Optimizing new components of PanDA for ATLAS production on HPC resources

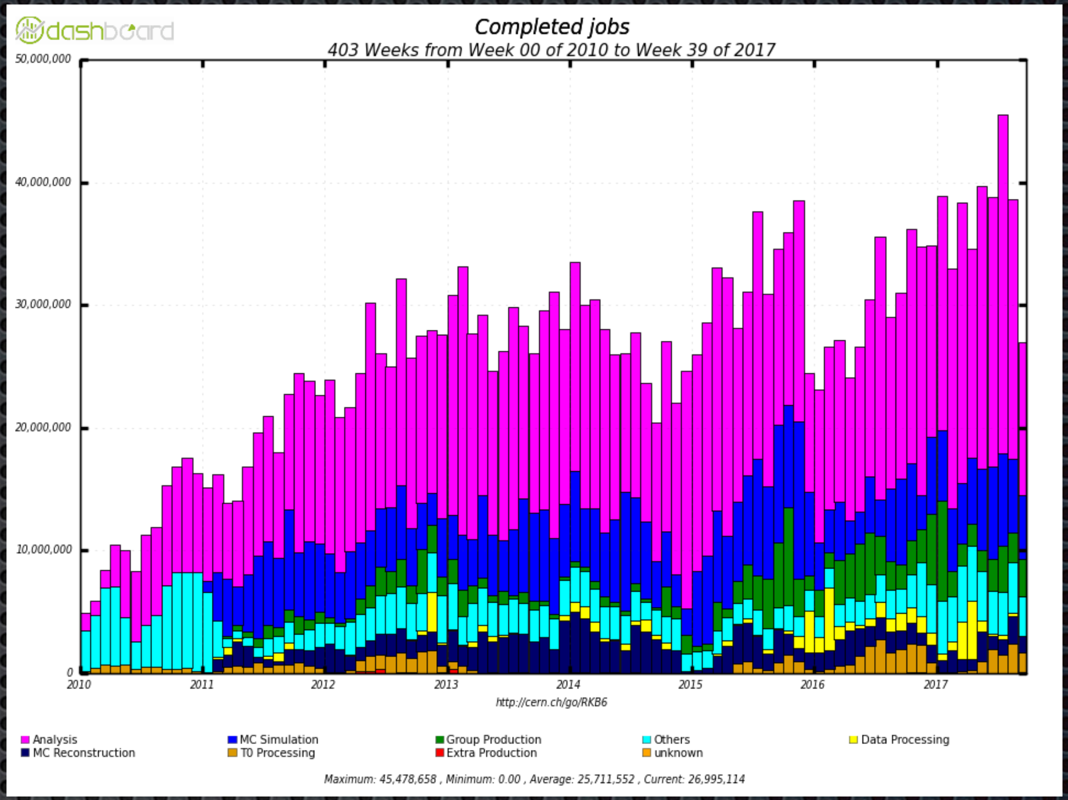
Tadashi Maeno, Fernando Barreiro, Paul Nilsson, Danila Oleynik on behalf of the ATLAS Collaboration

