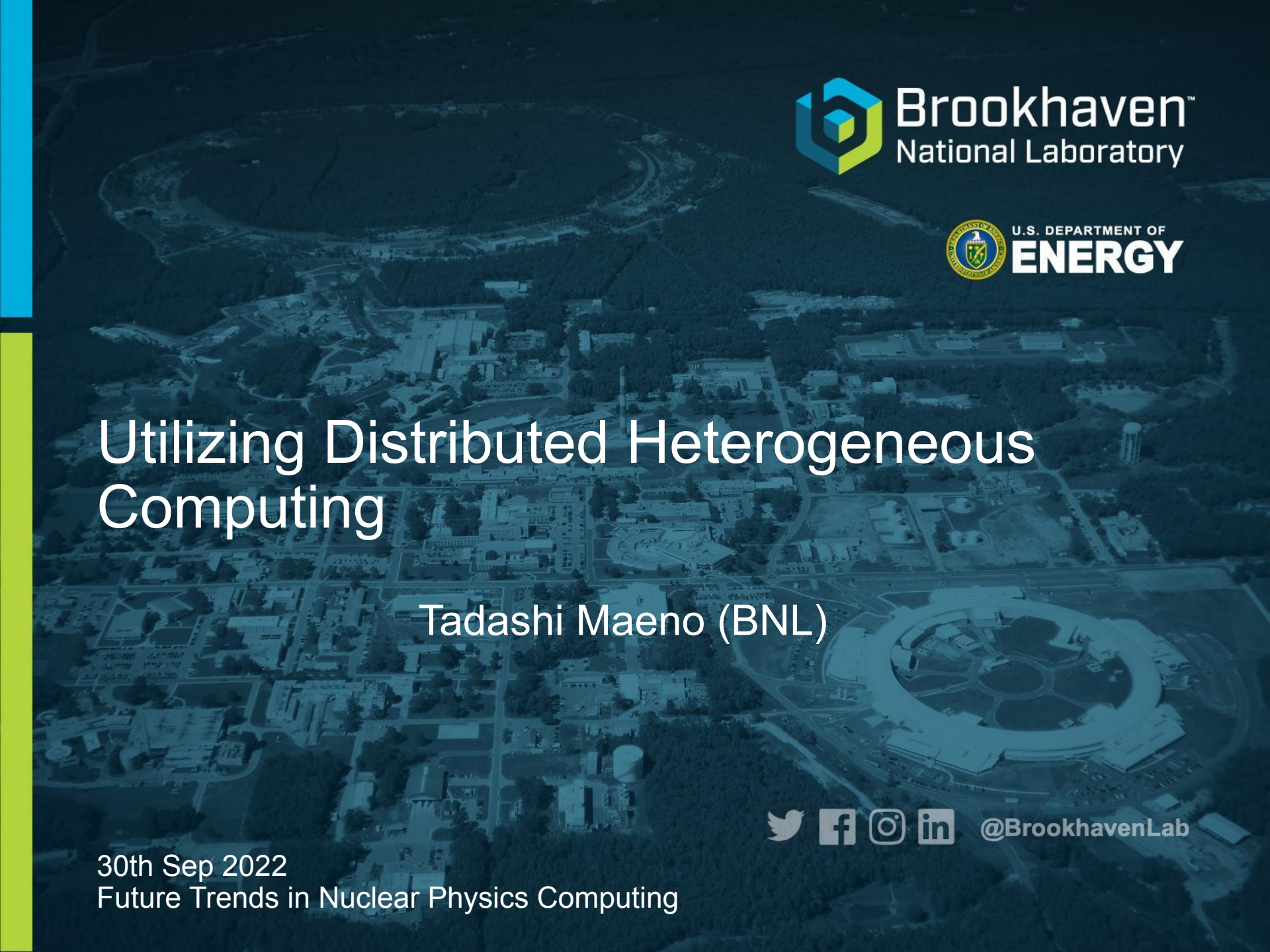


Utilizing Distributed Heterogeneous Computing

Tadashi Maeno (BNL)

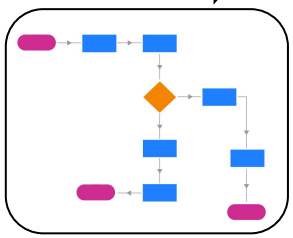
30th Sep 2022
Future Trends in Nuclear Physics Computing

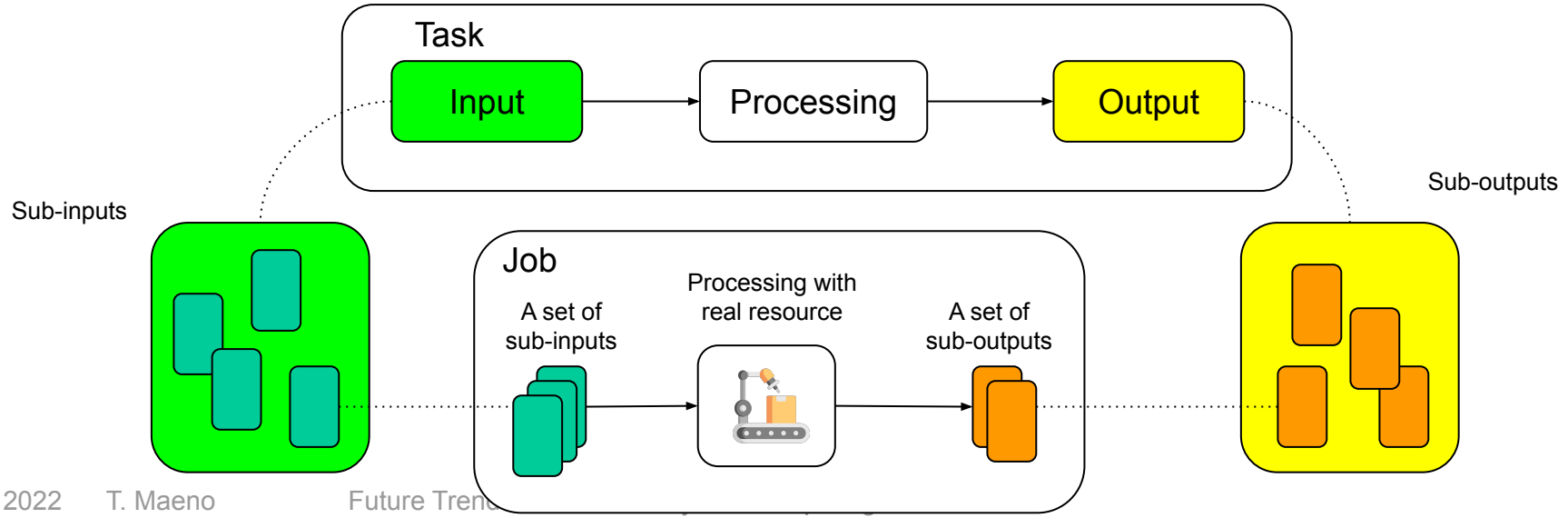
Needs for Distributed Heterogeneous Computing

