libEnsemble: A Python Library for Dynamic Ensemble-Based Computations

David Bindel[1,2] Stephen Hudson[1] Jeffrey Larson[1]
John-Luke Navarro[1] Stefan M. Wild[1]
[1]Argonne National Laboratory [2]Cornell University

Overview

libEnsemble is a Python library for coordinating the concurrent evaluation of dynamic ensembles of calculations. The library is developed to use massively parallel resources to accelerate solving design, decision, and inference problems and expand the class of problems that can benefit from increased concurrency levels.

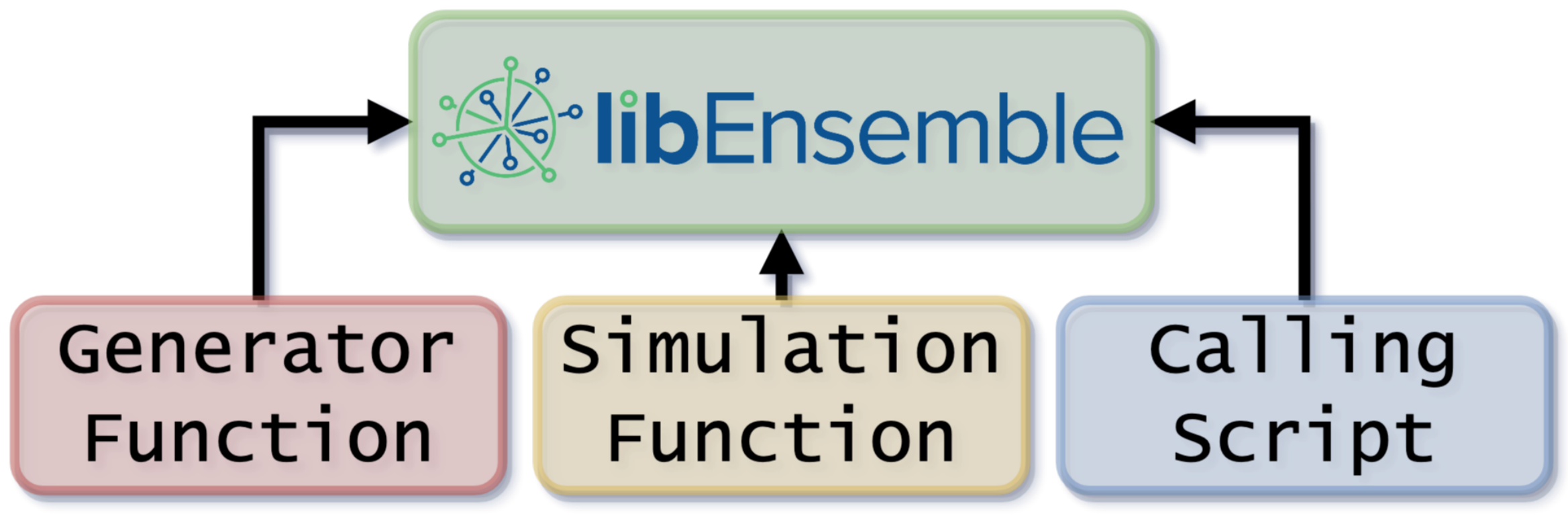
libEnsemble aims for the following:

- Extreme scaling
- Resilience/fault tolerance
- Monitoring/killing of tasks (and recovering resources)
- Portability and flexibility
- Exploitation of persistent data/control flow

libEnsemble can coordinate large numbers of parallel instances (ensembles) of simulations at huge scales.

Using libEnsemble

The user selects or supplies a generator function **gen_f** that generates simulation input and a simulation function **sim_f** that performs and monitors simulations. Users parameterize these functions and initiate libEnsemble in a **calling script**.



For example, the **gen_f** may contain an optimization routine to generate new simulation parameters on-the-fly based on results from previous **sim_f** simulations.

