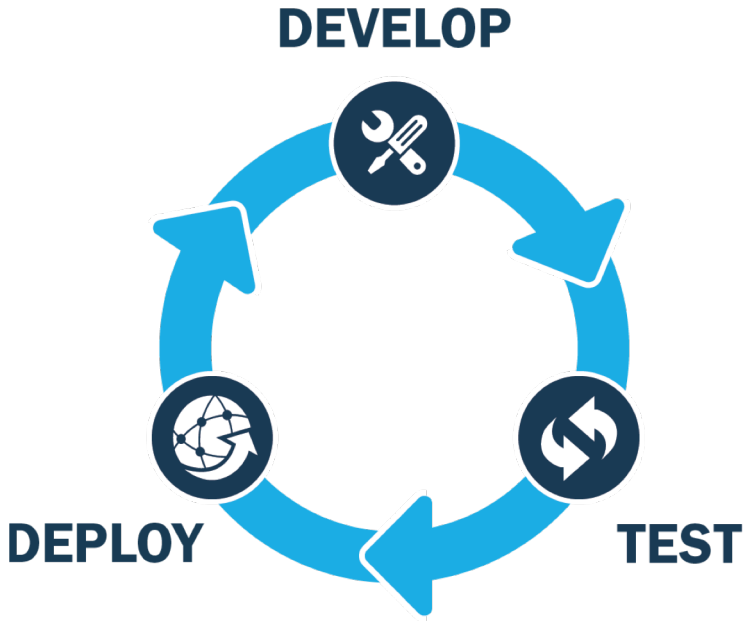
Supporting Continuous Integration at Large-Scale HPC Centers

Todd Gamblin (LLNL)
David Montoya (LANL)
Rob Neely (LLNL)



SIAM CSE19

Spokane, WA
March 1, 2019

EXASCALE COMPUTING PROJECT

National Nuclear Security Administration
U.S. DEPARTMENT OF ENERGY | Office of Science
EXASCALE COMPUTING PROJECT

What is Continuous Integration?

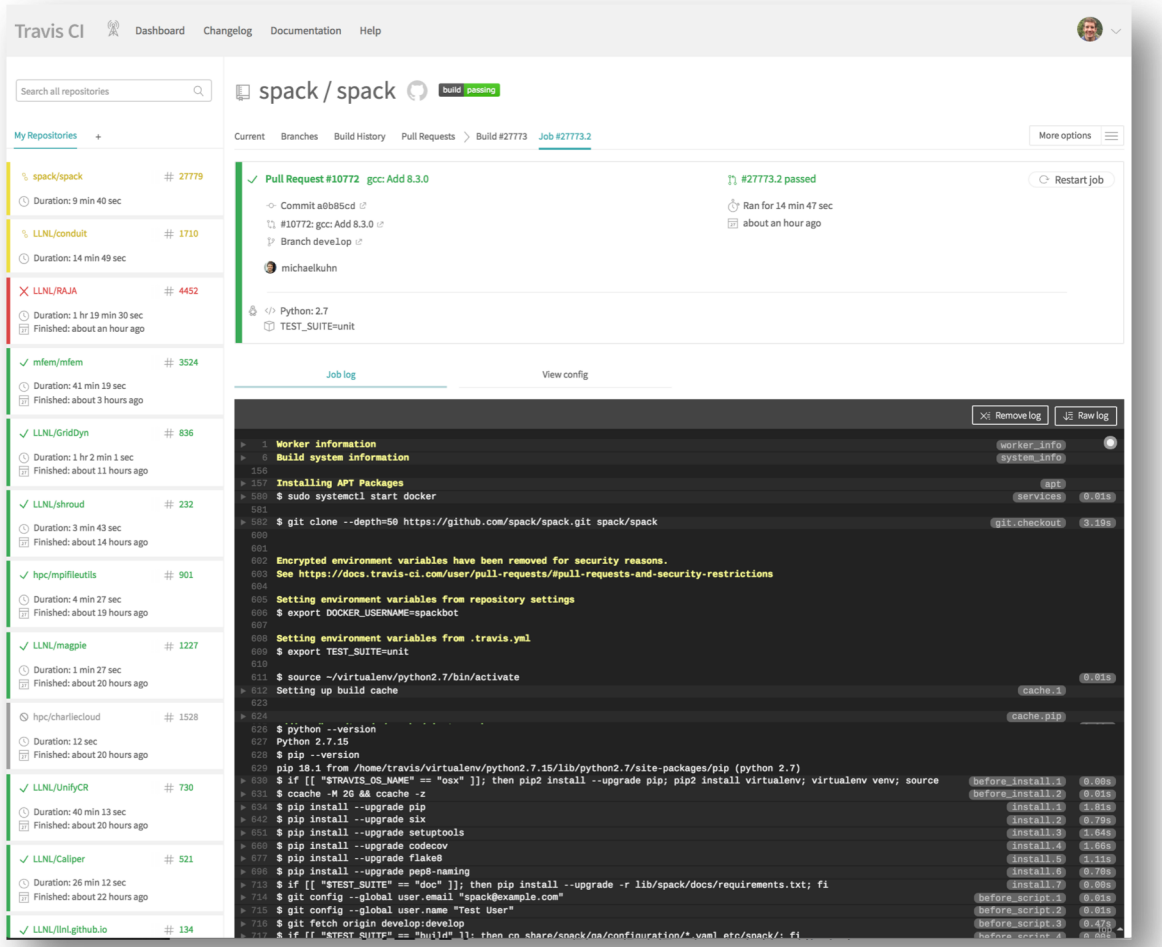
- **Originally meant merging all developer changes into the repository multiple times per day, to fail fast and fix integration issues fast.**
  - Required automated build and automated testing, typically before every check-in
  - People used build servers to continuously test the repository
- **Today, the term is used more broadly to include most ways of testing changes when they are integrated into the mainline.**
  - Nighty testing
  - Testing pull requests when they are submitted
  - Testing commits as they are added to master branch
  - Automatically committing contributions that pass tests
- **CI requires:**
  - Access to the repository by automated systems
  - A way to run jobs automatically on test systems of interest

