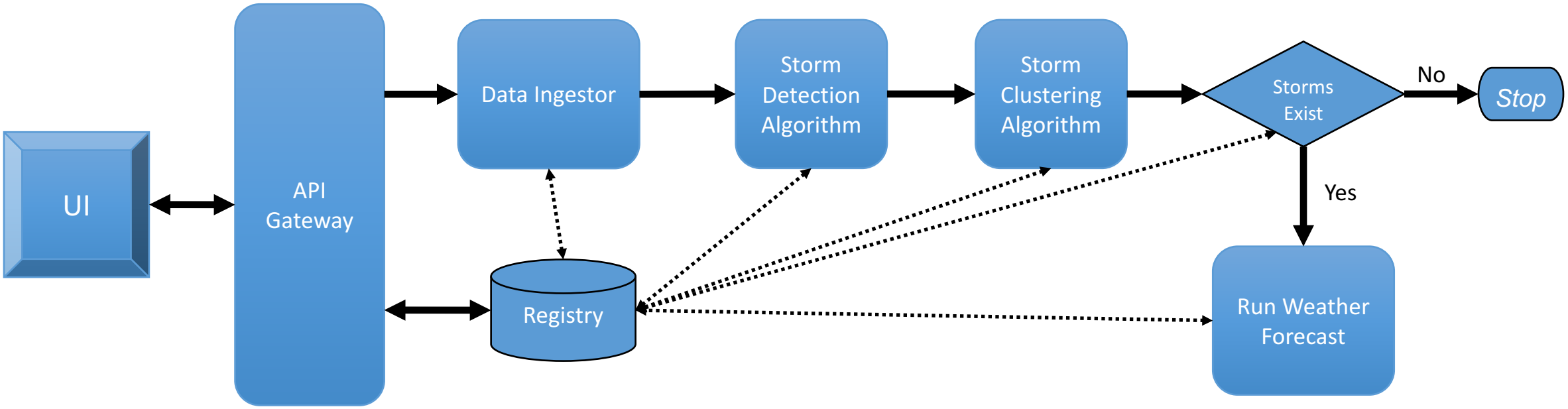
Your Microservice Layout



Many of these steps are actually very computationally intensive.

Project Milestone 4 Background

- Many steps in real science gateways require significant processing that is not directly done by the microservice.
- The microservice instead submits this work to an external computing cluster, which may be a resource that is shared with other users.
- This job may take many hours to complete and will go through a number of state changes that need to be tracked.
- It is also desirable to allow users to track the progress of their jobs, to cancel them, and to resubmit variants of an earlier job.

Project Milestone 4 Deliverables

- Using a Mesos cluster provided by the instructors, modify your microservices (such as Storm Clustering) to submit asynchronous jobs to the Mesos computing cluster. Monitor this execution.
- Provide state information on job progress through all microservices through the user interface.
- Allow a user to view all jobs they have submitted.
- Allow a user to modify an earlier, completed job so that it can be resubmitted with different inputs.

Project Milestone 4 Grading

- Demonstrate all the deliverables described above.
- Graders should be able to deploy your entire infrastructure, as in previous assignments.
- Demonstrate that your system can handle failures of the microservice instance that interacts with the Mesos cluster.

Final Presentation Schedule

- December 6th
  - Sangam, Omni, Omega
- December 8th
  - NPComplete, Flash, CodeRing
- December 13th
  - Bash, Aviato, Aurora

If you need to change times with another team, please make these arrangements yourselves and inform me by December 1.

Final Presentation Format

- Speaker 1: Demonstrate latest stable version of code
  - Milestone 3, minimally
- Speaker 2: Describe your current architecture
  - How did it evolve from your initial architecture?
- Speaker 3: Describe your approach to CI/CD and testing
- Speaker 4: Summarize your team's experiences
  - What did you learn?
  - What parts of the class need improvement?
  - What would you do differently if starting over?

The PHP Gateway for Airavata

A Science Gateway Demo, and How Gateways Deviate from the Elegance of Clouds

Research Computing: Buy or Rent?

- The current debate within universities is whether to buy and run hardware or rent it from cloud vendors.
- Many do a little of both
- Diversity of resources == opportunity for science gateways and cyberinfrastructure.
- So you need to understand both options.

Classic Cluster Scheduling and Queuing

Research Computing at IU

Resource	Description
Big Red 2 and Big Red 2+	Cray supercomputers with very high performance internal networking
Jetstream	OpenStack cloud based on commodity hardware
Karst	A commodity cluster

The Research Technologies division of UITS supports university-wide computing.