Potential use-cases include:

Generator Functions:

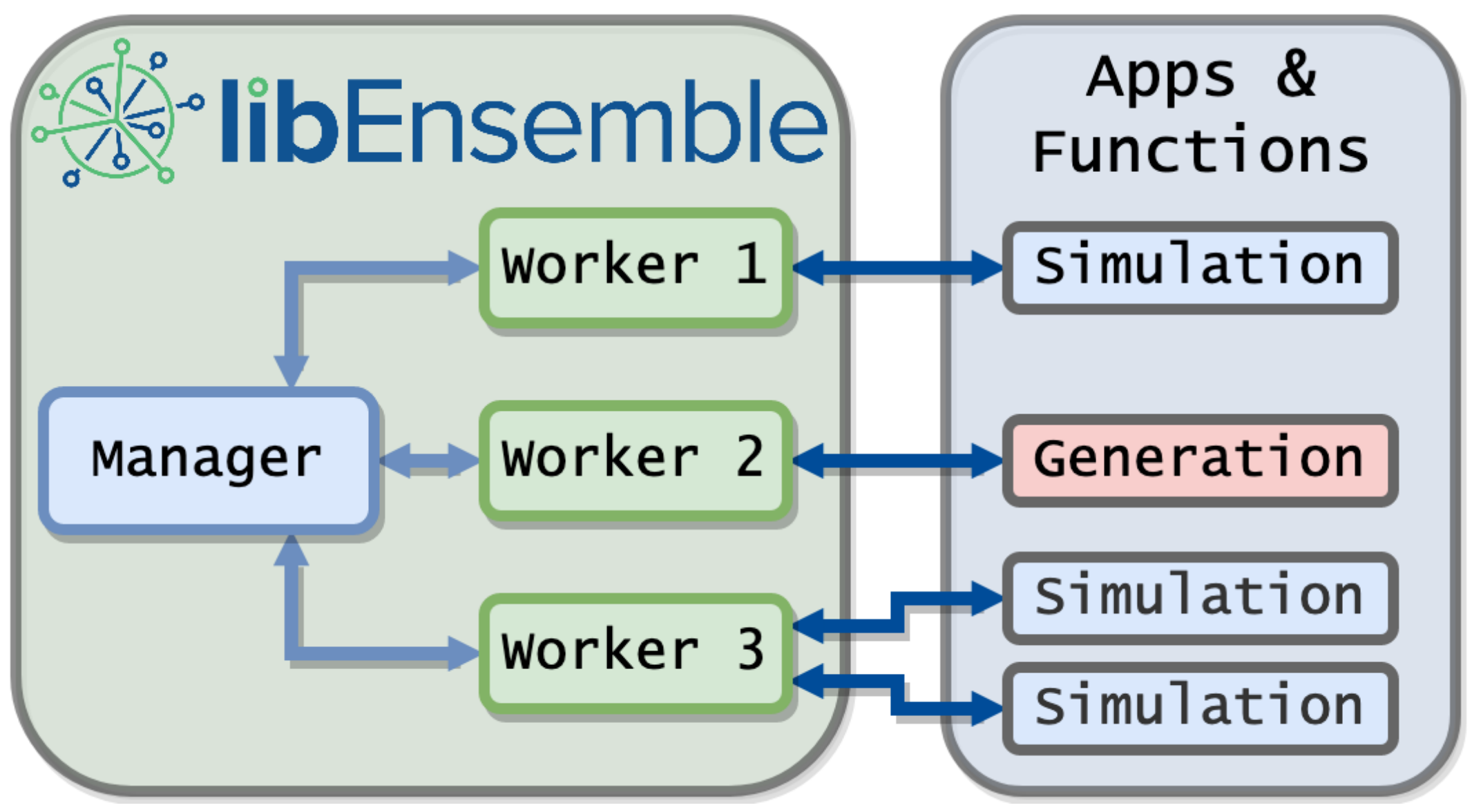
- Bayesian parameter estimation
- Surrogate models
- Sensitivity analysis
- Design optimization
- Supervised learning

Simulation Functions:

- Particle accelerator simulations
- Subsurface flow
- PETSc simulations
- DFT calculations
- Quantum chemistry

Manager and Workers

libEnsemble employs a **manager/worker** scheme that can communicate through **MPI**, Python's **multiprocessing**, or **TCP**. The manager allocates workers to asynchronously execute **gen_f** generator functions and **sim_f** simulation functions, directed by a provided **alloc_f** allocation function. Workers can control any level of work, from small sub-node tasks to huge many-node simulations.