- Most users love local resources, but have to go to remote providers when enough or suitable resources/services are locally unavailable
- A zoo of resource/service providers distributed worldwide with various benefits and constraints
  - The grid, commercial cloud service providers, High-performance computing (HPC) and Leading Computing Facilities (LCFs), volunteer computing, Platform-as-a-Service (PaaS), Function-as-a-Service (FaaS), …
- Complex and emerging workflows
  - Various resource/service requirements even in a single workflow
  - To leverage optimal services for a part of the workflow
- Distributed = Geographically distributed != Multi-node/process

Terminology: Workload Entities

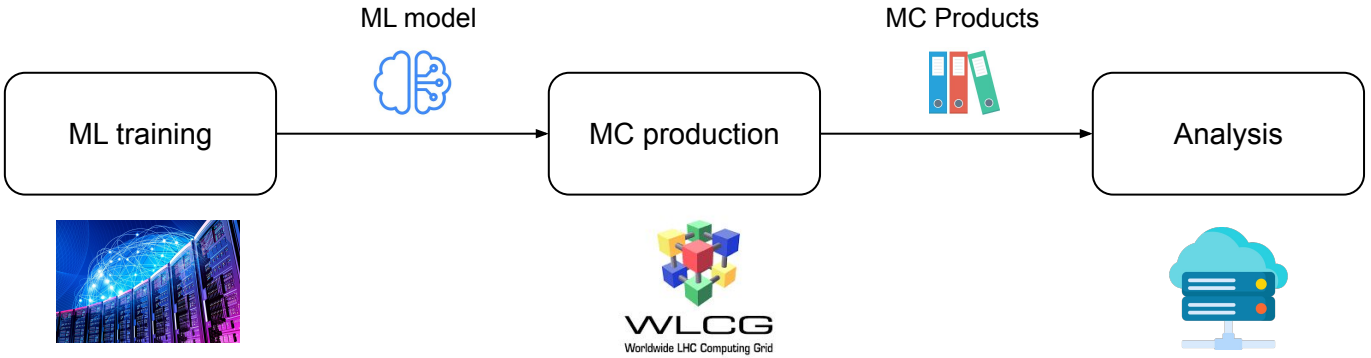
- A workflow: a top-level entity of workload to accomplish a scientific objective
- A task: a step in a workflow
  - A group of tasks = a workflow
  - A task processes input to produce output
- Input: a collection of the minimum processing quanta
  - Each quantum can be processed independently with the finest processing granularity
  - Generally it is a collection of sub-inputs, e.g. files
  - Each sub-input contains a subset of processing quanta
- A job: an artificial partition in a task
  - A group of jobs = a task
  - A job processes a set of sub-inputs to produce a set of sub-outputs using real resources





A Simple Usecase with Multiple Remote Providers 1/2

- A user has a workflow to perform analysis on Monte-Carlo (MC) samples produced with a Machine Learning (ML) model
  - Three tasks in the workflow
    - ML training, MC production, and Analysis
    - Each task could have different resource/service requirements
- The user happens to have
  - Allocation at a LCF where huge GPU resources are available
  - Approval from the experiment collaboration to use the grid
  - Credits for an analysis platform on a cloud service
- The user decided to run ML training at LCF, MC production on the grid, and Analysis on the cloud service



A Simple Usecase with Multiple Remote Providers 2/2

- Procedures
  - ML training
    - Logins to the edge node via two factor authentication at the LCF
    - Copies training datasets to local NFS
    - Deploys software on NFS
    - Submits ML training jobs to HPC's batch system
      - Job descriptions in batch scripts
    - Gets the best ML model
  - MC production
    - Configures jobs based on the ML model
    - Places input data on grid storages
    - Deploys software on CVMFS
    - Submits jobs through grid computing elements
      - Job descriptions in JDL
  - Analysis
    - Transfers MC products to somewhere reachable from the cloud service
    - Containerize analysis
    - Submits analysis jobs to the cloud service through the service API
      - Job descriptions in yaml or another language
    - Gets the final results