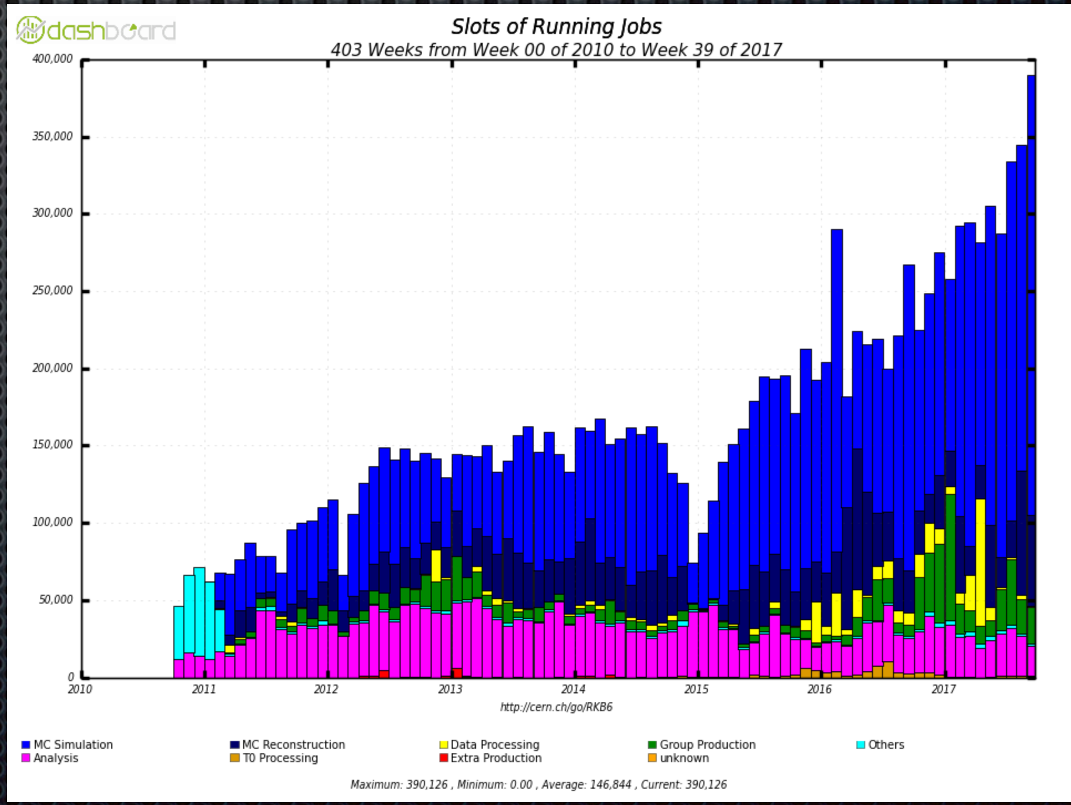
PanDA Workload Management System

- **The PanDA workload management system was developed for the ATLAS experiment at the Large Hadron Collider**
- **A new approach to distributed computing**
  - ***A huge hierarchy of computing centres and opportunistic resources working together***
  - ***Main challenge – how to provide efficient automated performance***
  - ***Auxiliary challenge – make resources easily accessible to all users***
- **Core ideas :**
  - ***Make hundreds of distributed sites appear as local***
    - *Provide a central queue for users – similar to local batch systems*
  - ***Reduce site related errors and reduce latency***
    - *Build a pilot job system – late transfer of user payloads*
    - *Crucial for distributed infrastructure maintained by local experts*
  - ***Hide middleware while supporting diversity and evolution***
    - *PanDA interacts with middleware – users see high level workflow*
  - ***Hide variations in infrastructure***
    - *PanDA presents uniform 'job' slots to user (with minimal sub-types)*
    - *Easy to integrate grid sites, clouds, HPC sites …*
  - ***Data processing, Production and Analysis users see same PanDA system***
    - *Same set of distributed resources available to all users*
  - ***Highly flexible – instantaneous control of global priorities by experiment***

PanDA Brief Story

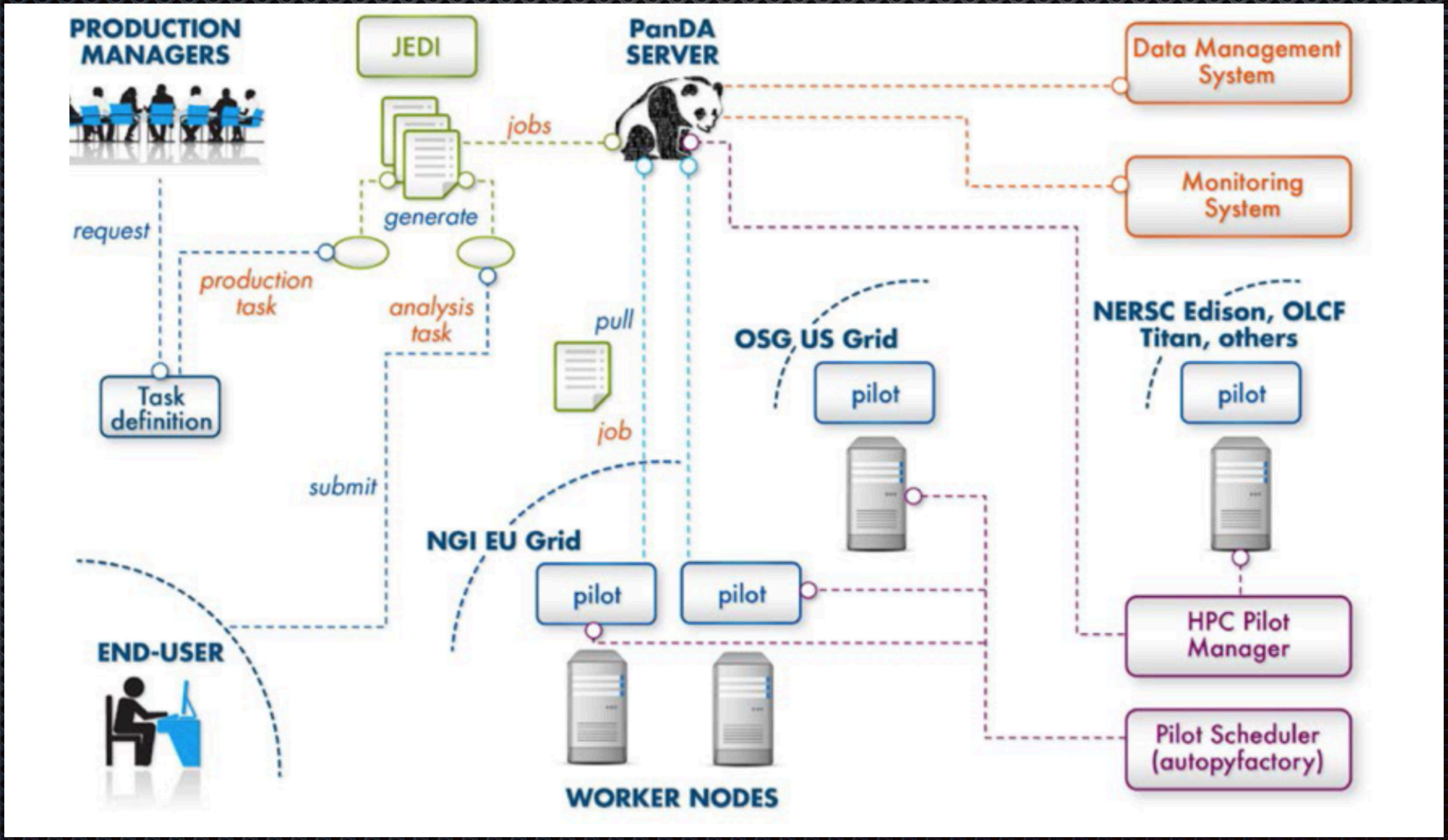
- **2005**: Initiated for US ATLAS (BNL and UTA)
- 2006: Support for analysis
- 2008: Adopted ATLAS-wide
- 2009: First use beyond ATLAS
- 2011: Dynamic data caching based on usage and demand
- 2012: ASCR/HEP BigPanDA project
- 2014: Network-aware brokerage
- 2014: Job Execution and Definition I/F (JEDI) adds complex task management and fine grained dynamic job management
- 2014: JEDI-based Event Service
- 2014: megaPanDA project supported by RF Ministry of Science and Education
- 2015: New ATLAS Production System, based on PanDA/JEDI
- 2015: Manage Heterogeneous Computing Resources
- 2016: DOE ASCR BigPanDA@Titan project
- 2016: PanDA for bioinformatics
- **2016**: COMPASS adopted PanDA (JINR, CERN, NRC-KI), PanDA beyond HEP : LSST, BlueBrain