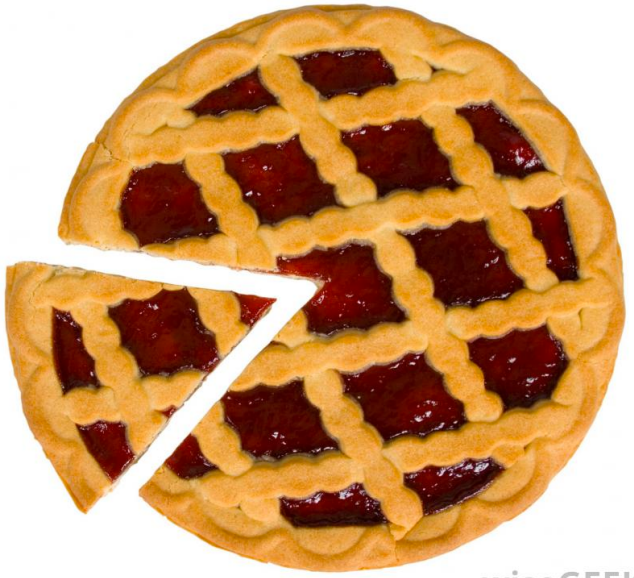
Other types of specialized research resources: large memory and GPU/coprocessor

The Karst Cluster at IU

- High-throughput rather than high performance
- Aggregate peak theoretical capability of 91.5 teraflops
- Aggregate RAM of 11 TB.
- Karst consists of 275 nodes total.
  - Red Hat Enterprise Linux 6.x
  - 2 Intel Xeon E5-2650 v2 8-core processors (so 16 cores/node)
  - 32 GB RAM
- 10-gigabit Ethernet interconnect.
- 450 TB of local spinning disk storage.
- See https://kb.iu.edu/d/bezu

Using Karst

- Karst is a resource shared with all other approved users.
- There are many users using part of the system at any given time.
- Problem: how do you get a portion of Karst to run your application?
- Solution: scheduling and resource management.



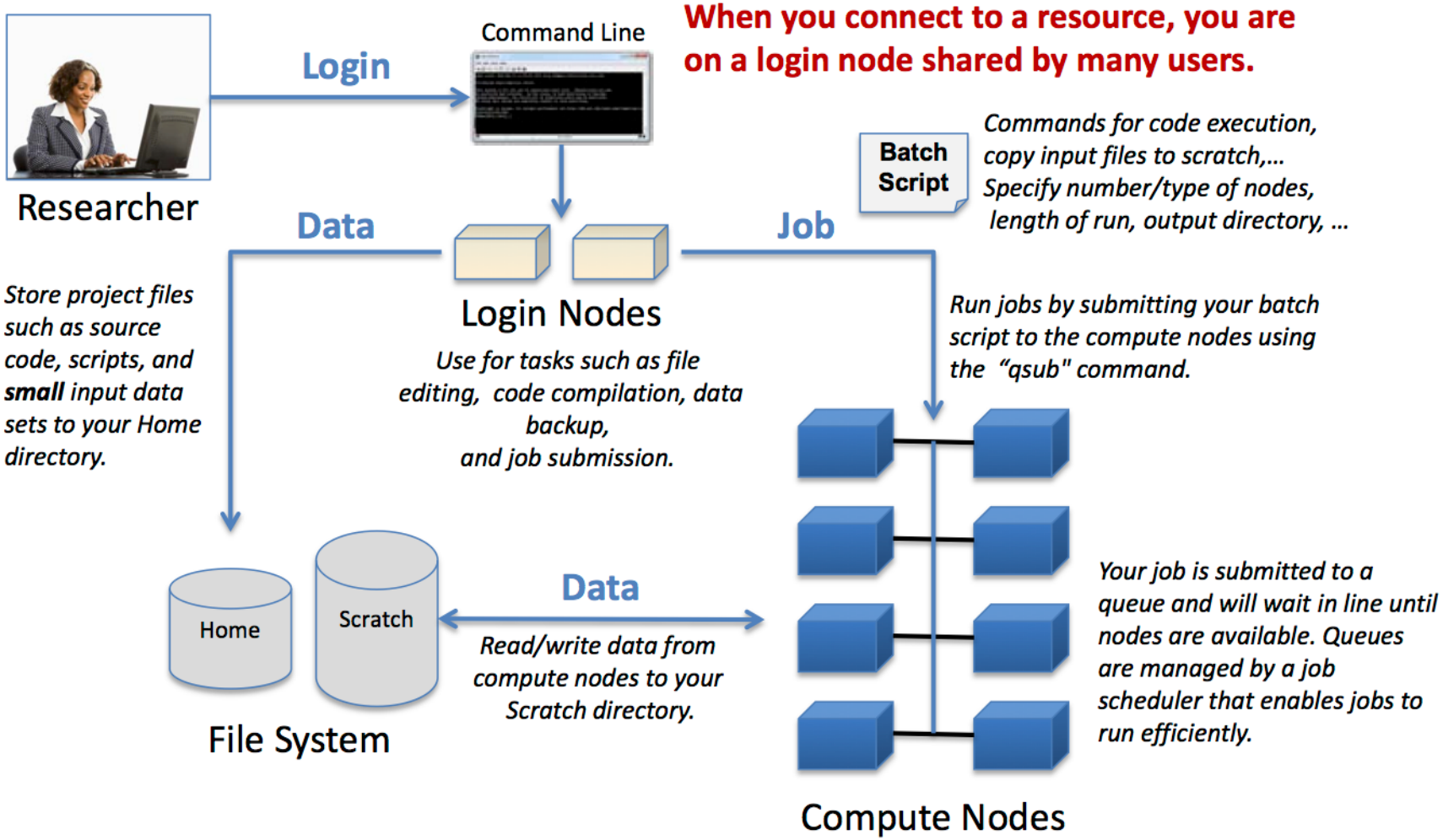
Classic Scheduling and Resource Management

