3.3 Software design

Jared R. Males

1 Introduction

The MagAO project was constructed around a working AO system, that of the LBT. We re-used the real-time control software as-is, and adapted the AO control software developed by Arcetri (hereafter the AdOpt system) to the Magellan computing environment. In addition, we developed a major extension to the LBTAO baseline by adding the VisAO camera which works seamlessly as part of the MagAO system and routinely and reliably takes science data on the 6.5 m Clay telescope. We also developed an interface for the Clio camera to work as part of MagAO without rewriting the Clio control software itself.

The overall philosophy of MagAO-X software development will closely follow that used for MagAO: we will base it on a working AO control architecture (in this case SCExAO) and adapt it for our use, minimizing truly new software development. We will save significant development time through our use of the same components which are already in use at SCExAO or on the existing MagAO system. These components include:

1. The BMC 2k Deformable Mirror
2. The OCAM-2k EMCCD PWFS detector
3. The PI TTM head
4. Filter wheel motors
5. Tip/tilt stage actuators
6. PI stages

In Section 3.2 we presented the preliminary design of the MagAO-X compute system. This includes the real-time computer (RTC), the instrument control computer (ICC), and the AO operations computer (AOC), as well as workstations in the Clay control room. Here we describe the software preliminary design.

2 Software Management

2.1 Version Control: `git` will be used for version control, with a repository hosted on `github`. The standard "centralized workflow" will be used, where development occurs on local copies, with changes committed to the central repository.

The `git` SHA-1 hash (essentially the version number) will be used as a reproducibility tracer. The SHA-1 of the git repo at the time data or calibrations are taken will be traceable, either via timestamps or (when appropriate) by writing the SHA-1 to metadata. To facilitate this, all processes will have the SHA-1 embedded at compile time and will record this in their log at startup[1]. This ensures that at any time the state of the software system can be recovered if needed to understand data recorded in the past.

Configuration files will also be kept under version control with `git`, and similarly the SHA-1 will be tracked and traceable.

Note: for this system to work, it will be policy that in general no data will be taken with uncommitted changes in the local repository. The user interface will warn when this is occurring (including compiler warnings). This will be strictly enforced on the telescope. Common sense will be allowed during development and lab testing[2].

2.2 Coding Standards and Documentation: The main language used for MagAO-X development will be c++. This is mainly driven by performance, and the PI/Software-lead is proficient in modern c++. The SCExAO real-time code is written in c, so adaptation of that code base for our use will be straightforward. The AdOpt low-level code is also primarily in c, so re-use of various motion control code will be straightforward. The INDI library is also provided in c++.

Python will also be used for scripting and other tasks.

All new code will be documented for processing with doxygen. Doxygen is a well known and maintained code-documentation system. It allows for programmers to document code as they go, with the addition of a few markup symbols. The result is nicely formatted html documentation, with browseable source code, indices, etc, all automatically generated from source. We will also use this to document application interfaces (command line options and configuration file parameters). The VisAO camera control software demonstrates this, `https://visao.as.arizona.edu/software_files/visao/html/annotated.html`, though we expect to improve on the application interface documentation significantly over what is shown there.

A minimum coding standard will be adhered to, which defines such things as header layouts, declare/define standards, documentation conventions, etc. We provide a draft version of this in Appendix A.

3 Computer Design

The MagAO-X computing system includes three custom computers: the Real-Time Computer (RTC), the Instrument Control Computer (ICC), AO Operations Computer (AOC). The specifications and mechanical design of these three computers is presented in Section 3.2. MagAO-X will also make use of the existing workstations in the Magellan Clay control room for science operations (zorro and guanaco).

3.1 Operating System: MagAO-X will standardize on 64-bit CentOS 7, chosen for long term stability. The expected lifetime for CentOS 7 is[3]

- Full Updates: through the end of 2020
- Maintenance Updates: through 2024-June-30.

This will ensure a stable computing environment throughout the development, commissioning, and first four operating years of the instrument.

[1] We already use this technique in data reduction, see this script which creates a header to accomplish it: `https://bitbucket.org/jaredmales/mxlib/src/6aec98c12c7fded062bcff3b7c58402e9ab62cb0/gengithead.sh?at=master&fileviewer=file-view-default`

[2] SHA-1s are free

[3] see `https://wiki.centos.org/About/Product`

CentOS 7 also has the advantage that up-to-date real-time (RT) kernel packages are readily available from the CERN[4] repositories. The RT kernel is used in the existing MagAO system on the VisAO computer, where priorities were optimized for low-latency in several critical processes. The RT kernel will be employed on the RTC and the ICC, and the AOC if needed.

4 Instrument Control

Here we describe our preliminary design for the instrument control software system (ICSS). This encompasses control of the various stages and motors, the science cameras, and high level AO loop control (stop/start, status monitoring etc).

4.1 INDI: We will employ the Instrument Neutral Distributed Interface (INDI) protocol (Downey, 2007) for communication between the various components of the ICSS. INDI is now the de-facto standard withing the Center for Astronomical Adaptive Optics (CAAO), where it is used for the LBTI control software and is an integral part of the planned MMTAO upgrade (Milton, 2017). Using it has the advantage that one of the main developers of INDI is a member of CAAO making support readily available.

INDI essentially replaces the real-time-database (RTDB) and message daemon (MsgD) middle-ware in the AdOpt architecture. The basic architecture is that INDI devices communicate with a simple protocol via an INDI server on the host machine, see Figure 1, left. INDI servers are connected over the network, providing communications between machines. A very nice CGI interface is possible, which will provide a light-weight interface for astronomers to use, see Figure 1, right.

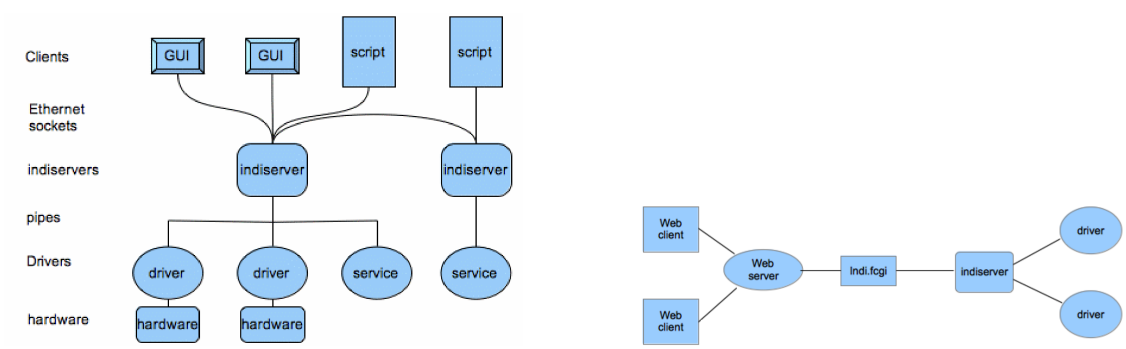


Figure 1: The INDI architecture. Source: the INDI wiki.

.

For MagAO-X, each of the RTC, ICC, and AOC will have an INDI server running and communicating with the others. On RTC INDI will provide for monitoring the status of the AO loop, high level AO control (start/pause/stop, etc), and show component status (PWFS camera, DMs). On the ICC INDI will be used to control the various stages, motors, and mechanisms, control and interact with the science cameras, and monitor the status of the LOWFS loop. On the AOC, an INDI device will interface to the TCS. The AOC INDI server will support the AO Operations Interface, and through the fast-CGI capability and a web server provide the astronomer's interface. The MagAO-X INDI architecture is shown in Figure 2.

[4] `ttp://linux.web.cern.ch/linux/rt/`

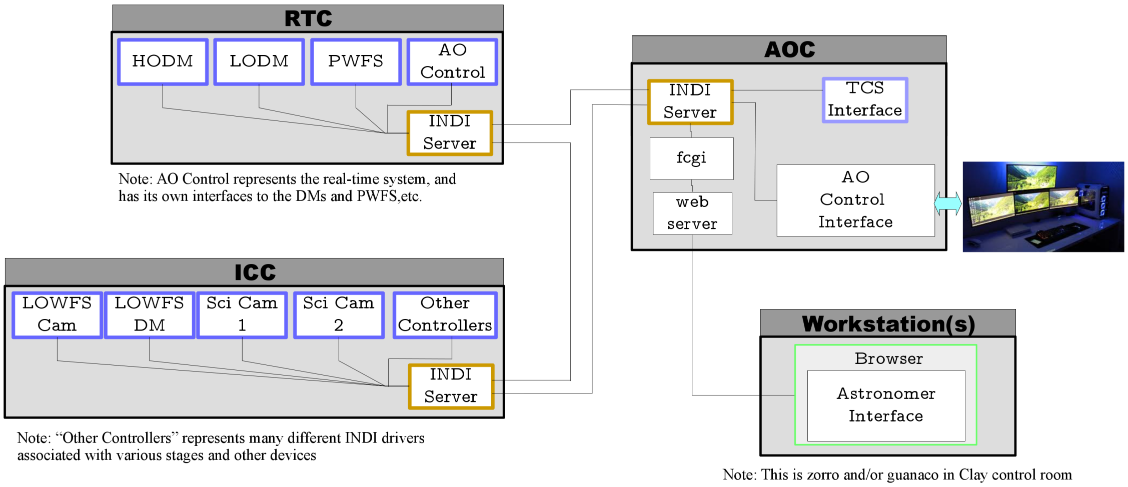


Figure 2: The MagAO-X INDI architecture. Purple borders indicate INDI device, gold borders are INDI servers. Note that several drivers are not shown on each machine, such as housekeeping and telemetry drivers which will publish properties.

.

4.2 MagAOXApp: The class defining an application in the MagAO-X ICSS will be derived from a standard class called `MagAOXApp`. Similar to the `AOApp` and `VisAOApp` base class in the AdOpt and VisAO systems, this will provide a standard set of functionality. This will include the INDI driver and client facilities, configuration, logging, and management of real-time priority.

