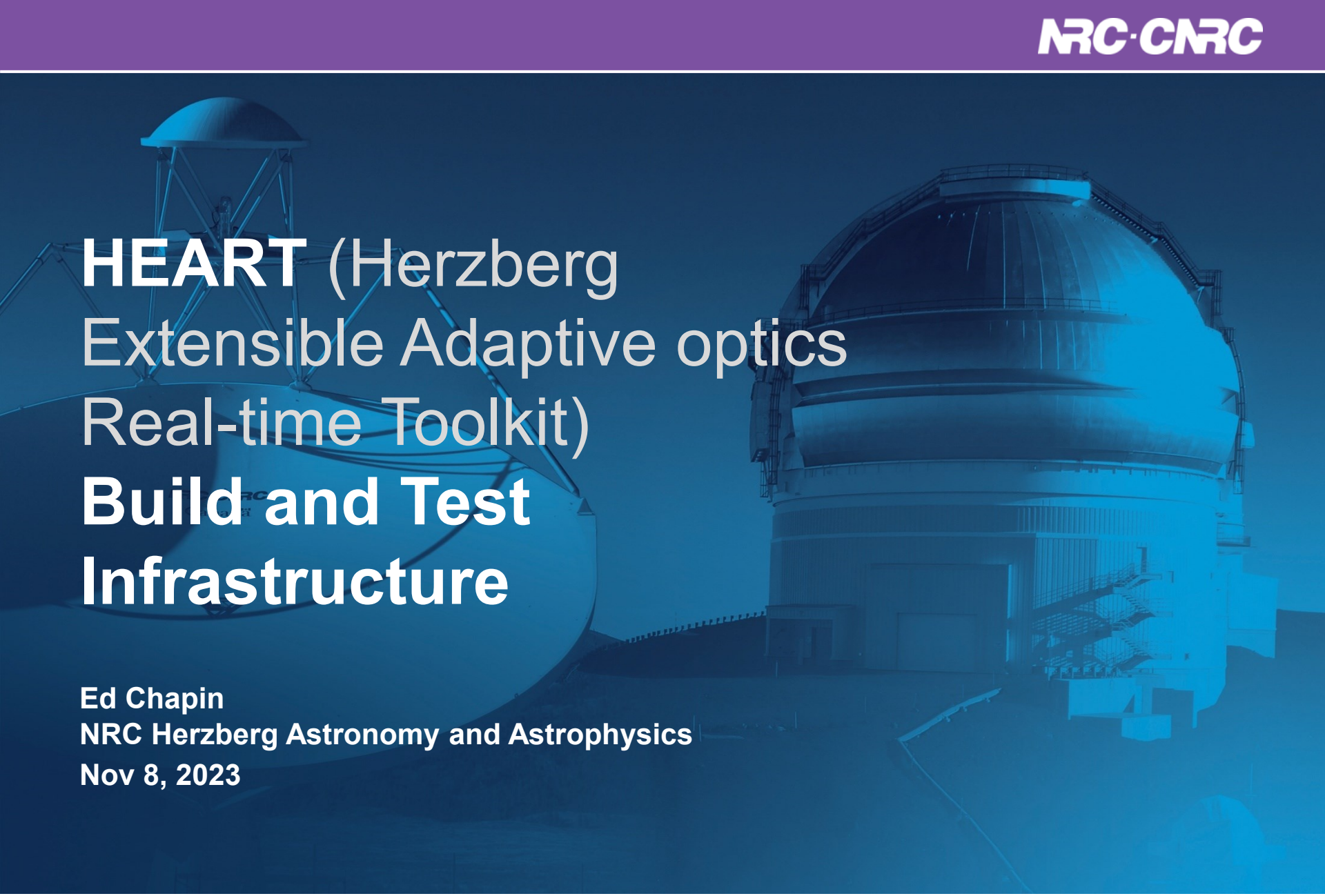


HEART (Herzberg Extensible Adaptive optics Real-time Toolkit) **Build and Test Infrastructure**

Ed Chapin
NRC Herzberg Astronomy and Astrophysics
Nov 8, 2023

Introduction

We need you to use CentOS 7!

The Soft Realtime processes must run on the same server as the Hard Realtime!

A special PCIe card that only runs with Linux Kernel x.y.z must be installed in the RTC server!

We need you to use AlmaLinux 9!

The Soft Realtime processes must *NOT* run on the same server!

The complexity of the Hard Realtime processes requires multiple servers!

We need a release every month with test and coverage reports!

Continuous integration + extensive automated tests are essential for QA, and so we can refactor and extend our code “fearlessly”.

