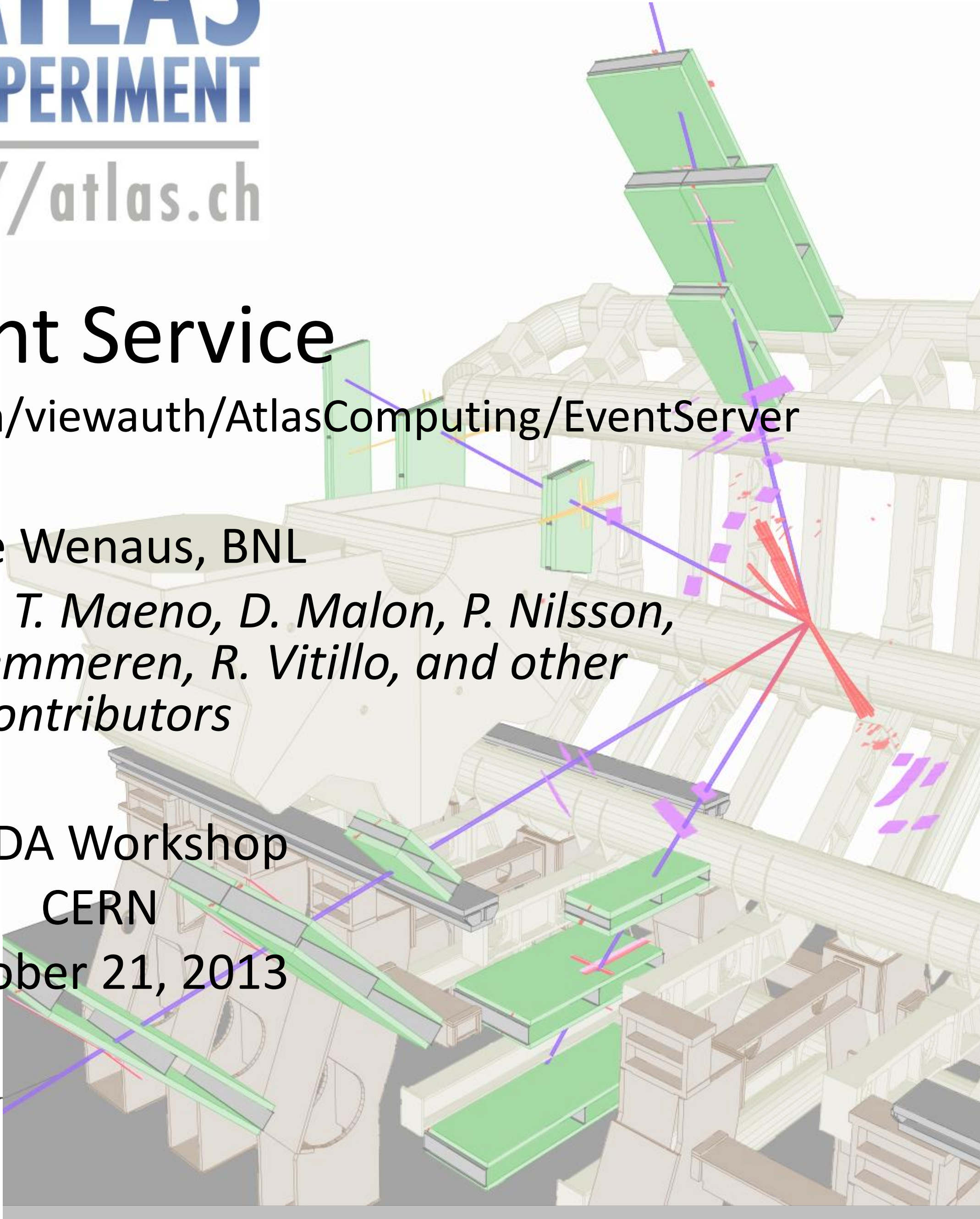


Event Service

https://twiki.cern.ch/twiki/bin/viewauth/AtlasComputing/EventServer

Torre Wenaus, BNL

For P. Calafiura, K. De, T. Maeno, D. Malon, P. Nilsson, V. Tsulaia, P. Van Gemmeren, R. Vitillo, and other contributors

PanDA Workshop

CERN

October 21, 2013

What’s an event service?