- Scheduling
  - Determines which jobs to run based on resource requests versus availability.
  - Implement policies
    - Fair share: the more you use a resource, the lower your priority.
    - Large versus small requests
    - Important users
    - Reservations
- Resource Management
  - Manages the actual execution of the request on the resource
  - Monitors the health of nodes.
  - Supports fault tolerant execution.
  - Extendable to support different schedulers
  - NOTE: Mesos supports different schedulers *at the same time*.

Using Shared Research Computing Resources

Specific information on Karst, Big Red II, and other IU resources is available from http://kb.iu.edu.

Running Jobs Overview



https://**bluewaters**.ncsa.illinois.edu/running-your-jobs

Set Up Your Environment

- Karst administrators maintain many scientific applications, libraries, and tools for compiling codes.
- These can have complicated and conflicting libraries and other dependencies.
- Karst uses the “module” command to help users set up their environments for particular applications.
  - Just a simplified way to managing environment variables
  - No containerization

Create a Job Script

- Job scripts are just scripts but have special directives at the beginning for the scheduler.
  - #PBS for PBS-based schedulers
- Use these directives to specify
  - Number of nodes you want
  - Processors per node
  - Walltime
  - Job name (-N)
  - Notification options (-m)

```
#/bin/bash
#PBS -l nodes=2:ppn=8,walltime=2:30:00
#PBS -m bae
#PBS -N job_myprog

#Move to your working dir
cd $PBS_O_WORKDIR

# Run a parallel job
mpirun -np 16 lmp_openmpi -suffix omp
< in.cmdfile
```

Submit and Monitor Your Job Script

- Submit:
  - qsub ~/work_directory/my_job_script.pbs
- Monitor:
  - qstat -u *username* (Torque)
  - checkjob *job_id* (Moab)
- Get notified by email
  - Use appropriate job script settings
  - Email when **jobs begin, end, or abort**
- Schedulers/queuers track and log more states than these.
  - Useful to science gateways
- You may want to expose your application's internal state
  - This is an open problem
  - Resource managers don't provide this capability.

