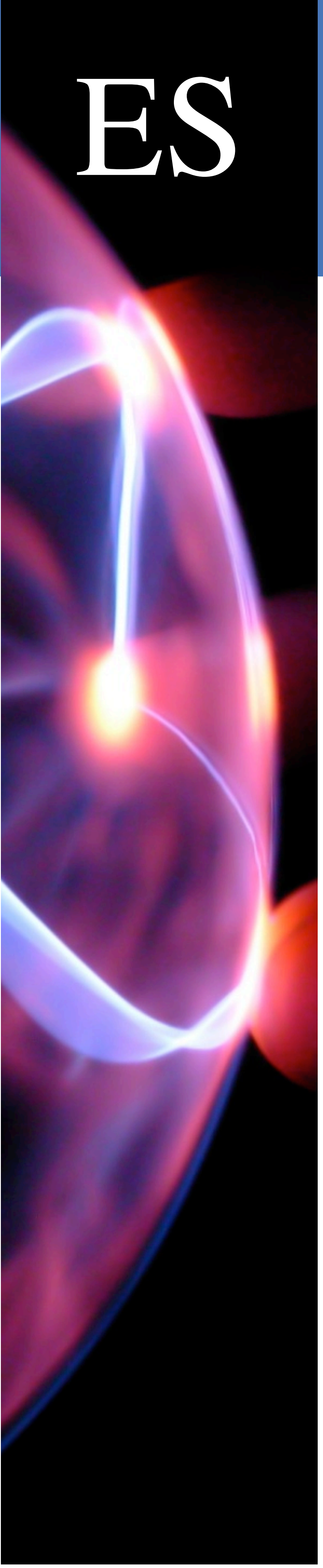
WLCG Collaboration Workshop
New York, 20 May 2012

Feasibility Study on Common Analysis Framework

Common architecture & Proof of Concept

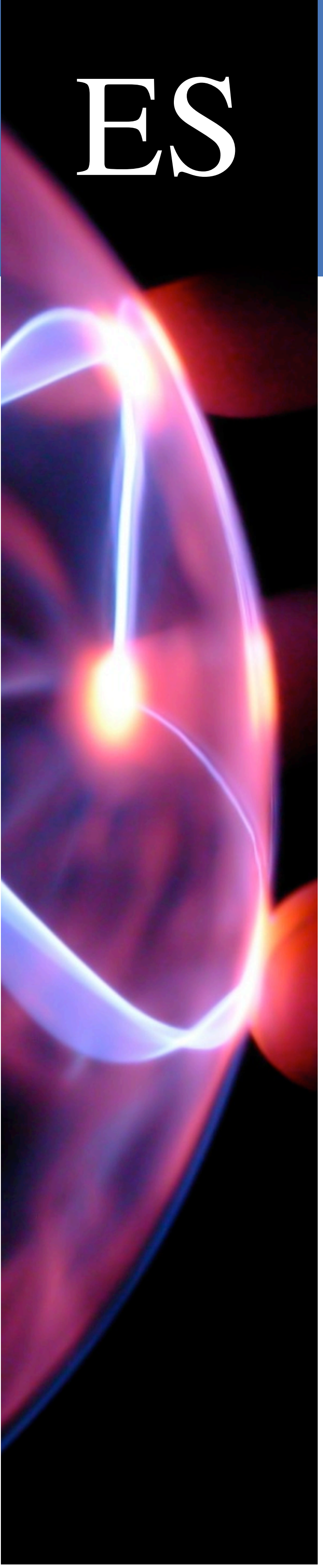
Fernando Barreiro, Mattia Cinquilli,
Daniele Spiga, Dan van der Ster