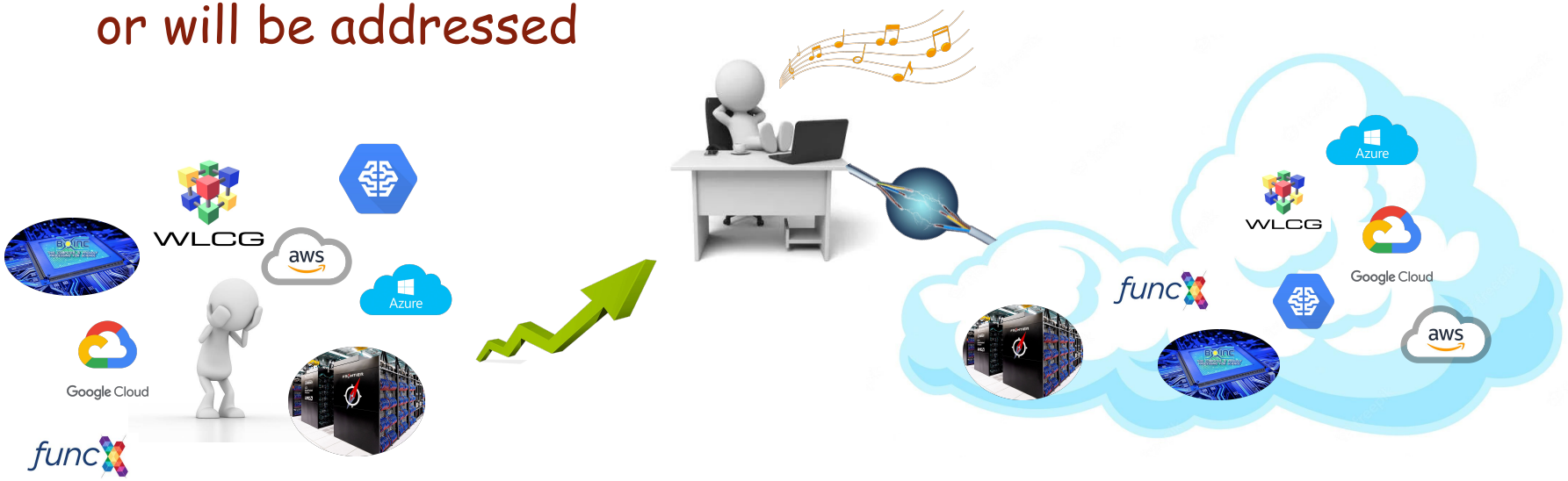
Dealing with Multiple Remote Providers

- The obvious advantage is availabilities of more resources and services
- However, workflows wouldn't become N times faster even if N providers are exploited concurrently
- Many things to consider
  - Economic, development, operational, environmental costs
  - Allocations
  - Parallelizability and interdependence in workflows
  - Automation
  - Identity management
  - Resource requirements
  - Software deployment
  - Data placement and aggregation
  - Constraints, benefits, and downtime of each provider
  - …

Goals of Distributed Heterogeneous Computing

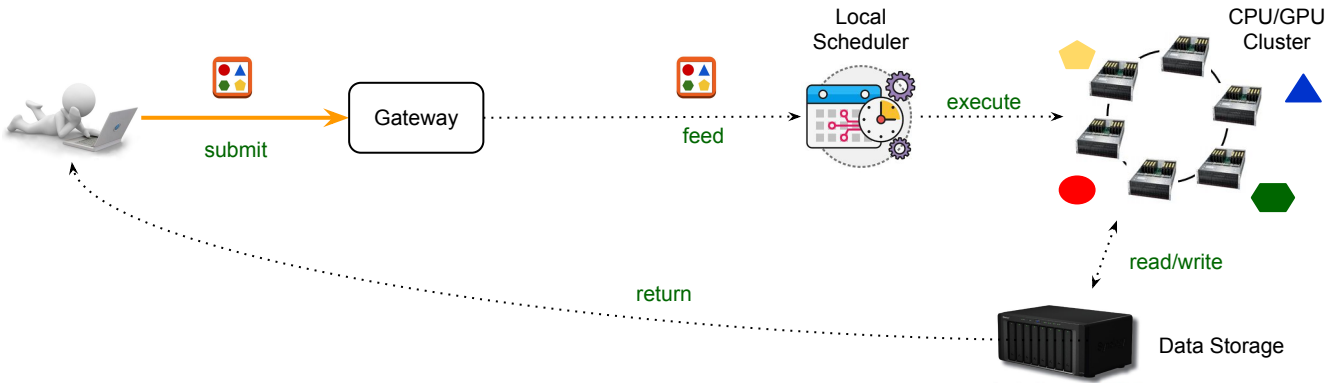
- The ultimate goal is to hide details in distributed remote resources/services from users, but also drill-down transparency and diagnostics, and bring an easy and quasi-interactive interface with quick turnaround
  - Isolation of users from resources/services
  - Seamless workflow execution
  - Smart workload routing
  - Dynamics resource provisioning and cost saving
  - Automatic data placement and software deployment

Will showcase in next slides how those issues have been or will be addressed



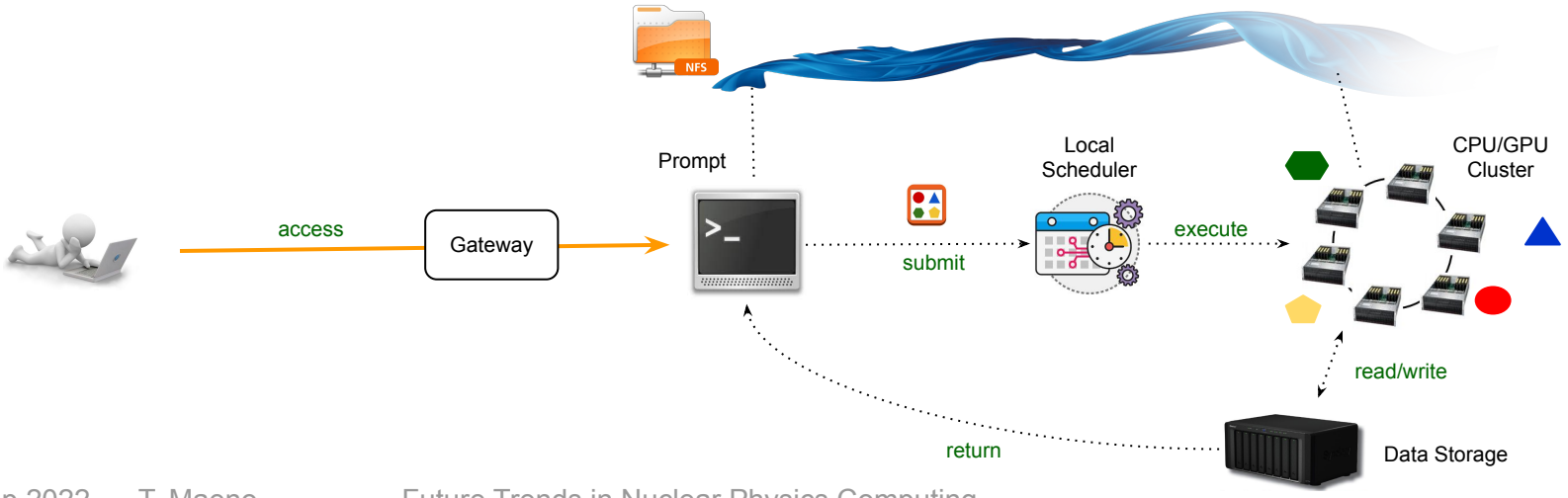
Hiding Remote Resource Access 1/3

- Batch-like access to remote resources
  - The user describes a workload in a file
  - A remote gateway service authenticates the user to receive the file from the user
  - The gateway service feeds the file to a workload scheduler to process the workload on the computing resources behind
  - Outputs are delivered to the user somehow
- Gateway service: HTCondor Computing Element (CE), ARC CE, Kubernetes API service, …
- The user doesn't have direct access to the remote resources