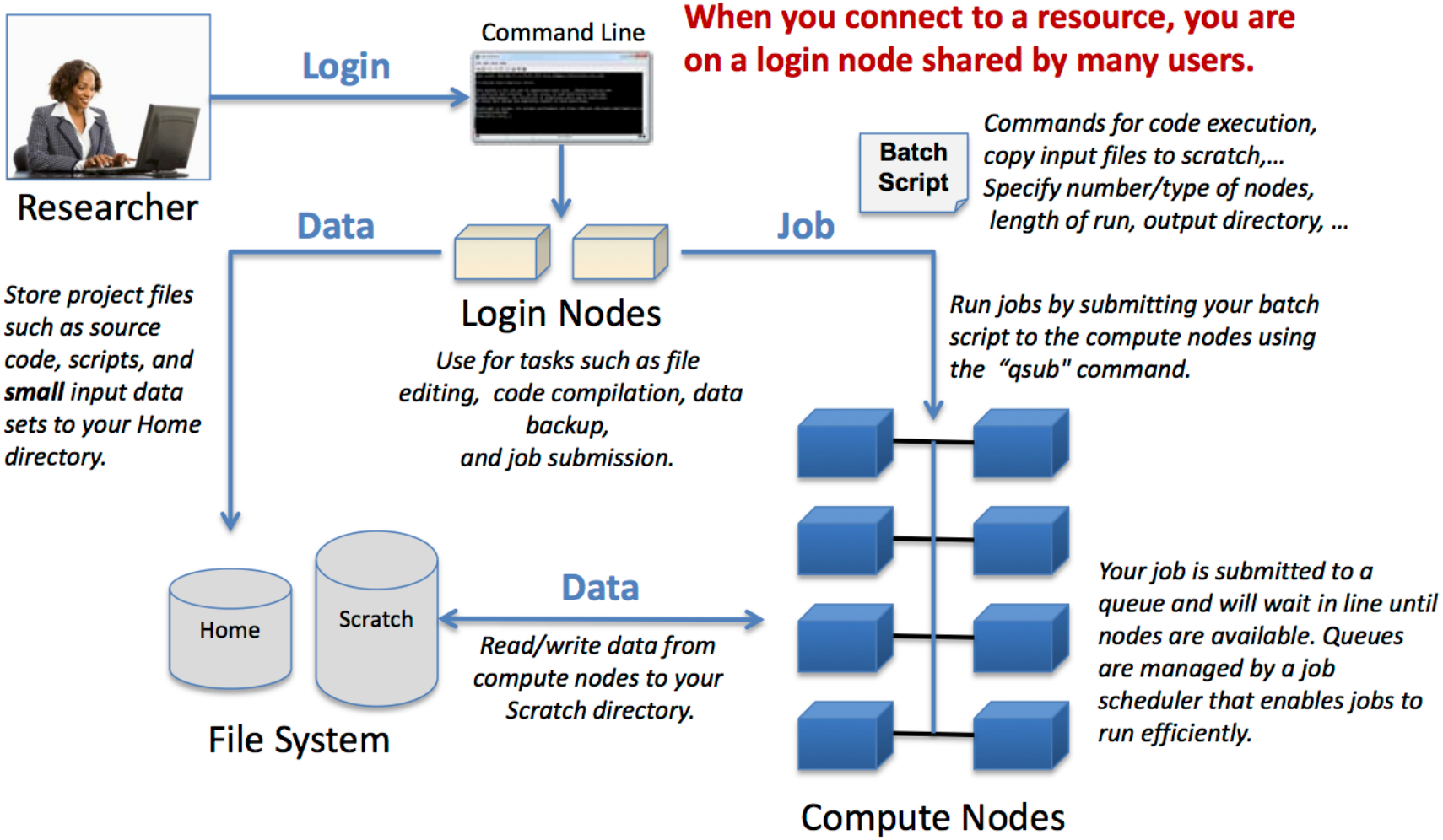
Congratulations!

You've seen now how to run 1 application on 1 cluster at 1 university. There are many applications on many clusters at many universities. Each is its own special snowflake.

Science gateways hide these complexities while enabling more powerful usage.

More on Scheduling and Resource Management

Running Jobs Overview



https://**bluewaters**.ncsa.illinois.edu/running-your-jobs

A Classic Configuration

- A Torque cluster consists of one head node and many compute nodes.
- The head node runs the pbs_server daemon
- The compute nodes run the pbs_mom daemon.
- The pbs_server process must by high availability.
- The pbs_mom processes can be less so (elastic).

http://docs.adaptivecomputing.com/torque/6-0-0/help.htm

The Job Runs

- Users submit jobs to **pbs_server** using the qsub command.
- The **pbs_server** informs the **scheduler**.
  - Maui, MOAB, FIFO
- When the **scheduler** finds nodes for the job, it sends instructions to run the job with the node list to **pbs_server**.
- Then, **pbs_server** sends the new job to the first node in the node list and instructs it to launch the job.
- This node is designated the execution host and is called ***Mother Superior***.
- Other nodes in a job are called ***sister MOMs***.

Scheduling

- The head node also runs a scheduler daemon.
- The scheduler interacts with pbs_server to make local policy decisions for resource usage and allocate nodes to jobs.
- A simple FIFO scheduler, and code to construct more advanced schedulers, is provided in the Torque source distribution.
- Most Torque users choose to use a packaged, advanced scheduler such as Maui or Moab.