4.3 Configuration: For MagAO-X we will use ini-style configuration files. This a standard format using key=value pairs and allows sections. For example

```
[basic]
name=The Name
rt_priority=0

[section2]
avector=0,3,5,6,3
```

Each derived class is responsible for knowing the intended type of each value. A template-based configuration parsing system will be used for ease of coding.

The MagAOXApp will employ a cascaded configuration system. At startup, the application will configure itself using the following sources in order

1. Default configuration [compiled in]
2. System Global configuration [set by environment variable, common to all MagAOXApp processes]
3. App global configuration [location set by environment variable, name compiled in]
4. Command line specified configuration file.

Only the default configuration needs to exist. Each level overrides the previous. By specifying the location of configuration files via environment variables, we will have a straightforward way to maintain several configurations (e.g. for lab, cleanroom, and telescope).

4.4 Interprocess Communication: Non-real-time interprocess communication will generally take place via the INDI protocol. Any process which needs the status of another will subscribe to the appropriate property. An example is a focus stage which may need to know the position of several filter wheels in order to go to the correct position. This will also suffice for such things as AO setup (i.e. which reconstructor to load could depend on beamsplitter selection, among other things).

Real-time IPC on the ICC will make use of shared memory and semaphores. For instance, the science camera controllers will notify the framegrabber process for that camera when the format has changed via a semaphore, cuing the framegrabber process to read the shared memory buffer containing the configuration details. The framegrabber in turn notifies the frame-writer process every time a new image is ready to write to disk.

The main AO control real-time software is described below.

4.5 Logging: Event logging is a crucial facility for a system such as MagAO-X. Here we include recording specific events (“loop closed”) as well as telemetry such as WFS images and telescope position. These data will be used for system performance analysis and diagnosis, and perhaps more importantly for data reduction. Given our goal of recording all system data for future use in data reduction, we want to have a very efficient log system.

A lesson learned from the AdOpt system is that ASCII logs can use a great deal of disk space over time — especially when things going wrong causes frequent logging, the time when we least want to be managing disk space. Furthermore, having a somewhat rigid log structure should be more efficient for later analysis. In MagAO-X we will address these issues by logging only an an event code and a time-stamp, along with data of known format based on the event code, all in binary. That is log files will not be easily human-readable as stored on disk.

The event code is a 32-bit fixed width unsigned integer, `uint32_t`. This gives 4,294,967,296 independent event codes, which we assess is more than enough.

The time-stamp will be stored as two fixed-width integers, where the first `int64_t`holds the whole seconds since the Unix epoch and the second `int32_t`holds the nanosecond. This is the `timespec` structure, except we are explicit about the integer width (another lesson learned from AdOpt, where the `timeval` structure was a source of 32/64 compatibility problems, though we are unlikely to use 32-bit systems).

So a time stamp consists of a minimum of 12 bytes. For comparison logging to nanosecond precision with ASCII requires a string of the form YYYYMMDDHHMMSSNNNNNNNNN, which is 23 bytes. This could be hex encoded, say, reducing it to 12 bytes. We could use a 32-bit unsigned integer for the seconds field, but this would reduce compatibility with the standard library on 64-bit linux[5]. Consider, though, that MagAO-X has a maximum operating frequency of 3700 Hz. Assume that an average of 10 events are logged each timestep (a conservatively high number). The extra 4 bytes then amounts to 3700*10*4 = 16 kB/sec = 5 GB/10-hrs. This is a relatively small overhead compared to the several TB of image data we will record in the same time and so we consider it negligible.

Human readable logs in ASCII would take the form:

```
YYYY-MM-DDTHH:MM:SS.SSSSSSSSS INF loop closed\n [ 46 bytes ]
```

[5]and place us at risk of the Unix millenium bug in 2038, or 2106 if we used unsigned integers. We may not operate for that long though, so this if of minor concern.

```
YYYY-MM-DDTHH:MM:SS.SSSSSSSSS ERR unable to connect\n [ 52 bytes ]
YYYY-MM-DDTHH:MM:SS.SSSSSSSSS INF telra 12:00:00.00\n [ 52 bytes ]
```

In contrast, these three log entries will take 16, 16, and 24 bytes respectively, or 37% of the disk space.

Logs will then be saved as binary records in HDF5 files on a per-process basis. For efficient access, these records will be have a maximum length (to be optimized) and have the timestamp of the first entry recorded as an attribute in the file. Parsing each record will require determining the type from the event code and calling an appropriate function (with a pointer to the entry as argument) to read the data.

Simple utilities will display logs in human-readable format as needed (i.e. a replacement for tools like cat).

Using c++ templates we will provide a very simple logging interface within the code. A sketch of how this will work is shown in Appendix B.

A drawback to this system will be the overhead of creating a new event code. This overhead will be paid during development, every time a new log entry is needed. The minimum steps to create a new log entry will include:

1. declare log entry structure containing the event code
2. define `length()` member of the struct
3. declare and define a specialized `log<>()` function to do the work of logging.

In general this will also necessitate updating the log parsing facility to handle this new event type. A database of event codes will also be maintained automatically with a code analyzer minimizing effort to safely generate a new one.

4.6 RT Priority: MagAOXApp based processes will have the ability to set their real-time (RT) priority. This will be determined by a configuration setting, allowing for optimization. This requires installing processes with mode 4755. Upon startup, processes will immediately decrease privileges to the lowest setting, and only increase privileges to set RT priority. The MagAOXApp will do this by default during construction [6].

5 TCS Interface

The instrument-TCS interface at Magellan is well documented in Eychaner (2015). Instruments connect via a TCP/IP socket and send and receive formatted ASCII. We have already implemented a process called the MagAOI (for MagAO Interface) which handles TCS queries at 1 Hz. This retrieves all data available from the TCS which is relevant to the MagAO system. We will adapt this code to work as an MagAOXApp (and INDI Driver and Client) and use it to manage interfacing with the TCS. In Appendix C we list all the TCS parameters which will be used.

As part of the MagAO test system, we developed a software TCS simulator which will be used for MagAO-X software development and lab testing.

6 User Interfaces

Based on long experience using it on the MagAO project, we plan to avoid running GUIs via x-forwarding on ssh. There are two main GUIs to be provided.

[6] See VisAO_base() at `https://visao.as.arizona.edu/software_files/visao/html/VisAOApp__base_8cpp_source.html` for a working example

6.1 The Astronomer Interface: Following typical practice in CAAO INDI implementations (such as at the LBTI), astronomers will interact with the instrument using a GUI implemented using the jQuery UI framework, and running on a web browser. This will be served by a web server running on the AOC, connected to the AOC INDI server via the W3 fcgi protocol. This will be very flexible, allowing astronomers to use the workstations in the Clay control room with minimal fuss.

We will also provide support for observation scripting through this interface.

6.2 The AO Operator Interface: Experience on MagAO has shown that reliable high-speed display of AO status, including PWFS images and DM commands and positions, is extremely helpful in optimizing AO performance. To that end, we will implement a custom AO interface served on the AOC. It will make use of 4 monitors, and its organization will be optimized for ease of use. For instance, all buttons needed to operate the AO system will be located on a single pane – it will not be necessary to switch tabs or windows while operating the loop. Where appropriate, this may also make use of a web-browser interface (likely re-using code from the Astronomer Interface). Where needed, compiled Qt will be used for high performance.

To support the reliable high-speed AO updates, we will send telemetry and diagnostic data from the RTC and ICC to the AOC on an *as-displayable* basis. For instance, it is typically only possible to display PWFS images at ∼30 Hz. In this case, a decimator process on the RTC will send frames on only 30 fps to the AOC. This will minimize network traffic, and processing time devoted to sending telemetry.

7 Real-time Software

For the real-time control of the AO loops we will use the RT software (RTS) developed by MagAO-X Co-PI Olivier Guyon for the Subaru SCExAO instrument. It is Linux-based, open-source C code along with high-level scripts. It uses publicly available libraries, including CUDA and MAGMA (for GPU computing), FFTW, FISIO, GSL. The source code is available at `https://github.com/oguyon/AdaptiveOpticsControl`.

Because we are using essentially identical hardware to SCExAO (BMC 2k MEMS and OCAM-2K EMCCD) we save significant development time in implementing our RTS. Here we provide a very brief overview of the highlights of this system. More details are given in Appendix D to this Section.

7.1 Performance on Hardware: The RTS runs on a single multi-core computer. Minimum 15 cores system, 128GB ram (heavy use of shared memory and shielded processes running on single core). Supports NVIDIA hardware (CUDA lib). Interfaces to hardware through shared memory structure. Hardware already coupled with RTS: BMC deformable mirror, Ocam2k camera, SAPHIRA camera (with UH readout electronics), OwlCam InGaAs Raptor Photonics camera, Andor sCMOS.

7.2 Speed: Largely limited by hardware. Fully system timing stable at 10us level, and RTS latency due to IPC, TCP transfers between computers, and GPU transfers is $< 100\ \mu$s total, so it can drive a ∼10 kHz loop on multi-computer system, and ∼20 kHz loop on single computer. SCExAO implementation drives 2000-actuator, 14,400-sensor loop at 3.5kHz, limited by camera readout speed.

7.3 Flexible architecture: All input, output and intermediate data is stored as shared memory. A common format for all shared memory data streams facilitates software development. Multiple processes run simultaneously to perform operations on shared memory streams. Additional processes can be deployed (for example, real-time analysis of an intermediate data stream) without impacting existing processed.

IPC is built in the shared memory structure which contains POSIX semaphores (default of 10 semaphores, more if needed): 10 different processes can run on the same input. Each process waits on input stream(s), and posts output stream(s) semaphore(s), so real-time operations can be chained, with multiple branches.

References

Downey, E. C. 2007, 755, L28

Eychaner, G. 2015, Instrument Communication with the Magellan Telescopes, Tech. rep.

Milton, M. 2017, MMT AO ASM Upgrade Software Architecture, Tech. rep.

