HPC Usage in ATLAS - Where Does XRootD fit in

Doug Benjamin
Duke University

Taylor Childers
ANL

Vakho Tsulaia
LBNL

Acknowledgments

- The real authors of this talk (most of the slides come from them)
  - Taylor Childers (ANL)
  - Vakho Tsulaia (LBNL)
- Folks working on event service -
  - D Benjamin, P Calafiura, T Childers, K De, W Guan, T Maeno,P Nilsson, V Tsulaia, P Van Gemmeren and T Wenaus

DOE HPC Machines used by ATLAS

High Performance Computers

- 48k Nodes: 64 threads, 16GB each
- 1.6 GHz BlueGeneQ

- 5,200 nodes: 24 cores per node
- 2x2.4GHz Intel Ivy Bridge
- 24 GB DDR3 1866 MHz
- 1.1B core-hours/year (Grid ~2.5B/year)

OLCF
Oak Ridge Leadership Computing Facility

- 18,688 nodes: 16 CPU cores, 1 NVIDIA Kepler GPU
- 2.2GHz AMD Opteron with 32GB

High Performance Computers

- 48k Nodes:
  each
- 1.6 GHz

- 5,200 nodes; 24 cores per node
  e
  id ~2.5B/year)

CPU cores, 1

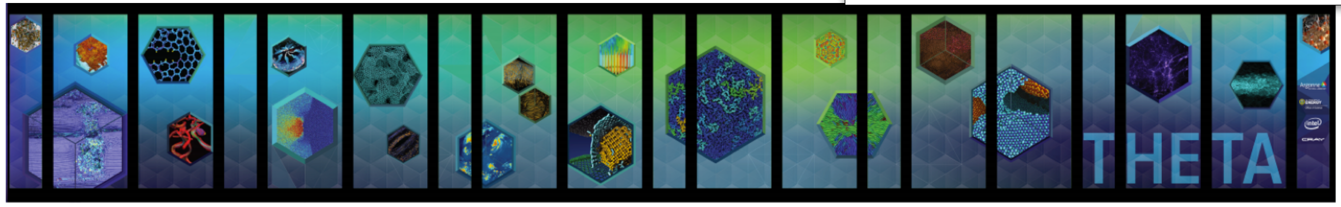
with 32GB

Currently: 10B core-hours per year on HPCs

LHC Grid Usage ~2.5B per year

High Performance Computers

Coming online in the next few months



Argonne NATIONAL LABORATORY | Leadership Computing Facility

- 3240 nodes: 64 cores x 4 HW threads
- 256 threads/node
- Intel Xeon Phi (Knights Landing)
- 16GB on-chip memory

- 9,304 nodes: 68 cores x 4 HW threads
- 272 threads/node
- Intel Xeon Phi (Knights Landing)

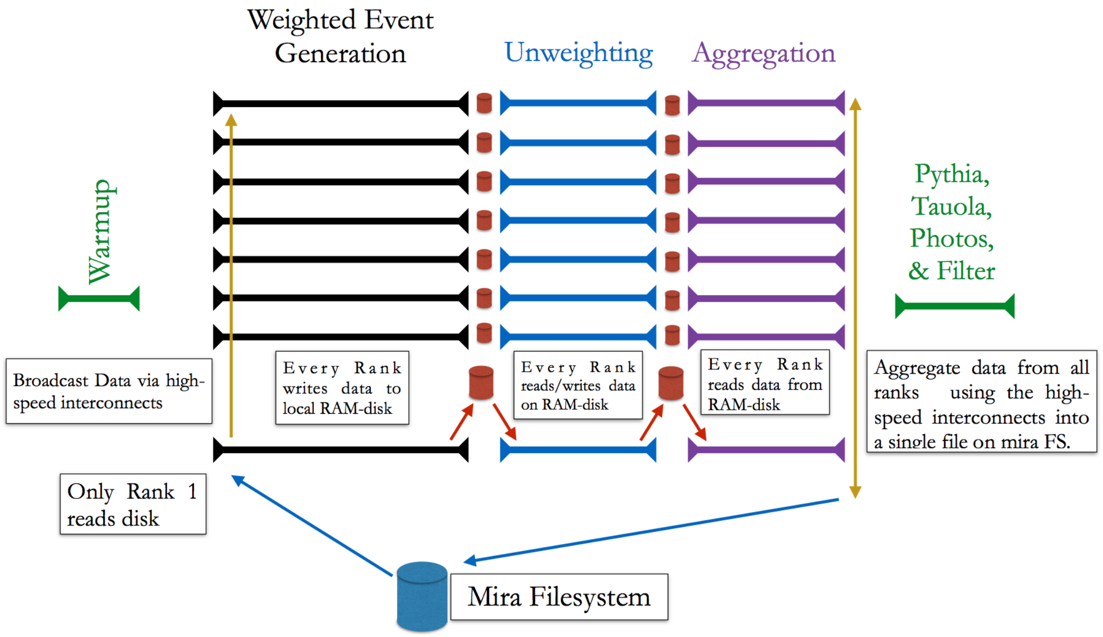
Current/Future work loads

Event Generation

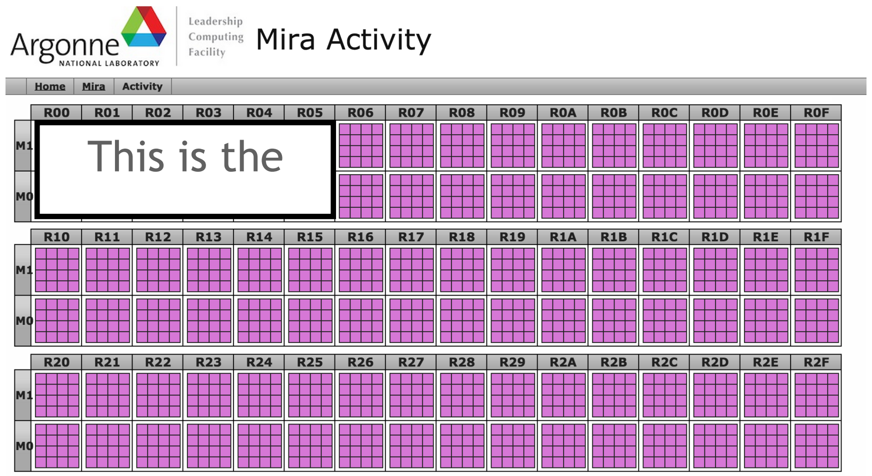
tiny input (card deck) small output - ~ MB's - GB's

Where We Started: Alpgen, an LO Generator

- Alpgen is an LO parton generator written in Fortran
- Every process gets a binary
- Most configurable settings are values of physics constants and do not affect program flow
- Ran serial Alpgen in parallel with minimal MPI additions for random number seeds and file I/O
- Used RAM-disks for intermediate data
- Allowed to fill Mira (6th fastest on the Top500) with the largest generation job ever. 1.5M parallel processes



Weighted Event Generation
Unweighting
Aggregation
Warmup
Pythia, Tauola, Photos, & Filter
Broadcast Data via high-speed interconnects
Every Rank writes data to local RAM-disk
Every Rank reads/writes data on RAM-disk
Every Rank reads data from RAM-disk
Aggregate data from all ranks using the high-speed interconnects into a single file on mira FS.
Only Rank 1 reads disk
Mira Filesystem



Argonne NATIONAL LABORATORY
Leadership Computing Facility
Mira Activity
Home
Mira
Activity
R00 R01 R02 R03 R04 R05 R06 R07 R08 R09 R0A R0B R0C R0D R0E R0F
R10 R11 R12 R13 R14 R15 R16 R17 R18 R19 R1A R1B R1C R1D R1E R1F
R20 R21 R22 R23 R24 R25 R26 R27 R28 R29 R2A R2B R2C R2D R2E R2F
M1
M0
This is the