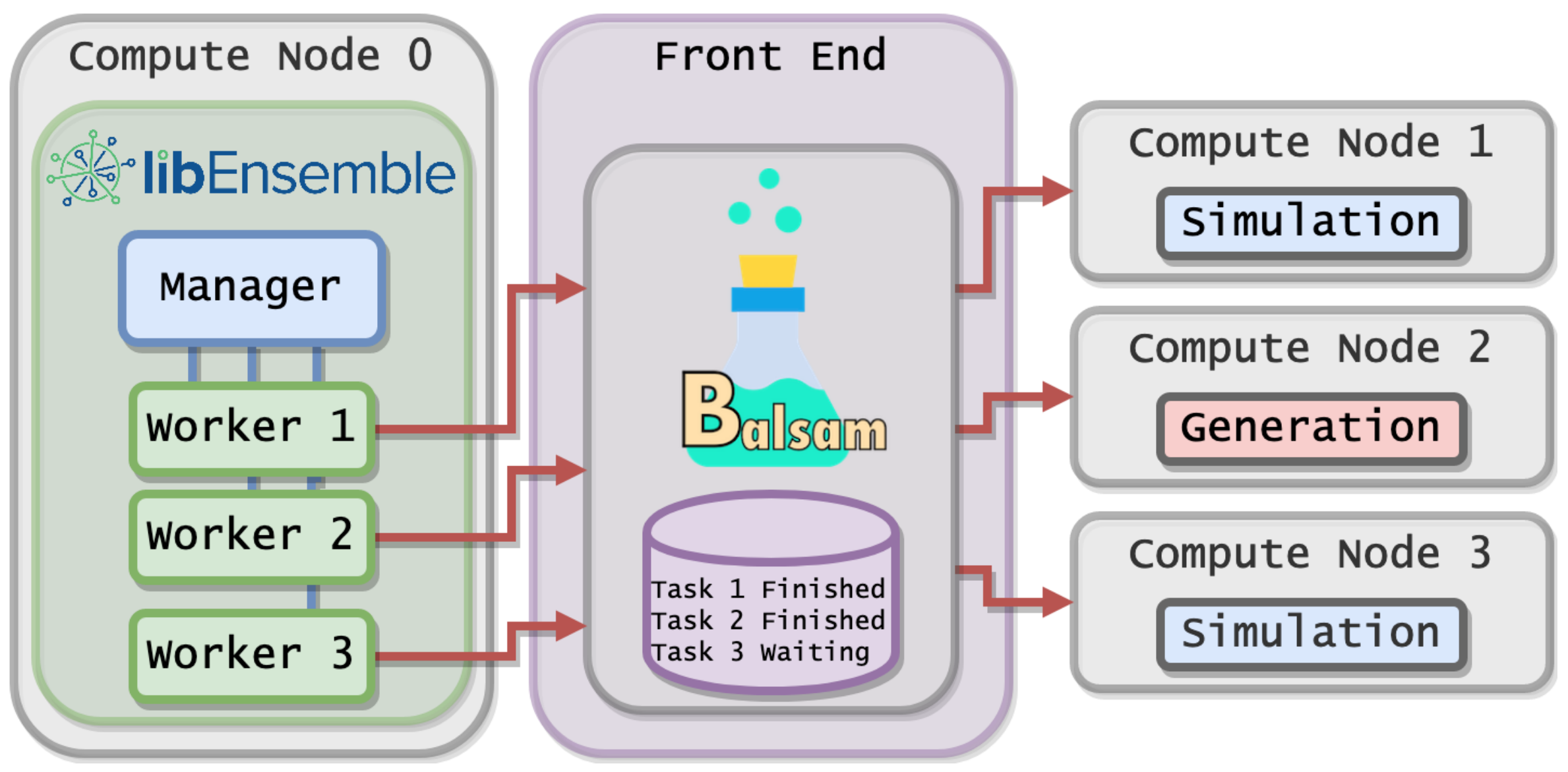


Executor Module

An **Executor** interface is provided so libEnsemble routines that coordinate **tasks** (user applications) are portable, resilient, and flexible. The Executor automatically detects allocated nodes and available cores and can split up tasks if resource data is not supplied.

The Executor is agnostic of both the job launch/management system and selected manager/worker communication method. The main functions are **submit()**, **poll()**, and **kill()**.