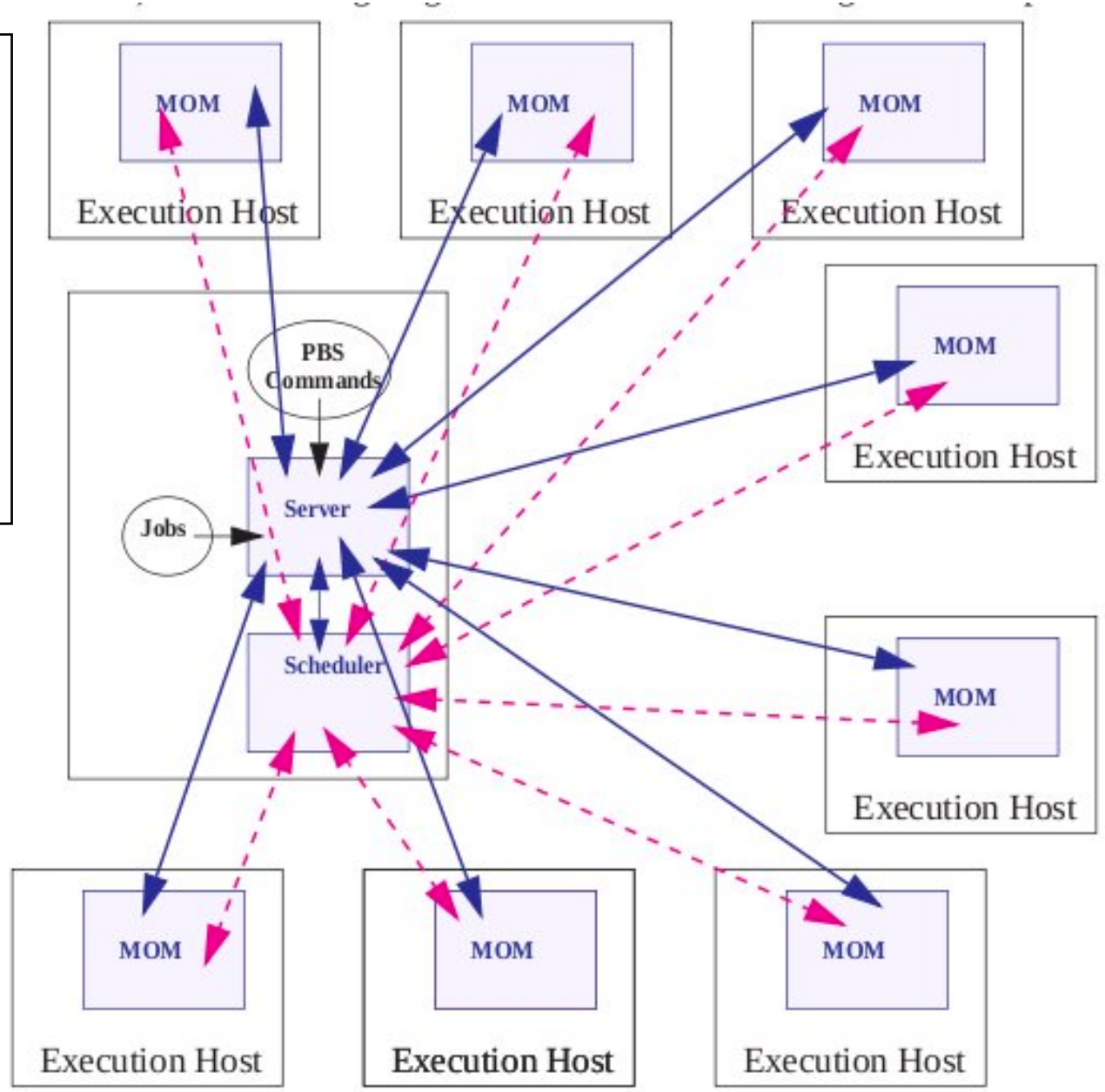
http://docs.adaptivecomputing.com/torque/6-0-0/help.htm

Basic Torque Cluster Deployment

- User submits jobs to server
- Server contacts scheduler
- Scheduler decides when things run.
- Resource managers run applications when told by the scheduler.