Biggest Single Event Generation Job Ever Run

1,572,864 threads — Produced 85M W+5jet events

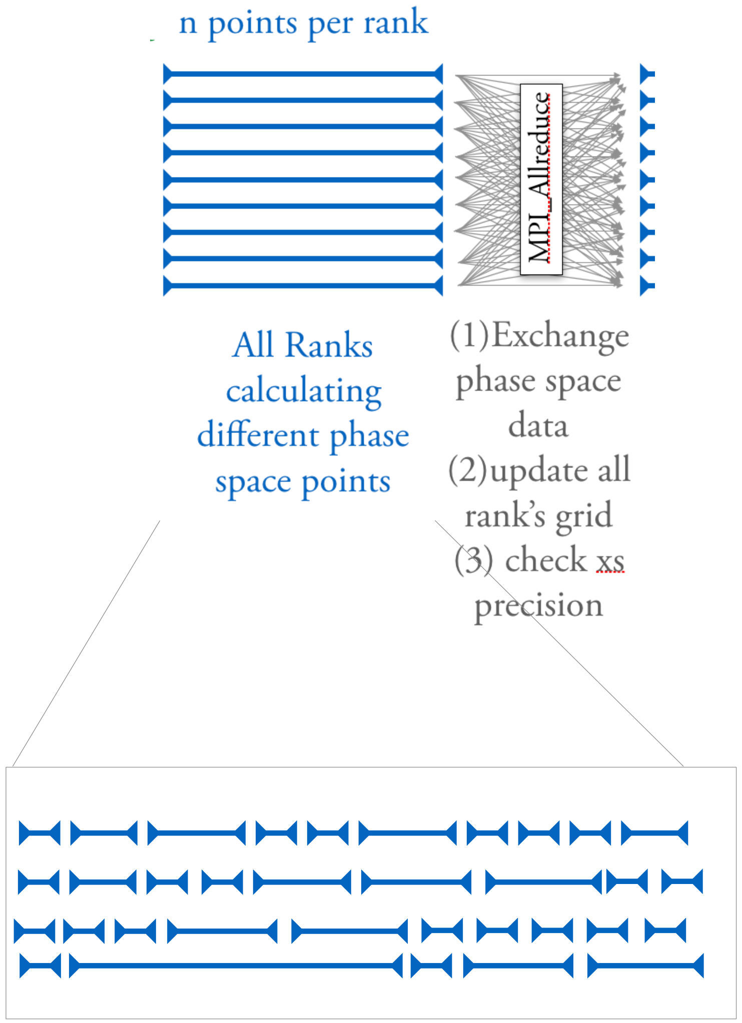
Run Time 10:52 — 250,000 CPU-hours

Sherpa, a Next-to-Leading-Order Generator

- Sherpa is an NLO event generation framework that supports many pluggable algorithms (both LO & NLO)
- Sherpa is a much more complex code AND framework than Alpgen.
- Since it supports multiple plugins and integrators it has much more program flow, meaning the CPU spends much of its time deciding which code is going to be run next.
- Unfortunately, increased flexibility causes decreased performance
- But there's hope, Sherpa supports MPI and supported pthreads in the past.

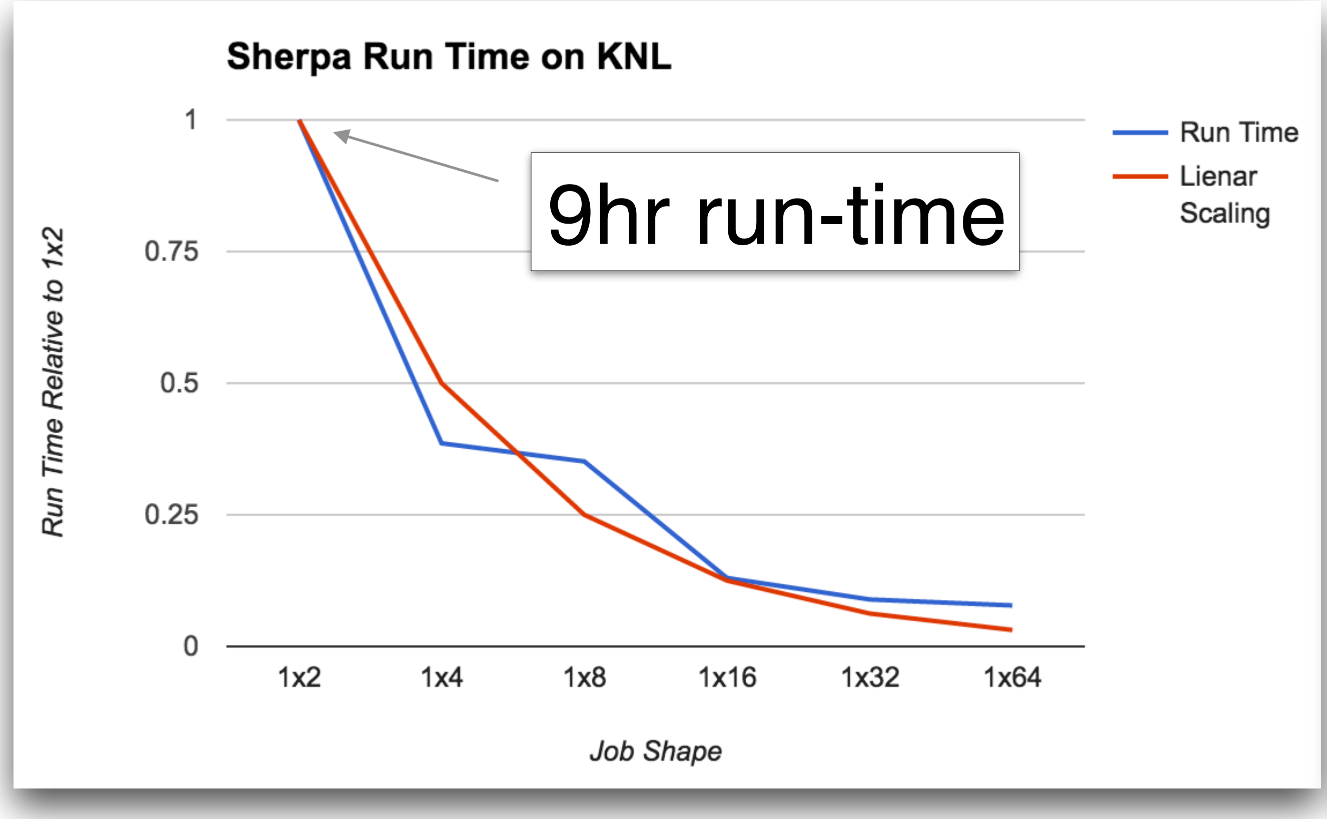
Some of the Sherpa Developments

- Stefan Hoche (SLAC) has been very supportive of our efforts to run Sherpa at large scales, providing many updates and patches to improve performance, remove old inefficiencies.
- Many Framework improvements
- Reduced 'fstat' calls and file system crawling which is slow with thousands of processes
- Reducing number of shared libraries to load
- Removing system specific code
- Reductions in memory consumption
- Instead of each process calculating N phase space points between MPI_Allreduce, a time period can be specified, allowing process to accomplish as much as they can between communication steps.

