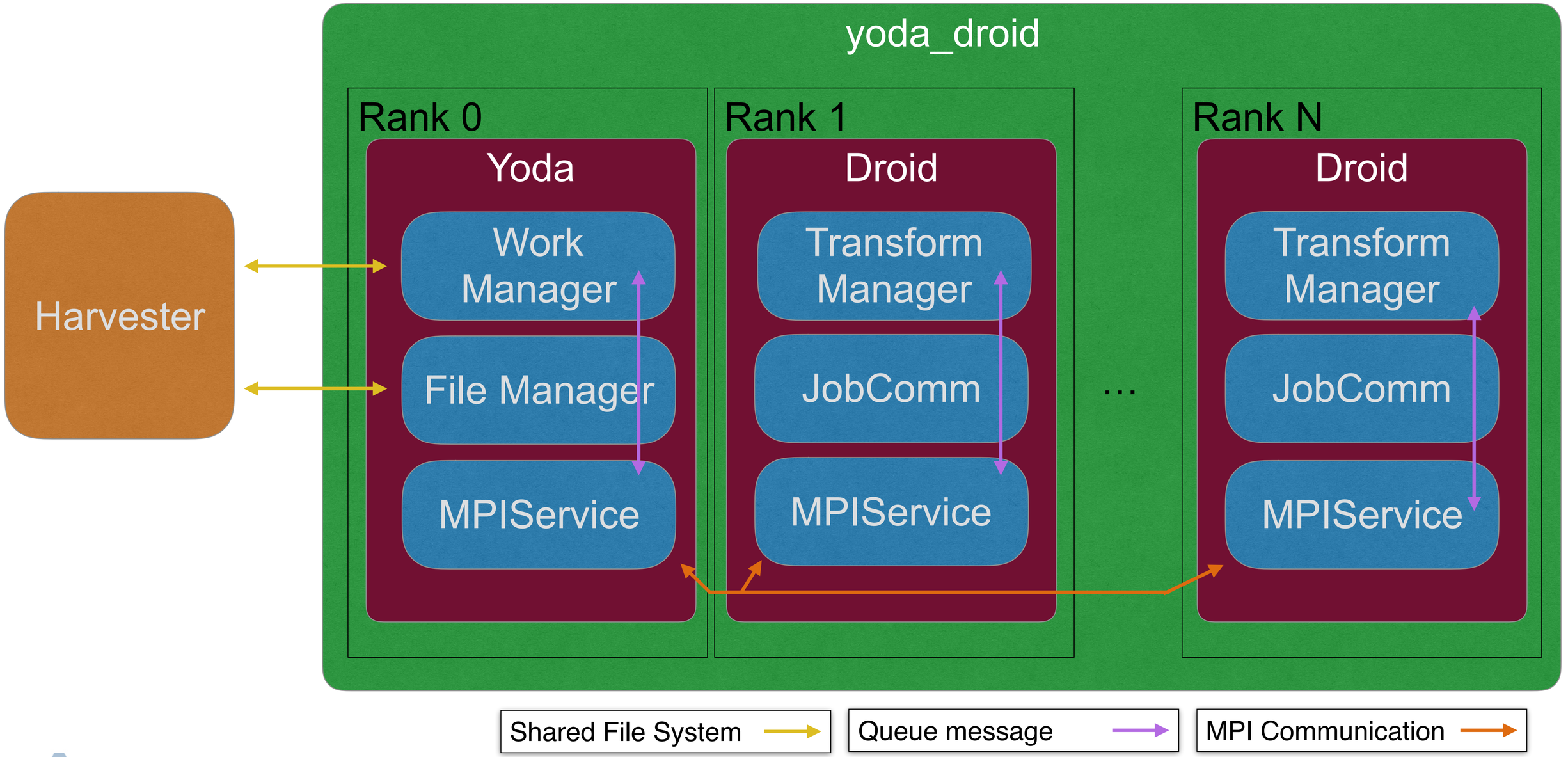
Argonne NATIONAL LABORATORY

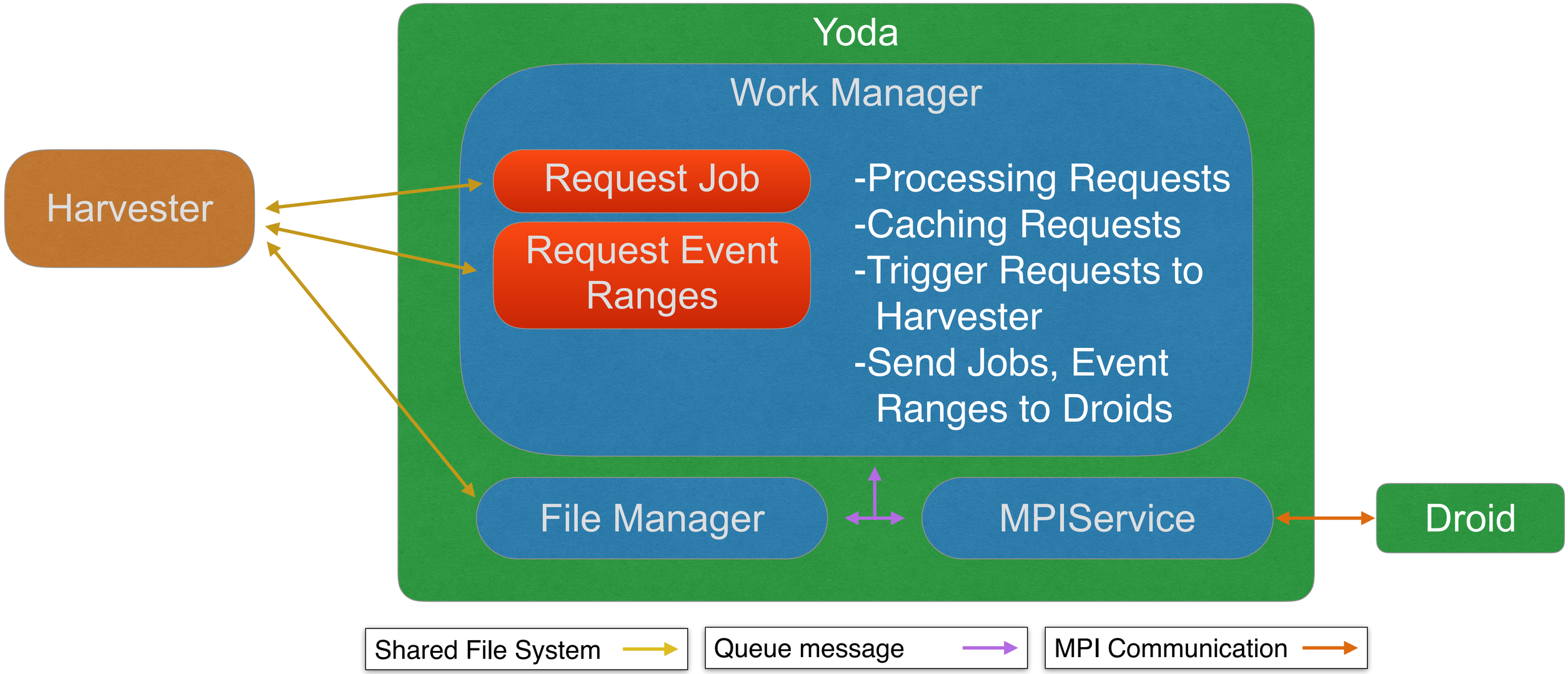
Yoda Technical Details and Current State