You could think of each MOM as a **microservice**. The pbs_scheduler is your **API gateway**

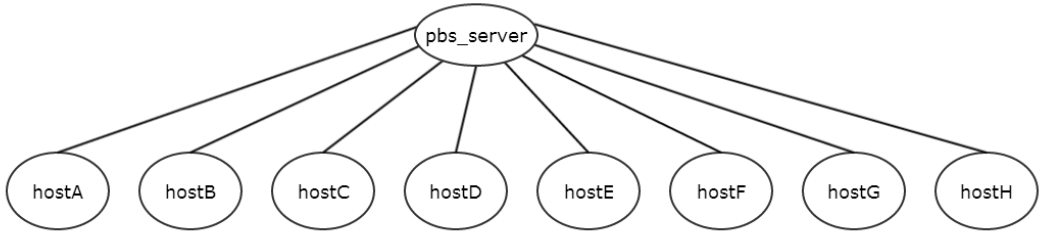
What are some problems with this layout?



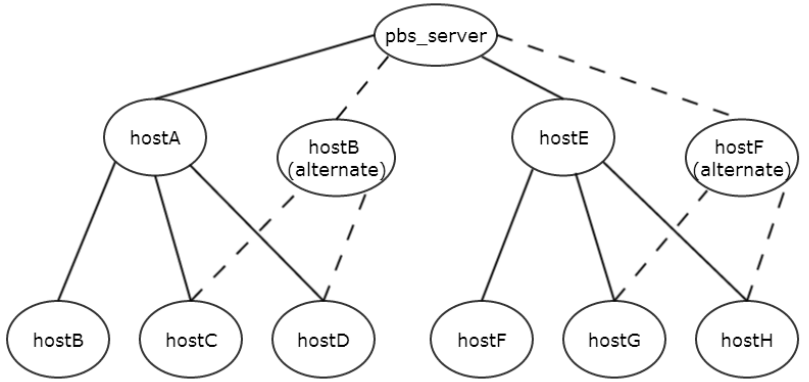
MOM Hierarchy for Scaling

- Override the compute nodes' default behavior of reporting status updates directly to the pbs_server.
- Each node sends its status update information to another compute node.
- The compute nodes pass the information up a tree or hierarchy until eventually the information reaches a node that will pass the information directly to pbs_server.
- This can significantly reduce network traffic and ease the load on the pbs_server in a large system.