- On the model of… ask for exactly what you need, have it delivered to you by a service that knows how to get it to you efficiently
- Why ask for files when what you really want are (particular) events
- (Perhaps the events you’re asking for don’t even exist in the required form, and they’ll be transparently created for you – that’s another discussion… virtual data… but on the same path)

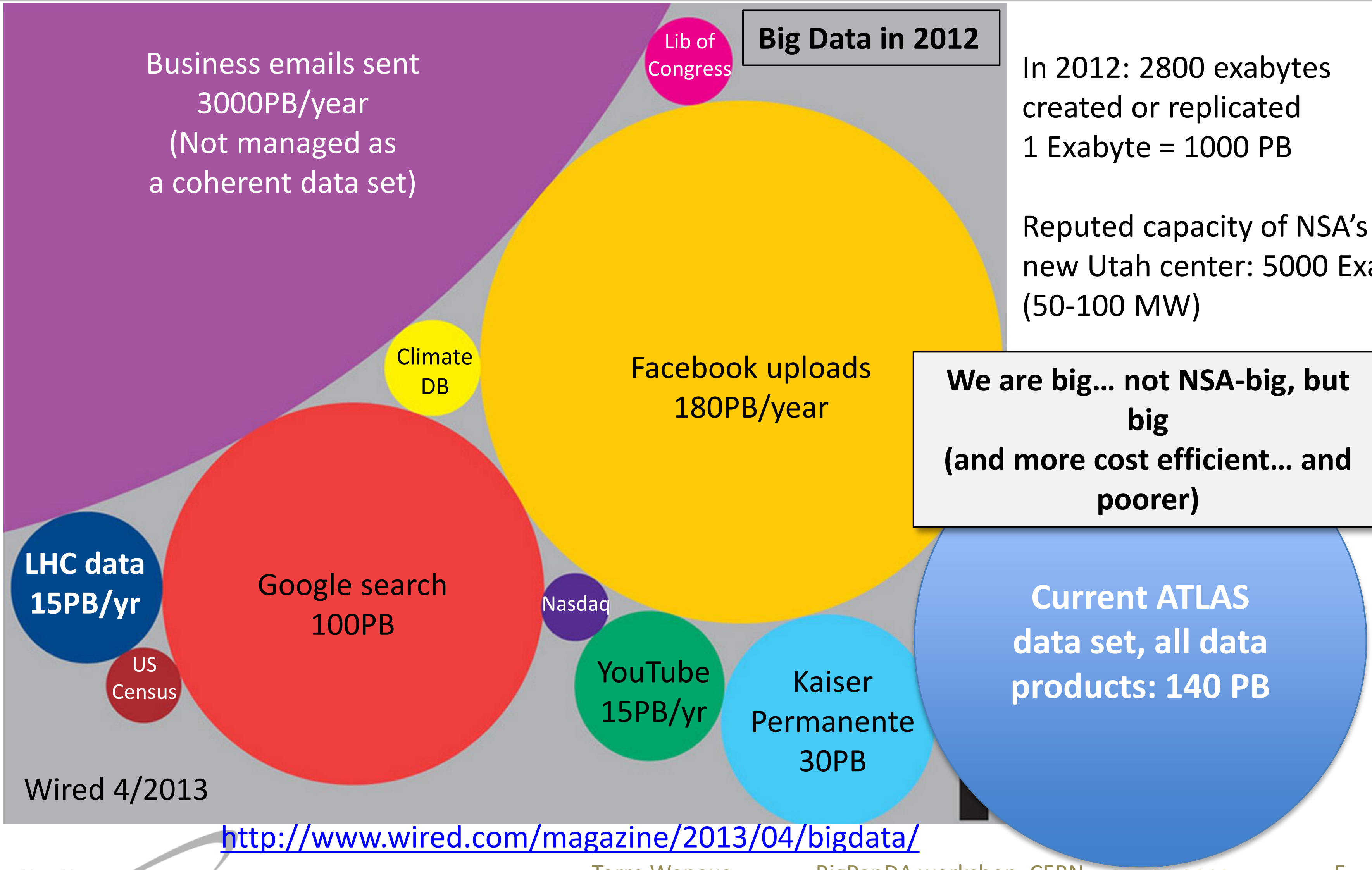
Why an event service? Do for data what PanDA did for processing