A Coding Standards

Here we show some sketches of our standard coding practices, including use of doxygen comments.

```
///Brief description for one parameter function
/** Long description
  *
  * \returns functions should return 0 on success, and a negative integer to
      indicate error.
  *
  * \tparam T document the type here.
  */
template<typename T>
int aFunction( T & param /**< [in/out] documentation for param*/ )
{
   //code goes here.

   return 0; ///\retval 0 on success.
}

///Brief description for two or more parameter function
/** Long description
  *
  * \returns functions should return 0 on success, and a negative integer to
      indicate error.
  *
  * \tparam T1 document the type of param1 here.
  * \tparam T2 document the type of param2 here.
  */
template<typename T1, typename T2>
int aFunction( T1 & param1 //< [out] documentation for param1, an output
               T2 & param2 //< [in] documentation for param2, an input
             )
{
   //code goes here.

   return 0; ///\retval 0 on success.
}

///Brief description for a class
/** Long description
  *
  * \tparam _T document type _T/T here.
  */
template<typename _T>
class aClass
```

```
{
public:
  typedef _T T; ///< public typedefs first, with documentation. All template
     parameters typdef-ed as shown.

  aClass(); ///<Default c'tor

  ~aClass(); ///<Destructor.

protected:
  typeT member1; ///<Document protected members.

public:

  int actionFoo( T & inPlace /**< [in/out] parameter documentation*/ );

  int actionfoo( T & after, ///< [out] parameter documentation
                 T & before ///< [in] parameter documenation
               );

}
```

B Logging Code Sketch

Here we sketch the logging framework.

```
namespace logger
{

struct timespecX
{
   int64_t time_s;
   int32_t time_ns;
};

//Logger events are declared:
struct loop_closed
{
   const uint32_t eventCode = 1000;

   void length( uint32_t * logPtr /**< A pointer to a log entry, in this case not
       used */ )
   {
      return 12;
   }
};

struct tel_pos
{
   const uint32_t eventCode = 103458;

   void length( uint32_t * logPtr /**< A pointer to a log entry, in this case not
       used */ )
   {
      return 12 + 8 + 7 + 6 + 6 + 4 + 7; // the size of the TCS responses, after
         '.' is removed.
   }

};

struct user_log
{
   const uint32_t eventCode = 38958;
};

//etc...

//And template specializations of the log function:
template<typename logT>
```

```
void log();

///Log specializaton for the loop closed event
template<>
inline void log<loop_closed>()
{
   // Step 1: get timestamp
   // Step 2: format and store log
}

///Log specialization for telescope position
template<>
inline void log<tel_pos>( char[8] telra, ///< Telescope RA as returned by TCS,
    with '.' removed
                          char[7] teldc, ///< Telescope Dec as returned by TCS, with '.'
                              removed
                          char[6] telep, ///< Telescope Equinox as returned by TCS, with
                              '.' removed
                          char[6] telha, ///< Telescope HA as returned by TCS, with '.'
                              removed
                          char[4] telam, ///< Telescope Airmass as returned by TCS, with
                              '.' removed
                          char[7] rotangle ///< Telescope rotator angle as returned by
                              TCS, with '.' removed
                        )
{
   // Step 1: get timestamp
   // Step 2: format and store log
}

///Log specializaton for the loop closed event
template<>
inline void log<user_log>( const std::string & fromUser )
{
   // Step 1: get timestamp
   // Step 2: format and store log
   // Note: here the format must include a string length.
}


//etc...

}; //namespace logger
```

And then within the code itself entries such as

```
using namespace logger;
```

```
log< loop_closed >();

log< tel_pos >( telra, teldc, telep, telha, telam, rotangle );

//User enters a log from a GUI:
// std::string fromUser <--- "Photometric conditions"

log< user_log >( fromUser );
```

C TCS Parameters

Here we collect the various telescope and environment parameters which will be queried and logged.

Table 1: Telescope and Environment Parameters

TCS Name	TCS Format	Stored As	Size [Bytes]	Rate [Hz]	Notes
Telescope Position					
dateobs	YYYY-MM-DD	char[8]	8	0.1	UT date in year month day format.
ut	HH:MM:SS	char[6]	6	1	UT time in hours minutes and seconds.
st	HH:MM:SS	char[6]	6	1	Sidereal time in hours minutes and seconds.
ra	HH:MM:SS.SS	char[8]	8	1	Right ascension in hours, minutes, and seconds.
dec	DD:MM:SS.S	char[7]	7	1	Declination in degrees, minutes, and seconds.
epoch	YYYY.YY	char[6]	6	1	Equinox of current telescope coordinates.
ha	HH:MM:SS	char[6]	6	1	Hour angle in hours, minutes, and seconds.
airmass	A.AAA	char[4]	4	1	Observational airmass.
telaz	AAA.AAAA	char[7]	7	1	Azimuth angle, in degrees.
telel	EE.EEEE	char[6]	6	1	Elevation angle, in degrees.
zd	ZZ.ZZZZ	char[6]	6	1	Zenith angle, in degrees.
telpa	PPP.PPPP	char[7]	7	1	Parallactic angle, in degrees.
teldm	DDD	char[3]	3	1	Dome azimuth angle, in degrees.
dmstat	DD	char[2]	2	1	Dome status (0 = closed; 1 = open; -1 = unknown)
telguide	ab	char[2]	2	1	a: 0 = not tracking, 1 = tracking; b: guider number of active guider, or 0 if not guiding
gdrmountmv	abc	char[3]	3	1	Telescope and guider motion status (see below)
mountmv	abcd	char[4]	4	1	Telescope and rotator motion status flags (see below)
telfocus	FFFFFF	char[6]	6	1	Secondary mirror focus (Z axis) set (instrument) offset, in microns.
vefocus	FFFFFF	char[6]	6	1	Secondary mirror focus (Z axis) encoder reading, in microns.
vezima	FFFFFF	char[6]	6	1	Secondary mirror Z axis ima (Shack-Hartmann) offset, in microns .
vezpsn	FFFFFF	char[6]	6	1	Secondary mirror Z axis psn (flexure) offset, in microns.
vexset	FFFFFF	char[6]	6	1	Secondary mirror X axis set (instrument) offset, in microns.
vexenc	FFFFFF	char[6]	6	1	Secondary mirror X axis encoder reading, in microns.
vexima	FFFFFF	char[6]	6	1	Secondary mirror X axis ima (Shack-Hartmann) offset, in microns
. vexpsn	FFFFFF	char[6]	6	1	Secondary mirror X axis psn (flexure) offset, in microns.
veyset	FFFFFF	char[6]	6	1	Secondary mirror Y axis set (instrument) offset, in microns.
veyenc	FFFFFF	char[6]	6	1	Secondary mirror Y axis encoder reading, in microns.
veyima	FFFFFF	char[6]	6	1	Secondary mirror Y axis ima (Shack-Hartmann) offset, in microns.
veypsn	FFFFFF	char[6]	6	1	Secondary mirror Y axis psn (flexure) offset, in microns.
vehset	FFFFFF.FFF	char[9]	9	1	Secondary mirror H axis (rotation) set (instrument) offset, in arcseconds.
vehenc	FFFFFF.FFF	char[9]	9	1	Secondary mirror H axis (rotation) encoder reading, in arcseconds.
vehima	FFFFFF.FFF	char[9]	9	1	Secondary mirror H axis (rotation) ima (Shack-Hartmann) offset, in arcseconds.
vehpsn	FFFFFF.FFF	char[9]	9	1	Secondary mirror H axis (rotation) psn (flexure) offset, in arcseconds.
vevset	FFFFFF.FFF	char[9]	9	1	Secondary mirror V axis (rotation) set (instrument) offset, in arcseconds.
vevenc	FFFFFF.FFF	char[9]	9	1	Secondary mirror V axis (rotation) encoder reading, in arcseconds.
vevima	FFFFFF.FFF	char[9]	9	1	Secondary mirror V axis (rotation) ima (Shack-Hartmann) offset, in arcseconds.
vevpsn	FFFFFF.FFF	char[9]	9	1	Secondary mirror V axis (rotation) psn (flexure) offset, in arcseconds.
telroi	R	char[1]	1	0.1	Rotator of interest (0 to 5 are NASW, NASE, CASS, AUX1, AUX2, and AUX3 respectively).
rotmode	R	char[1]	1	0.1	Rotator tracking mode; normally either 0 (OFF; no tracking) or 2 (EQU; equatorial tracking, rotator tracks sky).
rotangle	RRR.RRRR	char[7]	7	1	Current rotator offset angle, in degrees.
nrotoff	RRR.RRRR	char[7]	7	1	Angle between rotator zero and sky north for input coordinates and rotator offset, in degrees.
rotatore	RRR.RRRR	char[7]	7	1	Current rotator encoder angle, in degrees.
User Catalog Input					
catra	HH:MM:SS.SS	char[8]	8	0.1	Current catalog object right ascension.
catdc	DD:MM:SS.S	char[7]	7	0.1	Current catalog object declination.
catep	YYYY.YY	char[6]	6	0.1	Current catalog object equinox.
catro	RRR.RRRR	char[7]	7	0.1	Current catalog object rotator offset angle, in degrees.
catrm	TTT	char[3]	3	0.1	Current catalog object rotator offset mode; one of OFF, EQU, GRV, or HRZ.
catobj	string	char[30]	30	0.1	Current catalog object name (up to 30 characters, containing no spaces).
Environment					
fwhm	FF.FF	char[4]	4	0.1	30-second average FWHM value from the active guider.
dimmfw	FF.FF	char[4]	4	0.1	DIMM seeing, available from wx database, not TCS.
mag1fw	FF.FF	char[4]	4	0.1	Baade seeing, available from wx database, not TCS.
wxtemp	TTT.TT	char[5]	5	0.1	Outside temperature (degress Celcius).
wxpres	PPPP.PP	char[6]	6	0.1	Outside pressure (millibars).
wxhumid	HHH.HH	char[5]	5	0.1	Outside humidity (percent).
wxwind	VVV.VV	char[5]	5	0.1	Outside wind intensity (mph).
wxwdir	DDD.DD	char[5]	5	0.1	Outside wind direction (degrees).
wxdewpt	TTT.TT	char[5]	5	0.1	Outside dewpoint (degress Celcius).
ttruss	TT.TTT	char[5]	5	0.1	Telescope truss temperature (degress Celcius).
tcell	TT.TTT	char[5]	5	0.1	mirror cell temperature (degress Celcius).
tseccell	TT.TTT	char[5]	5	0.1	Secondary mirror cell temperature, skyward side (degress Celcius).
tambient	TT.TTT	char[5]	5	0.1	Dome air temperature (degress Celcius).
tair	TT.TTT	char[5]	5	0.1	Primary mirror air temperature (degress Celcius).

