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

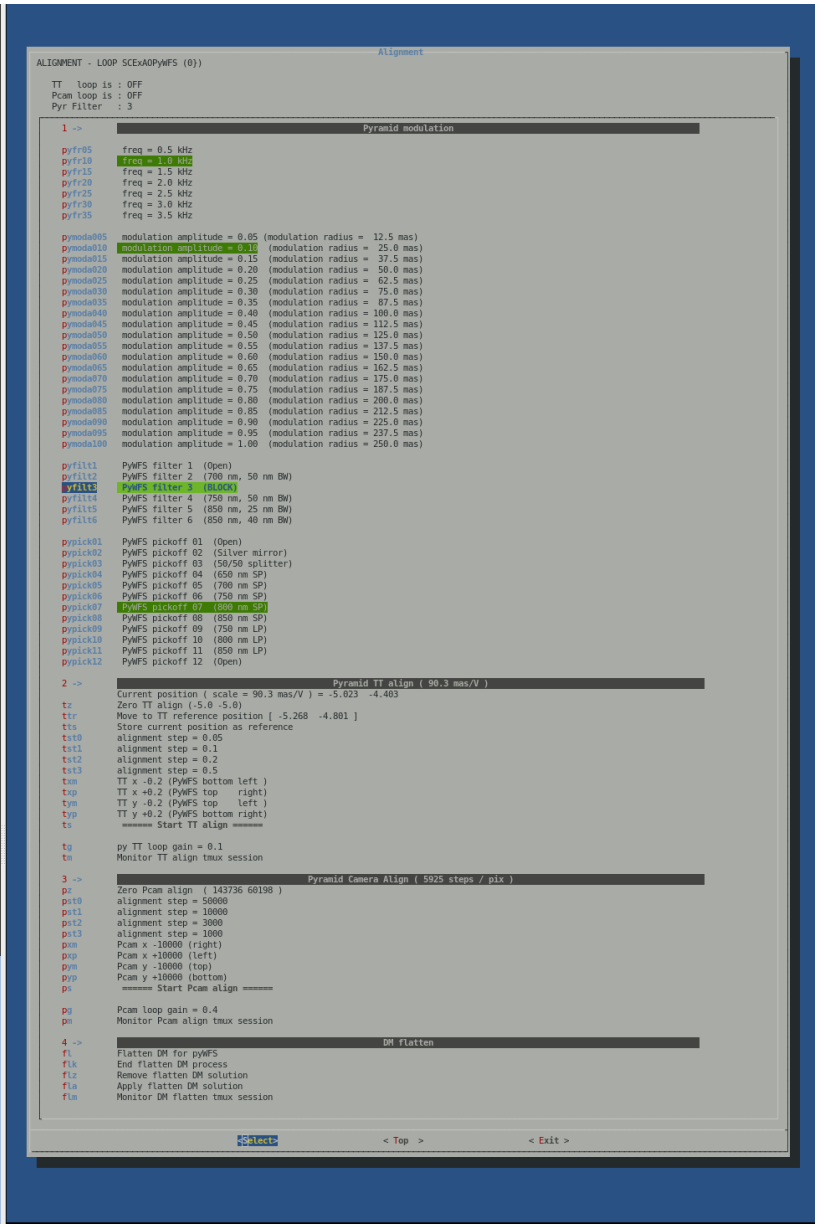
Largely limited by hardware. Fully system timing stable at 10us level, and RTS latency due to IPC, TCP transfers between computers, and GPU transfers is <100us total → can drive ~10 kHz loop on multi-computer system, and ~20 kHz loop on single computer. SCExAO implementation drives 2000-actuator, 14,400-sensor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

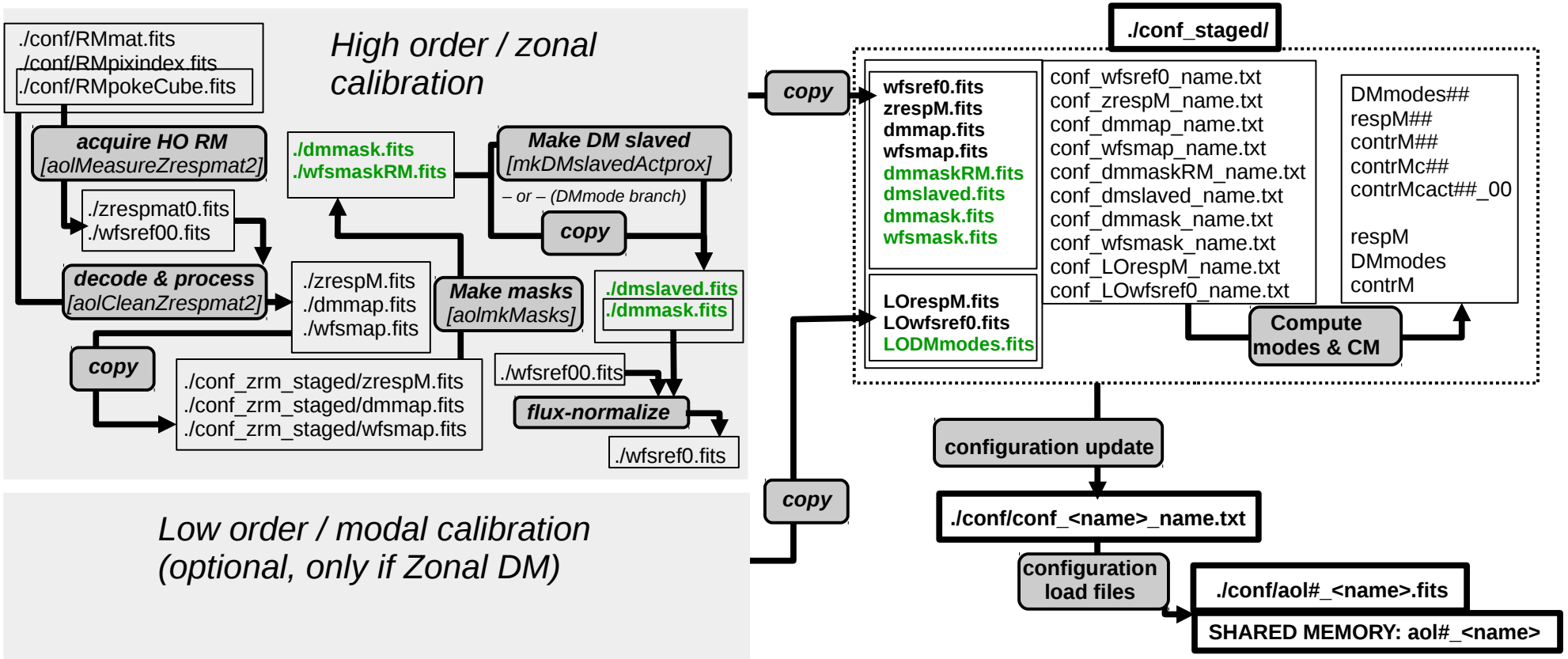
→ No spatial filtering options

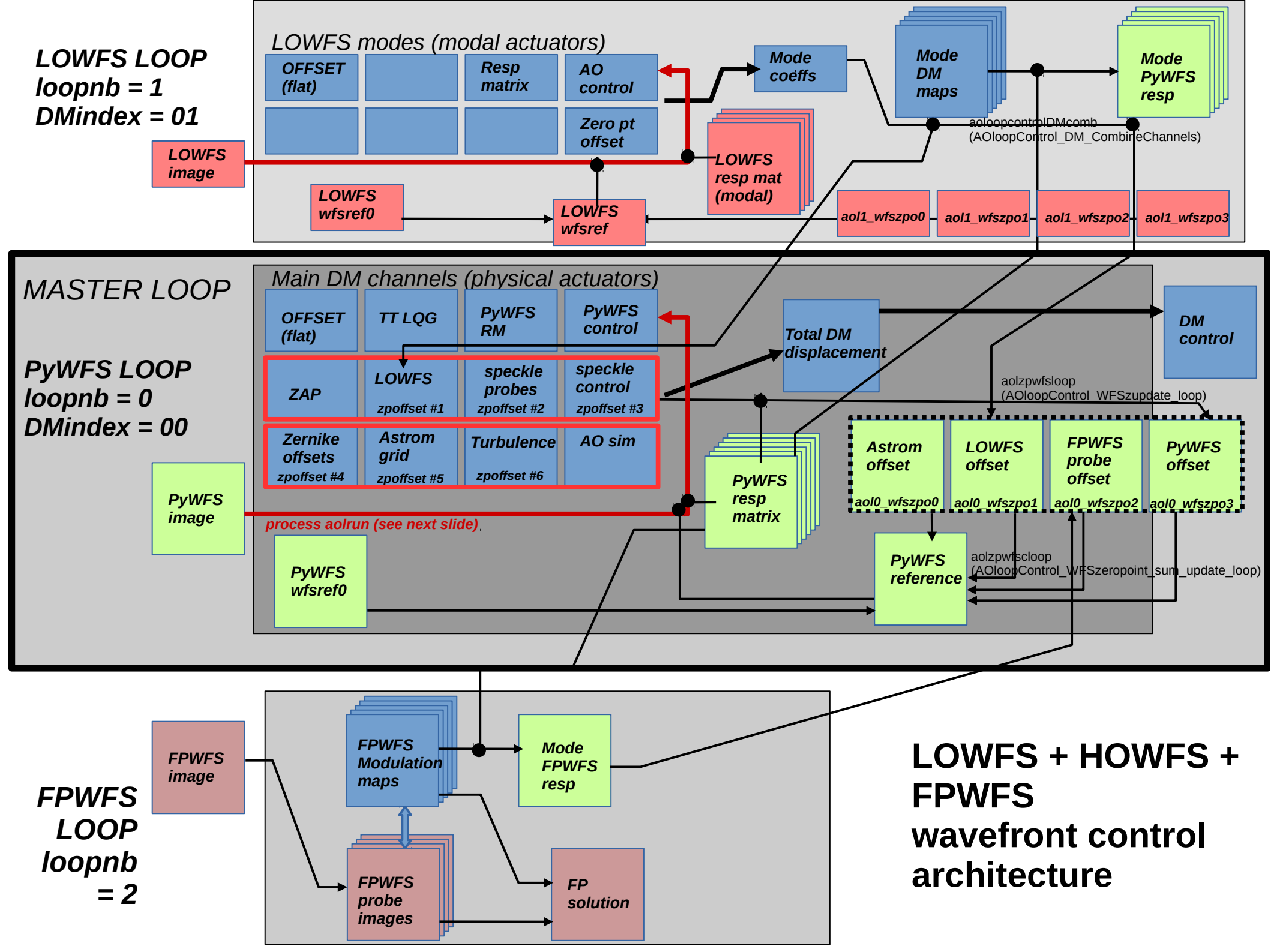
→ “Direct write” CM and “Modal” CM are the same (1 mode = 1 actuator)