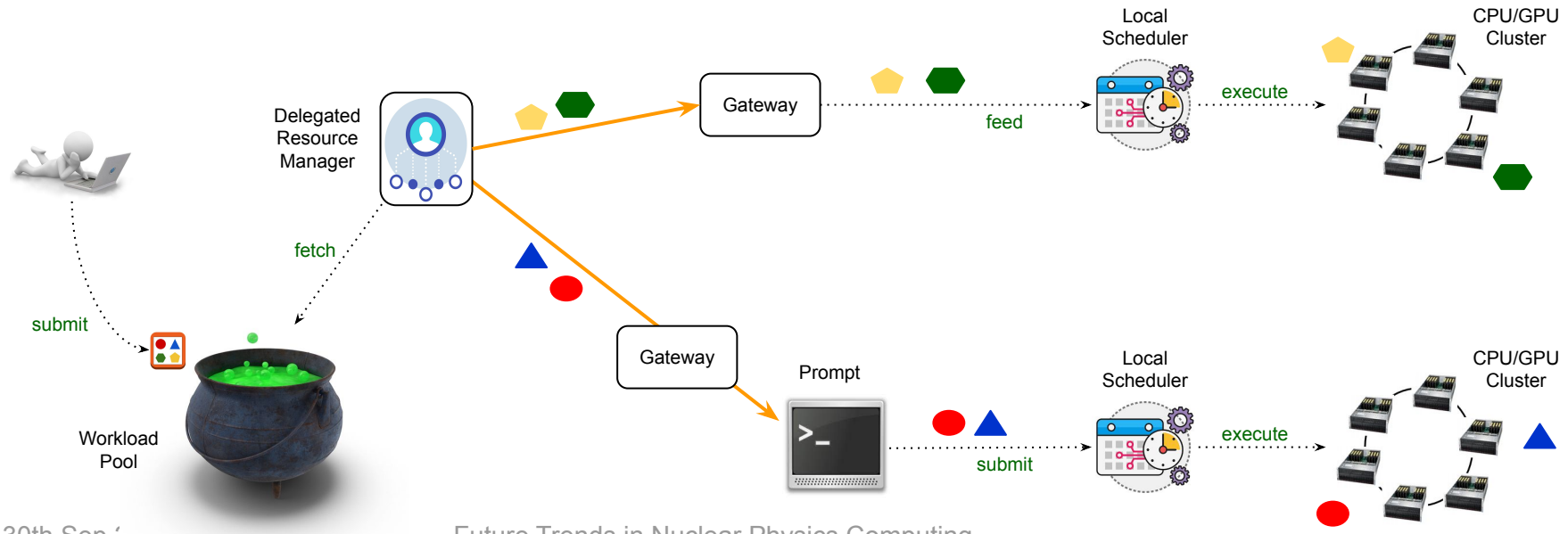
Hiding Remote Resource Access 2/3

- Interactive access to remote resources
  - A remote gateway service authenticates the user to give an interactive prompt
  - The user submits a workload to a workload scheduler from the prompt
  - The user sees outputs in data storage from the prompt
- Gateway service: sshd, jupyter hub, AI platforms, …
- Most providers give interactive access to the workload schedulers instead of computing resources
  - The user may communicate with the computing resources through NFS from the prompt, which is indirect but still quite convenient



Hiding Remote Resource Access 3/3

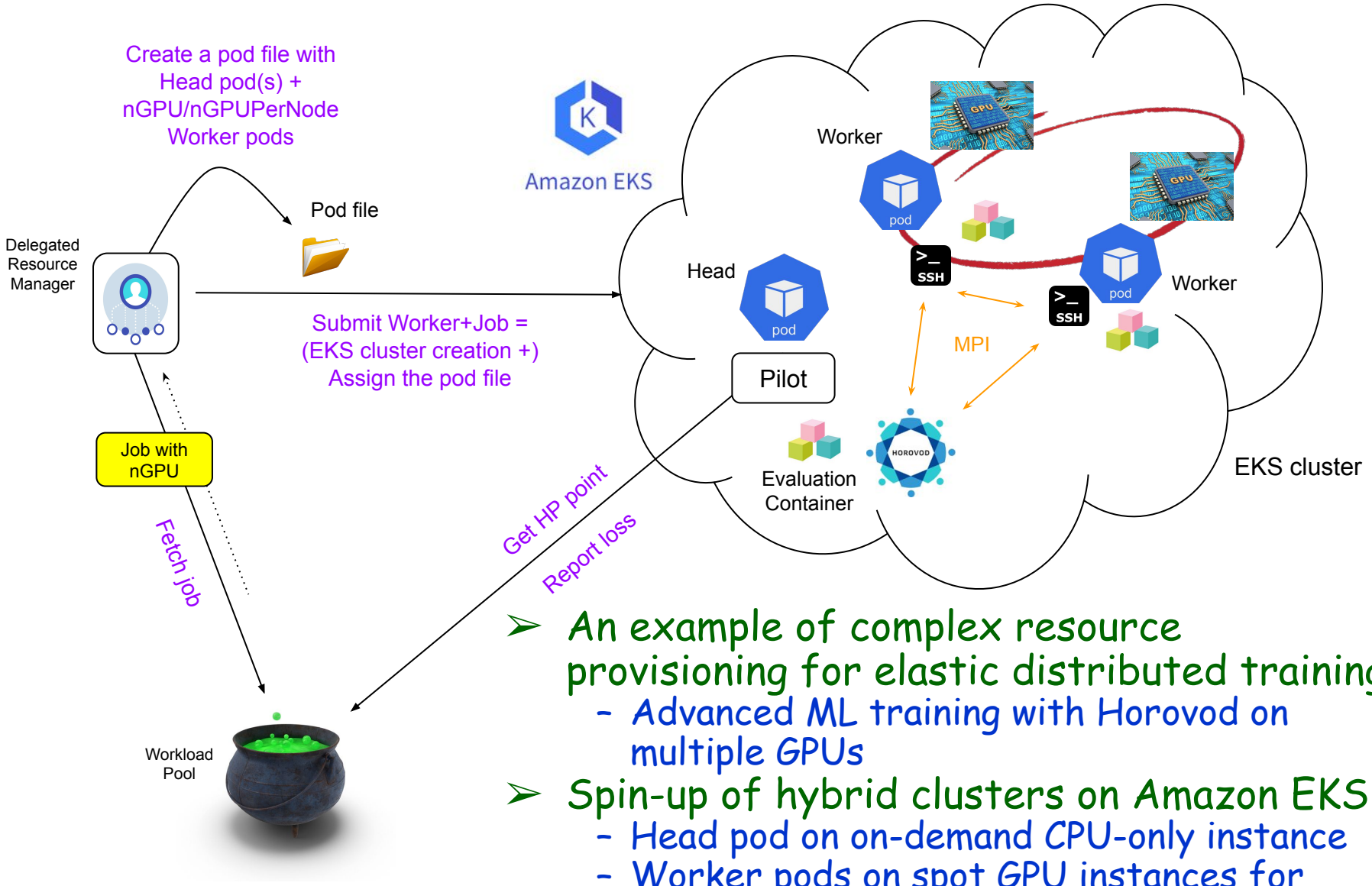
- Central workload pool + delegated resource access
  - The user submits workflows to the central workload pool where workflows are decomposed to tasks and jobs
  - The delegated resource manager accesses to resources on behalf of users using common or user's credentials
  - Small agents, a.k.a. pilots, may be scheduled instead of jobs and they may fetch jobs from the pool if the scheme is applicable
- Advantages
  - Isolation between the user and resources
  - Centrally managed fair-share and priorities among multiple users
  - Workload routing based on fine-grained requirements and central knowledge of resource/service characteristics and availabilities