- PanDA shows that intelligent automated dispatch of jobs to intelligent clients works very well
  - Makes processing self-correcting – you use the resources that work
- The biggest operational load is in data management
  - We are hit hard by storage resources that don't work, or are full
  - DDM is complex
- PD2P showed that dynamically distributing only the data that is needed can improve resource usage efficiency – use the network dynamically rather than using storage statically
  - **In general it's much cheaper to transport data than to store it**
- The event service is a further step
  - In making the client agnostic to where the data is – transparently use the data sources that work – to reduce the operational burden and user impact of storage problems
  - In making full use of the network to deliver just the data needed from an optimized (and minimal) set of sources
- We have a distributed processing infrastructure very well suited to give it a try
  - PanDA brokers and dispatches jobs in an (increasingly) intelligent way
  - Let's use the same infrastructure to deliver events
  - Make resilient data access an integral, automated part of the processing; no DDM in the production workflow and no precondition of data pre-placement
- We have a software infrastructure ready to support it

Why an event service? Opportunistic resources to expand computing throughput

- Opportunistic use of HLT after LS1
- Opportunistically soaking up cycles on supercomputers
- Free research clouds, commercial clouds
  - More/cheaper resources available if we can be highly opportunistic
    - e.g. Amazon spot market – cheap, even free under 1 hour
- ‘ATLAS@Home’ via BOINC on tens of thousands of computers in China?
- Common characteristic to using opportunistic resources: we have to be agile
  - Quick start (rapid setup of software, data and workloads) when they appear
  - Quick exit when they’re about to disappear
  - Robust against their disappearing with no notice : minimize losses
  - Use them until they disappear – soak up unused cycles
  - Fill them with fine grained workloads
    - Send a steady stream of events, and return outputs in a steady stream
      - No heavy data prestaging, and a hasty exit loses very little
- **Event service is a means of enabling agile, efficient use of opportunistic resources**

Why an event service?
Efficient use of storage is essential for us



In 2012: 2800 exabytes created or replicated
1 Exabyte = 1000 PB

Reputed capacity of NSA's new Utah center: 5000 ExaB (50-100 MW)

We are big… not NSA-big, but big (and more cost efficient… and poorer)

http://www.wired.com/magazine/2013/04/bigdata/

Other event service advantages

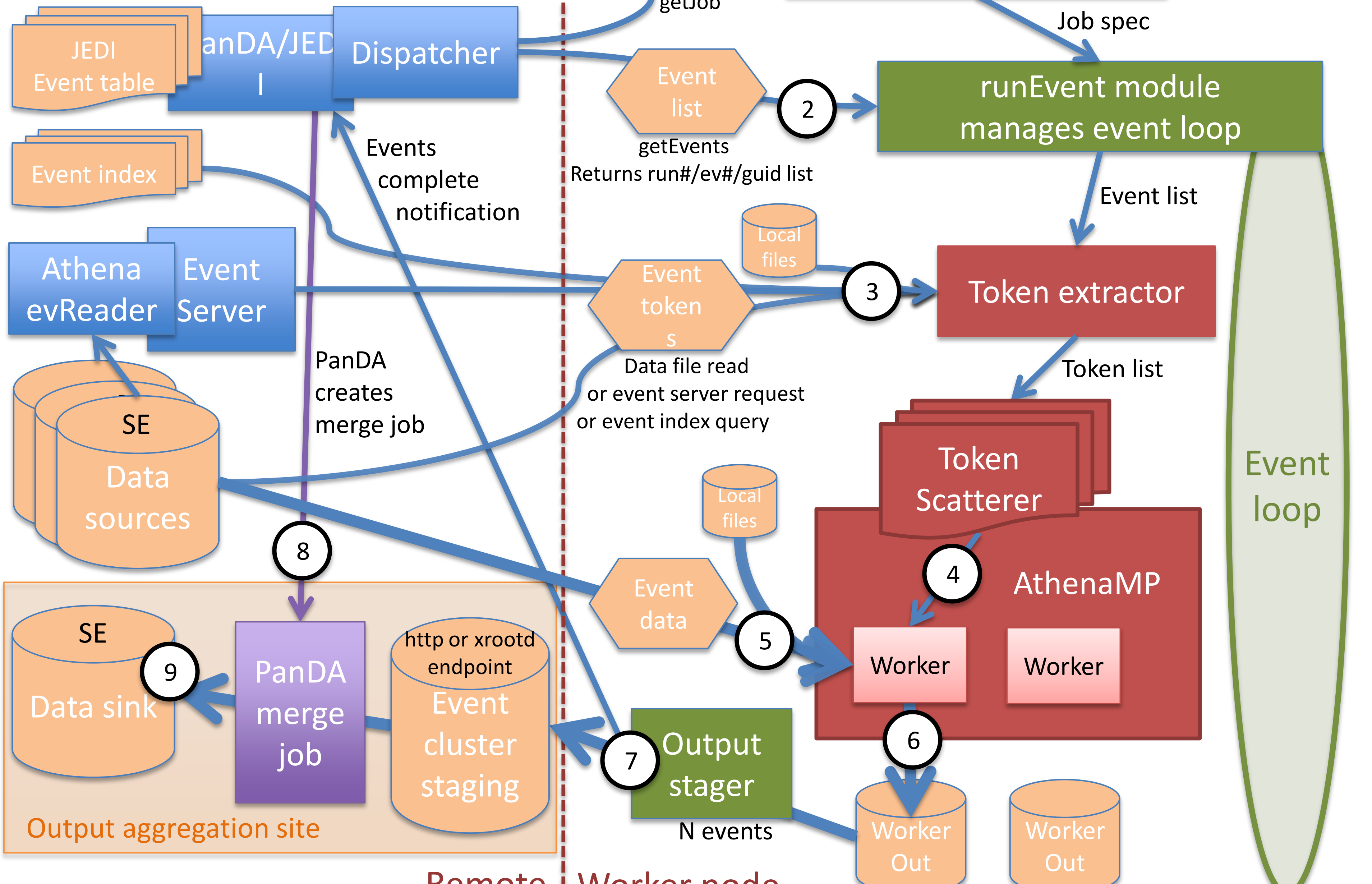
- Avoid idle cores in athenaMP processing – if parallel threads process events in large chunks (files), cores can sit idle while the slowest one completes
- DDM simplification – no DDM involvement on input or output
- Output merging flexible and simple: file size is tunable, aggregation of outputs to merge site proceeds concurrently with processing, JEDI knows when to trigger merge
- A crash doesn't lose the job, only a few events which will be re-dispatched