3.3 Software
Appendix D

SCExAO Real-Time Architecture

AO Loop Control Software

Overview

Linux-based

Open-source, no closed library

C code (~100k lines) + high-level scripts (baseline control interface using bash scripts provided)

Uses libraries: CUDA & MAGMA (GPU computing, optional), FFTW, FITSIO, GNU scientific library, readline

Source code + example simulated AO system:
https://github.com/oguyon/AdaptiveOpticsControl

Hardware

Hardware Requirements / compatibility:

RTS runs on a single multi-core computer. Minimum ~15 cores system, 128GB ram (heavy use of shared memory and shielded processes running on single core)

CPU only or CPU+GPU computing engine. Requires GPU(s) for high speed / high actuator count. Supports NVIDIA hardware (CUDA lib).

Can span multiple computers (for example, camera or DM driven by computer other than main RTS). Software uses and configures fast private low-latency TCP link (eg. 10GbE or 40GbE fibers) for transfers.

Interfaces to hardware through shared memory structure. Hardware already coupled with RTS: BMC deformable mirror, Ocam2k camera, SAPHIRA camera (with UH readout electronics), OwlCam InGaAs Raptor Photonics camera, Andor sCMOS.

Capabilities

Speed

Largely limited by hardware. Fully system timing stable at 10us level, and RTS latency due to IPC, TCP transfers between computers, and GPU transfers is <100us total → can drive ~10 kHz loop on multi-computer system, and ~20 kHz loop on single computer.
SCExAO implementation drives 2000-actuator, 14,400-sensor loop at 3.5kHz, limited by camera readout speed.

Flexible architecture

All input, output and intermediate data is stored as shared memory. A common format for all shared memory data streams facilitates software development. Multiple processes run simultaneously to perform operations on shared memory streams. Additional processes can be deployed (for example, real-time analysis of an intermediate data stream) without impacting existing processed.

IPC is built in the shared memory structure which contains POSIX semaphores (default of 10 semaphores, more if needed): 10 different processes can run on the same input. Each process waits on input stream(s), and posts output stream(s) semaphore(s) → Real-time operations can be chained, with multiple branches

Example control GUI (bash scripts)

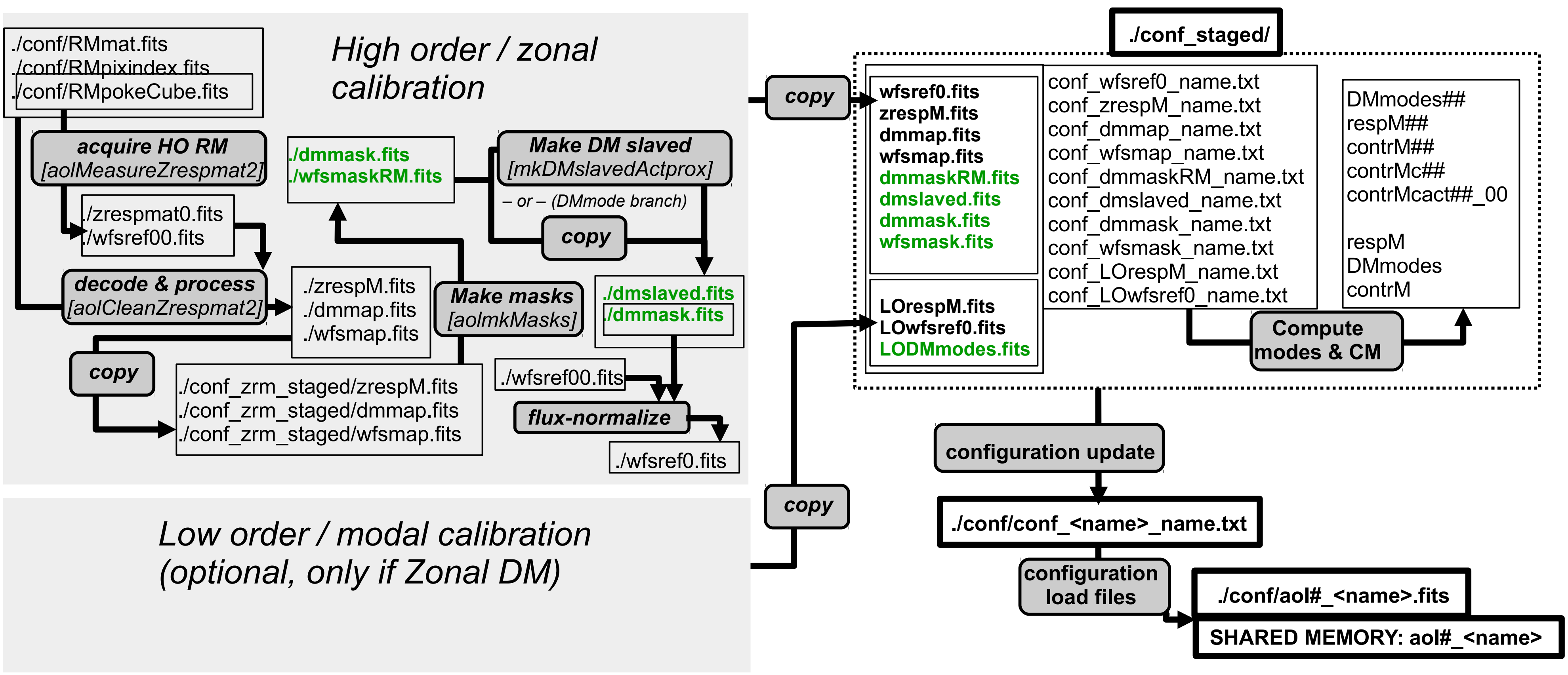
For each file:
conf_<name>_name.txt points to archived file location

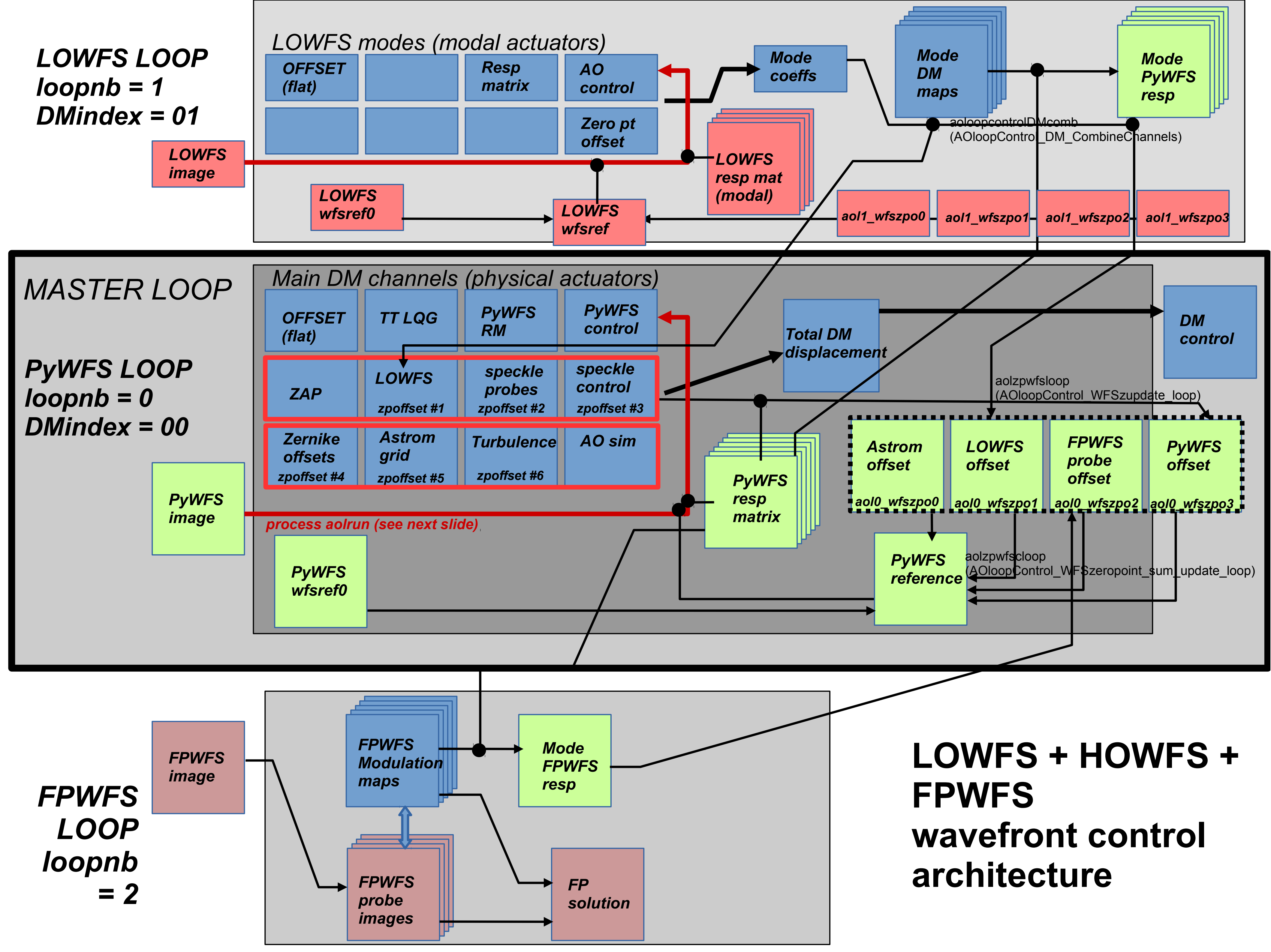
conf/conf_<name>_name.txt are read by function ReadConfFile for loading into shared memory and FITS copy to ./conf/aol#_<name>.fits