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

Control Matrix Computation Modes

WFSnorm *(./conf/param_WFSnorm.txt)* ***WFS normalization mode*** C code: WFSnormalize

0: Do not normalize WFS images
1: Normalize WFS images (default)

WFSnorm should be left unchanged between RM acquisition and Loop control

DMprimaryWrite (./conf/param_DMprimaryWriteON.txt) ***DM primary write*** C code: DMprimaryWriteON

0: DM primary write is off (default)
1: DM primary write is on

DMfilteredWrite (./conf/param_DMfilteredWriteON.txt) ***DM filtered write*** C code: DMfilteredWriteON

0: DM filtered write is off (default)
1: DM filtered write is on

CMMODE *(./conf/param_CMMODE.txt)* ***Combined Control matrix mode*** C code: CMMODE

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/param_DMMODE.txt)* ***DM mode (zonal vs. modal)*** C code: DMMODE

0 (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
1 (MODAL): DM pixels correspond to abstract modes
→ no spatial filtering, setting 1 block only

Note: **DMMODE**=0 → **CMMODE**=0 (CPA-splitting of modes into blocks)

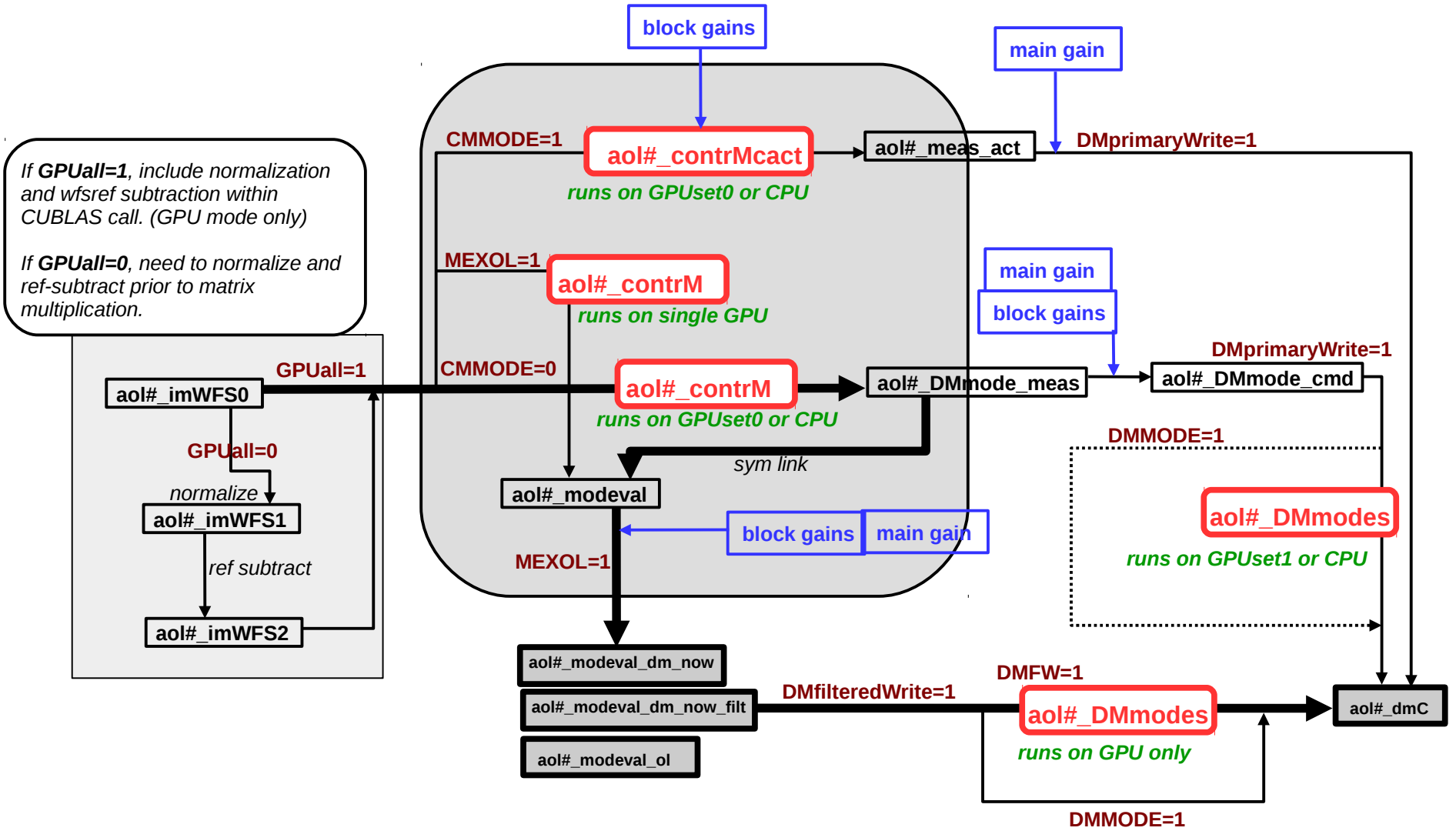
GPU *(./conf/param_GPU.txt)* ***# of GPUs to use for CM multiplication*** C code: AOconf[loop].GPU

0: use CPU
1+: number of GPUs

if CMmode=1 and GPUmode>0:
GPUall *(./conf/param_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

Computing Modes and Control Matrices



Supported Computing Modes

GPUset0=0 → GPUall=OFF

CMMODE	*GPUset0*	*GPUall*	*DMprimaryWrite*	*GPUset1*	*DMFWB*	*Comments*
0 (modal control)	0 (CPU mode)	OFF	0	N/A	0	Note: DOES NOTE WRITE TO DM
0 (modal control)	0 (CPU mode)	OFF	0	N/A	1	OK
0 (modal control)	0 (CPU mode)	OFF	1	0 (CPU mode)	0	OK
0 (modal control)	0 (CPU mode)	OFF	1	0 (CPU mode)	1	Note: 2 DM writes per iteration
0 (modal control)	0 (CPU mode)	OFF	1	>0 (GPU mode)	0	OK
0 (modal control)	0 (CPU mode)	OFF	1	>0 (GPU mode)	1	Note: 2 DM writes per iteration
0 (modal control)	>1 (GPU mode)	OFF	0	N/A	0	Note: DOES NOTE WRITE TO DM
0 (modal control)	>1 (GPU mode)	OFF	0	N/A	1	OK
0 (modal control)	>1 (GPU mode)	OFF	1	0 (CPU mode)	0	OK
0 (modal control)	>1 (GPU mode)	OFF	1	0 (CPU mode)	1	Note: 2 DM writes per iteration
0 (modal control)	>1 (GPU mode)	OFF	1	>0 (GPU mode)	0	OK
0 (modal control)	>1 (GPU mode)	OFF	1	>0 (GPU mode)	1	Note: 2 DM writes per iteration
0 (modal control)	>1 (GPU mode)	ON	0	N/A	0	Note: DOES NOTE WRITE TO DM
0 (modal control)	**>1 (GPU mode)**	**ON**	**0**	**N/A**	**1**	**PREFERRED MODE FOR GPU-BASED MODAL ExAO PRIMARY LOOP**
0 (modal control)	>1 (GPU mode)	ON	1	0 (CPU mode)	0	OK
0 (modal control)	>1 (GPU mode)	ON	1	0 (CPU mode)	1	Note: 2 DM writes per iteration
0 (modal control)	>1 (GPU mode)	ON	1	>0 (GPU mode)	0	OK
0 (modal control)	>1 (GPU mode)	ON	1	>0 (GPU mode)	1	Note: 2 DM writes per iteration
1 (zonal control)	0 (CPU mode)	OFF	0	N/A	0	Note: DOES NOTE WRITE TO DM (→ aol0_meas_act)
1 (zonal control)	0 (CPU mode)	OFF	1	0 (CPU mode)	0	OK
1 (zonal control)	0 (CPU mode)	OFF	1	>0 (GPU mode)	0	OK
1 (zonal control)	>1 (GPU mode)	OFF	0	N/A	0	Note: DOES NOTE WRITE TO DM
1 (zonal control)	>1 (GPU mode)	OFF	1	N/A	0	OK
1 (zonal control)	>1 (GPU mode)	ON	0	N/A	0	Note: DOES NOTE WRITE TO DM
1 (zonal control)	>1 (GPU mode)	ON	1	N/A	0	OK

Green: GPU computation
Gray : CPU computation

Processes, output to DM (main loop)

[process name] (same name as tmux session)
aol0RT : CPU set

01 status index
00 statusM index
00 statusM1 index
00 timer index

gain[m] = loopgain * gainb[block] * aol#_DMmode_GAIN[m]
mult[m] = loopmult * multb[block] * aol#_DMmode_MULT[m]
limit[m] = loopmaxlimit * limitb[block] * aol#_DMmode_LIMIT[m]

Zonal DM only

Modal DM only

log to disk [logshim] in aol#log
CPU
disk

Telemetry
aol#_modeval_ol_logbuff0
aol#_modeval_ol_logbuff1

Extract Open Loop WFS modes [aol#meol] in aol#RT
runs in AoloopControl, CPU

Open loop mode coefficients
aol#_modeval_ol
08
09

Predictive filter block input watch [aol#PFb0watchin]
Open loop mode coefficients buffer for predictive block
aol#_modevalol_PFb0

Predictive filter compute [aol#PFb0comp]
Prediction filter
aol#_outPFb0

Predictive filter engine [aol#PFb0apply] in aol0RT1
Predicted mode coefficients
aol#_modevalPFb0