Required for an event service

- Managing workflows at the event level the way PanDA has been managing them at the file level, with the necessary event-level bookkeeping
  - *Check* – JEDI now provides this
- Excellent networking; *check*
- Workloads with high CPU/IO; *check* – simu
- Network-efficient event data access; *check* – the event I/O optimizations of recent years makes remote event streaming possible
  - *Preferably buffered by asynchronous caching*
  - Just as JEDI is an enabler for the event server scheme on the grid side, athenaMP and associated I/O developments and optimizations are enablers on the core sw side
    - Queueing and streaming events to be consumed by workers
    - I/O optimizations making event reading over WAN practical
    - Asynchronous pre-fetch to remove network latencies from processing workflow

Event Service schematic V4

Oct 2013



ATLAS Event Service Workflow

Pilot requests a job from PanDA

PanDA sends the pilot an event mode job

Event loop

Pilot enters event service mode and requests event list

PanDA selects and dispatches event(s), updating its event bookkeeping

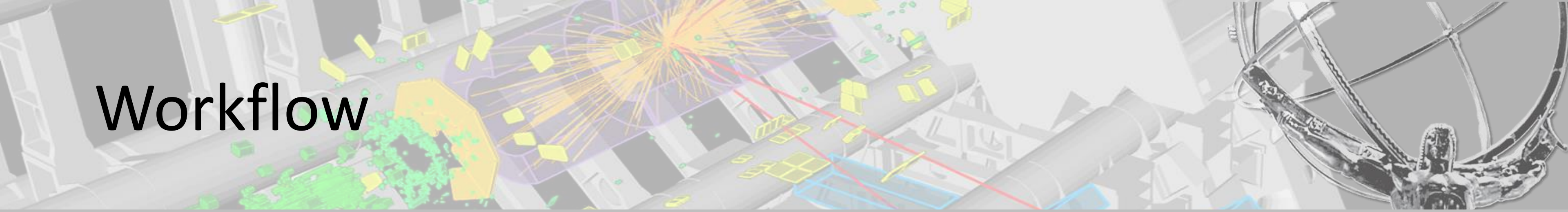
Pilot forwards event list to AthenaMP, GUID resolved to PFN in PFC

AthenaMP payload receives event list, translates to event header tokens (specifying how to read event), distributes tokens to workers