CERN IT
Experiment Support



Outline

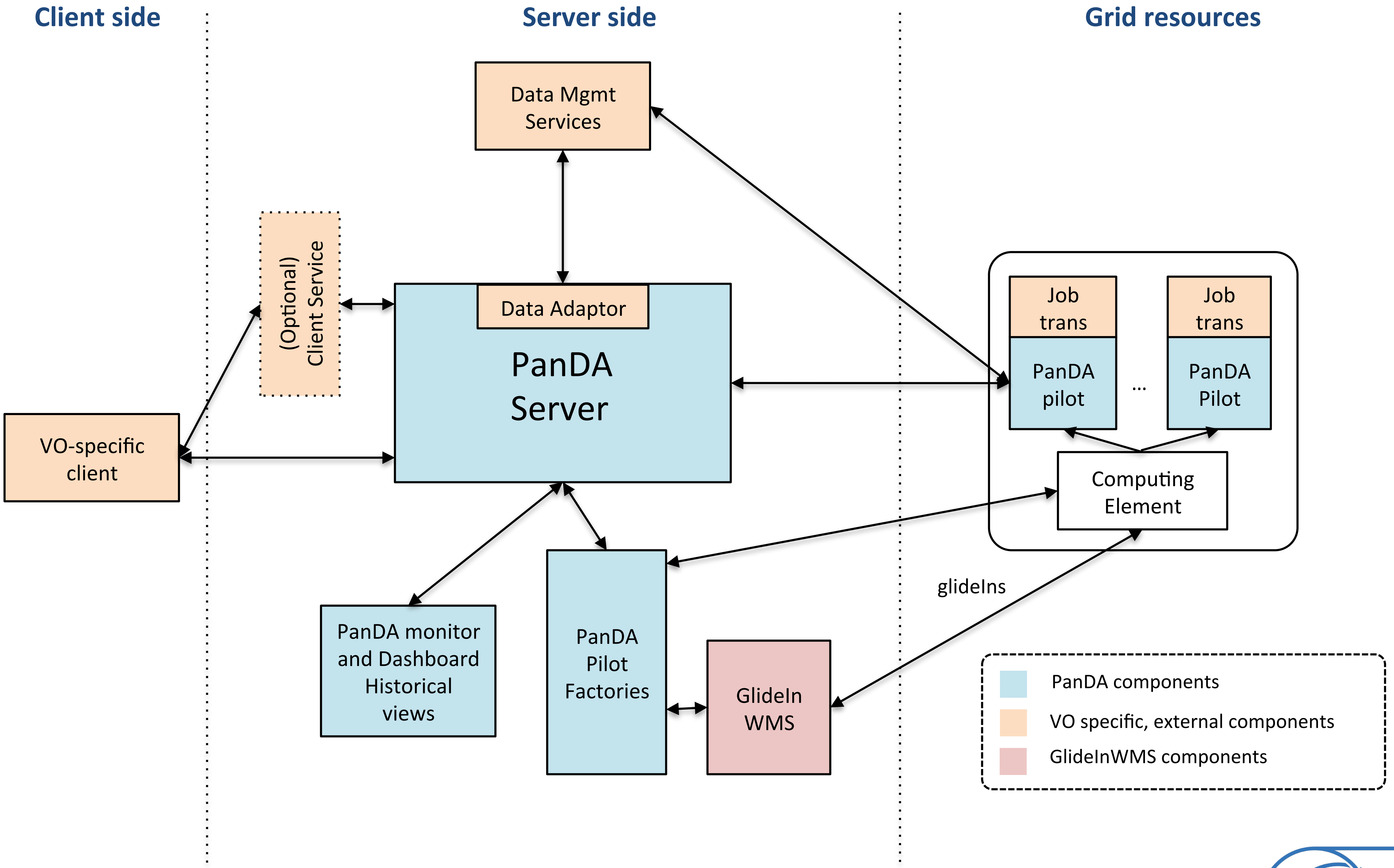
- Proposal for a common architecture
  - Schematic overview
- Strategy for a Proof of Concept prototype development
  - Steps and expected goals
- Summary and next steps



Common architecture proposal

- The proposed common architecture is based on PanDA and GlideinWMS (as Fernando explained)
- Main goal is to use a common engine while preserving the possibility to run experiment specific workflows and to interface experiment-specific services (e.g. data management)
    - Crucial point is to provide handles to the experiment for their specific needs

Architecture Overview



Client

- Analysis workflows are experiment specific
  - distinct client tools would be required
- Job preparation not handled by PanDA (data location discovery and splitting)
- VOs will also be required to develop experiment-specific job transformations (which run on the worker node).