J. Taylor Childers (Argonne)

https://github.com/PanDAWMS/panda-yoda

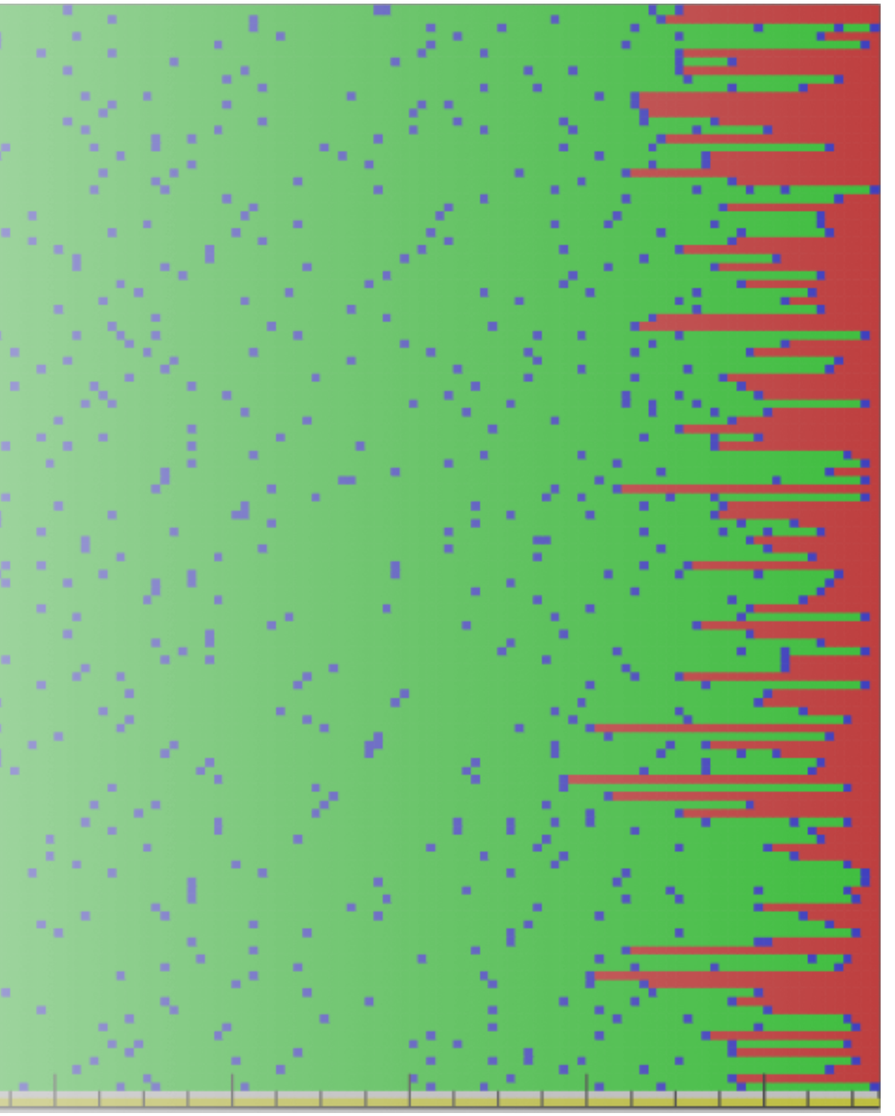
Yoda


yoda_droid
Rank 0
Yoda
Work Manager
File Manager
MPIService
Rank 1
Droid
Transform Manager
JobComm
MPIService
...
Rank N
Droid
Transform Manager
JobComm
MPIService
Harvester
Shared File System
Queue message
MPI Communication


Yoda
Work Manager
Request Job
Request Event Ranges
-Processing Requests
-Caching Requests
-Trigger Requests to Harvester
-Send Jobs, Event Ranges to Droids
Harvester
File Manager
MPIService
Droid
Shared File System
Queue message
MPI Communication

