Roadmap towards WM standardization for HL-LHC: OSG perspective

Brian Bockelman
WLCG Workshop 2018

Words of the day:
standardization
common APIs
common approaches
common projects

WARNING:
Technology-Free Talk Ahead

Detour: What Workflow Management occurs in OSG?

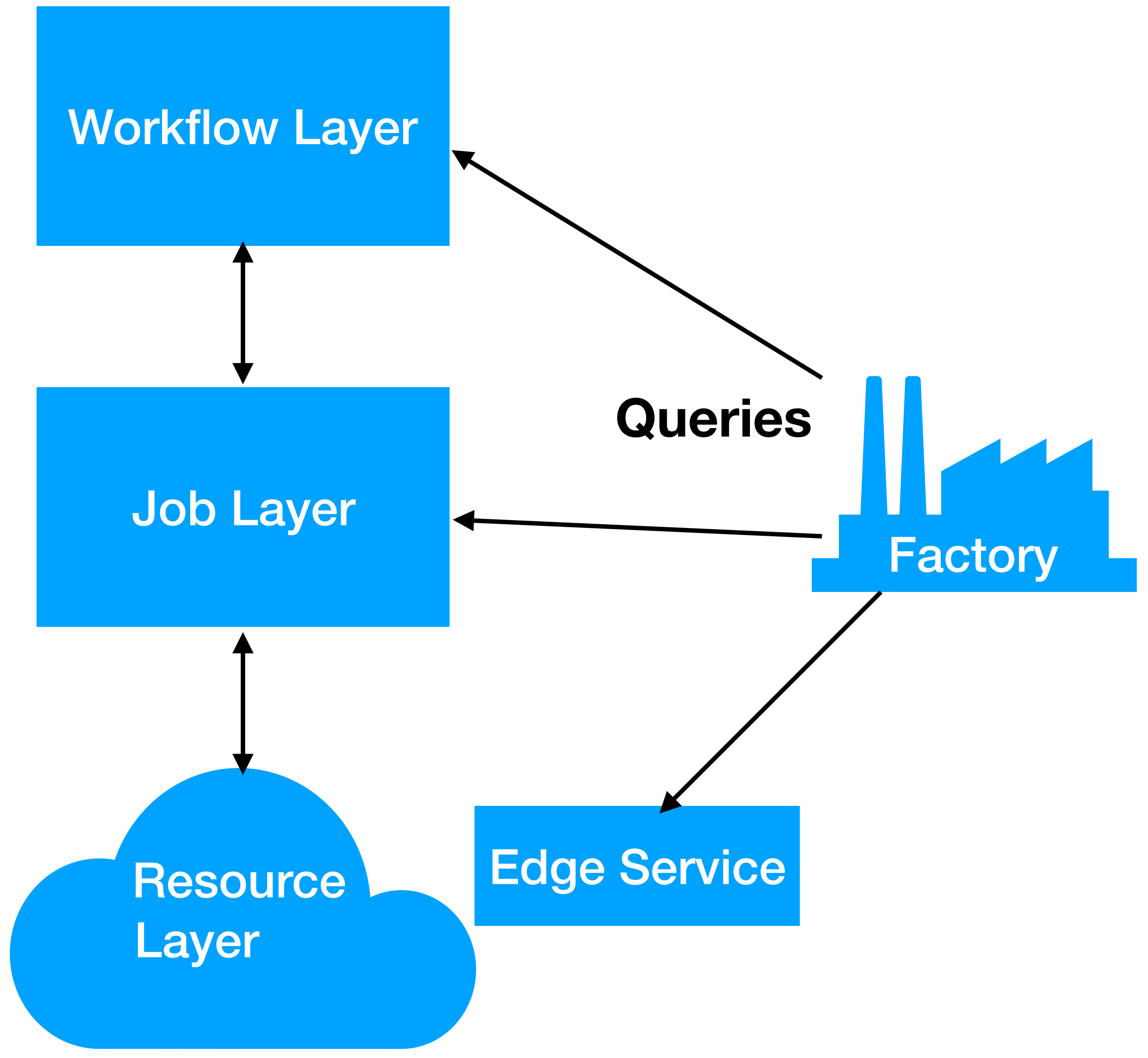
- Workflow management in OSG follows the overlay model:
  - User defines a workflow in a VO-specific workflow scheduling component.
  - “Factory” requests resource allocations from a distributed set of computing resources to meet workflow needs. When started, these new resources are unassigned and go into a central pool of free resources.
  - Workflow scheduling component examines available resources and assigns workflows according to some policy.