PanDA

- Pilot based job execution system
  - Pilot manages job execution on local resources, as well as data movement for the job
- Payload is sent only after pilot execution begins on Compute Element
  - Minimize latency, reduce error rates



Motivation for new PanDA Pilot

- Some of the Pilot 1.0 code base is getting a bit too old and is difficult to maintain
  - Refactoring is a slow process that has already been going on for years and does not always have highest priority
  - More manpower made available to alleviate a steady increase of feature requests
  - New features/workflows are often challenging to implement/support
- "Complete" rewrite
  - Keeping some recent new developments (not cut-and-paste)
  - Getting rid of all legacy code and outdated mechanisms
  - Rethink of basic pilot flow
- New PanDA Pilot Project launched in April 2016
  - Project to span the next few years
  - Development and support of the old pilot ("Pilot 1.0") will continue as it remains the production pilot until new pilot ("Pilot 2.0") is ready

Motivation for Harvester

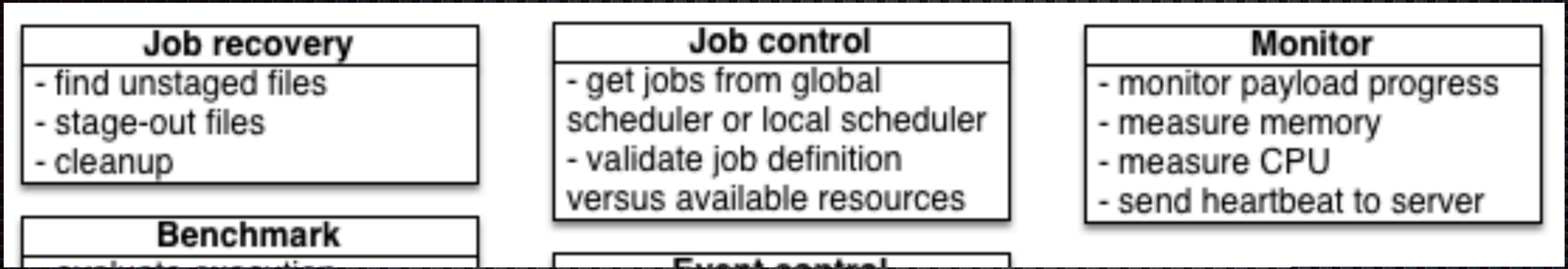
- PanDA currently relies on server-pilot paradigm
  - PanDA server maintains state and manages workflows with various granularities, such as task, job, and event – Pilots are job-centric and independently run on worker nodes with limited view of local resource
- Works well for the grid with 250k cores 24x7 as underlying resources are not very heterogeneous
  - But missing capability to dynamically optimize resource allocation among differences of architectures (limitations by number of cores, amount of RAM per core, limitations of wall time etc.)
- Not very well for HPC or large-scale clouds
  - Each HPC has a different edge service and operational policy, leading to over-stretched pilot architecture and incoherence in implementation at different HPCs
  - PanDA itself has no means of managing and monitoring cloud utilization by using native cloud API which is far more optimal than that of an intermediate service like condor