Travis CI

Jenkins

Basic CI is *free* and extremely easy to set up (for cloud projects)

- **Tools like Travis CI popularized simple configuration and free cloud services for CI**
- **Just edit a YAML file in your repo, and your tests run in the cloud, for free!**
- **Limited set of environments for typical HPC development**
  - Basic Linux, Windows environments
  - Mostly x86_64
  - No advanced architectures, GPUs (need to set up your own)
- **Still good for basic unit testing of HPC projects.**

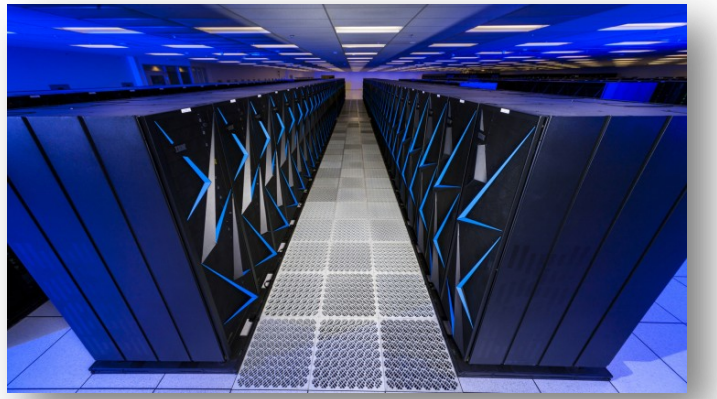


Continuous Integration tools pose a number of security challenges for large, multi-user HPC centers.

1. **CI server is a persistent service; not suited to HPC batch model.**
   - Runner daemons need to be persistent
   - Batch jobs typically have a fixed time limit, but HPC centers built around batch.

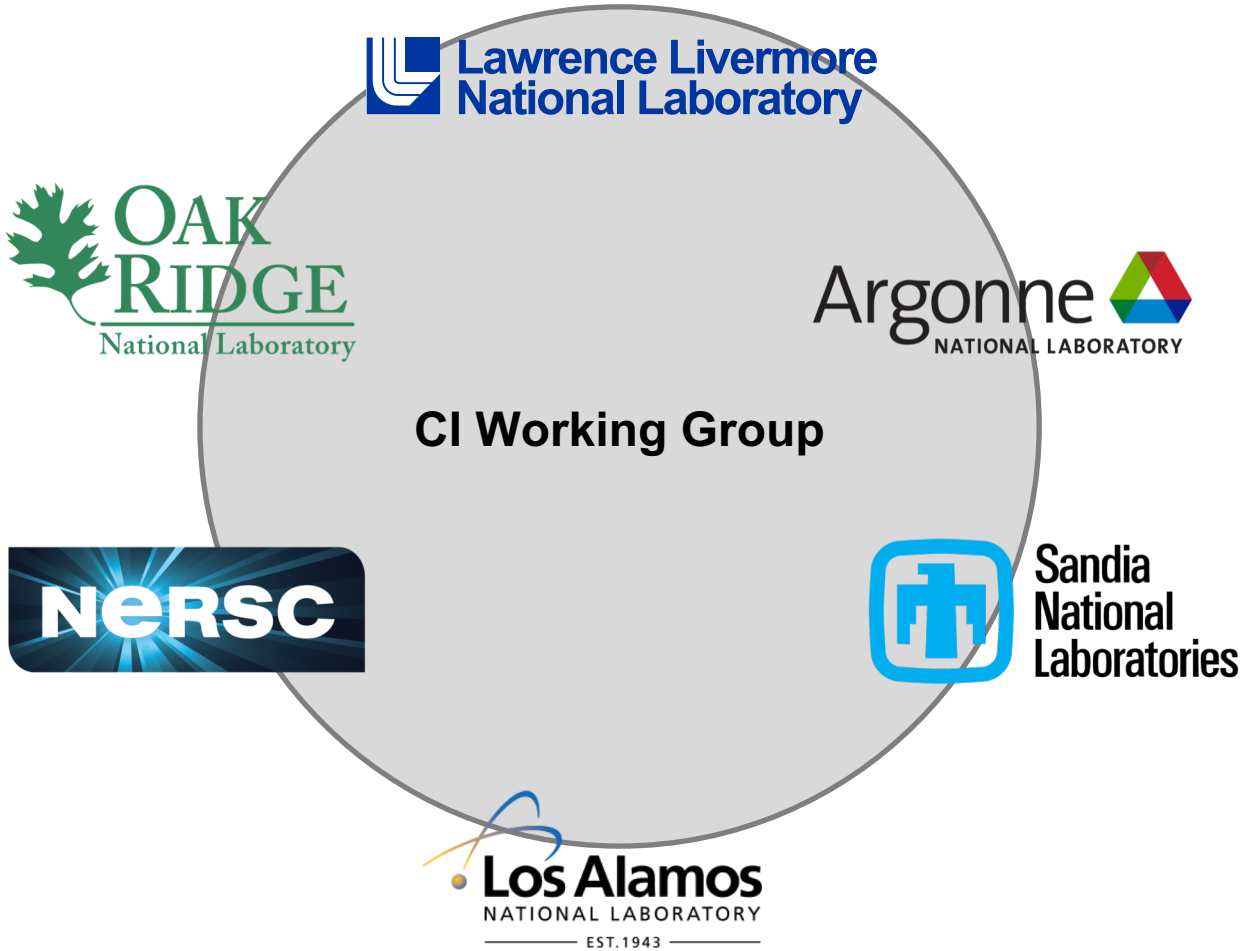
2. **Need to run arbitrary code on machines, automatically.**
   - Often in response to *external* repository check-ins
   - How do we know who ran the code?
   - How do we trust users, and who do we blame if it the code is malicious?

