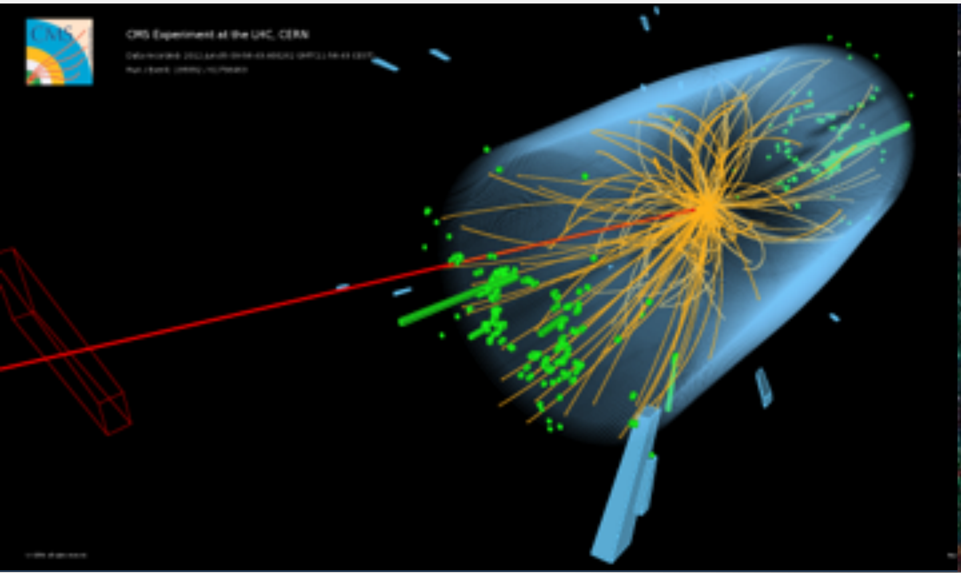
Workflow management in CMS

Eric Vaandering



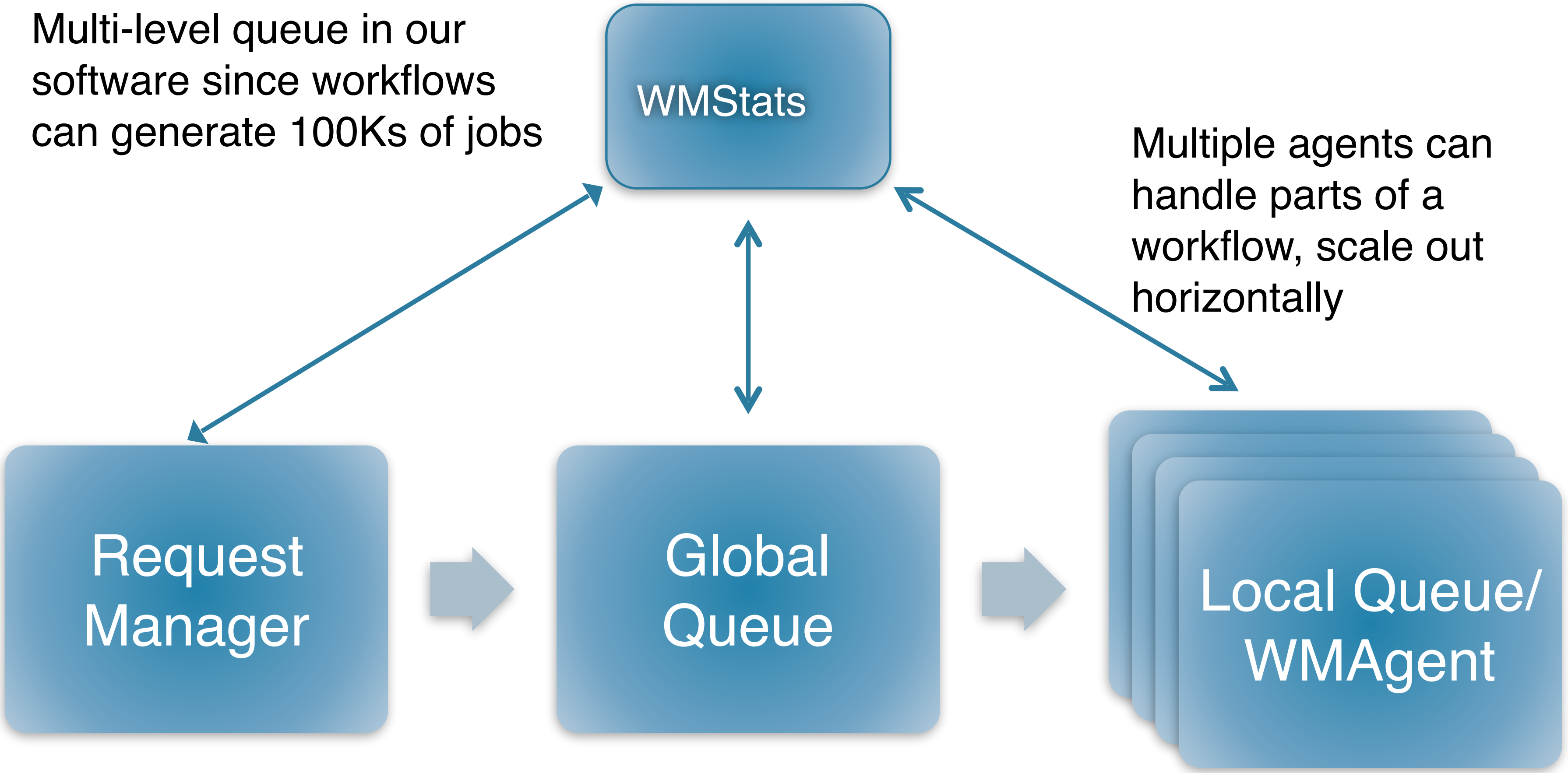
Intersection with Data Management

- It's difficult to discuss workflow management without saying at least something about data management
- The CMS model is to send jobs to data (most of the time).
  - We have way too much data/too many sites (~60) to ignore this
  - Very few replicas of most data
- Primary data management consists of
  - DBS (dataset/file catalog, runs, sub-runs, other meta data)
  - PhEDEx (location metadata and movement system, subscriptions)
    - Uses FTS underneath
- Secondary data management:
  - AAA/xrootd: remotely readable global namespace
  - Dynamic DM: issues PhEDEx commands to replicate(delete) (un)popular data

WMAgent diagram

Multi-level queue in our software since workflows can generate 100Ks of jobs

Multiple agents can handle parts of a workflow, scale out horizontally



WMAgent interactions

- Resource provisioning and job execution delegated to HTCondor/GlideinWMS
  - DAGs are interesting, but not really flexible enough for our workflows
  - Tell GlideinWMS all the places a job can run, resources needed, it takes care of the rest
  - Part of "rebrokering" is handled by GlideinWMS: jobs waiting can be overflowed to other sites well connected (xrootd) to the data
    - Other way is that new locations from DDM can be included before jobs are submitted to GlideinWMS
  - Plans to enforce overall job limits within GlideinWMS
    - e.g. merge jobs are hard on sites, need to limit the overall # running per site
    - currently managed by restricting number submitted *per agent*