Motivation for Harvester

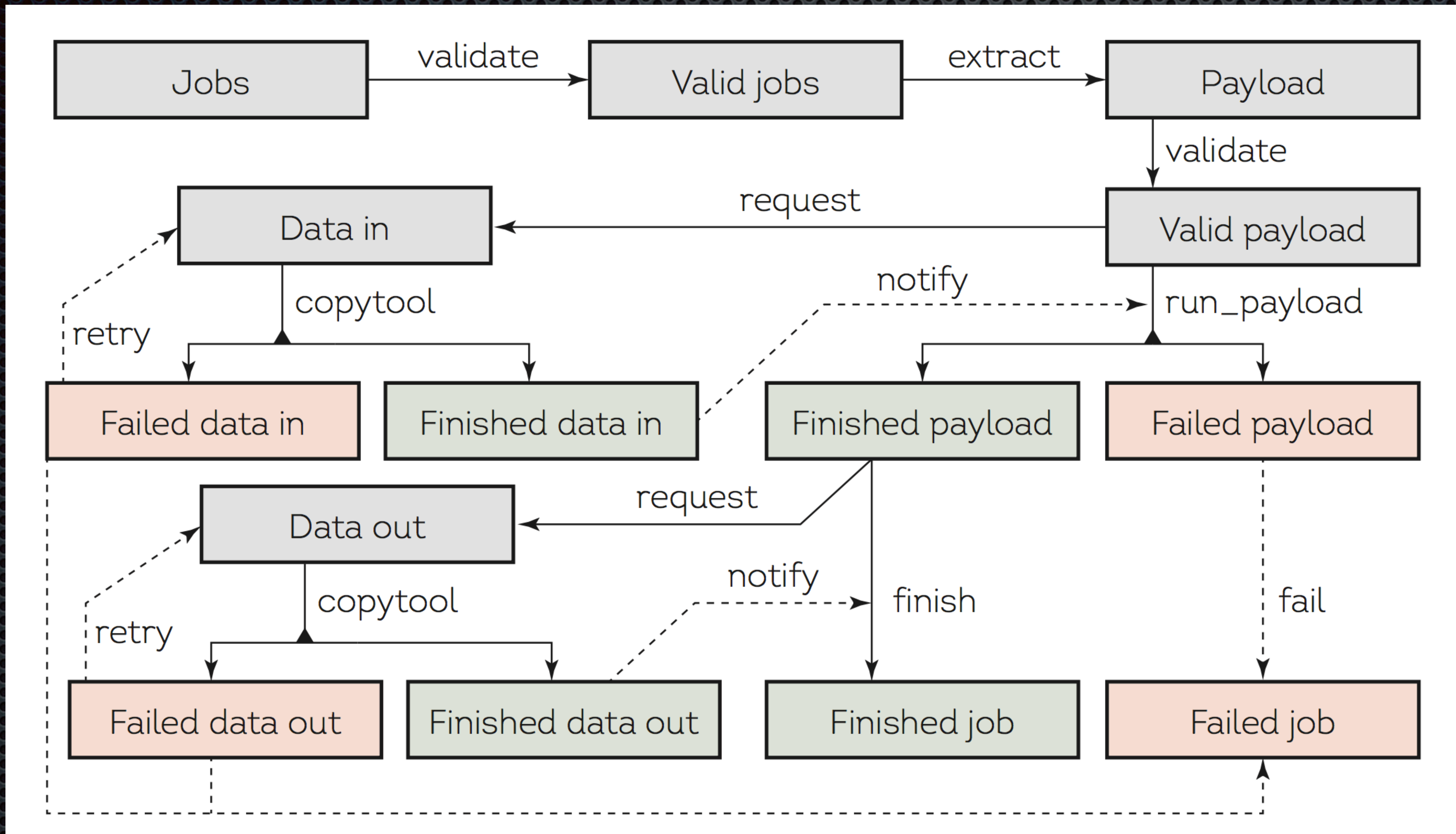
- New model : server-harvester-pilot
  - Harvester is a resource-facing service between PanDA server and collection of pilots
  - Stateless service with knowledge of resource
  - Modular design for different resource types
  - Many harvester instances running in parallel
  - To provide a single view of a large or uniform resource that optimizes pilot and/or workload management
  - To provide a commonality layer in bringing coherence to HPC implementations
  - Better integration with PanDA system for various (new) workflows, such as job/event-level late-binding and jumbo jobs