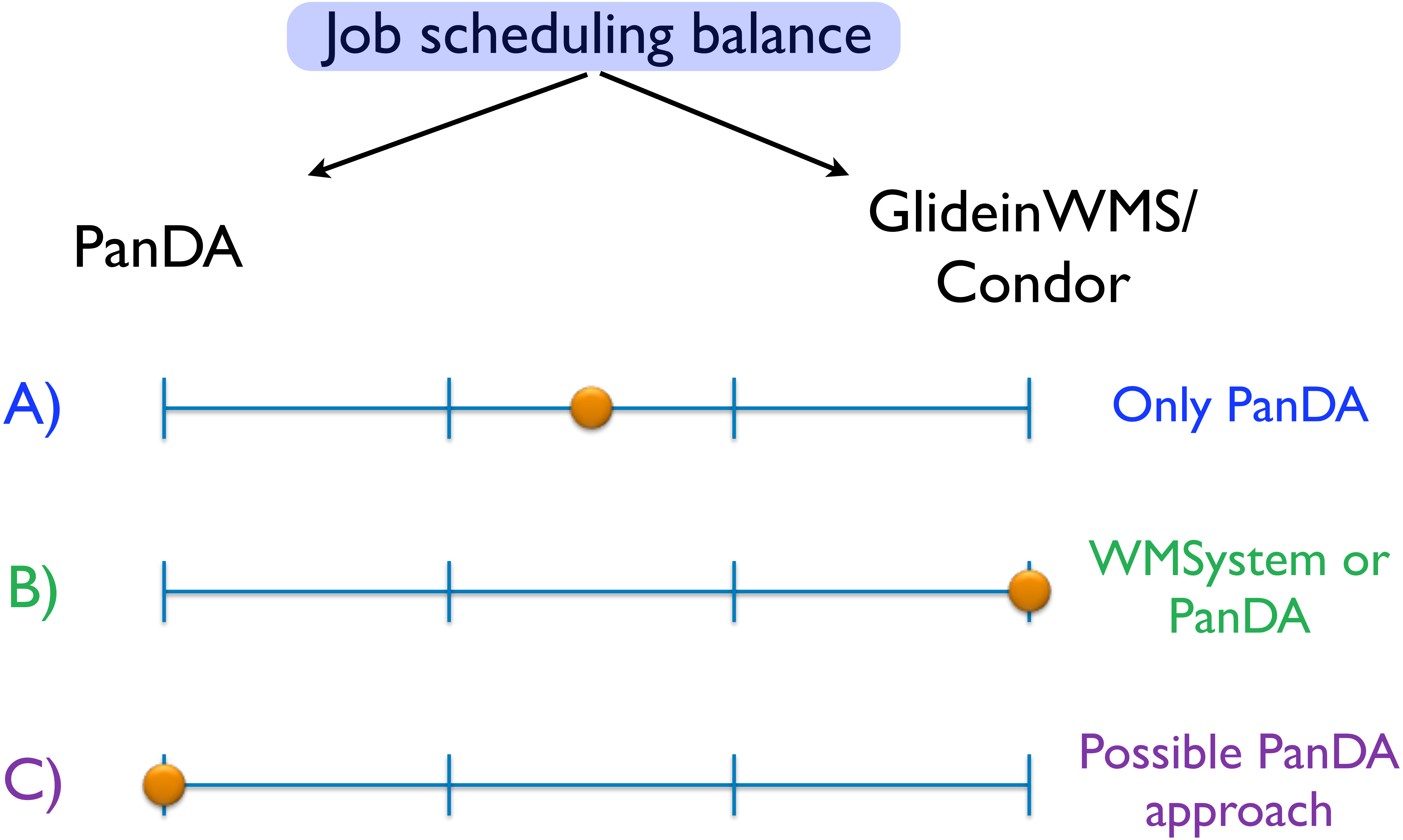
➡ Two solutions can be foreseen:

1. Fat client performing extra operations to translate user request into PanDA subjobs
2. Thin client and a central service automatizing some or all of the client operations
   - as data location discovery and job splitting
   - can also include experiment-specific input sandbox cache