Schedulers in Resource/Service Providers

- Traditional batch systems
  - HTCondor, Slurm, Pbs, Torque, …
  - Many academic institutes including LCFs
- Kubernetes-based schedulers
  - Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes Service (EKS), Azure Kubernetes Service (AKS), …
  - De-facto standard available in many cloud services
  - The entire job or the entrypoint of the job to be containerized
  - Considerable cost difference between spot and on-demand instances
- Multi-node software
  - Dask, Horovod, Ray, …
  - Application-level resource scheduling
- PaaS and FaaS
  - Google AI, Amazon ML service, REANA, funcX, ServiceX, …
  - Platforms optimized for specific analysis workflows
    - Very powerful for particular usecases
    - Not for all types of analysis, not straightforward to port existing analysis

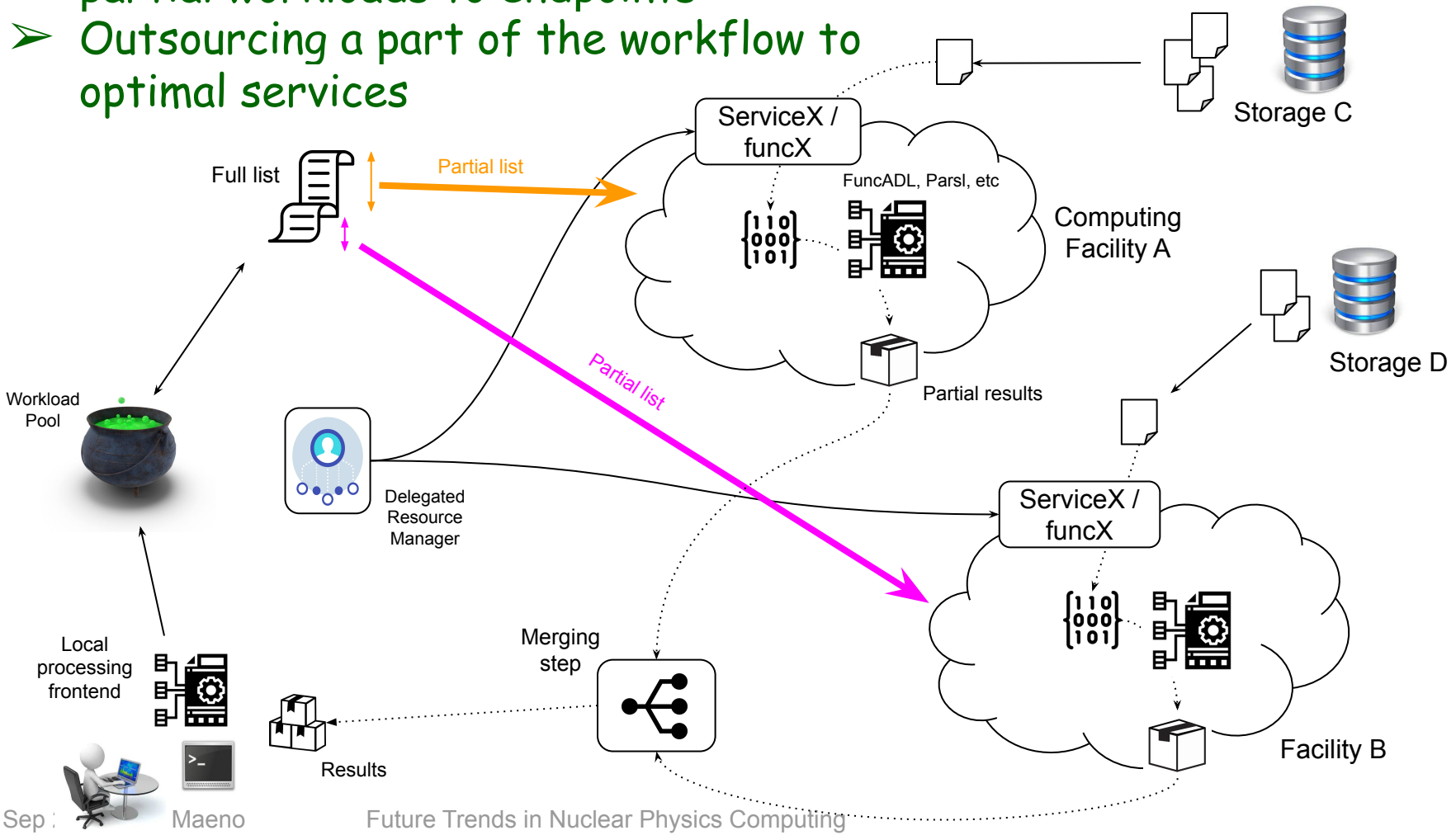
Complex Resource Provisioning



- An example of complex resource provisioning for elastic distributed training
  - Advanced ML training with Horovod on multiple GPUs
- Spin-up of hybrid clusters on Amazon EKS
  - Head pod on on-demand CPU-only instance
  - Worker pods on spot GPU instances for cost-saving + dynamic scaling-up/down of Horovod

