MIT HAYSTACK OBSERVATORY

MSRI HOPS Redevelopment Review

J. Barrett, G. Crew, D. Hoak, V. Pfeiffer

MIT Haystack Observatory

February 4, 2021

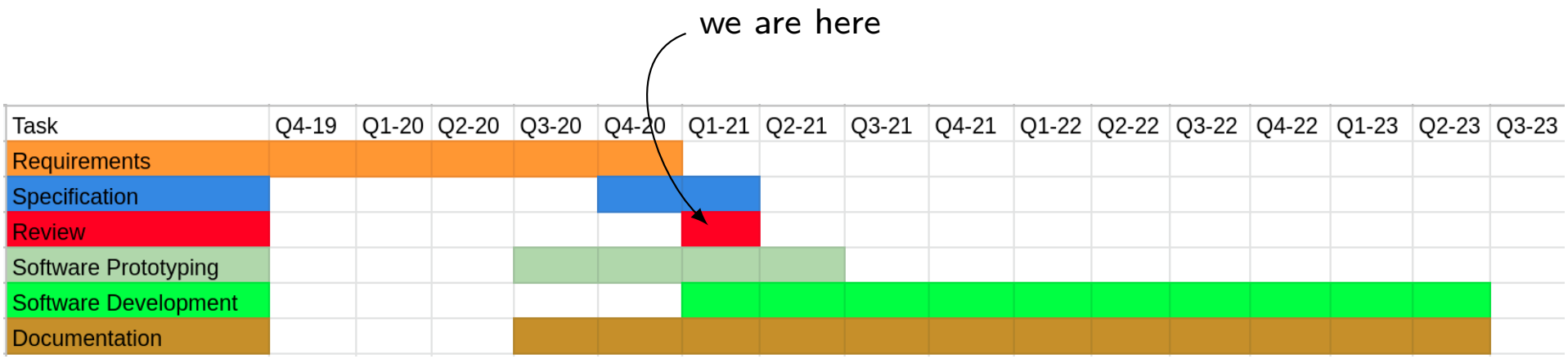
Outline

MIT HAYSTACK OBSERVATORY

1. Snapshot & Background
2. Scope, Requirements, and Assumptions
3. Software architecture and identified interfaces
4. Development plan
5. Current design state
6. Schedule
7. Risks/Mitigation

Quick Snapshot

MIT HAYSTACK OBSERVATORY



1. Delayed start, but original schedule had plenty of margin
2. Largely on track/schedule - dev. plan document under revision
3. Software development/prototyping stage has started and in progress.

Background

MIT HAYSTACK OBSERVATORY

HOPS - Haystack Observatory Post-processing System

1. HOPS's primary purpose is to provide post (correlation) data processing utilities for VLBI data. These utilities include fringe-fitting, calibration, data flagging/reduction, and data quality checking.
2. HOPS has a multi-decade history as a post-processing/analysis package and as such has evolved gradually over time.
    1. Current C-code was written in the early 90's by Colin Lonsdale, Roger Cappallo and Cris Niell. The basic algorithms were adopted from FRNGE, by Alan Rogers in the late 70's
    2. Efforts in the 00's with tools to optimize SNR and in response to software correlators such as DiFX.
    3. For the 2017 EHT campaign, HOPS was augmented with python-based packages. HOPS performed well when measuring delays and correlation coefficients (compared to AIPS, CASA)
3. Primary component is the fringe-fitting software *fourfit*.
4. Also includes data summarization software (alist/aedit) – provide diagnostics for debugging correlation and evaluating data quality.
5. HOPS has reached the limits of what can be fixed with "bandaids"
6. Due to current architecture, HOPS (and *fourfit*) is difficult to extend/modify

Assumptions: Use-case model

MIT HAYSTACK OBSERVATORY

1. Assumed that general use-cases for the new HOPS software package are expected to follow the existing use model and be a combination of processes consisting of:
    1. Import/Export of data from a software correlator (DiFX).
    2. Data flagging/removal of corrupted visibilities in time/frequency.
    3. Calibration and phase/amplitude correction of data, as well as other manipulation (summing over polarization products, etc.)
    4. Fringe-fitting (solving for residual phase/delay w.r.t to correlator model) on either a per-baseline or global basis.
    5. Data quality analysis and visualization on a per-scan and per-experiment basis.
    6. Data export to archival and/or imaging formats (examples: HDF5, UVFITS).