3. **Job runners on most systems don't run as specific HPC users**
   - Can't allow different users' jobs to share data.
   - Need isolation between jobs run by user A and jobs run by user B
   - Users could set up their own runners, but this has steep maintenance requirements.



EXASCALE COMPUTING PROJECT

ECP convened a working group in 2017 to assess CI requirements



1. Enumerate security requirements for DOE HPC facilities
2. Design a statement of work to add features to some existing CI solution
3. Find a subcontractor to implement all of the features

EXASCALE COMPUTING PROJECT

The working group produced a call for proposals.
DOE selected GitLab as the CI system, and Onyx Point as the implementor.

Done----------

- **Milestone 1:** Single-center SetUID Runners
- **Milestone 2**: Single-center Batch Runners

To-do ----------

- **Milestone 3**: Securely run-as team user
- **Milestone 4**: Cloud UI to enable runners at *multiple* centers.
- **Milestone 5**: Enhanced auditing

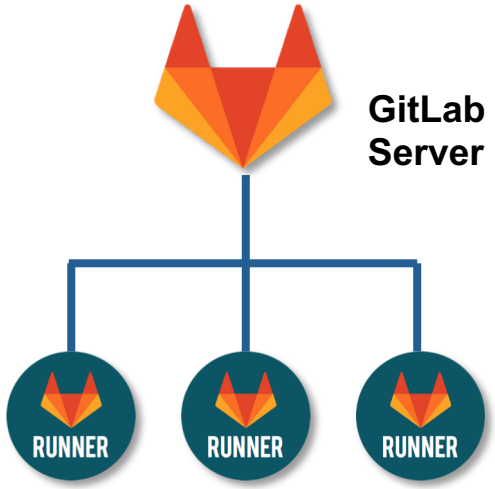
EXASCALE COMPUTING PROJECT

Milestone 1: SetUID runner - completed

Enhancements over current Gitlab CI

- **Facilities** deploy and maintain trusted runners (not users)
- Runners **run as users** (trust GitLab server to say who to run as)
- **Facilities set whitelists and blacklists** for both users and groups, per runner
    - final authority on who to run as is with the runner

On a normal GitLab instance, there would not be sufficient isolation between runners to meet the needs of HPC sites.



GitLab and runners are *trusted*
Runners run as users

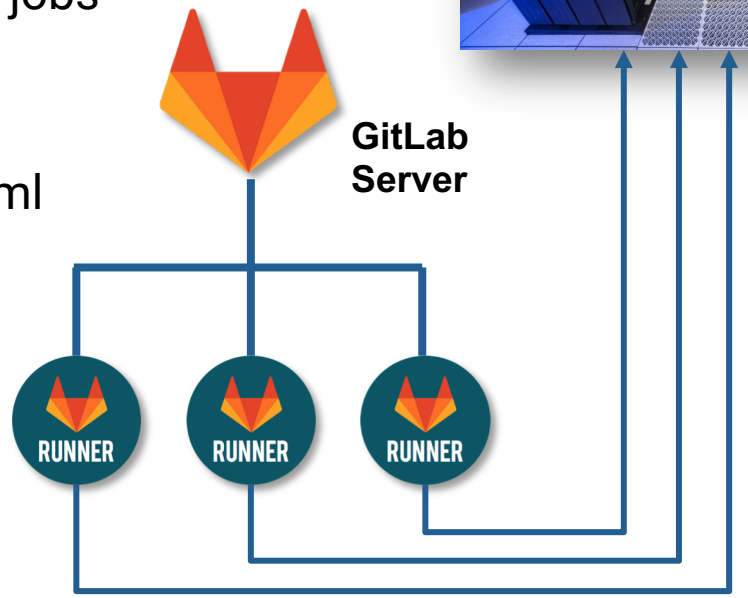
source: www.gitlab.com

Milestone 2: Batch runner - completed

Batch runners are special SetUID runners

Enhancements over regular SetUID runners

- Runners use batch system, and **do not block** when running jobs
  - Allows sites to leverage all their HPC resources
- Integration with **SLURM, LSF, Cobalt batch systems**
- Users can specify **parallel resource requests** in .gitlab-ci.yml