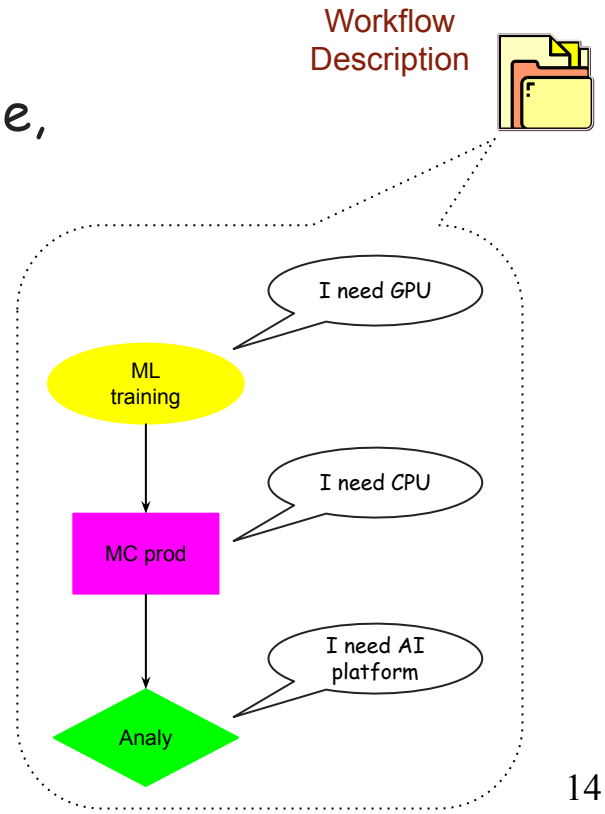
FaaS/PaaS Integration

- FaaS/PaaS as a distributed endpoint
  - Local service at each computing facility, i.e., local to the computing facility, sorting out compliance with local security and operational policies
- Central bookkeeping of processing on the full list to dispatch partial workloads to endpoints
- Outsourcing a part of the workflow to optimal services



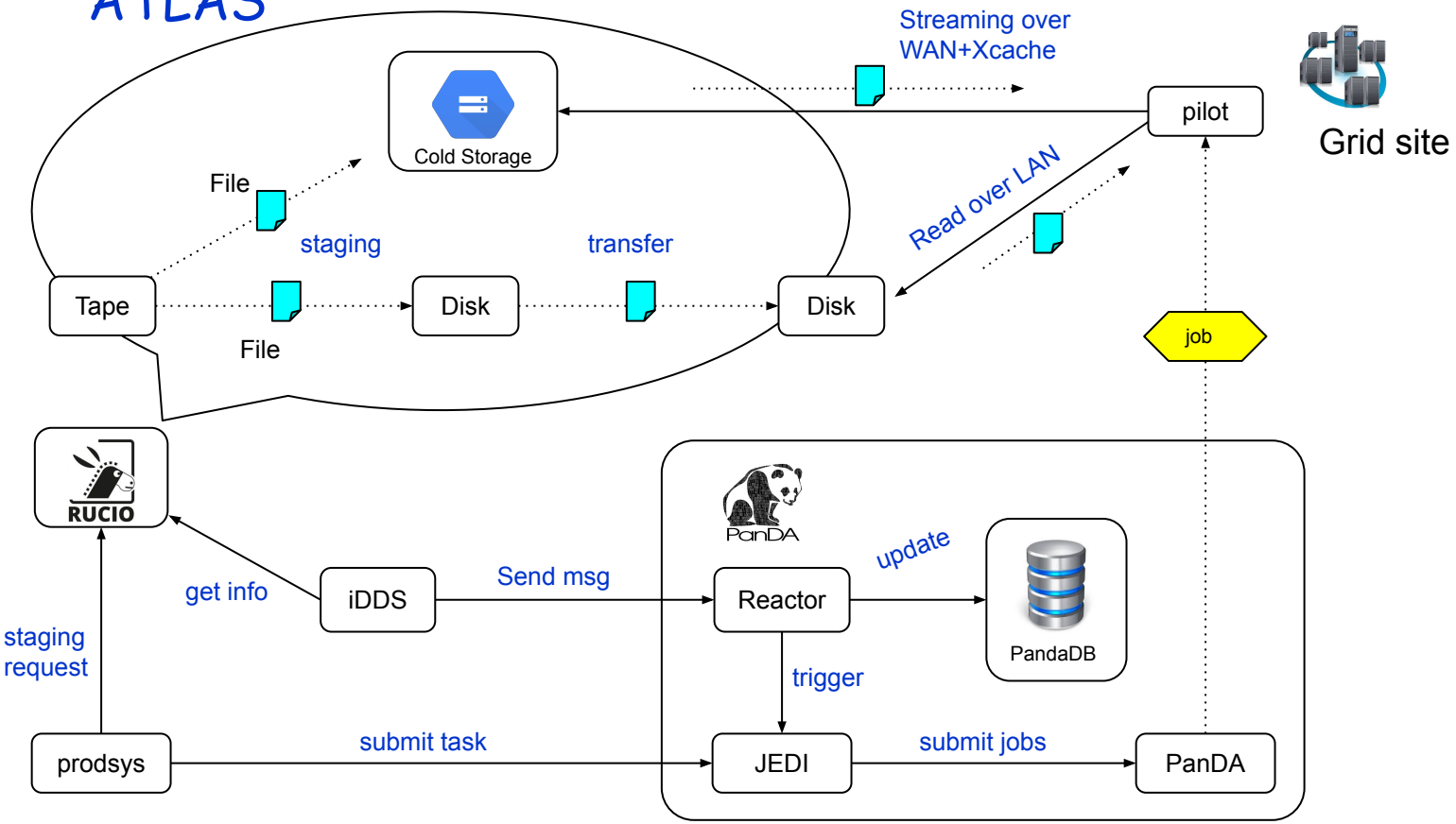
Workload Routing

- Resource/service requirements for each task in the workflow description
- Selection of providers based on matching between
  - Resource/service requirements
  - Published and real-time information of providers
    - Hardware (HW) spec
    - Service description
    - The number slots currently available
    - Data locality
    - Allocation
    - To consider also costs, electricity price, carbon footprint, …
- The user may wrongly specify resource requirements
  - E.g. too large/small RAM requirements
  - Dynamic correction of the requirements based on actual resource usage in the first few jobs before processing all the rest
    Scouts → Avalanche