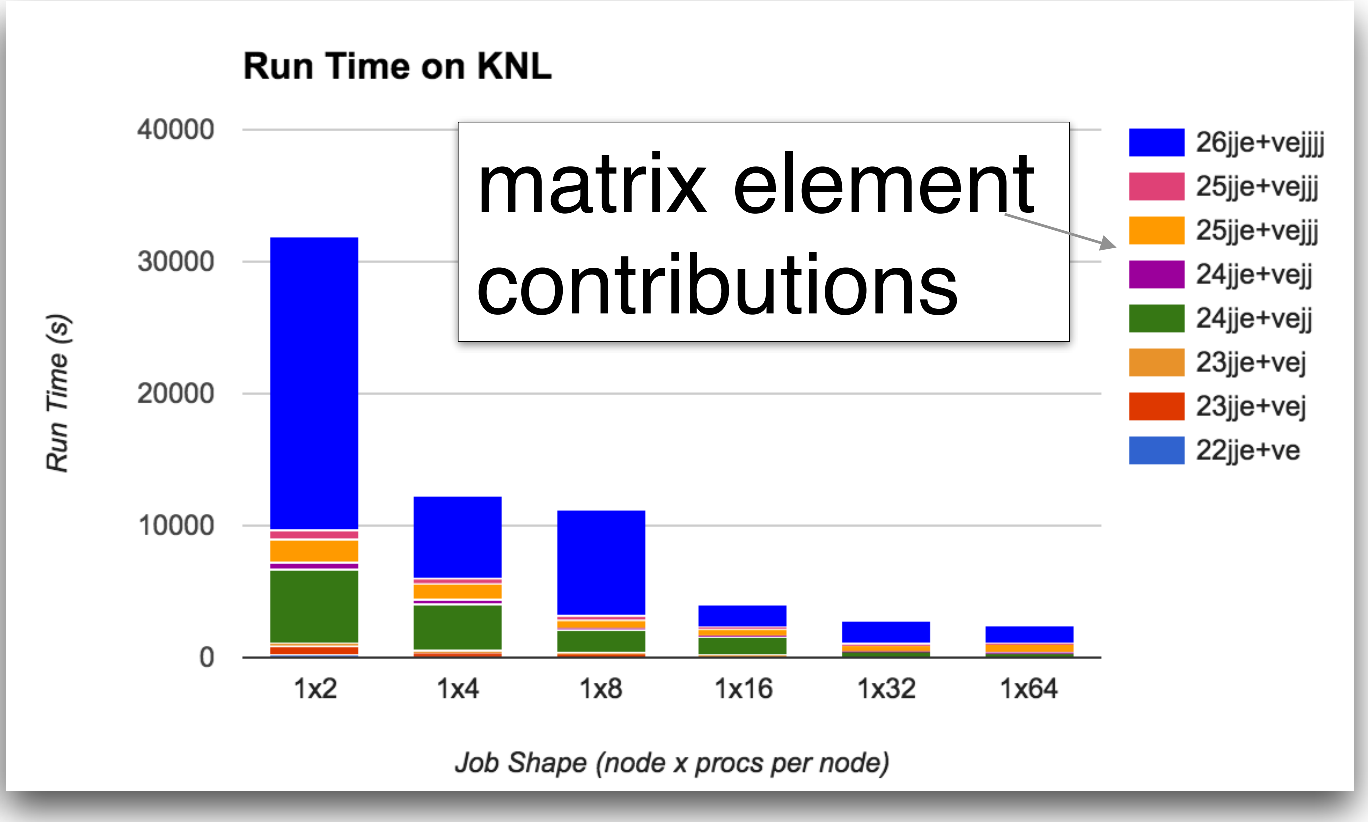

n points per rank
MPI_Allreduce
All Ranks calculating different phase space points
(1)Exchange phase space data
(2)update all rank's grid
(3) check xs precision

Care has to be taken w/ large shared file systems to ensure good performance new many core architecture presents a challenge to the our payloads

Code improvements enable scaling on KNL



9hr run-time



matrix element contributions

New results from two days ago… next test on Mira to see if we see similar improvements

Event Generators used by ATLAS in HPC sites

- Alpgen - Leading Order
  - used extensively in Run 1 less so in Run 2
- Sherpa - NLO
  - widely used in Run 2
- MadGraph5_aMC@NLO
  - Very popular in Run 2
  - Taylor Childers is working with Olivier Mattelaer from to parallelize MadEvent binaries. This involves having the MG5 framework generate MPI wrappers for these FORTRAN binaries
- Event Generators comprise ~20-30% of CPU time on the Grid - anything done on HPC's saves Grid cycles for other uses

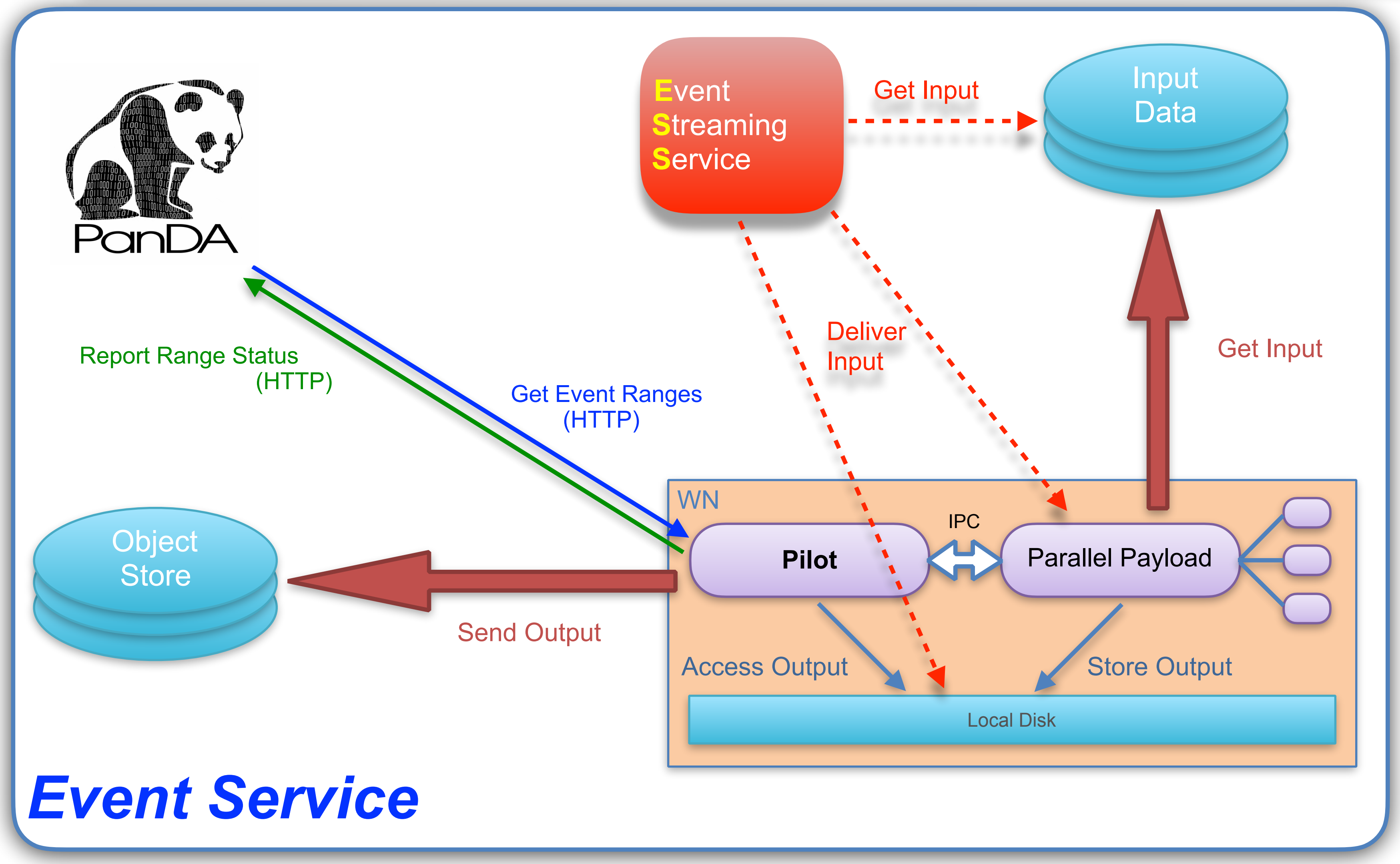
ATLAS MC Event simulation

(Largest CPU usage on grid)

MC Event Simulation

- Input files much smaller than the output files
- GEANT 4 used for Detector simulation
- Massively Parallel (each Grid Node independent of one another)
- Simulation code not designed for checkpointing
- HPC's nodes are designed with fast inter connects for MPI work loads
- HPC file systems are designed for massive parallel writing - used for checkpointing of the computation work loads
- Need to develop a solution to approximate checkpointing and make good use of MPI architecture of the HPC machines.