Pilot 2.0 Key features

Component model



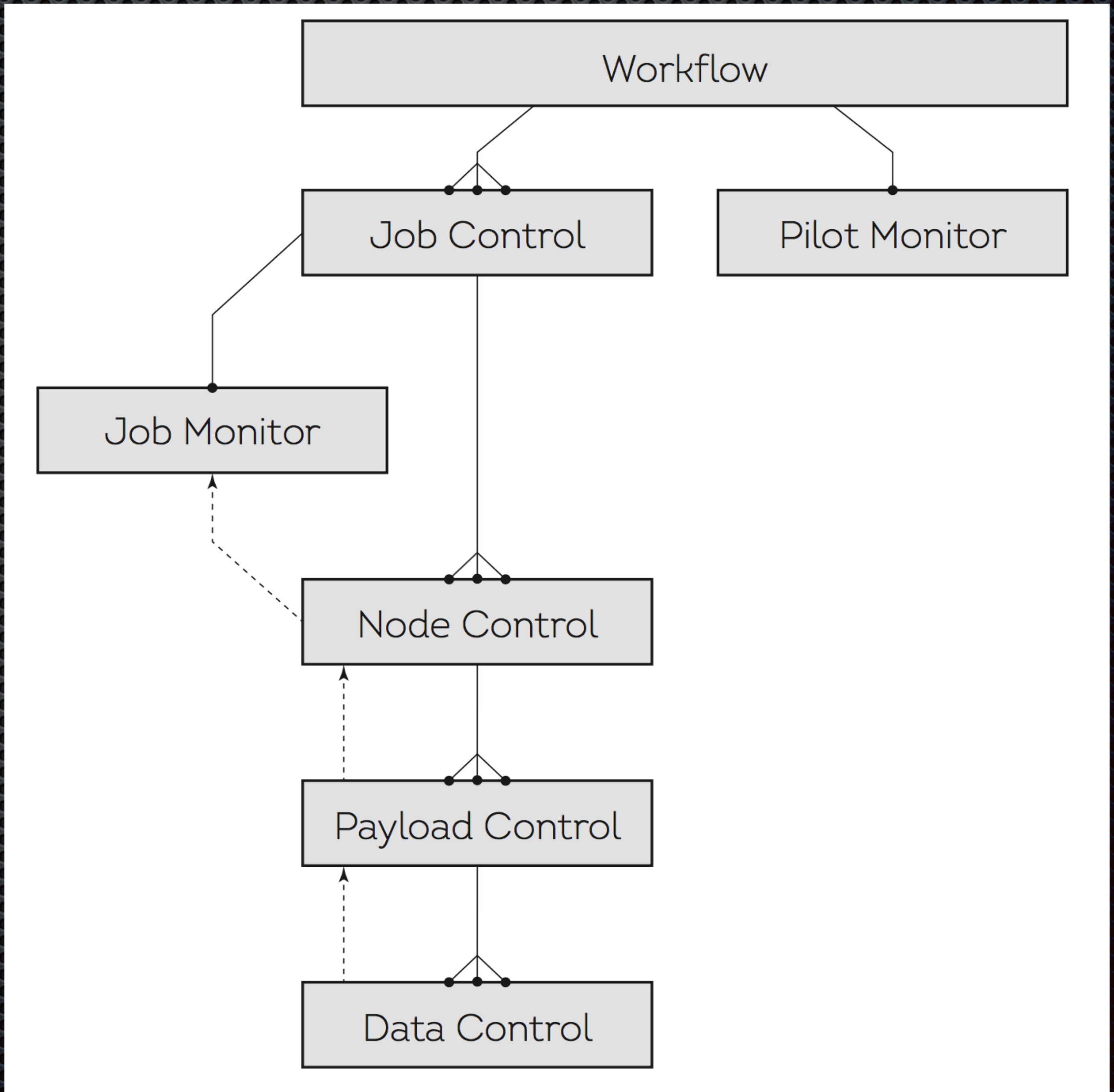
Internal Flow of the Jobs Objects



- Job objects are kept in a job queue and are handled by the different pilot components

Component model update

- Extended monitoring
  - Pilot monitoring of internal threads
  - Job monitor
    - Thread lives and dies with payload
      - *Heartbeats*
      - *Size measurements*
      - *Looping jobs*
      - *Proxy lifetime*
      - *Pilot running time*
      - *User ("experiment") specific services*
        - *Benchmark reports*
        - *Memory monitoring*