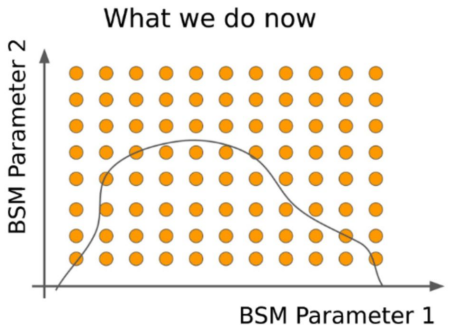
Automatic Data Placement

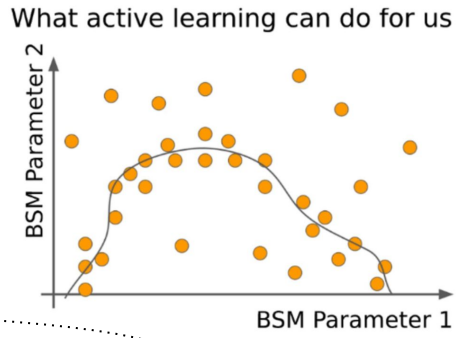
- Data placement is vital and must be fully automated in complex workflows
- E.g. Data carousel
  - Fine-grained disk-efficient tape staging
  - On-demand disk replicas and their aggressive deletion
  - Mitigating the disk problem that is the most crucial in ATLAS

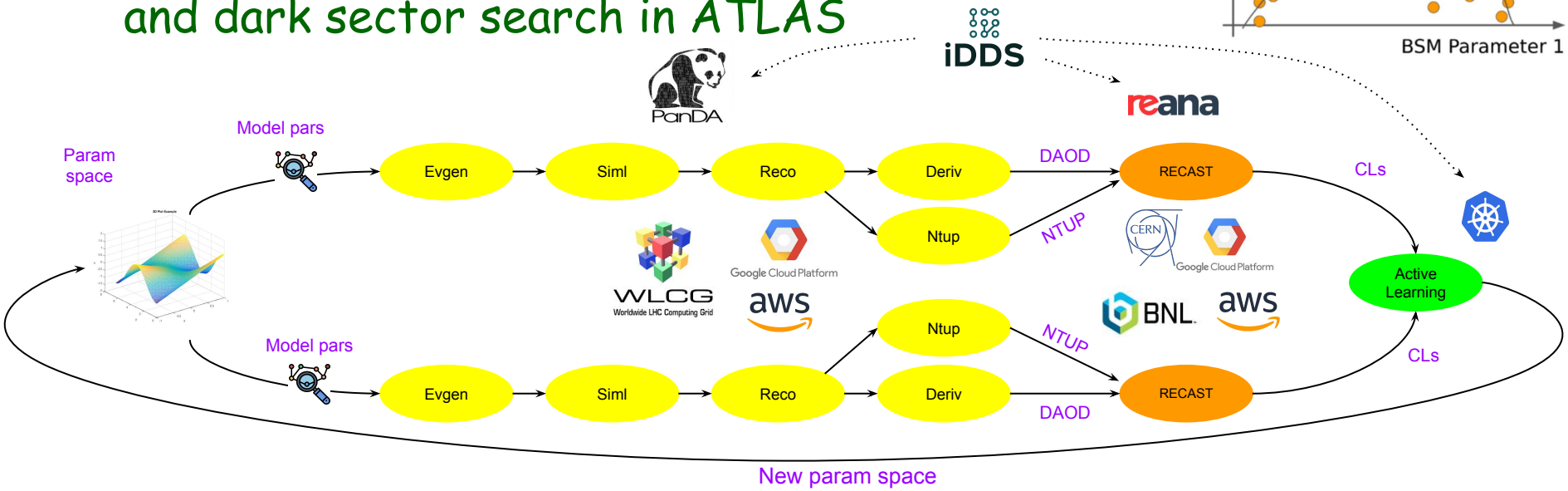


Seamless Execution of Advanced Workflows

- Active Learning (AL) using iterative regression on limit setting to increase analysis efficiency
  - Traditional: A brute-force grid/random search
  - New: A search session with multiple iterations
    - The search space for the next iteration is defined based on the results of old iterations
- A single workflow with multiple loops
  - Various tasks on different resource/service providers
  - No human intervention in transitions from one provider to another
- AL-based analysis for mono-Hbb exclusion limit and dark sector search in ATLAS





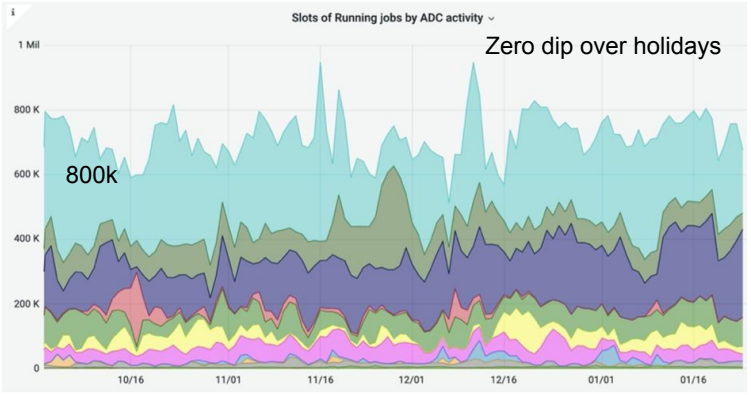


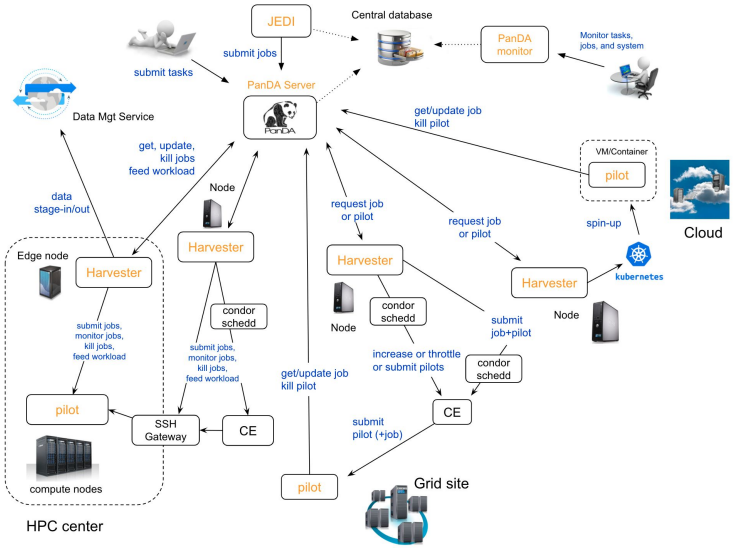
PanDA and Its Work Program For Distributed Heterogeneous Computing