Calibration Work Flow

Conventions :
Modal DM: “actuators” indices have no spatial meaning
→ No spatial filtering options
→ “Direct write” CM and “Modal” CM are the same (1 mode = 1 actuator)
Zonal DM: actuator indices correspond to spatial coordinates
→ Need linear transformation between mode coefficients and actuators

If re-using masks, keep from previous calibration





LOWFS LOOP
loopnb = 1
DMindex = 01
LOWFS modes (modal actuators)
OFFSET (flat)
Resp matrix
AO control
Zero pt offset
Mode coeffs
Mode DM maps
Mode PyWFS resp
aoloopcontrolDMcomb
(AOloopControl_DM_CombineChannels)
LOWFS image
LOWFS resp mat (modal)
LOWFS wfsref0
LOWFS wfsref
aol1_wfszpo0
aol1_wfszpo1
aol1_wfszpo2
aol1_wfszpo3
MASTER LOOP
PyWFS LOOP
loopnb = 0
DMindex = 00
Main DM channels (physical actuators)
OFFSET (flat)
TT LQG
PyWFS RM
PyWFS control
ZAP
LOWFS zpoffset #1
speckle probes zpoffset #2
speckle control zpoffset #3
Zernike offsets zpoffset #4
Astrom grid zpoffset #5
Turbulence zpoffset #6
AO sim
Total DM displacement
DM control
aolzpwfsloop
(AOloopControl_WFSzupdate_loop)
Astrom offset aol0_wfszpo0
LOWFS offset aol0_wfszpo1
FPWFS probe offset aol0_wfszpo2
PyWFS offset aol0_wfszpo3
PyWFS image
process aolrun (see next slide)
PyWFS resp matrix
PyWFS wfsref0
PyWFS reference
aolzpwfscloop
(AOloopControl_WFSzeropoint_sum_update_loop)
FPWFS LOOP
loopnb = 2
FPWFS image
FPWFS Modulation maps
Mode FPWFS resp
FPWFS probe images
FP solution
LOWFS + HOWFS + FPWFS
wavefront control architecture

Control Matrix Computation Modes

WFSnorm *(./conf/conf_WFSnormalize.txt)* ***WFS normalization mode*** C code: AOconf[loop].WFSnormalize

0: Do not normalize WFS images

1: Normalize WFS images

WFSnorm should be left unchanged between RM acquisition and Loop control

DMprimaryWrite (./conf/conf_DMprimWriteON.txt) ***DM primary write*** C code: DMprimaryWrite_ON

0: DM primary write is off

1: DM primary write is on

CMmode *(./conf/conf_CMmode.txt)* ***Combined Control matrix mode*** C code: MATRIX_COMPUTATION_MODE

0: not combined: control matrix is WFS pixels → modes

→ Linking aol#_DMmode_meas ↔ aol#_modeval

→ modesextractwfs reads from aol_DMmode_meas instead of computing

1: combined: control matrix is WFS pixels → DM actuators

DMMODE *(./conf/conf_DMMODE.txt)* ***DM mode (zonal vs. modal)*** Bash script only, only affects bash scripts and options

ZONAL: pixel coordinates correspond to DM actuators physical location

→ spatial filtering enabled for DM modes creation

→ blocks built by spatial frequencies, user can set independent gain values for mode blocks

MODAL: DM pixels correspond to abstract modes

→ no spatial filtering, setting 1 block only

Note: **DMMODE**=ZONAL → **CMMODE**=MODAL (CPA-splitting of modes into blocks)

GPUmode *(./conf/conf_GPUmode.txt)* ***# of GPUs to use for CM multiplication*** C code: AOconf[loop].GPU

0: use CPU

>0: number of GPUs

if CMmode=1 and GPUmode>0:

GPUallmode *(./conf/conf_GPUall.txt)* ***Use GPU for all computations*** C code: AOconf[loop].GPUall = COMPUTE_GPU_SCALING

0: Use CPU for WFS reference subtraction and normalization

→ WFS reference subtraction and normalization done by CPU (imWFS0→ imWFS1→ imWFS2)

→ CM multiplication input is imWFS2 (GPU or CPU)

1: Use GPU for all computation

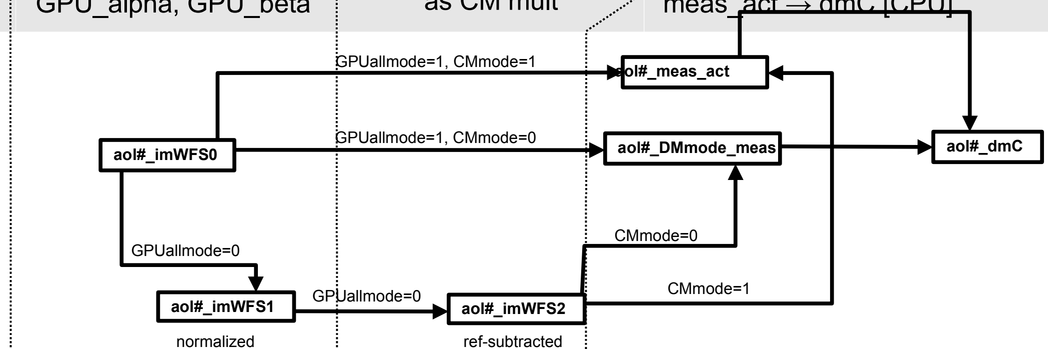
→ WFS reference subtraction and normalization done by GPU

→ GPU-based CM multiplication input is imWFS0

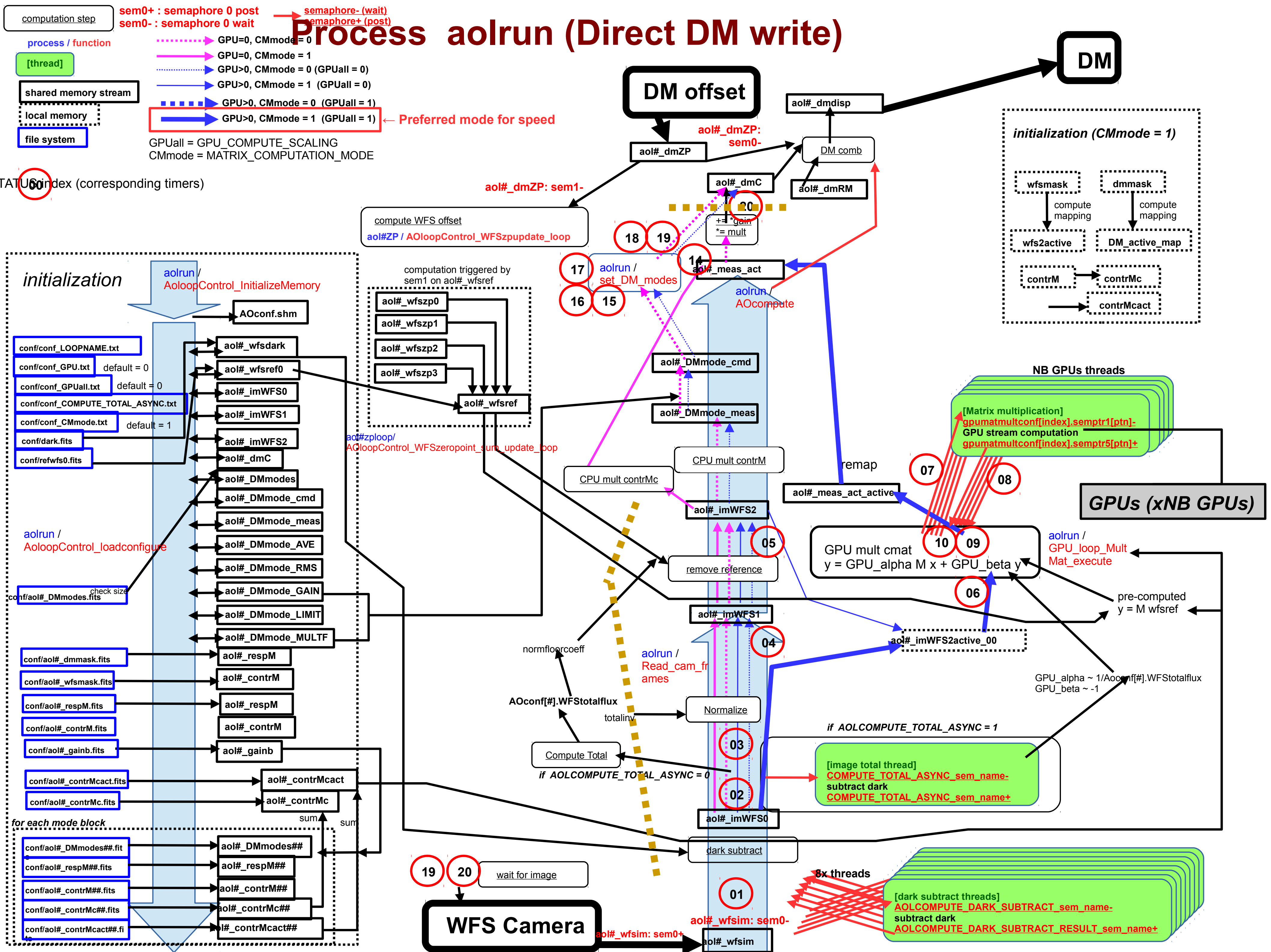
Control Matrices

Matrix	Description	Input→ output	Gain control (primary write)	Notes
contrM (CMmode=0)	***Full modal control matrix*** *Split in multi-GPU*	WFSpix → DMmodes	0.0<**loopgain**<1.0 0.0<**DMmodes_GAIN[m]**<1.0	gainMB has no effect and will not update contrM
contrMc (CMmode=1, GPUmode=0)	***Full combined control matrix*** *Split in multi-GPU*	WFSpix → DMactuators	0.0<**gainMB[k]**<1.0 0.0<**loopgain**<1.0	contrMc re-built for each change of gainMB If DM is MODAL: gainMB has no effect and will not update contrM
contrMcact (CMmode=1, GPUmode=1)	***Combined control matrix, only active pixels*** *Split in multi-GPU*	Active WFS pixels → Active DM actuators	0.0<**gainMB[k]**<1.0 0.0<**loopgain**<1.0	contrMcact re-built for each change of gainMB If DM is MODAL: gainMB has no effect and will not update contrM