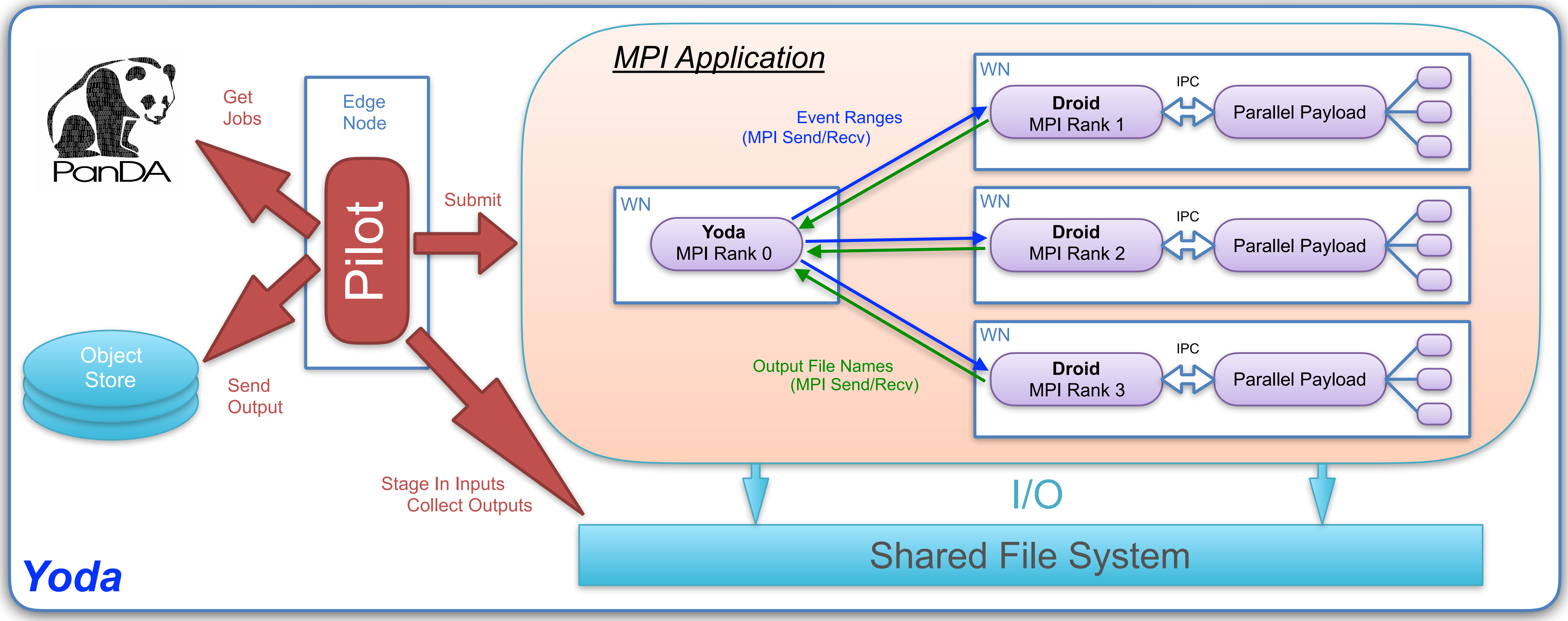
Event Service. Schematic



- Pilot delivers fine-grained workloads to the running payload application in real time
  - ✓ Workload: **Event Ranges**
- Payload application: **process-parallel version of Athena (AthenaMP)**
  - ✓ Serial initialization in the master process
  - ✓ Then fork worker processes
  - ✓ Workers process the events

- Payload directly reads input files for the event data (either local or remote file access)
- Payload uses **Output File Sequencer** for writing intermediate outputs (one per range), which are sent to **Object Stores**
- **Missing Component: Event Streaming Service.** Intelligent asynchronous delivery of the input data to the worker nodes
  - ✓ Presently in early design/prototyping phase

Yoda - Event Service on Supercomputers



- Lightweight versions of the conventional Event Service components
  - ✓ Yoda - mini **JEDI** (**J**ob **E**xecution and **D**efinition **I**nterface)
- Yoda components communicate with each other over MPI
  - ✓ As opposed to the HTTP-based communication implemented in the conventional Event Service

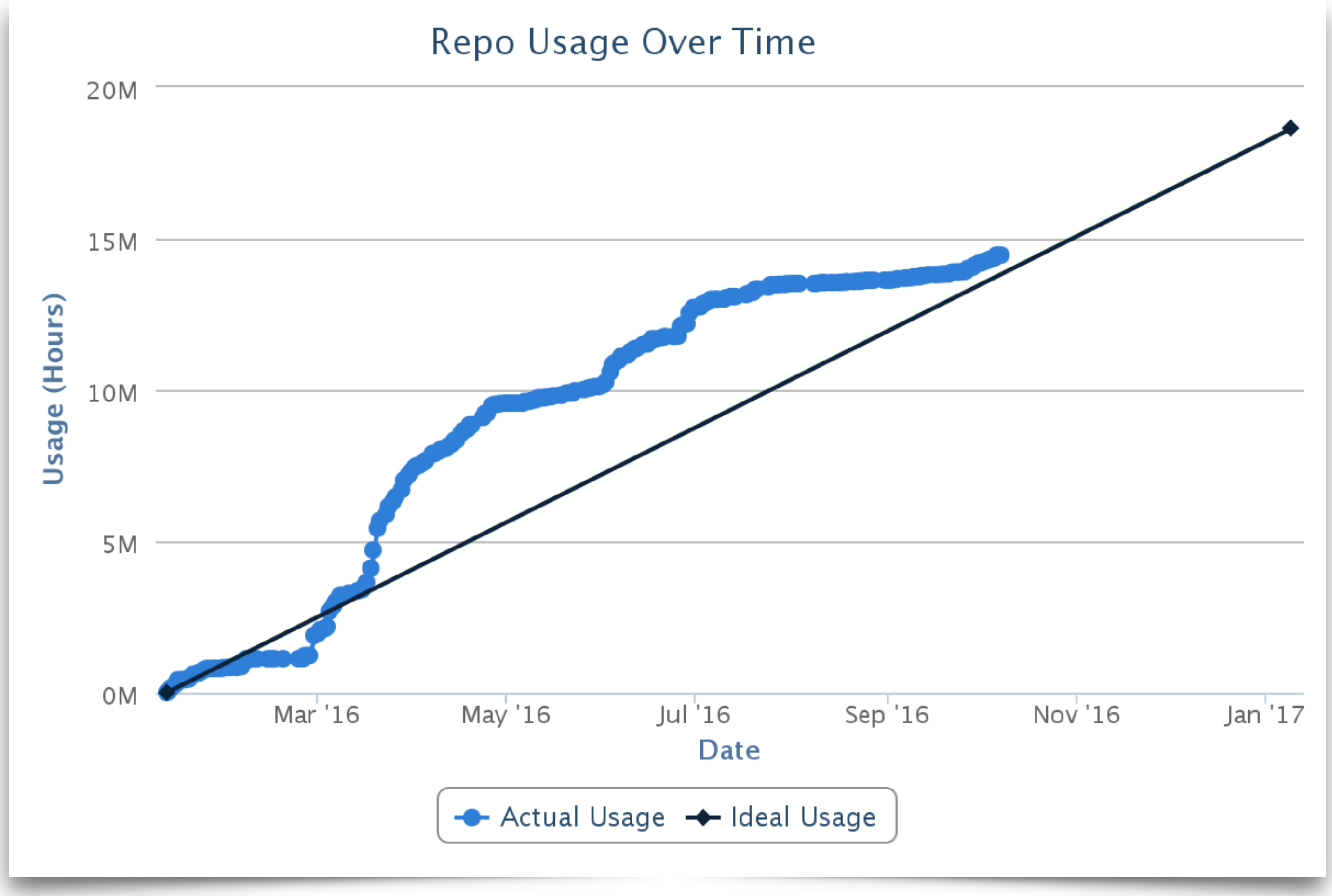
Commissioning and running in production

- First use-case for the Event Service: ATLAS detector simulation with Geant4

- The supercomputers at **NERSC** (**N**ational **E**nergy **R**esearch **S**cientific **C**omputer Center, LBNL, USA) have been the main platform for the commissioning of the Event Service and for running production workloads
  - ✓ Commissioning activity on the Grid is well underway

- Since late 2015 Yoda has been running ATLAS Simulation production on Edison HPC at NERSC
  - ✓ 15M CPU-hours delivered to ATLAS in 2016

NERSC CPU time allocation usage by Yoda in 2016



CPU efficiency

- During the commissioning phase of Yoda we studied various factors which can have a visible effect on the efficient usage of CPU resources of the compute nodes