Pilot 2 APIs

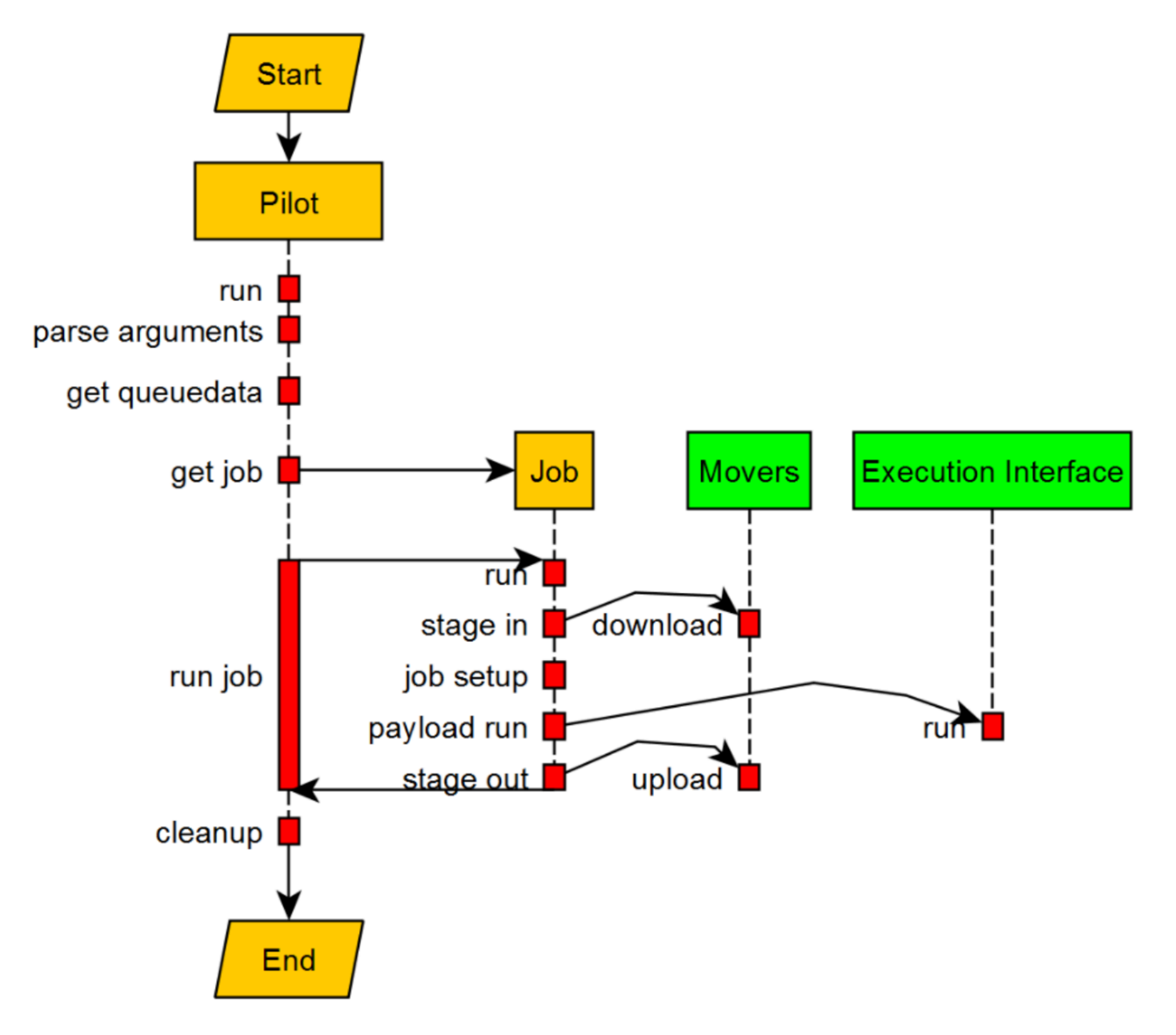
- Some Pilot functionality is exposed to external users by APIs; currently being planned for, or is already available
  - *Data API*
    - *Basic stage-in/out already used by Harvester*
    - *New request: asynchronous stage-in/out*
  - Communicator API
    - Functions for communicating with PanDA server, Harvester
    - API defined; contains functions for downloads/updates of job sand event ranges
  - Environment API
    - Interface to the job execution environment on HPCs
  - Services API
    - Possible new API which could expose functionalities related to services run by the pilot (being discussed), see later slides
  - Container API

Utilities

- **Pilot 1 has hundreds of major and minor functions**
  - A large part of Pilot 2 development is to re-implement many of these
- **Pilot 2 has utilities organized in dedicated util/ folder**
  - Current code base include functions in multiple modules
    - E.g. constants, disk, filehandling, https, information, ..
    - Preliminary information module presents interface to AGIS and schedconfig
      - To be replaced by a full Information Service component (where AGIS/ schedconfig are not hardcoded but accessed via user code)
      - Development to start as soon as possible
  - Pilot 2 now supports standard configuration files
    - Config files are shipped with pilot source (default values), but can be preplaced either in /etc or in init directory

MiniPilot

- A minimal pilot has been developed by Daniel Drizhuk (Kurchatov Inst.)
  - To be used by the developers primarily during the initial development and testing stage
  - For module and component testing
  - Can eventually result in a SimplePilot for external use / starting point for new PanDA users
- Documentation/instructions in GitHub – https://github.com/PanDAWMS/pilot-2.0/tree/dev/lib/minipilot
- Easy to use by design
- Using proper/standard [python] logging
- Following coding conventions
  - Enforced by testing framework