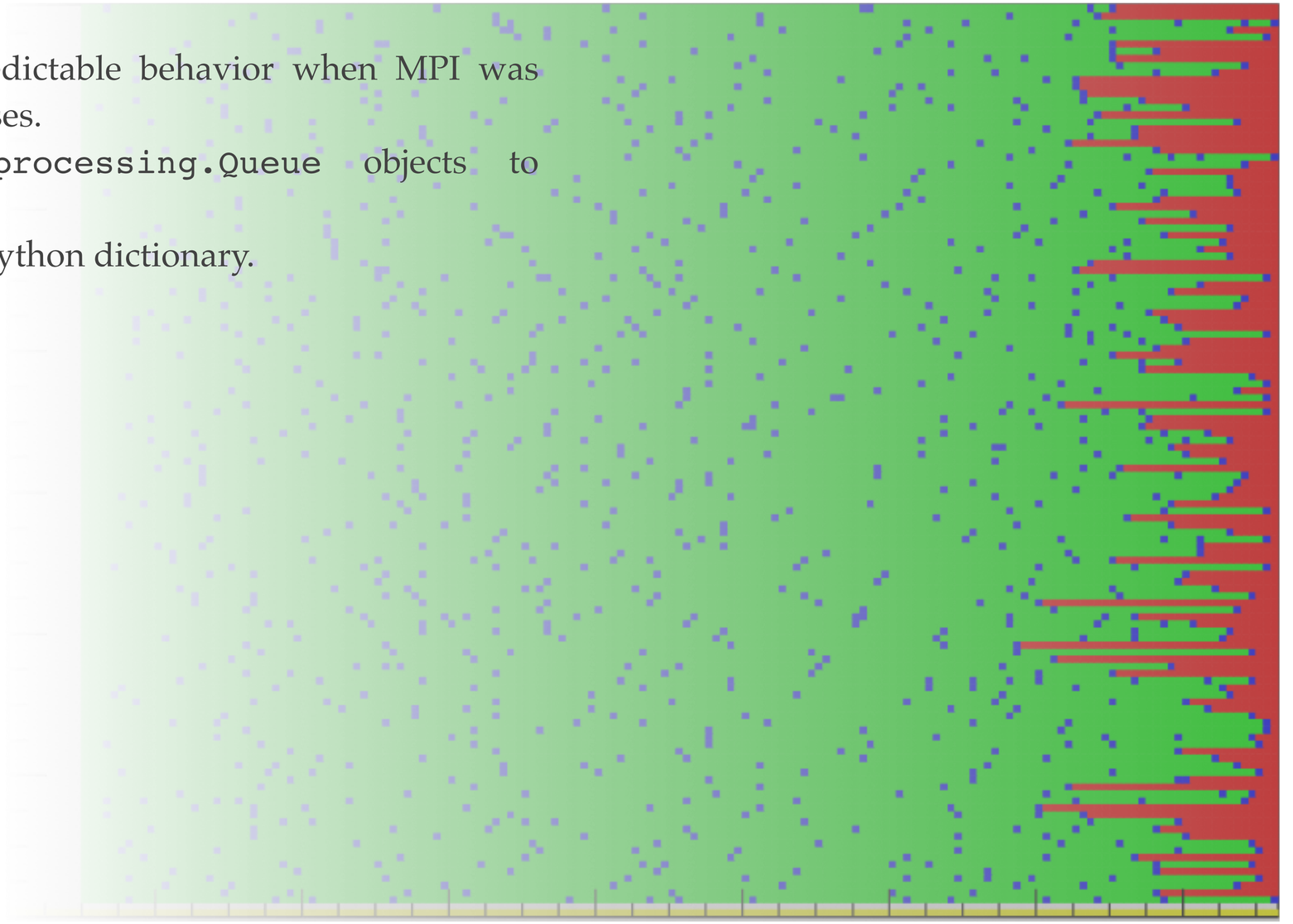
Yoda Module Design Details

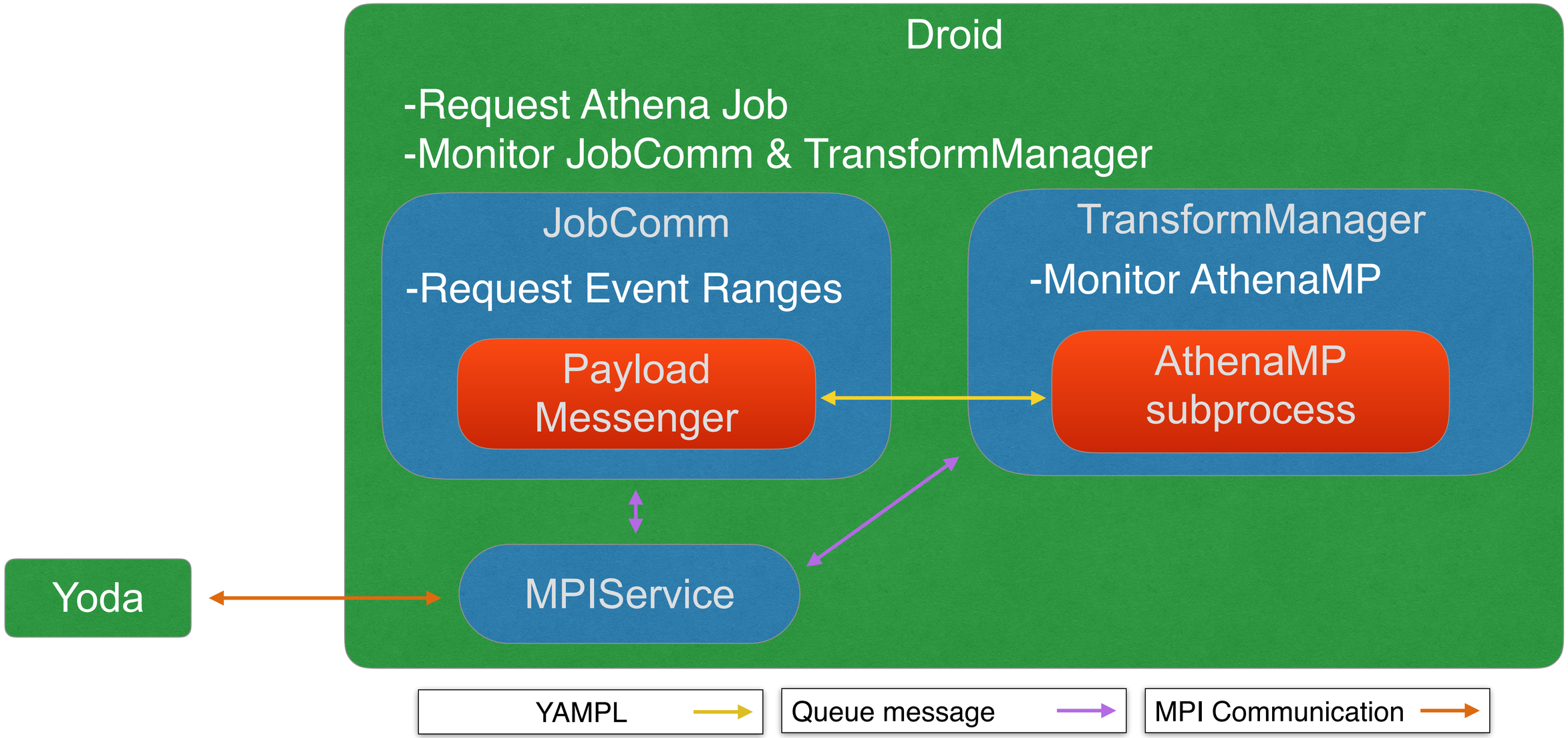
- Yoda-Droid is a server/client model, meaning Droids make requests to Yoda and Yoda fulfills those request.
  - The reason for this choice is to avoid needing a database for tracking the current state of each Droid.
  - Since our processing rate is on the scale of minutes per event, we don't expect Yoda to be deluged by incoming requests.
- Yoda launches the WorkManager, which processes all the incoming messages from Droids and triggers the requests from jobs or event ranges from Harvester.
- The Harvester interaction is treated as a plugin, currently using shared file system for communication, but could move to communication model.
  - Shared File System communication in this case is small and since only Rank 0 (Yoda) is writing/reading it is not a bottleneck.
  - However, it does create race conditions.
  - Should move to communication where possible, just need to implement the plugins for Harvester & Yoda.
- Yoda can be given an wall-clock time limit. It will run for the specified number of minutes, then kill all AthenaMP processes and exit cleanly.
  - Needed for supporting running AthenaMP in temporary storage locations.
  - Athena logs must be saved for posterity.
- Yoda launches a FileManager process that receives details from Droid ranks on files produced, which are then communicated to Harvester for stage out.



Note about MPIService Module

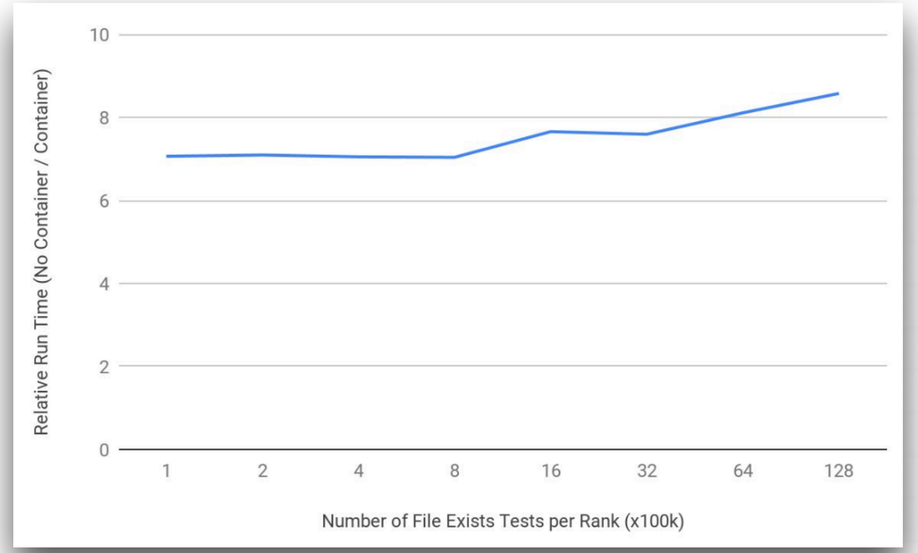
- All MPI traffic is handled in the MPIService
- This was driven because I have seen unpredictable behavior when MPI was spread out among Python Threads and Processes.
- Therefore all subprocesses use `multiprocessing.Queue` objects to communicate.
- Messages are always in the form of a pickled python dictionary.