Predictive Filter (shown here for block #0)

subtract
07
latency [frame] = hardlatency_frame + wfsmextrlatency_frame

modal DM correction at time of available WFS measurement
aol#_modeval_dm

modal DM correction, circular buffer
aol#_modeval_dm_C

Predicted mode coefficients
aol#_modevalPF

DM map (test) script aolPFcoeffs2dmmap
Predicted DM map
aol#_dmPFout

06

Extract WFS modes [aol#mexwfs] in aol#RT1
auxscripts/modesextractwfs
GPU or CPU

WFS-measured DM mode coefficients
aol#_modeval
03

Current modal DM correction
aol#_modeval_dm_corr

sem3 wait

Current modal DM correction, filtered
aol#_modeval_dm_now_filt

GPU-based DM filtered write [aol#dmfw] in aol0RT
auxscripts/aolmcoeffs2dmmap
GPU or CPU

sem4 wait

04
loop ARPFgain

06

05
Modal filtering (clipping)
aol#_DMmode_LIMIT

Main process [aol#run] in aol#RT
script auxscripts/aolrun
CPU (+ GPU)

sem3 wait
02
WFS modes pixels → modes

block gains
aol#_gainb

aol#_DMmode_GAIN
aol#_DMmode_MULTF

Current modal DM correction
aol#_modeval_dm_now

sem2 wait
10
Modes → DM actuators

WFS image
dark subtract
00

if CMMODE=0
Direct DM Write → actuators
loop gain
loop mult

should match

100
20

if DMprimaryWrite_ON

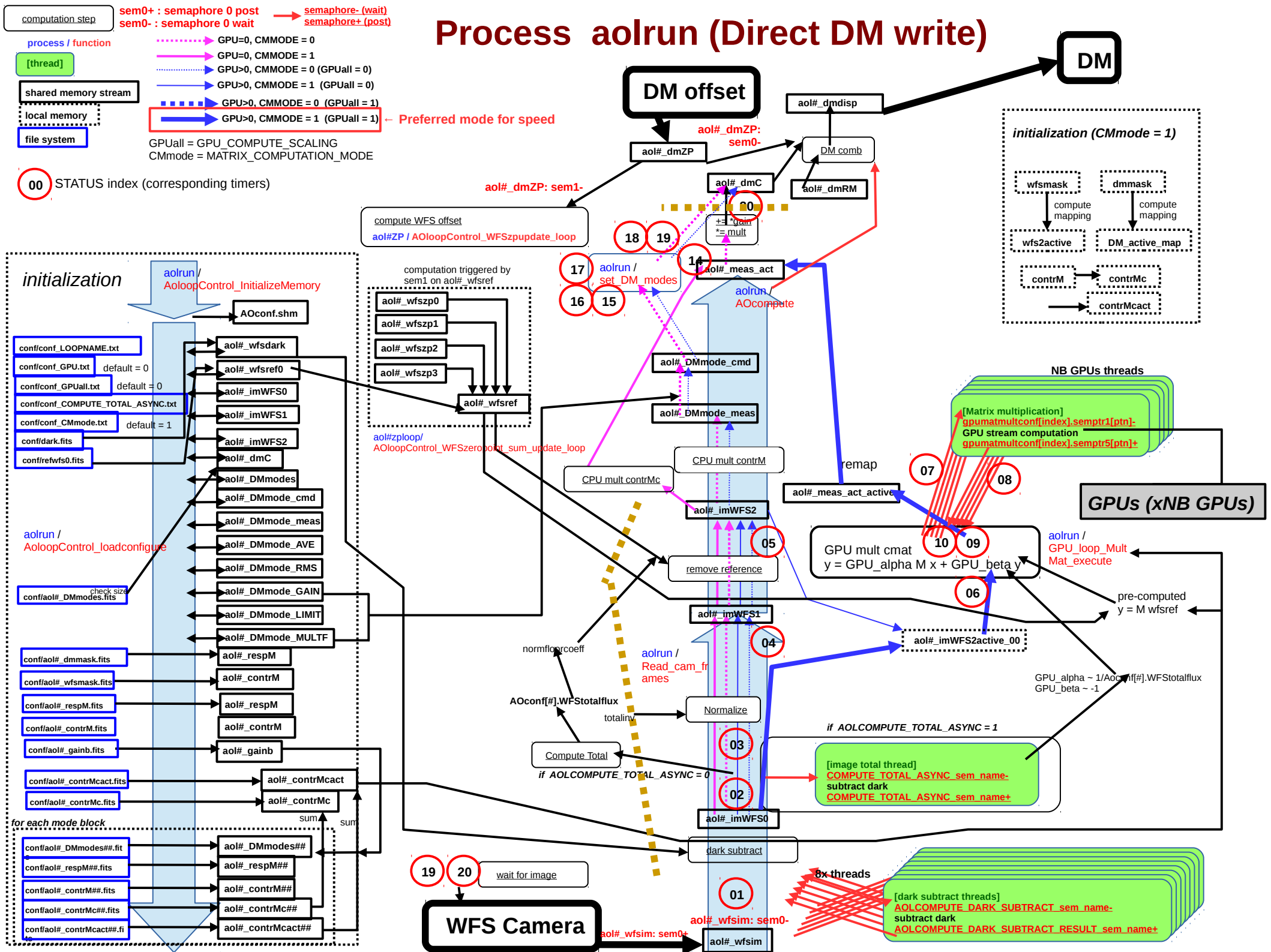
DM primary Write ***DM "actuators"*** ***DM filtered Write***

20 01 ***complatency_frame (measured by aolMeasureTiming)*** 20

wfsmextrlatency_frame (measured by aolMeasureTiming)

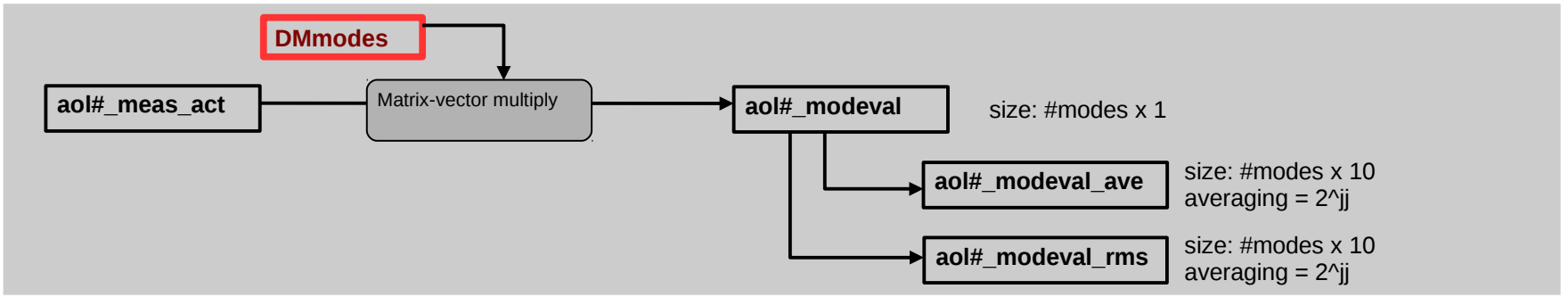
20 00 20

Note: DM map & coefficients show correction applied
→ open loop = WFS residual – dm
→ Wfresidual = Open loop WF + dm
→ dm = Wfresidual – open loop