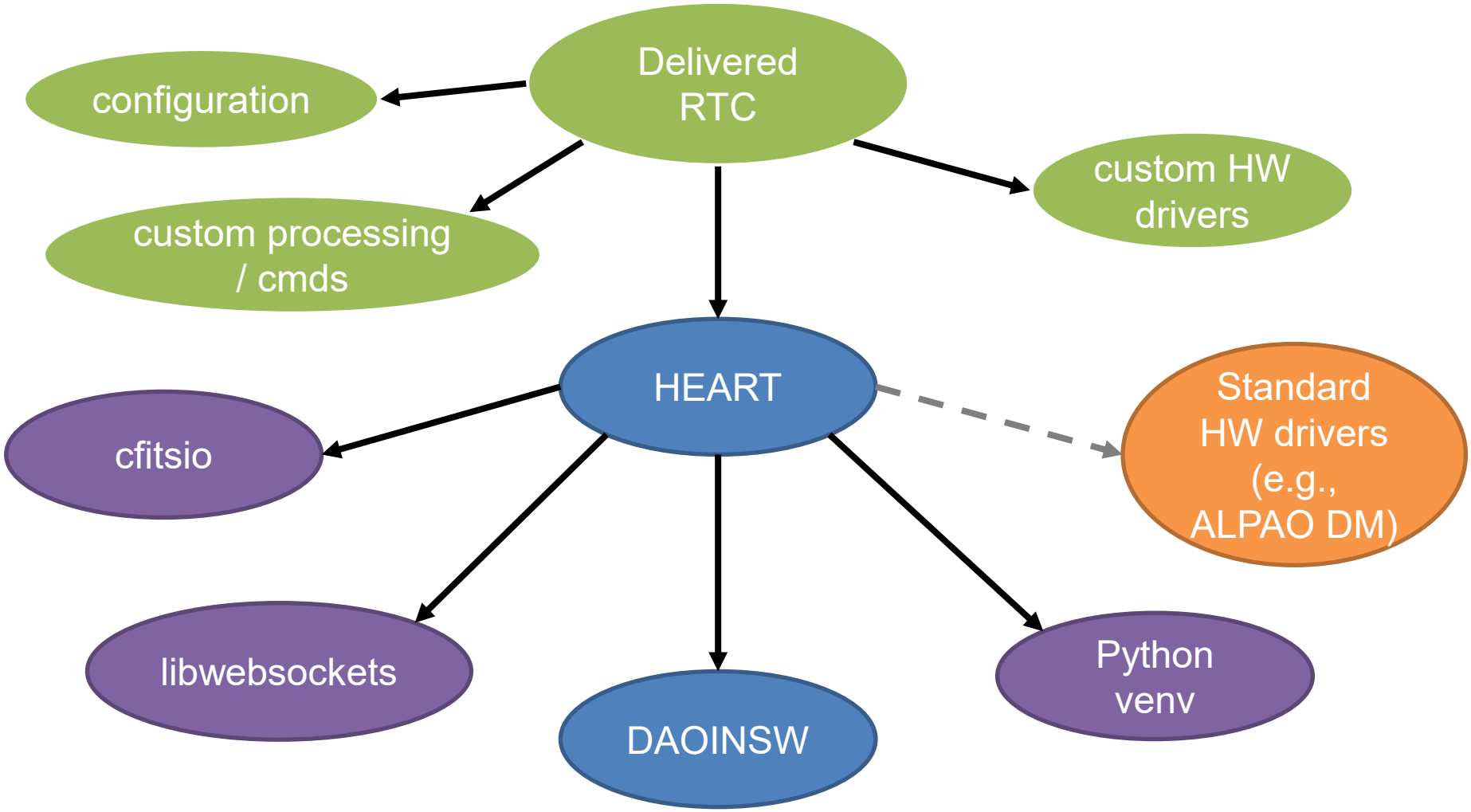
Overview

- **Managing dependencies and revision control** for multiple systems/clients
- **Languages** (C / Python) and **tooling**
- Types of automated testing:
  - Low-level unit tests
  - Application-level component tests
  - System tests
- (near) **future updates**:
  - Container-based test runners for different architectures
  - Automated Performance tests (currently manual)

Dependencies

- **Facility instrument requirements typically conservative** (need to run for many years), and vary by observatory (e.g., target OS)
- **Result**: HEART predominantly written in C and Python, *with minimal external dependencies*:
  - **Doxygen** for documentation
  - **GNU Make**
  - C: **cfitsio**, **libwebsockets, *CMocka, lcov***
  - Python: **numpy**, **astropy**, **matplotlib, *PyTest***
  - Optional: **support for different hardware based on availability of vendor SDKs** and build flags

Dependencies (cont'd)



Revision Control

```
+--gpi2.0-rtc   # root of the repo
  |--doc        # Top level documentation
  |--source     # GPI2.0 source code
  +--external   # Location of submodules
     |--cfitsio
     |--daoinsw
     |--heart
     |--heart-testdata
     +--libwebsockets
```

Submodules encode both the location of the external dependency and the particular version (commit) in its history

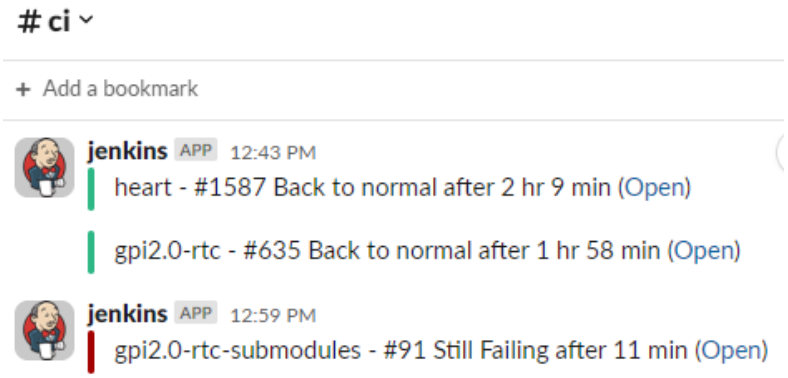
Tooling

- **Makefiles:** build all of the C source code (**gcc**), execute tests (C/Python), and produce documentation (Doxygen)
- **Python:** venv + requirements.txt
- **Git:** develop in branches labeled by Jira tickets
- **Jenkins:** merges trigger automated builds
- **Slack:** #ci sends annoying messages if someone caused a build to fail
- **Documentation:** built documentation includes test report