Runners run as users;
submit jobs to batch system;
don't block while jobs run

source: www.gitlab.com

The interface is *nearly the same* as GitLab.com or any other GitLab instance

```
job1:
  script:
    - ./build.sh
    - ./run-tests.sh
```

Add a simple `.gitlab-ci.yml` file at the top level of your repository

- **.gitlab-ci.yml is a simple YAML file that controls what tests are runs.**
  - Contains a list of jobs, each with its own script.
- **Facilities administer runners, so users don't have to do anything but edit this file**
  - Tests are run on pushes, pull requests, and other changes to the repository
  - Frequency and other options are customizable.

There are some things to keep in mind for HPC: Tags

- **CI systems like this were designed with a homogeneous Linux cloud in mind**
  - Tests run on Linux
  - Scripts work the same everywhere
  - No arch differences, etc.

- **HPC is more complex!**
  - Different runners have different characteristics
  - Login vs. batch
  - x86_64 vs. Power vs. ARM

- **Runners will be *tagged* to indicate these differences**
  - Jobs can specify tags to say where they should run

- **Scheduler parameters are specified as variables**

```
job1:
  script:
    - ./build.sh
    - ./run-tests.sh
  tags:
    - nobatch

job2:
  script:
    - ./build.sh
    - srun run-parallel-tests.sh
  tags:
    - batch
  variables:
      SCHEDULER_PARAMETERS:
          "-P STF002 -J pythontest -W 0:01 -nnodes 2"
```

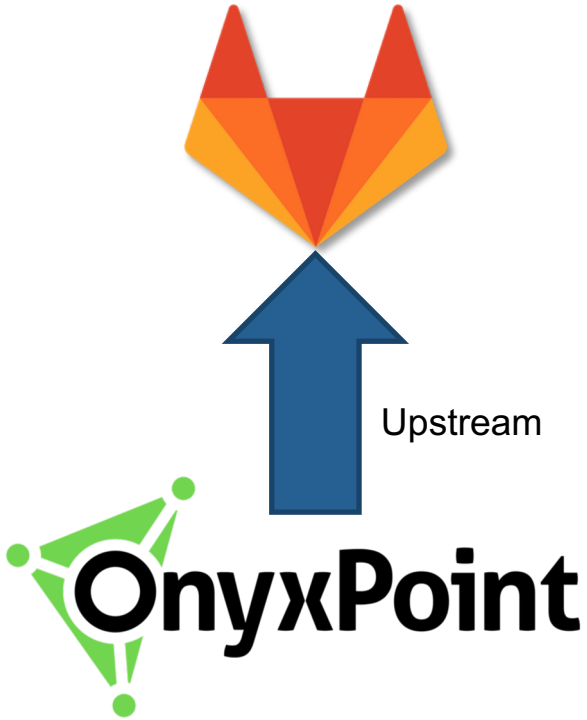
Ability to run as team users will give more control to users and facilities

- **Many teams want to run as a "service user" dedicated to CI**
  - Provides a clean testing environment free of particular users' setup
  - Only certain users on the team can run as these service users
  - User varies by team
- **Will be specified in .gitlab-ci.yml with a service user variable**
- **Facilities will have control over who runs as which service user**
  - Customizable security logic to check access
  - Each facility can do this their own way

```
job1:
  script:
    - ./build.sh
    - ./run-tests.sh
  tags:
    - nobatch
  variables:
      SERVICE_USER: myapp-testuser
```

Onyx Point is working to upstream these features to GitLab

- **SetUID runners are generally usable at other sites**
  - GitLab is interested in integrating this feature into their product
  - Other features TBD
- **We have tried to make ECP general enough to release**
  - Target simplicity – try not to be HPC specific unless we have to
- **We expect open source contributions from ECP to have a lasting effect.**
  - Any HPC site will be able to do this with GitLab, not just labs



The remainder of the project focuses on enabling continuous integration *across* DOE sites

Done----------

- **Milestone 1:** Single-center SetUID Runners
- **Milestone 2**: Single-center Batch Runners