CMmode MATRIX_COMPUTATION_MODE	GPUmode	GPUallmode COMPUTE_GPU_SCALING	Camera read output (Read_cam_frame)	WFS reference subtraction	Control Matrix operation(s)
0	0	0	→ imWFS1	CPU subtraction → imWFS2	**contrM** x imWFS2 → DMmode_meas [CPU] DMmode_meas → cmd_modes [CPU] DMmodes x cmd_modes →dmC [CPU]
0	>0	0	→ imWFS1	CPU subtraction → imWFS2	**contrM** x imWFS2 → DMmode_meas [GPU] DMmode_meas → cmd_modes [CPU] DMmodes x cmd_modes →dmC [GPU]
0 [to be done]	>0	1	→ imWFS0 / GPU_alpha, GPU_beta	done in GPU as part as CM mult	**contrM** x imWFS0 →DMmode_meas [GPU] DMmode_meas → cmd_modes [CPU] DMmodes x cmd_modes →dmC [GPU]
1	0	0	→ imWFS1	CPU subtraction → imWFS2	**contrMc** x imWFS2 → meas_act [CPU] meas_act → dmC [CPU]
1	>0	0	→ imWFS1	CPU subtraction → imWFS2	**contrMcact** x imWFS2_active → meas_act_active [GPU] meas_act → dmC [CPU]
1	>0	1	→ imWFS0 / GPU_alpha, GPU_beta	done in GPU as part as CM mult	**contrMcact** x imWFS0_active → meas_act_active [GPU] meas_act → dmC [CPU]

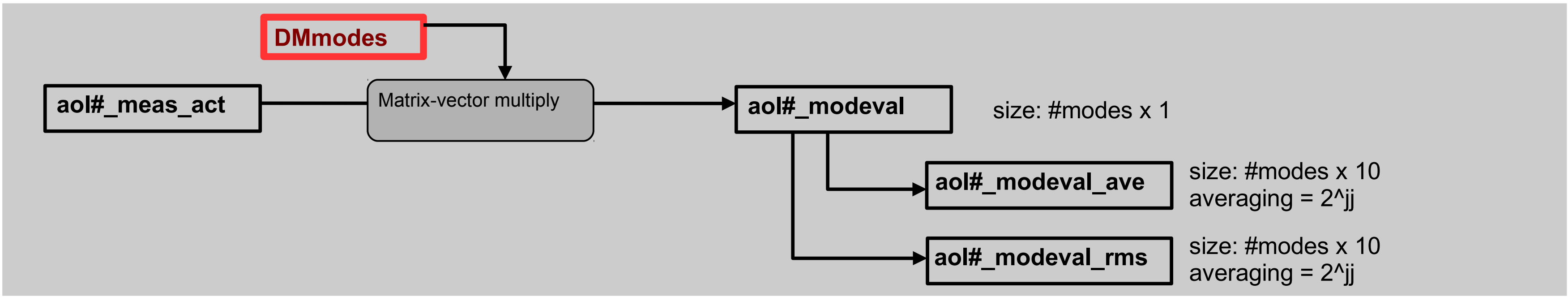


Process aolrun (Direct DM write)