Workers read event data, process, and write 1- or few-event files

Pilot monitors outputs, uploads new files to aggregation site, informs PanDA

PanDA detects when a job's events are done and uploaded, and initiates a merge



Workflow

- Pilot makes a getJob request to PanDA
- PanDA determines event service type job is appropriate, selects and dispatches one
- Pilot’s runEvent module invoked to manage event processing
  - Determines input files, makes POOL catalog file
  - Triggers athenaMP launch, configuration
  - Enters event processing loop
    - Requests event list from PanDA dispatcher
    - Receives list, GUID, retry number; PanDA marks events as in process
    - Event list passed to token extractor
      - Obtains tokens via direct data file read, event server or event index
    - Token list sent to athenaMP TokenScatterer for distribution to workers
    - Workers read event data from source and process events
    - Each worker writes its own output file
      - Single-event files initially; appropriate for simu with long event processing time

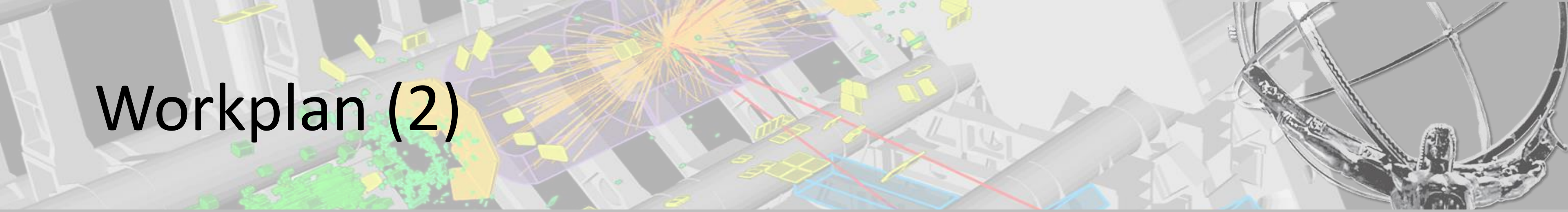
Workflow (2)

- Outputs managed in a near real time, granular way, just as inputs are
  - Output aggregation and monitoring concurrent with task processing
  - Minimal losses in case of sudden eviction from or disappearance of resource
- Output stager monitors output directory, detects completed output files
  - Transfers them to aggregation point using lightweight service (http, xroot) or just a copy if local
  - Informs PanDA of event completion; PanDA updates event table
  - Cleans up output files
- PanDA monitors event completion in the event table
- When event processing for a job is complete, PanDA merge job is generated to merge output files, store merged outputs on SE and register in DDM
- If processing of an event cluster by a consumer times out, PanDA invalidates that consumer by incrementing the retry number for those events and reassigning them to another consumer
  - If original consumer does finally return results, they will be ignored; their retry number is no longer the latest

Workplan as of Aug (with updates)

(done) = first prototype in place, only limited testing done

- Support Geant4 simulation (AtlasG4_tf) for event service jobs (Peter, Vakho) **(done)**
  - The prime candidate for the opportunistic resources that have the most to gain
  - Requires single-event EVGEN file to startup the payload in initial version
- Support closing/opening multiple output files per job, with associated metadata management (for simu in the first instance) (Peter) **(done)**
- Settle the dispatcher communication protocols for specifying a job as event type and sending an event cluster in response to a request from runEvent (Tadashi, Paul) **(done)**
- Set up POOL catalog file creation for event type jobs (Paul) **(done)**
- Implement token extractor and translation of event list received from pilot into token list delivered to Scatterer. Tools are mostly there to create the token extractor, in the form of tag collection utilities to extract tokens out of files. (Vakho, Peter) **(done)**
- Set up the invocation and execution of the athenaMP payload initialization step (Vakho, Paul) **(in progress)**
- Set up the athenaMP workflow in the event loop from receipt of token list, to WN event processing using the pilot-generated POOL file catalog, to output generation in dedicated output directory (Vakho, Peter) **(done)**
- Check WAN performance on POOL event read. Can we use TTreeCache asynchronous pre-fetch? **(Ilija reports TTreeCache asynch prefetch is working in his lastest tests)**
- Look into how to handle IOV metadata. (Peter) **(prototype under test)**