Further Assumptions

MIT HAYSTACK OBSERVATORY

1. Some assumptions are baked into the higher level requirements, low level specification details in work.
    1. HOPS4 will accept DiFX correlator input (including VEX at current v1.5.1)
    2. ngEHT data format will resemble EHT data, except for more stations/baselines/sub-bands.
    3. General computational performance of *fourfit* in HOPS3 was acceptable for EHT (with SPMD parallelization).
    4. General software run-time environments will be similar to HOPS3 (unix like)
2. Some ngEHT details have not been provided, example:
    1. Memory requirements of ngEHT data sets.
    2. Possible unspecified future use-cases
3. These assumptions entail risks – the assumptions may be wrong, and the code may require modification to meet them – can discuss in risks section.

Scope and Requirements

MIT HAYSTACK OBSERVATORY

1. The overarching goal of this project is to upgrade/replace the existing HOPS software with a modern and extensible version not subject to the limitations of the old package while still maintaining the existing capabilities.
2. Generally speaking the limitations we aim to eliminate are both technical and practical in nature:
   1. Technical:
      - Limited or no ability for dynamic memory allocation, which imposes a limit on the number of stations, APs, and sub-bands (channels).
      - Fully complex (amplitude and phase) band-pass corrections are not possible (only phase/delay corrections).
      - Only a single per-scan/baseline fringe-finding algorithm is available
      - Currently no support for multi-processing, apart from SPMD
   2. Practical:
      - Control file syntax is arcane, plotting and results are not decoupled
      - Custom data treatment (e.g. band-pass correction) is limited to that which is allowed by the control file parameters or ad-hoc phase files (i.e. no easy way to inject custom code, whether that be compiled-in or scripted).
      - File and data formats (Mk4-types) which are restrictive and not easily modified, and tightly coupled to code.
      - Reliance on out-dated/unsupported libraries (e.g. PGPLOT).
3. Further details in the distributed requirements document (v1.0).