  - Resubmissions handled by agent based on return codes (some retried, some not)

Other dependent services

- Components of WMAgent communicate with a number of CMS services (all REST based)
  - SiteDB/Dashboard for understanding grid configuration/site status
    - Evaluating CRIC (nee AGIS) as a common WLCG project
  - DBS/PhEDEx for data discovery (what data is in a dataset, where is it?)
  - Components that publish data into DBS and subscribe data to their final destination(s)
  - Have or planning to change out or upgrade all these layers with minimal disruption

Client services

- Workflow planning and checking was major operator overhead (1000s of simultaneous workflows)
- External services and scripts feed work into Request Manager via REST interfaces
  - McM and Unified used to construct workflows and prestage data
  - Back end checks prior to announcing data is ready, preparing recovery workflows
    - aim to vastly reduce the recovery workflows in next couple of years by incorporating into WMAgent
- Request Manager holds request information which can be aggregated with dataset metadata

Analysis system

- Second system, similar in design to JobSub, for user analysis
  - Some underlying code shared with WMAgent
  - Reliant on GlideinWMS, minimal use of DAGs
    - Jobs go to same global GlideinWMS pool for prioritization
- Differences with production system
  - Package and ship user code to worker nodes
  - Simpler workflows, better status tracking
  - User client driven — more interactive
  - No merging (yet) of outputs
  - Uses a different data movement system (also based on FTS)

