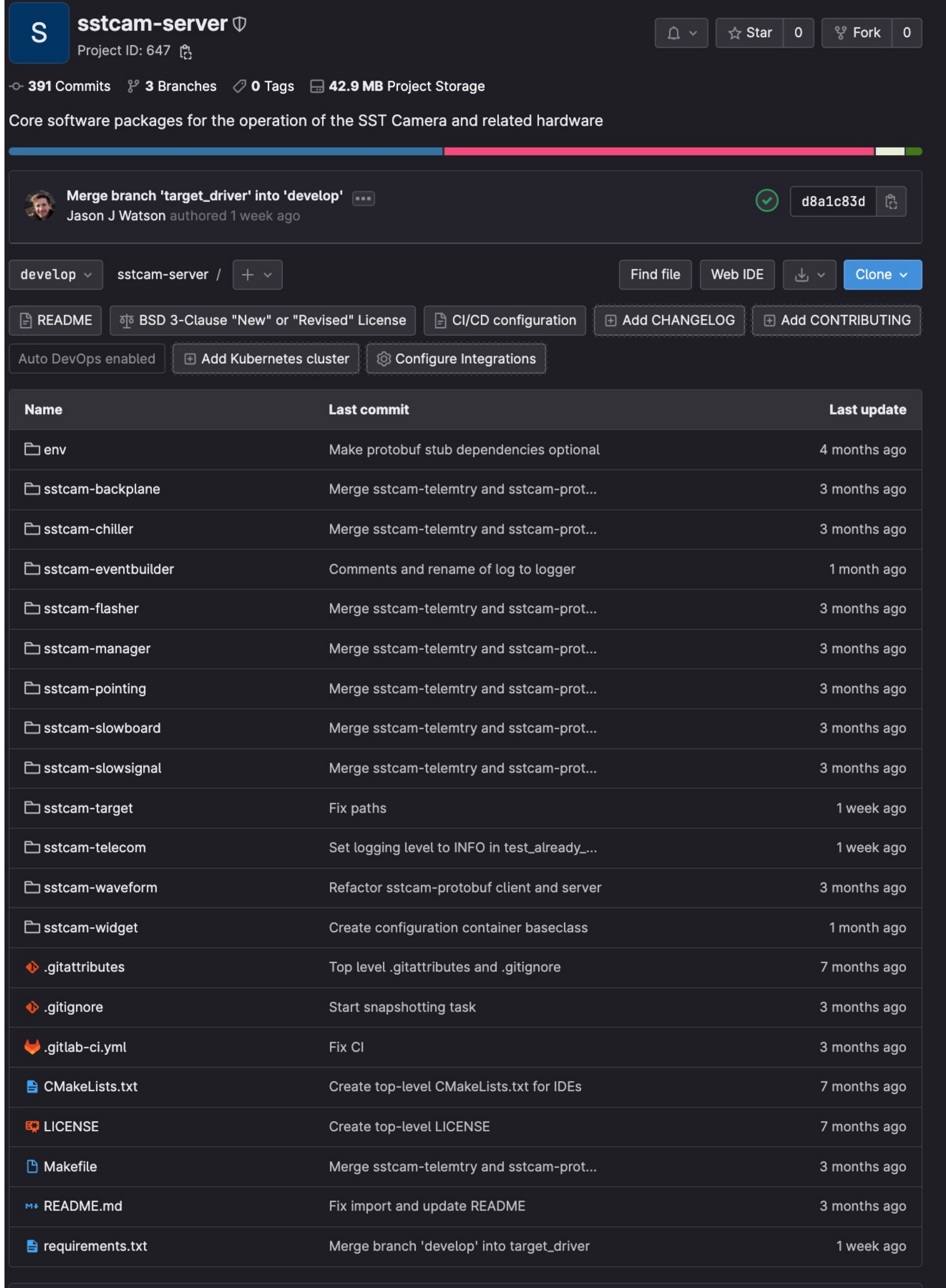



https://realpython.com/async-io-python/

# Version Control

https://gitlab.cta-observatory.org/cta-array-elements/sst/camera/server/sstcam-server