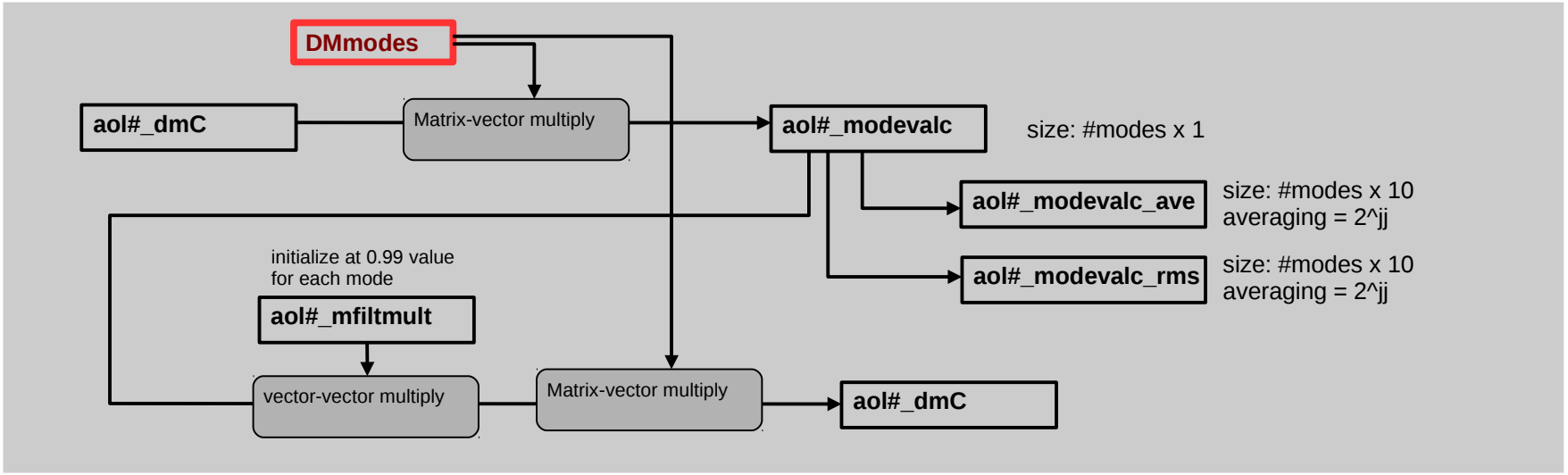

Process aolrun (Direct DM write)
computation step
sem0+ : semaphore 0 post
sem0- : semaphore 0 wait
semaphore- (wait)
semaphore+ (post)
process / function
[thread]
shared memory stream
local memory
file system
GPU=0, CMMODE = 0
GPU=0, CMMODE = 1
GPU>0, CMMODE = 0 (GPUall = 0)
GPU>0, CMMODE = 1 (GPUall = 0)
GPU>0, CMMODE = 0 (GPUall = 1)
GPU>0, CMMODE = 1 (GPUall = 1)
← Preferred mode for speed
GPUall = GPU_COMPUTE_SCALING
CMmode = MATRIX_COMPUTATION_MODE
00 STATUS index (corresponding timers)
DM
DM offset
aol#_dmdisp
aol#_dmZP: sem0-
aol#_dmZP
DM comb
aol#_dmZP: sem1-
aol#_dmC
aol#_dmRM
compute WFS offset
aol#ZP / AOloopControl_WFSzpupdate_loop
+= *gain
*= mult
18
19
17
16
15
14
aolrun / set_DM_modes
aol#_meas_act
aolrun / AOcompute
initialization (CMmode = 1)
wfsmask
dmmask
compute mapping
compute mapping
wfs2active
DM_active_map
contrM
contrMc
contrMcact
initialization
aolrun / AoloopControl_InitializeMemory
AOconf.shm
conf/conf_LOOPNAME.txt
conf/conf_GPU.txt
default = 0
conf/conf_GPUall.txt
default = 0
conf/conf_COMPUTE_TOTAL_ASYNC.txt
conf/conf_CMmode.txt
default = 1
conf/dark.fits
conf/refwfs0.fits
aol#_wfsdark
aol#_wfsref0
aol#_imWFS0
aol#_imWFS1
aol#_imWFS2
aol#_dmC
aol#_DMmodes
aol#_DMmode_cmd
aol#_DMmode_meas
aol#_DMmode_AVE
aol#_DMmode_RMS
aol#_DMmode_GAIN
aol#_DMmode_LIMIT
aol#_DMmode_MULTF
aolrun / AoloopControl_loadconfigure
conf/aol#_DMmodes.fits
check size
conf/aol#_dmmask.fits
conf/aol#_wfsmask.fits
conf/aol#_respM.fits
conf/aol#_contrM.fits
conf/aol#_gainb.fits
conf/aol#_contrMcact.fits
conf/aol#_contrMc.fits
aol#_respM
aol#_contrM
aol#_respM
aol#_contrM
aol#_gainb
aol#_contrMcact
aol#_contrMc
sum
sum
for each mode block
conf/aol#_DMmodes##.fits
conf/aol#_respM##.fits
conf/aol#_contrM##.fits
conf/aol#_contrMc##.fits
conf/aol#_contrMcact##.fits
aol#_DMmodes##
aol#_respM##
aol#_contrM##
aol#_contrMc##
aol#_contrMcact##
computation triggered by sem1 on aol#_wfsref
aol#_wfszp0
aol#_wfszp1
aol#_wfszp2
aol#_wfszp3
aol#_wfsref
aol#zploop/ AOloopControl_WFSzeropoint_sum_update_loop
aol#_DMmode_cmd
aol#_DMmode_meas
CPU mult contrM
CPU mult contrMc
aol#_imWFS2
remap
aol#_meas_act_active
NB GPUs threads
[Matrix multiplication]
gpumatmultconf[index].semptr1[ptn]-
GPU stream computation
gpumatmultconf[index].semptr5[ptn]+
GPUs (xNB GPUs)
07
08
GPU mult cmat
y = GPU_alpha M x + GPU_beta y
10
09
06
05
remove reference
aolrun / GPU_loop_Mult Mat_execute
pre-computed y = M wfsref
aol#_imWFS1
aol#_imWFS2active_00
04
normflorcoeff
aolrun / Read_cam_fr ames
AOconf[#].WFStotalflux
totalinv
Normalize
GPU_alpha ~ 1/AOconf[#].WFStotalflux
GPU_beta ~ -1
03
if AOLCOMPUTE_TOTAL_ASYNC = 1
Compute Total
if AOLCOMPUTE_TOTAL_ASYNC = 0
[image total thread]
COMPUTE_TOTAL_ASYNC_sem_name-
subtract dark
COMPUTE_TOTAL_ASYNC_sem_name+
02
aol#_imWFS0
dark subtract
8x threads
[dark subtract threads]
AOLCOMPUTE_DARK_SUBTRACT_sem_name-
subtract dark
AOLCOMPUTE_DARK_SUBTRACT_RESULT_sem_name+
19
20
wait for image
01
WFS Camera
aol#_wfsim: sem0-
aol#_wfsim: sem0+
aol#_wfsim

Auxillary processes

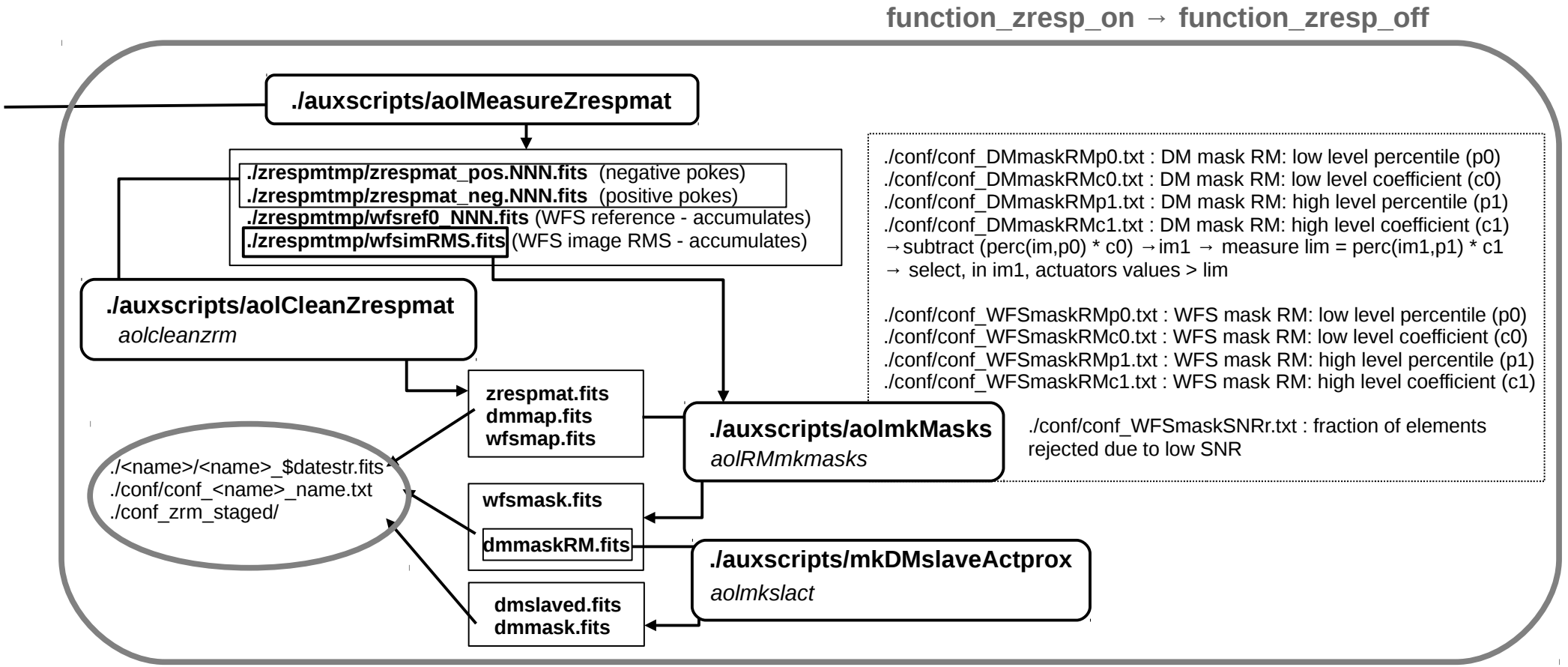
Decompose WFS measurements in modes



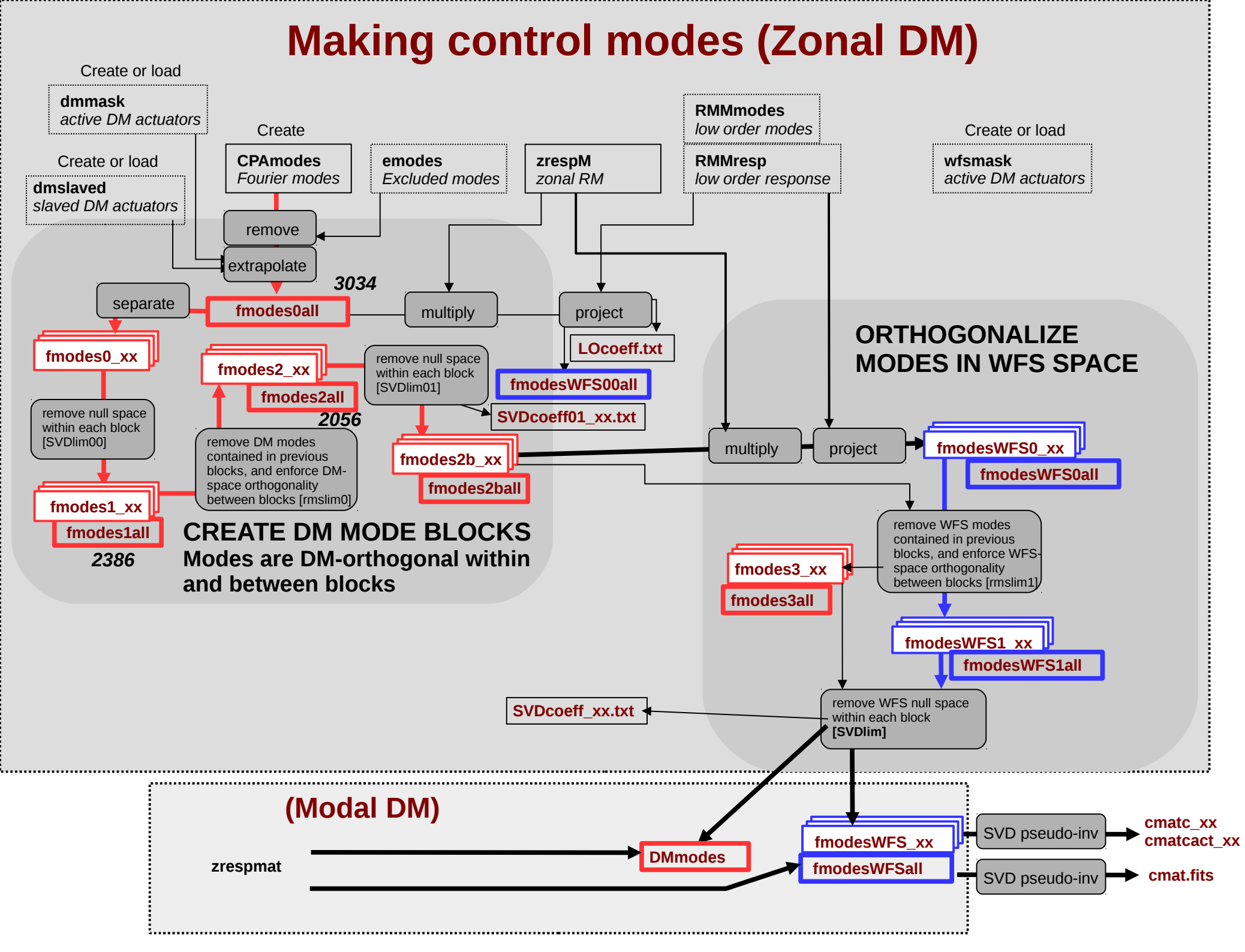
Decompose DM commands in modes + apply modal mult gains