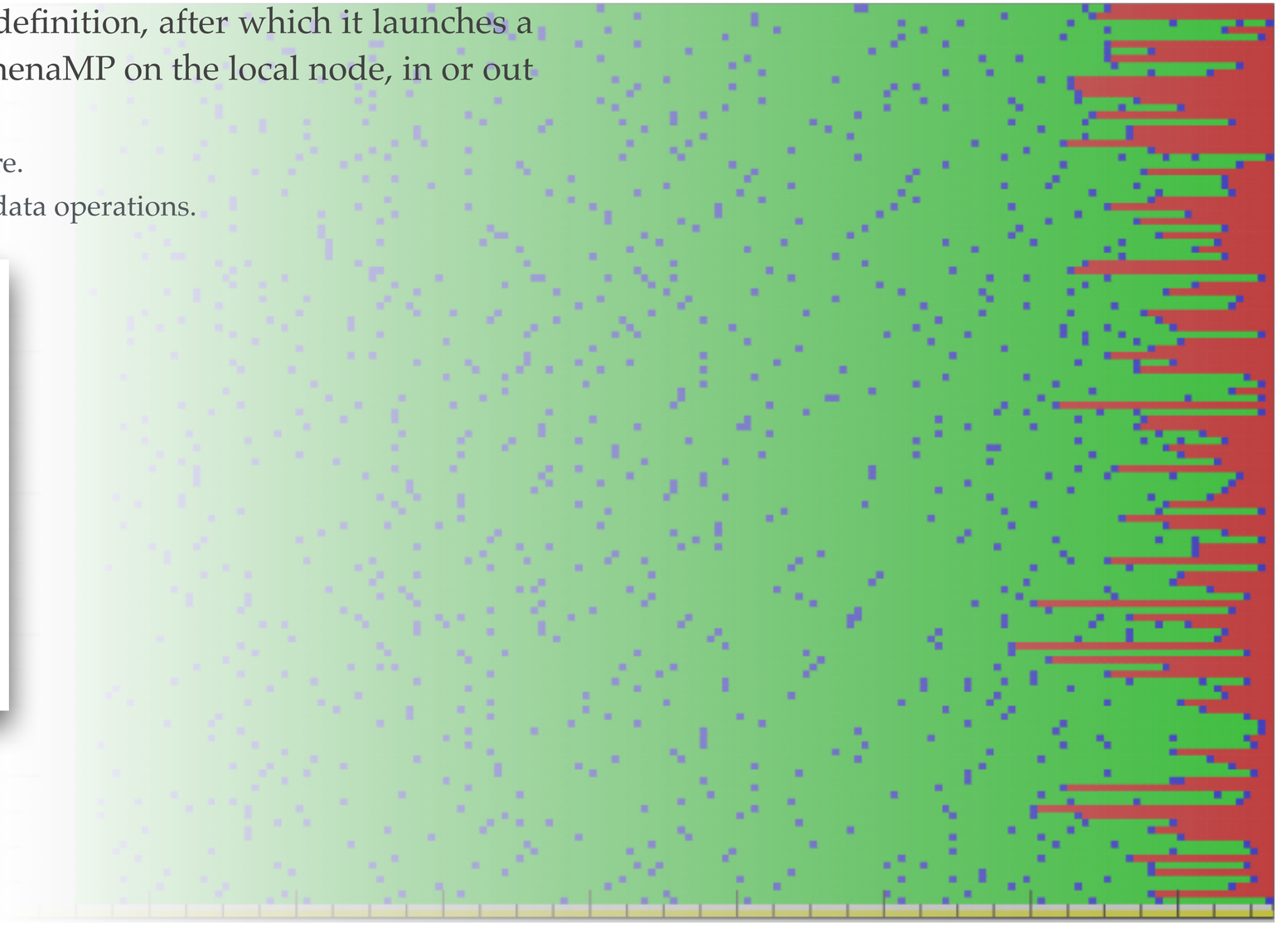

Droid
-Request Athena Job
-Monitor JobComm & TransformManager
JobComm
-Request Event Ranges
Payload Messenger
TransformManager
-Monitor AthenaMP
AthenaMP subprocess
Yoda
MPIService
YAMPL
Queue message
MPI Communication

Droid Module Design Details

- Droid sends a request to Yoda for a Panda Job definition, after which it launches a TransformManager sub-process which runs AthenaMP on the local node, in or out of a container as needed.
  - Containers are important for scaling the Atlas Software.
  - Provide a 7x speed-up when performing many meta-data operations.