To-do ----------

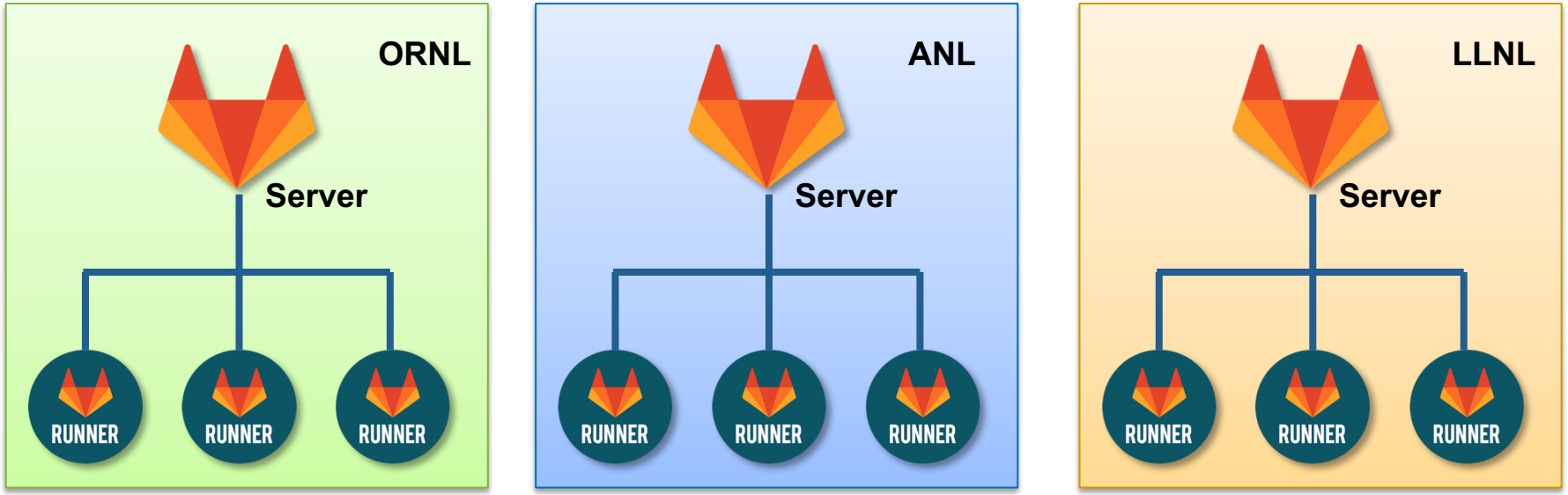
- **Milestone 3**: Securely run-as team user
- **Milestone 4**: Cloud UI to enable runners at *multiple* centers.
- **Milestone 5**: Enhanced auditing

Milestones so far allow HPC facilities to run CI for their own local users

- **SetUID functionality allows HPC sites to administer runners *on behalf of users***
  - Existing solutions at HPC sites put these burdens on users, now facilities manage them

- **Batch runners provide integration with existing resource management (SLURM, LSF, Cobalt)**
  - Users can make CI suites with parallel tests
  - Facilities can control queues and allocations used for CI
  - Allows for gradual phase-in of CI – learning period as sites discover user needs

- **Only locally administered GitLab instances are possible**
  - Facilities can use only their *own* runners