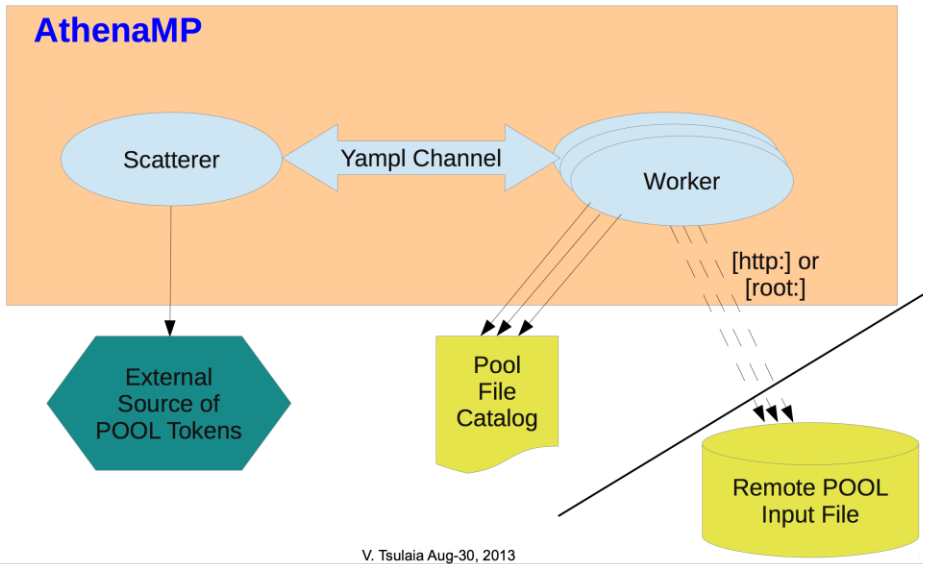
Workplan (2)

- Implement the communication from pilot to PanDA/JEDI to inform PanDA of completed events: status of completion and transmission to aggregation point, retry number (Tadashi, Paul) **(in progress)**
- Implement the output monitor to detect completed output files, invoke procedure to send them to aggregation point + inform PanDA/JEDI + and clean them up (Paul) **(in progress)**
- Implement prototype output aggregation point (local and/or remote/http/xrootd implementation)
  - No effort on this. Ideal item for someone new to come in and help with… particularly if they're interested in trying out an object store…
- Implement PanDA merge job to build cluster files from aggregation area into full output file for registration with DDM (Tadashi) **(done, same as prod/analysis workflow)**
- Implement PanDA/JEDI service that detects event job completion (from accumulation of event completion metadata) and triggers PanDA merge job, and times out/reassigns events held by non-responding consumers (Tadashi)
- Establish testing setup. NB runEvent can be used outside the context of the full pilot, for testing purposes
- Establish Hammercloud based testing setup for full infrastructure. (all)

Event reading and processing status

- Event reader and event server web service prototypes exist
- First reader implementation was with bytestream events, now extended to POOL files
  - Targeting G4 simulation as first use case
- AthenaMP workers receive POOL event tokens from the TokenScatterer, ROOT retrieves event data
  - First version exists, relies on POOL file catalog for file/protocol info
- Successfully tested with sample AtlasG4_tf job
  - Currently needs 1-event input file for configuration
  - Remote http read works, outputs are identical to local read

V. Tsulaia, R. Vitillo, P. Van Gemmeren



OutputFileSequencerSvc

- New component of Athena I/O infrastructure
  - Allows closing output files during the runtime of a job
    - Writes events and metadata processed so far
  - Opens new file for the remaining events (and metadata)
  - Creating a sequence of Output Files
    - Naming similar to RAW
  - Transition controlled via incident
    - Configurable

- Example: Splitting Output File on Input File transition

```
from AthenaCommon.AppMgr import ServiceMgr as svcMgr

from AthenaServices.AthenaServicesConf import
      OutputStreamSequencerSvc

outputStreamSequencerSvc = OutputStreamSequencerSvc()

outputStreamSequencerSvc.SequenceIncidentName =
            "EndFileIncident"

svcMgr += outputStreamSequencerSvc
```

Peter Van Gemmeren

Metadata

- Event data handling is rather straight forward.
  - Events never span file boundaries, store is cleared after execute
- In-File Metadata is not so simple.
  - Accumulated/summarized over the run time of the job
  - Or propagated from Input to Output File
    - Done by metadata tools, which a invoked by Begin/End File incidents
      - Not executed for every event
  - Currently written at finalize only, store is cleared at the end of the job
    - Unfortunately, not everyone plays nicely and some modules hold on to pointers to store (which are obsolete once the store gets cleared).
      - Fixed a few of them
  - Limit Output file transition to Input file transition (by policy)
    - No splitting of input file metadata records.
    - MetaDataTools can prepare for possible transition during EndFile incident