- Such factors include

  1. Initialization time of the payload application

  2. Sequential running of several instances of the payload application on a compute node during one MPI submission

  3. Handling of fine-grained outputs produced by the payload application

Payload initialization

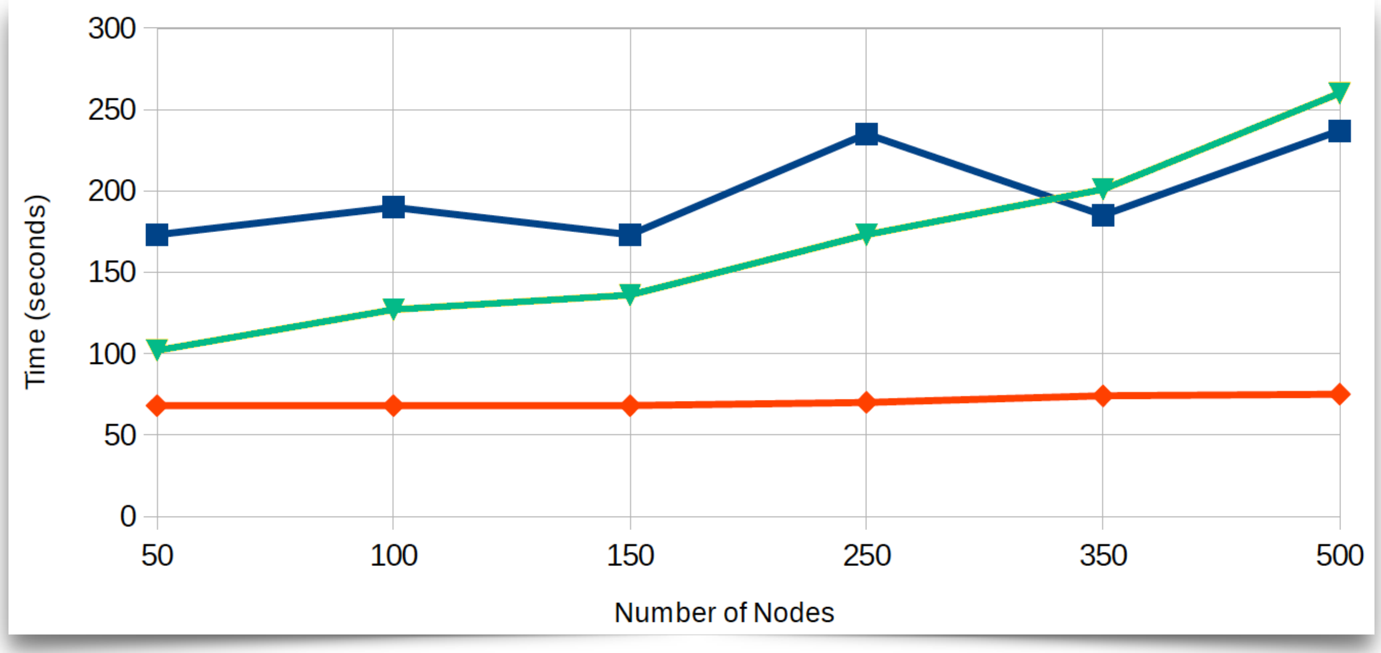
- The Event Service payload (AthenaMP) reads large number of files from the disk during the initialization step

- Concurrent reading of software installation from the HPC shared file system can lead to a serious performance bottleneck when running on many compute nodes simultaneously

- Solution currently used in production: copy software release into the memory of compute nodes

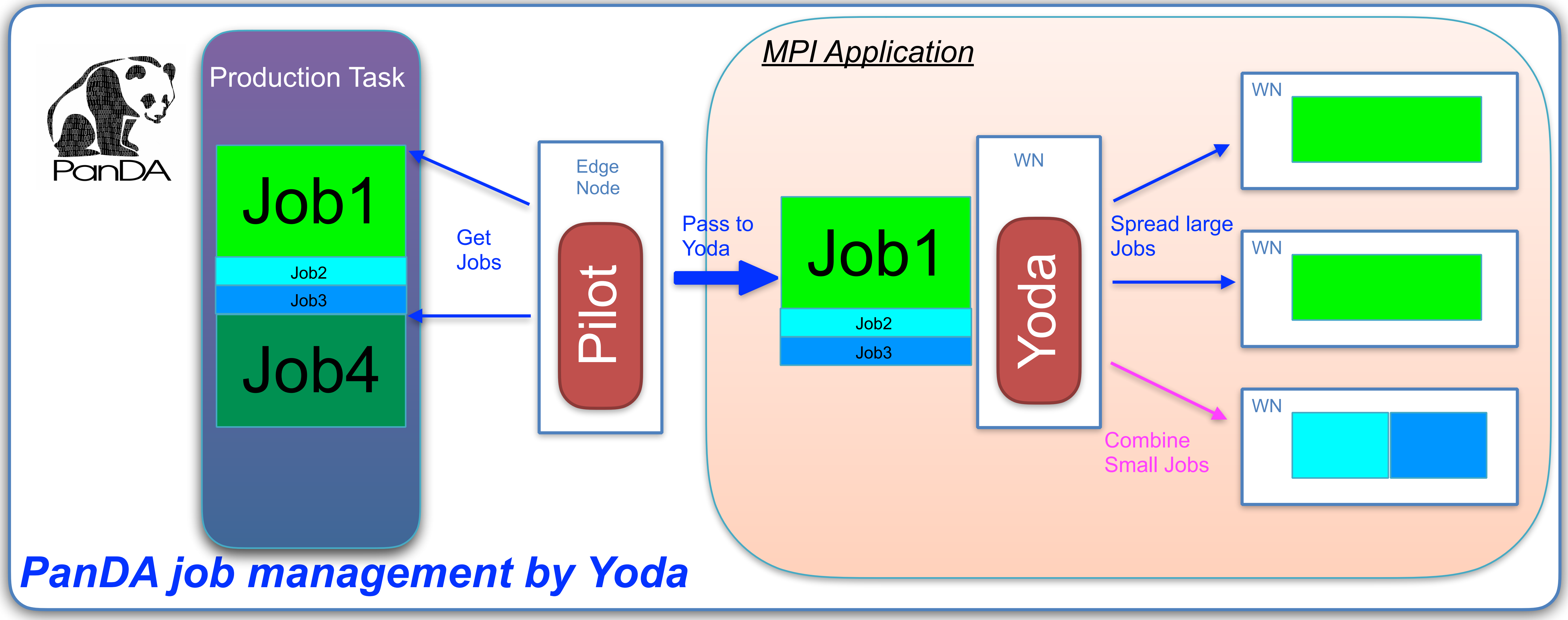
- On Cori Supercomputer at NERSC we also studied the scaling of AthenaMP initialization when installing software releases on different file systems
  - ✓ Lustre
  - ✓ Cori Burst Buffer
    - See the talk by W Bhimji at CHEP2016
  - ✓ Shifter
    - See the talk by L Gerhardt at CHEP2016

AthenaMP initialization time



PanDA jobs vs MPI jobs

- PanDA jobs are building blocks of PanDA production tasks
  - ✓ Thousands of jobs per task
- Yoda combines multiple PanDA jobs into single MPI submission
- If Yoda fails to process all events from some PanDA job during MPI allocation time, then PanDA **generates new job for the leftover events**
  - ✓ Hence different number of events in PanDA jobs in the Event Service tasks

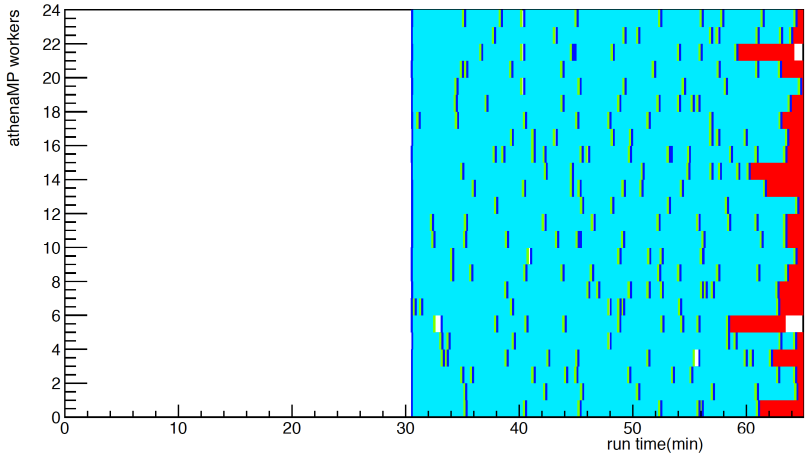


PanDA job management by Yoda

CPU efficiency of HPC compute nodes

- Examples of Yoda compute nodes with different CPU efficiency
  - ✓ **Edison Supercomputer** at NERSC, 24-core nodes