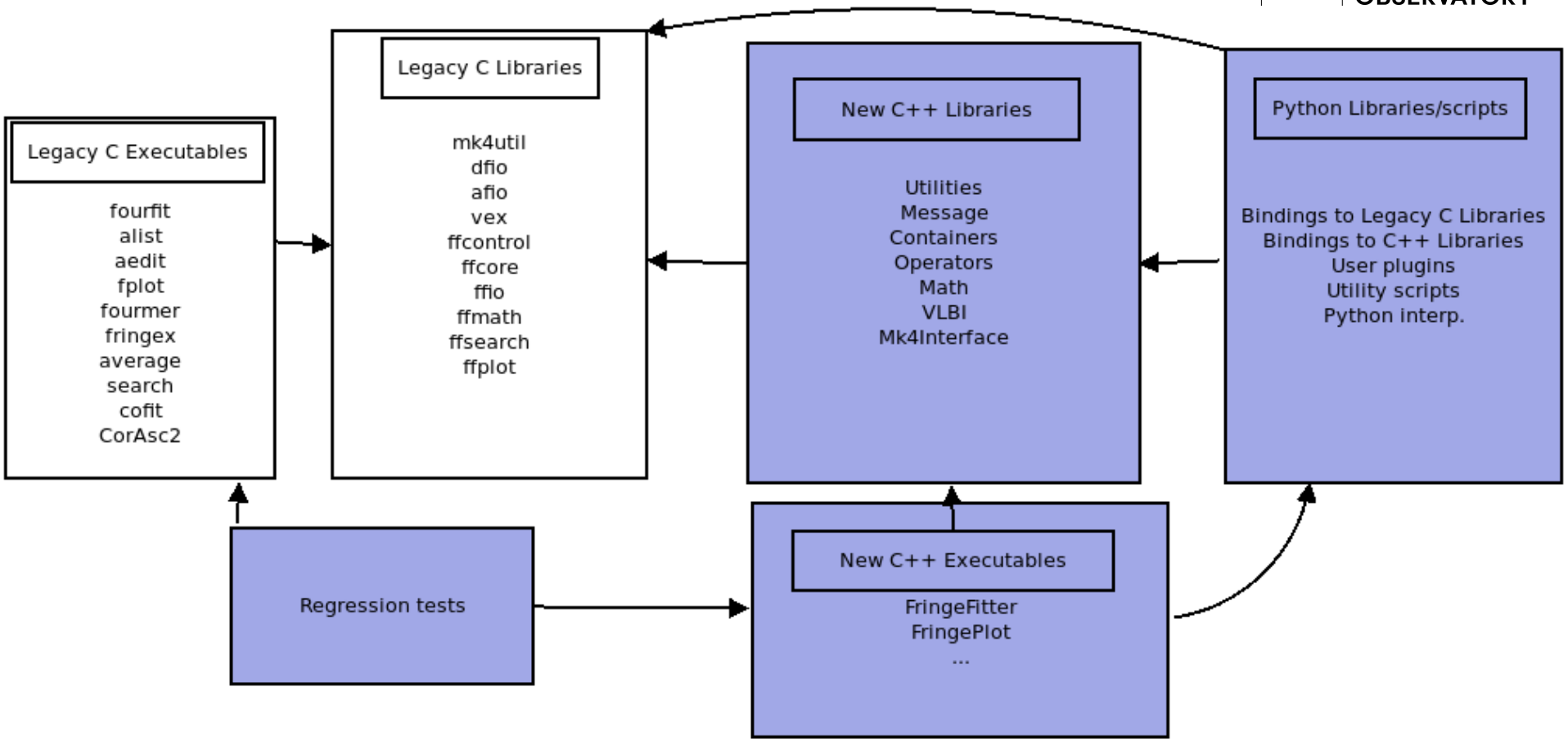
Software Architecture

MIT HAYSTACK OBSERVATORY

1. The previous/existing HOPS architecture is composed of three major monolithic applications (fourfit, alist, aedit) and several minor ones (cofit, average, fourmer, search, etc).
2. Software is supported by four low-level libraries (mkutil, dfio, afio, vex)
3. New architecture is an object oriented component-based design to be composed of a set of loosely coupled task-specific libraries along with a similar set of (extensible) applications.
4. The new architecture will also support extension of the library/application code in two ways:
    1. Developer (compile-time) extensions to data-manipulation will be added as needed as new derived classes of a basic set of 'data operators'. These can then be configured and inserted into the application control flow at run time.
    2. User (run-time) extensions will be supported by a python plug-in interface, consisting of bindings to the in-memory data containers and operators, and a series of hooks (at specific points) in the application code where a user-script can be injected.
5. The choice of C++ as the development language for the new libraries is discussed in the specifications document (v1.0).

Software Architecture (Internal dependencies)

MIT HAYSTACK OBSERVATORY



Dependency relationships of the main internal software components. Shaded boxes are new components of this project. Arrows indicate direction of dependence (point from child to parent).

Software Architecture (Libraries)

MIT HAYSTACK OBSERVATORY

1. Legacy c-libraries (made available for re-use and backwards compatibility)
   1. dfio - I/O for MK4 data types
   2. mk4util - utility library for MK4 data types
   3. afio - I/O library for alist manipulation
   4. vex - vex parsing library
   5. fourfit specific libraries (maintained to provide legacy-fourfit application):
      - ffcontrol - parse old-style fourfit control files
      - ffcore - core parameter structures
      - ffio - output for fringe file data
      - ffmath - trivial math routines used by fourfit
      - ffplot - fringe-plot generation library
      - ffsearch - core fringe-search algorithm library (grid-search)
2. New C++ libraries*
   1. Containers - In-memory data containers of visibility and meta-data and native format disk I/O
   2. Utilities - Commonly used utilities (e.g. date handling, profile timers, object toolboxes, etc.)
   3. Math - Math routines: FFTW, minimizers, interface to optional 3rd-party math libraries
   4. Message - Controls the topic and verbosity level of application messages/logs to user
   5. MK4Interface - Allows for import/export to legacy MK4-types for testing and backwards compatibility
   6. Operators - Base classes for data manipulation, and generic operations (array reduction, transposition, FFT, etc.)
   7. VLBI - VLBI task-specific operators (e.g. band-pass correction, per-channel phase offsets, etc.)
   8. Bindings - Python interface to C++ data containers and operators. Allows for Python script configuration of data operators at initialization, and user-defined direct manipulation of data at pre-defined hooks.

* current working names, subject to change

Software Architecture (Applications)