Without mom_hierarchy



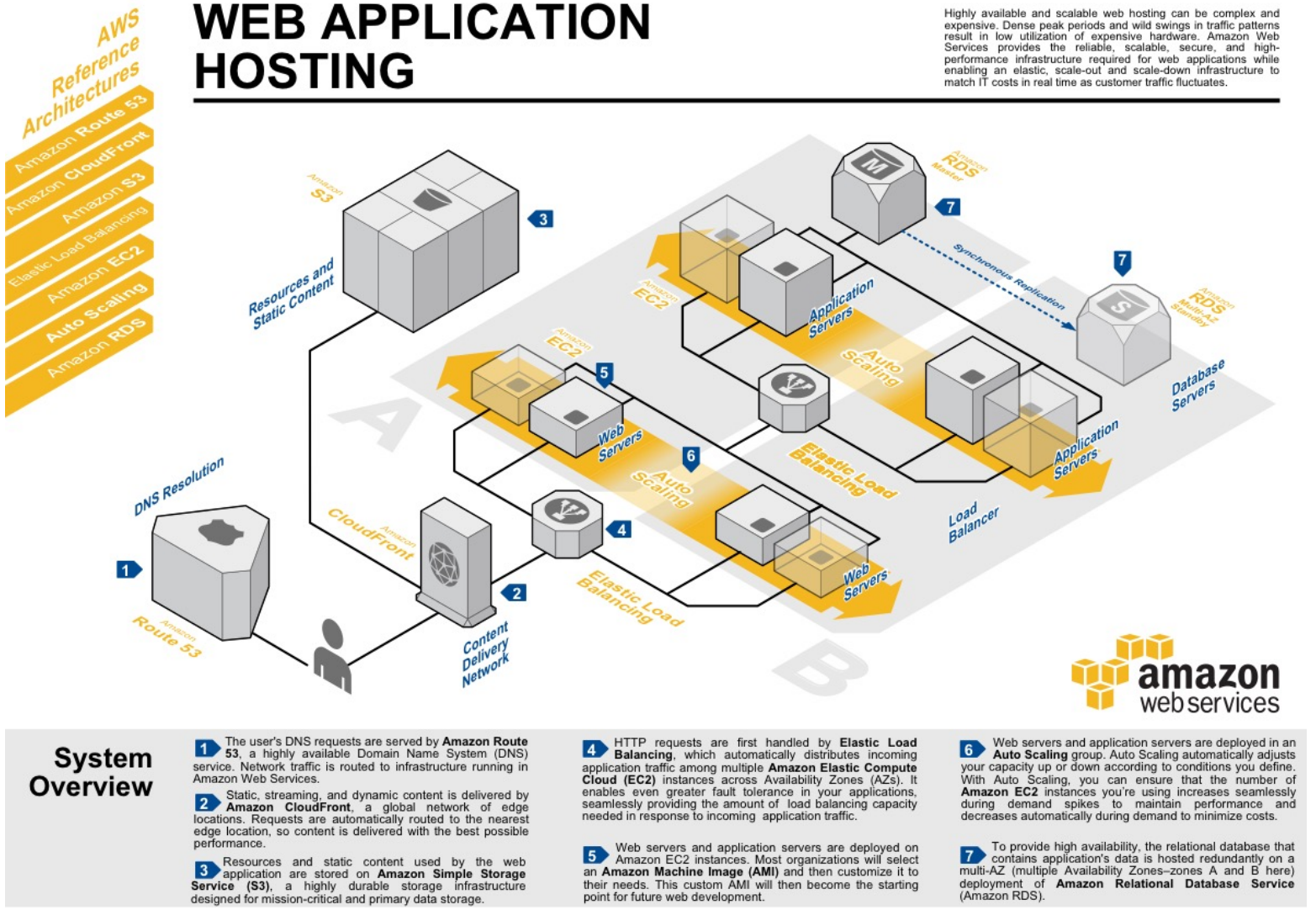
With mom_hierarchy



http://docs.adaptivecomputing.com/torque/6-0-0/help.htm

Compare and Contrast

- The manager-worker pattern and its improvements are common in many schedulers.
  - Manager knows the **state** of the entire system.
- Mixture of high availability and failable, elastic components.
  - Workers can fail.
- Reliable communication needed.
  - But not for everything
- Ephemeral components need to be easily configurable
- These patterns have found their way into other places ->



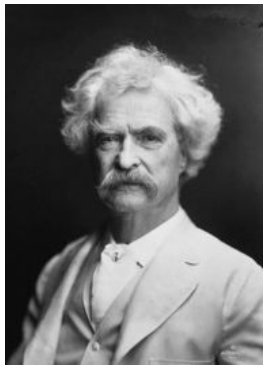
https://aws.amazon.com/architecture

Basic Science Gateway Ideas

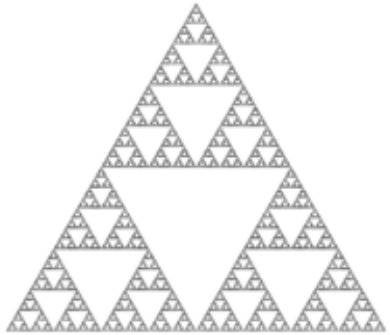
- Hide all those complicated details for submitting and monitor jobs.
- Provide programmatic access to clusters and supercomputers by wrapping command lines, generating scripts, etc
  - Eliminate user errors
- Scale the number of users of scientific applications on clusters.
  - There will always be power users
- Do all of this for multiple applications on multiple resources.
- Provide better user environments

Life Lessons for Gateway Builders

- Every cluster and supercomputer is different
  - Custom built
  - Commodity or custom hardware
- There are multiple scheduling and queuing systems that one can choose.
- Even if two clusters use the same scheduler, the local installation will be different
  - Hardware is different, scheduling policies are different, etc.
- Each application runs in a different way on each machine
  - LAMMPS requires different module recipes on different machines