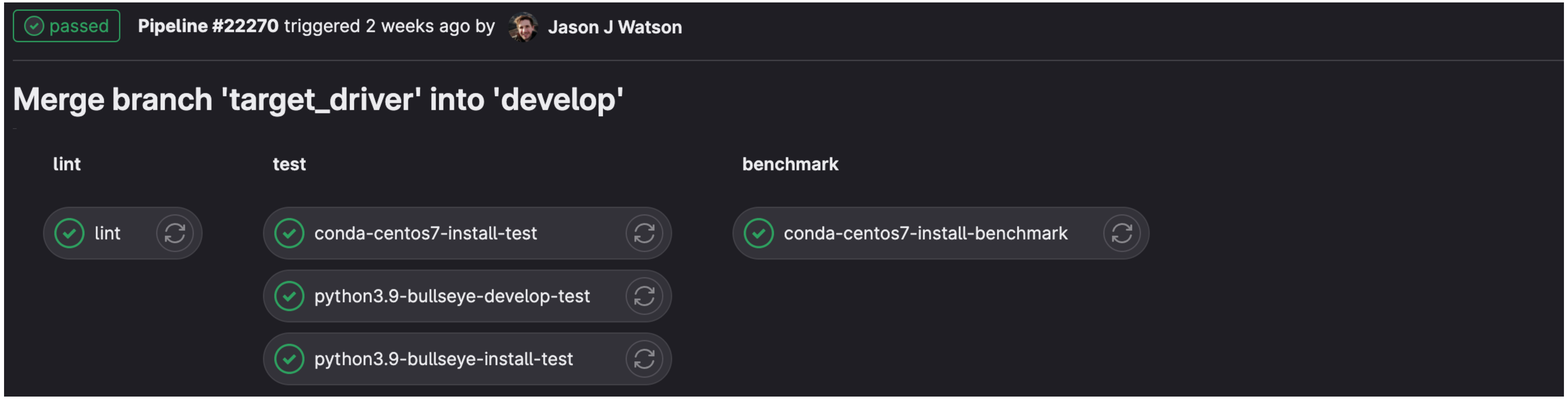
- Mono-repo on CTA Gitlab
- Simple (typical) Gitflow workflow
- Singular clone command for entire codebase
- Singular installation command (Makefile)
- Merge requests can be performed on multiple packages at once
- Continuous Integration (CI) pipeline tests for entire-system compatibility, in addition to individual package unit tests
- Minimal version tracking needed between the packages
- Semantic Versioning is used for the releases




| Name | Last commit | Last update |
|---|---|---|
| env | Make protobuf stub dependencies optional | 4 months ago |
| sstcam-backplane | Merge sstcam-telemtry and sstcam-prot... | 3 months ago |
| sstcam-chiller | Merge sstcam-telemtry and sstcam-prot... | 3 months ago |
| sstcam-eventbuilder | Comments and rename of log to logger | 1 month ago |
| sstcam-flasher | Merge sstcam-telemtry and sstcam-prot... | 3 months ago |
| sstcam-manager | Merge sstcam-telemtry and sstcam-prot... | 3 months ago |
| sstcam-pointing | Merge sstcam-telemtry and sstcam-prot... | 3 months ago |
| sstcam-slowboard | Merge sstcam-telemtry and sstcam-prot... | 3 months ago |
| sstcam-slowsignal | Merge sstcam-telemtry and sstcam-prot... | 3 months ago |
| sstcam-target | Fix paths | 1 week ago |
| sstcam-telecom | Set logging level to INFO in test_already_... | 1 week ago |
| sstcam-waveform | Refactor sstcam-protobuf client and server | 3 months ago |
| sstcam-widget | Create configuration container baseclass | 1 month ago |
| .gitattributes | Top level .gitattributes and .gitignore | 7 months ago |
| .gitignore | Start snapshotting task | 3 months ago |
| .gitlab-ci.yml | Fix CI | 3 months ago |
| CMakeLists.txt | Create top-level CMakeLists.txt for IDEs | 7 months ago |
| LICENSE | Create top-level LICENSE | 7 months ago |
| Makefile | Merge sstcam-telemtry and sstcam-prot... | 3 months ago |
| README.md | Fix import and update README | 3 months ago |
| requirements.txt | Merge branch 'develop' into target_driver | 1 week ago |