MIT HAYSTACK OBSERVATORY

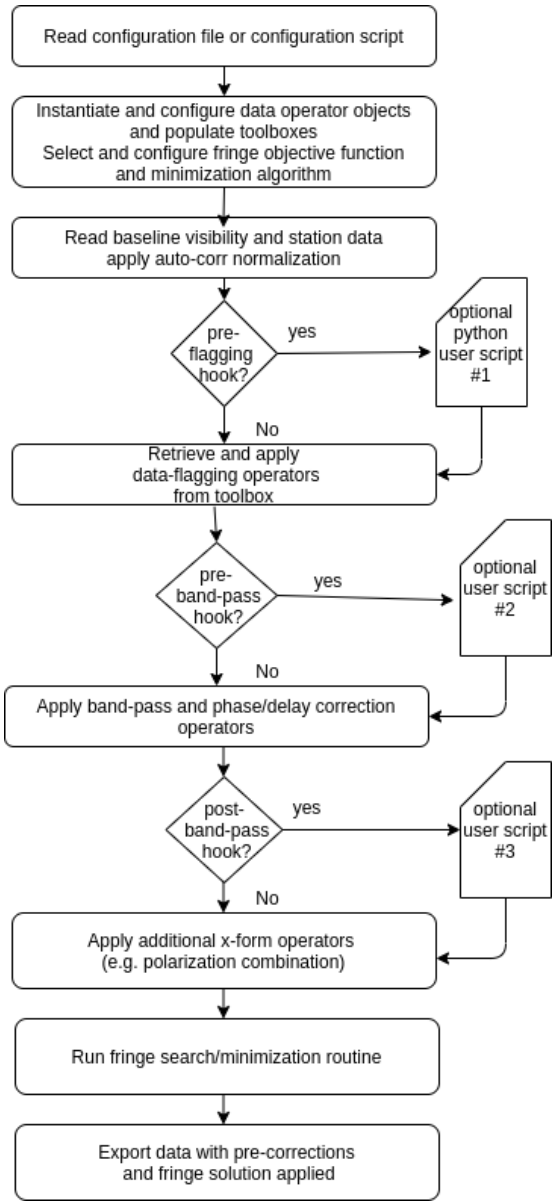
HOPS4 will maintain functionality that replicates the following HOPS3 applications.

1. essential tools:
   1. fourfit - (fringe fitter) - will be completely overhauled
   2. alist - (fringes $\rightarrow$ .alist files) - support version 6
   3. aedit - (manipulator of .alist files) - will be enhanced, with a Python interface
2. additional tools:
   1. average - (slicer/recombiner of alists) - will be reworked
   2. cofit - (coherence estimator) - will be reworked/replaced
   3. fourmer - (fringe joiner) - will be reworked
   4. fplot - (fringe display tool) - will be reworked
   5. search - (for marginal fringes) - will be reworked
   6. CorAsc2 - (data dumper) - used for mk4 regression
3. additional scripting

Again, the philosophy is to preserve the current functionality, refactor the back-end to meet new requirements, and significantly enhance capabilities for expansion.

Simplified Fringe Fitting Architecture

MIT HAYSTACK OBSERVATORY



Example of simplified fringe-fitter control flow.

1. Architecture to support single-baseline or global fringe fitter built from common library components.
2. Data containers and data operators will be as decoupled as possible.
3. Data operators will be configured/instantiated and then inserted into ordered slots
4. Data operator requirements on the data/meta-data will be specified via a schema for key:value retrieval through common interface to keep data contents from affecting the interface. Example: apply phase offset to channel.
5. Provide hooks to insert user python code (direct access to data and instantiated data operators) – important for specific tasks beyond immediate scope of this project (e.g. application of antenna gain cal.)
6. Python interface to data containers is TBD. Trade-offs between ease of use and performance (data-copy) need to be evaluated.

Identified Interfaces

MIT HAYSTACK OBSERVATORY

1. Data movement interfaces:
    1. Data I/O and conversion from correlator DiFX format to HOPS native format.
    2. HOPS native format to disk I/O for persistence.
    3. Import/Export of legacy MK4-types to HOPS native format.
    4. Export of HOPS native format to external archive format (e.g. HDF5/UVFITS).
    5. Reduction of fringe-files to data summary format (alist).
2. User interfaces:
    1. Configuration of fringe-fitting routine (requires python interp. and bindings).
    2. Custom user data-manipulation scripts (requires python bindings). Note this also allows custom data-export routines various points in the code.
    3. User interaction with plotting and data inspection utilities.

Interfaces cont.

MIT HAYSTACK OBSERVATORY

1. Python bindings to executable C/C++ objects/modules in Python (SWIG/pybind11)
2. Python bindings to in-memory data objects to python wrappers (SWIG/pybind11)
3. Conversion/interpretation of legacy control-file format
4. Linking to external dependencies:
    1. FFTW3 (with fallback to native library)
    2. SWIG/pybind11 is only required for developer/maintainers
    3. Doxygen/Sphinx for developer/maintainers
    4. Minimal set of Python required (numpy, matplotlib, ...TBD)