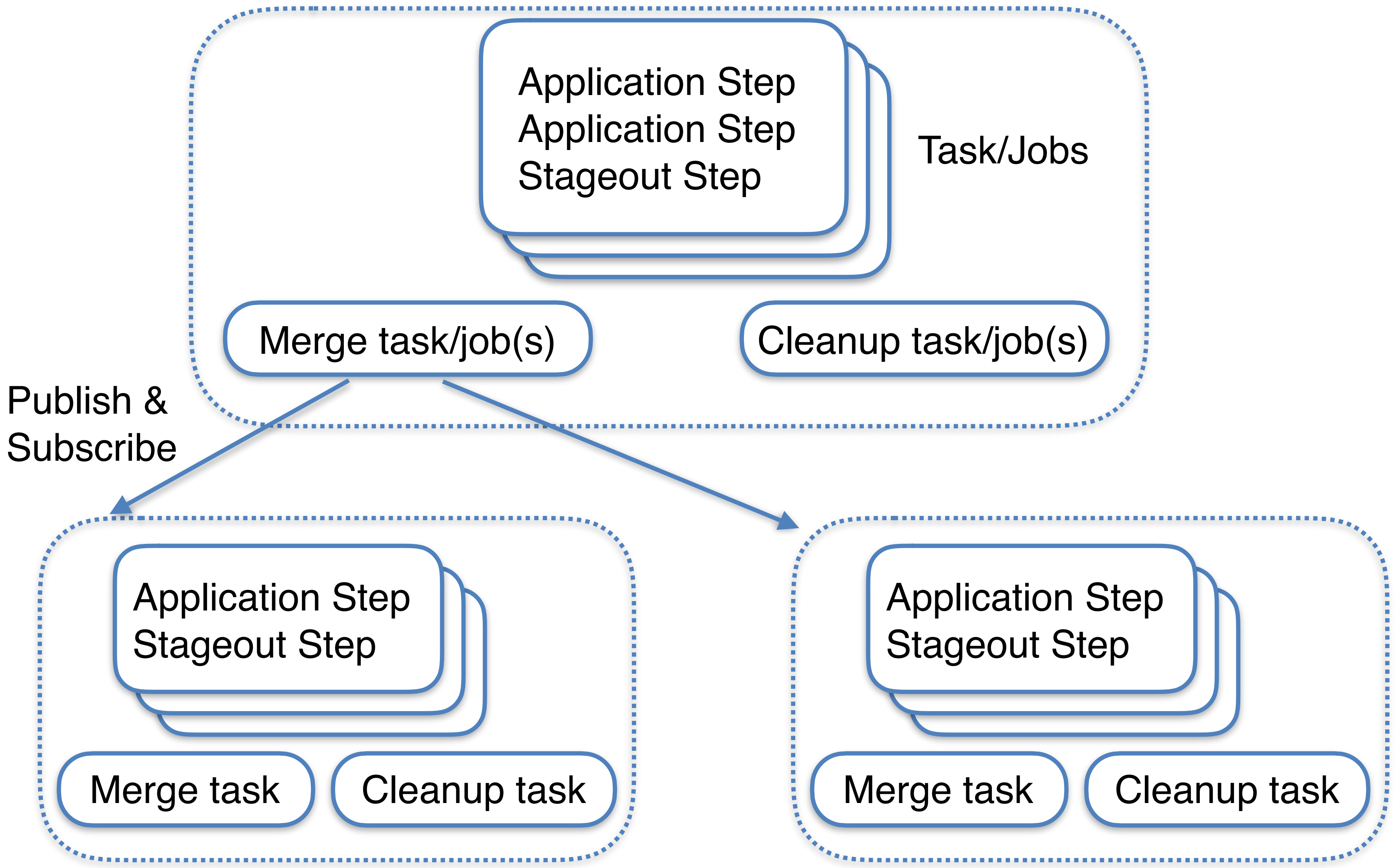
DUNE requirements

- WMS for resource management
  - ✓ exactly what WMAgent is designed to do
- record of SW configuration
  - ✓ keep record per workflow, new WMArchive keeps per job, finally DBS keeps for output files (more convenient)
- quickly suspend sites
  - ✓ resource control in WMAgent, pilot submission in GlideinWMS, submitted jobs may still start
- WF management layer
  - ✓ this is actually what WMAgent is
- monitoring system (both requirements)
  - ✓ have this in dashboard, HTCondor based monitoring & WMArchive