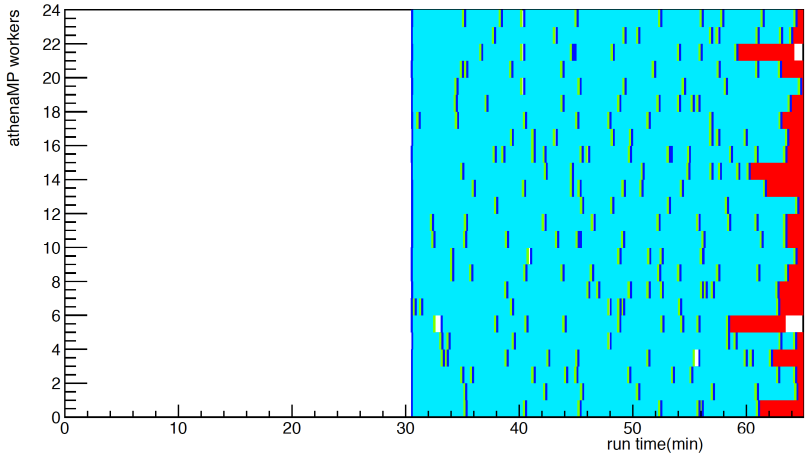
- Example 1. Poor efficiency
  - ✓ One PanDA job …
  - ✓ But very slow initialization



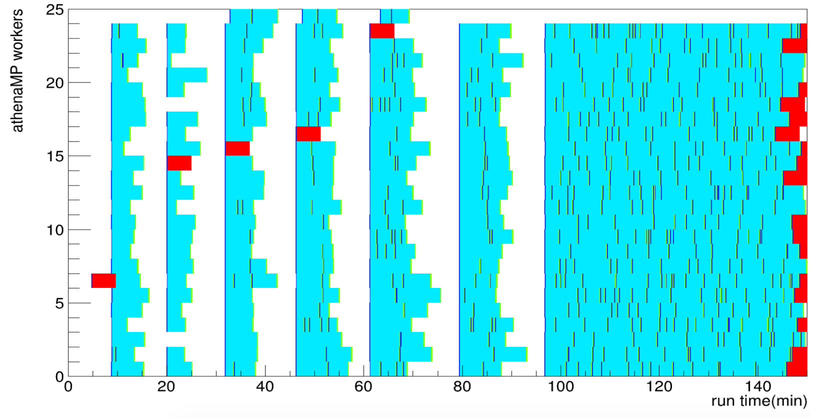
Legend. **White space**: core is idle
Turquoise: core processing an event
Red: event processing started but not finished

CPU efficiency of HPC compute nodes

- Examples of Yoda compute nodes with different CPU efficiency
  - ✓ **Edison Supercomputer** at NERSC, 24-core nodes
- Example 1. Poor efficiency
  - ✓ One PanDA job …
  - ✓ But very slow initialization



- Example 2. Poor efficiency
  - ✓ Several PanDA jobs on one node

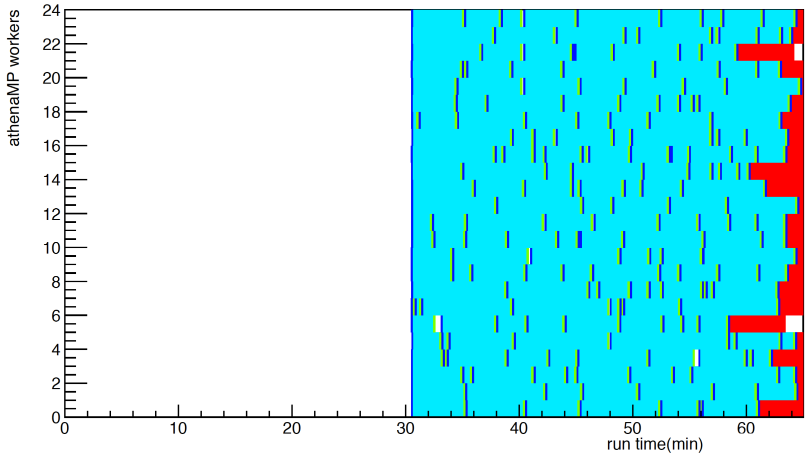


Legend. **White space**: core is idle
Turquoise: core processing an event
Red: event processing started but not finished

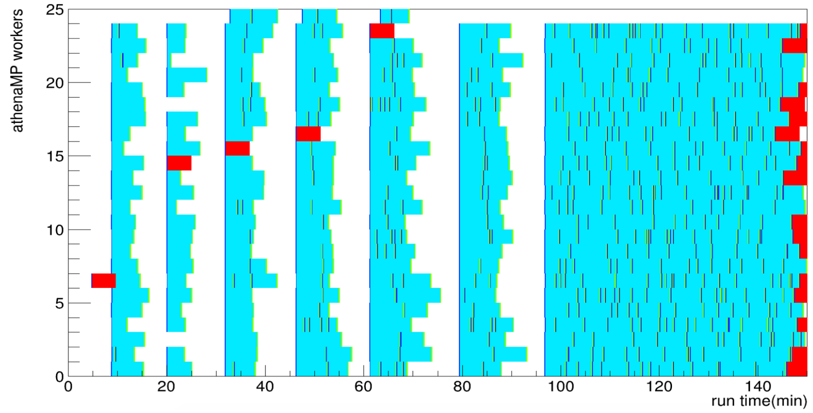
CPU efficiency of HPC compute nodes

- Examples of Yoda compute nodes with different CPU efficiency
  - ✓ **Edison Supercomputer** at NERSC, 24-core nodes

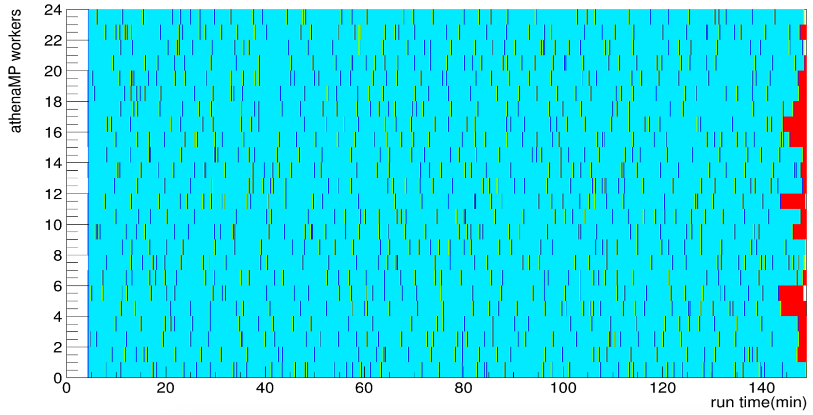
- Example 1. Poor efficiency
  - ✓ One PanDA job …
  - ✓ But very slow initialization



- Example 2. Poor efficiency
  - ✓ Several PanDA jobs on one node



- Example 3. Good efficiency
  - ✓ One PanDA job
  - ✓ Fast initialization



Legend. **White space**: core is idle
Turquoise: core processing an event
Red: event processing started but not finished