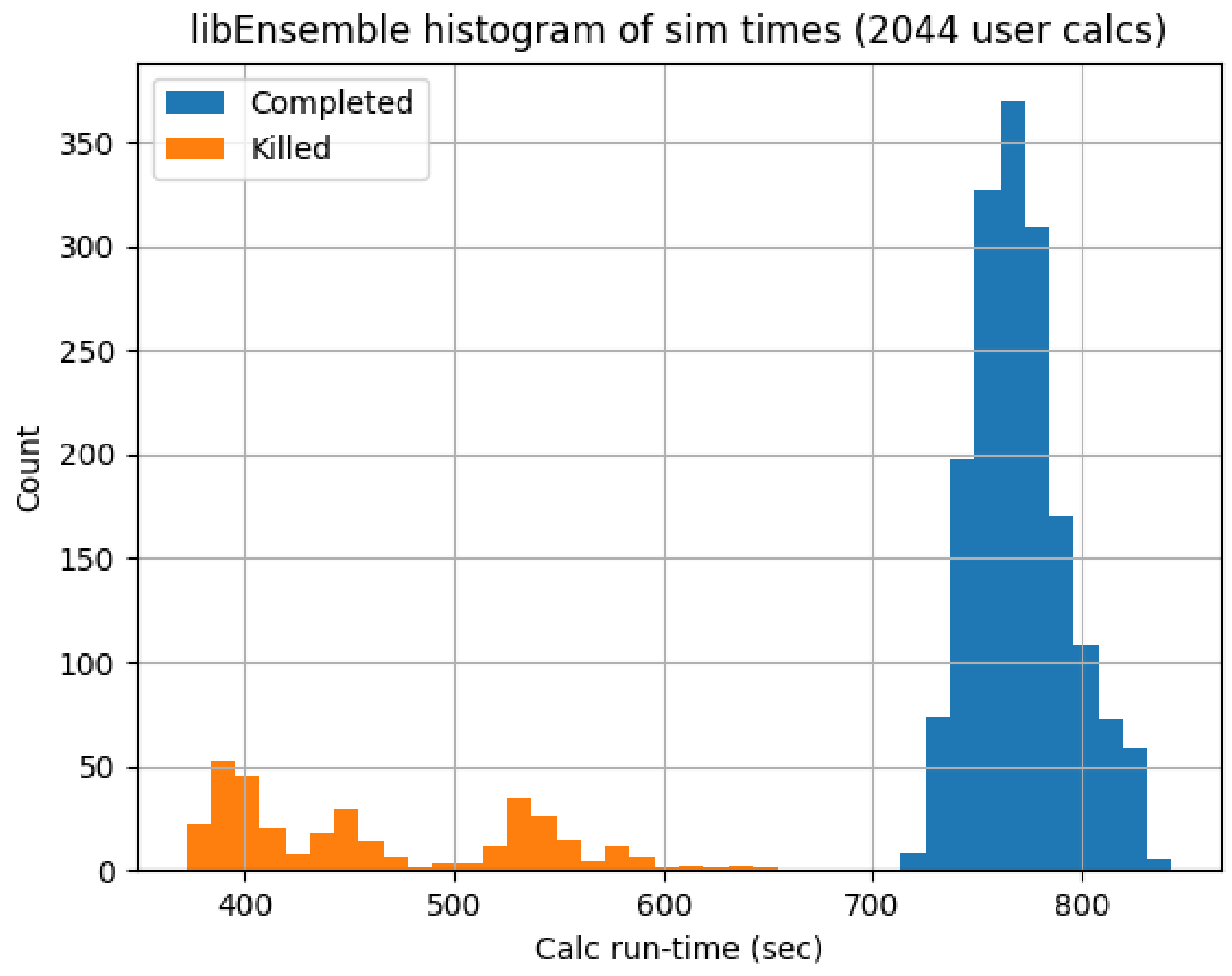
On machines that do not support launches from compute nodes, the Executor can interface with the **Balsam** library, which functions as a proxy job launcher that maintains and submits jobs from a database on front end launch nodes:



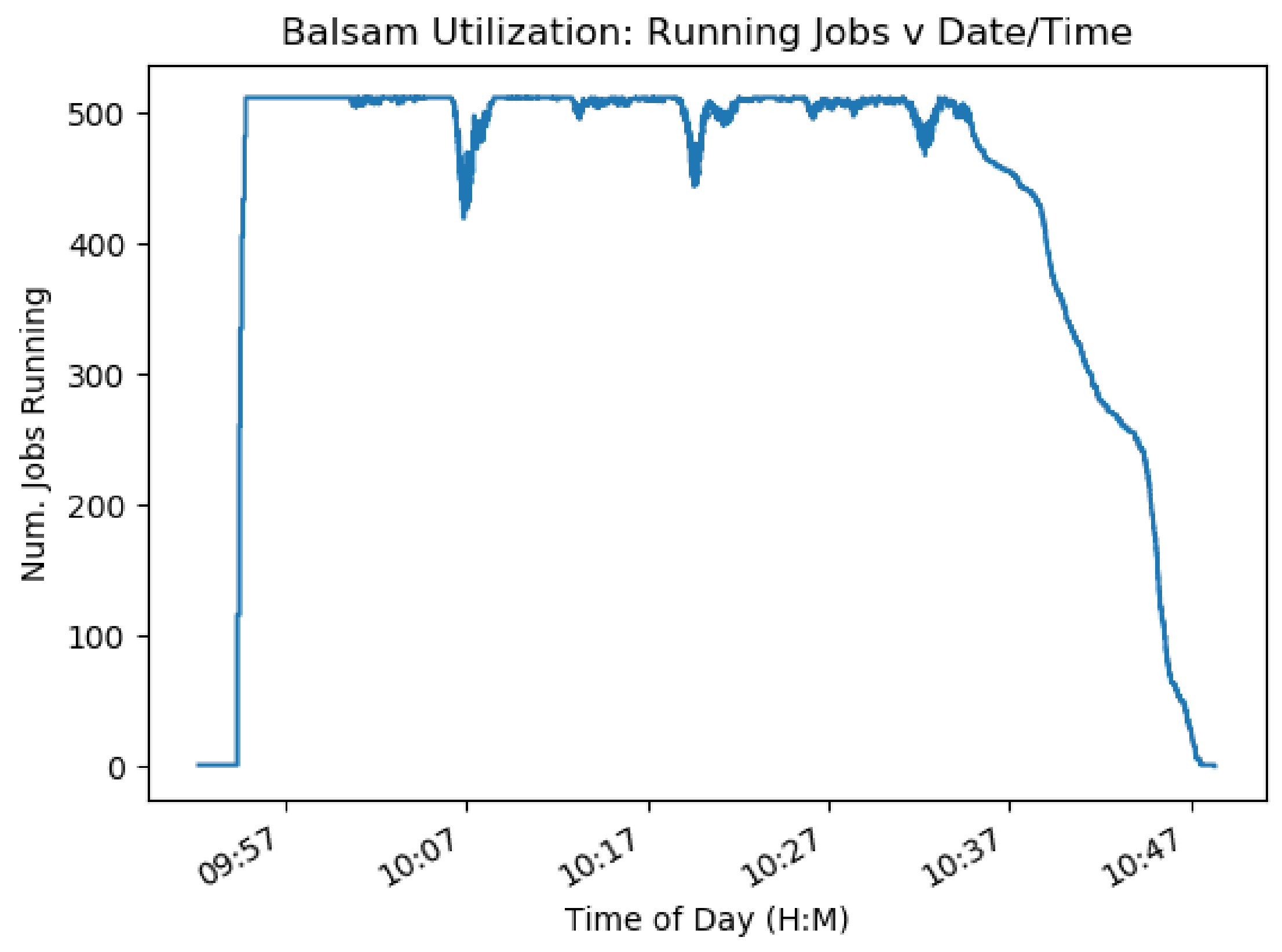
Running at Scale

OPAL Simulations

- ALCF/Theta (Cray XC40) with Balsam, at Argonne National Laboratory
- 1030 node allocation, 511 workers, MPI communications.
- 2044 2-node simulations
- Object Oriented Parallel Accelerator Library (OPAL) simulation functions.



Histogram of completed and killed simulations (binned by run time). Killing tasks once they are identified as redundant improves efficiency of ensembles.