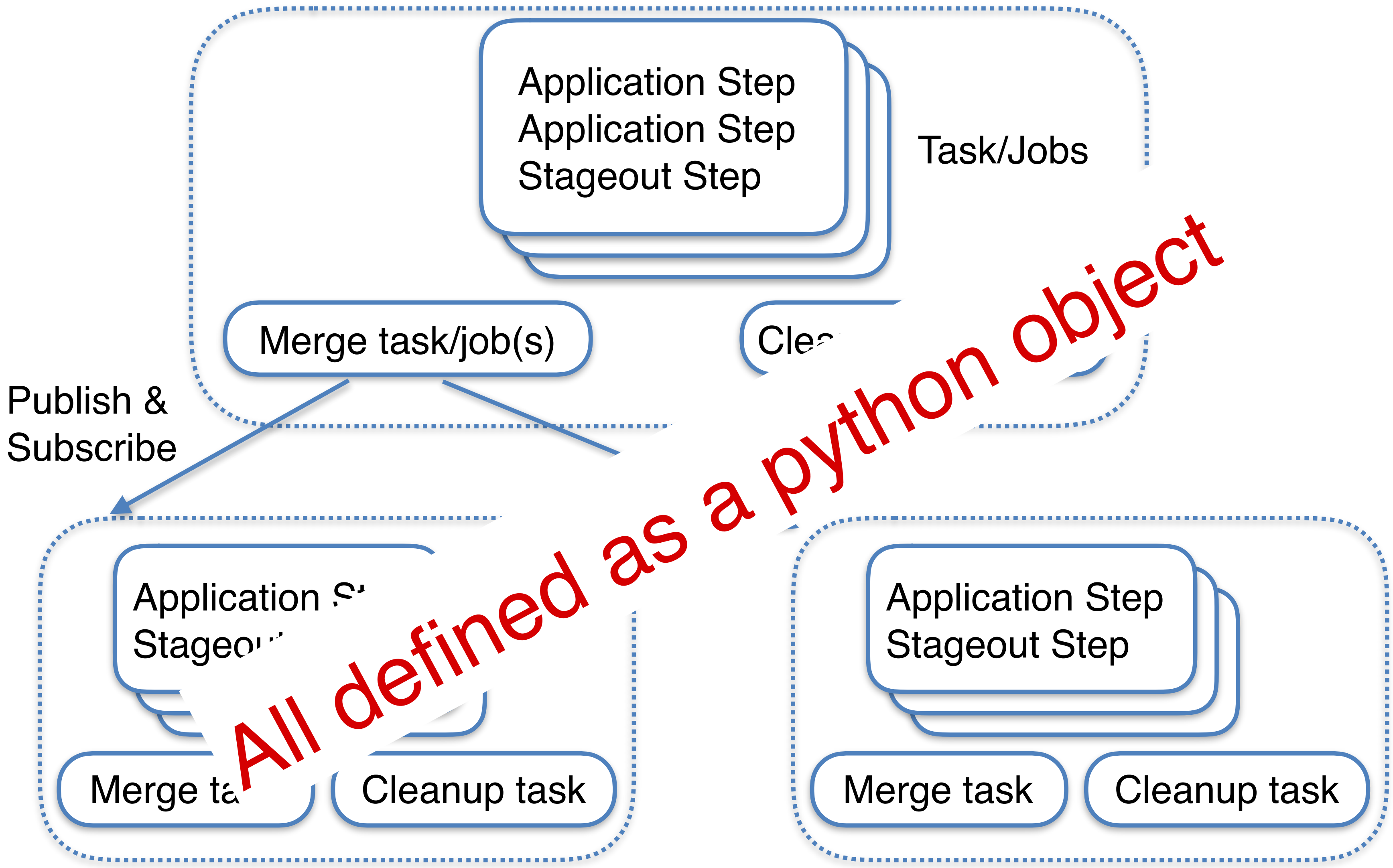
DUNE requirements & thoughts

- Human and machine interfaces
  - ✓ machine interfaces (REST) are robust. Human interfaces exist, prioritized for what we need
- Interact with data management
  - ✓ of course
- retries based on failure modes
  - ✓ including different back-off models
- DAGs
  - ✗ not flexible enough for the workflows we need to do