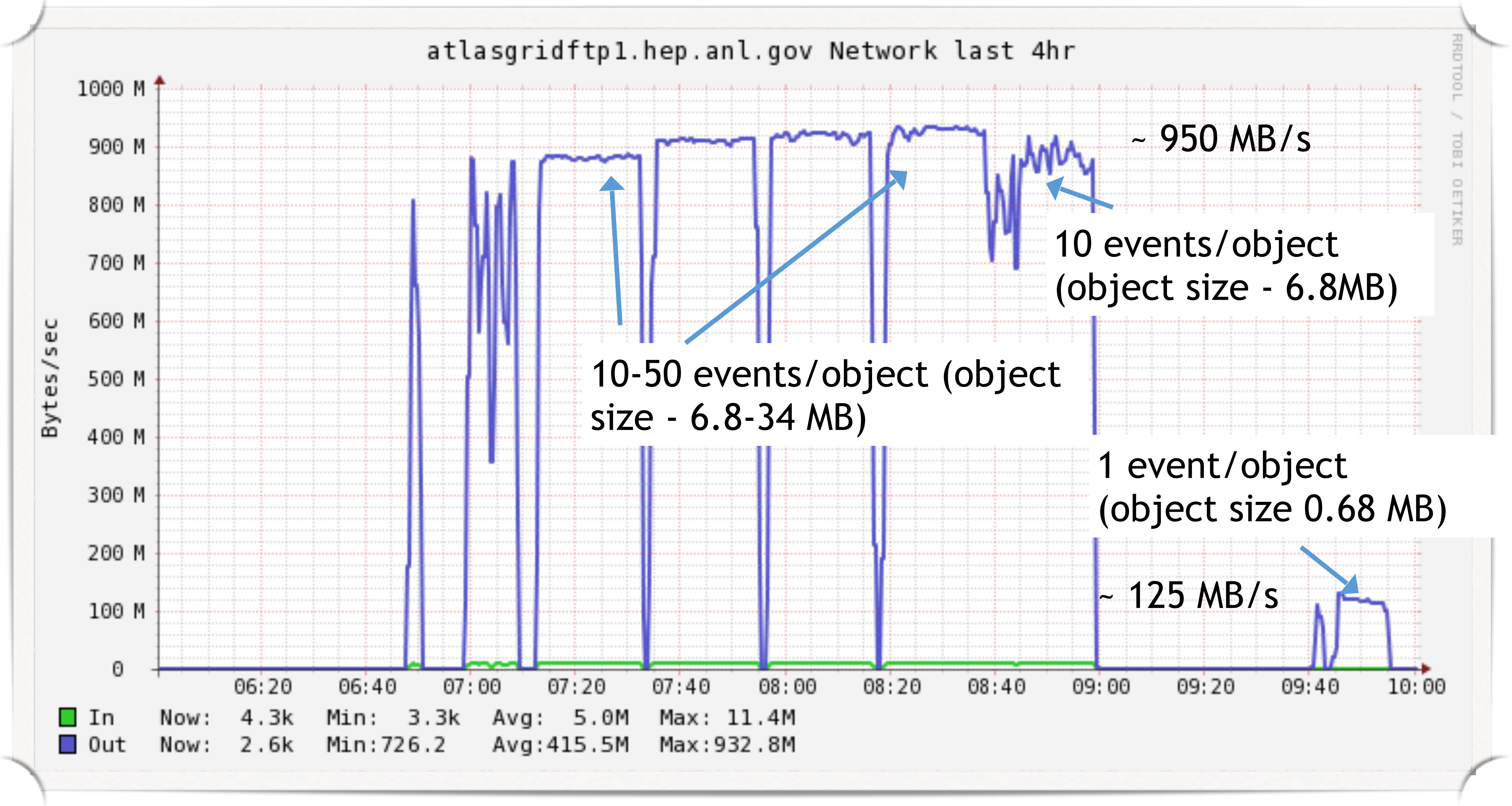
Handling of fine-grained outputs using Object Stores

- The Event Service Payload creates intermediate outputs, which are sent to Object Stores (OS)
  - ✓ Final outputs are produced later by specialized merge jobs
  - ✓ Yoda currently uses OS at BNL

- As part of Yoda commissioning at NERSC we studied the OS performance. Some observations/conclusions:
  - ✓ CEPH OS has no queuing or protection from overloading
  - ✓ When the clients overload CEPH OS various errors can occur
    - Authentication errors
    - Inability to connect to bucket
    - Inability to write object
    - Longer running writes
  - ✓ Client software must have retry and perhaps queuing capabilities. Otherwise we should use a system that can regulate the OS writes

OS Performance. Bandwidth vs Object Size

- Achieved ~7.2GiB/sec writing speed from ANL to BNL



~ 950 MB/s

10 events/object (object size - 6.8MB)

10-50 events/object (object size - 6.8-34 MB)

1 event/object (object size 0.68 MB)

~ 125 MB/s

Lessons learned

- Primary causes of sub-optimal usage of HPC compute nodes by Yoda:
  - ✓ Slow initialization of the payload
  - ✓ Combining multiple PanDA jobs into single MPI submission

- Large number of small transfers can saturate Object Stores
  - ✓ Initially Yoda was sending outputs one at a time directly from the compute nodes
  - ✓ Fixed this by asynchronous sending of pre-merged outputs (tar-balls)

- Prefer few large transfers to the Object Store to many small transfers

- Data stage-out has to be decoupled from the event processing
  - ✓ On HPC use **DTN** (**D**ata **T**ransfer **N**odes) for stage-out

Outlook and Future Work

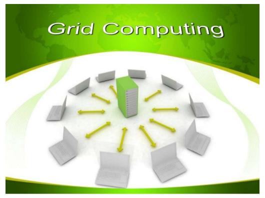
- Avoid fragmentation of PanDA jobs in the Event Service tasks by implementing the new concept of a **Jumbo Job** in PanDA
  - ✓ 1 Jumbo Job = 1 PanDA task

- Implement specialized I/O processes for AthenaMP
  - ✓ **Shared reader:** optimizes data reading on worker nodes, saves memory, also an important step towards the implementation of the Event Streaming Service
  - ✓ **Shared writer:** reduce the number of outputs produced by Event Service payloads

- Design and implement the **Event Streaming Service**

- Extend Event Service functionality to other ATLAS workflows beyond Simulation
  - ✓ Reconstruction, Analysis

- Make Event Service a unified workflow architecture across all ATLAS computing platforms



Grid

Clouds

HPC

Volunteer Computing
ATLAS@Home

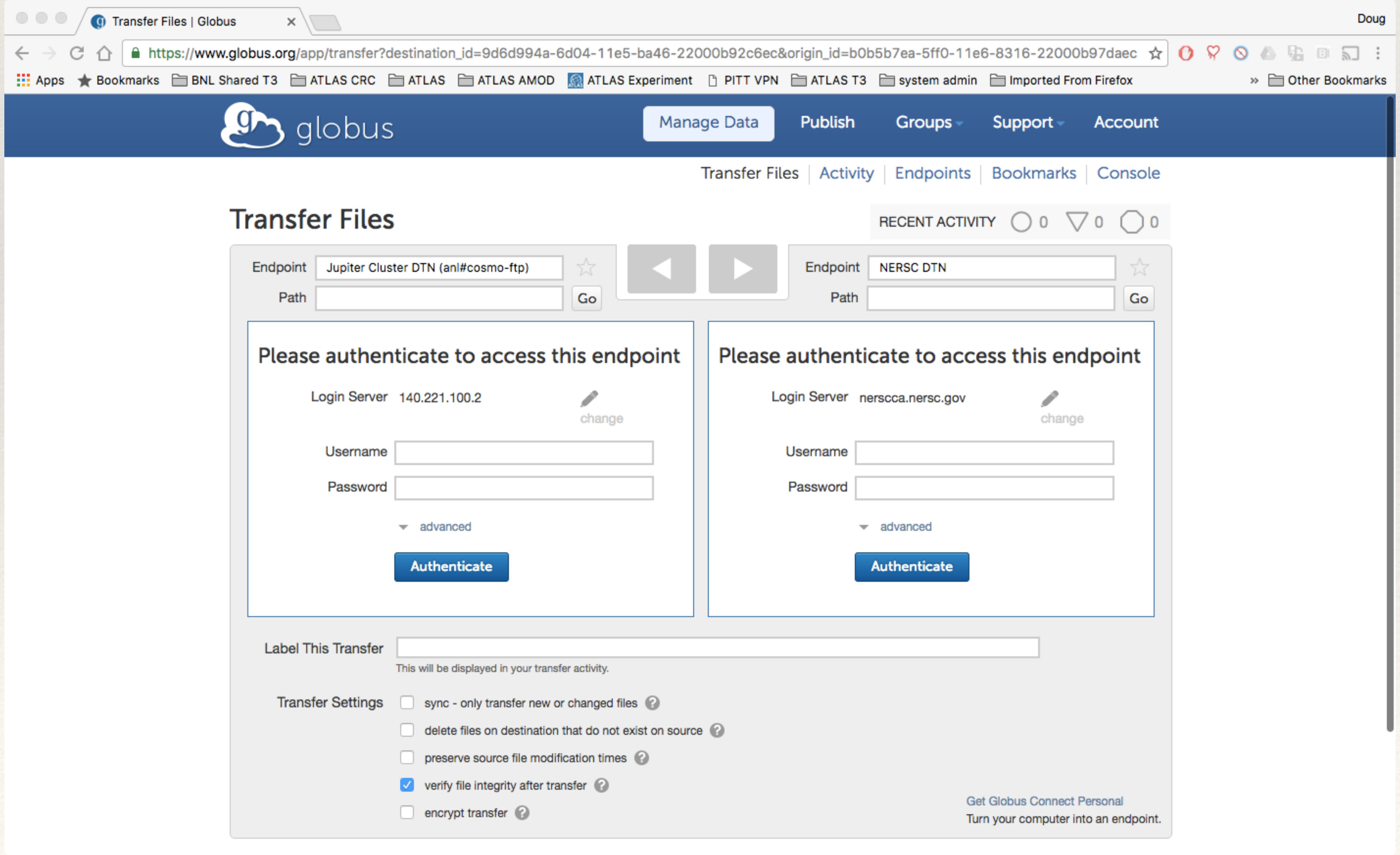
How Does XRootD fit into all of the this?