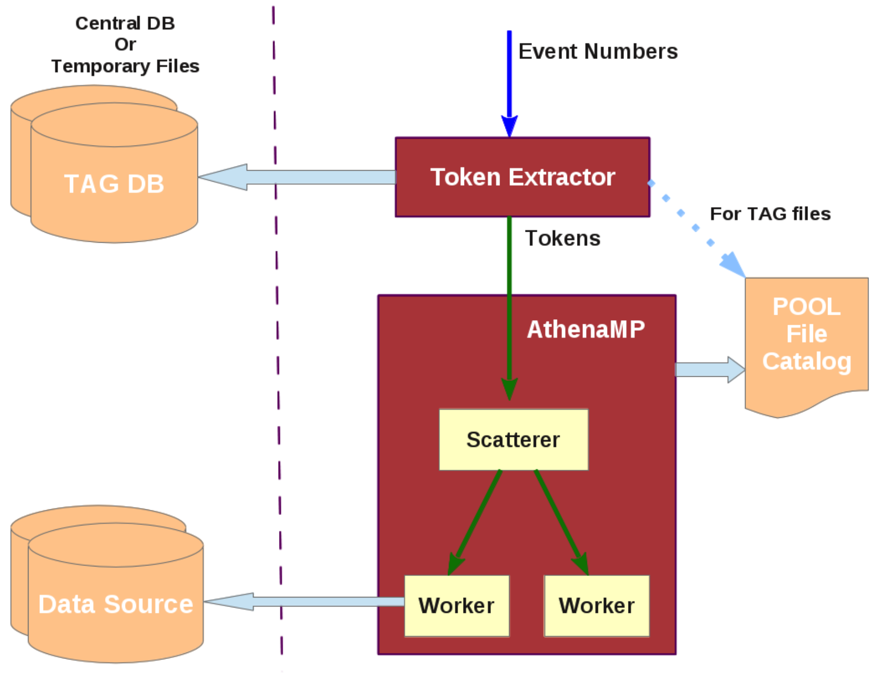
Peter Van Gemmeren

Output File Sequence for EventServer Simulation jobs

- Simulation only needs: IOVMetaData and EventStreamInfo
  - No Lumi or EventBookkeeper
- EventServer will write ‘transient’ 1 event output files, but merge them before use.
  - Can use EndEvent incident to trigger Output File Transition
    - Provide override for Input File Transition requirement
  - Use **`athena.py runargs.MergeHIT.py PATJobTransforms/skeleton.MergeHIT.py`** to merge Output Files
    - Please do slow merge
- First Prototype ~kind off works
  - Doesn’t crash, that was progress
  - Produces metadata (merging adds unneeded metadata due to over configuration)
  - Produces extra file, but that is ‘merged away’
  - ERROR logs from CutFlowSvc, but EventBookkeeper is not used…

Peter Van Gemmeren

Token Extractor



- Prototype implemented, derived from existing event collection utilities
- Turned the existing CollListToken utility into a TokenExtractor
- Receives a list of event numbers over incoming Yampl channel, queries the database (or, as at present for EVGEN, reads from file) in order to convert event numbers to event tokens and sends the tokens over outgoing Yampl channel
- Connected it to prototype AthenaMP Token Scatterer and successfully ran some tests

Vakho Tsulaia

Event Service Support in the Pilot

- Pilot runs as normally, downloads a job definition from the dispatcher
  - For a ***normal job***, pilot will fork and monitor the **`runJob`** subprocess
  - For an ***event job***, pilot will fork and monitor the new **`runEvent`** subprocess, currently in development
- **`runEvent`** will be selected by the pilot when it sees the presence of an ***`eventRangeID`*** field in the job definition
- For an event job, the dispatcher response(s) will contain
  - *`eventRangeID`*: <string>
  - *`startEvent`*: <string>
  - *`lastEvent`*: <string>
  - *`lfn`*: <list>
  - *`guid`*: <list>
  - *`attemptNr`*: <string>

Paul Nilsson

Pilot runEvent subprocess vs EventService module

- Basic functionality of runEvent:
  - Main loop running over all events within the given event range; download event(s) from Event Server and process them; download new event range until all ranges are exhausted (as determined by dispatcher)
  - An asynchronous thread is staging out finished output files
- EventService class also in development
  - Contains all the methods for interacting with the Event Server; download events, event processing, any server/dispatcher updates, etc
  - Same plug-in mechanism as other classes, i.e. an experiment can have its own subclass in case there are any differences with the default (read: ATLAS) implementation