Sample Workload



Sample Workload

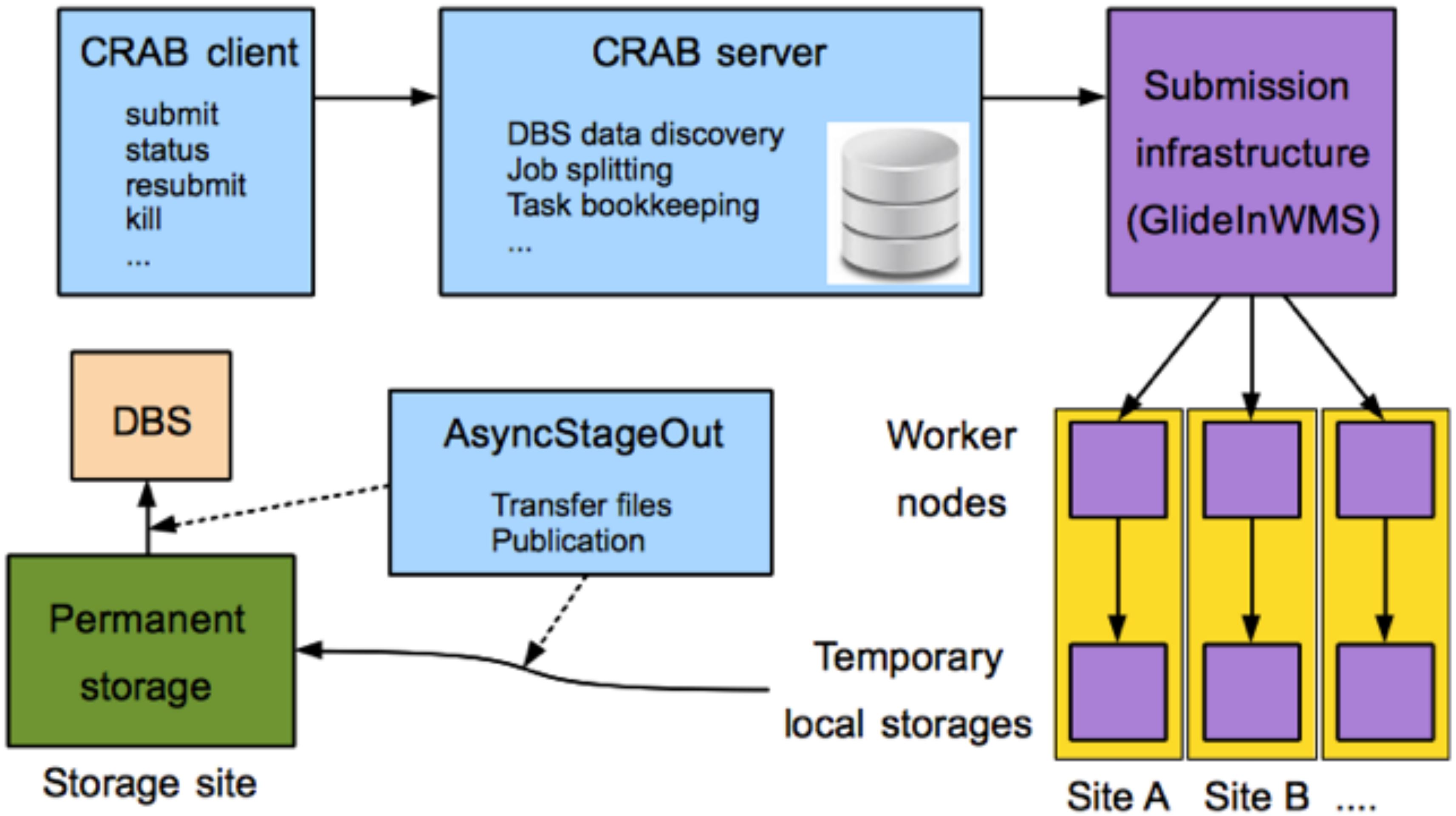


All defined as a python object

Recovery

- Recovery workflows are generated, parameters of the workflow can be changed, and workflows resubmitted to catch up missed work

CRAB Architecture



CRAB tasks

- User-friendly way to accomplish all needed steps of an analysis
  - Data discovery (what's in my data and where is it)
  - Job splitting (each job runs on a reasonable portion of the data)
    - Atomic unit in CMS is a luminosity section, 23s of data
  - Configure and run CMSSW (cmsRun) to run on correct files, lumis
  - Submit jobs
  - Publish resulting data in data catalog (DBS)
  - Move data to users' "local" institution (ASO and FTS)

Backup

- Backup slides

CRAB3 Condor/Glidein Interface

- We make light use of DAGMan and heavy use of Glideins
- DAGMan is used to separate tasks into job running and monitoring of data transfer, publication
- Glideins to limit execution sites, resources, etc.
    - US operates in failover mode — jobs waiting for some time redirected to other US sites, data streamed over xrootd

WMCore package

- Request Manager
- WorkQueue
- WMAgent
- WMStats (monitor)
- ACDC Server
- (T0- build on top of WMAgent, T0_WMStats)
- (DBS, CrabServer, DAS, SiteDB) – uses some WMCore library

What it does

- Help operation (monitor progress, trouble shooting, etc)
- Take request (workflow specification)
- Create jobs
- Submit jobs (to batch system, GlideIn/Condor)
- Track jobs, Retry jobs (job level, workflow level)
- Monitor jobs (by workflow)
- Archive workflow summary
- Archive data/statistics (outside the system – DBS, PhEDEx, Dashboard)