Development Process

MIT HAYSTACK OBSERVATORY

1. Agile philosophy, adapt to additional input as product takes shape
2. Feature additions progress through 5 stages:
    1. concept - what does this feature need to do?
    2. prototype implementation - initial code implementation
    3. integration test - test executable, does it function as needed?
    4. performance evaluation - is feature resource use appropriate?
    5. incorporation - merge back into code-base, put into use
    6. regression - demonstrate that new code matches (or improves on) performance and results of HOPS3
3. Weekly status meetings, otherwise meet when necessary
4. Internal due dates for code changes (weekly or biweekly, depending on feature).
5. Daily testing with CI server, initially to ensure functioning build system, progressing to include full regression and integration tests.
6. Automatic doxygen documentation generation

Development Plan

MIT HAYSTACK OBSERVATORY

1. Iterative process taking a hierarchical approach – build base components first, fill in details and specialized extensions later
2. Code experimentation/testing is key to locating deficiencies in the design as it progresses. Fail fast/early approach
3. Order in which components are developed/introduced depends on:
    1. Is it a core component of basic infrastructure? – e.g. build system, messaging and logging utils etc.
    2. Is it low-level code upon which many libraries/components depend – e.g base classes for data structures, data operators, and utilities, testing infrastructure
    3. Is it nearly independent from other components (no new or only well established deps.) – e.g. python plotting routines, break-out of HOPS3 code into shared libraries

Dev. Plan Roadmap

MIT HAYSTACK OBSERVATORY

1. Write the build system and testing infrastructure
2. Import HOPS3 libraries and 'frozen' executables
3. Implement data containers for visibility and meta-data
4. Convert libraries/executables to import/export visibility and meta-data (DiFX, MK4 formats)
5. Populate the basic data operators (FFT, sum/reduction, functor-broadcast)
6. Populate the minimally necessary (VLBI specific) data operators (e.g. normalization, manual phase-cal, delay-cal) to mimic existing HOPS3 fourfit functionality
7. Implement python bindings to data containers and data operators in parallel with the above.

Dev. Plan Roadmap cont.

MIT HAYSTACK OBSERVATORY

1. Implement minimally-viable grid-search single-baseline fringe-fitter
2. Continue build-out/reuse of rest of current existing fourfit control-file features as data operators
3. Augment with new data operators (fully complex band-pass correction, condition based data flagging, etc.)
4. Implement data-inspection (plotting, alist/aedit) and export tools
5. Implement a global fringe fitting algorithm as an alternative to baseline-based, either as a configurable-option or separate executable (precise algorithm not specified; option to descope)
6. Augment data operators with SIMD parallelism (option to de-scope)
7. Implement distributed (OpenMPI) fringe-fitter (option to de-scope)