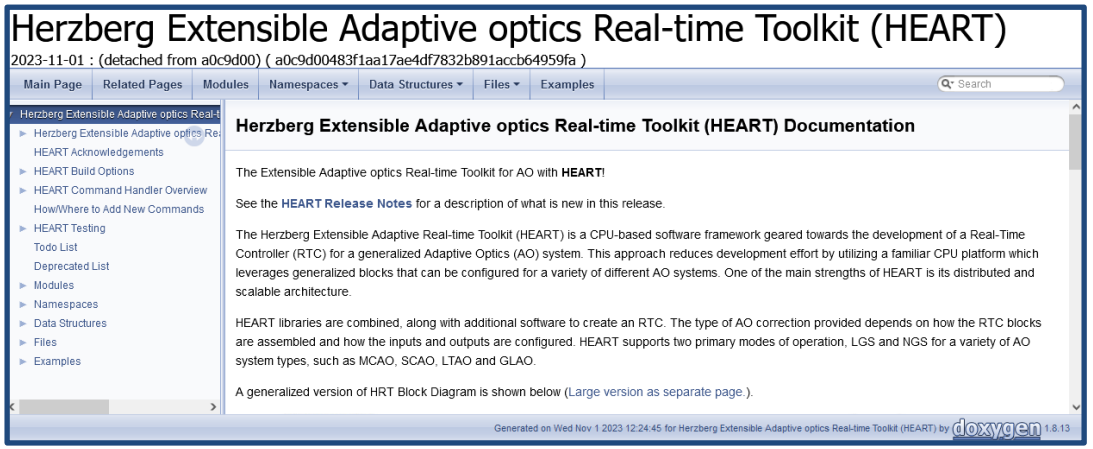
Automated Builds

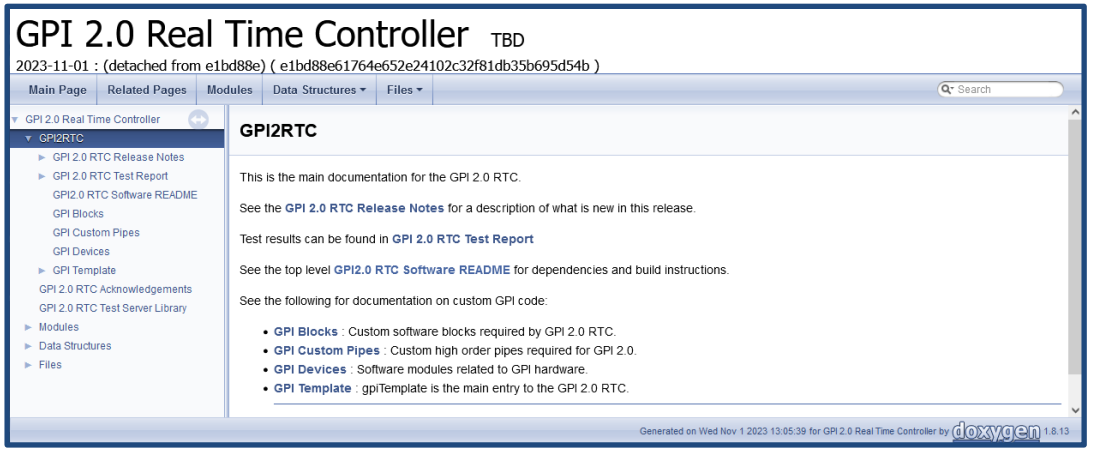
Merge of branch to master triggers builds (with dependencies)

Result sent to slack



Built Documentation





Unit Tests

- CMocka (https://cmocka.org/):
  - lightweight / capable framework for C + XML test report
  - Used for a mixture of “true” unit tests (single functions), but also some more complex (including multi-threaded) tests.
  - Use linker option “--wrap” to incorporate mocked functions when needed
  - **lcov** to measure code coverage
- PyTest
  - unit tests of our Python code
  - Measure code coverage + XML test reports

Unit Tests: CMocka

We use CMocka for both low-level tests (individual) functions), and higher-level tests that may involve complex interactions between multiple threads.

Since everything is running in the same process, our test code is able to **make assertions about internal variables, as well as inputs and outputs.**

Each test suite is a stand-alone executable with XML output that can be included in a test report.