Zonal response matrix acquisition → masks

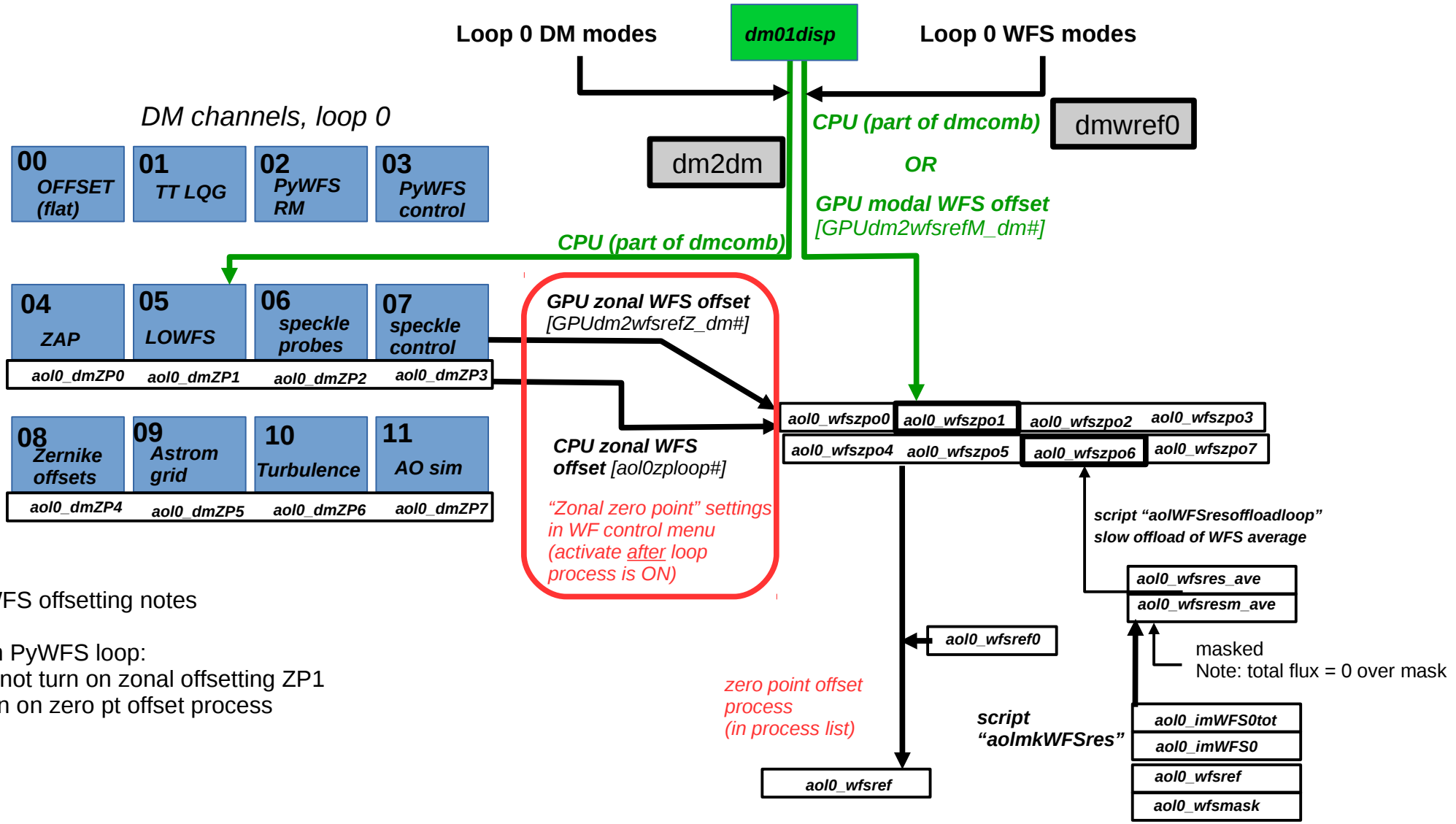


Making control modes (Zonal DM)



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

Green color: process is part of loop #1



LOWFS offsetting notes

From PyWFS loop:
- Do not turn on zonal offsetting ZP1
- Turn on zero pt offset process

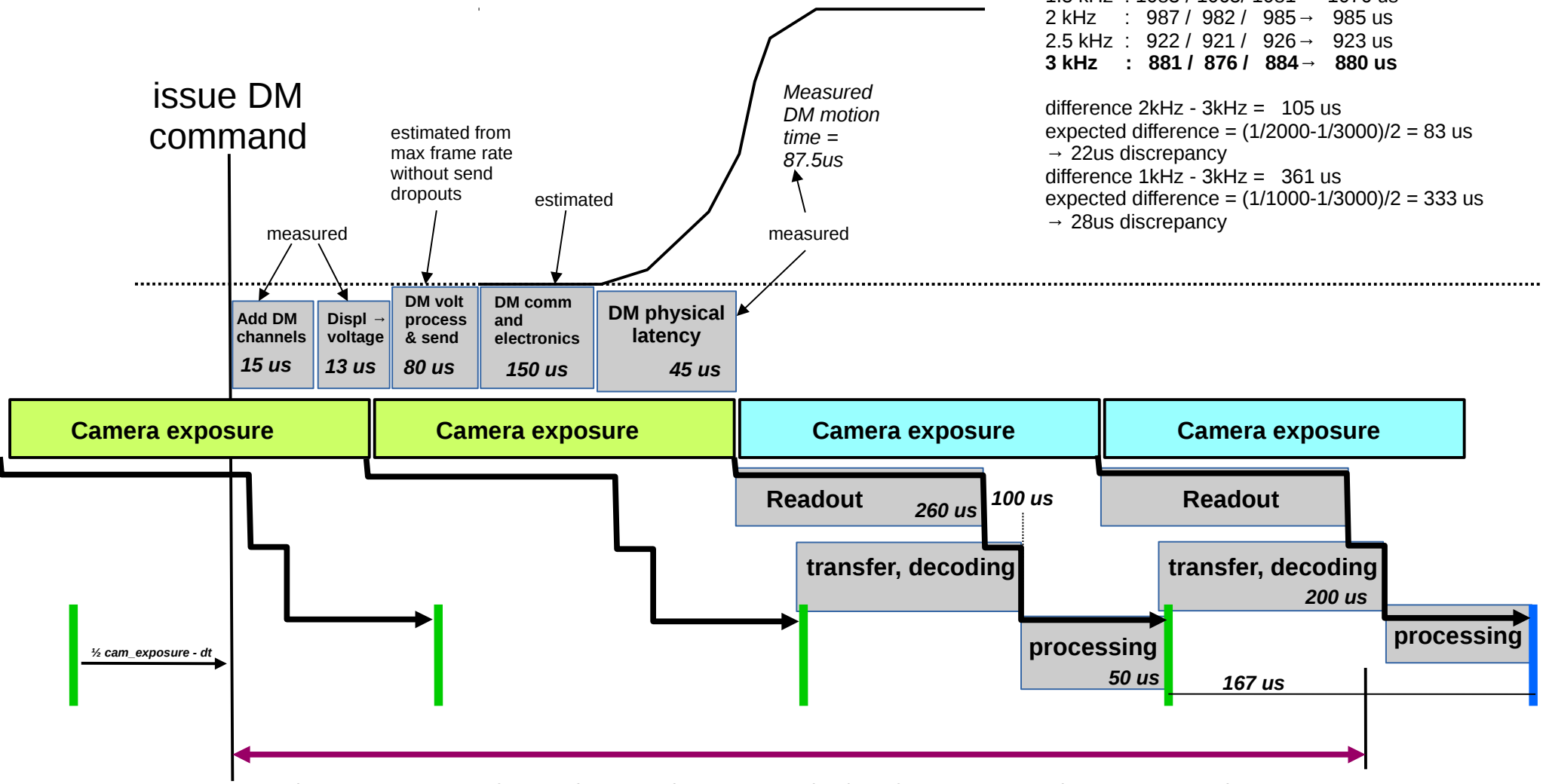
Hardware Latency

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

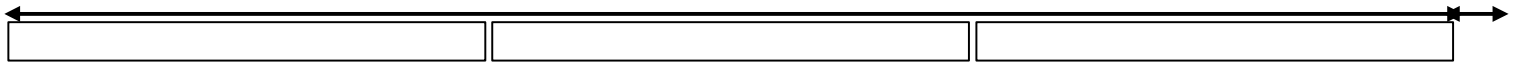
SCExAO measured hardware latencies:

1 kHz : 1253 / 1260/ 1269 → 1261 us
1.5 kHz : 1083 / 1065/ 1081 → 1076 us
2 kHz : 987 / 982 / 985 → 985 us
2.5 kHz : 922 / 921 / 926 → 923 us
3 kHz : 881 / 876 / 884 → 880 us

difference 2kHz - 3kHz = 105 us
expected difference = (1/2000-1/3000)/2 = 83 us
→ 22us discrepancy
difference 1kHz - 3kHz = 361 us
expected difference = (1/1000-1/3000)/2 = 333 us
→ 28us discrepancy