Production and Distributed Analysis System

- PanDA: The workload management system
  - Manages 24x365 processing on ~800k concurrent cores globally for ATLAS, all workloads from evgen to analysis, all resource types, ~150 computing centers, ~1500 users, ~300M jobs/yr
  - Leveraging Rucio for fully-automated data management
  - Expanding to Vera C. Rubin (astrophysics) and sPHENIX (nuclear physics) beyond ATLAS (HEP)
- Harvester: In-house resource manager
  - Delegated resource/service access
  - Flexibility and extensibility around the plugin architecture
- iDDS: intelligent data delivery system
  - A joint project between ATLAS and IRIS-HEP
  - Supports arbitrarily complex fine-grained workflows defined via directed acyclic graphs (DAGs) and workflow description languages

Have addressed the issues described up to here, while still some issues yet to be addressed → Next slides





Workflow Description

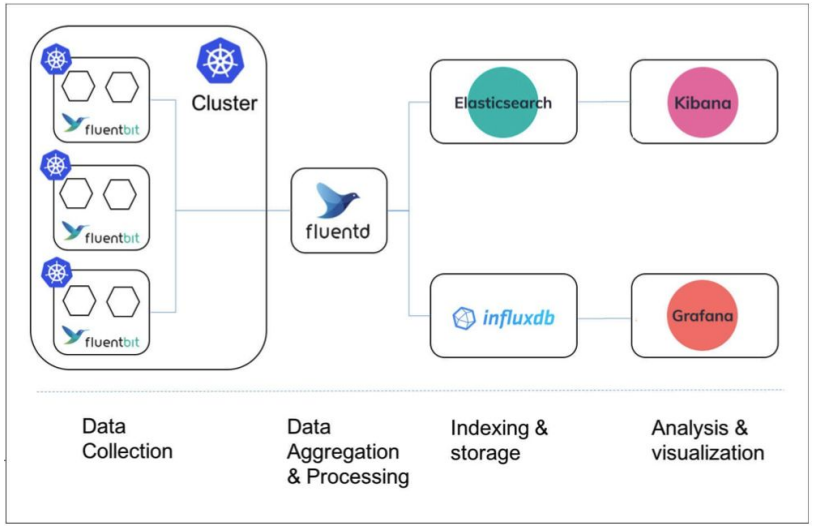
- Many workflow description languages are available on the market, but most of them are tightly coupled with the workflow processing backends, or cannot be easily enhanced to work with heterogeneous resources/services
- Quite tricky to describe complex workflows
  - E.g., no common language across ML training, MC production, and Analysis on a cloud platform
  - Requires sophisticated syntax to specify resource or service requirements translated to HW specs or service descriptions in various providers
  - The user has to explicitly specify parallelizable or critical sections and interdependency among tasks/jobs in the workflow, unless they are embarrassingly parallel and can be described declaratively
- Currently workflows are described in DAGs + yaml-based languages such as CWL
  - A bit error-prone
  - Python-based workflow description languages like airflow-sdk preferable
- GUI would also help to attract newbies

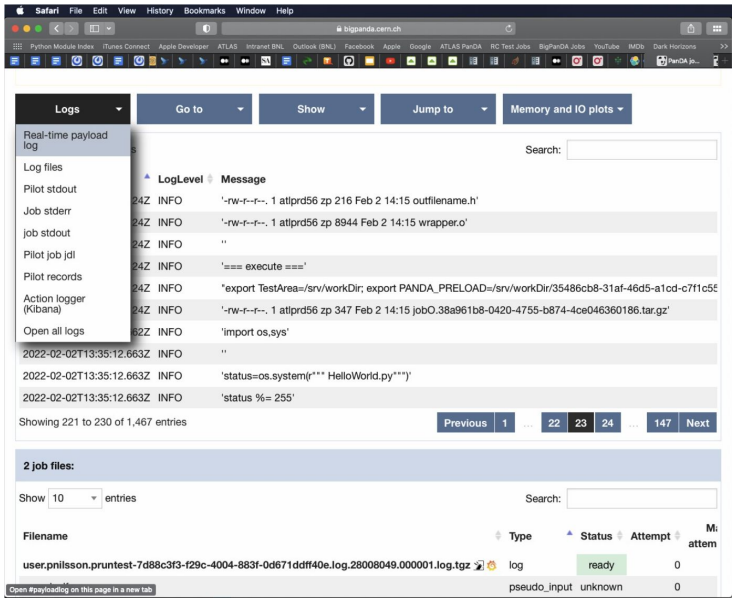
Latencies

- One of the most crucial obstacles for users to fully migrate from local resources
- Latencies are unavoidable due to intrinsic natures of remote resources
  - Underlying batch-like access
  - Data transfers
  - Queuing time in resource schedulers
  - Turnaround over the network
- Ongoing efforts to minimize latencies
  - Aggressive selection for optimal resources
  - Workload classification
  - Express shares
  - Removal of intermediate data transfers
  - Semi-persistent resource provisioning on dedicated resources

Streaming of Logs

- One of advantages to use local resources is easy and real-time access to logging information
  - Log files in job's work dir + tee, tail, grep, …
  - Not very sophisticated but quite efficient
- Similar level of convenience with remote providers
  - Collect tails of job's stdout periodically and bring them over the network in quasi real-time manner
- The first version available and to evolve
  - A unified look and feel
  - Scalability as well as usability





Conclusions

- The goal of distributed heterogeneous computing is to hide details in distributed remote resources/services from users, but also drill-down transparency and diagnostics, and bring an easy and quasi-interactive interface with quick turnaround
- Quite challenging due to a zoo of resource/service providers and complex and emerging workflows
  - Hard to support all possible combinations of resources, services, and workflows
  - Developments driven by real usecases
- PanDA and its work program for distributed heterogeneous computing
  - Many issues addressed
  - Versatility and flexibility for further evolutions
  - Ongoing significant efforts to attract more users