Code validation and documentation

- Pilot 2 GitHub is using TravisCI for automatic code verification/validation and unit tests
  - GitHub pull request into Pilot 2 repo triggers external service (runs pep8, flake8 and unit tests)
- Semi-automatic code documentation using Sphinx
  - Module to be documented must be accompanied by related sphinx file
  - Pull request followed by [currently] local sphinx script execution which builds the documentation
  - Output needs to be moved to www server
- Investigating of possibility of hosting of documentation in GitHub domain

Harvester for HPC

Harvester design key points

- **Lightweight**
  - To run on logon/edge nodes at HPC centres
- **Stateless for scalability + central database (oracle) + local database (sqlite3)**
  - Capability to rebuild the local database from the central database for auto restart
  - Local database to reduce redundant access to the central database
  - Only important checkpoints are propagated to the central database
- **Installation with or without root privilege**
- **Configurability**
  - To customize workflow for each resource
  - To turn on/off components with various plugins

Harvester design key points

- **Running on top of pilot API**
  - Core + plugins + resource specifics in resource managers or pilot components
  - Leveraging development effort for the pilot consistently with the evolution plan (pilot 2.0)
- **Direct bi-directional communication with PanDA**
  - Requesting workload to PanDA based on dynamic resource availability information and static configuration
  - Receiving commands directly from PanDA to throttle or boost the number of workers (worker = pilot, MPI job, or VM)

Constraints for Workload Management on HPC

- **Preemptable or very short walltime limit**
  - To shorten the execution time of jobs
    - Decreasing the number of events per job, and/or
    - Increasing the number of CPU cores per job
  - Or to enable event-level bookkeeping (event service)
- **Limitation on number of concurrent workers in the batch system**
  - To increase the number of CPU cores per worker
    - Combining multiple jobs to a single payload which is given to a worker (multi-job or ManyToOne)
    - Increasing the number of events per job (jumbo job)
- **No outbound network connectivities on compute nodes**
  - Edge service on edge node to mediate communication between PandDA and workers
- **Long waiting time in the batch queue**
  - To assign only low priority jobs
  - Or to enable parallel event consumption on pledged resources

Constraints for Workload Management on HPC

- **Intermittent and/or spiky resource availability**
  - To send "fake" pilot requests from edge service (get_job requests for job pre-fetching or update_job requests for jobs in stating state)
  - Or to request jobs before resources become available (proactive workload assignment)

Workflows 1/4 : Push+True Pilot

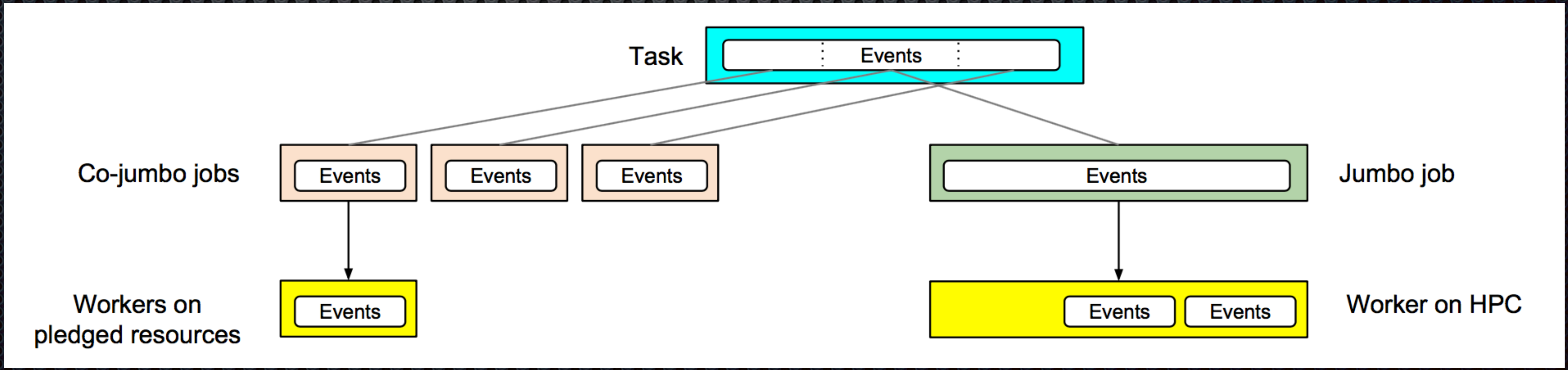
- **Prefetches jobs, submits workers(pilots)+jobs to the batch system, and lets workers communicate with panda once they get CPUs**
- **Advantages**
  - Easy to send get_job requests without empty workers to attract jobs before the resource becomes available
    - A pool of prefetched jobs as a buffer for fluctuated CPU availability
    - Automatic throttling of worker submission in case of no jobs
  - A well matured workflow in ATLAS as it has been used for some grid sites for a long time
- **Caveats**
  - Requires less restrictive operation policy
    - Outbound network connection on compute nodes, many batch workers running in parallel, long walltime limit with allocation
  - High priority jobs cannot get the first available CPUs