Workflow Management: One step more specific

- Within the OSG, there are two common setups for the overlay model:
    - PanDA serves as the workflow manager. AutoPyFactory serves as the resource factory.
    - HTCondor serves as the job manager (workflow layer can also be in HTCondor or elsewhere). GlideinWMS serves as the resource factory.
- The resources involve tend to be allocated from batch clusters in chunks of 1 core (to N cores, but all on one node).

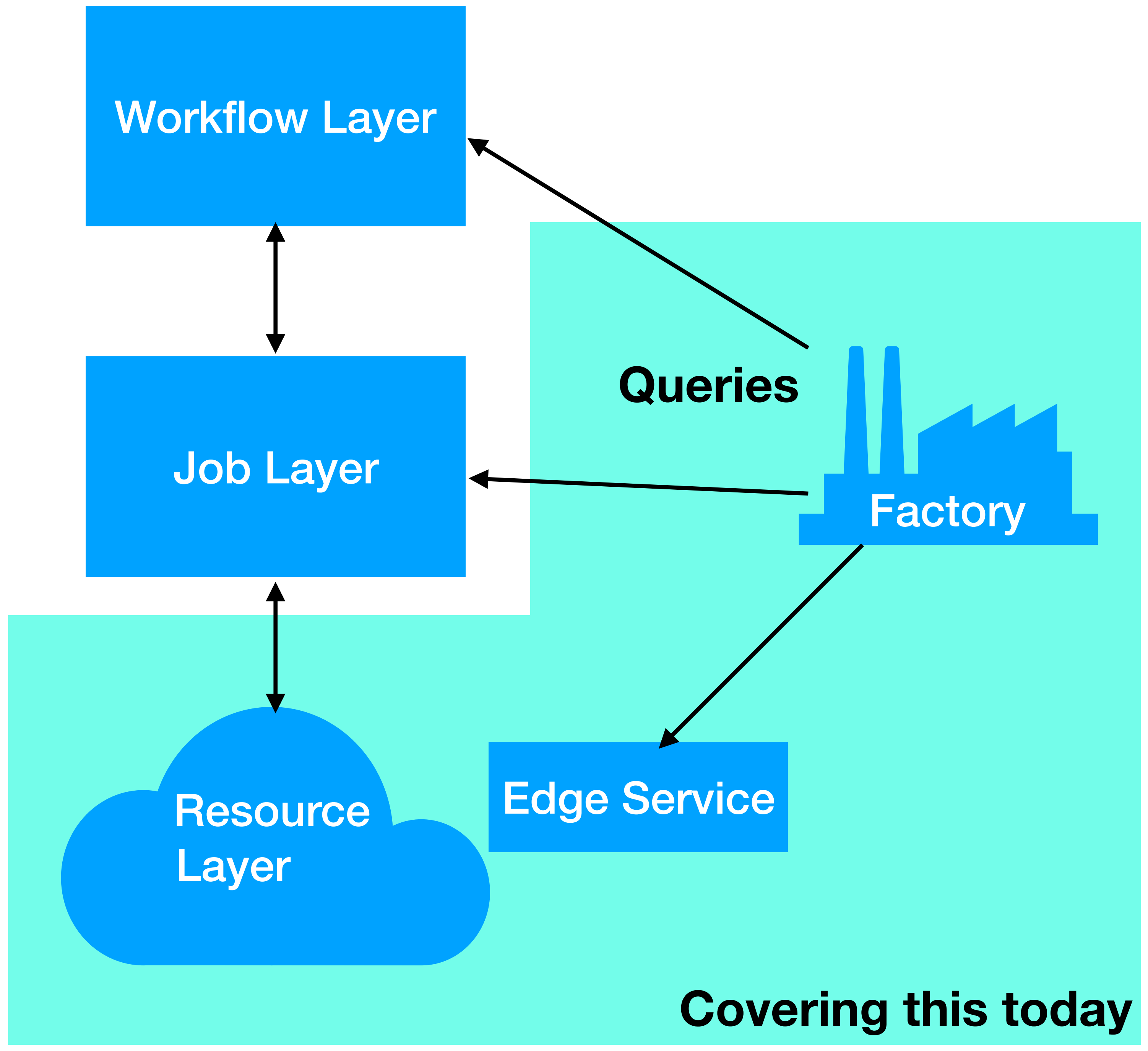
I'll be talking more about the resource allocation workflows today.