Unit Tests: CMocka (cont'd)

```
int main( DAO_ERR_COMP_UNUSED int argc, DAO_ERR_COMP_UNUSED char *argv[] )
{
    const struct CMUnitTest hrtMvmTests[] = {

        cmocka_unit_test( test_hrtMvm_subMvm_invalid ),
        cmocka_unit_test( test_hrtMvm_subMvm_column_matrix ),
        cmocka_unit_test( test_hrtMvm_subMvm_full_matrix ),
        cmocka_unit_test( test_hrtMvm_subMvm_row_matrix ),
        cmocka_unit_test( test_hrtMvm_mvm_invalid ),
        cmocka_unit_test( test_hrtMvm_mvm_full_matrix ),
        cmocka_unit_test( test_hrtMvm_partitioned_mvm ),
        cmocka_unit_test( test_hrtMvm_parallel_mvm_row_partitions ),
        cmocka_unit_test( test_hrtMvm_parallel_mvm_col_partitions ),
        cmocka_unit_test( test_hrtMvm_parallel_mvm_4x8_partitions ),
        cmocka_unit_test( test_hrtMvm_scratch_memory ),
        cmocka_unit_test( test_hrtMvm_parallel_invalid ),
    };

    cmocka_set_message_output( CM_OUTPUT_XML );
    return cmocka_run_group_tests( hrtMvmTests, NULL, NULL );
}
```

Test suite "cmocka_hrtMvm.c" for low-level MVM module

Unit Tests: CMocka (cont'd)

```
/*!
 * \test \ref test_hrtMvm_subMvm_column_matrix <br>
 *
 * \brief Test hrtMvm_subMvm function when matrix is a small column vector.
 *
 * The matrix M is set to a known column vector and X is set to a
 * known scalar. The vector length iterates through lengths 1 ... 32.
 */
static void test_hrtMvm_subMvm_column_matrix( DAO_ERR_COMP_UNUSED void **state )
{
    // ... Setup omitted ...
    for ( vLen = 1; vLen <= maxLen; ++vLen )
    {
        rv = hrtMvm_subMvm( V, &mat.full, X );
        assert_int_equal( rv.status, DAO_ERR_NONE );

        // Compute error vector as V - expected.
        rv = hrtVec_acc_ax_by( vLen, error, 1.0f, V, -1.0f, expected );
        assert_int_equal( rv.status, DAO_ERR_NONE );

        rv = hrtVec_norm( vLen, error, hrtVec_norm_L1, &errNorm, 0 );
        assert_int_equal( rv.status, DAO_ERR_NONE );

        assert_float_equal( errNorm, 0.0, 0.0 );
    // ... etc ...
```

Doxygen-friendly comments for test report

Failed assertions will appear in test report with line number

Unit Tests: CMocka (cont'd)

More complex tests using setup & teardown functions. Example for wavefront input sensor block:

```
cmocka_unit_test_setup_teardown( test_hrtWfsInputBlock_normal, setup, teardown )
```

setup:

- Configure and start the block (multi-threaded) with simulator for feeding pixels

teardown:

- Shutdown block, free resources

test code:

- Triggers reading of pixels (command), checks results

CMocka test reports

Our test reports appear in the doxygen-generated pages as a hierarchy, ranging from an overview, to modules (like “HEART math” in this example), and individual test functions

HEART math

Summary of Results

test suite	# of tests	failures	errors	skipped	time
cmocka_hrtFft	8	0	0	0	0.007
cmocka_hrtFilterIIR	16	0	0	0	0.000
cmocka_hrtMatrix	20	0	0	0	0.015
cmocka_hrtMvm	12	0	0	0	0.409
cmocka_hrtPixelCalibration	4	0	0	0	0.032
cmocka_hrtSparse	21	0	0	0	0.249
cmocka_hrtVec_acc_aacc_bx	1	0	0	0	0.006
cmocka_hrtVec_acc_ax_by	1	0	0	0	0.009
cmocka_hrtVec_alloc_free	3	0	0	0	0.000
cmocka_hrtVec_dot	4	0	0	0	0.002
cmocka_hrtVec_eProd	1	0	0	0	0.001
cmocka_hrtVec_mvm	6	0	0	0	0.426
cmocka_hrtVec_norm	4	0	0	0	0.689
cmocka_hrtVec_sum	1	0	0	0	0.001
cmocka_hrtVector	10	0	0	0	0.008
subtotal	**112**	0	0	0	**1.854**

result	test name	test description	time
passed	test_hrtMvm_subMvm_invalid	Test hrtMvm_subMvm function with NULL parameters and 1x1 matrix.	0.000
passed	test_hrtMvm_subMvm_column_matrix	Test hrtMvm_subMvm function when matrix is a small column vector.	0.000

CMocka code coverage (lcov)