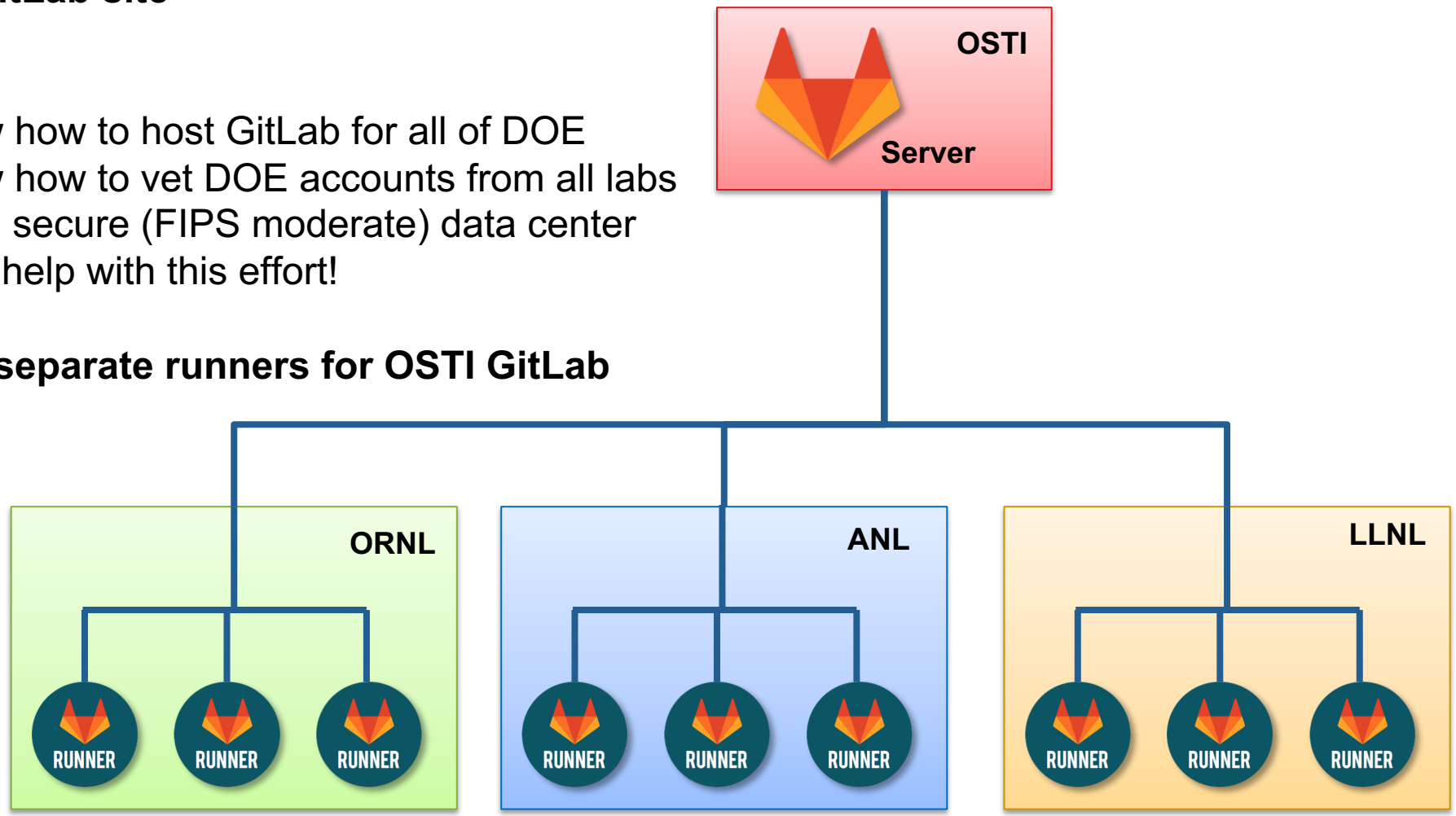


Milestone 4: Federation will allow us to bring together facilities and users from across ECP

- **OSTI will host an ECP GitLab instance alongside their existing DOE Code GitLab site**

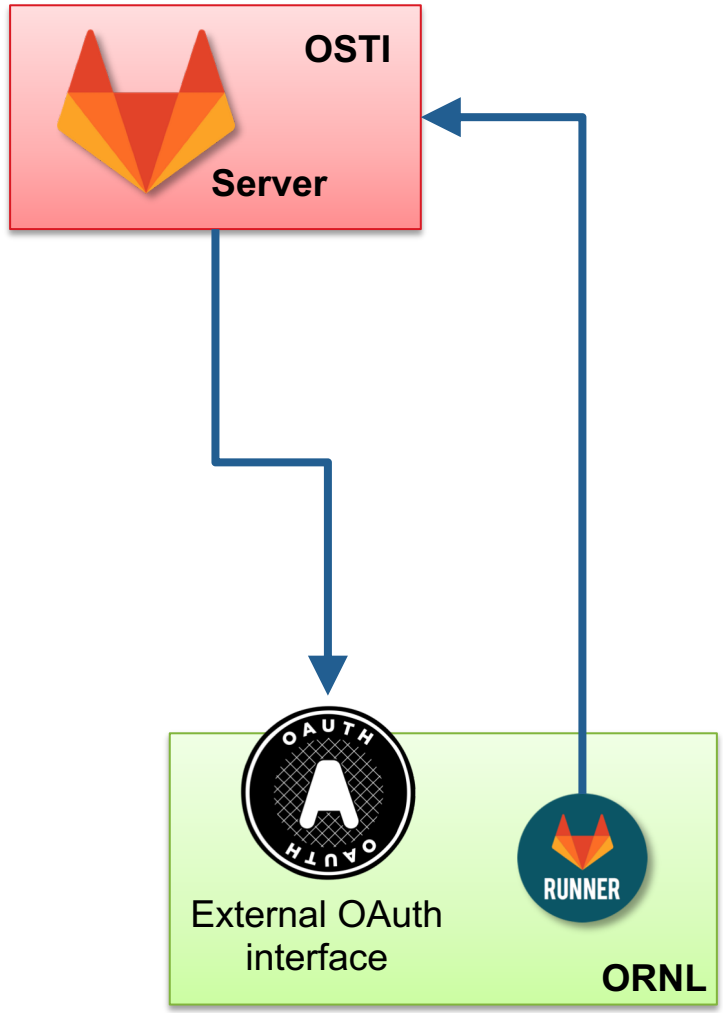
- **Why OSTI?**
  - Staff already know how to host GitLab for all of DOE
  - Staff already know how to vet DOE accounts from all labs
  - Staff already run a secure (FIPS moderate) data center
  - Well positioned to help with this effort!