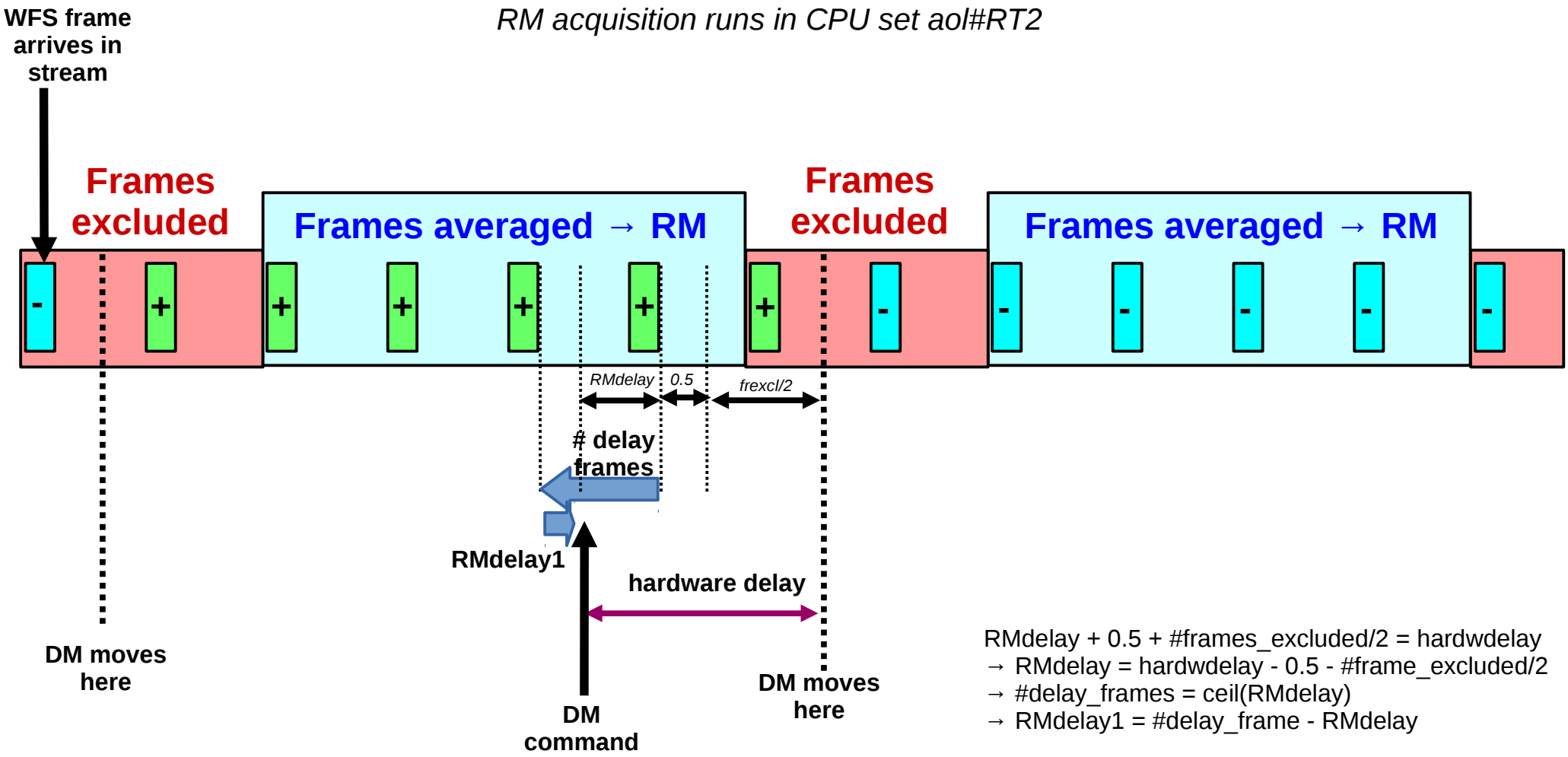


HardwareLatency = DM soft + DM elec + DM phys + CAM readout/transfer + CAM processing + ½ exposure time
HardwareLatency = N x cam_exposure + dt



RM acquisition - Timing

RM acquisition runs in CPU set aol#RT2



RMdelay + 0.5 + #frames_excluded/2 = hardwdelay
→ RMdelay = hardwdelay - 0.5 - #frame_excluded/2
→ #delay_frames = ceil(RMdelay)
→ RMdelay1 = #delay_frame - RMdelay

Loop:

Wait on and read WFS frame → allocate WFS frame to appropriate frame block
If poke required: wait RMdelay1, then poke

Predictive Control Implementation Notes

Predictive control implementation…

… **must be compatible with multi-WFS architecture**. See slide #6 for SCExAO multi-loop architecture, showing how master loop (Pyramid WFS) is offset by secondary loops (LOWFS and speckle control)

… **must be able to smoothly transition from/to conventional zonal DM write**. Slide #9 shows the details of the "direct fast DM write" mode, which is a GPU-based matrix multiplication going from WFS pixels to DM actuators (no modal step). Slide #9 shows the red arrow in the center-left part of slide #6

… **operates on modes that are constructed from the master WFS response**. Slide #6 shows how the modes are constructed. Slides #19-23 show actual reconstruction performance in real-time (in lab)

… **requires accurate and stable (at ~10us level) timings** to allow pseudo open-loop telemetry reconstruction. Slide #15 shows the timing diagram for SCExAO

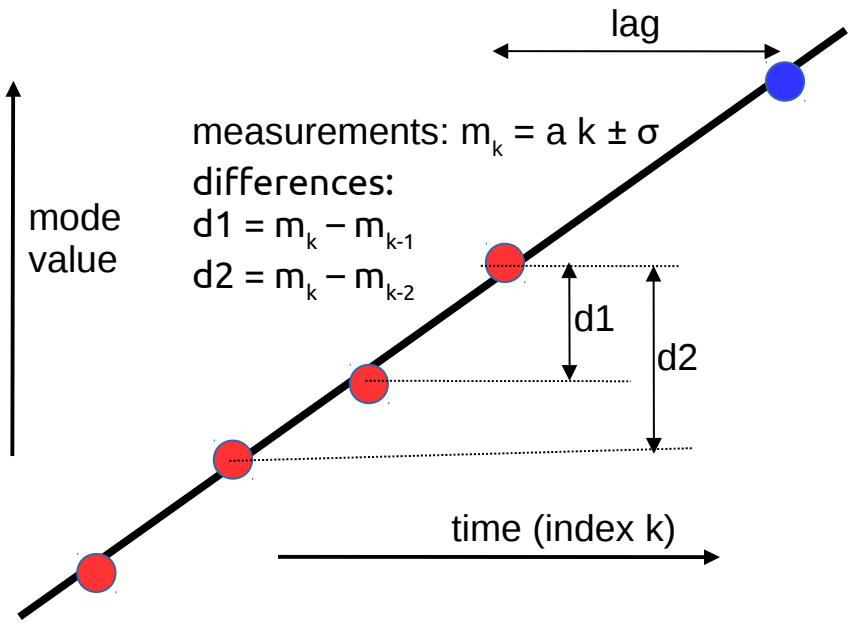
… **requires reliable WFS & DM calibration.** This is achieved by frequently taking on-sky response matrix at full speed modulation (3 kHz) while the loop the closed. Slide #16 shows how timing info allows fast RM acquisition: SCExAO now takes a RM in 2 sec by modulating the DM at 3 kHz (frames excluded = 0, frames averaged = 1)

Slide #14 shows the **overall data flow** with predictive control, including real-time pseudo open-loop reconstruction.

Automatic Gains Setting – Fast Mode

C code: *AOloopControl_AutoTuneGains* Script: *auxscripts/aolautotunegains*
Goal: **Find optimal gain for each mode in non-predictive mode in bright star regime**. This mode should be very reactive and robust, and able to recompute 1200 optimal gains in < 300 us to allow gain updates @ up to 3 kHz.
Bright star regime: input WF mode evolves linearly with time (control frequency > vibrations)

→ Error is quadratic sum of time lag and measurement noise, which can be expressed as simple functions of recent measurements.



With integrator (gain = g)

Time lag error:
$\sigma_{TL} = a\,(\,lag + 1/g\,)$

Measurement noise propagation:
$\sigma_{MN} = sqrt(g/(1-g))\,\sigma$

Estimating slope (a) and measurement noise (σ)

$\langle d1^2 \rangle = a^2 + 2\,\sigma^2$
$\langle d2^2 \rangle = 4\,a^2 + 2\,\sigma^2$
$\langle d3^2 \rangle = 9\,a^2 + 2\,\sigma^2$
$\langle d4^2 \rangle = 16\,a^2 + 2\,\sigma^2$

$a^2 = (\langle d2^2 \rangle - \langle d1^2 \rangle) / 3 = (\langle d4^2 \rangle - \langle d2^2 \rangle) / 12$
$\sigma^2 = (4\langle d1^2 \rangle - \langle d2^2 \rangle) / 6 = (4\langle d2^2 \rangle - \langle d4^2 \rangle) / 6$
$MstdDev^2 = \langle m^2 \rangle - \sigma^2$

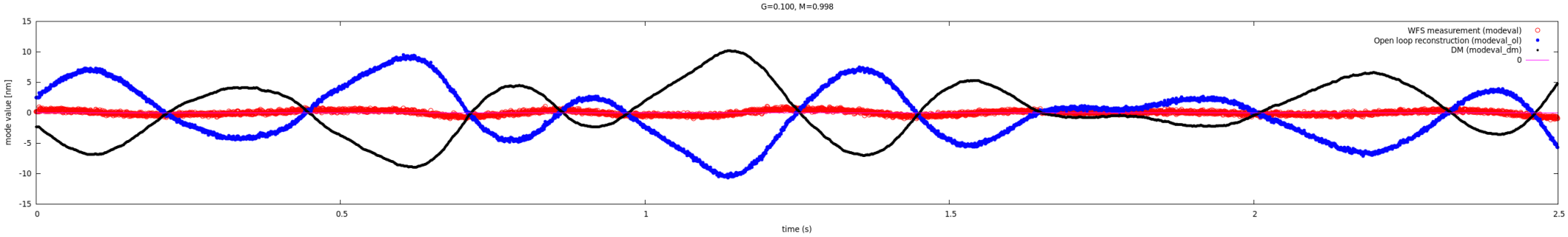
Real time process steps:
- Compute open loop coefficient mode values while loop is closed
- Update slope and measurement noise from running averages of $d1^2$ and $d2^2$
- Optimize $\sigma_{TL}^2 + \sigma_{MN}^2$ as a function of gain → update optimal gain