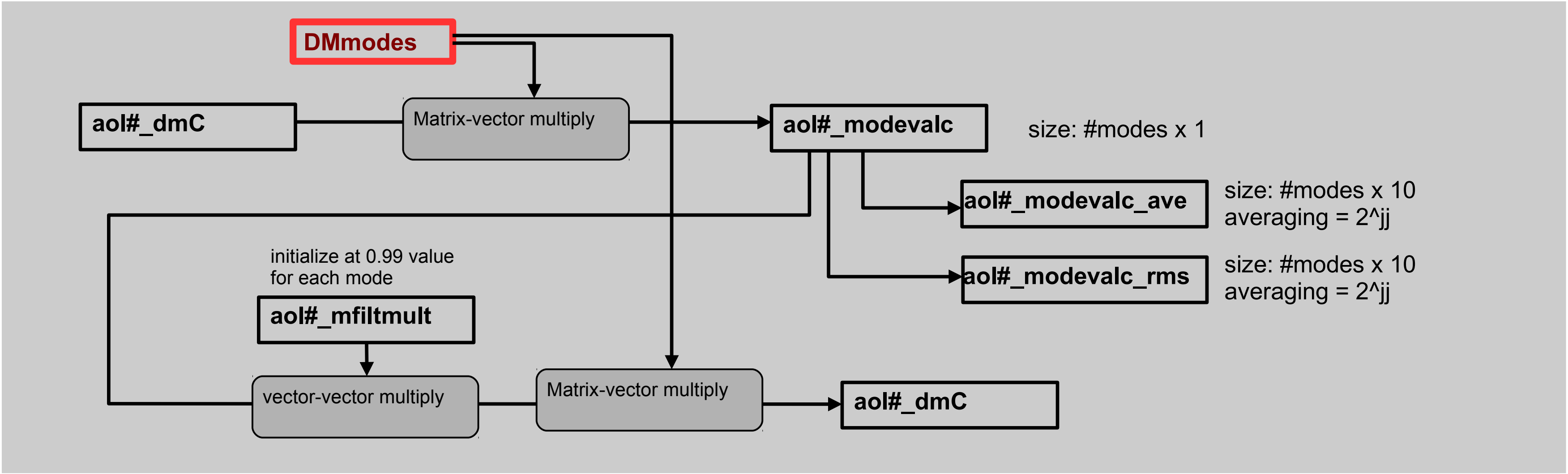


Auxillary processes

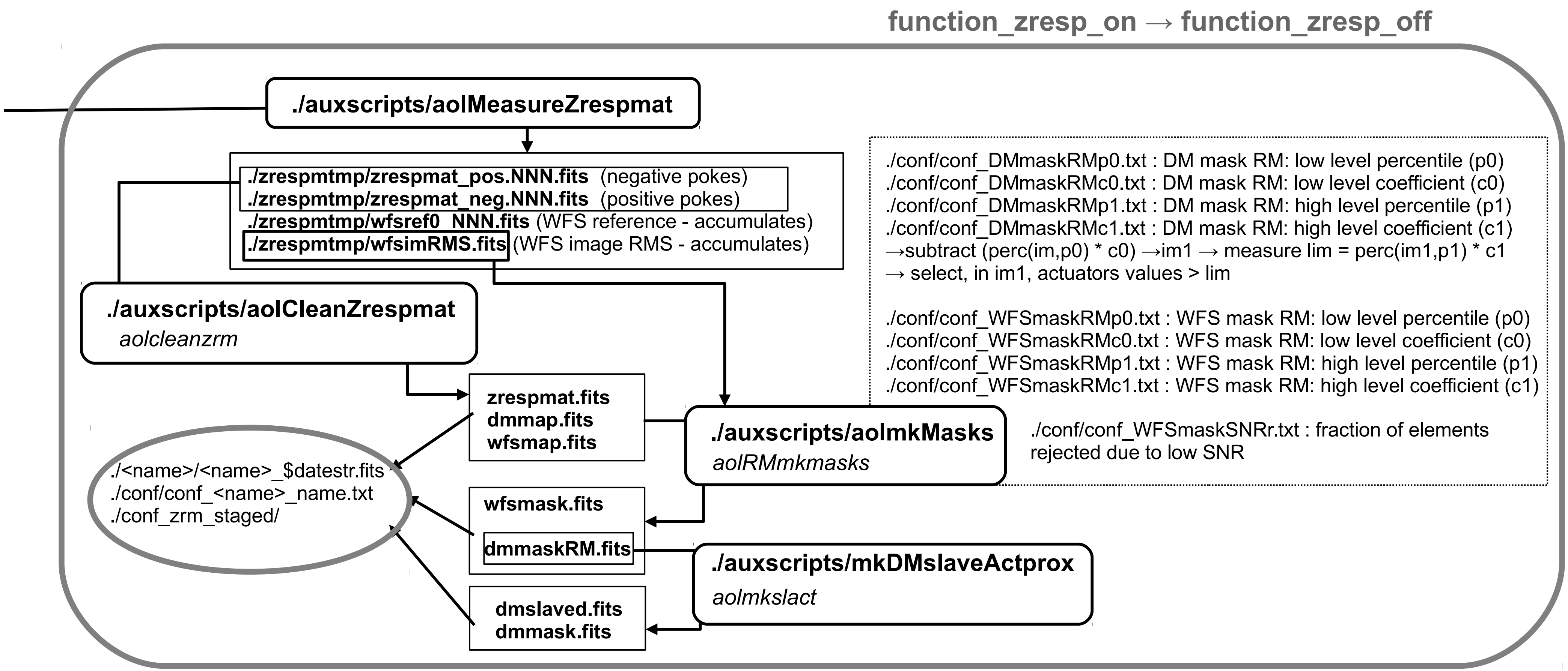
Decompose WFS measurements in modes

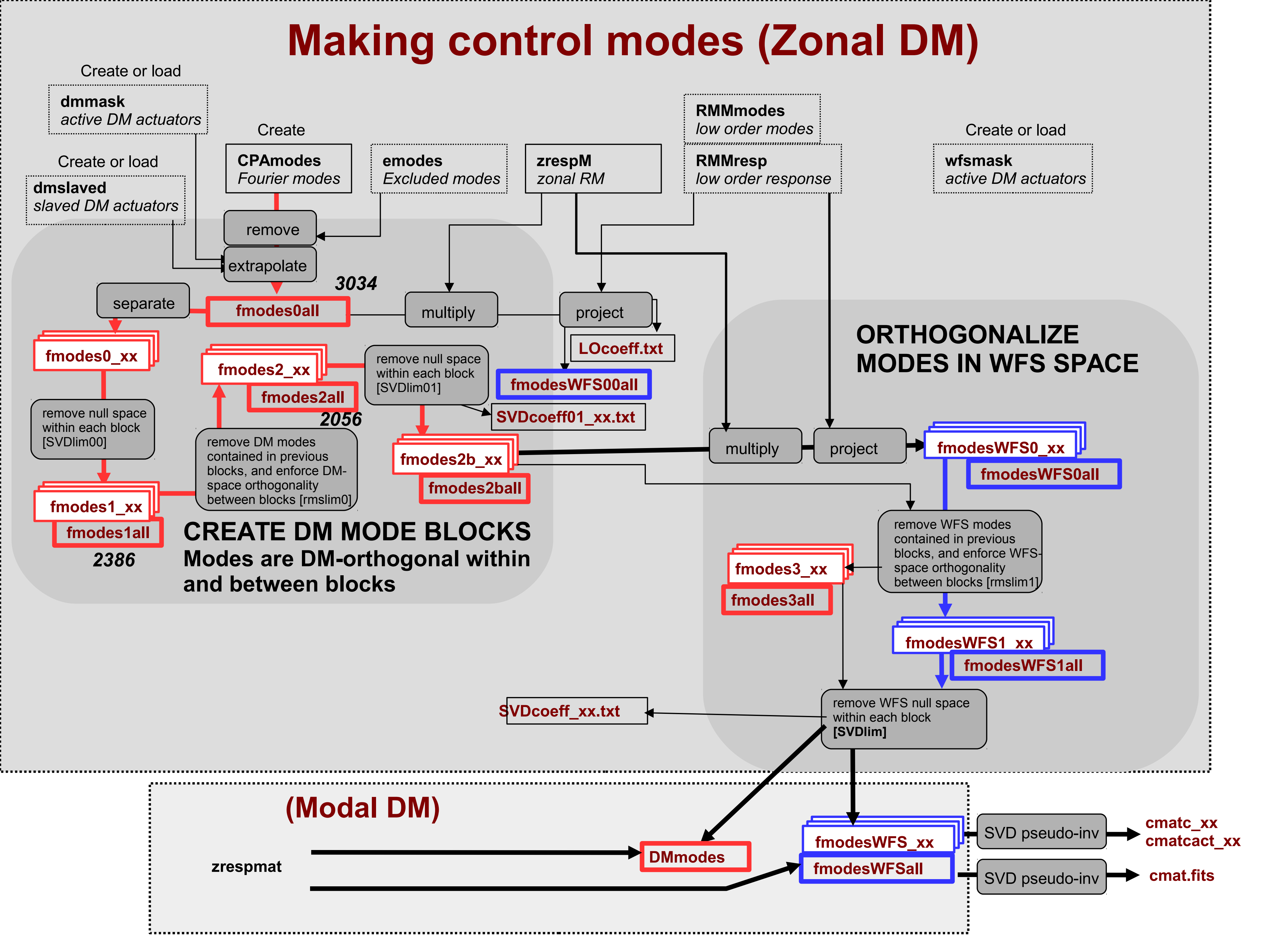


Decompose DM commands in modes + apply modal mult gains



Zonal response matrix acquisition → masks

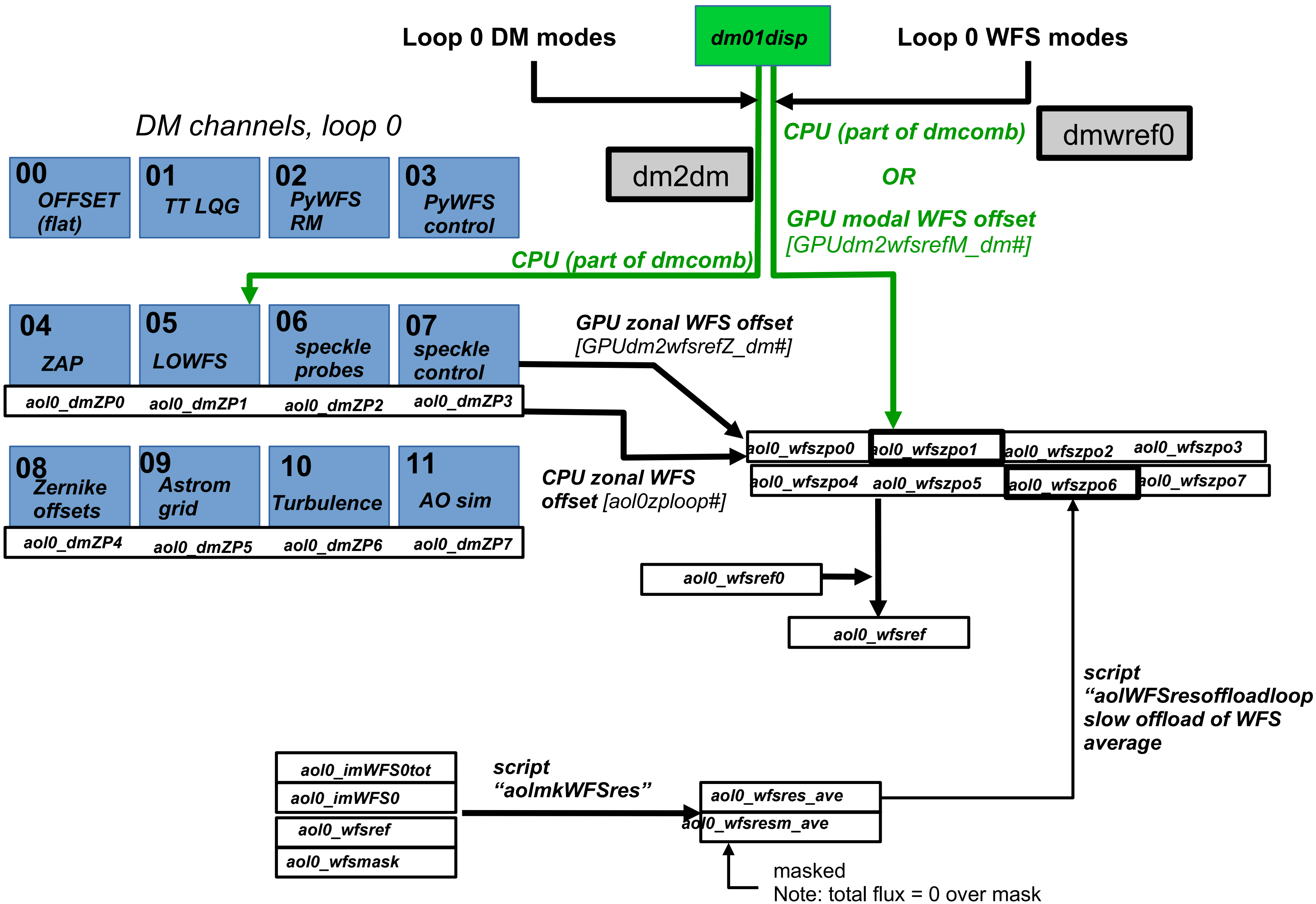




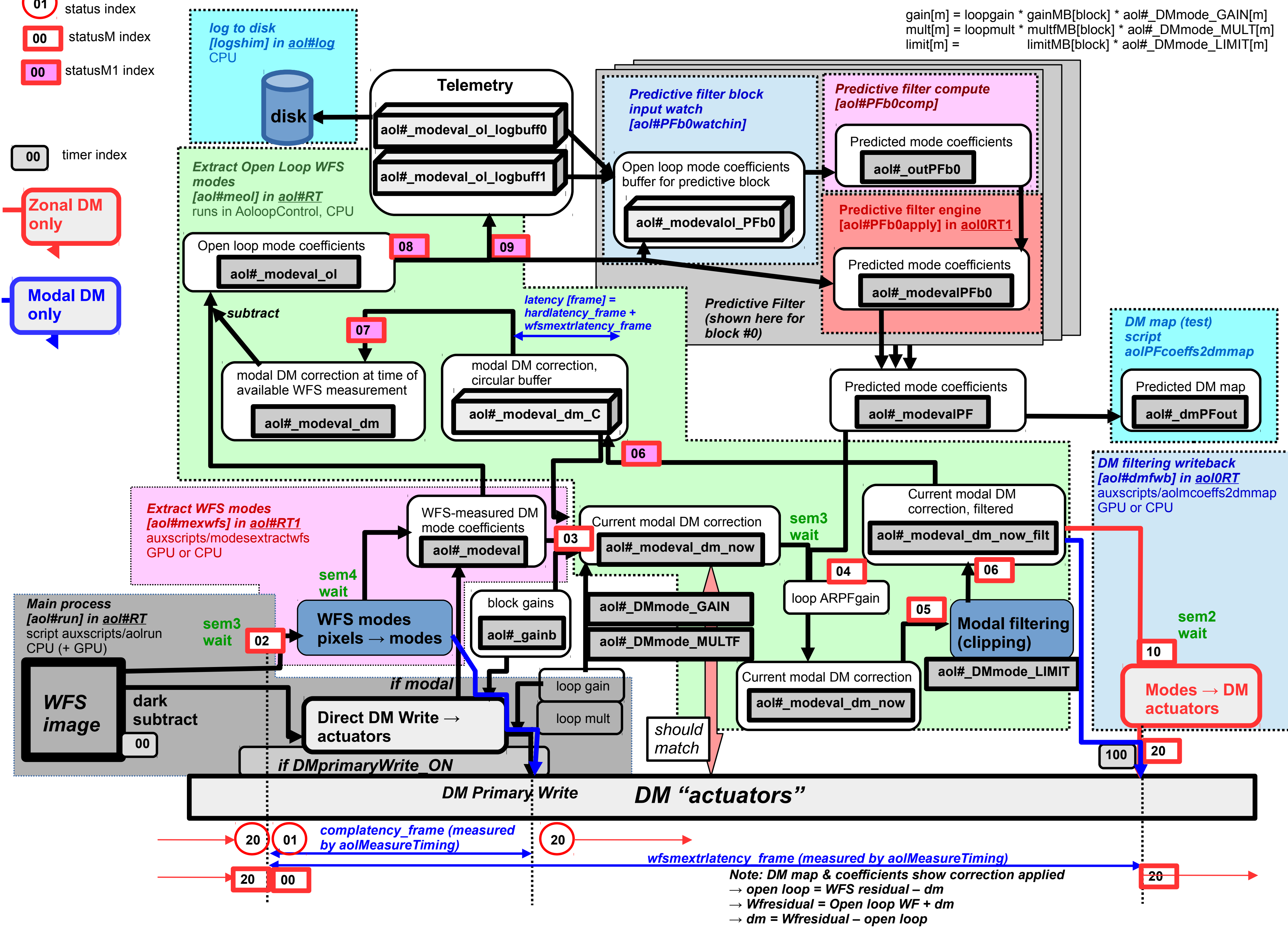
Making control modes (Zonal DM)
Create or load
dmmask
active DM actuators
Create or load
dmslaved
slaved DM actuators
Create
CPAmodes
Fourier modes
emodes
Excluded modes
zrespM
zonal RM
RMMmodes
low order modes
RMMresp
low order response
Create or load
wfsmask
active DM actuators
remove
extrapolate
3034
separate
fmodes0all
multiply
project
LOcoeff.txt
fmodesWFS00all
fmodes0_xx
fmodes2_xx
fmodes2all
remove null space
within each block
[SVDlim01]
SVDcoeff01_xx.txt
2056
remove null space
within each block
[SVDlim00]
remove DM modes
contained in previous
blocks, and enforce DM-
space orthogonality
between blocks [rmslim0]
fmodes2b_xx
fmodes2ball
fmodes1_xx
fmodes1all
2386
CREATE DM MODE BLOCKS
Modes are DM-orthogonal within
and between blocks
ORTHOGONALIZE
MODES IN WFS SPACE
multiply
project
fmodesWFS0_xx
fmodesWFS0all
remove WFS modes
contained in previous
blocks, and enforce WFS-
space orthogonality
between blocks [rmslim1]
fmodes3_xx
fmodes3all
fmodesWFS1_xx
fmodesWFS1all
SVDcoeff_xx.txt
remove WFS null space
within each block
[SVDlim]
(Modal DM)
zrespmat
DMmodes
fmodesWFS_xx
fmodesWFSall
SVD pseudo-inv
SVD pseudo-inv
cmatc_xx
cmatcact_xx
cmat.fits