Filename	Line Coverage			Functions		Branches	
hrtFft.c		89.8 %	273 / 304	100.0 %	14 / 14	76.3 %	145 / 190
hrtFilterIIR.c		67.6 %	200 / 296	85.7 %	6 / 7	66.7 %	108 / 162
hrtMath.c		0.0 %	0 / 5	0.0 %	0 / 1	0.0 %	0 / 2
hrtMatrix.c		97.1 %	435 / 448	100.0 %	15 / 15	94.6 %	176 / 186
hrtMvm.c		92.1 %	152 / 165	100.0 %	6 / 6	87.1 %	61 / 70
hrtPixelCalibration.c		90.6 %	58 / 64	100.0 %	2 / 2	85.0 %	34 / 40
hrtSparse.c		93.8 %	347 / 370				
hrtVec_acc_aacc_b...		100.0 %	14 / 14				
hrtVec_acc_ax.c		100.0 %	14 / 14				
hrtVec_acc_ax_by.c		100.0 %	18 / 18				
hrtVec_copySlice.c		100.0 %	17 / 17				
hrtVec_dot.c		100.0 %	40 / 40				
hrtVec_eProd.c		100.0 %	18 / 18				
hrtVec_mvm.c		93.5 %	58 / 62				
hrtVec_norm.c		100.0 %	38 / 38				
hrtVec_sum.c		100.0 %	19 / 19				
hrtVec_v_ax.c		100.0 %	14 / 14				
hrtVec_v_ax_b.c		100.0 %	22 / 22				
hrtVec_zero.c		100.0 %	5 / 5				
hrtVector.c		91.0 %	7... / 78				

```
629        [ +  + ]:         16 :     if ( M->mvmScratch == NULL )
630                 :            :     {
631                 :          2 :         rv = daoErr( HRT_MAT_ERR_MVM_SCRATCH,
632                 :            :                      "No MVM scratch memory available." );
633                 :          2 :         DAO_DEBUG( DAO_DEBUG_ERROR, "%s", rv.errMsg );
634                 :          2 :         return rv;
635                 :            :     }
636                 :            :
637                 :            :     /*
638                 :            :      * The upper bound should be based on current limits.
639                 :            :      */
640                 :         14 :     lastCore = firstCore + coreStride * ( numThreads - 1 );
641                 :            :
642        [ +  + ]:         14 :     if ( lastCore >= daoCpuSet_maxCores )
643                 :            :     {
644                 :          2 :         rv = daoErr_format( HRT_MAT_ERR_MVM_CORE,
645                 :            :                             "# threads (%d) != # partition (%u * %u).",
646                 :            :                             numThreads, M->numRowPartitions, M->numColPartitions );
647                 :            :
648                 :          2 :         DAO_DEBUG( DAO_DEBUG_ERROR, "%s", rv.errMsg );
649                 :          2 :         return rv;
650                 :            :     }
651                 :            :
652                 :            :     /*
653                 :            :      * Check requested priority and adjust if necessary.
654                 :            :      * This check may generate a warning but will not prevent the
655                 :            :      * MVM from being performed.
656                 :            :      */
657                 :            :
658                 :         12 :     mvmPriority = daoRT_getAllowedPriority( reqPriority );
659                 :            :
660        [ -  + ]:         12 :     if ( mvmPriority < reqPriority )
661                 :            :     {
662                 :          0 :         rv = daoErr_format( HRT_MAT_WARN_MVM_PRIORITY,
663                 :            :                             "Requested MVM thread priority (%d) clipped at %d.",
664                 :            :                             reqPriority, mvmPriority );
665                 :            :
666                 :          0 :         DAO_DEBUG( DAO_DEBUG_ERROR, "%s", rv.errMsg );
667                 :            :     }
```

PyTest: native code

- Python used in HEART for:
  - a number of **soft real-time tasks**, e.g., generating reconstructor matrices using statistics provided by hard real-time (C) code
  - **data analysis and plotting**
  - **general utilities**
- PyTest used to test native Python code. Test reports and coverage incorporated into documentation similar to CMocka

PyTest: native code (cont'd)

```
def test_telemetryStream_basic():
    """
    \test \ref test_telemetryStream_basic

    \addtogroup hrtTelemetryUnitTests
    @{
    \copybrief test_telemetryStream_basic
    @}

    \brief Basic test of streaming from a sender to a receiver
    - sender is started and not yet connected
    - receiver is started and after brief pause both report connected
    - verify that receive queue is initially empty
    - bucket is sent from sender to receiver
    - verify that receive queue has the (single) bucket
    """

    debug = True

    with TelemetryStream(TEST_ADDRESS, TEST_PORT, specs=TEST_SPECS, debug=debug) as sender:
        assert sender._streamThreadState == TelemetryStream.ThreadState.CONNECTING

    # ... etc ...
```

doxypypy allows us to include Doxygen commands in Python docstrings

PyTest: native code (cont'd)

test_telstream

Unit tests for **daoinsw.util.telstream**.

test suite	# of tests	failures	errors	skipped	time
pytest	**5**	0	0	0	4.513

result	test name	test description	time
passed	daoinsw.util.tests.test_telstream.test_telemetryStream_fails		0.009
passed	daoinsw.util.tests.test_telstream.test_telemetryStream_basic		1.158
passed	daoinsw.util.tests.test_telstream.test_telemetryStream_reconnect		0.478
passed	daoinsw.util.tests.test_telstream.test_telemetryStream_largeBucket		1.138
passed	daoinsw.util.tests.test_telstream.test_telemetryStream_customBucket		1.148

PyTest: native code (cont'd)