Open loop reconstruction

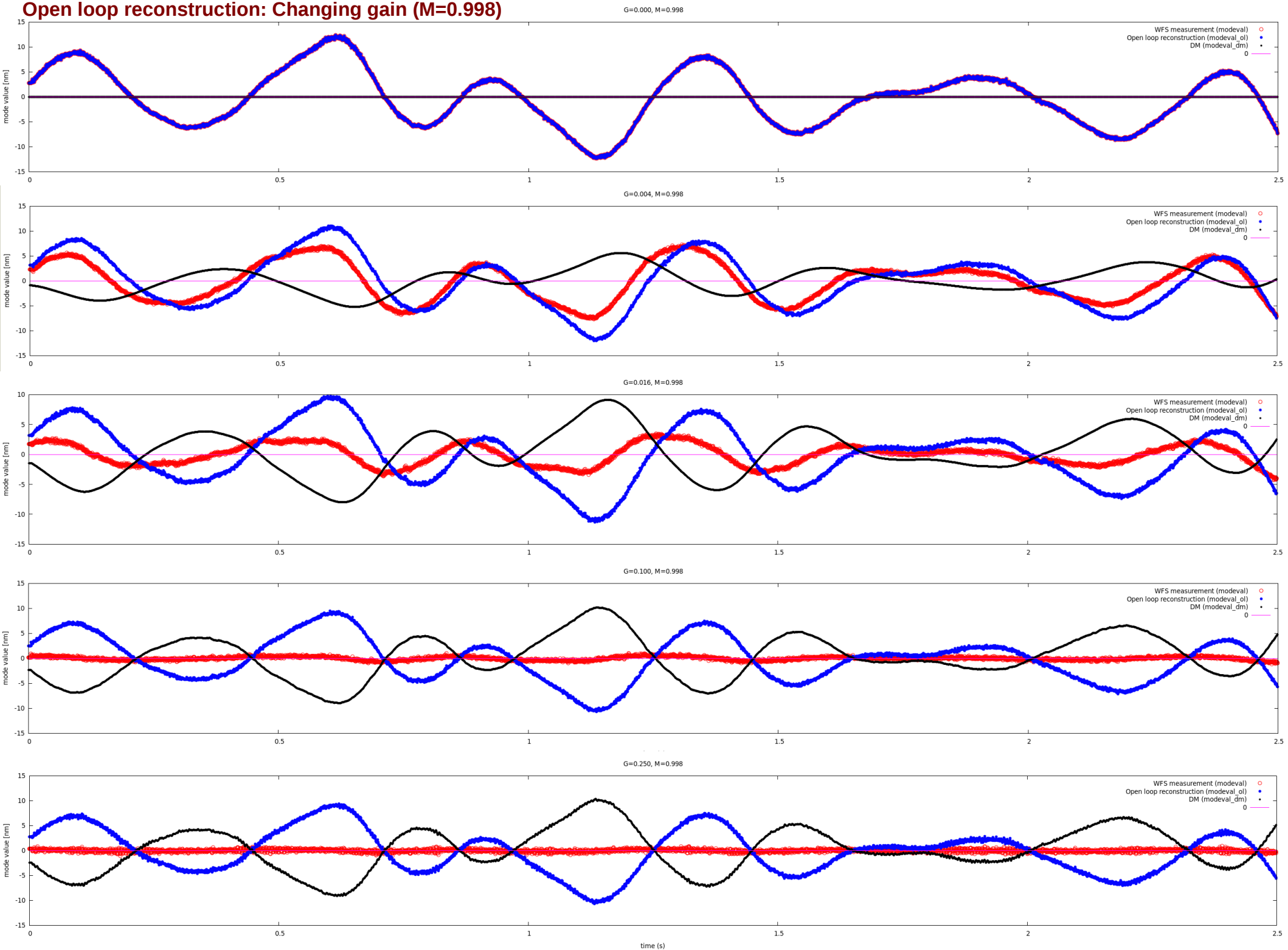
Turbulence injected:
8m/s wind speed, slightly filtered Kolmogorov spectrum (LOcoeff=0.1), 50nm RMS on DM (100nm WF), speed = 8 m/s
written to DM every 300us (3.333 kHz frequency)

Showing **Open loop (blue)** and **WFS measured (red)** modal coefficient #30
Data acquition start synchronized to turbulence sequence start

PyWFS running at 2kHz, 125mas modulation radius. Total delay = 2.6 frame

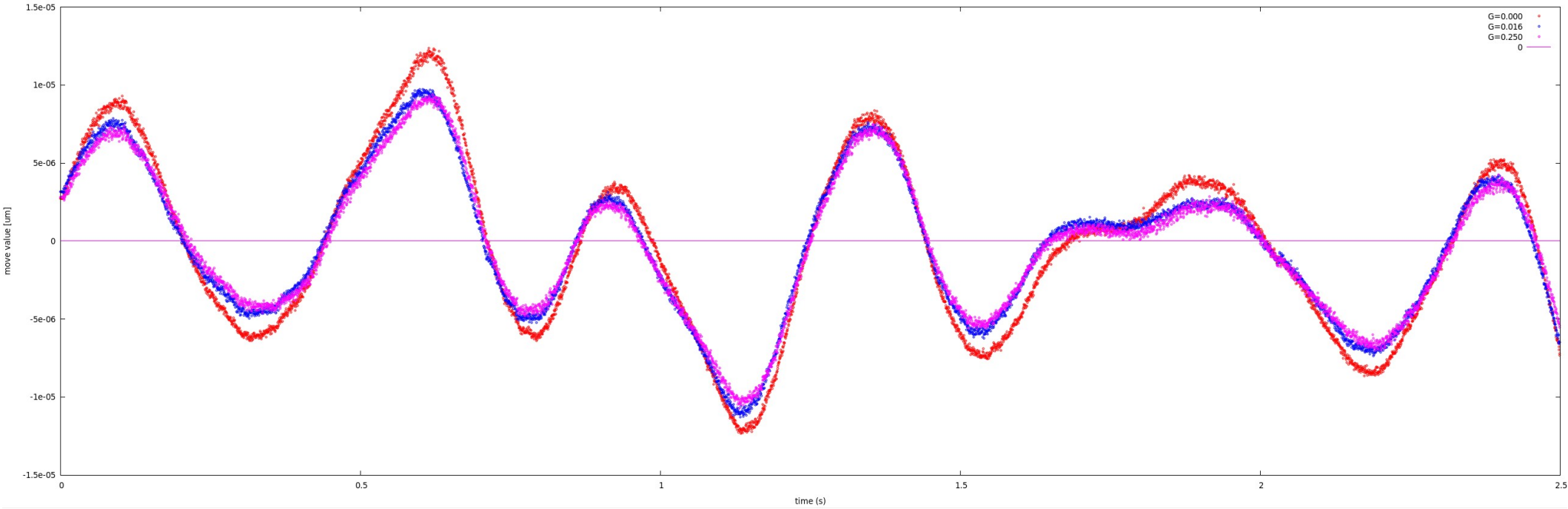


Open loop reconstruction: Changing gain (M=0.998)



Open loop reconstruction
Comparison between gain values

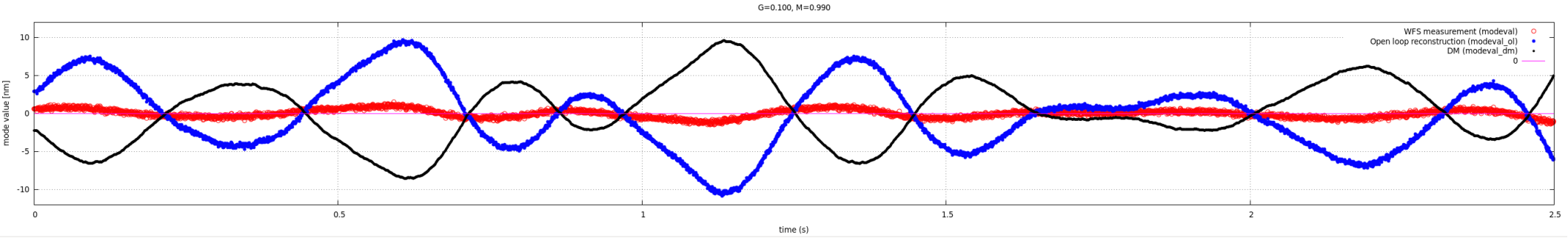
G=0.000 → over-estimates OL values
All G>0.0 reconstructions match at %-level