XRootD on the DOE HPC machines

- Not likely going to be allowed
- HPC compute nodes typically do not have TCP/IP code stack and are not connected to WAN
    - NERSC is an exception - Edison has some outbound connectivity not very performant.
- HPC storage system High performance parallel file system by design (typically Lustre or GPFS)
- Storage (except for tape) is short term - for the life of the CPU allocation. Disk are not for archival - think storage Cache
- Data Transfer Nodes (DTN's) are used for WAN data traffic. They see the shared file systems

WAN data transfers at DOE HPC's

- Data transfer nodes - are to be used for WAN transfers of data (into and out of the HPC machines)
    - These have gridftp servers running on them.
- What about third party managed transfers?



Transfer Files

Endpoint	Jupiter Cluster DTN (anl#cosmo-ftp)
Path	

Please authenticate to access this endpoint

Login Server	140.221.100.2
Username	
Password	

Endpoint	NERSC DTN
Path	

Please authenticate to access this endpoint

Login Server	nerscca.nersc.gov
Username	
Password	

Label This Transfer

This will be displayed in your transfer activity.

Transfer Settings

- □ sync - only transfer new or changed files
- □ delete files on destination that do not exist on source
- □ preserve source file modification times
- ☑ verify file integrity after transfer
- □ encrypt transfer

Get Globus Connect Personal

Turn your computer into an endpoint.

Yes - Globus Connect

Is there any way for XRootD to be used at the large HPC sites?

- Yes…
- Use Case - Have the data transfer nodes act as XRootD data servers
  - Pro's :
    - Efficient protocol for wide area transfers
    - used by LHC collaborations and ultimately LSST
  - Con's :
    - limited user base. More people use gridftp than XRootD.
    - Authentication.
    - HPC centers have limited experience with XRootD (too niche)
  - But ….

Issues to Solve

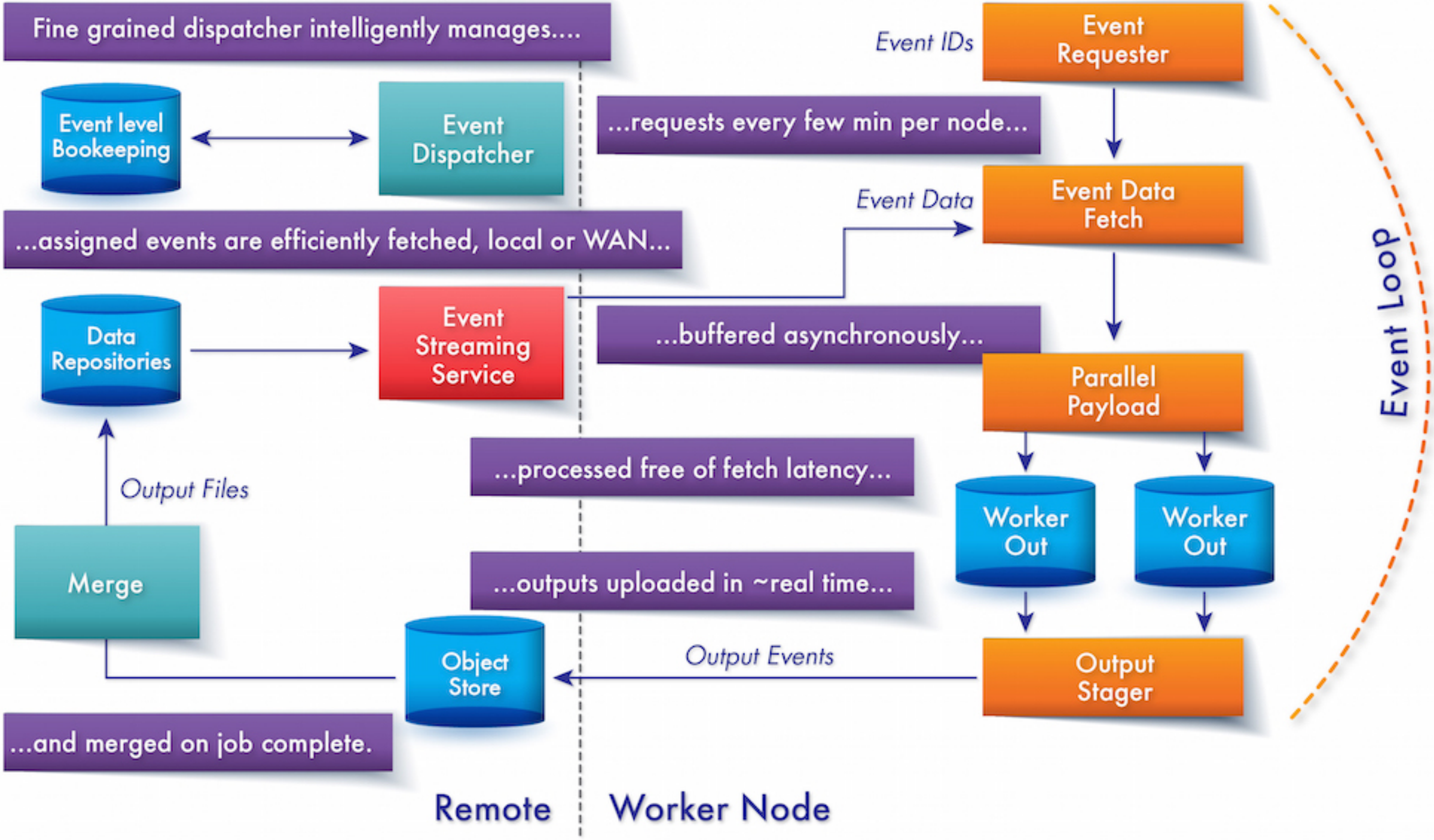
- Size of User base
  - hard one to solve.. most users of XRootD are not HPC users. Some exceptions (LHC)
- XRootD knowledge at HPC centers.
  - Possible solution - use friendly centers - NERSC (very user forward policies) and OLCF. (ALICE important user of OLCF)
- Authentication
  - no VOMS extensions on grid credentials
  - ALCF and OLCF only recognize their own CA
    - NERSC - allows CERN CA, ALCF and OLCF do not
  - Two factor authentication - On time passwords at ALCF and OLCF, NERSC is likely going to two factor authentication
  - Federated identity - already solved by Globus. HPC's accept Globus

Conclusions

- Over the past few years ATLAS has begun to use the largest HPC's in the US (and the world)
  - These machines come with unique challenges.
  - ATLAS codes and work flows have had to change
  - things are looking good
- There is a place for XRootD in the HPC centers
  - We the XRootD community have to convince the HPC operators this technology is worth it to them
  - Need to solve some of the issues outlined previously
  - Likely need to collaborate with Globus (especially on federated identity)

Backup

Event Service. Workflow



Fine grained dispatcher intelligently manages....

...requests every few min per node...

...assigned events are efficiently fetched, local or WAN...

...buffered asynchronously...

...processed free of fetch latency...

...outputs uploaded in ~real time...

...and merged on job complete.