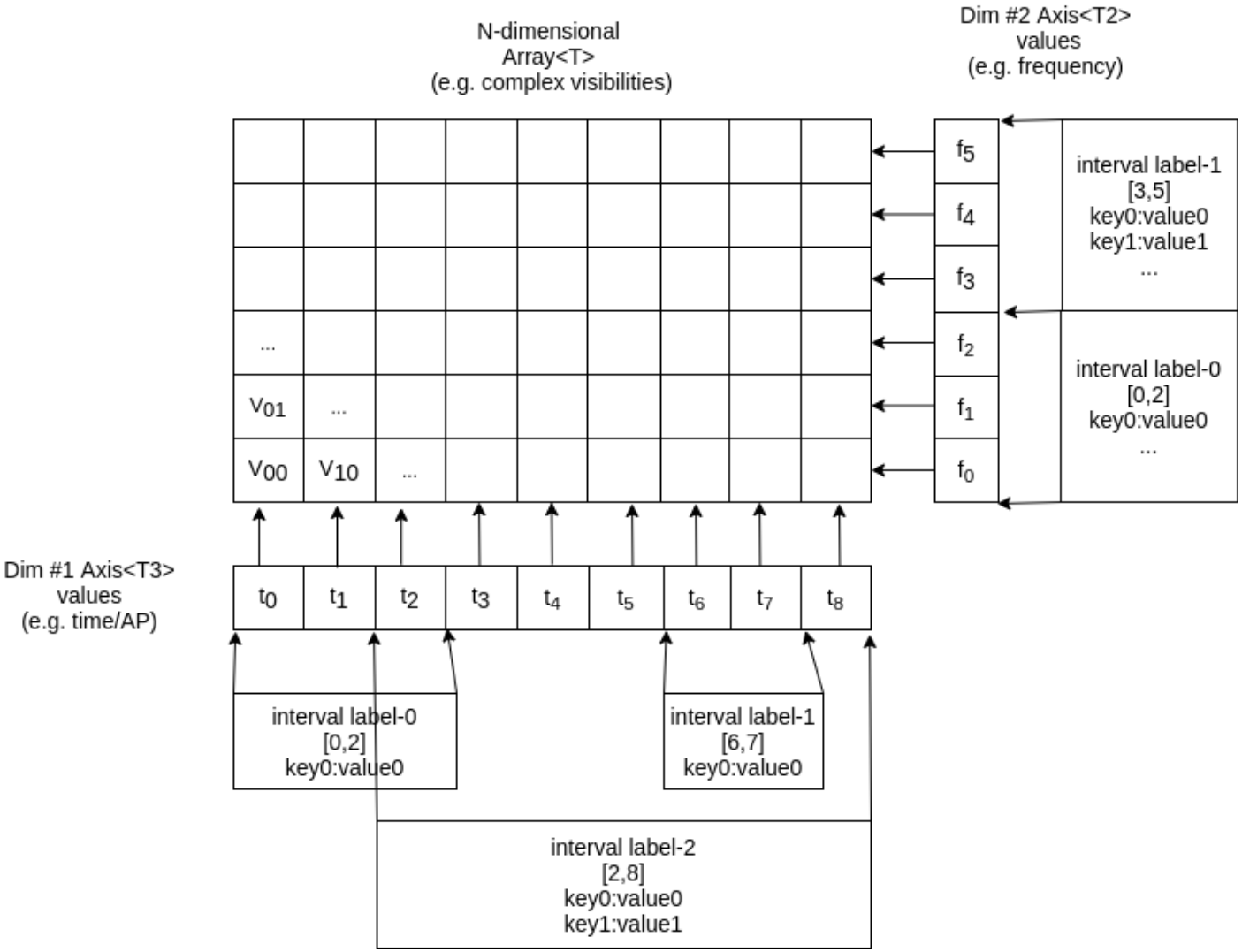
Current Design State

MIT HAYSTACK OBSERVATORY

1. HOPS3 is operational, will be maintained and patched until delivery of HOPS4.
    1. Maintained in SVN; will be "frozen" prior to delivery of HOPS4. Frozen copy will be maintained at Haystack for 4 years (TBD).
    2. The subset of HOPS3 libraries/executables to be retained in HOPS4 will be migrated to the new GIT repository (partially complete)
2. HOPS4 development has started and key technologies have been demonstrated
    1. basic utilities and data container/operator prototypes exist
    2. Mk4-type import/export code in progress
    3. Builds (mostly) with both Autotools and CMake
    4. Organizing bases classes of code in task-specific libraries
    5. Development allows re-use of HOPS3 code in decoupled manner (e.g. MK4Interface makes use of dfio and (new) container library but there is no direct link between the two)
    6. SWIG build infrastructure has been prototyped, pybind11 will also be explored.
    7. Fringe-plot code starting to be prototyped in Python/Matplotlib.