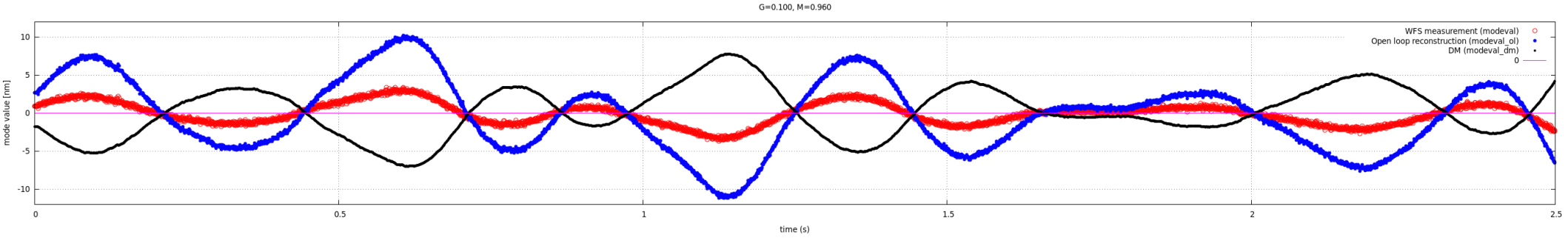


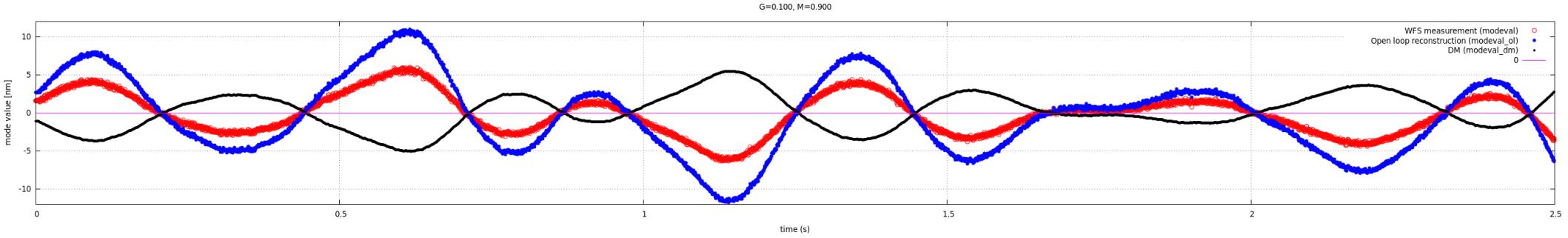
G=0.000 test relies entirely on WF residuals for OL estimation
G>0.000 tests rely mostly on DM values for OL estimation

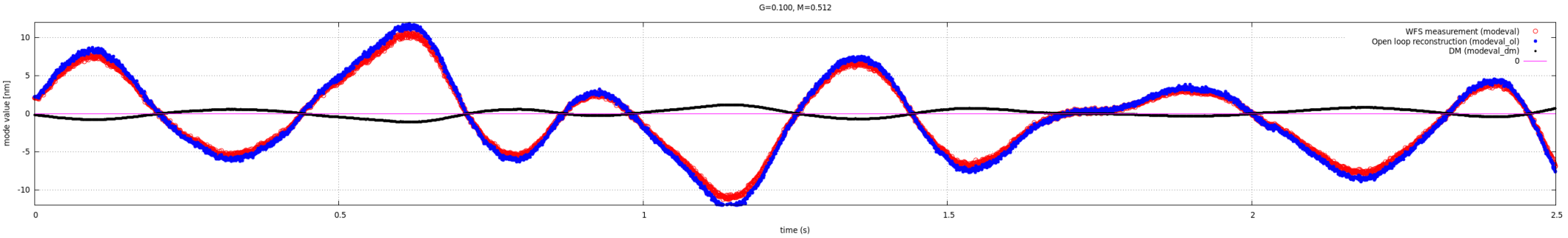
Test shown here uses full speed RM acquisition which underestimates RM by ~15% due to DM time-of-motion → reconstructed WFs from WFS are over-estimated by ~15%

Open loop reconstruction: Changing mult factor (G=0.100)



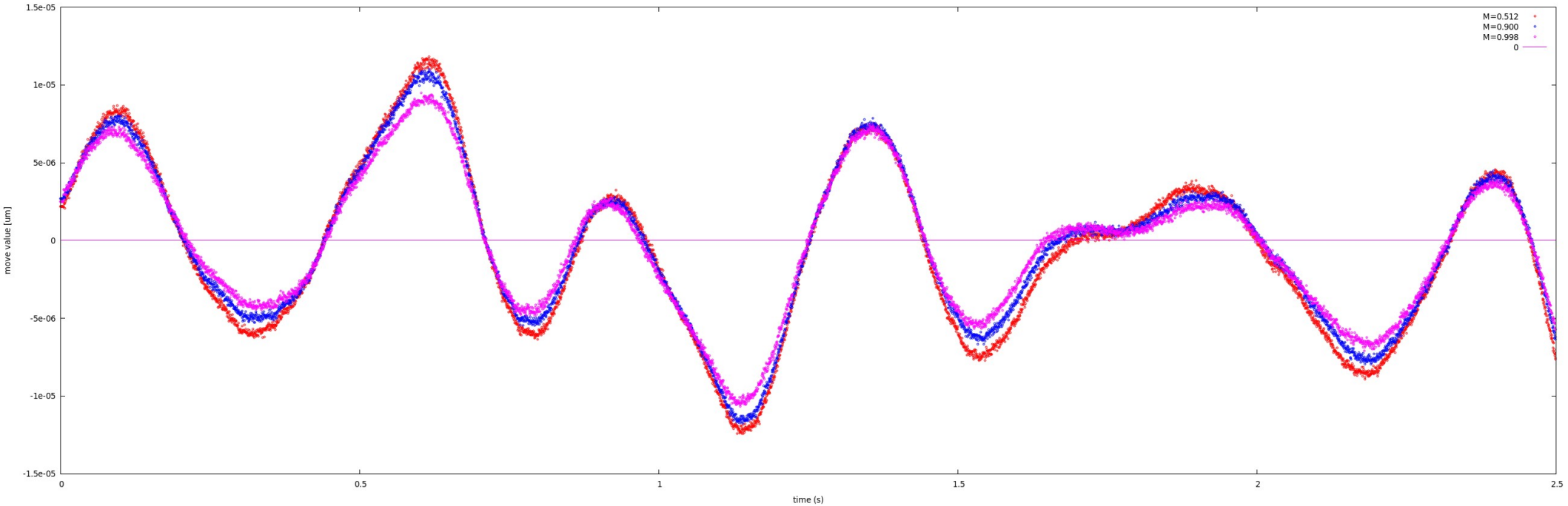






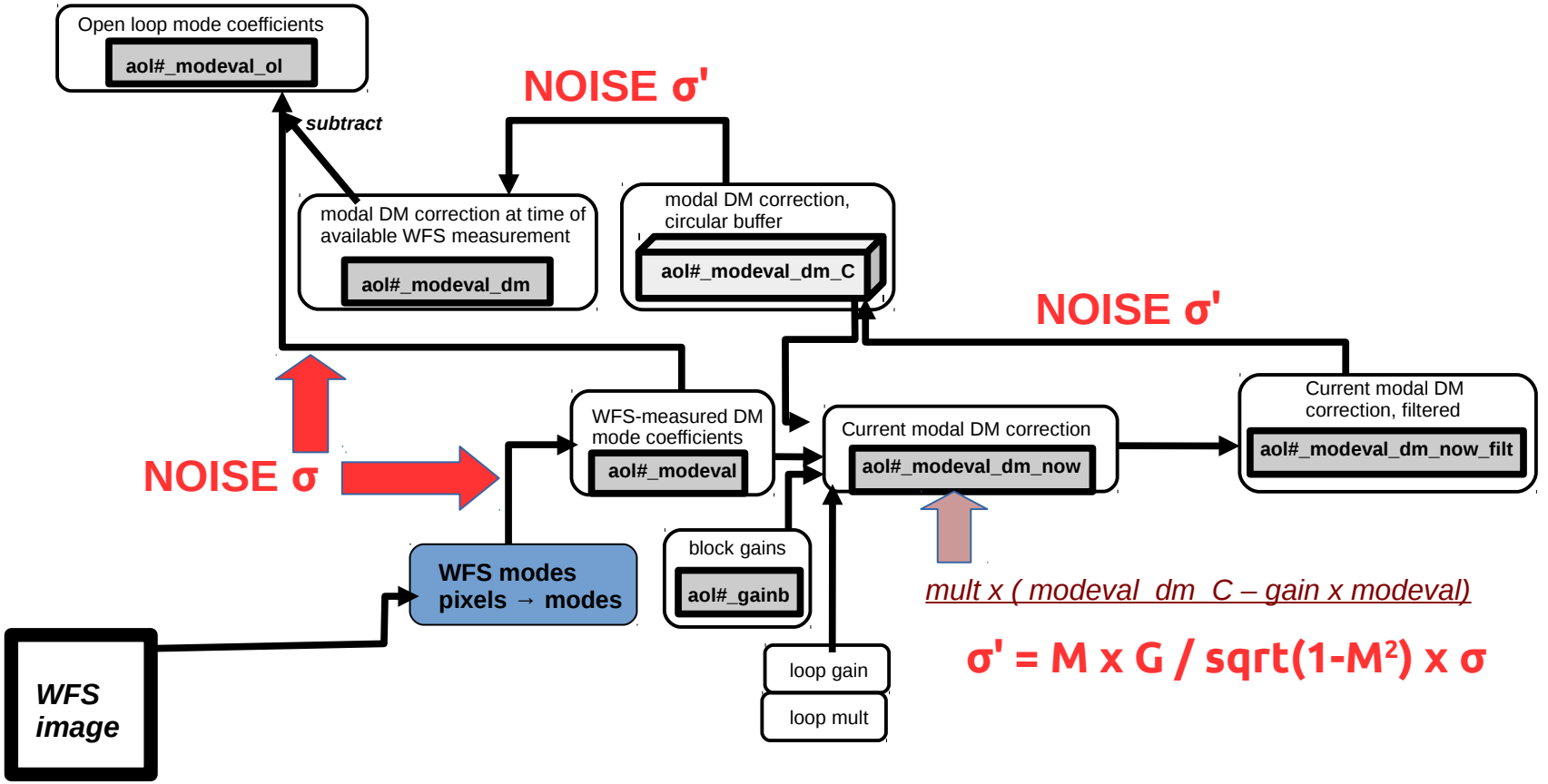
Open loop reconstruction
Comparison between Mult factor

Small M over-estimates OL values
→ same effect as G=0.0 over-estimation



M<<1 tests rely mostly on WF residuals for OL estimation
M~1 tests rely mostly on DM values for OL estimation

Test shown here uses full speed RM acquisition which underestimates RM by ~15% due to DM time-of-motion → reconstructed WFs from WFS are over-estimated by ~15%

Open loop reconstruction: noise propagation

$$NOISE = \sigma \times \sqrt{1 + M^2 G^2 / (1-M^2)}$$