My Mental View



This is a bit of the classic view, which is beginning to evolve as the "resource layer" starts to include more non-grid resources.

My Mental View



This is a bit of the classic view, which is beginning to evolve as the "resource layer" starts to include more non-grid resources.

An Aside on Standardization

Here, by *standardization*, I mean having interoperable “on-wire” representation.

- We have already been through a round (two rounds?) of standardization; most recently was the OGSA Basic Execution Service (BES), circa 10 years ago.
  - Who here is participating on the Workflow Execution Service (WES)? https://github.com/ga4gh/workflow-execution-service-schemas
- Lessons learned:
  - Standardization takes a long time! Long enough that the technologies involved may be obsolete by time it is done.
  - Standardization captures the lowest-common-denominator functionality
  - Standardization helps mostly for very large communities (it is expensive!) or where a high degree of interoperability is needed (e.g., transfers).
- The important aspect is **not the technology you use today but your ability to switch to the technology of tomorrow**. Accordingly, we will always need “multi-protocol clients”.

Alternates to Standardization

- If standardization is hard, should we all have a lovely gelato here, go home, and do different things? **NO!**
- There is a **benefit to consolidation in terms of overall effort savings**. We may not want one protocol, but there's definitely advantage in going from 10 to 3.
- We want our community to approach new sites with one voice, particularly in terms of **providing a baseline capacity that is experiment-independent**.
- We should **tackle greenfield projects together** (either common projects or at least coordinate projects), especially as we look at new site functionality.
- Long story short: **common approaches, common capabilities, maybe not standardization**.

Consolidating Implementations

- I think this is already happening naturally …
  - … but it’s important to recall there’s a very, very long tail.
  - I recently discovered that there’s one site still running GRAM.
- I don’t see much benefit in actively pushing particular implementations “off the edge”; rather, let the community pick the best-of-breed.

Baseline Capacity

- Through some combination of Brownian motion and detailed planning, we’ve arrived at some concept of a compute service:
  - Run as a site-level service.
  - Remotely accessible.
  - Some mechanism for remote authorization / authentication.
  - Bulk submit of jobs (job descriptions), query, cancel.
  - Movement of input and output sandboxes.
  - Transfer of credentials.
- Fits well with our “classic grid” model. Shouldn’t this be a starting point for discussing new resources?

Evolving beyond baseline

- We see hints of the baseline breaking down at the large HPC sites:
  - Sites are disinterested in running services for an experiment (maybe not even a community?)
  - No ability for pilots to be “pushed” to the edge service.
  - No ability for pilots to “pull” payloads due to network access restrictions.
  - Need to pre-stage data or pre-setup storage environment (configure burst buffers prior to job launch).
- In some cases, a complete breakdown: non-job-based models for backfilling HPCs.
- The community sees utilizing HPC as a core part of the HL-LHC challenge! So, how do we tackle this?