Paul Nilsson

Pilot runEvent workflow

- [Initial event range already downloaded with the first getJob() call by the pilot]
- Stage-in (unless already present in CVMFS) of necessary preliminaries (DBRelease)
- Configuration and launch of AthenaMP (or other payload)
  - AthenaMP goes through its initialization phase and forks worker processes
- runEvent will enter the event processing loop when the AthenaMP payload is ready (next slide)
- Stage-out files as they become available
  - Remove file immediately if transfer is successful

Paul Nilsson

Pilot event processing loop

- Request an event list from the dispatcher (if initial event range has been processed)
  - Size of event list can be optimized
  - Attempt number will be recorded in the event table for these events
- Determine the physical file replica(s) to be used as the source for input event data, based on the LFN and GUID lists in the event list
  - PFC is created
- Execute payload using event list and PFC
- Dispatcher is updated on a regular basis (heartbeat)
  - `eventRangeID`: <string>, `status`: <string>, `attemptNr`: <string>, `xml`: <string>, where `status` can be 'running','finished','failed'

Paul Nilsson

Panda Server/JEDI updates for Event Service 1/2

- Two functions for the server ↔ pilot communication available as extension of dispatcher's API
  - getEventRanges(PandaID)
    - To get a list of {eventRangeID, startEvent, lastEvent, attemptNr, LFN, GUID}
      - eventRangeID : unique identifier in the event table
      - attemptNr : incremented when the range is retried
  - updateEventRange(eventRangeID, attemptNr, status, xml)
    - To update status of an event range
      - status : ready → sent → running → finished/failed
      - eventRangeID + attemptNr : to update the latest attempt
      - xml : contains LFN and SURL (or PFN) of unmerged files
- If unmerged files are not registered to DDM, a new column for SURL/PFN would be required in the JEDI file or event table

Tadashi Maeno

Panda Server/JEDI updates 2/2

- Merging is being implemented as a part of normal production or analysis workflow
  - Use the same function for Event Service as well
  - Functions
    - Merge job generation
    - Creation and deletion of intermediate datasets
    - Job chaining from unmerge jobs to merge jobs
    - Killing unmerge jobs when merge jobs permanently fail
- Possible sequence
  - getJob $\rightarrow$
    getEventRanges $\rightarrow$ updateEventRange $\times$ N $\rightarrow$
    getEventRanges $\rightarrow$ updateEventRange $\times$ M $\rightarrow$
    …
    updateJob (last heartbeat) $\rightarrow$
    merge job generation

Tadashi Maeno

Issues, questions

- How will runEvent know when AthenaMP is ready to process events?
- Because unmerged files are not registered to DDM, a new column for SURL/PFN is required in the JEDI file or event table
  - Unless filename is derivable by construction, and location can be inferred from the aggregation point?
  - Add aggregation point as a PanDA job attribute?
- Where do we get the (EVGEN) event tokens for G4 simu production? (No EVGEN TAGs in DB)
  - Make TAGs during EVGEN production? Make them specifically for event service needs? Second is most practical in near term; event index in longer term
  - Entries for TAG files need to be in PFC available to payload jobs
- What exactly should the EventIndex contain such that it can serve as the token lookup for the event service?

Event Index content for Event Service token retrieval

- Basis of the query is run#, ev#, GUID (LFN also available)
  - As obtained from PanDA/JEDI
  - From GUID we also have PFN/TURL as obtained by pilot and recorded in PFC
  - Because the file is specified, the exact processing stage / format / version is specified
- What we want returned from the query is the token(s) for that event in that file
- Bulk queries should be supported, either individual event numbers or event ranges

Conclusion

- PanDA extensions for Prodsys2 are designed to enable efficient & flexible resource usage across a diverse range of resources – multi/many-core, HPC, clouds, opportunistic, …
- Together with AthenaMP, event I/O and FAX, we have the basis for implementing an event service to use these resources efficiently
- Effective WAN direct access to data is an enabler and a prerequisite
- Progress towards a prototype is good, thanks to enthusiasm of our key developers
- Most of the 'who' and 'how' questions for an initial implementation answered, but not all
  - In particular output aggregation needs attention
- Status following this meeting and discussions this week will be described by Vakho in an event service plenary on Friday
  - And will update twiki