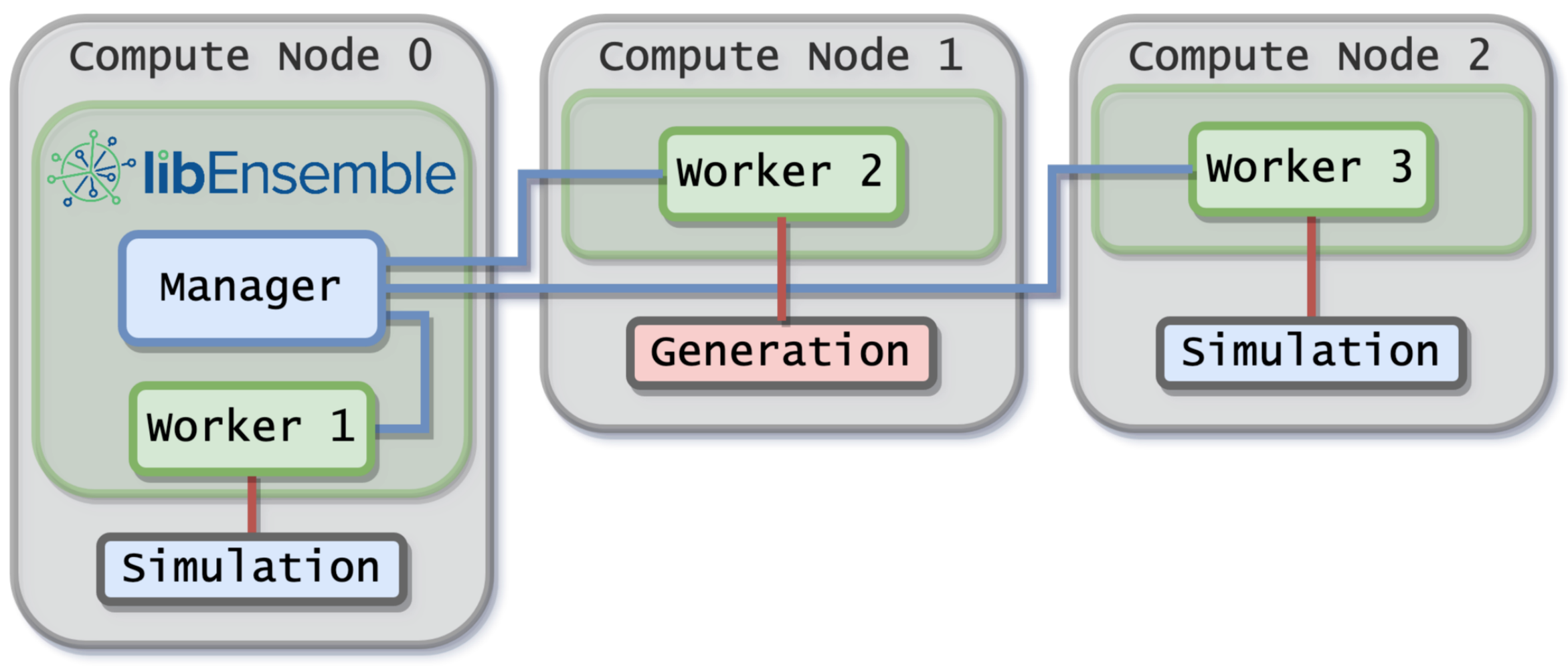


Total number of Balsam-launched applications running over time. The startup delay is due to parallel imports of Python libraries.