- **Facilities will deploy separate runners for OSTI GitLab**



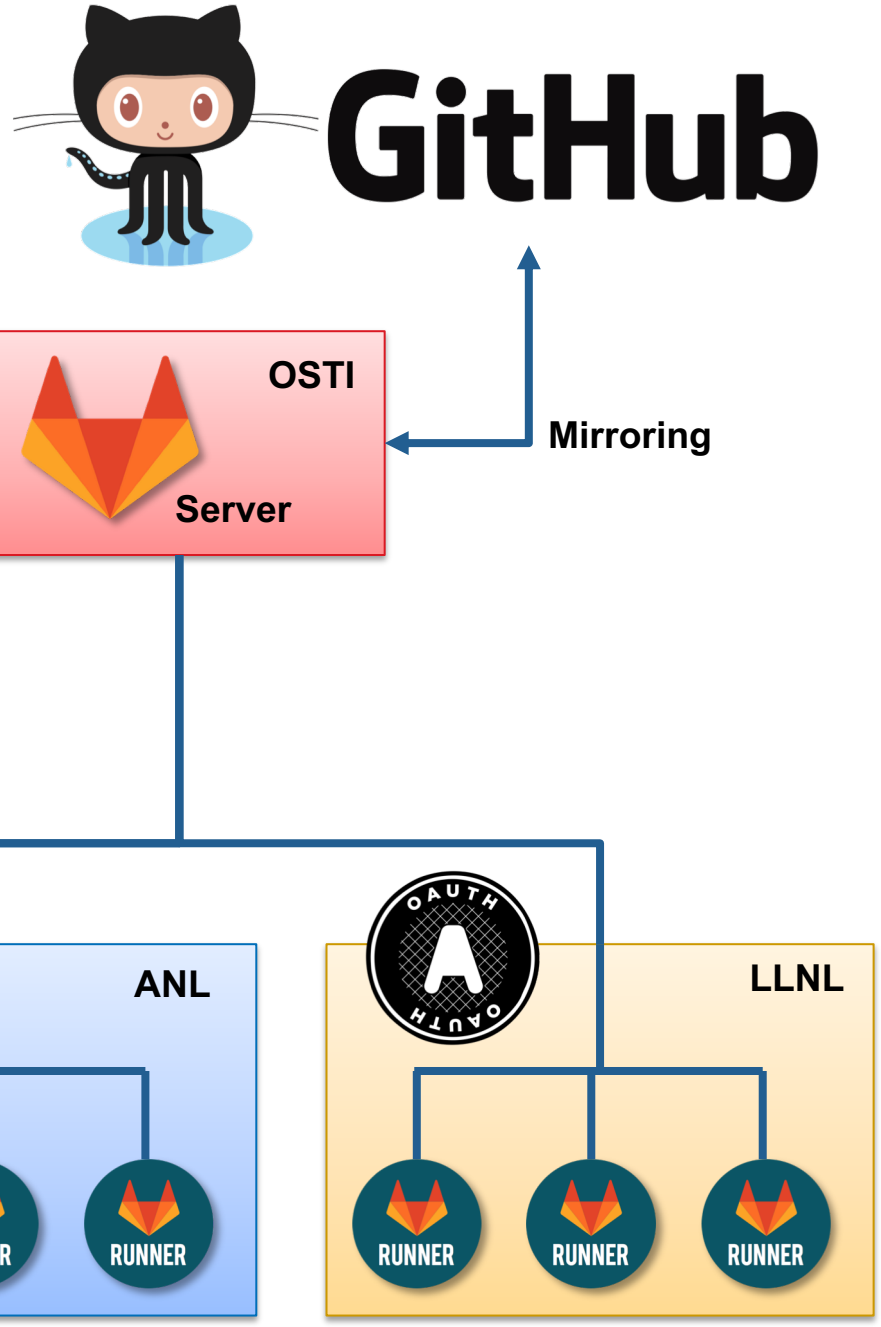
Federation brings new security challenges!

- **How does OSTI GitLab know about facility accounts?**
- **Current plan:**
  - Facilities provide an OAuth Interface to allow external authentication (note: NOT login)
  - When runners are registered, they are associated with an OAuth domain for their site by an administrator
  - Users will be able to authenticate with facility credentials to gain ability to run at different sites.
- **Only users with accounts at specific facilities can run there**
- **Facilities have final say (via runners) over who runs jobs**
  - Can revoke access or shut down runners at any time
- **Onyx Point will integrate this support into the existing SetUID runner implementation and into the GitLab server.**



We do not want to force all users onto GitLab

- **Many (likely most) projects are hosted in GitHub**
- **One of the reasons we chose GitLab was because you can *mirror* from external repositories and do CI on their behalf**
  - GitLab acts like Travis with special, fancier resources
- **We are still working out the security agreements necessary for this use case**
  - Likely need to make certain assertions about users committing to the GitHub repo
  - Final Auditing milestone may help



The ECP CI system has long-term ramifications for software robustness

- **It has been extremely difficult for developers to test their software and keep it up to date**
  - This system will enable us to automatically test in the environments we care most about

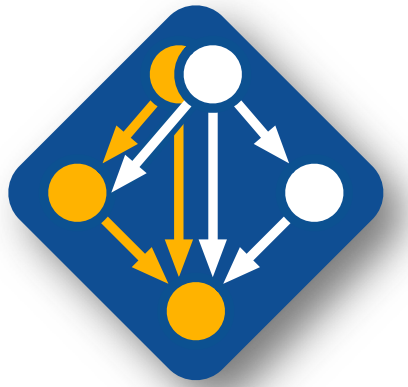
- **Teams can test their software as they develop**
  - Prevent regressions
  - Avoid tedious manual builds
  - Find bugs early!