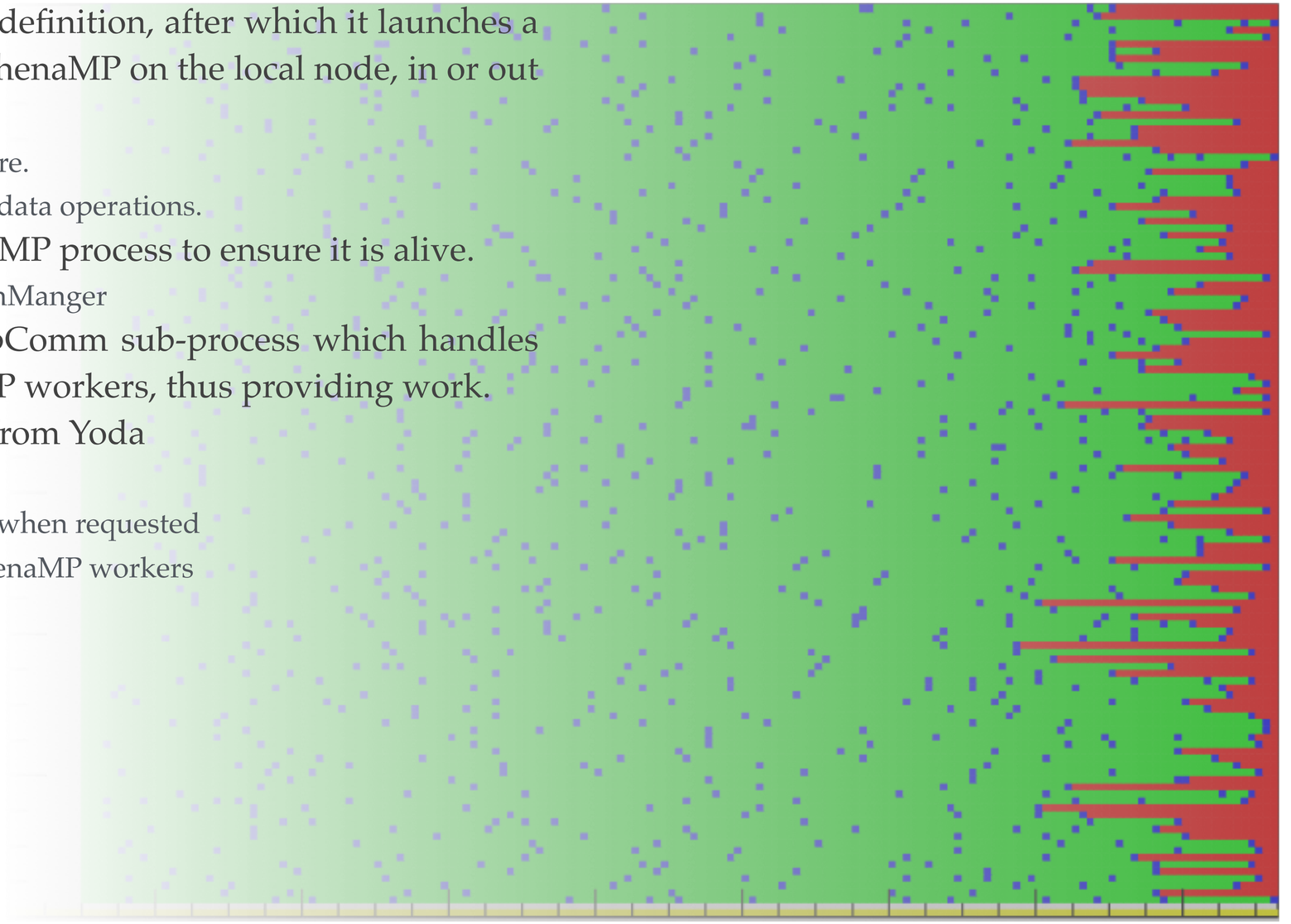


Spencer Williams
ANL Summer Student



Droid Module Design Details

- Droid sends a request to Yoda for a Panda Job definition, after which it launches a TransformManager sub-process which runs AthenaMP on the local node, in or out of a container as needed.
  - Containers are important for scaling the Atlas Software.
  - Provide a 7x speed-up when performing many meta-data operations.
- TransformManger simply monitors the AthenaMP process to ensure it is alive.
  - There is a lot to configure here, see slide on TransformManger
- Once a job has begun, Droid launches the JobComm sub-process which handles the YAMPL communication with the AthenaMP workers, thus providing work.
- JobComm manages the event ranges received from Yoda
  - sending requests to Yoda when more are needed
  - responding to AthenaMP workers with event ranges when requested
  - forwarding output file information to Yoda from AthenaMP workers



TransformManager Module

- Example for Singularity submission is here.
- Entries from PandaJob dictionary formatted into this template to fill things
- Using this template (following Harvester's example) enables local sites to configure their runtime environment.
  - Notice the DB settings.
- Current templates can be a bit Sim focused and will need to be made more flexible as we include more workflows, Gen/Reco.
- Or provide a way to change the template based on transforms.
- This assumes Event Service would be ported to other transforms.

```
1   #!/bin/bash
2   echo [$SECONDS] Start inside Singularity
3   echo [$SECONDS] DATE=$(date)
4   WORKDIR=$1
5   echo [$SECONDS] WORKDIR=$WORKDIR
6
7   cd $WORKDIR
8
9   export ATHENA_PROC_NUMBER=128 # AthenaMP workers per node
10
36  echo [$SECONDS] Setting up Atlas Software
37  RELEASE={release}
38  PACKAGE={package}
39  CMTCONFIG={cmtConfig}
40  echo [$SECONDS] RELEASE=$RELEASE
41  echo [$SECONDS] PACKAGE=$PACKAGE
42  echo [$SECONDS] CMTCONFIG=$CMTCONFIG
49  DBBASEPATH=$ATLAS_DB_AREA/DBRelease/current
50  export CORAL_DBLOOKUP_PATH=$DBBASEPATH/XMLConfig
51  export CORAL_AUTH_PATH=$DBBASEPATH/XMLConfig
52  export DATAPATH=$DBBASEPATH:$DATAPATH
53  mkdir poolcond
54  export DBREL_LOCATION=$ATLAS_DB_AREA/DBRelease
55  cp $DBREL_LOCATION/current/poolcond/*.xml poolcond
56  export DATAPATH=$PWD:$DATAPATH
57  unset FRONTIER_SERVER
70  echo [$SECONDS] Starting transformation
71  {transformation} {jobPars}
72  echo [$SECONDS] Transform exited with return code: $?
73  echo [$SECONDS] Exiting
```

https://github.com/PanDAWMS/panda-yoda/blob/master/templates/ThetaSubmitSingularity.sh

TransformManager Module

- When running on temp file systems, like RAM-disk or node-local SSDs, Yoda must clean up and stage out log files.
- To achieve this the user specifies the run path for the AthenaMP application.
- For instance, RAM-disk on Theta is '/tmp'
- The user also specifies BASH Globbing strings to specify which logs to target.
- For instance,
  "`log.EVNTtoHITS,athenaMP-workers-*/*/AthenaMP.log`"

```
263     # if job is to run elsewhere:
264     cwd = self.droid_working_path
265     if self.run_elsewhere:
266        cwd = self.run_directory
267
268     logger.info('running directory: %s',cwd)
269
270
271     logger.info('starting run_script: %s',cmd)
272
273     self.jobproc = subprocess.Popen(cmd.split(),
274                                     stdout=open(self.stdout_filename,'w'),
275                                     stderr=open(self.stderr_filename,'w'),
276                                     cwd=cwd,
277                                     env=env,
278                                    )
```

https://github.com/PanDAWMS/panda-yoda/blob/master/pandayoda/droid/TransformManager.py

TransformManager Module

- Sites sometimes need local modifications to the PandaJob dictionary in order to operate properly
- For instance, the Titan operators needed to remove a portion of the Athena command line options to get the database to work properly.
- Job mod is a plugin which takes the PandaJob dictionary, can modify it, and return the modified version for Yoda to run.

https://github.com/PanDAWMS/panda-yoda/blob/master/pandayoda/yoda_template.cfg

```
49  [TransformManager]
50  loglevel            = INFO
51  loop_timeout        = 60
52  template            = /lus/theta-fs0/projects/AtlasADSP/atlas/harvester/templates/ThetaSubmitSingularity.sh
53  run_script          = job_submit.sh
54  subprocess_stdout   = transform_stdout_r{rank}_pid{PandaID}.log
55  subprocess_stderr   = transform_stderr_r{rank}_pid{PandaID}.log
56  use_clean_env       = true
57  use_container       = true
58  container_prefix    = /usr/bin/singularity exec -B /lus/theta-fs0/projects/AtlasADSP:/lus/theta-fs0/projects/A
59  run_directory       = /tmp
60  run_elsewhere       = false
61  logs_to_stage       = log.EVNTtoHITS.athenaMP-workers-*/*/AthenaMP.log
62  jobmods             = no_file_validation,shorten_input_evnt_file_list,remove_dbrelease
```

https://github.com/PanDAWMS/panda-yoda/blob/master/pandayoda/droid/TransformManager.py

```
220     def apply_jobmods(self):
221         ''' apply job mods specified by the configuration settings '''
222
223         package_name = 'pandayoda.jobmod.'
224         for mod in self.jobmods:
225
226             module_name = package_name + mod
227             logger.info('importing jobmod: %s', module_name)
228             try:
229                 local_mod = importlib.import_module(module_name)
230             except ImportError:
231                 logger.exception('Failed to import jobmod: %s\n curret python path: %s',module_name,sys.path)
232                 raise
233
234             try:
235                 self.job_def = local_mod.apply_mod(self.job_def)
236             except Exception:
237                 logger.exception('Failed to execute jobmod: %s',module_name)
238                 raise
239
```