Prototyping Data Container Class Structure

MIT HAYSTACK OBSERVATORY

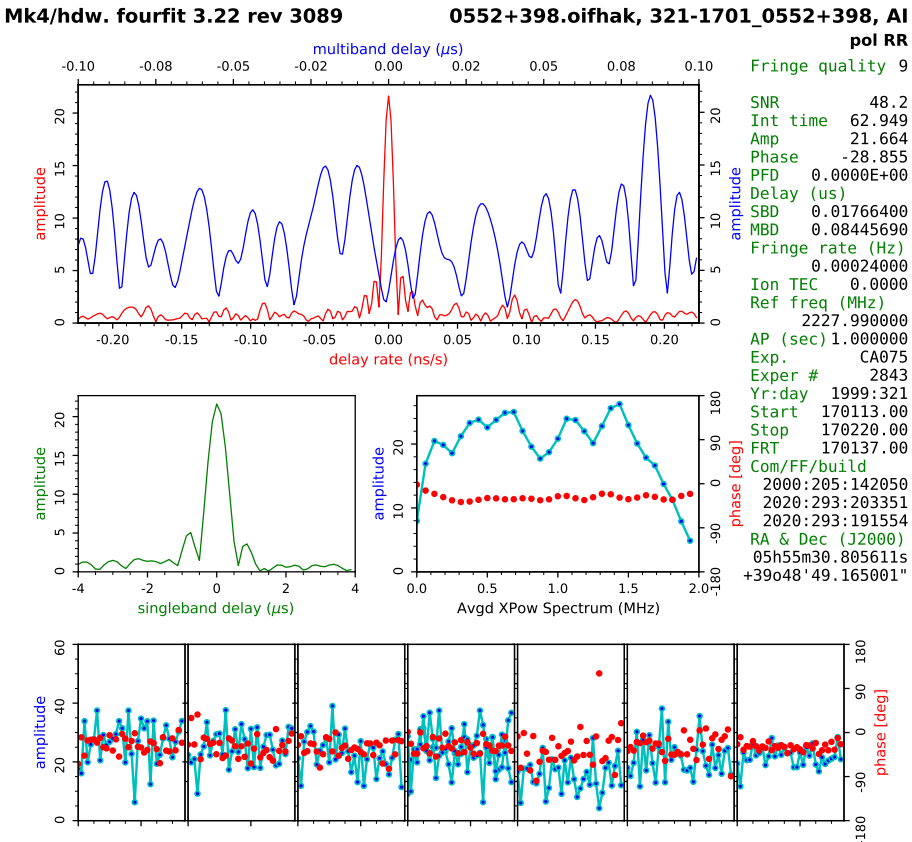


1. Graphical representation current data container prototype code
2. ND-array template class with type and rank parameters + STL-style iterators
3. Axes templated on coordinate type
4. Intervals of array can be labeled with type-agnostic key:value pairs for data selection

Prototyping Plotting Toolkit

MIT HAYSTACK OBSERVATORY

1. PGPLOT is no longer supported and will not be included in the required dependencies for HOPS4
2. Need to build new plotting functionality to distribute with HOPS4:
   1. Separate analysis results from plotting (these are tightly coupled in HOPS3)
   2. Write new plotting routines with Python/NumPy/Matplotlib
   3. Include support for arbitrary plotting tools, hooks for other packages
   4. Develop interactive tools to allow users to reformat plots, explore and select data, flag datapoints, etc.



Example of a fringe plot made with Matplotlib

Testing

MIT HAYSTACK OBSERVATORY

Software testing breaks down into following categories:

1. Unit/Integration - Code changes and new functions/modules will be delivered with appropriate units tests
2. Component - Test the individual task specific libraries/components of HOPS4 functionality as a whole.
3. Performance analysis - A profiling tool will be used to compare the time/memory use of computations in HOPS3 with HOPS4, to identify bottlenecks.
4. Regression - Tests that working code behaves consistently with known results (e.g. from HOPS3) given captured data.
5. Oracle - Where appropriate, HOPS3 will be used as a test oracle for HOPS4 comparison.
6. CI - A continuous integration server will be configured using Github Actions or be made compatible with our existing automated testing scripts.
7. Manual - Verify human-in-the-loop functionality (e.g. interactive tools)

High-level Schedule

MIT HAYSTACK OBSERVATORY

⊟ Documentation	10/1/19	5/31/23
Requirements list	10/1/19	12/1/20
External review of requirements list	12/2/20	12/9/20
Specification document	9/7/20	2/1/21
External review of specification	2/2/21	2/2/21
Help documentation - man pages	10/28/21	5/31/23
Developer guide	10/28/21	5/31/23
User guide	10/28/21	5/31/23
⊟ Code Developement	6/1/20	6/7/23
⊞ Create build system	6/1/20	12/31/20
⊞ Import legacy software	7/30/20	1/1/21
⊞ Develop new data container classes (DCC)	6/1/20	6/3/21
⊞ Develop VLBI data analysis library and executables	6/4/21	5/1/23
⊞ Develop utility libraries	10/1/20	3/5/21
⊞ Develop data operator library	12/14/20	4/22/21
⊞ Develop python bindings/libraries	11/2/20	12/21/22
⊞ Parallel acceleration & Extras	6/4/21	6/7/23

1. Status reports will indicate the readiness of the following:
   1. Mk4-types import/export utility
   2. DiFX import utility
   3. Minimally viable single-baseline fringe fitter $\rightarrow$ complete fringe fitter equivalent to HOPS3 fourfit
   4. Data reduction utilities (alist, aedit, fringe plot)
   5. augmented single-baseline fringe fitter
   6. Data export utilities.
   7. global fringe fitter
2. Tasks proceed largely asynchronously
   1. Can move some tasks in the schedule without affecting others
   2. Interdependency between tasks varies - does not waterfall

Budget

MIT HAYSTACK OBSERVATORY

1. Budget has sufficient margins to complete coding though a rebuilt fringe-fitter and utilities:
   1. DiFX/Mk4 data import/export
   2. fourfit equivalent with new architecture
   3. aedit
   4. alist
   5. data export
2. Resulting HOPS4 will be capable of everything which HOP3 can do now, but be more easily extended by users, and not suffer from artificial bandwidth/AP/station limitations. Goal is to refactor HOPS so it is easily extendable without requiring changes to core libraries.
3. If resources are pressured:
   1. best effort to:
      - Implementing prioritized critical features, e.g.:
      - solving for stationized band-pass correction
      - global fringe fitting algorithm
   2. option to de-scope:
      - SIMD or OpenMPI parallelization