- **We expect that this system will make capability class systems much easier to use**
  - Software will be available and tested
  - Facilities can rely on a much more robust software stack
  - Software can be built and available *before users need it*
  - Efforts can be leveraged across sites

EXASCALE COMPUTING PROJECT

ECP CI has potential not only for individual teams, but for larger-scale software testing and distribution

+



- **Spack is the software deployment technology used across ECP**
  - From-source builds of complex software for HPC systems
  - Spack allows others to reproduce builds with different options
  - Spack also provides a binary packaging capability

- **We can use ECP CI to automatically run Spack builds**
  - Builds can “rot” over time as environments change
  - Ensure that packages don’t go out of date
  - Notify developers when packages need to be updated

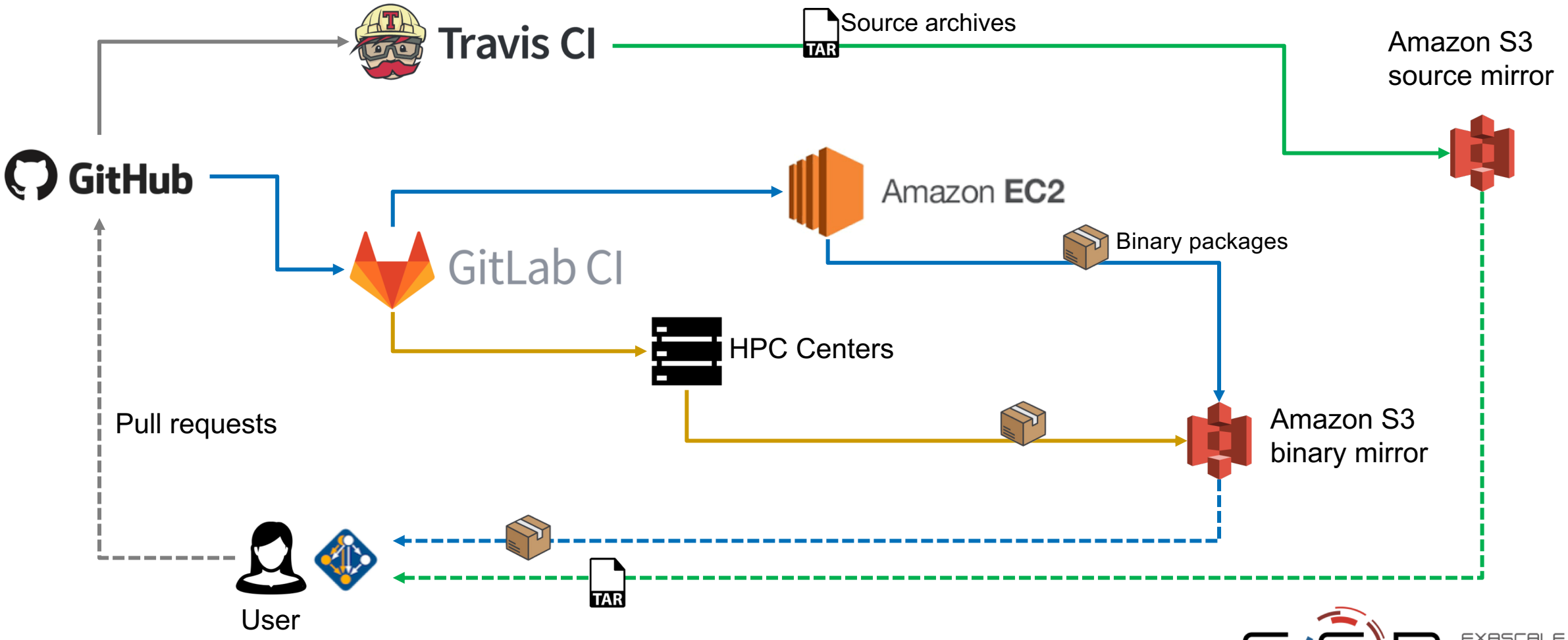
- **The Spack teams at LLNL and Kitware have developed a package build pipeline that relies on GitLab CI**
  - We are positioned to leverage the work of the ECP CI team
  - This would make a binary distribution for HPC feasible for the first time

https://spack.io

github.com/spack/spack

@spackpm

We are building CI infrastructure for source and binary distribution in Spack



There are many potential benefits to a curated, automated ECP software distribution

- **No more waiting for source builds**
  - Optimized binaries would be available pre-built for exascale systems
- **Facilities and users can leverage the same builds using Spack**
  - Binaries are relocatable
  - Same binary can be installed for site-wide use or in user home directory
  - Facilities can customize modules, but the actual installation is the same
- **Optimizations done for one team can be used by many!**
  - Spack provides a repository for build knowledge
  - Packages leverage collaborative efforts of many developers
- **Potential for greater software assurance through automation**
  - OSTI GitLab and Spack CI provide a central place where binaries could be scanned for malicious code.
  - Binaries are signed, so facilities and users would have assurance of their provenance and of the measures taken to vet them.

Many challenges remain ahead, but we hope ECP CI work will have a lasting effect on the HPC software ecosystem

- **We are working on security approvals**
  - Inter-site agreements are needed for OSTI GitLab
  - Also needed for a model for mirroring from GitHub
  - Major collaborative effort between labs and facilities

- **Working with facilities to manage CI resource demands will be an ongoing process**
  - We need to balance CI with production work
  - Containerized build environments provide a potential avenue to enable builds to run off of costly HPC resources
  - We could build *for* HPC systems in the cloud, or on private clouds of build nodes at different site

- **The potential for developers and for software robustness is huge!**
  - This kind of collaboration would **not** be possible without ECP