TransformManager Module

- Sites sometimes need local modifications to the PandaJob dictionary in order to operate properly
- For instance, the Titan operators needed to remove a portion of the Athena command line options to get the database to work properly.
- Jol
  di
  ve

https://github.com/PanDAWMS/panda-yoda/blob/master/pandayoda/yoda_template.cfg

```
[TransformManager]
loglevel            = INFO
loop_timeout        = 60
template            = /lus/theta-fs0/projects/AtlasADSP/atlas/harvester/templates/ThetaSubmitSingularity.sh
run_script          = job_submit.sh
subprocess_stdout   = transform_stdout_r{rank}_pid{PandaID}.log
subprocess_stderr   = transform_stderr_r{rank}_pid{PandaID}.log
use_clean_env       = true
use_container       = true
container_prefix    = /usr/bin/singularity exec -B /lus/theta-fs0/projects/AtlasADSP:/lus/theta-fs0/projects/A
run_directory       = /tmp
run_elsewhere       = false
logs_to_stage       = log.EVNTtoHITS.athenaMP-workers-*/*/AthenaMP.log
jobmods             = no_file_validation,shorten_input_evnt_file_list,remove_dbrelease
```

PanDAWMS / panda-yoda — Unwatch 6 | Star 0 | Fork 2

Code | Issues 8 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

Branch: master — panda-yoda / pandayoda / jobmod / — Create new file | Upload files | Find file | History

jtchilders adding plugin capability for job definition modification — Latest commit 59713bf on Aug 15

..		
__init__.py	adding plugin capability for job definition modification	a month ago
no_file_validation.py	adding plugin capability for job definition modification	a month ago
remove_dbrelease.py	adding plugin capability for job definition modification	a month ago
shorten_input_evnt_file_list.py	adding plugin capability for job definition modification	a month ago
unlimited_athenamp_messages.py	adding plugin capability for job definition modification	a month ago

https://github.com/PanDAWMS/panda-yoda/blob/master/pandayoda/droid/TransformManager.py

```
'
python path: %s',module_name,sys.path)
```

```
237             logger.exception('Failed to execute jobmod: %s',module_name)
238             raise
239
```

Custom Python Multiprocessing Module

- After investigating some 'bug' reports from Wen, I found that while the Python version inside the ATLAS environment claims to be 2.7 it contains some modules from 2.6 which are very different in implementation.
- To cope with this, I wrote the `yoda_multiprocessing` module to obfuscate this difference.
- Must set environment variable `RUN_YODA_IN_ATHENA` to get this custom module.
- Otherwise it is just a pass through for all standard
- Ideally somewhere down the road we can delete this and only use standard Python releases

```python
import os
from multiprocessing import *

# running inside the Athena release, at least for 21.0.15,
# the python version can be strange. While it reports Python 2.7
# some behaviors are like 2.6.
# Especially in multiprocessing module
# This file tries to masks these differences by overridding the
# module objects if the environment variable 'RUN_YODA_IN_ATHENA'
# is set.


if 'RUN_YODA_IN_ATHENA' in os.environ:
   print('Running with custom multiprocessing module')
   from multiprocessing import synchronize
   # for the Event object, in 2.6 the Event.wait(timeout=None) function
   # always returns None
   # for Python 2.7+ it returns True if the internal flag is set
   # it returns False otherwise. However the Python version 2.7
   # that comes with Athena has the 2.6 behavior

   class MyEvent(synchronize.Event):
      # overload the wait to behave like the 2.7+ version
      def wait(self,timeout=None):
         super(MyEvent,self).wait(timeout)
         return self.is_set()

   # override multiprocessing Event class with my version
   Event = MyEvent
else:
   print('Running with standard multiprocessing module')
```

https://github.com/PanDAWMS/panda-yoda/blob/master/pandayoda/common/yoda_multiprocessing.py

Deployment instructions

https://github.com/PanDAWMS/panda-yoda

- Instructions are on the github, people should send me their notes if there are hints for specific cases where the install needs to change.
- Typically I recommend installing inside a Python virtualenv.

README.md

panda-yoda

Requirements

- python 2.7+ (haven't actually tested with Python 3)
- mpi4py
- yampl
- python-yampl

Installation

If you are using a non-standard MPI library, which is common for most supercomputers, you will need to compile the `mpi4py` module against your local MPI libraries. In order to compile the `mpi4py` module you will need to compile the `Cython` module which `mpi4py` uses to wrap the C++ MPI library functions in Python.

In all cases you will need to compile the `yampl` program and the `python-yampl` wrapper for Yoda to use during communications with AthenaMP.

Here are rough instructions for this process, there are specific instructions for some sites further below.

Generic Instructions

```
# setup environment
# typically I setup the virtualenv of Harvester at this point
# if your HPC system uses modules to setup environments you may
# need to source some modules for an older GNU or move from
```

Deployment instructions

https://github.com/PanDAWMS/panda-yoda

- Instructions are on the github, people should send me their notes if there are hints for specific cases where the install needs to change.
- Typically I recommend installing inside a Python virtualenv.
- Build Cython, mpi4py, yampl, and python-yampl inside environment
- pip install yoda

```
# Custom build Cython with local compilers
git clone git@github.com:cython/cython.git
cd cython
git checkout tags/0.25.2 # later tags may also work (haven't tested)
# here cc and CC are place holders for whichever compiler your
# local system requires
CC=cc CXX=CC python setup.py build
export PYTHONPATH=$VIRTUAL_ENV/lib64/python2.7/site-packages
# this installs the py
# as harvester, etc.
CC=cc CXX=CC python se
```

```
# Custom build mpi4py with local compilers for worker nodes
git clone git@github.com:mpi4py/mpi4py.git
cd mpi4py
git checkout tags/2.0.0 # later tags may work (haven't tested)
##  edit mpi.cfg
##  under [mpi]
```

```
# next you need to install yampl which AthenaMP uses to communicate
git clone git@github.com:vitillo/yampl.git
cd yampl
# on some systems like Theta you need "LDFLAGS=-dynamic" to force dynamic linking
# if you find it causing problems on your system, remove it
./configure --prefix=$VIRTUAL_ENV/install CC=cc CXX=CC LDFLAGS=-dynamic
make -j 10 install
# in my case this fails in the zeromq folder
# so go into the zeromq directory and build by hand
cd zeromq
./configure --prefix=$
make -j 10 install
cd ..
# continue with yampl
make -j 10 install
```

```
# now you need to install the python bindings for yampl
git clone git@github.com:vitillo/python-yampl.git
cd python-yampl
# then run
python setup.py build_ext
# the last command may fail, if so,
# edit setup.py such that inside the 'include_dirs' inside the 'Extension' have
# both the path to the '$VIRTUAL_ENV/install/include/yampl'
# and '$VIRTUAL_ENV/install/include' and the 'extra_link_args' has
# '-L/path/to/yampl/install/lib'. Then run it again
# Run this to install the module
python setup.py install
```

```
# install yoda into the $VIRTUAL_ENV/python2.7/site-packages
pip install git+git://github.com/PanDAWMS/panda-yoda
```