Coverage report: 91%

coverage.py v7.2.1, created at 2023-10-16 16:45 -0700

Module	*statements*	*missing*	*excluded*	*coverage*
daoinsw/__init__.py	0	0	0	100%
daoinsw/util/__init__.py	2	0	0	100%
daoinsw/util/circbuf.py	153	11	0	93%
daoinsw/util/cstruct.py	85	13	0	85%
daoinsw/util/hist.py	159	44	0	72%
daoinsw/util/stringHelpers.py	21	0	0	100%
daoinsw/util/telstream.py	223	21	0	91%

```
227     def waitForConnection(self, timeout=None):
228         """Blocking method that returns when a connection has been made
229 
230         Args:
231             timeout: optionally specify a timeout in seconds
232         """
233 
234         if self._streamThreadState == TelemetryStream.ThreadState.CONNECTING:
235             self._connectedEvent.wait(timeout=timeout)
236 
237         if self._streamThreadState == TelemetryStream.ThreadState.STOPPED:
238             raise Exception("The stream thread has stopped.")
```

Component & System Tests

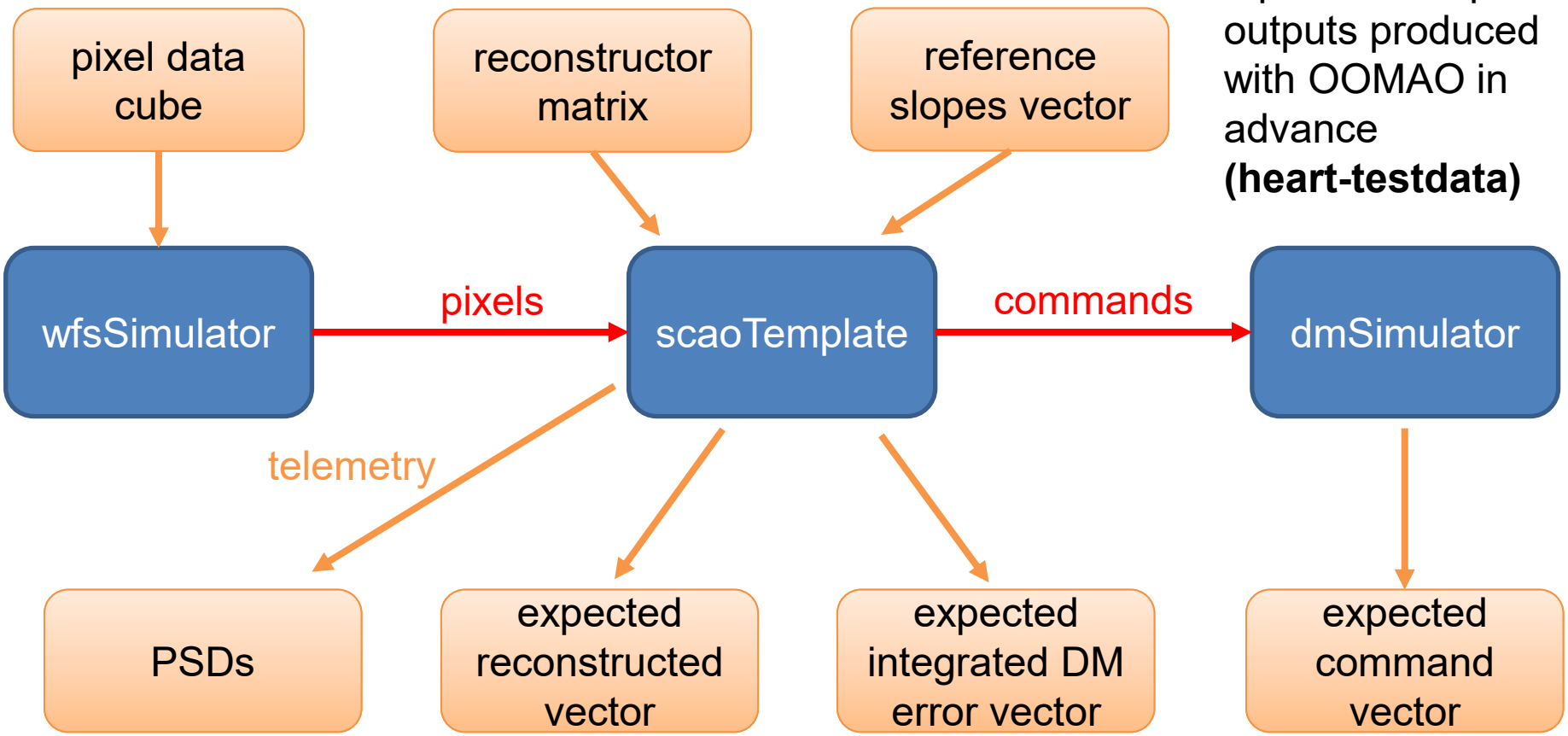
- Component and System tests are “black box”.
  - **Component test**: single application
  - **System test**: multiple (interacting) applications
  - Use external interfaces to provide inputs, and observe outputs (results)
- Many goals for these tests:
  - **Verify external interfaces** (commands, files, telemetry)
  - **Correctness** (RTC produces expected DM commands from WFS data?)
  - **Performance** (e.g., timing requirements)

Component & System Tests (cont'd)

Also use PyTest to orchestrate these tests:

- Makes sense to **re-use existing tool**
- Execute applications using **subprocess** module
- Elegant way to start and monitor multiple applications using **fixtures** (setup/teardown with dependencies)
- **Provide inputs** (commands, telemetry, input files)
- Perform **assertions on outputs**
- Create **test reports**

Example: SCAO RTC Validation



Example: SCAO RTC Validation

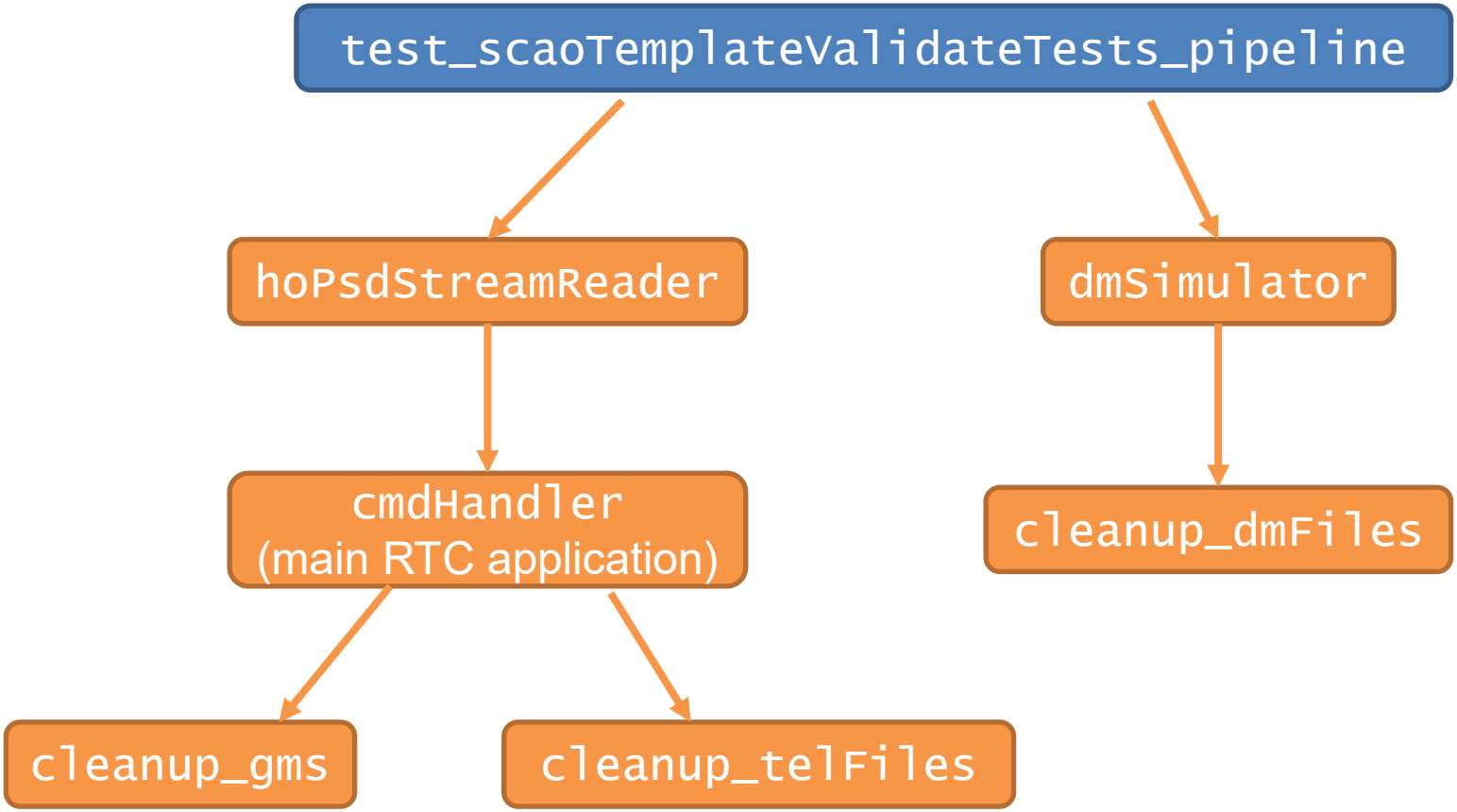
The main test depends on **fixtures,** which may themselves depend on other fixtures. Used to start up applications/processes in a particular order.

```
def test_scaoTemplateValidateTests_pipeline(hoPsdStreamReader, dmSimulator):
    """
    \test \ref test_scaoTemplateValidateTests_pipeline

    \addtogroup scaoTemplateValidateTests
    @{
    \copybrief test_scaoTemplateValidateTests_pipeline
    @}

    \brief Feed pipeline known WFS inputs and check correctness of results
    """
```

Example: SCAO RTC Validation



Example: SCAO RTC Validation

```
@pytest.fixture
def cmdHandler(cleanup_gms, cleanup_telFiles, dmSimulator):
    """
    Startup and shutdown of \ref scaoTemplate
    """

    # === Setup ===

    proc = None
    configPathFile = DEFAULT_CONFIG_PATH + "/" + DEFAULT_CONFIG_FILE_NAME
    cmdLine = [compPrg, "-config", configPathFile,
               "-host", "compTestScaoTemplate"] #, "-d", "4"]
    proc = ServiceProc(cmdLine, expect_str="About to create listening address")

    # === Teardown ===

    yield cmdHandler

    if proc is not None:
        print("Shutting down cmdHandler")
        proc.stopProc()
```

This fixture starts the main RTC process and returns once it is ready to accept commands.