OFFSETTING
LOWFS (loop #1, dm01) → PyWFS (loop #0, dm00)

Green color: process is part of loop #1



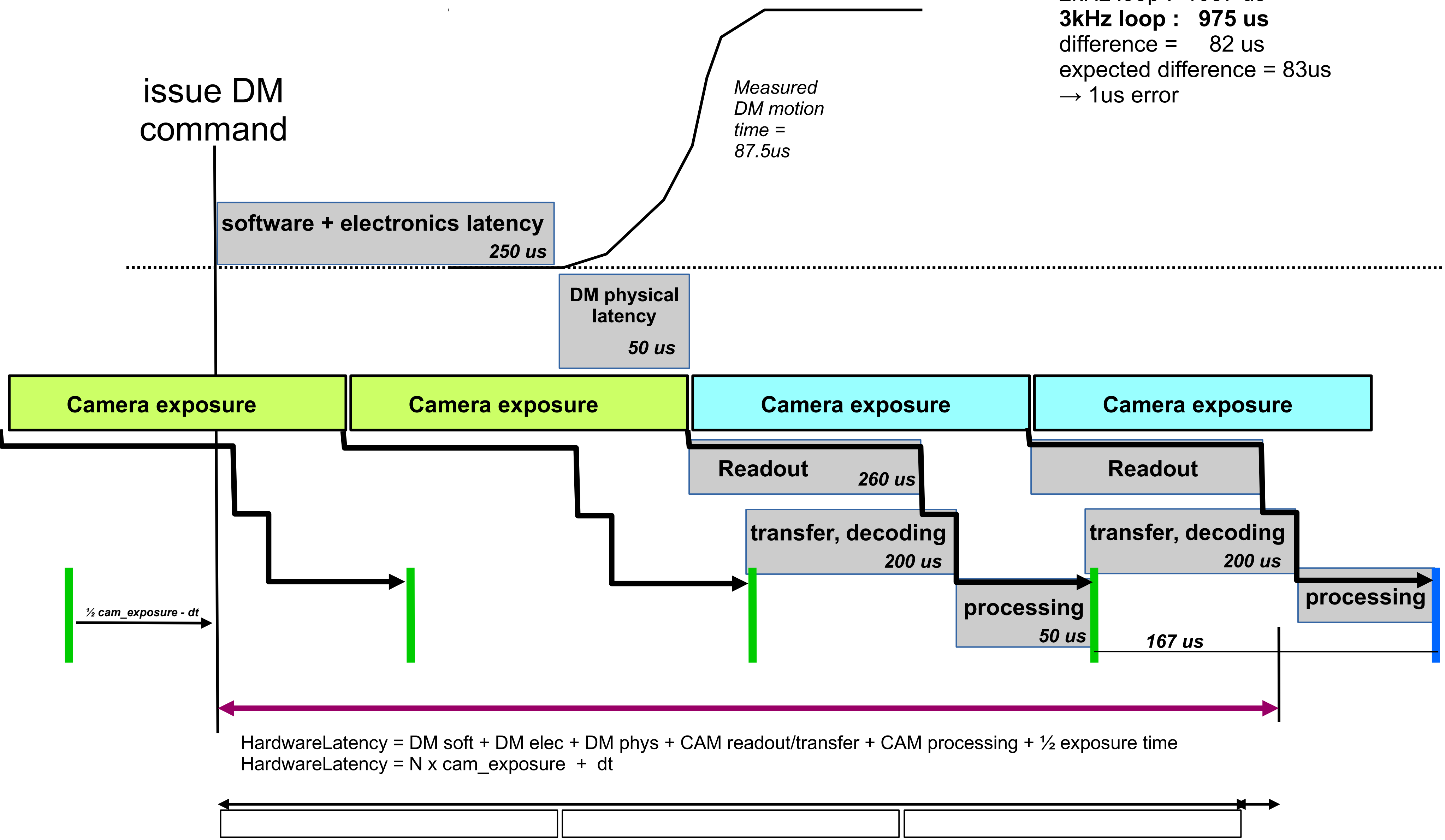
Processes, output to DM (main loop)



Hardware Latency

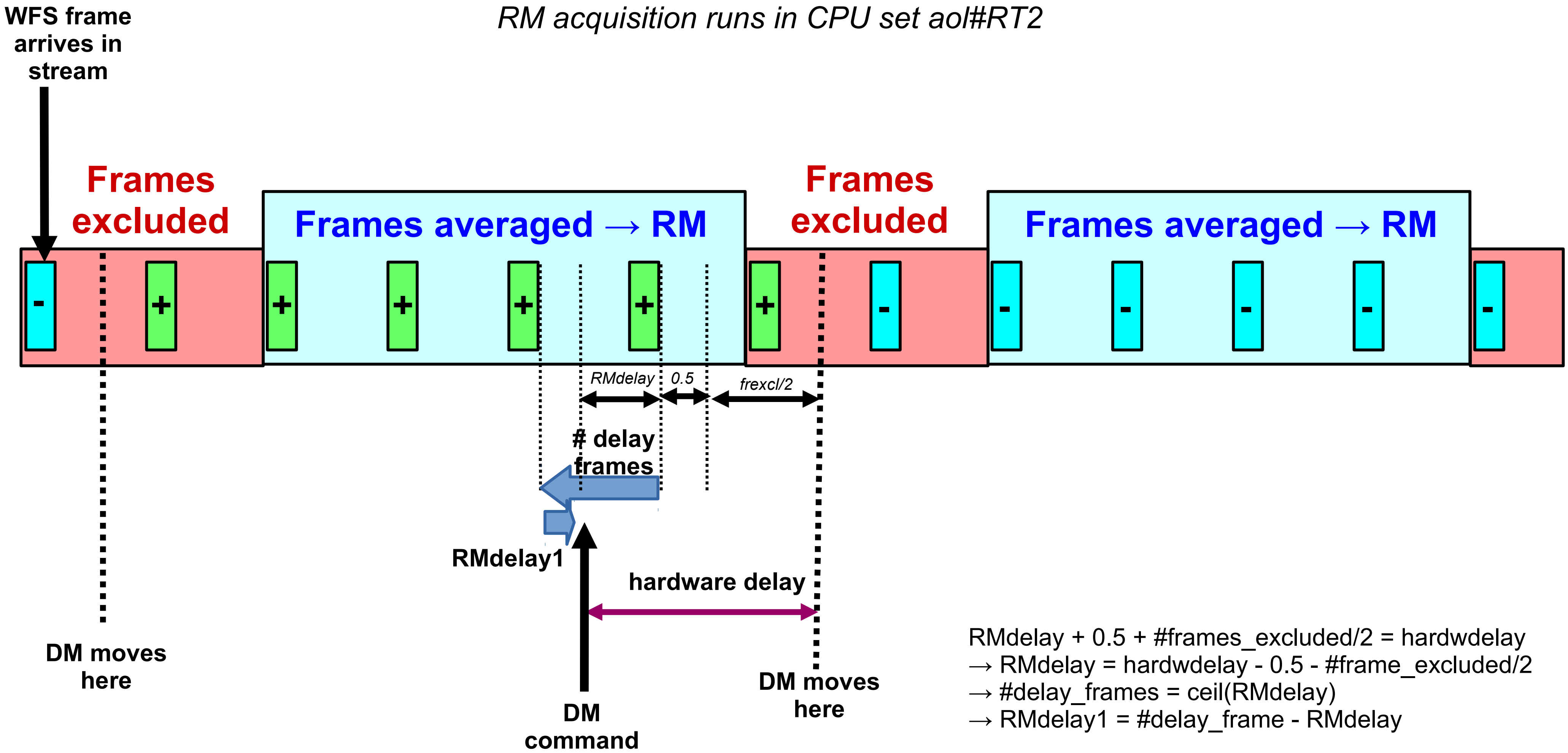
Definition:
Time offset between ***DM command issued****, and* ***mid-point between 2 consecutive WFS frames with largest difference***

SCExAO :
2kHz loop : 1057 us
3kHz loop : 975 us
difference = 82 us
expected difference = 83us
→ 1us error



RM acquisition - Timing

RM acquisition runs in CPU set aol#RT2



Loop:

Wait on and read WFS frame → allocate WFS frame to appropriate frame block

If poke required: wait RMdelay1, then poke