Interface PanDA and GlideinWMS

- Using GlideinWMS as a service to inject PanDA pilots to worker nodes
- Both are job scheduling systems, with independent fair share and priority mechanisms:
    - need to avoid conflict on their scheduling decisions
    - various scenarios available
- Should not exclude parallel support of other pilot factory execution backends:
    - preserving current PanDA implementation
    - guaranteeing a smoother transition

Possible scenarios



Job scheduling balance

PanDA

GlideinWMS/ Condor

A) Only PanDA

B) WMSystem or PanDA

C) Possible PanDA approach

Scenario A

User-Specific PanDA Pilots in GlideinWMS

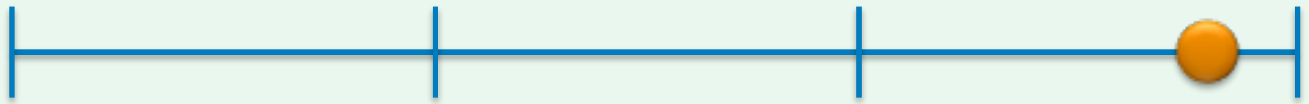
1. AutoPyFactory (one thread per site/user) submitting to a VO Condor schedd (with user's credential).
2. GlideIn factory submits appropriate glideins to the Grid, scheduling jobs for multiple users.
3. PanDA pilot run and retrieves payloads from PanDA server.

➡ Implications: two knobs to control job schedule

- Condor handles inter-user priority
- PanDA handles intra-user priority
- Current PanDA share logic to be moved to Condor
- APF needs some development

Scenario B

Full Job Payloads in GlideinWMS



Natural usage of GlideinWMS:

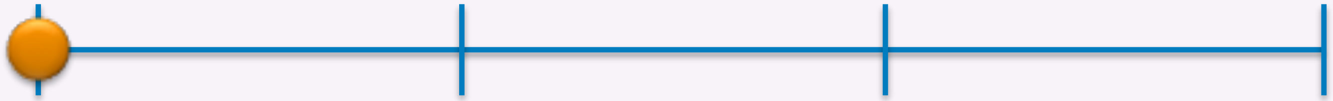
1. Injecting the complete job payload to Schedd (with user's credential).
2. GlideinWMS handles all aspects of scheduling and job ordering.

➡ Results: improved security, relying more on Condor

- Condor handles ~all job scheduling and fair share mechanism
- Rebrokerage would be more complicated
- Require development of an adapter to directly interface PanDA and GlideinWMS

Scenario C

User-independent Pilots in GlideinWMS

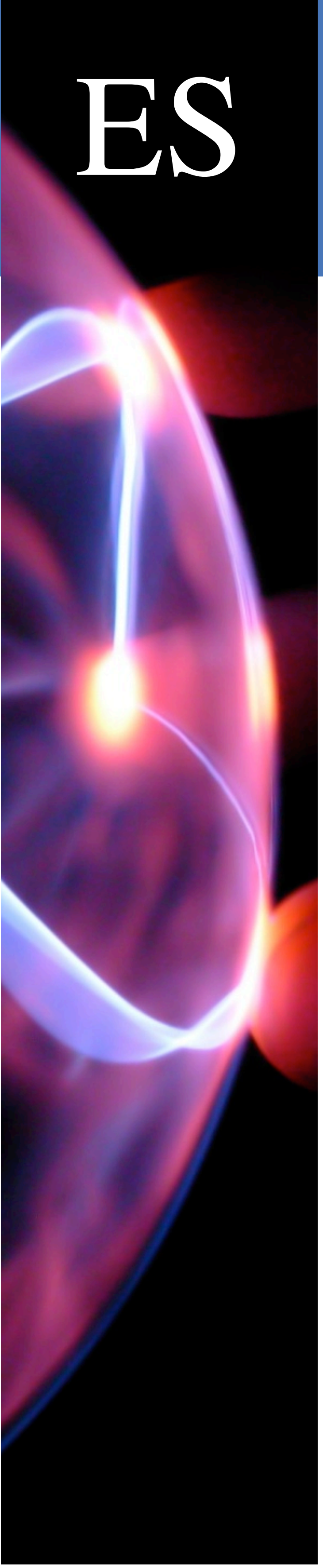


1. AutoPyFactory injects user-independent pilots into Schedd, using a credential able to retrieve user jobs from PanDA
2. PanDA pilot should then:
    1. get the payload from the PanDA server
    2. use MyProxy/gLExec to switch to the user.