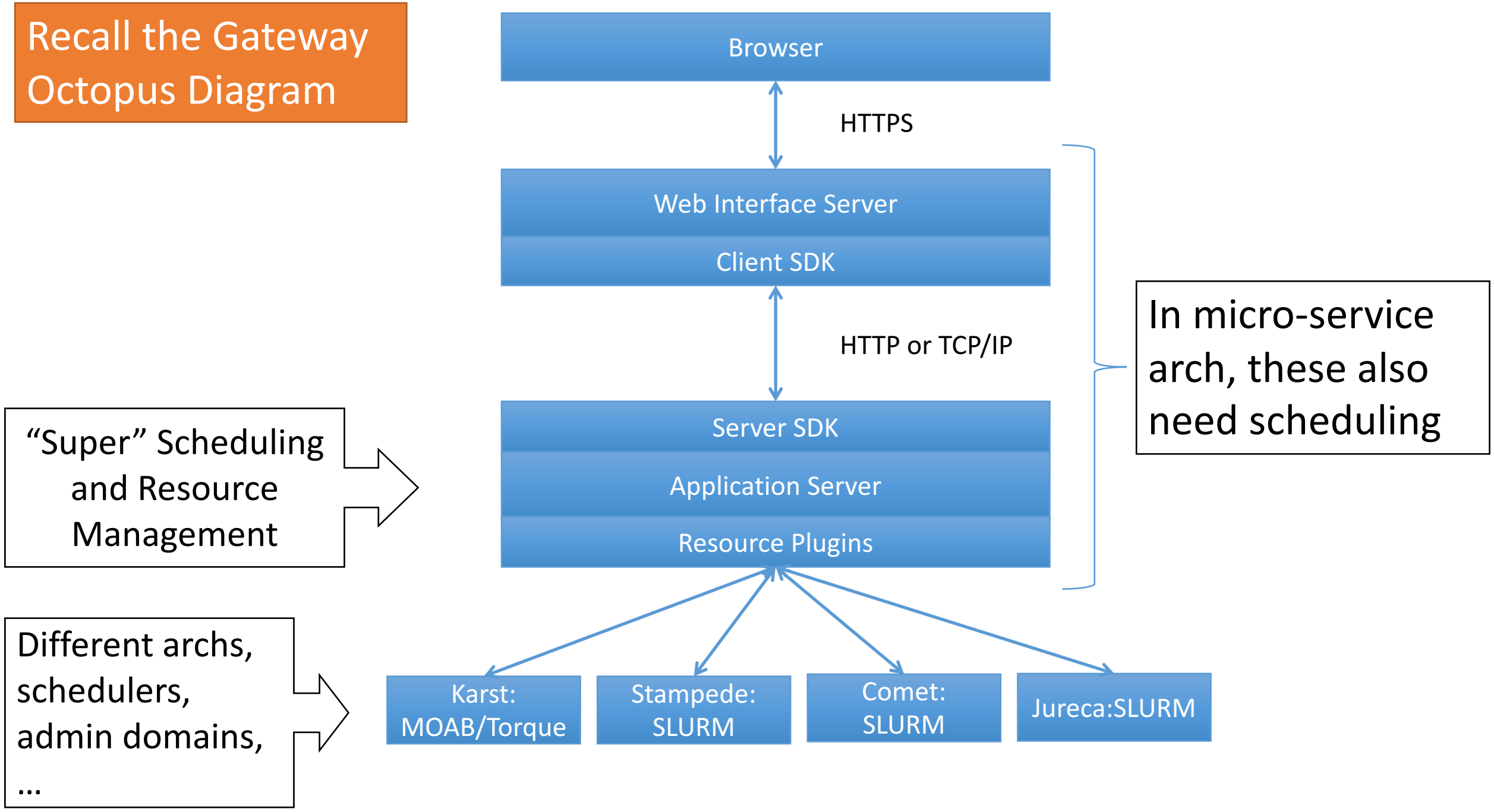


Distributed Computing Rhymes and Fractals



- Schedulers and Resource Managers are sophisticated and powerful software.
  - Scheduling is a classic computer science problem.
  - Provisioning and monitoring resources and running applications at scale is a challenging distributed computing problem.
  - Many task computing, map-reduce, job chaining and other sophisticated execution patterns
    - But some people also run schedulers inside schedulers: ”Glide-In”, ”Pilot Jobs”
- Gateways operate at a level above resources but have much to learn and borrow
  - Meta-scheduling
  - Scaling
  - Fault tolerance: what’s high availability and what’s not?
  - Security
  - Messaging
  - State management and recovery

Recall the Gateway Octopus Diagram



Logging In

- You log into Karst with SSH
  - ssh *username*@karst.uits.iu.edu
  - Type your password at the prompt
- karst.uits.iu.edu resolves to 1 of 4 publically accessible head nodes.
  - Most nodes are not reachable directly from outside
- You can and should set up public/private key authentication.
  - See https://kb.iu.edu/d/aews
  - Two-factor authentication: something you have (private key) and something you know (the key's passphrase).

LAMMPS on Karst: Modules

```
# Run these commands
# Option 1
module load gcc
module load  openmpi/gnu
module load lammps
```

```
#Option 2
module load intel
module load openmpi/intel
module load lammps
```