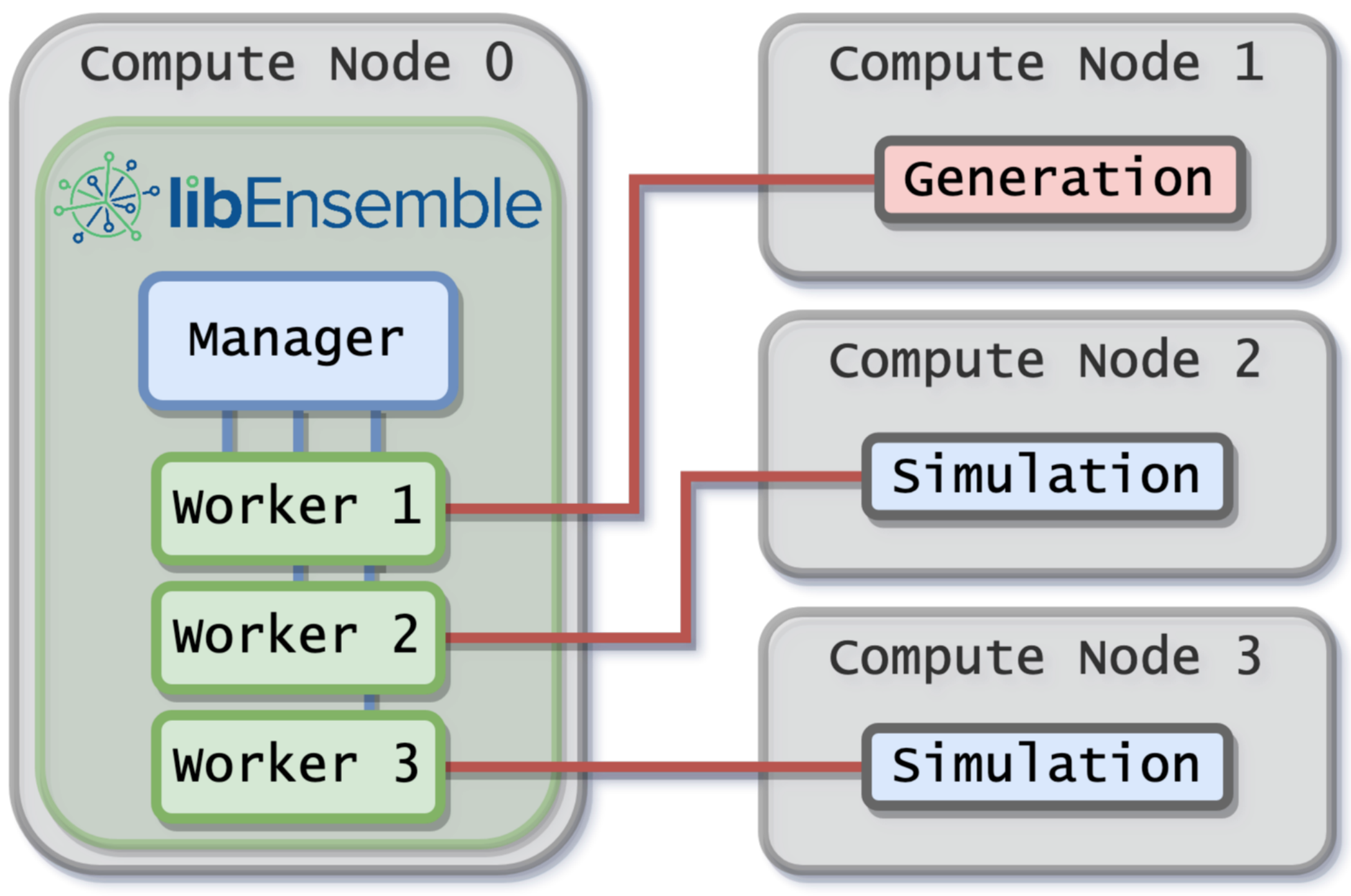
Flexible Run Mechanisms

libEnsemble is developed, supported, and tested on systems of highly varying scales, from laptops to machines with thousands of compute nodes. On multi-node systems, there are **two configuration modes** that determine how libEnsemble runs and launches tasks on available nodes.

Distributed: Workers are distributed across allocated nodes and launch tasks in-place. Worker processes share nodes with their applications.



Centralized: Workers run on one or more dedicated nodes and launch tasks to the remaining allocated nodes.



Dividing up workers and tasks to allocated nodes is highly configurable. Multiple workers and their functions can be assigned to a single node, or multiple nodes can be assigned to a single worker and it's routines.

Supported Research Machines

libEnsemble is tested and supported on the following high-performance research machines:

Machine	Location
Summit	Oak Ridge National Laboratory
Theta	Argonne National Laboratory
Cori	National Energy Research Scientific Computing Center
Bridges	Pittsburgh Supercomputing Center

Download or Contribute

libEnsemble is available through **pip**:

```
pip install libensemble
```

conda through the **conda-forge** channel:

```
conda install -c conda-forge libensemble
```

Spack:

```
spack install py-libensemble
```

libEnsemble is in-development on **GitHub**:
https://github.com/Libensemble/libensemble

Documentation and tutorials are available on **Read the Docs**:
https://libensemble.readthedocs.io