➡ Implications:

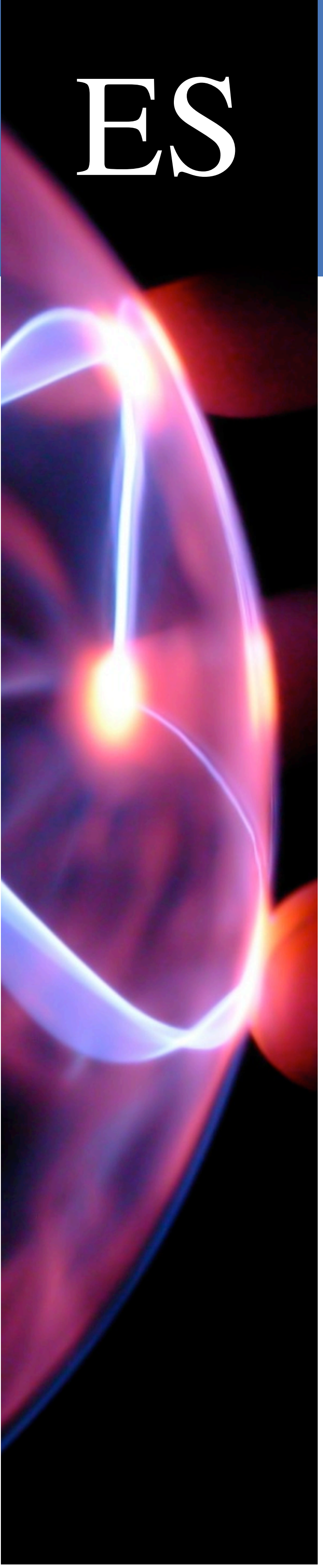
- PanDA would control the job schedule completely
- MyProxy/voms scalability would need to be assessed
- GlideinWMS' glexec feature would not be required

Worker Nodes



- PanDA pilot would be responsible for:
  - interactions with PanDA
  - environment setup
  - job data I/O
  - executing job transformation with job parameters
- These functionalities would need to be factorized and made modular
- Opportunity to put PanDA or VO-specific worker node validation requirements into glidein startup scripts

Proof of Concept proposal



Proposal

- Feasibility study has convinced us of the capability of PanDA+GlideinWMS as a common framework
- Proof of Concept (PoC) goals:
    - to demonstrate the common architecture exploring aspects which have not yet been technically proven
    - to develop a prototype of the common analysis framework PoC
        - using PanDA server as common core
        - interfaced to GlideinWMS as pilot factory to access remote resources