Deployment instructions

https://github.com/PanDAWMS/panda-yoda

- Configuration includes a configuration file with many settings, all documented in the README.md
- configuration is divided by python module `configparser`
- many things can be tuned, how many events Yoda should request from Harvester, or Droid from Yoda.
- log filenames, template names, log levels… and so on.

```
[yoda_droid]
loop_timeout                = 60
wallclock_expiring_leadtime = 300

[Yoda]
loglevel                    = INFO
loop_timeout                = 60
messenger_plugin_module     = pandayoda.yoda.shared_file_messenger

[WorkManager]
loglevel
loop_timeout
send_n_eventrang
request_n_eventr

[RequestHarveste
loglevel
loop_timeout

[RequestHarveste
loglevel
loop_timeout
eventrange_timeo
```

```
[shared_file_messenger]
loglevel                    = INFO
harvester_config_file       = /lus/theta-fs0/projects/AtlasADSP/atlas/harvester/etc/panda/panda_harvester.cfg

[FileManager]
loglevel                    = INFO
loop_timeout                = 60
output_file_type            = es_output
harvester_output_timeout    = 10

[Droid]
loglevel                    = INFO
loop_timeout                = 60
```

```
[TransformManager]
loglevel            = INFO
loop_timeout        = 60
template            = /lus/theta-fs0/projects/AtlasADSP/atlas/harvester/templates/ThetaSubmitSingularity.sh
run_script          = job_submit.sh
subprocess_stdout   = transform_stdout_r{rank}_pid{PandaID}.log
subprocess_stderr   = transform_stderr_r{rank}_pid{PandaID}.log
use_clean_env       = true
use_container       = true
container_prefix    = /usr/bin/singularity exec -B /lus/theta-fs0/projects/AtlasADSP:/lus/theta-fs0/projects/AtlasADSP:rw
run_directory       = /tmp
run_elsewhere       = false
logs_to_stage       = log.EVNTtoHITS,athenaMP-workers-*/*/AthenaMP.log
jobmods             = no_file_validation,shorten_input_evnt_file_list,remove_dbrelease

[MPIService]
loglevel                    = INFO
debug_message_char_length   = 200
default_message_buffer_size = 10000000
loop_timeout                = 30
```

Deployment instructions

https://github.com/PanDAWMS/panda-yoda

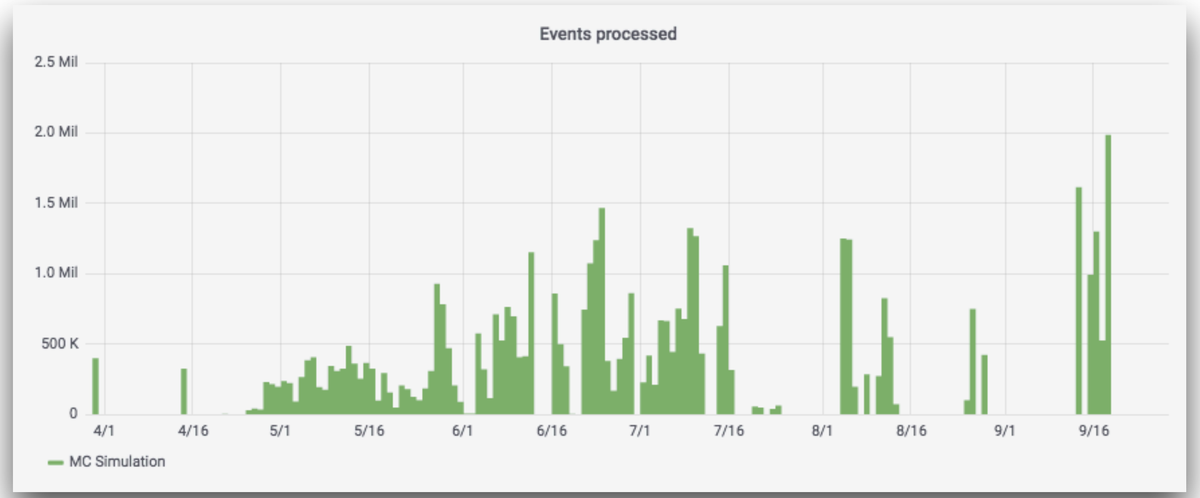
- Yoda should not require any coding unless you need custom ‘jobmods’
- Otherwise, the template run script for AthenaMP is the major piece.

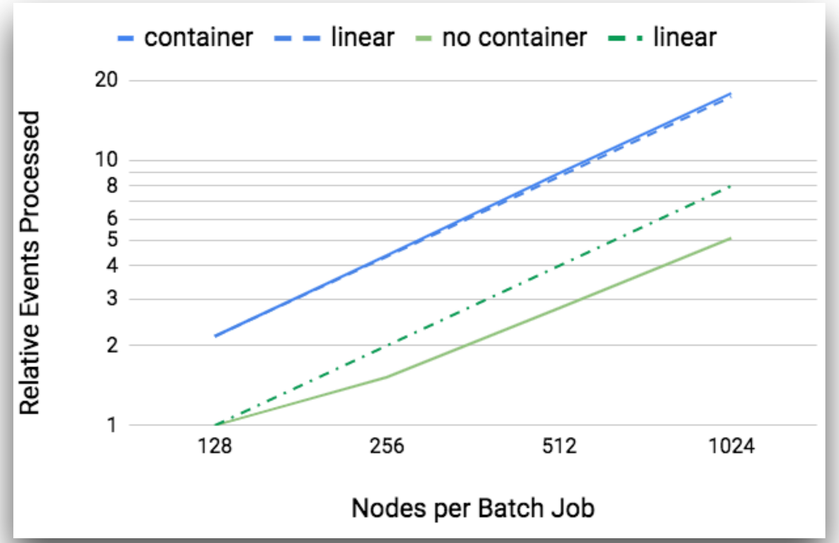
```
[yoda_droid]
loop_timeout                = 60
wallclock_expiring_leadtime = 300

[Yoda]
loglevel                    = INFO
loop_timeout                = 60
messenger_plugin_module     = pandayoda.yoda.shared_file_messenger

[WorkManager]
loglevel
loop_timeout
send_n_eventrang
request_n_eventr

[RequestHarveste
loglevel
loop_timeout

[RequestHarveste
loglevel
loop_timeout
eventrange_timeo
```

```
[shared_file_messenger]
loglevel                    = INFO
harvester_config_file       = /lus/theta-fs0/projects/AtlasADSP/atlas/harvester/etc/panda/panda_harvester.cfg

[FileManager]
loglevel                    = INFO
loop_timeout                = 60
output_file_type            = es_output
harvester_output_timeout    = 10

[Droid]
loglevel                    = INFO
loop_timeout                = 60
```