It also shuts it down after the test.

Example: SCAO RTC Validation

```
# ... Further down in the main test function ...

# Send simulated pixels to RTC and return when done
wfsDataPath = TEST_DATA_PATH+"/"+TIPTILT_DATA_PATH+WFS_DATA_FILE
wfsProc = run_WfsSimulator(wfsDataPath)
assert wfsProc.returncode == 0

# ... Command RTC to dump circular buffers to disk and also
#     load in expected values from OOMAO, then compare ...

for frame in range(nTestFrames):
    np.testing.assert_allclose(dmCmds[frame, :, 0],
                               dmExpected[frame, :],
                               atol=dmtol,
                               err_msg='DM commands from '+dmFile)
```

NumPy has some great testing routines for comparing arrays with tolerances

Component & System Test Reports

HEART Test Report

- HEART Test Coverage
- HEART cmocka Unit Test Results
- HEART Component Test Results
- heart Python Package Unit Test Results
- heart Python Package Unit Test Coverage
- HEART Test Li

This test report was pr duced using the following tool versions:

```
Computer   : ncc  24 of 24 cores available )
CPU        : Intel R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz
Compiler   : gcc ( CC) 4.8.5 20150623 (Red Hat 4.8.5-36)
Doxygen    : 1.8.1
Python3    : Python 3.9.13
pytest     : pytest 7.1.2
pytest-cov : pytest cov-4.0.0

Linux version 3.10.0 957.1.3.el7.x86_64 (mockbuild@kbuilder.bsys.centos.org)
(gcc version 4.8.5 2 150623 (Red Hat 4.8.5-36) (GCC) ) #1 SMP Thu Nov 29 14:49:4 UTC 2018
```

See HEART Testing for mo information on testing HEART.

HEART Component Test Results

library test suite	# of tests	failures	errors	skipped	time
HEART cmdHandler	8	0	0	0	14.644
HEART gui	2	0	0	1	12.387
HEART perfMon	16	0	0	0	11.531
HEART template	8	0	0	0	88.546
HEART testServer	12	0	0	0	18.945
HEART util	1	0	0	0	1.735
total	48	0	0	1	147.788

HEART template

Summary of Results

test suite	# of tests	failures	errors	skipped	time
comptest_scao2Wfs	1	0	0	0	36.833
comptest_scaoTemplate	3	0	0	0	5.821
comptest_scaoTemplateExtra	1	0	0	0	4.386
comptest_scaoTemplateValidate	1	0	0	0	10.594
comptest_scaoTemplateValidate_pyrPix_modal	1	0	0	0	17.287
comptest_scaoTemplateValidate_wooferTweeter	1	0	0	0	13.321
ubtotal	8	0	0	0	88.242

comptest_scaoTemplateValidate

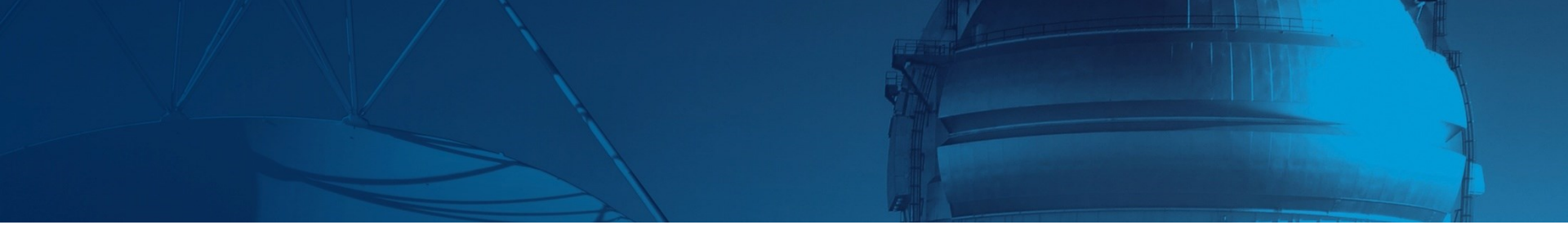
Validation testing for for scaoTemplate.

test suite	# of tests	failures	errors	skipped	time
pytest	1	0	0	0	10.594

result	test name	test description	time
passed	comptest_scaoTemplateValidate.test_scaoTemplateValidateTests_pipeline		10.303

The Future

- Currently updating test servers infrastructure:
  - **Install range of operating systems** to get more experience with tuning (e.g., Ubuntu vs. RedHat-style systems), identify toolchain issues
  - Use **docker containers** to test different operating systems *within* different host operating systems
- Performance tests currently manual. Want **standard set of automated performance** tests to check for regressions (**mean latency**, **tail**, **dropped frames**)
- Migrate from **Jenkins -> GitLab** (locally hosted) to manage test execution



QUESTIONS?

Thank you

Ed Chapin
Ed.Chapin@nrc-cnrc.gc.ca