Main topics to explore

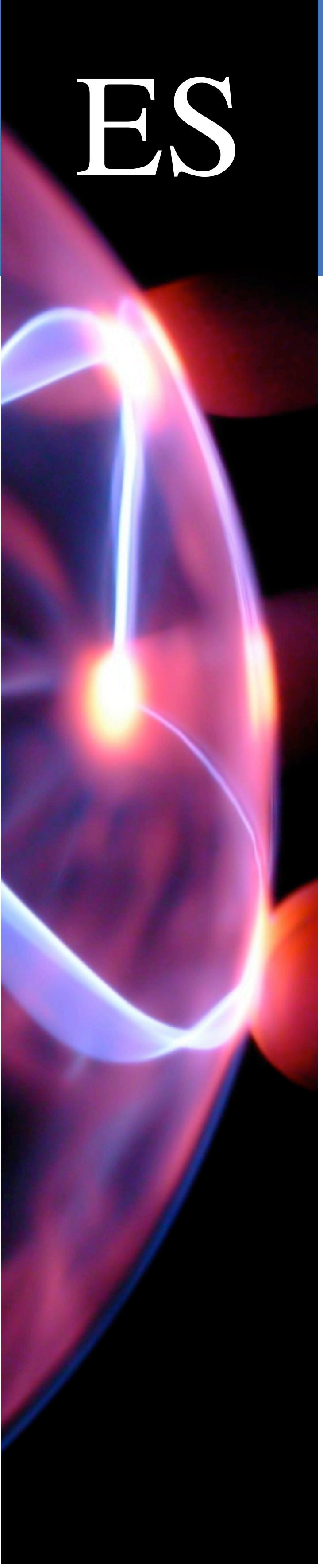
Two main topics have been identified

1. Enable PanDA for CMS

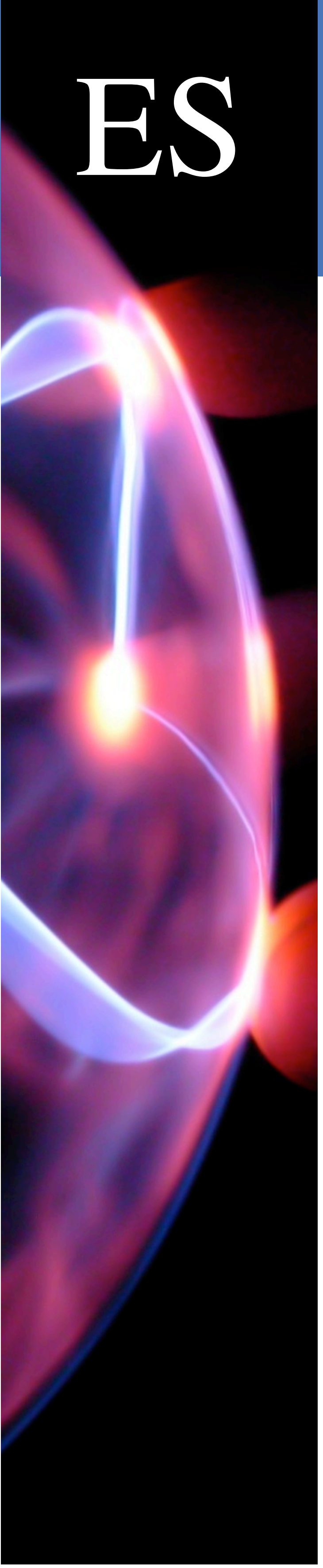
- defined a multi-step process isolating distinct set of changes at each step
- existing code or libraries from CRAB or WMSystem should be re-used, wherever possible

2. Define the best interfacing of PanDA to the GlideinWMS:

- need to consider the overlapping features
- guarantees that no functionality is lost
- Rod Walker started evaluating Scenario A (as Fernando explained)
  - PoC will also evaluate the results coming from this work to understand best interaction between the two



Proof of Concept
multi step process



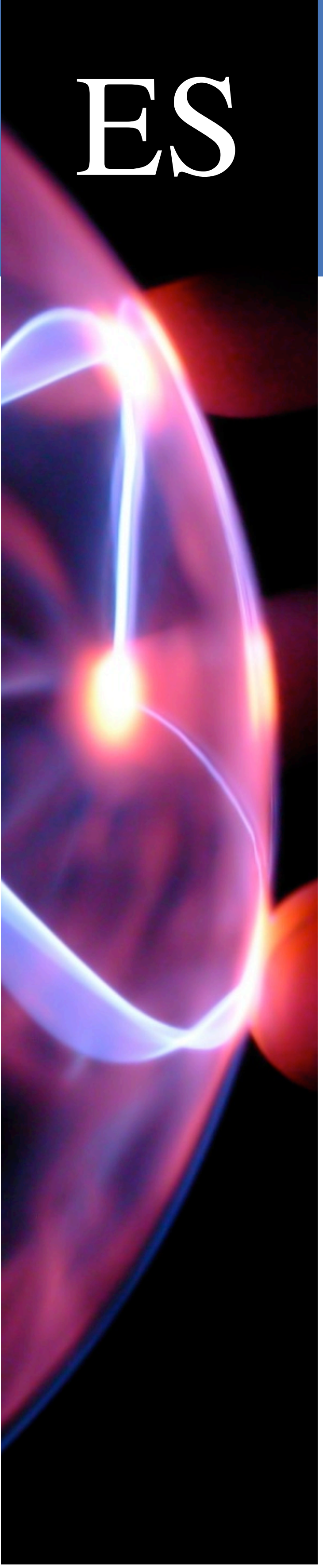
Run “basic” CMS jobs

STEP 0: Run Basic CMSSW job

- Goals:
  - to demonstrate the ability to bootstrap CMS environment
  - and to run CMSSW “Hello World” job
- No I/O needed at this stage
- Using PanDA client tools

STEP 1: Include pilot factory

- Goals:
  - add a pilot factory to facilitate the tests
  - use resources at few limited friendly sites
- The factory could optionally be provided by GlideinWMS or a new AutoPyFactory instance.