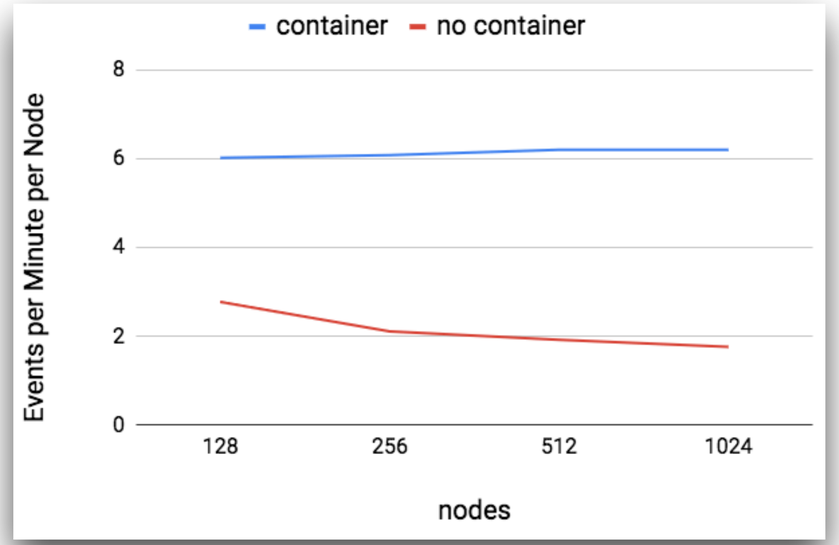
```
[TransformManager]
loglevel            = INFO
loop_timeout        = 60
template            = /lus/theta-fs0/projects/AtlasADSP/atlas/harvester/templates/ThetaSubmitSingularity.sh
run_script          = job_submit.sh
subprocess_stdout   = transform_stdout_r{rank}_pid{PandaID}.log
subprocess_stderr   = transform_stderr_r{rank}_pid{PandaID}.log
use_clean_env       = true
use_container       = true
container_prefix    = /usr/bin/singularity exec -B /lus/theta-fs0/projects/AtlasADSP:/lus/theta-fs0/projects/AtlasADSP:rw
run_directory       = /tmp
run_elsewhere       = false
logs_to_stage       = log.EVNTtoHITS,athenaMP-workers-*/*/AthenaMP.log
jobmods             = no_file_validation,shorten_input_evnt_file_list,remove_dbrelease

[MPIService]
loglevel                    = INFO
debug_message_char_length   = 200
default_message_buffer_size = 10000000
loop_timeout                = 30
```

Performance on Theta



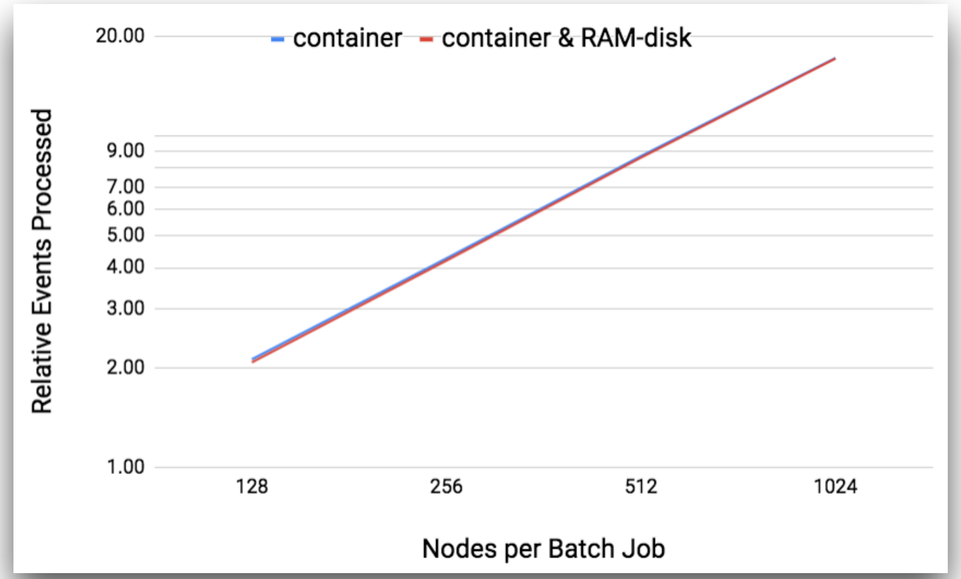


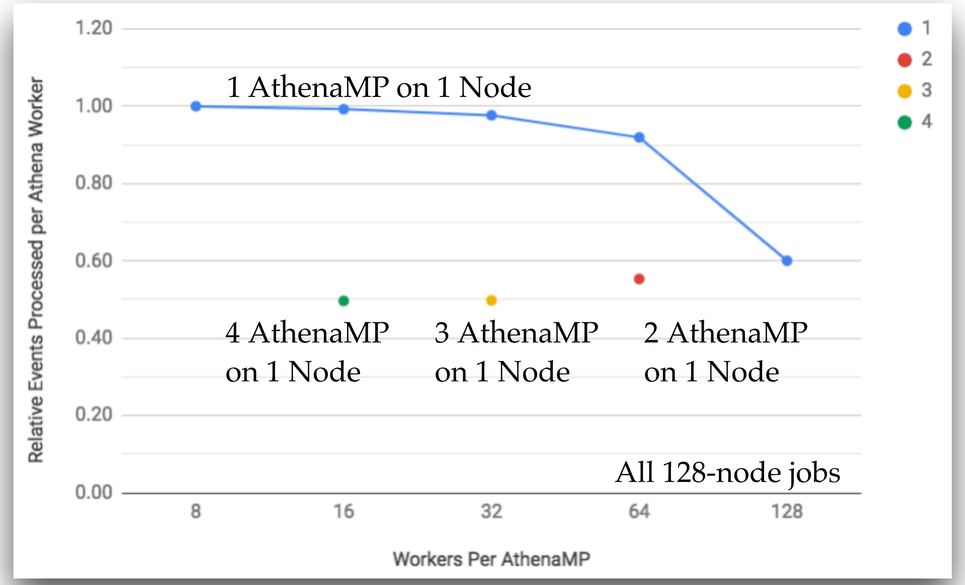
- Yoda has been running continuously since about May on Theta with Harvester.
- I have run scaling test of Yoda (outside of Harvester) up to 1024 nodes of Theta with very good performance inside a container.
- Grafana Link



Performance on Theta/NERSC

- One additional Study: with/without running AthenaMP work directory in RAM-disk
- No difference in runtime.
  Implies we are not limited by IO at this time.
- So are we limited by Memory? AthenaMP uses ~ 1.5GB + 0.25GB * (# AthenaMP workers)
- KNL has 16GB high bandwidth memory (MCDRAM) which has 4x the bandwidth as the 192GB DDR.
- Can run using 16GB in 'L3-cache' mode. If we are cache limited, reducing number of workers to fit AthenaMP inside 16GB (so < 32 workers) we should see improved performance.
- However, we do not see a big improvement at that step. So that is not a problem.
- Most likely remaining issue is L1/2 cache thrashing. Each worker requires different codes at any given time, therefore we never keep the same code in L1 cache for very long.
- Breaking up our event processing into smaller pieces may help us gain some performance by ensuring we limit the amount of code loaded on a CPU at any given time.





To Dos:

- Move from `shared_file_messenger` to communication layer for Harvester interaction.
  - more reliable, less race conditions
- More testing of ‘run elsewhere’ functionality to ensure it is generalized.
- Check Python 3 compatibility