# Testing

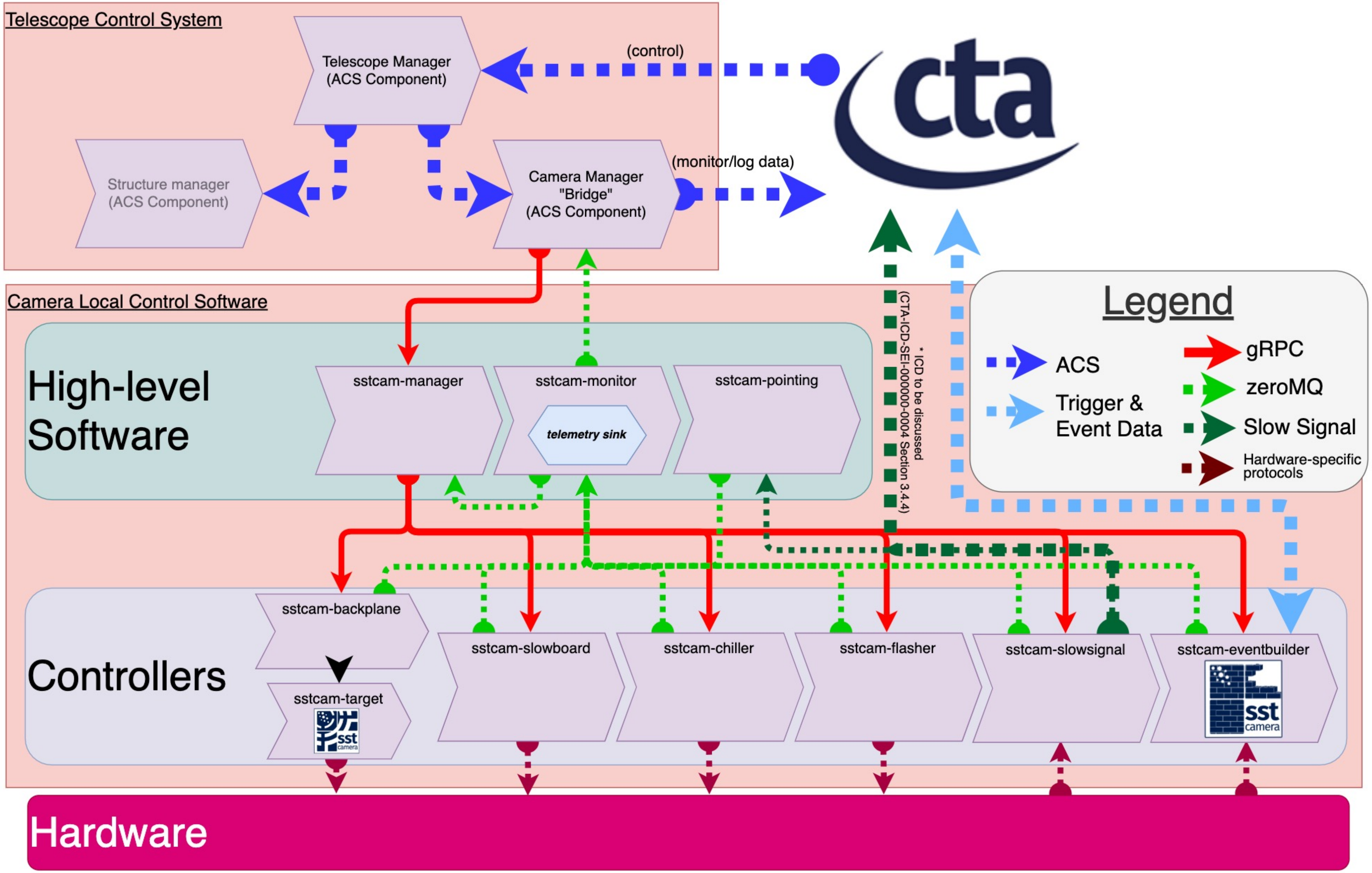
- Unit tests for each component in a package
  - Including tests on supportive non-code items (e.g. configuration, TM .def files…)
- Integration tests between driver and hardware
  - Define tests which can be identically run against the mock, and the hardware (when available)
  - Tests against the mock are ran in the CI to ensure against breaking functionality
  - Tests against hardware are ran before every release to confirm interface has not changed
- Integration tests between high-level software and controllers
  - Using the mocks, exercising the system, state machine, and event/alert handling



# Summary

- Lessons learnt from CHEC prototypes are incorporated into the new software design
- Software is designed with careful consideration of modularity and inter-dependencies
- Hardware-specific details are abstracted from the user for ease of use
- Coherent design across the software packages
- Frameworks used are Python asyncio and pybind11/C++
- Build system is kept simple
- Serious consideration given to the different end-users of the software
- Strong emphasis placed on testing

# Backup: IPCs



Telescope Control System
Telescope Manager (ACS Component)
(control)
Structure manager (ACS Component)
Camera Manager "Bridge" (ACS Component)
(monitor/log data)
cta
Camera Local Control Software
* ICD to be discussed
(CTA-ICD-SEI-000000-0004 Section 3.4.4)
High-level Software
sstcam-manager
sstcam-monitor
telemetry sink
sstcam-pointing
Legend
ACS
Trigger & Event Data
gRPC
zeroMQ
Slow Signal
Hardware-specific protocols
Controllers
sstcam-backplane
sstcam-target
sstcam-slowboard
sstcam-chiller
sstcam-flasher
sstcam-slowsignal
sstcam-eventbuilder
sst camera
Hardware