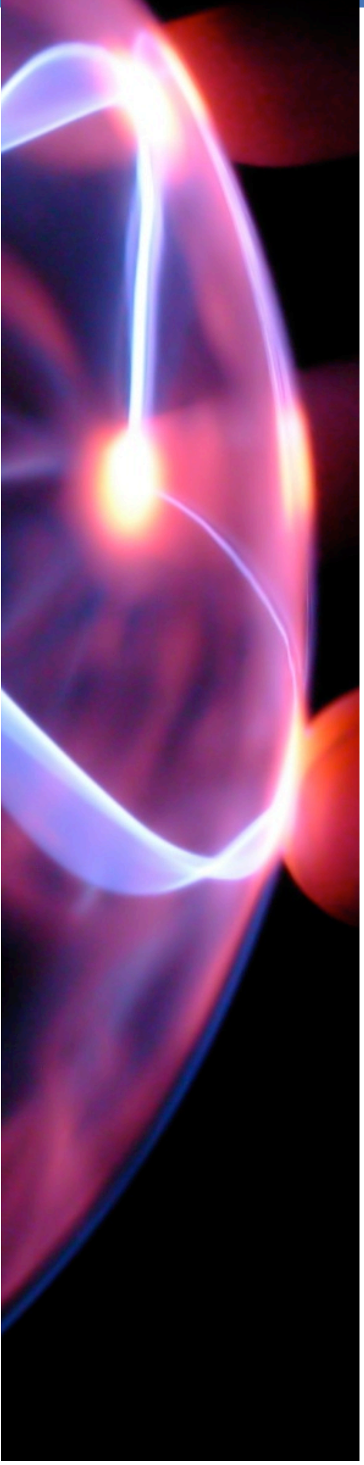


Client

STEP 2: CMS client tool

- Goals:
  - to develop a simple client to translate user request into jobs for PanDA server
  - introduce the input data management
- Requirements:
  - enable data and location discovery by interacting with CMS specific data services
  - implement CMS-specific transformation (e.g. runCMSSW)

➡ Evaluate effort needed to reuse existing CRABClient and CRABInterface for the eventual production client and thin Client Service layer.



Analysis Output

STEP 3: Output file handling

- Goals:
  - to demonstrate capability managing job results without ATLAS specific services (e.g. DQ2)
  - local stage-out of job output files, using CMS namespaces
- Requirements:
  - include stage-out in CMS transformation, using Trivial File Catalog
  - PanDA server stores the Logical File Names of generated output files, skipping DQ2 registration

STEP 4: CMS output management

- Goal:
  - to demonstrate asynchronous stage-out using the CMS AsyncStageOut tool
- Requirements:
  - PanDA server has to provide an API to access job information
  - a new source plugin for the AsyncStageOut tool will be needed

Access to results

STEP 5: Log and output access

- Goals:
  - to provide access to job output results
  - to validate the access to logging files in the PanDA web monitoring interface
- Requirements:
  - develop an extension of the experiment specific client tools to copy remote job output results, by retrieving the information form PanDA and AsyncStageOut
  - adapt the PanDA monitor to read the CMS FileSpec's to access log files.

Summary

- ✓ No show-stoppers are shown in the adaptation of PanDA for both experiments and its interfacing to the GlideinWMS (as from Feasibility Study)
- ✓ Defined a common architecture using PanDA and GlideinWMS
- ➡ Proof of Concept prototype will demonstrate if the proposed common analysis framework is achievable

Next Steps

✓ Defined a work plan to develop the proposed Proof of Concept

- If PoC demonstrates the soundness of the proposed architecture:
  - ➡ use the PoC experience to build the final architecture
    - once the final product is eventually ready need an integration phase