Workflows 2/4 : ManyToOne

- **Prefetches multiple jobs, combines them into a single payload, and submits the payload to the batch system**
- **No MPI : one job per rank/node**
- **Essentially the same as “multi-job pilot”**
  - One major difference is that jobs are prefetched and input files are asynchronously pre-staged before CPU slots become available, while multi-job pilot fetches jobs and stages input files once free CPU slots are found
- **Advantage**
  - The number of concurrent workers in the batch system can be reduced
- **Caveats**
  - Needs jobs with similar execution time so that all jobs in the same worker finish simultaneously to avoid having idle nodes
    - E.g., jobs from the same task or request. Cannot accept jobs from random tasks → Custom tasks
  - Or needs to enable event service
    - When the first job finishes all the rest could be killed

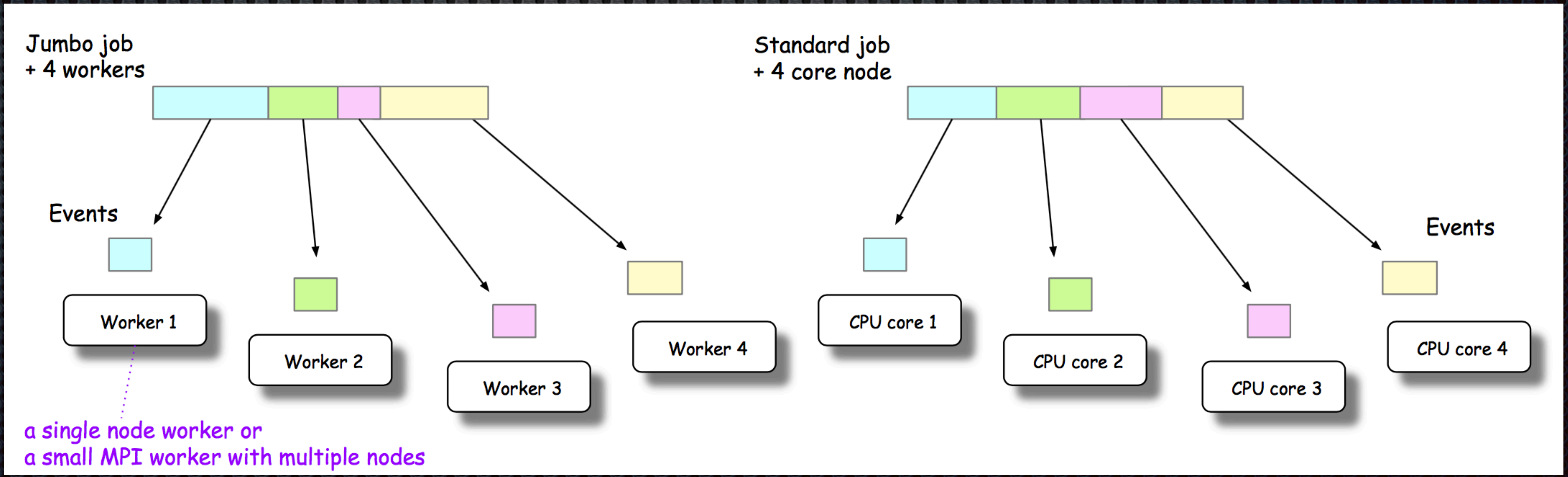
Workflows 3/4 : Jumbo Jobs

- **One single huge event set (jumbo job) including all events from one task**
  - A huge event set + event-level bookkeeping allows a big batch worker to process events at HPCs as much as possible
  - Multiple jumbo jobs per task to be assigned to different HPCs
  - Don't have to estimate optimal event sizes for each HPC
- **The huge event set is partitioned at the same time to small event sets (co-jumbo jobs)**
  - They are good to be processed by small batch workers at pledged resources
- **Workers for jumbo and co-jumbo jobs compete to grab events**
  - Each event is exclusively processed by one worker
  - Events are being consumed at pledged resources even if big workers are waiting in long HPC batch queues



Workflows 4/4 : Multi Workers

- **Many workers contributing to the same job**
- **Typical use-case : Jumbo jobs + small workers**
  - Single node workers
  - Small MPI workers with backfill mode
- **Job and file records for each jumbo job are huge in the database**
  - Not good to have one jumbo job for each small worker
- **One standard job is processed by many CPU cores → One MPI job is processed by many compute nodes → One jumbo job could be processed by many workers**
  - Workers don't have to pop-up simultaneously → Workload sharing with asynchronous workers without node-boundaries



Current status of Harvester on HPC

- Core components of Harvester were deployed on major US HPC facilities: ALCF, OLCF, NERSC
  - Each facility has its own policies of usage and stack of services and middleware
  - Ongoing process for development of specific plugins and tuning of workflows

Conclusions

- PanDA has performed well for ATLAS in the last decade including the LHC Run 1 and Run 2 data taking periods
- New challenges to come while steadily running for LHC Run 2
- New components and features have been adressed and implementation with continous integration in progress