Greenfield projects

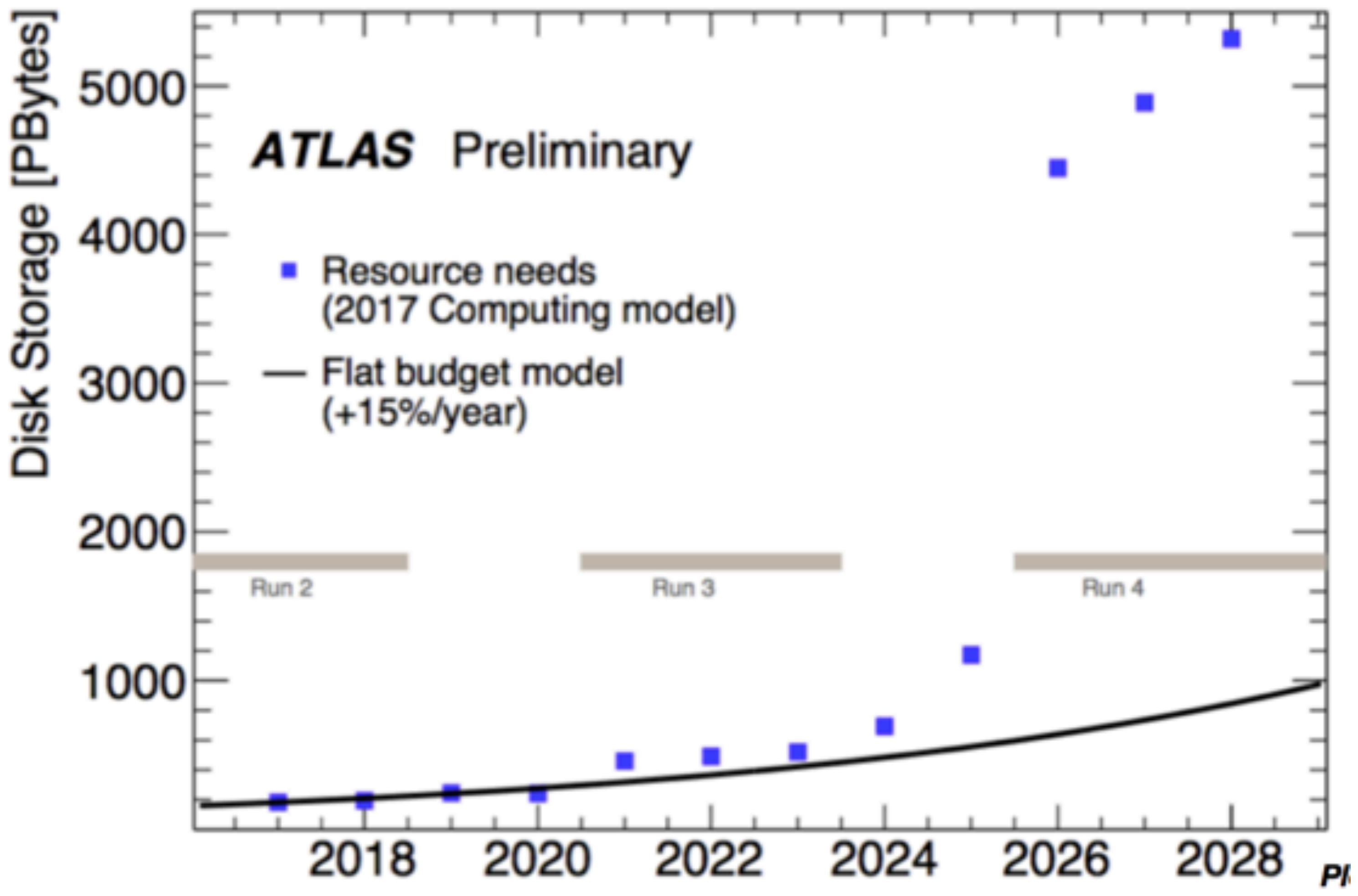
- Two ideas of items where we can work as a community.
- Finding commonality / one voice in edge services:
    - Again, where traditional CEs are applicable, use those!
    - Should we revisit “the original HTCondor-CE idea”, a gLite project that focused on a very lightweight CE that did not run jobs but only started a VO-specific CE.
- Design “service connectors” — ability to pull jobs from an upper layer and push into an external service.
    - Twist on the Condor-G approach!
    - Should we standardize on the “connecter interface” to *pull* from the job layer, as opposed to the *push* model?
    - It’s an interesting approach, but let’s not forget the reasons we went to late binding in the first place!

Concluding Thoughts

- I purposely focused on the layer below workflow management.
- Let’s be careful when we say “standardization”.
  - These efforts can be quite expensive and may require big payoffs.
- There’s natural consolidation occurring, but we should carefully think about having a common approach at new (particularly, HPC) sites.
- Do not retread old ground, but let’s find commonalities in the new challenges. Let’s explore:
  - Different push-vs-pull models.
  - Mixture of direct payload-vs-overlay.
  - Handling explicit setup of the local data environment.

The gelato here is great, but what’s the forum for tackling this?

DRINK!



ATLAS Preliminary
Resource needs (2017 Computing model)
Flat budget model (+15%/year)
Disk Storage [PBytes]
5000
4000
3000
2000
1000
2018
2020
2022
2024
2026
2028
Run 2
Run 3
Run 4