Risk Management

MIT HAYSTACK OBSERVATORY

1. Maintain internal prioritized risk-action list, reviewed in weekly meetings; any team member can raise issue for review.
2. Our risks are primarily *schedule* risks (i.e. developers will have to spend time mitigating them). The costs to mitigate are primarily budgetary ($$ for developers' time).
    1. Issues which rise beyond internal project scope will be raised to monthly ngEHT risk meeting.
    2. Minor software risks (bugs) are mitigated by testing and profiling code early, iterative development
3. Current hi-level risks:
    1. Requirements are incomplete due to unanticipated ngEHT needs or failure of assumptions.
    2. Unexpected bug or structural deficit in new code (amplified if discovered late in process).

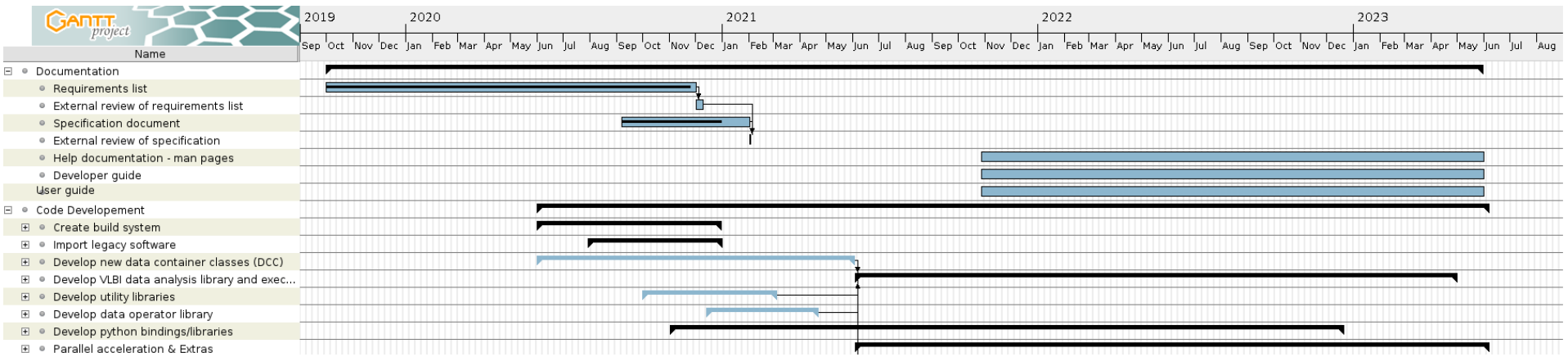
Thanks for listening!

MIT
HAYSTACK
OBSERVATORY

Questions/Comments/Follow-up report?

Backup (Schedule)

MIT HAYSTACK OBSERVATORY



Risk Details - I

MIT HAYSTACK OBSERVATORY

1. Incomplete ngEHT requirements:
   1. Description: The ngEHT requirement set is incomplete and as such our requirements list doesn't capture a critical item desired future data analysis or use cases.
   2. Preventative Mitigation: Keep code as modular as possible, so extensions/restructuring are limited in scope. Keep specialization out of base level libraries:
      - ngEHT may have heterogeneous bandwidth/channels/pols, so we need to ensure early data structure design can accommodate this. Example:
   3. Cost if realized: Code needs to be extended, adapted or redesigned to cover the necessary use case. Other features of project may need to be descoped to free up resources, or other alternatives considered (adoption of 3rd party libraries).
   4. Specific example(s):
      - ngEHT correlator output data format/content is not similar to the EHT and data structures have to be redesigned.
      - ngEHT data sets are too large to fit in commodity computer memory, so distributed computing is necessary and OpenMPI extensions need to be written.

Risk Details - II

MIT HAYSTACK OBSERVATORY

1. Unexpected bug/deficit:
    1. Description: A software deficiency or bug is discovered which significant requires developer resources in order to correct.
    2. Cost if realized: Developer time must be reallocated to correct issue – (same as last slide).
    3. Preventative Mitigation: Continuous testing/comparison to working (HOPS3) code and performance evaluation of features as they are completed. Issue a beta release of software to downstream users before project completes.
    4. Specific example:
        - Downstream users find that Python plugin interface is too slow, code must be refactored to reduce data movement.